Optimierung der Verdrahtbarkeit unter Berücksichtigung heterogener Verdrahtungsressourcen hierarchischer FPGA-Architekturen

vorgelegt von

Diplom-Ingenieur

Valerij Matrose

Von der Fakultät IV - Elektrotechnik und Informatik

der Technischen Universität Berlin

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften

Dr.-Ing.

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender : Prof. Dr.-Ing. Uwe Nestmann, Technische Universität Berlin

1. Gutachter : Prof. Dr.-Ing. Hans-Ulrich Post, Technische Universität Berlin

2. Gutachter : Prof. Dr.-Ing. Dietmar Tutsch, Bergische Universität Wuppertal

Tag der wissenschaftlichen Aussprache: 27. März 2009

Berlin 2009

D83

Danksagung

Die vorliegende Arbeit entstand im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter unter der Leitung von Herrn Prof. Dr.-Ing. Hans-Ulrich Post am Institut für Technische Informatik und Mikroelektronik der Technischen Universität Berlin, Fachgebiet Rechnertechnologie.

Ich danke Herrn Prof. Dr.-Ing. Hans-Ulrich Post für die zahlreichen wissenschaftlichen Anregungen, fachlichen Ratschläge und die vielen Diskussionen, Herrn Prof. Dr.-Ing. Dietmar Tutsch für die bereitwillige Übernahme des zweiten Gutachtens, Herrn Prof. Dr.-Ing. Hans Liebig für die fachlichen Gespräche, Herrn Dr.-Ing. Thomas Flik für das akribische Korrekturlesen der Arbeit, Dr.-Ing. Carsten Gremzow, Dipl.-Ing. Nico Moser, Dipl.-Ing. Sasa Vulinovic und Dipl. Ing. Waldemar Sawenko für die vielen fachlichen Gespräche und die moralische Unterstützung.

Ganz besonders danke ich meinen Eltern für die moralische Unterstützung und die zahlreichen erholsamen Stunden, die ich auf ihrem Anwesen verbringen konnte. Allen an dieser Stelle nicht Genannten, die zum Gelingen dieser Arbeit beigetragen haben, gilt mein besonderer Dank.

Valerij Matrose Berlin, im Oktober 2008

Abstract

The following work presents the architecture and the underlying algorithms of the clustering and placement tool PALLAs for commercial Altera FPGAs. During the development of PALLAs a variety of clustering and placement algorithms found in literature were re-implemented with respect to the particular architecture of the Altera FPGAs and then evaluated with respect to their efficiency. The algorithm's strengths and deficiencies were investigated and in turn used to develop and optimize new core clustering placement realized in the PALLAs system. Since this work is restricted to a particular commercial FPGA architecture one of the main deficiencies of the investigated clustering and placement algorithms where uncovered: Structural simplifications found in traditional methods designed for hierarchical FPGAs strongly degrade efficiency for real world FPGA architectures.

Most commercial FPGA vendors feature two different architecture types on their product portfolio: A low cost version with limited amounts of routing resources and a high-end variant with an abundance of routing resources. For target applications where speed is not critical, vertical migration to a device with limited routing resources might be possible which in turn leads to significant economical advantages. If the aforementioned vertical migration process is possible for large design with high interconnect complexity largely depends on the quality of the applied clustering and placement algorithms for a given design and how well they make use of the target architecture's interconnect properties. Therefore it is imminent that clustering and placement algorithms are aware of the target architecture's particular routing capabilities in order to improve on a design's overall routability.

During clustering the PALLAs systems seeks to improve interconnectivity within any given logic block which in turn increases the utilization of local routing resources. Hence, global interconnect resources will be spared and the overall routability of a design will increase. Previous work strongly underlines the impact of clustering on overall routability for any placed design. Yet, one aspect which was previously unaccounted for is the fact, that with simultaneous clustering and placement of logic cells the initial clustering of a logic cell strongly influences the pending phase of placement. Therefore clustering as a precursor or placement remains a critical design step independently from the actual placement algorithm.

The PALLAs system increases routability during placement by reducing the segment length of networks subject to global routing. Estimation of the segment length is performed using a modified bounding-box cost function which introduces additional weighting factors taking into account the particular interconnect capabilities of a given FPGA architecture and its cost efficient placement alternatives. In order to uncover areas with over-utilization of local interconnect resources for a given design, the latter is modelled for the FPGA's floorplan in total. An additional increase in overall routability is obtained by shortening global interconnect resources for areas with over-utilization of local interconnect resources on purpose. As a consequence the clustering and placement algorithms implemented in the PALLAs system provide the necessary means to allow vertical migration to cost efficient FPGAs for designs with a high demand of global interconnect resources.

The PALLAs system integrates seamlessly into the existing Altera Quartus II design flow which allows an in-detail analysis of resource usage and timing properties of the generated placements for complex real world designs. The efficacy of the new clustering and placement algorithms is proven by direct comparison to existing methods from academia as well as the commercial variants provided by Altera Quartus II. Results for numerous benchmark designs were investigated. As a result the new clustering and placement algorithms lead to an overall reduction of interconnect utilization of up to 21% as well as to a frequency speedup of up to 25%.

Zusammenfassung

Die vorliegende Arbeit beschreibt den Aufbau und die zugrunde gelegten Algorithmen des Clustering- und Platzierwerkzeugs PALLAs für die kommerziellen Altera-FPGAs. Während der Entwicklung von PALLAs wurden verschiedene Clustering- und Platzieralgorithmen im Hinblick auf die Architektur der kommerziellen Altera-FPGAs neu implementiert und miteinander verglichen. Die dabei aufgedeckten Stärken und Defizite der einzelnen Algorithmen wurden analysiert und dokumentiert, die dabei gewonnen Erkenntnisse flossen in die Entwicklung und Optimierung der Kernalgorithmen von PALLAs. Aufgrund der Festlegung auf eine kommerzielle FPGA-Zielarchitektur konnte eine der wesentlichen Schwächen der analysierten Clustering- und Platzierverfahren aufgedeckt werden: Strukturelle Vereinfachungen des häufig eingesetzten klassischen Modells eines hierarchischen FPGA schränken die Effektivität dieser Verfahren bei real existierenden FPGA-Architekturen ein.

Die großen FPGA-Hersteller bieten in der Regel low-cost FPGA-Familien mit einigen wenigen und high-end FPGA-Familien mit zahlreicheren Verdrahtungsebenen an. Ist die Geschwindigkeit eines Designs unkritisch, kann unter Umständen eine vertikale Migration zu einem Baustein mit weniger Verdrahtungsressourcen vollzogen werden, was in der Praxis mit einer Kostenersparnis verbunden ist. Ob ein Migrationsschritt möglich ist, hängt bei den Designs mit hohem Verdrahtungsressourcenbedarf davon ab, ob die Verdrahtbarkeit der Designs gewährleistet werden kann. Deshalb ist es von entscheidender Bedeutung, dass die Clustering- und Platzieralgorithmen bei der Verbesserung der Verdrahtbarkeit die besonderen Merkmale der FPGA-Zielarchitekturen berücksichtigen und ausnutzen.

Bei dem Clustering-Entwurfsschritt versucht PALLAs, die Interkonnektivität innerhalb der Logikblöcke und somit die Nutzung der lokalen Verdrahtungsressourcen zu erhöhen. Auf diese Weise werden der globale Verdrahtungsressourcenbedarf verringert und die Verdrahtbarkeit der Designs erhöht. Die bisherigen Arbeiten dokumentieren gut den großen Einfluss von Clustering auf die Verdrahtbarkeit der platzierten Designs. Ein bisher unterschätzter Aspekt ist hingegen die Tatsache, dass auch bei gleichzeitigem Clustering und Platzierung der Logikzellen in einem einzigen Entwurfsschritt das initiale Clustering der Logikzellen sich auf die nachfolgende Platzierung auswirkt. Deshalb bleibt Clustering als Vorstufe zur Platzierung, unabhängig von der Art der Platzierung, ein kritischer Entwurfsschritt.

Die Verbesserung der Verdrahtbarkeit erfolgt beim Platzieren mit PALLAs durch eine Reduzierung der zur globalen Verdrahtung erforderlichen Leitungslängen. Die Schätzung der erforderlichen Leitungslängen findet mit einer modifizierten Bounding-Box-Kostenfunktion statt. In diese Kostenfunktion fließen zusätzliche Gewichtungsfaktoren ein, die Besonderheiten der gegebenen Verdrahtungsarchitektur und verdrahtungsgünstigere Platzierpositionen berücksichtigen. Um FPGA-Bereiche mit lokaler Überbenutzung der Verdrahtungsressourcen aufzuspüren, kann deren Nutzung über die gesamte FPGA-Fläche modelliert werden. Eine weitere Steigerung der Verdrahtbarkeit wird erreicht, indem die globalen Verdrahtungsressourcen in den Bereichen mit überbenutzten Verdrahtungsressourcen künstlich verknappt und somit entlastet werden. Bei kostensensitiven Designs mit hohem Verdrahtungsressourcenbedarf können die in PALLAs implementierten Clustering- und Platzierstrategien eine vertikale Migration zu einem preiswerteren FPGA ermöglichen.

Das entwickelte Werkzeug PALLAs wurde in den bestehenden Entwurfsfluss der Altera-Quartus-II-Software eingebettet, was eine realistische Analyse des Ressourcenbedarfs und des Zeitverhaltens der erzeugten Platzierungen ermöglichte. Die Wirksamkeit der neu entwickelten gegenüber einigen bekannten Verfahren aus dem Forschungsumfeld und der herstellereigenen Software wird anhand zahlreicher Benchmarkschaltungen belegt. Mit den entwickelten Verfahren war es möglich, den Verdrahtungsressourcenbedarf der Schaltungen um bis zu 21% zu verringern und den maximalen Takt dieser um bis zu 25% zu erhöhen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ALM	Adaptive Logic Module
ALU	Arithmetic Logic Unit
ARH-Pack	Area, Routability and Hierarchy driven Packer
ASIC	Application Specific Integrated Circuit
ASSP	Application Specific Standard Products
BBox	Bounding Box
BGA	Ball Grid Array
CMOS	Complementary Metal Oxide Semiconductor
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
DLL	Dynamic Link Library
DSP	Digital Signal Processing
EDA	Electronic Design Automation
EEPROM	Electrically Erasable Programmable Read Only Memory
FF	Flipflop
fGREP	Fast Generic Routability Estimation for Placed FPGA circuits
FIFO	First In, First Out
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
HDL	Hardware Description Language
HLS	High-Level Synthesis
IC	Integrated Circuit
I/O	Input / Output
IP	Intellectual Property
iRAC	Interconnect Resource Aware Clustering
KDNF	Kanonische Disjunktive Normalform
KKNF	Kanonische Konjunktive Normalform
LAB	Logic Array Block (Altera-eigene Logikblockbezeichnung)
LB	Logikblock
LC	Logic Cell
LE	Längeneinheiten
LFF	Less Flexible First
LUT	Look-Up Table
MCNC	Microelectronics Center of North Carolina
MOS	Metal Oxide Semiconductor
MP3	MPEG-1 Audio Layer 3
MPCPlace	Multipass Clustering and Placement

NMOS	N-Channel Metal-Oxide Semiconductor
PLA	Programmable Logic Array
PMOS	P-Channel Metal-Oxide Semiconductor
QMCV	Quine und McCluskey Verfahren
QUIP	Quartus University Interface Program
QSF	Quartus Constraint File
RAM	Random Access Memory
RCF	Routing Constraint File
RISA	Routability modeling Strategy
RPack	Routability driven Packer
RTL	Register Transfer Level
SA	Simulated Annealing
SCPlace	Simultaneous timing driven Clustering and Placement
SMAC	Simultaneous Mapping And Clustering
SPCD	Simultaneous Placement with Clustering and Duplication
SPICE	Simulation Program with Integrated Circuit Emphasis
SPLD	Simple Programmable Logic Device
SRAM	Static Random Access Memory
Tcl/Tk	Tool command language / Toolkit
T-RPack	Timing and Routability driven Packer
T-VPack	Timing driven Versatile Packer
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale Integration
VPack	Versatile Packer
VPR	Versatile Place and Route
VQM	Verilog Quartus Module
XML	eXtensible Markup Language

1 Einleitung

1.1 Überblick

Seit ihrer kommerziellen Einführung im Jahr 1985 haben die feldprogrammierbaren Logikbausteine (engl.: Field Programmable Gate Arrays, FPGAs) die Art und Weise revolutioniert, wie ein digitales System entworfen werden kann. Die Ursachen dafür sind die Flexibilität der FPGAs, der technologische Fortschritt der Halbleiterfertigung und die wirtschaftlichen Rahmenbedingungen der Elektronikindustrie.

Die FPGAs enthalten programmierbare Funktionsgeneratoren (Logikzellen) und programmierbare Verdrahtungsstrukturen. Durch geeignete Konfiguration der Verdrahtungsstrukturen können die Funktionsgeneratoren flexibel untereinander verbunden werden, so dass nahezu jede logische Funktion implementiert werden kann. Um eine möglichst effiziente Abbildung logischer Funktionen aus unterschiedlichsten Anwendungsbereichen, wie z. B. digitaler Signalverarbeitung oder Kommunikation zu ermöglichen, werden zusätzlich dedizierte Recheneinheiten und Schnittstellen auf den FPGAs integriert. Damit ging im Laufe der letzten Jahre auch die Diversifikation der FPGA-Architekturen und ihrer Einsatzbereiche einher, die zahlreiche FPGA-Familien mit speziellen Einsatzgebieten hervorbrachte. Der Anwender kann sich heute für einen FPGA-Baustein entscheiden, der die Anforderungen seines Designs am besten erfüllt.

Der technologische Fortschritt ermöglicht Herstellung von integrierten Schaltungen (ICs) mit mehreren Milliarden Transistoren. Das stellt sowohl die Anwender als auch die Hersteller von Softwarewerkzeugen zur Entwicklung maßgeschneiderter ICs (ASICs) vor das immer wiederkehrende Problem der Beherrschung solcher Design-Komplexitäten. Dieses hat hingegen keine Relevanz für die FPGA-Hersteller. Sie können aufgrund der regelmäßigen FPGA-Strukturen automatische Werkzeuge einsetzen, um Masken für die Herstellung der FPGAs in kurzer Zeit zu entwerfen. Somit sind in der Regel zeitgleich mit der Einführung eines neuen Fertigungsprozesses, der noch kleinere Halbleiterstrukturen und somit die Integration von noch mehr Transistoren auf einem IC ermöglicht, FPGAs verfügbar, die mit diesem Prozess gefertigt werden (Abbildung 1-1).

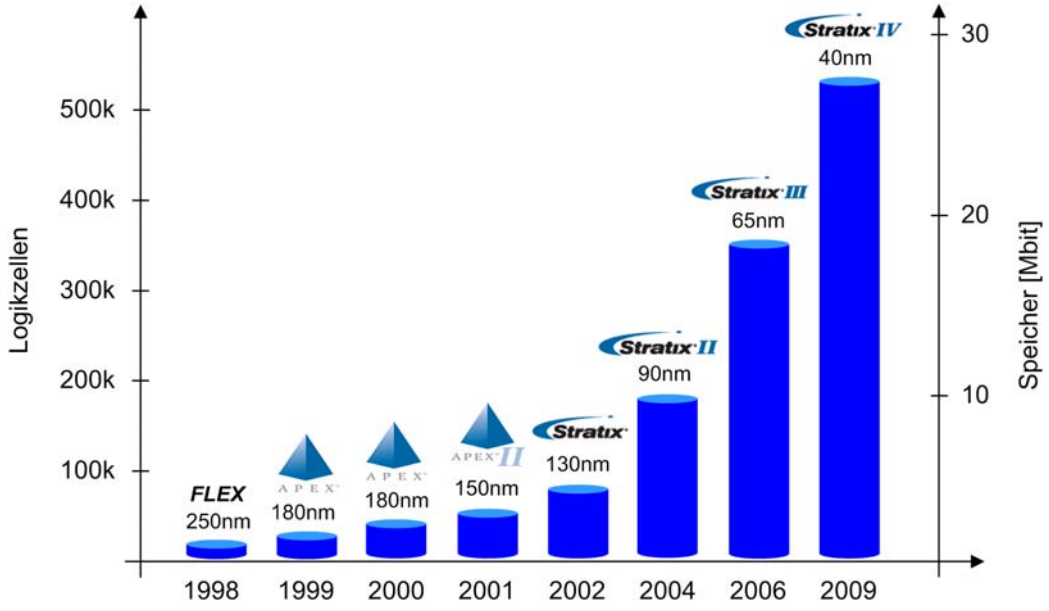


Abbildung 1-1: *FPGA-Integrationstrend im Zeitraum von über 10 Jahren am Beispiel der Bausteine des Herstellers Altera.*

Von den Vorteilen der neuen Fertigungsprozesse – z. B geringerer Leistungsbedarf, höherer Integrationsgrad und niedrigere Signallaufzeiten – profitiert der FPGA-Anwender unmittelbar. So sind bereits FPGAs erhältlich, die digitale Schaltungen mit bis zu 20 Millionen Gatter-Äquivalenten (ein Gatter-Äquivalent entspricht einem NAND-Gatter bzw. vier Transistoren) aufnehmen können und Taktfrequenzen von bis zu 500 MHz ermöglichen.

Für den Erfolg eines Unternehmens ist die Zeit zwischen einer Produktidee und der Produktfertigung (engl.: time-to-market) von entscheidender Bedeutung. Kann zur Implementierung eines digitalen Systems ein FPGA eingesetzt werden, so sinkt diese Zeit im Vergleich zu einer ASIC-basierten Implementierung um den Faktor zwei (Abbildung 1-2).

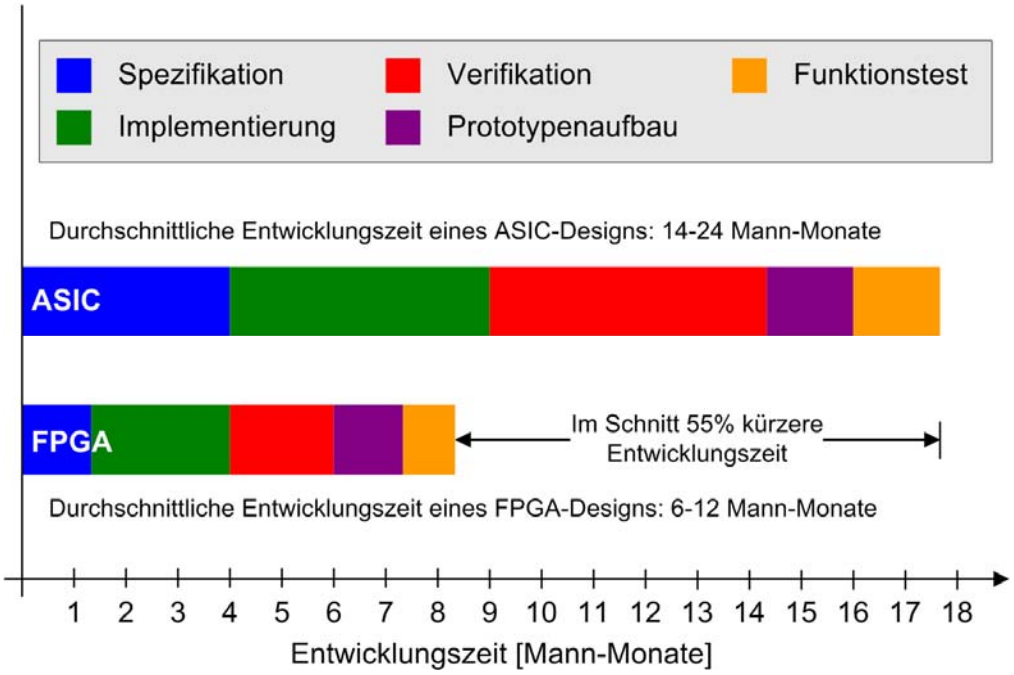


Abbildung 1-2: *Der Entwicklungsaufwand einer FPGA-basierten Implementierung ist nur halb so groß im Vergleich zu einem ASIC-basierten Ansatz [69].*

Ein weiterer Faktor, der über Erfolg und Machbarkeit eines Produkts entscheidet, sind die einmaligen Entwicklungskosten, die bei unterschiedlichen Implementierungen eines Produkts anfallen. Den größten Anteil hier stellen bei ASICs die Maskenkosten dar, die bei den aktuellen Fertigungstechnologien einige Millionen US-Dollar betragen können. Hinzu kommen mit einigen Hunderttausend US-Dollar die Kosten für die Entwicklungstools. Für diese fixen Kosten muss der IC-Auftraggeber allein aufkommen. Auf der anderen Seite können die FPGA-Hersteller sowohl die Maskenkosten als auch die Entwicklungskosten für Werkzeuge unmittelbar in den Verkaufspreis der angebotenen FPGA-Bausteine einkalkulieren. Hier profitiert der FPGA-Anwender von den überschaubaren Bausteinkosten und zunehmend kostenlosen Entwicklungswerkzeugen für die FPGAs.

Die hohen Kosten für eine ASIC-Entwicklung und die steigende Komplexität der verfügbaren FPGAs können als wichtigste Gründe dafür angesehen werden, dass immer weniger digitale Schaltungen als ASICs implementiert werden. So sank die Zahl der ASIC-Neuentwicklungen von 11.000 im Jahr 1997 auf 3.000 im Jahr 2005 [96] – Zur Implementierung digitaler Schaltungen bzw. Systeme eignen sich neben den ASICs und FPGAs weitere IC-Typen. Auf diese und auf eine Klassifizierung der integrierten Schaltungen im Allgemeinen wird im folgenden Abschnitt kurz eingegangen.

1.2 Anwendungsbezogene Klassifizierung integrierter Schaltungen

Integrierte Schaltungen können in die Hauptgruppen der Standard-ICs, der anwendungsspezifischen Standard-ICs (ASSPs) und der maßgeschneiderter ICs (ASICs) eingeteilt werden (Abbildung 1-3). Die Standard-ICs werden möglichst universell ausgelegt, so dass zahlreiche Anwender aus den unterschiedlichen Anwendungsbereichen angesprochen werden. Typische Beispiele sind programmierbare Prozessoren und Speicherbausteine. Mit den Standard-ICs können sehr leistungsfähige digitale Systeme realisiert werden. Trotz steigenden Integrationsgrads bleibt auch die Nachfrage nach sehr einfacher Standard-Logik erhalten.

Können die Anforderungen an ein digitales System nicht unter Verwendung der Standard-ICs erfüllt werden, kann für diese bestimmte Anwendung ein komplett maßgeschneiderter IC (engl.: full-custom ASIC) gefertigt werden. Dadurch ist es in der Regel möglich, schnellere und energieeffizientere Systeme aufzubauen, als dies mit den Standard-ICs möglich wäre. Um die Fertigungskosten zu reduzieren, werden auch teilvorgefertigte ICs (engl.: semi-custom ASIC) angeboten. Bei diesen ICs sind die unteren Lagen mit den Transistoren-Arrays (engl.: sea-of-gates) und der Spannungsversorgung bereits vorgefertigt. Der Kunde bestimmt das Aussehen der oberen Metalllagen und somit die Funktionalität des IC allein durch die von ihm zu bestimmende Verdrahtung. Deshalb wird bei den teilvorgefertigten ASICs auch von maskenprogrammierbaren ICs gesprochen. Da in diesem Fall nur die Masken für die oberen Metalllagen neu erstellt werden müssen, ist ein teilvorgefertigter IC wesentlich preiswerter als ein komplett maßgeschneiderter IC.

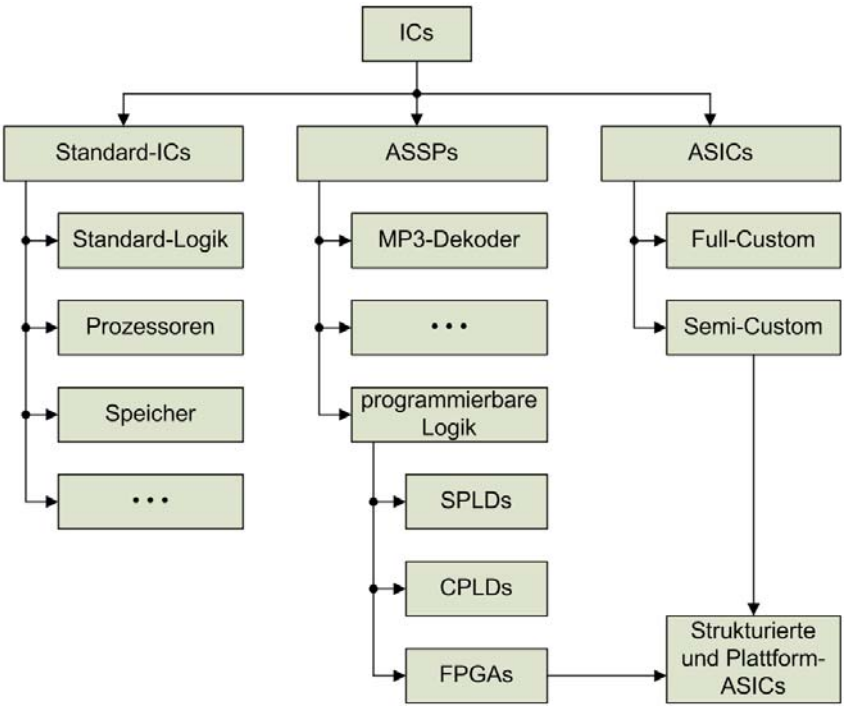


Abbildung 1-3: *Klassifizierung der digitalen integrierten Schaltungen.*

Stellt ein IC-Hersteller fest, dass ein ASIC in unveränderter oder in leicht modifizierter Form für einen breiten Kundenkreis interessant sein könnte und nimmt er dessen Herstellung auf, so handelt es sich um einen anwendungsspezifischen Standard-IC (ASSP). So kann beispielsweise ein MP3-Dekoder zwar mit Hilfe eines Standard-Prozessors implementiert werden. Doch wird ein ASSP, der die gleiche Aufgabe übernehmen kann, vor allem bei tragbaren Anwendungen aufgrund der geringeren Größe und der besseren Energieeffizienz bevorzugt.

Eine eigene Untergruppe der ASSPs stellen die programmierbaren Logikbausteine dar. Deren Einsatzbereich ist die Implementierung benutzerdefinierter Logikschaltungen ohne die Notwendigkeit, einen ASIC herstellen zu müssen. Bei den programmierbaren Logikbausteinen wird zwischen den einfachen und komplexen programmierbaren Logikbausteinen (SPLDs bzw. CPLDs) und feldprogrammierbaren Logikbausteinen unterschieden (FPGAs).

Die Komplexität des Schaltungsentwurfs für die Deep-Submicron-Fertigungsprozesse (130nm und darunter) förderte das Entstehen von Strukturierten ASICs und Plattform-ASICs (engl.: structured/platform ASICs). Bei diesen Bausteinen sind wie bei den FPGAs ein Teil der Ressourcen, wie dedizierte Recheneinheiten, Speicher, Schnittstellen und Taktbäume, fertig verdrahtet ausgeführt. Zur Aufnahme benutzerdefinierter Logik sind wie bei teilvorgefertigten ASICs maskenprogrammierbare Funktionsgeneratoren enthalten, was sich im Vergleich zu den FPGAs in einer etwa 90%-igen Flächenersparnis bei gleicher Logikkapazität auswirkt [98]. Auch große FPGA-Hersteller bieten Strukturierte ASICs an, die einen Umstieg ohne Redesign möglich machen. Ein nachträglicher Umstieg kann sich als ökonomisch sinnvoll erweisen, da Strukturierte ASICs bei großen Stückzahlen günstiger als FPGAs sind. Das Überführen eines FPGA-Designs in einen Strukturierten ASIC des entsprechenden FPGA-Herstellers gestaltet sich aufgrund des nahezu identischen Aufbaus (bis auf die Funktionsgeneratoren und Verdrahtungsressourcen) besonders einfach. Hierzu muss die Programmierung und Verdrahtung von FPGA-Funktionsgeneratoren in die entsprechende Verdrahtung der maskenprogrammierbaren Funktionsgeneratoren des Strukturierten ASIC umgesetzt werden [98]. Das kann kostengünstiger und schneller erfolgen als ein komplettes Redesign für einen teilvorgefertigten ASIC.

1.3 Programmierbare Logikbausteine

Die Einteilung der programmierbaren Logikbausteine in SPLDs, CPLDs und FPGAs aus technologischer Sicht kann nicht immer eindeutig erfolgen, da sich die Grenzen zwischen diesen zusehends verwischen [01]. Am besten kann die Einteilung der programmierbaren Logikbausteine aufgrund ihres Ursprungs und Einsatzzwecks erfolgen. Die einfachen programmierbaren Logikbausteine (SPLDs) sollten in der Anwendung nur wenige diskrete Standard-Logik-ICs ersetzen. Sie sind mit einer programmierbaren UND/ODER-Matrix (PLA) ausgestattet, die eine direkte Abbildung der Kanonischen Disjunktiven Normalform (KDNF) ermöglicht. Die Ein- und die Ausgänge des PLA sind in der Regel direkt mit den IC-Pins verbunden. Mit den SPLDs sind Schaltungen mit einigen wenigen hundert Gatter-Äquivalenten realisierbar.

Ein CPLD beinhaltet mehrere seitlich angeordnete PLAs, die über eine programmierbare Verdrahtungsmatrix miteinander verbunden werden können, um im Verbund komplexere Schaltungen implementieren zu können (Abbildung 1-4a). Den Ausgängen der PLAs können wahlweise Flipflops nachgeschaltet werden, womit einfache Schaltwerke implementierbar sind. Die Gehäuse-Pins der CPLDs sind über programmierbare I/O-Blöcke ebenfalls mit der Verdrahtungsmatrix verbunden – mit den CPLDs sind Schaltungen mit einigen tausend Gatter-Äquivalenten realisierbar. Aufgrund der regelmäßigen Verdrahtungsmatrix weisen die CPLDs (genau wie die SPLDs) ein deterministisches Verzögerungsmodell auf, welches unabhängig von dem tatsächlichen Signalpfad ist. Die zu erwartenden Verzögerungszeiten können in der Regel von einem geübten Entwickler manuell berechnet werden. Hierzu sind in den entsprechenden Hersteller-Datenblättern das genaue Verzögerungsmodell und die unterschiedlichen Verzögerungszeiten angegeben.

Bei den FPGAs sind die programmierbaren Funktionsgeneratoren in einem zweidimensionalen Feld angeordnet (Abbildung 1-4b). Zwischen den Funktionsgeneratoren befinden sich Verdrahtungskanäle, die programmierbare Verdrahtungsstrukturen beinhalten.

Anders als bei den CPLDs sind bei den FPGAs sowohl die Funktionsgeneratoren als auch die Verdrahtungsstrukturen gleichmäßig über den gesamten Baustein verteilt. Außerdem weisen die Funktionsgeneratoren der FPGAs eine geringere Komplexität als die PLAs der CPLDs auf, deshalb können sie weniger benutzerdefinierte Logik aufnehmen. In diesem Zusammenhang wird auch von der groben Granularität der CPLD-Funktionsblöcke bzw. feinen Granularität der FPGA-Funktionsblöcke gesprochen.

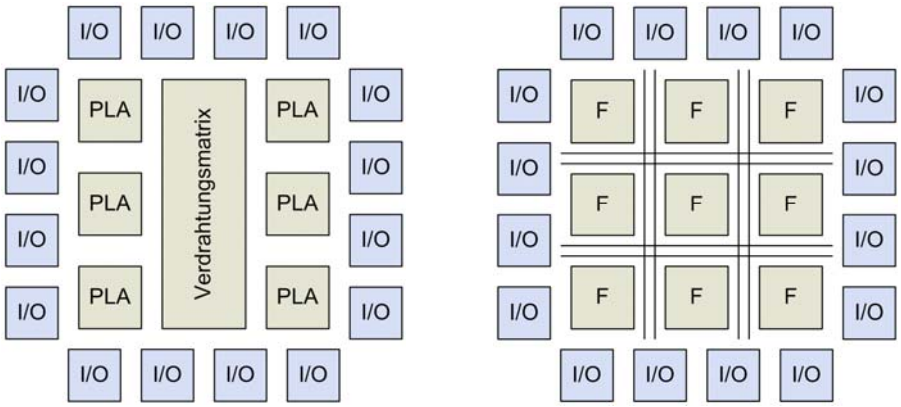


Abbildung 1-4: *Prinzipieller Aufbau eines CPLD (a) und eines FPGA (b).*

Die Programmierbarkeit der SPLDs, CPLDs und FPGAs wird durch die programmierbaren Schalter ermöglicht. Dabei wird zwischen den irreversiblen und reversiblen Programmiertechnologien unterschieden. Bei den irreversiblen Programmiertechnologien wird der Durchlasswiderstand programmierbarer Schalter auf elektromechanischem Wege unumkehrbar verändert, d. h. die Programmierbarkeit wird inhärent durch den Aufbau des Schalters selbst sichergestellt. Bei den reversiblen Programmiertechnologien beruht die Programmierbarkeit auf einer geeigneten Zusammenschaltung programmierbarer Speicherzellen und steuerbarer Schalter. Dabei werden Schalter in Form von Pass-Transistoren oder Transmission-Gattern von reprogrammierbaren Speicherzellen gesteuert.

Bei reversiblen Programmiertechnologien eingesetzte Speicherzellen verlieren nach einem Abfall der Spannungsversorgung entweder die gespeicherten Informationen (z. B. SRAM-Speicherzellen) oder behalten diese bis zum nächsten Programmiervorgang (Flash- oder EEPROM-Speicherzellen). Da die Integration der Flash- bzw. EEPROM-Speicherzellen zusammen mit CMOS-Schaltungen auf dem selben Substrat zusätzliche Prozessschritte und Masken erfordert, werden reversibel programmierbare FPGAs fast ausschließlich mit SRAM-basierten Speicherzellen hergestellt. Der Marktanteil der SRAM-basierten FPGAs an weltweit hergestellten FPGAs liegt bei über 85%.

1.4 Motivation

Im folgenden Abschnitt sollen die Entwurfsentscheidungen der FPGA-Hersteller aus ökonomischer Sicht betrachtet und der Zusammenhang zwischen den damit aufgeworfenen Problematiken und der vorliegenden Arbeit hergestellt werden.

1.4.1 Wirtschaftliche Aspekte

Die neuen FPGA-Familien sind aufgrund des Konkurrenzkampfes zwischen den FPGA-Herstellern in der jeweils aktuellen Halbleiter-Prozesstechnologie verfügbar. Deren Vorteile werden direkt an den Anwender in Form der niedrigeren FPGA-Stückkosten und höheren

Komplexitäten der damit realisierbaren Schaltungen weitergegeben. Mit jeder neu eingeführten FPGA-Familie wuchs in der Vergangenheit die Kluft zwischen dem, was technologisch zu einem hohen Preis machbar war und dem, was eine wachsende Anwendergruppe für ihre niedrig performanten Anwendungen tatsächlich benötigte. Deshalb sah sich der marktführende FPGA-Hersteller Xilinx 1998 zur Einführung einer neuen FPGA-Familie für das Marktsegment mit den kostensensitiven (low-cost) und niedrig performanten Anwendungen genötigt. In den Jahren 2000 bis 2006 wiesen die Verkaufszahlen der neu eingeführten Spartan-Familie ein jährliches Wachstum von 36,2% und bescherten Xilinx einen zusätzlichen kumulativen Gewinn von 2,1 Milliarden US-Dollar [03]. Der FPGA-Hersteller Altera folgte 2002 dem Beispiel von Xilinx mit einer eigenen low-cost FPGA-Familie. In diesem Markt der low-cost FPGAs sind traditionell auch noch weitere Mitbewerber wie Lattice und Actel tätig, so dass der Preiskampf unter mehr Anbietern erfolgt und das Preis-/Leistungsverhältnis besser ist, als bei den FPGAs des weniger umkämpften Marktsegments für high-end FPGAs [03].

Die geringeren Fertigungskosten der low-cost FPGAs im Vergleich zu den high-end FPGAs sind vor allem auf die geringere Anzahl an Metallisierungslagen zurückzuführen, weshalb weniger Belichtungsmasken und Prozessschritte für deren Fertigung erforderlich sind. So werden die Bausteine der Altera-Cyclone-III-Familie mit neun, die Altera-Stratix-III-Bausteine mit insgesamt zwölf Metalllagen gefertigt. Da die globalen Verdrahtungssegmente der programmierbaren Verdrahtungsstrukturen aktueller FPGAs in den Metalllagen Platz finden, äußert sich die Metalllagenreduzierung in den weniger zahlreichen globalen Verdrahtungsressourcen der low-cost FPGAs. Außerdem wird bei den low-cost FPGAs die pro implementierbare Gatterfunktion notwendige Silizium-Fläche gesenkt, da die Konfigurationszellen für die nicht mehr vorhandenen Verdrahtungssegmente entfallen. In der Abbildung 1-5 wird die Preisgestaltung einzelner Bausteine der beiden FPGA-Familien von Altera den verfügbaren Verdrahtungs- und Logikressourcen gegenübergestellt. Wie zu erkennen ist, sind für Designkomplexitäten mit bis zwei Millionen Gatter-Äquivalenten (ca. 120.000 Logikzellen) sowohl low-cost als auch high-end FPGAs verfügbar.

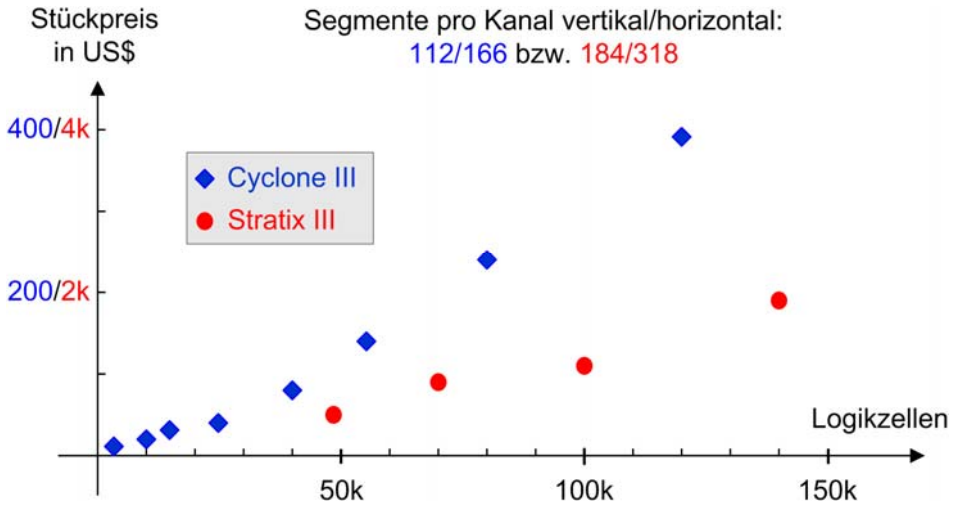


Abbildung 1-5: *Preisgestaltung vs. Verdrahtungs- und Logikressourcen bei den aktuellen Altera-Stratix-III- und Altera-Cyclone-III-Familien.*

Ist die Geschwindigkeit einer Anwendung unkritisch und ein low-end FPGA mit geforderten Logikressourcen verfügbar, kann unter Umständen eine vertikale Migration von einem high-end zu einem low-cost FPGA mit weniger Verdrahtungsressourcen vollzogen werden, was in der Praxis mit einer Kostenersparnis verbunden ist. Ob ein Migrationsschritt möglich

ist, hängt bei den Designs mit einem hohen Verdrahtungsressourcenbedarf davon ab, wie gut die Clustering- und Platzieralgorithmen die Merkmale der Zielarchitektur berücksichtigen und ausnutzen. Obwohl die Verdrahtungssegmente 70-80% der Chipfläche eines modernen FPGA belegen, kann bei Designs mit hoher Interkonnektivität nicht garantiert werden, dass die platzierte Logik vollständig verdrahtet werden kann. Da der überwiegende Anteil aller Netze der meisten FPGA-Designs kleine Netze mit 2 bis 4 Anschlüssen sind [04], spielen die lokalen Verdrahtungsressourcen beim Verdrahtungsentwurfsschritt eine besonders wichtige Rolle. Gelingt es, die Logik so zu platzieren, dass zur Verdrahtung der Netze mit einem geringen Fanout lokale Verdrahtungsressourcen des FPGA herangezogen werden können, kann der Nutzungsgrad globaler Verdrahtungsressourcen gesenkt, die Verdrahtbarkeit gesteigert und so eine vertikale Migration ermöglicht werden.

1.4.2 Verwertbarkeit akademischer Forschungsergebnisse

Den bisher veröffentlichen Forschungsarbeiten auf dem Gebiet der programmierbaren Logikbausteine liegt überwiegend (ca. 95%) eine bestimmte FPGA-Zielarchitektur zugrunde. Diese basiert auf hierarchisch aufgebauten Logikblöcken; die globale Verdrahtung erfolgt über Verdrahtungssegmente der Länge Eins und durch an den Kreuzpunkten der Verdrahtungskanäle angeordnete Switch-Boxen. Eine ausführliche Betrachtung dieser FPGA-Architektur gibt es unter anderem in [14]. Im Folgenden werden Bezüge auf diese FPGA-Zielarchitektur mit „klassische FPGA-Zielarchitektur“ bezeichnet, um eine Unterscheidung zu den FPGA-Zielarchitekturen der kommerziellen Hersteller zu ermöglichen.

Weder die klassische FPGA-Zielarchitektur noch irgendeine andere, ähnlich einfache FPGA-Zielarchitektur aus dem akademischen Forschungsbereich, wurde tatsächlich als integrierte Schaltung erfolgreich gefertigt und eingesetzt. Diese Tatsache offenbart das größte Dilemma der Forschung, die unter Verwendung der klassischen FPGA-Zielarchitektur durchgeführt wurde: Die hierbei erzielten Erkenntnisse sind möglicherweise auf sehr viel komplexere kommerzielle (und somit real existierende) FPGAs nicht direkt übertragbar. Auf der anderen Seite tun sich FPGA-Hersteller schwer, das Wissen über den genauen Aufbau ihrer FPGAs preiszugeben, weil von diesem Wissen auch die Konkurrenz profitieren könnte. Seit vier Jahren bemüht sich Altera als einziger FPGA-Hersteller darum, die Forscher aktiv in den Entwicklungsprozess alternativer EDA-Software für seine FPGAs einzubinden – erst durch diese Unterstützung wird die Integration eigener EDA-Werkzeuge in den Entwurfsfluss ermöglicht [93]. Aus dieser Betrachtung heraus erschien es reizvoll, die Wirksamkeit einiger bekannter Algorithmen anhand kommerzieller FPGA-Architekturen zu belegen.

Die FPGA-Architekturen des Marktführers Xilinx unterscheiden sich erheblich von der klassischen FPGA-Architektur. So weisen die Logikblöcke der Xilinx-FPGA keine lokalen Intra-Logikblockverdrahtungsressourcen auf, die einen großen Teil der Verdrahtung aufnehmen und so zu einer effizienten Entlastung der globalen Verdrahtungsressourcen und zu einer besseren Verdrahtbarkeit beitragen können. Für die vorliegende Arbeit wurden die FPGA-Architekturen des Herstellers Altera als Zielarchitekturen ausgewählt, da diese eine große Ähnlichkeit zu der klassischen FPGA-Architektur aufweisen und somit geeignet erschienen, bisherige Forschungsergebnisse auf ihre Verwertbarkeit zu überprüfen.

1.5 Eigener Beitrag

Im Rahmen der Entwicklung eines Werkzeugs zur Partitionierung, Flächenplanung und Platzierung (PALLAs) für Altera-Cyclone- und Stratix-FPGAs wurden die bekannten Clustering-Algorithmen VPack [43], RPack [44] und iRAC [38] hinsichtlich der Architektur der kommerziellen Altera-FPGAs neu implementiert und miteinander verglichen. Dabei zeigte es sich, dass die unter kommerziellen hierarchischen FPGAs weit verbreiteten Architekturmerkmale, wie vor- oder nachgeschaltete Flipflops und logikblockweite Flipflop-

Steuersignale, von diesen Clustering-Algorithmen nicht unterstützt werden, weshalb die Logik- und Verdrahtungsressourcen der Altera-FPGAs nicht effektiv genutzt werden können. Um die so aufgedeckten Defizite zu beheben, wurde ein neues „bottom-up"-Clustering-Verfahren ARH-Pack (Area, Routability & Hierarchy driven Packer) entwickelt, welches die Verdrahtbarkeit und die Logikausnutzung verbessert. Die Auswirkungen der neuen Optimierungsstrategien zur möglichst effektiven Nutzung der genannten Architekturmerkmale der Altera-FPGAs auf die Verdrahtbarkeit und die Logiknutzung wurden untersucht und mit Ergebnissen zahlreicher nichtsynthetischer Benchmarks belegt.

Zur Platzierung der Logik mit PALLAs wurde ein neues Platzierverfahren MPCPlace (Multipass Clustering and Placement) entwickelt, das durch gezielte Nutzung der lokalen Verdrahtungsressourcen, Reduzierung der Gesamtnetzlänge und Logikverschiebung in die weniger genutzten FPGA-Bereiche die Überbenutzung (Kongestion) der globalen Verdrahtungsressourcen reduziert und somit die Verdrahtbarkeit der FPGA-Designs steigert. Die Ergebnisse des neuen Platzierverfahrens wurden mit den Platzierverfahren SCPlace [71], VPR [68] und der kommerzieller Altera-Quartus-II-Software anhand mehrerer praxisbezogener Designs verglichen.

Das entwickelte Werkzeug PALLAs wurde in den bestehenden Entwurfsfluss der Altera-Quartus-II-Software eingebettet, was eine realistische Analyse des Ressourcenbedarfs und des Zeitverhaltens der erzeugten Platzierungen ermöglichte.

1.6 Aufbau der Arbeit

In Kapitel 2 werden die theoretischen Grundlagen und der Stand der Technik des digitalen Schaltungsentwurfs mit FPGAs betrachtet. Abschnitt 2.1 behandelt im Detail den prinzipiellen Aufbau der SRAM-basierten FPGAs auf der Schaltungsebene. Abschnitt 2.2 zeigt die Einflüsse unterschiedlicher Architekturmerkmale auf die Leistungsfähigkeit und Flexibilität hierarchischer FPGAs unter Angabe aktueller Forschungsergebnisse. Abschnitt 2.3 befasst sich mit dem Entwurf digitaler Schaltungen mit FPGAs aus der Softwaresicht.

Kapitel 3 gibt einen Überblick über die einzelnen Algorithmen und Entwurfsschritte des Werkzeugs PALLAs und wie sich diese in den bestehenden Entwurfsfluss der Altera-Quartus-II-Software einfügen. Kapitel 4 beschreibt das Packen der LUTs und Flipflops einer Netzliste in die Logikzellen nach dem Less-Flexible-First-Prinzip, welches die Anzahl der zur Verfügung stehenden Packpartner berücksichtigt. In Kapitel 5 wird das neue „bottom-up"-Clustering-Verfahren ARH-Pack vorgestellt. Ausführlich betrachtet werden die neue Kostenfunktion und die ihr zugrunde gelegten Optimierungskriterien. Kapitel 6 stellt das neue Platzierverfahren MPCPlace vor.

In Kapitel 7 wird die Wirksamkeit der neu entwickelten und in den Kapiteln 4 bis 6 beschriebenen Verfahren in einem direkten Vergleich mit den bereits bekannten Verfahren aus dem akademischen Umfeld anhand mehrerer Benchmark-Schaltungen belegt. Die Ergebnisse des Vergleichs werden ausführlich diskutiert. Die Implementierungsdetails der PALLAs-Software werden in Kapitel 8 behandelt. Die Arbeit schließt mit einem Ausblick auf weitere Verbesserungsmöglichkeiten der implementierten Algorithmen und der kommerziellen Softwarepakete.

2 Theoretische Grundlagen und Stand der Technik

2.1 Prinzipieller Aufbau SRAM-basierter FPGAs

Alle FPGAs beinhalten grundsätzlich die drei wichtigsten Baugruppen: programmierbare Funktionseinheiten, programmierbare Verdrahtung und programmierbare Ein- bzw. Ausgangszellen. Kommerzielle FPGA-Architekturen verfügen in der Regel über weitere dedizierte Funktionseinheiten wie RAM-Blöcke und DSP-Einheiten. Die Funktionalität einer mit dem FPGA implementierten digitalen Schaltung wird durch die Programmierung der FPGA-Baugruppen festgelegt. Um den Baugruppenprogrammiervorgang bei FPGAs von der Programmierung eines Mikroprozessors zu unterscheiden, verwendet man häufig den Begriff der Konfiguration. Im Folgenden werden die Funktionalität und die Konfigurationsmöglichkeiten der wichtigsten FPGA-Baugruppen auf der Schaltungsebene erläutert und anschließend die prinzipielle Struktur der SRAM-basierten FPGAs vorgestellt. Hier, wie für die gesamte Arbeit, gilt die Wert-Pegel-Zuordnung der positiven Logik.

2.1.1 SRAM-Speicherzelle

Die kommerziell erfolgreichsten FPGA-Architekturen basieren auf den SRAM-Speicherzellen. Sie werden innerhalb der FPGA-Architekturen in den programmierbaren Funktionseinheiten zur Ansteuerung der Pass-Transistoren, der Leitungstreiber und als Bestandteil der Look-Up-Tables (LUTs) eingesetzt. Eine SRAM-Speicherzelle kann aus zwei mitgekoppelten Invertern aufgebaut werden (Abbildung 2-1a). Die Mitkopplung bewirkt ein bistabiles Speicherverhalten der SRAM-Speicherzelle. Durch das Anlegen einer ‚1' an die *R/W*-Steuerleitung, wird der Eingangstransistor niederohmig und der Inhalt der Speicherzelle kann über die Datenleitung *D* gelesen oder geschrieben werden. Die beiden Ausgänge *Q* und */Q* werden zu Ansteuerung der Funktionseinheiten benutzt.

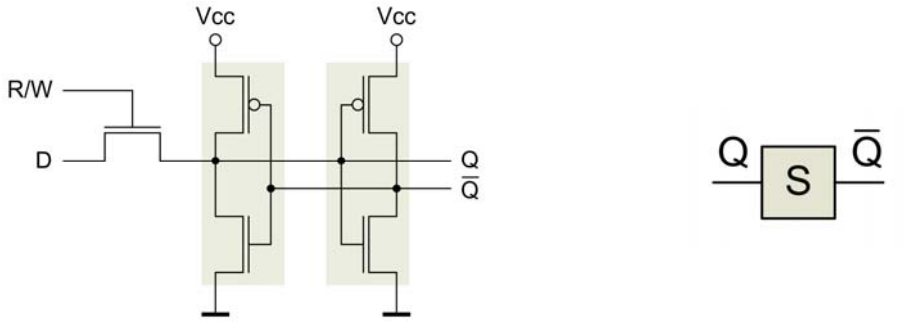


Abbildung 2-1: *Aufbau einer SRAM-Speicherzelle (a), vereinfachte Darstellung (b).*

Der Speicherinhalt einer SRAM-Speicherzelle geht verloren, sobald die Betriebsspannung *Vcc* abgeschaltet wird. Nach dem Wiederanlegen der Betriebsspannung nimmt die SRAM-Speicherzelle einen der beiden stabilen Zustände an. Da dies nichtdeterministisch erfolgt, muss zur Inbetriebnahme des FPGA die Konfiguration einmalig von außen in die SRAM-Speicherzellen geschrieben werden. Danach muss der Inhalt der SRAM-Speicherzellen, d. h. die Konfiguration, nicht mehr geändert werden, weshalb die Eingangstransistoren der SRAM-Speicherzellen über eine ‚0' an den *R/W*-Leitungen dauerhaft deaktiviert werden können – bei dem vereinfachten Symbol der SRAM-Speicherzelle wird auf die Darstellung der Steuerleitungen *R/W* und *D* verzichtet (Abbildung 2-1b).

2.1.2 Programmierbare Verdrahtung

Genau wie bei den ASICs erfolgt das Weiterleiten der Signale bei den FPGAs über die Leitungssegmente. Die Leitungssegmente sind untereinander über programmierbare Schalter verbunden, was eine selektive Weiterschaltung der Signale von einem Segment zum anderen ermöglicht. Den einfachsten programmierbaren Schalter stellt ein durch eine SRAM-Speicherzelle gesteuerter Pass-Transistor dar (Abbildung 2-2a). Der Ausgang *Q* der SRAM-Speicherzelle schaltet den Pass-Transistor entweder in einen niederohmigen oder einen hochohmigen Zustand.

Abbildung 2-2: *Programmierbare Verbindung über einen gesteuerten Pass-Transistor (a), vereinfachte Darstellung (b).*

Zur Realisierung eines Pass-Transistors können entweder NMOS- oder PMOS-Transistoren verwendet werden. Das Weiterschalten einer ‚0' erfolgt bei einem NMOS-Transistor optimal; der Signalpegel wird nahezu unverändert weitergereicht. Beim Weiterschalten einer ‚1' wird der Signalpegel von NMOS-Transistor nur degradiert weitergereicht. Bei einem PMOS-Transistor als Schalter ist das Durchschaltverhalten genau umgekehrt. Um der Signalpegel-Degradierung entgegenzuwirken, können je ein NMOS- und ein PMOS-Transistor parallel zueinander zu einem Transmission-Gatter zusammengeschaltet werden (Abbildung 2-3). Nun wird der ‚0'-Pegel vom NMOS-Transistor und der ‚1'-Pegel vom PMOS-Transistor optimal weitergereicht. Sowohl die Pass-Transistoren als auch die Transmission-Gatter weisen ein bidirektionales Durchlassverhalten auf. Die vereinfachte Darstellung der beiden Arten programmierbarer Schalter entspricht der Abbildung 2-2b.

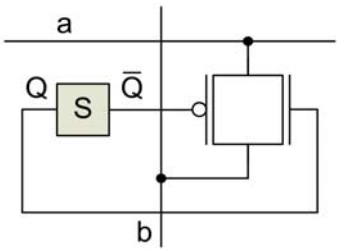


Abbildung 2-3: *Programmierbare Verbindung mit Transmission-Gatter.*

Die programmierbaren Schalter weisen im durchgeschalteten Zustand einen gewissen Ohmschen Widerstand auf. Aufgrund der eigenen parasitären Kapazität und die der Leitungssegmente können die Signalpegelwechsel über programmierbare Schalter deshalb nur zeitlich verzögert von einem Leitungssegment zum anderen weitergereicht werden. Die Verzögerungszeit steigt mit der kapazitiven Last einer Signalquelle bzw. mit der Anzahl der programmierbaren Schalter zwischen der Signalquelle und einer Signalsenke. Um dieses unerwünschte Verhalten zu mindern und das Signal zu regenerieren, können vom FPGA-Hersteller an den Anschlüssen der Leitungssegmente unidirektionale Signaltreiber eingefügt

werden. Diese sind aus zwei hintereinander geschalteten Invertern aufgebaut (Abbildung 2-4a). Trotz der Signalregeneration bilden die von den programmierbaren Schaltern verursachten Verzögerungen den größten Anteil an den Signalverzögerungszeiten [27].

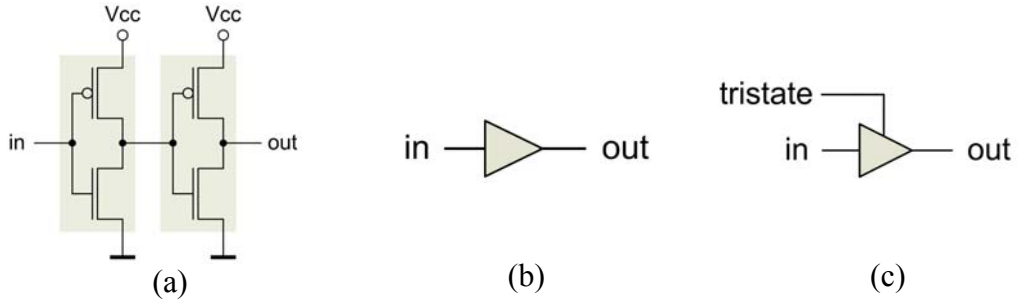


Abbildung 2-4: *Aufbau eines Leitungstreibers (a), vereinfachte Darstellung (b), vereinfachte Darstellung eines Tristate-Treibers (c).*

Ist einem Signaltreiberausgang ein Transmission-Gatter nachgeschaltet, kann der Ausgang des Signaltreibers elektrisch von dem zweiten Anschluss des Transmission-Gatters getrennt werden. Durch diese Zusammenschaltung wird ein dritter, hochohmiger Ausgangszustand ermöglicht. In diesem Fall liegt ein unidirektionaler Tristate-Treiber vor (Abbildung 2-4c). Durch eine entgegengerichtete Zusammenschaltung von zwei unidirektionalen Tristate-Treibern kann ein steuerbares bidirektionales Verhalten erreicht werden.

Bei den FPGAs sind die Leitungssegmente in den senkrechten und waagerechten Kanälen zwischen den programmierbaren Funktionseinheiten angeordnet. An den Kanalschnittpunkten befinden sich Switch-Boxen, die aus einzelnen Schaltpunkten bestehen und das Weiterschalten der Signale in einer beliebigen Richtung ermöglichen. Die Schaltpunkte sind aus mehreren Pass-Transitoren (Abbildung 2-5a) bzw. Transmission-Gattern aufgebaut.

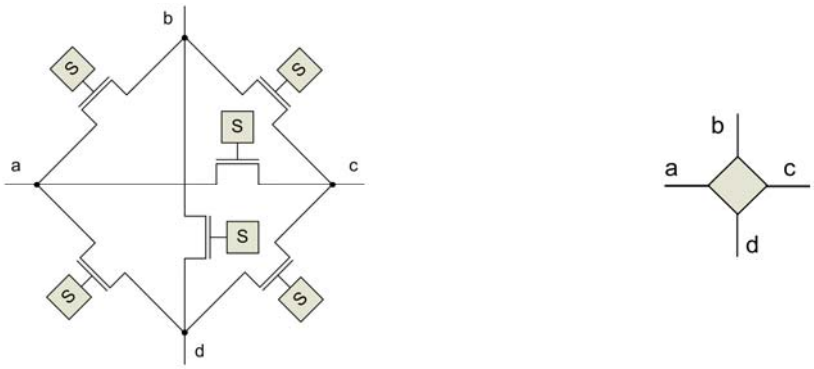


Abbildung 2-5: *Aufbau eines Schaltpunkts (a), vereinfachte Darstellung (b).*

2.1.3 Look-Up-Table

Die logische Verknüpfung der Signale erfolgt bei SRAM-basierten FPGAs in den Look-Up-Tables (LUTs). Sie sind aus einem SRAM-Speicher und einem nachgeschalteten Multiplexer aufgebaut (Abbildung 2-6a). Die Multiplexereingänge sind zugleich LUT-Eingänge. Die Art der logischen Verknüpfung der LUT-Eingänge muss vom Anwender vorgegeben werden.

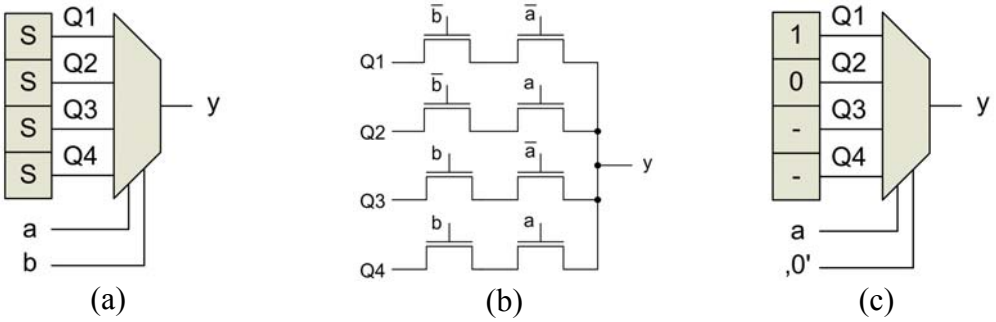


Abbildung 2-6: *Aufbau einer Look-Up-Table mit zwei Eingängen (a), Aufbau eines 4-zu-1-Multiplexers (b), mit einer LUT implementierter Inverter (c).*

Die Gesamtzahl der mit einer LUT implementierbaren logischen Funktionen ist 2^{2^n}, mit *n* als Zahl an LUT-Eingängen. Die in Abbildung 2-6a dargestellte LUT ist in der Lage, jede logische Funktion mit zwei oder einer Variablen abzubilden. Dazu werden die SRAM-Speicherzellen mit den Funktionswerten programmiert, die je nach Zustand der LUT- bzw. Multiplexereingänge *a* und *b* vom nachgeschalteten Multiplexer zum Ausgang *y* geschaltet werden. Die übliche Zahl an LUT-Eingängen bei aktuellen FPGA-Architekturen liegt zwischen 4 und 6.

Bei der Abbildung logischer Funktionen mit weniger Variablen als LUT-Eingänge vorhanden sind, werden die unbenutzten LUT-Eingänge dauerhaft auf einen definierten Wert (‚0' oder ‚1') gelegt und nur ein Teil der SRAM-Speicherzellen zur Funktionswertspeicherung benutzt. Als Beispiel zeigt Abbildung 2-6c einen Inverter für das Signal *a*, implementiert unter Verwendung einer LUT mit zwei Eingängen. Die nicht relevanten Funktionswerte sind als ‚-' (engl.: don't care) dargestellt. Soll eine Funktion mit mehr Variablen implementiert werden als LUT-Eingänge vorhanden sind, werden mehrere LUTs mit implementierten Teilfunktionen zu der gewünschten Funktion über programmierbare Verdrahtung zusammengeschaltet.

2.1.4 Logikzellen

Aus Effizienzgründen werden die in jedem Schaltwerk erforderlichen Zustandsspeicher nicht durch eine Zusammenschaltung von LUTs implementiert, sondern durch eigens zu diesem Zweck auf dem FPGA vorhandene Flipflops. So ist jedem LUT-Ausgang ein D-Flipflop (D-FF) nachgeschaltet (Abbildung 2-7). Gemeinsam bilden sie eine Logikzelle, die kleinste Einheit zur Logikabbildung. Dass die Verwendung eines Flipflops in einer Logikzelle grundsätzlich von Vorteil ist, da dadurch eine effizientere Logikabbildung von gängigen digitalen Schaltungen ermöglicht wird, wurde in [15] unter Beweis gestellt. Ob der taktsynchron zwischengespeicherte (engl.: registered) oder der aktuelle (engl.: non-registered) LUT-Ausgangswert zum Ausgang der Logikzelle gelangt, entscheidet ein programmierbarer 2-zu-1-Multiplexer (Abbildung 2-7).

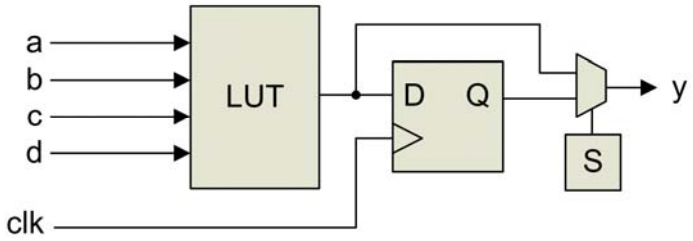


Abbildung 2-7: *Aufbau einer Logikzelle mit 4 Eingängen.*

2.1.5 Ein-/Ausgangszellen

Die Ein-/Ausgangszellen (I/O-Zellen) sind im Randbereich des FPGA angeordnet. Die primäre Aufgabe der I/O-Zellen ist die Anbindung des FPGA an die externen Leitungen bzw. Komponenten. Da in der Regel ein FPGA mit einer anderen Kernspannung betrieben wird, als die externen Schaltungskomponenten, übernehmen sie auch die Aufgabe der Logikpegelumsetzung und des Überspannungsschutzes. Die Abbildung 2-8 zeigt den prinzipiellen Aufbau einer Ein-/Ausgangszelle. Der Zustand des I/O-Pins liegt jederzeit in taktsynchron zwischengespeicherter und in der aktuellen Form vor. Die Ausgabe eines Pegels erfolgt nur beim durchgeschalteten Tristate-Treiber. Auch das Ausgangs- und Tristate-Signal können in der I/O-Zelle taktsynchron zwischengespeichert oder direkt ausgegeben werden.

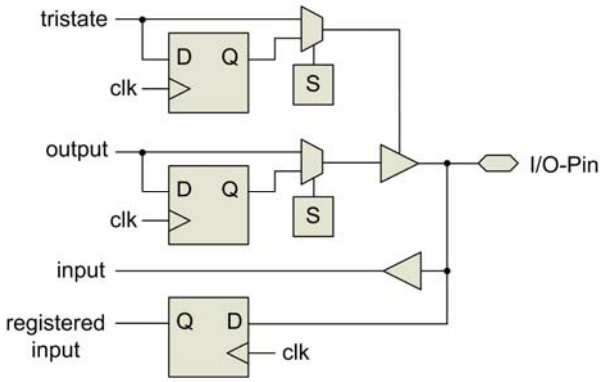


Abbildung 2-8: *Aufbau einer Ein-/Ausgangszelle.*

2.1.6 FPGA-Struktur

Alle Logikzellen eines FPGA sind zu regelmäßigen Strukturen auf dem FPGA angeordnet (Abbildung 2-9). Um die Logikzellen herum befinden sich in den Verdrahtungskanälen Leitungssegmente, die über programmierbare Schalter mit den Logikzellen verbunden werden können. Um die horizontal und vertikal verlaufenden Leitungssegmente verbinden zu können, gibt es an den Schnittpunkten der Verdrahtungskanäle Switch-Boxen mit jeweils drei Schaltpunkten. Als Kontakt zur Außenwelt befinden sich an der Peripherie des FPGA die I/O-Zellen.

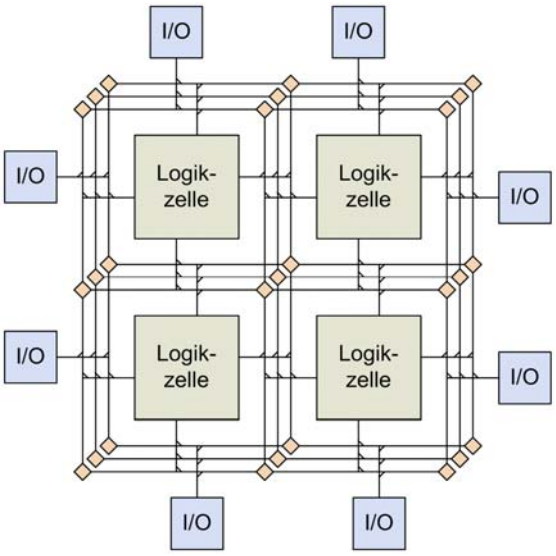


Abbildung 2-9: *Prinzipielle FPGA-Struktur.*

Im Gegensatz zu den ASICs hat der Anwender keinen Einfluss auf die Struktur und die Merkmale eines FPGA. Diese werden von den FPGA-Herstellern festgelegt. Da die Hersteller in der Regel keine eigenen Fertigungsfabriken besitzen (engl.: fabless), werden FPGAs von Dienstleistern mit eigenen Fertigungskapazitäten (engl.: foundries) als Massenware hergestellt. Aufgrund ihrer regelmäßigen Strukturen eignen sich FPGAs besonders gut, um neue Fertigungstechnologien zur Serienreife zu entwickeln.

2.2 Hierarchisch aufgebaute FPGA-Architekturen

Die kommerziell verfügbaren und in der Forschung eingesetzten SRAM-basierte FPGA-Architekturen unterscheiden sich zum Teil erheblich von der in Abschnitt 2.1 vorgestellten prinzipiellen Architektur der SRAM-basierten FPGAs. Im Folgenden werden die Architekturmerkmale hierarchisch aufgebauter FPGAs sowie die ihnen zugrunde gelegten Entscheidungen und die damit verbundenen Forschungsergebnisse vorgestellt.

2.2.1 Analyse und Bewertung der FPGA-Architekturen

Entscheidungen des FPGA-Herstellers über Struktur und Merkmale einer FPGA-Architektur beeinflussen direkt die Performance der mit dem FPGA implementierten digitalen Schaltungen. Leistungsverbrauch, maximale Taktfrequenzen und die Flexibilität bezüglich Designänderungen hängen maßgeblich von der FPGA-Architektur ab. Deshalb ist es von substanzieller Bedeutung zu verstehen, wie Aufbau und Struktur eines FPGA untersucht und bewertet werden.

Um den Einfluss eines Architektur-Parameters wie z. B. Größe der LUTs auf eine FPGA-Architektur zu untersuchen, wird in der Regel experimentell vorgegangen. Dabei werden Variationen einer FPGA-Architektur generiert, die sich nur in der Größe der zugrunde gelegten LUTs unterscheiden. Um mehrere möglichst repräsentative Designs zu platzieren und zu verdrahten, werden EDA-Werkzeuge eingesetzt. Anschließend werden die zur Implementierung dieser Designs erforderlichen FPGA-Flächen, die resultierenden maximalen Taktfrequenzen und der jeweilige Leistungsbedarf miteinander verglichen (Abbildung 2-10). Als weiterer Bewertungsfaktor wird die Verdrahtbarkeit der Designs herangezogen. Der Begriff der Verdrahtbarkeit umfasst drei wesentliche Aussagen: ob ein Design von dem Vedrahtungswerkzeug verdrahtet werden kann, wie kompliziert sich dieser Vorgang gestaltet und wie viele Verdrahtungsressourcen für eine vollständige Verdrahtung benötigt werden.

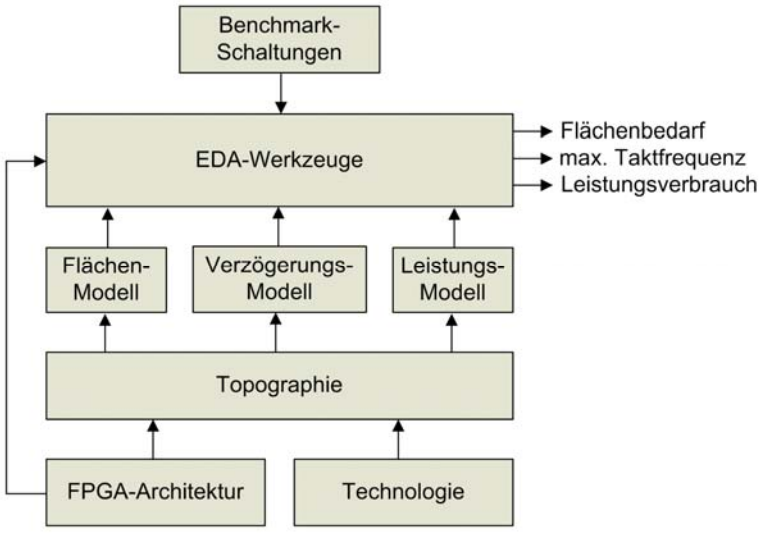


Abbildung 2-10: *Abhängigkeiten der experimentellen Analyse.*

Wie man der Abbildung 2-10 entnehmen kann, erfolgt die Analyse des Flächenbedarfs, der maximalen Taktfrequenz und des Leistungsverbrauchs auf der Grundlage der entsprechenden Modelle. Diese Modelle können aus der Topographie, d. h. der Anordnung und der Größe der Halbleiterstrukturen auf dem Silizium-Substrat extrahiert werden. Die Topographie der Halbleiterstrukturen wird ihrerseits durch die festgelegte FPGA-Architektur und durch die Technologie des ausgewählten Fertigungsprozesses bestimmt. In [05][06] wurde ein Werkzeug vorgestellt, mit dem eine FPGA-Architekturbeschreibung automatisch in die Herstellungsmasken für einen Halbleiterfertiger umgesetzt werden kann. Damit geht man der Notwendigkeit aus dem Wege, die abstakte Beschreibung einer FPGA-Architektur manuell in ein Maskenlayout umsetzen zu müssen.

Die Schwierigkeit in diesem experimentellen Analyseprozess besteht einerseits darin, eine für den praktischen Anwendungsbereich repräsentative Auswahl an unterschiedlichen Designs zu bestimmen, so dass zunächst noch unbekannte FPGA-Designs später keine zu stark abweichenden Anforderungen an die resultierende FPGA-Architektur stellen. Andererseits müssen exakte Kenntnisse über die Topographie der experimentell festgelegten FPGA-Architektur und die Auswirkungen des geplanten Fertigungsprozesses auf die der Analyse zugrunde gelegten Modellen bekannt sein. Die EDA-Werkzeuge müssen ihrerseits mit den unterschiedlichen experimentellen FPGA-Architekturen zurechtkommen und diese in jeder Variation optimal unterstützen. Ist dies nicht der Fall, kann es massive Auswirkungen auf das Analyseergebnis und die daraus resultierenden Schlussfolgerungen haben [07]. An dieser Stelle muss festgehalten werden, dass zur Auswahl einer optimalen FPGA-Architektur eine enge Zusammenarbeit und der Informationsfluss zwischen den Programmierern der EDA-Werkzeuge, den Entwicklern der FPGA-Architektur und den Halbleiterfertigern erforderlich sind.

Bei der geschilderten Vorgehensweise zur Einflussermittlung eines Architektur-Parameters wird von mehreren vollständig festgelegten FPGA-Architekturen ausgegangen. In vielen Forschungsarbeiten wird zunächst allerdings nur die Art, nicht aber die Anzahl der Verdrahtungssegmente und der Switch-Boxen in den Verdrahtungskanälen genau festgelegt. So steht erst nach Platzierung und Verdrahtung der Logikzellen der Bedarf an Verdrahtungsressourcen fest. Die erforderliche Breite der Verdrahtungskanäle wird ausgehend von diesem Bedarf bestimmt. Nach diesem Schritt liegt eine vollständig beschriebene FPGA-Architektur vor, die eine Analyse entsprechend Abbildung 2-10 ermöglicht.

2.2.2 Logikblockarchitektur

Eine FPGA-Architektur, bei der die Verdrahtungskanäle jeweils nur eine Logikzelle umgeben (vgl. Abbildung 2-9), stellt einen Sonderfall dar. Bei den hierarchisch aufgebauten FPGA-Architekturen werden mehrere Logikzellen zu einem Logikblock zusammengefasst (Abbildung 2-11). Grundlage für diese Entscheidung ist, dass bei einer Verdrahtung über kurze Distanzen nur kleine Treiber bzw. einfache Pass-Transistoren eingesetzt werden können. Das führt bei kurzen Signalwegen zu entsprechend geringen Verzögerungszeiten. Weiterhin besteht der überwiegende Anteil der zu verdrahtenden Netze aller FPGA-Designs aus kleinen Netzen mit nur 2 bis 4 Anschlüssen [04]. Diese verbinden in der Regel direkt benachbarte Logikzellen miteinander. Können Logikzellen mit hoher Interkonnektivität untereinander in einen Logikblock gepackt werden, erfolgt ein Teil der Verdrahtung innerhalb des Logikblocks unter Zuhilfenahme kurzer, lokaler Verdrahtungssegmente. Das entlastet die globalen Verdrahtungsressourcen und reduziert zugleich die Verzögerungszeiten.

Weiterhin werden mit dem Übergang zu hierarchisch aufgebauten FPGAs die Laufzeiten der Platzier- und Verdrahtungswerkzeuge reduziert. So skaliert der Aufwand der gängigen Platzieralgorithmen bei flach *oder* hierarchisch aufgebauten FPGAs mit der Anzahl der

Logikzellen *bzw.* mit der Anzahl der Logikblöcke. Bei den Verdrahtungsalgorithmen hängt der Rechenaufwand hingegen von der Anzahl der global zu verdrahtenden Netze und der Anzahl der Verbindungen pro Netz ab. Da deren Anzahl bei Verwendung von Logikblöcken sinkt, verringert sich entsprechend auch der Rechenaufwand [07].

Bei einer hierarchisch aufgebauten FPGA-Architektur wird zwischen den lokalen und globalen Verdrahtungsressourcen unterschieden. Die Verdrahtungssegmente innerhalb der Verdrahtungskanäle stellen die globalen Verdrahtungsressourcen dar, alle anderen Verdrahtungssegmente zählen zu den lokalen Verdrahtungsressourcen. Ein Logikblock bildet mit seinen lokalen und den ihn umgebenden globalen Ressourcen so genannte Kacheln. Die Kantenlänge einer Kachel wird als Längenmaß für die globalen Verdrahtungssegmente herangezogen.

Ein Logikblock ist aus *N* Logikzellen aufgebaut (Abbildung 2-11). Je nach Anzahl und Komplexität der Logikzellen in einem Logikblock und der damit realisierbarer Logik ist es üblich, entweder von der feinen Granularität (wenige bzw. einfache Logikzellen) oder von der groben Granularität (zahlreiche bzw. komplexe Logikzellen) einer FPGA-Architektur zu sprechen. Die Ausgangssignale aller *N* Logikzellen können sowohl nach Außen aus dem Logikblock zur globalen Verdrahtung als auch über einen Rückkoppelbus zurück zu den LUT-Eingängen der Logikzellen geführt werden. Befinden sich die Signalquelle und alle Signalsenken eines Netzes innerhalb desselben Logikblocks, kann die Verdrahtung dieses Netzes nur unter Zuhilfenahme der lokalen Verdrahtungsressourcen durchgeführt werden. Die globalen Verdrahtungsressourcen werden dabei nicht beansprucht.

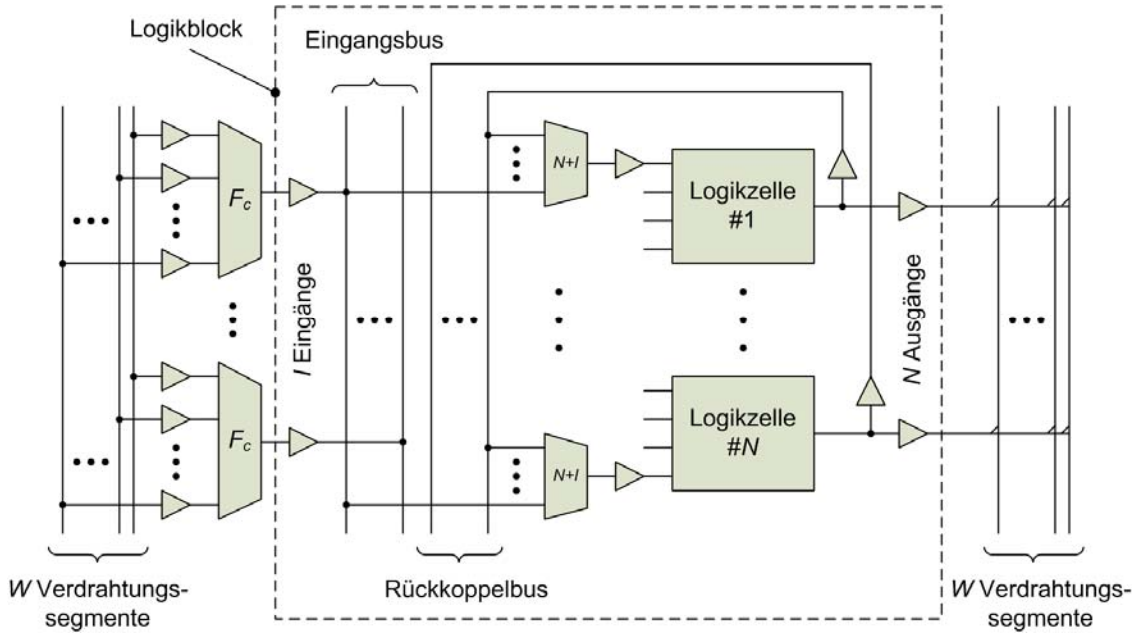


Abbildung 2-11: *Logikblockaufbau der klassischen FPGA-Architektur [14].*

Die optimale Anzahl an Logikzellen pro Logikblock hängt von der Größe der verwendeten LUTs und von der verwendeten Fertigungstechnologie ab. Wird die Anzahl oder die Komplexität der Logikzellen erhöht, steigt der Flächenbedarf einer Kachel. Dieser wiederum erhöht den Abstand zwischen den einzelnen Logikblöcken, was längere Verdrahtungssegmente zwischen den Logikblöcken erforderlich macht und so in höheren Signalverzögerungszeiten resultiert [14]. Auf der anderen Seite ermöglichen LUTs mit zahlreichen Eingängen geringere Logiktiefen, da für die Implementierung der Logikfunktionen insgesamt weniger LUTs benötigt werden, was den Anteil der logikbedingten Verzögerungszeiten

reduziert. Mit der Verkleinerung der Fertigungsstrukturen steigt jedoch der prozentuale Anteil der verdrahtungsbedingten Verzögerungszeiten gegenüber den logikbedingten Verzögerungszeiten überproportional an. Bei 130nm- und 90nm-Fertigungsprozessen beträgt der Anteil verdrahtungsbedingter Verzögerungszeiten etwa 60 bis 70% [67]. Bei einer Fertigungsprozessänderung ist demnach eine erneute Bestimmung der optimalen Logikblockgröße erforderlich. J. Rose hat im Jahr 2000 unter Berücksichtigung des damals aktuellen 0,35µm-Fertigungsprozesses festgestellt, dass die Verwendung von 4 bis 10 Logikzellen mit 4 bis 6 Eingängen in einem Logikblock eine optimale Flächennutzung mit geringen Verzögerungen erlaubt [16]. Vier Jahre später wurden bei gleicher Vorgehensweise und unter Berücksichtigung des 0,18µm-Fertigungsprozesses diese Ergebnisse bestätigt [18]. Der Nachweis der optimalen Logikblockgröße und LUT-Komplexität wurde in beiden Studien auf experimentelle Weise erbracht. Eine theoretische Ermittlung der optimalen Logikblockgröße und LUT-Komplexität wurde in [19] vorgestellt und deren Übereinstimmung mit der experimentellen Vorgehensweise erfolgreich bewiesen.

Komplexe LUTs mit zahlreichen Eingängen weisen eine geringe Flächeneffizienz auf, da bei der Abbildung einfacher Funktionen auf diese LUTs ein Teil der LUT-Eingänge und somit auch ein Teil der zur Verfügung stehenden Logikressourcen unbenutzt bleiben. Die größte Flächeneffizienz weisen LUTs mit 2 bis 3 Eingängen auf [18][19]. Deshalb wurde in [20] der Vorschlag gemacht, einen Teil der Logikzellen eines Logikblocks mit den komplexen und den anderen Teil der Logikzellen mit den einfachen LUTs auszustatten. In [21] wurde dieser Vorschlag in Form einer kommerziellen FPGA-Architektur umgesetzt. Aktuelle Stratix-III-FPGAs von Altera verwenden statt der festverdrahteten LUTs flexible Schaltungsstrukturen, die sich entweder zu einem Paar einfacher LUTs mit 3 bis 5 Eingängen oder zu einer komplexen LUT mit 6 Eingängen konfigurieren lassen [115]. Diese so genannten adaptiven Logikmodule (ALMs) weisen nur ein Flipflop für alle möglichen LUT-Konfigurationen auf, weshalb ein direkter Logikressourcenvergleich mit einer charakteristischen Logikzelle (LUT mit 4 Eingängen plus Flipflop) nicht mehr möglich ist.

2.2.3 Verdrahtungsarchitekturen

Die Verdrahtungsarchitektur wird durch Aufbau und Anordnung der programmierbaren FPGA-Verdrahtungsstrukturen festgelegt, die aus den Treibern, Verdrahtungssegmenten und programmierbaren Schaltern bestehen. Die Verdrahtungsstrukturen belegen den größten Teil einer FPGA-Fläche. Anders als bei den ASICs sind sie bei FPGAs fest vorgegeben und können von dem Designer einer digitalen Schaltung nicht mehr verändert werden. Deshalb müssen möglichst alle in der Praxis vorkommenden Designs mit einer vorliegenden FPGA-Verdrahtungsarchitektur verdrahtbar sein. Da die Interkonnektivität, d. h. der Verbindungsgrad der Logikzellen untereinander, von Design zu Design erheblich variieren kann, spielt die Flexibilität der Verdrahtungsarchitektur eine wichtige Rolle.

Die Flexibilität einer Verdrahtungsarchitektur wird durch Anzahl und Anordnung der programmierbaren Schalter bestimmt. Die Flexibilität einer Verdrahtungsarchitektur ist umso höher, je mehr programmierbare Möglichkeiten vorhanden sind, um ein Signal weiter zu leiten. Werden jedoch zu viele Verdrahtungssegmente und programmierbare Schalter auf dem FPGA vorgesehen, steigt der Abstand der Logikblöcke zueinander. Somit müssen mit globalen Verdrahtungssegmenten größere Entfernungen überbrückt werden, was zwangsläufig zu steigenden Signalverzögerungszeiten führt. Außerdem wird für die zahlreicheren Verdrahtungsstrukturen mehr Siliziumfläche benötigt, was die FPGA-Stückkosten erhöht.

Werden andererseits zu starre bzw. zu wenige Verdrahtungsstrukturen auf dem FPGA vorgesehen, muss ein Design mit hoher Interkonnektivität über eine größere FPGA-Fläche verteilt werden. In diesem Fall kann ein Teil der Logikressourcen nicht benutzt werden, da die zu deren Verdrahtung benötigten Verdrahtungsstrukturen nicht in dem erforderlichen Umfang

vorhanden sind bzw. für die Verdrahtung weiter entfernter Logikblöcke bereits belegt wurden. In diesem Fall muss die Verdrahtung über Umwege erfolgen, was wiederum zu längeren Verzögerungszeiten führt. Unabhängig davon, wie die Verdrahtungsstrukturen ausgelegt werden, wird immer entweder ein Teil der Verdrahtungsressourcen oder ein Teil der Logikressourcen nicht genutzt, da die Interkonnektivität der Logikzellen niedrig oder zu hoch ist. Die Zielsetzung bei der Festlegung einer optimalen Verdrahtungsarchitektur besteht darin, möglichst flexible und zugleich Platz sparende Verdrahtungsstrukturen zu finden, die das effiziente Verdrahten aller in der Praxis vorkommender Designs ermöglichen.

2.2.3.1 Heterogene Segmentierung der Verdrahtungsressourcen

Bei der in Abbildung 2-9 abgebildeten FPGA-Architektur erfolgt das Weiterleiten der Signale zwischen den Logikzellen über Verdrahtungssegmente der Länge Eins. Dies stellt die flexibelste, aber zugleich eine unpraktikable Lösung dar. Bei jedem FPGA-Design erfolgt ein Teil der Verdrahtung zum Teil zwischen den direkt benachbarten und zum Teil zwischen weit entfernten Logikzellen. Bei Verdrahtung weit entfernter Logikzellen müssen mehrere globale Segmente der Länge Eins (die Angaben zur Segmentlänge beziehen sich auf das Kachel-Raster) über Switch-Boxen zusammengeschaltet werden, was hohe Verzögerungszeiten zur Folge hat. Legt man sich auf eine einheitliche Segmentierung der Verbindungsleitungen mit der Segmentlänge größer als Eins fest, so können die weit entfernten Logikzellen über einige wenige miteinander verbundene Segmente verdrahtet werden. Bei Verdrahtung über kurze Entfernungen mit Hilfe langer Segmente entsteht jedoch unter Umständen ein nicht nutzbarer Segmentverschnitt. Bei den kommerziellen FPGA-Verdrahtungsarchitekturen werden in der Regel innerhalb der Verdrahtungskanäle Leitungssegmente unterschiedlicher Längen verwendet. Lokale Verbindungen werden über kurze Segmente verdrahtet, Verbindungen über weite Strecken unter Verwendung langer Segmente. Anzahl und Länge der einzelnen Segmentgruppen werden in der Regel experimentell bestimmt. Dazu werden anhand mehrerer Designs die Anforderungen an Verdrahtungsressourcen bestimmt und verschiedene Variationen der Verdrahtungsarchitekturen erzeugt. Nach der Verdrahtung der Designs werden die Verdrahtungsarchitekturen bezüglich des Flächenverbrauchs, der Signalverzögerungszeiten und der Verdrahtbarkeit analysiert und die besten ausgewählt.

Ein weiteres wichtiges Merkmal der Verdrahtungsarchitektur stellen die Anschlusspunkte der globalen Segmente an die Logikblöcke dar. Die Bestückbarkeit (engl.: population) gibt an, wie viele angrenzende Logikblöcke an ein Verdrahtungssegment angeschlossen werden können (Abbildung 2-12). Eine 100%-ige (volle) Bestückbarkeit der Segmente stellt die flexibelste Lösung dar. Da jedoch jede programmierbare Verbindung einen Mehraufwand an Fläche und eine zusätzliche kapazitive Last für die Segmenttreiber darstellt, erhöht die Verwendung vollbestückbarer Segmente die Signalverzögerungszeiten. Das Verdrahtungsmodell der entweder voll- oder nur am Rand bestückbaren (nur die äußeren Logikblöcke sind anschließbar) Verdrahtungssegmente [22] wurde in [14] um teilbestückbare Verdrahtungssegmente erweitert und weitgehend untersucht.

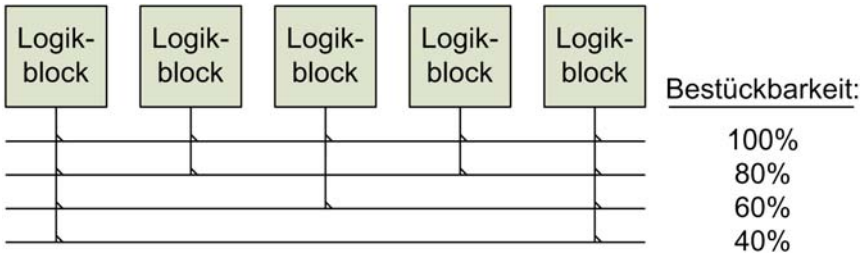


Abbildung 2-12: *Bestückbarkeit der Segmente mit der Länge 5 [14].*

Erfahrungswerte zeigen, dass der Bedarf an Verdrahtungsressourcen bei den praktischen Entwürfen zur Mitte des FPGA hin steigt. Um diesen Bedarf zu decken, wurden bei der kommerziellen ORCA-Architektur in der Mitte des FPGA breitere Kanäle mit zusätzlichen Verdrahtungssegmenten vorgesehen [23]. In [25] konnte die Wirksamkeit dieser Vorgehensweise nur für jeweils einen vertikal und einen horizontal in der Mitte angeordneten Verdrahtungskanal mit der doppelten Breite nachgewiesen werden. Einen anderen Weg beschreiten die Autoren von [24]. In ihrer Arbeit schlagen Sie eine neue FPGA-Architektur vor, bei der die Verdrahtungsressourcen im Zentrum des FPGA flexibler als zu den Rändern hin ausgelegt werden. Zu diesem Zweck werden sowohl die Segmentlängen als auch die Switch-Box-Architekturen innerhalb einer Verdrahtungsarchitektur variiert.

2.2.3.2 Logikblockanschlüsse an die globale Verdrahtung

Die Ein- und Ausgänge eines Logikblocks können über eine Anzahl von programmierbaren Schaltern direkt mit den globalen Verdrahtungssegmenten in den angrenzenden Verdrahtungskanälen verbunden werden. Die Flexibilität eines Logikblockanschlusses, F_c, gibt die Anzahl der globalen Verdrahtungssegmente an, an die dieser Logikblockanschluss angeschlossen werden kann. So ist z. B. in der Abbildung 2-12 die Flexibilität der Logikblockanschlüsse der beiden äußeren Logikblöcke $F_c = 4$ und die der drei mittleren $F_c = 2$.

In der Abbildung 2-9 sind die Ein- und Ausgänge der Logikzellen bzw. der Logikblöcke (bei den hierarchischen FPGAs) an allen vier Seiten der Logikzelle bzw. des Logikblocks angeordnet. Diese Anordnung ist bei Gleichverteilung der horizontalen und vertikalen Verdrahtungsressourcen auf dem FPGA am effizientesten [25]. Dabei kann jeweils die Logikblockseite zur Verdrahtung benutzt werden, die näher an der Signalquelle bzw. an der Signalsenke liegt. Dagegen ist die horizontale bzw. vertikale Anordnung der Ein- und Ausgänge bei doppelt so vielen horizontalen bzw. vertikalen Verdrahtungsressourcen zu bevorzugen [25]. Weiterhin wird in der gleichen Arbeit festgestellt, dass bei ausreichend großem F_c jeder Ein- oder Ausgang nur auf einer Seite des Logikblocks verfügbar sein muss, um eine möglichst große Flächeneffizienz zu erreichen. Ist F_c nur halb so groß wie die Anzahl der Segmente (W) in einem Verdrahtungskanal, bietet es sich an, für jeden Logikblockeingang bzw. jeden Logikblockausgang jeweils zwei Anschlüsse an unterschiedlichen Logikblockseiten vorzusehen.

Moderne FPGAs werden zur Implementierung digitaler Systeme eingesetzt, deren Komponenten unterschiedliche Anforderungen an die Taktfrequenzen stellen können. Um die Logikblöcke der Subsysteme eines Designs unabhängig voneinander mit Taktsignalen zu versorgen, integrieren FPGA-Hersteller auf den FPGAs dedizierte Verdrahtungsstrukturen, die die Taktverteilung sicherstellen. Werden die Anforderungen eines Systems an die Taktverteilung von diesen dedizierten Taktnetzwerken nicht erfüllt, ist die Implementierung des System nicht möglich, unabhängig davon, wie flexibel die Verdrahtungsstrukturen zur allgemeinen Signalverdrahtung ausgelegt sind. Aus diesem Grund ist der Entwurf dedizierter Taktnetzwerke von außerordentlicher Bedeutung. Folgenden Anforderungen müssen die dedizierten Taktnetzwerke genügen:

- Die Taktverteilung muss mit einem geringen Taktflankenversatz (engl.: clock skew) erfolgen. Dieser äußert sich in den unterschiedlichen Ankunftszeiten der Taktflanken eines Taktsignals an den räumlich entfernten Logikblöcken. Ein größerer Versatz führt nicht nur zu einer geringeren maximalen Taktfrequenz, sondern kann auch schwer zu bestimmende Fehlfunktionen einer Schaltung bewirken. Obwohl die H-Topologie einen geringen Taktflankenversatz aufweist, ist deren Abbildung auf ein FPGA mit Kachelstrukturen nur schwer möglich. Deshalb wird in der Regel bei den aktuellen FPGAs die „spine-and-ribs“-Topologie verwendet, bei der die Signalverteilung zunächst längs über ein Hauptsegment (spine) und dann von diesem aus quer über die

zu beiden Seiten angeordneten sekundären Segmente (ribs) zu den einzelnen Logikblöcken erfolgt [95].

- Der Leistungsbedarf der Taktnetzwerke muss gering sein. Da die Leitungskapazitäten eines aktiven Taktnetzwerks bei jeder Flanke umgeladen werden, verursachen sie einen beträchtlichen Anteil der anfallenden Verlustleistung eines FPGA-Designs [94]. Um Verlustleistung zu reduzieren, sollte eine wohlüberlegte Verdrahtungsarchitektur eine teilweise oder komplette Abschaltung unbenutzter Taktnetzwerke ermöglichen.
- Da die Anzahl der Subsysteme und deren Anordnung auf dem FPGA-Chip von Design zu Design variiert, müssen die Taktnetzwerke flexibel sein, um auch Teilbereiche eines FPGA-Chips mit unterschiedlichen Taktsignalen versorgen zu können. Bei unflexiblen Taktnetzwerken sind die EDA-Werkzeuge gezwungen, die Einschränkungen der Taktnetzwerke zu beachten, was zu ungünstigen Platzier- und Verdrahtungslösungen führen kann [95].

2.2.3.3 Switch-Box-Architekturen

Die Switch-Boxen stellen eine Ansammlung von programmierbaren Schaltpunkten bzw. Schaltern an den Kreuzungspunkten der vertikalen und horizontalen Verdrahtungskanäle dar. Sie stellen sicher, dass in ein Verdrahtungssegment eingespeistes Signal über die an beiden Segmentenden angrenzende Switch-Boxen in unterschiedliche Richtungen weitergeleitet werden kann. Die Anzahl aller möglichen programmierbaren Verbindungen eines Verdrahtungssegments zu weiteren Segmenten wird als die Switch-Box-Flexibilität, F_s, bezeichnet. Nicht nur die Flexibilität einer Switch-Box, sondern auch die Festlegung, welche Zielverdrahtungssegmente mit einem Quellverdrahtungssegment verbunden werden können, beeinflussen entscheidend die Verdrahtbarkeit und somit auch den Flächenbedarf und die Verzögerungszeiten des zu verdrahtenden Designs.

Maximale Verdrahtungsflexibilität bieten hyper-universelle Switch-Boxen. Eine Switch-Box mit W Anschlüssen an jeder Seite wird als universell bzw. *hyper-universell* bezeichnet, wenn jede Kombination von an die Switch-Box angeschlossenen Netzen durch diese Switch-Box hindurch verdrahtet werden kann. Jedes Netz kann dabei genau zwei Mal bzw. *mehrfach* an den unterschiedlichen Seiten der Switch-Box zur Signaleinspeisung oder Signalableitung angeschlossen werden. Die Switch-Boxen belegen in der Regel sehr viel Platz, weshalb es wünschenswert ist, diese mit möglichst wenigen programmierbaren Schaltern zu implementieren. Eine hyper-universelle Switch-Box, welche mit einem Minimum an programmierbaren Schaltern auskommt, wird als optimal betrachtet. In [11] wurde gezeigt, dass bereits mit einem Aufwand von 6,3W programmierbarer Schalter hyper-universelle Switch-Boxen realisierbar sind. Diese Klasse der Switch-Boxen ist zu bevorzugen, da bei praktisch auftretenden Designs ein an eine Switch-Box angeschlossenes Signal häufig gleichzeitig in mehrere Richtungen weiterzuleiten ist.

In [09] wurde festgestellt, dass für den Aufbau einer universellen Switch-Box mindestens 6W programmierbare Schalter erforderlich sind. Weiterhin wurde in der gleichen Arbeit gezeigt, dass die in der Abbildung 2-13a dargestellte, aus diagonal angeordneten Schaltpunkten aufgebaute Switch-Box weder hyper-universell noch universell ist. So können beispielsweise in der Abbildung 2-13e drei an die Switch-Box angeschlossene Netze (*a, b, c*) nicht über diese verdrahtet werden. Diese Switch-Box-Schaltung wurde unter anderem vom FPGA-Hersteller Xilinx für die FPGAs der XC4000-Familie verwendet. Da für deren Aufbau auch 6W Schalter erforderlich sind, muss deren Einsatz als suboptimal bewertet werden. Nicht jede denkbare Switch-Box-Architektur kann durch einzelne Schaltpunkte implementiert werden, deshalb hat sich eine alternative Darstellungsform aller möglichen programmierbaren Verbindungen durchgesetzt (Abbildung 2-13d).

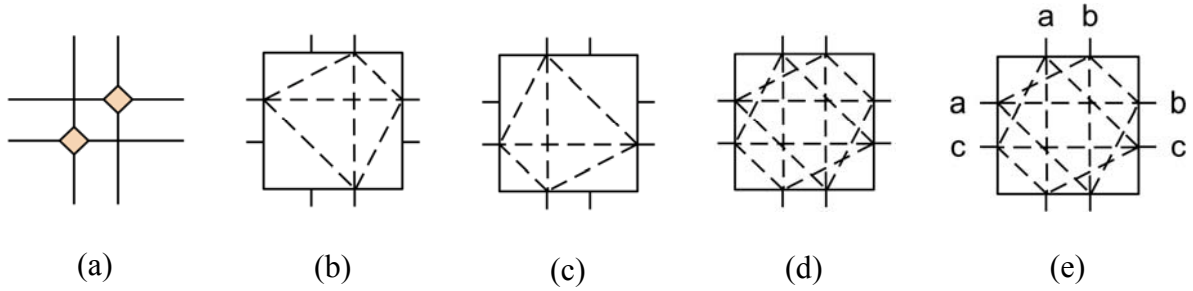


Abbildung 2-13: *Xilinx-XC4000-Switch-Box: Darstellung mit zwei Schaltpunkten (a), mögliche Verbindungen beider Schaltpunkte (b, c), Überlagerung aller möglichen Verbindungen (d), Beispiel für nicht verdrahtbare Verbindungen (e).*

In der Arbeit [09] wurde weiterhin ein allgemein gültiger Algorithmus für die Konstruktion universeller Switch-Boxen mit vier Seiten und erforderlichem Aufwand von 6W Schaltern vorgestellt. Außerdem wurde eine Methodik zur formalen Analyse der Verdrahtungskapazität einer Switch-Box eingeführt. Mit deren Hilfe wurde auch bewiesen, dass in den meisten Fällen die Flexibilität einer Switch-Box $F_s = 3$ in Verbindung mit hoher Interkonnektivität F_c ausreicht, um einen hohen Grad an Verdrahtbarkeit zu ermöglichen. Diese Vermutung wurde unter anderen auch in [14] aufgestellt und experimentell nachgewiesen. In der späteren Arbeit gleicher Autoren [10] wurden universelle Switch-Boxen mit einer von vier abweichenden Seitenzahl untersucht. In diesem Zusammenhang kann von einem Schritt zu dreidimensionalen FPGA-Verdrahtungsstrukturen gesprochen werden, die bisher aber nur von theoretischem Interesse sind.

Der Vorschlag, einen Teil der programmierbaren Verbindungen einer Switch-Box fest verdrahtet auszulegen, wurde in [13] unterbreitet. Dadurch könnten Fläche eines FPGA, Leistungsverbrauch und auch Verzögerungszeiten verdrahteter Designs gesenkt werden. Welche Verbindungen dabei als fest verdrahtet ausgelegt werden sollen, wurde anhand statistischer Analysen fertig verdrahteter Designs entschieden. In [32] schlagen die Autoren vor, einen Teil der Verbindungen zwischen den einzelnen LUTs fest zu verdrahten. Je nach Menge und Art dieser Verdrahtungsstrukturen sollten FPGA-Familien eingeführt werden, die jeweils nur bei bestimmten Designklassen zu schnelleren und kompakteren Ergebnissen führen würden. Die Entscheidung, welches Mitglied einer FPGA-Familie bei einem bestimmten Design einzusetzen ist, sollte bei diesem Konzept von einem EDA-Werkzeug getroffen werden.

In [12] werden effiziente Algorithmen zur Dimensionierung der Switch-Boxen unter Berücksichtigung der Verdrahtungssegmentierung vorgestellt. Die neu eingeführte Abschätzung der Verdrahtungsressourcennutzung kann sowohl zur Festlegung der Verdrahtungsarchitektur als auch von den Verdrahtungsalgorithmen eingesetzt werden.

2.2.3.4 Signaltreiber und programmierbare Verbindungen

Pass-Transistoren als programmierbare Verbindungen erfordern weniger Fläche als Signaltreiber, außerdem verringern sie die resultierenden Signalverzögerungszeiten bei Verdrahtungen über kurze Entfernungen. Die Analyse des Verzögerungsprofils der kritischen Pfade nach der Verdrahtung bei unterschiedlichen FPGA-Architekturen hat jedoch gezeigt, dass für den größten Teil der Signalverzögerungszeiten lange Ketten hintereinander geschalteter Pass-Transistoren verantwortlich sind [27]. Beim Entwurf einer FPGA-Verdrahtungsarchitektur besteht also die Aufgabe, festzustellen, welche programmierbaren Verbindungen der Switch-Boxen als Pass-Transistor und welche als Signaltreiber auszulegen sind. Diese Aufgabe kann nur unter Berücksichtigung der Länge und der Bestückbarkeit der

Verdrahtungssegmente gelöst werden. Liegt eine FPGA-Architektur mit vorwiegend langen Verdrahtungssegmenten und mit geringer Bestückbarkeit vor, können zur Überwindung langer Distanzen nur wenige zusammengeschaltete Segmente mit geringer kapazitiver Last eingesetzt werden. In einem solchen Szenario ist es sinnvoller, die Switch-Boxen mit mehr Pass-Transistoren als Signaltreibern auszustatten.

In [27] wurde das Modell einer nur aus Pass-Transistoren aufgebauten Switch-Box um bidirektionale Tristate-Signaltreiber erweitert. Je nach der Zeitkritizität des zu verdrahtenden Netzes kann sich das Verdrahtungswerkzeug für eine Verbindung über einen Pass-Transistor oder für eine schnellere Verbindung über einen Signaltreiber entscheiden. Als Voraussetzung für eine korrekte Entscheidung muss das Verdrahtungswerkzeug genaue Kenntnis über die Verzögerungszeiten der zur Verfügung stehenden programmierbaren Verbindungen haben. Die nachfolgende Analyse mehrerer FPGA-Architekturen hat ergeben, dass zum Aufbau einer schnellen und flächeneffizienten FPGA-Architektur Segmentlängen von 4 bis 8 und Switch-Boxen mit gleichem Anteil an Pass-Transistoren und Signaltreiber verwendet werden sollten.

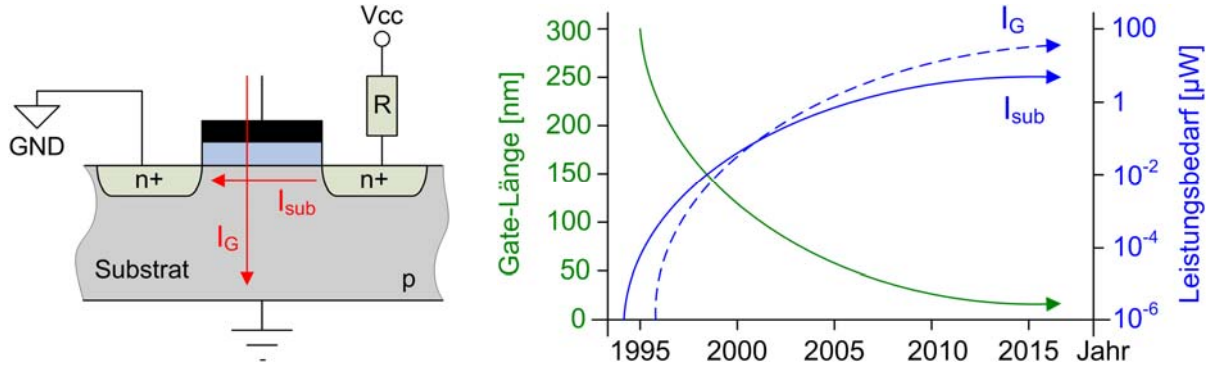
In der auf [27] aufbauenden Arbeit [28] stellten die Autoren fest, dass zusätzlich eine Verdrahtungstopologie sinnvoll ist, die eine Signalweiterleitung von über Pass-Transistoren angeschlossenen Leitungssegmenten an die über Signaltreiber gespeisten Leitungssegmente verhindert. Der umgekehrte Fall ist ausdrücklich erwünscht. Es wurde gezeigt, dass bei einer Verdrahtung zeitkritischer Netze die Signaltreiber näher an der Signalquelle liegen sollten. Das heißt, die Verdrahtung zu den Signalsenken sollte über Leitungssegmente erfolgen, die über Pass-Transistoren angeschlossen sind. Diese Vorgehensweise wird unter anderem zur Verteilung der Taktsignale mit Hilfe regelmäßiger Taktbäume verwendet.

Die bisher behandelten Leitungssegmente zur allgemeinen Signalverdrahtung besitzen keine ausgewiesene Richtung zur Signalweiterleitung, da die verwendeten Pass-Transistoren und Tristate-Signaltreiber ein bidirektionales Durchlassverhalten aufweisen. In [29] wurde die kommerzielle Altera-Stratix-FPGA-Architektur vorgestellt, die nur über unidirektionale Verdrahtungssegmente verfügt. Die Verdrahtungssegmente dieser Art haben einen ausgezeichneten Segmentanfang, der mit einem unidirektionalen Signaltreiber bestückt ist. Die Signalweiterleitung kann also nur vom Segmentanfang zum Segmentende erfolgen. Die Befürchtung, dass durch die Verwendung von unidirektionalen Verdrahtungssegmenten die Verdrahtungskanäle breiter ausgelegt werden müssten, um die gleiche Verdrahtbarkeit wie mit den bidirektionalen Verdrahtungssegmenten zu erreichen, hat sich nicht bestätigt. Es konnten nämlich die Transistoren der Switch-Boxen kleiner ausgelegt werden, da die Treiber der unidirektionalen Segmente die Aufgabe der Signalregenerierung übernahmen. Dadurch konnten die gesamte Verdrahtungsfläche sowie die resultieren Signalverzögerungszeiten sogar gesenkt werden. Aufgrund des 130nm-Fertigungsprozesses musste man bei der Stratix-Architektur jeweils auf die Pass-Transistoren komplett verzichten, da die Signalpegeldegradierung aufgrund der reduzierten Versorgungsspannung auf ein nicht akzeptables Maß anwuchs. Statt der Pass-Transistoren wurden konsequent Transmission-Gatter verwendet. Ein weiterer Vorteil der unidirektionalen Verdrahtungsarchitektur ist, dass die Switch-Boxen so aufgebaut sind, dass ein Weiterschalten der Signale nur von einer Signalquelle zu einer Signalsenke erfolgen kann. Es können also keine Kurzschlüsse auftreten, auch nicht bei falsch oder nicht konfigurierten Switch-Boxen.

2.2.4 Optimierung des Leistungsbedarfs

In der Vergangenheit waren die primären Optimierungsziele für den VLSI-Schaltungsentwurf ein geringer Flächenbedarf, hohe Taktfrequenz und hoher Integrationsgrad. Mit zunehmendem Integrationsgrad stiegen die Anzahl der Transistoren und somit auch die erzeugte Verlustleistung pro Flächeneinheit. Heute fällt bei modernen Desktop-Prozessoren eine Abwärmeleistung von mehreren Dutzenden Watt pro Quadratzentimeter Substrat an. Diese muss aufwendig abgeführt werden, um die fehlerfreie Funktion der Schaltung sicherstellen zu können. Bereits ein Temperaturanstieg von 10°C eines digitalen VLSI-Schaltkreises führt zu einer Verdoppelung der Ausfallwahrscheinlichkeit [17]. In die tragbaren Geräte, die eine der Grundsäulen der heutigen Informationsgesellschaft bilden, können keine energieaufwendigen Bauteile integriert werden, da in solchen Geräten die elektrische Energie nur in begrenzter Menge zur Verfügung steht. Aus diesen Gründen stellt der Leistungsbedarf der VLSI-Schaltungen, zu denen auch die FPGAs gehören, heute ein wichtiges Optimierungsziel bei deren Entwurf und Planung dar.

Die Ursachen für den Stromverbrauch der digitalen VLSI-Schaltungen können in zwei Kategorien unterteilt werden: den statischen und den dynamischen Stromverbrauch. Der zeitlich unabhängige statische Stromverbrauch wird durch die technologiebedingten Leckströme der Transistoren verursacht. Obwohl es belegt ist, dass FPGAs gegenüber den Mikroprozessoren und DSPs eine bessere Energieeffizienz aufweisen können [31], finden die SRAM-basierten FPGA-Architekturen kaum Einsatz in den tragbaren und auf den für niedrigen Stromverbrauch optimierten Geräten. Die Ursache dafür liegt in dem bisher hohen statischen Stromverbrauch der SRAM-basierten FPGAs. Die in den aktuellen 65nm- und 90nm-Fertigungsprozessen gefertigten Transistoren weisen gegenüber den mit früheren Prozessen gefertigten Transistoren erhöhte Leckströme auf, die nur mit zusätzlichen Fertigungsaufwand reduziert werden können. Verantwortlich dafür sind das dünnere Gate-Dielektrikum und die kürzeren Kanallängen der MOS-Transistoren (Abbildung 2-14).



(a) (b)

Abbildung 2-14: *Leckströme eines NMOS-Transistors (a), Auswirkung der Strukturverkleinerungen auf die Leckströme (b).*

Mit jeder Verkleinerung der Transistorstruktur wird die Dicke des Gate-Dielektrikums verringert und der Leckstrom durch das Gate-Dielektrikum erhöht. So ist bei dem 65nm-Fertigungsprozess das Gate-Dielektrikum ca. 1,2nm oder fünf Atomlagen dick. Durch die Verwendung eines *high-k* Dielektrikums kann die Dicke der isolierenden Gate-Oxid-Schicht bei gleichzeitiger Verringerung der Gate-Fläche gesteigert und somit der Gate-Leckstrom reduziert werden.

Eine weitere Einflussgröße für den Leistungsbedarf ist die Schwellspannung der MOS-Transistoren, sie hat Einfluss auf die Schaltgeschwindigkeit und den Unterschwell-spannungsleckstrom (engl: subthreshold-leakage current). Eine höhere Schwellspannung senkt nicht nur den Unterschwellspannungsleckstrom, sondern auch die Performance eines MOS-Transistors. So ist es seit langem Stand der Technik [91], auf den kritischen Pfaden MOS-Transistoren mit niedriger Schwellspannung und auf den unkritischen Pfaden solche mit höherer Schwellspannung zu verwenden. Die Schwellspannung wird hierbei bereits während der Fertigung durch die Strukturabmessungen und die Kanaldotierung des MOS-Transistors festgelegt. Bei den FPGAs kann eine selektive Festlegung der Transistoren-Schwellspannung nur bedingt bereits während der Fertigung erfolgen, da keine Kenntnis über die kritischen Pfade der Anwenderschaltungen vorliegt. Hier geht Altera bei den kommenden 40nm-Stratix-IV-FPGAs einen anderen Weg: Durch die Veränderung der Substrat-Vorspannung wird die Schwellspannung für jeweils alle Transistoren einer Kachel verändert [92]. Auf welche Vorspannung das Substrat einer Kachel zu legen ist, entscheidet hierbei die EDA-Software in Abhängigkeit von der Zeitkritizität der auf die Kachel abgebildeten Logik. Da durchschnittlich nur etwa 20% aller Kacheln zeitkritische Logik aufnehmen, kann durch diese Vorgehensweise der statische Leistungsverbrauch um bis zu 45% verringert werden [92].

Zum dynamischen Stromverbrauch tragen jene Ströme bei, die aufgrund der Transistorschaltaktivitäten fließen. Der zeitliche Verlauf des dynamischen Stromverbrauchs der SRAM-basierten FPGAs kann in drei zeitlich aufeinander folgende Phasen aufgeteilt werden: Einschaltvorgang, Konfigurationsvorgang und normaler Betrieb [26] (Abbildung 2-15). Beim Einschaltvorgang befinden sich die SRAM-Zellen in einem nicht definierten Zustand, was zu einer zufälligen Verdrahtung der Verdrahtungsressourcen des FPGA führt. Dies verursacht kurzzeitig zahlreiche Kurzschlüsse in den Verdrahtungsstrukturen, die erst nach der Initialisierung der SRAM-Zellen behoben werden. Bei den großen FPGAs kann der Einschaltstrom einige Ampere betragen [113]. Bei den FPGA-Architekturen des FPGA-Herstellers Altera mit unidirektionaler Verdrahtung ist kein Einschaltstromimpuls vorhanden [114], da die Architektur der Switch-Boxen und der Verdrahtungssegmente Kurzschlüsse der Signalquellen verhindert [29].

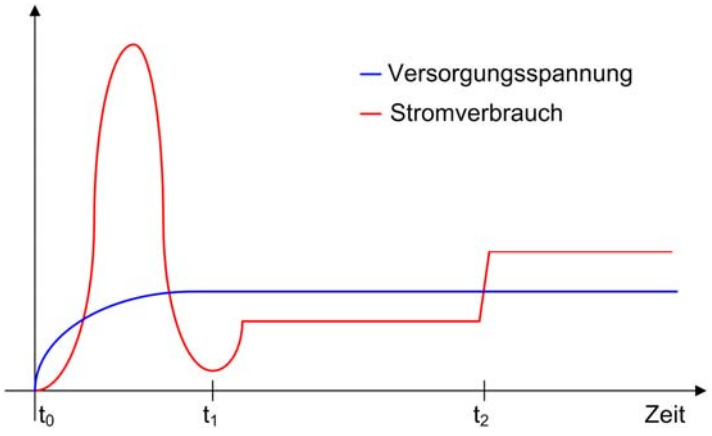


Abbildung 2-15: *FPGA-Stromverbrauch während des Einschaltvorgangs (t_0-t_1), der Konfiguration (t_1-t_2) und des normalen Betriebs (ab t_2) [26].*

Bei der Konfiguration eines FPGA werden die zu programmierenden Inhalte der SRAM-Speicherzellen aus einem externen, nichtflüchtigen Speicher geholt. Während der Konfiguration ist ein SRAM-FPGA zum einen Teil mit Benutzerdaten konfiguriert und zum anderen Teil noch im Initialzustand. Dies führt ebenfalls zu einer Anzahl von Kurzschlüssen

der Verdrahtung, wenn auch in wesentlich geringerem Umfang als in der Initialisierungsphase. Nach der Konfiguration folgt die Betriebsphase, die bis zum Ausschalten der FPGA-Spannungsversorgung andauert (Abbildung 2-15).

Während des Betriebs eines FPGA werden mit jedem Schaltvorgang der Treiber die parasitären Kapazitäten der angeschlossenen Verdrahtungssegmente und der Transistoren umgeladen. Der daraus resultierende dynamische Leistungsverbrauch ist proportional zur Anzahl der Schaltvorgänge pro Zeiteinheit, zu den parasitären Kapazitäten der Verdrahtungssegmente, zum Kurzschlussstrom der Transistoren und dem Quadrat der Versorgungsspannung [30]:

$$P_{dynamic} = \left[\frac{1}{2} \cdot C \cdot V^2 + I_{ShortCircuit} \cdot V \right] \cdot f \cdot activity$$

$P_{dynamic}$	- dynamischer Leistungsverbrauch
C	- Gesamtheit der parasitären Kapazitäten
V	- Betriebsspannung
$I_{ShortCircuit}$	- Kurzschlussstrom während der Transistorschaltvorgänge
f	- Taktfrequenz
$activity$	- durchschnittliche Schaltaktivität [0,0…1,0]

Um die Anzahl der Schaltvorgänge pro Zeiteinheit zu reduzieren, bietet es sich an, das Taktsignal für die nicht benötigten Komponenten der mit dem FPGA implementierten Schaltung zu deaktivieren. Neuere FPGA-Architekturen verfügen dazu über Schaltungsstrukturen zur dynamischen An- und Abschaltung ganzer Taktbäume (engl.: clock gating), ohne Störimpulse während des Umschaltvorgangs befürchten zu müssen. Weiterhin ist in der Regel die Möglichkeit gegeben, die Taktfrequenz dynamisch, d. h. während des Betriebs anpassen zu können. So kann während der Phasen geringer Aktivität die Taktfrequenz gesenkt und während der Phasen hoher Aktivität angehoben werden.

Durch die Verringerung der Versorgungsspannung kann sowohl die dynamische als auch die statische Verlustleistung reduziert werden. Aufgrund des quadratischen Einflusses auf die Verlustleistung ist die Reduzierung der Versorgungsspannung der effektivste Weg, die gesamte Verlustleistung zu minimieren. So kann durch die Halbierung der Versorgungsspannung die Verlustleistung auf ein Viertel abgesenkt werden. Ungünstigerweise hat die Reduzierung der Versorgungsspannung erhöhte Verzögerungszeiten zur Folge. So werden beispielsweise durch die Verringerung der Versorgungsspannung von 1,5 auf 1,2 Volt die Logikverzögerungszeiten der Igloo-FPGAs nahezu verdoppelt [112]. Bleibt hingegen durch eine Fertigungsprozessanpassung das Verhältnis der Schwellspannung der Transistoren zur Versorgungsspannung konstant, so kann die Versorgungsspannung ohne merklich angestiegene Verzögerungszeiten reduziert werden [30].

2.2.5 Dedizierte Funktionseinheiten

Wird eine digitale Schaltung sowohl mit einem ASIC als auch mit einem FPGA realisiert, so weist die FPGA-Implementierung der Schaltung deutlich mehr Silizium-Fläche, höheren Leistungsbedarf und eine niedrigere maximale Taktfrequenz auf. In den letzten Jahren wurden einige Versuche unternommen, den Leistungsunterschied zwischen ASICs und FPGAs quantitativ zu ermitteln [33][34][35]. In [35] wurden mehr als 20 unterschiedliche Schaltungen aus der Praxis sowohl mit einem 90nm-ASIC als auch einem 90nm-FPGA implementiert und miteinander verglichen. Dabei stellte sich heraus, dass die mit dem FPGA implementierten Schaltungen im Schnitt bis zu vierzigfach höheren Flächenbedarf hatten, drei

Mal so langsam waren und zwölf Mal mehr Energie benötigten wie die ASIC-basierten Pendants.

Die FPGA-Hersteller versuchen, das angesprochene Leistungsgefälle zwischen FPGAs und ASICs zu mindern, indem sie auf den FPGAs dedizierte Funktionseinheiten (engl.: hard-IPs) für die häufig vorkommende Schaltungsklassen fest verdrahtet implementieren. So erfordern viele Designs Speicherstrukturen in Form von FIFOs, RAMs oder Schieberegistern. Um diesem Sachverhalt Rechnung zu tragen, wurden bei den kommerziellen FPGA-Architekturen bereits relativ früh RAM-Blöcke vorgesehen [53]. Diese können in der Regel als Single- bzw. Dual-Port-RAMs, als FIFOs mit unterschiedlichen Wortbreiten und auch als sehr lange Schieberegister konfiguriert werden. Der Einsatz von dedizierten Speicherstrukturen verbesserte die Flächeneffizienz der FPGAs bei praktischen Designs um ca. 10% [35]. Diese nur geringe Verbesserung der Flächeneffizienz ist darauf zurückzuführen, dass bei der zum Vergleich herangezogenen FPGA-Architektur die einzelnen LUTs sich als 16x1-RAMs konfigurieren und nutzen lassen (engl.: distributed ram). Andererseits lassen sich die RAM-Blöcke auch als extrem breite LUTs verwenden. So wird in [36] der Vorschlag gemacht, die unbenutzten RAM-Blöcke eines Designs explizit als LUT-Ersatz zu verwenden, um zum einen die Logikressourcen zu entlasten und zum anderen die maximale Taktfrequenz der implementierten Designs zu erhöhen.

Der Speicherstrukturintegration auf den kommerziellen FPGAs folgte kurze Zeit später die Integration von eingebetteten Multiplizier- bzw. DSP-Einheiten im Festkommaformat. Diese ermöglichen, die in der digitalen Signalverarbeitung sehr oft vorkommende Multiplikation zweier Zahlen mit anschließender Addition zum vorhandenen Ergebnis (engl.: multiply-accumulate) sehr effizient abzubilden. In der Kombination mit den RAM-Blöcken stieg die Flächeneffizienz bei DSP-lastigen Designs im Durchschnitt auf das Doppelte [35].

Der Trend zur Integration von dedizierten Funktionseinheiten setzt sich weiter fort. So sind auf einigen FPGAs des Herstellers Xilinx bereits bis zu zwei PowerPC-Prozessoren für Steuerungsaufgaben enthalten. Die internen Busse dieser Prozessoren lassen sich flexibel über die Verdrahtungsressourcen des FPGA mit den internen Logikressourcen des FPGA verbinden. Man könnte in diesem Fall sogar von der Integration eines Prozessors in einen flexibel programmierbaren Coprozessor sprechen. Durch das steigende Daten-verarbeitungspotenzial der FPGAs steigt auch der Bedarf an Schnittstellen, über die schnell große Datenmengen zur Bearbeitung in den FPGA transportiert werden können. Deshalb werden leistungsfähige Punkt-zu-Punkt-Schnittstellen zur Übertragung von mehreren GBit pro Sekunde auf den FPGAs integriert. Zwangsläufig erfolgt durch die Integration von unterschiedlichsten Funktionseinheiten eine Aufgabenspezialisierung von FPGAs, die durch die Einführung von FPGA-Familien noch verstärkt wurde.

2.2.6 Kommerzielle FPGA-Familien

Die führenden Hersteller SRAM-basierter FPGAs Altera und Xilinx unterteilen ihre FPGAs in Familien, deren Mitglieder untereinander gleiche Hardwaremerkmale aber unterschiedliche Ressourcenkapazitäten aufweisen. Die verschiedenen Familien wiederum unterscheiden sich in den Hardwaremerkmalen und den Fertigungstechnologien. Dadurch ergibt sich eine zwangsläufige Spezialisierung der einzelnen FPGA-Familien auf bestimmte Aufgabengebiete. An dieser Stelle soll die aktuelle FPGA-Produktpalette von Xilinx und von Altera genauer betrachtet werden.

Die nachfolgenden Tabellen bieten einen direkten Vergleich der Hardwareressourcen unterschiedlicher Mitglieder der FPGA-Familien des oberen Preissegments von Xilinx (Tabelle 2-1) und Altera (Tabelle 2-2). Die einzelnen Tabellenangaben geben die Anzahl der entsprechenden FPGA-Ressourcen sowohl für das kleinste als auch für das größte Mitglied

einer FPGA-Familie an. Die Logikzellen der verglichenen FPGA-Architekturen weisen einen von der charakteristischen Logikzelle (LUT mit 4 Eingängen und Flipflop) abweichenden Aufbau auf. Deshalb geben die Hersteller sowohl die logikäquivalente Anzahl der LUTs mit 4 Eingängen als auch die Anzahl der Flipflops an. Weiterhin sind in den beiden Tabellen die seriellen Hochgeschwindigkeitsschnittstellen unabhängig vom Übertragungsstandard zusammengefasst, wenn diese Schnittstellen eine höhere Übertragungsgeschwindigkeit als 1,25GBit/s ermöglichen. Die langsameren Ein- und Ausgänge der FPGAs werden zu den Standard-I/Os hinzugerechnet.

	Virtex 5 LX	**Virtex 5 LXT**	**Virtex 5 SXT**
Technologie	65nm-Prozess, 12 Metalllagen, 1,1 Volt Versorgungsspannung		
LUT-Äquivalente	30.720-33.1776	19.968-33.1776	34.816-94.208
Flipflops	19.200-20.7360	12.480-20.7360	21.760-58.880
Block-RAM (KBit)	1.152-10.368	936-11.664	3.024-8.784
DSP-Blöcke	32-192	24-192	192-640
Taktgenerierung	6-18	3-18	6-18
High Speed-I/Os	0	7-29	13-21
Standard-I/Os	400-1.200	160-960	360-640
Mitgliederzahl	7	8	3

Tabelle 2-1: *Hardwareressourcen der Mitglieder von Xilinx-Virtex5 LX-, LXT- und SXT-Familien.*

Der Hersteller Xilinx sieht für seine FPGA-Bausteine der Virtex5-Familien Zielanwendungen besonders im Bereich LUT-intensiver Designs (Virtex 5 LX), LUT-intensiver Designs mit Bedarf an schnellen seriellen Schnittstellen (Virtex 5 LXT) und Designs der digitaler Signalverarbeitung mit Bedarf an schnellen seriellen Schnittstellen (Virtex 5 SXT). Der Anwender kann so, je nach Art seiner Anwendung, zwischen unterschiedlich ausgestatteten FPGA-Familien und deren Mitgliedern wählen.

Der Hersteller Altera verfolgt mit seinen Stratix-III-Familien eine ähnliche Strategie. Hier erfolgt eine Unterteilung der FPGAs in Familien, die dem Anwender besonders viele LUT-Logikressourcen zur Verfügung stellen (Stratix III L) bzw. speicher- und rechenintensive DSP-Designs aufnehmen können (Stratix III E). Die beiden Stratix-Familien besitzen gleichermaßen serielle Schnittstellen mit bis 1,6 GBit/s.

	Stratix III L	**Stratix III E**
Technologie	65nm-Prozess, 12 Metalllagen, 0,9 bis 1,1 Volt Versorgungsspannung	
LUT-Äquivalente	47.500-338.000	47.500-254.400
Flipflops	38.000-270.400	203.520
Block-RAM (KBit)	1.836-16.272	5.328-14.668
DSP-Blöcke	216-576	384-768
Taktgenerierung	4-12	4-12
High Speed-I/Os	72-276	72-276
Standard-I/Os	228-1.104	288-960
Mitgliederzahl	7	4

Tabelle 2-2: *Hardwareressourcen der Mitglieder Altera-Stratix-III L- und E-Familien.*

Für die kostensensitiven Anwendungen bieten beide Hersteller low-cost FPGAs an, die insgesamt über weniger zahlreiche dedizierte Funktionseinheiten verfügen. Diesen FPGAs fehlen die schnellen seriellen Schnittstellen gänzlich. Ein wesentliches Merkmal dieser preiswerten FPGAs ist weiterhin, dass sie mit weniger Verdrahtungsressourcen gefertigt werden, da im Vergleich zu FPGAs des oberen Preissegments weniger Metallisierungslagen zur Verfügung stehen. So werden die Bausteine der Cyclone-III-Familie mit neun, die Stratix-III-Bausteine mit insgesamt zwölf Metalllagen gefertigt. Da bei den low-cost FPGAs die SRAM-Konfigurationszellen für die nicht mehr vorhandenen Verdrahtungssegmente entfallen, wird pro implementierbare Gatterfunktion die notwendige Silizium-Fläche gesenkt. In der nachfolgenden Tabelle 2-3 sind die Hardwareressourcen der Altera-Cyclone-III-FPGAs und der Xilinx-Spartan-3-(DSP)-FPGAs zusammengefasst.

	Altera Cyclone III	**Xilinx Spartan 3**	**Xilinx Spartan 3 DSP**
Technologie	65nm-Prozess, 9 Metalllagen, 1,2 Volt Versorgungs-spannung	90nm-Prozess, 8 Metalllagen, 1,2 Volt Versorgungsspannung	
Logikzellen	5.136-119.088	1728-74.880	37.440-53.712
Block-RAM (KBit)	414-3.888	72-1.872	1.512-2.268
DSP-Blöcke	23-288	4-104	84-126
Taktgenerierung	2-4	2-4	8
Standard-I/Os	182-533	124-784	309-512
Mitgliederzahl	8	8	2

Tabelle 2-3: *Hardwareressourcen der FPGAs für kostensensitive Anwendungen.*

Anders als Altera führt Xilinx für die unterschiedlichen Anwendungsbereiche des preissensitiven Marktsegments mehrere FPGA-Familien im Programm. Aus Gründen der besseren Übersichtlichkeit sind in Tabelle 2-3 nur die Spartan-3-Familien für die LUT- und die DSP-intensiven Anwendungen aufgeführt.

2.3 Entwurf digitaler Schaltungen mit SRAM-basierten FPGAs

In diesem Abschnitt wird der Entwurf digitaler Schaltungen mit SRAM-basierten hierarchischen FPGAs in einer ganzheitlichen Perspektive dargestellt. Es wird neben der Betrachtung der Abstraktionsebenen und der Entwurfssichtweisen des VLSI-Schaltungsentwurfs ein FPGA-Entwurfsfluss mit den möglichen Optimierungszielen und den einzelnen Entwurfsschritten im Detail betrachtet. Die gegenseitigen Abhängigkeiten der einzelnen Entwurfsschritte werden aufgezeigt und anhand der Ergebnisse aktueller wissenschaftlicher Arbeiten belegt.

2.3.1 Abstraktionsebenen des Y-Diagramms

Nach [88] können die einzelnen Abstraktionsebenen des VLSI-Schaltungsentwurfs aus drei unterschiedlichen Perspektiven in einem Y-Diagramm dargestellt werden (Abbildung 2-16).

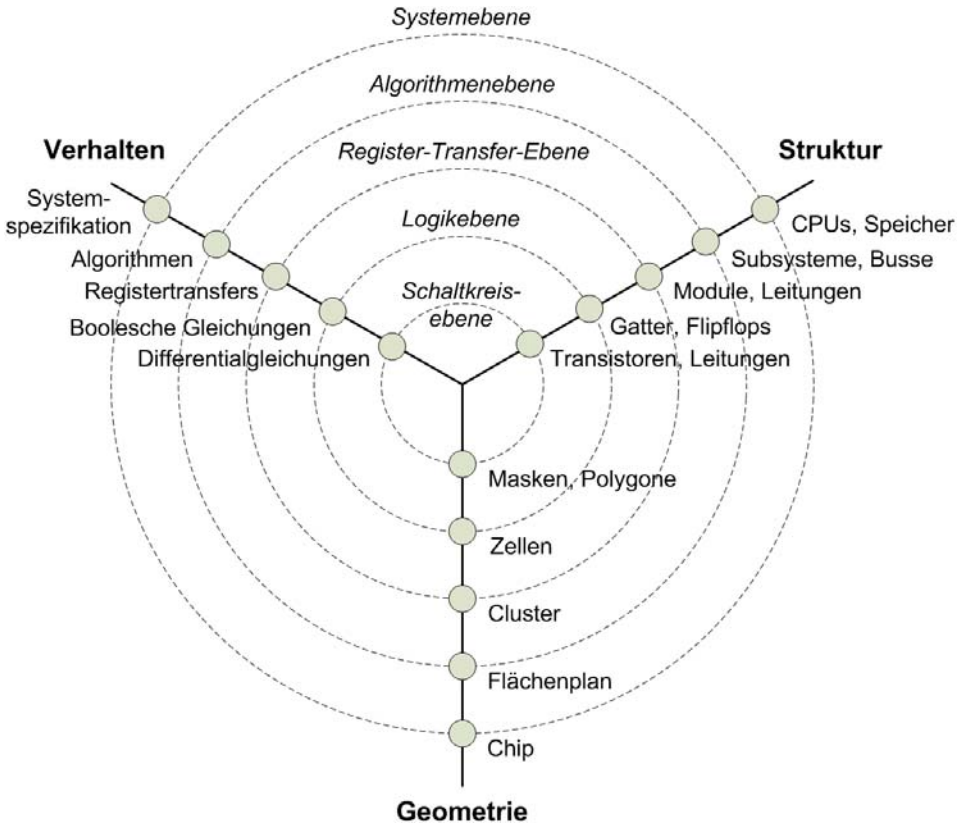


Abbildung 2-16: *Abstraktionsebenen und Entwurfsperspektiven im Y-Diagramm [88].*

Im Y-Diagramm werden die unterschiedlichen Abstraktionsebenen als konzentrische Kreise veranschaulicht, wobei der Abstraktionsgrad mit den weiter außen liegenden Kreisen zunimmt. Die Abstraktionsebenen repräsentieren Modelle einer integrierten Schaltung mit unterschiedlichem Grad an Implementierungsdetails. Durch Konkretisierung (Verfeinerung) erfolgt ein Übergang von einer höheren zu einer niedrigeren Abstraktionsebene innerhalb einer Perspektive und somit zu einem Schaltungsmodell mit mehr Implementierungsdetails. Die unterschiedlichen Perspektiven des Entwurfs werden im Y-Diagramm als Äste dargestellt; dabei wird zwischen Verhalten, Struktur und Geometrie unterschieden. Durch Transformation einer Schaltungsbeschreibung erfolgt der Übergang von einer Perspektive zu einer anderen, und zwar innerhalb der gleichen Abstraktionsebene – mit geringen Einschränkungen kann das Y-Diagramm des VLSI-Schaltungsentwurfs auch zur Darstellung der Abstraktionsebenen und Perspektiven des FPGA-Schaltungsentwurfs verwendet werden.

- Auf der **Systemebene** ist das Verhalten einer Schaltung in einer Systemspezifikation z. B. in Form eines Pflichtenhefts festgelegt. Die zu verwendenden Funktionseinheiten sind in einem Blockdiagramm festgehalten und repräsentieren die strukturelle Sicht. Die Abschätzung der für die Implementierung der Schaltung erforderlichen Chipfläche stellt die geometrische Sicht dar. Auf der Systemebene wird vom detaillierten Zusammenspiel und vom zeitlichen Verhalten der einzelnen Funktionseinheiten abstrahiert.
- Auf der **Algorithmenebene** ist das Verhalten einer Schaltung mit Hilfe nebenläufiger Algorithmen beschrieben. Die strukturelle Sicht repräsentiert Recheneinheiten, die zur Ausführung dieser Algorithmen benötigt werden und die miteinander über Signale kommunizieren. Ein Flächenplan weist den Recheneinheiten die exakte Position und Fläche auf den Chip zu und stellt die geometrische Sicht dar.
- Auf der **Register-Transfer-Ebene** (RTL) ist das Verhalten einer Schaltung durch die Datentransfers zwischen den Datenregistern beschrieben, auf die die grundlegenden Manipulationen angewendet werden. Auf der Strukturebene sind die erforderlichen Register, Multiplexer, (De-) Kodierer und ALUs durch Signale miteinander verknüpft. Cluster fassen topologisch benachbarte Strukturen aus der Geometrieperspektive zusammen. Das Modell einer digitalen Schaltung auf der Register-Transfer-Ebene ist taktgenau, somit kann der Datendurchsatz einzelner Recheneinheiten genau bestimmt werden.
- Auf der **Logikebene** ist das Verhalten einer Schaltung durch Boolesche Gleichungen und das zeitliche Verhalten einzelner Signale beschrieben. Aus der strukturellen Sicht wird die Schaltung durch Zusammenschaltung der Grundgatter (Netzliste) repräsentiert. Beim FPGA-Entwurf besteht die Netzliste aus zusammen geschalteten LUTs, Flipflops und eventuell vorhandenen dedizierten Funktionseinheiten.
- Den höchsten Grad an Implementierungsdetails weist die **Schaltkreisebene** auf. Hier sind bereits Positionen und Abmessungen eines jeden Schaltungselements und jeder Leitung auf dem Chip durch die Fertigungsmasken festgelegt. Das zeitliche und nichtlineare elektrische Verhalten der Schaltungselemente und der Leitungen kann mit Hilfe von Differenzialgleichungen beschrieben werden – beim Entwurf digitaler Schaltungen mit FPGAs kann nur eingeschränkt von der Schaltkreisebene gesprochen werden, diese wird alleine durch die Konfigurationsdaten für alle SRAM-Speicherzellen des Ziel-FPGAs repräsentiert.

2.3.2 Entwurfsfluss

Die steigenden Komplexitäten der FPGAs und die damit realisierbaren digitalen Schaltungen stellen eine Herausforderung an den Entwickler dar. Die Funktionalität einer digitalen Schaltung wird bei einem SRAM-basierten FPGA durch die Konfiguration aller SRAM-Speicherzellen festgelegt. Im Einzelnen müssen dazu bei aktuellen FPGAs die Zustände von einigen hundert Tausenden bis Millionen SRAM-Speicherzellen bestimmt werden. Das kann nicht manuell erfolgen, sondern erfordert Entwurfswerkzeuge, die eine Abstraktion weg von der Schaltkreisebene ermöglichen. Die Schaltungsmodellierung auf den höheren Abstraktionsebenen bietet den Vorteil der Ausblendung der für die Funktion der Schaltung nicht relevanten Details. Dadurch können mit weniger Zeilen Code komplexere Schaltungen innerhalb kürzerer Zeit, als das auf den niedrigeren Abstraktionsebenen möglich wäre, beschrieben werden. Hierzu kommen Hardwarebeschreibungssprachen wie Verilog [102] oder VHDL [101] auf der Register-Transfer-Ebene und auch optional oder ergänzend Programmiersprachen wie C/C++ [99] auf der Algorithmenebene zum Einsatz. Die Konkretisierung und die Transformation der Verhaltensbeschreibung auf der Algorithmen-

ebene zur Strukturbeschreibung auf der Register-Transfer-Ebene erfolgt mit Hilfe der High-Level-Synthese-Werkzeuge (HLS). Die Verhaltens- und Strukturbeschreibungen auf der Register-Transfer-Ebene wiederum werden mit den RTL- und Logiksynthese-Werkzeugen zu einer Strukturbeschreibung auf der Logikebene in Form einer Gatternetzliste umgesetzt (Abbildung 2-17). Weitere automatische Werkzeuge kommen bei den nachfolgenden Entwurfsschritten bis zum Erstellen einer Konfigurationsdatei für den FPGA zum Einsatz.

Während der RTL- und Logiksynthese wird die HDL-Schaltungsbeschreibung in eine zieltechnologiekonforme Netzliste, bestehend aus LUTs und Flipflops, umgesetzt. Anschließend erfolgt das Packen der LUTs und Flipflops in die Logikzellen. Beim Clustering wird eine Zuordnung der Logikzellen zu Logikblöcken durchgeführt. Die Anzahl der erforderlichen Logikblöcke auf dem FPGA ist erst nach diesem Entwurfsschritt bekannt, da in der Regel nicht jeder Logikblock bis an seine Kapazitätsgrenzen gepackt werden kann. Die Flächenplanung ist ein optionaler Entwurfsschritt, bei dem den einzelnen Komponenten oder Funktionsblöcken der Schaltung Bereiche auf dem FPGA-Chip zugewiesen werden. Die Flächenplanung kann entweder automatisch oder manuell vom Entwickler durchgeführt werden. An dem erzeugten Flächenplan richtet sich die Platzierung der Logikblöcke. Nach der Platzierung, die jedem Logikblock eine Position auf dem FPGA zugewiesen hat, kann die Verdrahtung der Logikblöcke und der darin enthaltenen LUTs und Flipflops erfolgen. Da erst nach dieser eine exakte Kenntnis über die genauen Signalpfade vorliegt, kann nun ein exaktes Timing-Modell der Schaltung erzeugt werden (Abbildung 2-17).

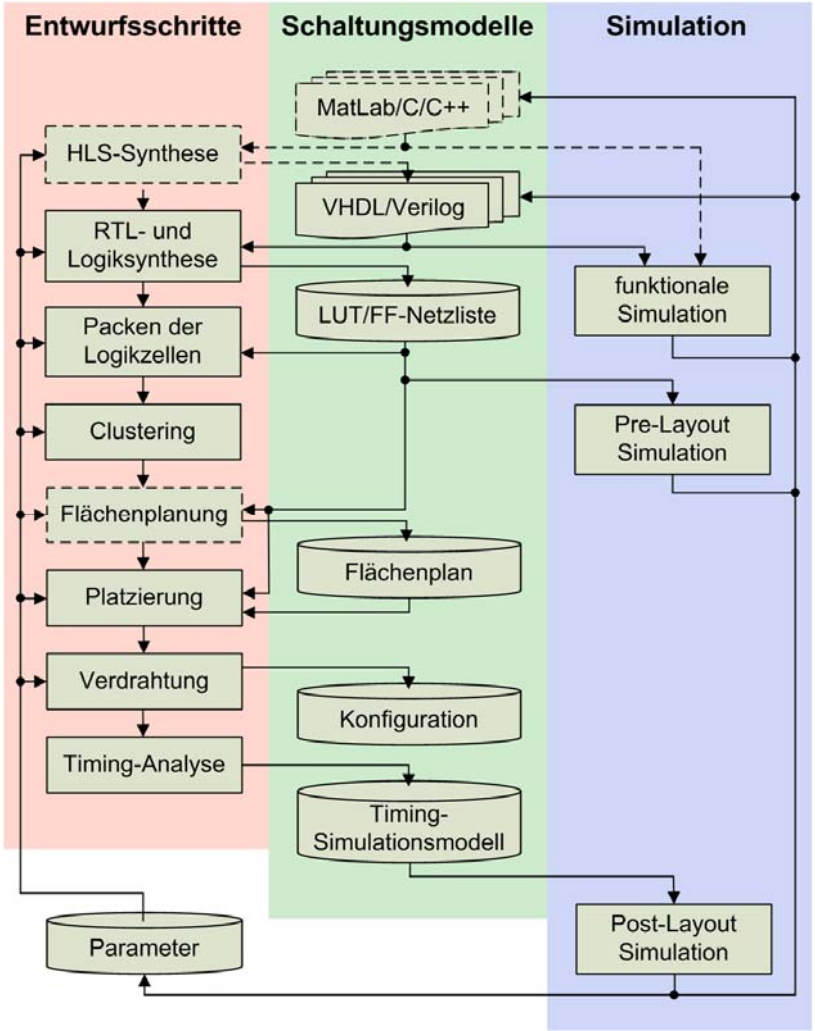


Abbildung 2-17: *Typischer FPGA-Entwurfsfluss.*

Zur Validierung des Designs ist es erforderlich, dass sowohl die ursprünglichen als auch die automatisch erzeugten Schaltungsmodelle vom Entwickler auf korrekte Funktion untersucht werden können. Hierzu werden Simulationswerkzeuge eingesetzt, die ein Schaltungsmodell mit den vom Entwickler festgelegten Eingangssignalen stimulieren und die anhand des Schaltungsmodells erzeugten Ausgangssignale mit den Sollwerten vergleichen bzw. graphisch aufbereiten. Ein Fehler kann mit der Simulation nur dann entdeckt werden, wenn die vom Entwickler vorgesehenen Stimuli diesen Fehler tatsächlich provozieren können. Eine Überprüfung aller möglichen Stimuli und somit der Nachweis, dass eine Schaltung fehlerfrei ist, kann aus Zeitgründen nur bei sehr einfachen kombinatorischen Schaltungen erfolgen. In allen anderen Fällen müssen die Stimuli auf eine in akzeptabler Zeit simulierbare Anzahl an Eingangsvektoren, die möglichst viele Fehler provozieren können, reduziert werden.

Eine Simulation kann beim FPGA-Entwurf an mehreren Stellen, jedoch mit unterschiedlichem Detailgrad erfolgen (vgl. Abbildung 2-17). Wird die algorithmische Beschreibung einer Schaltung simuliert, so kann nur die Funktion und nicht das zeitliche Verhalten der Schaltung überprüft werden. Bei einer RTL-Beschreibung kann zusätzlich das taktgenaue Verhalten der Schaltung simuliert werden. Wenn nach dem Technologie-Mapping eine Netzliste aus LUTs und Flipflops vorliegt, können bereits die Logik-Signalverzögerungszeiten berücksichtigt werden. Die Logik-Signalverzögerungszeiten haben jedoch einen geringen Anteil an der gesamten Verzögerungszeit [67]. Die genauen Verzögerungszeiten können erst nach der abgeschlossener Platzierung und Verdrahtung ermittelt werden, deshalb spricht man an dieser Stelle auch von der Pre-Layout-Simulation. Nach einer Timing-Analyse der verdrahteten Schaltung liegen sowohl die verdrahtungs- als auch logikbedingte Signalverzögerungszeiten in einem Timing-Simulationsmodell vor, welches dann zur Takt- und signalverzögerungszeitengenauen Post-Layout-Simulation benutzt werden kann.

Stellt der Anwender nach der funktionalen Simulation eine Fehlfunktion der Schaltung fest, so muss er die Schaltungsmodelle auf der Algorithmen- oder RTL-Ebene korrigieren. Die Post-Layout-Simulation kann zusätzlich zu den funktionalen Fehlern auch die Überschreitung von festgelegten maximalen Signalverzögerungszeiten aufzeigen. In diesem Fall ist die Veränderung der Schaltungsbeschreibungen der Algorithmen- und RTL-Ebene erforderlich, und zwar so, dass eine erneute Synthese eine Netzliste mit geringeren Logiktiefen erzeugt. Alternativ können die Parameter der einzelnen Werkzeuge, mit deren Hilfe diese gesteuert werden, an das gegebene Design angepasst werden.

2.3.3 Optimierungsziele

Beim FPGA-Entwurf gibt es vier wesentliche und gegenläufige Optimierungsziele, die sowohl einzeln und auch in Kombination verfolgt werden:

- Maximierung der möglichen Taktfrequenz (engl.: delay reduction),
- Verbesserung der Verdrahtbarkeit (engl.: routability improvement),
- hohe Ausnutzung der Logikressourcen (engl.: area reduction),
- Reduzierung des Leistungsbedarfs (engl.: power reduction).

Jedem Schaltungsmodell können Metriken oder Kostenfunktionen zugrunde gelegt werden, die bestimmte Faktoren berücksichtigen und deren Optimierung das Erreichen eines der gesetzten Ziele ermöglicht. Veränderungen an den Schaltungsmodellen unter Zuhilfenahme geeigneter Kostenfunktionen haben bei den frühen Entwurfsschritten eine wesentlich stärkere Auswirkung auf die Optimierungsziele als solche bei späteren Schritten. Andererseits muss berücksichtigt werden, dass die Genauigkeit der anhand von Schaltungsmodellen bestimmten Kostenschätzungen mit dem steigenden Detailgrad niedrigerer Abstraktionsebenen zunimmt.

Die maximale Taktfrequenz wird durch die Länge des kritischen Signalpfads bestimmt, wobei diese Länge durch die Verzögerungszeitensumme aller auf dem Signalpfad liegenden LUTs und zur Verdrahtung eingesetzter Verdrahtungsressourcen repräsentiert wird. Bei aktuellen FPGAs beträgt der Anteil der durch die Logik (LUTs) verursachten Verzögerungszeiten nur einen geringen Anteil von 20 bis 30% an der gesamten Verzögerungszeit [67], weshalb eine genaue Bestimmung des kritischen Pfads erst nach der Platzierung und Verdrahtung der LUTs möglich ist.

Der Begriff der Verdrahtbarkeit umfasst drei wesentliche Aussagen: ob ein Design von dem Vedrahtungswerkzeug verdrahtet werden kann, wie kompliziert sich dieser Vorgang gestaltet und wie viele Verdrahtungsressourcen für eine vollständige Verdrahtung benötigt werden. Eine hohe Nutzung der Verdrahtungsressourcen und die damit verbundene geringe Verdrahtbarkeit führen zu längeren Laufzeiten des Verdrahtungswerkzeugs. Bei Überbenutzung der Verdrahtungsressourcen kann ein Design nicht verdrahtet werden. Dies steht in der Regel erst nach sehr langer Laufzeit des Verdrahtungswerkzeugs fest.

Eine hohe Ausnutzung der Logikressourcen kann sinnvoll sein, wenn ein zu einem früheren Zeitpunkt ausgeliefertes Design um zusätzliche Funktionalität erweitert werden soll, das eingesetzte FPGA jedoch nur über wenig freie Logikressourcen verfügt. In der Praxis wird bei Designs mit unkritischen Signallaufzeitvorgaben häufig die Nutzung der Logikressourcen optimiert, um einen kleineren und somit preiswerten FPGA-Baustein einsetzen zu können.

Die Leistungsbedarfreduzierung spielte bei den FPGAs bis vor wenigen Jahren keine Rolle. Das hat sich grundlegend geändert, seitdem FPGAs auch in tragbaren Geräten eingesetzt werden, die nur über begrenzte Energieressourcen verfügen. Der Gesamtleistungsbedarf P_{total} einer im FPGA implementierten Schaltung setzt sich aus folgenden Komponenten zusammen:

$$P_{total} = P_{interconnect} + P_{logic} + P_{clock} + P_{static}$$

In der P_{static}-Komponente ist der statische Leistungsbedarf eines konfigurierten aber inaktiven FPGA zusammengefasst. Dieser weist nur eine Abhängigkeit zu der Versorgungsspannung und Substrattemperatur auf. P_{clock} repräsentiert den Leistungsbedarf der aktiven Taktbäume, die alle Flipflops des FPGA mit Takt versorgen. Solange kein „clock-gating“ erfolgt, ist diese Komponente konstant. Die Algorithmen des FPGA-Entwurfsflusses können also nur den Leistungsbedarf der Verdrahtungs- bzw. der Logikressourcen ($P_{interconnect}$ bzw. P_{logic}) optimieren, indem sie die Nutzung der Verdrahtungs- bzw. der Logikressourcen reduzieren. Der Logikressourcenbedarf ist bereits nach dem Entwurfsschritt der Technologie-Abbildung festgelegt, in den nachfolgenden Entwurfsschritten kann somit nur noch der verdrahtungsressourcenbedingte Leistungsbedarf $P_{interconnect}$ reduziert werden. Das erfolgt, indem die Anzahl der zur Verdrahtung erforderlichen Verdrahtungssegmente und somit auch die gesamte parasitäre Kapazität der Leitungen, die während der Schaltaktivitäten umgeladen werden muss, verringert werden. Unter diesem Gesichtspunkt stellen die Reduzierung von $P_{interconnect}$ und die Steigerung der Verdrahtbarkeit gleichläufige Optimierungsziele dar.

2.3.4 High-Level-Synthese

Die High-Level-Synthese (HLS) kann grob in drei Einzelschritte unterteilt werden: Allokation, Ablaufplanung und Bindung. Bei jedem einzelnen Schritt wird die abstrakte Verhaltensbeschreibung der Schaltung analysiert und das zu erzeugende RTL-Modell iterativ konkretisiert. Bei der Allokation werden zunächst Typ und Anzahl der Ressourcen festgestellt, die für eine Hardwarearchitektur zur Ausführung der beschriebenen Algorithmen erforderlich sind. Da die typischen Funktionseinheiten wie Addierer und Multiplizierer in der Regel mehrfach an unterschiedlichen Stellen des Algorithmus benötigt werden, besteht ein großer Freiheitsgrad bei der Festlegung der Mehrfachnutzung einzelner Hardwareressourcen. In der Regel kann der Anwender durch die Vorgaben an das HLS- Werkzeug den

Ressourcenbedarf, somit die Parallelität und den Datendurchsatz der Zielhardware beeinflussen. Bei der Ablaufplanung werden zeitliche Reihenfolge und Taktzyklenbedarf der einzelnen Operationen unter Berücksichtigung der bei der Allokation festgelegten Hardwareressourcen ermittelt. Anschließend werden bei der Bindung alle Operationen auf die einzelnen Funktionseinheiten verteilt. Die mehrfach benutzten Funktionseinheiten werden zusätzlich mit Steuerschaltungen zur Ablaufsteuerung versehen.

Obwohl die kommerziellen HLS-Werkzeuge bereits seit längerem auf dem Markt verfügbar sind, konnten sie sich in der Praxis nur in bestimmten Aufgabenbereichen durchsetzen. Der Grund dafür liegt in der höheren Qualität der manuell erzeugten RTL-Beschreibungen. Nur bei bestimmten Aufgaben wie dem Zustandsautomatenentwurf und der digitalen Signalverarbeitung liefern zweckbestimmte HLS-Werkzeuge vergleichbare Syntheseergebnisse bei gleichzeitig sehr viel höherer Produktivität.

Die graphische Modellierung der Zustandsautomaten bildet einen festen Bestandteil aktueller FPGA-Softwarepakete und ist eines der Beispiele für den naheliegenden und produktiven Einsatz der HLS. Das Verhalten eines Zustandsautomaten bestimmt der Anwender, indem er den zugehörigen Zustandübergangsgraph mit einem graphischen Werkzeug zeichnet. Der Zustandübergangsgraph wird dann durch ein HLS-Werkzeug in eine synthetisierbare RTL-Beschreibung überführt. Ohne sich um die Zustandskodierung kümmern zu müssen, kann der Anwender sowohl die Geschwindigkeit als auch die Fläche der Lösung direkt beeinflussen, indem er die HLS durch Vorgaben steuert. Auf der RTL-Ebene müsste dafür die HDL-Beschreibung zeitintensiv und fehleranfällig manuell angepasst werden.

In der digitalen Signalverarbeitung (DSP) sind die Programme von MathWorks wie MATLAB® und SIMULINK® [104] zur Modellierung, Analyse und Verifikation der DSP-Algorithmen zu einem industriellen Standard geworden. Die DSP-Algorithmen können mit MATLAB® durch das Zusammenschalten verschiedener Funktionsblöcke und durch eine abstrakte Verhaltensbeschreibungen auf der algorithmischen Ebene modelliert werden. Es existiert eine Reihe von Werkzeugen wie Synplify®DSP von Synplicity [105] und AccelDSP® von Xilinx [106], die zahlreiche DSP-Funktionsblöcke mitbringen und sich nahtlos in die Oberfläche und den Entwurfsfluss von MATLAB® einfügen. Nach einer erfolgreichen Verifikation mit MATLAB® erzeugen diese Werkzeuge eine synthetisierbare HDL-Beschreibung, die eine Abbildung des entsprechenden DSP-Algorithmus auf der RTL-Schaltungsebene darstellt.

Wegen der hochgesteckten und unerreichten Ziele konnte sich bisher ein automatischer C/C++-basierter Hardwareentwurf nicht etablieren. Der Vorteil für den Entwickler wäre ein Software/Hardware-Codesign mit nur einer einzigen Sprache. Praktisch genutzte Lösungen sind HLS-Werkzeuge wie Impulse C™ [107], die aus einem Bruchstück C-Quellcode FPGA-Coprozessorhardware für einen bereits vorhandenen Prozessor (auch auf dem FPGA) erzeugen. Dabei darf der zu synthetisierende Codeabschnitt nur bestimmte C-Sprachkonstrukte enthalten und muss vom Anwender manuell zur Synthese freigegeben werden. Die Qualität der erzeugten Lösung hängt also im hohen Maße von der Erfahrung des Anwenders ab und ist weit von den Automatismen der Logiksynthese entfernt.

2.3.5 RTL- und Logiksynthese

Die Synthese einer Schaltungsbeschreibung auf der RTL-Ebene besteht aus mehreren Schritten. Zunächst wird die HDL-Beschreibung durch einen Präprozessor eingelesen und nach einer syntaktischen und semantischen Überprüfung in Sprachgrundelemente wie Daten- und Kontrollstrukturen zerlegt. Die darauf folgende RTL-Synthese führt eine Abbildung dieser Daten- und Kontrollstrukturen auf die Speicherelemente und Booleschen Funktionen durch. Bei der dann folgenden Logiksynthese finden in der ersten Phase die technologie-

unabhängige Logikminimierung der kombinatorischen und die Anordnungsoptimierung der sequentiellen Schaltungsteile statt. In der zweiten Phase der Logiksynthese werden dann die technologieabhängigen Optimierungen und die Abbildung auf die gegebene Zieltechnologie durchgeführt. Das Ergebnis der Logiksynthese ist eine zieltechnologieoptimierte Netzliste, bestehend aus LUTs und Flipflops (Abbildung 2-18).

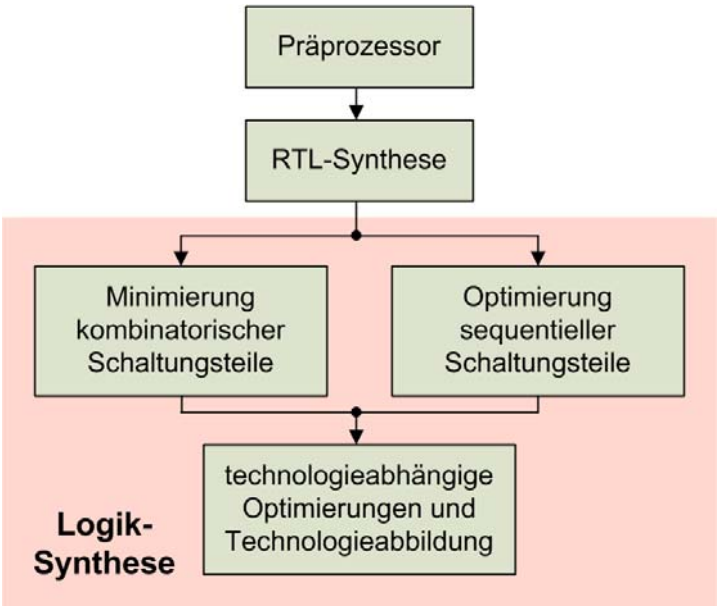


Abbildung 2-18: *Typischer RTL- und Logiksynthesefluss bei FPGAs.*

Bibliothekenbasierte Logiksyntheseverfahren, die die Netzlisten aus den in Bibliotheken enthaltenen Grundgattern erzeugen, sind auf SRAM-basierte FPGAs nicht direkt anwendbar, weil eine LUT mit K Eingängen 2^K unterschiedliche Funktionen implementieren kann. Dadurch würden die für diese Verfahren notwendigen Bibliotheken überexponentiell wachsen. Deshalb fand in den Jahren nach der Einführung der SRAM-basierten FPGAs eine intensive Forschung auf der Suche nach geeigneten Logiksynthesealgorithmen statt. Auf eine umfassende Darstellung dieses Forschungsgebiets wird an dieser Stelle verzichtet, es werden nur grundlegende Techniken im gegenseitigen Zusammenhang vorgestellt. Ein interessierter Leser wird auf die Übersichtsartikel [40] und [39] verwiesen.

In der technologieunabhängigen Phase der Logiksynthese wird die abstrakte Darstellung einer Schaltung Optimierungen unterzogen, deren Zweck die Reduzierung der Logiktiefe und des Logikressourcenbedarfs ist. Die abstrakte Darstellung einer Schaltung kann je nach Optimierungsalgorithmus in unterschiedlichen Darstellungsformen vorliegen [40]. Die Logikminimierung nach Quine und McCluskey (QMCV) erfordert z. B. eine Darstellung der kombinatorischen Schaltungen als KDNF oder KKNF. Die Darstellung einer Schaltung als Boolesches Netzwerk (als gerichteter azyklischer Graph) bietet den Vorteil einer direkten Abbildung der abstrakten Darstellung auf eine Gatter-Netzliste. Die Knoten des Graphen stellen dabei die logischen Verknüpfungen und die Kanten den Datenfluss dar (Abbildung 2-19).

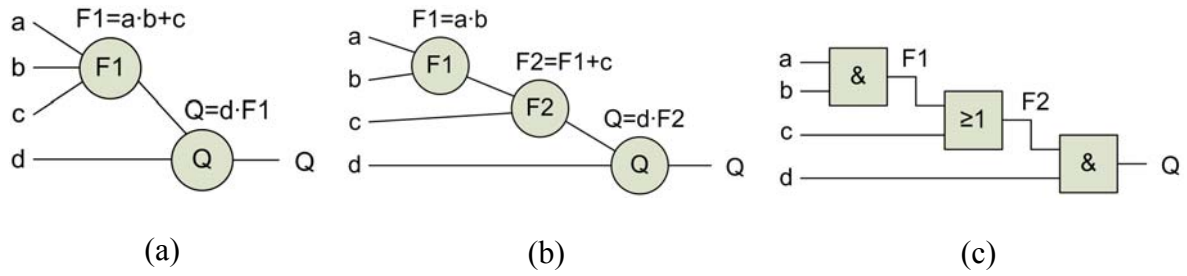


Abbildung 2-19: *Beispiel eines Booleschen Netzwerks (a), generische Form (b) und deren Abbildung auf eine Gatter-Netzliste (c).*

Bei der abstrakten Darstellung einer Schaltung als Boolesches Netzwerk stehen Operationen zweier Klassen zur Verfügung: die zur Umstrukturierung des Netzwerks und die zur Knotenminimierung. Die Umstrukturierung des Netzwerks erfolgt durch das Hinzufügen oder Entfernen von Knoten und Kanten. Bei der Knotenminimierung werden die mit einem Knoten assoziierten Funktionen mit Minimierungsverfahren wie QMCV minimiert. Nach Abschluß der ersten Phase der Logiksynthese ist die Kantenanzahl und die Knotenkomplexität eines Booleschen Netzwerks im Vergleich zur ursprünglichen Form reduziert.

Zweck der Logikumformungen in der zweiten Phase der Logiksynthese ist das Umstrukturieren des Netzwerks in ein anderes, funktional äquivalentes Netzwerk, welches für eine Abbildung auf die LUTs einer festgelegten Zieltechnologie besser geeignet ist [39]. Dabei werden Methoden der Logikduplizierung, des Zusammenfügens von Knoten und der Dekomposition [37] angewendet. Mit Hilfe der Dekomposition können Knotenfunktionen mit großer Anzahl an Eingangsvariablen durch das Einführen weiterer Zwischenknoten auf mehrere Knoten aufgespaltet werden (Abbildung 2-20). In der Regel führt das zu einer Schaltung, die weniger LUTs bei der Technologieabbildung erfordert [40].

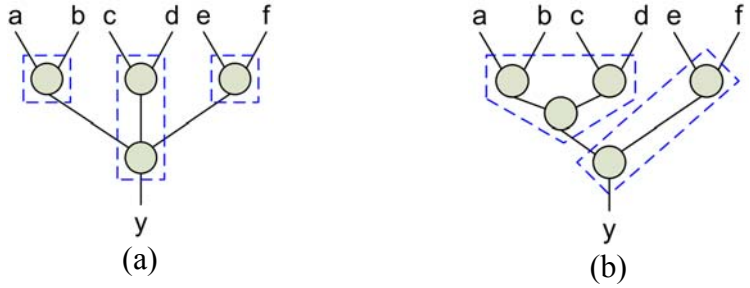


Abbildung 2-20: *Abbildung der Knoten auf LUTs vor (a) und nach der Dekomposition (b).*

Weil für ein gegebenes Netzwerk mehrere, auf LUTs abbildbare und funktional äquivalente Netzwerke bestimmt werden können, muss eine Lösung gefunden werden, die einem oder mehreren im Abschnitt 2.3.3 aufgezählten Optimierungszielen genügt. Soll z. B. die Zahl der bei der Technologieabbildung erforderlichen LUTs der Schaltung minimiert werden, müssen Lösungen bevorzugt ausgewählt werden, die eine überlappungsfreie Abbildung auf die LUTs ermöglichen. Dabei wird in der Regel die Parallelität reduziert, aber die Logiktiefe erhöht [41]. Um die Logiktiefe zu reduzieren, können Logikduplizierungen einzelner Netzwerkabschnitte und Überlappungen bei der Logikabbildung auf die LUTs erforderlich sein. Diese Wechselwirkungen zwischen einzelnen Optimierungszielen und gegenseitige Ausschließungen müssen somit bei der Technologieabbildung berücksichtigt werden.

2.3.6 Packen und Clustering der Logikzellen

Die atomaren Elemente einer nach der Technologieabbildung vorliegenden Netzliste sind LUTs und Flipflops. Sie werden vor dem Clustering in die Logikzellen gepackt. Dabei kann eine Logikzelle maximal eine LUT und ein Flipflop aufnehmen (vgl. Abbildung 2-7). Das Packen der LUTs und Flipflops in die Logikzellen erfolgte bisher mit einer einfachen Mustererkennung [42]. Dabei wurde überprüft, ob einer LUT ein Flipflop nachgeschaltet ist, um anschließend beide in eine Logikzelle zu packen. Nach dem Packen liegt eine nur aus Logikzellen bestehende Netzliste der Schaltung vor.

Beim Clustering-Entwurfsschritt werden die Logikzellen zu Gruppen (Clustern) angeordnet. Die in jedem Cluster enthaltenen Logikzellen müssen einem jedem auf dem FPGA enthaltenen Logikblock zuordenbar sein. Die Voraussetzung dafür ist, dass die Anzahl der Logikzellen in einem Cluster die der Logikzellen in einem Logikblock und die Anzahl der an den Cluster angeschlossenen Netze die der Ein- und Ausgänge eines Logikblocks nicht übersteigt. Da eine Netzliste streng genommen nicht aus Logikzellen bzw. LUTs und Flipflops bestehen kann, weil die entsprechenden Schaltungsstrukturen real nur auf einem FPGA-Baustein existieren, sind mit den LUTs und Flipflops einer Netzliste tatsächlich deren abstrakte Beschreibungen gemeint. Diese Vorstellung wird dahingehend auch auf die Cluster und die Logikblöcke erweitert.

Clustering-Algorithmen können grob in die Klasse der „top-down"- oder in die Klasse der „bottom-up"-Algorithmen eingeteilt werden. „Top-down"-Algorithmen partitionieren die gegebene Netzliste schrittweise in Cluster. „Bottom-up"-Algorithmen bestimmen im ersten Schritt eine Start-Logikzelle, die einem noch leerem Logikblock zugeordnet wird. Im zweiten Schritt werden diesem Logikblock sequentiell weitere Logikzellen hinzugefügt, bis dieser voll ist oder bis die verfügbaren Ein- oder Ausgänge des Logikblocks erschöpft sind. Die Auswahl der hinzuzufügenden Logikzellen erfolgt mit Hilfe einer Kostenfunktion. Die beiden Schritte werden solange wiederholt, wie die Netzliste freie Logikzellen aufweist (Abbildung 2-21).

```
pack_logic_cells(netlist);      // FFs und LUTs in Logikzellen packen
current = get_seed_cell(netlist);
do {
        cluster = construct_cluster(architecture, current);
        do {    // solange Logikblock nicht voll, Logikzellen hinzufügen
                netlist->remove_cell(current);
                cluster->insert_cell(current);

                if(!cluster->is_full) {
                        current = max_attraction_legal_cell(cluster, netlist);
                        if(!current) { current = get_seed_cell(netlist, cluster); }
                }
        } while(!cluster->is_full && current);
        netlist->add_cell(cluster);
        current = get_seed_cell(netlist);
} while (current);      // wiederhole, solange es freie Logikzellen gibt
```

Abbildung 2-21: *Pseudo C-Code für Clustering-Entwurfsschritt.*

Der Vorteil der „bottom-up"-Algorithmen ist, dass eine Logikzellen-Netzliste schnell geclustert werden kann, und dabei die Architekturvorgaben der Logikblöcke einfach eingehalten werden können. Nachteilig gegenüber den „top-down"-Algorithmen ist die lokal beschränkte Betrachtung der Logikzellen-Interkonnektivität beim Bestimmen der in die

Logikblöcke zu packenden Logikzellen. Die Qualität der Clustering-Lösung hängt deshalb im hohen Maße von der Auswahl der Start-Logikzelle ab [38].

2.3.6.1 Verbesserung der Verdrahtbarkeit beim Clustering

Im Clustering-Entwurfsschritt kann die Verdrahtbarkeit einer Netzliste nach folgenden Kriterien verbessert werden [46]:

- Anzahl der benutzten Logikblock-Anschlüsse,
- Anzahl der Signalsenken pro Netz (Fanout),
- Anzahl der Netze.

Für den Anschluss eines Logikblocks an die globale Verdrahtung werden genau so viele Segmente der angrenzenden Verdrahtungskanäle benötigt, wie es benutzte Logikblockanschlüsse gibt. Das heißt, bei einer Verringerung der zur Verdrahtung erforderlichen Logikblockanschlüsse steigt die Verdrahtbarkeit eines Designs, da im gleichen Maße weniger Verdrahtungssegmente bei der Verdrahtung gebraucht werden. Auf der globalen Verdrahtungsebene sind die Logikblockeingänge die Signalsenken (Fanout) der global zu verdrahtenden Netze. Da ein Verdrahtungswerkzeug die meiste Zeit zum Verdrahten der Netze mit einem hohen Fanout benötigt, ist es erstrebenswert, die Anzahl der Logikblockanschlüsse bzw. den Fanout der Netze zu senken.

Anzahl der Logikblockanschlüsse und somit Fanout der Netze wird dadurch reduziert, indem man in einen Logikblock Logikzellen packt, die untereinander möglichst viele Anschlüsse an die gleichen Netze aufweisen. Die Verdrahtung eines ganzen Netzes erfolgt dann sogar komplett innerhalb eines Logikblocks, wenn sich alle Signalsenken und die Signalquelle innerhalb des Logikblocks befinden. Auf diese Weise wird die Anzahl der global zu verdrahtenden Netze reduziert. Die Gesamtanzahl der global zu verdrahtenden Netze hat auch einen direkten Bezug zur Verdrahtbarkeit: Erhöht sich die Anzahl der Netze, so wird es aufgrund fest vorgegebener FPGA-Verdrahtungsressourcen für das Verdrahtungswerkzeug schwieriger, das Design zu verdrahten.

Die Verdrahtung eines Netzes komplett innerhalb eines Logikblocks ist aus mehreren Gründen die effizienteste Art der Verdrahtung eines hierarchisch aufgebauten FPGA. Aufgrund eines begrenzten Fanouts und der kurzen zu überbrückenden Strecken weisen die Verdrahtungssegmente innerhalb eines Logikblocks niedrige Verzögerungszeiten und geringen Leistungsverbrauch pro Schaltvorgang auf [38]. Da die Verdrahtung in dem betrachteten Sonderfall komplett innerhalb eines Logikblocks erfolgt, werden die Verdrahtungsressourcen in den Verdrahtungskanälen außerhalb des Logikblocks frei. Sind Verdrahtungsressourcen in größerer Anzahl vorhanden als zur Designverdrahtung notwendig, können vom Verdrahtungswerkzeug Netze mit einem hohen Fanout dupliziert werden, um den Fanout dieser Netze und somit auch die Verzögerungszeiten zu reduzieren.

2.3.6.2 „Bottom-up“-Clustering-Verfahren

Einer der ersten und bekanntesten Clustering-Algorithmen für hierarchische FPGAs ist VPack [43]. VPack verfolgt zwei Optimierungsziele: die Logikblöcke möglichst voll mit Logikzellen zu packen und die Nutzung der Logikblockanschlüsse zu minimieren. Durch diese Vorgehensweise soll die Interkonnektivität der Logikblöcke untereinander reduziert und somit die Verdrahtbarkeit gesteigert werden. Als Start-Logikzelle wird die Logikzelle mit den meisten Eingängen ausgewählt. Zum Packen in einen Logikblock werden Logikzellen bestimmt, die mit diesem Logikblock die meisten Anschlüsse an identische Netze teilen. Der kurze Zeit später vorgestellte Clustering-Algorithmus T-VPack erweitert VPack um die Optimierung der Verzögerungszeiten, wodurch die Tiefe der kritischen Signalpfade verringert

wird [42]. Bei T-VPack werden zunächst mit einer Timing-Analyse die kritischen Pfade einer Netzliste bestimmt. Logikzellen, die auf einem kritischen Pfads liegen, werden dann möglichst in denselben Logikblock gepackt. Da die Verzögerungszeiten bei der Verdrahtung innerhalb eines Logikblocks geringer sind als bei der Verdrahtung zwischen zwei Logikblöcken, wird die Gesamtverzögerungszeit eines kritischen Pfades dadurch verringert. Interessanterweise produziert T-VPack nicht nur um ca. 30% schnellere, sondern auch bis zu 20% kompaktere Clustering-Lösungen als VPack. Die Ursache dafür ist, dass bei dieser Vorgehensweise sehr viele kompakte Netze komplett innerhalb der Logikblöcke verdrahtet werden können und so die Nutzung von Logikblockanschlüsse gesenkt wird. Dadurch können auch mehr Logikzellen in die Logikblöcke gepackt werden [42].

RPack ist ein Clustering-Verfahren, welches nur die Verdrahtbarkeit einer Netzliste optimiert [44]. Die Autoren des Verfahrens identifizierten die die Verdrahtbarkeit beeinflussenden Faktoren und berücksichtigten diese in einer neuen Kostenfunktion. Diese Kostenfunktion bevorzugt bei der Auswahl der zu packenden Logikzellen solche, die zur Absorption der meisten Netze mit nur einer Signalsenke (Fanout = 1) innerhalb eines Logikblocks führen. Die Auswahl der Start-Logikzelle erfolgt auf die gleiche Weise wie beim VPack. Zum Verdrahten der von RPack erzeugten Clustering-Lösungen werden bei gleicher Logikausnutzung im Durchschnitt etwa 20% weniger Verdrahtungssegmente benötigt als bei VPack. Die Ergebnisse sind, bis auf die Verzögerungszeiten, mit denen von T-VPack vergleichbar. Später wurde das Clustering-Verfahren T-RPack vorgestellt, welches zusätzlich zur Verdrahtbarkeit auch die Verzögerungszeiten optimiert. Die dabei eingesetzte Vorgehensweise entspricht der von T-VPack. Insgesamt sind die Clustering-Lösungen von T-VPack und T-RPack gleichwertig.

Nach RPack wurde das Clustering-Verfahren iRAC vorgestellt, welches auch die Verdrahtbarkeit einer Netzliste optimiert [38]. iRAC versucht, möglichst viele Netze mit geringem Fanout komplett innerhalb der Logikblöcke zu verdrahten und somit die globalen Verdrahtungsressourcen zu entlasten. Anders als bei RPack gibt es dabei keine Einschränkung auf Netze mit nur einer Signalsenke. Bereits bei der Auswahl der Start-Logikzelle werden Logikzellen bevorzugt, die an möglichst viele Netze mit einem geringen Fanout angeschlossen sind. Im Vergleich zum RPack und T-VPack wurde die Anzahl der zur Verdrahtung benötigten Verdrahtungssegmente im Schnitt um etwa 30% gesenkt. Des Weiteren wird unter Berücksichtigung der Rents-Regel eine gleichmäßige Interkonnektivität aller Logikblöcke angestrebt, indem diese nicht bis an ihre Kapazitätsgrenzen gefüllt werden. Durch diese Vorgehensweise wird die lokale Überbenutzung globaler Verdrahtungs-ressourcen zum Preis eines höheren Flächenbedarfs effektiv verhindert.

Zuletzt wurde das Verfahren SMAC vorgestellt, welches die Technologie-Abbildung und das Clustering der Logikzellen gleichzeitig in einem einzigen Entwurfsschritt durchführt [45]. Im Vergleich zu den Verfahren, die beide Entwurfsschritte sequentiell vollziehen, konnte eine Reduzierung der Signalverzögerungszeiten um bis zu 12% erreicht werden. Der Logikressourcenbedarf der von SMAC erzeugten Netzlisten ist jedoch um 22% gestiegen.

2.3.7 Platzierung der Logikblöcke

Beim Entwurfsschritt der Platzierung wird jedem in der Netzliste enthaltenen Logikblock eine exklusive Position auf dem FPGA in Form einer Logikblockkachel zugewiesen. Dabei werden eines oder mehrere der im Abschnitt 2.3.3 aufgezählten Optimierungsziele verfolgt. Die Klassifizierung der Platzieralgorithmen erfolgt in auf Simulated Annealing, Partitionierung, genetischen Algorithmen, Kräftemodellen oder Methoden der numerischen Optimierung basierende Verfahren [47]. Eine alternative Klassifizierung der Platzier-Algorithmen befindet sich in [48].

2.3.7.1 Platzierung mit Simulated Annealing

Simulated Annealing (SA) ist eine Heuristik zur Lösung NP-harter kombinatorischer Optimierungsprobleme, zu denen auch die Platzierung gehört. Die SA-basierte Platzierung ist eines der am häufigsten eingesetzten FPGA-Platzierverfahren. Die Gründe dafür sind die einfache Anpassungsmöglichkeit der SA-Kostenfunktion an unterschiedliche Optimierungs-kriterien und gute Platzierungen, die mittels SA innerhalb akzeptabler Zeit erzeugt werden können. SA-basierte Platzierverfahren simulieren den physikalischen Prozess der Metallschmelzeabkühlung [49]. Wird der Schmelze langsam Energie entzogen, nehmen die Metallatome energiegünstige Positionen im Metallgitter ein. Beim Platzieren mit Simulated Annealing stehen die Logikblöcke bzw. Logikzellen in Analogie zu den Metallatomen.

Die initiale Platzierung erfolgt durch zufällige Verteilung der Logikblöcke auf die verfügbaren Logikblockpositionen des FPGA. Während der Platzieriterationen werden dann zufällige Logikblockbewegungen vorgeschlagen. Die Bewertung dieser Bewegungen erfolgt mit einer Kostenfunktion (Abbildung 2-22).

```
S = random_place(netlist);     // Logikblöcke initialplatzieren
T = initial_Temp();            // Starttemperatur bestimmen
R_lim = initial_Rlim();        // Bereichsbeschränkung
do {    // äußere Schleife
        do {    // innere Schleife
                S_new = random_move(S, R_lim);   // zufällige Platzierungspermutation
                ΔC = Cost(S_new) – Cost(S);      // Kostendifferenz bestimmen
                R = random(0,1);                 // Zufallszahl [0,0…1,0]
                if(R<e^(-ΔC/T)) { S = S_new; }
        } while(!inner_Loop_criterion());
        T = update_Temp();             // neue Temperatur bestimmen
        R_lim = update_Rlim();         // neue Bereichsbeschränkung berechnen
} while (!exit_criterion());    // wiederhole, solange Abbruchkriterium nicht erfüllt
```

Abbildung 2-22: *Pseudo-C-Code für SA-basierte Platzierung.*

Die vorgeschlagenen Bewegungen mit negativer Kostendifferenz werden immer akzeptiert (P_{accept} = 1), die Akzeptanzwahrscheinlichkeit $P_{accept}(\Delta C)$ der Bewegungen mit positiver Kostendifferenz wird gemäß der Funktion $e^{-\Delta C/T}$ berechnet (Abbildung 2-22). Diese Wahrscheinlichkeitsfunktion hängt sowohl von der Temperatur T als auch von der Kostendifferenz ΔC ab (Abbildung 2-23). Am Anfang des Platziervorgangs ist die Temperatur T hoch, fast alle vorgeschlagenen Logikblock-Bewegungen werden durchgeführt. Im Laufe der Platzierung wird die Temperatur schrittweise abgesenkt, so dass Logikblock-Bewegungen, die die Platzierung verschlechtern würden, immer seltener akzeptiert werden.

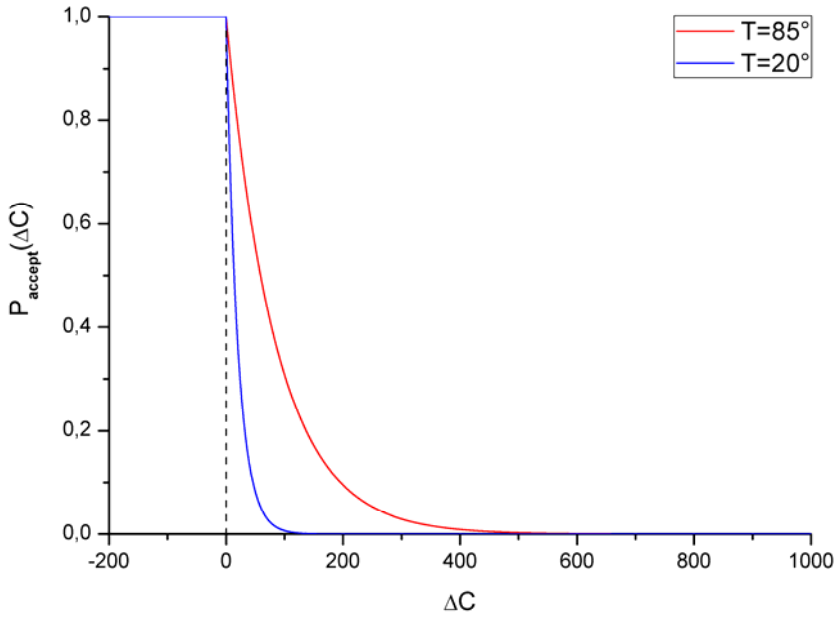


Abbildung 2-23: *Akzeptanzwahrscheinlichkeit $P_{accept}(\Delta C)$ einer vorgeschlagenen Bewegung in Abhängigkeit von den damit verursachten Kosten und der Temperatur.*

Die Möglichkeit, auch ungünstige Logikblockbewegungen zu akzeptieren, erlaubt es Simulated Annealing, die lokalen Minima der Platzierlösungen verlassen zu können (engl.: uphill-climbing). Mit dem Fortschreiten des Platzierungsprozesses steigt die Qualität der Platzierung und somit auch die Wahrscheinlichkeit, dass Bewegungen vorgeschlagen werden, die bei ihrer Akzeptanz positive Kosten verursachen würden (Abbildung 2-24).

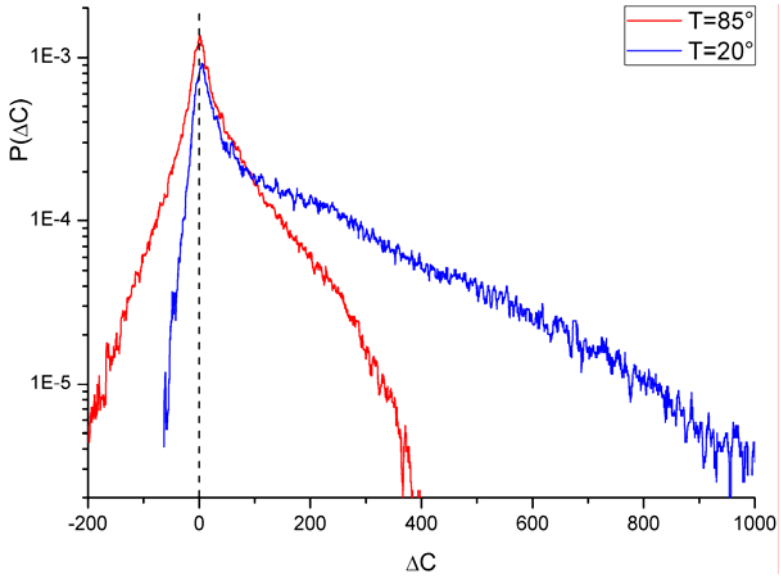


Abbildung 2-24: *Wahrscheinlichkeitsdichteverteilungen von ΔC für unterschiedliche Platzieretappen einer Netzliste („oc_mc8051"-Design).*

Wird bei einer Temperatur für eine größere Anzahl von Bewegungen keine Veränderung der Platzierkosten erreicht, kann die Temperatur abgesenkt werden, weil sich der Platzierungsprozess in einem thermodynamischen Gleichgewicht befindet. Formal ausgedrückt ist das Integral der Wahrscheinlichkeitsverteilung aller negativen Kosten, multipliziert mit der

Akzeptanzwahrscheinlichkeit, vom Betrag her gleich dem Integral der Wahrscheinlichkeitsverteilung aller positiven Kosten, multipliziert mit der Akzeptanzwahrscheinlichkeit:

$$E_- = E_+$$

$$E_- = \left| \int_{-\infty}^{0} \Delta C \cdot P(\Delta C) d\Delta C \right|, \quad E_+ = \int_{0}^{\infty} \Delta C \cdot P(\Delta C) \cdot e^{-\Delta C / T} d\Delta C$$

Das Abkühlungsschema (engl.: annealing schedule) legt das Abbruchkriterium der äußeren Schleife, die Anzahl der Logikblock-Bewegungen pro Temperaturschritt, die Art und Weise wie die Logikblockbewegungen vorgeschlagen werden und wie die Temperatur abgesenkt wird, fest. Bei einer geeigneten Wahl des SA-Abkühlungsschemas können die Kosten der Platzierung gegen ein globales Minima konvergieren, eine Garantie dafür kann es bei einem heuristischen Verfahren wie Simulated Annealing aber nicht geben [50]. Die Abkühlungsschemata mit fest eingestellten Steuerparametern sind nicht für jede Art von Optimierungsproblemen gleich gut geeignet. Deshalb wurde in der Vergangenheit viel Aufwand in die Entwicklung universeller und adaptiver Abkühlungsschemata investiert.

Die Höhe der Starttemperatur kann bei Simulated Annealing sowohl die Qualität der Lösung als auch die Laufzeit des Verfahrens beeinflussen. Bei zu niedriger Starttemperatur kann sich das Platzierverfahren in einem lokalen Minimum verfangen. Bei zu hoher Starttemperatur wird Rechenzeit verschwendet. In [51] wird vorgeschlagen, die Starttemperatur auf den zwanzigfachen Wert der Standardabweichung σ der Kostenänderung festzulegen, die bei großer Zahl zufälliger Bewegungen gemessen wird.

Unter der Annahme, dass eine gegebene Platzierung bei einer bestimmten Temperatur eines SA-Prozesses sich im thermodynamischen Gleichgewicht befindet, kann diese Temperatur bestimmt werden. Zur Temperaturschätzung wurde in [52] folgende Vorgehensweise vorgeschlagen:

1. Bestimmen der statischen Wahrscheinlichkeitsdichteverteilung von ΔC, indem die Kosten einer großen Anzahl von Zufallsbewegungen ausgewertet und in einer Datenstruktur abgelegt werden (die Platzierung selbst wird dabei nicht verändert),
2. Temperatur T auf einen geeigneten Anfangswert festlegen,
3. zunächst $P_{accept}(\Delta C)$ und anschließend E_- und E_+ bestimmen,
4. wenn $E_- < E_+$ dann T reduzieren; wenn $E_- > E_+$ dann T erhöhen und mit dem Schritt 3 fortfahren; bei $E_- = E_+$ ist die Suche beendet, die Temperatur T entspricht näherungsweise der gesuchten. Anmerkung: Die Verringerung oder Erhöhung der Temperatur T erfolgt in diesem Schritt gemäß den Vorschriften des Binäre-Suche-Algorithmus.

Bei einer Initialplatzierung würde die auf diese Weise bestimmte Temperatur der Temperatur entsprechen, mit welcher der SA-Prozess zu starten wäre.

Eine Temperaturiteration (innere Schleife in Abbildung 2-22) wird dann abgebrochen und die Temperatur abgesenkt, wenn sich die Platzierung für die gegebene Temperatur im thermodynamischen Gleichgewicht befindet. Dieser Zustand kann an den konstant bleibenden Kosten der Platzierung über eine große Anzahl von Permutationen erkannt werden [55]. Die zum Erreichen eines Zustands nahe dem thermodynamischen Gleichgewicht auszuwertende Zahl der Permutationen pro Temperaturschritt kann auch geschätzt werden. Diese ist näherungsweise $10 \cdot (N_{Blocks})^{1,33}$, wenn die Akzeptanzrate der vorgeschlagenen Bewegungen bei 0,44 liegt [56]. Wird die Anzahl der Blöcke N_{Blocks} verzehnfacht, so müssen nach dieser

Vorschrift zwanzig Mal so viele Permutationen ausgewertet werden. Um die Anzahl der zu platzierenden Blöcke effektiv zu verringern, kann entweder die Netzliste partitioniert („top-down"-Ansatz) oder die Blöcke können geclustert werden („bottom-up"-Ansatz). Der dazu erforderliche Aufwand darf jedoch nicht höher sein als die möglichen Aufwandsersparnisse beim Simulated Annealing. Ein schnelles, mehrstufiges Simulated Annealing-Verfahren für FPGAs mit einem „bottom-up"-Ansatz wurde in [54] vorgestellt. Die zu platzierenden Blöcke werden mit einem linearen Aufwand so geclustert, dass die Anzahl der Netze zwischen den Clustern minimiert wird. Simulated Annealing wird bei solcher „teile und herrsche"-Vorgehensweise mehrfach ausgeführt; vor jedem Anlauf wird die höchste Cluster-Hierarchiestufe aufgebrochen und die Anzahl der Blöcke pro Cluster verringert. Vor jedem SA-Anlauf wird die Temperatur der vorliegenden Platzierung bestimmt, um die Platzierungen nicht durch eine zu hohe Temperaturwahl zu zerstören.

In [55] wurde erkannt, dass bei niedrigeren Temperaturen fast nur lokale Bewegungen stattfinden. Vorgeschlagene Bewegungen über längere Entfernungen werden kaum akzeptiert, deren Auswertung erfordert jedoch viel Rechenzeit. Deshalb wurde eine Bereichs-Beschränkung (R_{lim} in Abbildung 2-22) für die Generierung der vorzuschlagenden Bewegungen eingeführt. Diese bestimmt, in welchem maximalen Abstand sich die beiden zu vertauschenden (Logik-)Blöcke befinden dürfen. Am Anfang wird R_{lim} auf die gesamte Chipfläche ausgedehnt. Im Laufe der Platzierung wird R_{lim} so angepasst, dass das Verhältnis der akzeptierten zu den vorgeschlagenen Bewegungen bei etwa 0,44 liegt. Durch die Einführung der Bereichsbeschränkung konnte die Effizienz der SA-Abkühlungsschemata gesteigert werden [55].

2.3.7.2 Optimierung und Modellierung der Verdrahtbarkeit

Eine Verbesserung der Verdrahtbarkeit wird beim Platzieren primär durch die Verringerung der Leitungslängen erreicht, die zur Verdrahtung des gesamten Designs erforderlich sind (engl.: wirelength driven placement). Die zur Verdrahtung eines Netzes erforderliche Leitungslänge kann mit Hilfe einer Bounding-Box abgeschätzt werden. Diese Bounding-Box-Abschätzung entspricht dem halben Umfang eines Rechtecks, das alle Anschlüsse eines Netzes einschließt (vgl. Abbildung 6-2) und ist für Anschlusszahlen kleiner als vier hinreichend genau. Die Bounding-Box-Abschätzungen für Netze mit höheren Anschlusszahlen fallen zu niedrig aus und müssen deshalb mit Korrekturfaktoren versehen werden, die aus fertig verdrahteten Designs statistisch ermittelt werden können [58].

Bei einer auf minimale Leitungslängen hin optimierten Platzierung kann es zu lokal beschränkter Überbenutzung (Kongestion) der Verdrahtungsressourcen kommen. Im Gegensatz zu der Überbenutzung der Logikressourcen, die sehr schnell festgestellt werden kann, hat die lokale Überbenutzung der Verdrahtungsressourcen schwerwiegendere Konsequenzen: Diese wird erst nach einer langen Laufzeit des Verdrahtungswerkzeugs erkannt. Deshalb ist es erforderlich, die Nutzung der Verdrahtungsressourcen bereits bei der Platzierung abzuschätzen und diese über den gesamten Chip homogen zu verteilen. Mit RISA [58] wurde eine Bounding-Box-basierte Modellierung des Verdrahtungsbedarfs aller Netze für ASICs vorgeschlagen. Durch die Überlappung aller Bounding-Box-Abschätzungen einer Platzierung entsteht eine zweidimensionale Übersicht, die die Nutzung der Verdrahtungsressourcen über die gesamte Chipfläche modelliert (engl.: congestion map). Bei den einzelnen Bounding-Box-Abschätzungen wird eine homogene Verteilung des Verdrahtungsbedarfs angenommen. In [59] wurde diese Modellierungstechnik bei den FPGAs erfolgreich angewendet, um vor der Verdrahtung zu erkennen, ob eine Überbenutzung der Verdrahtungsressourcen vorliegt. Aufgrund der dabei gewonnen Erkenntnisse konnte die Verdrahtungsstrategie des Verdrahtungswerkzeugs gesteuert werden.

Ein Zusammenhang zwischen der Nutzung von Logikressourcen und den benötigten Ein- und Ausgängen wurde 1960 vom IBM-Mitarbeiter E. F. Rent entdeckt und ist seitdem als Rents-Regel bekannt [60]:

$$N_{io} = K \cdot B^{\rho}$$

wobei N_{io} die Anzahl der Ein- und Ausgänge, B die Anzahl der (Logik-)Blöcke, K die Zahl der Anschlüsse pro Block und ρ Rents-Parameter ist. Rent selbst nutzte den von ihm erkannten Zusammenhang, um die Verdrahtbarkeit abzuschätzen. Rents-Parameter kann für ein Design aus der Interkonnektivität der Blöcke und der Gesamtnetzlänge bestimmt werden. Bei einer Abschätzung von ρ kann aus der Interkonnektivität der Blöcke auf die Gesamtnetzlänge geschlossen und die Verdrahtbarkeit vorhergesagt werden [62]. Diese Vorgehensweise hat sich im Bereich des ASIC-Entwurfs erfolgreich bewährt. In [63] wurde dieser Ansatz auf eine einfache FPGA-Architektur angewendet, um die Verdrahtbarkeit der Designs vorherzusagen. Für die Modellierung der Verdrahtbarkeit über die gesamte Chipfläche zur Bestimmung lokaler Überbenutzung der Verdrahtungsressourcen muss das Design partitioniert werden, um dann die lokalen Rents-Parameter der einzelnen Partitionen zu bestimmen [61]. Die Modellierung der Verdrahtbarkeit mit der Rents-Regel ist eine Vorgehensweise, die nur indirekte Faktoren auswertet [66]. Es wird bemängelt, dass es bisher keine Arbeiten gibt, die die Effektivität der Verdrahtbarkeitsmodellierung mit Rents-Regel auf FPGA-Architekturen mit heterogen segmentierten Verdrahtungsressourcen untersuchen [64].

Die empirisch hergeleiteten Methoden zur Bestimmung der Verdrahtbarkeit, die auf der Bounding-Box oder Rents-Regel beruhen, erfordern keine Kenntnisse über die zugrunde liegende Verdrahtungsarchitektur des Ziel-FPGA. Hingegen ist fGREP [65] ein neues Verfahren zur Modellierung der Verdrahtbarkeit, welches die Verdrahtungsarchitektur des Ziel-FPGA in Form eines Verdrahtungsgraphen berücksichtigt. Bei einer Änderung der Zielarchitektur, muss nur der Verdrahtungsgraph, nicht aber der Algorithmus selbst geändert werden. Weiterhin werden bei diesem Verfahren die grundlegenden Strategien der Verdrahtungsalgorithmen berücksichtigt. Dadurch ist fGREP in der Lage, die Nutzung der Verdrahtungsressourcen über den gesamten Chip mit hoher Genauigkeit vorherzusagen.

Wurde ein Modell zur voraussichtlichen Nutzung der Verdrahtungsressourcen erstellt, können die Erkenntnisse aus diesem Modell auf zwei möglichen Wegen bei der Platzierung berücksichtigt werden: entweder durch die Netzgewichtung oder durch die Verminderung der Logikressourcen in den kritischen Bereichen [70]. Bei der Netzgewichtung werden die Bounding-Boxen der Netzlängenabschätzungen entsprechend der Überbenutzung gewichtet. Während der Platzierung wird dann die Netzlänge höher gewichteter Netze stärker reduziert und somit die Verdrahtbarkeit in den Regionen mit überbenutzten Verdrahtungsressourcen verbessert.

Eine Verminderung der Logikressourcen kann bei einer unveränderlichen Clustering-Lösung nur durch ein gezieltes Einbauen leerer Logikblöcke in die kritischen Bereiche erfolgen. Bisher ist kein FPGA-Platzierverfahren bekannt, welches diese Strategie verfolgt. Damit die Berücksichtigung des erzeugten Modells bei der Platzierung tatsächlich zu der gewünschten lokalen Entspannung der Verdrahtungsressourcennutzung führt, ist nicht die absolute Genauigkeit des Modells ausschlaggebend. Vielmehr ermöglicht die relative Genauigkeit eines Modells die genaue Positionsbestimmung der FPGA-Bereiche mit hoher Nutzung der Verdrahtungsressourcen [66].

2.3.7.3 SA-basierte Platzierverfahren

Das bekannteste auf Simulated Annealing basierende Platzierverfahren für FPGAs ist Bestandteil von VPR [68]. Dieses Platzier- und Verdrahtungswerkzeug wurde entworfen, um die Vor- und Nachteile verschiedener FPGA-Architekturen evaluieren zu können. Dazu wurde das Abkühlungsschema von VPR universell ausgelegt, um so möglichst viele FPGA-Architekturen gleich gut zu unterstützten. Die Starttemperatur wird auf den zwanzigfachen Wert der Standard-Abweichung σ der Platzierkosten festgelegt, wie sie bei großer Zahl zufälliger Bewegungen gemessen wurden. Diese Vorgehensweise zur Temperaturbestimmung wurde [51] entlehnt. Wie in [56] entspricht die Anzahl vorgeschlagener (Logikblock-) Bewegungen pro Temperaturschritt $10 \cdot (N_{Blocks})^{1,33}$, wobei N_{Blocks} die Anzahl der Logikblöcke ist. Die Temperatur wird in Abhängigkeit von der Akzeptanzrate der vorgeschlagenen Bewegungen reduziert. Bei sehr hoher oder niedriger Akzeptanzrate wird die Temperatur schnell abgesenkt, bei mittlerer Akzeptanzrate langsamer. Dem liegt die Beobachtung zugrunde, dass die Akzeptanzrate für ein effizientes Abkühlungsschema möglichst lange bei 0,44 liegen sollte [55]. Die Abkühlung wird abgebrochen, wenn die Temperatur den Wert $0{,}005 \cdot C/N_{Nets}$ unterschritten hat, wobei C die Kosten der Platzierung und N_{Nets} die Gesamtzahl der Netze darstellt. Die Ermittlung der Platzierungskosten ist RISA [58] entlehnt; hier werden zusätzlich die Verdrahtungskanalbreiten berücksichtigt. Zur schnelleren Ermittlung der Abmessungen von Bounding-Boxen wurde in VPR ein inkrementelles Verfahren implementiert, das den dazu erforderlichen Aufwand erheblich reduziert [14]. Die Qualität der erzeugten Platzier- und Verdrahtungslösungen macht VPR zum Referenzverfahren der akademischen Platzier- und Verdrahtungswerkzeuge.

Das Clustering der Logikzellen ist ein kritischer Entwurfsschritt, weil Clustering die Verdrahtbarkeit einer Platzierung stark beeinflussen kann. Während des Clusterings ist der Lösungsraum aller möglichen Logikblockzusammensetzungen groß, jedoch ist weder eine Abschätzung noch eine Information über die späteren Positionen der Logikzellen auf dem FPGA verfügbar. Bei der Platzierung hingegen sind die Logikzellenpositionen bekannt, aber die Zusammensetzung der Logikblöcke ist fest vorgegeben, was den Lösungsraum möglicher Platzierungen unnatürlich einschränkt. Beim Clustering gemachte Fehler können beim Platzieren nicht mehr behoben werden. Diese Erkenntnis führte zur Entwicklung von SCPlace [71]. Bei diesem Werkzeug werden das Clustering und die Platzierung der Logikzellen gleichzeitig in einem gemeinsamen Entwurfsschritt durchgeführt. Das Abkühlungsschema, die Kostenfunktionen zur Minimierung der Leitungslängen und der Verzögerungszeiten entsprechen denen von VPR [68]. SCPlace kann sowohl die Logikblöcke als Ganzes bewegen als auch einzelne Logikzellen zwischen den Logikblöcken vertauschen. Dadurch kann die Zusammensetzung der Logikblöcke während des Platziervorgangs geändert werden. Logikzellenvertauschungen werden etwa zehn Mal häufiger als Logikblockbewegungen durchgeführt. Im Vergleich zu VPR erreicht SCPlace im Durchschnitt eine Reduzierung der Verdrahtungslänge um 22% und der maximalen Verzögerungszeiten um 25%. Kürzlich wurde von den gleichen Autoren mit SPCD der Nachfolger von SCPlace vorgestellt [72]. Dieses Verfahren erreicht durch die Logikduplizierung während der Platzierung eine weitere Reduzierung der maximalen Verzögerungszeiten. Das Wichtigste an SPCD ist jedoch, dass der Entwurfsfluss auch die Stratix-Familie von Altera unterstützt. Damit ist SPCD nach der Autorenaussage das erste nicht-kommerzielle Werkzeug, welches, mit Ausnahme der Verdrahtung, einen kompletten Entwurfsfluss für eine kommerzielle FPGA-Familie bietet. Die mit SPCD erzeugten Schaltungsimplementierungen wurden in der gleichen Arbeit mit deren von kommerziellen Altera-Quartus-Software für die Stratix-Architektur verglichen – die mit SPCD erzielten Verzögerungszeiten waren geringfügig besser (etwa um 4%).

2.3.7.4 Flächenplanung

Die Flächenplanung kann allgemein als Platzierung heterogener und verschieden großer Funktionsblöcke mit Formalternativen betrachtet werden. Sie ist beim FPGA-Entwurf ein optionaler Entwurfsschritt, der ggf. erst bei komplexeren Designs erforderlich wird. So können die Platzier- und Verdrahtungswerkzeuge bei komplexen Designs Probleme verursachen, die beim ASIC-Entwurf bereits vor Jahren bekannt waren: lange Werkzeuglaufzeiten, nicht reproduzierbare und unvorhersagbare Ergebnisse. Dadurch wird der Anwendungsbereich der FPGAs gegenüber den ASIC-basierten Entwürfen eingeschränkt. Eine direkte Übernahme der Methoden und Lösungsansätze der Flächenplanung zur Beherrschung der steigenden Komplexitäten aus dem ASIC-Entwurfsstil hat bisher nicht zu den gewünschten Ergebnissen geführt. Die Ursachen dafür sind in den unterschiedlichen Problemstellungen der FPGA- und der ASIC-Flächenplanung zu suchen [73].

Beim ASIC-Entwurf müssen Funktionsblöcke mit einer oder mehreren Formalternativen so positioniert werden, dass die Fläche des sie umschließenden Rechtecks und die Gesamtleitungslänge minimiert werden. Die Positionierung der zu platzierenden Blöcke innerhalb der Chipfläche kann beliebig erfolgen, wenn die überlappungsfreie Anordnung der Blöcke und deren Verdrahtbarkeit sichergestellt sind. Für die zu platzierenden Blöcke liegen in der Regel mehrere Formalternativen vor. Bei der FPGA-Flächenplanung dagegen müssen Blöcke mit beliebigen Formalternativen innerhalb einer FPGA-Fläche mit vorgegebener Kontur so positioniert werden, dass die Verdrahtbarkeit gewährleistet wird. Die Nichtüberlappung der Blöcke (wie bei der ASIC-Flächenplanung) ist keine notwendige Bedingung, solange sichergestellt ist, dass jedem Block ausreichend FPGA-Ressourcen zugewiesen wurden. Es ist sogar erwünscht, dass sich die Blöcke überlappen und die Zellen teilen [73]. Die Logikressourcen in Form der RAM-, DSP- und Logikblöcke sind bei aktuellen FPGAs in Spalten angeordnet (vgl. Abbildung 3-2). Deshalb sind die möglichen Positionen der Funktionsblöcke innerhalb der FPGA-Fläche diskret und werden von den innerhalb der Blöcke verwendeten Ressourcen bestimmt. Bei der ASIC-Flächenplanung wird ein Seitenverhältnis der Blöcke von Eins angestrebt. Dadurch sollen die Gesamtverdrahtungslänge innerhalb eines Blocks und die Verzögerungszeit minimiert werden [74]. Für den FPGA-Entwurf ist es eine überbestimmte (engl.: overconstrained) Randbedingung, da hier die Art der verwendeten Verdrahtungsressourcen für die Verzögerungszeiten ausschlaggebend ist. Die Aussage, ob eine bestimmte Form des Blocks sich negativ auf die maximale Verzögerungszeit auswirkt, kann erst nach Platzierung und Verdrahtung der Logik eines Blocks getroffen werden.

Bisherige FPGA-Flächenplanungsverfahren können nicht auf real existierende FPGA-Architekturen angewendet werden, da bei ihnen sehr starke Vereinfachungen bezüglich der FPGA-Zielarchitekturen vorgenommen wurden [76]. Die FPGA-Hersteller selbst bieten nur Tools, die eine manuelle Flächenplanung ermöglichen, wie PlanAhead von Xilinx [103]. In [76] wird ein konstruktives Flächenplanungs-Verfahren für die Xilinx-Spartan-3-FPGAs vorgestellt, welches die Funktionsblöcke mit geringer Flexibilität zuerst auf dem FPGA platziert. Die Flexibilität der Blöcke wird aus dem Ressourcenbedarf und aus der noch freien FPGA-Fläche bestimmt. Je größer ein Block und je kleiner die noch zur Verfügung stehende Fläche ist, desto geringer ist die Flexibilität. Weiterhin fließt in die Berechnung der Flexibilität die Interkonnektivität der bereits platzierten und des betrachteten Blocks ein. Je höher diese ist, desto geringer ist auch hier die Flexibilität.

2.3.8 Verdrahtung

Bei der Platzierung wird allen LUTs und Flipflops eine exakte Position auf dem FPGA-Chip zugewiesen. Bei der Verdrahtung werden dann die Verdrahtungsressourcen bestimmt, die zur Realisierung aller gewünschten Verbindungen zwischen den platzierten LUTs und Flipflops einer Netzliste zu verwenden sind. Dazu wird ein Verdrahtungsalgorithmus auf einem gerichteten Verdrahtungsgraphen des Ziel-FPGA ausgeführt. Der Verdrahtungsgraph repräsentiert alle in einem FPGA verfügbaren Verdrahtungsressourcen mit deren Interkonnektivität untereinander. Die Knoten eines Verdrahtungsgraphen stellen die Verdrahtungssegmente und die Anschlüsse der Logikressourcen dar. Die Kanten repräsentieren die möglichen programmierbaren Verbindungen. Die Aufgabe des Verdrahtungswerkzeugs ist es, für jede herzustellende Verbindung noch unbelegte Pfade im Verdrahtungsgraphen zu finden und die zugehörigen Knoten und Kanten zu reservieren. Keine Knoten und keine Kanten des Verdrahtungsgraphen dürfen von mehr als einem Netz gemeinsam in Anspruch genommen werden.

Die bekannten FPGA-Verdrahtungsverfahren können in zwei Kategorien eingeteilt werden: einstufige Verfahren mit detaillierter Verdrahtung und zweistufige Verfahren mit sequentieller globaler und detaillierter Verdrahtung [78]. Verdrahtungsgraphen moderner FPGAs können Größen von mehreren Millionen Kanten und Knoten aufweisen. So besteht der Verdrahtungsgraph für den größten Virtex-II-FPGA von Xilinx aus etwa 60 Millionen Kanten und 6 Millionen Knoten [78]. Um diese Komplexität besser handhaben zu können, führen die zweistufigen Verdrahtungsverfahren zunächst eine globale Verdrahtungsplanung auf einem weniger komplexen Verdrahtungsgraphen durch, der nur einzelne Verdrahtungskanäle und Logikblöcke repräsentiert. Im nachfolgenden Schritt der detaillierten Verdrahtung wird dann versucht, die geplanten Verdrahtungen im detaillierten Verdrahtungsgraphen zu konkretisieren. Diese Vorgehensweise ist nicht bei jeder FPGA-Architektur anwendbar, da der Verdrahtungsgraph zur globalen Verdrahtung beispielsweise von Implementierungsdetails der Switch-Boxen und Segmentlängen abstrahiert [79]. Durch die Vorstufe der globalen Verdrahtung entsteht zwangsläufig die zusätzliche Aufgabe, einen geeigneten globalen Verdrahtungsgraphen für eine gegebene FPGA-Zielarchitektur zu formulieren.

Ein nur auf dem detaillierten Verdrahtungsgraphen operierendes und somit für jede FPGA-Architektur geeignetes Verdrahtungsverfahren ist PathFinder [80]. Sein Erfolgskonzept beruht auf einer neuartigen Kostenfunktion und der erlaubten Mehrfachnutzung der Verdrahtungsressourcen. Die Mehrfachnutzung wird bei jeder Iteration immer stärker reduziert, bis schließlich eine exklusive Nutzung der Ressourcen vorliegt. Zuerst werden alle gewünschten Verbindungen unter Verwendung des Verdrahtungsgraphen auf den kürzesten Pfaden verdrahtet. Das führt zwangsläufig zu mehrfach belegten Ressourcen. Bei den nachfolgenden Iterationen wird dann immer wieder geprüft, welche Ressourcen mehrfach belegt wurden. Die Verdrahtung von weniger zeitkritischen Netzen durch die mehrfach benutzten Ressourcen wird aufgebrochen, um anschließend bei geringeren Verdrahtungskosten neu zu verdrahten. Die Kostenfunktion berücksichtigt die Zeitkritizität der Netze, die Verzögerungszeiten der einzelnen Verdrahtungsressourcenarten, die Überbenutzung und die Überbenutzungsvorgeschichte der Verdrahtungsressourcen. Entsprechend der Kostenfunktion werden die aufgebrochenen, zeitunkritischen Netze über langsame Verdrahtungsressourcen, die in der Vergangenheit seltener überbenutzt wurden, neu verdrahtet. Die Verdrahtung der zeitkritischen Netze auf den kürzesten Pfaden bleibt bestehen. Die Verdrahtungsiterationen werden solange wiederholt, bis jede Verdrahtungsressource von nur noch einem Netz exklusiv benutzt wird.

Der Verdrahtungsalgorithmus des bereits erwähnten Platzier- und Verdrahtungswerkzeugs VPR [68] basiert auf PathFinder. Neben einer leicht modifizierten Kostenfunktion, die

Verdrahtungen im rechten Winkel mit zusätzlichen Kosten bestraft, wurden zwei Strategien zur Reduzierung der Verdrahtungszeit implementiert. Zum einem wird der Suchraum für einen Pfad auf die Bounding-Box-Größe zuzüglich dreier Verdrahtungskanäle auf jeder Seite der Bounding-Box eingeschränkt. Zum anderen wurde eine Strategie zum Verdrahten der Netze mit hohem Fanout vorgestellt, die diesbezüglich die Rechenzeit um eine Größenordnung reduziert.

Die Optimierung der Verdrahtbarkeit und der Verzögerungszeiten kann beim Entwurfsschritt der Verdrahtung auch durch Vertauschen der logischen Funktion von LUT-Eingängen erfolgen [81]. In ihrer Funktionalität sind die LUT-Eingänge äquivalent, unterscheiden sich jedoch aufgrund der LUT-Inhalte in der logischen Funktion. Durch eine geeignete Permutation der LUT-Inhalte kann die logische Funktion der LUT-Eingänge vertauscht werden. Dadurch werden die Zielknoten in dem Verdrahtungsgraphen mitvertauscht, was eventuell eine einfachere und/oder schnellere Verdrahtung ermöglicht. Da bei realen FPGA-Architekturen die Signallaufzeiten von den einzelnen LUT-Eingängen zum Ausgang der LUT in der Regel unterschiedlich sind [97], stellt die Permutation der LUT-Inhalte eine sehr wirksame Strategie zur Reduzierung der Signallaufzeiten dar. Die Wirksamkeit beruht auch auf der Tatsache, dass die Platzierung nicht geändert werden muss und dass während der Verdrahtung eine genaue Timing-Analyse durchgeführt werden kann.

2.3.9 Statische Timing-Analyse

Die Timing-Analyse wird zur Bestimmung der maximalen Taktfrequenz einer Schaltung und des Schlupfes (engl.: slack) der einzelnen Pfade eingesetzt [75]. Grundlage für die Timing-Analyse bilden die Positionen der LUTs und Flipflops und die zu deren Verdrahtung verwendeten Verdrahtungsressourcen. Die Timing-Analyse erfolgt in der Regel ohne Berücksichtigung der Signalzustände und der Funktion der Logik, ist also statisch. In einem ersten Schritt wird die Struktur der kombinatorischen Schaltungsteile in gerichtete, azyklische Graphen umgesetzt. Dessen Knoten stellen die Logikressourcen, dessen Kanten die Verbindungen zwischen diesen dar (Abbildung 2-25).

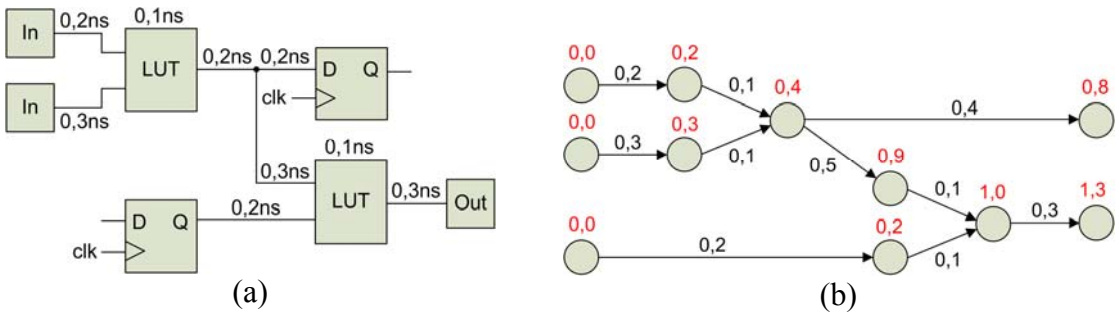


Abbildung 2-25: *Schaltungsstruktur (a), Darstellung als Graph und Vorwärtsrechnung (b).*

In einem zweiten Schritt werden die Kanten des Graphen mit den Verzögerungszeiten der zugehörigen Logik und mit den Verzögerungszeiten aller zu Verdrahtung benötigten Verdrahtungselemente versehen (Abbildung 2-25b). Bei der darauf angewendeten Vorwärtsrechnung wird durch Traversieren des Graphen die Summen aller Verzögerungszeiten für alle mögliche Signalpfade von den Eingängen zu den Ausgängen bestimmt und über die Knoten aufgetragen (Abbildung 2-25b). Die maximale Taktfrequenz einer Schaltung ergibt sich dann aus dem Kehrwert der höchsten unter allen Pfaden ermittelten Gesamtverzögerungszeit.

Vor der Schlupfberechnung muss zunächst eine Rückwärtsrechnung durchführt werden. Dazu werden die Ausgangsknoten mit der höchsten bei der Vorwärtsrechnung ermittelten Verzögerungszeit versehen und der Graph von den Ausgängen zu den Eingängen hin traversiert (Abbildung 2-26a). Zur Schlupfberechnung werden dann die Differenzen aus der Vorwärts- und Rückwärtsrechnung gebildet und über den Knoten des Timing-Graphen aufgetragen (Abbildung 2-26b). Der Schlupf ist als die Zeit definiert, die zu einer Verbindung hinzugefügt werden kann, ohne dass diese kritisch wird [75]. Knoten mit Schlupf Null liegen auf den kritischen Pfaden. Zusätzliche Verzögerungen auf den kritischen Pfaden führen zwangsläufig zur geringeren maximalen Taktfrequenz der Schaltung. Verbindungen auf Pfaden mit hohem Schlupf können ihrerseits über langsamere Verdrahtungsressourcen bzw. längere Entfernungen verdrahtet werden, ohne dass die Schaltung langsamer getaktet werden muss.

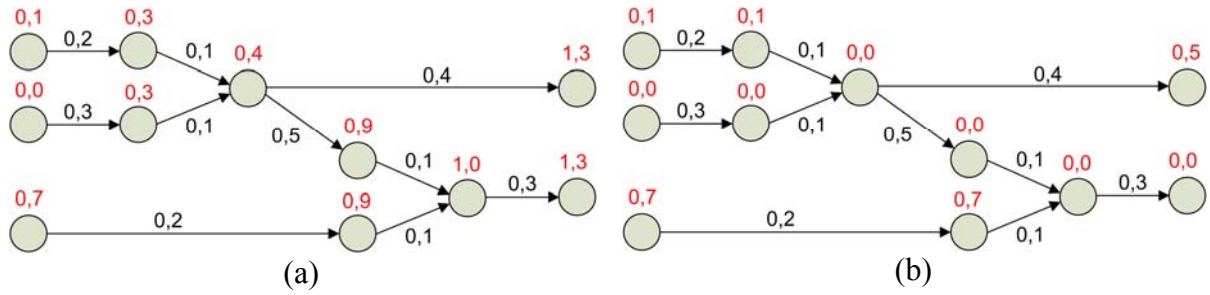


Abbildung 2-26: *Rückwärtsrechnung (a) und Schlupfberechnung (b).*

Über den Schlupf wird die Zeitkritizität *criticality* einer Verbindung definiert [14]:

$$criticality = 1 - \frac{slack}{maxslack}$$

wobei *slack* der Schlupf der betrachteten Verbindung und *maxslack* der maximale Schlupf der Schaltung ist. Zeitkritizität ist ein Wert zwischen Eins und Null und definiert den Einfluss einer Verbindung auf das Timing einer Schaltung. Wird die Timing-Analyse während der Platzierung durchgeführt, kann die Zeitkritizität zur verzögerungszeitenoptimierenden Platzierung (engl.: timing-driven placement) benutzt werden. Dabei werden entweder die Pfade oder die Netze entsprechend ihres Zeitkritizität mit Gewichtungsfaktoren versehen [70].

Die statische Timing-Analyse einer mit FPGA realisierten Schaltung erfolgt anhand der von den FPGA-Herstellern für alle Ressourcen spezifizierten Verzögerungszeiten. Neben den Werkzeugen zur Timing-Analyse werden von ihnen auch die Verzögerungszeiten in Form von Timing-Bibliotheken zur Verfügung gestellt. Die Bestimmung der Verzögerungszeiten wird von den FPGA-Herstellern aufwendig mit Schaltungssimulatoren wie SPICE [77] anhand der topographischen Modelle einer FPGA-Architektur durchgeführt, die auch Fertigungsprozessvariationen berücksichtigen. Für diese Variationen ist primär der selten zu vermeidende Maskenversatz bei den einzelnen Lithographieschritten verantwortlich, der sich in nicht exakt gefertigten Halbleiterstrukturen und unter Umständen in höheren Verzögerungszeiten bemerkbar macht. Nach der Fertigung werden die FPGAs einzeln auf das Einhalten der Verzögerungszeiten getestet und in unterschiedliche (in der Regel 3 bis 4) Geschwindigkeitsgrade (engl.: speed grade) eingeteilt. Anhand des vom Benutzer eingestellten Geschwindigkeitsgrades verwendet die statische Timing-Analyse jeweils die richtige Timing-Bibliothek. Diese Vorgehensweise bietet den Vorteil, dass die aufwendige Timing-Simulation anhand der Spice-Modelle nur einmal vom FPGA-Hersteller durchgeführt werden muss und der FPGA-Anwender die wesentlich einfachere statische Timing-Analyse verwenden kann.

3 Softwarepaket PALLAs

3.1 Altera Quartus University Interface Program

Der kommerzielle FPGA-Hersteller Altera bietet mit der Altera-Quartus-II-Software (im Folgenden einfach Quartus) eine integrierte Entwicklungsumgebung mit einem durchgehenden Entwurfsfluss für seine FPGAs an. Das seit dem Jahr 2004 verfügbare Altera Quartus University Interface Program (QUIP, [108]) beschreibt die Schnittstellen zwischen den einzelnen Entwurfsschritten der Quartus-Software und die Art und Weise, wie sich diese Entwurfsschritte durch eigene Algorithmen ersetzen lassen. Die Intention für das Offenlegen der Schnittstellen ist der Wunsch, dass Forscher die Wirksamkeit ihrer Algorithmen an real existierenden FPGA-Architekturen unter Beweis stellen können und somit indirekt durch daraus resultierenden Veröffentlichungen an der Verbesserung der kommerziellen Quartus-Software mitarbeiten. Für einen an bestimmten Entwurfsschritten interessierten Forscher erübrigt sich mit QUIP also die Notwendigkeit, einen kompletten FPGA-Entwurfsfluss von der RTL-Synthese bis zur Timing-Analyse implementieren zu müssen. Das wäre für real existierende FPGA-Architekturen auch gar nicht möglich, da der detaillierte Aufbau der kommerziellen FPGAs von den jeweiligen Herstellern geheim gehalten wird. Die Vergleichbarkeit der erreichten Ergebnisse wird durch eine mit QUIP mitgelieferte Benchmark-Sammlung und durch nicht manipulierbare Entwurfsschritte der Timing- und Ressourcenanalyse sichergestellt. Folgende Eingriffe in den Entwurfsfluss können bei der Quartus-Software einzeln oder in Kombination durchgeführt werden [108]:

- Ein eigenes Synthese-Werkzeug kann eine Gatter-Netzliste in einem vorgegebenen Format direkt dem Quartus-Entwurfsfluss zuführen. Die nachfolgenden Schritte der Logikminimierung und der Technologieabbildung übernimmt die Quartus-Software.
- Die von Quartus erzeugte Technologieabbildung kann nachträglich manipuliert werden, indem die ursprüngliche Netzliste exportiert, verändert und wieder importiert wird. Die Quartus-Software platziert und verdrahtet anschließend die Netzliste.
- Die Netzliste kann nach der Technologieabbildung von Quartus ausgegeben und mit eigener Software platziert werden. Nach der Platzierung wird jedes Element der Netzliste (LUTs, FFs, RAM-Blöcke) in einer Quartus-Constraint-Datei (QSF) mit den absoluten Platzierkoordinaten versehen. Anschließend werden diese Vorgaben von Quartus eingelesen, das Design verdrahtet und einer Timing-Analyse unterzogen.
- Über die Vorgaben in der QSF-Datei kann auch eine Flächenplanung durchgeführt werden. Dazu werden den RTL-Funktionseinheiten Bereiche auf dem FPGA-Chip zugewiesen, die bei der Platzierung mit der Quartus-Software von deren Platzieralgorithmus eingehalten werden.
- Da die detaillierten Verdrahtungsarchitekturen von Altera geheim gehalten werden, kann außerhalb von der Quartus-Software nur eine globale Verdrahtungsplanung implementiert werden. Hierbei können dem Quartus-Verdrahtungswerkzeug für jedes einzelne Netz die Art der zu verwendenden Verdrahtungssegmente und Verdrahtungskanäle über die QSF-Datei mitgeteilt werden. Die detaillierte Verdrahtung wird anhand dieser Vorgaben erzeugt.

Bisher werden von dem Altera Quartus University Interface Program die Altera-Cyclone- und Stratix-FPGAs der ersten und zweiten Generation unterstützt. Die Architekturen der einzelnen FPGAs und die detaillierte Verzögerungszeiten für die unterschiedlichen FPGA-Geschwindigkeitsgrade können aus den mitgelieferten XML-Dateien extrahiert werden.

3.2 PALLAs-Entwurfsfluss

Um das im Rahmen dieser Arbeit entwickelte PALLAs-Werkzeug in den Quartus-Entwurfsfluss einbetten zu können, wurden die im Altera Quartus University Interface Program beschriebenen Schnittstellen benutzt. Die Quartus-Software wird dabei zunächst für die Synthese der HDL-Schaltungsbeschreibungen in Form von VHDL- bzw. Verilog-Dateien eingesetzt. Nach der Synthese, die die Technologieabbildung beinhaltet, liegt dann eine Netzliste für die FPGA-Zielarchitektur vor (Abbildung 3-1).

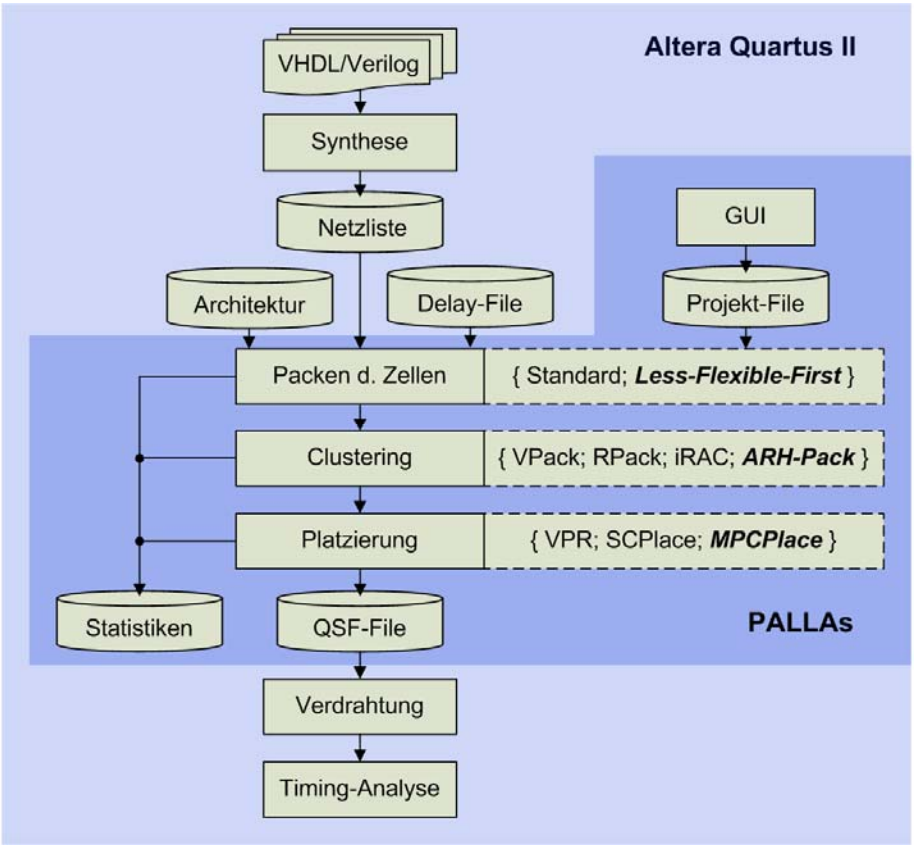


Abbildung 3-1: *In die Altera Quartus II Software eingebetteter PALLAs Entwurfsfluss. (Neu entwickelte Algorithmen sind fett hervorgehoben.)*

PALLAs liest dann die aus LUTs, Flipflops und RAM-Grundzellen bestehende Netzliste zusammen mit einer Architektur- und Verzögerungszeitendatei für die FPGA-Zielarchitektur ein. Die mit PALLAs durchführbaren Entwurfsschritte sind Packen der LUTs und Flipflops in die Logikzellen, Clustering der Logikzellen bzw. der RAM-Grundzellen, und Platzieren der Logikböcke bzw. der geclusterten RAM-Blöcke. Für jeden PALLAs-Entwurfsschritt stehen Algorithmen aus einem Pool an Alternativen zur Verfügung. Welche Algorithmen in den einzelnen Entwurfsschritten eingesetzt werden, entscheidet eine PALLAs-Projektdatei. Diese kann bequem aus der graphischen PALLAs-Benutzeroberfläche (GUI) angepasst werden. Während eines jeden Entwurfsschritts werden detaillierte Statistiken in Protokolldateien geschrieben, die später anhand der verwendeten Metriken für die Bewertung der Ergebnisse herangezogen werden können. Nach der Platzierung liegen für alle LUTs, Flipflops und RAM-Grundzellen der ursprünglichen Quartus-Netzliste absolute Koordinaten für das Ziel-FPGA vor. Diese Koordinaten werden von PALLAs in die QSF-Datei geschrieben. Die QSF-Datei wird von der Altera-Quartus-II-Software eingelesen, die Platzierkoordinaten übernommen und anschließend das Design verdrahtet. Das fertig verdrahtete Design kann danach einer Timing- und der Ressourcenanalyse unterzogen werden.

3.3 Implementierte Algorithmen

Wie bereits erwähnt, können bei dem PALLAs-Werkzeug für jeden Entwurfsschritt unterschiedliche Algorithmen ausgewählt werden. Folgende Alternativen stehen bei den einzelnen Entwurfsschritten zur Verfügung:

- Das **Packen** der LUTs und Flipflops in die Logikzellen kann entweder mit einer einfachen Mustererkennung (bisherige Standardvorgehensweise, u. a. beschrieben in [42]) oder nach dem Less-Flexible-First-Prinzip (LFF-Prinzip) erfolgen. Das Packen der LUTs und Flipflops nach dem LFF-Prinzip ist eine neue Vorgehensweise, die im Rahmen der PALLAs-Implementierung entwickelt wurde. Eine ausführliche Beschreibung der Strategien des neuen Pack-Verfahrens findet sich in Kapitel 4.
- Das **Clustering** der Logikzellen kann entweder mit VPack [43], RPack [44], iRAC [38] oder ARH-Pack durchgeführt werden. ARH-Pack (Area, Routability and Hierarchy driven Packer) ist ein neu entwickeltes „bottom-up"-Clusteringverfahren, welches die Verdrahtbarkeit und die Logikausnutzung der erzeugten Logikblöcke verbessert. Die Algorithmen des Verfahrens und die neuartige Kostenfunktion F_{gain}, die zur Auswahl der Logikzellen eingesetzt wird, werden in Kapitel 5 beschrieben.
- Das **Platzieren** der Logikzellen bzw. Logikblöcke und eventuell vorhandener RAM-Blöcke kann mit den Platzierverfahren SCPlace [71], VPR [68] oder MPCPlace (Multipass Clustering and Placement) erfolgen. MPCPlace wurde ebenfalls im Rahmen der Implementierung des PALLAs-Werkzeugs entwickelt und wird ausführlich in Kapitel 6 beschrieben.

3.4 Optimierungsziele und verwendete Metriken

Die neu entwickelten Algorithmen und die in PALLAs darüber hinaus implementierten Referenzverfahren haben das Ziel, die Verdrahtbarkeit über die einzelnen Entwurfsschritte hinweg zu optimieren. Um die Effektivität der neu entwickelten Algorithmen und der Referenzverfahren miteinander vergleichen zu können, werden bereits bekannte Metriken [14][46][84] auf die Ergebnisse der einzelnen Entwurfsschritte angewendet und ausgewertet. Je nach Entwurfsschritt mussten sowohl die verfolgten Teiloptimierungsziele zur Steigerung der Verdrahtbarkeit als auch die Metriken zur deren Auswertung angepasst werden. Mit den einzelnen Entwurfsschritten steigt der Detailgrad einer Design-Implementierung und ermöglicht so eine Optimierung und Auswertung zusätzlicher Faktoren mit zunehmender Korrelation zur Verdrahtbarkeit. Je nach Entwurfsschritt werden von den neu entwickelten Algorithmen folgende Optimierungsziele verfolgt:

- Beim **Packen** der LUTs und Flipflops in die Logikzellen ist die Gesamtzahl der entstehenden Logikzellen zu minimieren. Dadurch soll die möglichst effektive Nutzung der Logikblockressourcen in dem darauf folgenden Clustering-Entwurfsschritt ermöglicht werden.
- Beim **Clustering** der Logikzellen kann eine möglichst hohe Ausnutzung der Logikblockressourcen die Nutzung der globalen Verdrahtungsressourcen verringern und somit die Verdrahtbarkeit steigern [14][46]. Ein Maß für die effektive Nutzung der Logikblockressourcen ist die Gesamtanzahl der global zu verdrahtenden Netze, das Gesamtfanout dieser Netze und die Anzahl aller Logikblöcke einer Netzliste.
- Beim **Platzieren** sind die Gesamtanzahl der zu Verdrahtung erforderlichen globalen Verdrahtungssegmente und ihre lokale Überbenutzung zu minimieren. Da erst nach der Verdrahtung die dazu erforderlichen globalen Verdrahtungsressourcen feststehen, muss beim Platzierentwurfsschritt die Nutzung der Verdrahtungsressourcen auf geeignete Weise modelliert und minimiert werden.

Die Zusammenhänge zwischen den verfolgten Optimierungszielen und der Verdrahtbarkeit im Einzelnen werden im Detail bei den Beschreibungen der neu implementierten Algorithmen aufgezeigt. Die mit den implementierten Algorithmen erreichten Ergebnisse werden ausführlich in Kapitel 7 verglichen und analysiert.

3.5 Beschreibung der unterstützten FPGA-Zielarchitekturen

Als Zielarchitekturen für die im Rahmen der vorliegenden Arbeit implementierten und verglichenen Algorithmen wurden die FPGAs der Altera-Stratix- und Cyclone-Familien ausgewählt. Beide Familien werden in einem 130nm-Fertigungsprozess hergestellt. Die innerhalb der FPGAs angeordneten Logikblöcke bilden ein regelmäßiges Raster, in dem noch weitere Funktionseinheiten in Spalten angeordnet sind (Abbildung 3-2); die Logikblock-zwischenräume symbolisieren die Verdrahtungskanäle. Die RAM-Ressourcen sind über insgesamt drei verschiedene Block-Arten verteilt: kleine M512-Blöcke mit 576 Bit SRAM, mittlere M4K-Blöcke mit 4.608 Bit SRAM und große MegaRAM-Blöcke mit 576 KBit SRAM. Die MegaRAM-Blöcke, wie auch die DSP-Blöcke sind nicht auf den FPGAs der Cyclone-Familie enthalten. Die RAM-Blöcke der Altera-Stratix- und Cyclone-Architekturen sind wie Logikblöcke hierarchisch aufgebaut, d. h. die M512-, M4K- und MegaRAM-Blöcke bestehen aus unterschiedlich vielen, aber identischen RAM-Grundzellen [119].

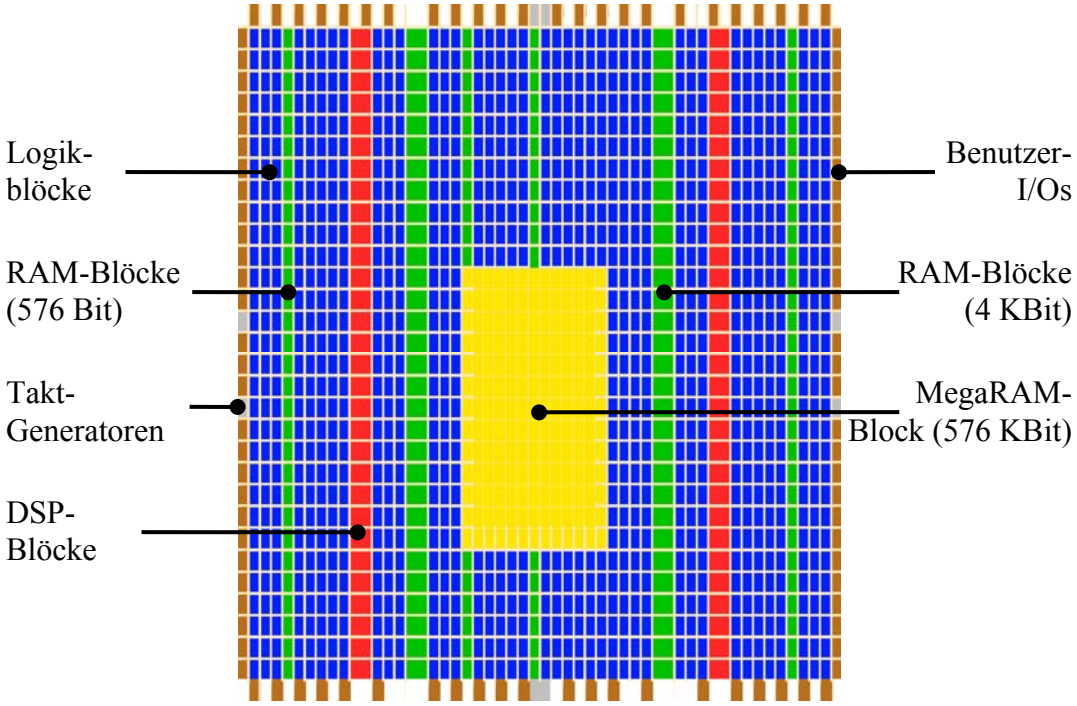


Abbildung 3-2: *Beispiel-Topologie eines Altera-Stratix-FPGAs.*

3.5.1 Logikzellenarchitektur

Eine Logikzelle der Altera-Stratix- und Cyclone-FPGAs besteht aus einer LUT mit vier Eingängen und einem D-Flipflop (Abbildung 3-3). Aus Effizienzgründen kann der Flipflop-Ausgang an den LUT-Eingang innerhalb der Logikzelle rückgekoppelt werden. Dazu wurde dem Eingang *a* der LUT ein programmierbarer Multiplexer vorgeschaltet, der zwischen dem Flipflop-Ausgang und einem Anschluss an den lokalen Eingangsbus entscheiden kann. So kann wahlweise ein der LUT vor- oder nachgeschaltetes Flipflop implementiert werden [117]. Durch die unabhängigen Flipflop- und LUT-Ausgänge können pro Logikzelle zwei Signale

erzeugt und für eine lokale Verdrahtung innerhalb des Logikblocks nach Außen geschaltet werden.

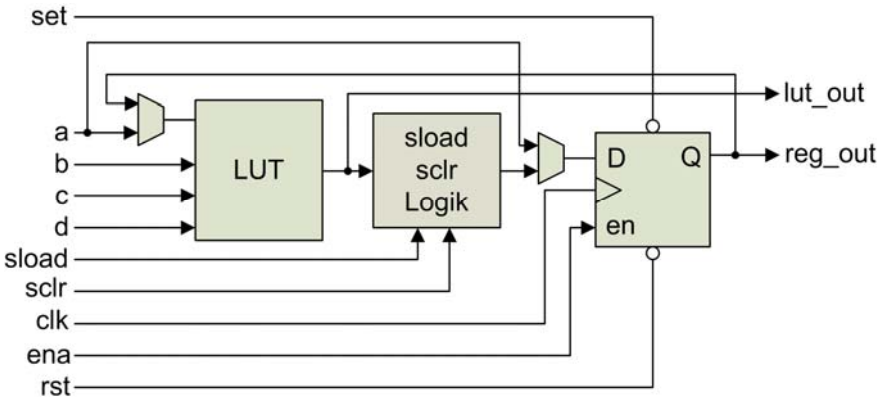


Abbildung 3-3: *Aufbau einer Logikzelle.*

Außer den Ein- und Ausgängen der LUT und des Flipflops werden innerhalb der Logikzelle Steuersignale (*set, sload, sclr, clk, ena, rst*) verwendet, mit denen sich der Flipflop-Zustand steuern lässt. So kann der Ausgang *Q* des Flipflops unabhängig vom Taktsignal *clk* mit den Signalen *set* und *rst* asynchron gesetzt bzw. gelöscht werden. Die Signale *sload* und *sclr* ermöglichen ein zum Taktsignal synchrones Laden bzw. Löschen des Flipflop-Ausgangs *Q*. Eine effizientere Abbildung oft verwendeter VHDL-Konstrukte auf die Logikzellen-ressourcen wird durch diese zusätzlichen Steuersignale (vgl. Abbildung 2-7) ermöglicht. Sie können mittels einer dedizierten Multiplexerstruktur sowohl aus den lokalen als auch aus globalen Signalen erzeugt werden [116] (Abbildung 3-4). Diese Schaltungsstruktur ist nur ein Mal pro Logikblock vorhanden. Das heißt, alle Logikzellen eines Logikblocks teilen sich die durch die Multiplexerstruktur erzeugten, logikblockweiten Signale. Dadurch ergeben sich einige Einschränkungen, weil im Einzelnen pro Logikblock nur folgende Signale und Kombinationen erlaubt sind:

- zwei CLK/EN – Signalpaare,
- ein synchrones und ein asynchrones Ladesignale,
- ein synchrones und zwei asynchrone Rücksetzsignale.

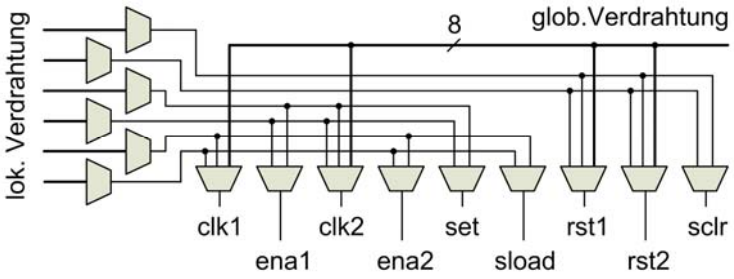
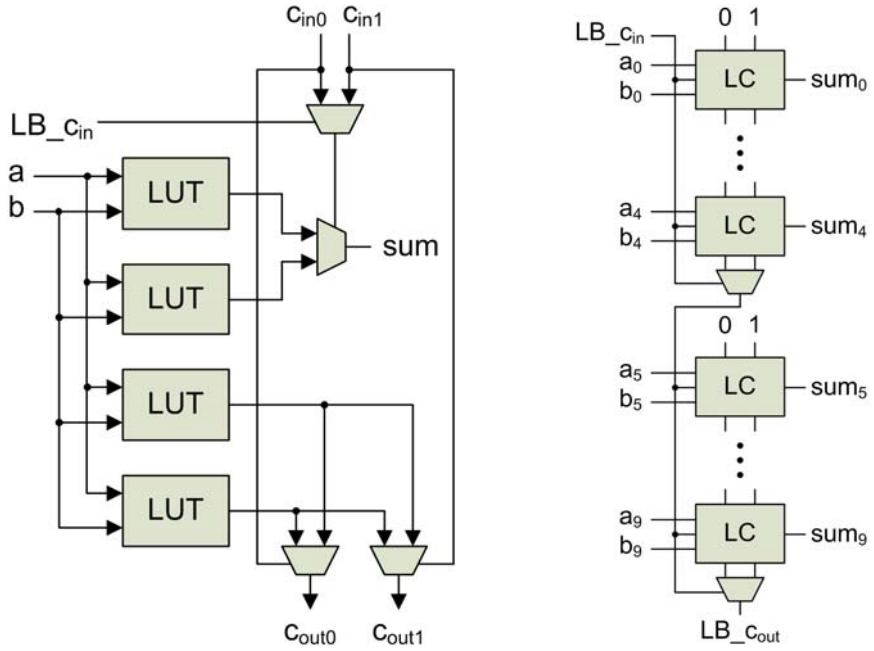


Abbildung 3-4: *Multiplexerstruktur zur der Erzeugung Flipflop-Steuersignale.*

3.5.2 Carry-Chains

Die Logikzellen der Altera-Stratix- und Cyclone-FPGAs können zur Implementierung arithmetischer Schaltungen in den arithmetischen Modus versetzt werden [116]. Dieser erlaubt unter anderem eine schnelle und effiziente Realisierung häufig vorkommender Addiererschaltungen. Im arithmetischen Modus kann ein kompletter 1-Bit-Volladdierer nach dem Carry-Select-Prinzip auf eine einzige Logikzelle abgebildet werden. Dazu wird die LUT einer Logikzelle in vier LUTs mit jeweils zwei Eingängen aufgeteilt (Abbildung 3-4a). Zwei LUTs bilden die Summen der beiden Dateneingänge und die beiden anderen LUTs die Überträge für jeweils beide möglichen Übertragswerte aus der bitniedrigeren Carry-Select-Addiererstufe. Je nach dem, wie der Übertrag nach der Berechnung in der bitniedrigeren Stelle tatsächlich ausfällt, wird mit den dedizierten Multiplexern zu dem korrekten Summen- bzw. Übertragsergebnis umgeschaltet. Da die Signallaufzeiten durch die Multiplexer unter denen von LUTs liegen, ist ein derart realisierter Carry-Select-Addierer schneller als einer, der nur aus LUTs aufgebaut ist.



(a) (b)

Abbildung 3-5: *LUT-Konfiguration einer Logikzelle (LC) im arithmetischen Modus (a), Zusammenschaltung der Logikzellen eines Logikblocks zu einer Carry-Chain (b).*

Die Logikzellen eines Logikblocks können die berechneten Überträge über dedizierte Verdrahtungssegmente direkt an die benachbarten Logikzellen weiterreichen. Durch die Logikblockstruktur ist die Zuordnung der Bitstellen zu den Logikzellen fest vorgegeben (Abbildung 3-5b). Die oberste Logikzelle eines Logikblocks muss die Summe der niedrigsten Bitstellen beider Summanden berechnen, die beiden möglichen Überträge werden zu der tiefer angeordneten Logikzelle weitergeleitet. Die auf diese Weise angeordneten und auf die Logikzellen abgebildeten 1-Bit-Volladdierer werden als Carry-Chains bezeichnet. Ein Logikblock kann eine Carry-Chain mit der Länge von genau zehn oder weniger bzw. zwei Carry-Chains mit der Länge von genau fünf oder weniger 1-Bit-Volladdierern aufnehmen. Der in einem Logikblock erzeugte Übertrag kann nur an den direkt unterhalb angeordneten Logikblock weitergereicht werden. Die maximale Länge einer Carry-Chain wird durch die Anzahl der in einer Spalte verfügbaren Logikblöcke auf einem FPGA begrenzt.

3.5.3 Logikblockarchitektur und lokale Verdrahtungsressourcen

Ein Logikblock der Altera-Stratix- und Cyclone-FPGAs beinhaltet 10 Logikzellen. Jede Logikzelle verfügt über zwei Ausgänge (vgl. Abbildung 3-3). Alle in einem Logikblock erzeugten Logikzellensignale können über einen 20 Leitungen umfassenden Ausgangsbus nach Außen zur globalen Verdrahtung geschaltet werden. Die Hälfte der im Logikblock erzeugten Signale kann über einen lokalen Rückkoppelbus an die Eingänge der LUTs und Flipflops zurückgeführt werden (Abbildung 3-6). Darüber hinaus können bis zu 26 (Cyclone) bzw. 30 (Stratix) global verdrahtete Signale über einen 30 Leitungen umfassenden lokalen Eingangsbus mit den LUT- und Flipflop-Eingängen verbunden werden. Die von der Multiplexerstruktur (vgl. Abbildung 3-4) erzeugten, logikblockweiten Flipflop-Steuersignale sind an den 9 Segmente umfassenden Steuerbus angeschlossen – die Signale dieses Steuerbusses können nur mit den Flipflop-Steuereingängen und nicht mit den LUT-Eingängen verdrahtet werden. Die aufgezählten Verdrahtungsbusse bilden die lokalen Intra-Logikblockverdrahtungsressourcen innerhalb eines Logikblocks.

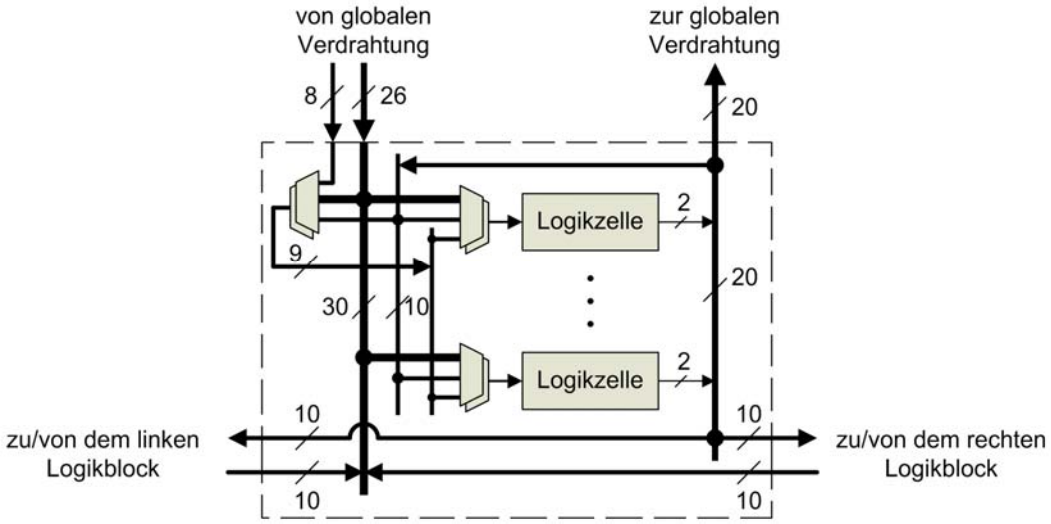


Abbildung 3-6: *Struktur eines Logikblocks der Altera-Cyclone-Architektur.*

Jeweils 10 intern generierte Signale können vom Ausgangsbus sowohl zu den direkt links als auch zu den direkt rechts angrenzenden Logikblöcken über die lokalen Inter-Logikblockverdrahtungsressourcen auf den lokalen Eingangsbus geschaltet werden (Abbildung 3-6). Auf diese Weise können bis zu 40 Signale zwischen einem Logikblock und seinen direkten Nachbarn lokal verdrahtet werden, ohne dafür globale Verdrahtungsressourcen in Anspruch zu nehmen.

3.5.4 Globale Verdrahtungsressourcen

Die Verdrahtung auf der globalen Ebene erfolgt bei den Altera-FPGAs der Stratix- und Cyclone-Familien mit Hilfe der unidirektionalen Verdrahtungssegmente und Switch-Boxen. Der Aufbau und die Vorteile der unidirektionalen Verdrahtungsarchitektur wurden bereits in den Abschnitten 2.2.3.4 und 2.2.4 angesprochen. Die Verdrahtung in den horizontalen Verdrahtungskanälen der Altera-Cyclone-FPGAs erfolgt mit Hilfe horizontaler Segmente, R4, die jeweils vier benachbarte Logikblöcke zur linken und rechten Seite des Quelllogikblocks überbrücken (Abbildung 3-7).

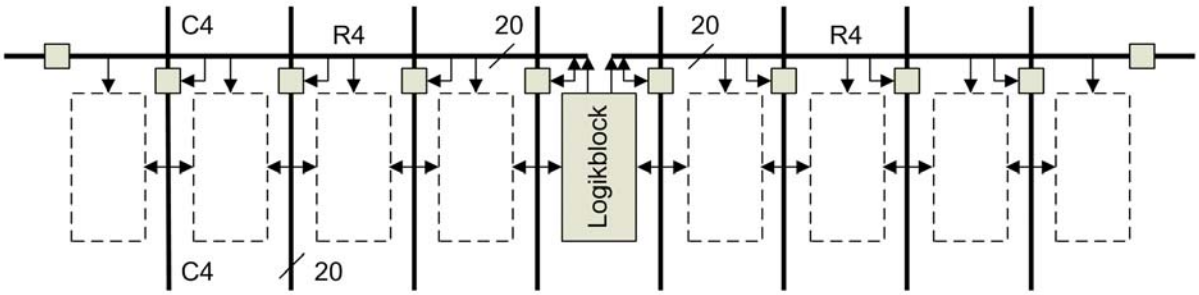


Abbildung 3-7: *Globale Verdrahtungsressourcen der Altera-Cyclone-Architektur.*

Jeder Logikblock bildet den Anfang von insgesamt 40 R4-Segmenten, die zu gleichen Teilen rechts als auch links von ihm liegen. Am Ende eines jeden Segmentbündels liegen Switch-Boxen, die das Weiterschalten der Signale in gleicher Richtung ermöglichen. In vertikaler Richtung erfolgt die Verdrahtung der Signale mit Hilfe vertikaler C4-Segmente, deren Ausdehnung auch jeweils vier Logikblöcke umfasst. An den Kreuzungspunkten der R4- und C4-Segmente liegen weitere Switch-Boxen, die das Verbinden der Segmente untereinander ermöglichen. Die von den horizontalen R4- und vertikalen C4-Segmenten überbrückten Logikblöcke können an diese angeschlossen werden.

Die Altera-Stratix-FPGAs besitzen zusätzliche R8- und R24-Verdrahtungssegmente, die horizontal 8 bzw. 24 Logikblöcke überbrücken können. Vertikal kommen zur Überbrückung der Entfernungen von 8 bzw. 16 Logikblöcken die C8- und C16-Segmente zum Einsatz. Diese zusätzlichen Verdrahtungsressourcen werden durch die größere Anzahl der Metallisierungslagen der Stratix-FPGAs im Vergleich zu den Cyclone-FPGAs ermöglicht. Das Seitenverhältnis der Logikblöcke ist circa 1:2, was dazu führt, dass bei den zusätzlichen Metalllagen die Anzahl der horizontalen Verdrahtungssegmente auf diesen Lagen etwa doppelt so hoch ist wie die Anzahl der vertikalen Segmente [29].

Jedes Verdrahtungssegment wird bei den Altera-FPGAs über einen ausreichend starken Treiber getrieben. Die Gesamtverzögerungszeiten unterschiedlich langer, globaler Segmente sind aufgrund der Treiber- und Segmentdimensionierungen näherungsweise identisch. Bei der horizontalen Verdrahtung eines Signals über eine Entfernung von 4 der 8 Kacheln können zur Verdrahtung bei den Stratix-FPGAs ein R4-Segment bzw. ein R8-Segment herangezogen werden. Obwohl im zweiten Fall die Entfernung etwa doppelt so groß ist, ergibt sich nur eine geringfügig höhere Verzögerungszeit [90]. Andererseits sind die Verzögerungszeiten bei der Überbrückung von 8 Kacheln mit zwei zusammengeschalteten R4-Segmenten etwa doppelt so hoch wie unter Verwendung eines R8-Segments. Die Annahme, dass die Signal-verzögerungszeit proportional zur überbrückenden Entfernung ist, gilt somit bei den aktuellen FPGAs mit heterogenen globalen Verdrahtungsressourcen nur eingeschränkt [89]. Vielmehr ist die Art der zur Verdrahtung benutzten Ressourcen ausschlaggebend. Erfolgt die Verdrahtung unter Zuhilfenahme gleich langer globaler Segmente, so wie es bei den Cyclone-FPGAs der Fall ist, kann weiterhin angenommen werden, dass die Signalverzögerungszeiten proportional zur Entfernung sind.

4 Packen der Logikzellen

Aufgrund des von der klassischen FPGA-Architektur abweichenden Logikzellenaufbaus der Altera-FPGAs hat sich die bisherige Standard-Vorgehensweise [42] zum Packen der LUTs und Flipflops in die Logikzellen als ineffektiv herausgestellt. Deshalb wurde im Rahmen der Entwicklung von PALLAs eine neue Packstrategie entworfen, die die Flexibilität der LUTs und Flipflops einer Netzliste bzgl. ihrer Packmöglichkeiten berücksichtigt. Bevor diese Strategie ausführlich erläutert wird, werden zunächst die hierbei genutzten Architekturmerkmale der Altera-Logikzellen vorgestellt und die Problematik der bisherigen Vorgehensweise an einem Beispiel aufgezeigt.

4.1 Mögliche Konfigurationen der Logikzellen

Die atomaren Schaltungselemente einer von der Altera-Quartus-II-Software erzeugten Netzliste sind LUTs, Flipflops, RAM-Grundzellen und Ein- bzw. Ausgabepins. Vor dem Clustering werden die LUTs und die Flipflops in die Logikzellen gepackt. Die Logikzellen der Altera-Stratix- und Cyclone-Architekturen können maximal eine LUT und/oder ein Flipflop aufnehmen (vgl. Abbildung 3-3). Ein Flipflop, das mit einer LUT in eine Logikzelle gepackt werden soll, kann entweder einem LUT-Eingang vor- oder dem LUT-Ausgang nachgeschaltet sein (Abbildung 4-1). Diese höhere Flexibilität der Logikzelle wird durch zwei zusätzliche Multiplexer ermöglicht.

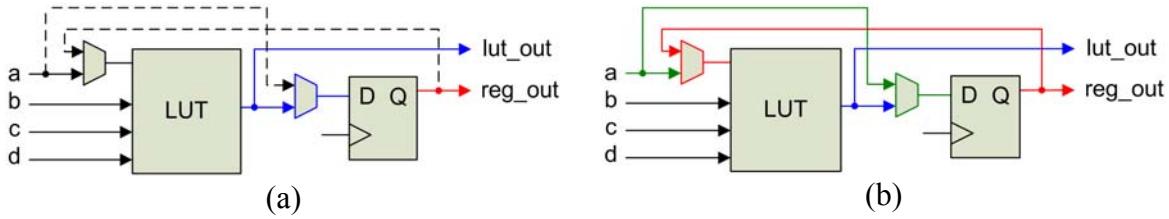


Abbildung 4-1: *Konfiguration der Logikzellen-Multiplexer mit einem einer LUT nachgeschalteten (a) bzw. vorgeschalteten (b) Flipflop. (Aus Gründen der besseren Übersichtlichkeit wurden die Flipflop-Steuersignale und die zugehörige Logik weggelassen).*

Zur Realisierung eines nachgeschalteten Flipflops wird das Signal des Logikzelleneingangs *a* über den Eingangsmultiplexer zur LUT geschaltet und das LUT-Ausgangssignal über den zwischen der LUT und dem Flipflop vorhandene Multiplexer an das Flipflop geführt (Abbildung 4-1a). Zur Realisierung eines vorgeschalteten Flipflops wird das Ausgangssignal des Flipflops zurückgeführt und über den Eingangsmultiplexer in die LUT eingespeist. Das am Logikzelleneingang *a* anliegende Signal wird über den zwischen der LUT und dem Flipflop vorhandenen Multiplexer auf den Flipflop-Eingang geschaltet (Abbildung 4-1b). In beiden Fällen erzeugt die Logikzelle zwei Ausgangssignale, was eine effizientere Nutzung der Logikzellenressourcen als bei der klassischen Logikzellenarchitektur darstellt. Um eine Unterscheidung zu den Logikzellen zu ermöglichen, die entweder nur ein Flipflop oder nur eine LUT enthalten, werden Logikzellen, die beide Komponenten aufweisen, im Folgenden als *voll belegte* Logikzellen bezeichnet.

Das Packen der Flipflops und LUTs in die Logikzellen erfolgte bisher mit einer einfachen Mustererkennung [42]. Dabei wurde überprüft, ob einer LUT ein Flipflop nachgeschaltet ist, um anschließend beide in eine Logikzelle zu packen. Dieses Verfahren erweist sich vor allem bei registerintensiven Schaltungen als nicht optimal, da die Aneinanderreihungen von Flipflops und LUTs nicht erkannt und somit nicht immer effizient in die Logikzellen gepackt werden können. Das soll an einem Beispiel verdeutlicht werden.

In den beiden folgenden Abbildungen 4-2a und Abbildungen 4-2b sind jeweils drei LUTs und drei Flipflops auf dieselbe Weise miteinander durch Signale verbunden. Bei einer flächen- und verdrahtungsoptimalen Lösung entstehen insgesamt drei Logikzellen mit zwei zu verdrahtenden Inter-Logikzellenverbindungen (Abbildung 4-2a). Wird bei der herkömmlichen Vorgehensweise zuerst die mittlere LUT auf ein nachgeschaltetes Flipflop untersucht und dann mit diesem in eine Logikzelle gepackt, entsteht eine Lösung, bei der insgesamt vier Logikzellen benötigt werden und drei Inter-Logikzellenverbindungen zu verdrahten sind (Abbildung 4-2b). Im Rahmen dieser Arbeit wurde nun ein Algorithmus entwickelt, der eine effizientere Nutzung der Logikzellen als die herkömmliche Vorgehensweise erlaubt und der auf der Flexibilitätsbewertung der LUTs und Flipflops basiert.

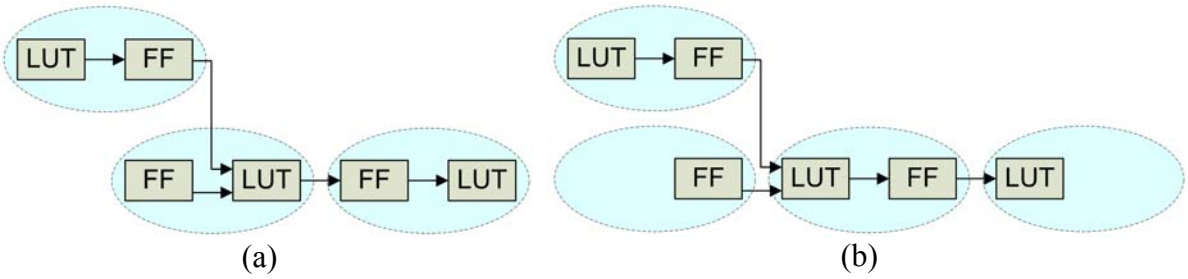


Abbildung 4-2: *Flächenoptimale Zuordnung der LUTs und FFs zu den Logikzellen (a), bzw. Zuordnung, die bei der herkömmlichen Vorgehensweise entsteht (b). (Aus Gründen der besseren Übersichtlichkeit sind nur die relevanten Signale eingezeichnet).*

4.2 Packen der Logikzellen nach dem Less-Flexible-First-Prinzip

Bei jeder Iteration des neuen Packverfahrens nach dem Less-Flexible-First-Prinzip (LFF-Prinzip) wird in einem ersten Schritt die Flexibilität *f* aller noch nicht gepackter LUTs und Flipflops bezüglich ihrer Packmöglichkeiten bestimmt. Die Flexibilität *f* gibt für jede LUT und jedes Flipflop einer Netzliste die Anzahl möglicher Partner für das Packen in eine Logikzelle an (Abbildung 4-3). Ist die Flexibilität $f = 0$, so muss das Flipflop oder die LUT einzeln in eine Logikzelle gepackt werden, weil ein passender Partner fehlt. Im zweiten Schritt einer jeden Iteration werden alle LUTs und Flipflops mit der Flexibilität $f = 1$ mit dem einen passenden Partner in die Logikzellen gepackt. In unserem Beispiel ergibt diese Vorgehensweise die bereits bekannte flächen- und verdrahtungsoptimale Lösung aus Abbildung 4-2a. Die Pack-Iterationen werden wiederholt, solange in der Netzliste freie Flipflops und LUTs mit der Flexibilität größer Null enthalten sind.

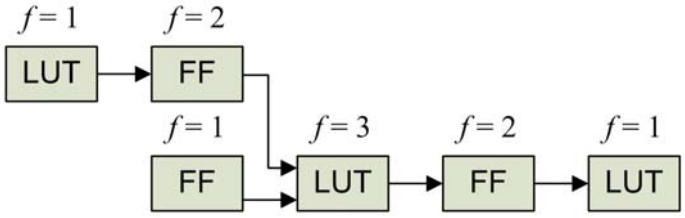


Abbildung 4-3: *Bestimmung der Flexibilität f.*

Haben alle ungepackten Flipflops und LUTs eine Flexibilität $f > 1$, d. h. mehr als einen passenden Partner, dann wird nur eine LUT oder ein Flipflop mit der geringsten Flexibilität zusammen mit einem der passenden Partner in eine Logikzelle gepackt und eine neue Iteration gestartet. In der Regel entstehen dabei genügend LUTs bzw. Flipflops mit der Flexibilität $f = 1$, so dass das Verfahren schnell konvergiert.

5 „Bottom-up“-Clusteringverfahren ARH-Pack

Das im Rahmen dieser Arbeit entwickelte neue „bottom-up“-Clusteringverfahren ARH-Pack (Area, Routability and Hierarchy driven Packer) entspricht in seiner Grundstruktur den bereits bekannten Greedy-Clusterungverfahren (vgl. Abbildung 2-21). Das heißt, zunächst wird für einen noch leeren Cluster eine Startlogikzelle bestimmt, deren Auswahl maßgeblich über das weitere Befüllen des Clusters entscheidet. Anschließend werden anhand einer Kostenfunktion immer wieder Logikzellen ausgesucht, die sequentiell dem Cluster hinzugefügt werden. Vor dem Hinzufügen einer Logikzelle zu einem Cluster muss überprüft werden, ob die Logikblockarchitektur des Ziel-FPGA über genügend Ressourcen verfügt, um die Logikzellen des damit entstehenden Clusters aufzunehmen und ihre Verdrahtung sicherzustellen. Ist das nicht der Fall, darf die ausgesuchte Logikzelle dem Cluster nicht hinzugefügt werden und die Suche wird fortgesetzt. – Die wesentlichen Optimierungsziele, die von dem neuen „bottom-up“-Clusteringverfahren ARH-Pack verfolgt werden, sind Verbesserung der Logiknutzung und Steigerung der Verdrahtbarkeit.

5.1 Clustering-Strategie

Beide von ARH-Pack verfolgten Optimierungsziele weisen gegenseitige Abhängigkeiten auf. So kann ein Logikblock nur dann vollständig mit Logikzellen befüllt werden, wenn noch genügend freie Verdrahtungsressourcen wie Logikblockanschlüsse und lokale Verdrahtungssegmente zur Verfügung stehen. Der Bedarf an beiden kann andererseits reduziert oder zumindest konstant gehalten werden, wenn einem Logikblock Logikzellen zugeordnet werden, die an möglichst viele gemeinsam genutzte Netze angeschlossen sind. Das soll an einem Beispiel erläutert werden (Abbildung 5-1). Die für das Packen in den Cluster *C* in Betracht gezogene Logikzelle *#4* ist mit dem Cluster *C* an drei gemeinsame Netze angeschlossen (*N1*, *N2* und *N4*). Das Netz *N3* ist bisher nicht an den Cluster *C* angeschlossen. Durch das Packen der Logikzelle *#4* in den Cluster *C* kann das gesamte Netz *N4* komplett innerhalb eines Logikblocks verdrahtet werden, wodurch der Anschluss *A4* frei wird, und für den Anschluss des Netzes *N3* an den Cluster *C* benutzt werden kann. Da die Netze *N1* und *N2* bereits an den Cluster *C* angeschlossen sind, sind hierfür keine zusätzlichen Anschlüsse eines Logikblocks erforderlich. Dadurch sinkt der Fanout dieser Netze auf der globalen Verdrahtungsebene, was die Verdrahtung des Netzes vereinfacht und somit die Verdrahtbarkeit erhöht.

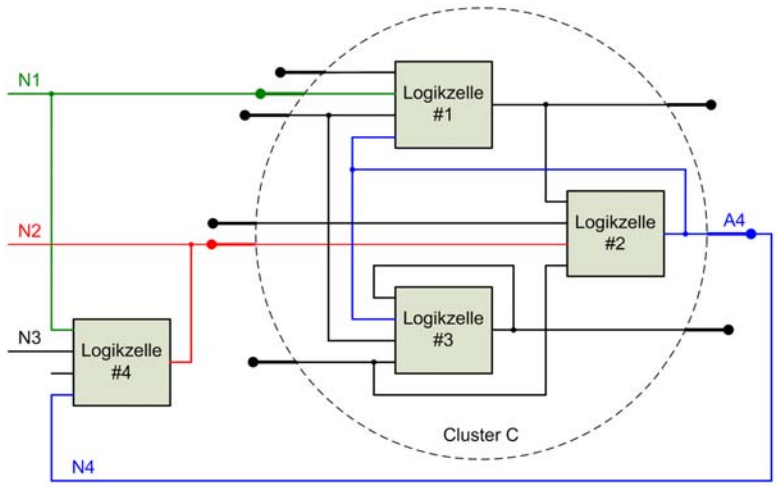


Abbildung 5-1: *Das Zuordnen der Logikzelle #4 zum Cluster C erfordert keine zusätzlichen Logikblockanschlüsse. Das Netz N4 kann komplett innerhalb eines Logikblocks verdrahtet werden; das Netz N3 wird über den freigewordenen Anschluss A4 angeschlossen.*

Wird die Nutzung der Logikblockanschlüsse auf die beschriebene Weise verringert, können die Cluster bei Designs mit hoher Konnektivität der Logikzellen bis an ihre Kapazitätsgrenzen gepackt werden. Bei solchen Designs ist die Anzahl der verfügbaren Logikblockanschlüsse einer der maßgeblich begrenzenden Faktoren, die eine maximale FPGA-Logikressourcennutzung verhindern. Die an einen Logikblock angeschlossenen Signale müssen über die Segmente der globalen Verdrahtungsarchitektur an das Logikblock herangeführt werden. Reduziert man die Anzahl der zur Verdrahtung erforderlichen Logikblockanschlüsse, werden im gleichen Maße weniger Verdrahtungssegmente der globalen Verdrahtungsarchitektur benötigt. Die ungenutzten Verdrahtungssegmente eines an die Logikblockanschlüsse angrenzenden Verdrahtungskanals können somit zu anderen Zwecken eingesetzt werden. Auf diese Weise kann die Verdrahtbarkeit eines Designs weiter gesteigert werden.

5.2 Auswahl einer Logikzelle zum Packen in den Logikblock

Soll einem Cluster eine weitere Logikzelle hinzugefügt werden, so muss die am besten geeignete Logikzelle ausgewählt werden, um die Logikblockressourcen möglichst optimal zu nutzen. Bei diesem Schritt des Verfahrens werden zunächst alle ungeclusterten Logikzellen der Netzliste *N* gemäß der Funktion F_{gain} gewichtet. Die Logikzelle mit der höchsten Gewichtung wird dann dem Cluster zugeordnet und aus der Netzliste entfernt.

$$F_{gain}(C, B) = \alpha \cdot |\, nets(C) \setminus nets(N \setminus C) \,| + \beta \cdot |\, nets(B) \cap nets(C) \,| + \gamma \cdot \left(\sum |H_{Bn} - H_C| + 1 \right)^{-1}$$

N	- Menge aller Logikzellen und Cluster der Netzliste
B	- Menge der Logikzellen des zu füllenden Clusters ($B \not\subset N$)
C	- ungeclusterte Logikzelle der Netzliste ($C \in N$)
nets(M)	- Netze der Menge M
$\mathbf{H_{Bn}}$	- Hierarchieebene der n-ten Logikzelle des Clusters B
$\mathbf{H_C}$	- Hierarchieebene der Logikzelle C
α, β, γ	- Gewichtungsfaktoren ($\alpha >> \beta >> \gamma$)

- Der **erste Summand** der Gewichtungsfunktion F_{gain} bestimmt die Anzahl der Netze, die beim Packen der Logikzelle *C* in den Cluster *B* absorbiert werden. In diesem Fall befinden sich alle Signalsenken und die Signalquelle im gleichen Cluster, und es werden nur logikblockinterne Verdrahtungsressourcen zur effizienten Verdrahtung verwendet. Außerdem wird dadurch ein Logikblockanschluss eingespart, da das Netz nicht mehr nach Außen hin zur globalen Verdrahtung geführt werden muss. Da in den meisten FPGA-Designs Netze mit einem geringem Fanout zu einen sehr hohen Anteil vertreten sind (Abbildung 5-2) [04], weist der erste Summand der Gewichtungsfunktion, der die Anzahl absorbierbarer Netze auswertet, auch die höchste Gewichtung α unter den betrachteten Faktoren auf. Praktisch bedeutet das, dass der Rest der Gewichtungsfunktion F_{gain} nur dann ausgewertet wird, wenn mehrere Logikzellen gefunden werden, die eine gleiche Anzahl absorbierbarer Netze aufweisen.

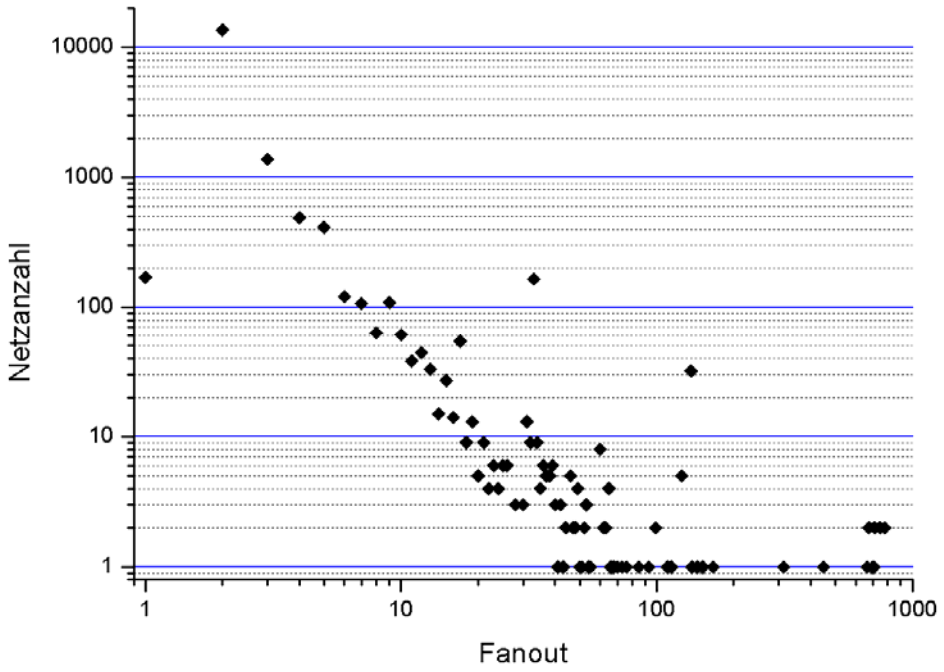


Abbildung 5-2: *Verteilung der Netze mit unterschiedlichem Fanout im „leon2"-Benchmarkdesign (11.504 LUTs und 6.211 Flipflops).*

- Der **zweite Summand** der Gewichtungsfunktion F_{gain} bestimmt die Anzahl der Netze, an die die betrachtete Logikzelle *C* und der Cluster *B* gemeinsam angeschlossen sind. Dieser Teil der Gewichtungsfunktion wird nur dann ausgewertet, wenn die Auswertung des ersten Summanden für mehr als eine Logikzelle die gleiche, höchste Gewichtung liefert. Die Logikzelle mit den meisten Verbindungen zum Cluster erhält durch den zweiten Summanden eine höhere Gewichtung. Somit wird sichergestellt, dass eine zu packende Logikzelle möglichst viele bereits vorhandene Anschlüsse und lokale Verdrahtungsressourcen innerhalb des Logikblocks wiederverwendet. Dieser Term der Funktion F_{gain} entspricht im Wesentlichen der Gewichtungsfunktion von VPack [43].
- Der **dritte Summand** der Gewichtungsfunktion F_{gain} bestimmt den Kehrwert der hierarchischen Distanz der Logikzelle *C* zu allen bereits in dem Cluster *B* enthaltenen Logikzellen. Alle Logikzellen einer Netzliste können anhand der von der Quartus-Synthese generierten Bezeichnungen zu den einzelnen Komponenteninstanzen der VHDL-Beschreibung und somit zu Hierarchieebenen zugeordnet werden. Zwischen zwei Logikzellen einer übergeordneten und einer untergeordneten Komponenteninstanz kann die hierarchische Distanz berechnet werden, welche die Anzahl der Hierarchieebenen zwischen den Komponenten angibt. So werden durch den dritten Summand Logikzellen bevorzugt, die aus der gleichen RTL-Instanz kommen, wie die meisten, bereits in dem Logikblock *B* enthaltene Logikzellen. Diesem Teil der Kostenfunktion liegt die Beobachtung zugrunde, dass ein Entwickler die HDL-Beschreibung so formuliert, dass die Signalanzahl einer Komponentenschnittstelle minimiert wird. Dadurch ist die Interkonnektivität der in der Komponente enthaltener Logik in der Regel sehr hoch, weshalb diese auch möglichst in den gleichen Cluster gepackt werden sollte.

5.3 Überprüfung der Clusterzusammensetzung

Bevor eine Logikzelle einem Cluster hinzugefügt werden kann, muss überprüft werden, ob die Logikzellen des Clusters nach diesem Hinzufügen von einem Logikblock der FPGA-Zielarchitektur aufgenommen und die notwendigen Verdrahtungen durchgeführt werden können. Die Logikblöcke der Altera-Cyclone- bzw. Stratix-Architekturen weisen eine beschränkte Anzahl der lokalen Logik- und Verdrahtungsressourcen auf (vgl. Abbildung 3-6). Das Überbenutzen dieser Ressourcen führt zu einem nicht platzierbaren und nicht verdrahtbarem Design. Folgende lokale und globale Logikblockressourcen müssen auf ihre Überbenutzung hin untersucht werden:

- Logikzellen (max. 10),
- Logikblockeingänge (max. 26 bei Cyclone- bzw. 30 bei Stratix-FPGAs),
- Rückkopplungen von den Logikzellenausgängen zu den LUT-Eingängen (max. 10),
- Segmente des Eingangsbusses (max. 30),
- Logikblockweite Flipflop-Steuersignale der dedizierten Multiplexerstruktur.

Die lokalen Inter-Logikblockverdrahtungssegmente können während des Clustering-Entwurfsschritts nicht berücksichtigt werden, da erst nach der Platzierung eine genaue Kenntnis darüber vorliegt, welche Logikzellen sich in den direkt links oder direkt rechts benachbarten Logikblöcken befinden. Das stellt eine prinzipbedingte Einschränkung dar, da man vom ungünstigsten Szenario ausgehen muss, dass diese Verdrahtungsressourcen überhaupt nicht genutzt werden können.

Die in PALLAs verwendete Funktion zur Überprüfung der Clusterzusammensetzung wird von allen in PALLAs implementierten Clustering-Verfahren gleichermaßen eingesetzt. Zusätzlich sammelt diese Funktion die Informationen darüber, wie oft und aus welchen Gründen das Hinzufügen einer Logikzelle zu einem Cluster abgelehnt wurde. Mit Hilfe dieser Informationen war es vor der Entwicklung von ARH-Pack möglich, festzustellen, welche der Logikblockressourcen beim Clustern mit den Referenzverfahren am häufigsten überbenutzt worden waren und somit kritischer als die anderen sind. Eine Logikzelle einem Cluster zuzuordnen scheitert bei großen Benchmarkschaltungen etwa zwei bis fünf Mal so häufig an den überbenutzten Flipflop-Steuersignalen wie an den überbenutzten Verdrahtungsressourcen. Da die Flipflop-Steuersignale vor allem von voll belegten Logikzellen (LUT+FF) verwendet werden, werden diese deshalb bei ARH-Pack als Startzellen bevorzugt. Diese Strategie soll beim Clustering ermöglichen, die aufgrund ihrer Flipflop-Steuersignale weniger flexiblen Logikzellen zu einem möglichst frühen Zeitpunkt den Clustern zuzuordnen.

5.4 Auswahl der Startzelle

Die Wahl der Startzelle hat einen entscheidenden Einfluss auf den weiteren Verlauf des Befüllens eines Clusters mit Logikzellen. Wird als Startzelle eine Logikzelle mit benutzten Flipflop-Steuersignalen genommen, so wird der Cluster mit hoher Wahrscheinlichkeit mit Logikzellen gefüllt, die die gleichen oder keine Flipflop-Steuersignale aufweisen. Dieses Verhalten ist auf die geringe mögliche Anzahl von logikblockweiten Flipflop-Steuersignalen zurückzuführen. Deshalb sind Logikzellen, die zahlreiche Flipflop-Steuersignale verwenden (das sind vor allem voll belegte Logikzellen) in ihrer Flexibilität, die Auswahl einer legalen Logikzellenkombination betreffend, eingeschränkt. Die Wahl einer voll belegten Logikzelle als Startzelle stellt jedoch sicher, dass diese Logikzellen mit ihrer geringen Flexibilität zu einem möglichst frühen Zeitpunkt des Clustering-Vorgangs konsumiert werden.

Um sicher zu stellen, dass beim Befüllen des Clusters genügend geeignete Logikzellen zur Verfügung stehen, wird bei ARH-Pack als Startzelle eine voll belegte Zelle ausgewählt, die eine Konnektivität zu möglichst vielen weiteren Logikzellen aufweist. Um den Konnektivitätsgrad zu bestimmen, werden nur die LUT-Eingänge, der LUT-Ausgang und der Flipflop-Ausgang in die Suche einbezogen. Die Flipflop-Steuersignale werden nicht einbezogen, da sie in der Regel ein hohes Fanout aufweisen, wodurch die Logikzellen mit möglichst vielen gleichen Flipflop-Steuersignalen bevorzugt würden. Sind keine voll belegten Zellen vorhanden, wird wie bei RPack [44] eine Zelle mit möglichst vielen Anschlüssen ausgewählt.

5.5 Behandlung der Carry-Chains und der RAM-Blöcke

Die von Quartus-Software durchgeführte Synthese erzeugt aus den erkannten arithmetischen Schaltungsstrukturen, z. B. Addierern, automatisch Carry-Chains. Hier müssen alle zu einer Carry-Chain gehörenden Logikzellen in den gleichen Cluster bzw. Logikblock gepackt werden. Übersteigt die Länge einer Carry-Chain die Kapazität eines Logikblocks, müssen die Logikzellen einer Carry-Chain der Reihenfolge nach auf mehrere Logikblöcke verteilt werden. Dies wird von ARH-Pack beachtet und deshalb wird die zunächst ungeclusterte LUT/FF-Netzliste nach vorhandenen Carry-Chains durchsucht. Die zu einer Carry-Chain gehörenden Logikzellen werden je nach Carry-Chain-Länge einem oder mehreren Logikblöcken zugeordnet. Die Überprüfung der Logikblockzusammensetzung ist beim Packen einer Carry-Chain in einen Logikblock nicht erforderlich, da die Vorgaben der FPGA-Zielarchitektur bereits vom Synthesewerkzeug eingehalten wurden. Kann ein Logikblock nicht vollständig befüllt werden, weil die Länge des zu diesem Logikblock zugeordneten Carry-Chain(-Abschnitts) kein Vielfaches von zehn ist, versucht ARH-Pack weitere Logikzellen in diesen Logikblock nach der im Abschnitt 5.2 beschriebenen Vorgehensweise zu packen. In diesem Fall findet die Überprüfung der Logikblockzusammensetzung statt.

Neben den LUTs und Flipflops sind in der von Altera-Quartus generierten Netzliste RAM-Grundzellen enthalten. Diese müssen aufgrund des hierarchischen Aufbaus der RAM-Blöcke auch geclustert werden. ARH-Pack ordnet die in einer Netzliste vorhandenen RAM-Grundzellen M4K-Blöcken zu, wie sie sowohl in den Cyclone- als auch in den Stratix-FPGAs vorhanden sind. Eine gültige Zuordnung liegt dann vor, wenn die RAM-Grundzellen innerhalb eins M4K-Blocks dieselben Adressleitungen und Datenleitungen des gleichen Datenworts verwenden.

6 Platzierverfahren MPCPlace

6.1 Überblick

Das im Rahmen von PALLAs entwickelte neue Clustering- und Platzierverfahren MPCPlace (Multipass Clustering and Placement) kombiniert die Ansätze und Vorteile bereits bekannter Algorithmen. Die Platzierung der Logikblöcke bzw. der Logikzellen mit MPCPlace erfolgt in zwei sequentiellen Durchläufen mit Hilfe von Simulated Annealing (Abbildung 6-1). Ausgangspunkt des ersten Durchlaufs ist die bereits mit einem in PALLAs implementierten Clustering-Verfahren geclusterte Netzliste einer Schaltung. Die Initialplatzierung der Logikblöcke beim ersten Platzierdurchlauf erfolgt zufällig, die Platzierpositionen der Ein- und Ausgänge werden übernommen, sofern diese vom Anwender in einer QSF-Datei festgelegt wurden. Danach wird die Temperatur der Initialplatzierung mit dem in [52] beschriebenen Verfahren bestimmt. Die Temperaturbestimmung beim ersten Durchlauf soll verhindern, dass das SA-Platzierverfahren mit einer viel zu hohen Temperatur gestartet und dadurch Rechenzeit verschwendet wird. Die anschließende SA-Platzierung der Logikblöcke erfolgt unter Kostenminimierung der Bounding-Box-Kostenfunktion C_I. Die Aufgabe des ersten Platzierdurchlaufs ist es, möglichst effizient eine gute Initialplatzierung für den zweiten Platzierdurchlauf zu erzeugen. Die Platzierung der Logikblöcke beim ersten Durchlauf erfolgt im Wesentlichen auf die bereits von VPR [68] bekannte Weise. Sie ist gültig und somit auch verdrahtbar.

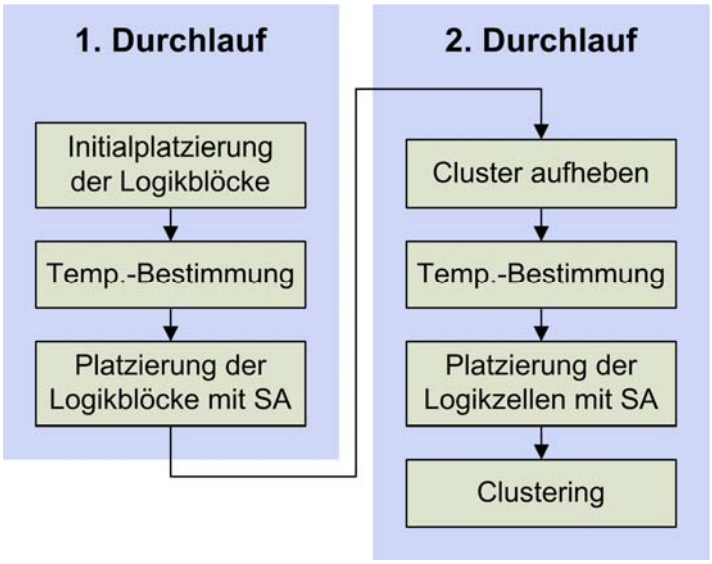


Abbildung 6-1: *Entwurfsfluss von MPCPlace.*

Beim zweiten Platzierdurchlauf wird die Verdrahtbarkeit der bereits aus dem ersten Durchlauf vorliegenden Platzierung weiter optimiert. Als erstes wird die Zuordnung der Logikzellen zu Logikblöcken unter Beibehaltung der beim ersten Durchlauf ermittelten Positionen auf dem FPGA aufgehoben. Die zusätzliche Beweglichkeit der Logikzellen im zweiten MPCPlace-Durchlauf bietet Potential für eine weitere Verbesserung der Platzierung. Nachdem die Logikzellenzuordnung aufgehoben wurde, erfolgt eine erneute Temperaturbestimmung der vorliegenden Platzierung, um die Zerstörung der Initialplatzierung durch eine zu hohe Temperaturwahl bei anschließender SA-Platzierung der Logikzellen zu verhindern. Aufgrund der bereits guten Initialplatzierung ist die beim zweiten Durchlauf bestimmte Starttemperatur niedriger als die Starttemperatur des ersten Durchlaufs. Die Bewegung der Logikzellen bei der zweiten Platzierung erfolgt deshalb vorwiegend nur lokal. Sie wird unter der Minimierung

der mit der neuen Bounding-Box-basierten Kostenfunktion C_{II} ermittelten Kosten durchgeführt. Nach der Platzierung erfolgt die endgültige Zuordnung der Logikzellen zu den Logikblöcken.

Die neue Bounding-Box-basierte Kostenfunktion C_{II} berücksichtigt die Nutzung der Intra-Logikblockverdrahtungsressourcen zum ersten Mal auch bei dem Entwurfsschritt der Platzierung. Bei den bisherigen Clustering- und Platzierverfahren für FPGAs war dies ein Optimierungskriterium nur beim Clustern der Logikzellen zu Logikblöcken. Die Nutzung der lokalen Inter-Logikblockverdrahtungsressourcen kann beim Clustering nicht berücksichtigt werden, da dazu die relative Position der Logikzellen zueinander bekannt sein muss, die jedoch erst beim Platzieren ermittelt wird. Beim Platzieren der Logikblöcke sind die Logikzellen fest den Logikblöcken zugeordnet, weshalb in diesen Entwurfsschritt eine Optimierung der Nutzung der lokalen Inter-Logikblockverdrahtungsressourcen nicht erfolgen kann. Beim zweiten Durchlauf des neuen Platzierverfahrens ist diese Einschränkung aufgehoben, so dass auch die Nutzungsoptimierung der lokalen Inter-Logikblockverdrahtungsressourcen durchgeführt werden kann.

6.2 Bestimmung der Starttemperatur

Die Bestimmung der Starttemperatur ist eine der kritischen Aufgaben bei der Platzierung mit Simulated Annealing. Startet die SA-Platzierung mit einer zufälligen Initialplatzierung, so wie bei MPCPlace im ersten Durchlauf, kann die Starttemperatur beliebig hoch gesetzt werden. Eine zu hoch gesetzte Starttemperatur bedeutet jedoch eine Verschwendung der Rechenzeit. Hier ist in den ersten Iterationen keine Minimierung der Kosten zu erwarten, da ja fast alle vorgeschlagenen Bewegungen akzeptiert werden. Bei einer Initialplatzierung von hoher Qualität, wie sie beim zweiten Durchlauf von MPCPlace vorliegt, wird bei zu hoch gesetzter Starttemperatur die Initialplatzierung zerstört. In diesem Fall wird wiederum Rechenzeit verschwendet, da erst nach mehreren SA-Iterationen eine Platzierung von gleicher Qualität wie die Initialplatzierung, erzeugt wird. Bei zu niedrig gesetzter Starttemperatur wird in beiden Fällen nur eine suboptimale Platzierlösung erreicht, da in der Anfangsphase der Platzierung die „uphill-climbing“-Strategie von Simulated Annealing sich nicht entfalten kann. Die Bestimmung der Starttemperatur wird deshalb bei beiden MPCPlace-Durchläufen nach der in [52] geschilderten Vorgehensweise durchgeführt, deren Grundlagen bereits im Abschnitt 2.3.7.1 erläutert wurden.

Die Annahme, dass sich eine zufällige Initialplatzierung in einem thermodynamischen Gleichgewicht befindet und somit auch die Voraussetzung für die Temperaturbestimmung mit dem verwendeten Verfahren sind beim ersten MPCPlace-Durchlauf nicht erfüllt. Es hat sich jedoch gezeigt, dass die Verwendung der mit dieser Vorgehensweise bestimmten Starttemperatur bei einer zufälligen Platzierung sinnvoller ist als die SA-Platzierung mit einer sehr hohen Temperatur zu starten. Beim zweiten Durchlauf von MPCPlace wird eine Initialplatzierung verwendet, die aus dem ersten Durchlauf vorliegt. Da die SA-Prozesse in beiden Durchläufen das gleiche Abkühlungsschema verwenden, liefert die Temperaturbestimmung im zweiten Durchlauf eine sehr genaue Schätzung der Starttemperatur.

6.3 SA-Abkühlungsschema

Wie bereits im Abschnitt 2.3.7.1 beschrieben, hat die Wahl eines geeigneten Abkühlungsschemas einen maßgeblichen Einfluss auf die Qualität der erzeugten Platzierungen und auf den dafür erforderlichen Aufwand. Zur Rekapitulation: Das Abkühlungsschema definiert die Anzahl der durchzuführenden Bewegungen, die Temperaturabsenkung und das Abbruchkriterium der Platzierung. Beide SA-

Platzierdurchläufe von MPCPlace verwenden das gleiche Abkühlungsschema, dessen Parameter von dem Anwender über die PALLAs-Benutzeroberfläche variiert werden können.

Die drei Bestandteile des Abkühlungsschemas wurden von dem bekannten SA-Platzierverfahren VPR [14][68] übernommen. Die Anzahl der auszuwertenden Logikblock- bzw. Logikzellenbewegungen pro Temperaturschritt beträgt $InnerNum \cdot (N_{Blocks})^{1,33}$, wobei N_{Blocks} die Anzahl der Logikblöcke bzw. der Logikzellen beim ersten bzw. zweiten Durchlauf ist. Mit dem Parameter *InnerNum* kann der Benutzer Einfluss auf die Anzahl der auszuwertenden Bewegungen nehmen. Als Standard ist der Parameter *InnerNum* auf zehn beim ersten und fünf beim zweiten Platzierdurchlauf voreingestellt. Durch diese Wahl des *InnerNum*-Parameters und weil die Logikzellenanzahl beim zweiten Durchlauf zehn Mal höher ist als die Logikblockanzahl beim ersten Durchlauf, ist die Anzahl der beim zweiten Platzierdurchlauf auszuwertender Bewegungen etwa zehn Mal so hoch wie beim ersten Durchlauf.

Für jede SA-Iteration wird ein neuer Temperaturwert nach der Formel $T_{neu} = \gamma \cdot T_{old}$ aus dem alten Temperaturwert bestimmt. Der Parameter γ hängt von der Akzeptanzrate α der ausgewerteten Logikblock- bzw. Logikzellenbewegungen der zuletzt beendeten Platzieriteration ab. Die genauen Werte von γ in Abhängigkeit von der Akzeptanzrate α sind in der Tabelle 6-1 zu finden und wurden experimentell bestimmt [14].

α	**γ**
$0{,}96 < \alpha$	0,5
$0{,}8 < \alpha \leq 0{,}96$	0,9
$0{,}15 < \alpha \leq 0{,}8$	0,95
$\alpha \leq 0{,}15$	0,8

Tabelle 6-1: *Abhängigkeit des Parameters γ von der Akzeptanzrate α.*

Bei hoher Temperatur wird fast jede vorgeschlagene Bewegung akzeptiert, was zu einer hohen Akzeptanzrate α führt; Kostenreduzierung und somit Verbesserung der Platzierung sind jedoch gering. Gegen Ende des Platziervorgangs ist die Qualität der Platzierung hoch und die Akzeptanzrate α gering; die Kostenreduzierung und somit die Verbesserung der Platzierung sind wiederum gering. Diese Zusammenhänge führten zu der Schlussfolgerung, dass die Temperatur in diesen beiden Fällen rascher abgesenkt werden sollte als bei mittleren Akzeptanzraten [14].

Die Abkühlung wird abgebrochen, wenn die Temperatur den Wert $\varepsilon \cdot C/N_{Nets}$ unterschritten hat, wobei C die Kosten der Platzierung und N_{Nets} die Gesamtzahl der Netze darstellt. Der Parameter ε kann vom Anwender angepasst werden, wobei sich die Werte zwischen 0,05 und 0,005 als sinnvoll erwiesen haben. Die geringeren Werte von ε erhöhen merklich die Platzierlaufzeiten, die Qualität der Platzierung steigt dabei nur geringfügig [14]. Bei MPCPlace ist der Parameter ε auf 0,05 voreingestellt.

6.4 Generierung der auszuwertenden Bewegungen

Bei niedrigen Temperaturen werden bei einer SA-Platzierung fast ausschließlich nur lokale Bewegungen der Logikblöcke bzw. Logikzellen akzeptiert. Die Bewegungen der Logikblöcke bzw. Logikzellen über längere Entfernung sind zumeist mit positiven Kosten verbunden, die bei fortgeschrittener SA-Platzierung immer seltener akzeptiert werden. Deshalb wurde in [55] eine Bereichsbeschränkung R_{lim} (engl.: range limiting) für die vorzuschlagenden Bewegungen eingeführt, die die Bewegungsgenerierung über einen festgelegten Abstand der Ziel- und Quelllogikblöcke hinaus verhindert. Dadurch können bereits während der

Bewegungsgenerierung und somit vor der rechenintensiven Kostenauswertung Logikblockbewegungen wirksam aussortiert werden, die mit hoher Wahrscheinlichkeit positive Kosten verursachen würden. Dieser Ansatz zur Reduzierung des Platzieraufwands wurde auch bei der Bewegungsgenerierung bei MPCPlace implementiert, berücksichtigt darüber hinaus die Besonderheiten der unterstützten FPGA-Zielarchitekturen.

Anders als die Implementierung in [55] oder bei VPR [14] ist die Bereichsbeschränkung R_{lim} bei MPCPlace nicht an den Bewegungsakzeptanzfaktor α gekoppelt. Sie wird bei MPCPlace vor der ersten SA-Iteration zunächst auf den Wert der diagonalen Chip-Ausdehnung des Ziel-FPGA gesetzt. Anschließend wird sie nach jeder SA-Iteration auf den anderthalbfachen Abstand zweier, während der letzten SA-Iteration über die längste Entfernung vertauschten Logikblöcke bzw. Logikzellen gesetzt.

Da eine zu platzierende Netzliste nicht nur Logikblöcke bzw. Logikzellen, sondern eventuell auch DSP- und RAM-Blöcke enthält, muss die Bewegungsgenerierung unterschiedliche (dedizierte) zu platzierende Funktionseinheiten behandeln können. Die Verwendung einer Bereichsbeschränkung bei der Bewegungsgenerierung für die DSP- und RAM-Blöcke kann jedoch zu schlechteren Platzierergebnissen führen. Die Ursache dafür liegt in den großen Abständen der senkrechten Spalten, in welchen die DSP- bzw. RAM-Blöcke untereinander angeordnet sind (vgl. Abbildung 3-2). Ist R_{lim} kleiner als der Abstand zweier benachbarten DSP- bzw. RAM-Spalten, so kann kein Austausch der DSP- bzw. RAM-Blöcke zwischen diesen stattfinden. Aus diesem Grund wird die Wirkung der Bereichsbeschränkung bei MPCPlace nur auf die Logikblöcke bzw. Logikzellen beschränkt.

Eine weitere Besonderheit der kommerziellen FPGAs sind Carry-Chains, die bei den bisherigen theoretischen FPGA-Architekturen und somit bei der Platzierung keine Beachtung gefunden haben. Eine sich über mehrere, direkt untereinander angeordnete Logikblöcken erstreckende Carry-Chain muss bei den Altera-FPGAs als eine Einheit betrachtet werden und kann nur als Ganzes verschoben werden. Wird zur Bewegung ein Logikblock oder eine Logikzelle bestimmt, die Bestandteil einer Carry-Chain ist, so wird das von MPCPlace erkannt und zunächst die Gesamtlänge der zu bewegenden Carry-Chain bestimmt. Danach wird per Zufallsgenerator eine benachbarte Logikblockspalte als Ziel der Bewegung vorgeschlagen und diese nach einem zusammenhängenden Logikblockbereich in der Länge der Carry-Chain durchsucht. Die Vertauschung des ausgesuchten Zielbereichs mit der Carry-Chain darf dabei keine eventuell an den Bereichsgrenzen des Zielbereichs vorhandene Carry-Chains auseinanderbrechen. Wenn ein solcher Zielbereich gefunden wurde, wird der Tausch aller Logikblöcke bzw. Logikzellen der Carry-Chain und des gesamten Zielbereichs vorgeschlagen und dessen Kosten ausgewertet. Auch im Falle der Carry-Chains ist die Bereichsbeschränkung R_{lim} nicht wirksam.

Die Kosten der vorgeschlagenen RAM- und Logikblock- bzw. Logikzellenvertauschungen werden bei MPCPlace mit Hilfe der Kostenfunktion C_I beim ersten Platzierdurchlauf bzw. mit Hilfe der Kostenfunktion C_{II} beim zweiten Platzierdurchlauf ermittelt. Entsprechend der im Abschnitt 2.3.7.1 beschriebenen Vorgehensweise wird aufgrund der ermittelten Kosten eine Entscheidung getroffen, ob eine vorgeschlagene Vertauschung durchgeführt wird.

Der Anwender hat die Möglichkeit, über die Vorgaben in der QSF-Datei die Positionen der Ein- und Ausgänge festzulegen, die während der Platzierung mit MPCPlace übernommen und beibehalten werden. Diese Festlegung hat keine Auswirkungen auf die Bewegungs-generierung oder die Kostenfunktionen C_I und C_{II}. Da jedoch mit den beiden Kostenfunktionen auch die Netze berücksichtigt werden, die an die Ein- und Ausgänge angeschlossen sind, führt die Positionsfestlegung der Ein- und Ausgänge zur Verschiebung der an diese Netze angeschlossenen Logikblöcke bzw. Logikzellen näher an die entsprechenden Ein- und Ausgangspins.

6.5 Kostenfunktion C_I für den ersten Platzierdurchlauf

Beim ersten Durchlauf der SA-Platzierung wird die Bounding-Box-Kostenfunktion C_I zur Bewertung der Logikblockplatzierung herangezogen. Durch die Minimierung der damit ermittelten Kosten wird der Verdrahtungsressourcenbedarf der global zu verdrahtenden Netze wirksam reduziert und somit die Verdrahtbarkeit eines Designs gesteigert. Die Kostenfunktion C_I summiert die Bounding-Box-Netzlängenabschätzungen aller global zu verdrahtenden Netze auf:

$$C_I = \sum_{i=1}^{Nnets} (h_{BBox,i} + l_{BBox,i})$$

N_{nets} - Anzahl aller Netze einer Netzliste
$h_{BBox,\,i}$ - Höhe der Bounding-Box des i-ten Netzes
$l_{BBox,\,i}$ - Länge der Bounding-Box des i-ten Netzes

Die Bounding-Box-Netzlängenabschätzung eines Netzes entspricht dem halben Umfang des kleinsten Rechtecks, welches alle Anschlüsse dieses Netzes einschließt (Abbildung 6-2). Sie ist hinreichend genau für Netze mit zwei bis vier Anschlüssen. Für Netze mit mehr Anschlüssen können Korrekturfaktoren verwendet werden [58]. Bei der betrachteten Kostenfunktion werden jedoch keine Korrekturfaktoren für die Netze mit mehr als vier Anschlüssen verwendet, da festgestellt wurde, dass ihre Verwendung die Verdrahtbarkeit nicht wesentlich beeinflusst. Das kann vor allem darauf zurückgeführt werden, dass ein Großteil der global zu verdrahtenden Netze bei den meisten FPGA-Designs Netze mit geringem Fanout sind (vgl. Abbildung 5-2) und weil die Segmentlängen der Altera-FPGAs grundsätzlich keine genaue Bounding-Box-Netzlängenabschätzungen zulassen.

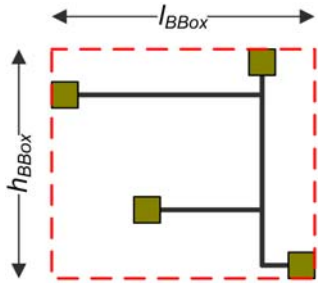
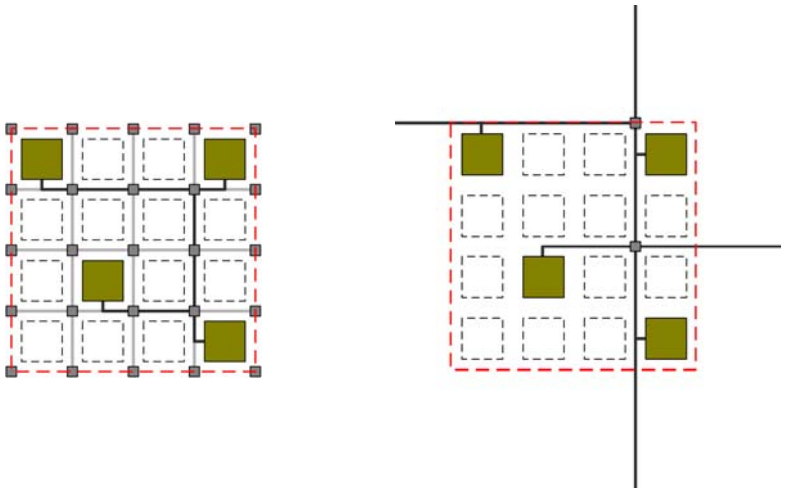


Abbildung 6-2: *Netzlängenabschätzung mittels einer Bounding-Box (BBox) am Beispiel eines ASIC. Die BBox-Schätzung liefert einen Netzlängenbedarf von 30 Längeneinheiten (LE); die tatsächliche Netzlänge ist 29 LE.*

6.6 Einschränkungen der Bounding-Box-Netzlängenabschätzung

Die Bounding-Box-basierte Netzlängenabschätzung ist sehr einfach zu berechnen und unter Berücksichtigung der Korrekturfaktoren hinreichend genau. Deshalb wird sie gerne als Bestandteil der Kostenfunktionen sowohl bei den ASIC- als auch den FPGA-Platzierverfahren eingesetzt [58][14]. Die Verwendung der BBox-Netzlängenabschätzung ist jedoch bei den FPGAs mit heterogenen Verdrahtungsressourcen und Segmentlängen größer Eins mit Einschränkungen verbunden. Das soll anhand von drei Beispielen erläutert werden, wozu ein zu verdrahtendes Netz mit vier Anschlüssen betrachtet wird, wobei die Anschlüsse bei allen drei Bespielen relativ zueinander die gleichen Positionen aufweisen (Abbildung 6-2 und Abbildung 6-3). Im ersten Beispiel wird die Netzlängenabschätzung bei einem ASIC

durchgeführt (Abbildung 6-2). Hier hat das Verdrahtungswerkzeug einen hohen Freiheitsgrad bei der Entscheidung, wie die Leitungen zu verlegen sind. Die tatsächliche Netzlänge (29 Längeneinheiten, LE) und die Bounding-Box-basierte Schätzung (30 LE) liegen eng beieinander.



(a) (b)

Abbildung 6-3: *BBox-Netzlängenabschätzung am Beispiel eines klassischen FPGA (a) und eines Altera FPGA (b). Die BBox-Schätzung liefert in beiden Fällen einen Netzlängenbedarf von 48 LE; die tatsächliche Netzlänge ist 46 LE (a) bzw. 100 LE (b).*

Im zweiten Beispiel (Abbildung 6-3a) wird die Netzlängenabschätzung bei einer klassischen FPGA-Architektur durchgeführt. Hier werden zur Verdrahtung Segmente mit der Länge eins und Switch-Boxen an den Kreuzungspunkten der Verdrahtungskanäle verwendet [14]. Die tatsächliche Netzlänge (46 LE) und die Bounding-Box-basierte Schätzung (48 LE) liegen wiederum eng beieinander.

Im dritten Beispiel, bei den Altera-Cyclone-FPGAs, stehen zur Verdrahtung auf der globalen Ebene nur C4/R4-Verdrahtungssegmente mit der Länge Vier zur Verfügung. Wird die BBox-Netzlängenabschätzung auf diese Architektur angewendet, so liefert die BBox-Schätzung den gleichen Netzlängenbedarf wie bei der klassischen FPGA-Architektur (Abbildung 6-3b). Die Gesamtlänge der zur Verdrahtung eingesetzter Segmente und somit die tatsächliche Netzlänge ist mit 100 LE aber etwa doppelt so hoch. Die Ursache für den Fehler bei der Schätzung liegt in den Verdrahtungssegmentabschnitten, die deutlich über die Grenzen der Bounding-Box hinausragen und somit nicht berücksichtigt werden (Abbildung 6-3b). Prinzipiell ließen sich weitere Logikblöcke an die aus der Bounding-Box herausragenden Segmentabschnitte anschließen, ohne dass sich der tatsächliche Verdrahtungsressourcenbedarf erhöhen würde.

Befinden sich die Signalquelle und alle Signalsenken eines Netzes in zwei horizontal direkt benachbarten Logikblöcken, so kann die Verdrahtung des Netzes allein unter der Zuhilfenahme der lokalen Inter- und Intra-Logikblockressourcen erfolgen. Der tatsächliche Netzlängenbedarf auf der globalen Verdrahtungsebene ist in diesem Fall gleich Null. Eine BBox-Netzlängenabschätzung liefert in diesem Fall einen Netzlängenbedarf, der dem halben Umfang zweier Logikblockkacheln entspricht. Um sowohl die Netzlängen aller global zu verdrahtenden Netze zu minimieren als auch die heterogenen Verdrahtungsressourcen der Altera-FPGAs zu berücksichtigen, wird im zweiten Platzierdurchlauf von MPCPlace die neue Kostenfunktion C_{II} verwendet.

6.7 Neue Kostenfunktion C_{II} für den zweiten Platzierdurchlauf

Die neue Bounding-Box-basierte Kostenfunktion C_{II} berücksichtigt die relativen Positionen der Signalsenken eines Netzes gegenüber der Signalquelle. Da die Signalquelle und die Signalsenken Logikzellen sind, müssen die Logikzellen beweglich sein, um eine Verringerung der mit C_{II} ermittelten Kosten zu erreichen. Deshalb wird die Kostenfunktion C_{II} erst im zweiten Durchlauf des MPCPlace-Platzierverfahrens zur Bewertung der aktuellen Logikzellenposition herangezogen. C_{II} stellt eine Bounding-Box-basierte Kostenfunktion dar, die mit Korrekturfaktoren versehen ist. In diesen sind die architekturbezogenen Bewertungen der relativen Positionen der Signalsenken zur Signalquellen sowie die Mehrfachnutzung der Verdrahtungsressourcen erfasst. Die Gesamtkosten für die Platzierung werden wie folgt berechnet:

$$C_{II} = \sum_{i=1}^{Nnets} ((h_{BBox,i} + l_{BBox,i}) / n \cdot \sum_{j=1}^{n} \alpha_j \beta_j)$$

N_{nets} - Anzahl aller Netze einer Netzliste
$h_{bbox,\,i}$ - Höhe der Bounding-Box des *i*-ten Netzes
$l_{bbox,\,i}$ - Länge der Bounding-Box des *i*-ten Netzes
n - Anzahl der Signalsenken des *i*-ten Netzes
α_j - architekturbezogener Korrekturfaktor der *j*-ten Signalsenke
β_j - Korrekturfaktor der Ressourcen-Mehrfachnutzung der *j*-ten Signalsenke (β_j = 0,9 bei Mehrfachnutzung, 1,0 sonst)

Der Korrekturfaktor α einer jeden Signalsenke wird in Abhängigkeit von den aktuellen Positionen der Signalsenke und der Signalquelle bestimmt. Die Werte der Korrekturfaktoren wurden experimentell anhand zahlreicher Platzierdurchläufe unterschiedlicher Benchmarkschaltungen ermittelt. In der nachfolgenden Tabelle 6-2 sind alle relevanten Fälle zusammengefasst.

Fall	Relative Position der Signalsenke (*dest*) in Bezug auf die Position der Signalquelle (*src*)	α
1	Im gleichen Logikblock, ($x_{dest} = x_{src}$, $y_{dest} = y_{src}$)	0,0
2	Im horizontal direkt angrenzenden Logikblock, ($\lvert x_{dest} - x_{src}\rvert=1$, $y_{dest} = y_{src}$)	0,3
3	Horizontal, innerhalb einer R4 Segmentlänge, ($5>\lvert x_{dest} - x_{src}\rvert>1$, $y_{dest} = y_{src}$)	0,7
4	Vertikal, innerhalb einer C4 Segmentlänge, ($x_{dest} = x_{src}$, $5>\lvert y_{dest} - y_{src}\rvert>1$)	0,8
	Sonstige Fälle	1,0

Tabelle 6-2: *Architekturabhängige Korrekturfaktoren der Kostenfunktion C_{II}.*

Die auf diese Weise bestimmten Kosten begünstigen Platzierungen, zu deren Verdrahtung mit hoher Wahrscheinlichkeit lokale Inter- und Intra-Logikblockverdrahtungsressourcen herangezogen werden. Anschaulich soll die Wirkung der architekturabhängigen Korrekturfaktoren anhand der Abbildung 6-4 erläutert werden. In der Mitte der Anordnung befindet sich der Logikblock mit der Signalquelle eines Netzes. Die architekturabhängigen Korrekturfaktoren berücksichtigen die relative Position der Signalsenke zu der Signalquelle und sind in der Abbildung 6-4 innerhalb der möglichen Logikblockpositionen angegeben. Die Platzierung der Signalsenken an farbig hervorgehoben Logikblockpositionen trägt zu Verringerung der Platzierkosten und somit zur höheren Verdrahtbarkeit eines Designs bei.

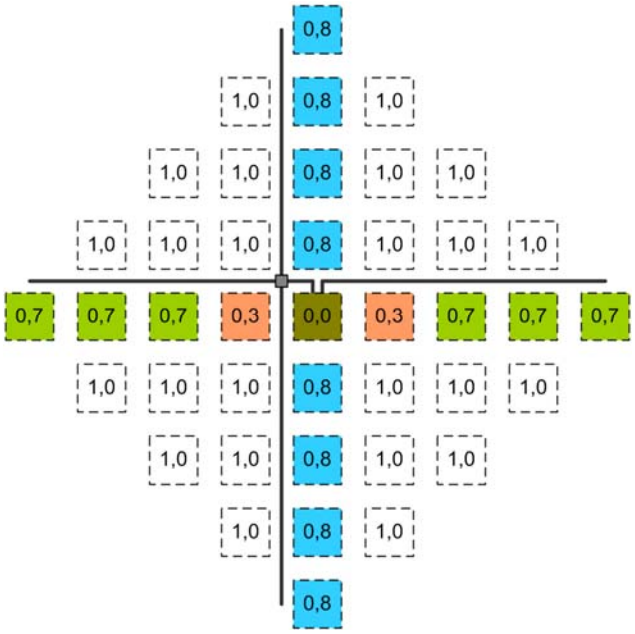


Abbildung 6-4: *Wirkungsbereich der architekturabhängigen Korrekturfaktoren. In der Mitte der Anordnung befindet sich die Signalquelle; farbig dargestellte Logikblockpositionen werden beim Platzieren bevorzugt.*

- Befinden sich eine Signalsenke und die Signalquelle innerhalb des gleichen Logikblocks (Fall 1 der Tabelle 6-2), werden zur Verdrahtung nur Intra-Logikblockverdrahtungsressourcen benutzt. Hierbei wird das Signal der Quelllogikzelle über den lokalen Rückkoppelbus des Logikblocks an den Eingang der Ziellogikzelle zurückgeführt. Das stellt die ressourcengünstigste Platziervariante dar, da hierbei der 30 Leitungen umfassende lokale Logikblock-Eingangsbus entlastet wird. Der architekturbezogene Korrekturfaktor ist in diesem Fall gleich Null.
- Befindet sich eine Signalsenke in dem zu der Signalquelle direkt links oder direkt rechts benachbarten Logikblock (Fall 2 der Tabelle 6-2), so können zur Verdrahtung des Netzes lokale Inter- *und* Intra-Logikblockverdrahtungsressourcen herangezogen werden. Auch in diesem Fall werden die globalen Verdrahtungsressourcen entlastet. Der Korrekturfaktor ist in diesem Fall mit 0,3 geringfügig höher als im Fall 1, da die Nutzung des lokalen Logikblock-Eingangsbusses steigt.
- Bei der Anordnung der Signalsenke innerhalb der Länge eines horizontalen R4-Verdrahtungssegments (Fall 3 der Tabelle 6-2) werden zur Verdrahtung des Signalquelle- und Signalsenkepaars globale Verdrahtungsressourcen in der Form eines R4-Segments und eines Logikblockanschlusses benötigt. Der architekturabhängige Korrekturfaktor ist in diesem Fall 0,7.
- Bei der vertikalen Anordnung der Signalsenke oberhalb oder unterhalb der Signalquelle innerhalb der Länge eines C4-Verdrahtungssegments (Fall 4 der Tabelle 6-2) werden zur Verdrahtung des Signals zusätzlich eine Switch-Box und ein C4-Segment benötigt (Abbildung 6-4). Aufgrund des gestiegenen Bedarfs an globalen Verdrahtungsressourcen ist hier der Korrekturfaktor im Vergleich zum Fall 3 mit 0,8 höher bemessen.

Der Wert des architekturbezogenen Korrekturfaktors α und somit auch die mit der Kostenfunktion C_{II} bestimmten Teilkosten steigen zusammen mit der Anzahl der zur Verbindung einer Signalquelle und einer Signalsenke erforderlichen Verdrahtungsressourcen. Die hierfür erforderlichen globalen und lokalen Verdrahtungsressourcen bei Verwendung der bevorzugten Platzierpositionen sind in Tabelle 6-3 zusammengefasst.

		lokale Verdrahtungsressourcen			**globale Verdrahtungsressourcen**				
Fall	α	Inter-Logikblock	lokaler Bus	Rückkoppel-Bus	LB-In	LB-Out	R4	C4	Switch-Box
1	0,0	0	0	1	0	0	0	0	0
2	0,3	1	1	0	0	0	0	0	0
3	0,7	0	1	0	1	1	1	0	0
4	0,8	0	1	0	1	1	1	1	1

Tabelle 6-3: *Die zur Verbindung einer Signalquelle und einer Signalsenke erforderlichen lokalen und globalen Verdrahtungsressourcen.*

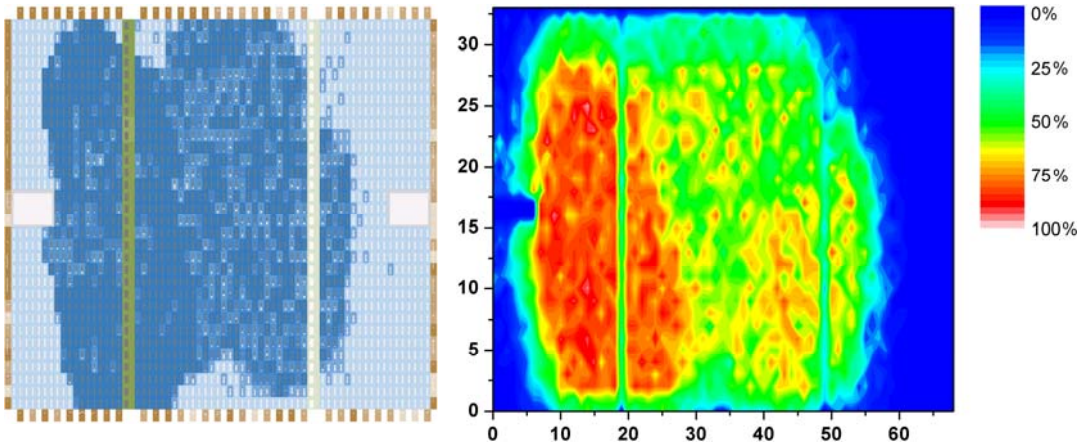
Beim Platzieren von mehr als einer Signalsenke eines Netzes in denselben Logikblock ist eine effiziente Mehrfachnutzung der in der Tabelle 6-3 aufgelisteten lokalen und globalen Verdrahtungsressourcen möglich. Sobald ein Netz an den lokalen Logikblock-Eingangsbus angeschlossen wurde, können weitere Logikzelleneingänge mit diesem Netz verbunden werden, ohne dass zur Verdrahtung zusätzliche globale oder lokale Ressourcen benötigt werden als die, die durch das Netz bereits in Nutzung sind. Diese Mehrfachnutzung der Verdrahtungsressourcen wird durch den Korrekturfaktor β der neuen Kostenfunktion C_{II} berücksichtigt. β wird der Wert von 0,9 zugewiesen, wenn beim Bewerten einer Signalsenkenposition erkannt wird, dass sich bereits eine Signalsenke des gleichen Netzes im selben Logikblock befindet. In sonstigen Fällen ist β gleich Eins und bewirkt keine Verringerung der Kosten. Auf diese Weise wird die Mehrfachnutzung der Verdrahtungsressourcen bevorzugt.

6.8 Überprüfung der Logikblockzusammensetzung

Da während des zweiten Platzierdurchlaufs die Zuordnung der Logikzellen zu den Logikblöcken fortwährend geändert wird, muss die Zusammensetzung der von einer Logikzellenbewegung betroffenen Logikblöcke auf ihre Gültigkeit überprüft werden. Zusätzlich zu den im Abschnitt 5.3 aufgezählten Ressourcen kann beim Platzieren der Logikzellen nun die Nutzung der Inter-Logikblockverdrahtungsressourcen berücksichtigt und überprüft werden, da die relative Position der Logikzellen zueinander bekannt ist. Durch die Anordnung eines Signalquelle- und Signalsenkepaares in den horizontal direkt benachbarten Logikblöcken können die globalen Verdrahtungsressourcen entlastet und anderweitig benutzt werden (vgl. Abschnitt 6.7). Wird eine Logikzelle bewegt, deren Ausgangssignal in den direkt links und direkt rechts benachbarten Logikblock über lokale Inter-Logikblockverdrahtungsressourcen gespeist wurde, muss das Ausgangssignal dieser Logikzelle nun über die globale Verdrahtung in die beiden Logikblöcke geleitet werden. Bei einer bereits vollständigen Nutzung der Logikblockeingänge der betroffenen Logikblöcke wird das Bewegen dieser Logikzelle zwei in ihrer Zusammensetzung ungültige Logikblöcke erzeugen, da deren Eingänge dadurch überbenutzt werden. Deshalb müssen beim Bewegen einer Logikzelle nicht nur die Zusammensetzungen des Ziel- und Quelllogikblocks, sondern auch die Zusammensetzungen der direkt links und direkt rechts neben dem Quelllogikblock angeordneten Logikblöcke überprüft werden.

6.9 Reduzierung der lokalen Kongestion globaler Verdrahtungsressourcen

Die Platzierverfahren VPR, SCPlace und MPCPlace waren zunächst nicht in der Lage, verdrahtbare Platzierungen des „*leon2*"-Designs zu generieren. Sie erzeugten Platzierungen, die zwar etwa gleich viele globale Verdrahtungsressourcen zur Verdrahtung des „*leon2*"-Designs wie die Quartus-II-Platzierung benötigen, es fand jedoch in einigen FPGA-Bereichen eine lokale Überbenutzung der Verdrahtungsressourcen statt (vgl. Tabelle 7-8). Diese verhinderte eine vollständige Verdrahtung der Platzierungen. Eine inhomogene Verteilung der benötigten C4/R4-Verdrahtungssegmente über die gesamte FPGA-Chipfläche kann besonders anschaulich graphisch dargestellt werden (Abbildung 6-5). Um nun die Verdrahtbarkeit der kritischen Designs sicher zu stellen, wurde bei MPCPlace eine zusätzliche Strategie implementiert, die die lokale Überbenutzung der Verdrahtungsressourcen wirksam verhindern kann und optional zuschaltbar ist.



(a) (b)

Abbildung 6-5: *Inhomogene Verteilung der Logik (a) und aller benötigten C4/R4-Segmente (b) über die gesamte FPGA-Chipfläche einer von MPCPlace generierten und nicht vollständig verdrahtbaren „leon2"-Platzierung.*

6.9.1 Modellierung der Nutzung globaler Verdrahtungsressourcen

Eine effektive Reduzierung der lokalen Überbenutzung globaler Verdrahtungsressourcen kann nur dann erfolgen, wenn das Wissen darüber vorliegt, in welchen Bereichen der Platzierung die entsprechenden Verdrahtungsressourcen des FPGA-Chips bei der späteren Verdrahtung nicht ausreichen, um eine vollständige Verdrahtung zu ermöglichen. Zur Modellierung des Verdrahtungsbedarfs wird bei MPCPlace eine an RISA [58] angelehnte Vorgehensweise verwendet, welche bereits im Abschnitt 2.3.7.2 vorgestellt wurde. Zur Rekapitulation: Bei RISA wurden die Oberfläche eines ASIC in NxM Bereiche aufgeteilt und die Kosten aller Bounding-Box-Abschätzungen zu den Bereichen aufaddiert, über die sich die entsprechenden Bounding-Boxen erstreckten. Diesem Modell nach erfordern die Bereiche mit dem höchsten Kostenanteil die meisten Verdrahtungsressourcen. Die Größe der einzelnen Bereiche verhält sich reziprok zu der Genauigkeit der Modellierung und dem dazu erforderlichen Aufwand [14].

Da bei einem hierarchischen FPGA bereits eine Strukturierung der FPGA-Oberfläche in Kacheln vorliegt (vgl. Abschnitt 2.2.2), bietet es sich an, diese natürliche Aufteilung anstatt einer benutzerdefinierten NxM-Matrix zu nutzen. Um den Berechnungsaufwand in Grenzen

zu halten, erfolgt die Modellierung der Verdrahtungsressourcen nicht nach jeder durchgeführten Logikzellenvertauschung, sondern nur einmal nach jeder abgeschlossenen SA-Iteration. Dabei werden für jede Kachel die Kongestionskosten ermittelt, die der Summe der mit der Kostenfunktion C_{II} ermittelten Kosten aller die entsprechende Kachel einschließenden Bounding-Boxen entsprechen (Abbildung 6-6). Weil die mit der Kostenfunktion C_{II} für ein Netz ermittelten Kosten umso geringer sind, je mehr Signalquelle- und Signalsenkenverbindungen unter Zuhilfenahme lokaler Verdrahtungsressourcen verdrahtet werden können, spiegelt ein solches Kongestionsmodell bei den Cyclone-FPGAs die voraussichtliche Nutzung der *globalen* Verdrahtungsressourcen in Form von C4/R4-Segmenten wider. Ein auf diese Weise für die „*leon2*"-Platzierung erstelltes Modell der Nutzung globaler Verdrahtungsressourcen ist in der Abbildung 6-7a zu sehen.

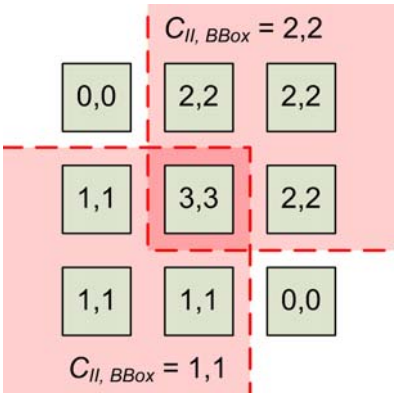


Abbildung 6-6: *Modellierung der Verdrahtungsressourcennutzung durch Aufsummierung der Kongestionskosten einzelner Bounding-Boxen.*

Liegt ein Modell zur Nutzung der globalen Verdrahtungsressourcen vor, so können die FPGA-Bereiche mit voraussichtlich hoher Nutzung der globalen Verdrahtungsressourcen identifiziert und die Platzierung so beeinflusst werden, dass diese Bereiche entlastet werden.

6.9.2 Beeinflussung der Platzierung zur Reduzierung der Kongestion

Die Reduzierung der lokalen Überbenutzung von Verdrahtungsressourcen kann über die Verringerung der Logikdichte in den überbeanspruchten Bereichen erfolgen [70]. Bei den hierarchischen FPGAs kann dazu die nutzbare Logikzellenanzahl in den betroffenen Logikblöcken eingeschränkt werden. Durch den durch die Rents-Regel postulierten Zusammenhang (siehe Abschnitt 2.3.7.2) verringert sich demnach der Bedarf an Logikblockanschlüssen und somit wird die Segmentnutzung der angrenzenden Verdrahtungskanäle reduziert.

Die Anzahl der benutzten Logikblockanschlüsse entspricht genau der Anzahl der zum Anschluss an die globalen Verdrahtungsressourcen erforderlichen C4/R4-Segmente in den angrenzenden Verdrahtungskanälen. Es erscheint deshalb naheliegender, bei den hierarchischen FPGAs nicht die Nutzung der Logikblockressourcen in Form von Logikzellen zu verringern und so die Nutzung der Logikblockanschlüsse, d. h. die Zahl der erforderlichen C4/R4-Segmente indirekt zu verringern, sondern die Anzahl nutzbarer Ein- und/oder Ausgänge der Logikblock direkt zu beeinflussen. Insbesondere bei den Logikzellen mit hoher Interkonnektivität untereinander bringt diese Vorgehensweise Vorteile. Diese können zu einer größeren Anzahl in einen Logikblock gepackt werden, wenn sich die Zahl der benötigten Logikblockeingänge nicht verändert und ein großer Teil der Verdrahtung unter Nutzung der lokalen Verdrahtungsressourcen erfolgen kann. Aus dieser Überlegung heraus wird bei MPCPlace in den Bereichen mit voraussichtlich hoher Nutzung der globalen

Verdrahtungsressourcen die Zahl der nutzbaren Logikblockeingänge eingeschränkt. Die Anzahl der nutzbaren Logikblockeingänge wird für jede Kachel einzeln anhand der Kongestionskosten der Kachel selbst und weiterer Kacheln in näherer Umgebung bestimmt. Das Einbeziehen benachbarter Kacheln in die Berechnung nutzbarer Logikblockeingänge soll den Transfer von unstetigen Bereichen des erzeugten Kongestionsmodells in die Vorgaben zur Nutzung der Logikblockeingänge verhindern. Die Festlegung der Anzahl nutzbarer Logikblockeingänge (*allowed_inputs*) erfolgt für jede Kachel nach jeder abgeschlossenen SA-Iteration direkt nach dem Erstellen des Kongestionsmodells, und zwar entsprechend folgender Vorschrift:

$$\left\{ \begin{array}{lll} allowed\_inputs_{tile} = max\_inputs & falls & \alpha_{tile} \le 1{,}0 \\ allowed\_inputs_{tile} = \dfrac{max\_inputs}{\alpha_{tile}} & falls & \alpha_{tile} > 1{,}0 \end{array} \right\}$$

wobei

$$\alpha_{tile} = \frac{\frac{1}{5}\left(C_{cong,tile} + C_{cong,top} + C_{cong,bottom} + C_{cong,left} + C_{cong,right}\right)}{\frac{1}{N}\sum_{i=0}^{N} C_{cong,i}}$$

$allowed\_inputs_{tile}$	- Nutzbare Logikblockeingänge der Kachel *tile*,
max_inputs	- Maximale Anzahl der Eingänge (26 bei Cyclone- bzw. 30 bei Stratix-FPGAs),
$C_{cong,x}$	- Kongestionskosten der direkt oberhalb (*x=top*), unterhalb (*x=bottom*), links (*x=left*) oder rechts (*x=right*) von der betrachteten Kachel (*x=tile*) angeordneten Kacheln,
N	- Anzahl aller Kacheln.

Die Vorgabe der nutzbaren Logikblockeingänge einzelner Kacheln wird von der Funktion zur Überprüfung der Logikblockzusammensetzung in die Entscheidung einbezogen, ob bei Simulated Annealing eine vorgeschlagene Bewegung bzw. Vertauschung der Logikzellen durchgeführt werden kann. Dadurch wird das Abwandern der Logikzellen aus den Bereichen mit beschränkter Anzahl nutzbarer Logikblockeingänge bzw. hoher Kongestion in die Bereiche mit weniger beschränkter Anzahl nutzbarer Logikblockeingänge bzw. niedriger Kongestion erzwungen (Abbildung 6-7b, c). In der Abbildung 6-7(c, d) ist deutlich eine homogenere Verteilung der Logik und des Verdrahtungsressourcenbedarfs einer „*leon2*"-Platzierung im Vergleich zu der Vorgehensweise ohne Modellierung und Reduzierung der lokalen Kongestion globaler Verdrahtungsressourcen (Abbildung 6-5) zu erkennen. Die homogenere Logikverteilung über den gesamten FPGA-Chip kann bereits vor der Platzierung mit MPCPlace erzwungen und die Aufgabe der Verringerung lokaler Kongestion während der Platzierung vereinfacht werden, indem die Logikblöcke beim Clustering-Entwurfsschritt nicht bis an ihre Kapazitätsgrenzen gepackt werden. Hier kann der Benutzer beim PALLAs-Werkzeug entsprechende Vorgaben an die Clustering-Algorithmen setzen.

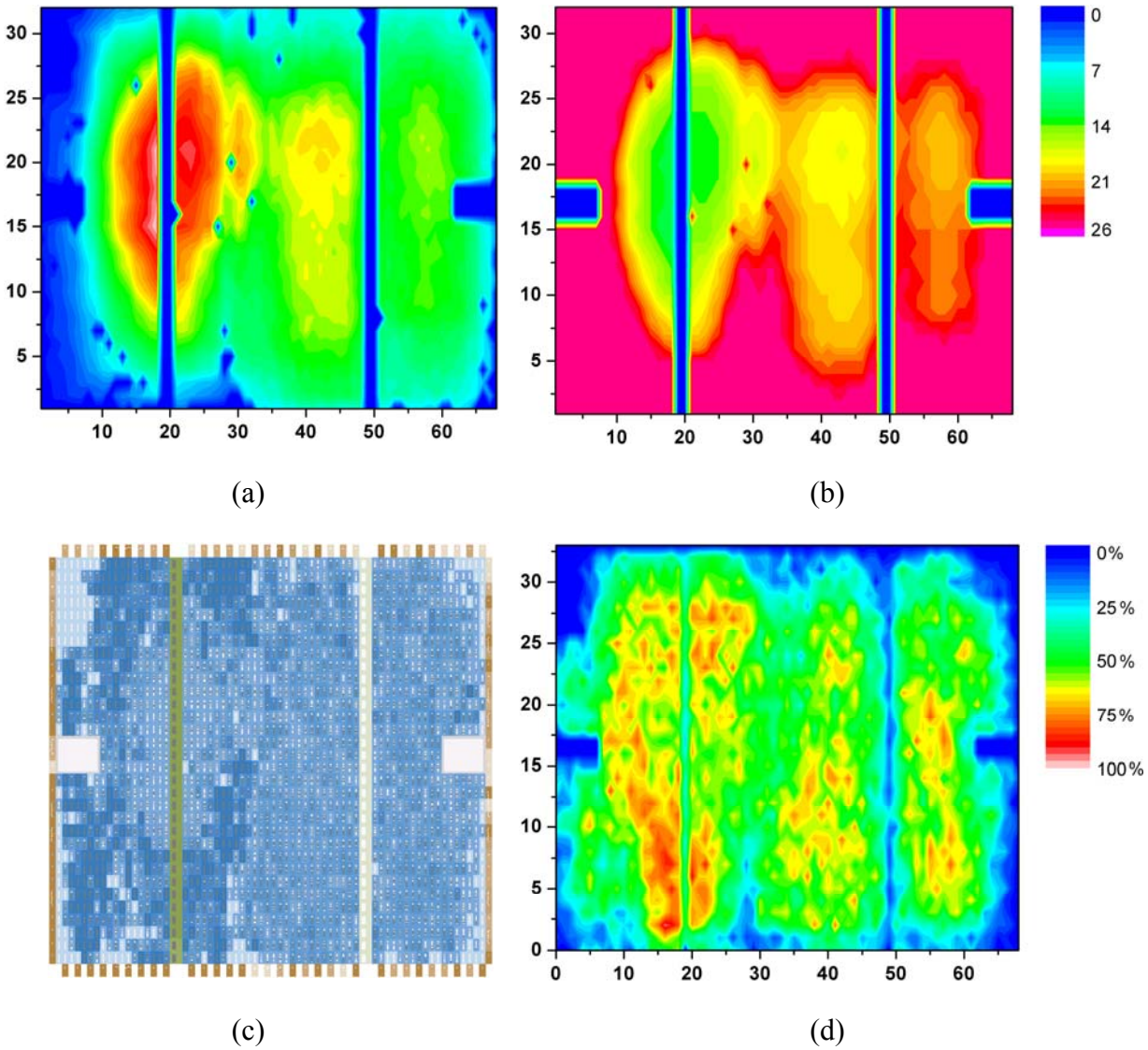


Abbildung 6-7: *Kongestionsmodell einer „leon2"-Platzierung (a), Einschränkung der nutzbaren Logikblock-Eingänge (b), daraufhin erzeugte Platzierung (c) und die resultierende Nutzung der globalen Verdrahtungsressourcen (d).*

7 Quantitative Untersuchungen

Im Folgenden werden die in PALLAs implementierten Verfahren einem Vergleich unterzogen. Zunächst werden im Abschnitt 7.1 die hierzu herangezogene Testmethodik und die verwendeten Designs vorgestellt. Die Wirksamkeit der Pack- und Clusteringverfahren wird in den Abschnitten 7.2 und 7.3 untersucht. Im Abschnitt 7.4 wird die Platzierung mit dem neuen Platzierverfahren MPCPlace zunächst ohne Modellierung und Reduzierung der lokalen Überbenutzung von Verdrahtungsressourcen (vgl. Abschnitt 6.9) und anschließend im Abschnitt 7.5 mit dieser optionalen Strategie durchgeführt. Zum Schluss werden im Abschnitt 7.6 die Laufzeiten der Platzierverfahren einander gegenübergestellt und im Abschnitt 7.7 die Auswirkung der verbesserten Verdrahtbarkeit auf den Stromverbrauch untersucht.

7.1 Testmethodik und verwendete Designs

Um die in PALLAs implementierten Algorithmen testen und vergleichen zu können, wurden zwölf von Altera vorgeschlagene „Open Cores"-Designs verwendet [123]. Die Designs wurden so ausgewählt, dass eine möglichst große Bandbreite praktischer Anwendungen abgedeckt wird [83]. Die Komplexität der ausgewählten Schaltungen liegt im Bereich von etwa 1.000 bis 14.000 Logikzellen (Tabelle 7-1). Die in den Veröffentlichungen häufig verwendeten MCNC-Benchmarkschaltungen [82] konnten nicht verwendet werden, da darin enthaltene Designs nur in Form von Netzlisten zur Verfügung stehen. Eine Synthese und somit eine auf die FPGA-Zielarchitektur optimierte Technologieabbildung dieser Schaltungen war aufgrund fehlender HDL-Quelltexte nicht möglich.

Design	#LUTs	#FFs	#LUTs in Carry-Chains	#M4K-Blöcke	#Logikblöcke im Ziel-FPGA
mux64_16bit	672	1.146	-	-	4.000
oc_ata_ocidec3	863	749	63	-	4.000
oc_ata_vhd_3	876	749	63	-	4.000
oc_cordic_p2r	1.163	719	910	-	4.000
oc_des_des3perf	11.742	5.850	11	4	20.060
oc_des_perf_opt	3.872	1.976	-	-	5.980
leon2	11.504	6.211	159	32	20.060
mc8051	3.040	1.741	120	-	4.000
oc_pavr	3.819	1.182	334	8	4.000
oc_simple_fm	1.106	226	-	-	4.000
oc_video_dct	10.157	3.549	5.733	9	12.060
oc_video_jpeg	9.510	3.972	5.408	8	12.060

Tabelle 7-1: *Logikressourcenbedarf der zum Testen der Platzieralgorithmen herangezogenen Designs und die Kapazitäten (5. Spalte) der einzelnen Ziel-FPGAs.*

Bisher konnten sich die Hersteller von FPGAs und die in den FPGA-Entwurf involvierten Softwarehersteller nicht auf eine einheitliche Benchmark-Methodik einigen [85]. Um einen sorgfältigen und reproduzierbaren Vergleich der implementierten Algorithmen sicherzustellen, wurde in dieser Arbeit eine an die Empfehlungen in [84] bzw. [85] angelehnte Benchmarking-Methodik angewendet. Alle verwendeten Benchmarkschaltungen wurden dazu zunächst mit Standardeinstellungen der Quartus-Software synthetisiert und die Verilog-Netzliste wurde an PALLAs übergeben. Als Ziel-FPGAs wurden Bausteine der Altera-Cyclone-Familie ausgewählt, die, bezogen auf die Logikressourcen, die einzelnen Designs gerade noch aufnehmen konnten. Diese Vorgehensweise entspricht dem Verhalten eines

FPGA-Anwenders, der versucht, die Kosten eines FPGA-basierten Designs niedrig zu halten. Nach dem Einlesen der Netzliste in PALLAs wurden die Entwurfsschritte zum Packen und Clustern der Logikzellen und die Platzierung der Logikblöcke bzw. Logikzellen durchgeführt. Die während der Platzierung ermittelten absoluten Positionen der Logikressourcen wurden über eine QSF-Datei wieder an Quartus übergeben. Die Ergebnisse des Packens und Clusterns der Logikzellen wurden von PALLAs ausgegeben. Nach der dann erfolgten Verdrahtung konnten mit den Analysewerkzeugen der Altera-Quartus-Software die Nutzung der Verdrahtungsressourcen und die maximale Taktfrequenz bestimmt werden [84].

7.2 Packen der LUTs und Flipflops in die Logikzellen

Um die Effektivität des Packens der Logikzellen nach dem LFF-Prinzip gegenüber der herkömmlichen Vorgehensweise möglichst unabhängig von der vorliegenden Altera-Logikzellenarchitektur vergleichen zu können, wurde ein drittes Packverfahren implementiert. Dieses wendet das bereits bekannte Prinzip der Mustererkennung [42] zum Finden der sowohl nach- als auch vorgeschalteten Flipflops. In der nachfolgenden Tabelle 7-2 sind die Ergebnisse der drei Verfahren zum Packen der Flipflops und der LUTs in die Logikzellen zusammengefasst.

Für die Tabellen in diesem und nachfolgenden Abschnitten gilt, dass die jeweils besten Ergebnisse dunkelgrau und die zweitbesten hellgrau hervorgehoben sind. Wo es sinnvoll erscheint, sind zusätzlich in der letzten Tabellenspalte die mit den neu entwickelten Verfahren (LFF-Prinzip, ARH-Pack, MPCPlace) erreichten Resultate mit den jeweils besten Werten der Referenzverfahren verglichen.

Design	#LC-std1	#LC-std2	#LC-LFF	Verbesserung
mux64_16bit	1.702	1.701	1.184	30,4 %
oc_ata_ocidec3	1.248	1.192	999	16,2 %
oc_ata_vhd_3	1.291	1.348	1.050	22,1 %
oc_cordic_p2r	1.184	1.183	1.183	0 %
oc_des_des3perf	15.990	13.734	13.734	0 %
oc_des_perf_opt	5.336	4.564	4.563	0,02 %
leon2	16.465	12.140	11.587	4,5 %
mc8051	4.254	4.147	3.595	13,3 %
oc_pavr	4.142	3.843	3.827	0,4 %
oc_simple_fm	1.286	1.285	1.109	13,7 %
oc_video_dct	10.192	10.186	10.186	0 %
oc_video_jpeg	9.962	9.859	9.859	0 %

Tabelle 7-2: *Anzahl der Logikzellen nach dem Packen der LUTs und Flipflops mit der herkömmlichen Vorgehensweise unter Berücksichtigung der nachgeschalteten Flipflops (std1) und der sowohl vor- als auch nachgeschalteten Flipflops (std2).*

Die zweite Spalte der Tabelle 7-2 fasst die Ergebnisse des Standardverfahrens nach [42] zusammen, bei dem nur die den LUTs nachgeschalteten Flipflops gesucht und in die Logikzellen gepackt werden. Werden mit der gleichen Vorgehensweise auch die vorgeschalteten Flipflops gesucht und in die Logikzellen gepackt, werden deutlich weniger Logikzellen benötigt, wie die dritte Spalte der Tabelle 7-2 belegt. Da dabei der Algorithmus im Kern unverändert ist, sind die besseren Ergebnisse dieser Spalte allein auf die effizientere Logikzellenarchitektur zurückzuführen. Das neue Packverfahren nach dem LFF-Prinzip, (vierte Spalte), benötigt teilweise bis zu 30% weniger Logikzellen verglichen mit der herkömmlichen Vorgehensweise des Erkennens der vor- und nachgeschalteten Flipflops.

Interessanterweise führt das an die Altera-Logikzellenarchitektur angepasste Standardverfahren mit dem Erkennen der sowohl vor- als auch nachgeschalteten Flipflops (dritte Spalte) bei dem „*oc_ata_vhd_3*“-Design zu schlechteren Ergebnissen als das ursprüngliche Packverfahren (zweite Spalte). Der Grund dafür ist, dass das angepasste Standardverfahren die Packmöglichkeiten bzw. die Flexibilität der noch ungepackten LUTs und Flipflops der Netzliste beim Packen der Logikzellen reduziert, weshalb diese bei späteren Iterationen nur noch einzeln in die Logikzellen gepackt werden können. Das neue Packverfahren nach dem LFF-Prinzip wertet hingegen die Flexibilität der LUTs und Flipflops explizit aus, weshalb eine Verschlechterung der Ergebnisse im Vergleich zu den beiden anderen Verfahren nicht möglich ist.

7.3 Clustering der Logikzellen

Die nachfolgenden Tabellen 7-3, 7-4 und 7-5 fassen die Clustering-Ergebnisse der drei bekannten Referenzverfahren VPack [43], RPack [44], iRAC [38] und des neuen Algorithmus ARH-Pack zusammen. Um die Vergleichbarkeit der Ergebnisse zu gewährleisten, wurden die LUTs und Flipflops bei allen vier Verfahren nach dem LFF-Prinzip in die Logikzellen gepackt.

VPack	RPack	iRAC	ARH-Pack	Verbesserung
6.881	6.900	7.011	6.862	0,27 %

Tabelle 7-3: *Gesamtanzahl der Logikblöcke aller Benchmarkschaltungen nach dem Clustern (aufgrund der geringen Unterschiede keine Auflistung der einzelnen Teilergebnisse).*

Die Gesamtzahl der Logikblöcke nach dem Clustern (Tabelle 7-3) verdeutlicht, dass die zur Verfügung stehenden Logikblockverdrahtungsressourcen der Altera-Cyclone-Logikblock-architektur keine wesentliche Einschränkung für eine effektive Logikblocknutzung darstellen. Die vier Clustering-Verfahren erreichen eine nahezu vollständige Belegung der Logikblöcke. Der geringe Unterschied in den Ergebnissen zwischen VPack [43] , RPack [44] und ARH-Pack ist darauf zurückzuführen, dass alle drei Verfahren als Startzellen voll belegte Logikzellen bevorzugen. Die Auswahl der Startzellen bei VPack und RPack erfolgt durch eine Kostenfunktion, die die Logikzellen mit den meisten Ein- und Ausgängen höher bewertet und somit indirekt voll belegte Logikzellen bevorzugt. Bei ARH-Pack werden die voll belegten Logikzellen von vornherein als Startzellen ausgewählt. Diese bei der Wahl eines passenden Clusters weniger flexiblen Logikzellen werden bei den drei Verfahren somit bereits in einem frühen Stadium des Clustering-Verfahrens konsumiert, was eine höhere Logikblockausnutzung ermöglicht. Der iRAC-Algorithmus [38] bevorzugt bei der Startzellenwahl Logikzellen, die über zahlreiche Netze mit möglichst geringem Fanout verfügen. Hier besteht die Gefahr, dass ein Logikblock nicht voll gepackt werden kann, da auf diese Weise eher kleine Logikzellenverbunde ausgewählt werden, die untereinander hohe Interkonnektivität aufweisen. Weiterhin können zum Ende des Clustering-Vorgangs voll belegte Logikzellen ungeclustert bleiben, deren Zuordnung zu Clustern dann immer schwieriger wird. Dieses Verhalten führt zu der schlechtesten Logikausnutzung mit einer relativ geringen Anzahl an global zu verdrahtenden Netze.

Unter den drei Referenzverfahren ist die Gesamtzahl der global zu verdrahtenden Netze nach dem Clustern (Tabelle 7-4) erwartungsgemäß bei iRAC [38] am geringsten, da dieses Verfahren gezielt versucht, möglichst viele Netze komplett in die einzelnen Logikblöcke zu packen. Von RPack [44] und VPack [43] werden bevorzugt Logikzellen mit vielen Eingängen zum Packen in die Cluster ausgewählt, was sich in der geringsten Anzahl der Logikblock-anschlüsse unter den Referenzverfahren äußert. Das neue Clustering-Verfahren ARH-Pack kombiniert die unterschiedlichen Ideen von VPack, RPack und iRAC. So wird die Anzahl der

global zu verdrahtenden Netze bei ARH-Pack nach den gleichen Prinzipien wie bei dem iRAC-Verfahren minimiert (vgl. Abschnitt 5.2), welches unter den drei Referenzverfahren in dieser Hinsicht die besten Ergebnisse aufweist. Diese Strategie wird durch den ersten Summand der Gewichtungsfunktion F_{gain} berücksichtigt. Andererseits ist die Anzahl der Logikblockanschlüsse nach dem Clustern unter den drei Referenzverfahren bei VPack und RPack am geringsten. Deren Strategie zur Minimierung der Logikblockanschlüsse wird durch den zweiten Term der Gewichtungsfunktion F_{gain} implementiert.

Design	VPack	RPack	iRAC	ARH-Pack	Verbesserung
mux64_16bit	1.295	1.251	1.053	893	28,6 %
oc_ata_ocidec3	1.019	1.024	947	898	5,2 %
oc_ata_vhd_3	1.023	1.010	937	861	8,1 %
oc_cordic_p2r	1.174	1.172	1.160	1.173	-1,1 %
oc_des_des3perf	11.212	11.234	8.923	7.572	15,1 %
oc_des_perf_opt	3.378	3.292	2.471	2.214	10,4 %
leon2	14.436	12.945	12.328	11.601	5,9 %
mc8051	3.008	2.947	2.507	2.298	8,3 %
oc_pavr	3.447	3.202	2.502	2.369	5,3 %
oc_simple_fm	949	750	712	591	17,6 %
oc_video_dct	8.715	8.656	8.645	8.511	1,6 %
oc_video_jpeg	8.276	8.234	8.202	8.054	1,8 %

Tabelle 7-4: *Anzahl der global zu verdrahtenden Netze nach dem Clustern.*

Design	VPack	RPack	iRAC	ARH-Pack	Verbesserung
mux64_16bit	3.092	3.107	2.896	2.640	8,8 %
oc_ata_ocidec3	2.879	3.015	3.087	2.823	2,0 %
oc_ata_vhd_3	2.899	2.957	3.121	2.820	2,7 %
oc_cordic_p2r	3.112	3.112	3.083	3.110	-0,9 %
oc_des_des3perf	29.435	31.188	30.551	26.565	9,8 %
oc_des_perf_opt	8.782	8.841	8.726	7.869	9,8 %
leon2	42.114	39.354	43.688	39.957	-1,5 %
mc8051	9.248	9.352	10.206	8.858	4,2 %
oc_pavr	10.241	9.871	10.621	9.811	0,6 %
oc_simple_fm	2.386	2.021	2.300	1.890	6,5 %
oc_video_dct	22.964	22.930	23.518	22.647	1,2 %
oc_video_jpeg	22.298	22.152	22.769	21.902	1,1 %

Tabelle 7-5: *Anzahl der Logikblockanschlüsse nach dem Clustern.*

Die Ergebnisse belegen, dass mit ARH-Pack Verbesserungen erreicht wurden, sowohl bei der Ausnutzung der Logik- als auch der Verdrahtungsressourcen. Die Ergebnisse fallen noch deutlicher zu Gunsten von ARH-Pack aus, wenn das Packen der nachgeschalteten LUTs und Flipflops in die Logikzellen bei den drei Referenzverfahren nicht nach dem neu entwickelten LFF-Prinzip, sondern auf die herkömmliche Weise erfolgt. Die Clustering-Gewichtungsfunktion F_{gain} berücksichtigt hier auf eine neuartige Weise mehrere Faktoren, die zwar bereits aus früheren Arbeiten bekannt sind, jedoch nicht in Kombination optimiert wurden. Weiterhin werden von ARH-Pack die Merkmale der zugrunde gelegten Logikblock-Architektur gezielt unterstützt und ausgenutzt. Auf diese Weise können insgesamt bessere Ergebnisse erreicht werden als mit einem der drei Referenzverfahren.

7.4 Platzierung und Verdrahtung

Die von verschiedenen Verfahren erzeugten Platzierungen wurden in die Altera-Quartus-Software importiert und anschließend verdrahtet, um deren Verdrahtbarkeit zu überprüfen. Die Altera-Quartus-Software liefert genaue Angaben über die Nutzung der einzelnen Verdrahtungsressourcenarten der fertig verdrahteten Designs [84]. Diese Angaben wurden zur Bewertung der einzelnen Platzierungen herangezogen. Neben den bekannten Platzierverfahren VPR und SCPlace und dem neu entwickelten Platzierverfahren MPCPlace wurde zum Vergleich Quartus als kommerzielle Software herangezogen. Die Nutzung der globalen Verdrahtungsressourcen in Form von C4/R4-Segmenten ist in der Tabelle 7-6 zusammengefasst. Die Angaben zur Nutzung der lokalen Inter-Logikblocksegmente zeigt Tabelle 7-7. Die Angaben zur höchsten lokalen Nutzung aller Verdrahtungsressourcen fasst Tabelle 7-8 zusammen.

Design	VPack + VPR	ARH-Pack + VPR	SCPlace	MPCPlace	Quartus	Ver-besserung
mux64_16bit	3.218	2.488	1.719	1.641	2.502	5 %
oc_ata_ocidec3	2.368	2.343	1.961	1.874	2.541	4 %
oc_ata_vhd_3	2.266	2.283	1.830	1.888	2.510	-3 %
oc_cordic_p2r	1.752	1.831	1.788	1.492	1.854	15 %
oc_des_des3perf	34.469	29.433	38.158	24.206	29.983	18 %
oc_des_perf_opt	10.075	8.649	7.993	6.287	8.097	21 %
leon2	48.755*	44.067*	45.942	38.442*	37.822	-2 %
mc8051	7.771	7.120	6.609	5.550	6.786	16 %
oc_pavr	9.497	9.805	9.659	7.764	8.047	3 %
oc_simple_fm	1.391	1.068	936	819	1.165	13 %
oc_video_dct	17.732	17.527	29.264	17.101	18.496	2 %
oc_video_jpeg	20.761	20.016	30.540	17.975	17.345	-4 %

Tabelle 7-6: *Anzahl der zur Verdrahtung erforderlichen C4/R4-Segmente (mit ‚*' sind nicht vollständig verdrahtbare Platzierungen markiert).*

Design	VPack + VPR	ARH-Pack + VPR	SCPlace	MPCPlace	Quartus
mux64_16bit	110	167	222	427	111
oc_ata_ocidec3	132	131	271	369	116
oc_ata_vhd_3	181	116	272	384	109
oc_cordic_p2r	255	261	242	432	204
oc_des_des3perf	1.299	1.100	1.506	3.802	1.104
oc_des_perf_opt	457	436	576	1.149	444
leon2	1.372*	1.215*	1.839*	2.528*	1.627
mc8051	469	367	637	1.172	373
oc_pavr	451	287	566	1.051	264
oc_simple_fm	251	191	250	400	187
oc_video_dct	1.680	1.528	1.134	2.769	1.637
oc_video_jpeg	1.482	1.506	986	2.544	1.369

Tabelle 7-7: *Anzahl der bei der Verdrahtung benutzten Inter-Logikblock-Verdrahtungssegmente (mit ‚*' sind nicht vollständig verdrahtbare Platzierungen markiert).*

Die mit einem Stern ‚*' markierten Prozentangaben bei dem „*leon2*"-Design weisen auf die nicht vollständig verdrahtbare Platzierungen hin. Bei den nicht verdrahtbaren „*leon2*"-Platzierungen basieren die Tabellenangaben auf den Schätzungen der Quartus-Software (Tabelle 7-6, Tabelle 7-8) bzw. der PALLAs Software (Tabelle 7-7). Die Verdrahtbarkeit der zunächst nicht verdrahtbaren und mit MPCPlace erzielten „*leon2*"-Platzierung wird im Abschnitt 7.5 mit einer optionalen Strategie zur Modellierung und Reduzierung der lokalen Kongestion (vgl. Abschnitt 6.9) verbessert und sichergestellt.

Design	VPack + VPR	ARH-Pack + VPR	SCPlace	MPCPlace	Quartus
mux64_16bit	21	17	16	15	22
oc_ata_ocidec3	22	22	32	16	29
oc_ata_vhd_3	21	18	29	16	29
oc_cordic_p2r	23	22	14	17	20
oc_des_des3perf	61	52	70	36	39
oc_des_perf_opt	45	41	36	22	30
leon2	124*	116*	104*	91*	68
mc8051	46	41	34	29	40
oc_pavr	53	54	47	43	45
oc_simple_fm	14	10	12	7	12
oc_video_dct	41	39	63	34	40
oc_video_jpeg	50	48	64	42	35

Tabelle 7-8: *Höchste lokale Nutzung der verfügbaren FPGA-Verdrahtungsressourcen (Angaben jeweils in %; mit ‚*' sind nicht vollständig verdrahtbare Platzierungen markiert).*

Anhand der Platzierergebnisse von VPR und der Clusteringergebnisse von VPack bzw. ARH-Pack kann die Korrelation zwischen der Verdrahtbarkeit (Nutzung der Verdrahtungsressourcen in Tabelle 7-6 und Tabelle 7-8) und der Anzahl global zu verdrahtenden Netze bzw. dem Fanout dieser Netze (Tabelle 7-4 bzw. Tabelle 7-5) belegt werden. Dieser Zusammenhang besteht nur, wenn von einer unveränderlichen Clustering-Lösung ausgegangen wird. Die zusätzliche Beweglichkeit der Logikzellen beim SCPlace im Vergleich zu VPR bietet Potential für eine weitere Reduzierung der Netzlängen. Dieses Potential kann jedoch von SCPlace nicht genutzt werden (Tabelle 7-9), da die herkömmliche Bounding-Box-basierte Kostenfunktion die Architekturmerkmale und Einschränkungen der FPGA-Zielarchitektur nicht berücksichtigt. Deshalb ist die Anzahl der global zu verdrahtenden Netze und deren Fanout bei SCPlace im Vergleich zu ARH-Pack höher, das gezielt versucht, möglichst viele Netze in die Logikblöcke zu packen. Insbesondere bei sehr großen Designs sind die mit ARH-Pack+VPR erzeugten Platzierungen besser verdrahtbar als die von SCPlace. Das belegt, dass beim gleichzeitigen Platzieren und Clustern der Logikzellen nicht nur die Gesamtverdrahtungslänge minimiert, sondern auch die Nutzung der lokalen Verdrahtungsressourcen gesteigert werden muss. Diese Erkenntnis wird in der Kostenfunktion C_{II} des neuen Platzierverfahrens MPCPlace umgesetzt.

Die Wirksamkeit der Kostenfunktion C_{II} des neuen Platzierverfahrens MPCPlace und die Verbesserung der Verdrahtbarkeit der erzeugten Platzierungen wird anhand der vorliegenden Ergebnissen belegt. Bei den meisten Designs konnte die Nutzung der globalen C4/R4-Verdrahtungssegmente gesenkt werden (Tabelle 7-6). Weiterhin wurde die Nutzung der lokalen Inter-Logikblocksegmente gesteigert, die zur Entlastung der globalen Verdrahtungsressourcen beitragen können (Tabelle 7-7). Interessanterweise ermöglichen die von der Quartus-Software erzeugten Platzierungen nur eine geringe Nutzung der lokalen

Inter-Logikblocksegmente. Das verwundert, wenn man bedenkt, dass diese Verdrahtungsressourcen von Altera eigens eingeführt wurden, um die globalen Verdrahtungsressourcen bei Designs mit hoher lokaler Interkonnektivität zu entlasten.

	Netzlängen [LE]		Netzanzahl		Fanout	
Design	**ARH-Pack + VPR**	**SCPlace**	**ARH-Pack**	**SCPlace**	**ARH-Pack**	**SCPlace**
mux64_16bit	6.162	5.901	893	1.104	2.640	3.265
oc_ata_ocidec3	5.912	6.020	898	934	2.823	3.064
oc_ata_vhd_3	5.791	5.895	861	909	2.820	3.014
oc_cordic_p2r	5.810	8.150	1.173	1.165	3.110	3.152
oc_des_des3perf	87.791	123.993	7.572	10.653	26.565	37.635
oc_des_perf_opt	25.596	29.845	2.214	3.021	7.869	10.048
leon2	129.666	148.948	11.601	14.480	39.957	46.310
mc8051	18.612	19.140	2.298	2.579	8.858	10.384
oc_pavr	26.858	29.408	2.369	3.172	9.911	11.704
oc_simple_fm	3.291	3.924	591	559	1.890	1.969
oc_video_dct	57.404	99.528	8.511	9.366	22.647	27.613
oc_video_jpeg	66.211	100.606	8.054	9.116	21.902	27.800

Tabelle 7-9: *Gegenüberstellung der Gesamtnetzlänge, der Anzahl global zu verdrahtender Netze, des Fanouts der von ARH-Pack+VPR und SCPlace erzeugten Platzierungen.*

Obwohl die in PALLAs implementierten Clustering- und Platzieralgorithmen versuchen, nur die Verdrahtbarkeit der erzeugten Platzierungen zu steigern, ermöglicht ein Vergleich der maximalen Taktfrequenzen, weitere Zusammenhänge zu erkennen (Tabelle 7-10). So weisen die mit MCPlace erzeugten Platzierungen die höchsten erreichten Taktfrequenzen unter den betrachteten Algorithmen auf, vor allem bei Designs mit hoher Auslastung der Logik- und Verdrahtungsressourcen (vgl. Tabellen 7-1, 7-6, 7-8 und 7-10). Das kann darauf zurückgeführt werden, dass die von der Kostenfunktion C_{II} bevorzugten lokalen Verdrahtungsarten zugleich die schnellste Verdrahtung ermöglichen.

Design	**VPack + VPR**	**ARH-Pack + VPR**	**SCPlace**	**MPCPlace**	**Quartus**
mux64_16bit	71,4	88,7	109,5	94,9	84,5
oc_ata_ocidec3	101,8	137,2	144,4	139,6	122,5
oc_ata_vhd_3	109,8	124,3	144,3	130,6	123,2
oc_cordic_p2r	184,6	178,7	163,1	190,3	149,6
oc_des_des3perf	79,1	71,5	69,5	85,1	98,9
oc_des_perf_opt	81,0	87,7	101,3	106,4	104,9
leon2	-*	-*	-*	-*	27,9
mc8051	29,4	34,2	34,9	36,0	28,1
oc_pavr	22,4	24,6	23,0	28,9	27,1
oc_simple_fm	32,1	33,3	34,9	35,2	32,0
oc_video_dct	55,5	50,0	46,4	50,6	48,8
oc_video_jpeg	55,8	55,8	42,4	61,2	55,4

Tabelle 7-10: *Maximale Taktfrequenzen der verdrahteten Designs (mit ‚*' markierte Designs konnten aufgrund unvollständiger Verdrahtung keiner Timing-Analyse unterzogen werden).*

7.5 Platzierung mit MPCPlace unter Minimierung der lokalen Kongestion

Findet bei einer Platzierung eine lokale Überbenutzung der Verdrahtungsressourcen statt und kann dadurch keine vollständige Verdrahtung durchgeführt werden, bietet MPCPlace eine optionale Strategie zur Modellierung und Reduzierung der lokalen Kongestion (vgl. Abschnitt 6.9). Um die Effektivität dieser Strategie zu belegen, wurden mit ihr drei Designs mit hoher lokaler Kongestion platziert und anschließend verdrahtet. Die Ergebnisse der Verdrahtungs- und Timing-Analyse sind in der Tabelle 7-11 zusammengefasst. In der Tabelle 7-12 werden die mit Quartus erreichten Ergebnisse erneut aufgeführt, um einen direkten Vergleich der beiden Verfahren zu ermöglichen.

Design	#C4/R4	#lok. Inter-LB	lok. Kongestion	max. Takt
oc_des_des3perf	21.272	3.429	32 %	123,2 MHz
leon2	39.986	3.588	55 %	31,5 MHz
oc_pavr	8.734	970	47 %	28,6 MHz

Tabelle 7-11: *Ergebnisse der Verdrahtungs- und Timing-Analyse der von MPCPlace erzeugten Platzierungen unter Reduzierung der lokalen Kongestion.*

Design	#C4/R4	#lok. Inter-LB	lok. Kongestion	max. Takt
oc_des_des3perf	29.983	1.104	39 %	98.9 MHz
leon2	37.822	1.627	68 %	27.9 MHz
oc_pavr	8.047	264	45 %	27.1 MHz

Tabelle 7-12: *Ergebnisse der Verdrahtungs- und Timing-Analyse der von Quartus erzeugten Platzierungen.*

Die lokale Überbenutzung der Verdrahtungsressourcen konnte bei den beiden größeren Designs (*oc_des_des3perf, leon2*) deutlich auf 32 bzw. 55 % reduziert werden. Die Reduzierung der Kongestion bei diesen beiden Designs wurde durch die gleichmäßigere Verteilung der benötigten Logik- und Verdrahtungsressourcen über eine größere Fläche des Ziel-FPGA ermöglicht (vgl. Abbildung 6-5b und Abbildung 6-7d). Beim dritten Design (*oc_pavr*) konnte keine Reduzierung der lokalen Kongestion erreicht werden, da das Ziel-FPGA keine freien Logikressourcen aufwies (96 % Logiknutzung, vgl. Tabelle 7-1). Dieses Design belegt, dass die optionale MPCPlace-Strategie zur Modellierung und Reduzierung der lokalen Kongestion nur dann erfolgreich ist, wenn eine Verteilung der Logik über eine größere FPGA-Fläche möglich ist. Diese Strategie sollte deshalb nur bei hierfür geeigneten Designs mit noch freien Logikressourcen eingesetzt werden.

Die zusätzliche Verbesserung der Verdrahtbarkeit durch Reduzierung der lokalen Kongestion vereinfacht die Aufgabe des Verdrahtungswerkzeugs, da in den zuvor kritischen FPGA-Bereichen Verdrahtungsressourcen frei werden. Deshalb steigt bei den betrachteten Designs die maximale Taktfrequenz umso mehr, je stärker die lokale Kongestion im Vergleich zur ursprünglichen Platzierung reduziert werden kann. Bei dem „*oc_des_des3perf*“-Design konnte die mit der MPCPlace-Platzierung erreichte Taktfrequenz von ursprünglich 85,1 MHz (vgl. Tabelle 7-10) um 45 % auf 123,2 MHz gesteigert werden. Sogar im Vergleich zum herstellereigenen Quartus-Werkzeug konnte dieses Designs mit der optionalen MPCPlace-Strategie um 25 % höher getaktet werden.

7.6 Testumgebung und Platzierlaufzeiten

Als Laufzeitumgebung für PALLAs und für die Altera-Quartus-II-v7.2 Software diente ein Windows-XP-System, ausgestattet mit einem 2,5 GHz schnellen AMD Athlon64 Doppelkern-prozessor und zwei Gigabyte Arbeitsspeicher. Die ermittelten Platzierlaufzeiten (Tabelle 7-13) sollen eine Abschätzung der Rechenzeit in Abhängigkeit von der Designgröße ermöglichen.

Design	VPack + VPR	ARH-Pack + VPR	SCPlace	MPCPlace	Quartus
mux64_16bit	9 s	7 s	44 s	61 s	7 s
oc_ata_ocidec3	7 s	6 s	47 s	63 s	5 s
oc_ata_vhd_3	7 s	7 s	52 s	65 s	6 s
oc_cordic_p2r	13 s	12 s	8 min	11 min	5 s
oc_des_des3perf	259 s	230 s	24 min	34 min	80 s
oc_des_perf_opt	50 s	56 s	292 s	441 s	19 s
leon2	366 s	339 s	33 min	4 h 42 min	333 s
mc8051	56 s	47 s	354 s	569 s	61 s
oc_pavr	158 s	142 s	13 min	28 min	56 s
oc_simple_fm	63 s	60 s	86 s	162 s	5 s
oc_video_dct	367 s	357 s	4 h 23 min	45 h 21 min	117 s
oc_video_jpeg	312 s	318 s	3 h 36 min	37 h 19 min	137 s

Tabelle 7-13: *Platzierlaufzeiten der in PALLAs implementierten Platzieralgorithmen und der herstellereigenen Software Altera Quartus II v7.2.*

Erwartungsgemäß weisen Platzierverfahren, die von einer festen Clustering-Lösung ausgehen (VPack/ARH-Pack+VPR), die geringsten Platzierlaufzeiten auf. Sehr viel höher sind die Platzierlaufzeiten bei den beiden Verfahren, die Beweglichkeit der Logikzellen beim Platzieren zulassen (SCPlace/MPCPlace). Zurückzuführen lässt sich die Zunahme der Rechenzeit auf die zehn Mal so hohe Anzahl der bewegbaren Blöcke bzw. auszuwertenden Bewegungen. Aufgrund der Komplexität der Kostenfunktion C_{II}, die beim zweiten MPCPlace-Durchlauf zur Bewertung der Logikzellenbewegungen eingesetzt wird, sind die Platzierlaufzeiten von MPCPlace höher als die von SCPlace. Besonders bei den Designs mit einem hohen Anteil an Carry-Chains (vgl. Tabelle 7-1) steigen die Laufzeiten beider Verfahren auf ein nicht akzeptables Maß an. Die Ursache dafür liegt in der zunehmenden Schwierigkeit, bei der Bewegungsgenerierung einen passenden Tauschbereich für eine zu bewegende Carry-Chain zu finden. Deshalb muss die implementierte Bewegungsgenerierung für die Carry-Chains als suboptimal bewertet werden. Das der Quartus-Software zugrunde gelegte Platzierverfahren ist leider nicht bekannt, deshalb kann nur spekuliert werden, ob die geringen Laufzeiten dieses Werkzeugs auf besonders effiziente Implementierung von Simulated Annealing (sofern dieses Verfahren eingesetzt wird) oder auf die Verwendung eines anderen (iterativen) Verfahrens zurückgeführt werden können.

7.7 Analyse und Bestimmung des Leistungsbedarfs

Wie bereits im Abschnitt 2.3.3 erwähnt, hängt die Leistungsaufnahme eines mit einem FPGA implementierten Systems vom Ressourcenbedarf dieses Systems ab. Für einen erheblichen Anteil des dynamischen Leistungsbedarfs sind die verwendeten Taktnetzwerke und Verdrahtungsressourcen verantwortlich [94]. Kann die Nutzung der Verdrahtungsressourcen verringert werden, so wie das mit MPCPlace durch Steigerung der Verdrahtbarkeit erreicht

wird, ist eine Leistungsbedarfsreduzierung eines FPGA-basierten Systems zu erwarten. Der Hersteller Altera bietet als Bestandteil seiner Quartus-Software ein Analysewerkzeug zur Abschätzung des Leistungsbedarfs einer Schaltung. Leider konnte nach der Quartus-Analyse der fertig verdrahteten Designs kein Unterschied im theoretischen Leistungsbedarf der mit verschiedenen Verfahren durchgeführten Platzierungen festgestellt werden. Angesichts der Platzier- und Verdrahtungsvarianten, deren Bedarf an globalen C4/R4-Segmenten sich teilweise um das Doppelte unterscheidet, war das Ergebnis dieser Leistungsanalyse nicht nachvollziehbar. Aus diesem Grund wurde eine praktische Messung des Leistungsbedarfs repräsentativ anhand eines Designs durchgeführt.

7.7.1 Messaufbau

Für die praktische Messung des Leistungsbedarfs einzelner Verdrahtungsvarianten wurde die „Open Cores"-Implementierung *oc_des_des3perf* [124] des weit verbreiteten DES3-Verschlüsselungsverfahrens (DES3-IP) gewählt. Bei diesem Design liegt der Bedarf an globalen Verdrahtungsressourcen zwischen 21.272 C4/R4-Segmenten (MPCPlace mit Minimierung der lokalen Kongestion) und 38.158 C4/R4-Segmenten (SCPlace). Erwartungsgemäß sollten hier die größten Leistungsunterschiede zu messen sein.

Beim DES3-Verschlüsselungslagorithmus werden die Daten durch eine Reihe von Permutationen und Additionen chiffriert. Sowohl die Daten als auch der Schüssel werden bei der „Open Cores"-Implementierung parallel zugeführt. Zur Generation der Eingangsvektoren wurde dem ursprünglichen „*oc_des_des3perf*"-Design ein 232 Bit breites Ringschieberegister hinzugefügt, das mit einem zufällig festgelegten, jedoch für alle Messungen identischen Wert initialisiert wird (Abbildung 7-1). Bei jeder Taktflanke wird der Wert innerhalb des Schieberegisters um eine Stelle verschoben, somit liegt bei jedem Taktzyklus ein neuer Klartext und Schlüssel vor. Dadurch wird ein Maximum an Schaltaktivitäten auf den benutzten Verdrahtungssegmenten innerhalb der DES3-Komponente erzeugt. Mit dem Reset-Signal *rst* können die Eingangsvektoren auf Null zurückgesetzt und somit alle Schaltaktivitäten auf den Verdrahtungssegmenten wirksam unterbunden werden. Weiterhin kann das Taktsignal *clk* und somit auch die Schaltaktivitäten der Taktbäume mit Hilfe des Enable-Signals *en* gänzlich abgestellt werden.

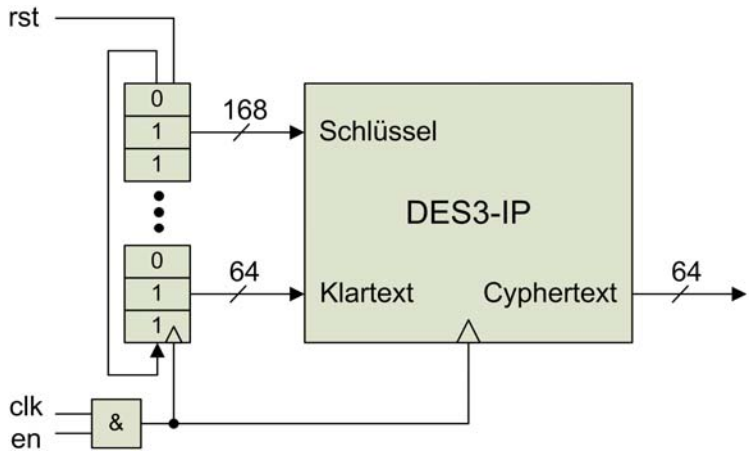


Abbildung 7-1: *Generation der Eingangsvektoren für die DES3-IP.*

Nach dem Einbringen des Schieberegisters in das „*oc_des_des3perf*"-Design wurde der Entwurfsfluss von der Synthese bis zur Verdrahtung erneut durchgeführt. Da hierbei die Datenbusse für Schlüssel- und Klartexteingabe nicht mehr mit den FPGA-Pins, sondern mit

den Ausgängen des Schieberegisters verbunden wurden, ergab sich insgesamt eine abweichende Nutzung der Verdrahtungsressourcen (Tabelle 7-14). Weiterhin stieg die Nutzung der Logikressourcen um die 232 Flipflops des Schieberegisters.

Leistungsbedarf	VPack + VPR	ARH-Pack + VPR	SCPlace	MPCPlace + Cong.	Quartus
#Logikblöcke	1.390	1.390	1.489	1.741	1.769
#C4/R4	32.548	27.301	33.325	18.181	24.011

Tabelle 7-14: *Ressourcennutzung des modifizierten „oc_des_des3perf"-Designs.*

Als Hardware-Plattform kam das Altera-NIOS-Development-Board zum Einsatz (Abbildung 7-2). Auf diesem Board sind neben einem Cyclone-EP1C20-Baustein im BGA-Gehäuse diverse Speicher- und Peripheriebausteine enthalten. Das eingesetzte FPGA weist getrennte Versorgungsschienen für die Ein- und Ausgangspins (3,3 Volt) und den FPGA-Kern (1,5 Volt) auf. Die unterschiedliche Nutzung der Logik- und Verdrahtungsressourcen der „*oc_des_des3perf*"-Designvariationen wirkt sich nur auf den Leistungsbedarf des FPGA-Kerns aus. Zur Messung des durch den FPGA-Kern fließenden Stroms wurde in den Strompfad ein Universalmessgerät eingefügt. Der resultierende Leistungsbedarf ergibt sich durch die Multiplikation des gemessenen Stroms mit dem Wert der FPGA-Kernspannung von 1,5 Volt.

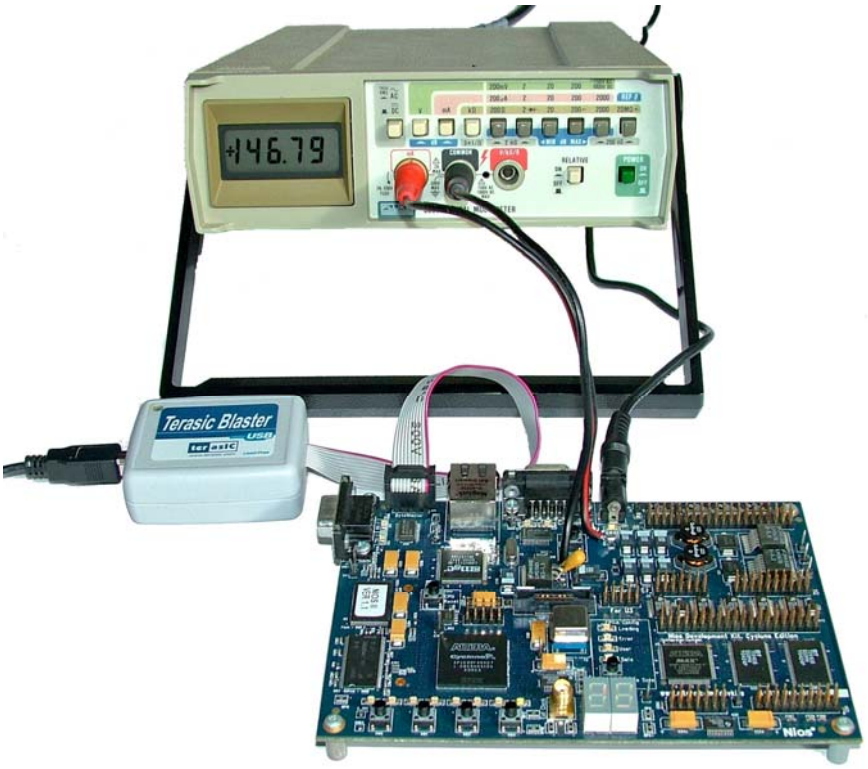


Abbildung 7-2: *Messaufbau zur Messung des Stroms durch den FPGA-Kern.*

7.7.2 Messung und Auswertung des Leistungsbedarfs

Der Strom durch den FPGA-Kern wurde für die „*oc_des_des3perf*“-Designvariationen nach drei folgenden Szenarien gemessen und anschließend der Leistungsbedarf ermittelt:

- Keine Schaltaktivitäten auf den Verdrahtungssegmenten und dem Taktbaum – ermittelt wird der statische Leistungsbedarf,
- Schaltaktivitäten nur auf dem Taktbaum – nach dem Abzug des statischen Leistungsbedarfs wird so der Leistungsbedarf des Taktbaums ermittelt,
- Schaltaktivitäten sowohl auf dem Taktbaum, als auch auf Verdrahtungssegmenten – nach dem Abzug des statischen Leistungsbedarfs wird so der dynamische Leistungsbedarf ermittelt.

Die ermittelten Leistungswerte sind wie folgt (Tabelle 7-15) zusammengefasst.

Leistungsbedarf	VPack + VPR	ARH-Pack + VPR	SCPlace	MPCPlace + Cong.	Quartus
statisch	44 mW	44 mW	42 mW	42 mW	45 mW
nur Taktbaum	53 mW	77 mW	81 mW	92 mW	90 mW
dynamisch	909 mW	914 mW	998 mW	831 mW	903 mW

Tabelle 7-15: *Leistungsbedarf der unterschiedlichen „oc_des_des3perf“-Designvariationen.*

Da der statische Leistungsbedarf nur von der eingesetzten Fertigungstechnologie, Temperatur und der FPGA-Größe abhängt, ist dieser bei den unterschiedlichen „*oc_des_des3perf*“-Designvariationen nahezu identisch. Der Leistungsbedarf des Taktbaums steigt zusammen mit der Anzahl der an diesen Taktbaum angeschlossener Logikblöcke, da mit jedem zusätzlichen Logikblock die an den Taktbaum angeschlossene und somit umzuladende Gesamtleitungskapazität erhöht wird. Hier zeigen Implementierungen der beiden Platzierverfahren, die die Anzahl der benutzten Logikblöcke verringern (VPack/ARH-Pack+VPR), erwartungsgemäß den geringsten Leistungsbedarf des Taktbaums auf. Wie bereits erwartet, weisen die „*oc_des_des3perf*“-Designvariationen mit geringerem Bedarf an globalen Verdrahtungsressourcen in Form der C4/R4-Segmente den niedrigeren dynamischen Leistungsbedarf auf. Somit wird belegt, dass die Steigerung der Verdrahtbarkeit ein mit Reduzierung des Leistungsbedarfs einhergehendes Optimierungskriterium darstellt.

8 Implementierung

8.1 Überblick

Das PALLAs-Werkzeug besteht aus einer Tcl/Tk-basierten graphischen Benutzeroberfläche (GUI) und einer maschinenlesbaren Dynamic-Link-Library (DLL), welche die Kernfunktionen beinhaltet (Abbildung 8-1). Die Aufgabe der GUI ist neben dem Bereitstellen einer graphischen Benutzeroberfläche das Aufrufen der DLL-Kernfunktionen und die Steuerung der Quartus-Werkzeuge. Die DLL wird während der Laufzeit von der Tcl/Tk-GUI eingebunden. Deren Kernfunktionen beinhalten neben den eigentlichen Clustering- und Platzieralgorithmen auch die Ein- und Ausgabefunktionen, welche über entsprechende Import- und Exportdateien die Schnittstellen zur Altera-QUIP bilden. Das Zusammenspiel der einzelnen Komponenten soll an einem kompletten Ablauf des PALLAs-Entwurfsflusses verdeutlicht werden.

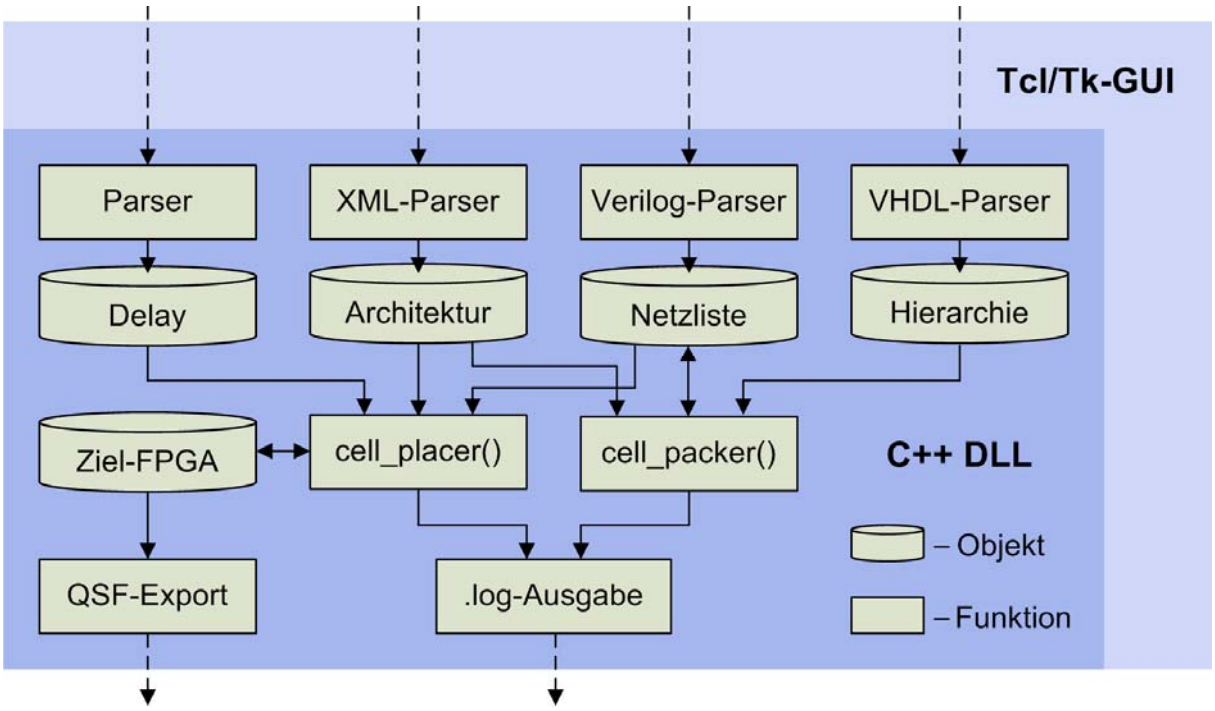


Abbildung 8-1: *Schematischer Aufbau des PALLAs-Werkzeugs.*

Bevor die Clustering- und Platzieralgorithmen ausgeführt werden können, müssen zahlreiche Importdateien von eigens zu diesem Zweck implementierten Parsern eingelesen und eine interne Repräsentation der eingelesenen Daten in Form von C++-Objekten angelegt werden. Die von Altera-Quartus erzeugten Netzlistendateien liegen als Schaltungsbeschreibungen in einer HDL vor, die auf einer Untermenge des Verilog-Sprachumfangs basiert [120]. Der Verilog-Parser liest die Netzliste ein und erzeugt ein entsprechendes Netzlistenobjekt. Von Altera werden mit QUIP XML-basierte [100] Architekturbeschreibungen der Cyclone- und Stratix-Familien zur Verfügung gestellt. Diese enthalten neben der Beschreibung des hierarchischen Aufbaus der Logikblöcke und der dedizierten Funktionseinheiten auch die exakte Position der Logikressourcen für alle verfügbaren Mitglieder der beschriebenen FPGA-Familien [122]. Der XML-Parser setzt eine XML-basierte Architekturbeschreibung in ein Architekturobjekt mit gleichem logischen Aufbau um. Mit Hilfe eines VHDL-Parsers werden aus der ursprünglichen Schaltungsbeschreibung die Struktur- und Hierarchieinformationen extrahiert. Sobald die internen Repräsentationen der FPGA-

Architektur, der Netzliste und der Hierarchie einer Schaltung vorliegen, können das Packen der LUTs und Flipflops in die Logikzellen und das Clustering der Logikzellen erfolgen. Diese beiden Entwurfsschritte werden von der Kernfunktion *cell_packer()* durchgeführt. Die von der Tcl/Tk-GUI an die *cell_packer()*-Funktion übergebenen Aufrufparameter bestimmen die dabei zu verwendenden Pack- und Clusteringalgorithmen. Die durchgeführten Manipulationen wirken sich direkt auf die interne Netzlistendarstellung aus, so dass nach dem Clustering-Entwurfsschritt das gleiche Datenobjekt nur noch geclusterte Logik- und RAM-Blöcke aufweist.

Die Platzierung einer Netzliste wird von der Kernfunktion *cell_placer()* durchgeführt, welche von der Tcl/Tk-GUI neben dem zu verwendenden Platzieralgorithmus auch die explizite Festlegung des Ziel-FPGA erhält. Aufgrund dieser Vorgabe wird vor der Platzierung zunächst eine topologische Abbildung des Ziel-FPGA mit den verfügbaren Logikressourcen aus der Architekturbeschreibung erzeugt. Während der Platzierung erfolgt eine Zuordnung der Logik- und RAM-Blöcke des Netzlistenobjekts zu den im Ziel-FPGA-Objekt enthaltenen Kacheln des gleichen Logikressourcentyps. Zur Auswertung der Verzögerungszeiten während der Platzierung wird von Altera eine entsprechende Funktionalität zur Verfügung gestellt. Nach der Platzierung werden die absoluten Koordinaten aller LUTs, Flipflops und RAM-Grundzellen als Vorgaben für die Quartus-Software in eine QSF-Datei geschrieben. Beim Ausführen von *cell_packer()* und *cell_placer()* anfallende Informationen werden sowohl über die Tcl/Tk-GUI wiedergegeben als auch zur späteren Auswertung in die Protokolldateien geschrieben.

Die Quellen der PALLAs-DLL liegen in der objektorientierten Sprache C++ [99] vor und umfassen etwa 15.000 Zeilen Quellcode. Die Tcl/Tk-basierte GUI umfasst circa 3.000 Zeilen. Die Skripte der Sprache Tcl/Tk sind aufgrund zahlreicher Interpreterportierungen auf nahezu jedem bekannten Betriebssystem (Linux, Unix, Windows, MacOS usw.) ausführbar [110]. Bei der Implementierung der PALLAs-GUI und der PALLAs-DLL wurde auf die Verwendung systemspezifischer Funktionsaufrufe verzichtet, weshalb zur Portierung auf weitere Systeme nur eine Neuübersetzung des DLL-Quellcodes erforderlich ist. Im Augenblick liegt eine PALLAs-DLL nur für Windows-Systeme vor. Die Aufteilung des PALLAs-Werkzeugs in die beiden Bestandteile erfolgte aufgrund der unterschiedlichen Anforderungsprofile der PALLAs-Funktionalität an die Programmiersprachen. Die in PALLAs implementierten Platzier- und Clusteringalgorithmen müssen, um die Laufzeiten möglichst gering zu halten, die verfügbare CPU-Rechenleistung effizient nutzen. Deshalb wurde für diese Algorithmen die maschinennahe Sprache C++ ausgewählt. Die graphische Benutzeroberfläche soll die Interaktion externer Quartus-Werkzeuge und interner PALLAs-Algorithmen starten und überwachen. Die hierzu eingesetzte Skriptsprache Tcl/Tk wurde als Werkzeugbefehlsprache entworfen und ist für diese Aufgaben prädestiniert [87]. Die Implementierung der geforderten PALLAs-Funktionalität unter Verwendung von zwei Programmiersprachen mit unterschiedlichen Einsatzprofilen ermöglichte deshalb einen sehr kompakten und gut lesbaren Quellcode.

Die Verwendung der PALLAs-Software erfordert eine vorinstallierte und lizenzierte Altera-Quartus-II-Entwicklungsumgebung [109], da u. a. Synthese, Verdrahtung und Timing-Analyse von der Altera-Quartus-II-Software durchgeführt werden. Weiterhin werden in der Altera-QUIP [108] enthaltene Architekturbeschreibungen und Verzögerungszeitendateien der Cyclone- und Stratix-FPGA-Familien benötigt. Zur Ausführung der Tcl/Tk-Skripte muss ein Tcl/Tk-Interpreter auf dem System eingerichtet sein. Hier bildet die ActiveTcl-Distribution von ActiveState den Industriestandard [111].

8.2 Interne Datenstrukturen

8.2.1 Architekturobjekt

Ein Architekturobjekt wird vom XML-Parser angelegt und anhand einer vorliegenden XML-Architekturdatei mit Inhalten in der Form von Datenstrukturen gefüllt. Nach dem Parsen beinhaltet ein Architekturobjekt eine vollständige Beschreibung einer Altera-Zielarchitektur. Die hierarchische Anordnung der Datenstrukturen spiegelt die logische Struktur der XML-Architekturbeschreibung (vgl. Abbildung 8-2 und [122]) wider.

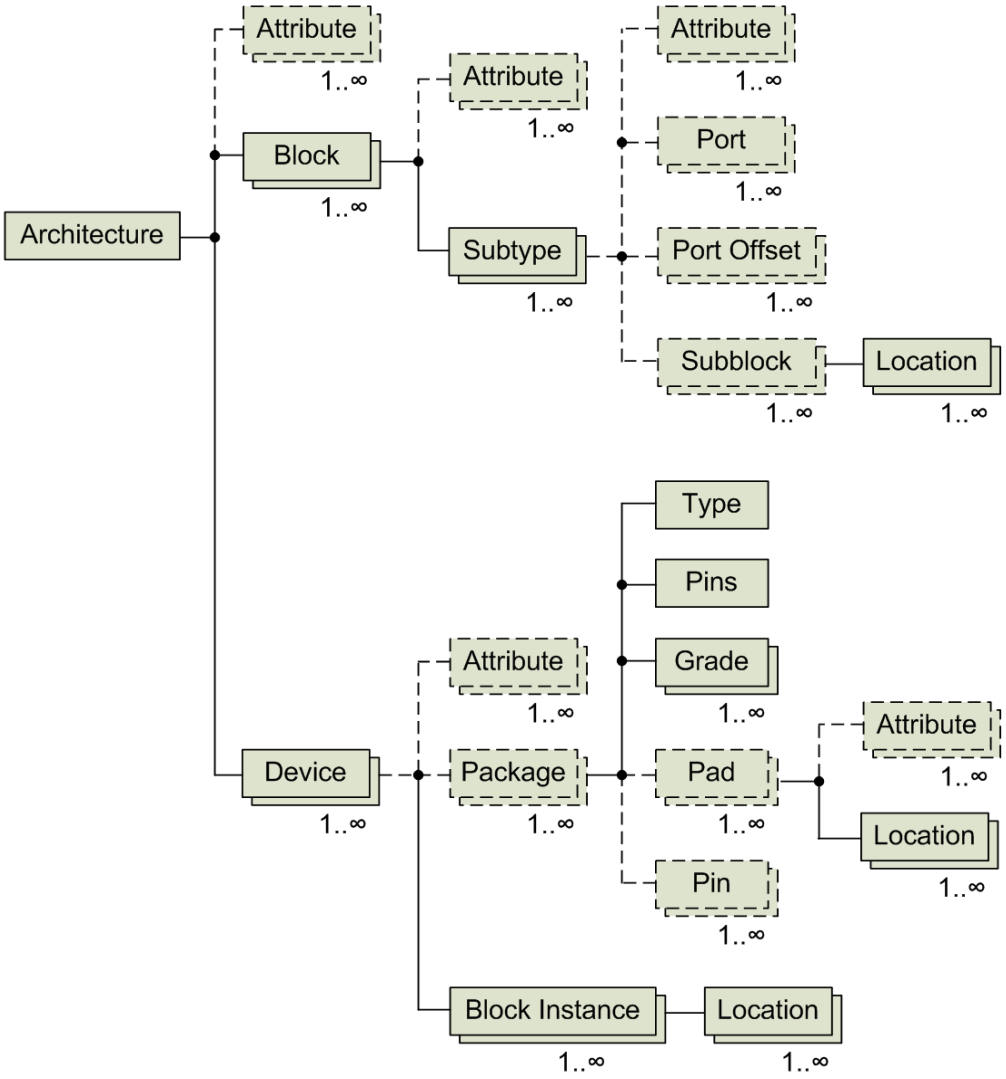


Abbildung 8-2: *Baumdarstellung eines Architekturobjekts (optionale Datenstrukturen sind gestrichelt dargestellt).*

Ein Architekturobjekt besteht in der Regel aus mehreren *Block*- und *Device*-Strukturen. Die *Block*-Strukturen deklarieren jede einzelne FPGA-Logikressource unter Angabe der verwendeten Schnittstellen, der relativen Position der Schnittstellen und der Kachelgröße. Die so beschriebenen Logikressourcen können wiederum über die *Subblock*-Struktur zur Beschreibung der übergeordneten und hierarchisch aufgebauten Logikressourcen, wie der Logik- und RAM-Blöcke, eingebunden werden.

Eine FPGA-Familie besteht in der Regel aus einigen wenigen, unterschiedlich großen Chip-Typen, die in eine Vielzahl von Gehäuseformen eingebettet werden. Die Beschreibung aller verfügbaren Mitglieder einer FPGA-Familie erfolgt über die *Device*-Struktur. Die darin enthaltenen *Block-Instance*-Strukturen beschreiben unter Angabe der verwendeten Logikressourcen und deren absoluten Position die topologische Struktur eines FPGA-Chips. Die Gehäuseformen, in welche die beschriebenen FPGA-Chips eingebettet werden können, sind wie ihre *Pin*-zu-*Pad*-Zuordnungen gesondert in der *Package*-Struktur zusammengefasst.

8.2.2 FPGA-Objekt

Die zur Erstellung eines FPGA-Objekts, welches einen bestimmten FPGA-Baustein repräsentiert, benötigten Informationen werden von der *cell_placer()*-Funktion aus dem Architekturobjekt der Zielarchitektur extrahiert. Ein FPGA-Objekt beschreibt die Topologie des FPGA-Bausteins, den der PALLAs-Anwender als Ziel-FPGA über die Vorgaben bestimmt hat. Die Verknüpfung der Kachelstrukturen über so genannte *Corner-Stitches*-Zeiger innerhalb eines FPGA-Objekt ermöglicht eine äußerst effiziente Implementierung grundlegender Funktionen wie Suchen der Nachbarkachel, Suchen einer Kachel an einer bestimmten Koordinate und vor allem „Durchwandern“ der RAM- bzw. Logikblockspalten [86]. Jede Kachel enthält unabhängig von ihrer Größe vier *Corner-Stitches*-Zeiger (zwei an der rechten oberen und zwei an der linken unteren Ecke), die auf die benachbarten Kacheln verweisen (Abbildung 8-3). Dadurch kann, ausgehend von einer Kachelstruktur, unabhängig von der tatsächlichen Anordnung der Kachelstrukturen im Arbeitsspeicher, mit geringem Aufwand eine beliebige andere Kachelstruktur erreicht werden.

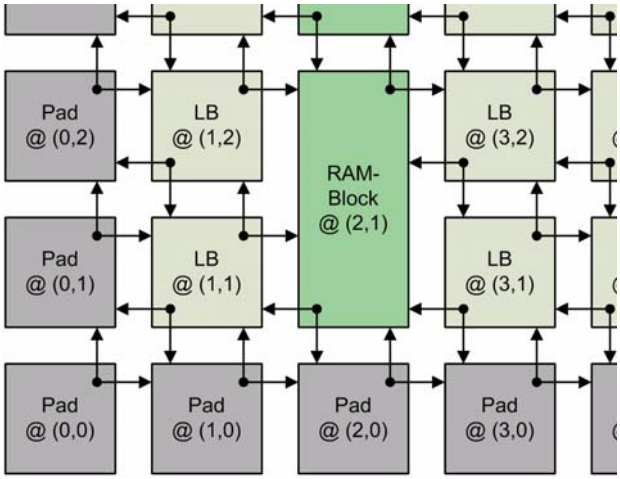


Abbildung 8-3: *Gegenseitige Verknüpfung der Kacheln über die Corner-Stitches-Zeiger.*

Ein FPGA-Objekt enthält neben den Angaben zu den Abmessungen des FPGA-Chips und der Kachelgesamtanzahl einen Speicherarray, in dem die Kachelstrukturen abgelegt sind (Abbildung 8-4). Wie bereits erläutert, sind die Kachelstrukturen untereinander über die *Corner-Stitches*-Zeiger verknüpft. Weiterhin sind in der Kachelstruktur die Abmessungen, die absoluten Koordinaten und der Kacheltyp vermerkt. Die Typangabe stellt sicher, dass die Platzierfunktion nur typidentische Logikressourcen den Kacheln zuordnen kann. Die Zuordnung der Logikressourcen zu einer Kachel erfolgt über den *Target*-Zeiger, der auf die entsprechende Logikressource in dem Netlistenobjekt verweist. Bei dem MPCPlace-Platzierverfahren kann die Überbenutzung der globalen Verdrahtungsressourcen modelliert und über die Einschränkung der benutzbaren Logikblock-Eingänge reduziert werden. Die Anzahl der nutzbaren Logikblock-Eingänge wird bei den Logikblockkacheln über die Variable *Usable Inputs* eingeschränkt.

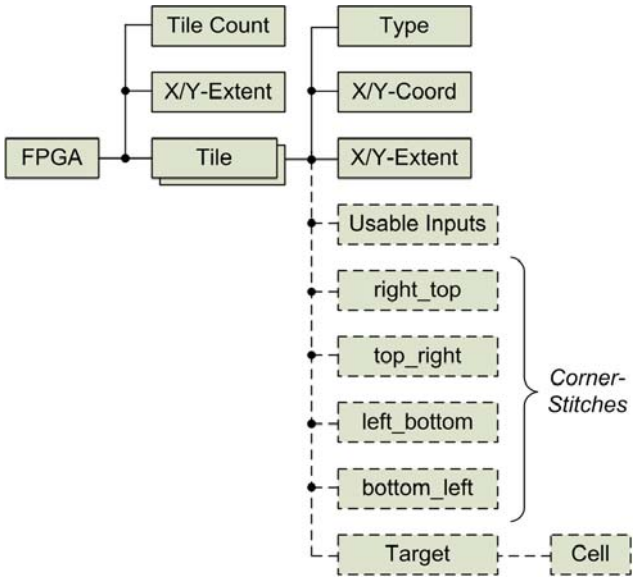


Abbildung 8-4: *Baumdarstellung eines FPGA-Objekts (optionale Datenstrukturen sind gestrichelt dargestellt).*

8.2.3 Netzlistenobjekt

Sowohl die Netzliste selbst als auch die in ihr enthaltenen logischen Ressourcen weisen eine Selbstähnlichkeit auf. So besteht eine Netzliste aus untergeordneten Modulen wie den RAM- und Logikblöcken, die miteinander über Netze verbunden sind. Die Schnittstelle nach Außen bilden die Ein- und Ausgänge des FPGA. Betrachtet man z. B. einen Logikblock, so ist dieser auch aus untergeordneten Modulen (Logikzellen) aufgebaut, die untereinander verdrahtet sind. Die Schnittstellen nach Außen sind die Ein- und Ausgänge des Logikblocks. Die Logikzellen bestehen wiederum aus LUTs und Flipflops. Es war deshalb nahe liegend, für die Netzliste und für jede einzelne Logikressource nicht ein jeweils spezifisches, sondern ein universelles Datenobjekt festzulegen. Die Grundlage zur Repräsentation einer Netzliste und aller Logikressourcen bildet das Zellenobjekt (Abbildung 8-5).

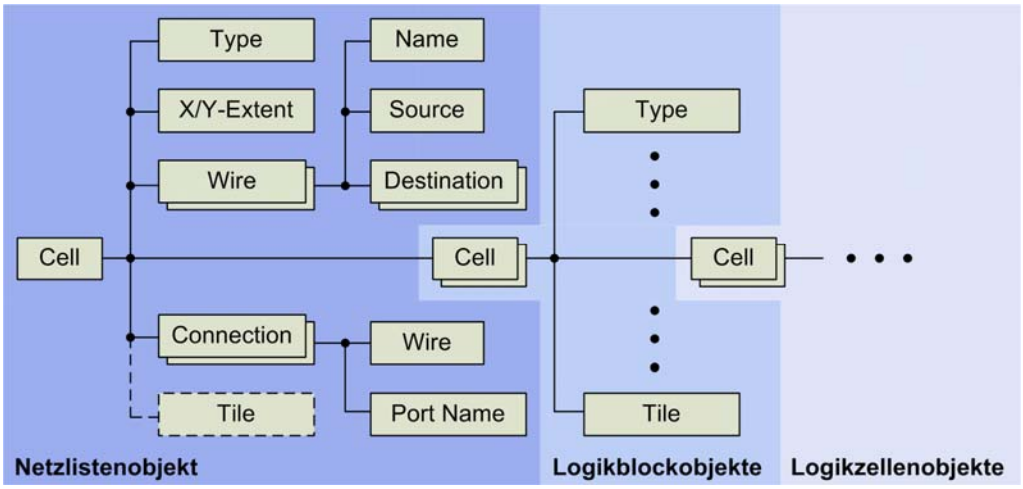


Abbildung 8-5: *Repräsentation einer Netzliste über verschachtelte Zellenobjekte (optionale Datenstrukturen sind gestrichelt dargestellt).*

Das Zellenobjekt wurde so ausgelegt, dass eine hierarchische Verschachtelung der einzelnen Objektinstanzen möglich ist. Die oberste Instanz des Zellenobjekts (in Abbildung 8-5 ist das

die Wurzel der Baumstruktur) repräsentiert die Netzliste selbst. Die im Netzlistenobjekt aufgelisteten Netze (*Wire*-Strukturen) müssen auf der globalen Ebene verdrahtet werden. Die Schnittstellen nach Außen (*Connection*-Strukturen) sind durch die Namen der FPGA-Pads und die Zeiger auf die mit diesen Pads verbundenen Netze vollständig beschrieben. Die in einer Netzliste enthaltenen Logikressourcen in Form von RAM- und Logikblöcken werden in dem zugehörigen Netzlistenobjekt über ein Zeigerarray referenziert und als Zellenobjekte eingebunden.

Die Bedeutung der im Zellenobjekt enthaltenen Strukturen variiert auf den unterschiedlichen Hierarchieebenen. So beschreiben die in den Logikblockobjekten enthaltene *Wire*-Strukturen Netze, die nur lokal innerhalb der entsprechenden Logikblöcke zu verdrahten sind. Die *Connection*-Strukturen der Logikblockobjekte legen fest, mit welchen global zu verdrahtenden Netzen des übergeordneten Netzlistenobjekts die Ein- und Ausgänge der Logikblöcke verbunden sind. Während der Platzierung wird die Zuordnung eines RAM- oder Logikblocks zu einer RAM- oder Logikblockkachel mit Hilfe eines *Tile*-Zeigers hergestellt, der im Netzlistenobjekt ohne Funktion war. Über ein Zeigerarray werden die in einem Logikblock enthaltenen Logikzellen referenziert und wiederum als Zellenobjekte eingebunden.

8.3 Export der Platzierungsinformationen

Nach der Platzierung sind zwischen allen Logikressourcen eines Netzlistenobjekts und den belegten Kacheln eines FPGA-Objekts Verknüpfungen in Form von gegenseitigen Referenzen hergestellt. Damit eine Platzierung von der Quartus-Software übernommen werden kann, müssen die individuellen Bezeichner der verwendeten Logikressourcen zusammen mit den absoluten Positionen der zugeordneten Kacheln in einer QSF-Datei vermerkt werden. Über diese werden die Positionen aller LUTs, Flipflops und RAM-Grundzellen an die Quartus-Software mitgeteilt.

Die Festlegung der absoluten Positionen der LUTs und Flipflops erfolgt über die Angabe der X/Y-Koordinaten der zugeordneten Logikblockkachel und ggf. über die explizite Angabe der zu verwendenden Logikzellenposition innerhalb des Logikblocks. Wird die optionale Angabe der Logikzellenposition weggelassen, entscheidet das Verdrahtungswerkzeug anhand des Verdrahtungsgraphen, welche Logikzellenposition die verdrahtungstechnisch günstigste ist und zu verwenden sei. Für die Übergabe der LUT- und Flipflop-Positionen empfiehlt Altera nur die X/Y-Koordinaten der entsprechenden Logikblockkachel zu übergeben, weil dadurch das Verdrahtungswerkzeug mehr Freiheitsgrade erhält [121]. Dabei geht allerdings die Information über die Logikzellenzusammensetzung verloren, die beim Packen der Logikzellen gewonnen wurde.

Leider ist festzustellen, dass das Quartus-Verdrahtungswerkzeug nicht immer in der Lage ist, die einem Logikblock über dessen X/Y-Koordinaten zugeordneten LUTs und Flipflops effektiv in die Logikzellen zu packen. Bei nahezu vollständig belegten Logikressourcen eines Logikblocks und zahlreichen, den LUTs vor- und nachgeschalteten Flipflops kann das Verdrahtungswerkzeug nicht erkennen, welche LUTs und Flipflops zusammen in die Logikzellen zu packen sind, damit eine flächenoptimale Lösung entsteht. Dieses Problem entspricht im Wesentlichen dem in Abschnitt 4.1 geschilderten (vgl. auch Abbildung 4-2). Um die Zugehörigkeit der LUTs und Flipflops zu einer Logikzelle sicherzustellen, überprüft PALLAs bei der Ausgabe der LUT- und Flipflop-Koordinaten, ob eine alternative Zuordnung der LUTs und Flipflops zu den Logikzellen durch das Verdrahtungswerkzeug möglich ist. Wenn dies der Fall ist, werden die betroffenen LUTs und Flipflops mit expliziten Angaben der Logikzellenposition versehen und somit wird Mehrdeutigkeit ausgeschlossen. Leider wird auf diese Weise auch der Freiheitsgrad des Verdrahtungswerkzeugs eingeschränkt und somit unter Umständen nur eine suboptimale Verdrahtung erzeugt.

9 Ausblick

Das im Rahmen dieser Arbeit entwickelte Werkzeug PALLAs hat demonstriert, dass, sofern seitens der FPGA-Hersteller detaillierte Informationen zum Aufbau ihrer FPGAs und die Softwareschnittstellen offen gelegt werden, durchaus Softwarelösungen möglich sind, die in der Qualität bzgl. eines Optimierungsziels bessere Ergebnisse liefern können als die herstellereigene EDA-Werkzeuge. Möglich wird das deshalb, weil die von FPGA-Herstellern angebotene Software in der Regel ein Gleichgewicht zwischen mehreren Optimierungszielen wie der maximalen Taktfrequenz, der Logiknutzung, der Verdrahtbarkeit und der Rechenzeit zu erreichen anstrebt. Die dabei verwendeten Algorithmen müssen zahlreiche FPGA-Architekturen unterstützen und nutzen deshalb die Besonderheiten einzelner FPGA-Architekturen, wie die lokalen Inter-Logikblockressourcen, nur unzureichend. Aufgrund der Spezialisierung des Werkzeugs PALLAs auf den Altera-Cyclone- und Stratix-Architekturen war es deshalb möglich, Platzierlösungen zu erzeugen, die nahezu durchgehend bessere Verdrahtbarkeit aufwiesen als die implementierten akademische Referenzverfahren und die Altera-Quartus-Software. Die Weiterentwicklung des Werkzeugs PALLAs kann in mehreren Richtungen erfolgen:

- Bisher ist nichts über die Verdrahtungsstrategien des Quartus-Verdrahtungswerkzeugs bekannt. Die Kostenfunktion C_{II} für den zweiten Platzierdurchlauf bevorzugt zwar die empirisch ermittelten verdrahtungsgünstigeren Platzierpositionen; es sind jedoch weitere Verbesserungen in der Platzierqualität denkbar, wenn die Kostenfunktion C_{II} die tatsächliche Verdrahtungsstrategie des Quartus-Verdrahtungswerkzeugs berücksichtigt. Hierzu ist die statistische Analyse zahlreicher fertig verdrahteter Designs erforderlich, die eine genauere Erkenntnis darüber liefern soll: 1.) welche Platzierpositionen, die bisher in C_{II} nicht berücksichtigt wurden, für weiter entfernte Signalsenken zu bevorzugen sind und 2.) wie genau die Korrekturfaktoren zu gewichten sind. Auf diese Weise wird auch eine Festlegung der Kostenfunktion C_{II} für neuere Cyclone- und Stratix-Architekturen vereinfacht, die über zusätzliche heterogene Verdrahtungsressourcen verfügen.
- Designs mit einem überdurchschnittlichen Anteil an Carry-Chains, wie das *oc_video_dct*-Design, führen bisher zu hohen Werkzeuglaufzeiten. Daraus lässt sich schließen, dass die bisherige Strategie zum Bewegen der Carry-Chains sehr lange Zeit benötigt, um einen Bereich zu finden, der eine zu bewegende Carry-Chain aufnehmen kann. Wenn diese Strategie angepasst wird, ist mit Reduzierung der Laufzeiten bei DSP-lastigen Designs zu rechnen.
- Die Laufzeiten von MPCPlace können reduziert werden, wenn die architekturbezogenen Korrekturfaktoren der Kostenfunktion C_{II} nicht wie bisher, aus der gegenseitigen Logikzellenpositionen zeitaufwendig berechnet, sondern aus einem Konstantenarray ermittelt werden.
- Der für die Platzierung mit Simulated Annealing erforderliche Aufwand wächst überproportional mit der Anzahl der zu platzierenden Blöcke. Dieses Verfahren erweist sich vor allem bei großen Designs und bei der Platzierung der Logikzellen als sehr rechenintensiv auf. Der Einsatz eines anderen (iterativen) Platzierverfahrens kann hier eine Reduzierung der Platzierlaufzeiten bewirken. Ob und wie die Kostenfunktion C_{II} dazu angepasst werden muss, soll hierbei unbedingt untersucht werden, da nicht alle Platzierverfahren mit derart komplexen Kostenfunktionen funktionieren.
- Berücksichtigung der Signallaufzeiten in der Kostenfunktion C_{II}, um höhere Taktfrequenzen der erzeugten Platzierlösungen zu erreichen.

Während des Einsatzes der QUIP-Schnittstellen und der Altera-Quartus-Software sind einige Unzulänglichkeiten aufgefallen (siehe Abschnitte 8.3, 7.4 und 7.7). Deshalb werden an dieser Stelle folgende Verbesserungswünsche an Altera gerichtet:

- Die Übergabe der Logikzellenkoordinaten über eine QSF-Datei sollte dahingehend erweitert werden, dass dem Verdrahtungswerkzeug die Zuordnung einer LUT und eines Flipflops zu einer Logikzelle explizit mitgeteilt werden kann. Dadurch wären vermutlich noch bessere Verdrahtungsergebnisse der mit PALLAs erzeugten Platzierungen erreichbar.
- Die Veröffentlichung der Verdrahtungsgraphen für die FPGA-Architekturen. Auf diese Weise können die Strategien der Tools noch mehr an die Zielarchitekturen ausgerichtet und vermutlich bessere Ergebnisse erreicht werden.
- Die Unterstützung der lokalen Inter-Logikblockverdrahtungsressourcen durch die Quartus-Software sollte verbessert werden, um die globalen Verdrahtungsressourcen zu entlasten.
- Es ist wünschenswert, bei der Leistungsbedarfanalyse die Nutzung der Verdrahtungsressourcen mit einzubeziehen. Nach Ansicht des Autors ist das bisher nicht der Fall.

Anhang A - Bedienung

A-1 Graphische Benutzeroberfläche

Die Steuerung und Bedienung der PALLAs-Software erfolgen komplett über die graphische Tcl/Tk-Benutzeroberfläche. Das Hauptfenster der PALLAs-Benutzeroberfläche ist funktional in sechs unterschiedliche Bereiche aufgeteilt (Abbildung A-1).

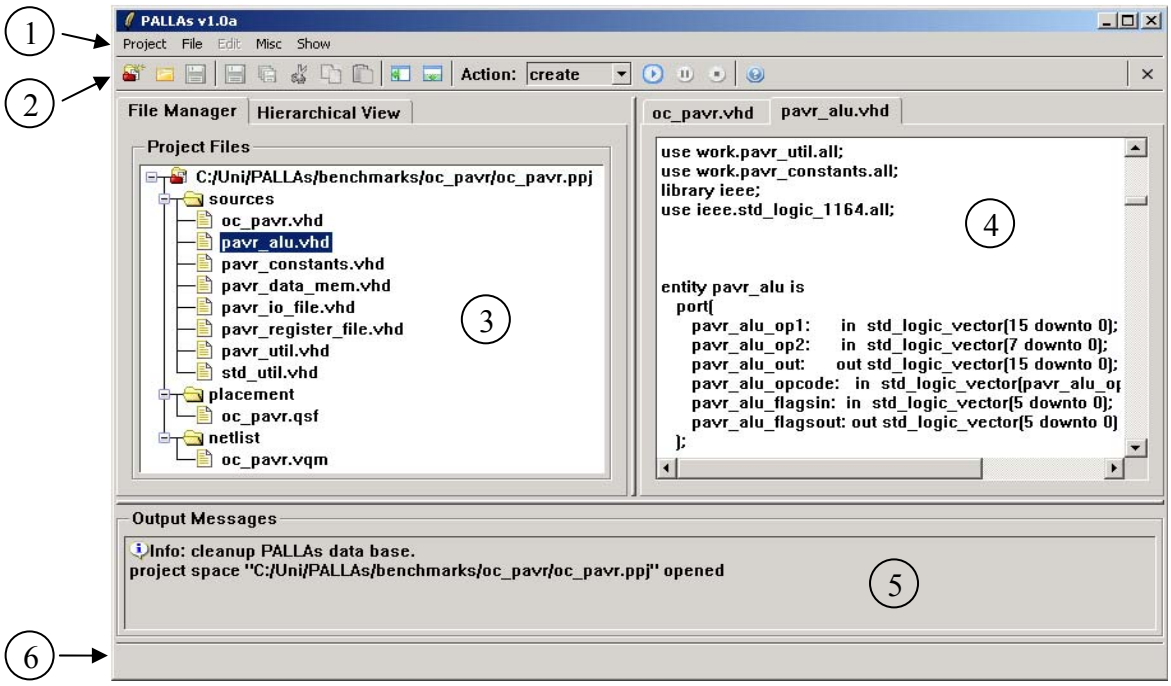


Abbildung A-1: *Unterteilung des Hauptfensters der PALLAs-Benutzeroberfläche in Hauptmenüleiste (1), Funktionstastenleiste (2), Projektbereich (3), Editorbereich (4), Textausgabe (5) und Statusleiste (6).*

In der Hauptmenüleiste (1) zusammengefasste Menüpunkte stellen Funktionen zu Projektverwaltung, Anpassen der Projektoptionen, Editieren von geöffneten VHDL-Dateien, Stapelverarbeitung mehrerer Projekte und Anpassung der Hauptfensterdarstellung zur Verfügung. Die Funktionstastenleiste (2) bietet einen schnellen Zugriff auf die häufig benötigten Funktionen und auf die Ablaufsteuerung des implementierten Entwurfsflusses. Der Projektbereich (3) zeigt die zu einem Projekt zugehörigen Dateien (File Manager) und die hierarchische Struktur (Hierarchical View) der vorliegenden Schaltungsbeschreibung unter zwei unabhängigen Reitern an. Er bietet außerdem Menüs mit kontextsensitiven Funktionen zur Projektverwaltung an. Das Editieren oder Betrachten der Dateien eines PALLAs-Projekts erfolgt im Editorbereich (4) des Hauptfensters. Die während der einzelnen Entwurfsschritte anfallenden Textmeldungen der Quartus-Werkzeuge und der DLL-Funktionen werden im Textausgabebereich (5) wiedergegeben. Die Statusleiste (6) stellt allgemeine Informationen zu einem angewählten Menüpunkt oder zu einem gerade ausgeführten Entwurfsschritt dar. Bei der Entwicklung der PALLAs-Benutzeroberfläche wurde auf möglichst intuitive und selbsterklärende Handhabung der grundlegenden Funktionen geachtet.

A-2 Entwurfsschritte

Das Auswählen der durchzuführenden Entwurfsschritte erfolgt über den Aktionsbutton der Funktionstastenleiste (Abbildung A-2). Dieser legt fest, welche Entwurfsschritte (einschließlich des ausgewählten Entwurfsschritts) ausgeführt werden sollen. Reihenfolge und Anordnung der einzelnen Entwurfsschritte innerhalb der Auswahlliste entsprechen dabei dem „top-down"-PALLAs-Entwurfsfluss (vgl. Abbildung 3-1). Durch Betätigen des direkt rechts neben dem Aktionsbutton positionierten Startknopfs wird die Ausführung der Entwurfsschritte gestartet. Von der Ausführung ausgenommen werden die Entwurfsschritte, deren Ergebnisse bereits vorliegen. So ist es möglich, die Schaltung nur ein Mal zu synthetisieren und dieselbe Netzliste bei unterschiedlichen Platzierdurchläufen zu verwenden.

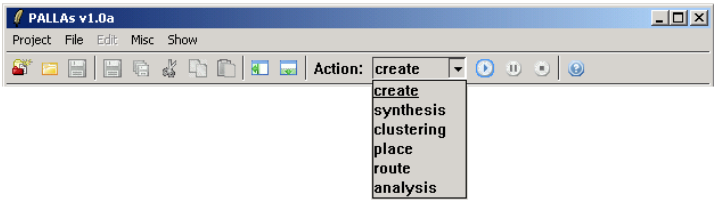


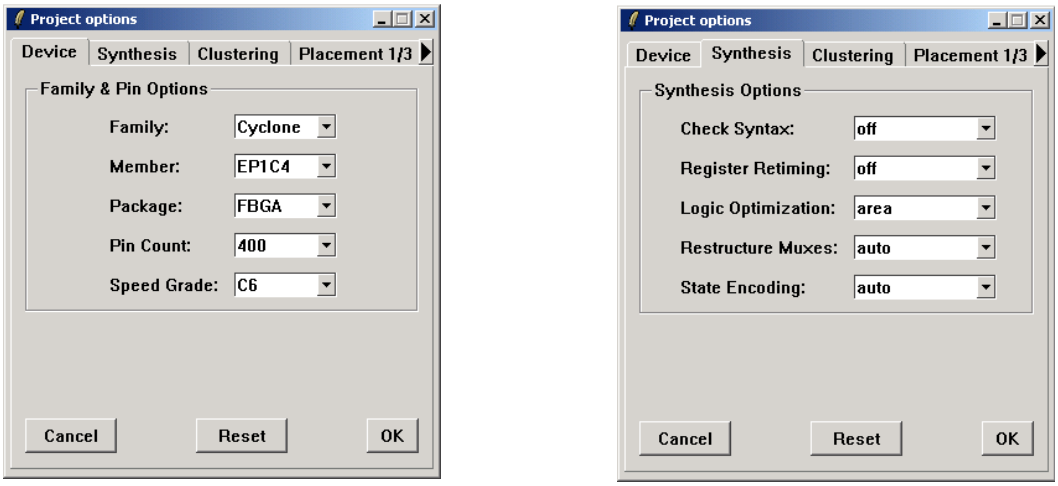
Abbildung A-2: *Auswahl der Entwurfsschritte über die Aktions-Auswahlbox der Funktionstastenleiste.*

Während einzelner Entwurfsschritte werden folgende Aktionen ausgelöst und durchführt:

- Bevor die Synthese einer Schaltungsbeschreibung mit Quartus stattfinden kann, muss ein Quartus-Projekt mit einer zunächst leeren Datenbank angelegt werden. Der Entwurfsschritt *create* übernimmt die Bausteinvorgaben und Syntheseeinstellungen und erzeugt ein Quartus-Projekt. Das erneute Ausführen dieses Entwurfsschritts ist unbedingt erforderlich, wenn die Baustein- bzw. Synthesevorgaben geändert oder neue HDL-Dateien zum PALLAs-Projekt hinzugefügt wurden. Weiterhin wird beim Entwurfsschritt *create* die Struktur der Schaltungsbeschreibung vom VHDL-Parser analysiert und danach als Baumstruktur im Projektbereich dargestellt.
- Beim Entwurfsschritt *synthesis* werden das Quartus-Synthesewerkzeug gestartet und die Synthese der Schaltungsbeschreibung durchgeführt. Nach einer erfolgreichen Synthese wird die Netzliste aus der Quartus-Datenbank extrahiert und als eine VQM-Datei dem PALLAs-Projekt hinzugefügt. Anschließend wird mit dem XML-Parser die Architekturbeschreibung der Zielarchitektur und mit dem Verilog-Parser die Netzliste eingelesen. Nach dem *synthesis*-Entwurfsschritt liegen PALLAs die internen Repräsentationen der Netzliste, der Schaltungsstruktur und der Zielarchitektur vor.
- Mit *clustering* werden das Packen der LUTs und Flipflops in die Logikzellen und das Clustern der Logikzellen durchgeführt.
- Nach dem Starten des *place*-Entwurfsschritts wird zunächst eine interne Repräsentation des Ziel-FPGA anhand der Architekturbeschreibung der FPGA-Familie erzeugt. Anschließend erfolgen Platzierung der bereits geclusterten Netzliste und Ausgabe der Platziervorgaben in eine QSF-Datei.
- Die Platziervorgaben werden von dem Quartus-Verdrahtungswerkzeug übernommen und die Verdrahtung durchgeführt, nachdem der Entwurfsschritt *route* gestartet wurde.
- Nach der Verdrahtung kann die Verdrahtungs- und Timing-Analyse durch den Entwurfsschritt *analysis* vorgenommen werden. Dabei werden Nutzung der Verdrahtungssegmente und maximale Taktfrequenz ermittelt.

A-3 Projektoptionen

Auswahl des Ziel-FPGA, Verhalten des Quartus-Synthesewerkzeugs und der implementierten Platzier- und Clustering-Algorithmen werden bei PALLAs zentral über die Projektoptionen gesteuert. Nach dem Aufrufen des Menüpunkts *Project→Options* können diese vom Benutzer festgelegt werden. Bei der Auswahl des Ziel-FPGA kann zwischen den verfügbaren Mitgliedern der Altera-Cyclone- und Stratix-Familien entschieden werden (Abbildung A-3a). Die Syntheseoptionen beinhalten die wichtigsten Einstellungen der Logikoptimierung (Abbildung A-3b).



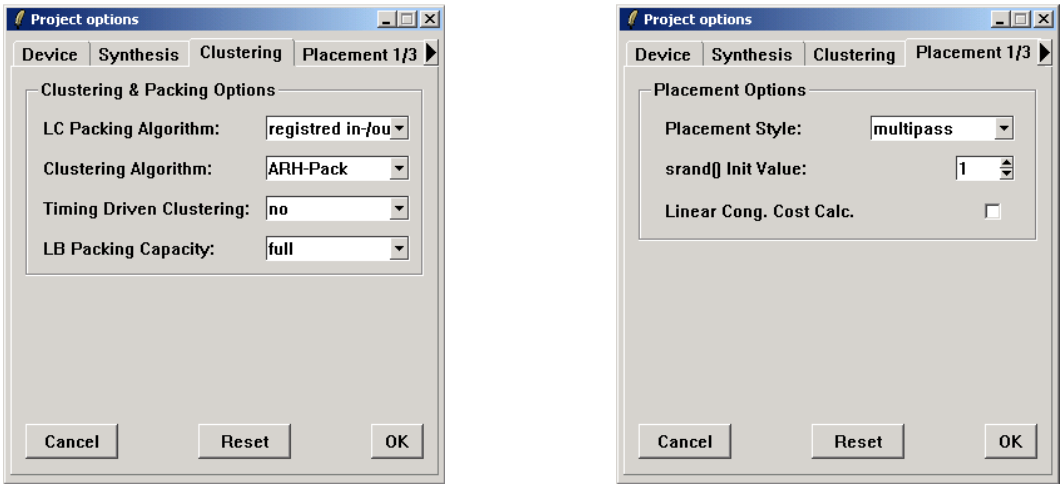
(a) (b)

Abbildung A-3: *Projektoptionen zur FPGA-Auswahl (a) und Syntheseeinstellungen (b).*

Die unter dem Reiter *Synthesis* vorgenommenen Einstellungen (Abbildung A-3b) beeinflussen direkt die Synthesestrategie des Quartus-Synthesewerkzeugs und haben folgende Auswirkungen [118]:

- Die Einstellung *Check Syntax* legt fest, ob vor der Synthese eine syntaktische Überprüfung der Schaltungsbeschreibung durchzuführen ist.
- Das Aktivieren der *Register Retiming*-Option erlaubt eine Verschiebung der vorhandenen Flipflops innerhalb der kritischen Pfade und eine Flipflopduplizierung während der Synthese. Dadurch kann unter Umständen die Tiefe der kritischen Pfade verringert werden.
- Die Vorgabe *Logic Optimization* legt fest, ob während der Synthese die Fläche, die Geschwindigkeit oder beides zu optimieren ist.
- Die Einstellung *Restructure Muxes* erlaubt dem Synthesewerkzeug, mit Hilfe von Multiplexern modellierte Busse zu identifizieren und in Hinblick auf die vorliegende Logikzellenarchitektur zu optimieren.
- Über die Vorgabe *State Encoding* wird die Art und Weise festgelegt, wie die Zustände der identifizierten Zustandsmaschinen zu kodieren sind. Möglich sind benutzerdefinierte, One-Hot und Ein-Bit-Kodierungen.

Das nächste Optionsfenster enthält Einstellungen für die Pack- und Clustering-Entwurfsschritte (Abbildung A-4a).



(a) (b)

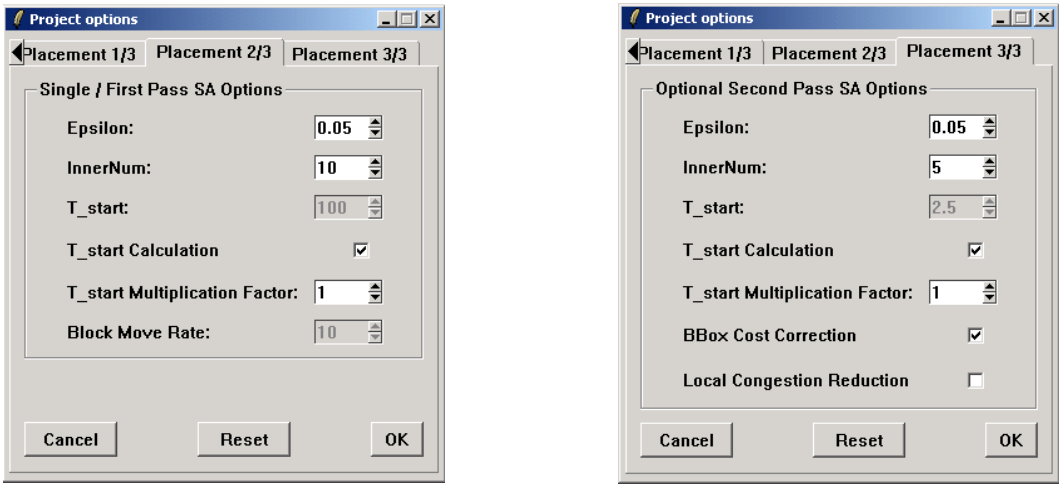
Abbildung A-4: *Projektoptionen der Clustering- (a) bzw. Platzierentwurfsschritte (b).*

Die unter dem Reiter *Clustering* (Abbildung A-4a) vorgenommenen Einstellungen bestimmen die zu verwendenden Pack- und Clustering-Verfahren:

- Als Packverfahren kann bei der Auswahlbox *LC Packing Algorithm* entweder die Standardvorgehensweise, bei der nur die nachgeschalteten bzw. auch die vorgeschalteten Flipflops berücksichtigt werden, oder das Packen der Logikzellen nach dem LFF-Prinzip bestimmt werden.
- In der Auswahlbox *Clustering Algorithm* kann einer der vier bei PALLAs implementierten Algorithmen (ARH-Pack, VPack, RPack und iRAC) als Clustering-Verfahren ausgewählt werden.
- Wird die Vorgabe *Timing Driven Clustering* gesetzt, versuchen die Clustering-Algorithmen die Läge der kritischen Pfade zu reduzieren, indem die zu einem kritischen Pfad zugehörenden Logikzellen bevorzugt in den gleichen Cluster gepackt werden.
- Über die Option *LB Packing Capacity* kann die Logikzellenkapazität der Logikblöcke auf unter zehn Logikzellen herabgesetzt werden. Die Einstellung *auto* führt zu einer Gleichverteilung der Logikzellen auf alle Logikblöcke des Ziel-FPGA.

Die Vorgaben an den Platzier-Entwurfsschritt sind über drei Optionsfenster verteilt (Abbildung A-4b und Abbildung A-5). Unter dem ersten Reiter, *Placement 1/3*, können folgende gemeinsame Einstellungen für alle verfügbaren Platzierverfahren festgelegt werden:

- Unter *Placement Style* kann der zu verwendende Platzieralgorithmus vorgegeben werden. Zur Auswahl stehen die Platzierungen mit VPR (*classic*), SCPlace (*flat*) und MPCPlace (*multipass*).
- Bei der Platzierung mit Simulated Annealing wird zur Generierung der Bewegungen eine Pseudo-Zufallsfunktion verwendet. Die von dieser Zufallsfunktion erzeugten Zufallszahlenreihen hängen von einem Initialwert ab, gleiche Initialwerte führen zu identischen Zufallszahlenreihen. Der Initialwert der Zufallsfunktion kann in dem Eingabefeld *srand() init value* vorgegeben werden.
- Beim Aktivieren der Option *linear cong. cost calc.* werden die Korrekturfaktoren für die Bounding-Boxen aller Netze in Abhängigkeit vom Fanout gemäß [58] berechnet.



(a) (b)

Abbildung A-5: *Weitere Projektoptionen des Platzierentwurfsschritts.*

Unter dem zweiten Reiter der Platzierung, *Placement 2/3* (Abbildung A-5a), sind die Abkühlungsschema-Einstellungen für den ersten (MPCPlace) bzw. den einzigen (VPR und SCPlace) SA-Platzierdurchlauf zusammengefasst:

- Die Faktoren ε und *InnerNum* (vgl. Abschnitt 6.3) bestimmen das SA-Abbruchkriterium und die Anzahl der pro Temperaturschritt auszuwertenden Bewegungen.
- Die SA-Starttemperatur, *T_start*, kann entweder fest vorgegeben oder näherungsweise bestimmt werden (vgl. Abschnitt 6.2; Schalter *T_start Calculation* in Abbildung A-5a). Die näherungsweise bestimmte Temperatur wird mit einem variablen Faktor, *T_start Multiplication Factor,* multipliziert.
- Bei der Platzierung mit SCPlace kann über *Block Move Rate* der prozentuale Anteil der Logikblockbewegungen an der Gesamtanzahl der durchzuführenden Logikzellen- und Logikblockbewegungen festgelegt werden.

Der dritte Reiter der Platzieroptionen, *Placement 3/3* (Abbildung A-5b), ist nur bei der Auswahl des MPCPlace-Platzierverfahrens verfügbar und legt die Optionen für den zweiten Platzierdurchlauf fest. Das Abkühlungsschema beim zweiten Durchlauf wird auf gleiche Weise wie beim ersten Durchlauf beeinflusst. Spezifisch für den zweiten MPCPlace-Durchlauf sind folgende Einstellungen:

- Wird die Option *BBox Cost Correction* aktiviert, werden die architekturabhängigen Korrekturfaktoren der Kostenfunktion C_{II} entsprechend der Tabelle 6-2 berechnet. Anderenfalls entspricht die beim zweiten Durchlauf verwendete Kostenfunktion der Kostenfunktion C_I.
- Die Modellierung und Reduzierung der lokalen Überbenutzung der Verdrahtungsressourcen (vgl. Abschnitt 6.9) wird über die Option *Local Congestion Reduction* aktiviert. Die Wirksamkeit der Vorgehensweise kann unter Umständen erhöht werden, wenn beim Clustering-Entwurfsschritt die Logikblöcke nicht bis an ihre Kapazitätsgrenzen gepackt werden (Option *Clustering/LB Packing Capacity)*.

A-4 Stapelverarbeitung

Das PALLAs-Werkzeug bietet die Möglichkeit, bereits angelegte Projekte über die Stapelverarbeitung den einzelnen Entwurfsschritten zu unterwerfen. Dabei werden alle Projekte des *Benchmarks*-Unterordners automatisch einer nach dem anderen geöffnet, gewünschte Entwurfsschritte durchgeführt, Ergebnisse in die Protokolldateien geschrieben und das jeweilige Projekt wieder geschlossen. Diese Automatisierung hat sich besonders bei der Festlegung und Optimierung der architekturbezogenen Korrekturfaktoren der Kostenfunktion C_{II} als sehr zeitsparend erwiesen. Um PALLAs in den Stapelverarbeitungsmodus zu versetzen, müssen zunächst die durchzuführenden Entwurfsschritte festgelegt werden. Die entsprechenden Optionen können nach dem Aufruf des Menüpunkts *Misc→Batch Options* festgelegt werden (Abbildung A-6).

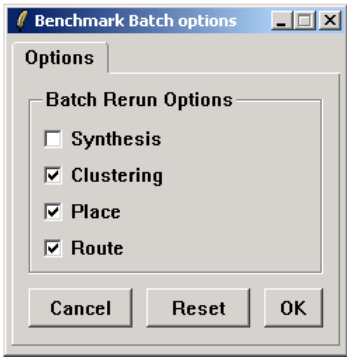


Abbildung A-6: *Optionsfenster der Stapelverarbeitung.*

Zum Starten der Stapelverarbeitung muss der Menüpunkt *Misc→Batch Run* aufgerufen werden. Nach der Stapelverarbeitung liegen die Protokolldateien in dem Ordner mit der graphischen Tcl/Tk-Benutzeroberfläche vor.

Anhang B - Dateiformate

Dieser Abschnitt enthält die Beschreibungen der PALLAs-eigenen Formate der Projekt-, Protokoll- und Kongestionsdateien, auf die Bezug in dieser Arbeit genommen wurde. Die **fett** gedruckten Zeilennummern sind nicht Bestandteil der beschriebenen Dateiformate und dienen nur der Referenzierung.

B-1 Pre-Routing-Kongestionsdatei

Die Pre-Routing-Kongestionsdatei (*pre_route.cmap*) wird vom MPCPlace-Algorithmus nach der letzten SA-Iteration angelegt, falls der Anwender die Modellierung und Reduzierung der Überbenutzung globaler Verdrahtungsressourcen selektiert hat. Sie spiegelt für jede Logikblockkachel die laut dem erzeugten Modell an dieser Kachel und in der näheren Umgebung auftretende Kongestion der Verdrahtungsressourcen wider. Sie protokolliert außerdem die Einschränkung der nutzbaren Logikblock-Eingänge.

1.	*X*	*Y*	*CONG*	*AVG*	*PINs*
2.	*1*	*1*	*171.873*	*208.944*	*26*
	...				
3.	*26*	*17*	*360.453*	*401.226*	*26*
4.	*27*	*17*	*222.261*	*365.937*	*26*

Zeile(n)		**Bedeutung**
1	-	Beschriftung der entsprechenden Spalten.
2-4	-	(1.) Angabe der X/Y-Koordinaten einer belegten FPGA-Kachel, (2.) die laut dem internen Modell an dieser Kachel auftretende Überbenutzung der globalen Verdrahtungsressourcen, (3.) der Kongestion-Durchschnittswert (gebildet aus fünf benachbarten Kacheln) und (4.) die Einschränkung der nutzbaren Logikblockeingänge.

B-2 Routing-Kongestionsdatei

Jede FPGA-Kachel der Altera-Cyclone-Architektur bildet einen Startpunkt von genau 40 R4/C4-Verdrahtungssegmenten. Während der Analyse einer erzeugten Verdrahtung extrahiert PALLAs die Zahlen über tatsächlich verwendete R4/C4-Verdrahtungssegmente aus der RCF-Datei und legt diese Information in der Routing-Kongestionsdatei (*Projektname.cmap*) ab.

1.	*X*	*Y*	*CNT*
2.	*0*	*0*	*6*
	...		
3.	*28*	*17*	*0*
4.	*28*	*18*	*0*

Zeile(n)		**Bedeutung**
1	-	Beschriftung der entsprechenden Spalten.
2-4	-	(1.) Angabe der X/Y-Koordinaten einer FPGA-Kachel und (2.) Angabe der Anzahl der tatsächlich verwendeten R4/C4-Segmente, deren Ursprung an derselben Kachel liegt.

B-3 Statistikdatei

In der Statistikdatei (*statistics.log*) werden die während der einzelnen Entwurfsschritte anfallenden Informationen über die (voraussichtliche) Nutzung der Logik- und Verdrahtungsressourcen gespeichert. Dieselben Informationen werden auch in der graphischen PALLAs-Oberfläche in dem Textausgabebereich wiedergegeben.

1. *----------*
2. *"C:/Uni/PALLAs/benchmarks/oc_pavr/oc_pavr.ppj"*
3. *----------*
4. *Info: Path length statistics (clustering is ignored):*
5. *Info: path length = 1, path count = 0*
...
6. *Info: path length = 43, path count = 8596*
7. *Info: path length = 44, path count = 1042*

8. *Info: average path length = 22, std. deviation = 3.7*
9. *Info: critical path depth: 44*

10. *Info: #lcells: 3827, thereof dual: 329; #ec: 8; #nets: 4418; #conns: 21420*
11. *Info: #is_legal_calls: 668355, #max. LAB input restriction fails 120462, #LAB wide signal restiction fails 94552*
12. *Info: #labs: 390; #m4k: 8; #nets: 2369; #conns: 9811, #loc. conns: 0*

13. *Info: #LABs[#LCELL=1] = 0 (0.00%)*
...
14. *Info: #LABs[#LCELL=9] = 36 (9.23%)*
15. *Info: #LABs[#LCELL=10] = 341 (87.44%)*
16. *Info: avg. #LCELLs/#LABs = 9.81*

17. *Info: #LABs[#lab_wide=0] = 167 (42.82%)*
...
18. *Info: #LABs[#lab_wide=5] = 109 (27.95%)*
19. *Info: #LABs[#lab_wide=6] = 24 (6.15%)*
20. *Info: total LAB wide signal usage = 1002*

21. *Info: #LABs[#conn_cnt=1] = 0 (0.00%)*
...
22. *Info: #LABs[#conn_cnt=29] = 15 (4.40%)*
23. *Info: #LABs[#conn_cnt=30] = 9 (2.64%)*
24. *Info: total LAB pin connection usage = 9689*
25. *Info: avg. LAB pin connection usage = 24.84*

26. *Info: final placement costs = 22726*
27. *Info: #labs: 390; #m4k: 8; #nets: 2369; #conns: 9811, #loc. conns: 399*
28. *...*
29. *Info: C4 segment count = 3801*
30. *Info: R4 segment count = 3963*
31. *Info: DIRECT_LINK segment count = 1051*

Zeile(n)		Bedeutung
2	-	PALLAs-Projektname.
4-7	-	Pfadlängenstatistik der ungeclusterten Netzliste.
8	-	Durchschnittliche Pfadtiefe und Standardabweichung des Mittelwerts.
9	-	Tiefe des kritischen Pfads.
10	-	Anzahl der Logikzellen, der voll belegten Logikzellen, der RAM-Grundzellen, der Netze und Gesamtfanout dieser Netze nach dem Packen.
11	-	Angabe, wie oft beim Clustern die Zusammensetzung der Logikblöcke überprüft wurde und wie oft diese an der Eingangsrestriktion bzw. den Einschränkungen der logikblockweiten Flipflop-Steuersignale gescheitert ist.
12	-	Anzahl der Logikblöcke, RAM-Blöcke, Netze, Gesamtfanout dieser Netze nach dem Clustern.
13-15	-	Statistik über die Belegung der Logikblöcke mit den Logikzellen.
16	-	Durchschnittliche Belegung der Logikblöcke mit den Logikzellen.
17-19	-	Statistik über die Nutzung der logikblockweiten Flipflop-Steuersignale.
20	-	Insgesamt benutzte logikblockweite Flipflop-Steuersignale.
21-23	-	Statistik über die Nutzung der Logikblockanschlüsse.
24	-	Gesamtanzahl der benutzten Logikblockanschlüsse.
25	-	Durchschnittliche Nutzung der Logikblockanschlüsse.
26	-	Gesamtkosten der Platzierung.
27	-	Anzahl der Logikblöcke, RAM-Blöcke, Netze, Gesamtfanout dieser Netze und Gesamtzahl der möglichen lokalen Inter-Logikblockverbindungen nach dem Platzieren.
28	-	Bei MPCPlace wiederholen sich die Statistiken entsprechend den Zeilen 13-27 für den zweiten Platzierdurchlauf.
29-31	-	Anzahl der benutzen C4/R4-Segmente und lokaler Inter-Logikblocksegmente nach der Platzierung.

B-4 PALLAs-Projektdatei

In der PALLAs-Projektdatei (*Projektname.ppj*) werden die gesamten Projekt-Einstellungen eines PALLAs-Projekts gespeichert.

1. *BmoveRate?10*
2. *InnerNum1?10*
3. *InnerNum2?5*
4. *LB_packing_capacity?full*
5. *LC_clustering_alg?ARH-Pack*
6. *T1calc?1*
7. *T1mult?1*
8. *T1start?100*
9. *T2calc?0*
10. *T2mult?5*
11. *T2start?2.5*
12. *arch_cost_red?1*
13. *design_flow?multipass*
14. *dev_family?Cyclone*
15. *dev_member?EP1C4*
16. *dev_package?FBGA*
17. *dev_pin_count?400*
18. *dev_speed_grade?C6*
19. *done_clustering?0*
20. *done_create?1*
21. *done_place?0*
22. *done_route?0*
23. *done_synthesis?1*
24. *epsilon1?0.05*
25. *epsilon2?0.05*
26. *hierarchy?oc_pavr pavr_rf_instance1(pavr_rf) pavr_iof_instance1(pavr_iof) pavr_dm_instance1(pavr_dm) pavr_alu_instance1(pavr_alu)*
27. *lin_cong?0*
28. *loc_cong_red?0*
29. *netlist?C:/Uni/PALLAs/benchmarks/oc_pavr/oc_pavr.vqm*
30. *pack_alg?registred in-/output LFF*
31. *placement?C:/Uni/PALLAs/benchmarks/oc_pavr/oc_pavr.qsf*
32. *sources?C:/Uni/PALLAs/benchmarks/oc_pavr/oc_pavr.vhd*
33. *sources?C:/Uni/PALLAs/benchmarks/oc_pavr/pavr_alu.vhd*
34. *sources?C:/Uni/PALLAs/benchmarks/oc_pavr/pavr_constants.vhd*
35. *sources?C:/Uni/PALLAs/benchmarks/oc_pavr/pavr_data_mem.vhd*
36. *sources?C:/Uni/PALLAs/benchmarks/oc_pavr/pavr_io_file.vhd*
37. *sources?C:/Uni/PALLAs/benchmarks/oc_pavr/pavr_register_file.vhd*
38. *sources?C:/Uni/PALLAs/benchmarks/oc_pavr/pavr_util.vhd*
39. *srand_init_value?1*
40. *syn_check_syntax?off*
41. *syn_optimisation?area*
42. *syn_reg_retiming?off*
43. *syn_restuct_mux?auto*
44. *syn_state_enc?auto*
45. *t_driven_clustering?no*

Zeile(n)		Bedeutung
1	-	Prozentualer Anteil der zu bewegenden Logikblöcke an der Gesamtzahl der Bewegungen bei SCPlace.
2-3	-	*InnerNum*-Faktor des SA-Schemas für den ersten bzw. zweiten Durchlauf.
4	-	Kapazitätsbegrenzung der Logikblöcke.
5	-	Clustering-Algorithmus.
6-8	-	Parameter zur Steuerung der Temperaturbestimmung beim ersten Durchlauf.
9-11	-	Parameter zur Steuerung der Temperaturbestimmung beim zweiten Durchlauf.
12	-	Bei einer ‚1' wird die Berechnung der architekturabhängigen Korrekturfaktoren der Kostenfunktion C_{II} durchgeführt.
13	-	Platzier-Algorithmus.
14-18	-	Ziel-FPGA, vgl. auch Abbildung A-3a.
19-23	-	Bei einer ‚1' wurde der entsprechende Entwurfsschritt bereits durchgeführt und muss nicht wiederholt werden.
24-25	-	Parameter ε des SA-Schemas für den ersten und zweiten Durchlauf.
26	-	Durch den VHDL-Parser ermittelte strukturelle Hierarchie der Schaltungsbeschreibung.
27	-	Bei einer ‚1' werden die Fanout-abhängigen BBox-Korrekturfaktoren nach RISA [58] bereichnet.
28	-	Bei einer ‚1' wird die Modellierung und Reduzierung der Verdrahtungs-Kongestion durchgeführt.
29	-	Name und Pfad der Netzlistendatei.
30	-	Packalgorithmus.
31	-	Name und Pfad der QSF-Datei.
32-38	-	In dem PALLAs-Projekt enthaltenen Schaltungsbeschreibungen.
39	-	Initialisierungswert der Zufallsfunktion.
40-44	-	Syntheseeinstellungen, vgl. auch Abbildung A-3b.
45	-	Bei einer ‚1' wird beim Clustering-Entwurfsschritt versucht, die Tiefe der kritischen Pfade zu reduzieren.

B-5 Sitzungsprotokoll

Das Sitzungsprotokoll dokumentiert die während einer PALLAs-Sitzung gestarteten und ausgeführten Aktionen, deren Aufrufparameter und die Ausführungszeiten. Dieselben Informationen werden auch in der graphischen PALLAs-Oberfläche im Textausgabebereich wiedergegeben.

1. *Info: cleanup PALLAs data base.*
2. *project space "C:/Uni/PALLAs/benchmarks/oc_pavr/oc_pavr.ppj" opened*
3. *project space "C:/Uni/PALLAs/benchmarks/oc_pavr/oc_pavr.ppj" saved*

4. *Info:* **********
5. *Info:* * *PALLAs - XML Architecture Parser* *
6. *Info:* * *Copyright (C) 2006, Valerij Matrose, TU Berlin* *
7. *Info:* * *seemann@cs.tu-berlin.de* *
8. *Info:* **********
9. *Info: parsing XML architecture file, it may take a few seconds!*
10. *Info: done!*
11. *Info: total time 0.197 seconds*
12. *Info:* **********

13. *Info:* **********
14. *Info:* * *PALLAs - VQM Netlist Parser* *
15. *Info:* * *Copyright (C) 2006, Valerij Matrose, TU Berlin* *
16. *Info:* * *seemann@cs.tu-berlin.de* *
17. *Info:* **********
18. *Info: parsing VQM netlist file, it may take a few seconds!*
19. *Info: logic cells sorted in alphanumerical order...done*
20. *Info: done!*
21. *Info: total time 0.790 seconds*
22. *Info:* **********

23. *Info:* **********
24. *Info:* * *PALLAs - Cell Packer* *
25. *Info:* * *Copyright (C) 2006, Valerij Matrose, TU Berlin* *
26. *Info:* * *seemann@cs.tu-berlin.de* *
27. *Info:* **********
28. *Info: using LFF-packing for registred in- and outputs*
29. *Info: using ARH-Pack clustering algorithm*
30. *Info: timing-driven clustering off*
31. *Info: logic block capacity limit is 10*
32. *Info: packing logic cells & RAMs into clusters ...*
33. *Info: done!*
34. *Info: total time 2.169 seconds*
35. *Info:* **********

36. *Info: ***
37. *Info: * PALLAs - Cell Placer *
38. *Info: * Copyright (C) 2006, Valerij Matrose, TU Berlin *
39. *Info: * seemann@cs.tu-berlin.de *
40. *Info: ***
41. *Info: place cells into target FPGA ...*
42. *Info: multipass design flow*
43. *Info: random function initialisation value is 1*
44. *Info: Simulated annealing engine first/single pass configuration:*
45. *Info: epsilon is 0.050, InnerNum is 10, block move rate is 0%,*
start temperature will be calculated and multiplied with 1.0
46. *Info: Simulated annealing engine second configuration:*
47. *Info: epsilon is 0.050, InnerNum is 5, start temperature is 2.500*
48. *Info: Simulated Annealing placement of clusters in target device ...*
49. *Info: bounding box costs = 47231.0; timing costs = 18742.6; congestion costs = 16624.3*
50. *Info: moves per temp = 29275, initial costs = 47231*
51. *Info: max. pos. cost difference = 1400; max. neg. cost difference = -633*
52. *Info: initial SA temperature = 39.0015*
53. *Info: bounding box costs = 36678.0; timing costs = 17006.7; congestion costs = 9745.6*
54. *Info: new weighted costs = 36678.0; SA temp = 39.0015; Rlim = 34;*
taken moves = 13908; proposed moves with neg. delta_C = 6703
...
55. *Info: bounding box costs = 22728.0; timing costs = 15014.8; congestion costs = 3125.0*
56. *Info: new weighted costs = 22728.0; SA temp = 0.4984; Rlim = 2;*
taken moves = 2175; proposed moves with neg. delta_C = 139
57. *Info: restore solution with lowest costs = 22728*
58. *Info: accept at last run only moves with neg. delta_C*
59. *Info: bounding box costs = 22726.0; timing costs = 15015.1; congestion costs = 3123.2*
60. *Info: done!*
61. *...*
62. *Info: total time 1685.693 seconds*
63. *Info: ***

Zeile(n)		Bedeutung
1	-	Interne Datenobjekte werden gelöscht.
2	-	Neues Projekt wird geöffnet.
4-12	-	Ausführung des XML-Architekturdateiparsers; ein Architekturobjekt wird erzeugt.
13-22	-	Ausführung des Netzlistenparsers; ein Netzlistenobjekt wird erzeugt.
23-35	-	Ausführung der Entwurfsschritte Packen und Clustering der Logikzellen.
28	-	Verwendeter Algorithmus zum Packen der LUTs und FFs in die Logikzellen.
29	-	Verwendeter Clustering-Algorithmus.
30	-	Signallaufzeitenoptimierung bei dem Clustering-Algorithmus.
31	-	Kapazitätsbegrenzung der Logikblöcke.
36-62	-	Ausführung des Platzier-Entwurfsschrittes.
42	-	Verwendeter Platzieralgorithmus.
43	-	Initialisierungswert der Zufallsfunktion.
44-45	-	Parameter des Abkühlungsschemas für den ersten bzw. einzigen Durchlauf.
46-47	-	Parameter des Abkühlungsschemas für den zweiten MPCPlace-Durchlauf.
49	-	Auflistung der Teilkosten der zufälligen Initialplatzierung.
50	-	Anzahl der pro Temperaturschritt auszuwertenden Bewegungen, initiale BBox-Kosten der Platzierung.
51-52	-	Bestimmung der Starttemperatur.
53-56	-	Iterationen der SA-Platzierung. Zu jedem Iterationsschritt werden ermittelte Kosten, SA-Temperatur, R_{lim}-Wert, Anzahl durchgeführter Bewegungen und Anzahl vorgeschlagener Bewegungen mit negativen Kosten ausgegeben.
57	-	Die bisher ermittelte Platzierung mit den geringsten Platzierkosten wird wiederhergestellt.
58-59	-	Bei der letzten Iteration werden nur Bewegungen mit negativen Kosten akzeptiert.
61	-	Zweiter Platzierdurchlauf bei MPCPlace-Platzierverfahren.
11, 21, 34, 62	-	Laufzeiten der einzelnen Entwurfsschritte.

Literatur

Veröffentlichungen

[01] *David Marsh;* Low-cost kits: the new FPGA-designer trend; 23 Nobvember 2006, http://www.edn.com/article/CA6391429.html

[02] *F. Kesel, R. Bartholomä;* Entwurf von digitalen Schaltungen und Systemen mit HDLs und FPGAs; Oldenburg-Verlag, 2006

[03] *Michael Santarini;* High noon for FPGAs: Low-cost-versus- high-end showdown; 5. November 2007, http://www.edn.com/article/CA6495296.html

[04] *P. Kannan, D. Bhatia;* Estimating pre-placement FPGA interconnection requirements; Proceedings of the 17th International Conference on VLSI Design, 2004

[05] *Ketan Padalia, Ryan Fung, Mark Bourgeault, Aaron Egier, Jonathan Rose;* Automatic transistor and physical design of FPGA tiles from an architectural specification; February 2003, FPGA '03: Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays

[06] *Ian Kuon, Aaron Egier, Jonathan Rose;* Design, layout and verification of an FPGA using automated tools; February 2005 FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays

[07] *Andy Yan, Rebecca Cheng, Steven J. E. Wilton;* On the sensitivity of FPGA architectural conclusions to experimental assumptions, tools, and techniques; February 2002, FPGA'02: Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays

[08] *Rose, J.; Brown, S.;* Flexibility of interconnection structures for field-programmable gate arrays; Solid-State Circuits, IEEE Journal of Volume 26, Issue 3, March 1991

[09] *Yao-Wen Chang, D. F. Wong, C. K. Wong;* Universal switch modules for FPGA design; January 1996, ACM Transactions on Design Automation of Electronic Systems (TODAES), Volume 1, Issue 1

[10] *Yao-Wen Chang, Kai Zhu, Guang-Ming Wu, D. F. Wong, C. K. Wong;* Analysis of FPGA/FPIC switch modules; January 2003, ACM Transactions on Design Automation of Electronic Systems (TODAES), Volume 8 Issue 1

[11] *Hongbing Fan, Jiping Liu, Yu-Liang Wu, Chak-Chung Cheung;* On optimum switch box designs for 2-D FPGAs; June 2001, Proceedings of the 38th conference on Design automation

[12] *Yao-Wen Chang; Wong, D.F.; Wong, C.K.;* FPGA global routing based on a new congestion metric; ICCD'95. Proceedings., 1995 IEEE International Conference on 2-4 Oct. 1995 Page(s):372 – 378

[13] *Satish Sivaswamy, Gang Wang, Cristinel Ababei, Kia Bazargan, Ryan Kastner, Eli Bozorgzadeh;* HARP: hard-wired routing pattern FPGAs; Feb. 2005, Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays

[14] *V. Betz, J. Rose, A. Marquardt;* Architecture and CAD for Deep-Submicron FPGAs; Kluwer Academic Publishers, 1999

[15] *Rose, J.; Francis, R.J.; Chow, P.; Lewis, D.;* The effect of logic block complexity on area of programmable gate arrays; Custom Integrated Circuits Conference, 1989., Proceedings of the IEEE 1989, 15-18 May 1989

[16] *Elias Ahmed, Jonathan Rose;* The effect of LUT and cluster size on deep-submicron FPGA performance and density; February 2000 FPGA '00: Proceedings of the 2000 ACM/SIGDA eighth international symposium on Field programmable gate arrays

[17] *Small, C.;* Shrinking devices put the squeeze on system packaging; EDN 39, 4 Feb. 1994, 41–46.

[18] *Elias Ahmed, Jonathan Rose;* The effect of LUT and cluster size on deep-submicron FPGA performance and density; Very Large Scale Integration (VLSI) Systems, IEEE Transactions on Volume 12, Issue 3, March 2004 Page(s):288-298

[19] *Haixia Gao; Yintang Yang; Xiaohua Ma; Gang Dong;* Analysis of the effect of LUT size on FPGA area and delay using theoretical derivations; Quality of Electronic Design, 2005. ISQED 2005. Sixth International Symposium on 21-23, March 2005

[20] *Jianshe He; Rose, J.;* Advantages of heterogeneous logic block architecture for FPGAs; Custom Integrated Circuits Conference, 1993, Proceedings of the IEEE 1993 9-12 May 1993 Page(s):7.4.1-7.4.5

[21] *Sinan Kaptanoglu, Greg Bakker, Arun Kundu, Ivan Corneillet, Ben Ting;* A new high density and very low cost reprogrammable FPGA architecture; February 1999, FPGA'99: Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays

[22] *Chow, P.; Soon Ong Seo; Rose, J.; Chung, K.; Paez-Monzon, G.; Rahardja, I.;* The design of an SRAM-based field-programmable gate array. I. Architecture; Very Large Scale Integration (VLSI) Systems, IEEE Transactions on Volume 7, Issue 2, June 1999 Page(s):191-197

[23] *Britton, B.K.; Oh, Y.T.; Oswald, W.; Nguyen, H.T.; Singh, S.; Lee, G.; Wai-Bor Leung; Spivak, C.; Steward, J.; Chen, C.T.;* Second generation ORCA architecture utilizing 0.5 µm process enhances the speed and usable gate capacity of FPGAs; ASIC Conference and Exhibit, 1994. Proceedings., Seventh Annual IEEE International 19-23 Sept. 1994 Page(s):474-478

[24] *Siozios, K.; Soudris, D.;* Wire Segment Length and Switch Box Co-Optimization for FPGA Architectures; Field Programmable Logic and Applications, 2006. FPL'06, International Conference on 28-30 Aug. 2006 Page(s):1-4

[25] *V. Betz and J. Rose;* On Biased and Non-Uniform Global Routing Architectures and CAD Tools for FPGAs; CSRI Tech. Rep. #358, Dept. of ECE, University of Toronto, 1996

[26] *Actel Corporation;* Total System Power – Understanding the power Profile of FPGAs

[27] *Vaughn Betz, Jonathan Rose;* FPGA routing architecture: segmentation and buffering to optimize speed and density; February 1999, FPGA '99: Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays

[28] *Mike Sheng, Jonathan Rose;* Mixing buffers and pass transistors in FPGA routing architectures; February 2001 FPGA '01: Proceedings of the 2001 ACM/SIGDA ninth international symposium on Field programmable gate arrays

[29] *David Lewis, Vaughn Betz, David Jefferson, Andy Lee, Chris Lane, Paul Leventis, Sandy Marquardt, Cameron McClintock, Bruce Pedersen, Giles Powell, Srinivas Reddy, Chris Wysocki, Richard Cliff, Jonathan Rose;* The Stratix routing and logic architecture; February 2003 FPGA '03: Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays

[30] *Massoud Pedram;* Power minimization in IC design: principles and applications; Jan.'96 ACM Transactions on Design Automation of Electronic Systems, Volume 1, Issue 1

[31] *P. Schumacher, M. Paluszkiewicz, R. Ballantyne, and R.Turney;* An Efficient JPEG2000 Encoder Implemented on a Platform FPGA; SPIE Annual Meeting, 2003.

[32] *Vaughn Betz, Jonathan Rose;* Using architectural "families" to increase FPGA speed and density; February 1995, FPGA'95: Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays

[33] *H. S. Jones Jr., P. R. Nagle, and H. T. Nguyen;* A comparison of standard cell and gate array implementations in a common CAD system. In IEEE 1986 CICC pages 228-232.

[34] *S. D. Brown, R. Franis, J. Rose, and Z. Vranesi;* Field-programmable gate arrays; Kluwer Academic Publishers, 1992.

[35] *Ian Kuon, Jonathan Rose;* Measuring the gap between FPGAs and ASICs; February 2006, FPGA '06: Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays.

[36] *Manoj A. Kumar, Jayaram Bobba, V. Kamakoti;* MemMap: Technology Mapping Algorithm for Area Reduction in FPGAs with Embedded Memory Arrays Using Reconvergence Analysis; February 2004, DATE '04: Proceedings of the conference on Design, automation and test in Europe - Volume 2.

[37] *R. Brayton and C. McMullen;* The Decomposition and Factorization of Boolean Expressions; Proceedings of the International Symposium on Circuits and Systems, 1982, pp. 49-54.

[38] *Amit Singh, Malgorzata Marek-Sadowska;* Efficient circuit clustering for area and power reduction in FPGAs; Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays.

[39] *Jason Cong, Yuzheng Ding;* Combinational logic synthesis for LUT based field programmable gate arrays; April 1996, ACM Transactions on Design Automation of Electronic Systems (TODAES), Volume 1, Issue 2

[40] *John Lockwood;* Logik Synthesis for Field Programmable Gate Array (FPGA) Technology; VLSI Handbook, second Edition, CRC Press 2006

[41] *Jason Cong; Yuzheng Ding;* On Area/Depth Trade-off in LUT-Based FPGA Technology Mapping; Design Automation, 1993. 30th Conference on 14-18 June 1993

[42] *Alexander Marquardt, Vaughn Betz, Jonathan Rose;* Using cluster-based logic blocks and timing-driven packing to improve FPGA speed and density; February 1999 FPGA'99: Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays.

[43] *Vaughn Betz, Jonathan Rose;* Cluster-based logic blocks for FPGAs: area-efficiency vs. input sharing and size; Custom Integrated Circuits Conference, 1997., Proceedings of the IEEE 1997, 5-8 May 1997 Page(s):551-554.

[44] *Bozorgzadeh, E.; Ogrenci-Memik, S.; Sarrafzadeh, M.;* RPack: routability-driven packing for cluster-based FPGAs; Proceedings of the Asia and South Pacific – Design Automation Conference 30 Jan.-2 Feb. 2001.

[45] *Lin, J.Y.; Deming Chen; Cong, J.;* Optimal simultaneous mapping and clustering for FPGA delay optimization; Design Automation Conference, 2006 43rd ACM/IEEE, 24-28 July 2006 Page(s):472 – 477.

[46] *E. Bozorgzadeh, S. O. Memik, X. Yang, M. Sarrafzadeh;* Routability-driven packing: Metrics and algorithms for cluster-based FPGAs; 2004, Journal of Circuits Systems and Computers. 13 (1), pp. 77-100.

[47] *K. Shahookar, P. Mazumder;* VLSI cell placement techniques ; Source ACM Computing Surveys (CSUR) archive, Volume 23, Issue 2 (June 1991), Pages: 143-220.

[48] *Sadiq M. Sait, Habib Youssef;* VLSI Physical Design Automation: Theory and Practice; World Scientific Publishing Company; 1st edition (November 15, 1999).

[49] *S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi;* Optimization by Simulated Annealing; Science, 13 May 1983, Volume 220, Number 4598, Pages: 671-681.

[50] *R. A. Rutenbar;* Simulated annealing algorithms an overview; IEEE Circuits and Devices Magazine, January 1989.

[51] *Huang, M.D., Romeo, F., Sangiovanni-Vincentelli, A.L.;* An efficient general cooling schedule for simulated annealing; Proceedings of IEEE International Conference on Computer-Aided Design, Santa Clara, 1986, (pp. 381–384).

[52] *Rose J.S., Klebsch W., Wolf J.;* Temperature measurement and equilibrium dynamics of simulated annealing placements; IEEE Transactions on CADICS, 1990.

[53] *Markus Wannemacher;* Das FPGA-Kochbuch; Thomson Publishing Company, 1998

[54] *Yaska Sankar, Jonathan Rose;* Trading Quality for Compile Time: Ultra-Fast Placement for FPGAs; Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays; 1999, Pages 157-166.

[55] *Jimmy Lam, Jean-Marc Delosme;* Performance of a new annealing schedule; Proceedings of the 25th ACM/IEEE conference on Design automation, 1988.

[56] *W. Swartz and C. Sechen;* New Algorithms for the Placement and Routing of Macro Cells; ICCAD, 1990, pp. 336-339.

[57] *W. Sun and C. Sechen;* Efficient and Effective Placement for Very Large Circuits; IEEE Transactions on Computer-Aided Design, vol. 14, no. 3, Mar. 1995, pp. 349-359.

[58] *C. L. E. Chang;* RISA: Accurate and Efficient Placement Routability Modeling; Proceedings of the International Conference on Computer Aided Design, November 1994, Pages 690-695

[59] *J. Swartz, V. Betz and J. Rose;* A Fast Routability-Driven Router for FPGAs; Proceedings of the ACM International Symposium on FPGAs, February 1998, Pages 140-149.

[60] *Landman B., Russo R.;* On a pin versus block relationship for partitions of logic graphs; IEEE Trans. Comput. C-20, 12. December. 1971, Pages 1469–1479.

[61] H. Van Marck, D. Stroobandt, J. Van Campenhout; Towards an extension of Rent's rule for describing local variations in interconnect complexity; Proceedings 4th International Symposium for Young Computer Scientists; 1995, Pages 136-141.

[62] *M. Feuer;* Connectivity of Random Logic; IEEE Transactions on Computers, vol. 31, no. 1, Pages 29-33, January 1982.

[63] S.D. Brown; Routing Algorithms and Architectures for Field-Programmable Gate Arrays; Ph.D. Thesis, University of Toronto, January 1992.

[64] *Michael Hutton;* Interconnect prediction for programmable logic devices; Proceedings of the 2001 international workshop on System-level interconnect prediction; Pages 125-131.

[65] *Kannan, P.; Bhatia, D.;* Interconnect estimation for FPGAs; IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems; Volume 25, Issue 8, August 2006, Pages 1523-1534.

[66] *David Yeager, Darius Chiu, Guy Lemieux;* Congestion estimation and localization in FPGAs: a visual tool for interconnect prediction; SLIP '07: Proceedings of the 2007 international workshop on System level interconnect prediction .

[67] *Valavan Manohararajah, Gordon R. Chiu, Deshanand P. Singh, Stephen D. Brown;* Difficulty of predicting interconnect delay in a timing driven FPGA CAD flow; SLIP'06: Proceedings of the 2006 international workshop on System-level interconnect prediction.

[68] *Vaughn Betz, Jonathan Rose;* VPR: A new packing, placement and routing tool for FPGA research; Proceedings of the 7th International Workshop on Field-Programmable Logic and Applications table of contents; 1997 Pages: 213-222.

[69] *Stefan H. Thomke;* Experimentation matters: Unlocking the potential of new technologies for innovation; Harvard Business School 2003.

[70] *Jason Cong, Tim Kong, Shinnerl, J.R., Min Xie, Xin Yuan;* Large-scale circuit placement: gap and promise; International Conference on Computer Aided Design; 9-13 Nov. 2003, Pages 883-890.

[71] Chen, G. Cong, J.; Simultaneous Timing Driven Clustering and Placement for FPGAs Lecture Notes in Computer Science; 2004, ISSU 3203, pages 158-167.

[72] *Gang Chen, Jason Cong;* Simultaneous placement with clustering and duplication; July 2006, ACM Transactions on Design Automation of Electronic Systems (TODAES), Volume 11, Issue 3.

[73] *Andrew B. Kahng;* Classical Floorplanning Harmful?; Proc. ACM Intl. Symp. On Physical Design, April 2000, Pages 207-213.

[74] *Maogang Wang, Abhishek Ranjan, Salil Raje;* Multi-Million Gate FPGA Physical design Challenges; Proc. ICCAD'03, November 11-13, 2003.

[75] *R. Hitchcock, G. Smith and D. Cheng;* Timing Analysis of Computer-Hardware; IBM Journal of Research and Development, Jan. 1983, Pages 100-105.

[76] *Jun Yuan, Sheqin Dong, Xianlong Hong, Yuliang Wu;* LFF Algorithm for Heterogeneous FPGA Floorplanning; Proceedings of the Asia and South Pacific Design Automation Conference 2005, pp. 1123-1126, 2005.

[77] Nagel, L. W, and Pederson, D. O.; SPICE (Simulation Program with Integrated Circuit Emphasis); Memorandum No. ERL-M382, University of California, Berkeley, 1973.

[78] *Rajeev Jayaraman;* Physical design for FPGAs; ISPD '01: Proceedings of the 2001 international symposium on Physical design.

[79] *Seokjin Lee, Hua Xiang, D. F. Wong, Richard Y. Sun;* Wire type assignment for FPGA routing; FPGA '03: Proceedings of the 2003 ACM/SIGDA eleventh international symposium on Field programmable gate arrays.

[80] *Larry McMurchie, Carl Ebeling;* PathFinder: a negotiation-based performance-driven router for FPGAs; FPGA '95: Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays.

[81] Catherine L. Zhou, Wai-Chung Tang, Wing-Hang Lo, Yu-Liang Wu; How much can logic perturbation help from netlist to final routing for FPGAs; June 2007, DAC '07: Proceedings of the 44th annual conference on Design automation.

[82] *S.Yang*; Logic Synthesis and Optimization Benchmarks, Version 3.0; 1991, Tech. Report, Microelectronics Center of North Carolina.

[83] *Altera Corporation;* Benchmark Designs For The Quartus University Interface Program (QUIP); Version 1.0, June 2005.

[84] *Altera Corporation;* Benchmarking Using the Quartus University Interface Program (QUIP); Version 1.0, June 2005.

[85] *Altera Corporation;* Guidance for Accurately Benchmarking FPGAs; Version 1.1, October 2007.

[86] *Naveed A. Sherwani;* Algorithms for VLSI Physical Design Automation; 2002 Kluwer Academic Publishers

[87] *John K. Ousterhout;* TCL and the TK Toolkit; 11. May 1994, Addison-Wesley

[88] *D. Gajski, R. Kuhn;* Guest Editor's Introduction: New VLSI Tools; IEEE Computer, Nr. 12, Pages 11-14, 1983

[89] *Taraneh Taghavi, Soheil Ghiasi, Abhishek Ranjan, Salil Raje, Majid Sarrafzadeh;* Innovate or Perish: FPGA Physical Design; ISPD'04, April 18-21, 2004

[90] *Dmitrij Koch;* Verzögerungszeitenbestimmung der unterschiedlichen Verdrahtungssegmentarten der Altera FPGAs und Überprüfung der Ergebnisse mittels Netzlistenanalyse der LEON2-CPU; Diplomarbeit, TU Berlin, 31.08.2006

[91] *James Kao, Anantha Chandrakasan, Dimitri Antoniadis;* Transistor sizing issues and tool for multi-threshold CMOS technology; Proceedings of the 34th annual conference on Design automation; United States, Pages: 409 - 414, Year of Publication: 1997

[92] *Altera Corporation;* Stratix IV FPGA Power Management and Advantages; May 2008, ver. 1.0

[93] *Shawn Malhotra, Terry P. Borer, Deshanand P. Singh, Stephen D. Brown;* The Quartus University Interface Program: Enabling Advanced FPGA Research; IEEE International Conference on Field-Programmable Technology, 2004

[94] *Vijay Degalahal, Tim Tuan;* Methodology for high level estimation of FPGA power consumption; ASP-DAC '05: Proceedings of the 2005 conference on Asia South Pacific design automation, January 2005

[95] *Julien Lamoureux, Steven J. E. Wilton;* FPGA clock network architecture: flexibility vs. area and power; FPGA '06: Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays, February 2006

[96] *Ed Sperling;* FPGA or ASIC?; Electronic News 06.06.2005; http://www.edn.com/index.asp?layout=articlePrint&articleID=CA606628

[97] *Altera Corporation;* Stratix vs. Virtex-II Pro FPGA Performance Analysis; 2004

[98] *Mike Hutton, Richard Yuan, Jay Schleicher, Gregg Baeckler, Sammy Cheung, Kar Keng Chua, Hee Kong Phoo;* A methodology for FPGA to structured-ASIC synthesis and verification; DATE '06: Proceedings of the conference on Design, automation and test in Europe: Designers' forum, March 2006.

Standards

[99] *ANSI:* Programming languages - C++, ISO/IEC 14882; 2003

[100] *W3C:* Extensible Markup Language (XML) 1.0; W3C Recommendation 4. Edition 2006

[101] *IEEE Computer Society:* IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-2002, March 2002

[102] *IEEE Computer Society:* IEEE Standard Verilog Language Reference Manual, IEEE Std 1364-2001

Anbieter und Produktbeschreibungen

[103] *Xilinx Inc.;* http://www.xilinx.com/ise/optional_prod/planahead.htm

[104] *Mathworks MATLAB® & SIMULINK®;* http://www.mathworks.de

[105] *Synplicity Synplify DSP®;* http://www.synplicity.com

[106] *Xilinx AccelDSP®;* http://www.xilinx.com/ise/dsp_design_prod/acceldsp/index.htm

[107] *Impulse C™;* http://www.impulsec.com

[108] *Altera QUIP;* http://university.altera.com/research/unv-quip.html

[109] *Altera Quartus II;* http://university.altera.com/research/unv-quip.html

[110] Tcl programming language and the Tk graphical user interface toolkit; http://www.tcl.tk

[111] *ActiveState Tcl Solutions;* http://www.activestate.com/solutions/tcl

[112] *Actel Corporation;* Actel Igloo FPGA Handbook

[113] *Xilinx Inc.;* Virtex-4 Data Sheet: DC and Switching Characteristics

[114] *Altera Corporation*; Stratix III Device Datasheet: DC and Switching Characteristics

[115] *Altera Corporation*; Stratix III Device Handbook

[116] *Altera Corporation*; Stratix Device Handbook

[117] *Altera Corporation*; Stratix EDA and Academic Developer Functional Description

[118] *Altera Corporation;* Quartus II Version 7.2 Handbook, Volume 1: Design and Synthesis

[119] *Altera Corporation;* Stratix RAM WYSIWYG User Guide; Version 2.1, May 2005

[120] *Altera Corporation;* VQM Extractor and Language Functional Description; Version 2.0, June 2, 2005

[121] *Altera Corporation;* QSF Assignment Descriptions; Version 1.0, January 6, 2004

[122] *Altera Corporation;* Altera XML Architecture Description File Detailed Design; Version 1.4, May 27, 2005

[123] *Open Cores Org;* http://www.opencores.org/

[124] *Rudolf Usselmann;* DES/Triple DES IP Core; http://www.opencores.org/projects.cgi/web/des/overview