

4 Analyse und Synthese von Schaltnetzen

Dieses Kapitel behandelt die Synthese von zweistufigen AND-OR-Schaltnetzen (disjunktive und konjunktive Normalform) unter Anwendung von Wahrheitstafeln und der De Morgan'schen Gesetze. Im weiteren Verlauf werden auch Schaltnetze mit mehreren Stufen behandelt.

Das Ziel der Synthese ist die Begrenzung der Anzahl der Gattereingänge (Gate-Array-Technik, Vollkundenentwurf oder Standardbausteine) und die Optimierung der Laufzeiten und der Verlustleistung

4.1 Begriffe

Eine boolesche Funktion ist die Abbildung von n binären Variablen auf einen Wert (0 oder 1). Die einzelnen Variablen symbole (z.B. x , \bar{x}) werden Literale genannt. Variablen können drei Werte annehmen: 0, 1 und – (unbestimmt oder don't care). Literale lassen sich zu Min- und Maxtermen zusammensetzen. Ein Minterm ist ein Konjunktionsterm und definiert einen einzelnen Punkt innerhalb eines Wertefeldes von Variablen. Ein Maxterm hingegen definiert ein ganzes Wertefeld bis auf einen Punkt. Ein Implikant definiert eine Gruppe von 1-Punkten.

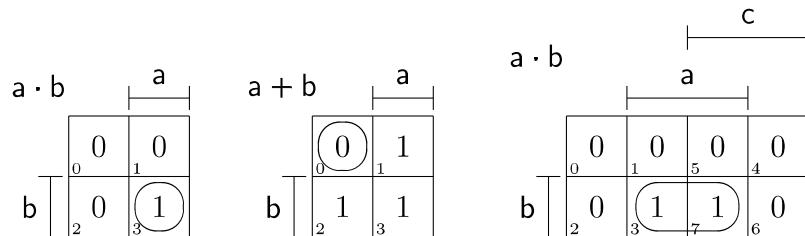


Abbildung 4.1: Ein Minterm, Maxterm und Implikant dargestellt in einem KV-Diagramm

Der Primimplikant ist ein Implikant, der so groß wie möglich ist, ohne dabei vollständig Teil anderer Implikanten zu sein. Dabei darf durch weglassen einer einzigen Variablen der Wert der Funktion nicht verändert werden.

Ein wichtiges Ziel der Analyse und Synthese ist die Optimierung der Funktion. Je nach Sichtweise kann „Optimierung“ verschiedenes bedeuten. Aus mathematischer Sicht kann beispielsweise ein Ergebnis mit möglichst wenig Gattern (Termen) und möglichst wenig

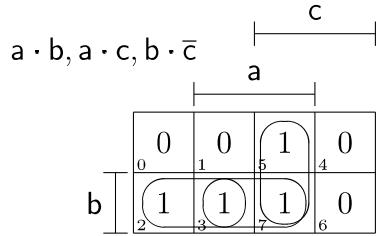


Abbildung 4.2: Primimplikanten in einem KV-Diagramm dargestellt

Eingängen (Literalen) als ideal angesehen werden. Aus physikalischer oder elektrotechnischer Sicht ist eine möglichst kleine Chipfläche (also wenige Bauelemente und gute Verdrahtbarkeit), eine minimale Laufzeit (wenige Gatterebenen, Fanin, Fanout) und eine minimale Verlustleistung (Gatterstruktur, Anzahl der Gatterebenen, Fanin, Fanout) optimal. Ein Problem ist, dass sich spezielle Schaltungstechniken wie dynamische Logik und Pass-Logik nur schwer in Logikansätze integrieren.

4.2 Gattersymbolik

Neben Funktionen, Wertetabellen und KV-Diagrammen ist noch eine weitere Darstellung etabliert - die Gattersymbolik. Bei der Gattersymbolik handelt es sich um eine grafische Darstellung, der die Grundstruktur einer Schaltung zu entnehmen ist. Dabei werden Terme der realisierten Funktion durch Grundgatter symbolisiert, deren Funktionalität den Grundfunktionen der Booleschen Algebra entsprechen. Die Anzahl der in den jeweiligen Term involvierten Eingangsvariablen spiegelt sich in der Anzahl der Eingänge eines jeden Gatters wieder. Die Negation von Eingangsvariablen und Termen wird durch eingangs- oder ausgangsseitige Kreise symbolisiert. In der Abbildung 4.3 wird die Funktion

$$y = \overline{(a \cdot \overline{b} \cdot c)} + (\overline{d} \oplus \overline{e})$$

dargestellt.

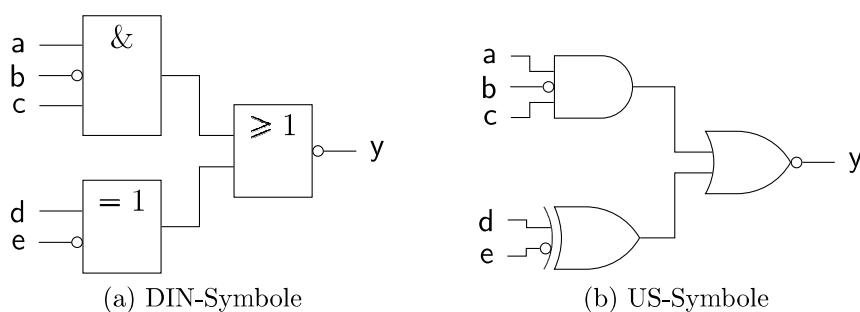


Abbildung 4.3: Beispiele für die Verwendung unterschiedlicher Gattersymbolik

Neben der DIN Symbolik aus Abbildung 4.3a, die auch im weiteren Verlauf des Skripts Verwendung findet, gibt es unter anderem auch die in Abbildung 4.3b gezeigte. Auf diese

stößt man hauptsächlich in englischsprachigen Publikationen, aber auch in Entwurfswerkzeugen, die im Digitalentwurf Gebrauch finden.

4.3 Elementare Verfahren zur Schaltnetzsynthese

Zur Synthese von Schaltungen betrachten wir drei Möglichkeiten:

- Die Realisierung durch Gatter mit Hilfe von kanonischen Abbildungen
- Die Realisierung durch Schalter mit Hilfe des Entwicklungssatzes von Shannon
- Iterative Strukturen

4.3.1 Kanonische Abbildungen

Bei der Bildung von kanonischen Abbildungen verwenden wir Normalformen. Dabei wird unterschieden zwischen der disjunktiven Normalform mit AND/OR-Strukturen (Darstellung der 1-Felder—NAND) und der konjunktiven Normalform mit OR/AND-Strukturen (Darstellung der 0-Felder—NOR).

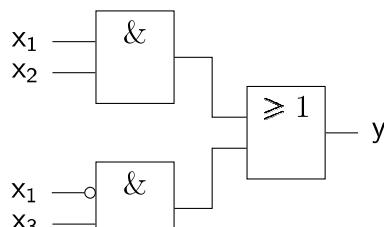
Beispiel 54. Disjunktive und konjunktive Normalform

	x_3		
y		x_1	
			x_2
	0	0	0
	0	1	1
	0	1	1
	0	1	1

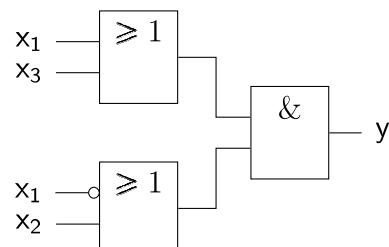
$$y = x_1 \cdot x_2 + \bar{x}_1 \cdot x_3 \quad \text{disjunktive Normalform}$$

$$\bar{y} = \bar{x}_1 \cdot \bar{x}_3 + x_1 \cdot \bar{x}_2 \quad \text{Umformung nach DeMorgan}$$

$$y = (x_1 + x_3) \cdot (\bar{x}_1 + x_2) \quad \text{konjunktive Normalform}$$



(a) Gatterrealisierung einer DNF



(b) Gatterrealisierung einer KNF

Abbildung 4.4: Gatterrealisierungen unterschiedlicher Normalformen

4.3.2 Entwicklungssatz (Shannon)

Der Entwicklungssatz nach Shannon erlaubt das Herausheben einer oder mehrerer Variablen aus einer Schaltungsfunktion.



Satz 12. Entwicklungssatz (Shannon)

$$y = f(x_1, x_2, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n)$$

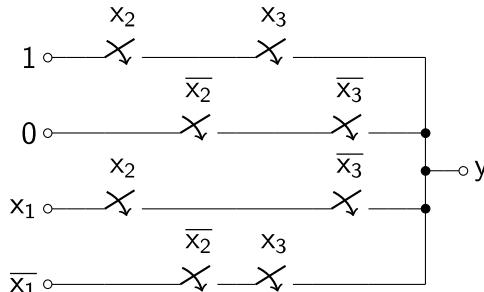
$$y = x_i \cdot f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n) + \bar{x}_i \cdot f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n)$$

Beispiel 55. Shannon

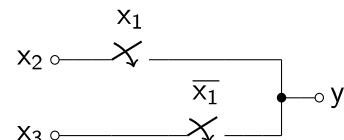
$$y = x_1 \cdot x_2 + \bar{x}_1 \cdot x_3$$

$$y = x_2 \cdot x_3 \cdot (1) + x_2 \cdot \bar{x}_3 \cdot (x_1) + \bar{x}_2 \cdot x_3 \cdot (\bar{x}_1) + \bar{x}_2 \cdot \bar{x}_3 \cdot (0) \quad \text{nach } x_2, x_3 \text{ entwickelt}$$

$$y = x_1 \cdot (x_2) + \bar{x}_1 \cdot (x_3) \quad \text{nach } x_1 \text{ entwickelt}$$



(a) nach x_2, x_3 entwickelt



(b) nach x_1 entwickelt

Abbildung 4.5: Schaltermatrizen für Entwicklung nach unterschiedlichen Variablen

4.3.3 Iterative Strukturen

Einfache iterative Netzwerke bestehen aus eindimensionalen Arrays von Gattern. Damit lassen sich jedoch nicht alle Funktionen abbilden.

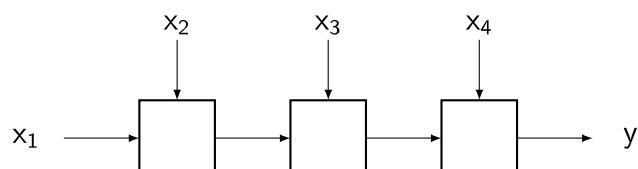
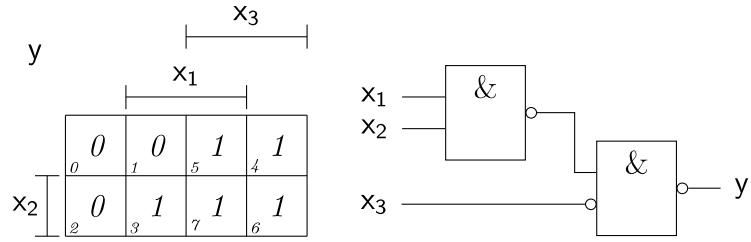


Abbildung 4.6: Grundstruktur eines eindimensionalen iterativen Schaltnetzes

Beispiel 56. Iterative Implementierung

$$y = x_1 \cdot x_2 + x_3$$



4.4 Minimale zweistufige AND-OR-Schaltnetze

Zur Minimierung einer Funktion in ein zweistufiges AND-OR-Schaltnetz müssen zuerst alle Primimplikanten der Funktion f bestimmt werden. Aus diesen werden dann die Primimplikanten ausgewählt, die alle 1-Punkte der Funktion f überdecken und minimale Kosten aufweisen.

4.4.1 Kosten

Kosten beschreiben die Zahl der Gatter und Eingänge z.B. bei der disjunktiven Normalform. Die Kosten eines Primimplikanten stehen in Relation zum Optimierungsziel (Zahl der Literale). Es wird davon ausgegangen, dass Literale sowohl komplementiert als auch nicht komplementiert zur Verfügung stehen.

Bei der Minimierung nach Gatteranzahl hat jeder Primimplikant mit zwei oder mehr Variablen einen Kostenwert von 1. Ein Primimplikant mit nur einer Variable hat einen Kostenwert von 0. Wenn mehrere Primimplikanten verknüpft werden, erhöhen sich die Kosten zusätzlich um 1. Der Kostenwert entspricht also der Anzahl der Gatter.

Soll nach Anzahl der Gattereingänge minimiert werden, so gilt für einen Primimplikanten mit einem Eingang ein Kostenwert von 1 und für Primimplikanten mit $n > 1$ Eingängen ein Kostenwert von $n + 1$.

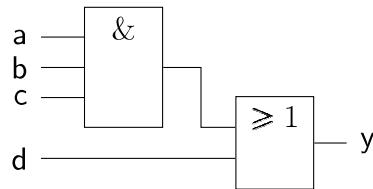


Abbildung 4.7: Schaltnetz mit zwei Gattern

4.4.2 Bestimmung der Primimplikanten

Bestimmung der Primimplikanten (Quine/McCluskey)



Eine Möglichkeit zur Bestimmung der Primimplikanten ist der Quine-McCluskey-Algorithmus:

1. Aufstellung einer Tabelle aller 1- und don't-care-Punkte (–) der Funktion f, sortiert nach der Anzahl der Variablen mit Wert 1, in die Gruppen S_0, S_1, \dots, S_n . Um don't-cares richtig zu beachten muss dabei auch der Funktionswert in die Tabelle eingetragen werden.
2. In allen benachbarten Gruppen S_i und S_{i+1} , für alle i mit $0 \leq i < n$, werden Paare gesucht, bei denen sich nur eine Variable unterscheidet. Aus diesen Paaren werden neue Implikanten gebildet, bei denen jeweils die unterschiedliche Variable auf den Wert „unbestimmt“ gesetzt wird. Der Funktionswert ist genau dann don't care, wenn beide kombinierten Terme an dieser Stelle don't care angeben, sonst 1. Die so erzeugten Implikanten werden in die Gruppe S'_i eingeordnet, die kombinierten Terme werden markiert.
3. Der zweite Schritt wird wiederholt, um beispielsweise S'_i mit S'_{i+1} zu S''_i zu kombinieren. Dabei müssen unbestimmte Variablen (don't care) immer übereinstimmen.
4. Wenn keine weiteren Paare gebildet werden können, sind alle Terme, die nie kombiniert wurden (also nicht markiert sind) Primimplikanten.

Beispiel 57. Bestimmung der Primimplikanten nach Quine/McCluskey

Eingangsvariable: x_3, x_2, x_1

1-Punkte: $\{(1, 0, 0), (0, 1, 0), (0, 0, 1), (1, 1, 0)\}$; m_4, m_2, m_1, m_6

(–)-Punkte: $\{(1, 1, 1)\}$; m_7

	x_3	x_2	x_1	f	
m_4	1	0	0	1	!
S_1	m_2	0	1	0	1
	m_1	0	0	1	1
S_2	m_6	1	1	0	1
S_3	m_7	1	1	1	–
					!
	1	–	0	1	
S'_1		–	1	0	1
S'_2		1	1	–	1

Ergebnis: Primimplikanten: $\{x_1 \cdot \bar{x}_2 \cdot \bar{x}_3, \bar{x}_1 \cdot x_3, \bar{x}_1 \cdot x_2, x_2 \cdot x_3\}$

Bestimmung der Primimplikanten (Tison 1965)

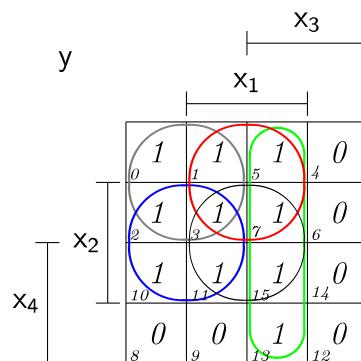
Eine weitere Möglichkeit zur Bestimmung der Primimplikanten ist der Algorithmus von Tison:

1. Die disjunktive Funktion $f = A + B + \dots + G$ mit $S = \{A, B, \dots, G\}$ ist als Ausgangspunkt gegeben. Zuerst werden alle Terme gelöscht, die vollständig überdeckt werden. Dann wird eine Variable ausgewählt, die in negierter und nichtnegierter Form vorkommt.
2. Bilden der Überdeckungsmengen aus jedem Term mit der nichtnegierten Variable mit jedem Term der negierten Variable. Die Überdeckungsmengenterme werden zur Menge S hinzugefügt und vollständig überdeckte Terme werden gelöscht.
3. Wiederholung von Schritt 2 mit allen anderen Variablen, die in negierter und nichtnegierter Form vorkommen. Wenn die Voraussetzung für eine Variable entfällt, entfällt auch dieser Schritt für diese Variable.
4. S enthält jetzt alle Primimplikanten.

Beispiel 58. Bestimmung der Primimplikanten nach Tison

$$y = x_1 \cdot x_2 \cdot x_4 + x_1 \cdot x_3 + x_2 \cdot \bar{x}_3 + \bar{x}_3 \cdot \bar{x}_4$$

Die beiden Variablen die sowohl negiert als auch nichtnegiert auftreten sind x_3 und x_4 . Von diesen wird zunächst x_3 gewählt. Ein mögliches Termpaar zum Bilden der Überdeckungsmengen ist $x_1 \cdot x_3$ und $x_2 \cdot \bar{x}_3$. Der Überdeckungsmengenterm lautet $x_1 \cdot x_2$. Er entsteht durch das „verunden“ der beiden Terme bei Vernachlässigung der vorher gewählten Variable x_3 . Das zweite und letzte mögliche Paar ist $x_1 \cdot x_3$ und $\bar{x}_3 \cdot \bar{x}_4$. Daraus folgt analog $x_1 \cdot \bar{x}_4$. Der Term $x_1 \cdot x_2 \cdot x_4$ der Ursprungsfunktion wird durch den neu gefundenen Term $x_1 \cdot x_2$ überdeckt und entfällt somit.



Ergebnis: Primimplikanten: $x_1 \cdot x_3$, $x_2 \cdot \bar{x}_3$, $\bar{x}_3 \cdot \bar{x}_4$, $x_1 \cdot x_2$, $x_1 \cdot \bar{x}_4$

Vergleich der Verfahren zur Bestimmung der Primimplikanten

Während die Tison-Methode vor allem bei großer Variablenzahl flexibel ist, kann die Quine-McCluskey-Methode don't cares berücksichtigen, ist günstiger für rechnergestützte Verfahren und vorteilhaft, wenn die Funktion durch Minterme vergeben ist.

4.4.3 Kostenoptimierung (minimale Überdeckung)

Die gefundenen Menge an Primimplikanten ist in der Regel nicht frei von Redundanz. Zur weiteren Kostenoptimierung müssen die Primimplikanten bestimmt werden, die zwar alle 1-Punkte der Funktion abdecken, sich dabei aber minimal Überdecken und somit minimale Kosten verursachen. Die Minimierung von Hand kann in tabellarischer Form durchgeführt werden. Dazu wollen wir vereinbaren, dass x_1 das niederwertigste Bit und x_n das höchstwertigste Bit ist.

Für eine Reihe der folgenden Verfahren können die bestimmenden Faktoren (Primimplikanten, Minterme) nach Kosten gewichtet werden. Da diese Gewichtung für das Verständnis der Verfahren jedoch unerheblich ist, gehen wir bei der Erläuterung der Verfahren von Primimplikanten und Mintermen mit egalisierter Gewichtung aus.

Wir werden vier Verfahren behandeln, die in der Regel in einer Kombination angewandt werden:

- wesentlicher Primimplikant
- Reihendominanz
- Spaltendominanz
- Branching



Der Einsatz einiger dieser Verfahren ist in der Praxis erst bei sehr komplexen Funktionen, die jedoch nicht didaktisch zur Erläuterung geeignet sind, ratsam. Im Folgenden werden daher Beispiele bemüht, die in dieser Form nicht unmittelbar aufgestellt werden können (zum Beispiel als Ergebnis eines Quine-McCluskey). Ihr Zustandekommen kann mit einer vorher erfolgten, in dem jeweiligen Beispiel aber nicht aufgeführten Anwendung anderer Minimierungsverfahren erklärt werden.

Methode der wesentlichen Primimplikanten

		m_1	m_2	m_4	m_6
P_1	$x_1 \cdot \overline{x_2} \cdot \overline{x_3}$	1			
P_2	$\overline{x_1} \cdot x_3$			1	1
P_3	$\overline{x_1} \cdot x_2$		1		1
P_4	$x_2 \cdot x_3$				1

Abbildung 4.8: Besetzungstabelle der Primimplikanten aus dem Beispiel 57

Ein wesentlicher Primimplikant überdeckt als einziger einen Minterm einer Funktion. Wenn P_i als einziger Primimplikant m_i überdeckt, muss P_i ein Bestandteil der minimalen Überdeckung sein. Alle Spalten (Minterme), die von P_i überdeckt werden, können dann eliminiert werden.

Beispiel 59. Methode des wesentlichen Primimplikanten anhand Abb. 4.8

Der Primimplikant P_3 überdeckt als einziger den Minterm m_2 und ist damit ein wesentlicher Primimplikant. P_3 überdeckt weiterhin m_6 . Es können also insgesamt zwei Spalten entfernt werden. Mit der Eliminierung von m_6 kann auch P_4 entfernt werden, denn der Funktionswert von m_7 ist lediglich don't care. Es bleibt damit die reduzierte Besetzungs-tabelle:

	m_1	m_4
P_1	1	
P_2		1

Somit ist die minimale Überdeckung der Primimplikanten P_1 , P_2 , P_3 bzw. $x_1 \cdot \bar{x}_2 \cdot \bar{x}_3$, $\bar{x}_1 \cdot x_3$, $\bar{x}_1 \cdot x_2$

Methode der Reihendominanz

Nachdem die Menge an Primtermen beispielsweise durch die Methode der wesentlichen Primimplikanten reduziert wurde, kann oft noch eine weitere Vereinfachung erfolgen.

Wie der Name Reihendominanz schon suggeriert, wird bei dieser Methode eine Reihe P_i gesucht, die eine andere Reihe P_j dominiert. Dominieren bedeutet, dass die Reihe P_i alle 1-Punkte der Reihe P_j abdeckt und zudem geringere (oder gleiche) Kosten aufweist ($C(P_i) \leq C(P_j)$). In diesem Fall kann die dominierte Reihe P_j entfernt werden, da so die minimale Kostenfunktion durch P_i erhalten bleibt.

Zusammengefasst: Seien M_{P_x} , M_{P_y} Mengen aller durch P_x oder P_y abgedeckten Minterme. Dann gilt für $M_{P_x} \subset M_{P_y}$: P_y dominiert P_x .

Beispiel 60. Methode der Reihendominanz

	m_a	m_b	m_c	m_d	m_e
P_1	1	1			
P_2	1	1	1		1
P_3			1	1	1
P_4	1			1	

In diesem Beispiel dominiert die Reihe P_2 die Reihe P_1 , die somit eliminiert werden kann. Daraus ergibt sich die reduzierte Überdeckungstabelle:

	m_a	m_b	m_c	m_d	m_e
P_2	1	1	1		1
P_3			1	1	1
P_4	1			1	

In dieser ist P_2 ein wesentlicher Primimplikant für m_b . Nach Anwenden der Methode des wesentlichen Primimplikanten bleibt nur die Spalte m_d übrig:

	m_d
P_3	1
P_4	1

Jetzt dominiert P_3 P_4 , allerdings dominiert P_4 auch P_3 . Der Primimplikant wird nach Kosten ausgewählt. In diesem Beispiel sind die minimalen Überdeckungen also: $\{P_2, P_3\}$ oder $\{P_2, P_4\}$.

Methode der Spaltendominanz

Dominanz lässt sich auch auf Spalten anwenden, dabei muss jedoch eine andere Methode verwendet werden. Eine Spalte m_i dominiert eine Spalte m_j , wenn jeder Primimplikant, der m_j überdeckt, maximal auch m_i überdeckt. Zum Beispiel kann m_j 1-Punkte in allen Reihen haben, in denen auch m_i 1-Punkte hat. Die Spalte m_j kann eliminiert werden, denn jede Überdeckungsmenge der reduzierten Tabelle überdeckt auch m_i .

Zusammengefasst: Seien P_{m_x}, P_{m_y} Mengen aller Primterme, die m_x oder m_y abdecken. Dann gilt für $P_{m_x} \subseteq P_{m_y}$: m_x dominiert m_y .

Beispiel 61. Methode der Spaltendominanz

	m_a	m_b	m_c	m_d	m_e
P_1	1	1			
P_2	1	1	1		1
P_3			1	1	1
P_4	1			1	

m_b dominiert m_a . Außerdem dominiert m_c m_e und umgekehrt. Es können also m_a und m_e (oder m_c) eliminiert werden.

	m_b	m_c	m_d
P_1	1		
P_2	1	1	
P_3		1	1
P_4			1

P_2 dominiert P_1 und P_3 dominiert P_4 (Reihendominanz). Das Ergebnis lautet somit: $\{P_2, P_3\}$ ist die minimale Überdeckung

Methode des Branching

Die drei bisher vorgestellten Methoden führen nicht zum Ziel, wenn keine der entsprechenden Kriterien zutreffen. Diese Tabellen werden als zyklisch bezeichnet und können mit der Methode des Branching reduziert werden.

Dazu wird eine Teilmenge R von Reihen ausgewählt, so dass jede Lösung ein Element von R enthalten muss. Danach wird P_i aus R gewählt. Die Spalten, die von P_i überdeckt werden, werden eliminiert. P_i ist nun ein Bestandteil der Lösung. Dieser Ablauf wird mit einer anderen Reihe von R wiederholt. Gegebenenfalls muss zum Auffinden des Minimums jedes Element von R gewählt werden.

Beispiel 62. Methode des Branching

	m_a	m_b	m_c	m_d	m_e	
P_1	1	1				1
P_2			1	1		
P_3	1			1		
P_4				1	1	
P_5				1	1	
P_6		1		1		

Es wird die Spalte m_a ausgewählt. Aus m_a ist ersichtlich, dass jede Überdeckungsmenge P_1 und/oder P_3 enthalten muss. Daraus folgt Branching um m_a . Es gibt zwei Möglichkeiten:

- P_1 wird gewählt und alle Spalten von P_1 werden eliminiert.
- P_3 wird gewählt und alle Spalten von P_3 werden eliminiert.

P_1	m_c	m_d	P_3	m_b	m_d	m_e
P_2	1		P_1	1		1
P_3	1		P_2	1		
P_4		1	P_4		1	1
P_5	1	1	P_5		1	
P_6		1	P_6	1	1	

Beide Tabellen können nach den bisherigen Regeln weiter reduziert werden. Die minimalen Überdeckungsmengen sind $\{P_1, P_5\}$, $\{P_3, P_6, P_1\}$, $\{P_3, P_6, P_4\}$ oder $\{P_3, P_1, P_4\}$. Gegenüber den anderen drei Lösungen weist $\{P_1, P_5\}$ die geringeren Kosten auf und ist deshalb die optimale Lösung dieser Spaltenauswahl.

Hinweis Branching kann mit jeder Spalte durchgeführt werden. Jedoch sollten aus Gründen minimaler Rechenzeit die Spalten mit geringer Anzahl von 1-Punkten gewählt werden.

4.5 Schaltnetze mit mehreren Stufen

Schaltnetze mit mehreren Stufen bieten einen Kompromiss aus Hardwareaufwand und Geschwindigkeit. Sie ermöglichen es, den Ein- und Ausgangslastfaktor zu beschränken (fanin, fanout, z.B. Gate-Array-Technik). Dies wird durch die Aufspaltung von Gattern mit vielen Eingängen möglich, denn dadurch kann die Zahl der Eingänge beschränkt werden. Die Ausgangslast sinkt somit.

4.5.1 Baumstruktur

Eine Schaltung, bei der die Ausgänge eines jeden Gatters nur mit einem Gattereingang verknüpft werden, wird als Baumstruktur bezeichnet.

Beispiel 63. Mehrstufiges Schaltnetz: Parityfunktion

$$f(x_1, x_2, \dots, x_n) = x_1 \oplus x_2 \oplus \dots \oplus x_n$$

Ist das Ziel eine Realisierung mit möglichst wenigen Stufen, so erfordert diese Realisierung die Verwendung von 2^{n-1} AND-Gatter mit jeweils n Eingängen, ein OR-Gatter mit 2^{n-1} Eingängen und n Inverter. Unter Berücksichtigung der Inverter ergeben sich insgesamt drei Stufen. Der Aufwand lässt sich jedoch reduzieren, indem Parityschaltungen für 2 Bit kaskadierend miteinander verbunden werden. Es sind dazu $\log_2 n$ bzw. unter Berücksichtigung der internen Struktur der einzelnen Stufen $3 \log_2 n$ Stufen notwendig. Der Aufwand verringert sich damit auf $3(n-1)$ NAND-Gatter mit 2 Eingängen und $2(n-1)$ Inverter. Die Erhöhung der Stufenzahl führt jedoch zur Erhöhung der Laufzeiten.

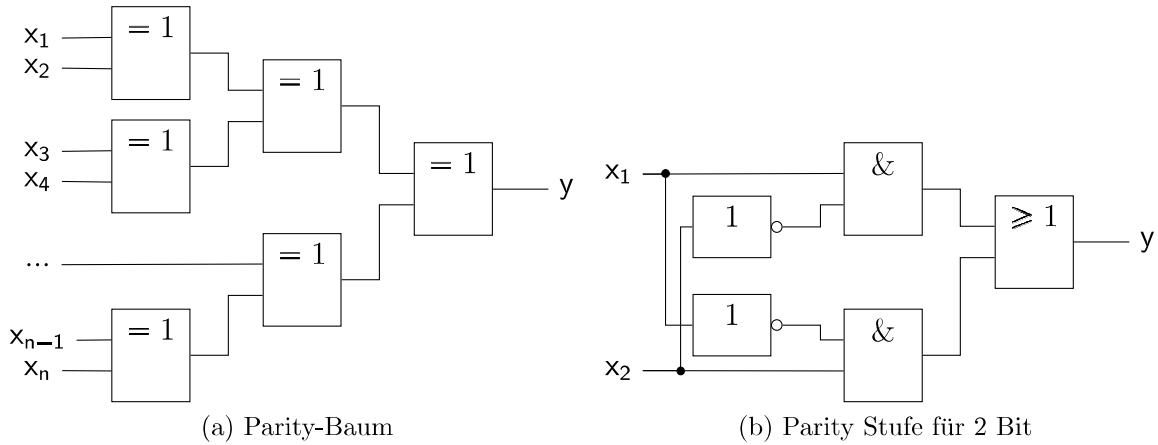


Abbildung 4.9: Parity-Baum

4.5.2 Faktorisierung

Die Umformung in Schaltungen mit mehr als zwei Stufen kann durch Faktorenzerlegung erfolgen.

Beispiel 64. Zerlegung in Faktoren

$$f = x_1 \cdot x_2 \cdot x_3 \cdot \overline{x_4} + x_1 \cdot x_2 \cdot \overline{x_3} \cdot x_4$$

Die Funktion kann wegen des gemeinsamen Faktors $x_1 \cdot x_2$ zerlegt werden in:

$$f = x_1 \cdot x_2 \cdot (x_3 \cdot \overline{x_4} + \overline{x_3} \cdot x_4)$$

Die Stufenzahl erhöht sich dabei von 2 auf 3, die Zahl der Eingänge pro Gatter reduziert sich von 4 auf 2. Es ergibt sich jedoch das Problem der unterschiedlichen Pfadlängen.

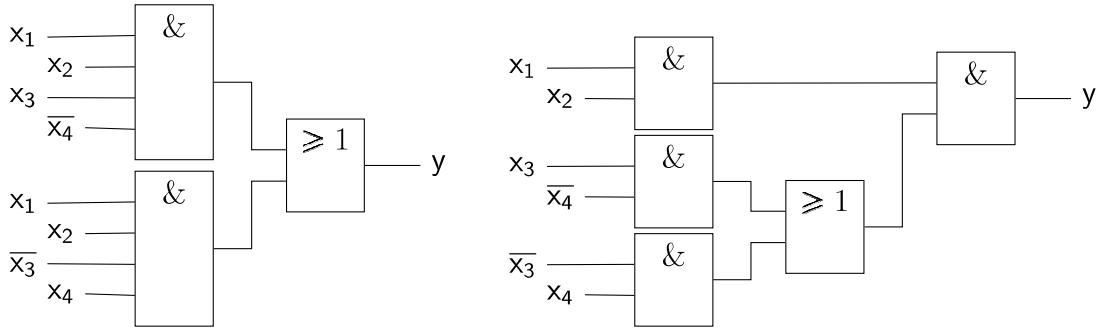


Abbildung 4.10: zweistufige Realisierung und dreistufige Realisierung mit zerlegten Faktoren

4.5.3 Zusammenfassung

Jedes Schaltnetz kann in eine äquivalente Schaltung umgeformt werden, um die Randbedingungen bezüglich der Eingangs- und Ausgangsfaktoren (fanin, fanout) zu erfüllen.

Beispiel 65. n -Eingangs-AND-Gatter

Ein AND-Gatter mit n Eingängen kann durch einen Baum von 2-Eingang-AND-Gatter ersetzt werden. Die Anzahl der Stufen beträgt $\log_2 n$ für $n = 2^N$, $N \in \mathbb{N}$ bzw. $\log_2(n + 1)$ für $n \neq 2^N$.

Ein Ausgangslastfaktor von n kann ersetzt werden durch eine Baumstruktur mit $\log_2 n$ Stufen, wobei jedes Gatter einen Eingang und ein Fanout von 2 aufweist (Verstärkerstufe). Bisher existiert allerdings kein allgemeiner Algorithmus zur effizienten Berechnung optimaler Lösungen von mehrstufigen Schaltnetzen.

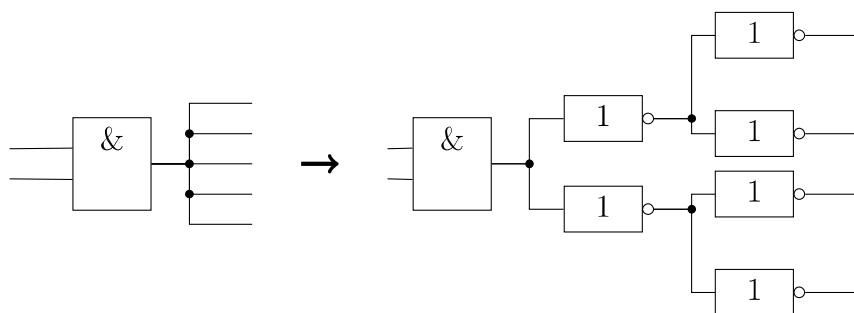


Abbildung 4.11: Reduzierung der Ausgangslast durch Baumstruktur