„Konzeption und Umsetzung eines Java-Applets zur Logikminimierung mit KV-Diagrammen“

Studienarbeit

am

Fachbereich Informatik

der

Universität Hamburg

vorgelegt von

Matthias Meyer

1998

Zusammenfassung

Die vorliegende Arbeit befaßt sich mit der Visualisierung der Logikminimierung von Funktionen durch Karnaugh-Veitch-Diagramme („KV-Diagramme“), um ein besseres Verständnis für den Prozeß der graphischen Minimierung zu ermöglichen. Aufbauend auf den theoretischen Grundlagen wurde ein Java-Applet für KV-Diagramme („KVD-Applet“) entwickelt. Das Applet stellt beliebige Funktionen mit bis zu sechs Variablen in Form eines KV-Diagrammes dar, das interaktiv durch den Benutzer bearbeitet werden kann. Eine Minimierung dieser Schaltungsfunktion wird in disjunktiver und konjunktiver Normalform unterstützt. Um den Vorgang der Minimierung didaktisch anschaulicher zu gestalten, wurde ein zweistufiges Verfahren implementiert. Ebenfalls wurde die Bündelminimierung mehrerer Funktionen gleichzeitig in der disjunktiven Normalform ermöglicht. Ein Exportieren der Schaltungsfunktion in textueller Form im PLA-Format [Espresso] wird unterstützt.

Abstract

This paper deals with the visualization of Boolean functions minimization with Karnaugh-Veitch diagrams („KV diagrams“). It presents a Java applet that was developed to ease the understanding of graphical minimization.. The applet is able to represent any function with up to six inputs in a KV diagram which can be modified interactively by the user. The minimization of the switching function is possible in both disjunctive and conjunctive normal form. Also, the applet supports minimization of multiple-output functions with up to eight outputs. A two-step visualization further helps to understand the graphical minimization process. Export of the switching function is possible in PLA format [Espresso] .

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Motivation

Zur Begleitung der Grundvorlesung zur technischen Informatik („B-Zyklus“) enstand dieses Applet zur Visualisierung der Logikminimierung mit Karnaugh-Veitch-Diagrammen. Es richtet sich hauptsächlich an Informatik-Studenten, die sich im Grundstudium befinden. Aber auch Fortgeschrittenen soll es als Werkzeug im HADES-Projekt der Universität Hamburg dienen. Mit ihm sollen die Studenten ein besseres Verständnis für die Methode der graphischen Logikminimierung bekommen. Dazu soll das Programm von Studenten als interaktives Lernmittel eingesetzt werden. In den entsprechenden Übungen können sie dann direkt den Zusammenhang zwischen Funktion, KV-Diagramm und der Schaltungsskizze erforschen und ein verbessertes Verständnis für den Prozeß des Schaltnetzentwurfs bekommen. Durch die verbesserte Visualisierung und Interaktion soll den Studenten insbesondere geholfen werden, zu verstehen, wie die graphische Zusammenfassung die Kosten einer Hardwarerealisierung minimieren kann.

2 Aufgabenstellung

Der Titel der Arbeit ist: „Konzeption und Umsetzung eines Java-Applets zur Logikminimierung mit KV-Diagrammen“. Die Aufgabe bestand darin, ein bereits bestehendes Applet [Hendrich 96] um folgende Funktionen zu erweitern:

- Eine korrekte, einheitliche Darstellung von beliebigen KV-Diagrammen verschiedener Größe. Die maximale Größe eines KV-Diagrammes soll bei 6 Eingangsvariablen liegen.
- Zur Veranschaulichung von Bündelminimierung sollen bis zu 8 KV-Diagramme gleichzeitig darstellbar und veränderbar sein.
- Die Funktion soll sowohl in disjunktiver als auch in konjunktiver Normalform als Schaltungsskizze anzeigt werden können. Eine Funktionsminimierung soll in beiden Darstellungen möglich sein.

- Der Benutzer soll interaktiv im KV-Diagramm die einzelnen Schleifen ziehen können. In der Schaltungsskizze soll gleichzeitig verdeutlicht werden, welche Gatter durch eine Schleife zusammengefaßt werden.
- Die Logikminimierung soll durch einen zusätzlichen Modus besonders anschaulich sein, Terme werden dazu erst auf Anforderung zusammengefaßt.
- Die Schaltungsfunktion soll in textueller Form exportierbar sein. Dabei wird das Berkley PLA-Format [Espresso] verwendet.
- Das Applet soll außerdem in den interaktiven Schaltungssimulator HADES der Universität Hamburg eingebunden werden [Hendrich 97].

3 Aufbau dieser Arbeit

Um einen schnellen Einstieg zu ermöglichen, wird im Kapitel 4: „Vorstellung der Anwendung" als erstes die Bedienung des Applets beschrieben. Nachdem die Benutzeroberfläche und die Funktionalität dargestellt wurde, wird an einem Beispiel ausführlich die Bedienung erklärt.
Anschließend beschreibt das Kapitel 5 die theoretischen Grundlagen von KV-Diagrammen. Dazu werden alle wesentlichen Begriffe für KV-Diagramme erläutert, sowie die Zusammenhänge der einzelnen Normalformen. In einem eigenen Abschnitt wird auf die grundlegende Theorie der Entwicklung von KV-Diagrammen, sowie die Minimierung mit ihnen eingegangen. Das Kapitel 6 beschreibt das Grundkonzept dieser Arbeit. In diesem Kapitel wird auf die zentralen Programmobjekte und ihrer Beziehung zueinander eingegangen. Zu jeder Klasse findet eine Aufzählung der Elemente und Methoden der Basisklassen mit kurzen Kommentaren statt. Das Kapitel endet mit einer Beschreibung des Algorithmus der Nachbarschaftserkennung mehrerer Terme im KV-Diagramm. In Kapitel 7 wird auf einige Details des KVD-Applets näher eingegangen. Es beginnt mit Darstellung der konjunktiven Normalform, danach wird der Student-Mode vorgestellt. Schließlich wird der Algorithmus für die Bündel-minimierung erläutert. Im Kapitel 11 findet eine Aufzählung der Resultate und eine rückblickende Bewertung statt. Die Literaturliste bildet das Kapitel 12.

4 Vorstellung des KVD-Applets

Das in dieser Arbeit erstellte KV-Diagramm-Applet verwendet eine graphische Benutzeroberfläche, die in zwei Fenster aufgeteilt ist. Das Hauptfenster beinhaltet die eigentliche Oberfläche für die Interaktion mit dem Benutzer und das KV-Diagramm. Das zweite Fenster beinhaltet die graphische Anzeige der Gatterschaltung.

Das KVD-Applet ermöglicht eine graphische interaktive Minimierung von Schaltfunktionen. Die Eingabe der Funktion erfolgt durch die direkte Modifikation eines bestehenden KV-Diagrammes. Hierzu stehen mehrere Beispiele, in verschiedenen Größen, sowie ein Eingabe-Dialog zur Verfügung, mit dem beliebige Schaltfunktionen mit bis zu 6 Eingängen und 8 Ausgängen realisiert werden können

Die Anwendung verhält sich je nach eingestelltem Modi anders; diese Hauptmodi schließen sich gegenseitig aus. Z.B. ist es nicht möglich, die Funktion im KV-Diagramm zu ändern, und sich parallel dazu die Feldindices anzeigen zu lassen.

Ein KV-Diagramm kann zunächst mittels des „*Edit Function*“ -Mode verändert werden. Dabei ist die Funktion ist in disjunktiver Normalform „*Add Loop 1 DNF*“-Mode oder in konjunktiver Normalform „*Add Loop 0 KNF*“-Mode als Schaltungs-Skizze darstellbar. Das Zusammenfassen und somit die Minimierung einzelner Terme erfolgt interaktiv durch eine Maus/Tasten-Steuerung. Diese Minimierung wird durch den „*Student Mode*“ besonders ausführlich veranschaulicht. Die Schaltungsfunktion kann zusätzlich mit dem Menüpunkt „*PLA Text*“ auch als Text, in Form einer PLA-Tabelle angezeigt und exportiert werden.

4.1 Das KV-Diagramm-Applet

Abbildung 1 zeigt das Hauptfenster des KVD-Applets. In diesem wird die Funktion in einem KV-Diagramm dargestellt. Es enthält die gesamte Oberfläche für die Interaktion mit dem Benutzer.

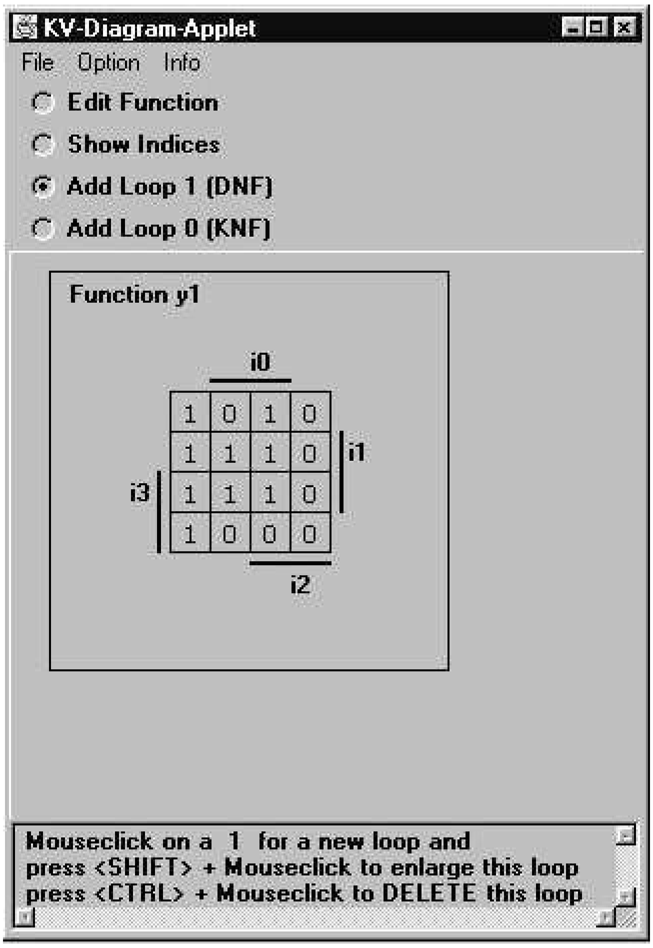


Abbildung 1: Das Hauptfenster

Es gibt vier verschiedene Modi, die sich gegenseitig ausschließen. Mit ihnen kann der Benutzter auf einfache Weise die Basiseinstellung verändern, und er kann immer erkennen, in welchen Modus er sich befindet. Ein Wechsel des Modus hat die Löschung aller bestehenden Schleife zur Folge. Die Ausnahme ist der „`show indices`"-Modus.

Die Hauptmodi:

- **Edit Function:**
  Die im KV-Diagramm dargestellte Funktion ist in diesem Modus direkt mit der Maus editierbar. Durch einfaches Anklicken eines Terms wird aus einer 1 eine 0, aus einer 0 ein * (don´t care) und aus einem * wieder eine 1. Durch interaktives Einstellen aller Terme kann daher jede beliebige boolsche Funktion ausgewählt werden.
- **Show Indices:**
  Die im KV-Diagramm angezeigte Funktion wird durch die Anzeige der Indizes des KV-Diagramms ersetzt, siehe Abb. 2. Zum Beispiel entspricht der Index 4 der Variablenbelegung (i_3=0, i_2=1, i_1=0, i_0=0). Bestehende Schleifen bleiben erhalten.

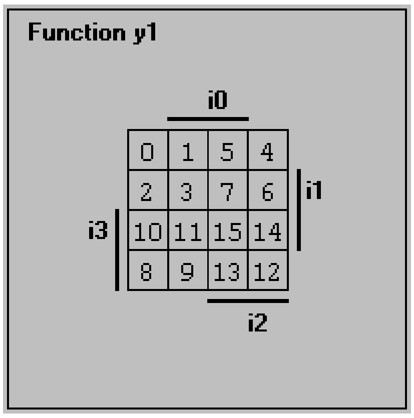


Abbildung 2: Darstellung eines KV-Diagrammes mit 4 Variablen

- **Add Loop 1 (DNF):**
  Das KV-Diagramm wird nun als DNF (disjunktive Normalform) in der Schaltungsskizze dargestellt. Im KV-Diagramm sind in diesem Modus nur die Minterme, d.h. die Einsen, durch Schleifen zusammenfaßbar.

- **Add Loop 0 (KNF):**
  Das KVD wird in der KNF (konjunktive Normalform) dargestellt und es sind somit die Maxterme, die Nullen, die zusammengefaßt werden können.

Über die Menüleiste des Applets sind zusätzlich folgende Funktionen erreichbar:

- **Menü: File: Examples**:

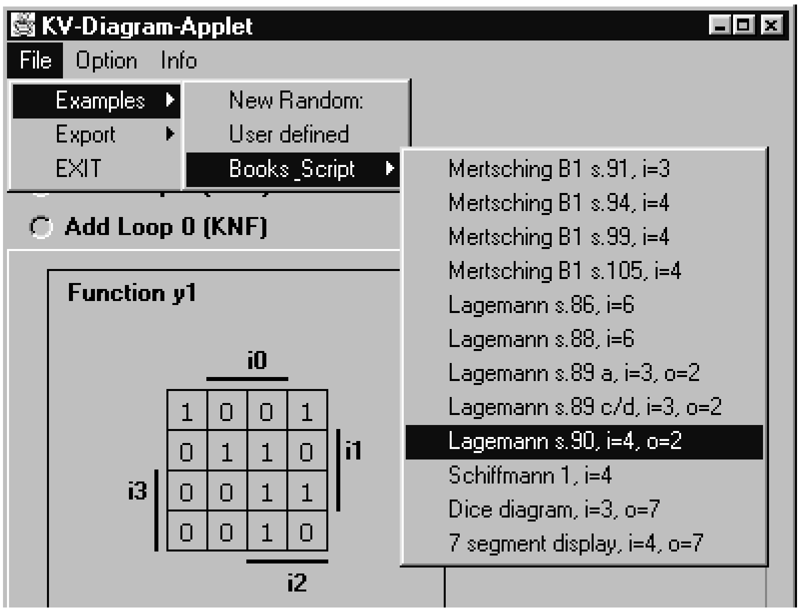


Abbildung 3: Das hierarchische Beispiel-Menü.
Für jedes Beispiel ist zusätzlich die Anzahl der Eingänge und bei Bedarf
die Anzahl der Ausgänge angegeben.

Über dieses Menü können hierarchisch Untermenüs ausgewählt werden. Neben vordefinierten Funktionen aus den Vorlagen von [Mertsching 94] und [Lagemann 87] können auch zufällige Funktionen („new random“) mit 2 bis 6 Eingängen oder benutzerdefinierte Funktionen ausgewählt werden. Für letztere Auswahl dient der Eingabedialog, s. Abb. 4:
Die Funktion wird zunächst mit der gewünschten Größe und einer zufälligen Belegung mit 50% Einsen erzeugt und kann danach vom Benutzer modifiziert werden.

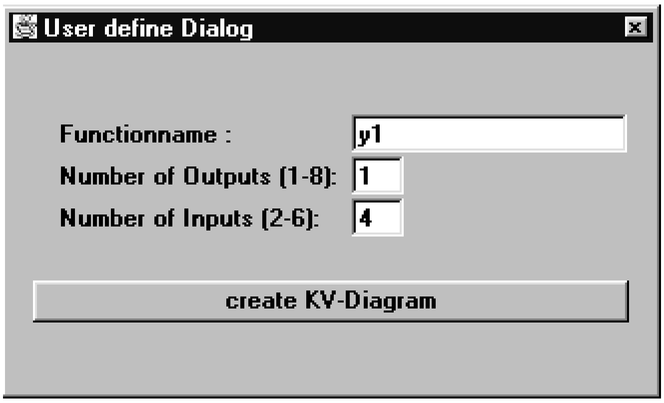


Abbildung 4: Der Dialog zur Eingabe einer neuen Funktion.

- **Menü: Export: PLA TEXT:**

Dieser Befehl erzeugt eine textuelle Ausgabe der Schaltfunktion im PLA-Format [Espresso]. Dieses Format kann mittels Clipboard übernommen werden. Ein direktes Speichern ist durch die Applet-Beschränkungen aus Sicherheitsgründen nicht möglich.

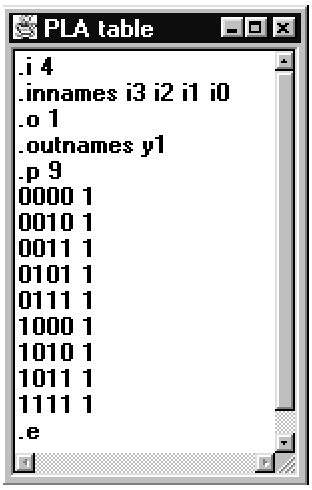


Abbildung 5: Ausgabe einer Funktion im PLA-Format [Espresso].

- **Menü: Option**
  - Beginner Mode:
    Das zweistufige Verfahren zur Funktionsminimierung wird aktiviert/deaktiviert, siehe Abschnitt 7.3.
  - Zoom 50%:
    Die Gatter in der Schaltungs-Skizze werden auf 50% verkleinert, um auch größere Funktionen auf dem Bildschirm darstellen zu können.

Im unterem Teil des Hauptfensters befindet sich ein Textfeld. Hier werden abhängig von Modus verschiedene Hinweise für die Bedienung des Applets eingeblendet.

4.2 Die Schaltungs-Skizze

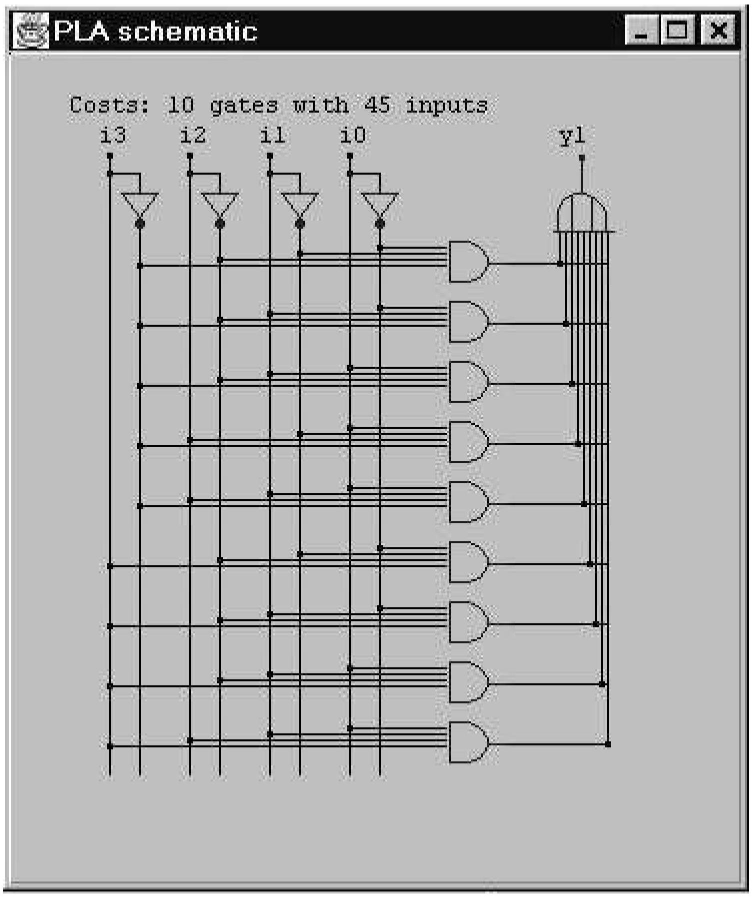


Abbildung 6: Funktion aus Abb.5 in disjunktiver (UND – ODER) Normalform.

Ein Schaltnetz ist die Realisierung einer Schaltfunktion. In disjunktiver oder konjunktiver Normalform wird eine gegebene Schaltfunktion als zweistufiges Schaltnetz verwirklicht. Die Inverter sorgen für die Negierung der Eingangsvariablen. Wie üblich werden die Inverter nicht als eigene Stufe angesehen.

In der disjunktiven Normalform besteht die erste Stufe aus UND-Verknüpfungen zur Bildung der Minterme aus den negierten oder nicht negierten Eingangsvariablen. Die zweite und letzte Stufe besteht aus einer ODER-Verknüpfung zur disjunktiven Verknüpfung aller einzelnen Minterme.

In der folgenden Abbildung 6 ist eine Schaltfunktion in disjunktiver Normalform dargestellt. Dies ist die Ausgangssituation, in der für alle Minterme ein UND-Gatter dargestellt wird, und diese UND-Gatter in der zweiten Stufe an ein ODER-Gatter angeschlossen werden. Diese graphische Ausgabe ist die der PLA (Programmable Logic

Array) Architektur. In der ersten Zeile des Schaltungsfensters sind zusätzlich die Kosten der Schaltung angegeben, d.h. die Anzahl der Gatter und der Gatter-Eingänge ohne die Inverter.

4.3 Die Schleifen

Der zentrale Punkt bei der Minimierung mit KV-Diagrammen ist die menschliche Fähigkeit, zusammenhängende Formen zu erkennen. Bei den KV-Diagrammen sind es die benachbarten Terme. Diese lassen sich mit Hilfe von Schleifen zu einem Term zusammenfassen. Im Applet wird dazu zunächst ein Term durch Anklicken mit der Maus direkt im KV-Diagramm selektiert. Man erhält eine neue Schleife, die genau diesen Term abdeckt. Zur aktuellen Schleife wird ein weiterer Term hinzugeführt, indem die `SHIFT`-Taste festgehalten und die Maustaste gedrückt wird. Soll eine bestehende Schleife gelöscht werden, so muß die `CTRL`-Taste gehalten und mit der Maus in die zu löschende Schleife geklickt werden.

Die Möglichkeiten der Bedienung auf einen Blick:

- **eine neue Schleife anlegen:** einfacher Mausklick
- **eine Schleife erweitern:** SHIFT-Taste halten und durch einen einfachen Mausklick die aktuelle Schleife erweitern. Dabei können natürlich nur gültige Terme (1 oder * in DNF, oder 0 und * in KNF) ausgewählt werden.
- **eine Schleife löschen:** CRTL-Taste halten und einen Mausklick in die entsprechende Schleife.

4.4 Ein einfaches Beispiel für die Benutzung des KV-Diagramm Applets

Als erster Schritt wird vom Benutzer ein geeignet großes Beispiel ausgewählt, siehe Abb.7. Die Beispiele sind über das entsprechende Menü erreichbar. Ein Dialog zur Spezifikation von KV-Diagrammen steht ebenfalls zur Verfügung.

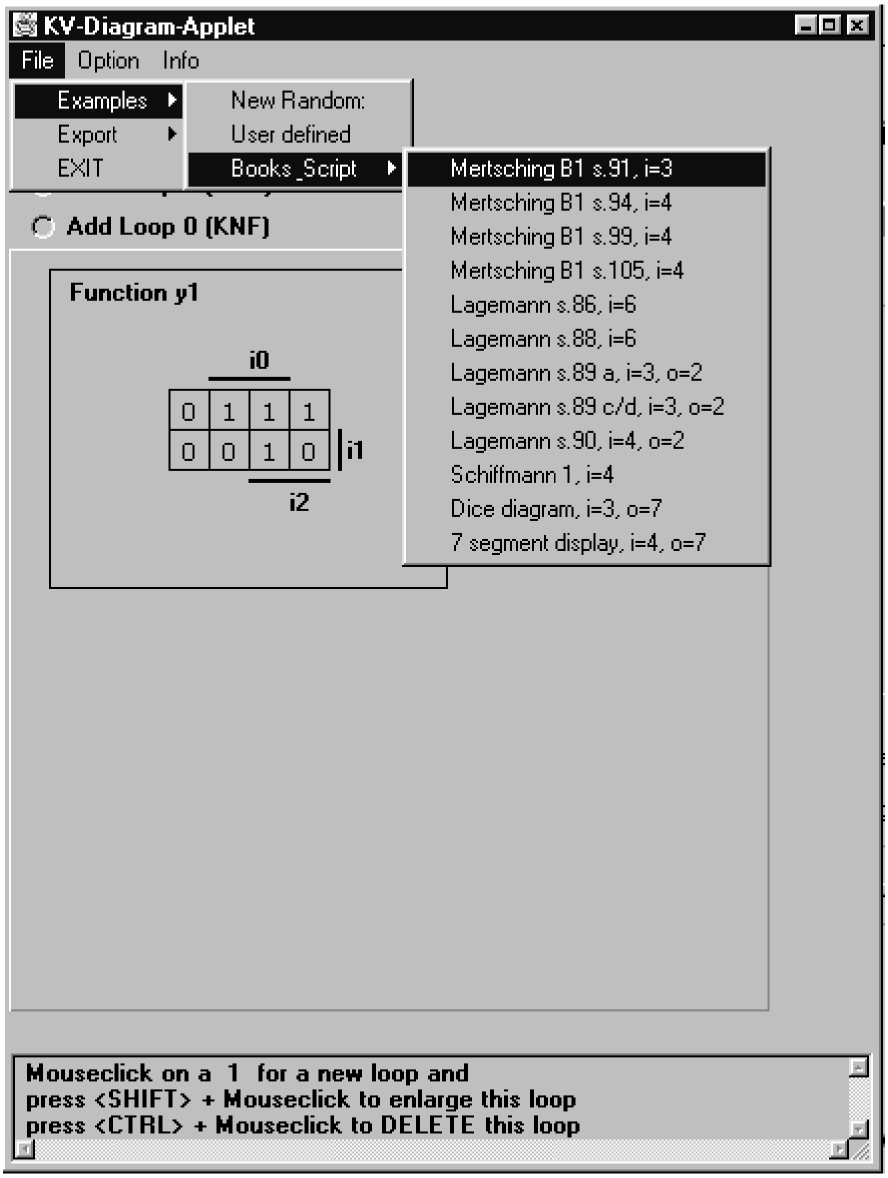


Abbildung 7: Ein Beispiel wird ausgewählt.

Jetzt kann das ausgewählte KV-Diagramm mittels des „**Edit Function**“-Modus solange manipuliert werden, bis die gewünschte Funktion dargestellt wird. Dieser Punkt wird in diesem Beispiel nicht dargestellt. Als nächstes wird entschieden, ob die Minimierung in DNF oder KNF vorgenommen werden soll. Als Standard ist der disjunktive Modus eingestellt.

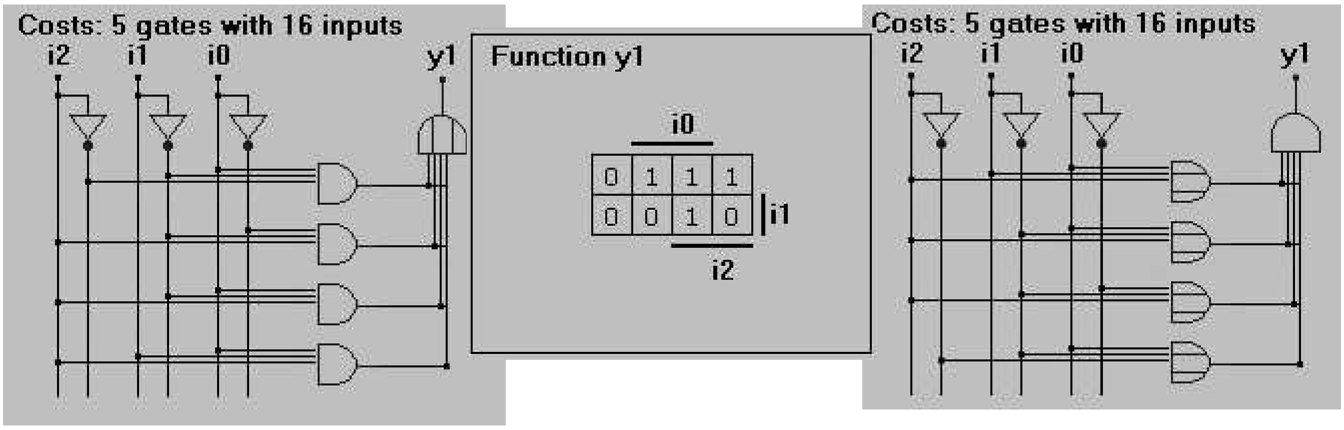


Abbildung 8: Eine Funktion in DNF (links) und KNF (rechts).

Nachdem die Funktion eingegeben und der richtige Modus eingestellt wurde, kann jetzt die erste Schleife mittels Mausklick im KV-Diagamm gezogen werden. Das entsprechende Gatter wird in der Schaltungsskizze in der Schleifenfarbe hervorgehoben:

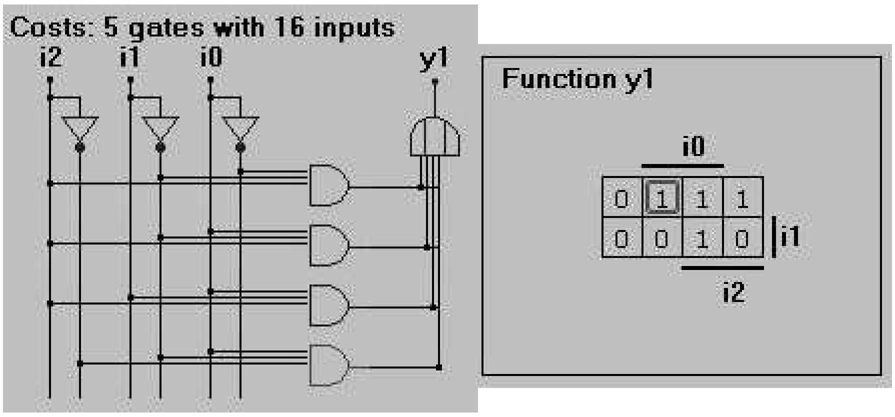


Abbildung 9: Die erste Schleife wurde angefangen.

Eine bestehende Schleife kann durch Halten der SHIFT-Taste und Mausklick um korrekte Terme erweitert werden. In der Schaltungsskizze wird dann sofort die entsprechende Minimierung vollzogen, siehe Abb.10.

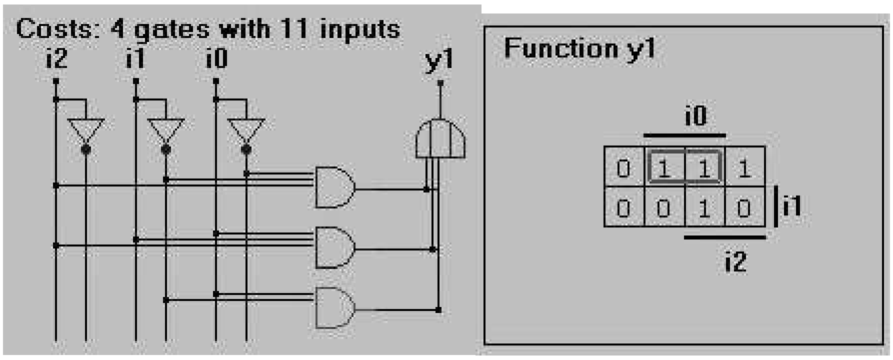


Abbildung 10: Die Schleife wurde um einen Term erweitert.

Durch die Wiederholung der vorigen beiden Schritte erhält man ein Ergebnis der Minimierung. Zu jeder Zeit ist es zusätzlich möglich, sich die Funktion als Text ausgeben zu lassen, siehe Abb. 11.

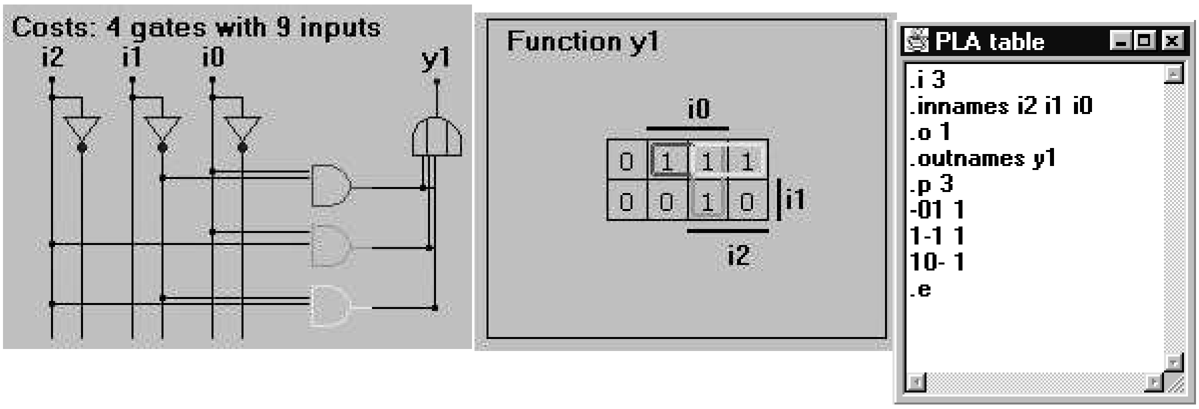


Abbildung 11: Das Ergebnis der Minimierung.

5 Theoretische Grundlagen

5.1 *Übersicht über die theoretischen Grundlagen*

Dieses Kapitel befaßt sich mit den theoretischen Grundlagen der Minimierung mit KV-Diagrammen. Nach einer kurzen Erläuterung, was ein KV-Diagramm ist, werden die wichtigsten Begriffe aufgelistet und erklärt. In dem folgenden Abschnitt werden die Zusammenhänge von Funktionstabelle und dem entsprechenden KV-Diagramm aufgezeigt. KV-Diagramme sind eine Möglichkeit um Funktionen zu minimieren, dieses wird in dem Abschnitt 5.5 in diesem Kapitel behandelt. Nachdem die Begriffe DNF und KNF erläutert wurden, folgt eine Aufzählung der Darstellungsformen boolescher Funktionen und den Umwandlungen zwischen diesen Darstellungsformen.

5.2 *Was ist ein KV-Diagramm ?*

Ein Karnaugh-Veitch-Diagramm ist die graphische Darstellung einer Wertetabelle oder Schaltfunktion. KV-Diagramme sind ein Hilfsmittel beim Vereinfachen (Minimieren) von Schaltfunktionen und bei der Übertragung einer Funktionstabelle in eine Schaltungsskizze. Das Diagramm ist in Matrixform angeordnet. Jeder Kombination der Eingangsvariablen wird ein Feld der Matrix zugeordnet. Hat die Wertetabelle oder Schaltfunktion n Eingangsvariable, dann hat das zugehörige KV-Diagramm 2^n Felder. Die Anordnung oder Zuordnung der Felder zu den 2^n möglichen Kombinationen der Eingangsvariablen wird so vorgenommen, daß sich beim Übergang von einem Feld zu einem benachbarten Feld nur der Wert *einer* Variablen ändert. Anfangsfeld und Endfeld einer Zeile oder Spalte sind benachbart.

5.3 *Grundlegende Begriffe für KV-Diagramme*

- Das Dualitätsprinzip des Boolschen Verbandes [Schiffmann-Schmitz 92],
  jedes Axiom und jedes Gesetz geht durch wechselseitigen Tausch der UND-Verknüpfung und ODER-Verknüpfung sowie die von Nullelement und Einselement in ein ebenfalls gültiges duales Gesetz über.
- benachbart,
  zwei Punkte heißen benachbart, g.d.w. sie in einer Koordinate differieren.

- Minterme,
  ein Minterm ist die Konjunktion aller Eingangsvariablen, von denen jede die normale Form oder die negierte Form annehmen kann. In der Anfangs-Schaltungskizze ist jeder 1-Term im KV-Diagramm (ein Punkt im KV-Diagramm der den Funktionswert 1 aufweist) durch ein Minterm-Gatter realisiert. Die Minterme realisieren in disjunktiver Normalform die Ausgangssituation einer Minimierung. Sie bestehen aus einem UND-Glied mit n Eingängen. Dabei ist n ist die Anzahl der Eingangsvariablen.
- Maxterme,
  durch Anwendung des Dualitätsprinzips lassen sich aus den Mintermen die Maxterme gewinnen. Die Maxterme bestehen in der Anfangs-Schaltungsskizze aus einem ODER-Glied mit n Eingängen. Sie repräsentieren die 0-Punkt-Menge der Funktion und bilden die konjunktive Normalform.

5.4 Wie wird aus einer Funktionstabelle ein KV-Diagramm ?

Das nachfolgende Beispiel soll den Weg von einer Funktionstabelle in ein KV-Diagramm veranschaulichen.

Die Tabelle ordnet jedem Index einen binären Vektor der Eingangsvariablen zu, eine 0 steht für die Negation des Einganges. In der letzen Spalte wird der Funktionswert eingetragen.

Index	i_2	i_1	i_0	**f(x)**
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Abbildung 12: Funktionstabelle mit 3 Varibalen

Im nächsten Schritt wird ein KV-Diagramm mit 3 Eingangsvariablen erzeugt. Die Indizierung ergibt sich aus den angegebenen Bereichen, in denen die Variablen den Wert 1 oder 0 haben. Beispiel Index 3: er entspricht umgangssprachlich: „Die Eingänge

i_0 und i_1 sind normal angeschlossen und i_2 wird negiert". Der letzte Schritt ist das Eintragen der Funktionswerte.

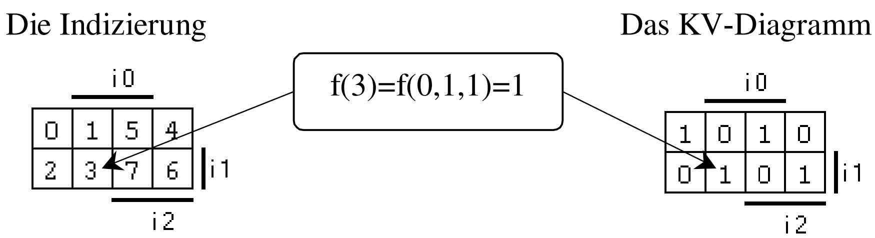


Abbildung 13: Darstellung der Beziehung zwischen Funktionstabelle, Variablenbelegung und KV-Diagramm.

5.5 *Wie minimiert man eine Schaltfunktion mit KV-Diagrammen ?*

Schaltfunktionen, die in einer Normalform angegeben sind, oder die aus einer Funktionstabelle hergeleitet wurden, sind oft redundant. Sie lassen sich dann vereinfachen. Vereinfachung soll hier bedeuten: Funktionsgleichungen werden in möglichst „kurzen" Ausdrücken dargestellt, das heißt mit möglichst wenig Variablen und möglichst wenig Verknüpfungen.

Das Minimieren mit den KV-Diagrammen ist ein graphisches Verfahren. Ist in dem KV-Diagramm eine Schaltfunktion in DNF dargestellt, dann werden benachbarte Felder, die mit 1 belegt sind, in einer Schleife zusammengefaßt.

Die Zusammenfassung wird so vorgenommen, daß jede Schleife möglichst viele Einsen einschließt. Dabei dürfen aber nur Blöcke mit zwei, vier, acht usw. Feldern zusammengefaßt werden. Felder in der ersten und letzten Zeile, aber in der gleicher Spalte, sind benachbart und können in einer Schleife zusammengefaßt werden. Ebenso Felder in der ersten und letzten Spalte, aber in der gleichen Zeile. Die vereinfachte Schaltfunktion wird dann aus den Termen der zusammengefaßten Schleifen und der übriggebliebenen Einzelfelder gebildet. Die Terme der Schleifen enthalten also nur die Variablen, die sich innerhalb einer Schleife nicht ändern.

Die Minimierung basiert auf der folgenden Boolschen Gleichung:

$$a \wedge b \vee a \wedge \neg b = a \wedge (b \vee \neg b) = a$$

Wendet man diese Gleichung auf die KV-Diagramme an, hat sie folgende Bedeutung: „Zwei benachbarte 1-Terme unterscheiden sich genau in einer Eingangsvariable, die durch die ODER-Verknüpfung dieser Terme überflüssig wird“
Die Vereinfachung von Schaltfunktionen mit KV-Diagrammen eignet leider nur sich für Schaltfunktionen mit maximal 6 Schaltvariablen. Sind mehr als 6 Schaltvariablen in der Schaltfunktion enthalten, dann wird dieses Verfahren unübersichtlich.

Die disjunktive Normalform einer Schaltfunktion mit genau zwei benachbarten 1-Punkten und *n* Variablen läßt sich vereinfachen zu einer Konjunktion von *n-1* Variablen, wobei gerade die Unterschiedskoordinate fehlt.

5.6 Was bedeutet DNF und KNF ?

Die disjunktive Normalform einer Funktion erhält man folgendermaßen: Jeder Klammerausdruck wird gebildet aus den Argumenten x, y, z einer Zeile der Wertetabelle, an der Funktionswert der Funktion gleich 1 ist. Diese 1-Punkt-Menge definiert die Schaltfunktion. Die Argumente werden mit der UND-Funktion verknüpft, wobei jedes Argument, das den Wert 0 besitzt, negiert wird. Verknüpft man die einzelnen Klammerausdrücke mit der ODER-Funktion, erhält man die disjunktive Normalform der Funktion. Z.B. für die Funktion aus Abb. 12 lautet die DNF:

$$(\neg i_2 \wedge \neg i_1 \wedge \neg i_0) \vee (\neg i_2 \wedge i_1 \wedge i_0) \wedge (i_2 \wedge \neg i_1 \wedge i_0) \wedge (i_2 \wedge i_1 \wedge \neg i_0)$$

Die konjunktive Normalform einer Funktion erhält man folgendermaßen: Jeder Klammerausdruck wird gebildet aus den Argumenten x, y, z einer Zeile der Wertetabelle, an der Funktionswert der Funktion gleich 0 ist. Dir Argumente werden mit der ODER-Funktion verknüpft, wobei jedes Argument, das den Wert 1 besitzt, negiert wird. Verknüpft man die einzelnen Klammerausdrücke mit der UND-Funktion, erhält man die konjunktive Normalform der Funktion.

$$(i_2 \vee i_1 \vee \neg i_0) \wedge (i_2 \vee \neg i_1 \vee i_0) \wedge (\neg i_2 \vee i_1 \vee i_0) \wedge (\neg i_2 \vee \neg i_1 \vee \neg i_0)$$

5.7 Normalformen

Normalformen beschreiben eine Schaltfunktion ausgehend von einer Wertetabelle in Gleichungsform. Sie sind eine Bezeichnung für eine standardisierte Darstellung von boolschen Funktionen durch boolsche Ausdrücke. Für eine gegebene Wertetabelle läßt sich mit Hilfe der Minterme oder Maxterme die Schaltfunktion in normierter Form darstellen.

- Minterme,
  Minterme sind UND-Verknüpfungen, die alle Schaltvariablen einmal enthalten, wobei diese negiert oder nicht negiert vorkommen können.
- Maxterme,
  Maxterme sind ODER-Verknüpfungen, die alle Schaltvariablen einmal enthalten, wobei diese negiert oder nicht negiert vorkommen können.
- don´t-care - Terme,
  Bei Schaltfunktionen in Normalform ist für alle möglichen Kombinationen der Eingangsvariablen ein Wert (**0** oder **1**) der Ausgangsvariablen definiert. Es gibt Wertetabellen, bei denen nicht für jede Kombination der Eingangsvariablen ein Wert der Ausgangsvariablen definiert ist. Für diese Kombinationen darf der Wert der Ausgangsvariablen mit **0** oder **1** belegt werden. Diese Kombinationen werden don´t-care - Terme genannt, und mit einem „*“ gekennzeichnet.

Eine Schaltfunktion in disjunktiver Normalform (DNF) besteht aus der disjunktiver Verknüpfung aller Minterme:

$$f(x_1, x_2, x_3) = (x_1 \wedge x_2 \wedge x_3) \vee (\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (\neg x_1 \wedge \neg x_2 \wedge x_3) \vee \ldots$$

Die konjunktive Normalform (KNF) besteht aus der konjuntiven Verknüpfung aller Maxterme:

$$f(x_1, x_2, x_3) = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \vee \ldots$$

Beide Darstellungen sind äquivalent und ineinander überführbar (Dualitätsprinzip).

5.8 Darstellungsformen Boolescher Funktionen und ihre Umwandlungen

Die folgende Aufzählung beschreibt das Vorgehen zur Umwandlung verschiedener Darstellungen boolescher Funktionen in eine andere Darstellung.

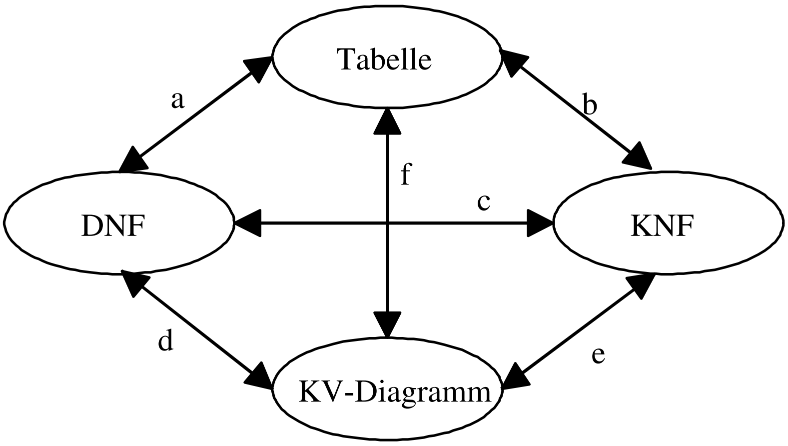


Abbildung 14: Einige Darstellungsformen für boolesche Funktionen.

a. **Tabelle → DNF:**
Alle Minterme der Tabelle entnehmen (Zeilen mit dem Funktionswert Eins) und ODER-verknüpfen.
DNF → Tabelle:
Alle Minterme der DNF entnehmen und mit dem Funktionswert Eins in der Tabelle notieren.

b. **Tabelle → KNF:**
Alle Maxterme der Tabelle entnehmen (Zeilen mit dem Funktionswert Null) und UND-verknüpfen.
KNF → Tabelle:
Alle Maxterme der KNF entnehmen und mit dem Funktionswert Null in der Tabelle notieren.

c. **DNF → KNF:**
Theorem von DeMorgan anwenden und mit komplementären Termen auf die KNF erweitern.
KNF → DNF:
Theorem von DeMorgan anwenden und mit komplementären Termen auf die DNF erweitern.

d. **DNF → KV-Diagramm:**
Größe des KV-Diagramms durch die Anzahl der Variablen festlegen und zeichnen, Minterme eintragen (Eins-Muster).
KV-Diagramm → DNF:
Jede Eins im KV-Diagramm entspricht einem Minterm der Funktion.

e. **KNF → KV-Diagramm:**
Größe des KV-Diagramms durch die Anzahl der Variablen festlegen und zeichnen, Maxterme eintragen (Null-Muster).
KV-Diagramm → KNF:
Jede Null im KV-Diagramm entspricht einen Maxterm der Funktion.

f. **Tabelle → KV-Diagramm:**
Jede Zeile in der Tabelle entspricht einem Feld im KV-Diagramm und umgekehrt. Eine schnelle Zuordnung kann über die Dezimaläquivalente der Variablenbelegungen erfolgen.

5.9 KV-Diagramme: Entwicklung und Variablenzuordnung

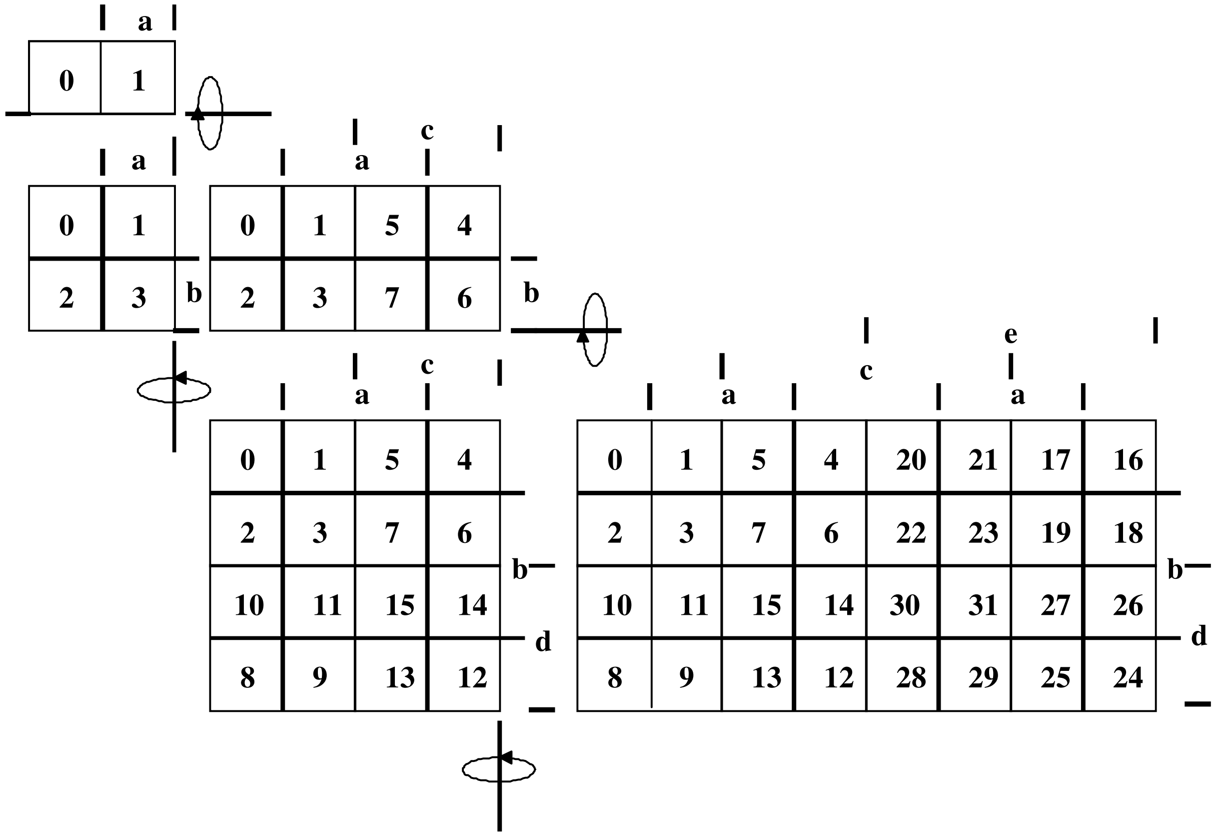


Abbildung 15: Darstellung von Funktionen mit 1 (links oben),
2 (links unten), 3 (mitte unten) und 5 (rechts unten) Eingangsvariablen in KV-Diagrammen.

In der Abb. 15 wird die Entwicklung von KV-Diagrammen in bezug auf die Vaiablenzuordnung gezeigt. Links oben befindet sich ein KV-Diagramm mit einer Variablen $a=x_0$, durch umklappen entsteht ein KV-Diagramm mit zwei Variablen $a=x_0$ und usw. In der Abb.2 ist ein KV-Diagramm mit 4 Variablen und eingezeichneten Indizes dargestellt. Die Variablenzuordung in der Abb. 15 entsprechend anzupassen $a=i_0$, $b=i_1$, $c=i_2$ und $d=i_3$.

6 Aufbau des KVD-Applets

6.1 Übersicht über das Grundkonzept

Grundlage dieser Arbeit war ein einfaches Applet [Hendrich 96], das allerdings weder KNF noch Bündelminimierung unterstützte. Die erste Aufgabe dieser Arbeit bestand darin, den bestehenden Programmcode und seine Objekt-Architektur zu verstehen, um ihn später zu erweitern. In diesem Kapitel wird auf die wichtigsten Programmobjekte und ihre Beziehung zu einander eingegangen. Dabei wird jede Klasse wird eigens vorgestellt, es werden ihre Elemente sowie ihre wichtigsten Methoden beschrieben. Dieses Kapitel schließt mit dem Algorithmus zur Erkennung von Nachbarschafts-beziehungen, um korrekte Schleifen ziehen zu können.

In dieser Arbeit wurden meistens vollständige Algorithmen benutzt, dieses stellt aber keine Einschränkung dar, denn die Datenmengen z.B. Anzahl der Gatter oder Schleifen sind wegen der Beschränkung auf maximal sechs Eingangsvariablen klein genug. Die Algorithmen sind für diese Zwecke auch ausreichend performant.

6.2 Welche Objekte?

Da es sich bei Java um eine rein objektorientierte Sprache handelt, ist die Darstellung der wesentlichen Objekte eine der ersten Aufgaben für die Beschreibung des Grundkonzeptes. Sie ergab sich aus der Aufgabenstellung und natürlich aus dem ersten Grunddesign, daß in einer späteren Überarbeitungsphase geändert wurde.

Die beiden zentralen Objekte sind:

- das KV-Diagramm, im Programm durch die Klasse KVD umgesetzt
- die Schaltungsskizze, sie wird technisch durch die Klasse PLA implementiert.

Diese beiden Hauptobjekte beinhalten die eigentliche Datenstrukturen und damit die logischen und technischen Details. Um ihre korrekte graphische Darstellung kümmern sich teilweise andere Klassen. Jedes der beiden Objekte verwaltet eine geschlossene doppelt verkette Liste von anderen Objekten. Die Klasse KVD verwaltet die Schleifen aus der Klasse LOOP. Die Klasse PLA beinhaltet eine Liste aller Schaltungsgatter aus der Klasse CUBE. Zu ihr gehören die Minterme sowie auch die Gatter nach einer Minimierung. Das nachfolgende Diagramm soll diesen Sachverhalt verdeutlichen.

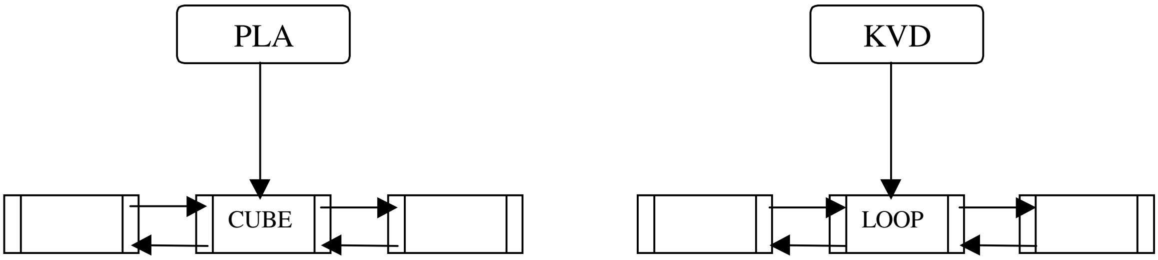


Abbildung 16: Grundgerüst des KVD-Applets.

Diese Objekte werden von einem Objekt der Klasse KVDdialog erzeugt, verwaltet und angezeigt. Diese Klasse stellt somit die Hauptklasse für die Benutzteroberfläche da. Sie erzeugt alle GUI-Elemente und verarbeitet auch die Benutzeraktionen. Dieses Konzept wurde in einer späteren Überarbeitung leicht verändert. Ich werde im folgenden die Klassen KVD und PLA näher beschreiben. Diese Beschreibung ist nicht vollständig; sie wird sich auf die wichtigsten Methoden und Felder konzentrieren.

6.2.1 Die Klasse KVD.

Die Datenelemente der Klasse KVD lassen sich in drei Gruppen einteilen:

- Felder, die für ein KV-Diagramm wichtig sind. Diese sind die Anzahl der Eingänge `nvars`, die eigentliche Funktion `f[]`, sowie der Name der Funktion `fname` und die Namen der Eingänge `innames[]`.
- Die zweite Gruppe sind Elemente die für die graphische Darstellung des KV-Diagrammes benötigt werden. Es sind die Koordinaten der wichtigsten Eckpunkte (z.B. die linke obere Ecke `x0` und `y0`), die totale Größe des Diagrammes, die Ausmaße eines Feldes und deren Anzahl in x- bzw. y-Richtung.
- Ein Verweis auf die Liste der in dem KV-Diagramm gezogenen Schleifen.

Der Konstruktor der Klasse KVD hat die folgende Form:

```
KVD( int nvars, String inn[], String fname, int f[], int
x0, int y0 )
```

Um ein KVD-Objekt zu erzeugen, müssen also die Anzahl der Eingabevariablen, ein Array der Beschriftungen der Eingänge, die Funktion ebenfalls in der Form eines Arrays und die linke obere Ecke angeben werden. Die Liste der Schleifen wird mit einem Wurzelobjekt der Klasse Loop initialisiert, es wird hierzu der Default-Konstruktor der Klasse verwendet.

Die wichtigsten Methoden der Klasse KVD sind:

- Die Konvertierungsfunktionen IntToArray bzw. IntToArray_KNF.
  - `public int[] IntToArray( int val, int length, int[] A)`
  - `public int[] IntToArray_KNF( int val, int length, int[] A)`

  Diese beiden Methoden konvertieren einen Integer-Wert in ein Integer-Array gewünschter Länge. Die Länge ist z.B. die Anzahl der Eingabevariablen, wobei dem eine 1 für eine Verbindung zu dem entsprechenden Eingang steht.
- Die Gruppe der Methoden die zum Erzeugen, Verwalten und Löschen von Schleifen benötigt werden. Einige dieser Methoden sind überladen, um sie in verschieden "Situation" zu benutzen.
  - addLoop:
    - `public Loop addLoop( int indices[] )`
    - `public Loop addLoop( int indices[], Color c )`

    Diese Methoden fügen eine neue Schleife in die doppelt verkette Liste aller bestehenden Schleifen ein. Dies kann wahlweise mit Angabe der Farbe geschehen. Eine Überprüfung, ob es diese Schleife gibt, findet an dieser Stelle nicht statt.
  - deleteLoop:
    - `public void deleteLoop( Loop l )`
    - `public void deleteLoop( KVD kvd, int index)`

    Die erste deleteLoop Methode stellt die Basisfunktionalität zur Verfügung. Sie löscht die übergebene Schleife aus der doppelt verketten Liste durch Umbiegen der Zeiger und anschließendes Löschen des Objektes.

    Die zweite, erweiterte Methode iteriert über alle Schleifenobjekte des übergebenen KVD-Objektes. Für jede gefundene Schleife wird geprüft, ob der entsprechende Index innerhalb der Schleife vorhanden ist. Fällt dieser Test positiv aus, so wird die Schleife gelöscht.

- deleteAllLoops,

  durchläuft die komplette Liste der vorhandenen Schleifen und ruft für jedes Schleifenobjekt die `deleteLoop` Methode auf, so daß alle Schleifen gelöscht werden.
- getRootLoop und nextLoop sind die beiden Hauptmethoden für die Iteration durch die Liste der Schleifen. Die Methode `getRootLoop` gibt das Wurzelobjekt zurück und `nextLoop` die Schleife, auf die der next-Zeiger zeigt.
  - `public Loop getRootLoop()`
  - `public Loop nextLoop( Loop l )`
- addTermToLoop:
  - `public int addTermToLoop( Loop l, int index, int Mode )`
  - `public int addTermToLoop( Loop l, int index, int Mode, int o_index, int no )`

  Mit dieser Methode wird versucht, eine bestehende Schleife zu erweitern. Es wird als erstes abgeprüft, ob die Schleife mit dem neuen Feldindex korrekt bleibt. Dieses wird durch einen Nachbarschaftalgorithmus realisiert. Ist die Schleife abhängig vom Mode als richtig erkannt, so werden nun alle graphischen Eckdaten ermittelt und das Schleifenobjekt aktualisiert.

- Die Gruppe der Methoden für die graphische Darstellung:
  - showIndices,

    es wird ein interner Mode umgeschaltet, und anstatt die Funktion im KV-Diagramm anzuzeigen, werden die Indizes der einzelnen Felder angezeigt (siehe Abb. 2).
  - drawLabels,

    durch diese Methode werden die beiden Hilfsmethoden drawLabelHoriz und drawLabelVert mit den richtigen Parametern aufgerufen. Die korrekte Beschriftung für die einzelnen KV-Diagrammgrößen wird hier realisiert.
  - drawLoops,

    es werden alle Schleifen eines KV-Diagrammes gezeichnet. Die korrekte Darstellung wurde durch die Richtigkeit der Schleife mittels des Nachbarschaftstests vorher ermittelt.

- drawKV D,

  hier wird nun das komplette KV-Diagramm gezeichnet. Es beinhaltet die Überschriften, die Variablenmarkierungen, die zu repräsentierende Funktion und alle vorhandenen Schleifen. Von dieser Hauptmethode der graphischen Darstellung aus werden die oben beschriebenen Hilfsmethoden aus aufgerufen.

- Die folgenden Methoden gehören zu dem Nachbarschaftsalgorithmus, der im Abschnitt 6.3 noch ausführlich erklärt wird:
  - areNeighbors:
    - `private boolean areNeighbors( int i1, int i2)`

    Es wird abgeprüft, ob die beiden und die Indices der ausgewählten Terme direkte horizontale oder vertikale Nachbarn sind.
  - isRightNeighbor:
    - `private boolean isRightNeighbor( int src, int test )`

    Diese Methode prüft, ob der Testindex genau rechts vom Ursprungsindex liegt.
  - isBottomNeighbor:
    - `private boolean isBottomNeighbor( int src, int test )`

    Diese Methode prüft, ob sich der Testindex unter dem Ursprungsindex befindet.
  - findTopLeftField:
    - `private int findTopLeftField( int[] IX, boolean[] visited )`

    Es wird der Index des linken oberen Feldes einder Schleife zurückgeben.
  - findRightNeighbors:
    - `private int findRightNeighbors( int[] IX, boolean[] visited, int istart )`

    Mit der zugehörigen Hilfsmethode `isRightNeighbor` werden nun alle rechten Nachbarn des Startindex ermittelt.
  - findBottomNeighbors:
    - `private int findBottomNeighbors( int[] IX, boolean[] visited, int istart )`

    Diese Methode findet durch Aufrufen von isBottomNeighbor alle unteren Nachbarn des Startindex. Für jeden gefundenen unteren Nachbarn werden per `findRightNeighbors` die einzelnen rechten Nachbarn ermittelt.

– BuildLoopRectangleList:
  - `public void buildLoopRectangleList( Loop lp )`

  Durch die geometrischen Informationen aus dem KVD-Objektes können nun effizent alle benötigten Daten für das Zeichnen der Schleife erzeugt werden. Mit Hilfe der oben beschrieben Methoden ist es nun möglich, alle benachbarten Felder der Schleife zu ermitteln. In die graphische Repräsentation der Schleife werden alle Felder eingetragen und gesetzt.

- Zuletzt noch eine Gruppe von Hilfsmethoden.
  – inside:
    - `public boolean inside( int x, int bottom, int top )`

    Es wird true zurück gegeben, wenn sich der x-Wert innerhalb der beiden Grenzen befindet.
  – outside:
    - `public boolean outside( int x, int bottom, int top )`

    Es wird geprüft, ob sich der x-Wert außerhalb der Grenzen befindet.
  – getIndexFromPos:
    - `public int getIndexFromPos( int x, int y )`

    Der Index für die entsprechenden (x,y) Koordinaten wird zurückgegeben. Befindet sich der Koordinatenpunkt außerhalb eines KV-Diagrammes, so wird eine -1 zurück gegeben.

6.2.2 Die Klassen PLA, DrawPLA, DrawPLA_DNF und DrawPLA_KNF

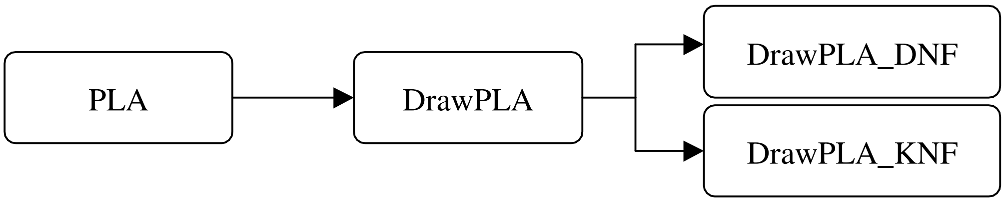


Abbildung 17: Vererbungshierarchie

Zwischen diesen vier Klassen besteht eine eindeutige Vererbungshierarchie. Die Superklasse PLA enthält alle Informationen und Methoden, um eine Schaltungsskizze funktional zu beschreiben. Diese Basisklasse wird in DrawPLA um die allgemeinen Voraussetzungen der graphischen Darstellung erweitert. In den beiden Subklassen DrawPLA_DNF und DrawPLA_KNF sind die graphischen Repräsentationen auf die gewünschte Darstellungsform angepaßt. Die Beschreibung dieser Klassen erfolgt im Abschnitt 7.2. Nur von diesen beiden Endklassen werden Objekte erzeugt, die dann die Schaltungsskizze darstellen.

Die Klasse PLA

Diese Klasse beschreibt eine allgemeine Schaltungsskizze. Sie verwaltet die Gatter (Cubes) und ermittelt ihre Realisierungskosten. In dieser Klasse werden nur die technischen Elemente realisiert.

Die Felder der Klasse PLA:

- `ni`,
  die Anzahl der Eingänge
- `no`,
  die Anzahl der Ausgänge
- `np`,
  die Anzahl der Terme
- `innames`,
  eine Liste der Eingangsbeschreibungen

- `outnames`,
  eine Liste der Beschriftungen der Ausgänge
- `cubes`,
  die doppelt verkette Liste der Gatter (Cubes)

Die Konstruktoren der Klasse PLA:

Mit Hilfe des ersten Konstuktors kann ein minimales und leeres PLA-Objekt erzeugt werden. Der zweite hat die folgende Form:

`PLA( int ni, int no, String inn[], String onn[] )`

Er erzeugt ebenfalls eine leeres PLA-Objekt, aber mit den übergebenen Eigenschaften. Bevor ein Objekt erzeugt wird, findet eine Abprüfung der Parameter `ni` und `no` statt. Die Anzahl der Eingänge darf nicht größer als 6 sein. Die Anzahl der Ausgänge ist auf 8 beschränkt. Sind alle Parameter korrekt, kann das Objekt erzeugt werden. Die Ein- und Ausgänge werden mit ihren Beschriftungen angelegt und die Liste der Gatter mit einem Wurzelobjekt initialisiert.

Die wichtigsten Methoden der Klasse PLA sind:

- die Hauptgruppe zur Verwaltung und Erzeugung der einzelnen Gatter.
  - addCube:
    - `public void addCube( int x[], int y[] )`
    - `public void addCube( Cube c )`

    Diese Methode fügt ein neues Gatter hinten an die Liste aller Cubes an. Es sind zwei Aufrufe möglich: entweder es werden die Eingangs- und Ausgangsbelegungen angegeben oder ein schon erzeugtes Cube-Objekt. Bei beiden Alternativen wird vor dem Anhängen des Objektes geprüft, ob es korrekt ist, z.B. ob die Anzahl der Ausgänge gleich der des PLA-Objektes ist. Sind alle Tests positiv ausgefallen, wird das Gatter als letztes Element an die Liste angehängt und die Anzahl der Terme erhöht.
  - deleteCube,
    löscht ein Cube-Objekt aus der Liste der Gatter.
  - deleteAllCubes,
    iteriert über alle Gatter und ruft für jedes die deleteCube-Methode auf.
- zwei wichtige Konvertierungsmethoden:

- printPLA:
  - `public String printPLA()`

  Diese Methode konvertiert das PLA-Objekt in einen String. Dieser wird dann in einem eigenen Java-Frame durch eine andere Klasse angezeigt. Diese textuelle Darstellung lehnt sich an das PLA-Format an.
- writeAsString:
  - `public String writeAsString()`

  Das PLA-Objekt wird in eine „Liste“ transformiert, als Trennzeichen der Elemente wird ein Leerzeichen benutzt. Damit kann ein PLA-Objekt innerhalb anderer Programme im HADES-Projekt eingelesen werden.

• Die Methoden für die Abschätzung des Realisierungsaufwandes:

- gateInputCosts:
  - `public int gateInputCosts()`

  Die Realisierungskosten für die Eingänge des PLA-Objektes werden hier errechnet. Jeder Gattereingang eines UND- bzw. ODER-Gatters, invertiert oder direkt, zählt als ein Eingang. Die Inverter an den Schaltungseingängen werden nicht mitgerechnet.
- gateCosts:
  - `public int gateCosts()`

  Die Kosten aller Gatter die für die Realisierung des PLA-Objektes wird ermittelt. Es werden alle angeschlossene Gatter berücksichtigt, auch wenn sie nur einen Eingang haben.

Die Klasse DrawPLA

Diese Klasse erweitert seine Basisklasse PLA um die Möglichkeit der graphischen Realisierung von Schaltungsskizzen. Sie zeichnet die Basisbausteine wie UND-, ODER-Glieder und die Anschlüsse. Logische Erweiterungen beinhaltet diese Klasse nicht.

Die Felder der Klasse DrawPLA:

- Die meisten der Felder sind Koordinaten und Größenangaben. Über das boolean Flag `klein` wird das Zoomen der Schaltungsskizze realisiert.

Die Konstruktoren der Klasse DrawPLA:

Die Konstuktoren unterscheiden sich kaum von denen der Superklasse, diese werden sogar explizit zur Erzeugung aufgerufen. Das Einflechten der Zoomvaribalen ist zu dieser Zeit die einzige Erweiterung. Anhand dieser Variablen werden die richtigen Proportionen der Bausteine ermöglicht.

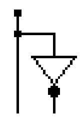
Die wichtigsten Methoden der Klasse DrawPLA sind:

Es sind graphische Ergänzung zur Basisklasse. Die Spezialisierung für die eigentlichen Darstellungen in DNF oder KNF findet in den beiden Subklassen DrawPLA_DNF und DrawPLA_KNF statt.

- drawpla:
  - `public void drawpla( Graphics g, int x0, int y0 )`

  Dieses ist die Hauptmethode der graphischen Darstellung. Außer dem nötigen Java Graphics-Objekt wird noch der Ursprungspunkt der Schaltungsskizze übergeben. Nachdem dieser gesetzt ist, werden die Methoden drawInputs, drawInputLabels und drawOutputLabels nacheinander aufgerufen.
- drawInputs:
  - `public void drawInputs( Graphics g )`

  Die Position eines Einganges wird aus den Basisdaten errechnet und die Methode drawOneInput mit diesen Parametern aufgerufen.
- drawOneInput:
  - `public void drawOneInput( Graphics g, int x, int y )`

  Ein kompletter Eingang mit Inverter wird an der entsprechenden Position gezeichnet, siehe rechte Abbildung.
- drawInputLabels und drawOutputLabels,

  Die Beschriftungen der Ein- und Ausgänge werden an die entsprechenden Koordinaten der Skizze geschrieben, siehe Abb.5 (zweite Zeile).
- drawConns,

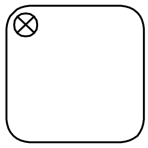
  Das Grundkonzept der möglichen Verbindungen, die PLA-Matrix wird gezeichnet. Die Anzahl der Verbindungen wird aus den PLA-Daten ermittelt. Von jedem Eingang aus werden die senkrechten Verbindungen gezeichnet, dies geschieht für die invertierten als auch für die direkten Eingänge. Als nächster

Schritt wird die Liste der Gatter durchlaufen und für die vorhandenen Gatter alle horizontalen Eingänge erzeugt. Um die invertierten Eingänge in der verkleinerten Variante von den direkten zu Unterscheiden, werden diese rot gezeichnet.

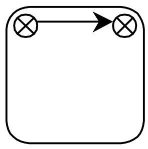
6.3 Der Algorithmus zur Erkennung von Nachbarschaft

Ein Hauptpunkt des ersten KVD-Applets [Hendrich 96] war der Algorithmus zur Erkennung von Nachbarschaft in KV-Diagrammen. Um Schleifen vom Benutzer ziehen zu lassen, muß abgeprüft werden ob die markierten Felder benachbart sind. Dieser Algorithmus wurde in der Klasse KVD implementiert. Ich werde ihn in seinen Grundzügen darstellen.

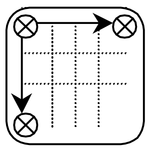
1. Dieser Algorithmus findet als erstes das linke obere Feld der Schleife, markiert es als ersten Ausgangspunkt und als besucht.



2. Vom Ausgangsfeld ausgehend werden alle rechten Nachbarn gesucht und als besucht markiert. Der Index des am weitesten rechts stehenden Feldes der Schleife beschreibt den rechten Rand der Schleife.



3. Nachdem alle rechten Nachbarn des Ausgangspunktes erkannt wurden, beginnt nun wieder vom Ausgangspunkt aus die Suche nach allen unteren Nachbarn. Existiert ein unterer Nachbar, so wird er als besucht markiert. Von ihm aus werden alle seine rechten Nachbarn gesucht und ebenfalls als besucht markiert. Der Index des untersten Feldes der Schleife beschreibt deren untere Begrenzung.



4. Am Ende eines Durchlaufes findet eine Überprüfung statt, ob alle Felder der Schleife bereits als besucht markiert wurden. Fällt dieser Test negativ aus, wird ein neuer Ausgangspunkt genommen und der Algorithmus der Nachbarschafts-erkennung erneut durchlaufen. Sind alle Felder der Schleife erkannt, ist die Schleife korrekt, und es stehen alle Parameter zur Verfügung, um die Schleife zu zeichnen.

Es folgen nun einige Abbildungen mit Beispielschleifen, die mit dem Algorithmus zur Nachbarschaftserkennung richtig erkannt werden.

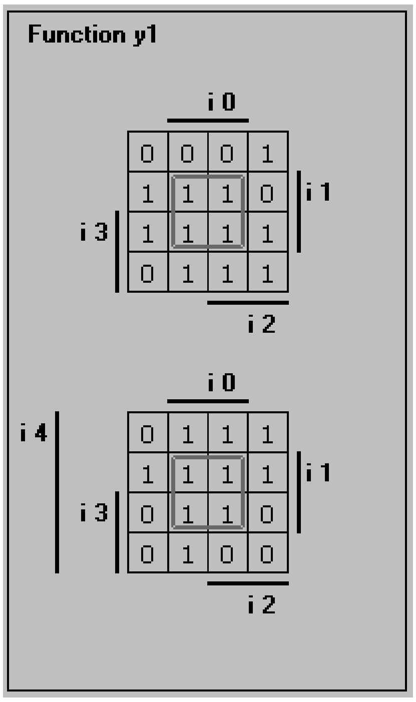


Abbildung 18: Schleife s=($i_0 \wedge i_1$) bei 5 Eingangsvariablen.

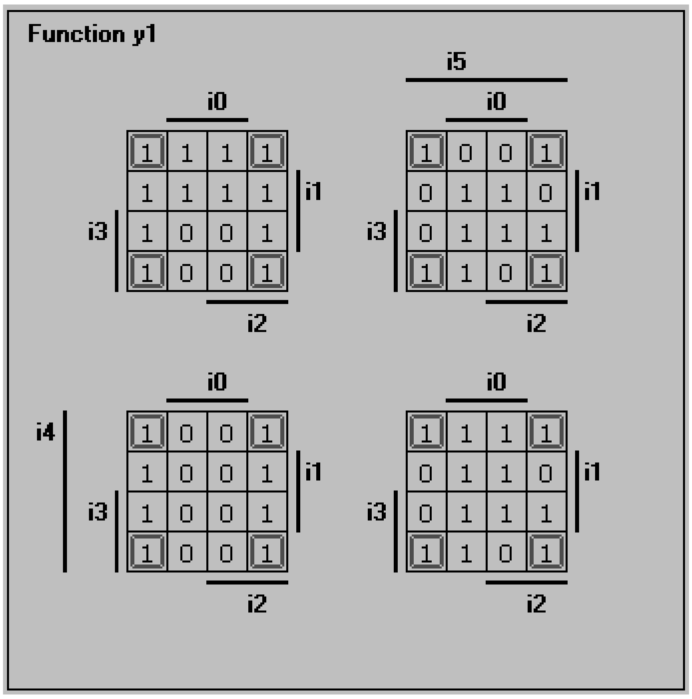


Abbildung 19: Schleife s=($\neg i_0 \wedge \neg i_1$) bei 6 Eingangsvariablen

7 Beschreibung der Erweiterungen

7.1 Übersicht über die Erweiterungen

Dieses Kapitel umfaßt die erweiterte Funktionalität dieser Arbeit. Es beginnt mit der Beschreibung der Darstellung einer Funktion in konjunktiver Normalform und der Minimierung in diesem Modus. Im darauf folgenden Abschnitt wird der Student-Mode erklärt. Er realisiert die Gatterminimierung als ein zweistufiges Verfahren, damit der Benutzer genauer erkennen kann, welche Gatter durch eine Schleife minimiert werden.
Die letzte Erweiterung ist die Bündelminimierung. Damit ist es möglich, gleichzeitig mehrere Funktionen zu minimieren.

7.2 Darstellung der Funktion in konjunktiver Normalform (KNF)

Die erste Erweiterung, die zu realisieren war, ist die Darstellung der Schaltungsskizze in konjunktiver Normalform (siehe Abb. 8 auf Seite 4-16 rechts), sowie die Möglichkeit, die Funktion in KNF im KV-Diagramm zu minimieren.
Um diesen Modus zu implementieren, wurde die ursprüngliche Klasse DrawPLA, da sie nur die disjunktive Darstellung realisierte, in zwei Klassen aufgeteilt. Es entstand die Superklasse DrawPLA (siehe Kapitel 6.2.2) und deren Subklasse DrawPLA_DNF (siehe Abb. 17 auf Seite 6-32). Neu hinzu kam die Subklasse DrawPLA_KNF, sie realisiert die konjunktive Darstellung. Beide Subklassen sind von der Struktur identisch, somit beschränke ich mich auf die Beschreibung der Klasse DrawPLA_KNF.

Die Klasse DrawPLA_KNF

Diese Klasse beinhaltet alle Methoden, um ein DrawPLA-Objekt in konjunktiver Normalform darzustellen. Für die einzelnen Komponenten einer Schaltungsskizze wie ODER-Glied, UND-Glied und die Anschlüsse existiert je eine Methode. Für die ODER-Gatter stehen zwei Repräsentationen in unterschiedlicher Größe zur Verfügung, somit wurde das Verkleinern auf 50% der Orginalgröße realisiert. Eine zentrale Methode sorgt dafür, daß alle Komponenten der Schaltungsskizze an der richtigen Stelle gezeichnet werden, unabhängig welcher Größenmodus eingestellt ist.

Alle hierfür relevanten Daten wurden durch die beiden Superklassen PLA und DrawPLA auch der Klasse DrawPLA_KNF bereit gestellt.

7.3 Der Student-Mode, Schleifen ziehen als ein zweistufiges Verfahren

Um das Minimieren mittels Schleifen didaktisch besser darzustellen zu können, wurde ein zusätzlicher Modus entwickelt, der Student Mode. Dabei sollte die Minimierung der Gatter in der Schaltungsskizze so lange verzögert werden, bis der Benutzer die Schleifenbildung bestätigt.

Der Student Mode veranschaulicht, welche Gatter von einer Schleife zusammengefaßt werden. Dies wird durch eine Zweiteilung des Minimierungsvorganges erreicht. Im ersten Schritt wird die Schleife im KV-Diagramm gezogen und die betroffenen Gatter werden in der Schleifenfarbe hervorgehoben. Im zweiten Schritt wird die Schleife durch den *build Loop* Button bestätigt und die markierten Gatter werden zu einem minimiert. Ohne diesen Zusatzmodus werden die betroffenen Gatter sofort durch die Schleife minimiert. Es war somit schwer möglich, den Zusammenhang zwischen Mintermen oder Maxtermen und dem minimierten Gatter zu erkennen. Der Student-Mode verbessert dieses durch das Warten auf eine Benutzeraktion, um das minimierte Gatter zu erzeugen.

Es folgt ein Beispiel für den Student-Mode [Mertsching 95/96]

In der nachfolgenden Grafik (Abb.20) ist nun eine Schleife im Student Mode gezogen worden, und die vier überdeckten Gatter in der Schaltungsskizze wurden in der Schleifenfarbe hervorgehoben. Die Schleife wird im KV-Diagramm angezeigt, aber eine Minimierung fand noch nicht statt.

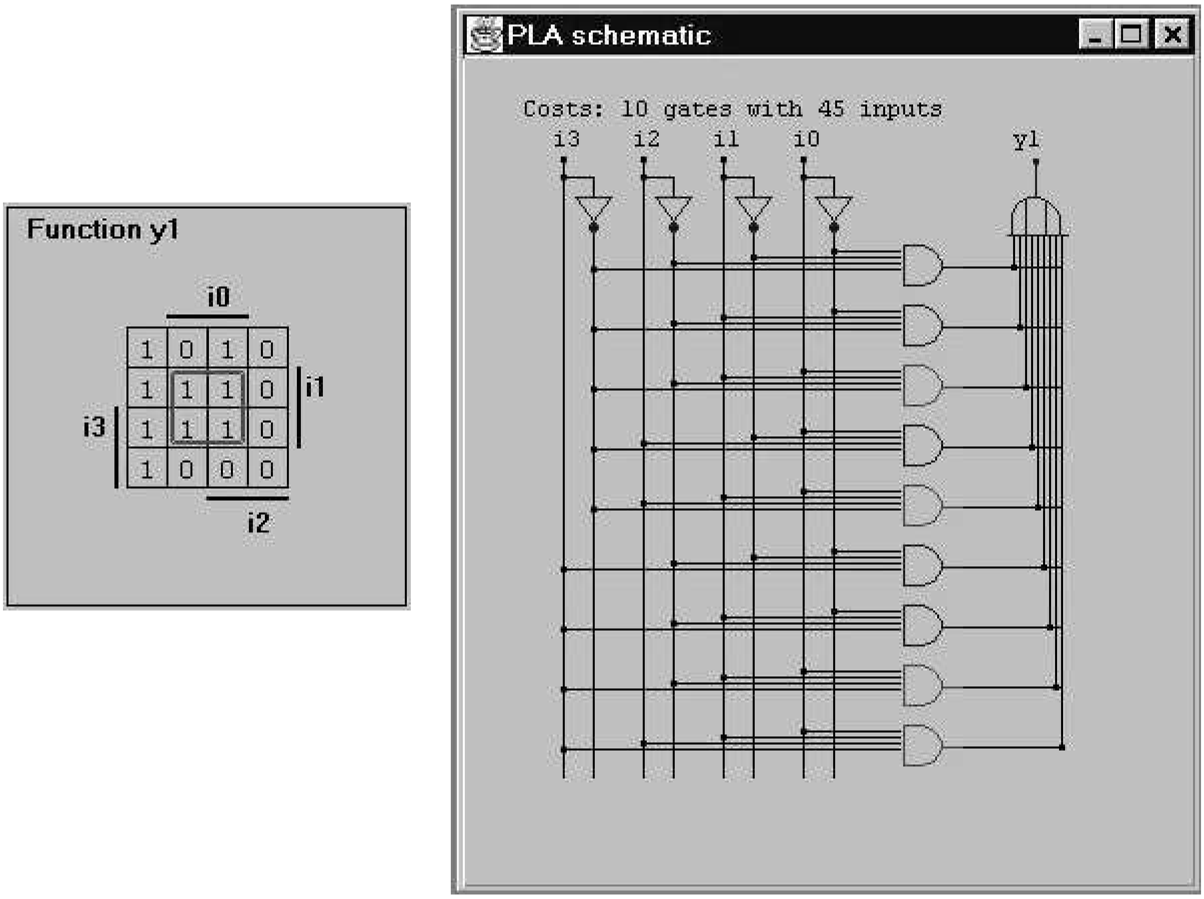


Abbildung 20: Ein Beispiel für den Student Mode.

Im zweiten Schritt wurde die Schleife durch Betätigen des `Build-Loop`-Buttons bestätigt, und die eben noch hervorgehobenen Gatter wurden zu einem minimierten Gatter zusammengefaßt, siehe Abb. 21.

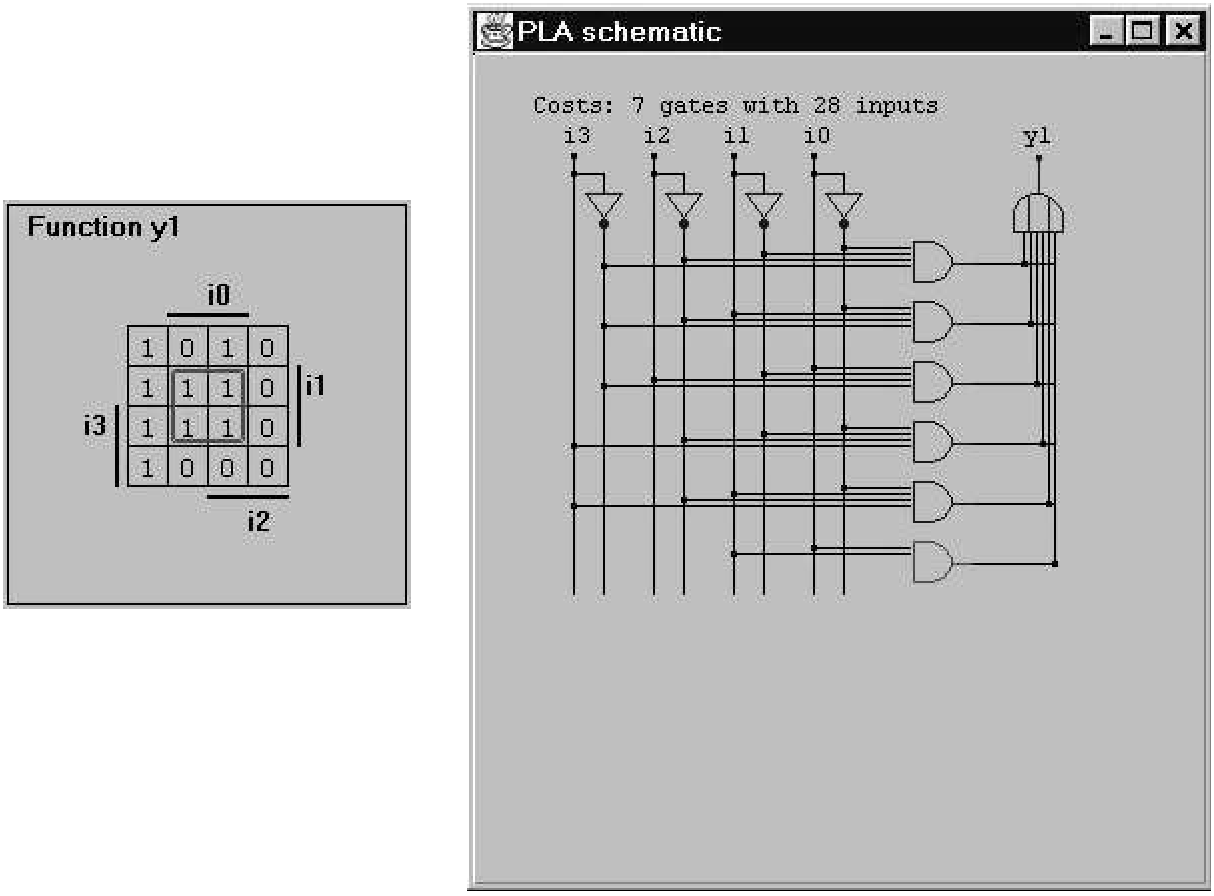


Abbildung 21: Die Schleife wurde gebildet.

7.4 Die Bündelminimierung

Um eine Minimierung mehrerer Ausgänge in der DNF zu realisieren, mußten erhebliche Umbauten am Applet vorgenommen werden. Das Grundprinzip mit einem KVD-Objekt und dem dazu gehörigen PLA-Objekt reichte nicht mehr aus, um mehrere Funktionen zu verwalten. Das Applet wurde deshalb um einem Vektor von KVD-Objekten und einem PLA-Objekt erweitert. Es war nun möglich, mehrere verschiedene Funktionen, in verschiedenen KV-Diagrammen darzustellen. Die Klasse PLA und somit seine Subklassen wurden auf die Darstellung und Verwaltung mehrere Ausgänge erweitert. Die Ausgangssituation einer Bündelminimierung konnte nun angezeigt werden, siehe Abb. 24.

Das nächste Problem war das gleichzeitige Anlegen und Verwalten von Schleifen in mehreren KV-Diagrammen. Der Benutzer sollte wie gewohnt eine Schleife in einem

KV-Diagramm ziehen (siehe Abb. 22) und diese Schleife sollte in allen anderen KV-Diagrammen gezogen werden, in denen sie möglich ist, siehe Abb. 23.
Bevor eine Schleife in einem anderen KV-Diagramm gezogen wird, finden einige Tests statt, ob die Schleife vielleicht schon existiert, oder ob sie von einer anderen Schleife überdeckt und so mit in einen KV-Diagramm redundant ist. Durch dieses Testen der Schleifen wird die Benutzerfreundlichkeit erhöht, denn der Anwender muß möglichst wenige unerwünschte Schleife löschen. Nachdem nun die Schleifen in den KV-Diagrammen gezogen wurden, muß die Schaltungsskizze der neuen Minimierungs-situation angepaßt werden.
Der nachfolgende Algorithmus erfüllt diese Aufgabe und erzeugt eine neue Schaltungsskizze mit den Minimierungen durch die Schleifen. Es wird nur der Grundalgorithmus der Bündelminimierung dargestellt und auf langwierige Einzelheiten verzichtet.
Im ersten Schritt wird ein neues leeres PLA-Objekt erzeugt, und für jeden Minterm der einzelnen KV-Diagramme wird ein entsprechendes UND-Gatter angelegt. Sollten keine Schleifen vorhanden sein, ist mit diesem Schritt die Ausgangssituation einer Bündelminimierung bereits erreicht, siehe Abb. 24.
Der nächste Schritt iteriert über alle vorhandenen KV-Diagramme, d.h. die folgenden Schritte werden zunächst für ein KV-Diagramm durchgeführt. Wenn diese alle abgeschlossen wurden, wird mit dem nächsten KV-Diagramm dieselbe Sequenz durchlaufen. Als erstes wird die Liste der vorhanden Schleifen des KV-Diagrammes durchgegangen. Für jede existierende Schleife wird ein passendes UND-Gatter im PLA-Objekt eingetragen. Anschließend wird die Gatterliste des PLA-Objektes durchlaufen und abgeprüft, ob das Gatter aus der Liste vom „Schleifen-Gatter" überdeckt wird. Ein „Listen-Gatter" wird überdeckt von einem „Schleifen-Gatter", wenn es genau an den selben Ausgang angeschlossen ist und die Eingangsbelegung des „Schleifen-Gatters" gleich oder allgemeiner ist. Sollte ein Gatter aus der Liste als redundant erkannt werden, wird es aus der Liste gelöscht. Nachdem alle überdeckten Gatter aus dem aktuellen KV-Diagramm erkannt und gelöscht wurden, müssen nun alle anderen KV-Diagramme durchgesehen werden, ob sie die selbe Schleife enthalten. Wenn in einem KV-Diagramm die selbe Schleife gefunden wurde, werden wieder alle überdeckten Gatter aus der Liste im PLA-Objekt gelöscht und das „Schleifen-Gatter" zusätzlich mit dem entsprechenden Ausgang verbunden, siehe Abb. 25/26. Dieser Vorgang ist für alle

Schleifen zu wiederholen, dabei sollten bereits bearbeitete Schleifen nicht mehr bearbeitet werden. Das folgende Beispiel soll das Vorgehen bei einer Bündelminimierung verdeutlichen.

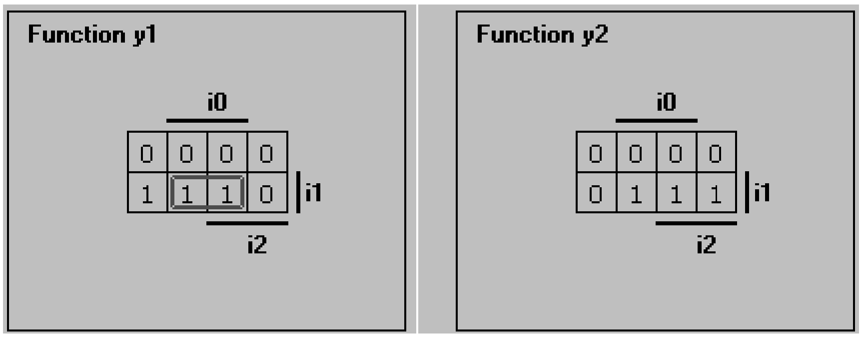


Abbildung 22: Der Benutzer hat in der Bündelminimierung eine Schleife gezogen.

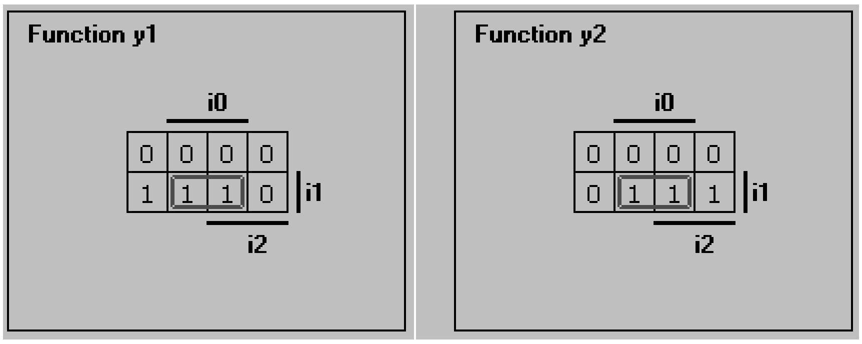


Abbildung 23: Die gezogene Schleife paßt auch das andere KV-Diagramm, und wird angelegt.

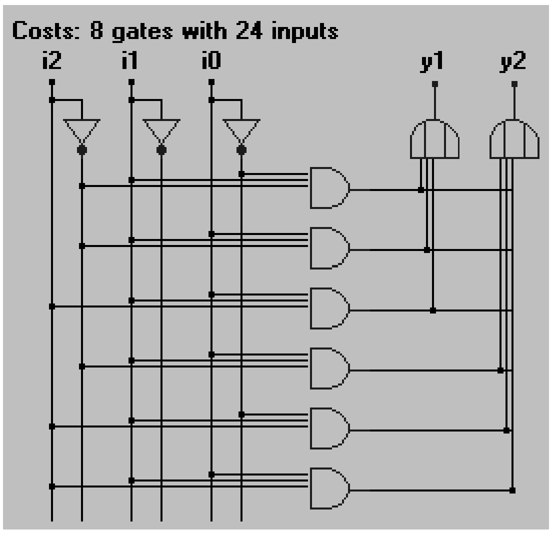


Abbildung 24: Die Ausgangssituation der Bündelminimierung.

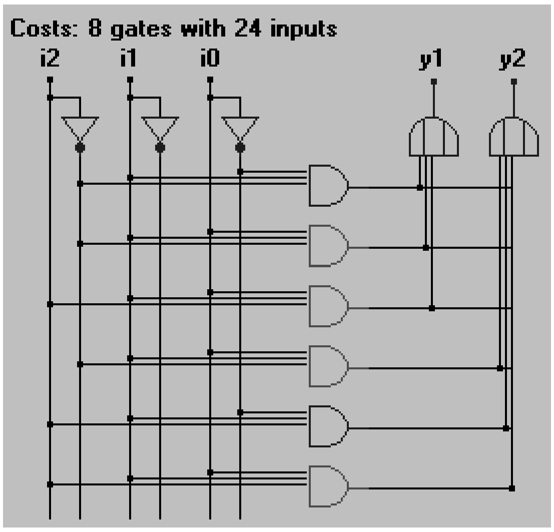


Abbildung 25: Ein Zwischenergebnis der Bündelminimierung im Student-Mode, alle Überdeckungen wurden erkannt.

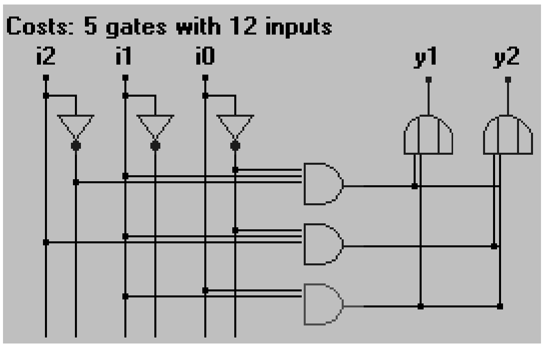


Abbildung 26: Die Bündelminimierung wurde durchgeführt, das Gatter für die Schleife ist an beide Ausgange angeschlossen und alle überdeckten Gatter wurden gelöscht.

8 Anhänge

8.1 Beispiele der Minimierung

8.1.1 Würfel-Anzeige

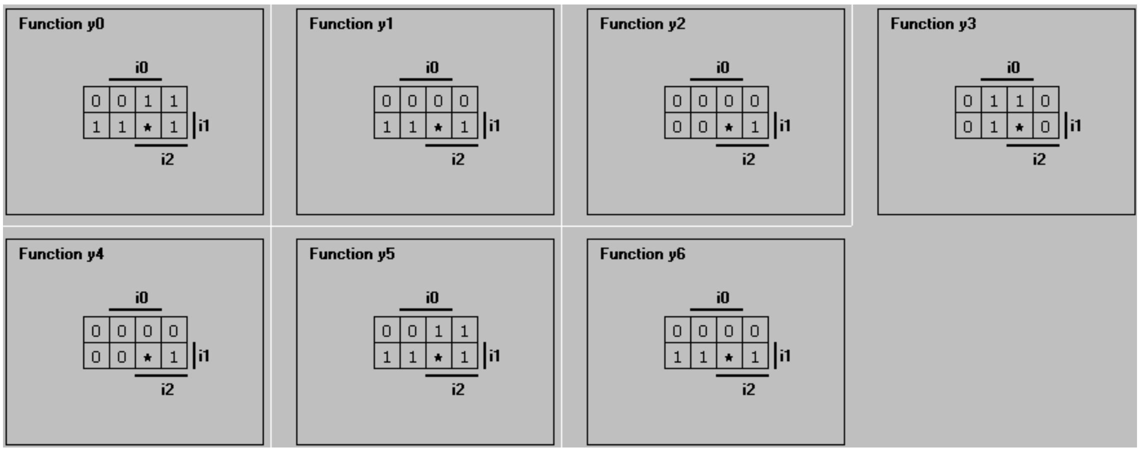


Abbildung 27: Die sieben KV-Diagramme der Würfel-Anzeige („dice diagram“).

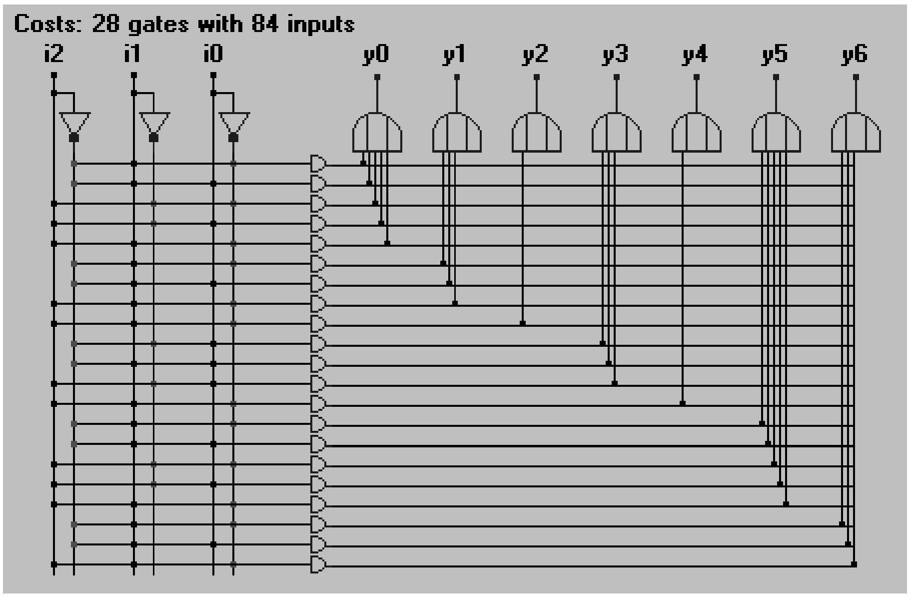


Abbildung 28: Die Ausgangsschaltungsskizze des „dice diagram“.

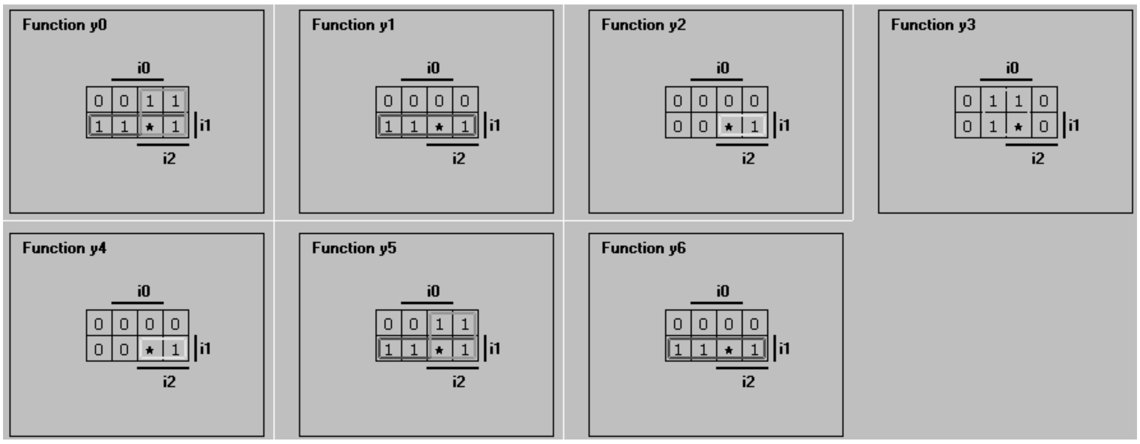


Abbildung 29: Die KV-Diagramme mit den Schleifen.

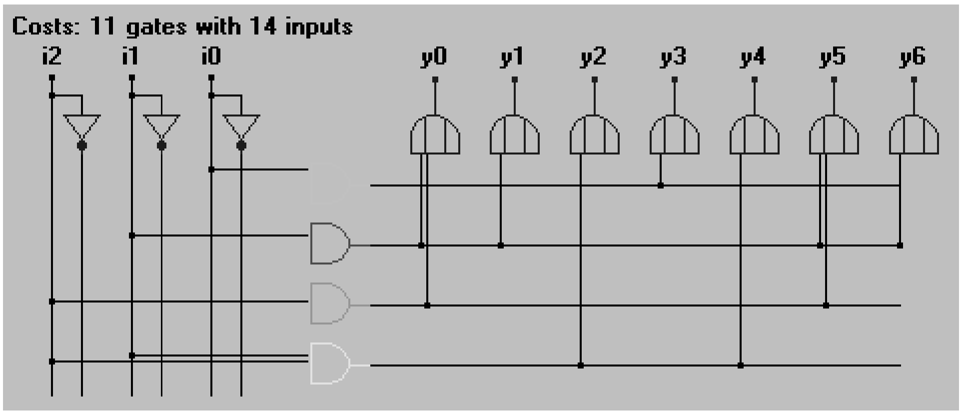


Abbildung 30: Die Schaltungsskizze der Minimierung.

8.1.2 Sieben-Segment-Anzeige

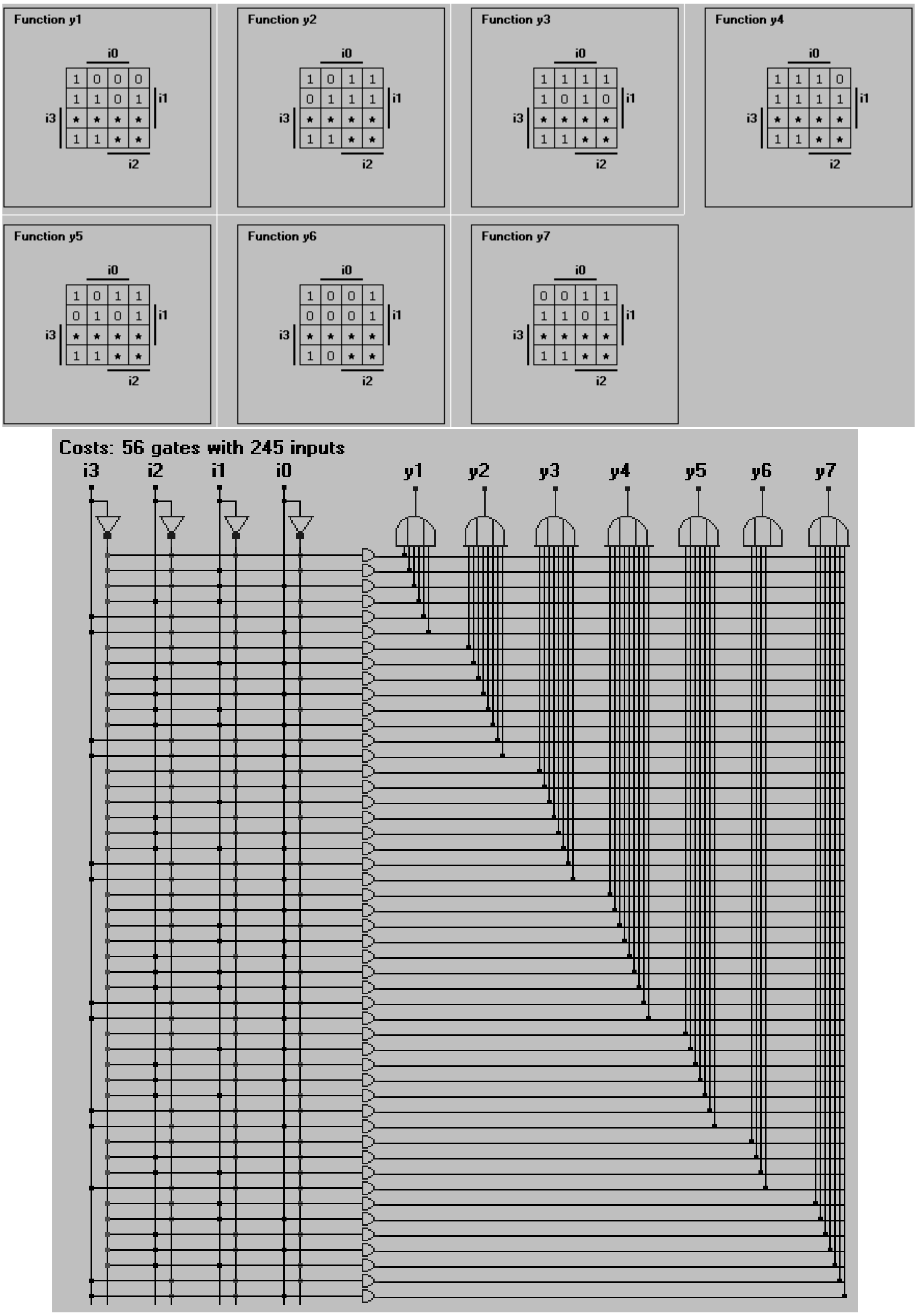


Abbildung 31: Die Ausgangssituation der 7-Segment-Anzeige.

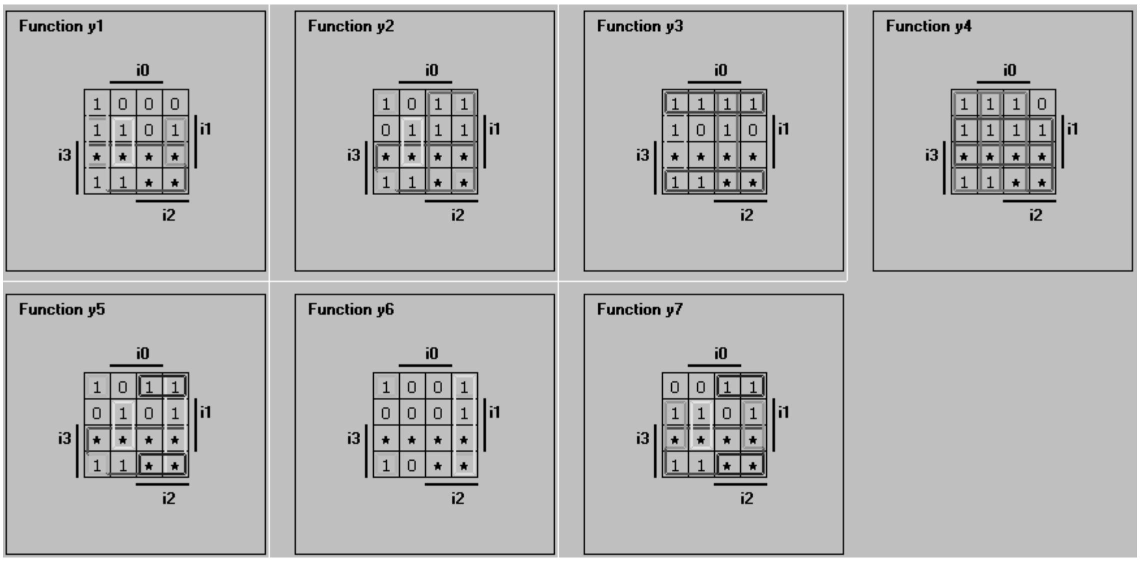


Abbildung 32: Die 7-Segment-Anzeige mit allen Schleifen [Lagemann 87]

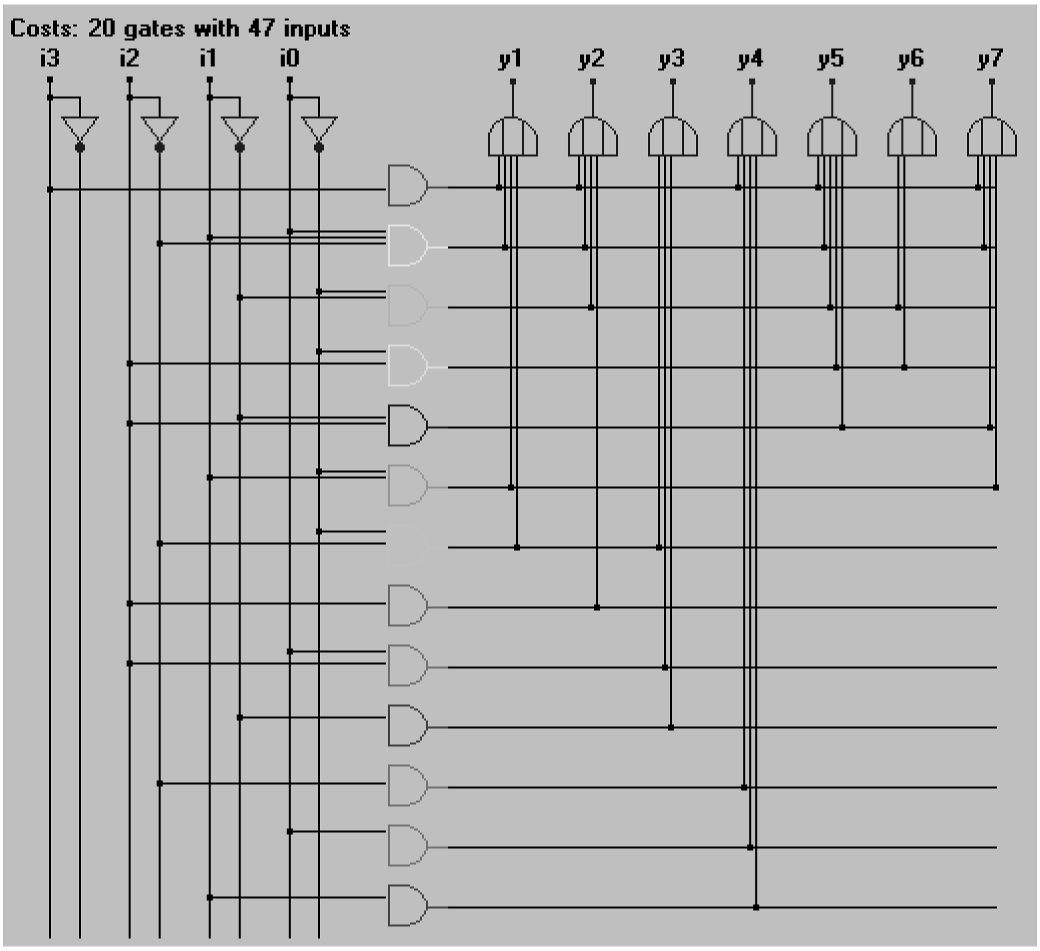


Abbildung 33: Eine Minimierung [Lagemann 87].

9 Literaturliste:

[Lagemann 87] K.Lagemann, **Rechnerstrukturen**, Springer-Verlag 1987,

Eine Einführung in das Gebiet der Rechnerstrukturen. Das Buch wendet sich an Studenten der Informatik, die sich im Grundstudium befinden, und behandelt umfassend die nötigen Grundlagen.

[Schiffmann, Schmitz 92] W.Schiffmann, R.Schmitz, **Technische Informatik 1**: Grundlagen digitalen Elektronik, Springer-Verlag 92,

Dieses Buch entstand aus den Skripten zu den Vorlesungen der Technischen Informatik an Universität Koblenz. Es führt in die für die Elektronik wesentlichen Gesetzte der Physik und Elektrotechnik ein.

[Mertsching 95/96] B. Mertsching, **Grundzüge der Informatik B1**,

Dieses ist das Skript zur Grundstudiums Vorlesung B1, des technischen Zyklus der Universität Hamburg.

[Hendrich 96] N. Hendrich**, KVD-Applet**, http:\\tech-www.informatik.uni-hamburg.de /applets/kvd/

[Hendrich 97] N. Hendrich, **HADES**, http:\\tech-www.informatik.uni-hamburg.de /applets/hades/

[Hendrich 97a] N.Hendrich, **Java für Fortgeschrittene**, Springer-Verlag 97.

[Flanagan 96] D. Flanagan, **Java in a nutshell**, O´Reailly 96.

[Middendorf, Singer, Strobel] S. Middendorf, R. Singer, S. Strobel**, Java Programmier Handbuch und Referemz**, dpunkt 96.

[Gosling, Yellin 96] J. Gosling, F. Yellin**, The Java Application Programming Interface Volume 1**, Addison Wesley 96.

[Gosling, Yellin 96a] J. Gosling, F. Yellin**, The Java Application Programming Interface Volume 2**, Addison Wesley 96.

[Back 96] S. Back, **Professionelle Java-Programmierung**, Internat. Thomson Publ. 96.