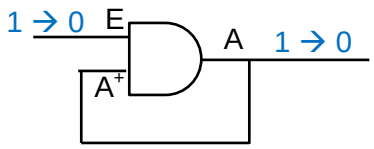
Schaltwerke

Als Schaltwerke bezeichnet man Logikschaltungen mit Speicher, die interne Zustände behalten können. Besitzt der Speicher neben den statischen Dateneingängen noch einen dynamischen Eingang, der als Takteingang bezeichnet wird, handelt es sich um ein "synchrones Schaltwerk". Der Takt markiert den Zeitpunkt des Speicherns. Abhängig davon, ob das Speichern durch einen Taktpegel oder durch eine Taktflanke ausgelöst wird, werden die Speicher in "Latches" (von Schloss, Riegel) und "Flip-Flops" (vom gleichnamigen Strandschuh) eingeteilt.

Das Set- / Reset-Latch

Speicher sind Logikschaltungen mit Hafteigenschaften, d.h. ein einmal angelegter Logikpegel bleibt gespeichert. Z. B.:



	E	A^+	A
a)	1	1	1
b)	0	0	0
c)	1	0	0 (gespeichert)
d)	0	1	--

Trennt man gedanklich die Verbindung zwischen A und A^+ auf, sind die Zeilen a) und b) sofort einsichtig. Da in b) A und A^+ den gleichen Wert haben, kann man die Verbindung jetzt schließen, ohne dass dies einen Einfluss auf den Ausgang hat. Nun ändert sich aber der Ausgangspegel nicht mehr, unabhängig vom Pegel an E. Im verbundenen Zustand ist die Kombination d) nicht herstellbar, da E den Wert 0 an A erzwingt.

Um den Speicher universell einsetzen zu können, muss es möglich sein, den Anfangszustand wiederherstellen zu können. Er braucht also mindestens 2 Eingänge und 1 Ausgang:

- einen Setzeingang (engl. "set", Abk. "S"), und
- einen Rücksetzeingang (engl. "reset", Abk. "R"), und außerdem
- einen Ausgang, der üblicherweise mit "Q" bezeichnet wird

Die boolesche Funktion des Speichers wird am Einfachsten mit Hilfe einer Wahrheitstabelle notiert, die das gewünschte Verhalten angibt. Wieder geht man davon aus, dass eine Rückführung des Ausgangs auf den Eingang erfolgt und definiert dafür eine Hilfsgröße Q^+. Man bestimmt nun die Funktion, die Q so in ein Q^+ umsetzt, dass Speichern auftritt:

S	R	Q	Q^+
0	0	0	0 (speichern)
0	0	1	1 (speichern)
0	1	0	0 (rücksetzen)
0	1	1	0 (rücksetzen)
1	0	0	1 (setzen)
1	0	1	1 (setzen)
1	1	0	x (undefiniert)
1	1	1	x (undefiniert)

Hier eine "2D-Form" der Wahrheitstabelle, das sog. Karnaugh Diagramm. Man erkennt darin besser, welche Vereinfachungen möglich sind:
Q^+ ist …

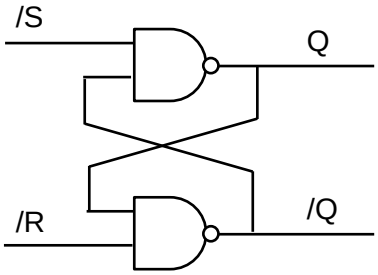
	R		/R	
S	x	x	1	1
/S	0	0	1	0
	/Q	Q		/Q

Aus dem Karnaugh Diagramm liest man die minimale boolesche Funktion ab:

$$Q^+ = S + Q \bullet \overline{R}$$

Diese Form ist technologisch aufwendig, da sie zwei verschiedene Operationen benötigt. In integrierten Schaltungen sind gleichartige Verknüpfungen besser realisierbar.

Eine gern benutzte Möglichkeit ist die Verwendung von zwei NAND-Gliedern:

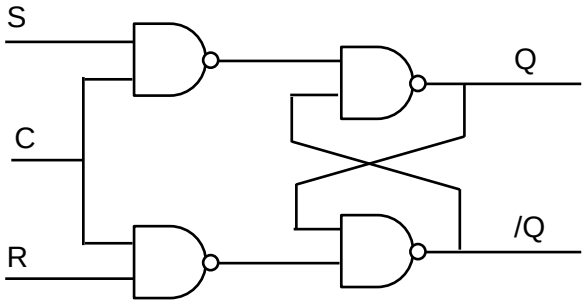


Beachten Sie, dass $Q^+(S, R, Q)$ die Verkopplung definiert, die dazu führt, dass sich folgende Ergebnisse einstellen:

S=0 und R=0 → $Q^+ = Q$ (speichern)
S=1 und R=/S → $Q^+ = Q = 1$
R=1 und S=/R → $Q^+ = Q = 0$

Für S=R=1 sind beide Ausgänge 1, beim anschließenden Speichern kommt es aber zu einer "critical race" mit undefiniertem Ausgang!

In der angegebenen Form ist die Schaltung eine rein asynchrone Form eines Speichers. Dadurch ist der Einsatzbereich relativ beschränkt (z.B. zum Entprellen mechanischer Schalter). Um einen definierten Übernahmezeitpunkt zu erhalten, wird eine Schaltung vor die Eingänge gesetzt, die einen zusätzlichen Takteingang mitbringt: das "Clock Gate".



C = 1: die Stellung der Eingänge S und R entscheidet über die Werte an den Ausgängen

C = 0: S und R haben keinen Einfluss auf die Ausgänge, die Schaltung speichert den letzten Zustand unter C = 1

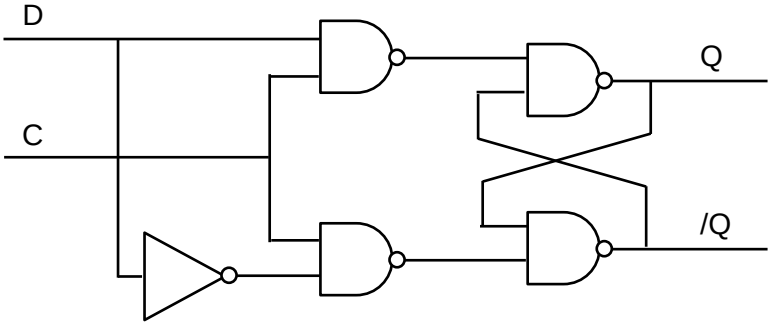
Der kritische Fall S = R = 1 ist immer noch möglich! Der Speicher kann jetzt aber für das Design taktsynchroner Logik verwendet werden und wird als "SR-Latch" bezeichnet.

Die Funktionsgleichung lautet:

$$Q^+ = S \bullet C + \overline{R} \bullet Q + Q \bullet \overline{C}$$

Das D-Latch

Das D-Latch ist eine Erweiterung des SR-Latch in der Form, dass der nicht-speicherbare Zustand (S=R=1) durch eine zusätzliche Beschaltung ausgeschlossen wird. Das D-Latch hat nur noch einen Takt- und einen Dateneingang:



Die Funktionsgleichung berücksichtigt die Bedingung D = S = /R:

$$Q^+ = D \bullet C + Q \bullet \overline{C}$$

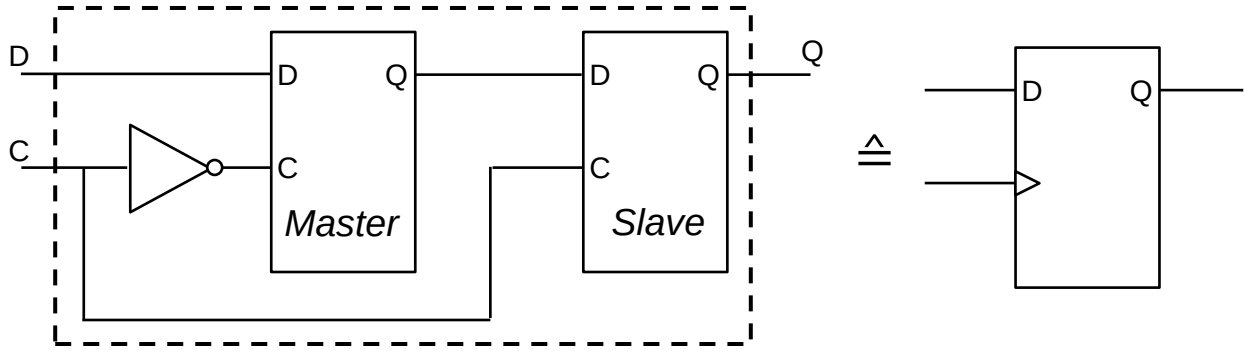
C = 1: der Ausgang Q folgt dem Dateneingang D

C = 0: die Schaltung speichert den letzten Wert von D, der unter C = 1 anlag

Das D-Flip-Flop

Das D-Latch hat einen einfachen Aufbau, braucht wenige Schaltelemente und fügt sich problemlos in kombinatorische Logik ein. Daher ist es verbreitet im Einsatz. Das Verhalten bei aktivem Taktpegel kann jedoch störend sein, da der Dateneingang direkt mit dem Ausgang verbunden ist. Dem kann man begegnen, indem die Schaltung so modifiziert wird, dass der Dateneingang nur bei Änderungen des Taktpegels auf den Ausgang wirkt. Das Ergebnis ist ein "Flip-Flop".

Grundsätzlich erreicht man die Sensibilisierung auf Taktflanken durch den Einsatz von zwei Latches, die miteinander gekoppelt sind:



Flip-Flops eignen sich deshalb ganz besonders für den Einsatz in synchronen Schaltwerken, die im nächsten Abschnitt behandelt werden.

Abhängig von der inneren Beschaltung gibt es noch eine Anzahl weiterer Flip-Flops (SR-FF, JK-FF, Toggle-FF, …), die im Rahmen dieser Einführung in die Schaltwerke nicht behandelt werden.

Technische Ausführungen von Latches und Flip-Flops besitzen noch einen oder mehrere der folgenden Eingänge:

Clear oder Reset	setzt den Ausgang auf 0; wirkt meist asynchron
Preset	setzt den Ausgang auf 1; wirkt meist asynchron
Enable	synchroner Eingang, der den Takt für das speichernde Element aktiviert (dient zur Kopplung synchroner Logik mit gemeinsamem Takt)

Synchrone Schaltwerke

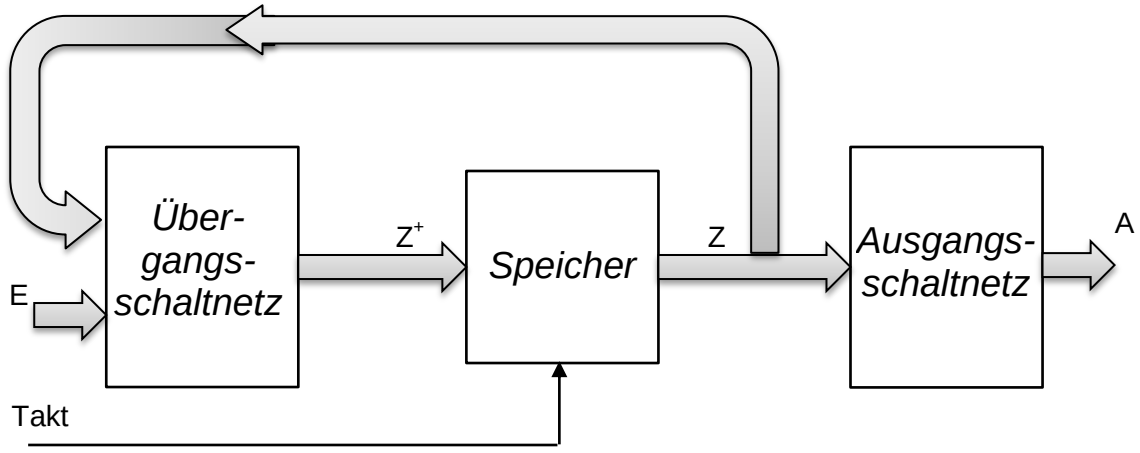
Synchrone Schaltwerke zeichnen sich dadurch aus, dass ihre speichernden Elemente durch gemeinsame Taktung Zustandsfolgen zeitlich hintereinander abarbeiten können. Die wissenschaftliche Basis liefert die Theorie endlicher, deterministischer Automaten (finite state machines). Sie behandelt allgemein das Ein-/Ausgangsverhalten von Systemen mit einer beschränkten Anzahl innerer Zustände, soweit überhaupt ein funktionaler Zusammenhang zwischen Ein- und Ausgangswerten vorhanden ist.

Synchrone Schaltwerke lösen gegenüber den Schaltnetzen eine neue Klasse von Aufgabenstellungen, da die Reaktion des Schaltwerks nicht nur von den Eingangsdaten abhängt, sondern zusätzlich vom momentanen inneren Zustand, der typisch in Flip-Flops gespeichert wird. Nachfolgend verwenden wir als Speicher nur D-Flip-Flops.

Für die praktische Realisierung ist die Automatentheorie viel zu allgemein. Es haben sich für Klassen von Problemen Typen von spezialisierten Automaten herausgebildet, die eine geeignete Struktur besitzen und eine Untermenge des allgemeinen Prinzips darstellen. Der in der Computertechnik wichtigste und häufigste Typ ist der Moore-Automat (nach Edward Moore), der im Folgenden als Beispiel vorgestellt wird.

Der Moore-Automat

Beim Moore Automat wird aus dem inneren Zustand und den Eingangssignalen durch ein Schaltnetz ein Folgezustand erzeugt, der beim nächsten Takt zum neuen inneren Zustand wird. Die Ausgangssignale werden alleine aus dem inneren Zustand abgeleitet, d.h. ein Schaltnetz am Ausgang ist nicht zwingend notwendig, kann aber sinnvoll sein.



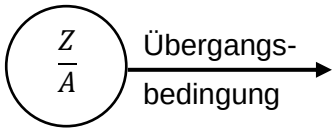
Die Taktperiode muss länger gewählt werden als die maximale Laufzeit des Signals im Übergangsschaltnetz! Hohe Taktraten sind deshalb nur mit einem optimierten Schaltungslayout möglich, d.h. mit speziell gefertigten Silizium Chips (ASIC: application-specific integrated circuit). Programmierbare Logikbausteine (FPGA, CPLD) sind um den Faktor 10 bis 100 langsamer!

Im einfachsten Fall reduziert sich das Schaltungsdesign auf die Bestimmung des Übergangsschaltnetzes.

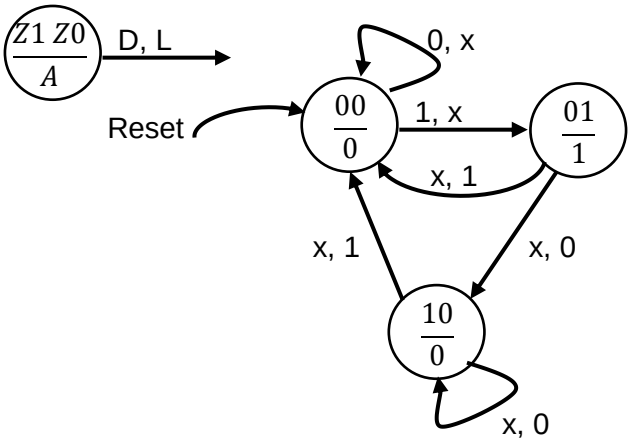
Das Zustandsdiagramm

Im Zustandsdiagramm werden die internen Zustände und deren Abfolge grafisch notiert. Beim Moore Automat sind keine Ausgaben bei Zustandsübergängen möglich, somit werden die Ausgabendaten mit den Zuständen notiert. An die Verbindungen zwischen Zuständen werden die Übergangsbedingungen angeschrieben.

Grundsätzlich sieht die "Legende" so aus:



Beispiel:
Es ist die Zustandsfolge eines Moore-Automaten anzugeben, der erkennt, ob ein Eingangssignal D auf 1 steht. Ist es 1, soll für genau 1 Takt eine 1 ausgegeben werden. Erst wenn an einem zweiten Eingang L eine 1 eingegeben wird, darf der Vorgang von vorne beginnen.



Merke:
- Von einem Zustand gehen immer so viele Übergänge aus, wie Kombinationen von Eingangswerten möglich sind, mindestens aber einer.
- Jedes Zustandsdiagramm braucht einen Startpunkt, der nach Reset eingenommen wird.

Die Zustandsfolgetabelle
Um die boolesche Funktion des Übergangsschaltnetzes zu ermitteln, wird eine Wahrheitstabelle über den Zusammenhang zwischen dem aktuellen Zustand, zusammen mit den Eingangssignalen, und dem Folgezustand aufgestellt. Außerdem kann die Wahrheitstabelle die Ausgangssignale enthalten, sodass auch das Ausgangsschaltnetz darin beschrieben ist. Übergänge zwischen Zuständen erfolgen immer mit einem Takt, der in der Tabelle jedoch nicht explizit auftritt. Diese Form der Wahrheitstabelle für synchrone Schaltwerke wird als "Zustandsfolgetabelle" bezeichnet.

Beispiel:

Z1	**Z0**	**D**	**L**	**Z1⁺**	**Z0⁺**	**A** (=Z0)
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	0	1
0	1	0	1	0	0	1
0	1	1	0	1	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	0	1	1	0	0	0
1	1	x	x	?	?	?

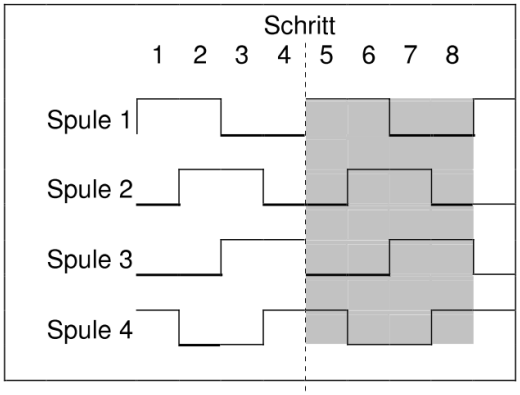
Im Beispiel sind 2 Speicherbits für den inneren Zustand des Automaten physikalisch notwendig, es werden aber nur 3 der 4 möglichen Zustandswerte gebraucht. Der Zustand "3" ist jedoch prinzipiell möglich, wenn auch bei korrekter Realisierung nur aufgrund externer Störungen einnehmbar. Der Zustand 3 wird als unerlaubter (bzw. "illegaler") Zustand bezeichnet. In produktiven Designs müssen unerlaubte Zustände behandelt werden, sodass auch bei Fehlern immer ein definiertes Verhalten des Automaten vorliegt.

Weitere Designregeln:

- Alle Zustandsspeicher eines Automaten werden vom selben Takt versorgt.
- Keine kombinatorische Logik in der Taktversorgung.
- Keine kombinatorische Logik in asynchronen Eingängen (Reset, Preset).
- Kopplung von Automaten nur über "Enable" Eingänge.
- Asynchrone Eingangssignale oder Signale von Automaten mit anderem Takt müssen vor dem Automaten auf dessen Takt synchronisiert werden (physikalische Notwendigkeit für Kippschaltungen wie Flip-Flops).

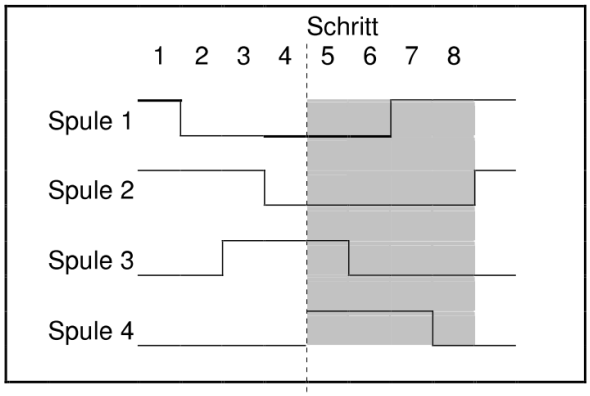
<u>Übungsbeispiele (Labor):</u>

- Geben Sie die Logik für das Übergangsschaltnetz für das obige Beispiel an. Testen Sie den Automaten am Evaluationsboard.
- Schaltwerke ohne externen Eingang (außer Reset und Takt) werden als Zähler bezeichnet. Realisieren Sie einen 2 Bit Graycode Zähler, der folgende Zählfolge realisiert: 00 → 01 → 11 →10 → 00 … (Takt: mit Taster B1, von Hand)
- Erstellen Sie einen Automaten, der die Steuersequenz für einen Schrittmotor mit 4 Phasen und 400 Schritten je Umdrehung erzeugt. Takten Sie ihn mit 10ms/Schritt:



Ch1	1	1	0	0	1	1	0	0	0xCC
Ch2	0	1	1	0	0	1	1	0	0x66
Ch3	0	0	1	1	0	0	1	1	0x33
Ch4	1	0	0	1	1	0	0	1	0x99

Abb. 4 Vollschritt



Ch1	1	0	0	0	0	0	1	1	0x83
Ch2	1	1	1	0	0	0	0	0	0xE0
Ch3	0	0	1	1	1	0	0	0	0x38
Ch4	0	0	0	0	1	1	1	0	0x0E

Abb. 5 Halbschritt

Durch eine Ansteuerung nach Abb. 4 (Vollschritt) dreht sich der Motor in eine Richtung. Pro Umdrehung werden 200 Schritte benötigt. In der Regel haben diese Schrittmotoren eine maximale Pulsrate von 200 Hz (5 ms). Um diese Pulsrate zu erreichen muss der Motor allmählich beschleunigt bzw. wieder abgebremst werden. Wenn man die Pulssequenz umkehrt, so ändert sich die Drehrichtung des Motors.
Durch eine Ansteuerung nach Abb. 5 (Halbschritt) werden zu den vier Zuständen in Abb. 4 weitere vier Zwischenzustände eingefügt. Dadurch braucht der Motor nun 400 Schritte pro Umdrehung. Der Drehwinkel pro Schritt hat sich also um die Hälfte reduziert.
Es gibt noch weitere Möglichkeiten Schrittmotoren anzusteuern um zum Beispiel das Drehmoment oder die Präzision zu verbessern. Diese werden hier nicht weiter betrachtet.