Projektpraktikum Informatik Hardware Evolution

Praktikumbericht und Anleitung für das Tracking System (siehe auch Powerpoint Folien)

Stefan Zimmer
Betreuer: Martin Trefzer

Thema: Entwurf und Implementierung von Methoden zur Analyse der Hardware Evolution analoger elektronischer Schaltungen.

Abstract: Tracking System für die Hardware Evolution (Lade- und Speichermöglichkeit von Genomen; Automatisches erstellen eines Genom Repositories mit Genomen bestimmter Fitness; Aufzeichnungsmöglichkeit eines Evolutionsprozesses inklusive eines Tools zur Visualisierung und grafischen Analyse)

Inhalt:

1 Idee hinter der Hardware Evolution

Ziel der Hardware Evolution ist es eine elektrische Schaltung automatisch durch einen genetischen Algorithmus generieren zu lassen. Dieser Algorithmus bekommt als Eingabe eine Tabelle, die die gewünschte Ausgangsspannung bzw. Ausgangsspannungen bei gegebener Eingangsspannung bzw. Eingangsspannungen beschreibt.

Vorbild für solch einen Algorithmus ist die natürliche Evolution. Auch sie versucht die für die jeweiligen Umweltbedingungen am besten angepasstesten Individuen hervorzubringen. Wie bei ihr gibt es in der Hardware Evolution einzelne Individuen (=Genome). Jedes Genom entspricht einer bestimmten elektrischen Schaltung. Um nun die für das Problem beste Schaltung zu finden startet man mit meist zufällig erzeugten Schaltungen. Der Generation Null. Nun erzeugt der Algorithmus aus dieser Generation die nächste Generation durch Crossovern (das heißt Mutter und Vater erzeugen ein Kind) oder Selektion (Auswahl aufgrund bestimmter guter Schaltungscharakteristik,

die evaluiert werden muss). Zusätzlich können die Genome unabhängig davon in jedem Evolutionsschritt mutiert (das heißt willkürliches Verändern eines Genoms) werden. Die Aufgabe des Crossovers sollte es sein möglichst gute Schaltungsteile zu eine neuen noch besseren Schaltung zu kombinieren. Die Selektion sorgt dafür, dass der Algorithmus gegen die beste Schaltung konvergiert indem es geschickt für die nächste Generation auswählt und die Mutation bringt neue "Ideen" in den Algorithmus.

2 Experimentelle Umsetzung

Hauptaufgabe des Experiments ist es eine elektrische Schaltung aufzubauen und auf ihre Performance bezüglich der gewünschten Schaltungscharakteristik zu evaluieren. Diese Daten werden durch den Algorithmus ausgewertet und als Grundlage für die nachfolgende Generation genommen. Der Algorithmus und die Steuerung des Experimentes geschieht über eine in C++ geschriebene, stark modulare Software auf einem Messrechner. An diesen Meßrechner ist ein FPTA (FieldProgrammableTransitorArray) angeschlossen, der Transistorschaltungen direkt in Hardware realisiert. Der FPTA besteht aus 16 * 16 Zellen. Jede dieser Zellen beinhaltet mehrere Transistoren, die durch Software zu einem virtuellen Transistor mit programmierbarem Gateabstand und Gateweite zusammengeschaltet werden können. Zusätzlich können Gate, Drain und Source beliebig mit den Nachbarzellen und untereinander verbunden werden.
Der FPTA kann durch einen Simulator (Spice) auf dem Computer ersetzt werden. Hierdurch können, wenn der entsprechende Code für den Algorithmus geschrieben ist, auch Schaltungen, die über die Möglichkeiten des FPTAs hinausgehen (Kondensatoren, Spulen, Widerstände), evaluiert und für eine Evolution verwendet werden.

3 Die Softwareumgebung

Ein typischer Evolutionsrun sieht folgendermaßen aus:

1) In hannee (GUI) wird die Klasse esimpleGA geladen und die gewünschten Parameter für den Evolutionsprozess eingegeben.
2) Nach dem Starten des Evolutionsprozesses in hannee wird durch esimpleGA ein Prototyp einer Generation erstellt.
3) Im Dritten Schritt wird die initialize Funktion des ausgewählten GAs aufgerufen. Dieser initialisiert die Population und deren Genome indem den Population und den Genome Initializer aufruft.
4) Nach der Initialisierung, die nullte Generation ist jetzt fertig, startet hannee die exec() von esimpleGA()
5) diese ruft wiederum step() des gewählten GAs auf. Nun werden die Mutations-, Crossover sowie Analyse-Operatoren der genome verwendet.

Wie man aus Illustration 1 sieht besteht die Software aus mehreren Klassen. Diese werden zur Laufzeit wie oben erwähnt durch ein grafisches user interface hannee konfiguriert. Neben einfachen Parametern werden hier aber auch ganze Funktionen mittels Pointer eingehängt. So werden beispielsweise die verschiedenen Operatoren (Crossover, Mutatoren, Analysatoren) und Algorithmen für den Evolutionsprozess (SimpleGA, NonDomGA, SimpleTrackGA, NonDomTrackGA, CreateRepositoryGA) bestimmt. Die Evaluatoren bekommen die Fitness des jeweiligen Genoms von den Analysatoren. Die Analysatoren bauen das jeweilige Genom (= elektrische Schaltung) auf und messen es. Zwei Optionen sind möglich: die erste simuliert die Schaltung in Spice, die zweite programmiert den FPTA und regelt, bzw. misst die entsprechenden Ein- und Ausgänge.

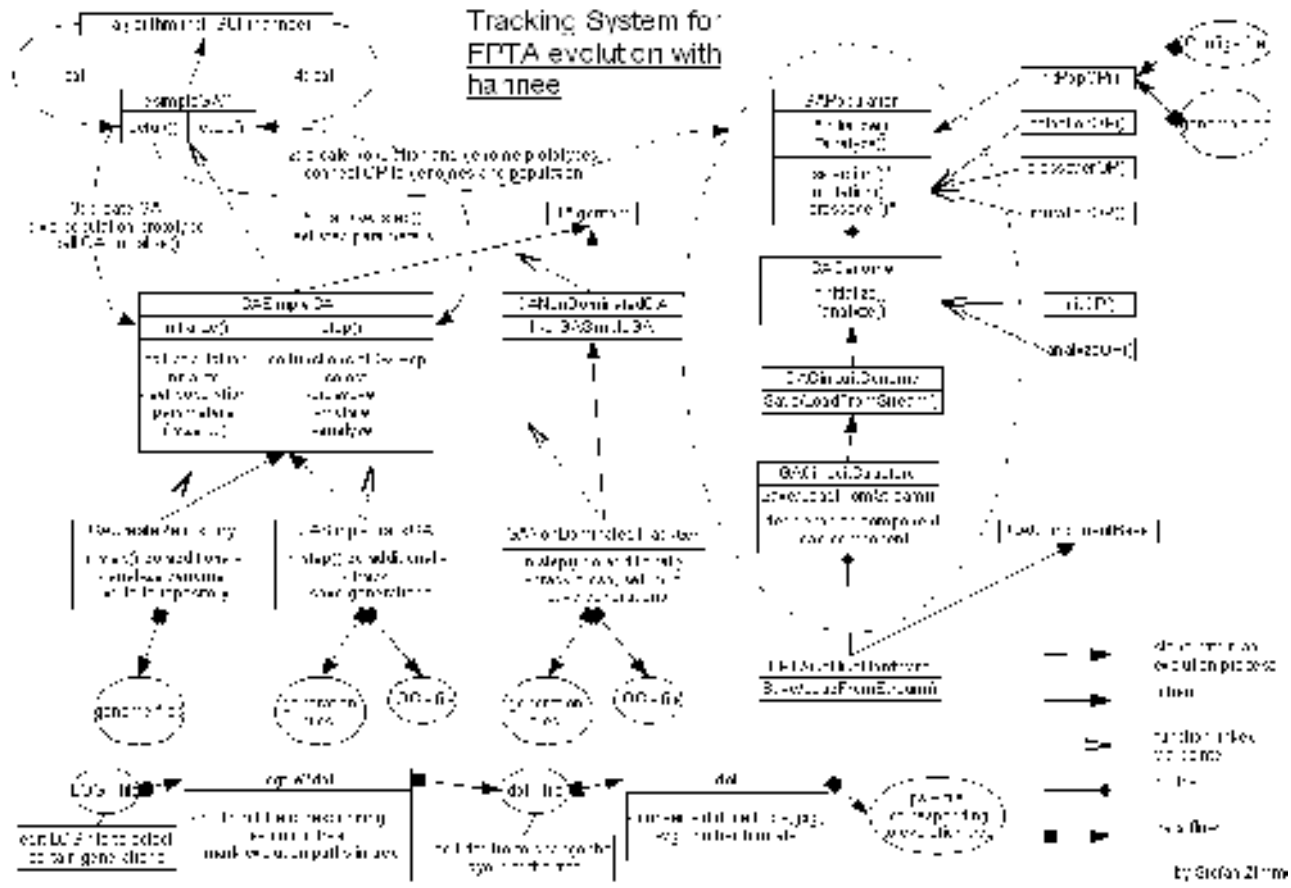


Illustration 1: Der Aufbau der Software

4 Neuerungen in der Software

Nachdem Erfolge mit dieser Softwareumgebung erzielt wurden, war es wünschenswert die so gewonnenen Ergebnisse genauer zu verstehen um dadurch weitere Verbesserungen entwickeln zu können.

Im Rahmen diese Praktikums wurde zuerst eine Speicher und Laderoutine für die einzelnen Genome entwickelt, sowie eine Klasse zum Erstellen des Repository programmiert. Aus diesem Repository können verschiedene Genome ausgewählt werden und als definierte Startgeneration für den Evolutionsprozess geladen werden. Somit ist es möglich verschiedene Algorithmen und Mutations-, Crossover- oder Selektionsoperatoren auf ihre Performance hin zu vergleichen. Da das Speicherformat der Genome ein relativ einfaches Textformat ist, kann die Evolution nun auch mit selbst erstellten Genomen gestartet werden.

Um die bisherigen Evolutionsstrategien besser zu verstehen, sowie um auf Ideen für neue Strategien zu kommen, ist ein Stammbaum der die Evolution beschreibt nützlich. Durch eine Backtrackingmögichkeit kann man zudem den Evolutionsweg eines Genoms betrachten. Beides wurde durch entsprechende Programme und Klassen implementiert.

Im folgenden ein kurzen eher anwendungsbezogenen Überblick über die Neuerungen. Für weiter Dokumentation sei auf den Quellcode und die dort gemachten Kommentare, verwiesen.

4.1 Das Repository

Ein Repository ist eine Menge verschiedener Genome. Jedes Genom wird durch ein einzelnes Zip-File repräsentiert. In jedem File wird eine FPTA Konfiguration gespeichert. Es werden für jede Zelle des FPTAs der Typ (zur Zeit ist nur eine FPTAcell möglich), die Breite und Länge des Gates, sowie die Position und die Konfiguration der Verbindungen gespeichert. Ein Repository ist weiterhin in disjunkte Teilmengen (Bins) aufgeteilt. Die Charakteristik eines jeden Bins ist ein bestimmtes Fitnessintervall. Alle Genome einer Teilmenge liegen in ihrem charakteristischen Fitnessintervall.
Um ein Repository zu erstellen benutzt man den CreateRepositoryGA. Als Parameter kann man im hannee GUI die Startfitness, die Endfitness, die Maximale Anzahl der Genome sowie die Anzahl der Unterteilungen zwischen Start und Endfittness einstellen. Der CreateRepositoryGA startet dann einen Evolutionsprozess und versucht die einzelnen Bins mit echt verschiedenen Genomen zu füllen.

In den erzeugten Genome Files werden die Konfigurationen der FPTA Zellen abgespeichert. Das heißt Gatebreite, Gatelänge Typ der Zelle, Verbindungen der Ports, Positon im FPTA der Zelle sowie der Anschluss von Drain, Gate und Source.

Um einen Evolutionsprozess mit einer definierten Anfangsgeneration aus dem Repository zu initialisieren wählt man alle Operatoren und Parameter wie gewünscht aus. Als Besonderheit muss man nun aber den initPopFromRepository als Initializer auswählen. Dieser bekommt als Parameter ein Konfigurationsdatei. Jede Zeile entspricht einem Fitnessbin. Die durch Leerzeichen getrennten Zahlen entsprechen den Ids der zu ladenden Genome aus diesem Fitnessbin. Zeilen die mit einem Gartenzaun ('#') beginnen sind Kommentare und werden genauso wie in den Genomen des Repositories ignoriert beim Laden.

4.2 Das Tracking- und Visualisierungssystem

Um einen Evolutionsprozess zu visualisieren wählt man den zu seinem gewünschten genetischen Algorithmus entsprechenden TrackGA aus. Dieser schreibt an den in hannee konfigurierten Orten die Individuen jeder Generation gepackt in für jede Generation einem Generationsfile. Zusätzlich erzeugt es ein logfile indem protokolliert wird wann welcher Operator angewendet wurde. Dieses Logfile kann mittels logfile2dot geparst und analysiert werden. Siehe ./logfile2dot - - help. So kann man die Ahnen eines Genoms ausfindig machen, sowie deren prozentualen Beitrag zum Erbgut diese Genoms bestimmen.

Ein Logfile hat für jede Generation eine Sektion. Jede Sektion besteht aus vier Zeilen und mehreren Spalten. Jeder Spaltenvektor, die letzte Zeile gehört nicht dazu, repräsentiert die Geschichte dieses Genoms in diesem Evolutionsschritt. Aus der “sel:” Zeile findet man heraus aus welcher Spalte es aus der vorherigen Generation ausgewählt wurde. Die “cro:” Zeile gibt an mit wem es gecrosst wurde. -1 bedeutet nicht gecrosst. Die “mut:” Zeile gibt die Stärke der Mutation an. Die “elt:” Zeile gibt die durch Elitismus hervorgerufene Ersetzung wieder. Die Zeile ist als Hintereinanderreihung 0 bis n liegender Vektoren der Dimension zwei zu verstehen. Der erste Eintrag gibt das Individum (die Spalte) in der vorherigen Generation an, das in dieser Generation das Individuum mit der Nummer (Spalte), die im zweiten Eintrag des Vektros steht ersetzt an. Meist ersetzt man so das jeweils schlechteste Individuum einer Generation durch das beste aus der vorherigen.

Das logfile2dot Programm hat einen letzten Parameter die sogenannte Threshold. Da beim

angewendeten rekursiven Zurückschreiten die Anzahl der möglichen Wege exponentiell mit der Anzahl der zurückgetrackten Generationen steigt und somit die Rechenzeit ins Unendliche steigt wurde dieser nötig. Beim Zurückverfolgen und dem bestimmen der Ahnengewichte, wird der unter einem Genom liegende Weg nur dann aktualisiert, wenn er seit dem letzten Aktualisieren um diesen Faktor gewachsen ist. So ist sichergestellt, dass das Problem in endlicher Zeit gelöst werden kann, gleichzeitig wird aber auch kein Weg vergessen. Der Preis ist, dass das Ergebnis nur auf die Größenordnung des Faktors genau ist.

Wenn nur der Baum gezeichnet werden soll, wird jeder Weg nur einmal gegangen. Hier werden harte Sperren in jedem Genom gesetzt wenn es bereits besucht wurde.

Dieses durch logfile2dot erstellte dotfile kann mittels dot (www.graphviz.org) in einen Grafen um gewandelt werden. Hierzu sind aber für 250Generationen mit jeweils 100 Individuen mindesten 1GB Hauptspeicher und ca. 4 Stunden Rechenzeit auf einem Pentium4 nötig. Das anschauen des 65 MB großen ps – Files ist auch mit normalen Viewern nicht möglich. Als Workaround hat sich die Konvertierung in PNG mit geringer Auflösung oder nur Ausschnittsweise mittels convert von www.ImageMagick.org erwiesen. In Gimp lassen sich die ps-files auch direkt mit geringerer Auflösung importieren und durch Farbtransformationen die geeigneten Informationen extrahieren.

In Dia 9 ist beispielsweise ein Ausschnitt eines Evolutionsbaum für ein AND über 250 Generationen mit dem simpleGA mit allen verfügbaren Informationen dargestellt, um die Möglichkeiten des Visualisierungssystems zu demonstrieren. Jede Spalte steht für eine Generation, jedes Genom(= Ellipse oder Fünfeck) für ein Individuum dieser Generation. Die durchgezogenen Linien stehen für Selektion oder bei Crossover für den dominanten Teil, die gestrichelten für den nicht dominanten Partner, die Anzahl der Ränder um jedes Genom gibt die Stärke der Mutation an, ein elliptisches Genom bedeudet keine Mutation, die grünen Linien geben die duch Elitismus weitergekommenen und ersetzte Genome an, die roten Linien geben die Ahnenpfade eines aus einer weiter hinten liegenden Generation gebacktrackten Genoms an, die Zahlen an den roten Linien geben das Gewicht des jeweiligen Ahnenpfades an. Zahl 0.034 bedeutet: Wenn ich diesen Pfad nicht hätte würden 3.4 Prozent des Erbgutes des gebacktrackten Genoms fehlen. Die Summe der Gewichte der einlaufenden Ahnenpfade muss nicht der Summe der auslaufenden Ahnenpfade entsprechen, da duch Mutation neues Erbgut entstehen kann.

Eine mögliche Verbesserungen: Einen eigenen Viewer schreiben, der on the fly backtrackt und plottet. Er sollte sich den Einsprungspunkt merken und von dort ab einen wie durch logfile2dot erstellten doppelt verketteten Baum ablaufen. Nachteil keine Übersicht wie sie durch dot geschaffen wird. Beobachtungen wie in 5.3 wären so nicht möglich. Auch die Ausgabe als eps ist dann nur mit enormen Aufwand zu bewerkstelligen. Zudem ist die Einarbeitung in die kde-Library nötig was mit großem Zeitaufwand verbunden ist.

5 Erste Messungen mit den neuen Systemerweiterungen

5.1 Rauschen des FPTAs

Um das Rauschen des FPTAs zu untersuchen haben wir eine große Generation mit immer dem selben Individuum initialisiert. Die Abweichung vom ursprünglichen Wert bei der anschließenden Messung sind für ein AND schlechter Fitness in Dia 1 und Dia 2 angegeben. Das AND wurde zweimal mit vertauschten Eingängen gemessen. Jeder Operator (TGA bzw. BGA) hatte sein eigenes Repository. Zwischen rms 0.5 V bis 2.8 V rms. Das heißt die gemittelte quadratische Abweichung im Eingangsspannungs - Ausgangsspannungsplot des evolutionierten ANDs und eines idealen ANDs (das seinen Ausgang auf 5V schaltet, wenn beide Eingänge über 2.5 Volt liegen) lag zwischen 0.5V rms bis 2.8V rms. In Illustration 2 ist beispielsweise ein Plot für ein evolutioniertes

AND dargestellt.

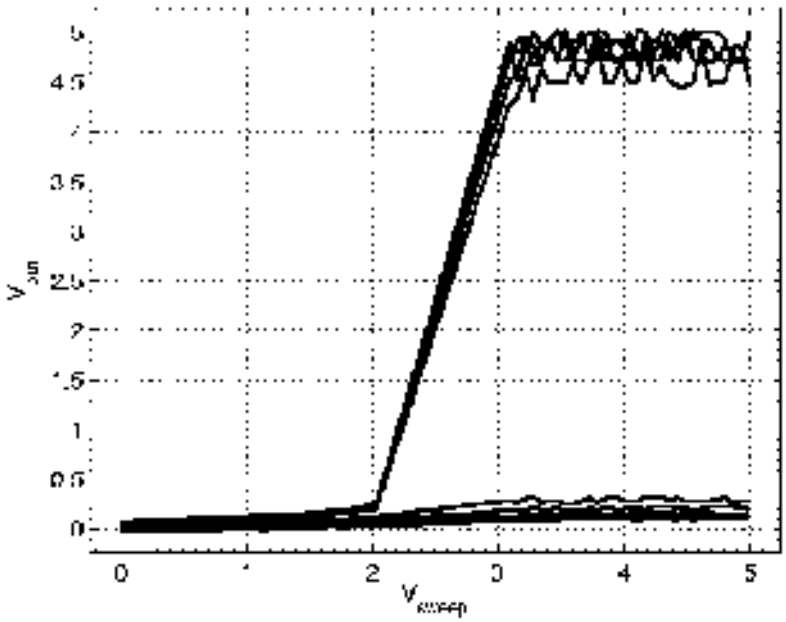
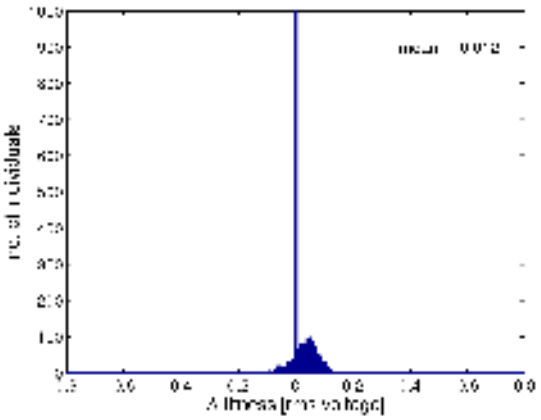


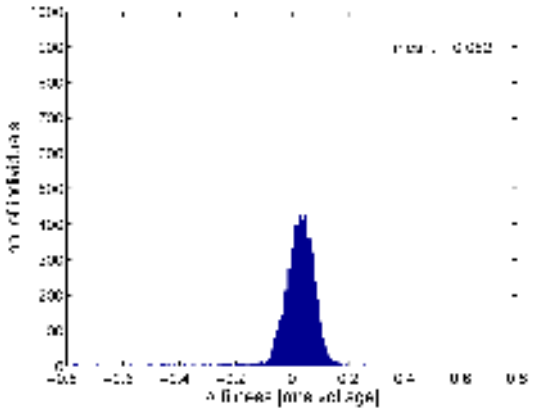
Illustration 2: Schaltungscharakteristik eines evolutionierten ANDs. Auf der y-Achse ist die Ausgangsspannung aufgetragen. Die Spannung am ersten Eingang des ANDs auf der x-Achse. Der zweite Eingang ergibt sich aus den verschiedenen Kurven. Jede Kurve hat eine verschiedene Spannung am zweiten Eingang. Die Daten wurden direkt in Hardware gemessen.

Es fällt auf das die TGA Operatoren wesentlich stabilere Schaltungen liefern als die BGA Operatoren. Dies liegt an der Eigenschaft der TGA Operatoren immer geschlossene Schaltkreise zu produzieren. Wohingegen bei den BGA Operatoren auch offene Enden akzeptiert werden.

Geplottet ist die Differenz zwischen dem geladenen und dem gemessenen rms - Fitnesswert. Der Fitnesswert lag bei ungefähr 1.8V rms. Da auch der geladene Fitnesswert einmal gemessen wurde und so einem Messfehler unterliegt, könnte ein Stück weit die gewisse Verschiebung des Maximums weg vom Nullpunkt erklären (Besonders beim BGA). Es bleibt aber immer noch eine Asymmetrie zu erklären und warum der Peak bei den TGA Operatoren genau bei Null liegt.



Dia1: Rauschen des FPTAs für ein mittels TGA Operatoren evolutionierten ANDs mittlerer Fitness in Hardware



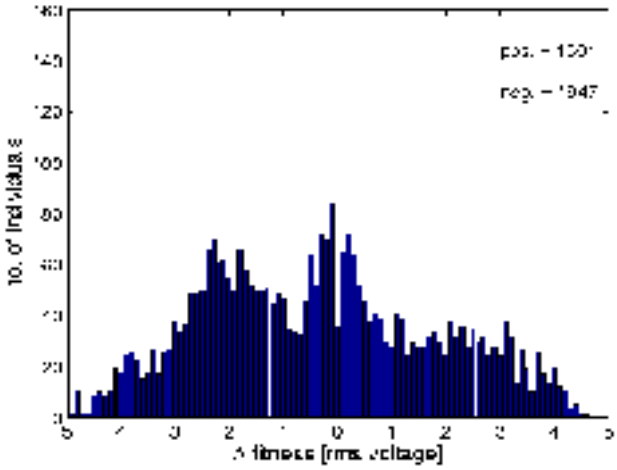
Dia2: Rauschen des FPTAs für ein mittels BGA Operatoren evolutionierten ANDs mittlerer Fitness in Hardware

5.2 Vergleich von TGA und BGA Operatoren

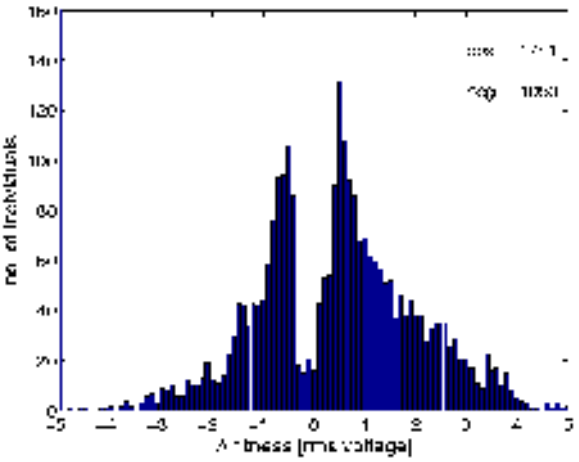
Eine Generation wurde mit 10 Individuen aus jedem ihrer 100 Fitnessbins initialisiert. Nach einem Evolutionsschritt in dem jeweils nur Crossover oder Mutation zugelassen war wurde der Fitnessunterschied zum initialisierten Genom gemessen. Das Kriterium war wieder ein AND. Um das Rauschen des FPTAs etwas auszublenden, wurden nur solche Differenzen beachtet, die größer als der Mittelwert des Rauchens war.

In Dia 3 bis 6 fällt auf das kaum ein Genom die alte Fitness beibehält. Ungefähr ein Drittel wird besser und zwei Drittel schlechter. In Kombination mit dem Selektionsoperator ergibt dies eine Steigerung der Fitness von Generation zu Generation.

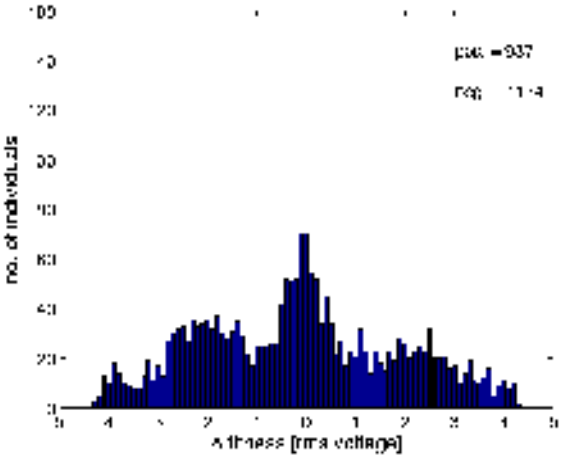
Weitere Messungen, die die Qualität der Operatoren in verschiedenen Fitnessintervallen untersuchen werden zur Zeit gerade durchgeführt.



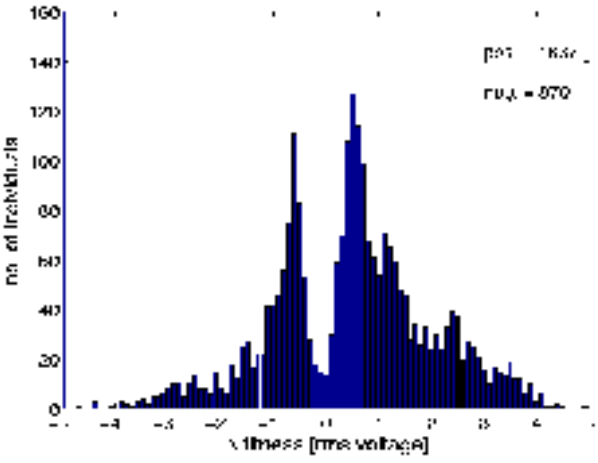
Dia 3: Performance des TGA crossover Operators in einem Evolutionsschritt in Harware



Dia 5: Performance des BGA crossover Operators in einem Evolutionsschritt in Hardware



Dia 4: Performance des TGA mutations Operators in einem Evolutionsschritt in Hardware

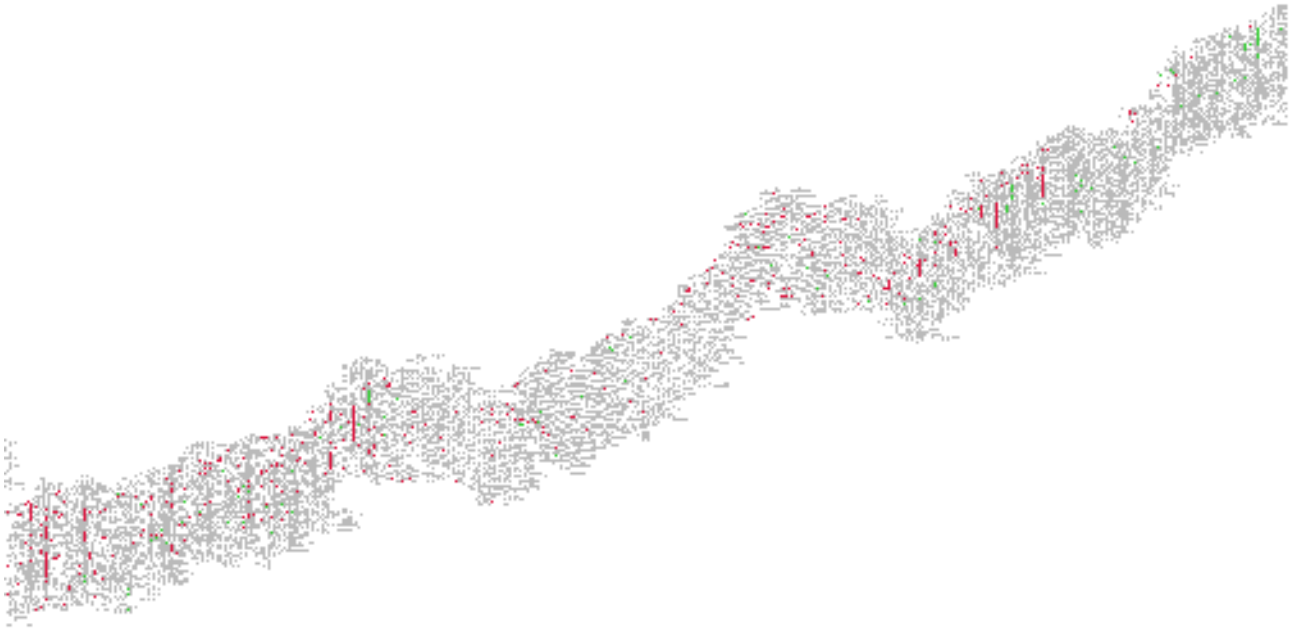


Dia 6: Performance des BGA mutations Operators in einem Evolutionsschritt in Hardware

5.3 Vergleich von Simple und NonDominated GA

In Dia 8 und 9 wurde aus einer zufälligen Startkonfiguration 250 Evolutionsschritte mitgetrackt und ein repräsentatives Genom gebacktracked.

Es fällt auf, dass beim NonDominatedGA kaum Verzweigungen auftreten. Der Beste wurde also oft selektiert und kam nur nach Mutation und ohne Crossover weiter. Des Weiteren fällt auf, dass das Schaubild des simpleGA nach oben steigt, wohingegen das des nondomGA gerade ist.



Dia 7: 250 Generationen einer SimpleGA AND Evolution in Hardware

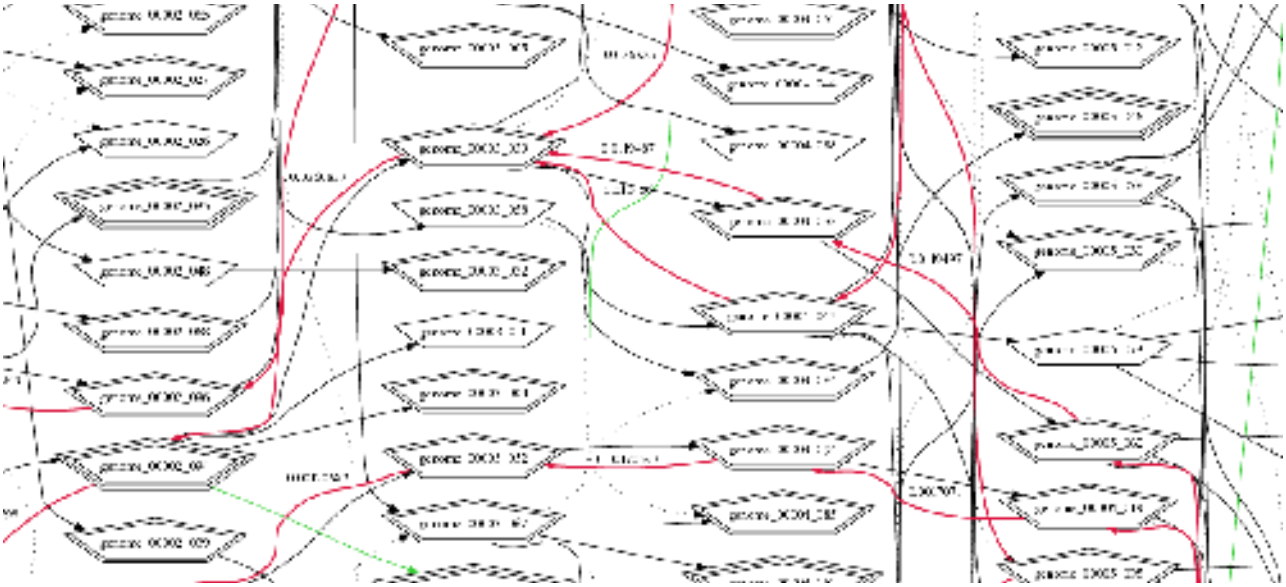
Dia 8: 250 Generationen einer NonDominated AND Evolution in Hardware

5.4 Der Einfluss der Ahnen auf ein Genom

In Dia 9 ist ein Ausschnitt eines mittels des neuen logfile2dot erzeugten Baumes zu sehen. Er stammt von einem simpleGA mit 100 Genarationen. Ziel der evolutionierten Schaltung war ein AND. Rot sind die Backtracklinien eines Genoms der 250ten Generation gezeichnet. Grün ist der Elitismus. Durchgezogen schwarz die Selektion, bzw. der dominante Teil beim Crossovern, gestrichelt der nicht dominante Crossoverpartner. Die Ränder um die Individuen gegen die Anzahl stärke der Mutation an. Crossover wurde 12.5% zu 87.5% auf die Partner aufgeteilt. Die Mutation sehr moderat mit 0.001% pro Rahmen angenommen.

Man sieht bereits nach einer eher geringen Anzahl von 250 Generationen, dass die Ahnen nur wenig zu ihren Kindern beitragen. Der wichtigste Impulsgeber der die Evolution voranbringt ist demnach die Mutation. Bei längeren Evolutionen über beispielsweise 20 000 Generationen ergab sich für die Initialisierungsgeneration nahezu ein Gewicht von null.

Erstaunlich ist auch, dass es auch richtige Knotenpunkte gibt in denen viele Wege zusammenlaufen.



Dia 9: Ausschnitt aus einem SimpleGA für ein AND. Rot aus der 250ten Generation gebacktracktes Genom.

6 Persönliches Fazit

Es war schwierig den Einstieg zu finden, da es wenig Dokumentation zu diesem Projekt gab. Am Allein aus Kommentaren im Code wurde der große Zusammenhang nicht richtig klar. Zum Glück hat mir mein Betreuer vieles geduldig erklärt, wodruch ich auch Neues über C++ Klassen, dynamic casts und co gelernt habe. Das Einrichten der Software, Libraries ... war langwierig. Das Beta - Stadium der Software sorgte auch für die ein oder andere Überaschung. Was das Praktikum auf jeden Fall gebracht hat ist einiges an Routine im Umgang mit (Linux) Softwaretool, Python (der Helfer bei korrupten log files), das remote Arbeiten auf Rechnern und natürlich C++ Programmiererfahrung. Das wichtigste aber war ein Gefühl dafür bekommen zu haben wie in einem großen Softwareprojekt gearbeitet wird und welche Probleme dabei eine Rolle spielen.

Anhang: Neue und veränderte Dateien

GACreateRepository	neu
initPopFromRepository	neu
GASimpleTrackGA	neu
GANonDominatedTrackGA	neu
logfile2dot	neu
FPTACelllikeHardware	Speicher und Laderoutine neu
GACirrcuitStructure	Lade und Speicher Funktionalität angelegt
GACircuitGenome	Lade und Speicher Funktionalität angelegt
GAGenome	Genom ID implementiert
esimpleGA	neue Einträge im GUI implementiert Möglichkeit des Einhängens der neuen Tools in das System geschaffen