SYSTEMATISCHER ENTWURF EINES 32 BIT MIKROPROZESSORS ALS AUSBILDUNGSAUFGABE

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

Kurzfassung

In diesem Beitrag soll erläutert werden, wie der systematische Entwurf komplexer Rechnersysteme durch die Betreuung von aufeinander abgestimmten Diplomarbeiten und einer vorbereitenden Lehrveranstaltung gelehrt werden konnte. Als Fallbeispiel diente ein 32 Bit Mikroprozessor mit komplexem Instruktionssatz und aufwendiger interner Architektur. Zunächst wird ein projektorientiertes Ausbildungskonzept vorgestellt, wie es in Dortmund und in Paderborn praktiziert wird. Das konkrete Beispiel wird anhand der Aufgabenstellung und einer kurzen Skizzierung des erzielten Ergebnisses eingeführt. Danach wird auf die gewählte Entwurfsmethode und die benutzten Hilfsmittel eingegangen. Zum Schluß wird versucht, die Erfahrung kritisch zu würdigen.

1. Überlegungen zu einem Ausbildungskonzept für den Entwurf von Hardware Systemen

Es ist, sicherlich zu Recht, einmal gesagt worden, daß die Informatik als Wissenschaft nur Sinn macht, wenn sehr komplexe Systeme bearbeitet werden. Damit sollte aber auch der Entwurf von Hardware Systemen nach Möglichkeit anhand komplexer Systeme gelehrt werden. Es ergibt sich dadurch aber sofort das Dilemma, daß derartige Systeme den Rahmen von Lehrveranstaltungen sprengen, insbesondere den der m.E. wichtigsten, dem Anfertigen einer Diplomarbeit. Verzichtet man aber auf Beispiele einer bemerkenswerten Komplexität, so sind Entwurfsverfahren, die auch die höheren Abstraktionsebenen umfassen, kaum zu vermitteln, und die Notwendigkeit, komplizierte CAD-Hilfsmittel einzusetzen, kann von den Studenten nicht eingesehen werden.

Sinnvolle Lehre auf dem Gebiet des Hardware-Entwurfs muß die zu entwickelnde Hardware als eingebettetes System sehen, das im Zusammenwirken mit der Systemumgebung, gesteuert von Systemsoftware arbeitet. Die so abgeleitete Aufgabenstellung muß nun auf verschiedenen Abstraktionsebenen einer spezifischen Ausgestaltung zugeführt werden. Als Abstraktionsebenen sind zu nennen:

- Systemebene

  Hier wird das Gesamtsystem als System kooperierender "Prozessoren" (d.h. realisierter abstrakter Datentypen) in objektorientierter Sicht gesehen. Informationsformate und Codierungen liegen nicht oder nur zum Teil fest. Als Zeitmodell zieht man sich meist auf ein Kausalitätsschema zurück, falls nicht für Leistungsbewertungen metrische Schätzwerte benötigt werden.

- Algorithmische Ebene

  Jeder der "Prozessoren" der Systemebene muß mit einem Interpretationsalgorithmus seines Instruktionssatzes ausgestattet werden. Man beachte die imperative Sicht. Die Informationsformate und Codierungen sind oft festgelegt, wenn auch nicht gezwungenermaßen. Auch auf dieser Ebene ist, abgesehen von Leistungsanalysen, eine Kausalitätsstruktur als Zeitmodell ausreichend.

  Man beachte, daß man beim Entwurf von Hardware i.d.R. keine sequentiellen, sondern hochgradig nebenläufige Algorithmen beschreibt.

- Register Transfer Ebene

  Beim Übergang von der algorithmischen Ebene zur RT-Ebene findet eine ganz entscheidende Transformation statt: Die imperative Sicht der algorithmischen Ebene wird in die reaktive Sicht der RT-Ebene überführt. Die Informationsformate und Codierungen liegen fest, man arbeitet mit einem relativ eingeschränkten Satz von Bausteintypen. Als Zeitpunkt bedient man sich meist der Zählung von Takten mit einem oft groben Modell der Abläufe innerhalb eines Taktes.

- Gatter Ebene

  Auf der Gatter-Ebene ist die Separation zwischen Datenmanipulation und Steuerung nicht mehr sichtbar. Man hat ein uninterpretiertes System stimulierbarer (d.h. aus dem Gleichgewicht zu bringender) Boolescher Gleichung vorliegen. Auf dieser Ebene arbeitet man meist mit recht feinen Realzeitmodellen mit diskretem Zeitraster.

- Schalter Ebene

  Dies ist eine Abstraktionsebene, die sich mit der Transistorrealisierung von Gattern (Transistor als idealer Schalter gesehen) befaßt, aber auch mit Schaltungen, die keine direkte Entsprechung auf der Gatterebene haben. Bei grundsätzlich gleicher Modellsicht wie die Gatterebene zwingen die Bidirektionalität von Feldeffekt-Transistoren und die beschränkte Speicherfähigkeit isolierter Regionen zu erweiterten Wertebereichen der unterlegten Logik.

- Symbolische Layout Ebene

  Diese Ebene ist auf Verhaltenssicht mit der Schalterebene identisch. Aus Struktursicht wird zusätzlich die relative Anordnung der Elemente und die Dotierung ("Färbung") der Verbindungen festgelegt.

- Layout Ebene

  Die Layout-Ebene unterscheidet sich von der des symbolischen Layout dadurch, daß alle Elemente mit Maßen versehen sind. Da Beschreibungen auf dieser Ebene von der unterliegenden Technologie nicht zu trennen sind, ist sie aus Verhaltenssicht mit der elektrischen Ebene identisch.

- Elektrische Ebene

  Hier hat man es mit einem analogen System zu tun, das typischerweise durch ein System von Differentialgleichungen modelliert wird. Es liegt ein dynamisches System mit kontinuierlichen Wertebereichen über einer kontinuierlichen Zeitachse vor.

Eine Lehrveranstaltung, die die Chance bietet, daß die Studenten mit diesen unterschiedlichen Systemsichten konfrontiert werden, und auch einsehen, daß sie sinnvoll sind, stellt die "Projektgruppe" dar. Diese Lehrveranstaltung ist am FB Informatik der Universität Dortmund bindend vorgeschrieben und wird derzeit am FB Mathematik-Informatik der Universität-GH Paderborn in der Informatikausbildung probeweise eingesetzt. Die Idee ist, daß eine Gruppe von etwa 10-12 Studenten ein realistisches Projekt über ein volles Jahr hinweg vollständig bearbeiten. Dies beinhaltet das Erarbeiten der notwendigen theoretischen Grundlagen, die Konzeption der Problemlösung, deren Organisation und schließlich Durchführung, Test und Dokumentation. Wichtig dabei ist, daß den beteiligten Studenten die gesamte Projektverantwortung übertragen wird, der Veranstalter soll sich auf eine (oftmals doch lenkende) Beratertätigkeit zurückziehen.

Im vorliegenden Fall lautete die Aufgabe, einen Mikroprozessor für Realzeitan-

wendungen zu entwerfen, wobei der Entwurfsprozeß das Spektrum von Systemebene bis elektrische Ebene umfassen sollte.

Typischerweise sind die engagiertesten Teilnehmer einer Projektgruppe mit dem erzielten Ergebnis nicht zufrieden. Für sie ist die Projektgruppe der erste Durchlauf, anhand dessen Fehler man lernt, das Problem richtig zu lösen. Diese Studenten bearbeiten dann meist aus der Projektgruppe kommende Themen im Rahmen ihrer Diplomarbeiten. Im hier beschriebenen Fall kam eine Gruppe von sechs Studenten zu dem Schluß, daß der gesamte Prozessorentwurf neu zu überdenken ist. Nun übersteigt der Entwurf eines komplexen Prozessors sicherlich den Rahmen einer einzelnen Diplomarbeit bei weitem. Damit bot sich das Konzept aufeinander abgestimmter Diplomarbeiten an. Üblicherweise würde man entweder auf eine streng horizontale Gliederung verfallen (jeder Kandidat bearbeitet eine spezielle Abstraktionsebene, eine spezielle Phase des Entwurfs) oder auf eine streng vertikale (jeder Kandidat bearbeitet eine spezielle Komponente des Prozessors). Die erste Alternative verbot sich allein aus zeitlichen Gründen. Zudem wäre sie didaktisch bedenklich (Widerspruch zum "long thin man"-Konzept). Ebenso didaktisch bedenklich erschien die zweite Alternative, da hier die Gesamtsicht verloren gegangen wäre.

Als Lösung bot sich schließlich ein "Puzzle-Ansatz" an. Hier bearbeiten verschiedene Studenten auf den verschiedenen Abstraktionsebenen jeweils verschiedene Teilkomponenten. Mir erscheint dieser Ansatz unabhängig von der Situation "Diplomarbeit" sehr tragfähig zu sein, da das Gesamtteam sehr widerstandsfähig gegenüber externen Störungen wird (Personalfluktuation) und eine generelle Identifikation mit dem Gesamtsystem ermöglicht wird. Weiterhin zwingt diese Aufgabenverteilung zu einer permanenten präzisen Dokumentation (nachlässiges Nachdokumentieren ist unmöglich!) und zu einer konsequenten Abkapselungsstrategie der Teilmoduln gegenüber der Umwelt. Dadurch werden außergewöhnlich robuste Entwürfe ermöglicht, die zudem lokale Modifikationen einfach zulassen. Bei der Erläuterung des Ausbildungskonzepts sollte darauf hingewiesen werden, daß die Teilnehmer von Projektgruppen mit nicht unerheblichen Vorkenntnissen in diese Lehrveranstaltung gehen. Sowohl in Dortmund wie auch in Paderborn wird ein abgestimmtes System von Vorlesungen und Seminaren als Vorbereitung angeboten /EIS 85/.

2. Aufgabenstellung im vorliegenden Fall

Ursprüngliche Aufgabe der Projektgruppe war es, für eine an der Universität Dortmund entwickelte Realzeit-Programmiersprache (CAP/RTL, /PS 83/ einen dedizierten Prozessor zu entwickeln. Dieser Prozessor sollte durch einen komplexen angepaßten Instruktionssatz mit aufwendigen Adressierungsarten und einem extrem leistungsfähigen Interrupt-Konzept diese Sprache optimal unterstützen.

Im Rahmen der Diplomarbeit wurde von dieser Bindung an eine Sprache abgegangen. Ziel war nun, einen Prozessor zu entwickeln, der blockstrukturierte höhere Programmiersprachen unterstützen sollte. Dabei waren Prozedurkonzepte (PASCAL, /JW 78/, C, /KR 78/), Blockkonzepte (ALGOL, /NA 63/, SIMULA, /BE 78/) und Klassenkonzepte (SIMULA, /BE 78/, MAINSAIL, /XI 85/) zu unterstützen. Typische Datenstrukturen wie array, record aber auch Text-Strings sollten einfach verarbeitet werden. Neben der Verarbeitung des Objektcodes sollte jedoch auch dessen Erzeugung durch den Prozessor unterstützt werden. Der Compiler sollte von harten Aufgaben (z.B. Register-Zuweisung) nach Möglichkeit entlastet werden. Für beide Aspekte sollte noch ein einfaches Konzept für Kontext-Wechsel bereitgestellt werden, insbesondere um rekursive Strukturen zu unterstützen. Frühzeitig fand die Entscheidung für eine reine 32-Bit Lösung statt. Damit sollte nicht nur modernen Trends Rechnung getragen werden, sondern auch eine homogene interne

Struktur des Prozessors bei hinreichend großem Adressraum möglich werden. Die externe Schnittstelle sollte mit der Prozessorschnittstelle eines handelsüblichen Prozessors kompatibel sein, um zu gewährleisten, daß der Prozessor in vorgegebene Umgebungen eingebettet werden kann. Die Entscheidung fiel recht zufällig auf die auf 32 Bit erweiterte Schnittstelle des Z8000.

Da der Entwurf von der externen Architektur bis zum Erstellen der Layouts für den CMOS-Prozeß der Universität Dortmund durchzuführen war, sollte bei jeder Entwurfsentscheidung der Realisierungsaspekt als VLSI-Chip berücksichtigt werden. So stand in der Aufgabenstellung bereits fest, daß der Datenpfad in konsequenter Bitslice/Functionslice-Technik /SA 84/ in einem Zweibussystem zu realisieren war. Auch beim Entwurf der Steuerungen sollte auf höchstmögliche Regularität geachtet werden. Bei der relativen Unerfahrenheit der Entwerfer sollte Robustheit ein wichtiges Entwurfskriterium darstellen. Dieser Aspekt wurde aufgrund der Aufteilung der Teilaufgaben (s.o.) automatisch verfolgt.

3. Kurze Schilderung des Ergebnisses

Es ist nicht Ziel dieses Papiers, das Ergebnis der Diplomarbeiten darzustellen. Dies ist ausführlich in den Diplomarbeiten /AHLAP 85a/ und in /AHLAP 85b/ nachzulesen. Es soll hier nur kurz umrissen werden, um zu demonstrieren, welche bemerkenswertes Ergebnis sich mit den hier beschriebenen Prinzipien erzielen läßt.

3.1 Externe Architektur des Prozessors AHLAP

Die Abkürzung AHLAP steht für "Advanced High Level Languages Assisting Prozessor". Es handelt sich um eine CISC (Complex Instruction Set Computer) Architektur mit etwa 150 Instruktionen. Nach strengem Orthogonalitätsprinzip werden für alle Instruktionen alle Adressierungsarten angeboten. Es handelt sich in der Regel um 3-Adress-Instruktionen, wobei für jeden Operanden die Adressierungsart frei gewählt werden kann. Aus Codeeffizienzgründen wird auch ein kurzes Format mit nur 2 Adressen und eingeschränkten Adressierungsarten angeboten. Auch dieses Konzept steht orthogonal zum Instruktionscode.

Die Adressmodi sind so gewählt, daß typische Datenstrukturen höherer Programmiersprachen wie array oder record mit unterstützt werden. Es gibt sogar eine Stackpointer- oder Datapointer-relative indirekte Adressierung speziell für array of records. Bei den Instruktionen sind neben einem reichen Satz konventioneller Instruktionen besonders solche hervorzuheben, die der Textverarbeitung dienen, den Kontextwechsel vereinfachen und solche für die Interprozeßkommunikation. Als Interrupt-Konzept wurde der übliche Ansatz über Interrupt-Vektoren gewählt.

Bei der Festlegung der externen Architektur wurde davon ausgegangen, daß moderne Codegeneratoren, insbesondere nach dem Graham/Glanville-Verfahren /GL 77/ sehr wohl in der Lage sind, einen komplexen Instruktionssatz auszunutzen. Auf der anderen Seite ist die Registerzuteilung ein hartes Problem und ist es besonders auch für ein "bottom up"-Verfahren wie im Falle Glanville. Daher wurde der Vorschlag von Ditzel und McLellen /DL 82/ aufgegriffen und anstelle einer Registerbank ein Stackcache vorgesehen. Es hält statisch den jeweiligen "top of stack". Die Größe dieses Stackcache ergab sich aus Platzgründen als 32 Worte à 32 Bit. Dieser relativ kleine Cache zeigte bei Testläufen eine mittlere Trefferrate von immerhin 89 %.

3.2 Die interne Architektur des AHLAP

Die interne Architektur des Prozessors AHLAP läßt sich charakterisieren als im Pipelining betriebenes System lose gekoppelter Subprozessoren, die über FIFO-Puffer kommunizieren. Die Subprozessoren ergeben sich aus dem Ablauf des Pipe-

lining: IFETCH, OPFETCH und EXEC. Dazu kommt ein I/O-Prozessor, der alle drei Pipelinestufen bedient und ebenfalls über FIFO-Puffer entkoppelt ist.

Dieses Grundkonzept für die interne Architektur hat eine Reihe von Vorzügen. Zunächst streut die Komplexität der Instruktionen wie auch die der Adressierungsarten sehr stark. Bei einer engeren Kopplung der Pipelinestufen wäre der Prozentsatz von Wartezyklen der einzelnen Subprozessoren signifikant gestiegen. Beim vorliegenden Ansatz haben Messungen ergeben, daß eine in etwa gleichmäßige Auslastung der Komponenten erreicht wird mit einer leichten Lastspitze bei der Komponente OPFETCH. Der entscheidende Vorteil dieses Entwurfsprinzips ist jedoch die weitgehende Isolierung der einzelnen Pipelinestufen. Nach Festlegung eines relativ einfachen Protokolls zur Bedienung der Puffer konnten die einzelnen Komponenten weitgehend autonom entwickelt werden. Dies führte zu bemerkenswert einfachen Steuerungen, die zudem sehr wenig Interdependenzen zur Umwelt haben ("design for robustness"). Der Zugriff zur Außenwelt geschieht über eine weitere über Puffer lose gekoppelte Komponente, den I/O-Prozessor. Er arbeitet intern ebenfalls im Pipelining-Modus und hat für die drei Komponenten IFETCH, OPFETCH und EXEC je einen gesonderten Puffer. Dadurch wird die Konfliktwahrscheinlichkeit bzgl. Externzugriffen wesentlich vermindert (hierzu trägt natürlich auch die hohe Trefferrate des internen Stackcache bei). Ein weiterer Vorteil dieses isolierten I/O-Prozessors ist, daß die Bedienung einer anderen Prozessorschnittstelle (z.B. MC68020) durch lokale Modifikationen möglich wäre.

Die zu zahlenden Kosten für dieses Konzept sind ein erhöhter Synchronisationsaufwand. Synchronisation ist nötig im Fall gemeinsamer Ressourcen, im Falle von Datenkonflikten zwischen Makroinstruktionen und bei der Initialisierung der Pipeline (z.B. nach Sprüngen). Es zeigte sich jedoch, daß die gesamte Synchronisation mit einigen hundert Transistoren zu lösen war.

Konzeptionell liegen beim AHLAP drei Subprozessoren je mit Steuerung und Datenpfad vor. Da jedoch ein relativ häufiger Intermodul-Datenaustausch nötig ist, wurde ein gemeinsamer Datenpfad in einem Zweibus-System konzipiert. Beide Busse sind jedoch in je drei Abschnitte (die leicht versetzt zueinander liegen) teilbar. Somit ist es doch wieder möglich, jede Steuerung direkt mit "ihrem" Datenpfad zu koppeln und die Konfliktwahrscheinlichkeit auf den Bussen signifikant zu senken. Im Idealfall liegt logisch ein 6-Bus-System vor. Ein ähnliches Konzept ist beispielsweise beim MC68000 zu beobachten.

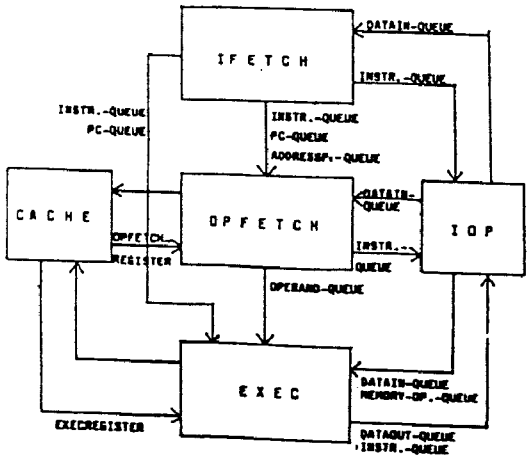


Abb. 1: Logische Struktur von AHLAP (aus /AHLAP 85a/)

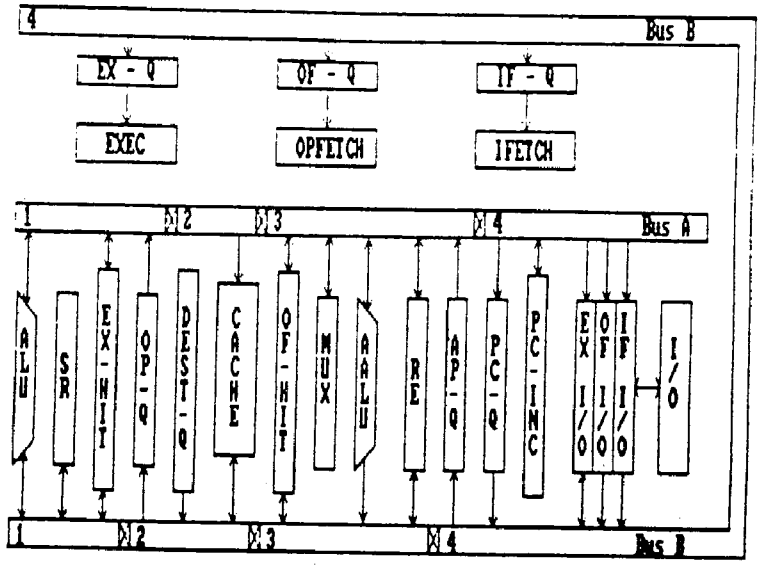


Abb. 2: Datenpfad von AHLAP (aus /AHLAP 85a/)

3.3 Die Realisierung des Prozessors AHLAP

Im Prinzip war ein Datenpfad und für jede Subkomponente eine Steuerung zu konzipieren. Alle Steuerwerke sind als konventionelle Mikroprogrammwerke aufgebaut, wobei der Ablauf natürlich wesentlich vom Zustand der Pufferspeicher und der Verfügbarkeit gemeinsam genutzter Ressourcen geprägt wird. Die Mikroprogrammwerke ihrerseits arbeiten im Pipelining-Modus.

Der Datenpfad wurde konsequent in Bitslice-/Functionslice-Technik entworfen. Der zeitliche Ablauf wird durch ein sehr einfaches Einphasenschema gegeben. Schaltungstechnisch wurde das Konzept "statische Speicher/dynamische Logik" verfolgt. Die Trenngates auf den Bussen dienen gleichzeitig als Treiber, so daß auch hier recht günstige Vorlade- und Evaluierungszeiten erreicht werden konnten. Da bei dem der Realisierung zugrundeliegenden Dortmunder CMOS-Prozeß nur eine Metallisierungsebene zur Verfügung steht, mußten auch für die quer zu den Bussen laufenden Steuerleitungen nach je 8 Bit Treiber vorgesehen werden. Es wurde ein hochgradig regelmäßiger Entwurf erhalten, bei dem die meisten Verbindungen durch "abutment" gezogen werden konnten. So weist der Floorplan auch nur wenig Fläche für Verdrahtung auf. Die gesamte Komplexität des Chips beträgt rund 100 000 Transistoren auf einer Fläche von etwa 11 mm x 11 mm.

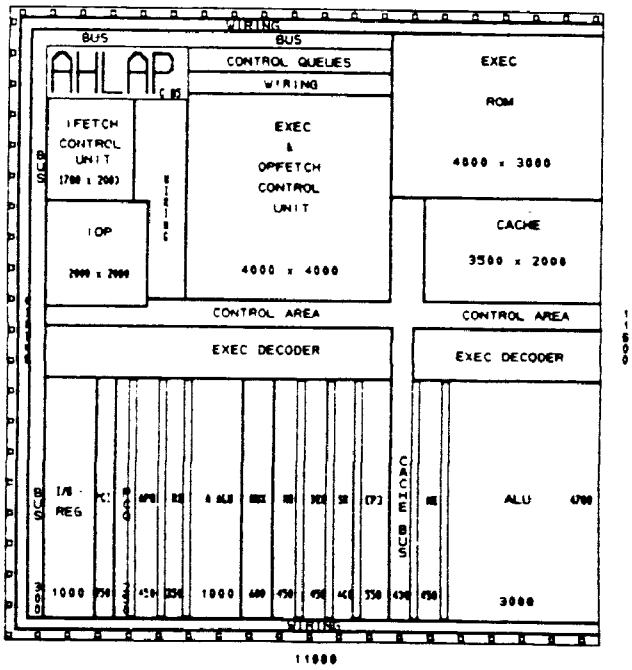


Abb. 3: Floorplan mit Größenangaben (aus /AHLAP 85a/)

4. Entwurfsmethode

Die von mir in der Lehre vertretene und im vorliegenden Fall angewandte Entwurfsmethode läßt sich als YOYO-Methode charakterisieren (iteriertes Top-Down und Bottom-Up mit Vorzugsrichtung Top-Down). Diese Methode erfordert eine Modellbildung auf verschiedenen Abstraktionsebenen. Zwischen diesen Modellen finden dann wohldefinierte Transformationen statt. Da stets eingebettete Systeme zu entwerfen sind, ist es notwendig, zunächst ein Grobmodell der externen Architektur anzufertigen. Anhand dieses Modells kann die Interaktion mit der Umwelt und das Verhalten unter der vorgesehenen Software untersucht werden. Sind diese Untersuchungen, evtl. nach Modifikationen, befriedigend abgelaufen, findet die Grobfestlegung der internen Architektur statt. Das Gesamtsystem wird in ein System interdependenter Agenten (objektorientierter Ansatz) aufgebrochen. Diese Transformation erfordert ein tiefes Verständnis in die zu erwartenden dynamischen Abläufe. Neben der Aufteilung ist auch die Kommunikationsstruktur festzulegen. Im vorliegenden Fall wurde zu diesem Zeitpunkt aufgrund eingehender Untersuchungen die Entscheidung getroffen, die Einzelmoduln weitgehend zu entkoppeln. Damit konnten im vorliegenden Fall die weiteren Entwurfsschritte auch relativ isoliert für die verschiedenen Komponenten durchgeführt werden.

Nach Festlegung der internen Grobarchitektur ist für jede Teilkomponente, die ja in jedem Fall als Prozessor im weiteren Sinn aufgefaßt werden kann, der Interpretationsalgorithmus für ihren Instruktionssatz anzugeben. Zu diesem Zeitpunkt wird die Komplexität der später zu realisierenden Steuerwerke und der Umfang des Datenpfades im wesentlichen vorgeprägt. Selbstverständlich kann sich in dieser Phase zeigen, daß die vorgelegte Modularisierung ungünstig ist, was zur Interaktion des vorhergehenden Entwurfsschritts führt. Dieses Problem tauchte im vorliegenden Fall wegen der "kanonischen" Modularisierung nicht auf.

Ein weiterer zentraler Entwurfsschritt ist durch den Übergang von der algorithmischen Ebene zur RT-Ebene gegeben. Hier ist ein imperatives Modell in ein reaktives zu übersetzen. Dazu sind zunächst Steuerung und Datenpfad, die im Algorithmus verflochten sind, zu separieren. Durch Datenflußanalyse ist festzustellen, wie die logischen Operationswerke und Transportkanäle auf physikalische Objekte abzubilden sind. Da durch diese Abb. in der Regel Konflikte eingeführt werden, muß das aus der Kontrollstruktur des Algorithmus abgeleitete Steuerwerk um die Auflösung dieser Konflikte erweitert werden. Modelliert wird das Ergebnis in reaktiver Sicht, d.h. als Geflecht von Objekten, die gewisse Operationen immer dann ausführen, wenn bestimmte Ereignisse stattfinden (Rückkehr zur zwischenzeitlich aufgegebenen objektorientierten Sicht). Selbstverständlich wird man in der Praxis (und so auch im vorliegenden Fall) zwischen algorithmischer Ebene und RT-Ebene öfters iterieren müssen.

Die Gatter-Ebene und die Schalter-Ebene sollte man im Falle von MOS-Entwürfen als Einheit sehen. Nach der Festlegung bestimmter globaler Regeln (im vorliegenden Fall die Entscheidung für dynamische Logik aber statische Speicher, die Entscheidung, nahezu alle kombinatorischen Schaltkreise als dynamische PLAs zu realisieren, Festlegung der Flipflop-Typen) ist die Transformation von der RT-Ebene auf die Gatter-/Schalter-Ebene relativ einfach, erfordert aber den intensiven Einsatz von Optimierungsalgorithmen. Auch hier wird man in der Regel öfter iterieren müssen.

Die Überführung in die elektrische Ebene ist wieder eine Transformation, die viel Know-How und Fingerspitzengefühl erfordert, falls effiziente Lösungen erwarten werden. Im Falle von CMOS-Schaltungen ist das Problem der Dimensionierungen zwar nicht ganz so groß wie im Fall von reiner "ratio"-Logik, doch ergeben sich auch hier im Falle dynamischer Logik nichttriviale Probleme.

Der so skizzierte Verhaltensentwurf wird von einem Strukturentwurf begleitet. Auf den höheren Abstraktionsebenen findet ein logisches Plazieren und Entflechten statt, das durch ein frühes Floorplanning (relatives Plazieren) begleitet werden sollte. Ist man auf der Schalterebene angelangt, kann man für die Einzelzellen (Blattzellen) symbolische Layouts angeben. Durch Kompaktieren nach den geltenden globalen Entwurfsregeln und Kontextbedingungen hat man die Information, wie der Pitch für Daten- und Steuerleistungen gewählt werden kann. Unter Zugrundelegung dieses Pitch kann das endgültige Layout der Zellen und Makrozellen angefertigt werden. Eine globale Verdrahtung sollte bei einem gut durchgeführten Entwurf von geringer Bedeutung sein.

Im beschriebenen Fallbeispiel wurde in Ermangelung eines geeigneten Hilfsmittels das symbolische Layout übersprungen und der Pitch durch Handlayout geeigneter Zellen festgelegt. Für diesen Pitch wurden dann alle Zellen entworfen.

5. Benutzte CAD-Werkzeuge

Der beschriebene Entwurf wurde mit bemerkenswert wenig Werkzeugen durchgeführt. Zur Verfügung standen die Werkzeuge CAP/DSDL, DOMOS und RUDI.
CAP/DSDL ist eine Breitband-Hardware-Beschreibungssprache mit Mixed Level Simulator /RA 81/. Das System wurde in Zusammenarbeit zwischen der Universität Dortmund und der Fa. Siemens AG unter Leitung des Autors entworfen und implementiert. Unter dem Namen DACAPO wurde eine portable Neufassung der Software, nun insbesondere unter UNIX ablauffähig an der Universität Dortmund implementiert. Gleichzeitig fand bei Siemens eine Integration von CAP in das Simulationssystem SMILE /FR 84/ statt.

CAP/DSDL überdeckt den Bereich Systemebene bis Schalterebene und erlaubt dabei beliebige Mischungen von Beschreibungen und Simulationen auf verschiedenen Abstraktionsebenen. Durch ein Modularisierungskonzept, das auch abstrakte Datentypen umfaßt, wird der oben beschriebene Entwurfsstil unterstützt. CAP kennt imperative und reaktive Sprachmittel, so daß alle Ebenen adäquat beschrieben werden können. Soweit möglich ist die Sprache an PASCAL angelehnt, so daß sie mit relativ geringem Trainingsaufwand erlernbar ist. Um einen Konzeptdinosaurier zu vermeiden, dienen interpretierte zeitbehaftete Petri-Netze als einheitliches semantisches Modell. Wo nötig kann das Zeitverhalten in hoher Präzision (rise-/fall-Zeiten, Unsicherheitsintervalle) angegeben werden. Die Benutzung von CAP erlaubte im beschriebenen Fall das Anfertigen und Austesten von hierarchischen Modellen auf verschiedenen Abstraktionsebenen.

DOMOS ist ein an der Universität Dortmund entwickelter Simulator für die elektrische Ebene, speziell für MOS-Schaltungen /SI 82/. DOMOS liefert für diesen Anwendungsfall exakte Ergebnisse bei relativ geringen Rechenzeiten. Für den Dortmunder CMOS-Prozeß standen alle Parameter zur Verfügung, so daß präzise Ergebnisse erzielt werden konnten. Es wurden alle kritischen Schaltungen und alle Blattzellen mit DOMOS simuliert und gegebenenfalls aufgrund der Ergebnisse modifiziert (meist bzgl. der Dimensionierung).

RUDI ist ein Layouteditor, der an der Universität Dortmund nach dem Vorbild von CESAR der UCB entwickelt und implementiert wurde. Ein derartiges Hilfsmittel ist natürlich auf die Dauer unbefriedigend. Es stellt sich aber heraus, daß sich beim Verfolgen streng hierarchischer Entwurfsprinzipien auch mit einem derart einfachen Hilfsmittel komplexe Entwürfe durchführen lassen.

Als besonders schmerzlich wurde das Fehlen von Optimierungspaketen (z.B. LOGE /GL 79/ bzw. ESPRESSO /RU 83/), das Fehlen eines TPG und eines in den Editor integrierten DRC/ERC empfunden.

6. Kritik

In der Summe kann das beschriebene Vorhaben als großer Erfolg gewertet werden. Dies wird besonders deutlich, wenn man sich vor Augen hält, daß es sich um das Ergebnis von 6 Diplomarbeiten handelt. Bedingt durch fehlende Hilfsmittel, aber auch knapper Zeit wurden jedoch wesentliche Entwurfsschritte übergangen. Das krasseste Manko ist sicherlich, daß der Testbarkeitsaspekt nicht berücksichtigt wurde. Vor einer eventuellen Fertigung ist hier mit Sicherheit eine Überarbeitung notwendig. Ein weiteres Manko ist, daß die Simulationsmodelle nicht hinreichend mit Last versehen werden konnten. Erforderlich wäre ein Betriebssystemkern und Compiler für gängige Sprachen gewesen. Letztere hätten sich nach dem Glanville-Verfahren relativ einfach generieren lassen. Dennoch wäre damit endgültig der Rahmen von Diplomarbeiten gesprengt worden.

Bei all diesen Unzulänglichkeiten wurde m.E. jedoch demonstriert, daß bei Verfolgung einer strikten Entwurfssstrategie von einer relativ kleinen Mannschaft in relativ kurzer Zeit ein hochkomplexer Baustein sauber entworfen und dokumentiert werden kann.

7. Literatur

/AHLAP 85a/ Kleinjohann, B./Kowalski, P./Kupitz, E./Preuß, W./Reinhardt, M./Strauß, J.: Entwurf eines 32-Bit Mikroprozessors (6 Diplomarbeiten, hier unter Sammeltitel aufgeführt). Universität Dortmund, FB Informatik, 1985

/AHLAP 85b/ Kleinjohann, B./Kowalski, P./Kupitz, E./Preuß, W./Rammig, F./Reinhardt, M./Strauß, J.: AHLAP - An Advanced Highlevel Languages Assisting Processor. Euromicro '85, Bruxelles 1985

/BE 78/ Belsnes, O.: The Use of SIMULA for Real-Time System Implementation. Norwegian Computing Center, 1978

/DL 82/ Ditzel, D./Mc.Lellen, H.: Register Allocation for Free: The C Machine Stach Cache. Bell Labs, MIT, 1982

/EIS 85/ Projekt E.I.S., Workshop "Lehre". GMD 1985 (nur für projektinternen Gebrauch)

/GEF 84/ Gonauser, M./Egger, F./Frantz, D.: SMILE - A Multilevel Simulation System. ICCD '84, 1984

/GL 77/ Glanville, R.: A machine independent algorithm for code generation and its use in retargetable compilers. University Microfilms, 1977

/GL 79/ Grass, W./Lipp, H.: LOGE - A Highly Effective System for Logic Design Automation. SIGDA Newsletter, Vol. 9, 1979

/JW 78/ Jensen, K./Wirth, N.: Pascal User Manual and Report. Springer 1978

/KR 78/ Kernighan, B./Ritchie, D.M.: The C Programming Language. Addison Wesley 1978

/NA 63/ Naur, P. (ed.): Revised Report on the Algorithmic Language ALGOL 60. CACM 6 (1963)

/PS 83/ Poppsieker, J./Schulz, P.: CAP/RTL: A Universal Microprocessor Software Development System. Microprocessing & Microprogramming 11/1983

/RA 81/ Rammig, F.: CAP/DSDL Preliminary Language Reference Manual. Forschungsbericht 129, Universität Dortmund, Abt. Informatik, 1981

/RU 83/ Rudell, R.: ESPRESSO - IIc Manual. CAD Toolbox User's Manual, U.C. Berkeley

/SA 84/ Sandweg, G.: A Highly Regular Peripheral Processor. NATO Advanced Study Institute on Microarchitecture of VLSI Computers. Sogesta, Urbino, 1984

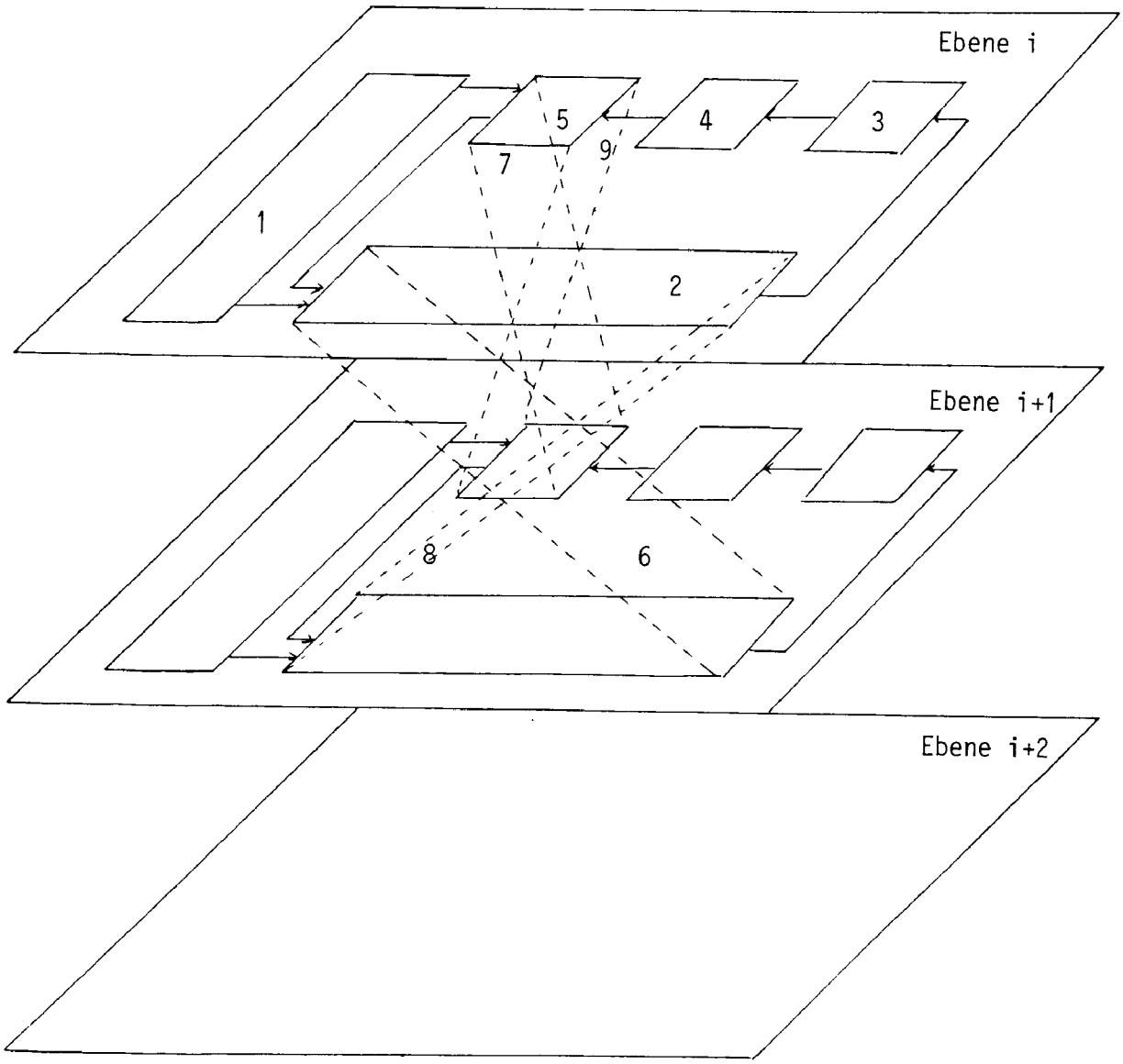
/SI 82/ Sibbert, H.: DOMOS, A nonlinear transient simulation and optimization program for integrated circuits. Universität Dortmund, Abt. Elektrotechnik, 1982

/XI 85/ Xidak Corp., Menlo Park, CA: Mainsail Language Manual, 1985

Anhang 1: Beispiel für eine Beschreibung auf Systemebene: Abstrakter Datentyp Queue in CAP/DSDL, erlaubte operationen in Exportliste. (Aus /AHLAP85a/)

```
type queue =
   export (enq, deq, read, full, empty, reset) procedure queue;

      const n = (*Breite*);
      const m = (*Laenge*);

      var speicher          : array [0:m-1] of bit(n);
          first, last       : integer;
          fu, em            : bit;

      procedure enq ( in data : bit(n) ) ;
         seqbegin
            speicher[last]  := data;
            last            := (last + 1) mod m ;
            fu              := (first = last);
            em              := 0
         end;

      function deq : bit(n) ;
         seqbegin
            deq             := speicher[first];
            first           := (first +1) mod m;
            em              := (first = last);
            fu              := 0
         end;

      function read : bit(n) ;
         seqbegin read      := speicher[last]  end;

      function full : bit;
         seqbegin full      := fu end;

      function empty : bit ;
         seqbegin empty     := em end;

      procedure reset ;
         conbegin
            first, last     := 0;
            fu              := 0;
            em              := 1
         end;

      end;
```

Anhang 2 : Yoyo - Entwurfsprozess



1 : Spezifikation auf Ebene i
2 : Intraebenen - Generierung (z. B. Optimierung)
3 : Bewertung
4 : Modifikationsentscheidung
5 : Modifikationssteuerung
6 : Top Down Interebenen - Generierung (z.B. Implementation)
7 : Top Down Interebenen - Modifikationssteuerung
8 : Bottom Up Interebenen - Generierung (z.B. Komposition)
9 : Bottom Up Interebenen - Modifikationssteuerung