Neuere Trends der Hardwareentwurfmethodik als Richtschnur für die Softwareentwicklung

Franz J. Rammig
Universität-GH Paderborn
Fachbereich Mathematik/Informatik

1 Kurzfassung

In diesem Beitrag wird versucht, neuere Entwicklungen bei der Methodik des Hardwareentwurfs aufzuzeigen und zu untersuchen, inwieweit sich dadurch der Entwurf und die Implementation von Software beeinflussen lassen. Zunächst wird versucht, den Hardwareentwurf als einen vielfach rückgekoppelten zyklischen Prozess über eine Anzahl von Abstraktionsebenen hinweg zu charakterisieren. Dabei gelingt es, wenige Klassen von Entwurfsaktivitäten zu identifizieren, die auf allen Abstraktionsebenen immer wieder vorkommen. Sodann werden grundsätzliche Entwurfsstrategien vorgestellt, die beim Hardwareentwurf sowohl von der Methode der schrittweisen Verfeinerung wie auch, und dies vor allem auf niedrigeren Abstraktionsebenen, von der Aggregation vorentworfener Standardbausteine geprägt ist. Bedingt durch enorme Änderungskosten ist beim Hardwareentwurf eine sehr rigide Verifikations- und Testpraxis verbreitet. Darauf wird am Ende des Beitrages eingegangen werden.

2 Sichten und Abstraktionsebenen

Es haben sich beim Entwurf komplexer Hardwaresysteme eine Reihe von Abstraktionsebenen herausgebildet, auf denen Modelle der zu konstruierenden Hardware angefertigt werden. Orthogonal zu diesen Abstraktionsebenen gibt es verschiedene Sichten, von denen aus das zu entwerfende Objekt betrachtet werden. Hier hat sich, basierend auf einen Vorschlag von Gajski [GAJ87], ein System von drei Sichten etabliert:

- die **Verhaltenssicht** beschränkt sich darauf, das Verhalten, d.h. die Werteverläufe der Bestimmungsgrößen des zu modellierenden Objekts über die Zeitachse hinweg darzustellen. Dazu bedient man sich auf den unterschiedlichen Abstraktionsebenen ganz unterschiedlicher Paradigmen.
- Mit der **Struktursicht** wird dargestellt, wie sich das Objekt aus Unterobjekten zusammensetzt. Es werden sowohl die hierarchische Struktur ("besteht aus"-Relation) wie auch detaillierte Konnektivitätsstrukturen modelliert.
- Die **Geometriesicht** schließlich beschreibt, wie die Objekte geometrisch im Raum angeordnet sind. Je nach Ebene beschränkt man sich dabei auf topologische (d.h. nur mit relativen Maßen und Orten versehene) Informationen, oder gibt präzise geometrische Angaben.

Jede dieser drei Sichten ist auf jeder Abstraktionsebenen präsent, wenn auch zu beobachten ist, daß die geometrische Information von den höheren Ebenen zu den tieferen an Bedeutung gewinnt. Es gibt kein allgemein vereinbartes Ebenenmodell, doch spiegelt das in [RAM89] wiedergegebene System weit verbreitete Vorstellungen wider. Hier wird von sechs Abstraktionsebenen ausgegangen. Sie sollen hier nur von der Verhaltenssicht aus kurz diskutiert werden, da aus dieser Sicht die sehr unterschiedlichen Modellierungskonzepte besonders klar sichtbar werden.

Als oberste Ebene betrachtet man beim Hardareentwurf die **Systemebene**. Hier hat man die Vorstellung kooperierender, bis auf wohldefinierte Kommunikation autonomer Bausteine. All diese Bausteine sind Prozessoren im weitesten Sinn, d.h. sie haben einen Instruktionssatz über den Operationen (Dienste) angefordert werden können. Der objektorientierte Ansatz ist daher, lange bevor er unter diesem Namen populär wurde, ein im Hardwareentwurf vertrautes Konzept. Es muß allerdings darauf hingewiesen werden, daß im Hardwarebereich oft von Bausteinen ausgegangen wird, die sich aktiv um Aufträge bemühen. Ein klassischer Prozessor ist nur durch Abschalten davon abzubringen, sich aus dem Speicher ständig neue Instruktionen zu beschaffen. Hier sind Entsprechungen auf der Softwareseite weniger verbreitet. Auf der Systemebene hat man meist ein rein kausales Zeitmodell. Dies bedeutet, daß die beim Hardwareentwurf so essentiellen zeitlichen Abläufe auf Kausalabhängigkeiten von Ereignissen reduziert werden.

Geht man auf der Systemebene von der Existenz von Bausteinen mit Instruktionssätzen aus, so gilt es auf der **algorithmischen Ebene** für diese Instruktionssätze Interpretationsalgorithmen zu spezifizieren. Diese Algorithmen sind in der Regel hochgradig nebenläufig, was sich aus Leistungsanforderungen ergibt, die nur durch parallel arbeitende Operationswerke und Pipelining erfüllt werden können. Für diese Ebene haben sich daher Spezifikations- und Beschreibungssprachen entwickelt, die ein hochparalleles imperatives Programpmieren erlauben. Als Beispiel sei hier DACAPO III [DAC87] genannt. Auch auf dieser Ebene benutzt man in der Regel noch ein Kausalsystem zur Beschreibung der Zeitabläufe. Dies hat eine Entsprechung in semantischen Modellen der benutzten Sprachen. Dies ist im Falle von DACAPO III z.B. das der interpretierten Petrinetze.

Der Übergang zur nächst niedrigen Ebene, der **Registertransferebene** stellt die wohl wichtigste Transformation in der Modellierung eines Hardwaresystems dar. Hat man auf der algorithmischen Ebene eine imperative Sichtweise, man kann auch sagen die Sichtweise der Steuerung, so betrachtet man auf der Registertransferebene das Objekt reaktiv aus Sicht der Einzelkomponenten. Eine derartige Komponente wird hier als passiv angesehen, bis ein spezielles, für die jeweilige Komponente aktivierendes Ereignis eintritt. In diesem Fall führt die Komponente ihre dem aktivierenden Ereignis zugeordnete Operation durch, um danach wieder in einen passiven Zustand zurückzukehren. Zur Beschreibung dieser Ebene haben sich wieder spezielle Techniken herausgebildet, die konzeptionell eine enge Verwandschaft mit den "Guarded Commands" [DIJ75] haben. Eine semantische Bedeutung der Anordnung von Anweisungen in Beschreibungen auf dieser und allen niedrigeren Ebenen gibt es natürlich nicht (nonprozedurale Modellierung). Grundsätzlich kann man sagen, daß für den Hardwareentwurf Parallelität den Normalfall darstellt, sequentielle Abläufe nicht nur die Ausnahme sind, sondern auch als Leistungsbremsen höchst unerwünscht sind. Auf der Registertransferebene führt man üblicherweise zum ersten mal Realzeitkonzepte ein, indem man die Aktionen an Takte und Taktphasen bindet.

Auf der Registertransferebene hat man bereits implizit eine strukturelle Auflösung von Gesamtverhalten in Einzelverhalten vorgenommen. Diese Auflösung wird auf der **Gatterebene** noch weiter fortgesetzt. Hier gibt man nun die Unterscheidung zwischen Kontrollereignissen und Datenmanipulationen auf, und beschreibt das System rein funktional durch ein System von, im Idealfall Booleschen, Gleichungen. Dieses Gleichungssystem kann durch externe Ereignisse stimuliert, d.h. aus dem Gleichgewicht gebracht werden. Dies bedeutet aber nichts anderes, daß aktuell keine Lösung des Gleichungssystems mehr vorliegt. Man geht bei der Modellierung auf dieser Ebene nun davon aus, daß ein derartiges destabilisiertes System aktiv eine neue Lösung zu erreichen anstrebt. Dies gelingt auch immer, sofern nicht ein inhärent astabiles Sytem vorliegt. Jedes Hardwaresystem besitzt zumindestens eine astabile Komponente, die den elementaren Antrieb des Systems darstellt. Da man auf dieser Ebene das Wissen über Kontrollereignisse aufgegeben hat, kennt man auch keine Takte mehr, kann das Zeitverhalten auch nicht mehr auf dieser Basis angeben. Man modelliert daher auf der Basis sehr präziser Realzeitangaben. Zumindest bei Hochleistungshardware besteht eine wesentliche Kunst des Hardwareentwurfs darin, mit diesen Realzeitparametern optimal umzugehen.
Das grundsätzliche Paradigma des stimulierbaren Gleichungssystems wird auf den letzten beiden Ebenen beibehalten, nur die Wertebereiche der Darstellungsgrößen und die benutzten Elementarkomponenten unterscheiden sich. Auf der **Schalterebene** geht man davon aus, daß man als Elementarkomponenten Knoten, die Ladung speichern können, und Schalter hat. Um die unterschiedliche Speicherfähigkeit verschiedener Knoten darstellbar zu machen, ist man darauf angewiesen, unterschiedliche Signalstärken darstellen zu können. Dies erweitert natürlich den Wertebereich über den Booleschen hinaus. Zudem muß hier die unidirektionale Sichtweise aufgegeben werden. Man hat also Gleichungssysteme vorliegen, bei denen die rechten und linken Seiten der Gleichungen absolut gleichberechtigt und austauschbar sind.
Dies gilt im gleichem Maße natürlich auch für die **elektrische Ebene**, auf der das exakte elektrische Verhalten in einer Wechselwirkung von Spannungen, Strömen, Kapazitäten, Induktivitäten und Widerständen beschrieben wird. Hier hat man es natürlich mit reellen Wertebereichen zu tun. Der Wertebereich des Zeitmodells ist schon ab der Gatterebene reell, sodaß die elektrische Ebene am adäquatesten durch ein System von Differentialgleichungen beschrieben werden kann.

3 Idealisierte Modelle des Hardwareentwurfs

Eine Spezifikation auf jeder der beschriebenen Ebenen besteht aus zwei Teilen: der Spezifikation der erwarteten Funktionalität und der Angabe der zu respektierenden Restriktionen [GUH78]. Diese Zweiteilung findet man bei den Entwurfsaktivitäten wieder. Hier gibt es eine generierende Aktivität, die aufgrund der funktionalen Spezifikation eine Systembeschreibung erstellt und eine überprüfende, die testet, ob dieses so erhaltene System den genannten Restriktionen genügt. Ist dies nicht der Fall, so wird die generierende Aktivität mit modifizierter Eingabe reaktiviert. Dieser rückgekoppelte Prozess wird solange fortgesetzt, bis ein stabiler Zustand erreicht ist. Die generierende Aktivität kann entweder auf einer Abstraktionsebene bleiben (Optimierung, Transformation), eine niedrigere Ebene als Ziel haben (Implementierung), oder

eine höhere (Aggregation). Dabei muß eine ebenenübergreifende Generierung stets von einer gegenläufig gerichteten Überprüfung begleitet sein. Um dies einzusehen, sei von einer Aggregation ausgegangen, die von einer niedrigeren Ebene i auf eine höhere Ebene j aggregiert. Das Ergebnis ist dann auf der Ebene j gegen die dort gültigen Restriktionen zu überprüfen. Sind diese nicht erfüllt, wird man zunächst durch Optimierungen auf der Ebene j versuchen, den Restriktionen doch noch zu genügen. Gelingt dies nicht, so ist die Aggregation gescheitert und der niedrigeren Ebene i ist durch eine dorthin gerichtete Überprüfungsaktivität mitzuteilen, daß ein anderer Ansatz gewählt werden muß. Im Falle einer "topdown"-Vorgehensweise gilt analoges. Die "reinen Lehren" des Entwurfs, nämlich striktes "top down" und striktes "bottom up" werden auch für den Hardwareentwurf genannt, sind aber auch hier in ihrer reinen Form unrealistisch. Beim strikten "top down"-Entwurf versucht man, eine abstrakte Spezifikation durch schrittweise Verfeinerung in eine Implementierung auf einer niedrigen Abstraktionsebene zu überführen. Im Softwarebereich kann die Idee der strukturierten Programmierung als programmtechnische Ausprägung dieser Vorstellung angesehen werden. Ein wesentlicher Unterschied besteht darin, daß es in den meisten Programmiersprachen, die vorgeben, die strukturierte Programmierung zu unterstützen, nicht möglich ist, die noch nicht verfeinerten Spezifikationen auszuführen. Dies liegt nicht zuletzt daran, daß im Softwarebereich das Konzept der allgemein akzeptierten Abstraktionsebenen fehlt. Im Hardwarebereich liegt es nahe, abstrakten Spezifikationen eine auf der jeweiligen Abstraktionsebne gültige operationale Semantik zuzuordnen und damit auch ausführbar zu machen. Dies geschieht entweder durch spezielle, den jeweiligen Ebenen zugeordnete Simulatoren, oder durch Breitbandsimulatoren. Es besteht natürlich kein Zwang, Verfeinerungsschritte (Implementationsschritte) nur zur jeweils unmittelbar benachbarten nächst niedrigeren Ebene vorzunehmen. Es ist im Extremfall denkbar, eine Systemspezifikation unmittelbar in Layout umzusetzen. Wegen der enormen Komplexität der Aufgabenstellung findet jedoch fast immer eine schrittweise Implementation auf die jeweils nächst niedrigere Ebene statt.

Entspricht die strukturierte Programmierung der "top down"-Vorgehensweise, so gilt eine ähnliche Entsprechung zwischen der "bottom up"-Strategie und der objektorientierten Programmierung. Bei dieser Entwurfsstrategie versucht man durch geeignete Kombination vorgefertigter Bibliothektskomponenten eine Implementation der, auch in diesem Fall auf abstrakter Ebene vorgegebenen, Spezifikation zu finden. Diese Vorgehensweise hat beim Hardwareentwurf eine lange Tradition, wobei die benutzten Komponenten und damit die angebotenen Methoden immer komplexer wurden. Stand am Anfang ein Angebot elementarer logischer Gatter und einfacher Flipflops (Small Scale Integration, SSI) so entwickelte sich das Angebot über Registertransferkomponenten wie Register, ALUs, Multiplexer (Medium Scale Integration, MSI) oder Prozessor Bitslices, UARTs (Large Scale Integration, LSI) bis hin zu Komponenten der Systemebene wie Prozessoren, LAN-Bausteinen, Kanälen, Digitale Signalprozessoren (DSP), Coprozessoren, PLAs (Very Large Scale Integration, VLSI). Die Umsetzung einer Spezifikation in eine gefertigte Hardwarekomponente ist überaus langwierig und teuer. Dies hatte von Anfang an die Wirkung, daß ein Hardwareentwickler fast immer zunächst versucht, das zu realisierende System möglichst nahe durch ein vorgefertigtes Bibliothekselement oder eine Verschaltung derartiger Elemente zu approximieren. Nur für die Teile, die unbedingt notwendig sind, um eine so gefundene Lösung an die Spezifikation anzupassen, nimmt ein Hardwareent-

wickler detaillierte Schaltungsentwicklungen vor. Im Idealfall beginnt dieser Entwurfsstil auf der Systemebene, hinterläßt dort Anpassungsteile (sogenannte "glue logic"), die in gleicher Weise auf der Registertransferebene behandelt wird, und so fort, bis der endgültige Entwurf erhalten ist. Oft findet aber ein direkter Entwurf von "glue logic" auf der Gatterebene für Systementwürfe statt. In der Praxis werden stets Mischformen dieser Entwurfsstrategien verfolgt, in denen iterativ verfeinernde und aggregierende Aktivitäten eingesetzt werden. Wird bei einem derartigen Vorgehen als Hauptrichtung eine "top down"-Methode verfolgt, so spricht man vom Yoyo-Entwurfsstil, im Falle einer hauptsächlichen "bottom up"-Vorgehensweise scherzhaft vom australischen Yoyo.
Man beobachtet also, daß es für den Hardwareentwurf gelingt, ein relativ starres Raster von Aktivitäten zu identifizieren und auf dieser Basis wohl definierte Entwurfsstrategien zu verfolgen. Dabei ist es von großer Bedeutung, daß es ein etabliertes System von Abstraktionsebenen und orthogonal dazu ein ebenso etabliertes System von Sichten gibt. Dies erlaubt es, Entwurfsaktivitäten in einheitlicher Weise zu identifizieren und damit auch Werkzeuge mit hinreichend großer Verbreitung anzubieten. Eine ähnliche Situation sehe ich beim Softwareentwurf hauptsächlich im Compilerbau [WHG85]. Der dort weit verbreitete Konsens über die einzelnen Aktivitäten hat auch hier dazu geführt, daß weit verbreitete Werkzeuge zur Verfügung stehen. Dies wiederum führt zu einer vergleichsweise hohen Produktivität in diesem Bereich des Softwareentwurfs. Daß der Compilerbau an Hochschulen so intensiv gelehrt wird, liegt nicht zuletzt daran, daß dadurch eine strenge Systematik des Entwurfs unter Verwendung vieler vordefinierter Moduln trainiert werden kann. Die Etablierung allgemein akzeptierter Abstraktionsebenen hatte im Bereich des Hardwareentwurfs zudem den Effekt, daß man sich auf genormte Schnittstellen einigen konnte. Hiervon scheint man im Softwarebereich noch weit entfernt zu sein. Diese Schnittstellen sind in vielen Fällen als Defakto- oder Industriestandards entstanden. Beispiele sind SPICE (die Eingabesprache des gleichnamigen Simulators auf der elektrischen Ebene) als allgemein akzeptierte Schnittstelle für die Verhaltenssicht auf der elektrischen Ebene, GDS II oder CIF als Schnittstelle für die Geometriesicht auf der elektrischen Ebene oder EDIF [EDI87] als Schnittstelle für die Struktur- und die Geometriesicht für die Ebenen unterhalb der Gatterebene einschließlich. Es soll hier nicht verschwiegen werden, daß es für diese Schnittstellen auch einen enormen Bedarf gibt, da Hardware in der Regel extrem arbeitsteilig, oft durch verschiedene Institutionen durchgeführt wird. Als Beispiel mag dienen, daß der Entwurf bis hinunter zur Gatterebene von einem Systemhaus und darunter von einem Halbleiterhersteller durchgeführt wird, eine Vorgehensweise, die ohne genormte Schnittstellen und Austauschformate undenkbar wäre.

4 Werkzeuge für den Hardwareentwurf

Der breite Konsens über Abstraktionsebenen und Entwurfsaktivitäten hat auch zu Folge, daß es sich lohnt, für einen breiten Markt umfangreiche Bibliotheken vorgefertigter Bausteine und ein ebenso breites Angebot von standardisierten Entwurfswerkzeugen anzubieten. Für die Elemente der Bauteilbibliotheken lohnt sich wegen der möglichen Marktverbreitung ein enormer Entwicklungsaufwand, der in der Regel zu hochgradig optimierten Komponenten führt. Kom-

ponenten vergleichbarer Qualität lassen sich kaum für Einzelentwicklungen zu vertretbaren Kosten entwickeln. Der Hardwarebereich erhält einen wesentlichen Impetus seiner Leistungsexplosion aus dieser Tatsache. Hier sollen nun einige Werkzeuge für den Entwurfsprozess kurz diskutiert werden. Dabei sollen Werkzeuge zur (ausführbaren) Modellierung der Entwürfe am Anfang stehen. Im Hardwarebereich ist es schon lange nicht mehr möglich, eine Implementierung "mal eben" vorzunehmen und mit ihr zu experimentieren. Man ist also gezwungen mit Simulationsmodellen zu arbeiten. Dieser Zwang hat auf der anderen Seite dazu geführt, daß im Hardwarebereich in der Regel von ausführbaren Spezifikationen ausgegangen wird. Die ersten Simulatoren beschränkten sich auf die elektrische Ebene und die Gatterebe. Daher gab es für diese Ebenen auch die ersten Modellierungssprachen, im Hardwarebereich Computer Hardware Description Languages (CHDLs) genannt. Schon auf diesen Abstraktionsebenen modelliert man weitgehend deklarativ, d.h. durch Aufzählung von Instanzen von Bibliothekselementen und Angabe ihrer Verschaltung. Dies wird auch dadurch erzwungen, daß die in der Realisierung benutzten vorgefertigten Komponenten sehr präzise Restriktionen für ihren Betrieb fordern und andererseits über die Zeitachse hinweg auch sehr fein wirken. Realitätstreue Modelle lassen sich daher nur dadurch erreichen, daß man für die Komponenten dedizierte sehr präzise Modelle in Bibliotheken ablegt. Etwas anders sieht die Situation im Fall einer reinen Spezifikation aus. Hier kann man sich darauf beschränken, das intendierte logische Verhalten in Form Boolescher Ausdrücke anzugeben. Dieses Verhalten wird dann durch implementierende Aktivitäten in eine Verschaltung von Bibliothekselementen transformiert. Das Ergebnis ist gleichzeitig ein Plan für den physikalischen Entwurf und ein präzises Verhaltensmodell. Auf der Registertransferebene hat es sich durchgesetzt, die Operationen auf einen Typ zu "ikonisieren", den Registertransfer. Hier geht man im Prinzip von der Existenz von nur zwei Typen parametrisierbarer Standardbauteile aus: dem Register als Datenspeicher und der beliebigen Transformationsfigur auf den Transferwegen. Register haben eine direkte Entsprechung in Hardwarekomponenten und auch die beliebig parametrisierbaren Transformationsfunktionen kann man sich als "Programmable Logic Arrays" (PLA) realisiert vorstellen. Auf der algorithmischen Ebene werden Hardwaresysteme in einer zu Software sehr ähnlichen Weise dargestellt. Die meisten Beschreibungssprachen hier sind daher auch an imperative Programmiersprachen wie PASCAL angelehnt. Zur Darstellung der Parallelität und der notwendigen Synchronisation mussten allerdings neuartige Konzepte erarbeitet werden. Petrinetze [PET77, REI85] haben sich hier für die operationale Semantik bewährt. Dieses Konzept wird im Falle von DACAPO III [DAC87] explizit benutzt. Die Systemebene wird derzeit nur sehr wenig durch CHDLs unterstützt. Es gibt jedoch eine Reihe von Ansätzen, beispielsweise ODICE [MUR89]. Diese Sprache verfolgt sehr konsequent einen objektorientierten Ansatz. In letzter Zeit gewinnen Breitbandsprachen im Bereich der CHDLs eine steigende Bedeutung. Als Beispiele sind hier DACAPO III, welches den Bereich von der Systemebene bis zur Schalterebene überdeckt, und VHDL [VHD87] für den Bereich von der Registertransferebene bis zur Schalterebene zu nennen. DACAPO III lehnt sich in der Notation und dem Modulkonzept an MODULA II an, VHDL an ADA. Alle CHDLs werden durch mehr oder weniger ausgefeilte Simulatoren komplettiert. Sie bilden das Laufzeitsystem für die Beschreibungssprachen. Die Schwierigkeit liegt darin, ein hochgradig nebenläufiges Realzeitmodell auf eine Zielarchitektur, die meist ein konventionellen v.Neumann-Rechner ist, abzubilden. Das Verhalten ist natürlich nur eine der zu modellierenden Sichten. Ebenso wichtig

ist die Struktursicht. Hier sind im Hardwareentwurf komfortable graphische Strukturediteren üblich ("schematic capture"). Sie erlauben eine hierarchische Darstellung der Hardwarestruktur, wobei wieder vorgefertigte Bibliothekselemente instantiiert, plaziert und verdrahtet werden. Diese Editoren sind in der Regel unabhängig von der Abstraktionsebene. Es gibt aber auch ebenenspezifische Editoren. ABL [HRT77] ist ein derartiges Beispiel für die Registertransferebene. Hardwareentwickler haben sich derart daran gewöhnt, in Bildern zu denken, daß eine große Nachfrage danach besteht, auch das Verhalten graphisch darzustellen. Ein sehr mächtiges Beispiel für einen derartigen Ansatz stellen die "Statecharts" [HRE87] dar. Editoren für die Geometriesicht sind natürlich von erheblich höherer Komplexität. Da man im Hardwareentwurf bestrebt ist, möglichst regelmäßige Strukturen zu erzeugen, werden generative Konzepte für Geometrieeditoren besonders wichtig. Als Beispiel kann hier HILL [LEM84] genannt werden. Derartige generative Ansätze werden auch in neueren Verhaltenssprachen wie ODICE oder MoDL [SMI86] benutzt. Für die verschiedenen Übergänge zwischen den Abstraktionsebenen wurden eine Reihe von Synthesewerkzeugen entwickelt. Ein relativ neues Gebiet ist dabei der Übergang von der algorithmischen Ebene zur Registertransfereben. Dieser Schritt wird meist "High Level Synthesis" genannt. Hier finden Methoden des modernen Compilerbaus weite Anwendung. Ein wesentlicher Unterschied besteht darin, daß im Compilerbau in der Regel von einer vorgegebenen Zielarchitektur ausgegangen wird, während man bei der "High Level Synthesis" diese Zielarchitektur erst erzeugt. Dieses Gebiet wird besonders aktiv in Deutschland verfolgt [JOG86, MAR85, PFA88, ROC87]. Die Elemente der Registertransferebene werden durch sogenannte Modulgeneratoren in Verschaltungen der jeweils angebotenen Bibliothekselemente auf der Gatterebene überführt. Dabei stehen für den Datenpfad meist passende Elemente zu Verfügung, die nur noch passend parametrisiert werden müssen. Für das Steuerwerk hat man verschiedene Optionen (Mikroprogrammierwerk, PLA-Automat, Automat in krauser Logik). Welche Option man wählt, ist von der jeweiligen Aufgabenstellung abhängig. In jedem Fall hat man einen transformierenden Automaten zu realisieren. Für dessen Erzeugung aus einer abstrakten Spezifikation stehen wieder mächtige Werkzeuge zur Verfügung, beispielsweise das LOG/IC-System [GRL79]. Die Überführung von Gatterschaltungen in ein geometrisches Layout wird von sogenannten "Silicon Compilern" geleistet. Hier werden verschiedenartige Ansätze verfolgt. Ein Ansatz für den "Full Custom"-Entwurfsstil ist es, das Strukturbild des Schalterebene mit Information über die relative Positionierung der Elemente und der Ebene der benutzten Verdrahtung anzureichern, womit man sogenannte "Stick Diagramme" erhält. Diese enthalten noch keine metrische Information. Fügt man diese hinzu, indem man aus Bibliotheken die notwendige Information erhält, so hat man ein zunächst noch wenig platzoptimales Layout erhalten. Dies kann man dann durch geeignete Algorithem (Kompaktoren) optimieren. Das HILL-System ist ein Beispiel für diesen Ansatz. Weiter verbreitet sind Standardzellen-, Makrozellen- und Gatearray-Ansätze. Diese Unterscheiden sich im Fertigungsprozess und in der Art wie Zellen geformt und plaziert werden können. Gemeinsam ist ihnen aber, daß es vordefinierte Bibliothekselemente gibt, die parametrisiert, dann plaziert und schließlich verdrahtet werden. Nur durch Silicon Compiling- Techniken ist es heute möglich, die inzwischen hochkomplexen VLSI-Bausteine in vertretbarer Zeit optimal zu entwerfen. Optimierungsaktivitäten wurden bereits implizit oder explizit genannt. Sie spielen im Hardwareentwurf, der sehr harte Restriktionen zu berücksichtigen hat, eine zentrale Rolle. Im Bereich der "High Le-

vel Synthesis" kann dabei auf Methoden des Compilerbaus zurückgegriffen werden. Alle dort benutzten Codeoptimierungsverfahren finden auch hier Anwendung. Ein weiteres wichtiges Optimierungsgebiet ist das Scheduling, d.h. die Anordnung von Operationen des zu implementierenden Algorithmus über die Zeitachse in einem parallelen System von Resourcen. Dies ist ein sehr aktuelles Gebiet der Forschung. Zur Optimierung auf der Registertransferebene und der Gatterebene stehen reichhaltige Werkzeuge zur Verfügung. Die Schaltwerktheorie als die zugrundeliegende Mathematik wurde bereits in der ersten Hälfte des Jahrhunderts weit entwickelt. Dies hatte zur Folge, daß sehr effiziete Algorithmen zur Zustandsreduktion von Automaten und zur logischen Minimierung kombinatorischer Schaltnetze entwickelt werden konnten. Diese Algorithmen sind entweder in die entsprechenden Synthesewerkzeuge integriert oder werden getrennt eingesetzt. Für den physikalischen Entwurf stehen ebenfalls mächtige Optimierungswerkzeuge zur Verfügung. Hier sind besonders Algorithmen zur Faltung von PLAs, zur Plazierung ("Floorplanning") und Verdrahtung zu nennen. Entwurfsfehler sind im Falle von Hardware erheblich gravierender als bei der Software, da eine nachträgliche Reparatur entweder sehr teuer (Leiterplatten) oder gar unmöglich (VLSI-Chip) ist. Den Verifikationsaktivitäten kommt daher enorme Bedeutung zu. Das Gebiet der formalen Verifikation wird wissenschaftlich seit Jahren intensiv bearbeitet. Zwar sind inzwischen schon vollständige Mikroprozessoren und Spezialchips als korrekt bewiesen worden, doch ist man von einer breiten Anwendung in der praxis noch weit entfernt. Erheblich weiter entwickelt sind Systeme zum Korrektheitsnachweis der Realzeitbedingungen, sogenannte "Timing Verifier". Sie erlauben es, die Simulation auf das rein logische Verhalten zu reduzieren, was zu einer enormen Effizienzsteigerung führt. Die Simulation schließlich ist heute immer noch das dominierende Hilfsmittel zur Verifikation von Hardwareentwürfen. Dabei muß man sich natürlich im klaren darüber sein, daß die Simulation selbst noch keine Verifikation ist, sondern nur eine Methode, die dazu nötigen Daten zu beschaffen. Durch intellegente Ergebnisauswerter und erweiterte "Assertion"-Mechanismen hofft man, auch für die eigentliche Verifikation auf Simulationsbasis geeignete Werkzeuge anbieten zu können.

Bei der Komplexität des Hardwareentwurfs und der Vielzahl an Werkzeugen ist eine einfache Aneinanderreihung von Werkzeugen nicht mehr praktikabel. An verschiedenen Stellen wird daher intensiv an integrierten Entwurfsumgebungen gearbeitet. Eine derartige Umgebung stellt eine leistungsfähige Datenhaltung, ein generatives Konzept für eine einheitliche Benutzeroberfläche und eine Steuerung des Entwurfsmanagements, einschließlich Versionsverwaltung zur Verfügung. Hier gibt es naheliegende Parallelen zu Softwareentwursumgebungen wie PCTE. Die dort benutzten Datenhaltungskonzepte sind für den Hardwareentwurf mit seiner sehr feinen Struktur und der Notwendigkeit der Verwaltung großer Mengen von Objekten feiner Granularität nur bedingt geeignet. Hardwareentwurfsumgebungen, wie sie beispielsweise im CADLAB entwickelt wurden [MWS88] bieten daher höher entwickelte Konzepte für diesen Bereich an. Wegen des viel mehr formalisierten Ablaufs des Entwurfsprozesses können auch erheblich mächtigere Managementkomponenten angeboten werden.

5 Testproblematik

Die Testproblematik im Hardwarebereich unterscheidet sich von der bei der Softwareentwicklung grundlegend. Im softwarebereich ist ein Entwurfsfehler wahrscheinlich, ein "Fertigungsfehler" jedoch weitgehend ausgeschlossen. Bei der Hardware sind beide Fehlerarten möglich. Die Sprachregelung im Hardwarebereich ist meist, daß man das Suchen von Entwurfsfehlern Entwurfsverifikation nennt, auch dann, wenn man im Sinne des Softwareentwurfs testet, also simuliert. Nur die Überprüfung nach Fertigungsfehlern wird Testen genannt. Dabei kommt man ebenso wie im Softwarebereich sehr schnell zum Ergebnis, daß das eigentliche Ziel, einen Nachweis der korrekten Funktion aus Komplexitätsgründen nicht führen kann. Im Hardwarefall hat man jedoch einen sehr mächtigen Ersatz dadurch geschaffen, daß man sehr aussagekräftige Fehlermodelle gefunden hat. Man testet nun nicht, ob der Prüfling funktional korrekt arbeitet, sondern, ob keiner der nach dem zugrundeliegenden Fehlermodell möglichen Defekte vorliegt. Als einfaches und dennoch mächtiges Fehlermodell hat sich beispielsweise das erwiesen, das als einigen möglichen Defekt Leitungen, die ständig "true" oder ständig "false" sind, kennt (Haftfehlermodell [RBS67]). Hierzu ist natürlich Kenntnis über die Struktur des Prüflings notwendig. Für das Haftfehlermodell gibt es leistungsfähige Werkzeuge, die Testmengen mit fast 100möglichen Defekte erzeugen. Weiterhin hat die Erfahrung eine weitgehende Koinzidenz zwischen korrekter Funktion und Abwesenheit von Haftfehlern gezeigt. Allerdings hängt die Testbarkeit und die Komplexität der Testmusterberechnung in großem Maße von der Struktur der zu testenden Objekte ab [BEN84]. Hier haben sich Regelsätze herauskristallisiert, deren Einhaltung eine geringe Testkomplexität sicherstellt [EIW77]. Für deren Überprüfung gibt es seinerseits leistungsfähige Werkzeuge, zum Teil unter Verwendung wissensbasierter Methoden [BID88]. Es wäre sicherlich von großem Interesse, zu untersuchen, ob dieser Qualitätsbegriff der Testbarkeit über den Fertigungstest von Hardware hinaus übertragbar ist.

6 Zusammenfassung

Der Hardwareentwurf wird, bedingt durch technische Randbedingungen hochgradig formalisiert und unter intensiver Benutzung vorgefertigter Komponenten durchgeführt. Diese intensive Verwendung vorgefertigter Bauteile zwingt nicht nur zur Einhaltung deren funktionaler Spzifikation, sondern auch der Bedienungsprotokolle. Dies hat zu einer frühzeitigen Normung dieser Protokolle geführt. Die weitgehende Standardisierung hatte zur Folge, daß mit vertretbarem hohen Entwicklungsaufwand umfangreiche Bibliotheken von zum Teil hochkomplexen Standardbausteinen aber auch eine Vielzahl von weit verbreiteten Entwicklungswerkzeugen entstanden sind. Die, technologisch bedingte, Aufgabe individueller Freiheitsgrade beim Entwurf haben in diesem Bereich zu einer unverhältnismäßig hohen Produktivitätssteigerung zu einer nahezu explosionsartigen Leistungssteigerung geführt. Dabei sollte aber nicht übersehen werden, daß die Komplexität auch hochkomplizierter Hardwaresysteme immer noch um Größenordnungen unter der von Mammutsoftwaresystemen liegt. Hier nimmt der Hardwareentwurf derzeit intensiv Anleihen aus dem Gebiet des Software-Engineering auf. Ein weitaus intensiverer Dialog sollte für beide Aspekte des Systementwurfs von großem Vorteil sein.

7 Literatur

[BEN84] **R.G. Bennets :**
Design for Testable Logic Circuits
Addison-Wesley, 1984

[BID88] **M. Bidjan-Irani :**
Regelbasierte Systeme zur Sicherstellung der Entwurfsqualität hochintegrierter Schaltungen und Systeme
Dissertation Univ. -GH Paderborn, FB 17, 1988

[DAC87] **- :**
DACAPO III System-User-Manual
DOSIS GmbH, Dortmund,

[DIJ75] **E.W. Dijkstra :**
Guarded Commands, Nondeterminancy, and Formal Deviration of Programs
Comm. ACM, 18,8, Aug. 1975

[EDI87] - : EDIF - Electronic Design Interchange Format Version 2 0 0.
Electronic Industries Association
Washington D.C., 1987

[EIW77] **E.B. Eichelberger, T.W. Williams :**
A Logic Design Structure for LSI-Testability
Proc. 14th Design Automation Conference, 1977, pp. 462-468

[GAJ87] **D.D. Gajski :**
The Structure of a Silicon Compiler
in: Proceedings of IEEE ICCD, 272-276, 1987

[GRL79] **W. Grass, H.-M. Lipp :**
LOGE - a highly effective system for logic design automation
AMC SIGDA Newsletter 9, No. 2, 1979

[GUH78] **J. Guttag, J.J. Horning :**
The Algebraic Specification of Abstract Data Types
Acta Informatica, 10, 1978

[HRE87] **D. Harel :**
Statecharts: A Visual Formalism for Complex Systems
Science of Computer Programming, 8, 1987

[HRT77] R. Hartenstein :
Fundamentals of Structurd Hardeware Design
North Holland, 1977

[JOG86] H. Joepen, M. Glesner :
Optimal Structing of Hirachical Controll-Pathes in a Silicon Compiler System
in: Proceedings ICCAD'86, Nov. 1986

[LEM84] T. Lengauer, K. Mehlhorn :
The HILL System : A Design Environment for the Hirachical Specification, Compaction, and Simulation of Integrated Circuit Layouts
in: Proceedings Conference on Advanced Research in VLSI, MIT, 1984

[MAR85] P. Marwedel :
Ein Software-System zur Synthese von Rechnerstrukturen und zur Erzeugung von Mikrocode
Habilitationsschrift, Univ. Kiel, 1985

[MUR89] W. Müller, F.J. Rammig :
ODICE: Object-Oriented Hardware Description in CAD Environment
in: Proceedings CHDL89, North Holland, 1989

[NWS86] J. Miller, C. White, G. Schulz, k. Gröning :
The Object-Oriented Integration Methodology of the Cadlab-Workstation
Cadlap Report 5/88, 1988

[PET77] J.L. Peterson :
Petri Nets
ACM Computing Surveys, 1977

[PFA88] P. Pfahler :
Übersetzermethoden zur automatischen Hardware-Synthese
Dissertation Univ. -GH Paderborn, 1988

[RAM89 F.J. Rammig :
Systematischer Entwurf digitaler Systeme
Teubener, 1989

[RBS67] J.P. Roth, W.G. Bouricius, P.R. Schneider :
Programmed Algorithms to Compute Tests to Detect and Distinguish between Failures in Logic Circuits
IEEE ToC, Vol. EC-16, No. 5, Oct. 1987, pp. 567-580

[REI85] W. Reisig :

Petri Nets: An Introduction
Springer, 1985

[ROC87] W. Rosenstiel, R. Camposano :
The Karlsruhe DASL Synthesis System
in: D. Borrione (Ed.): From HDL Description to Guranteed Correct Circuit Designs, North Holland, 1987

[SMI86] J. Smit et al. :
Definition of the Syntax and Semantics of the Modelling and Design Language MoDL
in: Dewilde (ed.): The Integrated Circuit Design Book
Delft University Press, Delft, 1986

[VHD] - :
IEEE Standard VHDL Language Reference Manual,
IEEE, iStd 1076 - 1987

[WHG85] W.M. Waite, G. Goos :
Compiler Construction
Springer, 1985