Heuristiken für das TechnologyMapping

Alexander Zorn

Geboren am 26. Mai 1996 in Bonn

25. Mai 2018

Bachelorarbeit Mathematik

Betreuer: Prof. Dr. Stephan Held

Zweitgutachter: YYYY YYYY

Forschungsinstitut für Diskrete Mathematik

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Inhaltsverzeichnis

1 Einleitung

Der zunehmende Gebrauch elektronischer Geräte verlangt nach immer leistungsfähigeren Computerchips. Ein solcher wenige Quadratzentimeter große Chip beherbergt bis zu mehreren Milliarden Transistoren, welche, durch Drähte verbunden, gemeinsam eine Logische Funktion errechnen. Das Chipdesign beschreibt die Aufgabe aus einer gegeben Logischen Funktion einen herstellbaren Chip zu entwerfen, welcher diese Funktion realisiert.

Mithilfe von, aus wenigen Transistoren konstruierten, Bauteilen (genannt Gates, z.B.: AND, OR, INV, OAI) lässt sich eine Logische Funktion nachbilden. Abbildung 1 (links) zeigt dies an einem kleinen Beispiel. Die Realisierung einer solchen Funktion ist jedoch nicht eindeutig, wie die in Abbildung 1 (links und rechts) gezeigte Nachbildung, beweist.

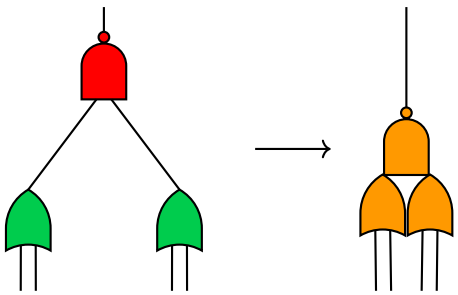

Abbildung 1: Zwei Realisierungen der Logischen Funktion $\neg((w \vee x) \wedge (y \vee z))$

Die Größe der Menge aller möglicher Baupläne (später Circuit) für eine Logische Funktion hängt maßgeblich von der Anzahl der zur Verfügung stehenden Bauteile, sowie von dem Aufbau der Funktion, ab. Es stellt sich heraus, dass im Allgemeinen eine Vielzahl möglicher Realisierungen einer Logischen Funktion existieren. Jedes Bauteil besitzt physikalische Eigenschaften an Größe, Geschwindigkeit (Delay) etc.. Somit besitzt auch jede Realisation solche Eigenschaften.

Ziel des TechnologyMapping ist es für eine Logische Funktion eine Realisierung zu finden, welche eine Kostenfunktion (bestehend aus den physikalischen Eigenschaften) optimiert. Die Wahl der Implementierung hat direkte Auswirkungen auf die Schnelligkeit, Größe und den Stromverbrauch des fertigen Chips. Hierbei geht das TechnologyMapping von einer bereits realisierten Logischen Funktion aus und baut diese um zu einer möglichst kostengünstigen Alternative um.

Der optimale mögliche Umbau lässt sich bei kleinen oder eingeschränkten gegebenen Bauplänen noch in akzeptabler Zeit finden. Die Lösung dieses Problem für allgemeine Baupläne und Kostenfunktionen ist jedoch ein NP vollständiges Problem. Aus diesem Grund entwickelt die folgende Arbeit eine Heuristik, welche für sehr (mehrere 10.000 Bauteile) große Baupläne in akzeptabler Zeit einen möglichst kostengünstigen Umbau ermöglicht.

2 Terminologie & grundlegender Algorithmus

2.1 grundlegende Definitionen

Es folgen ein paar grundlegende Definitionen zur Beschreibung des Problems.

Definition 2.1. Boolesche Variable und Funktion:
Eine boolesche Variable ist eine Variable mit Werten in $\{0,1\}$. Sei $n, m \in \mathbb{N}$. Eine boolesche Funktion ist eine Funktion $f : \{0,1\}^n \to \{0,1\}^m$ mit n inputs und m outputs.

Definition 2.2. Gate und Library:
Ein Gate g mit Eingangsgrad $n \in \mathbb{N}$ ist ein Tripel $(f_g, d_g, area_g)$. Hierbei sind $d_g, area_g \in \mathbb{R}_{\geq 0}$. Des Weiteren gilt f_g ist eine boolesche Funktion mit $f_g : \{0,1\}^n \to \{0,1\}$.
Eine Library L ist eine Menge von Gates und sei $fanin_{max} := max\{arity(g) | g \in L\}$.

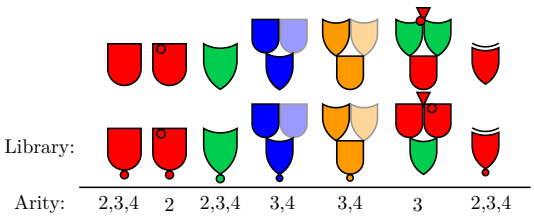

Abbildung 2: Beispiel einer Library

$area_g$ gibt die Größe des physikalischen Bauteils an und d_g beschreibt die Zeit die ein Signal braucht um von den inputs des Gates zu seinem Output zu gelangen. Dieser Wert lässt sich noch weiter differenzieren indem man $d_g \in \mathbb{R}^n$ wählt und somit Zeiten für jeden der Inputs angeben werden kann.

Definition 2.3. Circuit:
Ein Circuit ist ein gerichteter kreisfeier Graph (directed acyclic graph DAG) mit folgenden Eigenschaften. Jeder Knoten gehört zu einer der aufgelisteten Kategorien:

- **Input** Knoten mit Eingangsgrad Null.
- **Gates** mit mindestens einer eingehenden Kante und ausgehenden Kante. Diese korrespondieren zu der Definition oben mit dem Zusatz dass an jedem der Inputs optional ein Inverter liegen kann.
- **Outputs** mit genau einer eingehenden Kante und keiner ausgehenden.

Ein Gate mit mehr als einer ausgehenden Kante wird auch Highfanoutgate genannt.
Ein Circuit realisiert durch Verschachtelung der booleschen Funktionen seiner Gates ebenfalls eine boolesche Funktion.
Zwei Circuits heißen äquivalent, wenn sie die gleiche boolesche Funktion realisieren.

In einem Circuit lassen sich Teilgraphen durch ein Gate der Library austauschen. Voraussetzung für einen solchen Tausch ist, dass der veränderte Circuit äquivalent zu dem originalen ist. Dies sicher die folgenden Definitionen.

Definition 2.4. Match und Kandidat:
Sei g ein Gate in einem Circuit C. Ein (invertiertes) Match m ist ein Tupel (p_m, I_m, f_m, inv_m) welches folgendes enthält:

- Ein Gate p der Library
- Eine Menge X von Knoten aus der Circuit und eine Bijektion $f : X \to inputs(p)$
- Ein Funktion $inv : inputs(p) \to \{not\_inv, inv\}$

So dass der Circuit C', welcher durch den Austausch des Sub-Circuits von X bis g durch das Match (mit den durch inv definierten Invertern an den Inputs) entsteht, äquivalent zu C ist. Ein invertiertes Match auf g ist ein Match auf g mit einem Inverter an jedem seiner Outputs.
Ein (invertierter) Kandidat auf g besteht aus einem (invertierten) Match auf g und einem Kandidaten für jeden Input Knoten von g (welcher kein Input von C ist).

Definition 2.5. Circuit-Kandidat: Sei C ein Circuit mit Outputknoten Menge O. Eine Circuit-Kandidat K von C ist eine Menge von Kandidaten, sodass $\forall o \in O \, \exists! h \in K : h$ ist Kandidat von o und an jedem Knoten von C an dem sich mehrere Kandidaten überschneiden ist dasselbe Match gewählt.

Folgendes Beispiel visualisiert die vorherigen Definitionen.
BILD EINSETZEN

Ein Circuit-Kandidat C ist eine Möglichkeit den Circuit physikalisch zu realisieren. Wie bereits in der Einleitung bemerkt gilt es nun den besten Kandidaten auf C auszuwählen. Dafür ist ein Maß für Implementierungen von Circuits notwendig. Es folgen zwei geläufige Beispiele. In der Praxis (und im späteren Verlauf dieser Arbeit) wird in der Regel eine convex-Kombination aus beiden verwendet.

Definition 2.6. Area und Delay eines Kandidaten:
Sei C ein Circuit und K ein Circuit-Kandidat auf C. Dann gilt:

- $area(K) = \sum_{g \in gates(C)} (a_g + \sum_{i \in inputs(g)} \mathbb{1}_{\{inv_g(i) == inv\}} area_{inv})$
  wobei $area_{inv}$ die Größe eines Inverters ist.
- $AT(K) =$
  $\max_{k \in can(K)} \{ \max_{i \in inputs(k)} \{ d_{gate(k)} + \mathbb{1}_{\{inv_g(i) == inv\}} d_i + AT(inp\_can(k,i)) + d_{w(k,i)} \} \}$

Wobei $can(K)$ die Menge der Kandidaten von K sind und $inputs(k)$ sind die Inputknoten des Outputknoten des Kandidaten k. Des Weiteren ist d_i das Delay eines Inverters und $d_{w(k,i)}$ das Delay der Kante zwischen den Knoten k und i. $inp\_can(k,i)$ gibt den Kandidaten des i'ten Inputs von k zurück.

Das Delay (AT) gibt an wann das letzte Signal aus einem der Outputs des Circuit kommt.

2.2 Kern Algorithmus

Es folgt ein grundlegender Algorithmus, welcher auf eingeschränkten Circuits arbeitet, jedoch im weiteren Verlauf dieser Arbeit zu einer Heuristik für allgemeine sehr große Circuits erweitert wird.

(einfaches) Technology Mapping	
Instanz:	Circuit C ohne Highfanoutknoten, mit eindeutigem Output o, Library L mit beschränktem $fanin_{max}$
Aufgabe:	Finde einen Kandidaten K auf o, welcher die Arrivaltime/Area minimiert.

Algorithmus : (einfaches) Technology Mapping

```
Input : Circuit C kreisfrei mit finalem Output o, Library L
1 bester_kandidat[] ← ∅
2 bester_inv_kandidat[]← ∅
3 foreach Knoten v ∈ V(G) in topologischer Reihenfolge do
4     berechne alle (invertierten) Matches auf v
5     foreach Match m auf v do
6         Berechne besten Kandidaten mit m auf v
7         Update best_(inv)_kandidaten
8 Implementiere C entsprechend bester_kandidat[o]
```

Dieser geht in topologischer Reihenfolge durch die Knoten v des Graphen und berechnet alle Matche auf v. Diese Matche werden dann zu einem Kandidaten ergänzt. Dieser Schritt (6) lässt sich sehr schnell implementieren, da für jedes Match m die besten Kandidaten der Inputs von m bereits bekannt sind.

Ohne Highfanout-Knoten überschneiden sich diese nicht, und der beste Kandidat (inklusive der invertierten Verisonen) für jedes Match ist schnell gefunden. Von diesen wird der beste (in Bezug auf Area oder Arrivaltime) in Schritt 7 zu zur Liste der besten Kandidaten hinzugefügt.

Korollar 2.7. *Das (einfache) TechnologyMapping besitzt* $\mathcal{O}(|V(C)||L|fanin_{max})$*-Laufzeit*
Schritt 1 und 2 besitzen Laufzeit $\mathcal{O}(1)$*. Schritt 4 lässt sich in wielange braucht es ein zu checken ob ein gate der Library ein Match eines Knoten sein kann ? ? plus Begründung! Laufzeit anpassen. Schritt 6 ist wie bereits erwähnt schnell implementierbar, da für jeden der max* $fanin_{max}$ *Inputs der beste Kandidat verlinkt werden muss. Die Invertierten Matche werden nur gebarucht wenn der Input invertiert war. Somit braucht* $\mathcal{O}(fanin_{max})$*. Schritt 3 und 5 sind zwei verschachtelten Schleifen mit* $|V(C)|$ *und max* $|L|$ *Durchläufen.*
Daraus folgt eine Laufzeit von $\mathcal{O}(|V(C)||L|fanin_{max})$*.* □

3 Allgemeiner Algorithmus und Heuristik

3.1 Tradeoffprobleme

Der oben vorgestellte Algorithmus ist in der Lage den bestmöglichen Umbau eines eingeschränkten Circuits zu bezüglich Area oder Delay zu errechnen. Es existiert ein Tradeoff zwischen beiden Area und Delay. Dies hat zur Folge, dass ein möglichst kleiner Circuit im Allgemeinen sehr langsam ist und man bei einer sehr schnellen Lösung mit einem großen Platzverbrauch rechnen muss. In der Anwendung des TechnologyMapping ist jedoch weder ein sehr langsamer noch ein besonders grosser Circuit akzeptabel. Daraus folgt die Nachfrage nach einem Algorithmus, welcher in der Lage ist bezüglich einer Konvexkombination oder einer Schranke zu optimieren. Daraus ergeben sich die beiden folgenden Optimierungs-Probleme:
Hier beide Probleme einfügen

Diese diese Probleme sind äquvalent Beweis? oder verweis aus quelle

Dadurch ergibt sich folgende Problemstellung für den Algorithmus: An jedem Knoten v lässt sich jetzt nicht mehr nur der Kandidat speichern, welche die Kostenfunktion an v optimiert. Angenommen v liegt in einem sehr Delay kritischen Gebiet des Graphen, dann kann es sehr gut sein dass es für einen Nachfolger von v sehr günstig ist v vollständig auf Delay hin zu optimieren und seine Kosten zu verbessern, was jedoch nicht möglich ist wenn der einzige an v gespeicherte Kandidat dieser ist der die Kosten (Konvexkombination) an optimiert. diesen satz verbessern!
Die Kosten eines Kandidaten k sind somit nicht $\lambda AT(k) + (1 - \lambda) area(k)$, sondern das Tupel $(AT(k), area(k))$. Es gibt jedoch eine Klasse von Kandidaten, welche nicht gespeichert muss. Dazu folgende Definition

Definition 3.1. (dominierte Kandidaten)
Seien k_1, k_2 Kandidaten desselben Knotens. Dann wird k_1 von k_2 dominiert, wenn gilt:
$AT(k_1) < AT(k_2) und area(k_1) \leq area(k_2)$
$AT(k_1) \leq AT(k_2) und area(k_1) < area(k_2)$

Eine optimale Lösung des TechnologyMapping verwendet offenbar (in einem Korollar beweisen ?) nur nicht-dominierte Kandidaten, woraus folgt, dass nur diese während der Ausführung des Algorithmus gespeichert werden müssen.
Die Menge der noch bleibenden Kandidaten lassen sich in sogenannten Tradeoff-Kurven speichern (siehe Abbildung ?? Abbildung hinzufügen.. kann ich Lucas verwenden ?). Welche jeden Kandidaten zweidimensional anhand seiner Kosten erfasst.

Next things todo: Algorithmus erweitern und dann auf die Filterung mit Buckets eingehen inclusive der funktionierenden epsilon betrachtung -¿ ohne beweis mit erklärung

ZSFG:

1. Vorstellung der Probleme: die da wären :

-Tradeoffprobleme -¿ mehrere Kandidaten müssen gespeichert werden. Einführung der Tradeoff Kurven und Bucket Filterungen (wir sind noch in Bäumen dies lässt sich also noch in den Alg einbauen und beweisen)

-Highfanoutgates -¿ Einführung von Klassen widerrum erweitern des TM algos

=¿ bei konstantem k gibt es zu diesem Punkt noch einen FPTAS -¿ besonder hilfreich auf schlechteste Wege pfade kann auch erst später erwähnung finden

-required ATs einführen und sagen dass sie aktuell noch äquivalent sind

-multiple outputs -¿ sagen, dass optimieren auf AT nicht mehr Funktioniert da es nicht nur eine AT gibt. man könnten die latest AT verbessern, das ist jedoch nicht das was man möchte. neue Kosten optimierung mit RAts definieren. Des weiteren nicht vergessen, dass über outputs nicht gemacht werden darf

-reduandant gates -¿ soll das hier hin ?

=¿ Algorithmus verallgemeiner und erste Heuristik bauen, welche auf Area und RATs hin optimiert

4 Präprozessing zusätliche Addons

-auseinanderbauen von gates : Beispiele und einegehen auf vor und nachteile genauerer bezug in der Laufzeitanalyse

-errechnen kleiner optimal gelöster häufig vorkommender Instanzen -¿ unabh. von dem auseinanderbauen. -¿ lässt sich beweisen dass eine auseinandergebaute instanz sich in das bestmögliche Machting matchen lässt ?

5 Weitere Optimierungskriterien

-Vt Optimierung -¿ Optimierung bezüglich Power

-Layer assignment -¿ sehr kurz und grobe übersicht kommt später noch auf die TODOs

6 Version der Heuristik, welche obige Kriterien beherzigt

7 Laufzeitanalyse

8 Fazit und Ausblick