Rechnertechnik

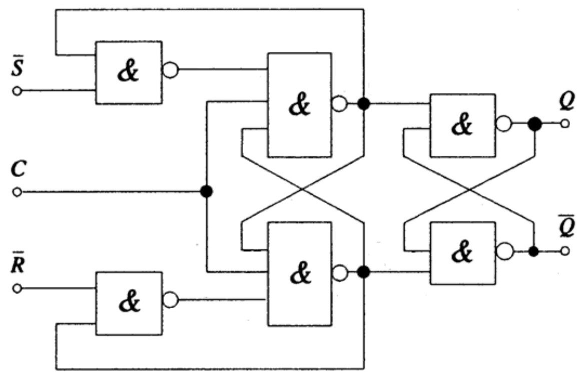
Schaltwerke

Prof. Dr. Alexander Metzner

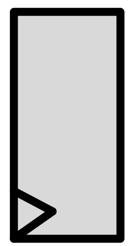
Schaltwerke

- Kombination Schaltnetze mit Kippgliedern = **Schaltwerke:**
  - Ausgabe der Schaltung hängt nicht nur von aktueller Eingabe ab, sondern auch von allen bisherigen Eingaben
  - Kippglieder **speichern** Eingaben der Vergangenheit
  - Sei $E(t_i)$ = Eingabe zum Zeitpunkt t_i, dann berechnet ein Schaltwerk zum Zeitpunkt t_n die Ausgabe $A(t_n)$ mit:

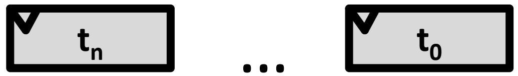
$$\mathbf{A(t_n) = f(E(t_n),E(t_{n-1}), E(t_{n-2}), \ldots , E(_0))}$$



=



Schaltwerke

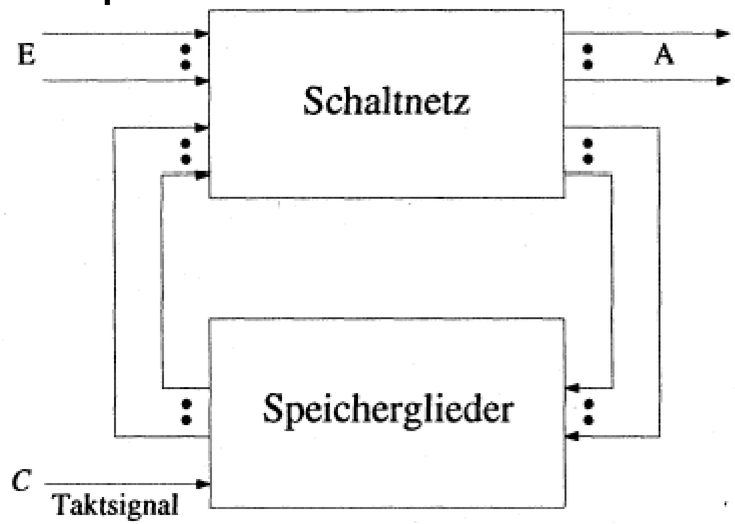
- Beobachtung: **Jede** digitale Eingabe benötigt zu **jedem** Zeitpunkt t_i einen **1-Bit**-Speicher: 

- Da Schaltungen unbegrenzt laufen, würde man **unendlich große Speicher** benötigen

➔Abstrahiere die Vergangenheit der Eingaben auf **endliche Menge von Zuständen**

– Übergangsfunktion berechnet den zu speichernden Zustand:

$$Z(t_n) = g(E(t_n),Z(t_{n-1}))$$
$$= g(E(t_n),g(E(t_{n-1}),Z(t_{n-2})))$$
$$= \ldots$$
$$A(t_n) = f(E(t_n),Z(t_{n-1}))$$



Endliche Automaten

- Zustände, die gespeichert werden und in andere Zustände transformiert werden, kennt man als endlicher Automat:

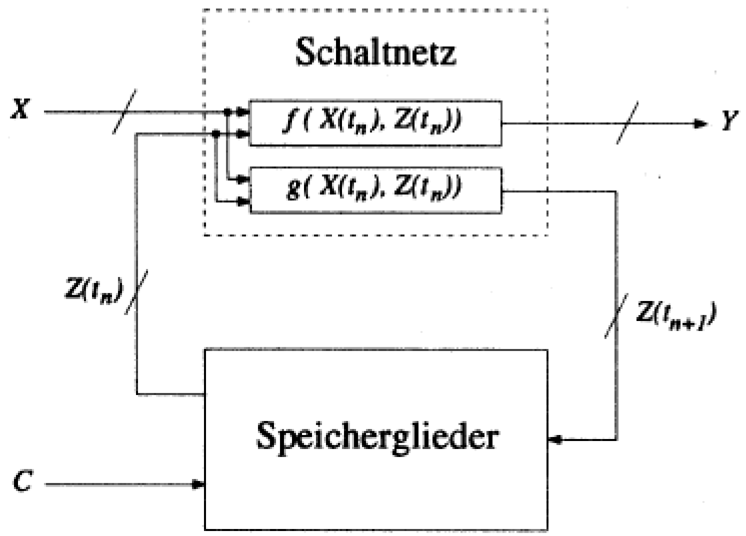
A = (X, Y, Z, z_0, F, g, f) mit:

$$
\begin{array}{rcll}
X & = & x_1, x_2, \ldots, x_n & \text{das Eingabealphabet} \\
Y & = & y_1, y_2, \ldots, y_m & \text{das Ausgabealphabet} \\
Z & = & z_1, z_2, \ldots, z_m & \text{die Zustandsmenge} \\
z_0 & \in & Z & \text{der Anfangszustand} \\
F & \subseteq & Z & \text{die Menge der Endzustände} \\
g : (x_i, z_j) & \rightarrow & z_k & \text{die Übergangsfunktion} \\
f : (x_i, z_j) & \rightarrow & y_r & \text{die Ausgangsfunktion}
\end{array}
$$

- Man unterscheidet 2 Arten von endlichen Automaten
  - **Mealy**-Automat und **Moore**-Automat

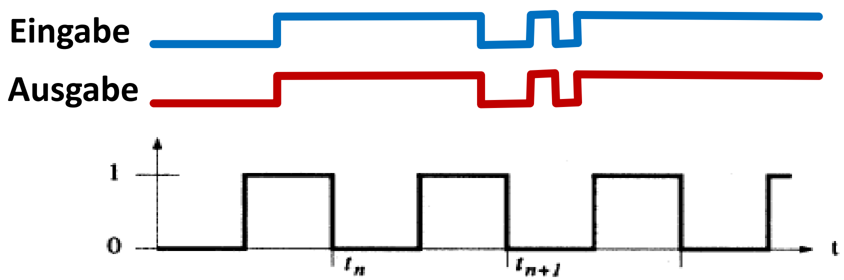
Mealy-Automat

- Mealy-Automaten reagieren in der Ausgabe **sofort auf sich verändernde Eingaben**, auch wenn sich der Zustand noch nicht geändert hat



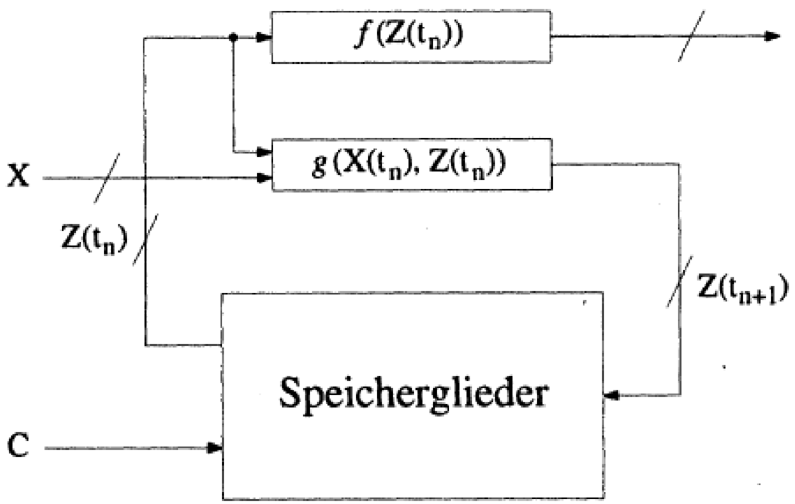
$$
\begin{aligned}
Y(t_n) &= f(X(t_n), Z(t_n)) && \text{Ausgangsfunktion} \\
Z(t_{n+1}) &= g(X(t_n), Z(t_n)) && \text{Übergangsfunktion}
\end{aligned}
$$

- Automaten sind stets getacktete Systeme:



Moore-Automat

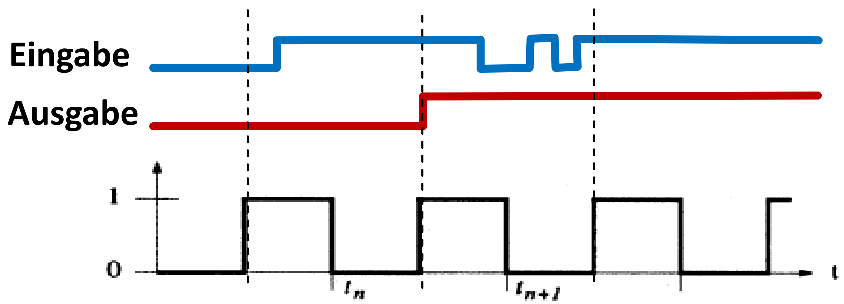
- Moore-Automaten hängen in ihrem Ausgabeverhalten **nur vom Zustand** ab (reagieren als auf Veränderungen der Eingabe erst **nach dem aktualisieren des Zustands**)



$$
\begin{aligned}
Y(t_n) &= f(Z(t_n)) && \text{Ausgangsfunktion} \\
Z(t_{n+1}) &= g(X(t_n), Z(t_n)) && \text{Übergangsfunktion}
\end{aligned}
$$

$$
\begin{aligned}
Y(t_{n+1}) &= f(Z(t_{n+1})) \\
&= f\,[g(X(t_n), Z(t_n))]
\end{aligned}
$$

- Automaten sind stets getacktete Systeme:



Funktionale Beschreibung & Analyse

Äquivalente Beschreibungsmöglichkeiten für Schaltwerke:

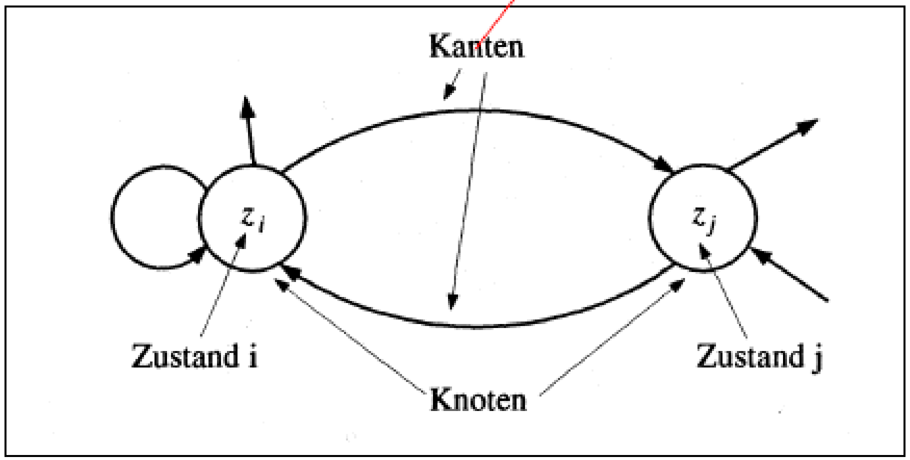
- Zustandfolgetabellen
- KV-Diagramme
- Schaltfunktionen (enthalten die Ausgabe- und Überangsfunktionen)
- Zustandsgraphen

Vereinfachte Notation des Zustandvektors:

$$
\begin{aligned}
Z(t_n) &=: Z \text{ in Komponenten } z_0, z_1, \cdots \\
Z(t_{n+1}) &=: Z^+ \text{ in Komponenten } z_0^+, z_1^+, \cdots
\end{aligned}
$$

mit Eingabe/Ausgabe

Zustandsgraph:



Zustandfolgetabelle:

Eingangsvariable						Ausgangsvariable					
z_0	$\cdots$	z_n	x_0	$\cdots$	x_i	z_0^+	$\cdots$	z_n^+	y_0	$\cdots$	y_j

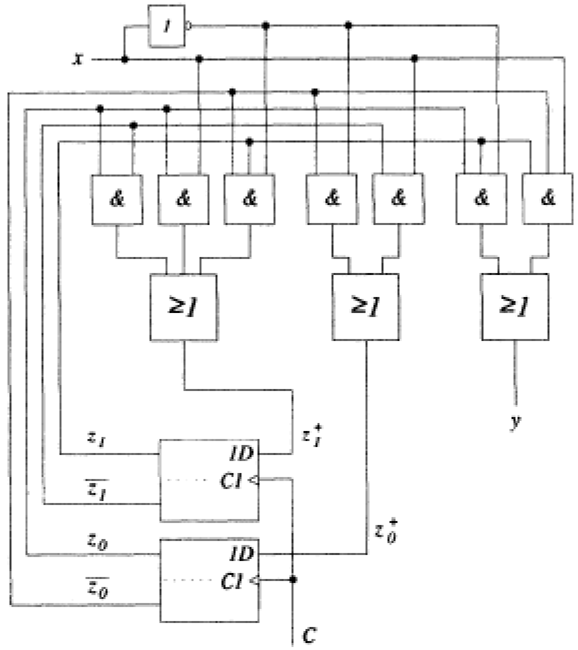
Realisierung von Schaltwerken

- Schrittweises Vorgehen ausgehend von verbaler Spezifikation des gewünschten Verhaltens:
  - Definition der Eingangs- und Ausgangsvariablen
  - Festlegung der Zustandsmenge und des Anfangszustand
  - Erstellung des Zustandsgraphen
  - Wahl einer Kodierung der Zustände, üblich sind:
    - Logarithmische Kodierung
    - Gray-Kodierung
    - One-Hot-Kodierung
  - Erstellung der Übergangsfunktion
  - Erstellen der Ausgabefunktion
  - DMF/KMF, Gatter-Implementierung mit Speicher

Reverse Engineering

- Das geht übrigens auch wieder anders herum:

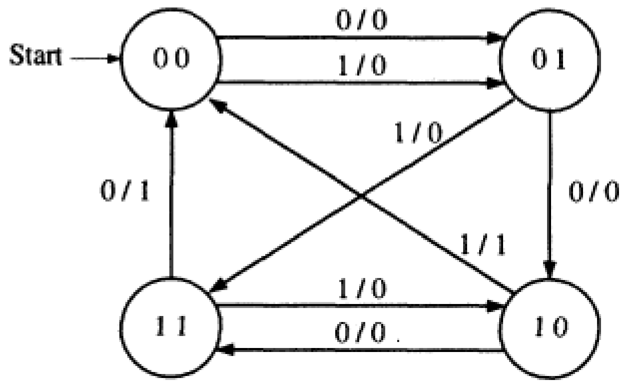
Schaltwerk:



Zustandsfolgetabelle:

z_1	z_0	x	z_1^+	z_0^+	y
0	0	0	0	1	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	1	1	0
1	0	1	0	0	1
1	1	0	0	0	1
1	1	1	1	0	0

Zustandsgraph:



Übergangsfunktionen:

$$z_0^+ = (\overline{z}_0 \wedge \overline{x}) \vee (\overline{z}_1 \wedge x)$$
$$z_1^+ = (z_0 \wedge \overline{z}_1) \vee (z_0 \wedge x) \vee (\overline{z}_0 \wedge z_1 \wedge \overline{x})$$

Ausgabefunktion:

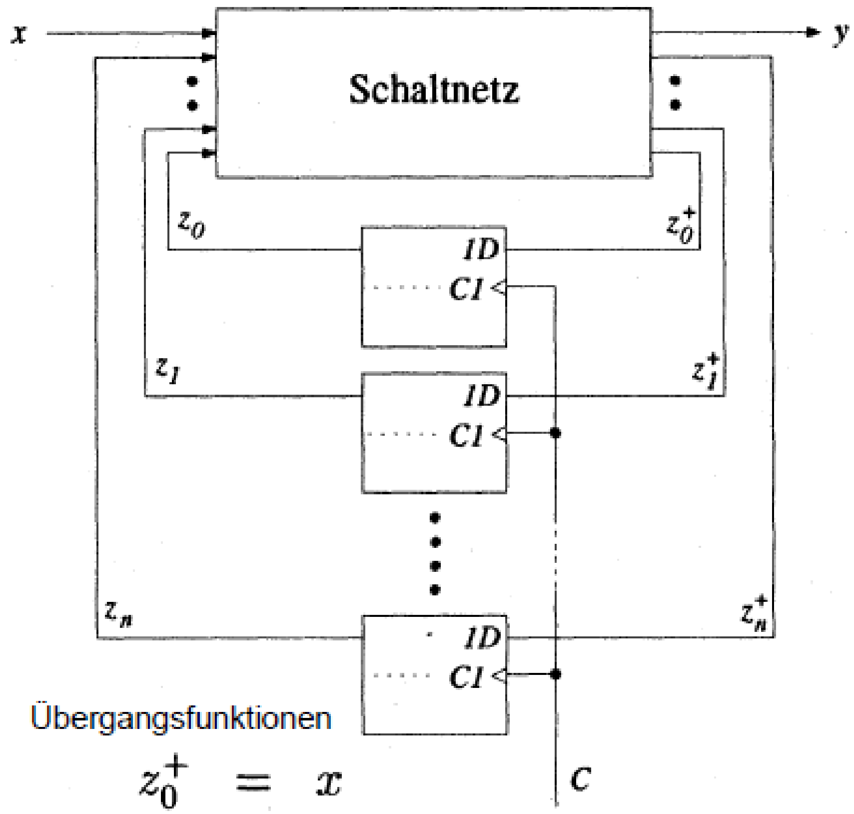
$$y = (z_0 \wedge z_1 \wedge \overline{x}) \vee (\overline{z}_0 \wedge z_1 \wedge x)$$

Schieberegister

Zustandsfolgetabelle:

n.Takt	Zustände nach Takt n				
	x	z_0	z_1	z_2	y
0.	1	0	0	0	0
1.	1	1	0	0	0
2.	0	1	1	0	0
3.	0	0	1	1	0
4.	0	0	0	1	1
5.	0	0	0	0	1
6.	0	0	0	0	0

x	z_2	z_1	z_0	z_2^+	z_1^+	z_0^+	y
0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0
0	0	1	0	1	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1
0	1	1	1	1	1	0	1
1	0	0	0	0	0	1	0
1	0	0	1	0	1	1	0
1	0	1	0	1	0	1	0
1	0	1	1	1	1	1	0
1	1	0	0	0	0	1	1
1	1	0	1	0	1	1	1
1	1	1	0	1	0	1	1
1	1	1	1	1	1	1	1



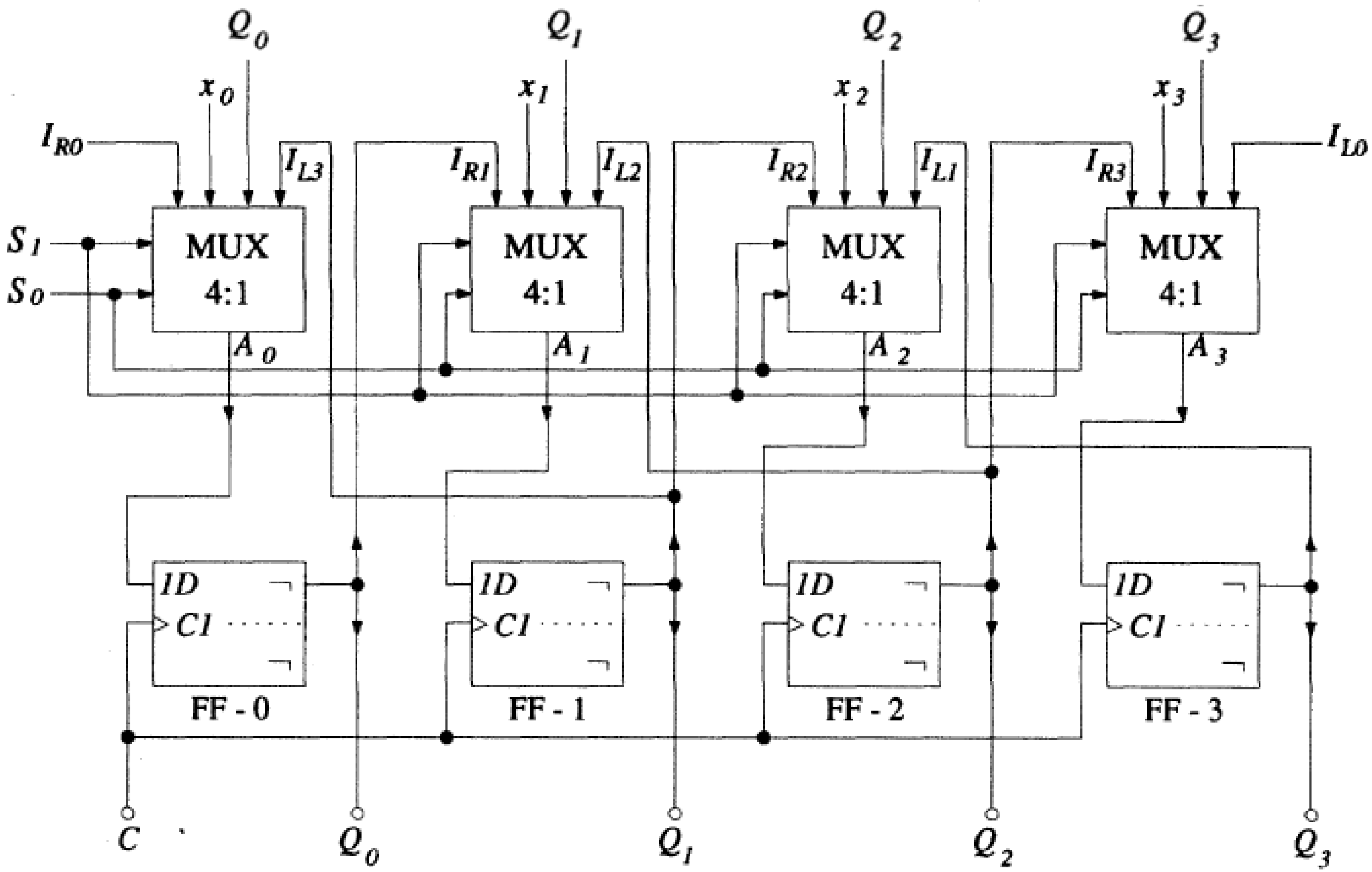
Übergangsfunktionen

$$z_0^+ = x$$
$$z_1^+ = z_0$$
$$z_2^+ = z_1$$

Ausgabefunktion:

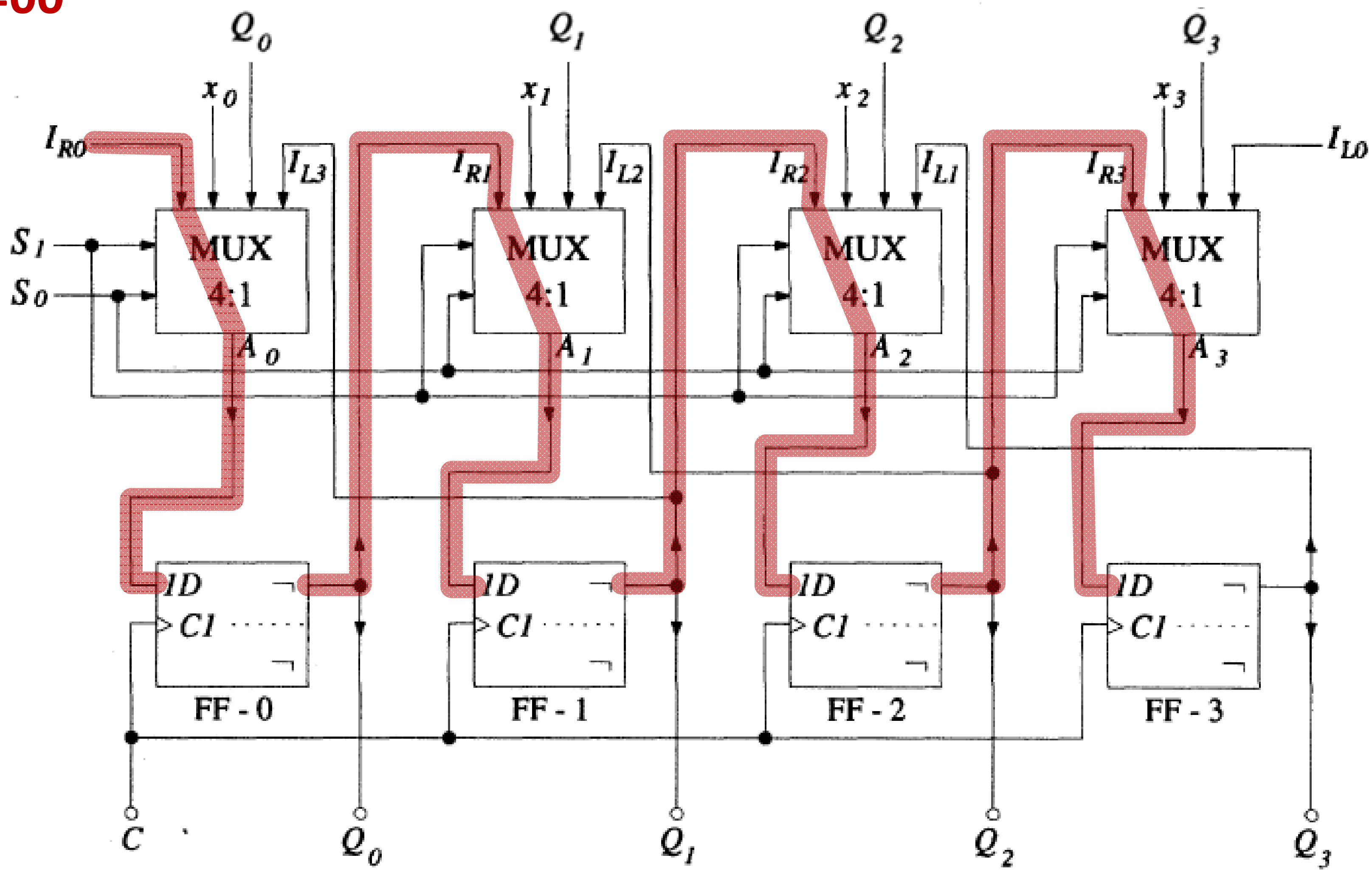
$$y = z_2$$

Mal etwas komplizierteres …



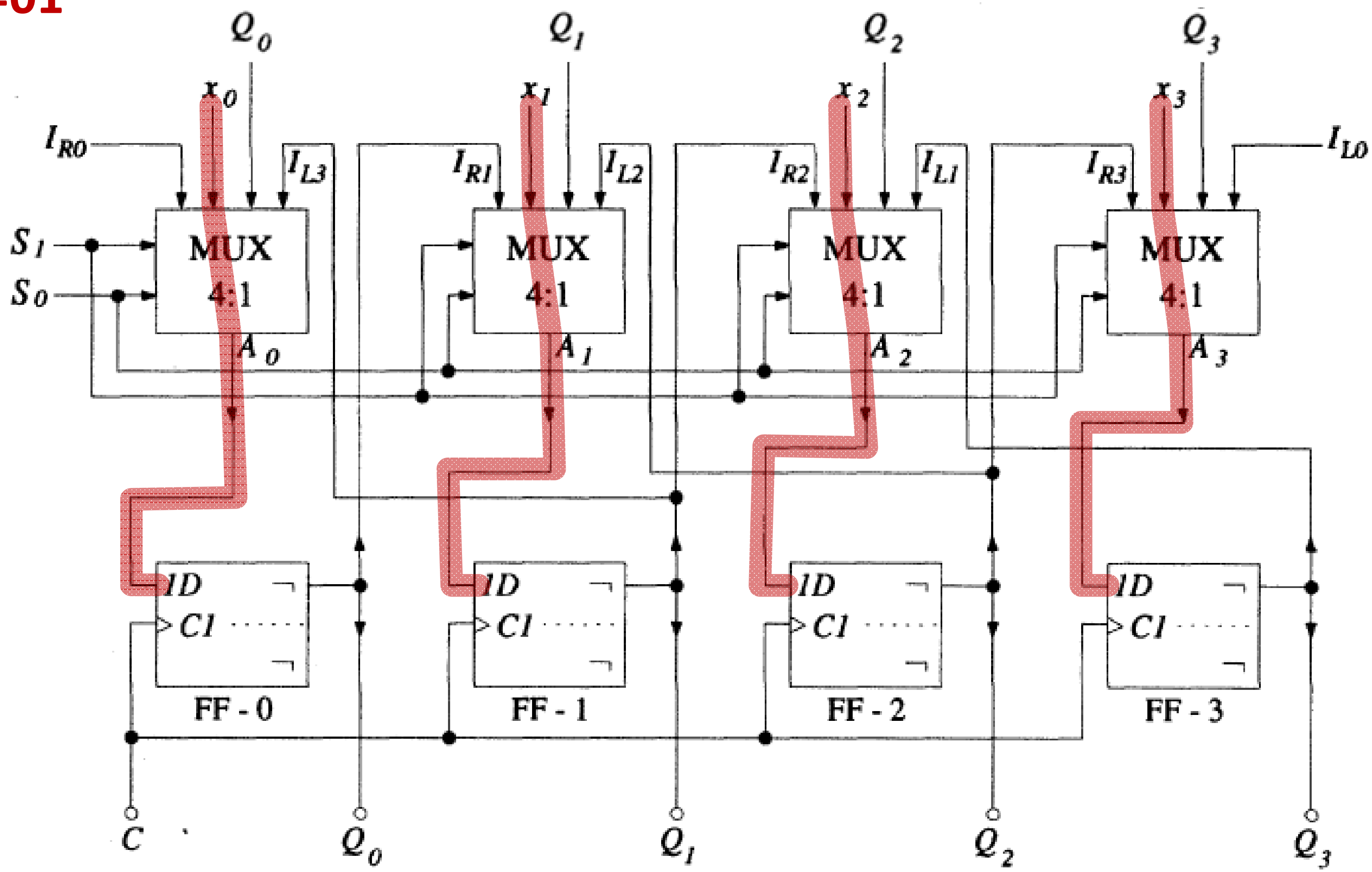
Mal etwas komplizierteres …

Sel=00



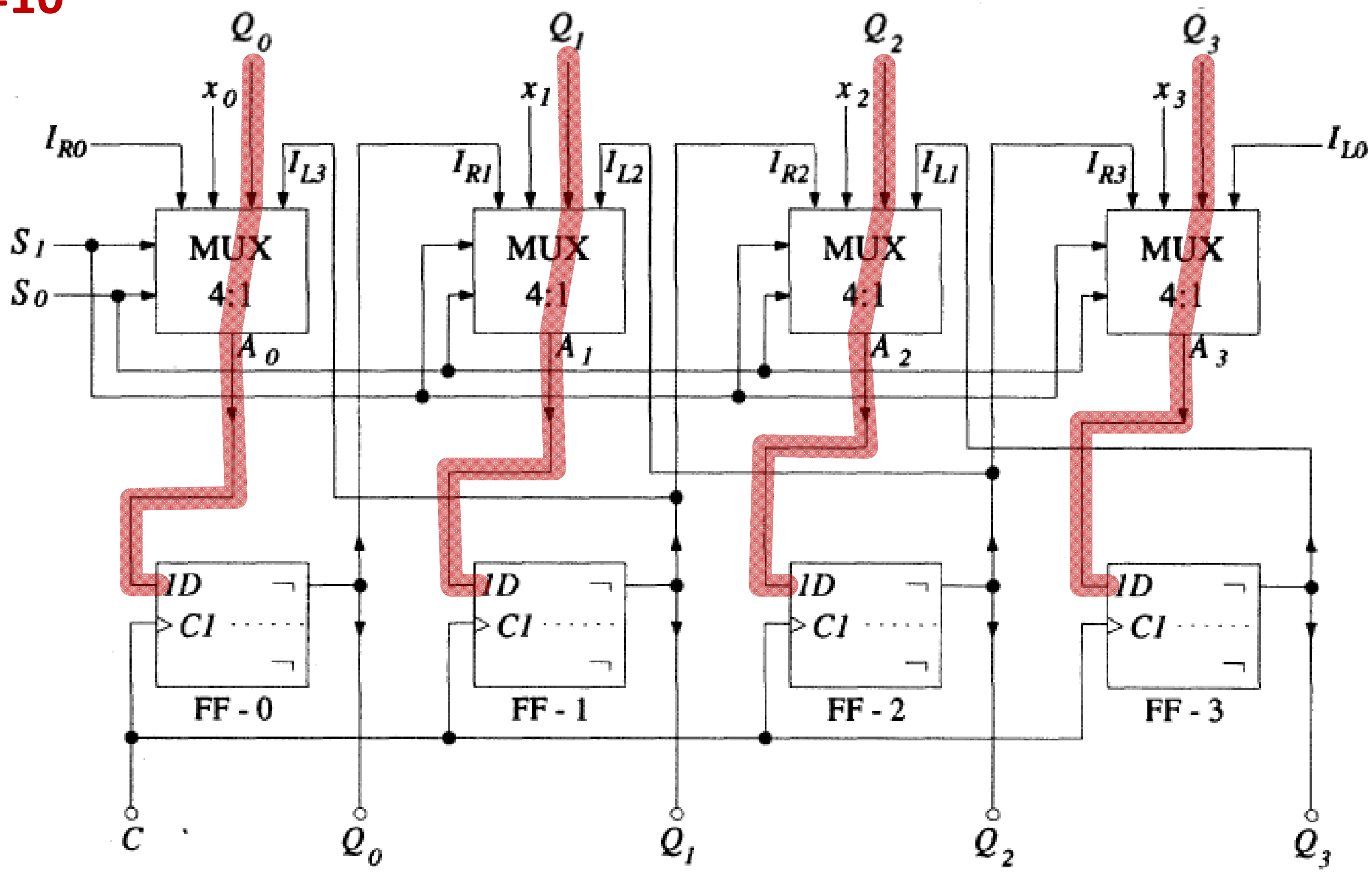
Mal etwas komplizierteres …

Sel=01



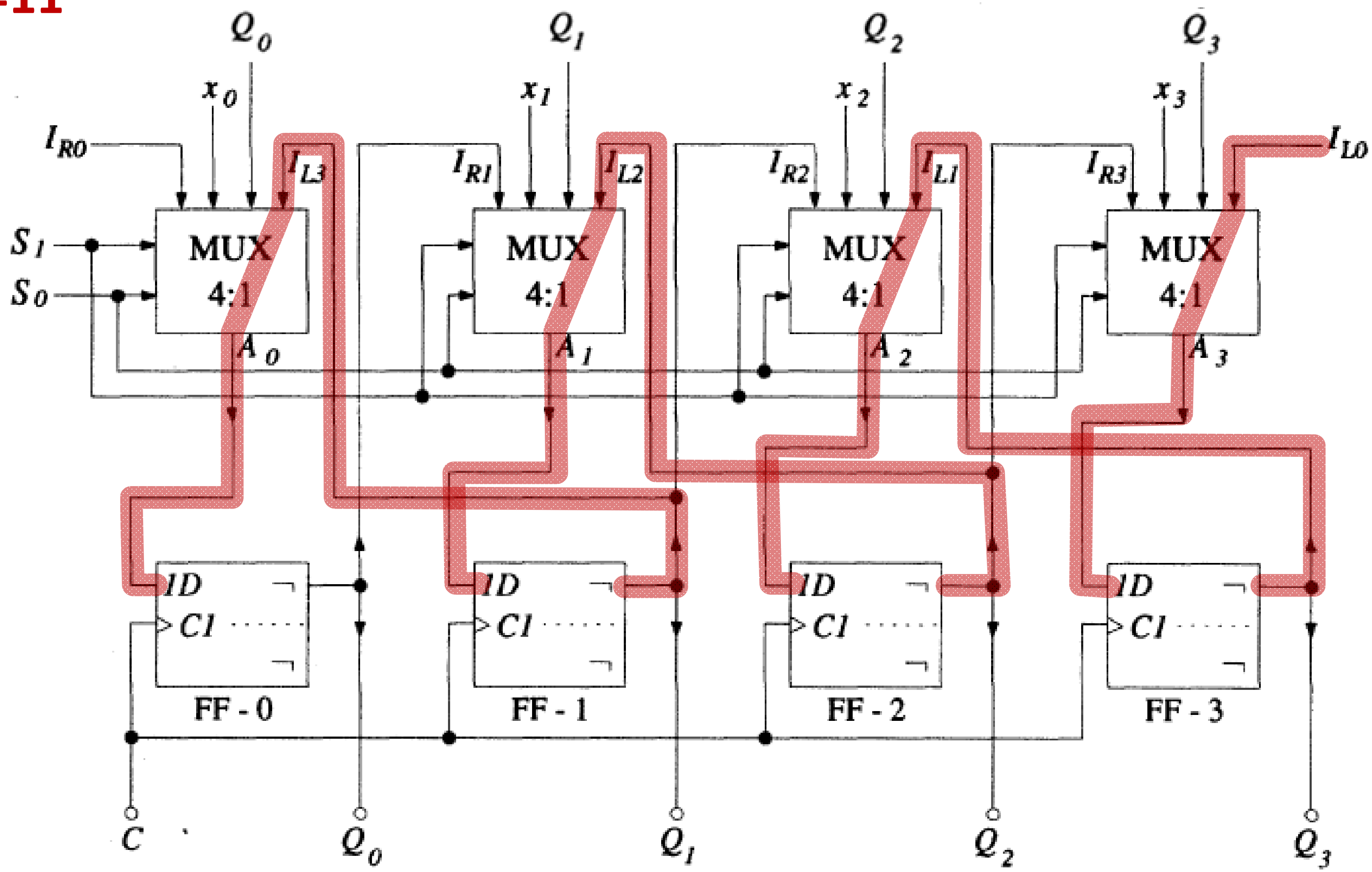
Mal etwas komplizierteres …

Sel=10



Mal etwas komplizierteres …

Sel=11



Schaltwerke: Anwendungsbeispiel

Ein einfacher Wechselautomat soll 1.-DM – und 2.-DM-Münzen in 10-Pfennig-Münzen wechseln. In einem Wechselvorgang können bis zu zwei Mark umgetauscht werden. Die Ausgabe des Wechselgeldes erfolgt durch Drücken einer speziellen Wechseltaste.

Mögliche Eingaben:

Einwurf 1DM
Einwurf 2DM
Drücken der WT
Keine Eingabe

Mögliche Ausgaben:

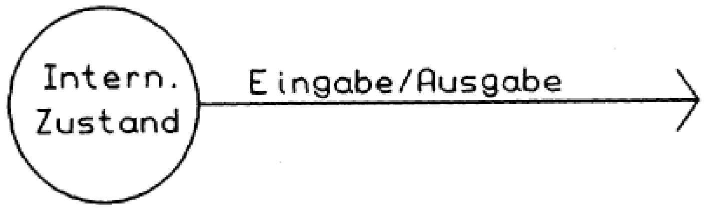
Zehn 10Pfennige
Zwanzig 10Pfennige
1 DM
2 DM
Keine Ausgabe

Interne Zustände:

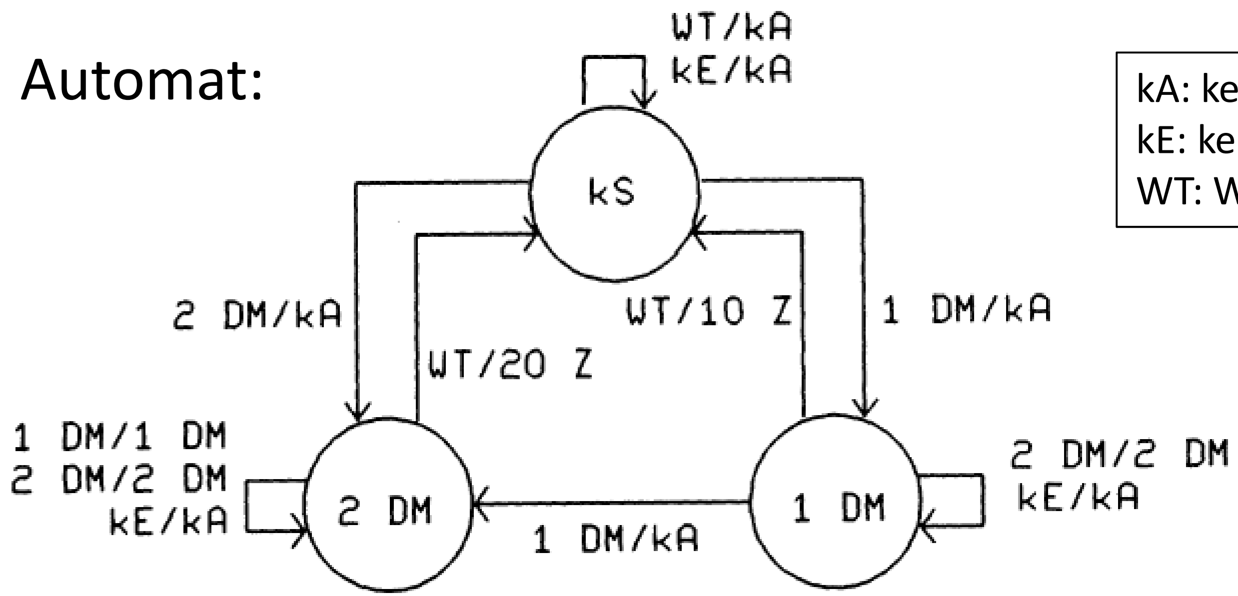
1 DM ist eingeworfen worden
2 DM sind eingegeben worden
Keine Schulden

Anwendungsbeispiel – Entwurf

- Zustandsgraph des Beispiels
  - Mealy-Automat:



  - Automat:



kA: keine Ausgabe
kE: keine Eingabe
WT: Wechseltaste

Anwendungsbeispiel – Kodierung

- Zustandsfolgetabelle:

Eingabe	Zustand	Folgezustand	Ausgabe
kE	kS	kS	kA
kE	1 DM	1 DM	kA
kE	2 DM	2 DM	kA
1 DM	kS	1 DM	kA
1 DM	1 DM	2 DM	kA
1 DM	2 DM	2 DM	1 DM
2 DM	kS	2 DM	kA
2 DM	1 DM	1 DM	2 DM
2 DM	2 DM	2 DM	2 DM
WT	kS	kS	kA
WT	1 DM	kS	10 Z
WT	2 DM	kS	20 Z

- Kodierung:

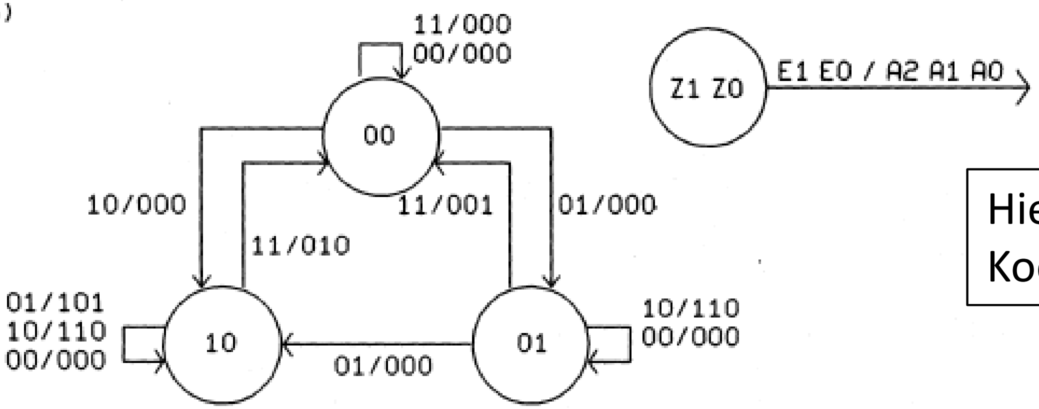
a)

Eingabe	E1	E0
kE	0	0
1 DM	0	1
2 DM	1	0
WT	1	1

Ausgabe	A2	A1	A0
kA	0	0	0
10 Z	0	0	1
20 Z	0	1	0
1 DM	1	0	1
2 DM	1	1	0

Int. Zustand	Z1	Z0
kS	0	0
1 DM	0	1
2 DM	1	0

b)



Hier überall logarithmische Kodierung gewählt

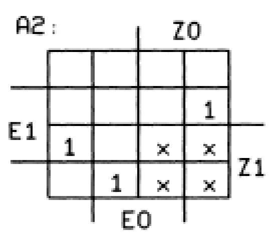
Anwendungsbeispiel – DMF bestimmen

Zustandsfolgetabelle:

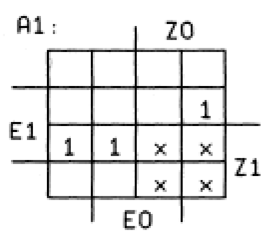
E1	E0	Z1	Z0	Z1⁺	Z0⁺	A2	A1	A0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0
0	0	1	0	1	0	0	0	0
0	0	1	1	×	×	×	×	×
0	1	0	0	0	1	0	0	0
0	1	0	1	1	0	0	0	0
0	1	1	0	1	0	1	0	1
0	1	1	1	×	×	×	×	×
1	0	0	0	1	0	0	0	0
1	0	0	1	0	1	1	1	0
1	0	1	0	1	0	1	1	0
1	0	1	1	×	×	×	×	×
1	1	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	1
1	1	1	0	0	0	0	1	0
1	1	1	1	×	×	×	×	×

KV-Diagramme zur Bestimmung von Übergangs- und Ausgabefunktion

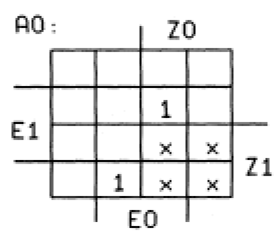
a)



$A2 = (E1 \wedge \overline{E0} \wedge Z1) \vee (\overline{E1} \wedge E0 \wedge Z1) \vee (E1 \wedge \overline{E0} \wedge Z0)$

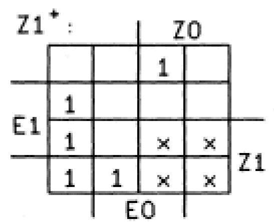


$A1 = (E1 \wedge Z1) \vee (E1 \wedge \overline{E0} \wedge Z0)$

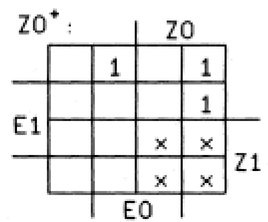


$A0 = (E1 \wedge E0 \wedge Z0) \vee (\overline{E1} \wedge E0 \wedge Z1)$

b)

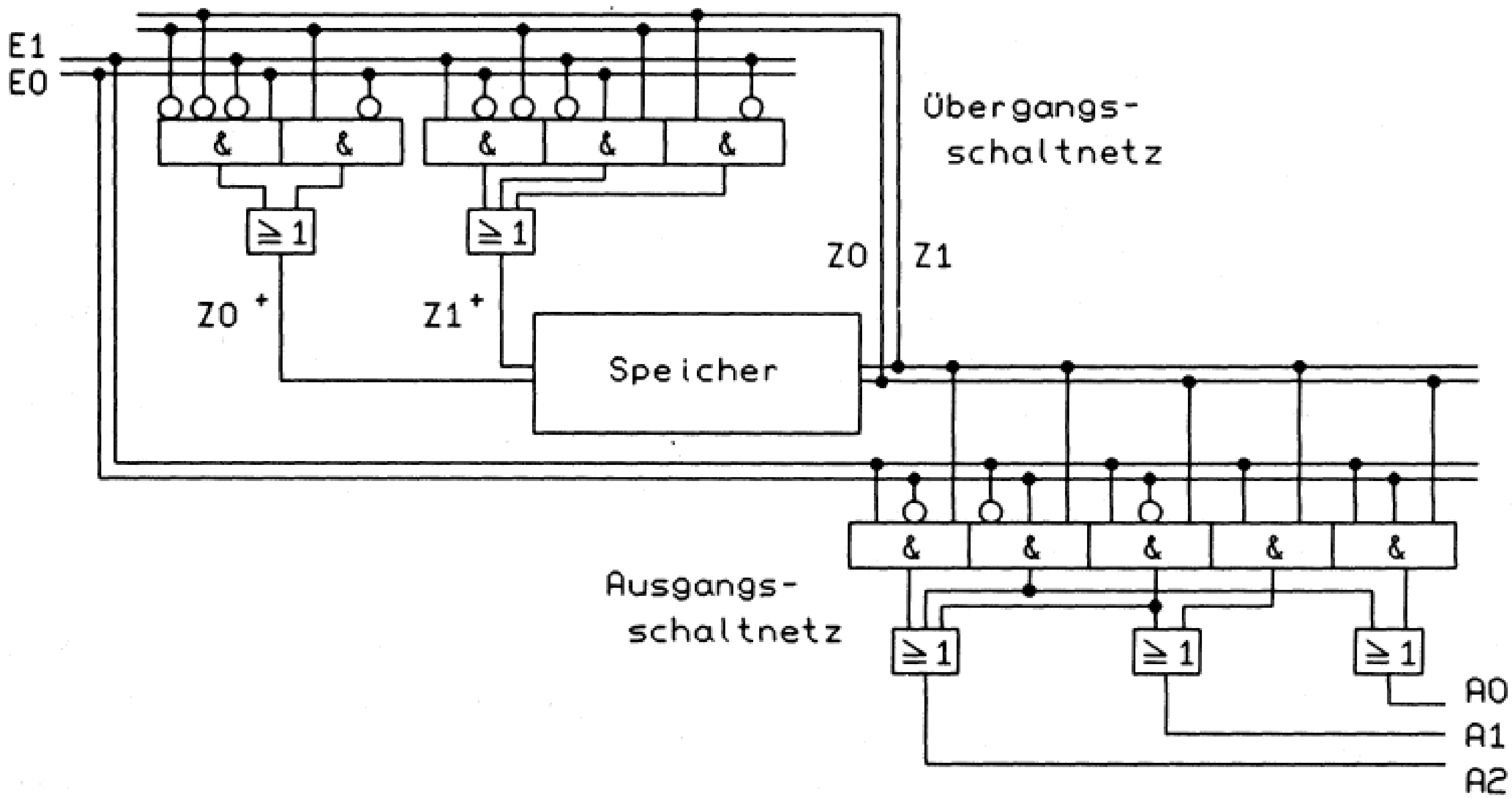


$Z1^{+} = (E1 \wedge \overline{E0} \wedge \overline{Z0}) \vee (\overline{E1} \wedge E0 \wedge Z0) \vee (\overline{E1} \wedge Z1)$



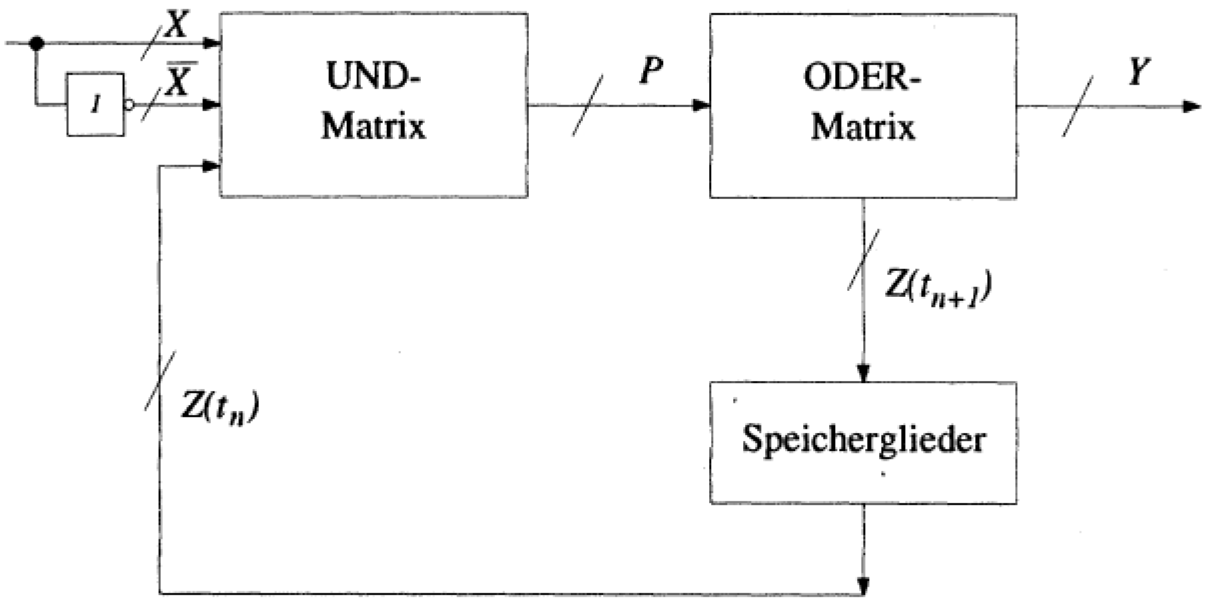
$Z0^{+} = (\overline{E0} \wedge Z0) \vee (\overline{E1} \wedge E0 \wedge \overline{Z1} \wedge \overline{Z0})$

Anwendungsbeispiel – Gatter-Implementierung



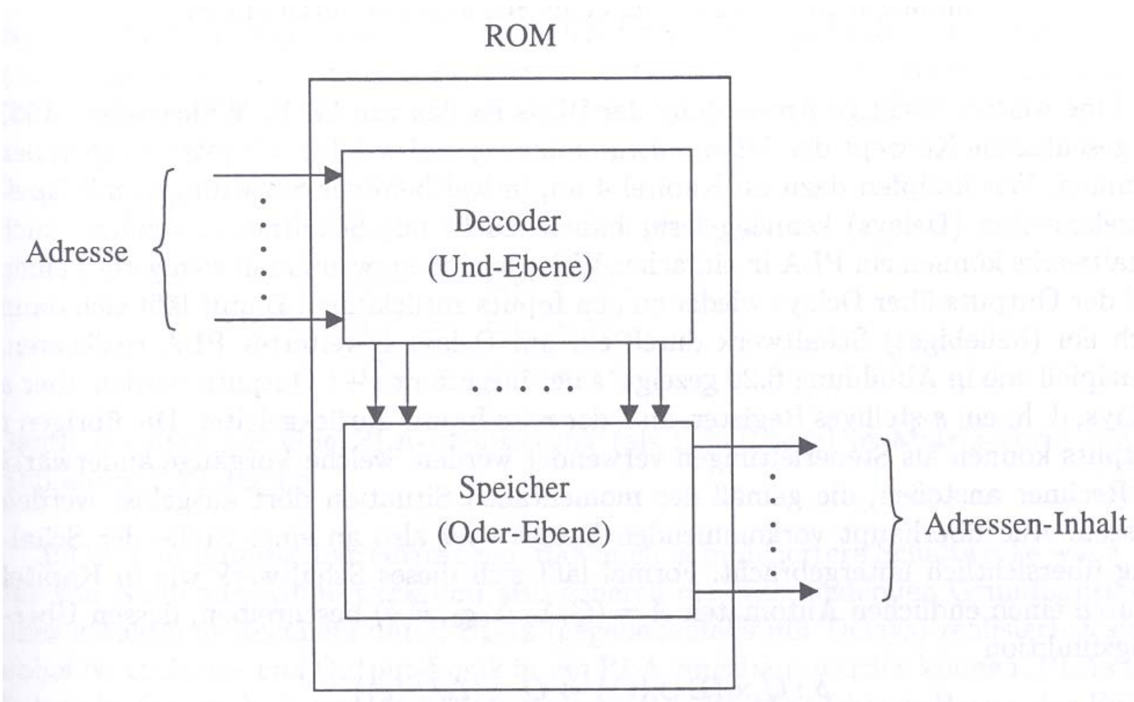
Anwendungsbeispiel – Implementierung auf programmierbaren Bausteinen

- Gemäß allg. Struktur der Bausteine mit Und-Matrix gefolgt von Oder-Matrix
- Alle Bausteine haben Register an den Ausgängen der Minterme



Anwendungsbeispiel – Implementierung in PROM

- Implementierung als Speicherbaustein, wobei jede Speicherzelle einer Zeile in der Wahrheitswerte-Tabelle entspricht
- **PROM** = **P**rogrammable **R**ead-**O**nly **M**emory



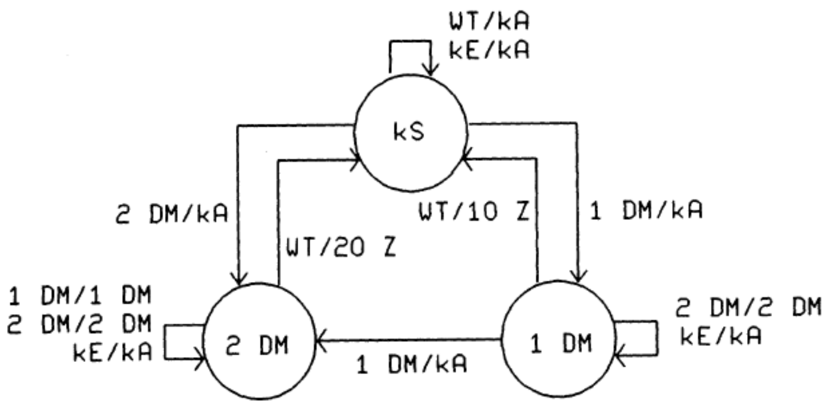
Beispielschaltung: siehe Tafel

Anwendungsbeispiel 2 – Erweiterung

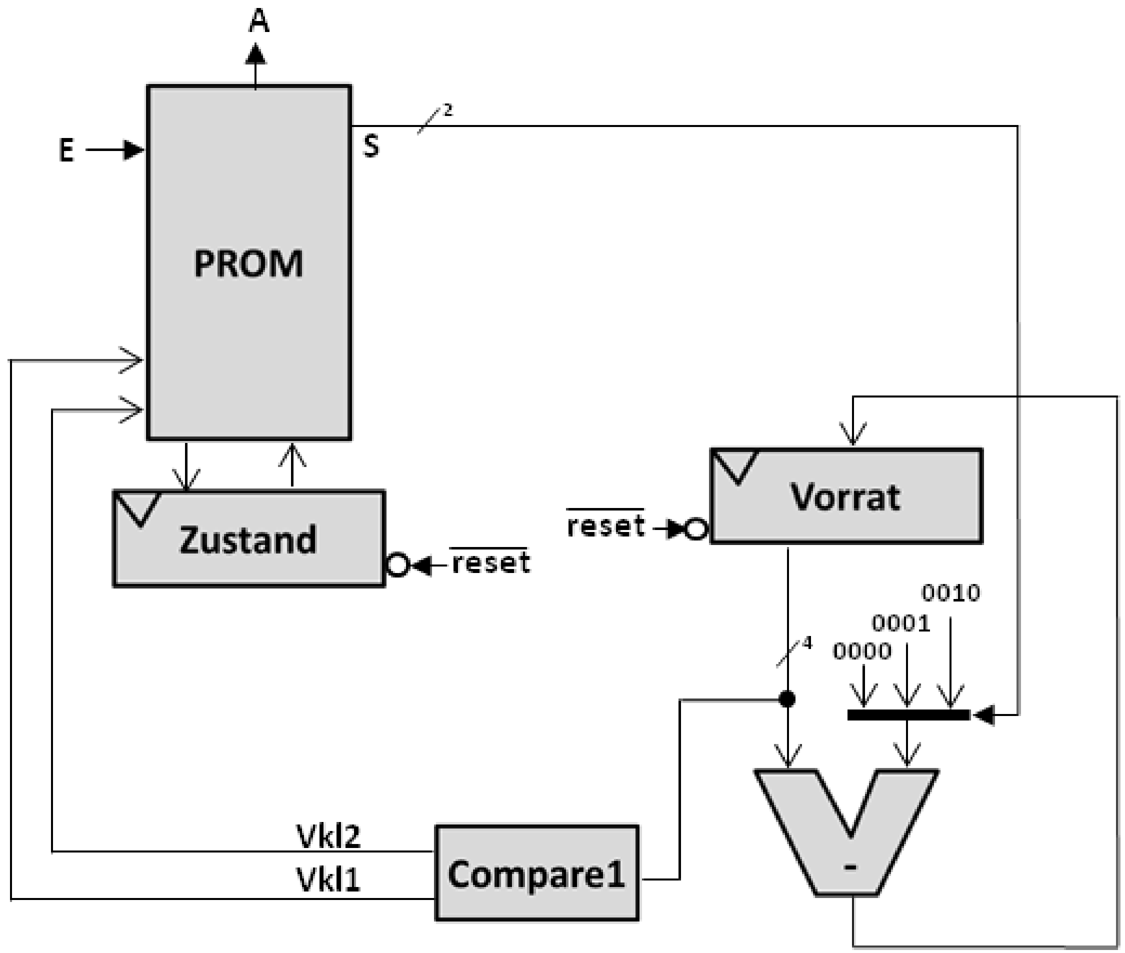
- Neue Anforderung an den Wechselgeldautomaten:

Der Automat verwaltet die Anzahl der noch vorhandenen 10er-Münzen. Er akzeptiert nur dann Geld, wenn genügend 10er Münzen zum Wechseln vorhanden sind. Auffüllen bei Reset des Systems.

- Annahme: Aufnahmekapazität von 150 10er-Münzen. Was passiert dann mit unserem Automaten?



Anwendungsbeispiel 2 – Schaltung



Eingabe	E1	E0
kE	0	0
1 DM	0	1
2 DM	1	0
WT	1	1

Ausgabe	A2	A1	A0
kA	0	0	0
10 Z	0	0	1
20 Z	0	1	0
1 DM	1	0	1
2 DM	1	1	0

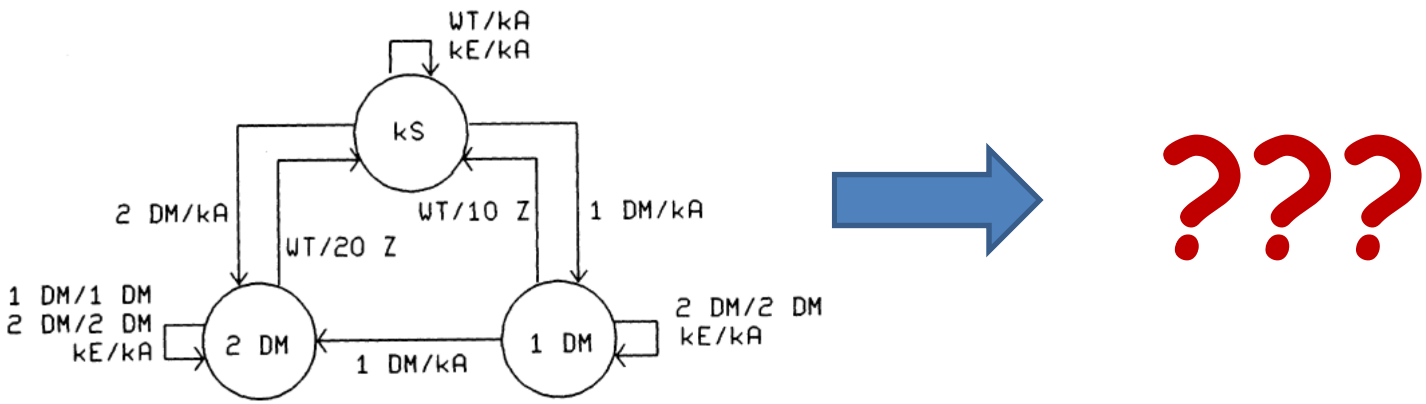
Int. Zustand	Z1	Z0
kS	0	0
1 DM	0	1
2 DM	1	0

Anwendungsbeispiel 3 – Erweiterung

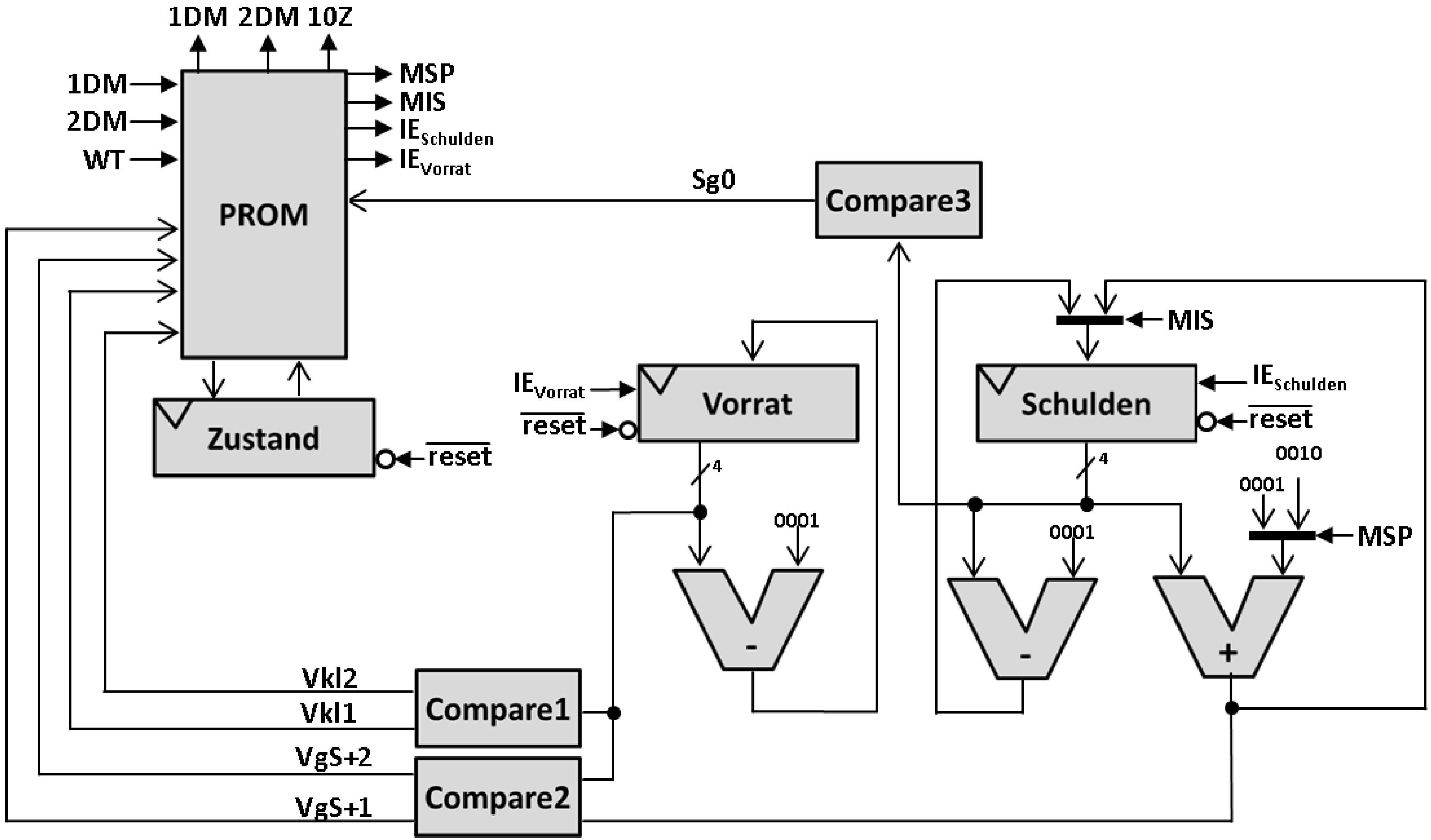
- Neue Anforderung an den Wechselgeldautomaten:

  Zusätzlich zu den bestehenden Anforderungen soll der Kunde vor dem Drücken der Wechseltaste so viele 1DM oder 2DM Münzen einwerfen dürfen, wie er möchte, aber maximal so viele, wie gewechselt werden können.

- Annahme: Aufnahmekapazität von 150 10er-Münzen. Was passiert dann mit unserem Automaten?



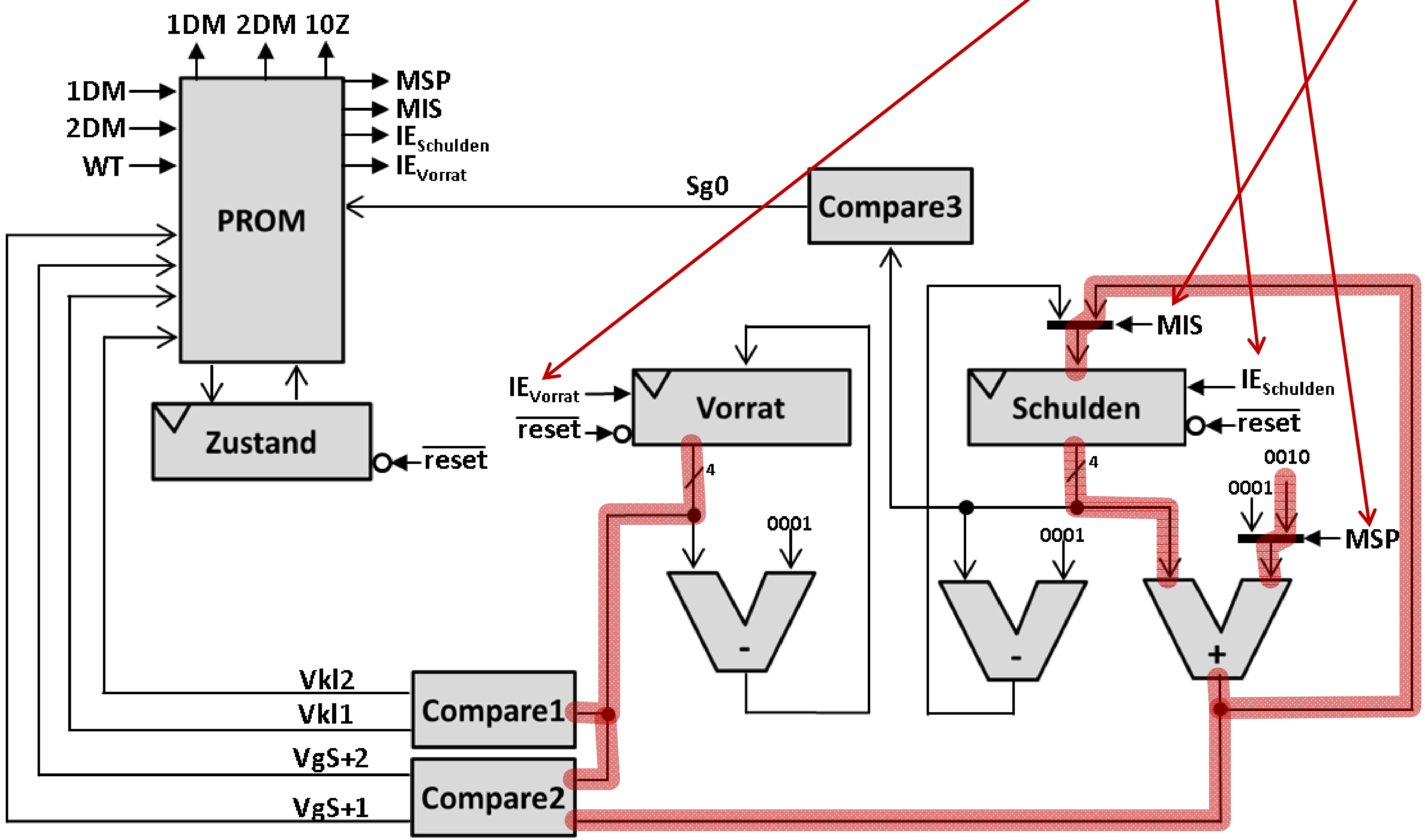
Anwendungsbeispiel 3 – Schaltung



Anwendungsbeispiel 3 – Schaltung

z_1	z_0	1DM	2DM	V<1	V<2	V>=S+1	V>=S+2	S>0	z^+_1	z^+_0	IE_V	IE_S	MSP	MIS
0	0	0	1	0	0	1	1	X	0	1	0	1	1	1

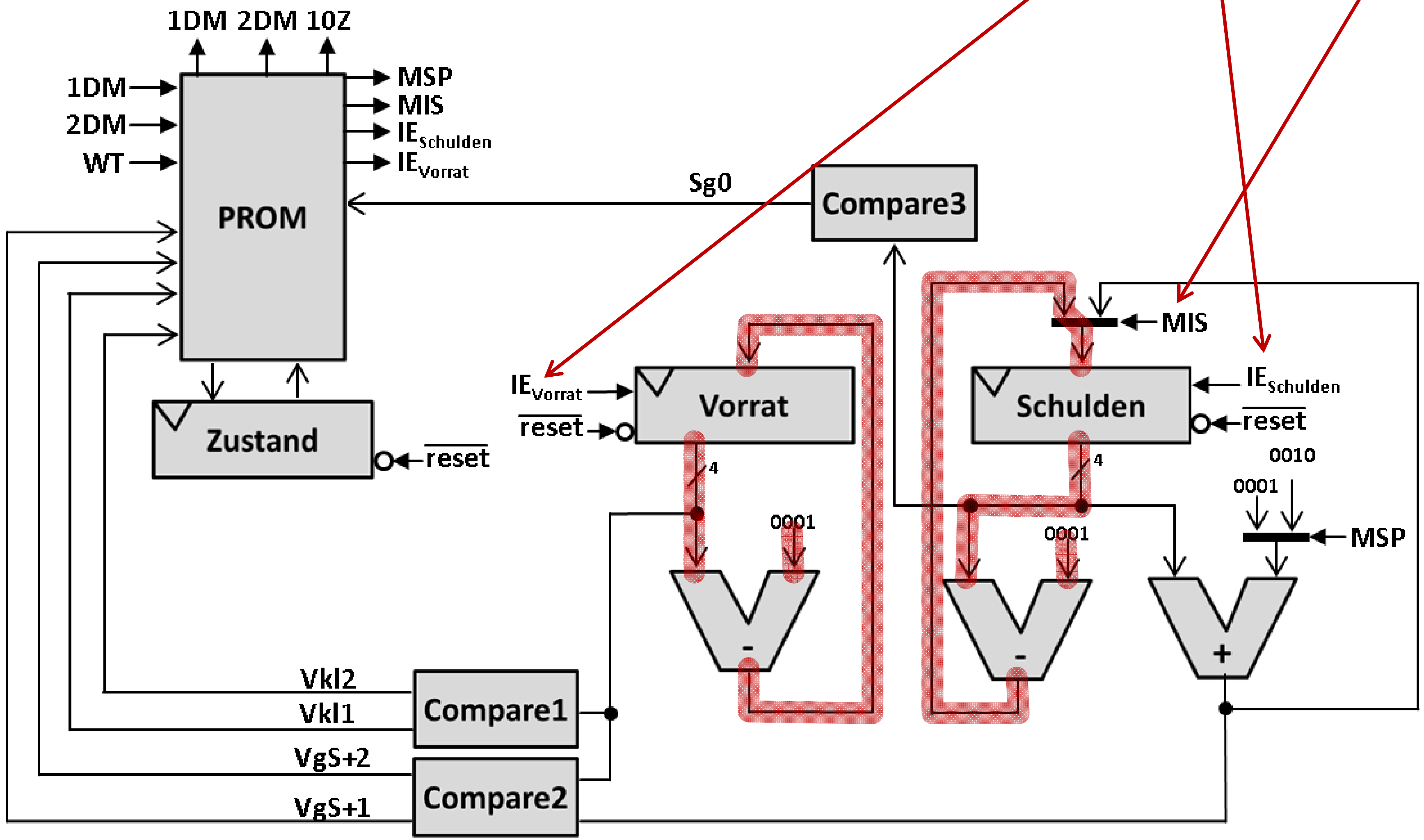
KS



Anwendungsbeispiel 3 – Schaltung

z_1	z_0	1DM	2DM	V<1	V<2	V>=S+1	V>=S+2	S>0	z^+_1	z^+_0	IE_v	IE_S	MSP	MIS
1	0	0	0	X	X	X	X	1	1	0	1	1	X	0

Auszahlung

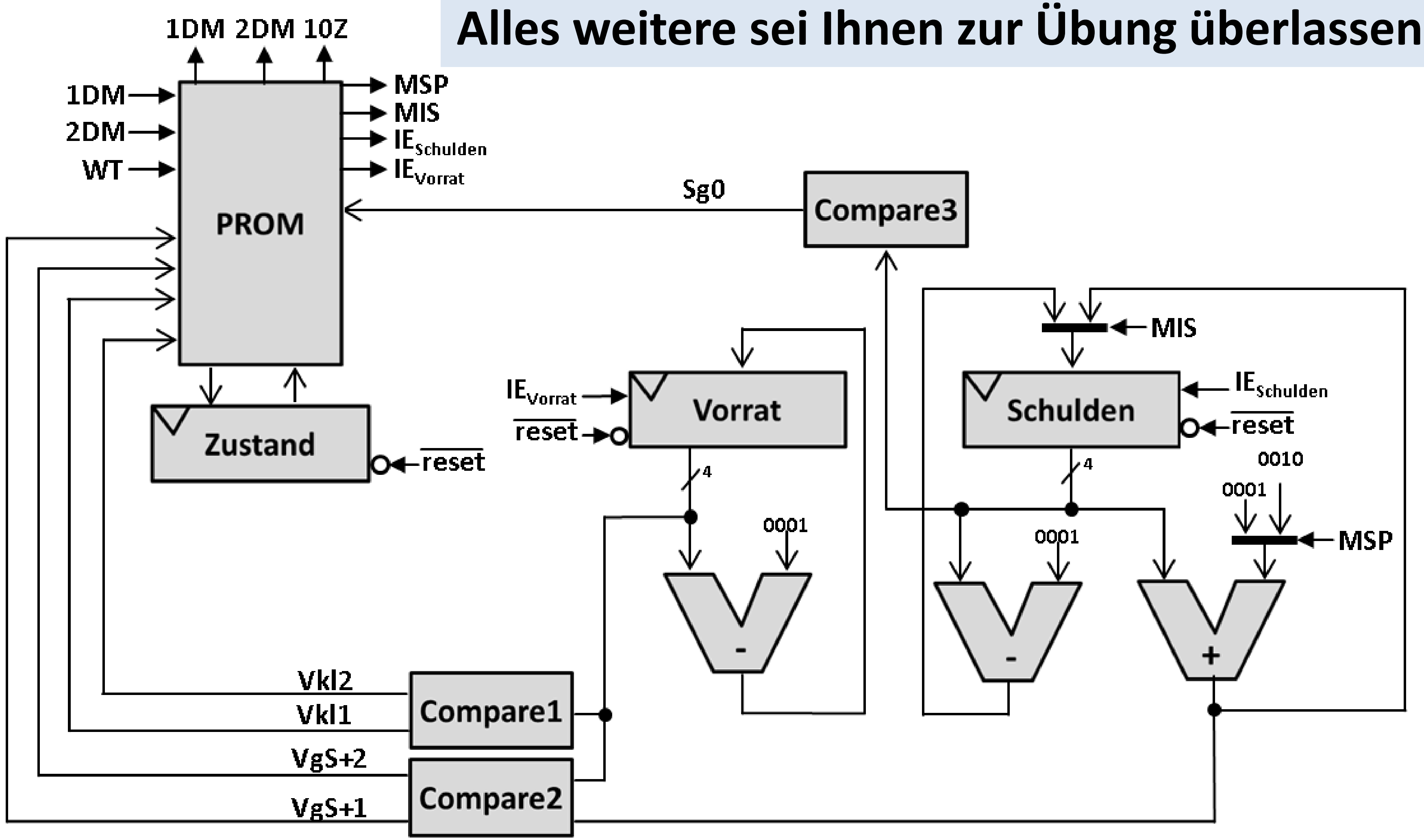


Anwendungsbeispiel 3 – Schaltung

z_1	z_0	1DM	2DM	V<1	V<2	V>=S+1	V>=S+2	S>0	z^+_1	z^+_0	IE_V	IE_S	MSP	MIS
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

???

Alles weitere sei Ihnen zur Übung überlassen



Anwendungsbeispiel – Beobachtung

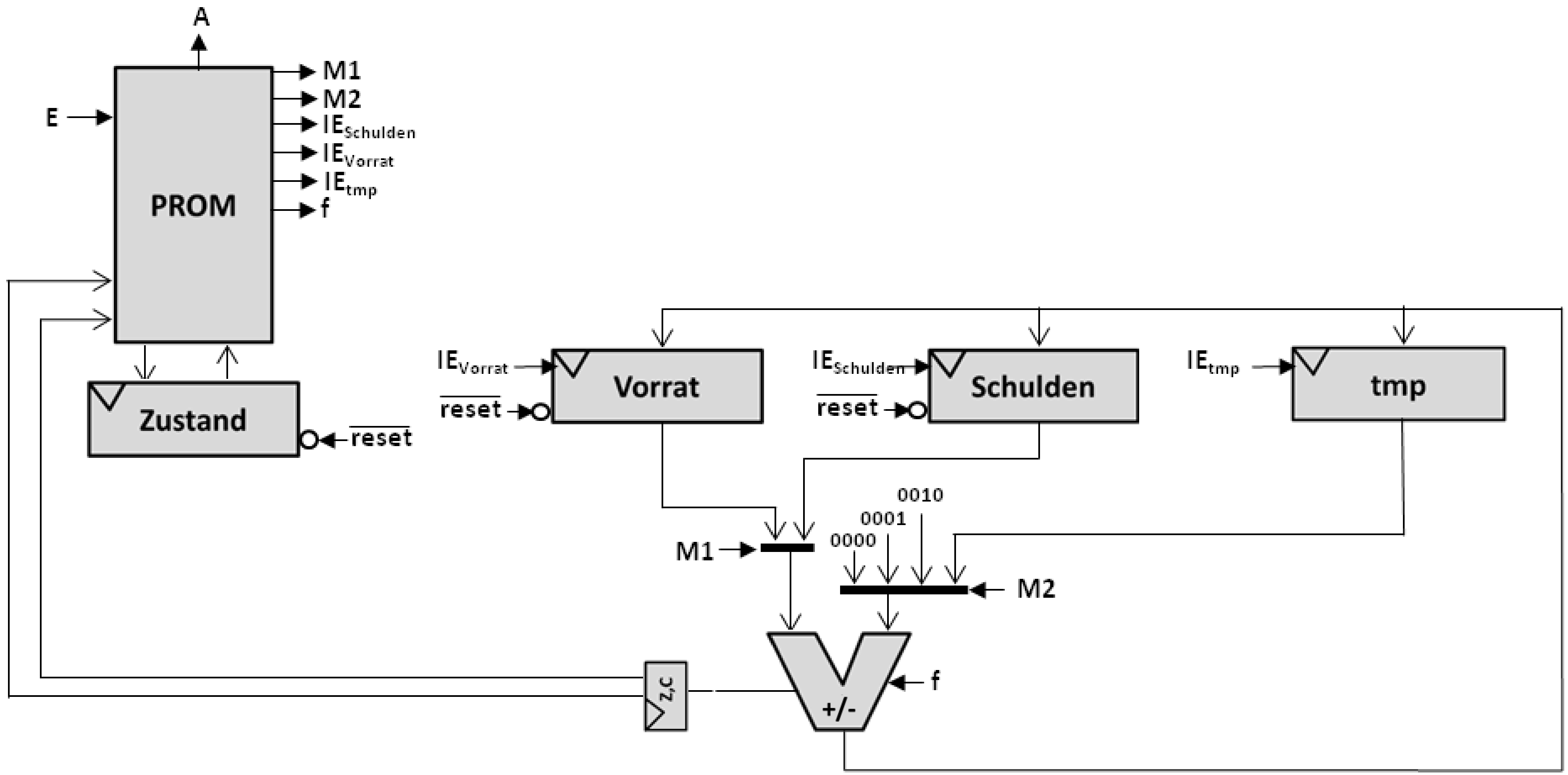
- Je mehr Anforderungen gestellt werden, desto komplizierter wird die Architektur
- Warum? Weil Architektur maximale Parallelität zur Verfügung stellen will

➔Einfachere Architektur wählen, die sequentielle Abläufe enthält, aber dafür weniger Teile
  - Hier Erinnerung Schaltnetze: Wähle einen Addierer/ Subtrahierer mit Funktionsauswahl *f*

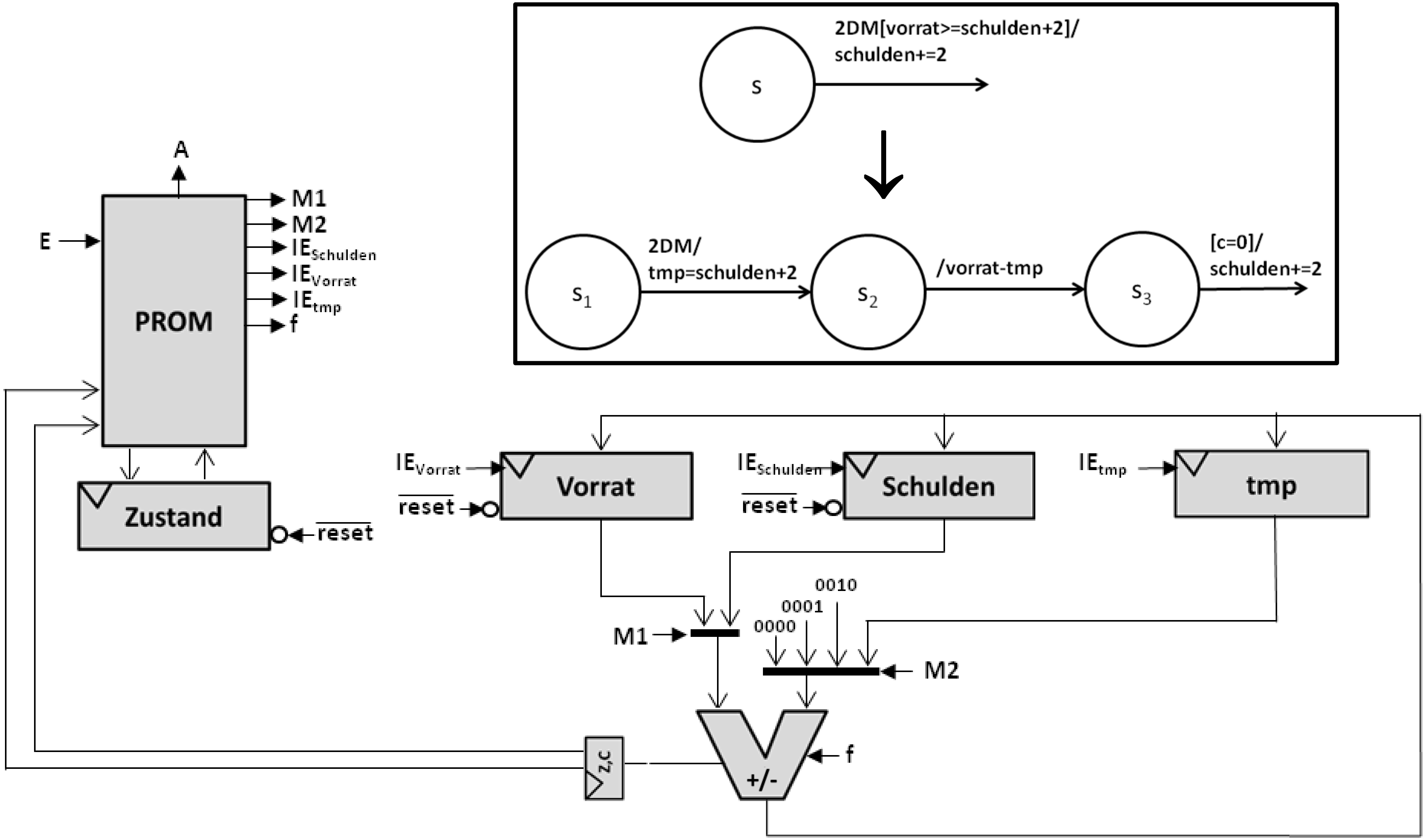
- Preis: Automat wird komplexer!

Anwendungsbeispiel – Einfache Architektur

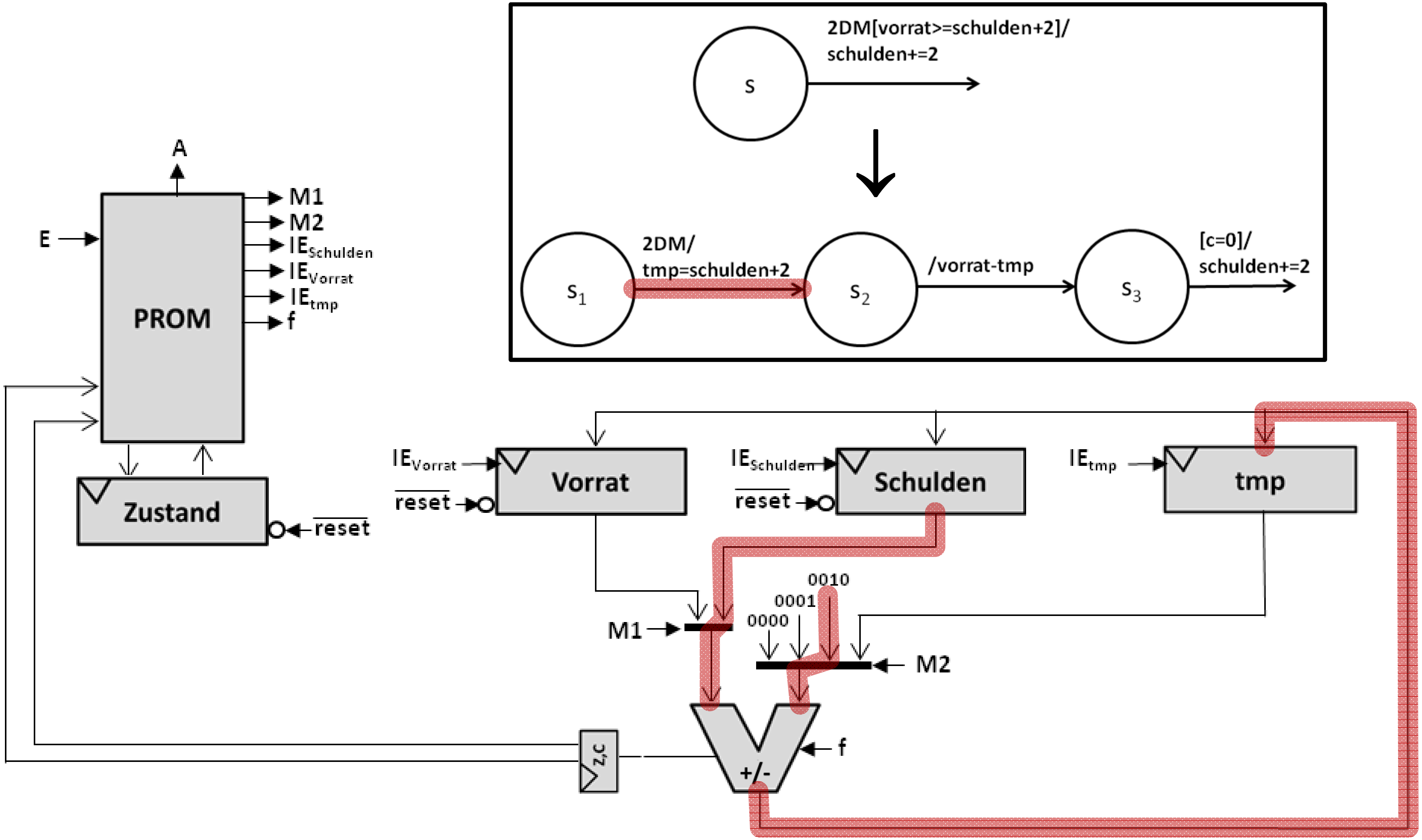
- Ein Addierer/Subtrahierer → Sequentielle Abarbeitung



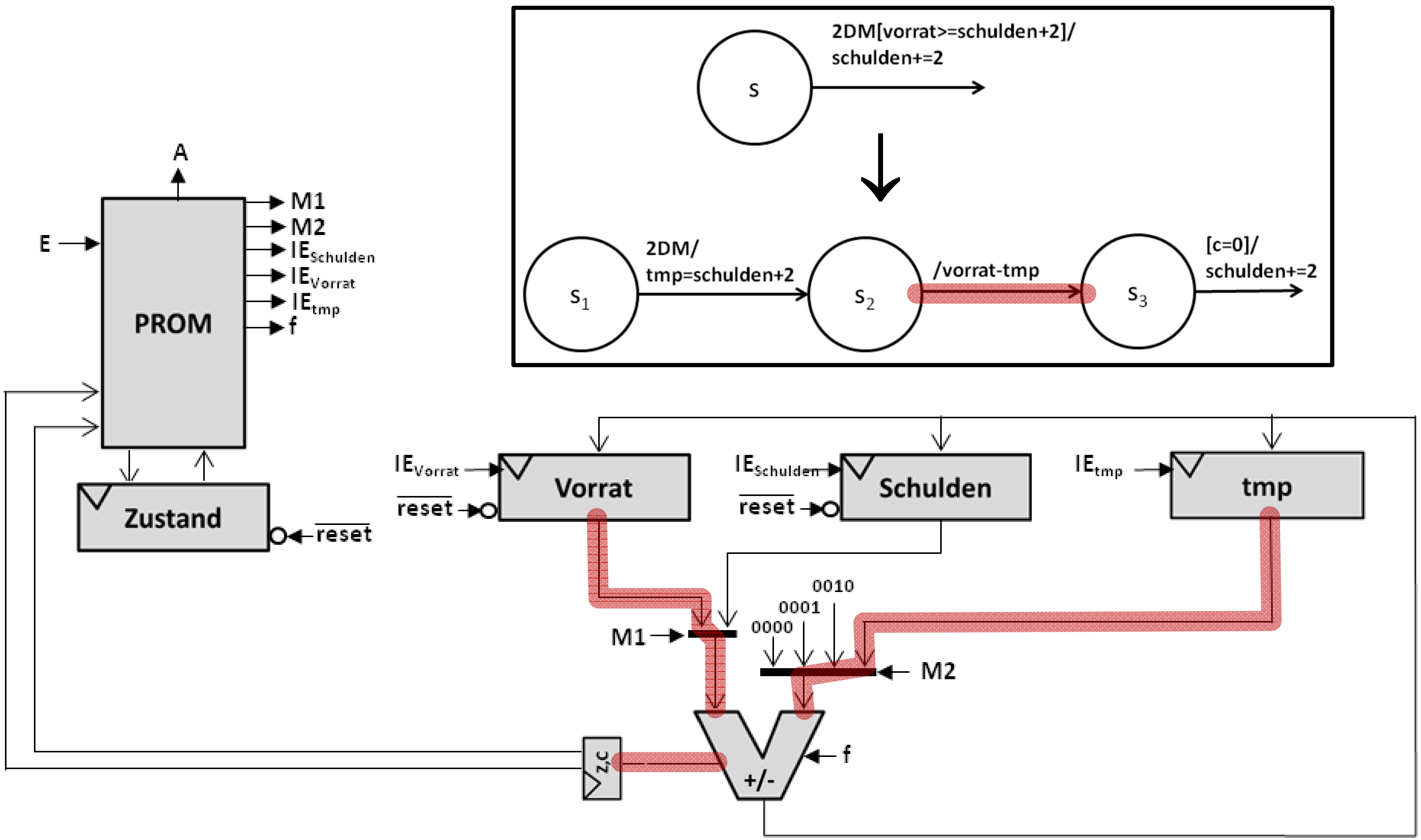
Anwendungsbeispiel – Einfache Architektur



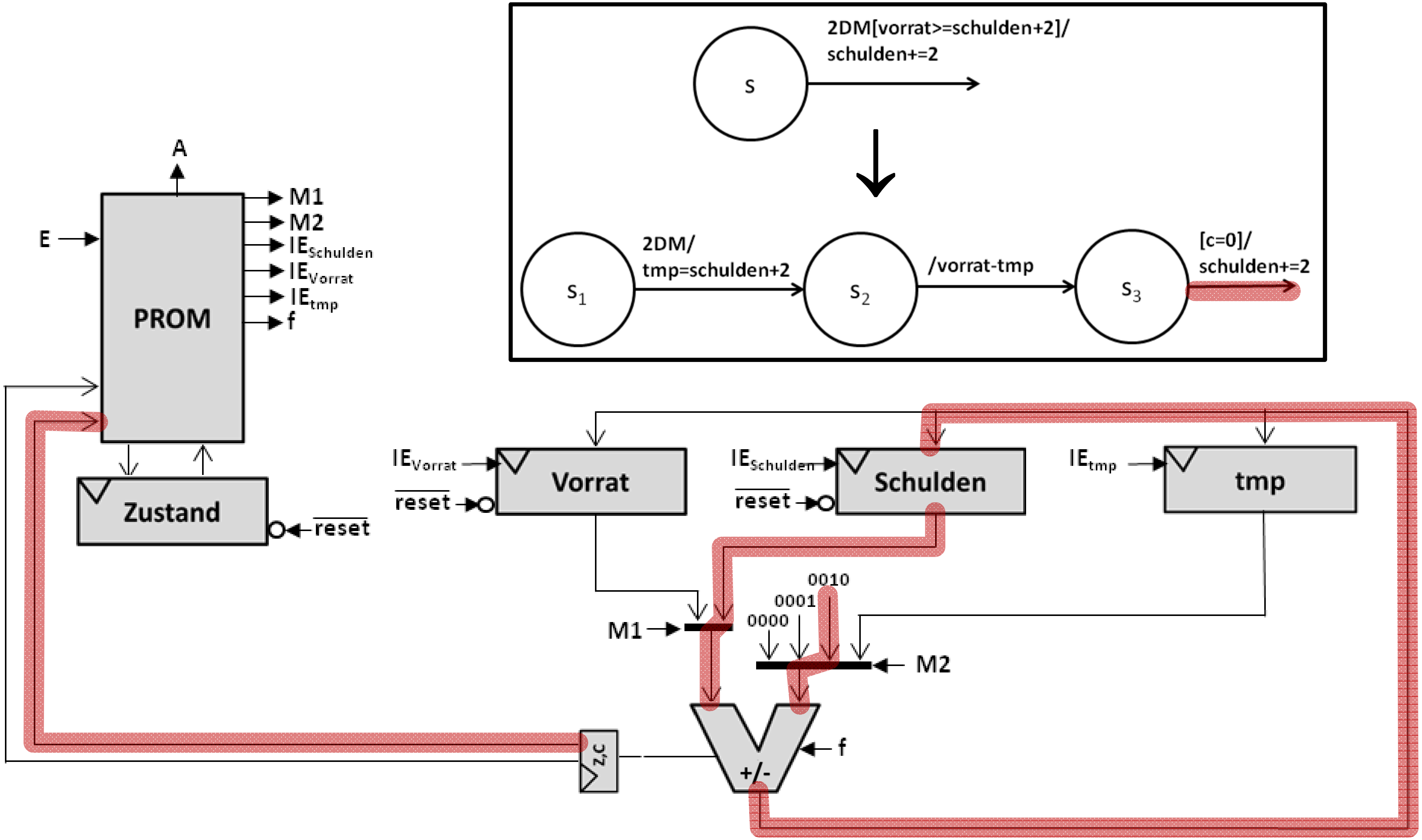
Anwendungsbeispiel – Einfache Architektur



Anwendungsbeispiel – Einfache Architektur



Anwendungsbeispiel – Einfache Architektur



Anwendungsbeispiel 4 – Erweiterung

- Münzeinwurfschacht ist nach jedem Einwurf verriegelt, bis er wieder freigeschaltet wird (MSF=‘1‘)
- Inputs werden in Registern gesammelt, pro Input 1 Register
- Outputs werden aus Registern abgeholt, pro Output 1 Register (Achtung: 10Z-Output muss wieder auf ‘0‘ gesetzt werden nach Auswurf der Münzen)
- Konstante Eingänge auf Operand 2 des Addierers werden vom PROM bereitgestellt
- Operand 1 des Addierers bekommt noch einen zusätzlichen Eingang mit der Konstante “0000“

➔ Datenpfad siehe Tafel

Anwendungsbeispiel 4 – Beobachtung

- Zustandsgraph: siehe Tafel
- Zustandsübergangstabelle: siehe Tafel
- Beobachtungen:
  - Zustand+1 ist Mehrheit aller Zustandswechsel
  - Manchmal 2 mögliche Nachfolgezustände abhängig von den Flags Z und C, aber
  - Einer davon ist immer Zustand+1

➔ Fasse Zustandsübergang als Addition im Zweier-komplement auf!

➔ Architekturerweiterung: siehe Tafel

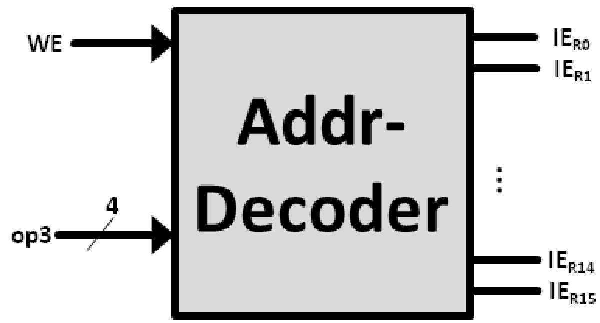
Anwendungsbeispiel 4 – Vereinfachung

- Vereinfache Datenpfad aus Sicht der Zustände:
  - Spendiere weitere Register, so dass die Anzahl der Register eine 2er-Potenz ist → 16 Register
  - Verdrahte das Register mit der Ordnungsnummer 0 fest mit dem Wert 0 (Entfall der Konstante 0 für Operanden), welches nicht beschrieben werden kann
  - Aus der Information WE und zusätzlicher Angabe des Registers kann gefolgert werden, welche IEs zu setzen sind → kodiere RegisterNr logarithmisch

    (Tabelle siehe Tafel)

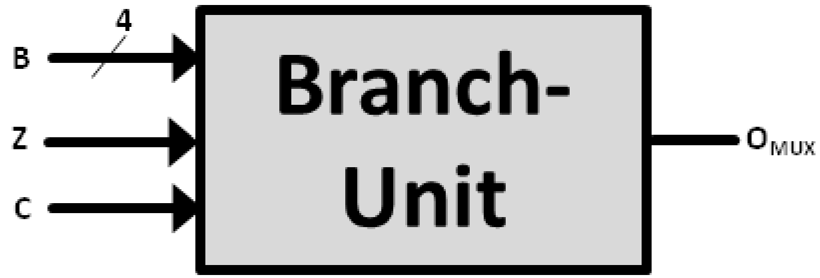
Registerbelegung

Nr	Inhalt
0	0x00
1	1DM_out
2	2DM_out
3	10Z_out
4	MSF_out
5	Schulden
6	Vorrat
7	tmp
8	1DM_in
9	2DM_in
10	WT
11	*ungenutzt*
12	*ungenutzt*
13	*ungenutzt*
14	*ungenutzt*
15	*ungenutzt*



Anwendungsbeispiel 4 – Vereinfachung

- Vereinfache Datenpfad aus Sicht der Zustände:
  - Trenne die Ansteuerung des Multiplexers vor dem Addierer für den Nachfolgezustand vom PROM und spendiere Schaltnetz dafür, dass flexibler ist:

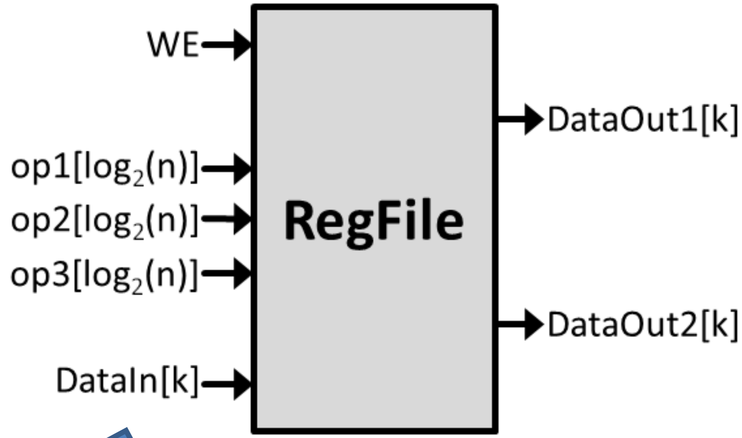


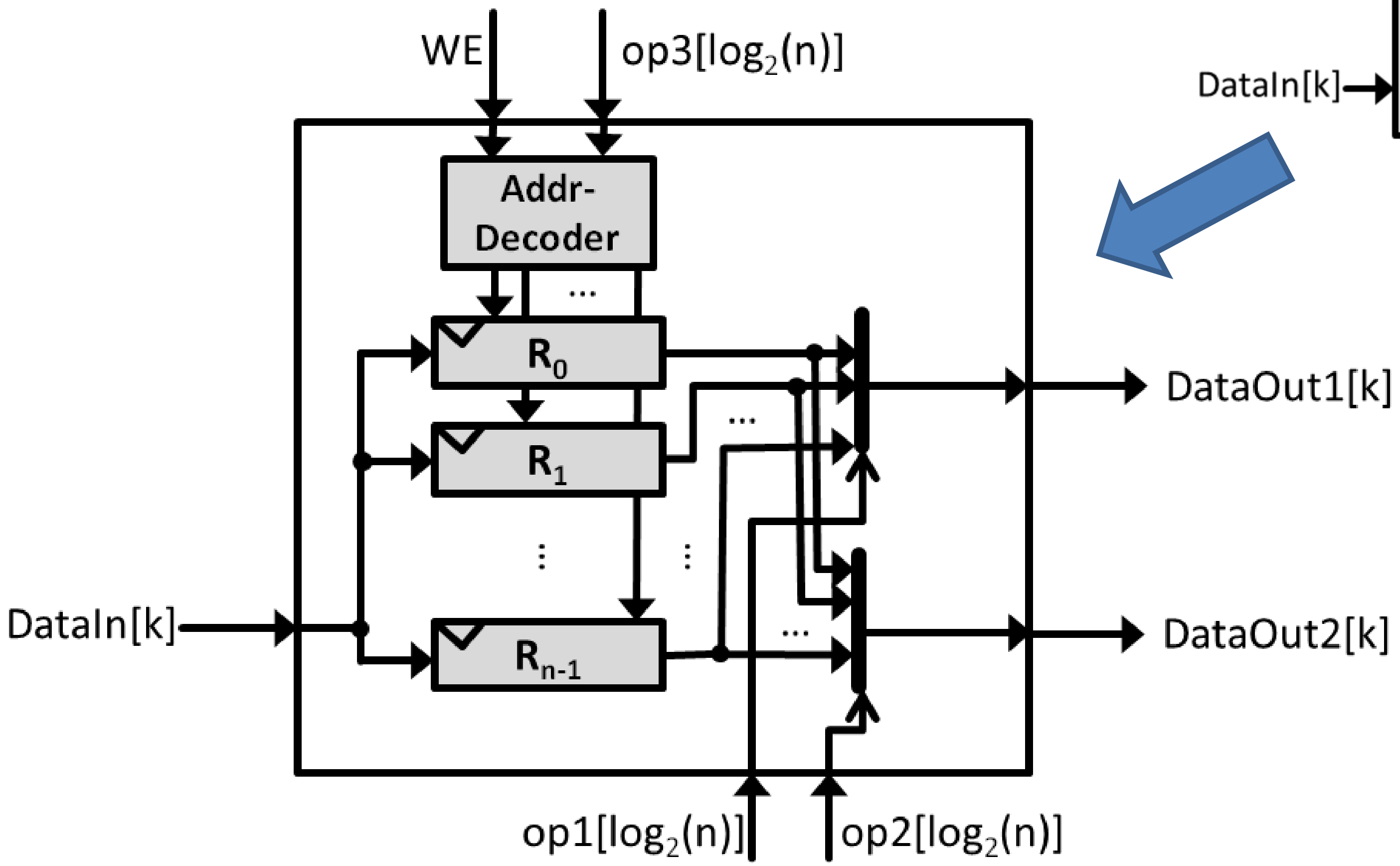
  - Wobei das 4-Bit-Signal B vom PROM aus angesteuert wird
  - Idee:
    - 1 Bit (B_3) kodiert, ob überhaupt ein Offset anstelle von +1
    - 1 Bit (B_2) kodiert, ob Flag oder invertiertes Flag geprüft wird
    - 2 Bit (B_1,B_0) kodieren, welches Flag (Z oder C)

(Wahrheitswerte-Tabelle siehe Tafel)

Anwendungsbeispiel 4 – Vereinfachung

- Mehrfache (n) Register der Breite k nennt man **Registerfile**





Anwendungsbeispiel 4 – Vereinfachung

- Mit den Vereinfachungen ergibt sich neuer Datenpfad → Tafel
- Daraus folgt neue Zustandsübergangstabelle → Tafel
- Anschließend folgende Namenskonvention:
  - Zustände mit B_3=0, f=0/1 und WE=1 heißen ADD/SUB
  - Zustände mit B_3=0, f=0/1 und WE=0 heißen TADD/TSUB
  - Zustände mit B_3=1 heißen Bxx, wobei xx aus Tabelle (siehe Tafel) entnommen wird

➔Damit entsteht Namenstabelle → Tafel

Anwendungsbeispiel 4 – Schreibweisen

- Zustandsgraphen werden nun nicht als Graph, sondern als Text aufgeschrieben, wobei die folgende Namenskonvention gilt:
  - Anstelle der Zustände benutzen wir die Namen
  - Lasse Felder mit “xxxx“ weg
  - Konstanten werden “#*const*“ geschrieben (*const*=Zahlwert)
  - Schreibe Registernummern als “rop_i“ (op_i=Dezimalzahl)

- Für unser Beispiel: siehe Tafel