Digitaltechnik
Wintersemester 2017/2018
8. Vorlesung

TECHNISCHE UNIVERSITÄT DARMSTADT

Inhalt

Einleitung

Überblick der heutigen Vorlesung

- Endliche Zustandsautomaten
  - Konzept, Notationen und Anwendungsbeispiele
  - Moore vs. Mealy
  - Zerlegen von Zustandsautomaten

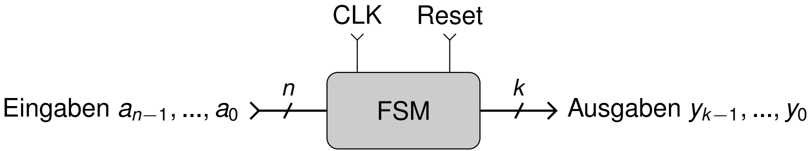
Harris 2013
Kap. 3.1-3.3
Seite 103 - 117

Konzept, Notationen und Anwendungsbeispiele

Endliche Zustandsautomaten
Finite State Machines (FSM)

TECHNISCHE UNIVERSITÄT DARMSTADT

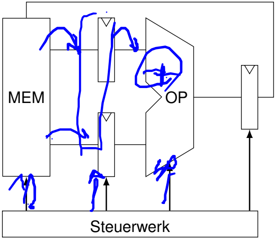
- synchrone sequentielle Schaltungen mit
  - *n* Eingabebits
  - *k* Ausgabebits
  - *ein* interner Zustand (besteht aus $m \geq 1$ Bits)
  - Takt und Reset
- in jedem Takt (zur steigenden Flanke)
  - Reset aktiv $\rightarrow$ Zustand = Startzustand
  - Reset inaktiv $\rightarrow$ neuen Zustand und Ausgaben aus aktuellem Zustand und Eingaben berechnen



FSM Anwendungsbeispiele

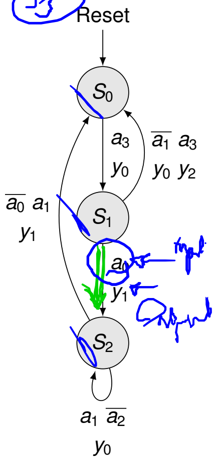
- Zahlenschloss (bspw. an Tresor)
  - Eingaben: Taste i gedrückt
  - Ausgaben: Schloss öffnen, Fehlermeldung anzeigen
  - Zusammenhang zwischen Zuständen: nur Öffnen, wenn letzte (4) Eingaben korrekt und in richtiger Reihenfolge

- Steuerwerk von Rechnern (Mikroarchitektur)
  - Eingaben: Bits des aktuellen Instruktionswortes
  - Ausgaben: Steuersignale für
    - Arithmetik (welche Operation)
    - Speicher (welche Operanden)
  - Zusammenhang zwischen Zuständen: bspw. in Pipeline-Stufen
- vieles mehr (sehr häufig verwendetes Konzept)



FSM Diagramme als gerichtete Graphen

- Zustände als Knoten
  - symbolische Namen (typ. S_0, S_1, ...)
  - binäre Zustandskodierung ist unabhängiges Problem
- Eingangsbedingung für Zustandsübergang als Kante
  - als boole'scher Ausdruck (leere Bedingung entspricht 1)
  - Disjunktion aller ausgehenden Kanten ergibt 1
  - Keine zwei Kantenbedingung gleichzeitig erfüllbar
  - Vereinfachte Notation (keine Selbstschleifen):
    Zustand bleibt unverändert, wenn keine Bedinung erfüllt
- genau eine Kante ohne Start-Punkt für Reset
- Ausgaben
  - an Kanten (Mealy-Automat)
  - in Zuständen (Moore-Automat)
  - als vollständiger boole'scher Ausdruck (Minterm)
  - oder nur gesetzte Ausgaben



FSM Beispiel für Ampelsteuerung

TECHNISCHE UNIVERSITÄT DARMSTADT

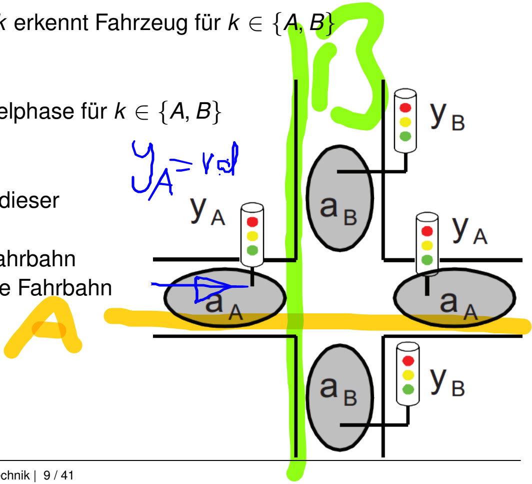
$a_A = 1$

- Eingänge:
  - $a_k = 1 \Leftarrow$ Induktionsschleife k erkennt Fahrzeug für $k \in \{A, B\}$
- Ausgänge
  - $y_k \in \{$rot,grün,gelb$\} \Rightarrow$ Ampelphase für $k \in \{A, B\}$

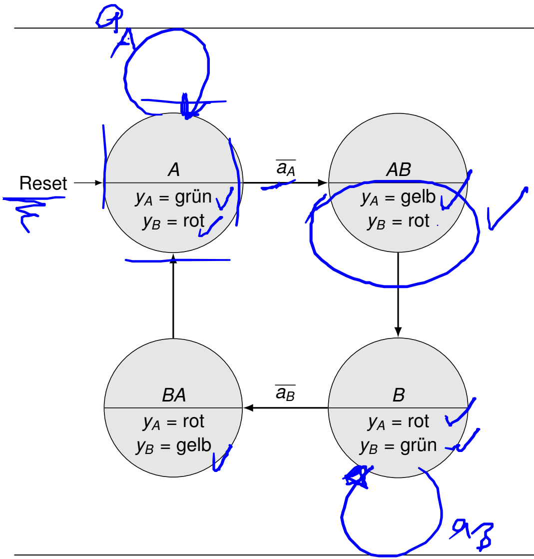
$\Rightarrow$ FSM für Bedarfssteuerung

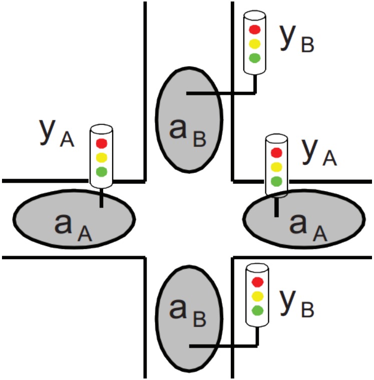
- halte Spur grün, solange auf dieser Fahrzeuge erkannt werden
- ansonsten schalte aktuelle Fahrbahn über gelb nach rot und andere Fahrbahn auf grün

y_A = rot

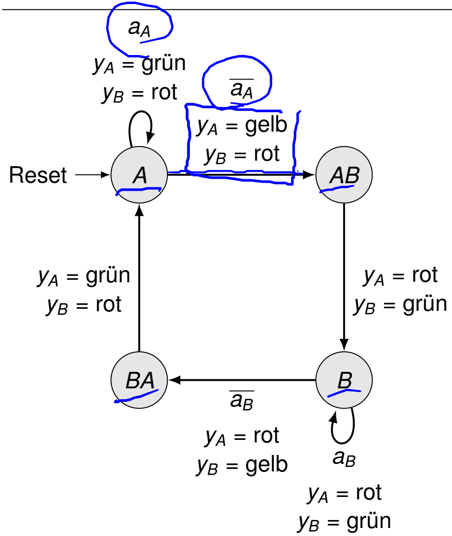


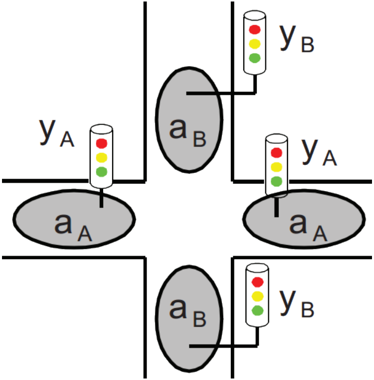
Moore-Automat für Ampelsteuerung





Mealy-Automat für Ampelsteuerung





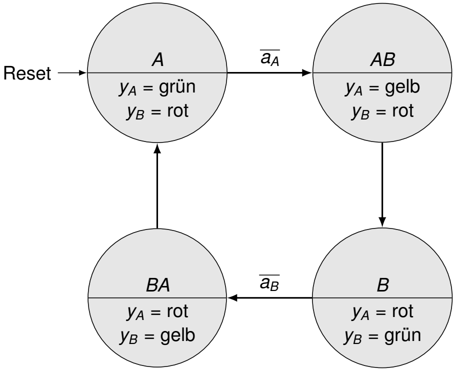
Zustandsübergangs- und Ausgabetabelle

- kompaktere (maschinenlesbare) Darstellung
- kann noch mit abstrakten Zuständen und Ausgaben arbeiten
- kann Don't Cares verwenden
- Kurzschreibweise
  - aktueller Zustand S
  - nächster Zustand S'
- *Achtung:* implizite Bedingungen (bspw. Selsbstschleifen) beim Ableiten aus Diagrammen beachten

Zustandsübergangs- und Ausgabetabelle für Moore-Automat der Ampelsteuerung

S	a_A	a_B	S'
A	1	*	A
A	0	*	AB
AB	*	*	B
B	*	1	B
B	*	0	BA
BA	*	*	A

S	y_A	y_B
A	grün	rot
AB	gelb	rot
B	rot	grün
BA	rot	gelb

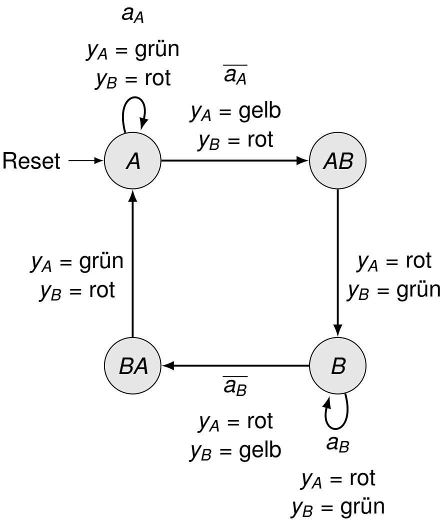


Zustandsübergangs- und Ausgabetabelle für Mealy-Automat der Ampelsteuerung

TECHNISCHE UNIVERSITÄT DARMSTADT

S	a_A	a_B	S'
A	1	*	A
A	0	*	AB
AB	*	*	B
B	*	1	B
B	*	0	BA
BA	*	*	A

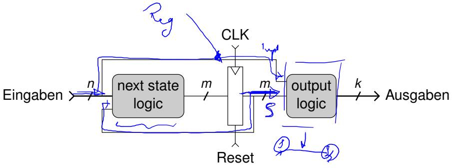
S	a_A	a_B	y_A	y_B
A	1	*	grün	rot
A	0	*	gelb	rot
AB	*	*	rot	grün
B	*	1	rot	grün
B	*	0	rot	gelb
BA	*	*	grün	rot

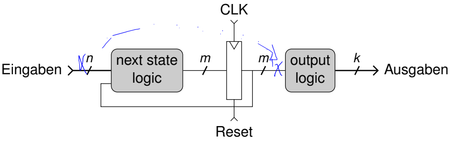


FSM als synchrone sequentielle Schaltungen

- Zustandsregister
  - speichert aktuellen Zustand
  - überimmt nächsten Zustand bei Taktflanke
- kombinatorische Logik realisiert
  - Zustandübergangstabelle („next state logic“)
  - Ausgangstabelle („output logic“)

⇒ binäre Kodierung der Zustände und Ein-/Ausgaben notwendig





Zustandskodierung $cs : S \to \mathbb{B}^m$

A → 00
B → 01

- weist jedem Zustand einen m Bit breiten Wert zu
- kann idR. frei gewählt werden (da nach außen nicht sichtbar)
- bspw. „Durchnummerieren“: $cs(S_k) = (s_{m-1} \ldots s_0)$ mit $u_{2,m}(s_{m-1} \ldots s_0) = k$
- manchmal führen aber andere Kodierungen zu effizienterer kombinatorischer Logik, auch wenn mehr Zustandsbits benötigt werden
  - One-Hot ← 001 | ~~011~~ | 010 ✓
  - bestehende Ausgabekodierung (wenn jeder Zustand eine spezifische Ausgabe verursacht)
- Kodierung der Ein-/Ausgänge ist idR. von der Anwendung vorgegeben
  - kann ansonsten für jede Ein/Ausgabe spezifisch gewählt werden

Kodierte Tabellen für Moore-Automat der Ampelsteuerung

TECHNISCHE UNIVERSITÄT DARMSTADT

S	s_1	s_0
A	0	0
AB	0	1
B	1	0
BA	1	1

y_A	y_3	y_2
grün	0	0
gelb	0	1
rot	1	0

y_B	y_1	y_0
grün	0	0
gelb	0	1
rot	1	0

S	s_1	s_0	a_A	a_B	s'_1	s'_0
A	0	0	1	*	0	0
A	0	0	0	*	0	1
AB	0	1	*	*	1	0
B	1	0	*	1	1	0
B	1	0	*	0	1	1
BA	1	1	*	*	0	0

S	s_1	s_0	y_3	y_2	y_1	y_0
A	0	0	0	0	1	0
AB	0	1	0	1	1	0
B	1	0	1	0	0	0
BA	1	1	1	0	0	1

- $n = 2$ Eingängssignale, $m = 2$ Zustandsbits, $k = 4$ Ausgabesignale

$\Rightarrow$ sechs bool'sche Funktionen aus Wahrheitswertetabellen ableiten

Minimierte kombinatorische Logik für Moore-Automat der Ampelsteuerung

ampel/state.esp

```
.i 4
.o 2
001- 00
000- 01
01-- 10
10-1 10
10-0 11
11-- 00
```

S_0 a_1 a_0 S_1' S_0'

Minimierte kombinatorische Logik für Moore-Automat der Ampelsteuerung

`ampel/state.esp`

```
.i 4
.o 2
001- 00
000- 01
01-- 10
10-1 10
10-0 11
11-- 00
```

`espresso ampel/state.esp`

```
.i 4
.o 2
.p 4
10-0 01
000- 01
01-- 10
10-- 10
.e
```

`ampel/output.esp`

```
.i 2
.o 4
00 0010
01 0110
10 1000
11 1001
```

$$s_1' = s_1 \oplus s_0$$

$$s_0' = \underbrace{s_1\,\overline{s_0}\,\overline{a_B}} + \underbrace{\overline{s_1}\,\overline{s_0}\,\overline{a_A}}$$

Minimierte kombinatorische Logik für Moore-Automat der Ampelsteuerung

TECHNISCHE UNIVERSITÄT DARMSTADT

ampel/state.esp

```
.i 4
.o 2
001- 00
000- 01
01-- 10
10-1 10
10-0 11
11-- 00
```

espresso ampel/state.esp

```
.i 4
.o 2
.p 4
10-0 01
000- 01
01-- 10
10-- 10
.e
```

ampel/output.esp

```
.i 2
.o 4
00 0010
01 0110
10 1000
11 1001
```

espresso ampel/output.esp

```
.i 2
.o 4
.p 4
01 0100
11 0001
1- 1000
0- 0010
.e
```

$$s_1' = s_1 \oplus s_0$$

$$s_0' = s_1\ \overline{s_0}\ \overline{a_B} + \overline{s_1}\ \overline{s_0}\ \overline{a_A}$$

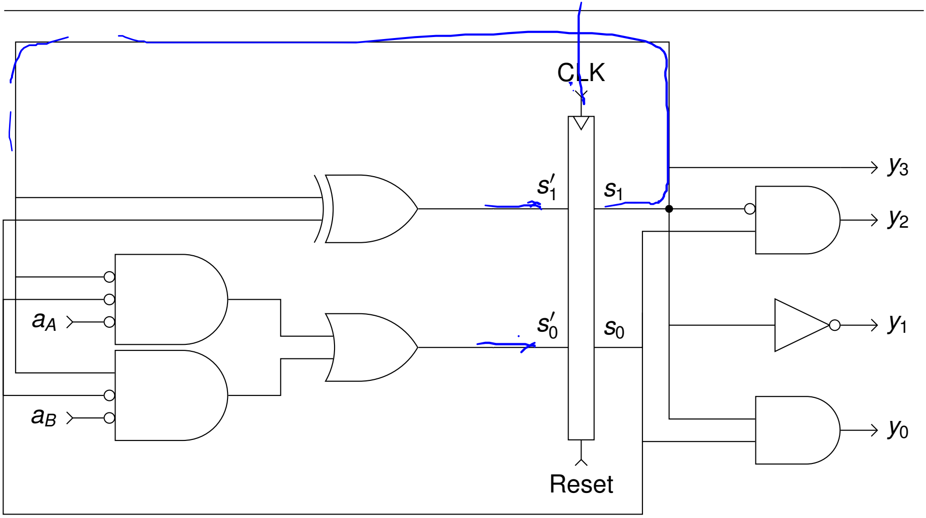
$$y_3 = s_1$$

$$y_2 = \overline{s_1}\ s_0$$

$$y_1 = \overline{s_1}$$

$$y_0 = s_1\ s_0$$

Schaltplan
für Moore-Automaten der Ampelsteuerung



Zusammenfassung FSM Entwurfverfahren

- definiere Ein- und Ausgänge
- wähle zwischen Moore- und Mealy-Automat
- zeichne Zustandsdiagramm
- kodiere Zustände (und ggf. Ein-/Ausgänge)
- stelle Zustandsübergangstabelle auf
- stelle boole'sche Gleichungen für Zustandsübergangs- und Ausgangslogik unter Ausnutzung von Don't Cares auf
- entwerfe Schaltplan: Gatter + Register

Mealy vs. Moore

Mealy vs. Moore

- für Ampelsteuerung war Moore-Automat effizienterer
- das ist aber nicht allgemein so

⇒ muss von Fall zu Fall neu bewertet werden

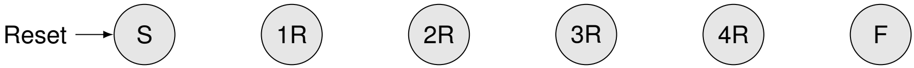
- in der Regel
  - Moore besser, wenn Ausgaben statisch
  - Mealy besser, wenn Ausgaben kurzfristige Aktionen auslösen
  - Mealy reagiert schneller auf Änderungen der Eingabe
- Verdeutlichung durch weitere Beispiele

FSM Beispiel für Zahlenschloss

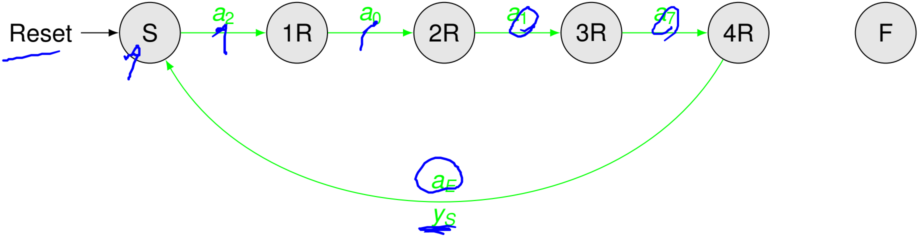
4

- Eingänge:
  - $a_k = 1 \Leftarrow$ Taste k gedrückt für $0 < k \leq 9$
  - $a_C = 1 \Leftarrow$ Taste „Cancel“ gedrückt
  - $a_E = 1 \Leftarrow$ Taste „Enter“ gedrückt
- Ausgänge
  - $y_S = 1 \Rightarrow$ Schloss entriegeln
  - $y_F = 1 \Rightarrow$ Fehlermeldung angezeigen
- Vereinfachungen
  - Zustandsübergang nur dann, wenn überhaupt eine Taste gedrückt
  - immer nur eine Taste gleichzeitig aktivierbar
- Passwort: 2017
- *Achtung:* Fehlermeldung nicht direkt bei erster falscher Ziffer zeigen

Mealy-Automat für Zahlenschloss

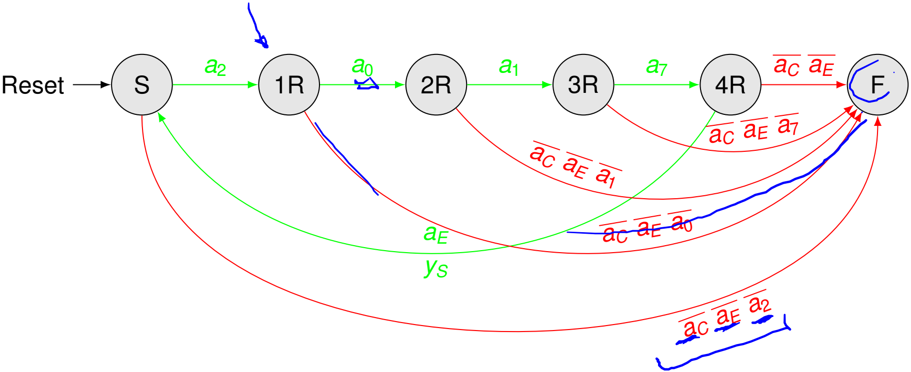


Mealy-Automat für Zahlenschloss

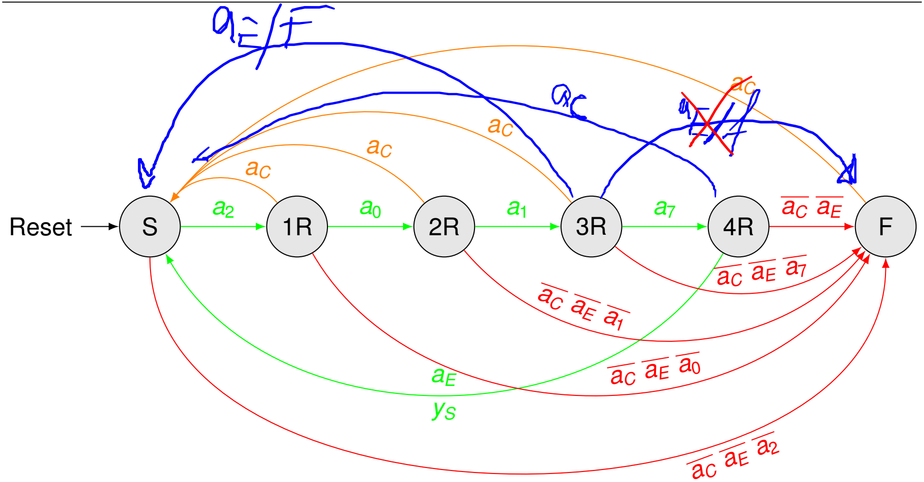


Mealy-Automat für Zahlenschloss

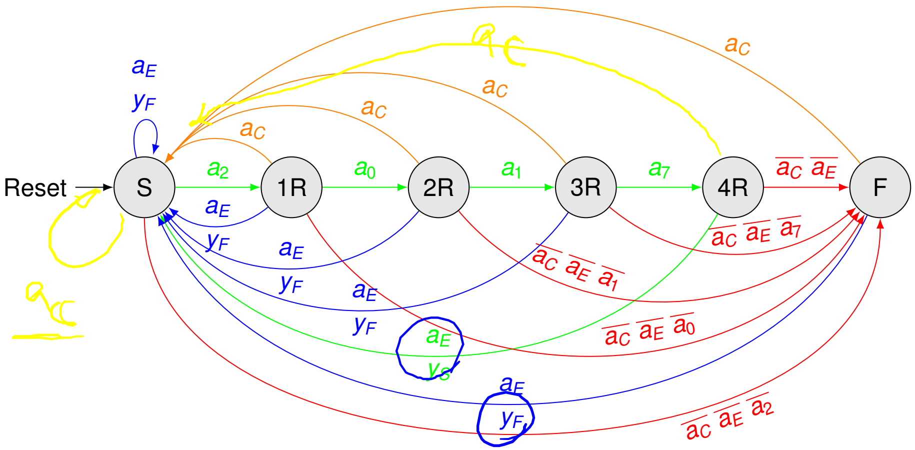
2017



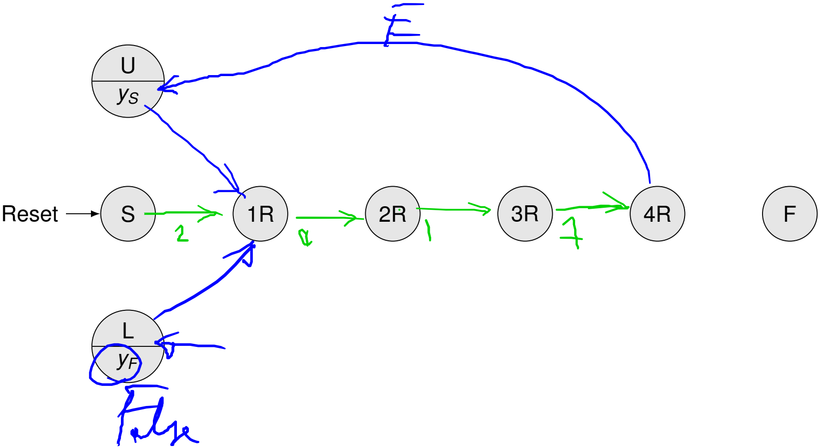
Mealy-Automat für Zahlenschloss



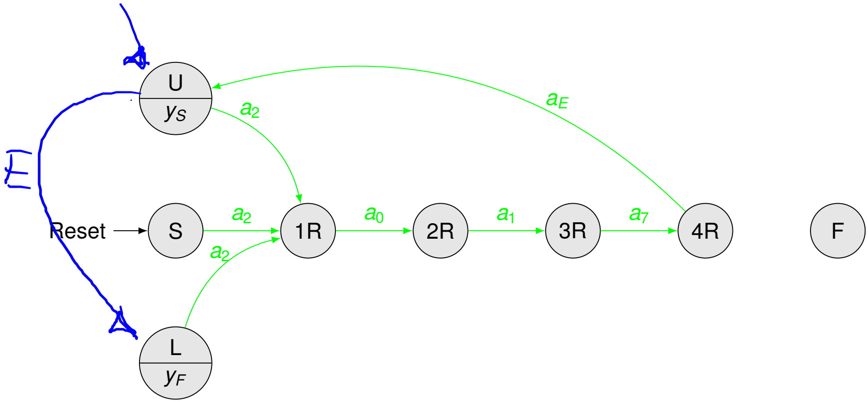
Mealy-Automat für Zahlenschloss



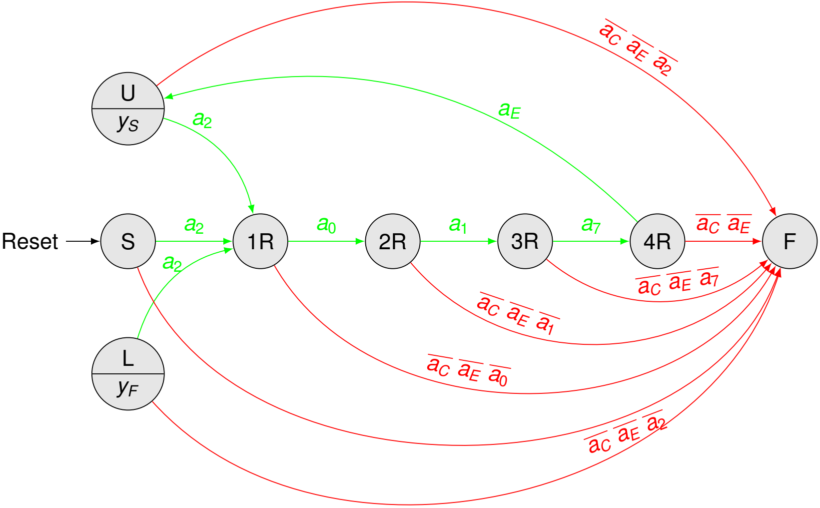
Moore-Automat für Zahlenschloss



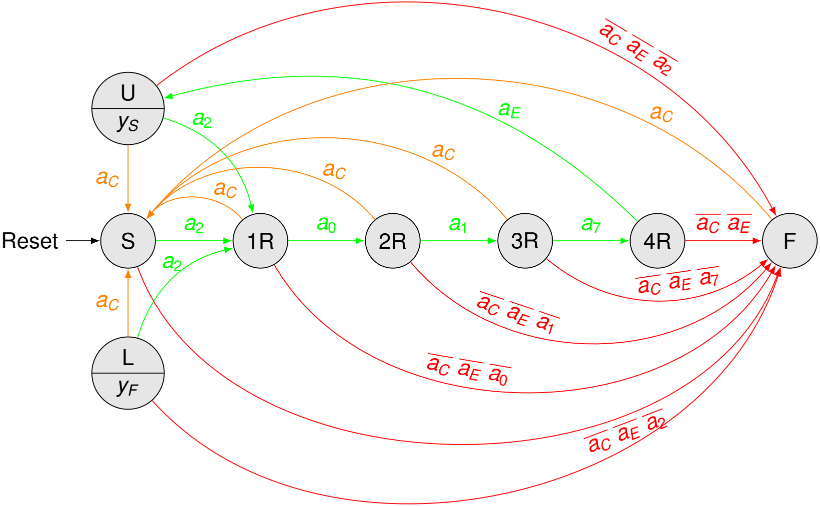
Moore-Automat für Zahlenschloss



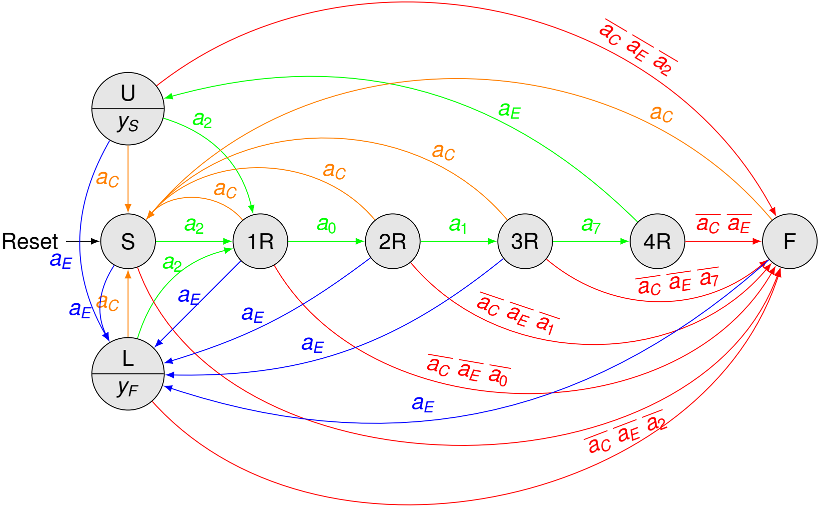
Moore-Automat für Zahlenschloss



Moore-Automat für Zahlenschloss



Moore-Automat für Zahlenschloss



Mealy vs. Moore für Zahlenschloss

TECHNISCHE UNIVERSITÄT DARMSTADT

- Moore-Automat braucht zwei zusätzliche Zustände, um die beiden unterschiedlichen Übergänge zurück in den Ausgangszustand (nach richtiger oder falscher Eingabe) voneinander zu unterscheiden
- Ausgaben beschreiben eher
  - Aktionen (Schloss öffnen, Fehler anzeigen) als
  - Zustände (Schloss ist geöffnet, Fehler wird angezeigt)

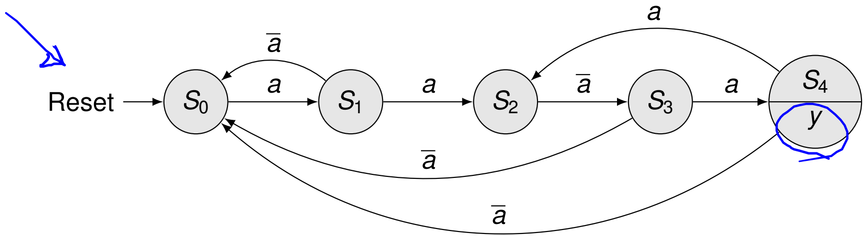
$\Rightarrow$ Mealy-Automat besser geeignet

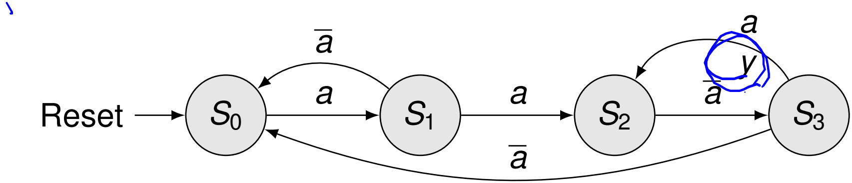
Weiteres Beispiel: Mustererkennung

TECHNISCHE UNIVERSITÄT DARMSTADT

- typisch in Bild- und Textanalyse (bspw. Suche nach regulären Ausdrücken)
- bspw.: Erkenne Bitfolge „1101“ in zufälliger Bitsequenz
- Eingänge: das nächste Bit $a \in \mathbb{B}$
- Ausgabe: $y = 1 \Rightarrow$ gesuchte Bitfolge erkannt

Moore- und Mealy-Automat für 1101 Mustererkennung





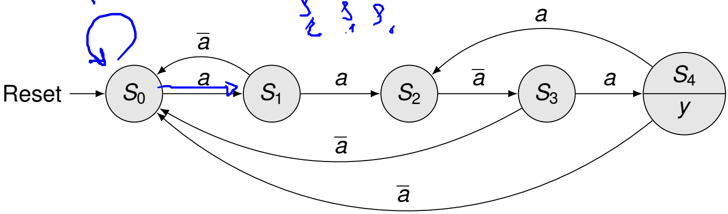
Moore-Automat für 1101 Mustererkennung: Zustandübergangs- und Ausgabetabellen

TECHNISCHE UNIVERSITÄT DARMSTADT

S	a	S'
S_0	0	S_0
S_0	1	S_1
S_1	0	S_0
S_1	1	S_2
S_2	0	S_3
S_2	1	S_2
S_3	0	S_0
S_3	1	S_4
S_4	0	S_0
S_4	1	S_2

S	s_2	s_1	s_0
S_0	0	0	0
S_1	0	0	1
S_2	0	1	0
S_3	0	1	1
S_4	1	0	0

S	y
S_0	0
S_1	0
S_2	0
S_3	0
S_4	1



Moore-Automat für 1101 Mustererkennung: Logikgenerierung mit vielen Don’t Cares

TECHNISCHE UNIVERSITÄT DARMSTADT

pattern/moore/state.esp

```
.i 4
.o 3
0000 000
0001 001
0010 000
0011 010
0100 011
0101 010
0110 000
0111 100
1000 000
1001 010
1010 ---
1011 ---
1100 ---
1101 ---
1110 ---
1111 ---
```

espresso pattern/moore/state.esp

```
.i 4
.o 3
.p 6
0001 001
-100 001
-111 100
-011 010
1--1 010
-10- 010
.e
```

$$s_2' = s_1 \; s_0 \; a$$

$$s_1' = \overline{s_1} \; s_0 \; a + s_2 \; a + s_1 \; \overline{s_0}$$

$$s_0' = \overline{s_2} \; \overline{s_1} \; \overline{s_0} \; a + s_1 \; \overline{s_0} \; \overline{a}$$

pattern/moore/output.esp

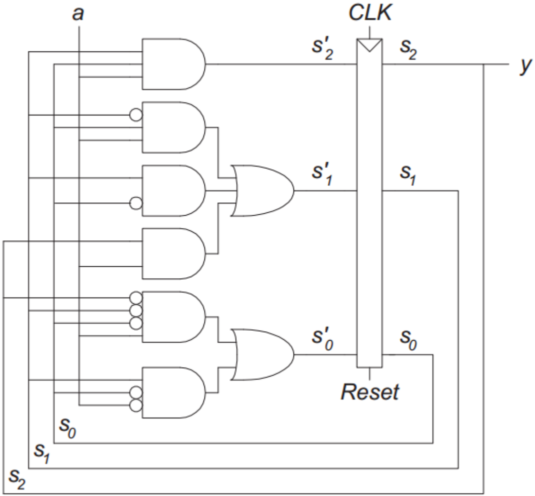
```
.i 3
.o 1
000 0
001 0
010 0
011 0
100 1
101 -
110 -
111 -
```

espresso pattern/moore/output.esp

```
.i 3
.o 1
.p 1
1-- 1
.e
```

$$y = s_2$$

Moore-Automat für 1101 Mustererkennung: Schaltwerk

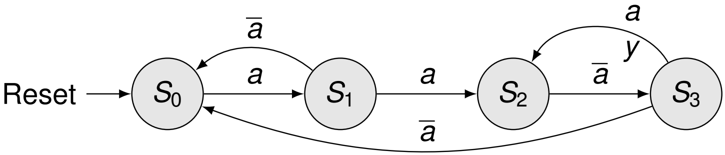


Mealy-Automat für 1101 Mustererkennung: Zustandübergangs- und Ausgabetabellen

S	a	S'
S_0	0	S_0
S_0	1	S_1
S_1	0	S_0
S_1	1	S_2
S_2	0	S_3
S_2	1	S_2
S_3	0	S_0
S_3	1	S_1

S	s_1	s_0
S_0	0	0
S_1	0	1
S_2	1	0
S_3	1	1

S	a	y
S_0	0	0
S_0	1	0
S_1	0	0
S_1	1	0
S_2	0	0
S_2	1	0
S_3	0	0
S_3	1	1



Mealy-Automat für 1101 Mustererkennung: Logikgenerierung ohne Don't Cares

pattern/mealy/state.esp

```
.i 3
.o 2
000 00
001 01
010 00
011 10
100 11
101 10
110 00
111 01
```

espresso pattern/mealy/state.esp

```
.i 3
.o 2
.p 5
011 10
100 01
001 01
111 01
10- 10
.e
```

pattern/mealy/output.esp

```
.i 3
.o 1
000 0
001 0
010 0
011 0
100 0
101 0
110 0
111 1
```

espresso pattern/mealy/output.esp

```
.i 3
.o 1
.p 1
111 1
.e
```

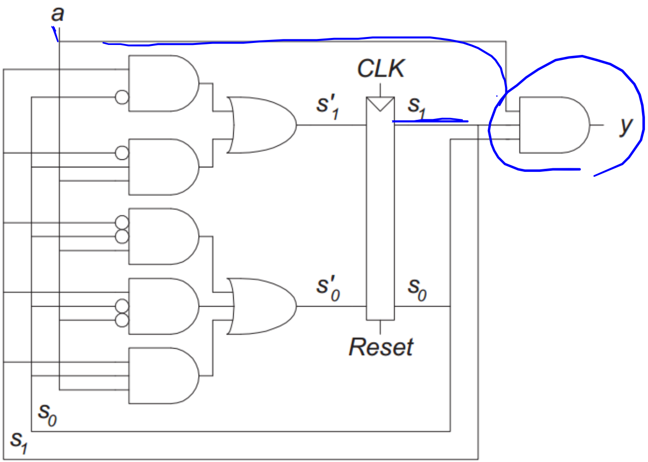
$$y = s_2\ s_1\ a$$

$$s_1' = \overline{s_1}\ s_0\ a + s_1\ \overline{s_0}$$

$$s_0' = s_1\ \overline{s_0}\ \overline{a} + \overline{s_1}\ \overline{s_0}\ a + s_1\ s_0\ a$$

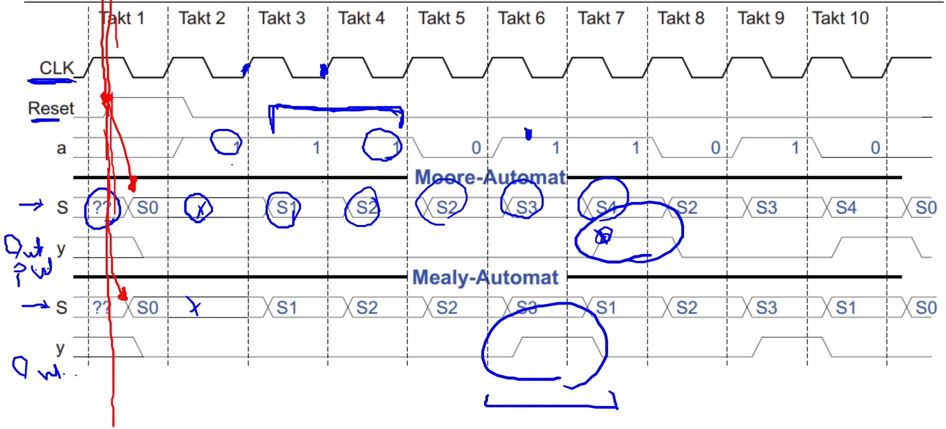
Mealy-Automat für 1101 Mustererkennung: Schaltwerk

TECHNISCHE UNIVERSITÄT DARMSTADT



Moore- und Mealy-Automaten: Zeitverhalten

1101

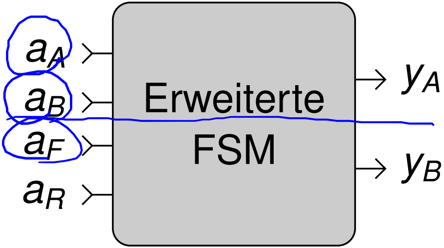


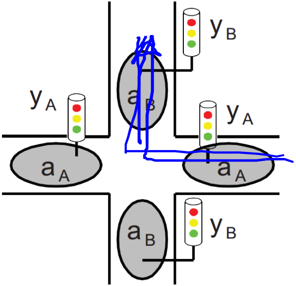
- Mealy-Automat erkennt Muster einen Takt früher

Zerlegen von Zustandsautomaten

Zerlegen von Zustandsautomaten (FSM Dekomposition)

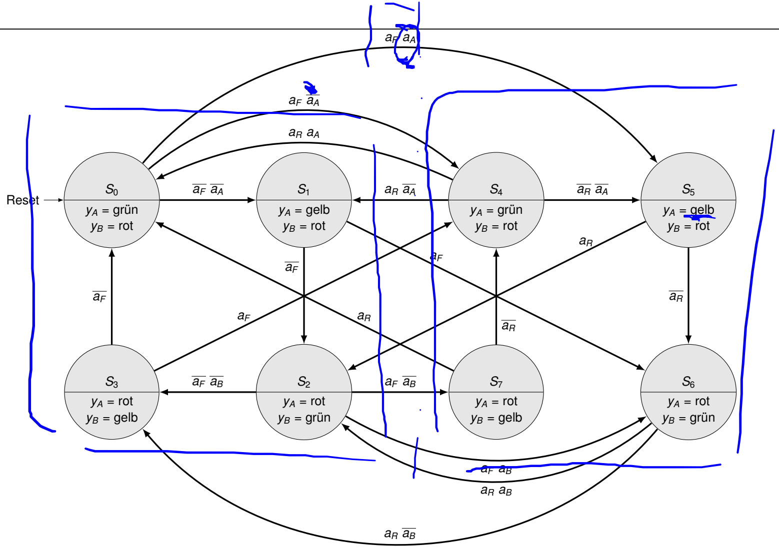
- Aufteilen komplexer FSMs in einfachere interagierende FSMs
- Beispiel: Ampelsteuerung mit Modus für Festumzüge (Ampel B bleibt permanent grün)
  - FSM bekommt zwei weitere Eingänge: a_F, a_R
  - $a_F = 1 \Rightarrow$ aktiviert Festumzugsmodus
  - $a_R = 1 \Rightarrow$ deaktiviert Festumzugsmodus



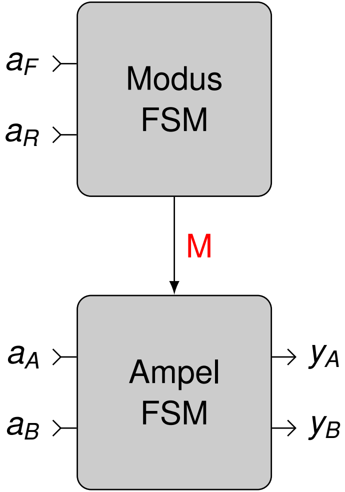


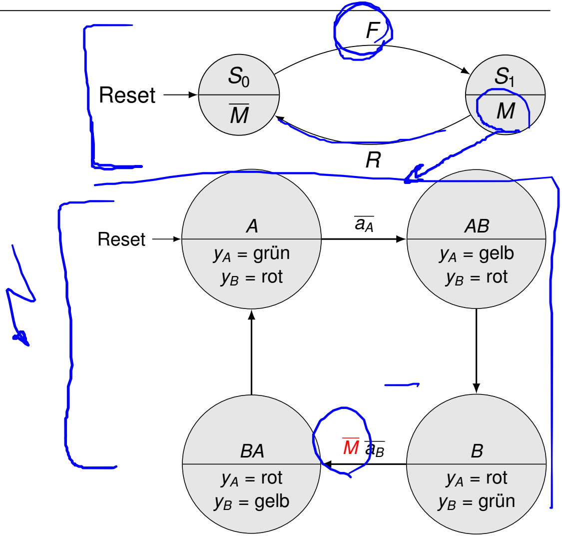
Unzerlegte FSM

TECHNISCHE UNIVERSITÄT DARMSTADT



Zerlegung in kommunizierende FSMs





Zusammenfassung

Zusammenfassung und Ausblick

- Endliche Zustandsautomaten
  - Konzept, Notationen und Anwendungsbeispiele
  - Moore vs. Mealy
  - Zerlegen von Zustandsautomaten

- Nächste Vorlesung behandelt
  - Zeitverhalten Sequentieller Schaltungen
  - Parallelität