OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

Synchrone (sequentielle) Logik (synclogic)

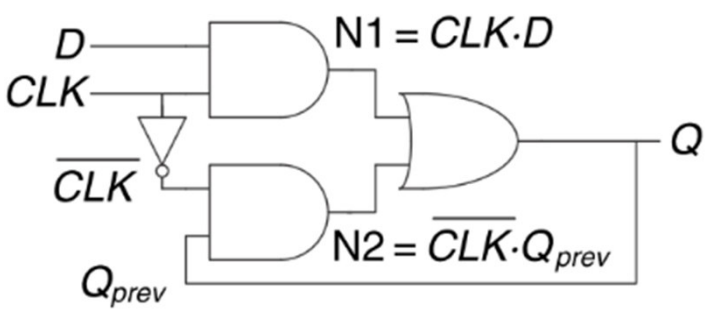
Synchrone Logik - Einleitung und Begriffsklärung

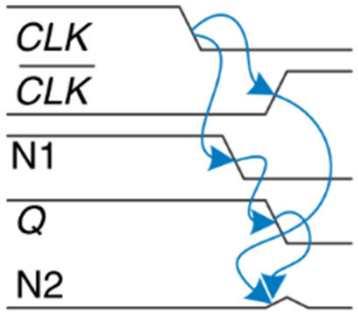
- Bisher haben wir uns kombinatorische Logik angesehen
- Kombinatorische Logik lässt sich mit bool'schen Funktionen beschreiben und der aktuelle Ausgang hängt nur von aktuellen Eingangswerten ab
- Bei synchroner oder sequentieller Logik hängt der aktuelle Ausgangswert neben aktuellen Eingangswerten und auch von vorhergehenden Eingangswerten ab
- Synchrone Logik hat also Speicher
- Entweder werden die vorherdegehenden Eingangswerte direkt gespeichert oder in eine kleinere Informationsmenge (state, Zustand) verdichtet
- Der Zustand ist eine Menge von Bits (auch state variables genannt), die alle Informationen (über die Vergangenheit) enthalten, die benötigt werden, das zukünftige Verhalten zu beschreiben
- Als Realisierung des Speichers dienen die bereits besprochen FFs bzw. Register

Synchrone Logik - Motivation I/II

- Asynchrone Schaltungsentwicklung beinhaltet auch direkte feedbacks (Rückführung) von den Ausgängen auf die Eingänge (cyclic paths, kombinatorische Schleifen)
- Solche Strukturen funktionieren oft nur bei bestimmten Temperaturen oder bestimmten angelegten Spannungen bei denen die Delays gerade passen
- (Fehler-)Analyse in solchen Schaltungen ist extrem zeitaufwendig und schwierig

Synchrone Logik - Motivation - Beispiel Asynchrone Logik (B)





- CLK=D=1 -> Q=1;
  (CLK=D=0 -> Q=1?: CLK=0-> N1=0, N2=1 -> Q=1;)
- Wenn aber Tpd_inv>>Tpd_and, Tpdp_or:
  CLK=D=0 -> Q=1?: N1=0, N2=0 -> Q=0!!!
  (N1und N2 fallen beide auf 0 bevor !CLK ansteigt -> schafft es unter diesen Bedingungen wegen dem UND nicht mehr 1 zu werden)

Synchrone Logik - Motivation II/II

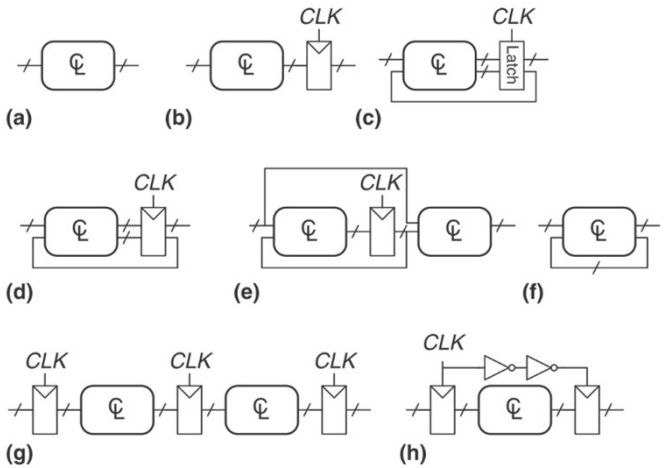
- Um solche Probleme zu vermeiden, ersetzt man diese direkten feedbacks durch feedbacks mit Registern bzw. FFs (synchrone Logik)
- Damit haben wir Sammlungen von kombinatorischer Logik mit Registern
- Die Register beinhalten den Zustand, der sich nur bei der Taktflanke ändern (auf Takt synchronisiert)
- Man spricht dann von synchroner sequentieller Logik oder kurz synchroner Logik

Synchrone Logik - Definition

- Synchrone Logik muss folgende Bedingungen erfüllen:
  - Jedes Schaltungselement ist entweder kombinatorische Logik oder ein Register/FF
  - Zumindest ein Element ist ein Register
  - Alle Register arbeiten mit demselben Taktsignal
  - Wenn es einen cyclic path gibt, muss dieser zumindest ein Register enthalten

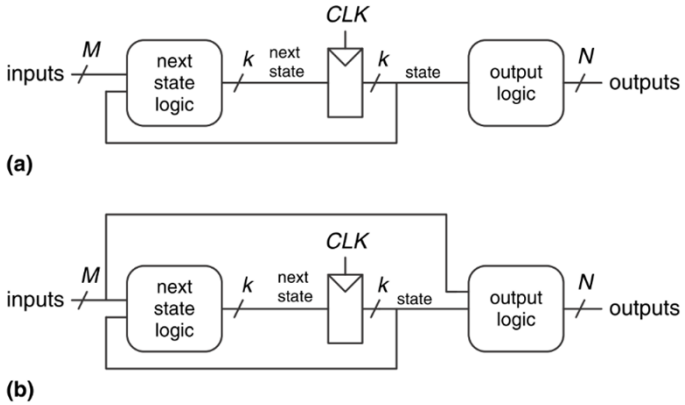
Synchrone Logik - Übung

- Welche Schaltungen sind synchrone Logik? Begründen Sie Ihre Wahl.



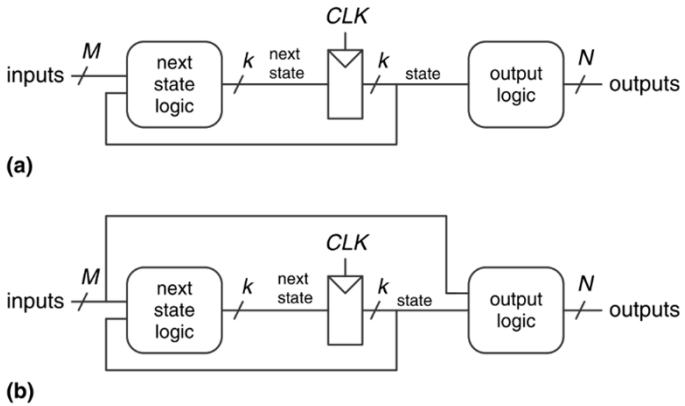
FSMs (Finite State Machines) - Übersicht I/II

- Der Schaltplan von synchroner Logik kann in Form von sogenannten finite state machines (FSM, endlicher Zustandsautomat) gezeichnet werden



- Der Name FSM kommt daher, dass eine Schaltung mit k Register-Bits (FFs) sich in einem von 2^k Zuständen befinden kann
- FSM hat zwei Blöcke von kombinatorischer Logik (next state logic und output logic) und ein Register, dass den Zustand speichert
- Bei jeder Taktflanke wechselt die FSM den Zustand zum nächsten Zustand (next state), der aus dem aktuellen Zustand (current state) und den Eingangssignalen berechnet wurde

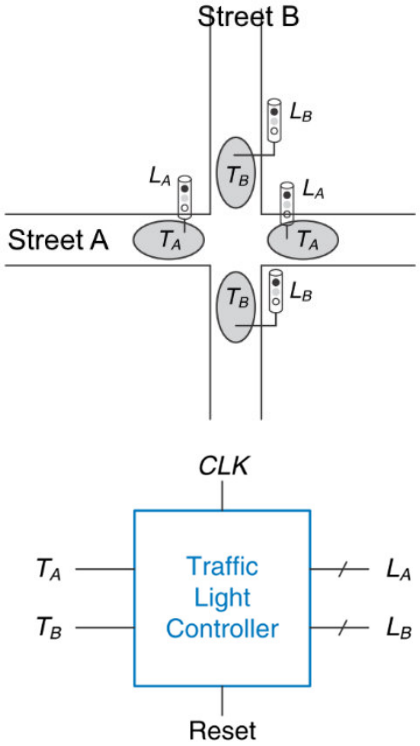
FSMs (Finite State Machines) - Übersicht II/II



- Es gibt zwei prinzipielle Klassen von FMS
  - a) Moore FSM:
    Die Ausgänge hängen nur von dem aktuellem Zustand ab
  - b) Mealy FSM:
    Die Ausgänge hängen von dem aktuellen Zustand sowie den Eingangssignalen ab
- Anwendung: FSMs sind der systematische Weg eine synchrone Schaltung zu einer gewünschten Funktion zu entwickeln

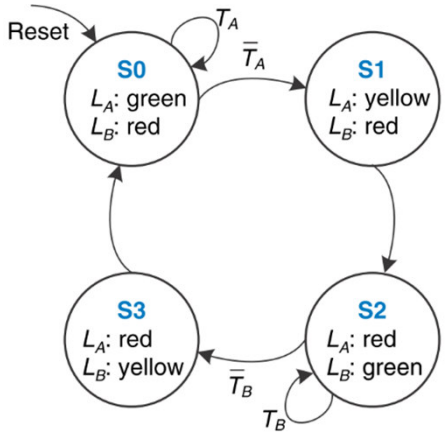
FSMs - Beispiel Ampelschaltung - Problemstellung - Funktionsbeschreibung

- Wenn das System einen Reset bekommt, ist La grün und Lb rot
- Alle 5s werden die Verkehrssensoren geprüft und neu entschieden was gemacht wird
- Solange es Verkehr gibt, werden die Ampeln nicht umgeschaltet
- z.B. Wenn La grün ist und es keinen Verkehr mehr auf Ta gibt, wird auf Lb umgeschaltet:
  - La wird auf gelb geschalten, dann nach 5s auf rot sowie Lb auf grün



FSMs - Beispiel Ampelschaltung - State chart

- Als nächsten Schritt zeichnet man einen state chart, um die Funktion zu erfassen und übersichtlich darzustellen



- Kreise repräsentieren Zustände
- Pfeile repräsentieren Zustandsübergänge (transitions)
- Die Zustandsübergänge finden bei der steigenden Flanke statt
- Der Pfeil aus dem Nichts mit dem Label Reset zeigt an, das das System diesen Zustand beim Reset annimmt und zwar egal in welchem Zustand das System vorher war
- Wenn mehrere Pfeile einen Zustand verlassen, zeigen die Labels den Auslöser für jeden Zustandsübergang (Beispiel S0)
  - Wenn es nur einen Pfeil ohne Label gibt, so wird dieser immer ausgeführt unabhängig von den Eingangssignalen (Beispiel S1)

FSMs - Beispiel Ampelschaltung - (State) transition table

- Üblicherweise schreibt man den state chart (Zustands-diagramm) dann in die state transition table (Zustands-übergangstabelle) um

Current State S	Inputs T_A	Inputs T_B	Next State S'
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

- Die state transition table gibt an, für welchen state und welche Eingangssinale sich welcher next state (S‘) ergibt
- ‚X‘ bedeutet „dont‘ care“ (-> next state hängt nicht von Eingangsignalen ab)
- Reset steht üblicherweise nicht in der Tabelle -> FFs mit Reset, der immer auf den state S0 geht

FSMs - Beispiel Ampelschaltung - (State) transition table - Encodings - Motivation

- Der state chart und die state transition haben abstrakte states {S0, S1, S2, S3} und abstrakte Ausgänge {rot, gelb, grün}
- Damit lässt sich keine Schaltung bauen
- Um eine echte Schaltung bauen zu können müssen den Zuständen und Ausgangssignalen Binärkodierungen (binary encodings) zugewiesen werden

FSMs - Beispiel Ampelschaltung - (State) transition table - Encodings

- In diesem Fall wurde die einfache Binärkodierung gewählt (einfach binär durchgezählt)
- Jeder state wird mit zwei Bits kodiert

Current State S	Inputs T_A	Inputs T_B	Next State S'
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

State	Encoding $S_{1:0}$
S0	00
S1	01
S2	10
S3	11

- Damit ergibt sich dass die kodierte state transition table:

Current State S_1	Current State S_0	Inputs T_A	Inputs T_B	Next State S'_1	Next State S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

FSMs - Beispiel Ampelschaltung - (State) transistion table - Optimization and equations

- Die kodierte state transition table lässt sich optimieren und daraus pro Ergebnissignal die bool'sche Gleichung gewinnen (aus Tabelle rauslesen oder KV-Diagramm)
- Es ergeben sich folgende Gleichungen für die Berechnung des next states:

$$S_1' = S_1 \oplus S_0$$

$$S_0' = \overline{S}_1 \overline{S}_0 \overline{T}_A + S_1 \overline{S}_0 \overline{T}_B$$

Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S_1'	S_0'
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

FSMs - Beispiel Ampelschaltung - Output - Encodings and table

- Ähnlich wie die states werden in diesem Fall auch die Ausgangssignale einfach binär kodiert
- Das Ausgangsignal La und Lb wird jeweils mit zwei Bits kodiert

Output	Encoding $L_{1:0}$
green	00
yellow	01
red	10

- Damit ergibt sich dann die kodierte output table

Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

FSMs - Beispiel Ampelschaltung - Output - Optimization and equations

- Die kodierte output table lässt sich optimieren und daraus pro Ausgangssignal die bool'sche Gleichung gewinnen (aus Tabelle rauslesen oder KV-Diagramm)

$$L_{A1} = S_1$$

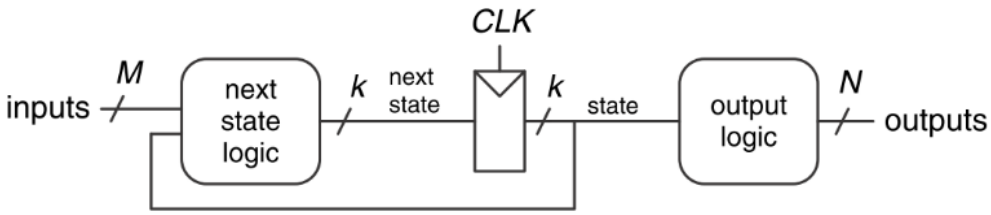
$$L_{A0} = \overline{S}_1 S_0$$

$$L_{B1} = \overline{S}_1$$

$$L_{B0} = S_1 S_0$$

FSMs - Beispiel Ampelschaltung - Schematic - Prinzipielles Vorgehen

- Wenn man die Gleichungen hat, ist es mit relativ kleinen Aufwand möglich, die Schaltung zu zeichnen
- Man geht dabei dreistufig vor



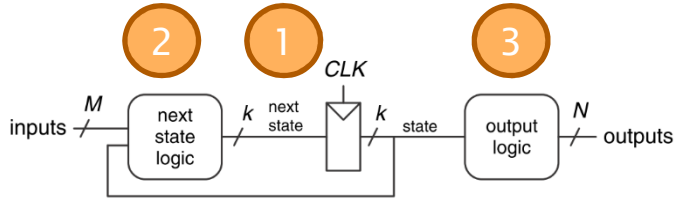
- Zuerst zeichnet man das Zustandsregister, dann die next state logic und dann die output logic

FSMs - Beispiel Ampelschaltung - Schematic

2

$$S_1' = S_1 \oplus S_0$$

$$S_0' = \overline{S}_1 \overline{S}_0 \overline{T}_A + S_1 \overline{S}_0 \overline{T}_B$$



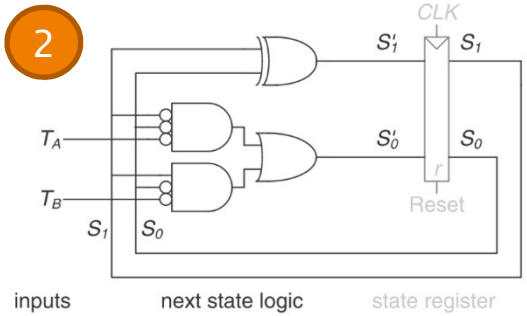
3

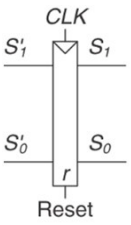
$$L_{A1} = S_1$$

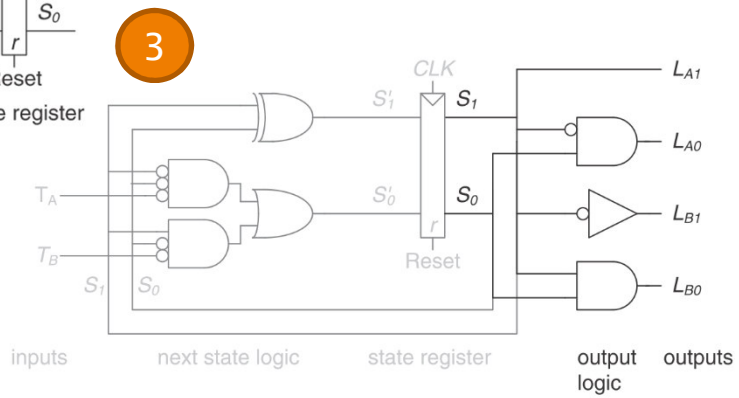
$$L_{A0} = \overline{S}_1 S_0$$

$$L_{B1} = \overline{S}_1$$

$$L_{B0} = S_1 S_0$$

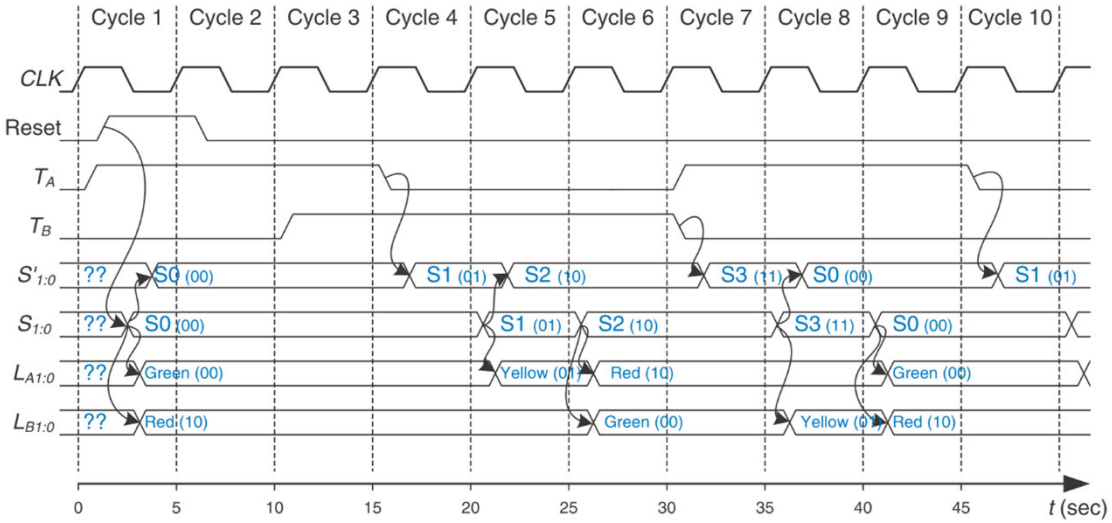






FSMs - Beispiel Ampelschaltung - Timing diagram

- Das folgende timing diagram zeigt die Ampel-FSM, wie sie durch eine Anzahl von states geht
- Der Takt (CLK) hat eine Periode von 5s so dass sich die Ampel höchstens alle 5s ändert



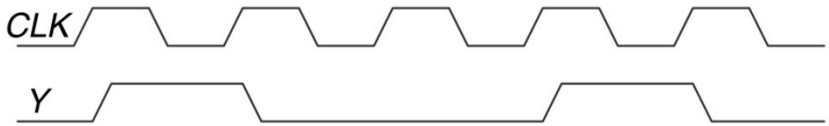
- Reset->S=S0, La=green, Lb=red; Ta=0->S=S1, La=yellow; S=S2->La=red, Lb=green; Tb=0-> S=S3,Lb=yellow; S=S0-> La=green, Lb=red

FSMs - State encodings

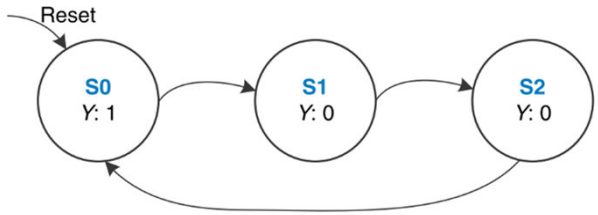
- Im Ampelbeispiel haben wir die Zustände einfach klassisch binär durchgezählt (binary encoding)
  - Bsp: S0=“00“; S1=“01“, S2=“10“
  - Vorteil: kompakte Kodierung
- Eine weitere Möglichkeit ist die „one-hot“ Kodierung
  - Pro state wird ein Bit verwendet, wobei immer nur ein Bit ‚1‘ ist
  - Bsp: S0=“001“, S1=“010“; S2=“100“
  - Nachteil: Braucht mehr FFs; Vorteil: next state und output Logik oft einfacher

FSMs - State encodings - Beispiel Nfach-Frequenzteiler - Problemstellung

- Anhand eines Nfach-Frequenzteilers (divide-by-N counter) wollen wir die Kodierung von „binär“ vs. „one-hot“ untersuchen
- Der Nfach-Frequenzteiler hat einen Ausgang und neben CLK und Reset keinen weiteren Eingang mehr
- Der Ausgang y ist 1 für einen Takt von N-Takten / Der Ausgang teilt die Taktfrequenz durch N
- Das timing diagram (auch waveform genannt) zeigt den konkreten Fall eines 3fach Frequenzteilers



FSMs - State encodings - Beispiel Nfach-Frequnzteiler - State chart and tables - Übung



- Erstellen Sie mit dem state chart die state tranistion table und die output table

Current State	Next State

Current State	Output

Teil 2

FSMs - State encodings - Beispiel Nfach-Frequenzteiler - Binary encoding

Current State	Next State
S0	S1
S1	S2
S2	S0

State	Binary Encoding S_1	S_0
S0	0	0
S1	0	1
S2	1	0

Current State S_1	S_0	Next State S_1'	S_0'
0	0	0	1
0	1	1	0
1	0	0	0

$$S_1' = \text{So}$$
$$S_0' = \overline{S}_1\overline{S}_0$$
$$Y = \overline{S}_1\overline{S}_0$$

- Das Aufstellen der output table sowie das Ableiten der Gleichung erfolgt analog zur state transition table

FSMs - State encodings - Beispiel Nfach-Frequenzteiler - One-hot encoding

Current State	Next State
S0	S1
S1	S2
S2	S0

State	One-Hot Encoding		
	S_2	S_1	S_0
S0	0	0	1
S1	0	1	0
S2	1	0	0

Current State			Next State		
S_2	S_1	S_0	S_2'	S_1'	S_0'
0	0	1	0	1	0
0	1	0	1	0	0
1	0	0	0	0	1

$$S_2' = S_1$$
$$S_1' = S_0$$
$$S_0' = S_2$$
$$Y = S_0$$

- Das Aufstellen der output table sowie das Ableiten der Gleichung erfolgt analog zur state transition table

FSMs - State encodings - Beispiel Nfach-Frequenzteiler -Vergleich encoding binär vs. one-hot - Übung

- Zeichnen Sie die Schaltung der Binärkodierten FSM und der one-hot kodierten FSM

- Binärkodierung

$$S_1' = \text{So}$$
$$S_0' = \overline{S}_1 \overline{S}_0$$

$$Y = \overline{S}_1 \overline{S}_0$$

- One-hot Kodierung

$$S_2' = S_1$$
$$S_1' = S_0$$
$$S_0' = S_2$$

$$Y = S_0$$

FSMs - Moore and Mealy

- Bisher haben wir nur Moore-FSMs betrachtet, bei denen die Ausgänge nur von dem Zustand abhängen
- Deshalb werden bei Moore FSMs die Ausgänge auch in die Zustandskreise geschrieben

- Bei Mealy FSMs können die Ausgänge sowohl von den Zuständen als auch von den Eingängen abhängen
- Die Ausgänge werden auf die Pfeile statt in die Kreise geschrieben

FSMs - Moore and Mealy - Beispiel Erkennen einer Bitsequenz - Problemstellung

- Eine FSM hat einen Eingang A, in dem jeden Takt ein neues Bit reingeschoben wird
- Die FSM soll dabei das Bitmuster „01“ erkennen und wenn es erkannt worden ist für einen Takt eine 1 an dem Ausgang y ausgeben

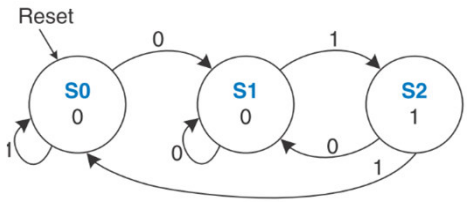
FSMs - Moore and Mealy - Beispiel Erkennen einer Bitsequenz - Moore state chart - Übung

- Eine FSM hat einen Eingang A, in dem jeden Takt ein neues Bit reingeschoben wird
- Die FSM soll dabei das Bitmuster „01“ erkennen und wenn es erkannt worden ist für einen Takt eine 1 an dem Ausgang y ausgeben

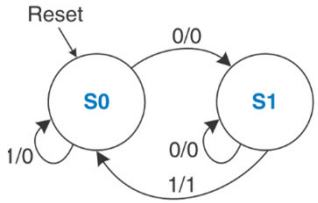
- Zeichen Sie den Moore-FSM state chart

FSMs - Moore and Mealy - Beispiel Erkennen einer Bitsequenz - State charts

- Moore



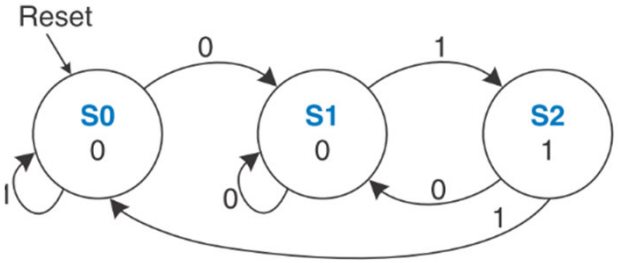
- Mealy



- Jeder Pfeil hat den Syntax A/Y, was bedeutet, dass wenn das Ereignis A eingetroffen ist, dieser Zustandsübergang aktiv wird und dabei Y ausgeben wird

FSMs - Moore and Mealy - Beispiel Erkennen einer Bitsequenz - Transition tables

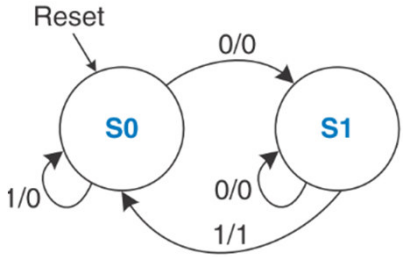
- Moore



Current State S	Input A	Next State S'
S0	0	S1
S0	1	S0
S1	0	S1
S1	1	S2
S2	0	S1
S2	1	S0

Current State S	Output Y
S0	0
S1	0
S2	1

- Mealy



Current State S	Input A	Next State S'	Output Y
S0	0	S1	0
S0	1	S0	0
S1	0	S1	0
S1	1	S0	1

FSMs - Moore and Mealy - Beispiel Erkennen einer Bitsequenz - Transition tables encoded

- Für das encoding wird hier die einfache Binärkodierung verwendet

- Moore

Current State S	Input A	Next State S'
S0	0	S1
S0	1	S0
S1	0	S1
S1	1	S2
S2	0	S1
S2	1	S0

Current State S	Output Y
S0	0
S1	0
S2	1

Current State S_1	S_0	Input A	Next State S'_1	S'_0
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

Current State S_1	S_0	Output Y
0	0	0
0	1	0
1	0	1

- Mealy

Current State S	Input A	Next State S'	Output Y
S0	0	S1	0
S0	1	S0	0
S1	0	S1	0
S1	1	S0	1

Current State S_0	Input A	Next State S'_0	Output Y
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

FSMs - Moore and Mealy - Beispiel Erkennen einer Bitsequenz - Equations

- Der nicht existierende Zustand „11“ ist in den Moore-Tabellen nicht enthalten, wird aber zur Optimierung verwendet (don’t care)

Moore

Current State S_1	Current State S_0	Input A	Next State S_1'	Next State S_0'
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

Current State S_1	Current State S_0	Output Y
0	0	0
0	1	0
1	0	1

$$S_1' = S_0 A$$
$$S_0' = \overline{A}$$

$$Y = S_1$$

Mealy

Current State S_0	Input A	Next State S_0'	Output Y
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

$$S_0' = \overline{A}$$

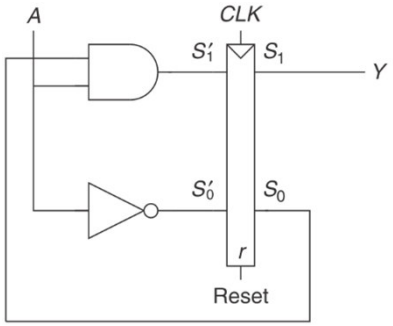
$$Y = S_0 A$$

FSMs - Moore and Mealy - Beispiel Erkennen einer Bitsequenz - Schematics

- Moore

$$S_1' = S_0 A$$
$$S_0' = \overline{A}$$

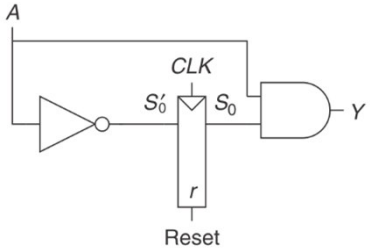
$$Y = S_1$$



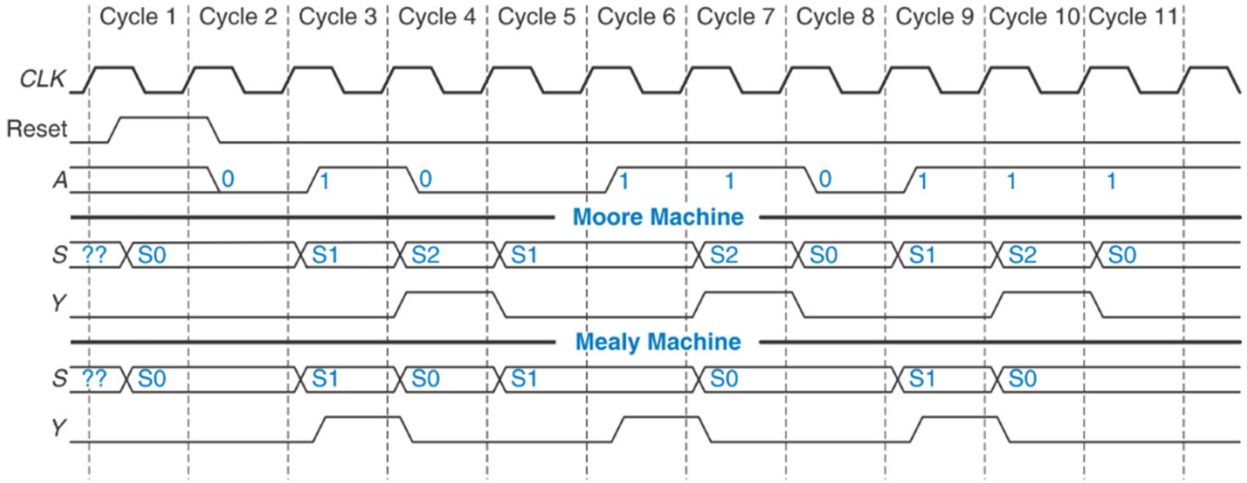
- Mealy

$$S_0' = \overline{A}$$

$$Y = S_0 A$$



FSMs - Moore and Mealy - Beispiel Erkennen einer Bitsequenz - Timing Diagram



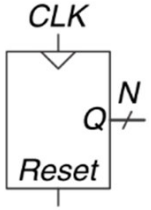
- Die Moore und Mealy FSM haben eine unterschiedliche Abfolge von Zuständen
- Das Ausgangssignal der Mealy FSM ist eher vorhanden, da es direkt auf das Eingangssignal reagiert und nicht wartet, dass der Zustand geändert wird
- Wenn der Mealy-Ausgang durch ein FF geschickt wird, entspricht er dem Moore Ausgang

FSMs - Zusammenfassung

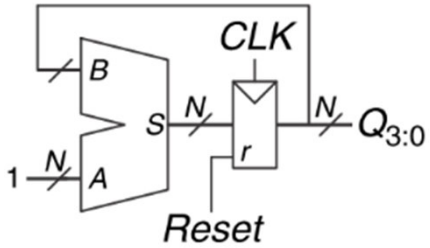
- FSMs sind mächtige Werkzeuge, praktisch alle Digitalelektronik fußt darauf
- Das Vorgehen eine FSM zu entwerfen kann man wie folgt zusammenfassen:
  - Eingänge und Ausgänge identifizieren
  - State chart zeichnen
  - Wenn Moore FSM: state transition und output table
  - Wenn Mealy FSM: Kombinierte state tranistion und output table
- Festlegen der Kodierungen
- Berechnen der bool'schen Gleichungen
- Zeichnen des Schaltplans

Counter

- Ein N-Bit Binärzähler (binary counter) ist eine sequentielle arithmetische Schaltung mit Eingängen für CLK und Reset und einem Ni-bit Ausgang Q



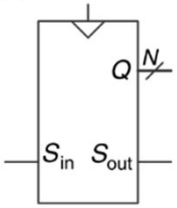
- Reset inilialisiert den Ausgang Q auf 0
- Taktflanke für Taktflanke werden wird der Binärwert inkrementiert bis alle 2^N möglichen Binärwerte durchschritten sind
- Danach wieder bei Null begonnen



- Ein Zähler kann man aus einem Addierer und einem Register mit Reset realisieren
- Bei jeder Taktflanke wird 1 zum dem gespeicherten Wert hinzuaddiert
- Anwendung: Steuerung von Prozessoren (Programmcounter (PC)), Ablaufsteuerungen, diverse Perfomance/Ereigniszähler, Taktteiler

Shift register - Funktion

- Ein Schieberegister(shift register) hat einen Takt, einen seriellen Eingang Sin und einen seriellen Ausgang Sout sowie N parallele Ausgänge Q_n-1:0

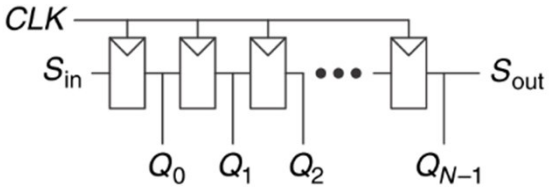


- Bei jeder Taktflanke wird bei Sin ein bit hereingeschoben und alle weiteren Stufen um eins weitergeschoben
- Das letzte Bit/Ausgang des Schieberegisters ist Sout

- Ein Schieberegister kann auch als Seriell-zu-Parallel-Wandler (serial-to-parallel converter) betrachtet werden
- Der Input wird seriell (bit für bit) in Sin eingeführt
- Nach N Taktzyklen sin die alten Eingangsbits parallel am Ausgang Q verfügbar

Shift register - Realisierung

- Ein Schieberegister wird aus der Serienschaltung aus N FFs aufgebaut (vgl. Parallelschaltung von FFs ist eine normales Register)

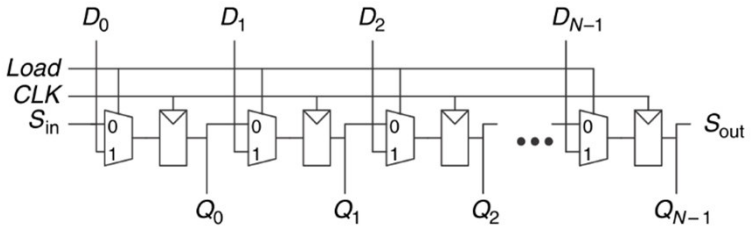


- Einige Schieberegister haben ein Reset-Signal um alle FFs auf 0 zu setzen

- Anwendung:
  - Wie bereits gesagt, ist eine oft verwendete Anwendung von Schieberegistern die Umwandlung von serielle Datenübertragung in parallele Datenübertragung (deserializer)
    - Sowas wird z.B. bei allen seriellen Übertragungsverfahren benötigt (serieller Empfänger) (RS232, I2C, CAN, SATA, SPI, AC97, PCIe …)
  - Multiplikation und Division

Shift register - Serialisierung und Deserialisierung (SERDES) (R)

- Ein mit dem Schieberegister engverbundene Schaltung ist der Parallel-zu-Seriell-Wandler (parallel-to-serial converter) (serializer)
- Eine solche Schaltung läd N Bits parallel und schiebt diese dann Bit für Bit zum Ausgang
- Ein Schieberegister kann modifiziert werden, sowohl Serialisierung als auch Deserialisierung durchführen zu können
- Dafür werden ein paralleler Dateninput D_n-1:0 und ein Steuersignal Load hinzugefügt



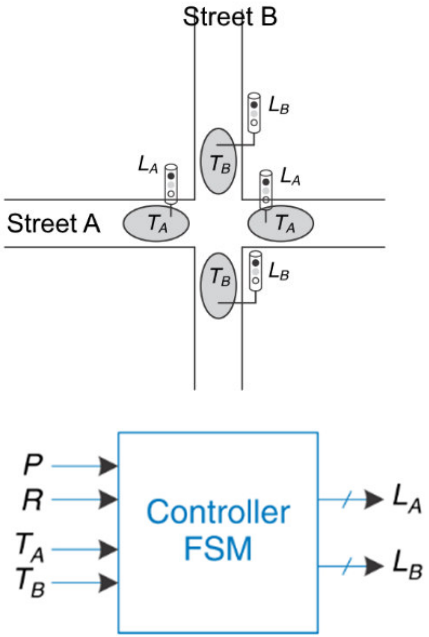
- Wenn Load=1 -> Q_0:n-1=D_0:n-1;
  ansonsten normale Shift-Operation
- Anwendung: Serializer=serieller Sender (z.B. PCIe), Deserializer=serieller Empfänger (RS232, I2C, CAN, SATA, SPI, AC97, PCIe …)

FSMs - Factoring of FSMs (R)

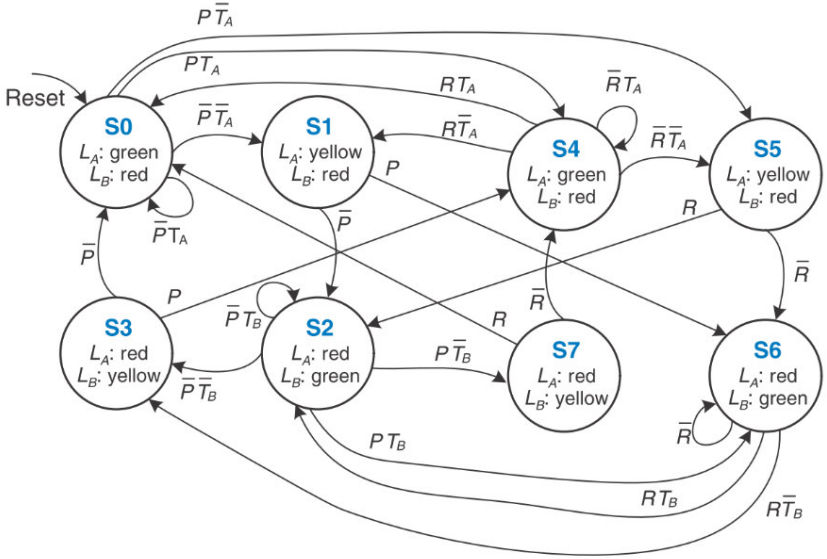
- Wenn man komplexe FSMs entwirft ist es oft einfacher, die FSM in mehrere kleinere miteinander interagierende FSMs „runter zu brechen“
- Diese Anwendung von Hierarchie und Modularität wird „factoring of FSMs“ genannt

FSMs - Factoring of FSMs - Beispiel erweitere Ampelsteuerung - Problemstellung (R)

- Die bereits bekannte Ampelsteuerung soll um einen Umzugsmodus für Paraden und Festzüge erweitert werden
- Lb soll dabei auf grün bleiben und La entsprechend auf rot bleiben
- Der Ampel-Controller bekommt dabei zwei zusätzliche Eingänge
  - P: Wenn zumindest 1 CLK auf 1 -> Starten des Umzugsmodus (parade)
  - R: Wenn zumindest 1 CLK auf 1 -> Zurückkehren (return) zum normalen Betrieb

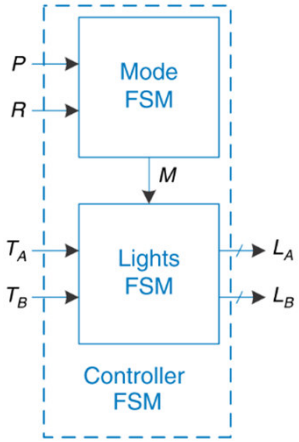


FSMs - Factoring of FSMs - Beispiel erweitere Ampelsteuerung - State chart (R)



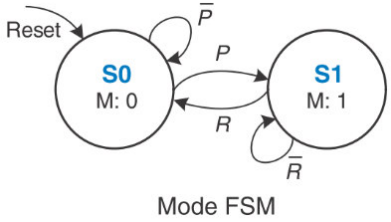
- Die Zustände S0 bis S3 sind der normale Betriebsmodus, S4 bis S7 kümmert sich um die Umzugsmodus
- Die zwei Hälften sind identisch mit Ausnahme, dass die FSM in S6 stehen bleibt
- Die FSM wird bereits unübersichtlich und fehleranfällig

FSMs - Factoring of FSMs - Beispiel erweitere Ampelsteuerung - State chart - factored (R)

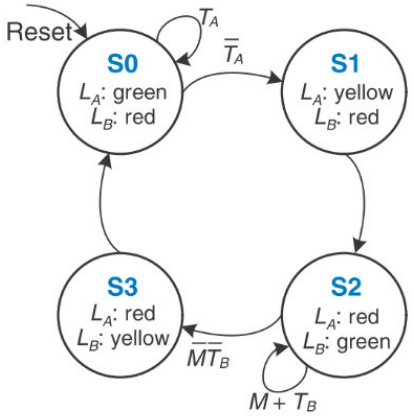


- M: soll aktiv (1) sein, wenn der Umzugsmodus aktiv ist

- Die Mode-FSM hat zwei Zustände für normalen Betrieb und Umzugsbetrieb.
- Die Lights-FSM ist modifiziert, so dass sie in S2 bleibt, solange M=1 ist



Mode FSM



Lights FSM

Backup Übungen Lösungen

FSMs - State encodings - Beispiel Nfach-Frequenzteiler -Vergleich encoding binär vs. one-hot - Übung

- Zeichnen Sie die Schaltung der Binärkodierten FSM und der one-hot kodierten FSM

- Binärkodierung

$$S_1' = \text{So}$$
$$S_0' = \overline{S}_1 \overline{S}_0$$

$$Y = \overline{S}_1 \overline{S}_0$$

- One-hot Kodierung

$$S_2' = S_1$$
$$S_1' = S_0$$
$$S_0' = S_2$$

$$Y = S_0$$

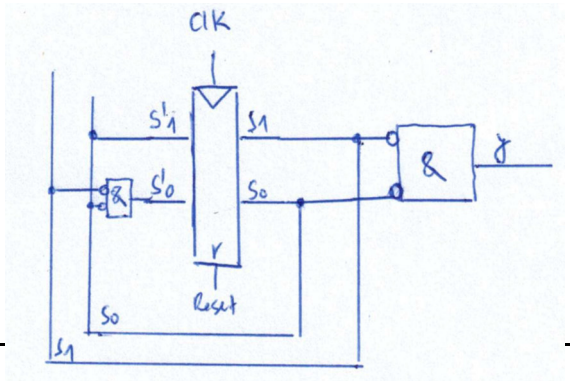
FSMs - State encodings - Beispiel Nfach-Frequenzteiler -Vergleich encoding binär vs. one-hot - Lösung

- Zeichnen Sie die Schaltung der Binärkodierten FSM und der one-hot kodierten FSM

- Binärkodierung

$$S_1' = \text{So}$$
$$S_0' = \overline{S}_1\overline{S}_0$$

$$Y = \overline{S}_1\overline{S}_0$$



- One-hot Kodierung

$$S_2' = S_1$$
$$S_1' = S_0$$
$$S_0' = S_2$$

$$Y = S_0$$