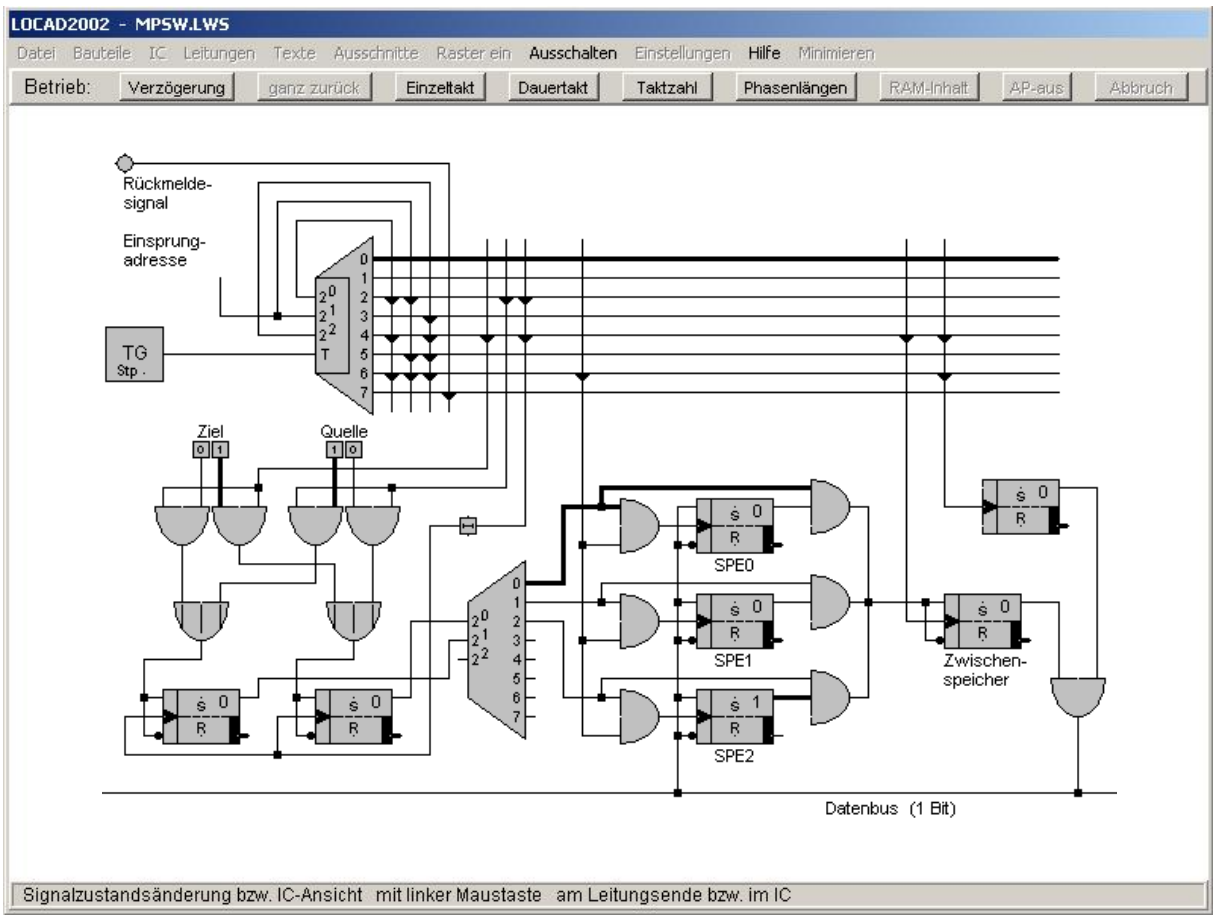
Karl-Heinz Loch

Technische Informatik mit *LOCAD2002*



Technische Informatik mit *LOCAD2002*

Weinesch 16
41812 Erkelenz

Vorwort zur ersten Auflage

Dieses Buch mit Programm ist für alle diejenigen gedacht, die auf eine einfache – fast spielerische – Weise die grundlegenden Prinzipien und Arbeitsweisen einer Datenverarbeitungsanlage auf der Hardware-Ebene kennen lernen möchten. Es ist eine Lehrsystem für die Grundlagen der Digitaltechnik, das auch den Nicht-Techniker den einfachen Aufbau logischer Schaltungen ermöglicht – ohne Lötkolben und die damit verbundenen Enttäuschungen und Kosten.

Das zum Buch gehörende CAD-Programm *LOCAD2002* ist auf allen gängigen Betriebssystemen lauffähig. Es bietet Ihnen in Verbindung mit diesem Buch die Möglichkeit, die Funktionsweise wesentlicher elektronische Baugruppen von Computern zu verstehen. Im direkten Umgang mit digitalen Schaltungen erfahren Sie, dass die Digitaltechnik relativ leicht zu durchschauen ist. Die Arbeit mit *LOCAD* erleichtert nicht nur das Begreifen von grundlegenden Schaltungen, sondern ermutigt auch zu eigenen Schaltungsentwürfen. Um Leistungsfähigkeit und Anwendungsmöglichkeiten von Datenverarbeitungsanlagen richtig einschätzen zu können, benötigt man Kenntnisse über deren Aufbau und Funktionsweise. *LOCAD* ist bei der Erarbeitung der Grundlagen hierzu behilflich. Stufe um Stufe werden aus einfachen Grundschaltungen immer komplexere Schaltungen mit wesentliche Funktionsteilen eines Computers aufgebaut und deren Zusammenspiel verdeutlichet.

Grundkenntnisse aus der Elektrizitätslehre und Mathematik sind nützlich, aber nicht unbedingt erforderlich.

Erkelenz, 1989

Vorwort zur zweiten Auflage

Durch die ständige Weiterentwicklung des CAD-Programms in den Jahren nach Erscheinen der ersten Auflage sind Änderungen, Erweiterungen und sonstige Anpassungen des Textes an die aktuelle Programm-Version notwendig geworden.
Den vielen Anwendern, die mit ihren Verbesserungsvorschlägen zur Optimierung und Steigerung der Stabilität des Programms beigetragen haben, sei hiermit gedankt.

Erkelenz, im Juni 2002
Karl-Heinz Loch

Inhaltsverzeichnis

1 Grundlagen

1.1 Begriffsbestimmungen 5
1.2 Grundfunktionen 8
1.2.1 UND-Funktion 8
1.2.2 ODER-Funktion 9
1.2.3 NICHT-Funktion 10
1.3 Normalformen 11
1.3.1 Disjunktive Normalform 11
1.3.2 Konjunktive Normalform 13
1.4 Synthese von Schaltnetzen 15
1.4.1 Vorgehensweise 15
1.4.2 Übungsaufgabe 16
1.5 Technische Realisierung der Grundfunktionen 16
1.5.1 Kontaktlogik 16
1.5.2 Transistor – Logik 18
1.5.2.1 Transistor als Schalter 19
1.5.2.2 Reihenschaltung zweier Transistoren 20
1.5.2.3 Parallelschaltung zweier Transistoren 23

2 Schaltnetze

2.1 Rechenschaltungen 24
2.1.1 Das Dualsystem 24
2.1.1.1 Umwandlung Dual → Dezimal 26
2.1.1.2 Umwandlung Dezimal → Dual 26
2.1.1.3 Addition im Dualsystem 26
2.1.2 Halbaddierer 27
2.1.3 Volladdierer 29
2.1.4 Paralleladdierer 31
2.1.5 Parallelsubtrahierer 32
2.1.5.1 Subtraktion von Dualzahlen 32
2.1.5.2 Negative Dualzahlen 33
2.1.6 Umschaltbare Rechenschaltung 40
2.2 Code-Umsetzer 41
2.2.1 Dezimal-Dual-Umsetzer 42
2.2.2 Dual-Dezimal-Umsetzer 43
2.3 Sieben-Segment-Anzeige 47
2.4 Koppeldioden 48
2.5 ROM und PROM 52
2.6 Multiplexer 54
2.6.1 Torwirkung der UND-Schaltung 55
2.7 Impulserzeugung 57

3 Schaltwerke

3.1 Digitale Speicherelemente 58
3.1.1 RS-Flipflop 59
3.1.2 Getaktetes RS-Flipflop 60
3.1.3 D-Auffangflipflop 62
3.1.4 RS-Master-Slave-Flipflop 62
3.1.5 JK-Master-Slave-Flipflop 64
3.2 Rechenwerke 68
3.2.1 Schieberegister 68
3.2.2 Serienaddierwerk 71
3.2.3 Umschaltbares serielles Rechenwerk 73
3.3 Zählschaltungen 74
3.4 Autonome Schaltwerke 79
3.4.1 Ampelsteuerung 81

4 Programmsteuerung eines Rechners

4.1 Struktur einer DVA 83
4.1.1 Rechenwerk 83
4.1.2 Arbeitsspeicher 84
4.1.2.1 RAM-Baustein 88
4.1.3 Steuerwerk 89
4.2 Steuersignalerzeugung 91
4.2.1 Handsteuerung 91
4.2.2 Mikroprogramm-Steuerwerk 94
4.3 Programmsteuerung 97
4.3.1 Ablaufsteuerung mit einem 1-Bit-Befehl 97
4.3.2 Einzelteile des Modell-Computers 99
4.3.2.1 Takteinheit 99
4.3.2.2 Befehlsregister, Befehlszählregister und Speicher 99
4.3.2.3 Rechenwerk 100
4.3.2.4 Befehlsdecoder 102
4.3.2.5 Betrieb des Modells 103

Anhang A: Boolesche Algebra 105

Anhang B: Gesetze der Schaltalgebra 106

Anhang C: Lösungen zu einzelnen Aufgaben aus dem Text 107

Anhang D: Weitere Aufgaben 112

Literaturverzeichnis 114

Stichwortverzeichnis 115

1 Grundlagen

1.1 Begriffsbestimmungen

Der englische Mathematiker George Boole (1815 – 1864) entwickelte in seinem Buch „The Laws of Thought“ zur systematischen Behandlung der Logik eine algebraische Struktur, die heute als Boolesche Algebra[1] bekannt ist. Die **Schaltalgebra** ist eine wichtige Anwendung der Booleschen Algebra im technischen Bereich und dient zur Beschreibung und Untersuchung logischer Schaltungen und somit zur Entwicklung automatischer Steuerungen und elektronischer Rechenanlagen.

Eine **Schaltvariable** ist eine Veränderliche, die nur eine bestimmte Anzahl verschiedener Zustände annehmen kann. Sie wird deshalb auch als **digitale**[2] Veränderliche bezeichnet. In der Technik werden meistens zweiwertige (**binäre**) Schaltvariable verwendet. Diese Variable können nur zwei Zustände, die sogenannten **Binärzustände**, annehmen, die mit den Symbolen 0 und 1 gekennzeichnet werden.

In logischen Schaltungen werden fast ausschließlich Spannungswerte als Variable verwendet. Man definiert entweder niedrige Spannung als 0 und hohe Spannung als 1 (positive Logik) oder hohe Spannung als 0 und niedrige Spannung als 1 (negative Logik). In der TTL Technik (TTL = **T**ransistor-**T**ransistor-**L**ogik) werden den Binärzuständen 0 und 1 nach Ein- und Ausgang verschieden folgende Spannungspegel zuordnet:

Eingang:	$0 \triangleq 0 - 0{,}8$ V;	$1 \triangleq 2{,}0 - 5$ V
Ausgang:	$0 \triangleq 0 - 0{,}4$ V;	$1 \triangleq 2{,}4 - 5$ V

Auch Schalter in einem Stromkreis stellen zweiwertige Schaltvariable dar. Sie können offen ($\triangleq 0$) oder geschlossen ($\triangleq 1$) sein.

Eine **n-stellige Schaltfunktion f** ordnet jedem n-Tupel von Schaltzuständen aus $D \subset \{0, 1\}^n$, $n \in N$ eindeutig einen der beiden Schaltzustände 0 oder 1 zu.
Schreibweise: $f : D \rightarrow \{0, 1\}$

$$(a_1, a_2, ..., a_n) \rightarrow f(a_1, a_2, ..., a_n)$$

Eine elektronische Schaltung, die eine Schaltfunktion realisiert, heißt **logische Schaltung**.

Die unabhängigen Variablen $a_1, a_2, ..., a_n$ nennt man **Eingangsvariable** oder kurz **Eingänge**, das Funktionsergebnis $f(a_1, a_2, ..., a_n)$ **Ausgangsvariable** oder kurz **Ausgang**.

Die an den Eingängen vorliegenden Schaltzustände werden **Eingangssignale**, der Schaltzustand des Ausganges wird **Ausgangssignal** genannt.

---

[1] siehe Anhang A

[2] von lat. digitus = Finger; bezeichnet die Eigenschaft, **nur diskrete** Werte annehmen zu können. Beim Zählen mit den Fingern gibt es keine Zwischenwerte. Im Gegensatz dazu bezeichnet „analog“ die Eigenschaft, Zwischenwerte annehmen zu können.

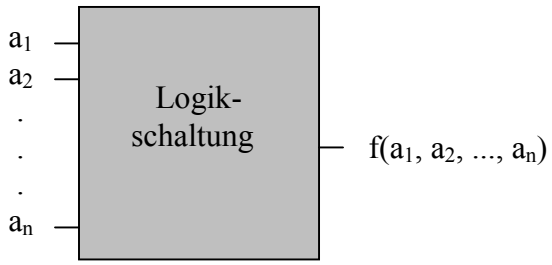


Abb. 1.1-1 Prinzip einer Logikschaltung

Eine Schaltfunktion f wird meist in Form einer **Funktionstabelle** angegeben und häufig als **Verknüpfung** der Schaltvariablen bezeichnet.

Beispiel:
a und b seien Schaltvariable und f eine 2-stellige Schaltfunktion, die durch folgende Tabelle definiert ist

a	b	f (a, b)
0	0	1
0	1	0
1	0	1
1	1	1

Abb. 1.1-2 Definition einer Schaltfunktion durch eine Tabelle

Schaltungen mit mehreren Ausgängen lassen sich aus einzelnen Schaltungen mit je einem Ausgang zusammensetzen, wie die folgende Abbildung verdeutlicht.

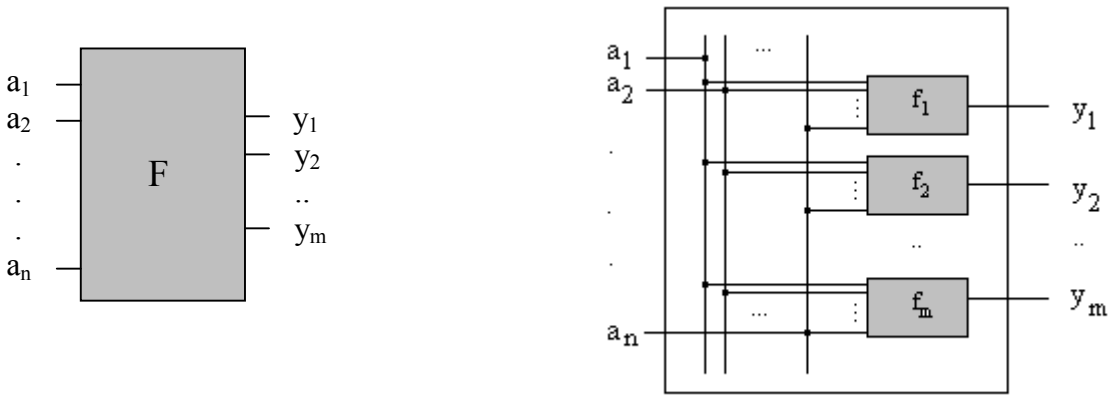


Abb. 1.1-3 Beispiel für die Realisierung eines Schaltfunktionsbündels

F stellt hier ein Funktionsbündel dar:

$F(a_1, a_2, ..., a_n) = (y_1, y_2, ..., y_m)$ mit

$y_1 = f_1(a_1, a_2, ..., a_n)$
$y_2 = f_2(a_1, a_2, ..., a_n)$
..
$y_m = f_m(a_1, a_2, ..., a_n)$

Eine Logikschaltung heißt **Schaltnetz**, wenn der Zustand des Ausganges zu jedem Zeitpunkt nur von den Schaltzuständen der Eingänge abhängt.

Hängt der Ausgangszustand jedoch noch von früheren Eingangszuständen ab – dies ist durch ein Rückkopplung eines oder mehrer Ausgänge über eine Verzögerungseinheit auf einen oder mehrere Eingänge möglich – so handelt es sich um ein Schaltung „mit Gedächtnis“ und wird **Schaltwerk** genannt.

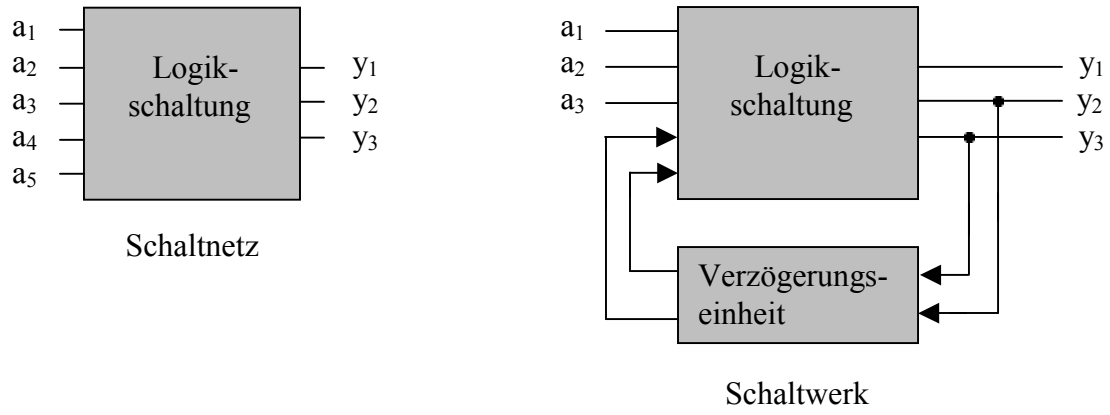


Abb. 1.1-4 Unterschied zwischen Schaltnetz und Schaltwerk

Die folgende Tatsache ist für den Aufbau logischer Schaltung und besonderer Bedeutung.

Jede Schaltfunktion lässt sich auf die drei Grundfunktionen UND, ODER und NICHT zurückführen.

Diese Grundfunktionen – häufig auch Grundverknüpfungen genannt - bilden damit ein vollständiges System[1].

Im Kapitel 1.3 wird gezeigt, wie sich eine Schaltfunktion auf die Grundverknüpfungen zurückführen lässt.

[1] In der Schaltalgebra heißt eine Menge von Verknüpfungen **vollständig**, wenn sich **jede** Schaltfunktion allein mit Hilfe von Verknüpfungen dieser Menge darstellen lässt.

1.2 Grundfunktionen

1.2.1 UND-Funktion

Die UND-Funktion wird auch als UND-Verknüpfung oder **Konjunktion** bezeichnet und ist eine 2-stellige Schaltfunktion mit folgender Funktionstabelle:

a	b	UND (a, b)
0	0	0
0	1	0
1	0	0
1	1	1

Mit dem Verknüpfungszeichen $\wedge$ lässt sich UND(a, b) kürzer schreiben:
$\boldsymbol{a \wedge b}$ (lies: a und b).

Eine elektronische Schaltung zur Realisierung der UND-Funktion wird UND-Schaltung, UND-Glied oder auch AND-Gatter genannt.

Unabhängig von der praktischen Ausführung gibt es hierfür ein nach DIN 40700 genormtes Symbol.

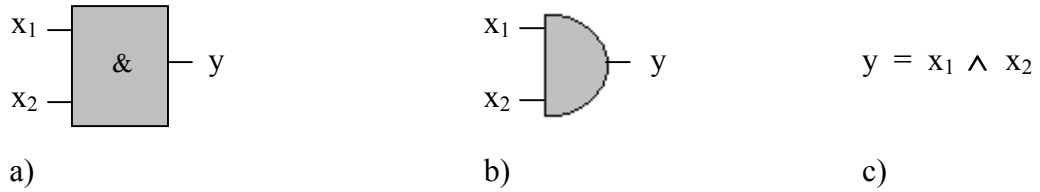


Abb. 1.2.1-1 UND-Schaltung
a) DIN 40700 b) frühere Fassung c) schaltalgebraische Schreibweise

In den folgenden Kapiteln wird weiterhin das alte Symbol benutzt, da es wegen der deutlicheren Wirkrichtung der Signale zu übersichtlicheren Schaltplänen führt.

Es gibt auch UND-Schaltungen mit mehr als zwei Eingängen:

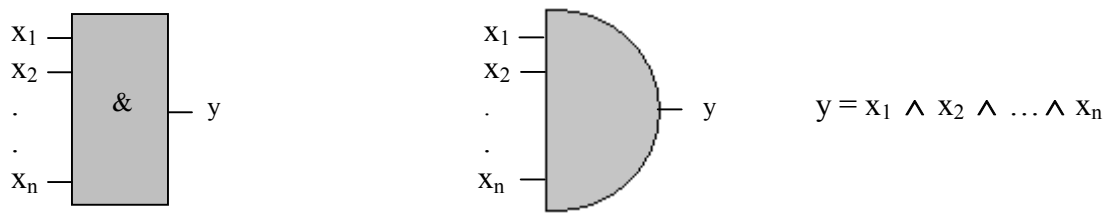


Abb. 1.2.1-2 UND-Schaltung mit n Eingängen

Der Ausgang einer UND-Schaltung hat nur dann den Zustand 1, wenn **alle** Eingänge den Zustand 1 haben.
In allen anderen Fällen hat der Ausgang den Zustand 0.

1.2.2 ODER-Funktion

Die ODER-Funktion wird auch als ODER-Verknüpfung oder **Disjunktion** bezeichnet und ist eine 2-stellige Schaltfunktion mit folgender Funktionstabelle:

a	b	ODER (a, b)
0	0	0
0	1	1
1	0	1
1	1	1

Mit dem Verknüpfungszeichen $\vee$ lässt sich ODER(a, b) kürzer schreiben:
$\boldsymbol{a \vee b}$ (lies: a oder b).

Eine elektronische Schaltung zur Realisierung der ODER-Funktion wird ODER-Schaltung, ODER-Glied oder auch OR-Gatter genannt.

Die folgende Abbildung zeigt das genormte Symbol für das ODER-Glied.

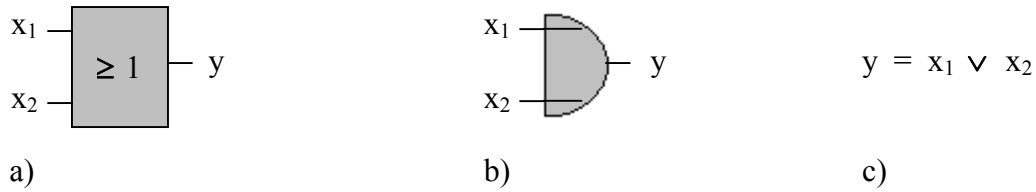


Abb. 1.2.2-1 ODER-Schaltung
a) DIN 40700 b) frühere Fassung c) schaltalgebraische Schreibweise

Es gibt auch ODER-Schaltungen mit mehr als zwei Eingängen:

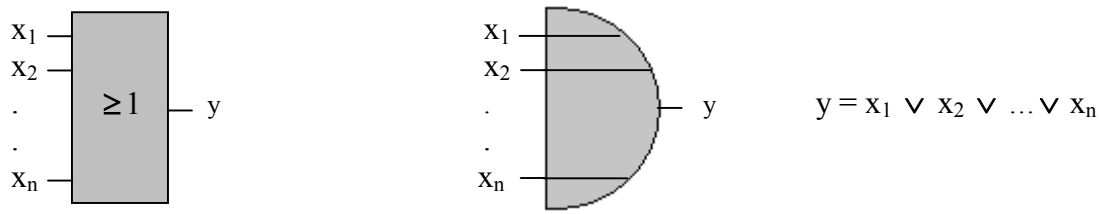


Abb. 1.2.2-2 ODER-Schaltung mit n Eingängen

> Der Ausgang einer ODER-Schaltung hat den Zustand 1, wenn **wenigstens ein** Eingang den Zustand 1 hat.

Oder anders ausgedrückt:

> Der Ausgang einer ODER-Schaltung hat nur dann den Zustand 0, wenn **alle** Eingänge den Zustand 0 haben.

1.2.3 NICHT-Funktion

Die NICHT-Funktion wird auch als NICHT-Verknüpfung oder **Negation** bezeichnet und ist eine 1-stellige Schaltfunktion mit folgender Funktionstabelle:

a	NICHT (a)
0	1
1	0

Mit dem Verknüpfungszeichen $\bar{}$ lässt sich NICHT(a) kürzer schreiben: $\bar{a}$ (lies: nicht a)
Man nennt $\bar{a}$ auch das **Negat** oder das **Komplement** von a.
Eine Schaltung zur Realisierung der NICHT-Funktion wird NICHT-Schaltung, NICHT-Glied, NOT-Gatter, Negator oder auch Inverter genannt.

Die folgende Abbildung zeigt das genormte Symbol.

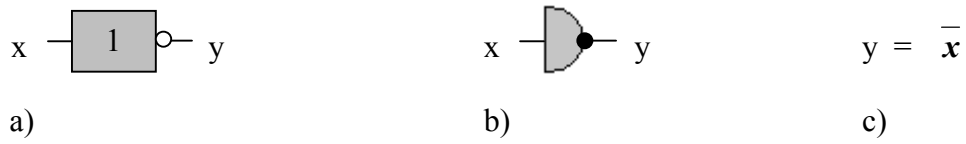


Abb. 1.2.3-1 NICHT-Schaltung
a) DIN 40700 b) frühere Fassung c) schaltalgebraische Schreibweise

Der Ausgang einer NICHT-Schaltung hat stets den Zustand, den der Eingang **nicht** hat.

Bei einer Zusammenschaltung von NICHT-Gliedern mit UND-Gliedern bzw. ODER-Gliedern braucht die NICHT-Verknüpfung nicht gesondert dargestellt zu werden. Stattdessen kann man die Negation am jeweiligen Eingang oder Ausgang durch einen kleinen Kreis (neue DIN-Norm) oder einen Punkt (alte DIN-Norm-Fassung) kenntlich machen.

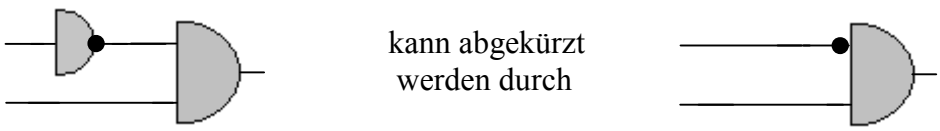


Abb. 1.2.3-2 Vereinfachung mit Negationspunkt

Um bei der schaltalgebraischen Beschreibung von Schaltfunktionen Klammern sparen zu können, werden folgende Vorrangregeln vereinbart:

$\bar{}$ hat Vorrang vor $\wedge$
$\wedge$ hat Vorrang vor $\vee$

1.3 Normalformen

Die Normalformen und deren Herstellung sollen hier an einem konkreten Beispiel eingeführt werden.

Aufgabe: Überwachung einer Maschine

> Eine Maschine soll so überwacht werden, dass ein Warnsignal (W = 1) abgegeben wird, wenn eine Störung eintritt (S = 1) und während der Störung der Aufsichtsposten nicht besetzt ist (A = 0) und eine Resettaste nicht gedrückt ist (R = 0). Weiterhin soll das Warnsignal gegeben werden, wenn die Resettaste betätigt ist, obwohl keine Störung vorliegt.

Eine Analyse der Aufgabe zeigt, dass das Warnsignal W von den drei Variablen S, A und R abhängt. W ist damit eine 3-stellige Schaltfunktion [W = f(S, A, R)], deren Funktionstabelle folgendermaßen aussieht (machen Sie sich das bitte aus auf Aufgabenstellung klar):

	S	A	R	W
Zeile 1	0	0	0	0
Zeile 2	0	0	1	1
Zeile 3	0	1	0	0
Zeile 4	0	1	1	1
Zeile 5	1	0	0	1
Zeile 6	1	0	1	0
Zeile 7	1	1	0	0
Zeile 8	1	1	1	0

1.3.1 Disjunktive Normalform

Die obige Schaltfunktion kann nach folgendem Verfahren aus den drei Grundfunktionen zusammengesetzt werden:

Für alle Kombinationen, für die der Funktionswert W den Zustand 1 hat, werden die Konjunktionsterme aufgeschrieben und diese durch ODER-Verknüpfungen zusammengefasst.

Ein **Konjunktionsterm** ist eine UND-Verknüpfung, die alle Schaltvariablen in negierter oder nicht-negierter Form genau einmal enthält.

Beispiele:
$S \wedge A \wedge R$ und $\overline{S} \wedge A \wedge \overline{R}$ sind Konjunktionsterme der Schaltvariablen S, A und R.
$S \wedge A$ ist *kein* Konjunktionsterm der Schaltvariablen S, A und R, *da* R *nicht enthalten ist*.

Bei n Schaltvariablen gibt es 2^n verschiedene Konjunktionsterme.

Da jeder Konjunktionsterm nur bei einer bestimmten Belegung der Schaltvariablen den Wert 1 hat, wird ein Konjunktionsterm auch **Minterm** genannt.
Den Minterm, der für eine bestimmte Kombination den Wert 1 hat, erhält man, indem man die UND-Verknüpfung aller vorkommenden Schaltvariablen hinschreibt und die Variablen negiert, die bei dieser Kombination den Zustand 0 haben.

Also:
Minterm zu Zeile 2: $\overline{S} \wedge \overline{A} \wedge R$
Minterm zu Zeile 4: $\overline{S} \wedge S \wedge R$
Minterm zu Zeile 5: $S \wedge \overline{A} \wedge \overline{R}$

Diese Minterme werden jetzt mit ODER zusammengefasst. Insgesamt ergibt sich unter Beachtung der Vorrangregeln:

$$W = \overline{S} \wedge \overline{A} \wedge R \vee \overline{S} \wedge A \wedge R \vee S \wedge \overline{A} \wedge \overline{R}$$

Der damit erhaltene Ausdruck heißt **disjunktive Normalform**. Die Minterme sind disjunktiv verknüpft.

Das Schaltnetz dazu kann man sofort zeichnen.

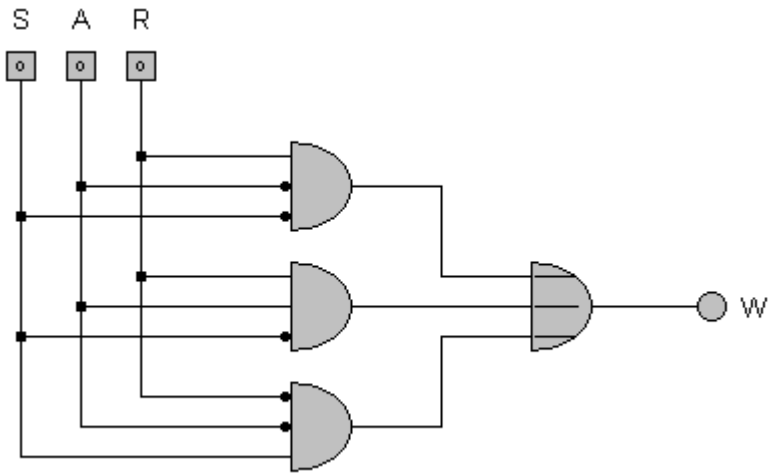


Abb. 1.3.1-1 Schaltnetz zur Beispielaufgabe (disjunktive Normalform)

Überzeugen Sie sich davon, dass diese Schaltung die Bedingungen erfüllt, indem Sie alle Eingangskombinationen durchspielen.

1.3.2 Konjunktive Normalform

Der Aufbau der obigen Schaltfunktion aus den drei Grundfunktionen ist ebenso nach dem folgenden Verfahren möglich:

Für alle Kombinationen, für die der Funktionswert W den Zustand 0 hat, werden die Disjunktionsterme aufgeschrieben und diese durch UND-Verknüpfungen zusammengefasst.

Ein **Disjunktionsterm** ist eine ODER-Verknüpfung, die alle Schaltvariablen in negierter oder nicht-negierter Form genau einmal enthält.

Beispiele:
$S \vee A \vee R$ und $\overline{S} \vee A \vee \overline{R}$ sind Disjunktionsterme der Schaltvariablen S, A und R.

Selbstverständlich gibt es bei n Variablen 2^n verschiedene Disjunktionsterme.

Da jeder Disjunktionsterm nur bei einer Kombination der Variablen den Wert 0, sonst immer den Wert 1 hat, heißt ein Disjunktionsterm auch **Maxterm**.
Den Maxterm, der für eine bestimmte Kombination den Wert 0 hat, erhält man, indem man die ODER-Verknüpfung aller vorkommenden Schaltvariablen hinschreibt und die Variablen negiert, die bei dieser Kombination im Zustand 1 sind.

Also:
Maxterm zu Zeile 1: $S \vee A \vee R$
Maxterm zu Zeile 3: $S \vee \overline{A} \vee R$
Maxterm zu Zeile 6: $\overline{S} \vee A \vee R$
Maxterm zu Zeile 7: $\overline{S} \vee \overline{A} \vee R$
Maxterm zu Zeile 8: $\overline{S} \vee \overline{A} \vee \overline{R}$

Diese Maxterme werden jetzt mit UND zusammengefasst. Insgesamt ergibt sich unter Beachtung der Vorrangregeln:

$$W = (S \vee A \vee R) \wedge (S \vee \overline{A} \vee R) \wedge (\overline{S} \vee A \vee \overline{R}) \wedge (\overline{S} \vee \overline{A} \vee R) \wedge (\overline{S} \vee \overline{A} \vee \overline{R})$$

Der so erhaltene Ausdruck heißt **konjunktive Normalform**. Die Maxterme sind konjunktiv verknüpft.
Das zugehörige Schaltnetz dazu lässt sich sofort zeichnen.

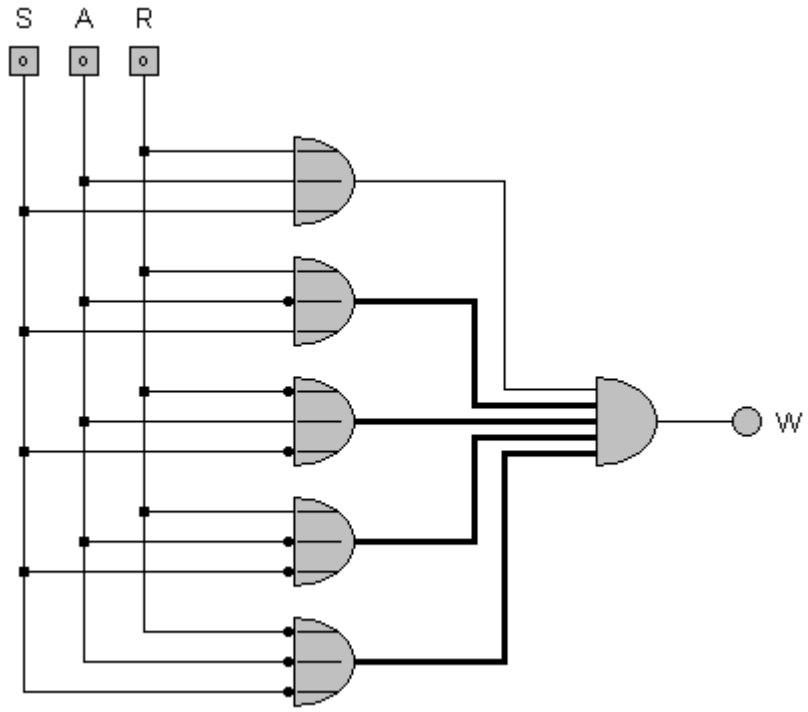


Abb. 1.3.2-1 Schaltnetz zur Beispielaufgabe (konjunktive Normalform)

Vergewissern Sie sich auch hier, dass die Bedingungen erfüllt werden.

Die Normalformen zeigen, dass jede Schaltfunktion wenigstens in zwei Formen dargestellt werden kann:

1. als ODER-Verknüpfung der Minterme (disjunktive Normalform)
2. als UND-Verknüpfung der Maxterme (konjunktive Normalform)

Beide Formen können mit Hilfe der De Morganschen Gesetze[1] ineinander umgewandelt werden.

Mit den oben beschriebenen Verfahren können wir somit zu jeder Schaltfunktion über die Normalformen das zugehörige Schaltnetz entwickeln. Der Nachteil bei dieser Methode ist, dass die Ausdrücke oft sehr groß und redundant werden. Mit Hilfe des Gesetze der Schaltalgebra[1] lassen sie sich meistens stark vereinfachen. Hierzu braucht man allerdings sehr viel Übung und Einfühlungsvermögen. Es existieren aber auch systematische Vereinfachungsverfahren wie das Quine-McCluskey-Verfahren und das graphische Verfahren von Karnaugh-Veitch (KV-Diagramme).
Nähre Informationen zum Quine-McCluskey-Verfahren finden Sie unter anderem in [1], [3] und [4], zum KV-Diagramm in [1] und [2]. Auf diese Verfahren wird hier nicht weiter eingegangen.

[1] siehe Anhang B

1.4 Synthese von Schaltnetzen

Mit dem in Kapitel 1.3 behandelten Beispiel zur Einführung der Normalformen haben wir bereits eine sogenannte **Schaltnetzsynthese** durchgeführt, indem wir zu einer konkreten Aufgabe ein Schaltnetz entwickelt haben, das die gestellten Anforderungen erfüllt.

1.4.1 Vorgehensweise

Bei der Synthese geht es also darum, zu einem vorgegebenen Problem ein Schaltnetz zu entwickeln, dass den Problembedingungen genügt. Dazu muss erst einmal die Aufgabe so formuliert werden, dass hieraus die Funktionstabelle erstellt werden kann. Danach ist es möglich, die disjunktive oder konjunktive Normalform aufzustellen und eventuell zu vereinfachen. Anschließend wird geprüft, ob der so erhaltene Ausdruck mit den zur Verfügung stehenden Bauelementen realisiert werden kann. Unter Umständen ergeben sich Verknüpfungen mit mehr Eingängen als bei den vorliegenden Bauelementen vorhanden sind. So besitzen z.B. alle UND- bzw. ODER-Glieder im CAD-System nur drei Eingänge. In der zweiten Lösung der obigen Aufgabe müssen aber fünf Terme mit UND zusammengefasst werden. Man erreicht das, indem man drei UND-Glieder nach folgender Abbildung zusammenschaltet. Dabei werden nicht alle Eingänge genutzt.

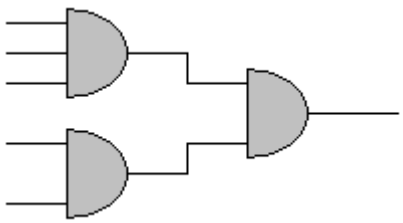


Abb. 1.4.1-1 UND-Schaltung mit 5 Eingängen

Bei Ausnutzung aller zur Verfügung stehenden Eingänge kann man mit vier UND-Gliedern eine UND-Schaltung mit neun Eingängen herstellen.

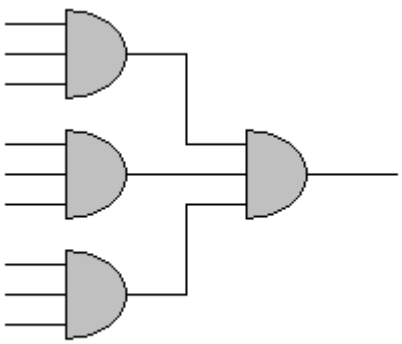


Abb. 1.4.1-2 UND-Schaltung mit 9 Eingängen

1.4.2 Übungsaufgabe

Wechselschaltung

Die Beleuchtung eines Treppenhauses soll mit den drei Schaltern S_1, S_2 und S_3 an verschiedenen Stellen jederzeit ein- bzw. ausgeschaltet werden können.

Eingangsvariable des zu entwickelnden Schaltnetzes sind die Zustände der Schalter S_1, S_2 und S_3. Ausgangsvariable ist der Zustand der Beleuchtung B.

Es soll bedeuten:
$B = 0 \rightarrow$ Beleuchtung aus
$B = 1 \rightarrow$ Beleuchtung an
$S_1 = 0 \rightarrow$ Schalter S_1 nicht betätigt
$S_1 = 1 \rightarrow$ Schalter S_1 betätigt
$S_2 = 0 \rightarrow$ Schalter S_2 nicht betätigt
$S_2 = 1 \rightarrow$ Schalter S_2 betätigt
$S_3 = 0 \rightarrow$ Schalter S_3 nicht betätigt
$S_3 = 1 \rightarrow$ Schalter S_3 betätigt

Entwickeln Sie Schaltnetze in konjunktiver und disjunktiver Normalform hierzu.
(Eine Lösung finden Sie im Anhang C)

1.5 Technische Realisierung der Grundfunktionen

Im bisherigen Text wurde deutlich, dass sich jede Schaltfunktion aus den Grundfunktionen UND, ODER und NICHT zusammensetzen lässt. Damit eine technische Apparatur ein Funktionsergebnis selbsttätig ermitteln kann, müssen lediglich diese drei Grundfunktionen mit entsprechenden technischen Bauelementen realisiert werden.

Eine einfach zu verstehende Realisierung ist mit Relais möglich. So beinhaltete der von Konrad Zuse im Jahre 1941 fertiggestellte erste programmierbare Rechenautomat der Welt ca. 2600 Fernmelderelais.

1.5.1 Kontaktlogik

Ein Relais ist ein elektromechanisches Bauelement, das vereinfacht aus einer Spule mit Eisenkern (Elektromagnet) und einem Anker besteht. Sobald ein Strom durch die Spule fließt, wird der Anker durch die auftretende magnetische Kraft angezogen. Dabei betätigt er einen Schalter (Kontakt). Die folgende Abbildung zeigt das genormte Schaltsymbol.

Abb. 1.5.1-1 a) Schaltsymbol eines Relais b) Zustand mit angezogenem Anker

Die Binärzustände 0 und 1 werden hier folgendermaßen interpretiert:
„1“ bedeutet „es fließt Strom“
„0“ bedeutet „es fließt kein Strom“

Damit ist ohne weiteres erkennbar, dass die nachfolgende Schaltung eine Realisierung der UND-Funktion darstellt.

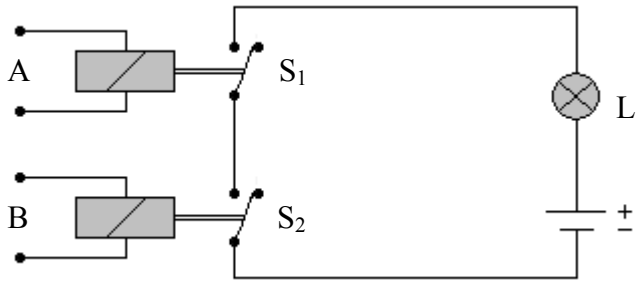


Abb. 1.5.1-2 Realisierung der UND-Funktion (Reihenschaltung der Kontakte)

Die Lampe L leuchtet nur dann, wenn im Stromkreis ein Strom fließt. Das ist aber nur möglich, wenn Schalter S_1 **und** Schalter S_2 geschlossen sind. Ein Schalter ist aber nur dann geschlossen, wenn im entsprechenden Relais ein Strom durch die Spule fließt.

Also:
Im Lampenstromkreis liegt nur dann Zustand 1 vor, wenn in Relais A **und** Relais B Zustand 1 vorliegt.

Die nächste Schaltung stellt die Realisierung der ODER-Funktion dar.

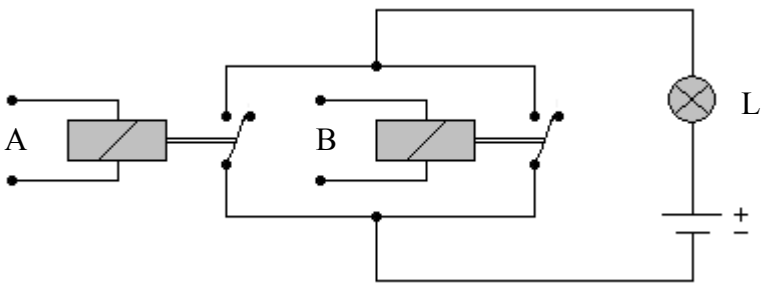


Abb. 1.5.1-3 Realisierung der ODER-Funktion (Parallelschaltung der Kontakte)

Im Lampenstromkreis liegt Zustand 1 vor, wenn in Relais A **oder** Relais B Zustand 1 vorliegt.

Hier wird nochmals deutlich, dass ODER im nichtausschließenden Sinne zu verstehen ist.

Ebenso einfach lässt sich die NICHT-Funktion realisieren.

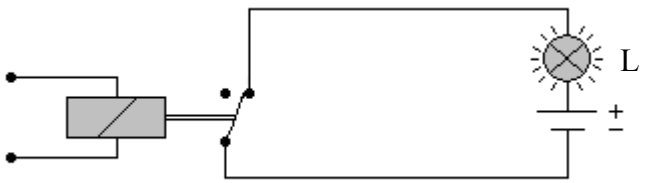


Abb. 1.5.1-4 Realisierung der NICHT-Funktion

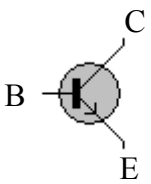
Die Lampe brennt nur, wenn der Anker des Relais **nicht** angezogen ist, d.h. wenn im Relais der Zustand 0 vorliegt.

Mit diesen Realisierungen der Grundfunktionen wäre es prinzipiell – von schaltungstechnischen Feinheiten abgesehen – möglich, einen Computer zusammenzubauen, der allerdings wegen der relativ großen Schaltzeiten der Relais äußerst langsam wäre und außerdem erheblichen Lärm verursachen würde. In der heutigen Datentechnik hat die oben dargestellte Kontaktlogik keine Bedeutung mehr. Seit Mitte der 50er Jahre werden Transistoren als kontaktlose Schalter verwendet.

Transistoren schalten schnell und geräuschlos, sind wartungsfrei und haben eine hohe Lebensdauer.

1.5.2 Transistor-Logik[1]

Der Transistor ist ein Halbleiterbauelement, das bei entsprechender Ansteuerung als kontaktloser Schalter verwendet werden kann. Zum Verständnis der Prinzipschaltungen zur Realisierung der Grundfunktionen muss man nur diese Schaltereigenschaft des Transistors kennen.
Ein Transistor besitzt drei Anschlüsse, die als **Basis**, **Emitter** und **Kollektor** bezeichnet werden. Die folgende Abbildung zeigt das Schaltsymbol eines npn[2]-Transistors.



B = Basis
E = Emitter
C = Kollektor

Der Pfeil am Emitteranschluss zeigt die technische Stromrichtung an (von + nach -)

Abb. 1.5.2-1 npn-Transistor

[1] Logische Schaltungen, in denen man für jeden Eingang einen Transistor verwendet, werden als Transistor-Transistor-Logik (TTL) oder einfacher als Transistor-Logik bezeichnet.
[2] Diese Bezeichnung hängt mit dem inneren Aufbau zusammen.

1.5.2.1 Transistor als Schalter

Die folgenden Schaltungen sollen die Schaltereigenschaft verdeutlichen.

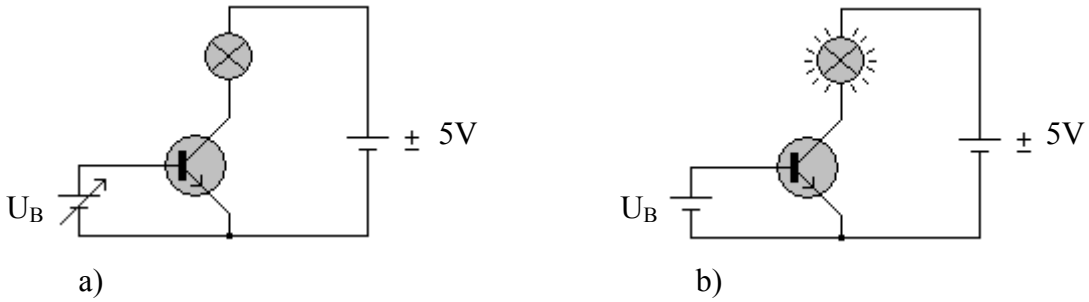


Abb. 1.5.2-2 Transistor als Schalter a) $U_B < 0{,}6$ V b) $U_B \approx 0{,}8$ V

Wir beginnen mit der Abbildung a). Die regelbare Spannungsquelle U_B sei zu Beginn auf 0 V eingestellt. Die Lampe im Kollektor-Emitter-Kreis leuchtet nicht, d.h. die Kollektor-Emitter-Strecke hat einen sehr großen Widerstand. Sie verhält sich wie ein geöffneter Schalter. Man sagt, der Transistor „sperrt“.

Bei einer langsamen Erhöhung der Basis-Emitter-Spannung bleibt dieser Zustand so lange bestehen, bis die sogenannte Schwellenspannung (ca. 0,6 Volt bei Silizium-Transistoren) überschritten wird.

Bei geringfügiger Überschreitung der Schwellenspannung leuchtet die Lampe mit voller Leistung. In diesem Bereich hat die Kollektor-Emitter-Strecke einen sehr geringen Widerstand. Sie verhält sich wie ein geschlossener Schalter. Man sagt, der Transistor ist „durchgeschaltet“.

Eine weitere Erhöhung der Basis-Emitter-Spannung ändert nichts an diesem Zustand. Allerdings darf diese Spannung einen vom Transistortyp abhängigen Grenzwert nicht überschreiten, da ansonsten der Transistor zerstört würde.

Mit diesen Kenntnissen lässt sich die nachstehende Schaltung analysieren. Den Binärzuständen 0 und 1 werden jetzt die auf Seite 5 erwähnten Spannungspegel der TTL-Technik zugeordnet.

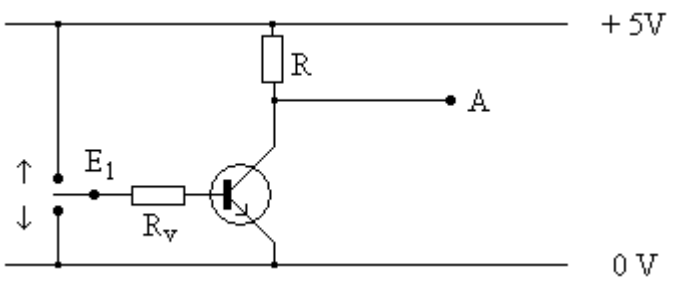


Abb. 1.5.2-3 Transistorschaltung

Der Eingang E_1 kann über einen Schalter mit dem Minuspol (0V ≙ 0-Zustand) oder mit dem Pluspol (5V ≙ 1-Zustand) der Spannungsquelle verbunden werden.

Der Widerstand R_v in der Basiszuleitung ist notwendig, damit bei Verbindung mit dem Pluspol der Grenzwert für die Basis-Emitter-Spannung nicht überschritten wird. Er muss allerdings so bemessen sein, dass sich in diesem Fall eine Basis-Emitter-Spannung einstellt, die leicht über der Schwellenspannung liegt.

Wir legen den Schalter jetzt nach unten um und bringen den Eingang damit in den 0-Zustand. Zwischen Basis und Emitter liegt dann eine Spannung von 0V an. Der Transistor sperrt somit und es kann kein Strom in der Kollektor-Emitter-Leitung fließen. Damit fällt nach dem Ohmschen Gesetz $U = R \cdot I$ keine Spannung am Widerstand R ab und der Ausgang A hat das Potential 5V, befindet sich also im 1-Zustand.

Wir bringen jetzt den Eingang in den 1-Zustand, indem wir den Schalter nach oben umlegen. Die Spannung zwischen Basis und Emitter liegt dann über der Schwellenspannung und der Transistor ist durchgeschaltet. Der Widerstand der Kollektor-Emitter-Strecke ist jetzt so gering, dass das Potential des Ausgangs A nur knapp über 0V liegt und sich der Ausgang damit im 0-Zustand befindet.

Zusammenfassend haben wir folgende Zusammenhänge zwischen Ein- und Ausgang vorliegen:

E_1	A
0	1
1	0

Das ist die Funktionstabelle der NICHT-Verknüpfung. Die obige Schaltung stellt somit eine Realisierung der NICHT-Funktion dar.

1.5.2.2 Reihenschaltung zweier Transistoren

Die Realisierung der UND-Funktion durch eine Reihenschaltung zweier Schalter in Kapitel 1.5.1 legt die Vermutung nahe, dass eine Schaltung mit zwei in Reihe liegenden Transistoren ein ähnliches Verhalten zeigt. Wir untersuchen deshalb die nachfolgende Schaltung. Die Schalter an den Eingängen E_1 und E_2 können wir uns an dieser Stelle sparen.

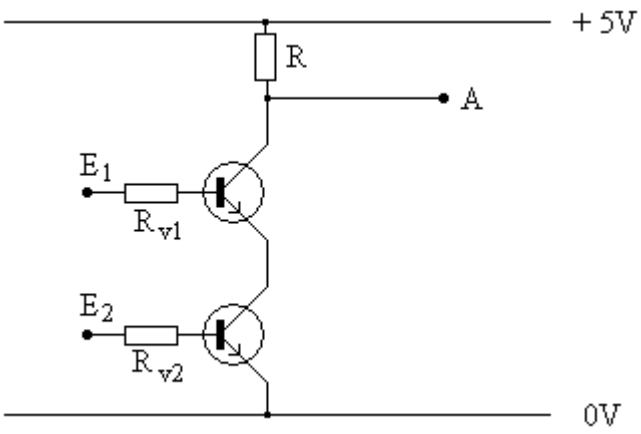


Abb. 1.5.2-4 Reihenschaltung zweier Transistoren

Solange kein Strom über den Widerstand R fließt, hat der Ausgang A das Potential 5V und befindet sich damit im 1-Zustand. Nur in dem Fall, dass beide Transistoren durchgeschaltet sind, liegt das Potential des Ausganges knapp über 0V (≙ 0-Zustand). Das führt zu folgender Funktionstabelle:

E_1	E_2	A
0	0	1
0	1	1
1	0	1
1	1	0

im Vergleich dazu die Tabelle der UND-Funktion

E_1	E_2	$E_1 \wedge E_2$
0	0	0
0	1	0
1	0	0
1	1	1

Wenn wir diese Funktion der Transistorschaltung mit der UND-Funktion vergleichen, stellen wir fest, dass die obige Schaltung gerade die Negation der UND-Funktion liefert. (NICHT UND).

Wegen der einfachen technischen Realisierung wird diese Schaltung auch als eigenständiger Digital-Baustein hergestellt. Er wird als NAND-Gatter bezeichnet (gebildet aus: **N**OT **AND**).

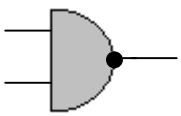


Abb. 1.5.2-5 Schaltsymbol eines NAND-Gatters

Um aus der NAND-Schaltung eine UND-Schaltung zu erhalten, brauchen wir nur noch den Ausgang mit einer NICHT-Schaltung zu negieren. Das führt zu folgendem Schaltplan:

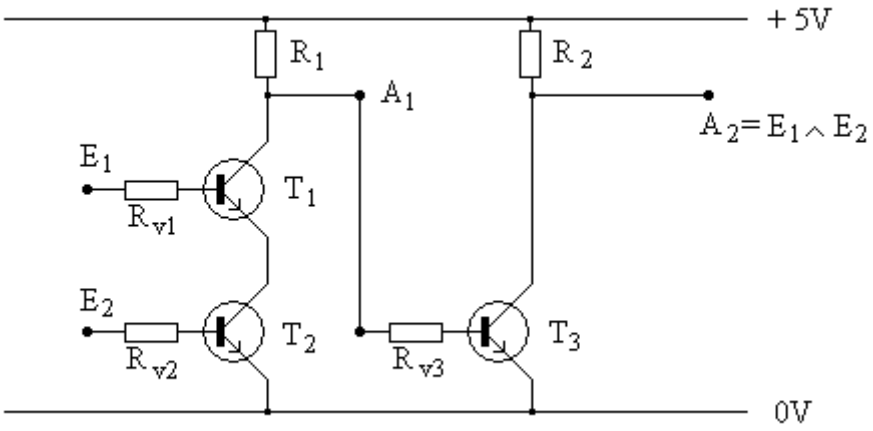
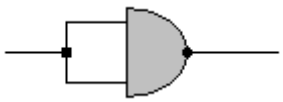


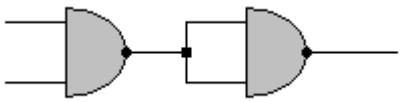
Abb. 1.5.2-6 UND-Schaltung in TTL-Technik

NAND-Gatter sind universelle Logik-Bausteine, da sich damit UND-, ODER- und NICHT-Glieder realisieren lassen und somit jede Schaltfunktion unter ausschließlicher Verwendung von NANDs aufgebaut werden kann.

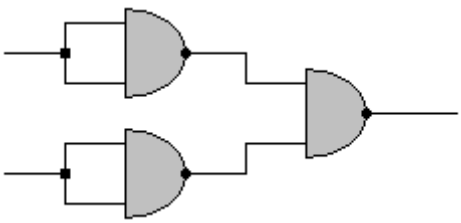
Realisierung eines NICHT-Gliedes mit einem NAND-Gatter:



Realisierung eines UND-Gliedes mit zwei NAND-Gattern:



Realisierung eines ODER-Gliedes mit drei NAND-Gattern:



Aufgabe 1.1: Machen Sie sich die einzelnen Realisierungen mit Hilfe von Funktionstabellen klar.

1.5.2.3 Parallelschaltung zweier Transistoren

Der Vollständigkeit halber wollen wir jetzt auch noch untersuchen, welches Verhalten eine Parallelschaltung zweier Transistoren zeigt.

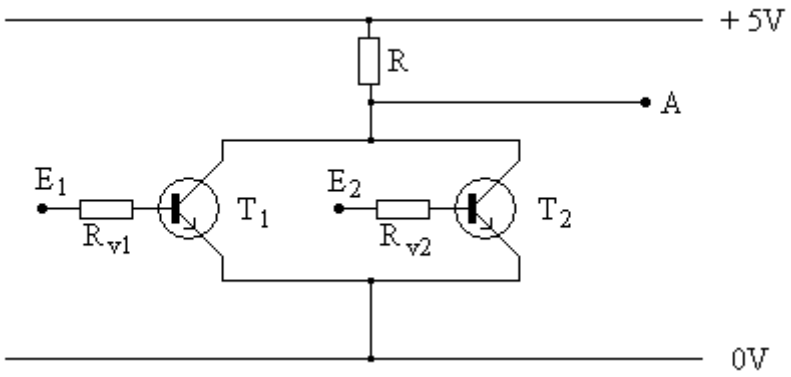


Abb. 1.5.3-1 Parallelschaltung zweier Transistoren

Wenn kein Strom über den Widerstand R fließt, hat der Ausgang A das Potential 5V und befindet sich damit im 1-Zustand. Das nur dann der Fall, wenn beide Transistoren sperren, also an den Eingängen E_1 und E_2 0-Zustand vorliegt.

E_1	E_2	A
0	0	1
0	1	0
1	0	0
1	1	0

im Vergleich dazu die Tabelle der ODER-Funktion

E_1	E_2	$E_1 \vee E_2$
0	0	0
0	1	1
1	0	1
1	1	1

Wenn wir diese Funktion der Transistorschaltung mit der ODER-Funktion vergleichen, stellen wir fest, dass die obige Schaltung gerade die Negation der ODER-Funktion liefert. (NICHT ODER).

Auch diese Schaltung wird als eigenständiger Baustein hergestellt und als NOR-Gatter bezeichnet (gebildet aus: **N**OT **OR**).

Auch mit NOR-Gattern lassen sich UND-, ODER- und NICHT-Glieder realisieren und damit jede Schaltfunktion unter ausschließlicher
Verwendung von NORs aufbauen.

Aufgabe 1.2: Realisieren Sie die drei Grundfunktionen mit NOR-Gattern.
(Lösung im Anhang C)

2 Schaltnetze

Mit den Kenntnissen aus Kapitel 1 können wir nun damit beginnen, den Aufbau wichtiger Funktionseinheiten von Computern zu erarbeiten. Sie sollten die entwickelten Schaltungen sofort mit *LOCAD* aufbauen und testen. Durch die deutliche Hervorhebung der Leitungszustände wird es Ihnen leicht fallen, die Funktionsweise der einzelnen Schaltungen zu verstehen. Als erstes soll geklärt werden, wie eine Maschine in der Lage ist, Rechnungen auszuführen.

2.1 Rechenschaltungen

Multiplikation bzw. Division können auf wiederholte Addition bzw. Subtraktion zurückgeführt werden. Wie wir weiter sehen werden, lässt sich auch die Subtraktion auf die Addition zurückführen, so dass eigentlich nur ein Addierwerk benötigt wird. Rechenoperationen höherer Ordnung werden durch Rechenprogramme mit Hilfe der vier Grundrechenarten durch Reihenbildung und/oder Iterationsverfahren erzeugt. Ein Addierwerk ist damit eine wichtige Einheit im Rechenwerk eines Computers. Mit den bisher bekannten Grundbausteinen soll jetzt ein einfaches Addierwerk realisiert werden.

Da die Ein- und Ausgänge digitaler Schaltungen nur die zwei Binärzustände 0 und 1 annehmen können, ist es zweckmäßig, zur Darstellung von Zahlen das Zweiersystem
(Dualsystem) zu verwenden.

2.1.1 Das Dualsystem

Das Dualsystem ist ein Stellenwertsystem mit der Basis 2. So wie im bekannten Dezimalsystem mit der Basis 10 die zehn Ziffern 0, 1, ... 9 zur Darstellung einer Zahl zur Verfügung stehen, sind im Dualsystem nur zwei Ziffern verfügbar, die mit 0 und 1 bezeichnet werden.

In einem Stellenwertsystem hat jede Ziffer neben ihrem Zahlenwert noch einen Stellenwert. Man erhält den Zahlenwert einer Ziffernkombination, indem man jede Ziffer mit ihrem Stellenwert multipliziert und die so gebildeten Zahlen addiert.

Im Dezimalsystem steht somit die Ziffernkombination 345 abkürzend für
$3 \cdot 10^2 + 4 \cdot 10^1 + 5 \cdot 10^0$
(3 Hunderter + 4 Zehner + 5 Einer)

Entsprechendes gilt für das Zweiersystem. Hier treten die Zweierpotenzen an die Stelle der Zehnerpotenzen. Die Stellenwerte lauten demnach von links nach rechts: ... 2^4 2^3 2^2 2^1 2^0.

Im Dualsystem steht die Ziffernkombination 1101 abkürzend für
$1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$
($1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1$)

Der Dualzahl 1101 entspricht also die Dezimalzahl 13.

In der folgende Tabelle sind die ersten 15 Zahlen in beiden Systemen einander gegenübergestellt.

Dezimalsystem		Dualsystem			
10^1	10^0	2^3	2^2	2^1	2^0
10	1	8	4	2	1
	0				0
	1				1
	2			1	0
	3			1	1
	4		1	0	0
	5		1	0	1
	6		1	1	0
	7		1	1	1
	8	1	0	0	0
	9	1	0	0	1
1	0	1	0	1	0
1	1	1	0	1	1
1	2	1	1	0	0
1	3	1	1	0	1
1	4	1	1	1	0
1	5	1	1	1	1

Zur Darstellung der Dezimalzahl 16 muss im Dualsystem bereits die fünfte Stelle (Sechzehnerstelle) benutzt werden.
Um die dreistellige Dezimalzahl 128 darzustellen, benötigt man im Dualsystem schon 8 Stellen. Mit diesen 8 Stellen kann man maximal die Dualzahl 11111111 darstellen, die den Dezimalwert 255 hat.

Allgemein gilt, dass die größte n-stellige Dualzahl den Dezimalwert $2^n - 1$ hat.

Da bei einer Zahl, die nur 0 und 1 als Ziffern enthält, nicht ohne weiteres klar ist, in welchem System sie angegeben ist, schreiben wir in Zweifelsfällen die Basis des Stellenwertsystems als Index an die Zahl.

Also: $11_{10} = 1011_2$

Wie im Dezimalsystem werden auch im Dualsystem die Wertigkeiten der Nachkommastellen gebildet, indem die Basis mit den entsprechenden negativen Exponenten potenziert wird.

Beispiele: $12{,}35_{10} = 1\ 10^1 + 2 \cdot 10^0 + 3 \cdot 10^{-1} + 5 \cdot 10^{-2}$

$0{,}1_2 = 1 \cdot 2^{-1}$
$= 0{,}5_{10}$

$101{,}11_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-2}$
$= 4 + 0 + 1 + 0{,}5 + 0{,}25$
$= 5{,}75_{10}$

Im folgenden beschränken wir uns auf ganze Zahlen, da das Dualsystem üblicherweise nicht zur Darstellung gebrochener Dezimalzahlen verwendet wird. Hierzu wird normalerweise ein anderes Codierungsverfahren benutzt.

2.1.1.1 Umwandlung Dual → Dezimal

Nach den obigen Ausführungen ist klar, wie eine Dualzahl in eine Dezimalzahl umgewandelt wird.

Beispiel:

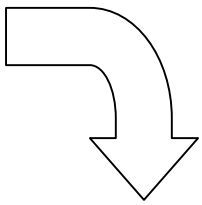
$$\begin{aligned} 100111_2 &= 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= 32 + 0 + 0 + 4 + 2 + 1 \\ &= 39_{10} \end{aligned}$$

2.1.1.2 Umwandlung Dezimal → Dual

Der umgekehrte Umwandlungsvorgang ist etwas ungewohnter. Ein übersichtliches Verfahren ist die sogenannte *Restwert-Methode*. Hierbei wird die umzuwandelnde Dezimalzahl durch 2 dividiert und das Ergebnis und der Rest werden notiert. Das erhaltene Ergebnis wird wieder durch 2 dividiert und das neue Ergebnis mit Rest notiert usw. Das Verfahren wird so lange fortgesetzt, bis sich das Divisionsergebnis 0 ergibt. Die Reste ergeben dann von unten nach oben gelesen die gesuchte Dualzahl.

Beispiel: Umwandlung von 94_{10} in eine Dualzahl

```
94 : 2 = 47 Rest 0
47 : 2 = 23 Rest 1
23 : 2 = 11 Rest 1
11 : 2 =  5 Rest 1
 5 : 2 =  2 Rest 1
 2 : 2 =  1 Rest 0
 1 : 2 =  0 Rest 1
```



1 0 1 1 1 1 0

Ergebnis: $94_{10} = 1011110_2$

2.1.1.3 Addition im Dualsystem

Die Addition zweier Zahlen im Dualsystem wird genauso durchgeführt wie die Addition im Dezimalsystem. Zu beachten ist lediglich, dass im Dualsystem „1“ + „1“ nicht „2“, sondern „10“ ergibt, d.h. eine 0 in der Summenzeile und eine 1 als Übertrag auf die nächste Stelle.

Additionsregeln im Dualsystem:

```
0 + 0 =  0
0 + 1 =  1
1 + 0 =  1
1 + 1 = 10
```

Beispiel:

```
        10010110      Summand 1
   +      111011      Summand 2
----------------------
        11111         Übertrag
======================
        11010001      Summe
```

2.1.2 Halbaddierer

Zur Entwicklung einer Schaltung, die eine Addition ausführen kann, beginnen wir mit dem einfachsten Fall, der Addition zweier einstelliger Dualzahlen. Nach den obigen Additionsregeln ergibt sich folgende Funktionstabelle:

a	b	ü	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Die Schaltung muss für jede Dualziffer einen Eingang und für die Summe und den Übertrag jeweils einen Ausgang haben. Mit den Kenntnissen aus Kapitel 1 können wir diese Tabelle sofort in die folgende formale Schreibweise umsetzen:

$$\ddot{u} = a \wedge b$$
$$s = (\bar{a} \wedge b) \vee (a \wedge \bar{b})$$

Der Übertrag wird somit durch eine einfache UND-Schaltung realisiert. Die Summe ist etwas komplizierter zu bilden. Unter Beachtung der Prioritäten der Operatoren ergibt sich das in b) dargestellte Schaltnetz:

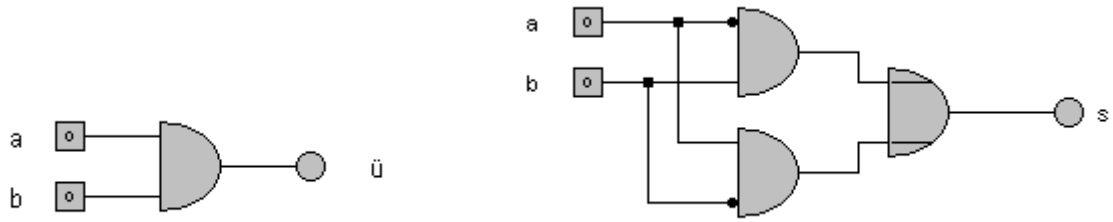


Abb. 2.1.2-1 Schaltungen für Übertrag und Summe

Die beiden Teilschaltungen können zu einem Gesamtschaltnetz zusammengefasst werden. Man erhält damit eine Schaltung, die die Addition zweier Dualziffern realisiert. Da diese Schaltung nicht in der Lage ist, einen Übertrag aus einer vorhergehenden Stelle zu verarbeiten, kann sie bei der Addition von Dualzahlen auch nur an der Einerstelle eingesetzt werden. Sie hat daher den Namen ***Halbaddierer*** .

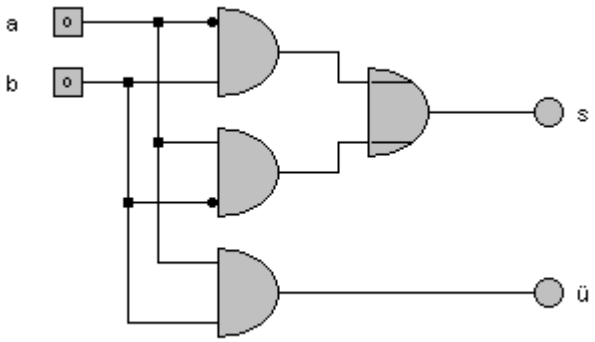


Abb. 2.1.2-2 Halbaddierer

Aufgabe 2.1: Bauen Sie diese Schaltung auf und überprüfen Sie ihre Funktionsweise.

Nachdem wir uns vergewissert haben, dass die obige Schaltung die Addition richtig ausführt, werden wir im folgenden einen Halbaddierer nicht mehr aus einzelnen Grundschaltungen zusammensetzen, sondern im Menü *Bauteile* einen fertigen Halbaddierer-Baustein verwenden.

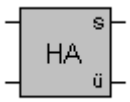


Abb. 2.1.2-3 Blockschaltsymbol eines Halbaddierers

2.1.3 Volladdierer

Da bei der Addition mehrstelliger Dualzahlen ein eventueller Übertrag aus vorhergehenden Stellen berücksichtigt werden muss, brauchen wir auch eine Schaltung, die drei einstellige Dualzahlen addieren kann.

Die folgende Tabelle zeigt die benötigte Zuordnung:

a	b	c	ü	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Hieraus lassen sich die Ausdrücke für ü und s in disjunktiver Normalform entwickeln:

$$\ddot{u} = (\bar{a} \wedge b \wedge c) \vee (a \wedge \bar{b} \wedge c) \vee (a \wedge b \wedge \bar{c}) \vee (a \wedge b \wedge c)$$

$$s = (\bar{a} \wedge \bar{b} \wedge c) \vee (\bar{a} \wedge b \wedge \bar{c}) \vee (a \wedge \bar{b} \wedge \bar{c}) \vee (a \wedge b \wedge c)$$

Eine entsprechende Umsetzung in eine Schaltung liefert das gewünschte Schaltnetz mit dem Namen ***Volladdierer***.

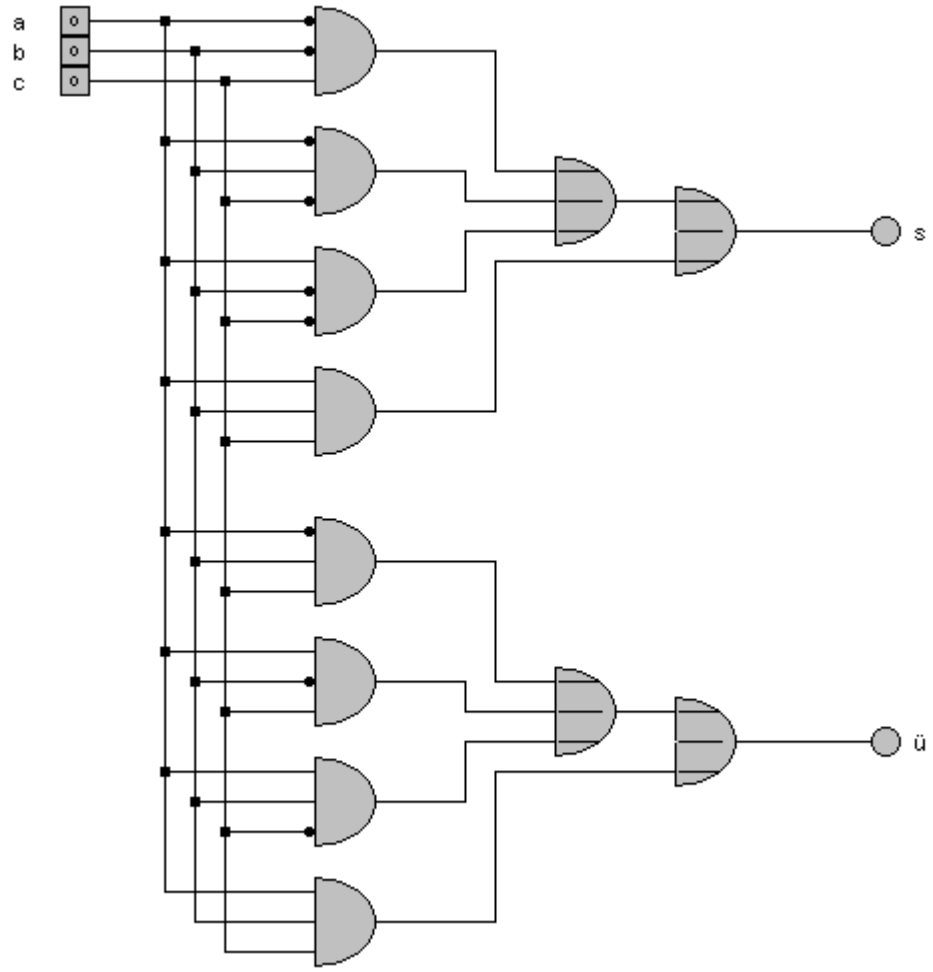


Abb. 2.1.3-1 Volladdierer

Aufgabe 2.2: Bauen oder laden Sie diese Schaltung und überprüfen Sie ihre Funktionsweise. Sicherlich haben Sie bemerkt, dass die Schaltung vereinfacht werden kann. Führen Sie die Vereinfachung durch und testen Sie die Änderung.

Aufgabe 2.3: Bauen Sie auch die folgende aus zwei Halbaddierern und einem ODER-Glied bestehende Schaltung auf und zeigen Sie, dass damit ebenfalls eine Volladdition realisiert wird.

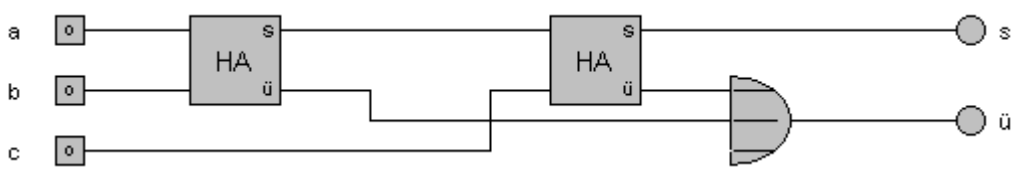


Abb. 2.1.3-2 Volladdierer mit zwei Halbaddierern und einem Oder-Glied

Wir werden im folgenden fertige Volladdierer-Bausteine verwenden. Diese stehen im Menü *Bauteile* zur Verfügung.

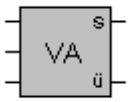


Abb. 2.1.3-3 Blockschaltsymbol eines Volladdierers

2.1.4 Paralleladdierer

Mit den bisher entwickelten Rechenschaltungen können wir nun einen Addierer für zwei mehrstellige Dualzahlen aufbauen. Wir wollen uns zur Verringerung des Schaltungsaufwandes auf 4 Stellen beschränken.
Für die Einerstelle wird nur ein Halbaddierer benötigt, da hier nur zwei Dualziffern zu addieren sind, für jede weitere Stelle brauchen wir einen Volladdierer. Jeder Volladdierer muss einen eventuellen Übertrag aus der vorhergehenden Stelle verarbeiten. Diese Überlegung führt zu folgendem Schaltbild.

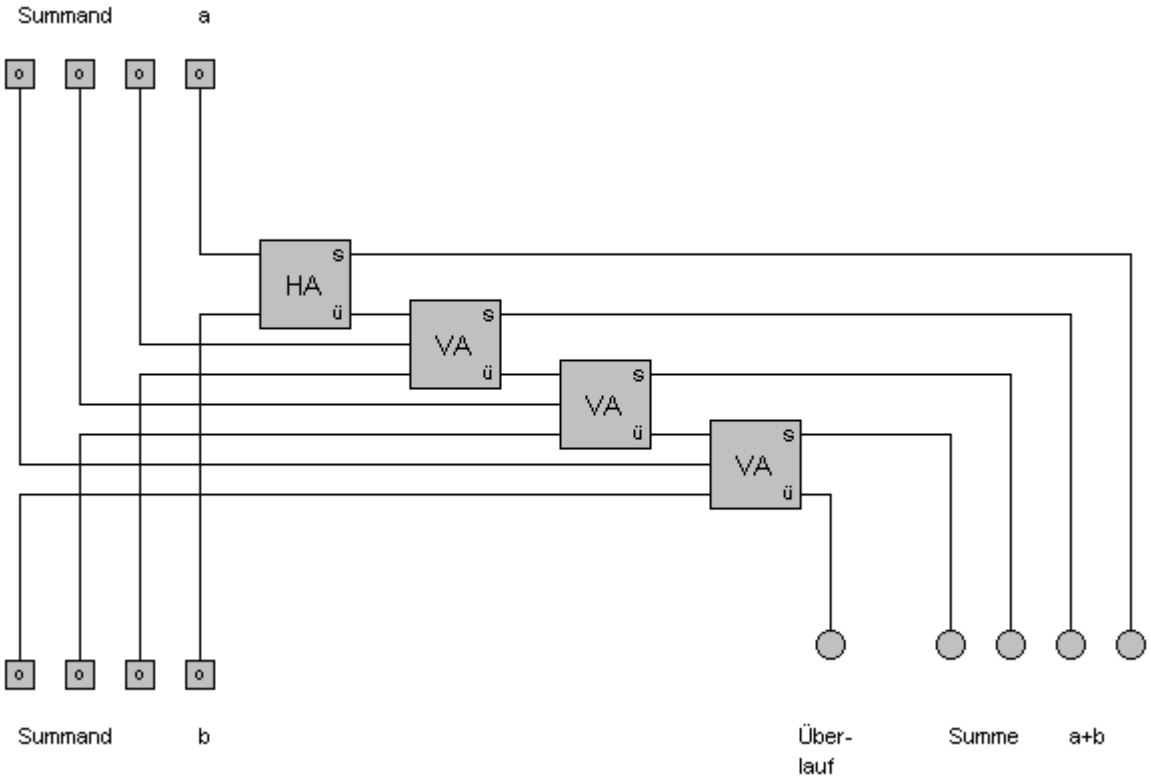


Abb. 2.1.4-1 Paralleladdierer

Der Name *Paralleladdierer* kommt daher, dass die Berechnung im Gegensatz zum Serienaddierer in einem Rechenschritt durchgeführt wird.

Die Kreise an den Ausgangsleitungen sollen Leuchtdioden (LED = Licht emittierende Diode) darstellen, die den Leitungszustand noch etwas deutlicher hervorheben. Leuchtdioden können mit *LOCAD* auf bestehende Leitungen aufgesetzt werden. Sie stehen im Menü *Bauteile* als kleine und große LEDs zur Verfügung. In der obigen Schaltung wurden kleine LEDs verwendet.

Der Übertrag des letzten Volladdierers zeigt eine eventuelle Bereichsüberschreitung an. Eine Bereichsüberschreitung liegt vor, wenn das Ergebnis nicht mehr mit den zur Verfügung stehenden Dualstellen ausgedrückt werden kann.
Mit vier Dualstellen kann man den Zahlbereich von 0 bis 15 darstellen. Werden dezimal 9 und 7 addiert, so kann das Ergebnis nicht mehr mit den vier Dualstellen dargestellt werden und es liegt eine Bereichsüberschreitung vor.

Aufgabe 2.4: Bauen oder laden Sie den Paralleladdierer und testen Sie ihn.

2.1.5 Parallelsubtrahierer

Ebenso wie der Volladdierer entworfen wurde, ließe sich ein Vollsubtrahierer entwickeln, mit dem dann ein Parallelsubtrahierer aufgebaut werden könnte. Dieser Weg wird nicht weiter beschritten. Statt dessen wollen wir von der bereits erwähnten Tatsache, dass sich alle Rechenarten auf die Addition zurückführen lassen, Gebrauch machen, um ein Subtrahierwerk zu entwickeln.

Als erstes muss gezeigt werden, wie die Subtraktion - und damit auch die Division als wiederholte Subtraktion - auf eine Addition zurückgeführt werden kann.

2.1.5.1 Subtraktion von Dualzahlen

Die Subtraktion kann über eine Addition durchgeführt werden.

Beispiel: $7 - 4 = 7 + (-4) = 3$

Die Subtraktion der Zahl 4 wird hier durch die Addition der Gegenzahl - 4 (inverses Element bezüglich der Addition) ausgeführt. Wir haben damit zwar die Subtraktion umgangen, uns dafür aber das Problem eingehandelt, zu einer Zahl ihre Gegenzahl zu ermitteln.

Zu jeder Zahl gibt es bezüglich der Addition eine Gegenzahl und es besteht folgender Zusammenhang:

$$\text{Zahl} + \text{Gegenzahl} = 0$$

Die Gegenzahlen zu den positiven Zahlen sind die negativen. Wir müssen also erst einmal überlegen, wie negative Zahlen im Dualsystem dargestellt werden können. Hierbei spielt die Tatsache, dass jedes Rechenwerk nur mit einer endlichen Anzahl von Stellen zur Darstellung der Zahlen arbeiten kann, eine entscheidende Rolle.

2.1.5.2 Negative Dualzahlen

Ein Zahlensystem mit begrenzter Stellenzahl hat eine interessante Eigenschaft:

> ***Beim Weiterzählen kommt man irgendwann einmal wieder auf Null***
> (siehe Kilometerzähler beim Auto)

Die Zahlengerade schließt sich hier sozusagen zu einem Zahlenkreis.

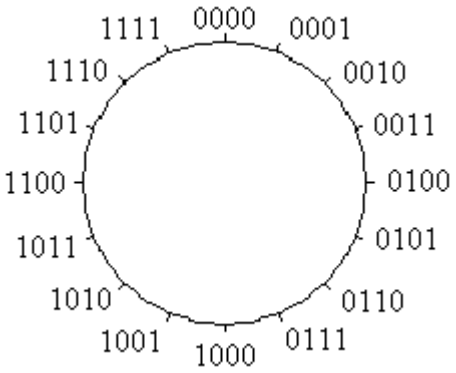


Abb. 2.1.5.2-1 Zahlenkreis für vierstellige Dualzahlen

Gehen wir im Uhrzeigersinn den Kreis entlang, so kommen wir irgendwann einmal zur größten darstellbaren Zahl (hier $2^4 - 1 = 15_{10} = 1111_2$) und im nächsten Schritt sofort wieder zu Null.

Geht man auf der Zahlengerade von 0 aus nach links, so gelangt man in den negativen Bereich. Entsprechend müsste man im Zahlenkreis bei Bewegung gegen den Uhrzeigersinn zu den negativen Zahlen kommen und damit 1111_2 als -1 , 1110_2 als -2, 1101_2 als -3 usw. interpretieren.

Wir überprüfen, ob eine solche Interpretation überhaupt möglich ist, indem wir zu einigen Zahlen aus dem rechten Halbkreis die auf gleicher Höhe stehenden Zahlen aus dem linken Halbkreis - die vermutlichen Gegenzahlen - addieren.

	1111	0010	0011
	+ 0011	+ 1110	+ 1101
	----------	----------	----------
Übertrag	1 111	1 11	1 111
	=======	========	========
Summe	(1)0000	(1)0000	(1)0000

In diesen Beispielen ergibt sich als Summe jeweils 0000, da die 1 auf der fünften Stelle wegen der auf vier Stellen beschränkten Zahlendarstellung keine Bedeutung hat. Die Untersuchung weiterer Beispiele führt zum gleichen Ergebnis.

An dieser Zahlenkreiseigenschaft ändert sich auch nichts, wenn zur Zahlendarstellung mehr als vier Stellen benutzt werden, hierbei wird lediglich der Zahlenkreis größer.

Eine Sonderrolle spielen die Zahlen 0000 und 1000. Sie sind mit ihren Gegenzahlen identisch. Diese beiden Zahlen markieren die Trennlinie zwischen den positiven und negativen Zahlen. üblicherweise zählt man $1000_2 = 8_{10}$ zu den negativen Zahlen und kann damit das Vorzeichen einer Zahl direkt an der ersten Stelle ablesen:

1 an erster Stelle $\triangleq$ negative Zahl
0 an erster Stelle $\triangleq$ positive Zahl

Diese Interpretation der Dualzahlen hat zur Konsequenz, dass bei vierstelliger Darstellung der Zahlenbereich nicht mehr von 0 bis 15 reicht, sondern von -8 bis +7. Der Zahlenbereich umfasst aber nach wie vor 16 Zahlen. Bei 8-stelliger Darstellung reicht der Zahlenbereich demnach von -128 bis +127.

Es bleibt noch zu klären, wie man auf einfache Weise die Gegenzahl zu einer vorgegebenen Zahl ermittelt, ohne dazu einen Zahlenkreis aufzuzeichnen.
Man erhält zu einer gegebenen Zahl die Gegenzahl durch Ausführung der folgenden beiden Schritte:

1. Man bildet das sogenannte **Einerkomplement** der Zahl, indem man Stelle für Stelle 1 durch 0 und 0 durch 1 ersetzt.

   Beispiel: vorgegeben: 0101
   Einerkomplement: 1010

2. Durch Addition von 1 zum Einerkomplement wird das sogenannte **Zweierkomplement** gebildet.

   Beispiel: Einerkomplement: 1010
   \+ 1

   Zweierkomplement: 1011

   Die Dualzahl 1011 ist damit das Zweierkomplement zu 0101.

Das Zweierkomplement einer Dualzahl stimmt mit der Gegenzahl bezüglich der Addition überein. Wir erhalten also die Gegenzahl zu einer vorgegebenen Zahl durch Zweierkomplement-Bildung.

Merkregel:

Das ***Einerkomplement*** einer Dualzahl erhält man, indem man alle Ziffern invertiert.

Das ***Zweierkomplement*** einer Dualzahl wird gebildet, indem man zuerst das Einerkomplement der Zahl bildet und dann dazu eine 1 addiert.

Bei Verwendung des Zweierkomplements zur Darstellung negativer Zahlen reicht der darstellbare Zahlenbereich bei n Stellen von -2^{n-1} bis $+2^{n-1}-1$.

Bei der üblichen Darstellung ganzer Zahlen mit 16 Stellen (2 Bytes) ergibt sich somit ein Zahlenbereich von -32768 bis +32767.

Wenn Sie die Zahlenkreiseigenschaft der Zahlendarstellung mit beschränkter Stellenzahl beachten, sollte es Sie nicht mehr wundern, wenn Ihr Computer folgende Rechnung aufmacht

$$25 \cdot 1000 + 5 \cdot 3000 = -25536$$

Das Beispiel macht deutlich, dass ohne Beachtung von Bereichsüberschreitungen bei der Addition zweier positiver Zahlen ein unkorrektes Ergebnis entstehen kann. Um zu korrekten Ergebnisse zu kommen, müssen Zahlbereichsüberschreitungen verhindert werden.

Bevor Sie weiterlesen, sollten Sie die folgende Aufgabe bearbeiten:

Aufgabe 2.5: Entwickeln Sie eine Schaltung, die eine 4-stellige Dualzahl in ihr Zweierkomplement umwandelt.

Der erste Schritt zur Bildung des Zweierkomplements kann durch NICHT-Glieder realisiert werden, der zweite durch ein einfaches Addierwerk aus Halbaddierern.

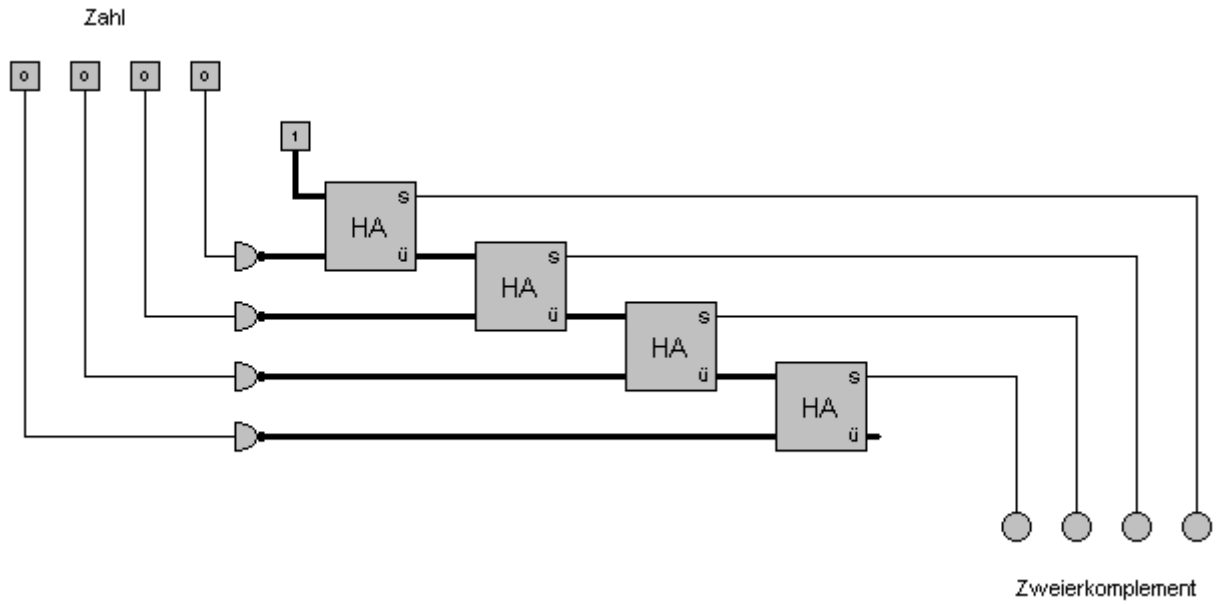


Abb. 2.1.5.2-2 Schaltnetz zur Zweierkomplement-Bildung

Damit sind wir nun in der Lage, eine Subtraktion über die Addition des Zweierkomplements auszuführen. Sollte die Summe eine negative Zahl sein (1 an der ersten Stelle), so muss zu dieser Zahl wieder das Zweierkomplement gebildet werden, um das Ergebnis mit Vorzeichen und Betrag notieren zu können.

Beispiel:

4	4	0100
- 7	+ (-7)	+ 1001
-----	--------	----------
- 3	- 3	1101

↑ negative Zahl ⇒
Zweierkomplement liefert den Betrag
1101 → 0010 (Einerkomplement)
\+ 1

0011 ⇒ Ergebnis: - 3

Die nachfolgende Schaltung zeigt ein 4-Bit[3]-Parallelsubtrahierwerk.
Die Addition von 1 zum Einerkomplement wird hier durch Anlegen eines 1-Signals an den ersten Eingang des Volladdierers für die Einerstelle realisiert. Deshalb musste der Halbaddierer für die Einerstelle des Addierwerks aus Abb. 2.1.4-1 durch einen Volladdierer ersetzt werden.

[3] Bit ist eine Abkürzung für **bi**nary digi**t** (Binärzeichen). Ein Bit stellt damit die kleinste Informationseinheit dar. Eine Gruppe von 8 Bits wird als Byte bezeichnet.

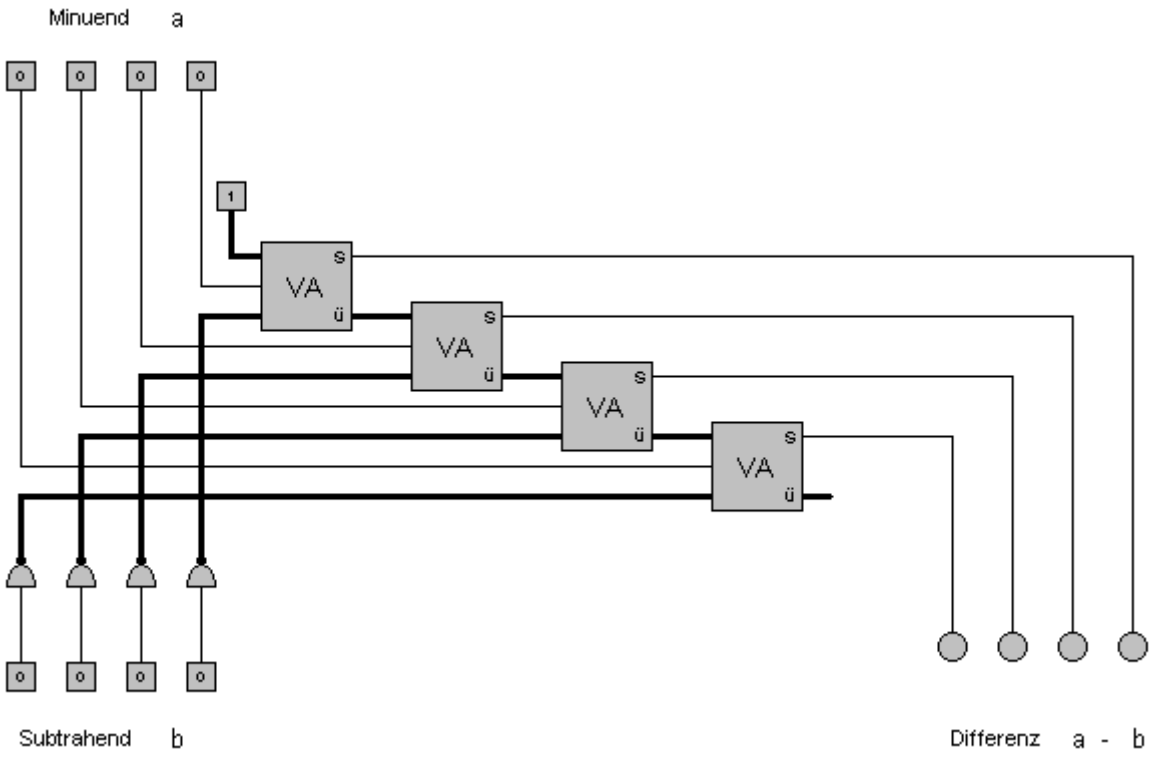


Abb. 2.1.5.2-3 4-Bit-Parallesubtrahierer ohne Überlaufsanzeige

Diese Schaltung hat noch den Nachteil, dass keine Bereichsüberschreitung signalisiert wird. Wir können hier nicht einfach den Übertrag des letzten Volladdierers benutzen, um eine Bereichsüberschreitung zu erkennen.
Durch die Zweierkomplementdarstellung der Dualzahlen reicht der Zahlenbereich von - 8 bis + 7. Das macht die Erkennung der Bereichsüberschreitung etwas komplizierter.

Die folgenden Beispiele sollen das zeigen:

-1	1111	
+ (-1)	+ 1111	
-------	---------	
	1 111	
	=====	
-2	(1)1110	der letzte Übertrag ist 1, obwohl *kein* Überlauf vorliegt, da -2 im vorgegebenen Bereich liegt

1	0001	
+ (-8)	+ 1000	
-------	---------	
-7	(0)1001	der letzte Übertrag ist 0 und es liegt *kein* Überlauf vor, da -7 im vorgegebenen Bereich liegt

Hieraus ist zu ersehen, dass der letzte Übertrag nicht ohne weiteres als Überlaufanzeige benutzt werden kann.

Aufgabe 2.6: Untersuchen Sie weitere Beispiele und versuchen Sie festzustellen, unter welchen Bedingungen ein Überlauf eintritt.

Die Bearbeitung der Aufgabe liefert folgendes Ergebnis:

Ein Überlauf liegt vor, wenn

1. die höchstwertigen Summandenbits gleich sind
   und
2. der letzte Übertrag ungleich dem höchstwertigen Ergebnisbit ist.

Bezeichnen wir allgemein die höchstwertigen Bits der Summanden mit x_n und y_n, das höchstwertige Ergebnisbit mit e_n und den letzten Übertrag mit $ü_n$, so lässt sich das Auftreten eines Überlaufs $ü_L$ folgendermaßen formal beschreiben:

$$ü_L = (x_n \equiv y_n) \wedge (e_n \not\equiv ü_n)$$

Hierbei steht das Zeichen $\equiv$ für die ÄQUIVALENZ-Funktion und das Zeichen $\not\equiv$ für die EXKLUSIV-ODER-Funktion.

Die Funktionstabellen dieser Funktionen sehen folgendermaßen aus:

a	b	A ≡ b
0	0	1
0	1	0
1	0	0
1	1	1

a	b	a ≢ b
0	0	0
0	1	1
1	0	1
1	1	0

Die ÄQUIVALENZ-Funktion liefert nur dann den Wert 1, wenn beide Eingänge gleiches Signal haben. Die EXKLUSIV-ODER-Funktion liefert den Wert 1, wenn die beiden Eingänge verschieden sind. Es wird deutlich, dass ÄQUIVALENZ-Funktion und EXKLUSIV-ODER-Funktion komplementär zueinander sind.

Eine Realisierung der EXKLUSIV-ODER-Funktion haben wir bereits bei der Entwicklung eines Halbaddierers kennengelernt, und zwar die Schaltung für die Summenbildung (siehe Abb. 2.1.2-1 b)).

Da eine EXKLUSIV-ODER-Funktion relativ häufig eingesetzt werden kann, wurde diese Schaltung in einen Baustein integriert, der im Menü Bauteile zur Verfügung steht. Er sieht wie ein UND-Glied aus und trägt zur Unterscheidung das EXKLUSIV-ODER-Zeichen $\not\equiv$

Ein ÄQUIVALENZ-Glied lässt sich durch Negation des Ausganges eines EXKLUSIV-ODER-Gliedes realisieren.

Abb. 2.1.5.2-4 a) EXKLUSIV-ODER-Glied b) ÄQUIVALENZ-Glied

Damit sind wir nun in der Lage, den Parallelsubtrahierer so zu erweitern, dass eine Bereichsüberschreitung signalisiert wird.

Hierzu müssen die beiden höchstwertigen Summandenbits mit einem ÄQUIVALENZ-Glied und der letzte Übertrag und das höchstwertige Bit des Ergebnisses mit einem EXKLUSIV-ODER-Glied verknüpft werden. Die beiden Verknüpfungsergebnisse werden über ein UND-Glied zum Bereichsüberschreitungssignal zusammengefasst.
Damit ergibt sich folgende Schaltung.

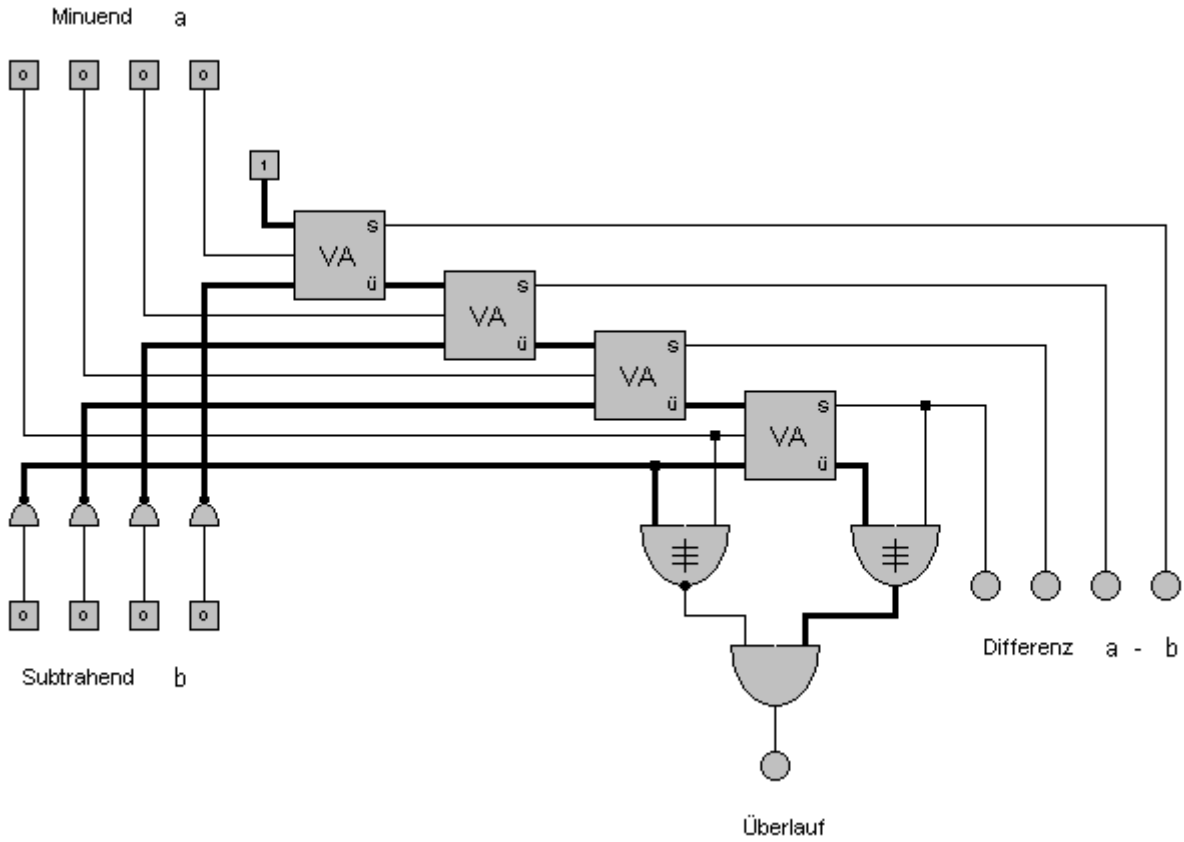


Abb. 2.1.5.2-5 4-Bit-Parallelsubtrahierer mit Überlaufanzeige für den Wertebereich von -8 bis +7

Aufgabe 2.7: Bauen oder laden Sie diese Schaltung und überprüfen Sie die Funktionsweise.

2.1.6 Umschaltbare Rechenschaltung

Die weitgehende schaltungstechnische Übereinstimmung von Paralleladdierer und Parallelsubtrahierer führt zu der Überlegung, ob man nicht auf einfache Weise eine zwischen Addition und Subtraktion umschaltbare Rechenschaltung konstruieren könnte.

Wenn wir beim Paralleladdierer aus Abb. 2.1.4-1 den Halbaddierer für die Einerstelle durch einen Volladdierer ersetzen und dessen ersten Eingang mit 0 belegen, verhält sich diese Schaltung genauso wie der bisherige Paralleladdierer, da ein Volladdierer mit einem festen 0-Signal an einem Eingang wie ein Halbaddierer wirkt. Die Belegung dieses Einganges mit 1-Signal hat die Addition einer 1 zur Folge und stellt somit den zweiten Schritt zur Zweierkomplementbildung dar.
Wir müssen jetzt nur noch dafür sorgen, dass in diesem Fall auch noch die notwendige Invertierung der Subtrahendenbits stattfindet. Hierbei hilft uns wiederum das EXKLUSIV-ODER-Glied mit einer interessanten Eigenschaft weiter.

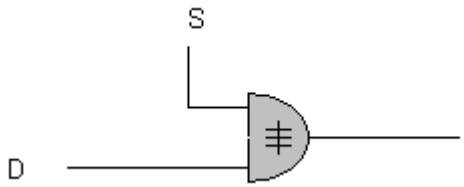


Abb. 2.1.5.2-6 EXKLUSIV-ODER-Glied mit Steuer- und Dateneingang

Liegt am Steuereingang 0-Signal an, so verhält sich das EXKLUSIV-ODER-Glied für die Datenleitung wie ein ununterbrochener Draht, d.h. die an der Datenleitung anliegenden Signale gehen unverändert hindurch.
Liegt hingegen 1-Signal an der Steuerleitung an, so wird das Datensignal invertiert.

Aufgabe 2.8: Überprüfen Sie diese Eigenschaft mit *LOCAD*.

Ersetzen wir die NICHT-Glieder in den Datenleitungen der unteren Dualzahl durch EXKLUSIV-ODER-Glieder, deren zweiter Eingang mit der Steuerleitung zur Rechenartenumschaltung verbunden ist, so wird beim Anlegen eines 1-Signals an diese Steuerleitung erreicht, dass die Datenbits der Zahl invertiert werden. Da auch der obere Eingang des ersten Volladdierers an diese Steuerleitung angeschlossen ist, wird insgesamt eine Zweierkomplementbildung durchgeführt. Liegt 0-Signal an der Rechenartensteuerleitung an, so verhält sich die Schaltung wie ein normaler Paralleladdierer.

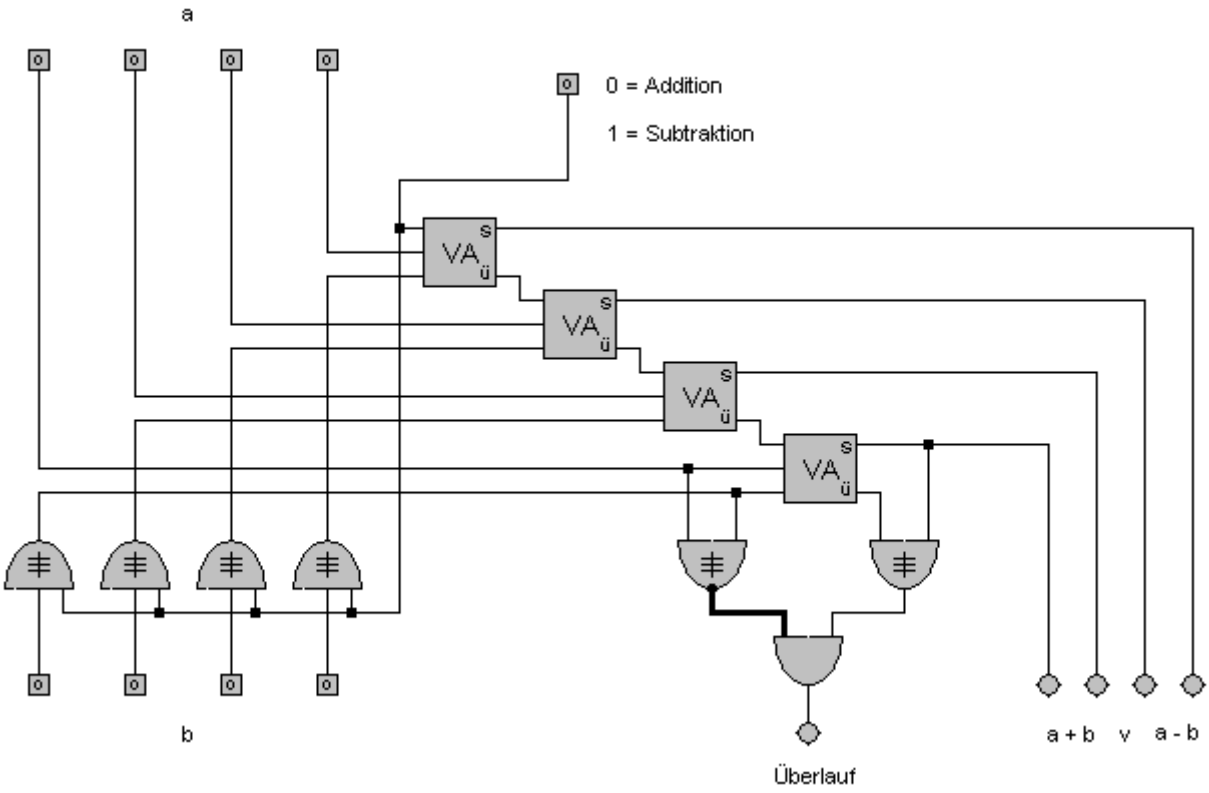


Abb. 2.1.5.2-7 Umschaltbare Rechenschaltung

Aufgabe 2.9: Bauen oder laden Sie die obige Schaltung und überprüfen Sie die Funktionsweise.

Die Schaltung hat noch den Nachteil, dass ein negatives Ergebnis komplementiert werden muss, um den Betrag zu ermitteln.

Aufgabe 2.10: Entwickeln Sie eine Schaltung, die jede vierstellige Dualzahl in Zweierkomplementdarstellung in Betrag und Vorzeichen zerlegt.
(Lösung im Anhang C)

2.2 Code-Umsetzer

Wir haben in Abschnitt 3.1 gesehen, wie mit Zahlen in Dualdarstellung auf elektronischem Wege gerechnet werden kann. Um Dezimalzahlen mit der entwickelten Rechenschaltung zu verarbeiten, mussten wir diese in Dualzahlen umsetzen (**codieren**), ebenso musste das Ergebnis in eine Dezimalzahl rückübersetzt (**decodiert**) werden. Diese Umwandlungen können durch Schaltnetze automatisch vorgenommen werden.

2.2.1 Dezimal-Dual-Umsetzer

Wir wollen uns hier nur die Umwandlung einer einstelligen Dezimalzahl in eine Dualzahl ansehen, also ein Schaltnetz entwickeln, das den zehn Dezimalziffern 0 bis 9 die entsprechende Dualzahl zuordnet. Die Schaltung muss somit zehn Eingänge und vier Ausgänge besitzen.

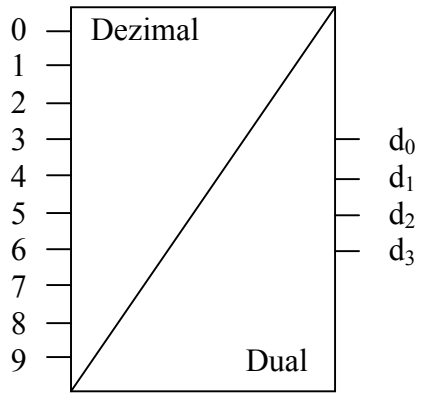


Abb. 2.2.1-1 Blockschaltsymbol für einen Dezimal-Dual-Umsetzer (Codierer)

Wenn wir die zehn Eingänge des Codierers mit den zehn Zifferntasten einer Tastatur verbinden, erhalten wir beim Druck auf eine dieser Tasten die entsprechende Dualdarstellung und haben damit eine Verbindung zwischen der Außenwelt und dem Computerinneren geschaffen. Den Druck auf eine Taste stellen wir in der zu entwickelnden Schaltung als 1-Signal auf der entsprechenden Eingangsleitung dar.
Die folgende Tabelle zeigt, welche Dualzahlen bei welchen Tasten erzeugt werden müssen:

Taste	Dualzahl			
	d_3	d_2	d_1	d_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Für jeden der vier Ausgänge d_0 bis d_3 muss eine eigene Logikschaltung aufgebaut werden. Die Zusammenfassung dieser Schaltungen liefert dann das Schaltnetz zur Dezimal-Dual-Umwandlung, einen sogenannten Codierer.

Aus der obigen Tabelle kann man leicht die formalen Zusammenhänge erkennen:

$d_3 = T9 \vee T8$
$d_2 = T4 \vee T5 \vee T6 \vee T7$
$d_1 = T2 \vee T3 \vee T6 \vee T7$
$d_0 = T1 \vee T3 \vee T5 \vee T7 \vee T9$

Damit erhält man die folgende Codier-Schaltung:

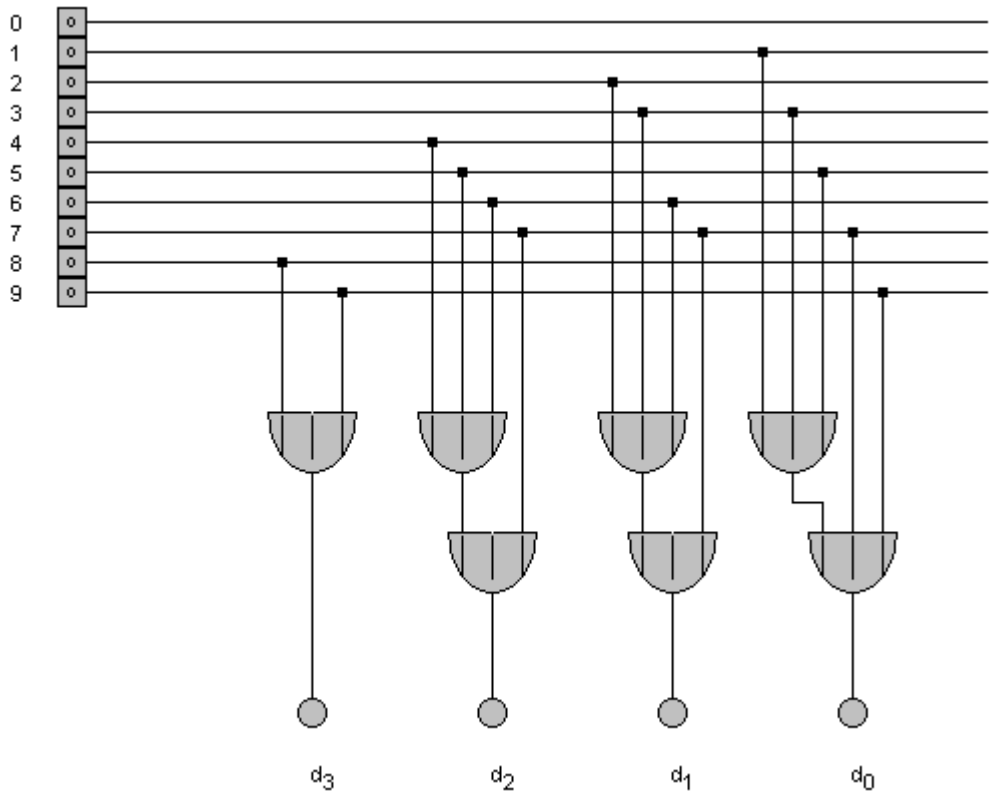


Abb. 2.2.1-2 Codier-Schaltnetz für die Umwandlung der Dezimalziffern 0 bis 9 in Dualzahlen

2.2.2 Dual-Dezimal-Umsetzer

Die Dual-Dezimal-Umsetzung ist die Umkehrung zur Dezimal-Dual-Umsetzung und wird als Dekodierung bezeichnet.
Wir beschränken uns wieder auf die Dezimalziffern von 0 bis 9, d.h. auf die Dualzahlen von 0000 bis 1001. Die Schaltung muss also vier Eingänge und zehn Ausgänge haben.

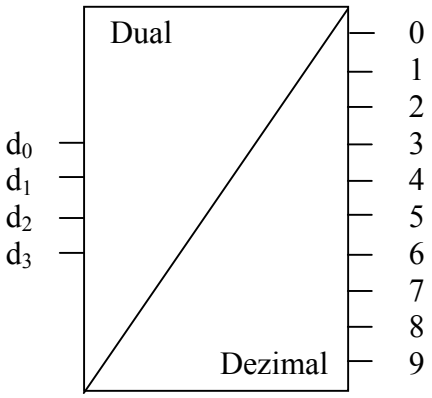


Abb. 2.2.2-1 Blockschaltsymbol für einen Dual-Dezimal-Umsetzer (Decodierer)

Die nebenstehende Tabelle zeigt die Zusammenhänge zwischen den Ein- und Ausgängen:

Dualzahl				Ausgang
d_3	d_2	d_1	d_0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9

Damit ergeben sich folgende formale Zusammenhänge:

$$0 = \overline{d_3} \wedge \overline{d_2} \wedge \overline{d_1} \wedge \overline{d_0}$$

$$1 = \overline{d_3} \wedge \overline{d_2} \wedge \overline{d_1} \wedge d_0$$

$$2 = \overline{d_3} \wedge \overline{d_2} \wedge d_1 \wedge \overline{d_0}$$

$$3 = \overline{d_3} \wedge \overline{d_2} \wedge d_1 \wedge d_0$$

$$4 = \overline{d_3} \wedge d_2 \wedge \overline{d_1} \wedge \overline{d_0}$$

$$5 = \overline{d_3} \wedge d_2 \wedge \overline{d_1} \wedge d_0$$

$$6 = \overline{d_3} \wedge d_2 \wedge d_1 \wedge \overline{d_0}$$

$$7 = \overline{d_3} \wedge d_2 \wedge d_1 \wedge d_0$$

$$8 = d_3 \wedge \overline{d_2} \wedge \overline{d_1} \wedge \overline{d_0}$$

$$9 = d_3 \wedge \overline{d_2} \wedge \overline{d_1} \wedge d_0$$

Hiermit lässt sich das Schaltnetz ohne weiteres aufbauen. Es ist allerdings mit *LOCAD* etwas umständlich zu realisieren, da keine UND-Glieder mit 4 Eingängen existieren und deshalb jeweils zwei UND-Glieder kombiniert werden müssten.

Eine genauere Betrachtung der Gleichungen zeigt, dass zur Decodierung der Zahlen 8 und 9 lediglich UND-Glieder mit zwei Eingängen benötigt werden, da sich diese beiden Zahlen durch d_3 von allen anderen unterscheiden und untereinander durch d_0 unterschieden werden können. Damit lassen sich die Gleichungen für 8 und 9 folgendermaßen vereinfachen:

$$8 = d_3 \wedge \overline{d_0}$$

$$9 = d_3 \wedge d_0$$

Ebenso lassen sich folgende weitere Vereinfachungen finden:

$$0 = \overline{d_3} \wedge \overline{d_2} \wedge \overline{d_1} \wedge \overline{d_0}$$

$$1 = \overline{d_3} \wedge \overline{d_2} \wedge \overline{d_1} \wedge d_0$$

$$2 = \overline{d_2} \wedge d_1 \wedge \overline{d_0}$$

$$3 = \overline{d_2} \wedge d_1 \wedge d_0$$

$$4 = d_2 \wedge \overline{d_1} \wedge \overline{d_0}$$

$$5 = d_2 \wedge \overline{d_1} \wedge d_0$$

$$6 = d_2 \wedge d_1 \wedge \overline{d_0}$$

$$7 = d_2 \wedge d_1 \wedge d_0$$

Machen Sie sich das klar, indem Sie Folgen von weniger als 4 Dualziffern suchen, die nur einmal vorkommen.

Insgesamt erhält man die folgende Decodier-Schaltung:

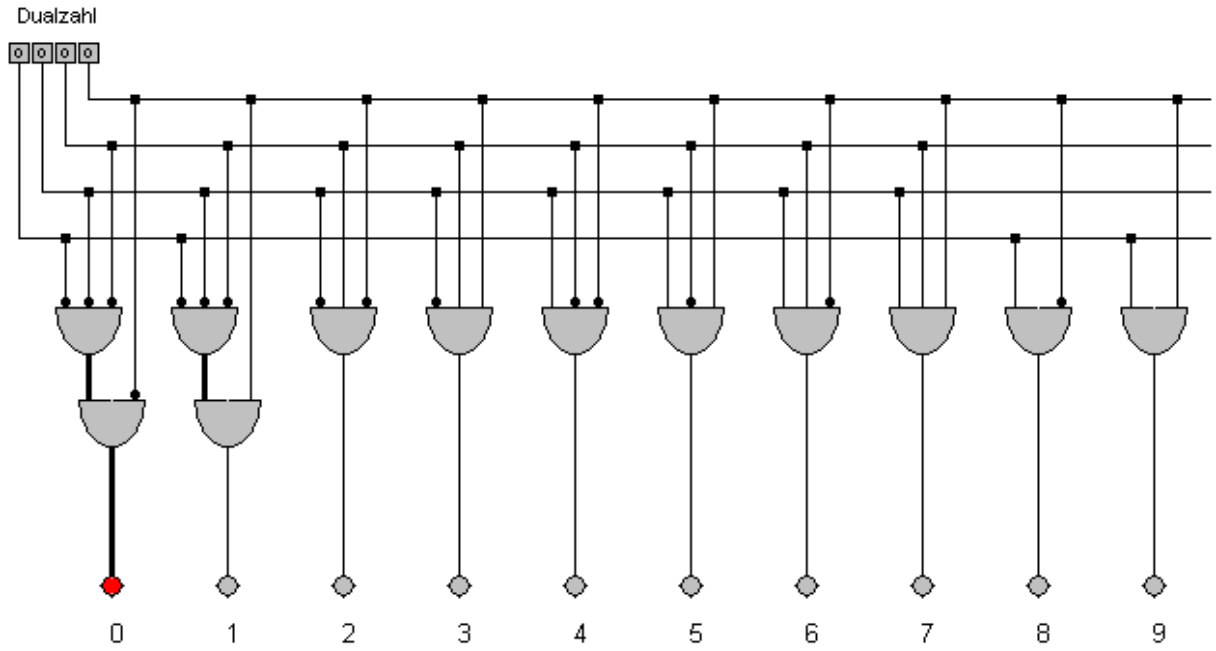


Abb. 2.2.2-2 Decodier-Schaltnetz für die Umwandlung der Dualzahlen 0000 bis 1001 in Dezimalzahlen

Aufgabe 2.11: Bauen oder laden Sie diesen Decoder und untersuchen Sie ihn. Was passiert bei Eingangskombinationen, die nicht in der obigen Tabelle aufgeführt sind ? Wie kommt es zu diesem Effekt ?

Aufgabe 2.12: Entwickeln Sie einen Decoder, der das gleiche Verhalten zeigt wie die im IC 74141 integrierte Schaltung, die bei einer Eingangskombination größer als 1001 **keinen** Ausgang auf 1-Signal legt (Pseudotetraden-Ausblendung). (Lösung im Anhang C)

Aufgabe 2.13: Entwickeln Sie einen 3-zu-8-Decoder, der 3 Eingänge besitzt und 8 Ausgänge hat, die von 0 bis 7 durchnumeriert sind. Bei einer Eingangkombination soll **der** Ausgang 1-Signal führen, dessen Nummer der Dezimalwert der als Dualzahl gelesenen Eingangskombination ist.
(Lösung im Anhang C)

Im folgenden brauchen wir einen 3-zu-8-Decoder nicht mehr so mühsam aufzubauen. Im Menü Bauteile können Sie diesen Decoder als Block-Baustein mit oder ohne Taktsteuerung auswählen. Bis geklärt ist, was es mit der Taktsteuerung auf sich hat, wählen Sie den Decoder ohne Taktsteuerung.

Sie erhalten dann folgendes Blockschaltbild:

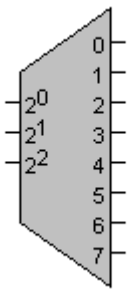


Abb. 2.2.2-3 3-zu-8-Decoder ohne Taktsteuerung

2.3 Sieben-Segment-Anzeige

Die bisherige Möglichkeit zur Ausgabe einer Dualzahl durch Decodierung ist zwar besser als die Kopfrechenarbeit, aber noch nicht besonders befriedigend. Es wäre optimal, wenn es gelingen würde, ähnlich wie in Taschenrechnern die Segmente einer Sieben-Segment-Anzeige so anzusprechen, dass die Zahl direkt dezimal dargestellt wird.

Bei einer Sieben-Segment-Anzeige sind sieben Leuchtdioden- bzw. Flüssigkristallstreifen in Form einer Acht angeordnet, wobei jedes Segment einzeln hervorgehoben werden kann. Durch die unterschiedlichen Kombinationsmöglichkeiten lassen sich alle Ziffern in der von Taschenrechnern, Digitaluhren, Zapfsäulen usw. gewohnten Form darstellen.

Mit *LOCAD* kann eine Sieben-Segment-Anzeige mit Hilfe der Leuchtdioden realisiert werden. Dazu müssen mehrere Leuchtdioden wie in der folgenden Abbildung direkt aneinandergesetzt werden.

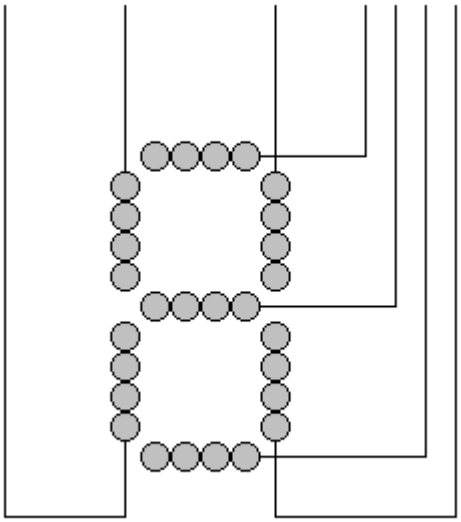


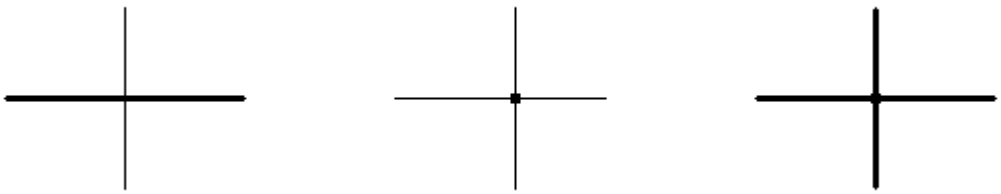
Abb. 2.3-1 Sieben-Segment-Anzeige mit LEDs realisiert

Mit den großen Dioden entsteht eine gut sichtbare Darstellung. Wenn Sie die kleineren Dioden verwenden, müssen Sie darauf achten, dass Sie zu Beginn nicht alle Dioden einsetzen, sondern diejenigen, die an die Leitung angeschlossen werden, erst nachträglich auf das Leitungsende setzen. Das ist deshalb notwendig, weil die Eingänge der kleinen Dioden nicht auf Rasterpunkten liegen und somit keine Leitung direkt bis an die Eingänge gezogen werden kann. Es ist aber immer möglich, eine Leuchtdiode auf eine bestehende Leitung aufzusetzen, also auch auf das Ende einer Leitung. Sollten Sie aus Versehen bereits zu viele Leuchtdioden gesetzt haben, brauchen Sie nicht die ganze Schaltung zu löschen und von neuem anzufangen. Löschen Sie die störende Leuchtdiode, ziehen Sie dann die Leitung bis zu dieser Stelle und setzen Sie die Leuchtdiode erneut an diese Stelle.

Unklar ist jetzt allerdings noch, wie die einzelnen Segmente an die Decoderausgänge angeschlossen werden sollen, damit die Ziffern dargestellt werden. Die Segmente müssen bei den verschiedenen Ziffern in den unterschiedlichsten Kombinationen aufleuchten. Es sieht so aus, als wäre hierzu ein weiterer großer Aufwand an Decodierlogik notwendig. Eine einfach Lösung ist jedoch mit Hilfe von Koppeldioden möglich.

2.4 Koppeldioden

Wie bereits in Kapitel 2 erwähnt, sind zwei gekreuzte Leitungen als isoliert gegeneinander zu betrachten, solange sie an der Kreuzungsstelle nicht mit einem Verbindungspunkt „verlötet“ wurden. Die folgende Abbildung soll das nochmals verdeutlichen:



a) b)

Abb. 2.4-1 gekreuzte Leitungen a) ohne Verbindung b) mit Verbindung

Es ist auch möglich, zwei Leitungen über eine Diode miteinander zu verbinden.

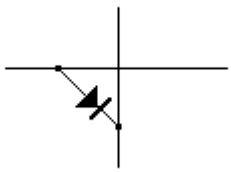


Abb. 2.4-2 Verbindung zweier Leitungen über eine Koppeldiode

Eine solche Verbindung zeigt ein besonderes Verhalten. Die Diode wirkt wie ein Ventil, sie lässt den Strom nur in der durch die Pfeilspitze angegebenen Richtung durch (technische Stromrichtung). Das hat folgende Konsequenzen:

- Ein Signal auf der waagerechten Leitung wirkt sich auf die senkrechte Leitung aus.
- Ein Signal auf der senkrechten Leitung kann sich nicht auf die waagerechte Leitung auswirken.

Mit *LOCAD* können Koppeldioden auf Leitungskreuzungen gesetzt werden, indem man im Menü *Leitungen* den Button „koppeln“ aktiviert. Man kann dann mit der Marke die Stelle auswählen und dort eine Koppeldiode setzen bzw. löschen.
Koppeldioden werden in *LOCAD* durch eine pfeilförmige Spitze nach unten dargestellt. Es ist nur die in Abbildung 2.4-2 dargestellte Richtung vorgesehen und damit haben die Koppeldioden auch nur die oben beschriebene Wirkung.

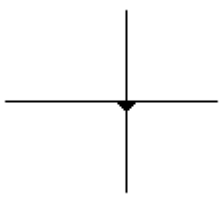


Abb. 2.4-3 Darstellung einer Koppeldiode in *LOCAD*

Wie können nun die Koppeldioden benutzt werden, um die Sieben-Segment-Ansteuerung zu vereinfachen ?

Die folgende Abbildung soll diese Möglichkeit verdeutlichen.

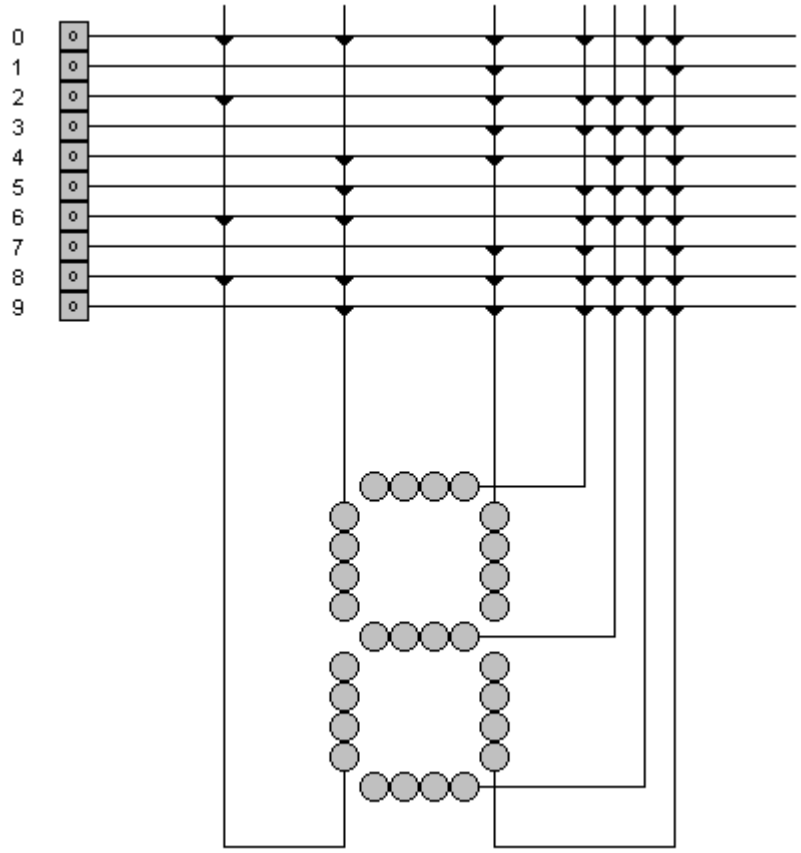


Abb. 2.4-4 Sieben-Segment-Ansteuerung mit einer Diodenmatrix

Wir betrachten jetzt nicht mehr den Decoder aus Abb. 2.2.2-2 selbst, sondern nur noch die 10 Ausgangsleitungen. Wegen der vorgegebenen Richtung der Koppeldioden müssen diese waagerecht angeordnet sein. Die Anschlussdrähte der Sieben-Segmentanzeige kreuzen dieses Leitungsbündel. Wir beginnen mit Leitung 0.

Liegt hier ein 1-Signal vor, so müssen alle Segmente der Anzeige mit Ausnahme des mittleren leuchten, um die 0 darzustellen. Wir koppeln also bis auf das mittlere Segment alle anderen mit Koppeldioden an die Leitung 0 an.

Entsprechend verfahren wir mit der Leitung 1. Hier sollen nur die beiden rechten Segmente leuchten, also werden auch nur diese an die Leitung 1 angekoppelt. So werden nach und nach an alle Leitungen die Ansteuerleitungen der Sieben-Segment-Anzeige angekoppelt.

Wenn wir jetzt eine der 10 Leitungen auswählen und ein 1-Signal anlegen, zeigt die Anzeige die entsprechende Zahl in Sieben-Segment-Form.

Machen Sie sich bitte klar, dass diese Ansteuerung nur wegen der Koppeldioden funktioniert. Die Koppeldioden können nicht durch Verbindungspunkte ersetzt werden. Durch die Koppeldioden wird verhindert, dass ein Signal auf einer Zuführungsleitung zu den Segmenten über eine der waagerechten Leitungen ein anderes Segment beeinflusst.

Mit dem in *LOCAD* vorgesehenen 3-zu-8-Decoder können wir nun eine Schaltung aufbauen, die den Dezimalwert einer dreistelligen Dualzahl direkt über eine Sieben-Segment-Anzeige anzeigt.

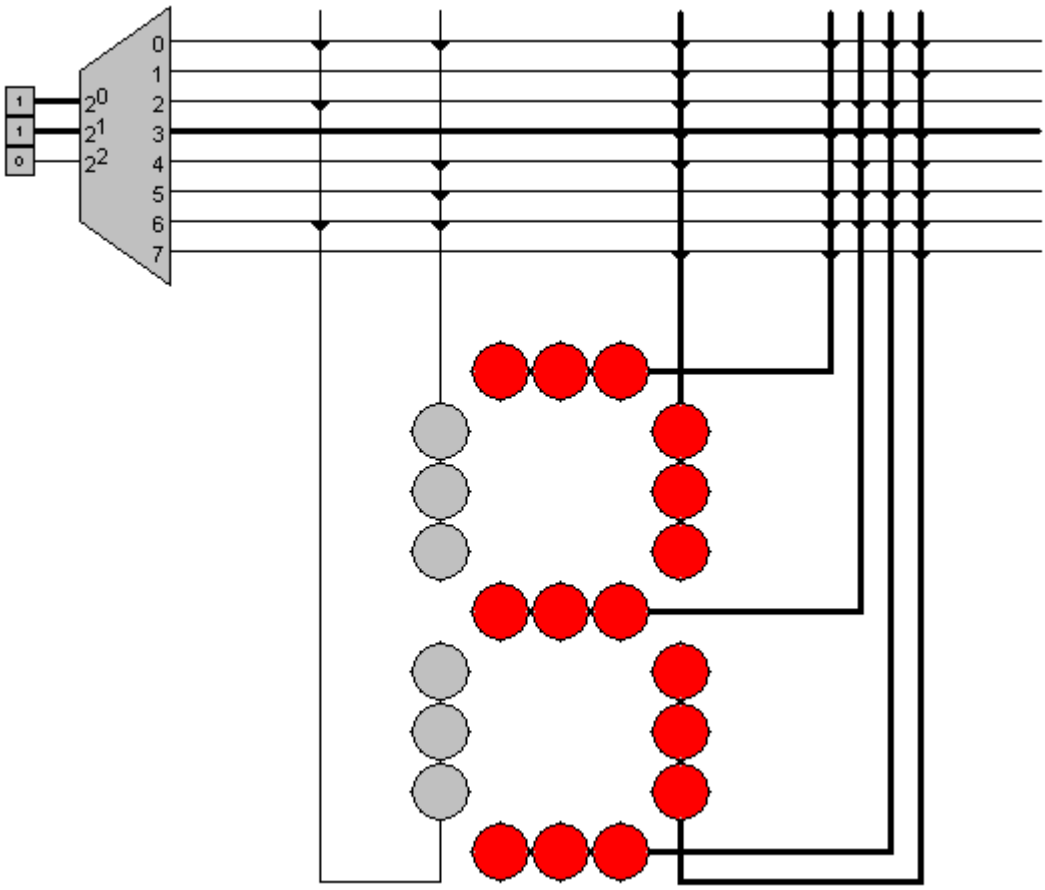


Abb. 2.4-5 Dezimale Anzeige einer dreistelligen Dualzahl

Koppeldioden spielen auch in anderen wichtigen Bauteilen eines Computers eine entscheidende Rolle. Hier sind die ROMs und PROMs zu erwähnen.

2.5 ROM und PROM

ROM ist die Abkürzung für **r**ead-**o**nly-**m**emory (Nur-Lese-Speicher).
Es handelt sich hierbei um einen Speicher, dessen Inhalt bei der Herstellung festgelegt wird und danach nicht mehr geändert werden kann. Der Inhalt eines ROMs kann nur gelesen werden und bleibt auch bei Stromausfall erhalten. ROMs werden als Speicher für feste Programme und Daten benutzt.

PROM ist die Abkürzung für **p**rogrammable **r**ead-**o**nly-**m**emory (programmierbarer Nur-Lese-Speicher). Es handelt sich hierbei um einen Speicher, dessen Inhalt vom Anwender mit einem speziellen Gerät einmal festgelegt werden kann und sich danach nicht mehr ändern lässt. Nach der Programmierung verhält sich ein PROM wie ein ROM.

Die folgende Abbildung zeigt eine Möglichkeit, mit Hilfe von Koppeldioden ein PROM als Diodenmatrix zu realisieren:

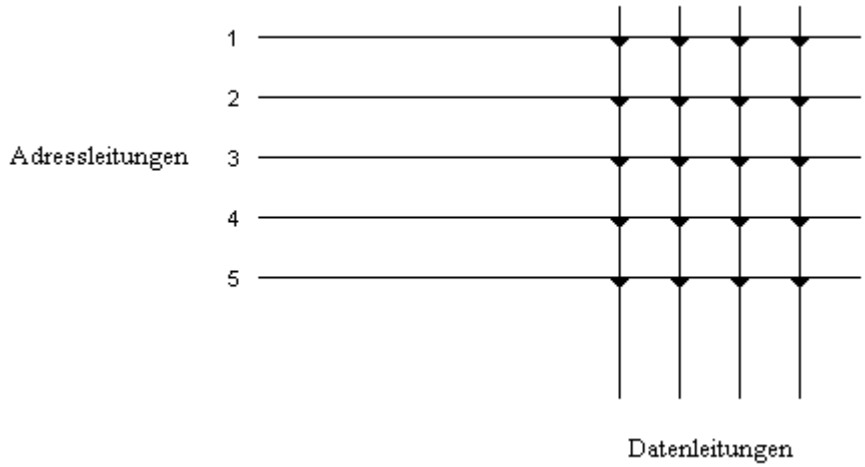


Abb. 2.5-1 PROM-Realisierung mit Diodenmatrix

Die waagerechten Leitungen sind die Adressleitungen, die senkrechten die Datenleitungen. Allen Leitungen sind an den Kreuzungspunkten über Koppeldioden verbunden. Mit Hilfe des Programmiergerätes können nun ganz bestimmte Koppeldioden durch wohldefinierte Stromstöße „durchgebrannt“ werden. Damit ist die Verbindung der Adressleitung zu der entsprechenden Datenleitung zerstört und ein möglicher 1-Zustand auf der Adressleitung kann sich nicht auf die Datenleitung fortsetzen. Eine zerstörte Koppeldiode repräsentiert somit eine 0, eine unbeschädigte eine 1.

Wir wollen jetzt ein PROM so programmieren, dass es die ersten 4 Primzahlen als Festwerte enthält, die dann durch Anlegen eines 1-Signals an die entsprechende Adressleitung in Dualdarstellung auf den Datenleitungen erscheinen sollen.
Dazu gehen wir von einer 4 x 3-Diodenmatrix aus, in der auf allen Kreuzungspunkten Koppeldioden angebracht sind. Auf Adressleitung 1 soll die erste Primzahl (2) in Dualdarstellung programmiert werden. Dazu müssen die Koppeldioden zerstört werden, die zu den Datenleitungen d_0 und d_2 gehören. Mit *LOCAD* wird die Zerstörung durch Löschen realisiert.

Eine entsprechende Vorgehensweise für die anderen Adressleitungen liefert schließlich folgende Matrix:

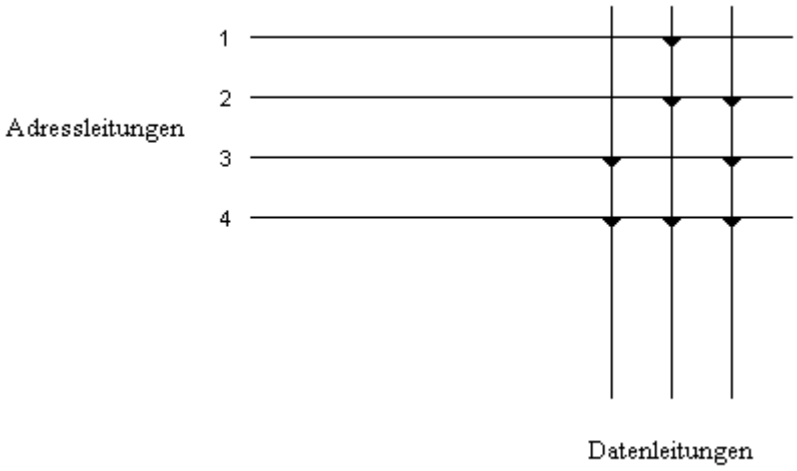


Abb. 2.5-2 4 x 3-Diodenmatrix mit den ersten 4 Primzahlen als Festwerte

Die obige Matrix repräsentiert die folgenden Speicherinhalte:

Adresse	Inhalt
1	010
2	011
3	101
4	111

Beim Abruf eines gespeicherten Inhalts durch Anlegen eines 1-Siganls an eine bestimmte Adressleitung ist darauf zu achten, dass sich alle anderen Adressleitungen im 0-Zustand befinden, weil es andernfalls zu einer Datenüberlagerung kommt.

Aufgabe 2.14: Bauen Sie den Festwertspeicher mit *LOCAD* auf und überprüfen Sie die Funktionsweise.

Durch Kombination mit einem 3-zu-8-Decoder und der Sieben-Segment-Anzeige können wir auf einfache Weise dafür sorgen, dass die aus dem Festwertspeicher abgerufene Primzahl direkt dezimal angezeigt wird.

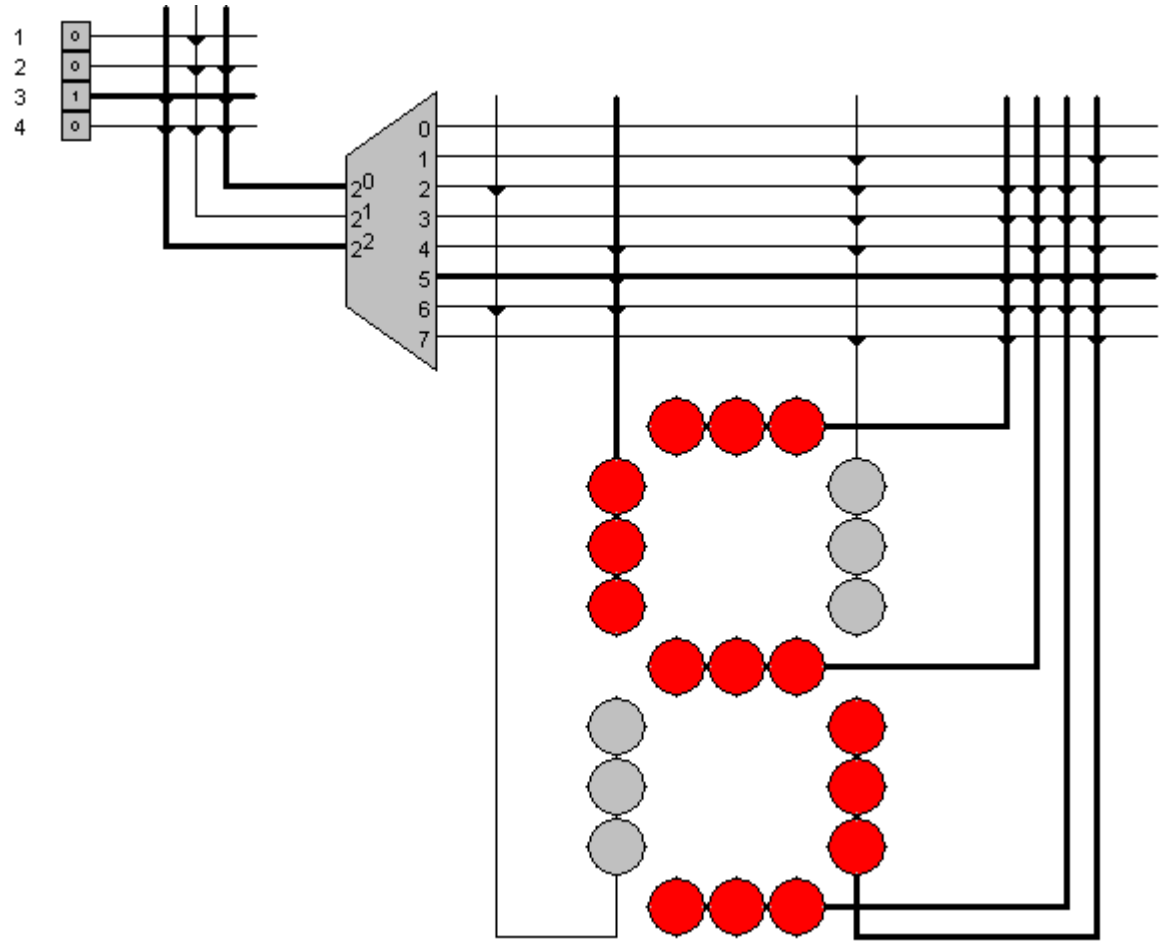


Abb. 2.5-3 Dezimale Anzeige eines Festwertspeicherinhalts

2.6 Multiplexer

Der Datenaustausch zwischen den verschiedenen Bereichen eines Computers erfordert Schaltungen, die eine von mehreren Eingangsleitungen auf eine Ausgangsleitung durchschalten können. Da hier eine Auswahl unter mehreren Leitungen getroffen wird, bezeichnet man eine solche Schaltung auch als Quellenauswahlschaltung oder Datenselektor. Gebräuchlicher ist allerdings die Bezeichnung Multiplexer (Abkürzung: MUX). Wir wollen hier einen Multiplexer entwerfen, der einen von vier Eingängen auf den Ausgang durchschaltet (1-aus-4-Multiplexer).
Um einen der vier Eingänge auszuwählen, werden zwei Adressleitungen benötigt, so dass die Schaltung insgesamt 6 Eingänge besitzt. Statt die Schaltung über die sehr umfangreiche Funktionstabelle herzuleiten, gehen wir intuitiv vor und benutzen eine besonders nützliche Eigenschaft der UND-Schaltung, die sogenannte Torwirkung.

2.6.1 Torwirkung der UND-Schaltung

Um diese Wirkung einer UND-Schaltung zu verstehen, betrachten wir folgende Abbildung:

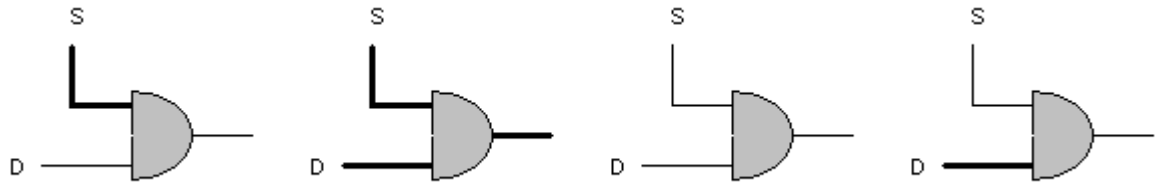


Abb. 2.6.1-1 UND-Glieder mit Steuer- und Dateneingängen

Hat der Steuereingang S 1-Signal, so nimmt der Ausgang immer den Zustand der Datenleitung D an. Man sagt, das Tor ist geöffnet, die Datenleitungszustände können ungehindert durch das UND-Glied hindurch gehen.
Befindet sich der Steuereingang S im 0-Zustand, so ist das Tor geschlossen, der Ausgang hat unabhängig vom Datenleitungszustand stets den Zustand 0. Es werden also abhängig vom Steuerleitungszustand Daten hindurchgelassen oder zurückgehalten. Wegen dieser Wirkung wird die UND-Schaltung häufig auch als *Torschaltung* oder *elektronisches Tor* bezeichnet.

Hiermit ist es auf einfache Weise möglich, mit einer Steuerleitung eine von zwei Leitungen auszuwählen, die auf eine andere Leitung durchgeschaltet werden soll.

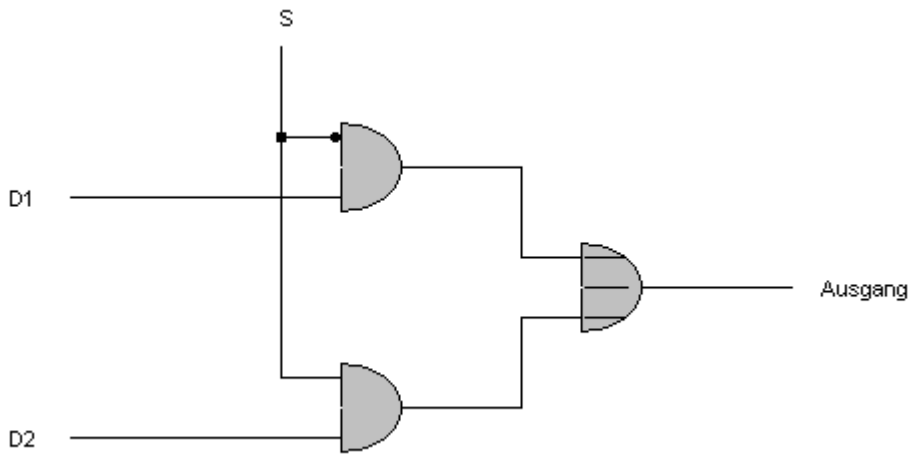


Abb. 2.6.1-2 1-aus-2-Multiplexer

Bei jedem Zustand der Steuerleitung S ist genau ein Tor geöffnet, und zwar beim Steuerleitungszustand 0 das obere, beim Zustand 1 das untere. Die beiden Ausgänge der UND-Glieder sind über ein ODER-Glied mit dem eigentlichen Ausgang verbunden.

Die Erweiterung auf vier Leitungen sollte jetzt nicht mehr schwer fallen. Es ist klar, dass zur Auswahl einer der vier Leitungen zwei Steuersignale benötigt werden und dass in Abhängig-

keit von diesen Steuersignalen ein ganz bestimmtes Tor geöffnet werden muss. Wenn wir zwei der drei möglichen Eingänge eines UND-Gliedes zur Torsteuerung verwenden, muss beachtet werden, dass das Tor nur geöffnet ist, wenn sich beide Steuereingänge im 1-Zustand befinden. Damit kommen wir zu folgender Lösung.

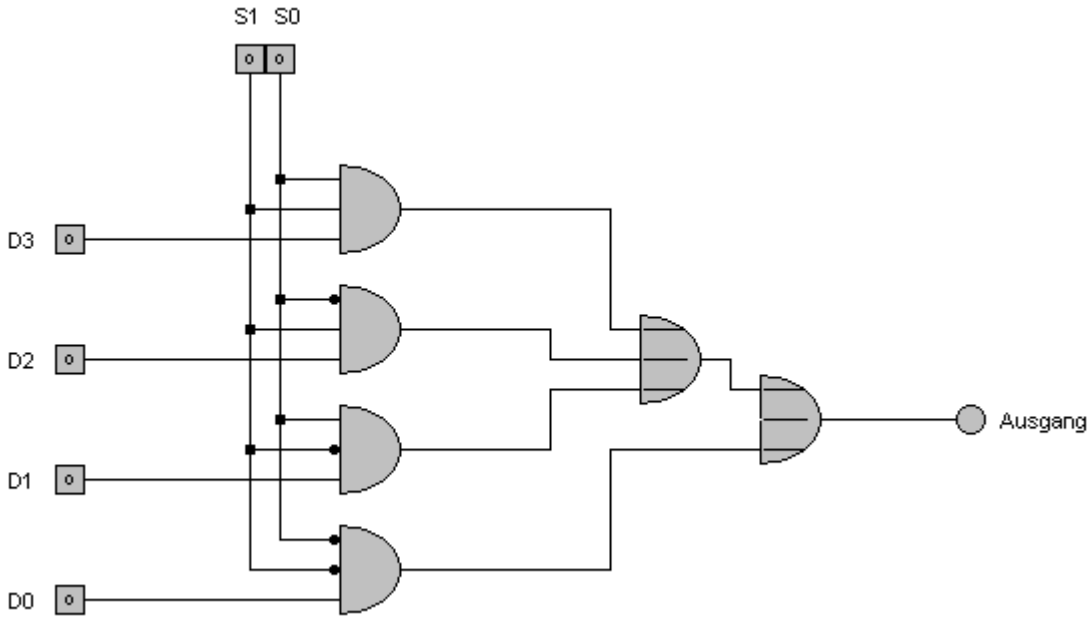


Abb. 2.6.1-3 1-aus-4-Multiplexer

In *LOCAD* steht ein fertiger 1-aus-4-Multiplexer mit dem folgenden Blockschaltsymbol zur Verfügung.

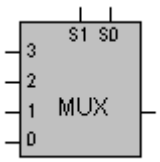


Abb. 2.6.1-4 Blockschaltsymbol des 1-aus-4-Multiplexers

Es wird jeweils der Eingang auf den Ausgang durchgeschaltet, dessen Nummer der Dezimalwert der als Dualzahl gelesenen Steuereingangskombination ist.

Aufgabe 2.15: Entwerfen Sie einen Demultiplexer, der eine Eingangsleitung auf 4 Ausgangsleitungen durchschalten kann.
(Lösung im Anhang C)

2.7 Impulserzeugung

Zum Abschluss dieses Kapitels wollen wir uns noch zwei Schaltungen ansehen, die im nachfolgenden Kapitel über Schaltwerke benötigt werden.

Aufgabe: Bauen Sie die folgenden Schaltungen auf und untersuchen Sie diese.
Erklären Sie das beobachtete Verhalten.
Worin unterscheiden sich die beiden Schaltungen ?

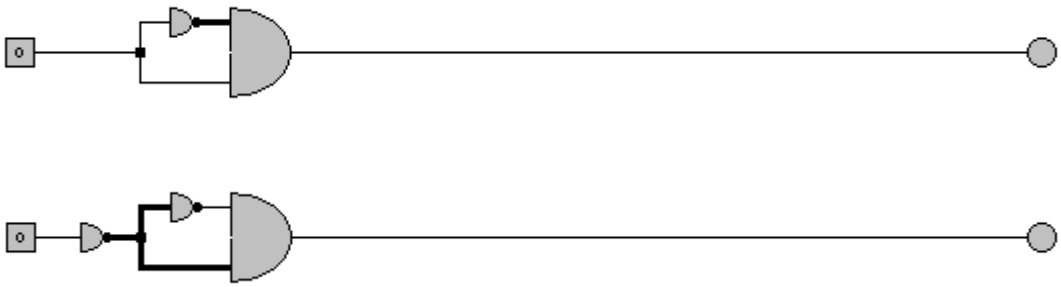
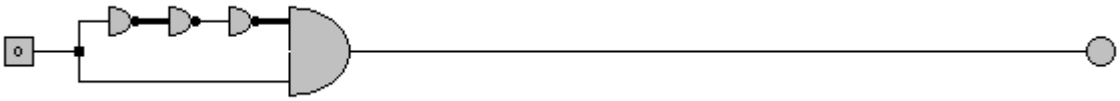


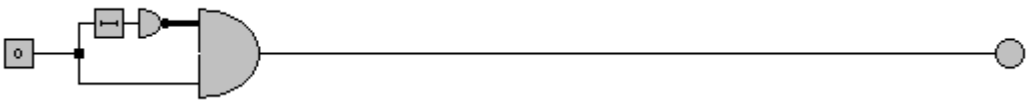
Abb. 2.7-1 Schaltungen zur Impulserzeugung

Falls Sie einen sehr schnellen Rechner haben, müssen Sie eventuell eine Signalverzögerung einstellen, um den Effekt beobachten zu können. (Button „Verzögerung“ im Menü *Betrieb*)

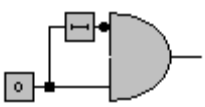
Durch Einsetzen weiterer Inverter kann der Impuls verlängert werden.



Wenn eine besonders große Impulsdauer benötigt wird, bietet sich statt einer größeren Anzahl von Invertern der Einsatz eines oder mehrerer Verzögerungsglieder an.



Eine platzsparendere Version der oberen Schaltung sieht folgendermaßen aus:



3 Schaltwerke

Alle in Kapitel 2 beschriebenen Schaltungen hatten die Eigenschaft, dass sie zu jeder Eingangskombination unabhängig von den bisherigen Ein- und Ausgangszuständen neue Ausgangssignale lieferten. Mit solchen Schaltungen ist es aber nicht möglich, Schaltzustände wie z.B. das Ergebnis einer Addition zu speichern. Dazu sind Schaltungen notwendig, deren Ausgangssignale sowohl von den momentanen Eingangssignalen als auch von den bisherigen Ausgangssignalen abhängen.
Eine Schaltung mit dieser Eigenschaft verhält sich zeitabhängig und wird als Schaltwerk oder sequentielle Schaltung bezeichnet. Im nächsten Abschnitt beschäftigen wir uns mit Schaltwerken zur Speicherung eines Binärzustandes.

3.1 Digitale Speicherelemente

Eine Schaltung zur Speicherung eines Binärzustandes muss in der Lage sein, einen zeitlich begrenzten Zustand am Eingang zu einem dauerhaften Ausgangszustand zu machen. Damit der Ausgangszustand auch nach dem Wegfall des Eingangszustandes weiterhin bestehen bleibt, muss er selbst auf einen Eingang zurückwirken können. Eine erste Versuchsschaltung könnte folgendermaßen aussehen:

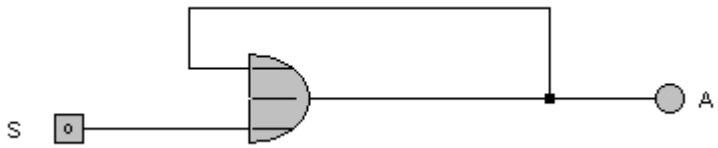


Abb. 3.1-1 ODER-Glied mit Rückkopplung des Ausganges auf einen Eingang

Die dargestellte Schaltung befindet sich in einem stabilen Zustand. Wenn wir den Eingang S mit 1-Signal belegen, geht der Ausgang in den 1-Zustand über. Durch die Rückkopplung erhält der zweite Eingang dann ebenfalls 1-Signal, so dass auch nach dem Wegfall des 1-Zustandes am Eingangs S das Ausgangssignal dauerhaft auf 1 bleibt. Die Schaltung „merkt“ sich sozusagen, dass der Eingang S einmal den Zustand 1 angenommen hatte.

Diese Schaltung hat aber den Nachteil, dass sich das Ausgangssignal nicht mehr verändern lässt, d.h. es kann nicht wieder zurückgesetzt werden. Dazu wäre eine Unterbrechung der Rückführungsleitung notwendig. Eine Unterbrechung können wir mit einer Torschaltung (UND-Glied) realisieren, so dass sich dieser Nachteil mit der folgenden Schaltung beheben lässt:

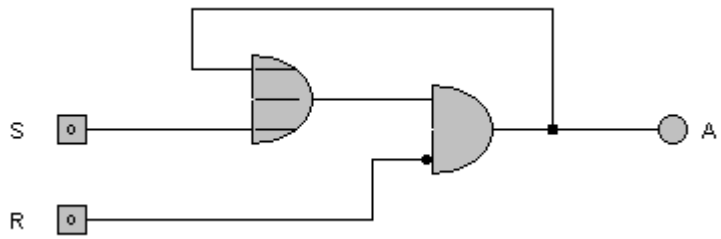


Abb. 3.1-2 ODER-Glied mit Unterbrechung der Rückführung durch eine Torschaltung

Ein 1-Signal auf der Leitung R (R=1) bewirkt wegen der Negation am UND-Eingang eine Unterbrechung der Rückführung. Durch R=1 kann die Schaltung also in den Ausgangszustand „gekippt“ werden, falls sich die Leitung S im 0-Zustand befindet.
Wegen der Möglichkeit, zwischen zwei Zuständen „hin- und herzukippen“, wird die eine solche Schaltung als *Kippschaltung* bzw. *Flipflop* bezeichnet.

3.1.1 RS-Flipflop

Üblicherweise werden Flipflops aus NOR- oder NAND-Gattern aufgebaut. Ein Austausch der Bauelemente und ein einfaches Umzeichnen der Schaltung aus Abb. 3.1-2 liefert das charakteristische Schaltbild eines Flipflops mit NOR-Gattern.

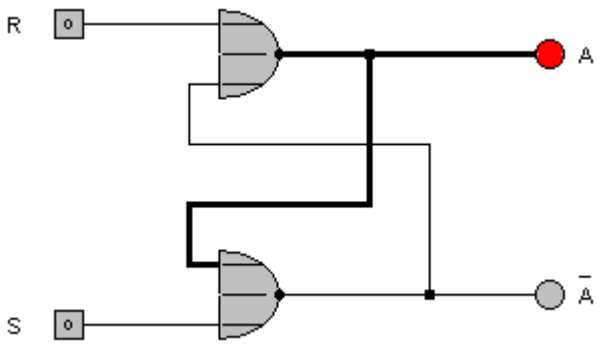


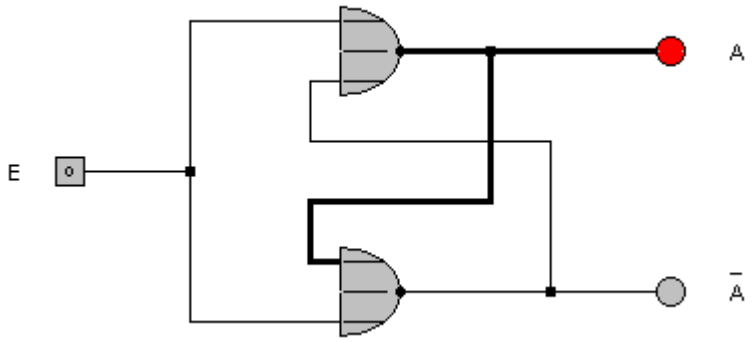
Abb. 3.1.1-1 RS-Flipflop aus NOR-Gattern

Die Schaltung wird als RS-Flipflop bezeichnet und besitzt neben dem Ausgang A noch einen invertierten Ausgang $\overline{A}$.

Am Eingang S (Setzeingang) lässt sich mit S=1 das Signal des Ausganges A auf 1 setzen. Mit dem Eingang R (Rücksetzeingang) kann der Ausgang A durch R=1 auf 0 gesetzt werden. Legt man S=1 und R=1 an, so erhalten beide Ausgänge 0-Signal. Werden jetzt S und R beide auf 0 gelegt, so erhält man unterschiedliche Ausgangszustände, je nachdem ob zuerst S=0 oder R=0 erfolgt ist.

Aufgabe 3.1: Überprüfen Sie dieses Verhalten mit LOCAD

Aufgabe 3.2: Um eine gleichzeitige Änderung von R und S zu ermöglichen, wird die Schaltung folgendermaßen abgeändert:



Stellen Sie fest, was passiert, wenn die beiden Eingänge der NOR-Gatter gleichzeitig auf 0-Signal gelegt werden.

Wegen der oben gezeigten Schwierigkeiten muss der Zustand S=R=1 vermieden werden und wir erhalten folgende Zustandstabelle für das RS-Flipflop:

R	S	A	$\overline{A}$	
0	0	keine Änderung, d.h.		speichern
0	1	1	0	setzen
1	0	0	1	rücksetzen
1	1	-	-	vermeiden

Das RS-Flipflop hat grundlegende Bedeutung (Grundflipflop), da sich alle komplexeren Flipflop-Typen damit realisieren lassen.

Ein zeitlich begrenztes Signal an einem Eingang eines RS-Flipflops kann den Zustand der Ausgänge bleibend verändern. Daher lässt sich damit die kleinste Informationseinheit (1 Bit) speichern.

3.1.2 Getaktetes RS-Flipflop

In der Regel müssen bei Speichervorgängen mehrere Bits (meistens 8) gleichzeitig übernommen werden. Hierzu ist es notwendig, die Flipflops zu synchronisieren. Das heißt, eine am Eingang vorliegende Information darf erst auf ein bestimmtes Signal hin übernommen werden. Bei einem einfachen RS-Flipflop erfolgt der Zustandswechsel der Ausgänge aber sofort beim Eintreffen der Eingangssignale.

Wir erreichen eine Synchronisation, indem wir vor die Eingänge der RS-Flipflops Tore legen, die durch das Signal T (Takt), das allen Flipflops zugeführt wird, geöffnet bzw. geschlossen werden können.

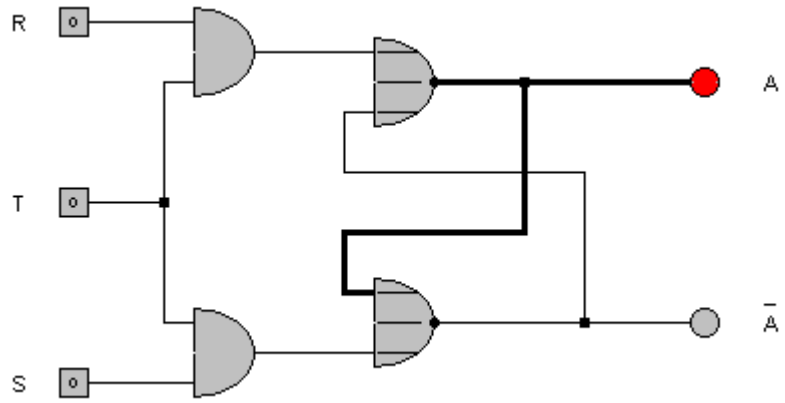


Abb. 3.1.2-1 Getaktetes RS-Flipflop (RS-Auffang-Flipflop)

Hierdurch wird die Information aufgefangen und erst auf Befehl (T=1) weitergegeben. Bei T=0 geben beide Tore am Ausgang 0-Signal ab, so dass das Grundflipflop seinen bisherigen Zustand beibehält. Man sagt, die Eingänge R und S sind gesperrt.
Wir erhalten folgende Zustandstabelle, in der die Zustände vor und nach dem Eintreffen des Taktimpulses durch die Indizes v und n unterschieden sind.

S	R	T	A_n	
b	b	0	A_v	
0	0	1	A_v	
1	0	1	1	
0	1	1	0	
1	1	1	-	vermeiden

(b = beliebig)

3.1.3 D-Auffangflipflop

Eine spezielle Variante des getakteten RS-Flipflops ist das sogenannte D-Auffangflipflop, das häufig auch als Data-Latch bezeichnet wird.

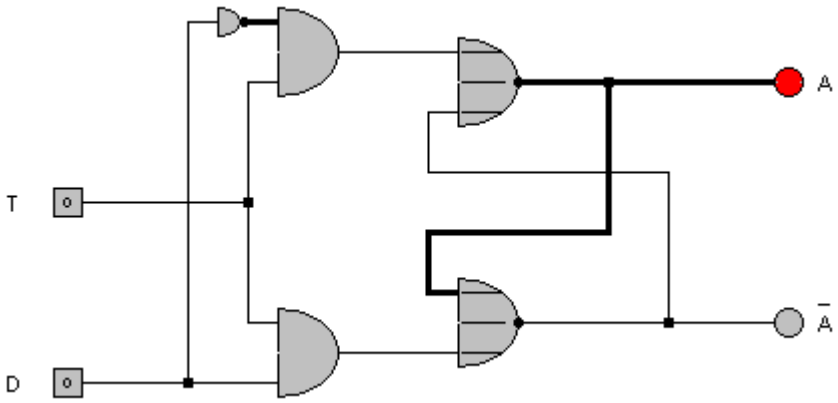


Abb. 3.1.3-1 D-Auffangflipflop

Der R-Eingang wird über einen Inverter mit dem S-Eingang verbunden, so dass nur noch ein Eingang übrigbleibt, der als Dateneingang bezeichnet wird.
Damit haben R und S nie den gleichen Zustand und solange das Taktsignal im 1-Zustand ist wird die gerade an D anliegende Information (Date) übernommen. Die an D ankommenden Daten hält es erst dann fest, wenn das Taktsignal in den 0-Zustand geht. In diesem Sinne fängt es Daten auf und wird deshalb als Auffangflipflop bezeichnet. Solange der Takt im 1-Zustand ist, „rastet“ der Ausgang stets auf das Signal am D-Eingang ein. Deshalb wird dieses Flipflop auch D-Latch genannt (Latch [engl.] = Raste, Klinke).
D-Auffangflipflops werden häufig als schnelle Zwischenspeicher verwendet.

Zustandstabelle:

D_n	A_n
0	0
1	1

3.1.4 RS-Master-Slave-Flipflop

In vielen Anwendungen müssen zwei Speicherelemente so hintereinandergeschaltet werden, dass das nachfolgende die gespeicherte Information des vorhergehenden übernimmt, während dieses eine neue Eingangsinformation aufnehmen muss.
Mit getakteten RS-Flipflops ist das nicht realisierbar, denn bei T=1 werden die von den Eingängen übernommenen Zustände sofort auf die Ausgänge übertragen, so dass das nachfolgende Flipflop ebenfalls diese Zustände übernehmen würde.

Aufgabe 3.3: Schalten Sie zwei getaktete RS-Flipflops in oben beschriebenem Sinne hintereinander und untersuchen Sie das Verhalten der Schaltung.

Es muss verhindert werden, dass sich die Eingangszustände sofort auf die Ausgänge auswirken. Das ist durch eine schleusenartige Hintereinanderschaltung zweier getakteter RS-Flipflops zu einem Speicherelement möglich.

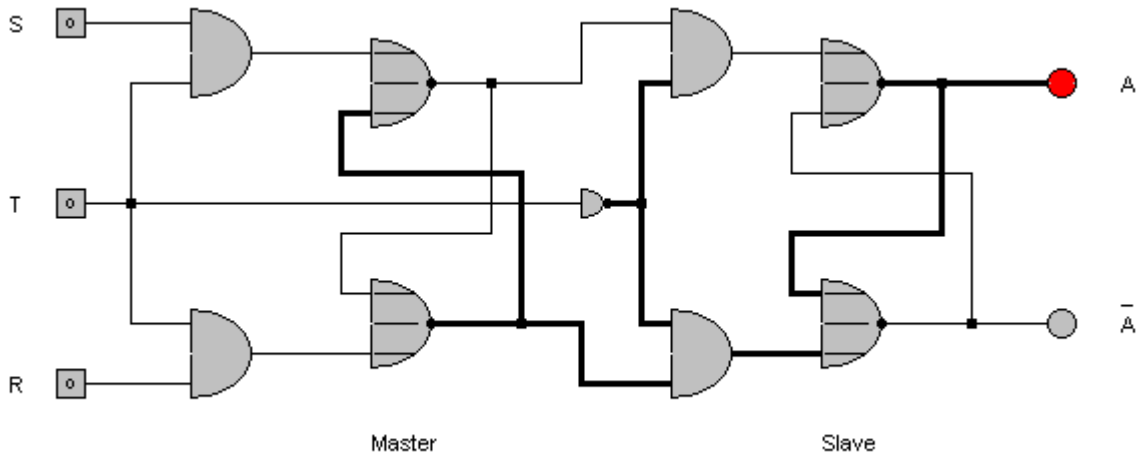


Abb. 3.1.4-1 RS-Master-Slave-Flipflop

Entscheidend für die Funktionsweise ist die Tatsache, dass die beiden getakteten RS-Flipflops durch den Inverter in der Taktleitung im „Gegentakt“ arbeiten. Wenn das eine Grundflipflop Werte übernehmen kann, ist das andere gesperrt.

Das erste Flipflop nennt man Master, das zweite Slave. Diese Bezeichnung ist darauf zurückzuführen, dass das zweite Flipflop stets vom ersten abhängt und dessen Werte übernimmt.
Liegt der Takt auf 0, so ist das Master-Flipflop gesperrt. Das Slave-Flipflop hat dann den Takt 1 und übernimmt damit die Werte des Master-Flipflops. Da diese aber stabil sind, bleiben auch die Ausgänge stabil.

Wenn der Takt auf 1 wechselt, werden die Tore zum Master-Flipflop geöffnet und die anliegenden Eingangszustände werden übernommen.

Da der Ausgang eines jeden Schaltgatters erst nach einer gewissen Verzögerungszeit auf eine Änderung am Eingang reagiert, liegen die Ausgangssignale am Master-Flipflop wegen der größeren Zahl der zu durchlaufenden Gatter später vor als das Takt-Signal am Ausgang des Inverters in der Taktleitung. Somit wird das Slave-Flipflop gesperrt, bevor es durch die veränderten Werte des Masters beeinflusst werden kann.
Erst wenn der Takt wieder auf 0 abfällt, übernimmt das Slave-Flipflop die Werte des Masters und stellt sie am Ausgang zur Verfügung.
Das obige Master-Slave-Flipflop reagiert somit auf fallende Taktflanken. Man sagt, es ist „negativ flankengesteuert“. Es gibt auch positiv flankengesteuerte Master-Slave-Flipflops.

Die schleusenartige Hintereinanderschaltung zweier Flipflops zu einem Speicherelement ermöglicht es also, während der Ausgabe der Information durch das Slave-Flipflop, im Master-Flipflop bereits eine neue Information aufzunehmen, ohne dass sich dadurch die Information am Ausgang ändert.
Diese Eigenschaft ist für den Aufbau von Schieberegistern von besonderer Bedeutung.

3.1.5 JK-Master-Slave-Flipflop

Auch das RS-Master-Slave-Flipflop hat noch die unschöne Eigenschaft, dass die Eingangskombination S=R=1 vermieden werden muss. Mit der folgenden Modifikation wird nun dieser Nachteil behoben:

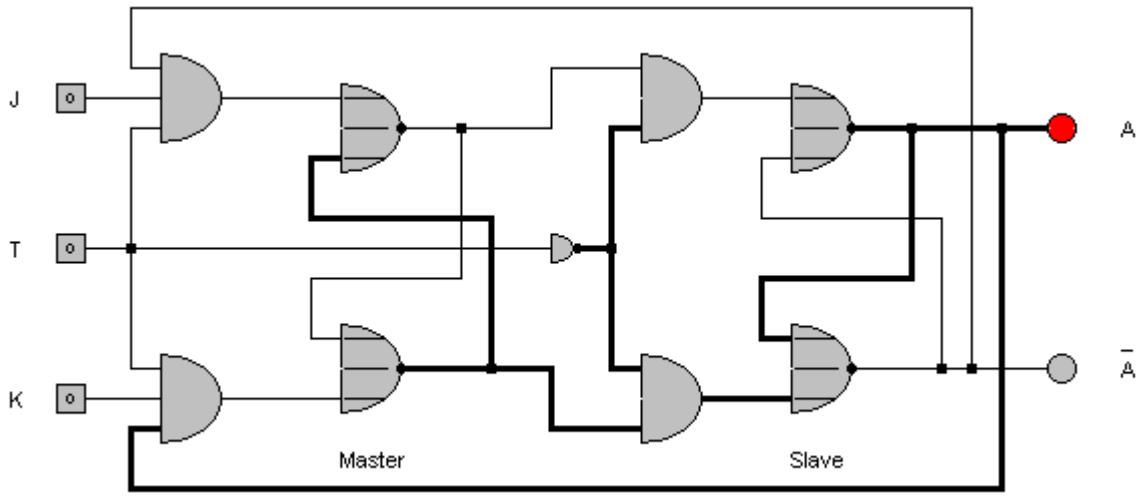


Abb. 3.1.5-1 JK-Master-Slave-Flipflop

Die bisherigen Eingänge S und R sind hier mit J und K bezeichnet. Der Unterschied zum RS-MS-FF besteht darin, dass die Ausgänge über Kreuz auf die Eingänge zurückgeführt sind. Das hat zur Folge, dass immer nur derjenige Eingang durchgeschaltet wird, der eine Änderung des momentanen Zustandes auslösen kann. Damit ist auch die Kombination J=K=1 zugelassen und bewirkt mit jeder negativen Taktflanke einen Wechsel des Ausgangszustandes.

Man erhält folgende Zustandstabelle:

J	K	A_n		
0	0	A_v	speichern	
1	0	1	setzen	bei der nächsten
0	1	0	rücksetzen	negativen Takt-
1	1	$\overline{A}_v$	wechseln	flanke

Aufgabe 3.4: Bauen oder laden Sie das JK-MS-FF und machen Sie sich die Funktionsweise klar.

In der Praxis werden JK-Master-Slave-Flipflops hergestellt, die noch zusätzlich statische R- und S-Eingänge haben, mit denen ohne Taktimpuls gewünschte Anfangswerte eingeschrieben werden können.
Damit diese Eingänge Vorrang haben, wird durch das Einschreibe-Signal der Takteingang gesperrt. Die folgende Abbildung zeigt die erweiterte Schaltung.

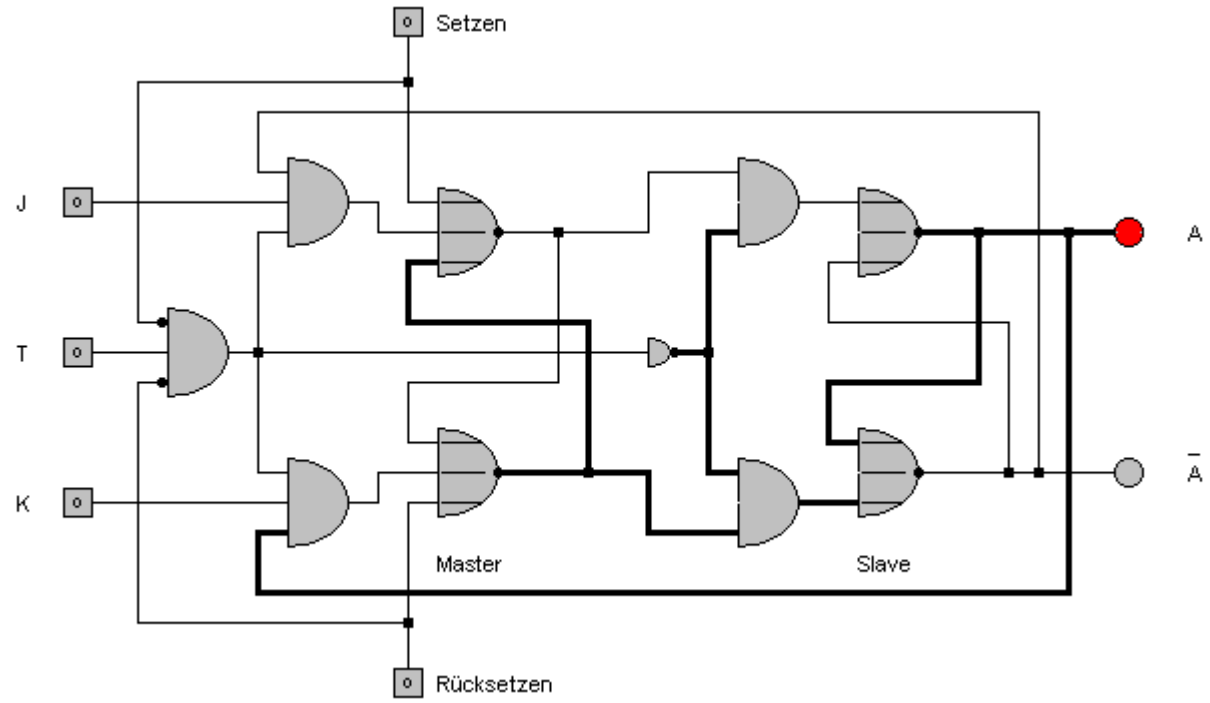


Abb. 3.1.5-2 JK-MS-FF mit statischem Setz- und Rücksetzeingang

Dieses Flipflop hat nun fast alle Eigenschaften, um als universell einsetzbares Speicherelement zu gelten. Lediglich die Störsicherheit lässt noch etwas zu wünschen übrig. Mit einer kleinen schaltungstechnischen Erweiterung ist aber auch dieser Nachteil zu beheben.

Der Slave dieses Flipflops reagiert wie oben bereits beschrieben auf fallende Taktflanken.
Das Ausgangsverhalten (Schaltverhalten des Slave) entspricht also einer Flankentriggerung (Flankensteuerung).
Das Probleme der Störimpulsanfälligkeit liegt in der Taktzustandssteuerung des Masters. Er ist so lange geöffnet, wie das Taktsignal den Zustand 1 hat und kann somit während dieser Zeit kurzfristige - von Störsignalen hervorgerufene - Eingangssignaländerungen aufnehmen.
Sie können sich dieses Verhalten klarmachen, indem Sie in der obigen Schaltung am T-Eingang ein 1-Signal anlegen und danach am K-Eingang durch einen 1-0-Wechsel ein kurzes Störsignal erzeugen. Der Master wird dadurch umgeschaltet und gibt die geänderte Information beim Rückfall des Taktsignals auf 0 an den Slave weiter. In diesem Sinne ist das obige Flipflop nicht störsicher. Ein Störimpuls während des 1-Zustandes des Taktsignals kann das Flipflop in einen nicht beabsichtigten Zustand bringen.

Das führt in einer umfangreichen Schaltung unter Umständen zu einem Fehlverhalten. Einen Ausweg aus dieser Situation bietet eine echte Zweiflankensteuerung. Hierbei wird die anstehende Information nur zu einem ganz definierten Zeitpunkt, der durch eine Taktflanke festgelegt wird, vom Master übernommen. Dadurch ist das Flipflop gegen Störimpulse auf den Eingangsleitungen geschützt, sofern diese nicht gleichzeitig mit der Taktflanke eintreffen (was allerdings viel seltener vorkommt).
Um das obige Flipflop in ein zweiflankengesteuertes FF umzuwandeln, müssen wir jetzt nur noch dafür sorgen, dass die Eingangsinformation nur bei ansteigender Taktflanke in den Master übernommen werden kann. Wir erreichen das, indem wir die Tore zum Master mit einem Impuls nur kurzzeitig öffnen, und zwar im Moment des Taktflankenanstiegs. Der Impuls muss allerdings so lang sein, dass ein sicheres Schalten des Mastes gewährleistet ist.

Das Prinzip der Impulserzeugung ist aus Abschnitt 2.7 bekannt. Wenn wir eine Impulserzeugungsschaltung nach folgender Abbildung in die Taktleitung zum Master legen, ist das Flipflop zweiflankengesteuert und damit gegen Störimpulse weitestgehend gesichert.

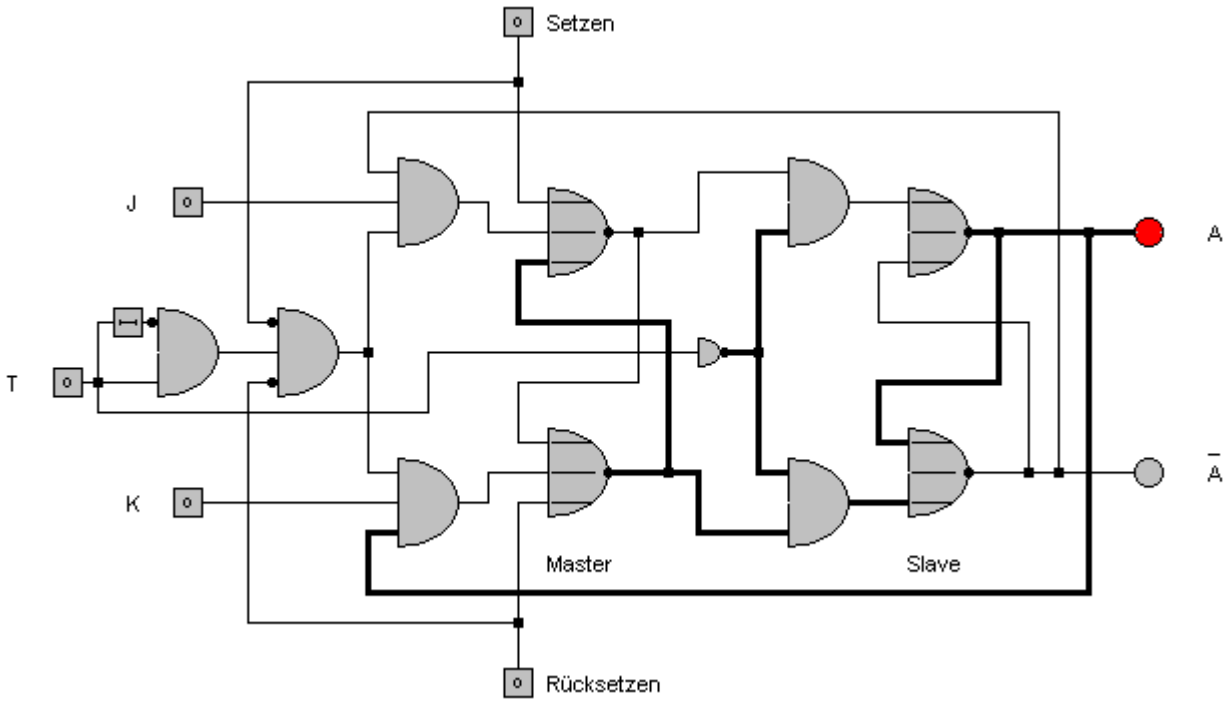
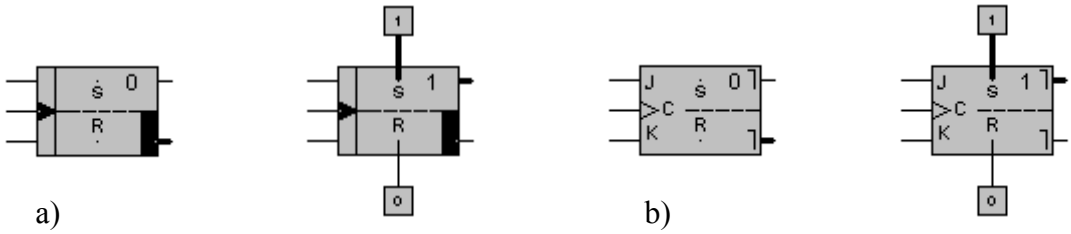


Abb. 3.1.5-3 a) JK-MS-FF mit statischen Eingängen b) mit Leitungen zu den statischen Eingängen

Da dieses Flipflop ein universell einsetzbares Speicherelement ist und damit eine große Bedeutung in der Digitaltechnik hat, wurde es auch in *LOCAD* als eigenständiges Bauelement vorgesehen.

Beim Einsatz dieses Flipflops ist das folgende Schaltverhalten zu beachten:

> Nur während der positiven Taktflanke werden Eingangssignale in den Master übernommen. Signaländerungen und Störimpulse während der 1-Phase des Taktes bleiben ohne Einfluss.



a) b)

Abb. 3.1.5-4 Schaltsymbole für das zweiflankengesteuerte JK-MS-Flipflop mit statischen Eingängen S und R und Leitungen hierzu
a) alte DIN-Norm b) neue DIN-Norm

Zum Abschluss noch einige Bemerkungen zu den Schaltsymbolen für das Flipflop nach der alten und neuen DIN-Norm.

In beiden Norm-Varianten gilt:

Ein offenes Dreieck am Takteingang symbolisiert das Schalten bei positiver Taktflanke.

In der alten Norm wurde als Symbol für das Schalten bei negativer Taktflanke ein ausgefülltes Dreieck benutzt. Die neue Norm benutzt hierfür das offene Dreieck mit einem vorgesetzten Negationspunkt (das ausgefüllte Dreieck gibt es hier nicht mehr).

Schaltsymbol in der alten Norm:
Das Dreieck am Takteingang zeigt die Triggerung für den Ausgang (Slave) an. Da der Slave bei negativer Taktflanke schaltet, wird ein ausgefülltes Dreieck gezeichnet.
Der Balken am unteren Ausgang zeigt an, dass dieser beim Einschalten des Flipflops in den 1-Zustand geht (Vorzugslage).

Schaltsymbol in der neuen Norm:
Am Takteingang wird hier die Triggerung für den Master angegeben. Da der Master bei positiver Taktflanke schaltet, wird ein offenes Dreieck gezeichnet. Das verzögerte Schalten des Slave wird hier durch das ⌉- Zeichen an den Ausgängen deutlich gemacht.

Um das Flipflop mit den statischen Eingängen setzen bzw. zurücksetzen zu können, muss eine Leitung zum entsprechenden Eingang (Punkt oberhalb von S bzw. unterhalb von R) gezogen werden. Setzen bzw. Rücksetzen ist dann mit einem 1-Signal möglich.

Achten Sie bitte bei der Verwendung von Flipflops auf folgende Besonderheiten:

1. An nicht beschalteten Ein- und Ausgängen werden bei Inbetriebnahme der Schaltung die herausgeführten Leitungsstücke gelöscht. Selbstverständlich können Sie anschließend hier wieder Leitungen anschließen.

2. Nicht beschaltete Eingänge werden wie bei realen Flipflops als mit 1-Signal belegt betrachtet. Damit schaltet ein Flipflop, dessen JK-Eingänge offen sind, mit jeder fallenden Taktflanke seinen Ausgangszustand um.

3. Um platzsparende Schaltpläne zu erhalten, können Sie anstelle eines Negators in der Zuleitung zum K-Eingang eines Flipflops den K-Eingang direkt mit einem Negationspunkt versehen. Auf diese Weise erhalten Sie ein D-Flipflop, das mit jeder negativen Taktflanke den Wert am J-Eingang übernimmt.

3.2 Rechenwerke

Die im letzten Abschnitt besprochenen Flipflops bilden die Grundlage für die Entwicklung von Speicherelementen, die mehrere Bits (in der Regel 8 bzw. 16) speichern können. Diese Elemente bestehen im Prinzip aus einer Reihe von Flipflops und werden als Register bezeichnet. Sind die einzelnen Flipflops so miteinander verbunden, dass mit jedem Taktimpuls eine Verschiebung der Information um eine Stelle nach links bzw. rechts erfolgt, so liegt ein sogenanntes Schieberegister vor. Diese Registerart hat in der EDV eine besondere Bedeutung, da sich damit beispielsweise Daten von der seriellen Form in die parallele und umgekehrt umsetzen lassen. Weiterhin lässt sich die Multiplikation einer Dualzahl mit 2 durch eine Linksschiebung um eine Stelle durchführen. Entsprechend liefert eine Rechtsschiebung um eine Stelle eine Division durch 2.

3.2.1 Schieberegister

Damit der Schaltungsaufwand nicht zu groß wird, wollen wir uns im folgenden auf 4 Bit beschränken. Um ein 4-Bit-Schieberegister aufzubauen, das bei jeder negativen Taktflanke die eingeschriebene Information um eine Stelle nach rechts schiebt, müssen wir 4 JK-MS-Flipflops so hintereinanderschalten, dass die Eingänge direkt mit den Ausgängen des jeweils davorliegenden Flipflops verbunden sind. Alle Flipflops müssen synchron getaktet werden, d.h. alle Takteingänge müssen an die gleiche Taktleitung angeschlossen werden.

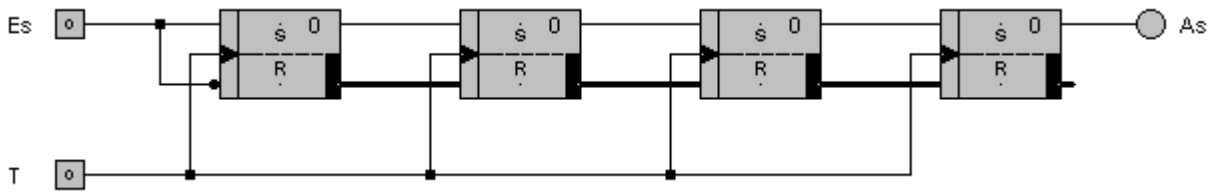


Abb. 3.2.1-1 4-Bit-Rechtsschieberegister

Der J-Eingang des ersten Flipflops ist der serielle Dateneingang des Schieberegisters. Er ist über einen Negator mit dem K-Eingang verbunden. Damit sind die Eingänge J und K immer komplementär zueinander und es werden nur die Funktionen Setzen bzw. Rücksetzen ausgeführt (siehe Zustandstabelle). Das Datensignal auf der Leitung *Es* (*E*ingabe *s*eriell) wird bei negativer Taktflanke vom ersten Flipflop übernommen, während das folgende den bisherigen Zustand des vorhergehenden übernimmt.

Aufgabe3.5: Bauen Sie das Schieberegister mit *LOCAD* auf und testen Sie es.

Eine Ausgabe der im Schiebregister gespeicherten Informationen ist auf zwei Arten möglich:

- serielle Ausgabe
  Am Ausgang *As* (*A*usgang *s*eriell) des letzten Flipflops werden die durch Verschiebung nacheinander eintreffenden Binärzustände abgegriffen und weiter verarbeitet

- parallele Ausgabe
  Die Ausgänge aller Flipflops werden über Tore, die mit einem Ausgabesignal geöffnet werden können, auf die Ausgabeleitungen geschaltet

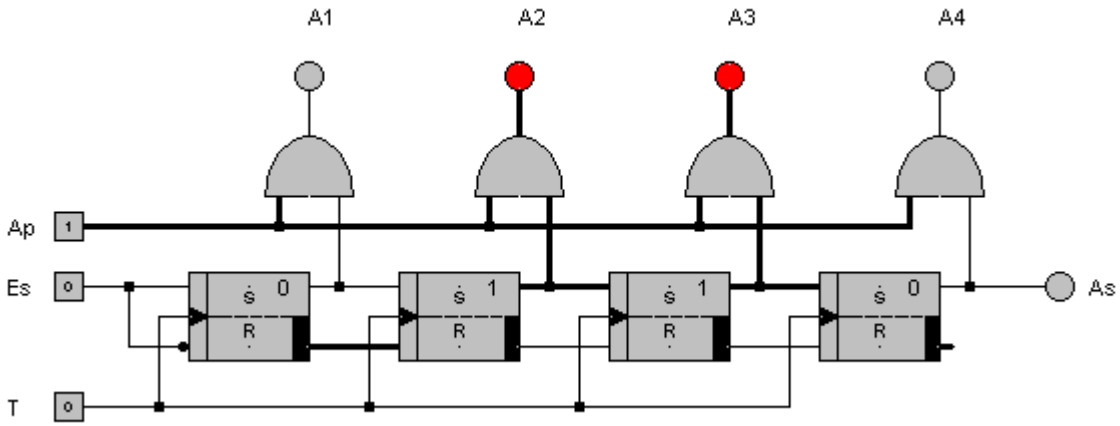


Abb. 3.2.1-2 Parallele Datenausgabe

Ein 1-Signal auf der Leitung *Ap* (*A*usgabe *p*arallel) führt dazu, dass die einzelnen Flipflop-Zustände auf die Ausgabeleitungen A1 bis A4 durchgeschaltet werden.

Die folgende Abbildung zeigt eine Möglichkeit zur parallelen Dateneingabe in ein Schieberegister.

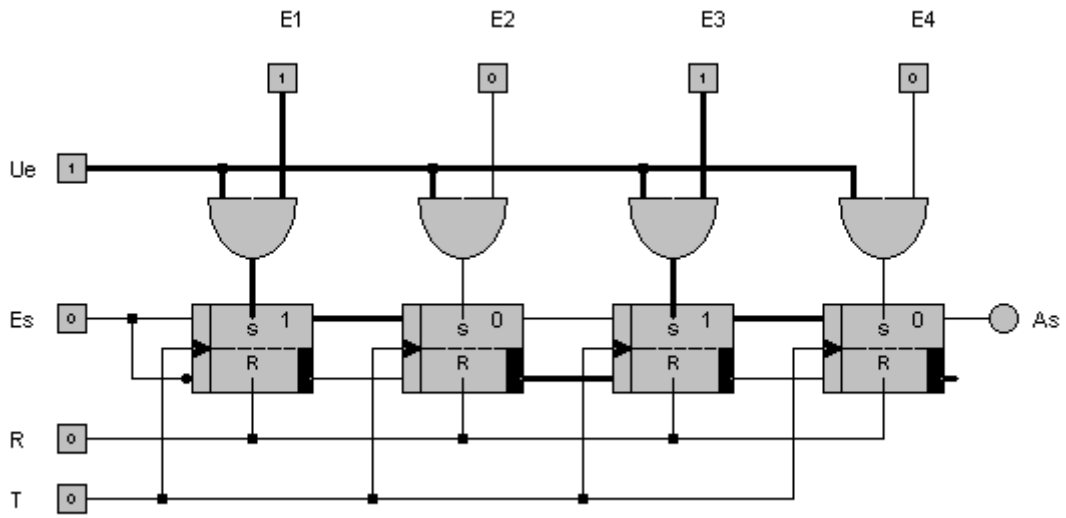


Abb. 3.2.1-3 Parallele Dateneingabe

Die statischen Setzeingänge sind über Tore mit den Datenleitungen verbunden. Bei einem 1-Signal auf der Übernahmeleitung *Ue* werden die Datenleitungszustände in die einzelnen Flipflops übernommen.

Ue = 1 : Paralleles Einlesen der Zustände E1, ..., E4
Ue = 0 : normaler Schiebebetrieb

Aufgrund der verschiedenen Ein- und Ausgabemöglichkeiten kann ein Schieberegister als Serien-/Parallelcode- bzw. Parallel-/Seriencodeumsetzer benutzt werden.

Eine Anwendung dieser Umsetzungsmöglichkeiten sehen Sie in der folgenden Abbildung.

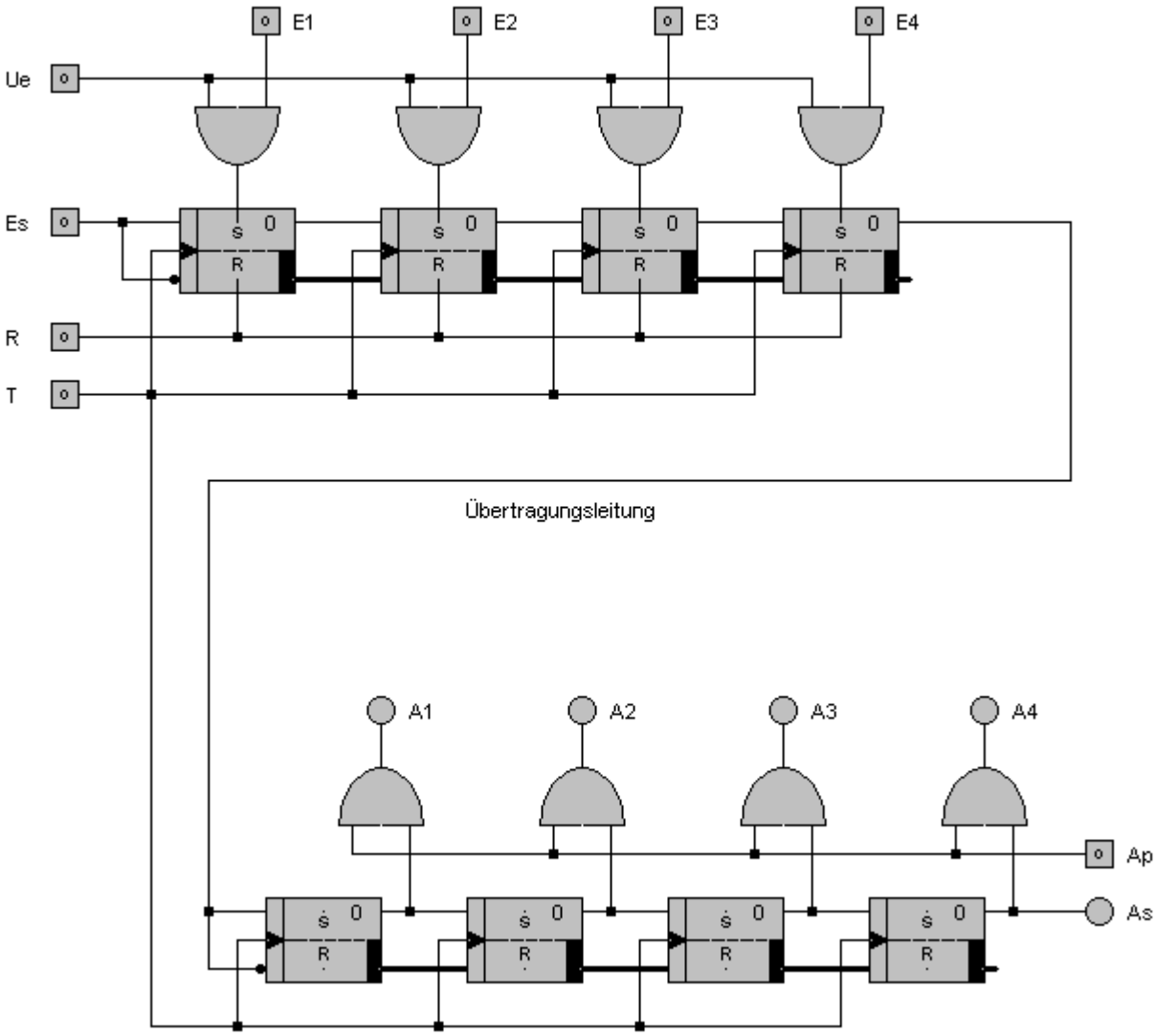


Abb. 3.2.1-4 Serielle Übertragung paralleler Daten über eine Leitung

Wenn eine bestimmte Zustandskombination an den Datenleitungen E1 bis E4 vorliegt kann diese durch ein 1-Signal auf der Übernahmeleitung *Ue* in das Schieberegister eingeschrieben werden. Nach dem Zurücksetzen des Übernahmesignals werden die einzelnen Binärzustände nacheinander mit vier Schiebtakten auf die Übertragungsleitung gelegt. Das synchron getaktete Schiebregister am anderen Ende der Leitung übernimmt die Zustände nach und nach. Hier können sie dann durch ein 1-Signal auf der Leitung *Ap* (*A*usgabe *p*arallel) auf die Ausgabeleitungen A1 bis A4 durchgeschaltet werden.
Wir haben damit parallel vorliegende Daten über **eine** Leitung zu einer entfernten Einheit transportiert und dort wieder in paralleler Form abgegeben.

Schieberegister sind in *LOCAD* als eigenständige Bausteine vorgesehen.

a) b)

Abb. 3.2.1-5 Schieberegister in *LOCAD*
a) mit Takteingang nach alter DIN-Norm b) mit Takteingang nach neuer DIN-Norm

Das abgebildete Bauelement beinhaltet zwei voneinander unabhängige 4-Bit-Schieberegister. Die unteren Zahlen sind also nicht als Komplementärwerte der oberen zu verstehen. Mit diesem Bauelement lässt sich sehr einfach ein 8-Bit-Schieberegister realisieren, indem man den Ausgang der oberen Reihe auf den Eingang der unteren Reihe schaltet.
Die einzelnen Schieberegisterstellen können mit Anfangswerten belegt werden. Während des Betriebs einer Schaltung können die Schieberegisterinhalte durch Anklicken mit der linken Maustaste geändert werden.

Wichtiger Hinweis:
Auf diese Weise können auch die Inhalte einzelner Flipflops geändert werden.

Aufgabe 3.6: Entwickeln Sie mit einzelnen Flipflops ein 4-Bit-Schieberegister, das seinen Inhalt wahlweise nach links und rechts verschieben kann.
(Lösung im Anhang C)

3.2.2 Serienaddierwerk

Mit Hilfe von Schieberegistern sind wir nun in der Lage, ein sehr einfaches Addierwerk aufzubauen. Das in Kapitel 2 entwickelte Paralleladdierwerk arbeitet zwar sehr schnell, hat aber den Nachteil, dass zur Addition zweier n-stelliger Dualzahlen n-1 Volladdierer und ein Halbaddierer benötigt werden. Schieberegister bieten die Möglichkeit, mit einem Volladdierer auszukommen, indem die Zahlen stellenweise addiert werden, so wie man es von der schriftlichen Addition her gewohnt ist. Hierbei wird die entstehende Summe in ein Ergebnisregister eingeschrieben, ein eventuell vorliegender Übertrag in einem Flipflop zwischengespeichert

und anschließend alle Bits um eine Stelle nach rechts geschoben. Bei 4-stelligen Dualzahlen sind also vier Schiebetakte erforderlich. Steht nach dem letzten Takt noch eine 1 im Übertragsflipflop, so bedeutet das, dass das Ergebnis nicht mit 4 Stellen dargestellt werden kann, also ein Überlauf vorliegt. Die folgende Abbildung zeigt die Schaltung.

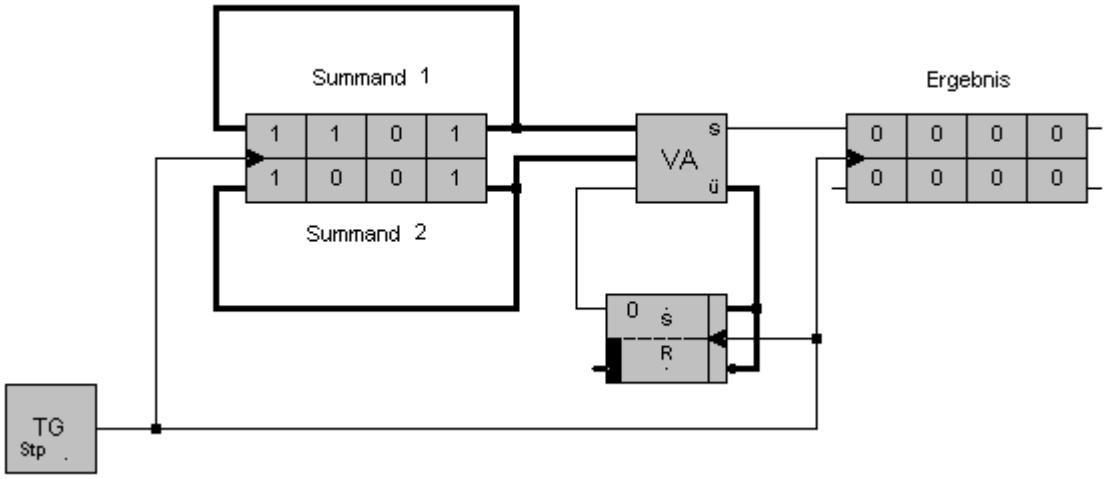


Abb. 3.2.2-1 Serienaddierwerk

In der Schaltung wurde ein Taktgeber eingesetzt, damit die Schiebetakte nicht immer von Hand ausgelöst werden müssen. Bei der Inbetriebnahme der Schaltung über den Hauptmenüpunkt *Einschalten* haben Sie in der Schaltflächenzeile die Möglichkeiten Einzeltakte, Dauertakt oder eine festgelegte Anzahl von Takten ablaufen zu lassen. Genauere Informationen zur Einstellung des Taktgenerators finden Sie in Abschnitt 2.7 in der Anleitung zur Handhabung.

Im obigen Serienaddierwerk müssen vier Takte ablaufen. Wenn Sie den Button *Taktzahl* anklicken, werden Sie aufgefordert, die Anzahl der Takte anzugeben, die Sie anschließend mit Klick auf den Button *Taktfolge* ausführen lassen können.

Der Taktgeber besitzt einen Stop-Eingang (Punkt rechts neben STP).
Sie können hier eine Leitung anschließen, mit der der Taktgeber durch ein 1-Signal gestoppt werden kann. Wenn der Stop-Eingang 1-Signal hat, können Sie keine Einzeltakte auslösen und auch keine Taktfolge starten.

Aufgabe 3.7: Bauen Sie das Serienaddierwerk auf und machen Sie sich die Funktionsweise klar, indem Sie einzelne Takte auslösen und sich die Zwischenzustände ansehen.

Es ist nun naheliegend, auch das umschaltbare Rechenwerk aus Kapitel 2 mit Hilfe von Schieberegistern zu einem umschaltbaren seriellen Rechenwerk umzubauen.

3.2.3 Umschaltbares serielles Rechenwerk

Schauen Sie sich in Kapitel 2 noch einmal das Prinzip an, nach dem es möglich ist, eine Subtraktion mit Hilfe eines Addierers durchzuführen.
Das umschaltbare Rechenwerk soll wie in Kapitel 2 mit Hilfe eines Signals umgeschaltet werden können. Mit einen 1-Signal wollen wir das Werk in den Subtraktions-Modus versetzen. Hierbei müssen die Bits des Subtrahenden invertiert und das Übertragflipflop mit einer 1 vorbelegt werden.
Die Invertierung der Bits erreichen wir wieder durch Unterbrechung der Leitung vom Subtrahenden zum Volladdierer mit einem EXKLUSIV-ODER-Glied, dessen zweiter Eingang mit der Steuerleitung verbunden ist.
Wir können aber nicht den Setzeingang des Flipflops ebenfalls an die Steuerleitung anschließen, denn dann würde andauernd eine 1 am Flipflop-Ausgang vorliegen. Das gleiche gilt für den Additions-Modus des Rechenwerks. Der Rücksetzeingang darf nicht dauernd mit 1-Signal belegt sein, sondern das Flipflop muss am Anfang einmal auf Null gesetzt werden und danach Überträge zwischenspeichern können. Hier können wir die Impulserzeugungsschaltungen aus Abschnitt 2.7 wiederum gut einsetzen.

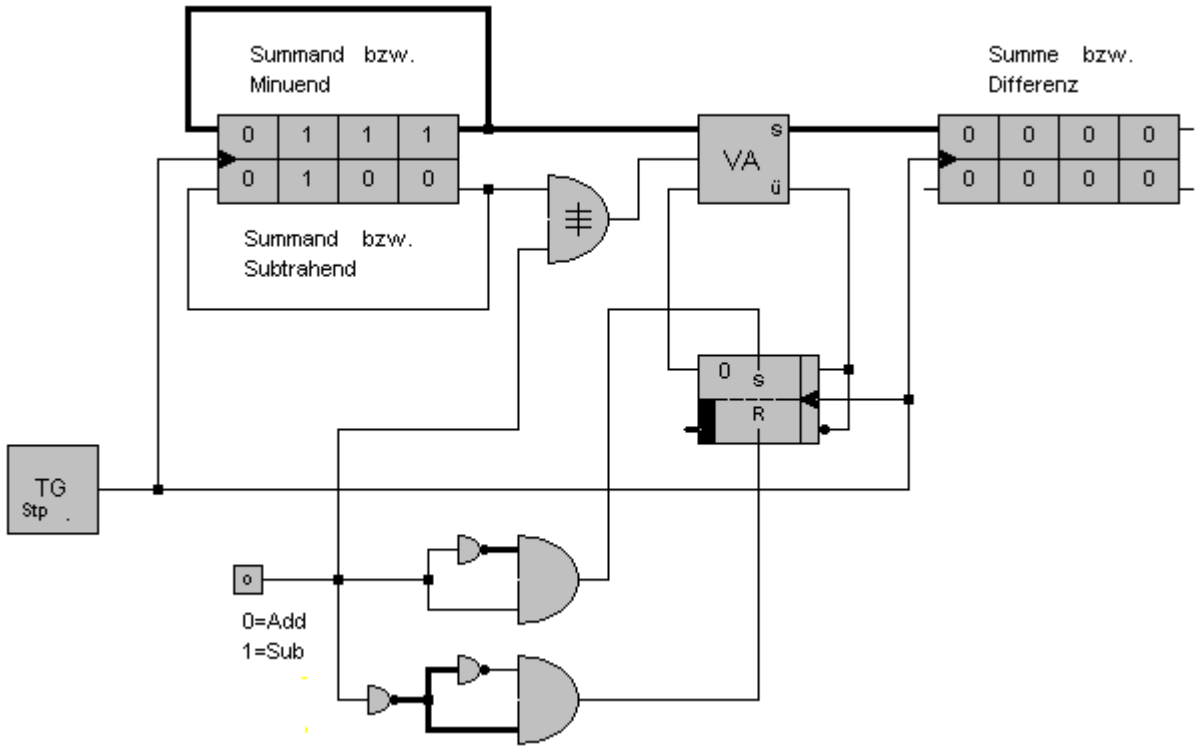


Abb. 3.2.3-1 Umschaltbares serielles Rechenwerk

Aufgabe 3.8: Bauen Sie das Rechenwerk auf und überprüfen Sie die Funktionsweise.

3.3 Zählschaltungen

In den oben entwickelten Rechenwerken konnte die Berechnung mit Hilfe des Taktgebers durch eine vorgegebene Taktfolge automatisch ausgeführt werden. Nachdem die angegebene Anzahl von Takten abgelaufen war, wurde der Taktgeber angehalten. In realen Systemen wird aber üblicherweise nicht der Taktgeber gestoppt, wenn eine bestimmte Aktion beendet ist, sondern aus dem andauernden Taktsignal wird eine ganz bestimmte Anzahl von Takten „ausgekoppelt“. Mit Hilfe von Zählschaltungen, die sich sehr einfach mit JK-MS-Flipflops aufbauen lassen, können die benötigten Takte abgezählt werden.

Wenn wir beachten, dass die JK-Eingänge der Flipflops in unbeschaltetem Zustand den Signalwert 1 annehmen, können wir uns die Funktionsweise der nachstehenden Schaltung leicht klarmachen.

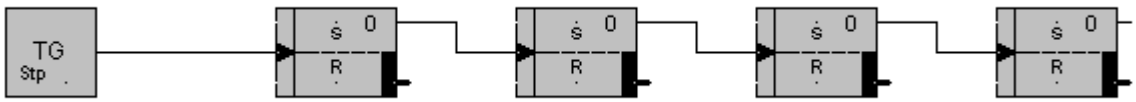


Abb. 3.3-1 Dualzähler

Aufgabe 3.9: Bauen Sie den Zähler mit LOCAD auf und untersuchen Sie seine Funktionsweise

Wenn man die Reihenfolge der Flipflopzustände umkehrt, erhält man die Anzahl der gezählten Takte in Dualdarstellung.

Mit Hilfe einer Ziffernanzeige lässt sich das Zählergebnis direkt dezimal anzeigen.

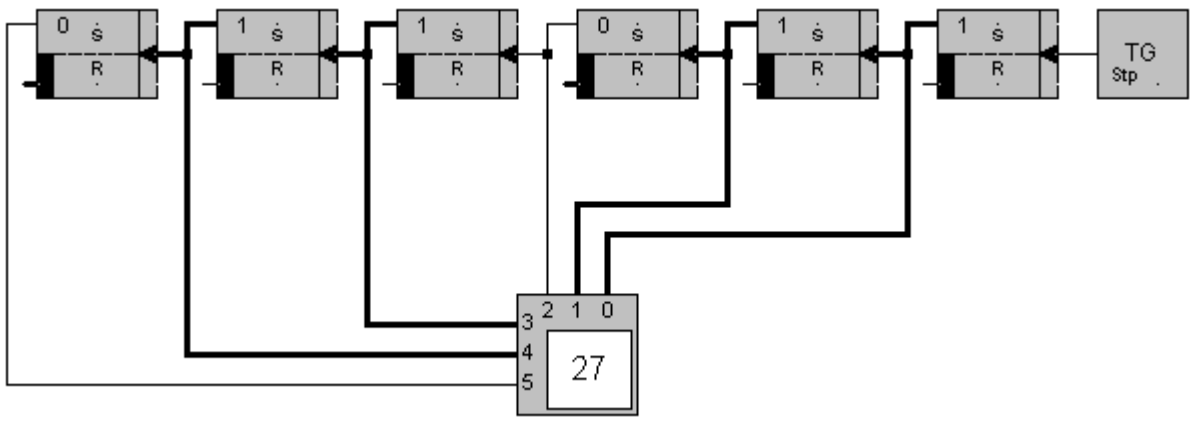


Abb. 3.3-2 Dualzähler mit Ziffernanzeige für 6 Bit

Die in der obigen Schaltung eingesetzte Ziffernanzeige stellt die Dualzahlen 000000 bis 111111 als Dezimalzahlen 0 bis 63 dar.

Für umfangreichere Anwendungen wurde in *LOCAD* auch noch eine Anzeigeeinheit für 8 Bit vorgesehen, die die Zweierkomplementdarstellung von 8-Bit-Zahlen auswertet. Die Bitkombination 00000111 wird demnach als Dezimalwert 7 und 10000111 als –121 dargestellt.

Abb. 3.3-3 Ziffernanzeige für 8-Bit-Zahlen im Zweierkomplement

Bei den Zählern nach Abbildung 3.3-1 und 3.3-2 handelt es sich um einen Asynchron-Zähler, da nicht alle Flipflops gleichzeitig getaktet werden, sondern außer dem ersten alle weiteren vom Ausgang des jeweils davor liegenden gesteuert werden. Das hat zur Folge, dass im ungünstigsten Fall mit dem nächsten Taktimpuls so lange gewartet werden muss, bis das letzte Flipflop in der Kette seine Zustandsänderung abgeschlossen hat. Der einfache Aufbau hat also die Konsequenz, dass die Zählgeschwindigkeit mit der Anzahl der Flipflops abnimmt.

Mit Synchron-Zählern, bei denen alle Flipflops gleichzeitig getaktet werden, lässt sich dieser Nachteil vermeiden. Sie erfordern aber einen größeren Schaltungsaufwand.

Aufgabe 3.10: Bauen Sie die folgende Schaltung auf und machen Sie sich ihre Funktionsweise klar.

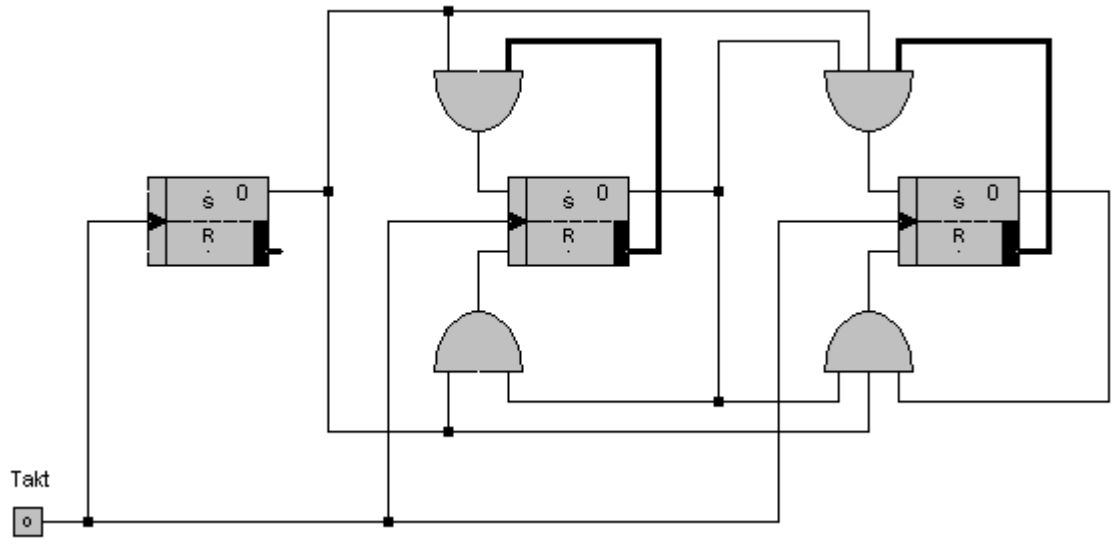


Abb. 3.3-4 Synchronzähler

Um die Signalzustände besser zu überblicken bzw. zu dokumentieren, können Sie Analysepunkte einsetzen. In *LOCAD* stehen 8 Analysepunkte mit den Bezeichnungen „A“ bis „H“ zur Verfügung, die als „Mess-Spitzen“ auf Leitungen aufgesetzt werden können. Die Analysepunkte sehen wie kleine LEDs aus und tragen zur Unterscheidung den kennzeichnenden Buchstaben in der Mitte.

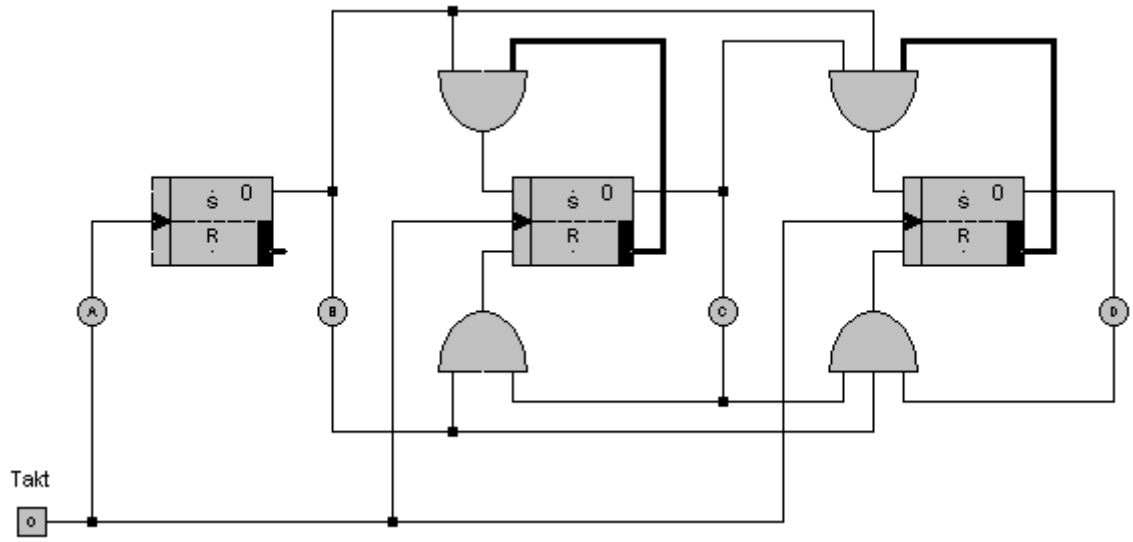


Abb. 3.3-5 Synchronzähler mit Analysepunkten auf einigen Leitungen

Wenn eine Schaltung Analysepunkte enthält, steht im Menü *Betrieb* ein Button zum Ein- und Ausschalten der Analysepunkte zur Verfügung. Die Aufschrift „AP-aus“ zeigt an, dass die Analysepunkte im Moment nicht aktiviert sind. Wenn Sie den Button einrasten, werden die Analysepunkte aktiv geschaltet (erkennbar an der Aufschrift „AP-ein“) und die an den Mess-Spitzen ankommenden Signale werden registriert. Beim Ausschalten der Analysepunkte wird dann automatisch der Oszillograph eingeblendet, mit dem das aufgezeichnete Impulsdiagramm genauer betrachtet werden kann. Weitere Informationen zur Handhabung des Oszillographen finden Sie im Kapitel 2.8 der Anleitung.
Für die obige Synchronzählerschaltung sieht das Impulsdiagramm bei 8 Taktimpulsen folgendermaßen aus:

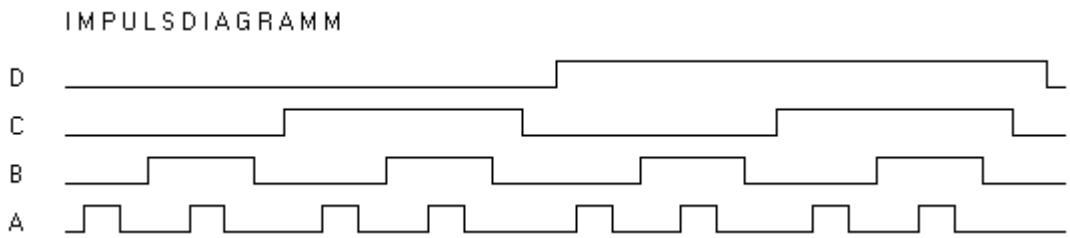


Abb. 3.3-6 Impulsdiagramm mit 8 Taktimpulsen

Mit Hilfe von Analysepunkten und einem Binärzähler, der automatisch alle möglichen Eingangskombinationen eines unbekannten Schaltnetzes erzeugen kann, können Sie sich einen genauen Überblick verschaffen. Hierzu müssen Sie nur einen Binärzähler mit so vielen Flipflops in die Schaltung einsetzen, wie Eingänge vorhanden sind und die Analysepunkte passend setzen. Die folgende Abbildung zeigt ein Beispiel.

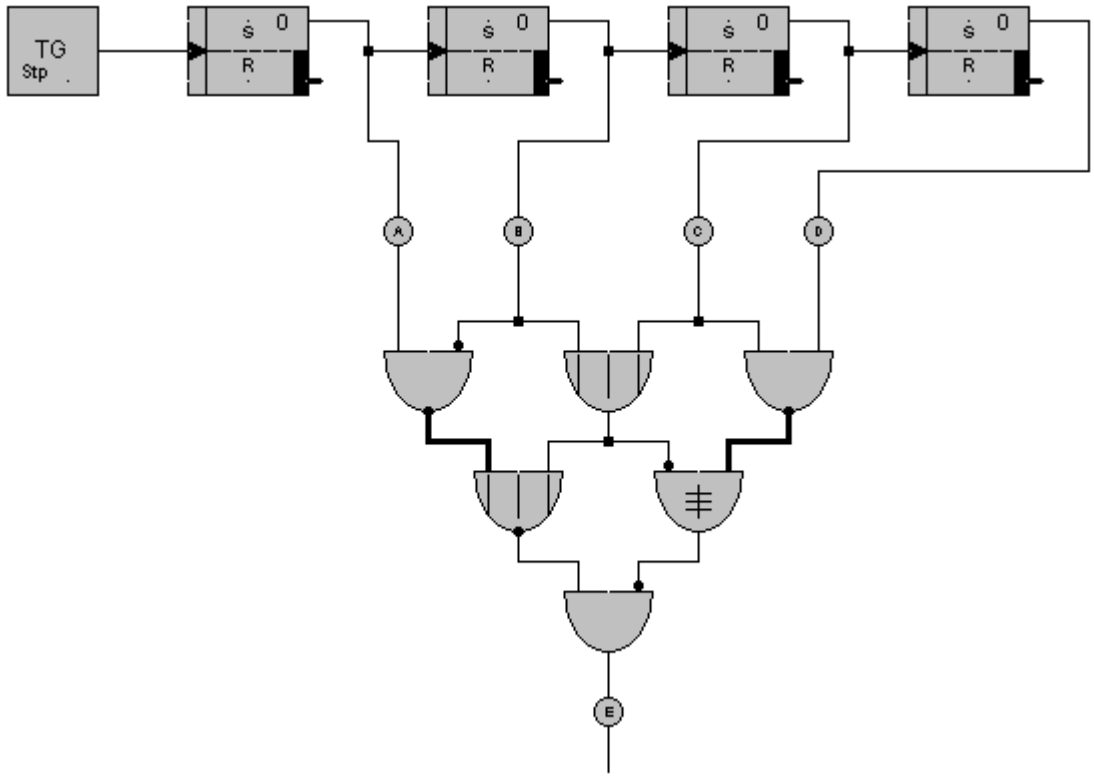


Abb. 3.3-7 Schaltnetz mit Binärzähler und Analysepunkten

Wenn Sie den Taktgeber 15 Takte lang laufen lassen, erhalten Sie das folgende Impulsdiadiagramm.

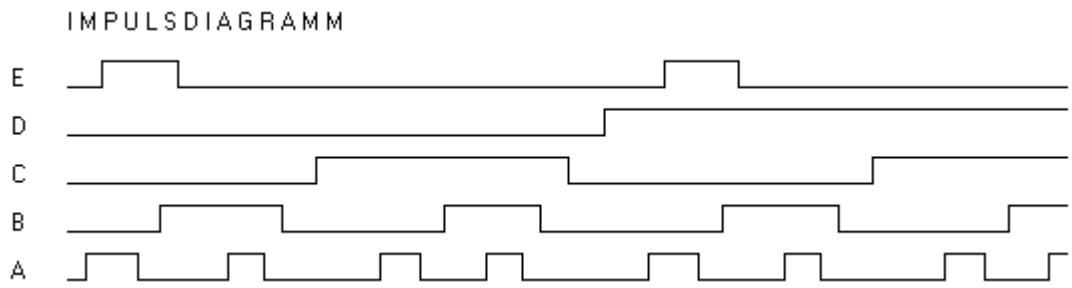


Abb. 3.3-8 Impulsdiagramm zur Schaltung in Abb. 3.3-7 mit 15 Taktimpulsen

Für die Anwendung von *LOCAD* im Zusammenhang mit dem Thema „Messen, Steuern und Regeln“ wird ein größerer Binärzähler benötigt, der auch noch zwischen Vor- und Rückwärtszählen umgeschaltet werden kann. Da der Aufbau eines solchen Zählers aus einzelnen Flipflops und Gattern etwas mühsam ist und viel Platz beansprucht, wurde ein kompakter Binärzählerbaustein vorgesehen.

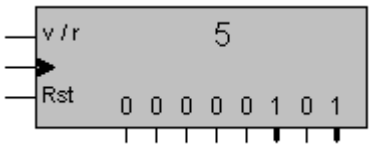


Abb. 3.3-9 8-Bit-Binärzähler

Dieser Zähler schaltet bei negativer Taktflanke am Takteingang weiter und gibt die entsprechende Bit-Kombination an den 8 Ausgängen ab. Mit einem 1-Signal am *Rst*-Eingang kann er auf Null gesetzt werden. Solange der *Rst*-Eingang auf 1 liegt, zählt der Zähler trotz eventuell eintreffender negativen Taktflanken nicht weiter.
Mit einem 1-Signal am v/r-Eingang kann er auf Rückwärtszählen eingestellt werden.

Mit einem Zähler ist es auch möglich, eine ganz bestimmte Anzahl von Takten aus einem dauerhaften Taktsignal auszukoppeln.

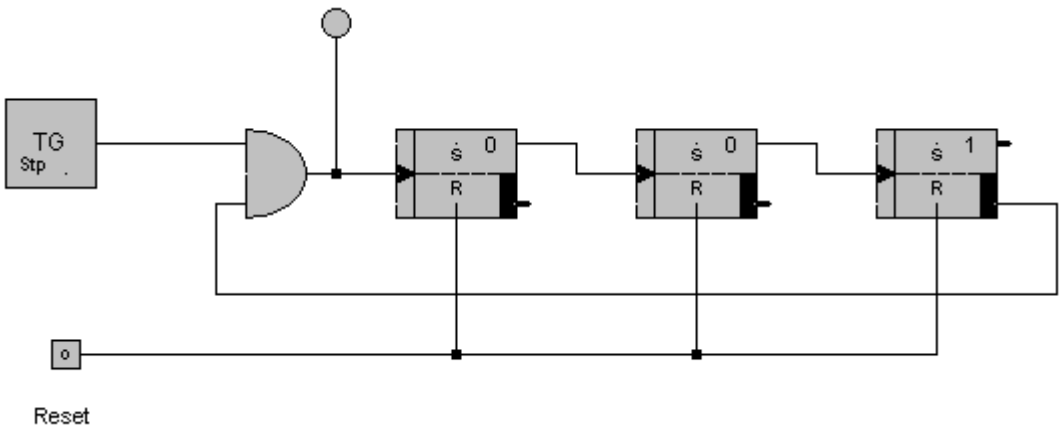


Abb. 3.3-5 Auskopplung von vier Takten aus einem Dauertaktsignal

Aufgabe 3.11: Entwickeln Sie eine asynchronen Rückwärtszähler.
(Lösung im Anhang C)

Aufgabe 3.12: Entwickeln Sie einen 4-Bit-Zähler, der sich automatisch nach jedem 12. Takt auf Null zurücksetzt.
(Lösung im Anhang C)

In der Digitaltechnik gibt es noch weitere Anwendungen für Zähler. Sie werden u.a. eingesetzt zur Impulszählung, digitalen Frequenzmessung, digitalen Zeitmessung und Frequenzteilung. Ein einziges Zählflipflop stellt einen 1:2-Frequenzteiler dar, denn mit jeder negativen Taktflanke wechselt der Ausgangszustand, so dass nach zwei Takten der Anfangszustand wieder eingenommen wird.

Die folgende Schaltung zeigt einen 1:3-Frequenzteiler.

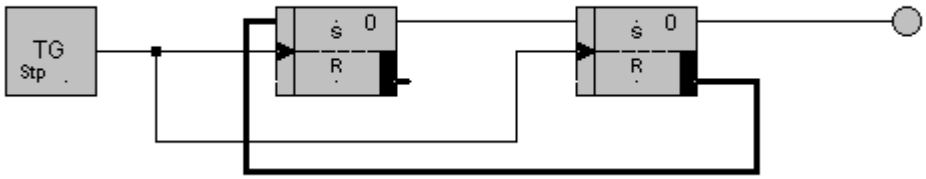


Abb. 3.3-6 1:3-Frequenzteiler

3.4 Autonome Schaltwerke

Ein Schaltwerk, das von außen nicht beeinflussbar ist, heißt ***autonom***. Es gibt synchrone und asynchrone autonome Schaltwerke. Bei den asynchronen werden die Ausgangssignale und die Schaltgeschwindigkeit durch den inneren Aufbau bestimmt. Für die Schaltgeschwindigkeit sind die Verzögerungszeiten der einzelnen Bauelemente verantwortlich. Bei den synchronen autonomen Schaltwerken bestimmt ein Taktsignal, wann Zustandswechsel stattfinden. Ein Taktsignal wird nicht als Eingangssignal angesehen, so dass hier keine Einschränkung der Autonomie vorliegt.

Autonome Schaltwerke haben vor allem dort eine besondere Bedeutung, wo gleichbleibende Folgen von Zuständen durchlaufen werden müssen, wie es z.B. in einem Mikroprogramm-Steuerwerk der Fall ist.

Ein synchrones autonomes Schaltwerk lässt sich mit Hilfe von Decodern, Flipflops, Taktgeber und Koppeldioden realisieren.

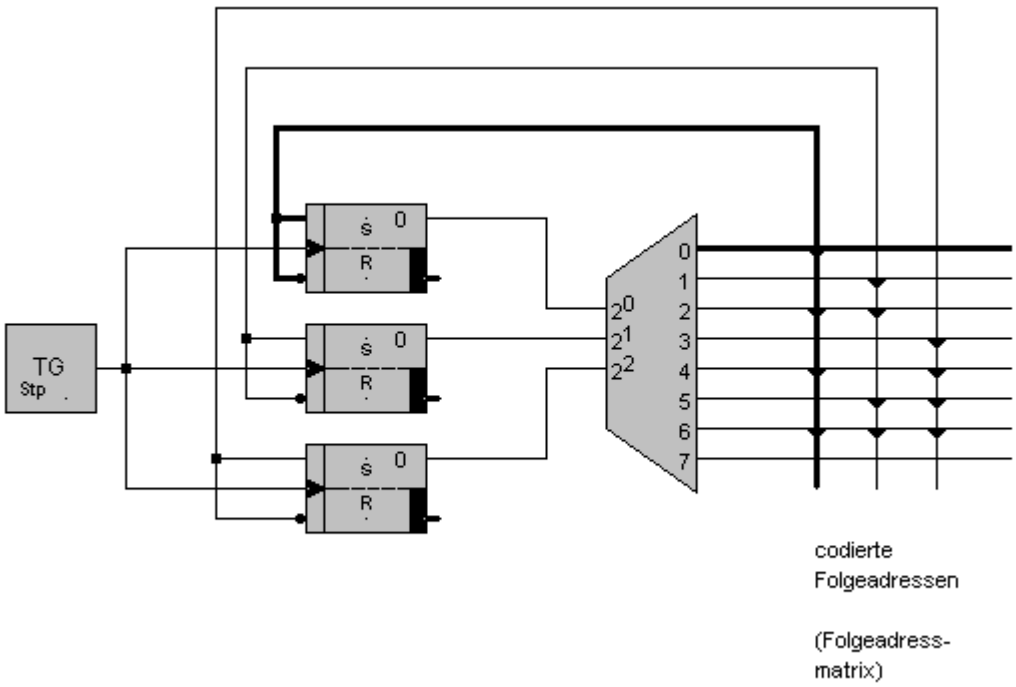


Abb. 3.4-1 Synchrones autonomes Schaltwerk

Ein Signal auf einer Ausgangsleitung überträgt sich auf die über Koppeldioden damit verbundenen Eingangsleitungen der Flipflops. Mit den Koppeldioden wird also zu jeder Ausgangsleitung eine neue Eingangskombination festgelegt, so dass sich ein eindeutiger Folgezustand einstellt. Das Schaltwerk durchläuft eine Menge möglicher Zustände in einer durch die Koppeldioden festgelegten Reihenfolge. Wenn das Schaltwerk n Zustandsvariablen hat, sind 2^n verschiedene Zustände möglich, wovon aber nicht alle durchlaufen werden müssen. Spätestens nach 2^n Schritten befindet sich das Schaltwerk allerdings in einem Zyklus. Das System der Koppeldioden zur Codierung der Folgeadressen bezeichnet man als **Folgeadressmatrix.**

Aufgabe 3.14: Bauen Sie das autonome Schaltwerk mit LOCAD auf und machen Sie sich die Funktionsweise klar, indem Sie Einzeltakte auslösen.

.

Aufgabe 3.15: Ändern Sie die Folgeadressmatrix so ab, dass das Schaltwerk die folgende Zustandsfolge durchläuft: 0 7 1 2 5 6 4 3 0

Um einen einfacheren Aufbau von autonomen Schaltwerken zu ermöglichen wurden in *LOCAD* die Flipflops und der Decoder zu einem Bauelement (Decoder mit Takteingang) zusammengefasst.

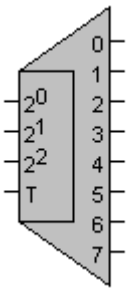


Abb. 3.4-2 Decoder mit integrierten Flipflops und Takteingang

3.4.1 Ampelsteuerung

Die Tatsache, dass das autonome Schaltwerk die vorgegebene Folge von Zuständen andauernd in gleicher Reihenfolge durchläuft, macht es für die Steuerung von starr ablaufenden Prozessen interessant.

An den Ausgangsleitungen können mit Koppeldioden weitere Leitungen zur Steuerung eines Vorganges, wie z.B. die Schaltung einer Ampel, angekoppelt werden.

Um eine Ampelsteuerung zu realisieren, braucht man nur die drei Lampen mit den Ausgangsleitungen des Schaltwerks so durch Koppeldioden zu verbinden, dass die gewünschte Folge von Lampen-Zuständen auftritt. Da nur vier Zustände zu durchlaufen sind (rot, rot-gelb, grün, gelb), muss die Folgeadressmatrix so geändert werden, dass auf den vierten Zustand (Ausgangsleitung 3) wieder Zustand 1 (Ausgangsleitung 0) folgt. Die folgende Abbildung zeigt die Schaltung.

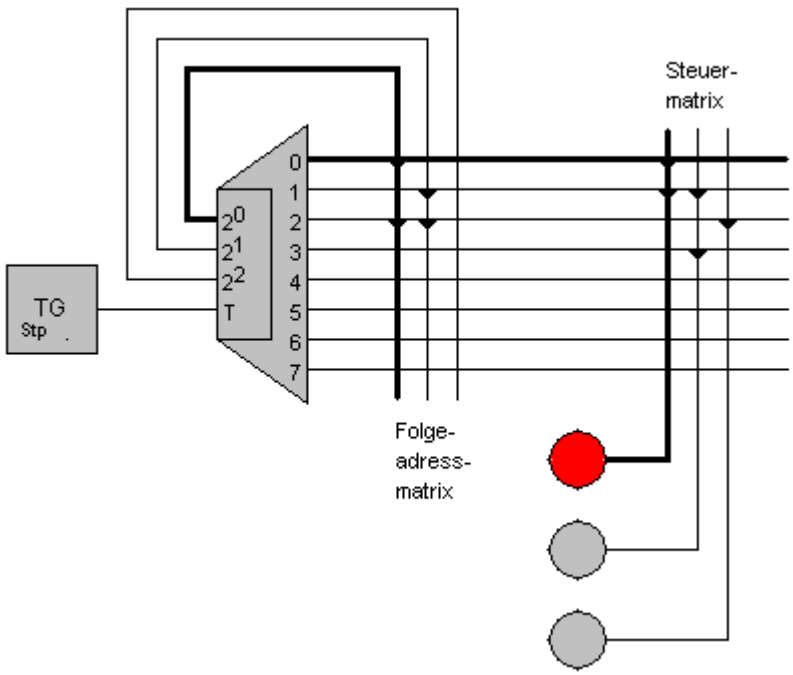


Abb. 3.4.1-1 Ampelsteuerung mit einem autonomen Schaltwerk

Das System der Koppeldioden zur Steuerung des Vorganges nennt man **Steuermatrix**.

Aufgabe 3.16: Bauen Sie die Schaltung auf und machen Sie sich die Funktionsweise klar.

Es ist Ihnen sicherlich aufgefallen, dass beim Umschalten größeren Pausen an der Ampel entstehen. Mit Hilfe von Flipflops, die die Lampenzustände zwischenspeichern, lässt sich dieser Effekt vermeiden.

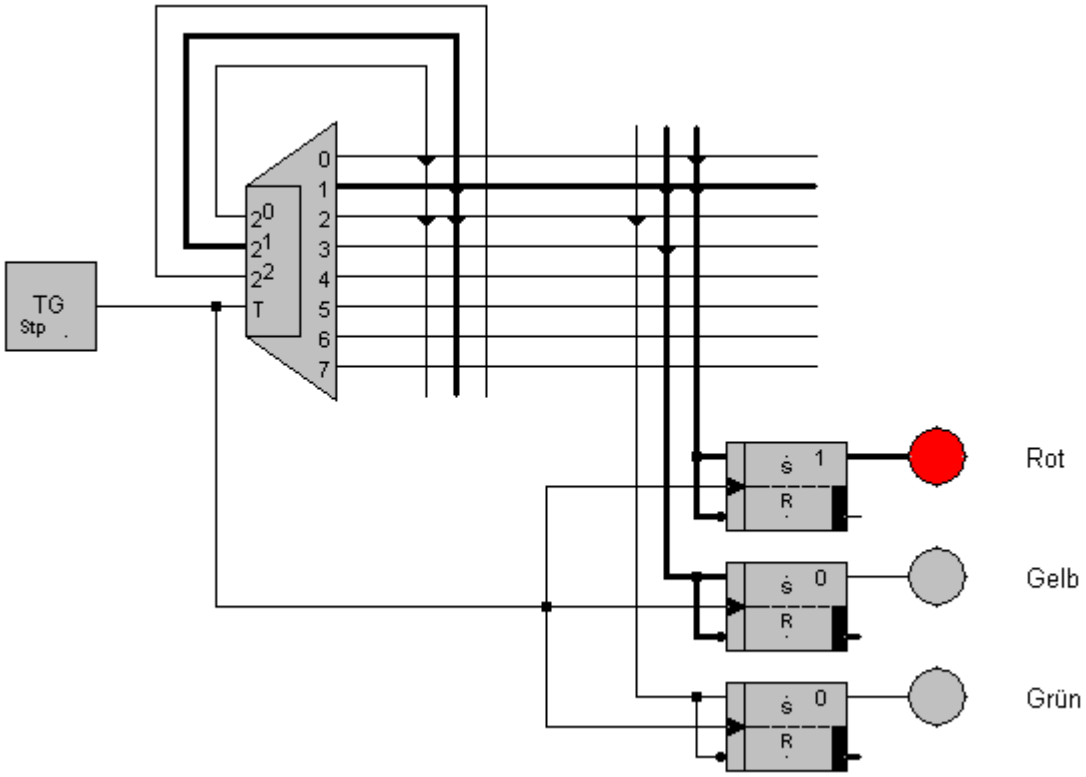


Abb. 3.4.1-2 Ampelsteuerung mit Zwischenspeicherflipflops

Aufgabe 3.17: Ändern Sie die Schaltung so ab, dass folgende Phasenlängen entstehen:

Rot-Phase	2 Takte
Rot-Gelb-Phase	1 Takt
Grün-Phase	3 Takte
Gelb-Phase	1 Takt

4 Programmsteuerung eines Rechners

4.1 Struktur einer DVA

Die folgende Abbildung zeigt die vereinfachte Grobstruktur einer Datenverarbeitungsanlage.

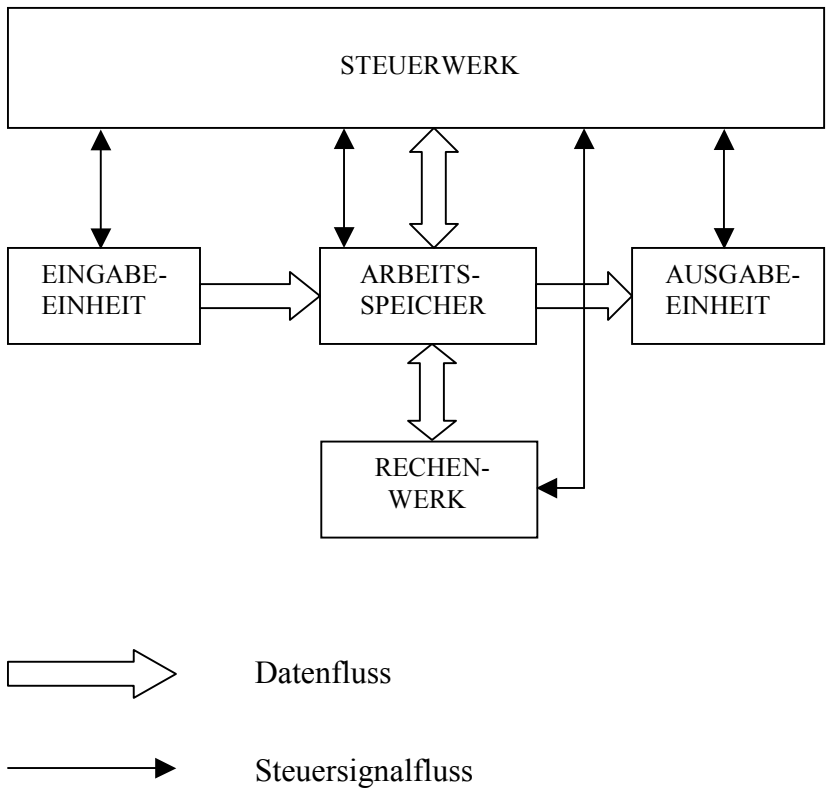


Abb. 4.1-1 Grobstruktur einer DVA

Rechenwerk, Arbeitsspeicher und Steuerwerk bilden zusammen eine Funktionseinheit, die üblicherweise als Zentraleinheit bezeichnet wird (engl. **C**entral **p**rocessing **u**nit, abgekürzt **CPU**)

4.1.1 Rechenwerk

Das Rechenwerk ist eine Funktionseinheit, die arithmetische (Addition, Subtraktion, Multiplikation, Division) und logische (UND-, ODER-, NICHT-Verknüpfung) Operationen ausführen kann. Es besteht aus einem oder mehreren Registern zur Zwischenspeicherung der Operanden und einer arithmetisch-logischen Einheit (ALU). Da alle arithmetischen Operationen letztlich auf die Addition zurückgeführt werden können, beinhaltet eine einfache ALU als grundlegende Verknüpfungselemente ein Addierwerk und einen Komplementierer.

Welche Operationen das Rechenwerk ausführen soll, wird vom Steuerwerk ermittelt und durch Steuersignale mitgeteilt.

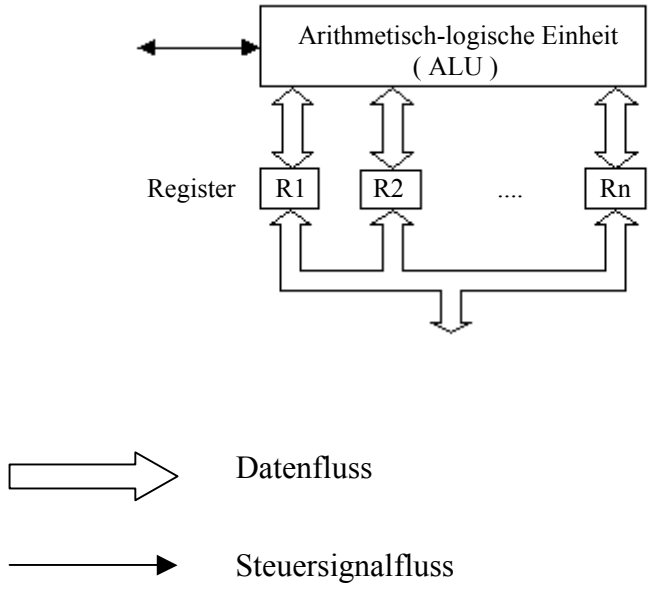


Abb. 4.1.1-1 Struktur des Rechenwerks

Mit den Registern und Rechenschaltungen im letzten Kapitel haben wir bereits die grundlegenden Teile eines Rechenwerks kennen gelernt.

4.1.2 Arbeitsspeicher

Ein programmgesteuerter Rechner kann eine vom Benutzer vorgegebene Folge von Befehlen selbstständig ausführen. Die Befehlsfolge und die zu bearbeitenden Daten müssen dazu in einem Speicher, dem sogenannten Arbeitsspeicher, direkt greifbar in dualer Darstellung zur Verfügung stehen.[1]

Ein Speicher ist also eine Funktionseinheit zum Aufbewahren von Daten und Befehlen. Die einzelnen Speicherelemente bestehen bei den sogenannten **statischen RAM**s (**R**andom **A**ccess **M**emory, Speicher mit wahlfreiem Zugriff, Schreib-Lese-Speicher) aus Flipflops, in denen die Information so lange gespeichert bleibt, bis das betreffende Flipflop angesprochen und mit einem neuen Wert belegt oder die Stromversorgung abgeschaltet wird. Ein Schreib-Lese-Speicher ermöglicht sowohl das Einschreiben als auch das Auslesen von Daten.

[1] Die in einer höheren Programmiersprache fast umgangssprachlich formulierten Anweisungen können mit einem Compiler in eine Befehlsfolge in dualer Darstellung übersetzt werden.

Bei den **dynamischen RAM**s werden anstatt Flipflops interne Kapazitäten benutzt. Die Information wird als Kondensatorladung gespeichert. Da hierbei sogenannte Leckströme auftreten, muss der Ladungsverlust in Abständen von einigen Millisekunden ausgeglichen werden (Refresh), was zusätzliche Hardware erforderlich macht. Trotzdem ist die Verwendung dynamischer RAMs vorteilhaft, da die Speicherzellen viel kleiner sind als bei statischen RAMs und daher die Kosten pro Bit niedriger liegen.

Mit den in *LOCAD* zur Verfügung stehenden Mitteln können wir nur ein Modell eines statischen RAMs aufbauen. Die wesentlichen Prinzipien kommen aber hierin zum Ausdruck.

Im allgemeinen ist nicht jedes Speicherelement (1 Flipflop ≙ 1 Bit) isoliert ansprechbar, sondern es werden mehrere Speicherelemente zu einer gemeinsam adressierbaren Gruppe zusammengefasst. Eine solche Gruppe bildet die kleinste adressierbare Einheit des Speichers und wird **Speicherzelle** genannt. In der Regel umfasst eine Speicherzelle 8 Bit (≙ 1 Byte).

Die in Kapitel 3 beschriebenen Schieberegister stellen im Prinzip 4-Bit-Speicherzellen dar. Da bei jeder negativen Taktflanke der Speicherinhalt um eine Stelle verschoben wird und die gespeicherten Information nach und nach am Ausgang des letzten Flipflops erscheinen, wird ein Schieberegister auch als **Serienspeicher** bezeichnet. Im Unterschied dazu erfolgt bei einem **Parallelspeicher** die Ein- und Ausgabe der Daten für alle Speicherelemente der Zelle gleichzeitig.

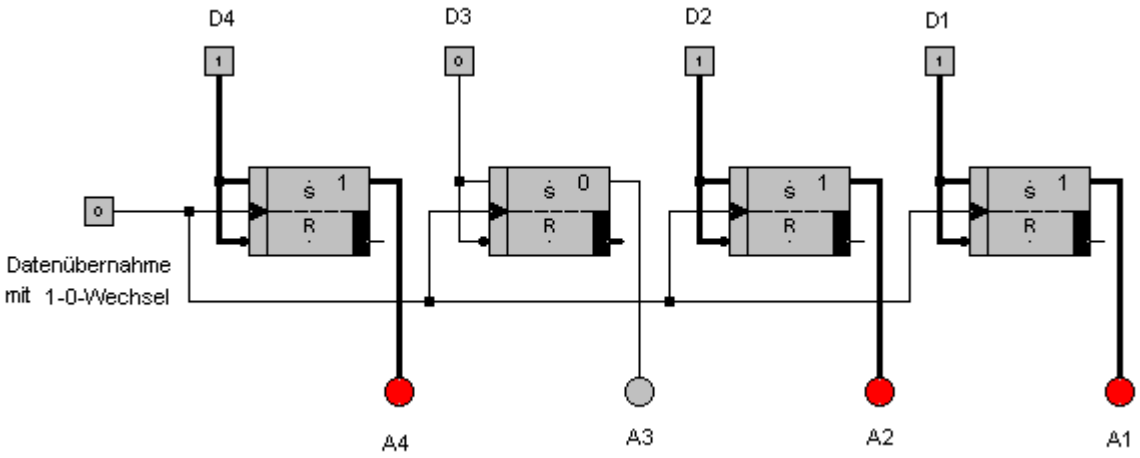


Abb. 4.1.2-1 Parallelspeicher für eine 4-Bit-Information

Um aus mehreren Speicherzellen eine ganz bestimmte zum Beschreiben oder Auslesen auszuwählen, sind sogenannte Adressleitungen notwendig. Die folgende Abbildung zeigt, wie mit einer Adressleitung die Datenein- und -ausgabe bei einem Element einer Speicherzelle freigegeben bzw. gesperrt werden kann.

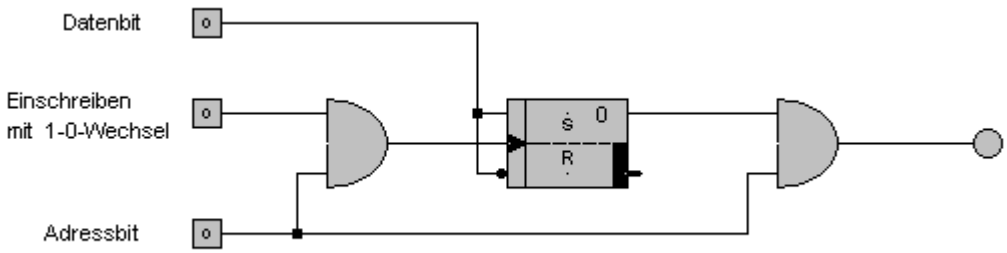


Abb. 4.1.2-2 Schreib-Lese-Schaltung für ein Speicherelement

Das Adressbit gibt die Dateneingabe und die Datenausgabe gleichzeitig frei.

Die folgende Schaltung ist ein Modell eines RAMs. Wegen der notwendigen Beschränkung auf eine Bildschirmseite wurden nur 4 Zellen mit jeweils 3 Bit aufgebaut.

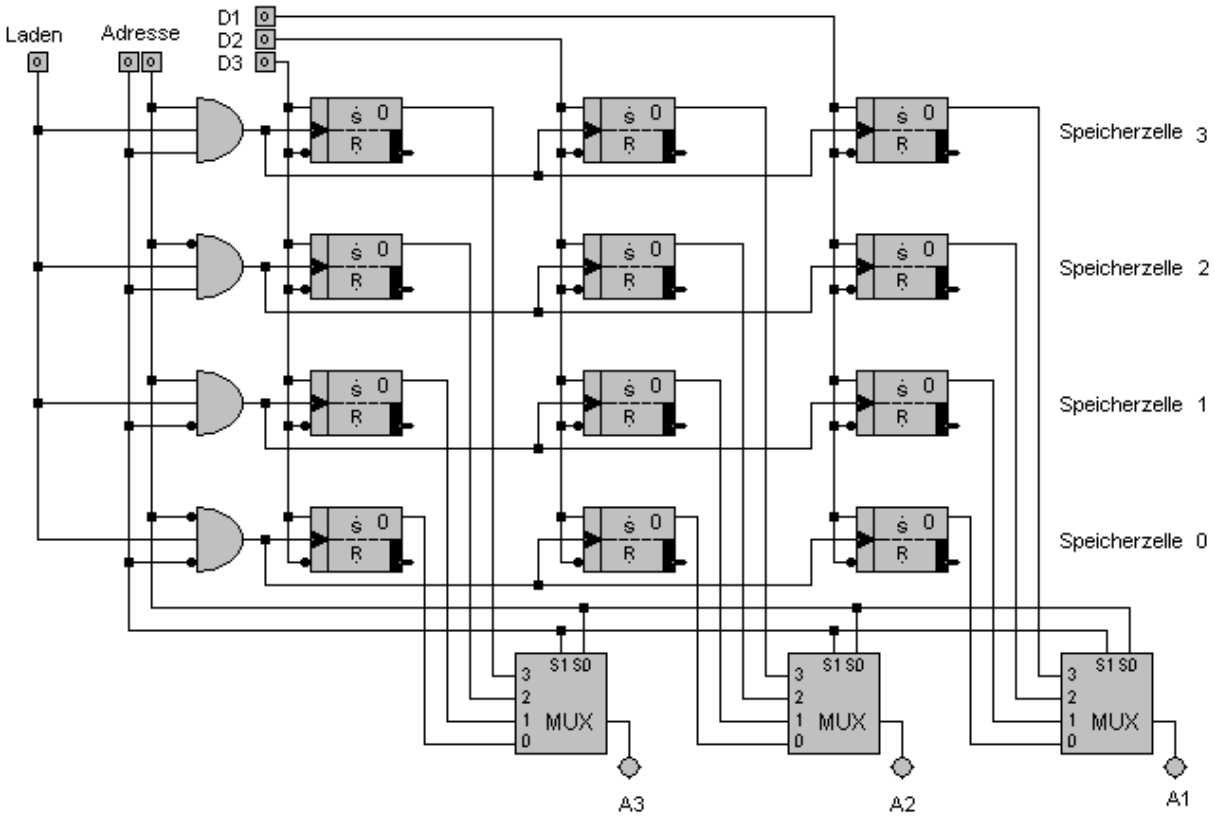


Abb. 4.1.2-3 4 x 3-Bit-Speicher

Jede Flipflop-Zeile stellt eine 3-Bit-**Speicherzelle** dar. Die Nummern der einzelnen Zellen nennt man **Adressen.**

Die an den Datenleitungen D1 bis D3 (Speichereingänge) anliegenden Signale werden bei einer negativen Signalflanke auf der Ladeleitung in eine Speicherzelle übernommen. Welche Zelle die Daten übernimmt, wird durch die 2-Bit-Adresse bestimmt. Bei jeder Adressbit-Kombination wird ein ganz bestimmtes Tor zu den Takteingängen der Flipflops geöffnet.

Die Adressbits steuern auch die Datenwege in den Multiplexern, so dass nur die Information der adressierten Speicherzelle auf die Speicherausgänge A1 bis A3 geschaltet wird.

Aufgabe 4.1: Laden Sie die Schaltung und machen Sie sich die Funktionsweise beim Einschreiben und Auslesen der Daten klar.

Es ist etwas aufwendig, sowohl auf der Eingangs- als auch auf der Ausgangsseite eine Adressdecodierung vorzunehmen. Mit sogenannten **Tri-State**-Gattern kann der Aufwand verringert werden. Ein Tri-State-Gatter ist im Prinzip ein elektronisches Tor, bei dem im Unterschied zu dem ebenfalls als Tor verwendbaren UND-Gatter der Ausgang nicht auf Nullpegel gehalten, sondern quasi abgetrennt (hochohmig) wird, wenn die Aktivierung abgeschaltet ist. Der hochohmige Zustand eines Tri-State-Gatters wird in *LOCAD* dadurch kenntlich gemacht, dass die Ausgangsleitung unterbrochen wird.

Die Verwendung von Tri-State-Gattern vereinfacht den Aufbau der Schaltung. Unter Ausnutzung dieser Tatsache lässt sich der 4 x 3-Bit-Speicher etwas übersichtlicher aufbauen.

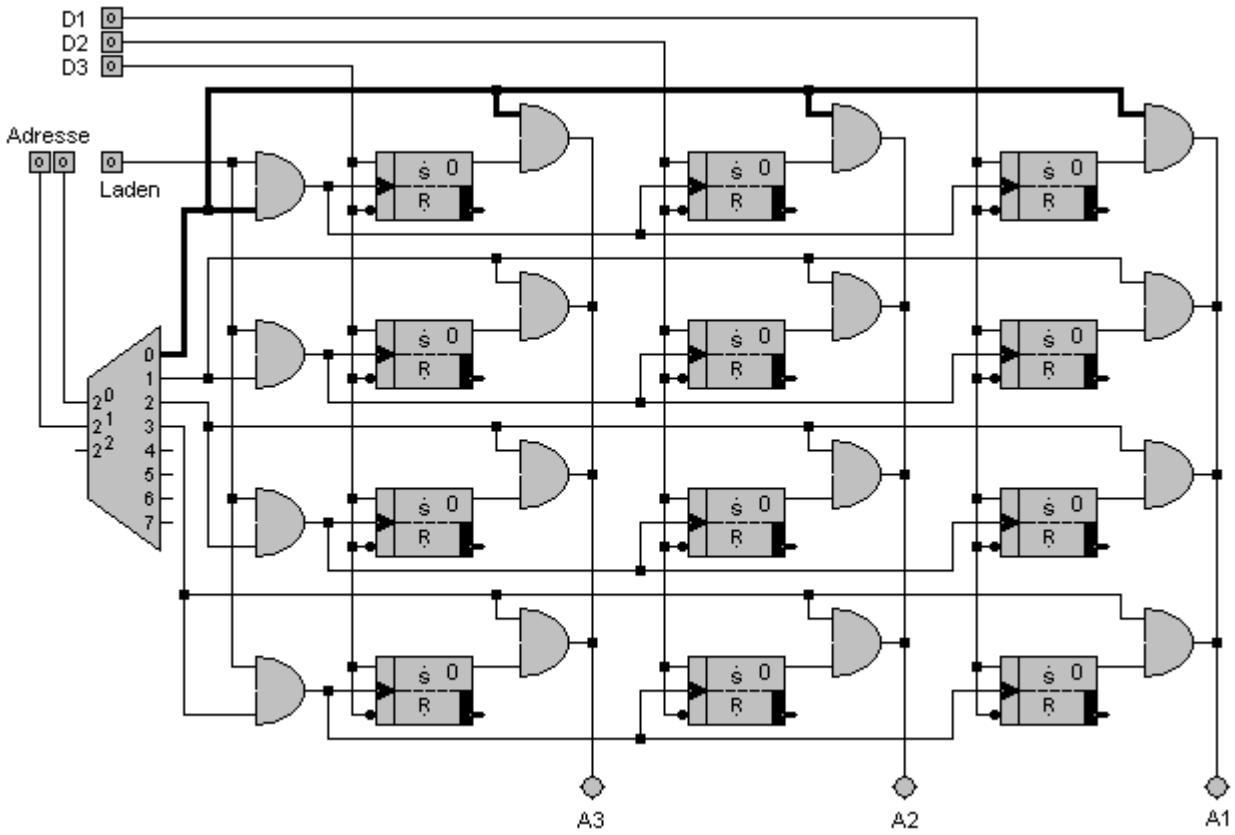
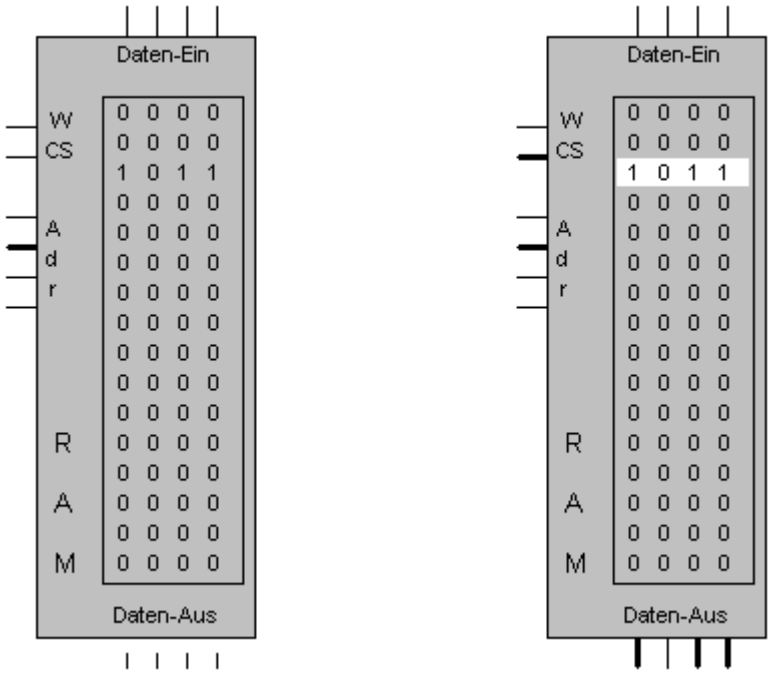


Abb. 4.1.2-3 4 x 3-Bit-Speicher unter Verwendung von Tri-State-Gattern

Die Adressierung der Flipflop-Zeilen wird hier über einen sogenannten **Adressdecoder** vorgenommen. Der Inhalt der adressierten Reihe wird über die Tri-State-Gatter auf die Ausgänge durchgeschaltet. Mit einer negativen Signalflanke auf der Ladeleitung wird die ans den Dateneingängen anliegende Information in die adressierte Flipflop-Zeile übernommen.

4.1.2.1 RAM-Baustein

Da der 4 x 3-Bit-Speicher bereits den größten Teil der Arbeitsfläche beansprucht wurde für komplexere Anwendungen ein kompakter RAM-Baustein mit 16 4-Bit-Speicherzellen vorgesehen.



a) b)

Abb. 4.1.2.1-1 RAM-Baustein a) nicht aktiviert (CS = 0) b) aktiviert (CS = 1)

Die 16 Speicherzellen sind intern von oben nach unten von 0 bis 15 durchnummeriert.
Der Baustein besitzt 4 Adresseingänge, wobei der oberste Eingang dem niederwertigsten Adressbit zugeordnet ist. Mit einer an den Adresseingängen anliegenden Bitkombination wird die Speicherzelle angesprochen, deren Nummer dieser Bitkombination entspricht.

Mit dem Auswahleingang CS (Chip Select) kann der Baustein ausgewählt bzw. deaktiviert werden. Wenn er mit einem 1-Signal am CS-Eingang ausgewählt ist, wird die momentan adressierte Speicherzelle zur besseren Übersicht unterlegt dargestellt (siehe Abbildung b). In diesem Fall kann auch die an den Dateneingängen anliegende Information mit einem 1-0-Signalwechsel am Eingang W (Write) in die adressierte Zelle eingeschrieben werden. Das geht nicht, wenn der Baustein nicht ausgewählt ist (CS = 0).

Die Ausgänge sind intern mit Tri-State-Gattern versehen, so dass sie quasi abgetrennt (hochohmig) werden, wenn der Baustein nicht ausgewählt ist (siehe Abbildung a).

Nach der Aktivierung einer Schaltung können die Bits der einzelnen Speicherzellen durch Anklicken mit der linken Maustaste in den jeweils anderen Wert geändert werden. Dies funktioniert auch dann, wenn der Baustein nicht mit einem 1-Signal am CS-Eingang ausgewählt ist.

4.1.3 Steuerwerk

Das Steuerwerk ist die wichtigste Funktionseinheit einer Datenverarbeitungsanlage („Herz des Computers“). Es hat die Aufgabe, die Programmbefehle in der richtigen Reihenfolge aus dem Arbeitsspeicher zu holen, sie zu interpretieren und die notwendigen **Steuersignale** zu erzeugen, mit denen dann die an der Ausführung des Befehls beteiligten Einheiten versorgt werfen müssen, damit diese entsprechende Aktionen ausführen.

Zur Steuerung der Befehlsabarbeitung muss das Steuerwerk zwei wichtige Informationen festhalten:

- den Befehl, der gerade in Bearbeitung ist
- die Adresse des nächsten auszuführenden Befehls

Diese Informationen werden in zwei besonderen Registern aufbewahrt, dem **Befehlsregister** (**IR**, **i**nstruction **r**egister) und dem **Befehlszählregister** [1] (**PC**, **p**rogram **c**ounter).

Die folgende Abbildung zeigt den prinzipiellen Aufbau des Steuerwerks.

[1] Hier wäre die Bezeichnung „Befehlsadressregister“ besser.

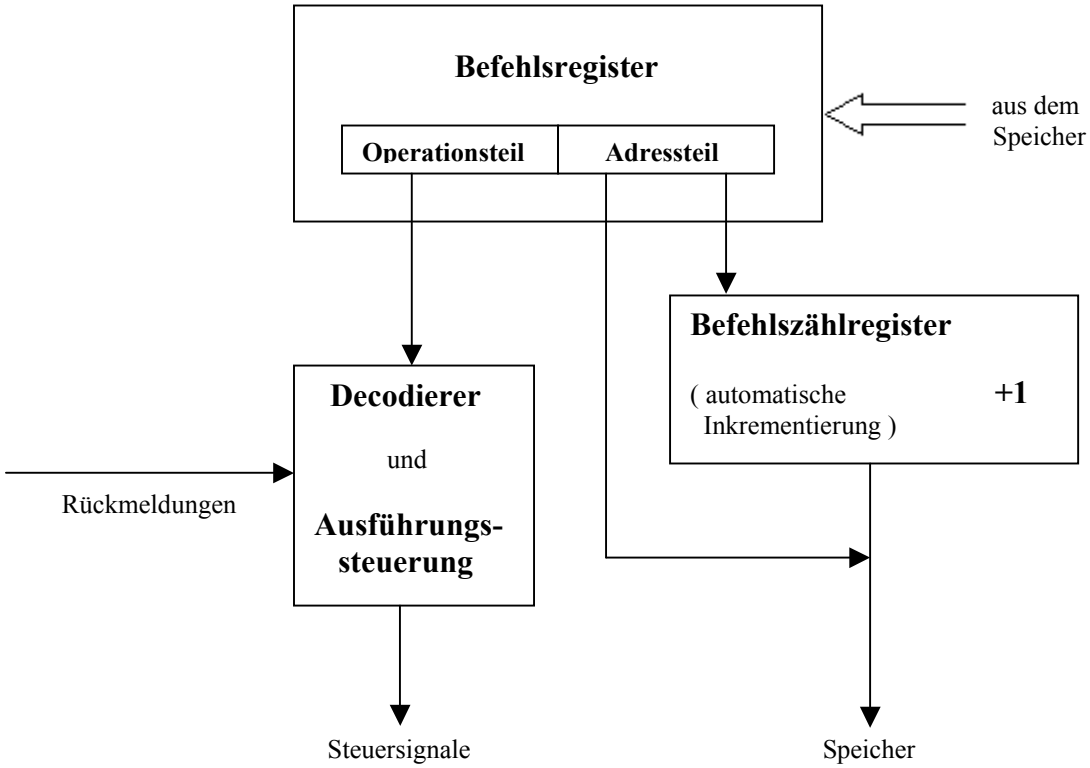


Abb. 4.1.3-1 Aufbau des Steuerwerks

Ein Befehl besteht im allgemeinen aus einem **Operationsteil** und einem **Adressteil**. Der Operationsteil gibt an, **was** getan werden soll, der Adressteil, **wo** die Daten zu finden sind bzw. **wo** sie gespeichert werden sollen.

Das Befehlszählregister hält die Adresse des nächsten auszuführenden Befehls fest. Bei linearen Programmen steht der jeweils nächste Befehl in der auf den momentanen Befehls folgenden Speicherzelle, so dass der Inhalt des Befehlszählregisters meistens nur um 1 erhöht werden muss, da alle Programms streckenweise linear sind. Um eine automatische Inkrementierung zu ermöglichen, kann das Befehlstählregister als Dualzähler aufgebaut werden, der während jeder Befehlsausführung um 1 weiterzählt. Liegt allerdings ein Sprungbefehl vor, so wird eine im Adressteil des Befehls stehende Adresse in das Befehlszählregister übertragen.

Die Abarbeitung eines Befehls erfolgt in zwei Phasen:

1. Phase: (**Holen**) Die Adresse des nächsten Befehls steht im Befehlszählregister. Der Inhalt der damit adressierten Speicherzelle wird in das Befehlszählregister übertragen.

2. Phase: (**Ausführen**) Der Operationsteil des Befehls wird entschlüsselt und die entsprechenden Steuersignale werden erzeugt und weitergeleitet.
Die angesprochenen Funktionseinheiten des Systems reagieren auf die Steuersignale mit ganz bestimmten Aktionen.

Abschließend ist nur noch zu klären, wie die Steuersignale erzeugt und in der richtigen Reihenfolge ausgegeben werden können.

4.2 Steuersignalerzeugung

Wir betrachten exemplarisch den folgenden (modifizierten) Assemblerbefehl:

```
MOV  Ziel, Quelle
```

wobei *Ziel* und *Quelle* zwei Speicherzellen bezeichnen.
Dieser Befehl bewirkt bei seiner Abarbeitung, dass die in der Speicherzelle *Quelle* stehende Information in die Speicherzelle *Ziel* kopiert wird.

Um diesen Kopiervorgang auszuführen, muss eine ganze Reihe von Steuersignalen erzeugt werden, und zwar Signale zur

- Übernahme der Adressinformationen
- Zwischenspeicherung der Daten
- Öffnung und Schließung von Toren für den Datenfluss
- Übernahme der Daten in die entsprechende Speicherzelle
- Rückmeldung nach Abschluss aller Aktionen.

Wir wollen uns an einem Modell eines Speichers klarmachen, welche Signale in welcher Reihenfolge erzeugt und auf die einzelnen Leitungen gelegt werden müssen, um eine solche Datenübertragung von einer Zelle in eine andere durchzuführen.

4.2.1 Handsteuerung

Um die Übertragung im Modell zu realisieren, müssen wir uns aus Platzgründen auf drei Speicherzellen und hierbei wiederum auf jeweils ein Bit beschränken.
Das ist keine wesentliche Einschränkung, da der Vorgang für alle Bits einer Speicherzelle in der gleichen Weise abläuft. Was wir uns für jeweils ein Bit klarmachen gilt gleichermaßen für alle anderen der jeweiligen Zelle. In unserem Modell existiert damit auch nur ein 1-Bit-Datenbus. Als Datenbus bezeichnet man eine Gruppe von mehreren Leitungen, über die der Transport der Daten zwischen den Funktionseinheiten ermöglicht wird, indem alle Einheiten an diese Übertragungsleitung angeschlossen werden.

Die folgende Abbildung zeigt das Modell zur Erarbeitung der Steuersignalfolge.

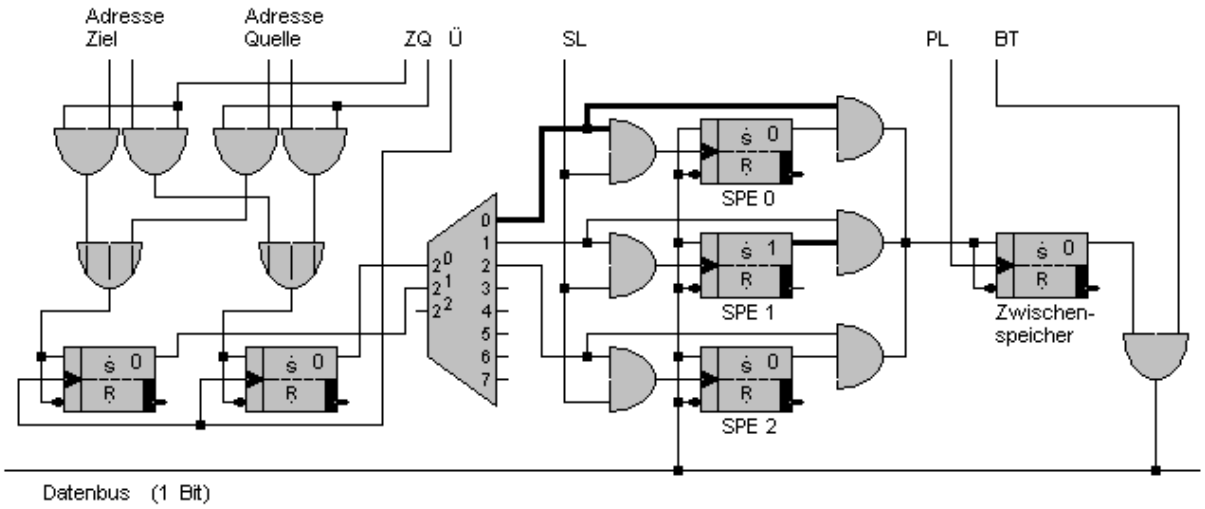


Abb. 4.2.1-1 Speichermodell (pro Speicherzelle nur 1 Bit)

Die drei Speicherelemente sind von oben nach unten durchnummeriert: SPE 0, SPE 1 und SPE 2 (siehe auch Decoderausgänge). Die beiden linken Flipflops bilden das sogenannte **Adressregister**. Hier werden die 2-Bit-Adressen zur Adressierung der Speicherelemente festgehalten.

Da das Modell nur drei Speicherelemente umfasst, macht es keinen Sinn, die Bitkombination 11 in das Adressregister zu laden, da dadurch kein Speicherelement angesprochen würde.

Die Kombinationen aus UND- und ODER-Gliedern oberhalb des Adressregisters ermöglichen die Übernahme einer Adresse (Ziel oder Quelle) in das Adressregister. Hierzu muss das jeweilige Tor geöffnet und anschließend ein Übernahmesignal (1-0-Wechsel) auf der Leitung *Ue* erzeugt werden.

Der Decoder decodiert die im Adressregister stehende Bitkombination und erzeugt auf genau einer Ausgangsleitung ein 1-Signal. Hiermit wird dann ein bestimmtes Speicherelement zum Lesen bzw. Beschreiben freigegeben. In der Abbildung ist durch den 00-Inhalt des Adressregisters das Speicherelement 0 adressiert.

Das Zwischenspeicherflipflop auf der rechten Seite kann die von einem Speicherelement kommende Information aufnehmen und über einen längeren Zeitraum speichern.

Im Speicherelement 2 ist eine 1 gespeichert. Wir wollen jetzt diese Information in das Speicherelement 1 kopieren. Die Adresse der Quelle ist somit $2_{10} = 10_2$ und die Zieladresse $1_{10} = 01_2$.

Aufgabe 4.2: Laden Sie die Schaltung, damit Sie die Auswirkungen der einzelnen Steuersignale direkt verfolgen können. Beachten Sie, dass die Adressen von Ziel und Quelle in der Schaltung bereits fest eingestellt sind.

Zuerst müssen wir uns jetzt klarmachen, welche Steuersignale in welcher Reihenfolge den Kopiervorgang ausführen.

1. Die Quelle muss adressiert werden, d.h. die Speicherzelle, aus der die Information zum Kopieren entnommen werden soll, muss erst einmal zum Auslesen angesprochen werden. Wir erreichen das durch ein 1-Signal auf Leitung *Q*. Hierdurch wird das Tor für die Adresse der Quelle geöffnet und die entsprechende Bitkombination gelangt an die Eingänge der Adressregister-Flipflops.

2. Ein 1-0-Wechsel auf der Übernahmeleitung *Ue* sorgt dafür, dass die an den Eingängen anliegende Information in die Flipflops übernommen wird.

3. Das Toröffnungssignal für die Quelladresse kann jetzt wieder auf 0 zurückgesetzt werden.
   Über den Decoder wird durch die Bitkombination im Adressregister das entsprechende Speicherelement zum Lesen freigegeben. Die Information gelangt über das geöffnete Tor am Ausgang auf den Eingang des Zwischenspeichers.

4. Durch einen 1-0-Wechsel auf der Leitung *PL* (Puffer laden) übernimmt der Zwischenspeicher diese Information und hält sie vorläufig fest.

5. Mit einem 1-Signal auf der Leitung *Z* muss jetzt das Tor für die Zieladresse geöffnet werden. Die Zieladresse gelangt damit an die Adressregistereingänge.

6. Ein 1-0-Wechsel auf der Übernahmeleitung *Ue* wird sie in das Adressregister eingeschrieben.

7. Das Toröffnungssignal für die Zieladresse kann jetzt wieder auf 0 zurückgesetzt werden.
   Über den Decoder wird durch die Bitkombination im Adressregister das entsprechende Speicherelement zum Beschreiben freigegeben.

8. Mit einem 1-Signal auf der Leitung *BT* (Bus-Tor) wird das Tor zwischen Zwischenspeicher und Bus geöffnet. Die zwischengespeicherte Information gelangt damit auf den Datenbus. Da die Eingänge aller Speicherelemente mit dem Datenbus verbunden sind, wird die auf dem Datenbus vorliegende Information allen Speicherelementen zur Übernahme angeboten. Weil aber durch die im Adressregister stehende Adresse ein ganz bestimmtes Speicherelement zum Beschreiben freigegeben ist, kann auch nur dieses eine Element die Information übernehmen.

9. Mit einem 1-0-Wechsel auf der Leitung *SL* (Speicher laden) wird die angebotene Information vom adressierten Speicherelement übernommen.

10. Mit einem 0-Signal auf der Leitung *BT* wird das Tor zum Bus wieder geschlossen.

Damit die der Kopiervorgang abgeschlossen und der Datenbus ist wieder frei.

Aufgabe 4.3: Führen Sie diesen Ablauf mit der Schaltung durch und beobachten Sie die Zustände, die sich durch die einzelnen Steuersignale einstellen.
Kopieren Sie ebenso den Inhalt des Speicherelementes 0 in das Speicherelement 3.

Die Beispiele zeigen, dass zur Ausführung eines Assemblerbefehls (**Makrobefehl**) u.U. viele elementare Operationen notwendig sind. Eine Elementaroperation wird **Mikrobefehl** genannt. Eine Folge von Mikrobefehlen, die die Ausführung eines Makrobefehls bewirkt, wird als **Mikroprogramm** bezeichnet.
Zu jedem Makrobefehl muss ein Mikroprogramm existieren, das die Ausführung des Makrobefehls ermöglicht. Jeder einzelne Mikrobefehl legt die Aktionen auf der untersten Ebene der Hardware fest.

Jetzt ist nur noch die Frage nach einer automatischen Erzeugung der Steuersignale und einem Mechanismus, mit dem zu jedem Makrobefehl die zugehörige Folge von Mikrobefehlen ausgewählt werden kann, zu beantworten.

4.2.2 Mikroprogramm-Steuerwerk

Wenn Sie an die Ampelsteuerung zurückdenken, werden Sie sicherlich sofort erkennen, dass eine automatische Erzeugung der Signale mit Hilfe eines Schaltwerkes und einer entsprechenden Steuermatrix realisiert werden kann.

Wir müssen nur dafür sorgen, dass das Schaltwerk mit einem Signal gestartet werden kann und dann eine ganz bestimmte Anzahl von Zuständen durchläuft, wobei es über die Steuermatrix in richtiger Reihenfolge die entsprechenden Signale abgibt. Weiterhin muss es nach Abarbeitung des Mikroprogramms anhalten und ein Rückmeldesignal an das übergeordnete Steuerwerk liefern, woran dieses erkennt, dass der nächste Makrobefehl ausgeführt werden kann.

Die folgende Abbildung zeigt eine Realisierungsmöglichkeit.

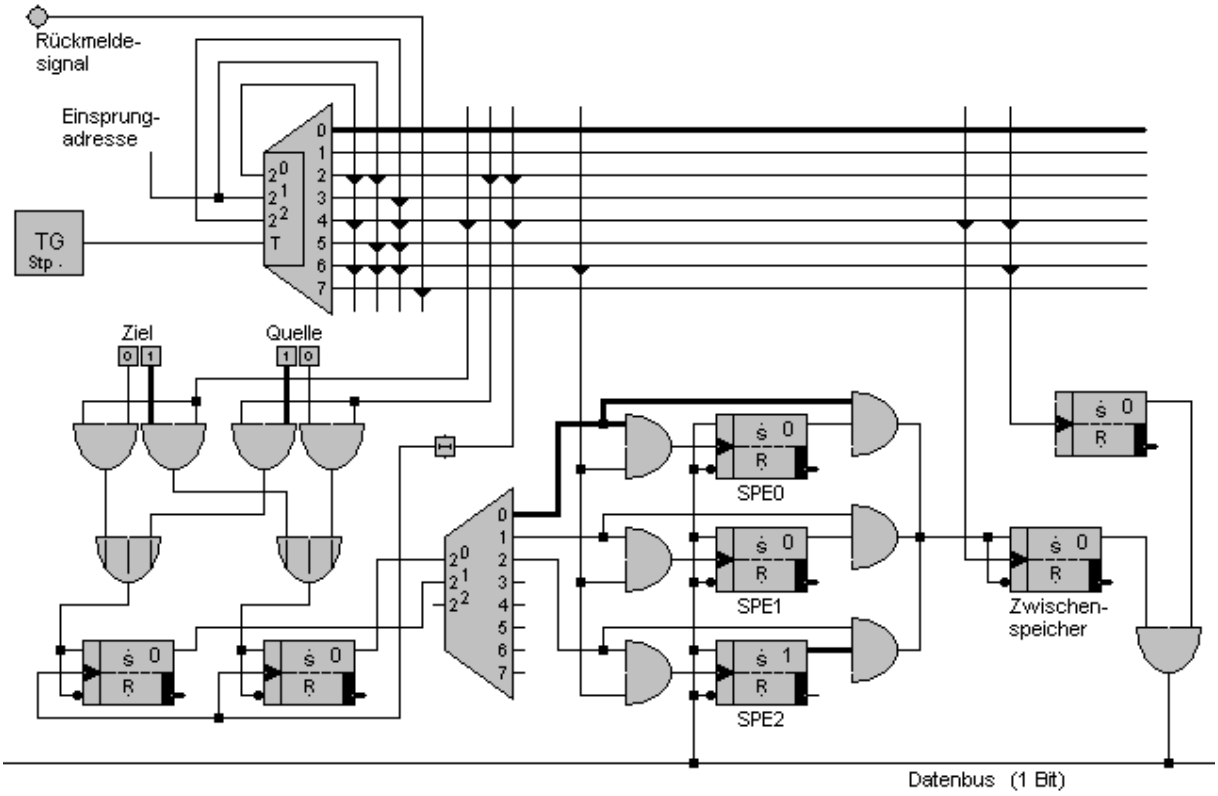


Abb. 4.2.2-1 Steuersignalerzeugung mit einem Mikroprogramm-Schaltwerk

Im Unterschied zu Abb. 4.2.1-1 wurde hier ein zusätzliches Flipflop eingesetzt, das das Toröffnungssignal für das Bus-Tor einen Takt lang konstant hält. Weiterhin wurde ein Verzögerungsglied in die Taktleitung zu den Adressregister-Flipflops eingesetzt, damit das Taktsignal erst dann ankommt, wenn die Informationen an den Adressregister-Eingängen stabil anstehen.

Als größte Erweiterung wurde das Schaltwerk hinzugefügt und die Steuerleitungen über Koppeldioden damit verbunden. Das Mikroprogramm ist durch die Koppeldioden festgelegt. Da es erst bei der Decoderausgangsleitung 2 beginnt, muss mit einem gesonderten Signal dafür gesorgt werden, dass das Schaltwerk in diesen Zustand kommt (Einsprungadresse). Dies wird durch ein 1-Signal auf der Leitung zum 2^1-Eingang des Decoders erreicht.
Wenn wir den Taktgeber auf 6 Taktimpulse einstellen (Taktfolge im Menü *Betrieb*) und starten, wird die Übertragung ausgeführt und das System stoppt mit einem 1-Signal auf der Rückmeldeleitung, um damit einem übergeordneten Steuerwerk die vollständige Abarbeitung des Mikroprogramms zu signalisieren.

Wichtiger Hinweis: Achten Sie bitte beim Testen der Schaltung darauf, dass Sie nicht die Bitkombination 11 als Adresse anlegen, da es kein Speicherelement mit der Nummer 3 in der Schaltung gibt.

Die folgende Abbildung in Anlehnung an Bild 3.5-1 aus [12] zeigt, wie über eine Mikroprogrammadresse zu jedem Makrobefehl eine Folge von Mikrobefehlen in einem Mikroprogrammspeicher aktiviert werden kann.

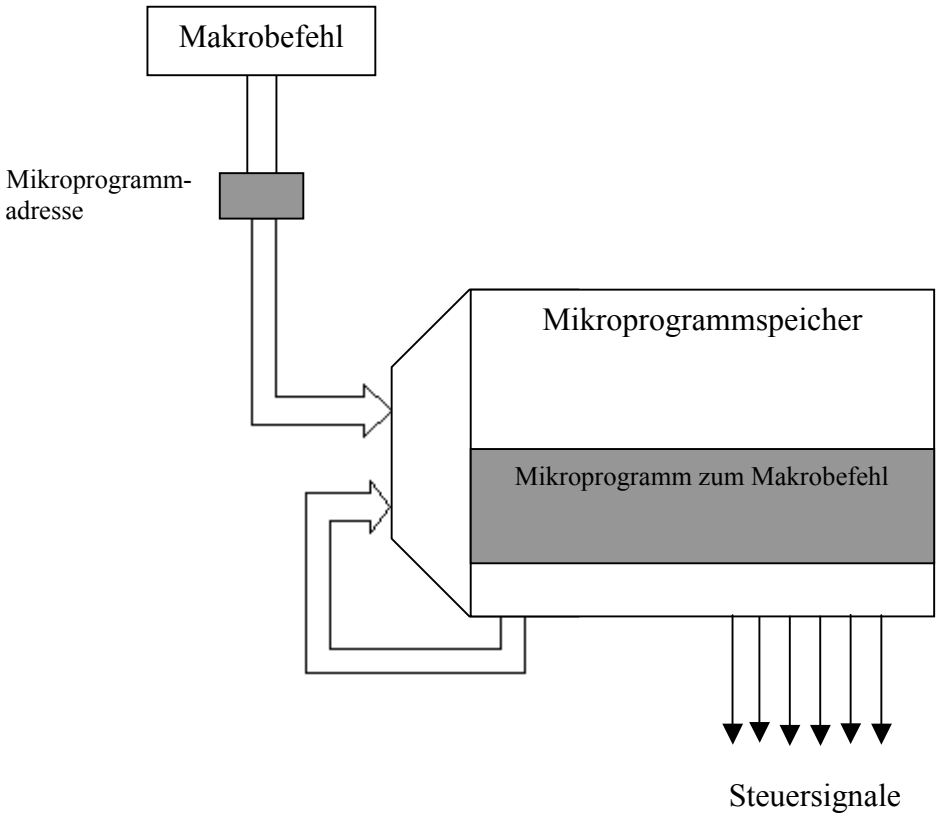


Abb. 4.2-1 Aktivierung eines Mikroprogramms zu einem Makrobefehl

Wenn Sie sich die Festwertspeicher aus Kapitel 2 in Erinnerung rufen, wird klar, dass das Koppeldiodensystem zur Bildung der Folgeadress- und Steuermatrix durch einen Festwertspeicher realisiert werden kann. Damit ist es dann möglich, durch Austausch des Mikroprogrammspeichers den Befehlssatz eines Rechners zu ändern bzw. anderen Anforderungen anzupassen, ohne die weitere Hardware zu verändern.

4.3 Programmsteuerung

In diesem Kapitel soll nun an einem Modellrechner gezeigt werden, wie die Abarbeitung eines Programms abläuft.
Bei der Abarbeitung eines im Arbeitsspeicher abgelegten Programms werden die einzelnen Befehle nach und nach aus dem Speicher geholt, entschlüsselt und jeweils entsprechende Aktionen durchgeführt.

Hier soll nun die Zentraleinheit eines Modell-Computers zur Verdeutlichung wichtiger Prinzipien der Programmsteuerung in Anlehnung an das Modell aus [2, Seite 93] entwickelt werden. Zur Vereinfachung des Modells werden nur Befehle mit impliziter Adressierung verwendet, d.h. der Operationsteil enthält bereits Informationen über die zu verwendenden Register, ein Adressteil ist also nicht notwendig.

In einer Vorstufe wird zuerst gezeigt, wie Abläufe mit einem Befehl gesteuert werden können.

4.3.1 Ablaufsteuerung mit einem 1-Bit-Befehl

Die nachstehend abgebildete Schaltung enthält nur ein Eingaberegister, ein Speicherregister und einen Akkumulator. Mit dem Befehl "*addiere*" soll der Inhalt des Eingaberegisters zum Akkumulatorinhalt addiert werden. Der Befehl "*speichere*" soll dazu führen, dass der Inhalt des Akkumulators in das Speicherregister übertragen wird.

Erreichen lässt sich das, indem man dafür sorgt, dass die Taktsignale im Fall des Addierens nur das Eingaberegister, den Akkumulator und das Flipflop zur Übertragsspeicherung erreichen und im Falle des Speicherns nur zum Akkumulator und zum Speicherregister gelangen.
Zur Unterscheidung der beiden möglichen Befehle wird nur eine 1-Bit-Speicherstelle als Befehlsregister benötigt. Wie die einzelnen Befehle im Befehlsregister codiert werden, kann willkürlich festgelegt werden. Wir wollen hier vereinbaren, dass der Befehl "*speichere*" durch eine 1 und der Befehl "*addiere*" durch eine 0 codiert wird.

Die Taktsignale werden über zwei als elektronische Tore wirkende UND-Gatter geleitet. Durch die spezielle Verbindung der UND-Gatter mit dem Befehlsregister wird eine dem Befehl entsprechende Verteilung der Taktsignale erreicht. Da der Befehl hierdurch decodiert wird, bezeichnet man diesen Teil der Schaltung als **Befehlsdecoder**. Der Inhalt des Befehlsregisters legt also fest, welche Teile des Modells die Taktsignale erhalten.

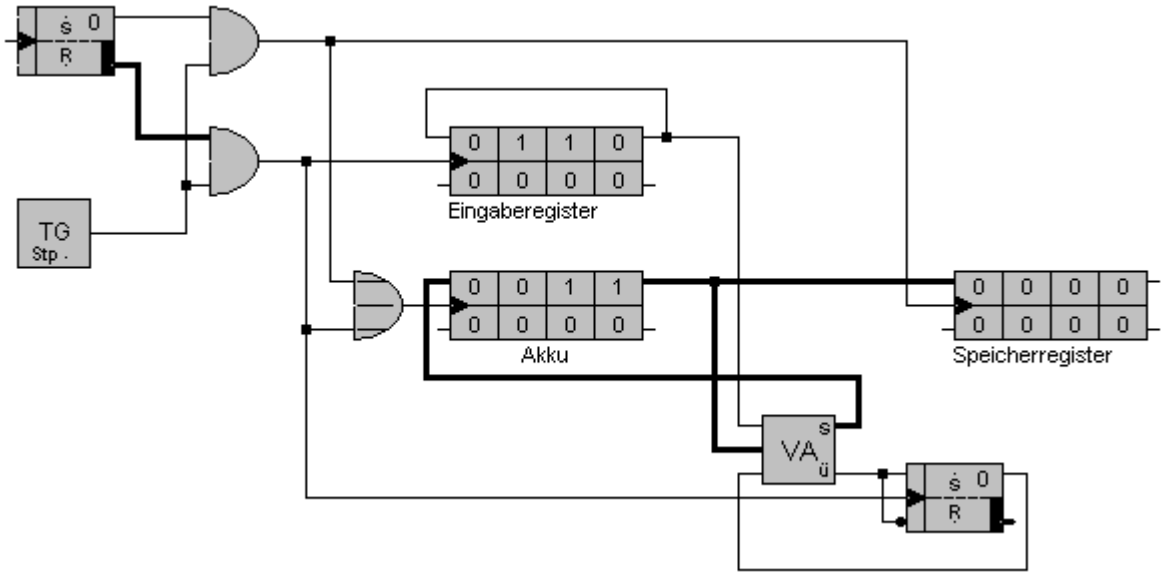


Abb. 4.3.1-1 Ablaufsteuerung

Nehmen Sie die Schaltung zum Testen in Betrieb und setzen Sie durch Anklicken mit der linken Maustaste eine Dualzahl in das Eingaberegister. Stellen Sie - sofern es nicht bereits der Fall ist - das Befehlsregister auf 0 und lassen Sie den Taktgeber 4 Takte lang laufen. Hierdurch wird die Zahl aus dem Eingaberegister zum Akkumulatorinhalt addiert. Weitere 4 Takte addieren den Inhalt des Eingaberegisters nochmals zum Akkumulator. Durch Setzen den Befehlsregisters auf 1 wird die Schaltung auf "*speichere*" eingestellt. Mit den folgenden 4 Takten wird der Inhalt des Akkumulators nun zum Speicherregister übertragen.

Dieses sehr einfache Modell kann nur äußerst beschränkte Aufgaben erledigen. Um es leistungsfähiger zu machen, sind weitere Befehle erforderlich, d.h. wir benötigen ein Befehlsregister mit mehr Stellen. In der einfachsten Erweiterung besitzt es 2 Stellen, so dass 4 verschiedene Befehle codiert werden können.

Die einzelnen Befehle sollen in dem erweiterten Modell auch nicht mehr von Hand in das Befehlsregister gesetzt werden, sondern in einem Speicher zur Verfügung stehen und automatisch abgearbeitet werden. Hierzu müssen die Befehle nacheinander aus dem Arbeitsspeicher geholt und decodiert werden. Anschließend müssen der Decodierung entsprechende Aktionen ausgeführt werden. Zur Ausführung der einzelnen Befehle werden jeweils 4 Takte benötigt, da das Modell mit 4-Bit-Schieberegistern arbeitet. Mit dem folgenden Takt wird dann der nächste Befehl aus dem Speicher geholt. Zur Übernahme eines Befehls in das Befehlsregister und zur Ausführung des Befehls sind also insgesamt 5 Takte erforderlich.

4.3.2 Einzelteile des Modell-Computers

4.3.2.1 Takteinheit

Die notwendige Unterteilung der 5 Takte in einen Befehlsübernahmetakt und 4 Ausführungstakte wird durch die folgende Schaltung realisiert.

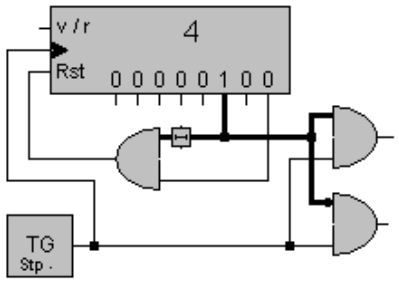


Abb. 4.3.2.1-1 Takteinheit für den Modell-Computer

Der Binärzähler ist als Modulo-5-Zähler geschaltet. Mit jedem 5. Takt wird er auf Null zurückgesetzt, da er beim Erreichen des Zustandes 5 ≙ 00000101 über das UND-Gatter ein 1-Signal am Rücksetzeingang erhält. Das Verzögerungsglied ist in dieser Schaltung notwendig. Ohne die Verzögerung würde aufgrund der Leitungslängen andernfalls bereits bei einem Wechsel des Binärzählers von 3 auf 4 ein Rücksetzen stattfinden, da dann kurzzeitig zwei 1-Signale am UND-Gatter anliegen und somit verfrüht einen Rücksetzimpuls hervorrufen.
Mit den beiden anderen als elektronische Tor geschalteten UND-Gattern wird erreicht, dass die 4 Ausführungstakte und der Befehlsübernahmetakt auf gesonderten Leitungen zur Verfügung stehen.

4.3.2.2 Befehlsregister, Befehlszählregister und Speicher

Zur Übernahme eines 2-Bit-Befehls aus dem Speicher werden zwei Flipflops als Befehlsregister benötigt. Um die entsprechende Speicherstelle zu adressieren braucht man ein 4-Bit-Adressregister. Da mit dem Modell-Rechner nur lineare Programme abgearbeitet werden sollen, d.h. die Befehle in aufeinanderfolgenden Speicherzellen stehen, kann dieses Adressregister als Binärzähler aufgebaut werden, der bei jedem Befehlsübernahmetakt automatisch die nächste Speicherzelle adressiert.
Befehlsregister und Befehlsadressregister (üblicher Sprachgebrauch: Befehlszählregister, program counter, PC) müssen somit an die Leitung angeschlossen werden, die nur jeden 5. Takt erhält.

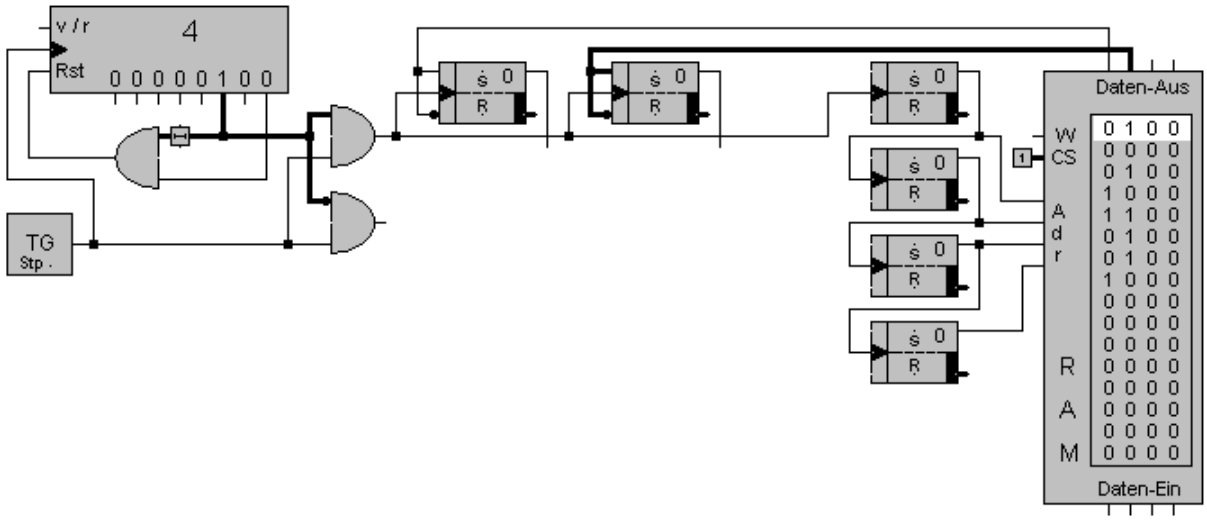


Abb. 4.3.2.2-1 Takteinheit, Befehlsregister und Speicher des Modell-Computers

Durch die Belegung des Chip-Select-Anschlusses am RAM mit 1-Signal stehen die durch das Befehlsadressregister adressierten Daten am Daten-Ausgang des RAMs ständig zur Verfügung und liegen somit an den Eingängen des Befehlsregisters an.

Bei einem 1-0-Wechsel des Signals auf der Taktleitung wird diese Bitkombination in das Befehlsregister übernommen und der Inhalt des Befehlsadressregisters um eins erhöht, so dass der nächste Befehl adressiert wird und zur Übernahme an den Befehlsregistereingängen zur Verfügung steht.

Als nächsten Teil des Modell-Computers betrachten wir das Rechenwerk.

4.3.2.3 Rechenwerk

Das in der nachstehenden Abbildung gezeigte Rechenwerk des Modell-Rechners besteht aus drei 4-Bit-Schieberegistern, von denen zwei mit einem Volladdierer und einem Flipflop zur Übertragsspeicherung als Serienaddierwerk geschaltet sind und das dritte als Zwischenspeicher dient.

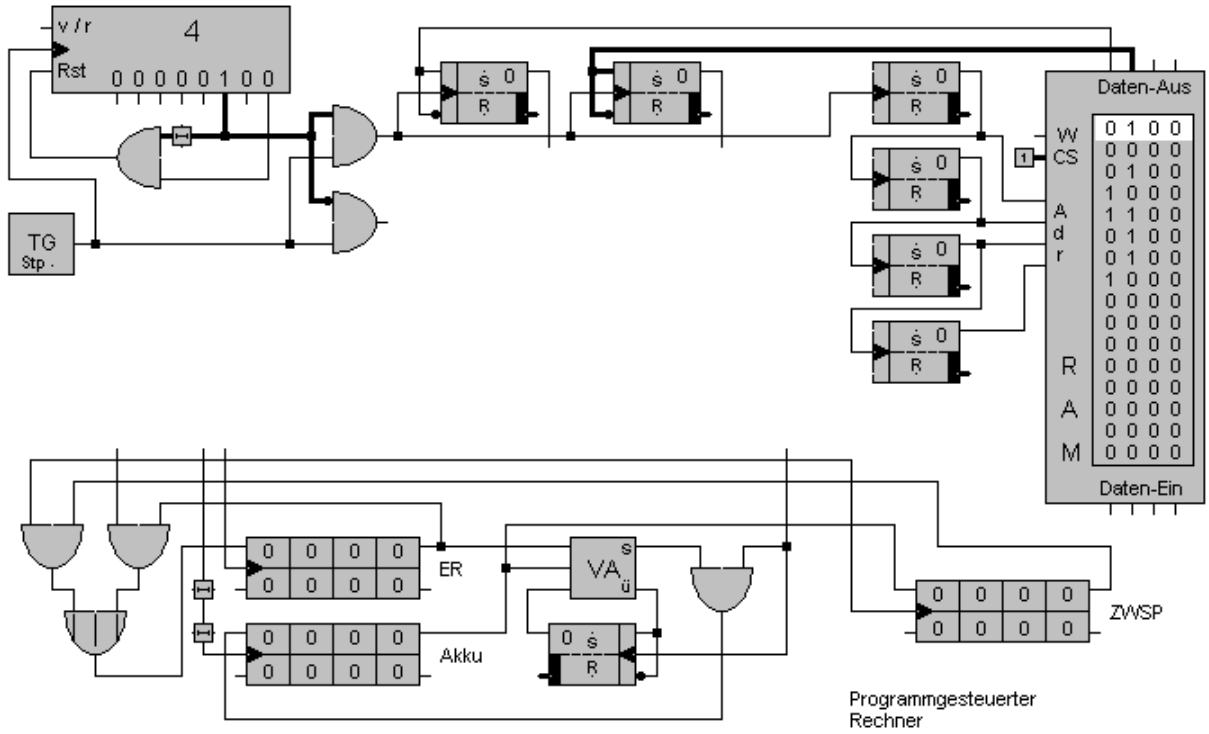


Abb. 4.3.2.3-1 Rechenwerk des Modell-Computers

Das hier realisierte Serienaddierwerk enthält kein besonderes Ergebnisregister wie die in Abschnitt 3.2.2 auf Seite 72 abgebildete Schaltung, sondern die Summe wird in das zweite Summandenregister übertragen, das hier nicht mehr als Ringschieberegister geschaltet ist. Damit geht bei der Addition der zweite Summand zwar verloren, aber stattdessen befindet sich am Ende die Summe in diesem Register. Ein in dieser Weise arbeitendes Register wird als Akkumulator bezeichnet.
Das Register für den ersten Summanden dient als Eingaberegister (ER). Die einzelnen Bits können nach dem Einschalten durch Anklicken mit der linken Maustaste verändert werden, so dass sich das Register mit einer beliebigen 4-stelligen Dualzahl belegen lässt. Es kann weiterhin als Ringschieberegister bei der Addition als auch zur Übernahme der Information aus dem Zwischenspeicher benutzt werden. Durch die UND-ODER-Kombination vor dem Eingang wird zwischen den beiden Möglichkeiten unterschieden. Wie das Register benutzt werden soll, hängt von dem auszuführenden Befehl ab, so dass die Steuerung der beiden Tore über den Befehlsdecoder erfolgen muss.

Das UND-Gatter rechts neben dem Volladdierer sorgt dafür, dass beim Übertragen des Akkumulatorinhaltes in den Zwischenspeicher der Akkumulator geleert wird. Nur wenn addiert wird, gelangt die Summe aus dem Volladdierer an den Akkumulatoreingang.

Als letzter Teil im Modell fehlt jetzt nun nur noch die Verbindung von Befehlsregister und Rechenwerk über den Befehlsdecoder. Die schaltungstechnische Realisierung des Befehlsdecoders ist aber erst möglich, wenn eine Festlegung bezüglich der Codierung der einzelnen Befehle erfolgt ist.

Wir wollen für die vier möglichen Befehle folgende Codierung festlegen:

Befehl	Kurzform	Codierung
Stoppe den Taktgeber	STP	00
Addiere den Inhalt des Eingaberegisters ER zum Akkumulatorinhalt (Ergebnis im Akku)	ADD	01
Übertrage den Akkumulatorinhalt in den Zwischenspeicher ZWSP	STO	10
Hole den Zwischenspeicherinhalt in das Eingaberegister ER	GET	11

4.3.2.4 Befehlsdecoder

Zur Realisierung der Decodierschaltung muss man sich klarmachen, zu welchen Teilen des Rechenwerks die Ausführungstaktsignale beim Vorliegen eines bestimmten Befehls weitergeleitet werden müssen.

Im Falle des STP-Befehls braucht nichts zum Rechenwerk weitergegeben zu werden, sondern der Stp-Eingang des Taktgebers muss ein 1-Signal erhalten. Nachdem der Taktgeber gestoppt wurde, muss dieses Signal zurückgesetzt werden, damit der Taktgeber auch wieder aktiviert werden kann. Man erreicht das, indem an das den STP-Befehl decodierende UND-Gatter eine Impulserzeugungsschaltung angeschlossen wird. Bei Auftreten des STP-Befehls (Bitkombination 00) wird am Ausgang des UND-Gatters ein 1-Signal hervorgerufen, das zu einem Impuls führt, der den Taktgeber stoppt.

Beim ADD-Befehl müssen die 4 Ausführungstakte zum Eingaberegister, zum Akkumulator, zum Übertragsspeicher und dem Tor für den Summenausgang des Volladdierers gelangen. Auch ist es notwendig, dass das Eingaberegister in diesem Fall als Ringschieberegister arbeitet, d.h. das als Tor wirkende rechte UND-Gatter vor dem Eingang des Eingaberegisters muss geöffnet werden.

Der STO-Befehl verlangt nur, dass die Ausführungstakte den Akkumulator und das Zwischenspeicherregister erreichen.

Für den GET-Befehl benötigt man Taktsignale am Zwischenspeicherregister und Eingaberegister, wobei das linke Tor geöffnet werden muss.

Die nachfolgende Schaltung zeigt die Realisierung dieser Bedingungen. Zusätzliche wurden einige Verzögerungsglieder eingesetzt, damit die Taktsignale erst dann an den Takteingängen der Register ankommen, wenn die Eingangssignale stabil anliegen.

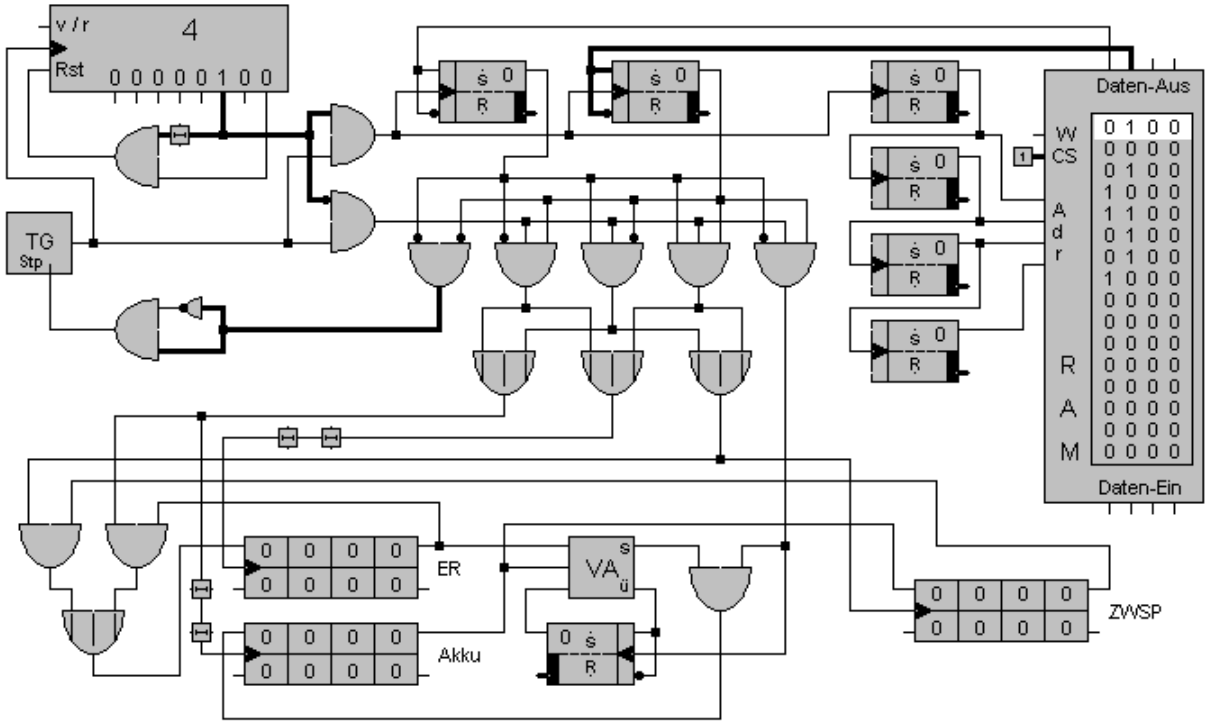


Abb. 4.3.2.4-1 fertiger Modell-Computers (Befehlsdecoder ergänzt)

Die Takteinheit, das Befehlsregister, das Befehlsadressregister und der Befehlsdecoder bilden das Steuerwerk der Modell-Computers.

4.3.2.5 Betrieb des Modells

Mit den oben angegebenen Befehlen ist es nun möglich, kleine Programme zu schreiben und von dem Modell abarbeiten zu lassen.
Hierzu müssen die Befehle im Speicher abgelegt werden. Die entsprechenden Bitkombinationen können Sie nach dem Einschalten eintragen, indem Sie die einzelnen Bits mit der linken Maustaste anklicken, wodurch sich der aktuelle Wert ändert.

Beispiel 1:

Die beiden nacheinander ins Eingaberegister eingegebenen Zahlen a und b sollen addiert, anschließend mit 2 multipliziert und das Ergebnis in den Zwischenspeicher gebracht werden.

Kurzform : $2 \cdot (a + b) \rightarrow$ ZWSP

Das folgende Programm leistet dies:

Befehl	Codierung	Aktion
		Eingabe: a in ER
ADD	01	a → Akku
STP	00	Eingabe: b in ER
ADD	01	a + b → Akku
STO	10	a + b → ZWSP
GET	11	a + b → ER
ADD	01	a + b → Akku
ADD	01	2(a + b) → Akku
STO	10	2(a + b) → ZWSP
STP	00	

Beispiel 2:

Kurzform: $2 \cdot a + 3 \cdot b \rightarrow$ ZWSP

Das Programm hierzu lautet:

Befehl	Codierung	Aktion
		Eingabe: a in ER
ADD	01	a → Akku
ADD	01	2a → Akku
STO	10	2a → ZWSP
STP	00	Eingabe: b in ER
ADD	01	b → Akku
ADD	01	2b → Akku
ADD	01	3b → Akku
GET	11	2a → ER
ADD	01	2a + 3b → Akku
STO	10	2a + 3b → ZWSP
STP	00	

Anhang A

Boolesche Algebra

Eine Menge **B** mit mindestens zwei Elementen und den zwei Verknüpfungen $\wedge$ und $\vee$ heißt „*Boolesche Algebra*“, wenn für alle $a, b, c \in \mathbf{B}$ folgende Gesetze gelten:

I. Kommutativgesetze:
1) $a \wedge b = b \wedge a$
2) $a \vee b = b \vee a$

II. Distributivgesetze:
1) $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$
2) $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$

III. Existenz neutraler Elemente:
Es gibt in **B** ein Einselement **1** (neutrales Element bzgl. $\wedge$)
mit der Eigenschaft $a \wedge \mathbf{1} = a$ (für alle $a \in \mathbf{B}$)

Es gibt in **B** ein Nullelement **0** (neutrales Element bzgl. $\vee$)
mit der Eigenschaft $a \vee \mathbf{0} = a$ (für alle $a \in \mathbf{B}$)

IV. Existenz eines komplementären Elementes:
Zu jedem $a \in \mathbf{B}$ gibt es genau ein $\bar{a} \in \mathbf{B}$ mit den Eigenschaften
1) $a \wedge \bar{a} = \mathbf{0}$
2) $a \vee \bar{a} = \mathbf{1}$
$\bar{a}$ heißt Komplement von a

Anhang B

Gesetze der Schaltalgebra

Eine Vereinfachung von Ausdrücken ist in vielen Fällen mit Hilfe der folgenden Gesetze möglich.

a, b, c seien Schaltvariablen, dann gelten:

Kommutativgesetze:
$$a \wedge b = b \wedge a$$
$$a \vee b = b \vee a$$

Assoziativgesetze:
$$(a \wedge b) \wedge c = a \wedge (b \wedge c)$$
$$(a \vee b) \vee c = a \vee (b \vee c)$$

Distributivgesetze:
$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$
$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$

Idempotenzgesetze:
$$a \wedge a = a$$
$$a \vee a = a$$

Komplementgesetze:
$$a \wedge \bar{a} = 0$$
$$a \vee \bar{a} = 1$$

Gesetz der doppelten Negation:
$$\bar{\bar{a}} = a$$

Absorptionsgesetze:
$$a \wedge (a \vee b) = a$$
$$a \vee (a \wedge b) = a$$

Gesetze von de Morgan:
$$\overline{a \wedge b} = \bar{a} \vee \bar{b}$$
$$\overline{a \vee b} = \bar{a} \wedge \bar{b}$$

Gesetz mit 0 und 1:
$$a \wedge 0 = 0$$
$$a \wedge 1 = a$$
$$a \vee 0 = a$$
$$a \vee 1 = 1$$
$$\bar{0} = 1$$
$$\bar{1} = 0$$

Anhang C

Lösungen zu einzelnen Aufgaben aus dem Text

Lösung zur Übungsaufgabe 1.4.2:

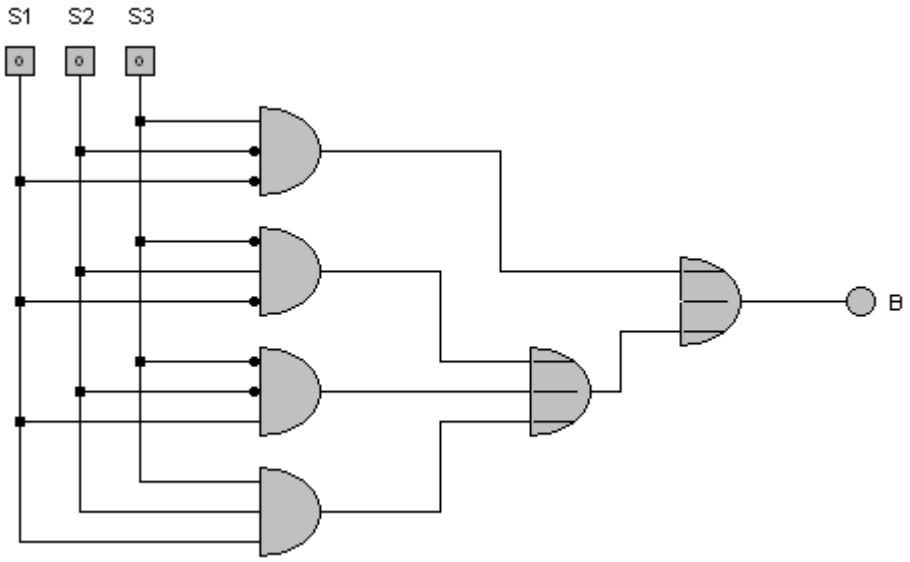
Aus dem Aufgabentext lässt sich die Funktionstabelle aufstellen. Mit den drei Schaltvariablen ergeben sich 8 mögliche Zustände für die Schalterstellungen:

S_1	S_2	S_3	B
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Wir bilden die disjunktive Normalform:

$$B = (\overline{S_1} \wedge \overline{S_2} \wedge S_3) \vee (\overline{S_1} \wedge S_2 \wedge \overline{S_3}) \vee (S_1 \wedge \overline{S_2} \wedge \overline{S_3}) \vee (S_1 \wedge S_2 \wedge S_3)$$

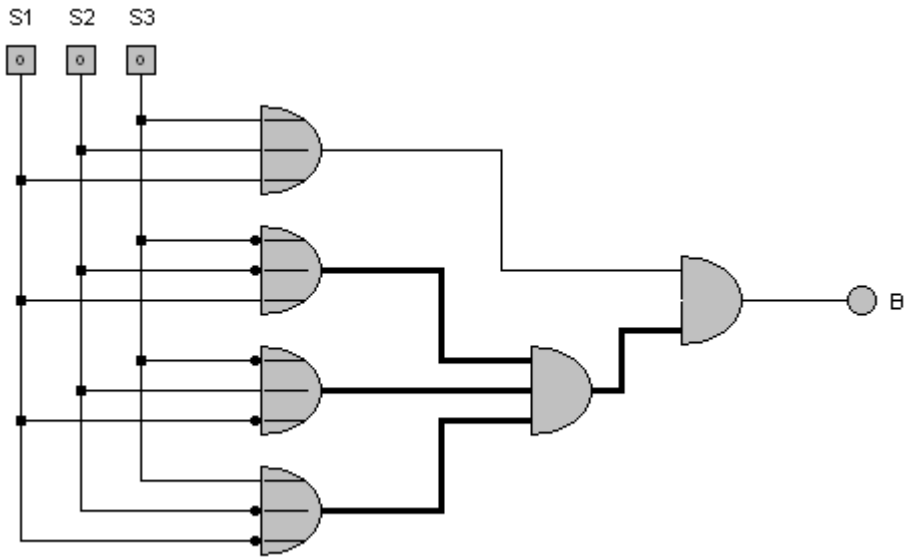
Das zugehörige Schaltnetz sieht folgendermaßen aus:



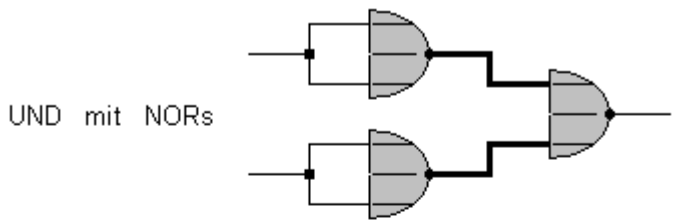
Wir bilden die konjunktive Normalform:

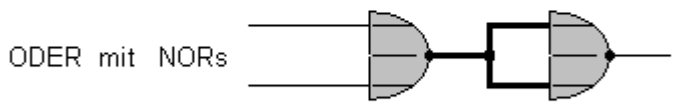
$$B = (S_1 \vee S_2 \vee S_3) \wedge (S_1 \vee \overline{S_2} \vee \overline{S_3}) \wedge (\overline{S_1} \vee S_2 \vee \overline{S_3}) \wedge (\overline{S_1} \vee \overline{S_2} \vee S_3)$$

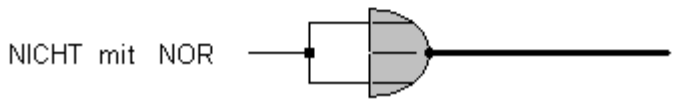
Das zugehörige Schaltnetz sieht folgendermaßen aus:



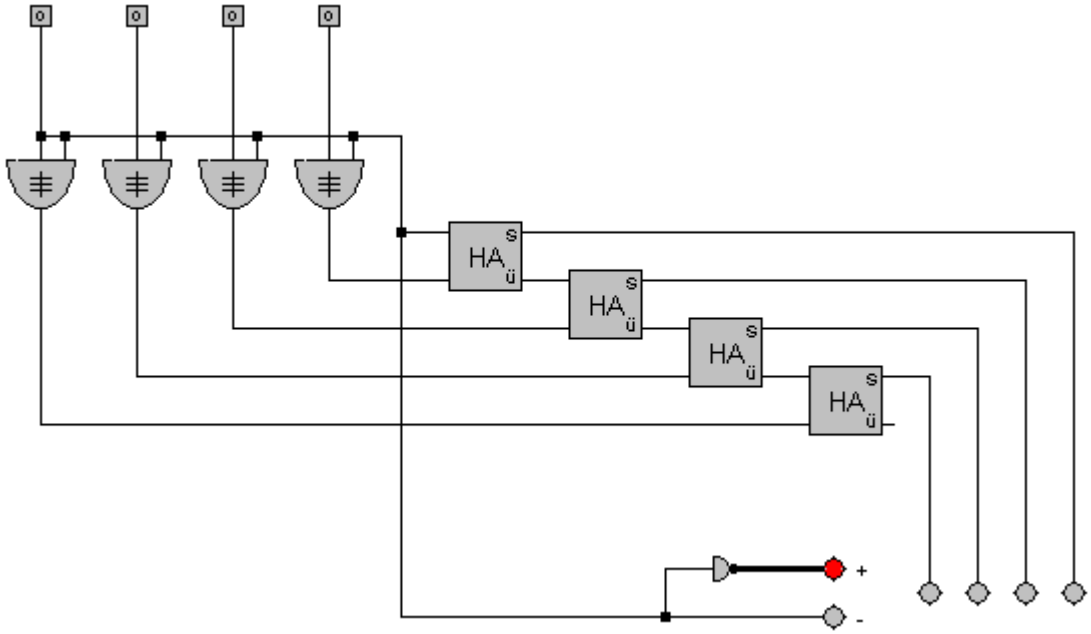
Lösung zur Aufgabe 1.2:



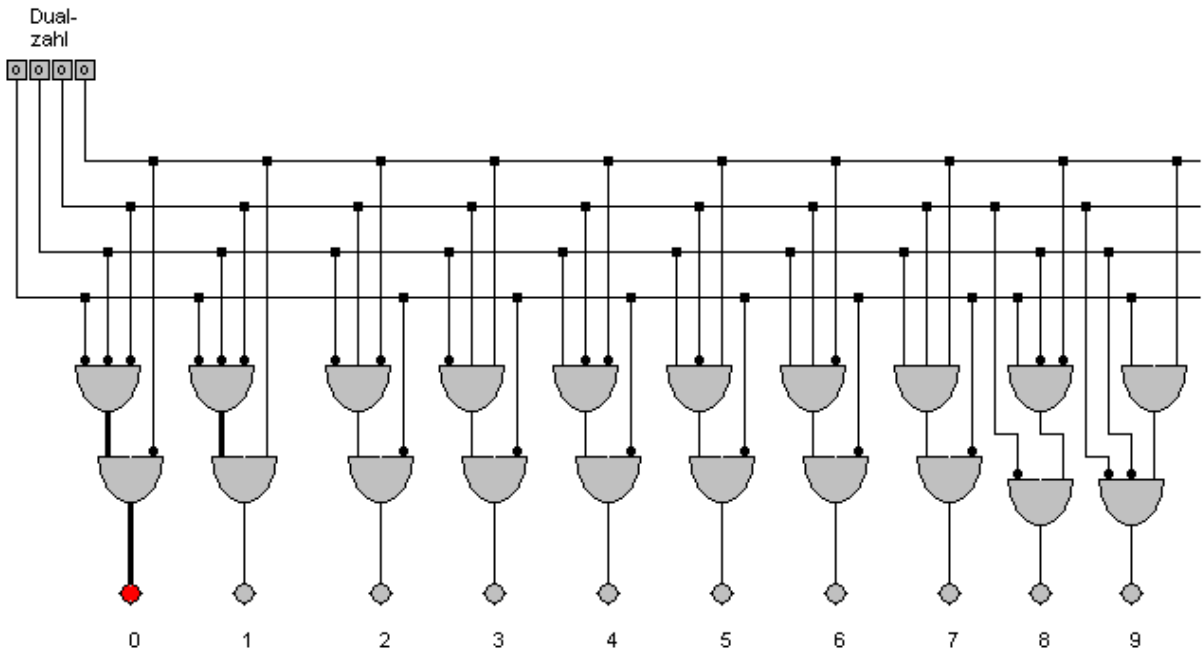




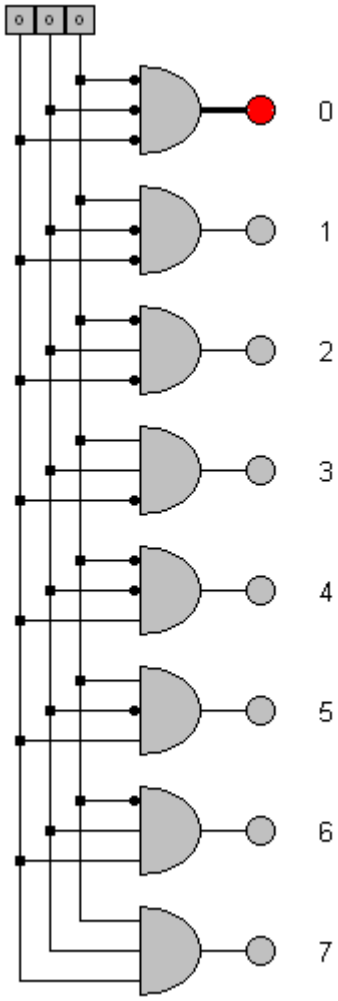
Lösung zur Aufgabe 2.10:



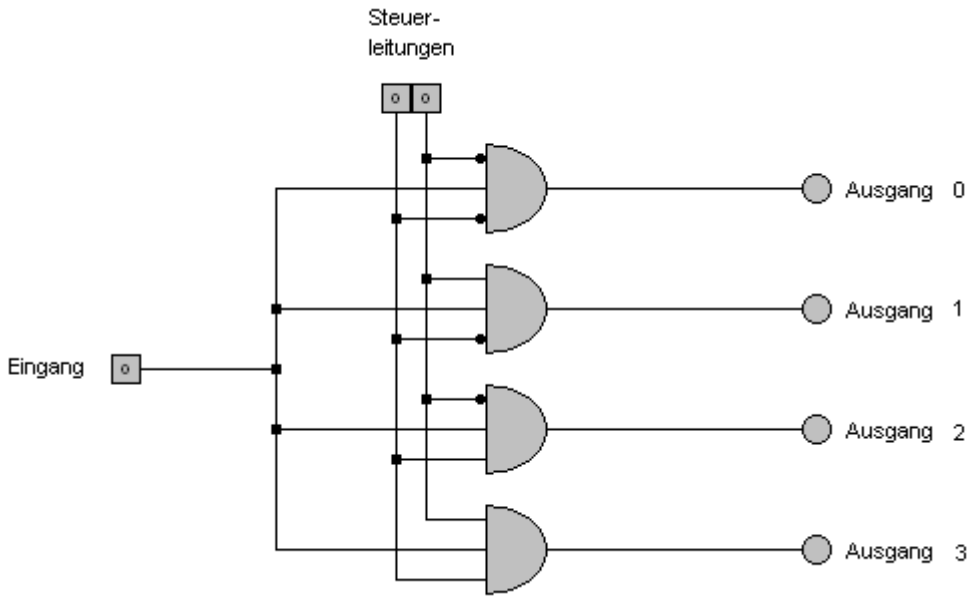
Lösung zur Aufgabe 2.12:



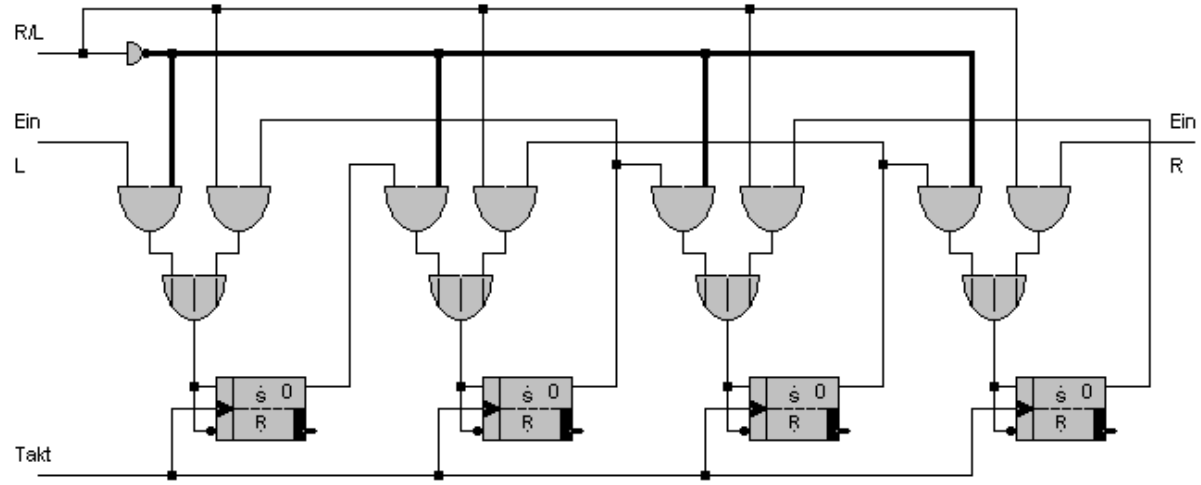
Lösung zur Aufgabe 2.13:



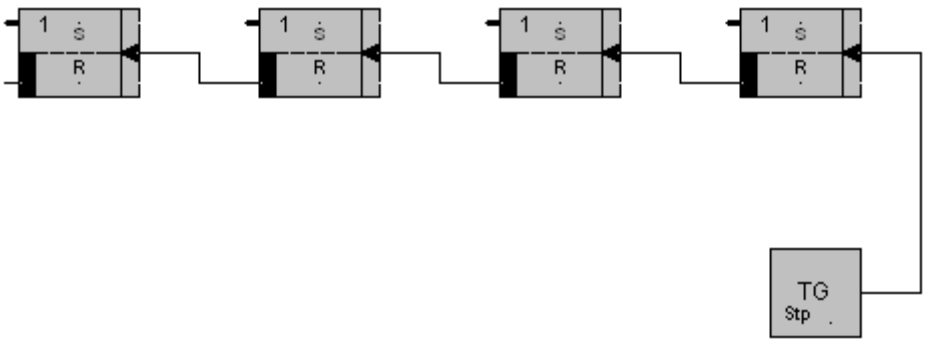
Lösung zur Aufgabe 2.15:



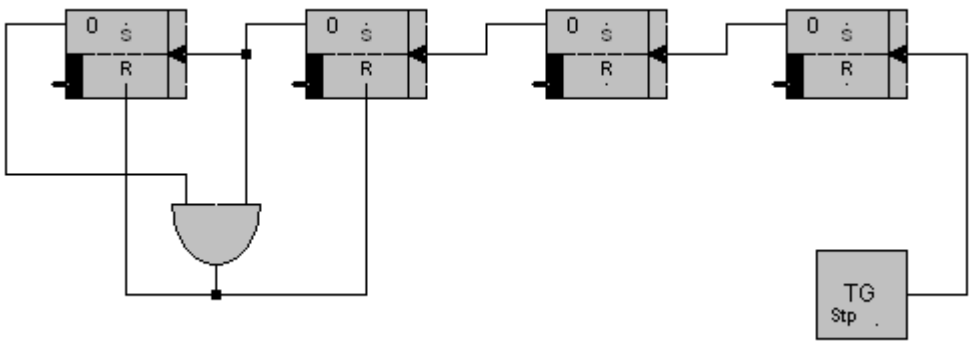
Lösung zur Aufgabe 3.6:



Lösung zur Aufgabe 3.11:



Lösung zur Aufgabe 3.12:



Anhang D

Weitere Aufgaben

1. Gegeben sind folgende Codes:

Dezimal	Aiken	Drei-Exzeß	Gray	Glixon
0	0 0 0 0	0 0 1 1	0 0 0 0	0 0 0 0
1	0 0 0 1	0 1 0 0	0 0 0 1	0 0 0 1
2	0 0 1 0	0 1 0 1	0 0 1 1	0 0 1 1
3	0 0 1 1	0 1 1 0	0 0 1 0	0 0 1 0
4	0 1 0 0	0 1 1 1	0 1 1 0	0 1 1 0
5	1 0 1 1	1 0 0 0	0 1 1 1	0 1 1 1
6	1 1 0 0	1 0 1 0	0 1 0 1	0 1 0 1
7	1 1 0 1	1 1 0 1	0 1 0 0	0 1 0 0
8	1 1 1 0	1 0 1 1	1 1 0 0	1 1 0 0
9	1 1 1 1	1 1 0 0	1 1 0 1	1 0 0 0

   Entwickeln Sie zu jedem Code eine Codier- und Decodierschaltung.

2. Entwickeln Sie eine Codierschaltung, die eine 4-stellige Dualzahl in den Gray-Code umwandelt.

3. Mit 4 Bits (= 1 Tetrade) lassen sich 16 verschiedene Zeichen codieren. Bei jedem der oben aufgeführten Codes bleiben 6 Tetraden ungenutzt, die sogenannten **Pseudotetraden.** Entwickeln Sie für jeden Code eine Schaltung zur Pseudotetradenerkennung.

4. Entwickeln Sie eine Schaltung zur Multiplikation zweiereinstelliger Dualzahlen.

5. Entwickeln Sie ein Schaltnetz, das zwei vierstellige Dualzahlen auf Gleichheit testet.

6. Entwickeln Sie ein Schaltnetz, das die größere von zwei zweistelligen Dualzahlen bestimmt, indem es am Ausgang ein 1-Signal liefert, wenn die erste Zahl die größere ist.

7. Zur Überwachung einer technischen Anlage werden drei Computer eingesetzt. Weil der Ausfall eines Rechners nicht zum Ausfall des Gesamtsystems führen soll, wird aus den Ausgangssignalen der drei gleichartigen Rechner bitweise ein mehrheitliches Signal gebildet (was mindestens zwei Rechner „sagen“, wird als richtig angenommen).
   Entwickeln Sie eine Schaltung, die die Ausgangssignale (jeweils nur ein Bit) der Rechner verknüpft und anzeigt, welcher der drei Rechner höchstwahrscheinlich fehlerhaft arbeitet (in Anlehnung an eine Einsendeaufgabe zu [12]).

8. Bauen Sie ein Ringschieberegister auf und realisieren Sie damit eine Lauflicht.

9. Bauen Sie ein Schieberegister mit 8 Flipflops auf. Verknüpfen Sie die Ausgänge der letzten beiden Flipflops mit einem EXKLUSIV-ODER-Glied und leiten Sie das Ergebnis zum Eingang des ersten Flipflops.
   Welches Verhalten zeigt die Schaltung beim Betrieb ?

10. Entwickeln Sie einen asynchronen 4-Bit-Rückwärtszähler, der mit einer Startzahl geladen werden kann.

11. Entwickeln Sie einen synchronen 3-Bit-Rückwärtszähler.

12. Entwickeln Sie einen Zähler modulo 5.

13. Entwickeln Sie einen umschaltbaren Vorwärts-/Rückwärtszähler (asynchron).

14. Entwickeln Sie eine 1:6-Frequenzteiler.

15. Bauen Sie ein Serienaddierwerk, das mit zwei Schieberegistern auskommt.

16. Entwickeln Sie eine elektronische Weiche, die auf einer Leitung ankommende Impulse abwechselnd auf zwei Leitungen verteilt.

17. Entwickeln Sie ein Schaltwerk, das ein Folge von 5 Zuständen viermal durchläuft und dann anhält.

18. Entwickeln Sie ein Schaltwerk, das bei einem 1-Signal auf einer „Rückmeldeleitung“ die Zustandfolge 0 1 2 3 4 5 6 und bei einem 0-Signal die Folge 0 1 2 3 zyklisch durchläuft.

Literaturverzeichnis

[1] Päthe / Müller — Grundlagen der Technischen Informatik
Fernuniversität Hagen 1980-1984

[2] Harbeck / Jäschke / Küster / Reimers / Starke — Boolesche Algebra und Computer
Vieweg 1976

[3] Whitesitt / Stumpf — Einführung in die Boolesche Algebra
Bagel-Vieweg 1973

[4] Deller — Boolesche Algebra
Diesterweg Salle 1976

[5] Whitesitt — Boolesche Algebra und ihre Anwendungen
Vieweg 1973

[6] Kreß — Digitale Elektronik und Computer
Diesterweg Salle 1977

[7] Balzert — Informatik 2
Hueber-Holzmann 1978

[8] Modrow — Automaten / Schaltwerke / Sprachen
Dümmler 1986

[9] Leybold-Heraeus — Grundlagen der Digitaltechnik
Leybold-Heraeus 1976

[10] Dresch / Frobel / Koschorrek — Aufbau, Arbeitsweise und Anwendung von Datenverarbeitungsanlagen
Schöningh 1985

[11] Brockmann — Von der Aussagenlogik über Logikschaltungen zum Mikroprozessor
Schöningh 1985

[12] Schneeweiß / Waldschmidt — Mikroprogrammierte Schaltwerke und Mikrorechner
Fernuniversität Hagen 1981-1984

Stichwortverzeichnis

A

ADD-Befehl 102
Addition im Dualsystem 26
Adressbit 86
Adressdecoder 88
Adresse 86
Adressleitung 85
Adressleitungen 52
Adressregister 92
Adressteil 90
Aiken-Code 112
Akkumulator 97
Algebra, boolesche 105
ALU 83
Ampelsteuerung 81
Analysepunkt 76
AND-Gatter 8

Ä

ÄQUIVALENZ-Funktion 38

A

Arbeitsspeicher 83, 84
Assemblerbefehl 91
Ausgabe, parallel 69
Ausgabe, seriell 69
Ausgangssignal 5
Ausgangsvariable 5

B

Basis 18
Befehlsdecoder 97, 102
Befehlsregister 89
Befehlszählregister 89
Bereichsüberschreitung 37
Binärzähler 78
Binärzustand 5
Bit 36
Byte 36

C

Chip-Select 88
codieren 41
Codierer 42
CPU 83

D

Data-Latch 62
Datenbus 91
D-Auffangflipflop 62
Decoder 47
decodieren 41
Demultiplexer 56
Dezimal-Dual-Umsetzer 42
Diodenmatrix 52
Disjunktion 9
Disjunktionsterm 13
Drei-Exzeß-Code 112
Dual-Dezimal-Umsetzer 43
Dualsystem 24
Dualzahlen, negative 33

E

Einerkomplement 34
Eingaberegister 97
Eingangssignal 5
Eingangsvariable 5
Emitter 18
EXKLUSIV-ODER-Funktion 38

F

Festwertspeicher 53
flankengesteuert 63
Flankentriggerung 65
Flipflop 59
Folgeadressen 80
Folgeadressmatrix 80
Frequenzteiler 79
Funktionstabelle 6

G

Gegenzahl 32
GET-Befehl 102
Glixon-Code 112
Gray-Code 112
Grundflipflop 60
Grundfunktionen 8

H

Halbaddierer 27

I

Impulserzeugung 57
Inverter 10

J

JK-Master-Slave-Flipflop 64

K

Kippschaltung 59
Kollektor 18
Komplement 10
Konjunktion 8
Konjunktionsterm 11
Kontaktlogik 16
Koppeldioden 48

L

LED ... 32
Leuchtdioden ... 32

M

Makrobefehl ... 94
Maxterm ... 13
Mikrobefehl ... 94
Mikroprogramm ... 94
Mikroprogrammspeicher ... 96
Mikroprogramm-Steuerwerk ... 94
Minterm ... 12
Multiplexer ... 54
MUX ... 54

N

NAND-Gatter ... 21
Negat ... 10
Negation ... 10
Negator ... 10
NICHT-Funktion ... 10
NICHT-Verknüpfung ... 10
NOR-Gatter ... 23
Normalform, disjunktive ... 11
Normalform, konjunktive ... 13
NOT-Gatter ... 10
Nur-Lese-Speicher ... 52

O

ODER-Funktion ... 9
ODER-Verknüpfung ... 9
Operationsteil ... 90
OR-Gatter ... 9

P

Paralleladdierer ... 31
Parallelspeicher ... 85
Programm ... 104
PROM ... 52
Pseudotetrade ... 112

R

RAM, dynamisches ... 85
RAM, statisches ... 84
RAM-Baustein ... 88
Rechenwerk ... 83, 100
Rechenwerk, serielles ... 73
Refresh ... 85
Register ... 68
Relais ... 16
Restwert-Methode ... 26
ROM ... 52
RS-Flipflop ... 59
RS-Master-Slave-Flipflop ... 62
Rückkopplung ... 7, 58
Rücksetzeingang ... 59

S

Schaltalgebra ... 5
Schaltalgebra, Gesetze ... 106
Schaltfunktion ... 5
Schaltnetz ... 7
Schaltung, logische ... 5
Schaltung, sequentielle ... 58
Schaltvariable ... 5
Schaltwerk ... 7, 58
Schaltwerk, autonomes ... 79
Schieberegister ... 64, 68
Schreib-Lese-Speicher ... 84
Serienaddierwerk ... 71
Serienspeicher ... 85
Setzeingang ... 59
Sieben-Segment-Ansteuerung ... 50
Sieben-Segment-Anzeige ... 47
Signalverzögerung ... 57
Spannungspegel ... 5
Speicherelement ... 58
Speicherzelle ... 85
Steuermatrix ... 82
Steuersignal ... 84, 89
Steuerwerk ... 83, 89
STO-Befehl ... 102
STP-Befehl ... 102
Subtraktion im Dualsystem ... 32

T

Takt ... 61
Taktflanke ... 65
Tetrade ... 112
Tor, elektronisches ... 55
Torschaltung ... 55
Transistor ... 18
Transistor-Transistor-Logik ... 5
Tri-State-Gatter ... 87

Ü

Übertragung, serielle ... 70

U

UND-Funktion ... 8
UND-Glied ... 8
UND-Verknüpfung ... 8

V

Verknüpfung ... 6
Verzögerungsglied ... 57
Volladdierer ... 29

Z

Zähler, asynchron ... 75
Zähler, synchron ... 75
Zweierkomplement ... 34
Zweiersystem ... 24