Hauptbeiträge

Informatik Spektrum
© Springer-Verlag 1992
Informatik-Spektrum (1992) 15: 5–22

Synthese von Register-Transfer-Strukturen aus Verhaltensbeschreibungen

P. Marwedel und W. Rosenstiel

Universität Dortmund und Universität Tübingen/FZI

Zusammenfassung. Die starke Zunahme der Anzahl und der Komplexität anwendungsspezifischer integrierter Schaltungen (ASICs) macht eine wesentliche Verkürzung der Entwurfszeiten und eine entsprechende Verringerung der Entwurfskosten notwendig. Üblicherweise setzt die Entwurfsautomatisierung erst mit dem Logik-Entwurf ein. Hierbei wird das gewünschte Verhalten der Schaltung in Form von Booleschen Gleichungen vorgegeben und eine Struktur generiert, die als Bausteine Gatter enthält. Ebenfalls gängig ist die Vorgabe von Strukturen, die Register-Transfer(RT)-Bausteine (z.B. Register und Addierer) als Komponenten enthalten. Es gibt Werkzeuge, die eine solche Beschreibung in eine vollständige Implementierung in verschiedenen Technologien übersetzen. Sofern bereits die RT-Strukturen nach einem standardisierten Verfahren erzeugt werden, spricht man von der sog. „High-Level"-Synthese. Derartige Verfahren sind zur Zeit Forschungsgegenstand. Der vorliegende Beitrag erläutert die Prinzipien und Verfahren, die dieser Synthese zugrunde liegen.

Schlüsselwörter: CAD, Entwurfsautomatisierung, Synthese, Hardwarebeschreibungssprachen

Summary. The considerable increase in application-specific integrated circuits necessitates an essential reduction of design time as well as design cost. Usually the design automation starts with logic design. The task to deduce a first circuit structure from a behavioural description is up to the designer. Structure synthesis – often described as "High-Level Synthesis", too – has already the aim to generate automatically a circuit structure on register transfer level from a behavioural specification. The contribution on hand explains the principles and methods, which form the basis of this structure synthesis.

Key words: CAD, Design automation, Synthesis, Hardware description languages, High-level synthesis, Scheduling, Allocation, Assignment

Computing Reviews Classification: B.5.2

1. Einführung

Die weiter steigende Integrationsdichte und die zunehmende Chipfläche werden in den nächsten 10–20 Jahren integrierte Schaltkreise mit mehr als einer Milliarde Transistoren ermöglichen. Derart komplexe integrierte Schaltkreise machen die Implementierung vollständiger Systeme auf einem einzigen Chip möglich. Diese Entwicklung führt einerseits zu wachsender Spezialisierung und damit zu einem immer stärker dominierenden Entwurfskostenanteil. Die Konsequenz aus dieser Entwicklung ist die verstärkte Standardisierung der Chipstrukturen in Form von sog. Zellenschaltungen, Gate-Arrays, Sea-of-Gates oder den zunehmend wichtiger werdenden programmierbaren Gate-Arrays. Aus den gleichen Gründen ist eine verstärkte Tendenz zur Entwurfsautomatisierung und dabei ein Übergang zu Entwurfsverfahren zu beobachten, die auf immer höheren Abstraktionsebenen beginnen. Tabelle 1 zeigt mögliche Abstraktionsebenen digitaler Systeme im Zusammenhang.

Beim Entwurf eines digitalen Systems wird eine Beschreibung auf einer niedrigen Abstraktionsebene aus einer Spezifikation auf einer höheren Ebene erzeugt. Im wesentlichen sind dazu zwei Verfahren möglich: Der Entwurf per Hand mit anschließender Verifikation der „Im-

Tabelle 1. Mögliche Abstraktionsebenen digitaler Systeme

Name der Ebene	Komponenten
Systemebene	Prozessoren, Speicher etc.
algorithmische Ebene	Speicherzellen (Variable), Zuweisungen, Schleifen
RT-Ebene	ALUs, Busse, RAMs, Register
Logik-Ebene	Boolesche Ausdrücke, NAND- und NOR-Gatter
Schalter-Ebene	Schalter, Abschwächer, Verbindungen (Komponenten der CSA-Theorie [32])
elektrische Ebene	Kondensatoren, Widerstände, Strom- und Spannungsquellen in Ersatzschaltbildern von Transistoren (z.B. im SPICE-Modell [69])

plementierung“ gegen die Spezifikation und die Synthese auf der Grundlage der Spezifikation.

Def.: Verifikation ist der Nachweis, daß ein aus Komponenten einer niedrigeren Ebene zusammengesetztes System ein auf einer höheren Ebene beschriebenes Verhalten zeigt.
Bem.: Der manuelle Entwurf wird nicht durch ein vorgegebenes Syntheseverfahren eingeschränkt. Die Verifikation wurde bislang meist durch Simulation von Fallbeispielen ersetzt, kann aber zunehmend auch formal durchgeführt werden.

Def.: Synthese ist das Zusammensetzen von Komponenten oder Teilen zu einem Ganzen einer niedrigeren Ebene mit dem Ziel, ein auf einer höheren Ebene beschriebenes Verhalten zu erzielen.
Bem.: Der Korrektheitsbeweis per Verifikation ist überflüssig, falls das Syntheseverfahren selbst und seine Implementierung korrekt sind.

In diesem Zusammenhang kommt der sogenannten „High-Level“-Synthese wachsende Bedeutung zu. Mit „High-Level“-Synthese werden standardisierte und in der Regel automatisierte Verfahren des Entwurfs von RT-Strukturen bezeichnet. Derartige Verfahren gehen von einer abstrakten Beschreibung des gewünschten Verhaltens aus. Ihre Vorteile liegen in niedrigeren Entwurfskosten und weniger Entwurfsfehlern. Die Resultate der Schaltungssynthese, also die generierten Schaltungen, sind bislang meist ungünstiger in Bezug auf Fläche und Verarbeitungsgeschwindigkeit als die von erfahrenen (menschlichen) Chip-Designern entworfenen Schaltungen. Daher gibt es bis heute nur wenige industrielle Anwendungen [38]. Durch intensive Forschungsanstrengungen wird derzeit versucht, diesem Mangel abzuhelfen.

Nach einer kurzen Einführung in Sprachen zur Beschreibung des Verhaltens digitaler Schaltungen beschäftigt sich der vorliegende Beitrag mit den wesentlichen Syntheseschritten und den jeweils eingesetzten Verfahren und Algorithmen sowie der Erläuterung anhand von Beispielen. Die Syntheseaufgabe läßt sich relativ einfach wie folgt charakterisieren: Eine Verhaltensbeschreibung beschreibt den Zusammenhang zwischen Ein- und Ausgangssignalen in Form von auszuführenden Operationen sowie Datenfluß- und Kontrollfluß-Abhängigkeiten. Operationen sind dabei die einzelnen arithmetischen/logischen Operationen sowie die Schreib-/Lese-Operationen für die Variablen. Ausgehend von dieser Beschreibung legt die Synthese die genaue zeitliche Abfolge dieser Operationen fest (*Ablaufplanung, „scheduling“*), bestimmt die Anzahl und die Typen der zu verwendenden Verarbeitungs-, Speicher- und Verbindungs-Elemente (*Bereitstellung von Ressourcen, „allocation“*) und ordnet Operationen, Variablen und Datentransporten entsprechende Elemente zu (*Zuordnung, „assignment“*).

2. Spezifikationssprachen

Zur Spezifikation des Verhaltens digitaler Systeme sind im Prinzip mehrere Ansätze möglich. So könnte man von einer rein funktionalen (im Sinne der Mathematik) Beschreibung ausgehen, die keinerlei Angaben über Ablauf der Berechnung von Ausgangswerten m In Spezialfällen, z.B. beim Entwurf von sog. Signa zessoren, wird dies auch gemacht. Die am IME Leuven entwickelten CATHEDRAL-Synthesewerl ge verwenden zur Spezifikation die funktionale Spı SILAGE [18]. Meist werden jedoch imperative Spra auf der Spezifikationsebene benutzt, die neben bekannten Nachteilen auch einige Vorteile haben, die Möglichkeit, den Ablauf einer Berechnung r genau anzugeben. Beim gegenwärtigen Stand der nik ist die Qualität der erzeugten Entwürfe besser, eine recht detaillierte Berechnungsvorschrift vo [50].

Es wurden zahlreiche überwiegend imperative chen entwickelt, die sich zur Beschreibung des Verh von Hardwaresystemen eignen. Von vielen zur Pro mierung eingesetzten Sprachen wurden Varianten (Regel Untermengen) zur Spezifikation des Verh von Hardware verwendet (siehe z.B. [74]). Der V dieses Ansatzes liegt darin, daß eine derartige Bes bung nach Compilation sofort simuliert werden un spezielle Benutzerschulung entfallen kann.

Für ausgereifte Entwurfssysteme kann dagege spezielle „Hardwaresprachen“ nicht verzichtet w So werden z.B. flexible Methoden zur Beschreibur Nebenläufigkeiten, von Hardwarestrukturen usw. tigt, die in üblichen Programmiersprachen nicl Verfügung stehen. Es gibt mittlerweile eine Füll Hardwarebeschreibungssprachen, z.B. DDL [21], [5], DSL [66, 13], CONLAN [63], KARL [31], D PO [20], HSL-FX [55] als Basis für den japan Standard UDL/I, MIMOLA [8] usw. Um eine dardisierung in diesem Bereich zu ermöglichen, die Sprache VHDL [34] normiert. VHDL die Grundlage für die meisten Neu- und Weiterentwi gen. Allerdings wurde VHDL vor allem zur Simu entwickelt. Für Synthese- und Verifikationszweck nicht der volle Sprachumfang genutzt werden. Fü Zwecke besser geeignete, jedoch mit VHDL „ko ble“ Sprachen werden daher vermutlich weiter ren.

Die folgenden Beispiele zeigen Hardwarespe tionen in den beiden von den Autoren mitentwic Sprachen zur Schaltungssynthese, nämlich MIN (*m*achine *i*ndependent *m*icro programming *la*n und DSL (*d*igital system *s*pecification *l*anguage) rend das MIMOLA-Entwurfssystem primär Pro und Kontrollfluß-orientierte Strukturen unterstüt das auf DSDL/DSL basierende CALLAS/CADI Datenfluß-orientierte Strukturen. Entsprechend im folgenden die Beispiele ausgewählt. Zwischen Systemen gibt es aber Überschneidungen.

1. Spezifikation in MIMOLA

Ein programmierbarer Prozessor besitze einen speicher **main** mit 64 k Zellen und einen Registers **reg** mit 4 Zellen; er benutzt das Befehlsformat der

Die Opcodes 0 bis 3 seien zur Codierung von port- (load, store), Additions- und Subtraktionsbe

Bits:	15....13	12..........11	10......0
Bedeutung:	Opcode	Registernummer	Distanz

Abb. 1. Befehlsformat

nutzt. Dabei seien alle Adressen relativ zum Basisregister **reg[2].** Register 3 sei der Programmzähler.

Ein Interpreter für diesen Befehlssatz kann wie folgt formuliert werden:

```
PROGRAM interpreter IS
 LABEL Loop;
 TYPE word = (15:0);              -- 16-Bit-Wort
      irtyp=FIELDS                -- Befehlformat
             Opcode : (15:13); --Bits 15:13
             RegNr  : (12:11); --Bits 12:11
             Dist   : (10: 0); --Bits 10:0
            END;
 VAR  main : ARRAY [0..#FFFF] OF word;
      reg  : ARRAY [0..3]     OF word;
      ir   : irtyp;        -- Befehlsregister
 BEGIN
 :Loop:
  ir      := main[ reg[3] ];
  reg[3] := reg[3] +1;    -- Programmzaehler
  CASE ir.Opcode OF
   0: reg[ir.RegNr] :=             -- ''load''
       main[reg[2]+ir.Dist];
   1: main[reg[2]+ir.Dist] :=    --''store''
       reg[ir.RegNr];
   2: reg[ir.RegNr] :=             --''add''
       reg[ir.RegNr]+main[reg[2]+ir.Dist];
   3: reg[ir.RegNr] :=             --''sub''
       reg[ir.RegNr]-main[reg[2]+ir.Dist];
  END;
  GOTO Loop;
 END;
```

2. Spezifikation in DSL

Das folgende Beispiel beschreibt eine Schaltung zum Bestimmen der Anzahl der Wörter in einer Datei.

Derartige Verhaltensbeschreibungen werden *algorithmische Verhaltensbeschreibungen* genannt.

Viele Strukturen können ein algorithmisch vorgegebenes Verhalten zeigen. Beispielsweise könnte man die Verhaltensbeschreibungen für praktisch alle klassischen Rechner compilieren. Eine derartige Lösung wäre aber in der Regel zu langsam oder – wegen notwendiger Umgebung in Form von Speichern, Peripheriebausteinen usw. – zu aufwendig. Es kommt daher darauf an, den Entwurfsraum auf die Menge der interessierenden Lösungen zu begrenzen. Diesem Zweck dienen, als zweiter Teil der Spezifikation, die *Entwurfsrestriktionen.* Als Entwurfsrestriktionen sind sinnvoll:

- die zu erzielende Rechenleistung,
- Zeitbedingungen für Ein-/Ausgangssignale (siehe z. B. [1]),
- die maximalen Kosten,
- die technologischen und technischen Randbedingungen (Fertigungstechnologie, Zahl verfügbarer Anschlüsse usw.),
- Aspekte der Testbarkeit und der Zuverlässigkeit der Strukturen (siehe z. B. [68, 72, 36, 27]),
- Verwendung bestimmter Bausteinbibliotheken.

Manche Synthesesysteme enthalten darüber hinaus noch *implizite* Restriktionen, da sie speziell zur Synthese von

```
CIRCUIT WORDCOUNT;
  CONSTANT
   INTMAX        = 15;
   TRUE          = 1;
   FALSE         = 0;
   ENDSTR        = H'00';
   NEWLINE       = H'0A';
   ENDFILE       = H'03';
   BLANK         = H'20';
   TAB           = H'09';
  INTERFACE
   CLK           : CLOCK;
   C_IN(7..0)    : INPUT;
   NW_OUT(7..0)  : OUTPUT;
  PERFORMED FUNCTION
   NW_OUT := #WORDCOUNT(C_IN);
  END;

  VAR C          : LOGICAL(7..0);
      NW         : TWO_COMP(INTMAX..0);
      INWORD     : LOGICAL;
   SEQUENTIAL WORDCOUNT;
    NW     :=0;
    INWORD := FALSE;
    C      := C_IN;
    WHILE (C<>ENDFILE) DO
     IF (C=BLANK)OR(C=NEWLINE)OR(C=TAB)
      THEN INWORD := FALSE
      ELSE
       IF NOT INWORD THEN
         INWORD := TRUE; NW := NW + 1
       FI
     FI;
     C := C_IN;
    OD;
    NW_OUT := NW;
   END SEQUENTIAL
END.
```

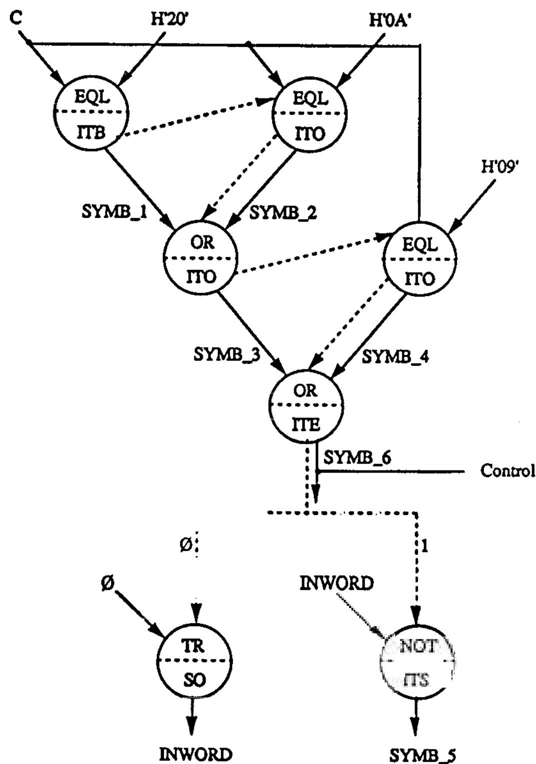


Abb. 2. Beispiel für einen Flußgraphen (Ausschnitt für obiges DSL-Programm) —▶ Datenabhängigkeiten; ----▶ Kontrollflußabhängigkeiten

Busarchitekturen [24], bit-seriellen Architekturen [17] oder für einen bestimmten Satz verfügbarer Bausteintypen [18] entwickelt wurden. Üblich ist ferner die Beschränkung auf *synchrone* Strukturen, d.h. Strukturen, die taktgesteuert Übergänge zwischen internen *Zuständen* ausführen. In jedem Zustand werden die Hardware-Bausteine dabei in ihrer Funktion durch *Steuersignale* kontrolliert, daher werden diese Zustände auch als *Kontrollschritte* bezeichnet.

3. Synthese von Register-Transfer-Strukturen

In den vergangenen 15 Jahren wurden viele Möglichkeiten zur automatischen Generierung von Registertransfer-Strukturen aus algorithmischen Verhaltensbeschreibungen entwickelt. Die eigentliche Aufgabe der Synthese besteht weniger in der Erzeugung „irgendeiner" verhaltensgleichen Struktur, sondern insbesondere darin, neben der Verhaltensgleichheit weitere, in erster Linie nicht-funktionale Eigenschaften der generierten Struktur zu erreichen. Diese Eigenschaften betreffen zum einen die geforderte Leistung und zum anderen die Einhaltung einer bestimmten Chipfläche bei der Implementierung der generierten Strukturen als anwendungsspezifische integrierte Schaltungen. Weitere nicht-funktionale Eigenschaften betreffen die Testbarkeit oder auch die Zuverlässigkeit der generierten Schaltungsstrukturen. Die entwickelten Syntheseverfahren und -systeme versuchen, diese Anforderungen durch verschiedene Ansätze zu erreichen. Da die Umsetzung einer Verhaltens- in eine Strukturbeschreibung bei beschränkten Ressourcen ein NP-hartes Problem ist, versucht man, durch die Unterteilung des Gesamtproblems in mehrere Einzelprobleme die Suche nach effizienten Näherungslösungen zu vereinfachen [53]. Als hierzu geeigneter Ansatz wird heute im wesentlichen die Unterteilung des Syntheseproblems in Teilprobleme gesehen.

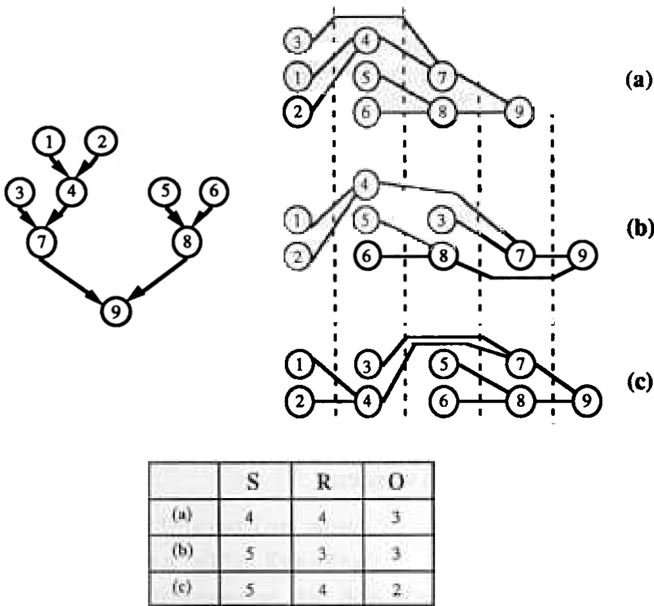
Um die eigentlichen Syntheseschritte von der Verarbeitung einer speziellen Eingabesprache abzukoppeln, verwenden die meisten der heutigen Synthesesysteme ein internes Zwischenformat. Beispiele sind der Value-Trace der an der CMU entwickelten Synthesesysteme [51], BIF [22], TREEMOLA [9] oder auch der sog. Flußgraph des CADDY-Synthesesystems [13]. Dem Flußgraph-Format sehr ähnlich sind die Formate YIF (Yorktown Intermediate Format) des bei IBM Yorktown entwickelten Synthesesystems [6] sowie SIF (Stanford Intermediate Format) des in Stanford entwickelten Synthesesystems [43]. Als Beispiel enthält Abb. 2 den Flußgraphen für einen Teil der obigen algorithmischen Spezifikation „WORD-COUNT".

Bevor derartige Flußgraphen erstellt werden können, werden von den Synthesesystemen in der Regel einige höhere Sprachelemente (z.B. Prozeduraufrufe) durch andere Sprachelemente (z.B. Zuweisungen) ersetzt. Diese *Programmtransformationen* sind in einigen Fällen fest in die Synthesesoftware eingebaut, in anderen Fällen durch Regelsysteme oder Vorcompiler flexibel gehalten. Ebenso ist es vom Synthesesystem abhängig, welche Sprachelemente ersetzt werden und welche von den späteren Phasen im Syntheseprozeß verarbeitet werden.

Die verbleibenden Syntheseschritte teilt man heute in der Regel auf in

- die *Ablaufplanung* („scheduling"),
- die *Bereitstellung* bzw. *Zuteilung* von Ressourcen („allocation") und
- die *Zuordnung* („assignment").

Unter Ablaufplanung versteht man die Zuweisung von Operationen der Verhaltensbeschreibung zu Kontrollschritten. Aus diesem Grund wird die Ablaufplanung auch als „operation to control step binding" bezeichnet. Die Ablaufplanung entscheidet über den Parallelitätsgrad der generierten Struktur und ist eine Näherungslösung des Problems, eine möglichst schnelle Schaltung unter gegebenen Flächenrestriktionen bzw. umgekehrt eine möglichst kostengünstige Schaltung unter Leistungsrestriktionen zu generieren. Für die Synthese bereitzustellende Ressourcen werden unterschieden in Verarbeitungseinheiten (auch „Prozessoren" genannt), Speicherelemente (Register, Registerbänke, Ein- und Mehrportspeicher etc.) und Verbindungsstrukturen (Multiplexer, Busse, Anschlüsse eines Chips nach außen etc.). Die Zuordnung entscheidet, welche Operationen von welchen Verarbeitungseinheiten ausgeführt werden, welche Werte bzw. Variablen in welchen Registern gespeichert werden und wie Verarbeitungseinheiten und Regi-



	S	R	O
(a)	4	4	3
(b)	5	3	3
(c)	5	4	2

Abb. 3. Abhängigkeiten zwischen Ressourcen-Bereitstellung und Ablaufplanung

ster durch die zugeordneten Verbindungseinheiten verbunden werden können. Die Zuordnung läßt sich also unterteilen in die von Verarbeitungseinheiten (Prozessor-Zuordnung), die Register-Zuordnung und die Verbindungs-Zuordnung. Die Lösung des Zuordnungsproblems hat erheblichen Einfluß auf die Verbindungskosten und damit sowohl auf die Fläche wie auch auf die Geschwindigkeit der generierten Schaltung. Die wechselseitig sehr starke Abhängigkeit der Bereitstellung von Ressourcen und der Ablaufplanung soll durch das Beispiel in Abb. 3 [62] demonstriert werden.

Für den Datenflußgraphen sind verschiedene „Schedules" angegeben, die sich hinsichtlich der benötigten Prozessoren- bzw. Registeranzahl unterscheiden. Das Beispiel macht deutlich, wie einerseits die Anzahl der benötigten Zeitschritte von der Anzahl der bereitgestellten Register und Verarbeitungseinheiten und andererseits auch einzelne Teilprobleme innerhalb der Bereitstellung voneinander abhängen. So lassen sich im obigen Beispiel Register gegen Verarbeitungseinheiten „austauschen" und umgekehrt. Da die Verbindungskosten (die großen Einfluß auf die Gesamtkosten haben) durch die Zuordnung bestimmt werden, ist es auch außerordentlich schwierig, Ablaufplanung und Ressourcen-Bereitstellung unabhängig von der Zuweisung zu behandeln. Wegen der starken Abhängigkeiten der Teilprobleme wurde in letzter Zeit versucht, das Syntheseproblem mit genetischen Algorithmen [75] sowie mit *Simulated Annealing* [16] zu lösen. In anderen Ansätzen werden geschlossene Algorithmen für die Lösung der o. a. Teilprobleme angegeben, wobei die Teilprobleme im allgemeinen streng sequentiell gelöst werden. Die gegenseitigen Abhängigkeiten versucht man durch geeignete Heuristiken und Bewertungsfunktionen zu berücksichtigen.

Im folgenden werden die Probleme und Lösungsansätze der drei Teilprobleme der Synthese, nämlich der Ablaufplanung, der Bereitstellung von Ressourcen sowie der Zuordnung, dargestellt, ohne die gegenseitigen Abhängigkeiten außer acht zu lassen.

3.1. Ablaufplanung (scheduling)

Die Beschreibung der Scheduling-Verfahren orientiert sich aus Gründen der Übersichtlichkeit an der Darstellung der Datenabhängigkeitsrelationen mittels expliziter Datenflußgraphen. Ein einfaches Scheduling-Verfahren, auf dessen weitergehende Erläuterung hier verzichtet werden soll, ist z. B.

- FCFS-Scheduling (first come, first served):
Bei diesem Verfahren werden lediglich die Datenabhängigkeiten berücksichtigt. Die bereiten Operationen werden in der Reihenfolge eingeplant, in der sie bearbeitet werden.

Beim FCFS-Scheduling gibt es – wie bei manchen anderen Schedulingverfahren auch – zwei Varianten:

- ASAP-Scheduling (as soon as possible):
Unter Berücksichtigung der Datenabhängigkeiten werden die Operationen $v \in V$ zum frühestmöglichen Zeitpunkt $e(v)$ eingeplant.
- ALAP-Scheduling (as late as possible):
Bei diesem Verfahren werden alle Operationen unter Berücksichtigung der Datenabhängigkeiten zum spätestmöglichen Zeitpunkt, der sich aus einer bestimmten oberen Zeitschranke ergibt, eingeplant. Oft wird als diese Zeitschranke die minimale Schrittanzahl zur Abarbeitung eines Datenflußgraphen (auch als „Schedule-Länge" F bezeichnet) gewählt. In diesem Fall ergibt sich der spätestmögliche Zeitpunkt der Operation v zu $l_F(v)$.

Die Differenz zwischen $l_F(v)$ und $e(v)$ wird Ausführungsintervall $I_F(v)$ genannt:

$$I_F(v) = l_F(v) - e(v) + 1$$

Entsprechend ergibt sich die Mobilität einer Operation zu $I_F(v) - 1$.

Sowohl das ALAP- wie auch das ASAP-Scheduling gehen von unbeschränkten Ressourcen aus. Sie liefern zwar die schnellstmögliche Implementierung, können aber auf Grund der angenommenen Unbeschränktheit der Ressourcen lediglich zum Auffinden von Schranken benutzt werden. Die im folgenden etwas näher erläuterten Ablaufplanungsstrategien versuchen, unter Ressourcenbeschränkung die schnellstmögliche bzw. unter Zeitbeschränkungen die kostengünstigste Lösung zu finden (i. a. bezogen auf die Anzahl bereitgestellter, aber noch nicht zugeordneter Ressourcen).

3.1.1. Kräfteverfahren (force directed scheduling)

Das in [57] vorgestellte Ablaufplanungsverfahren versucht, unter gegebenen Zeitbeschränkungen die Anzahl der benötigten Verarbeitungseinheiten durch eine möglichst gute und gleichmäßige Auslastung zu minimieren.

Die zugrunde liegende Modellvorstellung, die auch bei Plazierungsverfahren verwendet wird, greift auf eine physikalische Analogie zurück: Zwischen Kontrollschritten werden, bezogen auf eine bestimmte Verarbeitungseinheit, „Kräfte" eingeführt, deren Stärke von der unterschiedlichen Auslastung der Operationseinheiten in den verschiedenen Kontrollschritten abhängt. Die auf diesem Kräfteverfahren basierende Ablaufplanung versucht nun, durch Verschieben von Operationen zwischen Kontrollschritten diese Kräfte zu minimieren. Dadurch wird eine möglichst gleichmäßige Auslastung aller Verarbeitungseinheiten erreicht und eine Minimierung der Verarbeitungseinheiten erzielt.

Es wird davon ausgegangen, daß Operationen für jeden Zeitpunkt zwischen dem frühest- und dem spätestmöglichen mit gleicher Wahrscheinlichkeit eingeplant werden können. Aus dieser Annahme ergeben sich Wahrscheinlichkeiten für die Einplanung zu einem bestimmten Zeitpunkt. Das Kräfteverfahren geht nun im einzelnen wie folgt vor:

Zunächst werden die Operationen eingeplant, deren frühestmöglicher Zeitpunkt mit dem spätestmöglichen zusammenfällt. Diese Operationen liegen also offensichtlich auf dem kritischen Pfad, während für alle anderen Operationen eine Verschiebung innerhalb dieser Zeitspanne möglich ist, ohne den gesamten Ablauf zu verzögern.

In einem iterativen Verfahren zur Festlegung der Einplanung dieser freien Operationen wird versucht, die vorhandenen Komponenten möglichst gut auszunutzen, was einer Optimierung der Anzahl der benötigten Komponenten entspricht. Dabei werden mit derselben Komponente implementierbare Operationen verschiedenen Kontrollschritten zugewiesen.

Da auch der später vorgestellte und in CADDY verwendete Ablaufplanungsalgorithmus CASCH ähnliche Prinzipien verwendet, sollen die wichtigsten Begriffe hier kurz eingeführt werden, wobei auf die Charakterisierung nach [71] zurückgegriffen wird.

Wahrscheinlichkeit

Die Wahrscheinlichkeit, daß eine Operation an irgendeiner Stelle innerhalb des möglichen Ausführungsintervalls eingeplant wird, sei über alle Kontrollschritte in diesem Intervall gleichmäßig verteilt. $P(v,s)$ ist definiert als die Wahrscheinlichkeit, daß eine Operation v im Kontrollschritt s eingeplant wird. Wenn alle Operationen einen Kontrollschritt benötigen, sich also keine Operation über mehrere Kontrollschritte erstreckt, ist diese Wahrscheinlichkeit definiert als:

$$P(v,s) = \begin{cases} \frac{1}{I_F(v)} & \text{falls } e(v) \le s < l_F(v) \\ 0 & \text{sonst} \end{cases}$$

Verteilungsfunktionen

Für jede Typmenge p definiert eine Funktion $DF_p(s)$ („distribution function") die Dichte der Operationen v im Kontrollschritt s, deren Typ $t(v)$ in dieser Typmenge enthalten ist. Obwohl diese „distribution function" keine Verteilungsfunktion im strengen Sinne ist, soll diese eingeführte Begriffsbildung hier dennoch übernommen werden. Für einen Operationstyp t ist diese Verteilungsfunktion definiert als die Summe der Wahrscheinlichkeiten $P(v,s)$ aller Operationen v, die auf diesem Operationstyp auszuführen sind.

$$DF_p(s) = \sum_{t(v) \in p} P(v,s)$$

In Kontrollschritten, in denen die Verteilungsfunktion für einen gegebenen Operationstyp einen hohen Wert hat, sind demnach parallele Komponenten erforderlich. Um diese Parallelität zu reduzieren, müssen Operationen zu Kontrollschritten mit niedrigerem Wert der Verteilungsfunktion verschoben werden.

Kräfte

Wenn eine Operation v in einem festen Kontrollschritt k eingeplant wurde, gibt die Kraft $SF(v,k)$ den Einfluß dieser Festlegung auf die Verteilung wieder.

$$SF(v,k) = \sum_{s \in S} DF_p(s) * \Delta P_{v,k}(v,s)$$

$\Delta P_{v_i,k}(v_j, s)$ bezeichnet die Änderung der Wahrscheinlichkeit für die Einplanung der Operation v_j im Kontrollschritt s, wenn die Operation v_i im Kontrollschritt k eingeplant wird. Falls $v_j = v_i$, ergibt sich:

$$P_{v,k}(v,s) = \begin{cases} 1 - P(v,s) & \text{falls } s = k \\ P(v,s) & \text{sonst} \end{cases}$$

$SF(v,k)$ gibt für die Operation v selbst an, wie sich die Verteilung ändert, wenn diese Operation im Kontrollschritt k eingeplant wird. Diese Einplanung kann sich allerdings auch auf die vorhergehenden und nachfolgenden Kontrollschritte auswirken, was sich mit sog. Vorgänger- bzw. Nachfolgerkräften ausdrücken läßt, die zur Bestimmung der Gesamtkraft dann zu SF addiert werden müssen. Die Vorgänger- bzw. Nachfolgerkräfte sind definiert als:

$$PredF(v,k) = \sum_{v_i \in Vorg.(v)} \sum_{s \in S} DF_{p_i}(s) * \Delta P_{v,k}(v_i,s),$$

wobei $Typ(v_i) \in p_i$,

$$SuccF(v,k) = \sum_{v_i \in Nachf.(v)} \sum_{s \in S} DF_{p_i}(s) * \Delta P_{v,k}(v_i,s),$$

wobei $Typ(v_i) \in p_i$.

Der Einfluß der Einplanung von Operation v im Kontrollschritt k auf die Gesamtverteilung der Operationen kann jetzt als die Gesamtkraft $F(v,k)$ definiert werden

$$F(v,k) = SF(v,k) + PredF(v,k) + SuccF(v,k)$$

Mit diesen Grundlagen kann nun der Algorithmus zu Ablaufplanung nach dem Kräfteverfahren formuliert werden.

Algorithmus des Kräfteverfahrens

Zunächst werden aufgrund der ASAP- und ALAP-Ergebnisse die möglichen Ausführungsintervalle und damit die Verteilungsfunktionen berechnet. In jeder Iterationsschleife des Algorithmus wird eine Operation ausgewählt, und die Kräfte werden entsprechend obiger Formeln berechnet. Als Ergebnis wird die Operation v in dem Kontrollschritt k mit dem kleinsten $F(v,k)$ eingeplant:

```
Prozedur Kraefteverfahren
ASAP-scheduling;
ALAP-scheduling;
Berechne Verteilungsfunktionen;
WHILE nicht alle Operationen eingeplant
 DO
  waehle Operation aus;
  berechne alle Kraefte;
  plane Operation in dem Kontroll-
  schritt mit niedrigster Kraft ein;
  berechne Ausfuehrungsintervalle neu;
  berechne Verteilungsfunktionen neu;
ENDWHILE;
```

Ein Nachteil des Kräfteverfahrens ist, daß sich universelle Verarbeitungseinheiten, also Verarbeitungseinheiten, die verschiedene Operationen (u. U. mit einer verschiedenen Anzahl von Steuerungsschritten) ausführen können, schwer berücksichtigen lassen. In den folgenden Kapiteln wird daher ausführlicher auf Erweiterungen und Modifikationen des Kräfteverfahrens eingegangen. Dort finden sich dann auch Beispiele, die die Prinzipien dieser Ablaufplanungsverfahren verdeutlichen sollen. Auch Paulin selbst hat die Nachteile eines reinen Kräfteverfahrens bald erkannt und eine Modifikation als „force directed list scheduling" vorgeschlagen [58], eine Kombination des Kräfteverfahrens mit den im folgenden Kapitel beschriebenen listenorientierten Verfahren. Eine Weiterentwicklung ist die CADDY zugrunde liegende Ablaufplanungsstrategie, die im Kapitel 3.1.3 daher ausführlicher beschrieben werden soll.

3.1.2. Listenorientierte Ablaufplanung (list scheduling)

Die meisten der heute verwendeten Ablaufplanungsverfahren sind listenorientiert. Sie gehen auf den Algorithmus von Hu [3] zurück (Abb. 4). Dieser Algorithmus liefert offensichtlich dann ein optimales Ergebnis, wenn das Auswahlproblem „select $V_d \subseteq V$" optimal gelöst ist, bei dem eine Teilmenge bereiter Operationen in Abhängigkeit von der Anzahl der zur Verfügung stehenden Ressourcen ausgewählt wird. Da die optimale Auswahl einer Teilmenge von im nächsten Schritt einzuplanenden Operationen NP-hart ist, ist eine effiziente Lösung nur unter Verwendung von Heuristiken möglich. Folgende Heuristiken werden eingesetzt:

- Heuristik des kritischen Pfads (critical path scheduling):
Die Heuristik des kritischen Pfads, auch Mobilitätsheuristik genannt, will das Auswahlproblem des Hu-Algorithmus dadurch lösen, daß die einzuplanenden bereiten Operationen nach ihrer Mobilität sortiert werden. Da aufgrund von Ressourcen-Restriktionen nicht alle bereiten Operationen eingeplant werden können, versucht die Mobilitätsheuristik, wenigstens die Operationen auf dem kritischen Pfad einzuplanen, deren Mobilität aufgrund der Definition 0 ergibt. Bei knappen Ressourcen führt dieser Algorithmus allerdings nicht zu besonders guten Lösungen, weil dann die Mobilität als Auswahlkriterium keine gute Entscheidungshilfe ist.
- Lebenszeit-Heuristik:
Die Lebenszeit-Heuristik hat das Ziel, eine gute Lösung für das Auswahlproblem zu finden, indem die Anzahl der benötigten Register minimiert wird. Mit einer Lebenszeit-Analyse der Eingangswerte einer Operation kann festgestellt werden, ob durch das Einplanen der Operation die Register frei werden, in denen die Eingangswerte dieser Operation gespeichert sind. Das Auswahlproblem wird also bevorzugt dadurch gelöst, daß die Operationen dann eingeplant werden, wenn die Lebenszeiten der Eingangswerte dieser Operation in dem vorliegenden Kontrollschritt enden.
- Vorausschau-Heuristik (look ahead):
Während die bisher beschriebenen Heuristiken für die Lösung des Auswahlproblems im wesentlichen „greedy"-Strategien verfolgen und insbesondere keine Möglichkeit bieten, nachteilige Folgen einer einmal getroffenen Auswahl für spätere Kontrollschritte zu berücksichtigen, sollen die vorausschauenden Verfahren die Konsequenzen einer Auswahl für die Folgeschritte vorher bestimmen und berücksichtigen. Diese Verfahren sind zwar aufwendiger, führen aber in der Praxis zu wesentlich besseren Ergebnissen. Beispiele für solche Vorausschau-Heuristiken sind die Konflikt-Reduktions-Heuristik oder die Verzögerungs-Reduktions-Heuristik.
- Konflikt-Reduktions-Heuristik:
Konflikte sind definiert als die Differenz zwischen der Anzahl der bereiten Operationen und der Anzahl der zur Verfügung stehenden Ressourcen. Die Konflikt-

```
Procedure Hu (V,k)

V: operations in the data flow graph
k: available resources

BEGIN
      Thislevel := 0
      WHILE V ≠ nil DO
            BEGIN
                  select Vd ⊂ V, where Vd is a set of
                  nodes without ancestors in V and |Vd| ≤ k
                  V := V - Vd
                  Schedule(Thislevel) := Schedule(Thislevel) ∪ Vd
                  Thislevel := Thislevel + 1
            END
      ENDWHILE
END
```

Abb. 4. Der Algorithmus von Hu

Reduktions-Heuristik versucht nun abzuschätzen, wieviele Konflikte sich in späteren Kontrollschritten unter der Annahme einer bestimmten Auswahl ergeben. Die Folgekonflikte können durch ein ASAP-Scheduling ohne Ressourcen-Beschränkung sehr effizient abgeschätzt werden. Die Konflikt-Reduktions-Heuristik wurde implementiert [42], und praktische Erfahrungen mit diesem Verfahren haben insbesondere bei knappen Ressourcen Vorteile gezeigt.

3.1.3. Die Verzögerungs-Reduktions-Heuristik

Die Verzögerungs-Reduktions-Heuristik hat im praktischen Einsatz bisher die besten Ergebnisse gezeigt. Der Algorithmus bestimmt wie oben diejenige Teilmenge der bereiten Operationen, für die die Bewertung am besten ausfällt. Zusätzlich muß die Bewertung aber noch eine Abschätzung der zu erwartenden Latenzzeit berechnen.

Die Auswahl einer Kombination von Operationen wird im Algorithmus durch eine Bewertungsfunktion gesteuert, die damit wesentlich für die Qualität der Ergebnisse verantwortlich ist. Die Bewertungsfunktion wird im folgenden genauer beschrieben. Das Verfahren wurde in [29] implementiert und in [28] erstmals vorgestellt.

Bewertung

Die Bewertung bestimmt für einen Datenflußgraphen, wie die Verzögerungszeit bei der aktuellen Anordnung der Operationen voraussichtlich ausfällt. Falls eine Fließbandverarbeitung angestrebt wird, soll auch die voraussichtlich mögliche Latenzzeit abgeschätzt werden. Eine Bewertung, die die festzustellenden Werte genau berechnet, führt zu optimalem Scheduling. Da optimales Scheduling unter Berücksichtigung gegebener Komponenten NP-hart ist [26], ist keine effiziente Bewertung mit genauen Resultaten zu erwarten, und da die Bewertung aus dem Listen-Scheduling oft aufgerufen wird, ist die Effizienz eine wichtige Forderung. Man muß deshalb auf absolute Genauigkeit verzichten und eine Heuristik verwenden mit dem Ziel, die Gesamtverzögerungszeit und die Latenzzeit möglichst schnell und möglichst gut abzuschätzen.

Arbeitsweise

Die Bewertung bearbeitet den in der aktuellen Anordnung noch nicht eingeplanten Teil des Datenflußgraphen, d.h. die Operationen, die in einem späteren als dem aktuellen Taktzyklus noch eingeplant werden müssen. Sie arbeitet in zwei Schritten. Zunächst wird ein Scheduling ohne Komponentenbeschränkung durchgeführt. Dieses Scheduling ist effizient möglich und führt zur minimalen Verzögerungszeit. Im zweiten Schritt wird versucht, die zusätzliche Verzögerung durch die Beschränkung der Komponenten abzuschätzen. Diese Verzögerung entsteht aus der Konkurrenz der Operationen um Komponenten. Wenn es weniger Komponenten gibt als Operationen, können nicht alle Operationen so ausgeführt werden, wie es dem Scheduling ohne Komponentenbeschränkung entspricht; unter Umständen muß die

Algorithmus:

```
Prozedur CASCH (V)
V: Operationen im Datenflussgraphen
BEGIN
 akt_takt := 0;
 WHILE V <> {} DO
  BEGIN
   Vb := Menge der Operationen aus V
         ohne Vorgaenger in V;
   generiere und bewerte alle
   Kombinationen (Vb);
   waehle die Kombination Vd aus Vb
   mit der besten Bewertung;
   V := V \ Vd;
   Schedule(akt_takt) := Vd;
   akt_takt := akt_takt + 1;
  END
END
Prozedur generiere und bewerte alle
Kombinationen (B) (* mit B = bereite
Operationen im aktuellen Taktzyklus *)
BEGIN
 WHILE B <> {} DO
  BEGIN
   b := elem(B);
   B := B - {b};
   K := Menge der moeglichen Komponenten-
        typen fuer Operation b;
   fuer alle k aus K
    BEGIN
     falls eine Komponente
     des Typs k frei ist
      BEGIN
       belege eine Komponente des Typs k;
       plane Operation b im aktuellen
       Taktzyklus;
       generiere und bewerte alle
       Kombinationen (B);
       fuehre Bewertung fuer die aktuellen
       Anordnung durch;
       nimm Op. b aus aktuellem Taktzyklus;
       gib eine Komponente des Typs k frei;
      END
    END
  END
END
```

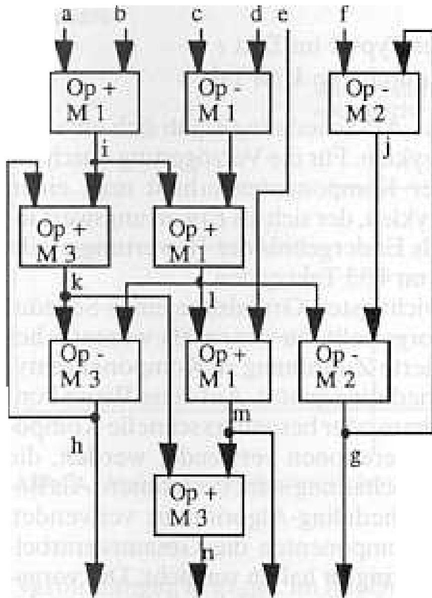


Abb. 5. Datenflußgraph einer Schaltung

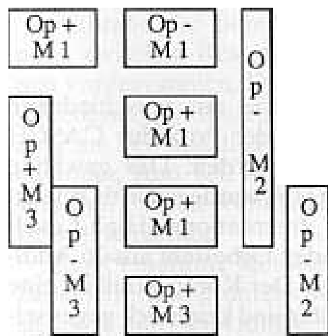


Abb. 6. Zeitbereiche der Operationen

Gesamtverzögerungszeit erhöht werden, um alle Operationen mit den gegebenen Komponenten ausführen zu können. Das Gesamtergebnis der Bewertung ist dann die Summe aus der minimalen Verzögerungszeit und der Verzögerung, die aus der Komponentenbeschränkung entsteht.

Als Beispiel sei der Datenflußgraph in Abb. 5 betrachtet, der als zu bewertender Teil eines größeren Graphen anzusehen ist. Es soll nun die voraussichtliche Verzögerungszeit abgeschätzt werden. Als Komponenten stehen ein Addierer und eine ALU zur Verfügung. Jede Komponente benötige für eine Operation genau einen Taktzyklus. Der Addierer kann eine Addition ausführen, während die ALU addieren und subtrahieren kann, d.h. für die Subtraktion muß die ALU verwendet werden, während für die Addition eine Wahlmöglichkeit zwischen Addierer und ALU besteht. Die Zeilen in Abb. 5 entsprechen den Taktzyklen als Ergebnis eines ASAP-Scheduling. Es werden mindestens 4 Taktzyklen benötigt, um alle Operationen auszuführen. Wie beim Kräfteverfahren werden nun die Ausführungsintervalle und die entsprechenden Wahrscheinlichkeiten und Verteilungen berechnet. Das Ergebnis zeigt Abb. 6.

In der Darstellung der Verteilung sind die Informationen über die Datenabhängigkeiten nicht mehr direkt vorhanden. Sie sind bereits in das ASAP- und ALAP-Scheduling eingegangen und implizit in den Ausführungsintervallen der Operationen enthalten. In der weiteren Verarbeitung werden die Datenabhängigkeiten nicht mehr benötigt.

Bewertungstabelle

An die Darstellung in Abb. 6 soll sich eine Berechnung der Verzögerung durch Komponentenbeschränkung anschließen. Für diese Berechnungen eignet sich eine numerische Darstellung der Verteilung der Operationen besser.

Tabelle 2 enthält für jede Operationsart, jeden Taktzyklus und jede Alternativ-Marke einen Wahrscheinlichkeitswert für die Operationen entsprechend Abb. 5 bzw. Abb. 6. Wenn die mit M1 markierten Operationen aus Abb. 5 die Operationen in einem Hauptzweig darstellen und die mit M2 bzw. M3 markierten Operationen sich gegenseitig ausschließen, erhält man als Ergebnis die Tabelle 3, die es erlaubt, die Information über sich gegenseitig ausschließende Operationen bei der Ablaufplanung zu verwenden.

Die hier vorgestellte Ablaufplanungsheuristik unterstützt auch die Vorab-Zuordnung von Operationen auf Typen von Verarbeitungseinheiten. Es ist sinnvoll, die Vorab-Zuordnung in das Scheduling zu integrieren, da die Ausführung derselben Operation durch unterschiedliche Typen von Verarbeitungseinheiten auch eine unterschiedliche Dauer bei der Ausführung dieser Operation bedeutet, was wiederum die Ablaufplanung beeinflußt. Beispielsweise kann eine Multiplikation durch sequentielles Addieren und Schieben oder durch einen parallelen Multiplizierer implementiert werden. Für nähere Informationen über diese Vorab-Zuordnung sei auf [28] verwiesen. Als sinnvolles Ergebnis dieser Vorab-Zuweisung ergibt sich in diesem Fall die Tabelle 4.

Im nächsten Schritt kann dann die Maximum-Bildung für die einzelnen alternativen Zweige durchgeführt werden, deren Ergebnis in Tabelle 5 angegeben ist.

Tabelle 2. Bewertungstabelle

	M1		M2		M3	
T	Op +	Op –	Op +	Op –	Op +	Op –

Tabelle 3. Aufsummierung der Zweige

	M1–2		M1–3	
T		Op –		Op –
		1,33		
2		0,33	1,5	
3		0,83	1,5	
4		0,5	1	

Tabelle 4. Umrechnung auf Komponenten

T	M1–2		M1–3	
	Add	ALU	Add	ALU
		1,33		
	0,67	0,67	0,7:	0,75
	0,92	0,92		
	0	0,5	0,7:	0,75

Tabelle 5. Maximumbildung der Zweige

T	M1-2-3	ALU
	0,75	
	0,75	

Tabelle 6. Ergebnisse der CASCH-Prozedur für ein elliptisches Filter

Ressourcen						Ergebnisse	
Add	Add*	Mul	Mul-P	ALU	ALU-P	CPU [s]	Takte
1	–	2				1,5	28
2	–	2					18
3	–	2				1,4	18
						1,5	28
						1,4	19
						1,3	18
						1,5	16
						1,4	16
0						2,0	23
1						3,1	19
2						3,6	18
1						2,5	20
2						2,8	18
						2,5	14
						2,4	14

Add: Addierer, Verzögerungszeit 1 Takt
Add*: Addierer, Verzögerungszeit 0,5 Takte
Mul: Multiplizierer, Verzögerungszeit 2 Takte
Mul-P: Multiplizierer mit interner Pipeline, wie Mul, Latenzzeit 1 Takt
ALU: ALU (Addition, Multiplikation), Verzögerungszeit für Add. 1 Takt, für Multipl. 2 Takte
ALU-P: ALU mit interner Pipeline, wie ALU, Latenzzeit 1 Takt

Die damit berechnete Anzahl von Verarbeitungseinheiten ist eine Abschätzung der Gesamtanzahl aller Verarbeitungseinheiten aller Typen, die erforderlich ist, um jeden möglichen alternativen Zweig zu bearbeiten.

Verzögerung durch Ressourcenbeschränkung

Aus Tabelle 5 muß jetzt die voraussichtliche Verzögerung durch eine beschränkte Komponentenanzahl bestimmt werden. Eine Verzögerung ergibt sich durch Komponenten mit einer Auslastung größer als 1. Die durch eine Komponente in einem Taktzyklus verursachte Verzögerung ist der Anteil der Auslastung, der über die Grenze von 1 hinausreicht. Sie ergibt sich also aus der Formel (1).

$$v_{k,t} = \max(a_{k,t} - 1{,}0) \quad (1)$$

$a_{k,t}$: Auslastung des Typs k im Takt t
$v_{k,t}$: Verzögerung durch Typ k im Takt t

Beim ASAP- und ALAP-Scheduling ergab sich eine Gesamtzeit von 4 Taktzyklen. Für die Verzögerung durch die Beschränkungen der Komponenten erhielt man einen Wert von 0,33 Taktzyklen, der sich als Erwartungswert interpretieren läßt. Als Endergebnis der Bewertung ergibt sich dann ein Wert von 4,33 Taktzyklen.

Es wurden die wichtigsten Grundzüge eines Scheduling-Algorithmus vorgestellt, zu denen als wesentliches Merkmal die integrierte Zuordnung zu Komponententypen während des Scheduling gehört. Auf diese Weise können begrenzt verfügbare oder besonders schnelle Komponenten gezielt für Operationen verwendet werden, die das Zeitverhalten der Schaltung stark bestimmen. Als Basis wird ein Listen-Scheduling-Algorithmus verwendet, der bei gegebenen Komponenten die Gesamtverarbeitungszeit möglichst gering zu halten versucht. Die vorgegebenen Komponenten bestimmen dabei die Struktur der resultierenden Schaltung.

Ergebnisse

Tabelle 6 bringt einige Ergebnisse mit verschiedenen Komponentenvorgaben, die mit der Prozedur CASCH (CAddy SCHeduling) erreicht wurden. Das gewählte Beispiel, ein elliptisches Filter [15], wurde auch als Benchmark für die Tagungsreihe „International High-Level-Synthesis Workshop" verwendet. Es besteht aus 26 Additionen und 8 Multiplikationen. Der Kontrollfluß ist eine einzige Endlosschleife. Im Filter sind keine sich gegenseitig ausschließenden Operationen zu berücksichtigen. Die Ergebnisse der ersten Hälfte der Tabelle benutzen die integrierte Komponententypenzuordnung nicht und sind mit den Scheduling-Algorithmen in [60, 57, 58] vergleichbar.

Die Ergebnisse in der zweiten Hälfte der Tabelle benutzen eine ALU. Durch Mitverwendung des Multiplizierers für Additionen können so Addierer bei fast gleicher Leistungsfähigkeit der Schaltung eingespart werden. Vergleichbare Ergebnisse von anderen Systemen liegen nicht vor. Alle CPU-Zeiten beziehen sich auf einen Arbeitsplatzrechner HP/Apollo DN 4500. Die Rechenzeiten sind so kurz, daß mehrere Alternativen durchgerechnet werden können. Die Untersuchung des Lösungsraums läßt sich stark ausweiten, wenn die Bewertungsfunktion für eine erste Abschätzung auch außerhalb des eigentlichen Scheduling-Algorithmus aufrufbar ist. Dies kann später als Basis für eine Automatisierung der Komponentenauswahl genutzt werden.

3.2. Bereitstellung von Ressourcen

Dieser Schritt legt fest, welche wesentlichen Hardware*ressourcen* zur Lösung des Syntheseproblems zur Verfügung gestellt werden sollen. Als Ressourcen kommen u. a. arithmetische/logische Bausteine, Speicherzellen und

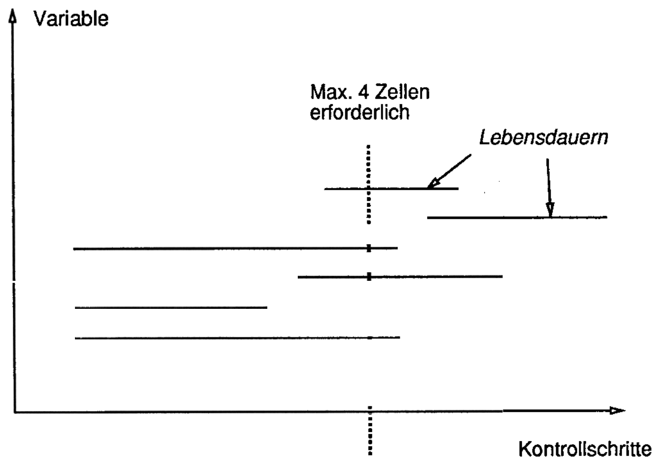


Abb. 7. Maximale Zahl gleichzeitig benötigter Ressourcen

Verbindungen in Frage. Im folgenden bezieht sich die Erklärung allgemein auf „Ressourcen". Im konkreten Fall ist dafür eine der erwähnten Ressourcen-Klassen einzusetzen.

Während der Bereitstellung wird noch keine Zuordnung zwischen Ressourcen und ausgeführten Operationen vorgenommen. Die Methode der Kopplung der verschiedenen Synthesephasen hängt vom Synthesesystem ab. Zum Teil sind die Phasen weitgehend unabhängig voneinander, zum Teil ist eine gewisse Vorausschau auf die späteren Phasen vorhanden und zum Teil werden Phasen zusammengefaßt.

Es gibt mehrere Methoden, Ressourcen zur Verfügung zu stellen. Diese unterscheiden sich häufig sowohl hinsichtlich der Reihenfolge, in der Ressourcen der einzelnen Klassen bereitgestellt werden, als auch in dem für jede Klasse benutzten Verfahren. Ebenfalls abhängig von den unterschiedlichen Ansätzen ist die Reihenfolge der Schritte zur Ablaufplanung und zur Ressourcen-Bereitstellung, die in der Regel miteinander verzahnt werden.

3.2. Vorgabe durch den Benutzer

Bei dieser Form erklärt der Benutzer, welche Ressourcen einer bestimmten Klasse in der späteren Struktur vorhanden sein sollen. Dies ist die einfachste Form, Ressourcen auszuwählen.

Beispiele sind:

- Das LALSD-II-Synthesesystem [33]: In diesem System werden arithmetische/logische Einheiten bereits durch den Benutzer ausgewählt.
- Das MIMOLA-Software-System (MSS): Hier werden die Speicherzellen für Variablen in der Regel vom Benutzer vergeben.
- CALLAS/CADDY: Die Verarbeitungseinheiten und Busse werden vorgegeben, die Speicherelemente (unter Berücksichtigung der Verbindungskosten) minimiert.

Die Möglichkeit zur Vorgabe von Ressourcen bereits in der Spezifikation sollte vorhanden sein. Dafür werden Spezifikationssprachen benötigt, die die Beschreibung einer Ressource-Bibliothek ermöglichen. Zusätzlich sollte jedoch auch eine selbständige Auswahl der Ressourcen durch das Synthesesystem möglich sein.

3.2.2. „Direkte Compilation"

Eine noch recht einfache Art der Bereitstellung ist, für jedes Vorkommen eines Operators eine eigene Ressource zu reservieren und keinerlei Mehrfachnutzung vorzusehen. Wie in gewöhnlichen Compilern wird für jede Operation „Code" erzeugt. Dieser Ansatz ist nur für kleine Verhaltensbeschreibungen geeignet, denn für eine Beschreibung mit 1000 Additionen würden ja 1000 Addierer erzeugt.

Vielfach wird einer Variablen der Verhaltensbeschreibung genau ein Register exklusiv zugeordnet. Auch dies ist direkte Compilation, bei der Bereitstellung und Zuordnung generell in derselben Synthesephase durchgeführt werden.

3.2.3. Mehrfachnutzung von Ressourcen mit gleichen Fähigkeiten

Bei dieser Form wird davon ausgegangen, daß jede Operation von Ressourcen aus genau einer Klasse ausgeführt werden kann. Durch Mehrfachnutzung von Ressourcen sollen ihre Zahl und damit auch ihre Kosten minimiert werden.

Ein Beispiel hierfür ist die Bereitstellung von Zellen für die Variablen der Verhaltensbeschreibung. Derartige Variable werden häufig nur bei einem Teil der Kontrollschritte benötigt. Das Intervall von der ersten definierenden bis zur letzten lesenden Referenz heißt *Lebensdauer* der Variablen. Die Lebensdauer in verzweigungsfreien Sequenzen von Kontrollschritten kann in Diagrammen dargestellt werden (Abb. 7). Variablen, deren Lebensdauern sich nicht überlappen, können derselben Zelle zugeordnet werden. Die insgesamt benötigte Zahl von Zellen ergibt sich danach aus der Maximalzahl gleichzeitig „lebendiger" Variablen.

3.2.4. Mehrfachnutzung von Ressourcen mit unterschiedlichen Fähigkeiten

Wenn die Ressourcen unterschiedliche Fähigkeiten haben, reicht das Zählen der maximal benötigten Ressourcen nicht aus. Vielmehr werden jetzt die Ressourcen auch unterschiedlich komplex und daher unterschiedlich teuer sein. Statt der Zahl der Ressourcen sind nunmehr ihre Kosten zu minimieren. Das können reale Kosten sein (z. B. die benötigte Chipfläche) oder auch fiktive Kosten, die lediglich ausdrücken, daß bestimmte Bausteine aufwendiger sind als andere.

Es müssen notwendige Bedingungen für die jeweils benötigte Mindestzahl von Ressourcen eingehalten werden. Wie bestimmt man nun diese Zahl?

Wir wollen uns die Verhältnisse zunächst anhand eines Beispiels klarmachen und benutzen dafür die Bereitstellung von ALUs. Da jetzt Kosten minimiert werden sollen, müssen für alle möglichen ALUs die Kosten bekannt sein (das bedeutet nicht, daß sie bis in alle Einzelheiten entworfen sein müssen). Für unsere Zwecke sei wie am Anfang des Artikels eine Bibliothek mit Bausteintypen t_1, t_2 und t_3 vorgegeben; t_1 sei ein Addierer mit Kosten c_1, t_2 ein Subtrahierer mit Kosten c_2 und t_3 ein alternativ zum Addieren oder Subtrahieren fähiger Baustein mit Kosten c_3. Gegeben sei die Anweisung des Subtraktionsbefehls aus dem einführenden Beispiel:

„reg[ir.RegNr]: = reg[ir.RegNr]-main[reg[2] + ir.Dist]“

Wir wollen annehmen, daß diese Anweisung während der Ablaufplanung nicht in mehrere Kontrollschritte aufgeteilt wird. Da innerhalb eines Kontrollschritts kein Zeit-Multiplexing von Ressourcen stattfinden soll, werden für die beiden arithmetischen Operationen zwei ALUs benötigt. Wenn x_i die zu wählende Zahl von ALUs vom Typ i ist müssen offensichtlich folgende Bedingungen erfüllt sein:

$x_1 + 0 + x_3 \geq 1$ (wegen der Addition)
$0 + x_2 + x_3 \geq 1$ (wegen der Subtraktion)
$x_1 + x_2 + x_3 \geq 2$ (wegen der Gesamtzahl der Operationen)

Berücksichtigt man noch den Additionsbefehl des einführenden Beispiels, so erhält man folgenden Satz von Ungleichungen:

$x_1 + 0 + x_3 \geq 2$ (wegen der zwei Additionen im Additionsbefehl)
$0 + x_2 + x_3 \geq 1$ (wegen der Subtraktion)
$x_1 + x_2 + x_3 \geq 2$ (wegen der Gesamtzahl der Operationen)

Für die angenommenen Kosten wäre $x_1 = 1$, $x_2 = 0$ und $x_3 = 1$, d.h. 1 Addierer und 1 Multifunktionsbaustein optimal. Der Name der betreffenden Exemplare sei im folgenden m1 bzw. m3.

Allgemein gilt: Für jede Kombination der vorkommenden Operationen muß die Gesamtzahl der Ressourcen, die mindestens eine der Operationen ausführen können, mindestens gleich der Zahl der Operationen sein [47, 48].

Mathematisch läßt sich dies so formulieren:

Sei
T die Menge der Ressourcetypen,
c_t die (ggf. fiktiven) Kosten des Typs $t, t \in T$,
o_t die Menge der vom Typ $t \in T$ bereitgestellten Operationen,
$f_{i,j}$ die Häufigkeit des Vorkommens der Operation j in Schritt i (d.h. z.B. die Zahl der 16-Bit-Additionen in diesem Schritt),
F_i die Menge der in Schritt i benutzten Operationen, d.h. $F_i = \{j \mid f_{i,j} > 0\}$,
F_i^* die Menge aller Teilmengen von F_i, d.h. $F_i^* = \wp(F_i)$,

dann muß gelten:

$$\forall i, \forall g \in F_i^*: \sum_{\substack{t \in T \\ g \cap o_t \neq \emptyset}} x_t \geq \sum_{j \in g} f_{i,j} \quad (2)$$

Für alle Kontrollschritte i und alle Elemente g der Potenzmenge der in i benutzten Operationen ist also die Summe der Exemplare jener Ressourcetypen, die mindestens eine benötigte Operation ausführen können, mindestens gleich der Gesamtzahl der ausgeführten Operationen.

Diese Aussagen sind zunächst einmal für einen einzelnen Kontrollschritt i hinreichend. Mit einigen einfachen Rechnungen kommt man zu Ungleichungen, die für alle Kontrollschritte ausreichen:

$$\forall g \in F^*: \sum_{t \in T} a_{g,t} * x_t \geq b_g \quad (3)$$

mit

$$\forall g \in F^*: b_g = \max_i \left(\sum_{j \in g} f_{i,j}\right),$$
$$F^* = \cup_i F_i^*,$$
$$a_{g,t} = \begin{cases} 1, & \text{falls } g \cap o_t \neq \emptyset \\ 0, & \text{sonst} \end{cases}$$

Formel (3) zeigt, daß die Randbedingungen in Form von in x_t linearen Ungleichungen mit binären Koeffizienten $a_{g,t}$ darstellbar sind. Damit ist die optimale Bereitstellung auf eine ganzzahlige lineare Optimierung zurückgeführt. Ein Verfahren zur Minimierung von Gatterschaltungen auf der Basis der ganzzahligen Programmierung gab Kodres 1972 an [39]. Die hier vorgestellte Methode wurde für TODOS[1] entwickelt [49]. Papachristou [59] hat dieses Verfahren um eine Kopplung mit der Ablaufplanung ergänzt.

Im HAL-System [56] wird regelbasiert jeweils eine Verarbeitungseinheit aus einer Bibliothek ausgewählt und Operationen des Datenflußgraphen zugeordnet. Anschließend wird der Datenflußgraph um diese Operationen reduziert und die nächste Verarbeitungseinheit nach demselben Verfahren ausgewählt. In diesem System werden also die Baustein-Auswahl und die Baustein-Zuordnung miteinander (nach heuristischen Methoden) kombiniert.

Im MAHA-System [61] erfolgen Auswahl und Zuordnung nach einem heuristischen Verfahren ebenfalls gleichzeitig, jedoch wird dabei zunächst nur der kritische Pfad betrachtet. Anschließend wird in einem relativ aufwendigen Verfahren (Komplexität $O(n^4)$) versucht, Operationen abseits des kritischen Pfades auf die im ersten Schritt ausgewählte Hardware abzubilden.

3.3. Zuordnung

Von der Bereitstellung wollen wir die Zuordnung („assignment“) unterscheiden. Hier gibt es zwei wichtige Fälle:

- Die Zuordnung von Variablen zu physikalischen Speichern (Registern oder RAM-Zellen): Diese Zuordnung wird auch als „Variablen/Speicher-Bindung“ bezeichnet.

[1] TODOS = *to*p *do*wn *s*ynthesis system, nachträglich vergebener Name für eines der drei für das MSS entwickelten Syntheseverfahren [47, 48].

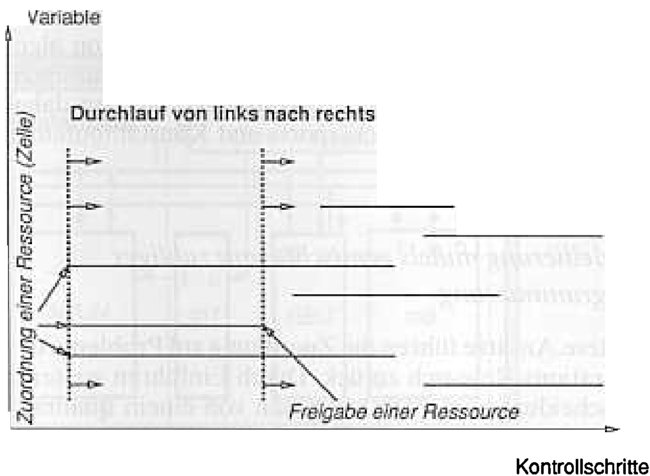


Abb. 8. Graphische Darstellung des „left-edge"-Algorithmus

```
FOR i:=1 TO nmax DO
 BEGIN  (*Schleife ueber Kontrollschritte*)
  gebe alle Zellen, die durch Variablen
  belegt werden, die in dem Kontrollschritt
  i zuletzt benutzt werden, frei;
  Ordne allen Variablen, die im Kontroll-
  schritt i zuerst benutzt werden, eine
  Speicherzelle zu;
 END;
```

Abb. 9. Der „left-edge"-Algorithmus für die Variablenzuordnung

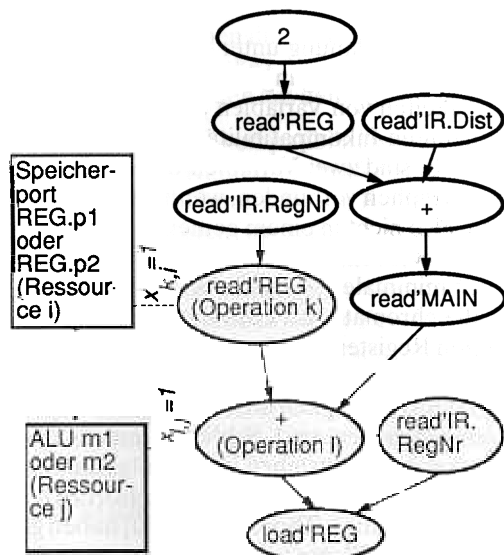


Abb. 10. Datenflußbaum

• Die Zuordnung von Operationen (arithmetische/logische Operationen, Schreib-Lese-Operationen, Konstantenbildung) zu ALUs, Speicherports und festverdrahteten Konstanten. Diese Zuordnung wird im Englischen als *„operation/operator binding"* bezeichnet.

Die Größen, zu denen zugeordnet wird, werden im folgenden als Ressourcen bezeichnet. Zuordnungsalgorithmen gehen davon aus, daß bereitgestellte Ressourcen bekannt sind.

3.3.1. Zuordnung ohne Rücksicht auf Verbindungsstrukturen

Wenn es genügt, zu jeder Operation eine passende Ressource zu finden, so läßt sich das Mehrfachnutzen auf einfache Weise durch einen Algorithmus nach Art des sog. „left-edge"-Algorithmus bestimmen. Der Name rührt daher, daß man von links mit einer Schnittlinie (vgl. Abb. 8) die Lebensbereiche überstreicht. Kommt man zu einem linken Ende eines Intervalls, so ordnet man dem Intervall eine Zelle (eine Ressource) zu. Kommt man zu einem rechten Ende, so gibt man diese Ressource wieder frei. Abbildung 9 zeigt die Formulierung für die Zuordnung von Speicherzellen zu Variablen.

Die Verwendung dieser Methode für die Speicherzuordnung wurde von mehreren Autoren – zum Teil unabhängig voneinander – vorgeschlagen, z. B. von Marwedel [47], Parker et al. [44] und Pfahler [62]. Auch im Compilerbau werden diese Methoden angewendet, z. B. bei Kim und Tan [35] sowie bei Chaitin [14]. Die Methode nimmt keinerlei Rücksicht auf eine eventuell unterschiedliche Funktionalität der Ressourcen und auf die Rückwirkung der Zuordnung auf Verbindungsstrukturen. Dennoch kann das Verfahren dazu benutzt werden, z. B. den Variablen gleichartige Zellen eines Speichers zuzuordnen.

3.3.2. Zuordnung mit Berücksichtigung von Verbindungsstrukturen

Für die Zuordnung von ALUs benötigt man Zuordnungsalgorithmen, die auf die unterschiedliche Funktionalität Rücksicht nehmen und die Verbindungskosten minimieren.

Die Verhältnisse seien anhand eines Beispiels verdeutlicht. Das Scheduling erzeugt eine Menge von Kontrollschritten, von denen jeder die Berechnung bestimmter Ausdrücke auslöst. Diese Berechnungen lassen sich anhand von Datenflußbäumen anschaulich darstellen, z. B. ergibt sich für den Subtraktionsbefehl aus dem einführenden Beispiel die Abb. 10. Jeder der durch Ovale dargestellten Operationen kann man Ressourcen zuordnen. Wenn wir dem Lesen der „Variablen" **REG [IR. RegNr]** Port $p2$ eines Speichers *REG* (geschrieben *REG.p*2) und der Addition die ALU $m1$ zuordnen, so macht dies eine physikalische Verbindung von *REG.p*2 nach $m1$ erforderlich.

Verallgemeinert gilt: *Die Zuordnung von Hardware-Ressourcen impliziert die Verbindungen der Hardware-Struktur.*

Die Zuordnung muß daher so vorgenommen werden, daß die Kosten für Verbindungen möglichst gering werden. Es hat verschiedene Ansätze gegeben, die Zuordnung von Hardware-Ressourcen auf Standard-Optimierungsprobleme abzubilden.

Modellierung als quadratisches Zuordnungsproblem

Die Modellierung für einen einzelnen Kontrollschritt:

Sei
R die Menge der Ressourcen,
$c_{i,j}$ die Kosten der Verbindung von Ressource i nach Ressource j,
$b_{k,l}$ = 1, falls Operation k Argument von Operation l ist,
= 0 sonst,
$x_{k,i}$ = 1, falls der Operation k Ressource i zugeordnet wird,
= 0 sonst.

Die gesamten Verbindungskosten ergeben sich dann als:

$$K = \sum_{i,j,k,l} b_{k,l} * c_{i,j} * x_{k,i} * x_{l,j} \quad (4)$$

$a_{k,i}$ = 1, falls die Operation k von Ressource i ausgeführt werden kann,
= 0 sonst

unter den Randbedingungen

$$\forall i : \sum_{k} x_{k,i} \leq 1 \quad (5)$$

$$\forall k : \sum a_{k,i} * x_{k,i} = 1. \quad (6)$$

Dabei sind $b_{k,l}$ und $c_{i,j}$ vorgegeben und die Entscheidungsvariablen x gesucht. Ist nämlich Operation k Argument von l und wird der Operation k die Ressource i sowie der Operation l die Ressource j zugeordnet, so entstehen Kosten für die Verbindung von Ressource i nach Ressource j. Diese Kosten sind unter den angegebenen Randbedingungen zu minimieren. Die Randbedingungen legen fest, daß jede Ressource höchstens eine Operation ausführen kann und daß jeder Operation genau eine zur Ausführung dieser Operation geeignete Ressource zugeordnet werden muß. Dieses Optimierungsproblem ist als *quadratisches Zuordnungsproblem* (abgekürzt: QAP) bekannt. Das Adjektiv „quadratisch" bezieht sich auf die Variablen x, die quadratisch vorkommen. Das entsprechende QAP-Entscheidungsproblem ist NP-vollständig [26]; die Rechenzeiten sind daher bei diesem Ansatz sehr hoch, und nicht immer kann eine Lösung in der verfügbaren Zeit bestimmt werden. Außerdem ist die angegebene Modellierung für die Behandlung mehrerer Kontrollschritte noch zu erweitern.

Der im Beispiel benutzte Datenflußbaum enthält mehrere Operationen, die innerhalb eines Kontrollschrittes nacheinander – ohne Zwischenspeicherung von Teilausdrücken – ausgeführt werden. Für arithmetische Operationen (wie hier für die beiden Additionen) bezeichnet man dies als *chaining*. Die QAP-Modellierung des Zuordnungsproblems erlaubt eine konsequente Behandlung von Fällen, in denen auf mehreren Ebenen des Datenflußbaums Wahlfreiheit für die Zuordnung von Ressourcen besteht, also insbesondere des *chainings*.

In TODOS wurde auf der Basis des QAP-Modells ein „Branch-and-Bound"-Verfahren realisiert, das für jeden Kontrollschritt die Zahl der Verbindungen zu minimieren versucht. Dabei werden zusätzlich gemeinsame Teilausdrücke berücksichtigt und außerdem wird die Ansteuerung der ALUs möglichst so ausgewählt, daß von algebraischen Gesetzen zur Optimierung der Verbindungen Gebrauch gemacht wird. Als Ressourcen werden dabei gleichzeitig ALUs, Speicherports und Konstantenfelder betrachtet.

Modellierung mittels gemischt-/ganzzahliger Programmierung

Weitere Ansätze führen die Zuordnung auf Probleme des Operations-Research zurück. Durch Einführen weiterer Entscheidungsvariablen kann man von einem quadratischen Zuordnungsproblem auf ein lineares Zuordnungsproblem bzw. auf ein ganzzahliges Programmierungsproblem übergehen. Dadurch entstehen aber so viele Variablen und Ungleichungen, daß keine Lösung für praktische Anwendungen erreicht wird. Dies zeigen insbesondere die Ergebnisse von Hafer et al. [30]. Ein Teil der Komplexitätsprobleme dieses Ansatzes ist allerdings darauf zurückzuführen, daß die Modellierung auch Scheduling und Zuteilung einschließt.

Wegen der Zahl der Variablen bleibt das Problem aber weiterhin zu komplex, um es optimal (im Sinne obiger Kostenfunktion) zu lösen. Daher sind viele Heuristiken im Gebrauch, die eine ausreichend gute Lösung (über alle Kontrollschritte gesehen) liefern sollen.

Modellierung mittels Graphenfärbung

Die sich aus der Ablaufplanung unter Berücksichtigung von datenabhängigen Verzweigungen und Schleifen ergebenden Abhängigkeiten von Variablen und Operationen können in sogenannten Inkompatibilitätsgraphen ausgedrückt werden. Z.B. sind zwei Variablen dann mit einer Kante in diesem Graphen verbunden, wenn sie gleichzeitig lebendig sind, also nicht in einem gemeinsamen Register abgespeichert werden können. Das Färben dieses Graphen mit der minimalen Anzahl von Farben würde demnach über die chromatische Zahl die minimale Anzahl der benötigten Register liefern.

Leider ist der oben erwähnte „left-edge"-Algorithmus [44] nur auf sog. Intervallgraphen anwendbar. Aus datenabhängigen Verzweigungen und Schleifen einer algorithmischen Beschreibung entstehen jedoch allgemeinere Graphen. Das Färben allgemeiner Graphen gehört zu den NP-vollständigen Problemen. Thomas et al. [70] haben gezeigt, daß für eine große Klasse von algorithmischen Beschreibungen dennoch in polynomialer Zeit eine Färbung mit minimaler Zahl von Farben berechnet werden kann.

Die Modellierung des Zuordnungsproblems mit Inkompatibilitätsgraphen berücksichtigt noch keine Verbindungskosten. In CADDY werden daher zur Lösung der Zuordnungsprobleme Heuristiken benutzt, die zusätzlich zum Färbungsmodell auch eine Minimierung von Verbindungskosten gestatten. Zu diesem Zweck wurden in CADDY neben diesen Inkompatibilitätsgraphen noch „Präferenzgraphen" eingeführt. Beispiele für Präferenzen, also für das Färben mit gleicher Farbe, was z.B. dem

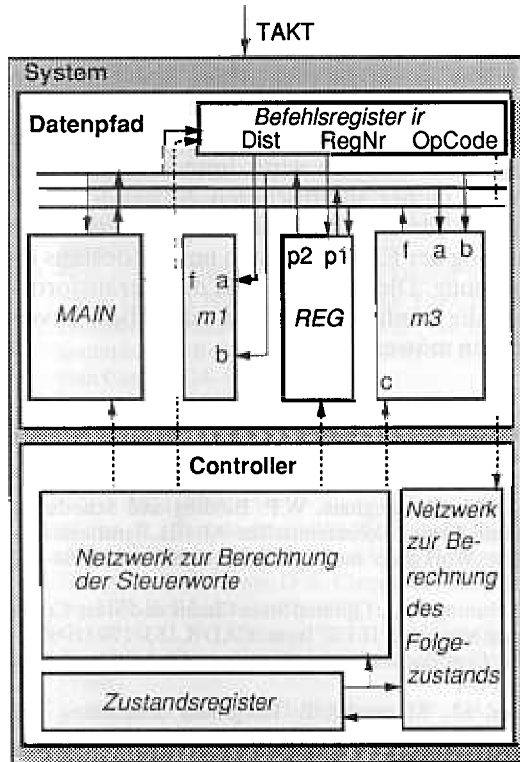


Abb. 11. Aufteilung des Entwurfs in Datenpfad und Controller

Abspeichern der mit einer Präferenzkante verbundenen Werte im selben Register entspricht, sind:

- Zwei Werte haben die gleiche Verarbeitungseinheit als Quelle (bzw. als Senke).
- Zwei Werte werden aus dem globalen Speicher gelesen.

Das Färben des Inkompatibilitätsgraphen unter Berücksichtigung von Kanten im Präferenzgraphen ermöglicht die gezielte Erhöhung z. B. von Registern, um die Verbindungsgesamtkosten zu senken. Beschreibungen dieses Verfahrens finden sich in [40] bzw. in [41].

Ein in [62] vorgeschlagenes Verfahren beruht darauf, die Speicherelemente in jedem Zeitschritt getrennt zu untersuchen. Der so erhaltene Graph disjunkter Cliquen läßt sich zwar in polynomialer Zeit färben, das dadurch evtl. benötigte Umspeichern von Variablen zwischen Registern kann aber zu einer höheren Verbindungszahl führen.

Berücksichtigung weiterer Entwurfsziele

ALUs können meistens auch Eingabewerte unverändert an den Ausgang weiterleiten. Zusammen mit Multiplexern und Bus-Treibern am Ein- und Ausgang der ALUs entstehen mehrstufige Verbindungsnetzwerke. Davon wurde in den frühen Synthesesystemen kein Gebrauch gemacht. Ein Modell für die Zuordnung bei mehrstufigen Verbindungsnetzwerken beschrieben kürzlich Ly, Elwood und Girczyc [46].

Die oben erwähnten Methoden können lediglich die *Zahl* der Verbindungswege möglichst gering halten. Das eigentliche Entwurfsziel ist aber die Minimierung der Chipfläche. McFarland zeigte 1987 [52], daß eine Kostenschätzung auf der Basis der Zahl der Verbindungen nur eine unzureichende Approximation der benötigten Fläche liefert. Es werden daher Syntheseverfahren entwickelt, die die benötigte Fläche genauer berücksichtigen. Z. B. wird bei Fasolt [37] zur Unterstützung der Entwurfsentscheidungen ein Flächenabschätzer aufgerufen, der mittels Plazierung der Bausteine in der Ebene eine Fläche abschätzt. Dabei wird stets eine völlig neue Plazierung berechnet. Die Flächenminimierung bei beliebigen Layout-Techniken ist insgesamt noch unbefriedigend. Für eingeschränkte Layout-Techniken, z. B. für die Beschränkung auf die Kommunikation mittels gekoppelter Busse, können deutliche Flächengewinne erzielt werden [24].

3.4. Steuerwerkssynthese

Bislang haben wir uns lediglich mit der Erzeugung von ALUs, Speichern und Verbindungen beschäftigt. Diese Bausteine bilden zusammen den sog. *Datenpfad.* Damit die Bausteine des Datenpfads in jedem Kontrollschritt die beabsichtigte Funktion ausführen, müssen sie entsprechend kontrolliert werden. Beispielsweise muß der oben angeführte Baustein m3 mit einem Steuercode versorgt werden, der die Addition oder die Subtraktion auswählt. Zu diesem Zweck wird ein *Steuerwerk* (ein sog. *„Controller“,* vgl. Abb. 11) benötigt. Wesentliche Teile des Controllers sind ein Zustandsregister, ein Lese-Speicher, der jedem Zustand ein Steuerwort zuordnet, sowie eine Logik, die anhand des Zustandes und von Ausgangsleitungen des Datenpfades den jeweils nächsten Zustand bestimmt. Ein derartiger Controller kann also gut als Moore-Automat modelliert werden. Eine unmittelbare Rückwirkung der Ausgaben des Datenpfades auf das Steuerwort – entsprechend einem Mealy-Automaten – ist überflüssig, da sie durch eine Rückkopplung innerhalb des Datenpfades möglich ist.

Zur Synthese von Automaten wurden (z. B. zur Logik-Optimierung [7, 67]) zahlreiche Optimierungswerkzeuge entwickelt. Auf die Logikoptimierung soll hier aus Platzgründen nicht näher eingegangen werden. Allerdings sei erwähnt, daß es für die Synthese von Automaten, die indirekt durch die oben angegebenen imperativen Programme definiert werden, spezielle Verfahren gibt. Diese nutzen aus, daß wegen der Art der Abfolge von Kontrollschritten ein Zähler als Zustandsregister die Logik zur Bestimmung des Folgezustandes deutlich reduziert. Dies (und damit eine systematische Verbindung der Techniken der Mikroprogrammierung mit denen der Automatentheorie) wurde von Amann und Baitinger [2] vorgeschlagen. Auf ihrem Verfahren bauen mehrere Controllersynthese-Verfahren auf.

3.5. Beispiele unterschiedlicher Syntheseabläufe

Nicht immer ist der detaillierte Ablauf in einem Synthesesystem für den externen Beobachter durchschaubar, auch die Abschätzung des wirklichen Leistungsumfangs allein aufgrund von Publikationen ist schwierig. Darum soll nochmals auf den Autoren detailliert bekannte Systeme zurückgegriffen werden, um exemplarisch die unter-

Tabelle 7. Reihenfolge der Schritte in *CALLAS/CADDY*

Schritt	Methode
ggf. Schleifenexpansion (loop unrolling)	Compiler (expandierbare Schleifen werden dem Benutzer vorgeschlagen)
Datenflußanalyse und -extra tion	Datenflußan[illegible]lyse
Vorgabe von Anzahl und Typen der Verarbeitungs-einheiten sowie der Busse	durch den Benutz[illegible]
Scheduling mit Typzuordnun[illegible]	modifiziertes List Scheduling
Allokieren und Zuordnen de[illegible] Speicher	Graphfärben mit Präferenzen
Zuordnung von Verarbeitung[illegible]	[illegible]hfärben mit
Zuordnung von Verbindungen	Graphfärben mit Präferenze[illegible]
Steuerwerkssynthese	eigenes Synthesesystem (CASTOR) [64]

Tabelle 8. Reihenfolge der Schritte in *TODOS*

	Schritt	Methode
1	Ersatz höherer Sprachelemente	mit expl. Trans regeln
2	Vorgabe der Typen von Verarbeitungseinheiten	durch Benutzer
	Bereitstellung und Zuordnung von Speicherzellen für gewöhnliche Variable	automatisiert; m[illegible] Benutzer hinweisen
	Scheduling	FCFS mit Ressource-Beschränkungen
	Bereitstellung von ALUs	ganzzahlige Programmie[illegible]
	Zuordnung der Speicher fü[illegible] Hilfsvariable	„left-edge"-Algorithmus
	Zuordnung von ALUs und Speicherports	Branch-and-Bound
	Steuerwerkssynthese	Subsystem für horizontale Mikroprogramm-Steuerwerke

schiedlichen Syntheseschritte und ihre Reihenfolge darzustellen (Tabellen 7, 8).

4. Zusammenfassung und Ausblick

In diesem Beitrag sollte ein Überblick über die wichtigsten in der sog. „High-Level-Synthese" eingesetzten Methoden und Algorithmen gegeben werden. Aufgrund der mittlerweile sehr zahlreichen Forschungsergebnisse auf diesem Gebiet gibt es allgemein akzeptierte Strategien zur Lösung des „Syntheseproblems". So gibt es erfolgversprechende Prototypen[2] leistungsfähiger Synthesesysteme, die eine wirtschaftliche Anwendung in einem industriellen Umfeld erwarten lassen.

Diese Tendenz zu breiten praktischen Anwendungen wird noch dadurch verstärkt, daß der Bedarf an Synthesesystemen mit der zunehmenden Komplexität integrierter Schaltungen weiter wachsen wird. Immer komplexere Schaltungen mit immer spezifischeren Anwendungsfeldern und immer kürzeren Innovationszyklen erfordern eine Reduzierung der Entwurfskosten um mindestens eine Größenordnung. Dies zu leisten, ist die Herausforderung, der sich die künftigen Arbeiten zur „High-Level-Synthese" stellen müssen.

[2] Ein Beispiel für ein u[illegible]ssendes Entwurfssystem, [illegible] mehrere algorithmische Ansätze [illegible]ich vereinigt, ist die „sys[illegible]n architect's workbench" [73].

Literatur

1. Abramson, J.M., Birmingham, W.P.: Binding and Scheduling under External Timing Constraints: The ARIEL Synthesis System. 5th Int. Workshop on High-Level Synthesis, S. 94–101 (1991)
2. Amann, R., Baitinger, U.: Optimal State Chains and State Codes in Finite State Machines. IEEE Trans. CAD *8*, 153–170 (1989)
3. Baker, K.R.: Introduction to Sequencing and Scheduling. New York: Wiley 1974
4. Balakrishnan, M., Marwedel, P.: Integrated Scheduling and Binding: A Synthesis Approach for Design Space Exploration. Proc. 26th Design Automation Conf., S. 68–74 (1989)
5. Barbacci, M.R., Barnes G.E., Cattell, R.C., Siewiorek, D.P.: The ISPS Computer Description Language. Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh 1977
6. Brayton, R., Camposano, R., de Micheli, G., Otten, R., van Eijndhoven, J.: The Yorktown Silicon Compiler, in: Gajski, D. (Hrsg.): Silicon Compilation, S. 204–310. Reading: Addison-Wesley 1988
7. Brayton, R., Rudell, R., Sangiovanni-Vincentelli, A., Wang, A.R.: MIS: A Multiple-Level Logic Optimization System. IEEE Trans. CAD *6*, 1062–1081 (1987)
8. Beckmann, R., Marwedel, P., Schenk, W., Jöhnk, R.: The MIMOLA Language Reference Manual – Version 4.0. Forschungsbericht Nr. 401, Fachbereich Informatik der Universität Dortmund, 1992
9. Beckmann, R., Pusch, D., Schenk, W., Jöhnk, R.: The TREEMOLA Language Reference Manual – Version 4.0. Forschungsbericht Nr. 391, Fachbereich Informatik der Universität Dortmund, 1991
10. Broß, O., Marwedel, P., Schenk, W.: Incremental Synthesis and Support for Manual Binding in the MIMOLA Hardware Design System. 4th Int. Workshop on High-Level Synthesis, Kennebunkport 1989
11. Camposano, R., Kunzmann, A., Rosenstiel, W.: Automatic Data Path Synthesis from DSL Specifications. IEEE Int. Conf. on Computer Design (ICCD), 1984
12. Camposano, R., Rosenstiel, W.: Synthesizing Circuits from Behavioral Descriptions. IEEE Trans. CAD *8* (1989)
13. Camposano, R., Weber, R.: Semantik und Interne Form von DSL. Fakultät für Informatik, Universität Karlsruhe, Interner Bericht 22/84 (1984)
14. Chaitin, G.J.: Register Allocation via Coloring. IBM Research Report RC 8395 (1980)
15. Dewilde, P., Deprettere, E., Nouta, R.: Parallel and Pipelined VLSI Implementations of Signal Processing Algorithms, in: Kung, S.Y., Whitehouse, H.J., Kailath, T. (Hrsg.): VLSI and Modern Signal Processing. Englewood Cliffs: Prentice-Hall 1985
16. Devadas, S., Newton, R.: Algorithms for Hardware Allocation in Data Path Synthesis. IEEE Trans. CAD *8*, 768–781 (1989)
17. Denyer, P.B.: An Introduction to Bit-Serial Architectures for VLSI-Signal Processing, Arbeitsunterlagen des Advanced Course on VLSI Architectures, Bristol 1982

18. De Man, H., Rabaey, J., Six, P.: CATHEDRAL II: A Synthesis and Module Generation System for Multiprocessor Systems on a Chip, in: De Micheli, G., Sangiovanni-Vincentelli, A., Antognetti, P. (Hrsg.): Design Systems for VLSI Circuits – Logic Synthesis and Silicon Compilation. The Hague: Nijhoff 1987
19. Duzy, P., Krämer, H., Neher, M., Pilsl, M., Rosenstiel, W., Wecker, T.: CALLAS – Conversion of Algorithms to Library Adaptable Structures. VLSI *89* (1989)
20. DOSIS GmbH: DACAPO II, User Manual, Version 3.0, Dortmund 1987
21. Duley, J.R., Dietmeyer, D.L.: A Digital System Design Language (DDL). IEEE Trans. Comput. *C-17,* 850–861 (1968)
22. Dutt, N.D., Hadley, T., Gajski, D.D.: An Intermediate Representation for Behavioral Synthesis. Proc. 27th Design Automation Conf., S. 14–19 (1990)
23. Dutt, N.D.: GENIUS: A Generic Component Library for High Level Synthesis. Technical Report 88–22, U. C. Irvine 1988
24. Ewering, C.: Automatic High Level Synthesis of Partitioned Busses. IEEE Int. Conf. on Computer-Aided Design (ICCAD), S. 304–307 (1990)
25. Floren, R.: Ein inkrementeller Plazierer und Verdrahter zur Flächenabschätzung. Diplomarbeit Univ. Dortmund 1991
26. Garey, M.R., Johnson, D.S.: Computers and Intractability. Murray Hill: Bell Laboratories 1979
27. Gebotys, C.H., Elmasry, M.I.: VLSI Design Synthesis with Testability. Proc. 25th Design Automation Conference, S. 16–21 (1988)
28. Gutberlet, P., Krämer, H., Rosenstiel, W.: CASCH – Ein Scheduling-Algorithmus für High-Level-Synthese. GI/ITG/GME-Fachtagung „Rechnergestützter Entwurf und Architektur mikroelektronischer Systeme“, Dortmund 1990
29. Gutberlet, P.: Entwurf eines Scheduling Algorithmus für das CADDY-Synthesesystem. Diplomarbeit am Forschungszentrum Informatik an der Universität Karlsruhe 1989
30. Hafer, L., Parker, A.C.: A Formal Method for the Specification, Analysis and Design of Register-Transfer Level Digital Logic. IEEE Trans. Computer-Aided Design *2,* 4–18 (1983)
31. Hartenstein, R., Lemmert, K.: KARL-III Language Reference Manual. Bericht, Fachbereich Informatik, Univ. Kaiserslautern 1984
32. Hayes, J.P.: A Unified Switching Theory with Applications to VLSI Design. Proc. IEEE *70,* 1140–1151 (1982)
33. Huang, C.-L.: Computer-Aided Logic Synthesis Based on a New Multi-Level Hardware Design Language – LALSD II. Dissertation State University of New York at Binghamton 1981
34. IEEE Design Automation Standards Subcommittee: IEEE Standard VHDL Language Reference Manual [IEEE Std. 1076-(1987)]. New York: IEEE 1988
35. Kim, J., Tan, C.J.: Register Assignment for Optimizing Microcode Compilers, Part I, IBM Research Report RC 7639, Yorktown Heights 1979
36. Kim, K., Tront, J.G., Ha, S.D.: Automatic Insertion of BIST Hardware Using VHDL. Proc. 25th Design Automation Conf., S. 9–15 (1988)
37. Knapp, D.: Feedback-Driven Datapath Optimization in Fasolt. IEEE Int. Conf. on Computer-Aided Design (ICCAD), S. 300–303 (1990)
38. Koster, M., Geiger, M., Duzy, P.: ASIC Design Using the High-Level Synthesis System CALLAS: A Case Study. ICCD 90
39. Kodres, U.R.: Partioning and Card Selection, in: Breuer, M.A., (Hrsg.): Design Automation of Digital Systems, Vol. 1. Englewood Cliffs: Prentice-Hall 1972
40. Krämer, H., Rosenstiel, W.: System Synthesis using Behavioural Descriptions. 1st EDAC, 1990
41. Krämer, H., Rosenstiel, W.: Automatic Generation of Single Chip Multiprocessor Systems. IFIP Working Conf. on Logic and Architecture Synthesis, Paris 1990
42. Krogseter, M., Moldeklev, K.: Synthese und Optimierung integrierter Schaltungen: Scheduling, Allokieren und Zuweisen. Diplomarbeit am Forschungszentrum Informatik an der Universität Karlsruhe 1989
43. Ku, D., De Micheli, G.: Synthesis of ASIC's from Behavioural Specifications. IFIP Working Conf. on Logic and Architecture Synthesis, Paris 1990
44. Kurdahi, F.J., Parker, A.C.: REAL: A Program for Register Allocation. Proc. 24th Design Automation Conf., S. 210–215 (1987)
45. Lerch, E.: IN[3] – An Incremental Integrated Hardware Synthesis Tool Using 0–1 Integer Programming. Diplomarbeit Univ. Kiel 1990
46. Ly, T.A., Elwood, W.L., Girczyc, E.M.: A Generalized Interconnect Model for Data Path Synthesis. Proc. 27th Design Automation Conf., S. 168–173 (1990)
47. Marwedel, P.: Ein Software-System zur Synthese von Rechnerstrukturen und zur Erzeugung von Mikrocode. Habilitationsschrift, Institut für Informatik und Praktische Mathematik der Universität Kiel 1985, unveränderter Nachdruck: Forschungsbericht Nr. 356 des Fachbereichs Informatik der Universität Dortmund, 1990
48. Marwedel, P.: A New Synthesis Algorithm for the MIMOLA Software System. Proc. 23rd Design Automation Conf., S. 271–277 (1986)
49. Marwedel, P.: Matching System and Component Behaviour in MIMOLA Synthesis Tools. 1st EDAC, S. 146–156 (1990)
50. Marwedel, P., Schenk, W.: Improving the Performance of High-Level Synthesis. Microprogram. Microprocess. *27,* 381–388 (1989)
51. McFarland, M.C.: The VT: A Database for Automated Digital Design. Rep. DRC-01-4-80, Design Research Center, Carnegie-Mellon University, Pittsburgh 1978
52. McFarland, M.C.: Reevaluating the Design Space for Register Transfer Level Synthesis. IEEE Int. Conf. on Computer-Aided Design (ICCAD), S. 262–265 (1987)
53. McFarland, M.C., Parker, A.C., Camposano, R.: Tutorial on High-Level Synthesis. Proc. 25th Design Automation Conf., S. 330–336 (1988)
54. Mead, C., Conway, L.: Introduction to VLSI Systems. London: Addison-Wesley 1980
55. Nippon Telegraph and Telephone Corp.: HSL-FX User's Manual. NTT LSI Laboratories, Japan 1988
56. Paulin, P.G., Knight, J.P., Girzyc, E.F.: HAL: A Multi-Paradigm Approach to Automatic Datapath Synthesis. Proc. 23rd Design Automation Conf., S. 263–270 (1986)
57. Paulin, P.G., Knight, J.P.: Force-Directed Scheduling in Automatic Data Path Synthesis. Proc. 24th Design Automation Conf. (1987)
58. Paulin, P.G., Knight, J.P.: Scheduling and Binding Algorithms for High-Level Synthesis. Proc. 26th Design Automation Conf. (Extended Version), (1989)
59. Papachristou, C.A., Konuk, H.: A Linear Program Driven Scheduling and Allocation Method Followed by an Interconnect Optimization Algorithm. Proc. 27th Design Automation Conf., S. 77–83 (1990)
60. Park, N., Parker, A.C.: SEHWA: A Program for Synthesis of Pipelines. Proc. 23rd Design Automation Conf. (1986)
61. Parker, A.C., Pizarro, J.T., Mlinar, M.: MAHA: A Program for Datapath Synthesis. Proc. 23rd Design Automation Conf., S. 461–466 (1986)
62. Pfahler, P.: Übersetzermethoden zur automatischen Hardware-Synthese. Dissertation am Fachbereich Mathematik/Informatik der Universität-GH Paderborn 1988
63. Piloty, R.: CONLAN Report. Bericht RO 83/1, Institut für Datentechnik, Technische Hochschule Darmstadt 1983
64. Rietsche, G., Neher, M., Rosenstiel, W., Amann, R.: CASTOR: Control Part Synthesis in a Behavioural Synthesis System. ESSCIRC 89
65. Rosenstiel, W., Camposano, R.: Synthesizing Circuits from Behavioral Level Specifications. 7th Int. Symp. on Computer Hardware Description Languages and their Applications, CHDL 85 (1985)
66. Rosenstiel, W.: DSL – Eine Sprache zur Spezifikation der Funktion digitaler Systeme. Bericht 9/82, Institut für Informatik IV, Universität Karlsruhe 1982

67. Rudell, R., Sangiovanni-Vincentelli, A.: ESPRESSO-MV: Algorithms for Multivalued Logic Minimization. IEEE Custom Int. Circ. Conf. (1985)
68. Rudolph, M., Neher, M., Rosenstiel, W.: Test Scheduling and Controller Synthesis in the CADDY System. 2nd EDAC, S. 278–282 (1991)
69. Ruehli, A. E. (Hrsg.): Circuit Analysis, Simulation and Design, Part 1 & 2. Advances in CAD for VLSI. Amsterdam: North-Holland 1986
70. Springer, D. L., Thomas, D. E.: Exploiting the Special Structure on Conflict and Compatability Graphs in High-Level Synthesis. IEEE Int. Conf. on Computer-Aided Design (ICCAD), S. 254–257 (1990)
71. Stok, L.: Architectural Synthesis and Optimization of Digital Systems. Doktorarbeit, ISBN 90-9003966-X (1991)
72. Stroud, C. E.: An Automated BIST Approach for General Sequential Logic Synthesis. Proc. 25th Design Automation Conf., S. 3–8 (1988)
73. Thomas, D. E., Dirkes, E. M., et al.: The System Architect's Workbench. Proc. 25th Design Automation Conf., S. 337–343 (1988)
74. Trickey, H.: Flamel: A High-Level Hardware Compiler. IEEE Trans. Computer-Aided Design *6*, 259–269 (1987)
75. Wehn, N., Glesner, M., Held, M.: A Novel Scheduling/Allocation Approach for Datapath Synthesis Based on Genetic Paradigms. IFIP Working Conf. on Logic and Architecture Synthesis, Paris 1990
76. Wecker, T., Kumar, R., Rosenstiel, W., Krämer, H., Neher, M.: CALLAS – Ein System zur automatischen Synthese digitaler Schaltungen. Informatik – Forsch. Entw. *4*, 37–54 (1989)

Eingegangen 15. 10. 1990; in überarbeiteter Form 9. 8. 1991

Peter Marwedel
Lehrstuhl Informatik XII
Universität Dortmund, Postfach 50 05 00, W-4600 Dortmund 50

Wolfgang Rosenstiel
Lehrstuhl für Technische Informatik, Universität Tübingen,
und Forschungszentrum Informatik an der Universität Karlsruhe (FZI)
Sand 13, W-7400 Tübingen