- Kapitel 2 -
Sequentielle Digitalschaltungen

Ausgabestand 1.1

- Nur zur Information -

2. Sequentielle Digitalschaltungen

Wie kombinatorische lassen sich auch sequentielle Schaltungen auf nahezu beliebig trickreiche Weise aufbauen. In der Praxis hat es sich aber seit langem bewährt, komplexe sequentielle Schaltungen aus elementaren (wie Flipflops, Schieberegistern, Zählern usw.) zusammenzusetzen. Wir wollen deshalb auf solche Grundschaltungen etwas näher eingehen. Dabei beschränken wir uns auf einen Überblick, der üblicherweise ausreicht (und den Zugang zum speziellen Fachschrifttum erleichtert, insbesondere zu den Applikationsschriften der Schaltkreishersteller).

2.1. Grundbegriffe

Kombinatorische Schaltungen liefern ihre Ergebnisse unmittelbar nachdem die Operanden-Werte an den Eingängen anliegen. Gültige Ergebnis-Belegungen sind nach einer Durchlaufzeit zu erwarten, die sich näherungsweise als Produkt von Gatterlaufzeit und Schaltungstiefe ergibt. Solche Schaltungen haben keine "Vorgeschichte"; jede Operanden-Kombination ergibt immer wieder dasselbe Ergebnis. Kombinatorische Zuordnungen lassen sich aber nur für beschränkte Operandenlängen und elementare Operationen technisch verwirklichen. Sind die beteiligten Informationsstrukturen zu lang bzw. die Informationswandlungen zu kompliziert, so ist die Ausführung in mehreren aufeinanderfolgenden Zeitschritten zu organisieren. Dafür sind elementare Speichermittel notwendig, die direkt mit kombinatorischen Netzwerken zusammengeschaltet sind. Derartige Einrichtungen wollen wir nachfolgend behandeln. Wir setzen dabei voraus, daß die elementaren Speicherfunktionen durch geeignete Zusammenschaltung *logischer* Funktionseinheiten (z. B. von Gattern) verwirklicht werden und nicht durch Ausnutzung *physikalischer* Effekte (Magnetismus, Ladungsspeicherung usw.).

Erklärung:
Aus Aufwands- bzw. Kostengründen hat man, von Beginn der Entwicklung an bis in die 70er Jahre hinein, kombinatorische Verknüpfungen und Speicherfunktionen zumeist auf unterschiedliche Weise realisiert. Um dies zu verstehen, müssen wir uns nur vorstellen, wir müßten eine elementare, auf "logischer" Grundlage beruhende Speicherzelle (ein Latch oder Flipflop) mit einzelnen Transistoren oder gar mit Elektronenröhren aufbauen (vgl. Abbildung 3.1). Verständlich, daß man sich alle möglichen Tricks hat einfallen lassen, um den Aufwand zu vermindern. So gab es Speicherelemente auf Grundlage von Ferritkernen, Umlaufspeicher mit Ultraschall-Verzögerungsleitungen, Ladungsspeicher mit Kondensatoren usw. Erst durch die Technologien der integrierten Schaltkreise sind die aus Gattern aufgebauten Speicherelemente zur Selbstverständlichkeit geworden. Wir finden aber auch in modernen Systemen sparsamere Lösungen. Insbesondere wird das Prinzip der Ladungsspeicherung ausgiebig genutzt. Das betrifft vor allem die DRAMs und die modernen Festwertspeicher (EPROM, EEPROM, Flash-ROM).

Zustand (Maschinen- bzw. Automatenzustand)
Eine Schaltung befindet sich zu einer beliebigen Zeit t in einem einzigen bestimmten Zustand z_i, der durch die jeweilige Belegung der Speichermittel bestimmt ist. Die Schaltung verharrt so lange in diesem Zustand, bis ein *Zustandswechsel* $z_i \rightarrow z_j$ bzw. $z(t) \rightarrow z(t+1)$ veranlaßt wird. Auslösende Ursachen sind entweder (1) externe *Ereignisse* oder (2) interne *Zeitvorgaben* (Takte).

Maschinenzyklus
Das Zeitintervall zwischen zwei Zustandswechseln bezeichnen wir als Maschinenzyklus (in anderer Redeweise: ein Maschinenzyklus ist ein Zeitabschnitt, in dem sich der Maschinenzustand nicht ändert).

Speicherung
Speichern ist das Halten eines Zustandes über die Zeit.

Takt
Takte sind zeitbestimmende Signale. Zustandswechsel werden durch Takte ausgelöst. In den meisten Fällen ist ein Takt eine kontinuierliche Folge von Impulsen mit konstanter Folgefrequenz (es gibt aber auch Einzeltakte und Takte mit veränderlicher Folgefrequenz).

Statische Arbeitsweise
Statisch arbeitende sequentielle Schaltungen sind bei beliebig langer Dauer des einzelnen Maschinenzyklus noch arbeitsfähig; man kann also den Takt anhalten. Statisch arbeitende Schaltungen (z. B. Prozessoren) sind daran zu erkennen, daß im Datenblatt keine Mindest-Taktfrequenz angegeben ist.

Dynamische Arbeitsweise
Dynamisch arbeitende Schaltungen sind unterhalb einer gewissen Mindest-Taktfrequenz (im Datenblatt angegeben) nicht mehr funktionsfähig. Dies ist durch die Technologie bedingt: man hat, zwecks Aufwandsverringerung, für bestimmte Funktionen dynamische (zeitabhängige) Effekte, z. B. die Ladungsspeicherung, ausgenutzt. Für die Funktionsweise an sich ist dies bedeutungslos; es ist allerdings nicht möglich, den Takt beliebig lange anzuhalten und trotzdem den jeweiligen Zustand zu bewahren.

Wann wäre es nützlich, den Takt anhalten zu können?
Begnügen wir uns mit drwei Beispielen: (1) Signalverfolgung beim Fehlersuchen, (2) Warten auf externe Signale (Wartezustände), (3) Stromsparen (wenn nichts zu tun ist).

Diskrete Maschinenzeit
Für die grundsätzliche Funktionsweise (natürlich nicht für das Leistungsvermögen) ist die absolute Dauer des Maschinenzyklus bedeutungslos. Um die Funktionsweise einer Schaltung zu erklären, kann man also von kontinuierlichen Zeitangaben absehen. Vielmehr spricht man von *diskreten* Zeitpunkten, an denen die Zustandsübergänge stattfinden. Diese werden üblicherweise mit t (aktueller Zeitpunkt), t+1 (nächster Zeitpunkt), t-1 (vorhergehender Zeitpunkt) usw. bezeichnet.

Eingebaute Zeitbestimmung (Self Timing)
Der Schaltkreis hat eingebaute Zeitstufen bzw. Impulsgeneratoren, die Steuersignale, interne Takte usw. im erforderlichen Zeitraster abgeben. Es ist also nicht nötig, Taktsignale von außen zuzuführen. Die wichtigste Anwendung: Speicherschaltkreise.

Asynchrone Arbeitsweise
Der Zeitrahmen ist variabel; die zeitbestimmenden Signale werden in Abhängigkeit vom jeweiligen Verarbeitungsablauf gebildet. Abbildung 2.1 veranschaulicht das einfachste Prinzip.

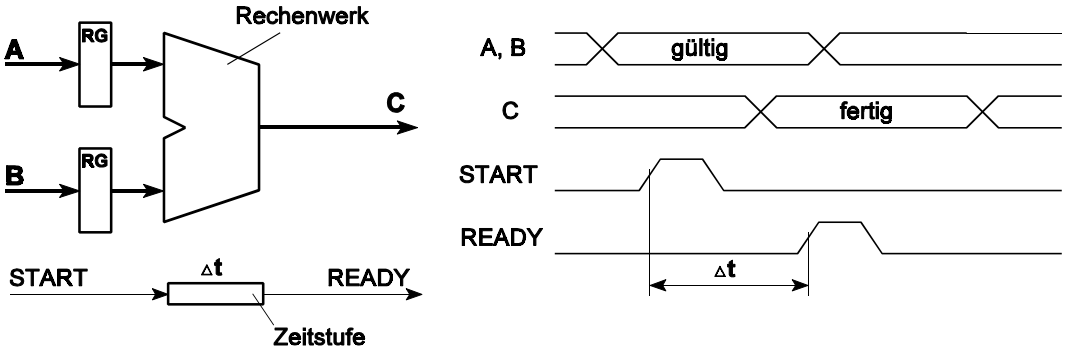


Abbildung 2.1 Beispiel der asynchronen Arbeitsweise: Rechenwerk mit starrer Zeitverzögerung (Self Timing)

Erklärung:
Nach dem Eintragen der Operanden in die Register wird ein Startimpuls gegeben. Dieser Impuls wird über eine Zeitstufe gemäß der maximalen Durchlaufzeit durch die kombinatorischen Schaltungen verzögert (Prinzip des Self Timing). Dieser verzögerte Impuls (READY) dient als Fertigmeldung (und kann beispielsweise als Startimpuls nachgeschalteten Einrichtungen zugeführt werden).

Hier muß man - ähnlich wie beim Festlegen eines Taktrasters - stets den ungünstigsten Fall berücksichtigen. Gibt es verschiedene, von vornherein unterschiedene Durchlauffälle, so kann man entsprechend umschaltbare Zeitstufen vorsehen. Beispielsweise könnte man eine ALU (Abbildung 1.47) so auslegen, daß bei bitweisen logischen Verknüpfungen die Fertigmeldung eher erscheint als bei Verschiebeoperationen und da wiederum eher als beim Addieren.

Höher entwickelte asynchrone Schaltungen signalisieren gleichsam von selbst, wenn sie fertig sind. Hierzu verwendet man beispielsweise je Ein- und Ausgangssignal zwei Leitungen. Ein Impuls auf der einen Leitung bedeutet eine Null, ein Impuls auf der anderen eine Eins. Verharren beide Leitungen auf festen Pegeln, so ist - grob gesagt - nichts los. Um zwei Leitungen je Ausgang beliefern zu können, müssen für jede der beiden Ausgangsbelegungen (0 und 1) eigene kombinatorische Netzwerke vorgesehen werden (Abbildung 2.2).

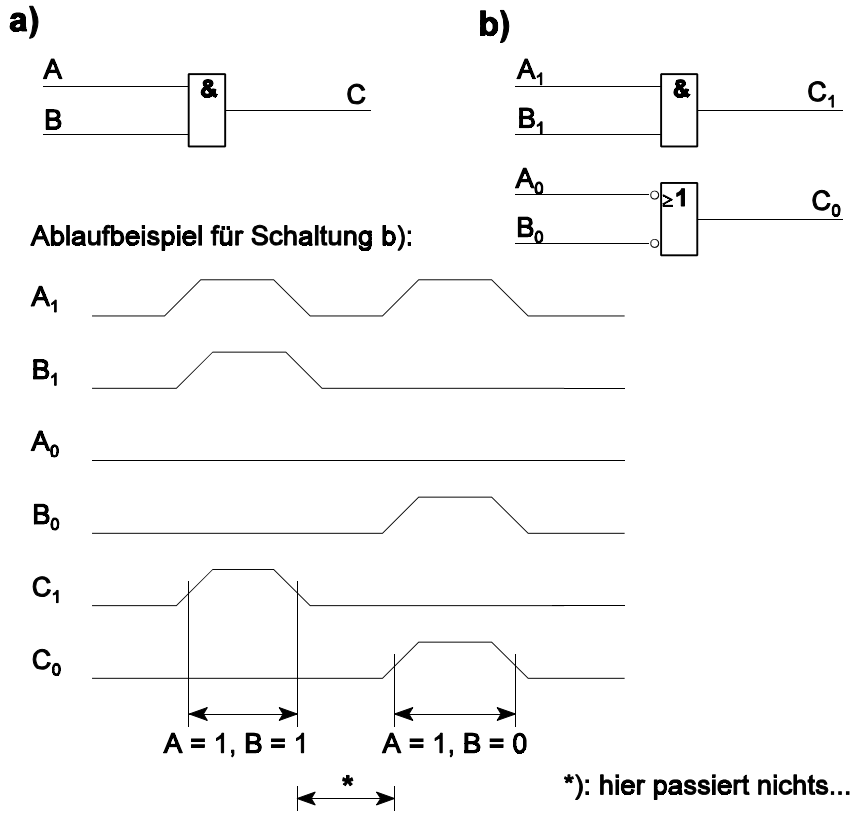


Abbildung 2.2 Kombinatorische Schaltungen im Vergleich. Beispiel: Übertragsbildung im Halbaddierer

Erklärung:
Wir verwenden eines der einfachsten Beispiele: die Bildung des Übertrags beim Halbaddierer.

a) die herkömmliche Auslegung. Ein Übertrag C ist zu bilden, wenn das Summandenbit A UND das Summandenbit B = 1 sind: $C = A \cdot B$. Ansonsten ist der Übertrag = 0[*)].

b) die Auslegung für eine asynchrone Maschine. A_0, B_0, C_0 führen die Nullimpulse, A_1, B_1, C_1 die Einsimpulse. Hier sind beide ausgangsseitigen Impulse C_0, C_1 zu bilden. Also:
- der Übertrag ist = 1, wenn A = 1 UND B = 1 sind: $C_1 = A_1 \cdot B_1$ (wie gehabt),
- daß der Übertrag = 0 ist, müssen wir jetzt gesondert signalisieren. Damit er Null wird, genügt es, daß einer der Operanden = 0 ist: C = 0, wenn A = 0 ODER B = 0; $C_0 = \overline{A_0} \vee \overline{B_0}$.

*): das erledigt das UND-Gatter gleichsam nebenbei (indem es den Wert 0 liefert, sofern nicht beide Eingänge = 1 sind). Wir müssen uns klarmachen: in taktgesteuerten (synchronen) Schaltungen sind die Ausgänge von kombinatorischen Verknüpfungen nur in Bezug auf den Takt von Bedeutung. Ohne diesen Zeitbezug sehen wir es - wir bleiben bei unserem Beispiel - dem Übertragssignal C nicht an, ob der Übertrag = 0 ist, ob er = 1 ist oder ob zur Zeit einfach überhaupt nichts passiert.

Änderungen erkennen (Transition Detection)
Ein weiteres Prinzip, das in manchen asynchronen Schaltungen genutzt wird (z. B. in modernen SRAMs)[*)]. Die bisherige Signalbelegung wird gespeichert und mit den zur Zeit anliegenden

Signalwerten verglichen. Wird dabei eine Änderung erkannt, so beginnen die internen Zeitgeberschaltungen anzulaufen. Die neue Signalbelegung wird dabei gespeichert und so zur bisherigen.

*): eine weitere Anwendung: besonders stromsparende Logikschaltkreise (z. B. manche CPLDs). Diese Schaltkreise warten darauf, daß sich an den Eingängen etwas tut - erst dann werden die eigentlichen Verknüpfungsschaltungen aktiviert.

Hinweis:
In der Literatur bezeichnet man oft alle Arten von Schaltwerken, die nicht von einem zentral gebildeten Takt gesteuert werden, als asynchron. In der Praxis kann man aber nur ganz einfache Schaltungen im wörtlichen Sinne asynchron auslegen, also ohne jegliche Schaltmittel, die Zeitintervalle vorgeben (Beispiel: einfachste Vermittlungsschaltungen (Arbitration Latches). Sobald es etwas komplizierter wird, braucht man zeitbestimmende Signale. "Asynchron" ist also nicht immer mit "völlig ungetaktet" gleichzusetzen.

Synchrone Arbeitsweise
Taktsignale geben einen festen Zeitrahmen für die Zustandsübergänge im System vor; die zeitbestimmenden Impulsfolgen haben feste und konstante Impulsfolgefrequenzen (Taktfrequenzen). Abweichungen sind nicht beabsichtigt, sondern lediglich Folge von Fertigungstoleranzen, Temperaturschwankungen usw.

Vollsynchrone Arbeitsweise
Es gibt nur ein Taktsignal, mit dem alle Flipflops des Schaltwerks betrieben werden. Dabei liegt der Takt an allen Flipflops ständig an (ungesteuerter Takt); alle Steuerwirkungen werden ausschließlich über kombinatorische Verknüpfungen vor den Dateneingängen der Flipflops erbracht (Abbildung 2.3). Die in einem bestimmten Taktzyklus (t) gebildeten Signale kommen erst im folgenden Taktzyklus (t+1) an den Ausgängen der Flipflops zur Wirkung (Abbildung 2.4).

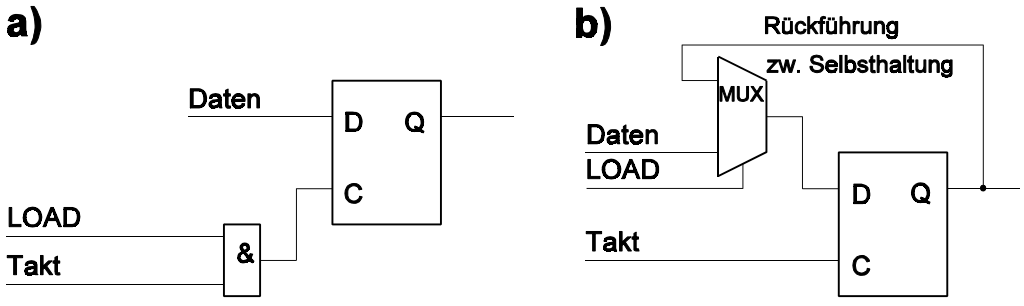


Abbildung 2.3 Zur Veranschaulichung der vollsynchronen Arbeitsweise

Erklärung zu Abbildung 2.3:
Es geht darum, ein Register zu laden (hier durch ein einzelnes Speicherglied - Latch oder Flipflop - symbolisiert). Die Datenübernahme wird durch ein Taktsignal bewirkt und soll durch ein Signal LOAD gesteuert werden. Ist LOAD aktiv, so werden die Daten in das Register übernommen, ist es inaktiv, so bleibt der bisherige Registerinhalt erhalten.

a) eine im allgemeinen Fall der synchronen Arbeitsweise mögliche Auslegung: das Steuersignal gibt den Takt zum Register entweder frei oder sperrt ihn (gesteuerter Takt),
b) vollsynchrone Arbeitsweise. Der Takt liegt ständig am Register an. Die Datenübernahme wird über eine Auswahlschaltung vor den Dateneingängen des Registers gesteuert. Damit - bei ständig anliegendem Takt - der gespeicherte Inhalt nicht schon im nächsten Taktzyklus wieder verloren geht, muß die Belegung der (D-) Flipflops über eine Rückführung gehalten werden.

Hinweis:
Auch in vollsynchroner Hardware werden gelegentlich Register gemäß Abbildung 2.3a beschaltet - und zwar zum Zwecke des Stromsparens (Clock Gating). Näheres in Kapitel 5.

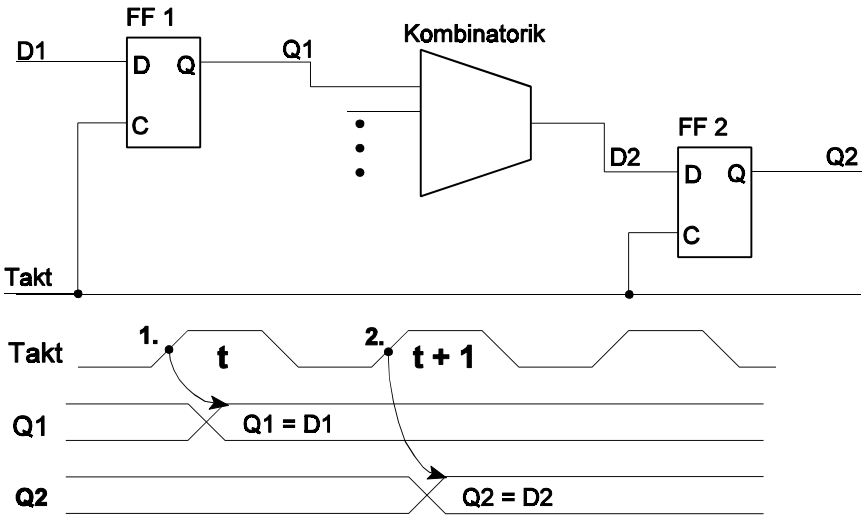


Abbildung 2.4 So schalten Flipflops bei vollsynchroner Betriebsweise

Erklärung:
Die Abbildung zeigt das sehr einfache Modell einer Register-Transfer-Struktur: Register (= Flipflops) - Kombinatorik - Register. Eine bestimmte Datenbelegung D1 wird im ersten Taktzyklus (= zum Zeitpunkt t) in das erste Flipflop (FF 1) übernommen. Nun wird die nachgeschaltete Kombinatorik durchlaufen. Die sich dadurch ergebende Signalbelegung wird erst mit Beginn des zweiten Taktzyklus (= zum Zeitpunkt t+1) in das zweite Flipflop (FF 2) übernommen. Am Ausgang Q2 sehen wir eine Reaktion auf D1 also erst nach der Low-High-Flanke des zweiten Taktzyklus.

Asynchron oder synchron?
Die asynchrone Betriebsweise hat offensichtliche Vorteile: jede Schaltung liefert ihr Ergebnis dann, wenn sie mit dessen Bildung fertig ist, man muß dem Zeitraster keine ungünstigsten Fälle (die nur selten vorkommen) zugrunde legen, die Anpassungen an ein starres Taktschema (Synchronisation) ist nicht notwendig, wenn sich nichts ändert, schaltet auch nichts (= (nahezu) keine Störstrahlung und kein Stromverbrauch) usw. Die Aufwendungen (z. B. zwei Signalleitungen je Bitposition und gesonderte kombinatorische Netzwerke, um jeweils beide Ausgangsbelegungen (Nullen und Einsen) zu bilden) sind aber beachtlich, ebenso die Entwurfsschwierigkeiten[1), 2)]. Deshalb hat sich derzeit die synchrone Arbeitsweise durchgesetzt (Grundgedanke: die Schaltungen werden so einfach wie möglich ausgelegt und so schnell wie möglich betrieben). Dabei hat die vollsynchrone Auslegung besondere Vorteile (Kapitel 5).

Es war bisher immer wieder die Schaltkreistechnologie, die das Leistungsvermögen entscheidend vorangebracht hat; somit war es nie erforderlich, grundsätzlich andersartige Ansätze aus dem akademischen Bereich in die Massenfertigung zu überführen[3)].

Anmerkungen:

1) Addierwerke sind noch vergleichsweise harmlos - sie werden deshalb auch gern als Lehrbeispiele verwendet. Man stelle sich aber vor, ein moderner Hochleistungsprozessor mit seinen Verarbeitungswerken, Caches, Schreibpuffern usw. sollte durchgehend nach solchen Prinzipien ausgelegt werden ...
2) hinzu kommt: die Signalisierung gemäß Abbildung 2.2b ist eigentlich eine RZ-Signalisierung (vgl. den Signalverlauf von A1). In der kürzest-möglichen Signalperiode kann man somit nur 1 Bit übertragen (maximale Datenrate je Signal ≙ Impulsfolgefrequenz). Der Synchronbetrieb ermöglicht hingegen NRZ-Signalisierung (maximale Datenrate je Signal ≙ 2 · Impulsfolgefrequenz).
3) wo solche Prinzipien zuerst in Frage kommen: im Bereich der GHz bei überschaubaren Funktionen (soll heißen: vor allem in den ersten digitalen Stufen der ganz neumodischen Mobiltelefone ...).

Nicht jeder Impuls ist erwünscht: Spikes und Glitches
Beide Fachbegriffe (sprich: Speicks, Glittches) bezeichnen Störimpulse, also zeitweilige Übergänge von Null nach Eins oder umgekehrt, die eigentlich nicht vorkommen sollten. Dafür, daß sie ziemlich oft vorkommen, gibt es vielfältige Ursachen, die teils elektrischer, teils logisch-funktioneller Natur sind. Hier wollen wir uns auf typische Probleme beschränken, die die "Logik" gleichsam von sich aus verursacht (solche Störungen würden auch dann auftreten, wenn ideale elektrische Betriebsbedingungen herrschen würden). Abbildung 2.5 veranschaulicht typische Beispiele.

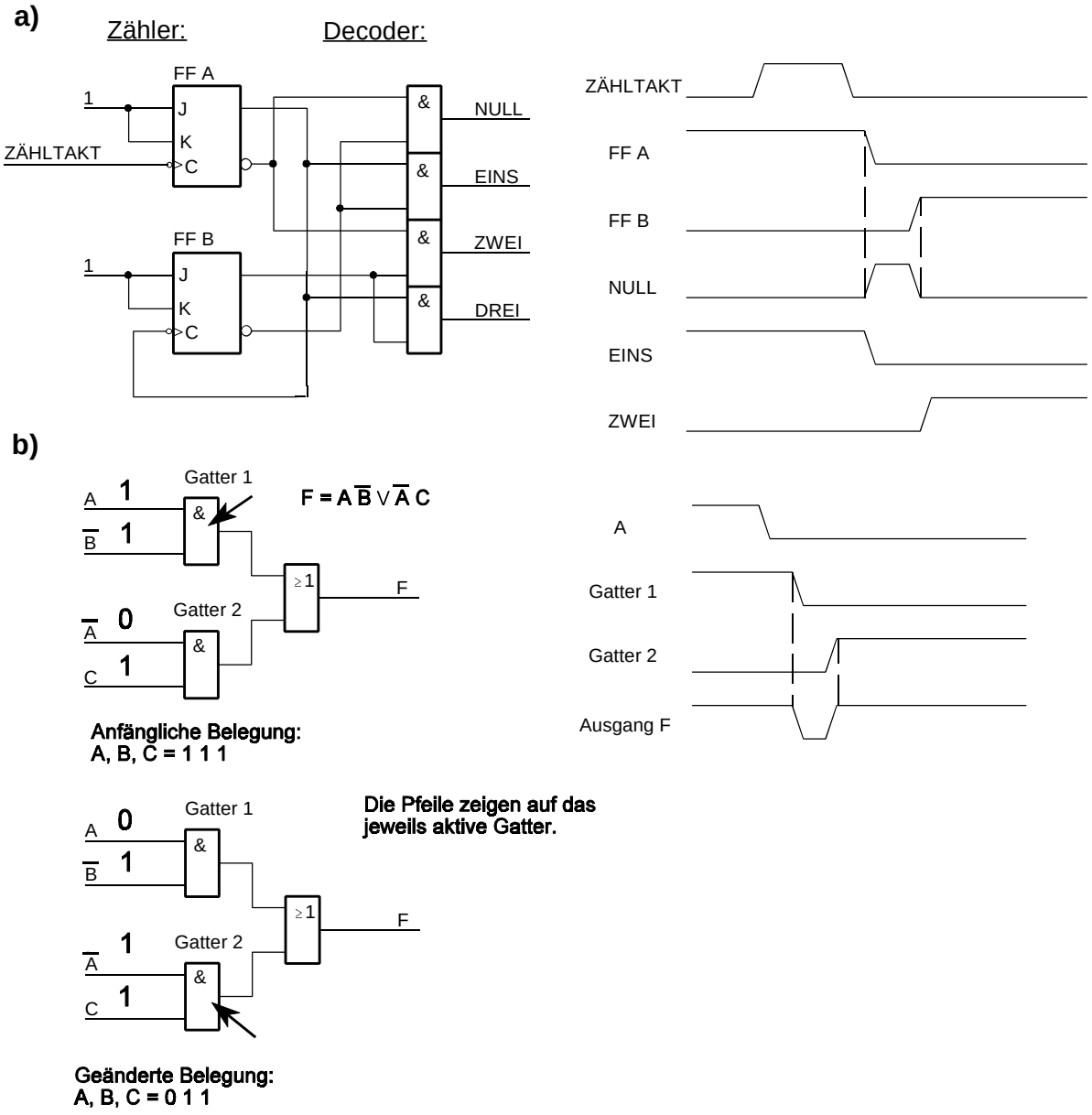


Abbildung 2.5 Typische Ursachen von Glitches (Beispiele)

Erklärung:

a) Unterschiede in den Verzögerungszeiten. Es ist ein Zähler mit nachgeschaltetem Decoder dargestellt. Nehmen wir an, der Zähler befinde sich in Stellung EINS (Belegung 0, 1). Jetzt tritt ein Zählimpuls auf; der Zähler gelangt in Stellung ZWEI (Belegung 1, 0), Nun schalten beide Flipflops nie mit absolut gleichen Verzögerungszeiten. Schaltet Flipflop A eher als Flipflop B, so tritt kurzzeitig die Belegung 0, 0 auf - und damit ein Störimpuls am Decoderausgang NULL (siehe Impulsdiagramm). Sinngemäß wird ein Störimpuls am Decoderausgang DREI auftreten, wenn Flipflop B eher schaltet als Flipflop A (kurzzeitige Belegung 1, 1).

b) Wettlauferscheinungen (Hazards) in kombinatorischen Netzwerken. Solche Erscheinungen können auch dann auftreten, wenn sich nur ein einziges Eingangssignal ändert. In unserem Beispiel ändert sich die Eingangsbelegung von 111 auf 011. Deswegen dürfte sich der Ausgang F nicht ändern; er müßte auf Eins verharren. Nun wird aber vor der Eingangsänderung die Eins von Gatter 1 geliefert, nach der Änderung hingegen von Gatter 2. Sollte Gatter 1 schneller abschalten (1 → 0) als Gatter 2 zuschaltet (0 → 1), so erscheint am Ausgang F ein kurzer Nullimpuls (siehe Impulsdiagramm).

Hinweise:

1. An den Ausgängen einzelner UND-Gatter (das betrifft u. a. auch Decoder (Abbildung 2.5a)) entstehen dann keine Störimpulse, wenn sich zu einer Zeit jeweils nur eine Eingangsbelegung ändert (solche Signalverläufe sind *glitchfrei* zu decodieren).
2. Unter der Bedingung, daß sich zu einer Zeit nur eine Eingangsbelegung ändert, kann man zweistufige kombinatorische Netzwerke so auslegen, daß Wettlauferscheinungen der soeben beschriebenen Art (Abbildung 2.5b) nicht vorkommen. Das erfordert aber höheren Aufwand.
3. In vollsynchronen Schaltungen schaden derartige Effekte nicht, sofern man nur einen hinreichend langen Taktzyklus vorsieht (Kapitel 5).
4. Auch dann, wenn Glitches aus funktioneller Sicht nicht schaden, haben sie doch unerwünschte Nebenwirkungen: nämlich (1) eine intensivere Störstrahlung (Stichwort: EMV) und (2) - in CMOS-Schaltungen - einen höheren Stromverbrauch.

2.2. Latches und Flipflops

2.2.1. Speicherung durch gesteuerte Selbsthaltung

Wie können wir mit "rein logischen" Mitteln, ohne direkte Ausnutzung physikalischer Effekte, Information speichern? Wir gehen systematisch vor und betrachten zunächst das einzelne Bit (Abbildung 2.6). Wenn wir den Ausgang eines ODER-Gatters auf einen seiner Eingänge zurückführen (Abbildung 2.6a), brauchen wir am anderen Eingang nur einmal eine Eins anzulegen. Daraufhin wird das Gatter die Ausgangsbelegung "Eins" über die Rückführung selbst halten.

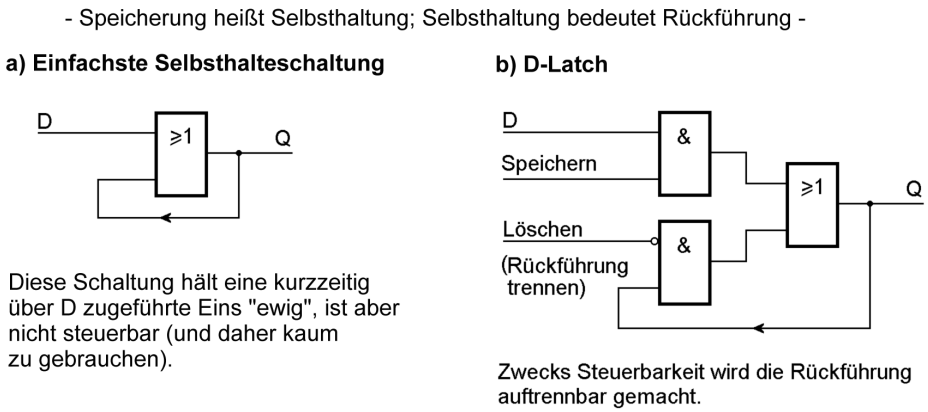


Abbildung 2.6 Elementare Speicherschaltungen nach dem Prinzip der Selbsthaltung

Wir haben aber keine Möglichkeit, wieder eine Null zu speichern. Damit ist die Schaltung kaum brauchbar. Wie müssen wir sie verbessern? - Offensichtlich bekommen wir den Ausgang des ODER-Gatters wieder auf Null, wenn wir die Rückführung auftrennen. Das gelingt mit einem UND-Gatter (Abbildung 2.6b). Eine solche Schaltung bezeichnet man als D-Latch. Der Fachbegriff (Latch = Klinke; sprich: Lätch) bringt das Selbsthalteprinzip bildhaft zum Ausdruck (das "D" steht für "Datenspeicherung").

Wie das D-Latch von Abbildung 2.6b funktioniert
Der Speichererlaubniseingang ist als aktiv High definiert, der Löscheingang als aktiv Low. Wollen wir ein Datenbit speichern, muß das Speichersignal auf High gesetzt werden. Um die einmal gespeicherte Information zu halten, muß "Speichern" mit Low und "Löschen" mit High belegt sein (damit die Rückführung durchgeschaltet bleibt). Vor jedem erneuten Speichern ist die Rückführung zu trennen, indem "Löschen" auf Low gelegt wird.

2.2.2. RS-Latches

Selbsthalteschaltungen lassen sich auch mit zwei gleichartigen Gattern (NAND oder NOR) aufbauen (Abbildung 2.7). Solche Schaltungen bezeichnet man als RS-Latches. "RS" steht für "Rücksetzen" (Reset) und "Setzen" (Set).

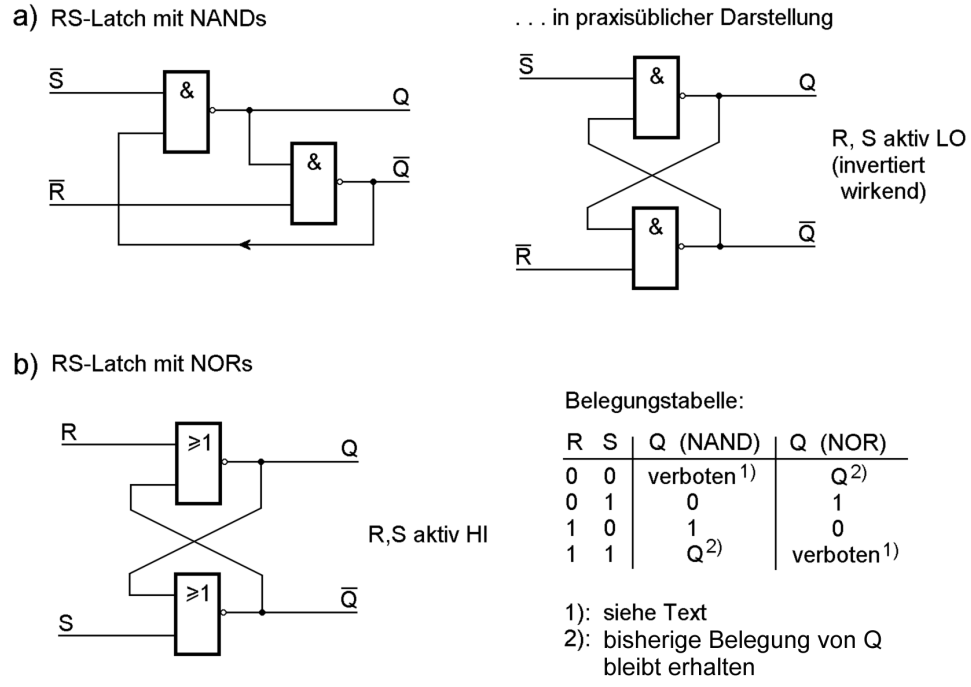


Abbildung 2.7 RS-Latch mit NAND- bzw. NOR-Gattern

Wollen wir eine Eins speichern, so müssen wir den S-Eingang aktivieren (Setzen), wollen wir eine Null speichern, den R-Eingang (Rücksetzen). Die Eingänge R und S sind hier jeweils gleichsinnig aktiv (NANDs: Low; NORs: High). Zudem können wir ausgangsseitig den gespeicherten Zustand sowohl in "wahrer" als auch in invertierter Form abnehmen.

Die Tatsache der Selbsthaltung wird meist durch eine besondere zeichnerische Darstellung mit überkreuzten Verbindungen kenntlich gemacht.

Problem: der "verbotene" Zustand
Was geschieht, wenn R und S beide aktiv sind? Diese Belegung wird üblicherweise als "verboten" bezeichnet. So haben wir es auch in die Belegungstabelle der Abbildung 2.7 eingetragen. Tatsächlich kann man diese Belegung aber anlegen. Sie führt dazu, daß *beide* Ausgänge gleiche Pegel annehmen (NANDs: High; NORs: Low). Das wird in manchen Schaltungen auch ausgenutzt*). Was wirklich "verboten" ist, ist das gleichzeitige *Ab*schalten (Deaktivieren) von R und S: dann nämlich ist der Zustand der Schaltung nicht mehr definiert; es ergibt sich eine zufällige Belegung.

*): beispielsweise in einfachen Vermittlungsschaltungen (Arbitration Latches).

Mehrere Setz- und Rücksetzeingänge
Die Gatter, die das RS-Latch bilden, können auf mehr als zwei Eingänge erweitert werden (Abbildung 2.8). Diese zusätzlichen R- und S-Eingänge wirken disjunktiv (ODER-Verknüpfung).

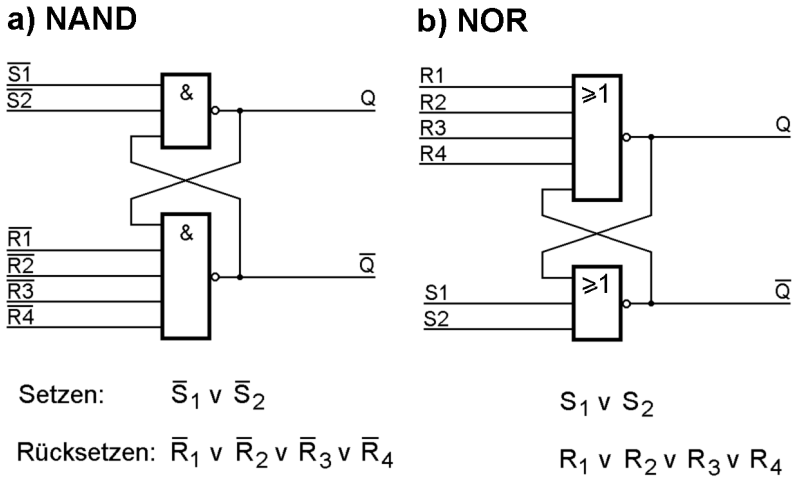


Abbildung 2.8 RS-Latches mit mehreren Setz- und Rücksetzeingängen

Taktsteuerung von RS-Latches
Soll das Schalten eines Latches nur bei Anliegen eines besonderen Signals (Taktsignals) erlaubt sein, so ist das Taktsignal*) mit den Eingängen R und S jeweils konjunktiv zu verknüpfen (Abbildung 2.9). Die Abbildung zeigt weiterhin, wie zusätzliche, unabhängig vom Takt wirkende Stelleingänge PRESET und CLEAR eingefügt werden können.

*): typische Bezeichnungen: C (Clock) oder G (Gate).

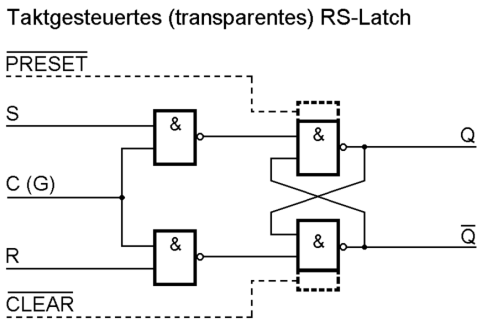


Abbildung 2.9 Taktgesteuertes (transparentes) RS-Latch

2.2.3. Latches als Datenspeicher (D-Latches)

Wenn wir ein Datenbit speichern wollen, brauchen wir einen Dateneingang (D) sowie eine Möglichkeit, die Übernahme des anliegenden Datenwertes zu erlauben bzw. zu sperren. Das Latch darf nur bei entsprechender Erlaubnis umschalten. Es ist bei anliegender Erlaubnis zu setzen, wenn eine Eins, und zu löschen, wenn eine Null zu speichern ist. Abbildung 2.10 zeigt verschiedene Ausführungsformen von D-Latches.

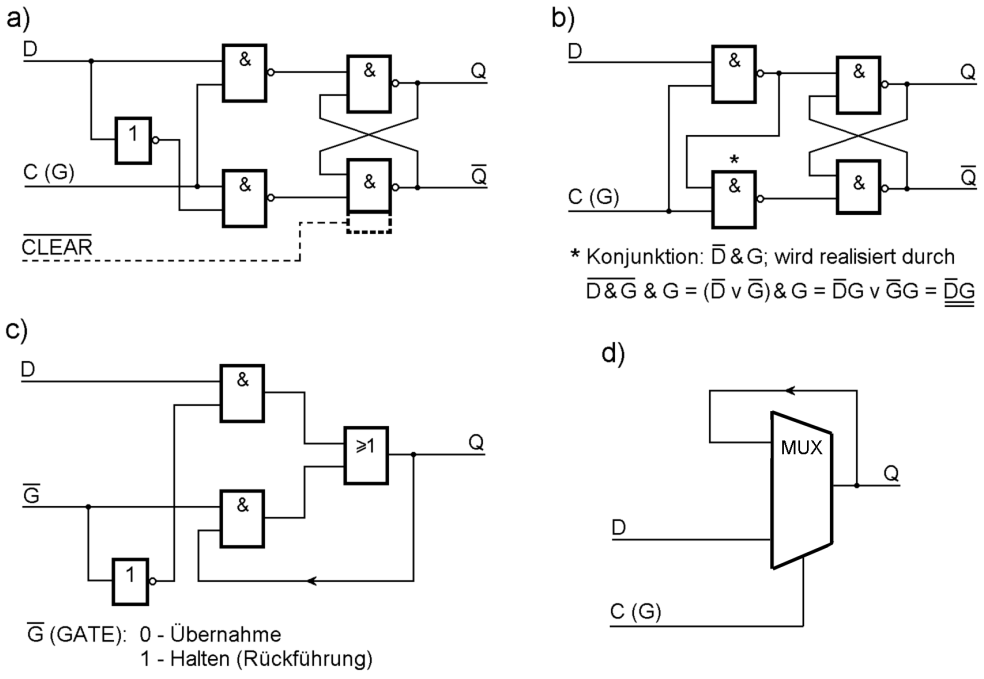


Abbildung 2.10 D-Latches

Erklärung:

a) durch Zusatzbeschaltung zum D-Latch erweitertes RS-Latch,

b) wie a), aber vereinfacht (Einsparung des Negators),

c) D-Latch auf Grundlage einer Auswahlschaltung. Übernahme: der Dateneingang wird ausgewählt; Halten des gespeicherten Bits: Rückführung wird ausgewählt.

d) ein 2-zu-1-Multiplexer als D-Latch.

Hinweise zu Abbildung 2.10c, d:

1. Wichtig ist, daß der Dateneingang noch zum Ausgang durchgesteuert bleibt, während die Rückführung ausgewählt wird (also beim Übergang vom Speichern zum Halten). Deshalb ist das Übernahmesteuersignal G so angeschlossen, wie hier gezeigt (die Verzögerungszeit des Negators wird ausgenutzt, um während des Umschaltens den Dateneingang weiterhin durchgesteuert zu halten).
2. Die UND-ODER-Strukturen in Abbildung 2.10c sind nichts anderes als Auswahlschaltungen, wie wir sie in jedem 2-zu-1-Multiplexer vorfinden. Wir merken uns: Latches kann man auf Grundlage von Multiplexern aufbauen. Wirkungsweise: das Taktsignal (C oder G) dient zur Datenwegauswahl:
   - Takt = 1: der Dateneingang (D) wird zum Ausgang durchgeschaltet; die Ausgangsbelegung folgt der Eingangsbelegung (Latch verhält sich wie Durchreiche),
   - Takt = 0: Ausgang ist auf sich selbst zurückgeführt; die jeweils letzte Belegung wird gespeichert (Selbsthaltung).

Anwendung: Latches und Flipflops in hochintegrierten Schaltkreisen. Sie sind oft aus rückgekoppelten Multiplexer-Strukturen aufgebaut. Die elementaren Funktionsblöcke (Zellen und Makrozellen) moderner programmierbarer Schaltkreise enthalten solche Strukturen, um damit Latches und Flipflops verwirklichen zu können (vgl. auch Kapitel 4).

2.2.4. Flipflops

Ein Flipflop ist ein taktgesteuerter 1-Bit-Speicher, der seine Ausgangsbelegung nur in Bezug auf eine jeweils spezifizierte Takt*flanke* ändert (d. h. auf die ansteigende bzw. abfallende Flanke des Taktimpulses). Manche Flipflops haben zusätzliche, unabhängig vom Takt wirkende Lösch- bzw. Einstelleingänge (Clear, Preset).

Latch und Flipflop
Abbildung 2.11 veranschaulicht, worin sich ein taktgesteuertes D-Latch und ein entsprechendes Flipflop (D-Flipflop) voneinander unterscheiden:

- Latch: der Dateneingang ist für die Dauer der Takt-Aktivierung zum Ausgang durchgeschaltet. Ist das Taktsignal aktiv, so folgt der Ausgang allen Änderungen des Eingangs nach (vgl. die Schaltungen in Abbildung 2.10). Man sagt, das Latch ist *transparent* für die Eingangssignale*).
- Flipflop: die Eingangsbelegung wird nur mit der jeweils spezifizierten Takt*flanke* übernommen.

*): in Datenblättern finden wir die Bezeichnung "transparent latches", und man bezeichnet das Taktsignal gelegentlich auch nicht als Takt (Clock), sondern (vgl. die Abbildungen 2.9 und 2.10) als Erlaubnis- bzw. Tor-Signal (Enable bzw. Gate).

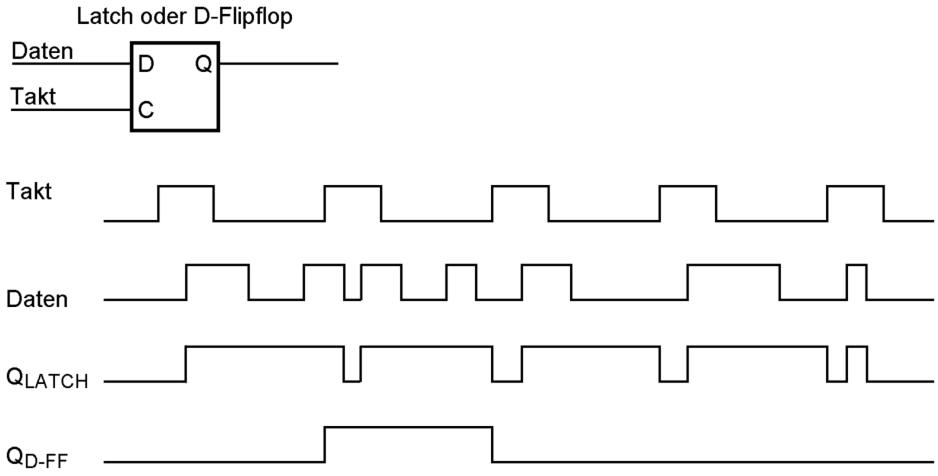


Abbildung 2.11 Schaltverhalten von transparentem Latch und echtem Flipflop im Vergleich

Latch und Flipflop in der Fachsprache
Es wird nicht immer streng unterschieden. Manche Theoretiker bezeichnen jeden 1-Bit-Speicher als Flipflop, unabhängig davon, wie die Taktsignale wirken. Hingegen finden wir in der praxisbezogenen Literatur (auch in Datenblättern) häufig die Bezeichnung "Latch" für jede Art von Halteregistern, auch wenn diese mit flankengesteuerten Flipflops bestückt sind.

Flipflop-Arten
Wieviele Flipflop-Arten gibt es? - Diese Frage läßt sich exakt beantworten: Das Verhalten eines Flipflops ist abhängig von der jeweiligen Belegung der Eingänge und von seinem aktuellen Zustand. Ein Flipflop hat als einfachstes binäres Speicherglied insgesamt nur zwei Zustände (0 oder 1). Wenn wir uns - wie beim Latch - mit höchstens zwei Eingängen begnügen (vgl. die RS-Latches), hängt der Folgezustand also von drei Binärvariablen ab.

Mit drei Variablen kann man $2^{2^3} = 256$ verschiedene Schaltfunktionen bilden. Es gibt also insgesamt 256 verschiedene Flipflops (darunter sind natürlich auch Lötstellen, Unterbrechungen, Festwerte sowie alle kombinatorischen Schaltfunktionen, die von einer und von zwei Variablen abhängen, und andere hier nicht besonders brauchbare Bauelemente). In der Praxis beschränkt man sich jedoch auf nur wenige Typen (Abbildung 2.12)

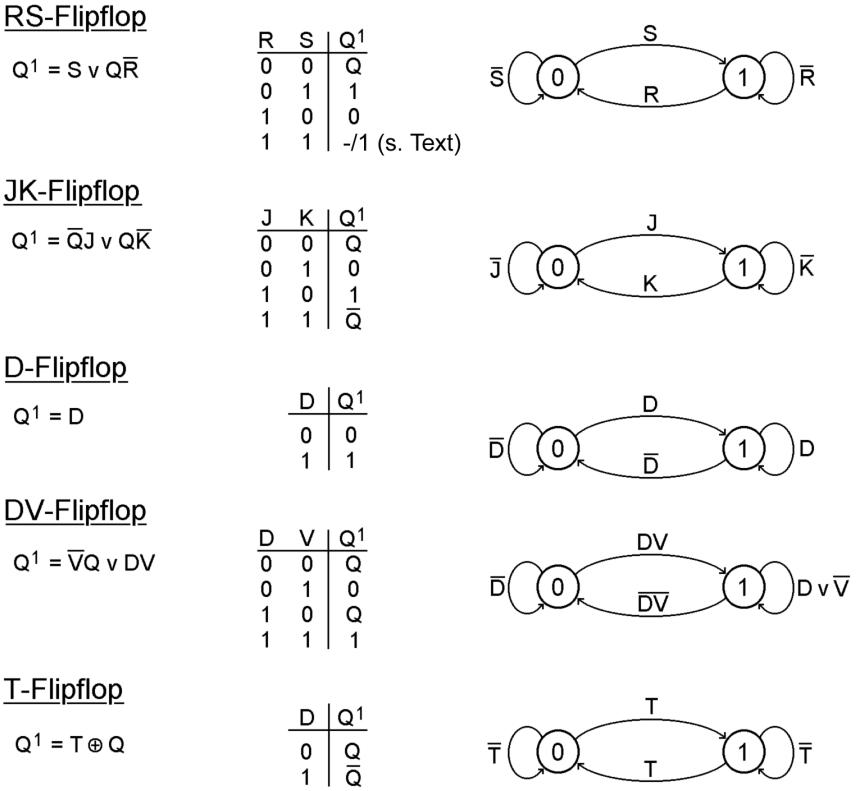


Abbildung 2.12 Wichtige Flipflop-Arten anhand der typischen Beschreibungsmittel: Automatengleichung, Automatentabelle und Zustandsgraph

RS-Flipflop
Das RS-Flipflop entspricht dem RS-Latch. *Zur Belegung 1, 1:* im allgemeinen Fall ist sie hier wirklich verboten (trifft eine schaltende Taktflanke auf diese Belegung, so verhält sich das Flipflop undefiniert). Beispiel: die Realisierung als Master-Slave-Flipflop (Abbildung 2.14a). Bei Realisierung über die Automatengleichung (= mittels D-Flipflop und Zusatzbeschaltung (Abbildung 2.13)) ergibt die Eingangsbelegung 1, 1 die Ausgangsbelegung 1.

JK- Flipflop
Es ist ein modifiziertes RS-Flipflop, bei dem der verbotene Zustand (J, K beide Eins) erlaubt ist und die Wirkung hat, die Ausgangsbelegung zu ändern (aus Eins wird Null und umgekehrt; Trigger- oder "Toggle"-Funktion).

D-Flipflop
Es entspricht dem D-Latch. Die Funktionsweise ist sehr einfach: jede schaltende Taktflanke übernimmt die Belegung des Eingangs D, die somit am Ausgang Q erscheint.

DV-Flipflop
Dieses Flipflop finden wir nicht als Einzelbauelement. Es ist ein D-Flipflop, erweitert um eine Auswahlschaltung (bzw. um einen Multiplexer) mit einem Erlaubniseingang V. Dessen Wirkung: nur bei aktivem V wird die Eingangsbelegung übernommen, ansonsten wird der bisherige Zustand gehalten.

T-Flipflop
Das T-Flipflop (Trigger- bzw. Toggle-Flipflop) wechselt mit jedem Takt seine Belegung. Anwendung: vor allem in Zählern und Frequenzteilern (Abschnitt 2.5.).

Wichtig ist: Alle rückkopplungsfähigen Flipflops lassen sich durch Zusatzbeschaltung in jede beliebige andere Art umwandeln. Abbildung 2.13 zeigt einige Beispiele.

Hinweise zu Abbildung 2.13:

1. Zu den Anschlußbezeichnungen: Eingänge heißen R, S, J, K, D usw., Ausgänge zumeist Q, Takteingänge C oder CL oder CLK. Manche Bezeichnungen (wie R, S, D, CLK) kann man sich erklären, manche nicht (wie J, K, Q). Sie haben sich aber seit Jahrzehnten eingebürgert. Nicht nachdenken! (Vgl. - auf ganz anderem Gebiet - die herrlich sinnvollen Kürzel der Unix-Kommandos ...)
2. Zum JK-Flipflop: die ersten integrierten Flipflops waren vom JK-Typ. Es ging seinerzeit darum, Universalbauelemente mit vielseitigen Einsatzmöglichkeiten anzubieten. Und in den 60er Jahren wurden die Flipflops vor allem in Zähl- und in Steuerschaltungen eingesetzt (u. a. beruhten die numerischen Werkzeugmaschinensteuerungen jener Zeit auf kunstvoll verschalteten Zähleranordnungen - echte Computer waren viel zu teuer). JK können wir also als eine Art Marketing-Idee ansehen, um beiden Hauptanwendungen jener Zeit gerecht zu werden (Triggerfunktion zum Zählen, RS-Funktion zu Steuerungszwecken).
3. Die DV-Rückführung ist die Grundschaltung der Nutzung von D-Flipflops in vollsynchronen Registern und Steuerschaltungen (State Machines) (Abschnitte 2.4.1. und 2.6.3.).

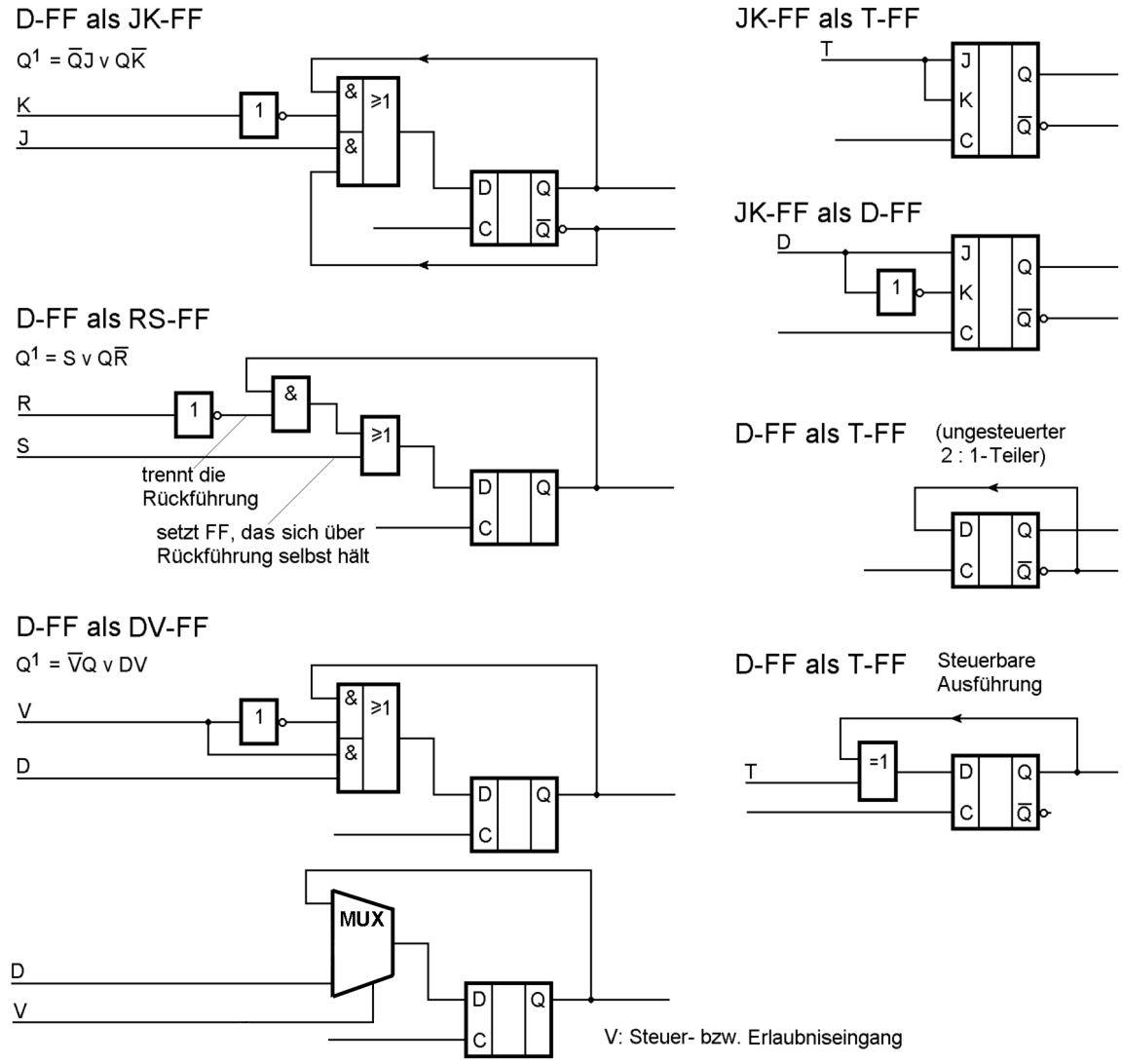


Abbildung 2.13 Wechselseitige Umwandlungen von Flipflop-Arten (Beispiele)

Master-Slave-Flipflops

Die Flankensteuerung erreicht man durch Hintereinanderschalten zweier taktgesteuerter Latches, wobei das zweite mit dem invertierten Takt beschaltet ist (Abbildung 2.14).

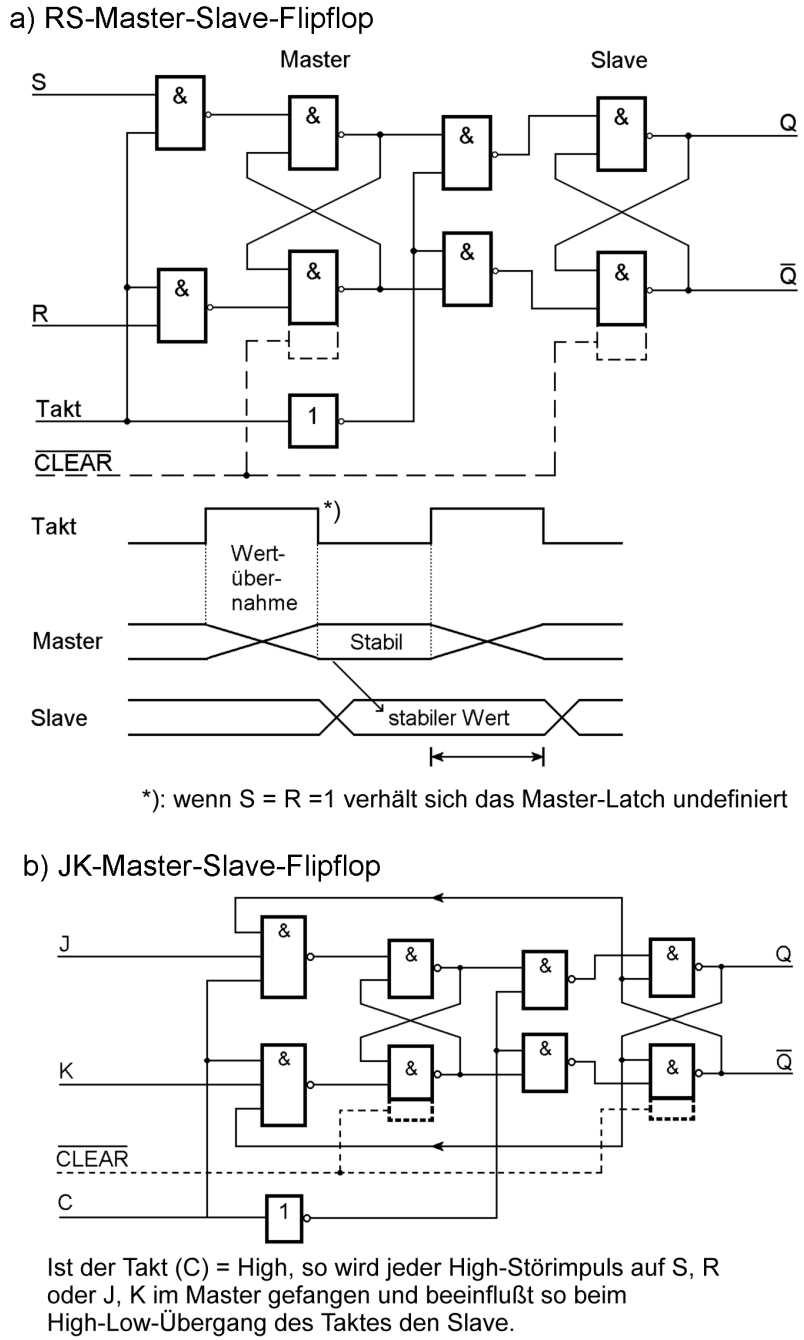


Abbildung 2.14 JK-Master-Slave-Flipflop

Erklärung:
Im Gegensatz zum Latch dürfen sich Änderungen an den Eingängen nicht unmittelbar an den Ausgängen bemerkbar machen; die Ausgangsbelegung soll sich allein mit der übernehmenden Taktflanke gemäß der dann jeweils aktuellen Eingangsbelegung ändern. Herkömmliche Master-Slave-Flipflops schalten mit der High-Low-Flanke des Taktes (Tabelle 2.1).

Latch	Takt = High	Takt = Low
Master	aktiv; Ausgänge folgen den Eingängen	gesperrt; die zur Taktflanke (High - Low) herrschende Belegung wird gehalten
Slave	gesperrt; die vorherige Belegung wird gehalten	aktiv. Damit wird die Belegung des Masters gleichsam zu den Ausgängen des Flipflops durchgereicht. Die Ausgangsbelegung ändert sich aber nicht, weil der Master gesperrt ist

Tabelle 2.1 Zur Funktionsweise des Master-Slave-Flipflops

Abbildung 2.14 weist auf eine charakteristische Störungsempfindlichkeit solcher Master-Slave-Flipflops hin: bei aktivem Takt wird jeder High-Störimpuls im Master gefangen und so mit dem Abschalten des Taktes in den Slave übernommen. In den Datenblättern ist deshalb die Übernahme der Eingangsbelegung mit der Takt-Vorderflanke (Low-High) spezifiziert, während die Ausgänge mit der Rückflanke (High-Low) schalten.

Data Lockout
Das Störproblem läßt sich vermeiden, wenn man den Master nur bis kurz nach der Vorderflanke des Taktes offenhält. Die Information erscheint aber trotzdem erst nach der Rückflanke an den Ausgängen. Es gibt Master-Slave-Flipflops mit und ohne Data Lockout.

Das JK-Master-Slave-Flipflop
Das "klassische" JK-Flipflop (vgl. den Hinweis 2 auf Seite 73) ist ein aus dem RS-Typ abgewandeltes Master-Slave-Flipflop, wobei die über Kreuz auf die Eingänge zurückgeführten Ausgänge das gewünschte Verhalten bei J = K = 1 gewährleisten (Abbildung 2.14b).

Flankengesteuerte Flipflops
Moderne Flipflops sind nahezu ausnahmslos flankengesteuert (Edge Triggered). Es gibt also wirklich nur eine schaltende Signalflanke; bei D-Typen Low-High (Positive Edge), bei manchen JK-Typen High-Low (Negative Edge*)). Die Flankensteuerung läßt sich mit trickreichen Verknüpfungen von Latches verwirklichen (Abbildung 2.15).

*): zweckmäßig beim Einsatz in asynchronen Binärzählern (Abschnitt 2.5.3.).

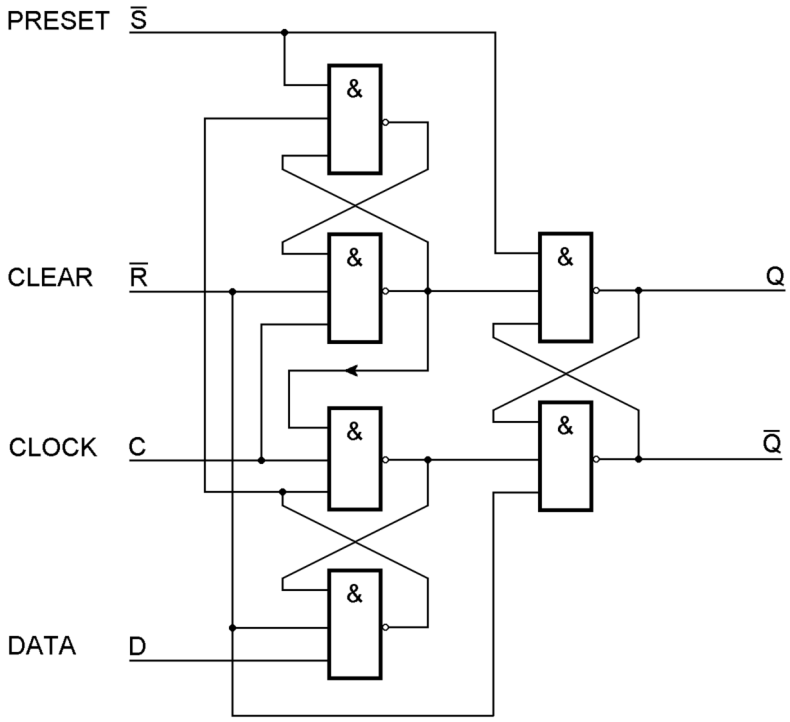


Abbildung 2.15 Flankengetriggertes D-Flipflop (Typ 7474; Texas Instruments)

Flankengesteuerte Flipflops in Master-Slave-Schaltung

Wir bauen die Master-Slave-Anordnung als D-Flipflop auf (Abbildung 2.16) und realisieren ein ggf. erforderliches anderes Schaltverhalten durch entsprechende Zusatzbeschaltung (vgl. Abbildung 2.13)*). Der Grundgedanke: liegt der Takt am Master auf High, so folgt das Master-Latch zwangsläufig dem D-Eingang. Somit können Störimpulse auch nicht im Master gefangen werden (ein solcher Impuls würde das Master-Latch kurzzeitig setzen und dann wieder zurücksetzen). Anwendungsbeispiele: (1) manche Schaltkreise in modernen Logikbaureihen (Abbildung 2.17), (2) Realisierung von Flipflops in den Zellen mancher FPGA-Familien (Kapitel 4).

*): damit das Flipflop - wie allgemein üblich - mit der Low-High-Taktflanke schaltet, müssen wir die Taktbeschaltung gegenüber dem herkömmlichen Master-Slave-Flipflop (Abbildung 2.14) invertieren.

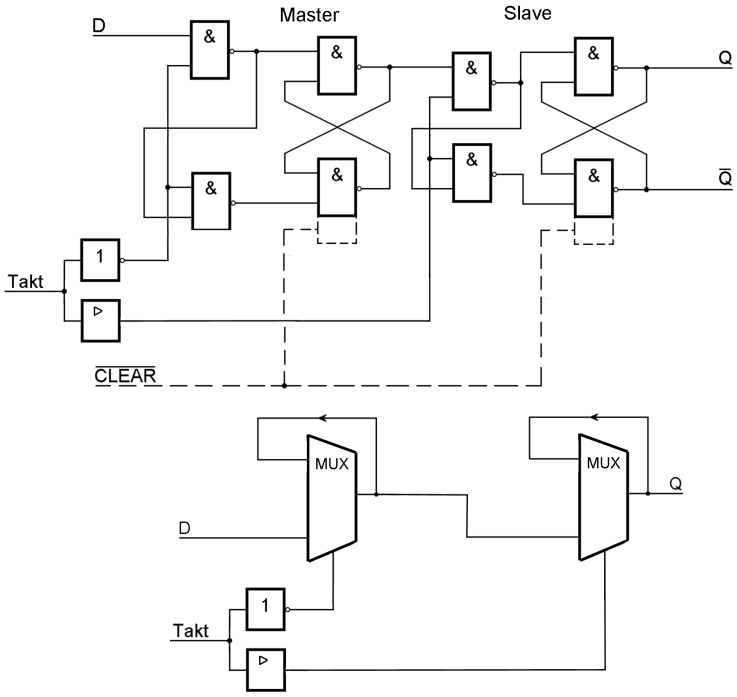


Abbildung 2.16 Flankengesteuertes D-Flipflop. Zwei D-Latches (vgl. Abbildung 2.10b, d) in Master-Slave-Schaltung

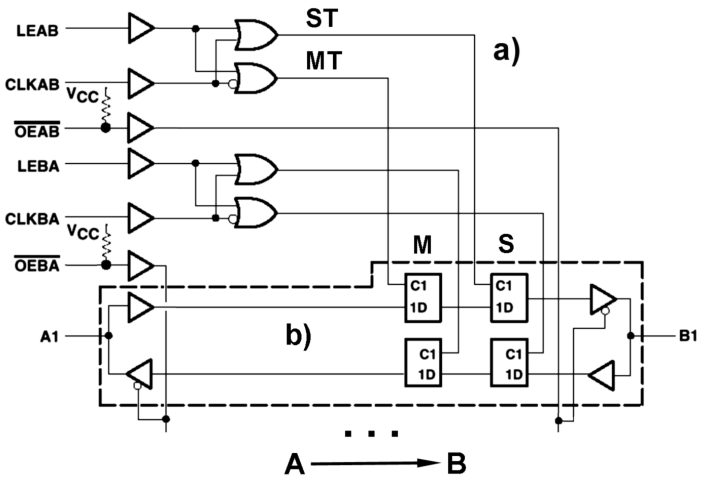


Abbildung 2.17 Eine Trickkiste: Flipflops in einem modernen Buskoppelschaltkreis (Texas Instruments)

Erklärung:
Solche universellen Buskoppelschaltkreise sind u. a. auf nahezu allen modernen Speichermoduln (SDRAM, DDR-DRAM) zu finden. Der Grundgedanke: ein Schaltkreis - vielfältige Einsatzmöglichkeiten (als Treiber ohne Speicherfunktion, als Latchregister, als D-Flipflop-Register). a) - zentrale Steuerung; b) - eine Bitposition (von beispielsweise 9). Die Bitpositionen sind für beide Übertragungsrichtungen ausgelegt. Uns interessiert hier nur, wie die Universalität durch trickreiche Ausnutzung der Eigenschaften von D-Latches zustande kommt. Das erklären wir anhand der Übertragungsrichtung von A nach B (Pfeil). M - Master; S - Slave; MT - Master-Takt; ST - Slave-Takt.

Das Steuersignal LEAB entscheidet über die Funktionsweise:

- LEAB = 0: ST und MT hängen allein vom Takteingang CLKAB ab. Das Master-Latch wird aktiviert bei CLKAB = 0, das Slave-Latch bei CLKAB = 1. Die Anordnung verhält sich also wie das aus D-Latches gebildete D-Flipflop von Abbildung 2.16.
- LEAB = 1: Über die ODER-Gatter werden sowohl ST als auch MT aktiv; CLKAB spielt dabei gar keine Rolle. Beide Latches hintereinander wirken somit als Durchreiche (Ausgang folgt Eingang).
- Betrieb als Latchregister: durch Schalten von LEAB bei Festbelegung von CLKAB:
  - CLKAB = 0: S wirkt als Latch, M ist Durchreiche (ST schaltet mit LEAB, MT wird von CLKAB fest auf 1 gehalten),
  - CLKAB = 1: M wirkt als Latch, S ist Durchreiche (MT schaltet mit LEAB, ST wird von CLKAB fest auf 1 gehalten).

Rückkopplungsfähigkeit
Im Gegensatz zu Latches sind Master-Slave- und flankengesteuerte Flipflops *rückkopplungsfähig*, das heißt, die Ausgänge dürfen direkt oder über kombinatorische Schaltungen auf die der Taktsteuerung unterliegenden Eingänge (z. B. auf J, K oder D) zurückgeführt werden. (Eingänge, die der Taktsteuerung unterliegen, werden als *synchrone* Eingänge bezeichnet.) "Setzen" und "Rücksetzen" (Preset/Clear) sind meist *keine* synchronen Eingänge!

Vorhalte- und Haltezeiten (Setup/Hold Timing)
Die Eingänge müssen - bezogen auf die jeweilige Taktflanke - mindestens um die Setup-Zeit voreilend stabil anliegen. Die Belegung muß mindestens für die Dauer der Haltezeit nach der jeweiligen Flanke stabil gehalten werden. Datenblätter beachten! (Siehe weiterhin Kapitel 5.)

2.2.5. Synchronisation, Eintaktierung, Metastabilität

Was geschieht, wenn sich Eingangsbelegungen eines Flipflops im Setup-Hold-Intervall einer schaltenden Taktflanke ändern? - Das hängt ganz vom Zufall ab. Manchmal wird nichts Besonderes geschehen; das Flipflop wird entweder seinen Zustand ändern oder den bisherigen beibehalten. Es gibt aber innerhalb des Setup-Hold-Intervalls ein sog. kritisches Zeitfenster (10...150 ps; abhängig von der jeweiligen Schaltkreis-Baureihe). Fällt die Eingangs-Änderung in dieses Zeitfenster, so kann das Flipflop in einen instabilen (*metastabilen*) Zustand gelangen. Seine Ausgänge verhalten sich dann undefiniert (sie nehmen keine eindeutigen Logikpegel an; gelegentlich kanne es auch zu wilden Schwingungen kommen).

Diese Tatsache hat eine große praktische Bedeutung. Trotz aller Mühe läßt sich nämlich nie vollkommene Synchronität gewährleisten. Auch wenn die eigentliche Schaltung vollsynchron arbeitet, sind die Schnittstellen zur Außenwelt naturgemäß immer asynchron (Schalter, Tasten, Sensoren, Interfaces anderer Funktionseinheiten usw. wissen nun einmal nichts vom internen Takt unserer Hardware). An diesen Schnittstellen ist also stets das Problem der *Synchronisation* (Eintaktierung) zu lösen. Die naheliegende Lösung: das asynchrone Signal auf den Dateneingang eines D-Flipflops legen, der mit dem jeweils passenden Takt angesteuert wird. Und genau hier ist mit metastabilen Zuständen zu rechnen (Abbildung 2.18).

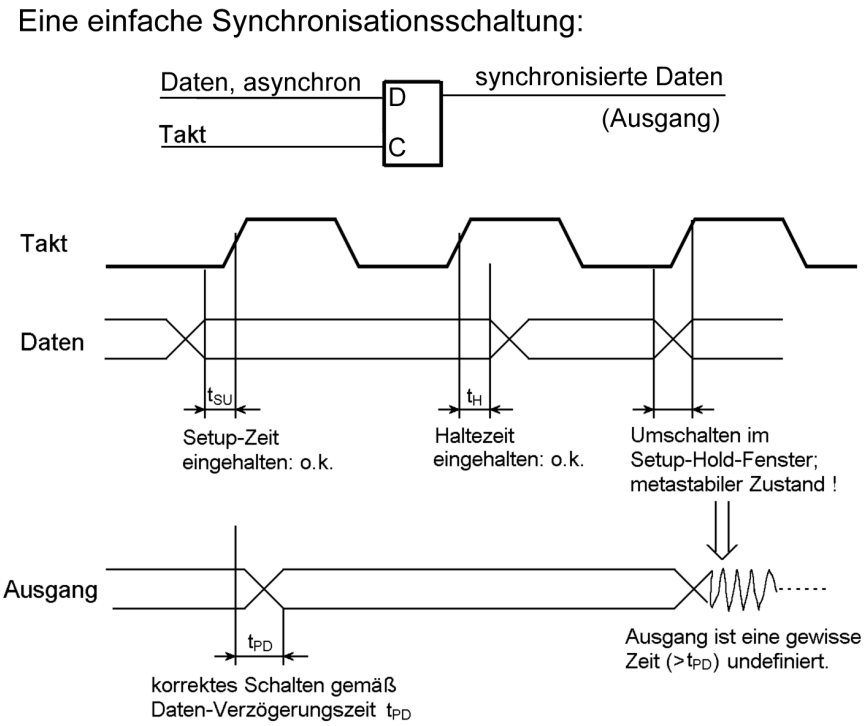


Abbildung 2.18 Das metastabile Verhalten eines Flipflops

Überlegen wir einmal, wie oft ein solcher metastabiler Zustand eintreten wird. Das Flipflop werde mit der Taktfrequenz f_{CLK} betrieben. Das einzutaktierende Signal ändere sich mit der Frequenz f_{IN} (grundsätzlich muß gelten $f_{IN} < f_{CLK}$; sonst würde die Eintaktierung gar nicht funktionieren). Die Dauer des kritischen Zeitfensters sei t_W. Nun muß, um einen metastabilen Zustand hervorzubringen, eine Datenflanke im kritischen Zeitfenster der Taktflanke auftreten. Das mittlere Zeitintervall zwischen zwei solchen Ereignissen (die MTBF; Mean Time between Failures) ergibt sich - wir verzichten auf die Herleitung - zu:

$$MTBF = \frac{1}{f_{\in} \cdot f_{CLK} \cdot t_W}$$

Im Beispiel: f_{CLK} = 2 MHz, f_{IN} = 10 kHz, t_W = 50 ps. Damit ergibt sich:

$$MTBF = \frac{1}{2\ MHz \cdot 10\ kHz \cdot 50\ ps} = 1\ s$$

Es ist also etwa alle Sekunde mit einer Fehlfunktion zu rechnen.

Wie läßt sich Abhilfe schaffen?
Grundsätzlich: Es gibt kein 100%ig wirkendes Gegenmittel. Man gibt sich vielmehr zufrieden, wenn man - rechnerisch und meßtechnisch nachgewiesen - einen so hohen MTBF-Wert erhält, daß Fehlfunktionen und Ausfälle aus anderen Ursachen viel wahrscheinlicher sind (so ist eine MTBF von beispielsweise 10^4 Jahren ein solcher ausreichender Wert). Dabei kann es nur darum gehen, den metastabilen Zustand selbst oder dessen Auswirkungen zu verhindern, nicht aber, ein definiertes Schaltverhalten des Flipflops zu erzwingen (wenn Takt- und Datenflanke zusammentreffen, ist es nach wie vor Glückssache, ob das Flipflop schaltet oder nicht). Die Gegenmaßnahmen im einzelnen:

- Auswahl von Flipflops, die von Natur aus nur selten in metastabile Zustände übergehen. Wichtig hierfür ist ein möglichst kleines kritisches Zeitfenster t_W.
- Ausgänge solcher Synchronisations- bzw. Eintaktierungs-Flipflops dürfen nach der schaltenden Taktflanke erst dann ausgewertet werden, wenn mögliche metastabile Zustände abgeklungen sind. Je länger die Wartezeit, um so höher die Funktionssicherheit (bzw. MTBF).
- durch Hintereinanderschalten von zwei Flipflops wird die Wartezeit gleichsam automatisch gewährleistet (Doppelsynchronisation; Abbildung 2.19). Das Signal wird dadurch um eine weitere Taktperiode verzögert.

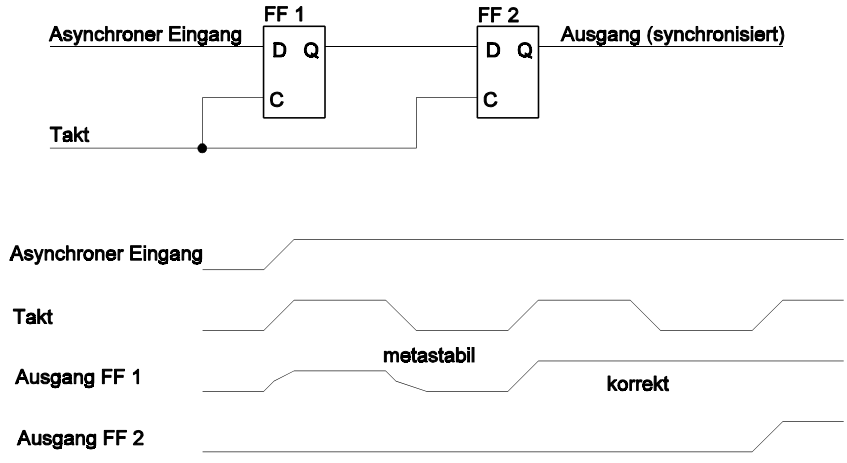


Abbildung 2.19 Doppelsynchronisation (nach: Texas Instruments)

Abbildung 2.20 veranschaulicht das metastabile Verhalten verbreiteter Logikfamilien. In der Waagerechten ist das Zeitintervall t_x aufgetragen, um das die Auswertezeit verlängert wird. Die Senkrechte enthält die verbleibende Unsicherheit (ausgedrückt als MTBF).

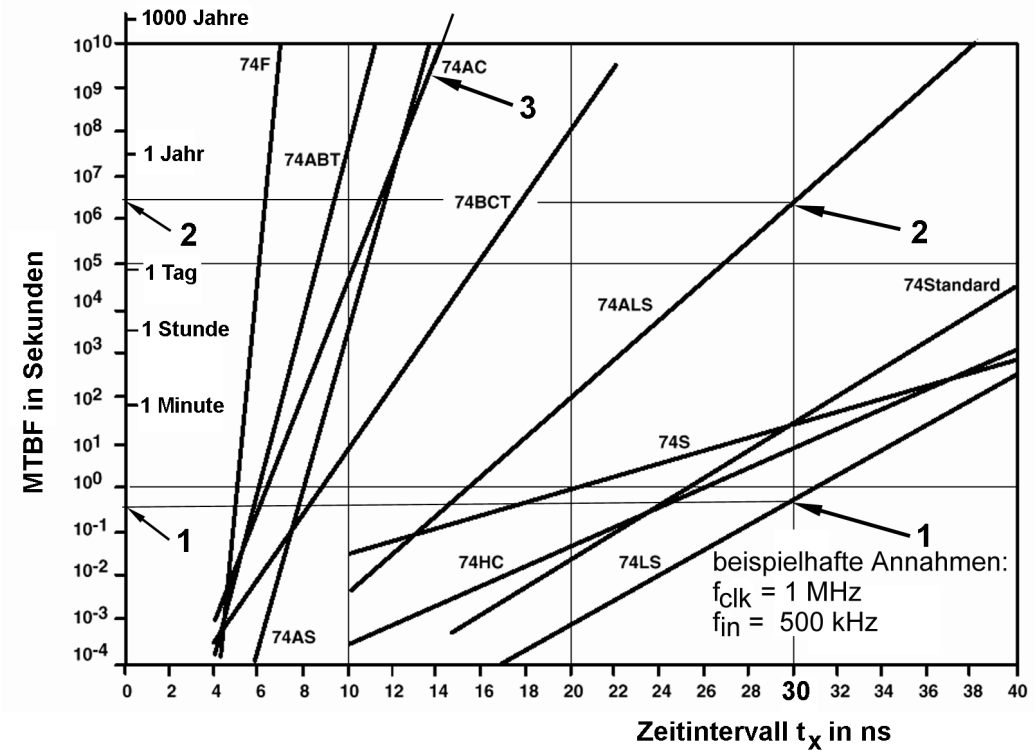


Abbildung 2.20 Das metastabile Verhalten verbreiteter Logikfamilien (Texas Instruments)

Ablesebeispiele:

1. Schaltkreis der Baureihe 74LS. 30 ns nach der übernehmenden Taktflanke werten wir das Signal aus. Die MTBF: zwischen 0,1 und 1 s. Das ist natürlich viel zuwenig - sogar im Vergleich mit der Zeit zwischen zwei Abstürzen eines typischen PCs ...
2. wir bleiben bei den 30 ns, nehmen aber einen Schaltkreis der Baureihe 74ALS. Die MTBF: 10^7 = 10 Millionen s. Ein Jahr hat 365 · 24 · 3600 = 31 536 000 s. Also rund 4 Monate (⅓ Jahr). Auch damit kann man sich nicht auf dem Markt blicken lassen (an Qualitätsvorschriften und Garantiefristen denken ...).
3. nun nehmen wir einen Schaltkreis der Baureihe 74AC. Bei 30 ns zeigt unsere Abbildung gar nichts mehr an (rechnerisch kommen ca. 18 · 10^{20} Jahre heraus...). Wenn wir uns mit einer MTBF von 1000 Jahren zufriedengeben, können wir den zeitlichen Abstand zwischen Taktflanke und Auswertung auf etwa 15 ns verkürzen.

Wir merken uns:

- wenn in Datenblättern, z. B. von Mikroprozessoren, Eingänge als synchron spezifiziert sind, dann muß dafür gesorgt werden, daß die Signale die Setup- und Haltezeitforderungen erfüllen, andernfalls funktioniert es nicht (dies äußert sich zumeist als sporadische Fehlfunktion). In einer gut entworfenen und funktionsfähigen Schaltung ist kaum mit Fehlern infolge metastabiler Zustände zu rechnen (die MTBF beträgt etliche Jahre). Ohne entsprechende Maßnahmen kann die MTBF hingegen im Bereich von Minuten bis Sekunden liegen. Bei wiederholten aufeinanderfolgenden Funktionsstörungen ist also die Synchronisations-Hardware entsprechend mit zu verdächtigen.

- sind Eingänge ausdrücklich als asynchron spezifiziert, so wird bei Nichteinhaltung der Anforderungen die Funktion nicht beeinträchtigt (Beispiel: Annahme von Interrupts). Erklärung: Einem solchen Eingang sind im Schaltkreis entsprechende Synchronisierschaltungen nachgeordnet.
- ein Signal, das synchronisiert werden soll, muß wenigstens so lange aktiv sein, daß es von einem Taktimpuls garantiert wenigstens einmal erfaßt wird. Faustregel: Aktivierung > 1 Taktperiode; Taktfrequenz wenigstens das Doppelte der Impulsfolgefrequenz des Eingangssignals ($f_{CLK} \leq 2\ f_{IN}$). Zu kurze Eingangsimpulse müssen passend verlängert werden (einige Schaltungen dazu zeigen wir im nächsten Abschnitt).

2.3. Elementarschaltungen mit Latches und Flipflops

Natürlich kann man Latches und Flipflops sozusagen wild verschalten, um bestimmte Funktionen zu verwirklichen. Es gibt aber bewährte Schaltungsprinzipien, die immer wieder eingesetzt werden. Die folgenden Beispiele sollen Ihnen dabei helfen, solche Schaltungen in komplexen Strukturen wiederzuerkennen*). Zudem bieten sie eine gute Gelegenheit, das Hineindenken in Zusammenhänge und Funktionsprinzipien zu üben.

*): gelegentlich stößt man mitten in der Beschreibung eines ganz modernen Schaltkreises auf so eine Spitzfindigkeit (Beispiel: das Arbitration Latch im Dual-Port-RAM).

2.3.1. Fangschaltungen

Die Aufgabe einer Fangschaltung: einen zu beliebiger Zeit auftretenden (kurzen) Impuls zu registrieren und solange aktiv zu bleiben, bis die Erregung weiterverarbeitet worden ist (dann wird über ein Rückstellsignal die Schaltung gleichsam wieder scharfgemacht).

Solche Schaltungen sind beispielsweise wichtig, um ankommende Impulse unabhängig von einem Takt bis zur Verarbeitung (auch: bis zur programmseitigen Abfrage) aufzufangen und zwischenzuspeichern.

Zum Fangen von Impulsen eignen sich sowohl RS-Latches als auch flankengesteuerte Flipflops (Abbildung 2.21).

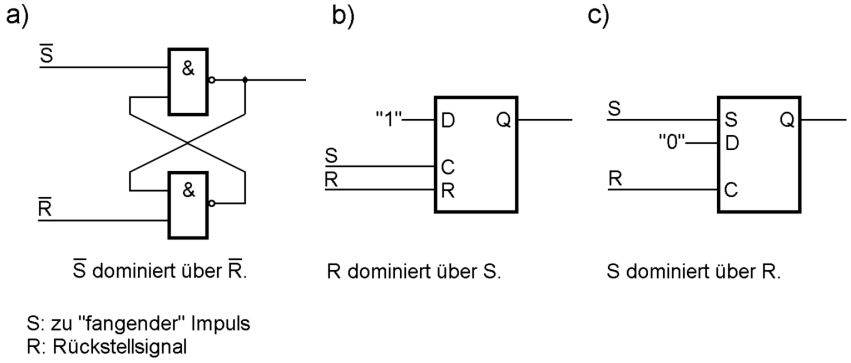


Abbildung 2.21 Fangschaltungen

Erklärung zu Abbildung 2.21:

a) Fangen mit RS-Latch,

b) Fangen der Low-High-Flanke mit D-Flipflop. Rückstellen über Löscheingang. Das Rückstellsignal dominiert über einen währenddessen ankommenden neuen Impuls (ein solcher Impuls würde verlorengehen).

c) Fangen des Impulses über den Setzeingang eines D-Flipflops. Rückstellen über Takteingang. Der zu fangende Impuls dominiert über das Rückstellsignal (bei Zusammentreffen von Impuls und Low-High-Flanke des Rückstellsignals bleibt das Flipflop gesetzt).

Hinweis:
Auch die als "asynchron" bezeichneten Setz- und Rücksetzeingänge von Flipflops (Preset/Clear) haben kritische Zeitfenster in Bezug auf den Takt. Bei deren Erregung sind also metastabile Zustände nicht grundsätzlich auszuschließen. Trickschaltungen (z. B. Abbildung 2.21 b, c), die diese Eingänge für funktionelle Zwecke ausnutzen, sind deshalb nicht völlig unbedenklich. In der Praxis funktionieren sie aber üblicherweise (und haben sich seit Jahrzehnten bewährt...).

2.3.2. Eintaktierungsschaltungen

Eintaktierung kurzer Impulse
Ein Eingangsimpuls, der zu kurz ist, kann von einem üblichen Synchronisations-Flipflop nicht immer erfaßt werden. Abbildung 2.22 zeigt zwei Kombinationen aus Fang- und Synchronisationsflipflops, die Abhilfe schaffen.

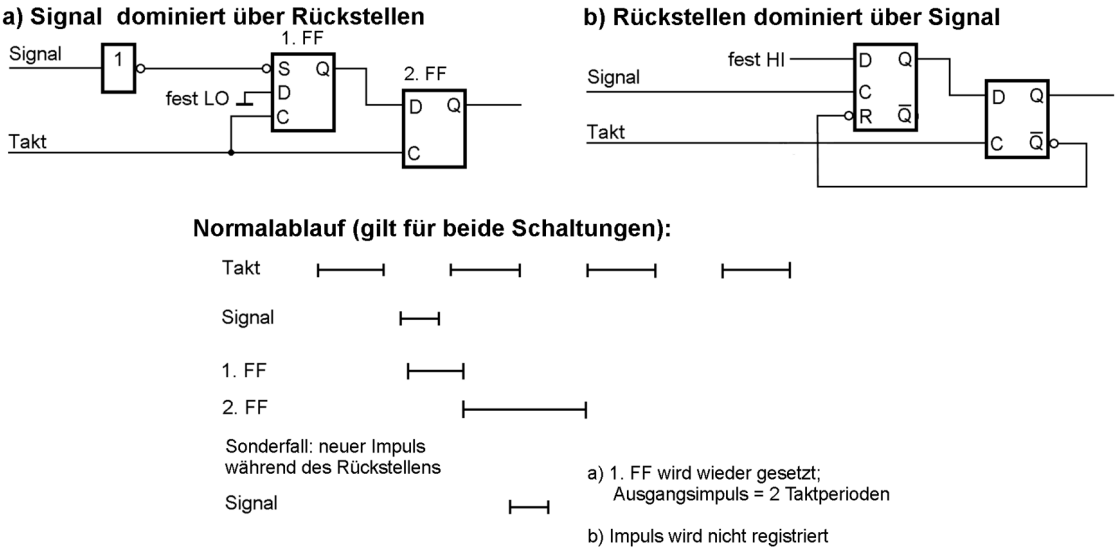


Abbildung 2.22 Eintaktierung kurzer Impulse (1). Grundschaltungen

Erklärung:
1. FF - Fangflipflop (vgl. Abbildung 2.21 b); 2. FF - Eintaktierungsflipflop. Der ankommende Impuls wird zunächst gefangen und im Fangflipflop bis zum nächsten Takt gehalten. Beide

Schaltungen unterscheiden sich darin, welcher Flipflopeingang zum Fangen und welcher zum Rückstellen verwendet wird. Tabelle 2.2 nennt die Besonderheiten. Abbildung 2.23 zeigt eine Trickschaltung, die (1) jeden Impuls fängt und (2) je gefangenem Impuls den Ausgang für die Dauer einer Taktperiode erregt.

Signalverlauf	Abbildung 2.22a	Abbildung 2.22b
während des Rückstellens tritt ein neuer Impuls am Eingang auf	Ausgangsimpuls verlängert sich um eine Taktperiode	der neue Impuls wird gar nicht registriert
der Eingangsimpuls ist länger als eine Taktperiode	Ausgangsimpuls wird ebenfalls entsprechend länger*)	Ausgangsimpuls ist stets eine Taktperiode lang

*): man kann somit nicht unterscheiden, ob mehrere Impulse eingetroffen sind oder ob ein einziger langer Impuls am Eingang anliegt

Tabelle 2.2 Zum Verhalten der Grundschaltungen

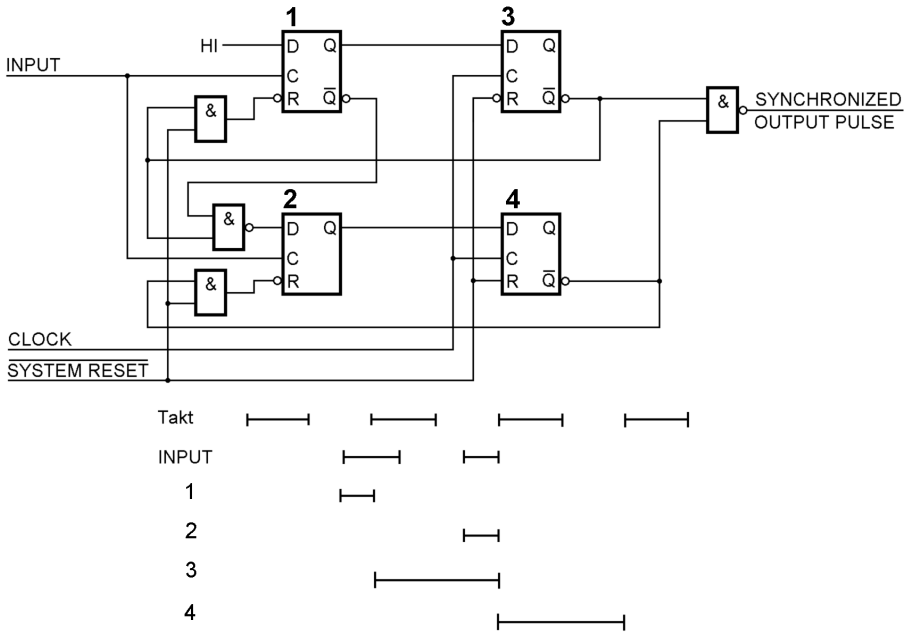


Abbildung 2.23 Eintaktierung kurzer Impulse (2). Beispiel einer Trickschaltung

Erklärung:
1, 2 - Fangflipflops; 3 , 4 - Eintaktierungsflipflops. Die Fangflipflops wirken gemäß Abbildung 2.22b. Ist Fangflipflop 1 gesetzt, so wird ein zwischenzeitlich eintreffender Impuls in Fangflipflop 2 gefangen.

Hinweis:
Die Abbildung ist vor allem als Übungsbeispiel zum Hineindenken in Funktionszusammenhänge gedacht. Achten Sie darauf, wie hier UNDs und NANDs als ODER-Verknüpfungen eingesetzt werden (vgl. Abbildung 1.10).

Ausblenden kurzer Störimpulse (Debouncing, Deglitching)
Mit der Schaltung nach Abbildung 2.24 wird ein Impuls nur dann als gültig erkannt, wenn er während wenigstens einer Taktperiode stabil anliegt. Alle kürzeren Impulse führen nicht zu einem aktiven Ausgangssignal.

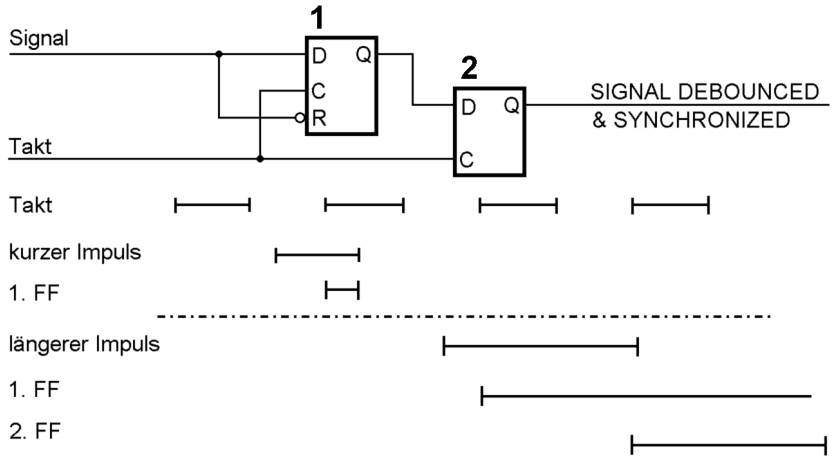


Abbildung 2.24 Ausblenden kurzer Störimpulse (1)

Erklärung:
Trifft ein kurzer Impuls gar nicht mit einer Low-High-Taktflanke zusammen, so passiert überhaupt nichts. Treffen beide zusammen, wird zunächst Flipflop 1 gesetzt. Wird nun der Impuls wieder inaktiv, so wird Flipflop 1 sofort zurückgesetzt. Ist der Impuls kürzer als eine Taktperiode, wird Flipflop 1 noch vor der folgenden Low-High-Taktflanke zurückgesetzt, so daß Flipflop 2 nicht einschaltet.

Die teils asynchrone Wirkungsweise (über den Rücksetzeingang des Flipflops 1) wird nicht immer gern gesehen (vgl. den Hinweis auf Seite 85). Abbildung 2.25 zeigt eine vollsynchrone Lösung.

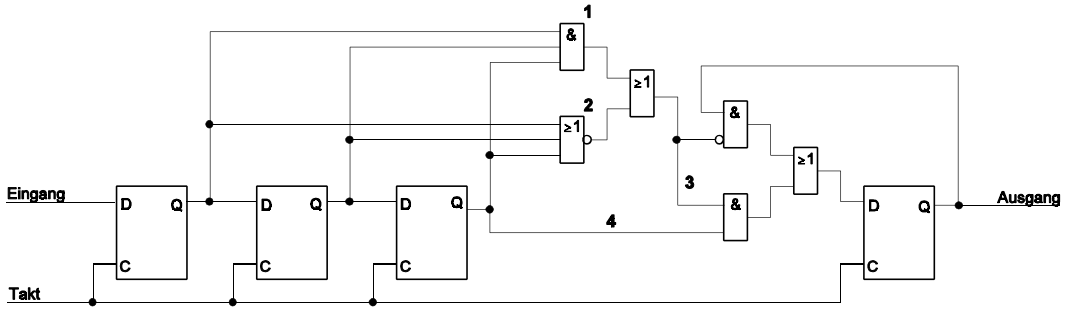


Abbildung 2.25 Ausblenden kurzer Störimpulse (2). Vollsynchrone Wirkungsweise (nach: Xilinx)

Erklärung:
Es handelt sich um eine Art Schieberegister (Abschnitt 2.4.2.). Mit jeder Low-High-Taktflanke wird eine Signalbelegung übernommen.

1) Schalten des Ausgangs auf Eins. Es müssen in wenigstens 3 aufeinanderfolgenden Taktperioden Einsen übernommen worden sein (Erkennung durch UND-Verknüpfung).
2) Schalten des Ausgangs auf Null. Es müssen in wenigstens 3 aufeinanderfolgenden Taktperioden Nullen übernommen worden sein (Erkennung durch NOR-Verknüpfung).
3) Umschalterlaubnis für Ausgang-Flipflop. Wird nur dann aktiv, wenn in wenigstens 3 aufeinanderfolgenden Taktperioden die gleiche Belegung abgetastet wurde (also 3 aufeinanderfolgende Nullen oder Einsen).
4) Übernahme der zeitverschobenen Eingangsbelegung bei aktiver Umschalterlaubnis.

Diskussion
Betrachten wir einen etwas spitzfindigen Signalverlauf (Abbildung 2.26). Unsere beiden Ausblendschaltungen werden hierauf unterschiedlich reagieren:

- gemäß Abbildung 2.24 erscheint kein ausgangsseitiger Impuls,
- gemäß Abbildung 2.25 wird ein Impuls abgegeben.

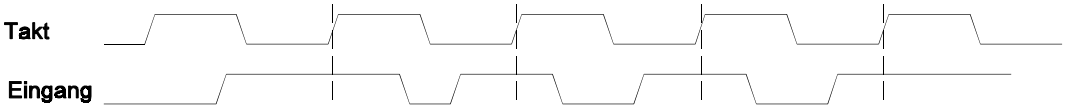


Abbildung 2.26 Ein Signalverlauf

Hinweis:
Das Synchronisieren durch Übernehmen in ein Flipflop ist ein Abtastvorgang (das Signal wird mit der betreffenden Taktflanke abgetastet). Wir erkennen eine typische Eigenheit des Abtastprinzips: was zwischen den Abtastzeitpunkten geschieht, bekommen wir einfach nicht mit. Ist also ein Verhalten gemäß Abbildung 2.24 gewünscht (auch die kürzeste Störung soll bewirken, daß kein Ausgangsimpuls erscheint), so kann man das mit einer vollsynchronen Auslegung nicht gewährleisten.

2.3.3. Flankenerkennung

Wenn man ein Signal verzögert und das verzögerte mit dem ursprünglichen Signal kombinatorisch verknüpft, lassen sich die Signalflanken erkennen (die jeweilige Flanke führt zu einem Impuls von der Dauer der Verzögerungszeit). Man kann wahlweise mit Flipflops (bzw. Schieberegistern) oder mit speziellen Baustufen verzögern (Abbildung 2.27).

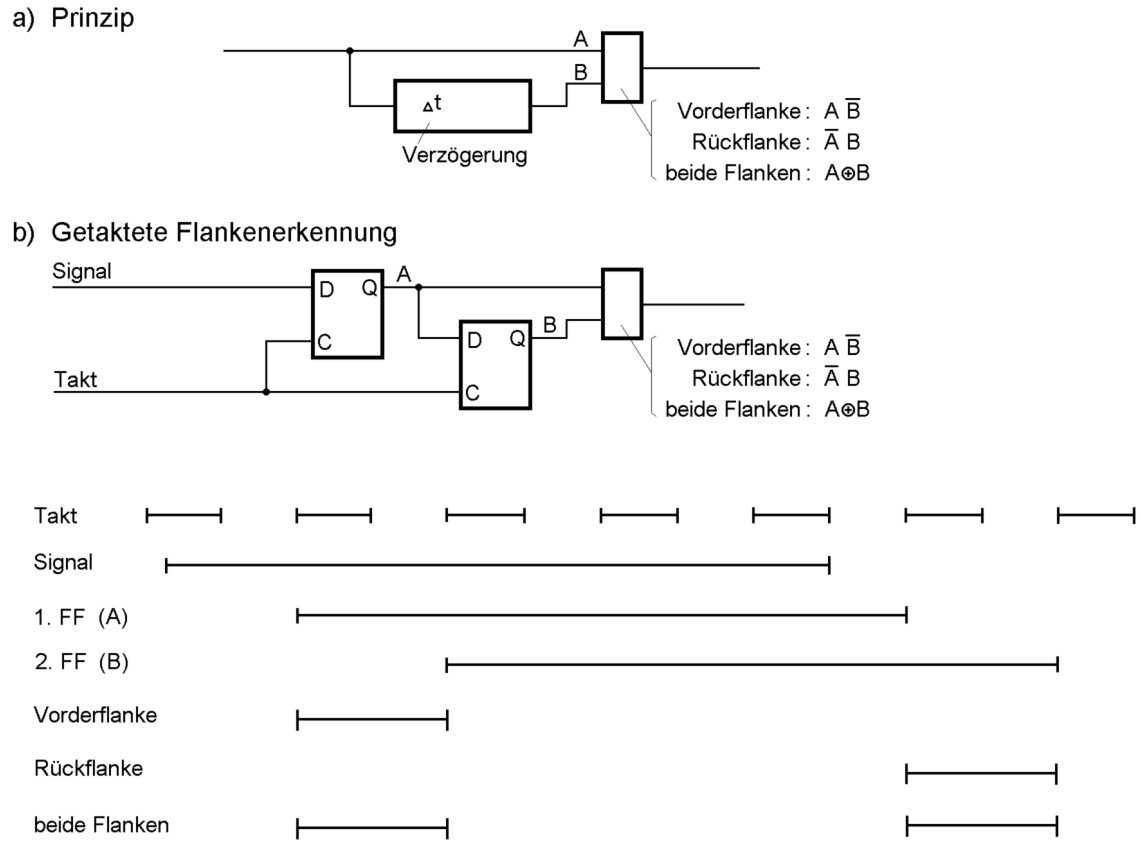


Abbildung 2.27 Flankenerkennung

2.3.4. Einzelimpulserzeugung (Single Shot)

Die Aufgabe: auf Grund eines auslösenden Signals wird ein einzelner Impuls (Single Shot) aus dem durchlaufenden Takt ausgeblendet. Das auslösende Signal muß erst wieder inaktiv werden, ehe durch erneute Aktivierung wieder ein Ausgangsimpuls abgegeben werden kann. Abbildung 2.28 zeigt eine einfache Schaltung.

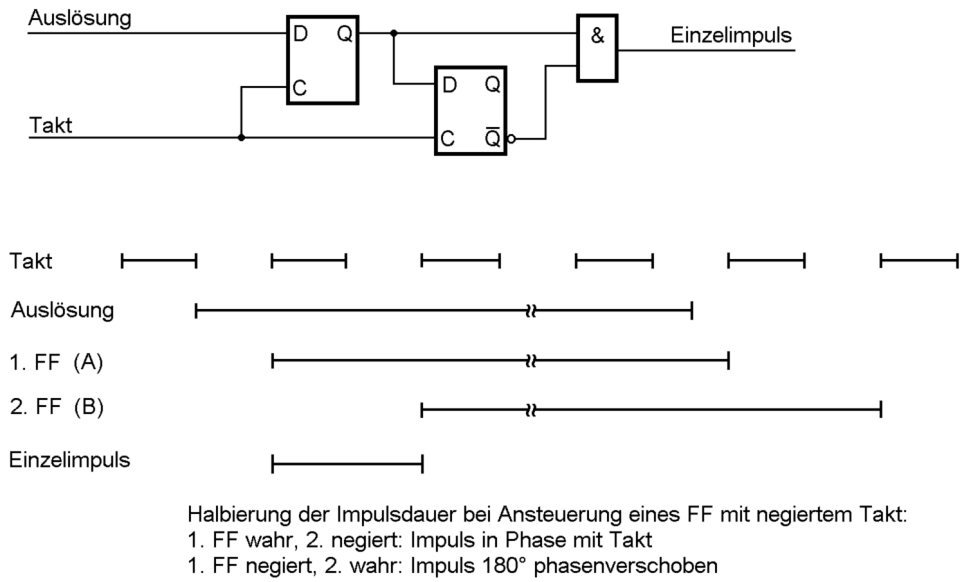


Abbildung 2.28 Einzelimpulserzeugung (Single Shot)

2.3.5. Synchrones Durchsteuern von Taktimpulsen

Die Schaltung gemäß Abbildung 2.28 liefert zwar einen taktsynchronen Ausgangsimpuls, jedoch ist dieser eine ganze Taktperiode lang (und zudem gegenüber dem eigentlichen Takt um eine Flipflop- und eine Gatter-Schaltverzögerungszeit verschoben). Hingegen ist es manchmal erforderlich, einzelne Taktimpulse mit möglichst geringer Verzögerung "auszublenden", also - abhängig von der Stellung eines Steuer-Flipflops - entweder durchzulassen oder zu sperren. Abbildung 2.29 zeigt eine entsprechende Schaltung.

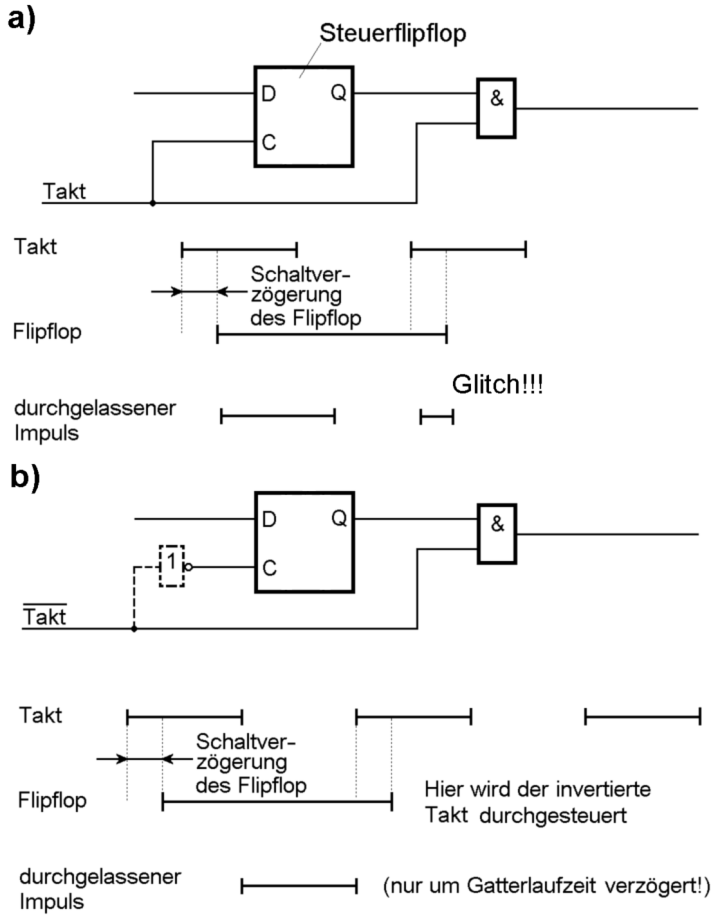


Abbildung 2.29 Synchrones Durchsteuern von Taktimpulsen

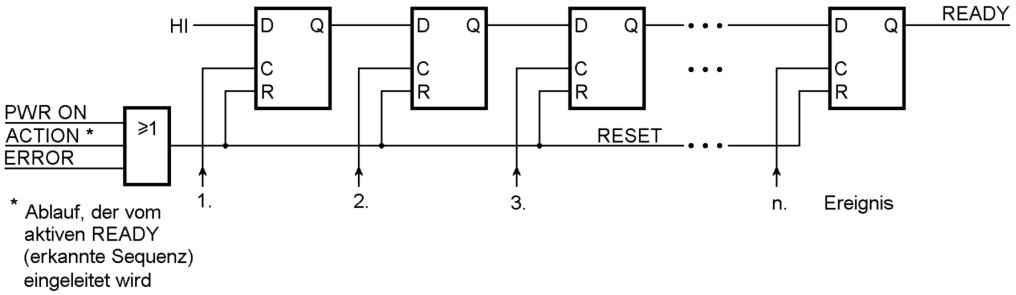
Erklärung:

a) die naive Lösung. Sie funktioniert aber nicht richtig - nämlich beim Ausschalten. Das Flipflop schaltet erst nach der betreffenden Taktflanke aus (Verzögerungszeit). Währenddessen hat aber das Taktisgnal schon das UND-Gatter passiert; am Ausgang entsteht ein Störimpuls.

b) um den Takt sauber (ohne störende "Nadeln") zu steuern, muß das Flipflop mit der jeweils anderen Taktflanke geschaltet werden. Das wird hier durch Invertieren des Taktsignals erreicht. *Hinweis:* Abbildung 5.7b (Seite 239) zeigt eine weitere Schaltungslösung.

2.3.6. Sequenzerkennung

Die Aufgabe: Wir haben impulsförmige Signale A, B, C usw., und wir wollen eine bestimmte Reihenfolge ihres Auftretens erkennen. Die betreffende Schaltung soll aktiv werden, wenn beispielsweise eine Folge A - C - D... eintritt, nicht aber bei beliebigen anderen Folgen. Dies erreicht man durch eine Kette von D-Flipflops, deren Takteingänge in der jeweils geforderten Weise mit den einzelnen Signalen beschaltet sind (Abbildung 2.30). Das letzte Flipflop der Kette wird nur dann aktiv werden, wenn alle Signale in der geforderten Reihenfolge geschaltet haben.



Das READY-Signal erscheint nur, wenn die Ereignisse (Impulse) in der "verdrahteten" Reihenfolge eintreffen.

Anwendungsbeispiel: Code - Schloß (erfordert aber noch Sicherheitsschaltung gegen "Knackversuche").

Abbildung 2.30 Sequenzdetektor

Auf Grundlage der gezeigten Schaltung kann man beispielsweise ein Codeschloß aufbauen. Wir müssen nur gemäß dem jeweils gewünschten Code die Tasten mit den Takteingängen der Flipflops verbinden. Siehe auch Aufgabe 20 in Kapitel 8.

2.4. Register

2.4.1. Datenregister

Register sind Aneinanderreihungen von Flipflops oder Latches mit gemeinsamem Takt. Abbildung 2.31 gibt eine Übersicht über die verschiedenen Ausführungsformen, die Abbildungen 2.32 und 2.33 veranschaulichen einige Einzelheiten.

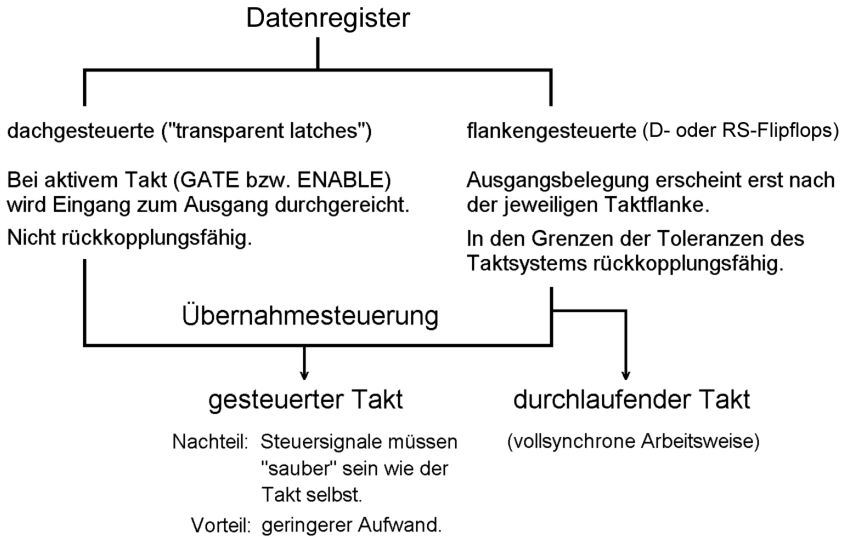


Abbildung 2.31 Datenregister: eine Übersicht

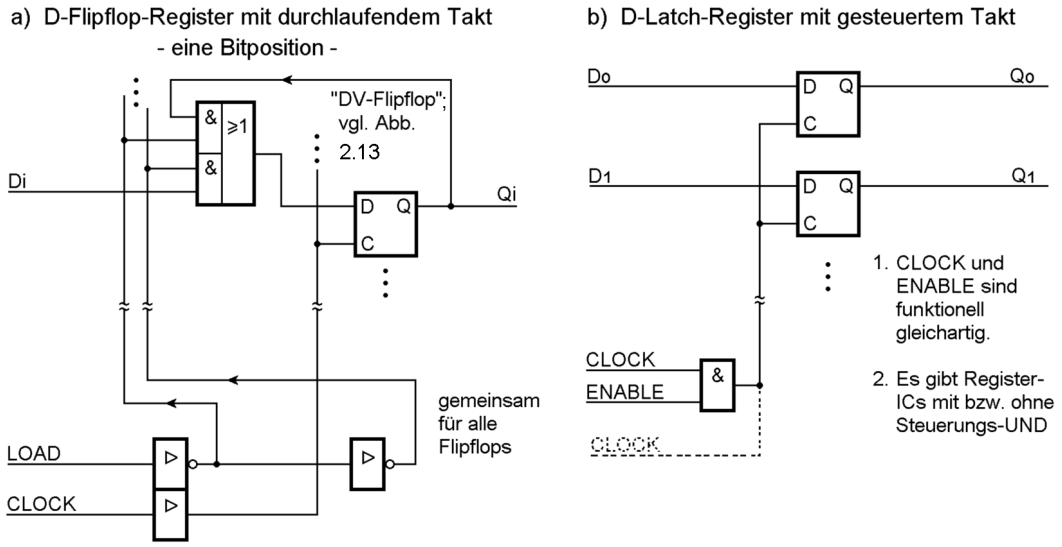


Abbildung 2.32 Einzelheiten: Register mit durchlaufendem und mit gesteuertem Takt

Erklärung zu Abbildung 2.32:

a) vollsynchrones Register. Der Takt liegt ständig an. Bei aktivem Steuersignal LOAD wird die eingangsseitige Datenbelegung in die Flipflops übernommen. Ist LOAD inaktiv, wird die aktuelle Belegung über die Rückführung gehalten (Selbsthaltung, DV-Prinzip). Diese Grundschaltung kann mit zusätzlichen Funktionen erweitert werden (Abbildung 2.33).

b) Register mit gesteuertem Takt. Um eine einmal gespeicherte Belegung zu halten, muß der Takt abgestellt (deaktiviert) werden. Manche Register haben konjunktiv verknüpfte Steuereingänge, manche nur einen einzigen Takteingang.

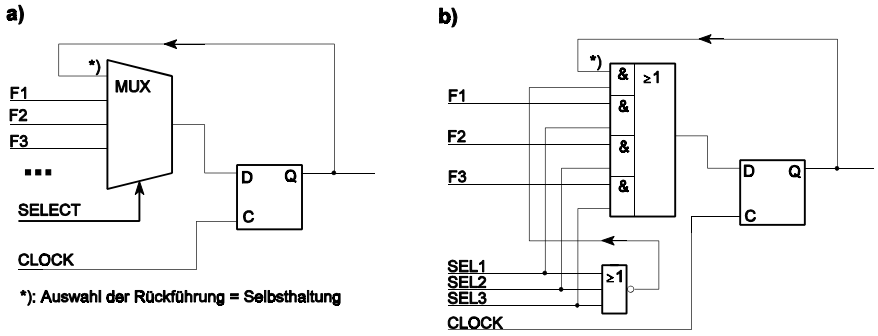


Abbildung 2.33 Vollsynchrone Mehrfunktionsregister

Erklärung:
Vollsynchrone Register können mehr als nur Daten speichern. Die Abbildung zeigt den grundsätzlichen Aufbau von Registern, die verschiedene Funktionen ausführen können (Daten parallel übernehmen, in verschiedenen Richtungen schieben, zählen usw.) Im allgemeinen Fall sind mehrere Funktionen F1, F2, F3 usw. vorgesehen, z. B. F1 = paralleles Laden, F2 = Linksschieben, F3 = Vorwärtszählen usw. Für n Funktionen ist jedem Flipflop ein Multiplexer oder Datenselektor mit n + 1 Eingängen vorgeschaltet. Der zusätzliche Eingang dient der

Selbsthaltung. An die verbleibenden n Eingänge sind die jeweiligen funktionellen Verbindungen und Netzwerke angeschlossen (Dateneingänge, Ausgänge benachbarter Flipflops (zum Schieben), Zählnetzwerke usw.)

a) Funktionsauswahl über Multiplexer. Sind die SELECT-Eingänge mit Nullen belegt, so geschieht nichts, und die Datenbelegung bleibt erhalten (Selbsthaltung). Ansonsten wählt jede SELECT-Belegung eine bestimmte Funktion aus.

b) Funktionsauswahl über Datenselektor. Die Auswahleingänge SEL1, SEL2 usw. werden im 1-aus-n-Code angesteuert. Sind alle Auswahleingänge mit Nullen belegt, so wird über das NOR-Gatter die Rückführung aktiviert (Selbsthaltung).

Industriestandards

Die Schaltkreishersteller haben im Laufe der Zeit viele Registertypen entwickelt. Einige davon sind zu echten Industriestandards geworden (daran erkennbar, daß sie in nahezu allen Baureihen angeboten werden). Abbildung 2.34 gibt - anhand einer kleinen Auswahl - einen Überblick über typische, vielseitig einsetzbare Register.

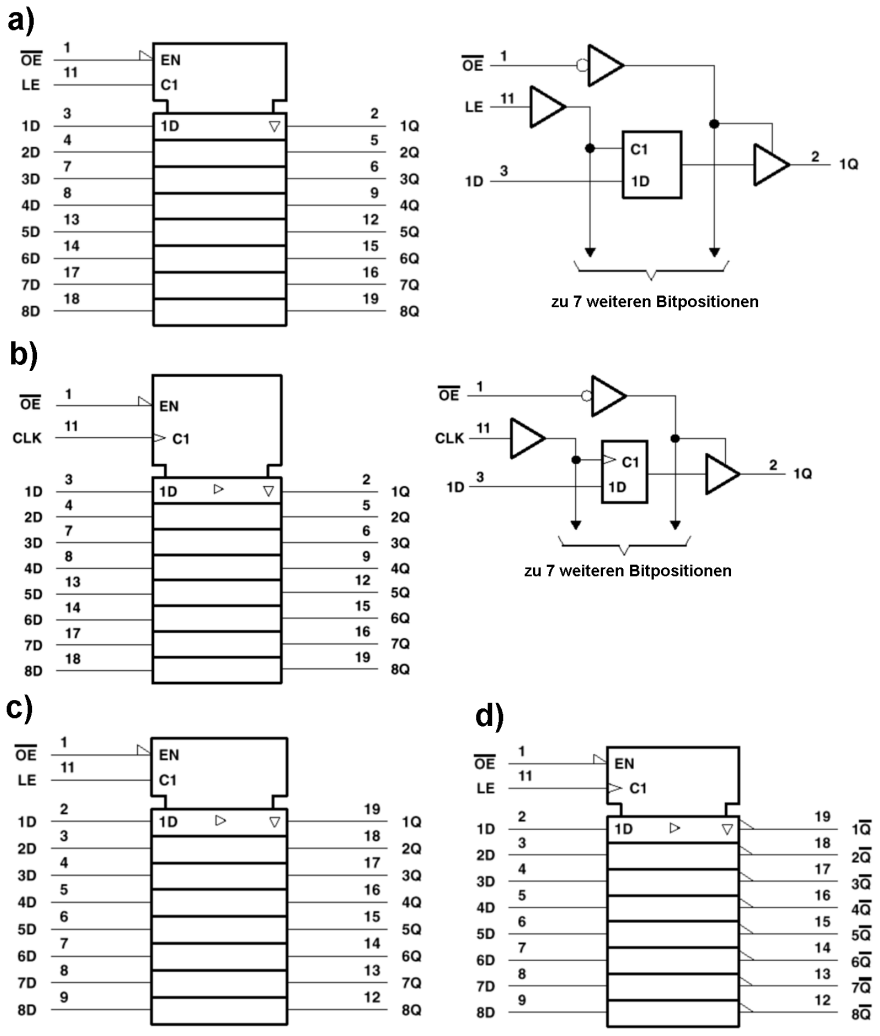


Abbildung 2.34 Typische Register - eine kleine Auswahl

Erklärung zu Abbildung 2.34:
a) - Latch-Register; b) - D-Flipflop-Register; c) - Register gemäß a) oder b) mit anderer Anschlußbelegung; d) D-Flipflop-Register mit negierten Ausgängen. Die meisten Registerschaltkreise haben Tri-State-Ausgänge (im Interesse vielfältiger Einsatzmöglichkeiten). Braucht man zweiwertige Ausgänge, ist der Erlaubniseingang (OE) fest mit Masse zu verbinden. Typischerweise werden Latch- und D-Flipfloptypen mit ansonsten gleicher Auslegung (Gehäuse, Anschlußbelegung, Technologie usw.) angeboten (vgl. a) und b)). Gängige Breiten (Anzahl der Bitpositionen): 8, 16, 18, 20, 32, 36.

Denksportaufgabe:
Woran erkennen Sie in der Abbildung, ob es sich um ein Latch- oder um ein D-Flipflop-Register handelt?

An den Takteingängen (hier: C1) der Schaltsymbole. D-Flipflop-Typen haben ein Dreieck, das die Flankensteuerung kennzeichnet.

Hinweis:
Achten Sie auf die Anschlußbelegung (die Nummern an den Ein- und Ausgängen) von a) und b) einerseits sowie von c) und d) andererseits. Bei a) und b) sind Ein- und Ausgänge gleichsam gemischt (Anschluß 2 = Ausgang, Anschluß 3 = Eingang usw.), bei c) und d) liegen sich hingegen Ein- und Ausgänge auf beiden Seiten gegenüber (1, 2 usw. sind Eingänge, 19, 18 usw. sind Ausgänge). Der Zweck: die Auslegung kostengünstiger Zweiebenen-Leiterplatten zu erleichtern.

2.4.2. Schieberegister

Im Schieberegister sind die Flipflops hintereinandergeschaltet. Vom Dateneingang wird ein Bit in das erste Flipflop geschrieben, mit dem nächsten Takt in das nächste Flipflop übernommen usw. Bei n Flipflops erscheint dieses Bit nach n Taktimpulsen am Ausgang des Schieberegisters. Abbildung 2.35 gibt einen Überblick über die verschiedenen Arten von Schieberegistern.

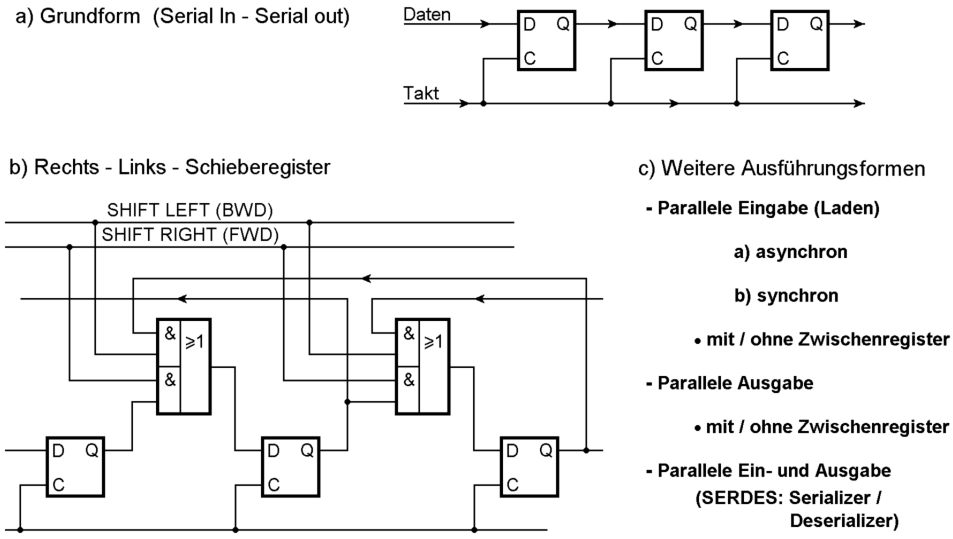


Abbildung 2.35 Schieberegister

Erklärung zu Abbildung 2.35:

a) das einfachste Schieberegister. Nur eine Schieberichtung (in zeichnerischen Darstellungen typischerweise von links nach rechts oder von oben nach unten).

b) Schieberegister mit umschaltbarer Schieberichtung. Hierzu dienen Auswahlschaltungen, die entweder den Ausgang des rechten oder des linken Nachbar-Flipflops zum jeweiligen Dateneingang durchsteuern.

Das Takttoleranzproblem

Der Takt in nachfolgenden Flipflops darf nicht später wirksam werden als in vorgeordneten, da es sonst vorkommen kann, daß die gelieferten Bits nicht übernommen werden, sondern verlorengehen. Abbildung 2.36 veranschaulicht Problem und Lösungsmöglichkeiten. Prinzip: erst die alten Daten übernehmen, dann die neuen einschieben.

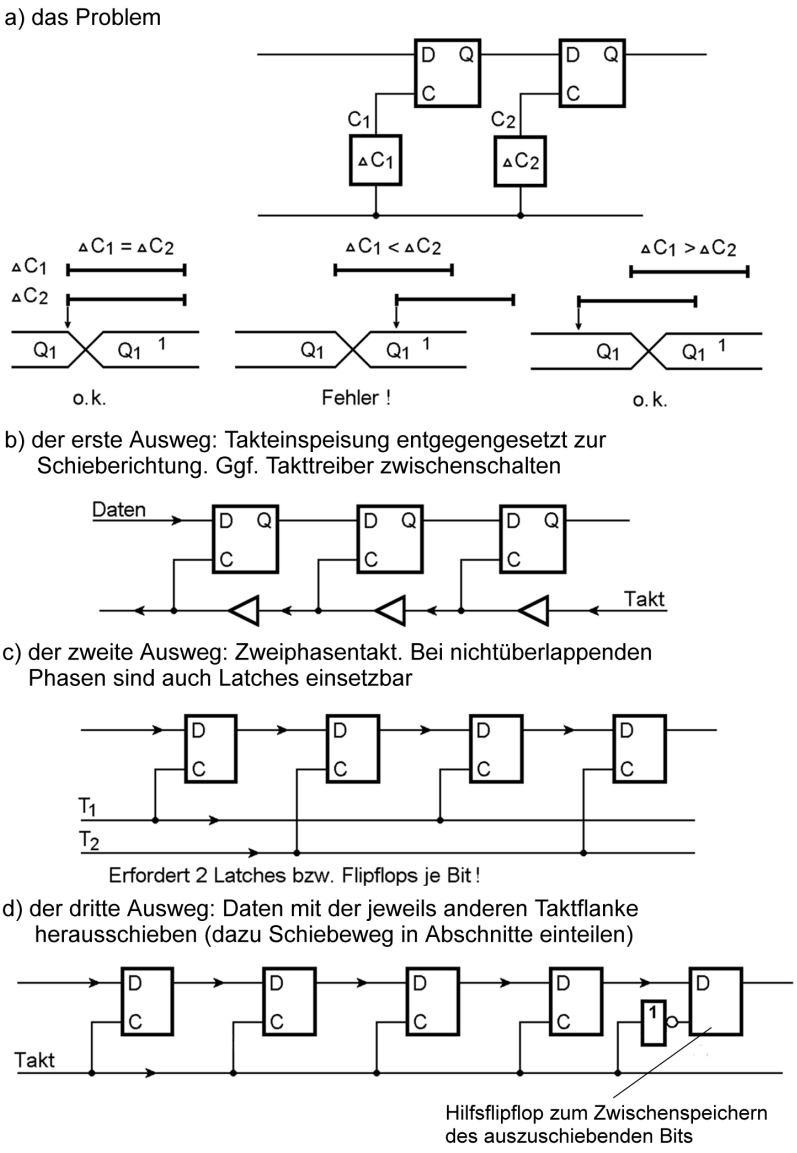


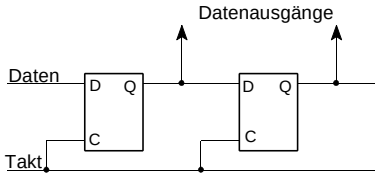
Abbildung 2.36 Takttoleranzproblem bei Schieberegistern

Hinweis:
Das Problem tritt vor allem dann auf, wenn das Schieberegister besonders lang ist oder aus mehreren Schaltkreisen aufgebaut wird. Lösungen gemäß Abbildung 2.36b, c finden wir vor allem im Innern hochintegrierter Schaltkreise. Das Prinzip von Abbildung 2.36d ist bei Schieberegister-Interfaces üblich, die mehrere Schaltkreise miteinander verbinden.

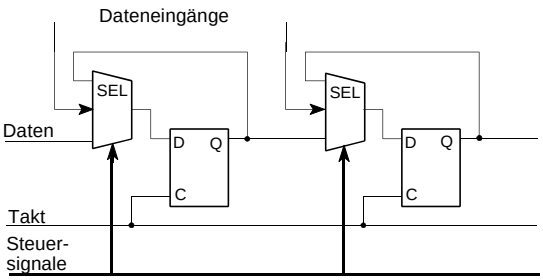
Vorzeichenerweiterung (arithmetisches Rechtsschieben)
Es gibt Schieberegister-Schaltkreise, bei denen ein Verschieben mit Vorzeichenerweiterung vorgesehen ist: bei Rechtsverschiebung bleibt das höchstwertige Bit erhalten und wird mit jedem Taktimpuls in eine weitere niederwertige Bitstelle übernommen.

Parallelausgabe und Paralleleingabe (Laden)
Wenn die einzelnen Flipflops des Schieberegisters ausgangsseitig zugänglich sind, kann man den Registerinhalt parallel abnehmen. Zum parallelen Laden müssen die Schieberegister-Flipflops eingangsseitig zugänglich sein (Abbildung 2.37).

a) Parallelausgabe



b) Paralleleingabe, synchron



c) Paralleleingabe, asynchron

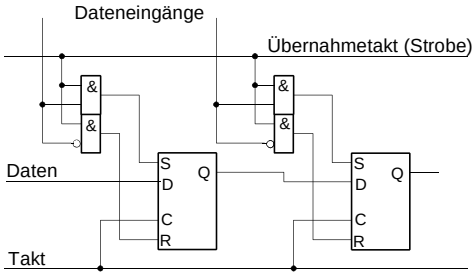


Abbildung 2.37 Prinzipien des Parallelzugriffs auf Schieberegister

Erklärung zu Abbildung 2.37:

a) Parallelausgabe. Die Daten werden einfach von den Ausgängen der Schiebe-Flipflops abgenommen.

b) synchrone Paralleleingabe. Alle Vorgänge beziehen sich auf denselben Takt. Den Schiebeflipflops sind Auswahlschaltungen (z. B. Multiplexer) vorgeordnet. Zum Schieben wird der Schiebeweg durchgeschaltet (Eingang an vorhergehenden Ausgang), zur Paralleleingabe (zum Laden) werden die parallelen Dateneingänge mit den Flipflops verbunden. Zudem hat unser Register eine Rückführung, die dann geschaltet wird, wenn weder geschoben noch geladen werden soll (Selbsthaltung; vgl. auch Abbildung 2.33).

c) asynchrone Paralleleingabe. Hierzu werden die Setz- und Rücksetzeingänge der Schiebeflipflops ausgenutzt; das asynchrone Laden ist ein durch ein Übernahmesignal gesteuertes Setzen (beim Laden einer Eins) oder Rücksetzen (beim Laden einer Null). Beim Laden verhält sich also das Flipflop wie ein Latch.

Pufferregister

Pufferregister sind erforderlich, wenn die Parallelzugriffe vom Schiebevorgang unabhängig sein sollen (Abbildung 2.38). Die Pufferregister können mit transparenten Latches oder mit flankengesteuerten Flipflops aufgebaut sein.

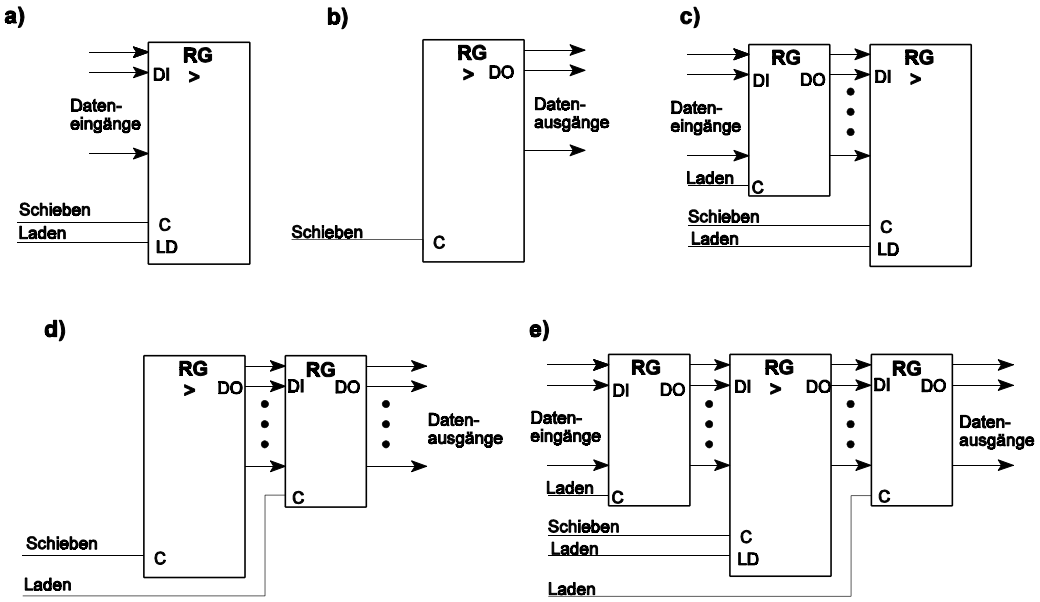


Abbildung 2.38 Schieberegisterstrukturen für Parallelzugriffe

Erklärung:

a) Schieberegister mit Paralleleingabe (Parallel In - Serial Out; vgl. Abbildung 2.37b, c). Schieben und Laden schließen einander aus.

b) Schieberegister mit Parallelausgabe (Serial In - Parallel Out; vgl. Abbildung 2.37a). Der Schiebeablauf macht sich an den parallelen Datenausgängen bemerkbar.

c) vorgeschaltetes Pufferregister zur Paralleleingabe. Während die eine Datenbelegung ausgeschoben wird, kann bereits die nächste in das Pufferregister geladen werden. Die typische Anwendung: die sog. Parallel-Serien-Wandlung (Serialisierung; z. B. um Daten Bit für Bit über ein serielles Interface zu transportieren).

d) nachgeschaltetes Pufferregister zur Parallelausgabe. Es kann gleichsam ein Schnappschuß der Schieberegisterbelegung übernommen werden. Während das Schieben weiterläuft, kann man die übernommenen Daten abtransportieren. Die typische Anwendung: die sog. Serien-Parallel-Wandlung (Deserialisierung; z. B. um Daten zu empfangen, die Bit für Bit über ein serielles Interface geliefert werden).

e) Schieberegister mit ein- und ausgangsseitigen Pufferregistern. Kann serielle Daten sowohl senden als auch empfangen. Fachbegriff: Serializer/Deserializer (SERDES). Derartige Baugruppen findet man in Floppy-Disk- und Festplatten-Controllern, in Netzwerk- (LAN-) Adaptern, in USB-Schaltkreisen usw. Siehe weiterhin Abschnitt 6.1.2.

Die universelle Bedeutung des Schieberegisterprinzips

Das Schieberegister hat einen wichtigen Vorteil: man kann es so lang auslegen wie man will und braucht, um Bits zu transportieren (z. B. zwecks Ein- und Ausgabe), nur einen Dateneingang, einen Datenausgang und einen Takt. Da man mit den übertragenen Bits natürlich noch etwas anfangen möchte, braucht man zusätzlich irgendwelche Steuersignale. Dafür genügt eine einzige Leitung: wird sie aktiviert, so wird die eingeschobene Information zwecks Ausgabe übernommen und durch einzugebende Information ersetzt, die dann nur noch herausgeschoben werden muß (Abbildung 2.39).

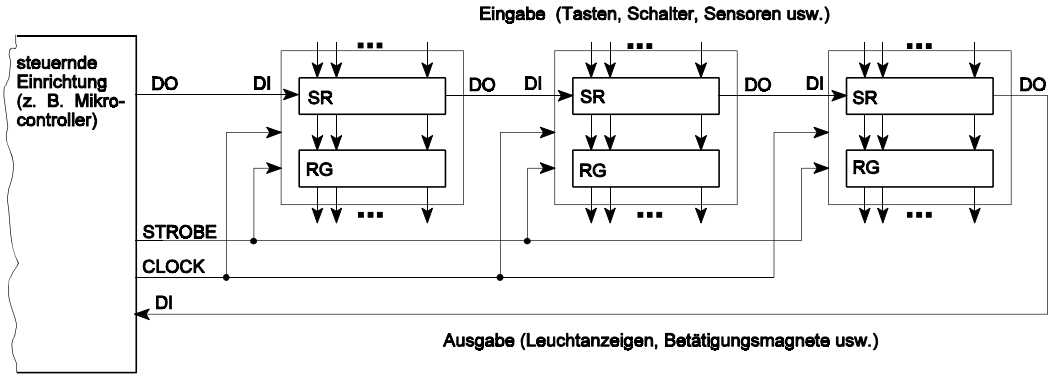


Abbildung 2.39 Ein einfaches Schieberegister-Interface

Erklärung:

SR - Schieberegister; RG - Parallelregister; DI - Dateneingang; DO - Datenausgang; CLOCK - Schiebetakt; STROBE - Parallelübernahmeimpuls. Die einzelnen Einrichtungen enthalten Schieberegisteranordnungen mit parallelen Ein- und Ausgängen. Die auszugebende Information wird Bit für Bit durchgeschoben. Anschließend wird STROBE erregt. Dies bewirkt zum einen das Laden der Parallelregister mit den eingeschobenen Daten (Ausgabe) und zum anderen das Laden der Schieberegister mit den Daten der jeweiligen Umgebung (Eingabe). Mit weiteren Schiebetakten wird dann die übernommene Information Bit für Bit herausgeschoben.

Hinweis:
Grundsätzlich muß man bei der Auslegung eines Interfaces Steuerungsaufwand und Leitungsaufwand gegeneinander abwägen. Eine Vielzahl von Einzelleitungen hat keinen Steuerungsaufwand, aber einen hohen Leitungsaufwand. Eine rein bitserielle Übertragung über einen einzigen Signalweg hat geringsten Leitungsaufwand, aber hohen Steuerungsaufwand (Serialisierung → Modulation → Demodulation → Deserialisierung; Ablaufsteuerung). Das Schieberegister hat hier eine bemerkenswerte Zwischenstellung: nur wenige Leitungen, aber sehr geringen Steuerungsaufwand. Vor allem: man braucht weder Modulation noch Demodulation noch Taktrückgewinnung. Infolge der statischen Arbeitsweise des Schieberegisters kann man solche Interfaces gleichsam Bit für Bit direkt mit Software ansteuern (= hohe Flexibilität bei geringsten Kosten).

Wichtige Anwendungen des Schieberegisterprinzips sind:

- elementare Ablaufsteuerschaltungen, insbesondere dann, wenn vorwiegend zeitstarre Impulsfolgen benötigt werden,
- das Prüfen von Digitalschaltungen (Stichworte: Scan Design und Boundary Scan),
- das Programmieren programmierbarer Schaltkreise (GALs, CPLDs usw.); diese haben serielle Schiebewege für die Programmierdaten,
- Elementar-Interfaces zwischen hochintegrierten Schaltkreisen (u. a. serielle EEPROMs).

Das Schieberegister als Elementar-Interface
Das Schieberegisterprinzip wird als kostengünstiges und universelles Interface zum Verbinden von Schaltkreisen und Funktionseinheiten innerhalb von Geräten (ja sogar auf Leiterplatten) verwendet (eine typische Anwendung ist z. B. der Anschluß eines Bedienfeldes an den zentralen Mikrocontroller). Es gibt sowohl herstellerspezifische Lösungen als auch Industriestandards.

Industriestandards
Die wichtigsten: I^2C-Bus, Microwire und SPI (Tabelle 2.3).

I^2C-Bus (Philips)	**Microwire (National Semiconductor)**	**SPI (Synchronous Peripheral Interconnect; Motorola)**
■ 2 Leitungen, ■ Open Drain, ■ echter Bus, ■ keine Contention, ■ Adreßdecodierung in den Einrichtungen, ■ echtes Multimaster-Protokoll, ■ massiv standardisiert	■ 3 bzw. 4 Leitungen*), ■ Tristate-Ausgänge, ■ zentraler Master, ■ Bus mit Einzelauswahl der Slave-Einrichtungen (keine durchgehende Schiebekette), ■ Taktpolarität und Schiebeordnung festliegend, ■ weitgehende Narrenfreiheit**)	■ 3 bzw. 4 Leitungen*), ■ Tristate-Ausgänge (in manchen Schaltkreisen auf Open Drain umschaltbar), ■ zentraler Master, ■ Bus mit Einzelauswahl der Slave-Einrichtungen (keine durchgehende Schiebekette), ■ Taktpolarität und (manchmal) Schiebeordnung programmierbar, ■ weitgehende Narrenfreiheit**)

*): 3 Interfaceleitungen + 1 Schaltkreisauswahlsignal; **): mit Ausnahme von Industriestandard-Einrichtungen (z. B. seriellen EEPROMs)

Tabelle 2.3 Industriestandards für Einfachinterfaces im Überblick

Im folgenden wollen wir uns darauf beschränken, Microwire und SPI als typische Schieberegister-Interfaces kurz vorzustellen (Abbildungen 2.40, 2.41).

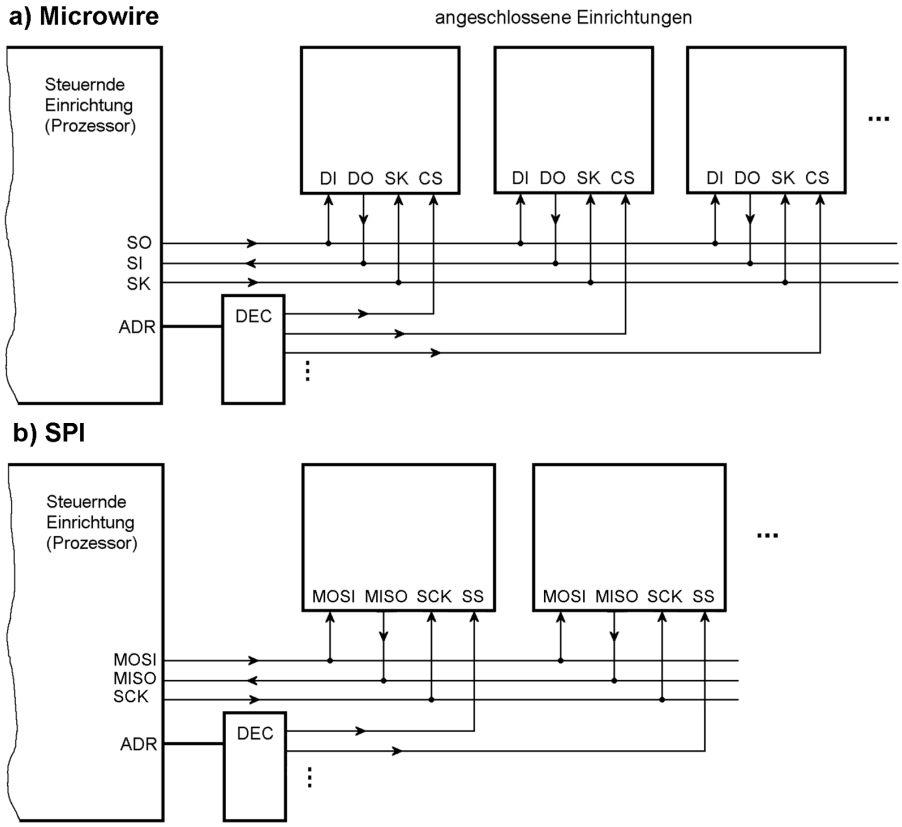


Abbildung 2.40 Microwire und SPI

Erklärung:
Die Interfaces verbinden periphere Einrichtungen mit einer zentralen Steuerung (das ist üblicherweise ein Mikrocontroller). Beide Interfaces umfassen - an der einzelnen angeschlossenen Einrichtung - 4 Leitungen:

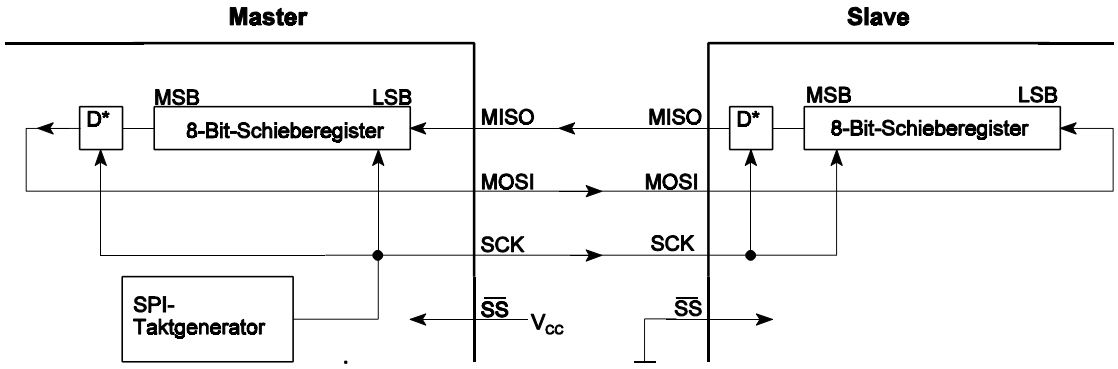
- Microwire: Dateneingang DI, Datenausgang DO, Schiebetakt SK und Schaltkreisauswahl CS. Es ist möglich, DI und DO zu einer bidirektionalen Datenleitung zusammenzuschalten.
- SPI: Dateneingang MOSI, Datenausgang MISO; Schiebetakt SCK, Schaltkreisauswahl SS. (MOSI = Master Out/Slave In; MISO = Master In/Slave Out; SCK = Shift Clock; SS = Slave Select.)

Keine Schiebekette, sondern ein bitserieller Bus
Es liegt nahe, die Schieberegister in den einzelnen Einrichtungen einfach hintereinanderzuschalten (Daisy-Chain-Prinzip), so daß - wie in Abbildung 2.39 gezeigt - der Verbund aller Einrichtungen wie ein einziges sehr langes Schieberegister erscheint. Der Vorteil: man braucht keine Schaltkreisauswahl und kommt (am steuernden Mikrocontroller) mit insgesamt 4 Anschlüssen aus, gleichgültig wieviele Einrichtungen angeschlossen sind. Aber keines der Industriestandard-Interfaces hat man so ausgelegt. Weshalb? - Es geht vor allem darum, Schaltkreise auf Leiterplatten untereinander zu verbinden. Und die Hersteller legen Wert

darauf, durch selektives (teilweises) Bestücken der Leiterplatten Varianten bauen zu können (z. B. verschiedene Rundfunk- oder Fernsehgeräte auf Grundlage einer einzigen Hardware-Plattform). In einer langen Schiebekette würde aber das Weglassen eines Schaltkreises den Datenweg unterbrechen.

Microwire
Die zentrale Steuerung wählt jeweils eine angeschlossene Einrichtung aus, indem sie die betreffende CS-Leitung aktiviert. Auszugebende Information wird über DO geliefert und in der ausgewählten Einrichtung mit der Vorderflanke der SK-Impulse übernommen; einzugebende Information wird über DI mit SK-Taktimpulsen Bit für Bit abgeholt. Das Schieben ist byteweise organisiert. Was die einzelnen Bytes bedeuten, ist jeweils schaltkreisspezifisch (und ist im Datenmaterial erklärt).

SPI
Der jeweilige Slave wird durch Aktivierung des Eingangs SS ausgewählt. Der Datenweg zwischen Master und ausgewähltem Slave ist an sich nichts anderes als eine Ringstruktur aus zwei Schieberegistern (Abbildungen 2.41, 2.42).



MSB - höchstwertiges Bit; LSB - niedrigstwertiges Bit.
Hier ist die Schnittstelle so konfiguriert, daß das höchstwertige Bit als erstes übertragen wird (MSB First) .

D*: Flipflop wird mit der jeweils anderen SCK-Flanke getaktet. Vgl. Abbildung 2.36d.

Abbildung 2.41 SPI: der Datenweg zwischen Master und Slave

Erklärung:
Die Abbildung zeigt eine Art Schaltungsmodell bzw. Ersatzschaltung, um die typischen Besonderheiten zu veranschaulichen. Es wird byteweise geschoben. Das Byte im Schieberegister des Masters gelangt in den Slave, während gleichzeitig das Byte im Schieberegister des Slaves in den Master geschafft wird. Dabei wird mit der einen Taktflanke der auf der jeweiligen Datenleitung anliegende Wert übernommen und in das jeweils empfangende Schieberegister eingeschoben, mit der anderen wird das jeweils nachfolgende Bit ausgeschoben. Diese Betriebsweise (vgl. auch Abbildung 2.36d) vermeidet das Takttoleranzproblem - deshalb kann SPI mit viel höheren Taktfrequenzen betrieben werden als Microwire.

Die SPI-Hardware in einem Mikrocontroller ist typischerweise als autonom arbeitende State Machine ausgelegt, die - von der Software gesteuert - wahlweise als Master oder als Slave betrieben werden kann. Im einzelnen sind die Konfigurationsmöglichkeiten schaltkreisspezifisch. Programmseitig einstellbar sind u. a.:

- das Übertragungsformat: Einzelbyteübertragung (nach Übertragung eines jeden Bytes wird SS wieder inaktiv) oder Mehrbyteübertragung (aufeinanderfolgende Bytes werden übertragen, während SS aktiv bleibt),
- die Polarität des SCK-Signals (aktiv High oder aktiv Low),
- die Schiebeordung (höchstwertiges oder niedrigstwertiges Bit zuerst).

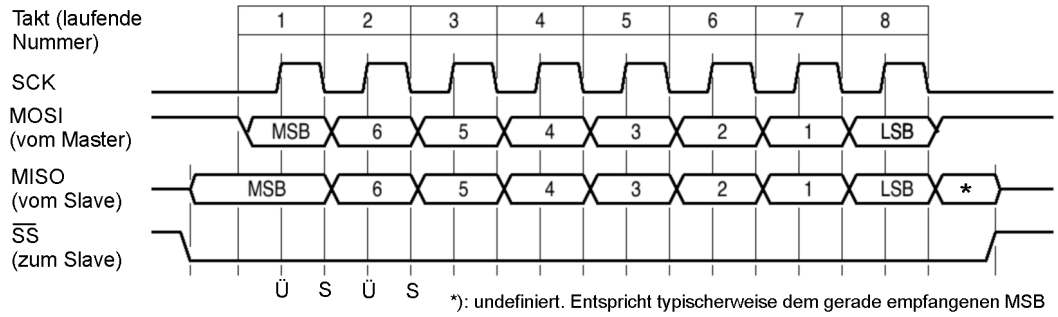


Abbildung 2.42 Datentransport über SPI (Motorola). Ü - Datenübernahme; S - Ausschieben

Erklärung:
Wir zeigen hier nur einen der möglichen Betriebsfälle. Das Schieben eines Bytes erfordert 8 Takte. Zuvor ist SS zu aktivieren und nach dem Schieben wieder zu deaktivieren. Datenübernahme mit der Low-High-Flanke von SCK, Weiterschieben mit der High-Low-Flanke. Die letzte High-Low-Flanke bewirkt im Slave ein weiteres Schieben, in dessen Ergebnis MISO typischerweise mit dem ersten der zuvor eingeschobenen Bits belegt wird.

Multi-Master-Betrieb
SPI kann als echter Bus betrieben werden, auf den sich mehrere Einrichtungen wahlweise aufschalten können. Die Mastervermittlung muß aber gesondert verwirklicht werden. (Typischerweise kann man den SS-Anschluß im Master-Betrieb so konfigurieren, daß dessen Aktivierung das Freigeben des Busses veranlaßt.)

Linear rückgekoppelte Schieberegister
Das linear rückgekoppelte Schieberegister (Linear Feedback Shift Register LFSR) ist eine ganz wichtige Schaltungsstruktur vor allem in der Fehlererkennungs-, Prüf- und Verschlüsselungstechnik (es dient u. a. zur Fehlererkennung und -korrektur mittels zyklischer Codes (CRC/ECC) sowie zum Erzeugen pseudo-zufälliger Testdaten und Schlüsselangaben). Ein Schieberegister heißt "rückgekoppelt", wenn Flipflop-Ausgänge über kombinatorische Netzwerke auf Eingänge zurückgeführt sind. Die Rückkopplung heißt "linear", wenn die Rückführungen nur Antivalenzverknüpfungen (XOR-Gatter) oder Direktverbindungen enthalten. Abbildung 2.43 gibt einen Überblick über solche Schieberegisterstrukturen.

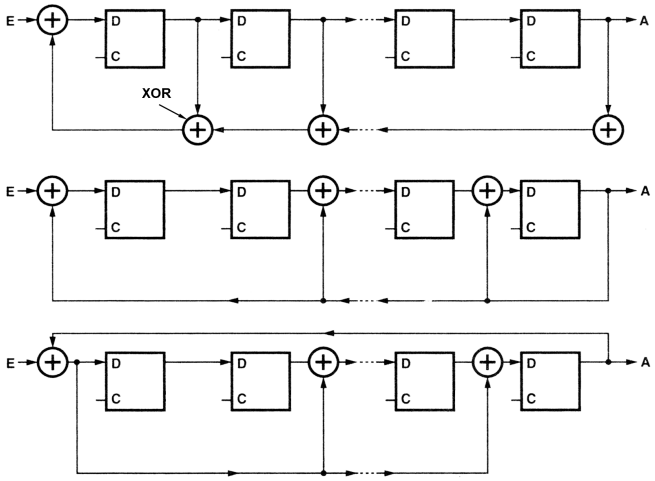


Abbildung 2.43 Linear rückgekoppelte Schieberegister (Grundstrukturen)

Schieberegisterfolgen maximaler Zykluslänge (Maximum Length Shift Register Sequences)
Schieberegister gemäß Abbildung 2.43 können auch ohne Eingangssignal (genauer: mit Festwert 0 am Eingang) betrieben werden (Abbildung 2.44). Ersichtlich ist, daß gar nichts passiert, wenn alle Flipflops anfänglich mit Null belegt sind: $0 \oplus 0 = 0$. Andererseits bringt jede von Null verschiedene Anfangsbelegung infolge des Verschiebens und der Rückführungen mit jedem Takt einen neuen Schieberegisterinhalt zustande. Demgemäß kann an jedem der Flipflops ein Impulsmuster bzw. eine Bitfolge abgenommen werden.

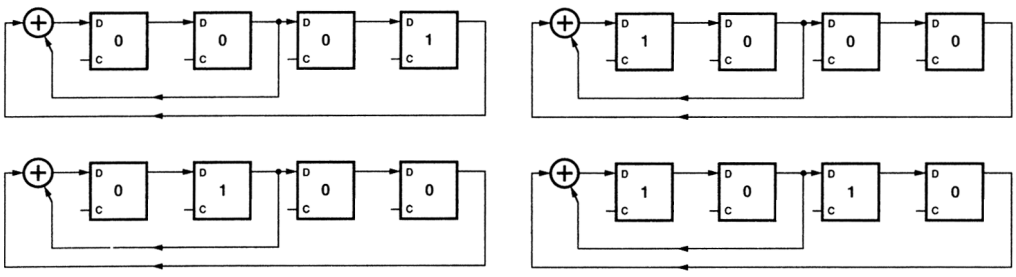


Abbildung 2.44 Beispiel eines rückgekoppelten Schieberegisters ohne Eingangssignal. Es sind die ersten Belegungen dargestellt, die sich vom Anfangswert 0001 an ergeben

Im Verlauf des Schiebens wird sich irgendwann die anfängliche Belegung wieder einstellen. Die Anzahl der Takte, die dafür notwendig ist, heißt die *Zykluslänge* des Schieberegisters. Von besonderem Interesse sind Schieberegister, die Bitfolgen mit *maximaler* Zykluslänge erzeugen. Die maximale Zykluslänge eines Schieberegisters mit n Flipflops ist $2^n - 1$ (alle 2^n möglichen Belegungen abzüglich der Belegung 00..0, die nicht vorkommen darf). Ein solches Schieberegister liefert Folgen, die sich als *pseudo-zufällige* Bitmuster (bzw. als binäre Pseudo-Zufallszahlen) eignen. Sie haben u. a. folgende Eigenschaften:

- die Anzahl der Einsen in der gesamten Folge ist um 1 größer als die Anzahl der Nullen (beide Binärwerte kommen also näherungsweise gleich häufig vor),
- eine Folge von n Einsen kommt genau einmal im Zyklus vor, eine Folgen von $^n/_2$ Einsen zweimal usw. Sinngemäß gilt das für Folgen von (n-1) Nullen, $^n/_2$ Nullen usw.

Hinweise:

1. Um reproduzierbare Bitmuster zu erhalten, muß das Schieberegister stets mit einem bestimmten Anfangswert (Seed Value; ≠ 0) initialisiert werden.
2. Mit 32 Flipflops beträgt die maximale Zykluslänge über 4 Milliarden (genau: 4 294 967 295), mit 64 Flipflops über 18 Trillionen (genau: 18 446 744 073 709 551 615) Takte. Die sich in jedem Taktzyklus ergebenden Muster von beispielsweise 32 oder 64 Bits Länge kann man als Binärzahlen interpretieren. Eine Folge derartiger Binärzahlen sieht "wie zufällig" aus (näherungsweise bzw. Pseudo-Zufallszahlen). Um das Bildungsgesetz zu rekonstruieren (beispielsweise um einen entsprechenden Schlüssel zu knacken) braucht man zweierlei: den Anfangswert und das Rückkopplungssschema.
3. Solche Schieberegister kann man auch als Zähler ausnutzen. Ihre Zählweite (= Zykluslänge) ist nur um Eins kürzer als die eines Binärzählers mit gleicher Anzahl an Flipflops. Sie sind deutlich schneller als synchrone Binärzähler, da hier den Flipflops keine Zählnetzwerke größerer Schaltungstiefe vorgeordnet sind. Infolge der pseudo-zufälligen Zählweise ist es aber aufwendig, Zählerstellungen zu decodieren und den jeweiligen Zählerstand anzuzeigen.

2.5. Zähler und Teiler

Zähler und Teiler liefern ihre Ausgangssignale in Abhängigkeit von der *Anzahl* der einlaufenden Taktimpulse.

Bezeichnungsweise
Bei einem Zähler (Counter) kann man die "irgendwie" codierte aktuelle Anzahl der gezählten Taktimpulse parallel abnehmen, beim Teiler (Divider; sprich: Difeider) hingegen nicht (anders ausgedrückt: Teiler haben nur einen Ausgang, Zähler so viele, wie Flipflops vorgesehen sind). Wir werden deshalb Zähler und Teiler weitgehend gemeinsam behandeln und dabei "Zähler" als Allgemeinbegriff verwenden.

Codierung
Hat man n Flipflops, so lassen sich damit maximal 2^n verschiedene Zustände codieren und also auch abzählen. Die Codierung des Zählers bestimmt, welche Flipflop-Belegungen bei aufeinanderfolgenden Taktimpulsen gebildet werden. Wir betrachten im folgenden nur jene Codierungen, die (1) in der Praxis häufig vorkommen, für die es (2) fertige Zählerschaltkreise gibt bzw. die (3) vergleichsweise einfach realisiert werden können. Abbildung 2.45 gibt einen Überblick über die verschiedenen Arten von Zählern und Teilern.

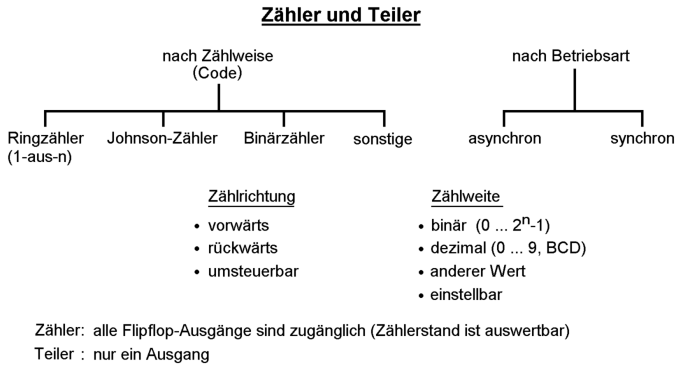


Abbildung 2.45 Zähler und Teiler: eine Übersicht

Zählen "modulo n"
Das ist die Bezeichnung dafür, daß sich nach jeweils n Taktimpulsen der Zählablauf wiederholt. Gleichbedeutender Begriff: *Zählweite*. Beispiel: der Zählablauf 0 - 1 - 2 - 3 - 4 - 0 - 1... wiederholt sich alle 5 Takte; es handelt sich also um einen modulo-5-Zähler.

Der Zähler als State Machine
Herkömmlicherweise wird oft nur der Binär- oder Dezimalzähler als Zähler im eigentlichen Sinne angesprochen. Diese Auffassung aber ist fachlich nicht exakt und erschwert das Verständnis moderner Schaltungslösungen. Wichtig sind zunächst allein die Anzahl der Zustände und die vorgesehenen Zustandsübergänge, unabhängig von der Zustands*codierung* (Abschnitt 2.6.2.). Wir merken uns deshalb: Ein modulo-n-Zähler ist eine State Machine mit n Zuständen und folgenden Merkmalen (Abbildung 2.46):

- jeder Takt veranlaßt einen Zustandswechsel zu einem eindeutig bestimmten Folgezustand,
- nach n Takten wird wieder der erste Zustand eingenommen.

Dieser einfache Ablauf (ständiges Zählen) kann durch Zustände des Ladens und durch Ruhezustände modifiziert werden.

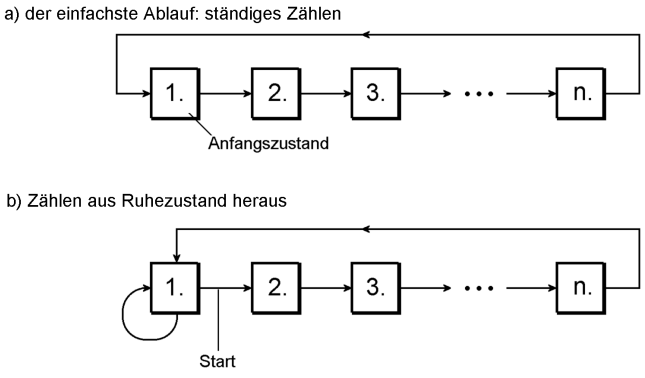


Abbildung 2.46 Der modulo-n-Zähler als State Machine

2.5.1. Ringzähler

Ringzähler zählen mit n Flipflops im 1-aus-n-Code modulo n. In jedem Zustand ist ein Flipflop aktiv, während die anderen inaktiv sind. Ein solcher Zähler ist praktisch ein rückgekoppeltes Schieberegister, in dem eine Eins zyklisch umläuft (Abbildung 2.47).

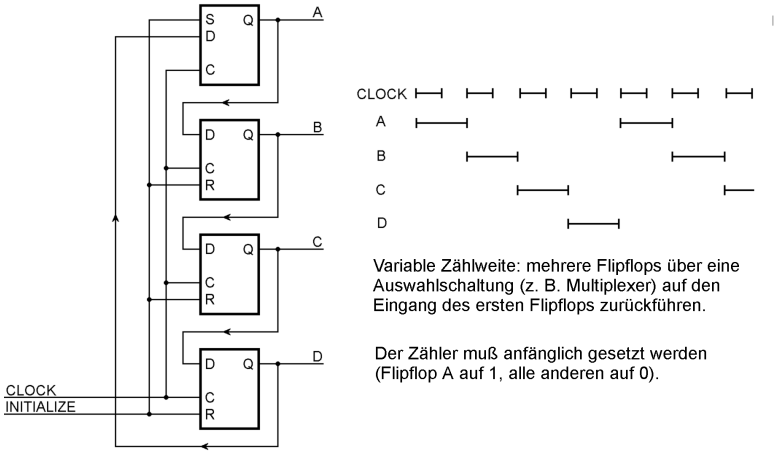


Abbildung 2.47 Ringzähler

Wird die Rückführung mittelsr eine NAND- oder NOR-Verknüpfung der ersten n-1 Flipflops gebildet (Abbildung 2.48), so schwingt sich die Schaltung nach spätestens n Taktimpulsen selbst ein. Bei NAND-Rückführung liefern die Flipflop-Ausgänge eine invertierte 1-aus-n-Codierung (umlaufende Null), bei NOR-Rückführung eine "wahre" (umlaufende Eins).

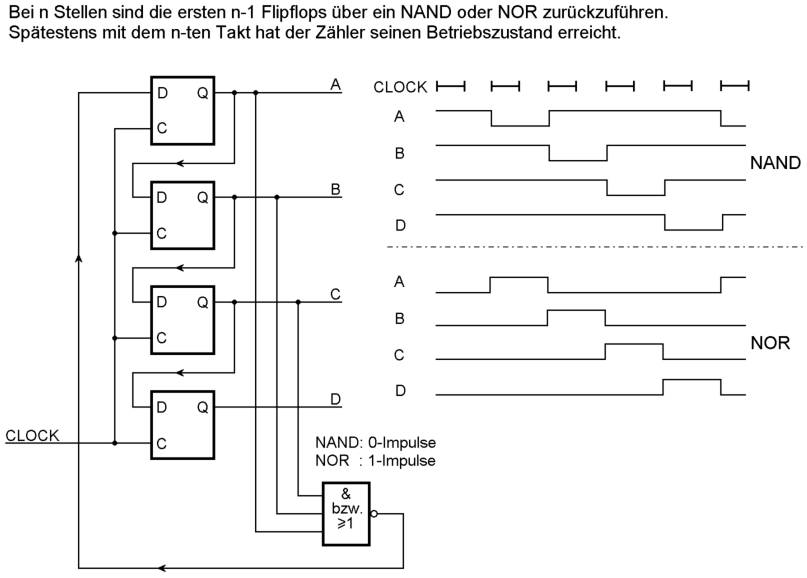


Abbildung 2.48 Selbsteinschwingende Ringzähler

2.5.2. Johnson-Zähler

Ein solcher Zähler zählt mit $^{n}/_{2}$ Flipflops modulo n. Dazu wird das Ringzählerprinzip abgewandelt: (1) wird die invertierte Ausgangsbelegung zurückgeführt, (2) werden anfänglich alle Flipflops gelöscht. Abbildung 2.49 veranschaulicht Schaltung und Zählweise.

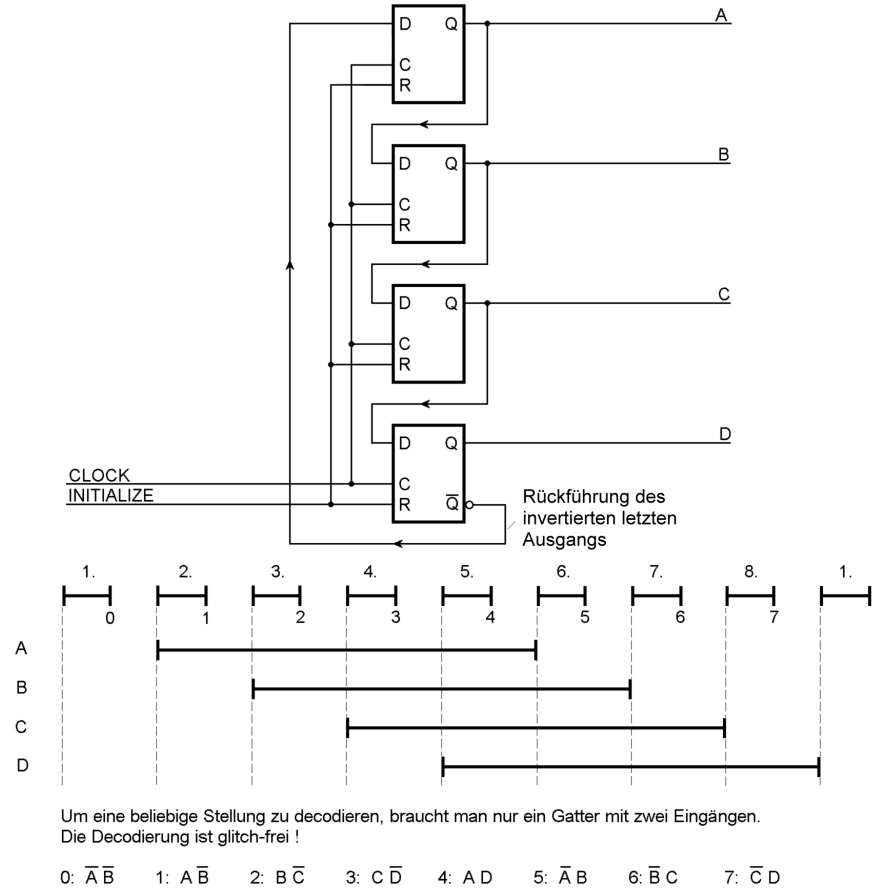


Abbildung 2.49 Johnson-Zähler

Hinweis:
Beim Johnson-Zähler ändert sich mit jedem Takt nur eine Flipflop-Belegung; er kann somit *glitchfrei* decodiert werden (vgl. Hinweis 1 auf Seite 66).

2.5.3. Binärzähler

Beim Binärzähler entspricht jedes Flipflop einer Binärstelle. Mit n Flipflops beträgt die maximale Zählweite 2^n. Die einzelnen Werte liegen zwischen 0 und 2^n - 1. Binärzähler können synchron oder asynchron aufgebaut sein. Sie können für beliebige Zählweiten und für beide Zählrichtungen (vorwärts oder rückwärts) eingerichtet werden.

Asynchronzähler (Ripple Counter)
Diese werden mit Toggle-Flipflops (T-Flipflops) aufgebaut. Zur Wiederholung (vgl. Abschnitt 2.2.4.): ein T-Flipflop erhält man aus einem JK-Flipflop durch Verbinden von J und K und

Belegen mit dem Festwert 1 oder aus einem D-Flipflop durch Rückführen des invertierten Ausgangs auf den Eingang (Abbildung 2.50). Tabelle 2.4 veranschaulicht das Zählscheima.

dezimal	C= 2^2	B = 2^1	A = 2^0	Erläuterung
0	0	0	0	
1	0	0	1	Π Stelle 2^0: ändert sich mit jedem Zählimpuls
2	0	1	0	Π Stelle 2^1: ändert sich mit jedem 2. Zählimpuls,
3	0	1	1	
4	1	0	0	Π Stelle 2^2 ändert sich mit jedem 4. Zählimpuls
5	1	0	1	usw.
6	1	1	0	
7	1	1	1	

A, B, C sind die Flipflops in Abbildung 2.50

Tabelle 2.4 Das Zählschema des Binärzählers (anhand von 3 Stellen)

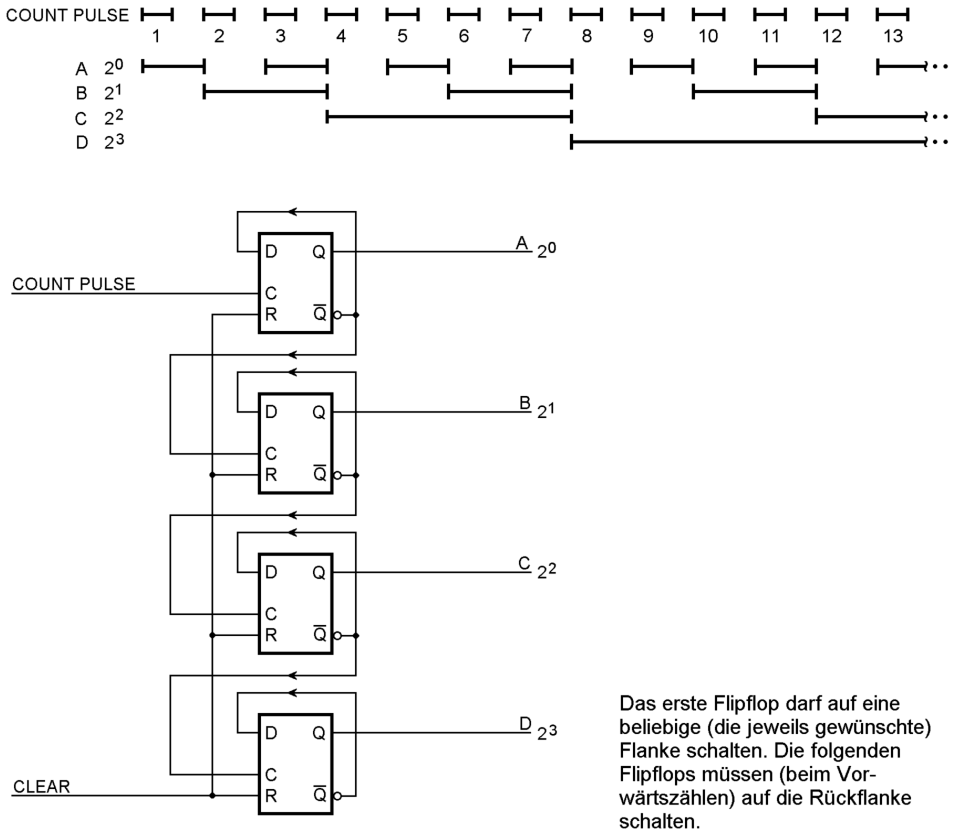


Abbildung 2.50 Ein Asynchronzähler

Erklärung zu Abbildung 2.50:
Die Funktionsweise können wir uns aus Tabelle 2.4 leicht erschließen:

- die niedrigstwertige Stelle (hier: Flipflop A) schaltet mit jedem Zählimpuls (0 - 1- 0 - 1 usw.). Das entspricht direkt der Schaltweise eines T-Flipflops.
- die nächsthöhere Stelle (Flipflop B) schaltet immer dann, wenn Flipflop A von 1 nach 0 schaltet,
- die wiederum nächsthöhere Stelle (Flipflop C) schaltet immer dann, wenn Flipflop B von 1 nach 0 schaltet usw.

Der Zähler ergibt sich somit als einfache Hintereinanderschaltung von T-Flipflops, die mit der High-Low-Taktflanke schalten. Dabei wird der Takteingang jeweils an den Ausgang des vorhergehenden Flipflops angeschlossen.

Wir erkennen sofort einen wesentlichen Nachteil: jedes Flipflop schaltet erst, nachdem das vorhergehende geschaltet hat. Bei n Flipflops dauert es somit n Flipflop-Schaltzeiten, bis der neue Zählwert gebildet ist. Der Asynchronzähler hat aber auch einen bedeutsamen Vorteil. Näheres in Abschnitt 2.5.4.

Synchronzähler
Alle Flipflops sind gemeinsam mit dem Zähl-Takt beschaltet. Für jedes Flipflop werden besondere Zählbedingungen mit kombinatorischen Verknüpfungen gebildet. Immer dann, wenn eine solche Bedingung aktiv ist, schaltet das betreffende Flipflop entsprechend um. Die Bedingungen lassen sich für Vorwärts- und Rückwärtszählen einrichten (Abbildung 2.51).

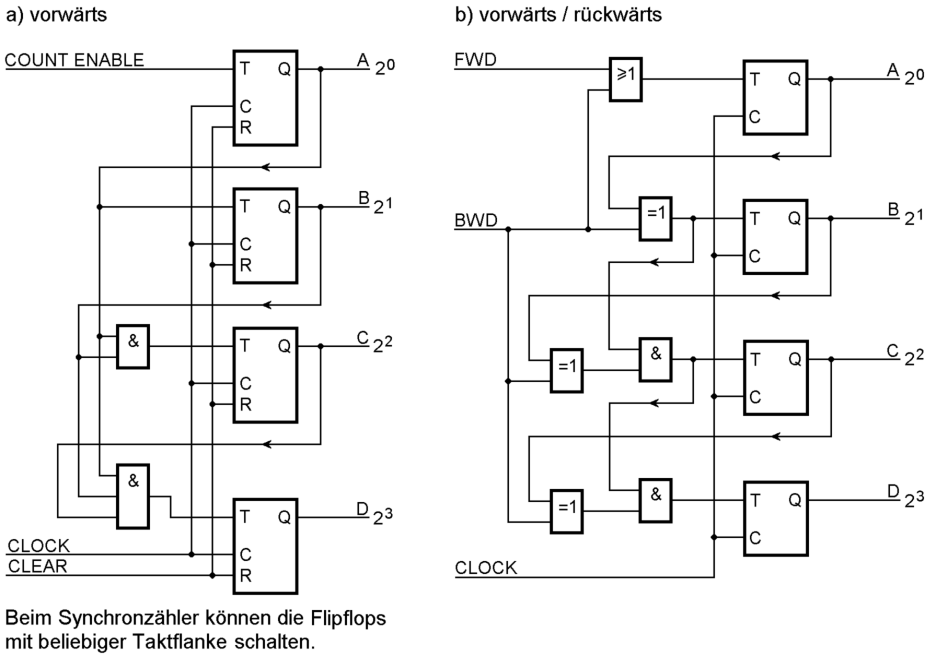


Abbildung 2.51 Synchrone Binärzähler

Erklärung zu Abbildung 2.51:
a) - Vorwärtszähler mit einstufigen Zählnetzwerken; b) - Vorwärts-Rückwärts-Zähler mit kaskadierten Zählnetzwerken.

Der Einfachheit halber haben wir hier T-Flipflops mit steuerbarem T-Eingang dargestellt (die man gemäß Abbildung 2.13 aus JK- oder aus D-Flipflops bauen kann). Bei vollsynchronem Betrieb liegen immer Taktimpulse an. Ob die Flipflops schalten oder nicht, wird durch die Kombinatorik bestimmt. Unsere Zähler zählen, solange das entsprechende Erlaubnissignal (COUNT ENABLE oder FWD oder BWD) aktiv ist. Eine Eins am T-Eingang hat zur Folge, daß sich die Belegung des Flipflops mit dem nächsten Takt ändert (von Low nach High oder umgekehrt). Wann sind nun solche Änderungen auszulösen? Wir wollen uns hier auf das Vorwärtszählen beschränken (vgl. Tabelle 2.4):

- Bitposition 2^0: offensichtlich mit jedem Takt (Schaltweise 0 - 1 - 0 - 1 usw.),
- Bitposition 2^1: offensichtlich dann, wenn Bitposition $2^0 = 1$ ist,
- Bitposition 2^2: offensichtlich dann, wenn die Bitpositionen 2^0 UND $2^1 = 1$ sind.

Zusammengefaßt: beim Vorwärtszählen schaltet eine Bitposition dann um, wenn alle jeweils niederwertigen Bitpositionen mit Einsen, beim Rückwärtszählen dann, wenn sie mit Nullen belegt sind.

Die Ansteuernetzwerke der T-Flipflops (Zählnetzwerke) sind somit einfache UND-Gatter. Deren Eingangszahl nimmt mit jeder Bitstelle um 1 zu (Abbildung 2.51a). Um den Aufwand gering zu halten, kann man die UND-Verknüpfungen kaskadieren. Im Extremfall kommt man je Bitstelle mit einem 2-fach-UND aus, dessen Ausgang den Eingang des nachfolgenden UNDs bildet (Abbildung 2.51b). Das begrenzt aber die erreichbare Zählfrequenz (da - bei n Bitpositionen - bis zu n-2 UND-Gatter durchlaufen werden müssen. In der Praxis wendet man deshalb meist Kompromißlösungen an. Der Zähler wird gleichsam in Stücke geschnitten (zu typischerweise 3...8 Bits), wobei in jedem Abschnitt UND-Verknüpfungen ähnlich Abbildung 2.51a angeordnet sind. Die einzelnen Abschnitte werden dann kaskadiert (Übertragsweitergabe). Der aus einer Zählerstufe heraus weitergegebene Übertrag ist die Zählerlaubnis für die nachfolgende Stufe.

Beliebige Zählweite
Um eine beliebige Zählweite ($< 2^n$) zu verwirklichen, gibt es mehrere Möglichkeiten (Abbildung 2.52):

- die Schaltbedingungen für die Flipflops passend aufstellen und die Zählnetzwerke entsprechend entwerfen (State-Machine-Entwurf),
- die letzte gewünschte Stellung decodieren und den Zähler zurücksetzen,
- den Zähler entsprechend voreinstellen, so daß er nicht von Null an, sondern von einem Anfangswert $a = 2^n - z$ zählt, wobei z die gewünschte Zählweite angibt.

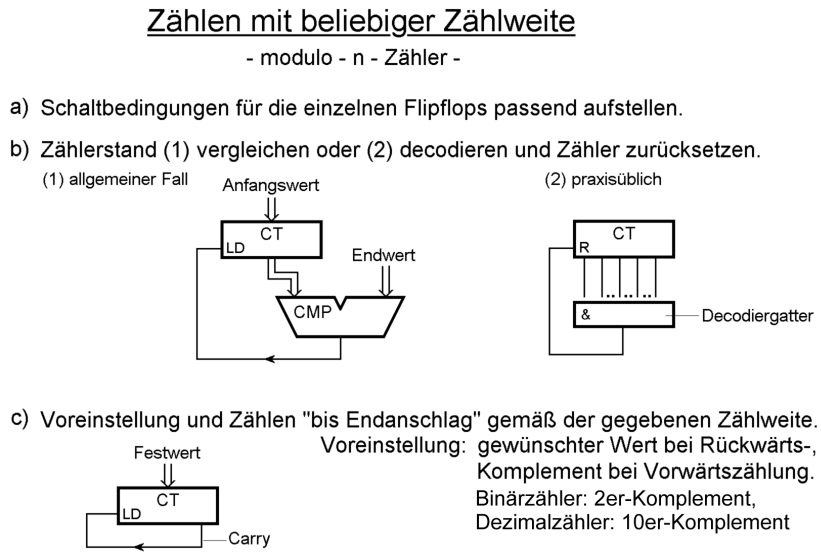


Abbildung 2.52 Zählen mit beliebiger Zählweite

Dezimalzähler
Dezimalzähler zählen in den einzelnen Dezimalstellen von 0 bis 9. Die typische Codierung: BCD. Demgemäß wird binär gezählt: 0000 - 0001 - 0010 usw. bis 1001.

Decodieren von Zählerständen
Oft braucht man Signale, die bestimmten Zählerstellungen entsprechen. Nur beim Ringzähler sind diese durch die Flipflop-Ausgänge direkt gegeben. Ansonsten müssen den Flipflops Decodierschaltungen nachgesetzt werden. Das Problem: Wir erhalten nicht immer saubere Signale. Vielmehr ist, wenn mehrere Flipflops gleichzeitig schalten, durch die unvermeidlichen Unterschiede in den Schaltzeiten mit Störungen (Glitches, Spikes) zu rechnen. Wenn man die Ausgänge der Decodierschaltung zum Umschaltzeitpunkt nicht auswertet, schaden die Störungen nicht.

Typische Zählerschaltkreise: die Dezimal- und Binärzähler '160...'163
Diese Zählerschaltkreise (Abbildung 2.53) arbeiten vollsynchron. Sie können nur vorwärts zählen. Es dürfen ständig Taktimpulse anliegen; die jeweilige Funktion kommt durch Erregen der entsprechenden Steuereingänge zustande. Besonderheiten:

- bei gleichzeitiger Erregung: Laden hat Vorrang vor Zählen, Rücksetzen hat Vorrang vor Laden.
- Zählerlaubnis: Damit der Schaltkreis zählt, müssen beide Erlaubniseingänge ENT, ENP aktiv sein.
- Kaskadierung: Der Schaltkreis liefert einen Ausgangsübertrag RCO (Ripple Carry Out). Dies ist eine rein kombinatorische Decodierung des Zähler-Endstandes. Störimpulse (Glitches, Spikes) auf RCO sind also nicht notwendigerweise Fehleranzeichen! Längere Zähler baut man auf einfachste Weise, indem man RCO an ENP und ENT des Nachfolge-Schaltkreises anschließt.
- Rücksetzen: Bei den Typen mit synchronem Rücksetzeingang ('161, '163) ist zum Rücksetzen ein Taktimpuls erforderlich. Mit anderen Worten: das Rücksetzsignal muß so lange aktiv bleiben, bis wenigstens ein Taktimpuls wirksam geworden ist.

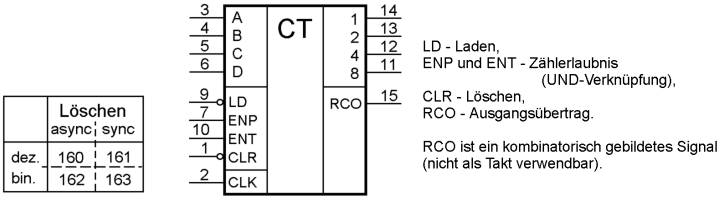


Abbildung 2.53 Typische Zählerschaltkreise

Es folgen zwei typische Einsatzfälle dieser Schaltkreise.

1. Schnelle Kaskadierung
Die Schaltung (Abbildung 2.54) zeigt, weshalb zwei Zählerlaubniseingänge (ENP, ENT) vorgesehen sind.

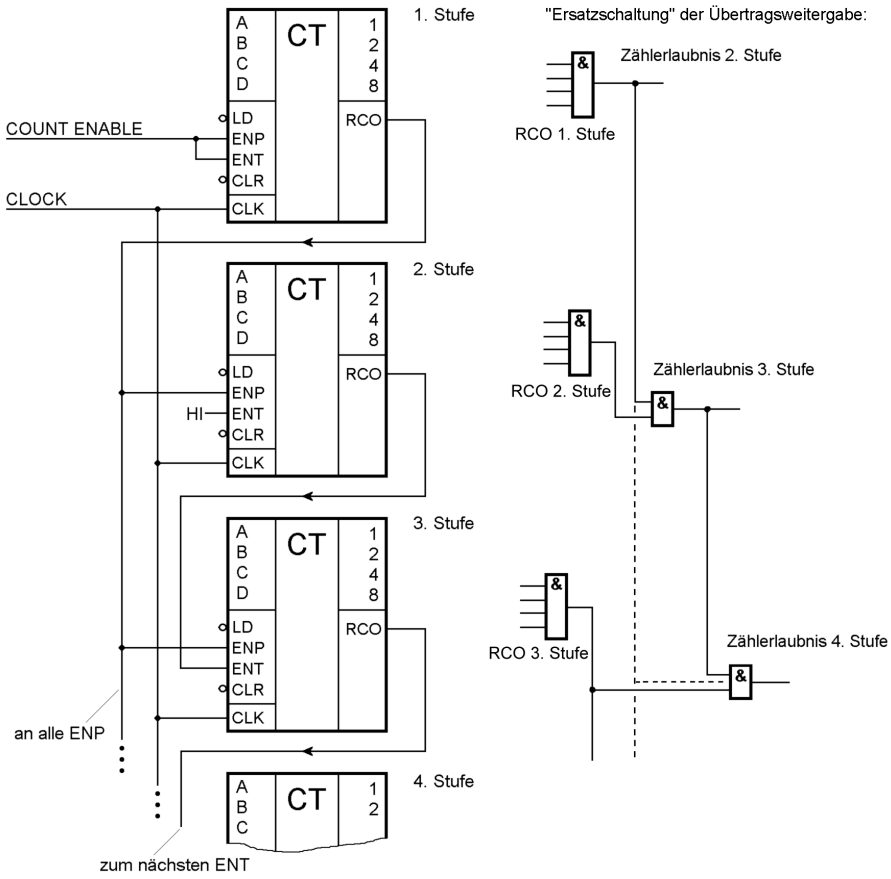


Abbildung 2.54 Schnelle Kaskadierung mit '160...'163

Erklärung:
Eine nachgeschaltete Stufe darf dann zählen, wenn alle vorgeschalteten Stufen ihren jeweiligen Endwert erreicht haben (1111B beim binären, 1001B beim dezimalen Zählen). Die Zählerlaubnis einer nachfolgenden Stufe ist also eine entsprechend breite UND-Verknüpfung der Flipflopausgänge aller vorherigen Stufen. Beim Aufbau des Zählers aus mehreren

Schaltkreisen werden diese UND-Verknüpfungen durch Kaskadierung gebildet (in Abbildung 2.54 rechts). Es ist ersichtlich, daß mit zunehmender Stufenzahl immer mehr Gatter nacheinander durchlaufen werden müssen. Dies begrenzt die Zählfrequenz (da, wenn der nächste Zähltakt kommt, alle Stufen wissen müssen, ob sie zählen sollen oder nicht).

Der Ausweg: wir machen davon Gebrauch, daß von Stufe zu Stufe eigentlich immer mehr Zeit zur Verfügung steht, weil immer seltener gezählt wird. Bleiben wir beim Binärzähler: die erste Stufe erreicht alle 16 Takte ihren Endwert, die zweite Stufe alle 256 Takte, die dritte alle 4096 Takte usw. Alle Stufen erhalten über ENP den Ausgangsübertrag der ersten zugeführt. Von der ersten zur zweiten Stufe können wir nichts tun. Deshalb wird an der zweiten Stufe ENT fest aktiviert. Von der dritten Stufe an wirkt der Übertragsausgang der jeweils vorgeordneten Stufe auf ENT. Der Erfolg: Wenn die erste Stufe einen Ausgangsübertrag abgibt, wird sofort die Zählerlaubnis aller anderen Stufen parallel erregt (vgl. die gestrichelten Verbindungen in Abbildung 2.54 rechts); es ist nicht mehr so, daß sich ein Übertrag von der ersten Stufe über alle Stufen hinweg ausbreiten muß. Von der zweiten Stufe an steht genügend Zeit zur Verfügung, um den Ausgangsübertrag über die RCO-ENT-Verschaltung durch alle Stufen zu leiten (beim Binärzähler 16, beim Dezimalzähler 10 Taktperioden).

2. Zählen mit einstellbarer Zählweite
Abbildung 2.55 zeigt ein entsprechendes Beispiel.

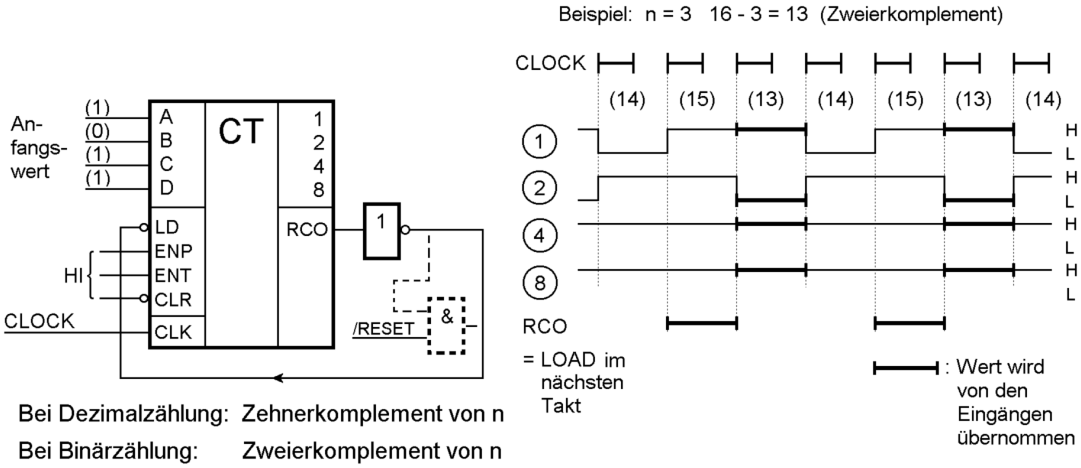


Abbildung 2.55 Einstellbarer Zähler (modulo-n-Zähler) mit '160...'163

Erklärung:
Der Zähler zählt vorwärts modulo 3 (Zählfolge: 13 - 14 - 15 - 13 usw.). Ist die Endstellung (hier: 15) erreicht, so wird über RCO das erneute Laden mit dem Anfangswert veranlaßt.

Hinweise:

1. Anfangswert = Zweierkomplement der Zählweite.
2. Damit alles richtig funktioniert, ist ggf. auch beim Rücksetzen der Anfangswert zu laden - vgl. das gestrichelt gezeichnete UND-Gatter.

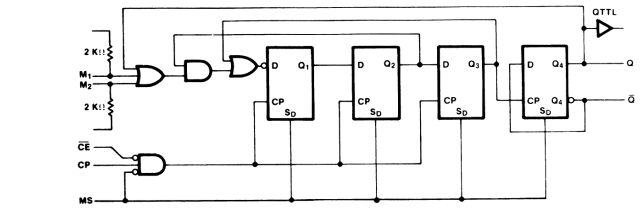
2.5.4. Teiler

Ein üblicher Teiler gibt nach n Eingangsimpulsen einen Ausgangsimpuls ab; er untersetzt die Impulsfolgefrequenz im Verhältnis n:1. Einfache Teiler werden vorzugsweise als Asynchronzähler aufgebaut. Die Vorteile: (1) schnellstes Schalten, da keine kombinatorischen Netzwerke durchlaufen werden müssen, (2) nur an das erste Flipflop werden die höchsten Geschwindigkeits-Anforderungen gestellt, das zweite wird bereits mit der halben Frequenz betrieben usw. (die Anforderungen an die Taktfrequenz der Flipflops halbieren sich mit jeder Stufe).

Vorteiler (Prescaler)
Wenn man Impulse sehr hoher Frequenz zu zählen hat, ist es möglich, durch einen vorgeschalteten Frequenzteiler die Zählfrequenz des eigentlichen Zählers in eine beherrschbare Größenordnung zu bringen. Der wichtigste Anwendungsfall: die Frequenzsynthese in Hochfrequenzschaltungen. Beim UKW-Rundfunk geht es um etwa 100 MHz, beim Fernsehen und beim Mobiltelefon um einige hundert MHz bis hinein in den GHz-Bereich. Dafür werden Teiler-Flipflops und Vorteiler-Schaltkreise (Prescaler) in besonders "schnellen" Technologien (wie ECL und GaAs) gefertigt (in ECL kommt man von 650...750 MHz bis auf ca. 3 GHz, in GaAs auf über 5 GHz; vgl. auch Abschnitt 3.8.).

Genaues Zählen trotz schneller Vorteilung: Pulse Swallowing
In den Typenlisten der Hersteller finden wir Vorteiler, die - über einen Eingang steuerbar - wahlweise durch 5 oder 6, durch 10 oder 11, durch 128 oder 129 usw. teilen können. Abbildung 2.56 zeigt die Innenschaltung eines solchen Schaltkreises.

Erklärung zu Abbildung 2.56:
Der 11C90 teilt durch 10 oder 11. Er ist in einer TTL-Umgebung einsetzbar. Beachten Sie, daß man hier die Zählfunktion durch "trickreiche" Zustandsübergänge verwirklicht hat (es geht darum, einerseits mit wenig Flipflops und andererseits mit geringer Schaltungstiefe der Rückführungen auszukommen).



Zählfolge:

	Q_1	Q_2	Q_3	Q_4(QTTL)	
	H	H	H	H	÷11
÷10	L	H	H	H	
	L	L	H	H	
	L	L	L	H	
	H	L	L	H	
	H	H	L	H	
	L	H	H	L	
	L	L	H	L	
	L	L	L	L	
	H	L	L	L	
	H	H	L	L	

Funktionstabelle:

Eingänge				Ausgang
MS	$\overline{CE}$	M_1	M_2	
H	X	X	X	Set HIGH
L	H	X	X	Hold
L	L	L	L	÷11
L	L	H	X	÷10
L	L	X	H	÷10

H = HIGH Voltage Level
L = LOW Voltage Level
X = Don't Care

Abbildung 2.56 650-MHz-Vorteiler in ECL-Technologie: der 11C90 (National Semiconductor)

Wozu ist die Umschaltbarkeit gut? - Man kann damit - trotz Vorteilung - auch bei höchsten Impulsfolgefrequenzen mit einem vergleichsweise langsamen Zähler bis auf den einzelnen Impuls genau zählen (Abbildung 2.57).

Erklärung:
1 - Vorteiler (Prescaler); 2 - Steuerzähler; 3 - Dezimalzähler (hier aus '160er-Schaltkreisen "nach Kochbuch" aufgebaut); 4, 5 - hier liegt das eingestellte Teilerverhältnis an*); 6 - Umsteuerung 1:10/1:11; 7 - Trickschaltung mit NAND, die es ermöglicht, anstelle des Zehnerkomplements das einfacher zu bildende Neunerkomplement anzulegen; LD - das Ausgangssignal dient zu-gleich als Ladeimpuls.

*): alle Dateneingänge sind entsprechend beschaltet. 4 - niedrigstwertiges; 5 - höchstwertiges Bit (LSB, MSB).

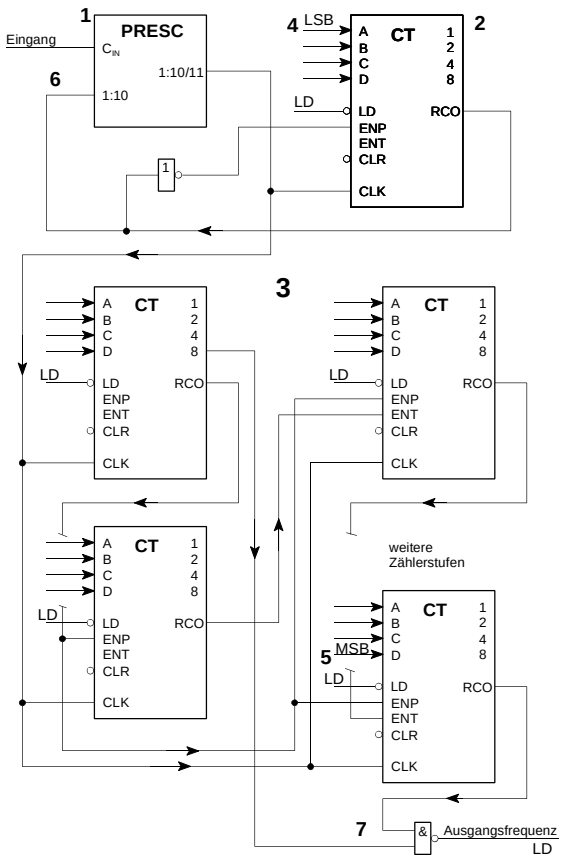


Abbildung 2.57 Einstellbarer schneller Frequenzteiler. Alle ungenutzten Anschlüsse sind hier unbeschaltet dargestellt

Die zu zählenden Impulse gelangen auf den Vorteiler 1, der zunächst durch 11 teilt. Jeder 11. Impuls schaltet also den Steuerzähler 2 weiter. Hat er seinen Endwert erreicht, setzt er sich selbst über die Rückführung in Ruhestellung und schaltet den Vorteiler auf 1:10-Teilung um. Dann wird von der zweiten Dezimalstelle "ganz normal" jeder 10. Impuls gezählt. Hat der gesamte Zähler den Endwert erreicht, wird der Steuerzähler 2 wieder aktiviert, der gesamte Zähler mit dem eingestellten Teilerverhältnis geladen und der Vorteiler 1 auf 1:11 umgestellt. Dann beginnt das Spiel von neuem.

Funktioniert das auch richtig? - Sei der Steuerzähler zum Zählen von a Impulsen voreingestellt, der "restliche" Dezimalzähler zum Zählen von b Impulsen. Insgesamt müssen also, gemäß der Einstellung, a + 10b Takte abgezählt werden (Stellenwertsystem).

Zunächst werden 11 · a Impulse gezählt. Dann schaltet der Steuerzähler 2 auf 1:10- Teilung um. Der Inhalt des restlichen Dezimalzählers 3 ist aber - da dieser ebenfalls vom Vorteiler angesteuert wird - heruntergezählt worden, und zwar mit a (1:11 geteilten) Zählimpulsen. Bis zum Ende sind also noch b-a Zählimpulse erforderlich. Der verbleibende Zählerinhalt wird alle 10 Eingangsimpulse verändert. Insgesamt werden also gezählt:

11a Impulse
+
10 (b-a) Impulse.

Das ergibt 11a + 10b - 10a = a + 10b Impulse, also genau so viel, wie es die exakte Zählung erfordert.

Durch diesen Trick erhält man einen exakt einstellbaren Frequenzteiler mit einer Eingangsfrequenz von beispielsweise ca. 650 MHz, kommt aber mit viel langsameren Zählerschaltkreisen aus (Zählfrequenz = $^1/_{10}$ der Eingangsfrequenz).

2.6. Reguläre Steuerschaltungen (State Machines)

2.6.1. Grundlagen

Steuerschaltungen erzeugen Ausgangssignale, die im allgemeinen Fall (1) von den Eingangssignalen und (2) vom aktuellen Zustand (State) abhängen. Die State Machine*) ist die technische Verwirklichung des Konzepts vom abstrakten Automaten. Ein abstrakter Automat ist eine ganz allgemeine Modellvorstellung, die alles umfaßt, was man irgendwie mit dem Begriff des automatischen (selbsttätigen) Arbeitens in Verbindung bringen kann. Ein solcher Automat wird durch drei Mengen und zwei Funktionen gekennzeichnet:

Die Mengen:

- $E = \{e_1, e_2,...\}$: Menge aller Eingänge,
- $A = \{a_1, a_2,...\}$: Menge aller Ausgänge,
- $Z = \{z_1, z_2,...\}$: Menge aller Zustände.

*): in der Literatur gelegentlich auch FSM (= Finite State Machine) bezeichnet ("finit" deshalb, weil die Anzahl der Zustände endlich ist).

Zeitdiskrete Arbeitsweise
Der Automat arbeitet zu diskreten Zeitpunkten. Es gibt also Zeitpunkte t (= der aktuelle Zeitpunkt), t+1 (= der darauf folgende Zeitpunkt) usw. (t-1, t-2 usw. bezeichnen vorhergegangene Zeitpunkte, also gleichsam die Vergangenheit.) Wie diese Zeitpunkte definiert werden (durch Taktflanken, durch das Erkennen von Änderungen usw.), ist der Theorie gleichgültig. Ebenso kennt die Theorie keine Flankensteilheiten, keine Metastabilität, keinen Zeitversatz (Skew) usw. Zu jedem diskreten Zeitpunkt t gibt es eine zugehörige Eingangsbelegung e(t), eine Ausgangsbelegung a(t) und einen aktuellen Zustand $z(t) \in Z$.

Die Funktionen:

- Überführungsfunktion. Sie beschreibt, welchen Zustand der Automat im nächsten diskreten Zeitpunkt (t+1) einnimmt (Folgezustand): z (t + 1) = f (irgendwas),
- Ausgabefunktion. Sie beschreibt, welche Ausgangsbelegung der Automat abgibt: a(t) = g (irgendwas).

Das "irgendwas" im Funktionsausdruck bestimmt den Unterschied: demgemäß gibt es zwei Grundtypen von Automaten (Abbildung 2.58).

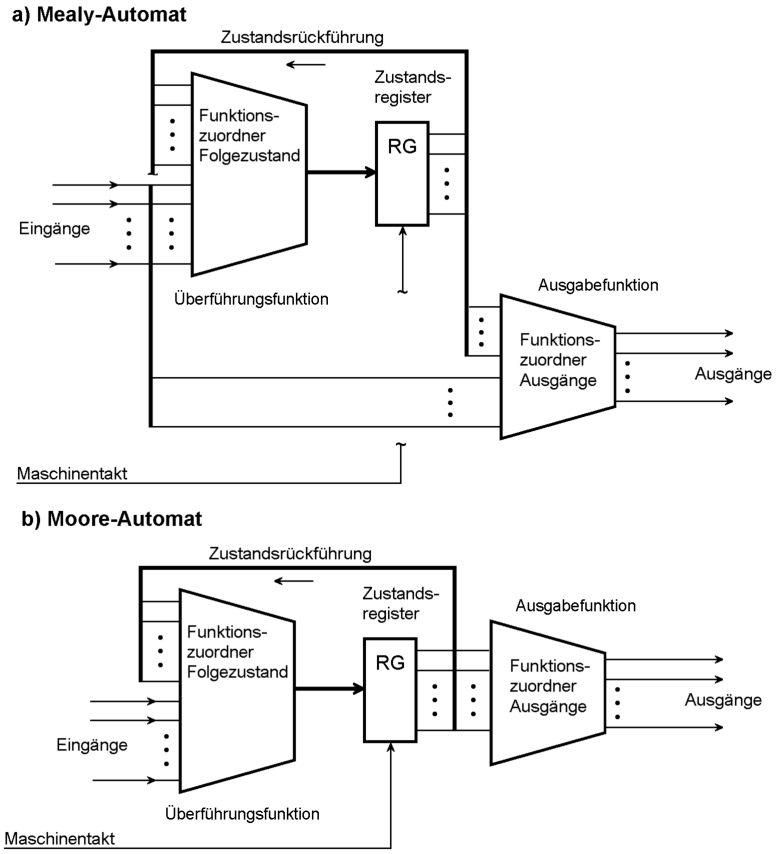


Abbildung 2.58 Mealy- und Moore-Automaten

Erklärung:

a) *Mealy-Automat.*

Überführungsfunktion: $z(t+1) = f(z(t), e(t))$

Der Folgezustand hängt ab vom aktuellen Zustand und von der aktuellen Eingangsbelegung.

Ausgabefunktion: $a(t) = g(z(t), e(t))$

Die Ausgangsbelegung hängt ab vom aktuellen Zustand und von der aktuellen Eingangsbelegung.

b) *Moore-Automat.*

Überführungsfunktion: $z(t+1) = f(z(t), e(t))$

Der Folgezustand hängt ab vom aktuellen Zustand und von der aktuellen Eingangsbelegung.

Ausgabefunktion: $a(t) = g(z(t))$

Die Ausgangsbelegung hängt nur vom aktuellen Zustand ab.

Hinweise:

1. Beim Mealy-Automaten folgt im aktuellen Zustand die Ausgangsbelegung den Änderungen der Eingangsbelegung nach; Änderungen an den Eingängen können noch im aktuellen Zustand an den Ausgängen wirksam werden. Das ist manchmal erwünscht und manchmal nicht (nämlich dann, wenn das Schalten der Eingangssignale ausgangsseitige Störimpulse (Spikes, Glitches) hervorruft).
2. Beim Moore-Automaten hingegen muß sich erst der Zustand ändern, ehe sich die Ausgangsbelegung ändern kann. Änderungen an den Eingängen werden somit erst mit dem nächsten Zustandsübergang an den Ausgängen wirksam (also mit dem nächsten Taktzyklus = 1 Takt später als beim Mealy-Automaten).
3. Jeder Mealy-Automat kann in einen Moore-Automaten mit gleichem Verhalten umgewandelt werden (von der Verzögerung der Ausgangserregung um 1 Taktzyklus abgesehen).
4. Die Theorie kennt noch einen weiteren Automatentyp: den Semiautomaten (Medwedjew-Automaten). Er hat keine Ausgabefunktion. Die Überführungsfunktion: z (t+1) = f (z(t), e(t)). Dieses Automatenmodell wird angewendet, wenn nur die Zustandsübergänge von Interesse sind.

Vollsynchrone Automaten
Bei vollsynchroner Arbeitsweise werden auch die Ausgangssignale über Register geführt (Abbildung 2.59). Damit verzögert sich die Ausgangsbelegung um jeweils einen Taktzyklus.

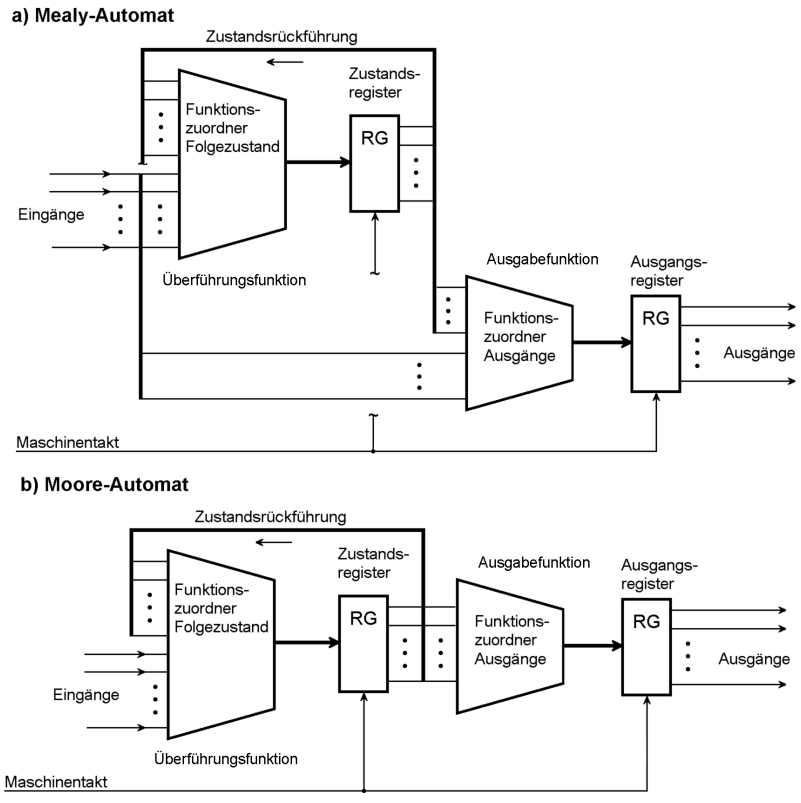


Abbildung 2.59 Vollsynchrone Automaten

Hinweise:

1. Beim vollsynchronen Mealy-Automat können Signaländerungen an den Eingängen nicht zu Störimpulsen an den Ausgängen führen.
2. Beim Moore-Automaten erfordert jede Ausgangsänderung erst eine Zustandsänderung. Solche Automaten können also erst im übernächsten Takt (t+2) auf eine Eingangsbelegung zum Zeitpunkt t reagieren.
3. Welchem Automatentyp entspricht die State Machines der folgenden Abbildung 2.60? Neuer Zustand und Ausgangsbelegung ergeben sich offensichtlich mit demselben Takt, also handelt es sich um Moore-Automaten gemäß Abbildung 2.58b, bei denen die Ausgangsbelegung gleichzeitig mit dem Folgezustand gebildet wird (eine gelegentlich verwendete Praxislösung).

2.6.2. Zustandscodierung

Wenn man n Maschinenzustände (States) vorsehen muß - wieviele Flipflops werden dazu benötigt und wie ordnet man diese den einzelnen Zuständen zu? Dies ist das Problem der Zustandscodierung. Grundsätzlich gibt es eine unabsehbare Vielfalt von Codierungsmöglichkeiten (auch sehr trickreiche), aber nur wenige haben in der Praxis weite Verbreitung gefunden:

Binärcodierung
n Zustände werden mit ceil (ld n) Flipflops binär codiert. Dies ergibt die geringste Flipflopanzahl, erfordert aber oft einen komplexen Funktionszuordner (viele Gatter, hohe Schaltungstiefe).

1-aus-n-Codierung (OHE)
Ganz einfach: jeder Zustand wird durch ein Flipflop repräsentiert. Man braucht n Flipflops für n Zustände, wobei jeweils nur ein Flipflop aktiviert ist (im Fachenglisch spricht man auch vom *One-Hot Encoding*, OHE). Das erfordert vergleichsweise viele Flipflops. Dafür wird die Zuordner-Kombinatorik einfach und schnell.

Gemischte Codierung
Manchmal sind bestimmte Codierungen in der Schaltungspraxis offensichtlich nicht anwendbar. Einen 20-Bit-Zähler (über eine Million Zustände!) wird man kaum mit 1-aus-n-Codierung verwirklichen. So liegt es nahe, Zustände, die durch Zählprozesse aufeinanderfolgen, binär zu codieren und solche, zwischen denen kompliziertere Übergänge vorgesehen sind, im 1-aus-n-Code. Ein weiteres Beispiel ist die Abwandlung des 1-aus-n-Codes dahin, daß der "Ruhezustand" (Idle State) nicht durch ein gesondertes Flipflop, sondern durch die Nullbelegung aller verbleibenden Flipflops codiert wird.

Auswahl
Ein guter Schaltungsentwickler strebt minimale Kosten im Rahmen der gegebenen Technologie an. "Akademische" Minimalkriterien (etwa aus der Schaltalgebra und Automatentheorie) sind dabei nur bedingt hilfreich. So ist es ein wichtiger Gesichtspunkt, ob man die State Machine noch in einem bestimmten CPLD, FPGA oder Gate Array unterbringt. Bei Nutzung von FPGA- und ASIC-Technologien liegt es nahe, die 1-aus-n-Codierung zu bevorzugen: an Flipflops herrscht kaum Mangel, aber komplizierte Verknüpfungsnetzwerke fressen Siliziumfläche bzw. sind (im vorgegebenen Schaltkreis) gar nicht realisierbar. Hingegen ist bei PROM-Realisierung

(Abbildung 2.60) die binäre (minimale) Zustandscodierung praktisch unbedingt erforderlich, um die Anforderungen an die Speicherkapazität zu reduzieren (ein Bit weniger, und wir kommen mit einem halb so großen PROM aus!). Die Kompliziertheit der kombinatorischen Verknüpfungen spielt hingegen beim PROM keine Rolle.

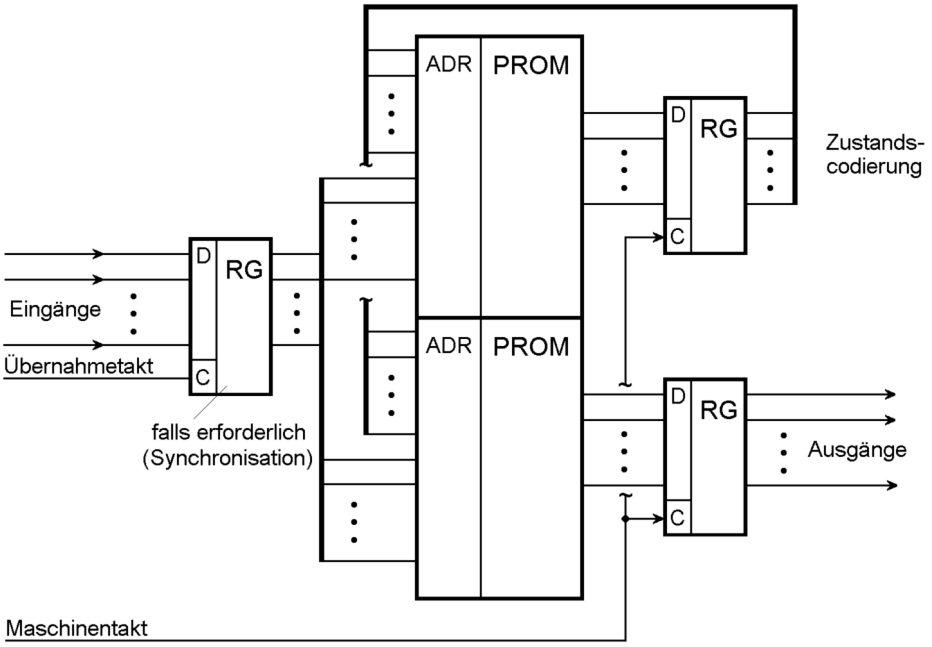


Abbildung 2.60 State Machine mit PROM-Zuordnern

Erklärung:
Die Funktionszuordner werden oft mit programmierbaren Schaltkreisen realisiert. Wenn es nicht auf die letzte Nanosekunde ankommt, liegt es nahe, die Funktionszuordner mit (PROM-) Speicherschaltkreisen aufzubauen. Die Vorteile: unbeschränkte funktionelle Komplexität (auch komplizierteste Zustandsübergänge kosten keinen Mehraufwand), schaltungstechnische Einfachheit, leichte Änderbarkeit.

2.6.3. Schaltungsbeispiele

Charakteristische Zustandsübergänge
Viele State Machines sind an sich gar nicht besonders kompliziert; sie beruhen auf einfachen, überschaubaren Mustern der Zustandsübergänge (Abbildungen 2.61 bis 2.65).

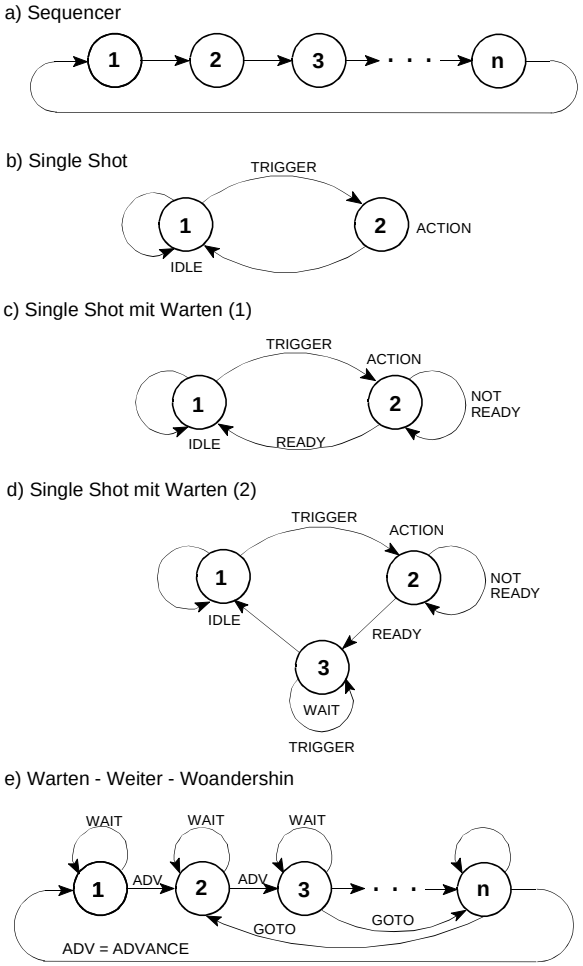


Abbildung 2.61 Zustandsgraphen einfacher State Machines (Auswahl)

Sequencer
Sequencer sind State Machines mit besonders einfachen Zustandsübergängen. Der einfachste Fall: das Erzeugen zeitstarrer Signalfolgen durch taktweises Weiterschalten von Zustand zu Zustand. Typische Schaltungslösungen: (1) Schieberegister (Abbildungen 2.62, 2.63), (2) Zähler, ggf. mit nachgeschalteten Vergleichs- oder Decodiernetzwerken, (3) mit dem jeweiligen Bitmuster geladene Speicher, die zyklisch adressiert werden.

Single Shot
Ein Auslösesignal (TRIGGER) veranlaßt, daß im nächsten Taktzyklus eine Steuerwirkung (ACTION) ausgeübt wird.

Single Shot mit Warten (1)
Der mit der Steuerwirkung (ACTION) verbundene Zustand wird solange beibehalten, bis eine Fertigmeldung (READY) aktiv wird.

Single Shot mit Warten (2)
Hier wird der Fall berücksichtigt, daß das Auslösesignal (TRIGGER) länger als einen Taktzyklus aktiv sein kann. In diesem Fall soll das Steuersignal nicht erneut aktiviert werden, sondern es wird zunächst gewartet (im Zustand WAIT), bis das Auslösesignal wieder inaktiv geworden ist.

Warten - Weiter - Woandershin
Von jedem beliebigen Zustand aus gibt es höchstens 3 weitere Zustände, die als Folgezustände in Frage kommen:

- derselbe Zustand (Warten (WAIT)),
- der nächste Zustand (Weiter (ADVANCE)),
- ein x-beliebiger anderer Zustand (Woandershin (GOTO)).

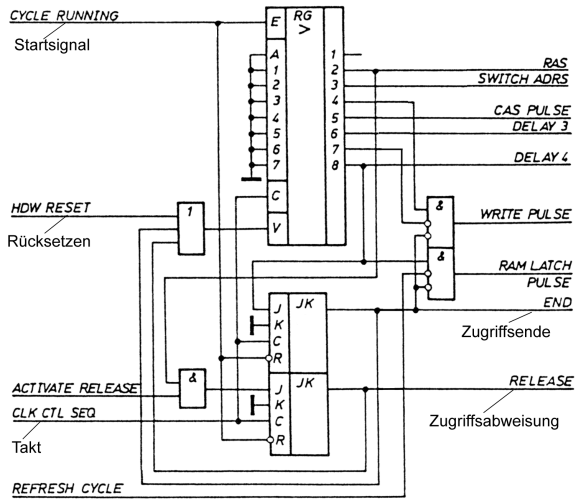


Abbildung 2.62 Sequencer (Praxisbeispiel). Erklärung im Anschluß an Abbildung 2.63

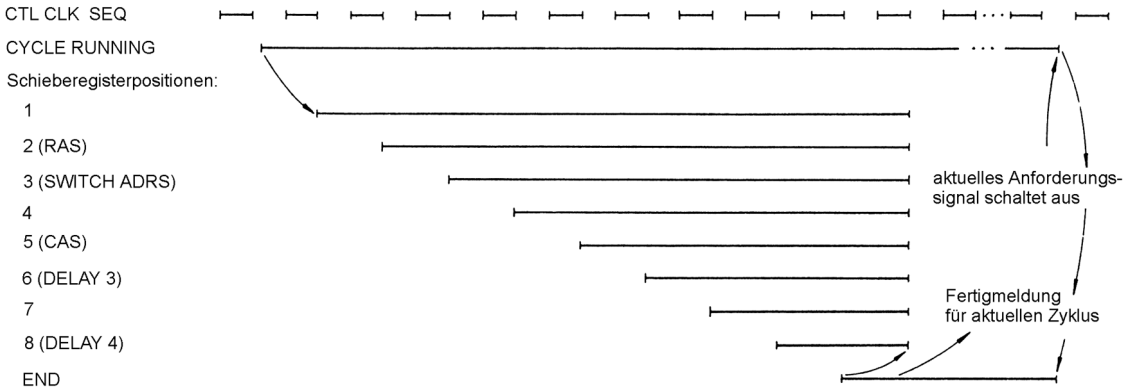


Abbildung 2.63 Ablauf eines DRAM-Zugriffs

Erklärung zu Abbildung 2.63:
Elementare DRAM-Subsysteme erfordern zeitversetzte Impulse (RAS, CAS usw.). Es liegt nahe, diese mit einem Schieberegister zu erzeugen. Die Abbildungen zeigen eine Praxisschaltung (lassen Sie sich von - hier unwesentlichen - Einzelheiten nicht ablenken). Das Schieberegister enthält im Ruhezustand Nullen. Mit Zyklusbeginn (Startsignal CYCLE RUNNING) wird eine Eins durchgeschoben. Aus den zeitversetzten Ausgangssignalen werden die benötigten Steuerimpulse gebildet. Am Ende (END oder RELEASE) wird das Schieberegister parallel mit Nullen geladen. Das jeweilige Endesignal bleibt so lange aktiv, bis das Startsignal inaktiv geworden ist (Handshaking mit vorgeordneten Steuerschaltungen). END kennzeichnet einen erfolgreichen Speicherzugriff. RELEASE wird aktiviert, wenn - aus anderweitigen Gründen - der Zugang zum Speicher zeitweilig blockiert ist. Siehe weiterhin - als einfacheres Beispiel für das gleiche Einsatzgebiet - Aufgabe 16 in Kapitel 8.

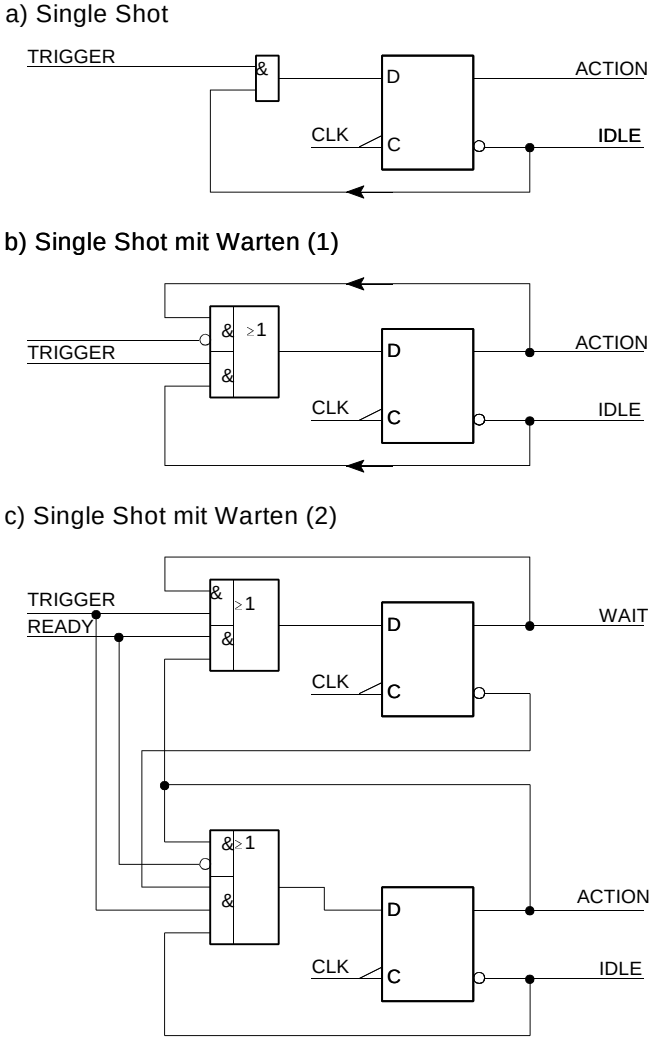


Abbildung 2.64 Einfache Single-Shot-Schaltungen

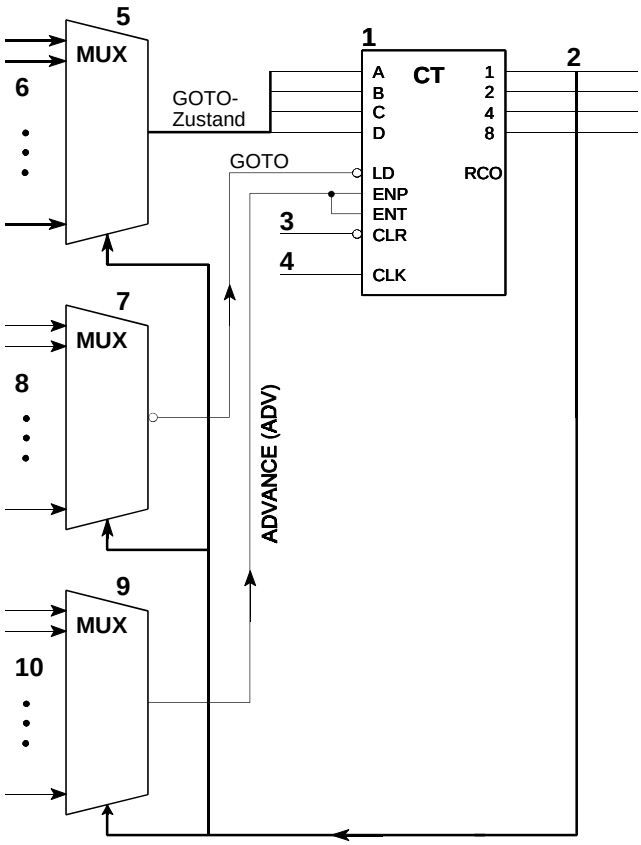


Abbildung 2.65 Warten - Weiter - Woandershin: eine ziemlich universelle State Machine

Erklärung:
1 - Zustandsregister (= Binärzähler); 2 - binär codierter Maschinenzustand; 3 - Löscheingang; 4 - Maschinentakt; 5 - Multiplexer für Zustandsverzweigung (GOTO-Folgezustand); 6 - Festwerte der Zustandsverzweigung; 7 - Multiplexer für Ausführung der Zustandsverzweigung; 8 - Eingangssignale, die Zustandsverzweigungen veranlassen können; 9 - Multiplexer für nächsten Zustand (ADV-Folgezustand; durch Weiterzählen erreicht); 10 - Eingangssignale, die Zustandsübergänge veranlassen können. Im Beispiel kann die State Machine bis zu 16 Zustände haben. Die jeweiligen Zustandsübergänge werden durch Beschaltung der Multiplexer-Eingänge 6, 8, 10 realisiert. Abbildung 2.66 veranschaulicht dies anhand eines kleinen Beispiels.

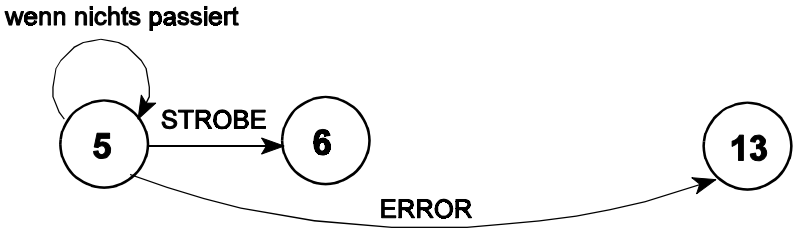


Abbildung 2.66 Beispiel für Zustandsübergänge

Erklärung zu Abbildung 2.66:
STROBE und ERROR sind Interfacesignale. Unsere State Machine soll darauf reagieren. Wir befinden uns in Zustand 5. Die Eingänge Nr. 5 der Multiplexer 5 sind mit dem Festwert 13 zu beschalten (binär 1101), um den Übergang in den Zustand 13 zu ermöglichen. Dieser Übergang soll stattfinden, wenn das Signal ERROR aktiv ist. Also ist Eingang Nr. 5 des Multiplexers 7 mit ERROR zu beschalten. Im Zustand Nr. 5 wird somit das ERROR-Signal zum Ladeeingang des Zählers durchgeschaltet. Ist ERROR aktiv, wird der Zähler über die Multiplexer 5 mit dem Festwert 13 geladen. Der Übergang in den Zustand 6 (= Weiterzählen) soll stattfinden, wenn das Signal STROBE aktiv ist. Also ist STROBE an den Eingang Nr. 5 des Multiplexers 9 anzuschließen. STROBE wirkt somit in Zustand 5 als Zählerlaubnis. Sind weder ERROR noch STROBE aktiv, so ist auch keiner der Steuereingänge des Zählers erregt; es passiert nichts (Zähler verharrt weiterhin in Zustand 5).

Hinweis:
Das anhand der Abbildungen 2.61e, 2.65 und 2.66 veranschaulichte Schema eignet sich u. a. dazu, viele typische Abläufe an Interfaces zu steuern. Anders herum gesehen: die Signalprotokolle mancher Interfaces werden eigens so gestaltet, daß sie diesem Schema entsprechen.

State Machines in moderner Hochleistungshardware
Die Schaltungen sind ziemlich kompliziert. Die Darstellung als detaillierter Schaltplan wäre viel zu unübersichtlich. Man entwirft deshalb die Schaltung unmittelbar als State Machine und dokumentiert sie über Zustandsgraphen und Schaltgleichungen. Diese Beschreibung wird dann in CPLDs, FPGAs oder ASCIs verwirklicht (einen Schaltplan braucht man gar nicht, weil die Entwurfssoftware aus den Gleichungen heraus unmittelbar die Programmier- oder Fertigungsdaten ableitet).