

Algorithmen für das Technology Mapping

Alexander Zorn

Geboren am 26. Mai 1996 in Bonn

17. Juli 2018

Bachelorarbeit Mathematik

Betreuer: Prof. Dr. Stephan Held

Zweitgutachter: Prof. Dr. Dr. h.c. Bernhard Korte

FORSCHUNGSINSTITUT FÜR DISKRETE MATHEMATIK

MATHEMATISCH-NATURWISSENSCHAFTLICHE FAKULTÄT DER
RHEINISCHEN FRIEDRICH-WILHELMS-UNIVERSITÄT BONN

Inhaltsverzeichnis

1	Einleitung	2
2	Terminologie und Kern-Algorithmus	3
2.1	Grundlegende Definitionen	3
2.2	Kern-Algorithmus	7
3	Entwicklung von FPTAS und Heuristik	8
3.1	Tradeoffprobleme	9
3.2	Highfanoutknoten	11
3.3	Zu lange Kanten	14
3.4	Teilweise überflüssige Subcircuits	14
3.5	Match-Probleme	16
3.6	Kandidaten-Probleme	19
3.7	FPTAS	23
3.8	Heuristik	25
4	Erweiterungen der Heuristik	27
4.1	DAGs mit mehreren Outputs	27
4.2	Premapping von Highfanoutknoten	30
4.3	Preprocessing	34
4.4	Weitere Optimierungskriterien	34
5	Resource Sharing	36
5.1	Definition und Lösungsansatz des Resource Sharing Problems	36
5.2	Resource Sharing und Technology Mapping	37
6	Qualitäts- und Laufzeit-Analyse	41
6.1	Struktur realer Instanzen	42
6.2	Analyse der Ergebnisse	43
6.3	Laufzeitanalyse	44
6.4	Vergleich von Güte und Laufzeit	45
7	Fazit und Ausblick	45
8	Literaturverzeichnis	46

1 Einleitung

Die ständig komplexer werdenden Anforderungen an die Informationstechnologie verlangen nach immer leistungsfähigeren Computerchips.

Die sich hieraus ergebenden Anforderungen an die Chipentwicklung wurden bereits 1965 von Gordon Moore in "Moore's law"[Moo65] beschrieben; hiernach ist regelmäßig eine Verdopplung der Integrationsdichte, der Anzahl von Transistoren pro Flächeneinheit, erforderlich und auch bisher technisch realisierbar. Nach [HNKF18] wird dies jedoch in absehbarer Zeit nicht mehr möglich sein, wodurch das Thema dieser Arbeit weiter an Bedeutung gewinnt.

Derzeit beträgt die Dichte an Transistoren, die zu einem integrierten Schaltkreis auf einem Chip miteinander verbunden sind, bis zu acht Milliarden. Diese sind so angeordnet, dass sie gemeinsam eine vorgegebene logische Funktion errechnen können. Die Aufgabe des Chip-Designs ist es, einen herstellbaren Chip zu entwerfen, der eine vorgegebene logische 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 (linker Teil) zeigt dies an einem kleinen Beispiel. Die Realisierung einer solchen Funktion ist jedoch nicht eindeutig, wie Abbildung 1 zeigt.

Die Größe der Menge aller möglichen Baupläne (später Circuits) für eine logische Funktion hängt maßgeblich von den auf dem Chip zur Verfügung stehenden Bauteilen, sowie von dem Aufbau der Funktion, ab. Hierdurch bedingt ergeben sich eine Vielzahl von Möglichkeiten eine logische Funktion zu realisieren.

Jedes Bauteil hat spezifische physikalische Eigenschaften an Größe, Geschwindigkeit (Delay) etc.. Demnach hat auch jeder Circuit entsprechende Eigenschaften.

Ziel des Technology Mappings ist es, für eine gegebene logische Funktion eine Realisierung zu finden, welche eine Kostenfunktion (bestehend aus den physikalischen Eigenschaften) optimiert. Die Wahl der Lösung hat direkte Auswirkungen auf die Schnelligkeit, Größe und den Stromverbrauch des fertigen Chips. Hierbei geht das Technology Mapping von einer bereits realisierten logischen Funktion aus und baut diese zu einer möglichst kostengünstigen Variante um.

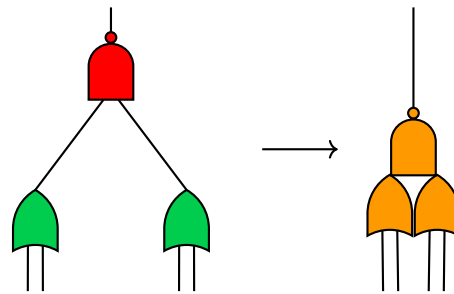


Abbildung 1: Zwei Realisierungen der logischen Funktion $\neg((w \vee x) \wedge (y \vee z))$

Der optimale mögliche Umbau lässt sich bei kleinen oder eingeschränkten gegebenen Bauplänen noch in kurzer Zeit finden. Die Lösung dieses Problems 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 möglichst kurzer Zeit einen kostenoptimierten Umbau approximiert.

Aufbauend auf den Arbeiten von Keutzer [Keu87] und [Elb17] wird ausgehend von einem Kern-Algorithmus auf eingeschränkten Instanzen in Kapitel 2 ein polynomielles Approximationsschema für einen beliebigen Approximationsfaktor $\varepsilon > 0$ und daraus eine Heuristik in Kapitel 3 entwickelt. Anschließend wird diese Heuristik in Kapitel 4 an möglichst allgemeine Instanzen und weitere reale Gegenbeispiele angepasst. Mithilfe des Resource Sharing Algorithmus wird in Kapitel 5 die entwickelte Heuristik Teil einer allgemeinen Logik-Optimierung von Circuits. Abschließend wird in Kapitel 6 die Implementierung der vorgestellten Heuristik ausführlich analysiert.

2 Terminologie und Kern-Algorithmus

2.1 Grundlegende Definitionen

Es folgen ein paar grundlegende Definitionen, die für die Beschreibung des Problems erforderlich sind.

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 (arity) $n \in \mathbb{N}$ ist ein Tripel $(f_g, d_g, area_g)$. Hierbei sind $d_g, area_g \in \mathbb{R}_{\geq 0}$. Des Weiteren ist f_g eine boolesche Funktion mit $f_g : \{0, 1\}^n \rightarrow \{0, 1\}$.

Eine Library L ist eine Menge von Gates und sei

$$fanin_{max} := \max\{arity(g) | g \in L\}.$$

Abbildung 2 gibt einen Überblick über eine praktisch genutzte Library. Jedes dieser Gates liegt, in diesem Beispiel, auch in einer invertierten Version vor (zweite Reihe der Library). Des Weiteren ist es für gewöhnlich der Fall dass von einem Gatetyp mehrere Varianten bezüglich der Anzahl seiner Inputs vorhanden sind. Die Zeile mit dem Titel Arity gibt einen Überblick, darüber welche Variationen auf Chips üblich sind.

Wie gerade definiert realisiert jedes Gate eine boolesche Funktion. In der entsprechenden Abbildung auf Zeile der Abbildung ist für jedes Gate (einmal nicht invertiert und einmal invertiert) die diese Funktion, für 2, 3 oder 4

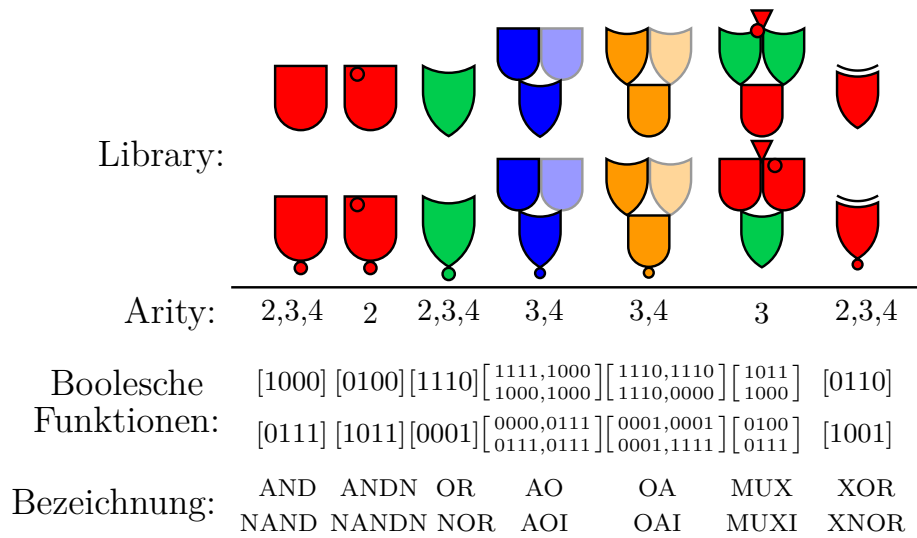


Abbildung 2: Beispiel einer Library

Inputs, abbildet. Die geschieht in Folgendem Format: $[1, \dots, 2^{\text{arity}(g)}]$ gibt für ein Gate g an der Stelle $i \in \{1, \dots, 2^{\text{arity}(g)}\}$ die Wahrheitsbelegung des Outputs bei Inputbelegung i_2 (dargestellt im Binärschreibweise). Es ist üblich an den Namen eines Gates die Arity zu knüpfen. Beispiel: AND2.

Die 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 $d_g \in \mathbb{R}^{\text{arity}(g)}$ wählt und somit Zeiten für jeden der Inputs angeben werden können. Hierauf wird jedoch erst in Kapitel 4.4 eingegangen.

Wenn die Signale der Inputs nicht zur selben Zeit ankommen, wird, falls nicht anderes angegeben, gewartet, bis das letzte Signal das Gate erreicht.

Definition 2.3. Circuit:

Ein Circuit ist ein gerichteter kreisfreier Graph (directed acyclic graph DAG) mit der Eigenschaft: Jeder Knoten gehört zu einer der aufgelisteten Kategorien:

- **Inputs** mit Eingangsgrad Null.
- **Gates** mit mindestens einer eingehenden und mindestens einer ausgehenden Kante. Diese Knoten entsprechen den Gates (Definition 2.2) und darüber hinaus gilt: An jeder eingehenden Kante kann ein Inverter liegen.
- **Outputs** mit genau einer eingehenden und keiner ausgehenden Kante.

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.

Hierbei ist es möglich, dass das Signal eines Gates sowohl direkt in einen Output-Knoten sowie noch weiter verarbeitet wird. Weitere Informationen über den Umgang mit mehreren Outputs befindet sich in Kapitel 4.1.

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 Original ist. Dies wird in den folgenden Definitionen formalisiert.

Definition 2.4. Match und Kandidat:

Sei g ein Gate in einem Circuit C . Ein (invertiertes) Match m auf g ist ein Tupel (p_m, X_m, f_m, inv_m) welches folgendes enthält:

- Ein Gate p_m der Library
- Eine Menge X_m von Knoten aus dem Circuit und eine Bijektion $f : X_m \rightarrow inputs(p_m)$
- Eine Funktion $inv_m : inputs(p_m) \rightarrow \{not_inv, inv\}$

sodass der Circuit C' , welcher durch den Austausch des Subcircuits von X_m bis g durch das Match (mit den durch inv_m 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 Tupel, welches die Äquivalenzbedingung evtl. nicht erfüllt wird potenzielles Match genannt.

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 Output-Knoten-Menge O . Ein Circuit-Kandidat K von C ist eine Menge von Kandidaten, sodass für jeden Output aus C genau ein Kandidat in O vorhanden ist und je zwei Kandidaten aus O bilden an überschneidenden Knoten das selbe Match.

Abbildung 3 visualisiert die vorherigen Definitionen.

Ein Circuit-Kandidat auf einem Circuit C beschreibt eine mögliche Realisierung der C zugrunde liegenden Funktion. Wie bereits in der Einleitung dargestellt gilt es nun den besten Circuit-Kandidaten auf C zu finden. Dafür

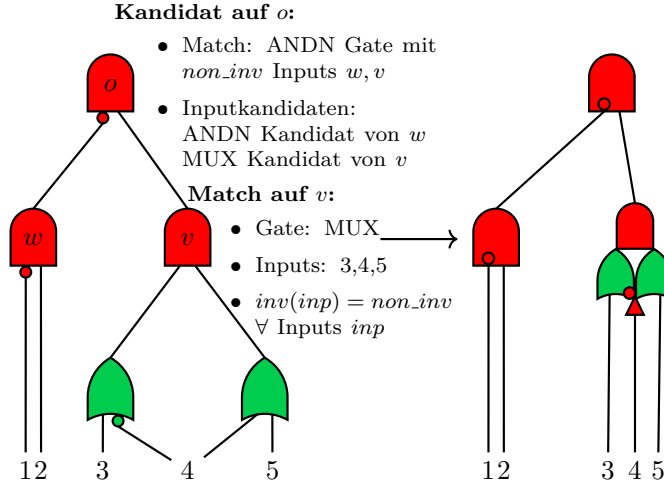


Abbildung 3: Beispiel eines Kandidaten und eines Matches

ist ein Maß notwendig, durch welches diese Kandidaten bewertet werden. Es folgen zwei praxisrelevante Beispiele einer solchen Funktion. In der Praxis und in dieser Arbeit wird hauptsächlich eine Konvexkombination aus beiden verwendet.

Wie bereits in der Gate Definition erwähnt, braucht ein Signale eine gewisse Zeit (Delay) um das Gate zu passieren. Die gilt ebenfalls für jede Kante eines Circuits. Ahängig von dem Ort der Endknoten und der Lage auf dem Chip, besitzt jede Kante einen Delay-Wert, welcher angibt, wie lange ein Signal benötigt um die Kante zu passieren.

Des Weiteren ist für die Inputs des Circuits eine Arrivalttime (AT) gegeben, welche angibt zu welchem Zeitpunkt das Signal an diesem Input ankommt. Von den Inputs ausgehend, lässt sich die Arrivalttime, mit Hilfe der Delay-Werte, durch den gesamten Circuit propagieren und entsprechende Werte für die Outputs angeben. Hierbei gilt, dass die Arrivalttime eines Knotens die Ankunftszeit des letzten Signals der Inputs am Output des Knotens. Dies lässt sich auch auf einen Kandidaten übertragen. Dazu folgende Definition:

Definition 2.6. Area und Delay eines Kandidaten:

Sei C ein Circuit und K ein Circuit-Kandidat auf C , dann gilt:

- $area(K) = \sum_{g \in gates(C)} (area_g + \sum_{i \in inputs(g)} \mathbb{1}_{inv_g(i)} area_{inv})$
- $AT(K) = \max_{k \in can(K)} \{ \max_{i \in inputs(k)} \{ d_{gate(k)} + \mathbb{1}_{inv_g(i)} d_{inv} + AT(inp_can(k, i)) + d_{w(k, i)} \} \}$

Wobei $can(K)$ die Menge der Kandidaten von K ist, $area_{inv}$ die Größe eines Inverters und $inputs(k)$ die Input-Knoten des Output-Knoten des Kandida-

ten k sind. Des Weiteren ist d_{inv} 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.

2.2 Kern-Algorithmus

Es folgt ein grundlegender Algorithmus, welcher auf nachfolgend dargestellten 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 bester_kandidat[]  $\leftarrow \emptyset$ 
2 bester_inv_kandidat[]  $\leftarrow \emptyset$ 
3 foreach Knoten  $v \in V(C)$  in topologischer Reihenfolge do
4   berechne_alle_(invertierten)_Matche( $v$ )
5   foreach Match  $m$  auf  $v$  do
6     Berechne_besten_Kandidaten( $m, v$ )
7     Update_best_(inv)_kandidaten
8 Implementiere  $C$  entsprechend bester_kandidat[ $o$ ]

```

Ein Technology Mapping (TM) Algorithmus liefert in der Regel nicht die bestmögliche Implementierung der C zugrunde liegenden logischen Funktion. Dies veranschaulicht Abbildung 4. In diesem Beispiel ist die zugrundeliegende Funktion konstant und somit könnte man auf alle Gates verzichten. Dies ist jedoch mit den bisher eingeführten Möglichkeiten des Technology Mappings nicht möglich.

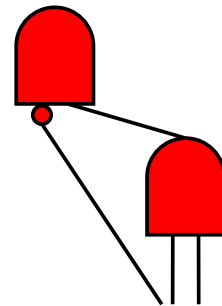


Abbildung 4: Ein Circuit, dessen boolesche Funktion $f = 0$ ist

Korollar 2.7. *Das einfache Technology Mapping liefert, bezüglich der vorgestellten Möglichkeiten des Matchens und der Kandidatenbildung, den bestmöglichen äquivalenten Circuit.*

Beweis:

Der Algorithmus geht in topologischer Reihenfolge durch die Knoten v des Graphen und berechnet alle Matche auf v . Diese werden dann zu einem Kandidaten ergänzt. Ohne Highfanoutknoten überschneiden sich diese nicht. Für jeden Knoten und jedes Match gibt es nur einen Kandidaten zur Auswahl, da für die Inputs des Matches jeweils nur ein Kandidat gespeichert wurde. An jedem Knoten wird nur das Match (mit dem dazugehörigen Kandidaten) gespeichert, welches die Kosten optimiert.

Es bleibt zu zeigen, dass, angenommen, dass für alle Knoten mit kleinerem topologischen Rang als $\text{rang}(v)$ der bestmögliche Kandidat bereits gespeichert ist, so wird, wie soeben dargestellt, auch für v der schnellste bzw. kleinste Kandidat k gespeichert.

Angenommen es gibt einen besseren Kandidaten k' , als k , welcher von dem Algorithmus gespeichert wurde. Sei k'' der Kandidat, welcher dasselbe Match wie k' benutzt und die besten Input Kandidaten. Da k'' die besten Input Kandidaten benutzt, ist er mindestens so schnell (bzw. klein) wie k' . k ist jedoch ebenfalls mindestens so kostengünstig wie k'' (andernfalls hätte der Algorithmus k'' , k vorgezogen). Dies ist ein Widerspruch zur Annahme. \square

Korollar 2.8. *Der Algorithmus für das (einfache) Technology Mapping besitzt $\mathcal{O}(|V(C)|^3|L|^2)$ -Laufzeit*

Beweis: Schritt 1 und 2 besitzen Laufzeit $\mathcal{O}(1)$. Schritt 4 lässt sich, aufgrund von einem beschränkten fanin_{\max} und ohne Highfanoutgates, in $\mathcal{O}(|V(C)|^2|L|)$ errechnen. Der Beweis dieser Aussage befindet sich in Kapitel 3.5.

Der Schritt 6 ist schnell implementierbar, da für jeden der $\max \text{fanin}_{\max}$ Inputs der beste Kandidat bereits errechnet wurde und somit nur verlinkt werden muss. Ein invertiertes Match wird nur gebraucht, wenn der korrespondierende Input des darüber liegenden Gates invertiert ist. Schritt 6 lässt sich somit in $\mathcal{O}(\text{fanin}_{\max})$ realisieren. Schritt 3 und 5 sind zwei verschachtelte Schleifen mit $|V(C)|$ und $\max |L|$ Durchläufen.

Daraus folgt eine Laufzeit von $\mathcal{O}(|V(C)|^3|L|^2)$. \square

3 Entwicklung von FPTAS und Heuristik

In diesem Kapitel wird ein Approximationsalgorithmus (ein FPTAS) für allgemeinere Instanzen und Zielfunktionen des Technology Mappings vorgestellt. Da auch dieser, um eine polynomielle Laufzeit zu erreichen, noch einige Einschränkungen an gegebene Circuits hat, welche für reale Instanzen eines Chips nicht gelten und sich herausstellen wird, dass das Technology Mapping bereits mit einer Konvexkombination als Kostenfunktion NP-vollständig ist, wird auf Basis des Approximationsalgorithmus eine Heuristik entworfen.

3.1 Tradeoffprobleme

Der in Kapitel 2.2 vorgestellte Algorithmus ist in der Lage den bestmöglichen Umbau eines eingeschränkten Circuits bezüglich Area oder Delay zu errechnen.

Es existiert ein Tradeoff zwischen Area und Delay. Dies hat zur Folge, dass ein möglichst kleiner Circuit in der Regel sehr langsam ist und bei einem schnellen Circuit ein großer Platzverbrauch zu erwarten ist. Bei der Lösungsentwicklung mittels Technology Mapping ist jedoch weder ein sehr langsamer noch ein besonders großer Circuit akzeptabel.

Daraus folgt die Suche nach einem Algorithmus, welcher in der Lage ist, Circuits bezüglich einer Konvexkombination oder einer Schranke zu verbessern. Hierbei ergeben sich die beiden folgenden Optimierungsprobleme:

TECHNOLOGY MAPPING MIT KONVEXKOMBINATION

- Instanz:** Circuit C , mit einem Output, Library L mit beschränktem $fanin_{max}$, $|L|$ beschränkt und Tradeoffparameter $\lambda \in [0, 1]$.
- Aufgabe:** Finde einen Circuit-Kandidaten K auf C , welcher $\lambda AT(K) + (1 - \lambda)area(K)$ minimiert.

TECHNOLOGY MAPPING MIT ARRIVALTIMESCHRANKE

- Instanz:** Circuit C , mit einem Output, Library L mit beschränktem $fanin_{max}$, $|L|$ beschränkt und Arrivaltimeschranke A_{max} .
- Aufgabe:** Finde den kleinsten Circuit-Kandidaten K auf C , für den $AT(K) \leq A_{max}$ gilt, oder entscheide, dass für jeden Circuit-Kandidaten K bereits $AT(K) > A_{max}$ gilt.

In Kapitel 4.1, werden diese Problemstellungen auf Circuits mit mehreren Outputs erweitert.

Beide Probleme sind gleich schwer und lassen sich mit dem noch vorzustellenden FPTAS lösen. Eine Beschreibung, wie sich das Technology Mapping mit Arrivaltimeschranke mit dem FPTAS lösen lässt, findet sich in [Elb17]. Da wie im weiteren Verlauf dieser Arbeit dargestellt, die Arrivaltime nicht der einzige zu beachtende Faktor bei der Geschwindigkeitsoptimierung ist, wird im Folgenden nur noch die Konvexkombination behandelt.

Bei der Implementierung eines Algorithmus für die beiden Optimierungsprobleme ergibt sich folgende Herausforderung:

Angenommen, an jedem Knoten v würde, wie im Kern-Algorithmus, nur derjenige Kandidat gespeichert werden, welcher die Kostenfunktion an v optimiert; dann kann nicht mehr garantiert werden, dass beim Errechnen

der Kandidaten für den Output, der für ihn optimale Kandidat noch vorhanden ist. Beide Inputs getrennt nach der Kostenfunktion zu optimieren, garantiert also nicht das optimale Ergebnis.

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; hierzu die folgende Definition.

Definition 3.1. (dominierte Kandidaten)

Seien k_1, k_2 Kandidaten desselben Knotens. Dann wird k_1 von k_2 dominiert, wenn mindestens eine der folgenden Bedingungen erfüllt ist:

- $AT(k_1) < AT(k_2)$ 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 Technology Mappings verwendet nur nicht-dominierte Kandidaten. Angenommen dies wäre nicht der Fall, dann ließe sich durch Ersetzen eines dominierten durch einen nicht dominierten Kandidaten eine bessere Lösung erzielen; dies ist jedoch ein Widerspruch zu Optimalität der Lösung.

Daraus 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 (s. Abb. 5), welche jeden Kandidaten zweidimensional anhand seiner Kosten erfasst.

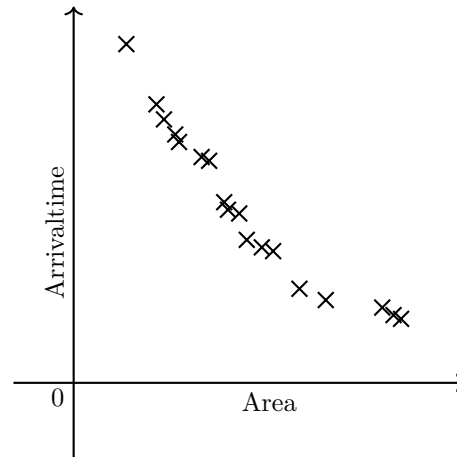


Abbildung 5: Ordnen der Kandidaten in einer Tradeoffkurve

Die beiden vorgestellten Probleme sind NP-Vollständig. Ein Beweis dieser Aussage und genauere Informationen zur Einordnung der Schwere dieser Probleme finden sich in [KR89] und [Elb17].

Daraus folgt, dass sich ab diesem Punkt wahrscheinlich kein polynomieller Algorithmus für das Technology Mapping finden lässt, welcher, hinsichtlich der vorgestellten Operationen des Technology Mappings, den kostengünstigsten Circuit-Kandidaten liefert.

Da bei zwei nicht dominierenden Kandidaten nicht eindeutig ist, welcher der Bessere ist, wird eine Vielzahl von Kandidaten an jedem Knoten gespeichert.

Dies zeigt sich in einem exponentiell großen Speicherbedarf.

3.2 Highfanoutknoten

Der beschriebene Kern-Algorithmus arbeitet nur auf Circuits, in denen keine Highfanoutknoten existieren. Diese kommen auf einem realen Chip jedoch sehr häufig vor; ca. 30% der Knoten sind Highfanoutknoten. Eine Analyse über die Auswirkungen von der Anzahl der Highfanoutknoten auf die Laufzeit des Algorithmus befindet sich Kapitel 6.

Es ist möglich, einen Circuit in kleinere Subcircuits zu unterteilen, welche solche Highfanoutknoten nicht beinhalten. Diese Subcircuits werden einzeln mit dem Algorithmus (sehr schnell) optimiert und daraufhin zu einem C äquivalenten Circuit C' zusammengesetzt. Diese Vorgehensweise findet sich ausführlich in [Keu87] wieder. Abbildung 6 stellt diesen Ansatz einer Heuristik dar. Der Anteil an Highfanoutknoten ist auf den mir vorliegenden Chips so groß, dass eine Vielzahl sehr kleiner Subcircuits entstehen, woraus folgt, dass die Möglichkeiten des Technology Mappings sehr eingeschränkt werden. Aus diesem Grund werde ich diesen Weg einer Heuristik nicht weiter verfolgen.

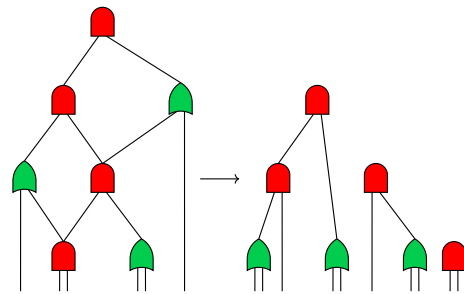


Abbildung 6: Unterteilen eines Circuit in Highfanoutfreie Subcircuits

Bei der Implementierung von Highfanoutknoten besteht die größte Herausforderung darin, dass bei der Konstruktion des äquivalenten Circuits die eingebauten Kandidaten aller Nachfolger eines Highfanoutknoten v an v übereinstimmen müssen.

Daraus folgt, dass bei der Wahl eines Kandidaten für einen Knoten w , die Input-Kandidaten von w , nicht unabhängig voneinander gewählt werden können.

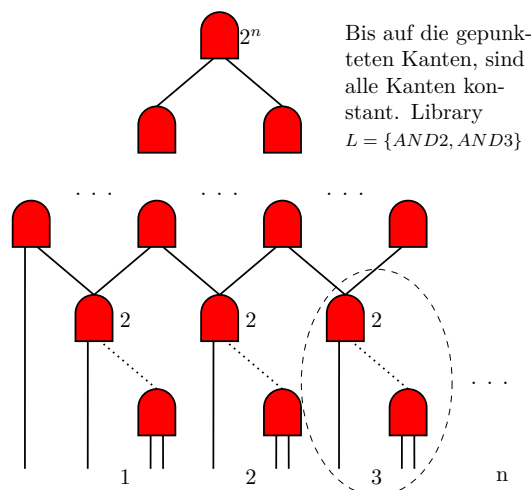


Abbildung 7: Exponentiell viele Kandidaten bereits bei sehr eingeschränkter Library

Abbildung 7 zeigt zudem eine weitere Herausforderung für die Implementierung auf. Die Anzahl der zu speichernden Kandidaten kann exponentiell bezüglich $|V(C)|$ sein. Zur Lösung der ersten Herausforderung helfen die folgenden Definitionen:

Definition 3.2. Cone eines Knoten:

Sei C ein Circuit und v ein Knoten von C , dann sei die Cone von v :

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

Die durch die $\text{cone}(v)$ berechnete logische Funktion wird die **bis v berechnete Funktion** genannt.

Definition 3.3. Offene Knoten:

Sei C ein Circuit und $v, w \in V(C)$, dann heißt w offener Knoten von v , wenn folgendes gilt:

- $w \in \text{cone}(v) \setminus \{v\}$
- $|\delta^+(w)| \geq 2$
- $\exists o \in V(C) \setminus \text{cone}(v) : \exists \text{ w-o-Weg in } C \text{ ohne } v$

Somit ist die Menge der offenen Knoten eines Circuit Knoten v , die Menge aller Highfanoutknoten w , von welchen aus man sowohl v als auch einen Knoten außerhalb der Cone von v erreichen kann. In dieser Menge ist v selber nicht enthalten. Die offenen Knoten von v sind gerade die Highfanoutknoten, welche durch die Kandidaten eines Knotens außerhalb von $\text{cone}(v)$ verändert werden können. Alle Kandidaten von Knoten mit Ausgangsgrad 1 und dieser Eigenschaft, sind durch den Nachfolger-Kandidaten (welcher auch zu einem offenen Knoten gehören muss), bereits eindeutig definiert.

Abbildung 8 visualisiert die vorangegangenen Definitionen. Hierbei entsprechen die mit o markierten Knoten den offenen Knoten.

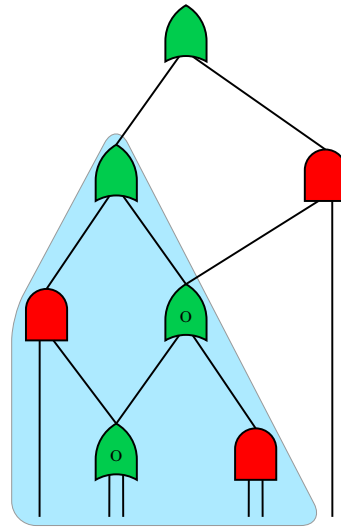


Abbildung 8: Visualisierung der Definitionen 3.2 und 3.3

Definition 3.4. Klasse eines Kandidaten:

Sei k ein Kandidat auf einem Knoten v und O die Menge der offenen Knoten von v . Die Klasse $class(k)$ ist eine Abbildung, welche jedem Element $w \in O$ den durch k festgelegten Kandidaten auf w zuordnet.

Hiernach lassen sich zwei Kandidaten k_1, k_2 eines Knoten v mit $class(k_1) \neq class(k_2)$ nicht miteinander vergleichen. Dies gilt auch für den Fall, wenn k_2 von k_1 dominiert wird, denn es ist möglich, dass dies zwar an der Stelle v gilt, jedoch nicht an allen offenen Knoten von v . Daraus folgt, würde man k_2 löschen, so löscht man evtl. den besten Kandidaten des Outputs von C . Um daher mit Highfanoutknoten arbeiten zu können, werden für jeden Knoten v und jede Klasse von v alle nicht dominierten Kandidaten gespeichert. Dann ist der noch verbleibende beste Kandidat des Outputs die beste Lösung. Dies führt jedoch zu einem nur exponentiell beschränkten Speicheraufwand.

Zur Speicherung der Kandidaten wird für jede Klasse eines Knotens eine Tradeoff-Kurve angelegt.

3.2.1 Klonen

Die Input-Kandidaten eines Kandidaten müssen aus nachfolgend dargestelltem Grund an deren offenen Knoten übereinstimmen:

Angenommen dies wäre nicht der Fall dann würden offene Knoten evtl. mehrere Male mit verschiedenen Kandidaten gebaut. Dieser Vorgang wird auch Klonen genannt. Dies kann von Vorteil sein, wenn zum Beispiel ein offener Knoten Teil eines sowohl sehr Delay- als auch sehr Area kritischen Gebietes ist. Dann würde einmal ein schneller und einmal ein sehr kleiner Kandidat realisiert. Dies führt in der Regel jedoch zu einem deutlich erhöhten Platzverbrauch und es werden mehr Kanten gebraucht, was zu vermeiden ist. Der erhöhte Verbrauch von Kanten bringt höhere Routing Kosten mit sich, welche im Technology Mapping jedoch nicht berücksichtigt werden. Um zu verhindern, dass nicht beachtete Ressourcen übermäßig verbraucht werden, ist das Klonen in den vorgestellten Algorithmen nicht erlaubt und wird durch die Klassen und die Routine, welche beim Verknüpfen von Input-Kandidaten zu einem neuen Kandidaten genutzt wird, verhindert.

3.3 Zu lange Kanten

Abbildung 9 veranschaulicht eine häufig auftretende Situation. Es handelt sich um das Matchen über eine auf dem Chip sehr lange Kante. Dadurch verbessert sich evtl. die Größe des Circuits, jedoch sind nach dem Match nun zwei sehr lange Kanten auf dem Chip vorhanden, was einen großen Routing Aufwand und weitere Kosten mit sich bringt und somit eine zu vermeidende Situation ist.

In Kapitel 4.1.2 wird eine zusätzliche Klasse von Kanten eingeführt, über welche man nicht matchen darf. Diese Kanten bezeichnet man als konstant. Deshalb wird bei der Bildung eines jeden Matches darauf geachtet, über keine konstante Kante zu matchen.

Durch das Hinzufügen der zu langen Kanten zu den konstanten Kanten, kann keine optimale Lösung mehr im

Kern-Algorithmus garantiert werden, denn es ist möglich, aber unwahrscheinlich, dass die zusätzlichen Routing-Kosten vernachlässigbar sind und ein Match über diese Kante für eine optimale Lösung notwendig ist.

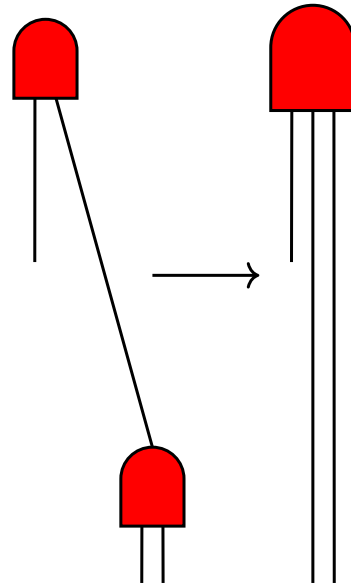


Abbildung 9: Visualisierung eines Matches über eine lange Kante

3.4 Teilweise überflüssige Subcircuits

Abbildung 4 beweist, dass nicht unbedingt alle Inputs eines Circuits relevant sind für die Outputs. Zur genaueren Einordnung folgt eine Definition:

Definition 3.5. Vollständig überflüssiger und teilweise überflüssiger Circuit

Sei C ein Circuit mit logischer Funktion $f : \{0,1\}^n \rightarrow \{0,1\}^m$.

C wird vollständig überflüssig genannt, wenn gilt:

$$\exists y \in \{0,1\}^m : \forall x \in \{0,1\}^n : f(x) = y$$

C wird teilweise überflüssig genannt, wenn es eine Teilmenge der Inputs gibt, von denen die Signale der Outputs nicht abhängen.

Die Berücksichtigung von vollständig überflüssigen Subcircuits würde bedeuten, dass Teile des Circuits entfernt werden und die Outputs der Inputs der nachfolgenden Knoten an permanenten Strom gelegt oder mit der Erdung

des Chips verbunden werden müssen. Dies lässt sich jedoch weiter optimieren. Da die Information an den nachfolgenden Gates vorhersagbar ist, muss sie auch nicht verarbeitet werden. Daraus folgt eine hohe Einsparung von Kosten, jedoch bedingt es ebenfalls einen großen Aufwand zur Implementierung in die aktuelle Architektur des Technology Mapping Algorithmus. In der Praxis ist das Vorkommen von vollständig überflüssigen Subcircuits verschwindend gering. Von daher werden die vollständig überflüssigen Subcircuits in dieser Arbeit nicht weiter behandelt und kommen in der Implementierung der noch folgenden Algorithmen nicht vor.

Im Gegensatz dazu kommen die teilweise überflüssigen Circuits durchaus vor. Bei der Konstruktion eines Chips geschieht dies durch das Zusammensetzen unterschiedlicher Circuits.

In den meisten Fällen werden die teilweise überflüssigen Circuits automatisch bei der Suche der Matche gefunden, da die irrelevanten Inputs nicht mehr unter den Inputs des Matche auftauchen und somit beim Bau des äquivalenten Graphen verschwinden. Dies veranschaulicht Abbildung 10.

Es gibt dabei jedoch noch eine Besonderheit. Wenn alle Inputs bis auf einen irrelevant sind, so ist das resultierende Gate des Matches entweder ein Inverter(INV) oder ein Buffer(BUFF). Ersteres lässt sich als Input-Invertierung des darüberliegenden Input-Pins speichern.

Ein Buffer ist jedoch nicht unbedingt in der Library für das Technology Mapping vorhanden und kann vermieden werden, indem kein Buffer, sondern nur die Kanten vom Input des Buffers zu seinen Outputs gebaut werden. Dies verhindert den Einbau eines nicht nötigen Gates. Da $fanout_{max}$ und $fanin_{max}$ im weiteren Verlauf dieser Arbeit beschränkt werden, lässt sich dies in linearer Laufzeit implementieren. Diese zusätzliche Bearbeitung läuft dann in jedem folgenden Algorithmus automatisch im Hintergrund und findet keine Erwähnung mehr. **oder doch ?**

Obwohl nun das Gegenbeispiel aus Kapitel 2.2 nicht mehr gültig ist, lässt sich mit den Möglichkeiten des Technology Mappings in der Regel keine optimale Realisierung der des Circuits zugrunde liegenden logischen Funktion finden.

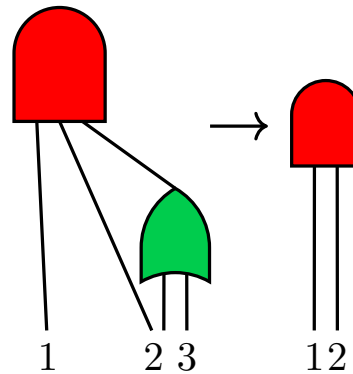


Abbildung 10: Nur Input 1 und 2 sind die relevanten Inputs

3.5 Match-Probleme

In Circuits mit beliebig vielen Highfanoutknoten steigt die Anzahl der möglichen Matche eines Knoten sehr stark an, was eine direkte Auswirkung auf die Größe der Menge der möglichen Kandidaten hat.

Es folgt eine weitere Betrachtung von Matches und daraus ableitend ein Algorithmus, der auf realen Instanzen fast alle möglichen Matche findet und in polynomieller Zeit implementierbar ist.

3.5.1 Obere Schranke für die Anzahl Matche eines Knoten

Folgendes Lemma legt eine äquivalente Definition für Matche nahe. Sei ein Circuit C ohne teilweise überflüssige Subcircuits gegeben:

$$\bar{C}_v := (V(C[\text{cone}(v)]), \{(u, w) | (w, u) \in E(C[\text{cone}(v)])\}).$$

Lemma 3.6. *Die Inputs eines Matches auf v korrespondieren zu den Kanten eines gerichteten v -Schnitts in \bar{C}_v .*

Beweis: Zu jedem Inputpin eines Matches gehört eindeutig eine Kantenmenge K aus C . Es ist möglich, dass $|K| > 1$, denn ein Match kann mehrere ausgehende Kanten eines Knoten zu einer Kante zusammenfassen. Dies veranschaulicht Abbildung 11 für ein Match eines MUX Gates. Sei I die Menge der mit den Inputs korrespondierenden Kantenmengen.

Es genügt zu zeigen, dass alle Kanten I' in I gerade die Kantenmenge eines gerichteten v -Schnitts in \bar{C}_v sind.

Angenommen I' bildet keinen gerichteten v -Schnitt, dann existiert in $\bar{C}_v \setminus I'$ ein Weg von einem Input von $\text{cone}(v)$ zu v , welcher keine Kante aus I' benutzt.

Eine Kante dieses Weges muss jedoch zu einem der Inputs des Matches gehören, denn sonst hätte das Match einen Seiteninput, was nicht erlaubt ist, oder es existiert ein teilweise überflüssiger Subcircuit. \square

Die Korrespondenz ist nicht eindeutig; da anhand des Schnitts nicht eindeutig hergeleitet werden kann, ob mehrere Kanten eines Outputs zusammengefasst oder getrennte Input-Kanten eines Matches sind. Dies wird ebenfalls durch Abbildung 11 veranschaulicht. Ohne Highfanoutknoten ist diese Zuweisung jedoch, abgesehen von den möglichen Invertierungen der Inputs und des Outputs, eindeutig, wenn es keine zwei Gates der Library gibt, welche die gleiche logische Funktion realisieren.

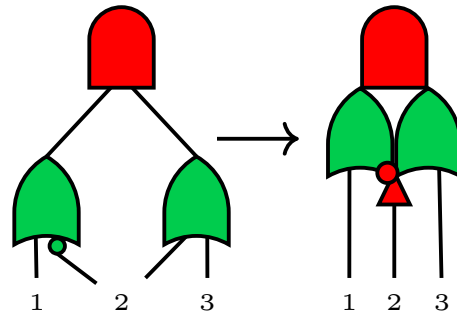


Abbildung 11: Das MUX Match verbindet zwei Kanten eines Highfanoutknoten

In einem Graphen ohne Highfanoutknoten gilt somit:

Die Menge der Matche von v ist somit gerade die Menge aller, maximal $fanin_{max}$ großen v -Schnitte in \bar{C}_v , inklusive aller möglicher Inputinvertierungen und Outputinvertierungen, die als Subgraph die logische Funktion eines Gates der Library realisieren.

Korollar 3.7. *Sei C ein Circuit. Die Anzahl der Matche eines Knotens $v \in V(C)$ mit $n_v := |E(C[cone(v)])|$, wobei $C[cone(v)]$ keine Highfanoutknoten enthält, ist durch $n_v^{fanin_{max}} 2^{fanin_{max}+1} fanin_{max}$ beschränkt.*

Beweis: Sei $X_v := \{E \subseteq E(C[cone(v)]) \mid |E| \leq fanin_{max}\}$. Die Menge der maximal $fanin_{max}$ großen v -Schnitte ist in $C[cone(v)]$ durch $|X_v|$ beschränkt. Es gilt hierbei:

$$|X_v| = \sum_{i \leq fanin_{max}} \binom{n}{i} \leq \sum_{i \leq fanin_{max}} \frac{n!}{i!(n-i)!} \leq n^{fanin_{max}} fanin_{max}$$

Für jeden Schnitt gibt es zwei verschiedene mögliche Output-Invertierungen und maximal $2^{fanin_{max}}$ viele Möglichkeiten die Inputs zu invertieren. Jedes mögliche Match ist nun durch ein Element aus X_v und eine Wahl von Invertierungen eindeutig charakterisiert. \square

Für allgemeinere Circuits lässt sich folgende Schranke angeben:

Korollar 3.8. *Sei C ein Circuit. Die Anzahl der Matche eines Knotens $v \in V(C)$ mit $n_v := |E(C[cone(v)])|$ mit $\Delta C[cone(v)] \leq fanout_{max}$, enthält, ist durch $n_v^{fanin_{max}} fanin_{max}^2 2^{fanin_{max}+1} 2^{fanout_{max}}$ beschränkt.*

Beweis: Korollar 3.7 gibt eine obere Schranke für die Anzahl aller Matche inklusive der möglichen Invertierungen an. Sei $u \in x \in X_v$ (Definition siehe oben) ausgehende Kante eines Highfanoutknoten w , so gibt es bis zu $2^{fanout_{max}-1}$ Möglichkeiten, weitere Kanten von w der korrespondierenden Kantemenge von u hinzuzufügen. Diese Möglichkeit besteht für alle maximal $fanin_{max}$ Kanten von x . Durch Multiplikation mit der oberen Schranke aus Korollar 3.7 folgt die Aussage. \square

In einem Circuit ohne Highfanoutknoten lässt sich die Schranke noch genauer angeben.

Korollar 3.9. *Sei C ein Circuit ohne Highfanoutknoten. Die Anzahl der Matche eines Knoten ist durch $2^{2^{fanin_{max}+1}} fanin_{max}$ beschränkt.*

Beweis: Ausgehend von einem Knoten $v \in V(C)$ mit maximal $fanin_{max}$ Inputs, gibt es $2^{fanin_{max}}$ Möglichkeiten, die an den Inputs liegenden Gates mit in das Match einzuschließen. Bis auf eine, schließt jede dieser Möglichkeiten mindestens ein Gate mit ein. Da C keine Highfanoutknoten enthält, wird der Fanin des Matches um mindestens eins erhöht, denn es ist nicht möglich,

Kreise zu schließen. Daraus folgt, dass dieser Schritt maximal $fanin_{max}$ mal durchführbar ist. Für jeden so berechneten Prototyp einen Matches gibt es noch $2^{fanin_{max}}$ mögliche Invertierungen der Inputs und 2 des Outputs, woraus die obige Aussage folgt. \square

Definition 3.10. Sei $\mathcal{M} := |V(C)|^{fanin_{max}} fanin_{max}^2 2^{fanin_{max}+1} 2^{fanout_{max}}$ eine Bezeichnung der oberen Schranke für die Anzahl der Matche eines Knotens in einem Graphen mit beschränktem $fanin_{max}$ und $fanout_{max}$.

3.5.2 Match-Suche in polynomieller Zeit

Der Kern-Algorithmus aus Kapitel 2.2 findet alle möglichen Matche eines beliebigen Knotens in polynomieller Zeit, da in dem Circuit keine Highfanoutknoten existieren. Aus Korollar 3.9 folgt, dass potenzielle Matche aller Knoten in $\mathcal{O}(2^{2fanin_{max}+1} fanin_{max} |V(C)|)$ errechnet werden können. Da $fanin_{max}$ als Konstante deklariert wurde, entspricht dies linearer Laufzeit. Es bleibt zu prüfen, ob ein solcher Prototyp einem Match in C entspricht, also ob die logische Funktion des Subcircuits einem Match der Library gleicht. Für jede der maximal $2^{fanin_{max}}$ möglichen Wahrheitsbelegungen der Inputs wird der Wahrheitsgehalt des Outputs errechnet. Dies ist linear in $|V(C)|$ möglich. Die dadurch errechnete Tabelle wird mit den $|L|$ Gates der Library verglichen.

Daraus folgt, dass das Finden aller Matche in $\mathcal{O}(|V(C)|^2 |L|)$ erfolgt und somit polynomiell möglich ist.

Bei Circuits mit beliebig vielen Highfanoutknoten, aber beschränktem $fanout_{max}$, lassen sich nach Korollar 3.8 und gleichem Vorgehen, wie soeben beschrieben, alle Matche ebenfalls in polynomieller Zeit finden. Dabei beträgt die Laufzeit $\mathcal{O}(|V(C)|^{fanin_{max}+2} |L|)$.

3.5.3 Heuristische Match-Suche

In der Praxis wird, ähnlich der Routine ohne Highfanoutknoten, von dem Gate eines Knoten ausgehend überprüft, ob dieser, oder eine mögliche Invertierung der Inputs und des Outputs, einem Gate der Library entspricht. Daraufhin wird für jede der maximal $2^{fanin_{max}}$ Möglichkeiten, die an den Inputs liegenden Gates mit in das potenzielle Match hinzuzufügen, ein weiteres potenzielles Match gebildet. Für dieses wird mit jeder möglichen Invertierung der Inputs und des Outputs überprüft, ob es einem Match entspricht. Daraufhin wird jede der maximal $2^{fanin_{max}}$ Möglichkeiten, die an den Inputs liegenden Gates mit in das potenzielle Match hinzuzufügen, inklusive möglicher Invertierungen überprüft. Dieser Vorgang wird für jede der Möglichkeiten so lange wiederholt, bis die Anzahl der Inputs des durch das potenzielle Match, beschriebenen Subcircuit größer als $fanin_{max}$ ist. Ab diesem Punkt wird das vorliegende potenzielle Match nicht mehr erweitert. Dies garantiert, bei einem Circuit mit Highfanoutknoten, nicht das Finden

aller möglicher Matche, da sobald ein potenzielles Match einen Highfanoutknoten mit einschließt, die Zahl der Inputs sinken kann.

In der Praxis werden so jedoch die überwiegende Mehrheit der Matche gefunden. Der Verlust einiger weniger Matche ermöglicht jedoch einen enormen Laufzeitgewinn. **die dissertation für den fall ohne highfanoutknoten erwähnen ? und nochmal genau durchsehen ob die teilweise überflüssigen Subcircuits überall rausgehalten wurden**

boolean Matching hier erwähnen und sinnvolle quellen mit einbringen

3.6 Kandidaten-Probleme

Die Anzahl der Kandidaten an einem Knoten ist im Allgemeinen nicht polynomiell beschränkt, wie Abbildung 7 beweist. Dies gilt offenbar auch wenn der Circuit C keine Highfanoutknoten hat. Dieses Problem wird durch das Filtern mit Buckets in Kapitel 3.6.2 gelöst.

3.6.1 Schranken für Arrivalttime und Größe

Zur Vorbereitung auf das Filtern mit Buckets wird noch folgender Algorithmus eingeführt.

Für die Bearbeitung einer Tradeoffkurve ist es notwendig Schranken angeben zu können, zwischen welchen sich alle Werte der Kurve befinden, damit sich nach dem Filtern in Kapitel 3.6.2 die Anzahl der noch vorhandenen Kandidaten abschätzen lässt.

Durch den vorhandenen Tradeoff befindet sich die AT eines Circuits C zwischen der schnellsten und der kleinsten Implementierung von C . Dementsprechend befindet sich $Area$ von C zwischen der kleinsten und schnellsten Lösung. Folgender Algorithmus nähert die entsprechenden Circuit-Kandidaten

an.

Algorithmus : Untere Schranke Arrivalttime

Input : Circuit C , Library L

```

1 schnellster_kandidat[]  $\leftarrow \emptyset$ 
2 schnellster_inv_kandidat[]  $\leftarrow \emptyset$ 
3 foreach Knoten  $v \in V(G)$  in topologischer Reihenfolge do
4     berechne_alle_(invertierten)_Matche( $v$ )
5     foreach Match  $m$  auf  $v$  do
6         berechne_schnellsten_Kandidaten( $m, v$ )
7         Update schnellster_(inv)_kandidat
8     if  $v$  ist Highfanoutknoten then
9         if schnellster_kandidat[ $v$ ].AT <
            schnellster_inv_kandidat[ $v$ ].AT then
10            lösche schnellster_inv_kandidat[ $v$ ]
11        else
12            lösche schnellster_kandidat[ $v$ ]
13 Baue  $C'$  entsprechend schnellster_kandidat[ $o$ ]
14 return  $C'$ 

```

Dieser Algorithmus führt den Kern-Algorithmus auf C aus und behält für jeden Knoten nur einen Kandidaten. Dadurch ist es nicht notwendig mehr als $|V(C)|$ Kandidaten zu speichern.

Die untere Schranke für Area wird mit einem entsprechenden Algorithmus berechnet.

Im Gegensatz zu dem Kern-Algorithmus garantiert dieser keine optimale Lösung bezüglich Arrivalttime oder Area, da es möglich ist, dass für einen Knoten v und einen seiner Inputs i der schnellste bzw. kleinste Kandidat auf v eine Invertierung an i vorsieht. Da der beste Kandidat mit invertiertem Output von i evtl. in Schritt 10 oder 12 gelöscht wurde, ist der optimale Kandidat für v möglicherweise nicht mehr vorhanden. Der Unterschied zum Kern-Algorithmus liegt darin, dass für i die besten Kandidaten beider Invertierungen gespeichert werden. Da i hier jedoch evtl. ein Highfanoutknoten ist würde dies zu einem Verlust der Polynomialität führen.

Unter welchen Bedingungen sich dieser Algorithmus polynomiell implementieren lässt, wird in Kapitel 3.8 weiter ausgeführt. Dort wird dieser Algorithmus mit der Laufzeit der Heuristik ohne diese Routine beschrieben. Da jedoch nur maximal zwei Kandidaten an jedem Knoten gespeichert werden und alle Matche nur einmal gefunden werden müssen, wird die Laufzeit der

Heristik mit der Hinzunahme dieses Algorithmus nur geringfügig beeinflusst.

Der Untere Schranke Arrivalttime Algorithmus wurde in [Elb17] entwickelt.

3.6.2 Filtern mit Buckets

Die Kandidaten eines beliebigen Knotens sind in Tradeoffkurven, nach Klassen sortiert, gespeichert. Die Werte einer solchen Kurve lassen sich in Abschnitte (Buckets) fester Größe einteilen. Dabei lässt sich eine Kurve sowohl in Delay-Buckets der Größe δ_{delay} , als auch in Area-Buckets der Größe δ_{area} unterteilen. Dies wird in Abbildung 12 veranschaulicht. Diese und Abbildung 5 wurden [Elb17] entnommen.

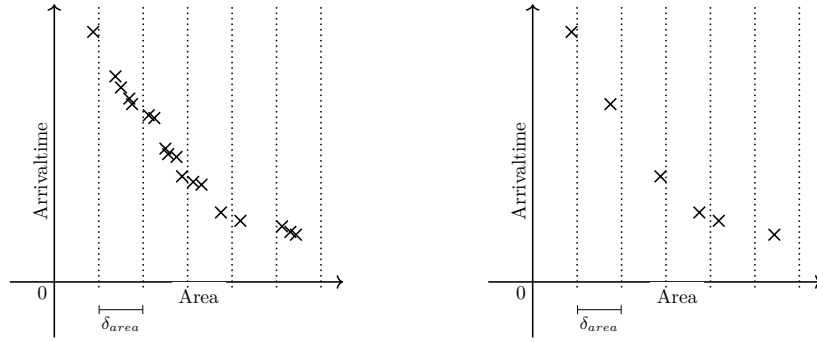


Abbildung 12: Einteilen und Filtern mit Buckets

Man wählt die Bucketgrößen $\delta_{delay} = \frac{\varepsilon X^-}{2n\lambda}$, $\delta_{area} = \frac{\varepsilon X^-}{2n(1-\lambda)}$ (mit Tradeoff λ), und speichert erst nur den kleinsten Kandidaten in jeder Area-Bucket und von den verbleibenden nur den schnellsten in jeder Delay-Bucket. Hierbei entspricht X^- der mit dem Untere Schranke Area Algorithmus errechneten Schranke und n der Anzahl der Knoten des Circuits.

Für $\lambda \in \{0, 1\}$ ist jeweils eine der Bucketgrößen nicht definiert. In diesem Fall wird für den nicht definierten Fall nur der beste Kandidat (schnellster bzw. kleinster) behalten; d.h. im nicht definierten Fall die Tradeoffkurve in nur ein Bucket unterteilt.

Die maximale Anzahl an Kandidaten in einer Tradeoffkurve ist nach dem Filtern polynomiell beschränkt. Filtert man jede Tradeoffkurve wie beschrieben, so ist der noch bestmögliche Circuit Kandidat nur um den Faktor ε von der optimalen Lösung entfernt. Ein Beweis dieser Aussage befindet sich in [Elb17].

Definition 3.11. Sei \mathcal{B} die maximale Anzahl an Kandidaten in einer Tradeoffkurve, nachdem sie gefiltert wurde.

Des Weiteren sei \mathcal{K} die maximale Anzahl von Klassen eines Knotens.

Korollar 3.12. *Für Circuits C mit konstanter Anzahl k an Highfanoutknoten sind \mathcal{B} und \mathcal{K} polynomiell beschränkt.*

Beweis: \mathcal{B} ist beschränkt durch die Größe des größtmöglichen äquivalenten Circuit C_A von C geteilt durch δ_{area} . Sei X^+ eine mit dem Algorithmus: Untere Schranke Arrivalttime errechnete obere Schranke für die Größe von C . Daraus folgt $\mathcal{B} \in \mathcal{O}(X^+ \cdot \frac{2n(1-\lambda)}{X-\epsilon})$. \mathcal{B} ist somit polynomiell in $\frac{X+n}{X-\epsilon}$. Da die Größe $area_{max}$ (Delay) des größten (langsamsten) Gates in der, in Abbildung 2, beschriebenen Library weniger als einen Faktor 10 größer (langsamer) ist die entsprechenden Werte des kleinsten (schnellsten) Gates dieser Library folgt, dass $\frac{X+n}{X-\epsilon}$ polynomiell beschränkt ist. Der größtmögliche zu C äquivalente Circuit ist nämlich nicht größer als $|V(C)|(fanin_{max} + 1)area_{max} + |O|area_{max}$. Die Größe jedes Gates möglichen Invertiern an seinen Inputs in C ist durch $(fanin_{max} + 1)area_{max}$ beschränkt und die Anzahl der an den Outputs O liegenden Inverter durch $|O|$.

Für eine allgemeine Library findet sich in [Elb17] eine Lösung zur Erhaltung der Polynomialität.

Sei $v \in V(C)$ und \mathcal{K}_v die Anzahl Klassen von v . Sei des Weiteren $K_{h,i}$ die maximale Anzahl Klassen eines Highfanoutknoten mit i weiteren Highfanoutknoten in seiner Cone. \mathcal{K}_v an einem Knoten v ist beschränkt durch die Anzahl aller möglichen Kombinationen von Kandidaten aller möglicher Klassen der Highfanoutknoten in $cone(v)$. Daraus folgt $\mathcal{K}_v \leq (\mathcal{B}K_{h,k-1})^k$; da v maximal k Highfanoutknoten in seiner Cone hat, welche jeweils maximal $k-1$ Highfanoutknoten in ihrer Cone haben. Deshalb gilt $\mathcal{K}_{h,k} \leq (\mathcal{B}K_{h,k-1})^k$. Es gilt $\mathcal{K}_{h,0} \leq 1$, da der Highfanoutknoten keine offenen Knoten besitzen kann.

Daraus folgt $\mathcal{K}_v \leq \mathcal{B}^{k!}$; da k beschränkt ist folgt, dass \mathcal{K} polynomiell beschränkt ist. □

Beschränkt man die maximale Anzahl von Highfanoutknoten in C , so folgt aus Korollar 3.12, dass die Anzahl der Klassen eines jeden Knotens und die Menge der sich darin befindlichen Kandidaten, polynomiell beschränkt ist. Deshalb geht der im Folgenden vorgestellte polynomielle Algorithmus von einer beschränkten Anzahl von Highfanoutknoten aus.

Es folgt eine kurze Beschreibung, wie die Kandidatenmenge eines Knotens gebildet wird.

3.6.3 Verknüpfen von Kandidaten

Ohne Highfanoutknoten lassen sich bei der Konstruktion eines Kandidaten für einen Knoten x , die Kandidaten k_i der Inputs i unabhängig voneinander wählen. Da das Klonen ausgeschlossen wurde, ist dies bei Circuits mit Knoten höheren Fanouts nicht möglich.

Die folgenden Bedingungen stellen sicher, dass Klonen verhindert wird. Seien für einen Knoten v , O_v die Menge seiner offenen Knoten.

1. $\forall v, w \in \text{Inputs}(x) \forall y \in O_v \cap O_w : \text{class}(k_v)(y) = \text{class}(k_w)(y)$
2. $\forall v, w \in \text{inputs}(x)$ mit $v \in O_w : k_v = \text{class}(k_w)(v)$

Bedingung 1 stellt sicher, dass an den offenen Knoten der Input-Kandidaten ein eindeutiger Kandidat festgelegt wird.

Bedingung 2 sichert diese Eigenschaft auch für die Inputs selber, denn es ist möglich, dass ein Input-Knoten von x auch ein offener Knoten eines weiteren Inputs ist. Für diesen wird dadurch ebenfalls ein eindeutiger Kandidat festgelegt.

Somit sind alle Input-Kandidaten-Mengen, welche diese beiden Bedingungen erfüllen, eine mögliche Grundlage für einen Kandidaten auf x .

3.6.4 Finden von Kandidaten

Für jedes der maximal \mathcal{M} Matche eines Knotens v von einem Circuit mit den oben genannten Einschränkungen gilt folgendes zur Findung aller passender Kandidaten:

Jedes Match m besitzt höchstens fanin_{\max} Inputs. Sei K_i die Menge der Klassen von Input i des Matches. Jedes Element aus der Menge der möglichen Klassen $\prod_{i \leq |\text{inputs}(m)|} K_i$ wird auf die obigen Bedingungen überprüft. Die Kombinationen O , welche beide Bedingungen erfüllen, bilden eine Klasse von v . In die dazugehörige Tradeoffkurve kommt nun jede nicht dominierte Kombination von Kandidaten der Tradeoffkurven von O .

Die lässt sich in Laufzeit $\mathcal{O}(\mathcal{M}(\mathcal{KB})^{\text{fanin}_{\max}})$ implementieren. Da alle Elemente dieser Formel polynomiell beschränkt sind und fanin_{\max} konstant ist, entspricht dies polynomieller Laufzeit.

3.7 FPTAS

Für Circuits mit einem Output, beschränktem $\text{fanin}_{\max}, \text{fanout}_{\max}$ ohne teilweise überflüssige Subcircuits und mit maximal k Highfanoutknoten gibt es für das folgende Problem einen FPTAS (fully polynomial time approximation scheme). Ein FPTAS ist ein Algorithmus, welcher, gegeben ein beliebiges $\varepsilon > 0$, eine Lösung des Problems errechnet mit der Eigenschaft, dass für deren Kosten $c \leq (1 + \varepsilon) \text{OPT}$ gilt. Hierbei sind OPT die Kosten einer optimalen Lösung des Technology Mappings, bezüglich der bisher eingeführten Operationen.

FPTAS FÜR DAS TECHNOLOGY MAPPING

Instanz: Circuit C mit einem Output, Library L mit beschränktem $fanin_{max}$, maximal k Highfanoutknoten mit beschränkten $fanout_{max}$, Tradeoffparameter $\lambda \in [0, 1]$, Toleranz $\varepsilon > 0$.

Aufgabe: Finde einen Circuit-Kandidaten K auf C , mit Kosten $\lambda AT(K) + (1 - \lambda) Area(K) \leq (1 + \varepsilon) OPT$.

Diese Problemstellung lässt sich mit folgendem Algorithmus lösen.

Algorithmus : FPTAS für das TM mit Konvexkombination

Input : Circuit C ohne teilweise überflüssige Subcircuits und mit finalem Output o , Library L mit beschränktem $fanout_{max}$, $\varepsilon > 0$, k Highfanoutknoten mit beschränktem $fanout_{max}$, Tradeoff $\lambda \in [0, 1]$

- 1 $M \leftarrow \text{finde_alle_matche}(C)$
- 2 $\text{lösche_konst_Kanten_überdeckende_Matche}(M, C)$
- 3 $X^- \leftarrow \text{Untere_Schranke_Area}(C)$
- 4 **foreach** Knoten $g \in V(C)$ **do**
- 5 $\text{berechne_offene_Knoten}(g)$
- 6 **foreach** Knoten $g \in V(C)$ in topologischer Reihenfolge **do**
- 7 $g.\text{tradeoff_curves}[] \leftarrow [\emptyset]$
- 8 **foreach** Match $m \in M[g]$ auf g **do**
- 9 **foreach** mögliche Klasse A auf g **do**
- 10 **foreach** Kandidaten k auf g mit m der A respektiert **do**
- 11 **if** k ist nicht dominiert in $g.\text{tradeoff_curves}[A]$ **then**
- 12 $g.\text{tradeoff_curves}[A].\text{push_back}(k)$
- 13 $\text{filter_mit_Buckets}(g.\text{tradeoff_curves}, X^-, \varepsilon)$
- 14 $C' \leftarrow \text{Circuit}(\text{besten_Kandidat}(o, \lambda))$
- 15 **return** C'

Hierbei wurde der Kern-Algorithmus um die in diesem Kapitel beschriebenen Routinen erweitert.

Für jeden Knoten werden zuerst alle, keine konstanten Kanten überdeckenden, Matche berechnet und seine offenen Knoten ermittelt. Dann werden für alle Knoten, in topologischer Reihenfolge, und deren Klassen alle nicht dominierten Kandidaten hinzugefügt und die Tradeoffkurven daraufhin gefiltert. Es bleiben für jeden Knoten, aufgrund der Beschränkung auf k Highfanoutknoten, polynomiell beschränkt viele Kandidaten übrig. So auch für den Output o . Anschließend wird aus o 's einziger Tradeoffkurve der beste Kandidat k^* ausgewählt und der zu C äquivalente Circuit C' anhand von k^* gebaut und zurückgegeben.

Lemma 3.13. *FPTAS ist polynomiell in $\mathcal{O}(|V(C)|^2 + |V(C)|\mathcal{MBK}^2)$ implementierbar.*

Beweis: Schritt 1 lässt sich nach Kapitel 3.5.2 in polynomieller Zeit implementieren. Schritt 2 ist linear in der polynomiell beschränkten Anzahl an Matches im gesamten Circuit. Die Schritte 4 und 5 lassen sich in $\mathcal{O}(|V(C)|^2)$ errechnen, da von einem Knoten v ausgehend durch iterieren über die Inputs alle Knoten der Cone hinzugefügt werden. Highfanoutknoten, welche Inputs eines offenen Knoten sind, werden als offen deklariert und Highfanoutknoten, welche einen Output außerhalb der $\text{cone}(v)$ besitzen, werden ebenfalls als offen markiert. Da fanout_{\max} beschränkt ist, folgt eine lineare Laufzeit in $|V(C)|$ für die Berechnung der offenen Knoten von v . Dies wird für jeden Knoten aus C wiederholt.

Die Schleifendurchläufe der Schritte 6, 7, 8, 9 und 10 sind beschränkt durch $|V(C)|$, \mathcal{M} , \mathcal{K} , \mathcal{B} . Schritt 11 und 12 sind in $\mathcal{O}(\mathcal{K})$ implementierbar.

Die Menge der Kandidaten in den Tradeoffkurven sind beschränkt durch die Anzahl der Schleifendurchläufe der Schritte 8-12. Somit liegt die Laufzeit polynomiell in der Anzahl der Schleifendurchläufe.

Schritt 14 ist durch die maximale Anzahl der Kandidaten einer Tradeoffkurve und $|V(C)|$ beschränkt. Schritt 3 liegt in der Summe der Laufzeit der bisher beschriebenen Schritte.

Daraus folgt eine Gesamtlaufzeit von $\mathcal{O}(|V(C)|^2 + |V(C)|\mathcal{MBK}^2)$. \square

3.8 Heuristik

Im folgenden Algorithmus ist die Menge der Knoten mit $\text{fanout} > 1$ beliebig groß. Die Anzahl der Highfanoutknoten bestimmt maßgeblich den Speicherbedarf an Kandidaten und die Laufzeit des FPTAS. Sie sind der Grund, warum sich das FPTAS für eine Anwendung des Technology Mappings auf einem gesamten Chip nicht eignet.

Von daher ist es ein naheliegender Ansatz für eine Heuristik, an jedem Highfanoutknoten nur einen Kandidaten zu speichern. Es hat einen sehr hohen Einfluss auf die Güte der Lösung, welchen man dort auswählt. Routinen für eine solche Auswahl werden in Kapitel 4.2 ausführlich behandelt. Im Folgenden wird beispielhaft eines dieser Verfahren erläutert.

Ist an jedem Highfanoutknoten ein Kandidat gewählt, so lässt sich der noch bestmögliche Circuit-Kandidat schnell errechnen, da für jeden Knoten nur

eine Klasse mit einer Tradeoffkurve vorhanden ist.

Algorithmus : Heuristik für das TM mit Konvexkombination

Input : Circuit C ohne vollständig überflüssige Subcircuits und mit
finalem Output o , Library L , Tradeoff $\lambda \in [0, 1]$

```

1  $M \leftarrow \text{finde\_alle\_matche}(C)$ 
2  $\text{lösche\_konst\_Kanten\_überdeckende\_Matche}(M, C)$ 
3 foreach Knoten  $g \in V(C)$  do
4    $\text{berechne\_offene\_Knoten}(g)$ 
5 foreach Knoten  $g \in V(C)$  in topologischer Reihenfolge do
6    $g.\text{tradeoff\_curves}[] \leftarrow [\emptyset]$ 
7   foreach Match  $m \in M[g]$  auf  $g$  do
8     foreach mögliche Klasse  $A$  auf  $g$  do
9       foreach Kandidaten  $k$  auf  $g$  mit  $m$  der  $A$  respektiert do
10        if  $k$  ist nicht dominiert in  $g.\text{tradeoff\_curves}[A]$  then
11           $g.\text{tradeoff\_curves}[A].\text{push\_back}(k)$ 
12   if  $g$  ist Highfanoutknoten then
13      $\text{guess} \leftarrow \underset{\text{Kandidat } k \text{ auf } v}{\text{argmin}} \{ \lambda AT(k) + (1 - \lambda) \text{area}(k) \}$ 
14     foreach Kandidat  $k$  auf  $g$  do
15       if  $k \neq \text{guess}$  then
16          $\text{lösche } k$ 
17    $\text{filter\_mit\_buckets}(g.\text{tradeoff\_curves}, 0)$ 
18  $C' \leftarrow \text{circuit}(\text{bester\_kandidat}(o, \lambda))$ 
19 return  $\text{entferne\_buffer}(C')$ 

```

Dieser Algorithmus unterscheidet sich nur in den Schritten 12-17 und 19 von dem FPTAS; diese Schritte beschreiben ein einfaches Verfahren zur Auswahl eines Kandidaten für einen Highfanoutknoten. Genau wie bei dem Kern-Algorithmus wird nur der Kandidat behalten, welcher den Tradeoff minimiert. Dies garantiert keine optimale Lösung.

Die Menge an Kandidaten in einer Tradeoffkurve ist in der Praxis, durch die Einschränkung auf einen Kandidaten für Highfanoutknoten, deutlich geringer als im FPTAS. Aus diesem Grund lässt sich in der Praxis ohne Laufzeiteinbußen mit $\varepsilon = 0$ filtern.

Weitere Informationen und Erweiterungen dieser Routine befinden sich in Kapitel 4.2 und 6.

Schritt 19 entfernt die durch teilweise überflüssige Subcircuits evtl. hinzugekommenen Buffer.

Laufzeit: Stellt man die gleichen Voraussetzungen an die Circuits wie der FPTAS und filtert mit einem $\varepsilon > 0$, so lässt sich diese Heuristik in derselben polynomiellen Laufzeit implementieren.

In allgemeinen Circuits ist dies wahrscheinlich nicht mehr möglich, denn die Anzahl möglicher Matche steigt, durch die teilweise überflüssigen Circuits, exponentiell (siehe Kapitel 3.5).

4 Erweiterungen der Heuristik

4.1 DAGs mit mehreren Outputs

Bisher wurde auf Circuits C mit nur einem Output gearbeitet. Reale Instanzen eines Chips sind jedoch mit beliebig vielen Outputs ausgestattet. Outputs können auch zusätzlich noch Nachfolger in C besitzen.

Da Signale von Output-Knoten mit Fanout in C aus der Cone darüberliegender Knoten laufen können, werden sie ebenfalls als offene Knoten ihrer Nachfolger deklariert. In einem Circuit mit mehreren Outputs ist AT alleine ein unzureichendes Optimierungskriterium.

4.1.1 Required Arrivaltimes

In Definition 2.6 wurde bereits der Begriff der Arrivaltime eines Knotens eingeführt. Dies ist die Zeit, zu welcher das letzte Signal bei einem Knoten ankommt. Diese Werte sind für die Input-Knoten eines Circuits C gegeben und werden von dort aus (unter Hinzunahme von Wire-, Gate- und Inverter-Delay) für jeden Knoten von C (in topologischer Reihenfolge) errechnet.

Im Designprozess eines Chips, gibt es neben der tatsächlichen Arrivaltime auch noch eine gewünschte Arrivaltime RAT (required AT), welche an den Outputs eines Graphen gegeben ist und ähnlich zur AT durch C propagiert wird. Somit ist sowohl AT und RAT eine Funktion auf $V(C)$.

Der Vollständigkeit wegen folgt die genaue Definition der RAT:

Definition 4.1. RAT:

Sei C ein Circuit und $v \in V(C) \setminus \text{Outputs}(C)$. Die RAT (required arrivaltimes) an v ist definiert durch:

$$RAT(v) := \min_{\substack{(v,x) \in E(C), \\ i: \text{inputs}(x)[i]=v}} \{RAT(x) - d_{w(v,x)} - d_{gate(x)} - d_{inv} \mathbb{1}_{inv_x(i)}\}$$

Die RAT der Outputs wird hierbei (wie das Delay der Input-Knoten) als gegeben angenommen.

In der Praxis kommen die Signale oft später an als erwartet. Der Betrag des Slack $slack(v) := RAT(v) - AT(v)$ gibt, wenn $slack(v) \leq 0$, an, um wie viel

Zeit sich das letzte Signal an v verspätet. Somit ist es viel sinnvoller einen gegebenen Circuit hinsichtlich des negativen Slacks zu verbessern.

Hieraus ergeben sich für einen Circuit die beiden folgenden Werte:

- Worst-Slack (WS): Wert des kleinsten Slacks für einen Knoten auf dem Circuit.
- Sum-of-Negative-Slacks (SNS): Summe aller negativer Slacks der Outputs eines Circuits.

Letzterer hat in der Praxis mehr Bedeutung, da eine sehr gute Verbesserung der SNS eine Verbesserung des WS in der Regel mit einschließt.

Bei der Analyse eines Chips lässt sich auf diesem ein Knoten v finden, an welchem der WS angenommen wird.

Sei C der Circuit, welcher nur aus dem Gate von v besteht. Füge nun zu v in C den Input von v hinzu, welcher den größten negativen Slack besitzt. Dies wiederhole man für das neu hinzugefügte Gate, bis man an einen Input des Chips gelangt oder der Slack nicht mehr negativ ist.

Hieraus entsteht ein Circuit C , welcher einen Output hat und aus einer hintereinander geschalteten Kette von Knoten besteht. Dieser lässt sich nun mit geeigneten Algorithmen **füge hier mal ein Beispiel oder einen Verweis an** zu einem äquivalenten Circuit C' , mit geringerer Tiefe (Anzahl Kanten eines bzgl. Kantenenge längsten Weges in C), umformen. Diese umgebauten Instanzen besitzen nur einen Output und wenige Highfanoutknoten mit geringem Ausgangsgrad. Dadurch sind diese Instanzen gut für den oben vorgestellten FPTAS geeignet. Der große Vorteil von diesem Vorgehen sind überschaubar große Instanzen und eine Beschleunigung des gesamten Chips durch eine geringe Laufzeit des Algorithmus. Der Nachteil ist jedoch, dass ein Chip oft sehr viele Wege mit einem schlechten negativen Slack hat und man somit den Chip nur inkrementell beschleunigt.

Eine alternative Herangehensweise an das Technology Mapping ist, einen Circuit dahingehend zu optimieren, dass die SNS minimiert wird. Dies ist jedoch bei den bisher betrachteten Circuits äquivalent zur Optimierung nach AT , da nur Instanzen mit einem Output betrachtet wurden und RAT für diesen eine Konstante ist.

4.1.2 Implementierung mehrerer Outputs

Wie in der Einleitung beschrieben, ist es das Ziel dieser Arbeit eine Heuristik für das Technology Mapping zu entwickeln, welche auf großen Teilen eines Chips lauffähig (bezüglich Laufzeit) ist. Da ein solcher Chip mehr als nur einen Output-Pin hat, lässt er sich in zusammenhängende Circuits unterteilen, welche mehr als einen Output-Knoten haben. Folgende Umbauten sind

notwendig, um mit dem Kern-Algorithmus auch diese Instanzen zu verbessern.

Als erstes fällt auf, dass sich, wenn der Algorithmus für jeden Knoten die Kandidatenmenge errechnet hat, nicht einfach der beste Kandidat für den Output aus seiner Tradeoff-Kurve auswählen lässt. Dieser besitzt bei mehreren Outputs nämlich in der Regel offene Knoten. Es wurde bereits dargestellt, wie man mehrere Kandidaten auswählt, sodass diese an allen offenen Knoten übereinstimmen. Somit lässt sich ein Circuit mit den Mitteln aus Kapitel 3.6.3 realisieren, welcher eine Kostenfunktion hinsichtlich Größe und WS optimiert.

Hieraus ergibt sich eine zweite Änderung. Bisher wurde das Delay eines Circuits C optimiert, indem das Signal des einen Outputs nach dem Umbau früher ankommt. Dies lässt sich auf einen Circuit mit mehreren Outputs übertragen. Da es mehrere Signale gibt, wählt man den Kandidaten des Outputs mit dem größten negativen Slack zuerst und die anderen folgen sortiert der Größe ihres Slacks nach (absteigend). Dies garantiert jedoch nicht, dass der WS des Circuits nach dem Umbau besser ist als vorher, da evtl. der Knoten der vorher den WS bildete, besser wird. Ein anderer Output könnte jedoch durch diesen Umbau schlechter werden.

Um dieses Problem zu umgehen, verändert man C vor dem Technology Mapping durch das Verbinden aller Outputs mit einem virtuellen Gate, mit nur einem möglichen Match (dem Gate an sich). Der veränderte Circuit lässt sich nun wie im Kern-Algorithmus optimieren und es wird automatisch das gerade aufgezeigte Problem gelöst.

Wie bereits in Kapitel 4.1.1 erläutert ist es in der Praxis profitabler die SNS des Circuits zu verbessern, anstatt den WS.

Dementsprechend muss aus den Kandidatenmengen der Outputs derjenige Circuit-Kandidat gebaut werden, welcher die SNS minimiert.

Dieses Kriterium ersetzt, von diesem Punkt an, das Kriterium der Delay-Optimierung in der Kostenfunktion.

Des Weiteren müssen nach dem Technology Mapping noch alle Outputs, mit der bis zu ihnen berechneten logischen Funktion, vorhanden sein. Daraus folgt, dass über einen Output-Knoten, welcher in dem Circuit noch mindestens einen Nachfolger hat, nicht gematcht werden darf, weil sonst ein nicht erlaubter Seitenoutput entstehen würde.

Dies lässt sich dadurch sicherstellen, dass man eine seiner ausgehenden Kanten als konstant deklariert, wie das bereits bei den zu langen Kanten geschehen ist.

4.2 Premapping von Highfanoutknoten

Der exponentielle Anstieg der Kandidatenmenge wird, wie in Kapitel 3.6 gezeigt, durch die Highfanoutknoten verursacht. Daraus folgt, dass ein sehr hohes Laufzeit-Potenzial in der Reduzierung der Kandidaten für diese Knoten liegt. Dieses Kapitel stellt mehrere Routinen für eine solche Reduzierung vor.

Die Kandidatenmenge eines jeden Highfanoutknotens wird, wie bereits in Kapitel 3.8 dargestellt, auf eins reduziert. Diese Routine wird auch das Premapping der Highfanoutknoten genannt. Hieraus folgt, dass jeder Knoten des Circuits C nur noch eine Klasse an Kandidaten besitzt, denn alle offenen Knoten seiner Cone sind Highfanoutknoten und somit festgelegt. Die Kandidatenmenge der Outputs, welche noch Nachfolger $o \in C$ haben, wird ebenfalls auf einen Kandidaten reduziert. Dies verhindert das Klonen in $cone(o)$, da auch die nicht offenen Knoten von o von allen Knoten der Menge $O := \{v \in Outputs(C) : o \in cone(v)\}$ mit einem Kandidaten belegt werden. Nun lässt sich der bestmögliche Kandidat eines jeden Outputs ohne Nachfolger finden, indem in der einzig verbleibenden Tradeoffkurve nach dem Kandidaten mit den geringsten Kosten gesucht wird. Dadurch ist das Finden des bestmöglichen Circuit Kandidaten, welcher keine der gelöschten Kandidaten benutzt, ohne Laufzeiteinbußen möglich.

Daraus folgt, dass der zu wählende Circuit-Kandidat eindeutig ist, sobald jedem Highfanoutknoten ein Kandidat zugewiesen wurde. Aus diesem Grund hat die Wahl der Premapping Routine eine zentrale Bedeutung in der Heuristik.

Im Folgenden werden drei verschiedene Routinen vorgestellt und auf deren Eigenschaften eingegangen. Weitergehende Analysen bezüglich der Unterschiede zwischen den Resultaten dieser Routinen sind in Kapitel 6 dargestellt.

4.2.1 Triviales Premapping

Diese Methode des Premappings wurde bereits in Kapitel 3.8 angewandt. Hierbei wird für jeden Knoten folgender Kandidat ausgewählt:

$$guess \leftarrow \arg \min_{\text{Kandidat } k \text{ auf } v} \{ \lambda AT(k) + (1 - \lambda) area(k) \}$$

Diese Methode lässt sich jedoch noch wie folgt verbessern.

Im Gegensatz zu dem Kern-Algorithmus aus Kapitel 2.2 kann auch mit $\lambda \in \{0, 1\}$ keine optimale Lösung garantiert werden. Dies beweist Abbildung ???. Jedoch befindet sich die dadurch gefundene Lösung nahe an der Optimalen.

Die Analyse, wie weit eine solche Lösung von der Optimalen entfernt ist, ist in Kapitel 6.2.5 dargestellt.

4.2.2 Premapping durch Schätzen

Beim Premapping durch Schätzen wird eine Vermutung für die Kosten des Kandidaten eines jeden Highfanoutknoten in einer optimalen Lösung des Technology Mappings aufgestellt. Ausgewählt wird dann der Kandidat, welcher die geringste Differenz zu den vermuteten Kosten besitzt.

Die im Folgenden vorgestellte Routine wurde durch [Elb17] für einen Output entwickelt und wird hier auf SNS erweitert.

Die Schätzung erfolgt durch zwei Technology Mapping Läufe, welche einmal einen möglichst schnellen und einmal einen möglichst kleinen Circuit errechnen. Der Algorithmus aus Kapitel 3.6.1 wird für die Schrankenberechnung benutzt.

Das Premapping durch Schätzen erfolgt nun durch folgenden Algorithmus:

Algorithmus : Premapping durch Schätzen

```

1  $C'_{AT} \leftarrow \text{Untere\_Schranke\_Arrivaltime}(C)$ 
2  $C'_{Area} \leftarrow \text{Untere\_Schranke\_Area}(C)$ 
3 foreach Output  $o$  do
4    $\alpha[o] \leftarrow \lambda \cdot AT_{C'_{AT}}(o) + (1 - \lambda) \cdot AT_{C'_{Area}}(o)$  //Finale AT Schätzung
5 foreach Highfanoutknoten  $v \in V(C)$  do
6    $\text{estim\_small}(v) \leftarrow \text{neg\_Slack}(v, C'_{Area}, \alpha)$ 
7    $\text{estim\_fast}(v) \leftarrow \text{neg\_Slack}(v, C'_{SNS}, \alpha)$ 
8    $\text{neg\_Slack\_Schätzung}(v) \leftarrow \lambda \cdot \text{estim\_fast}(v) + (1 - \lambda) \cdot \text{estim\_small}(v)$ 
9
10 ...
11
12 if  $v$  ist Highfanoutknoten then
13    $min \leftarrow \min_{\text{Kandidat } k \text{ auf } v} \{|\text{neg\_slack}(k) - \text{neg\_slack\_schätzung}(v)|\}$ 
14   foreach Kandidat  $k$  auf  $v$  do
15     if  $|\text{neg\_Slack}(k) - \text{neg\_Slack\_Schätzung}(v)| \neq min$  then
16        $\text{lösche } k$ 
17   if  $\text{neg\_Slack\_Schätzung}(v) = 0$  then
18      $\text{lösche\_alle\_Kandidaten\_bis\_auf\_den\_Kleinsten}(v)$ 

```

Die Schritte 1-9 werden an den Anfang der Heuristik für das TM mit Konvexkombination gesetzt. Die Schritte 13-18 ersetzen die Schritte 12-16 der Heuristik.

Es ist ausreichend, alle möglichen Matche der Knoten nur einmal zu errechnen und diese Werte sowohl für die untere Schranke als auch für die Heuristik zu nutzen. Mit diesem Trick lassen sich beide Schranken in kurzer Zeit errechnen. Genauere Informationen zur Laufzeit und Güte der Premapping Routinen befindet sich in Kapitel 6.

Diese Routine liefert in der Praxis sehr gute Ergebnisse, da durch die Berechnung der Schranken das Potenzial eines Knotens errechnet wird (wie klein oder schnell Kandidaten dieses Knotens werden können). Mithilfe des Tradeoffparameters errechnet sich hieraus ein geschätzter Kostenwert. Es wird nur der Kandidat ausgewählt, welcher die Differenz zu dieser Schätzung minimiert. Falls die Slack Schätzung Null ist, bleiben alle evtl. mehrere Kandidaten erhalten. Hier wird nur der Kleinste gespeichert, da dieser die anderen aus der Sicht des Slacks dominiert.

4.2.3 Erweitertes Premapping durch Schätzen

Dieses Kapitel beschäftigt sich damit, die gerade beschriebene Premapping Methode zu erweitern.

Das Premapping durch Schätzen errechnet einen kleinen und einen schnellen Circuit und daraus das Potenzial eines jeden Knoten bezüglich der Verbesserung hinsichtlich Größe und Geschwindigkeit. Der folgende Ansatz findet besonders Delay kritische Pfade in dem Circuit und ordnet dessen Knoten eine Kritikalität zu. Diese gibt Auskunft über den Einfluss eines Knoten bezüglich des Delays seiner Nachfolger. Das Erweiterte Premapping durch Schätzen verknüpft das Potenzial und die Kritikalität eines jeden Knoten bei der Auswahl eines geeigneten Kandidaten.

Jeder Knoten g in C erhält eine Kritikalität $krit_g$. Diese wird für jeden Knoten mit Null initialisiert.

Die Kritikalität wird mit Hilfe von Dijkstras Algorithmus errechnet. Dieser ist, da ein Circuit C ein kreisfreier gerichteter Graph ist, in der Lage längste Wege, bezüglich Kantengewichte, in C zu finden. Hierbei ist das Gewicht einer Kante das Delay dieser inklusive des Gate-Delay des Ende dieser Kante liegenden Gates (zuzüglich des evtl. am Input liegenden Inverters). Das Gewicht der, von einem Input von C , ausgehenden Kanten, wird noch um die Arrivalttime des entsprechenden Inputs erhöht.

Dijkstra wird nun für jeden Output o aufgerufen und $krit_g$ wird für jeden Knoten, welches auf einem längsten Weg zu o liegt, um eins erhöht.

Für jeden Knoten ist nun gespeichert für wie viele Outputs, dieser auf einem kritischen Pfad liegt. Jeder Kritikalitätswert wird nun durch die Anzahl der Outputs dividiert, wodurch diese Werte in $[0, 1]$ liegen.

Ziel ist es nun auf Grund von der Information der Kritikalität an jedem Highfanoutknoten v einen Faktor μ_v zu berechnen, mit welchen im Technology Mapping Algorithmus der Folgende Kandidat k für v gewählt wird:

$$\min_{\text{Kandidat } k \text{ auf } v} \{|neg_slack(k) - (1 + \mu_v)neg_slack_schätzung(v)|\}$$

Für den Fall, dass die Slack-Schätzung Null ist, wird auch hier der kleinste Kandidat mit Slack Null gewählt.

Der restliche Teil dieses Kapitel beschreibt eine Funktion für die Berechnung von μ_v .

Im Folgenden sei $f : [0, 1] \rightarrow [-\mu_{max}, \mu_{max}]$ die μ_v für jeden Knoten v , gegeben $krit_v$, berechnende Funktion. Hierbei sei μ_{max} eine obere Schranke für den Betrag von μ_v . Diese wird heuristisch festgelegt.

Um die Auswahl des Kandidaten sinnvoll zu beeinflussen gilt für f und zwei Kritikalitäten $krit_1$ und $krit_2$:

$$f(krit_1) > f(krit_2) \Leftrightarrow krit_1 < krit_2$$

. Anderweitig würde für einen bereits Delay kritischen Knoten ein langsamer Kandidat gewählt als das Premapping durch Schätzen vorschlägt. Dies führt sehr wahrscheinlich zu äquivalenten Circuit mit schlechterer Güte.

Sei $krit_{av} \in \mathbb{R}$ das arithmetische Mittel aller errechneter Kritikalitäten.

Sei f wie folgt definiert:

$$f(x) := \begin{cases} \frac{\mu_{max}}{krit_{av}-1} \cdot x + \mu_{max}(\frac{1}{1-krit_{av}} - 1) & \text{wenn } x \geq krit_{av} \\ \frac{-\mu_{max}}{1-krit_{av}} \cdot x + \mu_{max}(\frac{1}{1+krit_{av}} - 1) & \text{wenn } x < krit_{av} \end{cases}$$

Somit ist f aus zwei linearen Funktionen aufgebaut und lässt sich dadurch in konstanter Laufzeit errechnen. Abbildung ?? visualisiert f .

Das Erweiterte Premapping durch Schätzen ist in der Lage die Lösungen, welche mit dem Premapping durch Schätzen errechnet wurden, zu verbessern, jedoch geschieht dies nicht regelmäßig. Des Weiteren verursacht der Dijkstra Algorithmus einen nicht zu vernachlässigen Anstieg der Laufzeit. Somit bleibt es offen f, μ_{max} und $krit_{av}$ so zu wählen, dass eine regelmäßige Verbesserung eintritt. Des Weiteren würde ein Dijkstra Aufruf für jeden Knoten oder eine Beachtung von allen nahezu längsten Wegen eine größere Fülle an Informationen liefern. Da es jedoch nur exponentiell beschränkt viele Wege in C gibt führt dies wahrscheinlich zu markanten Laufzeiteinbußen.

Diese Premapping Methode wird in Kapitel 6, aufgrund der nur unregelmäßig auftretenden Verbesserung, nicht behandelt.

4.3 Preprocessing

Die Möglichkeiten durch das Matchen sind prinzipiell vielfältig, jedoch bei Gates p mit $|inputs(p)| = fanin_{max}$ auf das beliebige Invertieren der Inputs und des Outputs beschränkt.

Um diesem Problem aus dem Weg zu gehen, ist es sinnvoll, vor dem Technology Mapping Algorithmus jedes Gate mit mehr als zwei eingehenden Kanten durch einen kleinen Subcircuit, bestehend aus zwei Input Gates, zu ersetzen. Dies ist immer möglich, da jede logische Funktion nur mit Hilfe von NAND2 und INV Gates realisierbar ist ([Pos41]) und auf jedem realen Chip standardmäßig ein AND oder NAND sowie ein OR oder NOR in der Library vorhanden sind. INV Gates sind fester Bestandteil jeder realen Library.

Dabei werden die Gates nach folgender Routine zerlegt. **hier das huffman coding und ein Beispiel mit einbringen und decomposing begriff einführen**

Der Vorteil des Decompose ist, dass die Möglichkeiten des Technology Mappings deutlich erweitert werden. Jedoch werden für ein AND4 Gate beispielsweise 3 AND2 Gates eingesetzt, was dazu führt, dass sich meist die Kosten des Ausgangscircuits verschlechtern.

Eine ausführliche Analyse der Vor- und Nachteile des Decomposens finden sich in Kapitel 6.

4.4 Weitere Optimierungskriterien

Das Technology Mapping arbeitet im Chip-Design auf realen Instanzen. Dadurch kommen zu den bereits vorgestellten weitere mögliche Optimierungskriterien hinzu. Es handelt sich hierbei um Kriterien, welche in die Kostenfunktion mit eingebracht werden können und somit während des Technology Mappings optimiert werden. Des Weiteren verursacht der neu implementierte Circuit Kosten, welche im Technology Mapping nicht beachtet wurden, für die es jedoch einen übermäßigen Anstieg zu vermeiden gilt. Ein Beispiel hierfür sind die bereits erwähnten zu langen Kanten.

Darüber hinaus sind viele Gates teilweise symmetrisch aufgebaut (Bsp.: AND, OR ...) und die Signale der Inputs brauchen unterschiedlich lange zum Output des Gates. Daraus folgt, dass durch Permutierung von Teilmengen der Inputs, Geschwindigkeitsvorteile geschaffen werden können.

4.4.1 Pinabhängiges Delay

Bis hierhin war das Delay eines Gates als eine nicht negative reelle Zahl definiert. Die meisten Gates besitzen jedoch mehr als nur einen Input. Die Signale der Inputs brauchen nicht alle dieselbe Zeit um zum Output zu gelangen. Logisch werden die Signale der Inputs zwar alle miteinander verrechnet, jedoch geschieht dies physikalisch nicht gleichzeitig und somit müssen nicht alle Signale zur selben Zeit an den Inputs anliegen.

Die Möglichkeit eines Signals später an einem Input des Gates ankommen zu können lässt sich durch einen kleineren Delaywert, spezifisch für diesen Input, realisieren. Denn wenn das Signal schneller durch das Gate gelangen kann, so braucht es auch nicht so früh vorhanden zu sein, wie die anderen. Von nun an ist das Delay eines Gates g : $d_g \in \mathbb{R}_{\geq 0}^{arity(g)}$. Für das Technology Mapping ist dies eine weitere Möglichkeit der Verbesserung; denn viele Gates der Library besitzen mindestens eine Teilmenge von Inputs, welche logisch symmetrisch aufgebaut sind. Diese lassen sich beliebig permutieren. Durch die unterschiedlichen Delay-Eigenschaften der Inputs kann eine solche Permutierung das Delay des Outputs verbessern. Aus diesem Grund ändert sich die AT eines Knotens wie folgt:

Definition 4.2. AT mit pinabhängigen Delay:

Sei C ein Circuit und $v \in V(C)$.

Die AT von v mit pinabhängigen Delay ist wie folgt definiert:

$$AT_p(v) := \max_{i \in inputs(v)} \{d_{gate(v),i} + \mathbb{1}_{\{inv_g(i)\}} d_i + AT_p(i) + d_{w(k,i)}\}.$$

Im Folgenden sei mit AT immer das pinabhängige Delay gemeint.

In einem Match ist diese Information bereits abgespeichert, da die Inputs eines Matches mit einer Bijektion an Knoten des Circuits geknüpft werden. Es wird ein Kandidat für jede mögliche Permutation der Inputs gespeichert, falls dieser nicht dominiert ist.

Nach aktuellem Stand gilt $fanin_{max} \leq 4$. Das ist klein genug, um auch bei der Heuristik die max $fanin_{max}$ Permutation bei der Wahl eines Matches in Betracht zu ziehen.

4.4.2 Power-Optimierung

Jedes Gate besitzt neben seinen spezifischen Eigenschaften bezüglich Area und AT noch weitere physikalische Eigenschaften.

An einem Transistor liegt immer eine Spannung an. Daraus folgt, dass dieser auch ohne zu schalten Energie (power) verbraucht. Diese lässt sich einteilen in static power und dynamic power. Hierbei bezeichnet static power den Energieverbrauch unabhängig von der Benutzung des Transistors. Ein Transistor verbraucht jedoch mehr Energie, wenn er schaltet. Daraus folgt, dass der Energiebedarf abhängig vom Grad der Nutzung dieses Transistors ist. Dieser variable Energieverbrauch wird auch als dynamic power bezeichnet. Da ein Gate aus einer logischen Verknüpfung von Transistoren besteht, besitzt es ebenfalls einen static power Wert. Da der dynamic power Wert eines Gates abhängig von der aktuellen Implementierung des Chip ist, ist es schwer, diesen unabhängig von einem Circuit zu errechnen. Aus diesem Grund beschränke ich mich im Folgenden auf die static power.

Die static power eines Gates korreliert sehr stark mit der physikalischen

Größe dieses Bauteils. Aus diesem Grund lassen sich in den oben vorgestellten Algorithmen die Area Daten durch die static power Werte ersetzen. Der Circuit wird dadurch hinsichtlich Geschwindigkeit und Energiebedarf optimiert.

Static power lässt sich auch zusätzlich zu Area in die Kostenfunktion einbauen. Dies würde bedeuten, dass die Kosten eines Kandidaten k das Tripel $area(k), AT(k), static_power(k)$ sind, wobei $static_power(k)$ ähnlich wie $area(k)$ errechnet wird. Dies führt jedoch zu einer noch schlechteren Vergleichbarkeit von Kandidaten gleicher und verschiedener Knoten. Aus diesem Grund wird auf die Optimierung aller drei Kriterien zusammen verzichtet. In Kapitel 6 finden sich weitere Ausführungen über die Auswirkungen von diesem Austausch in der Kostenfunktion.

5 Resource Sharing

In diesem Kapitel wird ein weiterer Ansatz für eine Heuristik vorgestellt. Da dieser noch nicht implementiert wurde, wird dieser Ansatz in Kapitel 6 nicht berücksichtigt.

Es folgt die allgemeine Definition des Resource Sharing Problems und darauf basierend eine Heuristik, welche sich dieses Problem zunutze macht.

5.1 Definition und Lösungsansatz des Resource Sharing Problems

Eine Instanz des Problems besteht aus einer endlichen Menge von Kunden \mathcal{C} und einer konvexen Menge \mathcal{B}_c für jeden Kunden $c \in \mathcal{C}$. Diese wird auch Block genannt. Des Weiteren sei \mathcal{R} eine endliche Menge von Ressourcen. Gegeben seien des Weiteren Funktionen $g_c : \mathcal{B}_c \rightarrow \mathbb{R}^{\mathcal{R}_+}$, welche für jeden Kunden und jedes Element des entsprechenden Blocks einen Ressourcenverbrauch angeben.

Ziel des Resource Sharing ist es, jedem Kunden $c \in \mathcal{C}$ ein Element $b_c \in \mathcal{B}_c$ zuzuordnen, sodass

$$\lambda^* := \inf \left\{ \max_{r \in \mathcal{R}} \sum_{c \in \mathcal{C}} (g_c(b_c))_r \mid b_c \in \mathcal{B}_c \forall c \in \mathcal{C} \right\}$$

möglichst nah erreicht wird. Mit anderen Worten ist es das Ziel den maximalen Ressourcenverbrauch zu minimieren.

5.1.1 Lösungsansatz

Der in [MRV11] ausführlich beschriebene Lösungsansatz für das Resource Sharing iteriert über die Kundenmenge und errechnet in jeder Iteration mit einem Orakel und Preisen ω_r für alle Ressourcen $r \in \mathcal{R}$ ein Block Element

für jeden Kunden, sodass der durch die Preise gewichtete maximale Ressourcenverbrauch minimiert wird.

Das Orakel $f_c : \mathbb{R}_+^R \rightarrow \mathcal{B}_c$, auch Blocklöser genannt, wird als gegeben betrachtet und errechnet für gegebene Preise ω für jedes Element $c \in C$ ein $b_c \in \mathcal{B}_c$ sodass gilt: $\omega^T g_c(b_c) \leq \sigma \inf\{\omega^T g_c(b) | b \in \mathcal{B}_c\}$ mit einem Approximationsfaktor $\sigma \geq 1$.

Nachdem der Blocklöser aufgerufen wurde, wird geprüft ob verknüpfte Ergebnisse der bisherigen Iterationen eine zulässige Lösung der Resource Sharing Instanz liefert. Ist dies nicht der Fall so werden die Preise angepasst die nächste Iteration mit den neuen Preisen gestartet. Sei y_r die Summe des Ressourcenverbrauchs der Ressource r der vorangegangenen Iterationen. Die Preise werden am Anfang einer Iteration durch $\omega_r \rightarrow \omega_r \cdot e^{\gamma \cdot y_r}$ aktualisiert. Hierbei ist $\gamma > 0$ beliebig, hat jedoch Einfluss auf die Güte der Lösung des Algorithmus.

Ist der Algorithmus fertig so liegt das Ergebnis für jeden Kunden in Form einer Konvexkombination von Elementen des zugehörigen Blocks vor. Da jeder Block konvex ist repräsentiert diese Kombination wieder ein Element des Blocks.

Angenommen ein Block ist nicht konvex, so muss eine Lösung des Resource Sharing noch zu einem geeigneten Element des zugehörigen Blocks gerundet werden.

Das Resource Sharing Problem lässt sich mit einem FPTAS lösen. Ein Beweis hierzu befindet sich in [MRV11].

5.2 Resource Sharing und Technology Mapping

Der folgende Algorithmus wurde angelehnt an [DHHS18] entwickelt.

Das Technology Mapping Problem wird wie folgt durch eine Instanz des Resource Sharing modelliert.

Die Technology Mapping Instanz bestehe aus einem Circuit G ohne teilweise überflüssige Subcircuits und gegebenen RAT 's an den Outputs von G .

Gesucht sei ein zu G äquivalenter Graph G^* , dessen Outputsignale vor der entsprechenden RAT ankommen und $Power(G^*) \leq B$. Sei B vorerst gegeben.

Sei T_G der Timing Graph von G . **muss ich den erklären?** Dieser modelliert alle Delays der Knoten und Kanten von G . Ein Beispiel eines solchen Graphen befindet sich in Abbildung ??.

Sei nun $\mathcal{C} := \{G\}$, \mathcal{B}_G die Menge der nicht dominierten Circuit Kandidaten auf G und \mathcal{P}_G die Menge der inklusionsmaximalen Wege im Timing Graphen von G . Die Signale der Outputs eines Graphen kommen genau dann vor der RAT an, wenn alle $P \in \mathcal{P}$ vor der RAT des zu P gehörenden Outputs ankommen.

Somit lässt sich wie folgt festlegen: $\mathcal{R} = \{Power\} \cup \mathcal{P}_G$.

Das Orakel in jeder Iteration wird durch eine noch zu beschreibende Technology Mapping Heuristik ersetzt.

Das Ziel dieses Orakels ist, den gewichteten Ressourcenverbrauch

$$\omega_{Power} \frac{Power(G)}{B} + \sum_{P \in \mathcal{P}} \omega_P \frac{\sum_{e \in E(P)} Delay_e(G)}{RAT_P} \quad (1)$$

zu minimieren. Hierbei ist RAT_P die RAT des Outputs von P .

Durch diese Modellierung des Resource Sharing ergeben sich die folgenden Probleme, welche in den folgenden Unterkapiteln gelöst werden:

- $|\mathcal{P}|$ ist nur exponentiell beschränkt. Daraus folgt, dass die explizite Speicherung jedes Ressource Preises zu einem nicht polynomiellen Algorithmus führen würde.
- Die Technology Mapping Heuristik liefert einen zu G äquivalenten Circuit G' . Im Allgemeinen gilt jedoch $\mathcal{P}_G \neq \mathcal{P}_{G'}$. Somit lässt sich keine Ressourcenverbräuche von G' auf den Ressourcen vor dem Technology Mapping angeben.
- Der Block \mathcal{B}_G ist nicht konvex, wodurch noch eine geeignete Routine zur Rundung anzugeben ist.

5.2.1 Kantenpreise

Die Menge \mathcal{P}_G für einen Circuit G ist nur exponentiell beschränkt. Dadurch ist es nicht praktikabel, Preise für jede Pfad Ressource zu speichern. Es ist jedoch möglich, die Preise aller Pfade durch Gewichte auf den Kanten von T_G zu repräsentieren.

Seien $(\omega_P)_{P \in \mathcal{P}_G}$ Preise für die Pfad-Ressourcen von G . Setze $\omega_e := \sum_{P \in \mathcal{P}: e \in P} \omega_P$.

Die Minimierung von (1) durch das Orakel ist nun äquivalent der Minimierung von

$$\omega_{Power} \frac{Power(G)}{B} + \sum_{e \in E(T_G)} \frac{Delay_e(G)}{RAT_P}. \quad (2)$$

Des Weiteren gilt für $(\omega_e)_{e \in T_G}$ die Flusserhaltung $\sum_{e \in \delta^-(v)} \omega_e = \sum_{e \in \delta^+(v)} \omega_e$.

Der Preis eines Pfades lässt sich nun in linearer Zeit errechnen. Ein Beweis dieser Aussage befindet sich in [Häh15].

Neben den Preisen der Pfade lassen sich auch die Summe der Ressourcenverbräuche y_p eines Pfades in Verbräuche der Kanten des Pfades aufteilen; hierbei gilt:

$$y_e := \frac{1}{RAT_P} \sum_{k=1}^i \xi^{(k)} \cdot usg^{(k)}(e)$$

Hierbei sind $\xi^{(k)}$ in Iteration k des Algorithmus errechnete Werte in $[0, 1]$ und $usg^{(k)}(e)$ entspricht dem der Kante e durch den Algorithmus im folgenden Unterkapitel zugewiesenen Delay.

Auf Basis der Verbräuche auf jeder Kante, lassen sich im Resource Sharing Algorithmus die Pfadpreise, repräsentiert durch Kantenpreise, aktualisieren.

Der Algorithmus Edge Weights aktualisiert die Kantenpreise bei gegebenen Ressourcenverbräuchen $(y_e)_{e \in E(T_G)}$ und γ , in linearer Laufzeit und stammt aus [DHHS18]. Hierbei gilt für alle Kantenpreise: $\omega_e > 0$.

5.2.2 Ressourcenproblem

Das Technology Mapping Orakel gibt einen optimierten zu G äquivalenten Circuit G' zurück. Dieser besitzt jedoch neue Pfade und somit eine andere Ressourcenmenge als G . Dieses lässt sich durch eine Funktion lösen, welche die Ressourcenverbräuche $(y_e)_{e \in E(T_G)}$ auf Grundlage von G' aktualisiert. Anschließend lassen sich die Kantenpreise mit Hilfe des Algorithmus aus Kapitel 5.2.1 aktualisieren und die nächste Iteration des Resource Sharing kann gestartet werden.

Es gilt nun die Ressourcenverbräuche entsprechend anzupassen. Dies erfolgt durch Algorithmus:

Addiere zu allen Kanten, welche in G' noch existieren, den entsprechenden Ressourcenverbrauch. Wenn über eine Kante gematcht wurde, so wird das Delay aller Komponenten des Matches durch die Kanten im neuen Gate ersetzt. Dementsprechend bleibt der Ressourcenverbrauch von verschwundenen Kanten im Timing Graphen von G unverändert. Das Delay der eingehenden Kanten entspricht automatisch dem der längeren Kante zum neuen Gate. Das Delay neu hinzugekommener Inverter an Inputs werden auf das Delay der darüberliegenden Kante des Gates addiert. Die Vorgehensweise dieses Algorithmus wird durch Abbildung ?? veranschaulicht.

Wie bereits in Kapitel 3.2.1 erläutert, ist für die in dieser Arbeit verwendeten Technology Mapping Algorithmen, das Klonen von Gates nicht erlaubt. Lockert man diese Vorgabe, so bleibt es offen zu zeigen, wie Ressourcenverbräuche von neu entstandenen Gates und Kanten in die Aktualisierung der Verbräuche von G eingebracht werden.

5.2.3 Technology Mapping Orakel

Das Orakel ist ein Technology Mapping Algorithmus mit den folgenden Besonderheiten:

Zugrunde gelegt wird die Technology Mapping Heuristik aus Kapitel 3.8 mit dem Premapping durch Schätzen und erweitert für Circuits mit mehreren Outputs.

Damit dieser Algorithmus, eine bezüglich der Kantenpreise optimierte Lösung berechnen kann, werden diese wie folgt in dem Algorithmus integriert.

Das Delay einer Kante von G wird multipliziert mit dem entsprechenden Preis dieser Kante. Dies gilt ebenfalls für das pinabhängige Delay eines Gates. Das Delay jedes Inputpins eines Gates in G zu seinem Output lässt sich ebenfalls eindeutig einer Kante aus T_G zuordnen. Mit dessen Preis wird auch dieses Delay multipliziert. Des Weiteren werden die Powerwerte der Gates mit ω_{Power} multipliziert. Da $\omega_e > 0$ für jede Kante gilt, führt dies zu positiven Delay-Werten.

Durch diese Änderungen minimiert das Orakel heuristisch die Kostenfunktion (2).

5.2.4 Algorithmus

Im Folgenden wird ein Resource Sharing Algorithmus zur Lösung des Technology Mappings vorgestellt.

Algorithmus : Resource Sharing Algorithmus für das TM	
<hr/>	
Input : Instanz G des TM-Problems ohne teilweise überflüssige Subcircuits; ein Power-Budget B , $\gamma > 0$, $I \in \mathbb{N}$	
Output : Konvexkombination von Kandidaten auf der TM-Instanz	
1	$k[] \leftarrow \emptyset, \quad \phi[] \leftarrow \emptyset, \quad \Xi \leftarrow 0$
2	$y_e \leftarrow 0 \quad \forall e \in E(T_G), \quad y_{Power} \leftarrow 0$
3	for $i = 1, \dots, I$ do
4	$\omega_{Power} \leftarrow e^{\gamma \cdot y_{Power}}$
5	$(\omega_e)_{e \in E(T_G)} \leftarrow \text{Edge_Weights}(\gamma, (y_e)_{e \in E(T_G)})$
6	$k[i] \leftarrow \text{TM_Oracle}(\omega)$
7	$\xi \leftarrow \min\{\frac{B}{Power(k[i])}, \frac{RAT_P}{Delay(P)} \mid P \in \mathcal{P}_{k[i]}\}$
8	if $\xi \leq 1$ then
9	return $k[i]$
10	$(y_e)_{e \in E(T_G)} \leftarrow \text{Edge_Resources}(k[i], (y_e)_{e \in E(T_G)}, \xi)$
11	$\phi[i] \leftarrow \xi, \quad \Xi \leftarrow \Xi + \xi$
12	return $k, \frac{\phi}{\Xi}$

Wenn der Algorithmus nicht in Schritt 9 endet, so nehme man den Kandidaten in k mit dem größten Faktor in $\frac{\phi}{\Xi}$. Es bleibt offen eine Rounding Methode zu beschreiben, welche jeden berechneten Kandidaten berücksichtigt.

5.2.5 Abschließende Bemerkungen

Im bisherigen Verlauf dieses Kapitels wurde B als gegeben vorausgesetzt. Es lässt sich jedoch B durch eine binäre Suche und einem Aufruf des Resource Sharing Algorithmus für jede Iteration der Suche ersetzen. Dies ergibt eine, wenn möglich RAT -einhaltende, Lösung mit möglichst kleinem Power-Verbrauch.

Des Weiteren lässt sich auch mit einem durch Technology Mapping umgebauten Circuit in der nächsten Iteration des Resource Sharing weiterrechnen, jedoch müssen dann die Preise und Ressourcen entsprechend angepasst werden. Dies ist für den gesamten Circuit nicht sinnvoll, da sonst alle Preise evtl. ihre Gültigkeit verlieren. Sehr kritische Subcircuits lassen sich jedoch mit entsprechenden Veränderungen an Ressourcen und Preisen austauschen. Diese kritischen Bereiche lassen sich auch mit Hilfe des Preprocessings verändern und in der nächsten Iteration separat bearbeiten. Die Kantenpreise können, aufgrund der Flusserhaltung, leicht an den neuen Graphen übertragen werden, jedoch ist zu diesem Zeitpunkt noch offen, wie die Summe der Ressourcenverbräuche aufgeteilt auf die Kanten in die neue Umgebung transformiert werden müssen. Für Graphen mit polynomiell beschränkter Pfadmenge, beispielsweise Graphen mit konstant vielen Highfanoutknoten, lässt sich dies jedoch auch über die Pfadwerte direkt in polynomieller Zeit ändern.

Die Definition der Ressourcen und Kunden wurde so allgemein gehalten, dass in einer Iteration des Resource Sharing Algorithmus das Technology Mapping sehr einfach durch eine andere die Logik optimierende Routine ausgetauscht werden kann. Abhängig von den aktuellen Preisen und den bisher errechneten Kandidaten, ist es möglich zum Beispiel das Technology Mapping Orakel mit dem in [DHHS18] beschriebenen Gate Sizing Algorithmus zu ersetzen. Dadurch ist eine deutlich höhere Flexibilität bei der Logik-Optimierung eines Circuits.

Ob der Resource Sharing Algorithmus für das Technology Mapping polynomiell implementierbar ist hängt nur von dem gewählten Technology Mapping Algorithmus ab, denn alle anderen Schritte sind wie bereits in Kapitel 5.1.1 beschrieben, polynomiell implementierbar.

6 Qualitäts- und Laufzeit-Analyse

In diesem Kapitel werden die vorgestellten Heuristiken bezüglich Laufzeit und Güte analysiert. Vor dieser Analyse ist es jedoch Notwendig ein paar Angaben zur Struktur realer Instanzen zu machen, um die Ergebnisse der Algorithmen besser einordnen zu können.

6.1 Struktur realer Instanzen

Die Instanzen eines Chips sind alle maximal zusammenhängenden, mit Technology Mapping vollständig bearbeitbaren, Circuits eines Chips. Da auf einem Chip beispielsweise durch Register gerichtete Kreise entstehen, oder Bauteile existieren, welche nicht in der Library vorhanden sind, lässt sich der Logik-Graph eines Chips nicht vollständig mit dem Technology Mapping Algorithmus verarbeiten.

Das Chipdesign besteht aus sehr vielen Routinen, welche aus einem Bauplan einen produzierbaren Chip designen. Auf diesem Weg gibt es viele Zwischenstände (Snapshots genannt). Alle getesteten Chips wurden auf dem Stand des selben Snapshots bearbeitet. Dadurch lassen sich die Verbesserungen von Instanzen durch die Algorithmen miteinander vergleichen, auch wenn sie von verschiedenen Chips stammen.

Die folgenden Angaben betrachten alle Instanzen zusammengesetzt zu einem Circuit. Insgesamt wurden 1107 Instanzen getestet.

Sei H_C die Menge Highfanouknoten, welche keine Inputs sind, eines Circuits C und NI_C die Menge aller Knoten abzüglich der Input-Knoten von C . Der durchschnittliche Anteil von H_C an NI_C beträgt 30,95%. Die Menge der Highfanouknoten, welche ebenfalls Inputs sind, wurde hier nicht beachtet, da diese nur einen möglichen Kandidaten besitzen, und somit nicht zu dem Problem des exponentiellen Klassenwachstums beitragen. Diese Werte besitzen bei den größeren Instanzen eine kleine Varianz. Aus diesem Grund werden die Circuits im Folgenden nur anhand ihrer Knotenzahl analysiert.

Der Anteil von Outputs an der Gesamtzahl von Knoten eines Circuits liegt bei 15,26%. Von allen Outputs haben 4,13% noch weitere Nachfolger im gleichen Circuit. Outputs mit Nachfolgern werden ebenfalls auf einen Kandidaten beschränkt **oben erwähnt ?**, da sie zu den offenen Knoten darauffolgender Knoten gehören. Da diese nur einen solch geringen Anteil an allen Outputs besitzen, werden nur wenige Outputs so früh festgelegt. Auch hier wurden Outputs, welche auch Inputs sind, aus den gleichen Gründen wie oben, nicht betrachtet.

Die Inputs eines Circuits machen durchschnittlich 16,73% an der Gesamtzahl der Knoten aus. Daraus folgt, dass etwa jeder achte Knoten nur einen Kandidaten trägt. Dies ist bei der Interpretation der folgenden Analysen zu beachten.

$ V(C) $	#Instanzen	%
>10	693	63%
>100	385	32%
>1000	189	17%
>10000	19	1,7%
>20000	12	1%
>30000	9	0,8%
>40000	4	0,04%

Abbildung 13: Größverteilung der getesteten Instanzen

Abbildung 13 zeigt die Verteilung der getesteten Instanzen hinsichtlich der Größe ihrer Knotenmenge.

Kandidaten menge kann hier noch hin also menge der Kandidaten abh von Knotenmenge bzw anzahl highfanoutknoten und abh von den oben genannten zusatzfeatures (zb pinabh. delay, Library)

und dass vielleicht die varianz (anhand eines bildes beweisen?) des hoghfanout anteil ziemlich klein ist und somit Knotenmenge und highfanoutmenge ansich ausreichend aussagekräftig sind

6.2 Analyse der Ergebnisse

6.2.1 Tradeoffparameter

Die Werte von Area, SNS, WS und Energieverbrauch eines Circuits liegen nicht unbedingt in derselben Größenordnung. Dies kann zur Folge haben, dass selbst ein vielfacher Verbrauch einer Ressource sich in den Kosten des Circuits-Kandidaten nicht bemerkbar macht. In der Praxis liegen Power, Area oder Arrivalttime üblicherweise um mehrere Zehnerpotenzen auseinander.

Der Unterschied zwischen dem Verbrauch eines Gates mit dem geringsten zu dem Gate mit dem höchsten Ressourcenverbrauch liegt bei maximal Faktor 10.

Der Unterschied des Verbrauchs einer Ressource von dem Gate welches den geringsten Verbrauch in Anspruch nimmt zu dem mit dem größten Verbrauch liegt in der Regel bei maximal einem Faktor 10. Der Ressourcenverbrauch eines Gates wird nun durch dividieren durch den Verbrauch eines sparsamen Gates normalisiert; dann liegen alle Ressourcenverbräuche in derselben Größenordnung.

Im Folgenden wird ausgehend von der Technology Mapping Heuristik ohne Präprozessing, und mit dem Premapping durch Schätzen der Einfluss des Tradeoffparameters λ , abhängig von einer Optimierung nach Power oder Area, auf die Änderung der Kosten untersucht. Die Kostenfunktion ist hierbei $\lambda SNS(C) + (1 - \lambda)Area(C)$ bzw. $\lambda SNS(C) + (1 - \lambda)Power(C)$

Abbildung 14 veranschaulicht die durchschnittliche prozentuale Verbesserung der Kosten abhängig von λ .

Hier ist nur eine schwache Abhängigkeit bezüglich des Tradeoffparameters zu beobachten. Dies liegt daran, dass im Schnitt für jeden dritten Knoten ein Kandidat geschätzt wird. Des Weiteren liegen SNS und $Area$ oft in derselben Größenordnung, wodurch ein schnellerer Kandidat mit nahezu den gleichen Kosten geschätzt wird, wie ein kleinerer und somit unabhängig des λ , sehr kostenähnliche Kandidaten gewählt werden; wodurch sich die Kosten des äquivalenten Circuits sich in Abhängigkeit des λ nur gering verändern. Unabhängig von λ zeigt die Abbildung jedoch auch ein enormes Verbesse-

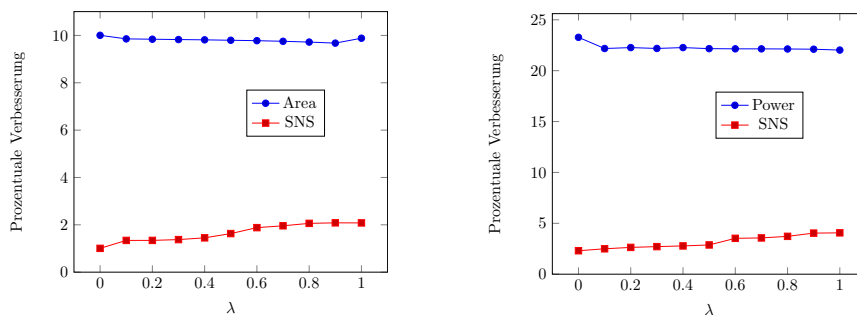


Abbildung 14: Einfluss von λ auf SNS Area und Power

rungspotenzial des Circuits in *SNS*, *Area* und besonders *Power*.

In folgenden Tests wird $\lambda = ?$ gesetzt, da dies in den getesteten Instanzen die größte Verbesserung im Schnitt gebracht hat.

Offenbar bringen 0,lala 0,trlala die besten Verbesserungen. Im Folgenden sei λ durch diese Werte vorgegeben.

genauer daruaf eingehen warum das lambda egal ist und auf Lukas schonere Bilder verweisen

6.2.2 Premapping

6.2.3 Preprozessing

inklusive der graphen gröÙe vergleiche

6.2.4 Vergleich von Power und Area

6.2.5 Gütevergleich kleiner optimal "gel instanzen

6.2.6 Verhalten weiterer Kosten

die anfallenden kosten die nicht im Technology Mapping gemessen werden beobachten und in der future work dran anschließen

6.2.7 Zusammenfassung?

oder am ende nur ein laufzeit güte tradeoff mit eigenem Unterkapitel ?

6.3 Laufzeitanalyse

Aus der theoretischen Laufzeitschranke aus Kapitel 3.8 folgt, dass die Laufzeit maßgeblich von der Menge der Highfanoutknoten abhängt. Andere Faktoren wie das Aufteilen von Gates, die Größe der Library oder die Wahl der

Premapping subroutine spielen ebenfalls eine wichtige Rolle. Dies verdeutlicht die Übersicht ??.

Bezüglich der Größe der Library lassen sich die Chips, mit Ausnahme weniger Unterschiede in zwei Gruppen einteilen. Die markierte Teilmenge der Abbildung ?? entspricht der Menge der (so genannten) komplexen Gates. Ein Chip lässt sich in der Praxis, abhängig davon, ob er die komplexen Gates grundsätzlich erlaubt, einer der zwei Gruppen zuordnen. Diese Unterscheidung findet sich ebenfalls in Abbildung ??.

6.3.1 Globale Laufzeitanalyse

welche größenordnungen sind überhaupt lösbar

6.3.2 Lokale Laufzeitanalyse

wie unterscheiden sich die einzelnen Varianten in der Laufzeit?
oder lässt sich beides gut in einem bild erkennen ?

hier auch die decompose laufzeit vgl mit rein ?

6.3.3 Laufzeitvergleich kleiner optimal "gel instanzen

6.4 Vergleich von Güte und Laufzeit

6.4.1 Kleine Instanzen

6.4.2 Allgemein

wahrscheinlich ist der unterschied bei decompose und ohne am größten den mit der besten premapping methode durchführen

7 Fazit und Ausblick

8 Literaturverzeichnis

- [DHHS18] S. Daboul, N. Hähnle, S. Held, and U. Schorr. Provably fast and near-optimum gate sizing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, February 2018.
- [Elb17] Lucas Elbert. Approximationsalgorithmen für das Technology-Mapping, 2017.
- [Häh15] N. Hähnle. Time-cost tradeoff and steiner tree packing with multiplikative weights. *Technical report no. 1511115, Research Institute for Discrete Mathematics, University of Bonn*, 2015.
- [HNKF18] David A. Hounshell Hassan N. Khan and Erica R. H. Fuchs. Science and research policy at the end of moore’s law. *Nature Electronics*, 1, January 2018.
- [Kau87] K. Keutzer. Dagon: Technology binding and local optimization by dag matching. In *24th ACM/IEEE Design Automation Conference*, pages 341–347, June 1987.
- [KR89] Keutzer and Richards. Computational complexity of logic synthesis and optimization. In *International Workshop on Logic Synthesis*, 1989.
- [Moo65] Gordon Earle Moore. Cramming more components onto integrated circuits. *Electronics Magazine*, 38, April 1965.
- [MRV11] Dirk Müller, Klaus Radke, and Jens Vygen. Faster min–max resource sharing in theory and practice. *Mathematical Programming Computation*, 3(1):1–35, Mar 2011.
- [Pos41] Emil Leon Post. The two-valued iterative systems of mathematical logic. *Annals of Mathematics*, 5, 1941.
- [Tra15] Khai Van Tran. Algorithmen für das Technology-Mapping, 2015.