

6. Register-Transfer-Entwurf

- **Entwurfsebenen**

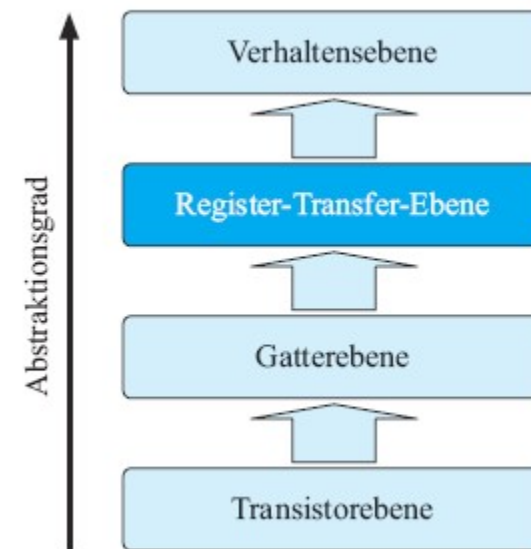
- ein komplexes System wird hierarchisch entworfen
- eine Ebene baut auf der nächsten auf
- von unten nach oben nimmt das Maß der Abstraktion zu

- **bisher**

- Transistorebene
 - Wie baut man Gatter aus Transistoren?
- Gatterebene
 - Wie baut man Schaltnetze und Schaltwerke aus Gattern?

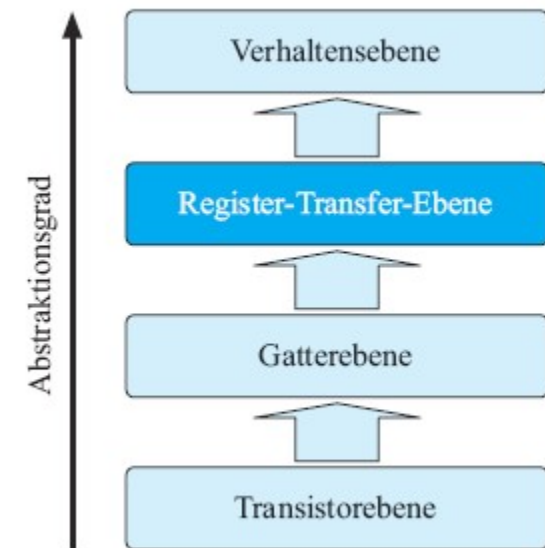
- **jetzt**

- Register-Transfer-Ebene



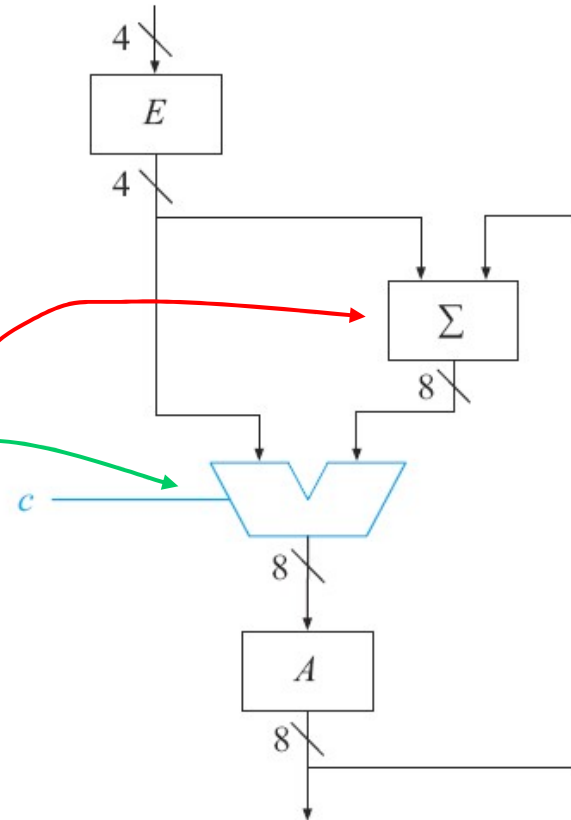
Register-Transfer-Ebene

- auch RT-Ebene oder RTL (Register Transfer Level) genannt
- höhere Abstraktionsstufe als Gatterebene um Produktivität beim Entwurf zu steigern
 - man muss sich nicht mehr mit so vielen technischen Details befassen
- **Abstraktion**
 - statt um 0 und 1 geht es um ganze Datenworte
 - zusammengehörige Signalleitungen werden zu Datenpfaden gebündelt
 - Register statt einzelner Flip-Flops
 - komplexe Funktionseinheiten statt einzelner Logikgatter
 - Multiplexer, Demultiplexer, Zähler, Schieberegister, Addierer, ALUs, etc.



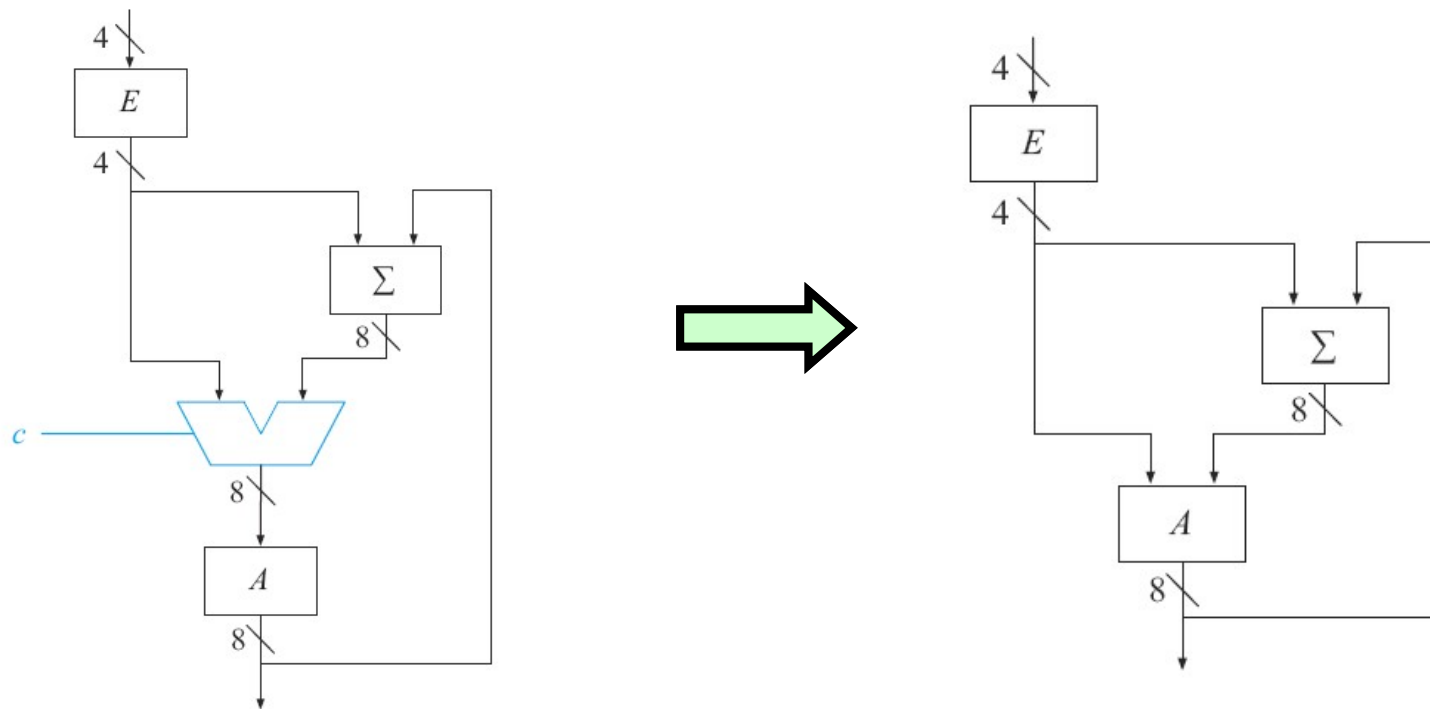
Beispiel: Akkumulator

- **Blockschaltbild besteht praktisch immer aus**
 - Register zum Abspeichern von Werten
 - Eingabewerte: Eingaberegister (E)
 - Ergebnisse: Ausgaberegister (A)
 - kann auch beide Rollen übernehmen (A)
 - sind getaktet
 - Funktionale Einheiten
 - kombinatorische Logik (Schaltnetze)
 - hier: Addierer
 - Multiplexer
 - zur Festlegung der Datenströme
 - Steuerlogik, bzw. Steuerleitungen
 - zur Steuerung der Multiplexer (c)
 - zur Steuerung der Register (z.B. WE, Reset)
 - zur Steuerung der funktionalen Einheiten (z.B. Add/Sub)



Vereinfachungen

- der Übersichtlichkeit halber werden beim RT-Entwurf die Multiplexer am Anfang häufig weggelassen:



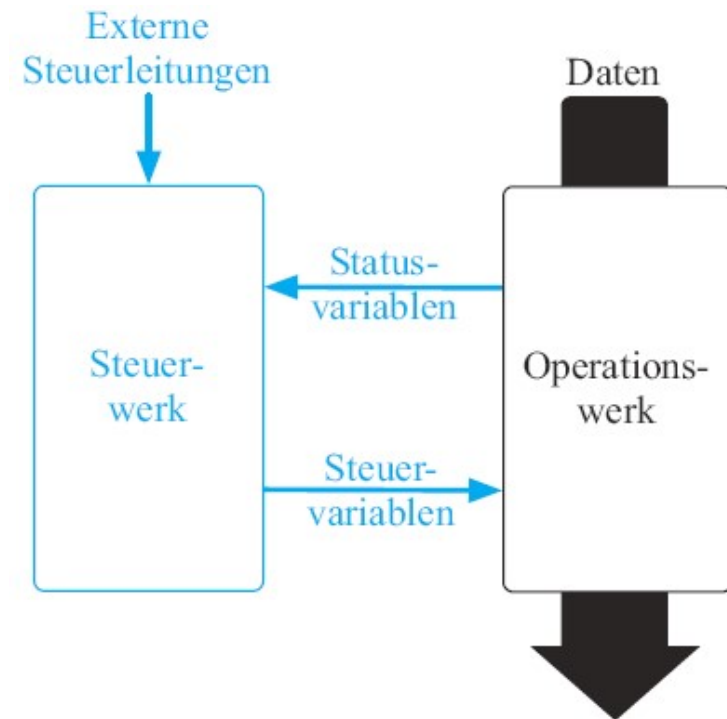
Aufteilung in Steuer- und Rechenwerk

- **Rechenwerk**

- im RT-Entwurf häufig auch Datenpfad oder Operationswerk genannt
- wird über Steuerleitungen gesteuert
- gibt dem Steuerwerk Feedback über Statusleitungen, z.B.
 - höchstwertiges Bit eines Registers: Vorzeichen
 - Carry-Bit zur Signalisierung eines Überlaufs

- **Steuerwerk**

- erzeugt die Steuersignale für den Datenpfad
- kann wieder selbst von außen Steuersignale bekommen
 - z.B. ein Befehlswort zur Auswahl der Operation, falls der Datenpfad verschiedene Berechnungen erlaubt



Aufteilung in Steuer- und Rechenwerk (2)

- **Vorteil**
 - Reduktion der Komplexität
 - Steuer- und Rechenwerk sind für sich weniger komplex als die Gesamtschaltung
 - Nach Spezifikation können beide Teile nahezu unabhängig voneinander entwickelt werden
 - verschiedene Entwickler
 - kürzere Entwicklungszeit durch Parallelisierung

Beispiel: Entwurf 4-Bit-Multiplizierer

- **Blockmultiplikation**

- Operanden A und B sollen hier in zwei 2-Bit Blöcke aufgeteilt werden, da 2-Bit Multiplizierer vorhanden sind

- A_H und A_L , bzw. B_H und B_L , mit jeweils 2 Bit

$$A = 2^2 \cdot A_H + A_L$$

$$B = 2^2 \cdot B_H + B_L$$

- Multiplikation

$$C = A \cdot B$$

$$= (2^2 A_H + A_L) \cdot (2^2 B_H + B_L)$$

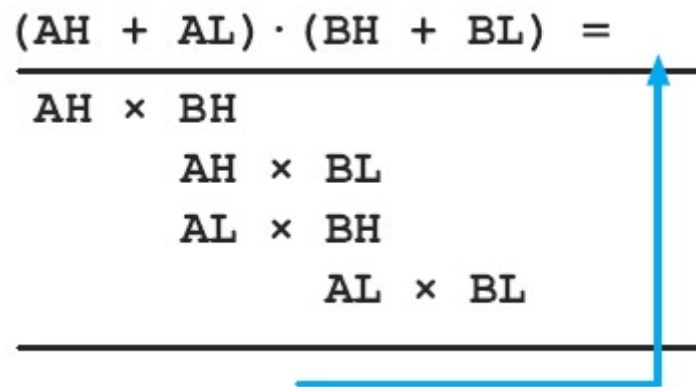
$$= 2^4 (A_H \cdot B_H) + 2^2 (A_H \cdot B_L) + 2^2 (A_L \cdot B_H) + A_L \cdot B_L$$

$$= 2^4 (A_H \cdot B_H) + 2^2 (A_H \cdot B_L + A_L \cdot B_H) + A_L \cdot B_L$$

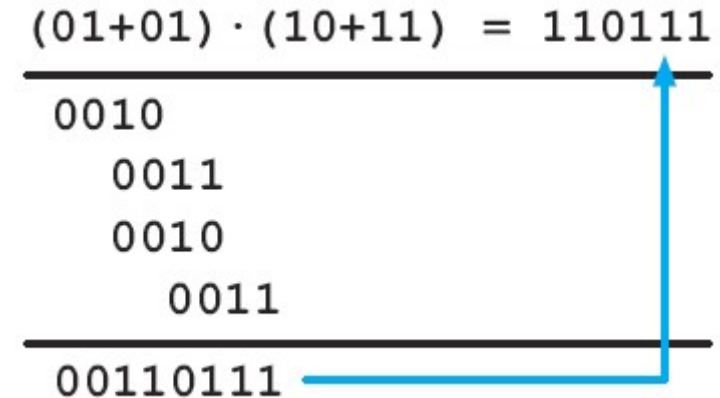
- es müssen also 4 Teilprodukte ($A_H B_H$, $A_H B_L$, $A_L B_H$ und $A_L B_L$) gebildet werden, die bitversetzt aufaddiert werden müssen

Beispiel: Entwurf 4-Bit-Multiplizierer (2)

- symbolisch dargestellt

$$\begin{array}{r} (AH + AL) \cdot (BH + BL) = \\ \hline AH \times BH \\ \quad AH \times BL \\ \quad \quad AL \times BH \\ \quad \quad \quad AL \times BL \\ \hline \end{array}$$


Schema

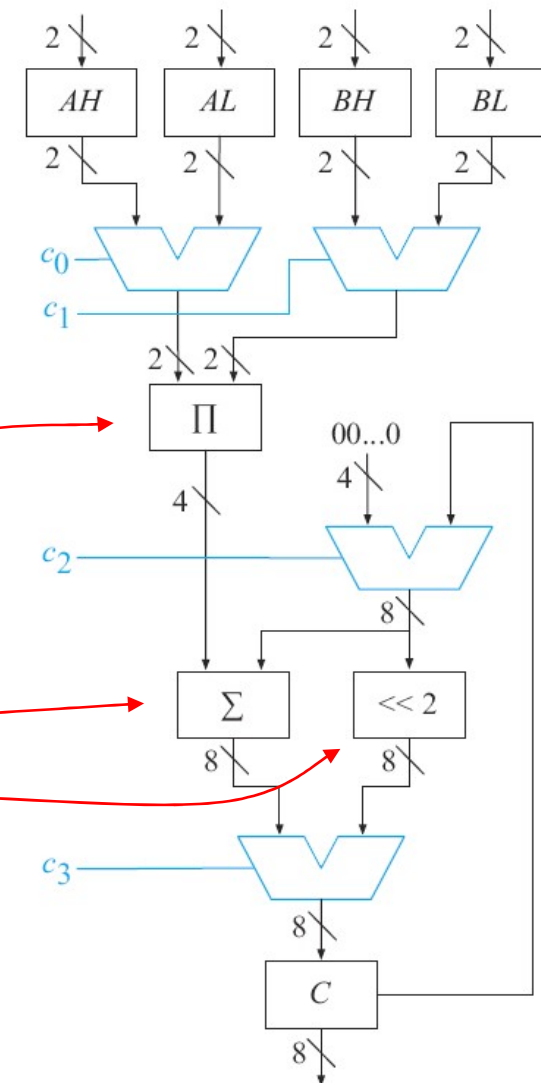
$$\begin{array}{r} (01+01) \cdot (10+11) = 110111 \\ \hline 0010 \\ \quad 0011 \\ \quad \quad 0010 \\ \quad \quad \quad 0011 \\ \hline 00110111 \end{array}$$


Beispiel: $5 \cdot 11 = 55$

- Vorteil
 - Multiplizierer muss nur 2x2 Bit multiplizieren (sehr viel kleiner als 4x4-Bit-Multiplizierer)
- Nachteil
 - Berechnung benötigt mehrere Schritte (oder evtl. auch mehrere Multiplizierer)
- Kompromiss zwischen Fläche und Laufzeit

Beispiel: Entwurf 4-Bit-Multiplizierer (3)

- **Register-Transfer-Entwurf**
 - zunächst den Datenpfad, dann das Steuerwerk entwerfen
- **Ziel hier: minimaler Hardwareaufwand**
 - Verzicht auf parallele Berechnung von Teilergebnissen
- **Datenpfad**
 - Funktionseinheiten
 - ein 2-Bit-Multiplizierer
 - ein 8-Bit-Addierer
 - ein 2-Bit-Links-Shifter
 - Multiplexer zur Steuerung des Datenflusses
 - Eingaberegister A und B
 - Ausgaberegister C



Beispiel: Entwurf 4-Bit-Multiplizierer (4)

- **Register-Transfer-Operationen, die mit dem Datenpfad möglich sind**

- jede Register-Transfer-Operation berechnet ein Ergebnis in C

$$C \leftarrow \dots$$

- das Ergebnis ist entweder eine Summe

$$C \leftarrow (0 \text{ oder } C) + \text{Teilprodukt}$$

- also: Laden

$$C \leftarrow \text{Teilprodukt}$$

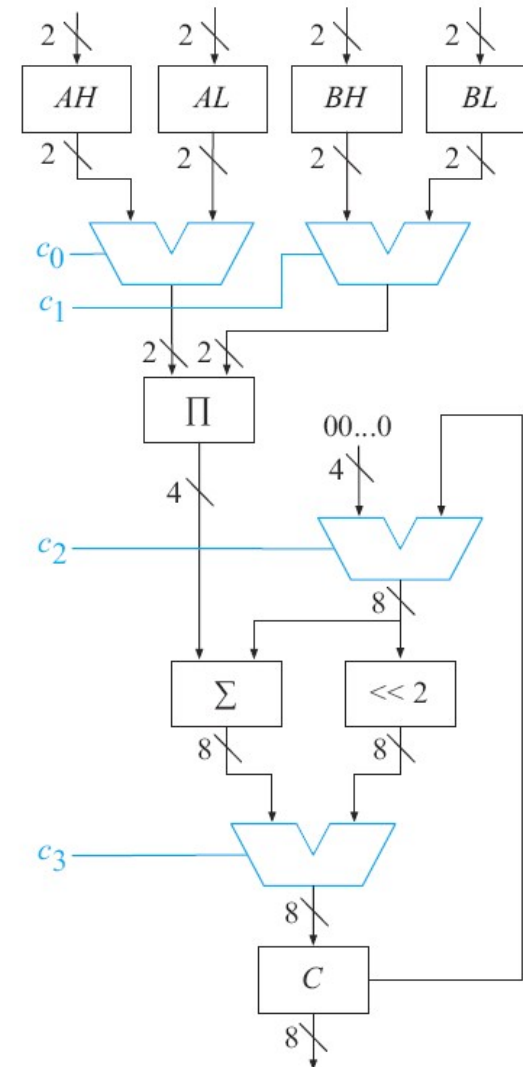
- oder: Akkumulieren

$$C \leftarrow C + \text{Teilprodukt}$$

- oder C um 2 Bit nach links geschiftet

- Shiften

$$C \leftarrow C \ll 2$$



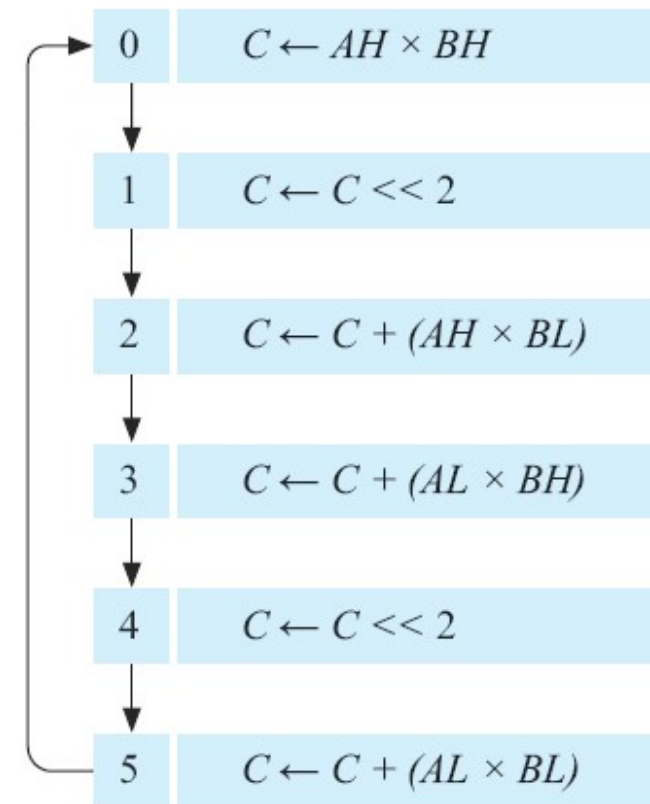
Beispiel: Entwurf 4-Bit-Multiplizierer (5)

- **Entwicklung Steuerwerk**

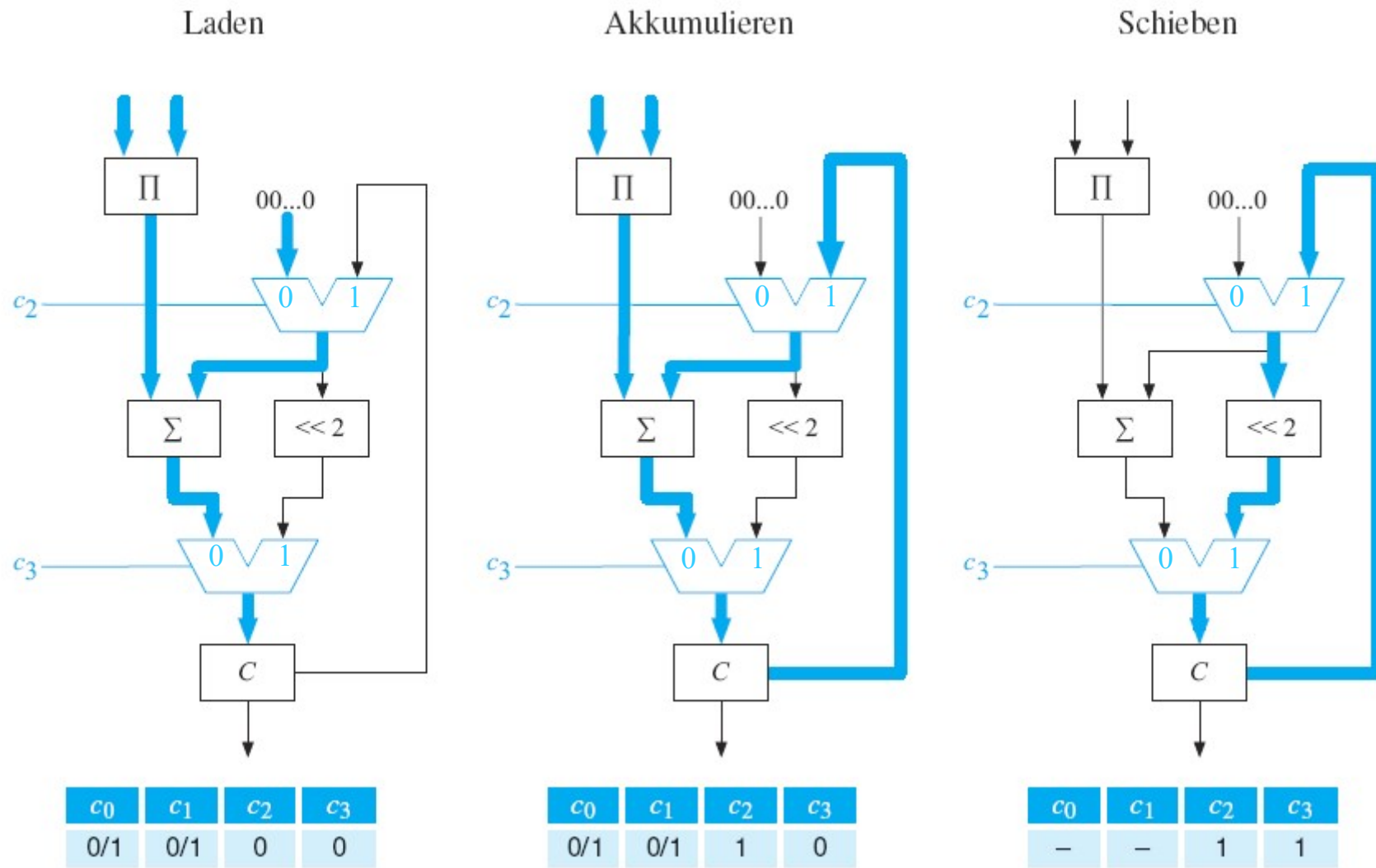
- Berechnung nach dem Horner-Schema

$$C = ((A_H B_H) 2^2 + A_H B_L + A_L B_H) 2^2 + A_L B_L$$

- kann mit 6 Register-Transfer-Operationen durchgeführt werden
- Benötigt werden nur drei verschiedene Register-Transfer-Operationen
 - Laden Takt 0
 - Schieben Takt 1 und 4
 - Akkumulieren Takt 2, 3 und 5
- Zusätzlich muss jeweils noch das richtige Teilprodukt (über c_0 und c_1) ausgewählt werden



Beispiel: Entwurf 4-Bit-Multiplizierer (6)



Beispiel: Entwurf 4-Bit-Multiplizierer (7)

- **Anschlüsse des Steuerwerkes**

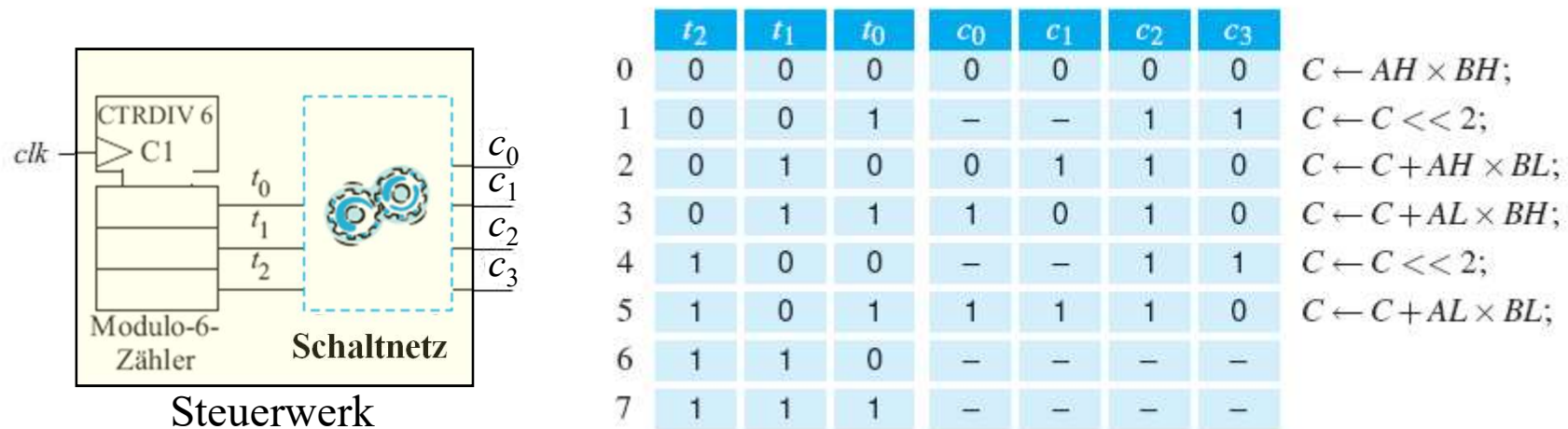
- Steuerleitungen c_0, \dots, c_3
 - sind die einzigen Leitungen zwischen dem Steuer- und Operationswerk
- Statusleitungen nicht vorhanden
 - der Ablauf der Multiplikation hängt nicht von den Werten im Datenpfad ab
- externe Steuerleitungen nicht vorhanden
 - man könnte aber z.B. noch ein Startsignal als Eingang hinzufügen
- Taktleitung

- **Schaltwerk**

- die Berechnung benötigt 6 Takte
- das entspricht einem Schaltwerk (FSM, Finite State Machine) mit 6 Zuständen
 - Moore-Automat, da es keine Eingänge gibt
- einfacher ist die Implementierung mit einem Modulo-6-Zähler, da die Zustände immer sequentiell durchlaufen werden (0, 1, 2, 3, 4, 5, 0, 1, 2, 3, 4, 5, 0, ...)

Beispiel: Entwurf 4-Bit-Multiplizierer (8)

- **Vollständige Spezifikation des Steuerwerkes**

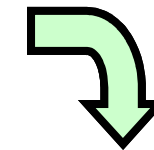
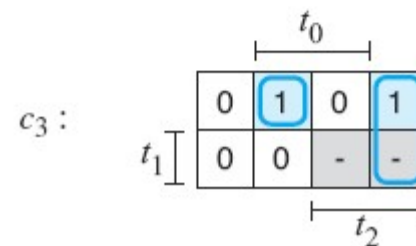
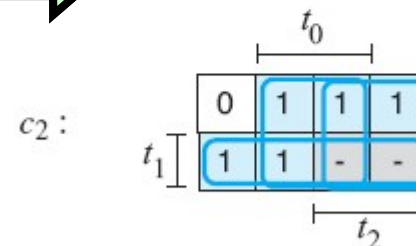
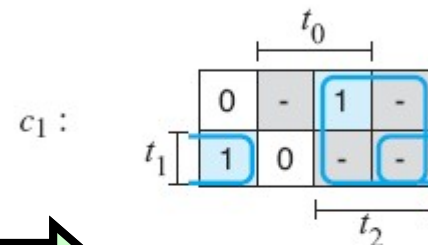
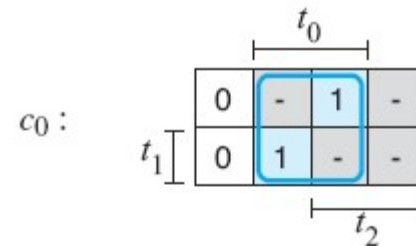
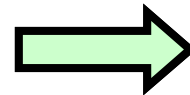


- im allgemeinen Fall werden Steuerwerke eher als Schaltwerke (FSMs. *Finite State Machines*) implementiert, da die Zustandsübergänge häufig noch von externen Steuersignalen und/oder Statusvariablen abhängen
 - Entwurfsablauf: Zustandsübergangsdiagramm, Zustandsübergangstabelle, Schaltfunktionen mit KV-Diagramm oder Quine McCluskey

Beispiel: Entwurf 4-Bit-Multiplizierer (9)

- KV-Diagramme

	t_2	t_1	t_0	c_0	c_1	c_2	c_3
0	0	0	0	0	0	0	0
1	0	0	1	-	-	1	1
2	0	1	0	0	1	1	0
3	0	1	1	1	0	1	0
4	1	0	0	-	-	1	1
5	1	0	1	1	1	1	0
6	1	1	0	-	-	-	-
7	1	1	1	-	-	-	-



$$c_0 = t_0$$

$$c_1 = \overline{t_0} t_1 \vee t_2$$

$$c_2 = t_0 \vee t_1 \vee t_2$$

$$c_3 = t_0 \overline{t_1} \overline{t_2} \vee \overline{t_0} t_2$$

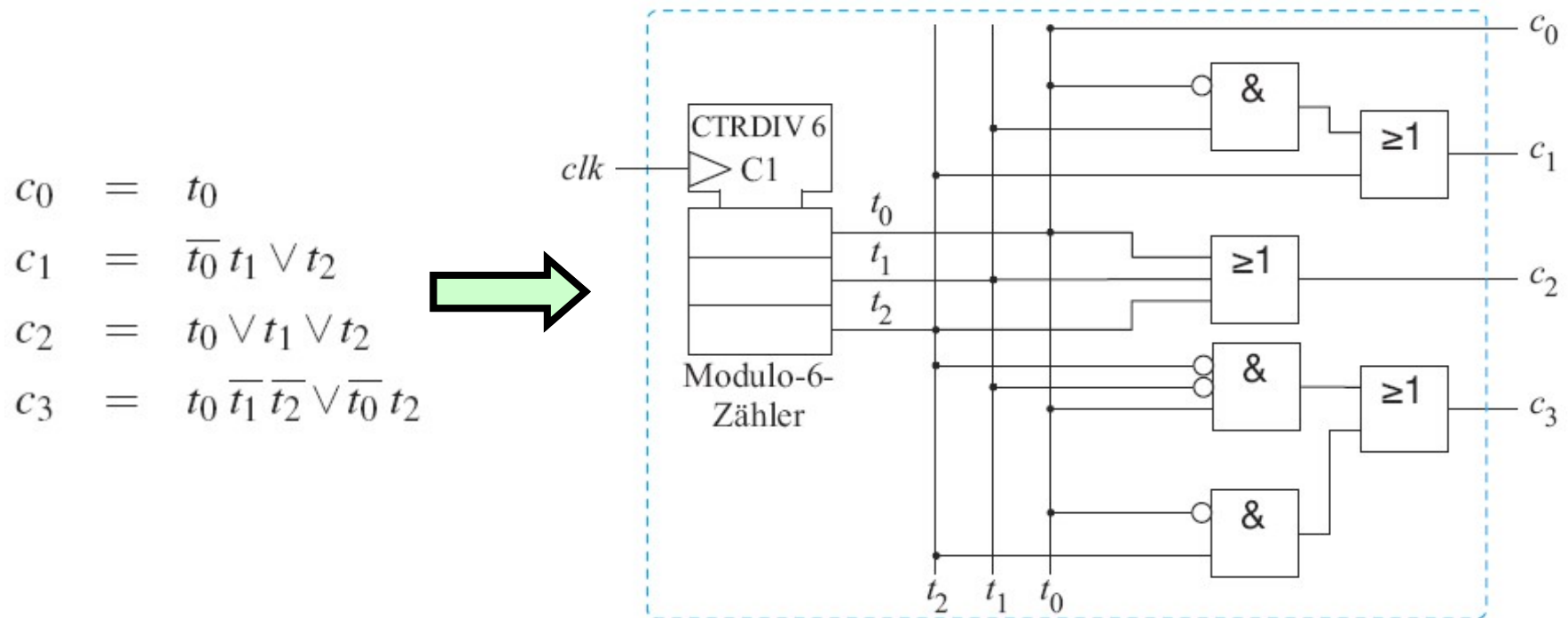
besser,
da schneller
auszufüllen:

c_0 :

	$t_2 \backslash t_1 t_0$	00	01	11	10
0			-	1	
1		-	1	-	-

Beispiel: Entwurf 4-Bit-Multiplizierer (10)

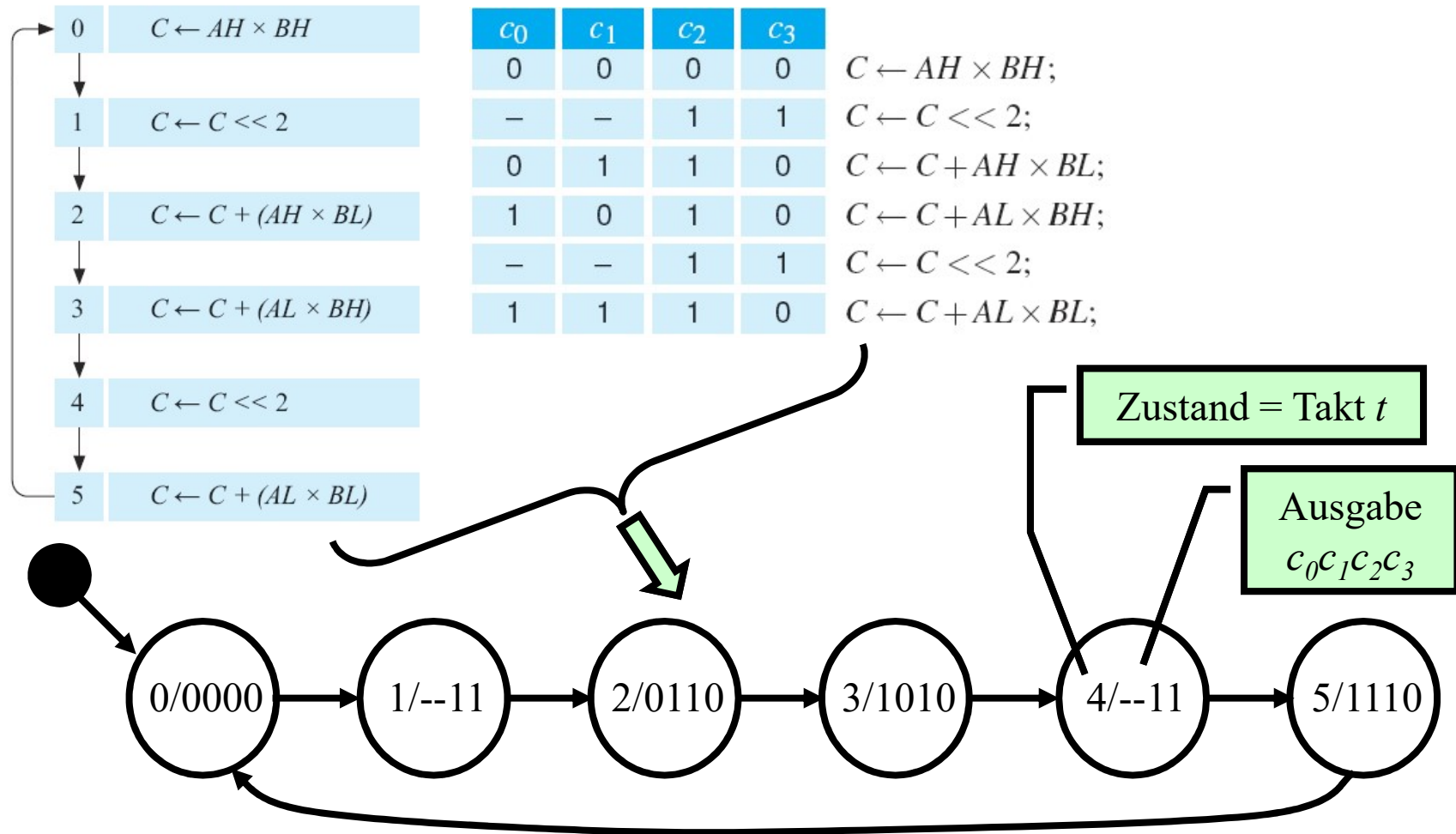
- vollständiges Steuerwerk



- Alternative: Implementierung als Moore-Automat (Standardmethode)
 - 3-Bit-Register statt Modulo-6-Zähler
 - drei zusätzliche Signale, die den nächsten Zustand kodieren
 - Übernahme der drei Signale ins Register

Standardmethode für Beispiel

- FSM als Moore-Automat



Standardmethode für Beispiel (2)

- aus dem Zustandsübergangsdiagramm entsteht die Zustandsübergangstabelle
- als Zustandskodierung wählt man sinnvollerweise die Taktnummer
 - drei Bit sind notwendig (t_2, t_1, t_0)

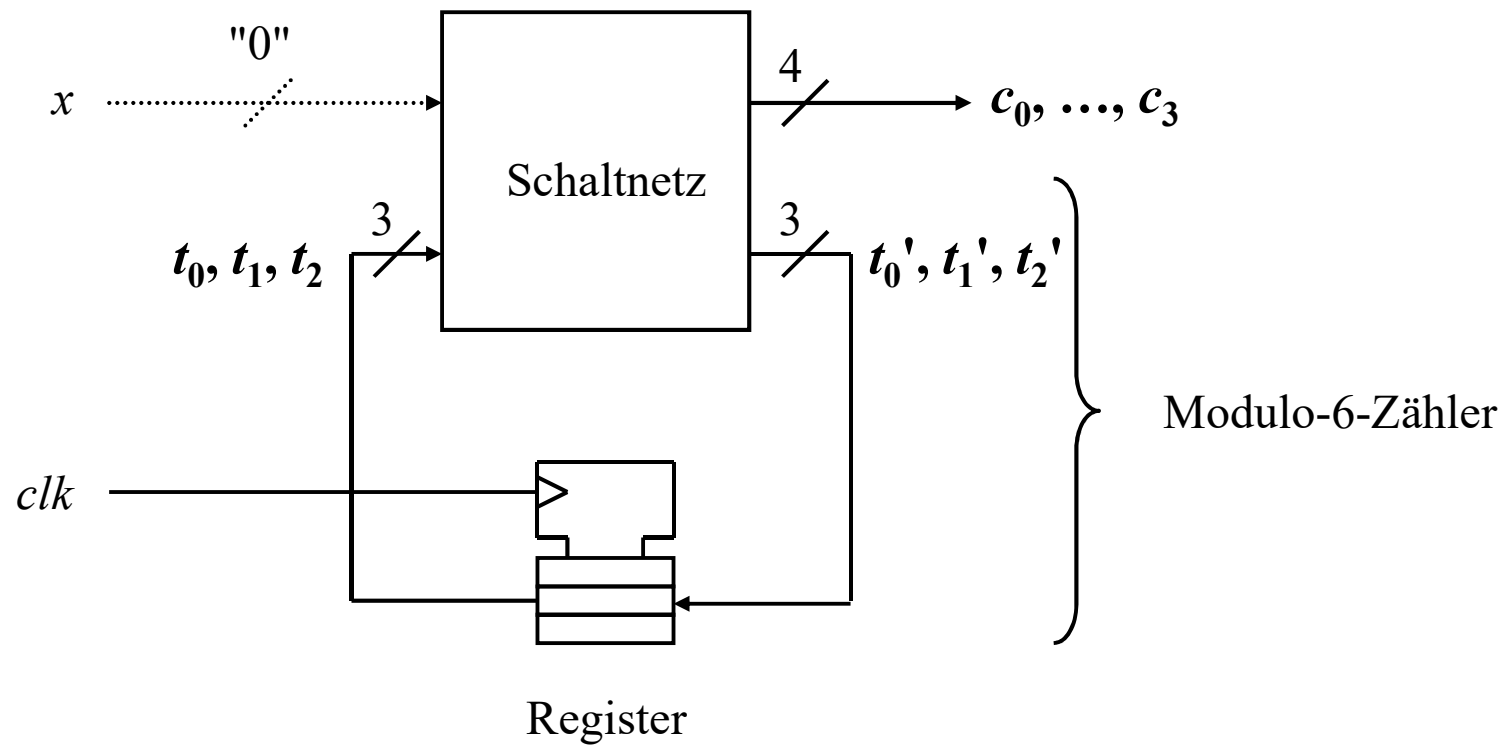
aktueller Zustand			nächster Zustand			Ausgabe			
t_2	t_1	t_0	t_2'	t_1'	t_0'	c_0	c_1	c_2	c_3
0	0	0	0	0	1	0	0	0	0
0	0	1	0	1	0	-	-	1	1
0	1	0	0	1	1	0	1	1	0
0	1	1	1	0	0	1	0	1	0
1	0	0	1	0	1	-	-	1	1
1	0	1	0	0	0	1	1	1	0
1	1	0	-	-	-	-	-	-	-
1	1	1	-	-	-	-	-	-	-

führt zum synchronen modulo 6 Zähler

führt zum Ausgabe-schaltnetz wie gehabt

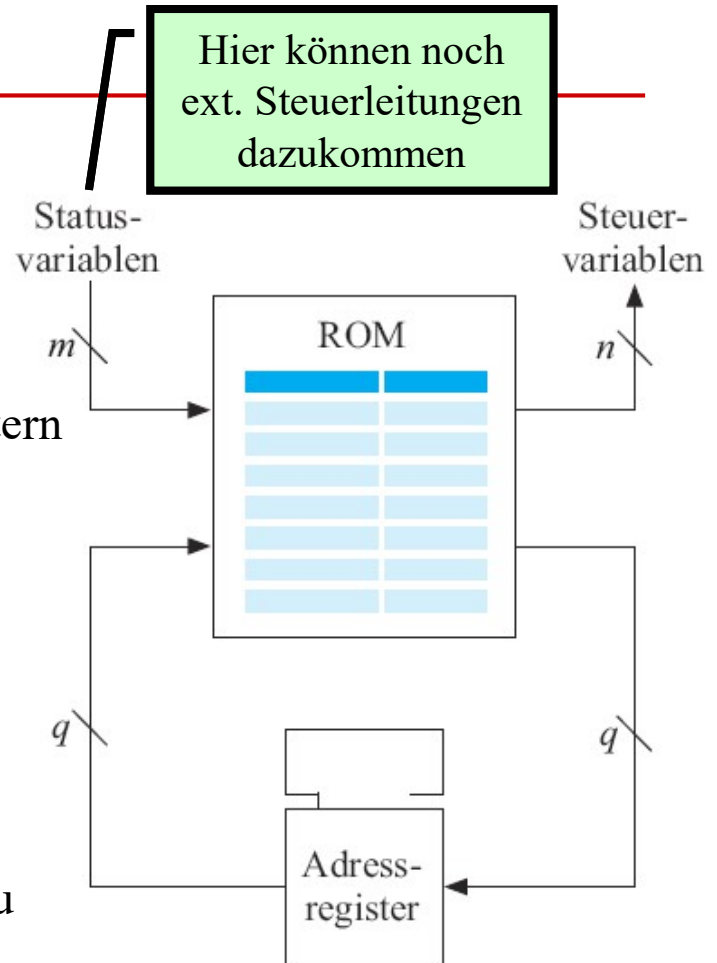
Standardmethode für Beispiel (3)

- Standard-Schaltwerk



Mikroprogrammierung

- **Steuerwerk zur Steuerung eines Datenpfades**
 - Implementierung als Schaltwerk
 - spezialisierte Schaltung aus einzelnen Gattern
 - solch ein Steuerwerk nennt man "fest verdrahtetes Steuerwerk"
 - Alternative: "mikroprogrammiertes Steuerwerk"
 - Kernelement ist ein Speicher, meist ROM
 - dadurch flexibler
 - Änderung des Speicherinhalts führt zu völlig neuen Abläufen



Äquivalent zu einem Mealy-Automaten mit ROM statt PLA zur Implementierung der Schaltfunktionen

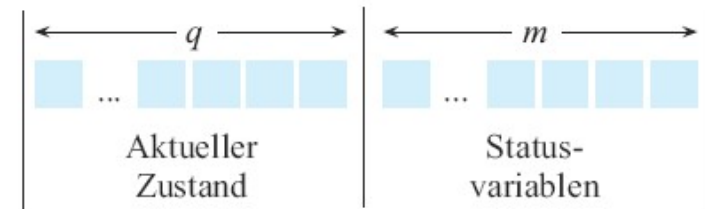
Mikroprogrammierung (2)

- **Adressregister**

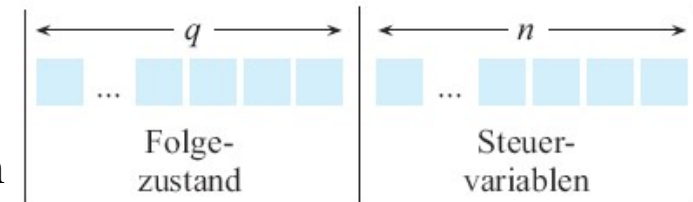
- enthält den Zustand des Steuerwerkes
- wird jetzt aber als Teil der Adresse der nächsten auszuführenden Mikroinstruktion (Register-Transfer-Operation) im ROM interpretiert

- **ROM**

- enthält Mikroprogramm
 - Adresse setzt sich aus aktuellem Zustand und Statusvariablen (+ ext. Steuersignale) zusammen
- Speicherinhalt an gegebener Adresse ist eine Mikroinstruktion
 - Mikroinstruktion enthält Steuersignale und Folgezustand (also im wesentlichen die Adresse der nächsten auszuführenden Mikroinstruktion)



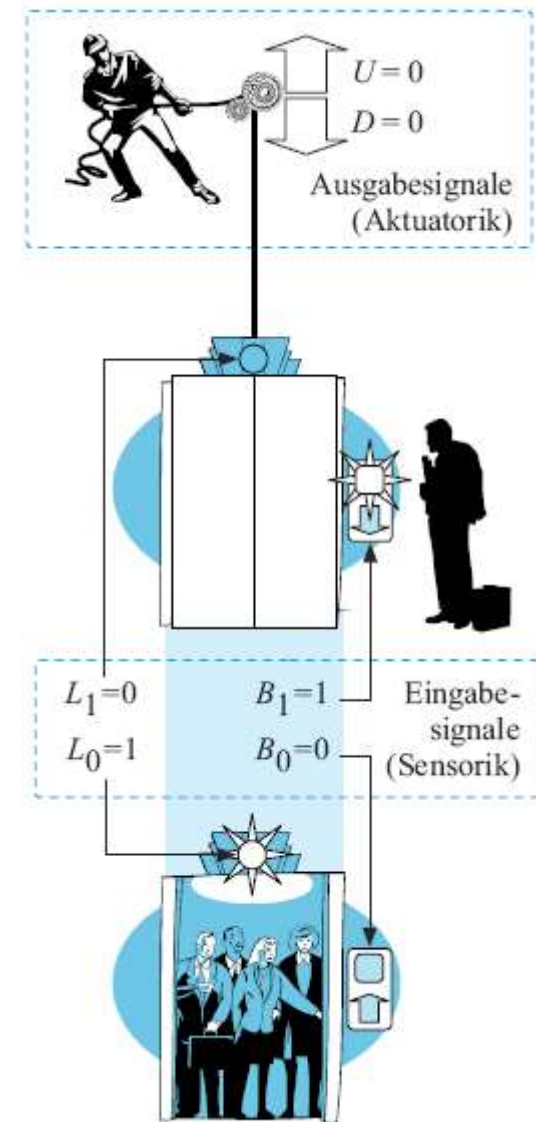
Adresse



Inhalt: Mikroinstruktion

Beispiel: Fahrstuhlsteuerung

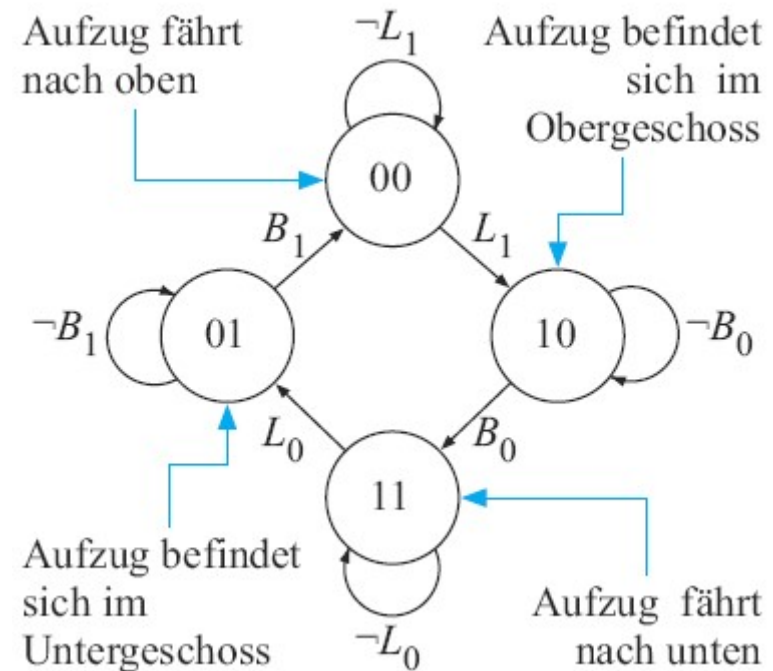
- **Fahrstuhl**
 - zwei Stockwerke (unten = 0, oben = 1)
- **Statusvariablen**
 - L_0 : Kabine befindet sich im unteren Stockwerk
 - L_1 : Kabine befindet sich im oberen Stockwerk
- **Eingabesignale (externe Steuerleitungen)**
 - B_0 : untere Ruftaste gedrückt
 - B_1 : obere Ruftaste gedrückt
- **Ausgabesignale**
 - U : Kabine bewegt sich nach oben (*Up*)
 - D : Kabine bewegt sich nach unten (*Down*)



Beispiel: Fahrstuhlsteuerung (2)

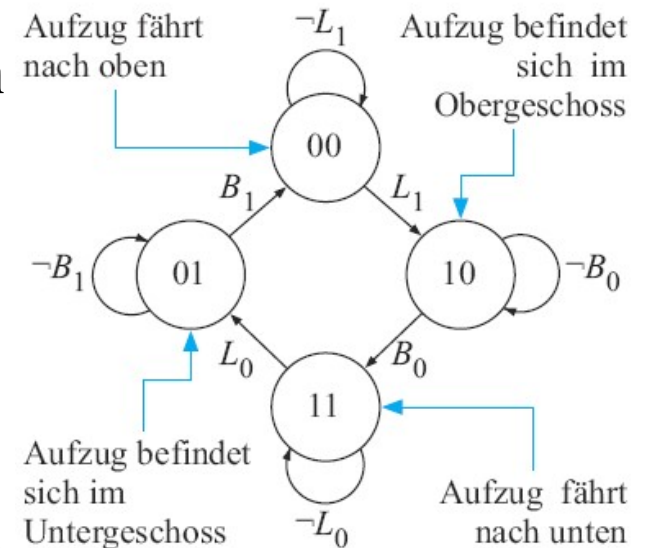
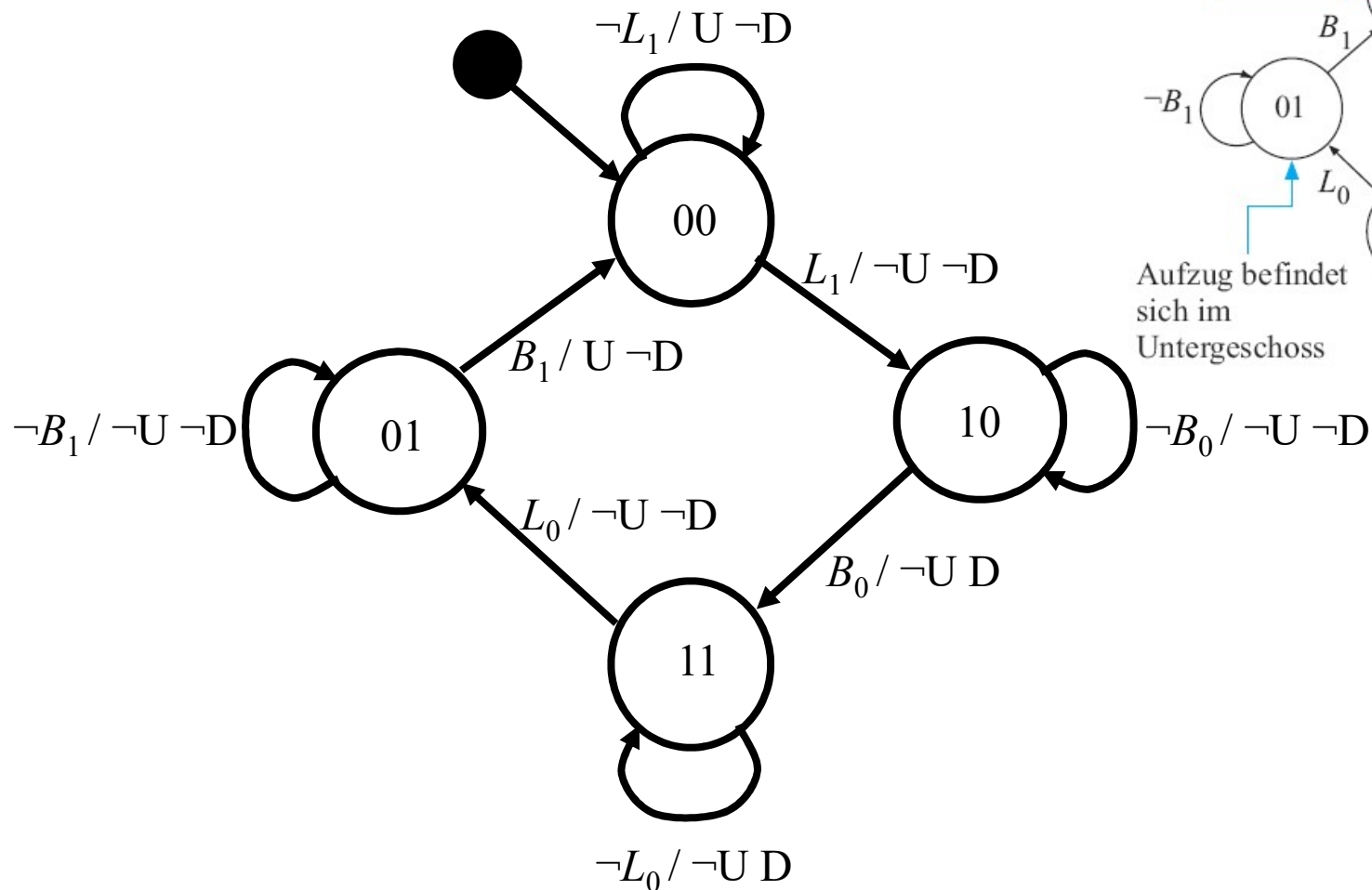
- **Zustandsübergangsdiagramm**

- vier Zustände modellieren das Verhalten des Fahrstuhls
- stark vereinfacht
 - z.B. registriert der Aufzug nicht, dass unten gedrückt wird, wenn der Fahrstuhl noch nach oben fährt
 - Türsteuerung fehlt
- Bemerkungen
 - die FSM (Finite State Machine, Schaltwerk) kann mit hoher Frequenz getaktet werden
 - für viele Takte bleibt die FSM im selben Zustand, bis ein Eingangssignal das ändert



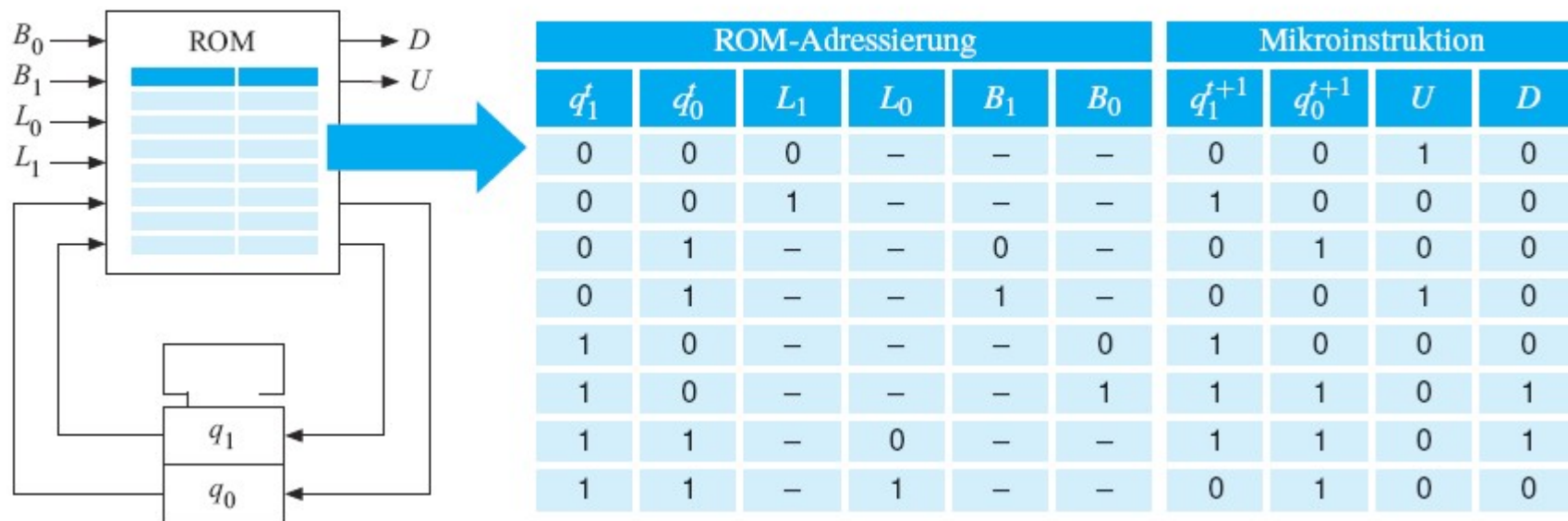
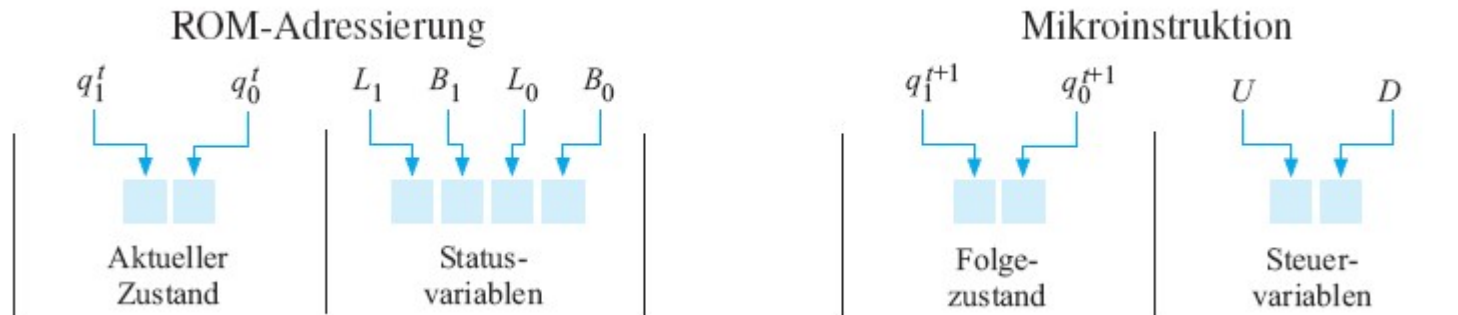
Beispiel: Fahrstuhlsteuerung (3)

- FSM als Mealy-Automat mit Ausgaben



Beispiel: Fahrstuhlsteuerung (4)

- Mikroinstruktionsformat**



Beispiel: Fahrstuhlsteuerung (5)

- **Bemerkung**

- bis hierhin ist ein mikroprogrammiertes Steuerwerk nur eine andere Sichtweise für einen Mealy-Automaten (mit ROM statt PLA)

- **Nachteil**

- Adressen bestehen aus 6 Bit
- Mikroprogramm besteht also aus $2^6 = 64$ Zeilen
- jedoch werden nur 4 verschiedene Mikroinstruktionen benötigt
 - viele Mehrfacheinträge durch die don't care Adressbits
 - 4 Ausgangsvariablen würden 16 verschiedene Mikroinstruktionen erlauben
- jede weitere Statusvariable verdoppelt die Größe des benötigten Mikroprogrammspeichers

- **Abhilfe**

- Kodierung der Eingänge (Statusvariablen und externe Steuersignale)
- Kodierung der Ausgänge (Steuersignale)
- adressmodifizierende Steuerwerke

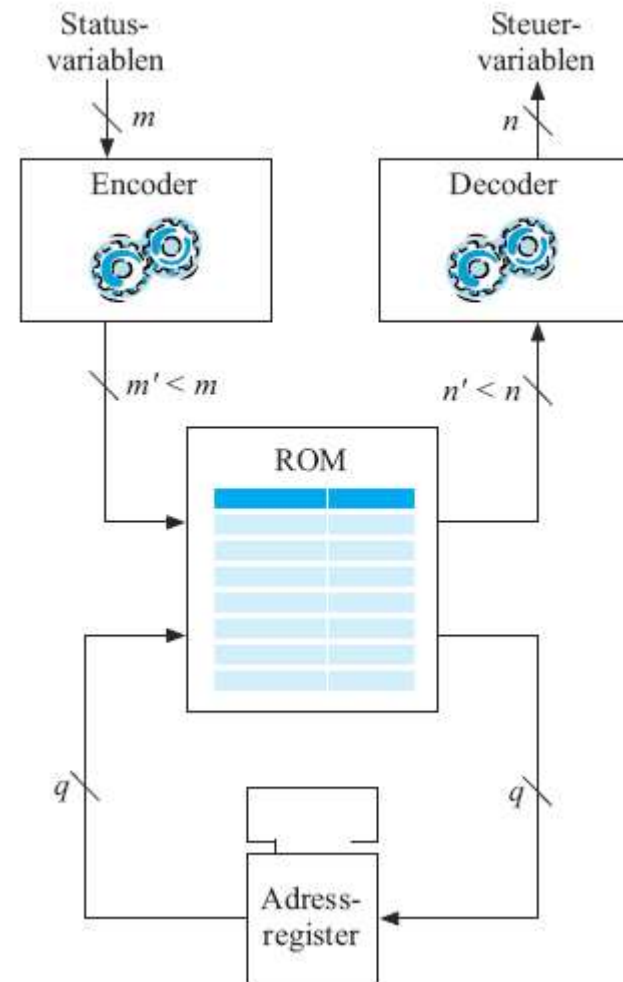
Kodierung der Eingänge und Ausgänge

- **Kodierung der Eingänge**

- nicht alle Eingangsbelegungen kommen vor
- Signale sind korreliert
 - z.B. können L_1 und L_0 niemals gleichzeitig 1 sein
- häufig können durch Kodierung Eingangssignale eingespart werden
 - jedes gesparte Signal halbiert den Speicheraufwand

- **Kodierung der Ausgänge**

- nicht alle möglichen Belegungen der Steuersignale werden benötigt
 - im Beispiel nur 4 von 16 möglichen
- Ausgangsbits können durch Kodierung eingespart werden



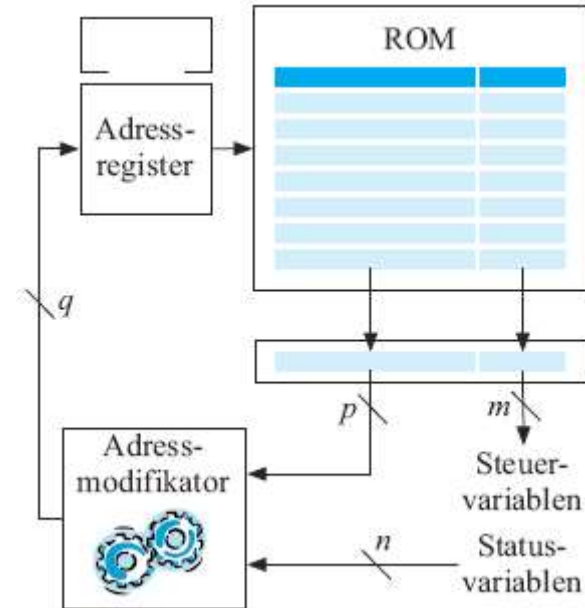
Kodierung der Eingänge und Ausgänge (2)

- Encoder und Decoder sind normale Schaltnetze
- Flexibilität geht verloren
 - z.B. könnten sich wegoptimierte Steuersignalbelegungen nachträglich als relevant für neu zu implementierende Operationen herausstellen
- das Speicherplatzproblem kann zwar deutlich reduziert werden, bleibt aber prinzipiell erhalten
- **Beobachtung**
 - die FSM(Schaltwerk)-Übergangsbedingungen hängen häufig nur von wenigen Statusvariablen ab
 - je nach Zustand können das andere sein
 - die Belegungen der vielen anderen Zustandsvariablen ist irrelevant
 - das führt zu vielen Kopien der Mikroinstruktion
- **Abhilfe bieten die adressmodifizierenden Mikrosteuerwerke**

Adressmodifizierende Mikrosteuerwerke

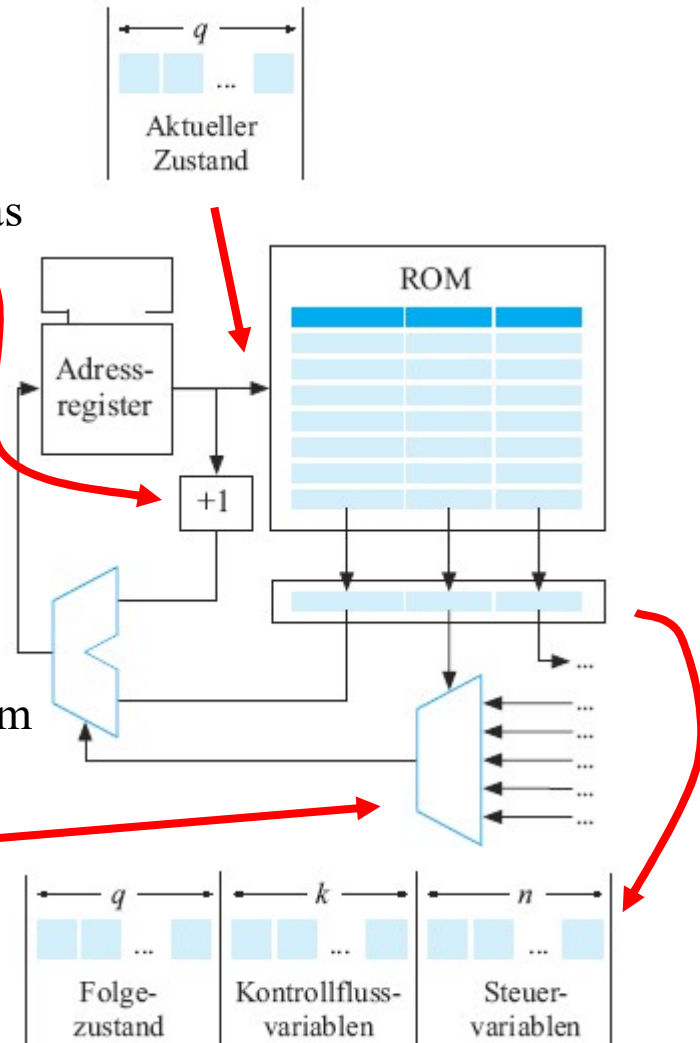
- **Grundidee**

- die Adressen des Mikroinstruktions-speichers beinhalten nicht mehr die Statusvariablen
 - die Statusvariablen beeinflussen die Mikroinstruktionsadresse nur indirekt über einen Adressmodifikator

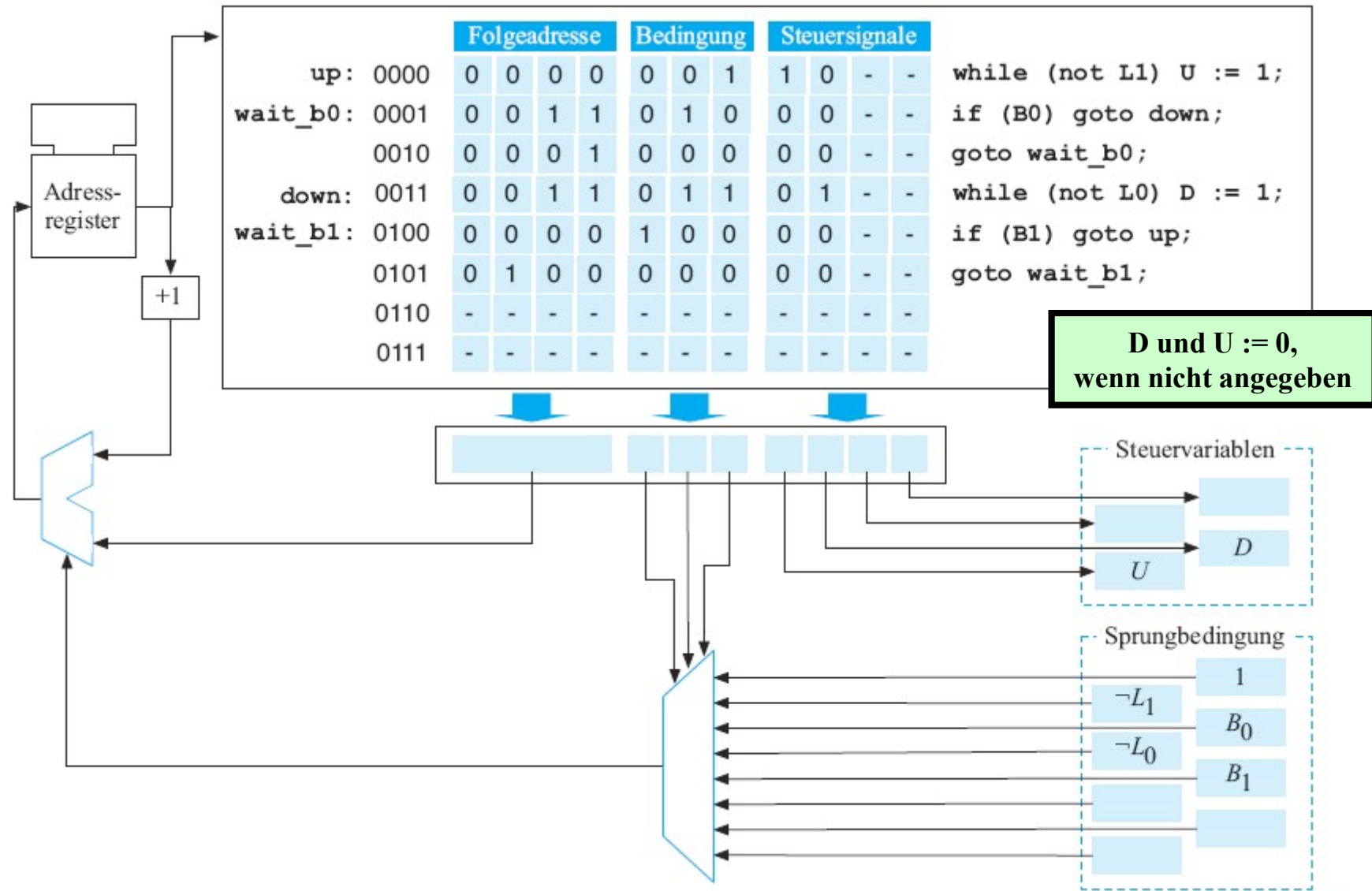


Adressmodifizierende Mikrosteuerwerke (2)

- **konkrete Ausführung**
 - Inkrementierer
 - (Addierer, der eine 1 addiert) sorgt für das Durchlaufen der normalen Reihenfolge durch das Mikroprogramm
 - Multiplexer
 - wählt die inkrementierte Mikroinstruktionsadresse, oder die in der Mikroinstruktion gespeicherte Folgeadresse
 - bedingter Sprung im Mikroprogramm
 - die Sprungbedingungen werden einem weiteren Multiplexer zugeführt
 - welche Sprungbedingung ausgewertet wird, legt die Mikroinstruktion fest
 - Kontrollflussvariablen

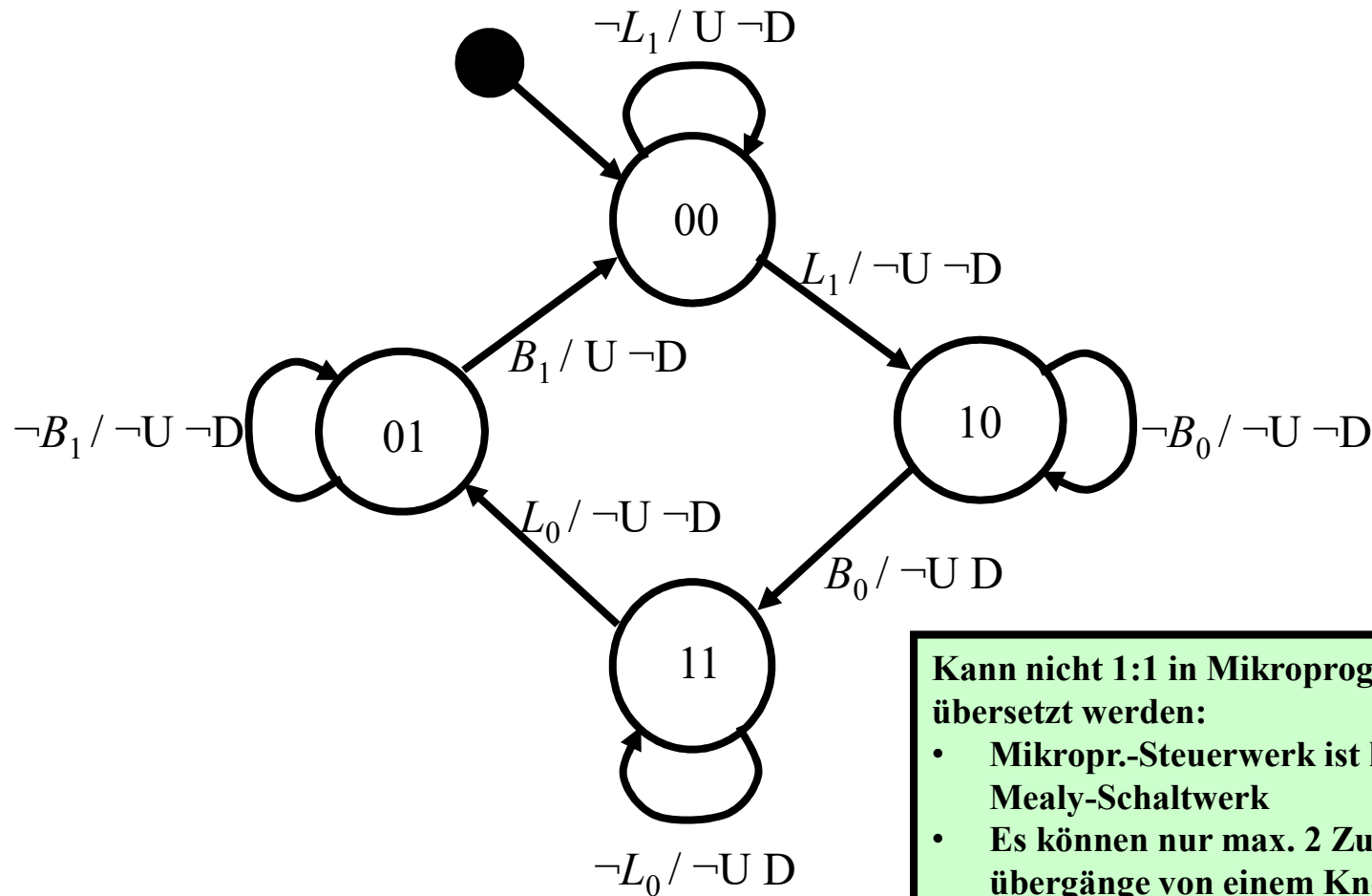


Adressmodifiz. Steuerwerk für Fahrstuhl



Zur Erinnerung

- Zustandsübergangsdiagramm



Kann nicht 1:1 in Mikroprogramm übersetzt werden:

- Mikropr.-Steuerwerk ist kein Mealy-Schaltwerk
- Es können nur max. 2 Zustandsübergänge von einem Knoten ausgehend realisiert werden

Adressmodifiz. Steuerwerk für Fahrstuhl (2)

- **Entwurf erfolgt in drei Schritten**

- Entwurf des Mikroprogramms
 - Mikroprogramm in Pseudocode
 - Labels zur Markierung eines Sprungziels
- Extraktion und Implementierung der Sprungbedingungen
 - Extraktion: not L1, B0, 1, not L0, B1
 - Verdrahtung des unteren Multiplexers
- Kodierung der Mikroinstruktionen

```

up:   while (not L1) U := 1;
wait_b0: if (B0) goto down;
        goto wait_b0;
down:  while (not L0) D := 1;
wait_b1: if (B1) goto up;
        goto wait_b1;
    
```



	Folgeadresse	Bedingung	Steuersignale	
up: 0000	0 0 0 0	0 0 1	1 0 - -	while (not L1) U := 1;
wait_b0: 0001	0 0 1 1	0 1 0	0 0 - -	if (B0) goto down;
0010	0 0 0 1	0 0 0	0 0 - -	goto wait_b0;
down: 0011	0 0 1 1	0 1 1	0 1 - -	while (not L0) D := 1;
wait_b1: 0100	0 0 0 0	1 0 0	0 0 - -	if (B1) goto up;
0101	0 1 0 0	0 0 0	0 0 - -	goto wait_b1;
0110	- - - -	- - -	- - - -	
0111	- - - -	- - -	- - - -	

Spezifikation vs. Implementierung

- **Spezifikation**
 - entweder Finite State Diagrams (Zustandsdiagramme)
oder
Microcode (Mikroprogramm)
 - beide sind sehr ähnlich
 - Zustand entspricht im Wesentlichen einer Programmzeile
 - unabhängig von den Details der Implementierung
 - ermöglicht uns, die Komplexität der Steuerung zu meistern
- **Implementierung**
 - ROM oder PLA
 - abhängig von der Technologie, die verwendet wird
 - Wahl ändert sich wahrscheinlich im Laufe der Zeit

Alternative Design Pfade

- zwischen Spezifikation und Implementierung gibt es noch weitere Designentscheidungen

