Seminarvortrag ALU-Konstruktion Logiksynthese, Steuerwerksentwurf

Lars Werner

4. November 1994

Seminar
der
Projektgruppe Systementwurf
WS94/95 und SS 95

Inhaltsverzeichnis

1 Einführung in die Logiksynthese

1.1 Was ist Logiksynthese und was sind ihre Aufgaben?

Die Logiksynthese beschäftigt sich damit Strukturbeschreibungen von Schaltungen auf Gatterebene in Verhaltensbeschreibungen (Polynome) umzusetzen und zu optimieren.
Optimiert werden solche Strukturbeschreibungen damit sie kostengünstig produziert werden können. Ein aber viel wichtigerer Grund ist die Verkürzung der Schaltzeit bei den optimierten Schaltungen und die Einsparung von Chipfläche.

Die Kosten für einen Booleschen Ausdruck kann man wie folgt definieren:

> Die Kosten eines Booleschen Ausdrucks berechnet sich aus der Addidion der Zahl der Disjunktionene und Konjunktionen des zugehörigen Schaltnetzes.

1.2 Überführung der Strukturbeschreibungen in Polynomform oder Tabellenform

Damit die Strukturbeschreibungen der zu optimierenden Schaltungen mit mathematischen Funktionen optimiert werden können werden die Gatterschaltungen in Polynome in disjunktiver Normalform (DNF) überführt, die die Schaltungen algebraisch beschreiben.

Beispiel: Halbaddierer

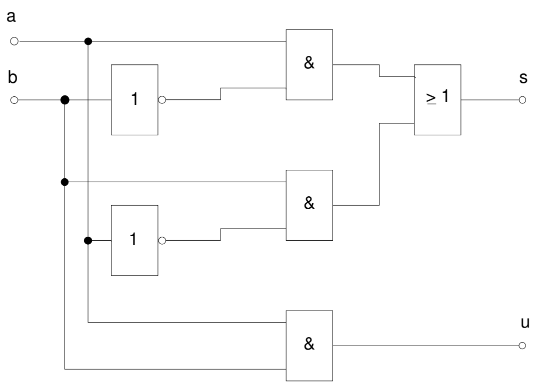

Abbildung 1: Halbaddierer

Überführt man den Halbaddierer aus Abbildung 1 in Polynome in disjunktiver Normalform, so erhält man die folgenden beiden Gleichungen:

- $s = a\bar{b} + \bar{a}b$
- $u = ab$

Diese beiden Gleichungen lassen sich in diesem Zustand mit den nachfolgend beschriebenen Verfahren optimieren.

Eine weitere Möglichkeit eine Gatterschaltung formal darzustellen ist die Darstellung in einer Funktionstabelle.

Beispiel: Halbaddierer aus Abbildung 1

a b	u s
0 0	0 0
0 1	0 1
1 0	0 1
1 1	1 0

Tabelle 1: Halbaddierer

1.3 Graphisches Optimierungsverfahren nach Karnaugh-Veitch

Ein sehr einfaches und anschauliches Verfahren ist das graphische Minimierungsverfahren von Karnaugh und Veitch. Leider eignet sich dieses Verfahren nur für maximal 4 Variablen, da es für mehr sehr kompliziert wird.
Bei dem Verfahren von Karnaugh wird ein Rechteck in 2^N Felder eingeteilt, die den Mintermen der Schaltfunktion entsprechen.

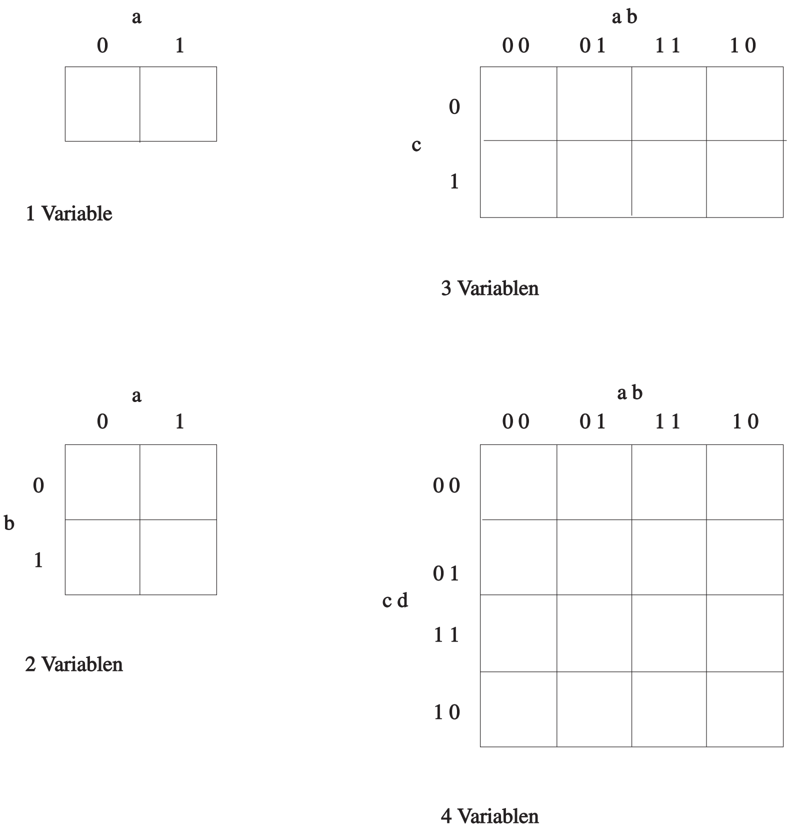

Abbildung 2: mögliche KV - Diagramme

Benachbarte Felder der KV - Diagramme unterscheiden sich jeweils nur in einer Variable. Man muß sich diese Diagramme torusförmig abgeschlossen vorstellen. Daß heißt die Nach-

barschaft erstreckt sich auch über die Ränder hinaus.
Eine Funktion wird mintermweise in das passende Diagramm eingetragen.

Beispiel: $f = \bar{b}\bar{c} + \bar{a}b\bar{d} + a\bar{b}c + abcd + ab\bar{c} + \bar{a}\bar{c}d$

cd \ ab	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11			1	1
10		1		1

Abbildung 3: Beispieldiagramm

Die Minimierung erfolgt durch die Verschmelzung benachbarter Minterme.
Es werden nach und nach die größte Bereiche gesucht, die nur Einsen ennthalten. Diese werden markiert.
Ein Bereich darf jedoch nur 1, 2 oder 4 Felder breit und 1, 2 oder 4 Felder hoch sein.
Hat man alle Einsen markiert, kann man die optimierte Funktion aufschreiben. Man braucht sich dazu nur die Variablen am Rand des Diagrammes anzusehen. Zu jeder Markierung gehört ein Minterm. Die Variablen des Mintermes werden am Rand des Diagrammes abgelesen. Es sind die, welche in Höhe der Markierung stehen und sich auf der gesamten Länge der Markierung nicht ändern.

Beispiel:

cd \ ab	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11			1	1
10		1		1

Abbildung 4: Beispieldiagramm mit markierten Mintermen

Nach der Optimierung ergibt sich folgende Funktion:

- $f = \bar{c} + ad + a\bar{b} + \bar{a}b\bar{d}$

Mit Hilfe von sogenannten Dont't Care Stellen ist eine weitere Optimierung möglich.
Bei diesen Stellen handelt es sich um Eingangsvariablen, die keinen Einfluß auf die Ausgangsvariablen haben. Solche Stellen werden in den KV-Diagrammen gesondert markiert und nur wenn nötig für eine überdeckung verwendet.

1.4 Das Optimierungsverfahren von Quine und McCluskey

KV - Diagramme sind prinzipiell auf Boolesche Funktionen mit höherer Stellenzahl erweiterbar, aber mit wachsender Anzahl der benötigten Dimensionen wird das Verfahren sehr kompliziert.
Das Verfahren von Quine McCluskey ermöglicht die Suche nach dem "billigsten" Schaltnetz dagegen bei beliebiger Stellenzahl nach einem recht einfachen Verfahren.

Auch bei diesem Verfahren benötigt man die Schaltfunktion in disjunktiver Normalform.
Bei diesem Verfahren notiert man alle Minterme mit einer gleichen Anzahl von Einsen in Blöcken untereinander. Anschließend werden zwischen den Blöcken die Minterme verglichen. An der Stelle an der sie sich unterscheiden wird eine Markierung gemacht und der neu entstandene Minterm hingeschrieben.
Auf diese Art versucht man möglichst viele Überdeckungen zu finden. Minterme für die man keine Überdeckung gefunden hat werden mit in die nächste Runde genommen, wo das Verfahren erneut angewendet wird.
Das Verfahren wird so lange fortgeführt, bis sich keine Überdeckungen mehr finden lassen. Auf diese Art lassen sich alle Primimplikanten finden. In vielen Fällen werden aber nicht alle Primimplikanten benötigt, so das man die "günstigsten" auswählen kann. Das sind in der Regel die, die die meisten Minterme der Ursprungsfunktion überdecken.

Beispiel:

$$f = \bar{x_1}\bar{x_2}\bar{x_3}\bar{x_4} + \bar{x_1}x_2\bar{x_3}\bar{x_4} + \bar{x_1}x_2x_3\bar{x_4} + x_1\bar{x_2}x_3x_4 + x_1x_2\bar{x_3}\bar{x_4} + x_1x_2\bar{x_3}x_4 + x_1x_2x_3\bar{x_4}$$

	Einschlägiger Index	Erste Iteration	Zweite Iteration
$\bar{x_1}\bar{x_2}\bar{x_3}\bar{x_4}$	0000	0*00	
$\bar{x_1}x_2\bar{x_3}\bar{x_4}$	0100	01*0 *100	*1*0 *1*0
$\bar{x_1}x_2x_3\bar{x_4}$ $x_1x_2\bar{x_3}\bar{x_4}$	0110 1100	*110 110* 11*0	
$x_1\bar{x_2}x_3x_4$ $x_1x_2\bar{x_3}x_4$ $x_1x_2x_3\bar{x_4}$	1011 1101 1110		

Tabelle 2: Iterationstabelle

Aus dieser Tabelle erhält man die Primimplikanten.

	0000	0100	0110	1100	1011	1101	1110
0*00	1	1					
*1*0		1	1	1			1
110*				1		1	
1011					1		

Tabelle 3: Überdeckungstabelle

Wie man aus Tabelle 3 sehen kann werden alle zuvor gefundenen Primimplikanten benötigt.

Die minimierte Funktion lautet also:

$f = \bar{x_1}\bar{x_3}\bar{x_4} + x_2\bar{x_4} + x_1x_2\bar{x_3} + x_1\bar{x_2}x_3x_4$

1.5 Heuristische Verfahren zur Logiksynthese

Das Problem der zuvor genannten Optimierungsverfahren und anderer exakter Verfahren ist, daß diese NP-vollständige Probleme darstellen und der beste Algorithmus von diesen exponentielle Laufzeit hat.
Für die Minimierung größerer Schaltungen kann man diese Verfahren also nicht gebrauchen. Dort werden Heuristiken verwendet, die nicht garantieren können, daß eine Lösung mit minimaler Anzahl von Produkttermen erzeugt wird.
Mann kann sich solche Verfahren so vorstellen, daß iterativ aus einer gegebenen Lösung solange neue Lösungen generiert werden, bis eine bessere gefunden ist. Hat man eine bessere Lösung gefunden, versucht man ausgehend von dieser noch bessere zu finden u.s.w. Dieser Vorgang wird solange wiederholt, bis keine Kostenreduktion mehr möglich ist.

1.6 Minimierung einer ALU mittels Faltung

Der Schaltungsaufbau nach dem Prinzip der Faltung dient dazu den Platzaufwand einer ALU möglichst gering zu halten. Man versucht dabei möglichst schon realisierte Funktionen in die neu erstellten Operationen einzubinden. Um den Datenfluß dann noch zu ermöglichen werden Multiplexer eingesetzt, die man sich als eine Weiche vorstellen kann mit der man bestimmte anliegende Leitungen durchschalten kann.

Ich möchte das Prinzip der Faltung an einem kleinen Beispiel erläutern:

Wir nehmen an, wir hätten in eine ALU ein Subtraktionswerk, ein Additionswerk und ein Negierwerk zu implementieren. Würde man alles einzeln realisieren, so würde das in etwa wie folgt aussehen:

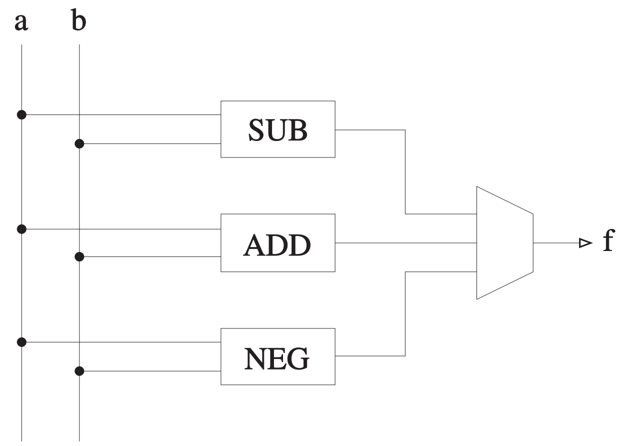

Abbildung 5: Aufbau nach der einfachen Methode

Wie man leicht sieht nimmt der Aufbau nach dieser Methode recht viel Platz ein, da jedes Rechenwerk einzeln realisiert wird.

Ein Subtraktionswerk ist jedoch gar nicht nötig, wenn man ein Additionswerk und ein Negierwerk hat, da man aus diesen beiden ein Subtraktionswerk realisieren kann.
Durch diese Feststellung ist es möglich Schaltungsplatz zu sparen. Man muß sich nur noch überlegen, wie man realisieren kann, daß man alle Schaltwerke auch einzeln verwenden kann. Abbildung 6 zeigt die Realisierung dieses Beispiels.

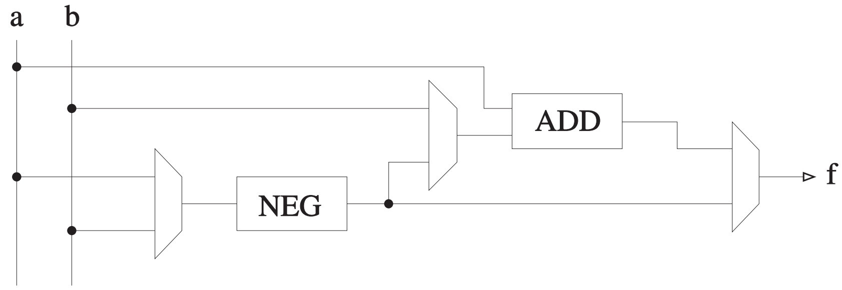

Abbildung 6: Aufbau nach der Methode der Faltung

1.7 Logiksynthese nach dem Verfahren von Buijs und Lengauer

Bei diesem Verfahren wird im Gegensatz zu den bisherigen Verfahren die in einer Hochsprache beschriebene Logik in eine symbolische Logik umgesetzt, in der viele Schaltungsdatails noch durch Symbole realisiert sind.
Es werden zum Beispiel die Steuerleitungen zu verschiedenen Rechenwerken zuerst durch Symbole realisiert und erst wenn die Rechenwerke wirklich optimiert sind das Steuerwerk erzeugt und optimiert.
Durch dieses Vorgehen ist die Logik nicht so komplex, weil viele Signale zu diesem Zeitpunkt noch nicht als binäre Signale verarbeitet werden, sondern als Symbole. Man braucht also nicht mit den zuvor beschriebenen heuristischen Verfahren zu arbeiten.
Das Verfahren von Buijs und Lengauer macht sich außerdem die Technik der Faltung zu Nutze. Es werden mit Hilfe von Symbolen bereits entworfene Schaltwerke mehrfach verwendet.
Außerdem wurden einige Algorithmen entworfen die eine noch bessere Optimierung erlauben als die herkömmlichen. Ich möchte die Algorithmen an dieser Stelle jedoch nicht vorstellen, da sie den Rahmen dieses Vortrages sprengen würden und deswegen eines eigenen Vortrags bedürfen, der sehr umfangreich würde. Wen ich dennoch nicht abgeschreckt habe, möchte sich die Promotinsarbeit von Herrn Buijs besorgen.
Praktisch wurde dieses Verfahren in dem CLASSY-Tool (Cadlab ALU Synthesis System) des Cadlab realisiert, auf welches ich am Ende meines Vortrages noch näher eingehen möchte.

2 ALU-Konstruktion

Die ALU (engl. arithmetic logical unit) ist das Rechenwerk eines Computers. Sie führt die arithmetischen und logischen Operationen aus. Die benötigten Operanden und Steuersignale werden dem Rechenwerk durch das Steuerwerk zur Verfügung gestellt.

2.1 Typische Komponenten einer ALU

2.1.1 Addierwerke

Aus 2 der in Abbildung 1 vorgestellten Halbaddierer und einem ODER-Gatter kann man, wie Abbildung 7 zeigt, einen Volladdierer konstruieren der 2 binäre Ziffern und einen Übertrag mit Übertrag addieren kann.

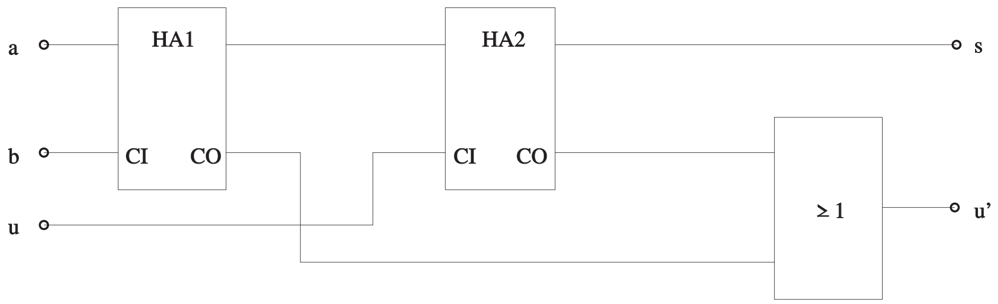

Abbildung 7: Volladdierer

Durch Parallelschaltung solcher Volladdierer entsteht der "ripple carry"-Addierer. (Siehe Abbildung 8.) Er ist einer der am einfachsten aufgebauten Addierer. Durch das Weitergeben des Übertrags ist er jedoch ziemlich langsam.

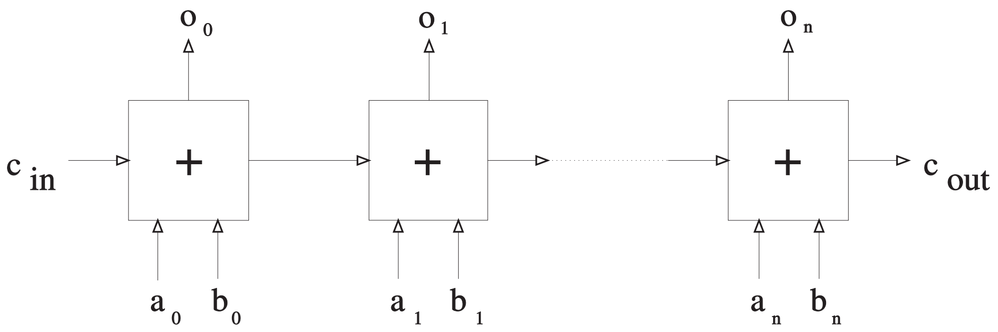

Abbildung 8: "ripple carry"-Addierer

Weil man schnellere Addierer brauchte hat man den "carry-lookahead"-Addierer entwickelt. Wie der "ripple carry"-Addierer verwendet auch er Volladdierer für die Addition.
Der Übertrag wird jedoch nicht seriell weitergegeben, sondern in einem eigenen Schaltkreis verarbeitet. (Siehe Abbildung 9.)
Durch dieses Verfahren wird der Addierer wesendlich schneller, da die Addition jetzt parallel ausgeführt werden kann.

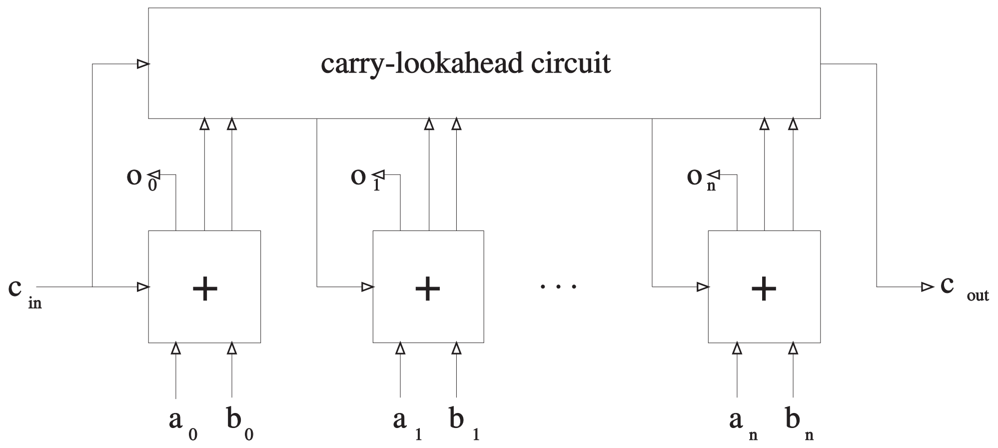

Abbildung 9: "carry-lookahead"-Addierer

Die Konstruktion eines "carry-lookahead"-Addierers erfordert bei größerer Stellenanzahl einen sehr hohen Schaltungsaufwand, den man jedoch oft nicht gebrauchen kann. Deshalb verwendet man häufig Mischformen von "carry-lookahead"- und "ripple carry"-Addierer. Es werden dann zum Beispiel, um einen 12-Bit-Addierer zu erreichen, 3 4-Bit-"carry-lookahead"-Addierer zu einem Addierwerk verknüpft.

2.1.2 Multiplizierwerk

Die Multiplikation zweier ganzer Zahlen wird in einem Multiplizierwerk analog zu der gläufigen Papier-und-Bleistift-Methode durch wiederholte Addition und Stellenverschiebung durchgeführt.
Abbildung 10 zeigt ein paralleles Mutiplizierwerk füer 3-Bit-lange Zahlen x und y. An den Eingängen der Volladdierer liegen die UND-verknüpften Ziffern der beiden Binärzahlen an. Man sollte um die Funktion dieses Multiplizierwerkes zu verstehen, diesen einfach mal mit einer Beispielrechnung durchlaufen.

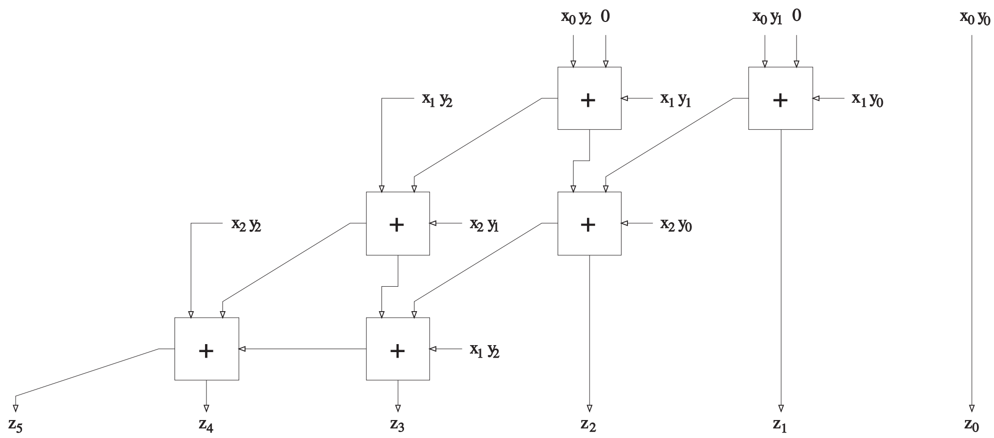

Abbildung 10: paralleles 3-Bit-Multiplizierwerk

2.1.3 Dividierwerk

Auch die Division zweier Zahlen im Divisionswerk wird wie die Division nach der Papier-und-Bleistift-Methode durch wiederholte Subtraktion und Stellenverschiebung realisiert.
Zur Realisierung eines Divisionswerkes wurde die Divisionszelle entwickelt, auf der ein solches Divisionswerk aufgebaut werden kann. Einen solchen Baustein kann man sich wie folgt vorstellen:

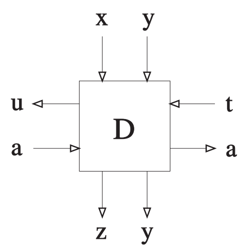

Abbildung 11: Divisionszelle

Im Inneren der Zelle sind folgende Verknüpfungen definiert:

$$z = x \oplus \bar{a}(y \oplus t)$$
$$u = \bar{x}y + \bar{x}t + yt$$

Der Zusammenhang zwischen a und z stellt sich wie folgt dar:

$$z = \begin{cases} x - (y + t) & : \ falls \quad a = 0 \\ x & : \ sonst \end{cases}$$

Ein paralleles Divisionswerk wird dann mit Hilfe dieser Divisionszellen wie in Abbildung 12 realisiert.

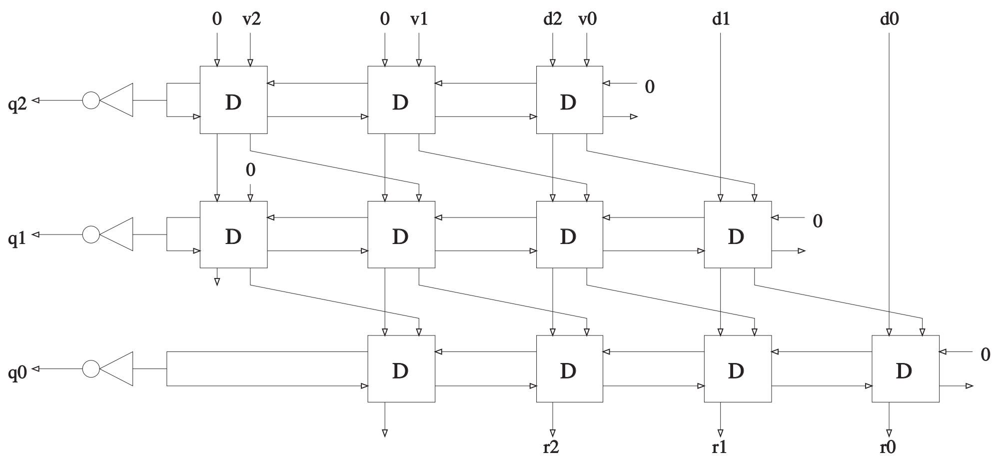

Abbildung 12: Dividierwerk für eine 3-Bit-Integerzahl

In der ersten Zeile wird vom Dividend $d_2, ..., d_0$ der Divisor $v_2, ..., v_0$ subtrahiert. Ist das Ergebnis größer Null, gilt immer a=0 und das Ergebnis wird am Ausgang z der D-Zellen an die nächste Zeile durchgereicht. Ist das Ergebnis der Subtraktion kleiner Null, gilt immer a=1 und die Eingangsparameter werden durchgereicht. Dieser Vorgang setzt sich bis zuer letzten

Zeile analog fort und liefert schließlich das gewünschte Ergebnis. Um die Funktion genau zu verstehen empfehle ich auch hier einfach mal eine Beispieldivision durchzuführen.

2.2 Steuerwerk

Das Steuerwerk (auch Leitwerk genannt) ist die zentrale Komponente einer ALU. Es hat folgende Aufgaben:

- Laden der Programmbefehle aus dem Speicher.
- Decodierung der Befehle.
- Interpretation der Befehle.
- Versorgung der an der Ausführung der Befehle beteiligten Funktionseinheiten mit den nötigen Steuersignalen.

Da an der Ausführung eines Befehls das Rechenwerk und der Speicher beteiligt sind, ist das Steuerwerk mit all diesen Komponenten verbunden. Den prinzipiellen Aufbau zeigt Abbildung 13.

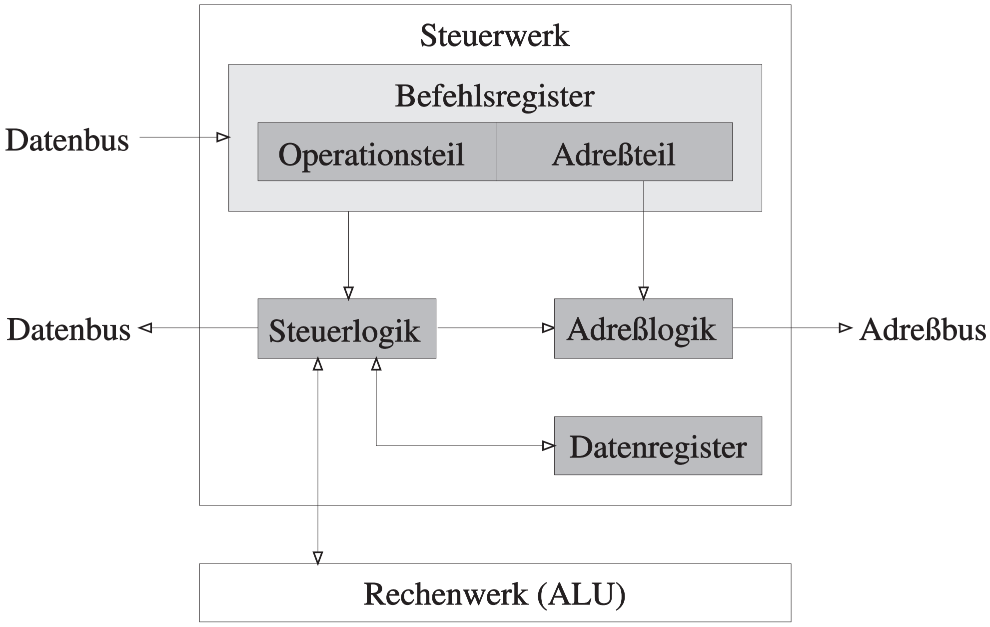

Abbildung 13: Aufbau eines typischen Steuerwerkes

2.3 Erzeugung von ALUs mit Hilfe des CLASSY-Tools

Da es insbesondere bei ALUs mit vielen Operationen schwierig ist diese noch von Hand zu erzeugen sind Tools entwickelt worden mit deren Hilfe sich der gesamte Logikentwurf auf Gatterebene und die anschließende Logiksynthese automatisieren läßt.
Solchen Tools übergibt man die logische Beschreibung einer ALU in einer Hochsprache und diese setzt sie dann in Gatterlogik um.
Die meisten dieser Tools gehen nach dem folgendem Verfahren vor:

1. Operationen wie +,−,<... werden einzeln in Gatterschaltungen umgesetzt, die einer Bibliothek entnommen werden.
2. Die Operationen werden logisch verknüpft.
3. Die entstandene Gatterschaltung wird optimiert.
4. Die optimierte Gatterschaltung wird auf eine Technologie abgebildet.

Bei einer Logik die nicht so regulär wie eine ALU aufgebaut ist ist dieses Verfahren recht gut, bei der Konstruktion einer ALU versagt dieses Verfahren jedoch, da es deren Regularität nicht berücksichtigt. Allerdings hat dieses Verfahren den Vorteil, das es technologieunabhängig ist.
Im Rahmen seiner Promotion entwickelte Frank Buijs im Cadlab jedoch das bereits beschriebene Verfahren, welches die Regularität einer ALU berücksichtigt und entwickelte das CLASSY-Tool, welches eine Optimierung nach dieser Methode erlaubt.
Bei diesem Verfahren wird die Logik ähnlich wie beim Handentwurf entworfen.
Anders als bei anderen Verfahren werden bereits bei der Übersetzung der Logik aus der Hochsprache Operationen verschmolzen. Die zu dem CLASSY-Tool gehörende Bibliothek enthält wesendlich komplexere Logik, als die der herkömmlichen Tools. Es werden für jede Operation der ALU eine oder mehrere Realisierungsalternativen erzeugt. So ist es möglich für jeden Fall die beste Realisierung auszuwählen. Die Auswahl geschieht mit Hilfe einer Kostenfunktion, in der der Benutzer Fläche und Verzögerungszeit gewichten kann.
Bei der Realisierung des Steuerwerkes wird mit der bereits beschriebenen symbolischen Logik gearbeitet mit deren Hilfe eine Decoderlogik ähnlich wie bei einem Handentwurf erzeugt wird. Bis zur Berechnung der internen Steuerleitungen wird mit einem Satz von Symbolen gearbeitet. Erst nach der Logikoptimierung werden die symbolischen Werte auf einen Satz binärer Werte abgebildet und das Steuerwerk erzeugt.
Zum Zeitpunkt der Entstehung des CLASSY-Tool lieferte es eine um bis zu 50% kleinere Logik als herkömmliche Systeme.

Literatur

[Esc93] B. Eschermann. *Funktionaler Entwurf digitaler Schaltungen.* Springer-Lehrbuch, 1993.

[Nag93] C. Nagel, editor. *Projektgruppe Systementwurf.* WS1992/93 SS1993

[Bui91] F. Buijs. Synthesis of multi-level logic with one symbolic input. In *Proceedings of the European Conference on Design Automation - EDAC,* pages 60 - 64. IEEE, 1991.

[Bui92] F.Buijs. Alu synthesis from hdl descriptions to optimized multi-level logic. In *European Design Automation Conference - Eurodac,* pages 175 - 180. IEEE, 1992

[Ram93] F. J. Rammig. *Vorlesung Rechnerarchitektur* WS1993/94

[Ram94] F. J. Rammig. *Vorlesung Spezielle Probleme des Hardwareentwurfs* SS1994