

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	2
2	Terminologie & grundlegender Algorithmus	3
2.1	grundlegende Definitionen	3
2.2	Kern Algorithmus	5
3	Allgemeiner Algorithmus und Heuristik	7
3.1	Tradeoffprobleme	7

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 gegebenen 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.

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 TehnologyMapping 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 TehnologyMapping 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.

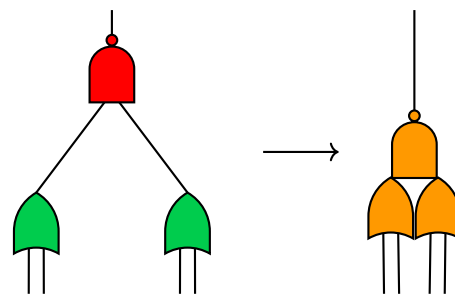


Abbildung 1: Zwei Realisierungen der Logischen Funktion $\neg((w \vee x) \wedge (y \vee z))$

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 $f : \{0, 1\}^n \rightarrow \{0, 1\}^m$ 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 $f_g : \{0, 1\}^n \rightarrow \{0, 1\}$.

Eine Library L ist eine Menge von Gates und sei $fanin_{max} := \max\{arity(g) | g \in L\}$.

$area_g$ 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.

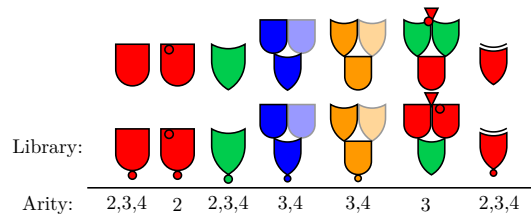


Abbildung 2: Beispiel einer Library

Definition 2.3. Circuit:

Ein Circuit ist ein gerichteter kreisfreier 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 $f : X \rightarrow inputs(p)$
- Ein Funktion $inv : inputs(p) \rightarrow \{not_inv, inv\}$

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 Arrivaltme/Area minimiert.

Algorithmus : (einfaches) Technology Mapping

Input : Circuit C kreisfrei mit finalem Output o , Library L

- 1 $best_kandidat[] \leftarrow \emptyset$
 - 2 $best_inv_kandidat[] \leftarrow \emptyset$
 - 3 **foreach** Knoten $v \in V(G)$ 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 $best_kandidat[o]$
-

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 Arrivalttime) in Schritt 7 zu zur Liste der besten Kandidaten hinzugefügt.

Korollar 2.7. *Das (einfache) TehnologyMapping 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 $max |L|$ Durchlaufen.*

Daraus folgt eine Laufzeit von $\mathcal{O}(|V(C)||L|fanin_{max})$.

□

3 Allgemeiner Algorithmus und Heuristik

3.1 Tradeoffprobleme

a

AB HIER LUCAS VORLAGE

Definition 3.1. Library: Eine Library ist eine Menge L von Gates (boolesche Funktionen) mit zwei Abbildungen $d, \text{area} : L \rightarrow \mathcal{R}_{\geq 0}$, die jedem Gate sowohl eine Verzögerung d_l , als auch eine Fläche area_l zuordnen.

Definition 3.2. Circuit: Ein Circuit C auf der Library L ist ein zusammenhängender gerichteter azyklischer Graph (DAG), bei dem jeder Knoten einer dieser 3 Arten entspricht:

- einem Inputknoten ohne eingehende Kanten
- einem Gate aus L mit ≥ 1 eingehenden und ≥ 1 ausgehenden Kanten
- einem Outputknoten ohne ausgehende Kanten

Jeder Gateknoten kann an jeder seiner eingehenden Kanten einen Inverter vorschalten. Für einen Knoten v sei $\text{fanin}(v)$ die Zahl seiner eingehenden, $\text{fanout}(v)$ die Zahl seiner ausgehenden Kanten. Knoten mit $\text{fanout}(v) > 1$ heißen Highfanoutknoten. Wir betrachten vorerst nur Circuits mit exakt einem Outputknoten.

Definition 3.3. cone: Für einen Knoten g aus einem DAG S bezeichne

$$\text{cone}(g) := S[V \cup \{g\}], V = \{v \in V(S) : \exists v\text{-}g\text{-Weg in } S\}$$

Sowie für eine Knotenmenge G sei $\text{cone}(G) := S[\cup_{g \in G} V(\text{cone}(g))]$

Algorithmus : TechnologyMapping auf einer Arboreszens

Input : Circuit C kreisfrei mit finalem Output o , Library L
verfügbarer Gates

```

1 bester_kandidat[]  $\leftarrow \emptyset$ 
2 bester_inv_kandidat[]  $\leftarrow \emptyset$ 
3 foreach Knoten  $n \in V(G)$  in topologischer Ordnung do
4   berechne alle (invertierte) Matches auf  $n$ 
5   foreach Match  $m$  auf  $n$  do
6     Berechne besten Kandidaten mit  $m$  auf  $n$ 
7     Update best_(inv)_kandidaten
8  $\text{best\_final} \leftarrow \text{best\_kandidat}[o]$ 
9 Implementiere  $C$  entsprechend  $\text{best\_final}$ 
```
