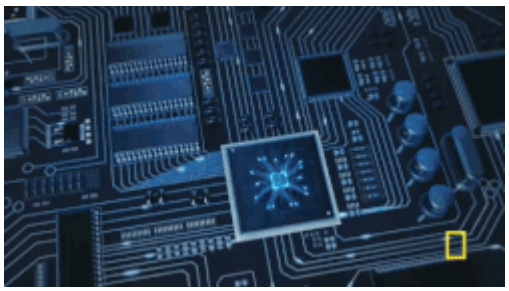
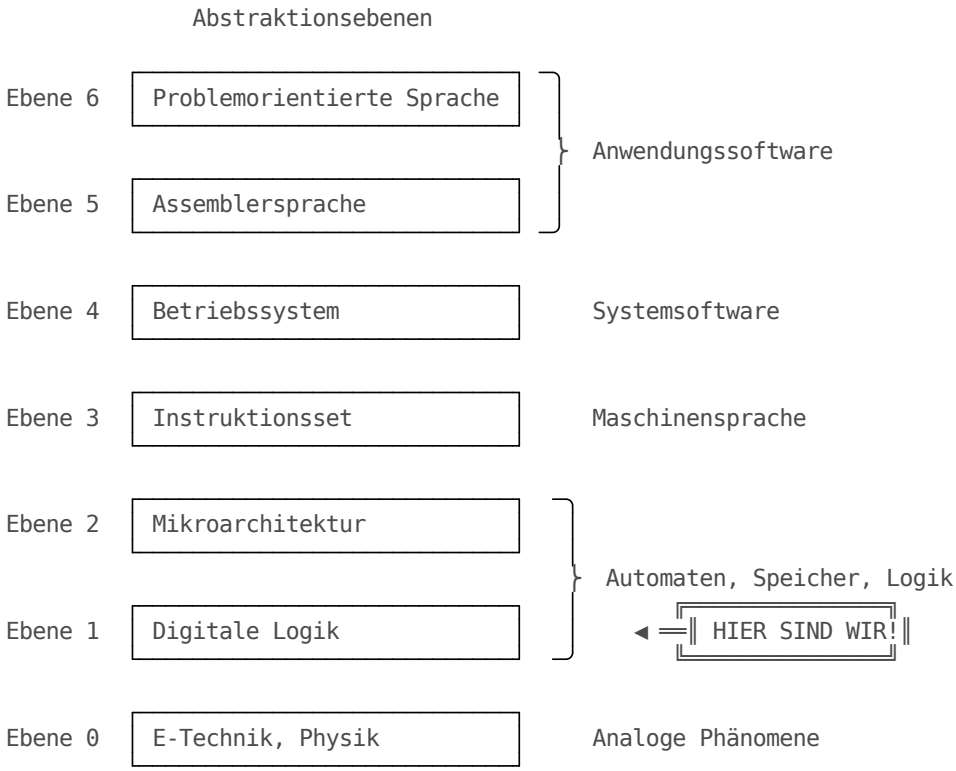
Schaltwerke

Parameter	Kursinformationen
Veranstaltung:	`Digitale Systeme / Eingebettete Systeme`
Semester:	`Wintersemester 2025/26`
Hochschule:	`Technische Universität Freiberg`
Inhalte:	`Endliche Automaten, Zustandsdiagramme, Mealy- und Moore-Automaten`
Link auf GitHub:	https://github.com/TUBAF-IfI-LiaScript/VL_EingebetteteSysteme/blob/master/07_Schaltwerke.md
Autoren:	Sebastian Zug & André Dietrich & Fabian Bär



Fragen an die Veranstaltung

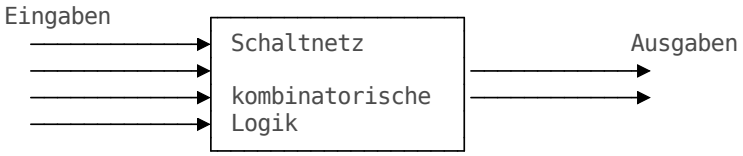
- Welche Kernelemente hat ein Schaltwerk?
- Worin unterscheiden sich die Varianten von Mealy und Moore?
- Welches Vorgehen ist für die Umsetzung eines Schaltwerkes notwendig?
- An welcher Stelle ist die invertierte Wahrheitstabelle eines Flip-Flops wichtig?
- Wie erstellen Sie ein Zustandsdiagramm für ein gegebenes Problem?
- Was verstehen Sie unter einem deterministischen endlichen Automaten?
- Welche Rolle spielt die Zustandskodierung bei der Schaltwerksynthese?
- Wie können Sie die Anzahl der benötigten Flip-Flops für ein Schaltwerk bestimmen?



Motivation

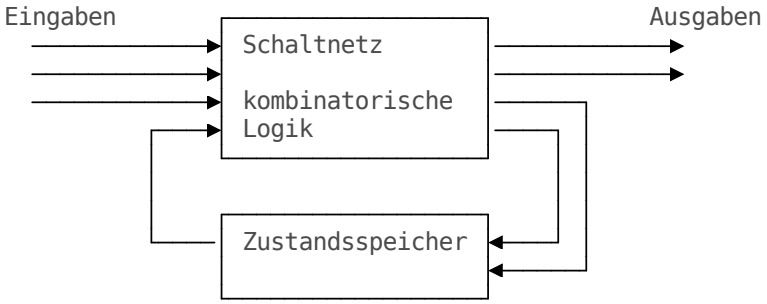
In der kombinatorischen Logik:

- werden Gatter als verzögerungsfrei angenommen (Idealisierung, die oft zu Problemen führt !)
- sind keine Rückkopplungen gestattet
- werden Schaltungen als Schaltnetze bezeichnet
- können Schaltungen als gerichteter azyklischer Graph dargestellt werden



In der sequentiellen Logik:

- sind Rückkopplungen gestattet
- werden Schaltungen als Schaltwerke bezeichnet
- können Schaltungen als gerichteter zyklischer Graph dargestellt werden



.

Und wo brauche ich sequentielle Logik? Die Abarbeitung von Befehlen im Rechner basiert darauf. In diesem Fall wird über die Eingabesteuerleitungen ein Befehl codiert und der Zustandspeicher beinhaltet den erreichten Realisierungsschritt. **Damit wird jeder Rechner zu einem Schaltwerk!**

Wiederholung: Beschreibung von Flip-Flops

Wahrheitstafel zeigt die Eingaben, die notwendig sind, um eine bestimmte Zustandsänderung herbeizuführen.

$R(t)$	$S(t)$	$Q(t+1)$
0	0	Q
0	1	1
1	0	0
1	1	Vermeiden

Invertierte Wahrheitstafel zeigt die Eingaben, die notwendig sind, um eine bestimmte Zustandsänderung herbeizuführen.

$Q(t)$	$Q(t+1)$	S	R
0	0	0	d
0	1	1	0
1	0	0	1
1	1	d	0

Merke: Die invertierte Wahrheitstafel beantwortet die Frage, wie die Eingänge beschaltet werden müssen, um einen bestimmten Zustand zu realisieren.

Aufgabe: Stellen Sie die invertierten Wahrheitstafeln für ein D-Flip-Flop auf!

D-Wahrheitstafel

$D(t)$	$Q(t+1)$	**Modus**
0	0	Löschen
1	1	Setzen

D-Invertierte Wahrheitstafel

$Q(t)$	$Q(t+1)$	D
0	0	
0	1	
1	0	
1	1	

$Q(t)$	$Q(t+1)$	D
0	0	0
0	1	1
1	0	0
1	1	1

Aufgabe: ... und noch mal!

JK-Wahrheitstafel

$J(t)$	$K(t)$	$Q(t+1)$	**Modus**
0	0	Q	Beibehalten
0	1	0	Löschen
1	0	1	Setzen
1	1	$\overline{Q}$	Toggeln

JK-Invertierte Wahrheitstafel

$Q(t)$	$Q(t+1)$	J	K
0	0		
0	1		
1	0		
1	1		

$Q(t)$	$Q(t+1)$	J	K
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

Grundkonzept

Wie kann man systematisch ein synchrones Schaltwerk ausgehend von der Problembeschreibung entwerfen?

Automat ist gekennzeichnet durch:

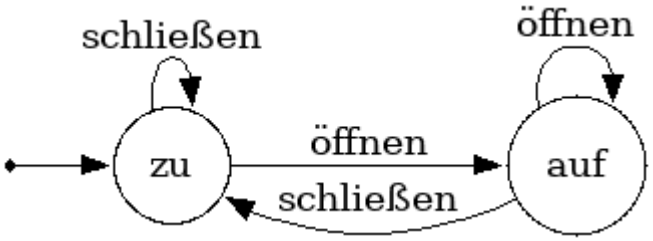
- beliebige (jedoch endliche) Menge von Zuständen
- Zustandsübergänge in jedem Takt abhängig von Eingangssignalen
- Ausgangssignale werden durch ein Schaltnetz generiert

Mathematisch kann ein Deterministischer Endlicher Automat als Tupel $A = (Q, \Sigma, \delta, q_0, F)$ dargestellt werden.

- Q ist eine endliche Zustandsmenge.
- Σ ist das endliche Eingabealphabet, also die Menge erlaubter Eingabesymbole.
- $\delta : Q \times \Sigma \to Q$ ist die Übergangsfunktion (oder Transitionsfunktion). Sie ordnet jedem Paar bestehend aus einem Zustand $q \in Q$ und einem Eingabesymbol $a \in \Sigma$ einen Nachfolgezustand $p \in Q$ zu.
- $q \in Q$ ist der Startzustand (auch Anfangszustand oder Initalzustand).
- $F \subseteq Q$ ist die Menge der akzeptierenden Zustände, die sogenannten Finalzustände (oder Endzustände).

> Beispiel I: **Zustand einer Tür**

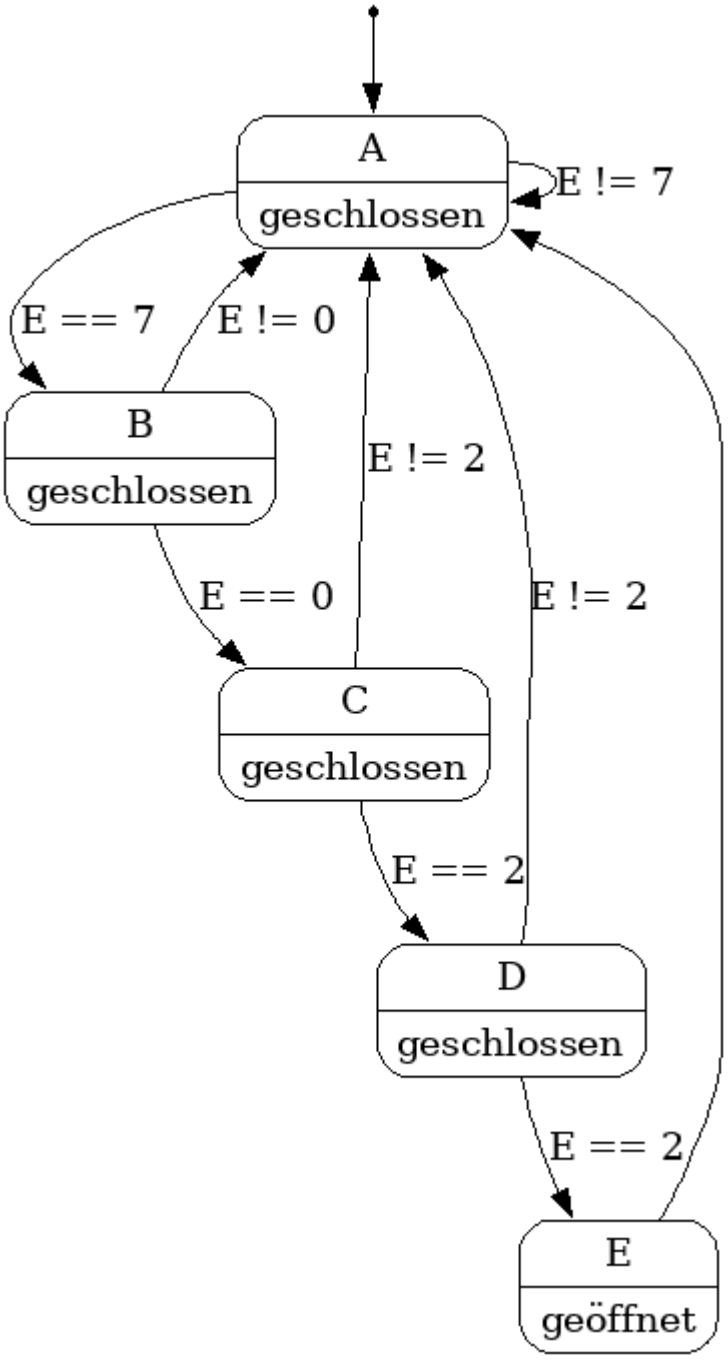
Darstellung als Graph



Darstellung in einer Übergangstabelle

	"Öffnen"	**"Schließen"**
Zustand offen	offen (unverändert)	geschlossen
Zustand geschlossen	offen	geschlossen (unverändert)

Beispiel II: **Codeschloss für die Erfassung der Sequenzfolge** `7022`

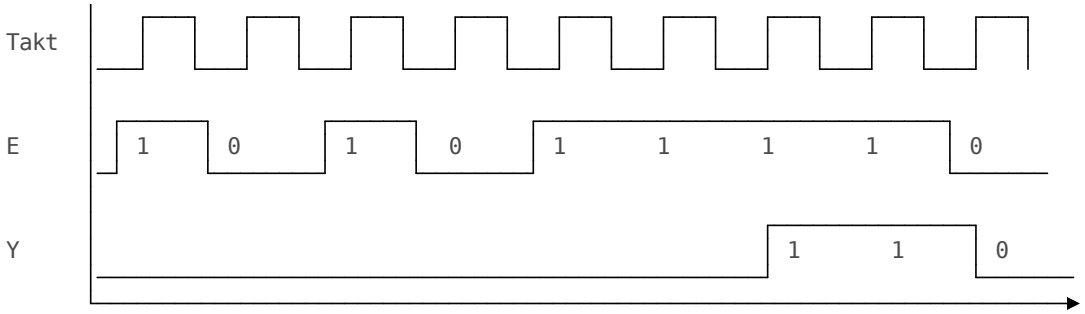


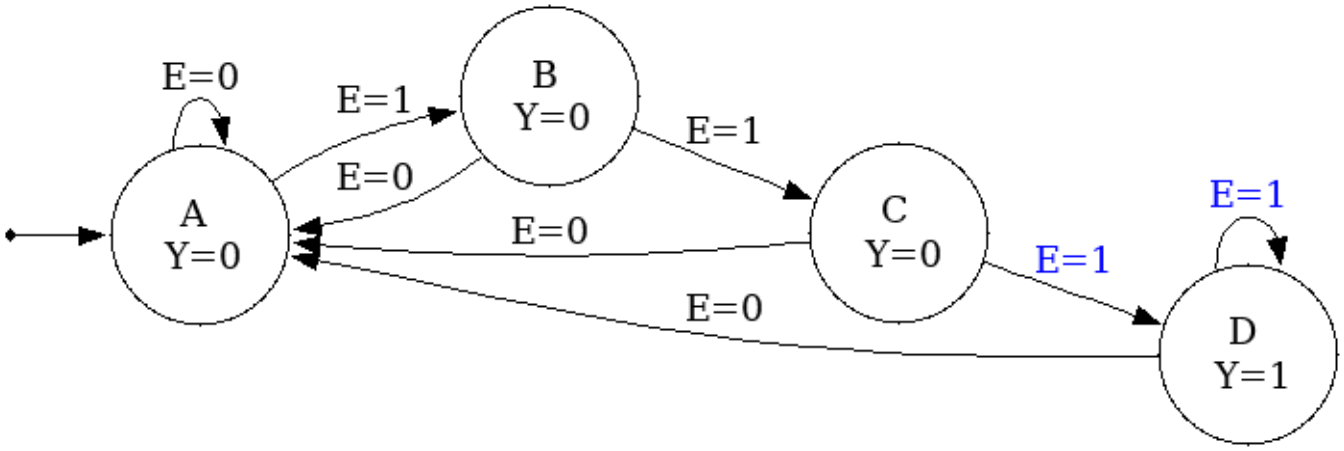
Welche Beschränkungen sehen Sie in diesem Entwurf?
Was passiert bei der Sequenzfolge `707022`?

Wie realisieren wir nun aber das theoretische Modell des endlichen Automaten mit realen Bauteilen?

Beispiel

Binärsequenzdetektor, der drei aufeinander folgende `1` erkennt. Dabei gehen wir davon aus, dass die Übernahme der Werte mit den steigenden Flanken des Taktsignales erfolgt.





- Zustand `A`: die letzte Eingabe war keine `1` (`0` oder Start)
- Zustand `B`: die letzte Eingabe war `1`, die davor war nicht `1`
- Zustand `C`: die letzten beiden Eingaben waren `1`, die davor war nicht `1`
- Zustand `D`: mindestens die letzten drei Eingaben waren `1`

Zustandsübergänge

aktueller Zustand	Eingabe E	neuer Zustand
A	0	A
B	0	A
C	0	A
D	0	A
A	1	B
B	1	C
C	1	D
D	1	D

Kodierung

Für unsere digitalen Bauteile müssen wir diese Zustände aber mit `1` und `0` kodieren.

Zustände	Flip-Flops	Mögliche Zustände	Ungenutzte Zustände
1	-	-	-
2	1	2	0
3	2	4	1
4	2	4	0
5	3	8	3
...			
8	3	8	0
9	4	16	7
...			

Da wir insgesamt 4 Zustände haben, braucht es $\lceil log_2(4) \rceil = 2$ Speicherelemente, also Flip-Flops. Die zwei Flip-Flops werden im folgenden als `F` und `G` bezeichnet. Die Ausgabe bezeichnen wir mit `X`.

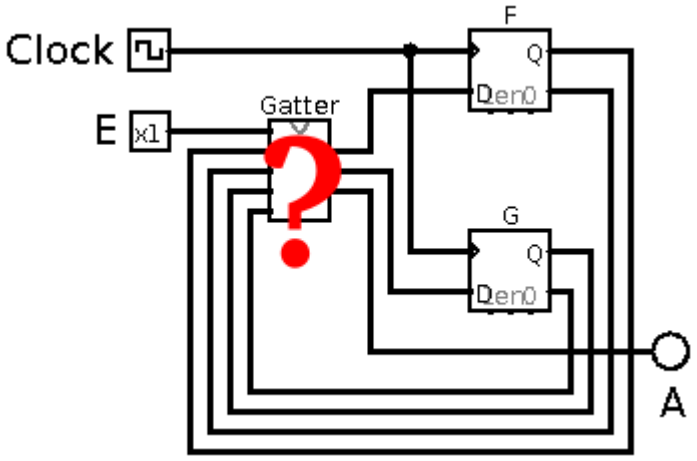
Mit unserem Schaltwerk wollen wir also eine Funktion abbilden, die die Ausgabe `X` in Abhängigkeit von (historischen) `E` Eingaben generiert. Um letztgenannten Anteil zu integrieren, braucht es die Zustände, die in `F` und `G` erfasst sind.

Zustand	Flip-Flop F	Flip-Flop G
A	0	0
B	0	1
C	1	0
D	1	1

Damit ergibt sich dann eine neue Zustandstabelle:

Zustand	F	G	E	Zustand'	F'	G'
A	0	0	0	A	0	0
B	0	1	0	A	0	0
C	1	0	0	A	0	0
D	1	1	0	A	0	0
A	0	0	1	B	0	1
B	0	1	1	C	1	0
C	1	0	1	D	1	1
D	1	1	1	D	1	1

Realisierung der Schaltfunktion



Um die entsprechende Schaltfunktionen für die Änderung der Zustände und die Ausgabe aufzustellen, brauchen wir die invertierte Wahrheitstafel des intendierten Flip-Flops.

> **Merke:** Unterschiedliche Flip-Flops für die Speicherung der Zustände führen zu unterschiedlichen Beschaltungen!

Zustandstabelle

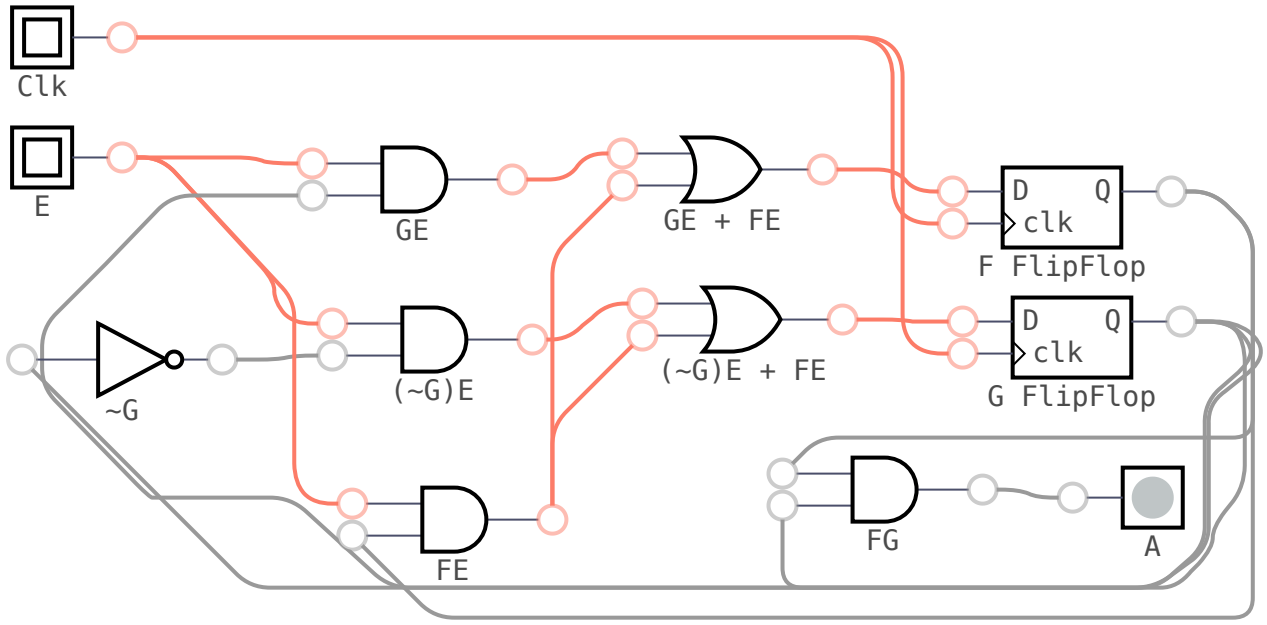
F	G	E	F'	G'
0	0	0	0	0
0	1	0	0	0
1	0	0	0	0
1	1	0	0	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	1
1	1	1	1	1

Ausgaben

F	G	A
0	0	0
0	1	0
1	0	0
1	1	1

$$
\begin{aligned}
F' &= \overline{F}GE + F\overline{G}E + FGE \\
&= GE + FE \\
G' &= \overline{F}\overline{G}E + F\overline{G}E + FGE \\
&= \overline{G}E + FE \\
A &= FG
\end{aligned}
$$

Bitte mit Eingabe 0 starten, um Startzustand korrekt zu setzen!



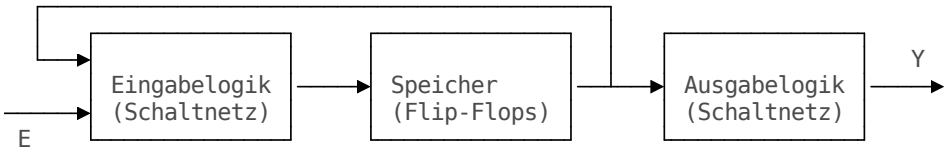
Vorgehensweise

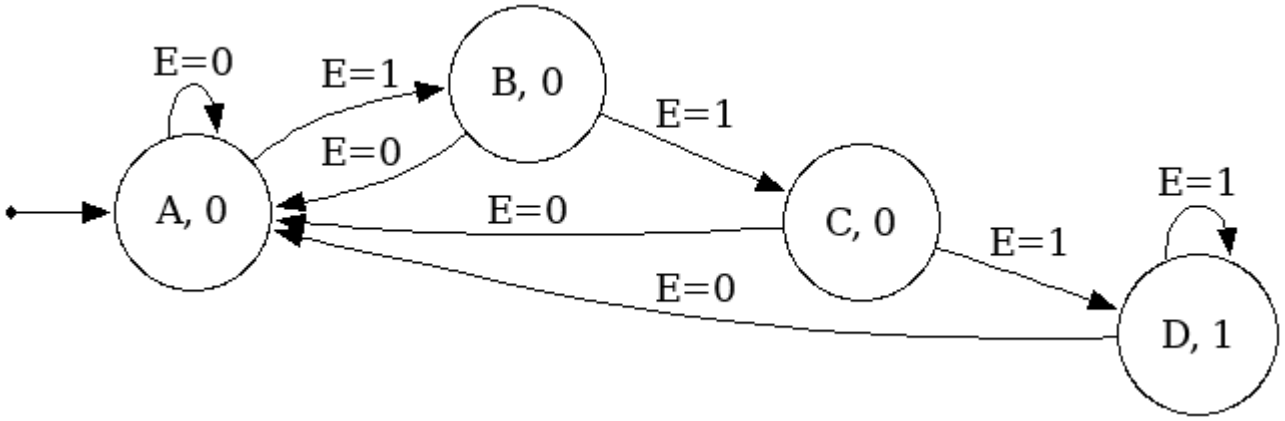
1. Schritt: Spezifikation des Zustandsdiagramms
2. Schritt: Transformation in eine Zustandstabelle
3. Schritt: Zuordnung von Zuständen zu Flip-Flop Belegungen
4. Schritt: Erstellung der Wahrheitstafel für Zustände und Ausgaben
5. Schritt: Ableitung einer KNF oder DNF
6. Schritt: Minimierung

Automaten Typen

Moore-Automat

Edward Forrest Moore (1925 – 2003, Bell Labs)

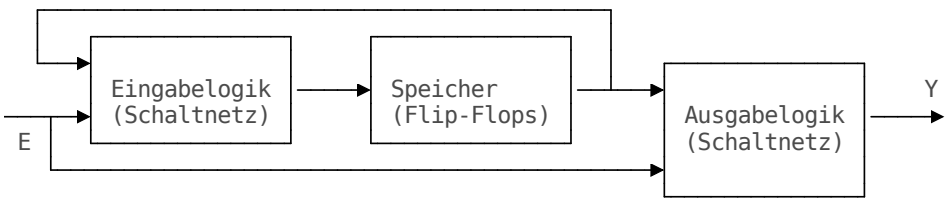


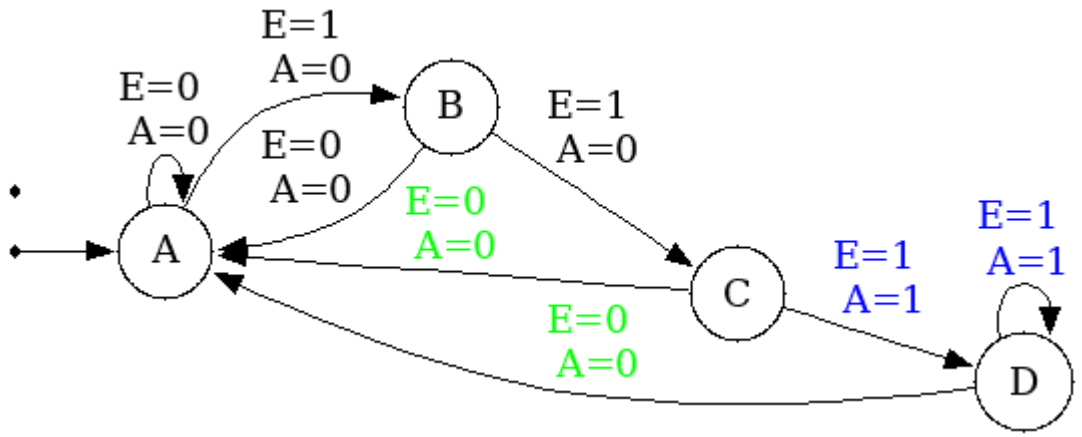


Die Ausgabelogik bestimmt Ausgabe Y hängt nur vom aktuellen Zustand ab.

Mealy-Automat

George H. Mealy (1927 – 2010, IBM)



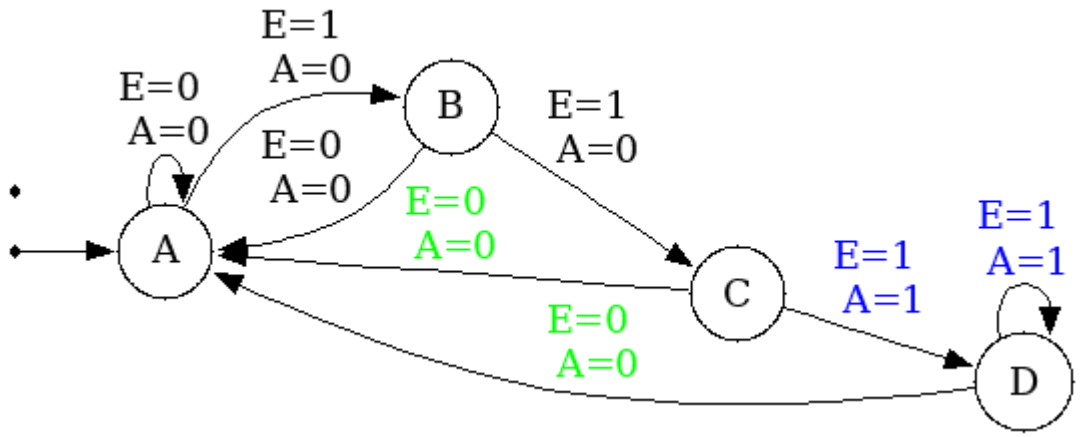


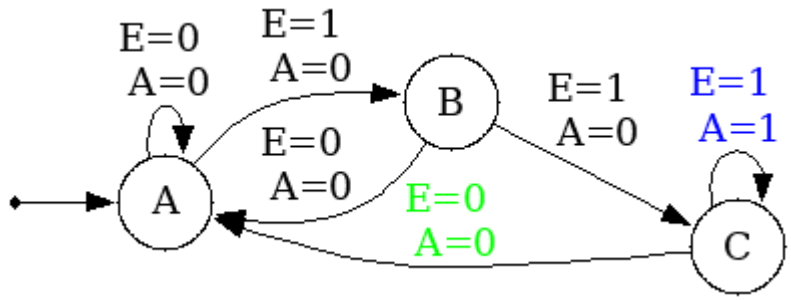
Die Ausgabe Y der Ausgabelogik hängt sowohl vom aktuellen Zustand als auch vom Eingabesignal E ab

Der Mealy-Automat ist die generellere Form. Der Moore-Automat unterbindet den Einfluss des Einganges. Eine weitere Spezialisierung ist der sogenannte Medwedew-Automat, bei dem ganz auf die Ausgabelogik verzichtet wird.

	Mealy-Automat	**Moore-Automat**
Vorteile	schnellere Reaktion auf Veränderung der Eingabesignale E	taktsynchrone Ausgabe A, asynchron auftretende Störungen der Eingabesignale wirken sich nicht auf A aus
	Realisierung ist mit einer kleineren Anzahl an Zuständen möglich, wenn mehrere Zustandsübergänge zu einem Zustand verschiedene Ausgaben erfordern	geringerer Schaltungsaufwand für Ausgabelogik, wenn Ausgabe A eigentlich nur vom Zustand abhängt

Noch mal zurück zum Beispiel des Binärsequenzdetektors. Welche Konsequenzen hätte die Umsetzung als Mealy-Automat?





Zustandstabelle

Zustand	F	G	E	Zustand'	F'	G'
A	0	0	0	A	0	0
B	0	1	0	A	0	0
C	1	0	0	A	0	0
D	1	1	0	don't care	d	d
A	0	0	1	B	0	1
B	0	1	1	C	1	0
C	1	0	1	C	1	0
D	1	1	1	don't care	d	d

Ausgaben

F	G	E	A
0	0	0	0
0	1	0	0
1	0	0	0
1	1	0	d
0	0	1	0
0	1	1	0
1	0	1	1
1	1	1	d

$$
\begin{aligned}
F' &= \overline{F}GE + F\overline{G}E \\
G' &= \overline{F}\overline{G}E \\
A &= F\overline{G}E
\end{aligned}
$$

Wir nutzen die *Don't-care*-Zustände noch weiter aus und ermöglichen eine zusätzliche Vereinfachung.

F'

	$\overline{F}\overline{G}$	$\overline{F}G$	FG	$F\overline{G}$
$\overline{E}$			d	
E		1	d	1

A

	$\overline{F}\overline{G}$	$\overline{F}G$	FG	$F\overline{G}$
$\overline{E}$			d	
E			d	1

G'

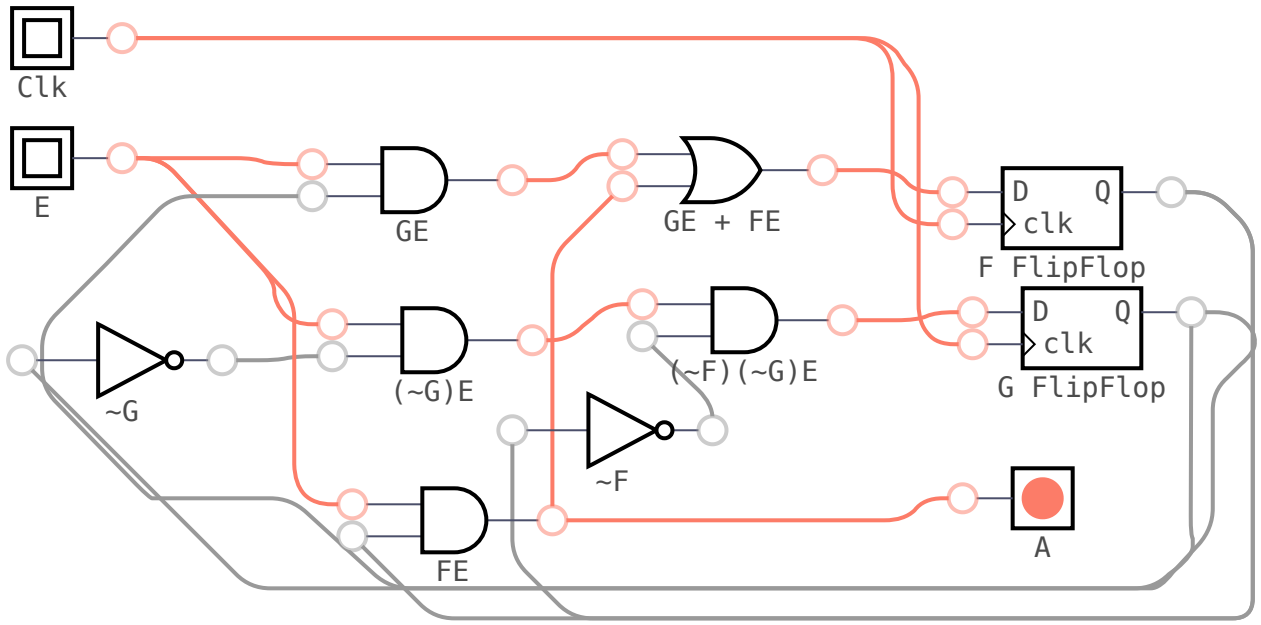
	$\overline{F}\overline{G}$	$\overline{F}G$	FG	$F\overline{G}$
$\overline{E}$			d	
E	1		d	

.

$$
\begin{aligned}
F' &= \overline{F}GE + F\overline{G}E \\
&= GE + FE \\
G' &= \overline{F}E\overline{G} \\
A &= F\overline{G}E + FGE = FE
\end{aligned}
$$

Damit ergibt sich eine alternative Realisierung unseres Schaltwerkes.

Bitte mit Eingabe 0 starten, um Flipflops zurück zu setzen!



In der Simulation sehen Sie, dass wir gegenüber dem Moore-Automaten ...

$$F' = GE + FE$$
$$G' = \overline{G}E + FE$$
$$A = FG$$

Bedeutung des Flip-Flop Typs

Nehmen wir an, dass die Realisierung nicht mit einem D sondern einen JK-Flip-Flop erfolgen soll.

J	K	$Q(t+1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	$\overline{Q(t)}$

Der JK-Flip-Flop wechselt beim setzen von J in einen 1 Zustand und kann mit K resetet werden. Eine gleichzeitige Aktivierung beider Eingänge führt zu einem Togglen des Zustandes.

$Q(t)$	$Q(t+1)$	J	K
0	0	0	d
0	1	1	d
1	0	d	1
1	1	d	0

Der Wechsel von $Q(t) = 0$ nach $Q(t+1) = 1$ kann entweder über ein Setzen ($J = 1$) oder ein Togglen ($J = K = 1$) umgesetzt werden. Daher spielt der Zustand des Einganges K keine Rolle.

Zwar haben wir es nun mit jeweils zwei Eingängen für den Flip-Flop zu tun (im Unterschied zum D-Flip-Flop). Dies äußert sich in einer zusätzlichen Spalte der Zustandsübergangstabelle. Die *Don't-care*-Konfigurationen ermöglichen aber eine höhere Flexibilität beim Entwurf.

Wie muss also die Beschaltung vorgenommen werden, um die bereits bekannte Zustandsübergangstabelle mit dem JK-Flip-Flop umzusetzen? Beginnen wir zunächst mit unserem ersten Flip-Flop F und seinen Eingängen JF und KF.

F	G	E	F'	G'	JF	KF	JG	KG
0	0	0	0	0	0	d		
0	1	0	0	0	0	d		
1	0	0	0	0	d	1		
1	1	0	0	0	d	1		
0	0	1	0	1	0	d		
0	1	1	1	0	1	d		
1	0	1	1	1	d	0		
1	1	1	1	1	d	0		

Analog wird die Zustandsübergangstabelle für JG und KG befüllt.

F	G	E	F'	G'	JF	KF	JG	KG
0	0	0	0	0	0	d	0	d
0	1	0	0	0	0	d	d	1
1	0	0	0	0	d	1	0	d
1	1	0	0	0	d	1	d	1
0	0	1	0	1	0	d	1	d
0	1	1	0	0	1	d	d	1
1	0	1	0	1	d	0	1	d
1	1	1	0	1	d	0	d	0

JF

$\overline{E}$	$\overline{F}\overline{G}$	$\overline{F}G$	FG	$F\overline{G}$
$\overline{E}$			d	d
E		1	d	d

JG

$\overline{E}$	$\overline{F}\overline{G}$	$\overline{F}G$	FG	$F\overline{G}$
$\overline{E}$		d	d	
E	1	d	d	1

KF

$\overline{E}$	$\overline{F}\overline{G}$	$\overline{F}G$	FG	$F\overline{G}$
$\overline{E}$	d	d	1	1
E	d	d		

KG

$\overline{E}$	$\overline{F}\overline{G}$	$\overline{F}G$	FG	$F\overline{G}$
$\overline{E}$	d	1	1	d
E	d	1		

.

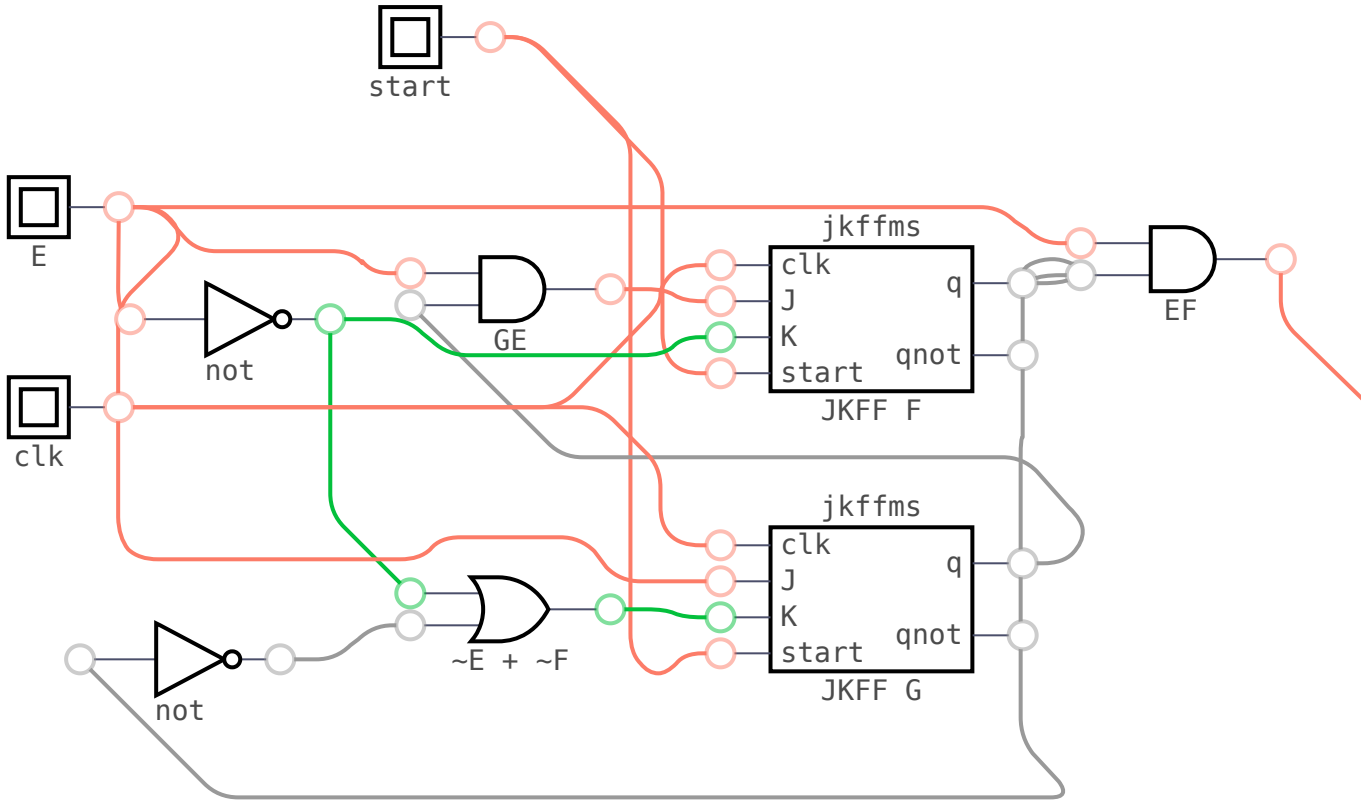
Damit lassen sich folgende Funktionen ablesen:

$$JF = GE$$
$$KF = \overline{E}$$
$$JG = E$$
$$KG = \overline{E} + \overline{F}$$

Und die Ausgabe? Die bleibt ja unabhängig von der konkreten Umsetzung mit Flip-Flops. Entsprechend gilt $A = EF$

Damit ergibt sich folgendes Simulationsbild:

Bitte am Anfang `start` auf AN, dann `clk` puls, dann `start` auf AUS, um JK Flipflops in den Startzustand zu versetzen.



Welche Unterschiede sehen Sie gegenüber der Realisierung mit D-Flip-Flops?

Lösung	JK-Flip-Flop	D Flip-Flop
Gleichungen	$JF = GE$ $KF = \overline{E}$ $JG = E$ $KG = \overline{E} + \overline{F}$ $A = EF$	$F' = GE + FE$ $G' = \overline{G}E + FE$ $A = EF$
Bauteile	Negation von E, 2 x AND, 1 x OR	3 x AND, 2 x OR

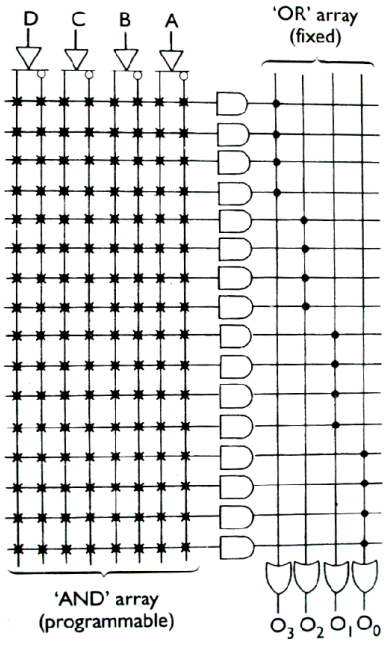
Merke: Jeder beliebige getaktete Flip-Flop Typ kann für die Umsetzung verwendet werden.

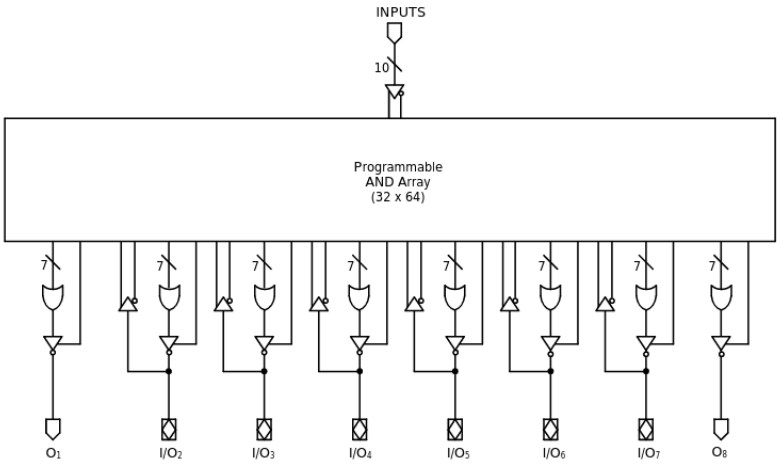
Dabei wirken sich die unterschiedlichen inversen Zustandsübergangstabellen mit entsprechenden „don't care" Einträgen auf die Komplexität des Schaltwerkes aus.

Realisierung von Schaltwerken

Eine Umsetzungsmöglichkeit für Schaltnetze sind die sogenannten PAL (Programmable Array Logic) die bereits in der Vorlesung 4 eingeführt wurden Link.

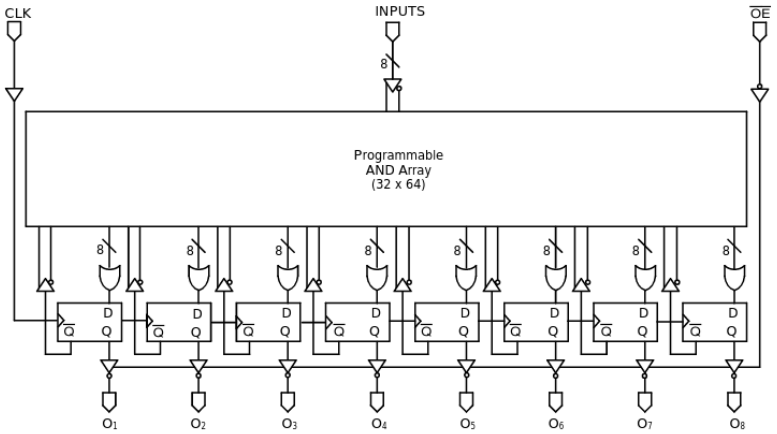
An dieser Stelle wurden die 2 stufigen Schaltfunktionen mit einem programmierbaren `AND` Array vorgestellt.





Advanced Micro Devices [1]

Diese erweitern wir nun um die Speicherglieder und deren Rückkopplung. Beachten Sie die Ergänzung auf der Ausgangsseite und die zusätzliche Clockleitung.



Advanced Micro Devices [2]

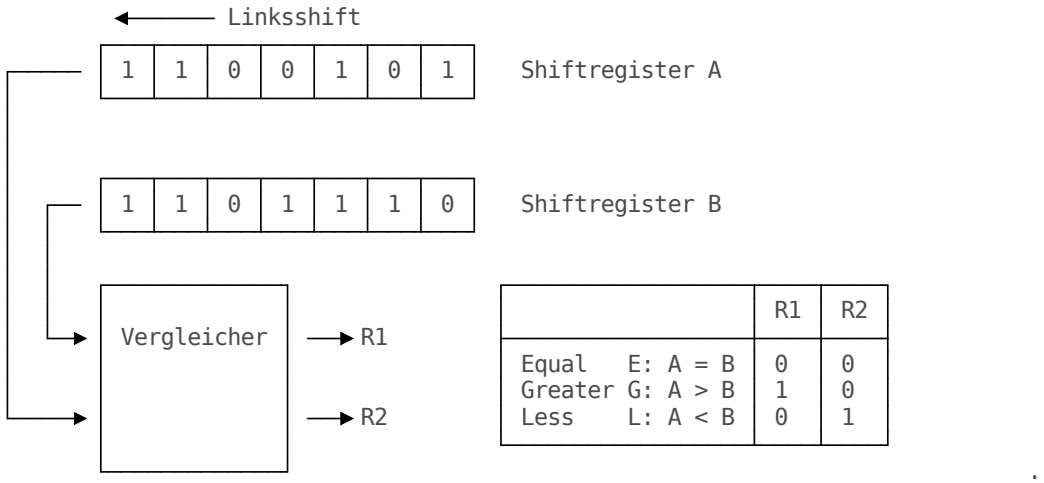
---

[1] Datenblatt PAL16R8 Family, Advanced Micro Devices, link, 1996

[2] Datenblatt PAL16R8 Family, Advanced Micro Devices, link, 1996

Beispielanwendung

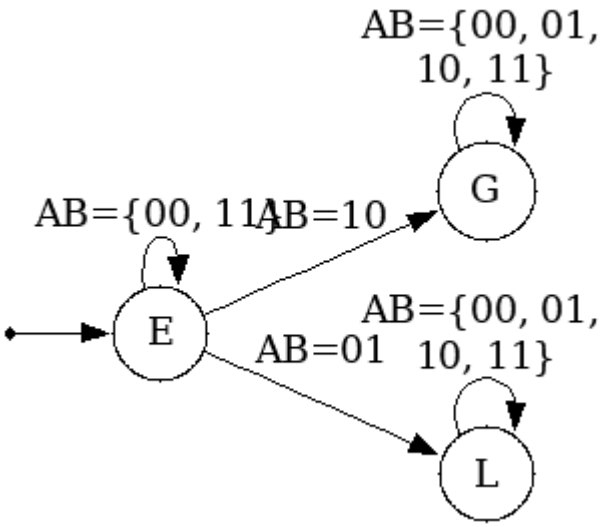
Sequentieller Binärzahlenvergleicher - zwei Zahlenwerte werden sequenziell entsprechend ihren Stellen durch den Vergleicher bewegt und verglichen. Das Schaltwerk speichert das Resultat sobald ein Wert größer als der andere ist.



	R1	R2
Equal E: A = B	0	0
Greater G: A > B	1	0
Less L: A < B	0	1

1. Schritt: Aufgabenspezifikation, Erstellen eines Zustandsdiagramms

Für die Aufgabe ergibt sich folgender Graph:



Im Beispiel liegt ein Medwedew-Automat vor. Die Zustände werden direkt auf den Ausgang abgebildet.

2. Schritt: Erstellen der Zustandstabelle

Hier wäre eine Zustandstabelle denkbar, die alle Eingangskombinationen mit allen Zuständen zeilenweise verknüpft.

aktueller Zustand	A	B	Folge-zustand
E	0	0	E
E	0	1	L
E	1	0	G
...			

Eine kompaktere Darstellung fasst die Kombinationen der Eingänge zusammen und ordnet sie den Folgezuständen zu.

aktueller Zustand	AB==00	AB==01	AB==10	AB==11
E	E	L	G	E
G	G	G	G	G
L	L	L	L	L

Schritt 3: Auswahl einer binären Zustandskodierung und Generierung einer binären Zustandstabelle

Insgesamt sind 3 Zustände zu kodieren, entsprechend werden wiederum 2 Flip-Flops benötigt. Dabei wird die Kodierung wie folgt vorgenommen:

Zustand	**F**	**G**
E	0	0
G	0	1
L	1	0

Damit ergibt sich folgende binäre Zustandstabelle

aktueller Zustand	**AB==00**	**AB==01**	**AB==10**	**AB==11**
00	00	10	01	00
01	01	01	01	01
10	10	10	10	10

In der traditionellen Darstellung zeigt sich diese wie folgt:

F_t	G_t	A_t	B_t	F_{t+1}	G_{t+1}
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	1	0
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	D	D
1	1	0	1	D	D
1	1	1	0	D	D
1	1	1	1	D	D

Schritt 4: Auswahl eines Flip-Flop Typs und Ermittlung der für jeden Zustandsübergang benötigten Flip-Flop Ansteuerungen

Wir entscheiden uns für einen D Flip-Flop für die Realisierung. Die entsprechende invertierte Wahrheitstafel haben Sie zwischenzeitlich im Kopf:

$Q(t)$	$Q(t+1)$	D
0	0	0
0	1	1
1	0	0
1	1	1

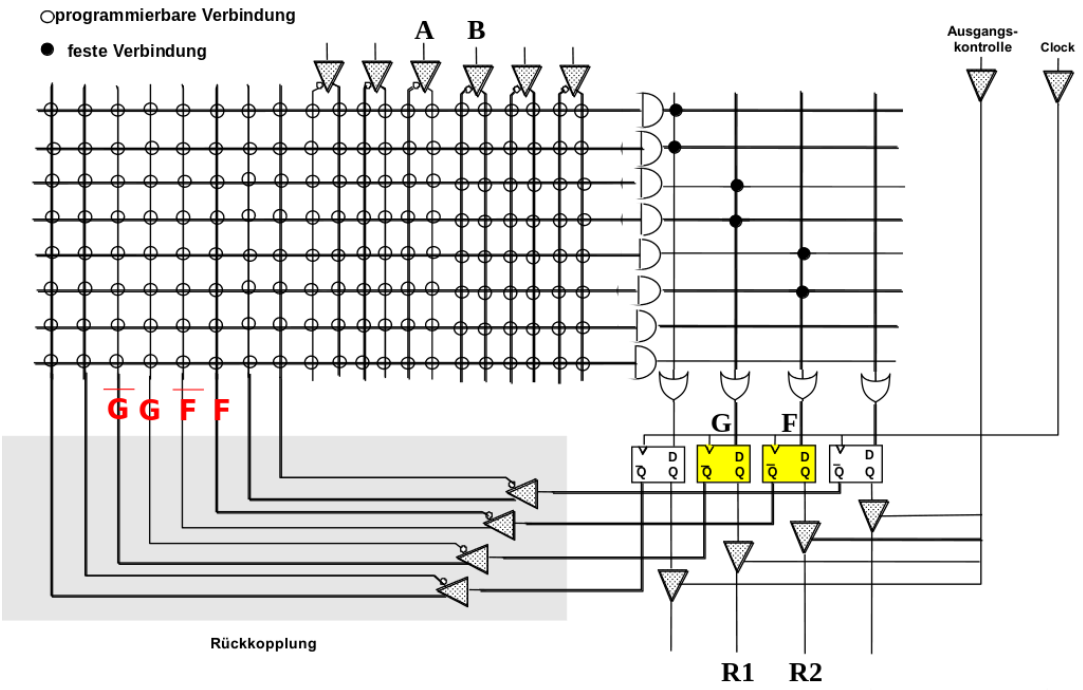
Damit lässt sich die Zustandsübergangstabelle entsprechend einfach um die zugehörige Eingangsbelegung ergänzen. Für die D-Flip-Flops ist dies einfach eine Kopie der Zustandsspalten.

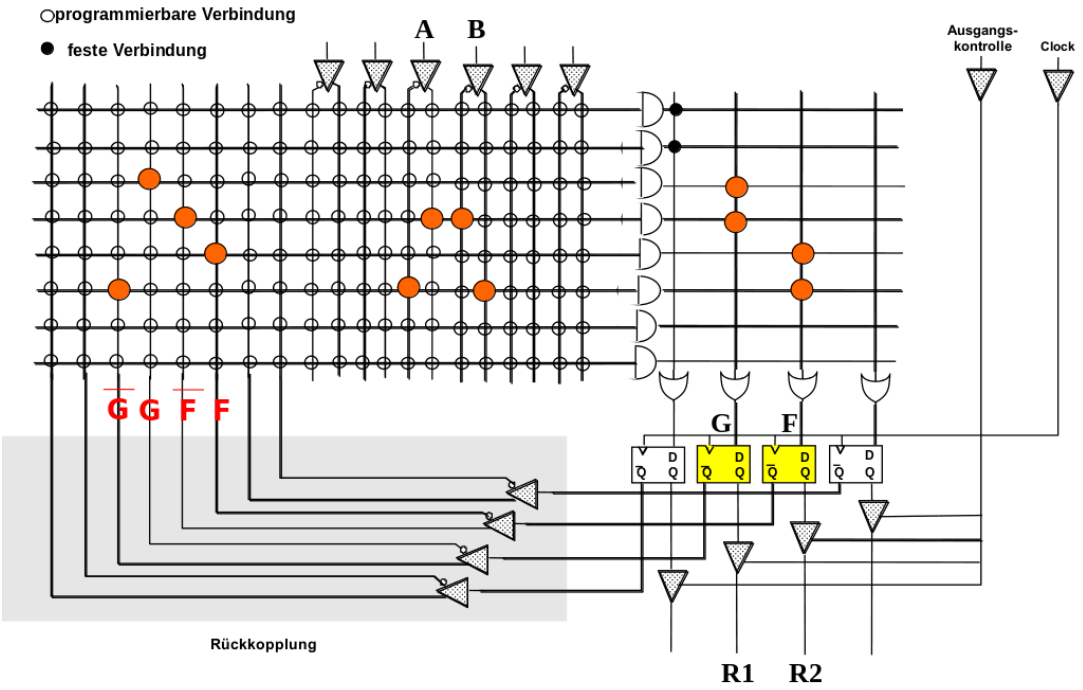
F_t	G_t	A_t	B_t	F_{t+1}	G_{t+1}	DF	DG
0	0	0	0	0	0	0	0
0	0	0	1	1	0	1	0
0	0	1	0	0	1	0	1
0	0	1	1	0	0	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	0	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	1
1	0	0	0	1	0	1	0
1	0	0	1	1	0	1	0
1	0	1	0	1	0	1	0
1	0	1	1	1	0	1	0
1	1	0	0	D	D	D	D
1	1	0	1	D	D	D	D
1	1	1	0	D	D	D	D
1	1	1	1	D	D	D	D

Aufgabe: Lesen Sie die minimale Funktion für `DF` und `DG` ab!

$$DF = F + \overline{G}\,\overline{A}\,B$$
$$DG = G + \overline{F}\,A\,\overline{B}$$

Aufgabe: Setzen Sie die Gleichungen mit einem PAL um!





AND-Verbindungen, welche dasselbe OR-Gatter besitzen, sind kommutativ.

Race Conditions in Schaltwerken

Eine **Race Condition** tritt auf, wenn das Verhalten einer Schaltung vom relativen Timing verschiedener Signale abhängt.

Typen von Race Conditions:

1. **Critical Race**: Das Endergebnis hängt vom Timing ab (problematisch)
2. **Non-Critical Race**: Das Endergebnis ist unabhängig vom Timing (unkritisch)

Beispiel einer Critical Race:

Betrachten Sie ein Schaltwerk mit zwei Flip-Flops, deren Ausgänge sich gleichzeitig ändern sollen:

```
Zustandsübergang: 11 → 00

FF1: Q1(t) = 1 → Q1(t+1) = 0
FF2: Q2(t) = 1 → Q2(t+1) = 0

Mögliche Zwischenzustände bei Race:
11 → 01 → 00  (FF1 schaltet zuerst)
11 → 10 → 00  (FF2 schaltet zuerst)
11 → 00       (beide schalten gleichzeitig)

Falls nachfolgende Logik auf Zwischenzustände reagiert,
kann unterschiedliches Verhalten entstehen!
```

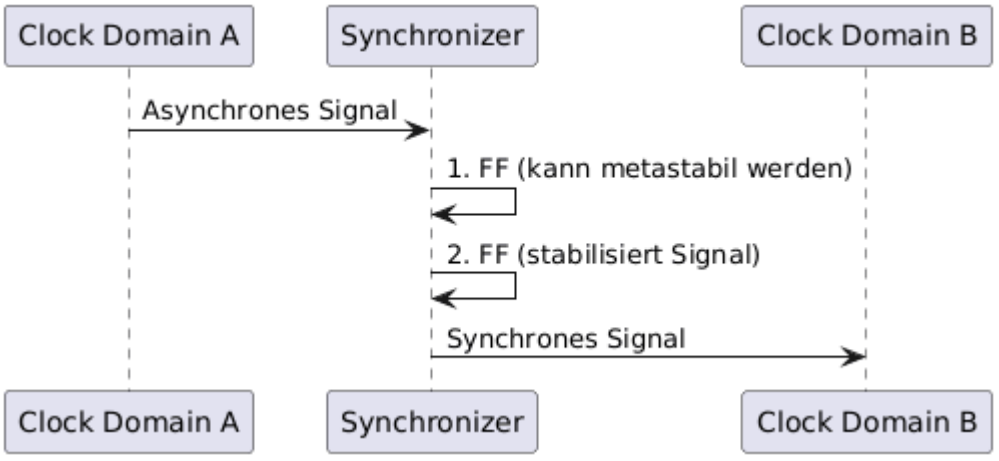
Vermeidungsstrategien

1. Synchrones Design: - Alle Zustandsübergänge erfolgen mit dem Takt - Vermeidung asynchroner Rückkopplungen - Verwendung von Flip-Flops statt Latches

2. Timing-Analyse: - Statische Timing-Analyse (STA) - Worst-case-Betrachtungen - Margin-Reserven einplanen

3. Clock Domain Crossing: - Synchronizer für Signale zwischen verschiedenen Clock-Domains - Doppel-Flip-Flop-Synchronizer für einzelne Bits - FIFO-Buffer für Datenströme

4. Reset-Strategien: - Synchrone Reset-Freigabe - Vermeidung von Reset-Race-Conditions



Praktische Designregeln: - Minimierung der kombinatorischen Logik zwischen Flip-Flops - Verwendung einheitlicher Clock-Flanken - Sorgfältige Placement & Routing bei Hardware-Implementierung - Pipeline-Register zur Pfadverkürzung

Debugging von Timing-Problemen

Symptome:

- Sporadische Fehlfunktionen
- Temperatur- oder spannungsabhängige Probleme
- Funktionsfehler bei hohen Taktraten

Debug-Methoden:

- Logic-Analyzer zur Timing-Betrachtung
- Simulation mit realistischen Delays
- Hardware-in-the-Loop-Tests
- Stress-Tests bei verschiedenen Betriebsbedingungen

Übungsaufgaben

- Entwerfen Sie einen Sequenzdetektor, der für ein alternatives Muster, als das behandelte die Evaluation eines Datenstromes übernimmt.