Rechnergestützter Entwurf digitaler Systeme

Kräfteplazierung

Fachrichtung technische Informatik

28. Februar 2022
Wintersemetester 2021/2022

Guillaume Fournier-Mayer
tinf-101922
tinf101922@stud.fh-wedel.de

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

Ziel des Praktikums ist das Entwerfen einer Software, die eine Logikschaltung auf eine gegebene FPGA-Architektur platziert. Die zu entwerfende Software wird dabei durch eine weitere Software Namens *Versatile Place and Route (VPR)* unterstützt und hält sich dabei an die vergebenen Schnittstellen, das heißt Ein- und Ausgabeformate von VPR. Der grobe Ablauf des gesamten Entwurfes besteht darin, die Netzliste einzulesen und mithilfe der FPGA-Architektur eine passende Platzierung zu ermitteln. In der Netzliste, die als *.net Datei abgelegt wird, befinden sich alle Informationen über Logikblöcke sowie Eingangs- und Ausgangspins des FPGA und wie diese miteinander verbunden sind. Die FPGA-Architektur die als *.arch Datei gespeichert wird beinhaltet Daten über die Beschaffenheit des FPGAs. Dazu gehören zum Beispiel Kanalbreite oder der genaue Aufbau der Logikblöcke.
War die Platzierung erfolgreich, wird eine Platzierungsdatei *.place mit den Positionen der einzelnen Logikblöcke erstellt. Als nächster Schritt ließt VPR die Platzierungsinformationen ein und verdrahtet die Blöcke mithilfe der Netzliste und der FPGA-Architektur. Die daraus entstehende *.route Datei beinhaltet wiederum die Informationen wie die Blöcke miteinander verbunden sind. Je besser die Platzierung, desto einfacher ist der Verdrahtungsschritt. Des Weiteren können schon beim Platzieren verschiedene Optimierungen vorgenommen werden um zum Beispiel den kritischen Pfad zu minimieren.

1.2 Kräfteplatzierung

Die Kräfteplatzierung ist ein Platzierungalgorithmus, der in Analogie zu einem System steht, in dem alle Blöcke durch Federn miteinander verbunden sind. Dabei üben die Federn in Abhängigkeit zum Abstand der Blöcke Kraft auf diese aus. Daraus folgt, dass die optimale Position der einzelnen Blöcke diejenige ist, in der ein Kräftegleichgewicht herrscht. Die Kraft (1.2) zwischen zwei Blöcken a und b kann dabei durch das Hooksche Gesetz beschrieben werden, wobei d_{ab} (1.1) den euklidischen Abstand zwischen a und b und $w_a b$ die Gewichtung der Verbindung beschreibt. Für ein Block in einem Netz, der mit n Blöcken verbunden ist, ist die Gesamtkraft F_{ges} dementsprechend die Summe aller Kräfte (1.3).

$$w_{ab} = \sqrt{\Delta x_{ab}^2 + \Delta y_{ab}^2} \tag{1.1}$$

$$F = w_{ab} \cdot d_{ab} \tag{1.2}$$

$$F_{ges} = \sum_{j=1}^{n} (w_{ij} \cdot d_{ij}) \tag{1.3}$$

1.2.1 Zero-Force-Target (ZFT)

Als Zero-Force-Target wird ein Zustand verstanden, indem ein Block im Kräftegleichgewicht ist. Block D befindet sich in der Abbildung (1.1) im Kräftegleichgewicht. Seine Position ist somit gleich der ZFT-Position [1][S. 107].

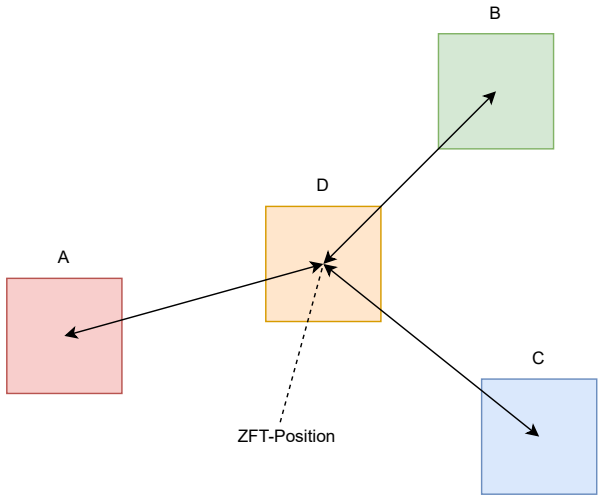

Abbildung 1.1: ZFT-Position

Zur Bestimmung der ZFT-Position werden die Kräftegleichungen (1.4) null gesetzt und nach der ZFT-Position (x_i^0, y_i^0) umgestellt sodass die Gleichungen (1.5) gebildet werden können [1][S. 107].

$$\sum_j w_{ij} \cdot (x_j^0 - x_i^0) = 0 \quad \sum_j w_{ij} \cdot (y_j^0 - y_i^0) = 0 \tag{1.4}$$

$$x_i^0 = \frac{\sum_j w_{ij} \cdot x_j}{\sum_j w_{ij}} \quad y_i^0 = \frac{\sum_j w_{ij} \cdot y_j}{\sum_j w_{ij}} \tag{1.5}$$

1.2.2 Grober Ablauf

Der grobe Ablauf der Kräfteplatzierung besteht zunächst darin, eine Initialplatzierung zu finden. War dies erfolgreich, werden iterativ die ZFT-Position aller Blöcke berechnet. Ist die ZFT-Position frei, kann der aktuell betrachtete Block auf die freie Position verschoben werden. Ist die Zielposition belegt, muss eine Belegungsoption (1.2.3) angewandt werden. Ist dies geschehen, wird der nächste Block betrachtet. Dies geschieht so lange, bis eine Abbruchbedingung erfüllt ist [1][S. 108].

1. Zufällige Initialplatzierung
2. Berechnen der ZFT-Position des aktuellen Blocks
   - ZFT-Position ist frei → Block auf ZFT-Position verschieben
   - ZFT-Position ist belegt → Belegungsoption (1.2.3) wählen
3. Schritt zwei wiederholen bis Abbruchbedingung erfüllt ist

1.2.3 Belegungsoptionen

- Verschieben des Blockes möglichst zu einer Zellenposition nahe der ZFT-Position
- Berechnen der Kostenveränderung beim Austausch von zwei Blöcken. Bei verringerten Kosten werden die Blöcke getauscht
- **Chain Move**: Der zu verschiebende Block wird an die Zielposition verschoben, ohne die Kostendifferenz zu berechnen. Der verdränge Block wird auf die nächstgelegende Position verschoben. Ist diese auch belegt, kommt es zu einer Kettenverschiebung (Chain Move)
- **Ripple Move**: Der zu verschiebende Block wird an die Zielposition verschoben und fixiert. Die ZFT-Position des verdrängten Blockes wird berechnet und nach demselben Prinzip verschoben. Dies geschieht so lange, bis alle Blöcke fixiert bzw. platziert sind.

[1][S. 107f]

2 Umsetzung

Die Entwicklung der Platzierungssoftware erfolgte in Java-SE 11. Der grobe Ablauf des Programmes besteht darin, die Netzliste sowie die fixierten Pads einzulesen und daraus einen Graphen zu erstellen. Dadurch, dass es im Praktikum nur eine relevante FPGA-Architektur zu betrachten gab, wurde vom Einlesen der *.arch Datei abgesehen und die benötigten Parameter wurden über Konstanten direkt im Programmcode abgelegt. Wurde der Graph erfolgreich erstellt wird eine Initialplatzierung vorgenommen. Kapitel (2.2) beschreibt die vom Grundalgorithmus abweichende Vorgehensweise um die Ergebnisse der Platzierung zu optimieren. Nach Abschluss der Initialplatzierung beginnt die eigentliche Optimierung der Platzierung. Kapitel (2.3) geht dabei auf die genauen Einzelheiten des Algorithmus ein. Nach Abschluss des Algorithmus wird aus der aktuellen Platzierung die Platzierungsdatei erstellt und ausgegeben.

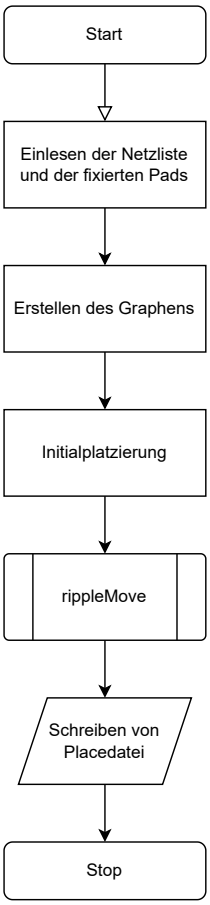

Abbildung 2.1: Programmablaufplan

2.1 Programmstruktur

Abbildung (2.2) zeigt das UML-Diagramm des Programmes. Die Hauptlogik des Programmes befindet sich in der FPGA-Klasse, die zuständig für die Platzierung der Logikblöcke ist. Als wichtigste Komponente gilt der Graph, der durch die eingelesenen Blöcke sowie die fixierten Pads erstellt wird und der FPGA-Instanz beim Instanziieren übergeben wird. Über die Methoden *initPlace* oder *randomPlace* wird eine Initialplatzierung vorgenommen. Des Weiteren wird über die Methode *rippleMove* die eigentliche Optimierung der Platzierung vorgenommen.

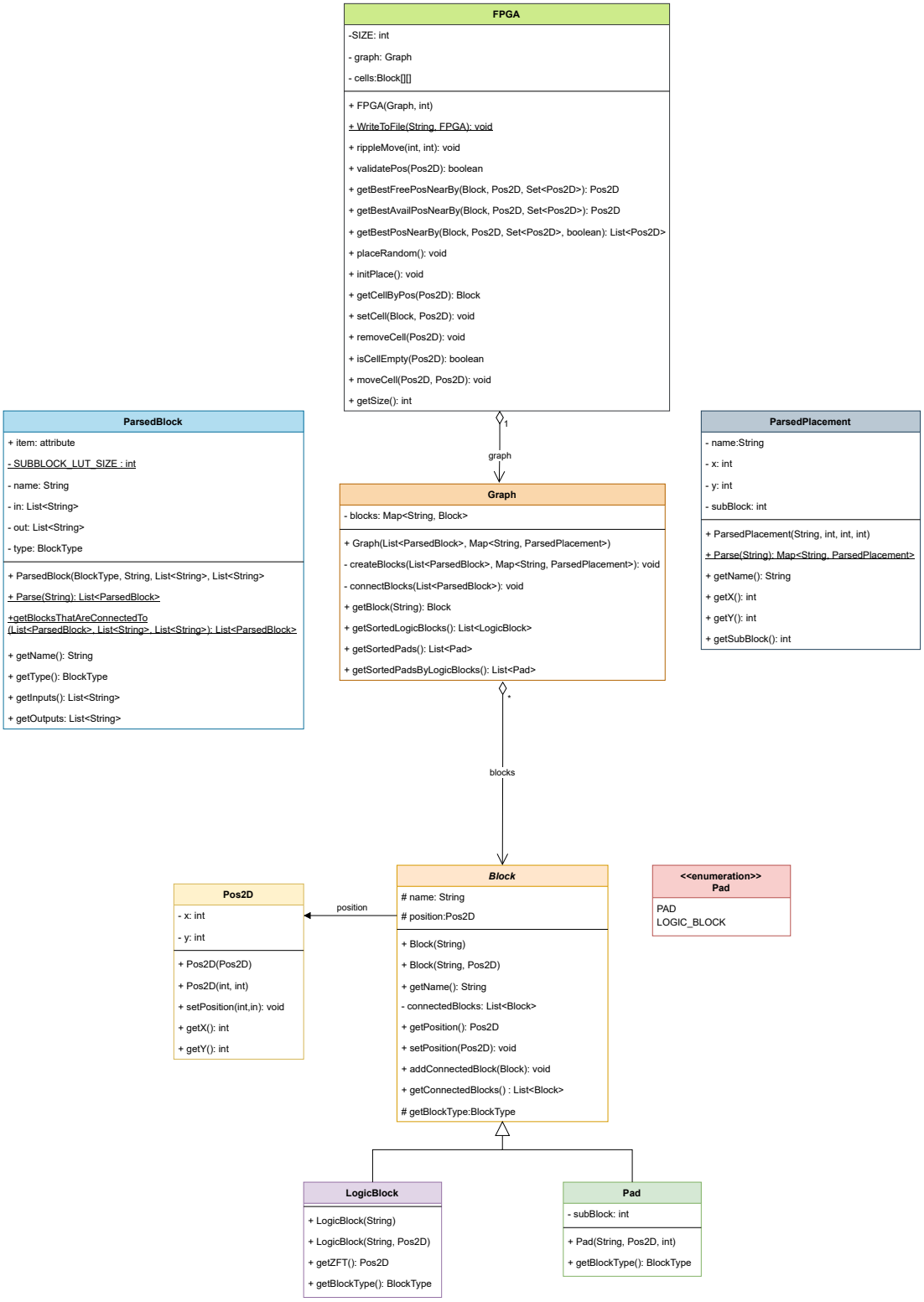

Abbildung 2.2: UML Diagramm

2.2 Initialplatzierung

Dadurch, dass die Kräfteplatzierung ein iterativer Algorithmus ist, müssen die Blöcke gesetzt sein, um die jeweilige ZFT-Position zu berechnen. Ein möglicher Ansatz ist es, die Blöcke zufällig zu platzieren. Der Vorteil dieser Lösung ist, dass dieses Verfahren einfach umgesetzt werden kann. Der eindeutige Nachteil liegt in einer potenziellen schlechten Initialplatzierung, welche die Laufzeit und Effektivität der Kräfteplatzierung beeinflussen kann. In dieser Arbeit wurde von einer zufälligen Platzierung abgesehen und ein Ansatz gewählt, in dem die Logikblöcke nach Möglichkeit direkt auf ihre entsprechende ZFT-Position gesetzt werden. Die Problematik ist dabei, dass zur Berechnung der ZFT-Position mindestens zwei Blöcke, die mit dem zu platzierenden Block verbunden sind, bereits gesetzt sein müssen. Dies ist jedoch im Initialzustand nicht gegeben. Eine Lösung dieses Problems sind die Pads, die bereits bei der Konstruktion des Graphens an ihren entsprechenden Positionen fixiert werden.
Der Ablauf der Initialplatzierung wird in Abbildung (2.3) dargestellt. Zunächst wird eine Liste mit den zu platzierenden Logikblöcken absteigend nach ihrem Verbindungsgrad erstellt. Dies erhöht die Wahrscheinlichkeit, in den ersten Iterationsschritten Blöcke zu platzieren, die mit mehr als zwei fixierten Pads verbunden sind. Des Weiteren wird eine Zählervariable auf die Anzahl der zu platzierenden Blöcke gesetzt. Als Nächstes werden der Reihe nach Blöcke aus der Liste entnommen und zunächst geprüft, ob diese schon platziert wurden. Ist dies der Fall, wird der nächste Block aus der Liste entnommen. Wurde der Block jedoch noch nicht platziert, wird die ZFT-Position des Blockes ermittelt und geprüft, ob diese der Initialplatzierung (-1, -1) entspricht. In diesem Fall ist der Block mit zu wenig Blöcken verbunden, die bereits platziert sind und die ZFT-Postion kann zurzeit nicht ermittelt werden. Ist dies nicht der Fall wird der Vorgang mit dem nächsten Block versucht. Kann jedoch eine valide ZFT-Position ermittelt werden, wird zunächst geprüft, ob diese bereits belegt ist. In solch einem Fall wird die nächste freie Position um die Zielposition gesucht und der Block auf diese gesetzt. Ist die Zielposition jedoch, kann der Block direkt gesetzt werden. In beiden Fällen wir der *unset_counter* dekrementiert und die Schleife versucht den nächsten Block zu platzieren.

Ein Nachteil dieser Lösung ist, dass die berechnete ZFT-Position eines Blockes nicht zwangsläufig der korrekten ZFT-Position entspricht. Dies ist dadurch zu erklären, dass zum Zeitpunkt der Positionberechnung noch nicht alle verbundenen Blöcke platziert sind und somit das Ergebnis verfälscht wird.

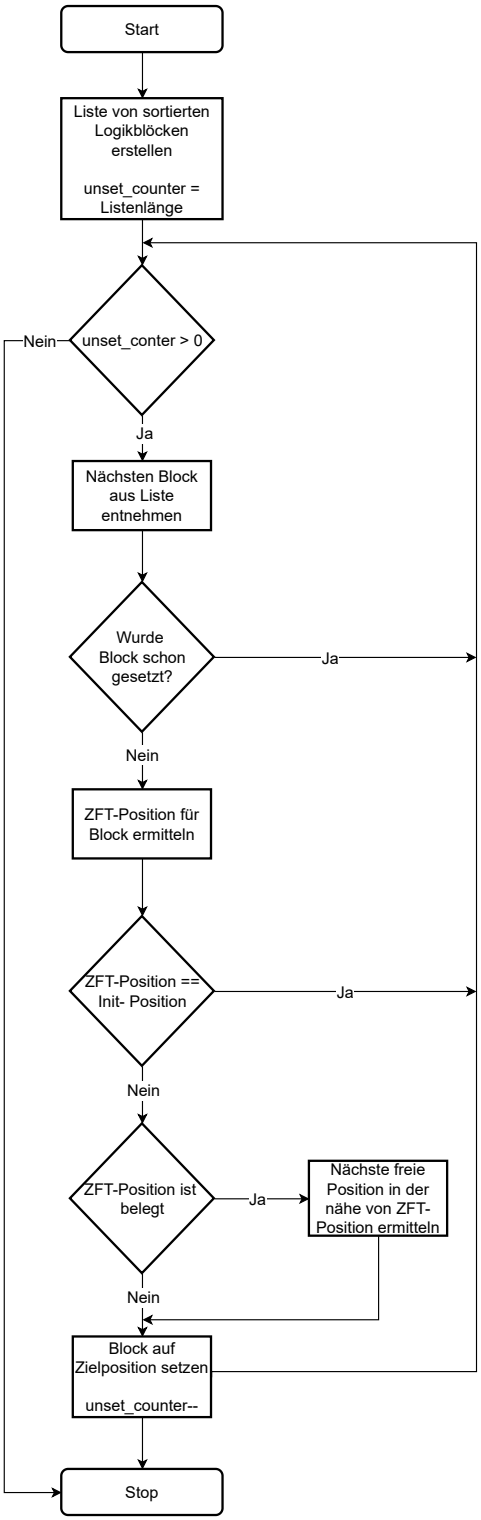

Abbildung 2.3: Initialplatzierung Ablaufplan

2.3 Ripple move

Zunächst werden die Logikblöcke absteigend nach ihrem Verbindungsgrad sortiert. Das heißt, dass Blöcke mit mehr Verbindungen weiter vorne in der Liste stehen. Dies hat den Vorteil, dass diese Blöcke besser platziert werden können. Als Nächstes werden die Blöcke iterativ durchlaufen und für jeden Block die ZFT-Position ermittelt.
Ist die Zielposition schon fixiert, das heißt, dass ein anderer Block in diesem Iterationsschritt auf die Zielposition gesetzt wurde, wird der *rippleIteration* Zähler erhöht und verglichen, ob dieser größer bzw. gleich der *maxRippleIteration* Variable ist. Ist dies der Fall, wird in der Nähe der Zielposition die nächste freie Zelle gesucht und der aktuelle Block auf diese Position gesetzt. Des Weiteren wird die maxIteration Zählervariable dekrementiert, alle Fixierungen werden gelöst und der Algorithmus beginnt eine neue Iteration.
In dem Fall, dass die *maxRippleIterations* nicht überschritten wurde, wird die beste Position in der Nähe der Zielposition gesucht. Dabei werden alle anliegenden Positionen betrachtet und die Position ausgewählt, an dem der aktuelle Block der niedrigsten Kraft ausgesetzt ist. Die neue Zielposition wird daraufhin wieder auf die drei Hauptbedingungen geprüft.
Ist die Zielposition nicht fixiert und der Block ist schon auf seiner Zielposition, wird die *rippleIteration* Variable zurückgesetzt und die Zielposition fixiert.
Ist die Zielposition jedoch belegt und nicht fixiert, wird der aktuelle Block auf die Zielposition gesetzt, die Position fixiert, der *rippleIteration* Zähler zurückgesetzt und der verdrängte Block wird auf den aktuellen Block gesetzt. Als Nächstes startet der Algorithmus an der Stelle, an dem die ZFT-Position für den aktuellen Block (verdrängter Block) berechnet wird.
Als letzte Bedingung kann die Zielposition unbelegt sein. Ist dies der Fall, wird der aktuelle Block auf die Zielposition gesetzt, die Position fixiert und der *rippleIteration* Zähler zurück gesetzt. Ist der *maxIterations* auf Null dekrementiert worden, wird der Algorithmus beendet.

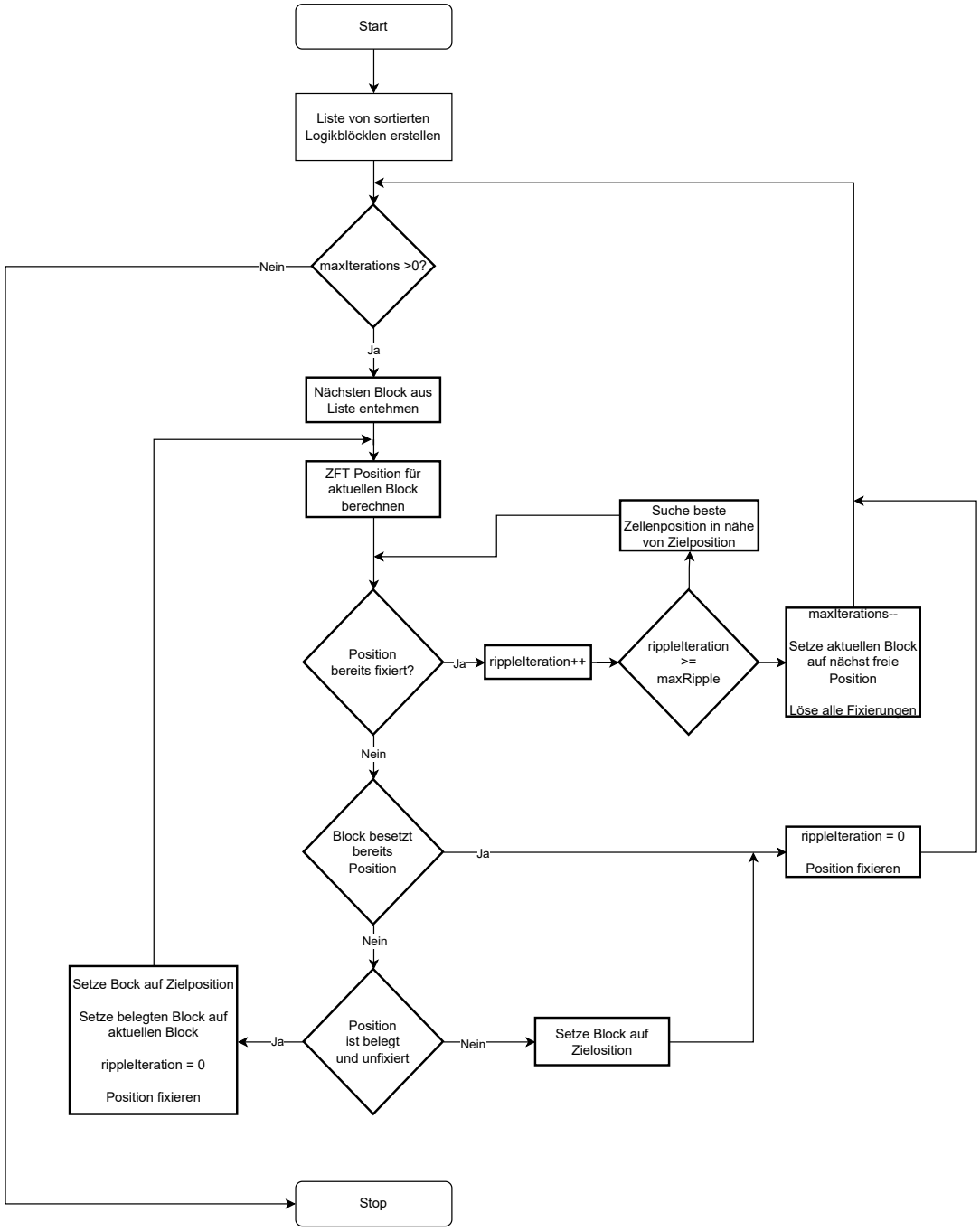

Abbildung 2.4: Ripplemove Ablaufplan

3 Auswertung

Im folgenden Kapitel werden die Ergebnisse ausgewertet und in Relation zu den Benchmarks von VPR gesetzt. Zunächst werden in Kapitel (3.1) die zwei verschiedenen Initialplatzierungen miteinander verglichen. Des Weiteren werden in Kapitel (3.2) die Kosten der Platzierung der jeweiligen Benchmarks mit den Platzierungskosten von VPR verglichen. Abschließend werden in Kapitel (3.3) die Ergebnisse der vorgenommenen Verdrahtung durch VPR verglichen. Alle Benchmarks wurden dabei mit einer *maxIterations* von 1000 und einer *maxRippleIterations* von 20 ausgeführt

3.1 Initialplatzierung

Die Ergebnisse der Platzierung sind unter anderem abhängig von der Initialplatzierung. Daher wird die in dieser Arbeit entwickelte Initialplatzierung in diesem Kapitel mit einer zufälligen Initialplatzierung verglichen. Als Vergleichsgrößen werden die Laufzeit des Gesamtalgorithmus sowie die Platzierungskosten verwendet. Dabei gilt für beide Größen, dass ein niedrigerer Wert ein besseres Endergebnis darstellt. Tabelle (3.1) stellt die oben genannten Zusammenhänge tabellarisch dar. Die Differenzspalte ist so zu verstehen, dass ein negativer Wert ein besseres Ergebnis darstellt. Für die Platzierungen ist die Laufzeit jedoch nicht ausschlaggebend. Eine höhere Laufzeit kann in manchen Fällen in Kauf genommen werden, um damit die Platzierungskosten niedriger zu halten.
Auffällig ist, dass nicht jeder Benchmark von der ZFT-Initialplatzierung profitiert. Wie in Kapitel (2.2) erwähnt, ist dies damit zu erklären, dass die berechnete ZFT-Position für manche Blöcke nicht exakt berechnet werden kann und somit die Initialplatzierung nicht ideal ist. Des Weiteren ist denkbar, dass die Zufallsplatzierung durch Zufall eine idealere Initialplatzierung für das iterative Verfahren der Kräfteplatzierung findet.
In diesem Fall kann durch die ZFT-Initialplatzierung in 11 von 20 Fällen eine bessere Platzierung erzielt werden. Zu erwähnen ist auch, dass sich in 8 von 11 Fällen zusätzlich die Laufzeit erhöht.

Benchmark	**ZFT-Platzierung**		**Zufallsplatzierung**		**Differenz (%)**	
	Zeit (ms)	Kosten	Zeit (ms)	Kosten	Zeit	Kosten
alu4	22347	390,582	22389	391.755	-0.19	-0.30
apex2	35065	538,017	31763	531.647	10.40	1.20
apex4	13855	310,617	13833	299.461	0.16	3.73
bigkey	16534	505,461	15362	515.345	7.63	-1.92
clma	408551	3555,85	507620	3493.84	-19.52	1.77

des	2272	368,297	1858	368.784	22.28	-0.13
diffeq	26617	385,295	25555	385.582	4.16	-0.07
dsip	22176	456,205	20358	460.81	8.93	-1.00
elliptic	346639	1256,4	319950	1314.94	8.34	-4.45
ex5p	18487	268,953	16410	276.82	12.66	-2.84
ex1010	123064	1266,36	108375	1239.9	13.55	2.13
frisc	138902	1245,97	142276	1359.29	-2.37	-8.34
misex3	14292	340,409	10634	337.241	34.40	0.94
pdc	213463	1731,39	158013	1725.91	35.09	0.32
s298	50932	538,604	48419	538.008	5.19	0.11
s38417	210034	1977,98	187441	1893.73	12.05	4.45
s38584.1	1179389	2621,59	884384	2701.24	33.36	-2.95
seq	33346	465,327	35338	464.718	-5.64	0.13
spla	101775	1188,14	98363	1218.62	3.47	-2.50
tseng	8084	213,566	8445	230.618	-4.27	-7.39

Tabelle 3.1: Kostenfunktionen und Laufzeit unterschiedlicher Initialplatzierung

3.2 Platzierungskosten

In diesem Kapitel werden die Ergebnisse der Platzierungen, die durch VPR und durch den entwickelten Algorithmus berechnet wurden, miteinander verglichen. Je niedriger die Kosten, desto besser ist das Endergebnis. Die Tabelle (3.2) stellt die Ergebnissee der 20 Benchmarks dar. Eine negative Differenz bedeutet, dass die VPR Platzierung um den jeweiligen prozentualen Wert weniger Platzierungskosten verursacht. Die Ergebnisse der Kräfteplatzierung sind bis zu $\approx 75\%$ schlechter als die der VPR-Platzierung.
Dabei ist zu erwähnen, dass VPR einen Algorithmus verwendet der auf Simulated annealing zurückzuführen ist. Dieser ist ein weit verbreiteter Algorithmus und liefert sehr gute Ergebnisse. Dies bedeutet im Umkehrschluss, dass die Ergebnisse schwer miteinander zu vergleichen sind.

Benchmark	**ZFT-Kosten**	**VPR-Kosten**	**Differenz (%)**
alu4	390,582	190,135	-51,32
apex2	538,017	269,765	-49,86
apex4	310,617	179,329	-42,27
bigkey	505,461	185,977	-63,21
clma	3555,85	1387,05	-60,99
des	368,297	227,843	-38,14
diffeq	385,295	146,394	-62,00
dsip	456,205	169,991	-62,74

elliptic	1256,4	457,203	-63,61
ex5p	268,953	162,012	-39,76
ex1010	1266,36	655,429	-48,24
frisc	1245,97	515,59	-58,62
misex3	340,409	190,205	-44,12
pdc	1731,39	898,44	-48,11
s298	538,604	203,949	-62,13
s38417	1977,98	671,75	-66,04
s38584.1	2621,59	657,87	-74,91
seq	465,327	247,658	-46,78
spla	1188,14	593,969	-50,01
tseng	213,566	92,0471	-56,90

Tabelle 3.2: Kosten der Platzierungen im Vergleich zu VPR

3.3 Kritischer Pfad und Kanalbreite

Im folgenden Kapitel werden die Ergebnisse der Verdrahtungen, die durch VPR durchgeführt wurden, miteinander verglichen. Dabei hat VPR jeweils die vorgegebenen Platzierungsbenchmarks sowie die berechneten Platzierungen durch die entworfene Software verdrahtet. Die wichtigsten Kenngrößen nach dem Verdrahtungsschritt sind die Kanalbreite sowie die Dauer des kritischen Pfades. Beide Werte sind je besser, desto niedriger sie sind und werden in Tabelle (3.3) dargestellt. Die negativen Werte in der Differenzspalte besagen, dass das VPR-Routing eine niedrigere Kanalbreite bzw. einen niedrigeren kritischen Pfad verursacht. Dass hierbei das VPR-Routing besser ist, liegt daran, dass wie schon in Kapitel (3.2) erläutert die Platzierungskosten geringer sind. Dementsprechend kann einfacher und mit niedrigerer Kanalbreite geroutet werden. Die Differenz der Kanalbreite hat jedoch keinen direkten Zusammenhang mit der Differenz des kritischen Pfades, wie das Beispiel des Benchmarks *bigkey* oder *des* zeigt. Obwohl in beiden Fällen die Kanalbreite im ZFT-Routing weit aus höher ist als im VPR-Routing, ist die Dauer des kritischen Pfades nur $\approx 8\%$ bzw $\approx 6\%$ niedriger.

Benchmark	**ZFT-Routing**		**VPR-Routing**		**Differenz (%)**	
	Kanalbreite	K. Pfad	Kanalbreite	K. Pfad	Kanalbreite	K. Pfad
alu4	22	1.54E-07	11	1.12E-07	-50.00	-27.50
apex2	24	2.01E-07	12	1.41E-07	-50.00	-29.87
apex4	21	1.47E-07	13	1.31E-07	-38.10	-10.62
bigkey	23	1.14E-07	6	1.06E-07	-73.91	-7.40
clma	36	4.11E-07	13	2.54E-07	-63.89	-38.30
des	12	1.29E-07	8	1.21E-07	-33.33	-5.66
diffeq	21	1.84E-07	8	8.82E-08	-61.90	-52.11

dsip	23	9.21E-08	6	7.98E-08	-73.91	-13.44
elliptic	32	2.55E-07	11	1.83E-07	-65.63	-28.45
ex5p	22	1.29E-07	14	1.09E-07	-36.36	-15.16
ex1010	22	2.87E-07	11	2.48E-07	-50.00	-13.61
frisc	27	3.01E-07	13	1.69E-07	-51.85	-43.85
misex3	20	1.34E-07	12	1.06E-07	-40.00	-20.86
pdc	34	3.17E-07	18	2.39E-07	-47.06	-24.52
s298	22	2.97E-07	8	1.98E-07	-63.64	-33.33
s38417	26	2.81E-07	8	1.77E-07	-69.23	-36.83
s38584.1	38	2.10E-07	9	1.27E-07	-76.32	-39.50
seq	23	1.39E-07	12	1.27E-07	-47.83	-9.00
spla	28	2.64E-07	14	1.76E-07	-50.00	-33.29
tseng	17	1.09E-07	7	9.00E-08	-58.82	-17.57

Tabelle 3.3: Kritischer Pfad und Kanalbreite im Vergleich zu VPR

4 Fazit

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] Jens Lienig. *Layoutsynthese elektronischer Schaltungen*. Springer-Verlag Berlin Heidelberg, 2006. ISBN: 3-540-29627-1.