Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Untersuchung der Verdrahtungsressourcen reprogrammierbarer Xilinx-FPGAs der 4000er Familie

Diplomarbeit

Betreuung: Prof. Dr. Wilhelm G. Spruth,
Dr. P. Herrmann

vorgelegt von: Nemier Al-Shawi

Leipzig, September 1998

Inhaltsverzeichnis

Einleitung

Das Gebiet der programmierbaren Logik hat seinen Ursprung Mitte der 60er Jahre. Der erste damals entwickelte programmierbare Logikbaustein bestand aus einer einfachen programmierbaren Diodenmatrix. Heute ist die Zahl der Programmable Logic Devices (PLDs) bereits auf über 5000 unterschiedliche Typen angestiegen. Weiter nahm die Komplexität der Bausteine zu. Field Programmable Gate Arrays (FPGAs) sind PLDs, die derzeit eine Komplexität von bis zu ca. 60.000 Gatteräquivalente erreichen. Die ersten FPGAs waren zu Beginn der 80er Jahre verfügbar. Ihr Einsatz wird heute in vielen Anwendungen aus Kostengründen anstelle von Gate Arrays und Standardzellen-ICs bevorzugt. Einige FPGAs, zum Beispiel Logic Cell Arrays (LCAs), sind rekonfigurierbar. Sie können beliebig oft *in-circuit* programmiert werden, ohne sie aus der Schaltung herausnehmen zu müssen. Damit kann die durch FPGAs bereitgestellte Logik wie Software geladen werden.

In [23] wurde ein gemeinsames Projekt des Lehrstuhls für technische Informatik der Universität Leipzig, des Lehrstuhls für Rechnergestütztes Entwerfen der Technischen

Universität München und des Lehrstuhls für technische Informatik der Universität Tübingen vorgestellt. Die Aufgabe ist die Erstellung eines Entwurfssystems für Xilinx-FPGAs von der Beschreibung in einer Hochsprache bis zum Erreichen der plazierten und verdrahteten Netzliste. Die Ergebnisse müssen so dargestellt werden, daß man beim Generieren der Bitströme für die endgültige Programmierung der FPGAs das von Xilinx bereitgestellte Programm Makebits des Entwicklungssystems XACT nutzen kann. Dazu erscheint es unumgänglich, Eingabedateien im LCA-Format für dieses Programm schreiben zu können.

Die Verdrahtung von FPGAs erfolgt über vorgefertigte Leiterbahnsegmente, die durch programmierbare Schalter miteinander verbunden werden können. Aufgrund dieser Struktur unterscheidet sich das Verdrahtungsproblem für FPGAs stark von demjenigen für Makrozellen und Standardzellen, denn man kann die Leitung nicht innerhalb vorgegebener Kanäle oder Flächen frei wählen.

Das führt dazu, daß vorhandene Verdrahtungsprogramme nicht direkt auf dieses spezielle Verdrahtungsproblem angewendet werden können bzw. die Ergebnisse nicht zufriedenstellend sind [8].

In der vorliegenden Arbeit wird zuerst das LCA-Format von Xilinx untersucht und beschrieben. Im zweiten Teil wird ein Algorithmus vorgeschlagen der in der Lage ist, eine optimale Verdrahtung zu berechnen.

In Kapitel 1 werden zunächst die Grundlagen vermittelt, welche an späterer Stelle zum Verständnis der Beschreibung des LCA-Formates und des Verdrahtungsalgorithmus erforderlich sind.

Die Beschreibung des LCA-Formats von Xilinx-FPGAs der 4000er Familie wird in Kapitel 2 ausführlich dargestellt.

Im dritten Kapitel wird die Korrektheit der Beschreibung (Kapitel 2) bewiesen.

In Kapitel 4 wird eine Beschreibung des Verdrahtungsproblems für FPGAs und ein Lösungsansatz, der zur signifikanten Verbesserung der Verdrahtbarkeit und Reduzierung der Verzögerung führen kann, vorgestellt.

1. Einführung in FPGAs

In der Arbeit wird das LCA-Format für Xilinx-FPGAs der 4000er Familie und ein durch Leistung und Verdrahtbarkeit gesteuerter Verdrahtungsalgorithmus (Ablaufverfolger_FPGA) für symmetrische FPGAs beschrieben. Aus diesem Grund folgt zunächst eine kurze Einführung in FPGAs. Anschließend wird der typische Designprozeß für einen FPGA-Baustein erklärt, um zu sehen, an welcher Stelle das LCA-Format und die Verdrahtung und damit der Algorithmus ins Spiel kommen.

1.1 Programmierbare Logik

Die Integrationsdichte insbesondere von Mikroprozessoren und Speichern ist in den letzten Jahren stark angestiegen. Entsprechend stieg auch der Bedarf an höher integrierten Logikschaltungen zur Realisierung der übrigen Systemkomponenten, beispielsweise Decoder oder Interface. Man verwendete hier bisher hauptsächlich Standardbauelemente (Abb 1.1 auf Seite 7) mit dem Nachteil des hohen Platzbedarfs,

Stromverbrauchs und entsprechend hoher Kosten. Daher werden zunehmend anwendungsspezifische Schaltungen (Application Specific Integrated Circuit, ASIC) eingesetzt. Sie unterscheiden sich in ihrer Komplexität und in ihrer Flexibilität, die dem Anwender bereitgestellt wird [3].

Gate Arrays, Standardzellen-Bausteine und Full-Custom ICs kann der Anwender nicht selbst programmieren, sie werden kundenspezifisch beim Hersteller gefertigt. Diese Bausteine sind aufgrund der aufwendigen Entwicklung nur bei relativ hohen Stückzahlen rentabel. Andererseits ist die Integration dieser Bausteine am höchsten. Die übrigen ASICs können vom Anwender programmiert werden, wobei die FPGAs eine so hohe Komplexität besitzen, daß sie teilweise anstelle von Gate Arrays verwendet werden können. Wird nur eine geringe Komplexität gefordert, dann ist der Eisatz von PLD (*Programmable Logic Devices*) empfehlenswert.

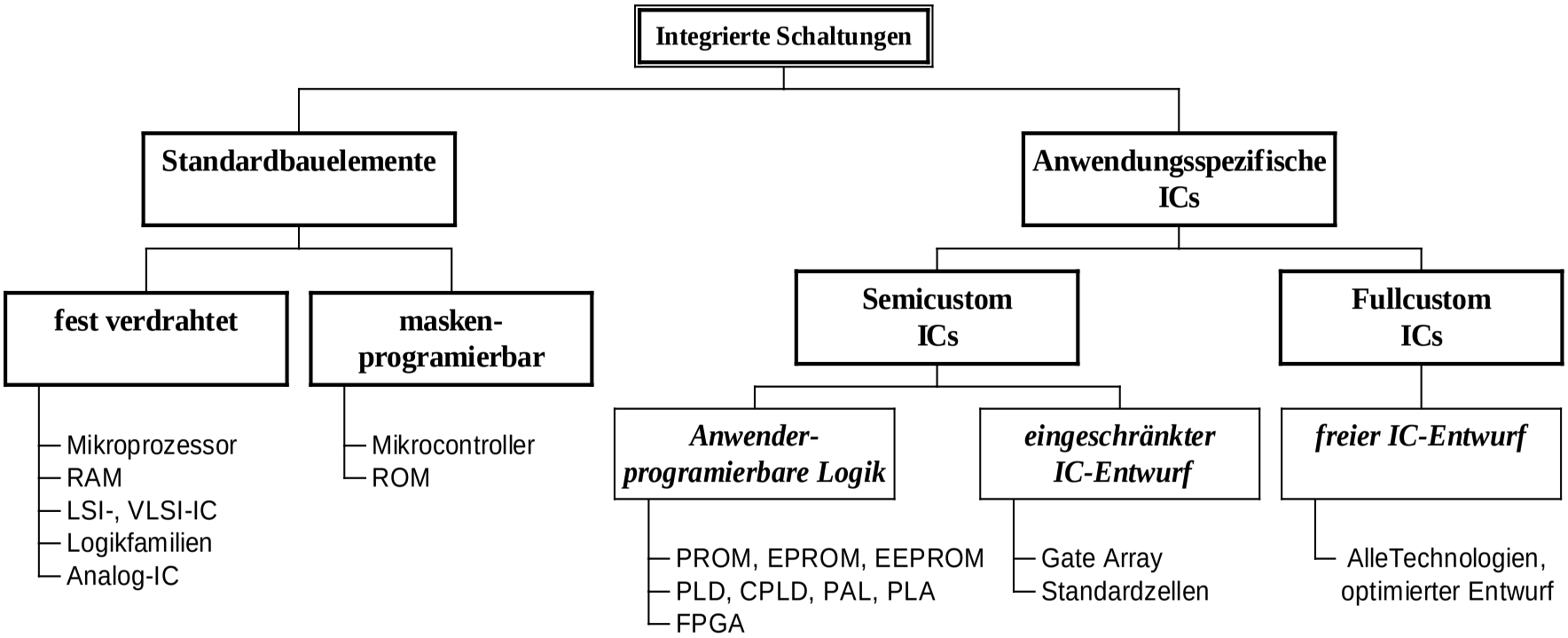


Abbildung 1.1: Einordnung Integrierter Schaltungen

1.2 Programmable Logic Devices

Diodenmatrizen waren die ersten *Programmable Logic Devices* (PLDs) [2], die schon Mitte der 60er Jahre auf den Markt kamen. Sie können vom Anwender mit Hilfe eines Programmiergerätes programmiert werden. Je nach Bausteintyp und Hersteller ist es möglich, die Programmierung wieder zu löschen. Die Zahl der Ausgänge sowie die der Flip-Flops hängt vom verwendeten PLD ab. Die Ausgangsfunktionen stehen in der Regel nur an bestimmten dafür vorgesehenen Anschlüssen zur Verfügung.

Die heutige Vielfalt der PLDs baut auf der Struktur und Realisierungstechnik von ROM- und PROM-Bausteinen auf. Die Grundstruktur besteht aus einer matrixförmigen Anordnung der Logikelemente, die vollständig durch Leitungen vernetzt sind und ein zweistufiges Schaltnetz realisieren. Nach der Programmierbarkeit der UND-Matrix und/oder der ODER-Matrix unterscheidet man drei PLD Haupttypen (PROM, PAL und PLA), dargestellt in der folgenden Übersicht [22]:

	UND-Matrix	**ODER-Matrix**
PROM (EPROM[1]) (Programmable Read Only Memory)	fest	programmierbar
PAL (Programmable Array Logic)	programmierbar	fest
PLA (Programmable Logic Array)	programmierbar	programmierbar

Gelegentlich findet man auch die Bezeichnung CPLD (Complex PLD), dies sind Bausteine, die die herkömmliche PLD-Struktur mehrfach enthalten und daher häufig hierarchisch aufgebaut sind.

1.2 FPGAs

Die ersten *Field Programmable Gate Arrays* (FPGAs) [2, 3 und 9] sind Anfang der 80er Jahre entwickelt worden.

Tabelle 1.1 zeigt FPGA-Hersteller mit ihren unterschiedlichen Konzepten und ihren unterschiedlichen Architekturen.

Hersteller	**Architektur**	**Logikblock**	**Programmier-bare Zelle**	**Wiederpro-grammierbar**
Actel, Texas	Zeilenform	Multiplexer	Anti-fuse	-
Algotronix	Sea-of-gates	Multiplexer + einfache Logik	SRAM	+
Altera	Hierachisches PLD	PLD Block	EPROM	+
AMS	Hierachisches PLD	PLD Block	EEPROM	+
Atmel (Concurrent)	Sea-of-gates	Multiplexer + einfache Logik	SRAM	+
Crosspoint	Zeilenform	Transistorpaar + Multiplexer	Anti-fuse	-
Plessey	Sea-of-gates	NAND-Gatter	SRAM	+
Plus	Hierarchisches PLD	PLD Block	EPROM	+
Quick Logic	Matrix	Multiplexer	Anti-fuse	-
Xilinx, AT&T	Matrix	Look-up Tabelle	SRAM	+

Tabelle 1.1: FPGAs einiger Hersteller

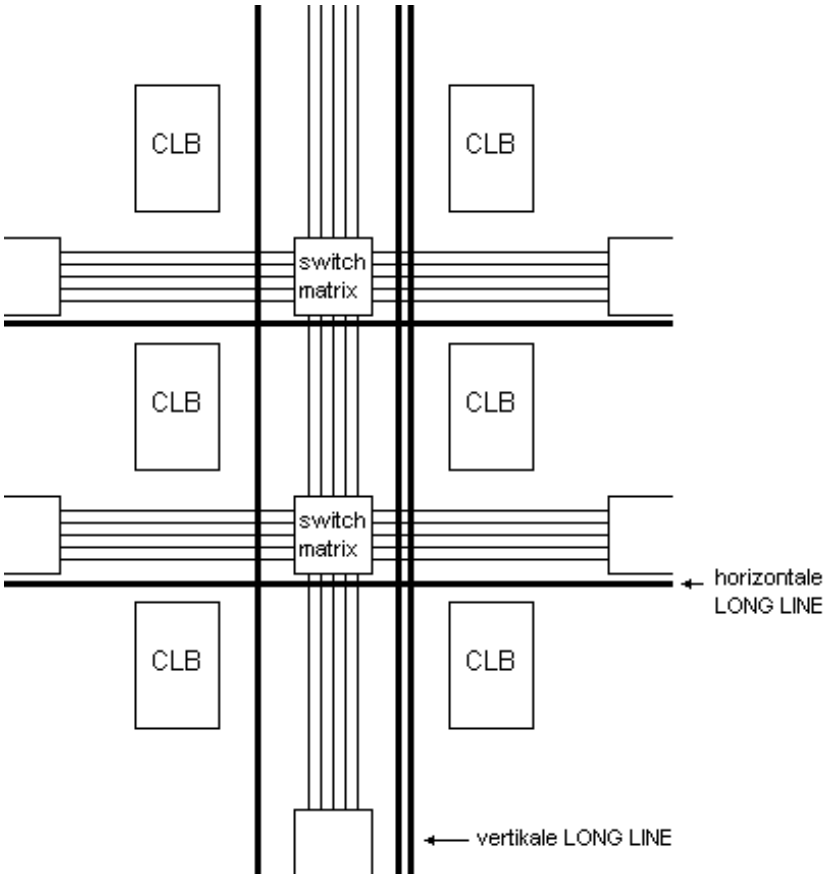
Die Firma Xilinx gehörte zu den ersten, die mit Chips auf den Bereich zwischen den einfach zu handhabenden aber eingeschränkten PALs und den leistungsfähigeren Gate Arrays zielte.

[1] EPROM steht für Erasable Programmable Read Only Memory. Diese Bausteine haben gegenüber den nur einmal programmierbaren PROMs den Vorteil, daß der Speicher dieser Bausteine gelöscht und dann

Diese FPGAs vereinen die einfache Programmierbarkeit der PALs mit der Möglichkeit, komplexe Funktionen zu implementieren, für die früher zumindest Gate Arrays notwendig waren.

Ein Xilinx Chip besteht im Prinzip aus einer Reihe von sogenannten Configurable Logic Block (CLBs), die in Form einer Matrix angeordnet sind. Diese CLBs sind für die Implementierung der logischen Funktionen zuständig. Sie besitzen eine Reihe von Ein-/ Ausgängen, interne Flip-Flops und eine kleine ladbare RAM-Zelle, die beschreibt, welche Funktionswerte an den Ausgängen in Abhängigkeit von den Eingangswerten erscheinen. Damit ist eine beliebige logische Funktion programmierbar, während die Flip-Flops für die Speicherung von logischen Zuständen verwendet werden können.

Die CLB Ein- und Ausgänge sind nun nicht unmittelbar mit den Pins des Chips verbunden. Hierfür sind die sogenannten I/O Blocks (IOBs) zuständig. Sie sind ebenfalls in eingeschränktem Maße konfigurierbar (Eingabe, Ausgabe, Tri-State etc.).



neu programmiert werden kann.

Abbildung 1.2: Das Generale Purpose Netzwerk eines Xilinx-FPGAs

Für die Verbindung zwischen IOBs und CLBs, sowie CLBs untereinander, gibt es mehrere Möglichkeiten.

- Zunächst einmal stehen eine sogenannte *Nächste Nachbar Verbindung* zwischen den CLBs zur Verfügung. Wie der Name schon sagt, erlauben sie nur Verbindungen zu den unmittelbar umliegenden Zellen.

- Der Hauptteil aller Verbindungen wird über eine sogenannte Verbindungsmatrix geführt, die horizontale und vertikale Leitungen über die Chipfläche erlaubt. Je nach Bausteintyp stehen dabei jeweils mehrere *single-length local Lines* zwischen zwei Zeilen bzw. Spalten zur Verfügung. Die 4000er Familie besitzt zusätzlich *double-length local Lines*. Sie überspringen immer eine Ver-bindungsmatrix, wodurch sie eine schnelle Signalverdrahtung über mittlere Strecken bieten.

- Für besondere Zwecke gibt es schließlich sogenannte *Long Lines*. Diese sind z.B. nützlich, wenn ein große Anzahl von CLBs an einem gemeinsamen Input hängt (wie ein Clock-Signal), oder zwei entfernte CLBs mit kurzem Delay verbunden werden müssen. Während über die im vorhergehenden Punkt erwähnte Matrix im Prinzip zwei beliebige CLBs verbunden werden können, gibt es bei den Long Lines einige Einschränkungen: So müssen die CLBs in der gleichen Spalte oder Zeile liegen, und die Anzahl der Long Lines selbst ist stark begrenzt. Der Designer muß also schon beim Plazieren seiner Bauteile auf diese Fälle Rücksicht nehmen.

Die verschieden Leitungen können auch mit Hilfe von PIPs (Programmable Inter-connect Point) untereinander und mit den Pins der CLBs und IOBs verschaltet werden.
Die tatsächlichen Ressourcen (wie Anzahl der CLBs, IOBs und interne Verbindungen) hängen vom Typ des eingesetzten Chips ab. Der XC4003/3A Baustein besitzt beispielsweise eine Matrix von 10×10 CLBs (d.h. 100 CLBs), einen maximalen RAM von 3200 Bits und 360 Flip-Flops. Die Anzahl der verwendbaren IOBs hängt wiederum von der verwendeten Gehäuseform ab (bis zu 80).

Bei näherer Betrachtung der Tabelle 1.2 ist zu erkennen, daß in den XC4000 die Anzahl der CLBs und IOBs im Einzelnen sehr differenziert ist, und damit ist schaltungsspezifisch eine optimale Bausteinwahl möglich.

	XC4002	**XC4004**	**XC4006**	**XC4010**	**XC4013**	**XC4016**	**XC4020**
Gatteräquivalenz	2000	4000	6000	10 000	13 000	16 000	20 000
CLB-Matrix	8×8	12×12	16×16	20×20	24×24	26×26	30×30
Anzahl von CLBs	64	144	256	400	576	676	900
Anzahl von IOBs	64	96	128	160	192	208	240
Max RAM-Bits	2048	4608	8192	12800	18432	21632	28800

Tabelle 1. 2: Eine Auswahl FPGAs der 4000er Familie

Die Funktion eines FPGAs hängt nun völlig von der augenblicklich geladenen Konfiguration ab, die in den RAM-Zellen gespeichert ist. Das Laden dieses RAMs erfolgt nach einem Power-On Reset über ein serielles oder paralleles Interface. Allerdings ist auch während des Betriebes eine Umkonfiguration möglich. Dies erlaubt beispielsweise die Änderung eines existierenden Designs, falls ein Fehler entdeckt wurde, und das Laden des geänderten Designs innerhalb einer kurzen Zeitspanne in ein laufendes System.
Eine andere Möglichkeit ist, verschiedene Funktionen für einen Chip vorzusehen, die er nacheinander erledigen kann, und die Konfiguration des Bausteins jeweils nach Bedarf zu ändern. Da die Ladezeit im Bereich von Millisekunden liegt, erscheint dies auch für Real-Time Anwendungen im Bereich des Machbaren zu liegen.

Offensichtlich bieten FPGAs eine Reihe von Vorteilen sowohl gegenüber PALs als auch herkömmlichen Gate Arrays:

- Höhere Komplexität als PAL
- Interne Verbindungen
- Dadurch schneller
- Löschbar (im Gegensatz zu Gate Arrays)
- Schnell konfigurierbar

Nachteile sind die im Vergleich zu PALs kompliziertere Entwicklungsumgebung und die im Augenblick vorhandene Abhängigkeit von einem bestimmten Hersteller. Meist existieren kaum Second Sources für die verschiedenen FPGAs. Da jeder Hersteller seine eigene Hardware entwirft, sind FPGAs verschiedener Hersteller auch nicht austauschbar.

1.3 Der FPGA-Designprozeß

Der Entwurf eines FPGA-Design unterscheidet sich erheblich vom dem anderer programmierbarer Bausteine wie z.B. PALs. Letztere haben meist eine einfache und nicht besonders flexible Struktur. Der Benutzer spezifiziert seine logischen Gleichungen und kann angeben, wie er seine Ein- und Ausgabepins belegt haben will. Normalerweise wird ihm das entsprechende Designprogramm irgendwelche Konflikte (zu komplizierte Gleichungen etc.) sofort mitteilen, so daß er es korrigieren kann.

Der Unterschied zwischen PALs und Xilinx FPGAs liegt nun nicht nur in der größere Anzahl der verfügbaren Gatter, sondern auch darin, daß der Benutzer eine größere Freiheit hat, wie er die CLBs innerhalb der Matrix belegen will. Die kann wichtig sein bei zeitkritischen Pfaden innerhalb des Design, wo z.B. eine bestimmte Verzögerung zwischen zwei Ein- / Ausgängen nicht überschritten werden darf.

Weiterhin sind die Ressourcen innerhalb eines Chips natürlich immer noch beschränkt. Selbst bei nicht besonders zeitkritischen Pfaden ist es nicht angebracht, zwei logisch verbundene CLBs an entgegengesetzten Enden des Chips zu Plazieren: Die entsprechende Verbindung kostet Ressourcen und Laufzeit innerhalb der Verbindungsmatrix, deren Fehlen evtl. an anderer Stelle zu Engpässen führen.

Damit die Implementierung des Designs den Anforderungen entspricht, sind also zwei Probleme zu lösen:

1. Die CLBs und IOBs müssen innerhalb des Chips *plaziert* werden, und zwar so daß
2. eine möglichst optimale *Verdrahtung* der Verbindungen auf dem Chip möglich ist.

Beides kann der Benutzer z.B. per Hand durchführen. Die Firma Xilinx stellt Entwicklungstools mit graphischer Oberfläche zur Verfügung, mit deren Hilfe der Benutzer die Logikelemente auf dem Chip manipulieren (plazieren, verschieben, löschen), sowie die Verbindungen auf das vorhandene Netz abbilden kann. Dies ist allerdings ein sehr zeitaufwendiger und fehlerträchtiger Prozeß.

Ein erfahrener Designer braucht etwa in der Größenordnung von zwei bis drei Wochen für ein Design. Änderungen sind nur sehr schwer durchzuführen, einmal weil die Entwicklungssoftware nicht unbedingt als ausgereift und benutzerfreundlich zu bezeichnen ist, andererseits weil ein Design aus vielleicht hunderten von CLBs in seiner Komplexität nicht mehr einfach zu durchschauen ist. Selbst kleine Änderungen wie die Verschiebung eines CLBs auf eine andere Position können Unmengen von Folgeänderungen notwendig machen.

Die Firma Xilinx bietet eine Reihe von Werkzeugen, die diesen Prozeß automatisieren sollen (Autoplacement und Autorouting). Allerdings liefert keines der Programme befriedigende Ergebnisse, insbesondere wenn es sich um ein dicht besetztes oder irreguläres Design handelt. Es sollte betont werden, daß dies natürlich die interessanten Fälle sind. Eine logische Schaltung mit nur wenigen Gattern ist auf einem andererseits leeren FPGA einfach unterzubringen. Auch eine reguläres Design, bei dem sich Strukturen regelmäßig wiederholen, kann oft effizient implementiert werden. Das eigentliche Problem liegt bei größeren Designs mit irregulärer Struktur.

Die Aufgabe des Entwicklers besteht also aus dem Plazierungs- und dem Verdrahtungsprozeß. Es handelt sich hier also um einen Optimierungsprozeß. Der Benutzer muß sein Design so auf die existierenden Hardwareressourcen abbilden, daß eine (noch genauer zu definierende) Kostenfunktion minimiert wird.

Im Normalfall wird es sich darum handeln, die Verbindungswege innerhalb des FPGAs möglich kurz zu halten, so daß das Gesamtsystem mit der höchstmöglichen Geschwindigkeit funktionieren wird. Natürlich hat jedes Design seine kritischen Punkte (Kpitel 4), deren Rahmenbedingungen auf jeden Fall erfüllt sein müssen.

Ein im Mittel zufriedenstellendes Gesamtsystem kann in der Praxis unbrauchbar sein, weil eine ganz bestimmte minimale Verzögerung an einer ganz bestimmten Stelle nicht eingehalten wird.

Im folgenden Kapitel wird eine Beschreibung des LCA-Format für Xilinx-FPGAs der 4000er Familie gegeben. Dateien dieses Formats werden im Design Flow des XACT Systems von Xilinx (Abb 1.3) als Ausgabe der Plazierungs- und Verdrahtungsprogramme APR bzw. PPR erzeugt.

Im vierten Kapital wird dann beschrieben, wie ein durch Leistung und Verdrahtbarkeit gesteuerter Algorithmus verwendet wird, um das im vorangehenden erwähnte *Verdrahtungsproblem* zu lösen.

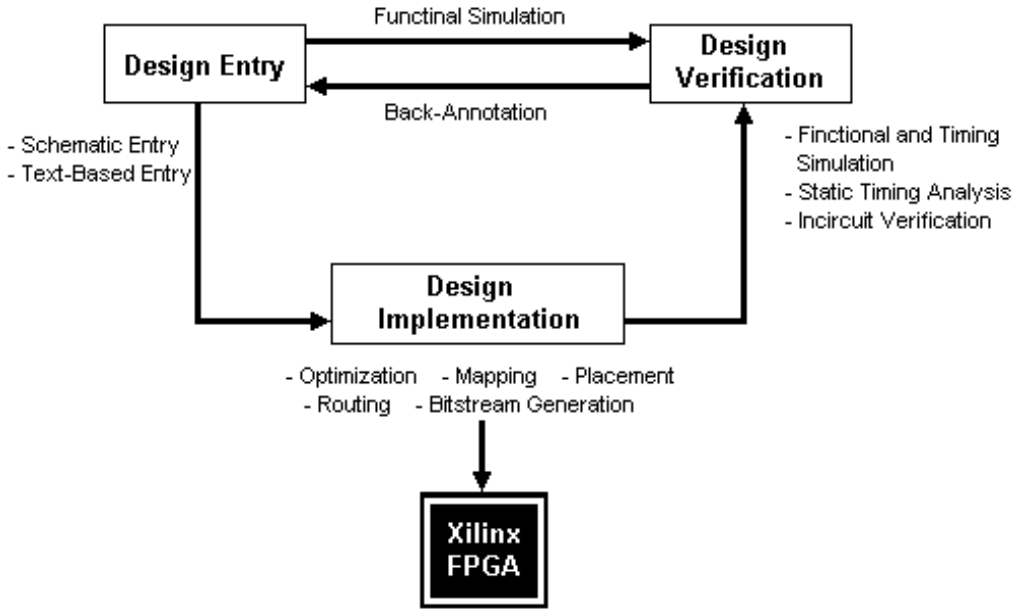


Abbildung 1.3: Design Flow des XACT Systems von Xilinx

2 Das LCA-Format (Xilinx 4000)

Wenn man ein Entwurfssystem für Xilinx-FPGAs von der Beschreibung in einer Hochsprache bis zum Erreichen der plazierten und verdrahteten Netzliste erstellen will, müssen die Ergebnisse so dargestellt werden, daß man beim Generieren der Bitströme für die endgültige Programmierung der FPGAs das von Xilinx bereitgestellte Programm Makebits des Entwicklungssystems XACT nutzen kann. Dazu erscheint es unumgänglich, Eingabedateien im LCA-Format für dieses Programm schreiben zu können.

Im folgenden werden zuerst die Bezeichnungen der Elementen eines Xilinx-FPGAs der 4000er Familie erklärt, dann wird das LCA-Format, das zur Beschreibung der plazierten und verdrahteten Netzliste dieser Familie dient, näher erläutert. Unsere Beschreibung wird sich auf das durch PPR von Xilinx erzeugte LCA-File COUNT.LCA (Anlage A) basieren.

Zur Untersuchung des LCA-Formats wurden das Datenbuch von Xilinx [30] und die XACT-Entwicklungsumgebung mit Beschreibung [31] genutzt.

2.1 Bezeichnung der Elemente eines 4000er FPGAs

Bevor man mit der Beschreibung der einzelnen Teile des LCA-Formats beginnt, müssen die Bezeichnungen der einzelne Elemente deutlich gemacht werden weil in dem LCA-Format besondere Bezeichnungen für die Elemente eines FPGAs benutzt werden. Für die Bezeichnung der Elemente eines FPGAs ist es wichtig, die Unterteilung des Chips in Zeilen und Spalten zu kennen (Abbildung 2.1). Die erste Zeile beginnt an der oberen Kante des Chips und endet vor der ersten Zeile von CLBs. Danach beginnt die zweite Zeile, die die erste Zeile von CLBs umfaßt, usw. Nach der letzten Zeile von CLBs schließt sich noch eine weitere Zeile an, die bis zur unteren Kante des Chips geht. Dementsprechend sieht es bei den Spalten aus.
Die Angabe von Zeilen oder Spalten erfolgt im LCA-Format durch Großbuchstaben; so steht ein A für die erste Zeile oder Spalte, ein B für die Zweite, usw.

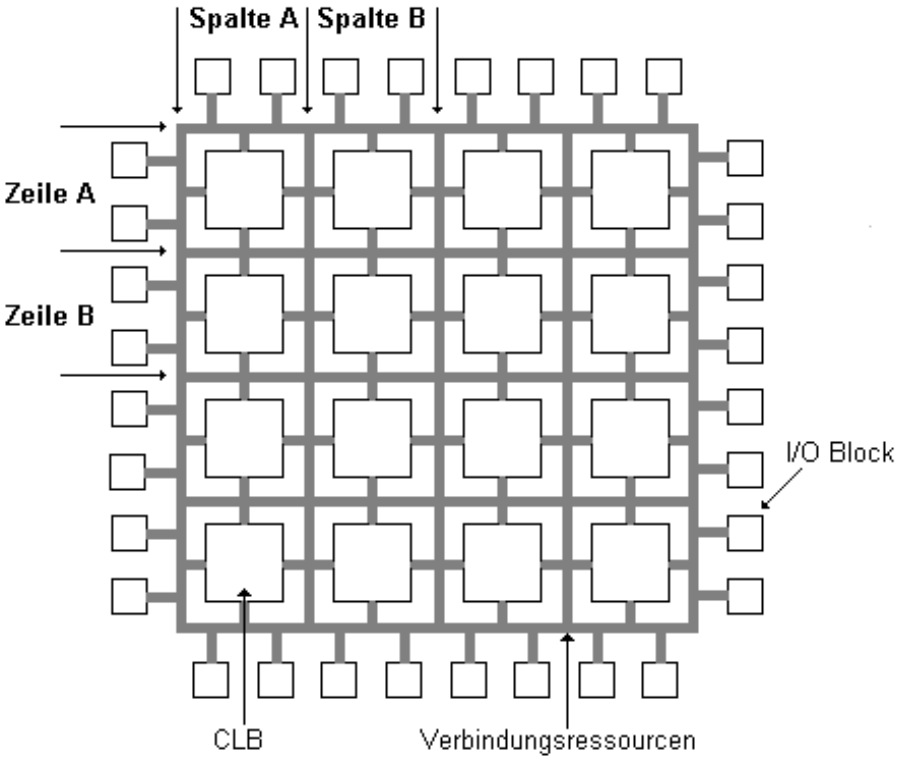


Abbildung 2.1: Skizze eines FPGAs

Bemerkung 1:

Es ist für die Bezeichnung von Elementen sehr wichtig zu wissen, daß jeder Chip eine mittlere Zeile bzw. Spalte besitzt, die nur Leitungssegmente (globale Netze) enthält.

Bemerkung 2:

In den Ecken eines 4000er FPGAs befinden sich spezielle Elemente für z.B. Clocksignale und Boundary Scane, aber keine CLBs, IOBs und Switchmatrizen.

Wir werden uns in unserer Erklärung auf den kleinsten Xilinx-FPGA der 4000er Familie (4002APC84) beziehen. Dieser FPGA hat 11 Zeilen und 11 Spalten aber nur 8 Zeilen und 8 Spalten von CLBs (d.h. die erste, die sechste und elfte Zeile bzw. Spalte umfassen keine CLBs).

2.1.1 CLBs und ihre Pins

Die Hauptelemente jedes CLBs eines Xilinx-FPGA der 4000er Familie werden in Abbildung 2.2 gezeigt.

Es hat acht unabhängige (F1-F4 und G1-G4) Inputpins für logische Variablen, einen allgemeinen Clock-Eingang (K) und vier Kontrolleingänge (C1-C4). C1-C4 können durch Multiplexer auf willkürliche Weise auf vier interne Kontrollsignale (H1, DIN, S/R und EC) abgebildet werden. H1 ist ein weiterer Eingang für logische Variablen. DIN wird für die direkte Eingabe benutzt. S/R könnte entweder zu einen Asynchron-Einsteller oder zu einen Reset-Eingang programmiert werden. EC ist zur Aktivierung der Clock.

Das CLB hat vier Outputpins (X, Y, XQ und YQ). X und Y werden als Ausgänge für die Signale der Funktionsgeneratoren eines CLBs benutzt. XQ und YQ werden durch die Flip-Flops eines CLBs gesteuert.

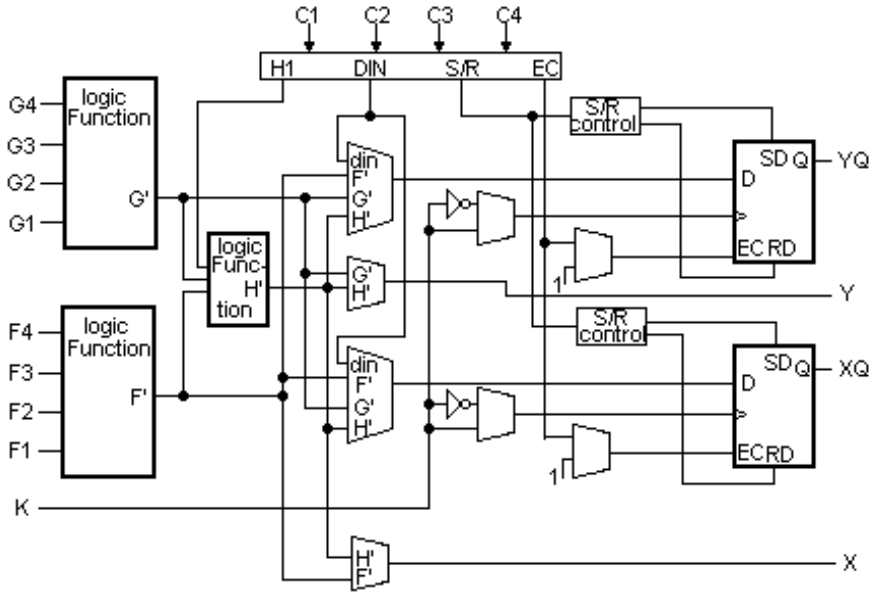
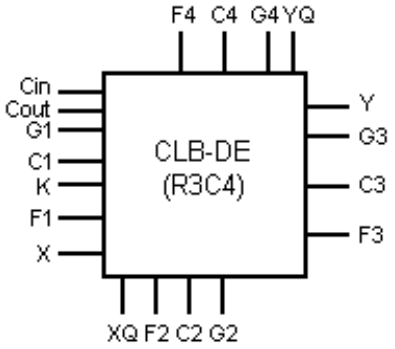


Abbildung 2.2 : Aufbau der CLB in den XC4000- Bausteinen

Wie schon erwähnt sind die CLBs (logischen Blöcke) in Form einer Matrix angeordnet. Die Bezeichnung eines CLBs besteht aus zwei Großbuchstaben.

Der erste Buchstabe gibt die Zeile wider, in der sich der Block befindet, der zweite Buchstabe die Spalte. Der Block BJ z.B. befindet sich in der zweiten Zeile (erste CLB Zeile) und zehnten (achte CLB Spalte) Spalte.



Die Bezeichnungen der CLB-Pins werden durch Anhängen eines Punktes und des entsprechenden Buchstabens an der Bezeichnung des CLBs gebildet. Z.B. BG.F1, JJ.G3, CB.C4 oder HH.K.

2.1.2 IOBs und ihre Pins

Die Elemente eines I/O-Blockes sind in Abbildung 2.3 aufgezeigt.

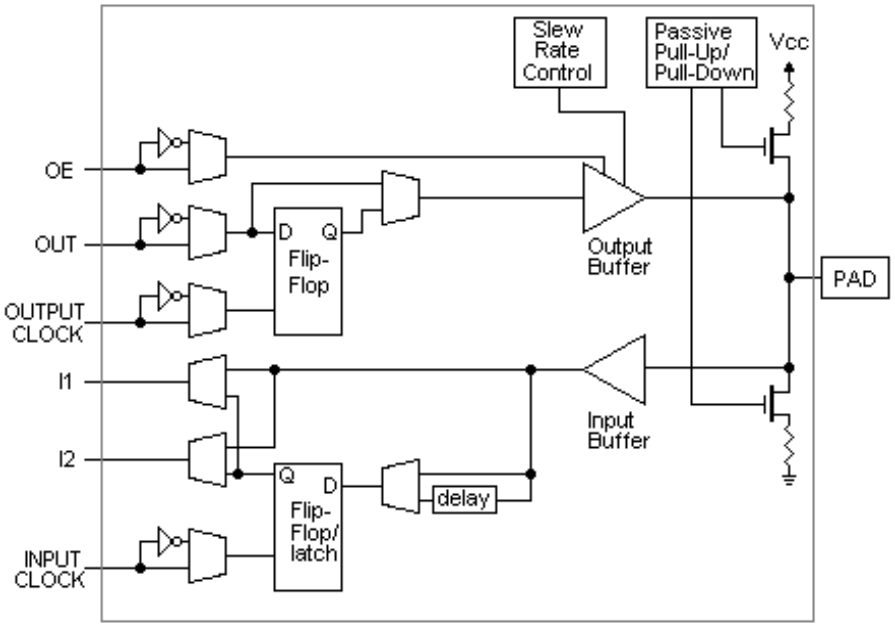
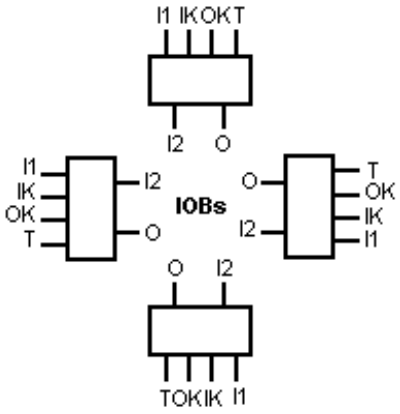


Abbildung 2.3 : Architektur des I/O-Blocks der XC4000 LCA

Die IOBs haben zwei Signalinputs (I1 und I2), einen Clock-Eingang (IK), einen Clock-Ausgang (OK), einen Output (O) und einen Outputaktivierungspin (EO).Die I/O-Blöcke

befinden sich an den Seiten des Chips. An der Ober- und Unterseite findet man jeweils zwei IBOs pro Spalte, rechts und links jeweils zwei IOBs pro Zeile.

Die Bezeichnung der IOBs besteht aus einem Wort PAD und einer Nummer (z.B. 16), die von der Position eines IOBs abhängig ist. Diese Nummer bekommt man, in dem man, von der linken oberen Ecke ab ausgehend, die IOBs einfach im Uhrzeigersinn zählt. PAD11 z.B., ist der erste IOB im Oberteil der Spalte H (4002APC84).

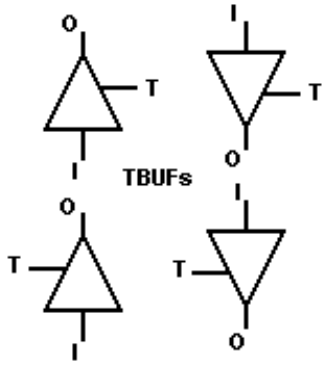


Die Bezeichnung der Pins läuft genauso ab wie bei den CLBs und zwar in dem man den Buchstaben, der den jeweiligen Pin bezeichnet, an den Namen des IOBs anhängt, wobei sie durch einen Punkt abgetrennt werden (z.B. PAD11.I2 oder PAD55.OK).

2.1.3 Tristate-Buffer (TBUF)

Tristate-Buffer werden von den CLBs benutzt, um Signale auf die am nächsten horizontalen Long-Lines, die über bzw. unter den Blocks liegen, zu übertragen. Dabei werden sie von den IOBs für die Implementierung von multiplexen oder bidirektionalen Bussen auf die horizontalen Long-Lines benutzt. Jeder Tristate-Buffer hat einen Eingang (I), einen Ausgang (O) und Pin (T) für das Steuersignal. Für (T=0) ist der Ausgang (O) hochohmig. Ist (T=1), so wird das Signal am Eingang (I) auf den Ausgang (O) geschaltet und verstärkt.

Pro Zeile und Spalte gibt es zwei TBUF (außer erste und letzte Zeile sowie mittlere Zeile bzw. Spalte). Die TBUF in der ersten bzw. letzten Spalte sind von den IOBs zu nutzen, die anderen von den CLBs.

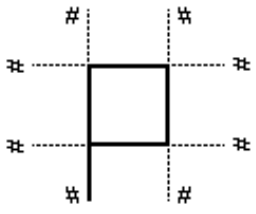


Die Bezeichnung eines Tristate-Buffers mit seinen Pins besteht aus vier jeweils durch einen Punkt getrennte Teile. Der erste Teil ist die Abkürzung TBUF. Der zweite Teil besteht aus zwei Großbuchstaben, von denen der erste die Zeile, und der zweite die Spalte angibt, in denen sich der Tristate-Buffer befindet. Der dritte Teil ist eine Zahl. Sie gibt an, ob es sich um den ersten oder zweiten Tristate-Buffer der beiden handelt, die sich in der Zeile und Spalte befinden.
Der vierte Teil zeigt, welcher Pin benutzt werden soll. Ein Beispiel dafür ist `TBUF.CD.2.I`. Hier handelt es sich um den Pin (I) vom zweiten (von oben) Tristate-Buffer, der sich in der 3 Zeile und 4 Spalte befindet.

2.1.4 Pull-Up-Resistors

Zur Realisierung von Wired-And-Funktionen sind die Leitungen, zu denen die Tristate-Buffer führen, durch Widerstände (pull up resistors) abgeschlossen.

Nur in der ersten und letzten Spalte gibt es pro Zeile (außer erste, mittlere und letzte Zeile) jeweils zwei Pull-Up-Resistors. Jeder Resistor hat nur einen Pin (O).



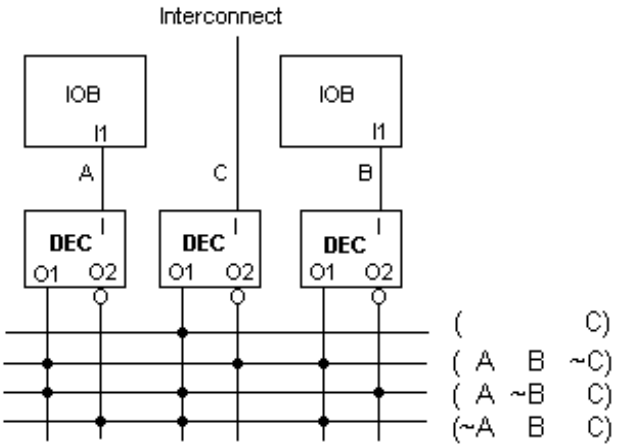
Pull Ups
(in any rotation)

Die Bezeichnungen von Pull-Up-Resistors werden wie die vom Tristate-Buffer gebildet; z.B. PULLUP.JA.2.O. Der erste Großbuchstabe nach dem ersten Punkt bezeichnet die Zeile, der zweite Buchstabe die Spalte, in der sich der Widerstand befindet. Die Zahl danach sagt aus, ob es sich um den ersten oder zweiten Widerstand in dieser Zeile und Spalte handelt. Der Buchstabe am Ende bezeichnet den einzigen Pin (O).

2.1.5 Wide Decoder

Die Wide-Decoder befinden sich an den Seiten des Chips. An der Ober- und Unterseite findet man jeweils drei Decoder pro Spalte, rechts und links jeweils drei Decoder pro Zeile.

Die Peripherie eines Chips hat an jeder Kante vier Wide-Decoder-Schaltkreise (zwei bei XC4000A), deshalb können alle IOBs einer Kante einen Decoder benutzen. Auch CLBs können die Decoder benutzen. Jeder Decoder hat einen Eingang (I) und zwei Ausgänge (O1,O2).



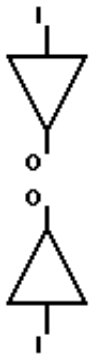
Die Namen der Wide-Decoder beginnen mit DEC. Danach folgt ein Punkt, zwei Großbuchstaben, die angeben in welcher Zeile und Spalte sich der entsprechende Decoder befindet. Die folgende Zahl getrennt durch einen Punkt besagt, ob es sich um den ersten, zweiten oder dritten (von oben nach unten, von links nach rechts) Decoder in dieser Zeile und Spalte handelt. Der letzte Buchstabe, bezeichnet den Namen des Pins. DEC.KG.3.O2 z.B. repräsentiert den (O2) Ausgang des dritten Decoders, der sich in der 11 Zeile bzw. 7 Spalte befindet.

2.1.6 Elemente für Clocksignale

Für Clocksignale gibt es drei verschiedenartige Elemente: Clockbuffer, globaler Clockbuffer und Oszillator.

2.1.6.1 Clockbuffers

In jeder Ecke eines Chips befinden sich zwei Clockbuffer (Bufgs, Bufgp) für Clocksignale. Jeder Puffer hat einen Eingang (I) und einen Ausgang (O).



BUFGP oder BUFGS

Die Bezeichnung ist von der Position (welche Ecke) abhängig. Die Bezeichnung ist in Englisch für oben (top = t), unten (bottom = b), links (left = l) und rechts (right = r). z.B. für die zwei Puffer oben an der rechten Kante des FPGAs `bufgs_tr` und `bufgp_tr` (tr = top right), und die der zwei Puffer unten an der linken Kante `bufgs_bl` und `bufgp_bl`. (tl = bottom left).
Die Namen der Pins werden durch Anhängen eines Punktes und entsprechenden Buchstaben an den Namen des Elementes gebildet (z.B. `bufgp_br.I`).

2.1.6.2 Global Clockbuffers

Auch die globalen Puffer (Global Clockbuffers) befinden sich jeweils zu zweit in jeder Ecke eines FPGAs. Die globalen Puffer haben jeweils nur ein Pin (I).

Die Namen dieser globalen Puffer sind wie die Namen der Puffer (2.1.6.1) gebildet, `i_bufgp_br.I` und `i_bufgs_br.I`. i_bufgp_br und i_bufgs_tr steht für „global clock buffer“, die sich unten an der rechten Kante (bottom right) befinden. Der Buchstabe am Ende bezeichnet den Pin.

2.1.6.3 Oszillator (osc4000)

Ein FPGA der 4000er Familie besitzt ein Oszillator OSC mit fünf Pins (F8M, F500K, F16K, F490 und F15). Der Oszillator befindet sich in der oberen rechten Ecke des Chips. Dieser Oszillator arbeitet mit einem 8 MHz Signal (F8M). Die anderen Pins repräsentieren die verschiedenen Signale, die ein OSC4000 in etwa liefern kann (500 kHz, 16 kHz, 490 Hz und 15 Hz).

Die Bezeichnung des Oszillators ist sehr einfach. Sie besteht aus OSC und die Bezeichnung von dem jeweiligen Pin getrennt durch einem Punkt (beispielsweise `OSC.F500K` oder `OSC.F15`).

2.1.8 Spezielle Modus Pins

Diese Pins können nach der Konfiguration als Hilfsverbindung benutzt werden : Mode 0 (`MD0.I`) und Mode 2 (`MD2.I`) als Inputs, und Mode 1 (`MD1.O` und `MD1.T`) als ein Output.

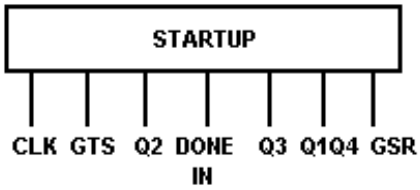
Das XACT-Entwicklungssystem wird diese Ressourcen ohne eine direkte Anweisung nicht benutzen. Die Blöcke dieses Pins befinden sich in der unteren linken Ecke eines Chips in der Nähe vom Readback.

2.1.7 Blöcke für spezielle Zwecke

Urladen (startup4000)

Die Outputs dieses Registers kann man programmieren, um die folgenden drei Ereignisse zu kontrollieren:

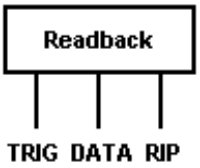
- Freigabe eines open-drain DONE Outputs.
- Aktivierung aller IOBs.
- Abschluß der globalen Set/Reset Initialisierung aller CLB und IOB Speicherelemente



Die 4000 Familie bietet große Flexibilität, denn die drei Ereignisse (DONE going High, User I/O going active und internal R/S being deactivated) können in jeder beliebigen Reihenfolge auftreten.

Readback (rdbk4000)

Der Benutzer kann den Inhalt vom Konfigurationsspeicher und den Level, einen internen Knoten, ohne die Intervention mit der normalen Operation eines Bausteins rücklesen.



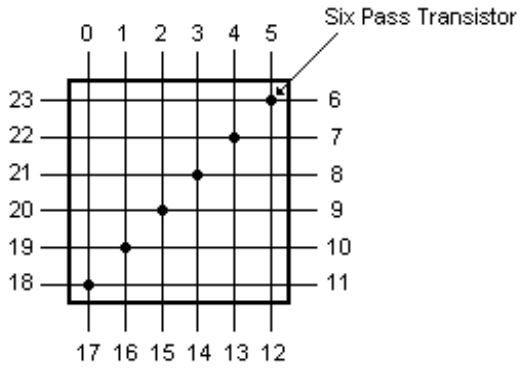
4000er Readback benutzt keine Standpins, sondern vier interne Netze (`RDBK.TRIG, RDBK.DATA, RDBK.RIP` und `RDBK.CLK`), die mit jedem IOB verdrahtet werden können.

Es existieren noch weitere Blöcke, wie Boundary Scan (bscan4000), Daten-Register (todo4000), (update4000), Clock-lesen (rdclk4000) und Dummyload, die nicht weiter beschrieben werden.

Die Bezeichnung aller dieser Blöcke ist im Prinzip gleich, sie besteht aus zwei Teilen, die durch einem Punkt getrennt sind. Der erste Teil bezeichnet den Block (z.B. `bescan` für Boundary Scan) und der zweite den Pin.

2.1.9 Die Switchmatrizen und ihre Pins

Jeder FPGA hat ein Gitter von Switchmatrizen. Lokale Leitungsstücke können mit der Hilfe der Switchmatrizen miteinander verschachtelt werden.
Die Anzahl der Switchmatrizen in einer Zeile bzw. Spalte ist um eins größer als die entsprechende Anzahl der CLBs.
In der ersten und mittleren Zeile bzw. Spalte und an den Ecken des Chips findet man keine Switchmatrizen.



Die Bezeichnung der Switchmatrizen beginnt mit zwei Großbuchstaben, die angeben, in welcher Zeile und Spalte sich die entsprechende Matrix befindet. Danach folgt ein Punkt, die Zahl 24, ein weiterer Punkt und eine 1. Die Bezeichnung eines Pins ergibt sich durch Anhängen einer Nummer, die den Pin widergibt, an die Bezeichnung der Switchmatrix, wobei wieder durch Punkt getrennt wird.

Das Durchnumerieren der Anschlüsse erfolgt im Uhrzeigersinn beginnend mit 0. Es wird von dem oberen linken Pin angefangen. Pin `GH.24.1.3` z.B. ist der vierte von links auf der oberen Kante der Switchmatrix `GH.24.1`.

2.1.10 PIPs

Die programmierbaren Schalter PIPs (programmable interconnect points) werden für das Verschalteln von verschiedenen Leitungen untereinander und mit den Pins der CLBs, IOBs, Tristate-Buffer usw. benutzt.

Die Namen der PIPs werden aus denen der Leitungsstücke und der Pins bzw. der anderen Leitungsstücke gebildet. Dazu werden einfach die Namen der zu verbindenden Pins oder Leitungen verwendet und diese durch Doppelpunkt aneinandergefügt.

2.1.11 Programmierbare Durchschaltungen

Alle internen Verbindungen für die gewünschte Verdrahtung sind aus Metallsegmenten mit programmierbaren Schaltern (PIPs programmable interconnect points) zusammengesetzt. Um eine effektive Verdrahtung zu erzielen, sind die einzelnen Verdrahtungsressourcen mehrmals vorhanden. Die Anzahl der Verdrahtungskanäle ist von der Größe des Arrays abhängig (im allg. wächst sie mit der Arraygröße).

Die FPGAs der 4000er Familie besitzen zwei Haupttypen von Leitungen, die sich durch die relative Länge ihre Segmente unterscheiden: lokale Leitungen (single-length Leitungen und double-length Leitungen) und lange Leitungen. Außerdem existieren noch globale Netze für Clocksignale.

Die Bezeichnung der Leitungen hat die Form `col.K.lokal.4` oder `row.J.lokal.6`. Der erste Bezeichner `row` (Zeile) oder `col` (Spalte) besagt, ob es sich um eine waagerechte oder eine senkrechte Leitung handelt.

Danach wird durch einen Großbuchstaben angegeben, in welcher Zeile oder Spalte sich die angegebene Leitung befindet. Der dritte Teil sagt aus, ob es sich um eine lange (`long`) oder lokale (`lokal`) Leitung handelt. Die Zahl am Ende gibt an, um welche Leitung dieser Art (in dieser Zeile bzw. Spalte) es sich handelt.

Lokale Leitungen

Die single-length Leitungen bilden ein Gitter von horizontalen und vertikalen Leitungen, die sich an den Switchmatrizen zwischen jedem einzelnen Block schneiden.

Abbildung 2.4 zeigt die single-length Leitungen um einen CLB eines Arrays. Die Funktionsgeneratoren und die Controlinputs zu den CLBs (F1-F4, G1-G4 und C1-C4) können von allen benachbarten single-length Leitungen bedient werden. Jeder CLB Output kann durch mehrere Leitungen angetrieben werden.

Single-length Leitungen werden normalerweise benutzt, um Signale in einem lokalen Bereich zu leiten. Sie bieten eine gute Verbindung zum nächsten Block und eine hohe Flexibilität für komplexe Verdrahtungen, verursachen aber eine Verzögerung beim Durchgang durch eine Matrix.

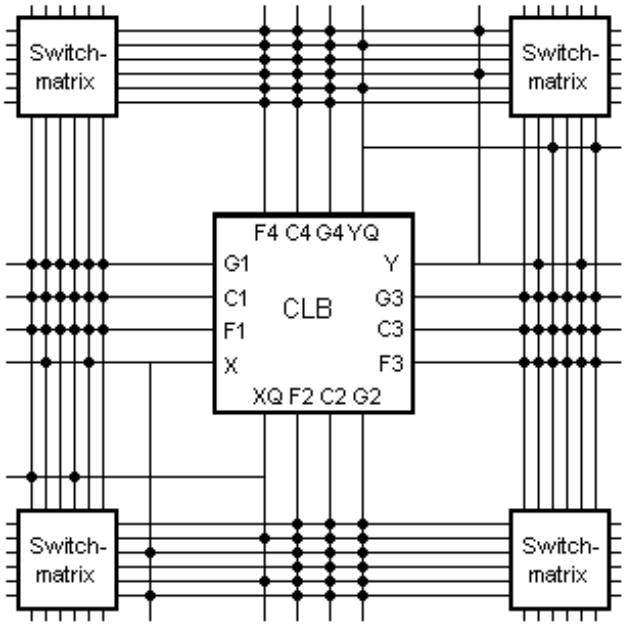


Abbildung 2.4 : CLB Verbindungen zur benachbarten Single-Length Leitungen

Die double-length Leitungen (Abbildung 2.5) bilden ein Gitter von Metallsegmenten, die die doppelte Länge der single-length Leitungen besitzen. Im allg. passiert eine double-length Leitung zwei CLBs, bevor sie in die nächste Matrix eingeht. Auch hier können alle CLB-Inputs von jeder benachbarten double-length Leitung gesteuert werden, während jedes CLB-Output von einer horizontalen oder vertikalen double-length Leitung erreichbar ist.

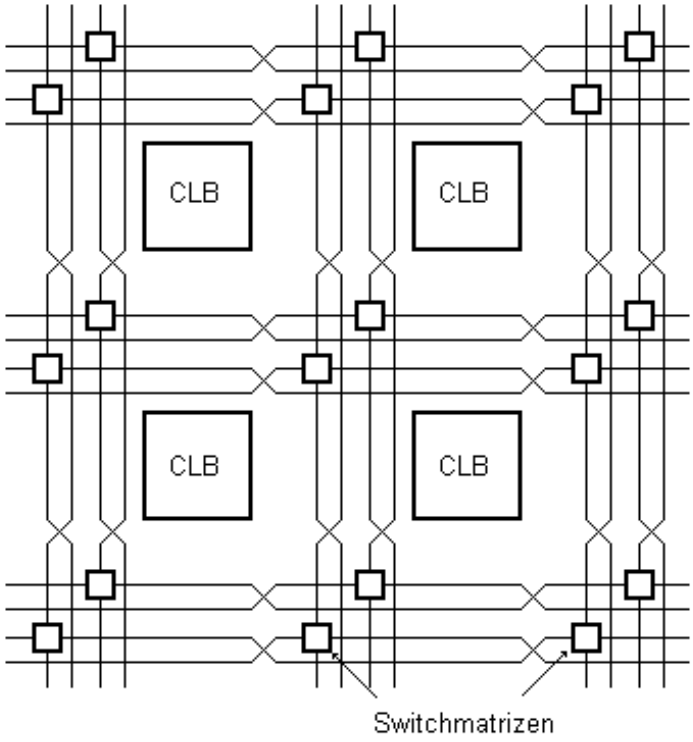


Abbildung 2.5 : Doubel-Length Leitungen

Double-length Leitungen bieten die effektivste Implementierung einer Mittelweg-Standverbindung.

Die Ordnung der lokalen Leitungen (single-length und double-length) sieht wie folgt aus:

- Um den Chip herum gibt es einen Ring aus vier double-length Leitungen (ohne Switchmatrizen). An der Stelle, wo die lokalen Leitungen einer Matrizenzeile bzw. -spalte am Ring enden, sind sechs Leitungen zu sehen, weil an diesem Punkt zwei Leitungen enden und zwei beginnen.
- Die Matritzenzeilen und -spalten werden durch eine Gruppe von 6 lokalen Leitungen (4 single-length und 2 double-length) verbunden. Die erste und letzte Leitung in jeder

Gruppe ist eine double-length Leitung. In der Region, wo eine Gruppe die CLB-Pins schneidet, sind acht lokale Leitungen zu vorhanden (die double-length Leitung der vor letzten Matrix).

- Die waagerechten lokalen Leitungen sind zeilenweise von oben nach unten und die senkrechten lokalen Leitungen spaltenweise von links nach rechts beginnend mit 0 durchnumeriert. Die Numerierung wird in den letzten Teil der Bezeichnung einer lokalen Leitung eingesetzt.

Bemerkung:
Manche lokale Leitungen erhalten dieselbe Bezeichnung. Das sind diejenigen vertikalen Leitungen mit derselben Numerierung, die sich in der gleichen Spalte in verschiedenen Zeilen befinden (auch für horizontale Leitungen).

Lange Leitungen

Lange Leitungen bilden ein Gitter aus Metallverbindungssegmenten, das über die gesamte Länge und Breite eines Chips geht. Einige waagerechte lange Leitungen können durch spezielle globale Puffer gesteuert werden, um Clock- und andere high fanout Kontrollsignale über den Array mit minimalen skew zu verteilen. Lange Leitungen sind für high fanout und zeitkritische Netzsignale vorgesehen. Jede lange Leitung hat am Mittelpunkt einen programmierbaren Splitterschalter, der die Leitung in zwei unabhängige Verdrahtungskanäle teilen kann.

CLB-Inputs können direkt durch einige benachbarte lange Leitungen gesteuert werden, während die CLB-Outputs nur durch tristate Puffer oder single-length Leitungen mit den langen Leitungen verbunden werden können.

In der ersten und letzten Zeile befinden sich jeweils zehn horizontale lange Leitungen, davon befinden sich vier über/unter den Wide-Decodern, vier zwischen den Decodern und den IOBs und zwei zwischen den IOBs und der ersten/letzten Zeile von Switchmatrizen. Jede weitere Zeile hat vier horizontale lange Leitungen, von denen sich zwei über den CLBs und zwei darunter befinden. Für die Bezeichnung von horizontalen

langen Leitungen sind die Leitungen einer Zeile von oben nach unten durchnumeriert (beginnend mit 0).

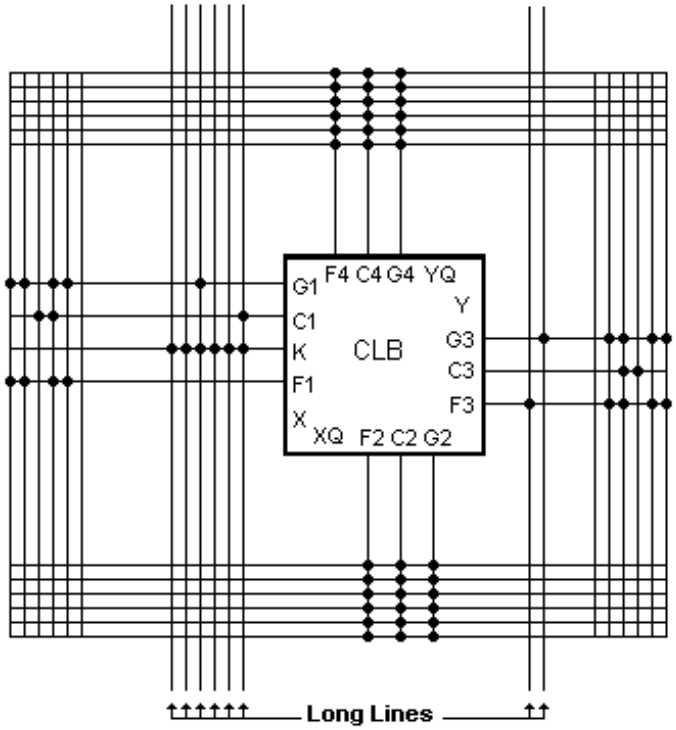


Abbildung 2.6 : Lange Leitung mit typische CLB Verbindungen

In der erste Spalte gibt es acht vertikale lange Leitungen, von denen sich vier links neben den Decodern und vier zwischen den Decodern und den IOBs befinden. Dagegen hat die letzte Spalte zwölf vertikale lange Leitungen, davon befinden sich vier rechts neben den Decodern, vier zwischen den Decodern und den IOBs, zwei direkt rechts neben den lokalen Leitungen und zwei direkt links neben den lokalen Leitungen.

In jeder anderen Spalte gibt es acht senkrechte lange Leitungen, zwei davon befinden sich direkt links neben den lokalen Leitungen und sechs rechts. Die Spalten sind von links nach rechts bei 0 beginnend numeriert.

Globale Netze

Globale Netze optimieren die Verteilung von clock und zeitkritischen Signalen. Vier Pad-gesteuerte (Primary Global) Netze bieten eine kürzere Verzögerung und geringfügigen Skew.

Alle anderen vier Pad-gesteuerte (Secondary Global) Netze haben durch stärkere Auslastung deutlich längere Verzögerung und mehr Skew.

Aus jeder Spalte haben die vier vertikalen langen Leitungen (links neben dem CLB) jeweils den Zugriff auf ein (Primary Global) Netz und auf alle (Secondary Global) Netze. Die globalen Netze umgehen Clock-Skew und potentielle Timingsprobleme. Der Benutzer muß diese Netze für alle zeitintensiven globalen Signalverteilungen benutzen.

Die Bezeichnung der globalen Netze sind nicht erforderlich, weil man nur den Pin des Zielpuffers angeben muß, `bufgs_bl.O` z.B. bezeichnet alle zu bufgs_bl führende globalen Netze.

Weitere Leitungen

Zu den jetzt beschriebenen Leitungen kommen noch Leitungen um die IOBs und die Spezialblöcke dazu, um die Pins dieser Blöcke mit anderen Pins und Leitungen zu verbinden. Die Bezeichnung dieser Leitungen ist gleich den der lokalen Leitungen, nur die letzte Nummer ist höher. Jedoch ist die Bezeichnung dieser Leitungen für die Verdrahtung nicht wichtig, weil man nur den Pin, der zu dieser Leitung gehört, benutzen muß.

2.2 Beschreibung des LCA-Formats

Diese Beschreibung basiert auf dem durch PPR von Xilinx erzeugte LCA-File COUNT.LCA (Anlage A).

2.2.1 Die Grundstruktur

Das Format enthält fünf wesentliche Teile; den Kopf, die Beschreibung der Netze, die Beschreibung der Blöcke, TimeGroup und Timespace Vereinbarungen und Angabe von internen Netzen.

Jede Zeile beginnt mit einem Schlüsselwort wie **`Addnet`**, **`NProgramm`**, **`Nameblk`**, **`Editblk`**, **`Base`**, **`Config`**, **`Equate`**, **`Endblk`**, **`System`** oder **`Intnet`**. Das LCA-Format kann auch Kommentarzeilen (Zeilen durch Semikolon eingeleitet) zur besseren Lesbarkeit enthalten.

Das LCA-Format beginnt und endet mit einer Kommentarzeile. In einer Anfangskommentarzeile wird beschrieben, um welche Datei es sich handelt, das dabei verwendete Design, das dazugehörende Programm und wann die Datei erstellt wurde. Beispiel:

```
;:count.lca(4002APC84-5),pprXilinx:ppr:5.2.0: 95/10/23,
1998/05/24 23:50:00
```

Die Endkommentarzeile bezeichnet nur das Ende des Files:

```
;: END OF FILE
```

2.2.2 Der Kopf

Der Kopf eines LCA-Formats besteht aus drei Zeilen, die das zu verwendende Design (Chip und Gehäuse) und den Speed-Grad festlegen.
Das Design enthält Informationen über den verwendeten Chip und das verwendete Gehäuse. Für prinzipiell gleiche Designs sind verschiedene Geschwindigkeiten der Signale möglich. Durch die Angabe des Speed-Grads werden die Eigenschaften, die die Signallaufzeiten beeinflussen, bestimmt. Z.B.:

```
Version 2
Design 4002APC84
Speed -5
```

Die erste Zeile des Kopfes beschreibt, um welche LCA-File Version es sich handelt. Die Version kann man durch die XACT-Option (*-lca ver=number)* einstellen, wobei *number* eine positive Ganzzahl sein muß. Wenn beispielsweise *number = 1* ist, dann wird das LCA-File in einem alten Format ausgegeben. Das alte LCA-File Format hat keine PIP Richtungsinformation, keine Versionsangabe und keinen Netznamen für PROGRAM und NPPROGRAM. Im Fall von number ≥ 2 ist, wird ein LCA-File mit Versionsangabe, PIP Richtungsinformation und Netznamen für PROGRAM und NPPROGRAM ausgegeben.

In den nächsten Zeilen kommen die Aussagen zum Design. Es handelt sich hier um den Designtyp 4002APC84. Dies bedeutet, daß es sich hier um einen Xilinx-FPGA aus der 4000er Familie handelt, mit (8 × 8) CLBs und einem Gehäuse mit 84 Pins. Der zu verwendende Speed-Grad ist -5.

2.2.3 Vereinbarung der Netze

Nach dem Kopf eines LCA-Formates folgt die Vereinbarung der Netze. Die Vereinbarung eines Netzes besteht aus zwei Hauptschritten:

- zuerst wird das benötigte Netz erstellt,
  (welches Netz also die verschiedene Blöcke miteinander verbinden soll)
- danach folgt die Verdrahtung.
  (über welche Verdrahtungsressourcen es geschehen soll)

Dann können noch zusätzliche Pins zu den Netzen addiert werden, dabei können zwei Netze zusammengefaßt werden.

2.2.3.1 Erstellung der Netze

Die Erstellung der Netze im LCA-Format von Xilinx sieht wie folgt aus:

```
Addnet B<1> PAD30.O JE.G1 JC.F3 JB.F4 JC.YQ
```

Diese Art von Zeilen sagen aus, daß ein Netz mit dem Namen `B<1>` erstellt werden soll, das den Pin `O` des IOBs `PAD30`, den Pin `G1` des CLBs `JE`, den Pin `F3` des CLBs `JC`, den Pin `F4` des CLBs `JB` und den Pin `YQ` des CLBs `JC` verbindet.

2.2.3.2 Verdrahtung der Netze

Nachdem die Erstellung eines Netzes abgeschlossen ist, wird seine Verdrahtung vorgenommen. Die Verdrahtung, als Ergebnis des Programms PPR, des oben erwähnten Netzes `B<1>` wird im LCA-Format wie folgt beschrieben:

```
Addnet B<1> PAD30.O JE.G1 JC.F3 JB.F4 JC.YQ
NProgram B<1> PAD30.O:row.I.long.3
col.G.local.2:row.I.long.3-L JG.24.1.1 JG.24.1.19
JE.24.1.10 JE.24.1.19 JE.G1:col.E.local.2
JE.24.1.16 JE.24.1.19
NProgram B<1> JD.24.1.10 JD.24.1.19 JC.YQ:row.J.local.5
JC.F3:col.D.local.5 JC.YQ:col.D.local.5 JB.F4:row.J.local.1
JC.YQ:row.J.local.0
```

Beginnend mit dem Befehl `NProgram` wird beschrieben, wie das Netz geführt wird. Nach dem Befehl folgt direkt der Name des betreffenden Netzes. Danach folgt die Angabe der Verdrahtung der Netze und die Angabe der zu schaltenden PIPs oder Switchmatrizen. Die Verbindung über PIPs wird durch den PIP-Name realisiert, der direkt zwei Pins, ein Pin und eine Leitung oder zwei Leitungen verbindet.

In diesem Beispiel beginnt die zweite Zeile mit NProgram, gefolgt vom Netznamen B<1>. Danach folgt die Zeichenkette PAD30.O:row.I.long.3. Diese bezeichnet den PIP, der den Pin O des IOBs PAD30 mit der vierten waagerechten langen Leitung der 9 Zeile verbindet. Während die nächste Zeichenkette col.G.local.2 :row.I.long.3-L den PIP bezeichnet, der die dritte senkrechte lokale Leitung der siebten Spalte mit der vierten waagerechten langen Leitung der neunten Zeile verbindet. Das Anhängen eines L (long) an diesem PIP bedeutet, daß das Signal von der langen Leitung zur der lokalen Leitung übergeht, beim Anhängen eines S (short) ist dies umgedreht.

Bei Verbindungen über Switchmatrizen ist ein paarweises Auftreten notwendig. Das findet man z.B. in der Zeile mit dem zweiten (NProgram) Befehl durch das Paar JD.24.1.10 JD.24.1.19 wieder. Aus der Bezeichnung der Switchmatrizen und ihrer Pins (Abschnitt 2.1.9) sieht man, daß es sich bei JD.24.1.10 um einen Anschluß an die Switchmatrix JD.24.1 handelt. Es ist von oben gesehen der fünfte Anschluß an der rechten Kante. Dieser führt zu einer Leitung mit der Bezeichnung row.J.local.5, die fünfte waagerechte lokale Leitung in der zehnten Zeile. Die Zuordnung der Leitung ist durch den Namen nicht eindeutig, wird dies aber durch die Lage zur Switchmatrix.

Durch die Zeichenkette JD.24.1.10 wird ausgesagt, daß die Leitung row.J.local.5 über die Switchmatrix mit einer anderen Leitung verbunden werden soll. Diese folgt in der nächsten Zeichenkette JD.24.1.19.

Aus den Bezeichnungen der Leitungen (Abschnitt 2.1.11) ist bekannt, daß die langen Leitungen in zwei unabhängige Verdrahtungskanäle durch einen programmierbaren Splitterschalter geteilt sind und, daß diese zwei Leitungen den gleichen Namen haben. Die Frage ist. Wie wird die Verbindung der zwei Teile einer Leitung untereinander im LCA-Format beschrieben? Zunächst folgendes Beispiel für eine Netzvereinbarung:

Addnet CTRL5 PAD6.I2 ID.C3 HD.C3

NProgram CTRL5 ID.C3:col.E.long.1
HD.C3:col.E.long.1 *row.F.local.1:col.E.long.1*
PAD6.I2:col.E.long.1

In der dritten Zeile ist die Zeichenkette *row.F.local.1:col.E.long.1* zu sehen. Diese Zeichenkette ist der Name des Splitterschalters, der die lange Leitung col.E.long.1 teilt. Durch diese Zeichenkette wird die Verbindung der Leitungsstücke einer lange Leitung beschrieben.

Der Name eines Splitterschalters besteht aus zwei durch Doppelpunkt getrennte Teile. Der zweite Teil ist die Bezeichnung der betreffenden langen Leitung. Der erste Teil ist row.F.local.1 für alle senkrechten langen Leitungen und col.F.local.1 für alle waagerechten. Diese beiden lokalen Leitungen existieren in der Realität nicht, weil es wie schon erwähnt, in der mittleren Zeile bzw. Spalte (bei 4002A F und bei 4003A G) es keine lokalen Leitungen gibt.

Der erste Teil ist immer die Bezeichnung der lokalen Leitung mit der Nummer 1 der mittleren Zeile/Spalte (für senkrechte Leitung / für waagerechte Leitung).

Bemerkung:
Der Befehl NProgram kann maximal bis zu sechs PIPs verbinden, deshalb sind, z.B. für ein Netz, das über 14 PIPs geht, drei NProgram Befehle nötig.

2.2.3.3 Addieren eines Pins

In manchen Fällen wird von einem Netz verlangt, eine große Anzahl von Pins zu verbinden, was problematisch ist, da der Befehl Addnet maximal 10 Pins verbinden kann.

Durch den Befehl Addpin können zu einem erstellten Netz weitere Pins zugefügt werden:

```
Addnet BUFGS HE.K JE.K GE.K HD.K ID.K JD.K GC.K
HC.K JC.K GB.K
Addpin BUFGS HB.K IB.K bufgs_bl.O
```

Wenn aber mehr als zehn Pins ergänzt werden sollen, muß ein zweiter Addpin Befehl benutzt werden, weil ebenfalls nur 10 Parameter möglich sind.

2.2.3.4 Zusammenfassung von Netzen

In diesem Abschnitt wird gezeigt, wie die Zusammenfassung von Netzen im LCA-Format dargestellt wird:

```
Addnet CTRL2_2 PAD42.I2 JC.C2 IB.F3
NProgram CTRL2_2 JC.C2:row.K.local.1 IB.F3:col.C.local.6
KC.24.1.6 KC.24.1.5 KE.24.1.23 KE.24.1.6
PAD42.I2:row.K.local.1
Addnet CTRL2_1 PAD42.I1 JE.C1 JD.F3
NProgram CTRL2_1 JE.C1:col.E.local.1 JD.F3:col.E.local.1
PAD42.I1:col.E.local.0
Beginpath CTRL2
Pathnet CTRL2_1 CTRL2_2
Endpath
```

Es werden hier zwei Netze (CTRL2_1 und CTRL2_2) definiert. Die Zusammenfassung wird durch die letzten drei Zeilen dargestellt, in denen zuerst ein gemeinsamer Name (CTRL2) ausgewählt wird. Danach werden die erwählten Netze festgelegt und am Ende wird der Vorgang mit Endpath abgeschlossen.

2.2.4 Programmierung der Blöcke

Im folgenden wird dargestellt, wie die Programmierung der Blöcke im LCA-Format erfolgt. Sie folgt aus die Vereinbarung der Netze.
Die Programmierung besteht aus zwei von einander unabhängigen Teilen: Das Umbenennen der Blöcke und die Konfiguration der Blöcke.

2.2.4.1 Umbenennen der Blöcke

In der Programmierung eines FPGAs kann eine Vergabe von expliziten Namen an die Blöcke vorgenommen werden. Im LCA-Format wird das Umbenennen durch die Zeile dargestellt, die mit dem Befehl `Nameblk` beginnt.

```
Nameblk HE C<0>
```

Diese Zeile besagt, daß das CLB `HE` zu `C<0>` umbenannt wurde. Das ist dann die Bezeichnung, die man in den Gruppenvereinbarungen wiederfinden wird.

Das Umbenennen von IOBs sieht genauso aus:

```
Nameblk PAD8 $1I132/PAD
```

2.2.4.2 Konfiguration der Blöcke

Um die Konfiguration eines Blocks (CLB, IOB und TBUF) zu verstehen, muß man die Block-Architektur unter die Lupe nehmen. Ein typischer Abschnitt des LCA-Formats, das die Konfiguration eines CLBs darstellt, sieht wie folgt aus:

```
Editblk  HJ
Base  FG
Config  F4:F4 I G2:G2I G3:G3I CIN: COUT: X: Y:H
XQ:QX YQ: FFX:K:RESET   FFY:RESET  DX:DIN  DY:
F:F2:F4:F3:F1  G:G4:G1:G3:G2 H:G:F  H1:  DIN:C2
SR:  EC:  RAM:  CDIR:
Equate  F=(~F4*~F2*(~F1*~F3))
Equate  G=(~G1*~G4*(~G2*~G3))
Equate  H=(F+G)
Endblk
```

eines IOBs:

```
Editblk  PAD 30
Base  IO
Config  INFF: I1: I2: OUT:O  PAD:  O:  OSPEED:SLOW
Endblk
```

Die Konfiguration eines Blocks fängt mit dem Befehl `Editblk` an, der festlegt, welcher Block (hier CLB `HJ` oder IOB `PAD30`) zu konfigurieren ist, indem man die Originalbezeichnung des Blocks nach dem Befehl einsetzt.

Durch den Befehl `Base` wird die Basiskonfiguration eines Blocks festgelegt. Tabelle 2.1 präsentiert die zulässigen Optionen für den Base Befehl.

Basisname	Nutzung
IO	Nur für IOBs
F	Eine Funktion von bis zu vier Variablen bei XC2000 FPGAs und fünf Variablen bei XC3000 FPGAs

FG	Zwei Funktionen (F und G), jede bis zu drei Variablen bei XC2000 FPGAs und vier Variablen bei XC3000 FPGAs oder zwei unabhängige Funktionen, jede vier Variable bei XC4000 FPGAs
FGM	Wie FG, aber die Outputs der zwei Funktionen werden zusammen multipliziert und durch den Input B bei XC2000 FPGAs und Input E bei XC3000 FPGAs kontrolliert.

Tabelle 2.1 : Optionen des Base-Befehls

Die `Base FG` ist die einzige Basis, die für die CLBs der 4000er FPGA - Familie in Frage kommt.

In der nächsten Zeile, die mit dem Befehl `Config` anfängt, werden die Logik und die (inneren und äußeren) Verbindungen eines Blocks konfiguriert. Der Syntax des Befehls ist (`Config` *`Öse`*`:`*`setting`*). *Ösen* sind Blockcharaktaristiken, die auf eine von mehreren Optionen gesetzt werden kann. Diese Optionen sind in Tabelle 2.2 aufgelistet.

Z.B. führt `Config X:F` dazu, daß das Ergebnissignal der logischen Funktion `F` direkt auf den Ausgang `X` durchgeschaltet wird. Anders bei `Config X:QX DX:F,` dort wird das Signal `F` nicht direkt an den Ausgang `X` weitergegeben, sondern ist Eingangssignal für ein Flip-Flop. Erst das Ausgangssignal dieses Flip-Flops wird an den Ausgang `X` weitergegeben. Außerdem wird durch `Config EC: CDIR: RAM:` angegeben, daß die Signale für Clock und Ramspeicher genutzt werden.

Öse	**Optionen**
CLB	
X	F oder H
Y	G oder H
XQ	QX oder DIN
YQ	QY oder EC

H1	C1, C2, C3 oder C4
DIN	C1, C2, C3 oder C4
SR	C1, C2, C3 oder C4
EC	C1, C2, C3 oder C4
DX	DIN, F, G oder H
DY	DIN, F, G oder H
FFX	K, NOT, SET, RESET, SR und EC
FFY	K, NOT, SET, RESET, SR und EC
RAM	F, G oder FG
G2	G2I oder COUTT0 (Carry Logik)
G3	G3I oder CIN (Carry Logik)
F4	F4I oder CIN (Carry Logik)
CDIR	UP oder DOWN (Carry Logik)
CIN	CINI (Carry Logik)
COUT	COUTI (Carry Logik)
IOB	
OUT	O, OQ, NOT, SET, RESET, OK und OKN
INFF	SET, RESET, IK, IKNOT und DELAY
PAD	PULLUP, PULLDOWN und FAST
I1	I, IQ oder IQL
I2	I, IQ oder IQL
TRI	T und NOT
O	GND (IOB internal ground)
OSPEED	FAST, MEDFAST, MEDSLOW, SLOW
TBUF	
TBUF	WAND, WANDT und WORAND
I	GND

Tabelle 2.2 : XC4000 Öse-Optionen

Der nächste Teil der Konfiguration, in dem die verwendeten logischen Funktionen angegeben werden, ist:

```
Equate F=(~F4*~F2*(~F1*~F3))
Equate G=(~G1*~G4*(~G2*~G3))
Equate H=(F+G)
```

Durch die Verwendung des Befehls `Equate` werden die logischen Funktionen gebildet. Der Boolesche Ausdruck kann eine logische Funktion von vier Eingängen

(XC4000 Base FG) bilden. Welche Funktionen durch die einzelnen Symbole gemeint sind, wird in Tabelle 2.3 dargestellt.

Symbol	Funktion
~	NOT
*	AND
@	XOR
+	OR

Tabelle 2.3 : Logische Operatoren

Schließlich wird am Ende mit dem Befehl `Endblk` die Konfiguration eines Blocks abgeschlossen.

2.2.5 Weitervereinbarung

In den vom Plazierungs- und Verdrahtungsprogramm PPR erzeugten Dateien findet man außerdem noch Zeilen mit Angaben über die maximale erlaubte Verzögerung zwischen der festgelegten Menge von Pfaden eines Designs:

```
System Xdelay Timegroup ffs -FF C<0> C<2> C<1> C<3>
System Xdelay Timegroup ffs -FF E<0> E<2> E<1> E<3>
System Xdelay Timegroup pads -IOB $1I128/PAD $1I132/PAD
System Xdelay Timegroup pads -IOB $1I136/PAD $1I150/PAD
System Xdelay TimeSpec DEFAULT_FROM_FFS_TO_FFS ffs ffs 0.0
System Xdelay TimeSpec DEFAULT_FROM_PADS_TO_FFS pads ffs 0.0
System Xdelay TimeSpec DEFAULT_FROM_FFS_TO_PADS ffs pads 0.0
```

Zuerst werden durch das Kommando **`System Xdelay Timegroup,`** durch die Definition einer Gruppe (TimeGroup) von Anfangs- und Endpunkte, eine Menge von Pfade identifiziert. Die Anfangs- und Endpunkte können Flip-Flops (FFs), I/O-Pads

(PADS) oder XC4000 RAMs sein. Es werden hier die Blocknamen, die durch das Umbenennen erzeugt worden, benutzt.

Danach wird die Standardverzögerung zwischen den einzelnen Gruppen durch den Befehl **`System Xdelay TimeSpec`** festgelegt. Die „ 0.0 “ in diesem Beispiel steht für „ auto “.

In der Beschreibung von Designs treten noch Zeilen wie z.B.

```
Intnet JC F SYNC_1/AND9_REG_ff_2
Intnet JC G SYNC_1/AND10_REG_ff_2
Intnet JC H SYNC_1/BUF4_REG_ff_2
Intnet ID F ASYNC/BUF4_SAT_ffy_1
Intnet ID G ASYNC/BUF2_SAT_ffx_0

Intnet PAD64 PAD $1I128/PAD
Intnet PAD6 PAD $1I132/PAD
Intnet PAD55 PAD P_A/PAD<1>
Intnet PAD56 PAD P_A/PAD<2>
```

auf.

Es handelt sich hier um die Bezeichnung von internen Netzen. In der ersten Zeile bekommt das F-Signal von CLBs `JC` den Namen `SYNC_1/AND9_REG_ff_2,` während in der letzten Zeile das PAD-Signal von IOBs `PAD56` den Namen `P_A/PAD<2>` bekommt.

Die Bezeichnung von internen Netzen bei CLBs zeigt wie das Signal entstanden ist. Während die Bezeichnung der internen Netze bei IOBs dem Namen der Blöcke entspricht.

3. Beispiel für ein manuell erzeugtes LCA-File

Im vorigen Abschnitt wurde das LCA-Format der Xilinx-FPGAs der 4000er Familie beschrieben. Als nächstes wird das LCA-File für eine einfache Schaltung, die auf Basis der gegebenen Beschreibung erstellt wurde, dargestellt und ausgewertet. Es wird hierbei nicht auf eine optimale Plazierung bzw. Verdrahtung geachtet. Im vierten Abschnitt wird dann ein Lösungsansatz für eine optimale Verdrahtung präsentiert.

3.1 Beschreibung des entry Designs

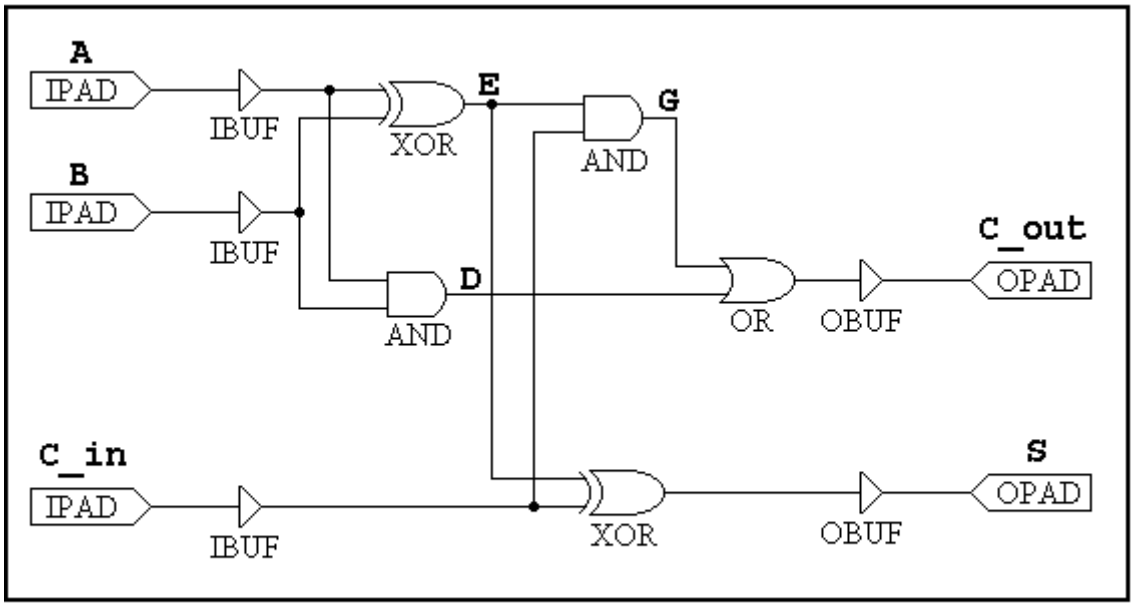


Abbildung 3.1: Die ASIC-Schaltung eines Volladdierers

In der Abbildung 3.1 ist die Schaltung eines Volladdierers angegeben. Die Schaltung besitzt 15 Logikelemente (2 XORs, 2 ANDs, 1 OR, 3 IPADs, 3 IBUFs, 2 OPADs und 2 OBUFs). Der Addierer besitzt drei Eingangssignale A, B und C_in (Carry = Übertrag) und zwei Ausgangssignale S (summe) und C_out. Es werden intern noch drei Signale erzeugt (E, D und G). Diese Schaltung wurde mit Hilfe des Viewdraw Programms erzeugt und mit ProSim und ProWave geprüft. Der Volladdierer addiert die Signale A und B unter Berücksichtigung des Übertrages C_in und erzeugt die Summe S und den Übertrag C_out.

3.2 Das manuell erzeugte LCA-File

```
;: VOLLADD.LCA (4002APC84-5), Al-Shawi(per Hand),
1998/05/14 12:04:47

Version 2
Design 4002APC84
Speed -5

Addnet A_input PAD62.I2 BB.F2 BB.G2
```

```
NProgram A_input row.C.local.0:BB.G2 row.C.local.0:BB.F2
row.C.local.0:PAD62.I2
Addnet B_input PAD2.I2 BB.F3 BB.G3
NProgram B_input col.C.local.6:BB.F3 col.C.local.6:BB.G3
BC.24.1.12 BC.24.1.5 col.C.local.7:PAD2.I2
Addnet C_in PAD64.I2 BB.F1 BB.G1
NProgram C_in col.B.local.3:BB.F1 col.B.local.3:BB.G1
col.B.local.3:PAD64.I2
Addnet C_out BB.X PAD63.O
NProgram C_out col.B.local.5:PAD63.O col.B.local.5:BB.X
Addnet Summe BB.Y PAD1.O
NProgram Summe row.B.local.2:PAD1.O row.B.local.2:BB.Y
Nameblk BB VADD
Editblk BB
Base FG
Config  X:F Y:G XQ: YQ: FFX:RESET FFY:RESET DX: DY:
F:F1:F3:F2 G:G1:G3:G2 H: H1: DIN: SR: EC: RAM: CARRY: CIN:
COUT: CDIR:
Equate F = (((F3@F2)*F1)+(F3*F2))
Equate G = ((G3@G2)@G1)
Endblk
Nameblk PAD62 A_input
Editblk PAD62
Base IO
Config INFF: I1: I2:I O: OUT: PAD: TRI: OSPEED:SLOW
Endblk
Nameblk PAD63 C_out
Editblk PAD63
Base IO
Config INFF: I1: I2: O: OUT:O PAD: TRI: OSPEED:SLOW
Endblk
Nameblk PAD64 C_in
Editblk PAD64
Base IO
Config INFF: I1: I2:I O: OUT: PAD: TRI: OSPEED:SLOW
Endblk
Nameblk PAD1 B_input
```

```
Editblk PAD1
Base IO
Config INFF: I1: I2:I O: OUT: PAD: TRI: OSPEED:SLOW
Endblk
Nameblk PAD2 Summe
Editblk PAD2
Base IO
Config INFF: I1: I2: O: OUT:O PAD: TRI: OSPEED:SLOW
Endblk

Intnet PAD1 PAD B_input
Intnet PAD2 PAD Summe
Intnet PAD62 PAD A_input
Intnet PAD63 PAD C_out
Intnet PAD64 PAD C_in

;:END OF FILE
```

3.3 Erläuterung

Für den Volladdierer (Abbildung 3.1) war die Vereinbarung von fünf Netzen, eines CLBs und fünf IOBs nötig.

Die Netze:

- Das Netz „A_input“ repräsentiert den Verlauf von Signal A. Das Signal A wird durch den Inputpad PAD62 eingegeben und dann zu den Eingängen F2 und G2 des CLBs BB übergeleitet.

- Das Netz „B_input“ repräsentiert den Verlauf von Signal B. Das Signal B wird durch den Inputpad PAD1 eingegeben und dann zu den Eingängen F3 und G3 des CLBs BB übergeleitet.

- Das Netz „C_in“ repräsentiert den Verlauf von Signal C_in. Das Signal C_in wird durch den Inputpad PAD63 eingegeben und dann zu den Eingängen F1 und G1 des CLBs BB übergeleitet.

- Das Netz „C_out“ repräsentiert den Verlauf von Signal C_out. Das Signal C_out wird durch den CLB BB erzeugt und über den Ausgang X an den Outputpad PAD64 weitergegeben.

- Das Netz „Summe“ repräsentiert den Verlauf von Signal S. Das Signal S wird durch den CLB BB erzeugt und über den Ausgang Y an den Outputpad PAD2 weitergegeben.

Für die interne Signale D, E und G waren keine Netze nötig, weil sie im CLB intern verarbeitet werden.
Die Verdrahtung der Netze verläuft nur über lokale Leitungen, so daß keine langen Strecken gebrauch werden.

Die CLBs:

Das CLB BB bekommt den Namen VADD und die Basiskonfiguration FG. In VADD sind zwei Gleichungen F und G definiert. Die Gleichung F berechnet aus A (Input F2), B (Input F3) und C_in (Input F1) die Summe S (Output Y). Die Gleichung G berechnet den Übertrag C_out (Output X) aus A (Input G2), B (Input G3) und C_in (Input G1).

Die IOBs:

Die fünf IOBs bekommen jeweils den Namen von den jeweiligen Netzen, die sie mit dem CLB verbinden und die Basiskonfiguration IO. Bei den drei Inputpads wird der Pin I2 als Eingang festgelegt. Dies bedeutet, daß I2 die von außen kommenden Signal zu den CLB BB leitet. Bei den Outputpads wird der Pin O als Ausgang festgelegt. Die Geschwindigkeit bei allen IOBs wird auf SLOW eingestellt.

Am Ende bekommen die internen Padsignale auch die dazugehörigen IOB-Namen.

3.4 Auswertung

Was das manuell erzeugte LCA-File bewirkt, wurde mit dem XACT-Design-Editor xde, der ebenfalls Dateien im LCA-Format als Eingabe- und Ausgabedateien nutzt, geprüft, und durch den Vergleich mit dem File volladde.lca (Anlage B), das durch PPR erzeugt wurde, ausgewertet.

1. Der Kopf stimmt völlig überein.

2. Die Anzahl der Netze ist identisch, aber die Reihenfolge und Benennung ist nicht gleich. Auch die Verdrahtung ist anders, weil die Plazierung nicht gleich ist.

3. Die Anzahl der CLBs ist auch gleich, aber die Plazierung ist sehr unterschiedlich (hier wurde das CLB JB ausgewählt).

4. Die Auswahl der IOB ist nicht identisch, bedingt durch die Auswahl des CLB.

Im Allgemeinen kann man feststellen, daß die Unterschiede nicht die Bedingungen für einen korrekten LCA-Files verletzen (weil z.B. die Wahl eines Netznamens frei ist) und daß die Beschreibung des LCA-Formats erfolgreich durchgeführt wurde.

4. Ein durch Leistung und Verdrahtbarkeit gesteuerter Verdrahtungsalgorithmus

4.1 Einleitung

Die Verdrahtung von FPGAs erfolgt über vorgefertigte Leiterbahnsegmente, die durch programmierbare Schalter miteinander verbunden werden können. Aufgrund dieser Struktur unterscheidet sich das Verdrahtungsproblem für FPGAs stark von demjenigen für Makrozellen und Standardzellen, denn man kann die Leitung nicht innerhalb vorgegebener Kanäle oder Flächen frei wählen. Das führt dazu, daß vorhandene

Verdrahtungsprogramme nicht direkt auf dieses spezielle Verdrahtungsproblem angewendet werden können bzw. die Ergebnisse nicht zufriedenstellend sind.

In den letzten Jahren wurden viele Ansätze vorgeschlagen, um eine detaillierte Verdrahtung von FPGAs zu realisieren. CGE[5] war der erste Ansatz, der sich die Verdrahtung von (SRAM) FPGAs vorgenommen hat. Es nutzt ein Verfahren [19], das ursprünglich für Standardzellen entwickelt wurde. Das Verfahren besteht aus zwei Schritten, einem Global- und einem Endverdrahtungsschritt. Beim Globalverdrahtungsschritt werden den Netzen Kanäle zugeordnet, aus denen im Endverdrahtungsschritt die Leiterbahnsegmente ausgewählt werden.
Während es bei der Endverdrahtung darum geht, die endgültige Zuordnung zu den Leiterbahnsegmenten vorzunehmen, kommt es bei der Globalverdrahtung darauf an, den Netzen Kanäle so zuzuordnen, daß das Zeitverhalten der Verdrahtung möglichst gut ist und eine Endverdrahtung existiert. Letzteres kann erreicht werden, indem die Kanäle möglichst gleichmäßig ausgenutzt werden.

Neben den zuvor genannten existieren mehrere Ansätze für eine graphenunterstützte Lösung des Verdrahtungsproblems von FPGAs [1],[28] und [29].

In [15] basiert die Verdrahtung auf einem simulierten Steinerspanning-Baum. Außerdem wurde in [6] einen auf simulated-evolution basierten Router für (SRAM) FPGAs präsentiert. Diese Lösung zeichnet eine beachtliche Reduzierung der gebrauchten Verdrahtungsleitungen auf Kosten von längeren Leiterbahnverzögerungen aus. Eine andere neue Lösung wurde in [24] und [25] vorgeschlagen, um 2-D FPGA Architekturen zu verdrahten.

Alle diese erwähnten Lösungen fokussieren hauptsächlich die Minimierung der Kanaldichte als ein Weg, um die Verdrahtbarkeit eines Design zu verbessern. Eine Ausnahme war die in [8] vorgeschlagene Lösung, wo man den Limit_bumpin Algorithmus [32] angewendet hat, um die Pufferzeiten für die durch Leistung gesteuerte Verdrahtung von FPGAs zu verteilen.

In dieser Arbeit wird ein neuer durch Leistung und Verdrahtbarkeit gesteuerter, Verdrahtungsalgorithmus (*FPGA-Ablaufverfolger*) für FPGAs präsentiert. Die Ziele des vorgeschlagenen Verdrahtungsalgorithmus sind:

1. *Verbesserung der Verdrahtbarkeit eines Designs.*
   (d.h. die maximale gebrauchte Verdrahtungskanaldichte zu minimieren)

2. *Verbesserung der gesamten Leistung eines Designs.*
   (d.h. Minimierung der gesamten Pfadverzögerung)

Die Verdrahtung wird in zwei Schritten durchgeführt:

- *Anfangsverdrahtung*
- *Anwenden des Algorithmus Auflösen_und_Wiederverdrahten*

Während der Anfangsverdrahtung werden die Netze sequentiell verdrahtet, in Abhängigkeit davon, wie kritisch die Netze sind und von ihrer Verdrahtbarkeit. Im zweiten Schritt wird ein Zweiphasen Algorithmus (*Auflösen_und_Wiederverdrahten*) benutzt, um die Verletzungen der Verdrahtungsressourcen, beispielsweise wenn zwei Netze die gleichen Leitungssegmente benutzen, und um die Verletzung der Zeitbeschränkung aufzuheben.
Der Algorithmus Auflösen_und_Wiederverdrahten hebt die Verletzungen durch die Benutzung von auf simulated-evolution basierte Optimierungstechniken auf. Diese Technik wurde bereits erfolgreich auf die verschiedenen CAD Applikationen (Partitionierung, Plazierung und Verdrahtung) angewendet [11], [16] und [21].

In dieser Arbeit werden zwei Kostenfunktionen vorgeschlagen, welche die Netze auswählen, die aufgelöst werden sollen. Die Kostenfunktionen berücksichtigen alle Informationen der Pfadverzögerung und der Verdrahtbarkeit.

4.2 Modelle und Problemformulierung

4.2.1 Die FPGA Architektur

In Abbildung 4.1 ist die typische (SRAM) FPGA-Architektur zu sehen. Diese besteht, wie bereits erwähnt, aus drei Hauptkomponenten:

- Konfigurierbare I/O Blocks (IOB)
- CLB
- Verbindungsressourcen

Die Verbindungsressourcen bestehen aus Leiterbahnsegmenten und programmierbaren Schaltern. Die Verdrahtungskanäle sind horizontal und vertikal angeordnet (vgl. Abbildung 1.2). Ein CLB-Pin kann durch die programmierbaren Schalter in der Verbindungsmatrix mit den Leiterbahnsegmenten von beliebigen Verdrahtungskanälen verbunden werden. Leiterbahnsegmente können durch die Nutzung von programmierbaren Schaltern, die sich in den Switchmatrizen befinden, gemischt werden. Dadurch können längere Verbindungen gebildet werden.

Ideal wäre es, wenn man die Verbindungsmatrizen und Switchmatrizen vollflexibel gestalten könnte, was aber zu einer große Steigerung der Komplexität der FPGA-Architektur führen würde (Je größer die Flexibilität umso größer die Leitungsverzögerung).

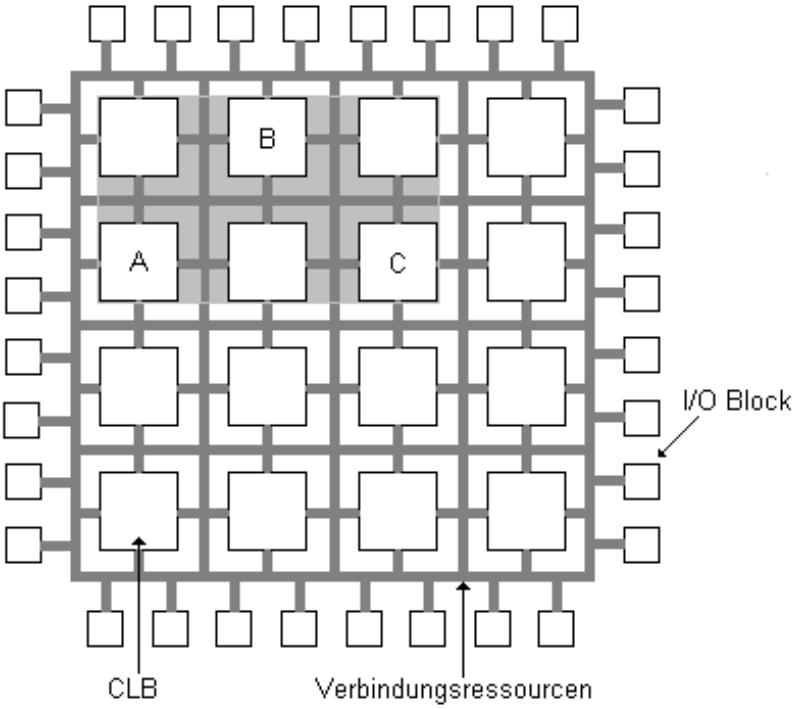


Abbildung 4.1 : Ein Flächenplan eines FPGAs

In [20] wurde bereits bewiesen, daß mit einer Switchmatrix-Flexibilität von drei (Fs = 3) und einer 100%igen Verbindungsmatrix-Flexibilität eine akzeptable Leitungsverzögerung und Verdrahtungsressourcen-Flexibilität erreicht werden kann. Deshalb besteht die Architektur, die in der vorliegenden Diplomarbeit in Betracht gezogen wird, aus Switchmatrizen mit der Flexibilität (Fs = 3) und vollflexible Verbindungsmatrizen.

4.2.2 Das Graphenmodell

Die Verbindungsressourcen werden als Graph modelliert, in der jeder Knoten ein Leitungssegment oder ein CLB-Pin repräsentiert. Die programmierbaren Schalter werden im Graphen als Kanten dargestellt. Diese Darstellung ist unabhängig vom speziellen Typ des FPGAs und ist somit geeignet, Algorithmen und Programme zu entwickeln, die sich leicht an unterschiedliche FPGAs anpassen lassen.
Abbildung 4.2 zeigt die drei Pins (*P1, P2* und *P3*) eines CLBs, die durch die drei Knoten (*VP1, VP2* und *VP3*) repräsentiert werden.

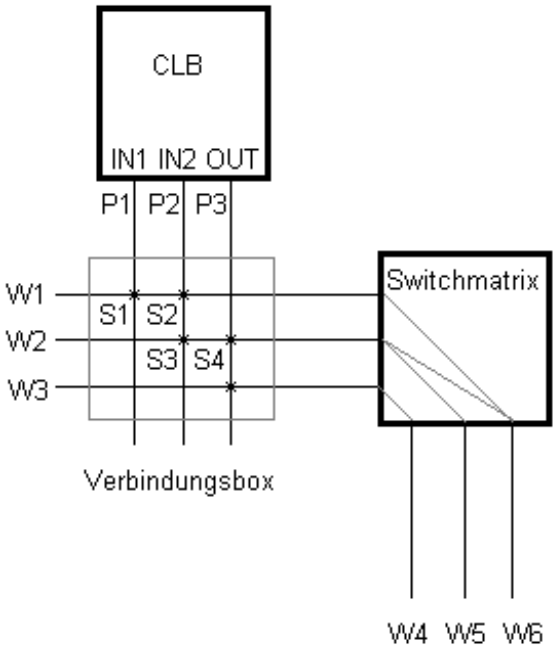


Abbildung 4.2 (a) : Die Verbindungsstruktur

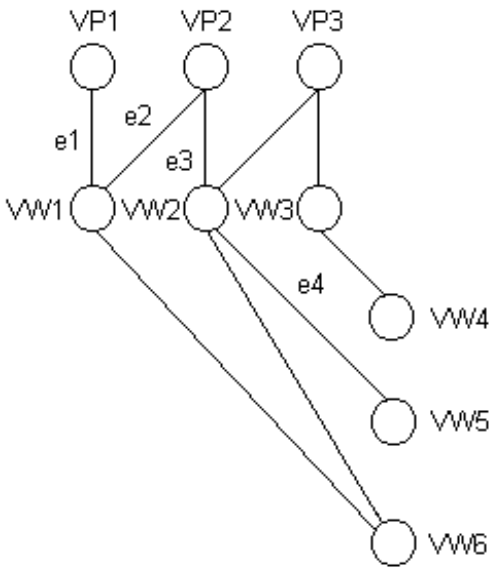


Abbildung 4.2 (b) : Das Graphenmodell

Weiterhin zeigt die Abbildung 4.2 einen horizontalen Verdrahtungskanal mit den Leitungssegmenten (*W1, W2* und *W3*), einen vertikalen Verdrahtungskanal mit den Leitungssegmenten (*W4 ,W5* und W6), eine Verbindungsmatrix mit fünf Schaltern und

eine Switchmatrix mit vier Schaltern. Jedes Leitungssegment ist einem Knoten zugeordnet. Wenn zwischen zwei Pins und/oder zwei Leitungssegmenten ein Durchlauftransistor existiert, wird eine Kante zwischen den zwei dazugehörigen Knoten addiert. Da zwischen Pin *P1* und Segment *W1* ein Verbindungsschalter existiert, sind die Knoten *VP1* und *VW1* mit einer Kante *e1* miteinander verbunden.

Bemerkung:
Es ist nicht erlaubt, die Verbindungsmatrix für den Wechsel zwischen den Leitungssegmenten zu benutzen. Z.B. ist in Abbildung 4.2(a) eine Verbindung Zwischen *P1* und *W5* durch die Aktivierung von Schalter *S1, S2, S3* und *S4* nicht erlaubt. Solche Verbindungen bezeichnet man als illegal.

4.2.3 Problemdefinition

Das leistungsgesteuerte FPGA-Verdrahtungsproblem wird wie folgt definiert:

„Gegeben sei eine CLB Netzliste mit ihrer Plazierung und ihren Zeitbeschränkungen. Es soll eine Verdrahtung aller Netze durchgeführt werden, so daß die maximale Verdrahtungskanaldichte minimiert und die Zeitbeschränkungen eingehalten werden“.

Das Verdrahtungsproblem wird als die Feststellung einer Menge von disjunkten Subgraphen (Bäume) formuliert, in der jeder Baum alle Anschlüsse eines Netzes verbindet.

4.2.4 Überlegungen und allgemeine Lösungen

Die FPGA-Architekturen haben zwei eindeutige Merkmale:

1. Die FPGAs sind durch die Verdrahtungsressourcen mehr als durch logische Ressourcen beschränkt [26].

2. Die Leistung eines FPGA-Layouts wird stärker durch die Pfadverzögerung als durch die logische (CLB) Verzögerung beeinflußt [4].

Mit Berücksichtigung dieser Merkmale wäre ein erster Schritt für die Verdrahtung von FPGAs ein verdrahtbares Design unabhängig von den Ressourcenbeschränkungen zu finden. Allerdings kann solch ein Design mit guten Pfadlaufzeiten nicht verdrahtbar sein.

Um die gesamte Leistung bei der Verdrahtung mittels konventioneller Methoden zu verbessern, hat man großen Wert auf die Netze entlang der kritischen Pfaden gelegt, damit die Verzögerung an diesen Pfaden verbessert werden kann. Aber die Verbesserung der kritischen Pfade kann dazu führen, daß andere unkritisch Pfade über längere Wege verdrahtet werden müssen, was zur Verletzung der Zeitbeschränkung und sogar zur Entstehung eines unverdrahtbaren Designs führen kann. Um ein zeitverletzungsfreies FPGA-Design zu garantieren, müssen die Informationen über Pfadverzögerungen und Verdrahtbarkeit durch den gesamten Verdrahtungsprozeß in Betracht gezogen werden.

Konventionelle physische Designslösungen teilen das Verdrahtungsproblem in zwei Unterprobleme: Globale Verdrahtung und detaillierte Verdrahtung. Der Hauptgrund für diese Teilung ist die Problemkomplexität. Dabei besteht jedoch die Gefahr einer Suboptimierung durch eine Optimierung der Unterprobleme. Nach Möglichkeit sollte deshalb solch eine Teilung vermieden werden.

Ebenso kann die Zerlegung eines multiple-terminalen Netzes in zwei-terminal Subnetze zu einer schlechten Verdrahtungsqualität führen. Das wurde in Recherchen über Steinerbaumkonstruktionen [10] gezeigt. Deshalb ist die Zerlegung von multiple-terminal Netzen ebenfalls zu vermeiden.

Mit der Berücksichtigung der genannten Überlegungen wird hier für die Verdrahtung von FPGAs eine Verdrahtungslösung vorschlagen, in der die globale- und die detaillierte

Verdrahtung in einem Schritt durchgeführt wird. Die Lösung soll in zwei Stufen durchgeführt werden: *Anfangsverdrahtung* und *Auflösen-und-Widerverdrahten*.
In beiden Stufen sollen die Netze durch die Verwendung von einem auf *maze-routing* basierenden Algorithmus verbunden werden.

Während der ersten Stufe werden die Verdrahtungsdichten und die Pufferzeiten der Pfade auf der Basis eines minimalen Steinerbaummodells berechnet. Die Netze werden dann in Abhängigkeit davon, wie kritisch sie sind und von ihrer Verdrahtbarkeit, nacheinander verdrahtet. Während der zweiten Stufe werden die Verletzungen der Verdrahtungsressourcen und die Zeitverletzungen iterativ aufgehoben. Unter Beachtung der Ressourcen- und Zeitbeschränkungen werden in der Iteration manche Netze aufgelöst und wiederverdrahtet.
Die Auswahl welche Netze aufgelöst und wiederverdrahtet werden sollen, ist von zwei Faktoren abhängig: Wie kritisch sind die Netze und die Verdrahtungsdichte der Regionen die mit den Netzen zusammenhängen. Die (auflösen-und-wiederverdrahten) Prozedur soll von auf simulated-evolution basierten Optimierungstechniken unterstützt werden.

4.3 FPGA-Ablaufverfolger

4.3.1 Pfadaufzählung

Im ersten Schritt generiert der FPGA-Ablaufverfolger sämtliche Signalpfade einer gegebene CLB Netzliste. Die CLB Netzliste wird als gerichteter Graph modelliert. Es wird eine breadth-first [10] Suchmethode benutzt, um alle Pfade eines Design aufzuzählen.

4.3.2 Pufferzeitberechnung

Im zweiten Schritt berechnet der FPGA-Ablaufverfolger die Pufferzeiten von allen Pfaden. Es wird empfohlen, das Elmor-delay Modell (Approximation von Signalverzögerung in einem RC-Baum Netzwerken), welches in [7] bzw. [18] vorgestellt wird, zu nutzen.

Die Verdrahtung eines m-terminalen Netzes mit minimaler Distanz ist äquivalent zu der Feststellung des minimal-langen Baumes in einem Graph, der alle verbundene Knoten eines Netzes umfaßt (d.h. Steinerbaumproblem SBP). Da das SBP NP-Vollständig ist [10], soll ein multiple-component growth wave Expansionsalgorithmus [12] benutzt werden, um die Approximation des Steinerbaums zu finden. Nachdem man den Steinerbaum eines Netzes gefunden hat, kann man die Anzahl von Leitungssegmenten, Verbindungsmatrizen und Switchmatrizen, die vom Netz genutzt werden, bestimmen. Danach werden die RC-Baum-Netzwerke der Netze konstruiert und die Leiterbahnverzögerungen der Netze berechnet. Während der Kalkulation von Leiterbahnverzögerungen wird für jedes Netz, unabhängig von den Beschränkungen der Verdrahtungsressourcen den minimal-langen Baum suchen.

Die geschätzte Leiterbahnverzögerung eines Netzes kann als Untergrenze für die Leiterbahnverzögerung benutzt werden. Nachdem die Verzögerungen aller Netze ermittelt wurden, können die Pfadpufferzeiten berechnet werden.

4.3.3 Schätzung der Verdrahtungsdichte

In diesem Schritt schätzt der FPGA-Ablaufverfolger den Schwierigkeitsgrad der einzelnen Netze ab. Der Schwierigkeitsgrad eines Netzes wird als die Verdrahtungsdichte der Region, die alle verbundene Knoten dieses Netzes umfaßt, definiert. Unter der Region, die alle verbundene Knoten eines Netzes umfaßt, versteht man den minimalen Rechteckbereich, der alle Knoten umfaßt.
Beispiel: Abbildung 4.1 (Seite 55) zeigt den minimalen Rechteckbereich, den ein Netz umfaßt, das drei Knoten *A, B* und *C* verbindet.

Nach der Anwendung des „*multiple-component growth wave*“ Algorithmus zur Bestimmung einer Steinerbaumapproximation für jedes Netz kann die Anzahl der belegten Leitungssegmente in jeder Verbindungs- und Switchmatrix ermittelt werden. Die Verdrahtungsdichte eines Netzes kann als Verhältnis zwischen den verfügbaren und

den belegten Leitungssegmenten in der Region, die alle verbundenen Knoten eines Netzes umfaßt, berechnet werden.
Ein großer Wert für die Verdrahtungsdichte eines Netzes bedeutet, daß dieses Netz schwierig zu verdrahten ist, weil es durch einen dichten Verdrahtungsbereich gehen muß.

4.3.4 Anfangsverdrahtung

Der Anfangsverdrahter verbindet die Netze einzeln in Abhängigkeit davon, wie kritisch sie sind. Wie kritisch $Krit(n_i)$ ein Netz n_i ist, wird wie folgt berechnet:

$$Krit(n_i) = \alpha_1 * \left(Max\_Pufferzeit - Pufferzeit(n_i)\right) + (1 - \alpha_1) * Dichte(n_i)$$

Dabei ist α_1 ein Parameter, der vom Benutzer gesetzt werden kann, um damit die Bevorzugung eines Terms gegenüber dem anderen auszudrücken. $Max\_Pufferzeit$ bezeichnet den maximalen Pufferzeitwert von allen Netzen. $Pufferzeit(n_i)$ ist die Pufferzeit vom Netz n_i. $Dichte(n_i)$ ist die Verdrahtungsdichte der Region, die alle verbundenen Knoten vom Netz n_i umfaßt.
Wenn ein Netz einen großen $Krit(n_i)$ erzeugt, dann bedeutet es, daß dieses Netz kritisch und schwierig zu verdrahten sein wird.

Zuerst sortiert der Anfangsverdrahter die Netze absteigend bezüglich $Krit(n_i)$. Dann benutzt er den „multiple-component growth wave“ Expansionsalgorithmus, um eine Steinerbaumapproximation zu finden.

Der Anfangsverdrahter verbindet die Netze einzeln. Dabei wird mit dem kritischsten Netz begonnen. Während der Verbindung eines Netzes berücksichtigt der Anfangsverdrahter die schon existierenden Netze und versucht, ohne die Verletzung der Verdrahtungsressourcenlimitierung einen unblockierten Pfad für das Netz zu finden. Wenn kein unblockierter Pfad existiert, muß der Verdrahter einen Pfad mit minimalen

Verletzungen finden. Deshalb können manche Leiterbahnsegmente und/oder Schalter von mehreren Netzen in Anspruch genommen werden. Diese Verletzung der Verdrahtungsressourcen wird dann durch den Auflöser-und-Wiederverdrahter aufgehoben. Es folgt eine Algorithmusbeschreibung in Pseudocode:

Algorithmus: *Multiple Component Growth*

$V = \{v_1, v_2, \ldots, v_n\}$;

if $|V| \leq 1$ **then** Return;
Anfangskomponente $G_i \leftarrow (\{v_i\}, \varnothing)$, $1 \leq i \leq n$;
$r \leftarrow n$;

while $r > 1$ **do**

Expansion auf der existierenden $\{G_i\}$ bis zwei Komponenten sich treffen;
Die zwei Komponenten sollen G_m und G_n sein, dann
$G_m \leftarrow G_m \cup G_n \cup S(G_m, G_n)$,
$G_n \leftarrow \varnothing$,
$r \leftarrow r - 1$;

endwhile.

Dieser Algorithmus ist eine Erweiterung des klassischen Lee Algorithmus [13], der für die Leitwegsuche genutzt wird.

Eine Komponente repräsentiert einen verbundenen Teilgraph eines Verdrahtungsgraphen. Anfangs besteht eine Komponente nur aus einem Pinknoten, später wird sie durch die Zunahme von anderen benachbarten Knoten erweitert. Ein Knoten ist erst dann mit einer Komponente benachbart, wenn eine Kante zwischen ihm und der Komponente existiert. Die Wahl, welcher Knoten während jeder Erweiterung hinzugenommen werden soll, kann in einer breadth-first Weise bestimmt werden.

Während des Erweiterungsprozesses prüft der Algorithmus die Legitimität einer zwischen zwei Komponenten bestehenden Verbindung. Das kann einfach getan werden, indem die Zwischenkomponente entlang der übergreifenden Verbindung geprüft wird. Wenn dort die Zwischenkomponente ein I/O-Pin ist, handelt es sich um eine illegale Erweiterung, andernfalls ist sie legal.

Beispiel: In Abbildung 4.2(b) Seite 56 ist die Erweiterung zwischen *VP1* und *VW5*, die durch *e1, VW1, e2, VP2, e3, VW2* und *e4* geht, zu sehen. Da *VP2* eine I/O-Pin Komponente ist, ist die Erweiterung von *VW1* zu *VW2* durch *e2,VP2* und *e3* illegal.

$S(G_m,G_n)$ bezeichnet die Prozedur für die Verbindungserweiterungen zwischen zwei Komponenten. Nach Abschluß der Erweiterung wird ein Wert mit den neu gewonnenen Knoten ermittelt, um die Distanz zwischen den Knoten und der original unerweiterten Komponente anzuzeigen.

Wenn zwei Komponenten zu einer zusammen gefaßt werden, wird durch Rückverfolgung eine übergreifende Verbindung zwischen diesen beiden originalen Komponenten gefunden. Beide Komponenten werden mit ihren Verbindungen gemischt und für spätere Erweiterungen als eine Komponente behandelt.

Dieser Prozeß wiederholt sich so lange bis nur noch eine Komponente bleibt, d.h. alle Pins wurden miteinander verbunden.

4.3.5 Auflösung und Wiederverdrahtung

Die Auflösung-und-Wiederverdrahtung besteht aus zwei Phasen: Aufhebung der Verdrahtungsressourcenverletzungen und Aufhebung der Zeitverletzungen.

4.3.5.1 Aufhebung der Verdrahtungsressourcenverletzungen

Wenn das Resultat nach der Beendigung der Anfangsverdrahtung verletzungsfrei (bezüglich Verdrahtungsressourcen) ist, wird das Design an die zweite Phase (Aufhebung der Zeitverletzungen) geschickt. Ansonsten hebt der Auflöser-und-Wiederverdrahter die Verdrahtungsressourcenverletzungen durch der Verwendung von auf simulated-evolution basierte Optimierungstechniken auf.

Das Wesentliche hierbei ist die Auswahl des geeigneten Netzes, um es aufzulösen und dann neu zu verdrahten. Es ist klar, daß ein Netz, das viele Verletzungen verursacht (im Vergleich mit anderen Netzen) ein guter Kandidat für die Auflösung ist. Aber eine so einfache Vorgehensweise kann zu schwachen Ergebnisse führen.

Eine simulierte Entwicklung (simulated-evolution) präsentiert ein Zufallverteilungs-verfahren, das die Auflösung von guten und schlechten Netzen erlaubt. Ein schlechtes Netz hat größere Chancen aufgelöst zu werden, während ein gutes Netz weniger aber keine Nullchancen hat, um aufgelöst zu werden.

Eine auf simulierte Entwicklung basierte (auflösen-und-wiederverdrahten) Prozedur kann wie folgt aussehen:

Algorithmus: *Auflösen-und-Wiederverdrahten*

while (nicht Zeitabgelaufen & nicht Durchführbar) **do**
alle Netze Auswerten;
Netzwerte normieren;
for jedes Netz n_i
if (normierte_Wert (n_i) ≥ random (0,1))
Netz n_i Auflösen;
for jedes aufgelöste Netz n_i
Netz n_i mit multiple-component growth Algorithmus wieder verdrahten;
endwhile
if (Zeitabgelaufen) **return** (Fehler);
return (Erfolg);

end.

Ein Netz wird nach seiner Verbindungslänge und nach der Anzahl der Verletzungen, die es verursacht, folgendermaßen ausgewertet:

$$wert(n_i) = \alpha_2 * \left(\frac{aktuelle\_länge_i}{geschätzte\_\min\_länge_i} \right)$$

$$+ \left(\beta_1 * anzahl\_verletzungen_i \right)$$

α_2 und β_1 sind Parameter, die durch den Benutzer gesetzt werden. Je schwieriger ein Netz zu verdrahten ist, um so größer ist $wert(n_i)$. Die Normierung wird so durchgeführt, daß alle normierten Werte zwischen 0.05 und 0.95 liegen, wobei der Wert vom besten Netz gleich 0.05 und vom schlechtesten Netz gleich 0.95 beträgt.

Die Berechnung des normiertes Wertes wird wie folgt durchgeführt:

$$norm\_Wert(n_i) = 0.05 + 0.9 * \left(\frac{Wert(n_i) - niedrigste\_Wert}{höchste\_Wert - niedrigste\_Wert} \right)$$

Um herauszufinden ob ein Netz aufgelöst werden soll, wird eine Zufallszahl zwischen Null und Eins generiert, und mit dem normierten Wert verglichen. Wenn die Zufallszahl kleiner ist, wird das Netz aufgelöst. Die Menge der aufgelösten Netze wird dann einzeln und beginnend mit dem schlechtesten Netz wieder verdrahtet. Die Verdrahtung soll wieder mit dem multiple-component growth Algorithmus durchgeführt werden, mit der Ausnahme, daß die bereits verdrahteten Netze nicht ignoriert werden. Dabei werden die Distanzen und Konflikte, die durch die Verwendung von verbundenen Ressourcen, welche während der Berechnung von Erweiterungskosten entstehen, dokumentiert. Eine Erweiterung auf einen Knoten, der schon von einem anderen Netz besetzt wurde, ist mit hohen Kosten verbunden.

Man kann z.B. bei der Algorithmusimplementierung die Kosten von einem besetzten Knoten zehnmal höher als von einem freien Knoten setzen.
Es wird wieder die Möglichkeit einer Erweiterung auf einen besetzten Knoten geben, was aber die Verletzungen nicht aufhebt. Später kann das Netz, welches vor der Erweiterung den Knoten besitzt, aufgelöst und während einer weiteren Iteration wiederverdrahtet werden.

Diese Betrachtungen zeigen eine gute Lösung für eine verdrahtungsressourcen-verletzungsfreie Verdrahtung.

4.3.5.2 Aufhebung der Zeitverletzungen

In der erste Phase wurde ein Design frei von Verletzungen der Verdrahtungsressourcen generiert. Nun werden in der zweite Phase der FPGA-Ablaufverfolger die Zeitverletzungen aller Pfade aufgehoben. Nach Abschluß der ersten Phase ist für jedes Netz die aktuelle Anzahl von Leitungssegmenten und von Schaltern bekannt, durch die das Netz gehen muß. Deshalb kann man für jedes Netz die aktuelle Leiterbahnverzögerung und folglich auch die Pfadverzögerungen und die Pfadpufferzeit berechnen. Das Netz mit einem negativen Pufferzeitwert ist ein zeitverletzendes Netz, das durch den (Auflösung-und-Wiederverdrahtungs) Prozeß behandelt werden muß.

In dieser Phase werden bei der Wiederverdrahtung die schon verdrahteten Netze beachtet und es wird keine Verdrahtung, die die Verdrahtungsressourcen-beschränkungen verletzt, erlaubt.

Wesentlich ist wiederum, wie man das Netz, das aufgelöst werden soll, auswählt. Es gibt zwei Netztypen, die aufgelöst-und-wiederverdrahtet werden müssen, um die Zeitverletzungen aufzuheben. Der erste Typ sind Netze, die entlang eines Pfades die Zeitbeschränkungen verletzen (d.h. die Netze mit negativem Pufferzeitwert). Diese Netze sind gute Kandidaten um aufgelöst zu werden, weil die Wiederverdrahtung dieser Netze (mit kürzerer Verdrahtungsdistanz) zu niedrigerer Netz- und Pfadverzögerung führen kann.

Aber auch hier kann eine einfache Vorgehensweise zu schwachen Ergebnissen oder sogar wegen der Ressourcenbeschränkungen zu sehr schlechten führen. Der andere Typ betrifft Netze, die mehr Verdrahtungsverzögerung mit der Zeitbeschränkungen vertragen können (d.h. Netz mit einem großen positiven Pufferzeitwert). Bei der Wiederverdrahtung dieser Netze durch Umleitungen, kann es in manchen überfüllten Bereichen zu einer erneuten Verbindung von kritischen Netzen führen.

Nach den zuvor erwähnten Überlegungen wird ein Netz nach zwei Faktoren ausgewertet: Die Verzögerungssensitivität des Netzes und der Sensitivität der Verdrahtungsdichte des Netzes. Diese Auswertung sieht folgendermaßen aus:

$$Wert(n_i) = \alpha_3 * VS(n_i) + (1 - \alpha_3) * VDS(n_i)$$

α_3 ist ein Parameter, der vom Benutzer gesetzt werden kann, um damit die Bevorzugung eines Terms gegenüber dem anderen auszudrücken. $VS(n_i)$ drückt die Verzögerungssensitivität einen Netz n_i aus, während die $VDS(n_i)$ die Sensitivität der Verdrahtungsdichte einen Netz n_i bezeichnet.

Bei der Berechnung der Verzögerungssensitivität eines Netzes n_i müssen zwei Fälle unterschieden werden:

1) Für ein Netz mit negativem Pufferzeitwert:

$$VS(n_i) = \frac{t(n_i) - t'(n_i)}{|Pufferzeit(n_i)|}$$

2) Für ein Netz mit positivem Pufferzeitwert (inklusive der Null):

$$VS(n_i) = \frac{Pufferzeit(n_i)}{t(n_i) - t'(n_i)}$$

Dabei ist $t(n_i)$ die aktuelle Verzögerung von Netz n_i und $t'(n_i)$ die geschätzte minimale Verzögerung von Netz n_i.

Bei einem Netz n_i mit negativem Pufferzeitwert und einem großen $VS(n_i)$-Wert ist es für die Verminderung der Verzögerung effektiver, dieses Netz mit kürzerer Distanz wieder zu verdrahten. Andererseits ist es bei einem Netz n_i mit einem positiven Pufferzeitwert und einem großen $VS(n_i)$-Wert effektiver, da dieses Netz mehr Leiterbahnverzögerung vertragen kann, über mehr Umleitungen wieder zu verdrahten. Dadurch werden Verdrahtungsressourcen für kritische Netze freigelassen.

Bei der Berechnung der Sensitivität der Verdrahtungsdichte von Netzen müssen ebenfalls zwei Fälle unterschieden werden:

1) Für ein Netz mit negativem Pufferzeitwert:

$$VDS(n_i) = \frac{Gesamte\_Verdrahtung(n_i)}{Belegte\_Verdrahtung(n_i)}$$

2) Für ein Netz mit positivem Pufferzeitwert (inklusive der Null):

$$VDS(n_i) = \frac{Belegte\_Verdrahtung(n_i)}{Gesamte\_Verdrahtung(n_i)}$$

$Gesamte\_Verdrahtung(n_i)$ repräsentiert die gesamte Anzahl von Verdrahtungsressourcen in der Region, die alle verbundene Knoten von Netz n_i umfaßt. $Belegte\_Verdrahtung(n_i)$ ist die gesamte Anzahl von belegten Verdrahtungsressourcen in der Region, die alle verbundene Knoten von Netz n_i umfaßt.

Für ein Netz n_i mit negativem Pufferzeitwert und einem großen $VDS(n_i)$-Wert sind mehr Chancen gegeben, dieses Netz mit kürzerer Distanz wieder zu verdrahten. Andererseits hat ein Netz n_i mit einem positiven Pufferzeitwert und einem großen $VDS(n_i)$-Wert mehr Chancen, über mehr Umleitungen wieder verdrahtet zu werden (d.h. kein durchgehen in der dichten Region). Dadurch werden Verdrahtungsressourcen in dieser dichten Region für kritische Netze verfügbar gemacht.

5. Zusammenfassung

Die vorliegende Diplomarbeit untersucht die Verdrahtungsressourcen von Xilinx-FPGAs, indem das LCA-Format der 4000er Familie beschrieben und ein Lösungsansatz für das Verdrahtungsproblem präsentiert wird.

Bei der Untersuchung des LCA-Formats war die Arbeit mit dem XACT-Design-Editor xde, der ebenfalls Dateien im LCA-Format als Eingabe- und Ausgabedateien nutzt, sehr wichtig. Die Beschreibung ist zwar auf den Baustein XC4002A bezogen, aber sie gilt für alle 4000er Bausteine, weil die Bausteine dieser Familie dieselben Elemente besitzt aber mit verschiedener Anzahl.

Um sicher zu sein, daß die Beschreibung des LCA-Formats exakt ist, wurde das LCA-File für einen Volladdierer manuell hergestellt und mit dem LCA-File, der durch das Programm PPR von Xilinx erzeugt wurde, verglichen. Als Ergebnis wurden nur Unterschiede bei der Benennung, Plazierung und Verdrahtung, aber keine grundsätzliche Abweichungen festgestellt.

Im letzten Teil dieser Arbeit wurde ein durch Leistung und Verdrahtbarkeit gesteuerter Verdrahtungsalgorithmus (*FPGA-Ablaufverfolger*) für FPGAs präsentiert. Die Verdrahtung wird in zwei Schritten durchgeführt: *Anfangsverdrahtung* und *Auflösen_und_Wiederverdrahten.*

Die wichtigsten Vorteile dieses Algorithmus sind:

- Die Information über Pfadverzögerungen und Verdrahtbarkeit werden durch den gesamten Verdrahtungsprozess in Betracht gezogen.
- Die globale und detaillierte Verdrahtung werden in einem Schritt durchgeführt.

Ziel weiterführender Arbeiten könnte die Implementierung des Algorithmus und der praktische Vergleich mit anderen Algorithmen sein, um eine weitere Bewertung dieses Algorithmus zu ermöglichen.

6. Literaturverzeichnis

[1] M. J. Alexander und G. Robins, „ New performance-driven FPGA routing algorithm“ in *Proc. 32th Design Automation Conf*., 1995, Seite(562-567).

[2] A. Auer, „Programmierbare Logik-IC“ Hüthig 1994.

[3] A. Auer, „Feldprogrammierbare Gate Arrays: FPGA“ Hüthig 1995.

[4] N. B. Bhat und D. D. Hill, „ Routable technology mapping for LUT FPGAs “ in *Proc. Int. Conf. Computer Design*, 1992, Seite (95-98).

[5] S. Brown, J. Rose und Z. G. Vranesic, „A detailed router for field programmable gate arrays“ *IEEE Trans. Computer-Aided Design*, vol. 11, no. 5, Seite (620-628), May 1992.

[6] C.-D. Chen, Y.-S. Lee, A. C.-H. Wu und Y.-L. Lin, „ TRACER-fpga: A router for RAM-based FPGAs“ *IEEE Trans. Computer-Aided Design*, vol. 14, no. 3, Seite (371-374), Mar. 1995.

[7] W. C. Elmore, „The transient response of damped linear networks with particular regard to wideband amplifiers“ *J. Appl. Phys.* vol. 19, no. 1, 1948, Seite (55-63).

[8] J. Frankle, „Iterative und adaptive slack allocation for performance-driven layout and FPGA routing“ in *Proc. 29th Design Automation Conf.*, 1992, Seite(536-542).

[9] M. Hermann „Technologieabbildung und Testvorbereitung für Programmierbare Logikbausteine mit komplexen Grundzellen“ Aachen 1997.

[10] J.-M. Ho, G. Vijayan und C. K. Wong, „New algorithms for the rectilinear Steiner tree problem“ *IEEE Trans. Computer-Aided Design*, vol. 9, no. 2, Feb 1990.

[11] R. M. Kling und P. Banerjee, „ESP: Placement by simulated evolution“ *IEEE Trans. Computer-Aided Design*, vol. 8, no. 8, Seite (245-256), Mar. 1989.

[12] E. S. Kuh und M. Marek-Sadowska, „Global routing“ *in Layout Design and Verification*, T. Ohtsuki, Ed. New York: North-Holland, 1985.

[13] C. Y. Lee, „An algorithm for path connections and its applications“ *IRE Trans. Electronic Comput.*, vol. EC-10, Seite (346-365), Sept. 1961.

[14] G. G. Lemieux und S. D. Brown, „ A detailed routing algorithm for allocating wire segments in field-programmable gate arrays“ “ in *Proc. ACM/SIGDA Physical Design Workshop*, 1993, Seite (215-226).

[15] F. D. Lewis und W. C.-C. Pong, „A negative reinforcement method for FPGA routing“ in *Proc. 30th Design Automation Conf.*, 1993, Seite (601-605).

[16] Y.-L. Lin, Y.-C. Hsu und F.-S. Tsai, „SILK: A simulated evolution router“ *IEEE Trans. Computer-Aided Design*, vol. 8, no. 10, Seite (1108-1114), Oct. 1989.

[17] U. Möhrke, M. Schmidt und P. Herrmann, „Das Xilinx-LCA-Format“ *Universität Leipzig/Institut für Informatik*, Report 2, 1996.

[18] P. Penfield, J. Rubinstein und M. A. Horowitz, „Signal delay in RC tree networks“ *IEEE Trans. Computer-Aided Design*, vol. CAD-2, no. 3, Seite (202-211), 1983.

[19] J. Rose, „Parallel global routing for standard cells“ *IEEE Trans. Computer-Aided Design*, vol. 9, no. 9, Seite (1085-1095), Sept. 1990.

[20] J. Rose und S. Brown, „ Flexibility of interconnection structures for field-programmable gate arrays“ *IEEE J. Solid-State Circuits*, vol. 26, no. 3, Seite (277-282), Mar. 1991.

[21] T. Saab und V. Rao, „An evolution-based approach to partitioning ASIC systems“ in *Proc. 26th Design Automation Conf.*, 1989, Seite (767-770).

[22] Schiffmann - Schmitz „Technische Informatik 1“ 2.Aufl. Berlin 1992.

[23] M. Schmidt, U. Möhrke und P. Herrmann, „Synthesis Tool for FPGA-Design“ *Universität Leipzig/Institut für Informatik*, Report 15, 1995.

[24] Y. Sun und C. L. Liu, „Routing in a new 2-dimensional FPGA/FPIC routing architecture“ in *Proc. 31th Design Automation Conf.*, 1994, Seite (171-176).

[25] Y. Sun, T.-C. Wang, C. K. Wang und C. L. Liu, „Routing for symmetric FPGAs and FPICs“ in *Proc. Int. Conf. Computer-Aided Design*, 1993, Seite (486-490).

[26] S. Trimberger und M.-R. Chene, „Placement-based partitioning for lookup-table-based FPGAs“ in *Proc. Int. Conf. Computer Design*, 1992, Seite (91-94).

[27] U. Tietze und CH. Schenk, „Halbleiter-Schltungstechnik“ *Springer-Verlage*, Berlin Heidelberg New York, 5 Auflage, 1980.

[28] Y.-L. Wu and M. Marek-Sadowska, „An efficient router for 2-D field programmable gate arrays“ in *Proc. EDAC*, 1994, Seite (412-416).

[29] Y.-L. Wu and M. Marek-Sadowska, „Graph based analysis of FPGA routing“ in *Proc. Euro-DAC*, 1994, Seite (104-109).

[30] Xilinx Inc., „The Programmable Logic Data Book, San Jose, CA, 1994.

[31] Xilinx Inc., „XACT Reference Guide, April 1994.

[32] H. Youssef and E. Sharagowitz, „Timing constraints for correct performance“ in *Proc. Int. Conf. Computer-Aided Design*, 1990, Seite (24-27).

7. Abbildungsverzeichnis

8. Anlagenverzeichnis

9. Anlagen

Anlage A: count.lca

```
;: count.lca (4002APC84-5), ppr Xilinx:ppr:5.2.0:95/10/23, 1998/05/24
23:58:08
```

```
Version 2
Design 4002APC84
Speed -5
Addnet CTRL2_2 PAD42.I2 JC.C2 IB.F3
NProgram CTRL2_2 JC.C2:row.K.local.1 IB.F3:col.C.local.6 KC.24.1.6
KC.24.1.5 KE.24.1.23 KE.24.1.6 PAD42.I2:row.K.local.1
Addnet CTRL2_1 PAD42.I1 JE.C1 JD.F3
NProgram CTRL2_1 JE.C1:col.E.local.1 JD.F3:col.E.local.1
PAD42.I1:col.E.local.0
Beginpath CTRL2
Pathnet CTRL2_1 CTRL2_2
Endpath
Addnet CTRL3 PAD8.I1 HE.F3 HE.G3 GE.F3 GE.G3
NProgram CTRL3 HE.F3:col.G.local.1 HE.G3:col.G.local.1 HG.24.1.0
HG.24.1.17 GE.F3:col.G.local.0 GE.G3:col.G.local.0 EG.24.1.0
EG.24.1.17
NProgram CTRL3 CG.24.1.0 CG.24.1.17 PAD8.I1:col.G.local.1
Addnet CTRL4 PAD7.I2 GE.C1 HE.C1
NProgram CTRL4 GE.C1:col.E.local.7 HE.C1:col.E.local.6 HE.24.1.5
HE.24.1.12 EE.24.1.5 EE.24.1.12 CE.24.1.5 CE.24.1.12
PAD7.I2:col.E.local.6
Addnet CTRL5 PAD6.I2 ID.C3 HD.C3
NProgram CTRL5 ID.C3:col.E.long.1 HD.C3:col.E.long.1
row.F.local.1:col.E.long.1 PAD6.I2:col.E.long.1
Addnet B<0> PAD31.O JE.G2 JC.F2 JB.F2 JD.F2 JD.XQ
NProgram B<0> PAD31.O:col.K.local.0 KK.24.1.18 KK.24.1.0 KI.24.1.18
KI.24.1.11 KG.24.1.18 KG.24.1.11 JE.G2:row.K.local.7
JC.F2:row.K.local.7
NProgram B<0> JB.F2:row.K.local.6 KD.24.1.18 KD.24.1.11
JD.F2:row.K.local.6 JD.XQ:row.K.local.6
Addnet B<1> PAD30.O JE.G1 JC.F3 JB.F4 JC.YQ
NProgram B<1> PAD30.O:row.I.long.3 col.G.local.2:row.I.long.3-L
JG.24.1.1 JG.24.1.19 JE.24.1.10 JE.24.1.19 JE.G1:col.E.local.2
JE.24.1.16 JE.24.1.19
NProgram B<1> JD.24.1.10 JD.24.1.19 JC.YQ:row.J.local.5
JC.F3:col.D.local.5 JC.YQ:col.D.local.5 JB.F4:row.J.local.1
JC.YQ:row.J.local.0
Addnet B<2> PAD45.O JB.F3 JE.G3 JE.XQ
NProgram B<2> PAD45.O:row.K.local.6 JB.F3:col.C.local.0 KC.24.1.11
KC.24.1.0 KE.24.1.18 KE.24.1.11 JE.XQ:row.K.local.6
JE.G3:col.G.local.3
NProgram B<2> KG.24.1.2 KG.24.1.20 JE.XQ:row.K.local.4
Addnet B<3> PAD47.O JB.F1 IB.XQ
```

```
NProgram B<3> PAD47.O:row.K.local.5 KB.24.1.1 KB.24.1.10
JB.F1:col.B.local.2 JB.24.1.1 JB.24.1.16 IB.XQ:col.B.local.2
Addnet C<0> PAD28.O GE.F1 HE.F1 HE.XQ
NProgram C<0> PAD28.O:row.I.local.0 IJ.24.1.23 IJ.24.1.6 IH.24.1.23
IH.24.1.6 IE.24.1.6 IE.24.1.5 GE.F1:col.E.local.6 HE.F1:col.E.local.7
NProgram C<0> HE.XQ:col.E.local.7
Addnet C<1> PAD27.O HE.G4 GE.G2 HE.YQ
NProgram C<1> PAD27.O:row.H.long.0 col.G.local.5:row.H.long.0-L
HE.YQ:col.G.local.5 HE.G4:row.H.local.3 GE.G2:row.H.local.3
HE.YQ:row.H.local.3
Addnet C<2> PAD26.O HE.F4 GE.F2 GE.XQ
NProgram C<2> PAD26.O:col.K.local.5 col.K.local.5:row.G.long.0-S
col.G.local.5:row.G.long.0-L HG.24.1.4 HG.24.1.22 HE.F4:row.H.local.2
GE.F2:row.H.local.2
NProgram C<2> GE.XQ:row.H.local.2
Addnet C<3> PAD25.O HE.G2 GE.G4 GE.YQ
NProgram C<3> PAD25.O:col.K.local.1 GK.24.1.18 GK.24.1.17 GI.24.1.18
GI.24.1.11 GG.24.1.11 GG.24.1.17 HE.G2:row.I.local.7 IG.24.1.18
IG.24.1.0
NProgram C<3> GE.YQ:col.G.local.1 GE.G4:row.G.local.5
GE.YQ:row.G.local.5
Addnet TC_DEC PAD56.O HC.F1 GB.F2 GC.F2 HB.Y
NProgram TC_DEC PAD56.O:col.B.local.5 HB.24.1.4 HB.24.1.7
HB.Y:row.H.local.2 HC.F1:col.C.local.2 HB.Y:col.C.local.2
GB.F2:row.H.local.1
NProgram TC_DEC GC.F2:row.H.local.0 HB.Y:row.H.local.1
Addnet TC_LFSR PAD52.O IC.X
NProgram TC_LFSR PAD52.O:col.B.local.0 JB.24.1.11 JB.24.1.0
IC.X:row.J.local.7
Addnet E<0> PAD55.O HB.F4 HB.G4 HC.F4 GC.F1 GB.F3 GC.XQ
NProgram E<0> PAD55.O:col.B.local.3 HB.24.1.2 HB.24.1.9
HB.F4:row.H.local.4 HB.G4:row.H.local.4 HC.24.1.9 HC.24.1.20
HC.F4:row.H.local.4
NProgram E<0> GC.XQ:row.H.local.4 GC.F1:col.C.local.4
GB.F3:col.C.local.4 GC.XQ:col.C.local.4
Addnet E<1> PAD3.O GB.F1 HB.F1 HB.G1 HC.F3 GB.XQ
NProgram E<1> PAD3.O:col.C.long.2 row.E.local.3:col.C.long.2-L
EC.24.1.8 EC.24.1.21 EB.24.1.8 EB.24.1.14 GB.24.1.3 GB.24.1.14
GB.F1:col.B.local.4
NProgram E<1> GB.XQ:col.B.local.4 HB.F1:col.B.local.2
HB.G1:col.B.local.2 HB.24.1.1 HB.24.1.16 GB.XQ:col.B.local.2
HC.F3:col.D.local.1
NProgram E<1> HD.24.1.18 HD.24.1.17 GB.XQ:row.H.local.6
Addnet E<2> PAD54.O HC.F2 HB.F3 HB.G3 HC.XQ
```

```
NProgram E<2> PAD54.O:col.B.local.3 IB.24.1.2 IB.24.1.9 IC.24.1.9
IC.24.1.20 HC.F2:row.I.local.4 HC.XQ:row.I.local.4 HB.F3:col.C.local.4
NProgram E<2> HB.G3:col.C.local.4 HC.XQ:col.C.local.4
Addnet E<3> PAD53.O HB.F2 HB.G2 HB.XQ
NProgram E<3> PAD53.O:col.B.local.4 HB.XQ:col.B.local.4
HB.F2:row.I.local.2 HB.G2:row.I.local.2 HB.XQ:row.I.local.2
Addnet CTRL1 PAD50.I2 JE.F4 JE.G4 JD.F4 JC.F4 JB.G4 JB.C4
NProgram CTRL1 JE.F4:row.J.local.1 JE.G4:row.J.local.1 JE.24.1.23
JE.24.1.6 JD.F4:row.J.local.0 JC.F4:row.J.local.1 JC.24.1.23 JC.24.1.6
NProgram CTRL1 JB.G4:row.J.local.0 JB.C4:row.J.local.0
PAD50.I2:row.J.local.0
Addnet BUFGS1/CLK HE.K JE.K GE.K HD.K ID.K JD.K GC.K HC.K JC.K GB.K
Addpin BUFGS1/CLK HB.K IB.K bufgs_bl.O
NProgram BUFGS1/CLK HE.K:col.E.long.4 JE.K:col.E.long.4
GE.K:col.E.long.4 bufgs_bl.O:col.E.long.4 HD.K:col.D.long.4
ID.K:col.D.long.4
NProgram BUFGS1/CLK JD.K:col.D.long.4 bufgs_bl.O:col.D.long.4
GC.K:col.C.long.4 HC.K:col.C.long.4 JC.K:col.C.long.4
bufgs_bl.O:col.C.long.4
NProgram BUFGS1/CLK GB.K:col.B.long.4 HB.K:col.B.long.4
IB.K:col.B.long.4 bufgs_bl.O:col.B.long.4
Addnet $1I166/PAD bufgs_bl.I i_bufgs_bl.I
NProgram $1I166/PAD bufgs_bl.I:i_bufgs_bl.I
Addnet LFSR_4COUNT<2> HD.XQ ID.C4 IC.F1
NProgram LFSR_4COUNT<2> ID.C4:row.I.local.2 IC.F1:col.C.local.5
IC.24.1.7 IC.24.1.13 ID.24.1.7 ID.24.1.22 HD.XQ:row.I.local.2
Addnet LFSR_4COUNT<3> HD.C4 HD.F4 IC.F2 HD.YQ
NProgram LFSR_4COUNT<3> HD.C4:row.H.local.3 HD.F4:row.H.local.3
HD.YQ:row.H.local.3 IC.F2:row.J.local.6 JE.24.1.18 JE.24.1.0
HD.YQ:col.E.local.1
Addnet LFSR_4COUNT<0> ID.XQ HD.F1
NProgram LFSR_4COUNT<0> HD.F1:col.D.local.2 ID.24.1.1 ID.24.1.16
ID.XQ:col.D.local.2
Addnet LFSR_4COUNT<1> ID.F4 IC.F4 ID.YQ
NProgram LFSR_4COUNT<1> ID.F4:row.I.local.0 IC.F4:row.I.local.1
ID.YQ:row.I.local.0
Addnet LFSR_FDBK/XNOR HD.X IC.F3
NProgram LFSR_FDBK/XNOR IC.F3:col.D.local.7 HD.X:col.D.local.6
Addnet A<0> PAD43.I2 JD.F1
NProgram A<0> JD.F1:col.D.local.2 KD.24.1.1 KD.24.1.10
PAD43.I2:row.K.local.5
Addnet A<1> PAD44.I2 JC.F1
NProgram A<1> JC.F1:col.C.local.5 KC.24.1.4 KC.24.1.7 KD.24.1.7
KD.24.1.22 PAD44.I2:row.K.local.2
```

```
Addnet A<2> PAD41.I2 JE.F2
NProgram A<2> JE.F2:row.K.local.3 PAD41.I2:row.K.local.3
Addnet A<3> PAD48.I2 JB.G2
NProgram A<3> JB.G2:row.K.local.2 PAD48.I2:row.K.local.2
Addnet SYNC_1/OR5_REG_ff_3 JB.Y IB.F2
NProgram SYNC_1/OR5_REG_ff_3 IB.F2:row.J.local.2 JB.Y:row.J.local.2
Nameblk PAD42 $1I128/PAD
Editblk PAD42
Base IO
Config INFF: I1:I I2:I OUT: PAD: TRI: O: OSPEED:
Endblk
Nameblk JC B<1>
Editblk JC
Base FG
Config F4:F4I G2: G3: CIN: COUT: X: Y: XQ: YQ:QY FFX:RESET FFY:K:RESET
DX: DY:H F:F2:F3:F4:F1 G: H:H1:F H1:C2 DIN: SR: EC: RAM: CDIR:
Equate F = (((F3@F2)*~F4)+(F4*F1))
Equate H = ((F*~H1)+H1)
Endblk
Nameblk IB B<3>
Editblk IB
Base FG
Config F4: G2: G3: CIN: COUT: X: Y: XQ:QX YQ: FFX:K:RESET FFY:RESET
DX:F DY: F:F3:F2 G: H: H1: DIN: SR: EC: RAM: CDIR:
Equate F = ((F2*~F3)+F3)
Endblk
Nameblk JE B<2>
Editblk JE
Base FG
Config F4:F4I G2:G2I G3:G3I CIN: COUT: X: Y: XQ:QX YQ: FFX:K:RESET
FFY:RESET DX:H DY: F:F4:F2 G:G2:G1:G3:G4 H:H1:G:F H1:C1 DIN: SR: EC:
RAM: CDIR:
Equate F = (F4*F2)
Equate G = (((G1*G2)@G3)*~G4)
Equate H = ((F+G)*~H1)
Endblk
Nameblk JD B<0>
Editblk JD
Base FG
Config F4:F4I G2: G3: CIN: COUT: X: Y: XQ:QX YQ: FFX:K:RESET FFY:RESET
DX:F DY: F:F3:F2:F4:F1 G: H: H1: DIN: SR: EC: RAM: CDIR:
Equate F = (((~F4*~F2)+(F4*F1))*~F3)
Endblk
Nameblk PAD8 $1I132/PAD
```

```
Editblk PAD8
Base IO
Config INFF: I1:I I2: OUT: PAD: TRI: O: OSPEED:
Endblk
Nameblk HE C<0>
Editblk HE
Base FG
Config F4:F4I G2:G2I G3:G3I CIN: COUT: X: Y: XQ:QX YQ:QY
FFX:SR:K:RESET FFY:SR:K:SET DX:G DY:F F:F3:F1:F4 G:G3:G4:G2 H: H1:
DIN: SR:C1 EC: RAM: CDIR:
Equate F = ((~F3*F1)+(F3*F4))
Equate G = ((~G3*~G2)+(G3*G4))
Endblk
Nameblk GE C<2>
Editblk GE
Base FG
Config F4: G2:G2I G3:G3I CIN: COUT: X: Y: XQ:QX YQ:QY FFX:SR:K:SET
FFY:SR:K:SET DX:G DY:F F:F3:F1:F2 G:G3:G2:G4 H: H1: DIN: SR:C1 EC:
RAM: CDIR:
Equate F = ((~F3*F2)+(F3*~F1))
Equate G = ((~G3*G2)+(G3*G4))
Endblk
Nameblk PAD7 $1I136/PAD
Editblk PAD7
Base IO
Config INFF: I1: I2:I OUT: PAD: TRI: O: OSPEED:
Endblk
Nameblk PAD6 $1I150/PAD
Editblk PAD6
Base IO
Config INFF: I1: I2:I OUT: PAD: TRI: O: OSPEED:
Endblk
Nameblk ID LFSR_4COUNT<0>
Editblk ID
Base FG
Config F4:F4I G2: G3: CIN: COUT: X: Y: XQ:QX YQ:QY FFX:EC:K:RESET
FFY:EC:K:RESET DX:F DY:DIN F:F4 G: H: H1: DIN:C4 SR: EC:C3 RAM: CDIR:
Equate F = F4
Endblk
Nameblk HD LFSR_FDBK/XNOR
Editblk HD
Base FG
```

```
Config F4:F4I G2: G3: CIN: COUT: X:F Y: XQ:QX YQ:QY FFX:EC:K:RESET
FFY:EC:K:RESET DX:DIN DY:F F:F4:F1 G: H: H1: DIN:C4 SR: EC:C3 RAM:
CDIR:
Equate F = ~(F4@F1)
Endblk
Nameblk PAD31 $1I154/PAD<0>
Editblk PAD31
Base IO
Config INFF: I1: I2: OUT:O PAD: TRI: O: OSPEED:SLOW
Endblk
Nameblk JB SYNC_1/OR5_REG_ff_3
Editblk JB
Base FG
Config F4:F4I G2:G2I G3: CIN: COUT: X: Y:H XQ: YQ: FFX:RESET FFY:RESET
DX: DY: F:F2:F4:F3:F1 G:G4:G2 H:H1:G:F H1:C4 DIN: SR: EC: RAM: CDIR:
Equate F = ((F3*F4*F2)@F1)
Equate G = (G4*G2)
Equate H = ((F*~H1)+G)
Endblk
Nameblk PAD30 $1I154/PAD<1>
Editblk PAD30
Base IO
Config INFF: I1: I2: OUT:O PAD: TRI: O: OSPEED:SLOW
Endblk
Nameblk PAD45 $1I154/PAD<2>
Editblk PAD45
Base IO
Config INFF: I1: I2: OUT:O PAD: TRI: O: OSPEED:SLOW
Endblk
Nameblk PAD47 $1I154/PAD<3>
Editblk PAD47
Base IO
Config INFF: I1: I2: OUT:O PAD: TRI: O: OSPEED:SLOW
Endblk
Nameblk PAD28 $1I155/PAD<0>
Editblk PAD28
Base IO
Config INFF: I1: I2: OUT:O PAD: TRI: O: OSPEED:SLOW
Endblk
Nameblk PAD27 $1I155/PAD<1>
Editblk PAD27
Base IO
Config INFF: I1: I2: OUT:O PAD: TRI: O: OSPEED:SLOW
Endblk
```

```
Nameblk PAD26 $1I155/PAD<2>
Editblk PAD26
Base IO
Config INFF: I1: I2: OUT:O PAD: TRI: O: OSPEED:SLOW
Endblk
Nameblk PAD25 $1I155/PAD<3>
Editblk PAD25
Base IO
Config INFF: I1: I2: OUT:O PAD: TRI: O: OSPEED:SLOW
Endblk
Nameblk PAD56 $1I163/PAD
Editblk PAD56
Base IO
Config INFF: I1: I2: OUT:O PAD: TRI: O: OSPEED:SLOW
Endblk
Nameblk HC E<2>
Editblk HC
Base FG
Config F4:F4I G2: G3: CIN: COUT: X: Y: XQ:QX YQ: FFX:K:RESET FFY:RESET
DX:F DY: F:F1:F4:F3:F2 G: H: H1: DIN: SR: EC: RAM: CDIR:
Equate F = (~F1*(F2@(F3*F4)))
Endblk
Nameblk GB E<1>
Editblk GB
Base FG
Config F4: G2: G3: CIN: COUT: X: Y: XQ:QX YQ: FFX:K:RESET FFY:RESET
DX:F DY: F:F2:F3:F1 G: H: H1: DIN: SR: EC: RAM: CDIR:
Equate F = (~F2*(F1@F3))
Endblk
Nameblk GC E<0>
Editblk GC
Base FG
Config F4: G2: G3: CIN: COUT: X: Y: XQ:QX YQ: FFX:K:RESET FFY:RESET
DX:F DY: F:F2:F1 G: H: H1: DIN: SR: EC: RAM: CDIR:
Equate F = (~F2*~F1)
Endblk
Nameblk HB TC_DEC
Editblk HB
Base FG
Config F4:F4I G2:G2I G3:G3I CIN: COUT: X: Y:G XQ:QX YQ: FFX:K:RESET
FFY:RESET DX:H DY: F:F4:F1:F3:F2 G:G4:G1:G3:G2 H:G:F H1: DIN: SR: EC:
RAM: CDIR:
Equate F = (F2@(F3*(F1*F4)))
Equate G = ~(~G2+G3+G1+~G4)
```

```
Equate H = (~G*F)
Endblk
Nameblk PAD52 $1I168/PAD
Editblk PAD52
Base IO
Config INFF: I1: I2: OUT:O PAD: TRI: O: OSPEED:SLOW
Endblk
Nameblk IC TC_LFSR
Editblk IC
Base FG
Config F4:F4I G2: G3: CIN: COUT: X:F Y: XQ: YQ: FFX:RESET FFY:RESET
DX: DY: F:F1:F2:F4:F3 G: H: H1: DIN: SR: EC: RAM: CDIR:
Equate F = ~(F3+F2+F1+F4)
Endblk
Nameblk PAD55 $1I69/PAD<0>
Editblk PAD55
Base IO
Config INFF: I1: I2: OUT:O PAD: TRI: O: OSPEED:SLOW
Endblk
Nameblk PAD3 $1I69/PAD<1>
Editblk PAD3
Base IO
Config INFF: I1: I2: OUT:O PAD: TRI: O: OSPEED:SLOW
Endblk
Nameblk PAD54 $1I69/PAD<2>
Editblk PAD54
Base IO
Config INFF: I1: I2: OUT:O PAD: TRI: O: OSPEED:SLOW
Endblk
Nameblk PAD53 $1I69/PAD<3>
Editblk PAD53
Base IO
Config INFF: I1: I2: OUT:O PAD: TRI: O: OSPEED:SLOW
Endblk
Nameblk PAD50 $1I89/PAD
Editblk PAD50
Base IO
Config INFF: I1: I2:I OUT: PAD: TRI: O: OSPEED:
Endblk
Nameblk PAD43 P_A/PAD<0>
Editblk PAD43
Base IO
Config INFF: I1: I2:I OUT: PAD: TRI: O: OSPEED:
Endblk
```

```
Nameblk PAD44 P_A/PAD<1>
Editblk PAD44
Base IO
Config INFF: I1: I2:I OUT: PAD: TRI: O: OSPEED:
Endblk
Nameblk PAD41 P_A/PAD<2>
Editblk PAD41
Base IO
Config INFF: I1: I2:I OUT: PAD: TRI: O: OSPEED:
Endblk
Nameblk PAD48 P_A/PAD<3>
Editblk PAD48
Base IO
Config INFF: I1: I2:I OUT: PAD: TRI: O: OSPEED:
Endblk
Nameblk PAD49 $1I166/PAD
Editblk PAD49
Base IO
Config INFF: I1: I2: OUT: PAD: TRI: O: OSPEED:
Endblk
System Xdelay Timegroup ffs -FF C<0> C<2> C<1> C<3> E<0> E<2> E<1>
E<3>
System Xdelay Timegroup ffs -FF LFSR_4COUNT<2> LFSR_4COUNT<0>
LFSR_4COUNT<3>
System Xdelay Timegroup ffs -FF LFSR_4COUNT<1> B<0> B<1> B<2> B<3>
System Xdelay Timegroup pads -IOB $1I128/PAD $1I132/PAD $1I136/PAD
$1I150/PAD
System Xdelay Timegroup pads -IOB $1I154/PAD<0> $1I154/PAD<1>
$1I154/PAD<2>
System Xdelay Timegroup pads -IOB $1I154/PAD<3> $1I155/PAD<0>
$1I155/PAD<1>
System Xdelay Timegroup pads -IOB $1I155/PAD<2> $1I155/PAD<3>
$1I163/PAD
System Xdelay Timegroup pads -IOB $1I166/PAD i_bufgs_bl $1I168/PAD
$1I69/PAD<0>
System Xdelay Timegroup pads -IOB $1I69/PAD<1> $1I69/PAD<2>
$1I69/PAD<3>
System Xdelay Timegroup pads -IOB $1I89/PAD P_A/PAD<0> P_A/PAD<1>
P_A/PAD<2>
System Xdelay Timegroup pads -IOB P_A/PAD<3>
System Xdelay TimeSpec DEFAULT_FROM_FFS_TO_FFS ffs ffs   0.0
System Xdelay TimeSpec DEFAULT_FROM_PADS_TO_FFS pads ffs   0.0
System Xdelay TimeSpec DEFAULT_FROM_FFS_TO_PADS ffs pads   0.0
```

```
Intnet PAD42 PAD $1I128/PAD
Intnet JC F SYNC_1/OR2_REG_ff_1
Intnet JC H SYNC_1/OR3_REG_ff_1
Intnet IB F SYNC_1/OR6_REG_ff_3
Intnet JE F SYNC_1/AND9_REG_ff_2
Intnet JE G SYNC_1/AND10_REG_ff_2
Intnet JE H SYNC_1/BUF4_REG_ff_2
Intnet JD F SYNC_1/BUF2_REG_ff_0
Intnet PAD8 PAD $1I132/PAD
Intnet HE F ASYNC/BUF4_SAT_ffy_1
Intnet HE G ASYNC/BUF2_SAT_ffx_0
Intnet GE F ASYNC/BUF8_SAT_ffy_3
Intnet GE G ASYNC/BUF6_SAT_ffx_2
Intnet PAD7 PAD $1I136/PAD
Intnet PAD6 PAD $1I150/PAD
Intnet HD F LFSR_FDBK/XNOR
Intnet PAD31 PAD $1I154/PAD<0>
Intnet JB F SYNC_1/SUM<3>
Intnet JB G SYNC_1/AND12_REG_ff_3
Intnet PAD30 PAD $1I154/PAD<1>
Intnet PAD45 PAD $1I154/PAD<2>
Intnet PAD47 PAD $1I154/PAD<3>
Intnet PAD28 PAD $1I155/PAD<0>
Intnet PAD27 PAD $1I155/PAD<1>
Intnet PAD26 PAD $1I155/PAD<2>
Intnet PAD25 PAD $1I155/PAD<3>
Intnet PAD56 PAD $1I163/PAD
Intnet HC F DECIMAL_1/BUF5_SAT_ffx_1
Intnet GB F DECIMAL_1/BUF4_SAT_ffy_0
Intnet GC F DECIMAL_1/BUF3_SAT_ffx_0
Intnet HB F DECIMAL_1/SUM<3>
Intnet HB G TC_DEC
Intnet HB H DECIMAL_1/BUF6_SAT_ffy_1
Intnet PAD52 PAD $1I168/PAD
Intnet PAD55 PAD $1I69/PAD<0>
Intnet PAD3 PAD $1I69/PAD<1>
Intnet PAD54 PAD $1I69/PAD<2>
Intnet PAD53 PAD $1I69/PAD<3>
Intnet PAD50 PAD $1I89/PAD
Intnet PAD43 PAD P_A/PAD<0>
Intnet PAD44 PAD P_A/PAD<1>
Intnet PAD41 PAD P_A/PAD<2>
Intnet PAD48 PAD P_A/PAD<3>
```

```
;: END OF FILE
```

Anlage B: volladde.lca

```
;: volladde.lca (4002APC84-5), ppr
Xilinx:ppr:5.2.0:95/10/23, 1998/05/18 22:14:48

Version 2
Design 4002APC84
Speed -5
Addnet CIN PAD49.I2 JB.F1 JB.G1
NProgram CIN JB.F1:col.B.local.2 JB.G1:col.B.local.2
PAD49.I2:col.B.local.2
Addnet S PAD1.O JB.X
NProgram S PAD1.O:col.B.long.2 row.F.local.1:col.B.long.2
row.K.local.3:col.B.long.2-L JB.X:row.K.local.3
Addnet COUT PAD50.O JB.Y
NProgram COUT PAD50.O:col.B.local.5 JB.24.1.7 JB.24.1.13
JB.Y:row.J.local.2
Addnet B PAD32.I2 JB.F4 JB.G4
NProgram B JB.F4:row.J.long.0 JB.G4:row.J.long.0
row.J.long.0:col.F.local.1 col.K.local.5:row.J.long.0-L
PAD32.I2:col.K.local.5
Addnet A PAD48.I2 JB.F2 JB.G2
NProgram A JB.F2:row.K.local.2 JB.G2:row.K.local.2
PAD48.I2:row.K.local.2
Nameblk PAD49 $1N73
Editblk PAD49
Base IO
Config INFF: I1: I2:I OUT: PAD: TRI: O: OSPEED:
Endblk
Nameblk JB S
Editblk JB
Base FG
Config F4:F4I G2:G2I G3: CIN: COUT: X:F Y:G XQ: YQ:
FFX:RESET FFY:RESET DX: DY: F:F1:F4:F2 G:G1:G4:G2 H: H1:
DIN: SR: EC: RAM: CDIR:
Equate F = ((F4@F2)@F1)
```

```
Equate G = (((G4@G2)*G1)+(G4*G2))
Endblk
Nameblk PAD1 $1N71
Editblk PAD1
Base IO
Config INFF: I1: I2: OUT:O PAD: TRI: O: OSPEED:SLOW
Endblk
Nameblk PAD50 $1N69
Editblk PAD50
Base IO
Config INFF: I1: I2: OUT:O PAD: TRI: O: OSPEED:SLOW
Endblk
Nameblk PAD32 $1N75
Editblk PAD32
Base IO
Config INFF: I1: I2:I OUT: PAD: TRI: O: OSPEED:
Endblk
Nameblk PAD48 $1N67
Editblk PAD48
Base IO
Config INFF: I1: I2:I OUT: PAD: TRI: O: OSPEED:
Endblk
System FGG 0 VERS 2 !
System FGG 1 GD0 0 !
System FGG 2 GD2 2 M:B M:A K:E !
System FGG 3 GD2A $IGNORE16_E S !
System FGG 4 GD2A $IGNORE15_E G !
System FGG 5 GD2E !

Intnet PAD49 PAD $1N73
Intnet PAD1 PAD $1N71
Intnet PAD50 PAD $1N69
Intnet PAD32 PAD $1N75
Intnet PAD48 PAD $1N67

;: END OF FILE
```

Erklärung

Ich versichere, daß ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, September 1998