Sharing and Binding for General Circuits

Benedikt Lipinski
Interaktionstechnik und Design
Hochschule Hamm-Lippstadt
Lippstadt, Germany
benedikt.lipinski@stud.hshl.de

Abstract

Benötigte Hardware wird aus Platz- und Kostengründen mittels bestimmter Algorithmen verringert. Aufgezeigt wird, welche Vorteile das Verteilen von Rechenoperationen auf eine verringerte Hardware hat, außerdem eine grobe Funktion bestimmter Algorithmen und die dabei auftretenden Probleme bezüglich der Latenz.

I. Introduction

Durch das menschliche Verlangen, Problemstellungen zugunsten schwindender Komplexität und erhöhten Komforts zu automatisieren und zu brechen, wurden genau diese Anforderungen auf diejenigen elektrischer Schaltungen umgelegt.

A. Problemstellung

Nicht erst durch moderne Entwicklungen wie digitale Vernetzung mit dem Internet der Dinge oder das autonome Fahren wird die Betrachtung zeitlicher und finanzieller Aspekte für die Entwicklung multidimensionaler Systeme notwendig. Allerdings lassen genau diese Entwicklungen die Komplexität sprunghaft ansteigen.

B. Logische Bausteine

Realisierbar werden elektrische Verschaltungen in der Digitaltechnik erst durch den Einsatz sogenannter Logikbausteine, die miteinander kombiniert ein vorher genau bestimmtes Verhalten der geplanten Schaltung verursachen. Hierbei übernehmen verschiedene Bausteine verschiedene Aufgaben, die mal weniger und mal mehr komplizierte Operationen beinhalten.

1) AND-Gatter: Eine der Grundoperationen, die in fast jeder Schaltung zu finden ist, ist die AND-Operation- zu deutsch die UND-Operation. Hierbei wird der Eingang erst auf logisch 1 geschaltet, wenn alle Eingänge dies ebenfalls sind.

TABLE I
Wahrheitstabelle AND-Gatter

x	y	Out
0	0	0
0	1	0
1	0	0
1	1	1

2) OR-Gatter: Das Ergebnis des OR-Gatters ist in dem Fall wahr, wenn einer seiner Eingänge wahr ist.

Simple Schaltungen lassen sich durch das gezielte Aneinanderreihen von AND- und OR-Gattern gut realisieren, sie stellen die einfachste Form von Operanden dar, aus ihnen können fast alle komplexeren Schaltungen und Operanden realisiert werden.

TABLE II
Wahrheitstabelle Or-Gatter

x	y	Out
0	0	0
0	1	1
1	0	1
1	1	1

3) Multiplexer: Multiplexer werden genutzt, wenn eine parallele Verarbeitung aus unterschiedlichsten Gründen nicht möglich ist. Das kann zum einen eine nicht ausreichende Verfügbarkeit an Kanälen sein, aber auch die bewusste Entscheidung, weniger Kanäle aus Kostengründen zu benutzen. Dieser Schritt kann gewählt werden, wenn Verläufe zwar parallel laufen, aber nur exklusiv ausgeführt werden. Das bedeutet, dass nach einer elementaren Entscheidung nur einer der beiden Stränge ausgeführt werden kann. Beeindruckend allerdings ist, dass Multiplexer trotz ihrer fortgeschrittenen Eigenschaften gegenüber der Grundbausteine doch recht simpel aus genau diesem erzeugt werden können. So ergibt sich für einen einfachen Multiplexer das Schaltbild aus zwei AND-Gattern, einem NOT-Gatter und einem OR-Gatter.

4) FPGA: Der Einsatz dieser Grundelemente ist für das Erreichen des Ziels völlig hinreichend, doch benötigen sie eine sehr hohe Fläche, auf der sie verbaut werden müssen. Ein zu erreichendes Ziel ist die Fläche zu verringern, auf der die Schaltung realisiert werden muss. Dies ist durch die Verwendung von Halbleitern und die Entwicklung von integrierten Schaltkreisen auch erfolgreich gelungen. Allerdings besitzen diese sogenannten ICs den Nachteil, dass ihre Schaltung- wie die einer Realisierung mit Bausteinen- auch fest gebunden ist. Ein Nachteil, den FPGA- (Field Programmable Gate ARRAY) Bauteile nicht besitzen, ist die Zusammensetzung aus zusammengeschalteten Baugruppen aus Logikbausteinen, FlipFlops und deren Verbindung mit Multiplexern. In FPGAs kann das Verhalten der Eingänge zu den Ausgängen programmiert werden.

C. Strategien zur Architektur-Optimierung

Mit steigender Komplexität der Verschaltungen steigt zwingend auch die Anzahl der zu nutzenden Bauteile. Zudem steigt auch die Verwendung performaterer Bausteine stetig, um die Abarbeitung bezüglich Leistung und aller zeitlichen Deadlines einhalten zu können. Nachteilig ist vor dem Hintergrund ebenfalls, dass Bausteine mit einer höheren Leistung oftmals auch einen erhöhten Platzbedarf einfordern [2, S.327].
Ziel ist es, Strategien umzusetzen, die die Chipfläche verringern, ohne dabei die Leistungsfähigkeit der Schaltung zu beeinflussen. Leider ist dieses Problem nicht mit der Einhaltung beider Anforderungen zu vereinbaren, wodurch zum Beispiel die Verminderung der Kosten durch Senkung der Chip-Anzahl und Fläche immer auch zum Nachteil der Performance agiert. Der Zusammenhang zwischen Fläche und der erzeugten Leistungsverminderung ergibt sich aus der Fläche a in der Einheit Gatteräquivalent $(GateEquivalentGE)$, das die Größe des simpelsten Gatters widerspiegelt [2, S.326] und der Performance, die den Kehrwert der Ausführungszeit T_{ges} widerspiegelt $\frac{1}{T_{ges}}$, welche sich wiederum aus der Taktzeit T_c multipliziert mit der Latenz λ zusammensetzt [2, S.325-326]. Eine mögliche Realisierung der Schaltung mit einer bestimmten Performance bei einer bestimmten Fläche lässt sich als Punkt in einem Diagramm darstellen, dessen Achsen sowohl die Chipfläche, die Latenz und den Takt abbilden [2, S.326]. Dieses Vorgehen [**?**] ist in diesem Fall notwendig,

da mehr als eine Variable optimiert werden muss, und es dementsprechend mehr als eine gültige Lösung dieses Problems gibt. Als mögliches Hilfsmittel kann hier die Pareto-Optimierung verwendet werden. Die Pareto-Optimierung erzeugt eine Kompromisslösung durch die Gewichtung jeder Einzelfunktion. Die Idee dahinter ist, dass die Gewichtung die Wichtigkeit der Einzelfunktion repräsentiert und ihr bei steigendem Gewicht mehr Aufmerksamkeit verleiht [4, S.45]. Als Ergebnis erhalten wir sogenannte Pareto-Punkte. Diese Punkte in unserem Entwurfsraum stellen verschiedene Verhältnisse unserer Optimierungsvariablen dar. Eine Pareto-optimale Lösung ist diejenige, die gültig gegenüber allen gestellten Anforderungen ist und die das bestmögliche Ergebnis aller Variablen liefert, ohne dass sich eine verschlechtert, nur um eine andere weiter zu optimieren.

D. Allocation

Nicht nur, aber vor allem aus Kosten- und Platzgründen, stehen der Entwicklung selten unbegrenzte Ressourcen zur Verfügung. Aus diesem Grund kann die Betrachtung der zur Verfügung stehenden Ressourcen zwingend notwendig werden. Somit ist das Ergebnis der Allokierung nach Teich [1] , eine Funktion $\alpha(r_k)$ die angibt, wie viele verfügbare Bauteile es zu einem jeweiligen Ressourcentyp gibt.
Damit würde, wenn vom Ressourcentyp "Multiplizierer" r_1 beispielsweise eine Einheit zur Verfügung steht, die Allokierungsfunkion $\alpha(r_1) = 2$ lauten. Die Funktion der Allokation zeigt deshalb ziemlich genau, welche Problemebeziehungsweise zeitliche Veränderungen durch die Grenzen entstehen, die durch die begrenzte Verfügbarkeit an Ressourcen gegeben werden. Dies kann ein nützlicher Zwischenschritt sein, um den Überblick zu behalten, den Teich mittels eines Ressourcengraphen anschaulich darstellt. In vielen Literaturen allerdings wird dieser Schritt auch in die Bindung integriert und nicht extra behandelt.

E. Binding

Im Unterschied zur Allokierung wird in der Bindung schließlich festgelegt, welche Ressource welche Operation zugeteilt bekommt, also die Zuweisung von Operationen zu bestimmten Hardwarekomponenten [3]. In einer utopischen Vorstellung, in der Kosten und Fläche keine Rolle spielen würden, könnte für jeden Rechenschritt, also jede Operation (also beispielsweise für jede Multiplikation oder jeden Vergleich) eine eigene Komponente benutzt werden. Leider ist diese Vorstellung in der Praxis nicht durchführbar, da jedes gesparte Bauteil auf eine große Stückzahl gerechnet den Kostendruck lindert. Das ist zudem auch nicht zwingend nötig, da beispielsweise ein Vergleich und eine Addition, welche beide durch eine ALU realisiert werden können, nicht unbedingt durch zwei verschiedene Hardwareeinheiten umgesetzt werden müssen, da der Vergleich erst stattfinden kann, wenn das Ergebnis der Addition vorliegt. In diesem Fall wird das "Optimal Ressource Sharing" genannt, oder zu deutsch: ein optimales Ausnutzen einer Ressource [2].

Die Bindung bietet ein Tool, das veranschaulicht, wie optimal die vorhandene Schaltung auf und mit den gegebenen Bauteilen realisiert wird. Da das endgültige Ziel eine optimale Ausnutzung der vorhandenen Ressourcen ist, müssen nun schrittweise die benötigten Bauteile minimiert werden. Dies wird erreicht, indem anfangs jede Operation einem Bauteil zugeordnet wird, es wird eine sogenannte dedizierte Ressource erzeugt, ein Spezialfall [3, S. 150]

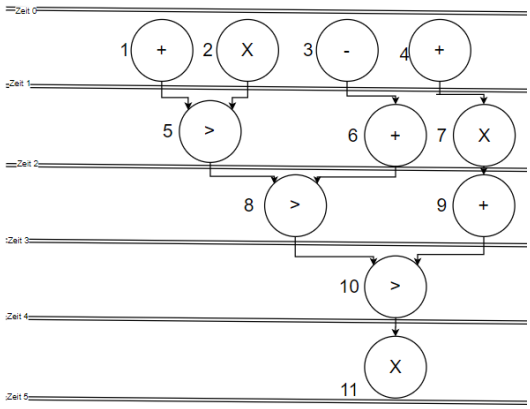

Fig. 1. Beispiel zur Bindung

Dem Bild ist zu entnehmen, dass in diesem Beispiel für die Zuordnung jeder Aufgabe zu einer Ressource elf Bauteile benötigt werden, die sich aus drei Multiplikationen = r_1 und aus acht Operationen für eine arithmetisch-logische Einheit r_2 ergibt.
Da es sich um ein recht einfach zu überblickendes Beispiel handelt, lässt sich durch einen geübten Blick bereits erkennen, wie eine Optimierung aussehen könnte. Im ersten Zeitslot ist abzulesen, dass drei arithmetische Operationen und eine Multiplikation durchgeführt werden, somit werden 3 ALU-Einheiten und ein Multiplizierer benötigt. Im weiteren Verlauf lässt sich unter anderem erkennen, dass in keinem weiteren Zeitslot mehr als 3 ALU-Einheiten gebraucht werden, deswegen ist dies auch unsere minimal benötigte Anzahl an Ressourcen dieses Typs. Ganz deutlich wird dies durch die Veranschaulichung in Tabelle IV.
Übertragen auf das Ergebnis aus Tabelle III kann nun abgele-

TABLE III
RESSOURCENTABELLE DES ABLAUFPLANS VOR DER OPTIMIERUNG

v_i	$\beta(v_i)$	$\gamma(v_i)$
v_1	r_2	1
v_2	r_1	1
v_3	r_2	2
v_4	r_2	3
v_5	r_2	4
v_6	r_2	5
v_7	r_1	2
v_8	r_2	6
v_9	r_2	7
v_10	r_2	8
v_11	r_1	3

TABLE IV
BENÖTIGTE EINHEITE PRO ZEITSLOT

Zeitslot	ALU	Multiplizierer
1	3	1
2	2	1
3	2	0
4	1	0
5	0	0

sen werden, dass Ressourcen des Typs "ALU" benötigt werden und eine des Typs "Mulitplizierer".
Übertragen auf die obere Tabelle V entsteht nun folgendes Binding:
Somit wurden in diesem Beispiel fünf Alu-Einheiten, die

TABLE V
RESSOURCENTABELLE DES ABLAUFPLANS NACH DER OPTIMIERUNG

v_i	$\beta(v_i)$	$\gamma(v_i)$
v_1	r_2	1
v_2	r_1	1
v_3	r_2	2
v_4	r_2	3
v_5	r_2	1
v_6	r_2	2
v_7	r_1	1
v_8	r_2	1
v_9	r_2	2
v_10	r_2	1
v_11	r_1	1

einen Anteil von $62,5\%$ ausmachen, eingespart. Bei den Multiplizierern wurden die benötigten Ressourcen auf $\frac{1}{3}$ reduziert.

F. Sharing

Das Sharing beschreibt zusammen mit dem Binding die effektive Mehrfachnutzung der Ressourcen, welches der Minimierung der verwendeten Hardware dient und somit zur Reduzierung von Flächen- und Kostenproblemen beiträgt, wobei das Ziel des Sharing die Frage nach dem Algorithmus zur Optimierung ist und anschließend mit einem Binding endet. In dem Kapitel Binding wurde ein einfach im Kopf zu lösendes Beispiel gezeigt. Leider bleibt es in der Realität nicht immer bei so einfach zu lösenden Systemen, sodass schnell der Überblick verloren gehen kann und zudem ein System unter Umständen nicht zu lösen wäre. Aus diesem Grund befasst sich dass Sharing mit der Frage, wie ein vorhandenes System effektiv optimiert werden kann. Zu den verwendeten Algorithmen zählen unter anderem die aus dem Bereich der Ressource Dominated Circuits, des Register Sharing, Multiport Memory Binding und des Bus Sharing and Binding und für den Bereich der General Circuits sind es Unconstrained Minimum - Area Binding, Performance Constrained Binding und zuletzt Performance Directed Binding.

II. GRUNDLEGENDES

A. Kompatibilitäts- und Konfliktgraphen

Als Voraussetzung, um überhaupt eine sinnvolle Bindung beziehungsweise Allokierung erzeugen zu können, muss sich im ersten Schritt einmal mit dem eigentlichen Problem befasst werden. Um das Problem darstellen zu können wird ein Graph aufgestellt, der die Eigenschaften der Elemente beschreibt. Dieser Schritt ermöglicht die Betrachtung der einzelnen Operationen miteinander und stellt auf, welche miteinander Kompatibel sind und welche ein kritisches Verhalten zueinander aufweisen.
Zwei oder mehr Operationen sind miteinander kompatibel und können an die gleiche Ressource gebunden werden, wenn sie nicht miteinander konkurrieren und wenn sie dem gleichen Ressourcen-Typen angehören. [3, S.231]
Das bedeutet konkret, dass Aufgaben miteinander kompatibel sind, wenn sie nicht im gleichen Zeitslot nebeneinander anliegen und wenn sie des gleichen Typs sind (zum Beispiel Addierer). Die Entwicklung der des Graphensystems kann grundsätzlich über zwei Ansätze erfolgen. Zum einen kann ein Kompatibilitätsgraph oder Verträglichkeitsgraph entwickelt werden. Dieser Graph gibt an, welche Operationen verträglich zueinander sind und somit keine Probleme darstellen. Komplementär dazu lässt sich durch den Konfliktgraphen angeben, welche Operationen im Konflikt zueinander stehen und einer genaueren Betrachtung bedürfen.
Ein Ressourcen-Verträglichkeitsgraph ist definiert durch $G_+(V, E)$, in denen die Knoten durch $V = \{v_i; i = 1, 2, ..., n\}$ definiert werden und die Kanten durch $E = \{\{v_i, v_j\}i, j = 1, 2, ..., n\}$. Die Knoten stellen die Operationen an sich dar, wohingegen die Kanten die kompatiblen Operationspaare angeben [3, S.231].
Zusammengefasst nach Teich [1, S. 181-182] lassen sich folgende Verträglichkeiten für den Kompatibilitätsgraphen aufführen:

Demnach ist eine **schwache Verträglichkeit** gegeben, wenn zwei Knoten eines Graphen miteinander verbunden sind und dem gleichen Typ angehören- also zum Beispiel eine Addition und ein Vergleich, da beide durch eine arithmetisch-logische Einheit verarbeitet werden können [1, S. 181].

Ablaufplanverträglich ist ein Graph hingegen, wenn seine Knoten als erste Bedingung vom gleichen Typ sind, also wie bei der schwachen Verträglichkeit zum Beispiel aus zwei Operationen vom Typ Multiplizierer. Zum zweiten muss die Bedingung erfüllt sein, dass die Startzeit des ersten Knoten vor der des zweiten Knotens liegen muss. Das bedeutet, dass Knoten die ablaufverträglich sind, nicht in dem selben Zeitslot ausgeführt werden dürfen, andernfalls wären sie nicht ablaufplanverträglich [1, S. 182].

Als letztes definiert sich die **starke Verträglichkeit** durch zwei Knoten, die wieder vom gleichen Typ sind und zudem voneinander abhängig sind. Das bedeutet, die beiden Knoten sind in einem logischen Strang durch einen Pfad im Ablaufplan miteinander verbunden. [1, S. 182]

Die graphische Umsetzung des Verträglichkeitsgraphen mit dem Beispiel aus den vorherigen Kapiteln sieht dann wie folgt aus:

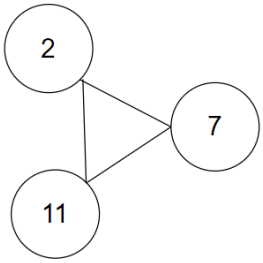

Fig. 2. Verträglichkeitsgraph Mulitplizierer

Aus den vorherigen Kapiteln, wissen wir bereits, dass für die Optimierung der Verschaltung nur ein Multiplizierer benötigt wird und genau das können wir in diesem Beispiel nun auch grafisch betrachten. Zu sehen ist, dass jeder Knoten mit einer Kante verbunden ist und somit sind alle Operationen miteinander ablaufplanverträglich, weswegen der Einsatz von nur einem Multiplizierer nun auch optisch besser nachzuvollziehen ist.

Als nächsten Schritt werden nun sogenannte Cliquen eingeze-

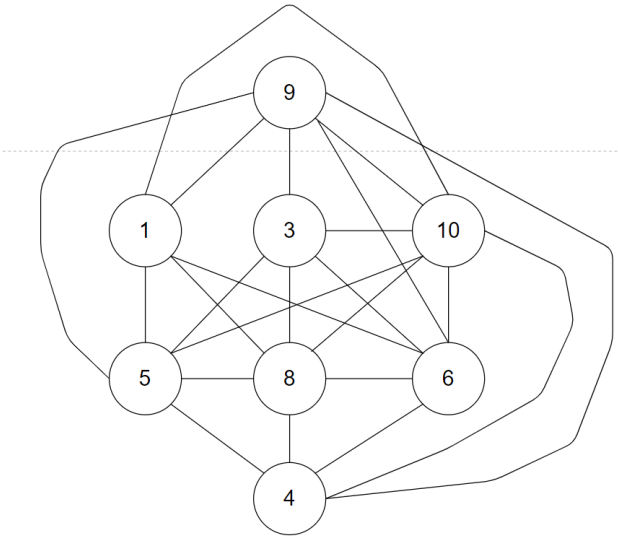

Fig. 3. Verträglichkeitsgraph arithmetisch-logische Einheit mit Cliquen

ichnet. Sie repräsentieren die Zusammenfassung mehrerer Operationen und zeigen ihre Zusammenführung auf eine Komponente, sodass für jede Clique genau eine Komponente eingesetzt wird. Zu sehen ist, dass bei dem Verträglichkeitsgraphen in diesem Beispiel für die Mulitplizierer genau eine Clique eingezeichnet werden muss. Da mit steigender Komplexität der Aufgabe auch die Komplexität für das Einzeichen der Cliquen steigt, bietet sich dieses Verfahren leider nur für die Anwendung auf simple Verschaltungen an. Der Grund hierfür ist die Klassifizierung dieses Problems als NP-hartes Problem. [2, S.361]

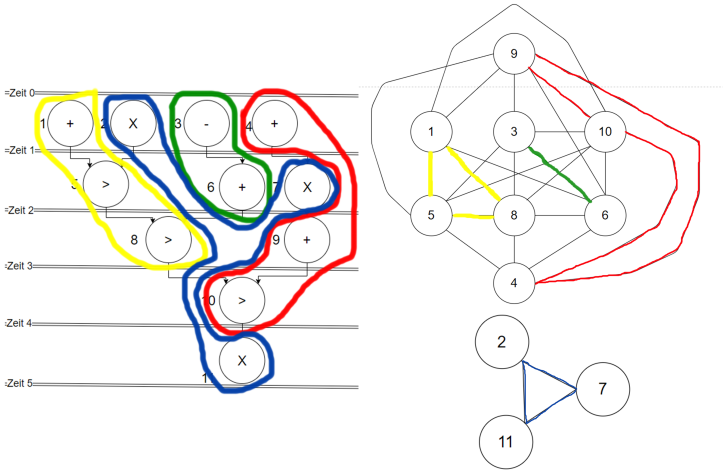

Fig. 4. Ablaufplan und Verträglichkeitsgraphen mit Cliquenbildung

B. *Ressource Dominated Circuits*

Das "Sharing and Binding for Resource Dominated Circuits" (Verteilung und Bindung für Ressourcen-dominierte Schaltkreise) bedeutet im Folgenden die Umsetzung und Kombination des Wissens und der Grundlagen aus den vorangegangen Abschnitten. Zur Erinnerung: das gesuchte Ergebnis ist eine Bindung. Das bedeutet eine Zuordnung von tatsächlicher Hardware und Operationen, in denen eine maximale Menge an Operationen einer minimalen Menge an Hardware zugeordnet wird. Hierfür ist im Vorfeld zwingend festzustellen, ob nicht durch andere dynamische Umwelteinflüsse und Anforderungen eine grundsätzliche Begrenzung der zur Verfügung stehenden Hardware und Fläche besteht (Allokierung). Des weiteren, um überhaupt feststellen zu können, welche Operationen sich auf der selben Hardware kombinieren lassen, muss ein Kompatibilitäts-/Konfliktgraph aufgestellt werden, mit dessen Ergebnis sich Operationen zu Cliquen bündeln lassen. Leider ist die Lösung des vorliegenden Problems nicht derart einfach, als einziges Ziel lediglich die Fläche und somit den Preis der Verschaltung zu optimieren. Gerade in Echtzeitsystemen muss auch immer die Aufmerksamkeit zu gleichen Teilen auf der Einhaltung der zeitlichen Vorgaben bleiben. Deswegen betrachtet das Sharing and Binding für Ressource Dominated Circuits die Optimierung der Schaltung bezüglich Fläche (Kosten) und Latenz (Zeitschranken) mittels der Optimierung der wirkenden Schaltung und mittels Verminderung der eingesetzten Hardware und Doppelnutzung dieser. [3, S. 156]

III. General Circuits

Die Betrachtung einer Schaltung unter dem Aspekt der Ressource Dominated Circuits ähnelt allerdings eher der Betrachtung einer idealen Komponente als einer, in denen auch Störgrößen und andere Umwelteinflüsse eine gewichteten Zuwendun erfahren.

A. Allgm.

Aus diesem Grund wird für das Sharing and Binding for General Circuits (Verteilung und Bindung für generelle Schaltkreise) nicht nur Latenz und Fläche der operierenden Elektronik betrachtet, sondern auch derer, die steuernde und speichernde Eigenschaften besitzen. Zudem wird auch die Verkabelung dieser nicht außen vorgelassen. [3, S. 156]

B. Baugruppen

1) Register: Da in Schaltungen wie sie in allen modernen Systemen vorhanden sind, die über eine einfache input/output-Addition hinaus gehen, Daten und Werte über einen einzelnen Zyklus hinaus gespeichert werden müssen, kommen in vielen Schaltungen Speicherelemente zum Einsatz. Beispielsweise müssen in Regelungsschaltungen Werte gespeichert werden, die über mehrere Zyklen zur Verfügung stehen.
Grundlegend verhält sich die Bindung und das Teilen von Registern ganz ähnlich zu den Vorgängen bei normalen Operatoren. Auch sind die Vorgaben, bei denen Register geteilt werden können, gleich zu denen der Operatoren. Das bedeutet, Register können geteilt werden, wenn sie nicht in Konkurrenz zueinander stehen, also, wenn sie exklusiv zueinander stehen oder erst nacheinander auftreten. Nicht analog allerdings ist hier die Definition der Lebenszeit einer Variable und dementsprechend auch der Registernutzung. Die Lebenszeit ergibt sich nach De Micheli aus dem Geburtszeitpunkt (*former*) und dem Todeszeitpunkt (*latter*) einer Variable. Der Geburtszeitpunkt ist der Zeitpunkt der Erstellung durch das Ergebnis einer Operation und der Todeszeitpunkt ist als der späteste Zeitpunkt anzunehmen, in dem die Variable als Input referenziert wird [3, S. 240] Zur Lösung der Registerbindung kann der Left-Edge-Algorithmus verwendet werden [2, S. 359].

2) Steuerlogik: Unter der Berücksichtigung der Steuerlogik kann sowohl für Multiplexer, die ganz ähnlich zu den aufgezeigten Vorgängen betrachtet werden können, als auch für Bussysteme bewertet werden. Hierzu wird das Augenmerk auf zwei zu berücksichtigende Komponenten gelegt. Als erstes wird der Bus an sich betrachtet, der einen Fixwert zugeordnet bekommt- einen sogenannten Overhead. Als zweite Komponente werden die Bustreiber aufgeführt, also die Funktionseinheiten, die zur Steuerung des Busses fungieren. Diese können als Verteilter-Multiplexer betrachtet werden, was eine Berechnung von Anzahl und Verbrauch möglich macht. [3, S. 157]

3) Verkabelung: Als ein weiterer, nicht zu vernachlässigender Bereich, gilt vor allem die für das Platzmanagement benötigte Verdrahtung zur Verbindung der platzierten Bauteile. Hierbei ist zu beachten, dass nun auch starke physikalische Beeinflussung in die Platzierung der Komponenten miteinbezogen werden muss, was sich somit wiederum auch auf die benötigte Fläche ausbreitet. So müssen beispielsweise Leitungen, die Betriebsspannung führen, möglichst von empfindlichen Busleitungen räumlich ferngehalten werden. Somit ist eine unumgängliche Voraussetzung für die Ermittlung des Platzbedarfs Kenntnis über die Struktur der Elemente. Dazu zählen ihre Größe, ihre Anzahl und ihre Platzierung. Zudem ist auch die Bindung der Elemente von Bedeutung, da diese einen direkten Einfluss auf ihre Verkabelung hat. [3, S.157] Zur Berechnung ergibt sich, dass die Länge der Verdrahtung proportional zur Anzahl der eingesetzten Funktionsblöcke hoch α ist, wobei $0 \leq \alpha \geq 1$ ist. [3, S.157]

4) Kontrolleinheiten: Da Signale zur Steuerung von Bausteinen unerlässlich sind und diese letztendlich den gleichen Beschränkungen und Einflüssen unterliegen wie Datensignale, kann es passieren, dass diese Steuersignale auf einem kritischen Pfad liegen und somit zu einem Problem für den Programmablauf werden.
Eine Möglichkeit der Optimierung ist die Reduzierung des ROM (**R**ead **O**nly **M**emory). Dies ist möglich, wenn es sich um eine Implementierung einer Microcode-basierten Kontrolleinheit handelt. Der Vorgang sieht in diesem Fall nicht eine sogenannte horizontale Umsetzung eines des Microcodes vor, der eine eins-zu-eins-Verbindung von Bits zu Steuersignalen vorsieht, sondern die Umsetzung in einer vertikalen Variante des Microcodes, der Bits in Gruppen zusammenfasst und demnach die Menge der zu speichernden Daten reduzieren kann. [2, S.371-372] Hart verdrahtete Codierungen hingegen lassen sich nur durch ihre unterschiedliche Bauweise optimieren und anpassen, wohingegen der Gedanke der Anpassung bedeutet, eine gewisse Störsicherheit einzubinden. So werden in kritischen Umgebungen, in denen es nicht zu Störsignalen kommen darf, da sie einen Bit kippen lassen würden, gray-code Impulse verwendet, um in den nächsten Zeitschritt zu springen. [2, S.373]

IV. Sharing and Binding for General Circuits

A. Unconstrained minimum - Area Binding

Zur Berechnung und Einschätzung der Kosten, und somit der Möglichkeit diese zu minimieren, muss wie bereits in den vorherigen Kapiteln festgestellt eine Möglichkeit geschaffen werden, mehr als eine Operation einer Hardwarekomponente zuordnen zu können. Hinzugekommen ist nun auch die Betrachtung der Steuerlogik, der Datenübertragung und der Verkabelung, die ebenfalls einen nicht unerheblichen Einfluss auf die Effizienz einer Schaltung nehmen. Ganz äquivalent wird nun auch unter Berücksichtigung dieser Punkte versucht herauszufinden, ob und in welchem Rahmen diese verteilt werden können. Durch den Umstand, dass die Vorgehensweise ähnlich zu der ist, die von der einfachen Betrachtung ohne die Steuerlogik bekannt ist, ergeben sich auch ganz ähnliche Probleme bei der Arbeit mit dieser.
Problematisch ist nun auch, dass auch in diesem Fall mit steigender Komplexität der Verschaltung das Bestimmen eines Optimums zunehmend schwieriger wird. Deswegen wird mittels bestimmter Algorithmen versucht, den perfekten Graphen zu finden. Da ein perfekter Graph die Annahme zulässt, dass er in Polynomialzeit berechenbar ist, kann

angenommen werden, dass ein Ergebnis in einer akzeptablen Zeit zu erwarten ist. Für den Fall, dass sich der perfekte Graph in Polynomialzeit berechnen lässt, gilt auch, dass sich all seine Parameter ebenfalls in akzeptabler Zeit berechnen lassen [6, S. 60].
Eine mögliche Vorgehensweise ist, die Kosten für zum Beispiel Multiplexer gleichmäßig über die Operatoren zu verteilen. [3, S.247] Ermöglicht wird dies über eine gänzliche Betrachtung der Cliquen, denen ein Set an Kosten zugeordnet wird, das sich aus den Kostenmengen ihrer Knoten ergibt. Ihre Gewichtung erhält die Clique durch die Anzahl ihrer hinzugefügten Knoten. Die Gesamtkosten wiederum ergeben sich aus der Summe der Cliquen, die zu einer Partition zusammengefügt werden. [3, S. 247]
So kann eine Formel für Multiplexer wie folgt aussehen:

$$area_{mux} = area^{\Delta}_{mux}(a-1) \tag{1}$$

und wird als:

$$area^{0}_{M} + \sum_{i=1}^{a} area^{i}_{M} \tag{2}$$

geschrieben
[3, S. 247]

Des weiteren kann ein Algorithmus nach Tseng und Siewiorek genutzt werden, indem Cliquen partioniert, also zusammengefasst werden. Das Ziel des Algorithmus ist es, ungewichtete Untergraphen zu erstellen, um im Anschluss eine Optimierung erhalten zu können [3, S. 247]. Erreicht wird dies durch das Erkennen und Auswählen des Knotenpaares mit den meisten gemeinsamen Knoten, die mittels einer Kante verbunden sind. Gesucht wird also das Paar mit den meisten gemeinsamen Nachbarn. Sobald das Paar mit den meisten Nachbarn gefunden wurde, werden diese zusammengeführt. Nach der Zusammenführung werden nun alle Kanten gelöscht, die nicht in Verbindung mit beiden zusammengeführten Knoten stehen. Übrig bleiben demnach nur noch die verbundenen Knoten und alle Kanten, die mit beiden Knoten in Verbindung standen. Dies kann solange durchgeführt werden, wie sich Partner für eine Zusammenführung finden lassen, andernfalls ist die erstellte Clique maximal und kann abgespeichert, gegebenenfalls gefärbt und anschließend geupdatet werden. Eine Aktualisierung des Graphen lässt sich mittels des Updating-Algorithmus nach Tseng und Siewiorek [5, S.393] durchführen, indem die Kanten, die zusammengeführt wurden, aus der Liste der Kanten entfernt werden und in eine Liste der gelöschten Kanten eingetragen werden. Anschließend werden in einem zweiten Schritt die Anzahl der nächsten Nachbarn und die Anzahl der Kanten, die gelöscht werden müssen, neu berechnet. Abgeschlossen ist die Cliquenpartionierung in dem Moment, in dem sich keine Kanten mehr in der Kantenliste befinden [3, S.248] [5, S.392-393].
Ein Graph ist in dem Moment perfekt, wenn auch alle seine Untergraphen $H(G)$ folgende Bedingungen erfüllen. Ein Untergraph ist perfekt, wenn die Mächtigkeit seiner größten Clique gleich ist mit der chromatischen Zahl. $\omega(H) = \chi(H)$. Des weiteren muss die Unabhängigkeitszahl $\alpha(H)$, die angibt, welche Knoten nicht miteinander durch eine Kante verbunden sind, also in einem nicht adjazenten Verhältnis stehen, ebenfalls gleich mit der Anzahl an unabhängigen Mengen, also der Zusammenhangszahl ($\kappa(H)$ gleich sein $\alpha(H) = \kappa(H)$ [1, S.507] [6, S.59 - 60] Das bedeutet konkret am Beispiel der vorangegangenen Algorithmen, dass versucht wird, einen Graphen zu optimieren, in dem der Graph in Untergraphen zerlegt wird und für diese geprüft wird, ob ein Optimum vorliegt.

B. Performance Constrained Binding und Performance Directed Binding

Neben der Optimierung der Kosten durch Reduktion der eingesetzten Hardware war es zwingend notwendig, einen weiteren Aspekt zu beobachten, da dieser direkt durch die Mehrfachnutzung der Hardwarekomponenten beeinflusst wird. Da die Reduktion der benutzten Hardware auch immer einen direkten Einfluss auf die Zeit hat, die zur Ausführung der Aufgaben benötigt wird, beeinflussen auch die verbaute Steuerelektronik, die Verkabelung und die Datenhaltung unmittelbar die Ausführungszeit. Gerade Multiplexer wirken sich negativ auf die benötigte Ausführungszeit aus, da eine direkte serielle eins-zu-eins Verkabelung immer schneller ist als eine gemulitplexte, in denen die Übertragung zugunsten von Chipfläche und weniger Verkabelung reduziert wird.
Des weiteren stellt sich somit die Frage, ob durch das Ersetzen von Direktverkabelungen durch Multiplexer, eine zeitliche Beeinträchtigung entsteht, die das Einhalten der Zykluszeit verletzt. Betrachtet wird ein Pfad, in einem Ablaufplan mit einer maximalen Menge von Knoten. In diesem können Operationen unter der Voraussetzung, dass sie die Zykluszeit Φ nicht verletzen, in den selben Taktzyklus gelegt werden. Hierbei spricht man von der Verkettung ((chaining)) von Operationen [3, S.249]. Unter dieser Zuhilfenahme ist es möglich, auch Multiplexer und Verkabelungen zu verketten und diese in einen normalen Ablaufplan und Bindung B einzubauen. Eine Formel, die die Pfadverzögerung angibt ist:

$$\sum_{i \in path} \widehat{d_i} + multiplexer\_delay(B) + wireing\_delay(B) \tag{3}$$

wobei $\widehat{d_i}$ die Einzeldelays der Ressourcen darstellen und $B = Binding$ angibt.
Das Ressource-Sharing unter zeitlichen Aspekten wird mit der Formel

$$\sum_{i \in path} d_i + multiplexer\_delay(B) + wireing\_delay(B) < Phi \forall path \tag{4}$$

beschrieben [3, S.249] Ein weiteres Problem bei der Entscheidung der verwendeten Übertragungsart ist nach G. De Micheli [3, S.250], dass eine ausschließliche Umsetzung mittels Multiplexer mit der einer reinen Umsetzung per Verkabelung im Konflikt steht. Das bedeutet für den Fall, dass eine der beiden genannten Verzögerungen vernachlässigbar wäre, die Wahl immer auf diese fällt. Daraus ergibt sich konkret, wenn die Verzögerung der Verkabelung $wirering_delay(B)$ vernachlässigbar wäre, dass es sich nicht lohnt Signale zu multitplexen, da die Verzögerung der Steuerlogik von Multitplexern eine Verschlechterung der Ausführungszeit zu verantworten hätte. Demnach ist auch in diesem Fall ein gesundes Mittelmaß und ein gezielter Einsatz beider Implementierungsarten das richtige Mittel, um eine möglichst kurze Gesamtverzögerung umzusetzen.

V. Register, Multi-Port Memory, Bussysteme

Ein weiteres Problem, das mit der realen Betrachtung und somit der Beeinflussung durch Verzögerungen, der Verkabelung und der Steuerlogik einhergehen, ist dass auch bei der Datenhaltung selbst Verzögerungen auftreten, die es so minimal zu halten gilt wie möglich, unter weiterer Berücksichtigung von Fläche und somit Kosten.
Somit gilt es auch bei Registern ,Multi-Port Memory, und Bussystemen darauf zu achten, dass Kabelwege so kurz wie möglich gehalten werden. Außerdem soll bei Multi-Port Memory und Bussystemen darauf geachtet werden Variablen, die das gleiche Ziel oder Quelle haben, zu bündeln [3, S.250]. Des weiteren darf bei Bussystemen nicht die Verzögerung der Steuerungslogik vergessen werden, die aufgrund von Bus-Treibern ebenfalls anfällt [3, S.250].

VI. Fazit

Durch das Reduzieren von Hardware mittels Teilen von Hardware für mehr als eine Rechenoperation konnte beobachtet werden, dass sich die Anzahl der eingesetzten Hardwareteile deutlich reduzieren lässt. Dies bietet somit einen erheblichen vorteilhaften Effekt auf eine umgesetzte Verschaltung bezüglich Flächen und Kosten einer Schaltung. In einem ersten Moment sind die Vorteile einer verkleinerten Schaltung sicherlich, dass eine Umsetzung auch in Systemen mit einem kritischen Platzangebot ermöglicht wird. Um ein Beispiel zu nennen, profitieren hiervon aktuell immer kleiner werdende Geräte für das Internet der Dinge. Ein weiterer Aspekt ist die Abhängigkeit von Kosten zur verwendeten Fläche einer Schaltung, demnach steigen mit wachsender Anzahl an verwendeter Hardware auch die Kosten für eine Schaltung. Folglich ist das primäre Ziel auch hier, die Schaltung so klein wie möglich zu halten. Was im ersten Moment für einen Unbeteiligten bei Bauteilkosten von wenigen Cent als marginal betrachtet werden kann, so ist es bei näherer Betrachtung doch sinnvoll, bei hohen Abnahmemengen durchaus auf jeden gesparten Cent zu achten. Vorteil ist neben geringerer Kosten für Unternehmen auch eine bessere Konkurrenzfähigkeit gegenüber anderen Herstellern. Zudem kann die Verringerung der Kosten ebenfalls an den Kunden weitergegeben werden, was den Vorteil hat, auch Menschen mit weniger finanziellen Mitteln einen Zugang zu Technik zu ermöglichen. Gezeigt wurde also, dass es sehr vielschichtige Gründe gibt, die Anzahl der verwendeten Hardwarekomponenten einer Schaltung zu reduzieren-jedoch ist leider auch die Reduzierung von Hardware nicht umsonst. Eine Verminderung von Hardwarekomponenten ist nur unter der Beeinflussung der Ausführungszeit möglich. Das bedeutet theoretisch wäre es durchaus möglich, eine komplette Schaltung mit nur einer Hardwarekomponente jeden Typs auszuführen, allerdings würde das in einem katastrophalen Anstieg der Ausführungszeit enden. Um dennoch ein Optimum zu finden, wird zum Beispiel das Pareto-Prinzip eingesetzt. Letztendlich ist es wichtig festzuhalten, dass es sich bei der Optimierung von Schaltungen keinesfalls um etwas Einfaches handelt. Die Optimierung mittels Teilen von Hardwarekomponenten fordert ein breit gefächertes Verständnis von Algorithmen und theoretischen Annahmen der Informatik- doch kann ein erfolgreiches Ergebnis einer Optimierung Großes bewirken.

References

[1] Jürgen Teich, Christian Haubelt, Digitale Hardware/Software-Systeme, Synthese und Optimierung, 2. Auflage,Springer,2007
[2] Dr.Walter Lange, Prof.Dr.Martin Bogdan, Entwurf und Syntese von Eingebetteten Systhemen, Ein Lehrbuch,Oldenbourg Verlag München, 2013
[3] Giovanni De Micheli, Synthesis and Optimization of Digital Circuits,McGraw-Hill, Inc. New York etc,1994
[4] Martin Pieper, Mathematische Optimierung, Eine Einführung in die Kontinuierliche Optimierung mit Beispielen,Springer Spektrum,2017
[5] Chia-Jeng Tseng, Daniel P. Siewiorek, Automated Synthesis of Data Paths in Digital Systems,IEEE Xplore, 1986
[6] Graphentheoretische Konzepte und Algorithmen, Sven Oliver Krumke & Hartmut Noltemeier, 3. Auflage,Springer Vieweg, 2012

List of Figures

List of Tables