

hierarchischer Beschreibungen, hierbei erfolgt eine Umsetzung des Hierarchie-Konzepts in Hardware mittels eines Handshake-Mechanismus.

#### 4.4 Repartitionierung und Resynthese

Für die weitere Bearbeitung der durch die High-Level-Synthese erhaltenen Register-Transfer-Struktur mittels eines Logik-Synthese-Werkzeugs ist in der Regel aus Komplexitäts-Gründen eine Partitionierung des Entwurfs in Blöcke erforderlich. Auf oberster Ebene erhält man hierbei eine Partitionierung in Steuerung und Datenpfad. Für eine günstige Steuerung der Partitionierung können Verfahren herangezogen werden, die Ähnlichkeiten von Bibliothekszellen oder den hierarchischen Aufbau komplexer Register-Transfer-Komponenten (z.B. aus 1-bit Addierern aufgebaute n-bit Addierer) berücksichtigen [2]. Ferner entsteht durch die Partitionierung an den Blockschnittstellen weiteres Optimierungspotential aufgrund zusätzlicher Don't-Care-Bedingungen. Ziel des Repartitionierungs-/Resynthese-Schrittes ist die Ausnutzung derartiger Partitionierungs-bedingter Optimierungspotentiale bei der Optimierung von Register-Transfer-Strukturen. Das hierfür eingesetzte Verfahren basiert auf einem iterativen Vertauschen von Schaltungsknoten über die Blockschnittstellen hinweg, gefolgt von einer partiellen Resynthese [3].

### 5 Zusammenfassung und Ausblick

In diesem Papier wurde eine Entwurfsumgebung für die Umsetzung von C-Verhaltensspezifikationen in Hardware-Strukturen vorgestellt, sowie wesentliche Merkmale und Eigenschaften der einzelnen Teilaktivitäten angegeben.

Ziele unserer augenblicklichen Forschungsarbeiten liegen im Bereich Hardware/Software-Codesign in der Bereitstellung von Verfahren, die eine schnelle Abschätzung der Auswirkung einer Hardware/Software-Partitionierung gestatten. Im Bereich High-Level-Transformationen erfolgt momentan eine Implementierung einer Transformationsumgebung, und im Bereich Repartitionierung/Resynthese werden Möglichkeiten für eine Integration von Restriktionen in die Zustandskodierung untersucht.

### Literatur

- [1] A.V.Aho, R.Sethi & J.D.Ullman, *Compilers: Principles, Techniques and Tools*. Addison Wesley, 1986.
- [2] H.-J. Eikerling. Repartitionierung regulärer, hierarchischer Datenpfade. In *40th Intern. Wiss. Kolloquium Ilmenau*, Ilmenau, September 18-21 1995. Also available as technical report SFB 358 - B2 - 1/95 at TU Dresden.
- [3] H.-J. Eikerling, R. Hunstock, and R. Camposano. Optimization of Hierarchical Designs using Partitioning and Resynthesis. In *Proceedings of the International Conference on Computer-Aided Design*, San Jose, CA, November 6-10 1994. Also available as technical report SFB 358 - B2 - 9/94 at TU Dresden.
- [4] D. Gajski, N. Dutt, A. Wu, S. Lin. *High-level Synthesis: Introduction to Chip and System Design*. Kluwer Academic Publishers, 1992.
- [5] W. Hardt, W. Rosenstiel. Performanz-Abschätzung für eingebettete Systeme. In diesem Band.
- [6] W. Rosenstiel. Optimizations in High-Level Synthesis. *Microprogramming and Microprocessing*, 18:543-549, 1986.
- [7] E.M. Sentovich, K.J. Singh, L. Lavagno et al. SIS: A system for sequential circuit synthesis. Technical Report Memorandum No. UCB/ERL M92/41, University of California at Berkeley, May 1992.
- [8] Synopsys, Inc., Mountain View, CA. *VHDL Design Analyzer (tm) Manual*, 3.3a edition, 1995.
- [9] W. Wolf. Hardware-software codesign of embedded systems. In *Proceedings of the IEEE*, 83((7)):967-989, 1994

## 4.1 Hardware/Software-Codesign

Ziel des Hardware/Software-Codesign-Schrittes ist die Partitionierung der C-Spezifikation in einen Teil, welcher in Form von Software realisiert und auf einem Standardprozessor (in unserem Fall einem SPARC-Prozessor) ausgeführt werden soll und in einen Teil, welcher als Spezial-Hardware (als ASIC) implementiert werden soll. Hierzu wird innerhalb einer Spezifikations-Analyse, welche sowohl statische als auch dynamische Charakteristika der einzelnen Module innerhalb der C-Spezifikation berücksichtigt, für jedes Modul ein Kostenvektor evaluiert, welcher für die Entscheidung, ob das entsprechende Modul am zweckmäßigsten in Hardware oder in Software realisiert werden sollte, herangezogen wird. Da eine detaillierte Betrachtung dieser Entwurfsaktivität im Rahmen eines weiteren Beitrags innerhalb dieses Workshops durchgeführt wird, sei an dieser Stelle auf das entsprechende Papier verwiesen [5].

## 4.2 High-Level-Transformationen

In diesem Schritt sollen die innerhalb der Hardware/Software-Partitionierung als für eine Hardware-Realisierung geeignet erkannten Segmente der C-Spezifikation im Hinblick auf die anschließende High-Level-Synthese optimiert werden [6]. Hierbei sollen sowohl Verfahren aus dem Bereich des Compilerbaus (z.B. Constant Propagation, Common Subexpression Elimination oder Loop Unrolling) [1] als auch Hardware-nahe Transformationen (z.B. Prozeß-Generierung) zum Einsatz kommen. Ein weiteres, wesentliches Ziel im Zusammenhang mit High-Level-Transformationen liegt in der Bereitstellung von Verfahren, die eine schnellen Abschätzung der Transformations-Ergebnisse erlauben, was Ausgangspunkt für eine Automatisierung der Transformationsanwendung darstellt. Zur Unterstützung einer schnellen Transformationsdurchführung bzw. Abschätzung des Transformations-Ergebnisses setzen unsere Transformationen direkt auf dem Kontroll-/Datenfluß-Graph (und nicht etwa auf dem Source-Code oder einem anderen höheren Zwischenformat) auf.

## 4.3 High-Level-Synthese

Die so optimierte Verhaltensbeschreibung wird innerhalb der High-Level-Synthese in eine Struktur-Beschreibung auf Register-Transfer-Ebene transformiert [4]. In Abbildung 3 sind die einzelnen Teilaktivitäten der High-Level-Synthese dargestellt. Nach dem momentanen Stand der Implementierung erfolgt das Scheduling wahlweise nach dem (Ressourcenbeschränkten) Static-List Scheduling-Verfahren oder nach dem (Zeit-beschränkten) Force-directed Scheduling-Verfahren. Dabei wurden beide dieser, in ihrer ursprünglichen Form Basic-Block-orientierten Verfahren derart erweitert, daß neben reinem Datenfluß auch eine Bearbeitung von Kontroll-Konstrukten möglich wird. Sowohl das List Scheduling- als auch das Force-directed Scheduling-Verfahren führen Scheduling und Allokierung von funktionalen Einheiten simultan durch. Für die Allokierung wurde dabei ein straight-forward-Algorithmus verwendet, welcher innerhalb einer gegebenen Bibliothek den jeweils ersten Modultyp selektiert, der für die Ausführung der betrachteten Operation verwendet werden kann. Beim anschließenden Binding wird die jeweils erste freie Instanz dieses Modultyps ausgewählt und der entsprechenden Operation zugeordnet. Register-Allokierung und Register-Binding erfolgen im Anschluß an eine Life-time-Analyse nach dem Left-edge-Algorithmus. Das Interconnection-Binding wird ebenfalls straight-forward mittels einer einfachen Traversierung des Datenfluß-Graphen durchgeführt.

Unser Synthese-System erzeugt Multiplexer-basierte Entwürfe, generiert also eine Multiplexer-Register-Multiplexer-Operator-Struktur. Ferner erlaubt unser System eine Bearbeitung

schiedlicher Module zur Verfügung gestellt werden, die dann - soweit der jeweilige Kontext dies erlaubt - in beliebiger Weise miteinander kombiniert bzw. gegeneinander ausgetauscht werden können. Sind sämtliche Synthese-Schritte durchlaufen, ist die Annotierung also vollständig, kann eine 1:1-Umsetzung des Kontroll-/Datenfluß-Graphen in eine Register-Transfer-Struktur bestehend aus einer Steuerung und einem Datenpfad vorgenommen werden.

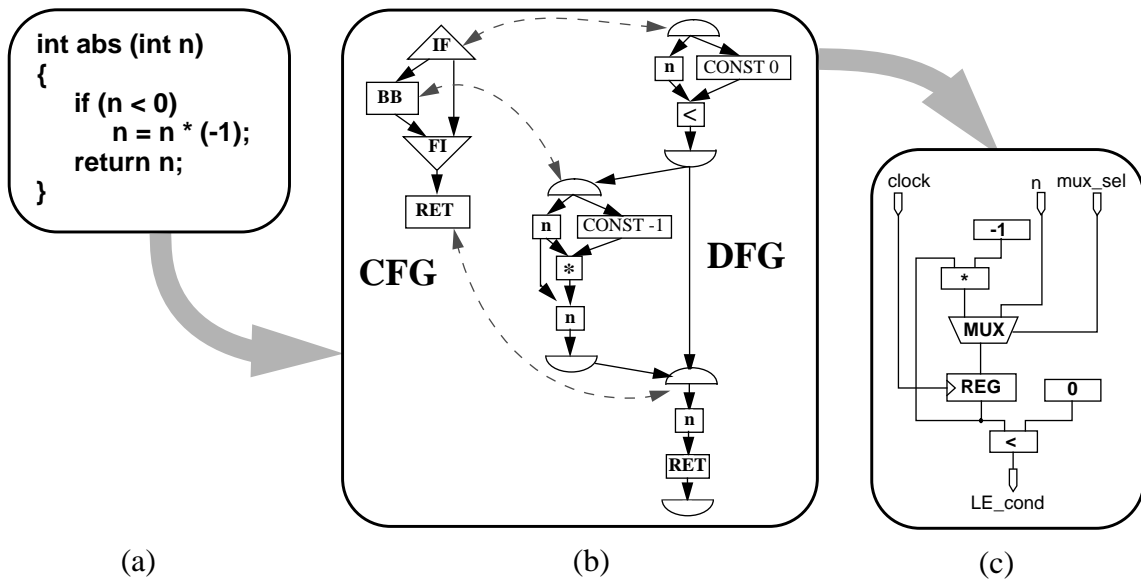


Abbildung 3. Formate für die Entwurfsdatendarstellung.

## 4 Entwurfsaktivitäten

Unser Entwurfswerkzeug umfaßt Entwurfsaktivitäten aus den Bereichen Hardware/Software-Codesign, High-Level-Transformationen, High-Level-Synthese sowie Repartitionierung/Resynthese. Ausgangspunkt ist dabei eine in Form von C-Code gegebene Problemspezifikation. Dabei wurde der Sprachumfang von C um zusätzliche Hardware-spezifische Attribute (z.B. zur Spezifikation von Wortbreiten von Variablen) erweitert.

Dieser Ansatz bietet gegenüber einer Verhaltensbeschreibung in einer „konventionellen“ Hardware-Beschreibungssprache wie etwa VHDL eine Reihe von Vorteilen:

- Man erhält auf diese Weise eine homogene Gesamtbeschreibung von sowohl Hardware- als auch Software-Komponenten, wodurch zum einen die Spezifikationsbeschreibung für den Entwurf, zum anderen auch die Co-Simulation von Hardware und Software wesentlich vereinfacht wird.
- Für den funktionalen Test der Spezifikation ist keine Simulation erforderlich, sondern es kann eine direkte Ausführung des übersetzten C-Codes vorgenommen werden.
- Die Gefahr einer Überspezifikation der Verhaltensbeschreibung, also der Festlegung von Entwurfsentscheidungen durch den Entwerfer, die sinnvollerweise durch das Synthesystem getroffen werden sollten, ist wesentlich geringer als etwa in VHDL.

In den folgenden Abschnitten sollen die einzelnen, von unserem System unterstützten Teilaktivitäten näher betrachtet werden.

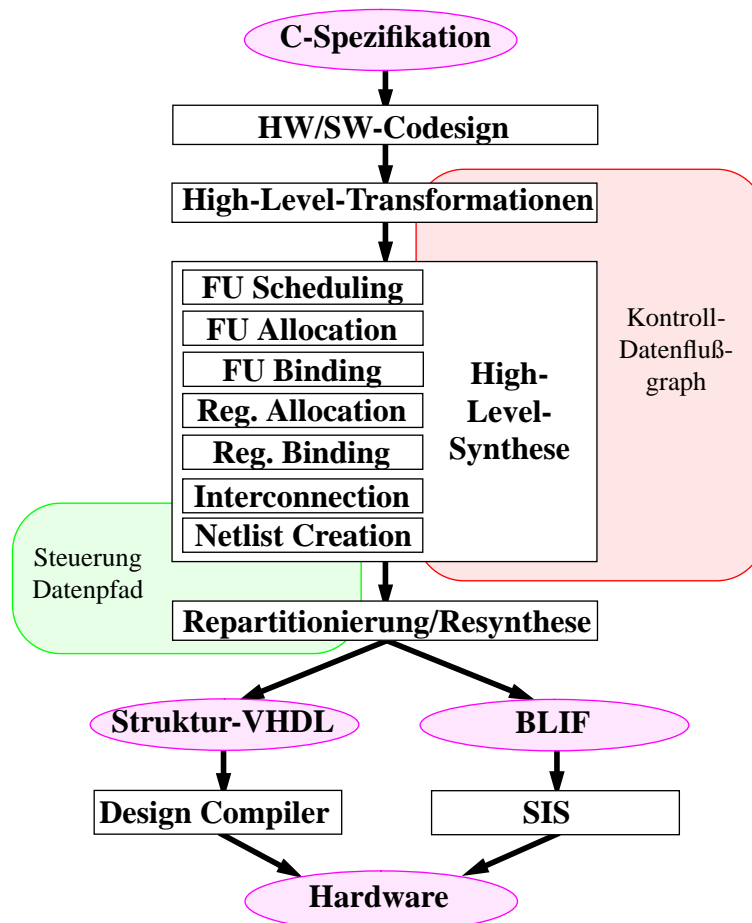


Abbildung 2. Transformation von C-Spezifikationen in Hardware-Strukturen

### 3 Darstellung der Entwurfsdaten

Wesentliches Merkmal unseres Synthese-Systems ist dessen hochgradig modularer Aufbau bezüglich Kombinierbarkeit und Austauschbarkeit einzelner Synthese-Algorithmen. Dies wird erreicht durch Verwendung einer einheitlichen Datenbasis in Form eines Kontroll-/Datenfluß-Graphen, sowie durch das zugrunde gelegte Annotierungs-Schema. Der Kontroll-Datenfluß-Graph besteht aus einem Kontrollfluß-Graph, einem Datenfluß-Graph sowie aus wechselseitigen Beziehungen zwischen diesen beiden Datenstrukturen. Abbildung 3 zeigt in (a) ein C-Code-Segment (als Beispiel diene ein Modul zur Berechnung des Absolutwerts einer Integer-Zahl). (b) zeigt die entsprechende Kontroll-/Datenfluß-Graph-Darstellung und (c) einen Ausschnitt des durch die High-Level-Synthese erzeugten Datenpfads.

Sämtliche Algorithmen der High-Level-Synthese beziehen die innerhalb des Algorithmus benötigten Informationen aus dem Kontroll-/Datenfluß-Graph und legen die im Algorithmus ermittelten Informationen (z.B. Cstep, Modultyp und Modulinstanz einer Operation) ebenfalls in diesem ab („annotieren“ diesen also mit Information). Dies hat zur Folge, daß die Ausführbarkeit eines bestimmten Synthese-Algorithmus rein durch das Vorhandensein der entsprechenden Information im Kontroll-/Datenfluß-Graph beschränkt wird. Auf diese Weise können für einzelne Synthese-Schritte verschiedenen Algorithmen in Form unter-

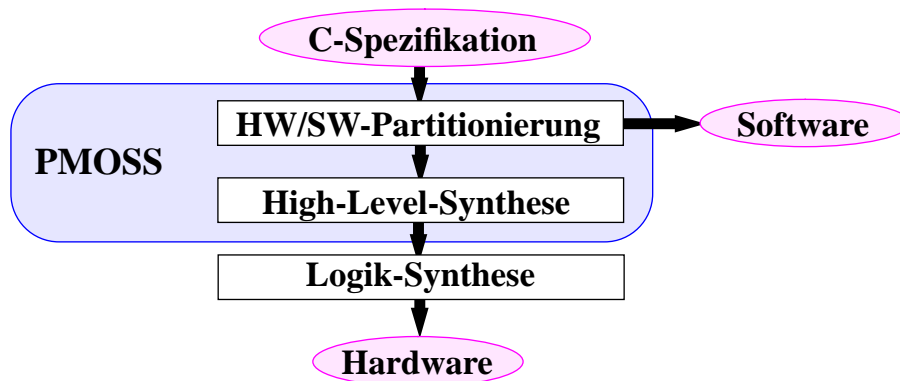


Abbildung 1. Entwurf von HW/SW-Systemen aus C-Spezifikationen.

Während für die Logik-Synthese, also die Umsetzung einer Register-Transfer-Struktur in eine Hardware-Struktur, bereits leistungsfähige kommerzielle Werkzeuge zur Verfügung stehen, stellen Hardware/Software-Partitionierung und High-Level-Synthese Probleme dar, die momentan im wesentlichen Gegenstand universitärer Forschungen sind. Die im folgenden vorgestellte Entwurfs-Umgebung PMOSS<sup>1</sup> umfaßt die oberhalb der Logik-Synthese angesiedelten Entwurfsaktivitäten.

Im folgenden Abschnitten wird zunächst ein Überblick über die von unserem Entwurfssystem unterstützte Vorgehensweise angegeben. Im Anschluß daran wird das innerhalb der Synthese für eine interne Entwurfsdatendarstellung gewählte Zwischenformat vorgestellt. Danach wird auf die einzelnen, von unserem Werkzeug unterstützten Entwurfsaktivitäten näher eingegangen. Den Schluß bildet eine Zusammenfassung mit Ausblick.

## 2 Überblick

Ausgangspunkt unserer Betrachtungen ist eine in Form von C-Code gegebene Problemspezifikation. Innerhalb dieser Beschreibung werden in einem ersten Schritt (*Hardware/Software-Codesign*) zunächst Code-Segmente identifiziert, welche sich für eine Realisierung als Hardware eignen. Diese werden anschließend in eine Struktur-Beschreibung auf Register-Transfer-Ebene transformiert (*High-Level-Synthese*). Dabei werden sowohl vor als auch nach der eigentlichen High-Level-Synthese Optimierungen der jeweiligen Entwurfsdarstellungen vorgenommen: *Vor* der High-Level-Synthese wird eine Optimierung der Verhaltensbeschreibung durchgeführt, innerhalb welcher hauptsächlich Techniken aus dem Bereich des Compilerbaus zum Einsatz kommen (*High-Level-Transformationen*), *nach* der High-Level-Synthese erfolgt eine Optimierung der Register-Transfer-Struktur durch Ausnutzung von Optimierungspotentialen, die sich aus einer (für die weitere Bearbeitung der Schaltung durch ein Logik-Synthese-System aus Komplexitäts-Gründen im allgemeinen erforderlichen) Partitionierung des Entwurfs ergeben (*Repartitionierung/Resynthese*). Die so erhaltene Register-Transfer-Struktur wird anschließend mittels eines externen Logik-Synthese-Systems in die eigentliche Hardware-Struktur transformiert. Hierfür kommt in unserem Fall der Design-Compiler der Firma Synopsys [8] oder das nicht-kommerziellen Logik-Synthese-System SIS [7] zum Einsatz. Abbildung 2 zeigt einen Überblick über die Vorgehensweise bei der Umsetzung einer C-Spezifikation in eine Hardware-Struktur.

1. Paderborn MOdular System for Synthesis and HW/SW-Codesign

# Von C nach Hardware: ein integratives Entwurfskonzept<sup>1</sup>

Joachim Gerlach<sup>†</sup>, Heinz-Josef Eikerling<sup>†</sup>, Wolfram Hardt<sup>††</sup>, Wolfgang Rosenstiel<sup>†</sup>

{gerlach, eikerlin, rosenstiel}@peanuts.informatik.uni-tuebingen.de  
hardt@uni-paderborn.de

<sup>†</sup>Sonderforschungsbereich 358  
Universität Tübingen,  
Arbeitsbereich Technische Informatik  
Sand 13,  
D-72 076 Tübingen

<sup>††</sup>Sonderforschungsbereich 358  
Universität Paderborn,  
FB 17 Mathematik-Informatik  
Fürstenallee 11,  
D-33 102 Paderborn

**Kurzfassung:** Dieses Papier gibt einen Überblick über eine Entwurfsumgebung, die eine abstrakte Problemspezifikation auf Algorithmen-Ebene, in unserem Fall gegeben in Form eines C-Programms, in ein System bestehend aus untereinander interagierenden Hardware- und Software-Komponenten transformiert. Unser Werkzeug umfaßt dabei Entwurfsaktivitäten aus den Bereichen Hardware/Software-Codesign, High-Level-Transformationen, High-Level-Synthese sowie Repartitionierung/Resynthese. Charakteristisch für unser System ist dessen hochgradig modularer Aufbau, was zum einen eine problemorientierte Auswahl und Kombination von Algorithmen und zum anderen eine flexible Integration unseres Systems in einen Gesamtentwurfsablauf gestattet.

## 1 Einleitung

Während noch vor wenigen Jahren Hardware- und Software-Entwurf als voneinander unabhängige Probleme betrachtet wurden, gewinnt in letzter Zeit ein unter dem Schlagwort *Hardware/Software-Codesign* bekannter Ansatz zunehmend an Wichtigkeit [9]. Dabei wird das Problem des Hardware- und Software-Entwurfs als *ein* Gesamtproblem verstanden, bei welchem eine abstrakte Verhaltensbeschreibung in eine Implementierung bestehend aus sowohl Hardware- als auch Software-Komponenten transformiert wird. Diese Vorgehensweise unterstützt die schnelle Generierung und Analyse eingebetteter Systeme, gestattet die Ausnutzung eines größeren Optimierungspotentials und versetzt auch Nicht-Experten in die Lage, komplexe Systeme zu entwerfen.

Dazu werden zunächst innerhalb einer Hardware/Software-Partitionierung Segmente der in Form von C-Code gegebenen Verhaltens-Spezifikation identifiziert, welche sich für eine Realisierung in Hardware eignen und diese anschließend in eine Hardware-Struktur synthetisiert. Dabei wird eine Zielarchitektur zugrunde gelegt, die aus einem Standardprozessor besteht, welcher um zusätzliche Hardware-Komponenten erweitert wurde. Abbildung 1 zeigt die Grobstruktur des Entwurfsablaufs.

---

1. Diese Arbeit wurde unterstützt durch die Deutsche Forschungsgemeinschaft, Teilbereich „Automatisierter Systementwurf“, SFB-358-B3.