Minimierung von Schaltfunktionen mit dem Quine-McCluskey-Verfahren

Dietmar Sommerfeld

1 Einführung

Schaltfunktionen, die in einer Normalform gegeben sind oder die aus einer Funktionstabelle hergeleitet wurden, enthalten oft redundante Terme. Vor der Realisierung in Hardware (z.B. mit programmierbaren Logikbausteinen) erfolgt daher in der Regel eine Vereinfachung. Ziel der Vereinfachung ist es, die Funktionsgleichungen in möglichst kurzen Ausdrücken darzustellen, d.h. mit möglichst wenig Variablen und möglichst wenig Verknüpfungen. Das Quine-McCluskey-Verfahren ist das Standardverfahren zur exakten computergestützten Vereinfachung von Schaltfunktionen.

2 Verfahrensablauf

Das Minimierungsverfahren nach W. Quine und E.J. McCluskey ist ein algorithmisches Verfahren zur exakten Minimierung, welches sich gut in ein Computerprogramm übertragen läßt. Das Verfahren arbeitet mit Tabellen und geht von Schaltfunktionen aus, die in der *kanonisch disjunktiven Normalform* (KDNF) vorliegen. Die KDNF einer Schaltfunktion besteht aus der disjunktiven (ODER) Verknüpfung aller Minterme. Die *Minterme* einer Funktion sind UND-Verknüpfungen, die alle Schaltvariablen einmal negiert oder nicht negiert enthalten und für die die Schaltfunktion den Funktionswert 1 annimmt.

Der Ablauf des Quine-McCluskey-Verfahrens unterteilt sich in zwei Abschnitte: Im ersten Teil des Minimierungsprozesses werden die sogenannten *Primterme* oder auch *Primimplikanten* der Schaltfunktion bestimmt. Im zweiten Teil wird dann eine *minimale Überdeckung* der Minterme durch die Primterme ermittelt.

Am einfachsten läßt sich das Verfahren anhand eines Beispiels erläutern. Dazu soll hier die folgende Funktion betrachtet werden:
$f(x_1, x_2, x_3, x_4) = (\overline{x}_1 \wedge \overline{x}_2 \wedge \overline{x}_3 \wedge \overline{x}_4) \vee (x_1 \wedge \overline{x}_2 \wedge \overline{x}_3 \wedge \overline{x}_4) \vee (\overline{x}_1 \wedge x_2 \wedge \overline{x}_3 \wedge \overline{x}_4) \vee (x_1 \wedge x_2 \wedge \overline{x}_3 \wedge \overline{x}_4) \vee (x_1 \wedge \overline{x}_2 \wedge x_3 \wedge \overline{x}_4) \vee (x_1 \wedge x_2 \wedge x_3 \wedge \overline{x}_4) \vee (\overline{x}_1 \wedge \overline{x}_2 \wedge \overline{x}_3 \wedge x_4) \vee (\overline{x}_1 \wedge x_2 \wedge \overline{x}_3 \wedge x_4) \vee (\overline{x}_1 \wedge \overline{x}_2 \wedge x_3 \wedge x_4) \vee (x_1 \wedge \overline{x}_2 \wedge x_3 \wedge x_4) \vee (x_1 \wedge x_2 \wedge x_3 \wedge x_4)$.

Die Schreibweise erfolgt im weiteren mit Blick auf die Umsetzung im Computer in Binärnotation, das bedeutet, die Terme werden nicht durch die negierten und nichtnegierten Variablen dargestellt, sondern durch ihre *Binäräquivalente*. Dabei steht

1	für eine nicht negierte Variable,
0	für eine negierte Variable,
-	für eine nicht auftretende Variable.

Dez	x_4	x_3	x_2	x_1	$f(x_1, x_2, x_3, x_4)$
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

Tabelle 1: Funktionstabelle zum Quine-McCluskey-Verfahren

Außerdem wird die Reihenfolge der Variablen in den Termen vertauscht, so daß die Indizes von rechts nach links absteigend sortiert sind. Beispiele:

$$\begin{array}{lcl} x_4 \wedge \overline{x}_3 \wedge x_2 \wedge \overline{x}_1 & \hat{=} & 1\,0\,1\,0 \\ \overline{x}_4 \wedge \overline{x}_3 \wedge x_1 & \hat{=} & 0\,0\,\text{-}\,1 \\ x_4 \wedge \overline{x}_1 & \hat{=} & 1\,\text{-}\,\text{-}\,0 \end{array}$$

Der erste Teil der Minimierung kann in vier Schritte unterteilt werden. Er beginnt mit dem Aufstellen der Wertetabelle. Die Wertetabelle der Funktion f ist in Tabelle 1 dargestellt.

Die Reihenfolge der Eingangskombinationen wird so angeordnet, daß die Binärwerte aufsteigenden Dualzahlen entsprechen. Zugleich entsprechen die Eingangskombinationen, deren Funktionswert 1 beträgt, den Mintermen der Funktion in Binäräquivalentschreibweise. Zur Vereinfachung der Bezeichnung der Minterme werden alle Terme mit den entsprechenden Dezimalzahlen als Nummern versehen.

Wie man sieht, ist der Funktionswert von f nur fünfmal 0, aber elfmal 1. In der Praxis wäre es daher vorteilhafter, die komplementäre Funktion zu verwenden und die Funktionswerte am Ausgang zu invertieren. Hier soll die Funktion f aber nur als Beispiel dienen.

Der zweite Schritt besteht darin, die aktuellen Terme (zu Beginn also die Minterme) nach Gewichten in Gruppen zu ordnen. Das Gewicht eines Terms ist dabei durch die Anzahl der Einsen im Binäräquivalent bestimmt. Gruppe 0 enthält daher die Terme ohne Eins, Gruppe 1 Terme mit einer Eins, Gruppe 2 mit zwei Einsen, usw.. Die nach Gruppen geordneten Terme werden dann in eine neue Tabelle eingetragen.

Im dritten Schritt werden die Terme von aufeinanderfolgenden Gruppen entsprechend der Regel

$$(x_1 \wedge x_2) \vee (x_1 \wedge \overline{x}_2) = x_1 \wedge (x_2 \vee \overline{x}_2) = x_1$$

zu kürzeren Termen zusammengefaßt, die eine Variable weniger enthalten. Für die Zusammenfassung gilt: Zwei Terme sind genau dann zusammenfaßbar, wenn sie sich nur an einer Stelle um 0 und 1 unterscheiden. Die entfallene Variable wird im Binäräquivalent durch - gekennzeichnet. Zum systematischen Vorgehen beginnt man mit dem ersten Term einer Gruppe und versucht, ihn mit allen Termen der nächsten Gruppe zu verschmelzen. Dabei können die Terme auch mehrmals verwendet werden. Alle Terme, die an einer Verschmelzung beteiligt waren, werden mit einem $\surd$ gekennzeichnet.

Auf die neu entstandenen, zusammengefaßten Terme werden dann wieder Schritt zwei und Schritt drei angewendet. Dies wird solange wiederholt, bis keine Verschmelzung mehr durchgeführt werden kann. Entstehen bei der Bildung einer Tabelle in einer Gruppe mehrfach gleiche Terme, so werden diese bis auf einen gestrichen (siehe Tab. 4). Mit den dezimalen Nummern, die in der zweiten Spalte der Tabellen eingetragen werden, wird vermerkt, aus welchen Mintermen der neue Term gebildet wurde.

Gruppe	Dez	x_4	x_3	x_2	x_1	
0	0	0	0	0	0	$\surd$
1	1	0	0	0	1	$\surd$
	2	0	0	1	0	$\surd$
	8	1	0	0	0	$\surd$
2	3	0	0	1	1	$\surd$
	5	0	1	0	1	$\surd$
	10	1	0	1	0	$\surd$
	12	1	1	0	0	$\surd$
3	7	0	1	1	1	$\surd$
	13	1	1	0	1	$\surd$
4	15	1	1	1	1	$\surd$

Tabelle 2: Tabelle der 1. Iteration

Für die Beispielfunktion f müssen die Schritte 2 und 3 dreimal wiederholt werden. Die Ergebnisse der einzelnen Iterationen sind in den Tabellen 2, 3 und 4 angegeben.

Im letzten Schritt des ersten Teils der Minimierung werden alle Terme zusammengetragen, die während der vorangegangenen Iterationen nicht mit einem $\surd$ gekennzeichnet wurden. Sie werden *Primterme* oder auch *Primimplikanten* genannt. Ein Primterm p von f ist ein Implikant der Funktion mit minimaler Länge, das heißt für alle Eingangskombinationen die p erfüllen, hat auch f den Funktionswert 1. Minimale Länge bedeutet, daß in p keine weitere Variable gestrichen werden kann, ohne daß die Eigenschaft „Implikant der Funktion“ verloren geht. Im Beispiel sind die Primterme mit geklammerten Zahlen durchnumeriert.

Die disjunktive Minimalform einer Schaltfunktion setzt sich aus den ermittelten Primtermen zusammen. Vielfach hat eine Funktion jedoch mehr Primterme als nötig wären, um alle Einswerte der Funktion vollständig zu erfassen. Wie durch die Nummern in der zweiten Spalte der Tabellen dargestellt, ist jeder Primterm aus bestimmten Mintermen entstanden. Diese Minterme werden von dem Primterm *überdeckt*, das heißt der Primterm ist genau dann erfüllt, wenn

Gruppe	Dez	x_4	x_3	x_2	x_1	
0	0,1	0	0	0	-	$\surd$
	0,2	0	0	-	0	$\surd$
	0,8	-	0	0	0	$\surd$
1	1,3	0	0	-	1	$\surd$
	1,5	0	-	0	1	$\surd$
	2,3	0	0	1	-	$\surd$
	2,10	-	0	1	0	$\surd$
	8,10	1	0	-	0	$\surd$
	8,12	1	-	0	0	(1)
2	3,7	0	-	1	1	$\surd$
	5,7	0	1	-	1	$\surd$
	5,13	-	1	0	1	$\surd$
	12,13	1	1	0	-	(2)
3	7,15	-	1	1	1	$\surd$
	13,15	1	1	-	1	$\surd$

Tabelle 3: Tabelle der 2. Iteration

Gruppe	Dez	x_4	x_3	x_2	x_1	
0	0,1;2,3	0	0	-	-	(3)
	~~0,2;1,3~~	~~0~~	~~0~~	~~-~~	~~-~~	
	0,2;8,10	-	0	-	0	(4)
	~~0,8;2,10~~	~~-~~	~~0~~	~~-~~	~~0~~	
1	1,3;5,7	0	-	-	1	(5)
	~~1,5;3,7~~	~~0~~	~~-~~	~~-~~	~~1~~	
2	5,7;13,15	-	1	-	1	(6)
	~~5,13;7,15~~	~~-~~	~~1~~	~~-~~	~~1~~	

Tabelle 4: Tabelle der 3. Iteration

einer der Minterme erfüllt ist aus denen er entstanden ist. Andererseits können Minterme in mehreren Primtermen enthalten sein.

Ziel des zweiten Teils des Quine-McCluskey-Verfahrens ist es daher, eine *minimale Überdeckung* durch *wesentliche Primterme* zu finden, das heißt eine kleinstmögliche Auswahl von Primtermen, die alle Minterme überdeckt. Oft existieren dafür mehrere Lösungen mit gleicher Primtermanzahl, aber unterschiedlich langen Primtermen. Dann werden die kürzesten Primterme zur Darstellung der Funktion ausgewählt. Haben die Primterme die gleiche Länge, existieren mehrere gleichwertige Lösungen für die Minimalform der Schaltfunktion.

Auch der zweite Teil arbeitet wieder mit Tabellen, den sogenannten *Primtermtabellen* oder *Primimplikantentafeln*. Die Primtermtabelle ist so aufgebaut, daß über den Spalten der Tabelle die Nummern der Minterme stehen und an den Zeilen die Nummern der Primterme. Für jeden Primterm werden alle Minterme markiert, die durch ihn abgedeckt werden. In der Tabelle wird dies durch ein Kreuz im Schnittpunkt der betreffenden Zeile und Spalte dargestellt.

Jetzt sucht man alle Spalten, in denen nur eine Markierung steht und kennzeichnet diese durch $\otimes$. Die Primterme der zu diesen Markierungen gehörigen

		0	1	2	3	5	7	8	10	12	13	15
(1)	8,12							×		×		
(2)	12,13									×	×	
(3)	0,1,2,3	×	×	×	×							
(4)	0,2,8,10	×		×				×	⊗			
(5)	1,3,5,7		×		×	×	×					
(6)	5,7,13,15					×	×				×	⊗
		√		√		√	√	√	√		√	√

Tabelle 5: Primtermtabelle

Zeilen sind wesentliche Primterme und werden *Kernimplikanten* genannt. Sie müssen auf jeden Fall in der Minimalform erscheinen. Zugleich wird für alle Minterme, die von diesen Primtermen überdeckt werden, ein $\surd$ in der letzten Zeile notiert.

Für die Minterme, die nicht durch einen der Kernimplikanten abgedeckt werden, muß aus den verbleibenden Primtermen eine minimale Anzahl ausgewählt werden, die alle restlichen Minterme überdeckt. Dies nennt man die *minimale Restüberdeckung*. Im Beispiel werden die Minterme 1, 3 und 12 nicht von den Kernimplikanten erfaßt. Minterm 1 und 3 sind jedoch beide in den Primtermen (3) und (5) enthalten. Einer der beiden muß daher für die Restüberdeckung ausgewählt werden. Ebenso müssen Primterm (1) oder (2) enthalten sein, um Minterm 12 zu überdecken. Es ergeben sich also vier gleichwertige Lösungen für die Minimalform der Schaltfunktion. Sie wird gebildet aus der Disjunktion der Kernimplikanten und der minimalen Restüberdeckung.

$$\begin{aligned} f(x_1,x_2,x_3,x_4) &\hat{=} (\overline{x}_1 \wedge \overline{x}_3) \vee (x_1 \wedge x_3) \vee (\overline{x}_3 \wedge \overline{x}_4) \vee (\overline{x}_1 \wedge \overline{x}_2 \wedge x_4) \\ &\hat{=} (\overline{x}_1 \wedge \overline{x}_3) \vee (x_1 \wedge x_3) \vee (\overline{x}_3 \wedge \overline{x}_4) \vee (\overline{x}_2 \wedge x_3 \wedge x_4) \\ &\hat{=} (\overline{x}_1 \wedge \overline{x}_3) \vee (x_1 \wedge x_3) \vee (x_1 \wedge \overline{x}_4) \vee (\overline{x}_1 \wedge \overline{x}_2 \wedge x_4) \\ &\hat{=} (\overline{x}_1 \wedge \overline{x}_3) \vee (x_1 \wedge x_3) \vee (x_1 \wedge \overline{x}_4) \vee (\overline{x}_2 \wedge x_3 \wedge x_4) \end{aligned}$$

Der aufwendigste Teil des Quine-McCluskey-Verfahrens ist, auch wenn es zunächst nicht den Anschein haben mag, die Bestimmung der minimalen Restüberdeckung. Wie man bereits im Beispiel sehen kann, wächst die Anzahl der Möglichkeiten exponentiell mit der Anzahl der in Frage kommenden Primterme. Die Auswahl einer kleinsten Menge von Primtermen entspricht dem Lösen eines *Überdeckungsproblems* (Set Cover Problem) und ist $\mathcal{NP}$-vollständig.

3 Implementation

Bei der Umsetzung des Quine-McCluskey-Verfahrens in ein Computerprogramm sind zunächst geeignete Datenstrukturen für die Repräsentation der Schaltfunktionen und Terme zu wählen. Funktionen können durch ein Array von Termen dargestellt werden. Für Terme existiert die sogenannte Positional-Cube-Notation, die jede Variable eines Terms mit einem 2-Bit-Wert codiert. Vereinfachend kann die Speicherung der Terme aber auch durch ein Array von Ganzzahlen erfolgen, wobei 0, 1 und 2 die Binäräquivalente 0, 1 und '-' repräsentieren.

Ausgehend von der internen Repräsentation der Terme sind einige Funktionen zu implementieren, die Aktionen zwischen Termen betreffen und im Verlauf der Vereinfachung benötigt werden. Dazu zählt zunächst das Feststellen, ob zwei Terme miteinander verschmolzen werden können. Dies läßt sich auf das Zählen der unterschiedlichen Stellen der Terme zurückführen. Des weiteren folgen das Verschmelzen zu einem neuen Term sowie die Bestimmung der Länge eines vereinfachten Terms, also der Anzahl der Stellen, die nicht entfallen sind. Für das Aufstellen der Primtermtabelle kommt dann noch eine Funktion hinzu, die ermittelt, ob ein Primterm einen gegebenen Minterm überdeckt. Alternativ könnte diese Information über die Primterme aber auch im Laufe der Verschmelzungen gespeichert werden.

Der Algorithmus für die Minimierung einer Schaltfunktion mit dem Quine-McCluskey-Verfahren kann, wie in Abschnitt 2 beschrieben, auf zwei Funktionen aufgeteilt werden. Im ersten Teil erfolgt das Bestimmen der Primterme, indem man passende Terme mit Hilfe der zuvor genannten Funktionen zusammenfaßt. Die dabei entstehenden Terme werden in einem neuen Termarray abgelegt und mehrfach auftretende gleiche Terme bis auf einen gestrichen. Die Primterme, die sich nicht verschmelzen lassen, werden in einem weiteren Array gespeichert. Auf die Einteilung der Terme in gewichtete Gruppen kann verzichtet werden. Das Vergleichen aller Terme bedeutet bei der Implementation als Computerprogramm keinen wesentlichen Mehraufwand.

Im zweiten Teil des Algorithmus erfolgt die Ermittlung einer minimalen Überdeckung der Minterme der Funktion durch die Primterme. Da dies den aufwendigsten Teil der Berechnungen darstellt, sollte hier auch der größte Optimierungsaufwand betrieben werden. Einen umfassenden Einblick in die Thematik geben die Werke von Molitor [MS99] und Wegener [Weg96], aus denen einige der Überlegungen wiedergegeben werden sollen. In Anlehnung an das ältere Werk von Giloi und Liebig [GL80] erfolgt die Darstellung der Primtermtabelle hier jedoch um 90 Grad gedreht, d.h. Zeilen und Spalten sind vertauscht und an den Zeilen sind die Minterme und an den Spalten die Primterme angetragen. Diese Vorgehensweise erleichtert die Implementierung, da die meisten Aktionen einzelne Zeilen betreffen und eine Tabelle im Rechner als zweidimensionales Array gespeichert wird. In die Felder der Tabelle wird jeweils eine 1 oder eine 0 eingetragen, je nachdem, ob der betreffende Primterm den entsprechenden Minterm überdeckt oder nicht. Ziel der Überdeckungssuche ist es nun, in jeder Zeile genau einen, insgesamt aber möglichst wenige Primterme auszuwählen.

Zuvor werden jedoch drei Regeln zur Vereinfachung der Tabelle angewendet, um die Anzahl der zu überprüfenden Kombinationen zu verringern. Die *erste Reduktionsregel* lautet: Entferne aus der Primtermtabelle alle Kernimplikanten und alle Minterme, die von diesen überdeckt werden. Dazu werden zuerst die Kernimplikanten bestimmt und gespeichert. Danach werden alle durch sie überdeckten Zeilen aus der Tabelle entfernt, was sich bei einem Array von Zeilen sehr einfach durchführen läßt. Zum Schluß werden in den Zeilen der Tabelle die Spalten gelöscht, die den Kernimplikanten entsprechen.

Auf die neue Tabelle wird dann die *zweite Reduktionsregel* angewendet, welche (in unserem Fall der vertauschten Zeilen und Spalten) auf der Zeilendominanz beruht. Sie lautet: Entferne aus der Primtermtabelle alle Minterme, die einen anderen Minterm dominieren. Ein Minterm a dominiert dabei einen Minterm b, wenn a von den gleichen Primtermen wie b und eventuell weiteren Primtermen überdeckt wird. Minterm a braucht dann nicht weiter betrachtet zu

werden, da a durch die Überdeckung von b auf jeden Fall mit überdeckt wird. In der Tabelle werden hierzu alle Paare (a, b) von Zeilen spaltenweise miteinander verglichen und Zeile a entfernt, wenn der Wert in Zeile a stets größer oder gleich dem Wert in Zeile b ist.

Die *dritte Reduktionsregel* beschäftigt sich schließlich mit der Spaltendominanz und lautet: Entferne aus der Primtermtabelle alle Primterme, die durch einen anderen nicht längeren Primterm dominiert werden. Dabei dominiert ein Primterm a einen anderen Primterm b, wenn a die gleichen Minterme wie b und eventuell weitere Minterme überdeckt. Primterm b braucht dann nicht weiter betrachtet zu werden, da für die Minimalform der Schaltfunktion auf jeden Fall Primterm a ausgewählt werden kann. In der Tabelle werden deshalb alle Paare (a, b) von Spalten zeilenweise miteinander verglichen und Spalte b entfernt, wenn der Wert in Spalte a stets größer oder gleich dem Wert in Spalte b und Primterm a nicht länger als Primterm b ist.

```
Ausgangstabelle:     nach der 1. Regel:
  0 0 1 1 0 0          0 0 1 1
  0 0 1 0 1 0          0 0 1 1
  0 0 1 1 0 0          1 1 0 0
  0 0 1 0 1 0
  0 0 0 0 1 1        nach der 2. Regel:
  0 0 0 0 1 1          0 0 1 1
  1 0 0 1 0 0          1 1 0 0
  0 0 0 1 0 0
  1 1 0 0 0 0        nach der 3. Regel:
  0 1 0 0 0 1          0 1
  0 0 0 0 0 1          1 0

Zum Schluß kann noch einmal die 1. Regel angewendet werden.
```

Abbildung 1: Primtermtabellen

Die Vereinfachungen werden solange nacheinander wiederholt, bis keine der drei Reduktionsregeln mehr angewendet werden kann. Die dann erhaltene Primtermtabelle heißt *reduziert*. Ist die reduzierte Tabelle nicht leer, so spricht man von einem *zyklischen Überdeckungsproblem*. Für das Beispiel aus Abschnitt 2 lassen sich alle Primterme durch Anwendung der Reduktionsregeln ermitteln. Die ursprüngliche und die vereinfachten Primtermtabellen sind in Abbildung 1 dargestellt.

Zum Lösen zyklischer Überdeckungsprobleme existieren verschiedene Verfahren. Bei kleinen Primtermtabellen kann das *Verfahren von Petrick* angewendet werden, welches auf der sogenannten *Primtermfunktion* basiert. Diese beschreibt die Primtermtabelle in einer konjunktiven Normalform. Dazu werden für jeden der Minterme die Bezeichner der ihn überdeckenden Primterme UND-verknüpft. Anschließend bildet man die ODER-Verknüpfung der Ausdrücke aller Minterme. Die Primtermfunktion für das Beispiel lautet:

$$(3 \vee 4) \wedge (3 \vee 5) \wedge (3 \vee 4) \wedge (3 \vee 5) \wedge (5 \vee 6) \wedge (5 \vee 6) \wedge (1 \vee 4) \wedge 4 \wedge (1 \vee 2) \wedge (2 \vee 6) \wedge 6$$

Formt man diese Funktion durch Ausmultiplizieren in eine disjunktive Nor-

malform um, so erhält man eine Funktion, deren Terme alle Möglichkeiten zur Überdeckung der Minterme repräsentieren. Die minimale Überdeckung entspricht dann dem kürzesten Term.

Große Überdeckungsprobleme werden mit *Branch-and-Bound-Verfahren* gelöst, die das Problem sukzessive in kleinere Überdeckungsprobleme zerlegen. Ziel ist es, mit Hilfe der Berechnung von unteren Schranken für die Kosten der Probleminstanzen, den Suchbaum so klein wie möglich zu halten und aussichtslose Varianten nicht weiter zu verfolgen. Details dazu finden sich in [MS99] und [Weg96].

Hier sollen noch zwei weitere einfache Lösungsansätze für die Überdeckungssuche beschrieben werden. Der erste wurde bereits angedeutet und besteht darin, in jeder Zeile der Primtermtabelle stets eine mit 1 belegte Spalte auszuwählen. Dabei wird systematisch vorgegangen, so daß sukzessive alle gültigen Überdeckungen ermittelt werden. Für jede einzelne ist dann die Anzahl und Länge der benutzten Primterme zu bestimmen und zu vergleichen, ob die aktuelle Lösung eine Verbesserung darstellt.

Problematisch ist dieses Verfahren jedoch dann, wenn alle Minterme von vielen Primtermen überdeckt werden und eigentlich nur wenige Primterme für eine Überdeckung nötig wären. Die Lösung hierfür ist, noch eine weitere Variante zur Überdeckungssuche zu implementieren, die der intuitiven Überlegung folgt, zunächst zu versuchen, mit möglichst wenig Primtermen eine gültige Überdeckung zu erreichen und die Anzahl dann sukzessive zu erhöhen, falls dies nicht gelingt. Der Algorithmus bildet dazu jeweils alle möglichen Kombinationen mit einer bestimmten Anzahl von Primtermen. Wird eine gültige Auswahl von Primtermen gefunden, müssen allerdings noch alle weiteren Kombinationen mit gleicher Termanzahl untersucht werden, um festzustellen, ob sie gegebenenfalls kürzere Terme enthalten.

Beide Ansätze lassen sich sinnvoll miteinander kombinieren. Während die erste Variante nur gültige Primtermauswahlen betrachtet, müssen bei der zweiten alle gebildeten Kombinationen darauf untersucht werden, ob sie eine gültige Lösung darstellen. Für die erste Variante ergibt sich die Anzahl der zu testenden Möglichkeiten, indem man in jeder Zeile der Primtermtabelle die Summe der Einträge bildet und diese dann miteinander multipliziert. Bei der zweiten Variante ist die Anzahl der getesteten Möglichkeiten davon abhängig, wie schnell sich eine vollständige Überdeckung der Minterme ergibt. Im schlechtesten Fall müssen jedoch alle Kombinationen mit jeder Anzahl von ausgewählten Primtermen getestet werden, dann beträgt die Anzahl der Schleifendurchläufe 2 hoch Primtermanzahl. Die Entscheidung, welche der beiden Varianten zum Einsatz kommt, erfolgt deshalb auf der Basis eines Vergleichs dieses Wertes mit der Anzahl der gültigen Überdeckungen.

Bei sehr großen Überdeckungsproblemen kann die Bestimmung einer exakten Lösung wegen des exponentiellen Aufwands unwirtschaftlich werden. Dann empfiehlt sich der Einsatz heuristischer Verfahren. Hierbei wird die Forderung nach der besten Lösung aufgegeben, statt dessen liegt das Augenmerk bei heuristischen Verfahren auf ihrer Effienz, d.h in vertretbarer Zeit möglichst gute Lösungen zu erhalten. Das Standardverfahren der heuristischen Logikminimierung ist der ESPRESSO-Algorithmus.

Literatur

[SS96] Wolfram Schiffmann, Robert Schmitz: *Technische Informatik Band 1.* 3. Auflage. Berlin: Springer, 1996.

[Bor97] Johannes Borgmeyer: *Grundlagen der Digitaltechnik.* München: Carl Hanser Verlag, 1997.

[GL80] Wolfgang Giloi, Hans Liebig: *Logischer Entwurf digitaler Systeme.* 2. Auflage. Berlin: Springer, 1980.

[MS99] Paul Molitor, Christoph Scholl: *Datenstrukturen und effiziente Algorithmen für die Logiksynthese kombinatorischer Schaltungen.* Stuttgart: Teubner, 1999.

[Weg96] Ingo Wegener: *Effiziente Algorithmen für grundlegende Funktionen.* 2. Auflage. Stuttgart: Teubner, 1996.

[Weg87] Ingo Wegener: *The Complexity of Boolean Functions.* Stuttgart: Wiley-Teubner, 1987.

[Vas98] George Vastianos: *Boolean Functions' Minimisation Software based on the Quine-McCluskey method.* Athen: Software Notes, 1998.

[RP02] Peter Rechenberg, Gustav Pomberger: *Informatik-Handbuch.* 3. Auflage. München: Carl Hanser Verlag, 2002.