Strukturierter, hierarchischer Entwurf

am Beispiel des µP-Kernels FHOP

O. Feißt, D. Jansen

ASIC - Design - Center, Fachhochschule Offenburg

email: o.feisst@fh-offenburg.de

Juli 95

An der Fachhochschule Offenburg wurde im Sept. 93 das Projekt eines implantierbaren 16 Bit Mikroprozessor - Kernels FHOP ins Leben gerufen. Ausgehend von dem in einem Testchip erfolgreich erprobten unstrukturierten Entwurf wurde durch gezielten Einsatz von strukturiertem Routen unter Nutzung der Fähigkeiten zum hierarchischen Arbeiten in der MENTOR - IC - Station eine erheblich verkleinerte und flächenmäßig optimierte Struktur abgeleitet, die sich mit 4 Quadratmillimetern Fläche durchaus mit kommerziellen Mikroprozessor - Kerneln vergleichen läßt.

Das FHOP-Projekt

Mit der Verfügbarkeit von High-Level Synthesetools und der Hardwarebeschreibungs-sprache VHDL besteht heute die Möglichkeit, auch im Bereich der Lehre und Forschung Projekte anzugehen, die früher große Teams erfordert hätten. So wird an der FH - Offenburg das Projekt FHOP (steht für "First Homemade Operational Processor") ins Leben gerufen mit dem Ziel,

- den Studenten Einblicke in den Aufbau und die Funktion von Mikroprozessoren zugeben,
- die neuen Werkzeuge wie VHDL und Synthese an einem anspruchsvollen Beispiel zu erproben,
- für den Einsatz in komplexen ASICs über einen eigenen lizenzfreien Mikroprozessorkern zu verfügen.

Das Ergebnis dieser Arbeiten ist ein Mikroprozessor - Kernel mit 16 bit Architektur, über dessen Funktion in [9] ausführlich berichtet wird und der in einem Testchip erfolgreich erprobt wird.

Bei diesem ersten Testchip steht nur die Funktion im Vordergund. Der komplett in VHDL beschriebene FHOP wird deshalb ohne hierarchische Strukturierung und ohne Rücksicht auf Flächenverbrauch plaziert und geroutet. Der Testchip umfaßt außer dem Kernel noch ein 256-Byte RAM und ein Peripherie-Modul (parallele Ein-/Ausgabe-Einheit). Der Kernel ist als flaches Standardzellen-Design (Microcode-ROM als Block) realisiert (Abb. 1) und weist eine Chipfläche von 13.2 mm^2 (davon Microcode-ROM 1.4 mm^2) auf. Er ist aus 20810 Transistoren aufgebaut. Die verwendete Technologie ist der ES2 - 1.0 µm - ECPD10 CMOS - Prozeß.

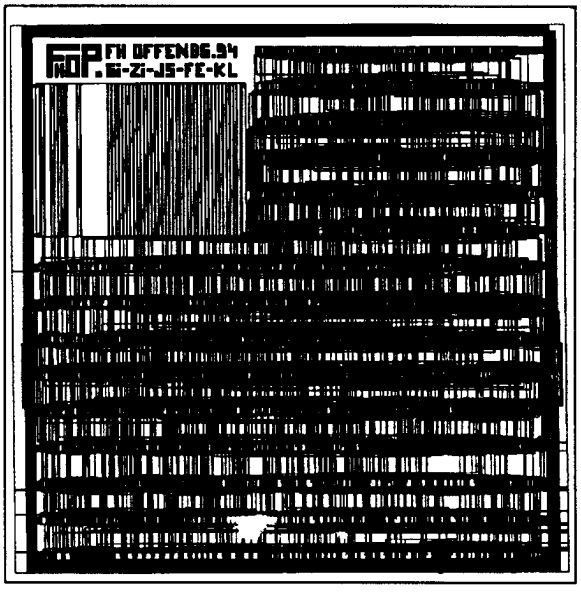

Abb. 1: Der Kernel FHOP V1.0 als Block

Der Chip wurde mit EuroChip Run #114 (Sept. 94) gefertigt und Ende November 94 geliefert. Er ist, abgesehen von einem kleinen Problem beim peripheren PIO-Modul, voll funktionfähig, was sich auch in der ersten Demonstrations-Applikation (Taschenrechner) bestätigt.

Entwurf einer flächenmäßig optimierten FHOP - Struktur

Die Untersuchung des Flächenverbrauchs der ersten FHOP - Version zeigt zwei Bereiche auf, die Ansatzpunkte für eine Optimierung bieten

- die sehr aufwendige ALU
- der Registersatz.

Weiterhin steht mit der neuen 0.7 um Technologie ein flächenmäßig verkleinerter Prozeß zur Verfügung. Die beiden Maßnahmen werden im folgenden ausführlich erläutert.

Optimierung der ALU

Da das VHDL-Modell der Original-ALU eine reine Verhaltensbeschreibung darstellt, kann man nicht mit einem optimalen Syntheseergebnis rechnen. Die auf ES2 optimierte ALU umfaßt folgerichtig auch 37 Sheets mit 890 Instances (bzw. 6400 Transistoren).

Um dieses Ergebnis zu verbessern, wird von Hand eine eigene ALU mit den gleichen Eigenschaften auf Schematic Capture Ebene entworfen und simuliert. Das Resultat ist mit 227 Instances (3132 Transistoren) auf 5 Sheets deutlich kleiner als die erste Version. Bei einer durchschnittlichen Flächenbelegung von 1600 Transistoren/mm^2 bei der ES2 1.0µ-Bibliothek ergibt sich somit ein Flächengewinn von ca. **2 mm^2**.

Migration von 1.0µ auf 0.7µ

Die Auslieferung der EuroChip CDROM Libraries V2.02 im März 95 ermöglicht uns den Zugriff auf die 0.7µ-Bibliothek von ES2.

Die Umstellung erweist sich als problemlos, da die ES2-Bibliotheken fast identisch aufgebaut sind. Auch die Datenorganisation sowie die Systemumgebungs- und Location Map-Variablen (Mentor-spezifisch) sind einheitlich. Es ist somit möglich, die auf ES2 1.0µ gemappten Schematics zugrunde zu legen, d.h. man kann sich einen zweiten kompletten Logiksynthese-Durchlauf sparen. Folgende Schritte sind beim Umstieg durchzuführen:

- Neugeneration aller verwendeten Megazellen
- Update der Schematics;

Der Update ist problemlos bis auf folgende Ausnahmen:

- LIBIECL2 hat keinen Ersatz in 0.7µ
- LIBADD2 wird ersetzt durch LIBFADD2
- geänderter Pfad zum Symbol hzpull_del bei Verwendung der Autologic-Bibliothek (.../hzpull_del ➢ .../LIBHZPULL/hzpull_del)

- Neugeneration **aller** Viewpoints

Reguläres Routing des Registersatzes

Als weiterer Verbesserungsansatz bietet sich an, die reguläre Struktur des aus sechs identischen 16-Bit-Registern bestehenden Registersatzes zu nutzen. Dazu ist es sinnvoll, zuerst ein 16-Bit-Register zu optimieren und als fertige Hardmakro-Zelle in die ES2-Library aufzunehmen. Anschließend baut man den Registersatz aus 6 einzelnen Registerzellen auf und nimmt schließlich den gesamten Registersatz als Block wiederum in die Library auf.

Um diesen Weg zu gehen, kann man schon auf Schematic Capture Ebene im **Design Architect** erste Vorbereitungen treffen:

- Vergabe der *"phy_comp"*-Property am Blocksymbol des 16-Bit-Registers und vorbereitend auch gleich am Blocksymbol des Registersatzes.

Die Symbole werden dadurch in der IC-Station als fertige Blöcke betrachtet und nicht weiter nach unten aufgelöst (Voraussetzung: man stellt beim *"logic loading"* die Option *"user defined"* ein).

- Vergabe von Preplace-Properties an den Instances des 16-Bit-Registers

Die relative Lage eines Instance im Floorplan kann schon hier definiert werden.

Es gibt vier Preplace-Properties (seed, group_seed, place, group_place), von denen die seed-Properties durch die automatischen Place&Route-Tools übergangen werden können, während die place-Properties unbedingt bindend sind. Eine seed-Property könnte z.B. den Wert *"T1:L3"* (entspricht: Top 1, Left 3) haben, was soviel bedeutet, daß dieses Instance in der ersten Floorplan-Reihe von oben an dritter Stelle von links plaziert werden soll. Diese Informationen werden beim Autofloorplan-Aufruf verarbeitet, so daß man ohne manuellen Eingriff sofort den Floorplan samt der plazierten Zellen erhält.

Die nächsten Möglichkeiten für entscheidende Eingriffe in den Design-Ablauf ergeben sich in der **IC-Station**:

Als wichtiger "Manipulationsknopf" erweist sich hier der *ecpd07-Prozess-* Beschreibungsfile der Firma ES2, der die Technologie, wichtige Design

Rules etc. beschreibt. Durch einige Änderungen kann dieser Prozeß für das Block-Routing optimiert werden und wird im Folgenden als *"ecpd07_block"* bezeichnet. Die wichtigsten Änderungen betreffen die Einstellungen der *"port_styles"*- und der *"power_styles"*-Variablen.

Durch die Definition der Power-Ports auf nur einem statt auf zwei Routing Levels (metal1 und metal2) mithilfe der *port_styles*-Variablen kann erreicht werden, daß evtl. nötige Vias (Durchkontaktierungen) auf der vollen Port-Länge plaziert werden. Führt man diese Änderung nicht durch, wird bei Bedarf nur ein einziges Via in der Portmitte gesetzt.

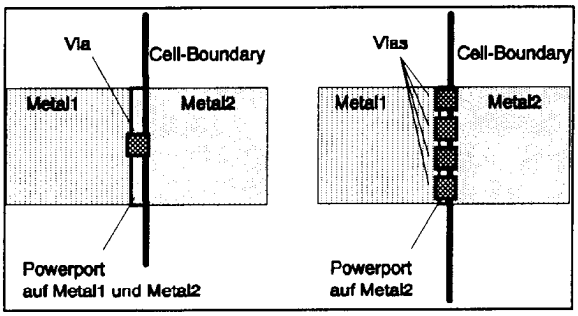

Abb. 2: Die Wirkung der port_styles-Variablen

Die zweite wichtige Änderung betrifft die *power_styles*-Variable und damit das Aussehen des Power-Routings. Mit der Zielrichtung minimales Power-Routing kann man hier bis auf die Stile *row_to_bus*, *internal_row* und *tie_down* alle Einstellungen entfernen.

Wenn man nun die neue Zelle, z.B. *16bitreg*, kreiert, sollte man beim Ausfüllen des entsprechenden Formulars auf folgende Einstellungen achten:

- Einstellung von ecpd07_block als Zellenprozess,
- Eintrag der verwendeten Preplace-Properties (z.B. seed) unter "logic properties to copy",
- Einstellung der "user_defined"-Option bei "logic loading" (s.o.).

Nachdem man den Floorplan mit den vorplazierten Zellen automatisch erzeugt hat, stößt man beim automatischen Plazieren der Ports auf ein weiteres Problem. Alle Anschlüsse, auch die breiteren Power-Ports, werden durch "autoplace ports" mit der Breite der Signal-Ports plaziert. Diese Breite kann aber mithilfe der *"net_comp"*-Property beeinflußt werden. Da es im Schaltplan jedoch kein physikalisches Äquivalent für die Block - Ports in der IC-Station gibt, kann man diese Property nicht schon auf Schaltplan-Ebene vergeben. Man muß die Ports erst in der IC-Station plazieren, die Power-Ports selektieren, die Property hinzufügen (wichtig: Value *"power_port"*, Owner *"port"*) und die Ports erneut plazieren lassen.
Durch diese Prozedur erscheinen die Power-Ports beim zweiten Plazierungsdurchlauf mit der korrekten Breite.

Bei der Bearbeitung der Strukturen ist weiter von Vorteil, daß komplette Ports dupliziert werden können, welche dann auch Mitglieder des gleichen Netzes sind.
Hierzu muß zuerst ein Bestandteil des Ports selektiert und anschließend der dazugehörige Port mit *activate port* für Editieroperationen (z.B. copy) vorbereitet werden. Ab diesem Zeitpunkt werden alle hinzugefügten Shapes/Paths automatisch Mitglieder des aktivierten Ports bzw. seines Netzes (normale Editieraktionen sind nur im *deactivate-status* sinnvoll). Zu beachten ist auch, daß bei einem erneuten *autoplace ports/pins* die Duplikationsinformation verloren geht, d.h. es wird wieder nur **ein** Port plaziert.

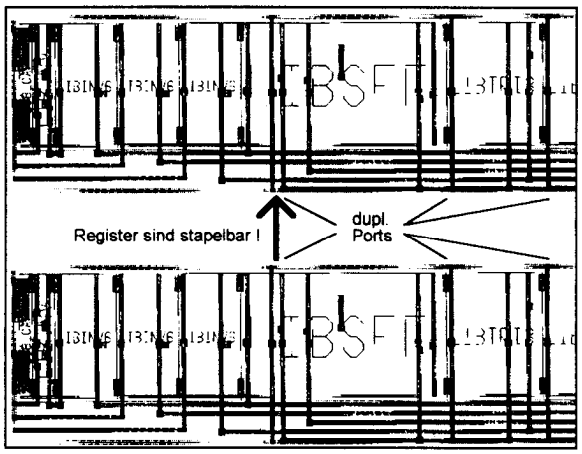

Abb. 3: Port-Duplikation ermöglicht ein Stapeln der 16-Bit-Register (abutting)

Wenn man diese Möglichkeit beim 16-Bit-Register nutzt, so daß in der Top- und der Bottom-Reihe des Blocks die Ports exakt an der gleichen Stelle sitzen, dann lassen sich die einzelnen Registerzellen ohne weiteren Routingaufwand zu einem Registerblock aufeinanderstapeln (s. Abb. 3). Es müssen dann im Registerblock nur noch die Enable-Leitungen an die einzelnen Register geführt und das Power-Routing vervollständigt werden.

Prinzipiell zeigt sich, daß das Arbeiten mit dem *autoroute overflow*-Kommando oft die besten Ergebnisse liefert (kürzeste Verbindungen), v.a. wenn man als Option *"non preferred routing length = 0"* einstellt.

Bei diesem Vorgehen müssen die Cap-Cells (Abschlußzellen einer Floorplan-Reihe) manuell plaziert werden.

In Abb. 4 sind zum Vergleich die verschiedenen Versionen des Registersatzes einander gegenübergestellt. Man erkennt, daß durch den strukturierten Entwurf ein beträchtlicher Flächengewinn erzielt wird, was besonders deutlich beim Vergleich der 1.0µ-Entwürfe (Version 1 und 2) auffällt. Dies liegt daran, daß der vollautomatische Entwurf (Abb 4 - 1) mit dem Original-Prozeß *ecpd10* erzeugt wurde und das Power-Routing somit in keinster Weise optimiert ist.

restliche Logik (ALU, Steuerung etc.) in der IC-Station interaktiv zu einem eigenen Block *(alu&rest)* aufbauen, um mehr Einflußmöglichkeiten auf das Aussehen des Gesamtchips zu haben.

Man wechselt dazu kurzzeitig in den *CE-Modus*, bricht die Hierarchie-Strukturen auf und baut sie neu auf. Dies ist mithilfe des *Floorplan Hierarchy Windows* und der *Flatten/Partition*-Kommandos sehr einfach möglich.

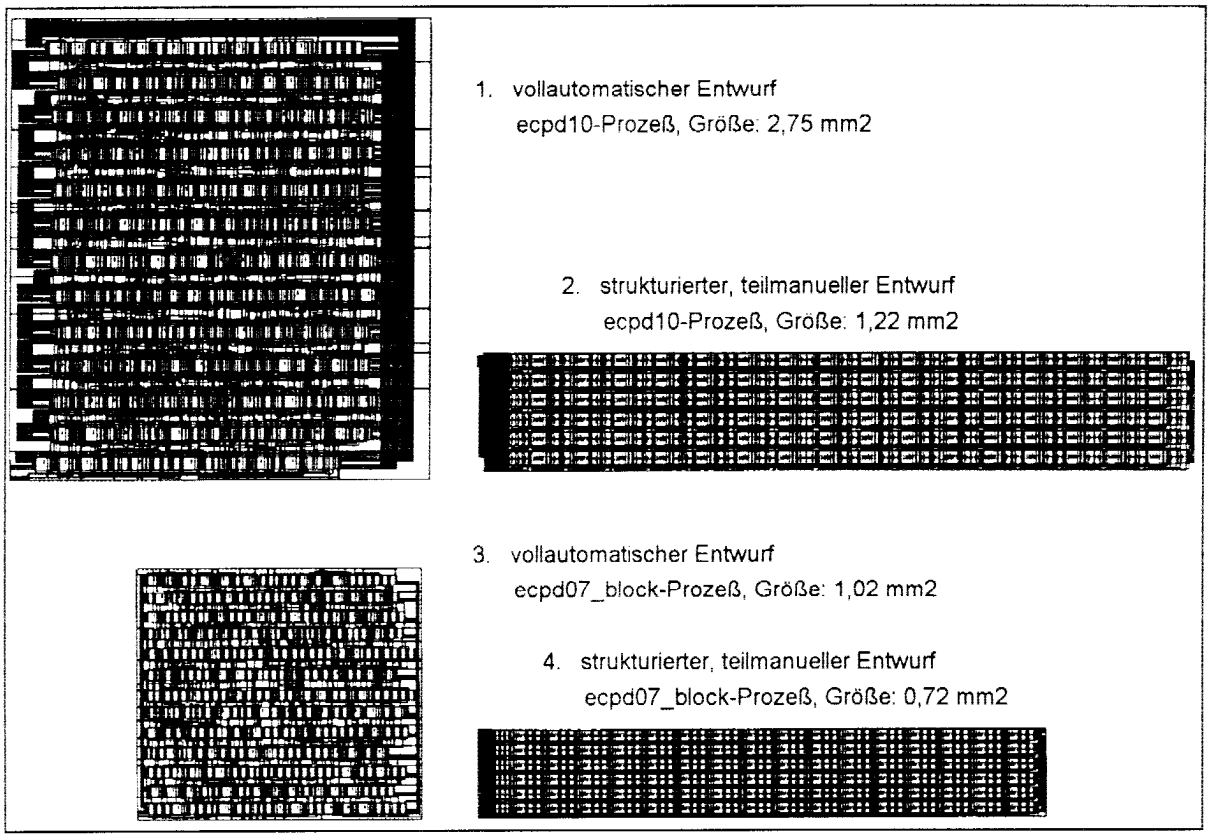

Abb. 4 Vergleich der verschiedenen Registersatz-Entwürfe

Der vollautomatische Entwurf (Abb 4-3) mit dem angepaßten Prozeß *ecpd07_block* wird demgegenüber schon recht klein, so daß der Gewinn durch strukturierten Entwurf (Abb 4-4) geringer ausfällt, aber immer noch ca. 30% beträgt.

Hierarchisches Routing des gesamten Kernels

Abschließend gilt es nun, den kompletten Kernel unter Verwendung der vorhandenen Blöcke hierarchisch neu aufzubauen.

Nachdem man die Zelle kreiert, den Floorplan automatisch erzeugt und die vorhandenen Blöcke (Microcode-Rom, Registersatz) nach den eigenen Wünschen plaziert hat (Abb. 5), kann man die

Nun führt man im Context des neuen Blocks *alu&rest* folgende Aktionen durch:

- *ecpd07_block* als Prozeß einstellen,
- *ES2-Library ecpd07* hinzubinden, *(attach library)*
- *autoplace std cells* mit Option "initial" (Zellen werden "vorläufig" plaziert),
- *autoplace ports; "net_comp"*-Property an Power-Ports,
- erneutes *autoplace ports*, (s. "Reguläres Routing des Registersatzes"),
- evtl. Ports duplizieren (z.B. Power, Clock).

Nachdem man so den Block *alu&rest* in seinem Aussehen grob definiert hat, kehrt man wieder in den Context der Topzelle zurück, um dort die Plazierung zu vervollständigen, d.h. die noch fehlenden Ports der Topzelle zu plazieren. Dabei kommt hier wieder das schon beschriebene Vorgehen mit der Vergabe der *"net_comp"*-Property (gewünschte Port-Breite über *"port_styles"*-Variable im Prozeß einstellen) und dem evtl. Duplizieren von Ports zur Anwendung.

Die Auswirkung der bis hierhin durchgeführten Aktionen (ohne Port-Duplikation) ist in Abb. 6 dargestellt. Man erkennt anhand der Overflows deutlich, daß die Port-/Pin-Plazierung bei Nutzung der Default-Einstellung ein ungünstiges Ergebnis liefert, was für den Autorouter einen erhöhten Verdrahtungsaufwand bedeutet.

Man hat nun aber die Möglichkeit, das Port-/Pin-Placement vom Topzellen-Context aus hierarchisch zu ändern, d.h. man kann sowohl die Lage der Topzellen-Ports, als auch die der Block-Pins beeinflußen. Dazu muß man sich in Erinnerung rufen, daß im Block *alu&rest* zum jetzigen Zeitpunkt nur die Standardzellen plaziert sind, aber noch keinerlei Routing existiert und somit auch die Pins uneingeschränkt bewegt werden können.

Der Designer hat also die Möglichkeit, von außen das Aussehen des Gesamtchips festzulegen. Zu beachten ist lediglich, daß Ports sowohl mit *autoplace* als auch mit *select & move_in_row* manipuliert werden können, während bei Pins nur das *autoplace*-Kommando greift. Hier gibt es nun das Problem, daß die Power-Ports der Blöcke normalerweise eine andere Breite haben als die Power-Ports der Topzelle (verschiedene *"power_port"*-Einstellungen in den Prozessen !). Ein *autoplace pins* vom Topzellen-Context aus würde somit die Power-Pins am Block mit der Breite der im Topzellen-Prozeß definierten breiteren Power-Ports erzeugen. Das bedeutet, daß man Power-Ports immer nur im zugehörigen Context automatisch plazieren sollte.

Das Ergebnis des so hierarchisch optimierten Port-/Pin-Placements mitsamt der duplizierten Ports (VCC und VSS 4-fach, Clock 3-fach) ist in Abb. 7 zu sehen.

In Abb. 8 ist zusätzlich auch noch die Verteilung der Overflows des Clock-Netzes hervorgehoben.

Man erkennt, daß die Overflows durch die Duplikation der Clock-Ports sowohl im Registersatz als auch im Block *alu&rest* günstig verteilt sind.

Nachdem so das äußere Aussehen definiert ist, kann man das Innere des Blocks *alu&rest* endgültig fertigstellen und begibt sich dazu wieder in den Block-Context.

Zuerst werden die Standardzellen unter Berücksichtigung des optimierten Port-Placements erneut plaziert (*autoplace std cells* mit Option *"resume"*) und anschließend wird der ganze Block verdrahtet.

Dabei sind folgende Einstellungen von Vorteil:

- **Definition kritischer Netze**, z.B. Clock, mit dem Kommandos *"change net priority"* (diese Netze werden vom Autorouter zuerst verarbeitet).
- **Port-Rows "unslideable" machen**; die relative Lage der Ports bleibt dadurch erhalten, die Rows werden aber als Ganzes verschoben.

Um die Verschiebung der Rows (vergl. Abb. 9) wieder auszugleichen und die äußere Verdrahtung fertigzustellen, kehrt man wieder in den Topzellen-Context zurück. Nachdem man dort das Block-Placement nochmals angepasst hat (Abb. 10), beginnt man mit dem abschließenden Routing.

Der in Abb. 11 dargestellte erste Versuch wird mit den Default-Einstellungen des Autorouters und mit *"internal ring"* ("vertical bus" etc. wurde entfernt) als *power_style* im Prozeß durchgeführt. Man sieht, daß der Autorouter für jeden äußeren Power-Port, bzw. für jedes Overflow, einen eigenen *internal ring* anlegt und außerdem die Blöcke etwas verschiebt.

Für den zweiten Versuch werden deshalb die duplizierten Power-Ports wieder entfernt, so daß nur **ein** Power-Paar übrig bleibt (Abb. 12). Außerdem werden die äußeren Port-Pows "unslideable" gemacht und beim Autorouter die Option *"fixed channel size"* eingestellt. Dadurch ist der Autorouter gezwungen, **alle** Plazierungen und Abstände beizubehalten (Abb. 13). Man sieht auch, daß sich die Anzahl der Power-Ringe deutlich verringert hat, nämlich auf ein Ringpaar für die äußeren Ports und ein Ringpaar für die Power an den Blöcken. Die Verbindung der beiden Ringsysteme wird über das am Block *alu&rest* duplizierte Power-Port-Paar hergestellt.

Um dieses Ergebnis noch zu verbessern, muß man manuell eingreifen. So zeigt Abb. 14, wie die Zelle aussieht, nachdem man eines der Ringpaare entfernt und den Rest optimiert hat.

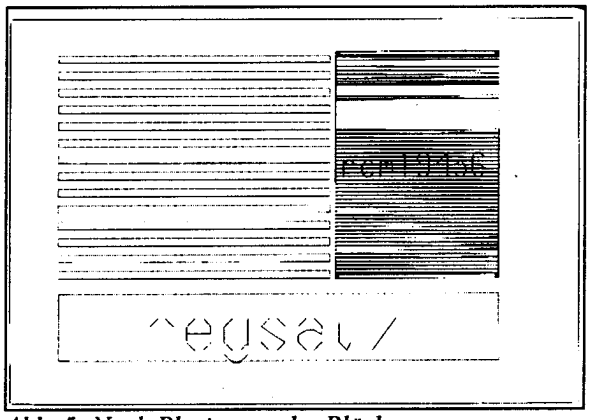

Abb. 5: Nach Plazierung der Blöcke

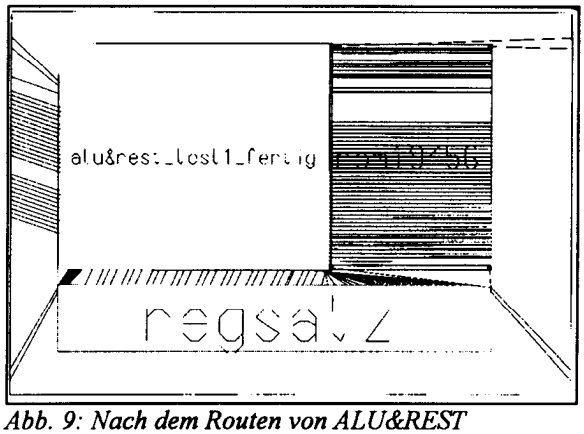

Abb. 9: Nach dem Routen von ALU&REST

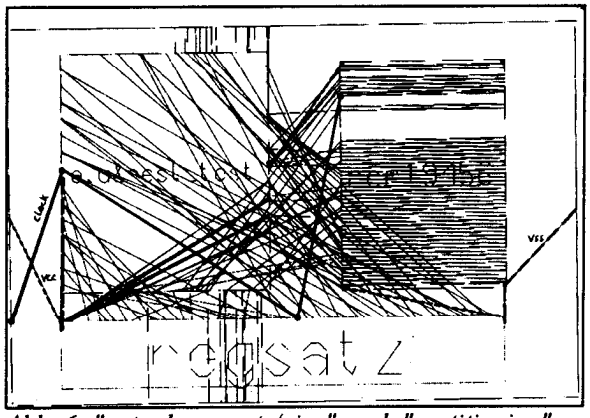

Abb. 6: "autoplace ports/pins" nach "partitioning"

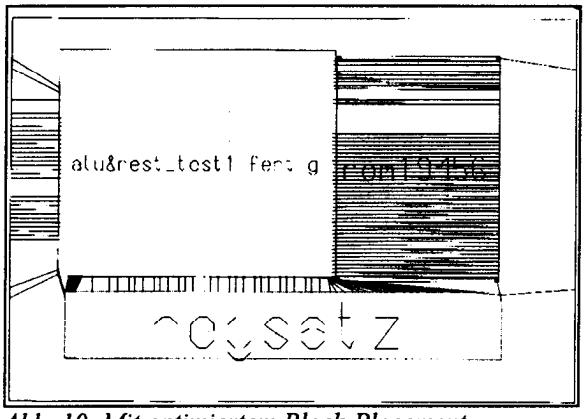

Abb. 10: Mit optimiertem Block-Placement

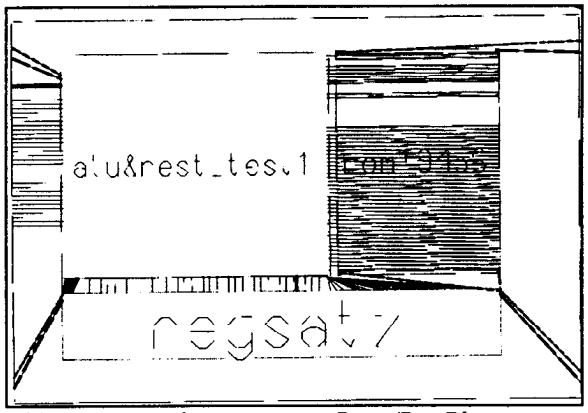

Abb. 7: Hierarch. optimiertes Port-/Pin-Placement

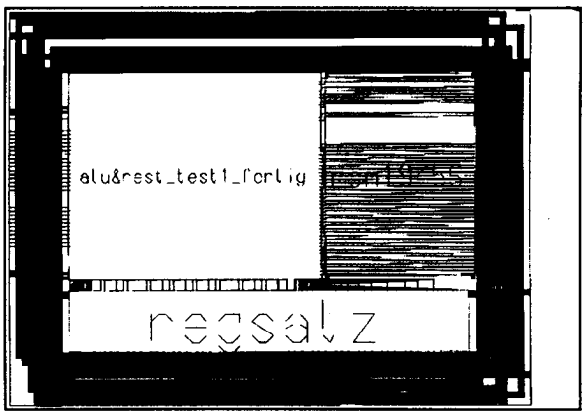

*Abb. 11: Mit einem "internal ring" **je** Power-Port*

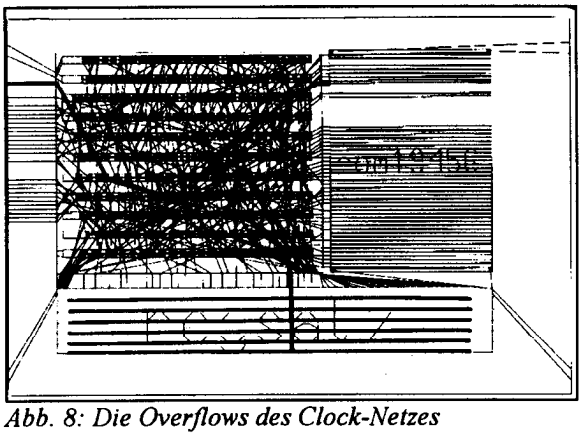

Abb. 8: Die Overflows des Clock-Netzes

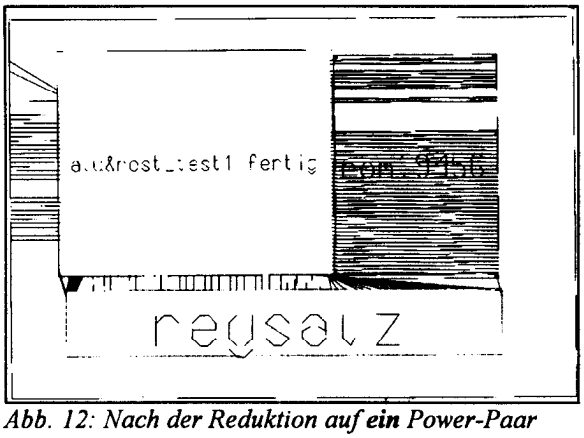

*Abb. 12: Nach der Reduktion auf **ein** Power-Paar*

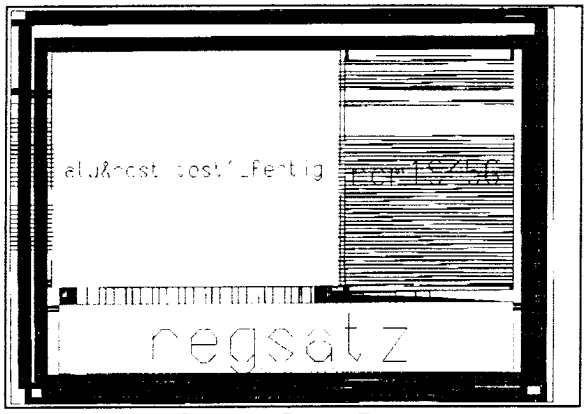

Abb. 13: Mit reduzierten Power-Ringen

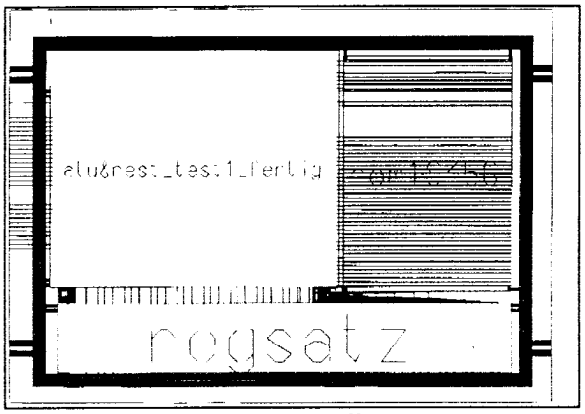

Abb. 14: Power-Routing manuell optimiert

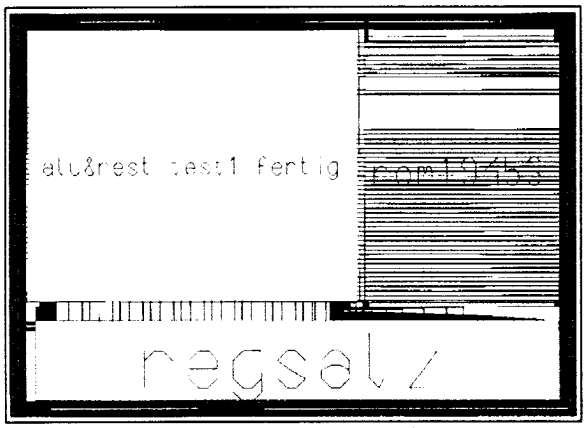

Abb. 15: Weiter optimiert und kompaktiert

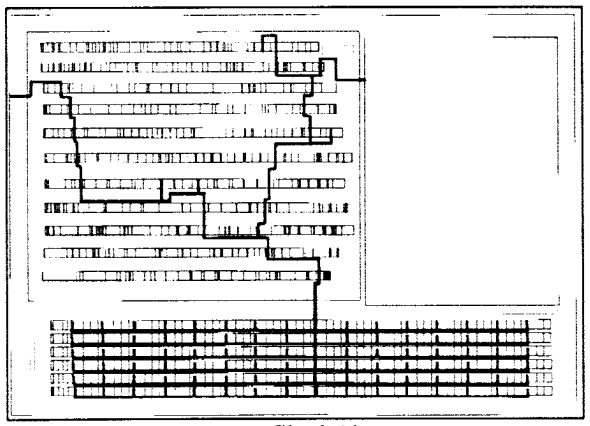

Abb. 16: Das geroutete Clock-Netz

Die endgültige Version ist in Abb. 15 zu sehen. Hier sind die äußeren Ports so nahe wie möglich mithilfe des *"compact"*-Kommandos an die Power-Ringe herangeschoben worden.

Als Letztes ist es noch interessant, das Routing der als kritisches Netz deklarierten Clock-Leitung zu betrachten. In Abb. 16 läßt sich erkennen, wie sehr sich der Autorouter an den Overflows des Netzes orientiert hat und dadurch ein recht zufriedenstellendes Ergebnis liefert.

Zusammenfassung

Der FHOP-Kernel wurde in mehreren Schritten soweit verbessert, daß die neue Version 1.1 nur noch 30 % des Flächenbedarfs der Version 1.0 aufweist. Dabei wurde der Flächengewinn in etwa gleich stark durch den Umstieg auf die kleinere 0.7µ-Technologie und den strukturierten Entwurf des Registersatzes erzielt.

Die Ersparnis durch Verwendung des 0.7µ-Prozesses beträgt nach Tabelle 1 etwa 40 %. Die etwas größere Zahl beim FHOP-Kernel ergibt sich durch die größere, synthetisierte Original-ALU und durch das relativ uneffektive Routing der 1.0µ-Version.

Das strukturierte Routing liefert eine Ersparnis von ca. 30 % (Tabelle 2). Auch hier liegt die Erklärung für den größeren Wert beim 1.0µ-Registersatz im uneffektiven Routing der unstrukturierten Version.

Einen Überblick über die Größenverhältnisse gibt Abb. 17. B4ei einer Fläche unter 4 Quadratmillimetern ist der FHOP-Mikroprozessor-Kernel V1.1 damit durchaus mit kommerziellen Prozessorkernen vergleichbar und kann in Zukunft effektiv in komplexen anwendungsspezifischen integrierten Schaltungen eingesetzt werden.

	1.0 µ	0.7 µ	Gewinn
Registersatz	1,22	0,72	41 %
19-kBit ROM	1,40	0,9	36 %
FHOP-Kernel	13,21	5,34	60 %

Tabelle 1: Flächengewinn (in mm^2) durch Umstieg von ES2 1.0µ auf ES2 0.7µ

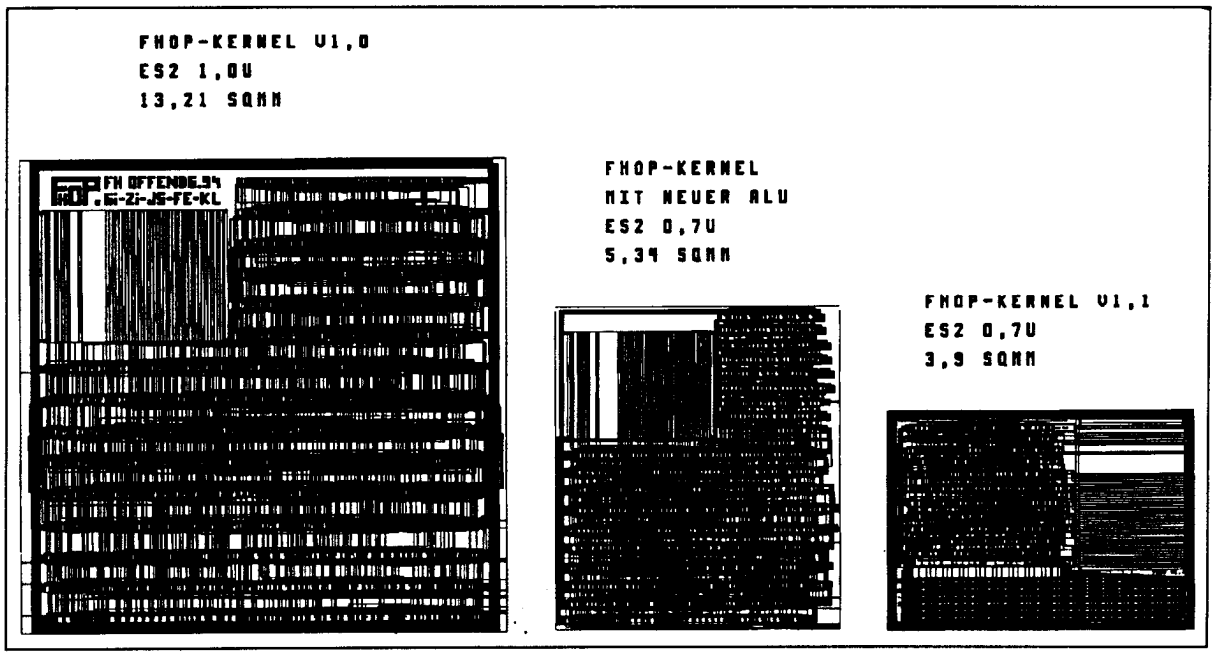

Abb. 17: Vergleich der verschiedenen FHOP-Entwürfe

	autoplace & -route	struktu-riert	Gewinn
Registersatz 1.0µ	2,75	1,22	56 %
Registersatz 0.7µ	1,02	0,72	29 %
FHOP-Kernel 0.7µ	5,34	3,9	27 %

Tabelle 2: Flächengewinn (in mm^2) durch strukturierten Entwurf

Literaturverzeichnis:

[1] Mentor Graphics, Software Version 8.4_1: "IC Station Automated Layout Procedures Manual" Juli 94

[2] Mentor Graphics, Software Version 8.4_1: "IC Station Technologie Definition Manual" Juli 94

[3] Mentor Graphics, Software Version 8.4_1: "IC Station Design Flow Manual" Juli 94

[4] Mentor Graphics, Software Version 8.4_1: "Properties Reference Manual" Juli 94

[5] Dantzer-Sorensen, M. , Andersen, A.C.: "Mentor Graphics V8.2 ES2 ecpd10 and ecpd07 IC Station User's Guide V1.1" EUROCHIP DTH, 14. Sept. 94

[6] European Silicon Structures: "ES2 Mentor V8 Design Kit Release Notes V3.0.0" 28. Okt. 94

[7] European Silicon Structures: "ES2 Mentor V8 Design Kit User Notes V3.0.0" 28. Okt. 94

[8] Klöser, F.: " Entwurf eines Ein-/Ausgabemoduls für den µP-Kernel FHOP mit VHDL und Integration aller FHOP-Komponenten auf einem Test-ASIC" Diplomarbeit an der FH Offenburg, Aug. 94

[9] Zimpfer, F. , Gieringer, Th. , Jansen, D.: "Entwicklung eines 16-Bit Mikroprozessor-Kernels mit Hilfe von VHDL" Bericht vom MPC-Workshop in Ulm, Jan. 94