FPGA-Design unter besonderer Berücksichtigung von Multiplexer-Struktur und Verdrahtbarkeit

Hans-Georg Martin Wolfgang Rosenstiel

WSI 99-17

15. September 1999

Wilhelm-Schickhard-Institut
Universität Tübingen
D-72076 Tübingen, Germany
email: martinh@informatik.uni-tuebingen.de

ISSN 0946-3852

FPGA-Design unter besonderer Berücksichtigung von Multiplexer-Struktur und Verdrahtbarkeit

Hans-Georg Martin, Wolfgang Rosenstiel

September 1999

Zusammenfassung

Beim Einsatz kommerzieller Synthese-Werkzeuge zur Abbildung einer High-Level-Schaltungsbeschreibung auf FPGAs zeigte sich, daß die Plazierung und Verdrahtung einen Engpaß darstellen und daß diese Entwurfsschritte viel zu langsam sind. Da die Verdrahtung und die Feststellung der Verdrahtbarkeit nach wie vor die größten Probleme bei kommerziellen Tools darstellen, wurden Lösungen gesucht und entwickelt, die im Rahmen einer technologieorientierten Synthese eine frühzeitige Berücksichtigung von Verdrahtungsaspekten ermöglichen. Es sollten insbesondere Verdrahtunsengpässe untersucht und beseitigt werden.

Im Ergebnis entstand ein Design-Fluß, der eine kürzere Entwurfszeit, eine bessere Verdrahtbarkeit, eine geringere Verlustleistung und außerdem kleinere FPGAs erlaubt. So konnte die Durchgängigkeit eines Design-Flusses, der auch kommerzielle Standard-Programm-Komponenten enthält, verbessert sowie die Laufzeiten der FPGA-Backend-Software verkürzt werden, indem das Verdrahtungs-Problem im Vorfeld bereits entscheidend entschärft werden konnte.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einführung

Beim Schaltungsentwurf ist durch die Komplexität der zu realisierenden Schaltungen der Einsatz rechentechnischer Hilfsmittel unerläßlich. Um von einer Verhaltensbeschreibung zum tatsächlichen Schaltkreis zu kommen, sind mehrere Entwurfsebenen zu durchlaufen: Dazu gehören vor allem die High-Level-Synthese, die Logiksynthese, die Technologie-Abbildung auf die Zielarchitektur, sowie die Layoutsynthese. Dafür gibt es verschiedene Werkzeuge, die für jeweils eine Ebene die Umsetzung der Entwurfsbeschreibungen und gleichzeitig eine Optimierung durchführen.

Wir haben im Rahmen der Arbeit zu diesem Report mehrere Design-Flüsse (s. Abb. 1) untersucht, die von einer Verhaltensbeschreibung aus eine FPGA-Realisierung durchführen. Neben der Möglichkeit, bei kleineren Schaltungen relativ schnell zu einer Hardware-Realisierung zu gelangen, zeigten sich dabei aber auch die Grenzen der vorhandenen Werkzeuge. Vor allem im Bereich der Verdrahtung wurde ein Engpaß sichtbar, der sich in langen Programmlaufzeiten bzw. in der Nicht-Verdrahtbarkeit einiger Entwürfe widerspiegelte. Deshalb wurden andere Design-Flüsse untersucht und bereitgestellt, die dieses Problem besser lösen (s. Abb. 9). Die erreichten Ergebnisse zeigen, daß die neuen Design-Flüsse geeignet sind, schneller zu einer günstigeren Hardware-Realisierung zu gelangen.

Der Verlustleistungs-Aspekt bei FPGAs ist neben den bisher betrachteten Kategorien Verdrahtbarkeit und Entwurfszeit ebenfalls ein wichtiger Gesichtspunkt. Auch wenn FPGAs nicht für batteriebetriebene Massenprodukte wie Mobiltelefone eingesetzt werden, so spielt die Verlustleistung trotzdem in einigen Anwendungen mit niedrigen Stückzahlen eine wichtige Rolle. In vielen mobilen Anwendungen z.B. der Automatisierungtechnik mit den bekanntermaßen kleinen Stückzahlen ist eine energieeffiziente Realisierung mit FPGAs erforderlich. Auch wenn die FPGAs auf Antifuse-Basis (z.B. von Actel) geringere interne Kapazitäten aufweisen, besitzen die SRAM-basierten FPGAs dennoch unter anderem die Vorteile der Reprogrammierbarkeit und besseren Testbarkeit [52]. Darüberhinaus gewinnen inzwischen speziell auch die XQR4000XL-FPGAs (wegen der inzwischen erreichten Strahlungs-Resistenz) für die Luft- und Raumfahrt Bedeutung - die Vorteile liegen dabei aufgrund der geringen Stückzahlen im geringen Stückpreis gegenüber ASICs, in der späten Programmierbarkeit sowie der Möglichkeit, das System auch im Einsatz weiterhin ändern zu können [49],[50],[51]. Eine verstärkte Betonung der Verlustleistungs-Aspekte kann den Einsatzbereich der FPGAs damit wesentlich erweitern.

Der Bericht ist im weiteren wie folgt gegliedert: Zuerst werden 5 existierende Design-Flüsse und ihre Ergebnisse vorgestellt (Kapitel 2 und Kapitel 3). Die Suche nach einem Ausweg aus dem Verdrahtungsproblem beginnt mit einer Beschreibung bisheriger FPGA-Abbildungen (Kapitel 4) und einer Betrachtung zur Verdrahtbarkeit bei FPGAs (Kapitel 5). Weiter werden verschiedene Multiplexer-Varianten untersucht, da sie das Verdrahtungsproblem maßgeblich beeinflussen (Kapitel 6). Daraus werden zwei Algorithmen abgeleitet, die eine bessere Verdrahtbarkeit gewährleisten und auch einen geringeren Leistungsverbrauch aufweisen (Kapitel 7). Diese führen zu drei neuen Design-Flüssen, deren Ergebnisse vorgestellt und bewertet werden. Eine Verallgemeinerung der Ansätze auf andere Entwurfsmethoden wird in Kapitel 8 vorgenommen. Hierfür kann auch die Klassifikation und Rückerkennung von Multiplexern aus Kapitel 9 weiterhelfen. Die Zusammenfassung erfolgt in Kapitel 10.

2. Verschiedene existierende Entwurfsstrategien

Im folgenden sollen zuerst einige Möglichkeiten vorgestellt und untersucht werden, wie z.B. ausgehend von einer Verhaltensbeschreibung eine FPGA-Schaltung erzeugt werden kann.

2.1. Fünf existierende Design-Flüsse zum FPGA

Im Rahmen der Arbeit zu diesem Report wurden 5 Design-Flüsse von der High-Level-Beschreibung zum FPGA in Bezug auf folgende Fragen untersucht:

- Sind die Entwürfe verdrahtbar ?
- Wie lange brauchen die Systeme für die Synthese bzw. bis zur Nachricht, daß die Synthese wegen Nichtverdrahtbarkeit unmöglich ist ?
- Wie wirkt sich insbesondere die Carry-Logik aus ?

Die Design-Flüsse 1-5 (Abb. 1) existieren bereits und dienen als Ausgangspunkt für die späteren Überlegungen. Sie werden durch folgende Merkmale charakterisiert:

`Design-Fluß 1:` Die High-Level-Synthese startet bei einer Verhaltensbeschreibung in der Sprache C und wird mit PMOSS [1] durchgeführt. Danach werden die Addierer und andere Funktionen durch schnelle Elemente der eingebauten Bibliotheken innerhalb eines kommerziellen Synthese-Systems realisiert (KSS). [Anmerkung: Die zur Logiksynthese eingesetzten Werkzeuge wurden den Berichterstattern zu besonders günstigen Bedingungen überlassen, die insbesondere jegliches Benchmarking in den abgeschlossenen Lizenzverträgen untersagen. Daher ist im folgenden anonym jeweils nur von mit KSS abgekürzten kommerziellen Synthesesystemen die Rede. Da hier die Logiksynthese nur als Hilfsmittel zur Generierung von Netzlisten für die FPGA-Abbildung eingesetzt wird, ist diese Anonymisierung problemlos möglich und schränkt die Verallgemeinerung der erzielten Ergebnisse in keiner Weise ein.] Nach der Netzlistengenerierung mit KSS und der kurz als „Mapping“ bezeichneten Phase der Technologie-Abbildung werden FPGA-Hersteller-spezifische Verdrahtungs- und Plazierungssysteme eingesetzt.

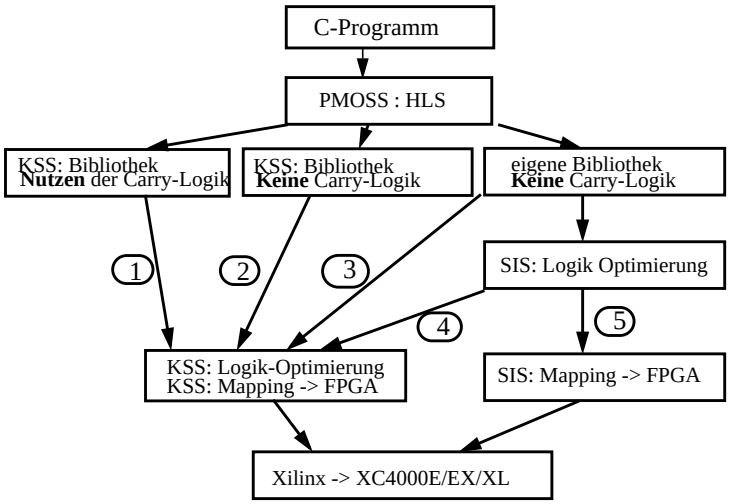

Abbildung 1. Fünf Design-Flüsse von der Verhaltensbeschreibung bis zum FPGA.

`Design-Fluß 2`: Der Unterschied zum Design-Fluß 1 liegt in der Verwendung von gewöhnlichen Bibliotheken des KSS, die nicht die schnelle Carry-Logik des FPGA ausnutzen.

`Design-Fluß 3:` Als Variation zum Design-Fluß 2 wurde hier eine eigene Bibliothek für die Operatoren eingesetzt.

`Design-Fluß 4:` Der Design-Fluß 4 entspricht Design-Fluß 3 bis auf einen zusätzlichen Logik-Optimierungsschritt mit dem Berkeley-Programm SIS [39], wobei “script.algebraic“ angewendet wurde.

`Design-Fluß 5`, benutzt - als ein Vergleich - SIS auch für das Mapping auf die LUT mit 4 Eingängen, wobei Collapsing, Cofactoring, und Binate Covering eingesetzt werden. Das SIS-Skript basiert auf [19] bzw. [39].

2.2. Entwurf mit anderen Entwurfssystemen - Entwurf mit CADDY

Als Alternative zu PMOSS, das in dem bisherigen Kapitel für die High-Level-Synthese verwendet wurde, kann auch CADDY [8],[9],[10] eingesetzt werden. CADDY untersucht mehr Design-Alternativen während des Schedulings und der Allokation bei der High-Level-Synthese als PMOSS, es erlaubt sehr komplexe Operator-Kombinationen und unterstützt auch Multiplexer-basierte Architekturen.

2.3. Manueller Entwurf auf RT-Ebene

Der Entwurf von Schaltungen wird oft auch auf RT-Ebene von Hand durchgeführt, um die Realisierung komplizierter Strukturen geeignet steuern zu können. Dabei hängt das Entwurfsergebnis stark von der zu realisierenden Schaltung sowie vom Geschick des Entwerfers ab. Dieser Weg wird oft für den Entwurf regulärer Strukturen oder für Controller genutzt. Der Weg zum FPGA führt auch hier über ein kommerzielles Synthesesystem und Xilinx.

3. Ergebnisse bisheriger Entwurfsstrategien

3.1. Ergebnisse der untersuchten 5 existierenden Design-Flüsse

Die Ergebnisse für die 5 Design-Flüsse sind in den Tabellen 1 und 2 enthalten. Die benutzten Beispiele sind 6 C-Programme (fibonacci-Zahlen, größter gemeinsamer Teiler, Ascii-to-Integer-Konvertierung, Elliptisches Wellen-Filter, Differentialgleichung sowie die diskrete Cosinus-Transformation nach Chen), die mit PMOSS für unterschiedliche Wortlängen synthetisiert wurde. Vom Hardware-Software-Codesign-Werkzeug PMOSS [1],[2],[3],[4],[5] wurde hier nur die High-Level-Synthese eingesetzt. Tabelle 1 enthält die Beispielnamen und ihre Wortlängen in Spalte 1, nach den I/O-Spalten folgen für jeden Design-Fluß das kleinste mögliche FPGA, die Zahl der erforderlichen CLB sowie die erzielte Verzögerungszeit. Die Rechenzeiten auf einer Sparc Ultra 2 (mit 2 Prozessoren, 640 MB Hauptspeicher und 296 MHz Taktfrequenz) sind in Tabelle 2 enthalten. Eine Aufstellung der Verlustleistungen findet sich in Tabelle 8.

3.2. Erreichte Schaltungs-Charakteristika : Fläche, Verzögerung und Rechenzeit

Design-Fluß 2 hat weniger Design-Probleme als Design-Fluß 1 und erlaubt kleinere FPGAs, dafür macht sich aber die Nicht-Benutzung der Carry-Logik bei der Verzögerungszeit bemerkbar. Erstaunlicherweise bringen die Makroblöcke der schnellen Addierer Verdrahtungsprobleme mit sich, d.h. die spaltenweise Anordnung der Makroblöcke bewirkt bei ihrer Anbindung Verdrahtungsprobleme. Diese Probleme wurden im Rahmen der Arbeit zu diesem Report gelöst (siehe Kapitel 7).

Die Design-Flüsse 3 und 4 vermögen für einige kleinere Beispiele die besten Lösungen zu erzeugen, für die großen Schaltungen spielen sie aber wegen der schlechten Ergebnisse keine Rolle.

Tabelle 1: Resultate für 5 Design-Flüsse von Abb. 1

			Des.-Fluß 1 KSS			Des.-Fluß 2 KSS			Des.-Fluß 3 KSS			Des.-Fluß 4 KSS			Des.-Fluß 5 SIS		
Schalt.	#in	#out	FPGA	CLB	ns	FPGA	CLB	ns	FPGA	CLB	ns	FPGA	CLB	ns	FPGA	CLB	ns
fibo04	6	5	4003E	***15***	25.7	4003E	***15***	25.1	4003E	16	***22.8***	4003E	16	28.3	4003E	16	32.2
fibo08	10	9	4003E	31	38.4	4003E	***25***	***34.3***	4003E	26	34.4	4003E	29	35.9	4003E	32	45.8
fibo16	10	17	4003E	43	***38.7***	4003E	***38***	54.1	4003E	***38***	57.7	4003E	44	62.3	4003E	49	60.4
fibo32	10	33	4003E	67	***36.7***	4003E	***61***	107.5	4003E	62	113.5	4003E	77	116.8	4003E	86	119.4
gcd04	10	5	4003E	23	41.5	4003E	20	45.0	4003E	***19***	42.5	4003E	***19***	46.1	4003E	24	43.1
gcd08	18	9	4003E	37	***58.7***	4003E	36	86.9	4003E	***32***	79.9	4003E	33	92.4	4003E	38	66.4
gcd16	34	17	4003E	64	65.5	4003E	66	127.2	4003E	***58***	150.4	4003E	62	146.6	4003E	68	132.6
gcd32	66	33	4005E	117	***82.9***	4005E	125	202.5	4005E	***111***	307.0	4005E	116	311.6	4005E	155	303.0
atoi08	17	16	4003E	56	***43.2***	4003E	***41***	62.4	4003E	43	75.4	4003E	44	69.1	4003E	47	71.2
atoi16	17	25	4003E	76	***38.0***	4003E	***61***	102.8	4003E	64	117.6	4003E	62	111.1	4003E	75	88.3
atoi32	17	41	4005E	116	***51.1***	4005E	***113***	201.6	4005E	116	216.2	4005E	114	209.5	4005E	123	218.0
ellip04	34	33	4005E	118	93.3	4005E	115	94.1	4005E	124	91.1	4005E	123	97.1	4005E	139	114.2
ellip08	66	65	4008E	191	130.9	4008E	195	148.9	4008E	205	136.6	4008E	217	152.5	4008E	231	140.9
ellip15	122	121	4025E	325	232.5	4025E	320	238.3	4025E	337	285.1	4025E	352	270.8	4025E	400	250.6
ellip16	130	129	4036EX	325	178.6	4036EX	341	226.9	4036EX	355	228.9	4036EX	396	262.1	4036EX	424	237.6
ellip23	186	185	4062XL	467	***175.4***	4062XL	481	233.6	4062XL	484	256.6	4062XL	536	222.3	4062XL	595	274.1
diffeq04	22	13	4003E	67	58.1	4003E	67	59.9	4003E	67	68.6	4003E	69	83.4	4003E	70	72.8
diffeq08	42	25	4005E	***133***	98.4	4005E	138	96.3	4005E	139	110.8	4005E	137	115.0	4005E	146	128.5
diffeq16	82	49	***4008E***	313	216.3	***4008E***	316	220.5	4010E	336	212.5	***4008E***	318	212.5	4010E	381	314.4
diffeq24	122	73	4025E	556	429.1	4025E	581	370.2	4025E	609	337.5	4025E	575	***320.9***	4025E	734	447.4
diffeq31	157	94	4028EX [a]	823	452.4	***4025E***	858	596.5	***4025E***	897	591.4	***4025E***	852	487.4	4036EX	1125	*421.1*
dct24	35	31	***4036EX***	1223	525.7	***4036EX***	1220	443.2	4044XL	1399	364.1	4044XL	1498	443.0	4044XL	1600	413.8
dct32	35	31	4062XL [b]	1910	526.9	***4052XL***	1889	812.9	4085XL [c]	2236	1424.6	4085XL [d]	2373	----	4085XL	2564	1557.1

a. Der nächst kleinere FPGA erlaubt kein Routing (Xilinx Nachricht nach 45:00 !!)
b. Der nächst kleinere FPGA erlaubt kein Routing (Xilinx Nachricht nach 9:56:00 !!)
c. Der nächst kleinere FPGA erlaubt kein Routing (Xilinx Nachricht nach 7:18:00 !!)
d. nicht verdrahtbar innerhalb von 4 Tagen (Xilinx Nachricht nach 97:20:00 !!)

Tabelle 2: CPU Programm-Laufzeiten (in Std:Min:Sec, auf einer Sparc Ultra 2, 640 MB Hauptspeicher) für die 5 Design-Flüsse von Tabelle 1

	Design-Fluß 1		Design-Fluß 2		Design-Fluß 3		Design-Fluß 4			Design-Fluß 5	
Schaltung	KSS	Xilinx	KSS	Xilinx	KSS	Xilinx	SIS	KSS	Xilinx	SIS	Xilinx
fibo04	1:12	0:21	1:13	***0:19***	1:11	0:20	0:01	1:08	0:20	***0:02***	***0:18***
fibo08	1:15	0:24	1:20	***0:21***	1:23	0:24	0:02	1:19	0:23	***0:06***	0:22
fibo16	1:22	0:29	1:26	0:25	1:43	***0:25***	0:03	1:35	0:29	***0:11***	0:27
fibo32	2:20	0:39	1:45	0:32	2:28	0:32	0:06	2:14	0:32	***0:23***	***0:31***
gcd04	1:14	0:24	1:15	0:22	1:15	***0:21***	0:01	1:10	***0:21***	***0:05***	0:22
gcd08	1:21	0:30	1:23	0:27	1:30	0:26	0:02	1:23	0:26	***0:12***	***0:25***
gcd16	1:40	0:39	1:42	0:33	2:18	***0:33***	0:04	1:57	0:35	***0:28***	0:33
gcd32	3:17	1:23	2:48	1:07	5:55	***1:05***	0:10	3:43	***1:05***	***0:40***	1:11
atoi08	1:37	0:31	1:33	0:29	1:46	0:27	0:03	1:31	***0:27***	***0:11***	0:28
atoi16	1:58	0:37	1:50	0:33	2:22	***0:32***	0:05	1:52	0:34	***0:23***	0:33
atoi32	2:22	1:08	3:03	***0:58***	3:53	1:09	0:09	2:48	1:00	***0:26***	1:00
ellip04	2:32	1:08	2:33	1:07	2:43	1:11	0:08	2:20	1:09	***0:24***	1:09
ellip08	3:28	*2:06*	3:32	2:10	4:36	2:19	0:14	4:06	2:07	***0:56***	2:10
ellip15	6:01	9:20	6:05	7:57	9:49	8:03	0:26	10:13	8:45	***1:41***	7:29
ellip16	6:08	4:14	6:06	4:13	10:18	***4:02***	0:27	10:04	4:46	***1:45***	4:36
ellip23	10:06	5:49	9:45	***5:00***	19:05	5:06	0:40	16:47	5:30	***2:58***	5:21
diffeq04	2:00	0:37	1:58	0:36	1:56	0:35	0:04	1:39	0:35	***0:23***	***0:34***
diffeq08	2:41	1:18	2:52	***1:07***	3:30	1:14	0:09	2:42	1:09	***0:37***	1:12
diffeq16	5:36	2:57	4:43	***2:25***	16:49	2:56	0:31	8:44	2:57	***2:01***	4:41
diffeq24	12:20	12:00	13:24	8:12	1:26:52	8:06	1:21	28:44	***6:31***	***4:29***	13:05
diffeq31	13:19	[a] ***7:44***	25:17	11:40	5:03:39	46:46	2:54	1:21:30	26:39	***8:07***	7:42
dct24	1:39:30	10:34	1:35:52	11:02	3:30:33	11:10	6:07	2:27:34	14:35	***14:23***	11:49
dct32	1:49:54	[b] 1:05:02	1:51:45	2:40:22	[c] 10:29:15	9:06:20	14:20	6:57:20	[d] 97:18:00	***48:32***	1:26:24

a. Für den nächst kleineren FPGA ist kein Routing möglich : (Xilinx Nachricht erst nach 45:00 !!)
b. Für den nächst kleineren FPGA ist kein Routing möglich : (Xilinx Nachricht erst nach 9:56:00 !!)
c. Für den nächst kleineren FPGA ist kein Routing möglich : (Xilinx Nachricht erst nach 7:18:00 !!)
d. Zeit bis zur Ausschrift „Design ist nicht verdrahtbar“

Der Design-Fluß 5 - der sich von den untersuchten Flüssen am meisten auf nichtkommerzielle Werkzeuge stützt - hier das Berkeley-Programm SIS - hat zwar die kürzesten Rechenzeiten, aber die schlechtesten Ergebnisse. Hieran sieht man, daß das akademische Werkzeug von der Industrie überholt wurde, und daß es für weitere universitäre Entwicklungen wichtig ist, sich an den aktuellen Software-Werkzeugen zu orientieren. Die in späteren Kapiteln entwickelten Design-Flüsse werden darum auch stark an das kommerzielle Synthese-System angekoppelt sein.

Die Design-Flüsse 1, 3 und 4 bringen für die großen Beispiel erhebliche Verdrahtungsprobleme mit sich, was sich auch in den langen Xilinx-Programmlaufzeiten niederschlägt, die im Falle der Nichtverdrahtbarkeit gerade besonders hoch sind.

Eine Analyse der Verlustleistung ist in Tabelle 8 dargestellt. Eine zusammenfassende Bewertung der Design-Flüsse ist auch in der Tabelle 9 in Abschnitt 7.3 enthalten.

3.3. Ergebnisse beim Entwurf mit CADDY

Die Entwurfsergebnisse bei Verwendung des High-Level-Entwurfssystems CADDY waren insgesamt etwas günstiger. Die durch CADDY erzeugten Multiplexerbäume erwiesen sich als kleiner und damit auch teilweise als einfacher zu verdrahten. Möglicherweise führt die in CADDY durchgeführte sehr umfangreiche Optimierung bei der High-Level-Synthese als Nebeneffekt zu leichter verdrahtbaren Strukturen.

Dennoch traten die oben beobachteten erheblichen Verdrahtungsprobleme auch bei Verwendung von CADDY auf: das Beispiel diffeq31 ließ sich mittels Design-Fluß 1 nicht verdrahten. Für die großen Schaltungen sind einige Ergebnisse in Tabelle 11 und Tabelle 12 enthalten.

3.4. Ergebnisse des RT-Entwurfes

Eine repräsentative Darstellung von Entwurfsergebnissen ist - wegen des großen Spektrums an Beispielschaltungen - schwer möglich. Es soll aber an dieser Stelle auf eine Klasse von Schaltungen hingewiesen werden, die sehr viele verschachtelte Verzweigungen enthalten und bei einer Abbildung auf XC4000E/EX auch deutliche Verdrahtungsprobleme verursachen (Tabelle 1 in [7]).

4. FPGA-Abbildung

4.1. Problembeschreibung

Bevor wir uns der sich im letzten Kapitel abzeichnenden Verdrahtungsproblematik näher zuwenden, soll der FPGA-Entwurf noch einmal insgesamt betrachtet werden.

FPGAs werden in steigendem Maße verwendet, z.B. für Produkte in kleinen Stückzahlen, oder etwa im Bereich des Rapid-Prototyping wegen der Möglichkeit, auch ohne lange Chip-Fabrikations-Zeiten Hardware testen zu können.

Auf dem Gebiet des automatischen FPGA-Entwurfes konzentrierte sich in der Vergangenheit die Forschung auf die Logikoptimierung, die Technologieabbildung für eine Lookup-Table-basierte Architektur (LUT), sowie Plazierung und Verdrahtung.

Das übliche Entwurfsziel besteht in der Reduktion der erforderlichen Fläche und/oder der Verzögerungzeit. Dieses wird durch eine große Zahl von Algorithmen unterstützt, aber die von uns bearbeiteten Beispiele zeigen, daß zwei andere Gesichtspunkte ebenfalls beachtet werden sollten: die Rechenzeit sowie die Verdrahtbarkeit. In der Literatur wird eine Technologie-Abbildung mit dem Ziel der Verdrahtbarkeit nur selten betrachtet. Unsere Experimente zeigen, daß für viele Schaltungs-Beispiele die aktuellen Werkzeuge keine geeigneten Designs erzeugen; ein geändertes Mapping-Verfahren ist hingegen schneller und macht manche Designs überhaupt erst möglich. Innerhalb des Design-Flusses hängt der Schritt der Verdrahtung von den Architekturdetails innerhalb des Chips ab und wird durch die kommerzielle Software automatisch durchgeführt, aber dieses Verdrahten kann vom Nutzer bereits im Mapping-Schritt beeinflußt werden, was ein wichtiges Ergebnis der vorliegenden Arbeit war, das in diesem Bericht dargestellt werden soll.

Ein Vergleich verschiedener Ansätze erfolgt aber auch im Hinblick auf den Leistungsverbrauch der fertigen Schaltung.

Unsere Zielarchitektur sind die FPGAs der Xilinx-Serien XC4000E, XC4000EX, XC4000XL, XC4000XV.

4.2. Stand der Technik

Die Vielzahl der Methoden zur Lookup-Table (LUT)-basierten Technologie-Abbildung kann in verschiedene Gruppen aufgeteilt werden: Algorithmen zur Flächenreduktion (z.B. mis-pga [19]), zur Minimierung der Verzögerungszeit (z.B. chortle-d [18], FlowMap [20], BoolMap [22], SeqMapII [23], TurboSYN [24]), kombinierten Methoden (CutMap [21], PddMap [25]) sowie Prozeduren mit anderen Gesichtspunkten (RMap [31]).

Nach einer anfänglichen Dekomposition von Knoten werden die neuen kleineren Knoten miteinander verschmolzen oder weiter aufgeteilt. Einige Werkzeuge führen ein reines Technologie-Mapping durch ([18],[21],[23]), andere hingegen beinhalten auch eine aufwendige Restrukturierung der Logik ([19],[24],[25]). Einige Ansätze behandeln nicht nur den kombinatorischen Teil einer Schaltung, sondern erlauben auch Retiming von Registern ([23],[24]).

Bei der Logik-Dekomposition und dem Mapping können zwei interessante neue Tendenzen beobachtet werden: Die Optimierung der Verzögerungszeit auf dem kritischen Pfad wird mit einer Flächen-Optimierung in den nichtkritischen Regionen sehr erfolgreich kombiniert ([21],[25]). Ein frühzeitiges Ausnutzen aller verfügbaren LUTs mit unterschiedlicher Größe erlaubt ein möglichst vollständiges Auslasten eines jeden CLBs (konfigurierbarer Logik-Block) ([25], HeteroMap [26]).

Eine Technologie-Abbildung mit Berücksichtigung der Verdrahtbarkeit wird in [31],[33] betrachtet. Im Kapitel 7.2 zeigt sich, daß dieses Kriterium nicht vernachlässigt werden sollte. In [31] wird versucht, einige Kriterien der Verdrahtbarkeit (Pin-zu-Zell-Verhältnis sowie Pin-zu-Netz-Verhältnis [32]) auf der Basis einer XC3000-Architektur zu beeinflussen. [33] führt ein inkrementelles Remapping nach der Plazierung und dem globalen Routing durch. Beide Ansätze beziehen sich auf das Überdecken der dekomponierten Logik. Im Rahmen der vorliegenden Arbeit wurden jedoch auch alternative Logik-Strukturen betrachtet.

4.3. Verlustleistungsberechnung bei FPGAs

Der Verlustleistungs-Aspekt bei FPGAs ist neben den bisher betrachteten Kategorien Verdrahtbarkeit und Entwurfszeit ebenfalls ein wichtiger Gesichtspunkt. Der FPGA-Entwurf mit Blickwinkel auf eine geringe Verlustleistung erfolgte aber hauptsächlich durch nachträgliche lokale Optimierung zwischen direkt verbundenen LUTs [44],[45],[46]. Eine weitere interessante Arbeit zum Thema FPGA und Verlustleistung findet sich unter [41].

Zur Bestimmung des Leistungsverbrauches einer Schaltung gibt es verschiedene Werkzeuge. Z.B. das Programmpaket SIS von Berkeley [39], DesignPower von Synopsys [48] oder Powermill von EPIC [47]. Diese sind aber auf Gatterlogik, nicht auf FPGAs abgestimmt.

Für ein fertiges FPGA-Design gibt es bisher keine Möglichkeit, durch Computerberechnungen den Leistungsverbrauch exakt zu bestimmen, es sei denn, man führt praktische Tests damit aus. Xilinx selbst bietet kein Werkzeug an, nur grobe Faustformeln ($P_{INT} = V_{CC} \times K_P \times F_{MAX} \times N_{LC} \times TOG_{LC}$, [42]), in die (neben Versorgungsspannung V_{cc}, Baustein-Parameter K_p und Taktfrequenz F_{max}) lediglich die Summe der Schaltaktivitäten aller CLB-Ausgänge eingehen ($N_{LC} \times TOG_{LC}$), nicht aber der Verbrauch auf den Leitungen.

Eine sehr eingeschränkte Leistungsberechnung für die alten FPGA-Serien XC2000 - XC4000, d.h. nicht für die neuen XC4000E und Nachfolger, ist unter [43] zu finden.

Als ein anderer Weg ist auch eine Übersetzung des plazierten und verdrahteten Designs (ncd-File) von Xilinx in ein annotiertes VHDL-File möglich, wobei aber die Information verlorengeht, daß mehrere logische Gatter in einem LUT gemeinsam untergebracht sind; somit würden hier die (virtuellen) Zwischenknoten fälschlicherweise in die Aktivitätsberechnung eingehen. Ferner beinhalten die Annotationen nur die Verzögerungen, nicht aber die Leitungskapazitäten.

5. Verdrahtbarkeit von FPGAs

Die Verdrahtbarkeit ist im FPGA-Design-Prozeß wesentlich. Denn ein Scheitern bei der FPGA-Abbildung wegen Überschreitung der Logik-Ressourcen kann ziemlich schnell festgestellt werden (nach dem Mapping-Schritt), hingegen kann ein Scheitern wegen zu wenig Routing-Ressourcen erst nach einer sehr langen Rechenzeit festgestellt werden (in den gerechneten Beispielen waren es bis zu 4 Tagen CPU-Zeit, siehe Tabelle 2). Deshalb ist eine Entspannung des späteren Verdrahtungsproblemes bereits während des vorherigen Mapping-Prozesses sehr nützlich.

5.1. Vorarbeiten und Begriffsbildung

Das Wort „Verdrahtbarkeit“ (engl. Routability) wird in mehreren Weisen verwendet: als quantitatives Maß sowie als qualitative Aussage. „**Routability** is the probability of automatic completion of a particular design“ nach dem Mapping [40]. Brown, Rose und Vranesic [29] benutzen das Wort **routability** als den Prozentsatz der 2-Punkt-Verbindungen, die verdrahtbar sind.

Einige Forscher untersuchen die Struktur der Switch-Boxes am Schnittpunkt der Kanäle sowie die notwendige und geeignete Anzahl der Verbindungspunkte [29],[30],[34]. Ein probabilistisches Modell des Routing-Prozesses in [35] extrahiert die Spanne eines Netzes (d.h. die größere Kante des umschreibenden Rechtecks) als ein Verdrahtbarkeits-Kriterium.

Ein Standardzell-Design kann leichter verdrahtet werden, da die Kanalbreite noch einstellbar ist. In FPGAs sind die Leitungskanäle fest, ebenso sind die Kontaktpunkte limitiert. Dieser letzte Aspekt ist der Grund für folgende Definition des „**Routability** problems“: „determine if a given global routing is feasible“ [34]. Von der sog. Switchbox-Struktur ist es abhängig, ob dieses Problem entweder sehr einfach ist, ob es zu P gehört, oder ob es NP-vollständig ist [30],[34]. Letzteres trifft für die XC4000-Architektur zu.

In den bisherigen Untersuchungen können 2 Einschränkungen beobachtet werden:

1. Netze werden oft in 2-Punkt-Netze aufgeteilt [31],[29].
2. Die Verdrahtbarkeit wird meist für FPGA-Strukturen mit einheitlicher Verbindungslänge in den Kanälen betrachtet [29],[35]. Die FPGA-Strukturen besitzen jedoch Direktverbindungen, die 1, 2, 4, oder 8 Basis-Einheiten lang sind, bzw. lange Verbindungen über das ganze Chip. Dieses wird in den Verdrahtungs-Algorithmen ausgenutzt, wird aber nicht in theoretischen Verdrahtbarkeits-Analysen betrachtet.

Die Bestimmung der Verdrahtbarkeit kann in 3 Stufen erfolgen:

Stufe 1: Die Verwendung quantitativer Werte zur Bestimmung des späteren Verdrahtungsaufwandes (z.B. das Pin-zu-Zell-Verhältnis [31]) erlaubt nur eine sehr grobe Aussage zur Verdrahtbarkeit eines Designs.

Stufe 2: Teilweise wird die Verdrahtbarkeit im Zusammenhang mit einem gleichzeitigen globalen Routing betrachtet [37]. Dies erlaubt es, die Auslastung von Verdrahtungskanälen abzuschätzen [37], aber nicht die Einschränkungen wegen der Verdrahtungspunkte.

Stufe 3: Führt man ein komplettes Routing durch, so erlaubt dies die Berücksichtigung der Verdrahtungskanäle, der Kontaktpunkte sowie der genauen Lage der inneren und äußeren Pins. Diese Stufe 3 ist die zuverlässigste, aber auch die zeitaufwendigste. In [28] werden praktische Tests dargestellt.

Die Verdrahtbarkeit hängt also ab von:

- *Charakteristika des Designs (Kriterien von Stufe 1):*
  1. Prozentsatz der CLB-Auslastung,
  2. Anzahl der Netze,
  3. Verhältnis der Block-Ein/Ausgangs-Anzahl zur Zahl der Netze (pin-to-net-ratio), sowie die Anzahl der Ein/Ausgänge je Block (pin-to-cell-ratio) [31],
  4. Makroblocks (diese sind beim Plazieren und Verdrahten hinderlich),
- *Ressourcen im FPGA (diese können auf den Stufen 2 und 3 betrachtet werden):*
  5. Anzahl der Verbindungen pro Kanal,
  6. Struktur der Switch-Boxes, speziell die Anzahl der Verbindungspunkte darin,
- *Eigenschaften der Programme und Werkzeuge (diese betreffen alle Stufen 3, 2 und 1):*
  7. Qualität des Plazierungs- und Verdrahtungsprogramms,
  8. Rechenaufwand beim Verdrahten,
  9. Beeinflussung der Verdrahtbarkeit während des Mappings,
- *Kriterien für andere Zielarchitekturen, z.B. ASICs:*
  10. Planarität (zum Vermeiden von Leitungskreuzungen [36]), sowie
  11. die Möglichkeit des Verdrahtens durch Makroblöcke hindurch.

Makro-Blöcke (Kriterium 4) stören nicht nur beim Plazieren, sondern erfordern auch Bündel von Leitungen, die zu ihnen hingeführt werden müssen.

Die neuen Architekturen XC4000E und nachfolgende Serien haben mehr Routing-Ressourcen als die ältere XC4000 Baureihe (Kriterium 5 und 6).

5.2. CLB-Reduktion und Verdrahtbarkeit

Eine Reduktion der Anzahl der Logikblöcke (Kriterium 1) ist ein übliches Ziel der Mapping-Werkzeuge, aber unsere Experimente zeigen, daß dies nicht immer vorteilhaft ist in Bezug auf die Verdrahtbarkeit. Bei früheren Untersuchungen [6] zur älteren XC4000 Reihe mit geringeren Kanalbreiten wirkte sich dieser Sachverhalt noch gravierender aus.

Eine mögliche Erklärung für eine schlechtere Verdrahtbarkeit bei weniger Logikblöcken wird in Abb. 2 dargestellt.

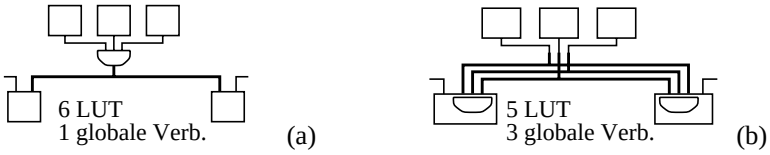

Abbildung 2. Einfaches Mapping (a), Verbessertes Mapping (b).

5.3. Multiplexer und Verdrahtbarkeit

In unseren Experimenten beobachteten wir, daß die Multiplexer ein Schlüsselproblem beim Design darstellen. Sie erwiesen sich als das Haupthindernis für die Verdrahtung vieler großer Designs.

Eines unserer Beispiele mit Verdrahtungsproblemen ist der Schaltkreis für die Diskrete Cosinus-Transformation. [27] stellt fest, daß für diese Schaltungsklasse die Hälfte der Chip-Fläche für die Verbindungen innerhalb des Chips erforderlich ist. Dieser Umstand unterstreicht die Notwendigkeit ein Mapping-Verfahren zur Reduktion der Routing-Probleme bereitzustellen.

6. Verschiedene Multiplexer-Varianten: Synthese von Multiplexern

Wie schon in Abschnitt 5.3 bemerkt, sollten die Multiplexer in das Mapping einbezogen werden. Deshalb werden hier zunächst verschiedene Varianten zusammengestellt.

6.1. Variante “Kommerzielles Synthese-System KSS“

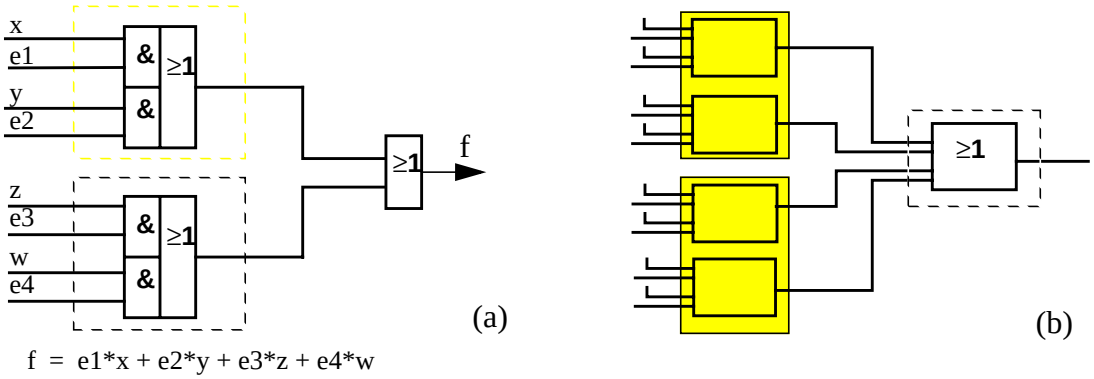

Abbildung 3. Multiplexer-Variante 1: 4 Eingänge (a), 8 Eingänge (b).

Bei diesem Multiplexer werden UND-ODER-Bausteine gebildet und mit einem abschließendem ODER-Gatter zusammengefaßt. In Abb. 3 wurde dabei zur besseren Verdeutlichung des Hardwareaufwandes die Zusammenfassung der Logik zu einem Look-Up-Table (LUT) durch eine gestrichelte Linie angedeutet; für den Multiplexer mit 8 Eingängen wird die in einem CLB gruppierbare Logik zusammengefaßt und grau unterlegt (Abb. 3b).

6.2. Variante “kanonischer 2-zu-1-Baum“

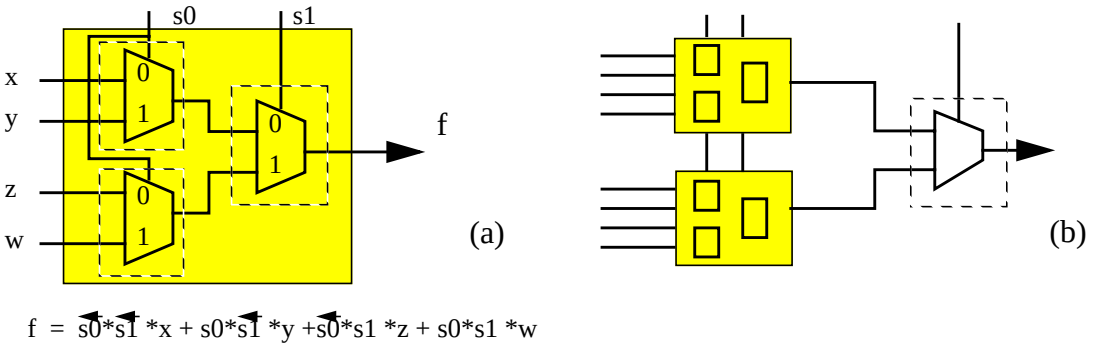

Abbildung 4. Multiplexer-Variante 2: 4 Eingänge (a), 8 Eingänge (b).

Dieser Multiplexer (Abb. 4) ist vollständig aus 2-zu-1-Multiplexer aufgebaut. Drei davon passen genau in einen CLB.

6.3. Variante “Dekomposition“

Diese Variante ist die sparsamste: nur 2 LUTs sind erforderlich, um 4 Eingänge zu multiplexen. (Abb. 5).

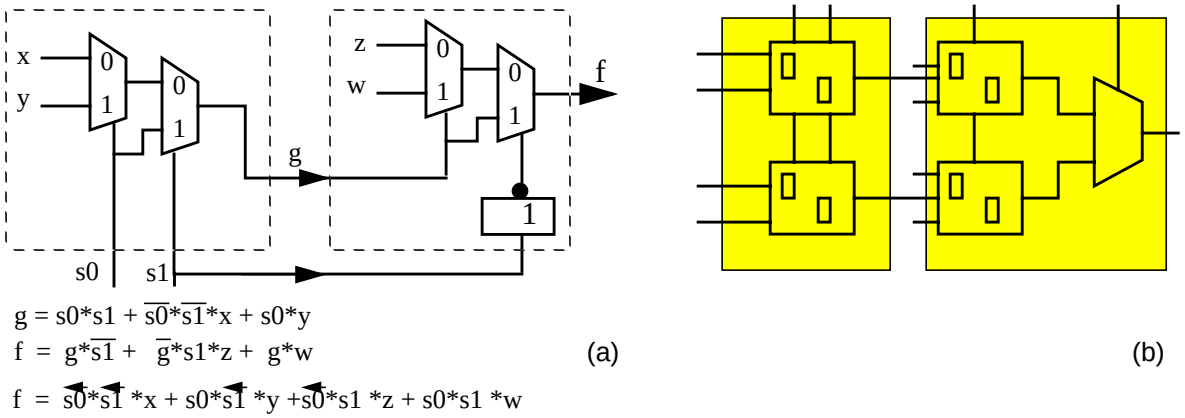

Abbildung 5. Multiplexer-Variante 3: kompakteste Form, für 4 Eingänge (a) und 8 Eingänge (b).

Für einen Multiplexer mit 4 Eingängen hat diese Variante aber den Nachteil, daß die beiden erforderlichen LUTs nicht so günstig in einen XC4000-CLB gepackt werden können: es wäre (für das Signal ‚g‘) eine Rückführungsleitung außen herum erforderlich. Somit ist für einen Multiplexer mit 4 Eingängen die zweite Variante (Abb. 4) sogar trotzdem günstiger.

6.4. Variante “Xilinx : Logiblox-Generator - ‘kleinste Verzögerung‘ “

Dieser Multiplexer ist sehr flächenintensiv, weil die Ansteuersignale für die TBUF immer wieder neu berechnet werden. (Die Und-Gatter befinden sich immer gleich neben dem jeweiligen T-Buffer, den sie steuern.) Und es wird für jedes Ausgabe-Bit eine der zahlenmäßig wenigen horizontalen Longlines benötigt. Dieser Multiplexer wurde nicht weiter untersucht, da er zu viele CLB- sowie Routing-Ressourcen benötigt. Er ist von Xilinx für 4 bis 8 Eingänge verfügbar.

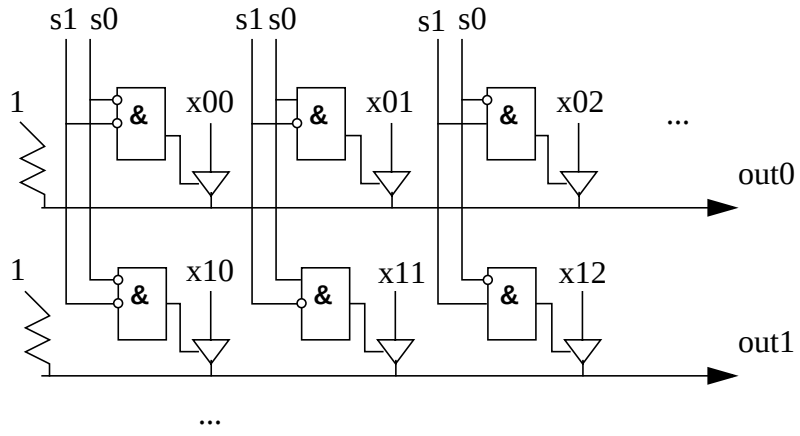

Abbildung 6. Multiplexer-Variante 4: Xilinx Logiblox Generator - ‘kleinste Verzögerung‘.

6.5. Variante “Xilinx : Logiblox-Generator - ‘kleinste Fläche‘ “

Dieser Multiplexer ist für 4 bis 8 Eingänge verfügbar und ähnelt der Variante aus Abschnitt 6.2. Er wurde nicht weiter untersucht.

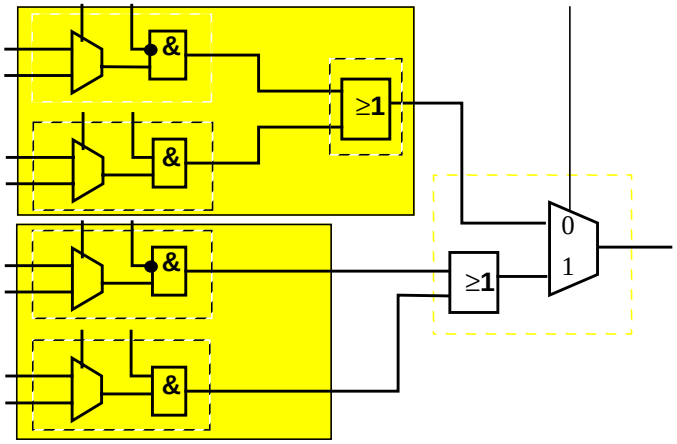

Abbildung 7. Multiplexer-Variante 5: Xilinx Logiblox Generator - ‘kleinste Fläche‘ für 8 Eingänge.

6.6. Analyse der Verlustleistung der Multiplexer-Varianten

Bevor die Multiplexer-Varianten umfassend verglichen werden, soll in diesem Abschnitt die Verlustleistung der vorgestellten 5 Multiplexer-Varianten bestimmt werden. Hierzu werden die Multiplexer als Einzelschaltungen betrachtet, und die Ermittlung der Verlustleistung erfolgt mit SIS [39] auf der Grundlage der Formel für P_{INT} aus Abschnitt 4.3.

Tabelle 3: Verlustleistungs-Ermittlung mit SIS / Multiplexer mit 8 Eingängen / 32 Bit Wortlänge / ohne Leitungs-Verzögerung

	Var1	Var2	Var3	Var4	Var5
Leistungs-verbrauch	***779***	1240	1080	1635 + TBUF	1060

In Tabelle 3 sieht es so aus, als sei unter Verlustleistungs-Gesichtspunkten die Multiplexer-Struktur 1 die beste. Die Tatsache, daß jedes Multiplexer-Daten-Eingangssignal in der ersten Schicht des Multiplexers an eine Art Sperrgitter kommt, setzt die Rate unnützer Signalwechsel deutlich herab. Aber durch den höheren Verdrahtungsaufwand und die zu erwartenden größeren Leitungslängen, Leitungsverzögerungen und Leitungskapazitäten steigt die Verlustleistung stärker als bei den anderen Alternativen (vgl. folgende Kap., insbesondere Kap. 7.2.2).

6.7. Vergleich der Multiplexer - Varianten

Die vorgestellten Multiplexer-Strukturen 1-3 haben einige Vor- und Nachteile, die im folgenden verglichen werden sollen. Die Struktur 1 hat geringe Verzögerung und geringe Leistungsaufnahme, aber sie ist ungünstig bezüglich der Verdrahtbarkeit. Dies wird an der Abb. 8 deutlich.

In Abb. 8 stellt Teil A den Controller dar, dessen Zustandscode über den Schnitt 1 hinweg im Teil B zur Erzeugung der Controller-Ausgabe-Signale bzw. Multiplexer-Steuer-Signale ver-

wendet wird. Den eigentlichen Multiplexer stellt Teil C dar, der über den Schnitt 2 versorgt wird. Dieser Schnitt2 ist problematisch, da ein mehr an Signalen hier sich im gesamten Multiplexer auswirkt, denn Teil C wiederholt sich für jedes Ausgabe-Bit des Multiplexers. Somit ist Struktur 1 wesentlich schlechter zu verdrahten.

Die mit F (bzw. H) gekennzeichneten Rechtecke in Abb. 8 und ebenso die Spalten FLUT (bzw. HLUT) in Tabelle 5 entsprechen den speziellen LUTs mit max. 4 (bzw. 3) Eingängen in den XC4000-FPGAs.

Abb. 8 stellt eine konkrete Anwendung des in Abb. 2 skizzierten Phänomens dar.

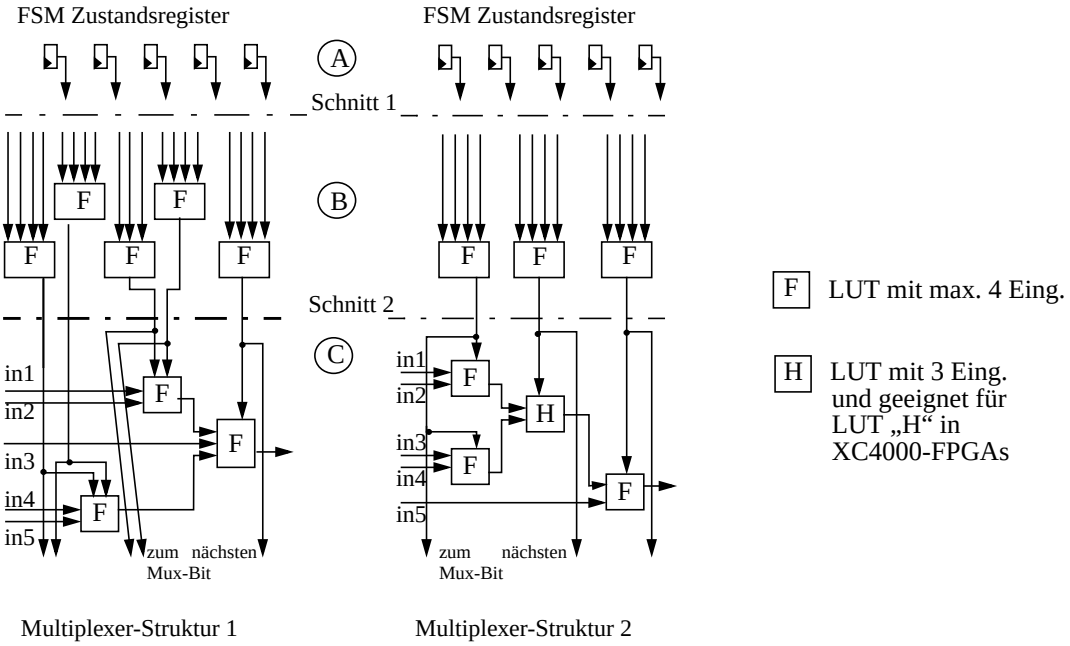

Abbildung 8. Vergleich der Multiplexer-Strukturen 1 und 2 im Umfeld mit dem Controller.

Für einen Multiplexer mit 5 Eingängen ist die Struktur 2 besser als Struktur 1, bei einem Multiplexer mit 6 Eingängen ist die Frage nicht mehr so eindeutig. In beiden Fällen ist Struktur 3 aber am besten.

Tabelle 4: Multiplexer mit 5 und 6 Eingängen

	Multiplexer 5 Eingänge			Multiplexer 6 Eingänge		
	Struktur 1	Struktur 2	Struktur 3	Struktur 1	Struktur 2	Struktur 3
Breite des Schnitt 2	5	**3**	**3**	6	**3**	**3**
F LUT in Bereich B	5	**3**	**3**	6	**3**	**3**
F LUT in Bereich C	b * 3	b * 3	**b * 2**	**b * 3**	b * 4	**b * 3**
H LUT in Bereich C	0	b * 1	b * 1	b * 1	b * 1	b * 1
b = 32: #CLB	50.5	**49.5**	**33.5**	**51**	65.5	**49.5**
b = 32: #Pins in C	320	**256**	**256**	384	**288**	**288**
längster Pfad	**F + F**	F + H + F	F + F + H	**F + H**	F + H + F	F + F + H
Fläche	++	+++	++++	+++	-	+++
Verzögerung	+++	++	++	+++	++	++
Power	+++	+	+	+++	+	+
Verdrahtbarkeit	-	+++	+++	-	+++	+++

Insgesamt ergeben sich als Flächen-Werte für die 3 Multiplexer-Varianten die Zahlen aus Tabelle 5. Dabei ist b die Wortlänge. Die günstigsten Flächenwerte sind fett hervorgehoben. Die beiden rechten Spalten wurden nur als Vergleichsinformation hinzugefügt.

Tabelle 5: Anzahl der LUTs für Multiplexer mit Wortlänge b=32

	Struktur 1		Struktur 2		Struktur 3		Struktur 4		Struktur 5	
#Eingänge	#F LUT	#H LUT	#F LUT	#H LUT	#FLUT	HLUT	#FLUT	#HLUT	#FLUT	#HLUT
2	33	0	33	0	-	-	-	-	-	-
3	34	32	34	32	-	-	-	-	-	-
4	68	32	[a] **66**	32	[b] 66	0	130	0	66	32
5	101	0	[a] 99	32	66	32	163	0	131	0
6	[c] 102	32	131	32	**99**	32	195	0	131	0
7	[c] 135	32	163	32	**132**	32	227	0	163	0
8	168	0	[a] 163	64	132	32	259	0	163	32
9	[c] 170	32	196	64	**165**	32	-	-	-	-
10	202	32	[a] **196**	96	**196**	32	-	-	-	-
11	235	0	[a] **228**	96	**228**	32	-	-	-	-
12	[c] 236	32	260	96	**228**	32	-	-	-	-
13	269	32	[a] **260**	128	**260**	32	-	-	-	-
14	302	0	[a] 292	128	**260**	64	-	-	-	-
15	[c] 303	32	324	128	**292**	64	-	-	-	-
16	336	32	[a] 324	160	**292**	64	-	-	-	-

a. Multiplexer-Struktur 2 ist kleiner als Struktur 1.
b. ungünstige Schaltung : sie paßt nicht in ein CLB.
c. Multiplexer-Struktur 1 ist kleiner als Struktur 2.

Im Ergebnis des Kapitels 6 kann man festhalten:

- Die Multiplexer-Struktur 1 ist ungünstig bezüglich der Verdrahtung. Multiplexer dieser Art sollten möglichst ausgetauscht werden.
- Die Strukturen 1 und 2 wechseln sich unregelmäßig bezüglich des Flächenverbrauches ab (Tabelle 5).
- Struktur 3 liefert die kleinsten Multiplexer.

7. FPGA-Mapping mit Berücksichtigung der Verdrahtbarkeit

Im weiteren sollen 2 Algorithmen zur Multiplexer-Auswahl aufgestellt werden.

Algorithmus 1 zum Mapping für Verdrahtbarkeit bezieht sich auf das Austauschen von Multiplexern mit Struktur 1 durch solche mit Struktur 2. Dazu definieren wir zwei Mengen: $Set1$ besteht aus den Eingangs-Zahlen, bei denen Struktur 1 kleiner ist als Struktur 2:

$$Set1 = \{6, 7, 9, 12, 15, 18\} .$$

Der umgekehrte Fall wird in $Set2$ erfaßt:

$$Set2 = \{4, 5, 8, 10, 11, 13, 14, 16, 17\} .$$

Somit kann für Multiplexer mit $numOfInputs \in Set2$ ein von der Verdrahtbarkeit verursachtes Ersetzen der Multiplexer durch solche mit Struktur 2 immer erfolgen. Hingegen kann ein

solches Austauschen für Multiplexer mit $numOfInputs \in Set1$ nur dann empfohlen werden, wenn noch genug freie Fläche im FPGA enthalten ist.

Man beachte noch, daß Multiplexer mit 2 oder 3 Daten-Eingängen in dieser Klassifikation ignoriert und damit nicht verändert werden, da sonst ein zu hoher Verlust an Flexibilität bezüglich einer Logik-Optimierung eintreten würde.

Der Algorithmus hierzu ist Algorithmus 1. Die Angabe “16 CLB“ (Schritt 6 des Algorithmus) entspricht als maximal zu erwartendem Flächenzuwachs dem größten Unterschied zwischen Spalte 4 und Spalte 2 in Tabelle 5 (dividiert durch 2).

// Algorithmus 1 Mapping_for_Routability
begin
0: High-Level-Synthese
1: Ausschneiden aller Multiplexer mit
$numOfInputs \in Set2 = \{4, 5, 8, 10, 11, 13, 14, 16, 17\}$
2: FPGA-Mapping der restlichen und ausgeflachten Schaltung
3: Hinzufügen logisch äquivalenter Multiplexer mit der speziellen Struktur 2
4: Xilinx Tools für Placement und Routing
5: **if** (Routing ist nicht möglich) **then**
6: { **if** (ca. 16 CLBs sind frei **and** Design enthält einen Multiplexer
mit der Struktur 1 und $numOfInputs \in Set1$)
7: **then** {Ausschneiden des Multiplexer mit den meisten Eingängen; **goto** 2 }
8: **else** {Übergang zum nächstgrößeren FPGA; **goto** 1}
}
end

Durch die flächengünstige Multiplexer-Variante 3 ergibt sich noch ein anderer Zugang: Man könnte alle Multiplexer mit 4 und mehr Dateneingängen austauschen, solange man beachtet, daß dadurch nicht das Optimierungspotential beeinträchtigt wird. Hierbei ist zu beachten, daß Konstanten an den Dateneingängen eines Multiplexers unbedingt schon während der Logikoptimierung berücksichtigt werden sollten; ein späteres Einbeziehen ergibt keine so günstigen Mapping-Lösungen mehr. Gleiches gilt, wenn Shift-Operatoren vor dem Multiplexer liegen. Hieraus ergibt sich Algorithmus 2.

// Algorithmus 2 Mapping_for_Routability
begin
0: High-Level-Synthese
1: Ausschneiden aller Multiplexer mit mindestens 4 Eingängen, sofern an den
Multiplexer-Eingängen nicht vorrangig Konstanten bzw. Shift-Operationen liegen.
2: FPGA-Mapping der restlichen und ausgeflachten Schaltung.
3: Hinzufügen logisch äquivalenter Multiplexer mit
der speziellen Struktur 3 (ab 5 Eingängen) bzw. Struktur 2 (4 Eingänge)
4: Xilinx Tools für Placement und Routing
5: **if** (Routing ist nicht möglich)
6: **then** {Übergang zum nächst größeren FPGA; **goto** 1}
end

7.1. Drei neue Design-Flüsse

In Ergänzung zu den anfänglichen Design-Flüssen zu Beginn der Arbeit aus Kapitel 2 werden jetzt die im Rahmen der Arbeit zu diesem Report entwickelten 3 neuen Design-Flüsse von der High-Level-Beschreibung zum FPGA vorgestellt.

Die Design-Flüsse 6-8 (Abb. 9) sind neu und sollen die Schwachpunkte der existierenden Design-Flüsse 1-5 (Abb. 1, Tabelle 1) beseitigen sowie ihre jeweiligen Vorteile übernehmen.

`Design-Fluß 6, 7 und 8:` Diese Design-Flüsse entsprechen Design-Fluß 1. Lediglich die Multiplexer werden - wie weiter oben in Abschnitt 7 beschrieben - gesondert behandelt. Dabei verläuft Design-Fluß 6 gemäß Algorithmus 1, Design-Fluß 7 entspricht Algorithmus 2 und Design-Fluß 8 benutzt als Spielart zu Design-Fluß 7 auch alle Multiplexer mit 4 Eingängen in der Multiplexer-Struktur 3.

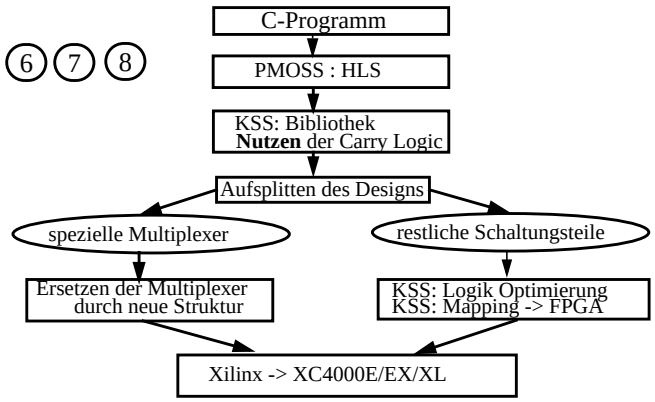

Abbildung 9. Neue Design-Flüsse von der High-Level-Synthese zum FPGA.

7.2. Ergebnisse der 3 neuen Design-Flüsse

Die Ergebnisse für die 3 neuen Design-Flüsse sind in den Tabellen 6 und 7 enthalten. Die benutzten Beispiele sind wieder die 6 C-Programme aus Kapitel 3. Die Ergebnistabellen stellen eine direkte Fortsetzung von Tabelle 1 und Tabelle 2 dar.

7.2.1 Erreichte Schaltungs-Charakteristika : Fläche, Verzögerung und Rechenzeit

In Kapitel 3 hatte Design-Fluß 2 weniger Design- und Routing-Probleme als Design-Fluß 1 (er erlaubt kleinere FPGAs), dafür machte sich aber die Nicht-Benutzung der Carry-Logik bei der Verzögerungszeit bemerkbar. Erstaunlicherweise bringen die Makroblöcke der schnellen Addierer usw. solche Verdrahtungsprobleme mit sich, daß erst durch die neuen Design-Flüsse 6-8 die Einbindung der Makroblöcke ohne negative Auswirkung, d.h. mit Verwendung des kleinsten nutzbaren FPGA, möglich wird.

Tabelle 6: Ergebnisse der 3 neuen Design-Flüsse, Fortsetzung von Tabelle 1

			Design-Fluß 6: KSS und wir			Design-Fluß 7: KSS und wir			Design-Fluß 8: KSS und wir		
Bsp.	#Eing.	#Ausg.	FPGA	#CLB	ns	FPGA	#CLB	ns	FPGA	#CLB	ns
fibo04	6	5	siehe Design-Fluß 1			siehe Design-Fluß 1			siehe Design-Fluß 1		
fibo08	10	9									
fibo16	10	17									
fibo32	10	33									
gcd04	10	5	4003E	24	***36.5***	4003E	24	***36.5***	4003E	24	46.4
gcd08	18	9	4003E	38	59.8	4003E	38	59.8	4003E	38	54.0
gcd16	34	17	4003E	66	***57.9***	4003E	66	67.3	4003E	66	65.4
gcd32	66	33	4005E	118	95.0	4005E	118	83.0	4005E	120	94.1
atoi08	17	16	siehe Design-Fluß 1			siehe Design-Fluß 1			siehe Design-Fluß 1		
atoi16	17	25									
atoi32	17	41									
ellip04	34	33	4005E	110	***81.5***	***4003E***	***96***	***80.9***	***4003E***	***96***	***79.0***
ellip08	66	65	4008E	186	***101.6***	4008E	***165***	120.1	4008E	***164***	110.4
ellip15	122	121	4025E	317	***167.0***	4025E	***282***	180.5	4025E	***282***	172.8
ellip16	130	129	4036EX	338	148.7	4036EX	***300***	***131.5***	4036EX	301	143.8
ellip23	186	185	4062XL	466	176.2	4062XL	***421***	***164.7***	4062XL	***418***	175.8
diffeq04	22	13	4003E	***65***	58.4	4003E	***65***	***51.3***	4003E	66	55.1
diffeq08	42	25	4005E	***133***	99.0	4005E	***133***	***88.4***	4005E	***133***	113.8
diffeq16	82	49	***4008E***	***310***	***174.9***	***4008E***	***310***	181.1	***4008E***	***310***	189.3
diffeq24	122	73	4025E	***555***	379.4	4025E	***555***	388.7	4025E	556	332.6
diffeq31	157	94	***4025E***	***822***	***424.3***	***4025E***	***822***	521.1	***4025E***	***822***	487.3
dct24	35	31	***4036EX***	1218	378.6	***4036EX***	***1108***	***342.2***	***4036EX***	***1106***	375.5
dct32	35	31	***4052XL***	1837	462.0	***4052XL***	***1728***	***381.1***	***4052XL***	1733	435.9

Tabelle 7: CPU-Programm-Laufzeiten für die 3 neuen Design-Flüsse (in Std:Min:Sec, auf einer Sparc Ultra 2, 640 MB Hauptspeicher) (Fortsetzung von Tabelle 2)

	Design-Fluß 6		Design-Fluß 7		Design-Fluß 8	
Beispiel	KSS	Xilinx	KSS	Xilinx	KSS	Xilinx
fibo04	siehe Des.-Fluß 1		siehe Des.-Fluß 1		siehe Des.-Fluß 1	
fibo08						
fibo16						
fibo32						
gcd04	1:50	0:24	1:50	0:24	1:55	0:24
gcd08	1:58	0:31	1:58	0:31	1:59	0:31
gcd16	2:18	0:39	2:36	0:54	2:24	0:38
gcd32	3:46	1:18	3:44	1:32	3:59	1:19
atoi08	siehe Des.-Fluß 1		siehe Des.-Fluß 1		siehe Des.-Fluß 1	
atoi16						
atoi32						
ellip04	5:24	***1:04***	6:29	***0:44***	6:16	0:52
ellip08	6:18	***2:00***	7:31	2:04	6:57	2:17
ellip15	8:18	***6:32***	9:39	***6:13***	9:35	***6:01***
ellip16	8:35	4:06	9:56	***3:46***	8:24	4:17
ellip23	12:00	5:26	13:25	5:54	12:11	5:27
diffeq04	3:30	0:35	3:33	0:35	3:46	0:34
diffeq08	4:06	1:11	4:05	1:14	4:08	1:18
diffeq16	6:07	2:53	6:24	2:45	6:07	3:02
diffeq24	10:06	9:04	15:31	9:45	10:02	9:00
diffeq31	14:32	16:53	14:27	14:03	14:36	1:36:02
dct24	1:20:04	***10:23***	1:17:13	11:58	1:29:33	10:53
dct32	1:41:46	***35:14***	1:41:17	***28:53***	1:48:54	53:53

7.2.2 Verlustleistungsbestimmung bei den Beispielschaltungen

Wie schon im Kapitel 4.3 erwähnt, kann der Leistungsverbrauch eines FPGA nicht direkt berechnet werden. Deshalb wurde zur Abschätzung der Verlustleistung folgender Weg beschritten:

- Die logische Funktion eines jeden CLB wurde aus dem Xilinx-Design (ncd-File) herausgelesen. Dies beinhaltet die 3 LUTs sowie die Carry-Logik.
- Die genaue Verdrahtung zwischen den CLB kann wegen der binären Codierung nicht aus dem ncd-File erhalten werden.
- Die Anfangs- und Endpunkte der Verdrahtung ist im Xilinx-dly-File enthalten, einschließlich der Punkt-zu-Punkt-Leitungsverzögerung.
- Aus diesen Informationen kann ein blif-File generiert werden, das die Leitungsverzögerungen als Kommentar mit enthält.
- Die weitere Analyse wurde mit dem Berkeley-Programmpaket SIS [39] durchgeführt.
- Durch Veränderung an SIS konnte erreicht werden, daß
  - SIS einen LUT als eine Einheit behandelt und für Kapazitätsberechnungen nicht noch weiter zerlegt,
  - SIS die Verzögerungen mit einliest und sie den Kanten zuordnet,
  - SIS bei der Power-Analyse diese Verzögerungen mit berücksichtigt.

Ein Problem stellen noch die Leitungskapazitäten im FPGA dar. Darüber gibt es keine Angaben und die Faustformeln (siehe Kapitel 4.3) berücksichtigen nur eine feste Lastkapazität pro CLB-Ausgang. In der vorliegenden Arbeit wurden die Leitungskapazitäten wie folgt zu berücksichtigen versucht:

Modell 1: Es wurde ein Zero-Delay-Modell betrachtet (verzögerungsfreie Verarbeitung der Daten), bei der keine unnötigen Signalwechsel (Glitches) auftreten. Ferner wurde angesetzt: Spannung 3.3V, Frequenz 1MHz, Lastkapazität eines CLB-Ausganges: 20 pF + 1pF je Fanout.

Modell 2 berücksichtigt in einem sog. General-Delay-Model die Leitungsverzögerungen. Außerdem wurde hier zur Modellierung der unterschiedlich langen Leitungen für die Lastkapazität eines CLB-Ausganges folgender Wert verwendet: 20pF + 1pF je Fanout + 1pF je 1ns der Summe aller Verzögerungen aller angeschlossenen Verbindungen.

Die nach Modell 1 berechnete Verlustleistung für die CLB ist genauer als die nach Modell 2, denn bei Modell 2 wird die Kapazität künstlich erheblich vergrößert. Andererseits wird der Leitungseinfluß durch Modell 2 besser erfaßt.

Tabelle 8 enthält die Ergebnisse der Verlustleistungsberechnung: Für jedes Beispiel und jeden Design-Fluß wurde entsprechend Modell 1 und 2 ein Wertepaar ermittelt. Trotz einiger Ungenauigkeiten in Tabelle 8 (die Carry-Logik-Ausgänge werden mit der vollen Kapazität der normalen CLB-Ausgänge belegt, was zu zu hohen Werten bei Design-Fluß 1,6,7,8 führt) schneiden die Design-Flüsse 6 und 7 auch bei der Frage der Verlustleistung gut ab.

Wegen Tabelle 3 wären die Design-Flüsse 1 und 2 mit der Multiplexer-Struktur 1 als die besten zu erwarten gewesen. Aber für die Gesamtschaltungen ergeben sich andere Ergebnisse (Tabelle 8).

Tabelle 8: Leistungsverbrauch in µW: Resultate von den 8 Design-Flüssen der Abb. 1 und Abb. 9 für 2 Modelle der Leitungskapazität

	Des.-Fluß 1		Des.-Fluß 2		Des.-Fluß 3		Des.-Fluß 4		Des.-Fluß 5		Des.-Fluß 6		Des.-Fluß 7		Des.-Fluß 8	
Bsp.	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
fibo04	2138	2798	***1772***	***2232***	1818	2248	1920	2658	1914	2664						
fibo08	3910	5730	3326	4362	***3112***	***4326***	3282	5060	3608	5708	siehe Design-Fluß 1					
fibo16	6460	9774	5438	***7868***	***5368***	8374	5444	8902	6156	10256						
fibo32	11328	19346	9876	***15026***	9772	15436	***9756***	17046	11690	20004						
gcd04	2200	4332	1866	3898	***1810***	***3108***	1822	3454	2300	4606	2392	4130	2364	4228	2536	5128
gcd08	3878	11088	3374	8198	***3054***	***6276***	3180	7558	3618	8178	3976	10298	4078	10252	4394	11106
gcd16	7026	20840	6320	15988	***5682***	***13652***	5982	15064	6196	14980	7054	20450	7196	17778	7904	22490
gcd32	12950	48514	12532	36852	***10914***	***31286***	11442	35618	14516	38824	13232	41506	13242	42686	14616	50580
atoi08	6366	11310	***5084***	***8914***	5174	9632	5322	9454	5692	11616						
atoi16	9842	17450	***7388***	***12486***	***7388***	12840	7680	13596	9380	19744	siehe Design-Fluß 1					
atoi32	16408	31888	12788	26244	***12658***	***25684***	12998	26026	13998	26306						
ellip04	9934	36694	***9666***	35014	12322	41782	11392	42434	12854	51298	10312	***33810***	10352	34682	10720	35254
ellip08	16982	82770	***17232***	79122	20886	97682	20666	104354	22840	108088	17948	***71214***	18258	84566	19262	85564
ellip15	***29370***	228598	30016	***183264***	33046	212782	30084	229038	34870	241726	31166	199454	32076	221230	33910	223216
ellip16	***31160***	234506	31914	***206862***	34764	244436	32700	268728	36954	285106	33146	221394	34152	243926	35768	248168
ellip23	***43430***	338218	44740	303224	48532	***299712***	44684	350222	52838	376460	46234	330512	47652	349434	50524	346436
diffeq04	6578	16490	6664	15898	6682	17268	6656	18420	7708	21020	6490	15818	***6392***	***15134***	6624	16014
diffeq08	12918	47450	13312	51258	13990	62996	13700	57302	14892	79006	12816	46642	***12738***	***44258***	13316	50446
diffeq16	29676	304502	30888	280740	32862	419098	31690	385918	35234	616494	29266	***246498***	***28966***	273570	30448	299062
diffeq24	50570	1317510	52934	***974560***	56230	1544858	54116	1585692	64024	[a]	50442	1047342	***49958***	1090410	52092	1075704
diffeq31	74164	a	78288	a	80464	a	77736	a	93872	a	***72260***	a	72464	a	74726	a
dct24	***81010***	a	81664	a	98952	a	100288	a	106292	a	87264	a	88830	a	87124	a
dct32	***110334***	a	113562	a	138626	a	-	a	141468	a	118014	a	137708	a	143262	a

a. Wegen zu großem Speicherbedarf konnte kein Wert berechnet werden.

Für die großen Schaltungen sind die Design-Flüsse 6 und 7 nicht nur bezüglich Verdrahtbarkeit, Fläche und Schaltungsverzögerungszeit günstig, sondern auch bezüglich des Leistungsverbrauches. Dies liegt nicht nur an der geringeren Zahl der erforderlichen CLBs, sondern auch an der entspannteren Verdrahtung und den daraus resultierenden geringeren Leitungslängen im FPGA-Baustein. Dieses Ergebnis ist auch insofern erstaunlich, weil es nach Tabelle 3 so nicht zu erwarten war, aber die Entspannung des Verdrahtungsproblems hilft offenbar auch bei der Lösung dieses Problems.

7.3. Gesamtbewertung der Design-Alternativen

Design-Fluß 8 ist stets schlechter als die Design-Flüsse 6 und 7, d.h. die Verwendung von 2 LUTs, die schlecht in einem CLB untergebracht werden können (im Abschnitt 6.3 wird die

Tabelle 9: Wertung der acht Design-Flüsse

Beispiel	Design-Fluß	1	2	3	4	5	6	7	8
ohne Multiplizierer (fibo, gcd, atoi)	#CLB	++	+++	+++	++	+	++	++	++
	Delay	+++	++	+	+	+	+++	+++	+++
	CPU-Time	++	+++	+++	+++	+++	++	++	++
	Design ist verdrahtbar	+++	+++	+++	+++	+++	+++	+++	+++
	Verlustleistung	++	+++	+++	+++	++	++	++	-
mit Multiplizierer (ellip, diffeq, dct)	#CLB	++	++	+	+	-	+++	++++	+++
	Delay	++	++	+	+	+	+++	+++	+++
	CPU-Time	++	++	--	---	++	+++	+++	++
	Design ist verdrahtbar	++	+++	++	--	++	+++	+++	+++
	Verlustleistung	++	+++	+	+	-	+++	+++	++
GESAMT - Wertung		+	++	-	--	--	+++	+++	++

Notwendigkeit zusätzlicher Verdrahtung erläutert), ist ungünstiger als die Benutzung von 3 LUTs, die in einem CLB angeordnet sind (Abb. 4a).

Die Gesamtbewertung in Tabelle 9 zeigt, daß die Design-Flüsse 7 und 6 die besten Design-Flüsse darstellen, um von der Verhaltensbeschreibung zum FPGA zu gelangen. Der Design-Fluß 8 ist wieder ein Beispiel dafür, daß zu starke Flächenminimierung große Verdrahtungs-probleme hervorruft.

Tabelle 10: Verdrahtungs-Charakteristika für 2 große Schaltungen

Bsp.	Design-Fluß / Verdrahten erfolgreich?	FPGA	CLB (a)	CLB Nutz. (b)	#Netz (c)	P/N (d)	P/C (e)	Kanal dichte (f)	Kanaldichte nur Select (g)
diffeq31	Design-Fluß 1	4028EX	823	80.3%	1758	4.706	10.053	8.18	2.09
	Design-Fluß 1, nicht verdrahtbar	4025E [a]	823	80.3%	1750	4.704	10.002	8.43	2.10
	Design-Fluß 6	4025E	822	80.2%	1701	4.752	9.833	7.56	2.01
	Design-Fluß 7	4025E	822	80.2%	1695	*4.766*	9.828	7.52	2.14
	Design-Fluß 8	4025E	822	80.2%	1762	4.641	9.948	8.31	2.04
	Design-Fluß 2	4025E	858	83.8%	1812	4.710	9.946	7.61	2.16
	Design-Fluß 3	4025E	*897*	*87.6%*	1904	4.635	9.839	8.61	2.31
	Design-Fluß 4	4025E	852	83.2%	*1943*	4.529	*10.329*	*8.93*	*3.46*
	Design-Fluß 5	4036EX	1125	86.7%	2427	4.567	9.852	9.01	2.71
chendct32	Design-Fluß 1, nicht verdrahtbar	4052XL [a]	*1910*	*98.7%*	*3561*	5.293	9.868	*17.29*	10,81
	Design-Fluß 6	4052XL	1837	95.0%	2941	*5.867*	9.392	14.58	8.66
	Design-Fluß 7	4052XL	1728	89.2%	3080	5.675	10.115	12.91	6.89
	Design-Fluß 8	4052XL	1733	89.5%	3213	5.472	*10.145*	13.69	6.29
	Design-Fluß 2	4052XL	1889	97.6%	3557	5.336	9.999	16.42	*11,74*
	Design-Fluß 1	4062XL	1910	82.9%	3602	*5.268*	*9.935*	13.87	9.30
	Design-Fluß 3, nicht verdrahtbar	4062XL [a]	*2236*	*96.9%*	*4119*	5.227	9.629	*16.40*	*9,36*
	Design-Fluß 3	4085XL	2236	71.2%	4119	*5.227*	9.629	11.48	6.96
	Design-Fluß 4, nicht verdrahtbar	4085XL [a]	2373	75.5%	4366	5.142	9.477	*13.05*	8.99
	Design-Fluß 5	4085XL	*2564*	*81.9%*	*4867*	5.138	*9.754*	12.91	*9.33*

a. Routing war nicht möglich !

Entsprechend einiger Kriterien aus Abschnitt 5.1 sind einige Analysen der Entwürfe nach dem Plazieren und Verdrahten in den Spalten 4-9 der Tabelle 10 dargestellt: (a) ist die Zahl der benutzten CLBs, (b) enthält den Prozentsatz der CLB Auslastung, (c) ist die Anzahl der Netze, (d) gibt das Verhältnis der Pins zu den Netzen an, (e) das Verhältnis zwischen der Netzzahl und der CLB-Anzahl, (f) entspricht einer theoretischen Kanaldichte: die Länge der Netze ist dazu mittels eines minimalen Spannbaums berechnet, wobei die Netzpins in der Mitte der CLBs angesiedelt wurden; sie wurde dann durch die doppelte Zahl der CLBs dividiert und stellt somit ein grobes Maß für die Anzahl der Verbindungen pro Kanal dar. (g) ist wie (f) ermittelt, gibt aber nur den Anteil der Select-Signale wieder, also den Anteil der kritischen Signale/Leitungen. Die Select-Signale wurden aus den Algorithmen von Kapitel 9 gewonnen. Eine genauere Berücksichtigung der Lage, der Form der Netze sowie der Pin-Lage, erfolgte bei der Leitungslängen-Ermittlung nicht, da diese Informationen in den Xilinx-Ausgabe-Files nur codiert vorliegen.

Tabelle 10 zeigt, daß einige klassische Verdrahtungs-Kriterien nicht geeignet sind, die Verdrahtbarkeit auszudrücken, da die schlechtesten Werte je FPGA nicht in den Zeilen der 4 nicht-verdrahtbaren Beispiele konzentriert sind. Die Kanaldichte (f) scheint noch die zuverlässigste

Größe zu sein. Ein Vergleich der Spalten (f) und (g) zeigt aber auch, daß über die Hälfte der gesamten Verdrahtung die Select-Signale selbst darstellen.

Das eigentliche Problem - die Verdrahtung der Multiplexer vor den spaltenweise angeordneten kompakten Makro-Blöcken der schnellen Addierer - kann offenbar durch Spalte (g) allein doch nur schlechter widergespiegelt werden als durch Spalte (f), die immerhin eine Chip-Gesamt-Bewertung vornimmt.

Eine Reduktion der Anzahl der CLBs (Kriterium 1) ist ein übliches Ziel der Mapping-Werkzeuge, aber unsere Experimente zeigen, daß dies nicht immer vorteilhaft ist in Bezug auf die Verdrahtbarkeit. So ist Design-Fluß 8 schlechter als Design-Fluß 7 trotz der angestrebten Reduktion an LUTs. Deshalb werden auch nur die Design-Flüsse 6 und 7 weiterempfohlen.

8. Entwurf mit anderen Entwurfssystemen

8.1. Entwurf mit CADDY

Das in Abschnitt 2.2 beschriebene CADDY-Synthesesystem kann ebenso mit den in Kapitel 7 beschriebenen Algorithmen kombiniert werden, denn CADDY erzeugt als Ergebnis der High-Level-Synthese auch eine hierarchische Schaltung, bei der die Multiplexer nachträglich verändert werden können.

Eine Besonderheit von CADDY ist allerdings, daß es den Wert der Multiplexer-Ansteuersignale nicht einfach durchnumeriert, sondern den freien Werte-Bereich bis zur nächsten vollen Zweierpotenz ausnutzt und die Ansteuerwerte so optimiert, daß der Controller kleiner wird. Dadurch entstehen Multiplexer ‘mit Löchern‘ an den Dateneingängen. Diese bewirken aber, daß diese Multiplexer nicht einfach durch die flächengünstigen Multiplexer-Strukturen aus Abb. 4 und 5 ersetzt werden können, da sie scheinbar mehr Eingänge und damit auch Logikelemente enthalten, die an sich überflüssig sind, aber (vor allem bei Multiplexer-Struktur 3 in Abb. 5) bei der Logiksynthese nicht wegoptimiert werden, wodurch sich die Fläche erheblich vergrößern würde. Deshalb erfordern solche mit CADDY erzeugten Multiplexer eine besondere Behandlung, um nach dem Ersetzen die an sich überflüssige Multiplexer-Logik doch wegoptimieren zu können. Dazu wurde speziell der untenstehende Algorithmus 3 entwickelt.

Der Design-Fluß 7 beruht wie in Kapitel 7 auf Algorithmus 2, der im folgenden ebenfalls benutzte Design-Fluß 7A benutzt den Algorithmus 3, der es erlaubt, mehr Multiplexer auszuwählen und geeignet zu ersetzen.

Zum Vergleich der Ergebnisse mit PMOSS wurden die in Abschnitt 3.1 und Abschnitt 7.2.1 untersuchten Schaltungen (nach einer manuellen Umsetzung der C-Programme nach VHDL) mittels CADDY auf eine RT-Struktur abgebildet und dann in Anlehnung an die Design-Flüsse 1 und 7 (Abb. 1 und Abb. 9) sowie 7A (siehe oben) weitersynthetisiert. Die Ergebnisse sind für die sich in Abschnitt 3.1 als kritische Beispiele erwiesenen Schaltungen in Tabelle 11 und Tabelle 12 dargestellt.

Insgesamt läßt sich sagen, daß CADDY kleinere Multiplexer-Bäume als PMOSS erzeugt. Der Grund hierzu liegt wohl in der Nutzung der kombinierten Operatoren (ALUs) sowie der umfangreicheren Suche während der High-Level-Synthese.

Trotzdem vermochte auch hier der Design-Fluß 7 bzw. Design-Fluß 7A das Ergebnis zu verbessern (Fläche, Verzögerung, Xilinx-Rechenzeit, kleinerer FPGA), was ein weiterer Hinweis für die Bedeutung der erzielten Ergebnisse ist.

Tabelle 11: Experimente mit CADDY: Fläche und Verzögerung

			Design-Fluß 1 nach CADDY			Design-Fluß 7 nach CADDY			Design-Fluß 7A nach CADDY		
	#Eing.	#Ausg.	FPGA	#CLB	ns	FPGA	#CLB	ns	FPGA	#CLB	ns
diffeq31	157	93	4028EX [a]	***773***	***352.4***	siehe Design-Fluß 1			***4025E***	786	507.0
chendct32	65	46	4052XL	1633	486.6	4052XL	1586	***416.3***	4052XL	***1551***	427.2

a. Der nächst kleinere FPGA erlaubt kein Routing (Xilinx Nachricht nach 2:47:48 !!).

Tabelle 12: Experimente mit CADDY: Rechenzeiten

	Design-Fluß 1		Design-Fluß 7		Design-Fluß 7A	
	KSS	Xilinx	KSS	Xilinx	KSS	Xilinx
diffeq31	28:43	[a] 6:23	siehe Design-Fluß 1		32:17	11:57
chendct32	3:01:07	14:49	3:02:00	13:54	3:02:18	***13:15***

a. Der nächst kleinere FPGA erlaubt kein Routing (Xilinx Nachricht nach 2:47:48 !!)

Die Grundlage für den Design-Fluß 7A bildet der folgende Algorithmus 3. Hier werden 3 Gruppen von Multiplexern gebildet: die Multiplexer, die genau wie im Fluß 7 ersetzt werden (Gruppe 1), die Multiplexer, die durch Schaltungen gemäß Abb. 4 ersetzt werden, wobei aber jeder von CADDY nicht benutzte Dateneingang des Multiplexers zur Einsparung genau eines 2:1-Multiplexers im Multiplexer-Baum führt (Gruppe 2), sowie die Multiplexer, die nicht verändert werden und so ihre Struktur von Abb. 3 behalten.

// Algorithmus 3 Mapping_for_Routability_CADDY
begin
0: High-Level-Synthese mit CADDY
1: Ausschneiden aller Multiplexer mit mindestens 4 Eingängen, sofern an den Multiplexer-Eingängen nicht vorrangig Konstanten bzw. Shift-Operationen liegen und auch keine auf ‘X‘ gesetzte Signale. → *Gruppe 1*
2: Ausschneiden aller restlichen Multiplexer mit
$numOfInputs \in Set2 = \{4, 5, 8, 10, 11, 13, 14, 16, 17\}$,
sofern an den Multiplexer-Eingängen auf ‘X‘ gesetzte Signale, aber keine Konstanten bzw. Shift-Operationen liegen. → *Gruppe 2*
3: FPGA- Mapping der restlichen und ausgeflachten Schaltung.
4: Für *Gruppe 1*: Hinzufügen logisch äquivalenter Multiplexer mit der speziellen Struktur 3 (ab 5 Eingängen) bzw. Struktur 2 (4 Eingänge)
5: Für *Gruppe 2*: Hinzufügen logisch äquivalenter Multiplexer mit der Struktur 2 bei max. Reduktion entsprechend der ‘X‘-Signale
6: Xilinx Tools für Placement und Routing
end

8.2. Anwendung auf ein RT-Design ohne High-Level-Synthese

In einer auf RT-Niveau entworfenen Schaltung treten die beschriebenen Effekte in ähnlicher Form auf, d.h. die verschachtelten IF- und CASE-Anweisungen auf RT-Niveau werden auf Multiplexer abgebildet. Dabei werden vom kommerziellen Synthesesystem wieder die Multiplexer der Form 1 (Abb. 3) bevorzugt. D.h. die entwickelten Ansätze können auch für diese Entwürfe das Verdrahtungsproblem vereinfachen.

Hierbei tritt aber ein Problem auf: der Vorteil einer zusätzlichen Hierarchiestufe, die durch die High-Level-Synthese entsteht, wodurch alle Operatoren und Multiplexer als eigenständige Blöcke erzeugt werden, tritt bei einem RT-Design leider nicht ein. Die aus den CASE- und IF-Anweisungen erzeugten symbolischen Multiplexer werden im Syntheseprozeß sofort implizit ausgeflacht und stehen also für die oben beschriebenen Manipulationen nicht (mehr) zur Verfügung. Dieses automatische Ausflachen konnte bei den bisherigen Experimenten leider nicht beeinflußt werden. Die beiden verbleibenden Auswege

(a) eine nachträgliche Manipulation auf XILINX-Netzlisten-Ebene, oder

(b) ein tiefgreifender Eingriff in die Bibliotheks-Struktur des kommerziellen Synthese-Systems

wurden vorerst nicht weiter verfolgt. Für (a) wäre an Hand des Cluster-Algorithmus (Kapitel 9) eine Multiplexer-Umformung denkbar, was aber auch noch eine Rückerkennung und logische Erzeugung der ursprünglichen Select-Signale erfordern würde und angesichts der beobachteten strukturellen Unregelmäßigkeiten einen sehr großen Aufwand darstellen würde. Die Variante (b) wäre eigentlich die effektivste Stelle zur Beeinflussung, sie gestaltet sich aber auch schwieriger als erwartet, da es sich nicht nur um eine Bibliotheksfrage, sondern (auch) um eine interne Synthesestrategie-Frage im Synthesesystem zu handeln scheint.

9. Analyse von Multiplexern

Eine Analyse der Multiplexer in einer auf FPGA abgebildeten Schaltung ist aus zwei Gründen wünschenswert:

- Zum einen ermöglicht sie die Analyse der Entwurfsergebnisse, bei der es darauf ankommt, die Baublöcke der RT-Ebene im Layout des FPGA nach Möglichkeit wiederzuerkennen, um daraus Schlußfolgerungen zu gewinnen bezüglich
  - der Güte des Entwurfes,
  - möglicher Schwachpunkte oder
  - bezüglich weiterer Einsparpotentiale,
- Zum anderen stellt sie die Vorbereitung für die im Abschnitt 8.2 unter (a) erwähnten Umformung bei der Synthese der Schaltung dar.

Will man also bei einer auf FPGA abgebildeten Schaltung analysieren, welche Multiplexer darin enthalten sind, so ergeben sich einige Probleme, da der ideale und regelmäßige Multiplexer-Aufbau (in einer Richtung baumartige Datenleitungen, quer dazu die Steuerleitungen), der eine verhältnismäßig leichte Rückerkennung ermöglichen würde, in der Praxis oft gestört auftritt:

- Konstanten an den Multiplexer-Eingängen werden in den Multiplexer „hineinoptimiert“ und verkleinern je nach ihrem Wert den Multiplexer,
- Operanden unterschiedlicher Wortlänge zerstören ebenfalls die Regelmäßigkeit.
- Ferner stellen die Grundschaltungen aus Abschnitt 6.1/Abb. 3 ein Problem dar, weil dort die Steuer-Eingänge nicht an ihrer logischen Funktion erkennbar sind, sondern mit den Dateneingängen verwechselt werden können.

Um diese Schwierigkeiten zu berücksichtigen, wurde ein Algorithmus zur Multiplexer-Rückerkennung entwickelt. Er besteht aus der LUT-Klassifikation (Abschnitt 9.1) und der eigentlichen Cluster-Bildung (Abschnitt 9.2).

9.1. Charakterisierung von Multiplexer-Bestandteilen

In diesem Abschnitt wird beschrieben, wie entschieden werden kann, ob eine boolesche Funktion - nach der Abbildung auf FPGA - einen Multiplexer (oder einen Teil eines Multiplexers) darstellt. Hierzu wurden 2 Kriterien entwickelt, die sich als sehr wirkungsvoll erwiesen, d.h. sie sind in der Lage, die in diesem Kapitel eingangs erwähnten Problempunkte zu berücksichtigen.

Kriterium 1:

Ein Multiplexer kann durch ein 5-Tupel erkannt werden:

$$\left(\|f\|, \left\|\frac{\partial f}{\partial x_1}\right\|, \left\|\frac{\partial f}{\partial x_2}\right\|, \left\|\frac{\partial f}{\partial x_3}\right\|, \left\|\frac{\partial f}{\partial x_4}\right\| \right),$$

d.h. die Anzahl der Einsen für die LUT-Funktion sowie ihrer 4 booleschen Ableitungen ergibt als Kombination einen Fingerabdruck der logischen Funktion.

Beispiele hierzu finden sich in Tabelle 13 im Anhang. Die letzte Spalte dort gibt an, ob aus dem 5-Tupel (bis auf Invertierung der Eingänge) eindeutig auf die Funktion geschlossen werden kann.

Kriterium 2:

Aus der Disjunktheit zweier Ableitungen einer LUT-Funktion f kann auf die Dateneingänge eines Multiplexers geschlossen werden:

$$\left((\forall x_1)(\forall x_2)(\forall x_3)(\forall x_4)\left(\frac{\partial f}{\partial x_1} \cdot \frac{\partial f}{\partial x_2} \equiv 0\right)\right) \Rightarrow \text{f ist meist ein 2:1-Multiplexer und } x_1 \text{ und } x_2 \text{ sind Daten-Eingänge}$$

Eine genaue Analyse der Funktionen, die Kriterium 1 erfüllen, sowie ihre Interpretation als Multiplexer ist im Anhang in Tabelle 15 dargestellt. Obwohl nur 30% dieser Funktionen als 2:1-Multiplexer gedeutet werden können, so zeigten die praktischen Tests bei den Beispielschaltungen, daß die anderen 70% fast nicht vorkommen, und somit Kriterium 2 doch eine gute Abgrenzung zwischen Multiplexern und sonstigen Logik-Funktionen erlaubt.

Kriterium 2 beruht im wesentlichen auf dem folgenden Satz:

Satz: Für jede Funktion f mit 4 Variablen sind folgende 2 Aussagen (A) und (B) äquivalent:

(A) f hat 2 zueinander disjunkte Ableitungen (vergl. Kriterium 2):

$$\left((\forall x_1)(\forall x_2)(\forall x_3)(\forall x_4)\left(\frac{\partial f}{\partial x_1}\cdot\frac{\partial f}{\partial x_2}\equiv 0\right)\right)$$

(B) f hat folgende Struktur:

$$f = f_1\cdot x_1 + f_2\cdot x_2 + f_3\cdot \bar{x}_1 + f_4\cdot \bar{x}_2 + f_5\cdot 0 + f_6\cdot 1 \qquad \text{(Gl.1)}$$

wobei die 6 Koeffizienten-Funktionen $f_i = f_i(x_3,x_4)$ für $i = (1, \ldots,6)$ disjunkt zueinander sind, d.h.:

$$(\forall x_3)(\forall x_4)(\forall i)(\forall j):((i \neq j) \Rightarrow (f_i \cdot f_j \equiv 0)) \qquad \text{(Gl.2)}$$

Der Beweis des Satzes und die Diskussion der beiden Kriterien erfolgen im Anhang.

Mit den Kriterien können alle Multiplexer mit 2 Dateneingängen erkannt werden. Aber nicht alle ermittelten Teilschaltungen sind - wie oben schon erwähnt - auch Multiplexer: auf Grund einiger - zum Teil allerdings „pathologischer“ - Ausnahmen. Dies wird aber auch im Anhang diskutiert. (Um die Lesbarkeit dieses Kapitels nicht zu beeinträchtigen, wurden die notwendigen Beweise und Details in den Anhang ausgelagert.) Für die praktische Analyse von synthetisierten Schaltungen stören diese Ausnahmen aber nicht, da sich die Cluster-Bildung (nächster Abschnitt) als weiteres Kriterium anschließt.

9.2. Clusterung von Logikblöcken zu Multiplexern großer Wortbreite

Mit den in Abschnitt 9.1 vorgestellten Kriterien können auf Wortebene noch keine ganzen Multiplexer, sondern nur ihre Teilstücke - bestehend jeweils aus einem LUT - erkannt werden. Die Zusammenfassung geschieht in diesem Abschnitt.

Bei diesem Algorithmus wird versucht, auch die Mehrfachverwendung von Select-Signalen sowie die Unregelmäßigkeiten durch Operanden unterschiedlicher Wortlänge an den Dateneingängen einiger Multiplexer zu behandeln. Für die praktischen Schaltungen aus den bisherigen Kapiteln konnten die meisten in Schritt 1 des folgenden Algorithmus „MultiplexerClusterBildung“ klassifizierten LUTs tatsächlich zu Multiplexern großer Wortbreite zusammengefaßt werden.

Im Schritt 4 und 5 des Algorithmus wird versucht, von einem LUT, der Kriterium 1 oder 2 erfüllt hat, ausgehend durch Vorwärts- und Rückwärtssuche weitere LUTs zu finden, die Kriterium 1 oder Kriterium 2 erfüllen und eine baumartige Struktur bilden. Dabei wird in Daten-, Select- und Semiselect-Signale unterschieden. Als Semiselect-Signale werden dabei im folgenden die symmetrischen Eingänge der Multiplexer-Variante 1 (Abb. 3) bezeichnet, bei denen erst aus der Umgebungs-Information klar wird, was Daten und was Steuer-Signale sind.

Diese Semiselect-Signale sind auch der Grund für die Prioritäts-Auswahl in Schritt 7 und 9. Dadurch können die Daten-Signale, die zu wenigen aber unterschiedlich großen 1-Bit-Multiplexern gehen, von den Steuersignalen, die (entsprechend der Wortbreite) zu vielen, aber ziemlich gleichgroßen 1-Bit-Multiplexern gehen, getrennt werden. Die Funktion $Laengengleichheit(lut(s))$ bewertet dabei eine möglichst große Gruppe homogener 1-Bit-Multiplexer, zu denen Signal s geht, höher als eine Gruppe unterschiedlich großer 1-Bit-Multi-

plexer. Die am Anfang von Kapitel 9 genannten Gründe für Unregelmäßigkeiten in der Multiplexer-Struktur werden weiterhin durch die Schritte 11 und 12 berücksichtigt. Enthält die Gruppe von 1-Bit-Multiplexern, die zu s_2 gehören, weniger Elemente als die zu s_1 gehörende Gruppe, so wird Select-Signal s_2 vollständig zum Cluster C hinzugenommen. Andernfalls verbleiben die überzähligen 1-bit-Multiplexer bei den bisher nicht zugeordneten LUTs.

```
// Algorithmus  MultiplexerClusterBildung
begin
// Klassifikation aller LUTs der FPGA-Schaltung
 1.  L = { alle LUT gemäß Abschnitt 6.1 bis Abschnitt 6.3 : per Kriterium 1 } +
           + { alle LUT nach Kriterium 2 }
 2.  M_1 = ∅, S = ∅
// Cluster-Bildung der 1-Bit-Multiplexer
 3.  while ( L ≠ ∅ ) {
 4.    von l ∈ L aus beidseitige Suche nach allen LUTs eines 1-bit-Mux: seien L_1 ⊆ L
 5.    L = L − L_1,   M_1 = M_1 ∪ {L_1},   S = S ∪ select(L_1) ∪ semiselect(L_1)
     }
// Cluster-Bildung aller n-bit-Multiplexer
 6.  M_n = ∅
 7.  ∀s ∈ S        priority(s) = Laengengleichheit(lut(s))
 8.  while ( S ≠ ∅ ) {
 9.    sei s_1 ∈ S das s aus S mit höchster Priorität priority(s_1); C = {s_1}
10.    ∀s_2 ∈ S {
11.      if ( lut(s_2) ⊆ lut(s_1) ) { C = C ∪ {s_2}; S = S − {s_2} }
12.      if ( lut(s_2) ⊇ lut(s_1) ) { C = C ∪ {s_2}; lut(s_2) = lut(s_2) − lut(s_1);
13.                                   aktualisiere priority(s_2) }
       }
14.    M_n = M_n ∪ {C} ; Ausgabe: Cluster C, lut(s_1)
     }
end
```

Einige Ergebnisse der Cluster-Bildung sind in der letzten Spalte von Tabelle 10 dargestellt.

10. Schlußfolgerung und Zusammenfassung

Bei der Abbildung von Verhaltensbeschreibungen auf FPGAs wurden verschiedene Design-Flüsse untersucht, die die Nutzung kommerzieller Programme einschließen.

Dabei zeigte sich, daß die Frage der Verdrahtbarkeit bisher nicht ausreichend berücksichtigt worden ist. Nach einer Untersuchung verschiedener Design-Flüsse konnten Schwachpunkte analysiert und diese durch die Entwicklung neuer Entwurfsstrategien abgebaut werden.

Zu den üblichen Entwurfszielen der Flächen- und/oder Verzögerungszeit-Reduktion wurde die Verdrahtbarkeit hinzugenommen und es entstand ein ergänzendes Werkzeug zu einem kommerziellen Synthesesystem, das einen durchgängigen Design-Fluß ermöglicht und dabei bessere Entwurfsergebnisse liefert. Das heißt, daß sowohl die Rechenzeit (z.T. sogar drastisch) verkürzt als auch die Fläche, Verzögerung und Verlustleistung der generierten FPGAs erheblich verbessert werden konnten. Die vorgestellten Design-Flüsse 6 und 7, denen die entwickel-

ten und vorgestellten Algorithmen 1 und 2 zugrundeliegen, berücksichtigen durch gezielte Einflußnahme schon während des Mapping-Prozesses die Verdrahtbarkeit und können damit später auftretende Plazierungs- und Verdrahtungs-Probleme von vornherein entschärfen, wodurch kleinere, schnellere und verlustleistungsärmere FPGA-Schaltungen erzielt werden.

11. Literatur

11.1. Eigene Arbeiten

[1] H.-J. Eikerling, W. Hardt, J. Gerlach, W. Rosenstiel. „A Methodology for Rapid Analysis and Optimization of Embedded Systems“. In Symposium on Engineering of Computer Based Systems, Friedrichshafen, Germany, March 1996.

[2] J. Gerlach, H.-J. Eikerling, W. Hardt, W. Rosenstiel. „Von C nach Hardware: ein integratives Entwurfskonzept“. 1. GI/ITG/GMM Workshop Allgemeine Methodik von Entwurfsprozessen, Paderborn, Germany, March 1996.

[3] W. Hardt, J. Gerlach, H.-J. Eikerling, W. Rosenstiel, B. Gregory. „Using the HDL-Advisor for Shortening the System Design Loop“. SNUG ‘96, Geneva, Switzerland, Sept. 1996.

[4] J. Gerlach, W. Rosenstiel. „A Scalable Methodology for Cost Estimation in a Transformational High-Level Design Space Exploration Environment“. Design, Automation and Test in Europe Conference and Exhibition (DATE ‘98), Paris, Febr. 23-26, 1998, pp. 226-231.

[5] H.-J. Eikerling, W. Rosenstiel. „Automatic Structuring and Optimization of Hierarchical Designs“. EURO-DAC ‘96, Geneva, Switzerland, Sept. 16-20, 1996, pp. 134-139.

[6] H.-G. Martin, W. Rosenstiel. „A Comparing Study of Technology Mapping for FPGA“. Design, Automation and Test in Europe Conference and Exhibition (DATE ‘98), Paris, Febr. 1998.

[7] K. Weiß, R. Kistner, A. Kunzmann, W. Rosenstiel. „Analysis of the XC6000 Architecture for Embedded System Design“. IEEE Symposium on Field-Programmable Custom Computing Machines FCCM ‘98, Napa (CA), 1998.

[8] P. Gutberlet, W. Rosenstiel. „Scheduling Between Basic Blocks in the CADDY Synthesis System“. EDAC ‘92, Brüssel, Belgien, March 16-19, 1992 , pp. 496-500.

[9] P. Gutberlet, J. Müller, H. Krämer, W. Rosenstiel. „Automatic Module Allocation in High Level Synthesis“. EURO-DAC ‘92/EURO-VHDL ‘92, Hamburg, 7.-10. September, 1992, pp. 328-333.

[10] O. Bringmann, W. Rosenstiel. „Resource Sharing in Hierarchical Synthesis“. ICCAD ‘97, San José, CA, USA, Nov., 9-13, 1997, pp. 318-325.

[11] O. Bringmann, W. Rosenstiel. „Cross-Level Hierarchical High-Level Synthesis“. Design, Automation and Test in Europe Conference and Exhibition (DATE ‘98), Paris, Febr. 23-26, 1998. pp. 451-456.

[12] T. Kuhn, W. Rosenstiel, U. Kebschull. „Description and Simulation of Hardware/Software Systems with Java“. 36. DAC, 1999, pp. 790-793.

[13] H.-G. Martin, W. Rosenstiel. „FPGA-Design unter besonderer Berücksichtigung von Multiplexer-Struktur und Verdrahtbarkeit“. Report WSI-99-17, Universität Tübingen, 1999.

[14] J. Bullmann, E. Schubert, U. Kebschull, W. Rosenstiel. „Library Based Technology Mapping Using Multiple Domain Representations“. EURO-DAC ‘96, Geneva, Switzerland, Sept. 16-20, 1996, pp. 146-151.

[15] W. Lange, P. Thole, W. Rosenstiel. „Modellierung einer ATM-Switch-Steuerung mit anwendungsspezifischen Funktionen“. GI/ITG-Workshop „Custom Computing“, ed. D. Monjau, Schloß Dagstuhl, 18.-21.05.96, pp. 73-81.

[16] W. Dreher, H.-G. Martin, W. Rosenstiel. „Das Weaver-II-Board als neue HW/SW-Codesign Plattform“. SDA ‘99, Rathen, Germany, April 19-20, 1999.

[17] K. Weiß, T. Steckstor, G. Koch, W. Rosenstiel. „Exploiting FPGA-Features during the Emulation of a Fast Reactive Embedded System“. FPGA ‘99, Monterey, CA, USA, February 18-23, 1999.

11.2. Weitere Referenzen

[18] R.J. Francis, J. Rose, and Z. Vranesic. „Technology Mapping for Lookup Table-Based FPGAs for Performance“. ICCAD, 1991, pp. 568-571.

[19] R. Murgai, N. Shenoy, R.K. Brayton, A. Sangiovanni-Vincentelli. „Improved Logic Synthesis Algorithms for Table Look Up Architectures“. ICCAD, 1991, pp. 564-567.

[20] J. Cong and Y. Ding. „FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs“. IEEE Trans. on Computer-Aided Design, 13:1-12, 1994.

[21] J. Cong and Y.-Y. Hwang. „Simultanious Depth and Area Minimization in LUT-Based FPGA Mapping“. Proc. ACM 3rd Int‘l Symp. on FPGA, Feb. 1995, pp. 68-74.

[22] C. Legl, B. Wurth, and E. Eckl. „A Boolean Approach to Performance-Directed Technology Mapping for LUT-Based FPGA Design“. 33rd DAC, 1996, pp. 730-733.

[23] P. Pan and C.L.Liu. „Optimal Clock Period FPGA Technology Mapping for Sequential Circuits“. 33. DAC 1996, paper 45.1.

[24] J. Cong and C. Wu. „FPGA Synthesis with Retiming and Pipelining for Clock Period Minimization of Sequential Circuits“. DAC 1997, pp. 644-649.

[25] J. Cong, Y.Y. Hwang. „Partially-Dependent Functional Decomposition with Applications in FPGA Synthesis and Mapping“. FPGA ‘97, Febr. 97, pp. 35-42.

[26] J. Cong and S. Xu. „Delay-Optimal Technology Mapping for FPGAs with Heterogeneous LUTs“. 35. DAC 1998, paper 43.2, pp. 704-707.

[27] C. Schneider et. all. „From Algorithms to Hardware Architecture: A Comparison of Regular and Irregular Structured IDCT Algorithms“. DATE ‘98, pp. 186-190.

[28] M.A.S. Khalid and J. Rose. „The Effect of Fixed I/O Pin Positioning on the Routability and Speed of FPGAs“. Technical Report CSRI-325, University Toronto, May 1995.

[29] S.B. Brown, J. Rose, Z.G. Vranesic. „A Stochastic Model to Predict the Routability of Field-Programmable Gate Arrays“. IEEE Tr. on CAD, Vol. 12, Dec. 1993, pp. 1827-1838.

[30] Y.L. Wu, D. Chang, M. Marek-Sadowska, and S. Tsukiyama. „Not Necessarily More Switches More Routability“. ASP-DAC 1997, Tokyo, pp. 579-584.

[31] M. Schlag, J. Cong, and P.K. Chan. „Routability-driven technology mapping for lookup table-based FPGA‘s“. IEEE Trans. on CAD, 13:13-26, 1994.

[32] P. K. Chan, M.D.F. Schlag, and J.Y. Zien. „On Routability Prediction for Field-Programmable Gate Arrays“. 30. DAC 1993, pp. 326-330.

[33] N. Bhat and D. Hill. „Routable technology mapping for LUT FPGA‘s“. Int. Conf. on Computer Design: VLSI in Computers and Processors. Oct 1992, pp. 95-98.

[34] Y. Takashima, A. Takahashi, Y. Kajitani. „Detailed-Routability of FPGAs with Extremal Switch-Block Structures“. ED&TC 1996, paper 4B.1.

[35] D. Bhatia, A. Chowdhary, S. Tragoudas. „Mathematical Model for Routability Analysis of FPGAs“. Great Lake Symposium on VLSI, March 1994, pp. 76-79.

[36] H. Vaishnav, M. Pedram. „Routability-Driven Fanout Optimization“. 30. DAC 1993, pp. 230-235.

[37] S.K. Nag, K. Roy. „Iterative Wirability and Performance Improvement for FPGAs“. 30. DAC 1993, pp. 321-325.

[38] Xilinx Inc., San Jose, CA-95125. „The Programmable Logic Data Book“, 1994.

[39] E.M. Sentowich, K.J. Singh et al. „SIS: A System for Sequential Circuit Synthesis“. Documentation included in the SIS programme package.

[40] Xilinx Inc., San Jose, CA. „XILINX Application Note: Fundamentals of Placement and Routing“. 1990, p. 34.

[41] T.A. Osmulski. „Implementation and Evaluation of a Power Prediction Model for a Field Programmable Gate Array“. A Masters Thesis Proposal. Department of Computer Science, Texas Tech. University. URL: http://hpcl.cs.ttu.edu/~tim/thesis/.

[42] Xilinx Inc. „A Simple Method of Estimating Power in XC4000XL/EX/E FPGAs“. Documentation. URL: http://www.xilinx.com/xbrf/xbrf014.pdf.

[43] Xilinx Inc. „Power Consumption Estimation utility for 2k, 3k and 4k (non e/ex/xl/xv) parts“. URL: ftp://ftp.xilinx.com/pub/utilities/fpga/power.zip.

[44] J.-M. Hwang, F.-Y. Chiang, T.T. Hwang. „A Re-engineering Approach to Low Power FPGA Design Using SPFD“. DAC 1998, pp. 722-725.

[45] C.-S. Chen, T.T.Hwang, C.L.Liu. „Low Power FPGA Design - A Re-engineering Approach“. DAC 1997, pp. 656-661.

[46] B. Kumthekar, L. Benini, E. Macii, F. Somenzi. „In-Place Power Optimization for LUT-Based FPGAs“. DAC 1998, pp. 718-721.

[47] EPIC Technology Group. „PowerMill Datasheet“. http://www.synopsys.com/products/etg/powermill_ds.html.

[48] Synopsys, Inc., 700 East Middlefield Road, Mountain View, CA 94043-4033 USA. „Power Products Reference Manual“. Synopsys Online Documentation. 1997.

[49] N.W. Bergmann and P.R. Sutton. „A High-Performance Computing Module for a Low Earth Orbit Satellite using Reconfigurable Logic“. FPL ‘98. Tallinn 1998.

[50] P. Alfke, R. Padovani. „Radiation Tolerance of High-Density FPGAs“. 1998 Military and Aerospace Applications of Programmable Devices and Technologies Conference (MAPLD 1998 Conference). URL: http://rk.gsfc.nasa.gov/richcontent/Ksymposium/Preliminary%20Program.htm.

[51] NASA project (Xilinx FPGA for earth orbit satellites). URL: http://www.pcs.cnu.edu/~rhodson/fpga.html

[52] M. Wannemacher. „Das FPGA-Kochbuch“. Int. Thomson Publishing, 1998, p. 130.

Anhang

Kriterium 1:

Ein Multiplexer kann durch ein 5-Tupel erkannt werden:

$$\left(\|f\|, \left\|\frac{\partial f}{\partial x_1}\right\|, \left\|\frac{\partial f}{\partial x_2}\right\|, \left\|\frac{\partial f}{\partial x_3}\right\|, \left\|\frac{\partial f}{\partial x_4}\right\| \right),$$

d.h. die Anzahl der Einsen für die LUT-Funktion sowie ihrer 4 booleschen Ableitungen ergibt als Kombination oft einen eindeutigen Fingerabdruck der logischen Funktion.

Beispiele hierzu finden sich in Tabelle 13. Die letzte Spalte gibt dabei an, ob aus dem 5-Tupel (bis auf Invertierung der Eingänge) eindeutig auf die Funktion geschlossen werden kann.

Tabelle 13: 5-Tupel für einige Multiplexer-Formen

Bemerkung	Formel	5-Tupel	Anzahl	Eind.
LUT-Grundschaltung eines Multiplexer aus Abb. 3	f = e1*x + e2*y	(7,6,6,6,6)	3*16	ja
LUT des Ausgangsknoten eines Multiplexers mit 3 und mehr Eingängen, Abb. 3	f = e1 * x + m + n	(13,2,2,6,6)	1*16	ja
	f = m + n	(12,8,8,0,0)	1*4	ja
	f = m + n + o	(14,4,4,4,0)	1*8	ja
	f = m + n + o + p	(15,2,2,2,2)	1*16	ja
normaler 2:1-Multiplexer, Abb. 4	f = ~s*x + s*y	(8,8,8,8,0)	3*8 + 8	nein
2:1-Multiplexer mit in den LUT hereingezogener zusätzlicher Logik, z.B. UND-Gatter	z.B. f = (~s*x + s*y) * u	z.B. (4,4,4,4,8)		nein
Multiplexer aus Abb. 5	f = s0*s1 + s0*y + ~s0*~s1*x	(8,12,8,4,4)	2*16	ja

Beweis von Kriterium 1:

Der Beweis ergibt sich aus einer rechnergestützten Analyse aller 65536 Funktionen mit max. 4 Eingängen.

In Tabelle 13 ist in der letzten Spalte die Anzahl der 5-Tupel innerhalb der 65536 Funktionen angegeben: Die Zerlegung in zwei Faktoren (Variablenpermutation * Variablenparität) zeigt, daß für spezielle Schaltungen die Kennung eineindeutig ist.

Ausnahme: der normale 2:1-Multiplexer. Dort gibt es 8 falsche Kandidaten. Ebenso ist die Verschmelzung mit Nachbargattern nicht mehr sauber zu erkennen.

Dieser Nachteil von Kriterium 1 kann durch Kriterium 2 behoben werden.

Kriterium 2:

Aus der Disjunktheit zweier Ableitungen einer LUT-Funktion f kann auf die Dateneingänge eines Multiplexers geschlossen werden:

$$\left((\forall x_1)(\forall x_2)(\forall x_3)(\forall x_4)\left(\frac{\partial f}{\partial x_1}\cdot\frac{\partial f}{\partial x_2}\equiv 0\right)\right)\Rightarrow \text{f ist meist ein 2:1-Multiplexer}$$

und x_1 und x_2 sind Daten-Eingänge

Eine genaue Analyse der Funktionen, die Kriterium 1 erfüllen, sowie ihre Interpretation als Multiplexer ist in Tabelle 15 dargestellt. Obwohl nur 30% dieser Funktionen als 2:1-Multiplexer gedeutet werden können, so zeigten die praktischen Tests bei den Beispielschaltungen, daß die anderen 70% fast nicht vorkommen, und somit Kriterium 2 doch eine gute Abgrenzung zwischen Multiplexern und sonstigen Logik-Funktionen erlaubt.

Vor der Diskussion von Kriterium 2 wird zuerst folgender Satz bewiesen:

Satz: Für jede Funktion f mit 4 Variablen sind folgende 2 Aussagen (A) und (B) äquivalent:

(A) f hat 2 zueinander disjunkte Ableitungen (vergl. Kriterium 2):

$$(\forall x_1)(\forall x_2)(\forall x_3)(\forall x_4)\left(\frac{\partial f}{\partial x_1}\cdot\frac{\partial f}{\partial x_2}\equiv 0\right)$$

(B) f hat folgende Struktur:

$$f = f_1\cdot x_1 + f_2\cdot x_2 + f_3\cdot \bar{x}_1 + f_4\cdot \bar{x}_2 + f_5\cdot 0 + f_6\cdot 1 \qquad \text{(Gl.3)}$$

wobei die 6 Koeffizienten-Funktionen $f_i = f_i(x_3,x_4)$ für $i = (1,\ldots,6)$ disjunkt zueinander sind:

$$(\forall x_3)(\forall x_4)(\forall i)(\forall j):((i\neq j)\Rightarrow(f_i\cdot f_j\equiv 0)) \qquad \text{(Gl.4)}$$

Beweis des Satzes: (A) -> (B):

Jede Funktion mit max. 4 Variablen läßt sich in folgender Form darstellen:

$$f = f_{00}\cdot\bar{x}_1\cdot\bar{x}_2 + f_{01}\cdot\bar{x}_1\cdot x_2 + f_{10}\cdot x_1\cdot\bar{x}_2 + f_{11}\cdot x_1\cdot x_2 \qquad \text{(Gl.5)}$$

(wobei die 4 Subfunktionen $f_{ij} = f_{ij}(x_3,x_4)$ von x_3 und x_4 abhängen.)

Dann wird:

$$\frac{df}{dx_1} = f|_{x_1=0}\oplus f|_{x_1=1}$$
$$= (f_{00}\cdot\bar{x}_2 + f_{01}\cdot x_2)\oplus(f_{10}\cdot\bar{x}_2 + f_{11}\cdot x_2)$$
$$= (f_{00}\cdot\bar{x}_2\oplus f_{01}\cdot x_2)\oplus(f_{10}\cdot\bar{x}_2\oplus f_{11}\cdot x_2)$$
$$= (\bar{x}_2\cdot(f_{00}\oplus f_{10}))\oplus(x_2\cdot(f_{01}\oplus f_{11}))$$
$$= (\bar{x}_2\cdot(f_{00}\oplus f_{10})) + (x_2\cdot(f_{01}\oplus f_{11}))$$

Analog wird:

$$\frac{df}{dx_2} = (\bar{x}_1 \cdot (f_{00} \oplus f_{01})) + (x_1 \cdot (f_{10} \oplus f_{11}))$$

Somit läßt sich die Disjunktheit der beiden Ableitungen wie folgt umwandeln:

$$\begin{aligned} 0 &= \frac{\partial f}{\partial x_1} \cdot \frac{\partial f}{\partial x_2} \\ &= ((\bar{x}_2 \cdot (f_{00} \oplus f_{10})) + (x_2 \cdot (f_{01} \oplus f_{11}))) \cdot ((\bar{x}_1 \cdot (f_{00} \oplus f_{01})) + (x_1 \cdot (f_{10} \oplus f_{11}))) \\ &= \overline{x_2} \cdot \overline{x_1} \cdot (f_{00} \oplus f_{10}) \cdot (f_{00} \oplus f_{01}) + \overline{x_2} \cdot x_1 \cdot (f_{00} \oplus f_{10}) \cdot (f_{10} \oplus f_{11}) + \\ &\quad + x_2 \cdot \overline{x_1} \cdot (f_{01} \oplus f_{11}) \cdot (f_{00} \oplus f_{01}) + x_2 \cdot x_1 \cdot (f_{01} \oplus f_{11}) \cdot (f_{10} \oplus f_{11}) \end{aligned}$$

Da letztere Gleichung für alle x_1 und x_2 gelten soll, ergeben sich 4 neue Bestimmungsgleichungen, die gleichzeitig gelten:

$$0 = (f_{00} \oplus f_{10}) \cdot (f_{00} \oplus f_{01})$$

$$0 = (f_{00} \oplus f_{10}) \cdot (f_{10} \oplus f_{11})$$

$$0 = (f_{01} \oplus f_{11}) \cdot (f_{00} \oplus f_{01})$$

$$0 = (f_{01} \oplus f_{11}) \cdot (f_{10} \oplus f_{11})$$

Durch Fallunterscheidung erhält man nun 6 mögliche Kombinationen für die 4 f_{ij}, und zwar die Werte in den ersten 4 Spalten von Tabelle 14(a):

Tabelle 14: **a)**

Wertekombination der 4 f_{ij}			
f_{00}	f_{01}	f_{10}	f_{11}
0	0	1	1
0	1	0	1
1	1	0	0
1	0	1	0
0	0	0	0
1	1	1	1

b)

Wert Gl. 5
f =
x_1
x_2
$\bar{x}_1$
$\bar{x}_2$
0
1

c)

Resultierende Werte im Ansatz Gl. 3					
f_1	f_2	f_3	f_4	f_5	f_6
1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

Jede Zeile in Tabelle 14 entspricht einer möglichen Wertekombination der f_{ij}. Diese können - in Gl. 5 eingesetzt - so zusammengefaßt werden, daß sich für f spezielle Werte ergeben. (zweiter Block in Tabelle 14(b)). Vergleicht man diese Werte nun mit dem Ansatz aus Gl. 3, so sieht man, daß Ansatz Gl. 3 berechtigt ist, und aus dem dritten Block von Tabelle 14(c) ersieht man dann auch: es gilt sogar die Nebenbedingung Gl. 4.

Die Rückrichtung (B) -> (A) ist einfach zu zeigen und wird hier weggelassen.

Damit ist der Satz bewiesen.

■

Bemerkung 1: Gibt es zwei Paare (x_1, x_2) und (x_1, x_3) mit disjunkten Ableitungen, dann konkretisiert sich Gl. 3 zu:

$f = f_1(x_4) \cdot x_1 + f_3(x_4) \cdot \bar{x}_1 + f_2(x_3,x_4) \cdot x_2 + f_4(x_3,x_4) \cdot \bar{x}_2 + f_6(x_3, x_4)$

und dies läßt sich - bis auf invertierte Eingänge und entartete Fälle - wegen Gl. 4 auf die Struktur von Gl. 6 abbilden:

$$f \approx g(x_2, x_3) \cdot x_4 + x_1 \cdot \bar{x}_4 \qquad \text{(Gl.6)}$$

Bemerkung 2: Gibt es aber zwei Paare (x_1, x_2) und (x_3, x_4) mit disjunkten Ableitungen, dann konkretisiert sich Gl. 3 zu der Schaltung des folgenden Typs, die aber kaum mehr als Multiplexer deutbar ist:

$$f \approx \bar{x}_3 \cdot \bar{x}_4 \cdot x_1 + x_3 \cdot \bar{x}_4 \cdot x_2 + x_3 \cdot x_4 \cdot \bar{x}_1 + \bar{x}_3 \cdot x_4 \cdot \bar{x}_2$$
$$= \bar{x}_1 \cdot x_2 \cdot x_3 + \bar{x}_1 \cdot \bar{x}_2 \cdot x_4 + x_1 \cdot \bar{x}_2 \cdot \bar{x}_3 + x_1 \cdot x_2 \cdot \bar{x}_4$$

Bemerkung 3: Ein Multiplexer mit 2 Daten- und einem Select-Eingang erfüllt Kriterium 2. Wird dieser Multiplexer mit einem beliebigen Logik-Gatter mit 2 Eingängen davor oder dahinter verschmolzen (siehe folgende 3 Gleichungen), so erfüllt die neue Schaltung auch wieder Kriterium 2 (Nachweis hier nicht notiert): Eine Analyse der 3 Möglichkeiten

$$f \approx g(x_3, x_4) \cdot x_1 + \neg g(x_3, x_4) \cdot x_2$$

$$f \approx g(x_2, x_3) \cdot x_4 + x_1 \cdot \bar{x}_4 \qquad \text{(entspricht Gl. 6)}$$

$$f \approx g(x_4, x_3 \cdot x_1 + \bar{x}_3 \cdot x_2)$$

zeigt für beliebige Funktionen g mit 2 Eingängen, daß Kriterium 2 hierbei gilt.

Jetzt müßte noch nachgewiesen werden, daß alle Schaltungen nach Gl. 3 und Gl. 4 Multiplexer sind. Dies gilt aber nur mit Einschränkungen, die jetzt diskutiert werden.

Diskussion Kriterium 2:

Die gewünschte Gleichungsform wäre die eines 2:1-Multiplexers. Gl. 3 ähnelt schon wesentlich mehr einem Multiplexer als die allgemeine Form Gl. 5, aber sie beschreibt noch keinen 2:1-Multiplexer. Genau genommen ist Gl. 3 und Gl. 4 ein Multiplexer mit 6 Eingängen $(x_1, x_2, \bar{x}_1, \bar{x}_2, 0, 1)$ und 2 Select-Signalen (x_3, x_4), wodurch max. 4 Eingänge ausgewählt werden.

Da f_1 bis f_6 jeweils von x_3 und x_4 abhängen, und da die Auswahl der x_1 und x_2 auch 6 Varianten zuläßt, gibt es noch $\binom{4}{2} \cdot (6^{2^2} - 4^{2^2} - 4^{2^2} + 2^{2^2}) = 4800$ logische Funktionen (von ehemals $2^{2^4} = 65536$), die nach Kriterium 2 übrigbleiben. (Die Subtrahenden ergeben sich aus der Nebenbedingung der Variablenanzahl, da die Koeffizienten f_1 oder f_3 sowie f_2 oder f_4 jeweils mindestens einmal vorkommen sollen. Wegen Doppelauswahl - siehe Bemerkung 1 und 2 - sind es aber tatsächlich nur 4296 verschiedene Funktionen.)

Tabelle 15 enthält (bis auf Permutation und Negation der Eingänge) alle diese 4296 Funktionen. Davon sind 1296 Multiplexer; die restlichen 3000 haben auch eine Auswahl-Funktion, aber eine wesentlich kompliziertere. Auch diese zuletztgenannten Funktionen werden in dem Cluster-Algorithmus aus Kapitel 9 mit einbezogen, aber da sie in den untersuchten Beispielen nur sehr selten vorkommen, verhindern sie die Multiplexer-Rückerkennung in der Regel nicht. Außerdem stehen im Algorithmus auch noch die Umgebungsinformationen zur Verfügung (z.B. ein Multiplexer mit 4 und mehr Eingängen besteht aus mehr als einem LUT, die unmittelbar benachbart sind), so daß eine ausreichende Rückerkennung durchgeführt werden kann.

Aus diesem Grund der Unschärfe der Funktionsbedeutung kann Kriterium 2 nicht in einer strengen Form bewiesen werden; vielmehr ist die Aussage, daß es sich (wie bei den praktischen Beispielen bestätigt) „meist“ um einen Multiplexer handelt, wichtig. Durch das Zusammenspiel mit dem Cluster-Algorithmus aus Abschnitt 9.2 kann die verbleibende Unschärfe abgefedert werden, da noch eine Umgebungs-Information ausgewertet werden kann.

Tabelle 15: Aufstellung aller Funktions-Typen, die Kriterium 1 erfüllen

	Anzahl	Funktion
Multiplexer 1296 Stück	96	reine 2:1-Multiplexer mit 3 Eingängen
	240	siehe Bemerkung 3, 1. Gleichung
	480	siehe Bemerkung 3, 2. Gleichung
	480	siehe Bemerkung 3, 3. Gleichung
sonstige Funktionen gemäß Kriterium 2 3000 Stück	192	$f = \overline{s0} \cdot \overline{s1} \cdot x + (s0 \oplus s1) \cdot y + s0 \cdot s1$
	192	$f = \overline{s0} \cdot \overline{s1} \cdot x + (s0 \oplus s1) \cdot y$
	192	$f = s0 \cdot x + s1 \cdot y + s0 \cdot s1$
	96	$f = s0 \oplus s1 + \overline{s0} \cdot \overline{s1} \cdot x + s0 \cdot s1 \cdot y$
	96	$f = \overline{s0} \cdot \overline{s1} \cdot x + s0 \cdot s1 \cdot y$
	384	Schaltung aus Abb. 5
	384	$f = \overline{s1} \cdot (s0 \cdot \bar{x} + \overline{s0} \cdot x) + s1 \cdot \overline{s0} \cdot \bar{y}$
	384	$f = \overline{s1} \cdot (s0 \cdot \bar{x} + \overline{s0} \cdot x) + s1 \cdot \overline{s0} \cdot \bar{y} + s0 \cdot s1$
	192	$f = \overline{s1} \cdot (s0 \cdot y + \overline{s0} \cdot \bar{x}) + s1 \cdot \overline{s0} \cdot \bar{y}$
	192	$f = \overline{s1} \cdot (s0 \cdot y + \overline{s0} \cdot \bar{x}) + s1 \cdot \overline{s0} \cdot \bar{y} + s0 \cdot s1$
	384	$f = \overline{s0} \cdot \overline{s1} \cdot y + s0 \cdot \overline{s1} \cdot \bar{x} + s1 \cdot \bar{y}$
	192	$f = (s0 \oplus s1) \cdot \bar{y} + s0 \cdot s1 \cdot y + \overline{s0} \cdot \overline{s1} \cdot x$
	96	$f = (s0 \oplus s1) \cdot \bar{y} + s0 \cdot \overline{s1} \cdot x + \overline{s0} \cdot s1 \cdot \bar{x}$
	24	s. Bemerkung 2