fhwedel
UNIVERSITY OF APPLIED SCIENCES

EDA-Praktikum

Implementation eines Platzierung-Algorithmus durch die Berechnung des Zero-Force-Targets

Von:

Janik Piepenhagen

Studiengang: Technische Informatik

Matrikelnummer: Tinf103697

Fachsemester: 6

Verwaltungssemester: 10

24.05.2023

fhwedel
UNIVERSITY OF APPLIED SCIENCES

Inhaltsverzeichnis

1. Einleitung

Diese Dokumentation ist im Rahmen des Praktikums „Rechner gestützter Entwurf digitaler Systeme“ (EDA) entstanden. Dabei wurde mittels der Kräfteplatzierung ein Algorithmus zur Platzierung von Schaltungselementen implementiert. Die Platzierung ist der nach dem Clustering ein wichtiger Schritt zur Bestimmung der Position der zerlegten Teilblöcke und legt maßgeblich die Verdrahtungskosten und den kritischen Pfad durch den Verlauf von Verbindungsleitungen fest. Gleichzeitig kann die Platzierung Auswirkungen auf die Laufzeiteffizienz des Verdrahtungsalgorithmus haben und kann somit auch ein softwaretechnischer Faktor sein.

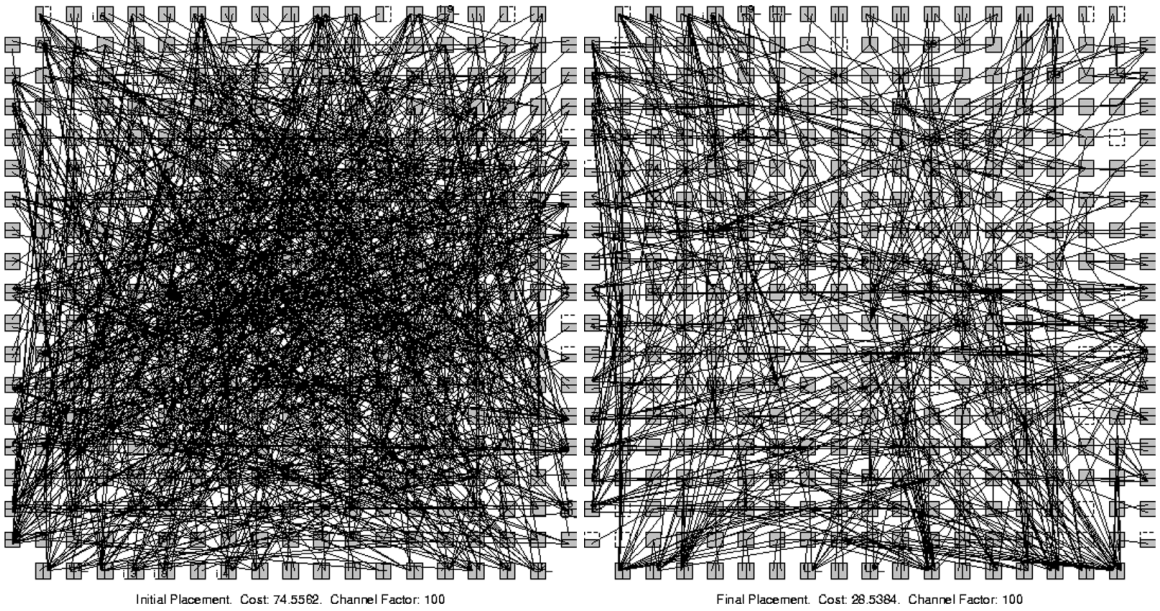

Abbildung 1.1: Gleiche Netzlisten mit unterschiedlichen Platzierungen

1.1. Strategie

Zur Platzierung wurde der Ansatz der Kräfteplatzierung gewählt. Die Kräfteplatzierung basiert auf einem physikalischen Modell, bei dem frei bewegliche Körper mit Federn verbunden sind. Je nach Masse der Körper und Steife der Feder, strebt das System einen Zustand des Kräftegleichgewichts an.

$$F = c * d \rightarrow Kraft = Federsteife * Entfernung$$

Das Modell lässt sich auf das Verdrahtungsproblem anwenden, indem Zellen als Körper und Verdrahtungskosten als Kraft interpretiert werden. Die Entfernung zweier Körper wird über den euklidischen Abstand berechnet. Die Gewichtung einer Zelle kann je nach Implementationsansatz unterschiedlich sein und kann beispielsweise über die Kostenfunktion des halben Netzumfangs (bounding box = bb) berechnet werden.

Die Kräfteplatzierung ist ein iterativer Prozess, bei dem die logischen Blöcke so lange verschoben werden, bis ein Abbruchkriterium erreicht wurde. Bei dem Verfahren wird für alle logischen Blöcke die *Zero-Force-Target*-Position (ZFT-Position) berechnet und dann versucht, diese zu verschieben. Da mit jeder Verschiebung sich die ZFT eines anderen Blockes ändern kann, wird der gesamte Prozess vielfach wiederholt und kann eine hohe Anzahl an Vertauschungen verursachen.

1.2. Zielstellung

Das voranstehende Ziel ist die Implementierung eines in der Vorlesung behandelten Algorithmus, um softwaretechnische Probleme zu behandeln, die in der Theorie schlecht vermittelbar sind. Die Implementation muss dabei nicht besser sein als bekannte Tools wie VPR. Die Implementierung muss allerdings eine legale Platzierung erzeugen, bei der die Netzlisten aus 4:1 LUT (look-up-tables) und Flip-Flops bestehen.

Für den kreativen Anteil wurde ein einfacher Algorithmus eingesetzt, um anstelle einer zufälligen Platzierung, eine gewichtsbasierte Platzierung zu verwenden. Damit sollen die initialen Kosten gesenkt, um die Laufzeit oder die Anzahl an Iterationen zu reduzieren. Dieser Ansatz legt jedoch nicht nahe, dass der eigentliche Algorithmus im Endergebnis besser sein wird.

2. Implementierungsdetails

Im folgenden Abschnitt soll näher auf die inhaltliche Implementation eingegangen werden. Dabei werden nur grundlegende Berechnungen und Algorithmen erläutert, nicht aber wie sie im Detail softwaretechnisch umgesetzt sind.

2.1. Initialisierungsphase

In der Initialisierungsphase werden die IO-Elemente und die logischen Blöcke initial platziert. Dies ist notwendig, da zur Berechnung der ZFT-Position jeder Block eine Koordinate auf der Layout-Fläche besitzen muss. Die Größe der Layout-Fläche berechnet sich aus der Anzahl der IO-Elemente unter Einbezug der IO-Rate, bzw. die Anzahl der logischen Blöcke.

Die IO-Elemente werden zufällig auf eine Pad-Position verteilt. Die Pads befinden sich am Außenrand der Layout-Fläche. Je nach Architecture können sich mehrere IO-Blöcke auf einem Pad befinden. Für *basic-logic-elements* (BLE oder CLB für *common-logic-block*) gilt diese Regel nicht.

Die Platzierung für BLE's wird initial in der Regel auch zufällig gewählt. Sowohl für die IO-Blöcke als auch für die BLE's werden alle möglichen freien Positionen in eine Liste geladen. Bei IO-Elementen mit einer höheren IO-Rate sind Positionen doppelt vertreten. In einem iterativen Schritt über die Blöcke wird zufällig ein Index bestimmt, mit dem eine Position aus der Liste geholt und dem Block zugewiesen wird. Diese Position wird dann aus der Liste entfernt.

2.1.1. Berechnung der Gewichte

Ein wichtiger Schritt zur Berechnung des ZFT ist die Gewichtung der einzelnen Blöcke. Je stärker das Gewicht eines Blockes, desto mehr zieht dieser andere Blöcke an, die sich mit diesem in einem Netz befinden. Die Gewichtung kann also ausschlaggebend für die Berechnung der ZFT bei der Platzierung sein.

Die Berechnung des Gewichts wurde in Teilen von VPR übernommen und bezieht sich auf die Anzahl an Kreuzungen in einem Netz. Diese werden über einen interpolierten Wert angegeben. Je nach Anzahl der Blöcke in einem Netz wird ein bestimmter Wert zurückgegeben. Die Werte werden dann für jedes Netz eines Blockes aufaddiert.

$w_i = \sum_{n \in N} d_n$, mit d_n als interpolierten Kostenfaktor eines Netzes n durch die Anzahl der Blöcke im Netz und N als Menge aller Netze.

```
/* By VPR: Expected crossing counts for nets with different #'s of pins.
From *
 * ICCAD 94 pp. 690 - 695 (with linear interpolation applied by me).   */
private final double[] crossCount = new double[]{   /* [0..49] */
        1.0, 1.0, 1.0, 1.0828, 1.1536, 1.2206, 1.2823, 1.3385, 1.3991,
1.4493, 1.4974, 1.5455, 1.5937, 1.6418,
        1.6899, 1.7304, 1.7709, 1.8114, 1.8519, 1.8924, 1.9288, 1.9652,
2.0015, 2.0379, 2.0743, 2.1061, 2.1379,
        2.1698, 2.2016, 2.2334, 2.2646, 2.2958, 2.3271, 2.3583, 2.3895,
2.4187, 2.4479, 2.4772, 2.5064, 2.5356,
        2.5610, 2.5864, 2.6117, 2.6371, 2.6625, 2.6887, 2.7148, 2.7410,
2.7671, 2.7933};
```

2.1.2. Gewichtsbasierte Platzierung (Kreativer Anteil)

Wie auch bei anderen iterativen Algorithmen, bspw. VPR, wird häufig eine zufällige Initialplatzierung festgelegt. Eine zufällige Platzierung kann jedoch Einfluss auf die Laufzeit des Algorithmus haben, da diese bei besonders schlechter Platzierung mehr Iterationen erfordern kann. Es kann daher Sinn ergeben, einen Ansatz zu wählen, welcher eine einfache Initialplatzierung generiert, der die Kosten reduziert.

Die Festlegung der Positionen besteht aus der Berechnung und Sortierung der Netze nach ihrem Kostenfaktor (2.1.1). Die interpolierten Werte wurden aus VPR übertragen und liegen zwischen 1.0 und 2.7 bei 1 bis 50 Blöcken pro Netz. Bei über 50 Blöcken pro Netz wird mit einem zusätzlichen Faktor und der Block-Anzahl der Faktor getrieben.

Anhand dieser Formel werden alle Netze nach Größe des Kostenfaktors sortiert, sodass an erster Stelle das Netz mit den kleinsten Kosten und an letzter das mit dem größten Kostenfaktor befindet. Nun werden alle Blöcke, die sich in dem Netz befinden, reihenweise auf der Layout-Fläche platziert. Blöcke, die bereits platziert wurden, werden bei der Platzierung nachfolgender Netze übersprungen, wenn diese sich dort ebenfalls enthalten sind. Die Platzierung nimmt auf diese Weise keinen Einfluss auf die Laufzeit, da lediglich einmal über alle Blöcke iteriert wird.

2.2. Iterationsschritte

Ein einzelner Iterationsschritt bei der ZFT-Methode besteht aus mehreren Abläufen. Grundsätzlich wird dabei über alle logischen Blöcke iteriert. Hat ein Block in einem Iterationsschritt seine Position getauscht, dann wird dieser erst wieder in der nächsten Iteration behandelt.

2.2.1. Berechnung der ZFT-Position

Zuallererst wird für einen logischen Block die ZFT-Position berechnet. Diese ergibt sich aus den Blöcken, die mit dem der logische Block verbunden ist und deren Gewichten. In Mehrpunktnetzen werden alle logischen Blöcke mit einberechnet. Dabei wird die X-Position und die Y-Position mit dem Gewicht multipliziert und mit den Blöcken aufsummiert. Diese wird dann nochmal durch die Summe der Gewichte geteilt, woraus sich die ZFT-Position für den Block ergibt.

$$x_1^0 = \frac{\sum_j (\mathrm{w_{ij}} * \mathrm{x_j})}{\sum_j (\mathrm{w_{ij}})} \quad und \quad y_1^0 = \frac{\sum_j (\mathrm{w_{ij}} * \mathrm{y_j})}{\sum_j (\mathrm{w_{ij}})}$$

2.2.2. Verschieben der Zellen

Nach der Berechnung der ZFT-Position soll der Block auch auf diese Position verschoben werden. Ist diese Position frei kann der Block einfach verschoben werden. Dies ist in der Regel aber eher selten bei kleinen Layout-Flächen, weshalb Lösungsansätze bei belegter Position implementiert wurden. Die Vertauschung kann dann wiederum Auswirkungen auf andere Blöcke haben.

Suchen einer freien Position in der Nähe

Eine Möglichkeit, die nach der Feststellung der belegten Position probiert wird, ist das Ausweichen auf eine freie Position in der Nähe der ZFT-Position. Dazu wird in einem bestimmten Umkreis um die ZFT-Position alle freien Positionen gesammelt und nach der euklidischen Entfernung sortiert. Der Radius des Umkreises kann entscheiden für den Erfolg dieser Methode sein. Allerdings hat dies den Nachteil, dass sich Positionen weit Weg vom eigentlichen Ziel einordnen können, wodurch die Anzahl der Iterationen für ein besseres Ergebnis erhöht werden müsste.

Tauschen nach Berechnung der Kosten

Je nach Größe des Umkreises, in dem eine freie Position gesucht wird, kann auch dieser Schritt zu keinem Vertauschen des Blockes führen. In diesem Fall wird berechnet, ob es sich lohnt, die Blöcke zu tauschen.

Hierbei werden die Kosten des Blockes durch die Berechnung des halben Netzumfang mal dem Kostenfaktor für alle Netze des Blockes aufsummiert.

$$Costs = \sum_{n \in N} (x_{\max} - x_{min} + 1) * crossCount_n + (y_{max} - y_{min} + 1) * crossCount_n$$

Die Kosten werden sowohl vorher als nachher für beide Blöcke berechnet. Sind die Kosten insgesamt nach dem Tausch höher als vorher, dann wird Tausch verworfen und umgekehrt. Damit soll vermieden werden, dass die Kosten des Gesamtnetzes durch den Tausch erhöht werden. Ist ein Tausch erfolgreich, dann hat dies meist starke positive Auswirkungen auf die Gesamtkosten.

3. Benchmarking

Im Benchmarking werden die generierten Platzierungen durch die ZFT-Methode von bestimmten Netzlisten gegen die idealisierten Platzierungen verglichen. Diese werden hier einfach als VPR generiert angezeigt. Die Platzierungen wurden beide in VPR eingelesen und verdrahtet. Die Netzlisten bestehen aus 4 LUT und einem Flip-Flop, sowie 6 Logikeingängen.

3.1. Ablaufbedingungen

Zur Berechnung der Kosten, des kritischen Pfades und zur Validierung wurde VPR verwendet. Das Programm wurde dabei mit ins Projekt integriert, indem mit dem Java Runtime Executor die vpr.exe aufgerufen wurde. Der Ausgabe-Stream des Programms wird dabei analysiert und in eine separate Datei geschrieben. Aufgrund der Anzahl an Netzlisten wurde die Versuchsdurchführung im Multithreading betrieben, wobei immer 6 Threads verwendet wurden. All jene Dingen können geringen Einfluss auf die Laufzeit nehmen, verändern aber weder das Verhalten noch die Platzierung oder Verdrahtung.

Die Versuche mit den idealisierten Netzlisten führen in der Regel zum selben Ergebnis, weshalb diese nur einmalig aufgenommen wurden. Die Versuche mit dem ZFT-Algorithmus wurden mehrfach ausgeführt und es wurde das beste Ergebnis ausgewählt. Die Varianz wird hier vor allem von der zufälligen Platzierung der IO-Pads getrieben. Ein Streumaß oder ein Mittelwert wurden nicht erstellt.

Die Anzahl an Iterationen beträgt bei den meisten Netzlisten 700. Eine Variation der Iterationen wurde bei einigen Netzlisten vorgenommen und wurde bis zu 1200 Iterationen erhöht. Die Größer des area swap hat in den meisten Fällen 8 betragen. In einigen Fällen wurde diese auf 12 angehoben, um ein optimaleres Ergebnis zu produzieren.

3.1.1. Hardwarekomponenten

Prozessor	Intel i7-9700k @ 3.60 GHz, 8 Kerne (kein Hyperthreading)
RAM	32 GB DDR 4, 2133 MHz

3.1.2. Softwarekomponenten

Betriebssystem	Microsoft Windows 10 Home Version: 10.0.19045 Build 19045
Entwicklungsumgebung	IntelliJ IDEA Version 2022.1.3
JDK	Version 17.0.3 Eclipse Temurin

3.2. Versuchsergebnisse

Netzliste	# Logische Blöcke	ZFT				VPR			
		Laufzeit	Kosten	Kr. Pfad	Channel Breite	Laufzeit	Kosten	Kr. Pfad	Channel Breite
alu4	1522	01:37.881	344.7	1.621e-07	19	00:58.615	190.13	1.088e-07	11
apex2	1878	04:45.581	548.4	1.954e-07	26	01:35.895	269.76	1.305e-07	12
apex4	1262	01:39.890	303.46	1.424e-07	23	00:13.719	179.32	1.124e-07	13
bigkey	1707	05:34.488	509.23	1.146e-07	19	04:36.372	185.97	1.070e-07	6
clma	8383	82:51.760	3515.29	4.347e-07	35	02:12.622	1387.05	2.532e-07	13
des	1591	23:21.286	605.1	1.879e-07	27	00:23.976	227.84	1.161e-07	8
diffeq	1497	01:25.400	365.6	1.571e-07	20	00:11.910	146.39	8.449e-08	8
dsip	1370	07:32.009	464.37	9.528e-08	20	03:44.058	169.99	7.972e-08	6
elliptic	3604	18:02.778	1142.76	2.687e-07	27	03:02.728	457.2	1.858e-07	11
ex1010	4598	16:09.696	1498.39	2.675e-07	27	05:26.518	655.4	2.436e-07	11
ex5p	1064	01:39.834	275.86	1.461e-07	23	00:13.346	162	1.095e-07	14
frisc	3556	18:09.126	1193.89	3.223e-07	29	00:31.944	515.59	1.757e-07	13
misex3	1397	01:16.819	319.52	1.515e-07	19	01:08.457	190.2	1.065e-07	12
pdc	4575	37:51.822	1695.8	3.052e-07	33	07:08.341	898.44	2.411e-07	18
s298	1931	03:15.341	460	3.952e-07	20	00:35.453	203.9	1.984e-07	8
s38417	6406	21:30.681	2111	3.01e-07	26	01:04.701	671.75	1.794e-07	8
s38584.1	6447	39:29.843	2580.51	2.680e-07	32	02:06.332	657.87	1.276e-07	9
seq	1750	03:21.307	509.3	1.843e-07	26	01:20.183	247.66	1.291e-07	12
spla	3690	31:06.102	1408.05	2.593e-07	34	01:31.851	593.96	1.656e-07	14
tseng	1047	01:03.855	226.5	1.284e-07	18	00:03.246	92.04	7.803e-08	7

-- Angaben: Laufzeit in mm:ss.ms, Kosten in bb, kritischer Pfad in Sekunden, Channel Breite als Faktor

4. Analyse

Im folgenden Abschnitt sollen die Ergebnisse der Verdrahtung näher analysiert und bewertet werden. Dabei wird auch nochmal näher auf die Auswirkungen des kreativen Anteils eingegangen, welcher sich im Benchmarking weniger gut Abzeichen lässt.

4.1. Bewertung der Platzierung

Angesichts der Versuchsergebnisse lässt sich schnell erkenne, dass die ZFT-Methode eine erheblich schlechtere Platzierung generiert. Dies wirkt sich auf die Breite der Verdrahtungskanäle, den kritischen Pfad und den Kosten aus. Mit den Kosten einhergehend benötigt auch die Laufzeit der Verdrahtung in Teilen erheblich länger, wobei bei der clma Netzliste mit fast 1,5 Stunden die längste Zeit für die Verdrahtung benötigt wurde. Vor allem größere Netzlisten wie s38584.1, s38417, pdc, clma, spla, pdc oder frisc performen besonders schlecht und bedeutend längere Laufzeiten. Der kritische Pfad ist in größeren Netzlisten auch anteilig länger als es bei kleineren Netzlisten der Fall ist.

Die Gründe für die Schlechte Performance des Algorithmus können mannigfaltig sein. Die Berechnung der ZFT-Position und die Gewichtung sind dabei eventuell weniger der ausschlaggebende Punkt. Viel eher ist es die Strategie, mit der Blöcke bewegt werden, wenn ihre ZFT-Position belegt ist.

Die Strategie, bei belegter Position eine freie Position in der Nähe zu finden, macht als ersten Schritt Sinn, wenn besonders viele Zellen frei sind. Damit kann der Block nah zu seiner ZFT-Position bewegt werden ohne Einfluss auf andere Blöcke zu nehmen. Es wurde bei diesem Schritt allerdings vermieden eine Prüfung einzubauen, ob die nächstliegende freie Position eventuell weiter von der ZFT-Position entfernt, ist als die aktuelle Position des Blocks. Die Prüfung kann theoretisch einfach verhindern, dass die Kosten erhöht, statt verringert wurden. In Versuchen hat sich allerdings gezeigt, dass die Anzahl an getauschten Positionen drastisch gesunken ist, wodurch sich Netz allgemein wenig „durchmischt“ hat und in der Verdrahtung ein wesentlich schlechter abgeschnitten hat.

Im zweiten Schritt werden die Kosten der Blöcke an der ZFT-Position und des aktuellen Blocks vor und nach einem Tausch berechnet, wobei ein Tausch durchgeführt wird, wenn die Summe der Kosten niedriger ist. Diese Methode ist sehr effektiv in der Reduzierung der Gesamtkosten, allerdings tritt der Fall der Vertauschung nicht besonders häufig ein.

Diese Punkte führen allgemein dazu, dass es je nach Parameter bei der Bereichssuche es zu vollständiger ausbleibender Vertauschung eines Iterationsschrittes kommt, wenn auch ein kostengünstiger Tausch nicht möglich ist. In diesem Fall wird nach 5 Iterationsschritten ohne einen einzelnen Tausch abgebrochen.

```
Successfully read net file alu4.net
1544 blocks, 1536 nets, 0 global nets
1522 clbs, 14 inputs, 8 outputs


Trying to place blocks by cost factors
The circuit will be mapped into a 40 x 40 array of clbs.

Breaking loop, because no changes occurred after 5 iterations.
Placing ended after 13 iterations. 680 blocks were switched
Placement runtime took: 00:00:00.091
Finished.
```

Abbildung 4.1: Beispiel für einen frühzeitigen Abbruch bei einer area swap Größe von 6 Feldern

Betrachtet man im Einzelnen die Laufzeit und die Anzahl der Vertauschungen zwischen VPR und dem ZFT-Algorithmus, dann lässt sich erkennen, dass VPR eine wesentlich größere Anzahl an Vertauschungen vornimmt. Daraus lässt sich nicht direkt schließen, ob ein Algorithmus eine effiziente Platzierung vornimmt, aber es spricht dafür, dass die Strategien zum Vertauschen nicht ausreichend sind. Dies erkennt man im speziellen an den Netzlisten clma und des. Die clma Netzliste ist besonders groß und dennoch wird mit ZFT eine deutlich geringere Anzahl an Vertauschungen vorgenommen. Hingegen ist des Netzliste groß aufgrund vieler IO-Ports, besitzt aber nicht besonders viele logische Blöcke. Daher können hier viele Vertauschungen vorgenommen werden, da viele freie Positionen vorhanden sind.

Netzliste	ZFT		VPR	
	Laufzeit	**Anzahl an Tauschungen**	**Laufzeit**	**Anzahl an Tauschungen**
alu4	00:02.704	180.139	00:06.724	21.587.731
apex2	00:02.165	268.577	00:09.721	28.370.671
apex4	00:01.326	276.448	00:05.495	16.145.310
bigkey	00:03.224	245.149	00:09.279	34.312.500
clma	00:10.535	24.657	01:23.183	229.881.696
des	00:02.808	847.280	00:09.937	34.238.464

-- Anzahl an Iterationen: 600, Swap area Größe: 8

Daraus kann sich schließen lassen, dass es weiterer Strategien benötigt, um effiziente Vertauschungen durchführen zu können. Eventuell könnte dabei beispielsweise eine vermischende Strategie weiterhelfen, bei der in einem Bereich ein Block gesucht wird, mit dem ein kostengünstiger Tausch durchgeführt werden könnten. Dieser heuristische Ansatz könnte allerdings die Laufzeit stärker negativ beeinflussen als die Strategien an sich, da hier Kosten für viele Blöcke berechnet werden müssten.

4.2. Zufällige gegen Gewichtsbasierte Initialplatzierung

Was sich mit den Ergebnissen der Verdrahtung nicht zeigen lässt, sind die Kosten der initialen Platzierung. Im kreativen Anteil wurde versucht mittels eines einfachen Algorithmus eine verbesserte initiale Platzierung zu generieren. In der folgenden Tabelle wurden die Kosten einer gültigen initialen Platzierung gegenübergestellt.

Die Platzierung mittels einer gewichte-basierten Sortierung der Netze führt zu einer starken Reduzierung der Kosten im Vergleich zu einer zufälligen Platzierung. Dabei konnten die Kosten und der kritische Pfad um etwa 20 % gesenkt werden. Bei einem iterativen Algorithmus kann eine solch verbesserte Initialplatzierung zu einer Reduzierung der Laufzeit führen, bzw. einer Reduzierung der Anzahl an Iterationen. Betrachtet man die Iterationsschritte von VPR, würden die geringeren Kosten der Initialplatzierung von Vorteil sein.

```
Initial Placement Cost: 606.586 bb_cost: 606.586 td_cost: 0 delay_cost: 0

        T        Cost Av. BB Cost  Av. TD Cost   Av Tot Del  P to P Del        d_max   Ac Rate  Std Dev  R limit      Exp  Tot. Moves    Alpha
--------  ---------- -----------  -----------    ---------  ----------        -----   -------  -------  -------  -------  ----------    -----
  66.754     611.298     611.298            0            0           0            0    0.9982     5.11       40        0      178411      0.5
  33.377     610.702     610.702            0            0           0            0    0.9963     5.13       40        0      356822      0.5
  16.689     609.168     609.168            0            0           0            0    0.9923     5.24       40        0      535233      0.5
  8.3443     607.799     607.799            0            0           0            0    0.9848     5.14       40        0      713644      0.5
  4.1722     605.126     605.126            0            0           0            0    0.9716     5.37       40        0      892055      0.5
  2.0861     598.174     598.174            0            0           0            0    0.9408     5.18       40        0     1070466      0.9
  1.8775     597.242     597.242            0            0           0            0    0.9346     5.06       40        0     1248877      0.9
  1.6897     595.147     595.147            0            0           0            0    0.9272     4.78       40        0     1427288      0.9
  1.5208     593.926     593.926            0            0           0            0    0.9184     5.13       40        0     1605699      0.9
  1.3687     592.237     592.237            0            0           0            0    0.9096     5.54       40        0     1784110      0.9
  1.2318     590.046     590.046            0            0           0            0     0.901     4.59       40        0     1962521      0.9
  1.1086     587.889     587.889            0            0           0            0    0.8914     5.14       40        0     2140932      0.9
 0.99776     585.382     585.382            0            0           0            0    0.8771     4.81       40        0     2319343      0.9
 0.89799     581.792     581.792            0            0           0            0    0.8639      5.2       40        0     2497754      0.9
 0.80819     578.856     578.856            0            0           0            0    0.8509     4.92       40        0     2676165      0.9
```

Abbildung 4.2: Beispiel für die iterative Reduzierung der Kosten von VPR

Netzliste	Zufällig		Gewichtsbasiert	
	Kosten	Krit. Pfad	Kosten	Krit. Pfad
alu4	610	2.16428e-07	492.3	1.96363e-07
apex2	901.6	2.86331e-07	730.9	2.51769e-07
apex4	507.2	2.04972e-07	415.9	1.7078e-07
bigkey	1098.2	1.55486e-07	786.7	1.27095e-07
clma	7992.1	9.65252e-07	6079	8.01516e-07
des	1267.6	3.41094e-07	909.7	2.76633e-07

Die Initialplatzierung könnte auch von einem mehr komplexen, konstruktiven Platzierungsalgorithmus übernommen werden. Hierbei stellt sich allerdings die Frage, wie groß der Implementierungsaufwand bei einer solchen Mischform ist. Die Sortierung der Netze hat zudem den Vorteil, dass dies in minimaler Zeit geschieht. In allen Fällen hat die Platzierung in etwa 20 ms gedauert, egal ob zufällig oder gewichtsbasiert.

5. Fazit

In der Arbeit wurde gezeigt, wie mit Hilfe der Kräfteplatzierung logische Blöcke auf einer Layout-Fläche platziert werden können. Die eingesetzten Strategien und die Implementierung haben gezeigt, dass die theoretische Formel zur Berechnung des ZFT von diversen Parametern abhängen kann. In diesem Fall ist es nicht gelungen, eine Platzierung zu generieren, welche sich dem Optimum nähern kann, da es nicht ausreichend Vertauschungen gegeben hat. Viele Veränderungen an den Strategien, die hier nicht alle genannt wurden, haben kein oder nur wenig Auswirkungen auf die Platzierung gehabt.

Darüber hinaus hat sich gezeigt, dass sich die Kosten der Platzierung auch mit effizienten Mitteln bei der Initialplatzierung reduzieren lässt, wobei eine Laufzeitorientierte Lösung implementiert wurde. Diese ist gegenüber einer zufälligen Platzierung im Vorteil, da hier kritische Pfade allgemein klein gehalten werden können.