Interner Bericht
DESY F 56-71/5
Dezember 1971

Elemente der Computer-Hardware
Band 5 - Kontrollfunktionen, Coder, Decoder

von
H.-J. Stuckenberg

1. Codieren

Codieren bedeutet das Verschlüsseln von Informationen, wobei wir hier unter Informationen Zahlen, Buchstaben und Sonderzeichen, allgemein alphanumerische Zeichen verstehen wollen.

Da wir es gewohnt sind, im Dezimalcode zu zählen oder zu rechnen, die Rechner aber intern sehr oft binär arbeiten, ist es für uns wichtig, die Codierung dezimal → binär und umgekehrt ausführen zu können. Die Zahlen von 0 bis 9, d.h. eine Dekade, können wir durch einen 4 Bit-Code darstellen, wie es die folgende Tabelle zeigt:

Dezimal	Binär
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Sollen mehrere Dekaden codiert werden, kann man entweder rein binär verschlüsseln oder mehrere Gruppen des 4 Bit-Codes als BCD-Code verwenden. Praktischerweise wird man die Codierung von Dezimalzahlen erst in BCD-Zahlen, dann anschließend in Binärzahlen ausführen. Der zweite Teil der Umwandlung ist etwas komplizierter, er wird in Abschnitt 3 beschrieben.

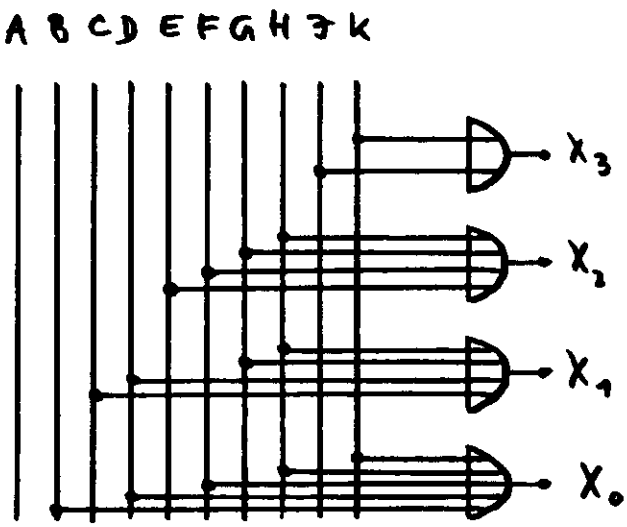
Lösen wir die Dezimalzahlen von 0 bis 9 in 10 einzelne Leitungen auf, können wir aus einer Wertetabelle leicht die Gleichungen eines Codierer für die Wandlung von Dezimal- in BCD-Zahlen auslesen. Wir nennen die 10 Leitungen A, B, ...J, K, die 4 Ausgangsleitungen für den BCD-Code X_0, ... X_3, dann lautet die Tabelle:

Eingangsvariable										Ausgangsvariable			
A	B	C	D	E	F	G	H	J	K	X_3	X_2	X_1	X_o
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	-	-	-	-	-	0	0	0	0	1
0	0	1	0	-	-	-	-	-	0	0	0	1	0
0	0	0	1	0	-	-	-	-	0	0	0	1	1
0	-	0	0	1	0	-	-	-	0	0	1	0	0
0	-	-	-	0	1	0	-	-	0	0	1	0	1
0	-	-	-	-	0	1	0	-	0	0	1	1	0
0	-	-	-	-	-	0	1	0	0	0	1	1	1
0	-	-	-	-	-	-	0	1	0	1	0	0	0
0	-	-	-	-	-	-	-	0	1	1	0	0	1

Hieraus die Gleichungen:

$$X_o = B + D + F + H + K$$
$$X_1 = C + D + G + H$$
$$X_2 = E + F + G + H$$
$$X_3 = J + K$$

Diese vier ODER-Bedingungen können folgendermaßen erzeugt werden (vgl. Bild 5.1):

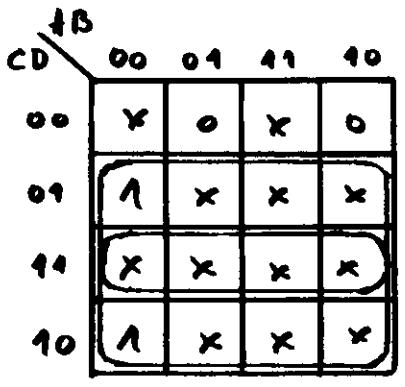


5.1

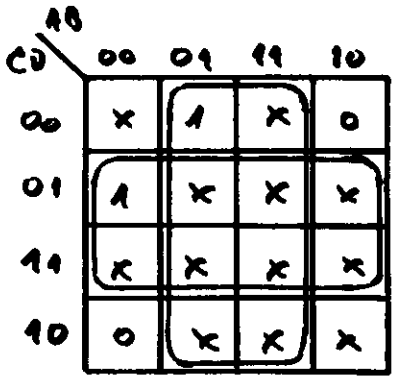
Der hier dargestellte Code ist als 8421-Code bekannt, X_3 hat die Wertigkeit 8, X_2 die Wertigkeit 4 usw.
Man kann zeigen, daß diese aufwendige Wertetabelle vereinfacht geschrieben werden kann. Dazu ein Beispiel: Denken wir uns, es sollen die Zahlen 0 bis 3 binär codiert werden. Dann lautet die ausgeschriebene (aufwendige) Tabelle

A B C D	X_1 X_0
1 0 0 0	0 0
0 1 0 0	0 1
0 0 1 0	1 0
0 0 0 1	1 1

Schreibt man für X_1 und X_0 das Karnaugh-Diagramm und minimiert, erhält man:



$X_1 = C + D$



$X_0 = B + D$

5.2

d.h. X_1 ist dort gleich 1, wo C oder D eine 1 ist und
X_0 ist dort gleich 1, wo B oder D eine 1 ist.
Das ist aber auch der direkt abgelesene Inhalt der Tabelle. Diese Vereinfachung gilt für alle Tabellen, in denen in einer Zeile von allen Eingangsvariablen immer nur eine gleich 1 ist. Die vereinfachte Tabelle lautet dann:

Eingangs-variable	X_1 X_0
A	0 0
B	0 1
C	1 0
D	1 1

→ $X_1 = C + D$
$X_0 = B + D$

Es gibt viele BCD-Codes, in denen die Wertigkeiten der Ausgangsleitungen X_0 bis X_3 unterschiedlich sind. Z.B. können die Leitungen im 2421-Code organisiert sein, d.h. X_3 hat die Wertigkeit 2, X_2 die 4, X_3 die 2 und X_1 die 1.

Die verkürzte Tabelle lautet also:

Eingangsvariablen	X_3	X_2	X_1	X_o
0 = A	0	0	0	0
1 = B	0	0	0	1
2 = C	0	0	1	0
3 = D	0	0	1	1
4 = E	0	1	0	0
5 = F	1	0	1	1
6 = G	1	1	0	0
7 = H	1	1	0	1
8 = J	1	1	1	0
9 = K	1	1	1	1

Die Zahl 4_{10}, d.h. E wurde als $X_2 = 1$ geschrieben. Dieser 2421-Code, auch Aiken-Code genannt, ist ein symmetrischer Code, da die Bitkombinationen 0 und 9 durch Inversion jedes Bits hervorgehen, ebenso die Kombinationen 1 und 8, 2 und 7, allgemein jede Ergänzung zu 9. Dies ist bei arithmetischen Operationen oft sehr wesentlich (vgl. Band 2, Abschnitt 3.2.1). Die Gleichungen des 2421-Aiken-Codierers lauten:

$$X_o = B + D + F + H + K$$
$$X_1 = C + D + F + J + K$$
$$X_2 = E + G + H + J + K$$
$$X_3 = F + G + H + J + K$$

Natürlich kann man die Zahl 4_{10} auch durch $2_{10} + 2_{10}$ aufbauen, man erhält dann den unsymmetrischen 2421-Code, dessen Tabelle sich wie folgt ergibt:

Eingangsvariablen	X_3	X_2	X_1	X_o
0 = A	0	0	0	0
1 = B	0	0	0	1
2 = C	0	0	1	0
3 = D	0	0	1	1
4 = E	1	0	1	0
5 = F	1	0	1	1
6 = G	1	1	0	0
7 = H	1	1	0	1
8 = J	1	1	1	0
9 = K	1	1	1	1

Die Gleichungen des unsymmetrischen 2424-Codes lauten:

$$X_o = B + D + F + H + K$$
$$X_1 = C + D + E + F + J + K$$
$$X_2 = G + H + J + K$$
$$X_3 = E + F + G + H + J + K$$

Ein häufig in arithmetischen Einheiten benutzter Code ist der Exzess 3-Code, auch XS-3 abgekürzt. Dieser symmetrische Code ergibt sich aus folgender Tabelle:

Eingangsvariablen	X_3	X_2	X_1	X_o
0 = A	0	0	1	1
1 = B	0	1	0	0
2 = C	0	1	0	1
3 = D	0	1	1	0
4 = E	0	1	1	1
5 = F	1	0	0	0
6 = G	1	0	0	1
7 = H	1	0	1	0
8 = J	1	0	1	1
9 = K	1	1	0	0

Die vier Ausgangsleitungen haben die Wertigkeit (Dezimalzahl + 3). Die Gleichungen des Codierers vom Dezimal in den XS-3-Code sind:

$$X_o = A + C + E + G + J$$
$$X_1 = A + D + E + H + J$$
$$X_2 = B + C + D + E + K$$
$$X_3 = F + G + H + J + K$$

Ein BCD-Code für Dezimalziffern ist immer eine Auswahl von 10 Kombinationen aus insgesamt $2^4 = 16$ vorhandenen. Es gibt also $\frac{16!}{6!}$ Möglichkeiten, einen 4 Bit-BCD-Code aufzubauen, insgesamt über 10^{10} Codes, von denen nur wenige, wie z.B. die erwähnten Bedeutung erlangten.

2. Decodieren

Während Codieren Verschlüsseln bedeutet, entspricht Decodieren der Entschlüsselung. Z.B. sollen die in einem Rechner ermittelten Binärzahlen nach einer Binär → BCD- und BCD → Dezimalwandlung ausgegeben bzw. angezeigt werden. Wir brauchen dafür Schaltungen, die die im vorigen Abschnitt beschriebene Dezimal-BCD-Wandlung wieder rückgängig machen, d.h. in die z.B. 4 Leitungen im BCD-Code hineingehen und 10 Leitungen dezimal herauskommen. Die aufwendige Wertetabelle für diese Decodierung lautet dann:

A B C D	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9
0 0 0 0	1	0	0	–	–	–	–	–	–	0
0 0 0 1	0	1	0	–	–	–	–	–	–	0
0 0 1 0	0	0	1	0	–	–	–	–	–	0
0 0 1 1	0	0	0	1	0	–	–	–	–	0
0 1 0 0	0	–	0	0	1	0	–	–	–	0
0 1 0 1	0	–	–	–	0	1	0	–	–	0
0 1 1 0	0	–	–	–	–	0	1	0	–	0
0 1 1 1	0	–	–	–	–	–	0	1	0	0
1 0 0 0	0	–	–	–	–	–	–	0	1	0
1 0 0 1	0	–	–	–	–	–	–	–	0	1

Daraus ergeben sich die Gleichungen für die decodierten Größen:

$$X_0 = \bar{A}\,\bar{B}\,\bar{C}\,\bar{D}$$
$$X_1 = \bar{A}\,\bar{B}\,\bar{C}\,D$$
$$X_2 = \bar{A}\,\bar{B}\,C\,\bar{D}$$
$$X_3 = \bar{A}\,\bar{B}\,C\,D$$
$$X_4 = \bar{A}\,B\,\bar{C}\,\bar{D}$$
$$X_5 = \bar{A}\,B\,\bar{C}\,D$$
$$X_6 = \bar{A}\,B\,C\,\bar{D}$$
$$X_7 = \bar{A}\,B\,C\,D$$
$$X_8 = A\,\bar{B}\,\bar{C}\,\bar{D}$$
$$X_9 = A\,\bar{B}\,\bar{C}\,D$$

Während man für die Codierung ODER-Gates benötigte, muß man zur Decodierung UND-Gates einsetzen, in unserem Fall 10 Vierfach-UND-Gates.

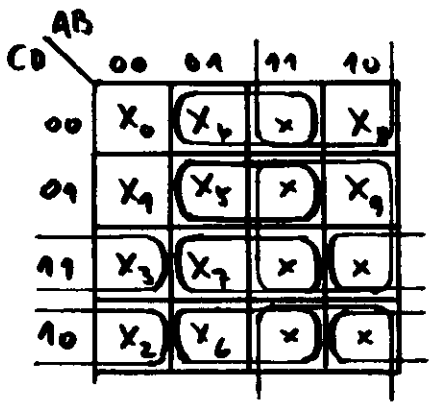
Auch hier ist eine vereinfachte Tabelle möglich, da auf jeder Ausgangsleitung nur zu einer Eingangskombination eine 1 erscheint. Die reduzierte Tabelle lautet also:

A B C D	Ausgangsvariablen
0 0 0 0	X_o
0 0 0 1	X_1
0 0 1 0	X_2
.	.
.	.
.	.
0 1 1 1	X_7
1 0 0 0	X_8
1 0 0 1	X_9

Daraus folgen die Gleichungen für X_o bis X_9:

$$X_o = \bar{A}\,\bar{B}\,\bar{C}\,\bar{D}$$
$$X_1 = \bar{A}\,\bar{B}\,\bar{C}\,D$$
$$X_2 = \bar{A}\,\bar{B}\,C\,\bar{D}$$
$$\vdots$$
$$X_7 = \bar{A}\,B\,C\,D$$
$$X_8 = A\,\bar{B}\,\bar{C}\,\bar{D}$$
$$X_9 = A\,\bar{B}\,\bar{C}\,D$$

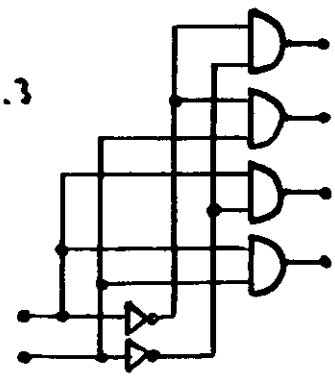
Diese Gleichungen können im Karnaugh-Diagramm minimiert werden:



$$X_o = \bar{A}\,\bar{B}\,\bar{C}\,\bar{D}$$
$$X_1 = \bar{A}\,\bar{B}\,\bar{C}\,D$$
$$X_2 = \bar{B}\,C\,\bar{D}$$
$$X_3 = \bar{B}\,C\,D$$
$$X_4 = B\,\bar{C}\,\bar{D}$$
$$X_5 = B\,\bar{C}\,D$$
$$X_6 = B\,C\,\bar{D}$$
$$X_7 = B\,C\,D$$
$$X_8 = A\,\bar{D}$$
$$X_9 = A\,D$$

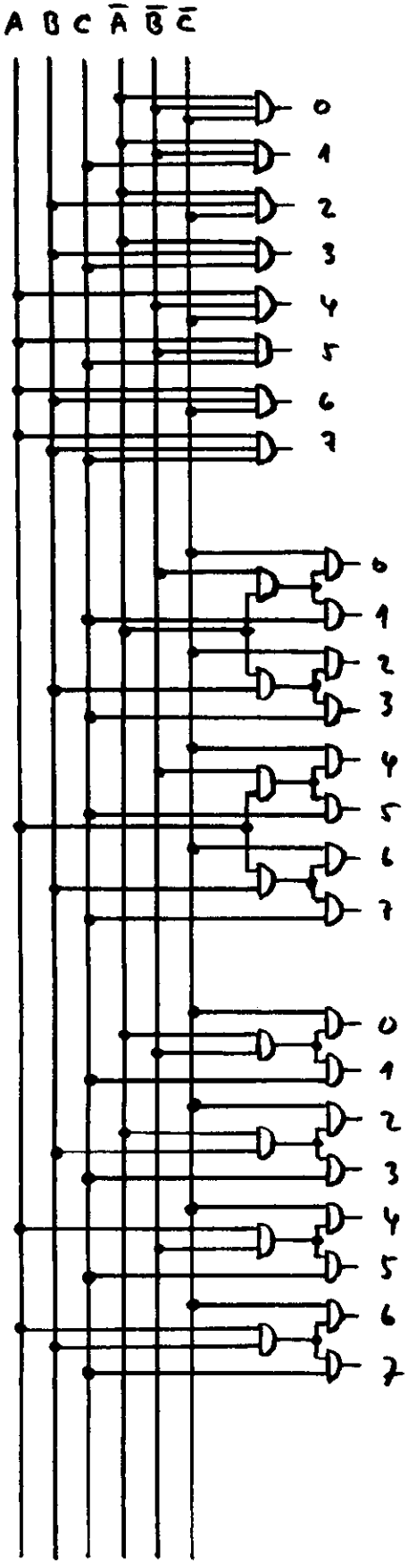
Eine solche Decodierung von BCD-Zahlen auf Dezimalzahlen nennt man einen "1 aus 10"-Schaltkreis, da aus den 10 Leitungen eine ausgewählt wird, die dann eine logische 1 annimmt. Allgemeinere Schaltungen, die nicht nur "1 aus 10" heraussuchen, sondern "1 aus N", werden in den meisten Fällen durch binäre Decoder realisiert.
Dies ist ein Schaltkreis mit n Eingangsleitungen und 2^n Ausgangsleitungen, die so angeordnet sind, daß für jeden möglichen der 2^n-Zustände der Eingangsleitungen eine Ausgangsleitung auf 1 liegt, während alle anderen eine 0 haben.
Nehmen wir z.B. n = 2, d.h. die Eingangsleitungen A und B, so können diese (vgl. Bild 5.3 und Tabelle) vier verschiedene Zustände annehmen. Das Ergebnis ist ein 1 aus 4 Decoder.

5.3



A B	Ausgangsvariablen
0 0	$\bar{A}\ \bar{B}$
0 1	$\bar{A}\ B$
1 0	$A\ \bar{B}$
1 1	$A\ B$

Für größere n-Werte kann die Zahl der Eingangsleitungen pro Gate unwirtschaftlich hoch werden. Dann können die Decoder anders aufgebaut werden. Eine Anordnung wie die in Bild 5.3, wo für jeden Ausgang ein und nur ein UND-Gate benutzt wird, nennt man Ein-Ebenen-Decoder. Ihr Vorteil ist die kurze Schaltzeit, da nur ein Gate durchlaufen wird; ihr Nachteil, daß die Zahl der Gateeingänge mit zunehmendem n steigt. Decoder können auch als Bäume aufgebaut werden, wobei Einfach-Bäume oder die noch billigeren Zweifach-Bäume benutzt werden. Bild 5.4 zeigt die drei Varianten für drei Eingangsvariablen.



Typ 1 - Decoder (Ein-Ebenen-Dec.)

5.4

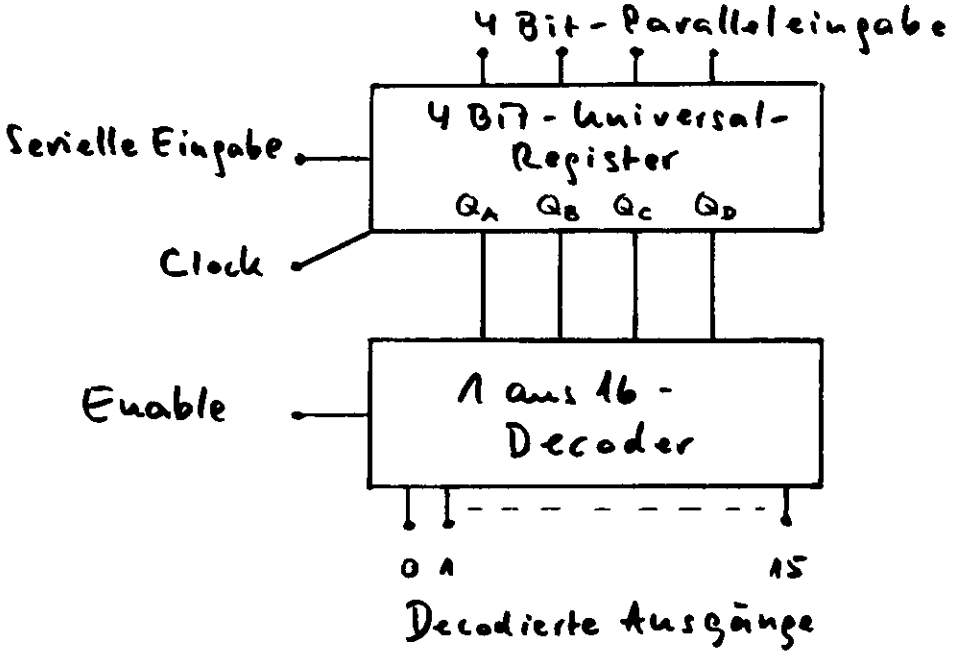
Typ 2 -Decoder (Einfach-Baum)

Typ 3 - Decoder (Zweifach-Baum)

Zahl der Eingänge n	Zahl der Ebenen		
	Typ 1	Typ 2	Typ 3
2	1	1	1
3	1	2	2
4	1	3	2
5	1	4	3
6	1	5	3
7	1	6	3
8	1	7	3

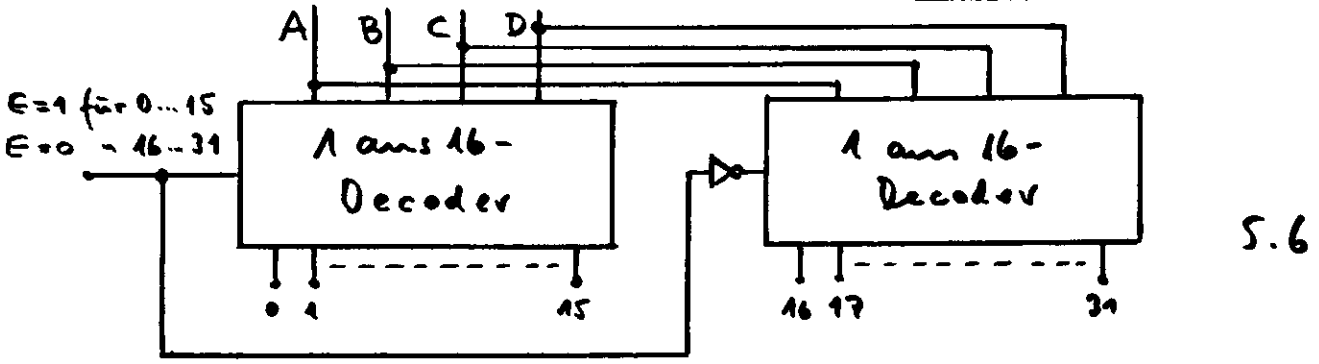
Bei den meisten integrierten Decodern haben die Ausgangsgates noch eine zusätzliche gemeinsame Eingangsleitung, die als Enable(Freigabe) bezeichnet wird. Liegt diese Leitung auf 1, können die decodierten Ausgangsleitungen auch auf 1 gehen, sonst bleiben sie gesperrt.

Das Aufrufen der zu decodierenden Leitungen geschieht oft mit einem zusätzlichen Speicherregister, wie es Bild 5.5 zeigt.



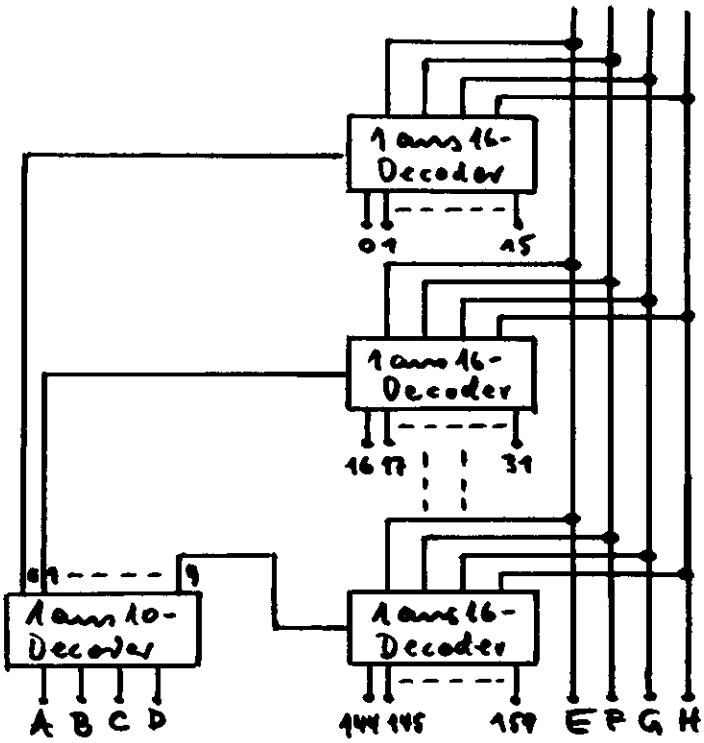
5.5

Falls mehr als die dargestellten 16 decodierten Ausgänge benötigt werden, kann man auch mehrere Decoder zusammenschalten. Soll z.B. ein 1 aus 32-Decoder gebaut werden, kann man nach den oben geschilderten Verfahren (Einebenen oder Bäume) vorgehen oder zwei 1 aus 16-Decoder verwenden, wie in Bild 5.6 dargestellt.



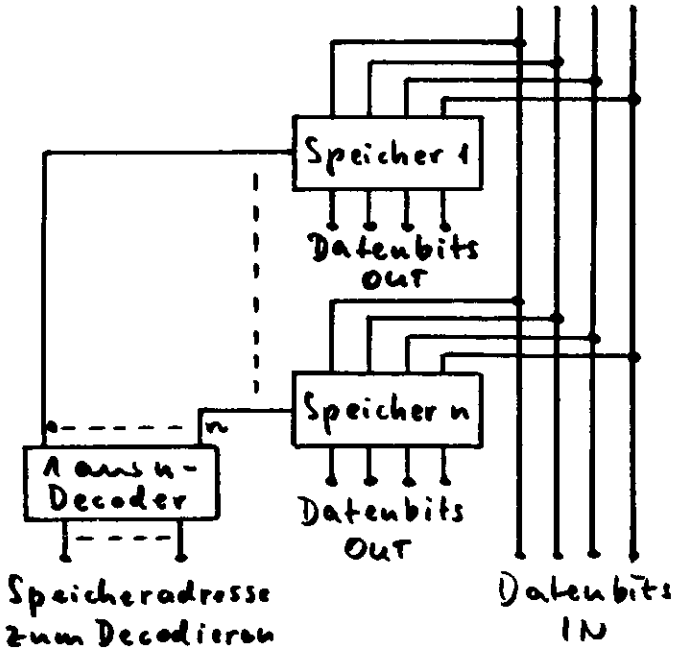
5.6

Bei einer großen Zahl von Eingangsleitungen kann auch die Enable-Leitung selbst über einen Decoder angesteuert werden. Das nächste Bild 5.7 zeigt einen 1 aus 160-Decoder, der aus zehn 1 aus 16- und einem 1 aus 10-Decoder besteht. Die 1 aus 16-Decoder werden mit 4 Bits parallel angesteuert, jeweils einer der 10 Enable-Eingänge von einem 1 aus 10-Decoder.



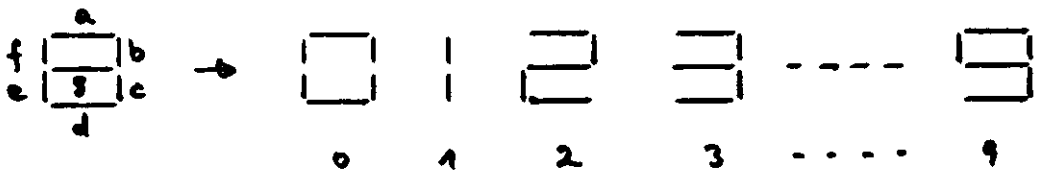
5.7

An die Stelle der 1 aus 16-Decoder in Bild 5.7 kann man in einem anderen Anwendungsbeispiel auch 4 Bit-Speicher setzen, deren Inhalt selektiv geladen bzw. abgeholt werden soll. Bild 5.8 zeigt das Prinzip.



5.8

Zur optischen Anzeige von Dezimalzahlen werden in zunehmendem Maße 7-Segment-Leuchten benutzt. Die darzustellenden Ziffern von 0 bis 9 werden aus 7 waagerechten oder senkrechten Balken zusammengesetzt, wie es Bild 5.9 zeigt:



5.9

Mit den Balkenbezeichnungen a bis g kann man die Wertetabelle für 7 Segment-Decoder von BCD auf Anzeige schreiben, aus der sich dann in gewohnter Weise die Gleichungen für a bis g ergeben.

Dezimalzahl	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	0	0	1	1

3. Binär $\overrightarrow{\leftarrow}$ B C D - Conversion

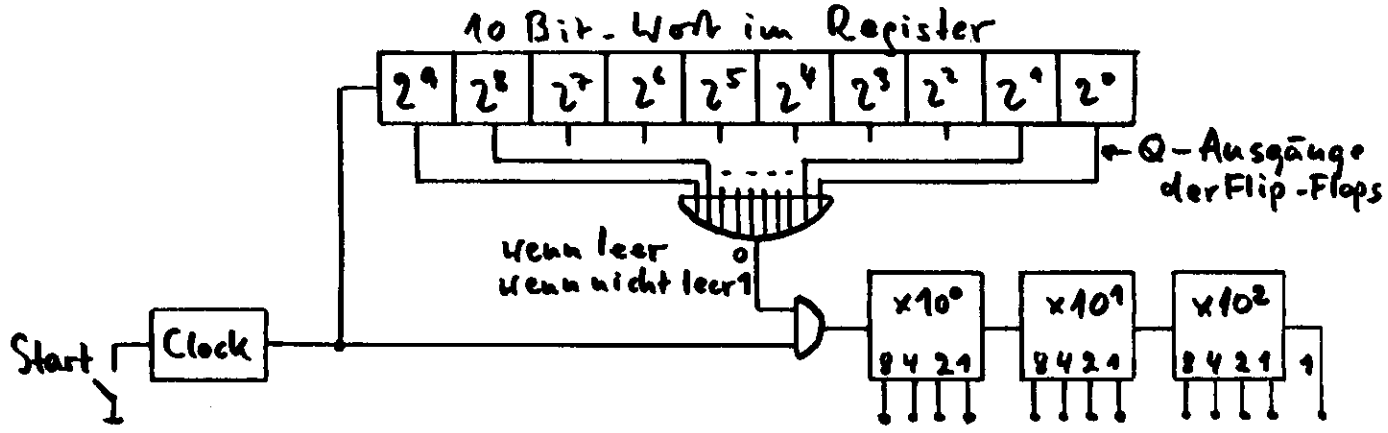
Es sind verschiedene Methoden entwickelt, die Binär $\overrightarrow{\leftarrow}$ BCD-Wandlung auszuführen; eine von ihnen, die sogenannte ADD-3-Conversion hat sich praktisch durchgesetzt, sie wird auf zweierlei Art durchgeführt, durch festverdrahtete Kombinatorik und durch Schieberegister.

Zusätzlich gibt es noch die ältere Zählervergleichsmethode, die zu Anfang beschrieben wird, sowie eine neuere, die die Umwandlung durch tabellierte Festwertspeicher vornimmt und die am Ende des Abschnitts behandelt wird.

3.1 Zählervergleichsmethode

3.1.1 Binär → BCD-Wandlung

Die Zählervergleichsmethode ist im Prinzip sehr einfach, die Conversion aber sehr zeitraubend. Man benötigt, wie Bild 5.10 zeigt, zwei Zähler, einen rückwärts-zählenden Binärzähler und einen vorwärts-zählenden BCD-Zähler.



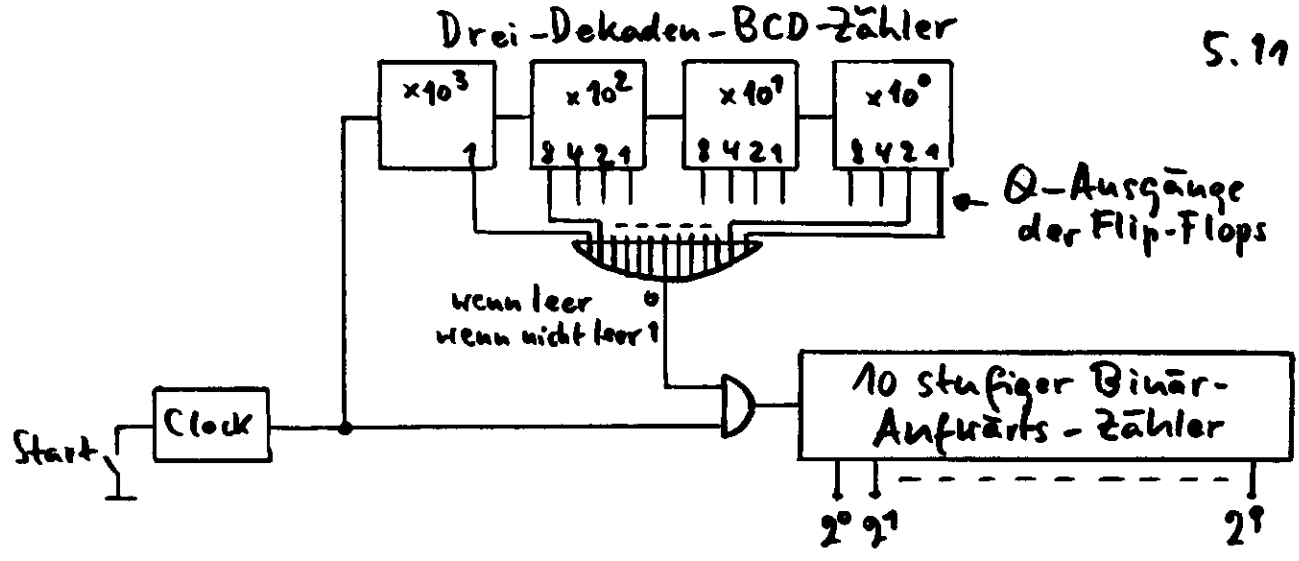
5.10

Die umzusetzende Binärzahl wird in einen binären Rückwärtszähler eingelesen, der anschließend von einem Clockgenerator leergezählt wird. Gleichzeitig werden die Clockpulse in einen BCD-Vorwärtszähler gegeben. Nach Leerung des Rückwärtszählers steht die Zahl, BCD-codiert, im Vorwärtszähler. Weitere Pulse des Clockgenerators sperrt des UND-Gate vor dem Vorwärtszähler durch eine 0 aus dem ODER-Gate. Der Zeitbedarf ergibt sich aus der Zahl der Clockpulse. Wählen wir z.B. eine Clockfrequenz von 10 MHz, d.h. einen Pulsabstand von 10^{-7} sec , so dauert die Conversion von

10 Bits: $1024 \cdot 10^{-7}$ sec = 102,4 µsec
16 Bits: $65384 \cdot 10^{-7}$ sec = 6.54 msec
24 Bits: $17.6 \cdot 10^{6} \cdot 10^{-7}$ sec = 1.76 sec

3.1.2 BCD → Binär - Wandlung

Mit einer ähnlichen Anordnung kann man auch die umgekehrte Wandlung ausführen. Man zählt, wie Bild 5.11 zeigt, den BCD-Rückwärtszähler leer und gleichzeitig den Inhalt in den Binäraufwärtszähler.



5.11

3.2 ADD-3-Methode mit festverdrahteter Kombinatorik

3.2.1 Binär → BCD-Wandlung

Die ADD-3-Methode beruht auf zwei wesentlichen Eigenschaften der binären Arithmetik (vgl. Band 2):

- die Multiplikation einer binären Zahl mit 2 erreicht man durch Schieben der Zahl um eine Stelle in Richtung auf das höchstwertige Bit. Multipliziert man z.B. die binäre Zahl 1000_2 (8_{10}) mit 2_{10}, erhält man 10000_2. Die angehängten Indizes geben den Zahlencode an, in dem die Zahl dargestellt ist.

- Sollen binäre Zahlen > 1001_2 (9_{10}) in BCD gewandelt werden, muß man eine 6_{10} addieren, wie man aus der folgenden Tabelle abliest, in der einige Zahlen in beiden Codes dargestellt sind (vgl. auch Band 2, Abschnitt 2.8)

Binär	BCD
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 0	0 0 1 0
.	.
.	.
1 0 0 0	1 0 0 0
1 0 0 1	1 0 0 1
1 0 1 0	1 0 0 0 0
1 0 1 1	1 0 0 0 1
1 1 0 0	1 0 0 1 0
.	.
.	.
1 1 1 0	1 0 1 0 0
1 1 1 1	1 0 1 0 1
1 0 0 0 0	1 0 1 1 0
1 0 0 0 1	1 0 1 1 1

Die Addition der 6_{10} kann man nach dem oben gesagten also auch durch eine Addition einer 3_{10} und einer anschließenden Multiplikation mit 2 ausführen, was technisch leichter zu realisieren ist.

Die Umwandlung vom Binärcode zum BCD-Code geschieht also durch Prüfung, ob die ersten 3 Bits größer als $5_{10} = 101_2$ sind. Ist das der Fall, wird eine 3_{10} addiert und mit 2_{10} multipliziert. Sind die ersten 3 Bits kleiner als 5_{10}, pas-

siert die ganze 4 Bit-Zahl ungeändert, da der Binärcode und der BCD-Code für Zahlen zwischen 0 und 9_{10} identisch ist.

Der Vorgang soll am Beispiel der Umwandlung der binären Zahl 10000 (16_{10}) nochmals verdeutlicht werden.

Schritte	Zehnerdekade	Einerdekade	Binäres Wort
			1 0 0 0 0
1. Schieben		1	0 0 0 0
2. Schieben		1 0	0 0 0
3. Schieben		1 0 0	0 0
4. Schieben		1 0 0 0	0
		Inhalt > 5_{10}	
5. ADD 3		1 0 1 1	0
6. Schieben	1	0 1 1 0	
	(1_{10})	(6_{10})	

BCD-Ausgang

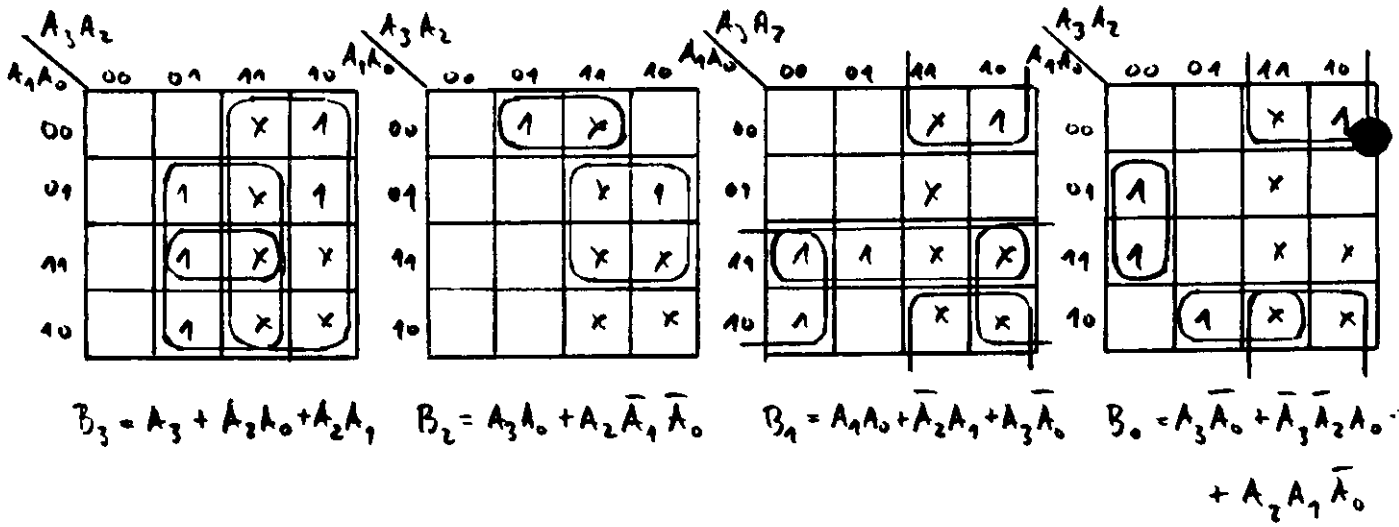
Für die ADD 3-Funktion kann man eine Wertetabelle aufstellen, aus der wir dann Gleichungen für die Kombinatorik auslesen können.

Eingangsvariablen				Ausgangsvariablen			
A_3	A_2	A_1	A_o	B_3	B_2	B_1	B_o
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

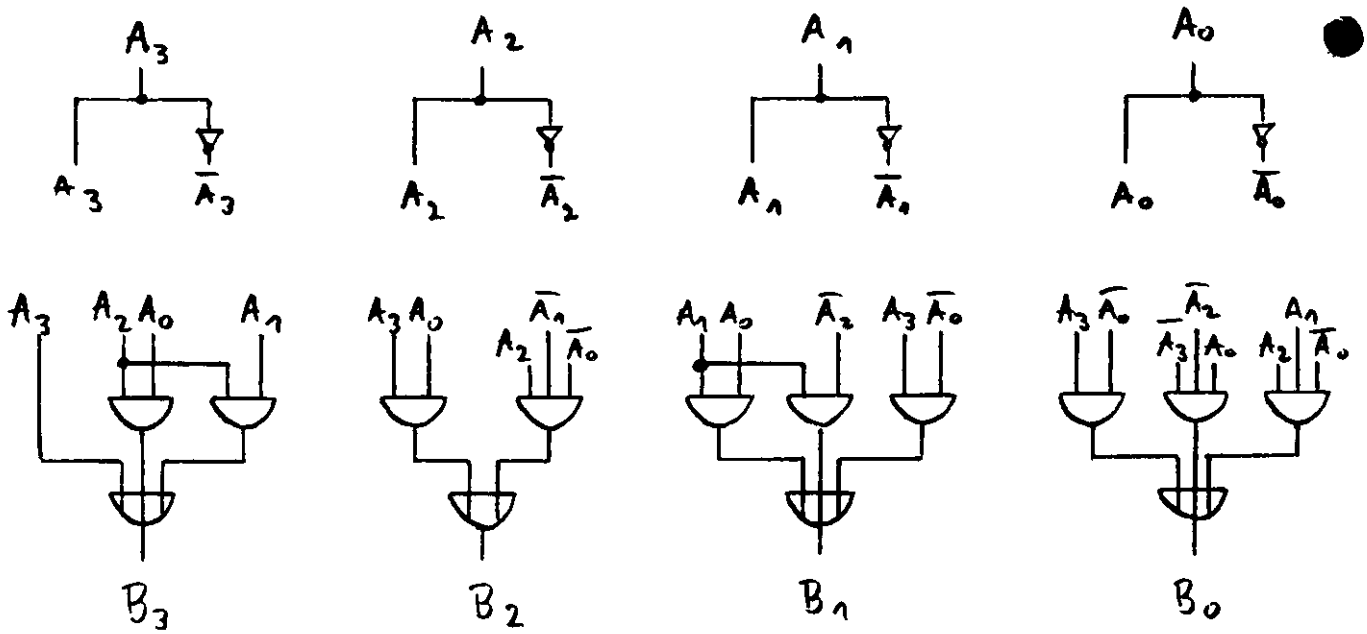
Die Gleichungen für die Gatefunktionen lauten abgekürzt (vgl. Band 1, Abschnitte 9 und 12.3):

$B_3 = \sum (5,6,7,8,9)$
$B_2 = \sum (4,9)$
$B_1 = \sum (2,3,7,8)$
$B_0 = \sum (1,3,6,8)$

Diese in Karnaugh-Diagramme eingetragen und minimiert:



Die ADD 3-Funktion kann nach folgender Schaltung festverdrahtet werden (Bild 5.12)

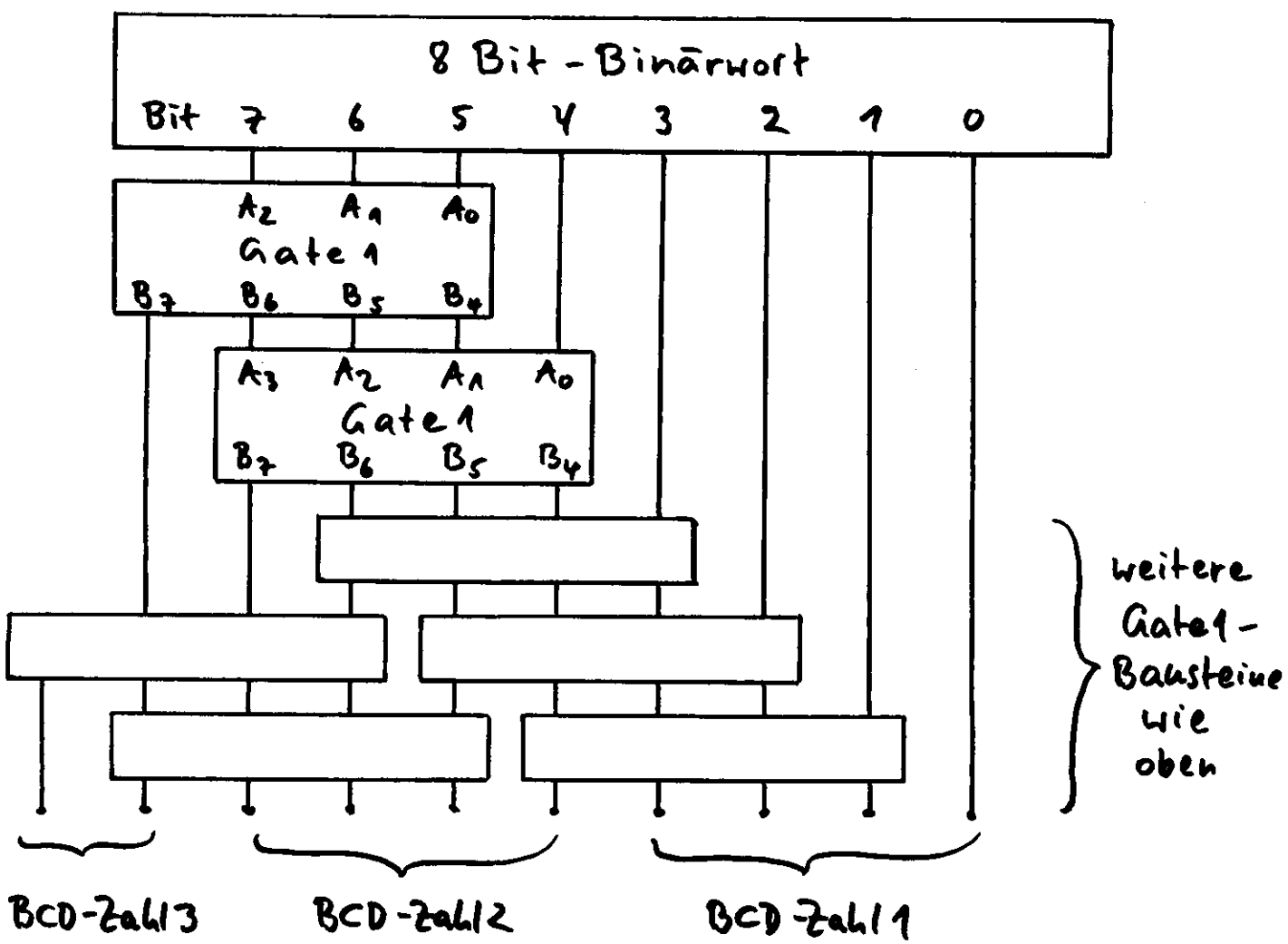


5.12

Diese Logik wird von manchen Firmen als Codierungsgate 1 geliefert, ein entsprechendes Gate 2 dient der BCD → Binär-Wandlung nach der ADD 3-Methode. Wenn man in einer Zahlentabelle die binären und die zugehörigen BCD-Zahlen vergleicht, stellt man drei Punkte fest:

- das letzte Bit in der binären Reihe ist identisch mit dem in der BCD-Reihe
- die Bits für die Dezimalzahlen 0 bis 9 sind in beiden Reihen identisch
- die Bits für die Dezimalzahlen > 9 sind in der BCD-Reihe um 6 größer als in der Binärreihe.

Daraus kann man die Prozedur der Umwandlung am Beispiel eines 8 Bit-Binärwortes wie folgt beschreiben (vgl. auch Bild 5.13):



5.13

Schritt 1
a) Prüfe die ersten 3 wichtigsten Bits (most significant bit = MSB)
b) Ist der Wert 5 oder größer? Wenn ja, ADD 3; wenn nein, Bits ungehindert passieren lassen.

Schritt 2
a) Prüfe die 3 weniger wichtigen Bits (least significant bit=LSB) der vorangegangenen Conversion zusammen mit dem nächsten MSB des binären Wortes.
b) Ist der Wert 5 oder größer? Wenn ja, ADD 3; wenn nein, Bits ungehindert passieren lassen.
c) Sind alle Bits außer dem letzten LSB geprüft? Wenn ja, Conversion beendet; wenn nein, gehe wieder zu Schritt 2 a).

Der Schritt 2 a) entspricht dem Schieben um eine Stelle nach rechts (Multiplikation), da das nächste MSB des Binärwortes benutzt wird. Das letzte Bit braucht nach dem oben gesagten nicht convertiert zu werden.

3.2.2 BCD → Binär-Wandlung

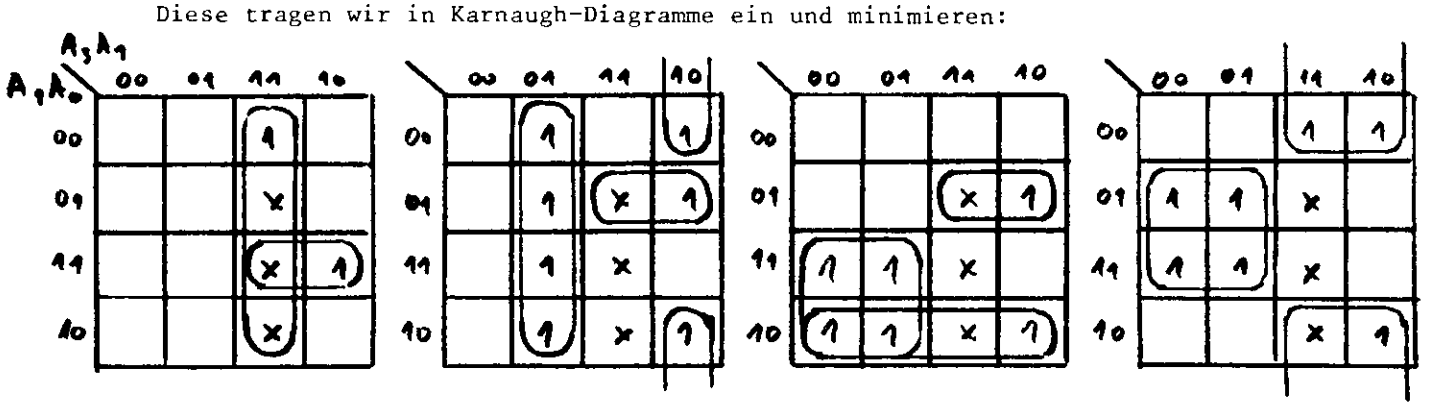
Für diese Conversion müssen wir nicht 3 x 2 addieren, sondern subtrahieren. Die Wertetabelle lautet also:

Eingangsvariablen				Ausgangsvariablen			
A_3	A_2	A_1	A_o	B_3	B_2	B_1	B_o
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	1
0	1	1	0	0	1	1	0
0	1	1	1	0	1	1	1
1	0	0	0	0	1	0	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1

Daraus folgen die Gleichungen für die Gatefunktion:

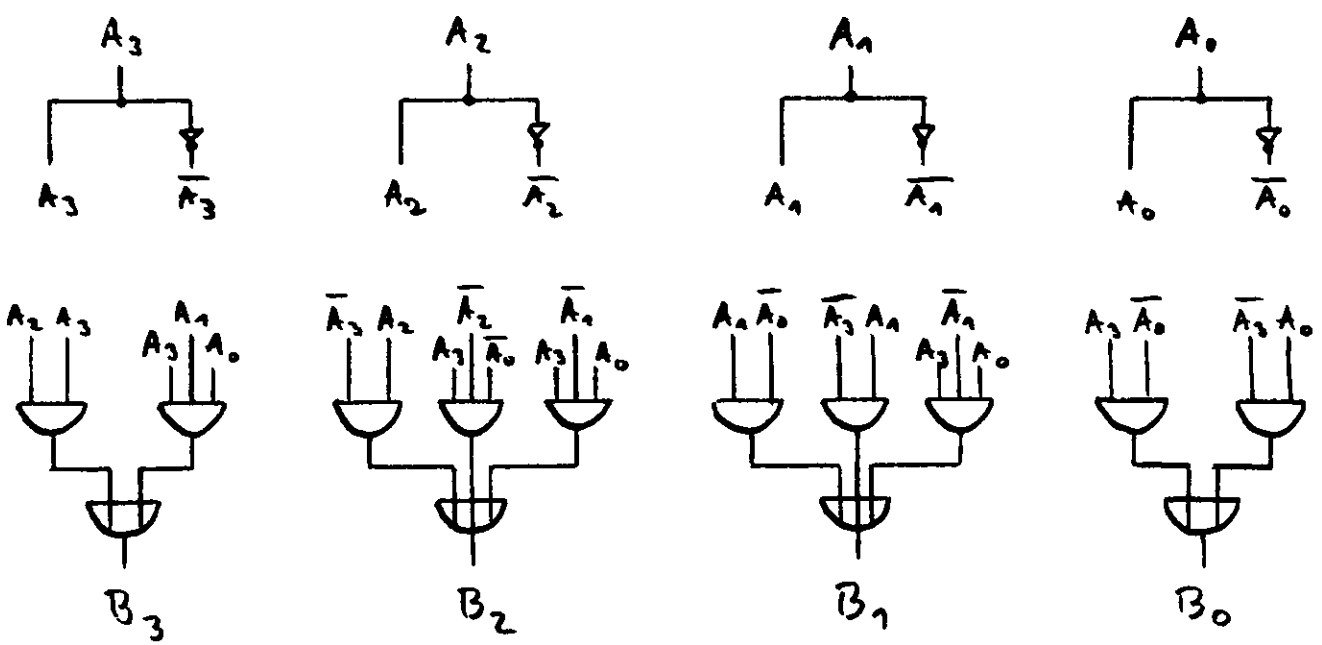
$$B_3 = \Sigma\ (11,12)$$
$$B_2 = \Sigma\ (4,5,6,7,8,9,10)$$
$$B_1 = \Sigma\ (2,3,6,7,9,10)$$
$$B_0 = \Sigma\ (1,3,5,7,8,10,12)$$

Diese tragen wir in Karnaugh-Diagramme ein und minimieren:



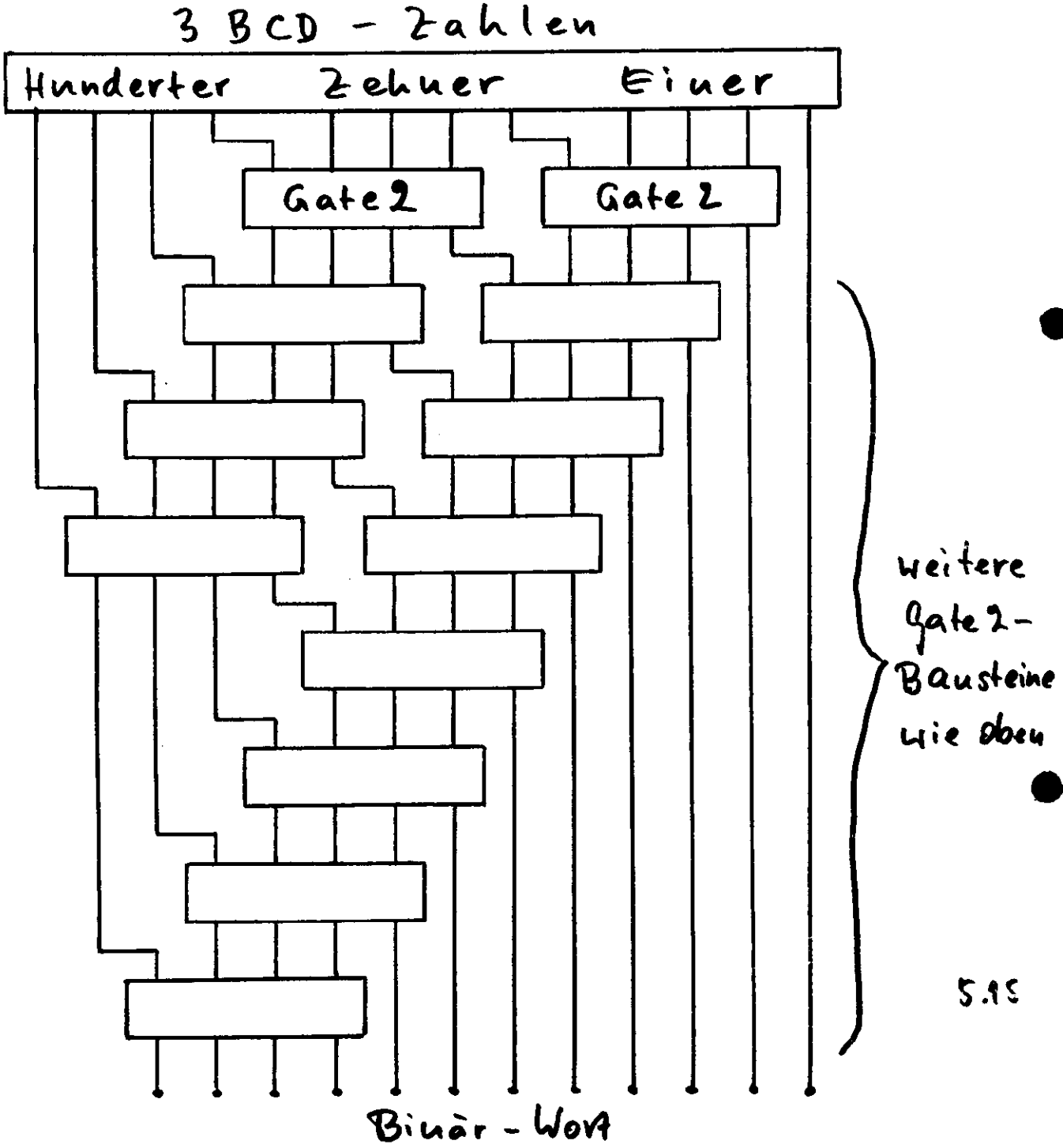
$$B_3 = A_2A_3 + A_3A_1A_0 \qquad B_2 = \bar{A}_3A_2 + A_3\bar{A}_2\bar{A}_0 + A_3\bar{A}_1A_0 \qquad B_1 = A_1\bar{A}_0 + \bar{A}_3A_1 + A_3\bar{A}_1A_0 \qquad B_0 = A_3\bar{A}_0 + \bar{A}_3A_0$$

Daraus ergibt sich in Bild 5.14 die Schaltung des Conversionsgate 2 für die SUB 3-Funktion



5.14

Die Prozedur einer Umwandlung von z.B. 3 BCD-Zahlen in eine entpsrechende Binärzahl geschieht nun in 3 Schritten, von denen einige wiederum mehrmals durchlaufen werden müssen (Bild 5.15):



5.15

Schritt 1

a) Beginne mit dem 2. LSB (in der Einer-Gruppe!), prüfe die BCD-Eingangsbits in Gruppen zu 4.

b) Ist der Wert der Gruppe 8 oder größer? Wenn ja, SUB 3; wenn nein, Bits ungehindert passieren lassen.

Schritt 2
a) Beginne mit dem nächsten LSB der noch nicht geprüften Bits, prüfe die Ausgangsbits der vorangehenden Stufen als Vierergruppe.
b) Ist der Wert der Gruppe 8 oder größer? Wenn ja, SUB 3; wenn nein, Bits ungehindert passieren lassen.

Schritt 3
Setze das Verfahren mit dem nächsten LSB des BCD-Wortes fort wie bei Schritt 2. Stoppe, wenn (n-4) Schritte beendet sind, wo n die Zahl der Bits im Urpsrungs-BCD-Wort bedeutet.

Im obigen Beispiel ist n = 12, es werden insgesamt n-4 = 8 Schritte ausgeführt, wie aus dem Bild auch ersichtlich ist.
Da wir dieses Mal bei den niederwertigsten Bits, den LSB, angefangen haben und in Richtung zum höchstwertigen Bit (MSB) fortschreiten, entspricht diese Richtung einer Division mit 2 pro Schiebe-Schritt.

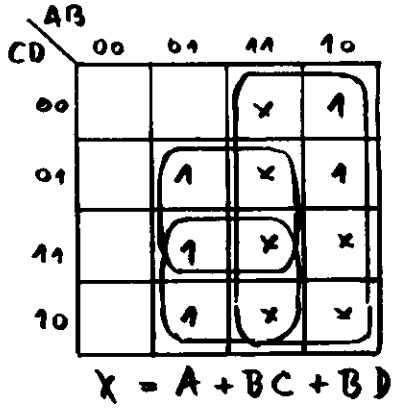
3.3 ADD - 3 - Funktion mit Schieberegistern

Man kann das Binärwort, das in ein BCD-Wort gewechselt werden soll, in ein Schieberegister einlesen, und zwar zunächst die ersten 3 Ziffern, diese dann prüfen, ob ihr Wert 5_{10} oder mehr enthält, in diesem Fall z.B. über einen Addierer eine festverdrahtete 3_{10} addieren und dann mit dem 4. Takt die 4. Ziffer einschieben. Nach dem 4. Takt steht dann im Register die BCD-Ziffer, der mögliche Addierübertrag geht in das nächste Register.

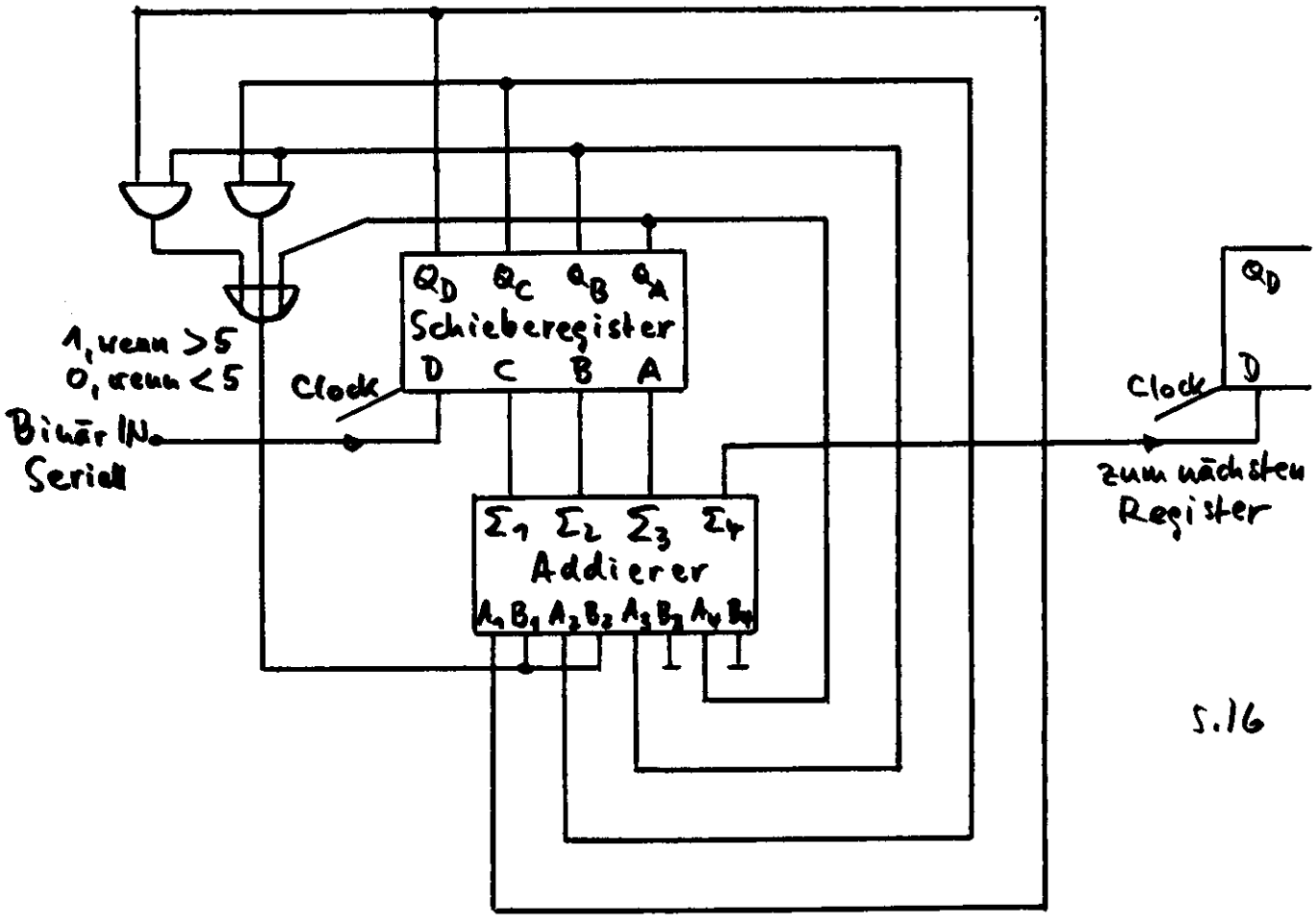
Die Prüfung der ersten 3 Ziffern auf > 5_{10} unter Einbeziehung der 8_{10} und 9_{10} geschieht mit einer Schaltung, die sich aus der folgenden Tabelle ergibt:

A	B	C	D	X
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1

$X = \sum (5,6,7,8,9)$



Die Gesamtschaltung, einschließlich eines 4 Bit-Addierers mit der festverdrahteten ADD 3-Funktion ist im nächsten Bild 5.16 gezeigt.



5.16

Die BCD → Binär-Conversion wird mit den gleichen Schieberegistern ausgeführt, nur daß jetzt in der anderen Richtung geschoben wird.

Der Nachteil dieser sequentiellen ADD 3-Methode ist der größere Zeitbedarf durch mehrere Schiebetakte. Bei einer 12 Bit-Umwandlung benötigt z.B. die ADD 3-Kombinatorik etwa 400 nsec, die Schieberegisterschaltung etwa 1300 nsec bei Verwendung eines Schiebetakts von 10 MHz.

3.4 Umwandlung mit tabelliertem Festwertspeicher

In einem Festwertspeicher (ROM = Read-only-memory) kann man das BCD-Äquivalent für jede Binärzahl speichern. Der Binärcode arbeitet als Adresse, unter dem das zugehörige BCD-Wort liegt. Dieses Verfahren wird jedoch sehr aufwendig, wie folgendes Beispiel zeigt. Ein 10 Bit-Binärcode verlangt 1024 Speicherworte je 13 Bit (nicht 4 x 4 = 16 Bits, da das 13. Bit nur 0 oder 1 sein kann, Bits 14,15, 16 immer 0 sind).

Es gibt aber die Möglichkeit, binäre Worte in 2 Teile zu teilen: $W_{bin} = X + Y$, X soll die MSB's, Y die LSB's darstellen. Ein 10 Bit-Wort kann man also teilen in

$$X = b_9\ b_8\ b_7\ b_6\ b_5\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0 \text{ und}$$
$$Y = 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ b_4\ b_3\ b_2\ b_1\ b_o$$

Nun können Y und Y getrennt in kleinen Festwertspeichern convertiert werden. Dann müssen aber die BCD-Ausgänge so addiert werden, daß das komplette Wort $W_{BCD} = X_{BCD} + Y_{BCD}$ entsteht. Hierbei werden natürlich die Regeln der BCD-Addition zweier Worte beachtet. Jeder Festwertspeicher braucht aber nur $2^5 = 32$ Worte zu enthalten. Durch diese Methode wird die Zahl der benötigten Speicherplätze stark reduziert.

4. Multiplexer (MPX)

In digitalen Systemen, in denen Informationen in verschiedenen Registern gespeichert ist, benötigt man Schaltungen, die selektiv den Inhalt bestimmter Flip-Flops oder ganzer Register weitertransportieren in andere Flip-Flops, Register, Akkumulatoren oder Speichersysteme, wo die Informationen verarbeitet oder abgelegt werden sollen.

Solche Schaltungen können von adressierbaren UND-Gates ausgeführt werden, deren Ausgänge auf einer Sammelleitung geodert werden. Man nenne diese Anordnungen Datenselektoren oder Multiplexer; Schaltungen, die mit n Adreßleitungen nacheinander 2^n Datenleitungen aufrufen und auf die Sammelleitung durchschalten.

Setzen wir n = 3, können wir 8 Dateneingänge verarbeiten. Die Adreßleitungen mögen A, B, C heißen, die Datenleitungen D_o bis D_7, der Ausgang X, dann lautet die Wertetabelle des 8fach-MPX:

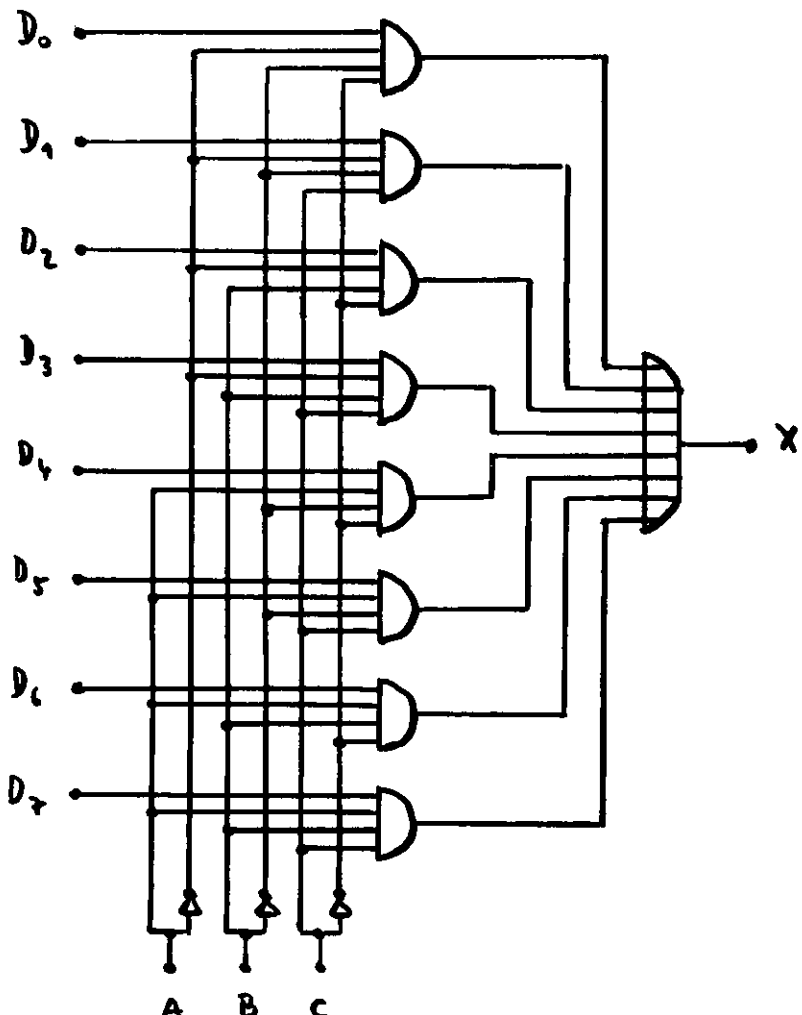
A B C	X
0 0 0	D_0
0 0 1	D_1
0 1 0	D_2
0 1 1	D_3
1 0 0	D_4
1 0 1	D_5
1 1 0	D_6
1 1 1	D_7

Die Gleichung für den Ausgang des 8fach-MPX wird also:

$$X = \bar{A}\,\bar{B}\,\bar{C}\,D_0 + \bar{A}\,\bar{B}\,C\,D_1 + \bar{A}\,B\,\bar{C}\,D_2 + \bar{A}\,B\,C\,D_3 + A\,\bar{B}\,\bar{C}\,D_4 + A\,\bar{B}\,C\,D_5 + A\,B\,\bar{C}\,D_6 +$$

$$+ A\,B\,C\,D_7$$

Daraus können wir die Schaltung entnehmen (Bild 5.17)

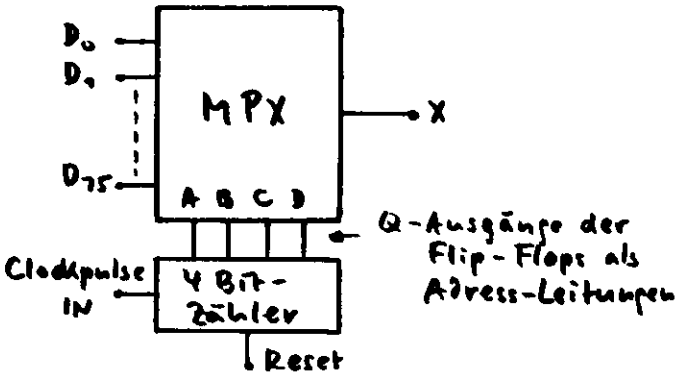


5.17

Haben an einem UND-Gate alle Adreßleitungen eine 1, wird der zugehörige Datenkanal am Ausgang erscheinen. Damit nicht grundsätzlich am Ausgang ein Kanal liegt, z.B. im Ruhezustand D_0, enthalten integrierte Multiplexer oft noch einen Enable-Eingang, der parallel an alle UND-Gate geschaltet ist und eine Ausgangsfunktion erlauben bzw. verhindern kann.

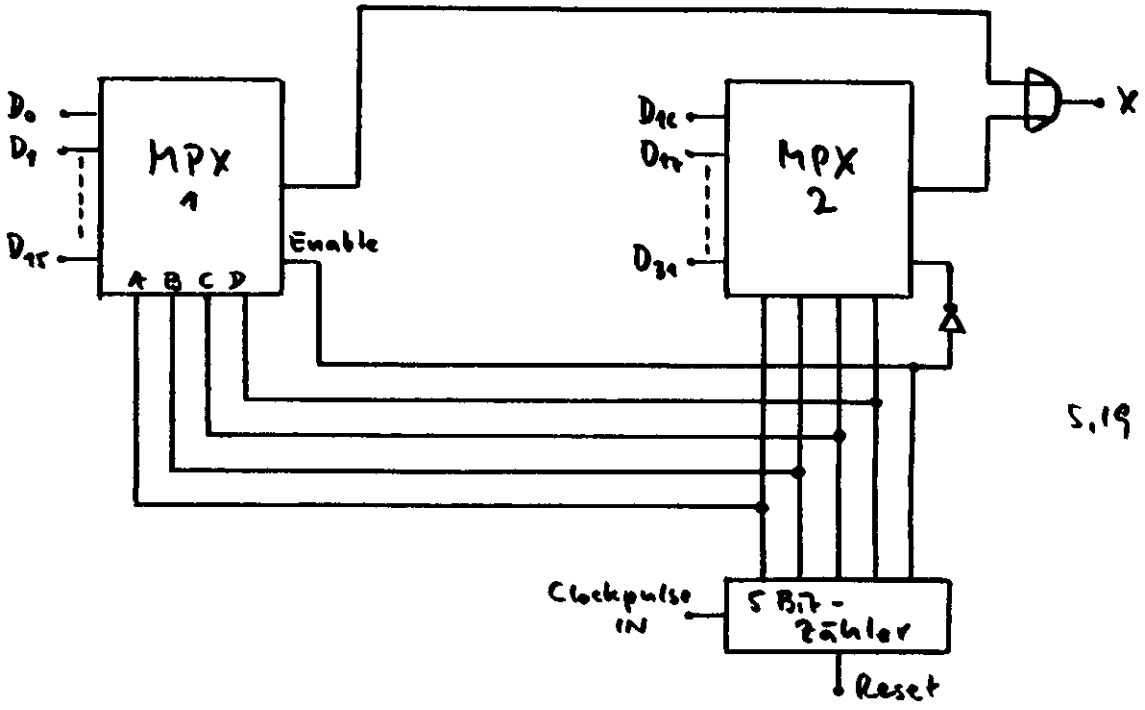
Da an einem Multiplexer mit n Adreßeingängen 2^n Eingangsleitungen parallen anliegen, aber nur 1 Ausgangsleitung vorhanden ist, wirkt ein MPX auch wie ein Parallel-Serien-Wandler.

Der Adreßaufruf geschieht zweckmäßigerweise mit einem Zähler oder Register. Bild 5.18 zeigt die Anschaltung für einen 16fach-MPX mit einem 4 Bit-Zähler.



5.18

Einen 32 fach-MPX kann man z.B. aus 2 x 16fach-MPX zusammenstellen, wenn man die Enable-Eingänge mit benutzt, Bild 5.19 zeigt dies.



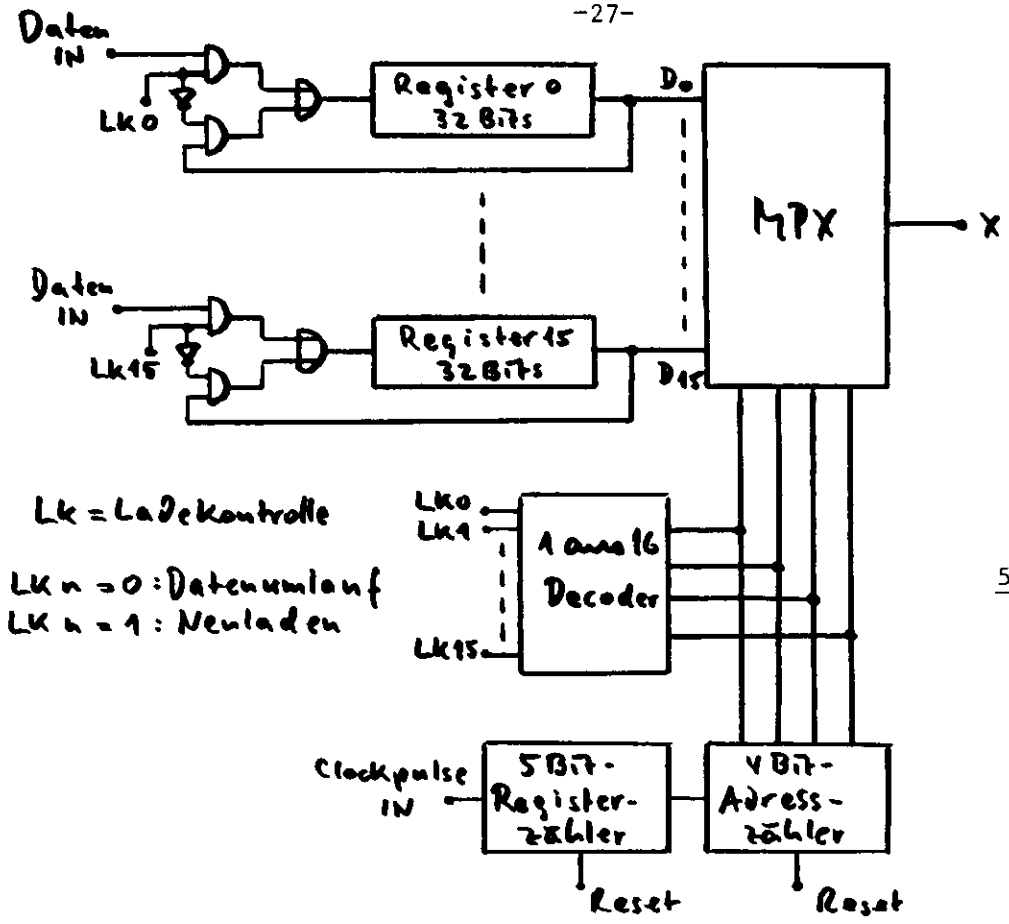
5.19

Die Ausgänge der beiden MPX müssen extern geodert werden.

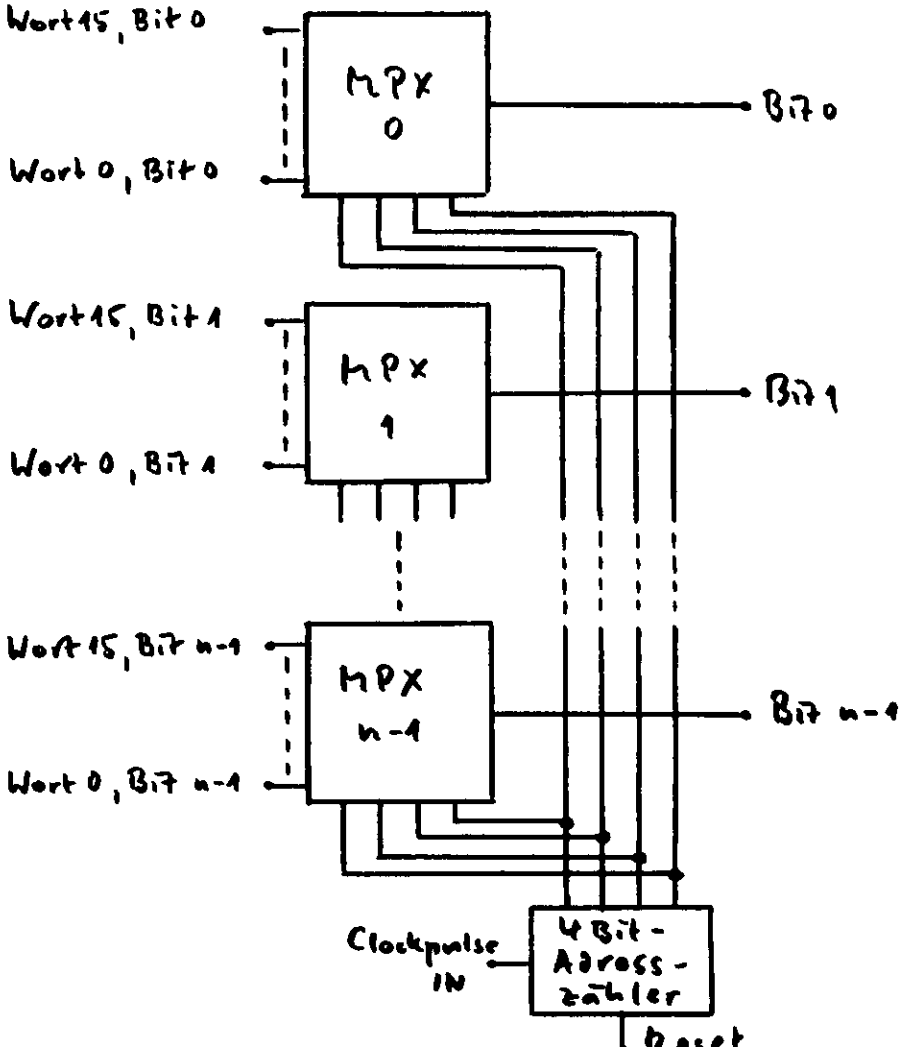
Wir können mit den MPX ein einzelnes Bit aus einem Register durch den Datenkanal transportieren; wir können aber auch ganze Worte seriell übertragen. Dazu lesen wir diese Worte entweder in ein Schieberegister oder ein Parallel-IN-Seriell-OUT-Register ein, rufen diesen Datenkanal auf und übertragen die n Bits des Registers. Hierbei muß natürlich die MPX-Adresse für die Dauer der Übertragung existent bleiben. Die Information muß entweder vorher in die Register geladen werden oder in umlaufenden Schieberegistern (vgl. Band 3, Abschnitt 12) dauernd vorhanden sein. Bild 5.20 zeigt das Beispiel von 16 Umlaufregistern, jedes 32 Bits lang, die über einen 16fach-MPX auf den Ausgang übertragen werden. Der Registerzähler, der die Weiterschaltung der Adresse steuert, muß dann 32 Takte zählen, d.h. 5 Bits enthalten. Die MPX-Adresse wird gleichzeitig decodiert durch einen 1 aus 16-Decoder, dessen Ausgänge die Kontrolle der Register übernehmen. Meist wird diese aus Zeitersparnis so geschaltet sein, daß, wenn Register 0 Daten während der 32 Takte überträgt, Register 1 während der gleichen Takte geladen wird, allgemein lädt Register n, wenn (n-1) überträgt. Dann muß LK0 an das Eingangsgate von Register 1 geschaltet werden, LK1 an Register 2 usw. Falls zu anderen Zeiten geladen werden soll, sind die LK_i aus dem Decoder entsprechend an die Registereingänge zu schalten. So kann jede beliebige Ladefolge hergestellt werden.

Die Umlaufregister im Bild 5.20 können natürlich auch durch andere, z.B. parallelsetzbare Register ersetzt werden, aus denen, wie oben bereits erwähnt, die Daten seriell über den MPX weitertransportiert werden.

Mit mehreren MPX kann man auch z.B. 16 Worte je n Bits auf n parallele Leitungen übertragen. Bild 5.21 zeigt ein entsprechendes Schaltbild.

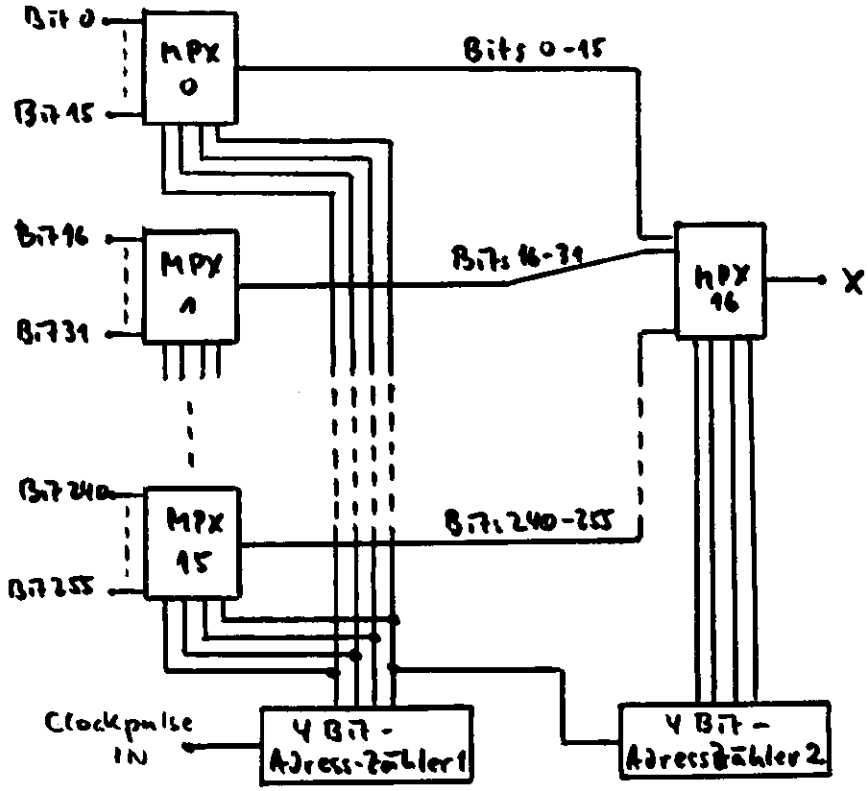

Daten IN
LK0
Register 0 32 Bits
D0
MPX
X
Daten IN
LK15
Register 15 32 Bits
D15
Lk = Ladekontrolle
LK n = 0 : Datenumlauf
LK n = 1 : Neuladen
LK0
LK1
LK15
1 aus 16 Decoder
Clockpulse IN
5 Bit-Registerzähler
4 Bit-Adresszähler
Reset
Reset

5.20


Wort 15, Bit 0
Wort 0, Bit 0
MPX 0
Bit 0
Wort 15, Bit 1
Wort 0, Bit 1
MPX 1
Bit 1
Wort 15, Bit n-1
Wort 0, Bit n-1
MPX n-1
Bit n-1
Clockpulse IN
4 Bit-Adresszähler
Reset

5.21

Während im vorigen Beispiel 16 Worte je n Bits übertragen werden sollten, kann es vorkommen, daß viele Einzelbits, mehr als Eingänge im MPX vorhanden sind, selektiv durchtransportiert werden sollen. Dann kann man eine Mehrebenen-Anordnung anwenden. Bild 5.22 zeigt eine Schaltung für 256 Datenquellen, die mit 16-fach-MPX durchgetaktet werden sollen.



5.22

Die ersten 16 Takte am Zähler 1 rufen die Bits 0-15 nacheinander auf, während Zähler 2 auf 0000 steht. Wenn Zähler 1 wieder auf 0000 springt, schaltet Zähler 2 auf 0001 und die Bits 16-31 passieren nacheinander den MPX 16. Nach 256 Takten, Zähler 1 ist 16 mal durchgelaufen, sind alle 256 Datenleitungen durchgeschaltet. Bei größeren Abfragesystemen sind auch mehr Ebenen aus MPX möglich.

5. Demultiplexer (DMPX)

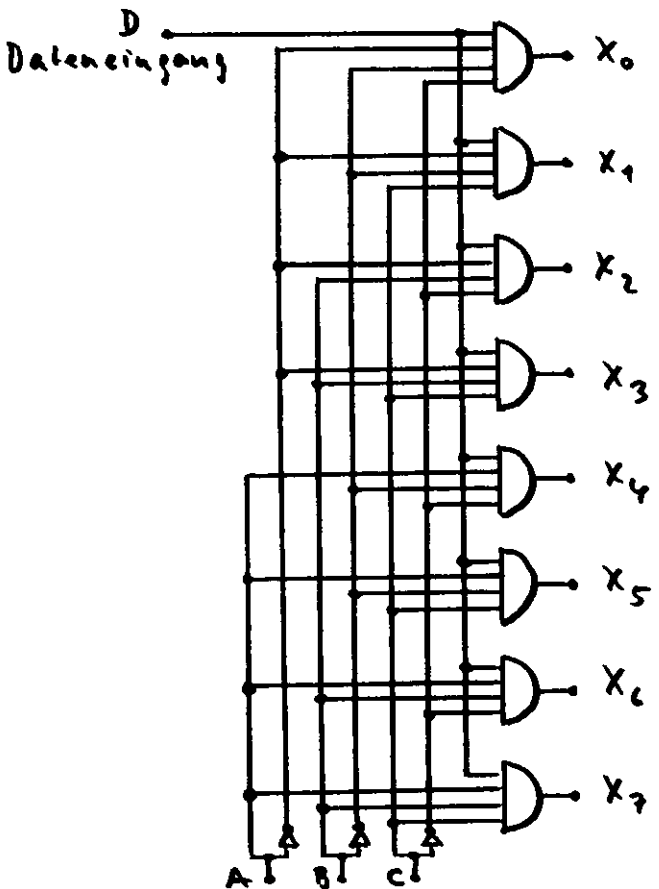
Demultiplexer führen die zu Multiplexern entgegengesetzte Funktionen aus, sie schalten eine einkommende Datenleitung auf eine von 2^n adressierte Leitung. Zur Adressierung sind wieder n Eingänge vorhanden. Schon aus dieser Beschreibung ist ersichtlich, daß DMPX schaltungsmäßig identisch mit 1 aus n-Decodern sind (vgl. Abschnitt 2), wenn man an die Adressiereingänge die zu decodierende Funktion legt, der Dateneingang wird dazu nicht benutzt. Die Wertetabelle für einen 8fach-DMPX lautet z.B.

Adreßleitungen A B C	X_o	X_1	X_2	X_3	X_4	X_5	X_6	X_7
0 0 0	D	0	0	–	–	–	–	0
0 0 1	0	D	0	–	–	–	–	0
0 1 0	0	0	D	0	–	–	–	0
0 1 1	0	0	0	D	0	–	–	0
1 0 0	0	–	–	0	D	0	–	0
1 0 1	0	–	–	–	0	D	0	0
1 1 0	0	–	–	–	–	0	D	0
1 1 1	0	–	–	–	–	–	0	D

Hieraus ergeben sich für die Ausgänge X_o ... X_7 die Gleichungen:

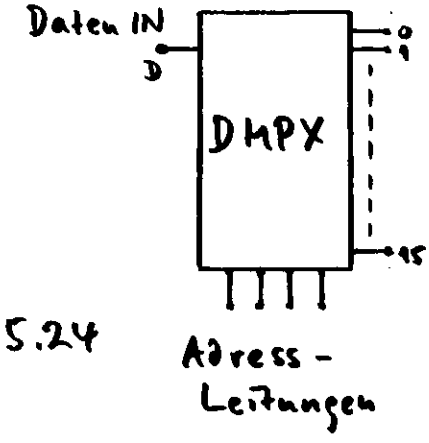
$$X_o = \bar{A}\,\bar{B}\,\bar{C}\,D$$
$$X_1 = \bar{A}\,\bar{B}\,C\,D$$
$$\vdots$$
$$X_6 = A\,B\,\bar{C}\,D$$
$$X_7 = A\,B\,C\,D$$

Wir können in Bild 5.23 die Schaltung des 8fach-DMPX zeichnen:

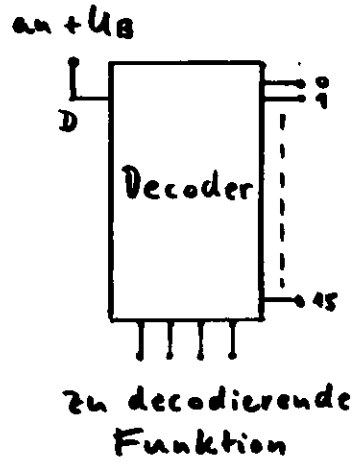


5.23

Wird die Schaltung als DMPX benutzt, wird sie, wie in Bild 5.24 am Beispiel eines 16fach-DMPX gezeigt, geschaltet. Die Decoderversion zeigt Bild 5.25.

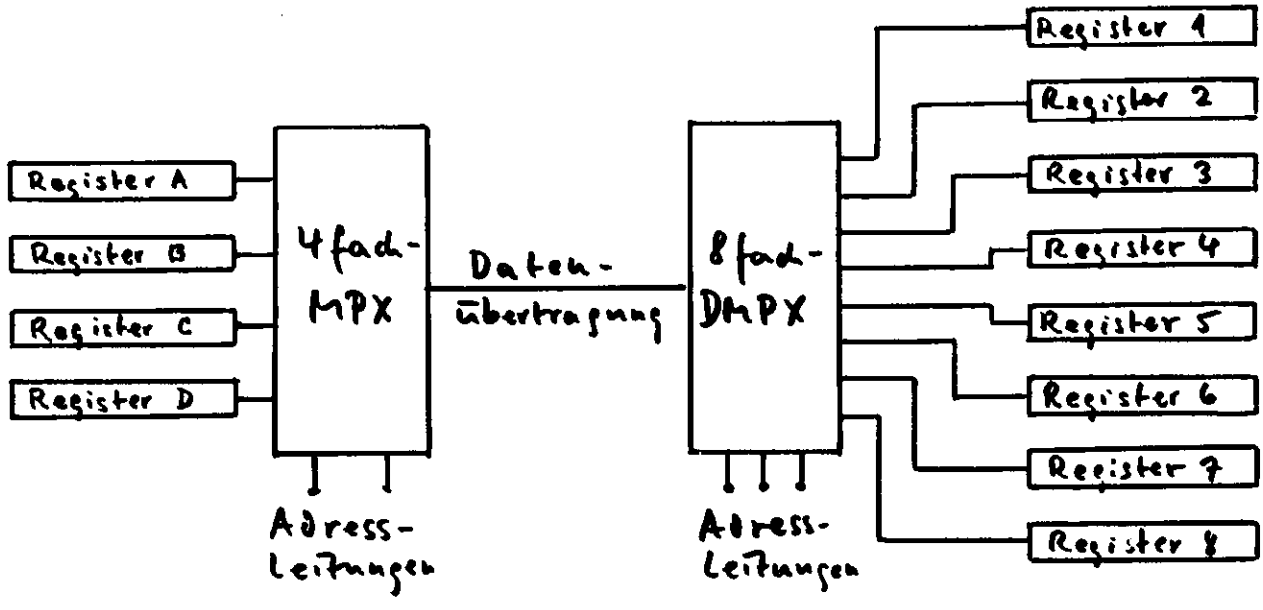


5.24



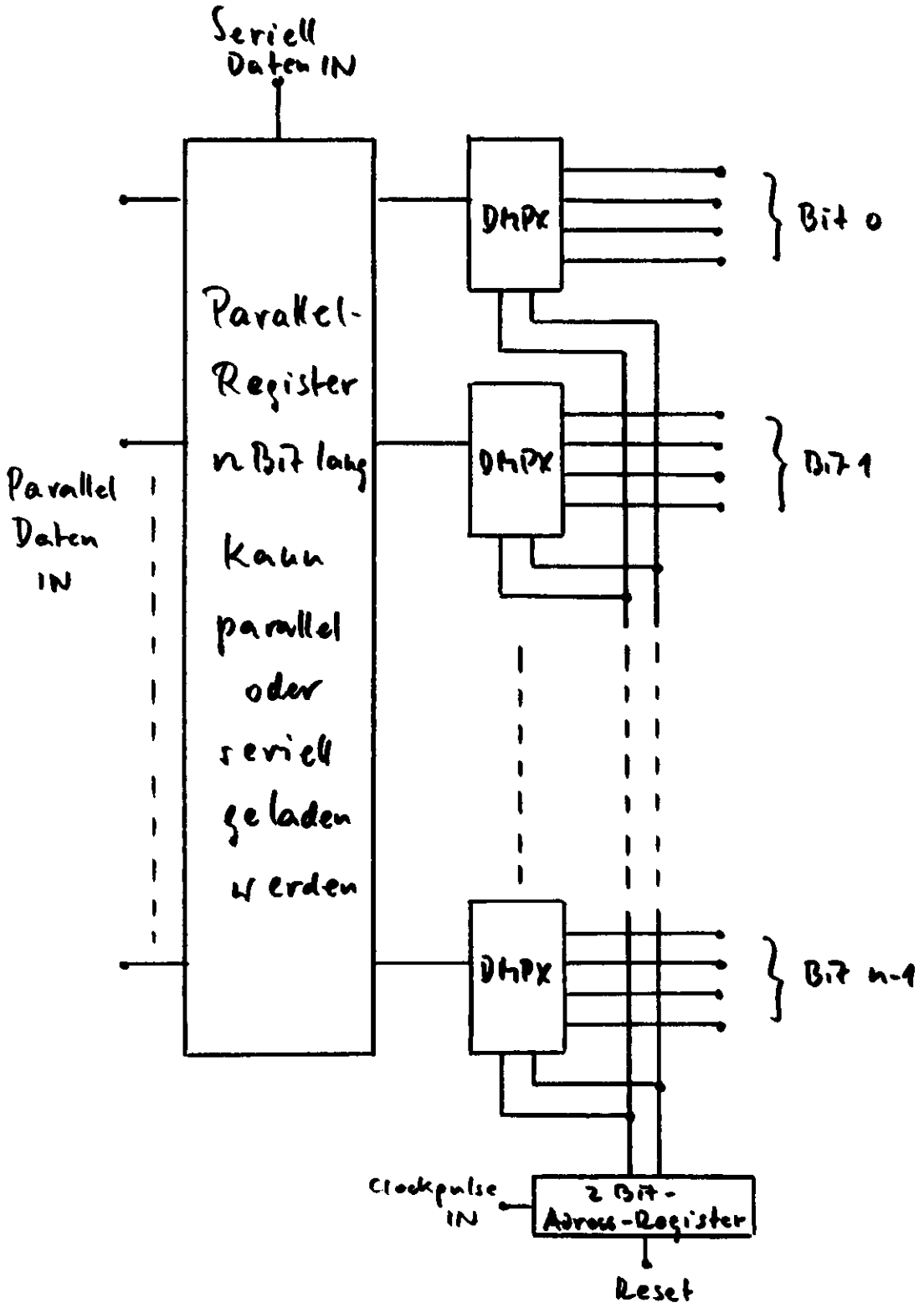
5.25

DMPX können z.B. zusammen mit MPX entsprechend den Beispielen des vorigen Abschnitts geschaltet werden. Bild 5.26 liefert ein Beispiel. Die Register A bis D sind geladen, ihr Inhalt soll über einen 4fach-MPX serialisiert werden, um auf einer einadrigen Datenfernleitung transportiert und am Ende der Leitung über einen DMPX auf 8 mögliche Register 1 bis 8 verteilt werden.



5.26

Statt serieller Worte können auch parallele Worte verteilt werden, z.B. wie in Bild 5.27 ein Wort mit n Bit Länge auf jeweils 4 Ausgänge, die vielleicht Paralleleingänge von 4 Registern sein können.
Die Adreßleitungen verteilen das Eingangswort auf einen der 4 Ausgänge:


Seriell Daten IN
Parallel-Register n Bit lang
Kann parallel oder seriell geladen werden
Parallel Daten IN
DMPX
DMPX
DMPX
Bit 0
Bit 1
Bit n-1
Clockpulse IN
2 Bit-Adress-Register
Reset

S.27

DMPX arbeiten wie Serien-Parallel-Converter, während MPX wie Parallel-Serien-Converter funktionieren.

6. Exclusiv-ODER-Gates, Paritätsprüfung, Komparatoren.

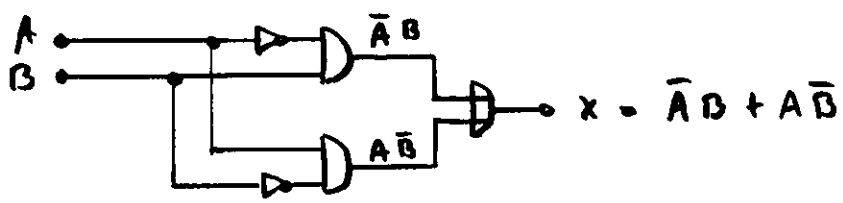
Beim normalen ODER-Gate, auch Inclusiv-ODER genannt, erscheint am Ausgang eine 1, wenn ein oder mehrere oder alle Eingänge eine 1 haben. Das Exclusiv-ODER-Gate erzeugt nur dann eine 1, wenn einer der Eingänge eine 1 hat; aber eine 0, wenn mehrere Eingänge gleichzeitig auf 1 sind. Die Wertetabelle für das 2fach-Exclusiv-ODER-Gate, das meist als XOR abgekürzt wird, lautet:

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Daraus folgt als Gleichung für das XOR-Gate:

$$X = \bar{A}\,B + A\,\bar{B}$$

Die Schaltung, die diese Funktion realisiert, ist dann (Bild 5.28):



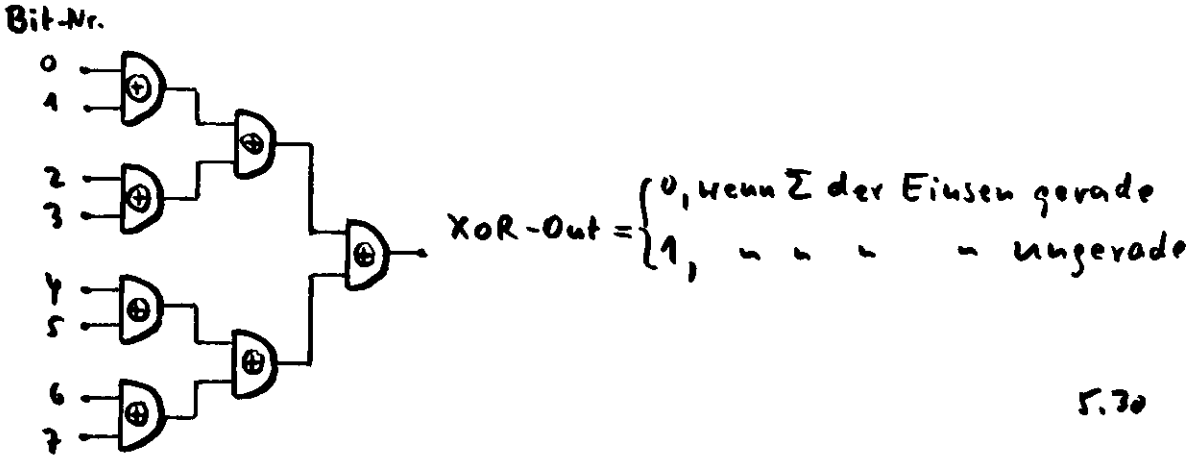
5.28

Bei der Paritätsprüfung (vgl. Band 3, Abschnitt 6.4) wird eine Anzahl von parallel anstehenden Bits daraufhin geprüft, ob die Summe der Einsen zu den Bits gerade oder ungerade ist. Das XOR-Gate ist für diese Prüfung geeignet, es liefert am Ausgang eine 0, wenn die Summe gerade ist, eine 1, wenn sie ungerade ist.

Die Paritätsprüfung von 2 Bits wird also mit einem XOR-Gate vorgenommen, dessen häufig verwendetes Abkürzungssymbol das Zeichen in Bild 5.29 ist.

5.29

Sollen mehr als 2 Bits auf Parität geprüft werden, müssen die Ausgänge jedes XOR-Gates wieder mit anderen geprüft werden, wie es Bild 5.30 für 8 Bits zeigt:



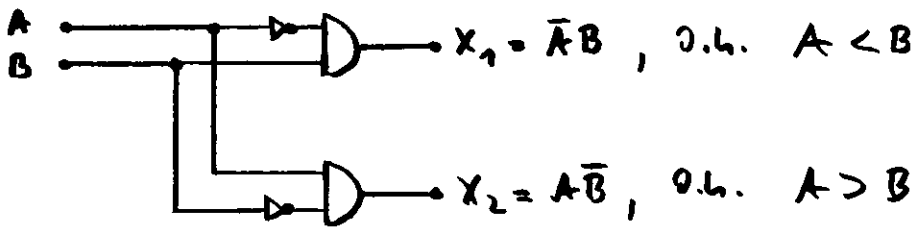
5.30

Für noch größere Bitzahlen muß der "Baum" aus XOR-Gates entsprechend vergrößert werden.

Der Ausgang des XOR-Gates, so wie es oben in Bild 5.28 geschaltet ist, sagt nur, daß entweder A = 0, B = 1, d.h. A < B, oder daß A = 1, B = 0, d.h. A > B ist. Da für manche Vergleichsoperationen in der Arithmetik ein eigenes Signal für A < B bzw. eines für A > B benötigt wird, brauchen wir in der Schaltung nur das Ausgangs-ODER-Gate zu entfernen. Dann haben die Wertetabelle

A	B	X_1 (A < B)	X_2 (A > B)
0	0	0	0
0	1	1	0
1	0	0	1
1	1	0	0

Daraus folgt $X_1 = \bar{A}\,B$, $X_2 = A\,\bar{B}$ und die Schaltung wird (Bild 5.31):

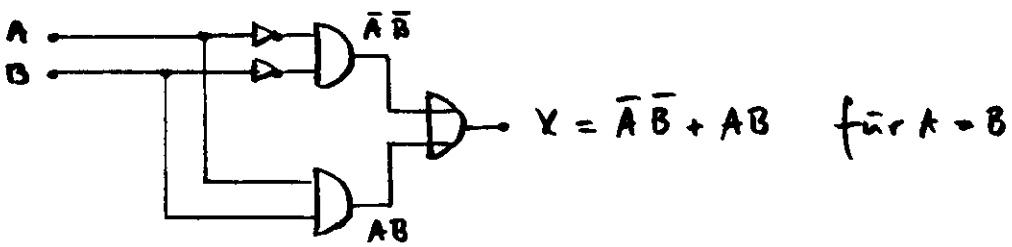


5.31

Wollen wir zwei Bits auf Gleichheit prüfen, muß die Wertetabelle für diese Schaltung lauten:

A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

Daraus folgt die Gleichung $X = \bar{A}\,\bar{B} + A\,B$, die sich durch die Schaltung in Bild 5.32 realisieren läßt:



5.32

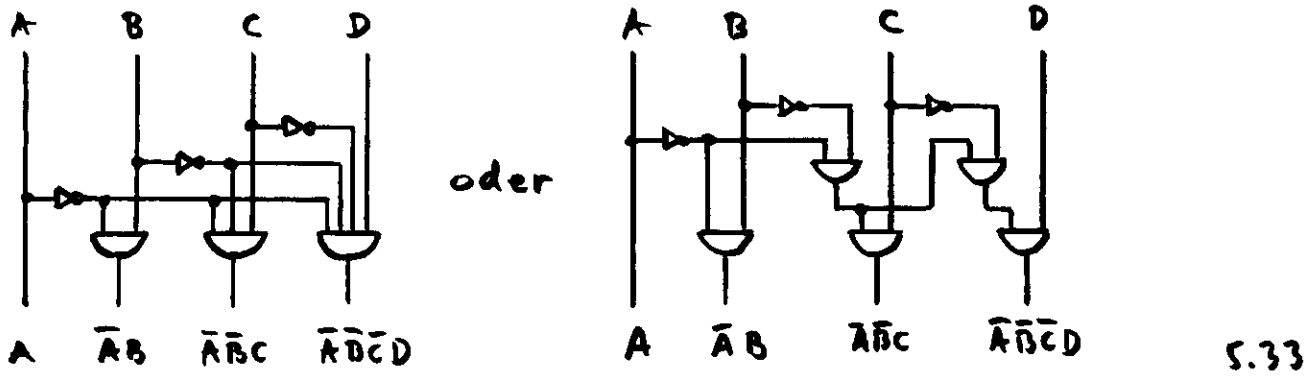
Schaltungen, die wie die in Bild 5.31 bzw. 5.32 oder mehr Bits auf Gleichheit oder Ungleichheit prüfen, heißen digitale Komparatoren.

7. Prioritätsschaltung

Das Prioritätsnetzwerk, meist als Schaltung für die Anforderung und Zuteilung von Speicherzyklen im Rechner eingesetzt, entscheidet nach einer festgelegten Dringlichkeit, welche von den vorliegenden Anforderungen zuerst abgearbeitet wird.
Haben wir z.B. 4 Anforderungsleitungen A, B, C und D, von denen A die höchste, B die zweithöchste, C die dritthöchste und D die niedrigste Priorität haben soll, können wir nach folgender Überlegung schalten:

- Anforderung A kommt immer durch, falls vorhanden,
- Anforderung B kommt durch, wenn A nicht vorhanden ist,
- Anforderung C kommt durch, wenn A nicht und B nicht vorhanden sind,
- Anforderung D kommt durch, wenn A nicht und B nicht und C nicht vorhanden sind.

Die Schaltung ergibt sich direkt (Bild 5.33):



5.33

Aus der nachfolgenden Tabelle lassen sich die Prioritäten und die Häufigkeit ihres Auftretens gut erkennen:

A B C D	A	$\bar{A}$ B	$\bar{A}$ $\bar{B}$ C	$\bar{A}$ $\bar{B}$ $\bar{C}$ D
0 0 0 0	0	0	0	0
0 0 0 1	0	0	0	1
0 0 1 0	0	0	1	0
0 0 1 1	0	0	1	0
0 1 0 0	0	1	0	0
0 1 0 1	0	1	0	0
0 1 1 0	0	1	0	0
0 1 1 1	0	1	0	0
1 0 0 0	1	0	0	0
1 0 0 1	1	0	0	0
1 0 1 0	1	0	0	0
1 0 1 1	1	0	0	0
1 1 0 0	1	0	0	0
1 1 0 1	1	0	0	0
1 1 1 0	1	0	0	0
1 1 1 1	1	0	0	0

8. Kontroll - Logik

Die Kontroll-Logik überwacht den Programmablauf und sorgt für den zeitlich richtigen Transfer von Befehlen und Daten. Sie arbeitet als Verbinder zwischen dem Speicher und der arithmetischen sowie den J/O-Einheiten. Der Datenfluß zwischen diesen Einheiten ist am höchsten, wenn sich ein Arbeitsspeicherzyklus, d.h. ein Lese-Schreibzyklus mit einer typischen Dauer zwischen 0.25 und 2 µsec, unmittelbar an den nächsten reiht. Meist gibt es unaufschiebbare Verzögerungen im Ablauf, z.B. zwischen dem Beginn der Befehlsübertragung und dem Eintreffen dort, wo dieser benötigt wird oder auch die Wartezeit, wenn eine Anforderung höherer Priorität durch ein Interuptsignal das laufende Programm abbricht. Die sich addierenden Wartezeiten können leicht einige Zyklen ausmachen.

Auf der anderen Seite kann die Zeit, die die arithemtische Einheit wartet, bis ein Operand eingetroffen ist, abgekürzt werden, wenn die Logik einige Befehle im voraus denkt und die neuen Operanden in Bufferregister schiebt, während die arithmetische Einheit noch vorangehende Operanden bearbeitet. Ebenso kann das Rechenergebnis in Registern abgelegt werden, aus denen sie in den Speicher transportiert werden, während bereits neue Daten berechnet werden. Durch geschickte logische Anordnung kann die Kontroll-Einheit den Datenfluß innerhalb der Zentraleinheit regularisieren.

Die Kontroll-Logik übernimmt also folgende Hauptaufgaben, sie steuert

- den Programmablauf
- den Datenverkehr zwischen dem Hauptspeicher und den J/O-Kanälen
- die Ausführung der Operatoranforderungen von der Konsole.

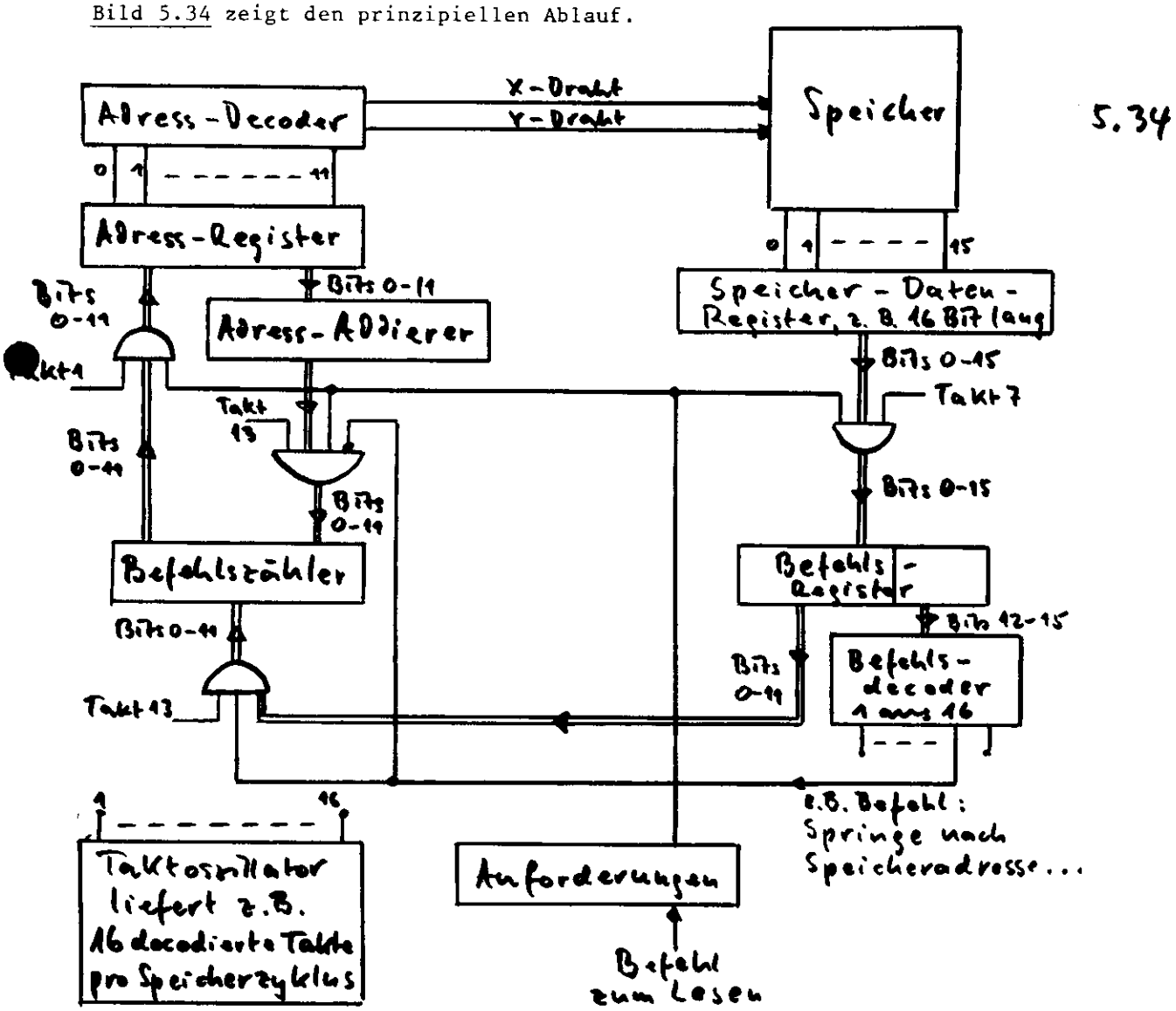
8.1 Steuerung des Programmablaufs

Hierzu werden zwei wesentliche Register benötigt, der Befehlszähler BZ und das Befehlsregister BR. Der Befehlszähler enthält n Bits, so daß 2^n Speicherzellen (Worte) adressiert werden können. zB. 12 Bits bei 4096 Speicherzellen. Er wird beim Start des ersten Programms vom Operator, bei weiteren Programmen auch automatisch mit der Anfangsadresse geladen. Normalerweise stehen die Befehle eines Programms in aufeinanderfolgenden Adressen, d.h. nach dem Lesen eines Befehls wird der Inhalt des Befehlszählers automatisch um 1 erhöht. Soll die folgende Adresse jedoch eine ganz andere, willkürlich wählbare sein, muß der Zähler durch

einen besonderen Befehl innerhalb der Kontroll-Logik mit der neuen Adresse geladen werden. Dieser Befehl heißt Sprungbefehl.
Sollen die folgenden Adressen nicht um 1 erhöht werden, sondern periodisch um eine andere Zahl zwischen 1 und 9, muß die neue Adresse aus Indexregistern geholt werden.

Der Inhalt des Befehlszählers kann über getaktete UND-Gates in ein Adreßregister (Speicheradreßregister SAR) übertragen werden, von wo aus er dekodiert die X- bzw. Y-Koordinate der gewünschten Speicherkerne aufruft. Die Übertragung erfolgt immer dann, wenn im Kernspeicherzyklus der Lesebefehl erscheint.

Das Befehlsregister enthält soviele Bits, wie es der Wortlänge des Rechners entspricht. Dieses Register kann von dem gleichlangen Speicher-Daten-Register SDR über UND-Gates geladen werden. Ist ein neues Befehlswort aus dem Arbeitsspeicher gelesen, steht es im SDR, auf einen bestimmten Takt gelangt das Wort dann ins Befehlsregister. Jetzt kann der Befehl dekodiert werden und zur Ausführung gelangen.
Bild 5.34 zeigt den prinzipiellen Ablauf.



Der decodierte Befehl wird im wesentlichen entweder im Kontrollsystem selbst, im Rechenwerk oder für J/O-Operationen verwendet. Für die Kontroll-Logik bestimmte Befehle sind Sprung- oder organisatorische Befehle. Das Springen kann entweder unbedingt erfolgen oder bedingt nach Vergleich, z.B. ob der Akkumulator gleich oder ungleich Null, positiv oder negativ ist. Im Bild ist ein Sprungbeispiel eingefügt.

Mit Takt 1 der z.B. 16 Takte pro Speicherzyklus lädt der Befehlszähler das Adreßregister, nach dem der Lesebefehl eingetroffen ist. Dann werden 6 Takte zum Decodieren und Lesen der Adresse benötigt, die dann im SDR steht. Von dort werden die Bits mit Takt 7 ins Befehlsregister übernommen und anschließend decodiert. Ist z.B. ein Sprungbefehl entschlüsselt worden, wird dieser und die 12-Bit-Adresse mit Takt 13 wieder in den Befehlszähler zurücktransportiert, von wo sie im nächsten Zyklus zur Ausführung gelangt.

Andere decodierte Befehle werden z.B. in der arithmetischen Einheit verwendet, dazu gehören die arithmetischen Befehle, Schiebe- und Transferanweisungen. Bei den meisten Rechenwerksbefehlen muß erst noch ein Operand aus dem Speicher gelesen werden, auf den der Befehl angewendet werden soll.

8.2 Datenverkehr zwischen Hauptspeicher und den J/O-Kanälen

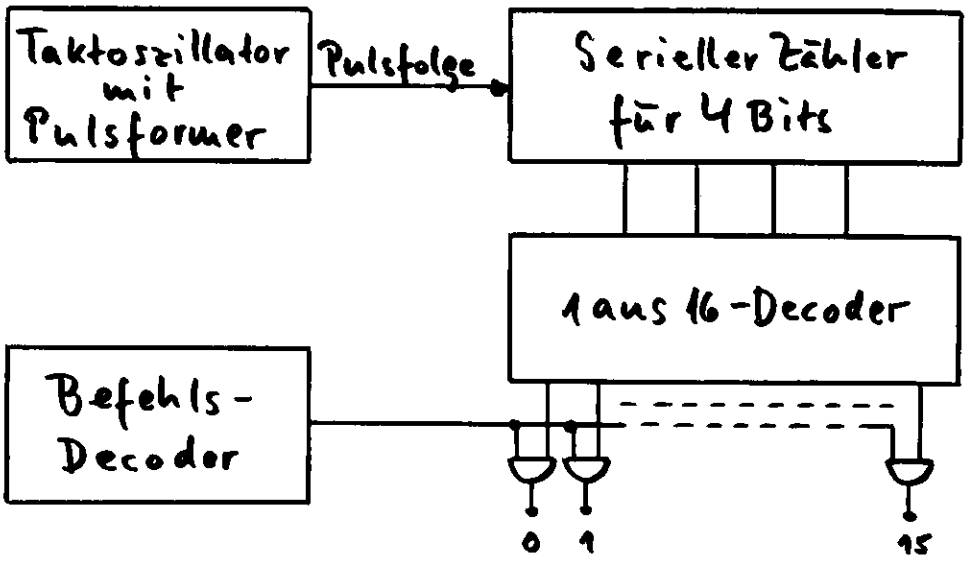
Soll der Befehl in den J/O-Kanälen verarbeitet werden, werden die Bits 0-11 benutzt, um die Adresse des J/O-Kanals festzulegen und an das angeschlossene Gerät die Adressen des Arbeitsspeichers zu geben, die die anzugebenden Daten enthalten, nicht die Daten selbst. Die J/O-Kanalsteuerung muß anschließend deren Übertragung aus dem Speicher veranlassen.

Die Steuerung der Ein-Ausgabekanäle hängt stark von der Organisation des Peripherieanschlusses ab. In Band 4, Abschnitt 8, wurde hierüber ausführlich berichtet und auch ein Steuerungsbeispiel mit gepufferten Kanälen dargestellt.

Generell kommt zu Beginn einer J/O-Operation eine Anforderung zur Datensteuerung, in der auch die Adresse der Arbeitsspeicherzelle, mit der die Operation beginnen soll, geliefert wird. Nach Kontrolle im Prioritätsnetzwerk werden dann die Speicherzyklen bereitgestellt. Dauert der Transfer eines Wortes wesentlich länger als ein Arbeitsspeicherzyklus, werden die freibleibenden Zwischenzyklen anderen Teilen des Rechners zur Verfügung gestellt.

8.3 Taktgenerator

Der Taktgenerator, ein quarzkontrollierter Oszillator, liefert eine Folge äquivalenter Pulse, die die einzelnen Speicherzyklen in mehrere Teile, z.B. 16 teilt. Der Abstand der Pulse richtet sich nach der Lese-Schreibzykluszeit, er liegt z.B. bei etwa 60 nsec für eine Zykluszeit von 1 µsec. Die Einzeltakte dienen dazu, Teiloperationen auszuführen wie Decodieren, Transfer in andere Register usw. Um die richtigen Taktpulse zur Ausführung einer bestimmten Operation zu erhalten, muß die Zeitfolge gegatet werden, wie es z.B. Bild 5.35 zeigt.



5.35