Fortgeschrittenenpraktikum WS 1996/97
Protokoll zum Versuch „Digitalelektronik“

Kay Römer Frank Pilhofer

24. Februar 1997

Zusammenfassung

In diesem Versuch werden verschiedene Schaltwerke und Schaltnetze entworfen, aufgebaut und untersucht.

1 Grundlagen

1.1 Boolesche Algebra

Digitale Schaltungen werden aus den drei Grundfunktionen *Und*, *Oder* und *Negation* zusammengesetzt. Die *Boolesche Algebra* beschreibt die Rechenregeln, die für diese drei Grundoperationen gelten, durch die folgenden Axiome:

Kommutativität 1	$x_1x_2 = x_2x_1$
Kommutativität 2	$x_1 + x_2 = x_2 + x_1$
Assoziativität 1	$x_1(x_2x_3) = (x_1x_2)x_3$
Assoziativität 2	$x_1 + (x_2 + x_3) = (x_1 + x_2) + x_3$
Distributivität 1	$x_1(x_2 + x_3) = x_1x_2 + x_1x_3$
Distributivität 2	$x_1 + x_2x_3 = (x_1 + x_2)(x_1 + x_3)$
Absorption	$x_1(x_1 + x_2) = x_1 + x_1x_2 = x_1$
Idempotenz	$xx = x + x = x$
Negation 1	$x\overline{x} = 0$
Negation 2	$x + \overline{x} = 1$
Negation 3	$\overline{\overline{x}} = x$
De Morgan 1	$\overline{x_1x_2} = \overline{x_1} + \overline{x_2}$
De Morgan 2	$\overline{x_1 + x_2} = \overline{x_1} \cdot \overline{x_2}$
Operationen mit 0 und 1	$x \cdot 1 = x$
	$x \cdot 0 = 0$
	$x + 0 = x$
	$x + 1 = 1$
	$\overline{0} = 1$
	$\overline{1} = 0$

Normalformen

Für ein und die selbe boolesche Funktion gibt es beliebig viele Möglichkeiten, sie als Formel der booleschen Algebra auszudrücken. Beispielsweise kann die Funktion $x \oplus y$ als $x\overline{y} + \overline{x}y$ oder als $(x + y)(\overline{x} + \overline{y})$ geschrieben werden. Um zu entscheiden, ob zwei Formeln die gleiche boolesche Funktion beschreiben, hat man mehrere Möglichkeiten:

- Aufstellen und Vergleichen der Wertetabellen für beide Formeln. Diese Methode eignet sich nur für Funktionen in wenigen Variablen, da die Zahl der zu untersuchenden Fälle exponentiell mit der Zahl der Variablen wächst.
- Umformung der einen Formel in die andere mittels der Regeln der booleschen Algebra. Für einfache Formeln kann man so die Identität zweier Funktionen nachweisen, jedoch ist es für nicht identische Funktionen sehr schwierig zu zeigen, das die eine Formel mittels der Regeln der booleschen Algebra nicht in die andere umgeformt werden kann.
- Darstellung beider Formeln in einer *kanonischen Normalform*

Bei letztgenannter Methode werden die Formeln in einer standardisierten Form (einer sogenannten Normalform) ausgedrückt und minimiert, d.h. alle überflüssigen Operationen werden gestrichen. Zwei boolesche Formeln sind genau dann gleich, wenn ihre minimierten Dartstellungen in einer Normalform gleich sind. Es sind unter anderem folgende Normalformen bekannt:

- Disjunktive Normalform (Disjunktion von Und-Klauseln)
- Konjunktive Normalform (Konjunktion von Oder-Klauseln)
- Reed-Muller-Normalform (Exklusives Oder von Und-Klauseln, die keine negierten Literale enthalten)

Die Wahl einer Normalform hängt von der Art der zu untersuchenden Formeln ab, beispielsweise hat ein exklusives Oder aus N Variablen sowohl in der disjunktiven als auch in der konjunktiven Normalfrom 2^N Terme, in der Reed-Muller-Form dagegen nur N. Dafür hat ein Oder aus N Variablen in der Reed-Muller-Normalform exponentiell viele Terme.

Ein AND aus EOR's?

Oftmals stellt sich die Frage, ob man mit einer bestimmten Art von Gattern eine bestimmte boolesche Funktion realisieren kann. In der Praxis ist es beispielsweise wünschenswert, mit nur einem Gattertyp alle möglichen booleschen Funktionen aufbauen zu können. Ein Beispiel für solch ein „Allround-Gatter“ ist das NAND wegen

$$\begin{aligned} \overline{x} &= \mathrm{NAND}(x,1) \\ xy &= \mathrm{NAND}(\mathrm{NAND}(x,y),1) \\ x+y &= \mathrm{NAND}(\overline{x},\overline{y}) \end{aligned}$$

Aber wie steht es zum Beispiel mit einem AND aus EOR's? Nachdem man eine Weile herumprobiert hat, wird man feststellen, daß dies nicht möglich ist. Aber wie kann man das *beweisen*? Eine Möglichkeit ist, die Mengen der booleschen Funktionen zu betrachten, die mit keinem, einem, zwei, usw. EOR Gattern erzeugt werden können. Nennen wir diese Mengen $W_0, W_1, \ldots$. W_0 enthält die

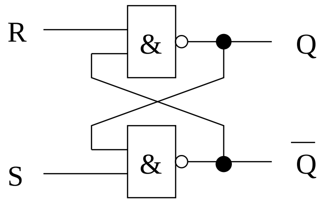

Abbildung 1: Basis Flip-Flop

Eingangsvariablen und die beiden konstanten Funktionen 0 und 1. Man errechnet nun W_{i+1} aus W_i, indem man zunächst $W_{i+1} := W_i$ setzt und dann für alle $u, v \in W_i$ $u \oplus v$ zu W_{i+1} hinzufügt, also

$$\begin{aligned} W_0 &:= \{0, 1, x, y\} \\ W_{i+1} &:= W_i \cup \{u \oplus v | u, v \in W_i\} \end{aligned}$$

Berechnet man nun sukzessive die W_i, so wird es irgendwann ein j mit $W_j = W_{j+1} = W_{j+2} = \cdots =: W_\infty$ geben, da es für N Eingagsvariablen nur 2^{2^N} verschiedene boolesche Funktionen gibt. Ist dann $xy \in W_\infty$, so kann man ein AND aus EOR's bauen, ansonsten nicht.

Berechnet man die W_i für EOR, so stellt man fest, daß $W_2 = W_3 = W_\infty = \{0, 1, x, y, \overline{x}, \overline{y}, x \oplus y, \overline{x \oplus y}\}$ ist, d.h. man kann kein AND aus EOR's bauen.

1.2 Schaltnetze vs. Schaltwerke

Digitale Schaltungen verfügen über eine Menge von Eingängen $x_1, x_2, \ldots$ und eine Menge von Ausgängen $y_1, y_2, \ldots$.

Bei *Schaltnetzen* hängen die Werte der Ausgänge y_i nur von den momentanen Werten der Eingänge x_i ab. Alle aus den drei Grundoperationen $x \cdot y$, $x + y$ und $\overline{x}$ ohne Rückkopplung aufgebauten Schaltungen sind Schaltnetze.

Bei *Schaltwerken* hängen die Ausgangswerte nicht nur von den momentaten Werten an den Eingängen, sondern darüber hinaus auch von den Werten der Ein- und Ausgänge zu früheren Zeitpunkten ab. D.h. Schaltwerke sind in der Lage, Informationen zu speichern.

Ein Vertreter von Schaltwerken sind die sogenannten Flip-Flops – Kippschaltungen mit zwei stabilen Zuständen.

Basis-Flip-Flop

Abbildung 1 zeigt ein aus zwei NAND-Gattern aufgebautes Basis Flip-Flop. Zur Ermittlung der Schaltfunktion wird der Wert von Q ermittelt, dann werden R und S angelegt und die neuen Werte für Q^+ und $\overline{Q}^+$ festgestellt:

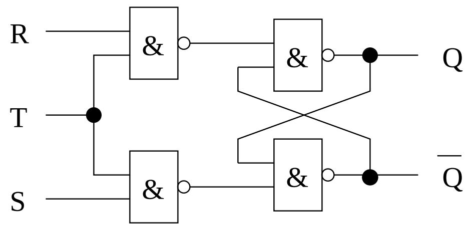

Abbildung 2: Basis Flip-Flop mit Takteingang

S	R	Q	Q^+	$\overline{Q}^+$
0	0	0	1	1
0	0	1	1	1
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	0	1
1	1	1	1	0

Man erkennt, daß bei $R = 1, S = 0$ Q^+ unabhängig von Q auf 0 gesetzt wird, man spricht daher auch von *Reset*. Bei $R = 0, S = 1$ wird Q^+ in jedem Fall auf 1 gesetzt (*Set*). $R = S = 1$ bewirkt, daß $Q^+ = Q$, d.h. der alte Wert beibehalten oder gespeichert wird.

Bei allen diesen Fällen sind die Ausgänge Q und $\overline{Q}$ komplementär. Im letzten Fall, $R = S = 0$, werden beide Ausgänge auf 1 gesetzt. Werden danach beide Eingänge gleichzeitig auf 0 gelegt, so hängt es vom Zufall [1] ab, in welchen stabilen Zustand das Flip-Flop dann fällt ($Q = 0, \overline{Q} = 1$ oder $Q = 1, \overline{Q} = 0$). Da dieser Nichtdeterminismus nicht wünschenswert ist, spricht man für $R = S = 0$ auch vom „verbotenen Zustand“. Die verkürzte Wertetabelle für das Basis Flip-Flop aus NAND-Gattern lautet damit:

S	R	Q^+	Bedeutung
0	0	1	verboten
0	1	0	reset
1	0	1	set
1	1	Q	speichern

Flip-Flop mit Takteingang

Das bisher betrachtete Basis Flip-Flop ist als Speicher nicht geeignet, da es die Informationen an den Eingängen sofort übernimmt. Man führt daher ein Taktsignal T ein, daß dem Flip-Flop zusätzlich zugeführt wird. Nur während $T = 1$ werden die Informationen am Eingang übernommen, ansonsten wird das Flip-Flop auf „Speichern“ geschaltet. Abbildung 2 zeigt die Schaltung des so erweiterten Flip-Flops.

Um den „verbotenen Zustand“ auszuschließen, kann dieses Flip-Flop mit einem weiteren NAND-Gatter zu einem D-Flip-Flop erweitert werden. Anstelle

[1] mit Zufall sind hier die Charakteristika der Gatter, die Kabellängen, die kosmische Hintergrundstrahlung und die Stellung des Mondes gemeint

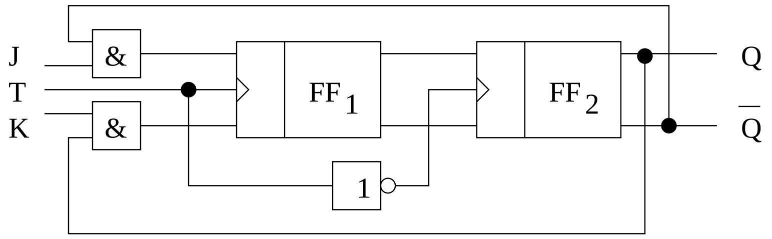

Abbildung 3: JK Flip-Flop

von R und S hat dieses Flip-Flop nur noch einen D Eingang, der bei logisch Eins ein $S = 1, R = 0$ und bei logisch Null $S = 0, R = 1$ bewirkt.

JK Flip-Flop

Das Flip-Flop mit Takteingang hat noch den Nachteil, daß sich während $T = 1$ Änderungen an den Eingängen direkt auf die Ausgänge auswirken. Um das zu verhindern schaltet man zwei solcher Flip-Flops in Reihe, wobei die beiden Flip-Flops mit gegenphasigem Takt betrieben werden. Dadurch ist das Eingangs-Flip-Flop FF_1 während $T = 1$ „geöffnet" und übernimmt die Informationen an den Eingängen. Wird $T = 0$, so öffnet FF_2 und übernimmt die in FF_1 zwischengespeicherten Informationen.

Außerdem ist eine Über-Kreuz-Rückkopplung angebracht, die bei $J = K = 1$ bewirkt, daß das JK Flip-Flop abwechselnd Set und Reset ausführt. Da für Set und Reset jeweils ein Taktimpuls benötigt wird, hat man einen Takthalbierer, der als Grundbaustein für Zähler verwendet wird. Für die anderen möglichen Kombinationen von J und K verhält sich das JK Flip-Flop äquivalent zum RS Flip-Flop:

J	K	Q^+	Bedeutung
0	0	Q	speichern
0	1	0	reset
1	0	1	set
1	1	$\overline{Q}$	zählen

2 Versuchsdurchführung

2.1 Einfache TTL Schaltungen

Die untersuchten NAND (7400) und EOR (7486) Schaltkreise wiesen die erwarteten Schaltfunktionen auf, offene Eingänge werden wie logisch 1 interpretiert.

Ein EOR Gatter kann man sogar mit nur vier NAND's realisieren, wie folgende Umformungen zeigen:

$$\begin{aligned} x \oplus y &= x\overline{y} + \overline{x}y \\ &= x(\overline{x} + \overline{y}) + y(\overline{x} + \overline{y}) \\ &= x(\overline{xy}) + y(\overline{xy}) \end{aligned}$$

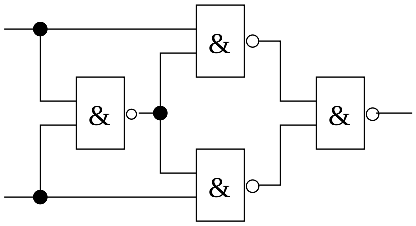

Abbildung 4: EOR aus vier NAND's

$$= \quad \overline{\overline{x(\overline{xy})} \cdot \overline{y(\overline{xy})}}$$

Das ergibt die in Abbildung 4 gezeigte Schaltung für ein EOR. Indem man an den Ausgang dieses EOR einfach ein NOT (realisierbar mit einem NAND) schaltet, erhält man ein $\overline{\text{EOR}}$ aus 5 Gattern.

Im folgenden sind die Realisierungen eines AND aus NAND's, eines AND aus NOR's und eines NOR aus NAND's angegeben.

$$\begin{aligned} \text{AND}(x,y) &= \text{NAND}(1, \text{NAND}(x,y)) \\ \text{AND}(x,y) &= \text{NOR}(\text{NOR}(\text{x},0), \text{NOR}(\text{y},0)) \\ \text{NOR}(x,y) &= \text{NAND}(1, \text{NAND}(\text{NAND}(1,x), \text{NAND}(1,y))) \end{aligned}$$

Ein NAND kann nicht aus AND's realisiert werden, da die Menge W_∞ der mit beliebig vielen AND Gattern erzeugbaren Funktionen gemäß Abschnitt 1.1 gleich $\{0, 1, x, y, xy\}$ ist und NAND nicht enthält. Ebensowenig kann man ein NAND aus EOR's erzeugen, da $\overline{xy} \notin W_\infty = \{0, 1, x, y, \overline{x}, \overline{y}, x \oplus y, \overline{x \oplus y}\}$.

2.2 Addierer

Ein N-Bit Addierer wird im einfachsten Fall aus N Volladdieren aufgebaut. Ein Volladdierer addiert drei Eingabebits (A, B, C_{in}) und liefert als Ausgabe Summe S und Übertrag C_{out}. Für die niederwertigste Stelle benötigt man nur einen Halbaddierer, der sich vom Volladdierer dadurch unterscheidet, daß ihm der Eingang C_{in} fehlt. Die Wertetabelle für einen Halbaddierer lautet:

A	B	S	C_{out}
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Man erkennt daran leicht, daß $S = A \oplus B$ und $C_{out} = AB$ ist. Aus zwei Halbaddieren und einem OR Gatter kann man gemäß Abbildung 5 einen Volladdierer bauen. Ein Volladdierer ist also mit fünf Gattern realisierbar.

Der Nachteil des aus Volladdieren aufgebauten einfachen Wellenübertragsaddierers liegt in der benötigten Rechenzeit, die proportional zur Anzahl zu addierender Bits ist, da ja jeder Volladdierer mit dem Berechnen so lange warten muß, bis der Vorgänger C_{out} berechnet hat.

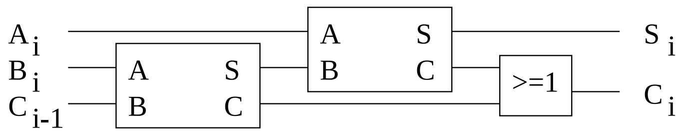

Abbildung 5: Volladdierer aus zwei Halbaddieren

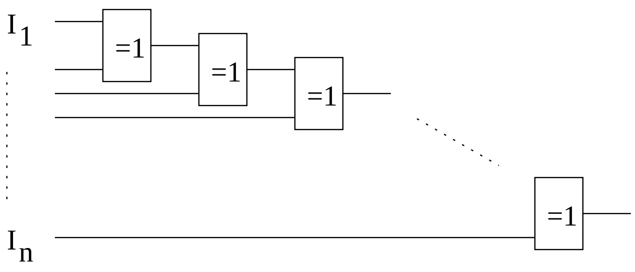

Abbildung 6: Parity Checker

2.3 Parity Checker

Einen N-Bit Parity Checker realisiert man durch ein N-fach EOR. Allerdings hat man in der Praxis meist nur EOR's mit zwei Eingängen. Daraus kann man durch kaskadieren von N-1 EOR's den in Abbildung 6 gezeigten Parity Checker aufbauen.

Eine solche Schaltung wird häufig bei der Datenübertragung verwendet. Sollen beispielsweise acht Bit parallel über eine unzuverlässige Leitung übertragen werden, so kann man vor der Übertragung die „Parity“ der acht Bit berechnen und über eine neunte Leitung mit übertragen. Auf der Gegenseite wird die Parity der empfangenen acht Bits errechnet und mit dem über die neunte Leitung übertragenen Wert verglichen. Stimmen die Werte nicht überein, so wurde eine ungeradzahlige Anzahl von Bits bei der Übertragung gestört, andernfalls geradzahlig viele.

Ist die Fehlerwahrscheinlichkeit der Leitung so klein, daß höchstens ein Bit bei der Übertragung gestört wird, so hat man mit diesem Aufbau einen einfachen Fehlererkennungsmechanismus: Stimmen berechnete und übertragene Parity beim Empfänger überein, so liegt kein Fehler vor, andernfalls ist ein Bit gestört worden.

2.4 Multiplexer und Demultiplexer

Multiplexer realisieren eine 1 aus N Auswahl, d.h. sie haben N Eingänge und einen Ausgang, wobei über $\log_2 N$ Steuereingänge festgelegt wird, welcher der Eingänge auf den Ausgang durchgeschaltet wird. Einen Multiplexer mit zwei Eingängen kann man wie in Abbildung 7 gezeigt aus vier Gattern realisieren.

Demultiplexer stellen das Gegenstück zum Multiplexer dar, d.h. sie haben einen Eingng und N Ausgänge, wobei über $\log_2 N$ Steuerleitungen festgelegt wird, auf welchen Ausgang der Eingang durchgeschaltet wird.

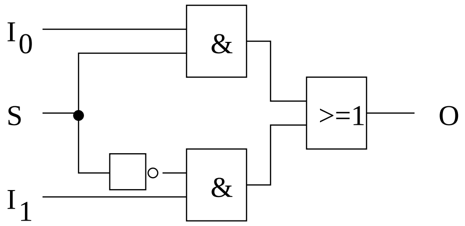

Abbildung 7: Zweifach-Multiplexer

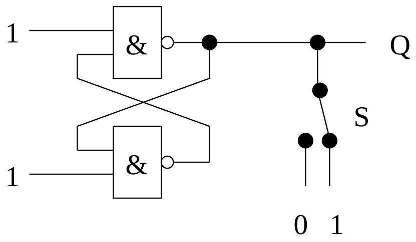

Abbildung 8: Entpreller

2.5 Flip-Flops

Zu Basis-, RS- und JK-Flip-Flop wurde bereits in den Grundlagen alles nötige gesagt. Das aus zwei RS-Flip-Flops aufgebaute Pseudo-JK-Flip-Flop entspricht im Prinzip einem vollwertigen JK-Flip-Flop, bei dem $J = K = 1$, d.h. der Zählmodus eingestellt ist. Man hat damit einen Takthalbierer geschaffen, der als Basis für Zähler und Schieberegister verwendet werden kann.

Die aus der Aufgabenstellung stammende, in Abbildung 8 gezeigte Schaltung zum Entprellen eines mechanischem Schalters entspricht im Grunde einem Basis-Flip-Flop im Speichermodus ($R = S = 1$). Der Schalter ist mit dem Q-Ausgang dieses Flip-Flops verbunden und erzwingt ein Set bzw. Reset des Flip-Flops beim Umschalten.

Allerdings bewirkt diese einfache Schaltung kein vollständiges Entprellen, wenn sich die durch das Prellen des Schalters hervorgerufene Schwingung des Spannungspegels um den Umschaltpunkt des Flip-Flops herum bewegt. Dann wird nämlich in schneller Folge ein Set bzw. Reset des Flip-Flops durchgeführt und der Ausgang prellt immernoch.

2.6 Asynchronzähler

Das kritische an diesem Versuch ist die Erzeugung von sauberen Eingangsimpulsen. Die im Abbildung 8 gezeigte Schaltung ist wegen in Abschnitt 2.5 genannten Gründen nicht ausreichend. Man erkennt das daran, daß der Zähler beim Umlegen des Schalters gleich mehrfach weiterzählt.

Wir haben daher ein D-Flip-Flop verwendet, dessen D und T Eingänge mit jeweils einem Schalter verbunden wurden. Um am Ausgang des Flip-Flops eine saubere 0-1 Flanke zu erhalten, wird zuerst $T = 0$, dann $D = 1$, dann $T = 1$ gesetzt. Ensprechend wird eine 1-0 Flanke durch $T = 0$, dann $D = 1$, dann $T = 1$ erreicht. Mit derart erzeugten Eingangsimpulsen zählt der Zähler dann

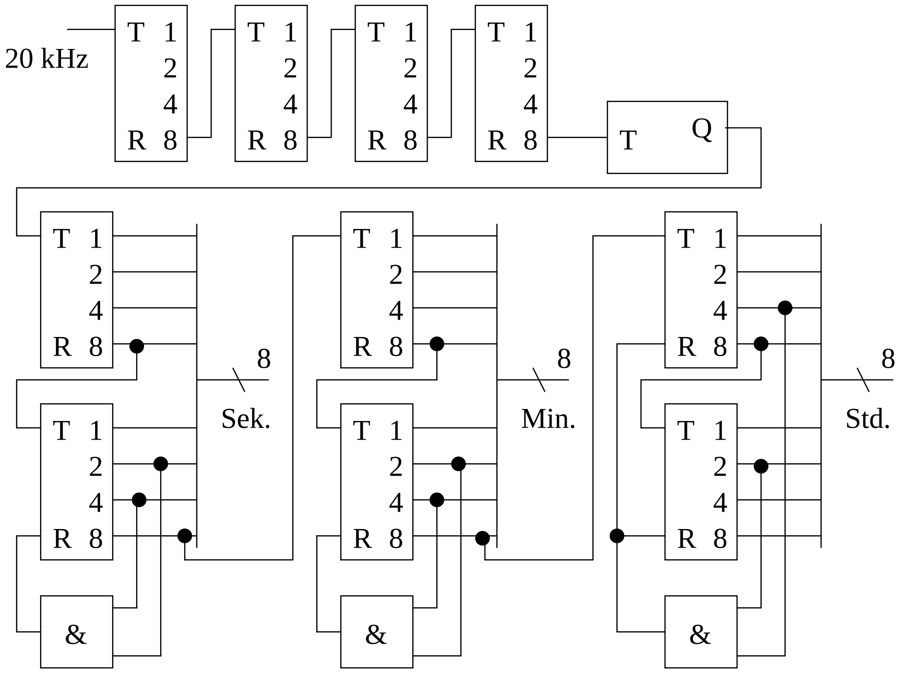

Abbildung 9: Uhr aus 10 BCD-Zählern, 1 JK-FF und 3 AND's

wirklich in Einerschritten.

Man kann aus dem Vortwärts- einen Rückwärtszählr machen, indem man die Takteingänge der JK-Flip-Flops nicht mit dem Q, sondern dem $\overline{Q}$ Ausgang der Vorgänger-Flip-Flops verbindet. Dadurch schalten die Flip-Flops bei steigender Flanke (bezogen auf den Q-Ausgang des Vorgängers), was gerade ein Rückwärtszählen bewirkt.

2.7 Uhr

Der zur Vefügung stehende Rechteckgenerator liefert eine Ausgangsfrequenz von 20 kHz. Um daraus einen 1 Hz Takt für die Sekunden zu erzeugen, muß als erstes ein bis-20000-Zähler aus vier BCD-Zählern und einem JK-FF gebaut werden.

Für die Sekunden und Minuten wird jeweils ein bis-60-Zähler benötigt, der jeweils aus 2 BCD-Zählern und einem AND aufgebaut werden kann. Für die Stunden braucht man noch einen bis-24-Zähler, der mit zwei BCD-Zählern und einem AND zu Buche schlägt. Abbildung 9 zeigt die Schaltung der Uhr.