Das Fanout–Problem

Analysen und Heuristiken für flache und hierarchische Schaltungsbeschreibungen

Diplomarbeit
von
Michael Kreutzer

Fachbereich 14 – Informatik
Universität des Saarlandes
D–66041 Saarbrücken

Saarbrücken, im Oktober 1996

Inhaltsverzeichnis

Einleitung

Die Faustregel: „Alle zwei Jahre verdoppelt sich die Integrationsdichte und die Taktfrequenz“, beschreibt den momentan rapiden Fortschritt in der Entwicklung von Computern. Der Taktfrequenz kommt hierbei eine besondere Bedeutung zu: Sie entscheidet fast alleine darüber, ob ein Rechner Chancen bei der Markteinführung hat oder nicht. Werte von 133 MHz und 166 MHz sind inzwischen Standard. Prozessoren mit 200 MHz werden entwickelt. Die 64–bit Alpha–Prozessoren der Firma DEC (Digital Equipment Corporation) wurden bereits mit 266 MHz eingeführt, neuerdings gibt es sie mit 333 MHz. Damit ist aber noch nicht einmal das Ende der Fahnenstange erreicht.
Aufgrund der hohen Integrationsdichte heutiger Chips ist es mittlerweile unmöglich, sie „von Hand“ zu entwerfen. Das Hauptkennzeichen der VLSI–Entwurfsmethoden (VLSI = very large scale integration - Entwurf höchstintegrierter Schaltungen) ist also die Rechnerunterstützung oder Automatisierung. Der Ablauf eines Entwurfs läßt sich grob in folgende Schritte untergliedern: die Beschreibung der gewünschten Schaltfunktion auf abstrakter Ebene (Spezifikation), die Überprüfung der logischen Korrektheit des Entwurfs (Logiksimulation), die Analyse des Zeitverhaltens, das Plazieren, Verdrahten und schließlich die Generierung der Testmuster (um Fertigungsfehler feststellen zu können). Abschließend können nun die Fertigungsdaten erzeugt werden. Das Fanout–Problem tritt an zwei Stellen des Entwurfsprozesses auf. Zunächst nach der Spezifikation: Nachdem die Logik einer Schaltung entworfen wurde, kann es innerhalb der Schaltung Stellen geben, an denen das Ausgangssignal eines Bausteins auf viele Eingänge verteilt wird. Der Strom am Ausgang des Bausteins muß eine Kapazität abbauen. Diese Kapazität besteht aus den Eingangskapazitäten der getriebenen Eingänge und den Kapazitäten der Leitungen zu den Eingängen. Ist die Gesamtkapazität zu hoch und der Strom zu niedrig, dann wird die Laufzeit des Signals verzögert, schlimmstenfalls bricht die Spannung zusammen und die Korrektheit der Schaltung ist nicht mehr gewährleistet. Eine Fanout–Korrektur ist also nach der Spezifikation notwendig, um die Korrektheit zu gewährleisten. Nach der Zeitanalyse kann eine Fanout–Korrektur hilfreich sein: Wenn die Schaltgeschwindigkeit des Entwurfs langsamer als das gewünschte zeitliche Verhalten ist, dann sucht man die zeitkritischen Stellen innerhalb der Schaltung und versucht beispielsweise durch Einsetzen von Fanout–Bäumen (das sind Treiberbäume, die hohe Kapazitäten verhindern und die Teilpfade kritischer Pfade beschleunigen sollen), ihre Verzögerung zu verringern.
Man stelle sich einmal vor, aufgrund eines schlechten Entwurfs der Steuerlogik kann ein Prozessor statt mit den erwarteten 166 MHz nur mit 133 MHz betriebssicher funktio-

nieren. Der Grund dafür kann eine einzige Stelle sein, für die das Fanout–Problem nicht gelöst wurde, beispielsweise ein zu hoher Ausgangsgrad eines Bausteins auf dem zeitkritischsten Pfad durch die Schaltung.
Die Fanout–Korrekturen sind nicht trivial, und Lösungen von Algorithmen sind meist qualitativ schlechter als Lösungen eines Entwurfsexperten. Aufgrund der Komplexität heutiger Entwürfe gibt es jedoch keine Alternative zu algorithmischen Lösungen.

In der vorliegenden Arbeit werden zwei Annäherungen an das Fanout–Problem vorgestellt.
Zunächst wird das Fanout–Problem auf „flachen“ Schaltungen betrachtet. Als flache Schaltung bezeichnen wir eine nicht hierarchisch aufgebaute Schaltung, d.h. auf ihrem einzigen Schaltplan befinden sich nur Bausteine, die elementare Funktionen realisieren. Flache Schaltungen sind kleine, für spezielle Zwecke entworfene Schaltungen. Sie werden häufig von einem Silicon–Compiler (er kann - für kleine Schaltungen - aus der Spezifikation direkt einen kompletten Entwurf generieren) erzeugt oder von Hand erstellt. Für das Fanout–Problem auf flachen Schaltungen gibt es eine Vielzahl von Heuristiken. Eine davon, die Heuristik in [SSV90], wird im Rahmen dieser Arbeit vorgestellt, da sie viele Techniken früherer Arbeiten des diskreten Fanout–Problems in sich vereinigt, beispielsweise die Reduktion der Länge der kritischsten Pfade ([Mic87] und [KJSSV88]) und das Auswählen von verbesserungswürdigen Bausteinen mit Hilfe von speziellen Graphalgorithmen (siehe auch [Mic87]). Die Idee, einen rekursiven Algorithmus zur Erzeugung von Fanout–Bäumen zu entwickeln, wurde in [SSV90] von [PP89] (diese Arbeit behandelt die optimale Zerlegung von Bausteinen mit vielen Eingängen in mehrere Bausteine mit wenigen Eingängen) übernommen.
In [BCD89] werden dynamische Programmiertechniken für die Fanout–Algorithmen benutzt. Der Ansatz in [BCD89] geht von einem vereinfachten Verzögerungsmodell aus, was für viele Technologien eine zu starke Restriktion darstellt. Der erste dynamische Algorithmus, der in [BCD89] präsentiert wird, hat eine Laufzeit von $O(N^5)$ (wobei N die Anzahl der Bausteine ist), der zweite hat zwar eine bessere Laufzeit, er erzeugt jedoch nur balancierte Fanout–Bäume.
Es gibt auch Ansätze, die den kontinuierlichen Charakter des Problems berücksichtigen. In [RWG77] können beispielsweise die Schaltgeschwindigkeit, die Kosten und die Treibereigenschaften durch Änderung der Transistorweiten kontinuierlich verändert werden. Dieses Verfahren wird für große Schaltungen jedoch zu komplex.
Das Problem, den Ausgangsgrad einer flachen Schaltung auf einen bestimmten Wert zu beschränken, wird in [HKP84] mit lediglich konstanter Zunahme der Größe und Tiefe der Schaltung gelöst.

Die zweite Annäherung an das Fanout–Problem hat als Grundlage eine hierarchische Schaltungsbeschreibung. Eine hierarchische Schaltung ist aus mehreren Teilschaltungen aufgebaut. Jede Teilschaltung besteht im allgemeinen aus logischen Bausteinen, Verdrahtung, Anschlüssen am Rand und „Black Boxes“. Mit „Black Boxes“ oder Makros werden Platzhalter für Vorkommen von Teilschaltungen bezeichnet. Alle großen Chips werden heutzutage hierarchisch entworfen, da hiermit der komplexe Gesamtentwurf in überschaubare Einheiten zerlegt werden kann. Wenn die zu realisierende Schaltfunktion einen regelmäßigen Aufbau hat, dann kann sie kompakt (platzsparend) in eine hierarchische Darstellung übertragen werden. Ein Algorithmus auf solch einer kompakten Darstellung kann sehr effizient sein, wenn er auf jede Teilschaltung (die eventuell sehr oft in der gedruckten Schaltung vorkommt) nur einmal angewendet werden muß. In vielen Fällen ist dies jedoch nicht möglich, da die Teilschaltung - je nachdem wo sie später eingesetzt wird - in unterschiedlichem Kontext vorkommt und die vom Algorithmus zu lösende Problemstellung kontextabhängig ist. Die Qualität der Lösung eines Algorithmus auf hierarchisch spezifizierten Schaltungen kann schlechter als im flachen Fall sein, allein schon dadurch, daß im hierarchischen Fall die Schaltung in mehrere Teile zerlegt ist und es für Manipulationen deswegen weniger Freiheiten gibt. Im Rahmen dieser Arbeit wird unter anderem untersucht, ob das Fanout–Problem auf einer bestimmten hierarchischen Schaltungsbeschreibung gut gelöst werden kann. Es kann im hierarchischen Fall Signale geben, die auf sehr viele verschiedene Teilschaltungen verteilt werden. Treten dort Fanout–Probleme auf, so sind diese nicht ohne weiteres lokalisierbar, ferner sind an solchen Stellen die Möglichkeiten von Korrekturen eingeschränkt, und eventuell muß eine kompakte, gefaltete Darstellung (d.h. es gibt für jede Teilschaltung nur ein definierendes Vorkommen, die angewandten Vorkommen werden durch Verweise darauf modelliert) teilentfaltet werden. Wir stellen eine Idee für einen Fanout–Algorithmus auf einer hierarchischen Schaltungsbeschreibung vor (hierbei verwenden wir die Zeitanalyse für hierarchische Schaltungen in [Sch92], die wir kurz skizzieren). Für das Problem, den Ausgangsgrad aller Bausteine zu beschränken, fanden wir einen Algorithmus, der im Rahmen dieser Arbeit implementiert wurde. Uns sind aus der Literatur zum Fanout–Problem keine Lösungsvorschläge für hierarchische Schaltungsbeschreibungen bekannt.

Alle Algorithmen, die im Rahmen dieser Arbeit programmiert wurden, wurden in die graphische Umgebung des Systems CADIC integriert. CADIC ist ein Entwurfssystem zum automatisierten Entwurf von Schaltungen und wurde im Teilprojekt B1 des Sonderforschungsbereiches 124 entwickelt. Mit CADIC ist es möglich, Schaltungen parametrisiert zu beschreiben, beispielsweise spezifiziert der Entwerfer einen n–Bit Multiplizierer, und CADIC erzeugt für einen konkreten Parameterwert von n die entsprechende Schaltung.

Die Arbeit gliedert sich in fünf Kapitel.

In Kapitel 1 nähern wir uns dem Fanout–Problem aus elektrotechnischer Sicht. Zunächst begründen wir, warum ein Signal am Ausgang eines Bausteins nur mit einer nach oben beschränkten Anzahl von Eingängen verbunden werden darf. Diese Anzahl wird maximales Fanout genannt und hat für jede Technologie einen bestimmten Wert. Wir geben das maximale Fanout für die gängigen Technologien an.
Anschließend wenden wir uns einem Optimierungskriterium des Logikentwurfes zu: Die Signallaufzeit durch eine Schaltung. Hierzu betrachten wir einzelne Gatter und analysieren ihre Verzögerungszeiten. Vereinfacht kann man für jedes Gatter eine Grundverzögerung und eine lastabhängige Verzögerung annehmen, die proportional mit der kapazitiven Last am Ausgang wächst.

Schließlich leiten wir aus den Ergebnissen des ersten Kapitels Forderungen an den VLSI–Entwurf ab.

In Kapitel 2 wird das Fanout–Problem definiert und analysiert. Zunächst motivieren wir mit drei Beispielen, welche Bedeutung die Fanout–Problematik für die Entwicklung von Schaltungen hat. Die sich daran anschließende Definition des Fanout–Problems bezieht sich auf die Signallaufzeit: Für bestimmte Stellen einer Schaltung sollen Treiberbäume gesucht werden, die vorgegebene Zeitspezifikationen einhalten. Ein Treiberbaum vervielfacht ein Signal mit Hilfe von Pufferbausteinen. Für das Ursprungssignal gibt es eine garantierte Ankunftszeit, und für jedes Ziel ist der Zeitpunkt gegeben, zu dem das Signal spätestens anliegen muß. Algorithmen für dieses Problem heißen Fanout–Algorithmen. Davon unterschieden werden Algorithmen, die dafür sorgen, daß in einer Schaltung ein festgelegter Fanout–Wert nicht überschritten wird, sogenannte Max–Fanout–Algorithmen. Am Ende dieses Kapitels wird die Komplexität des Fanout–Problems motiviert und bewiesen: Das diskrete Fanout–Problem ist NP–vollständig.

In Kapitel 3 stellen wir den Fanout–Algorithmus aus [SSV90] vor, der auf flache Schaltungen angewendet wird. Zunächst sucht der Algorithmus eine Teilschaltung der gegebenen Schaltung, die die Zeitforderungen (am stärksten) verletzt. Innerhalb dieser Teilschaltung werden Bausteine mit hohem Ausgangsgrad ausgewählt. Sie sind gute Kandidaten für eine Fanout–Korrektur, denn durch ihren hohen Ausgangsgrad entsteht eine hohe Kapazität und diese verzögert das Signal. Nun wendet der Algorithmus Techniken an, die die Signallaufzeit beschleunigen sollen. Eine einfache ist das Ersetzen von Bausteinen durch logisch äquivalente Bausteine aus dem Bausteinkatalog, die jedoch passendere Treibereigenschaften haben. Der Algorithmus kann darüber hinaus Fanout–Bäume erzeugen. Sie bestehen aus Puffern und dienen der Vervielfachung eines Signals - die Bäume ersetzen im Prinzip Fanout–Knoten mit hoher Summenkapazität, die vor allem durch die angeschlossenen Eingänge entstehen. Zur Erstellung eines solchen Baumes wird ein birekursiver Algorithmus verwendet, der einerseits kritische und unkritische Signale verschieden puffern kann, der andererseits aber auch eine balancierte Strategie für ungefähr gleich kritische Signale zur Verfügung stellt. Am Ende dieses Kapitels wird dieser Algorithmus analysiert, und es werden seine Ergebnisse auf Beispielschaltungen vorgestellt.

In Kapitel 4 gehen wir der Frage nach, ob das Fanout–Problem gut auf hierarchisch spezifizierten Schaltungen lösbar ist. Hierzu definieren wir zunächst hierarchische Beschreibungen von Schaltungen und die Operationen, die wir zur Modifikation solcher Schaltungen zulassen. Bei hierarchischen Schaltungen können aus verschiedenen Gründen nicht ohne weiteres Treiber und Treiberbäume eingesetzt werden. Einer dieser Gründe lautet: eine einzelne Leitung in der hierarchischen Schaltung hat in der entsprechenden flachen Schaltung eventuell mehrere Vorkommen mit verschiedenen Signalflußrichtungen. Aus diesem Grund stellen wir zwei Verfahren vor, die die Signalflußrichtungen in der hierarchischen Beschreibung eindeutig machen. Dann betrachten wir die Realisierbarkeit und Güte von Fanout–Algorithmen auf hierarchisch spezifizierten Schaltungen und liefern eine Idee für einen solchen Algorithmus. Anschließend beschreiben wir ausführlich einen von uns gefundenen Max–Fanout–Algorithmus für hierarchisch spezifizierte Schaltungen. Schließlich betrachten wir am Ende des Kapitels die Realisierbarkeit von Lösungen des Fanout–Problems auf parametrisierten Schaltungen und gehen auf ein Beispiel näher ein.

Kapitel 5 stellt die im Rahmen dieser Arbeit erstellte Software vor und beschreibt ihre Einbindung in die grafische Oberfläche des CADIC–Systems. Neben dem Algorithmus in [SSV90] und dem hierarchischen Max-Fanout–Algorithmus wurde eine Routine für die Leitungsführung implementiert. Anhand von Beispielen zeigen wir, wie die Algorithmen flache und hierarchische Schaltungen verändern.

Danksagungen

Mein Dank gilt vor allem Herrn Prof. Dr. Günter Hotz für die interessante Aufgabenstellung und die vielfältige Unterstützung. Desweiteren danke ich Herrn Prof. Dr. Paul Molitor für die Aufgabenstellung meines Fortgeschrittenenpraktikums. Besonders danken möchte ich Herrn Dr. Thomas Burch für die Betreuung, die entscheidende Mitarbeit bei der Konzeption und Implementierung der Algorithmen und das sorgfältige Korrekturlesen dieser Arbeit. Desweiteren danke ich Thomas Chadzelek, Frank Follert, Joachim Kreutzer, Dieter Luxenburger, Holger Müller, Dr. Silvia Melitta Müller, Armin Reichert, Jörg Ritter, Frank Schulz, Markus Sinnwell und Uwe Sparmann für ihre Anregungen und ihre Unterstützung. Last but not least gilt mein Dank meinen Eltern, nicht nur für das sorgfältige Korrekturlesen dieser Arbeit, sondern auch dafür, daß sie ihren drei Kindern ein Studium ermöglich(t)en.

Kapitel 1

Elektrotechnische Betrachtung

1.1 Das Fanout aus elektrotechnischer Sicht

In der elektrotechnischen Literatur wie in der Informatik–Literatur wird das maximale Fanout für eine Zellenbibliothek im allgemeinen wie folgt definiert (vergleiche beispielsweise [Has72], [Weg87] und [HKP84]):

Definition 1.1 :
Das maximale Fanout einer Zellenbibliothek ist die Anzahl der logischen Eingänge von Basiszellen, mit der ein Ausgang jeder Basiszelle belastet werden darf.

Man bezeichnet das maximale Fanout auch als Ausgangsbelastbarkeit. Die entsprechende kapazitive Last wird häufig in Lasteinheiten, *load units* ausgedrückt. Der Wert des maximalen Fanouts einer Basiszelle läßt sich nach [Has72] wie folgt berechnen:

$$\text{max. Fanout} = \lfloor \min\{\frac{I_{load}}{I_{in}(1)}, \frac{I_{sink}}{I_{in}(0)}\} \rfloor \tag{1.1}$$

I_{in} ist der Eingangsstrombedarf, „0“ und „1“ stehen für die logischen Funktionen in positiver Logik. I_{load} gibt an, wieviel Strom der Ausgang zu ziehen vermag (im „0“–Potential), I_{sink} wieviel Strom er liefern kann (im „1“–Potential). Die Formel 1.1 wurde aus der Knotenregel gewonnen, da der Strom sich zwischen dem Ausgang und den Eingängen verzweigt. Sie bezieht sich auf die ungünstigsten Betriebsbedingungen. Ungünstige Betriebsbedingungen sind zum Beispiel eine niedrige Betriebsspannung und eine hohe Chip–Temperatur. Für die Zellenbibliothek nimmt man als maximalen Fanout–Wert das Minimum aller Fanout–Werte der Basiszellen.

Berücksichtigt man den im Datenblatt für eine Zellenbibliothek angegebenen maximalen Fanout–Wert nicht, dann hat dies folgende Konsequenz:

Wird in einer Schaltung der maximale Fanout–Wert am Ausgang eines Bausteins überschritten, so ist das elektrische Funktionieren der Schaltung nicht gewährleistet. Die logische Funktion der Schaltung kann von ihrer Spezifikation abweichen, d.h. die Schaltung liefert ein inkorrektes Ergebnis.

Warum die Schaltung elektrisch nicht mehr korrekt funktioniert, läßt sich mit dem Schaltverhalten an einer Kapazität nach [SS93] erklären: Durch den zu hohen Kapazitätswert wird der Ladevorgang beziehungsweise Entladevorgang (je nach Richtung des Flankenwechsels) der Kapazität stark verzögert. Somit wird die zum Identifizieren des Signals nötige Spannung überhaupt nicht oder zu spät erreicht (vgl. [Has92]). In den Datenblättern jeder Technologie sind Grenzwerte für die Spannungspegel zum Erkennen von Signalen und für die Geschwindigkeit, mit der sie erreicht werden müssen, angegeben: Es gibt den maximal am Eingang zulässigen Low–Pegel U_{ilmax} (ilmax steht für input low maximum) und den minimal am Eingang geforderten High–Pegel U_{ihmin}. Die Flankensteilheit ist die Geschwindigkeit, mit der das Eingangssignal den Bereich zwischen U_{ilmax} und U_{ihmin} und umgekehrt durchfahren muß. Ist sie zu langsam, so werden die angesteuerten Bausteine zu Eigenschwingungen angeregt.

1.2 Die verwendete Technologie

Das maximale Fanout ist ein Kenndatum für jede Familie integrierter Schaltkreise. Die folgende Tabelle nach [HL87] stellt einige wichtige Kenndaten und ihre Werte vor:

Schaltkreis-Familie	Fan-out	Verzögerungs-zeit t_{pd}/ns	Verlustleistung pro Verknüpfungsglied P_v/mW	Speise-spannung U_0/V
TTL	10	10	10	5
Schottky–TTL	10	1,5	22	5
Low–Power Schottky	10	4	1	5
ECL	20	1	40	-5
CMOS	50	200 ... 40	0,01+10 f/Mhz	5...15

Die hier wiedergegebenen Tabelleneinträge sind Richtwerte.

TTL bedeutet Transitor–Transistor–Logik. Schottky–TTL Schaltkreise zeichnen sich durch geringe Verzögerungszeiten aus, Low–Power Schottky durch geringe Verlustleistung. ECL (Emitter Coupled Logic) Schaltkreise haben besonders geringe Verzögerungszeiten. Hier erreicht man hohe Schaltgeschwindigkeiten, weil die Transistoren nicht wie bei der TTL–Technik in die Sättigung übersteuert werden. Im Bereich der höchstintegrierten Schaltkreise findet heute überwiegend die MOS (metal oxid semiconductor) und die CMOS (Complementary MOS–fieldeffect–transitors) Technologie Anwendung.

Auf die (mittlere) Verzögerungszeit t_{pd} (*propagation delay time*) wird im nächsten Abschnitt näher eingegangen.

1.3 Verzögerungszeiten bei Flankenwechsel

Um möglichst schnelle Schaltungen zu erhalten, analysiert man die Verzögerungszeiten. Zunächst betrachtet man einzelne Gatter. Man kann versuchen, den kontinuierlichen Charakter des Problems zu modellieren. In [KMO89] heißt es hierzu: „Will man das zeitliche Verhalten von mehreren Transistoren beschreiben, erhält man aus dem Modell ein Differentialgleichungssystem, dessen Lösung mit Methoden der numerischen Mathematik angenähert wird. Die Berechnungskomplexität wird durch dieses relativ genaue Modell jedoch so groß, daß nur kleine Transistorschaltungen simuliert werden können. Die bekannten Simulatoren auf Transistorebene können zur Zeit nur Größenordnungen von einhundert Transistoren in angemessener Zeit simulieren. Dafür erhält man aber auch Aussagen, die von den tatsächlichen Werten nur um ±5% abweichen und kommt mit allen Schaltungen zurecht, wohingegen man Streuungen von ±10% bis ±50% und Einschränkungen an die Art der Schaltung bei einfacheren Modellen hinnehmen muß.“ Ein solches, einfacheres Modell wurde von uns gewählt: Abbildung 1.1a aus [Bü90] zeigt die zur Charakterisierung des Zeitverhaltens verwendeten Schwellwerte. Die Anstiegszeit t_{rise} (*rise time*) des Signals am Ausgang eines Bausteins ist nach [Has92] definiert durch die Zeit, die vergeht, bis das Signal von 10% auf 90% der vollen Amplitude angestiegen ist. Ebenso wird die Abfallzeit t_{fall} (*fall time*) durch die Zeit bestimmt, in der das Signal von 90% auf 10% der Amplitude abfällt. Für die Zeitanalyse einer Schaltung ist jedoch die Verzögerung durch ein Gatter von Interesse. In Abbildung 1.1a zieht ein Eingangsflankenwechsel einen entsprechenden Ausgangsflankenwechsel nach sich. Um die Verzögerung beim Hintereinanderschalten von Gattern korrekt zu erfassen, wird Verzögerungszeit zwischen Ein- und Ausgang auf den 50% Pegel bezogen. Man definiert sie für den Ausgang eines Gatters als t_{pdLH} für den Übergang von Low zu High und als t_{pdHL} für den Übergang von High zu Low.

t_{pd} ist die mittlere Verzögerungszeit. Sie wird durch eine interne und mehrere externe Komponenten bestimmt. Die *interne* Verzögerung (*intrinsic delay*), auch Grundverzögerung genannt, kann für jedes Gatter unabhängig von der Beschaltung angegeben werden. Die *externen* Größen sind nach [SE90]:

- Die Gesamtkapazität am Ausgang: sie beinhaltet die Kapazität des Ausgangs, die Kapazität der Verbindungsleitungen (Metallisierung) und die Eingangskapazitäten der Folgegatter.

- Die Betriebsspannung: bei höheren Betriebsspannungen wird die Schwellspannung, ab der der Ausgang der Folgegatter dem Eingangssignalwechsel folgt, schneller überschritten. Die Gatterlaufzeit wird deshalb bei höheren Versorgungsspannungen kleiner.

- Die Temperatur: bei höheren Temperaturen sinkt die Beweglichkeit der Ladungsträger im Kanal der Transistoren. Zwar reduziert eine erhöhte Temperatur die Beträge der Schwellspannungen, doch dies kann die Reduktion der Schaltgeschwindigkeit durch die verringerte Beweglichkeit nicht kompensieren. Die Umladung der

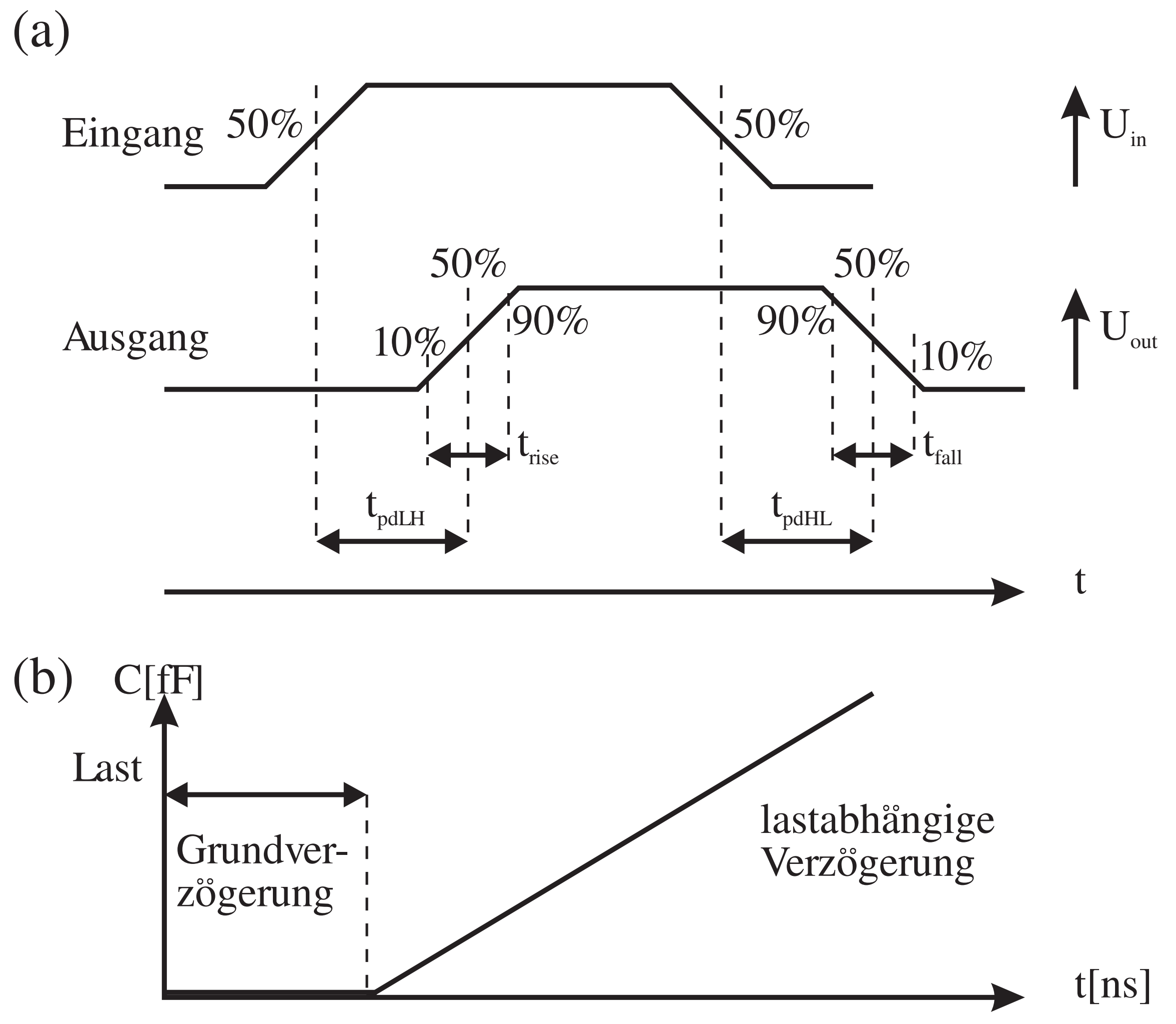

Abbildung 1.1: (a) Definition von Verzögerungs-, Anstiegs- und Abfallzeiten; (b) Lastkennlinie

Folgekapazitäten dauert deshalb bei höheren Temperaturen länger, d.h. die Gatterlaufzeit wird erhöht.

- Prozeßschwankungen: durch Toleranzen bei der Herstellung der Wafer können die Transistorkennwerte Streuungen unterliegen. Solche Prozeßschwankungen können deshalb die Gatterlaufzeit verkürzen, aber auch verlängern.

In [Has92] werden weitere *externe* Einflußmöglichkeiten auf die Verzögerungszeit angegeben:

- Schalten mehrerer Ausgänge einer Schaltung: Findet ein Flankenwechsel an mehreren Ausgängen gleichzeitig statt, so sind die in den Datenblättern angegebenen

Werte für die Verzögerungszeit nur bedingt anwendbar. Der Grund hierfür liegt darin, daß die Induktivitäten der Versorgungsspannungsleitungen der Schaltung einen nicht zu vernachlässigenden Einfluß auf sein Verhalten und damit auf die Verzögerungszeiten haben. Diese Induktivitäten bewirken, daß der Strom durch die Versorgungsleitungen und demnach auch durch den Ausgang der Schaltung nicht beliebig schnell ansteigen kann. Damit steht beim gleichzeitigen Schalten mehrerer Ausgänge nur ein verminderter Ausgangsstrom zur Verfügung. Bei Bustreibern und ebenso VLSI–Schaltungen, die mehr als acht gleichzeitig schaltende Ausgänge besitzen, werden auch dementsprechend viele Versorgungsanschlüsse vorgesehen.

- Negative Unterschwinger an Ausgängen: Bei bipolaren Schaltungen können negative Spannungen, die von Unterschwingern herrühren, die Funktion der Bauelemente beeinflussen. Ein Unterschwinger entsteht typischerweise durch einen mangelhaften Leitungsabschluß. Um diesen Effekt auszuschalten, wird in modernen Bustreibern eine Klammerschaltung integriert.

Die beiden letztgenannten Einflußmöglichkeiten können durch die erwähnten Abhilfemaßnahmen ausgeschlossen werden. Die einzige von den oben genannten sechs Größen, auf die der Logik–Entwerfer direkten Einfluß hat, ist die Gesamtkapazität am Ausgang. Nach [Has92] und [SS93] (und anderen Veröffentlichungen) gilt:

Satz 1.2:
Die Kapazität am Ausgang eines Bausteins beeinflußt die Verzögerungszeit näherungsweise linear.

Die Beeinflussung ist linear, wenn sich die kapazitive Last als konzentrierter Kondensator darstellen läßt, dies ist jedoch bei VLSI–Schaltungen nicht immer möglich. Vernachlässigt man die übrigen externen Einflußmöglichkeiten oder macht man worst case–Annahmen über sie, dann stellt Abbildung 1.1b aus [Bü90] die interne und externe Verzögerung entsprechend Satz 1.2 dar.

Beweis von Satz 1.2:
Nach [SS93] liefert die Maschenregel für den Spannungsverlauf beim Ladevorgang einer Kapazität C die Formel

$$u_C = U_0 \cdot (1 - e^{-\frac{t}{RC}})$$

Stellt man diese Formel nach t um und setzt die Werte für den Flankenwechsel von Low zu High ein, dann erhält man Formel 1.2 nach [Has92]:

$$t = \ln(\frac{U_{oh} - U_{ol}}{U_{oh} - U_{ihmin}}) \cdot R_o \cdot C_L = const \cdot C_L \tag{1.2}$$

U_{ol} ist die typische Ausgangsspannung der Schaltung im Low–Pegel, U_{oh} die typische Ausgangsspannung im High–Pegel. In [Has92] wird Formel 1.2 erläutert: „Der durch die kapazitive Last C_L bedingte Anstieg der Verzögerungszeit wird bestimmt durch die Zeit, die vergeht, bis die externe Lastkapazität durch die Spannungsquelle mit der Spannung U_{oh} und dem Innenwiderstand R_o von der Spannung U_{ol} bis zur Schwellspannung der Schaltung U_{ihmin} aufgeladen wird.“ Der Innenwiderstand kann nach [Has92] wie folgt berechnet werden: „Mit ausreichender Genauigkeit läßt sich der Innenwiderstand R_o einer Schaltung aus der Leerlaufausgangsspannung und dem im Datenblatt angegebenen Kurzschlußstrom I_{os} bestimmen: $R_o = \frac{U_{oh}}{I_{os}}$.“ ∎

Wie oben bereits erwähnt, besteht die Gesamtkapazität am Ausgang aus der Ausgangskapazität des Treibers, den Leitungskapazitäten und den Eingangskapazitäten der getriebenen Eingänge. Die Leitungslänge hat nicht nur Einfluß auf die Leitungskapazität, man muß in einigen Fällen auch die Ausbreitungsgeschwindigkeit der elektrischen Welle auf der Leitung (ca. zwanzig Zentimeter pro Nanosekunde) für die Verzögerungsberechnung mit in Betracht ziehen. Beträgt die Länge der Verbindungsleitungen nur wenige Zentimeter, wie dies auf gedruckten Schaltungen üblicherweise der Fall ist, dann kann vereinfacht angenommen werden, daß sich die Leitung wie eine Kapazität verhält. Im allgemeinen ist die Leitungslänge während des Logikentwurfes nicht bekannt. Man kann für die Leitungslängen (und damit auch für die Leitungskapazitäten) näherungsweise einen mittleren typischen Wert annehmen, der auf Messungen beruht. Entwickelt man einen VLSI–Chip, so kann man den Einfluß der Leitung auf die Verzögerungszeit im allgemeinen vernachlässigen.

1.4 Forderungen an den VLSI–Entwurf

Aus obiger Betrachtung ergeben sich zwei wesentliche Forderungen an den VLSI–Entwurf:

- Um korrekte Schaltungen zu erhalten, darf das erlaubte Fanout nicht überschritten werden.
- Um möglichst schnelle Schaltungen zu erhalten, sollte man die Verzögerungszeiten eines Entwurfs analysieren und den Entwurf nötigenfalls modifizieren. Als Grundlage für die Modifikation und die Analyse können Satz 1.2 und Formel 1.2 verwendet werden. Die Modifikation betrifft nicht die logische Schaltfunktion.

Kapitel 2

Das Fanout–Problem

2.1 Motivation

Die folgenden Beispiele geben einen Eindruck über die Bedeutung des Fanout–Problems für den Schaltungsentwurf:

- Zur Zeit machen Firmen große Anstrengungen, die Geschwindigkeit von Prozessoren und anderen VLSI-Produkten, z.B. Festplatten, zu steigern. Nun steht man bei diesen Entwürfen vor folgendem Problem: Die größtmögliche Taktfrequenz richtet sich in den meisten Fällen danach, wie schnell die langsamste Baugruppe getaktet werden kann. Ein Fanout–Algorithmus ist eine wichtige Komponente zur Lösung dieses Problems: Er wird darauf angesetzt, die zeitkritischen Pfade innerhalb der langsamsten Baugruppen in unkritische Pfade umzuwandeln.

- Für den im vierten Kapitel vorgestellten, mit CADIC entworfenen 2^n–Bit Addierer CSA[n] (CSA=conditional sum adder) kann man den größten Verzweigungsgrad in n angeben: Die größte Anzahl von Zielen eines Signals ist $2^{n-1} + 1$. Automatisch generierte, unmodifizierte Schaltungen sind für $n \geq 7$ inkorrekt, da sie das maximal zulässige Fanout für CMOS–Schaltungen überschreiten.

- In [MP95] sollen in einer Aufgabe zwei parallele Addierer miteinander verglichen werden. Dabei werden die Ursprungsentwürfe und Fanout–beschränkte Entwürfe (Fanout ≤ 2 und Fanout ≤ 4) betrachtet. Man macht folgende Beobachtung: Eine Beschränkung des Ausgangsgrades verbessert in manchen Fällen die Qualität (die Qualität ist ein in [MP95] definiertes Maß zur Beurteilung der Güte von Schaltungen) einer Schaltung.

2.2 Definition des Fanout–Problems

Im folgenden werden ausschließlich kombinatorische Schaltungen ohne asynchrone Rückkopplungen auf der Ebene des Logikentwurfs betrachtet. Eine Erweiterung auf sequentielle oder noch allgemeinere Schaltungen ist jedoch prinzipiell möglich:

- Die Signallaufzeitanalyse macht nur auf kombinatorischen Teilschaltungen Sinn, deswegen zerlegt man eine sequentielle Schaltung in kombinatorische Teilschaltungen entsprechend der Prüfbus (*Scan Path*) Technik.
- Das Problem, das maximale Fanout einzuhalten, kann auf sequentielle Schaltungen übertragen werden, beispielsweise indem man Speicherzellen in diesem Zusammenhang wie Gatter behandelt.

Eine Schaltung bestehe aus Anschlüssen, aus Bausteinen einer nicht näher spezifizierten Technologie und aus Verdrahtung. Die Bausteine realisieren logische Grundfunktionen. Die Kosten einer Schaltung sollen ein Maß für den Materialaufwand sein. Nach [KMO89] setzt sich der Materialaufwand für integrierte Schaltkreise nicht nur aus der Anzahl der Gatter, sondern auch aus der durch das Layout belegten Fläche zusammen, während in [BCD89] im Zusammenhang mit dem Fanout–Problem nur die Fläche als Kostenmaß angegeben wird. In [MP95] werden die Kosten wieder anders definiert, deswegen geben wir an dieser Stelle keine Definition an, sondern fordern zwei Eigenschaften für das Kostenmaß:

- Das Kostenmaß ist diskret: In jeder Technologie wird jedem Baustein ein fester Kostenwert zugeordnet.
- Das Kostenmaß ist additiv: Die Kosten einer kombinatorischen Schaltung oder Teilschaltung auf Logikebene ist die Summe der Kosten ihrer Bausteine.

Die Verzögerungszeiten der Bausteine der Technologie sollen entsprechend Satz 1.2 linear von der äußeren Beschaltung abhängen. Mit diesen Voraussetzungen wird das Fanout–Problem im allgemeinen wie folgt definiert (vergleiche [BCD89]):

Definition 2.1:
Finde eine Schaltung, die ein Signal mit Ankunftszeit t_a an N verschiedene Ziele (Senken) verteilt, so daß die Zeitforderungen an den Senken eingehalten werden. Die Kosten der Schaltung sollten minimal sein. Die verwendeten Bausteine müssen aus dem Bausteinkatalog der Technologie stammen.

Es wird angenommen, daß das Ursprungssignal zu einer gegebenen, garantierten Zeit t_a, der Ankunftszeit, auch *arrival time* genannt, korrekt anliegt. Die Zeitforderungen an den Senken sind konkret spezifizierte Zeiten $t_1, \ldots, t_N$. Sie werden *required times* genannt und geben an, zu welchem Zeitpunkt das Signal spätestens an der Senke korrekt anliegen muß.

Eine Instanz des Fanout–Problems kann unlösbar sein.

Nach [BCD89] gibt es ungefähr 100–200 Stellen, an denen das Fanout–Problem im Entwurf eines Chips auftritt; dabei liegt N in etwa im Bereich zwischen 30 und 50.

Das in 2.1 definierte Fanout–Problem ist NP–vollständig. Dies wird nach [BCD89] im übernächsten Abschnitt gezeigt.

Im ersten Kapitel stellten wir bereits fest, daß die diskrete Modellierung der Verzögerungszeiten ungenauer, aber auch wesentlich einfacher als die kontinuierliche ist. Wichtig bei der diskreten Modellierung ist, daß für Verzögerungszeiten „sichere“ (*worst case*) Annahmen gemacht werden. Sie garantieren, daß ein spezifizierter Schaltkreis auf jeden Fall, d.h. auch unter schlechtesten Betriebsbedingungen, die berechneten Verzögerungen einhält.

2.3 Klassifikation von Algorithmen

In der vorliegenden Arbeit werden zwei Typen von Algorithmen vorgestellt:

1. **Fanout–Algorithmen mit Zeitforderungen.**
   Gegeben ist eine Schaltung mit folgenden spezifizierten Zeiten: Arrival times an den primären Eingängen und required times an den primären Ausgängen. Ein Fanout–Algorithmus soll eine Ersatzschaltung finden, die die gleiche logische Funktion berechnet und die die Zeitforderungen an den primären Ausgängen erfüllt.
   In der Fachliteratur werden nur diese Algorithmen als „Fanout–Algorithmen“ bezeichnet. Man kann sie unter den Stichworten „Fanout Problem“, „Fanout Algorithm“, „Timing Optimization“, „Delay Optimization“, „Speeding Up Logic“ und „Logic Synthesis“ finden.
   Solche Algorithmen haben üblicherweise wenigstens die beiden folgenden Komponenten:

   - Zeitanalyse: Die Verzögerung der Signale durch die Schaltung wird analysiert.
   - Fanout–Problem: Mittels der Ergebnisse der Zeitanalyse kann man die Stellen finden, für die das Fanout–Problem gelöst werden soll. Zur Lösung dieser Probleme verwendet man meistens Bäume von Puffern, sogenannte Fanout–Bäume (*fanout trees*).

2. **Max–Fanout–Algorithmen.**
   Ein Max–Fanout–Algorithmus findet zu einer gegebenen Schaltung eine Ersatzschaltung, die die gleiche logische Funktion berechnet und die keine Stellen mehr hat, für

die das maximal zulässige Fanout überschritten ist. Ein Max–Fanout–Algorithmus garantiert also die Einhaltung des im ersten Kapitel eingeforderten maximalen Fanouts. In der Fachliteratur findet man solche Algorithmen unter dem Stichwort „Bounding Fanout“.
Ein Max–Fanout–Algorithmus ist im allgemeinen nicht dazu geeignet, das in Definition 2.1 angegebene Fanout–Problem zu lösen. In welchen Fällen dies dennoch möglich ist und wie die Resultate in diesen Fällen aussehen, wird im vierten Kapitel im Abschnitt über den hierarchischen Max–Fanout–Algorithmus näher erläutert.
In [HKP84] werden Max–Fanout–Algorithmen für flache Schaltungen vorgestellt, die die Schaltungsgröße und die Schaltungstiefe lediglich um einen konstanten Faktor vergrößern.

2.4 Die Komplexität des Fanout–Problems

2.4.1 Veranschaulichung an einem Beispiel

Um ein Fanout–Problem zu lösen, müssen zwei Größen gleichzeitig optimiert werden: die Verzögerungszeit und die Kosten. Das folgende Beispiel aus [BCD89] demonstriert dies.
Hierzu wird eine Technologie T konstruiert, die das Verhalten realer Technologien idealtypisch wiedergibt. Eine Besonderheit von T ist: jeder Ausgang eines Bausteins kann genau einen Eingang treiben (unabhängig von der Technologie nennt man solche Entwürfe „Fanout–freie“ Schaltungen). Um überhaupt ein Signal an mehrere Ziele leiten zu können, kann man Pufferbausteine mit mehreren Ausgängen verwenden oder Logik vervielfachen, inklusive der Vervielfachung von primären Eingängen. In T sei Logikvervielfachung nicht erlaubt. Pufferbausteine haben einen Eingang und sind in realen Technologien dadurch ausgezeichnet, daß sie Ausgänge mit guten Treibereigenschaften haben, daß also ein Ausgang viele Eingänge treiben kann. Das Eingangssignal wird von den Puffern zu ihren Ausgängen weitergeleitet, d.h. die logische Funktion eines Puffers ist die Identität. Ein Puffer mit o Ausgängen habe in T eine Verzögerungszeit von $o + 1$ ns. Die Wahl der Verzögerungszeit spiegelt den linearen Zusammenhang zwischen Anzahl der Senken und der Verzögerung wieder. Abbildung 2.1 zeigt drei Puffer aus T. Jeder Baustein in T koste 1.
Innerhalb einer Schaltung soll ein Signal am Ausgang eines Bausteins zu vier Eingängen geleitet werden. Um das Signal zu vervierfachen, muß eine Teilschaltung gefunden werden, die man als *Fanout–Schaltung* oder, wie im letzten Abschnitt bereits erwähnt, *Fanout–Baum* (der Ausgang ist die Wurzel, die Puffer sind die inneren Knoten, die Senken sind die Blätter und die Verbindungsleitungen werden zu Kanten) bezeichnet. Die Fanout–Schaltung kann Bedingungen bezüglich der Verzögerungszeit und der Kosten unterworfen sein. Abbildung 2.2 zeigt fünf mögliche Schaltungen, die obiges Fanout–Problem (es habe zunächst keine Randbedingung) lösen. An den jeweils vier Ausgängen stehen die Verzögerungszeiten (in *ns*) relativ zum Ursprungssignal. Die Reihenfolge der Zeiten soll hierbei

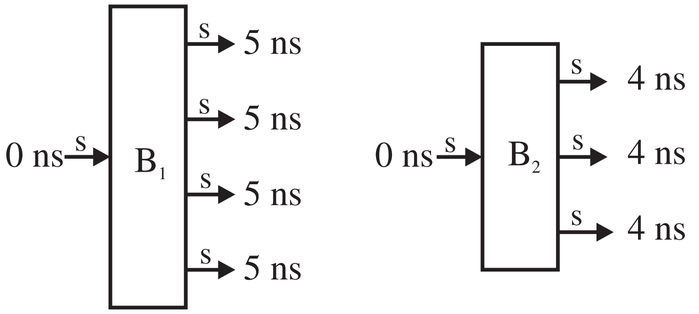

Abbildung 2.1: Drei Puffer zur Vervielfachung eines Signals s in der Technologie T

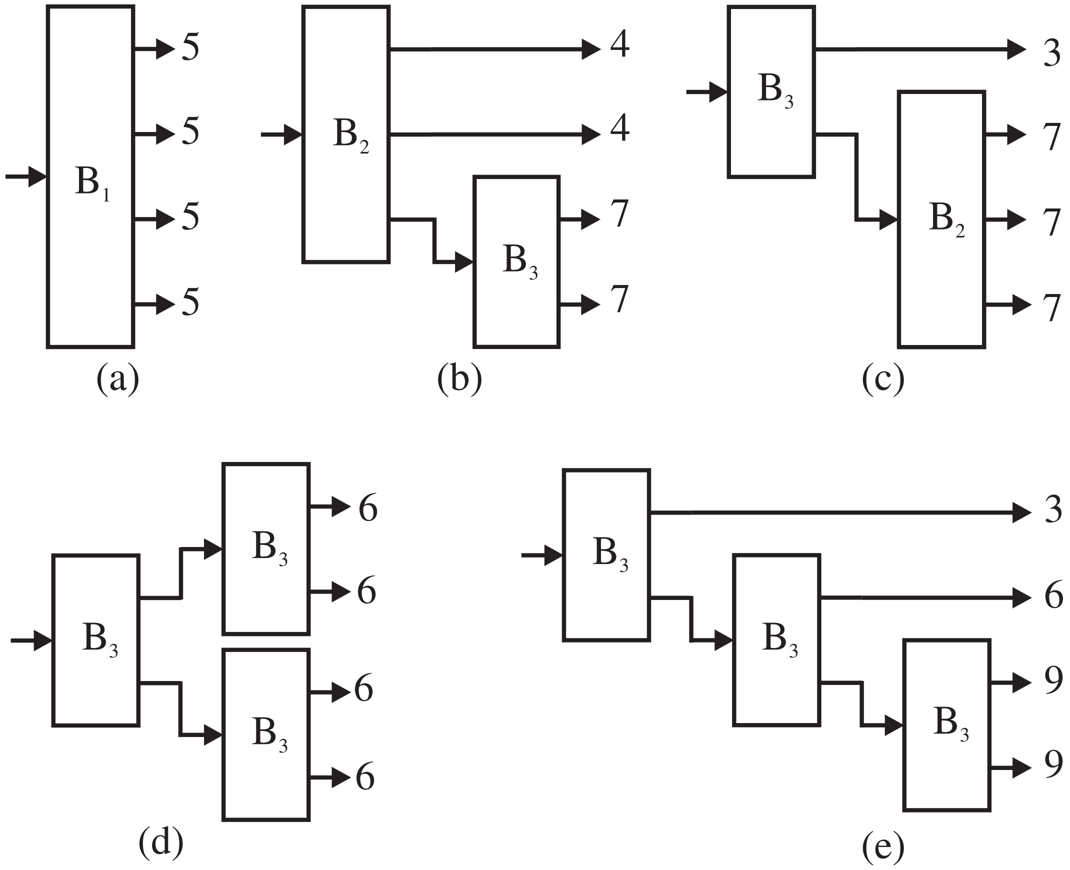

Abbildung 2.2: Fünf Fanout–Schaltungen

keine Rolle spielen, da alle Ausgänge das gleiche Signal führen und durch die nachfolgende Verdrahtung permutiert werden können. Welche Fanout–Schaltung ist die beste?

Mit Sicherheit gilt: (d) ist schlechter als (a), sowohl bezüglich der Verzögerungszeit als auch bezüglich der Kosten. (a) hat im Vergleich zu allen anderen Schaltungen minimale Kosten.
Wenn man Randbedingungen für die Signallaufzeit in die Betrachtung einbezieht, dann lassen sich weitere Aussagen machen.
Falls sämtliche required times an den Senken viel größer als die arrival time an der Quelle sind, dann erfüllen die 5 Fanout–Schaltungen die Zeitforderungen. In diesem Fall ist (a) die beste Fanout–Schaltung, da (a) minimale Kosten hat.
Angenommen, einer der Ausgänge soll nach 10ns, die anderen drei nach 14ns das korrekte Signal führen. Wann wäre dann der späteste Zeitpunkt für die arrival time? Für (a) ist es 5ns, (b) 6ns, (c) 7ns und für (e) 5ns (beachtenswert: bei (e) bestimmt einer der am wenigsten kritischen Ausgänge die arrival time). In diesem Fall ist (c) besser als (b) und (e), denn (c) hat eine höhere arrival time und kostet nicht mehr als (b) und (e). Falls die arrival time 7ns ist, dann löst nur (c) dieses Fanout–Problem.

2.4.2 Der Beweis der NP–Vollständigkeit

Nach dem vorangegangenen Beispiel kann man die Komplexität des Fanout–Problems bereits vermuten.
In [BCD89] wird bewiesen, daß eine einfache Instanz des in Definition 2.1 vorgestellten diskreten Fanout–Problems NP–vollständig ist. Dieses theoretische Resultat hat Einfluß auf die Entwicklung von Fanout–Algorithmen. Der Beweis in [BCD89] enthält für die meisten Lemmata lediglich die Beweisideen. Wir vollziehen ihn nach, geben jedoch für alle aufgestellten Lemmata Beweise an. Die von uns bewiesenen Lemmata sind als solche gekennzeichnet.

Vereinbarungen und Definitionen:
Für den Beweis wird die Technologie etwas abweichend von realen Technologien definiert: Eine *Technologie* ist eine Menge von Bausteintypen. Jeder Bausteintyp habe genau einen Ausgang und mindestens einen Eingang. Jedem Typ wird ein Tripel zugeordnet: $(d, c, f) \in \mathbb{N}^3$. d steht für die Verzögerung (*<u>d</u>elay*) durch den Bausteintyp, c gibt die Kosten (*<u>c</u>ost*) an. f ist die Anzahl der Eingänge, die sein Ausgang höchstens treiben kann.

Es sei eine elektrische Schaltung, bestehend aus Bausteinen der Bausteintypen der Technologie, Anschlüssen am Rand und Verdrahtung gegeben. Für die Zeitanalyse dieser Schaltung kann man von der Geometrie der Verdrahtung abstrahieren und ein einfacheres Beschreibungsmodell wählen. Hierzu wird die elektrische Schaltung in einen gerichteten Graphen $G = (V \cup \{IN, OUT\}, E)$ mit folgenden Eigenschaften umgewandelt:

1. Es gibt eine Bijektion zwischen den Bausteinen der elektrischen Schaltung und V.
2. Alle Knoten aus V werden mit ihren Bausteintypen beschriftet.

3. Die Kantenmenge wird wie folgt konstruiert: v_j und v_k seien Knoten aus V. i sei ein primärer Eingang, o sei ein primärer Ausgang der elektrischen Schaltung.
   (a) Ist ein Ausgang des Bausteins v_j mit einem Eingang eines Bausteins v_k verbunden, so existiert eine Kante in E von v_j zu v_k.
   (b) Ist i mit einem Eingang von v_j verbunden, so existiert eine Kante von IN zu v_j.
   (c) Ist ein Ausgang von v_j mit o verbunden, dann existiert eine Kante von v_j zu OUT.
   (d) Dies sind alle Kanten.
4. Die eingehenden Kanten der Knoten seien entsprechend der Reihenfolge der Eingänge auf dem Baustein geordnet, so daß man von der „x–ten Eingangskante“ sprechen kann. Die ausgehenden Kanten der Knoten sind analog geordnet.

Wie anfangs erwähnt, sind Rückkopplungen nicht erlaubt, weiter wird gefordert, daß die elektrische Schaltung keine Projektionen (Leitungen, die ohne Anschluß beginnen oder enden) und keine unbenutzten Ein- und Ausgänge hat. Daraus folgt für G:

- G ist ein gerichteter, azyklischer Graph, kurz dag (<u>d</u>irected <u>a</u>cyclic <u>g</u>raph).
- Der einzige Knoten, der keine eingehenden Kanten hat, ist der IN–Knoten. Seine ausgehenden Kanten werden primäre Eingänge genannt.
- Der einzige Knoten, der keine ausgehenden Kanten hat, ist der OUT–Knoten. Seine eingehenden Kanten werden primäre Ausgänge genannt.

G heißt *zeitbeschriftete kombinatorische Schaltung*, wenn jede Kante mit einer ganzen Zahl, der required time, beschriftet ist.

Die required times der primären Ausgänge sind vorgegeben. Daraus lassen sich die übrigen required times mit Hilfe der nachfolgenden Prozedur berechnen. Der zugrundeliegende Algorithmus ist nicht effizient, er soll lediglich die Vorgehensweise demonstrieren. Der Prozedur wird der Graph G übergeben. Zur besseren Veranschaulichung ordnen wir den Knoten Farben zu. Ein Knoten ist rot, wenn alle required times seiner ausgehenden Kanten bekannt sind, sonst ist er schwarz. Die Farbe eines Knotens kann sich von schwarz zu rot ändern. $OE(k)$ sei die Menge der ausgehenden Kanten des Knotens k, $IE(k)$ sei die Menge der eingehenden Kanten von k.

proc *Calculate required times*(G)
begin
  $M := V$;
  while $M \neq \emptyset$ **do**
    Update colours(M);
    $v :=$ *delete a red node*(M);
    $t := \min\{\textit{required times}(OE(v))\} - \textit{delay}(v)$;
    $\forall e \in IE(v):$ *required time*$(e) := t$;
  od
end

Jeder Knoten wird mit der required time seiner eingehenden Kanten beschriftet. Die required time einer Schaltung ist die früheste (minimale) required time ihrer primären Eingänge.

Eine *zeitbeschriftete kombinatorische Schaltung* kann eine negative required time haben. Ist dies der Fall, so erfüllt die Schaltung nicht die spezifizierten Zeitvorgaben.

Definition 2.2:
Ein einfaches Fanout–Problem (EFP) *ist eine Menge R von required times zusammen mit einer Technologie T.*

Intuitiv gesehen wird durch ein EFP folgende Aufgabe gestellt: Finde eine Schaltung, die ein Signal ausreichend oft vervielfacht und dabei die gegebenen required times einhält. Eine partielle Lösung ist dann eine Schaltung, die mehrere primäre Eingänge haben kann (anders formuliert: die das Problem „nicht vollständig löst", da für diese Lösung das Signal im allgemeinen mehrfach vorliegen muß), während eine Lösung des EFP aus einer Schaltung mit einem primären Eingang besteht. Formal:

Definition 2.3:
Eine partielle Lösung eines EFP *ist eine zeitbeschriftete kombinatorische Schaltung G, die die folgenden Restriktionen einhält:*

1. *Jeder innere Knoten in G hat den Eingangsgrad 1 und einen Ausgangsgrad ≥ 1.*
2. *Für jedes Element aus R gibt es genau einen primären Ausgang, der mit dem Wert des Elementes beschriftet ist.*

Hieraus folgt, daß eine partielle Lösung eines EFP aus einer Menge von Bäumen zwischen IN und OUT besteht.

Definition 2.4:
Eine partielle Lösung eines EFP *heißt Lösung eines* EFP*, wenn die partielle Lösung genau einen primären Eingang hat.*

Eine Lösung eines EFP besteht also aus einem Baum zwischen IN und OUT.

Für jedes EFP $\mathcal{P}$ gibt es eine Relation $\leq_{\mathcal{P}}$ auf der Menge der partiellen Lösungen von $\mathcal{P}$. Genauer: Falls C_1 und C_2 partielle Lösungen von $\mathcal{P}$ sind, dann gilt $C_1 \leq_{\mathcal{P}} C_2$ genau dann, wenn

1. beide partiellen Lösungen dieselbe Anzahl von primären Eingängen haben,
2. die required time von C_1 nicht kleiner als die required time von C_2 ist und
3. C_1 nicht mehr kostet als C_2.

Falls $C_1 \leq_{\mathcal{P}} C_2$ gilt, dann bedeutet dies intuitiv, daß C_1 besser als C_2 ist, da C_1 höchstens soviel wie C_2 kostet und wenigstens so „schnell“ wie C_2 ist.

Die Relation $\leq_{\mathcal{P}}$ ist reflexiv und transitiv.

Definition 2.5:
Eine (partielle) Lösung eines EFP $\mathcal{P}$ *ist* optimal*, wenn sie minimal bezüglich* $\leq_{\mathcal{P}}$ *ist.*

Intuitiv: Eine optimale Lösung kann nicht in der Zeit und den Kosten *gleichzeitig* verbessert werden.
Die Relation $\leq_{\mathcal{P}}$ ist nur partiell: Es kann viele unvergleichbare, optimale Lösungen eines Problems geben. „Wie eine Schere“ laufen die Kosten und die Zeiten dieser Lösungen auseinander.

Satz 2.6:
Das Fanout–Problem ist NP–vollständig.
Genauer: Gegeben sei ein EFP $\mathcal{P}$ *und eine Kostenschranke* K*. Dann ist das Entscheidungsproblem „Gibt es eine Lösung von* $\mathcal{P}$*, deren Kosten höchstens* K *sind und die eine nichtnegative required time hat?“ NP–vollständig.*

Im Beweis wird ein stärkeres Resultat gezeigt: Das EFP mit polynomiell beschränkten required times ist NP–vollständig.

Beweis von Satz 2.6:

<u>EFP∈NP?</u>
Die Kosten einer „erratenen“, gültigen Lösung können in polynomiell beschränkter Zeit berechnet werden.

NP vollständig?
Um zu beweisen, daß das Fanout–Problem NP–vollständig ist, wird eine Reduktion durchgeführt: Jede Instanz des X3C–Problems (Exact Cover by 3–Sets) kann auf eine Instanz des Fanout–Problems reduziert werden.

Eine Instanz des X3C–Problems hat die folgende Form:
N sei ein positives Vielfaches von 3, X sei die Menge der natürlichen Zahlen von 1 bis N und $C = \{S_1, S_2, \ldots, S_K\}$ (mit $K \in \mathbb{N}$) sei eine Menge, deren Elemente dreielementige Teilmengen von X sind. Das X3C–Problem lautet: „Gibt es eine Teilmenge D von C, für die die disjunkte Vereinigung ihrer Elemente die Menge X vollständig überdeckt?“, mit anderen Worten: „Gibt es $\frac{N}{3}$ Elemente aus C, d.h. $\frac{N}{3}$ dreielementige Mengen, die zusammen aus den Zahlen 1 bis N bestehen?“.

Das X3C–Problem ist NP–vollständig nach [GJ79]: Das 3DM Problem (3–Dimensional Matching - das „Verheiratungsproblem“ von Elementen dreier Mengen) läßt sich auf das X3C–Problem reduzieren. Das NP–vollständige Problem 3SAT (3–Satisfiability - Erfüllbarkeitsproblem mit 3 Literalen) kann auf das 3DM–Problem reduziert werden.

Gegeben sei eine Instanz $\mathcal{D}$ des X3C–Problems. Es wird daraus nun ein EFP $\mathcal{P}$ mit $2N + 1$ primären Ausgängen konstruiert: Der erste primäre Ausgang habe die required time 1. Den primären Ausgängen $2i$ und $2i+1$ mit $i = 1, \ldots, N$ werden die required times $(6N - i)K + 1$ zugeordnet.
Die Technologie habe drei verschiedene Klassen von Bausteinen:

- Eine „große Box“ mit $1 + \frac{N}{3}$ Ausgängen, Verzögerung 1 und Kosten $(NK)^3$.
- Für $i = 1, \ldots, K$ gibt es eine „i–mittlere Box“ mit 3 Ausgängen, Verzögerung $2NK + i$ und Kosten $(NK)^2 - 3i$.
- Für $i = 1, \ldots, K$ und $j = 1, 2, 3$ gibt es eine „ij–kleine Box“ mit 2 Ausgängen, Verzögerung $4NK - Ks_{ij} - i$ und Kosten $i + Ks_{ij}$.
  s_{i1}, s_{i2} und s_{i3} sind die Elemente der Menge S_i (der Instanz des X3C–Problems).

Lemma 2.7 :
Dieses EFP *hat eine Lösung mit nichtnegativen required times und Kosten* $\leq (NK)^3 + \frac{N}{3}(NK)^2 + \frac{N}{2}(N-1)K$ *genau dann, wenn das X3C Problem $\mathcal{D}$ eine Lösung hat.*

Eine „S_i–Einheit“ sei eine i–mittlere Box, die zu der i1–kleinen, der i2–kleinen und der i3–Box in der im Bild 2.3a gezeigten Weise verbunden sei. Ohne Beschränkung der Allgemeinheit kann angenommen werden, daß $S_{i1}, \ldots, S_{i\frac{N}{3}}$ eine exakte Überdeckung von $\mathcal{D}$ ist.
Die in Bild 2.3b dargestellte Schaltung, bei der die Ausgangskanten jeder ij–kleinen Box

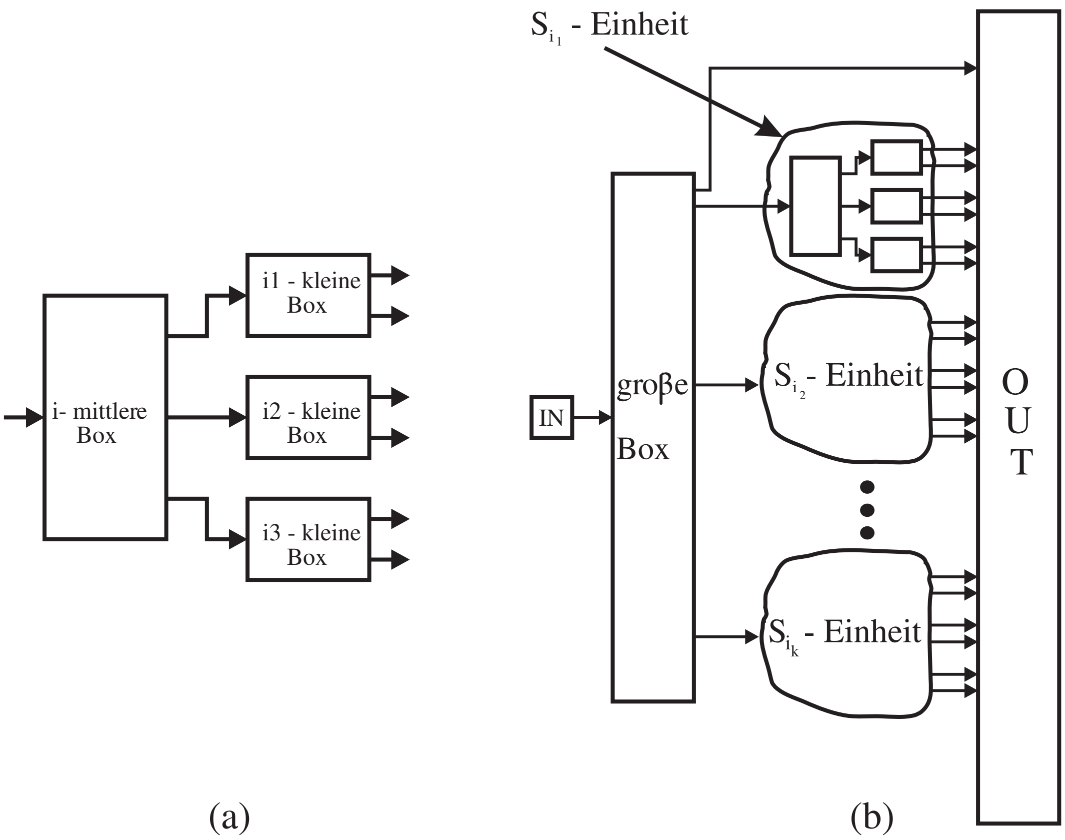

Abbildung 2.3: Konstruktion zum Beweis der NP–Vollständigkeit

mit den primären Ausgängen $2s_{ij}$ und $2s_{ij}+1$ verbunden sind, zeigt eine Situation, die die Überdeckung als Lösung des Fanout–Problems wiedergibt. Die Tatsache, daß $\mathcal{D}$ eine exakte Überdeckung ist, stellt sicher, daß jeder Ausgang genau eine Verbindung hat.

Aus Lemma 2.7 folgt Satz 2.6. Lemma 2.7 wird in zwei Schritten gezeigt. Zunächst wird bewiesen, daß die Schaltung in Abbildung 2.3b die Zeitforderungen und Kostenbeschränkungen erfüllt. Anschließend zeigen wir, daß die einzig mögliche Schaltung, die die Zeitforderungen und Kostenforderungen von $\mathcal{P}$ erfüllt, von der Form der Abbildung 2.3b sein muß.
Die folgenden Hilfslemmata, Lemmata und ihre jeweiligen Beweisideen stammen aus [BCD89], jedoch fehlen dort die entsprechenden Beweise.

Die Schaltung in Abbildung 2.3b erfüllt die Zeitforderungen und Kostenbeschränkungen:
Zunächst zwei Hilfslemmata:

- Die required time jeder S_i–Einheit ist 1.

- Die Kosten jeder S_i–Einheit sind $(NK)^2 + K(s_{i1} + s_{i2} + s_{i3})$.

Beweis dieser beiden Hilfslemmata:

- Wir betrachten die Einheit S_x. Sie besteht aus einer x–mittleren Box sowie x1-, x2- und x3–kleinen Boxen. Die mittlere Box hat die Verzögerung $2NK+x$, die xj–kleine Box hat die Verzögerung $4NK - K \cdot s_{xj} - x$ (für j=1,2,3). Die beiden ausgehenden Kanten der xj–kleinen Box seien in Abbildung 2.3b mit dem primären Ausgang mit der required time $(6N - s_{xj})K + 1$ verbunden. Dann ergibt sich für die required time am Eingang der xj–kleinen Box folgender Wert: $(6N - s_{xj})K + 1 - (4NK - K \cdot s_{xj} - x) = 2NK + x + 1$; dieses Ergebnis ist unabhängig von j, also identisch für die drei kleinen Boxen (d.h. Minimumbildung führt zum gleichen Ergebnis). Am Eingang der mittleren Box, also am Eingang der S_i–Einheit ergibt sich die required time zu $2NK + x + 1 - (2NK + x) = 1$

- Die mittlere Box der S_x Einheit kostet $(NK)^2 - 3x$, die xj–kleine Box kostet $x + K \cdot s_{x1}$
  Die Gesamtkosten der S_x–Einheit sind also $(NK)^2 + K(s_{x1} + s_{x2} + s_{x3})$

Da die große Box die Verzögerung 1 hat, werden die Zeitforderungen erfüllt.
Die Gesamtkosten der Schaltung ergeben sich aus den Kosten der großen Box zusammen mit der Addition der Kosten der S_i–Einheiten:

$$(NK)^3 + \sum_{x=1}^{K}[(NK)^2 + K(s_{x1} + s_{x2} + s_{x3})] = (NK)^3 + \frac{N}{3}(NK)^2 + K\frac{N}{2}(N-1)$$

Damit ist gezeigt, daß die Kostenforderung von der Schaltung in Abbildung 2.3b ebenfalls erfüllt ist. ■

Die einzig mögliche Schaltung, die die Zeitforderungen und Kostenforderungen von $\mathcal{P}$ erfüllt, muß von der Form der Abbildung 2.3b sein:
Die Schaltung stellt eine exakte Überdeckung für $\mathcal{D}$ dar. Die folgenden Lemmata werden benötigt, um den Beweis zu vervollständigen.

1. Der primäre Eingang muß an eine große Box angeschlossen werden.

2. Es darf höchstens $\frac{N}{3}$ mittlere Boxen geben.

3. Kein Pfad darf mehr als zwei Boxen hinter der großen Box durchlaufen.

4. Vorausgesetzt die vorangegangenen Lemmatas gelten, dann müssen $\frac{N}{3}$ Kanten von der großen Box zu mittleren Boxen führen. Die jeweils 3 Ausgangskanten der mittleren Boxen müssen mit kleinen Boxen verbunden sein.

5. Die $(NK)^3 + \frac{N}{3}(NK)^2 + \frac{N}{2}(N-1)K$–Kostenbeschränkung ist für das Problem eine untere Schranke, also scharf.
   Die einzige Möglichkeit, diese untere Schranke einzuhalten, ist die Schaltung in Abbildung 2.3b.

Die Beweise der Lemmata 1 bis 5:

1. Die required time des ersten primären Ausgangs kann nur dadurch eingehalten werden. Daraus folgt weiter: Der erste primäre Ausgang muß direkt mit einem Ausgang der großen Box verbunden sein.

2. Die Kostenforderung wird sonst überschritten.

   (a) Fall N=3: Dann gibt es 7 primäre Ausgänge und die große Box hat 2 Ausgänge. Wegen 1. muß man eine große Box verwenden, ihren Eingang an den primären Eingang anschließen und einen ihrer Ausgänge mit dem ersten primären Ausgang verbinden. Es verbleibt ein freier Ausgang an der großen Box. Mit zwei mittleren Boxen können wir das nun gewünschte Fanout (nämlich 6) nicht erreichen, haben jedoch die Kosten ausgeschöpft (beziehungsweise für $K > 1$ bereits überschritten): Die „billigste“ mittlere Box kostet $(NK)^2 - 3K$. Zwei davon und die große kosten für N=3 insgesamt $27K^3 + 18K^2 - 6K$, während die Kostenschranke sich auf $27K^3 + 9K^2 + 3K$ beläuft.

   (b) Fall $N \geq 6$: Die große Box kostet $(NK)^3$. Angenommen es gibt $\frac{N}{3} + 1$ mittlere Boxen. Die „billigste“ mittlere Box kostet $(NK)^2 - 3K$. Die minimalen Kosten für die mittleren Boxen sind also:

   $$(\frac{N}{3} + 1) \cdot [(NK)^2 - 3K] = \frac{N}{3} \cdot (NK)^2 + (NK)^2 - NK - 3K$$

   Vergleicht man dies mit der Kostenbeschränkung, so kann man folgende Ungleichung aufstellen. Falls sie gilt, dann ist das Kostenlimit überschritten.

   $$(NK)^2 - NK - 3K > \frac{N}{2}(N-1)K$$

   Formt man sie um, erhält man (mit $K \geq \frac{N}{3}$):

   $$6K^3 - 3K^2 - K - 2 > 0$$

   Dies gilt, da $K \geq 2$ (folgt aus $N \geq 6$ und $K \geq \frac{N}{3}$).

3. Die große Box hat bereits Kosten von $(NK)^3$, die zu untersuchenden Pfade haben also hinter der großen Box keine weitere große Box (sonst wird die Kostenbeschränkung überschritten). Es wird nun bewiesen, daß kein Pfad hinter der großen Box mehr als zwei Boxen durchlaufen darf, da sonst die Zeitbeschränkungen verletzt werden:

(a) Geht ein Pfad hinter der großen Box durch zwei kleine Boxen, so sind die Zeitforderungen verletzt.
Beweis: Zwei hintereinander geschaltete kleine Boxen verzögern ein Signal um $4NK - Ks_{i_1j} - i_1 + 4NK - Ks_{i_2j} - i_2$, die große Box hat die Verzögerung 1. Die Verzögerung durch diese Bausteine ist also kleiner oder gleich (da $s_{i_1j} + s_{i_2j} \leq N + N - 1$) wie $6NK - K + 2$. Dieser Term ist jedoch größer als $6NK - iK + 1$, unabhängig von $i \in \mathbb{N}$.

(b) Auf jedem Pfad hinter der großen Box dürfen keine zwei mittleren und eine weitere Box durchlaufen werden.
Beweis: Zwei mittlere und eine kleine Box haben eine Gesamtverzögerung von $2NK + i_1 + 2NK + i_2 + 4NK - Ks_{i_3j} - i_3$, was größer als $6NK + 1$ abgeschätzt werden kann. Dies verletzt die $6NK - iK + 1$ Zeitbeschränkung, unabhängig von $i \in \mathbb{N}$.
Die Verzögerung durch drei mittlere Boxen und die große Box ist mindestens $6NK + 4$, was ebenfalls das Zeitlimit übersteigt.

Hier wurde ein stärkeres Resultat gezeigt: Auf jedem Pfad hinter der großen Box dürfen höchstens 2 Boxen durchlaufen werden, wobei eine davon eine mittlere Box sein muß.

4. Es werden hierzu zwei Hilfslemmata gezeigt:

   (a) Jeder Pfad hinter der großen Box geht durch genau 2 Boxen.

   (b) Jeder Pfad hinter der großen Box geht zuerst durch eine mittlere und dann durch eine kleine Box.

Die Hilfslemmata können wie folgt bewiesen werden:

(a) Jeder Pfad hinter der großen Box geht durch genau 2 Boxen.
Daß jeder Pfad höchstens 2 Boxen hinter der großen Box durchläuft, wurde bereits in 3. gezeigt. Es bleibt noch zu zeigen: Es sind mindestens 2 Boxen.
Für diesen Beweis ist folgendes zu beachten: Die required times müssen eingehalten werden und alle $2N$ freien primären Ausgänge müssen das Signal erhalten. Die zweite Forderung ist kritisch (wie noch gezeigt wird), deswegen werden Boxen mit möglichst vielen Ausgängen eingesetzt. Weitere große Boxen können wegen der Kosten nicht eingesetzt werden, es wird jedoch versucht, alle $\frac{N}{3}$ im Beweis von 2. erlaubten mittleren Boxen einzusetzen, da sie drei Ausgänge haben, während die kleinen Boxen nur zwei Ausgänge haben. Im folgenden werden drei Annahmen über Pfade gemacht, die zum Widerspruch geführt werden, indem in jedem Fall gezeigt wird, daß das verlangte Fanout von $2N$ nicht erreicht wird. Damit ist bewiesen, daß mindestens zwei Boxen hinter der großen Box durchlaufen werden müssen.

- Annahme: Es gibt Pfade hinter der großen Box, die durch keine weitere Box laufen.

Wenn es nur einen solchen Pfad gibt, dann führt dies bereits zum Widerspruch: Durch diesen Pfad wird ein primärer Ausgang belegt. An die verbleibenden $\frac{N}{3} - 1$ freien Anschlüsse der großen Box können ebenso viele mittlere Boxen eingesetzt werden. Dadurch entstehen $N - 3$ freie Anschlüsse an den Ausgängen der mittleren Boxen. Man setzt die verbleibende mittlere Box und $N - 4$ kleine Boxen an diese freien Ausgänge. Das von dieser Schaltung insgesamt erreichte Fanout ist jedoch nur $1 + 3 + 2 \cdot (N - 4) = 2N - 4$ statt der verlangten $2N$.

- Annahme: Es gibt Pfade hinter der großen Box, die durch lediglich eine mittlere Box laufen.
  Auch hier führt die Existenz eines einzigen solchen Pfades bereits zum Widerspruch: Durch diesen Pfad werden 3 primäre Ausgänge belegt. An die verbleibenden $\frac{N}{3} - 1$ freien Anschlüsse der großen Box können die noch zur Verfügung stehenden mittleren Boxen eingesetzt werden. Dadurch gibt es $N - 3$ freie Anschlüsse, an die kleine Boxen angeschlossen werden. Das von dieser Schaltung insgesamt erreichte Fanout ist jedoch nur $3 + 2 \cdot (N - 3) = 2N - 3$ statt der verlangten $2N$.
- Annahme: Es gibt Pfade hinter der großen Box, die durch lediglich eine kleine Box laufen.
  Ein einziger solcher Pfad führt bereits zum Widerspruch: Durch diesen Pfad werden 2 primäre Ausgänge belegt. An die verbleibenden $\frac{N}{3} - 1$ freien Anschlüsse der großen Box können ebenso viele mittlere Boxen eingesetzt werden. Dadurch entstehen $N - 3$ freie Anschlüsse an den Ausgängen der mittleren Boxen. Man setzt die verbleibende mittlere Box und $N - 4$ kleine Boxen an diese freien Ausgänge. Das von dieser Schaltung insgesamt erreichte Fanout ist jedoch nur $2 + 3 + 2 \cdot (N - 4) = 2N - 3$ statt der verlangten $2N$.

Wenn man in den drei vorangegangenen Beweisen mehr als einen Pfad durch eine oder keine Box hinter der großen Box hat, dann werden noch geringere Fanout–Werte erreicht. Ebenso kann man das Fanout nicht dadurch erhöhen, daß man zuerst die kleinen und dann die mittleren Boxen einsetzt.

(b) Jeder Pfad hinter der großen Box geht zuerst durch eine mittlere und dann durch eine kleine Box.
Angenommen ein Ausgang der großen Box sei mit einem Eingang einer kleinen Box verbunden. Die kleine Box hat zwei Ausgänge, die beide an Boxen angeschlossen werden müssen, da jeder Pfad hinter der großen Box nach a) durch 2 Boxen laufen muß. Bereits in 3. wurde gezeigt, daß kein Pfad hinter der großen Box durch zwei kleine Boxen führen darf. Also verbindet man die beiden Anschlüsse der kleinen Box mit mittleren Boxen. Somit verbleiben für die $\frac{N}{3} - 1$ freien Ausgänge der großen Box nur noch $\frac{N}{3} - 2$ mittlere Boxen. Alle freien Anschlüsse müssen belegt werden und zwei Boxen durchlaufen, sonst wird das Fanout nicht erreicht. Es gibt also mindestens einen Pfad, bei dem 2 kleine

Boxen hintereinander geschaltet sind.

5. Nach 4. muß jeder Pfad vom primären Eingang zu einem primären Ausgang über die große, eine mittlere und eine kleine Box laufen. Bei der Wahl dieser Boxen und der Verbindungen zwischen ihnen muß gewährleistet sein: Es darf keine negativen required times geben. Gleichzeitig muß man die Kosten minimieren (es ist leicht nachprüfbar, daß ohne Kostenminimierung die Kostenschranke nicht erreicht wird). Im folgenden werden Boxen und Verbindungen ausgewählt, die die Zeitspezifikationen bei minimalen Kosten erfüllen.
   Man verfolgt hierzu den Pfad von den primären Ausgängen zu den primären Eingängen. Je zwei primäre Ausgänge haben dieselbe required time. Es macht keinen Sinn, die beiden Ausgänge einer kleinen Box an primäre Ausgänge mit unterschiedlichen required times anzuschließen, da man dann durch die Minimumbildung auf jeden Fall schlechtere Zeitwerte erzielt. Die jeweils zwei Ausgänge der i_1j-kleinen Box werden also an die primären Ausgänge mit den required times $(6N - i_2)K + 1$ angeschlossen. Dies ergibt eine required time am Eingang der kleinen Box von $2NK + K(s_{i_1j} - i_2) + i_1 + 1$. Drei kleine Boxen werden an die i_3-mittlere Box angeschlossen. Deren Eingang hat also eine required time von

$$\min_{j=1,2,3}\{2NK + K(s_{i_1j} - i_2) + i_1 + 1\} - 2NK - i_3 = K \cdot \min_{j=1,2,3}\{s_{i_1j} - i_2\} + i_1 - i_3 + 1$$

   Die große Box verzögert das Signal um 1. Der Term $K \cdot \min_{j=1,2,3}\{s_{i_1j} - i_2\} + i_1 - i_3$ muß also größer oder gleich 0 sein. Damit man minimale Kosten erhält, sollte s_{i_1j} möglichst klein gewählt werden. Ein negativer Wert von $K \cdot \min_{j=1,2,3}\{s_{i_1j} - i_2\}$ müßte jedoch durch den Term $i_1 - i_3$ ausgeglichen werden. Dies ist jedoch nicht möglich, da der Term $i_1 - i_3$ nicht größer als $K - 1$ sein kann. Der niedrigste erlaubte Wert für s_{i_1j} ist demnach i_2. Somit hat der erste Term den Wert 0 und $i_1 - i_3$ muß größer oder gleich 0 sein. Um minimale Kosten für die mittlere Box zu erhalten, versucht man i_3 möglichst groß zu wählen. i_3 kann jedoch nicht größer als i_1 sein, da sonst die Zeitforderungen verletzt sind. Zusammengefaßt: Versucht man die Zeitforderungen bei minimalen Kosten zu erfüllen, so folgern die Bedingungen $s_{i_1j} = i_2$ und $i_2 = i_3$. Mit diesen Bedingungen entstehen Gesamtkosten (die drei Summanden sind die Gesamtkosten für die große Box, die mittleren Boxen und die kleinen Boxen) von:

$$(NK)^3 + \sum_{i=1}^{K}[(NK)^2 - 3i] + \sum_{i=1}^{K}\sum_{j=1}^{3}(i + Ks_{ij}) = (NK)^3 + \frac{N}{3}(NK)^2 + \frac{N}{2}(N-1)K$$

   Daraus folgt: Die Kostenbeschränkung ist für das gestellte Problem eine untere Schranke, also scharf.

   Die Bedingungen $s_{i_1j} = i_2$ und $i_1 = i_3$ implizieren weiter: Die in Abbildung 2.3b gezeigte Schaltung ist die einzige Schaltung, die die Kosten- und Zeitforderungen erfüllt.

■

Kapitel 3

Der Algorithmus von Singh und Sangiovanni-Vincentelli

Im letzten Kapitel wurde festgestellt, daß keine Polynom–Zeit Lösung (einer deterministischen Turing Maschine) für das Fanout–Problem bekannt ist. Aus diesem Grund wurden eine Vielzahl von Heuristiken für flache Schaltungen entwickelt. Die in [SSV90] beschriebene Heuristik wird in diesem Kapitel näher betrachtet, da sie viele Ansätze und Techniken früherer Fanout–Algorithmen in sich vereint:

- Einfaches Zeitmodell von Bausteinen und einfache Zeitanalyse der Schaltung.
- ϵ–Netzwerk und Mincut–Technik zur Lokalisierung der Fanout–Probleme.
- Ersetzen von Bausteinen durch logisch äquivalente, die passendere Treibereigenschaften haben.
- Rekursive Zerlegung in kleinere Teilprobleme.
- Generierung vieler Lösungsvorschläge.
- Akzeptieren des besten generierten Lösungsvorschlags, wenn diese Lösung zu einer besseren Signallaufzeit führt.

3.1 Beobachtungen und Idee

Folgende Beobachtungen und Schlußfolgerungen liegen dem Algorithmus zugrunde:

- Technologien bestehen häufig aus Klassen von Bausteinen. Die Bausteine einer Klasse realisieren dieselbe logische Funktion, unterscheiden sich jedoch voneinander in ihren Treiberstärken. Um möglichst schnelle Schaltungen zu erhalten, sollte man einen Baustein einer Klasse abhängig vom Schaltungskontext auswählen, Entscheidungskriterium hierbei ist die zu treibende Kapazität am Ausgang.

- Aufgrund der Komplexität des Fanout–Problems betrachten die Autoren ein einfacheres Problem, indem sie das zweite Optimierungskriterium (die Kosten) weglassen (beziehungsweise nur indirekt berücksichtigen).
  Verzögerungszeit und Kosten von Bausteinen einer Technologie kann man jedoch im allgemeinen nicht voneinander trennen. Als Faustregel kann man sagen: Bausteine mit guten Treibereigenschaften (d.h. solche mit hohem Ausgangsstrom), die also viele nachfolgende Eingänge treiben können, sind teuer (zum Beispiel flächenintensiv) und haben im Vergleich zu logisch äquivalenten mit weniger guten Treibereigenschaften eine höhere interne Verzögerungszeit.
  Der Algorithmus modifiziert eine gegebene Schaltung in zweierlei Weise:

  1. Er führt ein *repowering* für Bausteine durch, deren Treiberstärke für ihren Schaltungskontext überdimensioniert oder unterdimensioniert sind. Eine überdimensionierte Treiberstärke eines Ausgangs hat zur Folge, daß eine unnötige Verzögerung durch die hohe interne Verzögerung des Treibers entsteht und daß die Kosten unnötig hoch sind. Ausgänge mit unterdimensionierter Treiberstärke verzögern die Durchlaufzeit durch den hohen Wert ihrer externen Verzögerung. Die erwähnten Bausteine werden - wenn möglich und nur an den kritischen Stellen der Schaltung - durch logisch äquivalente Bausteine ersetzt. Die Ersatzbausteine haben für den Kontext passendere Treiberstärken und verkürzen die Durchlaufzeit eines Signals. Bei überdimensionierten Treiberstärken wird hierbei eine Ersetzung durchgeführt, die sich zusätzlich positiv auf die Kosten der Schaltung auswirkt, bei unterdimensionierten Treiberstärken hat die Verbesserung der Durchlaufzeit höhere Priorität als die entstehenden Mehrkosten.
  2. Er setzt Treiberbäume ein. Auch hierbei wird versucht, Bausteine einzusetzen, deren Treiberstärke möglichst vollständig ausgenutzt wird.

  Das zweite Optimierungskriterium wird also indirekt - wenn auch nur teilweise - durch den Algorithmus berücksichtigt, indem er versucht, die Treiberstärken möglichst auszuschöpfen.

- Es gibt Schaltungen mit „großen" Fanout–Problemen, d.h. Fanout–Probleme mit vielen Senken. Bei automatisch generierten Schaltungen treten solche Probleme relativ häufig auf, da der Entwerfer Regeln zum Generieren spezifiziert, ohne oftmals voraussagen zu können (oder zu beachten), wie sich diese Regeln auf das Fanout–Problem auswirken. Der betrachtete Algorithmus trägt diesem Umstand Rechnung, indem er die Fanout–Signale partitioniert und jedes Teilproblem rekursiv löst.

- Beim Entwurf einer Schaltung wird in manchen Fällen nicht nur das Signal am Ausgang eines Bausteins abgegriffen, sondern auch das negierte. Wenn man einen Treiber als Hintereinanderschaltung zweier Inverter modelliert (in CMOS wird dies technologiebedingt erzwungen), dann kann man Algorithmen entwickeln, die diesen Umstand zum Abgriff des invertierten Signals ausnutzen.

3.2 Bezeichnungen

Für die (hier nicht näher spezifizierte) Technologie wird gefordert, daß sie nur Bausteine mit genau einem Ausgang enthält.

Eine kombinatorische Schaltung wird als gerichteter, azyklischer Graph (dag = directed acyclic graph) modelliert. Die Bausteine und primären Anschlüsse werden zu Knoten, die Verdrahtung wird in Kanten umgewandelt. Die primären Eingänge sind im dag die einzigen Knoten mit Eingangsgrad 0, die primären Ausgänge sind die einzigen Knoten mit Ausgangsgrad 0. Die Kanten werden aus der Verdrahtung wie folgt gewonnen: Falls es eine Verbindung vom Ausgang des Bausteins v_i zum Eingang des Bausteins v_j gibt, dann gibt es eine Kante vom Knoten v_i zum Knoten v_j (Bausteine und ihre entsprechenden Knoten im dag werden mit denselben Namen identifiziert). Falls ein primärer Eingang v_x mit einem Eingang des Bausteins v_k verbunden ist, gibt es eine Kante von v_x zu v_k. Schließlich: Ist ein Ausgang des Bausteins v_l mit einem primären Ausgang v_z verbunden, dann gibt es eine Kante von v_l zu v_z.
Für einen Knoten v gilt:

- Die Fanout–Menge FO_v ist die Menge der Ziele der ausgehenden Kanten des Bausteins v.
- Die Fanin–Menge FI_v des Knotens v bezeichnet die Menge der Knoten, die eine Ausgangskante mit Ziel v haben. Innerhalb von FI_v wird zusätzlich jedem Element die Nummer des von ihm in v belegten Eingangs zugeordnet.

3.3 Die Zeitanalyse

Um die Stellen zu finden, an denen Fanout–Probleme gelöst werden sollen, führt der Algorithmus eine Zeitanalyse der Signallaufzeiten der Schaltung durch.

Das Verzögerungsmodell

Die required times an den primären Ausgängen und die arrival times an den primären Eingängen sind bekannt. Die Verzögerung zwischen Bausteineingang und Bausteinausgang berechnet dieser Algorithmus anhand eines parametrisierten, linearen Verzögerungsmodells. Jedem Baustein g mit n Eingängen werden n Tripel zugeordnet. Das Tripel (c_k^g, i_k^g, e_k^g) mit $k = 1, \ldots, n$ steht für die Verzögerung des Bausteins zwischen dem k-ten Eingang und dem Bausteinausgang. Hierbei ist c_k^g die Eingangskapazität, i_k^g die *interne* Verzögerung dieses Eingangs, und e_k^g ist der Faktor für die *externe* Verzögerung. Die Kapazitäten werden als Vielfache einer technologieabhängigen Lasteinheit (*unit load*) angegeben, die Einheit für e_k^g ist somit: Verzögerungszeit pro Lasteinheit. Die arrival time an einem primären Eingang sagt aus, wann das Signal an den Eingängen der angeschlossenen Bausteine anliegt; das zum entsprechenden Knoten gehörende Tripel wird nicht in die Berechnungen einbezogen, es kann daher auf den Nullvektor gesetzt werden. Für das Tripel eines Knotens, der für einen primären Ausgang steht, muß lediglich die Kapazität spezifiziert werden, die anderen beiden Verzögerungswerte sind Null. Um die Leitungskapazität zu berücksichtigen, ordnet man jeder Kante die einheitliche, mittlere Kapazität C_w zu. Ein genauerer Wert kann nicht angegeben werden, da der Algorithmus auf der Ebene des Logikentwurfs arbeitet und der Verlauf der Verdrahtung noch nicht feststeht. Es sei I_g^h der Index des Eingangspins des Bausteins h, der von Baustein g getrieben wird. Dann berechnet sich die Verzögerung zwischen dem k-ten Eingang und dem Ausgang von Baustein g nach folgender Formel:

$$d_k^g = i_k^g + e_k^g \cdot \sum_{h \in FO_g} (C_w + c_{I_g^h}^h) \tag{3.1}$$

Die Notation r_{gh} bezeichne die required time eines Signals auf der Kante zwischen Baustein g und Baustein h. Mit r_g wird hingegen die required time am Ausgang des Bausteins g identifiziert (Anmerkung: diese Zuordnung der required times zu Kanten und Bausteinen unterscheidet sich von denen im Beweis der NP–Vollständigkeit des letzten Kapitels; siehe hierzu auch die weiter unten stehenden Formeln 3.3 und 3.4). Für einen Knoten, der einen primären Eingang repräsentiert, spezifiziert r_g seine required time. Die arrival time des Bausteins g sei a_g. Sie ist die Zeit, zu der das Signal am Ausgang des Bausteins korrekt anliegt. Der Wert von a_g für primäre Ausgänge heißt die arrival time dieses Anschlusses.

Die Routine delay_trace

Die Routine delay_trace berechnet die Signallaufzeiten durch die Schaltung auf zwei verschiedene Arten. Im ersten Verfahren werden mittels der arrival times an den primären Eingängen und der rekursiven Formel 3.2

$$a_g = \max_{h \in FI_g} (a_h + d_{I_h^g}^g) \tag{3.2}$$

die arrival times an den Ausgängen aller Bausteine berechnet. Das zweite Verfahren geht von den required times an den primären Ausgängen aus. Anhand der beiden Formeln 3.3 und 3.4

$$\begin{aligned} r_{gh} &= r_h - d_{I_g^h}^h && (3.3) \\ r_g &= \min_{h \in FO_g} r_{gh} && (3.4) \end{aligned}$$

kann die required time am Ausgang jedes Bausteins berechnet werden. Um die Formeln auf den dag anwenden zu können, sortiert man den dag vorher topologisch. Man geht je nach Formel in topologischer Sortierrichtung (bei Formel 3.2) oder entgegengesetzt der topologischen Sortierrichtung (bei den Formeln 3.3 und 3.4) vor.
Nach Berechnung der arrival times und der required times wird für alle Knoten der im nächsten Abschnitt vorgestellte Schlupf bestimmt.

Kritische Stellen

Der Schlupf (*slack*) eines Bausteins ist die Differenz zwischen seiner required time und seiner arrival time:

$$s_g = r_g - a_g \tag{3.5}$$

Ein negativer Schlupf bedeutet, daß die Zeitspezifikationen nicht eingehalten werden. Der minimale Wert des Schlupfes wird mit S_{min} bezeichnet, er bestimmt den kritischsten Pfad. Ein Pfad von einem primären Eingang zu einem primären Ausgang, auf dem alle Knoten einen Schlupf $\leq (S_{min} + \epsilon)$ haben, heißt ϵ–kritischer Pfad. Die Menge der ϵ–kritischen Pfade erzeugt einen Teilgraphen des dag, der ϵ–Netzwerk genannt wird.

Satz 3.1:
Faßt man alle primären Eingänge des ϵ–Netzwerkes zu einem IN–*Knoten und alle primären Ausgänge des ϵ–Netzwerkes zu einem* OUT–*Knoten zusammen, dann gilt: Das ϵ–Netzwerk ist zusammenhängend.*

Satz 3.1 wird in [SSV90] nicht bewiesen, wir fanden den folgenden Beweis.

Beweis von Satz 3.1:

Voraussetzung: Der Baustein g gehöre zum ϵ–Netzwerk, formal: $s_g \leq S_{min} + \epsilon$.
Behauptung 1: Es gibt mindestens einen Knoten, der zum ϵ–Netzwerk gehört und der eine eingehende Kante hat, die bei g startet.
Beweis von Behauptung 1:
Sei b derjenige Nachfolger von g, für den in der Berechnung von r_g das Minimum der r_{gh} angenommen wurde:
Für die required time von b läßt sich Hilfsformel (I) herleiten:

$$\begin{aligned}
r_g &= \min_{h \in FO_g} r_{gh} \\
r_g &= r_{gb} \\
r_g &= r_b - d^b_{I^b_g} \\
r_b &= r_g + d^b_{I^b_g} \quad \text{(I)}
\end{aligned}$$

Für die arrival time von b läßt sich Hilfsformel (II) herleiten:

$$\begin{aligned}
a_b &= \max_{h \in FI_b} (a_h + d^b_{I^h_b}) \\
a_b &\geq a_g + d^b_{I^g_b} \quad \text{(II)}
\end{aligned}$$

b gehört zum ϵ–Netzwerk, denn

$$\begin{aligned} s_b &= r_b - a_b \\ &= r_g + d^b_{I^b_g} - a_b \quad \text{wegen (I)} \\ &\leq r_g + d^b_{I^b_g} - a_g - d^b_{I^g_b} \quad \text{wegen (II)} \\ &= r_g - a_g \\ &= s_g \\ &\leq S_{min} + \epsilon \end{aligned}$$

Behauptung 2: Es gibt einen Baustein c, wobei c eine ausgehende Kante zu g hat und zum ϵ–Netzwerk gehört.

Der Beweis ist analog zum Beweis der Behauptung 1.

Satz 3.1 ist eine Folgerung aus Behauptung 1 und 2. ∎

Informal formuliert ist das ϵ–Netzwerk der Schaltungsteil, bei dem die Zeitforderungen am stärksten verletzt werden.

3.4 Der Algorithmus buffer_network

Gegeben sei eine kombinatorische Schaltung $\mathcal{C}$ mit spezifizierten arrival times an den primären Eingängen und spezifizierten required times an den primären Ausgängen. Die Schaltung soll so modifiziert werden, daß für jeden Pfad von einem primären Eingang zu einem primären Ausgang gilt: Ein Signal, das zu der gegebenen arrival time am primären Eingang garantiert korrekt anliegt, muß spätestens zu der spezifizierten required time das Signal am primären Ausgang beeinflußt haben. Mit anderen Worten: Falls der Pfad sensitiv ist, dann muß ein Flankenwechsel am Eingang zur arrival time einen Flankenwechsel am Ausgang spätestens zur required time hervorgerufen haben.

Ein einzelnes Fanout–Problem ist ein lokales Problem für eine Schaltung. Welche Fanout–Probleme in welcher Reihenfolge gestellt werden sollen, ist dagegen ein globales Problem. Es kann sein, daß die Lösung eines Fanout–Problems der gegebenen Schaltung keine Signallaufzeit verbessert, da es andere Pfade zwischen den primären Anschlüssen gibt, die die verbesserte Stelle nicht beinhalten, die jedoch genauso kritisch sind.

Der Algorithmus buffer_network versucht, das globale Problem zu lösen, indem Stellen des ϵ–Netzwerkes bestimmt werden, an denen die lokale Verbesserung schließlich zu einer globalen Verbesserung führen soll. Auf diese Stellen wird dann die Routine buffer_node angewandt. Dieser Prozeß wird iterativ so oft durchgeführt, bis die Zeitforderungen an die Schaltung erfüllt sind, oder bis auf diesem Wege keine Verbesserung mehr erreicht werden kann.

Eingabe für buffer_network ist die Schaltung $\mathcal{C}$ und die Konstante ϵ.

```
proc buffer_network(C,ε)
begin
  delay_trace(C);
  while ((∃v with Slack(v) <0) ∧ (delay_decreases)) do
        nw := generate_an_ε–network(C,ε);
        F := find_Fanout_problems(nw);
        for all n ∈ F do
             TR := get_target_required_time(n)
             (FO⁺,n_I,FO⁻) := get_fanout_data(n);
             buffer_node(n,n_I,FO⁺,FO⁻,TR);
        od
        delay_trace(C);
  od
end
```

Die Zeitforderungen an die Schaltung sind erfüllt, wenn es keine Stellen mit negativem Schlupf mehr gibt. Nur wenn die Zeitforderungen nicht erfüllt sind und die Signallaufzeit beim letzten Durchlauf durch die **while**–Schleife besser wurde (*delay_decreases*; initial, d.h. für die ursprüngliche Schaltung ist deren Rückgabewert TRUE), wird der Rumpf der **while**–Schleife ausgeführt. Beide Bedingungen lassen sich mit Hilfe des vorangegangenen *delay_trace($\mathcal{C}$)*–Aufrufes testen. In *nw* wird das gefundene ϵ–Netzwerk abgespeichert, die Routine *find_Fanout_problems* findet die Stellen, für die ein Fanout–Problem gelöst werden soll. Man kann zwischen zwei Algorithmen wählen, die dies tun: cutset und extract, beide werden weiter unten beschrieben. Drei der Übergabevariablen für *buffer_node* werden durch *get_fanout_data* generiert. *buffer_node* und die Bedeutung seiner Parameter werden in einem späteren Abschnitt beschrieben. Die Zeit TR wird im nächsten Abschnitt erläutert.

Die Bedeutung und Berechnung von TR

Wenn man die required time einer ausgehenden Kante eines Bausteins vergrößert, dann kann man die Signallaufzeit eines Pfades durch die Schaltung nur so lange verbessern, bis man den Wert der required time der nächst kritischen ausgehenden Kante des Bausteins erreicht hat. Vergrößert man darüber hinaus, so hat dies keinen Einfluß auf die Signallaufzeit, da Pfade über die andere Kante nun kritischer sind. Somit läßt sich eine obere Schranke für die required time des Bausteins g angeben, die aussagt, bis zu welchem Punkt eine Fanout–Korrektur sinnvoll ist. Diese Schranke wird TR (= *<u>T</u>arget <u>R</u>equired time*) genannt. Sie berechnet sich nach der Formel

$$TR \;=\; r_g + (r_i - r_j) \tag{3.6}$$

und wird in Abbildung 3.1 veranschaulicht. Der Eingang des Bausteins g, der auf dem

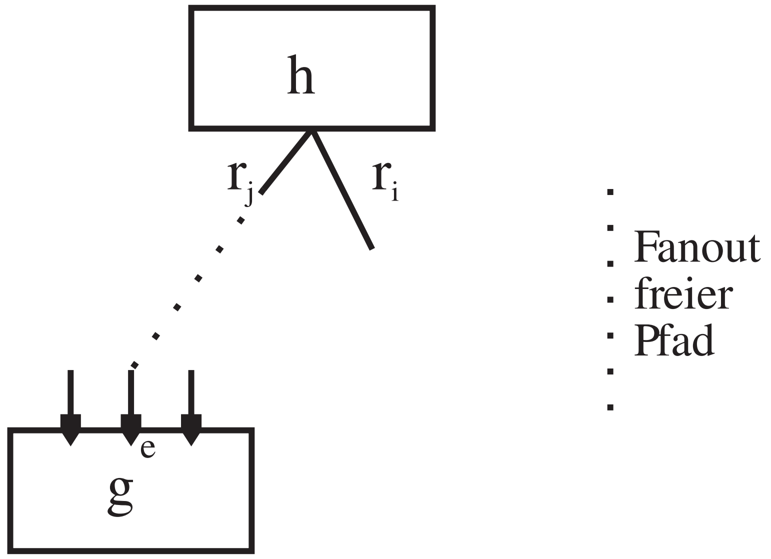

Abbildung 3.1: Die Berechnung von TR

kritischsten Pfad der Schaltung liegt und für dessen Ausgang das Fanout–Problem gelöst werden soll, sei mit e bezeichnet. Von e aus läuft man entlang des kritischsten Pfades gegen die Signalflußrichtung, bis man am Baustein h ankommt, wobei h der erste Baustein auf diesem Pfad ist, der mit seinem Ausgang mehr als einen Eingang treibt (der Pfad bis h wird als „Fanout freier Pfad" bezeichnet). j bezeichne die Ausgangskante von h mit der kritischsten required time. Sie liegt auf dem kritischsten Pfad. i sei die Kante mit der zweitkritischsten required time von h. Wenn nun der kritischste Pfad um mehr als $r_i - r_j$ verbessert wird, dann wird der Pfad über i kritischer als der über j.

Die Routine cutset

Die Idee, eine Schaltung „durchzuschneiden", um am Schnitt lokale Operationen durchzuführen, die eine globale Verbesserung der Signallaufzeiten der Schaltung erzielen sollen, wurde in [Mic87] vorgeschlagen und beispielsweise in [PP89] und [SSV90] umgesetzt. In diesen Heuristiken wird jedoch nicht formal nachgewiesen, daß die cutset ein gutes Konzept für dieses Problem ist, es wird lediglich qualitativ argumentiert. Im folgenden wird beschrieben, welche Eigenschaften der „Schnitt" (im folgenden *cutset* genannt) hat, und an welcher Stelle er nach [SSV90] gelegt wird.

Das ϵ–Netzwerk mit zusammengefaßten primären Anschlüssen ist nach Satz 3.1 ein zusammenhängender dag. Man betrachte nun folgenden dag: Die primären Anschlüsse seien wieder einzelne Knoten, es gibt jedoch einen IN Knoten, der Kanten zu allen primären

Eingangsknoten hat und einen OUT Knoten, der Ziel von Kanten aller primären Ausgangsknoten ist. IN ist dann der einzige Knoten mit Eingangsgrad 0, OUT der einzige mit Ausgangsgrad 0. Eine cutset ist eine minimale Menge (minimal im Sinne von: entfernt man Elemente aus dieser Menge, so geht die cutset–Eigenschaft verloren) von Kanten (diese Menge darf jedoch keine Kanten, die bei IN starten oder die bei OUT enden, enthalten) in diesem dag, die folgende Eigenschaft hat: Entfernt man die Kanten aus dem dag, so wird der dag in zwei disjunkte Teilgraphen partitioniert. Im folgenden wird mit dem Begriff cutset die Menge der Quellknoten dieser Kanten bezeichnet. Welche Knoten sollen nun die cutset für das Fanout–Problem bilden? Die Autoren in [SSV90] schlagen hierfür vor, daß Knoten mit hohem Ausgangsgrad bevorzugt ausgewählt werden. Die Knoten des dag erhalten zu diesem Zweck Gewichte. Die Gewichte werden umgekehrt proportional zum Ausgangsgrad eines Knotens vergeben. Wendet man nun einen minimum–cutset–Algorithmus wie z.B. den in [Law76] beschriebenen an, dann erhält man eine cutset, deren Knotengewichtssumme minimal ist.

Die Routine extract

Um die Effizienz der cutset–Strategie zu überprüfen, entwickelten wir eine eigene Routine für dieses Problem namens *extract*:

> **proc** *extract* (nw)
> **begin**
> $M := nw$;
> $R := \emptyset$;
>   **while** $M \neq \emptyset$ **do**
> $g :=$ *M.delete_node_with_minimum_weight* ();
> $R := R \cup \{g\}$;
> $\forall p$ *which are predecessors on paths through g: M. delete* (p);
> $\forall s$ *which are successors on paths through g: M. delete* (p);
>   **od**
>   **return** R;
> **end**

Als Parameter erhält extract das ϵ–Netzwerk *nw*. Anfangs enthält M alle Knoten des ϵ–Netzwerks. Die Funktion *delete_node_with_minimum_weight* ist eine Klassenfunktion und wird auf die Menge M angewendet. Sie löscht den (beziehungsweise einen der) Knoten mit minimalem Gewicht aus M und gibt ihn dem Aufrufer zurück. Anschließend werden alle Vorgänger (inklusive der transitiven) und Nachfolger (ebenfalls inklusive der transitiven) aus M gestrichen. Dieser Vorgang wird solange wiederholt, bis M leer ist.

Im letzten Abschnitt dieses Kapitels werden die Ergebnisse von *extract* und von *mincut* für Beipielschaltungen miteinander verglichen.

3.5 Der Algorithmus buffer_node zur Lösung des Fanout Problems

Die Routine *find_Fanout_problems* (d.h. entweder *mincut* oder *extract*) gibt eine Menge von Knoten zurück. Auf jeden dieser Knoten wird der birekursive Algorithmus buffer_node mit folgenden Parametern angewandt (vgl. Abbildung 3.2):

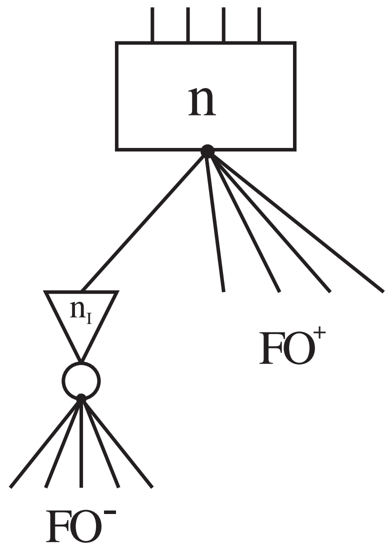

Abbildung 3.2: Aufteilung und Bezeichnungen der Fanout Mengen

- n ist der Knoten, für dessen Ausgang das Fanout–Problem gelöst werden soll.
- n_I ist der Inverter, der das invertierte Signal von n verteilt. Sind mehrere Inverter mit dem Ausgang von n verbunden und wurde buffer_node direkt von buffer_network aufgerufen (d.h. buffer_network ist der dynamische Vorgänger; es handelt sich nicht um einen rekursiven Aufruf), dann nimmt man für n_I den Inverter mit dem größten Ausgangsgrad.
- FO^+ ist die Menge der Senken des Ausgangs von n ohne n_I. Es gelte $k := |FO^+|$
- FO^- ist die Menge der Senken des Ausgangs von n_I. Es gelte $l := |FO^-|$
- TR (wie TR berechnet wird, wurde bereits erläutert)

Es gelte $FO := FO^+ \cup FO^-$ und im folgenden heiße FO die Fanout–Menge von n.

Für jeden Baustein h der Fanout–Menge FO kennen wir

1. die Kapazität c_h, die dem Eingang von h zugeordnet ist.
2. die required time r_{nh} des Signals am Eingang von h.

Die required times von FO^+ seien aufsteigend sortiert und nach ihrem Sortierindex benannt: $r_1^+ \leq r_2^+ \leq \ldots \leq r_k^+$. Auch FO^- liege sortiert vor: $r_1^- \leq r_2^- \leq \ldots \leq r_l^-$.

Sei M die Anzahl der verschiedenen Inverter der Technologie. Die Inverter verwendet man im hiesigen Algorithmus zum Aufbau von Fanout–Bäumen. In CMOS realisiert man Puffer durch zwei kaskadierte Inverter. Für den hier betrachteten Algorithmus werden Puffer prinzipiell als hintereinandergeschaltete Inverter ausgeführt. Dies hat - technologieunabhängig - den Vorteil, daß das negierte Signal in einfacher Weise abgegriffen werden kann. Der Nachteil dieser Vorgehensweise liegt jedoch für Technologien mit bereits spezifizierten nichtinvertierten Puffern auf der Hand: Diese Puffer können nicht ohne weiteres verwendet werden.

Das Fanout–Problem stellt sich intuitiv wie folgt: Aufgrund des hohen Ausgangsgrades von n (und eventuell von n_I) und der damit verbundenen hohen Kapazität erreicht man in vielen Fällen eine Verbesserung der Signallaufzeiten, indem man das Signal mittels eines Fanout–Baumes verteilt. Da die Inverter interne Verzögerungszeiten haben, sollten kritische Signale (Signale, deren Senken kleine required times haben) durch möglichst wenige oder keine Treiber geleitet werden, während unkritische Signale über mehrere laufen können.

```
proc buffer_node (g,g_I,FO+,FO-)
begin
  R0 := evaluate_repower (g,g_I,FO+,FO-);
  if R0 > TR then return ; fi;
  (R1,E+,E-) = evaluate_trans1 (g,g_I,FO+,FO-);
  if R1 > R0
    then
          create_configuration_trans1 (g,g_I,E+,E-);
          if (R1 < TR)
            then if recursion_will_help () then buffer_node(B,b,L-,L+,TR_B); fi;
                  buffer_node(g,g_I,E+ ∪ {B},E-,TR);
          fi
     else
          (R2,x,y) = evaluate_trans2 (g,g_I,FO+,FO-);
          if R2 > R0
            then create_configuration_trans2(g,g_I,k,l);
                  if R2 < TR then buffer_node(g,B,∪_y g_I,∪_x b,TR); fi
          fi
  fi
end
```

Die Routine buffer_node kann drei verschiedene Strategien ausführen: repower, trans1 und trans2. Diese drei Veränderungen der Schaltung zum Lösen des Fanout–Problems werden jeweils nur dann durchgeführt, wenn sie die Signallaufzeiten verbessern. Das Maß hierzu sind die erzielten required times R_0, R_1 und R_2, die später genauer erläutert werden. buffer_node überprüft nach jeder Anwendung einer Strategie, ob die Zeitforderungen nun erfüllt sind. Ist dies der Fall (d.h. das Fanout–Problem wurde nach den vorgegebenen Zeitforderungen gelöst), dann wird die Fanout–Korrektur beendet, um die Kosten nicht unnötig zu erhöhen. Im folgenden werden die noch unbekannten Funktionen und Variablen der Routine buffer_node erklärt.

repower: Die am besten geeigneten Ersatzbausteine

Wie bereits erwähnt, haben Technologien im allgemeinen für Bausteine gleicher logischer Funktionalität mehrere Versionen, die sich in den Treibereigenschaften am Ausgang aber auch in den Kosten und Verzögerungen unterscheiden. Die repower–Transformation ersetzt die Bausteine n und n_I durch die am besten passenden (in Bezug auf die Verzögerungszeit) Ersatzbausteine g und g_I der verfügbaren äquivalenten Bausteine der Technologie (vgl. Abbildung 3.3). Die required time R_0 ist ein Maß für die Güte dieser Ersetzungsstrategie.

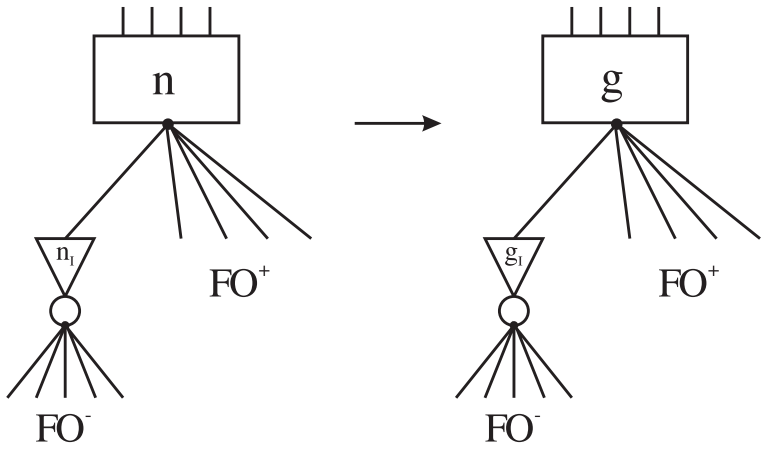

Abbildung 3.3: Die Transformation Repower

Der Wert R_0 wird von repower maximiert. Um R_0 zu berechnen, seien folgende Konventionen gegeben:

$$C_{FO^-} = \sum_{c_i \in FO^-} c_i \qquad C_{FO^+} = \sum_{c_i \in FO^+} c_i$$

f ist der Vorgänger von g auf dem kritischsten Pfad, der durch g verläuft. R_0 ist die required time am Eingang von g, der von f getrieben wird. Formel 3.7 berechnet den

Wert von R_0:

$$\begin{aligned} r_{g_I} &= r_1^- - (i_I^g + e^{g_I} \cdot C_{FO^-}) \\ R_0 &= \min\{r_1^+, r_{g_I}\} - (i^g + e^g \cdot (c^{g_I} + C_{FO^+})) - e^f \cdot c^g \end{aligned} \tag{3.7}$$

Diese Formel kann man - mit Ausnahme des Korrekturterms $e^f \cdot c^g$ - aus den Formeln 3.1, 3.3 und 3.4 gewinnen. Der Korrekturterm $e^f \cdot c^g$ wurde aus folgendem Grund eingeführt: Die Kapazität des Eingangs, der von f getrieben wird, kann je nach Ersatzbaustein g unterschiedlich sein und hat ebenfalls Einfluß auf die Signallaufzeit des kritischsten Pfades durch g. Sie verursacht eine Verzögerung von $e^f \cdot c^g$, die beim Vergleich verschiedener Ersatzbausteine berücksichtigt werden muß.

Um R_0 auch mit TR vergleichen zu können, ersetzt man die Formel 3.6 durch Formel 3.8, die den Korrekturterm (des Ursprungsbausteins) enthält:

$$TR = r_g + (r_i - r_j) - e^f \cdot c^n \tag{3.8}$$

trans1: Die erste Baumstrategie

Im folgenden wird davon ausgegangen, daß FO^+ und FO^- wenigstens zwei Elemente enthalten (für FO^+ ist die Lösung sonst trivial; die Fälle, bei denen FO^- leer beziehungsweise einelementig ist, sind einfache Sonderfälle). Die Transformation trans1 versucht, kritische und unkritische Signale voneinander zu trennen und getrennt zu puffern. Dazu zerlegt sie die Mengen FO^+ beziehungsweise FO^- nach folgender Vorschrift in je zwei disjunkte und überdeckende Teilmengen:

$$\begin{aligned} FO^+ = E^+ \cup L^+ \quad E^+ = \{r_1^+, r_2^+, \ldots r_s^+\} \quad L^+ = \{r_{s+1}^+, \ldots r_k^+\} \text{ mit } 1 \leq s < k \\ FO^- = E^- \cup L^- \quad E^- = \{r_1^-, r_2^-, \ldots r_t^-\} \quad L^- = \{r_{t+1}^-, \ldots r_l^-\} \text{ mit } 1 \leq t < l \end{aligned}$$

E steht für *early*, also kleine (und damit kritische) required times, während L die späten (*late*), also weniger kritischen Signale repräsentiert. Innerhalb der Mengen L^+ und L^- werden die Indizes transformiert: von den Indizes in L^+ wird s abgezogen, von den Indizes in L^- wird t abgezogen, so daß die jeweils kleinste required time den Index 1 hat. C_{E^+}, C_{L^+}, C_{E^-} und C_{L^-} sind Summenkapazitäten und werden analog zu C_{FO^+} und C_{FO^-} definiert. Die Signale der kritischsten Zeiten sollten durch möglichst wenige Treiber geleitet werden, da die internen Verzögerungszeiten der Treiber die Signallaufzeiten beeinträchtigen. Andererseits gilt: Je geringer der Ausgangsgrad am Ausgang eines Bausteins ist, umso geringer ist die Kapazität an diesem Ausgang, was wiederum eine kleinere externe Verzögerung bedeutet. Um einen „Interessensausgleich" dieser beiden Auswirkungen verschiedener Aufteilungen herzustellen, werden sämtliche Aufteilungen von FO^+ und FO^-, sowie sämtliche Kombinationen äquivalenter Bausteine von g_I, b und B für alle Aufteilungen durchprobiert (vergleiche Abbildung 3.4). Die Formel 3.9 ist die Berechnungsvorschrift

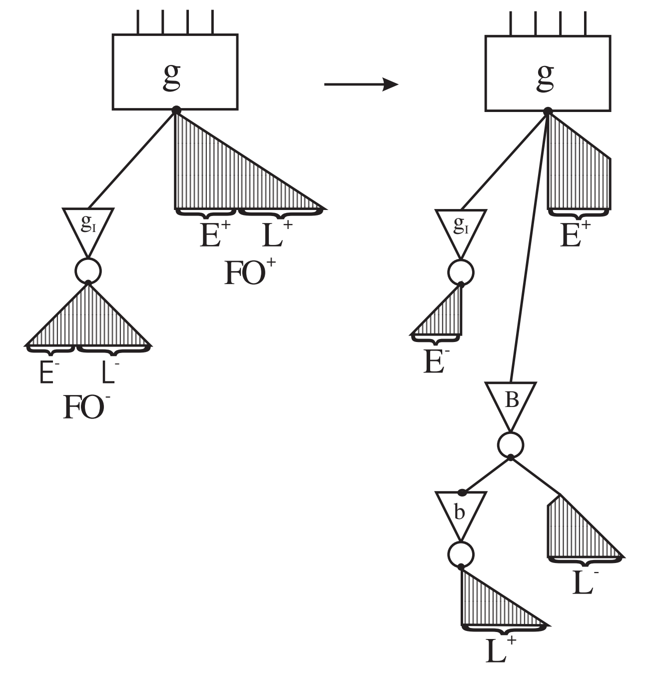

Abbildung 3.4: Die Transformation trans1

für den Wert R_1, also die required time am von f getriebenen Eingang von g.

$$
\begin{aligned}
r_b &= r_1^{L^+} - (i^b + e^b \cdot C_{L^+}) \\
r_B &= \min\{r_b, r_1^{L^-}\} - (i^B + e^B \cdot (c_b + C_{L^-})) \\
r_{g_I} &= r_1^- - (i_I^g + e^{g_I} \cdot C_{E^-}) \\
R_1 &= \min\{r_1^+, r_B, r_{g_I}\} - (i^g + e^g \cdot (c^B + c^{g_I} + C_{E^+})) - e^f \cdot c^g \qquad (3.9)
\end{aligned}
$$

Im Algorithmus wird das Maximum der R_1–Werte sämtlicher erzeugter Konfigurationen genommen. Falls das maximale R_1 die Zeitforderung TR erfüllt, wird die hierzu entsprechende Transformation ausgeführt und buffer_node wird verlassen. Falls das maximale R_1 sie nicht erfüllt, jedoch ein besseres Resultat liefert, als der durch repower erzielte Wert, dann wird trans1 ebenfalls ausgeführt, jedoch wird dann mittels zweier rekursiver Aufrufe von buffer_node versucht, eine weitere Verbesserung zu erzielen. Der erste rekursive Aufruf soll das Fanout–Problem für B und b lösen, der Parameter Target Required Time von buffer_node hat hier den Wert $TR_B = r_B + (r_B - \min\{r_1^+, r_{g_I}\}) - e^g \cdot c^B$. Falls $r_B > \min\{r_1^+, r_{g_I}\}$ ist, dann hat eine Verbesserung von r_B keine Auswirkung auf die required time von g, da der kritischste Pfad über das erste Element von FO^+ oder g_I läuft. In diesem Fall verhindert *recursion_will_help* den ersten rekursiven Aufruf.
Der zweite rekursive Aufruf wird auf die Knoten g und g_I angewendet.

trans2: Die balancierte Baumstrategie

Falls keine Verbesserung mittels trans1 möglich ist, der Ausgangsgrad von g jedoch so hoch ist, daß ein Treiberbaum die Zeiten verbessert, dann befinden sich die required times innerhalb eines kleinen Intervalles, d.h. r_1^+ ist nur wenig kleiner als r_k^+, r_1^- ist nur wenig kleiner als r_l^-. Ein balancierter Baum, wie in trans2 gebildet und in Abbildung 3.5 gezeigt, ist hier das Mittel der Wahl. Für eine x–Wege balancierte Aufteilung von FO^-

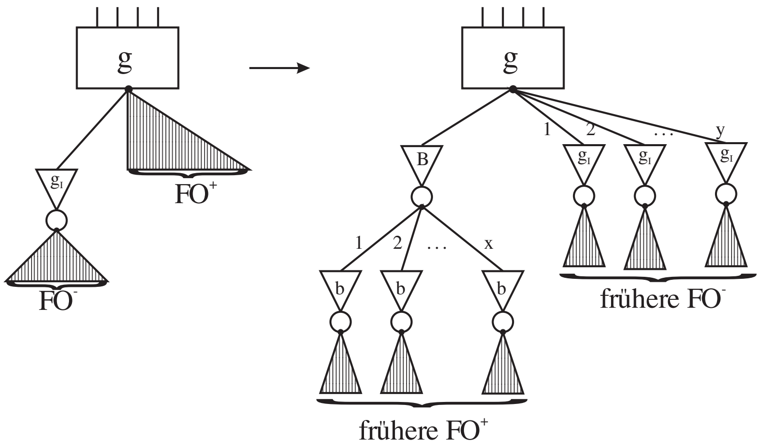

Abbildung 3.5: Die Transformation trans2

und eine y–Wege Aufteilung von FO^- wird die required time R_2 am Eingang von g wie folgt berechnet:

$$\begin{aligned} r_{b_1} &= r_1^+ - (i^b + e^b \cdot C^1_{FO+}) \\ r_B &= r_{b_1} - (i^B + e^B \cdot x \cdot c^b) \\ r_{g_{I1}} &= r_1^- - (i^{g_I} + e^{g_I} \cdot C^1_{FO-}) \\ R_2 &= \min\{r_B, r_{g_{I1}}\} - (i^g + e^g \cdot (c^B + y \cdot c^{g_I})) - e^f \cdot c^g \qquad (3.10) \end{aligned}$$

C^1_{FO+} ist die Gesamtkapazität der vom ersten Inverter b (hier b_1 genannt) getriebenen Eingänge, C^1_{FO-} die Kapazität der Eingänge, die vom ersten Inverter g_I (hier g_{I1} genannt) getrieben werden. Analog zu R_1 bezeichnet R_2 das Maximum über alle Aufteilungen (x,y) (mit $2 \le x \le \lceil \frac{k}{2} \rceil$ und $2 \le y \le \lceil \frac{l}{2} \rceil$) und kompatiblen Bausteine b, B und g_I.

Falls R_2 den Wert TR noch nicht erreicht hat, ruft man buffer_node in der in Abbildung 3.6 gezeigten Weise rekursiv auf.

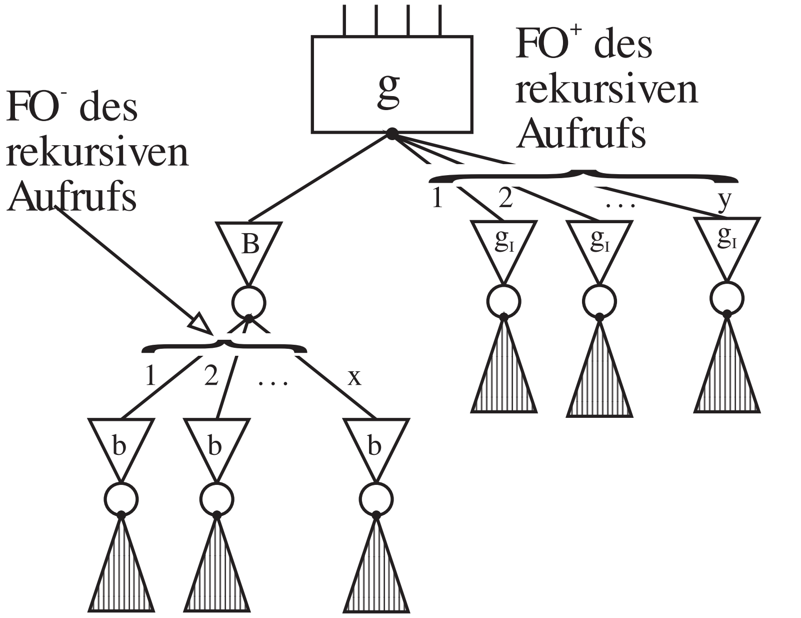

Abbildung 3.6: Der rekursive Aufruf für trans2

3.6 Analyse und Resultate

Die Laufzeit

Die Laufzeit des Algorithmus hängt unter anderem von Parametern ab, deren Werte erst bei Ausführung bekannt sind und für die nicht ohne weiteres Annahmen gemacht werden können (weiter unten gehen wir näher auf diese Problematik ein). Die Anzahl der Durchläufe des Rumpfes der **while**–Schleife in buffer_network ist beispielsweise schwierig bestimmbar: Die Werte der arrival times und required times an den primären Anschlüssen (und ihre Verteilung), die Technologie und der Graph der Schaltung bestimmen die Anzahl der Bausteine mit negativem Schlupf. Je nachdem wie viele und welche Bausteine es mit negativem Schlupf gibt und welchen Wert ϵ hat, gibt es mehr oder weniger Durchläufe durch die **while**–Schleife. Die folgende worst–case Laufzeitanalyse beschränkt sich deswegen auf den Rumpf dieser Schleife:
Es sei der Graph der Schaltung mit $G = (V, E)$ gegeben. M bezeichne die Mächtigkeit der größten Menge funktional äquivalenter Bausteine.

- delay_trace: Die topologische Sortierung benötigt die Zeit $O(|V| + |E|)$. Für die Berechnung der arrival time, der required time und des Schlupfes aller Knoten gilt die gleiche Abschätzung, da alle Knoten durchlaufen und zur Berechnung der Summenkapazitäten alle Kanten betrachtet werden müssen.

- Generiere ein ϵ–Netzwerk: Um das ϵ–Netzwerk zu erzeugen, muß jeder Knoten einmal besucht werden: $O(|V|)$. Im folgenden wird die worst–case Annahme gemacht, daß das ϵ–Netzwerk aus $|V|$ Knoten besteht.

- Finden der Fanout–Probleme: Die beiden verwendeten Routinen weisen den Knoten des ϵ–Netzwerks in der Zeit $O(|V|)$ Gewichte zu.

  - mincut: Nach [Law76] kann man die minimale cutset nach Durchlauf eines Max–Flow–Algorithmus erhalten. Die Laufzeit $O(|V^3|)$ von Max–Flow dominiert hierbei die Laufzeit von mincut. In [GT88] wird ein Max–Flow–Algorithmus von Goldberg und Tarjan vorgestellt, dessen Laufzeit in der gleichen Klasse $O(|V^3|)$ liegt. Der neue mincut–Algorithmus von Stoer und Wagner in [SW94] hat eine Laufzeit $O(|V| \cdot |E| + |V^2| \log |V|)$.
  - extract: Das Sortieren der Knoten nach ihren Gewichten benötigt die Zeit $O(|V| \cdot \log |V|)$, das Streichen der transitiven Vorgänger und Nachfolger hat die Zeitkomplexität $O(|V| + |E|)$.

  Im folgenden wird die worst–case Annahme gemacht, daß die Anzahl der gefundenen Stellen $O(|V|)$ ist.

- buffer_node: Für die Strategien zum Lösen des Fanout–Problems lassen sich folgende Abschätzungen machen:

  - repower: Für n werden logisch äquivalente Ersatzbausteine gesucht, für n_I werden alle Inverter durchprobiert: $O(M^2)$
  - trans1: Sei $|E^+| = s$ und $|E^-| = t$. Die rekursive Gleichung

    $$T_1(k,l) = O(M^3 \cdot k \cdot l) + T_1(l-s, k-t) + T_1(t,s)$$

    beschreibt die Komplexität von trans1. Sie setzt sich zusammen aus dem „Durchprobieren“ äquivalenter Bausteine (Term M^3), der verschiedenen Aufteilungen von FO^+ und FO^- (Term $k \cdot l$) und der Komplexität der rekursiven Aufrufe: ($T_1(l-s, k-t)$ für den Aufruf auf B, $T_1(t,s)$ für den Aufruf auf g). Anmerkung: Beim Aufstellen dieser Gleichung wurde angenommen, daß sämtliche rekursiven Aufrufe aus trans1 erfolgen. Die Lösung dieser Gleichung ist $T_1(k,l) = O(M^3 \cdot k^2 \cdot l^2)$. Da M^3 eine, wenn auch eventuell sehr große Konstante ist, schätzt man die Komplexität von trans1 nach der Formel 3.11 ab:

    $$T_1(k,l) = O(k^2 \cdot l^2) \tag{3.11}$$

  - trans2: Wenn x die Anzahl der Inverter b und y die Anzahl der Inverter g_I ist, dann gibt die rekursive Gleichung

    $$T_2(k,l) = O(M^3 \cdot k \cdot l) + T_2(x,y)$$

die Komplexität von trans2 an. Anmerkung: Beim Aufstellen dieser Gleichung wurde angenommen, daß sämtliche rekursiven Aufrufe aus trans2 erfolgen. Auch die Lösung dieser Gleichung $T_2(k,l) = O(M^3 \cdot k \cdot l \cdot \log(k \cdot l))$ kann mit der gleichen Begründung wie bei trans1 vereinfacht werden:

$$T_1(k,l) = O(k \cdot l \cdot \log(k \cdot l)) \tag{3.12}$$

Vergleicht man diese Lösungen, so dominieren die Transformationen die repower–Strategie. Weiter sieht man, daß es komplexer ist, wenn nur trans1 rekursiv auf einen Knoten aufgerufen wird, als wenn nur trans2 rekursiv auf einen Knoten aufgerufen wird (vgl. die Formeln 3.11 und 3.12). Weiter gilt: Eine Rekursion, bei der nur trans1 aufgerufen wird, ist komplexer als jede „Mischrekursion", d.h. Rekursionen, bei denen sowohl trans1 als auch trans2 aufgerufen wird. Insgesamt gilt: Die Formel 3.11 eignet sich als worst–case Laufzeit für buffer_node.

Zeitkritisch innerhalb von buffer_network sind mincut und buffer_node, wobei letztere bezüglich der O–Notation komplexer sein kann: Die Zeitkomplexität von mincut kann von buffer_node beispielsweise durch folgende konstruierte Schaltung $\mathcal{B}$ überschritten werden: Der Graph von $\mathcal{B}$ habe N Knoten. Auf $O(1)$ Knoten wird buffer_node angewendet. Der Ausgangsgrad dieser Knoten (und ihrer n_I–Knoten) sei $O(N)$. Alle rekursiven Aufrufe seien trans1–Aufrufe. Dann lautet die worst–case Abschätzung für buffer_node auf $\mathcal{B}$: $O(N^4)$.

Folgende Parameter können bei einer Laufzeitabschätzung nicht unberücksichtigt bleiben:

- Je nach spezifizierten arrival times und required times an den primären Ein- und Ausgängen kann die selbe Schaltung zu unterschiedlichen Laufzeiten des Algorithmus führen. Wenn die spezifizierten Zeiten viele und aufwendige Schaltungsmodifikationen implizieren, dann führt dies zu einer hohen Laufzeit. Solche extremen Beispiele lassen sich konstruieren, wie eben gesehen.
- Der Wert von ϵ. Hier gibt es zwei gegenläufige Tendenzen, die nahelegen, daß ϵ nur experimentell bestimmt werden kann:
  - Ein hoher Wert von ϵ bedeutet, daß viele Bausteine in das ϵ–Netzwerk aufgenommen werden. Dadurch verringert sich eventuell die Anzahl der Durchläufe der **while**–Schleife von buffer_network, gleichzeitig erhöht sich jedoch auch die Anzahl der Knoten, auf die buffer_node angewandt wird.
  - Ein niedriger Wert von ϵ bedeutet entsprechend: Das ϵ–Netzwerk besteht nur aus wenigen Knoten, und somit kann man erwarten, daß die Anzahl der Aufrufe von buffer_node gering ist. Dafür wird jedoch vermutlich die Anzahl der Durchläufe der **while**–Schleife von buffer_network hoch sein.

Experimentelle Ergebnisse, die für diese Annahmen sprechen, werden weiter unten im Abschnitt „Resultate" präsentiert.

Die oben gemachte Analyse stellt also lediglich einen Anhaltspunkt für die Laufzeit des Algorithmus dar und kann Messungen nicht ersetzen.

Der Platzbedarf

Um mit möglichst wenig Platz auszukommen, sollte man eine Datenstruktur wählen, die von der geometrischen Verdrahtung abstrahiert. Dann beläuft sich der Platzbedarf auf $O(|V| + |E|)$.

Resultate

CADIC ermöglicht einen top–down Entwurf von Schaltungen (d.h. schrittweise Verfeinerung). Mit Hilfe des zugrundeliegenden Netzkalküls ist es sogar möglich, parametrisierte Schaltungen einzugeben: Der Entwerfer spezifiziert eine rekursive Beschreibung inklusive einer oder mehrerer „Rekursionsbremsen“ und kann die Ausprägung für konkrete Parameterwerte vom System generieren lassen. Da man bei Schaltungen, die diese hierarchischen Entwurfsprinzipien in einem hohen Grade ausnutzen, den Überblick über die Leitungsführung und damit auch über das Fanout–Problem leicht verlieren kann, bietet sich ein Fanout–Algorithmus gerade für solche Schaltungen an. Die Tabellen 3.1 und 3.2 enthalten flache Schaltungen mit hohem Ausgangsgrad, die als parametrisierte Schaltungen entworfen wurden. Die IMS-Technologie (= Institut für Mikroelektronik Stuttgart), die den Tabellen zugrundeliegt, hat für jede logische Grundfunktion lediglich einen Baustein, Repower wird also nicht aufgerufen.
Die erste Spalte enthält den Namen der Schaltung und den Parameterwert. Der Parameterwert entspricht beim Multiplizierer (mul) und beim Dividierer (div) der Bitbreite, für den Conditional Sum Adder (ad), den Carry Lookahead Adder (cla) und den Dekodierer (FDec) ist die Bitbreite eine Zweierpotenz des Parameterwertes. Die zweite Spalte gibt den größten Ausgangsgrad innerhalb der Schaltung wieder. Die drei Spalten, die mit Zeit überschrieben sind, enthalten die Dauer der längsten Signallaufzeit. Diese kann man angeben, da für alle Schaltungen die arrival times an den primären Eingängen auf 0 gesetzt wurden. Die required times an den primären Ausgängen bekamen einen Wert, der durch keine Modifikation erreichbar ist, dadurch läuft der Algorithmus so lange, bis er die kritischen Signallaufzeiten nicht mehr weiter verbessern kann. Wir verwendeten ein einfaches Kostenmaß, nämlich die Anzahl der Bausteine. Die Zeit- und Kosten–Spalten ermöglichen den Vergleich zwischen den beiden Routinen mincut und extract. Letztere gewinnt im Signallaufzeitvergleich fast immer (Ausnahmen: beim cla sind sie gleich, beim Dividierer ist extract für kleine Parameterwerte schlechter). Immer, wenn eine der beiden Routinen bezüglich der Signallaufzeit besser als die andere abgeschnitten hat, hat sie mehr Treiber eingesetzt, also höhere Kosten verursacht. Dies entspricht dem Prinzip: Treiber werden nur dann eingesetzt, wenn sie die Signallaufzeit verbessern. In den letzten beiden Spalten stehen die Laufzeiten, sie wurden auf einer Sparc 10 gemessen. Werden die Transformationen häufig angewendet, dann werden viele Treiber eingesetzt. Man sieht dies in einem Zusammenhang zwischen der Laufzeit und der Zunahme der Kosten. Für ϵ wurden kleine Werte gewählt, um vor allem Knoten in der Nähe kritischer Pfade zu betrachten. In Tabelle 3.1 hat ϵ den Wert 0.5, in Tabelle 3.2 den Wert 0.01.

Name	Max Fan-out	Zeit			Kosten			Laufzeit	
		vor-her	Min-cut	Ex-tract	vor-her	Min-cut	Ex-tract	Min-cut	Ex-tract
ad[7]	65	45.2	29.5	26.4	1662	1694	1720	4.9	4.6
ad[8]	129	84.0	51.7	27.3	3582	3614	4033	11.8	29.1
ad[9]	257	161.1	95.3	34.1	7678	7716	8384	25.1	114.8
cla[7]	4	47.4	46.4	46.4	3198	3326	3326	4.5	5.4
cla[8]	4	53.2	52.3	52.3	7166	7422	7422	12.6	19.0
div[3]	22	124.7	118.0	115.6	204	218	234	0.4	0.5
div[4]	22	172.5	166.8	166.8	332	340	340	0.5	0.5
div[5]	22	223.1	205.3	220.3	490	514	494	1.4	0.5
div[6]	22	276.6	252.0	268.1	678	692	686	1.5	1.2
div[7]	22	333.0	294.4	321.6	896	916	904	2.9	1.4
div[8]	22	392.2	337.4	338.6	1144	1184	1180	5.0	5.1
div[16]	30	968.4	664.8	677.7	4208	4302	4279	40.4	35.2
div[24]	46	1727.0	1037.8	1016.0	9192	9332	9343	104.5	147.7
div[32]	62	2656.8	1365.6	1336.8	16096	16364	16421	427.3	674.9
FDec[8]	16	21.2	21.2	18.7	312	312	472	0.6	4.2
FDec[9]	32	30.7	30.7	21.9	593	593	833	1.4	10.6
FDec[10]	32	34.4	34.4	24.4	1130	1130	1578	2.7	13.9
mul[16]	2	68.1	68.1	68.1	2752	2752	2752	2.8	3.7
mul[32]	2	89.7	89.7	89.7	11648	11648	11648	15.5	19.7

Tabelle 3.1: Der Algorithmus auf CADIC–Schaltungen mit $\epsilon = 0.5$

Welchen Einfluß die Wahl von ϵ auf die Anzahl der Schleifendurchläufe und auf die Anzahl der Bausteine im ϵ–Netzwerk hat, wurde anhand von Beispielschaltungen gemessen. Die Ergebnisse, die in den Tabellen am Ende des Kapitels (siehe Seiten 52–54) wiedergegeben sind, legen nahe, daß die oben gemachten Annahmen über diesen Parameter wahr sind. Die zweite Spalte dieser Tabellen gibt an, ob die extract (e) oder die mincut (m) Strategie verwendet wurde. In der vierten Spalte befindet sich die Laufzeit des Algorithmus unter den gegebenen Parametern (gemessen auf einer Sparc 10 mit Ausnahme des Dividierers div[32], seine Laufzeiten wurden auf einer Sparc 20 gemessen). Die Kosten nach der Veränderung stehen in der fünften Spalte. Die sechste Spalte enthält die maximale arrival time der primären Ausgänge, ebenfalls nach der Modifikation. In der siebten Spalte steht die mittlere Anzahl der Bausteine mit negativem Schlupf (es wurde über die Anzahl der Schleifendurchläufe gemittelt). Die mittlere Anzahl der Bausteine im ϵ–Netzwerk kann man in Spalte acht nachlesen.
Der Einfluß von ϵ auf die Anzahl der Schleifendurchläufe und auf die Anzahl der Bausteine im ϵ–Netzwerk fällt für die Beispielschaltungen qualitativ wie erwartet aus, jedoch hat der Parameter ϵ quantitativ keine so große Bedeutung.

Name	Max Fan-out	Zeit			Kosten			Laufzeit	
		vor-her	Min-cut	Ex-tract	vor-her	Min-cut	Ex-tract	Min-cut	Ex-tract
ad[7]	65	45.2	28.7	26.4	1662	1726	1720	11.6	4.8
ad[8]	129	84.0	49.5	27.3	3582	3648	4016	25.7	49.2
ad[9]	257	161.1	90.7	34.2	7678	7744	8366	57.7	221.8
cla[7]	4	47.4	46.4	46.4	3198	3265	3262	6.0	6.6
cla[8]	4	53.2	52.3	52.3	7166	7294	7294	20.8	18.7
div[3]	22	124.7	117.4	117.4	204	225	225	0.5	0.5
div[4]	22	172.5	166.8	166.8	332	340	340	0.5	0.5
div[5]	22	223.1	210.3	210.3	490	500	500	0.8	0.8
div[6]	22	276.6	253.9	253.9	678	692	692	1.4	1.4
div[7]	22	333.0	296.6	296.6	896	919	919	2.0	2.2
div[8]	22	392.2	338.6	338.6	1144	1180	1180	3.9	4.5
div[16]	30	968.4	677.7	677.7	4208	4279	4279	28.3	31.3
div[24]	46	1727.0	1039.0	1016.0	9192	9328	9341	105.0	133.1
div[32]	62	2656.8	1384.3	1354.2	16096	16342	16375	311.5	442.3
FDec[8]	16	21.2	21.2	18.7	312	312	472	0.5	2.2
FDec[9]	32	30.7	30.7	21.9	593	593	833	1.1	6.0
FDec[10]	32	34.4	34.4	24.4	1130	1130	1578	2.2	11.4
mul[16]	2	68.1	68.1	68.1	2752	2752	2752	2.8	3.5
mul[32]	2	89.7	89.7	89.7	11648	11648	11648	16.2	19.2

Tabelle 3.2: Der Algorithmus auf CADIC–Schaltungen mit $\epsilon = 0.01$

3.7 Vor- und Nachteile der Heuristik

Der Algorithmus von Singh und Sangiovanni-Vincentelli wurde für folgende Problemstellung konzipiert: Gegeben ist eine flache Schaltung in einer Technologie mit arrival times an den primären Eingängen und required times an den primären Ausgängen. Versuche, die Zeitforderungen zu erfüllen und dabei die hinzukommenden Kosten möglichst gering zu halten.
Der gefundene Algorithmus hat folgende Einschränkungen und Nachteile:

- Kosten werden nur indirekt berücksichtigt, indem nur dann Treiber eingesetzt werden, wenn dadurch die Signallaufzeit verbessert wird. Eine dem Fanout–Problem näher kommende Konzeption fordert für einen Fanout–Algorithmus die Einhaltung einer Kostenschranke (zusätzlich zu den Zeitforderungen).

- Er arbeitet nur auf Bausteinen mit einem Ausgang. Diese Einschränkung ist theoretischer Natur, damit Formulierungen und Beweise einfacher sind. Im CADIC–System wurde der Algorithmus ohne diese Einschränkung implementiert.

- Repower ist für viele Technologien nicht anwendbar, da sie nur einen Baustein für jede logische Grundfunktion haben.

- Puffer werden prinzipiell durch eine gerade Anzahl von Invertern realisiert. Diese Einschränkung ist nötig, wenn man die Rekursionen bei trans1 und trans2 ermöglichen will. Falls der Algorithmus endet, weil mittels seiner Strategien keine Verbesserung der Signallaufzeit mehr möglich ist, die Signallaufzeit jedoch immer noch zu hoch ist, dann könnte man noch folgende Strategie anwenden: Man sucht je zwei Inverter, die hintereinandergeschaltet sind, wobei der Ausgangsgrad des ersten Inverters 1 ist, d.h. an seinem Ausgang hängt nur der zweite Inverter. Diese beiden Inverter können durch einen Puffer ersetzt werden. Falls diese Strategie die Signallaufzeit verbessert, dann wird - wie bei repower - der am besten geeignete Puffer eingesetzt.

- Für den Einsatz der Techniken zur Verbesserung der Signallaufzeit wurden nur „qualitative Argumente“ angegeben.

- Die Zeitanalyse ist eingeschränkt: Die unterschiedlichen Verzögerungszeiten durch verschiedene Flankenwechsel werden im Modell der Verzögerungszeiten der Bausteine nicht berücksichtigt.

Dem stehen folgende Vorteile gegenüber:

- Die Laufzeit: Das kontinuierliche Problem wird in ein diskretes Problem mit einer Näherungsformel für die Verzögerung umgewandelt.
  Unnötige rekursive Aufrufe werden verhindert.
  Die Beispielschaltungen legen nahe, daß der Algorithmus auch für große Schaltungen relativ kleine Laufzeiten hat.

- Repower ist ein effizientes Werkzeug, das vermutlich in vielen Fällen genügt (unter der Voraussetzung, daß es in der Technologie Varianten von logisch äquivalenten Bausteinen gibt), um die erwünschte Beschleunigung der Signallaufzeiten zu erhalten.

- In den Treiberbäumen von trans1 werden kritische und unkritische Signale durch unterschiedlich viele Treiberstufen geleitet.

- Durch das ϵ–Netzwerk reduziert sich die zu betrachtende Schaltungsgröße auf einen Bruchteil der ursprünglichen Größe.

Es folgen die Tabellen, die einen Eindruck über den Einfluß von ϵ geben:

Name	e/ m	ϵ	Lauf- zeit	Kosten nachh.	max. arr. ti. an PO	mittl. # B. m. Slack<0	mittl. # B. im ϵ–N.	# Durchl. Schleife
ad[9]	e	1.0	115.5	8456	34.1	3355	735	30
ad[9]	e	0.5	114.8	8384	34.1	3340	465	39
ad[9]	e	0.2	141.5	8368	34.2	3356	435	49
ad[9]	e	0.1	194.5	8366	34.2	3356	386	76
ad[9]	e	0.05	213.6	8366	34.2	3363	362	81
ad[9]	e	0.02	221	8366	34.2	3367	351	85
ad[9]	e	0.01	221.8	8366	34.2	3365	347	86
ad[9]	e	0.005	222.4	8366	34.2	3365	347	86
ad[9]	m	1.0	24.2	7710	95.3	3378	694	6
ad[9]	m	0.5	25.1	7716	95.3	3407	651	7
ad[9]	m	0.2	26.4	7722	95.0	3426	648	8
ad[9]	m	0.1	32.5	7724	95.0	3444	458	12
ad[9]	m	0.05	32.3	7726	95.0	3444	458	12
ad[9]	m	0.02	51.3	7744	90.7	3519	407	22
ad[9]	m	0.01	57.7	7744	90.7	3526	358	25
ad[9]	m	0.005	57.7	7744	90.7	3526	358	25

Name	e/ m	ϵ	Lauf- zeit	Kosten nachh.	max. arr. ti. an PO	mittl. # B. m. Slack<0	mittl. # B. im ϵ–N.	# Durchl. Schleife
cla[8]	e	1.0	28.1	7422	52.3	7230	1705	2
cla[8]	e	0.5	19.0	7422	52.3	7210	787	3
cla[8]	e	0.2	14.2	7294	52.3	7210	423	3
cla[8]	e	0.1	14.1	7294	52.3	7210	398	3
cla[8]	e	0.05	14.8	7294	52.3	7215	283	4
cla[8]	e	0.02	16.4	7294	52.3	7208	222	5
cla[8]	e	0.01	18.7	7294	52.3	7202	156	7
cla[8]	e	0.005	17.0	7294	52.3	7202	156	7
cla[8]	m	1.0	12.7	7422	52.3	7230	1705	2
cla[8]	m	0.5	12.6	7422	52.3	7210	787	3
cla[8]	m	0.2	12.1	7294	52.3	7210	423	3
cla[8]	m	0.1	12.0	7294	52.3	7210	398	3
cla[8]	m	0.05	13.3	7294	52.3	7215	283	4
cla[8]	m	0.02	15.1	7294	52.3	7208	222	5
cla[8]	m	0.01	20.8	7294	52.3	7202	156	7
cla[8]	m	0.005	20.9	7294	52.3	7202	156	7

Name	e/ m	ϵ	Lauf- zeit	Kosten nachh.	max. arr. ti. an PO	mittl. # B. m. Slack<0	mittl. # B. im ϵ–N.	# Durchl. Schleife
div[32]	e	1.0	430.4	16400	1365	16165	683	123
div[32]	e	0.5	370.5	16392	1367	16163	592	121
div[32]	e	0.2	341.4	16386	1369	16161	503	119
div[32]	e	0.1	300.5	16338	1390	16153	482	107
div[32]	e	0.05	299.2	16338	1390	16153	480	107
div[32]	e	0.02	313.5	16338	1390	16153	480	107
div[32]	e	0.01	320.2	16338	1390	16153	480	107
div[32]	e	0.005	333.5	16338	1390	16153	480	107
div[32]	m	1.0	388.9	16410	1372	16169	756	134
div[32]	m	0.5	297.9	16330	1389	16151	565	103
div[32]	m	0.2	302.3	16340	1326	16153	499	108
div[32]	m	0.1	280.1	16338	1390	16153	482	107
div[32]	m	0.05	265.6	16338	1390	16153	480	107
div[32]	m	0.02	263.7	16338	1390	16153	480	107
div[32]	m	0.01	265.5	16338	1390	16153	480	107
div[32]	m	0.005	266.4	16338	1390	16153	480	107

Name	e/ m	ϵ	Lauf- zeit	Kosten nachh.	max. arr. ti. an PO	mittl. # B. m. Slack<0	mittl. # B. im ϵ–N.	# Durchl. Schleife
FDec[10]	e	1.0	13.9	1578	24.4	1449	725	4
FDec[10]	e	0.5	13.9	1578	24.4	1463	514	6
FDec[10]	e	0.2	13.9	1578	24.4	1451	503	7
FDec[10]	e	0.1	11.5	1578	24.4	1451	453	7
FDec[10]	e	0.05	12.2	1578	24.4	1477	352	9
FDec[10]	e	0.02	11.6	1578	24.4	1464	218	13
FDec[10]	e	0.01	11.4	1578	24.4	1464	218	13
FDec[10]	e	0.005	11.4	1578	24.4	1464	218	13
FDec[10]	m	1.0	2.3	1130	34.4	1130	1089	1
FDec[10]	m	0.5	2.7	1130	34.4	1130	954	1
FDec[10]	m	0.2	2.2	1130	34.4	1130	803	1
FDec[10]	m	0.1	2.2	1130	34.4	1130	803	1
FDec[10]	m	0.05	2.2	1130	34.4	1130	803	1
FDec[10]	m	0.02	2.2	1130	34.4	1130	535	1
FDec[10]	m	0.01	2.2	1130	34.4	1130	535	1
FDec[10]	m	0.005	2.2	1130	34.4	1130	535	1

Name	e/ m	ϵ	Lauf- zeit	Kosten nachh.	max. arr. ti. an PO	mittl. # B. m. Slack<0	mittl. # B. im ϵ–N.	# Durchl. Schleife
mul[32]	e	1.0	19.6	11648	89.7	11648	1482	1
mul[32]	e	0.5	19.7	11648	89.7	11648	1456	1
mul[32]	e	0.2	19.5	11648	89.7	11648	1400	1
mul[32]	e	0.1	20.3	11648	89.7	11648	1400	1
mul[32]	e	0.05	19.5	11648	89.7	11648	1400	1
mul[32]	e	0.02	19.4	11648	89.7	11648	1400	1
mul[32]	e	0.01	19.2	11648	89.7	11648	1400	1
mul[32]	e	0.005	19.0	11648	89.7	11648	1400	1
mul[32]	m	1.0	15.0	11648	89.7	11648	1482	1
mul[32]	m	0.5	15.5	11648	89.7	11648	1456	1
mul[32]	m	0.2	16.2	11648	89.7	11648	1400	1
mul[32]	m	0.1	16.5	11648	89.7	11648	1400	1
mul[32]	m	0.05	16.4	11648	89.7	11648	1400	1
mul[32]	m	0.02	16.2	11648	89.7	11648	1400	1
mul[32]	m	0.01	16.2	11648	89.7	11648	1400	1
mul[32]	m	0.005	16.6	11648	89.7	11648	1400	1

Kapitel 4

Das Fanout–Problem und die Hierarchie

4.1 Definitionen

4.1.1 Definition einer hierarchischen Schaltungsbeschreibung

Im folgenden sei ein nicht näher spezifizierter Katalog von Basiszellen gegeben. Der top-down Entwurf beziehungsweise das hierarchische Vorgehen beim Schaltungsentwurf wird in [KMO89] wie folgt erläutert: „Ist ein Schaltkreis sehr groß, d.h. besteht er aus zehn- oder gar hunderttausend Gattern, so wird man ihn kaum in seiner vollen Pracht fehlerfrei zeichnen oder durch einen gigantischen Ausdruck spezifizieren können. Hier braucht man Methoden, kürzere Beschreibungen anzugeben, aus denen möglichst leicht der Schaltkreis selbst gewonnen werden kann. Eine Methode, die Übersicht zu wahren, leitet sich daraus ab, daß man sich zunächst einen Schaltkreis aus sehr mächtigen, komplexen Komponenten überlegt und diese Komponenten für sich entwickelt, wobei dort dieselbe Strategie angewendet wird, bis man die Teilaufgabe sehr einfach direkt über Grundelementen lösen kann.“

Diese schrittweise Verfeinerung der Komponenten kann in Anlehnung an [Bur94] wie folgt repräsentiert werden:

Definition 4.1:
Eine gefaltete Darstellung einer Schaltung ist ein zusammenhängender dag (directed acyclic graph = gerichteter azyklischer Graph) $G = (V, E)$.
Die Menge V der Knoten hat folgende Eigenschaften:

1. *Jede Teilschaltung S wird durch genau einen Knoten aus V repräsentiert.*

2. *Für jeden Typ B von Basiszellen des verwendeten Katalogs, der mindestens einen Repräsentanten in wenigstens einer Teilschaltung hat, gibt es genau einen Knoten im dag.*

3. *V besteht aus den in 1. und 2. eingeführten Knoten.*

4. *Es gibt eine Teilschaltung $r \in V$, die sämtliche primären Anschlüsse der Schaltung enthält. r kommt in keiner anderen Schaltung als Instanz vor.*

Die Knoten sollen mit denselben Namen bezeichnet werden wie die entsprechenden Teilschaltungen und Basiszellen.
Für die Menge der Kanten E gilt: Es sei $v \in V$ eine Teilschaltung oder eine Basiszelle. v komme wenigstens einmal als Instanz in der Teilschaltung t vor. Dann existiert genau eine Kante $t \longrightarrow v$. Die in dieser Weise erzeugten Kanten bilden die Menge E. Die Kanten aus E sind beschriftet: Eine natürliche Zahl gibt an, wie viele Verweise sie repräsentieren, d.h. wie oft die Instanz v als Makro in einer Teilschaltung t vorkommt.

Daraus folgt: r ist der einzige Knoten, der keine eingehenden Kanten hat. Die Knoten der Basiszellen und die Knoten von Teilschaltungen, die nur Verdrahtung enthalten, sind die einzigen Knoten, die keine ausgehenden Kanten haben.
Von einer Schaltung, die in Form der Definition 4.1 gegeben ist, läßt sich die Anzahl der Vorkommen von Teilschaltungen und Basiszellen in der entsprechenden flachen Schaltung berechnen: Es sei $b(e)$ die Beschriftung der Kante e. Mit $Path(r, v)$ bezeichnen wir die Menge der Pfade von r nach $v \in V$ innerhalb des dag, von einem Pfad p sei $E(p)$ die Menge der von ihm durchlaufenen Kanten. Die Anzahl a der Vorkommen der Teilschaltung $v \in V$ in der entsprechenden flachen Schaltung ist dann:

$$a := \sum_{p \in Path(R,v)} \prod_{e \in E(p)} b(e) \tag{4.1}$$

Definition 4.2:
Eine flache Darstellung einer Schaltung ist ein Baum $G = (V, E)$.
Für die Knoten gilt:

1. *Es gibt eine Teilschaltung r, die sämtliche primären Anschlüsse der Schaltung enthält und die in keiner anderen Teilschaltung vorkommt. Der Knoten $r \in V$ ist die Wurzel des Baumes.*

2. *Sei t eine Teilschaltung (mit $t \neq r$) oder eine Basiszelle. t wird in V durch so viele Knoten repräsentiert, wie t in der flachen Schaltung vorkommt (entsprechend der Formel 4.1).*

3. *V besteht aus den in 1. und 2. eingeführten Knoten.*

Wir bezeichnen die Elemente aus V mit indizierten Knoten: v_x.
Die Kanten werden in der Weise gebildet, daß jede Teilschaltung beziehungsweise Basiszelle (außer r) Ziel genau einer Kante ist. Dies ist möglich, da es nach 2. genau so viele Knoten gibt, wie Basiszellen und Teilschaltungen in der flachen Schaltung vorkommen: Die Teilschaltung oder Basiszelle v_j komme als Instanz in der Teilschaltung v_i vor. Diese Instanziierung wird durch genau eine Kante $v_i \longrightarrow v_j$ repräsentiert.

Die Blätter des Baumes sind Basiszellen der verwendeten Bibliothek oder Teilschaltungen, die nur Verdrahtung enthalten. Die flache Darstellung hat keine Kantenmarkierung, da jede Kante genau eine Instanziierung repräsentiert.

Die teilentfaltete Schaltungsdarstellung in Definition 4.4 ist eine hierarchische Schaltungsdarstellung, die „zwischen“ gefalteter und flacher Darstellung liegt (es gibt jedoch Sonderfälle, bei denen gefaltete Darstellungen nur in flache umgewandelt werden können). Falls die flache Darstellung durch sukzessives Teilentfalten entstanden ist, wird die Kantenmarkierung der resultierenden Darstellung gelöscht. Um eine teilentfaltete Darstellung zu erhalten, muß teilentfaltet werden.

Definition 4.3:
Die Operation einfache Entfaltung:
Gegeben sei eine hierarchische Schaltung, die durch einen gefalteten oder teilentfalteten dag C repräsentiert wird. C hat mindestens einen Knoten, der als Instanz mindestens zweimal vorkommt (d.h. die Beschriftung einer seiner eingehenden Kanten ist > 1, oder er ist Endknoten von wenigstens zwei Kanten). Jeder dieser Knoten ist Wurzel eines Baumes der Tiefe ≥ 0, den er zusammen mit seinen Nachfahren bildet. Sei x ein solcher Wurzelknoten.

- *Falls x genau eine eingehende Kante hat: Sei i die Kantenbeschriftung dieser Kante. Erzeuge j Kopien (mit $1 \leq j \leq i-1$) des Baumes, dessen Wurzel x ist und benenne ihre Knoten und Kanten eindeutig! Erzeuge j Kanten vom Vorgänger von x zu den neuen Wurzelknoten! Beschrifte die Kanten, die x und die Kopien von x als Ziel haben, so daß jede Kantenbeschriftung einen Wert ≥ 1 hat und daß die Summe der Kantenbeschriftungen (dieser $j+1$ Kanten) i ist!*

- *Falls x i eingehende Kanten (mit $i \geq 2$) hat: Jede der Quellen dieser Kanten repräsentiert eine Teilschaltung, in der x (eventuell mehrfach) vorkommt. Erzeuge i Kopien des Baumes, dessen Wurzel x ist und benenne ihre Knoten und Kanten eindeutig! Streiche den Ursprungsbaum mit Wurzel x! Die Ziele der i Kanten, die ursprünglich zu x führten, sind nun die Kopien der Wurzeln; jede Kante hat genau eine Kopie als Ziel, d.h. jede kopierte Wurzel repräsentiert die Instanzen einer Teilschaltung.*

Falls durch eine einfache Teilentfaltung eine flache Darstellung entsteht, werden die Kantenmarkierungen gelöscht.

Die konstruktive Definition 4.3 beschreibt ein Verfahren zur Entfaltung der hierarchischen Struktur. Dieses Verfahren wurde nicht unter Effizienzkriterien entwickelt, es dient in diesem Zusammenhang dem Verständnis der Operation.

Definition 4.4:
Eine teilentfaltete Darstellung einer Schaltung ist die Darstellung, die man erhält, wenn auf eine ursprünglich gefaltete Darstellung der Schaltung die Operation 4.3 einmal oder mehrmals hintereinander ausgeführt wird und die flache Darstellung noch nicht erreicht ist.

Die Darstellung nach Definition 4.1 ist die platzsparendste, es gibt jedoch Algorithmen, die die Hierarchie „aufweichen" (d.h. teilentfalten) müssen, um gute Resultate erzielen zu können. Hierzu werden durch eine Teilentfaltung (oder mehrere) und anschließende Modifkationen der identischen Teilschaltungen *Varianten* (siehe [KMO89]) gebildet. Diese sind für Algorithmen notwendig, die nur dann gute Resultate erzielen, wenn sie bestimmte Teilschaltungen in Abhängigkeit vom Kontext, in den sie eingesetzt werden, modifizieren können.

Ein Beispiel soll die verschiedenen Definitionen zur hierarchischen Schaltungsbeschreibung veranschaulichen: Abbildung 4.1 zeigt eine von CADIC generierte, hierarchische Beschreibung eines 2–Bit Conditional Sum–Adders (aus [KMO89]; vergleiche Abbildung 4.24 und

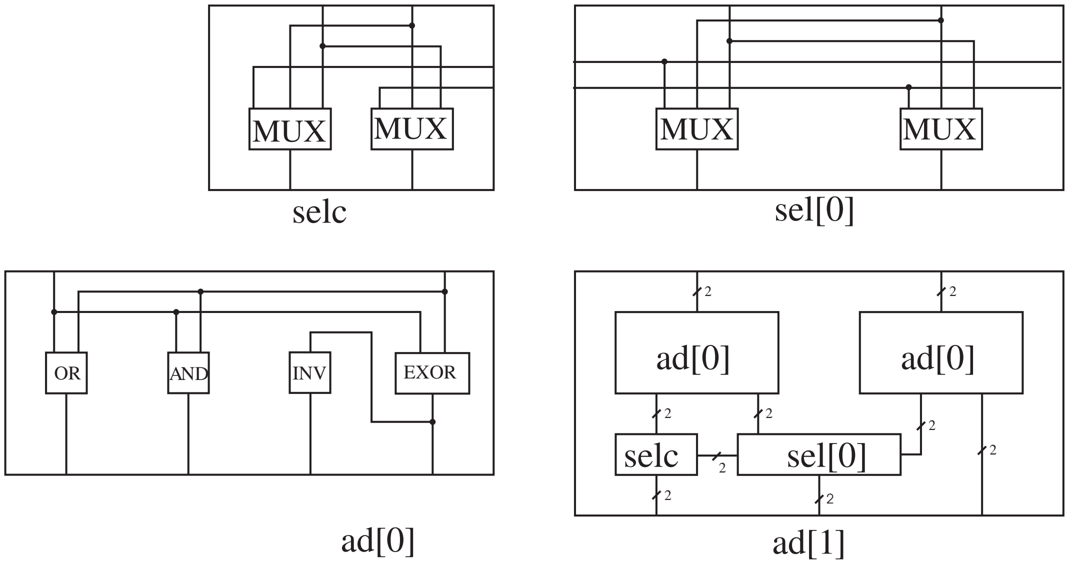

Abbildung 4.1: Ein 2–Bit Conditional–Sum–Adder

ihre Erläuterungen) Die Querstriche an den Leitungen bei ad[1] kennzeichnen Busse. Die Busbreite steht daneben. Abbildung 4.2 zeigt drei verschiedene Darstellungen der hier-

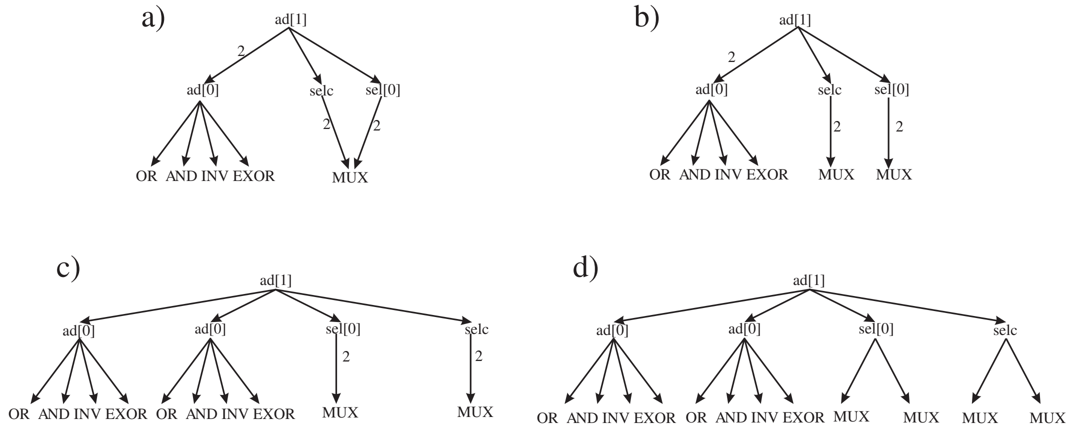

Abbildung 4.2: Darstellungen von ad[1] mittels dreier dags und des Baumes d)

archischen Struktur der Schaltung ad[1]. Abbildung 4.2a zeigt die gefaltete Darstellung, 4.2d zeigt die flache Darstellung, die Abbildungen 4.2 b und c zeigen Teilentfaltungen. Die Kantenmarkierungen bei 4.2 a, b und c geben an, wie viele Verweise die Kante repräsentiert (wenn die Anzahl der Verweise > 1 ist). Die Abbildung 4.2b geht aus 4.2a durch einmalige Anwendung der Operation einfache Teilentfaltung hervor, ebenso Abbildung 4.2c aus Abbildung 4.2b. Das Beispiel gibt einen kleinen Eindruck über die Kompaktheit der gefalteten Darstellung: Der MUX, der in der gefalteten Darstellung (siehe Abbildung 4.2a) ein definierendes Vorkommen hat, hat vier Vorkommen in der flachen Darstellung (siehe Abbildung 4.2b).

Die obigen Definitionen beschreiben die hierarchischen Beziehungen zwischen den Teilschaltungen. Genaueres über Entwurfsregeln, Topologie und Semantik wurde nicht ausgesagt. Wir gehen auf diese Punkte nicht weiter ein und verweisen stattdessen auf das Netzkalkül für integrierte Schaltungen, das seine Wurzeln in [Hot74] hat und das in [Mol86] sowie in [KMO89] beschrieben wird. Eine wichtige Entwurfsregel lautet dort beispielsweise (hier informal): Ein Makro als angewandtes Vorkommen seiner ihn definierenden Teilschaltung soll „anschlußkompatibel“ mit ihr sein, d.h. es soll die gleiche Reihenfolge und den gleichen Typ von Anschlüssen an den Rändern haben.
Liegt eine korrekt spezifizierte Schaltungsdarstellung vor, dann kann sie eindeutig zu einer der drei Definitionen 4.1, 4.2 und 4.4 zugeordnet werden.
Für elektrische Schaltungen gilt folgende Forderung: Jedes Signal hat in der flachen Schaltung genau eine Quelle und mindestens eine Senke. Quellen sind primäre Eingänge oder Bausteinausgänge, mit Senken werden hier Bausteineingänge oder primäre Ausgänge bezeichnet. Wir setzen im folgenden diese Forderung voraus und verwenden sie implizit (beispielsweise im Satz 4.7).

Die weiter unten vorgestellten hierarchischen Algorithmen ordnen zunächst den gegebenen Schaltungen Signalflußrichtungen zu. Damit dies eindeutig möglich ist, muß für die entsprechende flache Schaltung gelten:

- Alle primären Anschlüsse sind über Leitungssegmente mit mindestens einem Anschluß einer Basiszelle verbunden.
- Wenn zwei oder mehr primäre Anschlüsse über Leitungssegmente miteinander verbunden sind, dann müssen sie primäre Ausgänge sein, d.h. sie müssen das Signal eines Ausgangs einer Basiszelle führen.

Jede der obigen Darstellungen einer hierarchischen Schaltung ist ein dag und kann durchmustert werden. CADIC stellt beispielsweise die Funktion *DagDFS(T, netfunction)* zur Verfügung, die Teilschaltungen nach dem DepthFirstSearch–Schema besucht (vergleiche [Bur94]). T ist hierbei die Teilschaltung, auf der die Funktion *netfunction* erstmalig aufgerufen wird, anschließend wird *netfunction* auf allen Knoten, die von T aus erreichbar sind, genau einmal aufgerufen. Die meisten Algorithmen durchmustern den gesamten dag, der Parameter T ist dann die Wurzel des dag. Die Funktion *DagDFS* ist sehr mächtig, insbesondere ermöglicht sie viele verschiedene Möglichkeiten der Aufrufreihenfolge und von Aufrufen innerhalb von Teilschaltungen (vergleiche [BV94]). Es gibt für sie zwei prinzipielle Aufrufreihenfolgen: Bottom–up und Top–down. Ruft man *DagDFS* Bottom–up auf, dann wird ein Knoten erst besucht, wenn seine Kinder (und alle seine Nachfahren) besucht sind.
Besucht man die Teilschaltungen in der topologischen Sortierreihenfolge des dag, so wird folgende Invariante garantiert: Vor dem Aufruf von *netfunction* auf einem Knoten wurde *netfunction* auf seinen Vorgängern (und allen seinen Vorfahren) aufgerufen.

4.1.2 Hierarchische Netzlisten

In Anlehnung an die Dokumentation der CADIC–Datenstruktur [BV94] werden Netze definiert.
Flache Schaltungen bestehen aus Bausteinen, Verdrahtung und Anschlüssen. Die Anschlüsse an den Bausteinen heißen *Pins*, die Anschlüsse am Rand der Schaltung heißen *Pads*. Projektionen sind Leitungsabschlüsse, die weder an einem Pin eines Bausteins noch am Leitungsrand enden.

Eine Netzliste spiegelt für eine Schaltung die Verbindungsstruktur zwischen den Pins und Pads wider, ohne daß die Topographie berücksichtigt wird.

Definition 4.5 :
Ein Netz einer flachen Schaltung ist eine maximale Menge von Pins und Pads, die durch Leitungssegmente miteinander verbunden sind.

Definition 4.6:
Eine Netzliste einer flachen Schaltung ist die Menge aller Netze der Schaltung.

Aus elektrotechnischen Gründen (sonst wird die Schaltung fehlerhaft) verbieten wir unbelegte Eingänge: Jeder Eingang eines Bausteins ist Element genau eines Netzes.
I_p sei die Menge der Eingangspads (d.h. die Menge der primären Eingänge; primary Inputs), O_c sei die Menge der Ausgangspins in der Schaltung (Outputs of cells). S bezeichne die Vereinigung dieser beiden Mengen von Anschlüssen: $S := I_p \cup O_b$ (S steht für Quellen; sources). Einem Netz kann man ein logisches Signal zuordnen, denn es soll gelten: Jedes Netz enthält genau ein Element aus S. Jedes generierte Signal soll wiederum verwendet werden, insgesamt erhält man also Satz 4.7:

Satz 4.7:
Es gibt eine Bijektion zwischen der Menge S und der Netzliste.

Im folgenden werden hierarchisch spezifizierte Schaltungen betrachtet, die in teilentfalteter oder gefalteter Darstellung als dag vorliegen. Instanzen von Teilschaltungen in der aktuellen Teilschaltung heißen Makros. Auf dieser Ebene soll von den Makros nur die Anschlußbelegung bekannt sein, ansonsten sind sie „black boxes“.
Pins und Pads erhalten in hierarchisch beschriebenen Schaltungen eine neue Bedeutung: Anschlüsse an den Rändern von Basisbausteinen und Makros heißen Pins. Pads bezeichnen Anschlüsse am Rand der aktuell betrachteten Teilschaltung. Sie sind nur für die Teilschaltung auf oberster Hierarchieebene primäre Anschlüsse. Für die übrigen Teilschaltungen sind sie das „Interface“ für die Schaltungen, in denen sie als Instanz vorkommen.

Zu einem gegebenen dag betrachten wir eine Teilschaltung t. Entsprechend den flachen Schaltungen definieren wir:

Definition 4.8:
Ein Netz von t ist eine maximale Menge von Pins und Pads, die durch Leitungssegmente dieser Teilschaltung miteinander verbunden sind.

Ebenfalls analog zum flachen Fall wird die Netzliste definiert:

Definition 4.9:
Die Netzliste von t ist die Menge aller Netze von t.

Um anhand der Definitionen 4.8 und 4.9 die Netzliste einer Teilschaltung zu finden, benötigt man kein Wissen über die innere Verdrahtung der Makros.
Hierzu ein Beispiel: Man betrachte eine Teilschaltung, die ein Makro enthält. Zwei Pins dieses Makros sollen in der Netzliste der definierenden Teilschaltung dieses Makros zum

selben Netz gehören (dort als Pads). Diese beiden Pins befinden sich in der aktuellen Teilschaltung in zwei verschiedenen Netzen, wenn sie nicht über Leitungssegmente miteinander verbunden sind. Nach Definition 4.10 gehören sie jedoch auf jeden Fall zu demselben erweiterten Netz.

t sei eine Teilschaltung im dag. Für t wird ein ungerichteter Graph G_t wie folgt gebildet: Zwischen den Knoten im Graphen und den Anschlüssen in t gibt es eine Bijektion. Wenn zwei Anschlüsse im Graphen miteinander durch Leitungssegmente verbunden sind, so existiert eine ungerichtete Kante zwischen ihnen. Gehören zwei Anschlüsse eines Makros innerhalb der das Makro definierenden Teilschaltung zu demselben erweiterten Netz, dann enthält G_t eine Kante zwischen diesen beiden Anschlüssen. Das sind alle Kanten.

Definition 4.10:
Die erweiterten Netze von t werden rekursiv definiert:
Falls t keine Makros enthält, dann sind die erweiterten Netze identisch zu den Netzen der Definition 4.8.
Sonst gilt: Ein erweitertes Netz ist die Menge der Knoten einer Zusammenhangskomponente in G_t.

Es gibt eine Bijektion zwischen der Menge der Zusammenhangskomponenten und der erweiterten Netzliste:

Definition 4.11:
Die erweiterte Netzliste von t ist die Menge aller erweiterten Netze.

Die Netze und Netzlisten der Definitionen 4.10 und 4.11 werden für den hierarchischen Max–Fanout–Algorithmus benötigt, um das akkumulierte Fanout bottom-up berechnen zu können. Sie können durch einen bottom-up Lauf über die hierarchische Schaltung berechnet werden.

Abbildung 4.3 zeigt den Selektor sel[1] (vergleiche hierzu Abbildung 4.24 und ihre Erläuterungen), dessen Netzlisten im folgenden aufgeführt sind:

Schaltung	Netzliste	erweiterte Netzliste
sel[0]	$\{N_0^0, P_3, P_7\}, \{N_0^1, P_2, P_6\},$ $\{W_0^0, P_1, E_0^0\}, \{W_0^1, P_5, E_0^1\},$ $\{S_0^0, P_4\}, \{S_0^1, P_8\}$	$\{N_0^0, P_3, P_7\}, \{N_0^1, P_2, P_6\},$ $\{W_0^0, P_1, E_0^0\}, \{W_0^1, P_5, E_0^1\},$ $\{S_0^0, P_4\}, \{S_0^1, P_8\}$
sel[1]	$\{N_1^0, P_9\}, \{N_1^1, P_{10}\},$ $\{N_1^2, P_{11}\}, \{N_1^3, P_{12}\},$ $\{W_1^0, P_{13}\}, \{P_{14}, P_{15}\}, \{P_{16}, E_1^0\},$ $\{W_1^1, P_{17}\}, \{P_{18}, P_{19}\}, \{P_{20}, E_1^1\},$ $\{S_0^0, P_{21}\}, \{S_0^1, P_{22}\},$ $\{S_1^2, P_{23}\}, \{S_1^3, P_{24}\}$	$\{N_1^0, P_9\}, \{N_1^1, P_{10}\},$ $\{N_1^2, P_{11}\}, \{N_1^3, P_{12}\},$ $\{W_1^0, P_{13}, P_{14}, P_{15}, P_{16}, E_1^0\}$ $\{W_1^1, P_{17}, P_{18}, P_{19}, P_{20}, E_1^0\}$ $\{S_0^0, P_{21}\}, \{S_0^1, P_{22}\},$ $\{S_1^2, P_{23}\}, \{S_1^3, P_{24}\}$

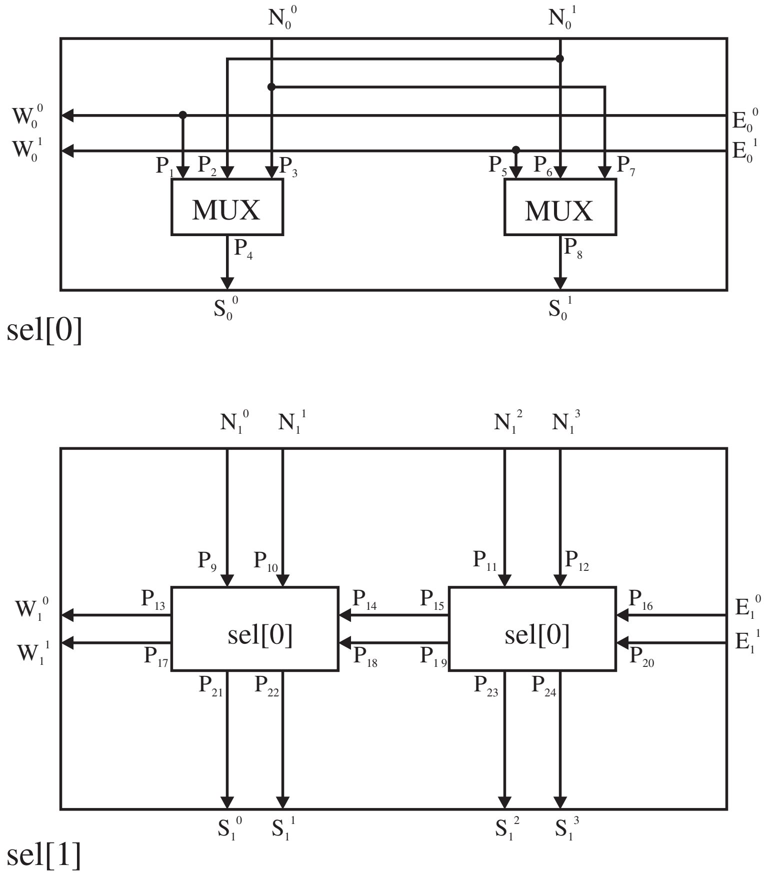

Abbildung 4.3: Veranschaulichung hierarchischer Netzlisten

4.1.3 Erlaubte und verbotene Operationen zur Schaltungsmodifikation

Wie schon im letzten Kapitel beschrieben, wird durch einen Fanout–Algorithmus eine Schaltung verändert: Es werden Bausteine durch logisch äquivalente Bausteine ersetzt (repower) oder Treiberbäume eingesetzt. Wendet man diese Strategien auf hierarchische Schaltungen an, dann muß man definieren, inwieweit die Schaltung und ihre hierarchische Struktur verändert werden darf. Erlaubt sei:

- die teilweise oder vollständige Entfaltung der hierarchischen Struktur,
- die Erzeugung von Varianten: falls eine Teilschaltung durch Entfaltung mehrfach im beschreibenden dag vorliegt, dann darf jedes Vorkommen - innerhalb der Grenzen, die durch Verbote vorgegeben sind - verändert werden.

- die Festschreibung der Signalflußrichtung eines Anschlusses einer Teilschaltung, falls diese Richtung in sämtlichen Vorkommen der Teilschaltung als Instanz gleich ist,
- die Anwendung der repower–Strategie und das Einsetzen von Treiberbäumen innerhalb von Teilschaltungen.
- das Ersetzen eines Instanzverweises durch einen Instanzverweis auf eine Teilschaltung, die die selbe logische Funktion berechnet und die anschlußkompatibel (siehe unten die Erläuterung zum „Interface“ einer Teilschaltung) ist, die aber ein anderes zeitliches Verhalten haben kann. Anmerkung: Falls durch solche Operationen eine Teilschaltung nach Ende aller Modifikationen keine angewandten Vorkommen mehr hat (d.h. sie kommt nirgendwo mehr als Makro vor), dann wird sie und alle von ihr ausgehenden Kanten aus dem dag gelöscht. Diese Löschoperation wird solange durchgeführt, bis es keine Teilschaltung mehr gibt, die kein angewandtes Vorkommen hat.

Diese Operationen gehen über die in [Mol86] erlaubten Operationen auf der Bikategorie der logisch–topologischen Netze hinaus. Eine Modifikation muß jedoch das „Interface“ einer Teilschaltung bestehen lassen: Die Veränderung der Anzahl, Reihenfolge und Typen der Anschlüsse am Rand einer Teilschaltung ist nicht erlaubt. Weiter ist eine Veränderung der Logik der Schaltung, die über das funktionserhaltende Einsetzen von Treiberbäumen und das Ersetzen von Bausteinen durch äquivalente Bausteine hinausgeht, verboten.

Andere Ansätze zur Lösung des Fanout–Problems wie Logikvervielfachung, Logikneusynthese oder Neusynthese des kritischen Pfades werden im Rahmen dieser Arbeit nicht behandelt.

4.2 Eindeutige Signalflußrichtungen

Wenn ein hierarchischer Max–Fanout oder ein hierarchischer Fanout–Algorithmus Treiber oder Fanout–Bäume einsetzt, so müssen die Signalflußrichtungen der Leitungen, die dabei ersetzt werden, bekannt und eindeutig sein. Wie man solche eindeutigen Signalflußrichtungen erzeugen kann, soll in den beiden folgenden Abschnitten gezeigt werden.

4.2.1 Teilentfaltungen

Eindeutige Signalflußrichtungen lassen sich mit Hilfe von Schaltungsmodifikationen erreichen, bei denen die Schaltung teilentfaltet wird. Wie man die Teilentfaltungen zur Herstellung eindeutiger Signalflußrichtungen effizient und mit möglichst wenig Varianten durchführt, soll im folgenden betrachtet werden.

Die Abbildungen 4.4 und 4.5 zeigen erweiterte Netze, bei denen der Signalfluß nicht durch die isolierte Betrachtung der Teilschaltung bestimmt werden kann. Die Fragezeichen an

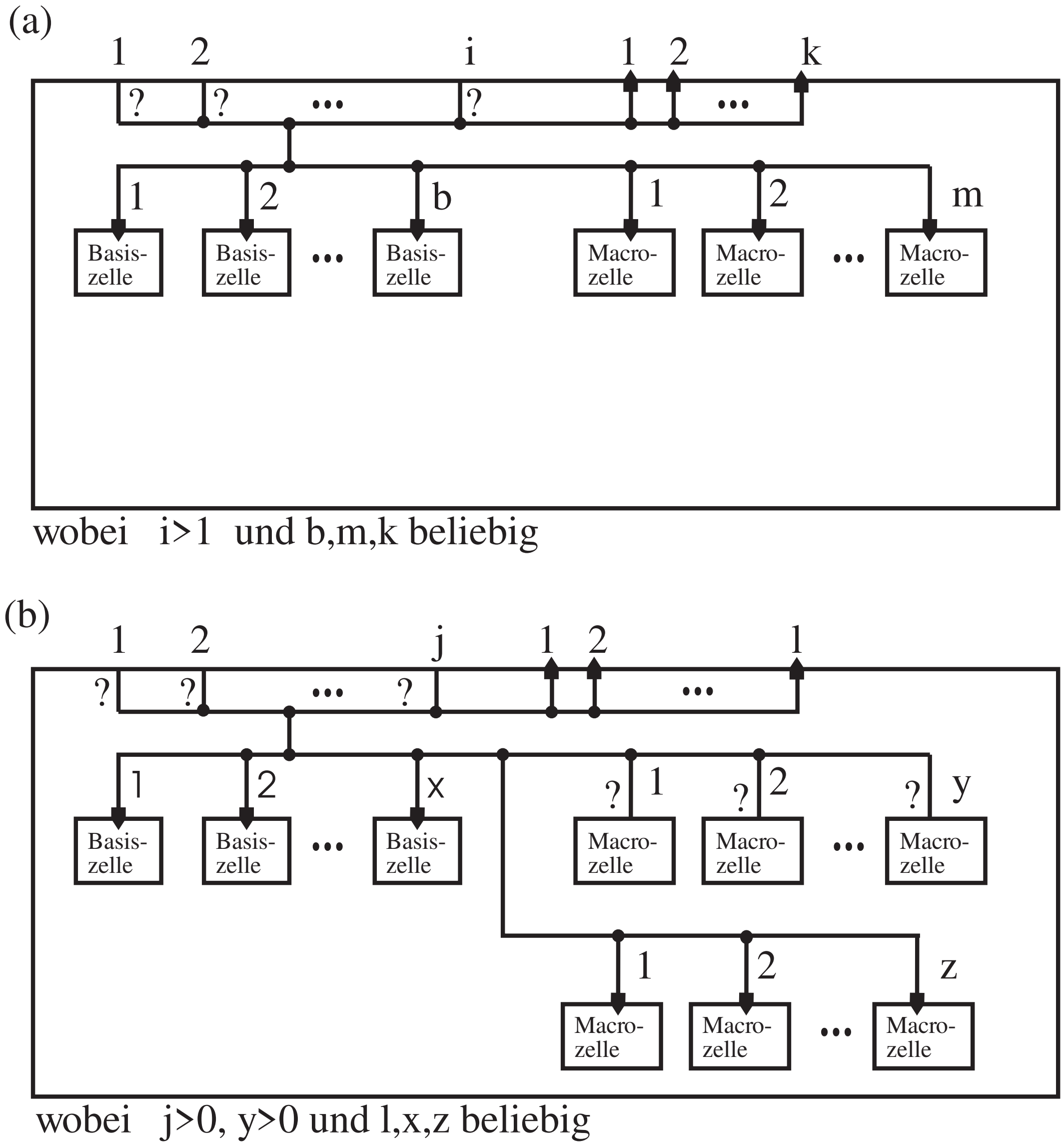

Abbildung 4.4: Netze mit Anschlüssen am Rand, die unbekannte Signalflußrichtungen haben

den Anschlüssen stehen für unbekannte Signalflußrichtungen. Die Abbildungen veranschaulichen allgemeine Situationen, d.h. vom Verlauf der Verdrahtung und der Position der Bausteine kann abstrahiert werden. Die Abbildungen vereinfachen den allgemeinen Fall jedoch wie folgt: Im allgemeinen Fall können (nicht erweiterte) Netze durch Anschlüsse an Makros miteinander elektrisch verbunden sein und so das erweiterte Netz bilden. Dies kommt in den Abbildungen nicht vor. Ferner hat jede Basiszelle und jedes Makro in den Abbildungen genau einen Anschluß mit dem erweiterten Netz gemeinsam

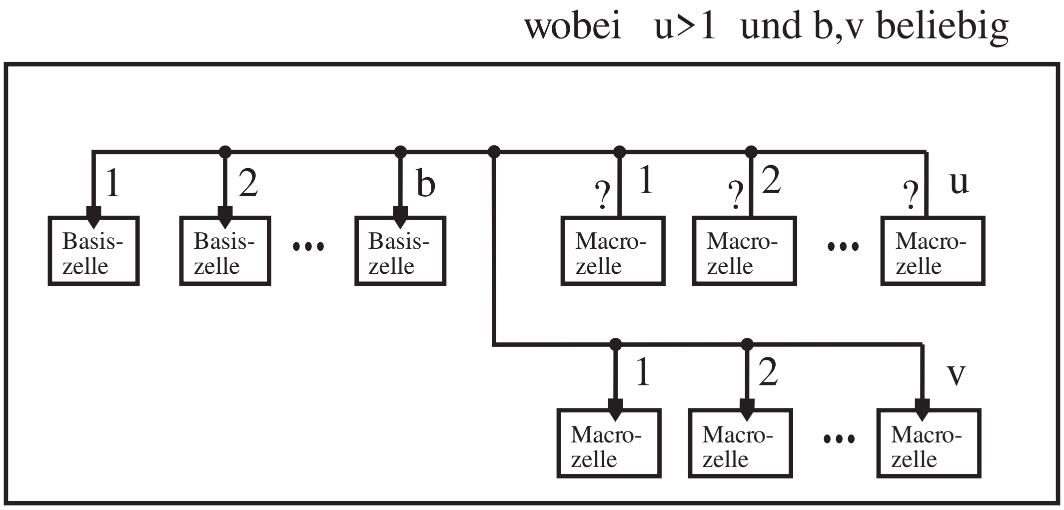

Abbildung 4.5: Netz ohne Anschluß am Rand, das unbekannte Signalflußrichtungen hat

(und nicht mehr), was ebenfalls eine Einschränkung des allgemeinen Falles darstellt. Sobald diese Einschränkungen im folgenden relevant werden, wird auf sie eingegangen.

Die verschiedenen Fälle unterscheiden sich in der Art der Anschlüsse: Abbildung 4.4a steht für Situationen, bei denen sämtliche Signalflußrichtungen nur dann bestimmt werden können, wenn entweder ein Anschluß am Rand als Eingang identifiziert ist oder wenn alle Anschlüsse am Rand, bis auf einen, als Ausgang identifiziert sind. Die Signalflußrichtung für Abbildung 4.4a läßt sich also nur bestimmen, wenn Informationen über die Signalflußrichtungen an den Stellen bekannt sind, wo die Teilschaltung eingesetzt wird (dies legt eine top–down Vorgehensweise bezüglich des dag nahe).
Um die Signalflußrichtungen für das erweiterte Netz der Abbildung 4.4b bestimmen zu können, benötigt man im allgemeinen entweder Informationen über die Signalflußrichtungen von Anschlüssen am Rand oder von Anschlüssen an Makros, oder von beiden. In diesem Fall muß man also im allgemeinen Informationen über die Stellen haben, in die die Teilschaltung eingesetzt wird und über die Signalflußrichtungen an den Makroanschlüssen (dies legt zwei Vorgehensweisen bezüglich des dag nahe: top–down und bottom–up).
Die erweiterten Netze der Abbildung 4.4 haben mindestens einen Anschluß am Rand der Teilschaltung im Gegensatz zum erweiterten Netz der Abbildung 4.5. Die Signalflußrichtungen von Netzen der Abbildung 4.5 lassen sich also bestimmen, wenn Informationen über Signalflußrichtungen der Anschlüsse der Makros bekannt sind (dies legt eine bottom–up Vorgehensweise bezüglich des dag nahe).

Den Signalfluß kann man mit folgenden hinreichenden Bedingungen bestimmen. Sie sind die Grundlage für den nachfolgenden Algorithmus. In Abbildung 4.4a wird der Signalfluß aller Leitungen bestimmbar, sobald einer der Anschlüsse am Rand als Eingang festgelegt wird. Für die Situation der Abbildung 4.4b gilt: Wenn ein Anschluß am Rand als Eingang festgelegt wird oder wenn das erweiterte Netz mit dem Ausgang einer Basiszelle in einer tieferen Hierarchiestufe verbunden ist (man stelle sich vor, das betreffende Makro sei als flache Schaltung in der Teilschaltung eingesetzt), dann sind ebenfalls die Signalflüsse aller Leitungen bestimmbar. In der Situation der Abbildung 4.5 muß das erweiterte Netz mit dem Ausgang einer Basiszelle in einer tieferen Hierarchiestufe verbunden sein (wie oben: es gibt eine Makrozelle, die, als flache Schaltung betrachtet, einen Ausgang einer Basiszelle enthält und dieser ist mit dem betrachteten erweiterten Netz der aktuellen Teilschaltung verbunden), was die eindeutige Festlegung der Signalflußrichtugen des erweiterten Netzes ermöglicht.

Der Algorithmus benutzt die in Satz 4.12 wiedergegebene Eigenschaft der erweiterten Netze:

Satz 4.12:
Für jedes erweiterte Netz, welches sich in einer Teilschaltung befindet und keinen Anschluß am Rand der Teilschaltung hat, und für jedes erweiterte Netz im Wurzelknoten des dag können die Signalflußrichtungen eindeutig zugeordnet werden.

Eine Folgerung aus Satz 4.12 lautet: In Netzen, die die Form aus Abbildung 4.5 haben, können die Signalflußrichtungen eindeutig festgelegt werden, ohne daß für diese Netze Varianten erzeugt werden müssen. Dies gilt auch für Netze der Abbildung 4.4, sofern sie im Wurzelknoten vorkommen. Die einzigen Netze, die je nach Vorkommen der Teilschaltung unterschiedliche Signalflußrichtungen haben können, sind die Netze aus Abbildung 4.4, wenn sie nicht im Wurzelknoten vorkommen.
Satz 4.12 ist eine Folgerung der Erweiterung von Satz 4.7 auf hierarchisch spezifizierte Schaltungen: Jedes Netz, von dem man alle primären Anschlüsse der Schaltung und alle Anschlüsse an Basiszellen kennt, hat genau eine Quelle. Die Quelle ist entweder ein Ausgang einer Basiszelle oder ein primärer Eingang.
Diese Tatsache kann man sich für den Algorithmus zunutze machen: Ist die Quelle des hierarchischen Netzes identifiziert, dann kann man die Signalflußrichtungen bestimmen.

Wir skizzieren kurz die Idee des Algorithmus. Vor dem ersten Teils des Algorithmus werden alle Anschlüsse an den Rändern von Teilschaltungen und die Anschlüsse der Makros mit zwei zweiwertigen Flags u_i (= <u>u</u>sed as <u>i</u>nput) und u_o (= <u>u</u>sed as <u>o</u>utput) beschriftet. Der Wertebereich ist {TRUE,FALSE}. Initial haben alle Beschriftungen die Werte: u_i =FALSE und u_o=FALSE. Im ersten Teil des Algorithmus werden die Signalflußrichtungen bestimmt, die bottom–up eindeutig festlegbar sind (vergleiche Abbildung 4.5). Darüber hinaus propagiert der Algorithmus bottom–up alle in dieser Phase verfügbaren Informationen über Signalflußrichtungen von den Teilschaltungen zu ihren Makrovorkommen. Falls ein Anschluß wenigstens einmal als Eingang geschaltet wird, dann setzt der Algorithmus u_i auf TRUE, wird er wenigstens einmal als Ausgang geschaltet, dann erhält u_o den Wert TRUE. Im zweiten Teil des Algorithmus werden die top–down berechenbaren Informationen über Signalflußrichtungen durch einen top–down Lauf propagiert und gleichzeitig die Teilentfaltungen durchgeführt. Hierbei erben die Makorvorkommen die TRUE–Werte von u_i und u_o ihrer definierenden Teilschaltungen. Die zweigeteilte Vorgehensweise - erst bottom–up, dann top–down - garantiert, daß die Informationen über die Quellen eines Netzes spätestens beim top–down Lauf bekannt sind. Die Signalflußrichtung an einem Anschluß kann beim top–down Lauf genau dann eindeutig festgelegt werden, wenn einer der Werte der Beschriftung FALSE bleibt. Beide Werte können nicht FALSE sein, da nach obigen Voraussetzungen die Signalflußrichtungen eindeutig bestimmbar sind. Sind beide TRUE, dann müssen Varianten erzeugt werden.

Bevor der Algorithmus gestartet werden kann, müssen die erweiterten hierarchischen Netzlisten berechnet sein und die Beschriftungen müssen initialisiert sein.

Der erste Teil des Algorithmus:
Der dag wird mit *DagDFS* bottom–up durchmustert: In jeder besuchten Teilschaltung erben zunächst die Anschlüsse der in ihr enthaltenen Makros die TRUE–Werte von u_i und u_o der sie definierenden Teilschaltungen. Danach wird für jedes erweiterte Netz geprüft, ob ihre Quelle bekannt ist. Dies ist der Fall, wenn das erweiterte Netz einen Ausgang einer Basiszelle oder einen Ausgang einer Makrozelle (d.h. u_o =TRUE) enthält (bzw. mehrere Ausgänge einer Makrozelle, die zu dem selben erweiterten Netz gehören). Falls ja, werden die übrigen Signalflußrichtungen dieses Netzes festgelegt: Man wendet die beiden folgenden Regeln so oft an, bis alle Richtungen festgelegt sind:

1. Ein Netz des erweiterten Netzes enthalte eine Quelle (also einen Ausgangspin einer Basiszelle oder eines Makros) und es gibt noch Anschlüsse innerhalb dieses Netzes, die nicht die folgenden Richtungen haben: Anschlüsse an Makros als Eingangspins und Anschlüsse am Rand als Ausgangspads. In diesem Fall werden diese Richtungen den Anschlüssen zugeordnet.

2. Wenn ein Netz ein Eingangspin eines Makros enthält, und dieser Pin ist über das erweiterte Netz innerhalb der ihn definierenden Teilschaltung mit weiteren Anschlüssen verbunden, dann sind diese Anschlüsse Ausgänge.

Anschließend werden die u_o Werte der Pads (Anschlüsse am Rand) dieses erweiterten Netzes auf TRUE und die Pins an den Makros entsprechend der gefundenen Signalflußrichtungen gesetzt.

Der zweite Teil des Algorithmus:
Im zweiten Teil des Algorithmus werden die Signalflußrichtungen der hierarchischen Schaltung eindeutig gemacht. Dies geschieht durch Vervielfachung von Teilschaltungen, in denen Leitungen je nach Vorkommen unterschiedliche Signalflußrichtungen haben (bei ihnen gibt es Anschlüsse am Rand mit $u_i = u_o$ =TRUE).
Der dag wird topologisch sortiert und entlang dieser Sortierung werden auf jeder Teilschaltung folgende Schritte ausgeführt (durch die Vorgehensweise in topologischer Sortierrichtung ist sichergestellt, daß alle direkten Vorfahren der aktuellen Teilschaltung besucht wurden und damit alle Makrovorkommen der aktuellen Teilschaltung behandelt wurden):

Zunächst erben die Pads der aktuell betrachteten Teilschaltung die TRUE Beschriftungen von ihren Vorkommen als Pins (sie kommen als Pins in den Teilschaltungen vor, wo die aktuelle Teilschaltung Makro ist). Wenn die Teilschaltung danach Netze mit wenigstens einem Pad mit der Beschriftung $u_i = u_o$ =TRUE hat, müssen Varianten von ihr gebildet werden. Der Algorithmus vergleicht hierzu alle ihre Makro–Vorkommen miteinander. Es werden von der Teilschaltung so viele Varianten erzeugt, wie es unterschiedliche Belegungen von Signalflußrichtungen an den Rändern der Makrovorkommen gibt. Varianten sind hier definierende Vorkommen von Teilschaltungen, die das bisher einzige definierende Vorkommen ersetzen, jedoch den gleichen Teildag als Nachfahren haben (vergleiche Abbildung 4.6). Da die Signalflußrichtungen an den Rändern der Varianten nun bekannt sind, lassen sich mit Hilfe der oben beschriebenen beiden Regeln die Signalflußrichtungen aller Leitungen innerhalb jeder Variante bestimmen. Die topologische Durchmusterungsreihenfolge muß nach jeder Modifikation der hierarchischen Struktur den neuen Gegebenheiten angepaßt werden.

Abbildung 4.6 zeigt ein einfaches Beispiel für die Vorgehensweise des beschriebenen Algorithmus.

Für die Max–Fanout und die Fanout–Algorithmen ist es möglich, die Anzahl der Teilentfaltungen minimal zu halten, indem man nur für die Teilschaltungen Varianten erzeugt, in denen Treiber an bidirektionalen Leitungen eingesetzt werden müssen.

4.2.2 Verbotene Schaltungsmodifikationen

Es ist auch möglich, die Signalflußrichtungen aller Leitungen der gefalteten Darstellung ohne Entfaltung eindeutig festzulegen. Dazu muß jedoch das am Anfang dieses Kapitels festgeschriebene Verbot bestimmter Modifikationen von Schaltungen aufgeweicht werden. Bidirektionale Signalflußrichtungen können nur bei Teilschaltungen auftreten, die erweiterte Netze mit mehreren Anschlüssen am Rand haben. Betrachtet man die flache Darstellung einer solchen Teilschaltung, dann können diese erweiterten Netze weder an einen primären Eingang noch an einen Ausgang einer Basiszelle angeschlossen sein (in diesen beiden Fällen wären die Signalflußrichtungen innerhalb des erweiterten Netzes eindeutig). Die Grundidee lautet nun: In allen Teilschaltungen, in denen Netze mit verschiedenen Signalflußrichtungen vorkommen, kann die Verdrahtung zu den entsprechenden Anschlüssen so geändert werden, daß am Ende die Signalflußrichtungen für jedes Vorkommen gleich sind.

Ein Beispiel hierzu: Ein Makro m habe zwei Anschlüsse P_1 und P_2, die beide zu demselben erweiterten Netz dieser Teilschaltung gehören. Je nach Vorkommen des Makros in anderen Teilschaltungen wird P_1 als Eingang und P_2 als Ausgang oder umgekehrt belegt. P_1 sei häufiger als Eingang belegt als P_2. Der Algorithmus legt nun P_1 als Eingang für alle Vorkommen von m fest. Nun werden die Teilschaltungen modifiziert, bei denen P_1 noch Ausgang ist: Die Verdrahtung an P_1 wird gelöscht. Dabei wird entgegen der Signalflußrichtung vorgegangen, bis man auf die erste Verzweigung oder einen Anschluß stößt. Der

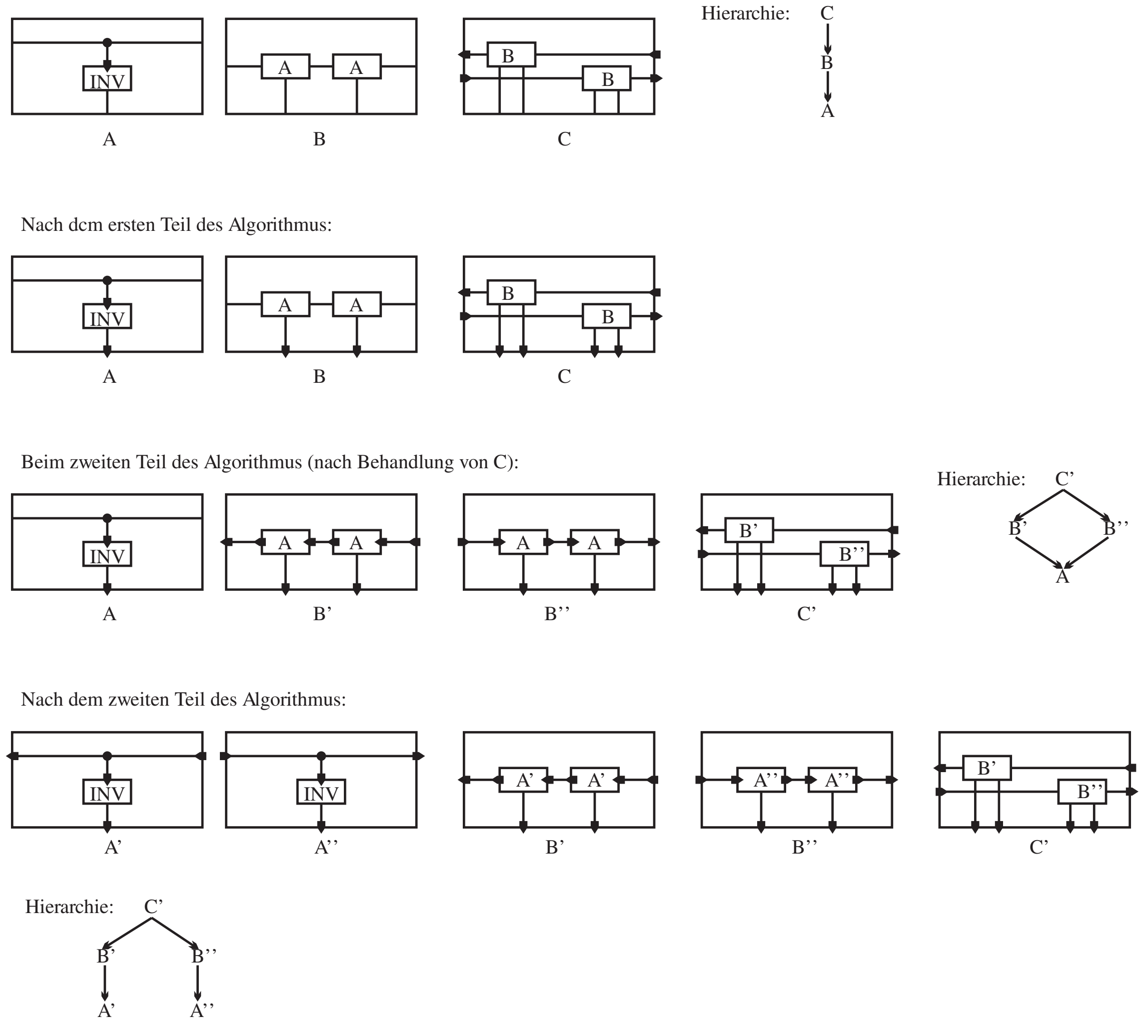

Abbildung 4.6: Beispiel für die Vorgehensweise des Algorithmus, der eindeutige Signalflußrichtungen erzeugt

Punkt, an dem nicht mehr weiter gelöscht wurde, heiße S_1. Von P_2 aus wird in Signalrichtung gelöscht, bis ebenfalls eine Verzweigung oder ein Anschluß erreicht ist. Dieser Punkt heiße S_2. Nun wird von S_1 zu P_2 und von S_2 zu P_1 eine Leitung gezogen.

Wir beschreiben den Algorithmus informal. Für die spezifizierte, hierarchisch beschriebene Schaltung wird vorausgesetzt, daß es in der entsprechenden flachen Schaltung in allen Netzen keine „Verdrahtungsschleifen“ gibt. Mit Verdrahtungsschleife ist jede Anordnung von Segmenten gemeint (Segmente, die die Anschlüsse der Netze miteinander verbinden), die in dem wie folgt konstruierten, ungerichteten Graphen einen Zyklus enthält: Die Knoten sind die Endpunkte der Segmente und die Kanten entsprechen den Verbindungen der Endpunkte durch die Segmente.

Zunächst wird der erste Teil des Algorithmus aus dem Abschnitt „Teilentfaltungen“ ausgeführt. Damit ist sichergestellt, daß die Signalflußrichtungen aller erweiterten Netze der Situation aus Abbildung 4.5 nun festgelegt sind. Es werden jedoch statt der Beschriftung mit binärem Wertebereich zwei Zähler verwendet. Erweiterte Netze, deren Signalflußrichtungen nun noch unbekannt sind, sind Kandidaten für nicht eindeutige Signalflußrichtungen. Mit Hilfe eines top–down Laufes (analog dem 2. Teil des Algorithmus über Teilentfaltungen, jedoch ohne Variantenbildung) aktualisiert man dann die Statistik darüber, wie häufig Anschlüsse am Rand von Teilschaltungen, deren Signalflußrichtung noch nicht bekannt sind, als Ausgang und als Eingang vorkommen. Nun durchläuft der Algorithmus die Teilschaltungen wieder bottom–up. Es werden in der aktuellen Teilschaltung alle erweiterten Netze betrachtet. Die Signalflußrichtungen an den Anschlüssen der Makros sind zu diesem Zeitpunkt festgelegt: Sie wurden entweder im obigen ersten bottom–up Lauf gefunden oder bei einem Besuch der entsprechenden Teilschaltung im aktuellen bottom–up Lauf festgelegt (siehe weiter unten). Aus diesem Grund legt das Programm einen Anschluß am Rand als Quelle (d.h. als Eingang) des erweiterten Netzes fest, und zwar denjenigen (oder einer von denjenigen), der nach der Statistik am häufigsten Eingang war. Durch die Festlegung dieses Randanschlusses als Quelle kann man für alle Anschlüsse des erweiterten Netzes (am Rand und an den Makros) die Signalflußrichtung für diesen Fall bestimmen, beispielsweise mit Hilfe von Tiefensuche (wenn das Signal an einem Makroanschluß ankommt, dann werden die übrigen Makroanschlüsse, die innerhalb der definierenden Teilschaltung zu demselben erweiterten Netz gehören, zu Ausgängen). Da die Signalflußrichtungen der Makros vorher ohne Rücksicht (beziehungsweise lediglich nach statistischen Gesichtspunkten) auf ihre angewandten Vorkommen in dieser Teilschaltung festgelegt wurden, kann es zu Widersprüchen mit den Signalflußrichtungen am Rand der Makros kommen. Diese Widersprüche lassen sich wie folgt auflösen: Man betrachtet die Anschlüsse der Makros in der Reihenfolge der vorangegangenen Tiefensuche. Wenn das Makro für sein internes erweitertes Netz einen anderen Eingang als den durch die Tiefensuche festgelegten hat, wird die Verdrahtung an den entsprechenden Randanschlüssen des Makros bis zum nächsten Anschluß oder bis zum nächsten Verzweigungsknoten (je nachdem was früher vorkommt) gelöscht. Anschließend wird neu verdrahtet: die Makro–Quelle (d.h. der festgelegte Eingang des erweiterten Netzes des Makros) wird mit dem aktuellen signalführenden Anschluß oder Verzweigungsknoten verbunden (dieser ist aus der Tiefensuche bekannt). Jede der übrigen Anschlüsse am Makro (sie sind Ausgänge) werden mit jeweils einer der frei gebliebenen Anschlüsse beziehungsweise Verdrahtungsknoten verbunden.
Die oben gemachte Voraussetzung, daß es keine Verdrahtungsschleife in der flachen Schaltung geben soll, ist notwendig, da es sonst für ein erweitertes Netz eines Makros mehrere signalführende Eingänge geben könnte, und damit wäre obiger Algorithmus nicht korrekt (diese Voraussetzung wurde implizit verwendet, da immer genau ein Randanschluß für ein erweitertes Netz als Eingang festgelegt wurde).

Die Zeichnungen 4.7, 4.8 und 4.9 zeigen die Schritte des Algorithmus anhand einer Beispielschaltung.

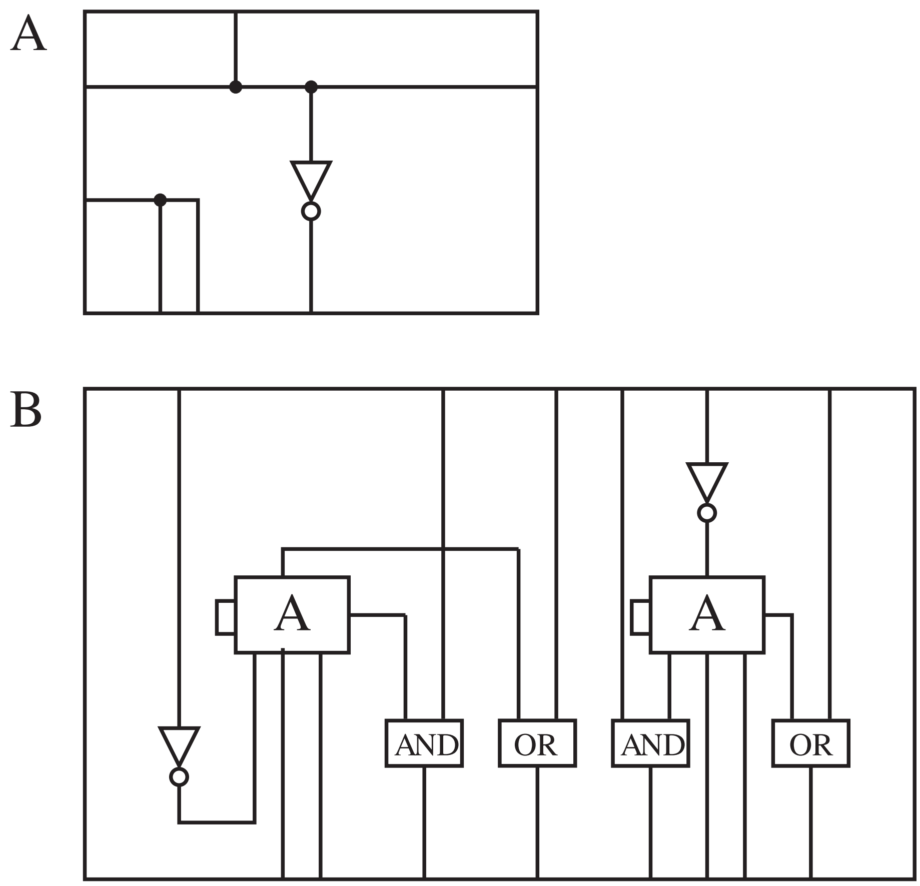

Abbildung 4.7: Die Beispielschaltung, deren Signalflußrichtungen ohne Teilentfaltungen eindeutig gemacht werden sollen.

4.3 Das Fanout–Problem mit Zeitanalyse auf der Hierarchie

4.3.1 Motivation

Die im dritten Kapitel beschriebene Heuristik für das Fanout-Problem wurde für flache Schaltungen entwickelt. Eine Erweiterung auf die hierarchische Darstellung könnte folgende Vorteile haben:

- Die hierarchische Darstellung von gefalteten, regelmäßigen Schaltungen ist kompakt: sie benötigt im allgemeinen wenig Platz.
- Es gibt Algorithmen für Problemklassen, die auf hierarchisch spezifizierten Schaltungen zeiteffizient sind. Nach [KMO89] heißt ein hierarchisches Verfahren zeiteffizient,

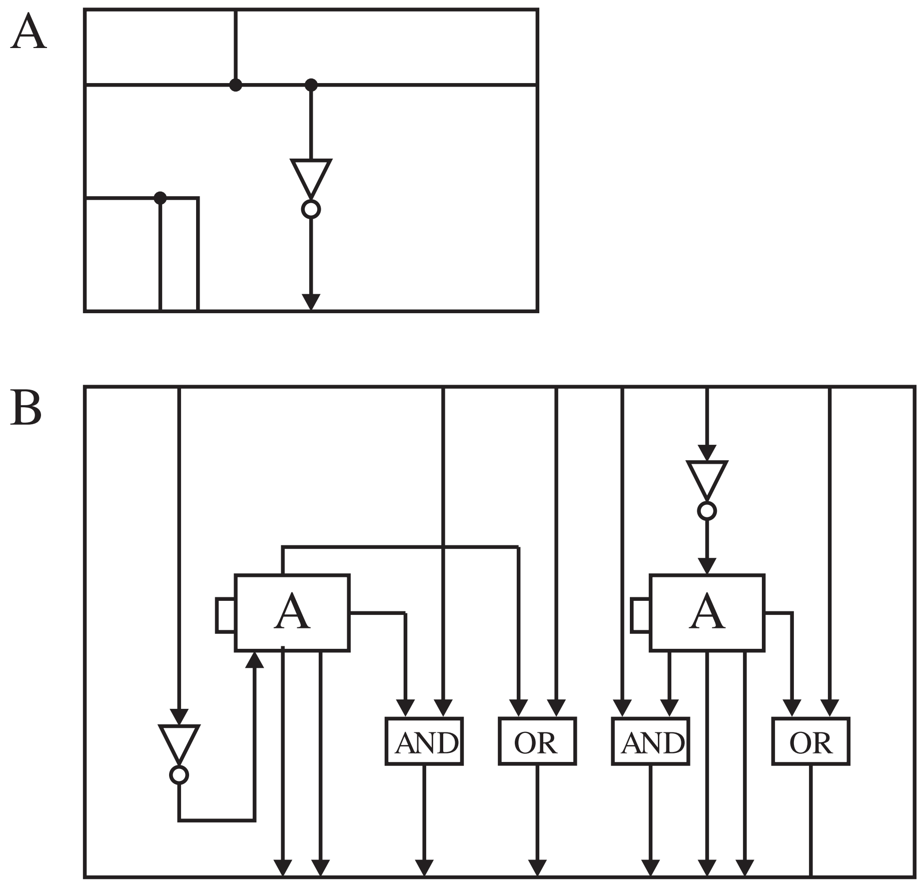

Abbildung 4.8: Die Beispielschaltung nach dem ersten bottom–up Durchlauf

„falls seine Laufzeit nachweislich klein ist, gemessen an der Größe der hierarchischen Eingabe. Ein solches Verfahren hat aus Zeitgründen nicht die Möglichkeit, die hierarchische Eingabe in eine nichthierarchische Darstellung umzuwandeln, da diese exponentiell größer als die Eingabe sein kann“. Ein zeiteffizienter, hierarchischer Fanout–Algorithmus (falls einer existiert) wäre also auf bestimmten Schaltungen (insbesondere solche mit hoher Regelmäßigkeit) wesentlich schneller als ein vergleichbarer „flacher Algortihmus“.

- Bei hierarchisch spezifizierten oder gar parametrisierten Schaltungen gibt es häufig Leitungen, die in mehreren Teilschaltungen (d.h. über Hierarchiegrenzen hinweg) verlaufen. Der menschliche Entwerfer „verliert deswegen leicht den Überblick“ über die Fanout–kritischen Stellen solcher Schaltungen. Der am Ende dieses Kapitels vorgestellte parametrisierte Conditional Sum Adder ist hierfür ein Beispiel. Durch einen Fanout–Algorithmus auf hierarchisch spezifizierten Schaltungen könnte also der Entwurfsprozeß unterstützt werden: Der Entwerfer kann während des Entwurfs

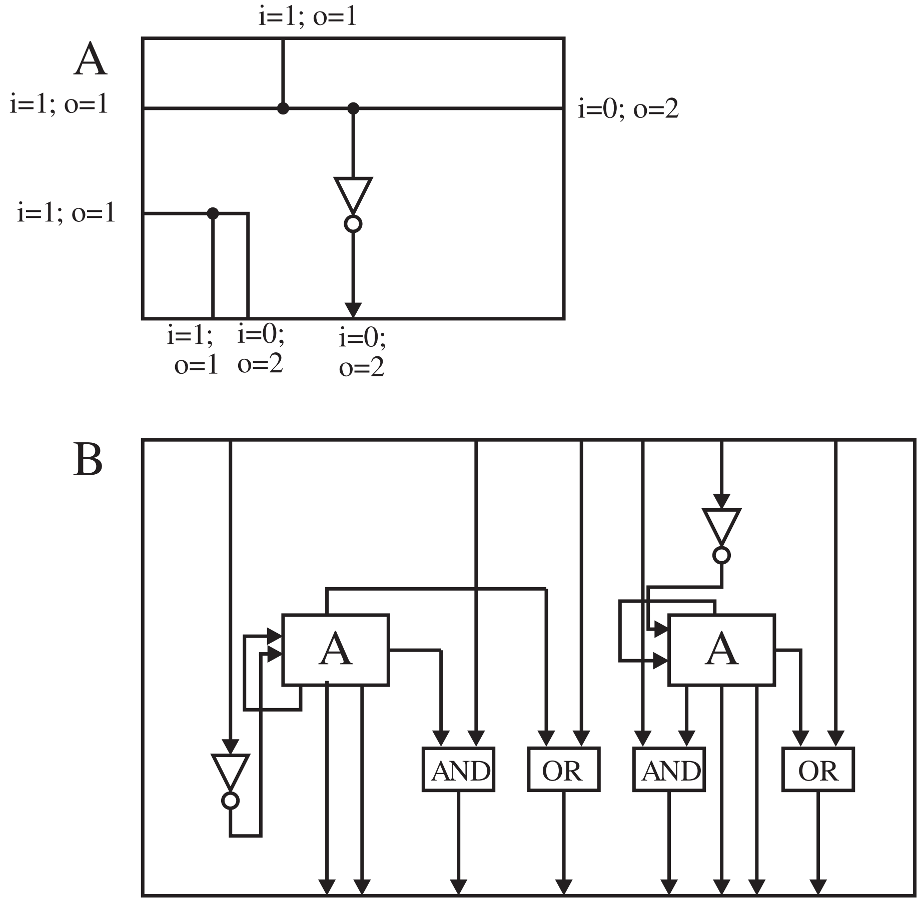

Abbildung 4.9: Die modifizierte Beispielschaltung: Die beiden westlichen Randanschlüsse bei A wurden als Eingang festgelegt.

durch einen Probelauf des Algorithmus feststellen, ob es Stellen gibt, an denen viele Treiber eingesetzt werden müssen. Danach kann er durch geringfügige Änderungen im Ursprungsentwurf das Fanout–Problem für diese kritischen Stellen entschärfen.

4.3.2 Einschränkungen

Im Rahmen dieser Arbeit war ein Fanout–Algorithmus mit Zeitforderungen gesucht, der

- auf allgemeinen (d.h. in der Topologie nicht wesentlich eingeschränkten) gefalteten, hierarchischen Schaltungen arbeitet,
- die hierarchische Darstellung möglichst effizient ausnutzt und

- das Fanout–Problem mit der gleichen Qualität wie der Algorithmus von Singh und Sangiovanni–Vincentelli löst.

Diese Forderungen können aus verschiedenen Gründen nicht gleichzeitig erfüllt werden:

1. Hierarchisch beschriebene Schaltungen sind partitioniert (sie werden durch die Aufteilung in Teilschaltungen „auseinandergerissen"): Dadurch kann die Lösung des Fanout–Problems auf hierarchischen Schaltungen höchstens so gut wie die auf flachen Schaltungen sein. Im allgemeinen muß man einen Qualitätsverlust hinnehmen.

2. Aufgrund der inhärenten Kontextsensitivität des hierarchischen Fanout–Problems (auf Schaltungen, die in der Topologie nicht eingeschränkt sind) muß ein „Kompromiß" zwischen der Güte der Fanout–Korrektur und der Beibehaltung der hierarchischen Struktur gefunden werden. Kontextsensitive Verfahren betrachten die Teilschaltungen nicht als eigenständige Einheiten, sondern berücksichtigen die Umgebungen, in die die Teilschaltungen auf höheren Hierarchieebenen eingesetzt werden (siehe [KMO89]). Das Fanout–Problem ist inhärent kontextsensitiv, weil die Kapazität eines Signals, das aus einer Teilschaltung herausgeführt wird, erst in höheren Hierarchieebenen berechnet werden kann und je nach eingesetzter Stelle unterschiedlich groß sein kann. Das Wissen um die Verteilung der Kapazität wird jedoch benötigt, um qualitativ gute Fanout–Bäume berechnen zu können. Je nachdem, wie die Kapazität auf höheren Hierarchieebenen verteilt ist, müßten unterschiedliche Fanout–Bäume erzeugt werden, um die kritischen Pfade entschärfen zu können. Dazu ist es jedoch im allgemeinen notwendig, Varianten von Teilschaltungen zu erzeugen.

Im folgenden werden die angesprochenen Punkte genauer untersucht.

4.3.2.1 Qualitätsverlust durch die Partitionierung

Unter der Partitionierung einer flachen Schaltung durch die hierarchische Darstellung wird in diesem Zusammenhang die Aufteilung der Schaltung in Teilschaltungen und insbesondere das damit verbundene „Trennen" von Schaltungsteilen verstanden. Im weiteren wird davon ausgegangen, daß eine Partitionierung einer Schaltung nicht modifiziert werden kann, denn es dürfte für allgemeine hierarchische Schaltungen und insbesondere für die gefalteten sehr aufwendig sein, die Partitionierung zu verändern. Durch die Partitionierung kann beispielsweise das Fanout am Ausgang eines Knotens „auseinandergerissen" werden, d.h. ein Signal am Ausgang eines Bausteins wird von mehreren Teilschaltkreisen als Eingang verwendet. Die Partitionierung schränkt also „die Freiheiten" eines Algorithmus beim Einsetzen von Treibern ein. Er wird dadurch im allgemeinen gezwungen sein, im Vergleich zur flachen Schaltung mehr Treiber einzusetzen.

Das nun folgende Beispiel illustriert die angesprochenen Punkte: Abbildung 4.10 zeigt eine Beispielschaltung mit den garantierten Ankunftszeiten an den primären Eingängen und

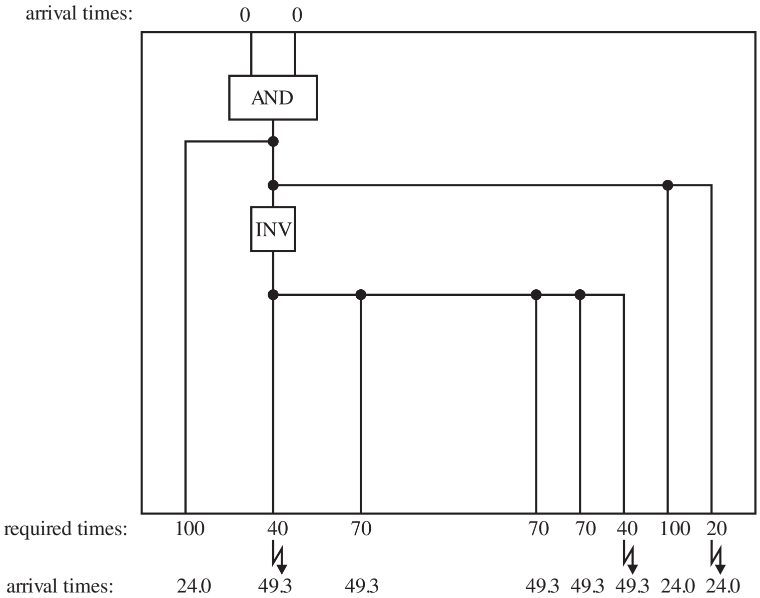

Abbildung 4.10: Die flache Beispielschaltung

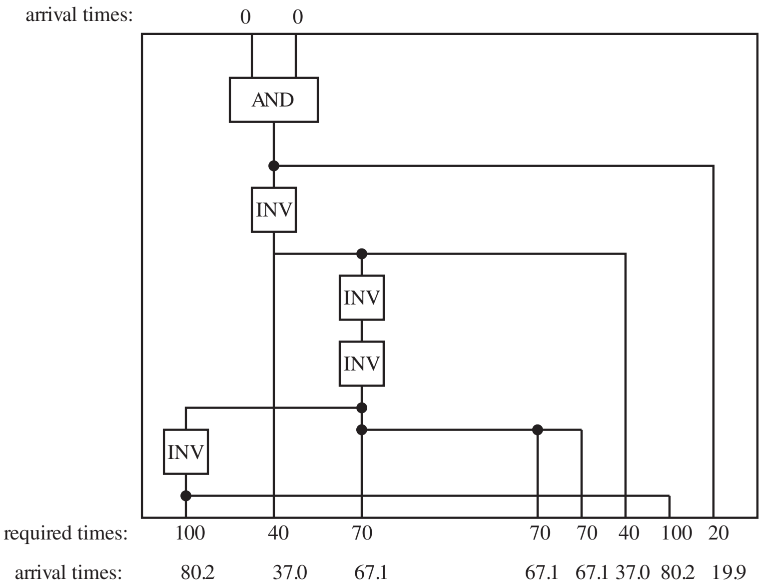

Abbildung 4.11: Die Beispielschaltung nach Anwendung des Fanout-Algorithmus

den Zeitforderungen an den primären Ausgängen. Folgende Daten (in der Größenordnung mit dem IMS–Zellkatalog [VLS90] vergleichbar) sollen dem Beispiel zugrunde liegen:

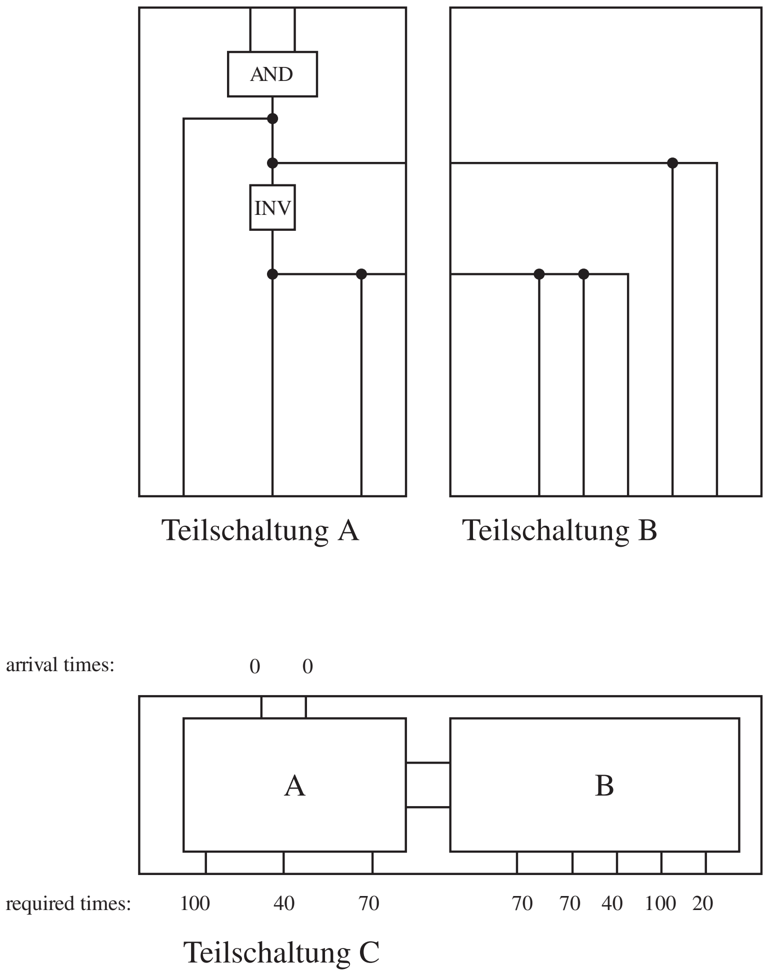

Abbildung 4.12: Die Beispielschaltung, aufgeteilt in zwei Teilschaltungen.

AND-Gatter, linker Eingang: $c_{in} = 83fF$; $i = 15ns$
AND-Gatter, rechter Eingang: $c_{in} = 84fF$; $i = 16ns$
AND-Gatter, Ausgang: $e = 0.027\frac{ns}{fF}$
Inverter, Eingang: $c_{in} = 73fF$; $i = 5ns$
Inverter, Ausgang: $e = 0.054\frac{ns}{fF}$
Die Kapazitäten an den primären Ausgängen seien einheitlich zu 75 fF angenommen. In Abbildung 4.10 stehen unterhalb der geforderten Ankunftszeiten an den primären Ausgängen die tatsächlichen Ankunftszeiten. Die Ankunftszeiten werden berechnet, indem man mit Hilfe obiger Verzögerungsdaten und den beiden Formeln 3.1 und 3.2 von den primären Eingängen zu den primären Ausgängen die Verzögerungen aufsummiert. Immer, wenn die tatsächlichen Ankunftszeiten die geforderten Zeiten überschreiten, ist in den Zeichnungen ein Widerspruchspfeil zu sehen. Abbildung 4.11 zeigt schließlich die

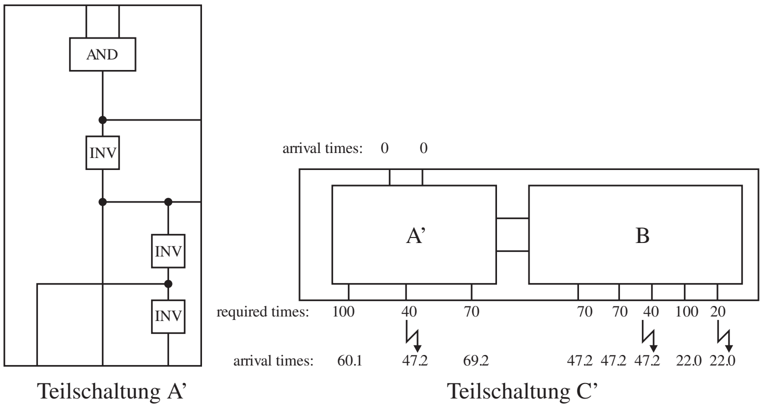

Abbildung 4.13: Die partitionierte Beispielschaltung mit Treibern

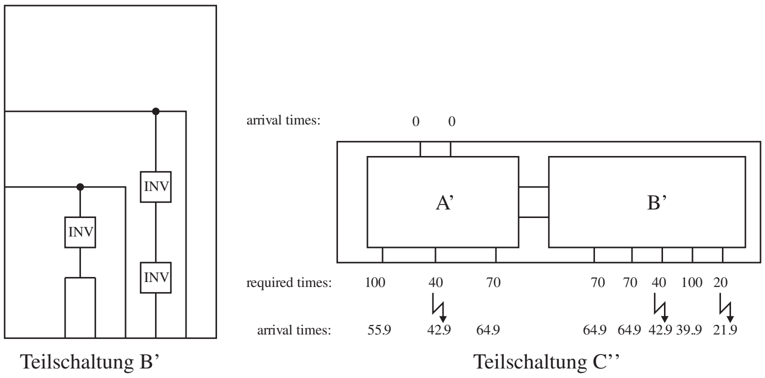

Abbildung 4.14: Auch diese Ersatzschaltung erreicht nicht die Zeitspezifikationen

nach Durchlauf des Algorithmus von Singh und Sangiovanni-Vincentelli modifizierte Beispielschaltung. Die Zeitforderungen an den primären Ausgängen sind erfüllt.
Die Ursprungsschaltung soll beim hierarchischen Entwurf auf zwei Teilschaltungen, wie in Abbildung 4.12 gezeigt, aufgeteilt werden. Das Einsetzen von Treibern führt beispielsweise zu der Schaltung in Abbildung 4.13. Wie man sieht, wird das Resultat im hierarchischen Fall schlechter, da die Verdrahtung zwischen den Teilschaltungen nicht verändert werden kann. Wenn man in der rechten Teilschaltung ebenfalls Treiber einsetzt, dann erhält man z.B. das in Abbildung 4.14 wiedergegebene Ergebnis. Trotzdem erreicht die Schaltung der Abbildung 4.14 nicht die Zeitforderungen. Egal an welchen Stellen man Treiber oder Treiberstufen einsetzt, der rechteste Ausgang wird nie die Zeitforderungen erfüllen.

4.3.2.2 Nachteile durch die Kontextsensitivität

Für den Versuch, den Algorithmus von Singh und Sangiovanni-Vincentelli auf hierarchisch spezifizierte Schaltungen zu übertragen, werden seine Eigenschaften hier nochmals kurz zusammengefaßt: Der Algorithmus hat als Eingabe eine Schaltung sowie spezifizierte arrival times an den primären Eingängen und spezifizierte required times an den primären Ausgängen. Er berechnet eine Ersatzschaltung mit dem gleichen logischen Verhalten. Durch Ersetzen von Bausteinen und das Einfügen von Treiberbausteinen inklusive einer Neuaufteilung des Fanouts wird versucht, die Zeitforderungen zu erfüllen. Entweder werden die Zeitforderungen an allen Ausgängen erfüllt, oder der Algorithmus berechnet die nach seinen Heuristiken bestmögliche Annäherung. Kann man obige Vorgehensweise auf hierarchische Schaltungen übertragen?

Hierbei ist nur eine Übertragung auf gefaltete (oder „schwach" teilentfaltete) Schaltungen interessant, da ein Algorithmus, der nur auf der flachen Darstellung angewendet werden kann, besser direkt auf flachen Schaltungen läuft. Diese sind nicht in Teilschaltungen zerlegt und verbrauchen weniger Platz. Wie oben bereits erwähnt, ist das Fanout–Problem auf der von uns definierten hierarchischen Darstellung kontextsensitiv. Setzt man Teilschaltungen in andere ein, so können unterschiedliche Fanout–Korrekturen je nach Einsetzung sinnvoll sein. Dies gilt sowohl für die einzusetzenden Teilschaltungen, als auch für die Teilschaltungen, in die eingesetzt wird.

Eine Konsequenz dieser kontextabhängigen Fanout–Korrekturen ist Satz 4.13:

Satz 4.13:
Gegeben ist eine Schaltung in gefalteter hierarchischer Darstellung mit arrival times an den primären Eingängen und required times an den primären Ausgängen. Gesucht ist eine Lösung dieses Fanout–Problems mit Zeitanalyse, die qualitativ möglichst nahe an die Lösung des Algorithmus von [SSV90] auf der entsprechenden flachen Schaltung heranreicht (oder sie erreicht). Im allgemeinen kann die Lösung im hierarchischen Fall nicht zeiteffizienter als die Lösung im flachen Fall gefunden werden.

Satz 4.13 kann durch das Beispiel in Abbildung 4.15 belegt werden. Die Schaltung und die Technologie in Abbildung 4.15 orientieren sich nicht an realen Daten, sondern wurden idealtypisch gewählt. So wird beispielsweise von den Leitungskapazitäten abstrahiert. Außerdem wird vereinfachend angenommen, daß jeder Baustein ein Fanout von 2 ohne Zeitverzögerung treiben kann. Ein höheres Fanout als 3 sei technologiebedingt untersagt. Für dieses Fanout–Problem auf der gegebenen Technologie ist die in Abbildung 4.16 dargestellte Lösung die einzig mögliche Lösung. Für jedes Vorkommen einer Teilschaltung, die durch einen inneren Knoten im dag repräsentiert wird, mußte eine Ersatzdarstellung gefunden werden. Betrachtet man nur die Teilschaltungen und nicht die Basiszellen, so gilt: Die Ursprungsschaltung muß in die flache Darstellung umgewandelt werden, um das Fanout–Problem lösen zu können. Dies ist ein extremes Beispiel für die Kontextsensitivität des Fanout–Problems. Am Beispiel sieht man außerdem: Für jedes Vorkommen

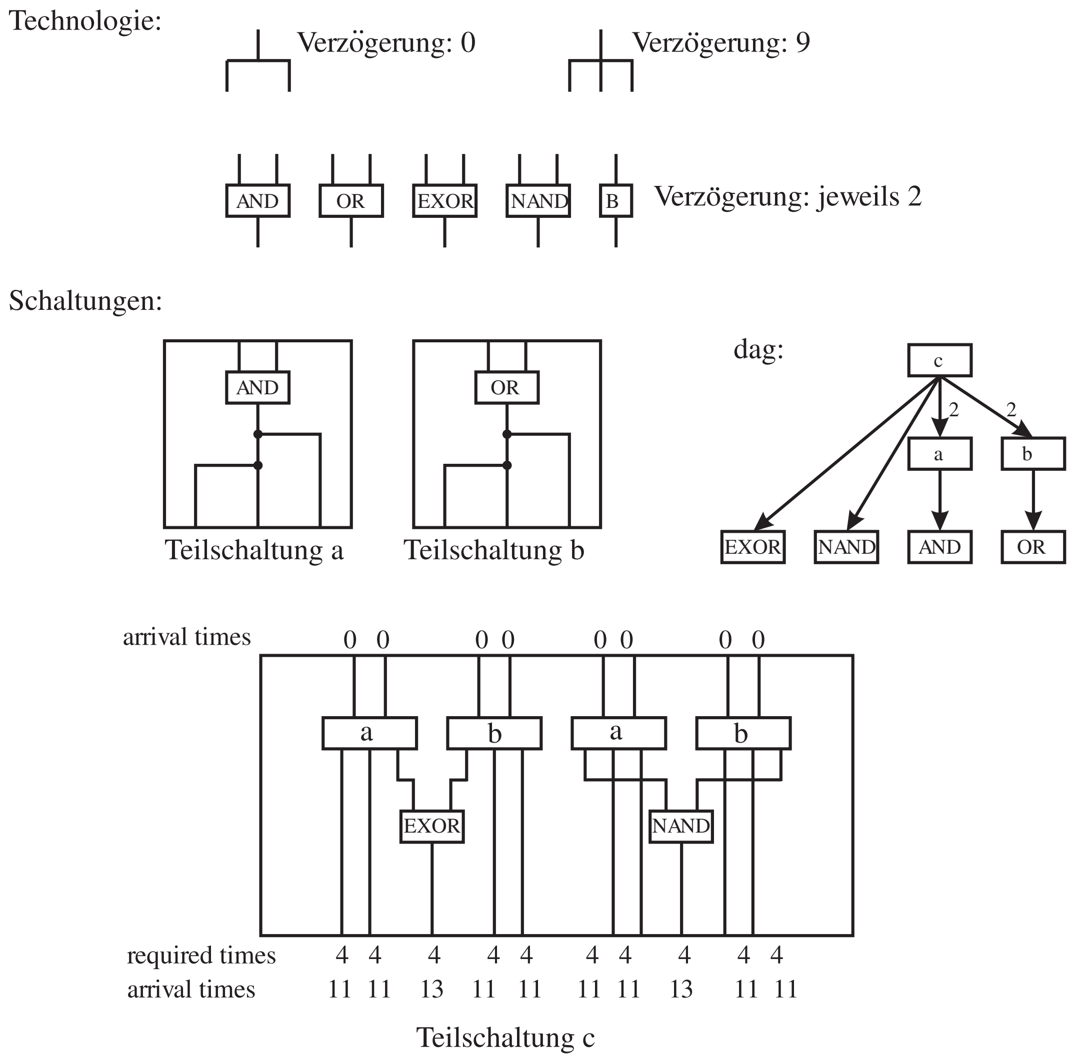

Abbildung 4.15: In diesem Beispiel wird die vollständige Entfaltung erzwungen

der Teilschaltungen a und b gab es unterschiedliche kritische Pfade, denn das kritischste Signal, das durch die Minimumbildung in Formel 3.4 bestimmt wurde, war immer unterschiedlich.

Das Beispiel der Abbildungen 4.15 und 4.16 kann verallgemeinert werden, indem man die Anzahl der Ausgangsanschlüsse der Teilschaltung a oder b beliebig hoch setzt und diese modifizierten Teilschaltungen in jeweils unterschiedlichem Kontext in Teilschaltung c als Makro einsetzt. Dann muß für jedes Vorkommen von a und b eine Ersatzdarstellung gefunden werden. Diese Verallgemeinerung beweist Satz 4.13, da die Entfaltung für die flache Schaltung nicht nötig ist und da gilt: Die Komplexität für die Analyse der Signallaufzeiten und für die Herstellung der Varianten ist für den flachen Fall und den hierarchischen

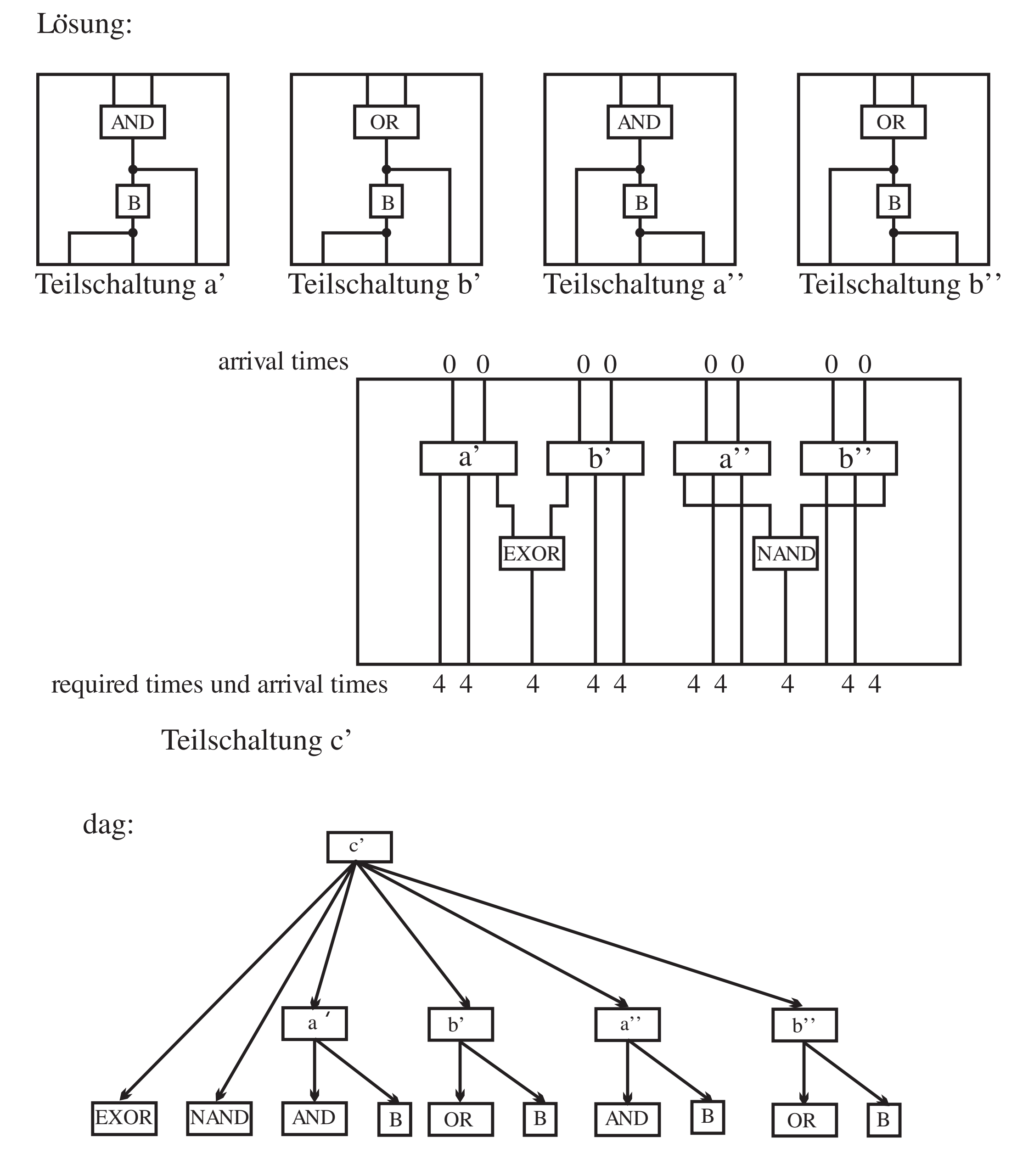

Abbildung 4.16: Die einzig mögliche Lösung

Fall - unter Vernachlässigung von Konstanten - gleich, da beide alle Bausteine und alle Fanout–Knoten für die Analyse miteinbeziehen müssen.

Gegen obige Vorgehensweise ist folgender Einwand zulässig: Die Beispielschaltung wurde speziell für den Beweis konstruiert, ist also künstlich.

Eine Folgerung, die man aus diesen Überlegungen für die Entwicklung von hierarchischen Fanout–Algorithmen ziehen kann, lautet: Es ist im allgemeinen - auch für reale Schaltungen und selbst, wenn die Signalflußrichtungen eindeutig bestimmt sind - nicht möglich, das hierarchische Fanout–Problem ohne die Einführung von Varianten zu lösen.

4.3.3 Eine Idee für einen hierarchischen Fanout–Algorithmus

In diesem Abschnitt werden die grundlegenden Probleme bei der Entwicklung von Fanout–Algorithmen für die gefaltete hierarchische Darstellung kurz angesprochen. Die vorgestellte Idee für einen hierarchischen Fanout–Algorithmus wird skizziert.

Zunächst betrachten wir die Zeitanalyse. Herr Christoph Schmitz hat im Rahmen eines Fortgeschrittenenpraktikums einen Algorithmus für die Signallaufzeitanalyse in CADIC gefunden und implementiert [Sch92]. Der Algorithmus wurde für hierarchische Schaltungen der Definitionen 4.1, 4.2 und 4.4 auf der logischen Entwurfsebene entwickelt. Das Zeitanalysemodell aus [SSV90] kann einfach auf die Signallaufzeitanalyse in [Sch92] übertragen werden (es wird jedoch für jeden Baustein ein von den Eingängen unabhängiges, einheitliches extrinsic delay angenommen). Wir skizzieren das Verfahren zur Berechnung der Pfadlängen in Teilschaltungen anhand eines Beispiels aus [Sch92].

Abbildung 4.17 zeigt, wie eine Teilschaltung auf unterster Hierarchieebene als sogenannter Signalgraph modelliert wird. Die Beschriftung enthält die zur Verzögerungsberechnung relevanten Daten: die Kapazität wird den Netzen zugeordnet, die im Signalgraphen durch Knoten dargestellt sind. Die Kanten repräsentieren das zeitliche Verhalten zwischen zwei Netzen (in der Kurzform des Signalgraphen: die Verbindung zweier Netze durch einen Pfad - siehe unten). Den Kanten wird die interne Verzögerung und die Ladeabhängigkeit (extrinsic delay) zugeordnet. In Abbildung 4.18 wird die Teilschaltung B vorgestellt, die zwei Vorkommen der Teilschaltung A enthält. Der entsprechende Signalgraph besitzt noch sogenannte Nullkanten (gestrichelt dargestellt), die durch das Zusammenschalten an Makroschnittstellen und am Rand entstehen. Diese Nullkanten können eliminiert werden, und man erhält den linken Signalgraphen der Abbildung 4.19. Dieser kann wiederum vereinfacht werden, da man Pfade, die nur über innere Knoten (außer dem Start- und Endknoten) laufen, als eine Kante darstellen kann. Betrachten wir beispielsweise die Kante zwischen dem linken oberen Randknoten und dem linken unteren Randknoten in der rechten Skizze von Abbildung 4.19. In der linken Skizze der Abbildung 4.19 kann man zwei verschiedene Pfade sehen, die zwischen diesen beiden Knoten verlaufen. Der längere der beiden Pfade ist derjenige, der vier Knoten enthält, da er eine höhere interne Verzögerung hat. Die interne Verzögerung 6 berechnet sich für diesen Pfad, indem dreimal die interne Verzögerung von 1 zu den externen Verzögerungen 1*2 (für den zweiten Knoten auf dem Pfad) und 1*1 (für den dritten Knoten auf dem Pfad) addiert wird. Die 1 als externe Verzögerung wird von der letzten Kante auf dem Pfad übernommen. Der Algorithmus [Sch92] geht nicht in der eben gezeigten Weise vor, sondern berechnet die Kurzform mit Hilfe einer rekursiven Routine, auf die hier nicht näher eingegangen wird. Die Kurzform der Signalgraphen läßt sich in einem bottom–up Lauf über alle Teilschaltungen berechnen

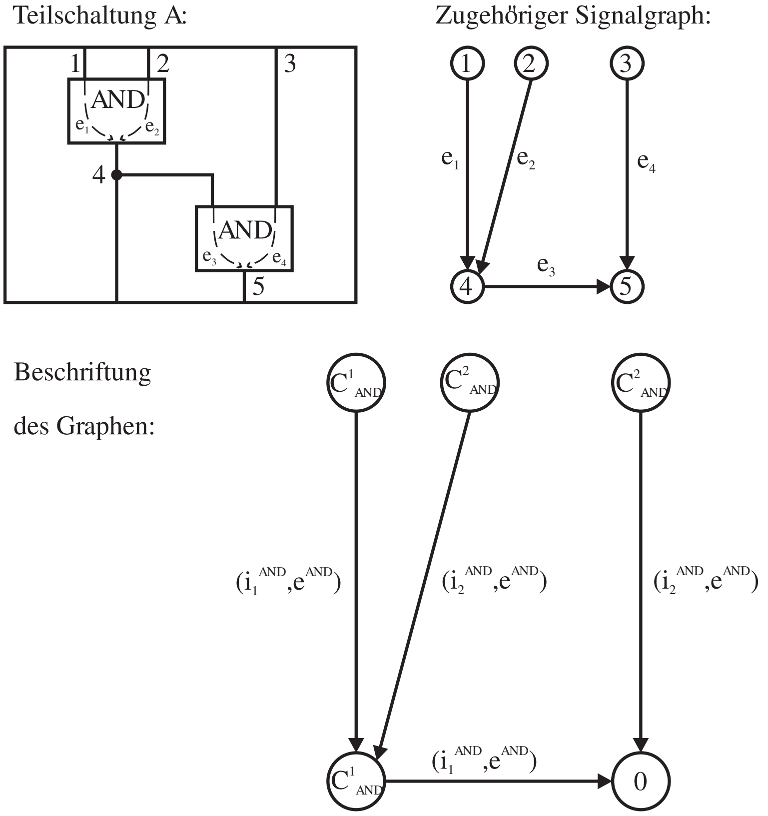

Abbildung 4.17: Die Teilschaltung A und ihr Signalgraph

und in einem Archiv ablegen. Anhand dieses Archivs kann ein Algorithmus die längsten Pfade berechnen. Wie man an beiden Teilschaltungen des Beispiels sieht, werden Pfade über unbekannte Kapazitäten (diese kommen in den Teilschaltungen jeweils am linken Anschluß des südlichen Randes vor) erst berechnet, wenn die Teilschaltungen in einen Kontext eingesetzt wurden, in dem die Kapazitäten bekannt werden (vergleiche hierzu die unterschiedliche Behandlung des linken südlichen Anschlusses der Teilschaltung A bei der Einsetzung in die Teilschaltung B).

Mit diesem Verfahren lassen sich arrival times, required times und Pfade mit Schlupfwerten innerhalb eines Intervalls (zwischen S_{min} und $S_{min}+\epsilon$ - vergleiche [SSV90]) berechnen. Möchte man nun eine Fanout–Korrektur auf der hierarchisch spezifizierten, gefalteten Schaltung durchführen, dann muß man folgendes beachten: Ein kritischer Pfad des ϵ–Netzwerkes erstreckt sich von einem primären Eingang zu einem primären Ausgang. Der Pfad geht im allgemeinen durch mehrere Teilschaltungen, d.h. mehrere Knoten des dag. Für die Fanout–Korrektur wird das Zeitverhalten an einer oder mehreren Stellen des Pfades verändert. Man kann für diese Stellen Ersatzbausteine suchen („repower“) oder an die Ausgänge Treiberbäume hängen. Für letztere Operation bieten sich Knoten auf dem

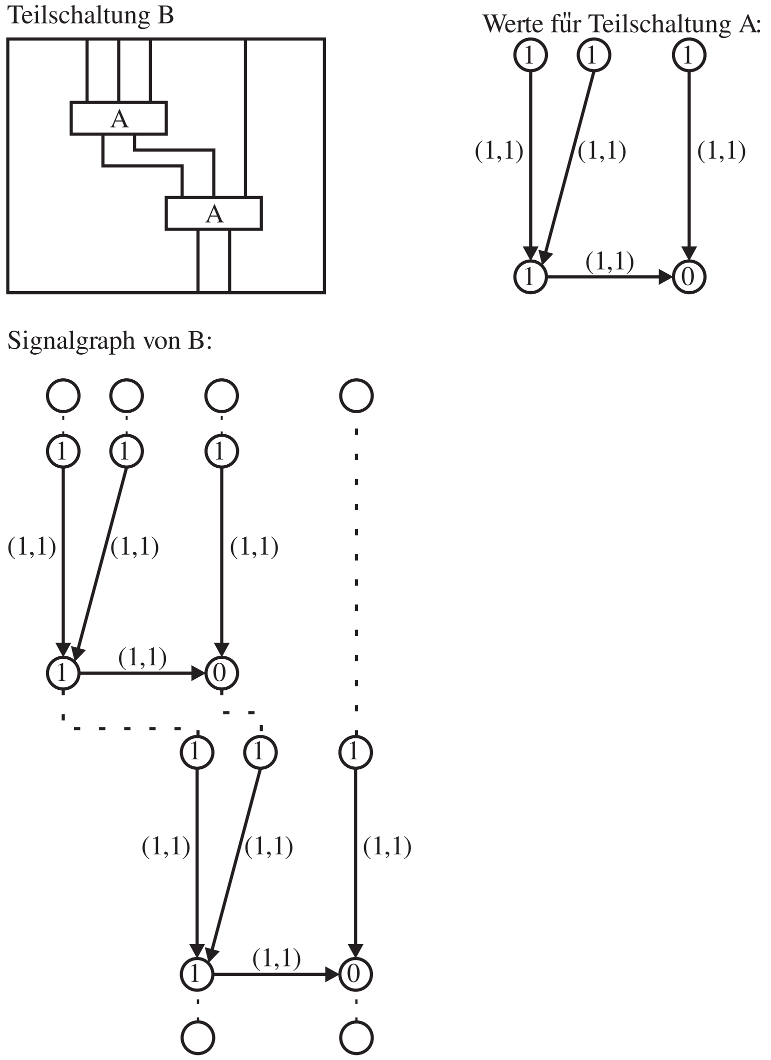

Abbildung 4.18: Die Teilschaltung B und die Langform ihres Signalgraphen ohne Nullkantenelimination

Pfad an, die einen hohen Ausgangsgrad haben (vergleiche [SSV90]). Für beide Operationen muß man jedoch die im letzten Abschnitt gefundene Regel über Veränderungen in hierarchischen Schaltungen beachten: Um das Fanout–Problem auf hierarchisch spezifizierten Schaltungen zu lösen, muß die Schaltung im allgemeinen teilentfaltet werden, denn ein Baustein hat in der gefalteten hierarchischen Darstellung im allgemeinen mehrere Schlupfwerte, da er im allgemeinen an mehreren Stellen in der flachen Schaltung vorkommt. Führt man die Modifikation in einer Teilschaltung aus, ohne die hierarchische Struktur der Schaltung zu verändern, so wird die Veränderung für sämtliche Vorkommen der Teilschaltung in der flachen Schaltung durchgeführt (der Repräsentant wurde

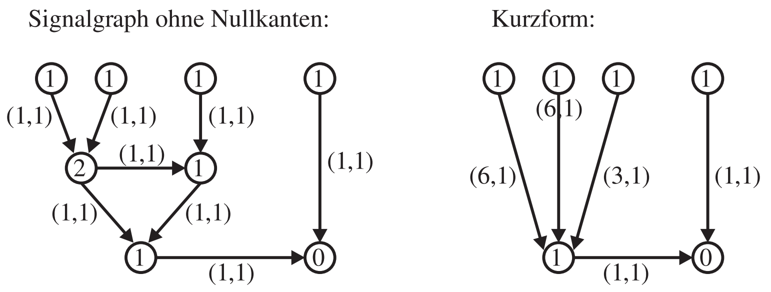

Abbildung 4.19: Die Teilschaltung B und die Langform ohne Nullkanten und schließlich die Kurzform

verändert und damit alle flachen Vorkommen). Wenn man in dieser Weise vorgeht, dann kann es sein,

- daß vormals unkritische Pfade, die durch diese Teilschaltung verlaufen, nun kritisch werden.
- daß der Ersatzbaustein beziehungsweise der eingesetzte Treiberbaum für die übrigen Vorkommen der Teilschaltung überdimensioniert ist, was zu erhöhten Kosten führt.

Wie im Abschnitt „Eindeutige Signalflußrichtungen“ gesehen, werden Teilentfaltungen in den Fällen erzwungen, bei denen Treiber an Stellen eingesetzt werden sollen, wo die Signalflußrichtung nicht eindeutig ist.

Nach einer Teilentfaltung ist die Schaltung weniger kompakt, der Platzverbrauch steigt. Da es mehr Knoten im dag gibt, sinkt die Ausführungsgeschwindigkeit der hierarchischen Algorithmen und für den menschlichen Entwerfer wird die Schaltung unübersichtlicher. Wenn stark teilentfaltet wird, so daß fast die flache Darstellung erreicht ist, dann hat ein hierarchischer Algorithmus keine Vorteile gegenüber einem, der auf flachen Schaltungen läuft.

Es gibt also zwei gegenläufige Tendenzen: Einerseits wirkt sich die Entfaltung im allgemeinen positiv auf die Güte der Lösung des Fanout–Problems aus, andererseits sollte man bestrebt sein, die Entfaltung auf ein Mindestmaß zu begrenzen, um die Vorteile der hierarchischen Darstellung möglichst beizubehalten.

Da ein kritischer Pfad sich von einem primären Eingang zu einem primären Ausgang erstreckt und da im allgemeinen viele Bausteine auf dem Pfad einen hohen Ausgangsgrad haben, kann man von mehreren Bausteinen einige auswählen, für die eine Fanout–Korrektur durchgeführt werden soll. Zuerst sollte der Algorithmus jedoch prüfen, ob nicht bereits eine Variante einer der betroffenen Teilschaltungen existiert, die das gegebene Problem löst. In diesem Fall muß die Hierarchie lediglich durch das Umsetzen eines Verweises geändert werden. Andernfalls bieten sich zur Auswahl des Bausteins zwei Kriterien an:

- Man versucht einen Baustein zu finden, bei dem die Korrektur am Ausgang die gewünschte Laufzeitbeschleunigung des Pfades erreicht, oder, falls dies nicht möglich ist, eine möglichst gute.
- Man nimmt einen Baustein, dessen Ausgangsignal sich möglichst wenig (am besten gar nicht) auf andere Teilschaltungen und Makros verteilt, da man dann voraussichtlich weniger teilentfalten muß.

Wie diese beide Kriterien sinnvoll gewichtet werden, müßte man anhand von Tests herausfinden.
Eine Teilentfaltung kann effizienter durchgeführt werden, als nach den Vorschriften der Definitionen 4.3 und 4.4.

Entwickelt man aus obigen Betrachtungen einen Algorithmus, so könnte dieser wie folgt aussehen: Eingabe für den Algorithmus ist die gefaltete Schaltung zusammen mit der zugrundeliegenden Technologie sowie spezifizierten arrival times an den primären Eingängen und spezifizierten required times an den primären Ausgängen. Folgende Schritte werden solange wiederholt, bis die Schranke erreicht ist, oder bis keine Verbesserung mehr möglich ist:

1. Berechne mit Hilfe der Signallaufzeitanalyse [Sch92] ein ϵ–Netzwerk.
2. Wähle für jeden Pfad einen Baustein mit hohem Ausgangsgrad, und zwar den nach obigen Kriterien besten. Führe eine Fanout–Korrektur aus und erzeuge dabei gegebenenfalls Varianten.

Ob es sinnvoll ist, die Anzahl der generierten Varianten nach oben zu beschränken, müßte ebenfalls anhand von Messungen geprüft werden.

4.4 Ein hierarchischer Max–Fanout–Algorithmus

In diesem Abschnitt wird ein hierarchischer Max–Fanout–Algorithmus vorgestellt. Falls für alle Vorkommen jeder Teilschaltung sämtliche Signalflußrichtungen der Verdrahtung identisch sind, dann bleibt die kompakte gefaltete Darstellung der Schaltung auch nach Ausführung des Algorithmus bestehen.

4.4.1 Einschränkung: Erzwungene Teilentfaltung

Wie im Abschnitt „Eindeutige Signalflußrichtungen“ bereits gesehen, werden Teilentfaltungen in den Fällen erzwungen, bei denen Treiber an Stellen eingesetzt werden sollen, wo die Signalflußrichtung nicht eindeutig ist. Abbildung 4.20 zeigt hierzu ein Beispiel. Diese Schaltung muß teilentfaltet werden, um das Max–Fanout–Problem lösen zu können (beziehungsweise: es muß vorher ein Verfahren angewendet werden, daß die Signalflußrichtungen eindeutig macht).

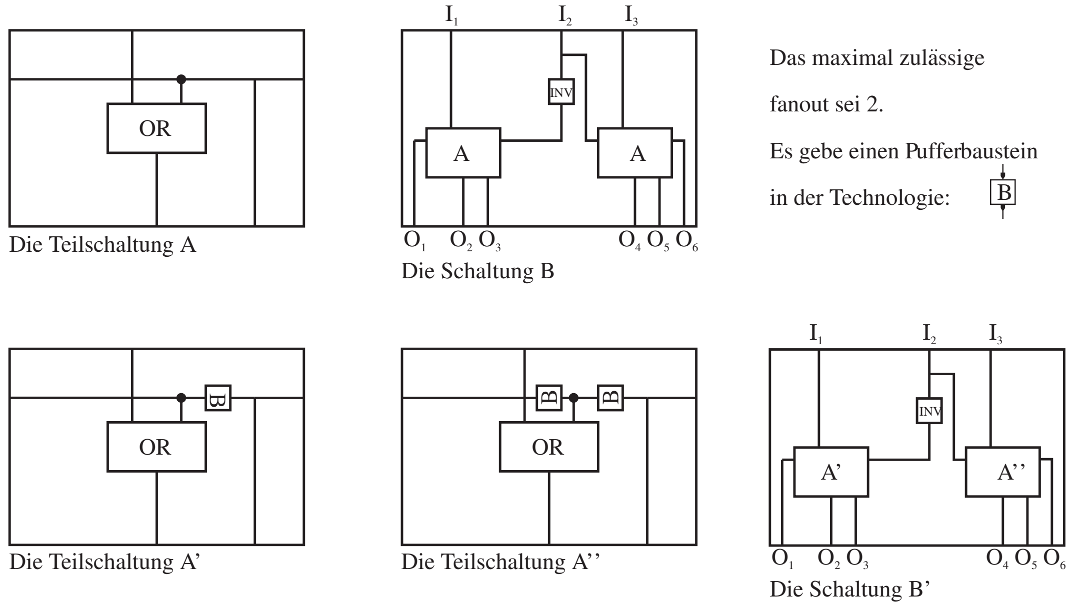

Abbildung 4.20: Zur Lösung des Max–Fanout–Problems muß B entfaltet werden

4.4.2 Der Algorithmus

Der Max–Fanout–Algorithmus kann nur aufgerufen werden, wenn gilt:

- Sämtliche Signalflußrichtungen auf allen Teilschaltungen sind eindeutig bestimmbar.
- Die primären Ausgänge der Schaltung werden bezüglich des Treiberverhaltens wie Eingänge von Basiszellen behandelt.
- Es gibt keine Projektionen.

Mit Hilfe der letzten beiden Voraussetzungen läßt sich der Algorithmus eleganter formulieren und beweisen. Wenn man den Algorithmus entsprechend abändert, dann können diese Voraussetzungen auch fallen gelassen werden. Beispielsweise sind in der Implementierung Projektionen erlaubt.
Die Schaltung liege in einer der drei Darstellungen der Definitionen 4.1, 4.2 und 4.4 vor, und die folgenden Vorberechnungen seien durchgeführt:

- Die Signalflußrichtungen seien berechnet.
- Die erweiterte Netzliste sei berechnet.

Ein Knoten des dag soll zwei mögliche Zustände haben können: den „unkorrigierten“ und den „korrigierten“ Zustand. Wenn ein Knoten im „korrigierten Zustand“ ist, dann erfüllen er und seine Nachfahren die Max–Fanout–Bedingung. Ein Knoten soll nur dann vom unkorrigierten zum korrigierten Zustand wechseln dürfen, wenn seine Nachfahren im dag im korrigierten Zustand sind.

Die Basiszellen sind initial im korrigierten Zustand. Der dag wird für die Korrektur mit *DagDFS* bottom–up durchmustert. Für die Berechnung des korrigierten Zustands verwendet der Algorithmus keine Informationen über Teilschaltungen in denen die zu korrigierende Teilschaltung eingesetzt wird, es handelt sich also um ein kontextfreies Verfahren. Die Routinen, die einen Knoten in den korrigierten Zustand überführen, werden im folgenden beschrieben. Um simultan die Korrektheit der Vorgehensweise zu zeigen, wird die Beschreibung mit dem Beweis verflochten.

Jedem erweiterten Netz eines Knotens, das mindestens einen Anschluß am Rand besitzt, wird ein Paar aus $\mathbb{N}_0^2$ zugeordnet: Die zweite Zahl des Paares ist die Anzahl der Ausgangspads des erweiterten Netzes am Rand seiner Teilschaltung.
Die erste Zahl enthält folgenden Wert: Angenommen, die aktuelle Teilschaltung liege flach vor. Dann entspricht dem erweiterten Netz ein flaches Netz. Die erste Zahl gibt an, wie viele Eingangspins (an Basiszellen) dieses flache Netz enthält. Diese Zahl wird vom Algorithmus berechnet, ohne daß die Teilschaltung flach vorliegt: Mit Hilfe der Durchmusterungsstrategie kann die Zahl bottom–up akkumuliert werden.
Die Summe der beiden Zahlen des Paares wird im folgenden mit S bezeichnet.

Der Algorithmus durchmustert den dag in einem Pass nach der bottom–up depth–first Strategie. Einmal besuchte Teilschaltungen können also nicht nicht wieder modifiziert werden. Wäre nach dem Besuch einer Teilschaltung der Wert von S für ein erweitertes Netz größer als der Max–Fanout Wert, dann wäre diese Überschreitung des zulässigen Wertes nicht mehr korrigierbar (dies ist eine Überschreitung, da in einer höheren Hierarchiestufe die Anschlüsse am Rand entweder mit Eingängen von Basiszellen verbunden werden, oder mit primären Ausgängen).

Induktionsanfang
Für den Induktionsanfang wird eine Routine für die Knoten des dag angegeben, die Teilschaltungen repräsentieren, in denen sich keine Makros befinden. Sie bestehen entweder aus Basiszellen, Anschlüssen und Verdrahtung oder aus Anschlüssen am Rand und Verdrahtung. Da es keine Makros gibt, sind die erweiterten Netze dieser Teilschaltungen identisch mit den Netzen. Es wurde vorausgesetzt, daß die Signalflußrichtungen festgelegt sind, deswegen besitzt jedes Netz eine Quelle. Die Quelle ist entweder ein Eingangspad oder ein Ausgang einer Basiszelle. Alle übrigen Anschlüsse des Netzes werden als Senken bezeichnet: Eingänge von Basiszellen und Ausgangspads. Das Fanout dieses Netzes sei die Anzahl der Senken. Überschreitet das Fanout eines Netzes den spezifizierten Max–Fanout–Wert, so kann man durch folgende Strategie Netze erhalten, die den Max–Fanout–Wert schließlich einhalten: Man setzt Treiber ein, die ein Netz in mehrere Netze mit jeweils kleinerem Fanout zerfallen lassen. Dies geschieht so lange, bis die Fanout–Bedingung erfüllt

ist. In [HKP84] wird ein effizienter Algorithmus mit guten Resultaten zur Lösung dieses Problems beschrieben. Ein Netz, welches Anschlüsse am Rand hat, exportiert das oben spezifizierte Paar: die Anzahl der Eingänge der Basiszellen und die Anzahl der Ausgangspads dieser Teilschaltung. Der soeben beschriebene Algorithmus sorgt dafür, daß der S–Wert des Paares jedes Netzes nicht größer als der Max–Fanout–Wert ist.

Induktion
Im Induktionsschritt werden Teilschaltungen betrachtet, die Makros enthalten. Wie bereits erwähnt, wird die Korrektur (d.h. der Induktionsschritt) nur für Knoten durchgeführt, deren Nachfahren im dag (also insbesondere die Teilschaltungen der Makrovorkommen) bereits korrigiert sind.

Nach *Induktionsvoraussetzung* gilt:
Jedem erweiterten Netz der korrigierten Knoten ist das oben beschriebene Paar (B, A) mit folgenden Eigenschaften zugeordnet: B ist die über die Hierarchie akkumulierte Anzahl der Eingänge der Basiszellen des erweiterten Netzes und A die Anzahl der Ausgangspads des erweiterten Netzes. Der S–Wert des Paares (d.h. $B + A$) ist nicht größer als der Max–Fanout–Wert. Weiter gilt: Wandelt man einen korrigierten Knoten in eine flache Schaltung um, dann wird von keinem Netz dieser Schaltung der Max–Fanout–Wert überschritten.

Der *Induktionsschritt*:
In einer Schleife läuft man über alle erweiterten Netze der Teilschaltung:
Für das aktuelle, erweiterte Netz N baut man einen Baum B auf:

- Knoten des Baumes sind die einzelnen Netze. Da die Signalflußrichtung eindeutig ist, hat jedes Netz genau eine Quelle. Die Quelle besteht aus einem der drei folgenden Typen:

  1. ein Eingangspad,
  2. ein Ausgangspin einer Basiszelle oder
  3. ein Ausgangspin einer Makrozelle.

  Die Beschriftung der Knoten wird weiter unten vorgestellt.

- Die Wurzel des Baumes ist der Knoten, der das Netz repräsentiert, das einen Eingangspad oder einen Ausgangspin einer Basiszelle enthält. Befindet sich in keinem der Netze ein solcher Anschluß, dann wird eine imaginäre Wurzel erzeugt.

- Die Kanten des Baumes werden durch die Makros gewonnen: Wenn das Netz X mit dem Netz Y durch ein erweitertes Netz innerhalb des Makros M verbunden ist und die Signalflußrichtung von X über M zu Y verläuft, so existiert in B eine gerichtete Kante von X nach Y. Falls die Knotenmenge eine imaginäre Wurzel enthält, dann gibt es ein Makro, das, als flache Schaltung dargestellt, einen Ausgang einer Basiszelle enthält, der mit einem oder mehreren Netzen aus N verbunden ist. Im

Baum gibt es dann eine Kante von der imaginären Wurzel zu jedem dieser Netze. Das erweiterte Netz im Makro, das den Ausgang der Basiszelle enthält, heisse rn.

Jede Kante wird mit dem von ihrem erweiterten Netz exportierten Paar (B_K, A_K) beschriftet (das K im Index steht für Kante).

Der Baum B wird mit Hilfe einer Postorder–Depth–First–Strategie durchmustert. Damit ist es möglich, Informationen von den Blättern zur Wurzel wandern (und auch akkumulieren) zu lassen. Anhand der Beschriftung an den Knoten (d.h. an den Netzen) läßt sich feststellen, ob und in welcher Art die Max–Fanout-Bedingung verletzt wird. Ist eine Bedingung verletzt, dann wird das entsprechende Netz durch das Einsetzen von Treibern modifiziert. Dabei entstehen im allgemeinen neue Netze sowie neue erweiterte Netze. Es gibt also neue Bäume, die durchlaufen werden müssen, und der aktuell behandelte Baum B wird verändert. Erst nachdem die Durchmusterungsreihenfolge des Baumes B aktualisiert wurde und alle seine Knoten besucht wurden, werden die neuen erweiterten Netze (d.h. die neuen Bäume) durchmustert.

Die Knoten aus B sind mit dem Quintupel $(B_N, A_N, M_N, B_G, A_G)$ beschriftet. Dabei bedeutet:

- B_N: Die Anzahl der Eingänge von Basiszellen, die Elemente des Netzes sind.
- A_N: Die Anzahl der Ausgangspads des Netzes.
- M_N: Die Anzahl der Eingänge von Makrozellen des Netzes.
- B_G: Die Anzahl der Basiszellen, die „geerbt“ wurden (siehe unten).
- A_G: Die Anzahl der Ausgangspads, die „geerbt“ wurden (siehe unten).

Für die imaginäre Wurzel gilt: $B_N = A_N = M_N = 0$.

Die Berechnung von B_G und A_G: Für die Blätter des Baumes B haben B_G und A_G den Wert 0. Wenn der aktuelle Knoten x ein innerer Knoten oder die Wurzel des Baumes B ist, dann sei FO^x die Menge der Kinder des Knotens. FO^x ist also die Menge der Netze, die Ziel einer von x startenden Kante sind. Die B_G und A_G Werte der Knoten der Menge FO^x sind berechnet (dies wird durch die Durchmusterungsstrategie sichergestellt). Für die Menge NE^x werden jeweils Kanten, die von x aus starten und durch das selbe erweiterte hierarchische Netz eines Makros laufen, zusammengefaßt: NE^x ist die Menge der erweiterten Netze der Makros, aus denen eine oder mehrere von x aus startende Kanten gebildet wurden (Beachte: Es kann in der aktuellen Teilschaltung mehrere Makros einer definierenden Teilschaltung geben. In der Menge NE sollen die erweiterten Netze dieser Makros voneinander unterschieden sein, beispielsweise indem jedes erweiterte Netz einen Index erhält, der sein Makrovorkommen wiederspiegelt). Falls x die imaginäre Wurzel ist, dann enthält NE^x ein Element: das erweiterte Netz rn.

Die Variablen B_G und A_G werden für das Netz x wie folgt berechnet:

$$B_G^x = \sum_{j \in NE^x} (B_K^j) + \sum_{i \in FO^x} (B_N^i + B_G^i)$$

$$A_G^x = \sum_{i \in FO^x} (A_N^i + A_G^i)$$

Im folgenden sind die Fälle aufgelistet, die die Modifikation eines Netzes indizieren. Anschließend wird ein Algorithmus angegeben, der die Modifikationen durchführt. Ein Ergebnis jeder Modifikation lautet: Jedes nach der Modifikation neu entstandene Netz erfüllt nicht mehr die Bedingung, die die Modifikation erzwang.

1. $B_N + A_N + M_N >$ Max–Fanout: In diesem Fall verzweigt sich das betrachtete Netz zu häufig.

2. $B_N + A_N + B_G + A_G >$ Max–Fanout und $B_N + A_N + M_N \leq$ Max–Fanout: In diesem Fall wird von den Makros ein zu hohes Fanout „geerbt".

3. Für diesen Fall muß eines der beiden folgenden Ereignisse beim Durchlaufen der dfs–Strategie eintreten:

   - Eine Prüfung auf diesen Fall erfolgt, nachdem alle Sohnknoten eines Knotens, im folgenden fa genannt, besucht wurden. Man betrachtet nun alle Elemente aus NE^{fa}, d.h. die erweiterten Netze in Makros, die über einen Eingangspin mit fa verbunden sind. Für jedes Element $ne \in NE^{fa}$ gibt es wenigstens eine Kante im Baum, die bei fa startet. Diese Kanten sollen sich in der Menge E_{ne}^{fa} befinden. FO_{ne}^{fa} sei die Menge der Netze, die die Ziele der Kanten aus E_{ne}^{fa} sind. Für alle Netze aus E_{ne}^{fa} treffen die Fälle 1 und 2 nicht (mehr) zu. Es sei B^{ne} der vom Netz ne exportierte B–Wert. Falls für ne

     $$B^{ne} + \sum_{i \in FO_{ne}^{fa}} (B_N^i + A_N^i + B_G^i + A_G^i) > \text{Max–Fanout}$$

     gilt, dann wird das maximal zulässige Fanout durch die Netze FO_{ne}^{fa} und die Hinzunahme der Eingänge der Basiszellen des erweiterten Netzes ne überschritten.

   - Eine Prüfung auf diesen Fall erfolgt, wenn die imaginäre Wurzel besucht werden soll. Man benutzt die selbe Formel wie im vorangegangenen Fall, setzt aber für ne das erweiterte Netz rn ein und setzt für FO_{ne}^{fa} die Kindernetze des imaginären Netzes im ein. Für alle diese Kinder treffen die Fälle 1 und 2 nicht (mehr) zu. Es wird also geprüft, ob das maximal zulässige Fanout durch

     $$B^{rn} + \sum_{i \in FO_{rn}^{im}} (B_N^i + A_N^i + B_G^i + A_G^i) > \text{Max–Fanout}$$

     überschritten wird.

Falls das Netz x die Wurzel des Baumes ist, der komplette Baum korrigiert wurde und das entsprechende erweiterte Netz wenigstens ein Pad (ein Anschluß am Rand der Teilschaltung) hat, dann wird $B_G^x + B_N^x$ als B–Wert exportiert (Ausnahme: für die imaginäre Wurzel wird $B^{rn} + \sum_{i \in FO_{rn}^{im}}(B_N^i + B_G^i)$ exportiert) und $A_G^x + A_N^x$ als A–Wert exportiert (Ausnahme: für die imaginäre Wurzel wird $\sum_{i \in FO_{rn}^{im}}(A_N^i + A_G^i)$ exportiert).

Es sei $NE_{\mathcal{C}}$ die Menge der erweiterten Netze in der Teilschaltung $\mathcal{C}$. Es folgt der Algorithmus, in dem obige Fälle erkannt werden und die entsprechenden Korrekturstrategien aufgerufen werden:

```
proc modify_circuit(C)
begin
  for all n ∈ NE_C do
      t := generate_tree(NE_C);
      while (∃ node in t which has not yet been visited) do
            n := next_node_in_bottom_up_dfs_order(t);
            if (n fulfills case 1) then execute_strategy_1(n); update_order(t); fi
            if (n fulfills case 2) then execute_strategy_2(n); update_order(t); fi
            if (n fulfills case 3) then execute_strategy_3(n); update_order(t);
            fi
      od
  od
end
```

Nach Anwendung jeder Strategie wird die Routine update_order aufgerufen. In dieser Routine wird zunächst der Baum an den veränderten Stellen entsprechend der neuen Situation modifiziert.
Falls durch die Veränderung der Teilschaltung neue, erweiterte Netze entstanden, werden diese nun der Menge der erweiterten Netze hinzugefügt. Die Durchmusterung wird in der Weise fortgesetzt, daß die modifizierten Knoten als erstes besucht werden.

Als Strategie 1 kann ein Algorithmus wie beim flachen Fall, z.B. [HKP84] benutzt werden. Als Strategie 2 kann folgendes Verfahren eingesetzt werden:

1. Jedem Makroeingang MI des aktuellen Netzes x kann eine oder mehrere startende Kanten zugeordnet werden. Sie werden durch ein erweitertes Netz innerhalb des Makros gebildet, das die Werte (B_{MI}, A_{MI}) exportiert. Die Menge T_{MI}^x enthalte die Zielnetze der Kanten, die am Makroeingang MI starten. Dann kann man jedem solchen Makroeingang MI die Zahl $S_{MI} := B_{MI} + \sum_{i \in T_{MI}^x}(B_N^i + A_N^i + B_G^i + A_G^i)$ zuordnen. Die Makroeingänge des Netzes werden nun absteigend nach der Summe S_{MI} sortiert.

2. Gehe in der absteigenden Sortierreihenfolge vor:
   Setze so lange Treiberstufen (d.h. einen Puffer oder zwei hintereinandergeschaltete Inverter) vor die Eingänge der Makros, bis $B_N^x + A_N^x + B_G^x + A_G^x \leq$Max–Fanout ist.

Folgendes Verfahren zur Korrektur wird bei Strategie 3 angewendet:

1. Sortiere die Elemente aus FO_{rn}^{im} absteigend nach der Summe $B_N^i + A_N^i + B_G^i + A_G^i$
2. Setze so lange Treiberstufen in dieser Reihenfolge hinter die Ausgangsanschlüsse des Makros m, bis gilt:

$$B^{ne} + \sum_{i \in FO_{ne}^{fa}} (B_N^i + A_N^i + B_G^i + A_G^i) \leq \text{Max–Fanout}$$

Der Induktionsschluß ist vollzogen: Strategie 1 garantiert, daß es kein Netz gibt, daß sich zu oft verzweigt. Strategie 2 sorgt dafür, daß ein Netz von seinen Söhnen keinen zu hohen Fanout–Wert „erbt“. Strategie 3 garantiert schließlich, daß interne erweiterte Netze innerhalb eines Makros nicht durch die äußere Beschaltung an ihren Ausgängen ein zu hohes Fanout „erben“. Nach Ausführung des Algorithmus enthalten also alle erweiterten Netze ein maximales Fanout, das höchstens so groß ist, wie Max–Fanout.

Betrachtet man die Teilschaltung, die im dag der hierarchischen Schaltungsbeschreibung die Wurzel darstellt, so gilt: Nachdem sie in den korrigierten Zustand gewechselt hat erfüllen alle ihre erweiterten hierarchischen Netze die Max–Fanout–Bedingung. Damit ist die Korrektheit des Verfahrens bewiesen. ■

Die in Abbildung 4.21 dargestellte Beispielschaltung soll die oben beschriebene Vorgehensweise veranschaulichen. In diesem Beispiel sind alle Netze identisch mit ihren erweiterten Netzen, mit einer Ausnahme: das erweiterte Netz $\{O_2, P_4, P_6, P_10, P_15\}$ der Schaltung M_1 besteht aus den beiden Netzen $\{P_4\}$ und $\{O_2, P_{10}, P_{15}\}$.
Zum Treiben von Signalen soll ein Puffer B zur Verfügung stehen, Max–Fanout habe den Wert 2.
Im dag, der die hierarchische Struktur der Schaltung beschreibt, wird bottom–up vorgegangen: Es wird zunächst die Schaltung M_2 betrachtet. Es gibt nur ein Netz, für das das Fanout korrigiert werden muß: $\{I_1, O_1, P_1, P_4\}$. Fall 1 trifft zu, da $B_N + A_N = 2 + 1 = 3$, also wird Strategie 1 angewendet. Das Ergebnis ist die Schaltung M'_2, die in Abbildung 4.22 wiedergegeben ist. Durch die Modifikation haben sich die Netze geändert. Das Netz $\{I_1, O_1, P_1, P_4\}$ existiert nicht mehr, dafür gibt es die beiden Netze $\{I_1, P_1, P_7\}$ und $\{O_1, P_4, P_8\}$. Statt M_2 wird M'_2 in M_1 eingesetzt, man beachte, daß nun alle erweiterten Netze in M_1 identisch mit den Netzen sind.

Die Teilschaltung M_2 wurde zu M'_2, welche die Max–Fanout Bedingung erfüllt. Nun wird die nächste Schaltung in der Durchmusterungsreihenfolge des dag behandelt: M_1. Der Algorithmus wählt (zufällig) $\{O_5, P_5, P_7, P_8\}$ als erstes Netz. Auch hier trifft Fall 1 zu, das Ergebnis der Modifikation ist die Schaltung M'_1. Wie bereits erwähnt, überführt jede

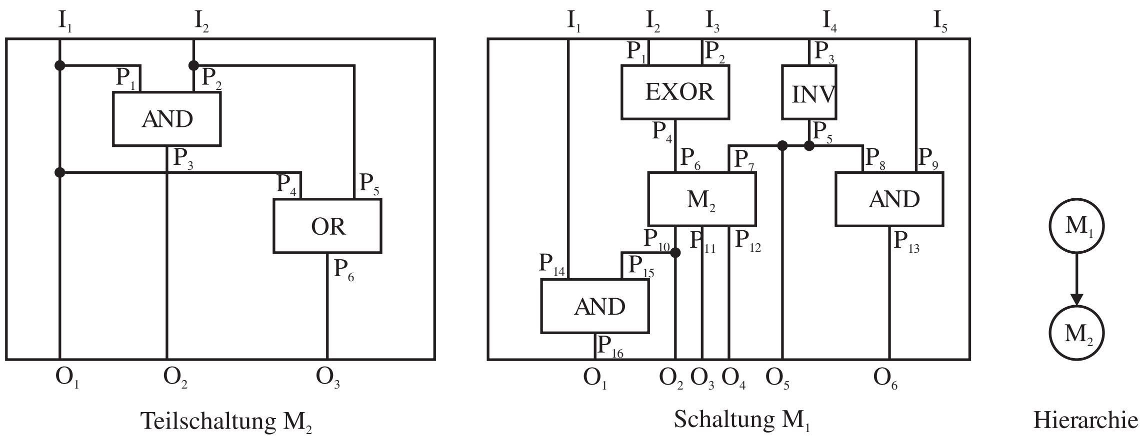

Abbildung 4.21: Beispielschaltung für den Max–Fanout–Algorithmus

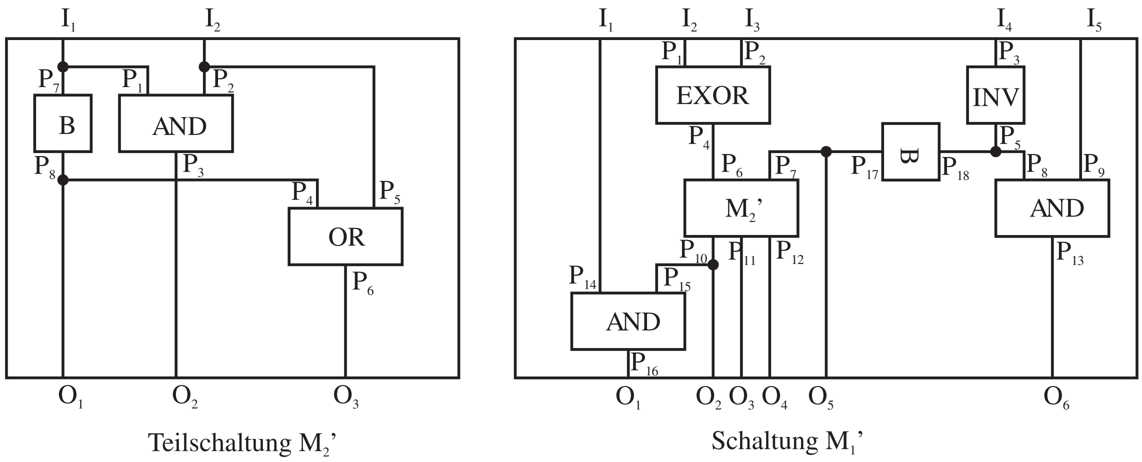

Abbildung 4.22: Die Modifikation von M_2 und eine erste Modifikation von M_1.

Strategie das von ihr behandelte erweiterte Netz in erweiterte Netze, die die Bedingung nicht erfüllen, die zur Anwendung der Strategie führte. Entsprechend trifft die Bedingung von Fall 1 für die beiden neuen Netze $\{P_5, P_8, P_{18}\}$ und $\{O_5, P_7, P_{17}\}$ nicht zu. Jedoch kann die Bedingung eines höheren Falles (hier: 2 oder 3) auf die Netze zutreffen: Für das Netz $\{O_5, P_7, P_{17}\}$ trifft die Bedingung von Fall 2 zu, denn $B_N + A_N + B_G + A_G = 0+1+2+0 = 3$. Die Schaltung nach der Korrektur durch Strategie 2 ist die linke der Abbildung 4.23, und sie heißt M''_1.

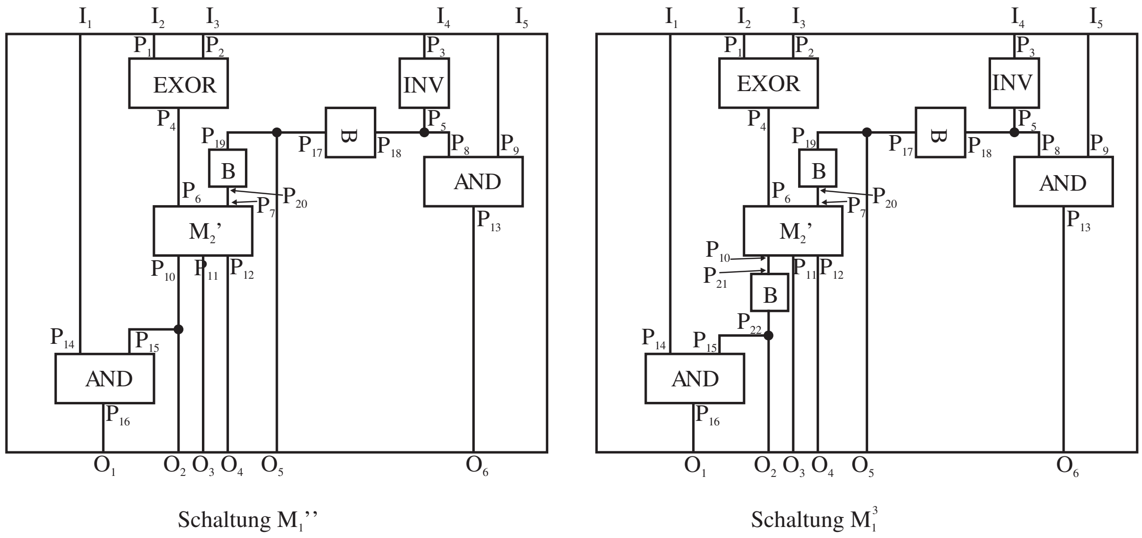

Abbildung 4.23: Nach Anwendung der Strategien 2 und 3

Wird in dieser Schaltung das Netz $\{O_2, P_{10}, P_{15}\}$ behandelt, so stellt der Algorithmus fest, daß für dieses Netz Fall 3 zutrifft:

$$B^{rn} + \sum_{i \in FO_{rn}^{im}} (B_N^i + A_N^i + B_G^i + A_G^i) = 3$$

Der Wert von B^{rn} ist 1 (der linke Eingang des OR–Gatters in M_2'), ebenfalls B^N und A_N haben den Wert 1. Entsprechend der Strategie 3 wird ein Treiber hinter den Ausgang P_{10} von M'_2 gesetzt. Das Ergebnis ist die rechte Schaltung in Abbildung 4.23: M_2^3. Sie erfüllt das spezifizierte Max–Fanout von 2.

4.4.3 Analyse und Resultate

4.4.3.1 Die Laufzeit

Die gefaltete hierarchische Beschreibung kann sehr kompakt sein und für bestimmte hierarchische Algorithmen und für bestimmte Schaltungen extreme Laufzeitvorteile gegenüber flachen Algorithmen haben. Andererseits ist es möglich, daß hierarchisch spezifizierte Schaltungen mehr Aufwand bedeuten können. Mehr Zeitaufwand hat man beispielsweise in obigem Max–Fanout Algorithmus gegenüber dem flachen Fall, wenn man eine flache Schaltung zwar hierarchisch darstellt, jedoch nicht faltet und die Partitionierung so erzeugt, daß viele Fanout–Knoten eines Netzes sich vielen verschiedenen Teilschaltungen befinden. Anhand dieses Beispiels kann man weiter sehen, daß die worst–case Laufzeit eines flachen Entwurfs, der hierarchisch dargestellt wird, durch beliebige Partitionierung beliebig hoch sein kann. Aus diesem Grund wird am Ende der folgenden Laufzeitbetrachtung ein grobes Maß für die Bewertung der Kompaktheit der hierarchischen Darstellung gegenüber der flachen Darstellung vorgestellt. Dieses Maß kann Anhaltspunkte zum Laufzeitvergleich zwischen den beiden Darstellungen geben.

Eine Laufzeitanalyse für den Algorithmus, der eindeutige Signalflußrichtungen erzeugt.

Es sei die Schaltung S in gefalteter hierarchischer Darstellung gegeben. Es sei T die Menge der definierenden Vorkommen von Teilschaltungen. Für die Teilschaltung $t \in T$ wird definiert:
$A_t :=$ Anzahl der Anschlüsse in t.
Es sei $A_T := \sum_{i \in T} A_i$
$S_t :=$ Anzahl der Leitungssegmente in t.
Es sei $S_T := \sum_{i \in T} S_i$
$R_t :=$ Anzahl der Anschlüsse am Rand von t.
$M_t :=$ Menge der Makros in t.
Die Makros der Menge M_t seien von 1 bis $|M_t|$ durchnummeriert. M_t^j bezeichne das Makro mit der Nummer j in t.
Weiter bezeichnet N_t^j die Anzahl der Netze, die wenigstens einen Pin mit dem Makro t gemeinsam haben.
Schließlich gebe V_t an, wie oft die Teilschaltung t als Makro in anderen Teilschaltungen in der gefalteten Darstellung vorkommt.

1. Der Aufwand zur Netzlistenberechnung
   (a) Alle Netze der Teilschaltungen können in Zeit $A_T + S_T$ berechnet werden.
   (b) Um die erweiterten Netze zu berechnen, werden die bekannten Netze über die Makros miteinander vereinigt: $\sum_{i \in T} \sum_{j \in M_i} N_i^j \cdot M_i^j$
2. Erster Teil (Berechnung der bottom–up Information)
   Es müssen die Netze aller Teilschaltungen behandelt werden, und die Informationen über die Makros werden hochpropagiert: $A_T + \sum_{i \in T} R_i \cdot V_i$
3. Zweiter Teil (Berechnung der top–down Information und Erzeugung von Varianten)
   Wenn keine Varianten erzeugt werden, dann läßt sich der Aufwand wie im bottom–up Fall abschätzen: $A_T + \sum_{i \in T} R_i \cdot V_i$. Ob man Varianten erzeugen muß, und wenn ja wie viele, läßt sich im voraus nicht angeben. Die Erzeugung einer Variante heißt das Kopieren aller die Teilschaltung definierenden geometrischen Informationen und das Erzeugen neuer Verweise. Letzter Punkt kann in konstanter Zeit erfolgen, das Kopieren benötigt (wenn B_t die Anzahl der Basiszellen in Teilschaltung t angibt) die Zeit $O(B_t + |M_t| + A_t + S_t)$

Erfahrungswerte zeigen, daß bei hierarchischen Schaltungen, die Instanzen parametrisierter Schaltungen für konkrete Parameterwerte sind, kaum Entfaltungen nötig sind. Je

höher die Parameterwerte, umso geringer ist der relative zeitliche Aufwand zur Erzeugung eindeutiger Signalflußrichtungen.
Wie man sieht, hängt das Laufzeitverhalten von einer Vielzahl von Parametern ab, die durch die Topologie der Schaltung und die Art ihrer hierarchischen Darstellung bestimmt sind.
Der im nächsten Abschnitt definierte Index i_{mf} kann auch für hiesige Laufzeitanalyse einen Anhaltspunkt geben.

Eine Laufzeitanalyse für den Max–Fanout Algorithmus.
Der Zeitaufwand des Max–Fanout Algorithmus hängt ebenfalls von der Spezifikation der gegebenen Schaltung in der Hierarchie ab. Zur Analyse der Laufzeit wird auf diesen Punkt im Vergleich zum flachen Fall näher eingegangen.
Sei N_f die Anzahl der Netze in der flachen Darstellung, N_e die Anzahl von definierenden Vorkommen von Netzen in der gefalteten hierarchischen Darstellung bei eindeutigen Signalflußrichtungen. Geht man davon aus, daß der Aufwand für die Behandlung eines Netzes im hierarchischen wie im flachen Fall in etwa gleich ist (diese Annahme berücksichtigt nicht den Aufwand für den bottom–up Lauf im dag, den Aufwand für die Tiefensuchen und den Aufwand für die Propagation der Informationen durch die Hierarchie; außerdem wird davon ausgegangen, daß überwiegend Strategie 1 verwendet wird), dann kann man mit Hilfe des Index

$$i_{mf} := \frac{N_e}{N_f}$$

feststellen, ob man Laufzeitvorteile durch den hierarchischen Algorithmus hat: Dies wäre dann bei $i_{mf} < 1$ der Fall. Im allgemeinen macht es keinen Sinn, hierarchische Schaltungen mit Index $i_{mf} > 1$ zu spezifizieren (dies bedeutet: durch die Hierarchie gibt es wegen der Partitionierung der Netze mehr Netze als im flachen Fall). Man kann erwarten, daß die spezifizierten Schaltungen einen Index i_{mf} haben mit $i_{mf} << 1$.

4.4.3.2 Der Platzbedarf

Der Platzbedarf ist höher als bei vergleichbaren hierarchischen Algorithmen (hier sind Algorithmen gemeint, die mit einem bottom–up Lauf durch die Hierarchie auskommen), da Busse aufgesplittet werden: Für jeden Anschluß jeder Teilschaltung werden Informationen abgespeichert. Viel Platz wird benötigt, wenn die Gesamtschaltung in viele Teilschaltungen zerfällt und wenn sich an den Rändern der Teilschaltungen viele Anschlüsse mit hoher Busbreite befinden.

4.4.3.3 Resultate

Die Tabellen 4.1, 4.2 und 4.3 geben die Resultate des Max–Fanout–Algorithmus für hierarchische Schaltungen wieder (gemessen auf einer Sparc 10). Die zugrundeliegende Technologie dieser Schaltungen ist der IMS–Katalog, die Kapazität an den primären Ausgängen

Name	Max. Ausg. grad		Zeit		Kosten			Laufzeit
	vorher	nachher	vorher	nachher	vorher	nachher	Δc	(sigfluss)
ad[7]	65	3	45.2	292.0	1662	2178	20	0.9 (5.8)
ad[7]	65	5	45.2	191.2	1662	1858	16	0.8 (5.8)
ad[7]	65	9	45.2	135.2	1662	1730	12	0.8 (5.8)
ad[7]	65	17	45.2	98.5	1662	1682	8	0.8 (5.8)
ad[7]	65	33	45.2	68.8	1662	1666	4	0.7 (5.8)
ad[8]	129	3	158.2	602.7	3582	4866	24	1.8 (10.0)
ad[8]	129	5	158.2	403.3	3582	4098	20	1.8 (10.0)
ad[8]	129	9	158.2	296.2	3582	3778	16	1.8 (10.0)
ad[8]	129	17	158.2	230.0	3582	3650	12	1.8 (10.0)
ad[8]	129	33	158.2	178.0	3582	3602	8	1.7 (10.0)
ad[8]	129	65	158.2	128.0	3582	3586	4	1.6 (10.0)
ad[9]	257	3	307.7	1227.6	7678	10754	28	4.9 (18.8)
ad[9]	257	5	307.7	832.4	7678	8962	24	4.9 (18.8)
ad[9]	257	9	307.7	625.4	7678	8194	20	4.9 (18.8)
ad[9]	257	17	307.7	505.5	7678	7874	16	4.8 (18.8)
ad[9]	257	33	307.7	419.0	7678	7746	12	4.7 (18.8)
ad[9]	257	65	307.7	336.5	7678	7698	8	4.7 (18.8)
ad[9]	257	129	307.7	245.8	7678	7682	4	4.6 (18.8)

Tabelle 4.1: Der Max–Fanout–Algorithmus auf dem Conditional Sum Addierer

Name	Max. Ausg. grad		Zeit		Kosten			Laufzeit
	vorher	nachher	vorher	nachher	vorher	nachher	Δc	(sigfluss)
FDec[8]	16	5	21.2	23.0	312	568	102	5.9 (19.5)
FDec[8]	16	6	21.2	25.4	312	504	68	3.9 (19.5)
FDec[8]	16	8	21.2	19.9	312	440	68	5.0 (19.5)
FDec[8]	16	9	21.2	24.3	312	376	34	3.5 (19.5)
FDec[9]	32	5	30.7	28.4	593	1113	198	23.6 (53.0)
FDec[9]	32	6	30.7	30.8	593	1049	136	18.7 (53.0)
FDec[9]	32	7	30.7	32.0	593	1017	134	17.5 (53.0)
FDec[9]	32	8	30.7	23.2	593	849	132	15.6 (51.2)
FDec[9]	32	9	30.7	27.7	593	785	70	11.6 (51.2)
FDec[9]	32	10	30.7	32.5	593	753	68	13.4 (51.2)
FDec[9]	32	16	30.7	24.6	593	657	4	12.2 (51.2)
FDec[9]	32	17	30.7	33.7	593	625	2	8.4 (51.2)

Tabelle 4.2: Der Max–Fanout–Algorithmus auf dem 2^n–Bit Dekodierer FDec[n]

Name	Max. Ausg. grad		Zeit		Kosten			Laufzeit
	vorher	nachher	vorher	nachher	vorher	nachher	Δc	(sigfluss)
div[8]	22	19	392.2	392.2	1144	1146	2	0.5 (10.4)
div[8]	22	13	392.2	410.2	1144	1226	34	1.1 (10.4)
div[8]	22	7	392.2	383.4	1144	1288	68	1.5 (10.4)
div[16]	30	29	968.4	1004.4	4208	4366	46	2.0 (15.9)
div[16]	30	15	968.4	857.8	4208	4494	114	4.0 (15.9)
div[16]	30	8	968.4	715.3	4208	4744	168	4.8 (15.9)
div[24]	46	45	1727.0	1781.0	9192	9430	62	3.6 (20.4)
div[24]	46	23	1727.0	1423.5	9192	9618	158	7.0 (20.4)
div[24]	46	12	1727.0	1101.3	9192	9998	246	9.6 (20.4)
div[24]	46	11	1727.0	1100.5	9192	10094	254	9.9 (20.4)

Tabelle 4.3: Der Max–Fanout–Algorithmus auf dem n–Bit Dividierer div[n]

wurde mit 85fF angegeben, die Leitungskapazität einheitlich mit 40fF. Die zweite Spalte in den Tabellen zeigt den maximalen Ausgangsgrad der Schaltungen vor und nach der Modifikation. Die dritte Spalte enthält die maximalen Signallaufzeiten (in ns), die vierte Spalte die Kosten (Anzahl der Bausteine), jeweils vor und nach der Ausführung der Korrektur. Δc in der vierten Spalte bezeichnet die Anzahl der Instanzen (genauer: Anzahl der Verweise auf Basiszellen), die in die Teilschaltungen der gefalteten Darstellung eingefügt werden. Die letzte Spalte wurde für die Laufzeit reserviert, der in Klammern gesetzte Wert ist die Laufzeit zur Herstellung der eindeutigen Signalflußrichtungen.

Interessant ist hierbei, daß beim Addierer die Select–Leitung sich ähnlich einem binären Baum verzweigt und der Algorithmus durch Einsetzen der Treiberstufen nach den Abzweigungen das Fanout in etwa halbiert. Wie später, im Satz 4.14 bewiesen wird, setzt der Algorithmus die Treiber in einer Strategie ein, die das Fanout von $2^{k-1}+1$ auf $2^{x-1}+1$ reduziert. Dabei werden $2\cdot(k-x)$ Puffer in der gefalteten Darstellung und $2\cdot\sum_{l=1}^{k-x} l\cdot 2^{l-1}$ Puffer in der flachen Darstellung eingesetzt. Für obige Schaltungen muß man diese Werte noch mit 2 multiplizieren, da ein Puffer dort durch die Hintereinanderschaltung von zwei Invertern realisiert wird.

Höhere Parameterwerte waren aufgrund mangelnden Platzes nicht möglich. Obige Schaltungen werden von CADIC sehr platzsparend dargestellt, da beim Entwurf die Möglichkeit besteht, nicht nur Schaltungen, sondern auch die in ihnen enthaltenen Busse parametrisiert zu definieren. In der Ausprägung für einen konkreten Parameterwert berechnet das System die gefaltete Darstellung inklusive der Busbreiten. In solchen Bussen ist es sogar möglich, daß einzelne Leitungen, je nach Vorkommen der Teilschaltung als Makro, in unterschiedlichen Richtungen betrieben werden. Diese Platzvorteile können für den vorgestellten Max–Fanout Algorithmus leider nicht genutzt werden, da Busse aufgesplittet werden müssen.

Im zweiten Kapitel wurde bereits im Abschnitt „Klassifikation von Fanout–Algorithmen“ erwähnt, daß Max–Fanout–Algorithmen im allgemeinen nicht dazu geeignet sind, das Fanout–Problem mit Zeitanalyse zu lösen. Bei einer bestimmten Klasse von Schaltungen ist dies jedoch möglich. Um diese Klasse zu benennen, wird zunächst die Verzögerung durch einen kritischen Pfad analysiert: Es sei $\mathcal{B}$ die Familie der Bausteine durch die ein zeitkritischer Pfad von einem primären Eingang zu einem primären Ausgang läuft. Folgende drei Faktoren spielen für die Verzögerung durch diesen Pfad eine Rolle:

1. die Tiefe, d.h. die Anzahl der Bausteine in $\mathcal{B}$,
2. die internen Verzögerungszeiten der Bausteine in $\mathcal{B}$ und
3. die Kapazitäten an den Ausgängen der Basiszellen, durch die der Pfad läuft.

Der erste Faktor wird durch einen Max–Fanout–Algorithmus eher verschärft (er kann durch critical path resynthesis Strategien entschärft werden). Der zweite Faktor kann eventuell durch eine repower–Strategie verbessert werden. Der letzte Faktor ist der für die Eingrenzung der gesuchten Klasse entscheidende: Wenn er der Hauptfaktor für die Laufzeiten der kritischen Pfade ist und wenn die Laufzeiten durch hohes Fanout auf diesen Pfaden zustande kommen, dann bringt die Anwendung des Max–Fanout–Algorithmus für das Fanout–Problem mit Zeitanalyse gute Resultate. Der Wert für das maximale Fanout muß in diesem Fall so gewählt werden, daß einerseits nicht zuviele Treiber eingesetzt werden, daß jedoch andererseits stark zeitverzögernde Stellen an kritischen Pfaden entschärft werden.
Am Anfang des zweiten Kapitels wird im letzten Beispiel eine solche Schaltung vorgestellt.
Beim Dekodierer FDec kann man in obiger Tabelle denselben Effekt beobachten: Ein beschränkter Fanout–Wert kann zu einer Verbesserung der Signallaufzeit durch eine Logik führen.

4.4.3.4 Bewertung des Algorithmus

Ziel war es, einen Max–Fanout Algorithmus anzugeben, der auf hierarchisch spezifizierten Schaltungen arbeitet, die Schaltung und insbesondere die Hierarchie möglichst wenig verändert und die Kosten der Schaltung möglichst wenig erhöht.

Der vorgestellte Algorithmus garantiert, daß der spezifizierte Max–Fanout Wert eingehalten wird.

Der Algorithmus behält weitestmöglich die hierarchische Struktur bei, der Entwurf hat auch nach Ausführung des Algorithmus einen guten Wiedererkennungswert für den menschlichen Entwerfer.

Es werden „so spät wie möglich“, in anderen Worten: auf möglichst hoher Hierarchiestufe, Treiber(stufen) eingesetzt, d.h. der Algorithmus basiert auf einer need–to–Strategie. Diese

Strategie ist einfach und erzielt gute Laufzeiten, sie ist jedoch nicht kostenminimal. Für das angesprochene Problem bietet sich der Einsatz einer verteilten Berechnung an: Mehrere Rechner behandeln das gleiche Problem mit unterschiedlichen Max–Fanout–Vorgaben, die jedoch alle kleiner oder gleich dem maximal zulässigen Fanout sind. Anschließend wählt man sich eine Lösung aus (z.B. in Abhängigkeit davon, wie man für konkrete Schaltung die beiden folgenden Kriterien gewichtet: eine schnelle Signallaufzeit und der Platz).

4.5 Parametrisierte hierarchische Schaltungen

In diesem Abschnitt beschränken wir uns auf die Betrachtung des Max–Fanout Problems.

Rekursive Netzgleichungen wurden in [Kol86] eingeführt. Sie ermöglichen parametrisierte Beschreibungen von Schaltungen auf der Grundlage des Netzkalküls für integrierte Schaltungen. Das Kalkül wurde in [Hot74] grundgelegt und in [Mol86] vorgestellt. Wir gehen auf das Kalkül und die rekursiven Beschreibungen nicht näher ein, der interessierte Leser sei auf die Originalarbeiten verwiesen. Das weiter unten vorgestellte Beispiel der rekursiven Definition des Conditional Sum Adders in Abbildung 4.24 und die zugehörigen Erläuterungen geben einen kleinen Eindruck.

4.5.1 Realisierbarkeit

Im folgenden soll untersucht werden, ob es möglich ist, anhand einer allgemeinen, parametrisierten Beschreibung (auf der Grundlage des Netzkalküls) einer Schaltung eine Aussage über das maximal in dieser Schaltung vorkommende Fanout zu machen.

Nach [Kol86] sind folgende Probleme für rekursive Netzgleichungen generell nicht entscheidbar:

- das Lösbarkeitsproblem
- die Existenz eines Expansionsfunktors
- das Terminierungsproblem
- das Wortproblem

Hieraus folgt auch die Unentscheidbarkeit des folgenden Problems:
Eine Schaltung sei mittels rekursiven Netzgleichungen definiert. Weiter sei ein Max–Fanout Wert gegeben. Für welche Parameterwerte dieser Schaltung überschreitet wenigstens ein Netz der entsprechenden flachen Schaltung den gegebenen Max–Fanout Wert?
Beweis: Das Terminierungsproblem ist unentscheidbar, d.h. man kann im allgemeinen z.B. nicht voraussagen, ob eine rekursive Netzgleichung zu einer endlich oder zu einer unendlich großen Schaltung führt. Daraus folgt, daß es nicht entscheidbar ist, ob das Fanout eines Netzes endlich oder unendlich groß ist. ■

4.5.2 Ein Beispiel: Der Conditional Sum Adder

Im ersten Kapitel wurde in einem Beispiel der mit CADIC entwickelte, parametrisierte Conditional Sum Adder CSA[k] erwähnt. In einem früheren Beispiel zeigt Abbildung 4.1 die von CADIC generierte Ausprägung für den Parameterwert k=1, d.h. den 2–Bit Conditional Sum–Adder.
Abbildung 4.24 zeigt die graphische Eingabe dieses Addierers über Netzvariablen.

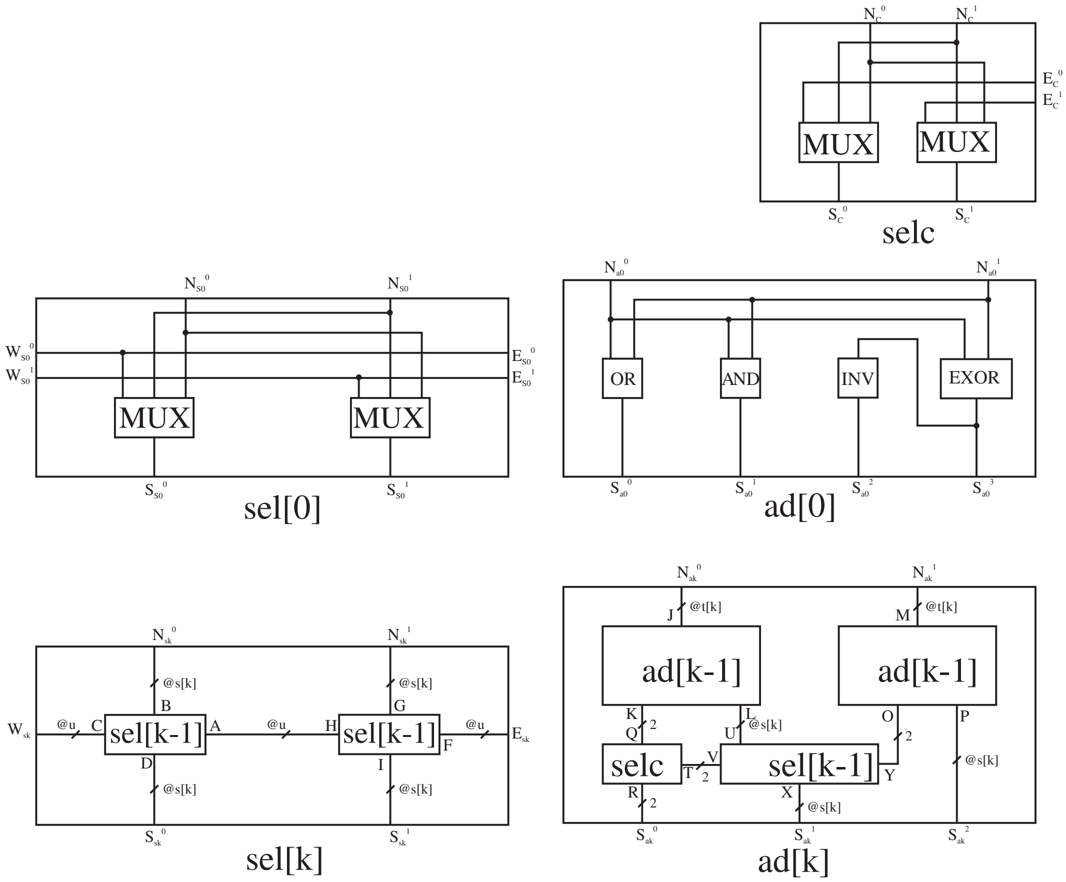

Abbildung 4.24: Der 2^k–Bit Conditional Sum Adder CSA[k]

Hierbei sind sel[k] und ad[k] „Rekursionsvorschriften", während sel[0] und ad[0] „Rekursionsbremsen" darstellen und selc schließlich eine „konstante Schaltung" ist.
Busse werden mit einem Strich durch die Leitung gekennzeichnet, daneben steht der Wert der Busbreite. Die Busbreite wird eventuell erst berechnet (z.B. @u) und kann auch von einem Parameter abhängen (z.B. @s[k]).
Diese Beschreibung ist stark verkürzt, eine ausführliche Beschreibung der graphischen Eingabe von Gleichungssystemen über Netzvariablen steht in [Bur94].

Die Funktionsweise dieses parallelen Addierers ist in [Skl60] (dies ist der Originalartikel von Sklansky) und in [KMO89] beschrieben. Informal wird die zugrundeliegende Idee in [KMO89] wie folgt beschrieben: „Man berechnet einfach zum führenden Teil der zu summierenden Zahlen zwei Versionen der Summe, nämlich die Summe und die Summe plus 1, parallel zur Berechnung der Summe des niederwertigeren Teils der Summanden. Ist der Übertrag des niederwertigeren Teils bekannt, so muß dann nur noch die richtige Version der Summe des vorderen Teils ausgewählt werden. Kann man diese Auswahl schnell, d.h. durch einen Schaltkreis konstanter Tiefe, berechnen und wendet man das Berechnungsschema rekursiv auf vorderen und hinteren Teil der Summanden an, so erhält man einen Schaltkreis logarithmischer Tiefe, sofern die Aufteilung in jeweils gleichgroße Teile erfolgt.“

Für diesen Addierer ist es möglich, das maximale Fanout der in ihm enthaltenen Netze über dem Parameter k anzugeben: $2^{k-1}+1$ (dieses Ergebnis stammt aus [KMO89], dort ist es jedoch ohne Beweis angegeben).
Beweis:

1. In selc sind E_C^0 und E_C^1 jeweils mit einem Eingangspin einer Basiszelle verbunden. Die Netze an S_C^0 und S_C^1 haben jeweils keinen Anschluß an einen Eingangspin einer Basiszelle und keinen weiteren Anschluß am Rand.

2. In sel[0] ist W_{S0}^0 mit E_{S0}^0 verbunden, ferner ist W_{S0}^1 mit E_{S0}^1 verbunden. Beide Netze besitzen einen Eingangspin einer Basiszelle.

3. In sel[k] gilt:

   (a) $W_{sk}[z]$ bildet mit $C[z]$ ein Netz.

   (b) $A[z]$ bildet mit $H[z]$ ein Netz.

   (c) $F[z]$ bildet mit $E_{sk}[z]$ ein Netz.

4. Behauptung: Für alle k ist in sel[k] der Anschluß W[z] mit dem Anschluß E[z] verbunden.
   Induktion:
   Induktionsverankerung: Betrachte k=0, d.h. sel[0]: klar (siehe 2.)
   Induktionsschluß: Die Behauptung gelte für k-1, d.h. es ist $C[z]$ mit $A[z]$, und es ist $H[z]$ mit $E_{sk}[z]$ verbunden. Hieraus folgt (mit Hilfe von 3.), daß $W_{sk}[z]$ mit $E_{sk}[z]$ verbunden ist.

5. Mit 4. läßt sich folgende Formel für das Fanout an einem östlichen (bzw. auch dem entsprechenden westlichen, da beide miteinander verbunden sind) Anschluß von sel[k] (mit k$\geq$1) herleiten:

   $$Fanout_{sel}[k] = 2 \cdot Fanout_{sel}[k-1]$$

   Mit 2. läßt sich hierfür ein Startwert angeben: $Fanout_{sel}[0] = 1$. Die Lösung dieser beiden Formeln ist:

   $$Fanout_{sel}[k] = 2^k$$

6. Für ad[0] gilt: Die Netze an S^0_{a0} und S^1_{a0} haben jeweils keinen Anschluß an einen Eingangspin einer Basiszelle und keinen weiteren Anschluß am Rand. Für ad[k] mit k>0 gilt entsprechend: Die erweiterten Netze an $S^0_{ak}[0]$ und $S^1_{ak}[1]$ haben jeweils keinen Anschluß an einen Eingangspin einer Basiszelle und keinen weiteren Anschluß am Rand (dies folgt aus den Eigenschaften der Süd–Anschlüsse von selc: siehe 1.).

7. Das Gesamtfanout des erweiterten Netzes bestehend aus den miteinander verbundenen Anschlüssen O[z], Y[z], V[z] und T[z] berechnet sich aus der Addition der folgenden Fanout–Werte:
   ad[k-1] hat nach 6. am Anschluß O[z] das Fanout 0.
   sel[k-1] hat nach 5. am Netz der Anschlüsse Y[z] und V[z] das Fanout 2^{k-1}.
   selc hat nach 1. das Fanout 1 am Netz des Anschlusses T[z].
   Insgesamt:
   $$Fanout_{ad}[k] = 0 + 2^{k-1} + 1 = 2^{k-1} + 1$$

Es wurde bewiesen, daß es erweiterte Netze gibt, die ein Gesamtfanout von $2^{k-1}+1$ haben. Nicht bewiesen wurde, daß dies die Netze mit dem höchsten Fanout sind. Um dies zu zeigen, kann beispielsweise bewiesen werden, daß alle anderen erweiterten Netze ein kleineres Gesamtfanout haben. Diese Beweise werden jedoch analog zu obigem Beweisschema geführt. ■

Der folgende Satz 4.14 macht eine Aussage über eine Puffereinsetzungsstrategie, die folgende Ergebnisse hat:

- Das maximale Fanout läßt sich mit je zwei Puffern, die in der gefalteten hierarchischen Darstellung eingesetzt werden, (bis auf eine additive Konstante von 1) halbieren.
- Die Kosten in der flachen Schaltung wachsen jedoch mit jedem Halbieren des maximalen Fanouts exponentiell.

Satz 4.14 ist eine Verfeinerung der in [KMO89] gemachten Kostenabschätzung für den Selektorbaustein.

Satz 4.14:
Es gibt eine Puffereinsetzungs–Strategie für den oben spezifizierten Addierer, die das Fanout von $2^{k-1}+1$ auf $2^{x-1}+1$ mit $1 < x < k$ reduziert. Dabei werden $2 \cdot (k-x)$ Puffer in der gefalteten Darstellung und $2 \cdot \sum_{l=1}^{k-x} l \cdot 2^{l-1}$ Puffer in der flachen Darstellung eingesetzt.

Die Strategie lautet wie folgt:
Es werden die definierenden Vorkommen der Selektorbausteine sel[y] (mit k-x $\leq$ y $<$ k) modifiziert. In jede der beiden Leitungen, die die zwei Selektorbausteine sel[y-1] verbindet, wird ein Puffer eingesetzt (die Signalflußrichtung dieser Leitungen ist von Osten nach Westen).

Für den Beweis wird zunächst folgendes Lemma gezeigt:

Lemma 4.15 :
Es gibt $l \cdot 2^{l-1}$ Selektoren mit dem Parameter k-l in der flachen Darstellung von ad[k].

Herleitung des Lemmas:
Die Anzahl der Vorkommen der Addierer ad[k-y] in der flachen Darstellung von ad[k] ist $2^k - y$, da die flache Darstellung bezüglich der ad–Bausteine einem binären Baum entspricht. Betrachtet man in der flachen Darstellung genau einen Teilbaum mit Wurzel ad[k-y], so gibt es in diesem Teilbaum $2^{(y-1)-(k-l)}$ Vorkommen der Bausteine sel[k-l], die im dag direkte Nachfahren des Bausteins sel[k-y-1] sind. Also gibt es insgesamt

$$\sum_{i=k-l+1}^{k} 2^{k-i} \cdot 2^{(i-1)-(k-l)} = l \cdot 2^{l-1}$$

Selektoren sel[k-l] in der flachen Darstellung von ad[k].

Beweis des Lemmas mit Induktion über l:

IV: l=1
Zeichnung 4.24 zeigt, daß ein Addierer ad[k] einen Selektorbaustein sel[k-1] enthält. Dies stimmt mit der Formel überein ($1 \cdot 2^{1-1} = 1$).

IS: $l \rightarrow l+1$
Die sel[k-(l+1)] Bausteine kommen entweder in sel[k-l] oder in ad[k-l] Bausteinen vor. Nach Induktionsvoraussetzung gibt es $l \cdot 2^{l-1}$ Selektoren sel[k-l], in denen also $2 \cdot l \cdot 2^{l-1}$ Selektoren sel[k-(l+1)] vorkommen. Es gibt 2^l Addierer ad[k-l] (die Addierer verzweigen sich in der Rekursion als vollständig binärer Baum.) in denen sich ebenso viele Selektoren sel[k-(l+1)] befinden. Insgesamt gibt es also $l \cdot 2^l + 2^l = (l+1) \cdot 2^l$ Selektoren sel[k-(l+1)].
■

Beweis des Satzes 4.14:
In obiger Betrachtung über das maximale Fanout der nicht modifizierten Schaltung wurde gezeigt, daß das maximale Fanout in zwei Netzen auftritt. Die beiden Netze verlaufen wie folgt (siehe Abbildung 4.24): Sie starten an den beiden Pins, die mit O markiert sind und haben dort Fanout 0. Weiter werden sie durch sel[k-1] an von Y zu V geführt und haben in sel[k-1] ein Fanout von 2^{k-1}. Von V geht es schließlich zu T am Baustein selc, wo von jedem Netz ein Eingang einer Basiszelle belegt wird. Innerhalb von sel[k-1] gibt es zwei Bausteine sel[k-2], wobei in jedem der beiden das Fanout 2^{k-2} angenommen wird. Verhindert man nun, daß diese beiden Bausteine miteinander verbunden sind (beispielsweise indem man einen Puffer in die Leitung zwischen ihnen einsetzt), dann ist das maximale Fanout nur noch vom linken Selektorbaustein abhängig, da er mit selc verbunden ist. Diese Argumentation läßt sich entsprechend fortsetzen, bis zum ersten Selektorbaustein, in dem kein Puffer eingesetzt ist. Damit ist bewiesen, daß das maximale Fanout gerade $2^{x-1} + 1$ ist, da sel[x] der letzte modifizierte und sel[x-1] der erste unmodifizierte Selektorbaustein ist.
Obige Strategie setzt in der gefalteten Darstellung in jeden der Selektorbausteine sel[x]

bis sel[k-1] (jeweils einschließlich) zwei Puffer ein, also sind die zusätzlichen Kosten in der gefalteten Darstellung gerade $2 \cdot (k - x)$.

Die Anzahl der eingesetzten Bausteine in der flachen Darstellung ergibt sich mit Hilfe obigen Lemmas durch Summenbildung.

Mit Hilfe dieser Ergebnisse läßt sich durch eine einfache Modifikation eine Fanout–Beschränkung der parametrisiert spezifizierten Schaltung erreichen: Wie in Abbildung

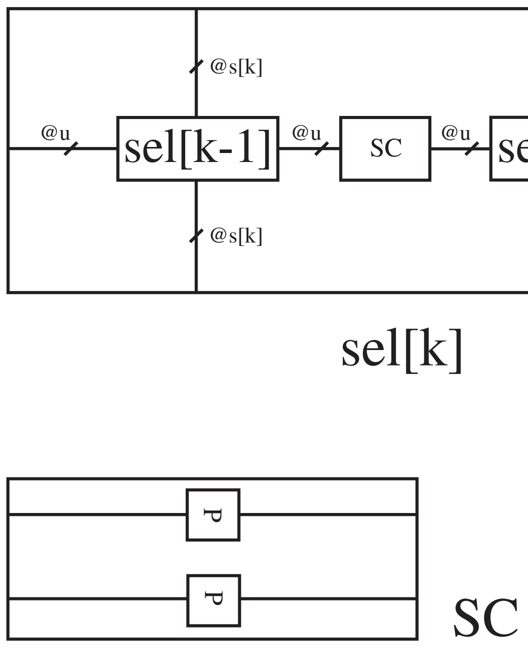

Abbildung 4.25: Mit dieser Modifikation wird das Fanout auf 3 beschränkt

4.25 gezeigt, kann man das Fanout der parametrisierten Schaltung auf 3 (für alle Parameterwerte) beschränken, indem man in den Selektorbaustein sel[k] zwischen die beiden Selektoren sel[k-1] Pufferbausteine einsetzt.

4.5.3 Eingeschränkte Klasse von Schaltungen

Interessant wäre es nun zu untersuchen, ob es möglich ist, allgemeine Aussagen über eingeschränkte Klassen von Schaltungen zu machen. Hier bietet sich folgende Klasse von Schaltungen an: Schaltungen, deren Rekursion auf jeden Fall terminieren wird, da der Parameterwert streng monoton fällt. Ein Beispiel für eine solche Schaltung ist der Conditional Sum Adder des letzten Abschnitts.

Um einen allgemeinen Algorithmus für solche Schaltungen zu erhalten, könnte man die Vorgehensweise des letzten Abschnittes untersuchen und der Frage nachgehen, ob sich diese Vorgehensweise verallgemeinern läßt: Es wurden zuerst Hypothesen über die erweiterten Netze aufgestellt, d.h. Hypothesen darüber, welche Pins und Pads miteinander verbunden sind. Wenn man über alle erweiterten Netze eine solche (parametrisierte) Formel angeben kann, dann kann man auch voraussagen, an welcher Stelle das größte Fanout auftreten wird und wie groß es sein wird. Diese Aussage könnte dann wiederum helfen, Korrekturstellen innerhalb des hierarchischen Entwurfs anzugeben, an denen er modifiziert wird, um das Max–Fanout–Problem zu lösen.

Kapitel 5

Einbindung in CADIC

CADIC ist offen konzipiert: das System erlaubt die Integration von Werkzeugen „auf unterschiedliche Art und Weise, je nachdem wie stark die Bindung an die vorgegebenen Datenstrukturen ausgeprägt ist.“ (siehe [Bur94]). Im folgenden werden die Implementierungen und die Integration in das CADIC–System beschrieben.

5.1 Der Algorithmus von Singh und Sangiovanni–Vincentelli

5.1.1 Datenstrukturen

Die Implementierung des Algorithmus in [SSV90] akzeptiert Schaltungen, die in CADIC spezifiziert wurden. Hierbei wird das CADIC–Format zur Schaltungsbeschreibung in ein internes Format umgewandelt. Dieses Format unterstützt nur die für das Fanout–Problem relevanten Informationen, insbesondere wird von der Leitungsführung und der Geometrie der Bausteine abstrahiert. Dadurch ist es einfacher möglich,

- Bausteine dynamisch einzufügen, denn die Modifikationen des grafischen Entwurfs werden erst durchgeführt, wenn alle Veränderungen der Schaltungsstruktur bekannt sind (zu modifizierende Schaltungsteile sind dann zusammengefaßt und damit grafisch besser handhabbar),
- die Darstellung in Datenstrukturen für Algorithmen der LEDA–Bibliothek [Näh93] oder andere Formate (siehe beispielsweise die Verwendung der nlv–Datenstruktur im nächsten Abschnitt) umzuwandeln.

Das Werkzeug liegt also als eigenständiges Modul vor, das CADIC–Dateien zum Austausch verwendet. Es erzeugt die modifizierten Schaltungen im CADIC–Format, so daß diese von der graphischen Oberfläche eingelesen werden können.

Der Algorithmus (ohne die Visualisierung der Ergebnisse in CADIC) wurde im Rahmen eines Fortgeschrittenenpraktikums implementiert. Die Dokumentation [Kre93] erläutert die hierbei verwendeten Datenstrukturen und die einzelnen Funktionen. Die topologische Sortierung der Bausteine und der MAX–FLOW–Algorithmus für die mincut–Berechnung werden durch LEDA–Algorithmen realisiert.

Für die Visualisierung der Ergebnisse in CADIC muß die vom Entwerfer eingegebene Schaltung modifiziert werden. Die beiden in den Abbildungen 3.4 und 3.5 wiedergegebenen Transformationen verändern die Leitungsführung und setzen neue Bausteine (Inverter) ein. Die nächsten beiden Abschnitte behandeln diese Modifikationen anhand einer Beispielschaltung.

5.1.2 Einfügen neuer Bausteine

Die Beispielschaltung von Abbildung 5.1 soll die Vorgehensweise demonstrieren. Wie bereits erwähnt, wirken sich die Modifikationen der Schaltung durch den Algorithmus zunächst nur auf die interne Schaltungsrepräsentation aus. Für die Modifikation der Schaltung auf grafischer Ebene werden die Bausteine lokalisiert, an deren Ausgang neue Bausteine eingesetzt werden. Im Beispiel ist dies nur der Ausgang der oberen AND–Zelle. Die Leitungen, die von diesem Ausgang zu seinen (alten) Zielen führen, werden nun gelöscht. Dadurch daß der Algorithmus birekursiv arbeitet, kann es sein, daß an dem betrachteten Ausgang Inverter durch mehrere Modifikationen eingefügt wurden. Man geht deswegen von einer allgemeinen Topologie des Treiberbaumes aus. Ein Ziel bei der Schaltungsmodifikation ist ein möglichst hoher Wiedererkennungswert des Ursprungsentwurfs. Hieraus erklärt sich folgende Vorgehensweise: Das Programm „verpackt“ die Treiberbäume jedes Ausgangs (egal wie sie entstanden sind, d.h. wie komplex sie sind) in je eine „Black Box“. Die AND–Zelle hat einen Ausgang, deswegen gibt es im Beispiel einen Treiberbaum, der in Abbildung 5.2 gezeigt ist. Ein Treiberbaum hat einen Eingang und so viele Ausgänge wie der ursprüngliche Ausgangsgrad des betrachteten Ausgangs. Anschließend wird eine weitere „Black Box“ erzeugt, im Beispiel die Teilschaltung der Abbildung 5.3. Sie enthält den betrachteten Baustein und sämtliche „Black Boxes“ der Treiberbäume seiner Ausgänge. Dieses Makro wird nun anstelle des ursprünglichen Bausteins in die Schaltung eingesetzt. Nun müssen lediglich die Ausgänge des Makros mit ihren Zielen verbunden werden. Dazu dient der im nächsten Abschnitt kurz beschriebene Algorithmus. Die resultierende Schaltung des Beispiels sieht man in Abbildung 5.4, ihre hierarchische Struktur ist in Abbildung 5.5 wiedergegeben.

Die Implementierung des Algorithmus aus [SSV90] hat einen Umfang von 8393 Programmzeilen (im folgenden mit loc abgekürzt = <u>l</u>ines <u>o</u>f <u>c</u>ode). Die Routinen wurden in C und in C++ geschrieben.

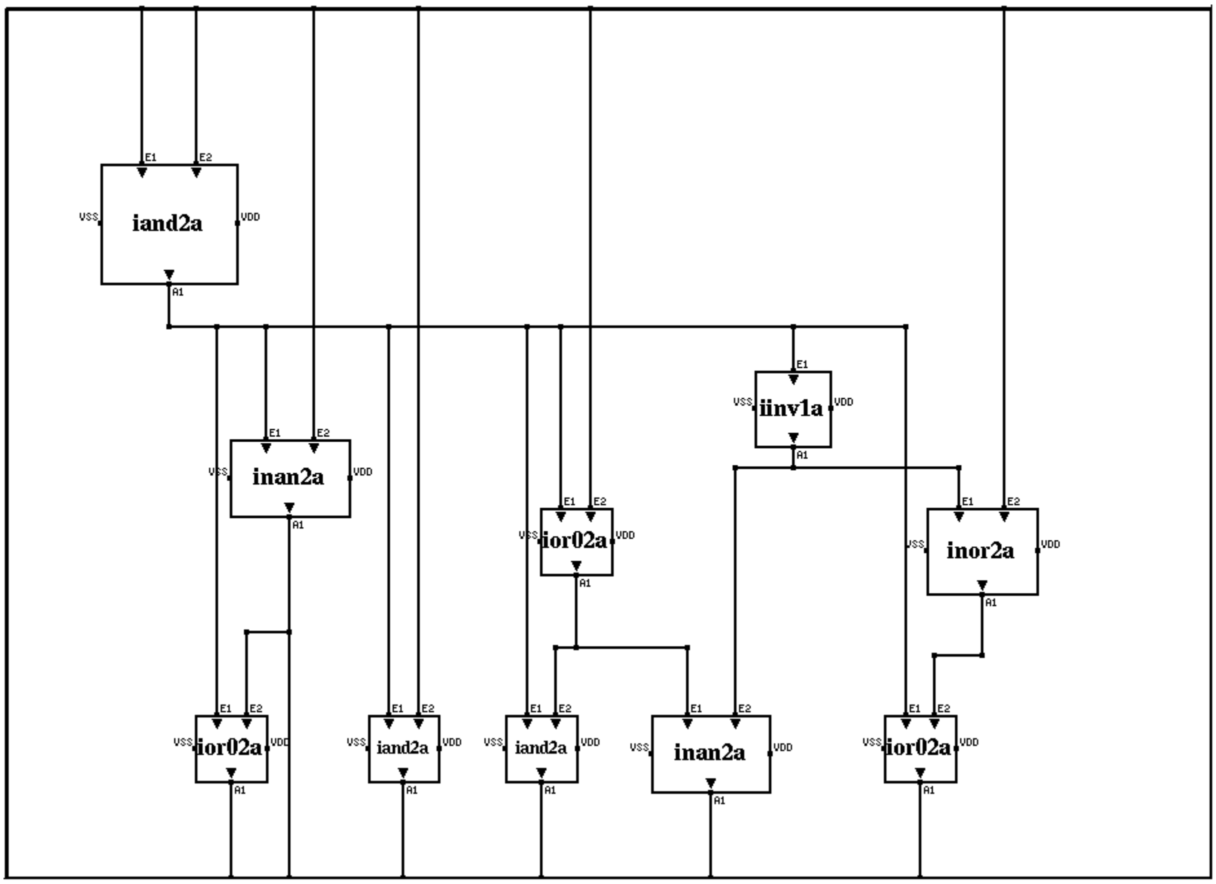

Abbildung 5.1: Eine Beispielschaltung für den Singh/Sangiovanni–Vincentelli Algorithmus

5.1.3 Algorithmus zur Leitungsführung

In [RLW85] stellen P. J. de Rezende, D. T. Lee und Y. F. Wu einen Algorithmus vor, der einen kürzesten Weg bezüglich der L_1–Norm (beziehungsweise Manhattan–Distanz) zwischen zwei Punkten in einer Szene mit sich nicht überlappenden, achsenparallelen Rechtecken findet. Der Algorithmus verwendet die Sweep–Line–Technik, die Korrektheit wird ebenfalls in [RLW85] bewiesen.

Wir modifizierten den Algorithmus für das im letzten Abschnitt beschriebene Verdrahtungsproblem. Darüberhinaus entwarfen wir eine einfache Routine zur Elimination von Knicken, da für den Algorithmus die Minimierung der Anzahl der Knicke kein Optimierungskriterium ist. Die Veränderungen des Algorithmus und seine Implementierung sind in [Kre94] dokumentiert. Im folgenden wird in groben Zügen beschrieben, wie der Algorithmus an das Verdrahtungsproblem angepaßt wurde.

Als blockierende Rechtecke werden die leicht vergrößerten Bausteine genommen. Die Bausteine wurden hierbei so weit ausgedehnt, daß zwischen je zwei erzeugten Rechtecken Platz

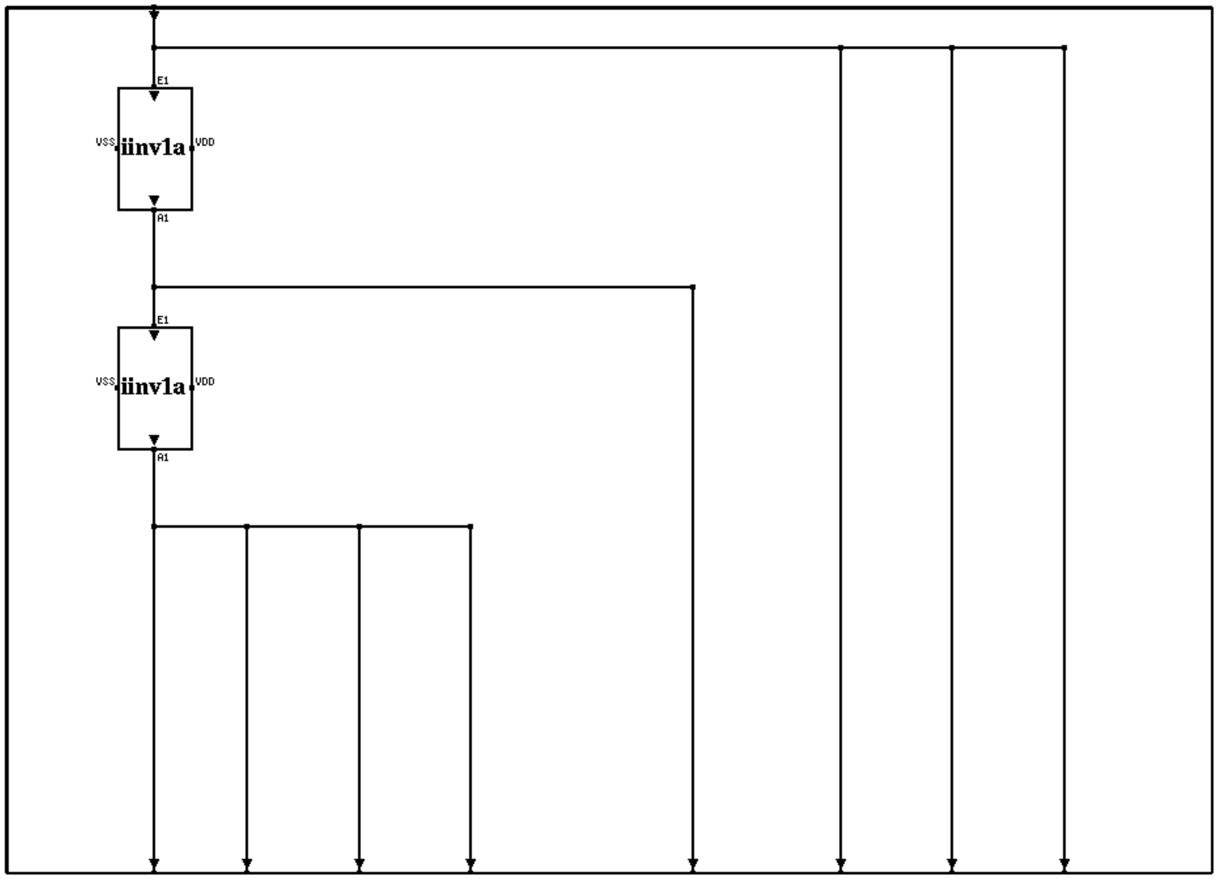

Abbildung 5.2: Der Treiberbaum

für die Verdrahtung bleibt. Durch diese Modellierung erhält man einen „Sicherheitsabstand“ zwischen den neuen Leitungen und den bestehenden Bausteinen.
Die bestehende Verdrahtung stellt zunächst kein Hindernis für den Algorithmus dar. Falls die gefundene Verdrahtung sich mit ihr überlappt oder sonstwie verbotenerweise mit ihr kollidiert, dann wird die neue Verdrahtung verschoben, so daß als einzige gemeinsame Punkte die erlaubten nichtkontaktierten Überkreuzungen verbleiben.

Ein Beispiel für den Verdrahtungsalgorithmus wird in der Abbildung 5.6 gezeigt.

Die Implementierung des Algorithmus aus [RLW85] hat einen Umfang von 2619 loc. Das Programm wurde in C++ geschrieben.

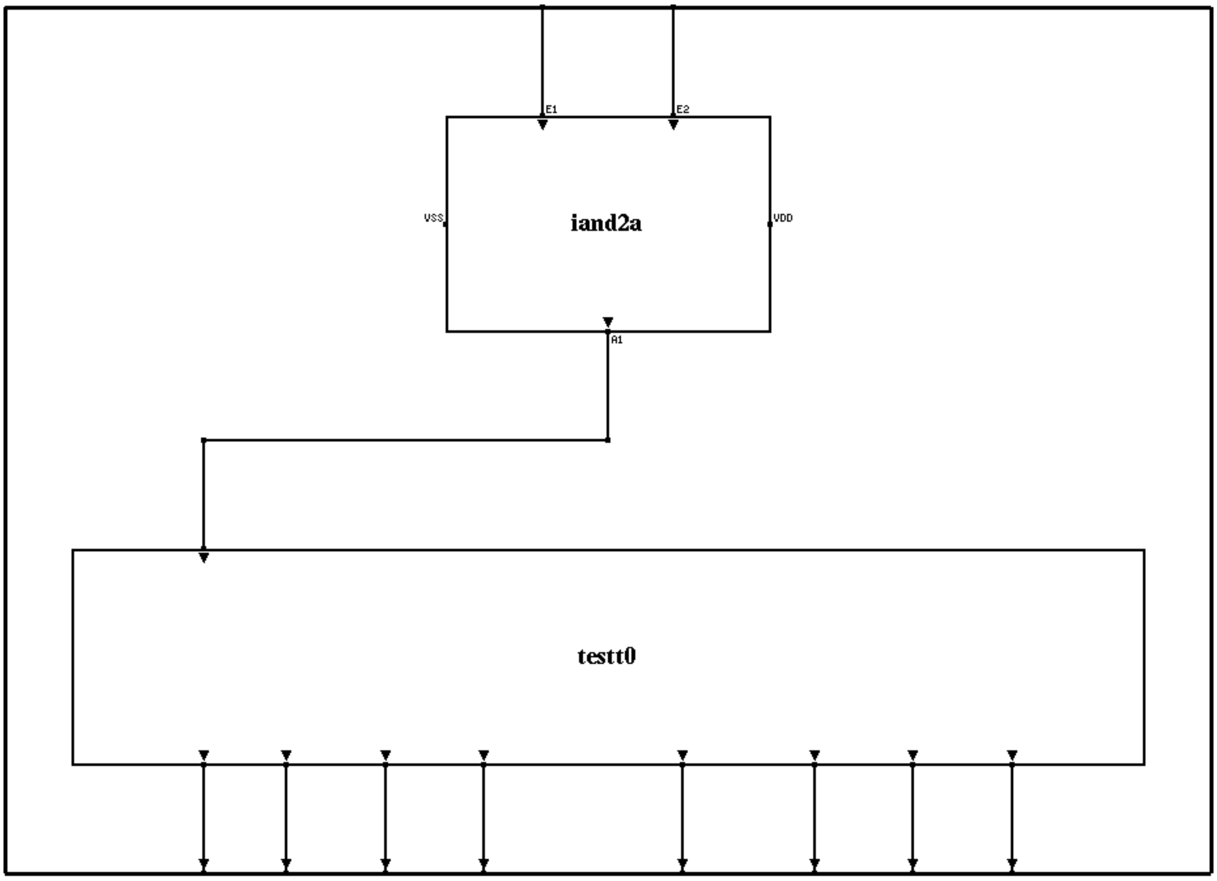

Abbildung 5.3: Der Ersatzbaustein für den AND–Baustein

5.1.4 Der Algorithmus als Analysewerkzeug für hierarchische Schaltungen

Der Algorithmus von Singh und Sangiovanni–Vincentelli wurde auch für hierarchisch spezifizierte Schaltungen implementiert, die „virtuell flach“ vorliegen. Genauer: Wenn das Programm eine hierarchische Schaltung als Eingabe erhält, dann löst es das Fanout–Problem, als wäre eine flache Schaltung gegeben. Möglich wird dies durch die in [BV94] beschriebene nlv–Datenstruktur. Sie stellt eine Darstellung der flachen Schaltung ohne Topologie zur Verfügung, indem für jedes Vorkommen einer Basiszelle (in einer flachen Schaltung) ein Objekt, ein sogenannter Selektor, generiert wird, sowie die Netze zwischen den Selektoren berechnet werden.
Die Ausgabe des Algorithmus sind die Bausteine und Netze, an denen Treiberbäume eingesetzt werden sollen, d.h. die Treiberbäume werden nicht grafisch in die Schaltung eingesetzt (dies wäre nur möglich, wenn die Schaltung flach vorliegt; vergleiche hierzu das vierte Kapitel).

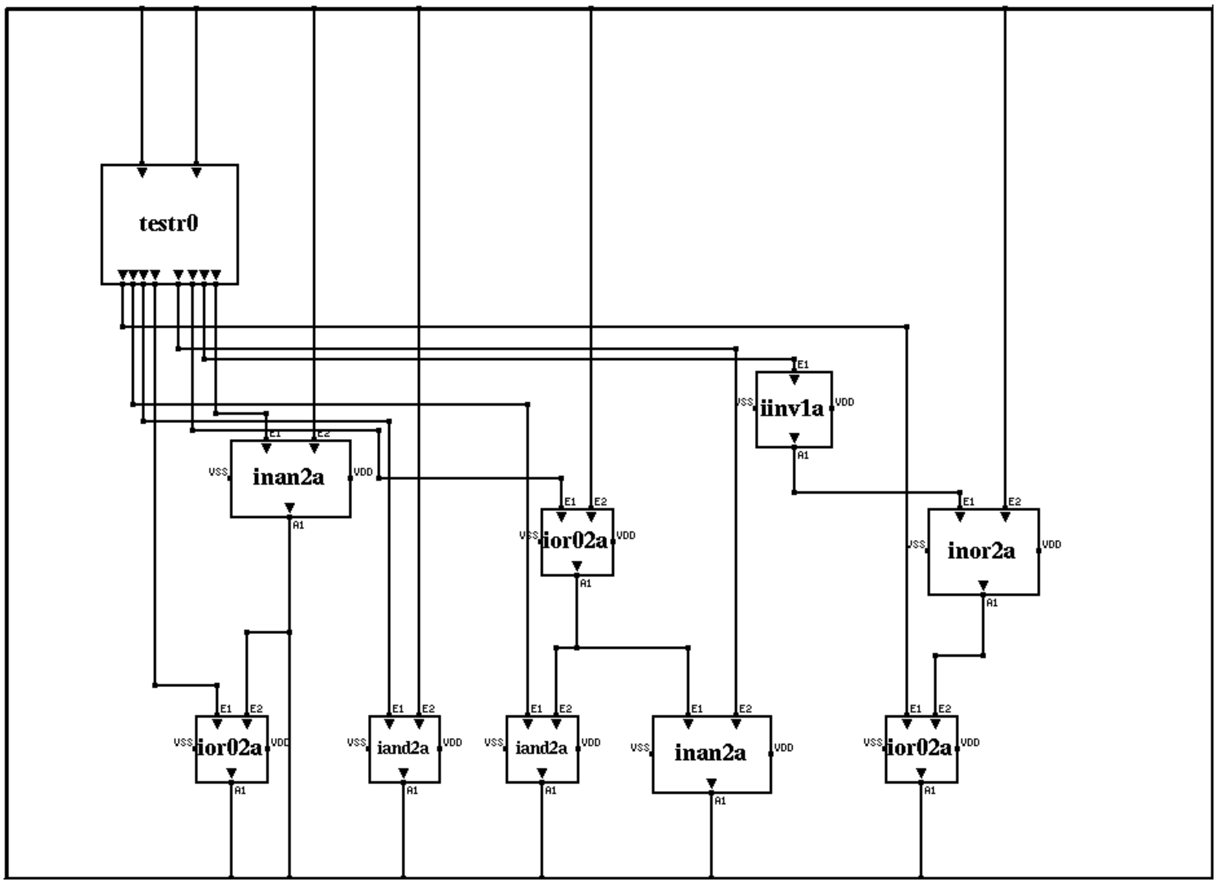

Abbildung 5.4: Die Beispielschaltung nach Anwendung des Singh/Sangiovanni–Vincentelli Algorithmus und mit neuer, automatisch generierter Verdrahtung

Diese Vorgehensweise hat folgende Vorteile:

- Der Algorithmus kann als Analysewerkzeug genutzt werden, denn er berechnet die Durchlaufzeiten der Signale.
- In einer hierarchisch spezifizierten Schaltung steht ein Vorkommen einer Basiszelle im allgemeinen für viele Vorkommen innerhalb der flachen Schaltung. Wendet man den Algorithmus auf der hierarchisch spezifizierten Schaltung an, dann kann man mit Hilfe des Visualisierungstools von Jörg Ritter [Rit96] die Basiszellen sehen, an denen viele Treiberbäume eingesetzt werden müssen. Es ist also möglich, die Güte des expandierten Entwurfs in Bezug auf das Fanout–Problem zu überprüfen und eventuelle Änderungen noch vor der Fertigung durchzuführen. Die Verknüpfung des hierarchischen Fanout–Algorithmus mit CADIC geschieht in diesem Fall mit Hilfe von Interprozeßkommunikation, die ebenfalls von Jörg Ritter für CADIC realisiert

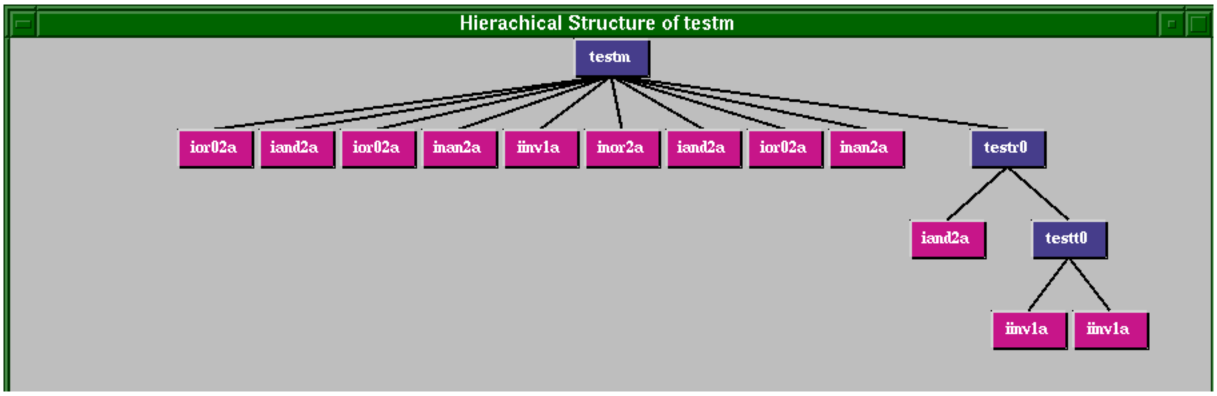

Abbildung 5.5: Die hierarchische Struktur der modifizierten Beispielschaltung

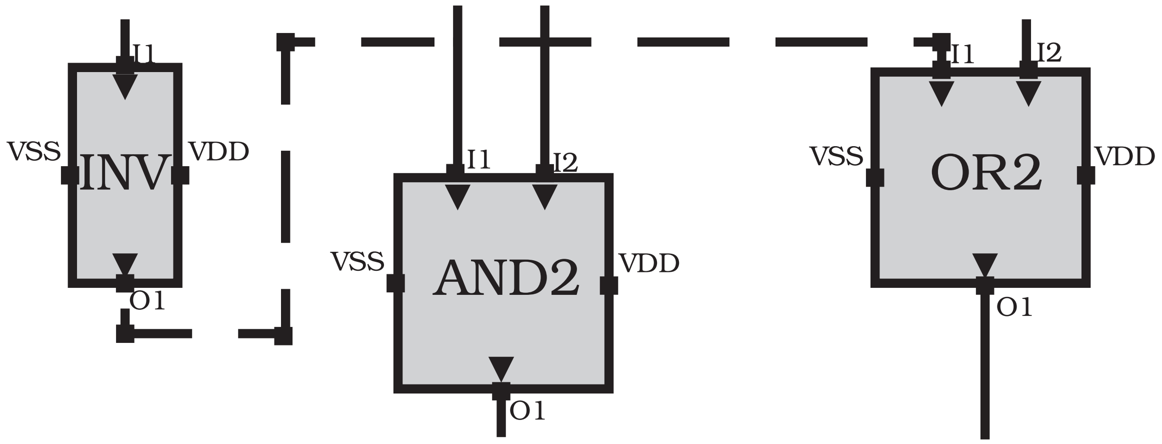

Abbildung 5.6: Der Ausgang des Inverters wird mit dem ersten Eingang des OR–Bausteins verbunden.

wurde. Abbildung 5.7 zeigt ein Beispiel, in dem für eine Basiszelle die verschiedenen Zeiten angegeben sind.

Jede Schaltung muß „früher oder später“ flach vorliegen, da die Maske des Chips, der gefertigt werden soll, flach ist. Die Verwendung der nlv–Struktur ermöglicht das Arbeiten auf einer flachen Darstellung,

- die platzsparend ist, da für jede Basiszelle lediglich ein Selektor reserviert wird,
- und die effizient ist, da die Geometrie der flachen Schaltung nicht berechnet wird.

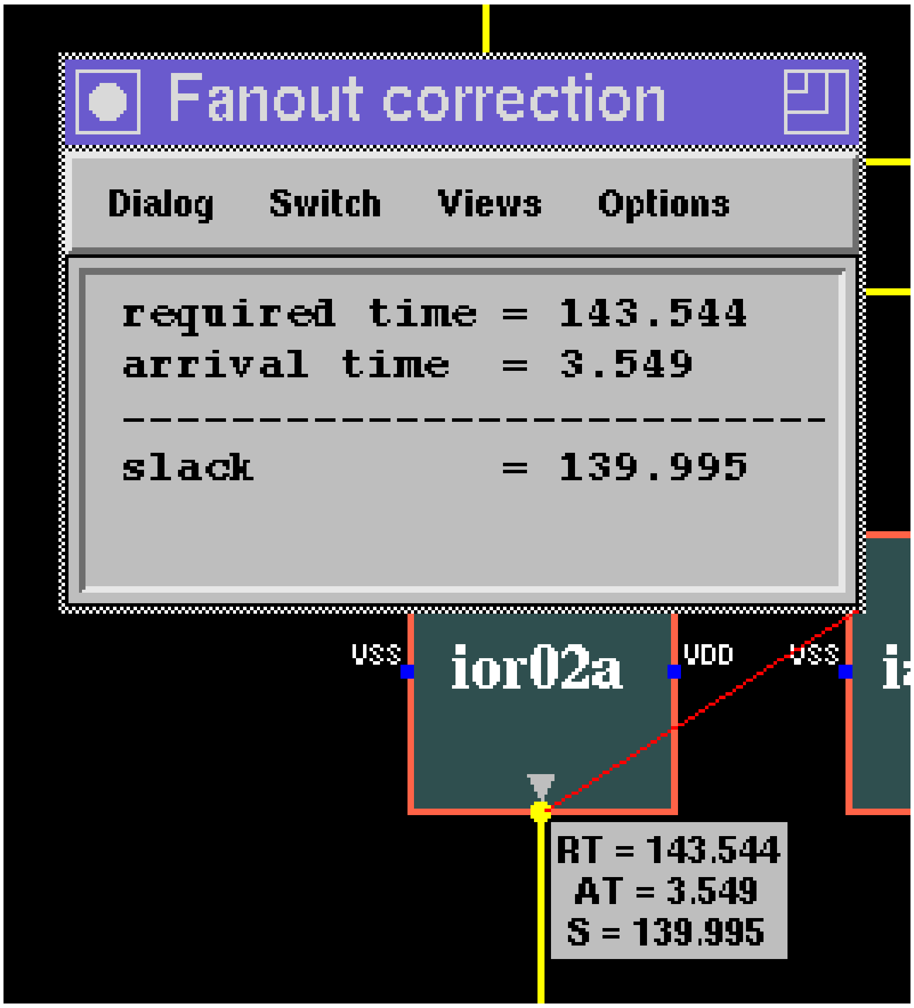

Abbildung 5.7: Ein Beispiel für die Visualisierung der Ergebnisse

5.2 Der hierarchische Max–Fanout–Algorithmus

Der hierarchische Max–Fanout–Algorithmus wurde ebenfalls implementiert und in das CADIC–System integriert. Zur Integration wurde die interne Bindung zwischen privaten Datenstrukturen und der Basisdatenstruktur von CADIC verwendet (siehe [Bur94]). Das Werkzeug ist in einer separaten Bibliothek abgelegt und in die Benutzeroberfläche integriert. Dies war einfach zu realisieren dank der von CADIC zur Verfügung gestellten leicht anzupassenden Menübeschreibung. Die Implementierung in C und in C++ umfaßt 4633 loc.

Man ruft den Algorithmus wie folgt auf: Nach dem Starten von CADIC klickt man im Hauptmenü auf die vierte Schublade von „Analysis“ namens „Fanout Correction“ (siehe Abbildung 5.8a). In der Menüleiste erscheint das Untermenü von Abbildung 5.8b. Dieses

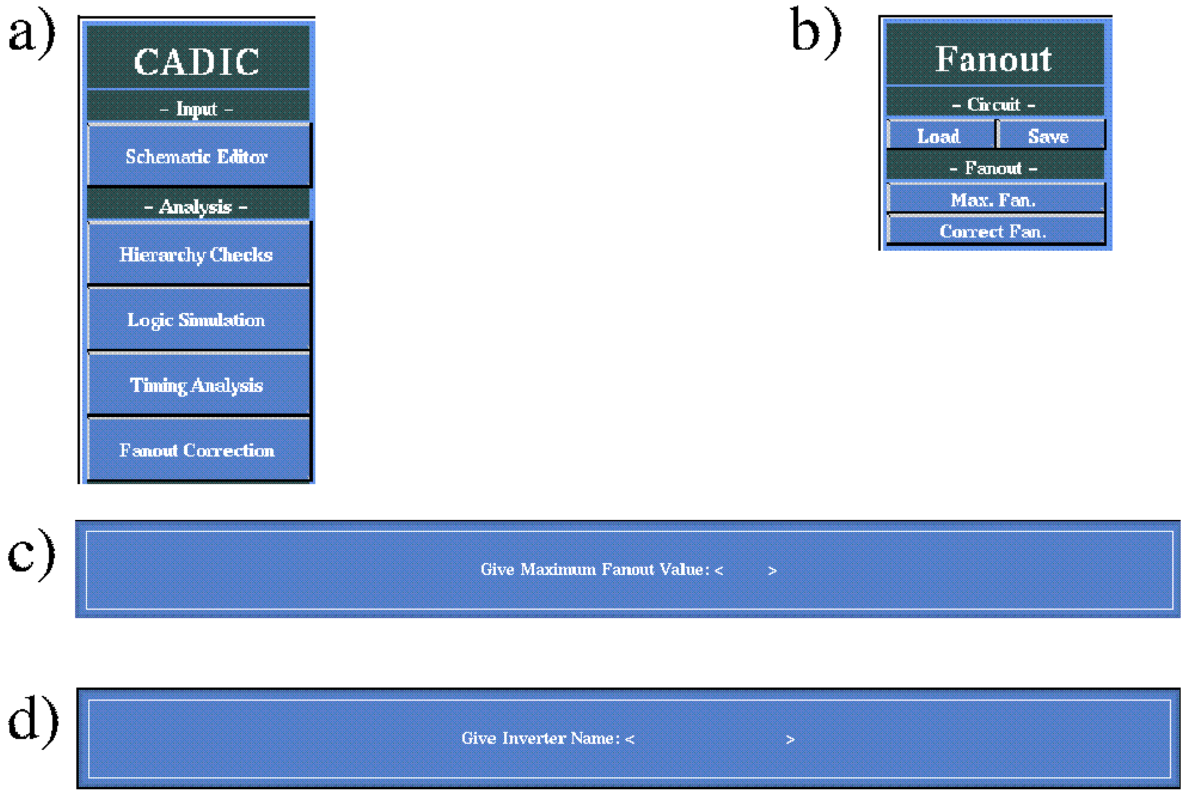

Abbildung 5.8: Menüführung und Eingabemasken für den Max–Fanout–Algorithmus unter CADIC

Untermenü hat die Funktionen „Max Fan“ und „Correct Fan“. Die übrigen Funktionen werden zum Laden und Speichern von Schaltungen, zum Navigieren durch die Hierarchie und schließlich für die Wahl des Bildschirmausschnittes verwendet. Wählt man den Menüpunkt „Max Fan“, so wird man aufgefordert, den maximalen Fanout–Wert einzugeben (siehe Abbildung 5.8c). Anschließend erwartet das Programm die Eingabe eines Namens eines Inverters (siehe Abbildung 5.8d). Zwei hintereinandergeschaltete Inverter dieses Typs bilden später eine Treiberstufe. Wenn eine Schaltung geladen ist und man nun die Funktion „Correct Fan“ startet, dann wird die gewünschte Fanout–Korrektur berechnet. Den Fortschritt dieser Funktion kann man im Fenster „CADIC–Messages“ beobachten.
Es ist auch möglich, auf eine geladene Schaltung direkt „Correct Fan“ anzuwenden (ohne vorherige „Max Fan“ Eingabe). In diesem Fall wird für den maximalen Fanout der vorbelegte Wert (d.h. 5) genommen, jedoch erscheint vorher das Fenster der Abbildung 5.8d zur Eingabe des Namens des Inverters.

Als Beispiel für dieses Tool wird der Dekodierer FDecoder[8] genommen, dessen Wurzelschaltung im dag in Abbildung 5.9 gezeigt wird. Sein maximales Fanout von 16 soll auf 4 reduziert werden. Die beiden Bausteine innerhalb von FDecoder[8], bei denen der

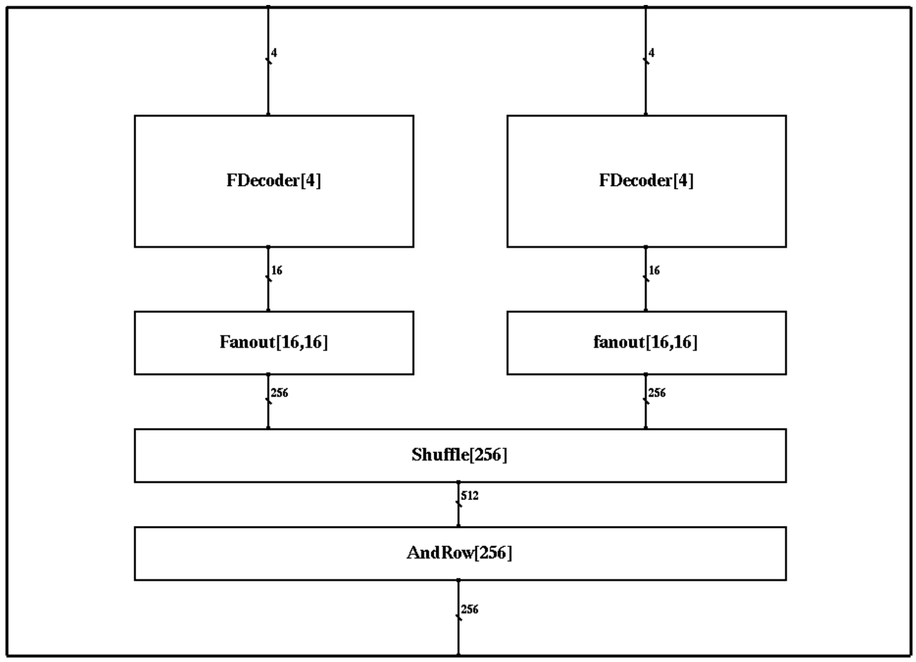

Abbildung 5.9: Die oberste Teilschaltung des 2^8–Bit Dekodierers

Fanout–Wert 16 auftritt, sollen hier näher betrachtet werden. Dies sind Fanout[16,16] und fanout[16,16], deren genauer Aufbau in Abbildung 5.10 dargestellt ist. Der Max–fanout–Algorithmus modifiziert die Teilschaltungen fanout[16,8] und Fanout[1,8]. In beide Teilschaltungen werden je zwei Makros eingesetzt, wie man in Abbildung 5.11 und 5.12 sehen kann. Die Makros beinhalten die Treiberstufen, wie in den Abbildungen 5.13, 5.14, 5.15 und 5.16 wiedergegeben.

5.2.1 Die Berechnung der Netzlisten

CADIC stellt drei Netzlistenfunktionen zur Verfügung:

- Die Funktion *calc_netlist* berechnet die in Definition 4.9 eingeführte Netzliste mit einer Ausnahme: Für *calc_netlist* gehören alle Leitungen eines Busses zu demselben Netz.
- Mit Hilfe eines Expansionsmoduls [Rit94] kann eine hierarchische Netzliste erzeugt werden. In diesem Modul werden jedoch die Busse soweit wie möglich erhalten.
- Wie schon erwähnt kann man mit Hilfe der nlv–Datenstruktur die Netzliste für eine virtuell flache Schaltung berechnen.

Der Max–Fanout–Algorithmus benötigt jedoch eine erweiterte Netzliste, wie sie in Definition 4.11 vorgestellt wird. Dazu werden die von *calc_netlist* gefundenen Netzlisten gesplittet, um die Netzliste von Definition 4.9 zu erzeugen. Die erweiterte Netzliste aus Definition 4.11 erhält man nun, indem man Netze mit Hilfe der sie verbindenden erweiterten Netze in den Makros miteinander vereinigt. Hierzu wird die Routine *calc_exp_netlist* mit *DagDFS* in bottom-up Depth First Reihenfolge auf dem dag aufgerufen. M sei die Menge der Makros der aktuellen Teilschaltung T.

proc *calc_exp_netlist(T)*
begin
  calc_netlist(T);
  split_busses(T);
  for all $m \in M$ **do**
    $\forall$*extended_nets* $\in m$: *unite_nets(m,T)*
  od
end

Die Funktion *unite_nets* vereinigt zwei Netze miteinander, wenn sie Pins haben, die sich als Pads in der selben erweiterten Netzliste von m' befinden. Mit m' wird hier die definierende Teilschaltung des Makros m bezeichnet.

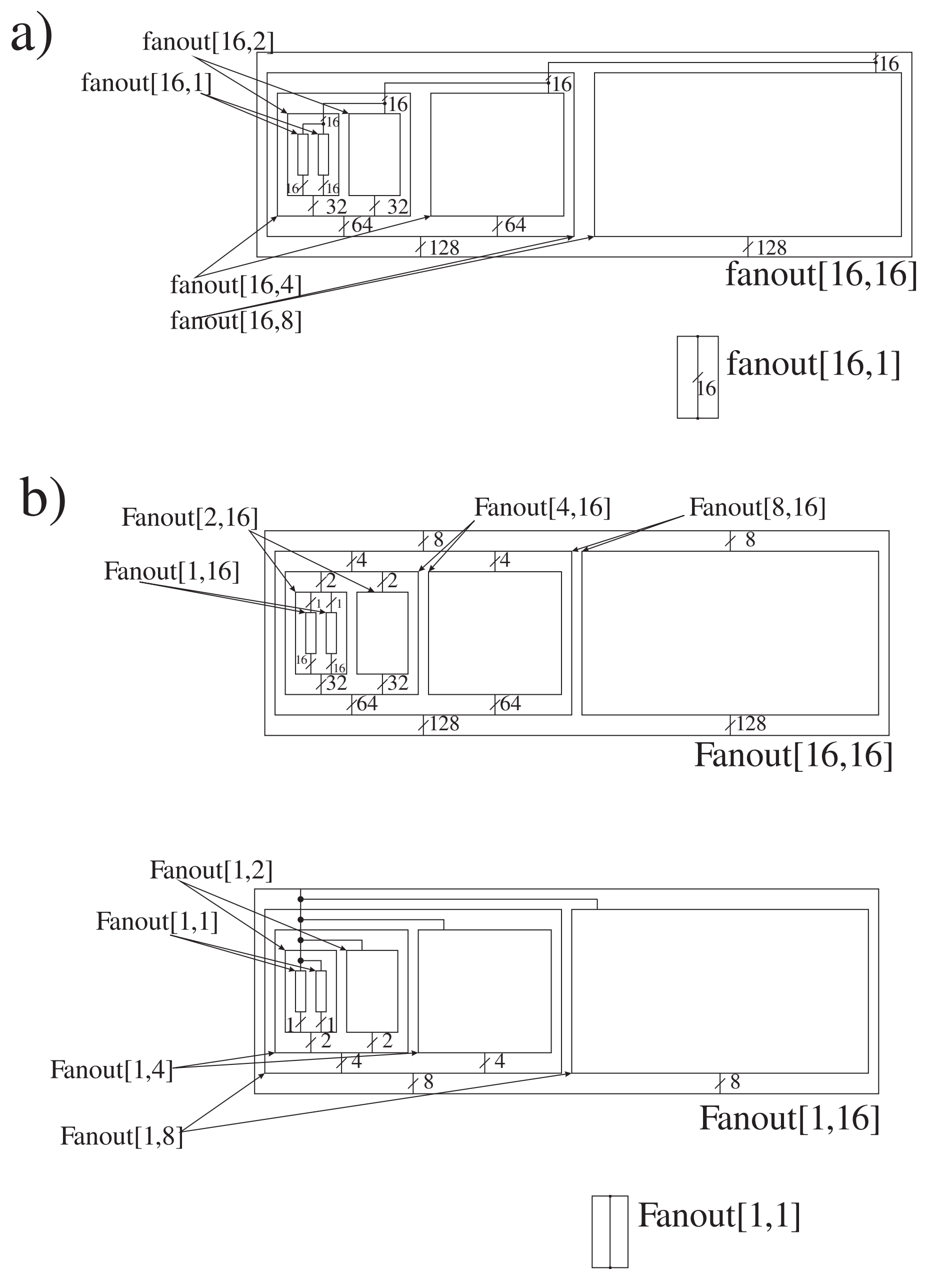

Abbildung 5.10: Der Aufbau der Fanout–Bausteine fanout[16,16] und Fanout[16,16]

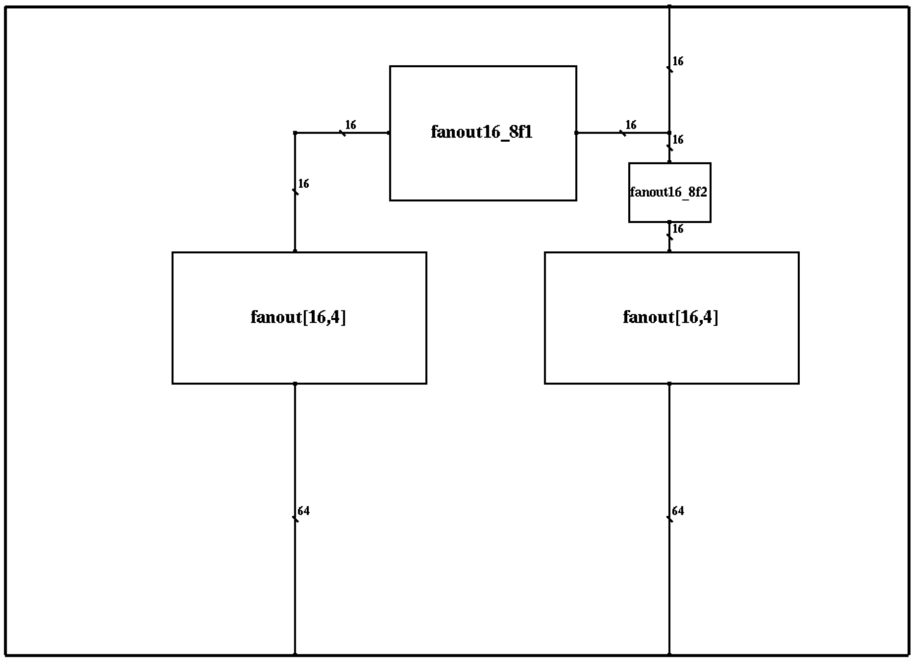

Abbildung 5.11: Die modifizierte Schaltung fanout[16,8]

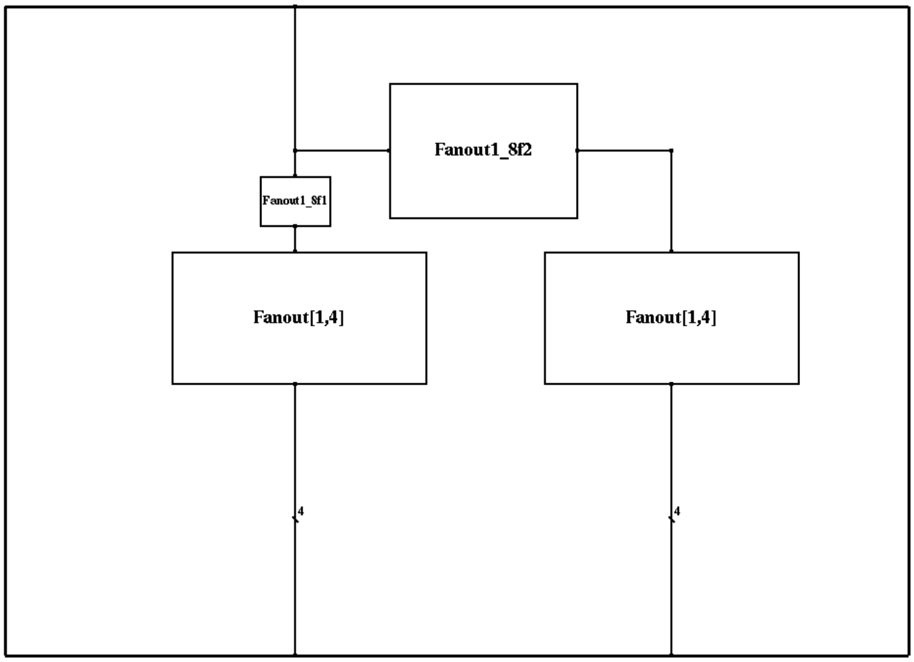

Abbildung 5.12: Die modifizierte Schaltung Fanout[1,8]

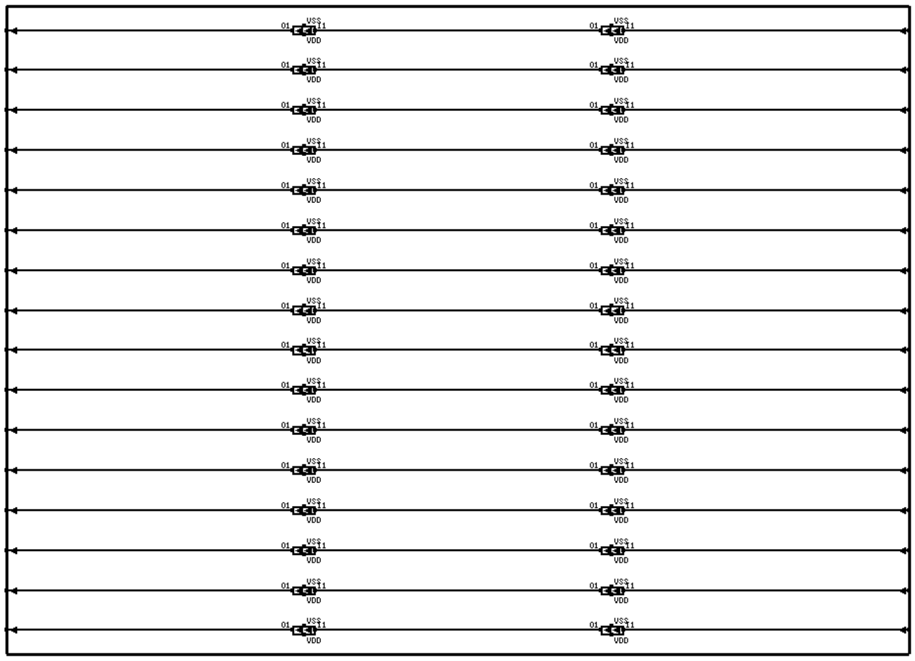

Abbildung 5.13: Die Teilschaltung fanout16_8f1

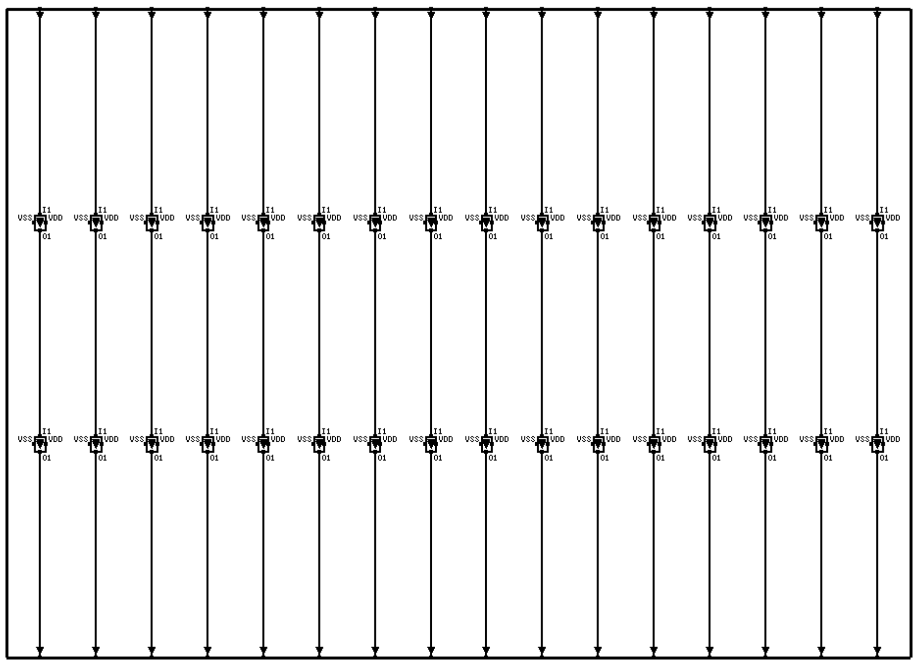

Abbildung 5.14: Die Teilschaltung fanout16_8f2

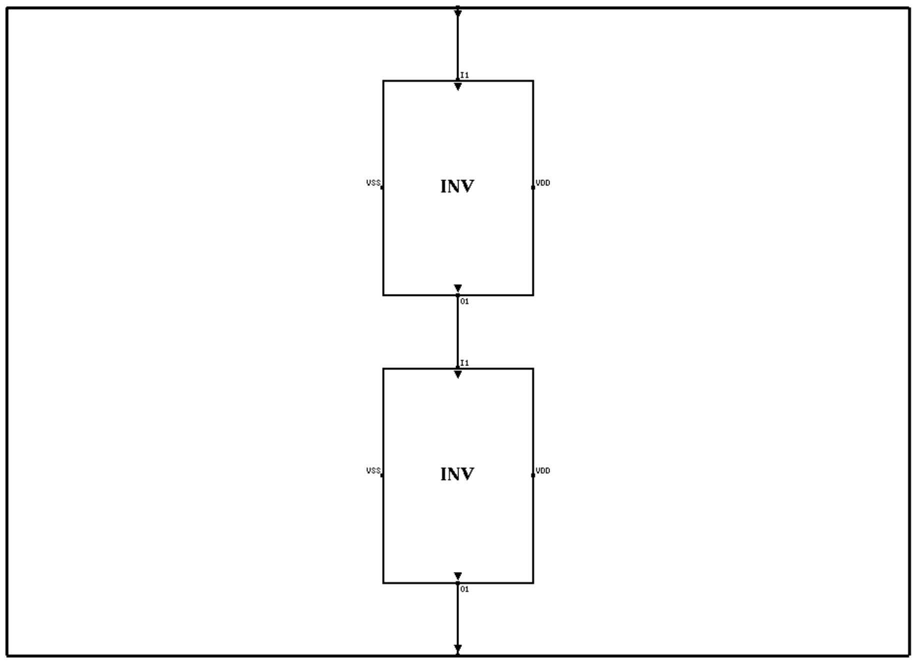

Abbildung 5.15: Die Teilschaltung Fanout1_8f1

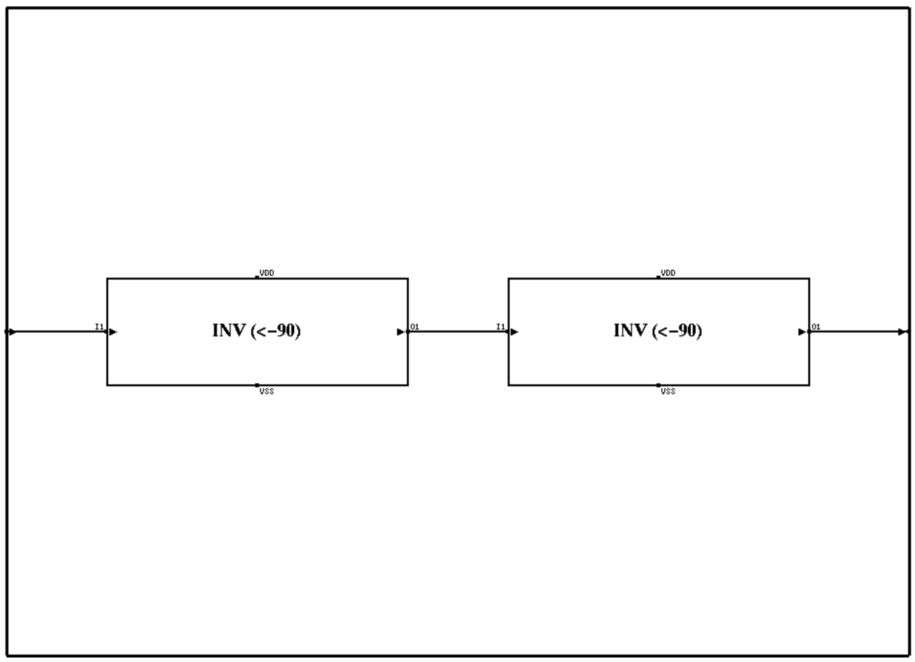

Abbildung 5.16: Die Teilschaltung Fanout1_8f2

Literaturverzeichnis

[BCD89] C.L. Berman, J. L. Carter, and K. F. Day. The Fanout Problem: From Theory to Practice. In C. L. Seitz, editor, *Advanced Research in VLSI: Proceedings of the 1989 Decennial Caltech Conference*, pages 69–99. MIT Press, März 1989.

[Bü90] Thomas Büchner. *Kursmaterial ASIC Training; Standardzellen–Bibliotheken.* Prof. Bernd Höfflinger, April 1990.

[Bur94] Thomas Burch. *Eine graphische Arbeitsumgebung für den parametrisierten Entwurf integrierter Schaltkreise.* PhD thesis, Universität des Saarlandes, 1994.

[BV94] T. Burch and W. Vogelgesang. Dokumentation der CADIC–Datenstruktur. Internes Handbuch für Programmierer, 1994.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: a Guide to the Theory of NP–Completeness.* A series of books in the mathematical sciences. W. H. Freeman and Company, 1979. 340 Seiten.

[GT88] Goldberg and R. E. Tarjan. A New Approach to the Maximum Flow Problem. *Journal of the ACM*, 35:921–940, 1988.

[Has72] E. Haselhoff. *Das TTL Kochbuch.* TID–Fachbuchreihe. Texas Instruments Deutschland GMBH, 1972. 352 Seiten.

[Has92] E. Haselhoff. *Was nicht im Datenblatt steht.* Texas Instruments Deutschland GMBH, Applikationslabor edition, 1992. 36 Seiten.

[HKP84] H. J. Hoover, M. M. Klawe, and N. J. Pippenger. Bounding Fanout in Logical Networks. *Journal of the Association for Computing Machinery*, 31(1):13–18, January 1984.

[HL87] E. Herter and W. Lörcher. *Nachrichtentechnik.* Studienbücher der technischen Wissenschaften. Carl Hanser Verlag München Wien, 1987. 428 Seiten.

[Hot74] G. Hotz. *Schaltkreistheorie.* de Gruyter Lehrbuch. Walter de Gruyter, 1974.

[KJSSV88] R. K. Brayton K. J. Singh, A. R. Wang and A. Sangiovanni-Vincentelli. Timing Optimization of Combinational Logic. *ICCAD–88*, pages 282–285, 1988.

[KMO89] R. Kolla, P. Molitor, and H. G. Osthof. *Einführung in den VLSI–Entwurf.* Leitfäden und Monographien der Informatik. B.G. Teubner Verlag, Stuttgart, 1989.

[Kol86] R. Kolla. *Spezifikation und Expansion logisch topologischer Netze.* PhD thesis, Fachbereich Informatik, Universität des Saarlandes, 1986.

[Kre93] M. Kreutzer. Implementierung des Algorithmus von Singh und Sangiovanni–Vincentelli zur Lösung des Fanout–Problems. Dokumentation zum Fortgeschrittenenpraktikum, 1993.

[Kre94] M. Kreutzer. Implementierung des algorithmus „Rectilinear shortest paths with Rectangular Barriers“. Erläuterung der Implementierung sowie der Modifikationen, 1994.

[Law76] E. L. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt, Rinehart and Winston, 1976.

[Mic87] G. De Micheli. Perfomance–Oriented Synthesis of Large–Scale Domino CMOS Circuits. In *IEEE Transactions on CAD*, pages 751–765, 1987.

[Mol86] P. Molitor. *Über die Bikategorie der logisch topologischen Netze und ihre Semantik.* PhD thesis, Fachbereich Informatik, Universität des Saarlandes, 66041 Saarbrücken, 1986.

[MP95] S. M. Müller and W. J. Paul. *The complexity of simple computer architectures.* Lecture Notes in Computer Science. Springer Verlag Berlin Heidelberg New York London Paris Tokyo Hong Kong Barcelona Budapest, 1995. 270 Seiten.

[Näh93] Stefan Näher. *LEDA Manual Version 3.0.* Max-Planck-Institut für Informatik, Saarbrücken, 1993.

[PP89] P. G. Paulin and F. J. Poirot. Logic Decomposition Algorithms for the Timing Optimization of Multi–Level Logic. In *Proceedings of ICCD 89*, pages 329–333, 1989.

[Rit94] J. Ritter. Netzlisten Expansion in CADIC. Dokumentation zum Fortgeschrittenenpraktikum, 1994.

[Rit96] J. Ritter. Flexible Visualisierung von Entwurfsdaten in CADIC. Master's thesis, Fachbereich Informatik, Universität des Saarlandes, 1996. to appear.

[RLW85] P. J. Rezende, D. T. Lee, and Y. F. Wu. Rectilinear shortest Paths with Rectangular Barriers. *ACM*, pages 204–213, 1985.

[RWG77] A. E. Ruehli, P.K. Wolff, and G. Goertzel. Analytical Power/Timing Optimization Techniques for Digital Systems. In *Proc. 14th DAC*, Juni 1977.

[Sch92] C. Schmitz. Signallaufzeitenanalyse in CADIC. Dokumentation zum Fortgeschrittenenpraktikum, 1992.

[SE90] J. Siegel and H. Eichele. *Hardewareentwicklung mit ASIC.* Mikroelektronik. Hüthig Buch Verlag Heidelberg, 1990. 293 Seiten.

[Skl60] J. Sklansky. Conditional–sum addition logic. *IRE–EC*, pages 9:226–231, 1960.

[SS93] W. Schiffmann and R. Schmitz. *Technische Informatik 1.* Springer Verlag Berlin Heidelberg New York London Paris Tokyo Hong Kong Barcelona Budapest, 1993. 282 Seiten.

[SSV90] K. J. Singh and A. Sangiovanni-Vincentelli. A Heuristic Algorithm for the Fanout Problem. In *27th ACM/IEEE Design Automation Conference*, pages 357–360. IEEE, 1990.

[SW94] M. Stoer and F. Wagner. A Simple Mincut Algorithm. *Algorithms - ESA '94*, LNCS 855:141–147, 1994.

[VLS90] VLSI Design Institut für Mikroelektronik Stuttgart. *GATE FOREST 2μ CMOS Standard Cell Library Version 1.1*, April 1990.

[Weg87] I. Wegener. *The Complexity of Boolean Functions.* Wiley–Teubner Series in Computer Science. B. G. Teubner Stuttgart, 1987. 457 Seiten.