Musterlösungen

zu den Aufgaben der Nachklausur zum

Kurs 1701 „Grundlagen der Technischen Informatik“

und

Kurs 1707 „Technische Informatik I“

im Sommersemester 2000

vom 16.09.2000

Zu Aufgabe 1: Schaltnetz analysieren

a) Gesucht war die Schaltfunktion des Ausgangs *S*.

$$S = abc \vee (a \vee b \vee c) \wedge \overline{ab \vee bc \vee ac}$$

b) Die Schaltfunktion S aus a) sollte nun durch algebraische Umformung in eine DNF umgewandelt werden. Diese Umformung ist durch Anwendung der Regeln von de Morgan und durch „Ausmultiplizieren“ zu erreichen.

$$\begin{aligned}
S &= abc \vee (a \vee b \vee c) \vee (\overline{ab} \vee \overline{bc} \vee \overline{ac}) \\
&= abc \vee a\,\overline{ab}\,\overline{bc}\,\overline{ac} \vee \\
&\qquad b\,\overline{ab}\,\overline{bc}\,\overline{ac} \vee \\
&\qquad c\,\overline{ab}\,\overline{bc}\,\overline{ac} \\
&= abc \vee a(\bar{a} \vee \bar{b})(\bar{b} \vee \bar{c})(\bar{a} \vee \bar{c}) \vee \\
&\qquad b(\bar{a} \vee \bar{b})(\bar{b} \vee \bar{c})(\bar{a} \vee \bar{c}) \vee \\
&\qquad c(\bar{a} \vee \bar{b})(\bar{b} \vee \bar{c})(\bar{a} \vee \bar{c}) \\
&= abc \vee (a\bar{a} \vee a\bar{b})(a\bar{b} \vee a\bar{c})(a\bar{a} \vee a\bar{c}) \vee \\
&\qquad (b\bar{a} \vee b\bar{b})(b\bar{b} \vee b\bar{c})(b\bar{a} \vee b\bar{c}) \vee \\
&\qquad (c\bar{a} \vee c\bar{b})(c\bar{b} \vee c\bar{c})(c\bar{a} \vee c\bar{c}) \\
&= abc \vee (a\bar{b})(a\bar{b} \vee a\bar{c})(a\bar{c}) \vee \\
&\qquad (\bar{a}b)(b\bar{c})(\bar{a}b \vee b\bar{c}) \vee \\
&\qquad (\bar{a}c \vee \bar{b}c)(\bar{b}c)(\bar{a}c) \\
&= abc \vee (a\bar{b} \vee a\,\bar{b}\,\bar{c})a\bar{c} \vee \\
&\qquad (\bar{a}b\bar{c} \wedge ((\bar{a}b \vee b\bar{c})) \vee \\
&\qquad (\bar{a}\,\bar{b}\,c \vee \bar{b}c)\bar{a}c \\
S_{DNF} &= abc \vee a\,\bar{b}\,\bar{c} \vee \bar{a}b\bar{c} \vee \bar{a}\,\bar{b}\,c
\end{aligned}$$

c) Hier sollte algebraisch nachgewiesen werden, daß die Übertragsfunktion $\ddot{U}_{DNF}$ eines Volladdierers durch den Ausgang C^+ des Schaltnetzes realisiert wird. Dazu benötigt man eine eindeutige Darstellung einer booleschen Funktion, wie sie etwa durch eine KDNF gegeben wird. Eine KDNF läßt sich durch eine Wertetabelle sowie durch Erweitern aller Terme einer DNF um die jeweils im Produktterm fehlenden Kombinationen von Literalen und deren Negationen erzeugen.

Aus einer Wertetabelle bzw. einfacher Überlegung ergibt sich für $\ddot{U}_{DNF}$:

$$\ddot{U}_{DNF} = \bar{a}bc \vee a\bar{b}c \vee ab\bar{c} \vee abc$$

(Übertrag = 1, wenn mindestens zwei Eingänge 1 sind).

Aus dem Schaltbild erhält man für C^+:

$$
\begin{aligned}
C^+ &= ab \vee bc \vee ac \\
C^+_{DNF} &= abc \vee ab\bar{c} \vee \\
&\quad abc \vee \bar{a}bc \vee \\
&\quad abc \vee a\bar{b}c \\
&= abc \vee ab\bar{c} \vee \bar{a}bc \vee a\bar{b}c
\end{aligned}
$$

Durch Umsortieren der Produktterme von $\ddot{U}_{DNF}$ ist gezeigt, daß $\ddot{U}_{DNF} = C^+_{DNF} = C^+$ gilt, also $\ddot{U}_{DNF}$ durch C^+ realisiert wird.

Zu Aufgabe 2: Schaltnetz optimieren

a) Es sollte eine Schaltung entworfen werden, die einen Binärcode in einen „Spielwürfel-Code“ umwandelt, ähnlich einer BCD-Siebensegment-Dekodierschaltung.

Würfelzahl	Dualzahl			LED(s)							
	d2	d1	d0		L1	L2	L3	L4	L5	L6	L7
1	0	0	0	4	0	0	0	1	0	0	0
2	0	0	1	1,7	1	0	0	0	0	0	1
3	0	1	0	1,4,7	1	0	0	1	0	0	1
4	0	1	1	1,3,5,7	1	0	1	0	1	0	1
5	1	0	0	1,3,4,5,7	1	0	1	1	1	0	1
6	1	0	1	1,2,3,5,6,7	1	1	1	0	1	1	1
	1	1	0	beliebig	x	x	x	x	x	x	x
	1	1	1		x	x	x	x	x	x	x

b) Die beiden letzten Zeilen der Tabelle von Aufgabe 2a) sind nicht spezifiziert worden und sind somit *Don’t Cares*. Diese können zur Reduzierung der Komplexität ausgenutzt werden, wenn man davon ausgeht, daß diese Kombinationen nicht an den Eingängen erscheinen. Allgemein nutzt man partiell definierte boolesche Funktionen auf diese Weise zur Einsparung von Gattern.

$$L_1 = d_2 \vee d_1 \vee d_0$$

$$L_2 = d_2 \wedge d_0 \quad \text{hier wird ein don't care für } d_1 \text{ benutzt}$$

$$L_3 = d_1 d_0 \vee d_2$$

L_3 d0

d1

0	0	1	0
x	x	x	x

d2

$L_4 = \overline{d_0}$

L_4	d0		
		d1	
1	0	0	1
1	0	x	x

d2

$L_5 = L_3$

$L_6 = L_2$

$L_7 = L_1$

Zu Aufgabe 3: Schaltwerk analysieren

a) Da die Eingangsvariable X direkt auf den Ausgang Y und den Zustand wirkt, handelt es sich um einen übergangsorientierten Mealy-Automaten.

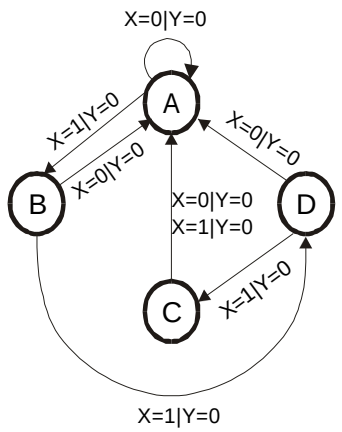
b) In den Klausurexemplaren fehlte die Beschriftung *D2* am zweiten D-FlipFlop.

$$Y(t) = X(t) \wedge Q_1(t) \wedge \overline{Q_2(t)}$$
$$Q_1(t+1) = D1 = X(t) \wedge Q_2(t)$$
$$Q_2(t+1) = D2 = X(t) \wedge \overline{Q_1(t)}$$

c) Aus den Übergangs- und Ausgangsgleichungen läßt sich eine (Zustands-)Übergangstabelle wie folgt erzeugen:

$X(t)$	$Q_1(t)$	$Q_2(t)$		$Q_1(t+1)$	$Q_2(t+1)$		$Y(t)$
0	0	0	A	0	0	A	0
0	0	1	B	0	0	A	0
0	1	0	C	0	0	A	0
0	1	1	D	0	0	A	0
1	0	0	A	0	1	B	0
1	0	1	B	1	1	D	0
1	1	0	C	0	0	A	1
1	1	1	D	1	0	C	0

d)



Zu Aufgabe 4: Schaltwerk entwerfen

Entscheidend war in dieser Aufgabe, daß ein *Modulo-6-Zähler* entworfen werden sollte. Damit sind nur die Zustände 0…5 definiert, aber nicht die Zustände 6 und 7. Diese können daher als don't care betrachtet werden.

a)

Q	Q^+	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Schon bei den Ansteuergleichungen des FlipFlops ist es wichtig, don't cares zu benutzen, um ein minimales Schaltwerk zu erhalten !

b) In dieser Übergangstabelle werden nun beide Don't care-Möglichkeiten ausgenutzt: zum einen die don't cares für die Zustände 6 und 7 und zum anderen die don't cares der Ansteuergleichungen des J-K-FF. Somit kann dann in den zugehörigen KV-Diagrammen die optimale Belegung mit Einsen erlangt werden.

Q_2	Q_1	Q_0	Q_2^+	Q_1^+	Q_0^+	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	0	0	0	X	1	0	X	X	1
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X	X

J_0

$Q_2 \backslash Q_1 Q_0$	00	01	11	10
0	1	x	x	1
1	1	x	x	x

K_0

$Q_2 \backslash Q_1 Q_0$	00	01	11	10
0	x	1	1	x
1	x	1	x	x

J_1

$Q_2 \backslash Q_1 Q_0$	00	01	11	10
0	0	1	x	x
1	0	0	x	x

K_1

$Q_2 \backslash Q_1 Q_0$	00	01	11	10
0	x	x	1	0
1	x	x	x	x

J_2

$Q_2 \backslash Q_1 Q_0$	00	01	11	10
0	0	0	1	0
1	x	x	x	x

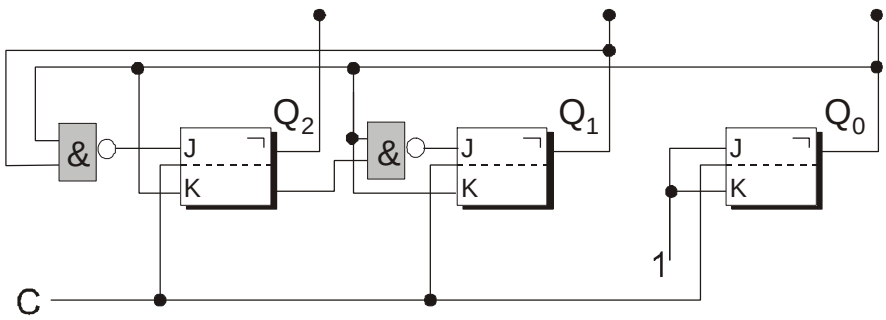
K_2

$Q_2 \backslash Q_1 Q_0$	00	01	11	10
0	x	x	x	x
1	0	1	x	x

c)

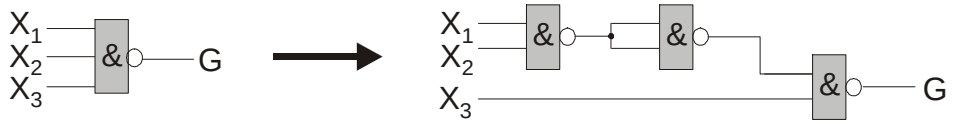
$$J_0 = 1 \qquad K_0 = 1$$
$$J_1 = Q_0 \wedge \overline{Q_2} \qquad K_1 = Q_0$$
$$J_2 = Q_0 \wedge Q_1 \qquad K_2 = Q_0$$

d)



Zu Aufgabe 5: Boole'sche Funktionen (Kurs 1701)

i) $G(X_1, X_2, X_3) = \overline{X_1 \wedge X_2 \wedge X_3} = \overline{\overline{\overline{(X_1 \wedge X_2)} \wedge X_3}}$



ii) $F(X_1, X_2, X_3, X_4) = \overline{\overline{\overline{X_1 X_3 X_4} \wedge \overline{X_1 \overline{X_2}} \wedge \overline{\overline{X_1 \overline{X_2}} \wedge \overline{X_4}}}}$

iii)

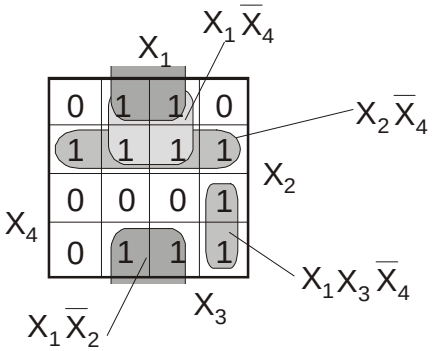
$$F(X_1, X_2, X_3, X_4) = \overline{\overline{\overline{X_1 X_3 X_4} \wedge \overline{X_1 \overline{X_2}} \wedge \overline{\overline{X_1 \overline{X_2}} \wedge \overline{X_4}}}}$$

$$= \overline{X_1} X_3 X_4 \vee X_1 \overline{X_2} \vee (\overline{\overline{X_1 \overline{X_2}}} \wedge \overline{X_4}) \qquad \text{De Morgan}$$

$$= \overline{X_1} X_3 X_4 \vee X_1 \overline{X_2} \vee ((X_1 \vee X_2) \wedge \overline{X_4}) \qquad \text{De Morgan}$$

$$= \overline{X_1} X_3 X_4 \vee X_1 \overline{X_2} \vee X_1 \overline{X_4} \vee X_2 \overline{X_4} \qquad \text{Distributivgesetz}$$

iv)

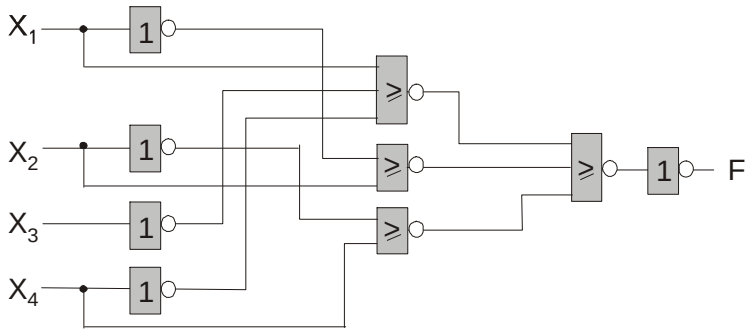


Der Term $X_1 \overline{X_4}$ ist überflüssig, da er von den Termen $X_1 \overline{X_2}$ und $X_2 \overline{X_4}$ überdeckt wird. Daraus folgt für die vereinfachte Funktion:

$$F(X_1, X_2, X_3, X_4) = \overline{X_1} X_3 X_4 \vee X_1 \overline{X_2} \vee X_2 \overline{X_4}$$

v)

$$F(X_1, X_2, X_3, X_4) = \overline{X_1} X_3 X_4 \vee X_1 \overline{X_2} \vee X_2 \overline{X_4} = \overline{\overline{\overline{(X_1 \vee \overline{X_3} \vee \overline{X_4})} \vee \overline{(\overline{X_1} \vee X_2)} \vee \overline{(\overline{X_2} \vee X_4)}}}$$



Zu Aufgabe 5: Mikroprozessor-Grundlagen (Kurs 1707)

a)

i)

Nr.	Befehl			Bedeutung
1		LD	R0,#$0010	R0 := 16
2		LD	R1,(ADR)	R1 := $C7AB
3		CLR	R2	R2 := 0
4	L:	RCL	R1	R1 um ein Bit nach links durchs Carry rotieren
5		BCC	M	Sprung nach M, falls CF=0
6		INC	R2	R2 := R2 + 1, falls CF=1
7	M:	DEC	R0	R0 := R0 - 1
8		BNE	L	Rücksprung, falls R0 $\neq$ 0
9		RCL	R1	R1 um ein Bit nach links durchs Carry rotieren
10		AND	R2,#$0001	R2 := R2 $\wedge$ 1, alle Bits außer LSB maskieren
11		ST	R2,(ADR+1)	R2 abspeichern

ii) R1 wurde 17mal durch das Carry Flag (links herum) rotiert. Da R1 16 bit lang ist, steht danach wieder der vorgegebene Wert in R1, d.h. R1 = $C7AB. In R2 werden die ‚1'-Bits von R1 gezählt, also ihre Anzahl berechnet. Hier gilt: R2 = 10 = $A.

iii) Durch den 10. Befehl wird nur das LSB von R2 ausgewählt. Dies gibt an, ob die Anzahl der ‚1'-Bits von R2 gerade oder ungerade ist. In ADR+1 steht also das (gerade) Paritätsbit des Wertes aus ADR. Dieser wurde durch das Programm nicht verändert, so daß in Speicherzelle ADR der Anfangswert $C7AB liegt.

b) 1. PUSH R 2. RTS

Durch den PUSH-Befehl wird der Inhalt von R, also die Sprungzieladresse, auf den Stack ausgelagert. Der nachfolgende RTS-Befehl liest diesen Wert in den Programmzähler und verzweigt – ohne weitere Stackoperation - zu der eingeladenen Adresse. Daher entspricht die Befehlsfolge einem Register-indirekten Sprung zu der in R stehenden Adresse.

c)

i) Aus $Z_{32} = (-1)^{VZ} * 2^{C} * (1.M)_2 \Rightarrow Z = (-1)^{VZ} * 2^{C-127} * (1.M)_2 = (-1)^{VZ} * 2^{E} * (1.M)_2$

folgt $Z_{64} = (-1)^{VZ} * 2^{C'} * (1.M)_2 = (-1)^{VZ} * 2^{E+1023} * (1.M)_2$

$= (-1)^{VZ} * 2^{C-127+1023} * (1.M)_2 = (-1)^{VZ} * 2^{C-128+1024} * (1.M)_2$

Von der Charakteristik der 32-bit-Zahl werden also 127 abgezogen und 1023 dazugezählt und das Ergebnis als 11-bit-Charakteristik der 64-bit-Zahl dargestellt. Bei Zahlen mit positivem Exponenten ist es leichter, 128 abzuziehen und 1024 zu addieren.

Beispiel: Z_{32} = $C3B4 0000 = $- 2^{7+128} *(1.01101)_2 \Rightarrow$

$Z_{64} = - 2^{(7+128)-128+1024} *(1.01101)_2 = - 2^{7+1024} *(1.01101)_2$

= $C076 8000 0000 0000.

Für die Wertigkeit des niederstwertigen Bit gilt: LSB = $1/2^{15}$.

Damit folgt: $- (0.d_1....d_{15} + 0.\bar{d}_1.....\bar{d}_{15} + LSB) = -1$ und somit

$$Z = - 0.d_1....d_{15} = -1 + 0.\bar{d}_1.....\bar{d}_{15} + LSB.$$

Da Z<0 sind nicht alle $d_i = 0$, also $\bar{d}_i = 1$. Die Addition von LSB kann also keinen Übertrag vor den Dezimalpunkt verursachen. Die Zahl im verlangten 1.15-Format lautet also:

$$Z' = 1.s_1.... s_{15},$$

wobei $s_1.... s_{15}$ durch bitweise Inversion der d_i und Addition von LSB entsteht.

Beispiel: Z = - 0.101101000000000 (Vorzeichen und Betrag)

$\Rightarrow$ Z' = 1.010010111111111 + 0.000000000000001 = 1.010011000000000.