FACHHOCHSCHULE
TECHNIKUM WIEN

Fachbereich: Embedded Systems

Themengebiet: Computerarchitektur

Sequentielle Logiksysteme

Version 2.2, August 2007

Peter Balog

Inhaltsverzeichnis

0. Übersicht

0.1. Lehrziele

Mit dem Studienbrief *Sequentielle Logiksysteme* erfolgt die Einführung in die sequentiellen digitalen Systeme. Sequentielle Systeme besitzen eine funktionale Zeitabhängigkeit und ermöglichen somit die Realisierung von zeitlichen Abläufen. Das Ziel dieser Einheit liegt im Aufbau eines Verständnisses für die Spezifikation von sequentiellen Vorgängen und deren Implementierung mit geeigneten Modellen (Automaten).

0.2. Lehrinhalt

Ausgehend vom implementierungsbedingten zeitlichen bzw. sequentiellen Verhalten von kombinatorischen Logiksystemen werden die Beschreibungsmethoden des Zeitdiagramms und des Zustandsdiagramms eingeführt. Anhand des elementarsten Speicherelementes, dem RS-Flip-Flop, wird das funktionale sequentielle Verhalten illustriert. Im Anschluss daran wird sofort das taktgesteuerte D-Flip-Flop definiert und der finite Automat sowohl allgemein als auch der synchrone Spezialfall eingeführt. Darauf aufbauend wird die taktgesteuerte FSM (*Finite State Machine*) vorgestellt, die es ermöglicht auf einfache konstruktive Art und Weise sequentielle Abläufe zu realisieren.

0.3. Anmerkung

Das wesentlichste Ziel dieser Studieneinheit ist, dass Sie die implementierungsbedingte von einer funktionalen Zeitabhängigkeit unterscheiden können und verstanden haben, warum das vorgestellte Automatenmodell geeignet ist, einen sequentiellen Vorgang zu implementieren.

1. Zeitabhängigkeit in kombinatorischen Systemen

Kombinatorische Logiksysteme besitzen keine funktionale Zeitabhängigkeit. D.h. ein bestimmter Eingangswert führt unabhängig von der Zeit immer zum selben Ausgangswert. Der Zusammenhang von Eingangs- und Ausgangswerten kann mit einer Wahrheitstabelle statisch beschrieben werden.

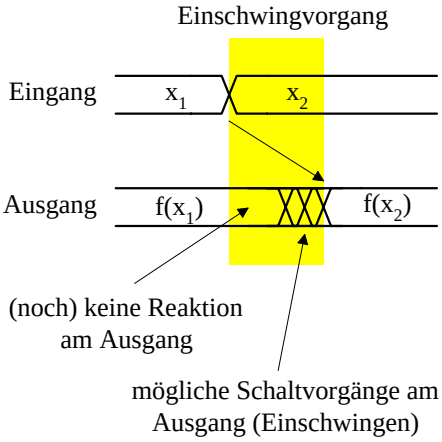
Kombinatorische Logiksysteme werden aus logischen Grundfunktionen (Gatter, *gates*) und Verbindungen (Leitungen, *wires*) realisiert. Sowohl die Gatter als auch die Leitungen besitzen eine Laufzeit (Verzögerungszeit, *delay*), da die Ausbreitungsgeschwindigkeit von elektrischen Signalen begrenzt ist (die Obergrenze ist die Lichtgeschwindigkeit).

Ändert sich der Wert des Eingangs eines kombinatorischen Logiksystems dauert es eine bestimmte Zeit bis der Ausgang den entsprechenden Wert gemäß Wahrheitstabelle annimmt. Die Zeit des sogenannten Einschwingvorganges wird als Durchlauf-Verzögerungszeit (*propagation delay*) bezeichnet. Die Wahrheitstabelle eines kombinatorischen Systems beschreibt somit den eingeschwungenen Zustand (auch als stationärer Zustand bezeichnet) des Systems. Das Verhalten des kombinatorischen Systems während des Einschwingens ist durch die Beschreibung mit der Wahrheitstabelle nicht erfasst. Der Einschwingvorgang ist ein sequentieller Vorgang,- kombinatorische Logiksysteme besitzen eine implementierungs-bedingte Zeitabhängigkeit.

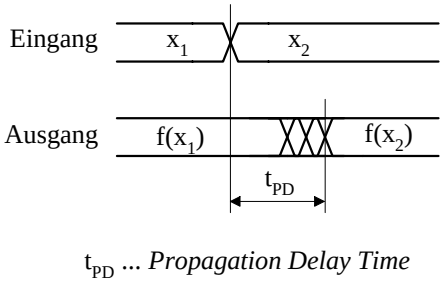
Zur Darstellung dieser dynamischen Eigenschaften eines Systems benötigen wir andere Darstellungsmethoden, welche in den folgenden Kapiteln vorgestellt werden.

1.1. Das Zeitdiagramm

Mit dem Zeitdiagramm können dynamische Vorgänge sowohl qualitativ als auch quantitativ dargestellt werden. Für jeden (relevanten) Ein- und Ausgang wird eine Zeitspur (*time track*) gezeichnet. Die zeitlichen Abhängigkeiten werden entweder durch Pfeile visualisiert (qualitativ) oder sie werden konkret bemaßt (quantitativ).



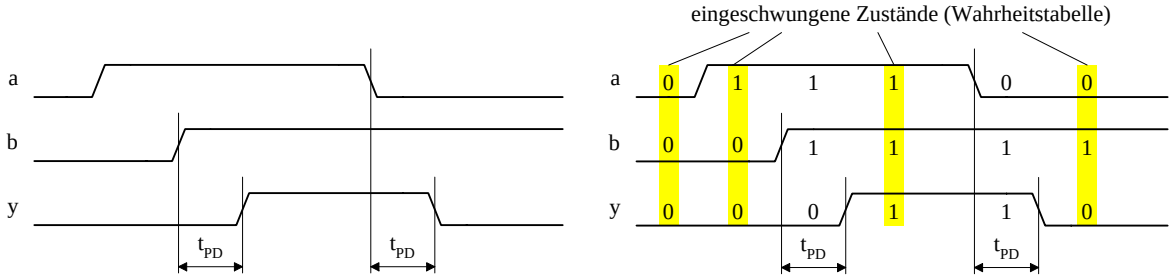
qualitatives Zeitdiagramm



t_{PD} ... *Propagation Delay Time*

quantitatives Zeitdiagramm

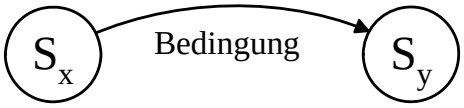
Das folgende Beispiel zeigt das Zeitdiagramm für ein UND2-Gatter:



Die gelb markierten Spalten im rechten Diagramm zeigen die eingeschwungenen Zustände wie sie in der Wahrheitstabelle auftreten. Dazwischen liegen die dynamischen Übergänge.

1.2. Das Zustandsdiagramm

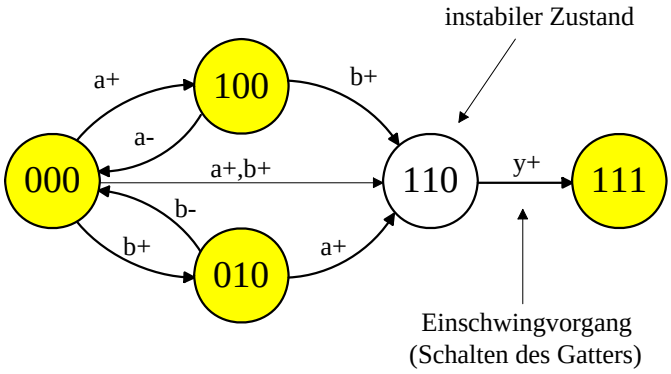
Das Verhalten von sequentiellen Systemen jeder Art wird oft mit Zustandsdiagrammen (*state diagram, state transition diagram, bubble diagram*) beschrieben. Das Zustandsdiagramm beschreibt das sequentielle Verhalten abstrahiert von konkreten zeitlichen Details. Die Grundelemente sind der Zustand (*bubble*) und der Übergang (*transition*) mit der Übergangsbedingung. Ein Zustandsdiagramm ist ein gerichteter Graph.



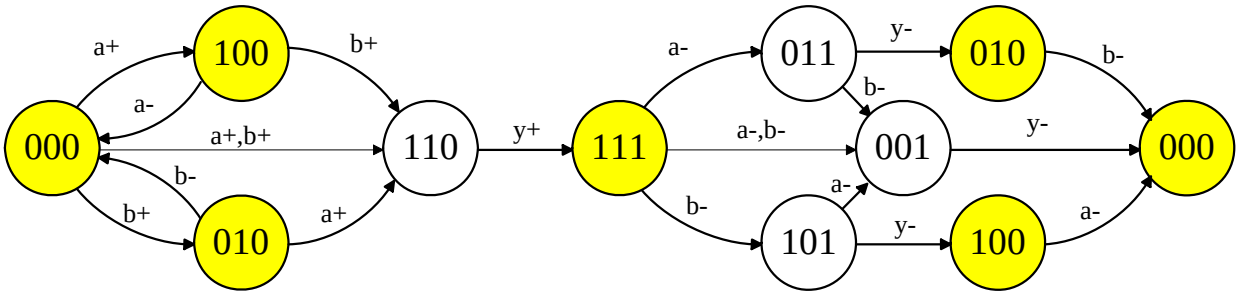
Zur Beschreibung von Einschwingvorgängen in kombinatorischen Systemen bildet man den Zustand aus den Eingangs- und Ausgangssignalen. Die Transitionsbedingungen sind die Änderungen dieser Signale. Dabei bedeutet `signal+` eine Änderung von 0 auf 1 (steigende Flanke) und `signal-` eine Änderung von 1 auf 0 (fallende Flanke).

Am Beispiel eines UND-Gatters mit 2 Eingängen soll nun ein Zustandsdiagramm entwickelt werden. Dabei bilden die Eingänge `a` und `b` sowie der Ausgang `y` den Zustand `{a,b,y}`.

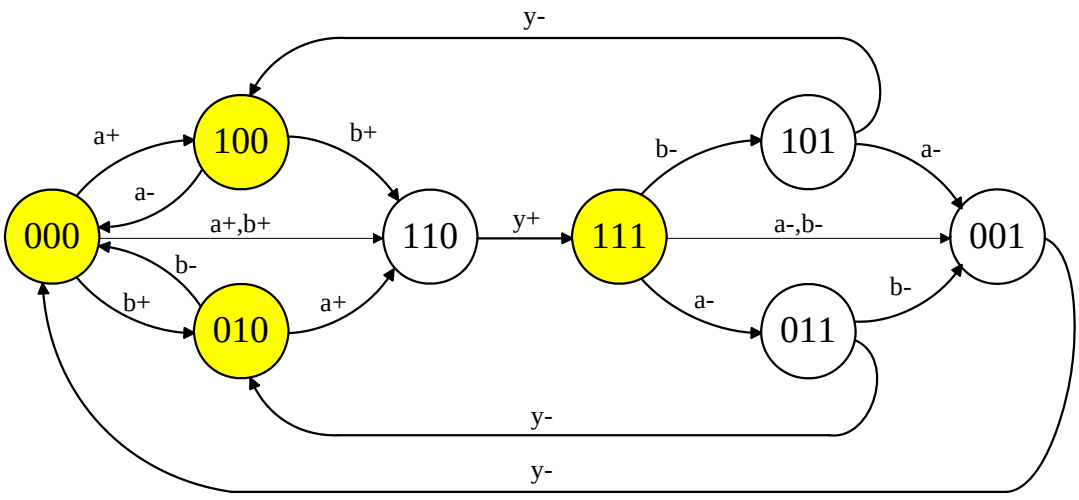
Ausgehend vom stabilen Zustand {a,b,y}={0,0,0} wird der Vorwärtspfad bis zum stabilen Zustand {1,1,1} visualisiert.



Vom {1,1,1} Zustand wird dann „zurück“ in den Anfangszustand geschalten.



Der obige Graph wird derart umgezeichnet, sodass jeder Zustand nur einmal vorkommt.



Die strichliert dargestellten Transitionen zeigen Übergänge die in der Praxis selten vorkommen, da es i.A, nicht möglich ist, dass 2 Eingangssignale „echt“ gleichzeitig ihren Wert ändern.

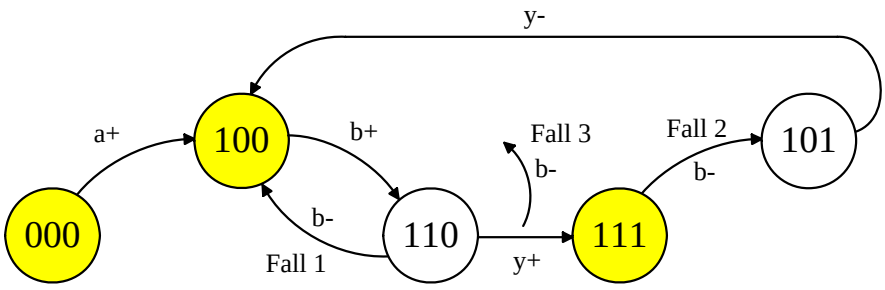
Zusammenfassung:

Die dynamische Betrachtung eines UND-Gatters mit 2 Eingängen führt auf ein System mit 8 möglichen Zuständen. Die 4 stabilen Zustände entsprechen den 4 stationären oder eingeschwungenen Zuständen wie sie ja von der Wahrheitstabelle erfasst sind. Die 4 instabilen Zustände (Übergangszustände) beschreiben die Systemzustände unmittelbar nach einer Eingangsänderung wo der Ausgang noch den „alten“ Wert hat.

1.3. Dynamische Störungen (optional)

Ändert sich ein Eingang zweimal innerhalb eines Zeitintervalls welches kürzer als die Laufzeit des Systems ist, so verhält sich der Ausgang i.A. nicht mehr „digital“. Dieser Effekt gehört in die Kategorie der „analogen Aspekte von digitalen Systemen“, mit denen sich (Mikro-) Elektroniker herumschlagen müssen.

Das folgende Beispiel visualisiert diese Problematik anhand des bekannten UND-Gatters mit 2 Eingängen.



Ausgangspunkt ist der Zustand {a,b,y}={0,0,0}. Mit der Eingangsänderungssequenz a+ und b+ gelangt man in den instabilen Zustand {1,1,0}. Danach soll b wieder ausgeschaltet (b-) werden. Abhängig vom Zeitpunkt von b- ergeben sich 3 Fälle:

- Fall 1: Das Ereignis b- erfolgt so schnell nach b+, dass das UND-Gatter den Impuls (b+,b-) nicht registriert.
- Fall 2: Das Ereignis b- erfolgt erst, nachdem das UND-Gatter den Ausgang geschaltet (y+) hat.
- Fall 3: Das Ereignis b- erfolgt in dem Zeitraum, indem das UND-Gatter den Ausgangswert von 0 auf 1 ändert; also zu einem Zeitpunkt wo der Ausgang nicht mehr 0 aber noch nicht 1 ist. Aus diesem illegalen (digitalen) Zustand kehrt der Ausgang wieder zum Wert 0 zurück.

Der Fall 3 führt i.A. auf eine dynamische Störung am Ausgang, wo ein (zu) kurzer Impuls mit nicht ausreichender Amplitude (der Zustand 1 wird ja nicht erreicht) entsteht.

Aufgabe 1:

Versuchen Sie diese 3 Fälle in einem Zeitdiagramm zu visualisieren.
(Lösung siehe Seite 34)

1.4. Zusammenfassung der Zeitabhängigkeit von kombinatorischen Systemen

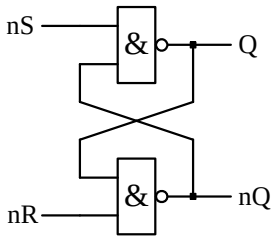
- Kombinatorische Logiksysteme haben keine funktionale Zeitabhängigkeit.
- Kombinatorische Logiksysteme haben eine implementierungsbedingte Zeitabhängigkeit.
- Die Wahrheitstabelle beschreibt den funktionalen Zusammenhang zwischen Ein- und Ausgängen. Dieser Zusammenhang entspricht dem eingeschwungenen Zustand.
- Während des Einschwingvorganges nach Eingangsänderungen verhält sich das System zeitabhängig und somit sequentiell.
- Das Verhalten des Einschwingvorganges wird durch die Wahrheitstabelle nicht erfasst.
- Der Einschwingvorgang, bzw. ganz allgemein ein sequentieller Vorgang, kann mit einem Zeitdiagramm oder einem Zustandsdiagramm beschrieben werden.

2. Sequentielle Systeme

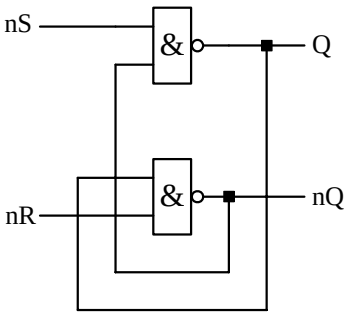
Sequentielle digitale Systeme haben eine funktionale Zeitabhängigkeit. Der Ausgangswert ist nicht nur vom momentanen Eingangswert sondern auch vom Zeitpunkt bzw. von der zeitlichen Abfolge (Sequenz) von Eingangswerten abhängig. Anders formuliert bedeutet dies, dass der Ausgangswert vom momentanen Eingangswert und von der Vorgeschichte der Eingangswerte abhängig ist. Ein sequentielles System verfügt über ein „Gedächtnis“. In einem sequentiellen System existieren somit speichernde Elemente, in denen die „Vorgeschichte“ gespeichert wird.

2.1. Elementarer Speicher

Die elementaren Speicherelemente sind die sogenannten Flip-Flops. Sie stellen die sequentiellen Grundfunktionen dar. Das funktionale sequentielle Verhalten wird am elementarsten aller Flip-Flops, dem asynchronen RS-Flip-Flop gezeigt. Im Prinzip basieren alle Flip-Flops auf dem RS-Flip-Flop.



klassischer Zeichenstil

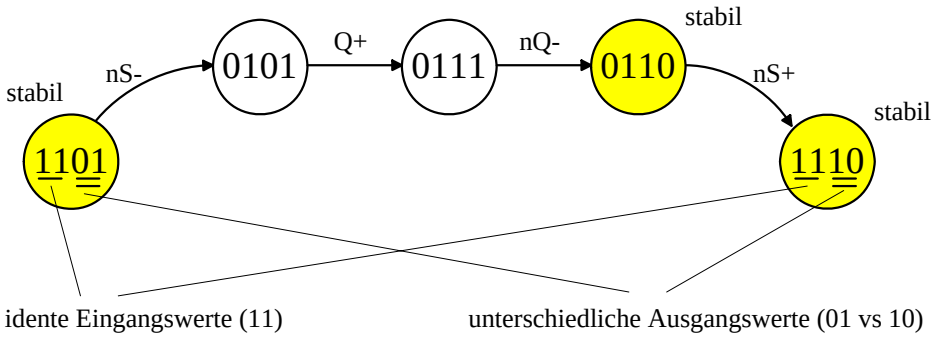


Rückkopplungen besser sichtbar

Der nS-Eingang ist der **Set**-Eingang; eine 0 an diesem Eingang setzt den Ausgang des Flip-Flops (Q=1 und nQ=0). Der nR-Eingang ist der **Reset**-Eingang; eine 0 an diesem Eingang setzt den Ausgang zurück (Q=0 und nQ=1). Q ist der Ausgang des Flip-Flops und nQ ist der invertierte Ausgang.

Sind beide Eingänge nS und nR gleich 1, so ist das Flip-Flop in der Speicher-Betriebsart. Ist nS=0 und nR=1 dann handelt es sich um die Set-Betriebsart und bei nS=1 und nR=0 um die Reset-Betriebsart. Die Eingangskombination nS=nR=0 ist nicht zulässig.

Der Zustand wird gebildet durch {nS,nR,Q,nQ}. Wie bei allen sequentiellen Systemen muss zuerst ein stabiler Anfangszustand (stationärer Zustand) gesucht werden. Ein möglicher Anfangszustand ist {nS,nR,Q,nQ} = {1,1,0,1}.



Das obige Zustandsdiagramm zeigt ausgehend vom Speicherzustand einen Setzvorgang (nS-) gefolgt von einem neuerlichen Speicherzustand (nS+). Man erkennt, dass trotz identem Eingangswert (nS=nR=1) der Ausgang unterschiedlich ist. Das Flip-Flop hat den Zustand geändert. Das Flip-Flop besitzt zwei stabile Speicherzustände!

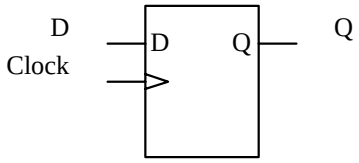
Anmerkung:

> In der „klassischen“ Literatur zum Thema Digitaltechnik wird nun das RS-Flip-Flop weiter „seziert“ und darauf aufbauend sämtliche asynchronen, taktzustands- und taktflankengesteuerten Flip-Flops entwickelt. Im Rahmen dieser Lehrveranstaltung wurde das RS-Flip-Flop lediglich dazu verwendet um das Prinzip eines funktional zeitabhängigen Systems zu zeigen.
> Wir wenden uns gleich dem taktflankengesteuerten D-Flip-Flop zu und definieren das Verhalten, da eigentlich nur das D-FF in den praktischen Anwendung von synchronen Automaten eine Bedeutung hat.

2.2. Das D-Flip-Flop (DFF)

Das (taktgesteuerte) D-Flip-Flop ist das Grundelemente der synchronen digitalen Schaltungstechnik.

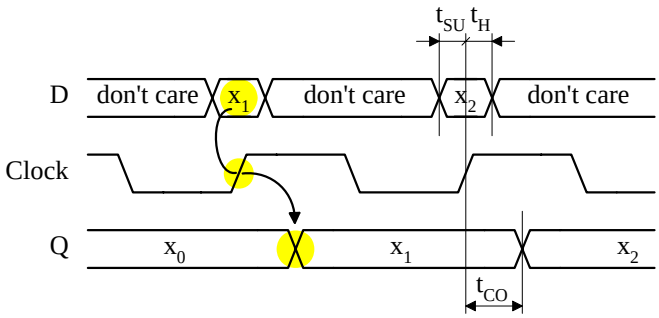
- Schaltsymbol:



- Funktion:

  Der Ausgang Q übernimmt mit der steigenden Flanke des Taktes (*clock*, *clk*) den Wert des D-Einganges. Das DFF hält bei allen anderen Werten des Taktes (0, 1, fallende Flanke) den Ausgang Q konstant (Speicherbetrieb).

- Zeitdiagramm:



  t_{CO} ist die Schaltzeit (*Clock-to-Output Time*) des Flip-Flops. In einem schmalen Zeitfenster um die Taktflanke (t_{SU} ... *Setup Time*, t_H ... *Hold Time*) muss der Dateneingang D stabil sein, damit das DFF korrekt arbeiten kann. Für funktionale Überlegungen kann dieses Zeitfenster vernachlässigt (d.h. 0 gesetzt) werden.

Für synchrone (taktgesteurte) Elemente und Systeme wird häufig wieder die Wahrheitstabelle zur Beschreibung verwendet. Um die (diskrete) Zeitabhängigkeit (Sequenz) zu modellieren werden 2 Spalten für den Ausgangswert eingeführt; eine Spalte beschreibt den momentanen Zustand (***present state***) und die andere beschreibt den Folgezustand (***next state***). Zwischen *present state* und *next state* liegt der eigentliche Schaltvorgang (die Taktflanke).

- Analyse Wahrheitstabelle:

D	$Q_{present}$	Q_{next}
0	x	0
1	x	1

Die Analyse-Wahrheitstabelle beschreibt die Funktion des D-Flip-Flops sehr anschaulich. Unabhängig vom momentanen Zustand folgt der Ausgang Q dem Eingang D nach der aktiven (i.A. steigenden) Taktflanke.

- Synthese Wahrheitstabelle:

$Q_{present}$	0	0	1	1
Q_{next}	0	1	0	1
D	0	1	0	1

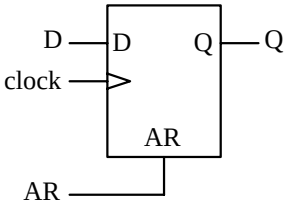
Die Synthese-Wahrheitstabelle geht vom *present state* aus und liefert Information darüber was am Eingang angelegt werden muss damit man nach der Taktflanke den *next state* erhält.

2.3. Spezielle D-Flip-Flops

Basierend auf der prinzipiellen Funktionalität des D-Flip-Flops gibt es einige Abwandlungen (Derivate) die für praktische Implementierungen von Bedeutung sind:

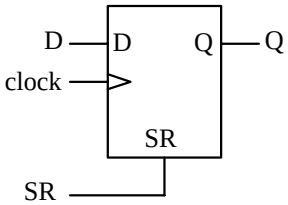
- DFF mit asynchronem Reset

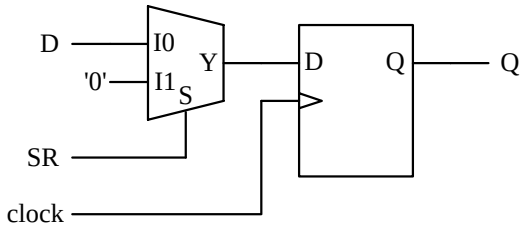
  Das DFF mit einem asynchronen Reset ist ein Grundelement (*primitive*) wie das „normale“ DFF; d.h. es kann nicht aus anderen Grundelementen zusammengesetzt werden.



  Funktion: Wenn AR aktiv ist, wird der Ausgang und damit das Speicherbit auf Null gesetzt. Dies passiert unabhängig vom Takt. Der AR-Eingang hat höherere Priorität als der D und der Takteingang, d.h. die asynchrone Operation hat Vorrang vor der synchronen Operation

- DFF mit synchronem Reset

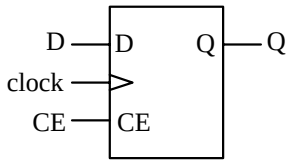


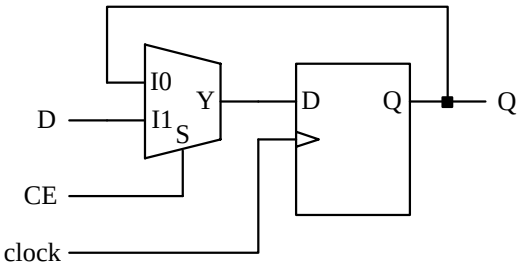


  Das DFF mit synchronem Reset kann aus einem „normalen“ DFF und einem 2-to-1 Multiplexer realisiert werden. Es ist auch als *Primitive* verfügbar.

  Funktion: Wenn SR aktiv ist (1 in obiger Abb.), dann wird mit der nächsten Taktflanke das DFF rückgesetzt. Solange SR aktiv ist, ist der D-Eingang außer Betrieb.

- DFF mit Clock-Enable (CEDFF)





Das CEDFF kann aus einem „normalen“ DFF und einem 2-to-1 Multiplexer realisiert werden. Es ist auch als *Primitive* verfügbar.

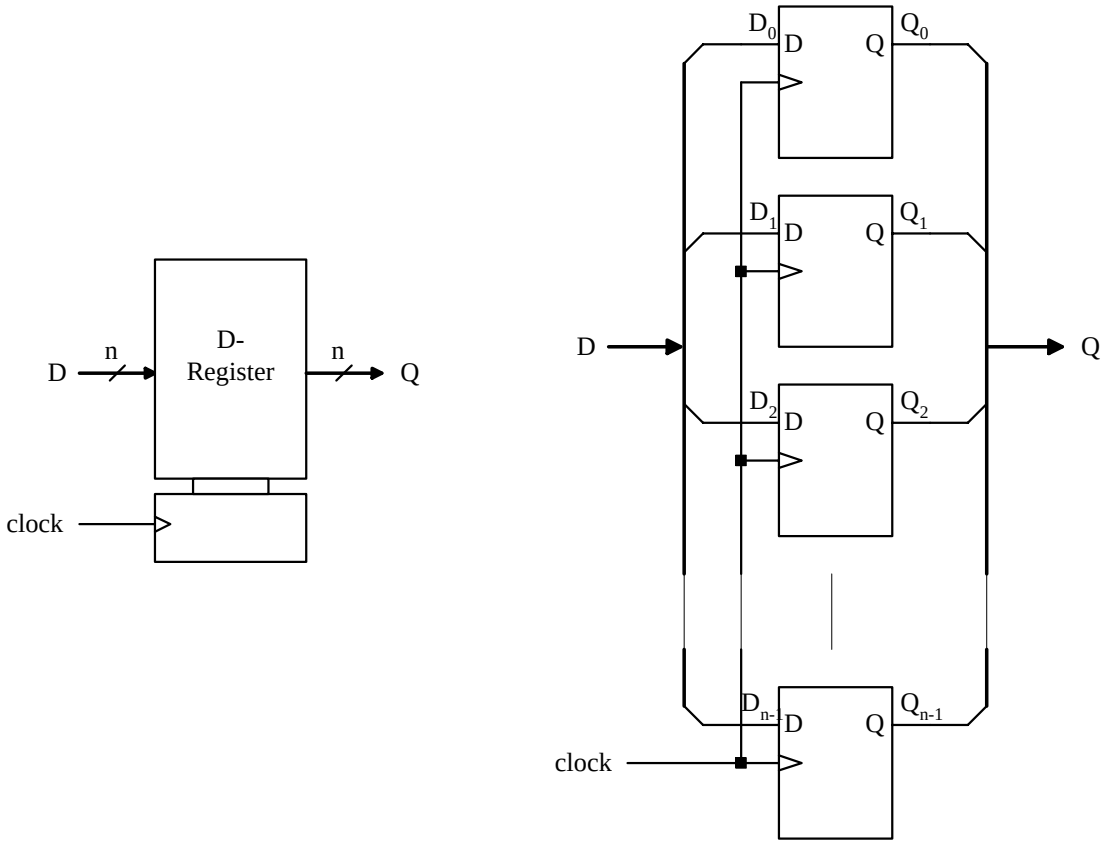
Funktion: Nur bei aktivem Clock-Enable (1 in obiger Abb.) wird der D-Eingang auf das interne DFF durchgeschaltet. Ist CE inaktiv wird der aktuelle Zustand gehalten.

Neben dem asynchronen Reset gibt es auch noch zusätzlich asynchrone Set-Funktionalität. Man kombiniert dabei ein synchrones DFF mit einem asynchronen RS-FF, wobei das asynchrone FF Vorrang hat. Wiederum handelt es sich um eine *Primitive*, d.h. die asynchrone Funktionalität kann nicht nachträglich eingebaut werden.

Weiters sind die oben angeführten Zusatzfunktionen beliebig kombinierbar. Z.B. kann einem DFF mit AR eine zusätzliche CE-Funktionalität gegeben werden.

2.4. Das D-Register

Das D-Flip-Flop ist der elementare (synchrone) 1-Bit Speicher. Müssen in Abhängigkeit vom selben Takt mehrere Bits gespeichert werden, verwendet man das D-Register. Dabei werden die entsprechende Anzahl von DFFs bzgl. des Taktes parallel geschaltet.



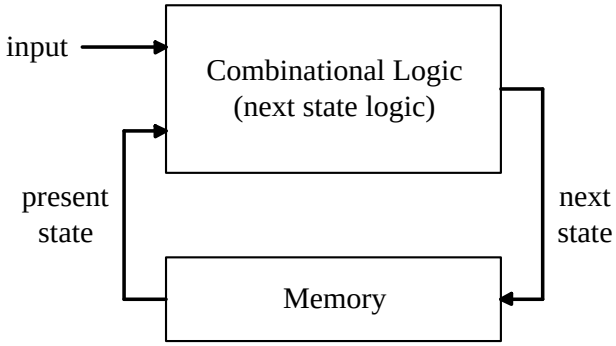
Mit der (steigenden) Taktflanke werden alle D-Eingänge auf die entsprechenden Q-Ausgänge übernommen. Ein n-Bit breites D-Register enthält n Stück DFFs und implementiert somit einen (synchronen) n-Bit großen Speicher.

3. Der finite Automat

3.1. Allgemeines zum finiten Automaten

Der finite Automat (FA) ist ein Modell welches die Implementierung von sequentiellen Abläufen ermöglicht. Der FA besteht aus:

- Zustandsspeicher (*memory*)
- Rückkoppellogik (*feedback logic*, *next state logic*)

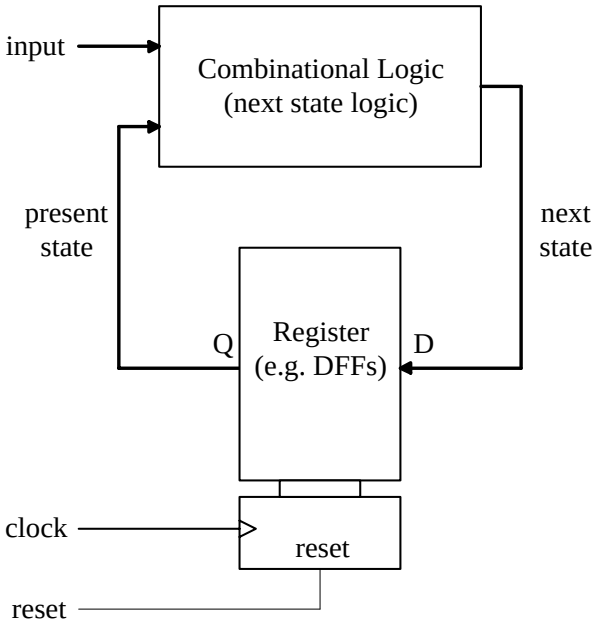


Der sequentielle Vorgang wird als (sequentieller) Ablauf von einer endlichen Anzahl von Zuständen beschrieben:

- aktueller oder momentaner Zustand (present state)
- Folgezustand (next state) ist eine Funktion von
  - den Eingängen (input)
  - und dem momentanten Zustand
- Zeitpunkt des Zustandswechsel (state change)
  - synchron, d.h. zu diskreten Zeitpunkten (z.B. Taktflanken)
  - asynchron, d.h. jederzeit

3.2. Der synchrone finite Automat

Der synchrone oder getaktete (*clocked*) FA verwendet ein taktflankengesteuertes Register als Zustandsspeicher (Zustandsregister, *state register*). Das Zustandsregister verfügt über eine optionale Rücksetzfunktion (*reset*), die entweder synchron oder asynchron sein kann. Diese Reset-Funktion hat i.A. keinen Einfluss auf die Funktion des FAs.



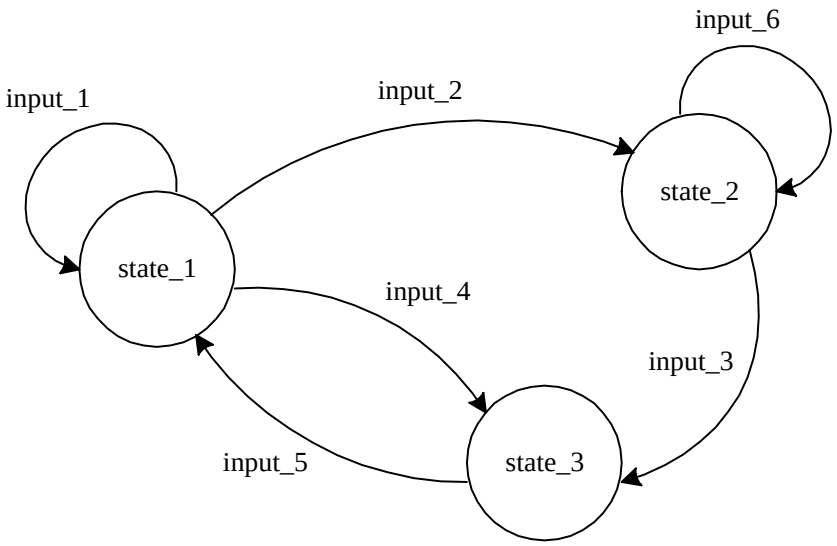
Die Funktion des synchronen FAs (ohne Reset) wird durch die folgenden Gleichungen beschrieben. Dabei bedeutet $:=$ eine taktabhängige Zuweisung, d.h. die Zuweisung erfolgt ausschließlich bei der aktiven (steigenden) Taktflanke.

$$next\_state = f(present\_state, input)$$

$$present\_state := next\_state$$

3.3. Spezifikation des FAs mit einem STD

Unabhängig davon ob der FA synchron oder asynchron arbeitet kann die Funktion mit einem Zustandsübergangsdiagramm (***state transisition diagram - STD***, *bubble diagram*) beschrieben werden.



Die Plätze (*places*, *bubbles*) bezeichnen die Systemzustände und die gerichteten Kanten (*arcs*, *transitions*) beschreiben das Übergangsverhalten.

Wenn z.B. der momentane Zustand des FA der **state_2** ist und die Eingangsbedinung **input_3** liegt an, dann wechselt der FA seinen Zustand nach **state_3**.

Die STD-Spezifikation enthält keinerlei Information **WANN** der Zustandswechsel erfolgt. Dieses Verhalten hängt vom „darunter liegenden“ Modell ab (synchron, asynchron)

Ein STD kann leicht auf **Konsistenz** (prinzipielle Korrektheit) überprüft werden:

- Ein STD ist dann konsistent, wenn es vollständig und eindeutig ist.
- Eindeutig bedeutet, dass (a) verschiedene Zustände verschieden benannt sind und (b) in jedem Zustand die relevanten Eingangsbedingungen eindeutig sind.
- Vollständig bedeutet, dass für jeden Zustand alle relevanten Eingangs-bedingungen spezifiziert sind.

Die funktionale Korrektheit wird durch die Konsistenzprüfung natürlich nicht überprüft!

Aufgabe 2:

Überprüfen Sie die Konsistenz der STD-Spezifikation der vorigen Seite. Überlegen Sie den, für eine Konsistenz notwendigen (logischen) Zusammenhang der Eingangs-bedingungen. (Lösung siehe Seite 34)

3.4. Zusammenfassung des (synchronen) FAs

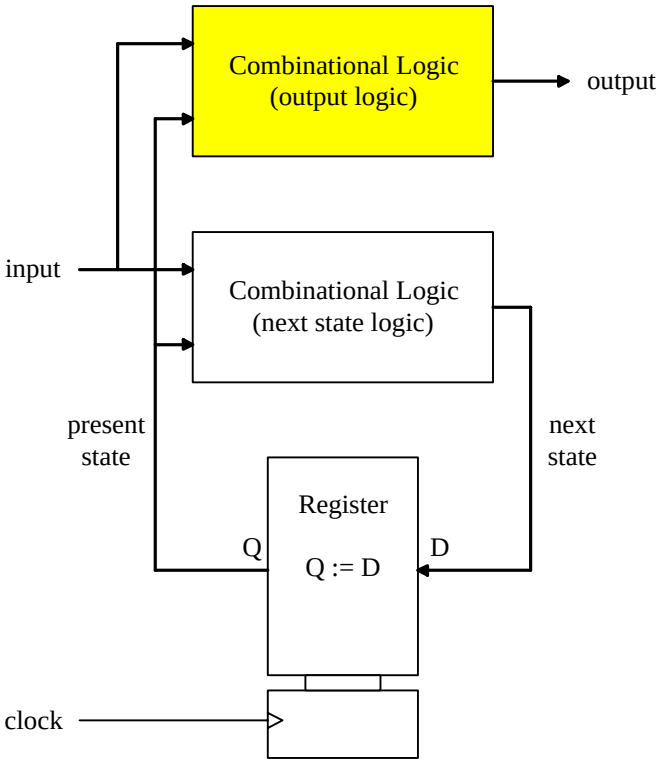
- Die funktionale (diskrete) Zeitabhängigkeit eines zu realisierenden sequentiellen Ablaufes kann mit dem Modell des (synchronen) FAs implementiert werden.
- Die (disrekte) Zeitsteuerung übernimmt der Takt.
- Der aktuelle Zustand wird im Zustandsregister gespeichert.
- Der eigentliche funktionale Ablauf, d.h. der Wechsel der Zustände infolge von Eingangsbedingungen, liegt in einem kombinatorischen Logikblock.
- Die Implementierung eines (synchronen) Automaten „reduziert“ sich somit auf die Synthese einer kombinatorischen Logikfunktion, welche aus der Spezifikation (STD) und dem Hardware-Modell des Automaten gewonnen werden kann !!!

4. Die *Finite State Machine*

Die synchrone oder getaktete Zustandsmaschine (*clocked Finite State Machine* - c´FSM) basiert auf dem synchronen FA. Da heutige sequentielle Systeme durchwegs dem synchronen Design-Paradigma folgen, wird das Attribut *synchron* sehr oft weggelassen und nur von der FSM gesprochen.

4.1. Allgemeines zur FSM

In einer kombinatorischen Logik werden aus den Eingängen und dem momentanen Zustand die Ausgänge berechnet.



Für die Zustandsübergänge gilt wie beim FA:

$$next\_state = f(present\_state, input)$$

$$present\_state := next\_state$$

Ein konkreter Ausgang ist entweder nur vom momentanen Zustand abhängig oder er ist zusätzlich auch noch (direkt) eine Funktion der Eingänge. Es werden deshalb zwei Ausgangstypen definiert:

- Moore-Ausgang:

$$output = f(present\_state)$$

- Mealy-Ausgang:

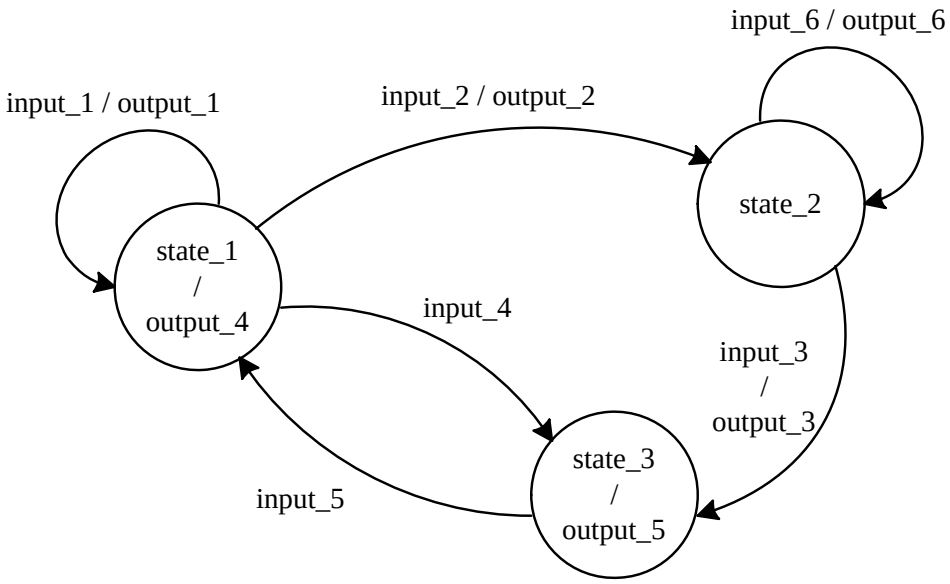
$$output = f(present\_state, input)$$

Hat eine FSM ausschließlich Mealy-Ausgänge wird sie als Mealy-FSM bezeichnet. Äquivalentes gilt für die Moore-FSM.

4.2. STD-Spezifikation des FSM

Die STD-Spezifikation des FAs muss um die Ausgänge erweitert werden. Moore-Ausgänge werden in oder bei den Zustands-*Bubbles* notiert. Mealy-Ausgänge werden bei den Übergängen unmittelbar hinter den Eingangs-bedingungen notiert. Wichtig: ein Ausgang ist entweder vom Moore- oder vom Mealy-Typ! Ein bestimmter Ausgang kann seinen Typ (sein Verhalten) nicht verändern.

Die nächste Abbildung zeigt ein STD mit Moore-Ausgängen (output_4, output_5) und Mealy-Ausgängen (output_1, output_2, output_3, output_6):

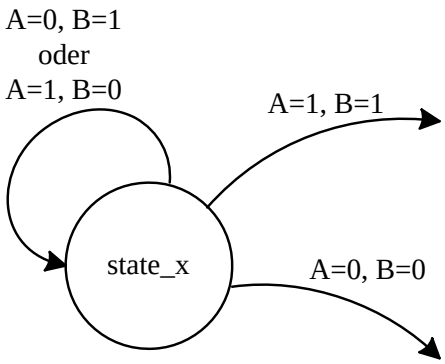


Bezüglich STD-Konsistenz gilt dasselbe wie für die FA-Spezifikation. Zusätzlich muss geprüft werden, ob jeder Ausgang, entsprechend seinem Typ, in allen Fällen einen Wert zugewiesen bekommt.

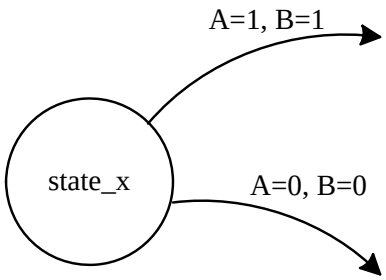
4.3. Vereinfachte Darstellungen bei STDs

Wird eine STD-Spezifikation vollständig gezeichnet, so ist sie i.A. sehr überladen und die eigentliche Funktion wird oft nicht deutlich sichtbar. Deshalb werden oft die folgenden Vereinfachungen getroffen:

- Bei den Zuständen werden nur jene abgehenden Transitionen eingezeichnet die für die Funktion notwendig sind. Alle nicht erfassten Eingangsbedingungen führen automatisch in den eigenen Zustand (Haltebedingung) zurück.
- Problematisch wird die obige Vereinfachung nur dann, wenn Mealy-Ausgänge vorliegen. Welchen Wert haben die Mealy-Ausgänge wenn die Haltebedingung aktiv ist?



Vollständige Darstellung

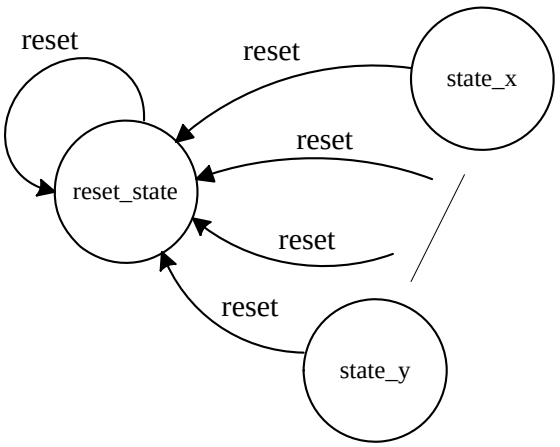


Vereinfachte Darstellung

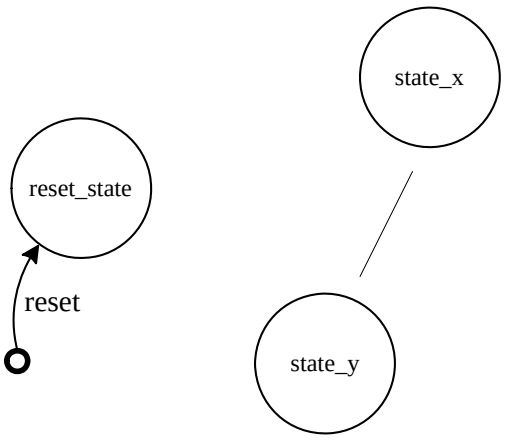
Für Ausgänge, speziell für Moore-Ausgänge gibt es 2 mögliche Schreibvereinfachungen, die jedoch nur wechselweise angewendet werden dürfen:

- Ein Ausgang wird nur dann geschrieben, wenn er seinen Wert ändern soll. Wird ein Ausgang nicht geschrieben, so behält er seinen Wert vom vorigen Zustand.
- Ein Ausgang wird nur dann geschrieben, wenn er (z.B.) den Wert 1 hat. Immer dann wenn der Ausgangswert 0 sein soll wird er nicht geschrieben.

Sogenannte „globale“ Übergänge, das sind Übergänge die unter einer bestimmten Eingangsbedingung aus jedem Zustand in einen speziellen Zielzustand übergehen, versucht man nicht einzutragen, sondern getrennt zu notieren.



Vollständige Darstellung



Vereinfachte Darstellung

4.4. Entwurf und Implementierung einer FSM

Wie schon beim finiten Automat erwähnt (siehe 3.4) reduziert sich die Synthese einer (synchronen) FSM auf die Synthese von kombinatorischen Logikfunktionen (*next state logic, output logic*), welche aus der STD-Spezifikation gewonnen werden können. Das (diskrete) Zeitverhalten ist durch das Hardware-Modell gegeben. Die folgenden Punkte fassen den Entwurf und die Implementierung einer FSM zusammen:

- Spezifikation des sequentiellen Ablaufs mit einem STD. Konsistenzprüfung des STDs.
- Aus der Aufgabenstellung und dem STD liegen nun die Eingänge, die Ausgänge, die Ausgangstypen (Mealy, Moore) und die Zustände vor.
- Abbilden der Zustände auf das Zustandsregister. Diesen Vorgang nennt man **Zustands-kodierung** (*state encoding*), d.h. jedem i.A. symbolisch vorliegenden Zustandsnamen wird ein (eindeutiges) Bitmuster des Zustandwortes zugeordnet. Die Bitbreite des Zustandswortes ergibt sich aus der Anzahl der Zustände und der verwendeten Art der Zustandskodierung. Damit wird auch die Breite des Zustandsregisters festgelegt.
- Zeichnen des Blockschaltbildes der FSM mit allen Eingängen, Ausgängen und Zustandsworte für den *present* und den *next state*. Damit liegen die Eingänge und Ausgänge der beiden kombinatorischen Logikblöcke fest.
- Aus den Übergängen des STD inklusive der Ausgangswerte können die Wahrheits-tabellen der beiden Logikblöcke aufgestellt werden.
- Synthese der kombinatorischen Logikfunktionen.

5. Lehrzielorientierte Fragen

1. Was beschreibt die Wahrheitstabelle eines kombinatorischen Systems?
2. Wodurch kommt es zur implementierungsbedingten Zeitabhängigkeit von kombinatorischen Systemen?
3. Wie schnell breiten sich elektrische Signale abhängig vom Medium aus?
4. Welche Möglichkeiten kennen Sie um das dynamische Verhalten in kombinatorischen Logiksystemen zu beschreiben?
5. Was ist der Unterschied zwischen einem quantitativen und einem qualitativen Zeitdiagramm?
6. Über welche Elemente verfügt das Zustandsdiagramm, wenn dynamische Vorgänge in kombinatorischen Systemen beschrieben werden sollen?
7. Welchen Aspekt erfasst das Zustandsdiagramm nicht?
8. Welche Eigenschaft weist ein sequentielles digitales System auf?
9. Hat die Wahrheitstabelle für sequentielle Systeme eine Bedeutung?
10. Was bedeutet das Attribut „synchron“?
11. Was ist das Grundelement der synchronen sequentiellen Systeme?
12. Was ist ein finiter Automat (FA)?
13. Wie kann die Funktion eines finiten Automaten beschrieben werden?
14. Worin liegt das konkrete zeitliche Verhalten eines Automaten bzw. einer Automaten-implementierung?
15. Was unterscheidet eine FSM vom FA?

16. Welche Ausgangstypen sind bei FSMs möglich?
17. Wann kann sich ein Mealy-Ausgang ändern?
18. Wann kann sich ein Moore-Ausgang ändern?
19. Ein typischer Prozessor eines Computers erlaubt die Abbildung eines sequentiellen Vorganges. Eine höhere Programmiersprache wie C erlaubt die Beschreibung eines sequentiellen Vorganges. Versuchen Sie aus einer allgemeinen STD-Spezifikation eine C-Programmsequenz abzuleiten.
20. Beschreiben Sie die Funktion des DFFs mit einem STD.
21. Beschreiben Sie die Funktion eines DFFs mit synchronem Reset mit einem STD.
22. Beschreiben Sie die Funktion eines CEDFFs mit einem STD.
23. Beschreiben Sie die Funktion eines CEDFFs mit zusätzlichem synchronem Reset mit einem STD.
24. Beschreiben Sie die Funktion eines 3-Bit UpCounters (0,1,2,3,4,5,6,7,0,1,2,usw) mit einem STD.
25. Versuchen Sie einen 3-Bit UpCounter mit bereits bekannten Komponenten zu realisieren?
26. Beschreiben Sie die Funktion eines 3-Bit UpDownCounters mit einem STD. Welcher zusätzliche Eingang wird benötigt?
27. Versuchen Sie einen 3-Bit UpDownCounter mit bereits bekannten Komponenten zu realisieren?

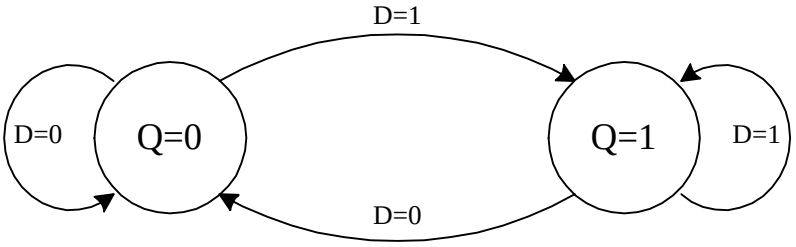
5.1. Antworten auf die lehrzielorientierten Fragen

1. Den stationären oder eingeschwungenen Zustand zwischen Eingängen und Ausgängen.
2. Durch die endliche Signallaufzeit der Logikelemente und Verbindungen.
3. Mit der jeweiligen Ausbreitungsgeschwindigkeit des Mediums die auf jeden Fall kleiner als die Vakuum-Lichtgeschwindigkeit ($c_0 \approx 3*10^8$ m/s) ist. Z.B. in einem Chip oder auf einer Printplatte ca. 50%, in einer Koaxialleitung ca. 60-70% der Lichtgeschwindigkeit.
4. Zeit- oder Zustanddiagramme.
5. Ein qualitatives Zeitdiagramm beschreibt zwar die zeitlichen Abläufe (was kommt früher, was kommt später, was hängt wovon ab) jedoch ohne die Angabe von konkreten Zeiten. Das quantitative Zeitdiagramm erweitert das qualitative um eine zeitliche Bemaßung.
6. Zustände (Bubbles) werden aus den Eingängen und Ausgängen gebildet. Übergänge werden mit Ereignissen (events, Signaländerungen) beschriftet.
7. Die konkrete Zeit.
8. Idente Eingangssituationen können zu unterschiedlichen Zeitpunkten zu unterschiedlichen Ausgangssituationen führen. Anders formuliert: der aktuelle Ausgangswert ist nicht nur vom momentanen Eingangswert sondern auch von der zeitlichen Abfolge (Sequenz) der Eingangswerte abhängig.
9. Die klassische WT (Eingänge, Ausgänge) nicht, da die zeitliche Komponente nicht formuliert werden kann. Ein Modifikation (Eingänge, momentaner Ausgang, nächster Ausgang) kann für finite Automaten oder FSMs sinnvoll sein.
10. Ein synchrones System schaltet nur zu diskreten Zeitpunkten. Diese Zeitpunkte werden aus den Übergängen (Flanken) eines speziellen Signals (Takt, clock) abgeleitet. Meistens wird die steigende Taktflanke verwendet.
11. Das (synchrone) D-Flip-Flop.
12. Ein finiter Automat hat eine endliche Anzahl unterscheidbarer Zustände. Der momentane Zustand ist in einem Zustandsspeicher gespeichert. Der Folgezustand wird aus dem momentanen Zustand und den Eingängen berechnet. Wann (also der konkrete Zeitpunkt) der Zustandswechsel erfolgt, hängt von dem „darunterliegenden“ Hardwaremodell ab.

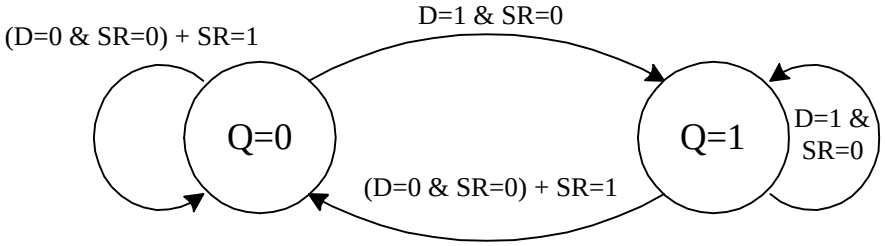
13. Die Funktion wird am Besten mit einem STD (state transition diagram) beschrieben.
14. Vom „darunterliegenden“ Hardwaremodell. Zwei Modelle sind üblich: beim synchronen Automaten erfolgt der Wechsel zum diskreten Zeitpunkt der Taktflanke, beim asynchronen jederzeit.
15. Die FSM basiert auf einem (oder anders formuliert „enthält einen“) FA. Zusätzlich gibt es eine Ausgangsfunktion, die ganz allgemein die Eingängen und den momentanen Zustand verknüpft.
16. Mealy (=f(Eingänge, momentaner Zustand))und Moore (=f(momentaner Zustand)).
17. Mealy-Ausgänge hängen von den Eingängen und dem momentanen Zustand ab. Sie können sich daher in Folge einer Eingangsänderung und in Folge eines Zustands-wechsels ändern.
18. Moore-Ausgänge hängen nur vom momentanen Zustand ab. Sie können sich daher nur in Folge eines Zustandswechsels ändern.
19. Hier nur die prinzipielle Idee in Form eines C-Skeleton:

```
...
// within a process sensitive for input- and present_state-changes
// next state an output logic
switch (present_state) {
  case state_1:  moore_outputs = values ;
                 if input_condition_1 {
                   mealy_outputs = values ;
                   next_state = value ;
                 } else if and so on ...
                 break ;
  case state_2: moore_outputs = values ;
                if input_condition_x {
                  and so on ...
                break ;
  default: any default action, e.g. illegal state processing
} ;
...
// within a time periodic process
present_state = next_state ; // perform the state transition
...
```

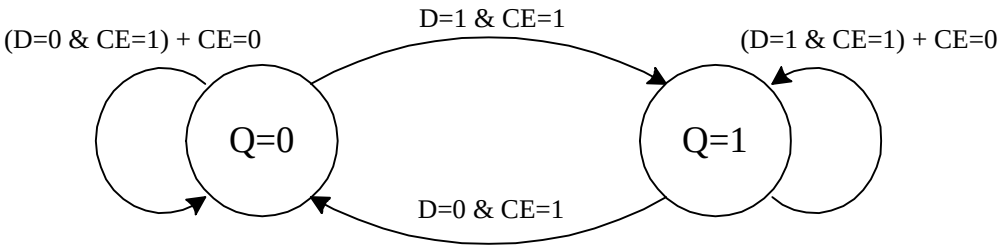
20. Das (synchrone) DFF als sequentielles Grundelement eines FAs bzw. einer FSM stellt selbst eine elementare FSM dar. Der innere Zustand Q (gleich dem Ausgang Q,- Moore-Ausgang) ändert sich gemäß dem D-Eingang (Transitionsbedingung) mit der nächsten steigenden Taktflanke (darunterliegendes HW-Modell, nicht im STD sichtbar).

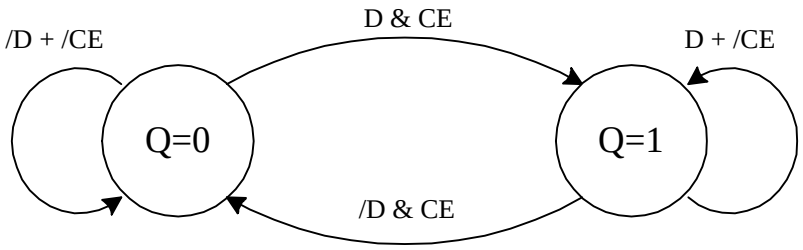


21. Die prinzipielle Struktur des STD ändert sich nicht, egal wie viele synchrone Zusatzfunktionen eingebaut werden. Ganz allgemein gilt diese Struktur für alle 1-Bit Flip-Flops (eh klar: 1 Bit Speicher = 2 Zustände, Zustandswechsel muss möglich sein, Zustand halten muss möglich sein)

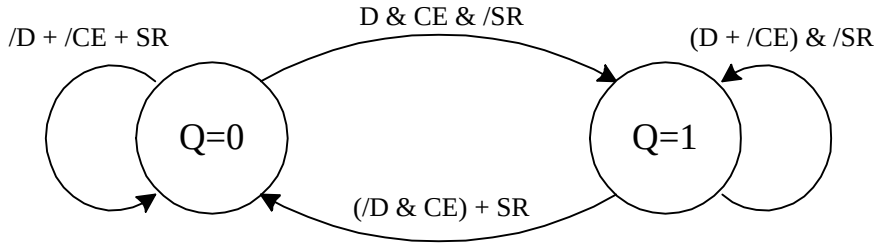


22. Die normale DFF-Funktion ist gegeben, wenn CE=1. Wenn CE=0, dann kann der Zustand nicht gewechselt werden. Die erste Abbildung zeigt die nicht optimierten Übergangsfunktionen. Die zweite Abbildung zeigt die optimierten Übergangsbedingungen in boolscher Notation (der Querstrich vor einem Teilausdruck bedeutet de Inversion des Teilausdrucks, z.B.: /D = not(D)).

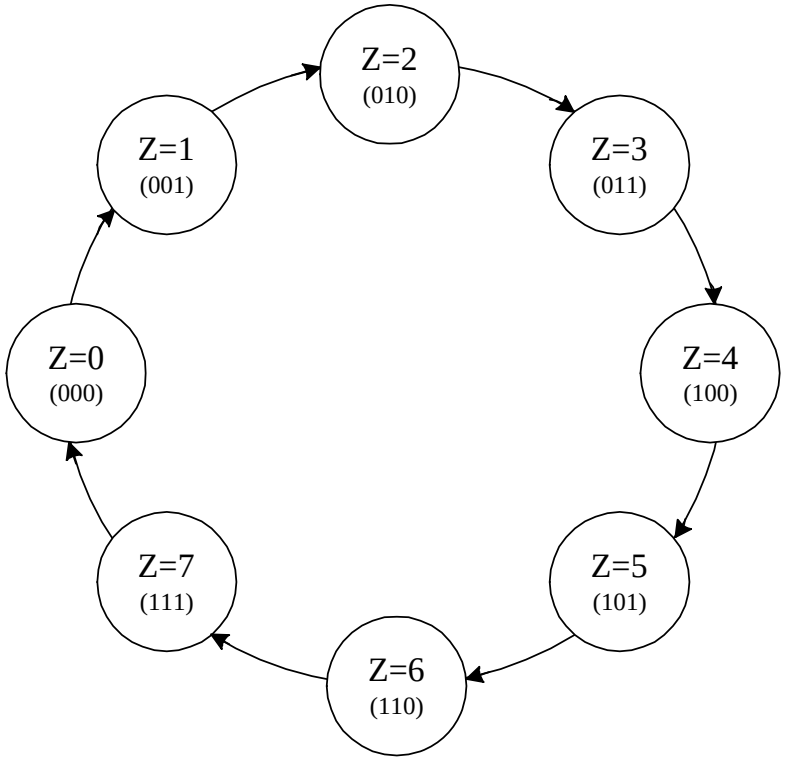




23. Übergangsbedingungen optimiert als boolesche Ausdrücke:

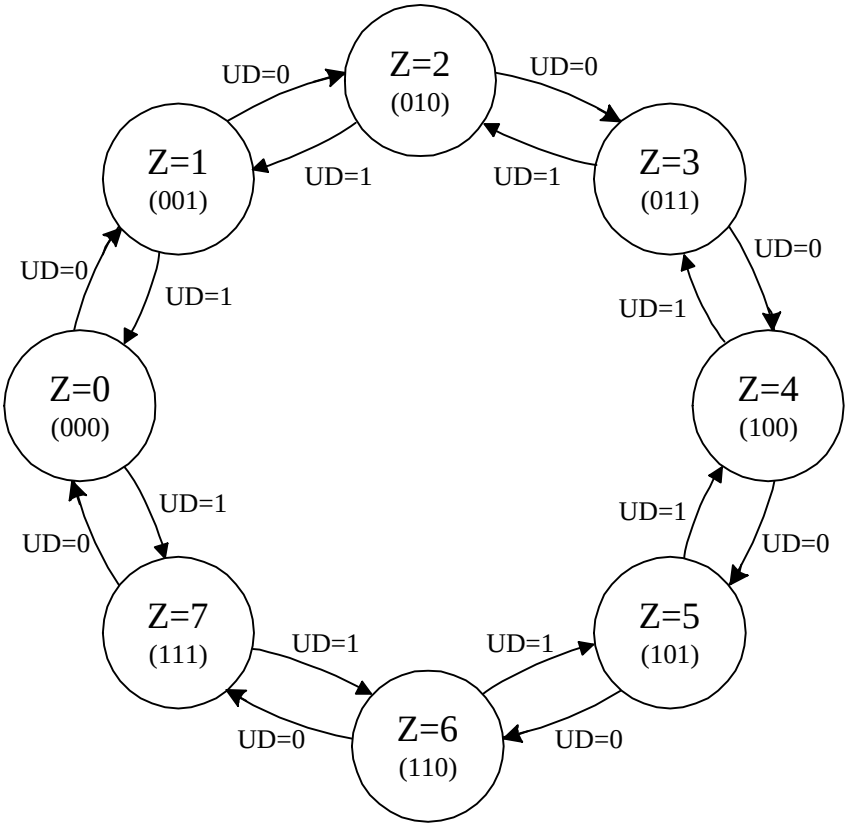


24. Vergleiche dazu auch Studienbrief *Synchrone Systeme* / Kapitel 1.1.1 (Der binäre Aufwärtszähler). Unbeschriftete Übergänge bedeuten unbedingte Übergänge. Man könnte sie auch mit einer „1“ beschriften (d.h. Bedingung immer erfüllt; vgl. dazu in der Programmiersprache C die Anweisung `while(1) {...}` mit der eine Endlosschleife realisiert werden kann).



25. Siehe dazu: *Synchrone Systeme* / Kapitel 1.1.1 (Der binäre Aufwärtszähler).

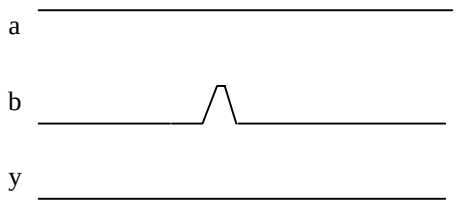
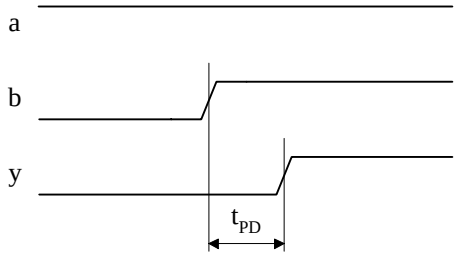
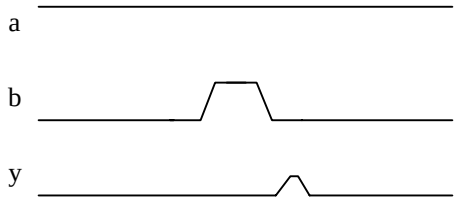
26. Mit dem FSM-Steuereingang UD (UD=0 bedeutet aufwärts und UD=1 bedeutet abwärts) kann die Zählrichtung umgeschaltet werden.



27. Aufbauend auf dem Kapitel 1.1.1 (Der binäre Aufwärtszähler) aus dem Studienbrief *Synchrone Systeme* muss lediglich der Addierer durch eine Addier/Subtrahierschaltung (siehe dazu Studienbrief *Kombinatorische Integer-Arithmetik*) ersetzt werden. Der zusätzliche FSM-Eingang UD ist der Steuereingang ADDSUB der Addier/Subtrahierschaltung.

6. Lösungen

Lösung zu Aufgabe 1:

Fall 1: Die Störung ist so kurz, dass das Gatter davon „nichts mit bekommt“.	
Fall 2: Eingang B schaltet von 0 auf 1 und bleibt 1,- der Ausgang folgt nach der Verzögerungszeit	
Fall 3: Die Störung ist länger als im Fall 1, jedoch noch immer kürzer als die Laufzeit des Gatters. Der Ausgang verhält sich „nicht digital“.	

Lösung zu Aufgabe 2:

$input\_1 = \overline{input\_2} \cdot \overline{input\_4}$, {input_1, input_2, input_4} bilden eine Partition.

$input\_6 = \overline{input\_3}$

$input\_5 = 1$