Ruprecht-Karls-Universität
Heidelberg

Kirchhoff-Institut für Physik

Entwurf und Implementierung von Building Blocks zum Einsatz in Genetischen Algorithmen auf einem FPTA

Dokumentation
zum Informatikpraktikum
ausgeführt von
Daniel Brüderle
September/Oktober 2002

daniel@bruederle.de

HD-KIP 02-25

Inhalt

In der Arbeitsgruppe *Electronic Vision(s)* im Kirchhoff-Institut für Physik werden u.a. elektronische Schaltungen mit Hilfe von Genetischen Algorithmen auf einem FPTA-Chip entwickelt. Diese Dokumentation beschreibt Ablauf und Ergebnis einer vierwöchigen Arbeit im Rahmen eines Informatikpraktikums. Das bereits vorhandene Programm DarkGAQT mit dem Genetischen Algorithmus als Kern ist für den Einsatz von fest vorzugebenden Bausteinen, sogenannten Building Blocks, erweitert worden. Eine Bibliothek von sinnvollen Building Blocks ist erstellt, die Funktionalität im Experiment gezeigt worden.

Inhaltsverzeichnis

Kapitel 1

Motivation

1.1 Aufwendige Entwicklung von Hardware

Im Gegensatz zu dem in weiten Teilen automatisierten Prozeß des digitalen Schaltungsentwurfes liegt das Problem bei der Entwicklung von analogen Bauteilen tiefer. Deren Entwurf ist bestimmt von häufigen Simulationen der entwickelten Schaltungen sowie daraus folgenden Nachkorrekturen und daher sehr zeitintensiv. Unter anderem muß Prozeßvariationen und parasitären Kapazitäten auf dem Chip Rechnung getragen werden, was diese Simulationen recht aufwendig werden läßt. Falls die entwickelte Schaltung als realer Prototyp getestet werden soll, so kostet auch dessen Herstellung Zeit. Mit Hilfe von sogenannten Genetischen Algorithmen versucht man eine Methode zu entwickeln, den Entwurfsprozeß zu beschleunigen und trotz Unwissenheit um die Prozeßvariationen diese in das Design miteinzubeziehen. Grundlage dafür ist ein rekonfigurierbarer Chip, der es erlaubt, das zu entwickelnde Bauteil während des Entwurfsprozesses in realer Hardware zu testen. Bei dem am Kirchhoff-Institut entwickelten Field Programmable Transistor Array (FPTA) handelt es sich um einen solchen rekonfigurierbaren Chip.

1.2 Die Experimentierplattform FPTA

Der FPTA-Chip besteht im Kern aus einem Feld von 16x16 Zellen, die jeweils einen Transistor sowie verschiedene Routingmöglichkeiten tragen (vgl. Abb. 1.1). Dabei sind PMOS und NMOS Transistoren in einem Schachbrettmuster angeordnet. Die Transistoren sind sowohl in ihren charakteristischen Größen W und L als auch in der Verschaltung der einzelnen Terminals (Source, Drain, Gate) programmierbar. Die quadratischen Zellen können mit Hilfe der ebenfalls programmierbaren Routingleitungen untereinander verschaltet werden (vgl. Abb. 1.2). Der Chip befindet sich auf einer PCI-Karte, die mittels eines FPGA[1] die Konfigurationsdaten vom Rechner an den FPTA weiterleitet, die Eingangssignale erzeugt sowie die Ausgangssignale zurückgibt (vgl. Abb. 1.3). Damit liegt ein elektronisches Bauteil vor, auf dem eine Vielzahl elektronischer Schaltungen realisiert werden kann.

[1] FPGA = Field Programmable Gate Array

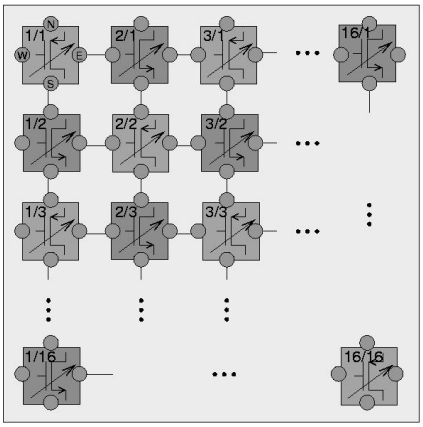

Abbildung 1.1: Schematische Darstellung des Transistorarrays (aus [Lang 02]). Die einzelnen Zellen sind an den Randpins (N, E, S, W) mit ihren jeweiligen Nachbarn verbunden.

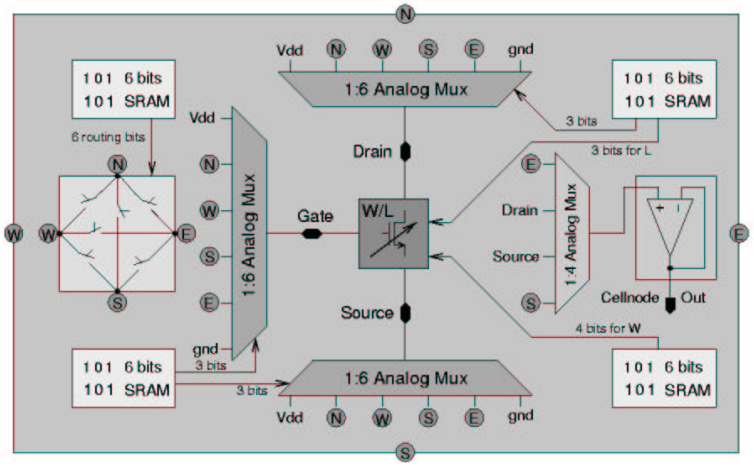

Abbildung 1.2: Schematische Darstellung der programmierbaren Transistorzelle (aus [Lang 02]). In der Mitte befindet sich der Transistor mit den programmierbaren Werten W und L. Die Transistorterminals können über Multiplexer an jeden der vier Randpins (N, E, S, W) oder an die Versorgungsspannung (VDD) bzw. an Ground (GND) verschaltet werden. Innerhalb der Zelle ist ein Routing von jedem Randpin zu einem beliebigen anderen möglich.

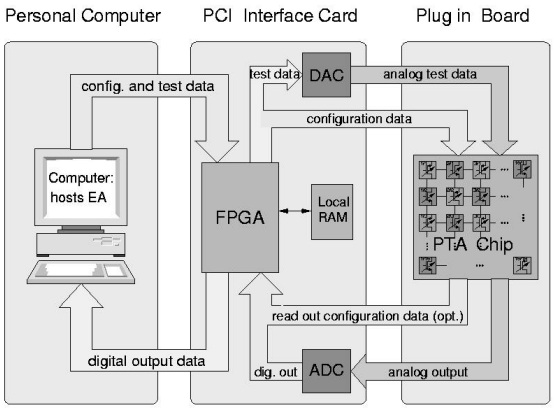

Abbildung 1.3: Die Experimentierumgebung (aus [Lang 02]).

1.3 Genetische Algorithmen

1.3.1 Vorbild Natur

Das Leben hat es geschafft, auf der Erde über Jahrmilliarden ununterbrochen zu bestehen, und das trotz teilweise widrigster und manchmal extrem schnell variierender Bedingungen. Dabei hat sich eine enorme Vielfalt verschiedenster Lebensformen entwickelt, die allesamt den Anforderungen ihrer Umwelt auf oft sehr unterschiedliche Art und Weise gerecht werden. Der komplexe Mechanismus der Evolution macht diese stetige Anpassung an sich ändernde Bedingungen möglich. Basis dafür ist die Kodierung aller vererblichen Merkmale eines Lebewesens in Form von Genen. Mittels der sexuellen Fortpflanzung (vor allem im Falle der Pflanzenwelt auch der asexuellen oder vegetativen Fortpflanzung, bei der ähnliche Mechanismen wirken) erzeugt eine Generation einer bestimmten Spezies eine neue Generation. Die nahezu unendlich hohe Anzahl von möglichen Genomen sowie die bei der Fortpflanzung wirkenden Mechanismen lassen praktisch keine identischen Individuen zu. Im Rahmen einiger für die Spezies typischen Merkmale unterscheiden sie sich derart, daß ihre Fähigkeiten zu überleben ebenfalls unterschiedlich ausfallen. Dies verhindert langfristig das Überleben und sich Fortpflanzen schwacher Individuen. Neben diesem "survival of the fittest" hat sich bei vielen Tieren die Partnerwahl nach bestimmten Merkmalen als weiterer Weg der Selektion entwickelt. Die erwähnten Mechanismen, die zu unterschiedlichen Gensequenzen führen, sind erst mit Kenntnis der Bildung eines neuen Genoms aus den Genen zweier Eltern zu verstehen. Dabei übernimmt das Kind Gensequenzen sowohl vom Vater als auch von der Mutter. Dieses sogenannte *Crossover* ist ein zufallsgesteuerter Prozeß und nur teilweise, z.B. was Ort und Größe der zusammenhängenden Informationssequenzen angeht, festgelegt. Doch vor allem durch die Mutation, d.h. durch die zufällige Veränderung, Zerstörung oder Erweiterung eines Quantums (vier verschiedene Basen bilden Alphabet: A T C G) oder ganzer Bereiche der vererbten Information kann das Einbringen wirklich neuer Wege zur Bewältigung von veränderten Anforderungen gewährleistet werden. Diese Mutationen werden verursacht durch Umwelteinflüsse oder Fehler im chemischen Prozeß der Vererbung.

1.3.2 Abstraktion, Nutzung im Hardware Design

Die Evolution kann also als erfolgreiches Suchverfahren nach Optima in einem sehr großen Suchraum beschrieben werden. Die unzähligen Möglichkeiten von Konfigurationen auf dem FPTA stellen einen solchen Suchraum dar. Dabei will man hier, anders als in der Natur, deren einziges erkennbares Ziel das Überleben darstellt, auf dem Chip Schaltungen entwerfen, die ganz konkrete Aufgaben lösen. Zum Entwickeln und Verbessern der Evolutionsstrategie bieten sich beispielsweise logische Gatter (AND, OR, XOR...) als einfachste digitale Vertreter oder ein Komparator an. Die komplette Konfiguration eines Chips repräsentiert das Genom (vgl. Abb. 1.4). Genetische Algorithmen (im folgenden GA) versuchen nun, die prinzipiellen Mechanismen der Evolution (Crossover, Mutation, Selektion) zu nutzen und damit den Suchraum einer vorher festzulegenden Fitneßverteilung nach Maxima zu durchforsten, die den gestellten Anforderungen bestmöglich genügen. Dazu wird im Falle des FPTA eine Zielfunktion definiert, die der Chip nach Anwendung des Algorithmus möglichst gut erfüllen soll. Dann wird eine Anfangspopulation, also eine bestimmte Anzahl von Chipkonfigurationen, zufällig erzeugt. Jede weitere Generation wird nun aus der vorhergehenden folgendermaßen gebildet:

Die Individuen der alten Generation werden sukzessiv auf den Chip geladen und ihre Fitneß mittels einer Sequenz von Eingangssignalen und Messung der daraus folgenden Ausgangs-

signale evaluiert. Ein fester Bruchteil der Individuen mit der größten Fitneß wird unverändert in die neue Generation übernommen. Diese wird weiter aufgefüllt mit Individuen, deren Bitstränge sich aus zwei Elternsträngen der älteren Generation zusammensetzen, und solchen, die durch bitweise Mutation eines Individuums aus der älteren Generation entstanden sind (vgl. Abb. 1.5). Dabei rekrutieren sich die Lieferanten für Crossover und Mutation wiederum aus einem definierten Bruchteil der besten Individuen der alten Generation (vgl. Abb. 1.6).

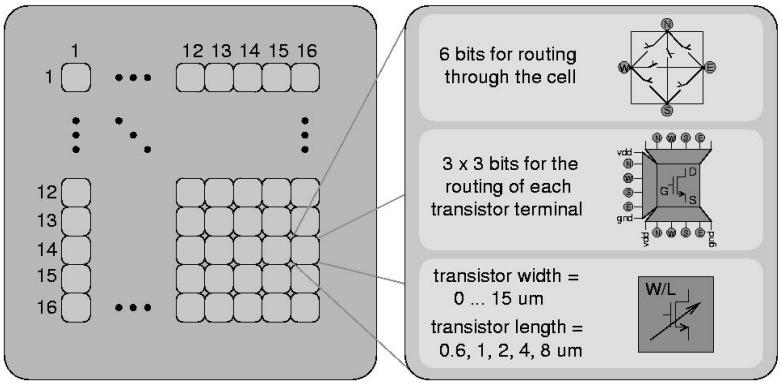

Abbildung 1.4: Das Genom einer FPTA-Konfiguration wird gebildet aus der Summe aller Einzelkonfigurationen (=Gene) der Transistorzellen auf dem Chip (aus [Lang 02]).

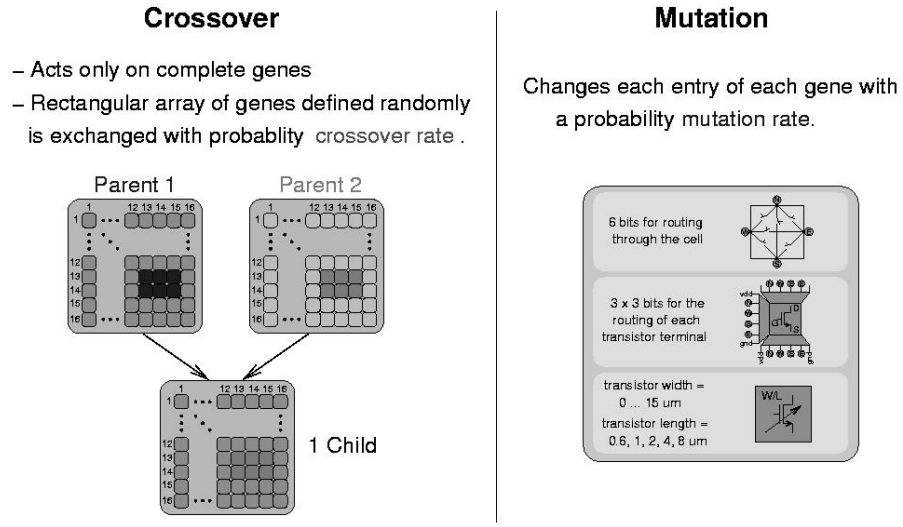

Abbildung 1.5: Evolutionäre Mechanismen auf dem FPTA: Crossover (links) und Mutation (rechts) (aus [Lang 02]).

GAs mit den erwähnten Mechanismen wurden schon länger in Software angewandt und weiterentwickelt. Auf dem Gebiet des Hardwaredesigns sind sie jedoch erst seit der Verfügbarkeit von rekonfigurierbaren Bauteilen so einsetzbar, daß die Evaluierung der Fitneß auf dem

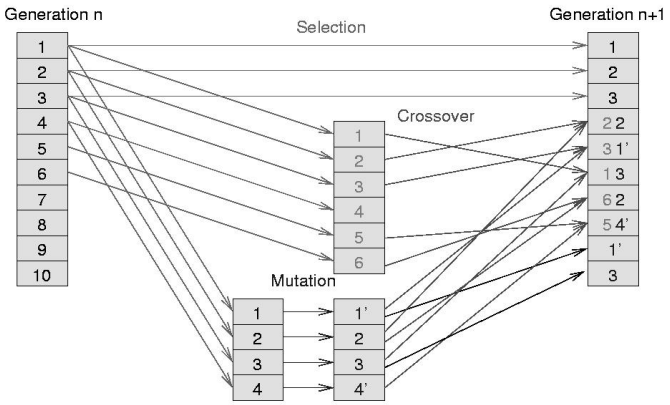

Abbildung 1.6: Abstraktion der natürlichen Fortpflanzung: Erzeugung einer neuen Generation (aus [Lang 02]).

realen Bauteil stattfinden kann. Der FPTA-Chip stellt hier eine ideale Plattform für solche evolutionären Strategien dar, da seine Wandlungsfähigkeit bereits auf der Ebene der Transistoren ansetzt und er speziell für den Einsatz von GAs entwickelt und in eine geeignete Experimentierumgebung eingebettet wurde. Vielversprechende Ergebnisse liegen bereits vor, siehe dazu [Lang 02].

Kapitel 2

Durchführung

2.1 Überblick

Beschrieben wird im folgenden die Modifikation des Programmes DarkGAQT, die die Nutzung von sogenannten Building Blocks im GA erlaubt. Die elementaren Bausteine sind dann nicht mehr die einzelnen Transistorzelleneigenschaften, sondern ganze Building Blocks. Anders als bisher kann sich der Algorithmus im Rahmen der Mutation aus einer Bibliothek von vorgefertigten Bausteinen bedienen, um bestimmte Bereiche des Chips zu füllen. Diese bleiben in ihrer inneren Struktur unveränderlich und können nur in Ausrichtung und Position vom GA manipuliert werden. Solche Bausteine heißen im folgenden "Building Blocks". Bereiche des Chips, die nicht mit diesen Bausteinen belegt werden, erfüllen nur Verdrahtungsfunktion. Die Verschaltung selbst unterliegt ebenfalls dem GA.

Langfristiges Ziel ist es, durch den Einsatz solcher Bausteine die Ergebnisse der evolutionären Designstrategie besser nachvollziehbar zu machen und den Entwurfsprozeß zu beschleunigen.

Zunächst wird eine sinnvolle Bibliothek entworfen und die erlaubten Positionen der Bausteine auf dem Chip werden festgelegt. Dabei muß ein Verhältnis von Building Blocks zu Zellen mit reiner Verdrahtungsfunktion gefunden werden, das dem Algorithmus die Erzeugung eines sinnvollen Signalflusses ermöglicht. Einzelne Entwürfe solcher Bibliotheken und Anordnungsschemata werden durch Realisierung einfacher Schaltungen wie der eines Komparators per Hand auf Tauglichkeit überprüft.

Dann wird die Steuersoftware des Chips, die für die Initialisierung der Population, den GA, die Eingangssignalerzeugung und Evaluierung der Individuen zuständig ist, modifiziert. Dabei wird eine Möglichkeit geschaffen, die zuvor entworfene Bibliothek in Software zu repräsentieren, abzurufen und dem GA zugänglich zu machen. Der GA selbst wird ebenfalls verändert. Er muß Building Blocks und Verdrahtungszellen zu unterscheiden wissen. Er soll im Falle der Mutation von Building Blocks nur ganze Blocks austauschen beziehungsweise im Falle der Mutation von Routing Zellen nur Routing Bits verändern. Während des Crossover dürfen nur ganze Bausteine oder Blöcke von Bausteinen ausgetauscht werden.

2.2 Entwurf einer Building Block Bibliothek

Die Auswahl der Building Blocks beschränkt sich auf die elementaren Ein- und Zwei-Transistorschaltungen der CMOS-Technik. Die Verwendung der symmetrischen Strukturen

Stromspiegel und Differentielles Paar prägen den analogen Schaltungsentwurf. Der GA soll zu diesem symmetrischen Entwurfsstil mit Hilfe der Building Block Bibliothek angeleitet werden.

Folgende Schaltungen sollen in jedem Falle in der Bibliothek enthalten sein:

Einfacher Transistor (in PMOS und NMOS)
Inverter
Stromquelle und -senke
Differentielles Paar (PMOS und NMOS)
Stromspiegel (PMOS und NMOS)

Als Blockgröße wird 2x2 Zellen gewählt, da absehbar ist, daß alle gewünschten Bausteine so zu realisieren sind. Das Routing wird ebenfalls in Einheitsgrößen von 2x2 stattfinden, da man somit der Chipsymmetrie mit der schachbrettartigen PMOS/NMOS-Anordnung gerecht wird. Solche Blöcke reiner Verdrahtungszellen werden im folgenden *Routing Block* genannt. Die Anordnung der Building Blocks ist nicht so naheliegend. Da der Signalfluß innerhalb der Blocks eine Vorzugsrichtung bekommen wird, stellt sich die Frage, ob die Blocks so entworfen werden können, daß sie in dieser Vorzugsrichtung ohne zwischengeschaltete Routing Zellen direkt aneinander gekoppelt sinnvolle Schaltungen ergeben. Dann wäre beispielsweise eine spaltenweise abwechselnde Anordnung von Routing und Building Blocks sinnvoll. Doch beim Versuch, solche Blöcke zu entwerfen, zeigt sich, daß hierbei in der Building Block Bibliothek nicht auf reine Routing Blocks verzichtet werden kann. Außerdem müßten innerhalb der Building Blocks die einzelnen Signale oft an mehreren Randpins gleichzeitig ausgegeben werden. Letzteres könnte den GA dazu verleiten, diese Verbindungen zwischen den Pins als Routing zu "mißbrauchen" und somit vor allem die Nachvollziehbarkeit des Ergebnisses einzuschränken. Also ist die Entscheidung letztlich auf eine zeilenweise Anordnung der Building Blocks gefallen, jedoch mit einem senkrecht dazu verlaufenden Signalfluß innerhalb der Blöcke. (Ob zeilen- oder spaltenweise spielt eigentlich keine Rolle, auf den zur Anordnung der Building Blocks *senkrechten* Signalfluß innerhalb der Blöcke kommt es an.) Dies ermöglicht, Schaltungen wie einen Komparator in recht übersichtlicher Weise und mit meist weniger Platzbedarf als in der zuerst beschriebenen Strategie zu realisieren.

2.3 Graphischer Baukasten

Abb. 2.1 zeigt die vorläufige Building Block Bibliothek. Sie wurde mit dem Programm *xfig* erstellt und bildet, zusammen mit einem ebenfalls erstellten Raster für den Chip, ein graphisches Baukastensystem. Mit dessen Hilfe können nun relativ bequem Lehrbuchschaltungen nachempfunden und somit die Funktionalität der Bibliothek wie auch des geplanten Anordnungsmusters verifiziert werden. Die Abbildungen 2.2 und 2.3 zeigen solche Schaltungen. Zu sehen ist ein XOR-Gatter als komplexer Repräsentant eines logischen Gatters sowie eine Folded Cascode als Beispiel fr eine nicht primitive Verstärkerschaltung. Zum besseren Verständnis der Schaltungen wie auch der Bausteine selbst sei folgende Anmerkung gemacht: Innerhalb jeder Zelle können oberer (N für North), unterer (S für South), rechter (E für East) und linker Rand (W für West) beliebig untereinander verschaltet werden (vgl. Abb. 1.2). Zwischen den einzelnen Zellen sind direkt benachbarte der gerade erwähnten Knotenpunkte, also beispielsweise Punkt W einer Zelle mit Punkt E einer linksseitig (falls Orientierung wie in Abb. 1.1) benachbarten Zelle miteinander verbunden. Da diese Randpunkte neben den

Transistorterminals die einzigen Möglichkeiten sind, Verbindungen herzustellen, sind in den mit xfig dargestellten Schaltungen nur Kreuzungen von Linien, die am Rand einer Zelle liegen, als leitende Verbindungen zu betrachten.

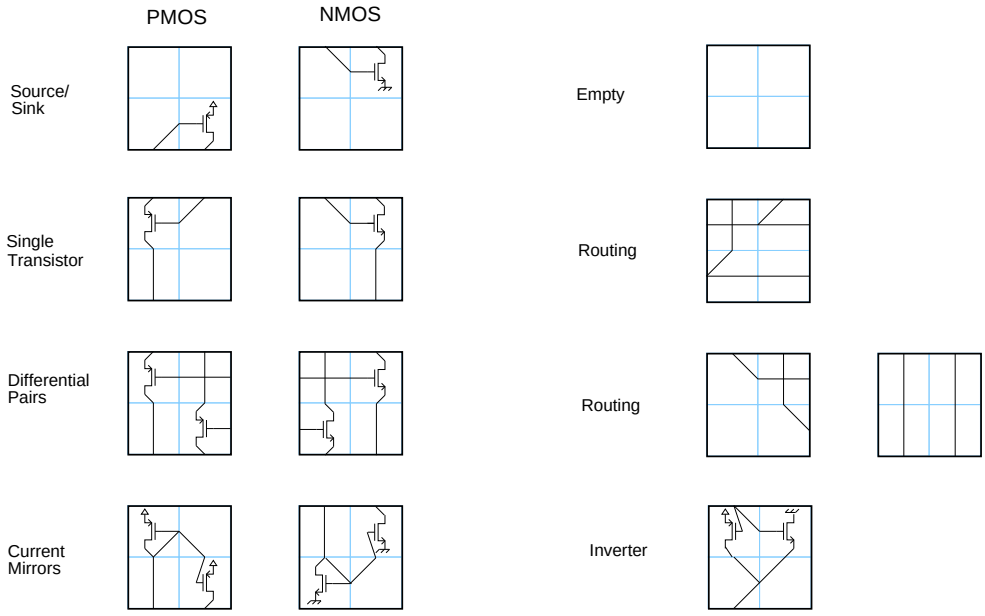

Abbildung 2.1: Building Block Library.

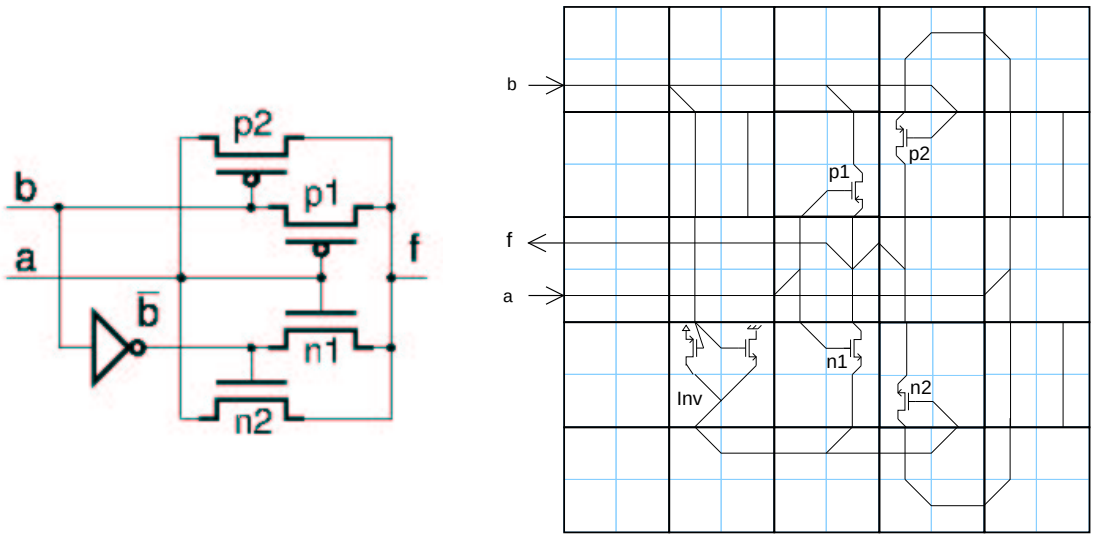

Abbildung 2.2: Links: XOR-Gatter Schematic, aus [Papli 01]. Rechts: XOR-Gatter, realisiert mit dem xfig-"Baukasten".

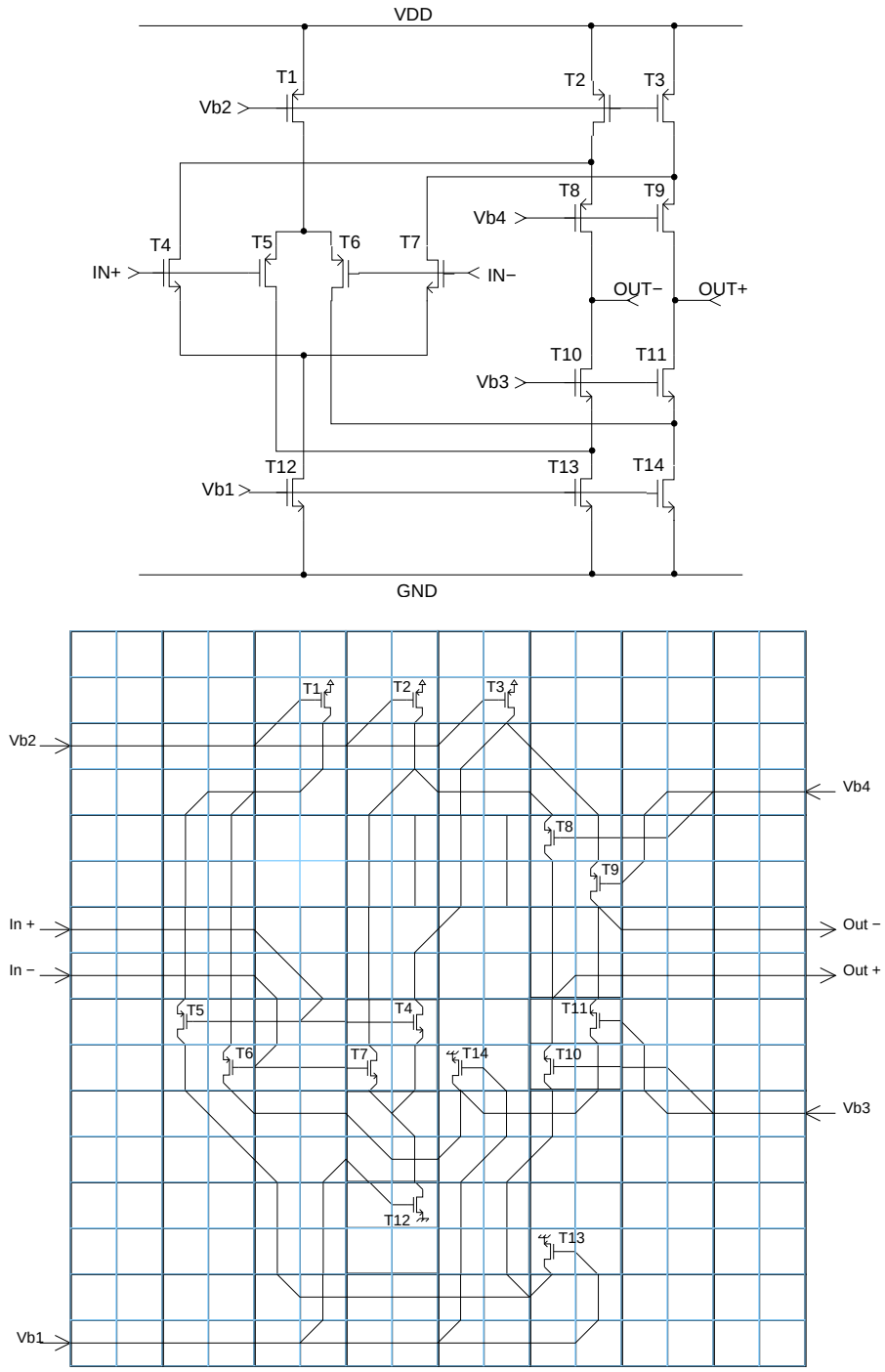

Abbildung 2.3: Oben: Folded Cascode Schematic, aus [LakSan 94]. Unten: Folded Cascode, realisiert mit dem xfig-“Baukasten“.

2.4 Das Programm DarkGAQT

2.4.1 Ursprüngliche Funktionalität

Das Programm DarkGAQT liegt als Sourcecode in der Programmiersprache C++ zu Beginn der Arbeit vor. Es umfaßt die komplette Ansteuerung des PTA-Chips sowie den GA. Darin existiert eine Klasse namens *PopManFPTA* (für "Population Manager"), die die genetischen Operationen Mutation, Crossover und Selektion durchführt. Dazu werden Parameter wie beispielsweise die Mutationsrate oder der Anteil der unverändert übernommenen Individuen benötigt. Sie werden dem Programm entweder mittels Konsole beim Programmstart übergeben oder durch Standardwerte belegt. Man kann unterschiedliche Raten für die Mutation der Transistorterminals, des Routings oder der Transistorparameter W und L einstellen. Der Population Manager ist auch für die Initialisierung der ersten Generation zuständig, bei der alle einstellbaren Werte einer Zelle zufällig belegt werden. Wird das Programm ausgeführt, so kann der Nutzer über eine Oberfläche den Evolutionsprozeß starten und stoppen, er kann die Generationen und einzelne Individuen einsehen und beliebig manipulieren sowie diverse Parameter für die Fitneßevaluation einstellen.

2.4.2 Erweiterung um Building Blocks und Bibliothek

Zur Verwendung von Building Blocks muß das Programm nun eine Repräsentation eben jener erhalten. Dies geschieht durch Einfügen einer Klasse *BuildBlock*, die aus einem Array beliebig einstellbarer Größe von Transistorzellen sowie elementaren Operatoren auf dieses Array besteht. Zu diesen Operationen gehören das Spiegeln an X- und Y-Achse sowie eine Rotation um 180 Grad. Beim Spiegeln werden wegen der Überführung von PMOS in NMOS und umgekehrt die eventuell vorhandenen Anschlüsse VDD und GND in den jeweils anderen verwandelt. So ist eine sinnvolle Transformation von PMOS- in NMOS-Bausteine möglich und die Bibliothek muß nicht so viele Elemente umfassen. Die Rotation wird durch eine Hintereinanderausführung der beiden Spiegeloperationen realisiert. Die Werte für die Größe der Building Blocks liest das Programm aus einer Datei namens "bb_config.cfg" ein (falls diese nicht vorhanden ist, gilt der Standardwert 2x2), ebenso die Anordnung der Building Blocks auf dem FPTA. Diese Anordnung wird programmintern durch ein zweidimensionales Integerfeld erfaßt und ist für die Bildung einer neuen Generation aus der alten sowie für die Initialisierung unverzichtbar. Das Programm DarkGAQT ist nun so modifiziert, daß durch Kommandozeilenübergabe der Einsatz von Building Blocks aktiviert oder deaktiviert werden kann (Standard: deaktiviert).

Außerdem wird eine Klasse *BB_Library* implementiert, die neben einem Array (Größe ebenfalls aus Datei "bb_config.cfg" einlesbar) von Building Blocks einige Funktionen zur Verwaltung dieser Bibliothek besitzt. Dazu gehören Hinzufügen eines neuen Building Blocks, Entfernen eines Building Blocks, Abrufen eines Building Blocks, Ausgabe der aktuellen Bibliotheksgröße sowie das Löschen der gesamten Bibliothek. Außerdem kann eine vormals abgespeicherte Bibliothek aus einer Datei geladen werden.

2.4.3 Modifikation des Genetischen Algorithmus

Im Falle eines Evolutionsdurchlaufes *mit* Building Blocks durchläuft der Algorithmus bei der Erstellung einer neuen Generation oder auch bei der Initialisierung nicht wie sonst jede einzelne Transistorzelle, sondern zunächst das zweidimensionale Integerfeld. Dieses teilt den

Chip in Building bzw. Routing Blocks ein und trägt dementsprechend eine 1 oder eine 0 (vgl. Tab. 2.1).

1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0

Tabelle 2.1: Integerfeld, das die standardmäßige Verteilung der Building Blocks auf dem Chip darstellt.

Soll nun ein Routing Block mutiert werden, so werden die Transistorterminals vom Algorithmus nicht verändert (bleiben also unverbunden), auch die Größen W und L werden nicht manipuliert. Lediglich für das Routing läuft der übliche Mutationsmechanismus ab, der sich über die programmierbare Mutationsrate speziell für das Routing steuern läßt. Handelt es sich um einen Building Block, so wird nur eine der drei folgenden Operationen durchgeführt:

Entweder wird der vorhandene Building Block um 180 Grad rotiert, an der X-Achse gespiegelt oder durch einen zufällig aus der Bibliothek entnommenen Building Block ersetzt. Das Spiegeln an der Y-Achse kann vom Algorithmus durch die Hintereinanderausführung der Rotation und der Spiegelung an der X-Achse erreicht werden. Alle drei Funktionen haben eigene programmierbare Mutationsraten.

Die Initialisierung erfolgt nach ähnlichem Prinzip. Es wird zunächst das Integerfeld durchlaufen und dann im Falle eines Routing Blocks ein zufälliges Routing gewählt sowie alle Terminalanschlüsse des Blocks deaktiviert oder im Falle eines Building Blocks ein zufällig aus der Bibliothek entnommener Block eingesetzt.

Die Funktionen *Spiegeln eines Building Blocks an der X-Achse*, *Spiegeln eines Building Blocks an der Y-Achse*, *Hinzufügen eines Building Blocks zur Bibliothek* sowie *Löschen aller Bibliothekseinträge* sind auch als Buttons in die Benutzeroberfläche integriert (vgl. Abb. 2.4). Die drei erstgenannten Funktionen wirken dabei auf den unteren rechten angezeigten Building Block (siehe auch Kapitel *Ausblick*).

2.4.4 Flexibilität

Mit den vorgenommenen Veränderungen ist der Nutzer nun in der Lage, nahezu beliebig dimensionierte Building Blocks (ihre X- bzw. Y-Ausdehnung in Einheiten von Transistorzellen muß ein Vielfaches von 2 sein, sonst funktionieren die Spiegel- und Rotationsoperatoren nicht) in selbst gewählter Anordnung und Anzahl auf dem Chip zu platzieren. Das Verhältnis von Building Blocks zu Routing Blocks kann beliebig gewählt werden. Der GA wird diese Vorgaben strikt einhalten. Über die Benutzeroberfläche kann eine eigene Bibliothek beliebigen Umfangs erstellt werden.

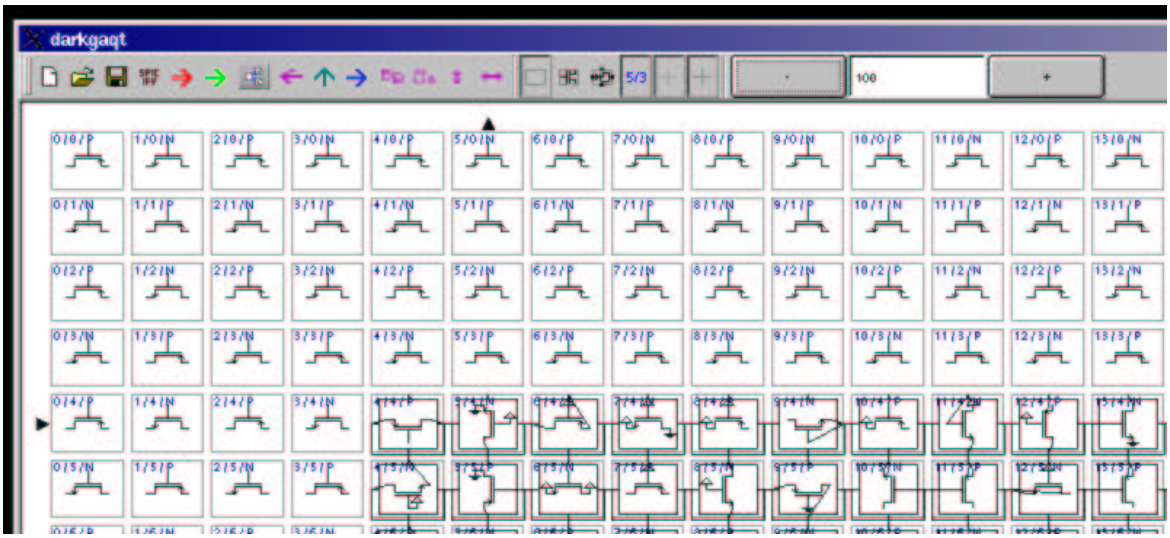

Abbildung 2.4: Bearbeitungsfenster (Ausschnitt) der Chipkonfiguration in der Benutzeroberfläche DarkGAQT. Neu eingefügte Schaltflächen: "BB" fügt den aktuellen (im Bild nicht zu sehen) Building Block zur Bibliothek hinzu, "cl lib" löscht die aktuelle Bibliothek, "↕" spiegelt den aktuellen Building Block an der X-Achse, "↔" spiegelt den aktuellen Building Block an der Y-Achse.

2.5 Experiment mit und ohne Einsatz von Building Blocks

Es werden diverse Probeläufe mit Building Block-Einsatz durchgeführt. Folgende kleine Experimentreihe soll belegen, daß für (verglichen mit logischen Gattern) komplexere Schaltungen, hier ein Komparator, der Einsatz von Building Blocks zu besseren Ergebnissen führt. Auch soll empirisch ein möglichst guter Wert für den Anteil der unverändert übernommenen Individuen gefunden werden.

2.5.1 Ablauf, Parameter

Der Chip wird mit zwei Eingängen belegt, die zur Evaluierung folgende Inputsequenz erhalten:

V_{in1} von 1.5 bis 4.5 V in 10 Schritten

V_{in2} von $V_{in1} - 0.5\,\mathrm{V}$ bis $V_{in1} + 0.5\,\mathrm{V}$ in 50 Schritten für jeden Wert von V_{in1}

Die Werte für V_{in2} werden in zufälliger Reihenfolge generiert.

Folgende Zielfunktion wird angestrebt:

$$V_{out} = \begin{cases} 0\,\mathrm{V} & \text{falls } \mathrm{V_{in1}} < \mathrm{V_{in2}} \\ 5\,\mathrm{V} & \text{falls } \mathrm{V_{in1}} > \mathrm{V_{in2}} \end{cases}$$

Als Fitneß wird die Summe der Abweichungsquadrate angesetzt, diese muß also *minimiert* werden:

$$RMS\ Error = \sqrt{\frac{\sum_{i=1}^{500}(V_{tar}(i) - V_{out}(i))^2}{500}}\ \text{x } 1000$$

Verwendet wird nur eine Hälfte des Chips.

In Tab. 2.2 sind die Parameter des GA für die Experimentreihe zu sehen.

Experiment Nr.	1-10	11-20	21-30	31-40	41-50	51-60
Populationsgröße	50	50	50	50	50	50
Unverändert übernommener Anteil	10%	20%	40%	10%	20%	40%
Einsatz von Building Blocks	ja	ja	ja	nein	nein	nein
Anteil der für Mutation in Frage kommenden Individuen	30%	30%	30%	30%	30%	30%
Mutationsrate Routing	10%	10%	10%	10%	10%	10%
Mutationsrate W und L	3%	3%	3%	3%	3%	3%
Mutationsrate Terminalverbindungen	3%	3%	3%	3%	3%	3%
Mutationsrate Building Blocks	2%	2%	2%	2%	2%	2%
Mutationsrate Spiegeln	2%	2%	2%	2%	2%	2%
Mutationsrate Rotieren	2%	2%	2%	2%	2%	2%
Anteil der für Crossover in Frage kommenden Individuen	100%	100%	100%	100%	100%	100%
Crossover Rate	30%	30%	30%	30%	30%	30%

Tabelle 2.2: Experimentreihe: Parameter des GA.

Die beim Crossover ausgetauschten Bereiche dürfen maximal 2x2 Building Blocks (Experimente mit Einsatz von Building Blocks) bzw. 4x4 Transistorzellen (Experimente ohne Building Blocks) betragen.
Es werden 6 x 10 Evolutionen durchlaufen, und zwar für einen unverändert übernommenen Anteil von 10, 20 und 40% jeweils 10 Experimente mit und 10 Experimente ohne Building Blocks.

2.5.2 Ergebnisse

Alle durchgeführten Experimente bringen nur mäßige Ergebnisse hervor, wie man in Abb. 2.5 erkennen kann. Hier liegt die durchschnittliche Abweichung des besten Individuums von der Zielfunktion in der Größenordnung 1 V. Dennoch wird ein deutlich besseres Abschneiden der Evolutionsexperimente *mit* Building Blocks gegenüber jenen *ohne* deutlich.

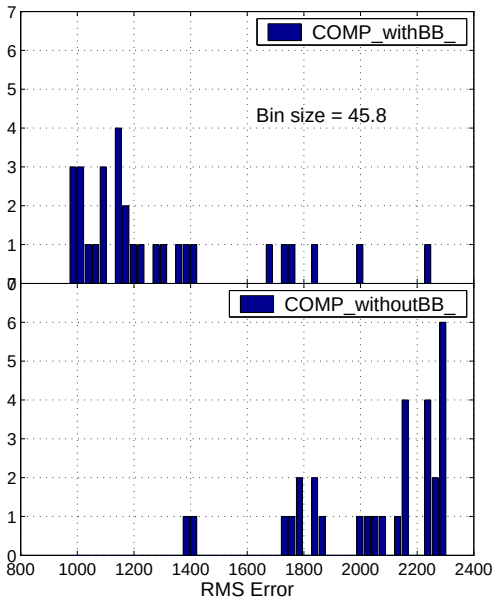

Abbildung 2.5: Histogramm der durchschnittlichen Abweichung von der Zielfunktion (in mV) aller 30 Evolutionsdurchläufe mit (oben) und ohne (unten) Building Blocks.

Die Abbildungen 2.6 und 2.7 zeigen die entwickelten Individuen, die am besten abgeschnitten haben. Sie werden mit einer Eingangssignalsequenz getestet, die sich von der während der Evolution eingesetzten unterscheidet. Es wird an derer statt für V_{in1} und V_{in2} der ganze Bereich von 0 bis 5 V durchgefahren. Während außerdem innerhalb der Evolution die Signale an Eingang 2 zufällsmäßig erzeugt waren, wird mit der breiteren Testsequenz jeweils ein Test mit Vorwärts-, Rückwärts- und Zufallsanordnung durchgeführt. In Abb. 2.6 und 2.7 sind jedoch nur die Experimente mit Vorwärtsanordnung zu sehen. Wie ein guter Komparator auf dem FPTA aussehen könnte, zeigt Abb. 2.8. Die dort dargestellte Schaltung wird auf dem Chip realisiert und liefert, ebenfalls mit der Testsequenz V_{in1} und V_{in2} von 0 bis 5 V, Ergebnisse, die dem Idealverhalten eines Komparators sehr nahe kommen (vgl. Abb. 2.9). Die Werte für W und L wurden für diese Schaltung nicht optimiert, es ist also ein noch besseres Ergebnis bei entsprechender Anpassung zu erwarten.

In Abb. 2.10 und 2.11 ist zu erkennen, daß die unterschiedlichen Werte für den unverändert zu übernehmenden Anteil keinen merklichen Einfluß auf das Ergebnis der Evolution haben.

In den Abb. 2.12 und 2.13 sind die besten evolutionierten Schaltungen aller Durchläufe einmal mit und einmal ohne Einsatz von Building Blocks zu sehen. Hier tritt der Vorteil der besseren Übersichtlichkeit im Falle der Building Blocks sehr deutlich zutage.

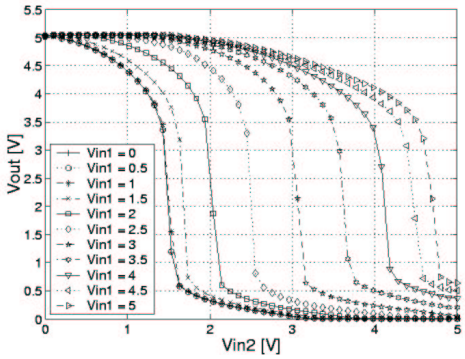

Abbildung 2.6: Testergebnis des besten mit Building Block Einsatz entwickelten Individuums, V_{in1} und V_{in2} von 0 bis 5 V.

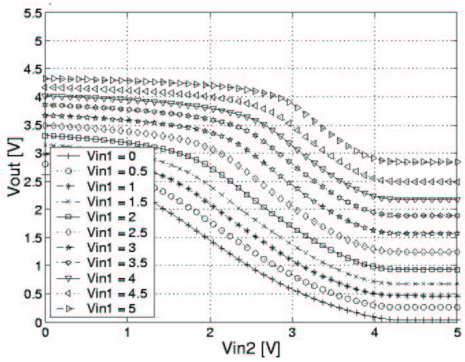

Abbildung 2.7: Testergebnis des besten ohne Building Block Einsatz entwickelten Individuums, V_{in1} und V_{in2} von 0 bis 5 V.

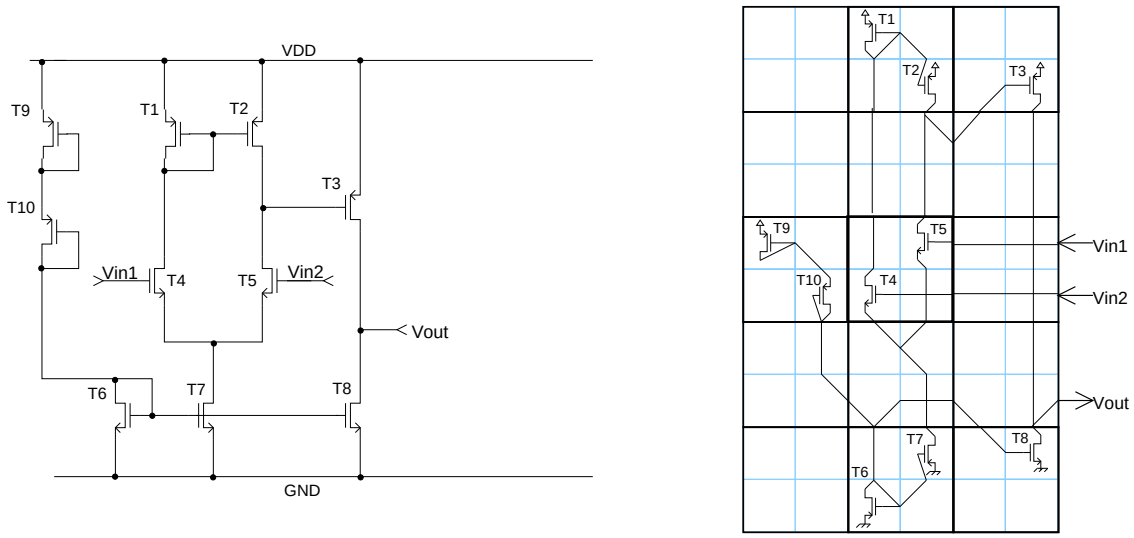

Abbildung 2.8: Links: Komparatorschaltung Schematic, aus [Lind 01]. Rechts: Auf dem FPTA realisierte Komparatorschaltung, dargestellt mit dem xfig-"Baukasten".

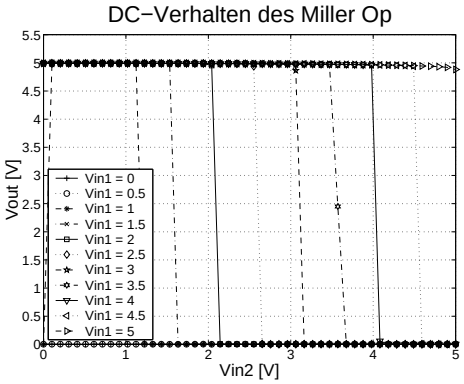

Abbildung 2.9: Testergebnis des in Abb. 2.8 gezeigten Komparators, V_{in1} und V_{in2} von 0 bis 5 V.

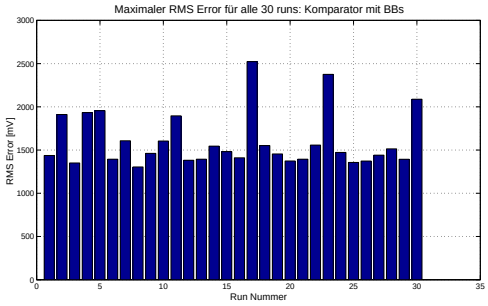

Abbildung 2.10: RMS-Error der besten Individuen aller 30 Experimente mit Building Blocks. Gezeigt ist die jeweils am schlechtesten abschneidende Evaluationsmethode.

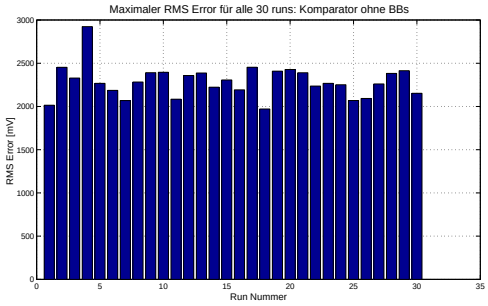

Abbildung 2.11: RMS-Error der besten Individuen aller 30 Experimente ohne Building Blocks. Gezeigt ist die jeweils am schlechtesten abschneidende Evaluationsmethode.

Abbildung 2.12: Beste evolutionierte Schaltung mit Einsatz von Building Blocks.

Abbildung 2.13: Beste evolutionierte Schaltung ohne Einsatz von Building Blocks.

Kapitel 3

Zusammenfassung

3.1 Erfolge

Die Nutzung fester Bausteine in der evolutionären Entwurfstrategie wurde in einer Weise erarbeitet, die ein sinnvolles Fortsetzen des angegangenen Weges möglich macht. In diesem konkreten Falle konnte durch den Einsatz von Building Blocks eine Verbesserung des Evolutionsergebnisses erreicht werden. Die Hoffnung scheint berechtigt, daß für weitere Schaltungen wie beispielsweise komplexere Operationsverstärker ein Mittel gefunden wurde, die Entwicklung zu beschleunigen, das Ergebnis zu verbessern und seine Verständlichkeit zu erhöhen. Allerdings müssen dazu die Versuchsparameter weiterhin optimiert werden, um für solche Schaltungen ein eventuell verwendbares Ergebnis zu erhalten. Keinen Erfolg brachte die Suche nach einer optimalen Einstellung des unverändert zu übernehmenden Anteils, da für die drei gewählten Werte kein Unterschied im Ergebnis zu erkennen war.

3.2 Ausblick

Folgende Aufgaben und Verbesserungen sind denkbar bzw. empfehlenswert:
Die Verwaltung der Building Block Bibliothek von der Benutzeroberfläche aus sollte verbessert werden. Konkret müßte vor allem der Inhalt der Bibliothek gezielt abrufbar und die einzelnen Elemente mit Namen belegbar sein. Weiterhin wären ein gezieltes Löschen einzelner Building Blocks und die Auswahl aus mehreren Dateien für den Wiederaufbau einer Bibliothek sehr hilfreich.

Die Spiegel- und Rotationsoperatoren sollten in der Benutzeroberfläche so ansprechbar sein (am besten per Dialog oder Mausklick), daß beliebige Blocks des angezeigten Genoms transformiert werden können.

In der Benutzeroberfläche sollten diverse Parameter ersichtlich sein, zum Beispiel die Mutations- und Crossover Raten des GA, vor allem aber der eventuelle Einsatz von Building Blocks.

Um eine optimale Nutzung der Building Block Strategie zu gewährleisten, sollten verschiedene Anordnungsmuster mit angepaßten Bausteinen getestet werden. Der bisherige Ansatz hat sich aus den Entwürfen einiger sehr weniger Schaltungen auf Papier entwickelt und ist keinesfalls als endgültig oder ideal anzusehen.

Anhang A

Erstellte Software

A.1 Liste der Software

Beschreibung	C++ Programmcode
Inhalt	DarkGAQT in modifizierter Version (inkl. neue Klassen *BuildBlock* und *BB_Library*)
Verzeichnisname	darkgaqt/
Ort	CVS: /project/evolution/
Beschreibung	Konfigurationsdatei
Inhalt	Größe und Positionierung der Building Blocks
Dateiname	bb_config.cfg
Ort	CVS: /project/evolution/darkgaqt/BB_Configuration/
Beschreibung	xfig-Baukasten
Inhalt	Vorlage zum Erstellen neuer Schaltungen in xfig, xfig-Schaltungen
Dateinamen	vorlage.fig, folded_cascode.fig, vergleicher.fig
Ort	CVS: /project/evolution/darkgaqt/xfig/
Beschreibung	Ergebnisse des Experimentes
Inhalt	Evolutionsergebnisse (beste Generation, Fitneßverlauf, Parameter)
Verzeichnisname	EXP_DATA/
Ort	bruederl@evolver5.kip: /project/evolution/darkgaqt/

Tabelle A.1: Erstellte bzw. modifizierte Software. Der CVS-Server ist *stan.kip.uni-heidelberg.de* mit dem Stammverzeichnis */home/cvsuser/cvsroot*.

Literaturverzeichnis

[Lang 02] J. Langeheine, K. Meier, J. Schemmel *Intrinsic Evolution of Quasi DC Solutions for Transistor Level Analog Electronic Circuits Using a CMOS FPTA Chip* Proceedings of the Fourth NASA/DOD Workshop on Evolvable Hardware, Long Beach, VA, USA July 15-18, IEEE Computer Society Press, pp. 75-84.

[Papli 01] Andrew P. Paplinski, *"Lecture Notes: CSE3142 Integrated Circuit Design"*, Monash University, Australia 2001.

[Lind 01] Volker Lindenstruth, *"Lecture Notes: Chip Design"*, Universität Heidelberg, 2001.

[LakSan 94] K.R.Laker, W.M.C.Sansen, *"Design of Analog Integrated Circuits and Systems"*, McGraw-Hill 1994.