Universität Hamburg, Fachbereich Informatik

Arbeitsbereich Technische Aspekte Multimodaler Systeme

Praktikum der Technischen Informatik

T1 – 1

Schaltnetze

Name: ....

Bogen erfolgreich bearbeitet: ....

Scheinkriterien

- Für die Erteilung des Scheins ist die erfolgreiche Bearbeitung **aller** nicht als optional gekennzeichneter Aufgaben notwendig. Maßgeblich sind die Aufgaben, die zum aktuellen Semester im WEB stehen. Der Schein wird nur bei Vorlage **aller** abgezeichneten Unterlagen vergeben.
- Für die Erteilung des Scheins weiterhin ist die Anwesenheit und das rechtzeitige Erscheinen zu allen Terminen des jeweiligen Kurses erforderlich. Ein Fehlen ist nur aus berechtigten Gründen zulässig. Der versäumte Termin muss dann nach Absprache mit den Betreuern im gleichen Semester nachgeholt werden.
- Zu jedem Termin müssen die TeilnehmerInnen alle vorher schon bearbeiteten Versuchsunterlagen mitbringen.
- Die Aufgaben sollten in Gruppen von höchstens zwei Personen bearbeitet werden. Die Zusammensetzung der Gruppen wird am ersten Termin festgelegt und bleibt für den Rest des Praktikums bestehen
- Die Bearbeitung der Aufgaben erfolgt in Gruppen, das Erfüllen der Scheinkriterien wird aber noch individuell geprüft.
- Die Lösung der Aufgaben muss in den Unterlagen nachvollziehbar dokumentiert werden. Bevor eine Aufgabe für die Gruppe nicht als erfolgreich bearbeitet abgezeichnet ist, sollte nicht mit der Bearbeitung der nächsten Aufgabe begonnen werden.
- Der Versuchsbogen wird nur als bearbeitet abgezeichnet, wenn alle darin vorhandenen Aufgaben erfolgreich gelöst worden sind, dies im Versuchsbogen dokumentiert worden ist und die Studierenden ein Grundverständnis des Stoffes gezeigt haben, das gegebenenfalls durch einfache Fragen geprüft wird.

Anmerkung:
Erfahrungen aus vergangenen Semestern haben gezeigt, dass häusliche Vorbereitung, die über das bloße Lesen der Aufgaben hinausgeht, unerlässlich ist.

Einführung

Unter einem Schaltnetz versteht man die technische Realisierung einer oder mehrerer boolscher Funktionen. Die Struktur der Realiserung wird repräsentiert durch einen schaltalgebraischen Ausdruck, bei dem nur die Verknüpfungen *ODER*, *UND* und *NICHT* verkommen. Umwandlung eines schaltalgebraischen Ausdrucks gemäß den bekannten Regeln bedeutet also Abwandeln der Struktur für dieselbe Funktion. Der Realisierungsaufwand wird gemessen in der Zahl der im Ausdruck benutzten Verknüpfungen. Wichtig in der Praxis ist auch die sog. Tiefe des Netzes. Dies ist die maximale Zahl der Verknüpfungen, die aufgrund von Datenabhängigkeiten nacheinander ausgeführt werden müssen. Wenn man alle Verknüpfungen klammert, ist dies die maximale Tiefe des Klammergebirges. Bereiten Sie sich gründlich auf diesen Themenkreis vor. Insbesondere sollten Funktionstabellen, KV-Diagramme und schaltalgebraische Ausdrücke für die Versuche dieser Reihe schon zuhaus ermittelt werden.

Beispiele dieser Versuchsreihe

Für den Gray-Code und den $\binom{4}{2}$-Code sind Codeprüfer zu realisieren, d.h. jeweils eine Funktion, die genau dann **1** liefert, wenn das als Argument angebotene Wort ein Codewort ist. Codewörter sind:

Gray-Code

x_4	x_3	x_2	x_1
0	0	0	0
0	0	0	1
0	0	1	1
0	0	1	0
0	1	1	0
0	1	1	1
0	1	0	1
0	1	0	0
1	1	0	0
1	0	0	0

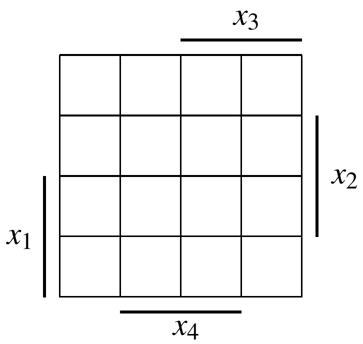
$\binom{4}{2}$-Code

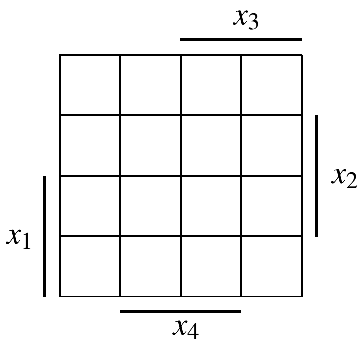
x_4	x_3	x_2	x_1
1	1	0	0
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	1
0	0	1	1

Funktionstabellen

x_4	x_3	x_2	x_1	Gray-Code Prüfer	$\binom{4}{2}$-Code-Prüfer
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

KV-Diagramme





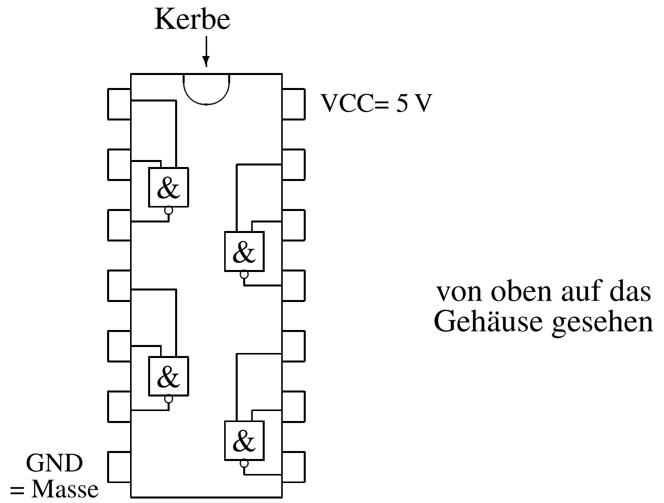
Schaltalgebraische Ausdrücke

Umwandlung in Ausdrücke, die nur die Verknüpfung *NAND* und *NOT* enthalten:

Technische Realisierung in Form eines Schaltplans, der nur *NAND*-Verknüpfungen mit zwei Eingängen und *NOT*-Elemente enthält:

<u>Versuch 1.1</u> Aufbau und Test eines Schaltnetzes

Bauen Sie den Gray-Code-Prüfer mit NAND-Gattern auf (TTL SN 7400). In dem 14-poligen Gehäuse sind vier solcher Gatter enthalten:



Verbinden Sie die Eingänge des Codeprüfers mit den Digitalquellen Do1 ... Do4 und den Ausgang mit dem Digitaleingang DI1. Falls Sie eine logische **1** gebrauchen, können Sie den Eingang einfach auf 5 V legen.

Starten Sie jetzt das Programm **WINETPS** und legen Sie auf Do1…Do4 einen Binärzähler (Menue **Digital/ Zähler setzen**), der automatisch die benötigten 16 Eingabemuster erzeugt. Gehen Sie dann in das Menue **Digital/Ein- und Ausgänge setzen**, um Ihren Messeingang zu aktivieren. [1] Im nächsten Schritt muss die Messung durchgeführt werden (Menue **Digital/Messung starten**). Danach kann man sich die Ergebnisse als Tabelle oder Grafik ansehen. Vergleichen Sie mit den Sollwerten!

Aufgabe gelöst:	Gruppe:	TeilnehmerIn: .... vom Betreuer auszufüllen

[1] In der aktuellen Version des Programms gibt es einen weiteren Menuepunkt **T1-Shortcut**, in dem man nur noch die Zahl der Eingabemuster und die Zahl der Messeingänge angeben muss, bevor man die Messung starten kann.

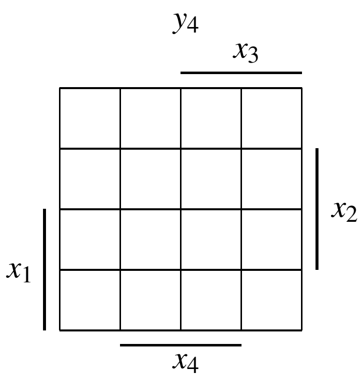
<u>Versuch 1.2</u> Funktionsbündel

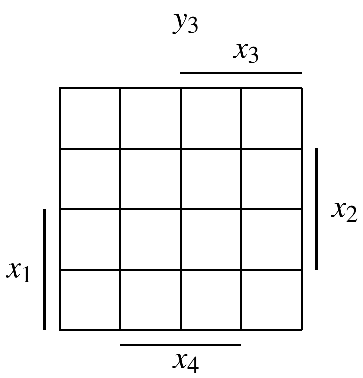
In der Praxis sind sehr häufig mehrere Funktionen derselben Argumente zu realisieren (ein sog. Funktionsbündel). Wir nehmen hierzu als Beispiel einen Binärmultiplizierer mit zwei Zweibit-Operanden $x_4x_3 \cdot x_2x_1$ (binärkodiert: 0, 1, 2, 3) und dem Dreibit-Produkt $y_3y_2y_1$ (binärkodiert: 0, 1, 2, 3, 4, 6). Zusätzlich wollen wir noch ein Signal y_4 als Overflow-Flag einführen, das genau dann den Wert **1** hat, wenn sich das Ergebnis der Multiplikation nicht mehr in drei Bit darstellen lässt. In diesem Fall wollen wir weiter annehmen, dass y_1, y_2 und y_3 undefiniert sind, d.h. beliebige Werte annehmen können.

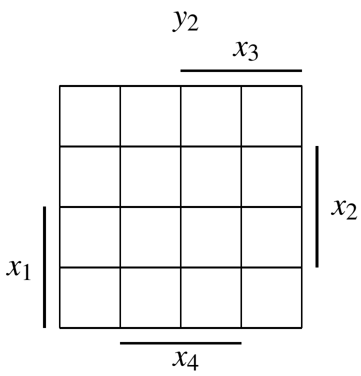
<u>Funktionstabelle</u>

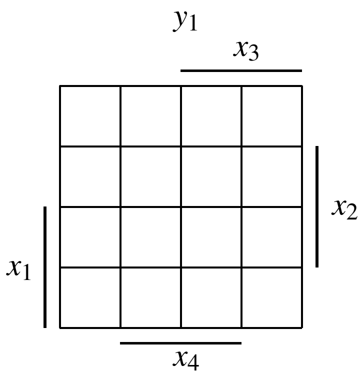
Operand1		Operand2					
x_4	x_3	x_2	x_1	y_4	y_3	y_2	y_1
0	0	0	0				
0	0	0	1				
0	0	1	0				
0	0	1	1				
0	1	0	0				
0	1	0	1				
0	1	1	0				
0	1	1	1				
1	0	0	0				
1	0	0	1				
1	0	1	0				
1	0	1	1				
1	1	0	0				
1	1	0	1				
1	1	1	0				
1	1	1	1				

<u>KV-Diagramme</u>









Schaltalgebraische Ausdrücke

$$y_1 =$$

$$y_2 =$$

$$y_3 =$$

$$y_4 =$$

Versuchen Sie dabei die Zahl der Verknüpfungen der vier Ausdrücke insgesamt durch Einführung gemeinsam benutzter Terme, die dann auch nur einmal realisiert zu werden brauchen, zu verringern .

Bauen Sie den Binärmultiplizierer auf und prüfen Sie dessen Funktion mit Hilfe sämtlicher Eingabebitmuster. Verwenden Sie den schon besprochenen TTL-Baustein SN 7400 (4-fach NAND) und den SN 7408 (4-fach AND mit gleicher Pinbelegung wie SN 7400).

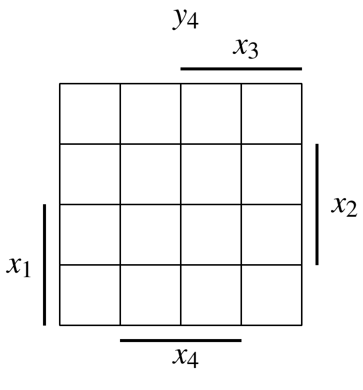
Aufgabe gelöst: Gruppe: .... TeilnehmerIn: ....
vom Betreuer auszufüllen

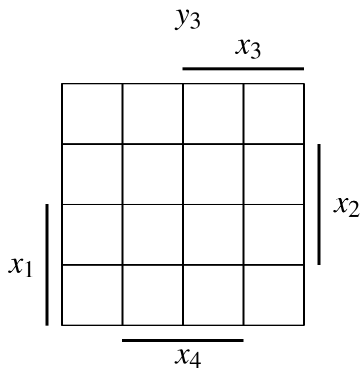
<u>Versuch 1.3</u> Gray-Code - Dual-Code Umwandler

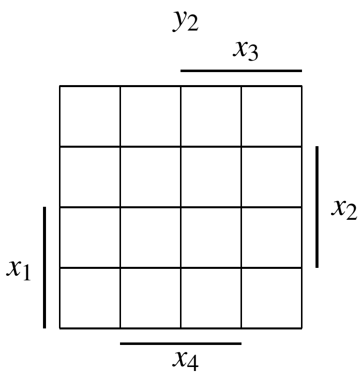
Der Gray-Code hat die für manche technische Anwendungen günstige Eigenschaft, dass sich beim Übergang von einem Codewort zum nächsten immer genau ein Bit ändert. Dafür ist er aber für arithmetische Operationen weit weniger geeignet als der normale Dual-Code. Will man also Berechnungen durchführen, ist es sinnvoll, je ein Schaltnetz für die Umwandlung vom Gray-Code in den Dual-Code bzw. vom Dual-Code in den Gray-Code zur Verfügung zu haben. Solche Schaltungen wollen wir nun entwerfen:

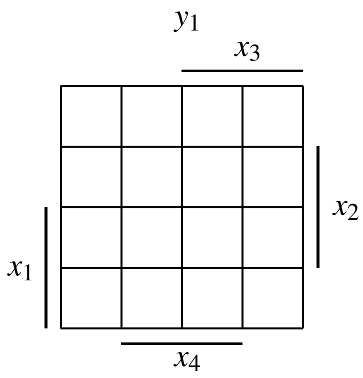
Zahl	Gray-Code				Dual-Code			
	y_4	y_3	y_2	y_1	x_4	x_3	x_2	x_1
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	1	0	0	1	0
3	0	0	1	0	0	0	1	1
4	0	1	1	0	0	1	0	0
5	0	1	1	1	0	1	0	1
6	0	1	0	1	0	1	1	0
7	0	1	0	0	0	1	1	1
8	1	1	0	0	1	0	0	0
9	1	1	0	1	1	0	0	1
10	1	1	1	1	1	0	1	0
11	1	1	1	0	1	0	1	1
12	1	0	1	0	1	1	0	0
13	1	0	1	1	1	1	0	1
14	1	0	0	1	1	1	1	0
15	1	0	0	0	1	1	1	1

Zuerst wollen wir den Umwandler vom Dual-Code in den Gray-Code entwerfen.









Schaltalgebraische Ausdrücke

$y_1 =$

$y_2 =$

$y_3 =$

$y_4 =$

Für y_1, y_2 und y_3 sollten sie ähnlich aufgebaute Terme erhalten. Welcher logischen Funktion entsprechen sie? Werfen Sie notfalls einen Blick in die Formelsammlung am Ende des Bogens.

Bauen Sie jetzt diese Schaltung auf und testen Sie sie.

Entwerfen Sie nun den Gray-Code-Dual-Code Umwandler. Das geht hier am einfachsten ohne KV-Diagramme durch direkte Umrechnung.

$x_1 =$

$x_2 =$

$x_3 =$

$x_4 =$

Aufgabe gelöst:	Gruppe:	TeilnehmerIn: .... vom Betreuer auszufüllen

<u>Versuch 1.4</u> Eine Addiererschaltung

In den Versuchen 1.1 - 1.3 hatten wir die Schaltungen mit realen Bauteilen aufgebaut. Für die Versuche 1.4 und 1.5 wollen wir einen anderen Weg gehen und sie nur noch simulieren. Dazu benutzen wir den im ehemaligen AB TECH (jetzt TAMS) des Fachbereichs entwickelten, frei verfügbaren Simulator HADES.

Entwerfen Sie zuerst eine Schaltung, die abhängig von einem Signal AS zwei vier Bit breite Binärzahlen (in Zweierkomplementdarstellung) $A = a_4a_3a_2a_1$ und $B = b_4b_3b_2b_1$ addiert (AS= 0) bzw. subtrahiert (AS= 1). Das Ergebnis ist eine vier Bit breite Zahl $S = s_4s_3s_2s_1$ und ein Übertrag (Carry).

Wenn man über diese Aufgabe etwas nachdenkt, lässt sich die ganze Schaltung recht einfach realisieren, indem man vor den einen Vier-Bit-Eingang des Addierers etwas Logik (vier Gatter) schaltet. Wer eine kompliziertere Lösung vorzieht, kann natürlich auch diese aufbauen.

Skizzieren Sie hier Ihre Schaltung:

Hinweise zur praktischen Durchführung:

Um Ihre Schaltung zu simulieren, starten Sie zunächst den Simulator durch Klicken auf das Icon **Runhades**. Es erscheint ein Fenster mit einer Zeichenfläche. Als erstes holen wir uns einen Vier-Bit-Addierer. Gehen Sie mit Maus auf diese Fläche und drücken Sie die rechte Maustaste. Es erscheint ein Popup-Menue. Wählen Sie **Create - RTL - adder** aus. In der Zeichenfläche erscheint dann ein

Symbol mit 14 Anschlüssen. Sollten diese keine Namen tragen, gehen Sie in das Menue **Display** und klicken Sie **port labels** an. Das Symbol lässt sich verschieben, indem man es mit der rechten Mausttaste anklickt und dann aus dem Popup-Menue **move** auswählt. A4-A1 und B4-B1 sind bei diesem Bauteil zu addierenden Vier-Bit-Zahlen, CIN der Übertrag aus einer möglichen vorhergehenden Addierstufe, S4-S1 das Ergebnis und COUT der Übertrag aus der Addition, der in eine mögliche nachfolgende Stufe weitergegeben werden kann.

Als Eingänge für unsere Schaltung nehmen wir hier zwei Hex-switches aus dem Popup-Menue **create - I/O**, die jeweils vier Bit breite Zahlen liefern. Plazieren Sie die Symbole geeignet, aber vergessen Sie nicht, noch etwas Platz für die Logik zu lassen, die Sie außerdem noch brauchen. Für den Eingang AS kann man ein Ipin aus dem Popup-Menue **create - I/O** nehmen, für die Ausgänge S ein Hex-Display und für COUT einen Opin oder eine LED.

Um jetzt z.B. den Ausgang COUT des Addierers mit dem Opin zu verdrahten, klickt man erst auf den roten Punkt bei COUT und dann auf den roten Punkt beim Opin. Man beachte dabei, dass sich eine Verbindung auch aus einzelnen Segmenten zusammensetzen lässt, damit man einen übersichtlicheren Leitungsverlauf erhält.

Vervollständigen Sie jetzt ihre Schaltung. Die Logik-Gatter findet man dabei unter **create - gates**.

Zu Testen der Schaltung klicken Sie nacheinander ganz unten im Fenster auf den Doppelpfeil nach links und den Doppelpfeil nach rechts. Nachdem Sie für AS durch Klicken auf den Ipin einen gültigen Wert eingestellt haben (grau - 0, rot - 1), sollte keine der Leitungen mehr hellblau (undefiniert) sein und das Display einen Wert anzeigen. Testen Sie jetzt Ihre Schaltung.

Aufgabe gelöst:	Gruppe:	TeilnehmerIn: .... vom Betreuer auszufüllen

<u>Versuch 1.5</u> Ein Addierer für zwei BCD-Ziffern

Es ist nicht immer günstig, die Zahlen, mit denen man arbeitet, im Dualsystem darzustellen. So kann man z.B. eine Dezimalzahl auch ziffernweise codieren. Man spricht dann von einem BCD-Code (**B**inary **c**oded **d**ecimal). Üblich, aber nicht unbedingt notwendig, ist es, für die einzelnen Ziffern einfach die normale Darstellung als Dualzahl zu wählen. Die Zahl 159 hätte also z.B. die BCD-Darstellung

$$\underbrace{0001}_{1}\,\underbrace{0101}_{5}\,\underbrace{1001}_{9}$$

Zur Unterscheidung von positiven und negativen Zahlen könnte man mit einem zusätzlichen Vorzeichenbit arbeiten. Im folgenden wollen wir aber nur positive Zahlen betrachten. Wenn man mit BCD-Zahlen rechnen will, stößt man auf gewisse Probleme. Wir wollen hier nun eine Schaltung entwerfen, die zwei Ziffern im BCD-Code addiert. Ein vollständiger Addierer für mehrere Stellen ließe sich dann analog zur Verschaltung mehrerer Dual-Volladdierern aufbauen.

Wenn man zwei BCD-Ziffern $A = a_4a_3a_2a_1$ und $B = b_4b_3b_2b_1$ addieren will, könnte man

> ein vollständiges Schaltnetz mit neun Eingängen A, B und dem Übertrag aus der vorhergehenden Stufe und fünf Ausgängen, der Summe von A und B sowie dem Übertrag aus der Addition, entwerfen

oder

> die Addition im BCD-Code auf die Addition von zwei vier Bit breiten Dualzahlen zurückführen und das Ergebnis später geeignet korrigieren.

Wir wollen hier den zweiten Weg gehen. Die Addition der BCD-Ziffern A und B lässt sich offenbar leicht mit Hilfe des aus der letzten Aufgabe bekannten Addierers (natürlich hier ohne die Logik zur Subtraktion) durchführen. Man erhält dann ein Ergebnis $S = s_4s_3s_2s_1$ und den Übertrag C_{out}. Diese Addition führt aber nicht in jedem Fall wieder auf eine gültige BCD-Ziffer S (z.B. 0101 + 0101 = 1010) oder sie liefert eine falsche BCD-Ziffer S (z.B. 1001 + 1000 = 0001). In beiden Fällen müssen also Korrekturmaßnahmen durchgeführt werden. Weiter muss dann ein BCD-Übertrag U auf die nächste Dezimalstelle generiert werden.

a) Bestimmen Sie U als logische Funktion von C_{out} und S. Beachten Sie dabei, dass es eine ganze Reihe von unmöglichen Bitkombinationen gibt.

Tragen Sie hier die Funktion, die Sie gefunden haben, ein:

$U=$

Liefert Sie für die Eingaben 0101 + 0100, 0101 + 0101 und 1001 + 1001 das richtige Ergebnis?

Wenn ja, simulieren Sie den BCD-Addierer und ihre Schaltung für den Dezimal-Übertrag U mit HADES und testen Sie ihn noch einmal für verschiedene Werte von A und B.

Achtung: Verwenden Sie hier und im Teil **b)** für die Ein- und Ausgänge der Schaltung nur IPINs und OPINs und geben Sie ihnen aussagekräftige Namen (mit der rechten Maustaste auf das Symbol klicken und dann *Edit* wählen). Auch den Eingang Cin des Addierers sollten Sie auf einen IPIN legen.

Es bleibt das Problem der Korrektur des Ergebnisses S. Offenbar ist eine Korrektur nur im Fall $U = 1$ notwendig. In diesem Fall müsste man $10_{10} = 1010_2$ von S subtrahieren, um die richtige Ziffer zu erhalten. Diese Subtraktion lässt sich aber auf eine Addition zurückführen und mit einem weiteren Addierers durchführen. Welche Zahl muss man nämlich zu S addieren, um auf das richtige Ergebnis zu kommen?

....

Skizzieren Sie hier Ihre Schaltung:

b) Erweitern Sie ihre HADES-Schaltung aus Teil a) entsprechend, so dass sie für jedes A und B das richtige Ergebnis und den richtigen Übertrag U liefert.

c) Wir haben jetzt eine Schaltung, mit der sich zwei BCD-Ziffern addieren lassen. Wenn man mit größeren Zahlen arbeiten möchte, lässt sich dieses Teilergebnis ausnutzen.

Speichern Sie dazu zuerst Ihr Design ab. Gehen Sie dann in das HADES-Menue *Edit* und wählen Sie den Punkt *Create Symbol* aus. Dadurch wird ein rechteckiges Symbol erzeugt, das sich in andere Schaltungen einbauen lässt.

Dies wollen wir jetzt tun und eine Schaltung entwerfen, mit der sich zweistellige BCD-Zahlen addieren lassen. Öffnen Sie eine neue Schaltung. Mit *rechte Maustaste –> create –> Create subdesign –> Name des gerade abgespeicherten Designs*[2] lässt sich jetzt das gerade erzeugte Symbol in die Schaltung einbauen. Da wir mit zweistelligen Zahlen arbeiten wollen, brauchen wir zwei solcher BCD-Addierer. Als Eingänge können Sie hier HEX-Switches nehmen, als Ausgänge HEX-Displays. Ansonsten brauchen Sie nur noch einen IPIN für das Cin des einen Addierers und einen OPIN für das Cout des anderen.

Verdrahten Sie ihre Schaltung und testen Sie sie!

Aufgabe gelöst: Gruppe: .... TeilnehmerIn: ....
vom Betreuer auszufüllen

[2] Wählen Sie dazu nicht die Datei mit der Extension .sym, sondern die mit der Extension .hds.

Nützliche Formeln

Übliche logische Funktionen

	and	or	nand	nor	exor
a b	$a \wedge b$	$a \vee b$	$\overline{a \wedge b}$	$\overline{a \vee b}$	$a \oplus b$
0 0	0	0	1	1	0
0 1	0	1	1	0	1
1 0	0	1	1	0	1
1 1	1	1	0	0	0

Regeln von de Morgan

$$
\begin{aligned}
\overline{a} \wedge \overline{b} &= \overline{a \vee b} \\
a \wedge b &= \overline{\overline{a} \vee \overline{b}} \\
\overline{a} \vee \overline{b} &= \overline{a \wedge b} \\
a \vee b &= \overline{\overline{a} \wedge \overline{b}}
\end{aligned}
$$

Weitere Formeln

$$
\begin{aligned}
\overline{a} &= \overline{a \wedge a} \\
&= \overline{a \wedge \mathbf{1}} \\
&= a \oplus \mathbf{1}
\end{aligned}
$$

$$
\begin{aligned}
a \oplus b &= \overline{a} b \vee a \overline{b} \\
a \oplus x = b &\Rightarrow x = a \oplus b
\end{aligned}
$$