Rechnergestützter Entwurf digitaler Systeme

Kräfteplazierung

Fachrichtung technische Informatik

25. 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 dabei 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].

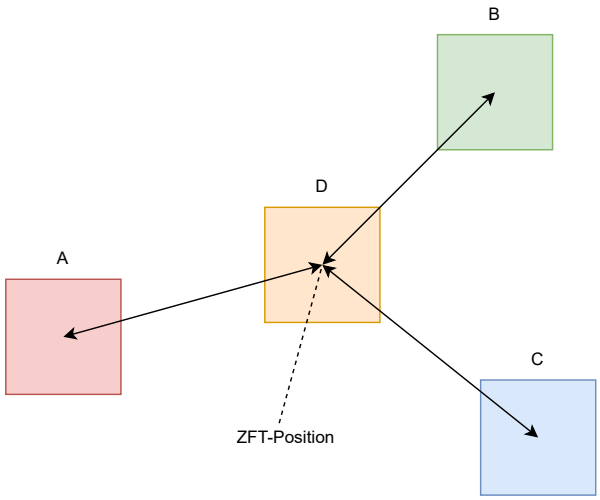

Abbildung 1.1: ZFT-Position

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 Initialplazierung

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, welches die Laufzeit und Effektivität der Kräfteplatzierung beeinflussen kann.

1.2.3 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 (1.2.4) 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 (1.2.4) wählen
3. Schritt zwei wiederholen bis Abbruchbedingung erfüllt ist

1.2.4 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 Kettenverschieben (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 ein 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 hierbei 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.

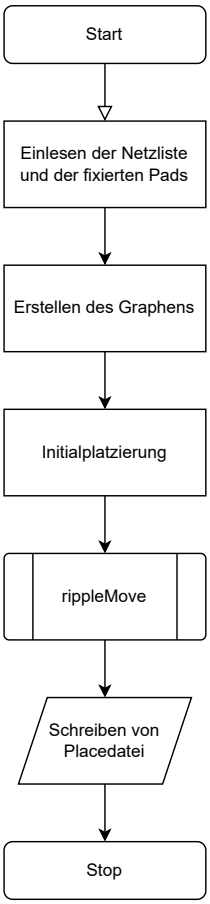

Abbildung 2.1: Programmablaufplan

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 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.

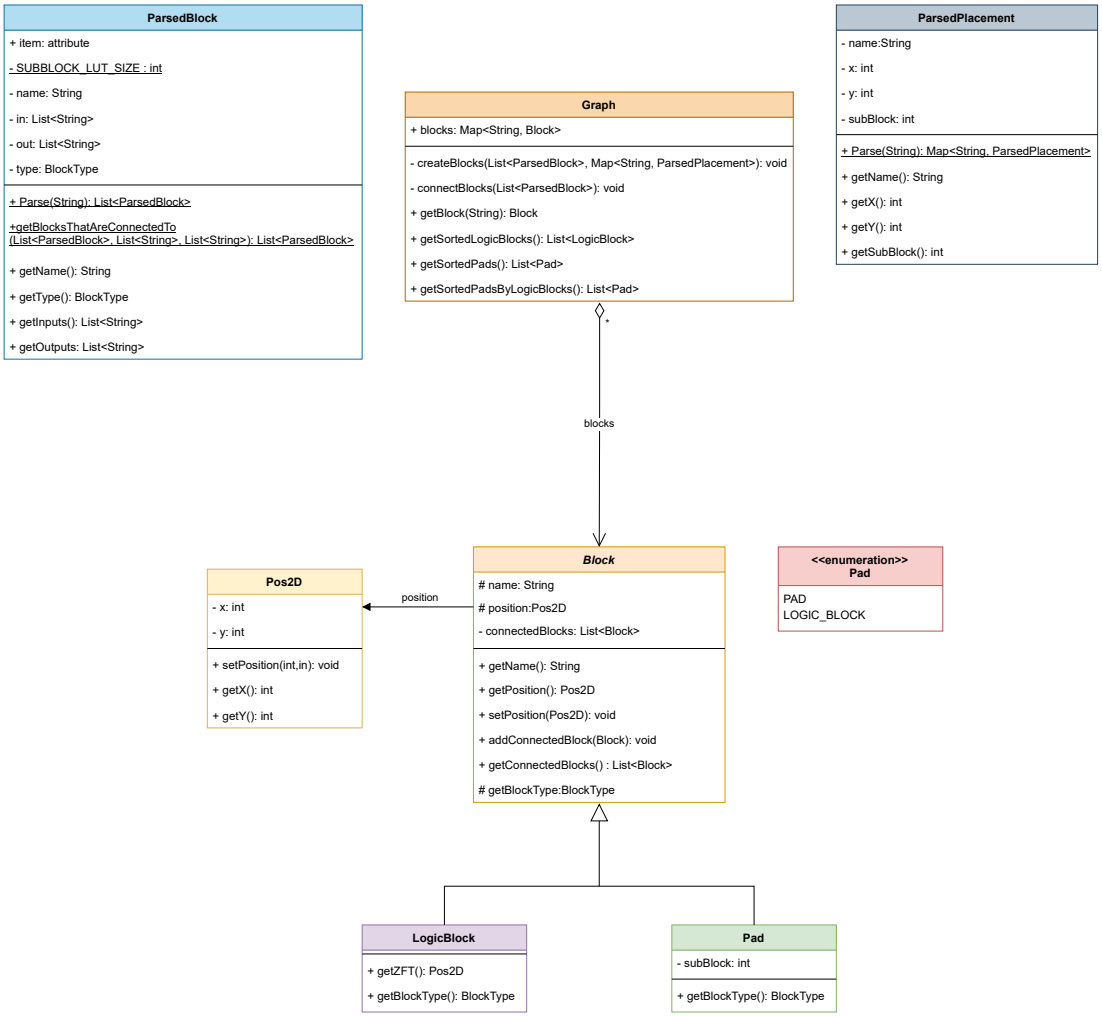

Abbildung 2.2: UML Diagramm

2.2 Initialplatzierung

2.3 Algorithmusablaufplan

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 Zunächst 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 werden 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.

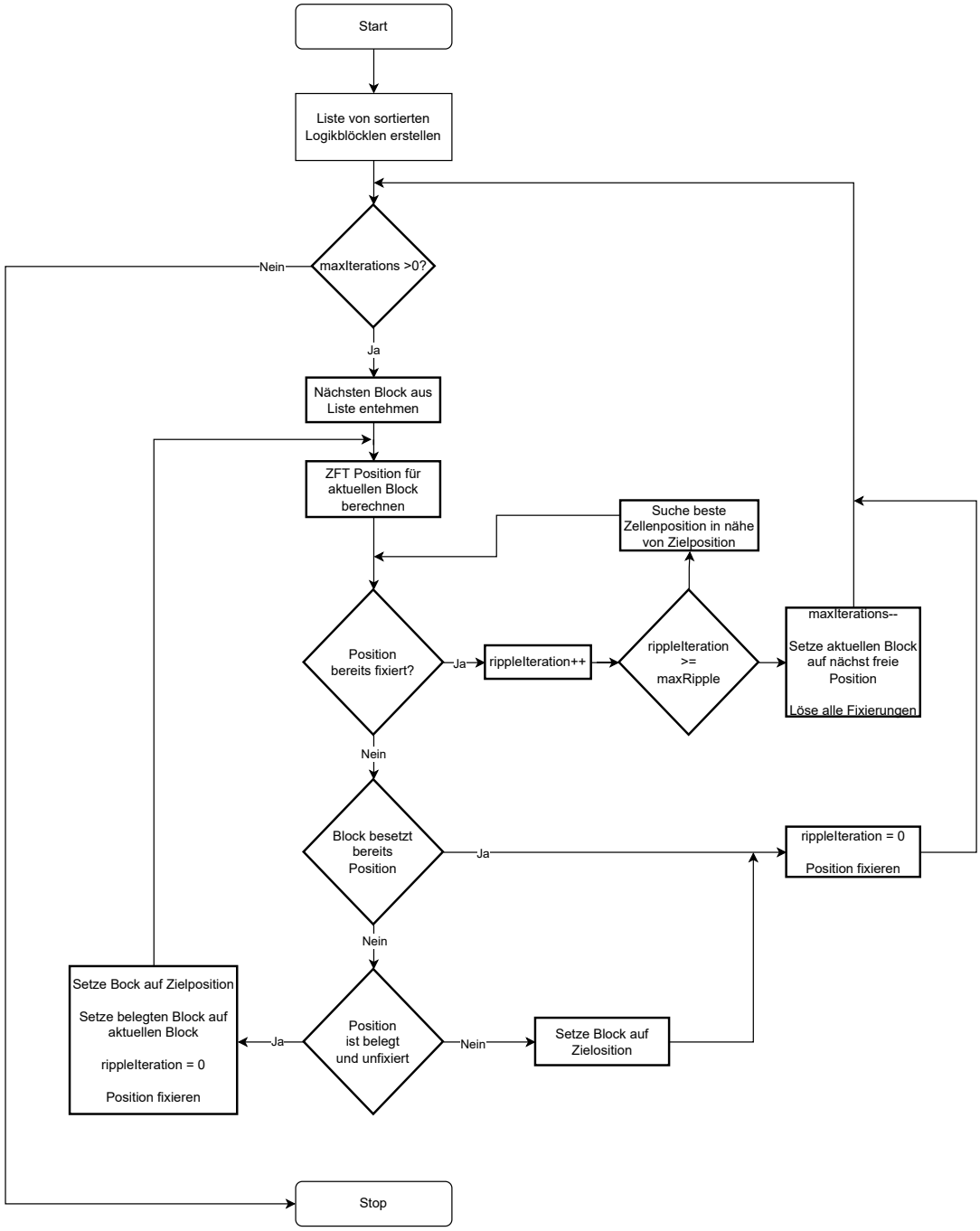

Abbildung 2.3: Algorithmusablaufplan

3 Auswertung

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

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