EDA-Praktikum

Kräfteplatzierung mit definierter Initialisierung

Von:

**Janik Piepenhagen**

Studiengang: Technische Informatik

Matrikelnummer: Tinf103697

Fachsemester: 6

Verwaltungssemester: 10

24.05.2023

Inhaltsverzeichnis

[1. Einleitung 3](#_Toc132137089)

[2. Entwicklungskonfiguration 4](#_Toc132137090)

[2.1. Skripte 4](#_Toc132137091)

[3. Grundlagen 5](#_Toc132137092)

[3.1. Was ist Electron? 5](#_Toc132137093)

[3.1.1. Prozess-Modell 5](#_Toc132137094)

[3.2. React 6](#_Toc132137095)

[3.2.1. Primereact 6](#_Toc132137096)

[3.3. TypeScript 6](#_Toc132137097)

[4. Problemanalyse und Realisation 7](#_Toc132137098)

[4.1. SerialPort Integration 7](#_Toc132137099)

[4.2. Speicherung von Einstellungen 7](#_Toc132137100)

[4.3. Dialogverarbeitung 7](#_Toc132137101)

[4.4. Darstellung von Daten 8](#_Toc132137102)

[5. Implementationsdetails 9](#_Toc132137103)

[5.1. Komponentendesign 9](#_Toc132137104)

[5.2. Interprozesskommunikation 10](#_Toc132137105)

[5.3. Applikation-Kontext 12](#_Toc132137106)

[5.4. SerialPort Integration 14](#_Toc132137107)

[6. Releasemanagement 16](#_Toc132137108)

[6.1. Releasebau mit Docker 16](#_Toc132137109)

[7. Test 17](#_Toc132137110)

[7.1. Test mit Jest 17](#_Toc132137111)

[8. Quellen 19](#_Toc132137112)

# 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 durch den Verlauf von Verbindungsleitungen fest.

## Strategie

Die gewählte Strategie zur Platzierung der Schaltungselemente basiert auf der Kräfteplatzierung. Die Kräfteplatzierung basiert auf einem physikalischen Modell, bei dem frei beweglichen Körper mit Federn verbunden sind. Je nach Masse der Körper und Steife der Federn strebt das System einen Zustand des Kräftegleichgewichts an.

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) berechnet werden.

## 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.

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.

# Implementierungsdetails

Im folgenden Abschnitt soll näher auf den inhaltlichen blabla eingegangen werden.

## Berechnung der ZFT-Position

## Iterationsschritte

## Kreativer Anteil

Zur Berechnung der ZFT-Position ist es notwendig, dass alle Blöcke eine Position besitzen. Dazu muss allerdings initial ein Zustand erzeugt werden. Wie auch bei anderen iterativen Algorithmen wird daher eine zufällige Initialplatzierung festgelegt. Eine zufällige Platzierung kann jedoch Einfluss auf die Laufzeit des Algorithmus haben, da diese bei 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. Die Kosten berechnen sich aus dem halben Netzumfang und der Anzahl an Kreuzungen des Netzes, welcher über die Anzahl an Blöcken in einem Netz interpoliert wird. Die interpolierten Werte wurden aus VPR übertragen und liegen zwischen 1.0 und 2.7 bei 1 bis 50 Blöcken pro Netz.

Anhand dieser Formel werden alle Netze nach Größe der Kosten sortiert, sodass an erster Stelle das Netz mit den Größten Kosten und an letzter das mit den Kleinsten Kosten befindet. Nun werden alle Blöcke, die sich in dem Netz befinden, reihenweise auf dem Netz 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.

# Benchmarking

## 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 Masse an Dateien wurde daher die Versuchsdurchführung im Multithreading betrieben, wobei immer 6 Threads verwendet wurden.

### Hardwarekomponenten

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

### Softwarekomponenten

|  |  |
| --- | --- |
| **Betriebssystem** | Microsoft Windows 10 Home  Version: 10.0.19045 Build 19045 |
| **Entwicklungsumgebung** | IntelliJ IDEA  Version 2022.1.3 |
| **JDK** | Version 17 |

| Netzliste | # Logische Blöcke | ZFT | | | | VPR | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Laufzeit | Kosten | Kr. Pfad | Channel Breite | Laufzeit | Kosten | Kr. Pfad | Channel Breite |
| alu4 | 1522 |  |  |  |  |  |  |  |  |
| apex2 | 1522 |  |  |  |  |  |  |  |  |
| apex4 | 1522 |  |  |  |  |  |  |  |  |
| bigkey | 1522 |  |  |  |  |  |  |  |  |
| clma | 8383 |  |  |  |  |  |  |  |  |
| des |  |  |  |  |  |  |  |  |  |
| diffeq |  |  |  |  |  |  |  |  |  |
| dsip |  |  |  |  |  |  |  |  |  |
| elliptic |  |  |  |  |  |  |  |  |  |
| ex1010 |  |  |  |  |  |  |  |  |  |
| exp5 |  |  |  |  |  |  |  |  |  |
| frisc |  |  |  |  |  |  |  |  |  |
| misex3 |  |  |  |  |  |  |  |  |  |
| pdc |  |  |  |  |  |  |  |  |  |
| s298 |  |  |  |  |  |  | 203.9 | 1.98464e-07 | 8 |
| s38417 |  |  |  |  |  |  |  |  |  |
| s38584.1 |  |  |  |  |  |  |  |  |  |
| seq |  |  |  |  |  |  |  |  |  |
| spla |  |  |  |  |  |  |  |  |  |
| tseng |  |  |  |  |  |  |  |  |  |

# Analyse

Im Benchmark hat sich gezeigt, dass die Kräfteplatzierung VPR nicht nahekommt.

## Zufällige gegen Netzbasierte Initialplatzierung

# Quellen

1. Electrons Prozess-Model: <https://www.electronjs.org/docs/latest/tutorial/process-model>
2. Node SerialPort: <https://serialport.io/>
3. Electron-Store: <https://www.npmjs.com/package/electron-store>
4. Primereact: <https://primereact.org/>
5. Webpack: <https://webpack.js.org/>
6. Releasebau mit Docker electronuserland: <https://hub.docker.com/r/electronuserland/builder>
7. Electron-React-Boilerplate: <https://opencollective.com/electron-react-boilerplate-594>
8. Electron-Logo: <https://commons.wikimedia.org/wiki/File:Electron_Software_Framework_Logo.svg>