Kapitel 6
Digitale Modulationsverfahren

Ausgabestand 1.1

- Nur zur Information -

6. Digitale Modulationsverfahren

6.1. Grundlagen

Als "Modulationsverfahren" bezeichnen wir hier digitale Signalcodierungen, die zum Übertragen bzw. Speichern beliebiger Bitfolgen vorgesehen sind. Das Szenarium (Abbildung 6.1): Wir haben einen bitseriellen Informationskanal (hierbei kann es sich um einen Übertragungsweg, aber auch um ein magnetisches oder optisches Speichermedium handeln), und wir wollen beliebige Folgen von Nutz-Bits in diesen Kanal (1) hineingeben und (2) unverfälscht wieder herausbekommen. Unsere Probleme:

- Modulation und Signalformung: die Datendarstellung im seriellen Informationskanal muß an die jeweiligen physikalischen Gegebenheiten angepaßt sein (Stichworte: Signalübertragung gegen Masse, differentielle Signalübertragung, optische Signalübertragung, magnetische Speicherung (Elementarmagnete), optische Speicherung (Pits und Lands),
- Serialisierung und Deserialisierung: die Daten werden abschnittsweise parallel geliefert (z. B. als Bytes oder als 32-Bit-Worte) und sollen auch so wieder ausgegeben werden,
- Zwischenpufferung: die Datenlieferanten und Datenempfänger sind typischerweise programmierbare Einrichtungen, z. B. der Prozessor im PC oder der Mikrocontroller in einem Laufwerk - und die haben ihre Eigenheiten (z. B. veränderliche, unvorhersagbare Latenzzeiten),
- Taktrückgewinnung: beim Senden bzw. Schreiben ist der Zeitbezug noch gewährleistet (üblicherweise durch entsprechende Taktierung). Beim Empfangen bzw. Lesen müssen wir den Zeitbezug jedoch aus dem Datenstrom wiederherstellen.
- Synchronisation: dem Empfänger muß zunächst etwas Zeit gegeben werden, um sich auf den ankommenden Datenstrom einzustellen. Zudem muß er erkennen - und zwar bis aufs Bit genau - wann die eigentliche Informationsübertragung beginnt.

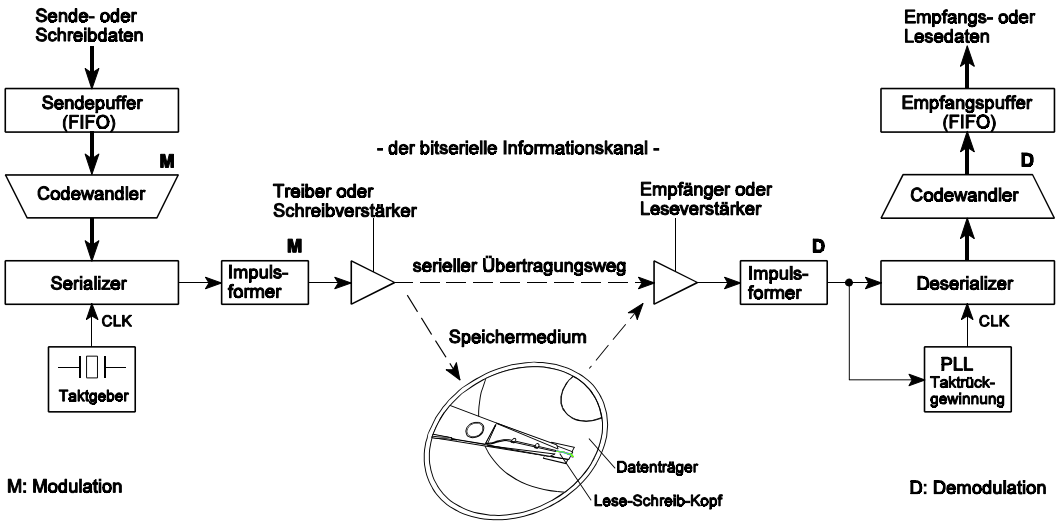

Abbildung 6.1 Zu den Anwendungsgebieten digitaler Modulationsverfahren

Hinweise:

1. Wir betrachten hier nur die Umsetzung von Bitfolgen, also gleichsam die unterste Ebene der Informationsdarstellung. Auf übergeordnete Ebenen, wie Sektor- und Spurformate, Informationsblöcke (Frames), Pakete usw. wollen wir nicht eingehen.
2. Wir beschränken uns auf nur jene Verfahren, die im PC-Bereich besonders wichtig sind (Tabelle 6.1).
3. Wir werden im folgenden ausschließlich binäre Signale darstellen (also Impulse) und von tatsächlichen Signalformen absehen, wie sie beispielsweise auf Netzwerkkabeln oder an Lese-Schreib-Köpfen von Plattenspeichern gemessen werden können (idealisierte Signaldarstellung). Wir verwenden dabei die positive Logik (Low = 0; High = 1).

Verfahren	Abkürzung	Taktrückgewinnung	Bedeutung und Anwendungsbeipiele
Return to Zero	RZ	-	einfachstes Prinzip der Signalisierung mit Impulsen (1 = Impuls, 0 = kein Impuls)
Non Return to Zero	NRZ	quarzstabiler Takt; Start/Stop-Übertragung	übliche Grundlage der Informationsdarstellung in der (getaktet arbeitenden) Hardware. Ausgangs-Darstellung für weitere Signal- wandlungen. Serielle Schnittstellen, IEEE 1394
Non Return to Zero and Inversion	NRZI	bei mehreren parallelen Spuren (Magnetband) selbsttaktend*); sonst: PLL	herkömmliche Magnetbandsysteme, herkömmliche synchrone Schnittstellen (z. B. IBMs BSC-Protokoll), USB
Frequenzmodulation	FM	selbsttaktend*); PLL	einfaches Verfahren; vor Jahrzehnten für Platten- und Diskettenlaufwerke genutzt. Veraltet; dient hier nur zu Lehrzwecken
Modifizierte Frequenzmodulation	MFM	PLL	verdoppelte Datenrate/Aufzeichnungsdichte gegenüber FM. Disketten, (sehr) alte Festplatten
Gruppencodes, insbesondere RLL-Codes	RLL, EFM, EFMPlus, 4B/5B, 8B/10B	PLL	RLL 2,7 - als einfachste Form - bietet 50 % mehr Bitdichte gegenüber MFM; Festplatten, optische Speichermedien, schnelle Netzwerke und Interfaces
Manchester-Codierung		selbsttaktend*); PLL	herkömmliche lokale Netzwerke (Token Ring, 10Basex-Ethernet), 3270 Bildschirmsysteme
Daten-Strobe-Codierung		selbsttaktend*)	IEEE 1394 (FireWire)

*): Takt kann aus dem Datenstrom unmittelbar rekonstruiert werden (mittels kombinatorischer Verknüpfungen, Zeitstufen und State Machines). Manchmal ist aber auch dann die PLL die bessere Lösung (Seite 296)

Tabelle 6.1 Wichtige digitale Modulationsverfahren im PC-Bereich

6.1.1. Bitzelle, Signalperiode, Einheitsintervall

Die Bitzelle

Die zu sendende bzw. zu speichernde Bitfolge wird in Bitzellen (Bit Cells) fester, gleichbleibender Länge angeordnet. Stellen Sie sich beispielsweise eine Folge von Taktimpulsen vor, die eine Bitfolge aus einem Schieberegister herausschiebt. Dann wird jede Bitzelle durch einen Taktzyklus repräsentiert. Beim Empfangen bzw. Lesen muß die ankommende Signalfolge in die ursprüngliche Folge von Einsen und Nullen umgeschlüsselt werden, wobei die Bits wiederum in Bitzellen einzuordnen sind (Taktrückgewinnung). Abbildung 6.2 veranschaulicht den Begriff der Bitzelle.

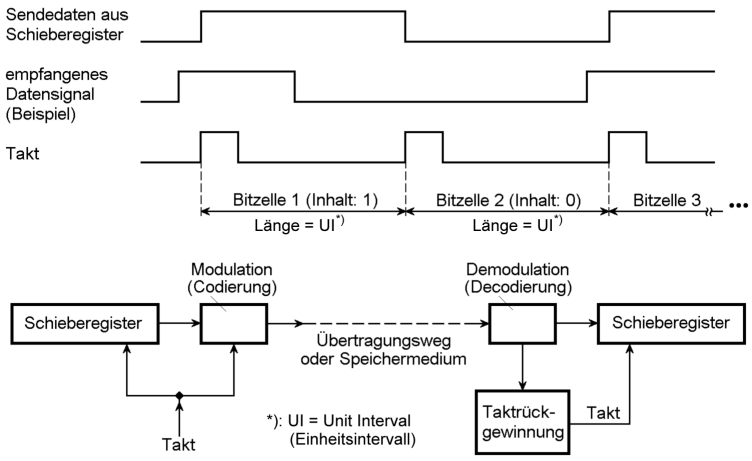

Abbildung 6.2 Die Bitzelle beim Senden und Empfangen

Je kürzer die Bitzelle, desto mehr Bits können in einem gegebenen Zeitabschnitt übertragen werden, je kleiner die Bitzelle, desto mehr Bits lassen sich auf einem bestimmten Speichermedium unterbringen.

Was letzten Endes die Dauer bzw. die Abmessung einer Bitzelle bestimmt, sind physikalische Tatsachen und technologische Gegebenheiten. Um uns diese Grenzen zu veranschaulichen, wollen wir zunächst von Bitmustern beliebiger Art absehen und eine einfache Impulsfolge mit 50 % Duty Cycle annehmen. Diese Impulsfolge wollen wir übertragen oder als gleichmäßiges Muster von Elementarmagneten bzw. Pit-Land-Folgen aufzeichnen.

Die minimale Signalperiode
Es leuchtet ohne weiteres ein, daß wir die Impulsfolgefrequenz nicht beliebig erhöhen können - irgendwann funktioniert es nicht mehr. Wir haben also eine obere Grenzfrequenz f_g, bei der sich die Impulsfolge gerade noch so übertragen oder aufzeichnen läßt, daß sie beim Empfangen oder Lesen eindeutig rekonstruiert werden kann, daß also keine Verfälschungen auftreten*). Der Kehrwert dieser Grenzfrequenz ist die minimale Signalperiode t_{pmin}. Daraus ergeben sich die minimalen High- und Low-Impulsbreiten zu $t_{pmin}/2 = 0{,}5\ t_{pmin.}$.

Mit anderen Worten: jeder Signalweg, jede Speichertechnologie ist durch eine bestimmte Mindestimpulsbreite $t_{pmin}/2$ und eine höchste Impulsfolgefrequenz $f_g = ½\ t_{pmin}/2$ gekennzeichnet. Wir wollen weiterhin annehmen, daß ein solcher Signalweg oder eine solche Speicheranordnung beliebige Folgen aus Low- und High-Impulsen übertragen oder speichern und wiedergeben kann, sofern die kürzesten Impulse (High oder Low) wenigstens jeweils eine Breite von $t_{pmin}/2$ haben**).

*): nochmals zur Beachtung: wir sprechen hier von einer Impulsfolgefrequenz, nicht aber von der Grenzfrequenz im Sinne der klassischen Elektrotechnik.

**): die Annahme läßt sich in der Praxis immer erfüllen, indem man f_g passend festlegt.

Das Einheitsintervall (Unit Interval UI)
Es liegt nahe, jedem Zeitabschnitt $t_{pmin}/2$ eine Bitposition zuzuordnen (Abbildung 6.3). Ein Aufeinanderfolgen von Nullen und Einsen (0 - 1 - 0 - 1 usw.) entspricht dann unserer maximalen Impulsfolgefrequenz f_g. In jeder Signalperiode ($1/f_g$) werden zwei Bits übertragen. Somit entspricht die maximale Datenrate (in Bits/s) $2\ f_g$. Der Kehrwert der maximalen Datenrate, also die Zeit, die zum Übertragen eines Bits zur Verfügung steht - mit anderen Worten: die Dauer einer Bitzelle -, ist das sog. Einheitsintervall (Unit Interval UI):

$$UI = \frac{1}{2\ f_g} = \frac{t_{pmin}}{2}$$

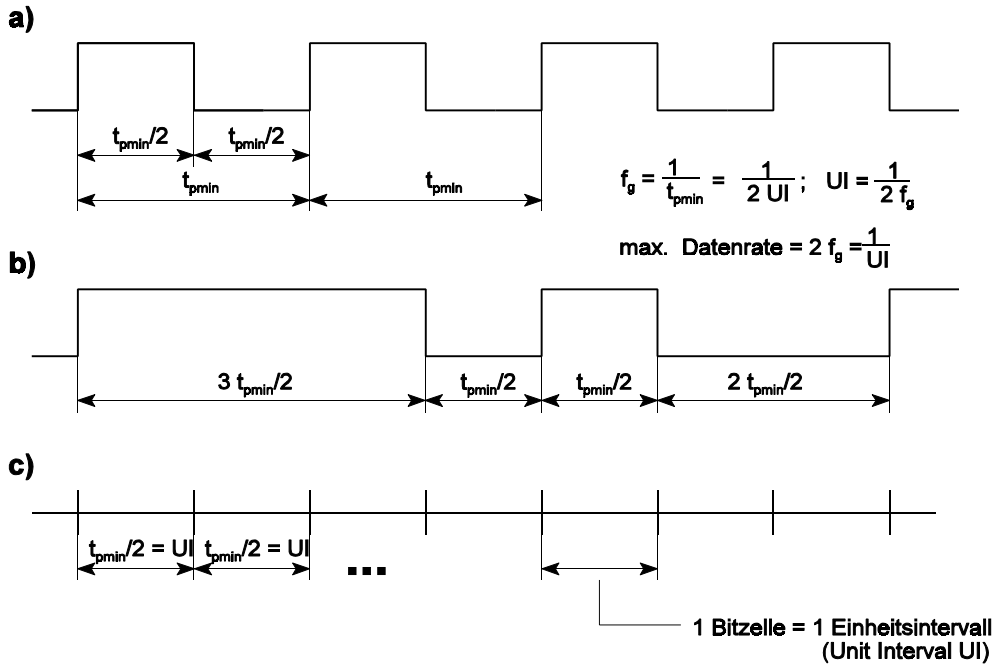

Abbildung 6.3 Einfachste Fälle der Signalübertragung

Erklärung:

a) Impulsfolge mit 50% Duty Cycle und maximaler Folgefrequenz,

b) Einrichtungen, die mit der Impulsfolge a) zurechtkommen, können auch solche Folgen fehlerfrei übertragen oder speichern*). Wichtig: jeder einzelne Low- oder High-Abschnitt dauert wenigstens $t_{pmin}/2$ oder ein Vielfaches davon.

c) es liegt deshalb nahe, $t_{pmin}/2$ als Länge einer Bitzelle zu definieren, mit anderen Worten: als Einheitsintervall.

*): zur Übung: welche Bitfolge wird hier übertragen?

...00101111...

Wir merken uns:
Ein Signalweg, der mit einer Impulsfolge der Frequenz f_g zurechtkommt, ermöglicht eine maximale Datenrate von 2 f_g Bits/s, wobei jede Bitposition einem Einheitsintervall von 1 : $2f_g$ entspricht. *Beispiele:*

- USB, Full Speed: 12 MBits/s; f_g = 6 MHz; Einheitsintervall = 83,3 ns,
- InfiniBand: 2,5 GBits/s; f_g = 1,25 GHz; Einheitsintervall = 0,4 ns (400 ps).

6.1.2. Serialisierung und Deserialisierung

Die auszugebenden Bits liegen parallel (z. B. als Bytes oder 32-Bit-Worte) vor und müssen in eine serielle Bitfolge umgesetzt werden (Parallel-Serien-Wandlung, Serialisierung). Sinngemäß sind die nacheinander ankommenen Bits zur weiteren Verarbeitung in paralleler Form bereitzustellen (Serien-Parallel-Wandlung, Deserialisierung). Die technischen Mittel hierfür: Multiplexer und Demultiplexer oder Schieberegister (Abbildung 6.4).

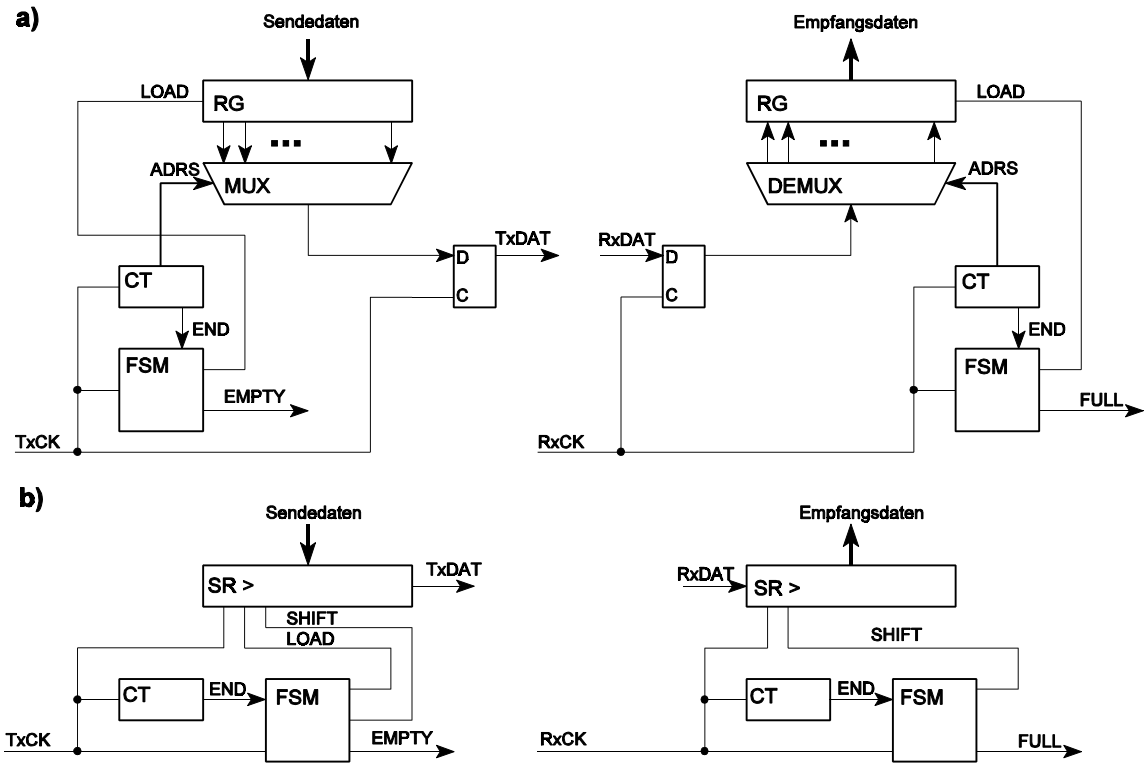

Abbildung 6.4 Serialisierung (links) und Deserialisierung (rechts)

Erklärung:
RG - Sende- oder Empfangsregister; LOAD - Ladeimpuls; ADRS - Bitauswahladresse; CT - Auswahladreß- oder Bitzähler; FSM - Ablaufsteuerung (Finite State Machine); SR - Schieberegister; TxDAT*) - Sendedaten; RxDAT*) - Empfangsdaten; TxCK - Sendetakt (Bittakt); RxCK - Empfangstakt; END - letztes Bit; SHIFT - Schiebeerlaubnis; EMPTY - Serializer leer (= neue Sendedaten liefern); FULL - Deserializer voll (= Empfangsdaten abholen).

*): Tx und Rx sind gängige Abkürzungen. Tx betrifft die Sendeseite (Transmitter), Rx die Empfängerseite (Receiver).

a) Abfrageprinzip. Die parallel vorliegenden Sendedaten werden über einen Multiplexer Bit für Bit abgefragt, der von einem Adreßzähler zyklisch adressiert wird. Auf der Empfängerseite werden die ankommenden Bits über einen Demultiplexer eines nach dem anderen in die einzelnen Flipflops des Empfangsregisters geladen.

b) Schiebeprinzip. Die Sendedaten werden in ein Schieberegister geladen und Bit für Bit ausgeschoben. Auf der Empfängerseite werden die ankommenden Bits in ein Schieberegister eingeschoben. Auf beiden Seiten zählt ein Bitzähler CT die Schiebetakte ab und meldet der Ablaufsteuerung, daß das letzte Bit aus- bzw. eingeschoben wird. Danach stehen die Empfangsdaten im Schieberegister zum parallelen Auslesen bereit, und das sendeseitige Schieberegister kann mit den nächsten Sendedaten geladen werden.

Hinweis:
Beide Verfahren haben ihre Vor- und Nachteile. Wir finden deshalb in modernen Schaltkreisen beide Prinzipien. Die typischen Vorteile:

- Abfrageprinzip: man kann an beliebigen Bitpositionen mit dem Wandeln anfangen und die Wandlungsreihenfolge nach Bedarf ändern - es genügt, den Adreßzähler mit einem entsprechenden Anfangswert zu laden oder in seiner Zählweise umzusteuern. Einsatzbeispiel: Abbildung 3.22.
- Schiebeprinzip: geringste Schaltungstiefe der Kombinatorik zwischen den Flipflops (nur Umsteuerung zwischen Laden und Schieben). Einsatzbeispiele: Abbildungen 6.8, 6.9 und 6.29.

Zur Taktfrequenz der Serialisierung und Deserialisierung
Hat unser Signalweg eine Grenzfrequenz f_g und wollen wir die maximale Datenrate von 2 f_g ausnutzen, so müssen wir auch mit einer Taktfrequenz von 2 f_g serialisieren und deserialisieren (jede Bitzelle erfordert einen Taktimpuls).

Ohne Zwischenpufferung geht es praktisch nicht
Oft werden die Daten von programmgesteuerten Einrichtungen geliefert und entgegengenommen (z. B. vom Prozessor des PCs und vom Mikrocontroller im Laufwerk). Stellen wir uns nun eine auf Abbildung 6.4b beruhende einfache Schaltungslösung vor (Abbildung 6.5).

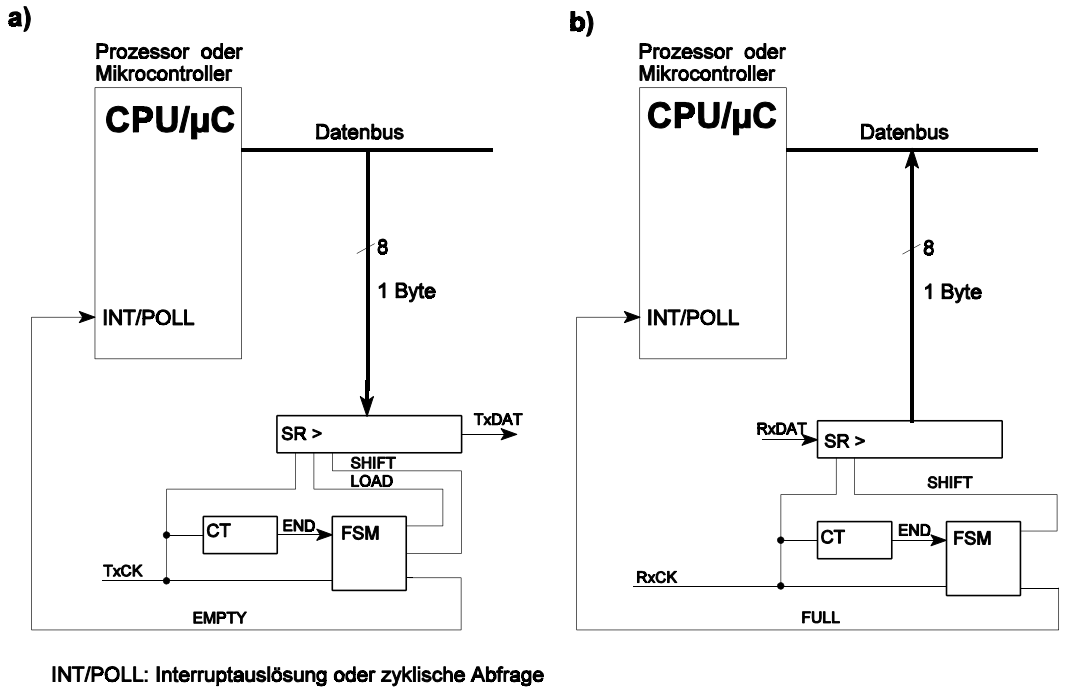

Abbildung 6.5 So einfach geht es in der Praxis kaum ...

Erklärung:

a) Senden. Wir nehmen an, daß ein zu seralisierendes Byte im Schieberegister steht und ausgegeben wird.Geschoben wird mit einer Taktfrequenz von 2 f_g. Wird das letzte Bit des ersten Bytes herausgeschoben, so aktiviert die Ablaufsteuerung das EMPTY-Signal. Daraufhin wird zum nächsten Sendetakt das nächste Byte auf dem Datenbus erwartet. Das heißt: es steht nur die Dauer einer einzigen Bitzelle ($1/2f_g$) zur Verfügung, um das nächste Byte zu laden.

b) Empfangen. Bit für Bit wird eingeschoben. Wurde ein Byte vollständig übertragen, so aktiviert die Ablaufsteuerung das FULL-Signal. Daraufhin muß der Schieberegisterinhalt sofort abgeholt werden, denn in der nächsten Taktperiode kommt schon das erste Bit des nächsten Bytes.

Betrachten wir einige Beispiele:

- serielle Schnittstelle mit 110 kBits/s. Einheitsintervall = 9,1 µs.
- USB (Full Speed) mit 12 MBits/s. Einheitsintervall = 83 ns.
- IEEE 1394 (Firewire) und USB 2 (High Speed) mit ca. 400...480 MBits/s. Einheitsintervall = 2,5...2,1 ns.

Offensichtlich kann ein Prozessor oder Mikrocontroller einen Übernahme- oder Ladezeitpunkt nicht auf wenige ns genau abpassen. Aber auch die rund 9 µs der seriellen Schnittstelle sind oft zuviel - nämlich dann, wenn sich der Prozessor nicht auschließlich darum kümmern kann, sondern erst über einen Interrupt auf diese Aufgabe umgeschaltet wird.

Was hier nicht hilft: die Schieberegister zu verlängern, beispielsweise von 8 auf 16 Bitpositionen. Damit bekommen wir zwar größere Abstände zwischen den Übernahmezeitpunkten, es steht aber nach wie vor jeweils nur ein Einheitsintervall zur Verfügung.

Der Ausweg: sende- und empfangsseitige Zwischenpuffer. Der Datenaustausch zwischen Puffer und Schieberegister wird über eine entsprechende State Machine gesteuert. Für den prozessorseitigen Zugriff auf das Pufferregister stehen dann - bei n Bitpositionen - n Einheitsintervalle zur Verfügung (beim byteweisen Schieben also acht). Reicht das nicht, so gibt es folgende Lösungen:

1. Einsatz von FIFO-Zwischenpuffern. Anwendung dann, wenn n Einheitsintervalle typischerweise ausreichen, aber mit zeitweiligen Verzögerungen zu rechnen ist. Beispiel: serielle Schnittstellen.
2. Einsatz größerer Pufferspeicher, die unter Steuerung schneller State Machines mit den Schieberegistern zusammenwirken*),
3. Datenaustausch in größerer Breite (z. B. 32-Bit-Worte anstelle von Bytes) - längere Schieberegister erhöhen die Zahl der verfügbaren Einheitsintervalle zwischen den Pufferzugriffen,
4. Anordnung entsprechender Pufferbereiche im Arbeitsspeicher, worauf die Übertragungshardware selbständig zugreift (DMA- oder Busmasterbetrieb). Eine vor allem im PC-Bereich typische Sparlösung (IDE/ATA, SCSI, USB usw.),
5. Kombinationen von 1. bis 4. Beispiele: (1) Telekommunikationseinrichtungen, (2) Abbildung 6.6.

*): in dieser Hinsicht besonders vorteilhaft: Übertragungsverfahren, die auf Paketen fester Länge beruhen. Es liegt dann nahe, die Pufferspeicher so auszulegen, daß sie komplette Pakete aufnehmen können.

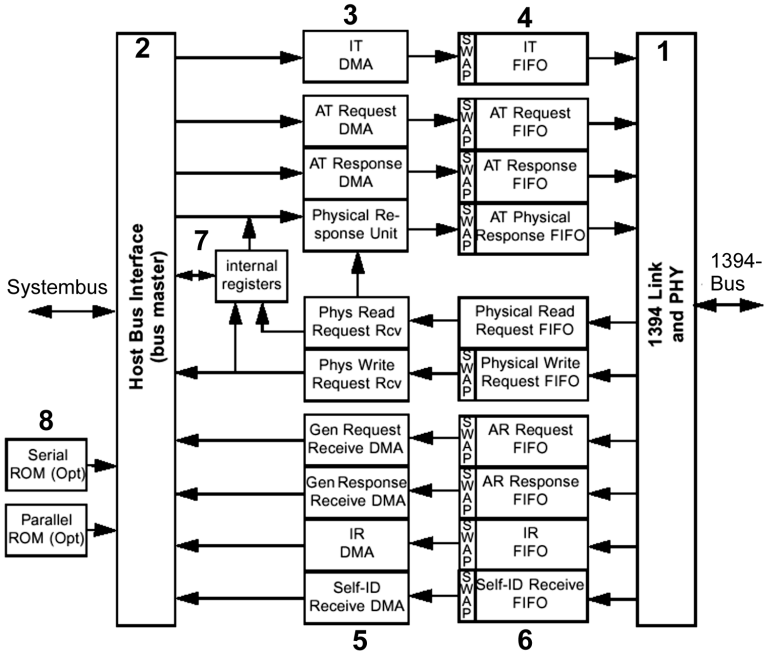

Abbildung 6.6 Beispiel einer modernen Schnittstelle: OHCI 1384

Erklärung zu Abbildung 6.6:
OHCI = Open Host Controller Interface. Es handelt sich um einen Interfacestandard, der eine vereinheitlichte, in vielen Systemen nutzbare Schnittstelle zwischen dem eigentlichen Interface (hier: IEEE 1394 (Firewire)) und der Systemplattform vorsieht. 1 - Interfaceanschlußsteuerung (physische Kopplung + Link-Schicht); 2 - Schnittstelle zum Systembus (z. B. PCI), als Busmaster wirkend; 3 - DMA-Kanäle für Ausgabezugriffe; 4 - Ausgabe-FIFOs; 5 - DMA-Kanäle für Eingabezugriffe; 6 - Eingabe-FIFOs; 7 - interne Register; 8 - ROMs mit Konfigurationsangaben; AT - asynchrone Ausgabe; IT - isochrone Ausgabe; AR - asynchrone Eingabe; IR -isochrone Eingabe (Ausgabe = Transmit (T); Eingabe = Receive (R)). Die Zugriffswege sind typischerweise 32 Bits breit. Es sind mehrere Zugriffskanäle (DMA-Kanäle) zum Systembus vorgesehen, denen je nach Übertragungsrichtung FIFO-Puffer vor- oder nachgeschaltet sind. Da IEEE 1394 eine Hochleistungsschnittstelle ist, hat jede Übertragungsart ihre eigenen FIFOs und DMA-Kanäle.

Hinweise:

1. Zum Begriff *DMA*: hier sind nicht die herkömmlichen DMA-Vorkehrungen der PCs gemeint. Als DMA-Kanal bezeichnet man vielmehr eine Hardware, die selbständig auf den Speicher zugreifen kann, um Datenblöcke zu lesen oder zu schreiben. Eine solche Hardware enthält u. a. Adreßzähler zum fortlaufenden Adressieren der einzelnen Datenworte, Längenzähler zum Abzählen der zu übertragenden Bytes und die entsprechenden Steuerschaltungen. Diese DMA-Kanäle werden im System typischerweise als Busmaster wirksam (z. B. am PCI-Bus). Die Zugriffsweise entspricht näherungsweise dem uns bereits bekannten Busmaster-DMA der modernen IDE/ATA-Laufwerke.
2. Was geschieht, wenn einmal eine Zugriffsanforderung nicht zeitig genug bedient wird? Dann gibt es einen Datenverlust. Um diese zu beheben, gibt es entsprechende Wiederanlauffunktionen
3. Moderne Hochleistungsinterfaces sind mit den herkömmlichen PC-Schnittstellen überhaupt nicht mehr zu vergleichen. Die einschlägigen Standards sind umfangreich und kompliziert (vgl. auch Abbildung 7.1). Die Entwicklungen dauern etliche Jahre, und es sind ganze Heerscharen von Fachleuten daran beteiligt. (Sehen Sie sich einmal auf den Homepages einiger Standardisierungsgremien um. Laden Sie sich einige Standards herunter - und sei es nur, um einen Eindruck davon zu bekommen, wie so etwas aussieht ...)

Serdes, Transceiver, SIE
Serdes = Serializer/Deserializer (sprich: Sieriäleiser/Desieriäleiser). Ein gängiger Fachbegriff für entsprechende Einrichtungen. Gelegentlich wird auch der Allerwelts-Fachbegriff *Transceiver*[*)] in diesem Sinne verwendet. Ein weiterer Begriff (der z. B. in USB-Literatur oft vorkommt): SIE = Serial Interface Engine. Da serielle Interfaces groß in Mode sind[**)], gibt es eine Vielzahl von Ausführungen. Die Abbildungen 6.7 bis 6.9 zeigen einschlägige Beispiele.

*): sprich: Trännsiefer = Transmitter + Receiver = Sender + Empfänger.

**): das betrifft nicht nur die durchgehend standardisierten E-A-Interfaces. U. a. werden Schaltkreise angeboten, um an sich beliebige Schnittstellen durch Zwischenschaltung

serieller Signalwege gleichsam verlängern zu können (Abbildung 6.9).

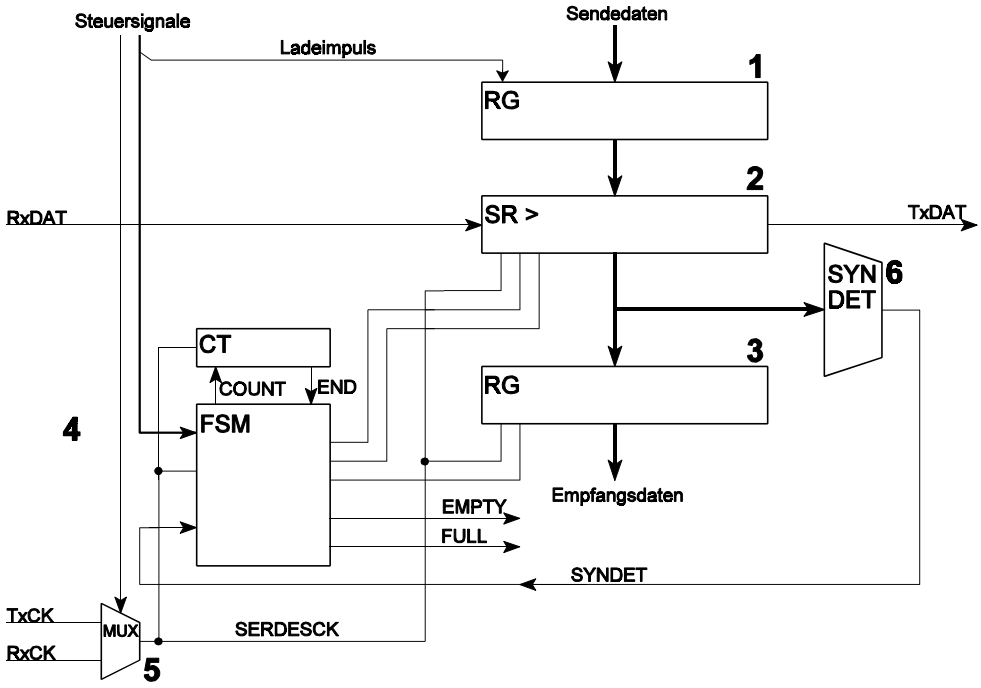

Abbildung 6.7 Herkömmlicher Serdes für Halbduplexbetrieb

Erklärung:
1 - Sendepuffer; 2 - Schieberegister; 3 - Empfangspuffer; 4 - Ablaufsteuerung mit Zähler und State Machine; 5 - Taktauswahl; 6 - Synchronzeichenerkennung (SYNDET = Sync Pattern Detect). Die Anordnung entspricht der Kombination der beiden Schaltungen von Abbildung 6.4b. Ein einziges Schieberegister mit paralleler Ein- und Ausgabe dient sowohl zum Senden als auch zum Empfangen. Deshalb kann man zu einer Zeit entweder nur senden oder nur empfangen (Halbduplexbetrieb)[*)]. Der Bittakt (SERDESCK) kann entweder vom Sendetaktgenerator (TxCK) oder von der empfangsseitigen Taktrückgewinnung (RxCK) geliefert werden. Die Funktion der Synchronzeichenerkennung 6 erklären wir weiter unten in Abschnitt 6.1.4. Eine solche Anordnung wird typischerweise von einem Mikroprozessor gesteuert, der u. a. Steuersignale an die Ablaufsteuerung 4 und Ladeimpulse für den Sendepuffer 1 liefert.

*): ein Anwendungsfall, bei dem das keine Einschränkung bedeutet: der Disketten- oder Festplattenanschluß.

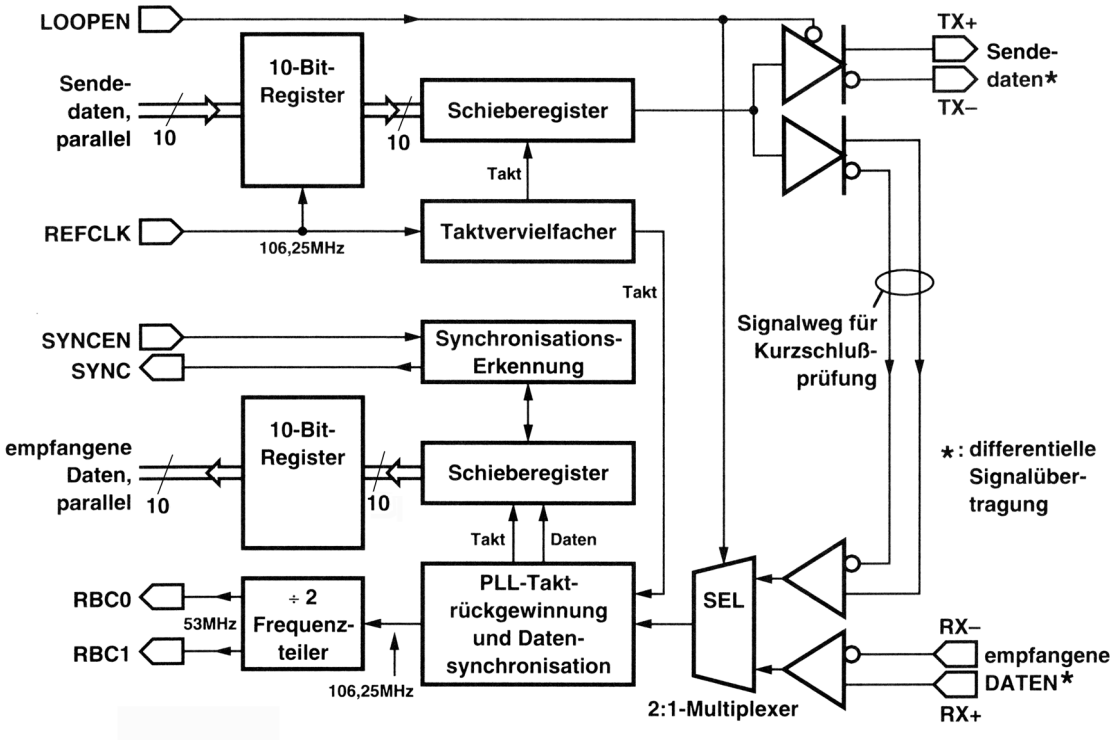

Abbildung 6.8 Ein Serdes-Schaltkreis für ein Hochleistungsinterface: der Fibre Channel Transceiver SN75FC1000 (Texas Instruments)

Erklärung:
Hochleistungsinterfaces werden typischerweise so ausgelegt, daß in beiden Richtungen gleichzeitig Daten übertragen werden können (Vollduplexbetrieb). Hierfür werden Serializer und Deserializer getrennt aufgebaut.

Fibre Channel nutzt die 8B/10B-Codierung (Abschnitt 6.6.4.). Unser Schaltkreis arbeitet mit 10-Bit-Worten (die Umschlüsselung wird in anderen Schaltkreisen erledigt). Die serielle Datenrate beträgt 1,0625 GBits/s; deshalb wird mit einem Bittakt von 1,0625 GHz geschoben. Hierzu erhält der Schaltkreis einen Referenztakt (Worttakt) von 106,25 MHz zugeführt, dessen Frequenz intern verzehnfacht wird.

Hinweis:
Beachten Sie die Vorkehrungen zur Kurzschlußprüfung. Durch Aktivieren des Steuereingangs LOOPEN kann man die Sendedaten vom Ausgang des Serializers über einen 2:1-Multiplexer auf den Eingang des Deserializers zurückführen. Dabei werden die Interfaceanschlüsse deaktiviert (Sendedaten TX) bzw. nicht ausgewertet (empfangene Daten RX).

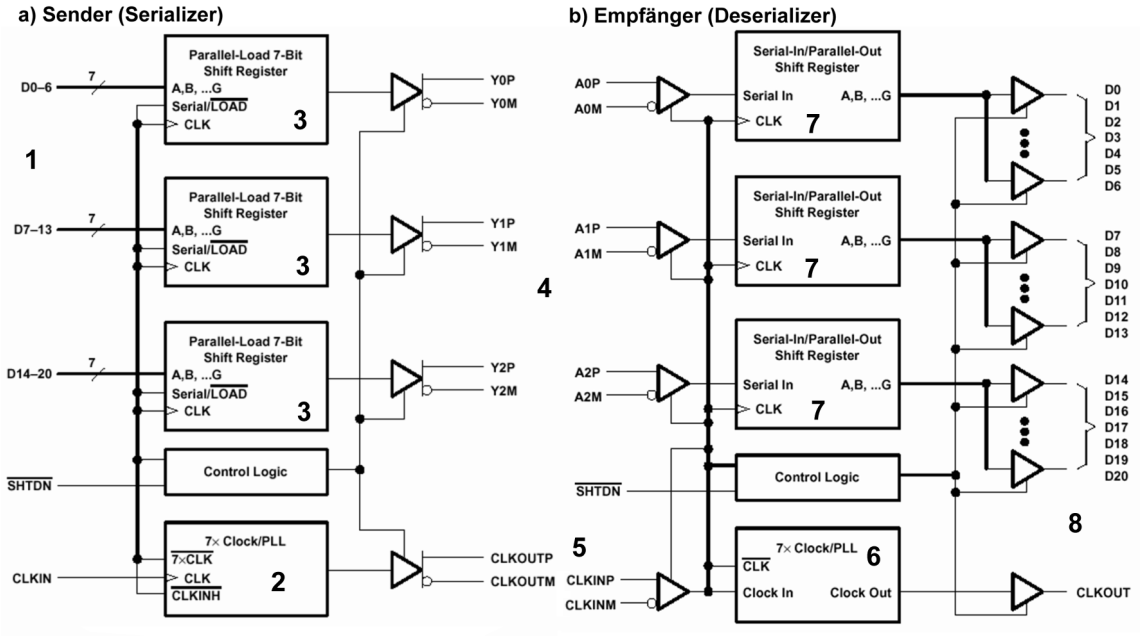

Abbildung 6.9 Schnittstellenwandlerschaltkreise SN65LVDS95/96 (Texas Instruments)

Erklärung:
1 - parallele Dateneingänge; 2 - sendeseitige Takterzeugung (PLL); 3 - Serializer (parallel ladbare Schieberegister); 4 - differentielle Datenwege (LVDS); 5 - differentieller Taktsignalweg; 6 - empfangsseitige Takterzeugung (PLL); 7 - Deserializer (Schieberegister mit parallelen Ausgängen); 8 - parallele Datenausgänge. Der Sender hat 21 Dateneingänge, die auf drei 7-Bit-Schieberegister geführt werden. Die Daten werden seriell über 3 differentielle Signalpaare ausgegeben. Sinngemäß hat der Empfänger drei 7-Bit-Schieberegister mit parallelen Ausgängen, an denen die jeweils übertragenen 21 Datenbits abgenommen werden können. Diese Schaltkreisfamlie ist lediglich dazu vorgesehen, parallel anliegende Datenbelegungen über wenige serielle Wege zu übertragen und auf der Empfängerseite wieder parallel bereitzustellen - und zwar gleichsam schmucklos, ohne jegliche höhere Protokollebenen, besondere Standards o. dergl.

Die Steuerung ist ganz einfach. Der sendeseitig anliegende Takt (CLKIN) ist der Takt, mit dem die parallelen Daten bereitgestellt werden. Die PLL 2 erzeugt einen siebenmal schnelleren Takt, der die serielle Datenübertragung steuert. Solange der Takt CLKIN anliegt, läuft eine einfache Folge ab: Parallelübernahme in die Schieberegister 3, Ausschieben, Parallelübernahme, Ausschieben usw.

Ein zu CLKIN synchroner Takt wird zum Empfänger übertragen. Dort erzeugt die PLL 6 einen siebenmal schnelleren Takt zum Einschieben der seriellen Daten. Des weiteren wird ein Ausgangstakt CLKOUT gebildet, der verwendet werden kann, um die Belegung der parallelen Ausgänge 8 in nachfolgende Schaltungen zu übernehmen.

Bereich des parallelen Taktes: 20...65 MHz. D. h. im Abstand von 15,4 bis 50 ns können 21 Bits übertragen werden. Das entspricht einer Datenrate von maximal 1,36 GBits/s. Die maximale Impulsfolgefrequenz auf dem einzelnen Signalweg beträgt dabei höchstens 227,5 MHz (NRZ-Signalisierung).

6.1.3. Taktrückgewinnung

Es gibt selbsttaktende und nicht selbsttaktende Modulationsverfahren. Ein selbsttaktendes Verfahren hat je Bitzelle wenigstens einen Signalübergang (eine Impulsflanke), aus der eindeutig ein Bezugstakt abgeleitet werden kann. Ist dies nicht gegeben, muß empfangs- bzw. leseseitig ein selbsttätig arbeitender (frei schwingender) Taktgenerator vorgesehen sein. Es gibt folgende Prinzipien:

- zusätzliche Übertragung eines Takt- bzw. Strobesignals (über einen eigenen Signalweg). Beispiele: I^2C-Bus, IEEE-1394-Interface (Abschnitt 6.5.).
- asynchrone Übertragung. Beide Einrichtungen (Sender und Empfänger) arbeiten jeweils mit einem eigenen Takt konstanter Frequenz. Start- und Stopbits dienen dazu, die einzelnen zu übertragenden Zeichen voneinander abzugrenzen. Erkennt der Empfänger ein Startbit, so beginnt er, den ankommenden Datenstrom mit seinem Takt abzutasten. Das Abtasten endet mit dem Empfang des (bzw. der) Stopbits. Danach wartet der Empfänger auf das nächste Startbit. Die zulässigen Takttoleranzen sind so festgelegt, daß über die wenigen (typisch 8...10) Takte zwischen Start- und Stopbit hinweg jeder Abtasttakt im Empfänger immer auf eine gültige Bitzelle trifft.
- selbsttaktende Modulation. Je Bitzelle gibt es wenigstens eine Signalflanke, aus der ein Taktimpuls abgeleitet werden kann. Extremfall: FM (Abschnitt 6.3.) - hier enthält jede Bitzelle einen eigenen Taktimpuls.
- Taktrückgewinnung durch Synchronisation. Der Empfänger enthält einen frei schwingenden Taktgenerator, der in der Lage ist, sich auf die jeweils aktuelle Bitzellen-Folgefrequenz einzustellen. Dabei werden die empfangenen Signalübergänge (Impulsflanken) genutzt, um erforderlichenfalls die Frequenz nachzuregeln. Technische Ausführung: spannungsgesteuerter Oszillator (Voltage Controlled Oscillator VCO) mit Phasenregelschleife (Phase Locked Loop PLL).

Das Start-Stop-Verfahren: Signalübertragung über serielle Schnittstellen
Das Prinzip (Abbildung 6.10) beruht darauf, daß sich sowohl Sender als auch Empfänger über die Dauer der Bitzelle und über die Zahl der nacheinander zu übertragenden Bits einig sind. Das heißt, beide Einrichtungen arbeiten mit gleichem Bittakt (1/UI), der jeweils mit hinreichender Genauigkeit (Frequenzkonstanz) bereitgestellt wird. Das funktioniert deshalb, weil man jeweils nur vergleichsweise wenige Bits überträgt (in der Praxis werden Zeichen von 5...10 Bits übertragen). Im Ruhezustand wird ein Eins-Pegel signalisiert. Die Übertragung eines Zeichens beginnt mit einer Null-Bitzelle (Startbit)[*]. Der erste Eins-Null-Übergang - aus dem Ruhezustand heraus - veranlaßt den Empfänger, mit dem Abtasten des ankommenden Signals zu beginnen. Es wird jede Bitzelle mehrmals abgetastet, beispielsweise mit einem Takt, der die 16-fache Frequenz des Bittaktes hat. Trifft der erste Abtastimpuls auf den Eins-Null-Übergang, so hat man nach weiteren 24 solchen Impulsen ziemlich sicher die Mitte der nachfolgenden Bitzelle getroffen (diese enthält das erste Nutz-Bit des übertragenen Zeichens). Mit jeweils 16 Abtastimpulsen Abstand werden dann die weiteren Bitzellen näherungsweise in der Mitte abgetastet. Sind alle Zeichen-Bits (und eventuell ein weiteres Bit zu Kontrollzwecken; Stichwort: Paritätskontrolle) übertragen worden, wird ein "1"-Bit als Endekennung (Stopbit) erwartet (kommt keine 1, liegt ein Fehler vor). Daraufhin gelangt der Empfänger in den Ruhezustand und erwartet das nächste Startbit. Es gibt auch Übertragungsformate mit 2 oder mit 1 ½ Stopbits ("1 ½ Bits" bedeutet, daß wenigstens 1 ½ Bitzellen mit High-Pegel belegt werden).

*): diese Signalbelegung (Ruhe = Eins, Start mit Null) ist eine altehrwürdige Tradition aus der Telegraphen- und Fernschreibtechnik.

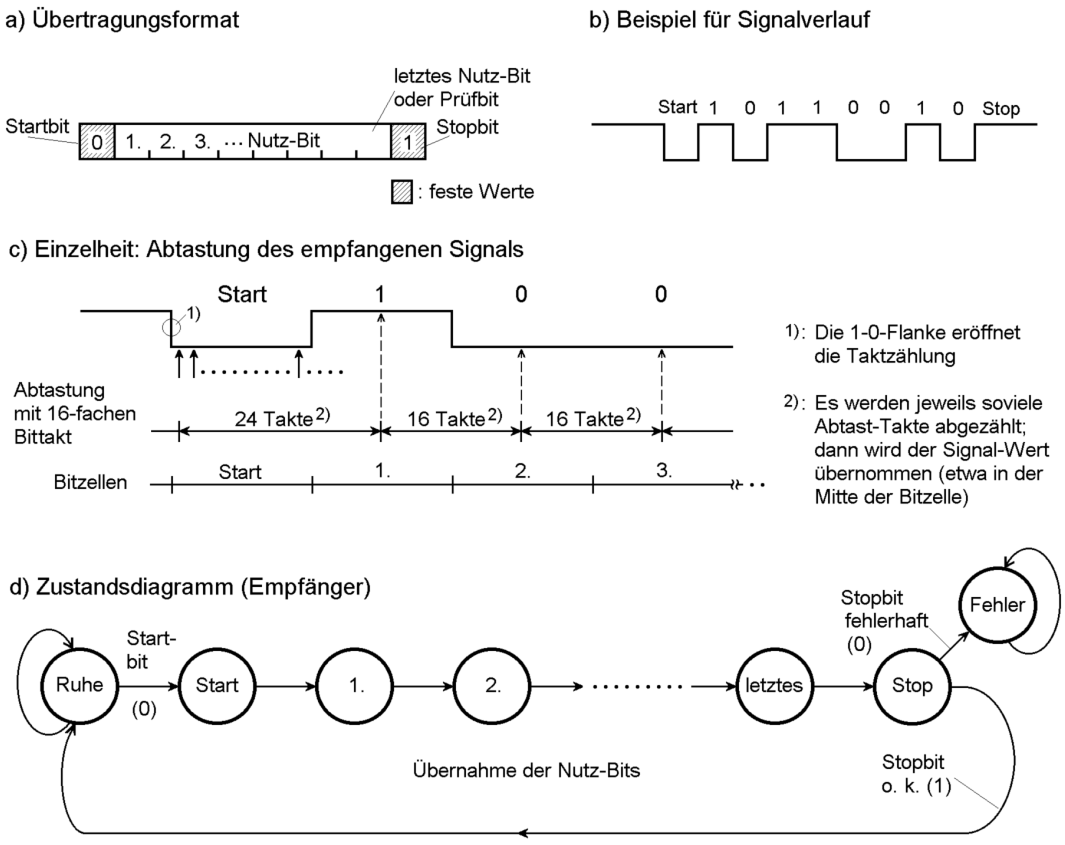

Abbildung 6.10 Signalübertragung nach dem Start-Stop-Verfahren

Wie einigen sich Sender und Empfänger?
Üblicherweise sind die Schnittstellenparameter, wie Bitrate (diese ist hier - wegen der zweiwertigen Signalübertragung - identisch mit der *Baudrate*), Anzahl der Bits je Zeichen, Anzahl der Stopbits usw., einstellbar (im PC programmseitig, in einfacheren peripheren Geräten über DIL-Schalter oder Steckbrücken; es gibt auch Geräte, in denen die Einstellungen in einem Speicher, z. B. einem EEPROM, gehalten werden). Beide Einrichtungen am Interface müssen auf gleiche Parameter eingestellt werden.

Automatische Einstellung (Autoconfiguration)
Diese Funktion ist beispielsweise in modernen Modems vorgesehen. Das bedeutet aber nicht eine fliegende Anpassung an ständig wechselnde Parameter. Vielmehr werden verschiedene Einstellungen solange durchprobiert, bis endlich eine fehlerfreie Übertragung zustande kommt. Diese Vorgänge spielen sich im wesentlichen auf höheren Protokollebenen ab. Ein typisches Beispiel ist das automatische Wählen der Baudrate und deren "Zurückfahren", falls zuviele Übertragungsfehler vorkommen (Automatic Baud Rate Switching; Fallback).

Wie funktioniert eine PLL? Eine erste Einführung
Abbildung 6.11 zeigt die Grundschaltung (vgl. die Abbildungen 5.15, 5.16). in einer Form, die zum Erklären des hier in Rede stehenden Einsatzfalls besser geeignet ist.

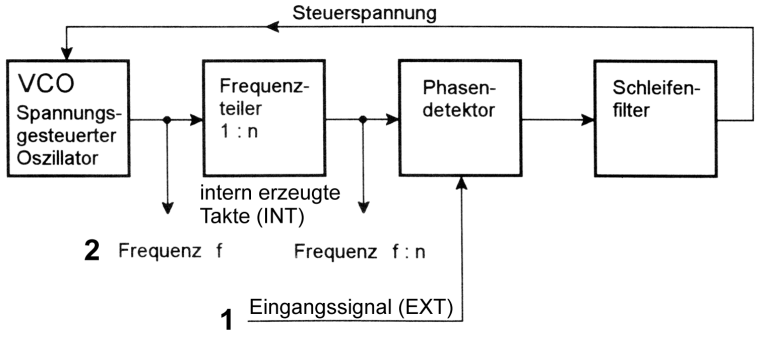

Abbildung 6.11 PLL-Grundschaltung

Erklärung:
1 - hier wird das empfangene Signal eingespeist; 2 - hier werden durchlaufende Taktimpulse geliefert. Die mit Frequenz f und Frequenz f : n bezeichneten Signale können zur Taktsignalbildung genutzt werden. Im einfachsten Fall entspricht f : n dem Bittakt des Eingangssignals. Am Frequenzteiler oder am Ausgang des VCOs können Taktsignale höherer Frequenz abgegriffen werden, beispielsweise um zeitverschobene Takte oder einen Mehrphasentakt zu bilden. Bleiben wir aber beim einfachsten Fall: der VCO wird in Frequenz und Phase so nachgeregelt, daß das interne Taktsignal (INT) zu den ankommenden Impulsen (EXT) im Rahmen zulässiger Abweichungen (Stichwort: Jitter) synchron bleibt (Abbildung 6.12).

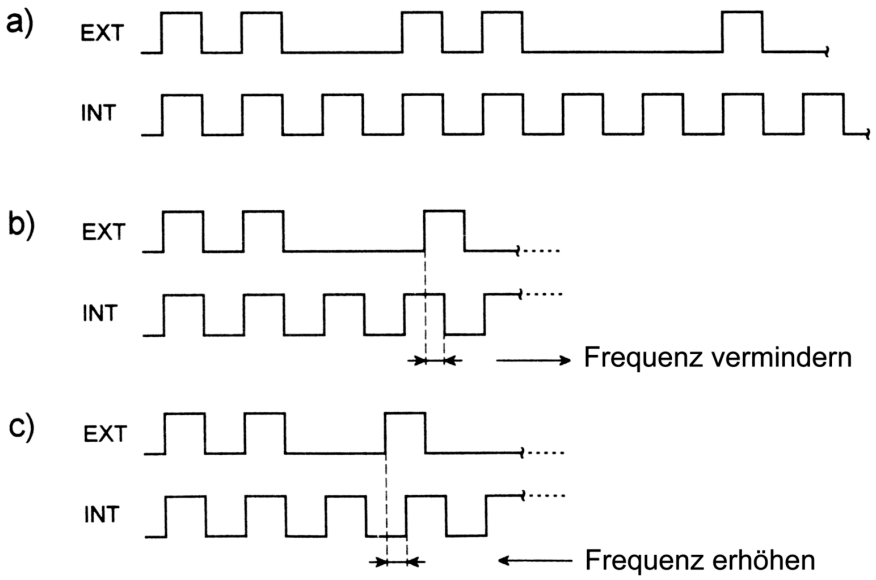

Abbildung 6.12 Zur Wirkungsweise der PLL

Erklärung:

a) so soll es sein: jedem ankommenden Impuls (EXT) steht genau ein Taktimpuls (INT) gegenüber. Wenn sich ankommende und Taktimpulse stets decken, kann man annehmen, daß die Takte auch in den Pausen "richtig liegen"; mit anderen Worte: die von der PLL gelieferten Taktimpulse können direkt zum Abtasten des ankommenden Signals

verwendet werden (um es beispielsweise in ein Schieberegister zu übernehmen). Trifft ein Taktimpuls auf einen ankommenden Impuls, so handelt es sich um eine Eins, findet ein Taktimpuls keinen ankommenden Impuls vor, um eine Null.

b) hier kommt ein ankommender Impuls etwas zu spät. Es ist also damit zu rechnen, daß die folgenden Impulse auch Verspätung haben werden. Folglich sind die folgenden Taktimpulse etwas zu verzögern (mit anderen Worten: die Taktfrequenz ist zu verringern), damit später wieder Takt und Daten genau zusammentreffen.

c) hier kommt ein ankommender Impuls etwas zu früh. Es ist also damit zu rechnen, daß die folgenden Impulse auch eher eintreffen werden. Folglich müssen die nächsten Taktimpulse in kürzeren Abständen abgegeben werden (mit anderen Worten: die Taktfrequenz ist zu erhöhen).

Die PLL will Impulse sehen
Kommen am Eingang (Position 1 in Abbildung 6.11) keine Impulse an, so hat die PLL nichts, womit sie das VCO-Signal vergleichen kann.

Wir merken uns:
Die Taktrückgewinnung über PLL erfordert, daß im ankommenden Datenstrom in nicht allzu langen Abständen Impulse auftreten, damit die PLL etwas zum Vergleichen hat.

Die verschiedenen Codierungen der Datenübertragung bzw. der Informationsspeicherung dienen genau dazu - es geht darum, in Abhängigkeit von der Datenbelegung Bits (= Impulse) in den Datenstrom einzufügen (die auf der Empfängerseite wieder extrahiert werden müssen). Beispiele:

- die verschiedenen RLL-Codes der magnetischen und optischen Informationsspeicherung (Abschnitt 6.6.),
- die 4B/5B-Codierung beispielsweise des 100Base-T-Ethernet (Abschnitt 6.6.3.),
- die 8B/10B-Codierung, wie sie im oberen Leistungsbereich üblich ist (Escon, Ficon, Fibre Channel, IEEE 1394b, InfiniBand usw.). Hierbei werden 8 Datenbits in 10 zu übertragende Bits umgeschlüsselt (Abschnitt 6.6.4.).
- Bit Stuffing. Nach einer bestimmten Anzahl von Nullen wird zwangsweise eine Eins in den Bitstrom eingefügt. (Abwandlung: Einfügen einer Null nach einer bestimmten Anzahl von Einsen: beispielsweise wird beim USB nach jeweils 6 aufeinanderfolgenden Einsen eine Null eingefügt.)

Hinweis:
Herkömmliche serielle Schnittstellen wandeln gar nichts, sondern hängen lediglich Start- und Stopbits an.

6.1.4. Synchronisation

Alle Signalflanken, die auf eine getaktete Schaltung treffen, sind zunächst einmal bezüglich dieses Taktsystems *asynchron* und müssen *synchronisiert* (eintaktiert) werden. Bei frequenzkonstantem (quarzstabilem) Empfängertakt müssen wir den Datenstrom lediglich abtasten.

Bei frequenzvariablem (PLL-) Empfängertakt sind hingegen zwei besondere Probleme zu lösen:

- zu Beginn des Empfangs- bzw. Lesevorgangs muß sich der VCO in der PLL zunächst einschwingen (dies erfordert eine gewisse Zeit; erst dann befindet er sich im Gleichtakt mit den ankommenden Signalflanken und ist in der Lage, Frequenz und Phasenlage exakt nachzuführen),
- die zeitliche Lage der ankommenden Impulse ist zunächst unbekannt. Woher soll die Empfangsschaltung wissen, zu welcher Bitposition die erste Signalflanke im ankommenden Datenstrom gehört?

Die Lösung beider Probleme: die Informationsübertragung bzw. -speicherung wird in festen Formaten (Sektoren, Paketen, Frames) organisiert. Wichtig ist: der eigentlichen Nutz-Information geht ein fest formatierter Vorspann (Präambel, Preamble) voraus. Dieser gibt der PLL Gelegenheit zum Einrasten. Dann folgen feste Bitmuster, die zum Synchronisieren des Paketanfangs dienen (Abbildung 6.13).

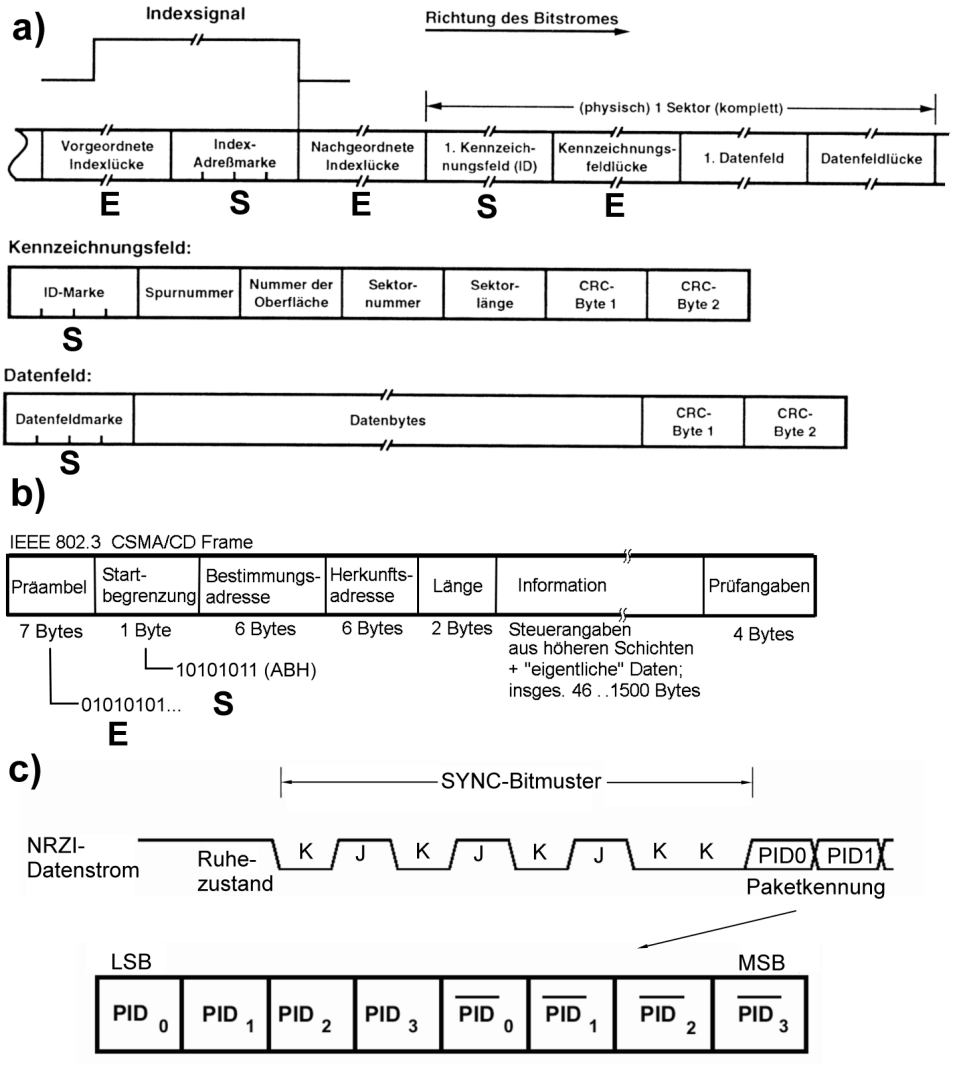

Abbildung 6.13 Synchronisationsvorkehrungen in Datenformaten (Beispiele)

Erklärung zu Abbildung 6.13:
E - Bitmuster, die das Einrasten der PLL (Capture) unterstützen; S - Bitmuster, die den Anfang der jeweiligen Informationsstruktur kennzeichnen (Synchronisations- bzw. SYNC-Muster).

Das Prinzip: die ersten Bitfolgen (E) dienen zum Einrasten; anschließend erwartet die empfangende Einrichtung bestimmte charakteristische Bitmuster (SYNC-Zeichen S). Betrachten wir nochmals Abbildung 6.7. Die Anordnung sei auf Empfangen geschaltet. Der Empfangstakt wird von der PLL geliefert (vgl. Abbildung 6.1). Zunächst kommen Impulsfolgen, die Einsen und Nullen in gleichmäßigem, dichtem Abstand enthalten. Diese geben der PLL Gelegenheit zum Einrasten. Dann folgen charakteristische Impulsmuster, die sich von den zuvor gesendeten Impulsfolgen deutlich unterscheiden (die sog. SYNC- bzw. Synchronzeichen). Im einfachsten Fall gibt es ein einziges Synchronzeichen, dessen Auftreten direkt am Schieberegister des Deserializers durch ein kombinatorisches Netzwerk erkannt wird (Decodierung oder Vergleich mit vorgegebenem Bitmuster). Wird ein Synchronzeichen erkannt, so weiß die Hardware, daß jetzt das eigentliche Informationspaket nachfolgt. (Der - meist programmseitig einstellbare - Betriebszustand, in dem der Deserializer auf Synchronzeichen wartet, wird typischerweise als SYNDET-Modus (Sync Pattern Detect), Hunt-Modus o. ä. bezeichnet.)

a) ein herkömmliches Spurformat auf Disketten und Festplatten. Die diversen Lücken (Gaps) enthalten Bitmuster, der der PLL Gegegenheit zum Einrasten geben, die Marken (Markers) sind typische Bitmuster, die den Beginn bestimmter Informationsfelder kennzeichnen.
b) ein LAN-Frame gemäß IEEE 802.3 (volkstümlicherweise = Ethernet). Er beginnt mit einem Bitmuster 010101... als Präambel (E). Dann folgt ein Byte 10101011 (ABH) als SYNC-Zeichen (Startbegrenzung; Start Frame Delimiter SFD).
c) Synchronisation beim USB. Dem SYNC-Bitmuster (8 Bits) folgt ein Byte, das als Paketkennung dient (Packet Identfier PID). Das SYNC-Muster entspricht - in NRZI-Codierung - 7 Nullen, gefolgt von einer Eins*). Die Paketkennung ist ein Byte, das einen 4-Bit-Kenncode enthält, gefolgt von dessen Negation (das ist eine Fehlerkontrollmaßnahme - damit ein PID als gültig erkannt wird, müssen die höchstwertigen 4 Bits die bitweise Negation der niedrigstwertigen 4 Bits enthalten).

*): in der besonderen Ausdrucksweise der USB-Spezifikation handelt es sich um die Signalfolge KJKJKJKK. J und K sind differentielle Belegungen des USB-Signalwegs. Ersichtlicherweise wird zuerst eine Impulsfolge mit 50 % Duty Cycle angeboten (damit die PLL einrasten kann). Die abschließenden beiden Nullen (KK-Belegung) leiten zum PID-Byte über.

Moderne Hochleistungsinterfaces
Man bevorzugt die 8B/10B-Codierung (Abschnitt 6.6.4.), die entsprechende Steuer- und Synchronisationszeichen enthält (diese Zeichen können nicht mit Datenbelegungen verwechselt werden). Die Pakete oder Frames haben aber typischerweise keinen Vorspann im herkömmlichen Sinne. Man begnügt sich nicht mehr damit, die PLL durch eine simple Impulsfolge zum Einrasten zu bewegen. Statt dessen sendet man von Zeit zu Zeit ausgeklügelte Signalfolgen, auf die der Empfänger reagieren muß. Man spricht hier vom Trainieren des Signalweges (Link Training). Tatsächlich ist einiges zu tun: der Empfänger ist aus Stromsparzuständen aufzuwecken, Schwellwerte sind zu justieren (wenn es um GBits/s geht,

kommt es auch darauf an...) und mehrere parallel betriebene serielle Verbindungswege sind - hinsichtlich des Skew - aufeinander abzustimmen (Beispiel: InfiniBand).

6.2. Elementare Signaldarstellungen (RZ, NRZ, NRZI)

6.2.1. Return to Zero (RZ)

Eine Null ist ein konstanter Low-Pegel in der jeweiligen Bitzelle; eine Eins ist ein Impuls, der noch innerhalb der Bitzelle auf Low zurückkehrt (Abbildung 6.14).

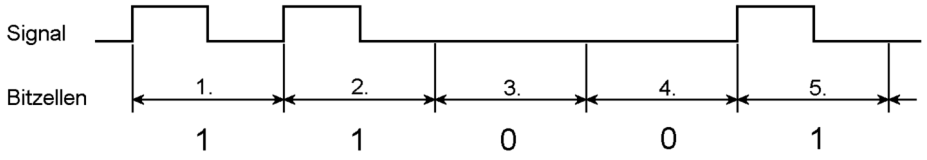

Abbildung 6.14 Signaldarstellung Return to Zero (RZ)

Der wesentliche Nachteil: eine maximale Impulsfolgefrequenz f_g erlaubt eine Datenrate von höchstens f_g Bits/s; das Einheitsintervall entspricht $1/f_g$. Deshalb wird diese Codierung praktisch nicht verwendet (vgl. auch den Hinweis auf Seite 294). Folgen von Nullen liefern keine Impulse, die zur Taktrückgewinnung genutzt werden können.

6.2.2. Non Return to Zero (NRZ)

Eine Null ist ein konstanter Low-Pegel in der jeweiligen Bitzelle, eine Eins ist ein konstanter High-Pegel. Folgen Einsen aufeinander, so kehrt das Signal zwischenzeitlich, im Gegensatz zu RZ, *nicht* auf Low zurück. NRZ ist gleichsam der standardmäßige Signalverlauf, wie er am Ausgang des Serializers (z. B. eines Schieberegisters) erscheint*). NRZ bildet somit praktisch die Grundlage jeder nachfolgenden Modulation.

*): jede einfache Synchronisation (Datenübernahme in ein D-Flipflop; vgl. Abbildung 2.18) verwandelt eine ankommende Impulsfolge in ein NRZ-Signal.

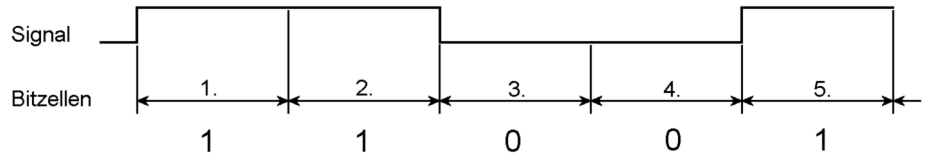

Abbildung 6.15 Signaldarstellung Non Return to Zero (NRZ)

Eine maximale Impulsfolgefrequenz f_g erlaubt eine Datenrate von $2f_g$ Bits/s; das Einheitsintervall entspricht $1/2f_g$.

Folgen von Nullen oder Einsen liefern keine Impulse, die zur Taktrückgewinnung genutzt werden können. NRZ kann deshalb nur unter folgenden Bedingungen eingesetzt werden:

- wenn die Signalübertragung von zusätzlichen Taktsignalen begleitet wird (herkömmliche Taktierung, Daten-Strobe-Codierung),
- im Rahmen von Start-Stop-Verfahren (Seite 286),
- wenn zusätzliche Impulse in den Datenstrom eingefügt werden (Bit Stuffing, Gruppencodes).

Hinweis:
Betrachten wir kombinatorische Netzwerke als Signalwege, so entspricht die Signalausbreitung in synchronen Schaltwerken der NRZ-Codierung (die Signale haben nahezu die gesamte Taktperiode Zeit, sich von den Eingängen zu den Ausgängen der Kombinatorik fortzupflanzen; Folgen Nullen und Einsen aufeinander, so ergibt sich jeweils nur eine Pegeländerung (= Signalflanke) je Taktperiode. Die Signalgebung in bestimmten asynchronen Schaltwerken (vgl. Abbildung 2.2) entspricht hingegen RZ; in jeder Signalperiode muß sich ein richtiger Impuls (mit *zwei* Signalflanken) fortpflanzen können. Nehmen wir die Schaltkreistechnologie als gegeben an, so könnte bei vorgegebenem Einheitsintervall eine synchron betriebene Logik eine wesentlich größere Schaltungstiefe haben als eine asynchrone; bei gleicher Schaltungstiefe wäre das Einheitsintervall der asynchronen Hardware merklich größer anzusetzen*). Andererseits muß man bei synchroner Auslegung das Taktraster immer auf den ungünstigsten Durchlauffall abstimmen. Es kommt also - wie so oft - auf die Abwägung der Vor- und Nachteile an. Beim derzeitigen Stand der Technik überwiegen die Vorteile der synchronen Auslegung die Nachteile bei weitem.

*): man könnte beispielsweise von einem Addierwerk mit vorgegebener Verarbeitungsbreite bei synchroner Auslegung in jeder ns ein Ergebnis erwarten, bei asynchroner Auslegung jedoch nur - Spitzfindigkeiten beiseite gelassen - alle 2 ns.

6.2.3. Non Return to Zero and Inversion (NRZI)

Während bei NRZ die jeweiligen Signalwerte (0 oder 1) unmittelbar durch den Signal*pegel* in der jeweiligen Bitzelle wiedergegeben werden, werden bei NRZI die Signalwerte über Pegel*änderungen* codiert.

Es gibt zwei Codierungsweisen (Abbildung 6.16): (1) Invert on Zero, (2) Invert on One. "Invert on ...” bedeutet: nur wenn der genannte Signalwert (z. B. Null bei Invert on Zero) zu übertragen ist, wird der Signalpegel geändert. Bei "Invert on Zero" bedeutet so jede Signal*flanke* eine Null, bei "Invert on One" eine Eins. Tritt in einer Bitzelle keine Signalflanke auf, so enthält diese bei "Invert on Zero" eine Eins, bei "Invert on One" eine Null. Je nachdem, welcher Signalpegel anfänglich vorgelegen hat, ergeben sich so zwei verschiedene (zueinander inverse) Signalverläufe. Deswegen bezeichnet man die Signalbelegungen meist nicht als Null oder Eins, sondern führt Bezeichner ein, die sich vom Üblichen unterscheiden (z. B. J und K in der USB-Spezifikation).

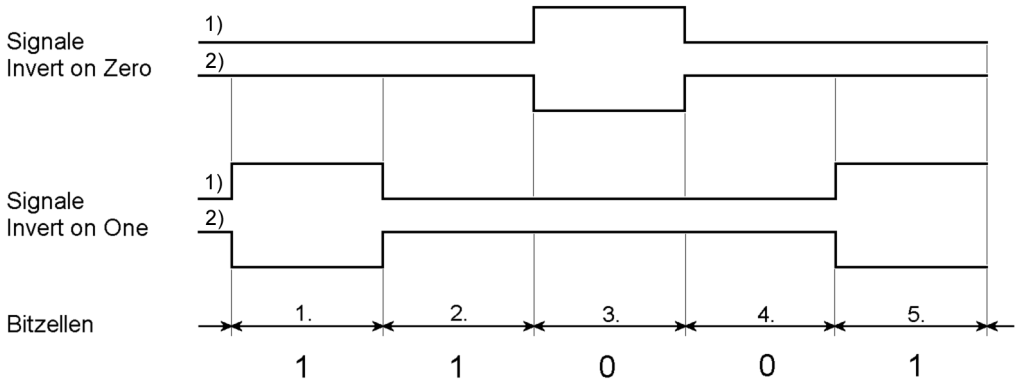

Abbildung 6.16 Signaldarstellungen Non Return to Zero and Inversion (NRZI). 1) - wenn Signal anfänglich Low; 2) - wenn Signal anfänglich High

Der Vorteil: Einer der beiden Signalwerte ist - da als Flanke dargestellt - selbsttaktend. Wechseln sich Nullen und Einsen hinreichend oft ab, so ergeben sich genügend Gelegenheiten für das Nachregeln des Taktes im Empfänger. Betrachten wir zwei typische Einsatzfälle:

- Mehrspur-Bandaufzeichnung: man sorgt (durch entsprechendes Umcodieren) dafür, daß wenigstens in einer Spur eine Belegung vorkommt, die eine Signalflanke bewirkt,
- serielle Übertragung: in den zu übertragenden Bitstrom werden ggf. Bits eingefügt, die Signalflanken hervorrufen (Bit Stuffing). Beispiel: USB. Hier verwendet man Invert on Zero, kombiniert mit Bit Stuffing (Einfügen einer Null nach 6 aufeinanderfolgenden Einsen).

6.3. Frequenzmodulation (FM, MFM)

6.3.1. Herkömmliche Frequenzmodulation (FM)

Jede Bitzelle enthält anfänglich einen Taktimpuls. Eine Eins wird durch einen weiteren Impuls in der Mitte der Bitzelle codiert, eine Null durch Weglassen dieses Impulses (Abbildung 6.17).

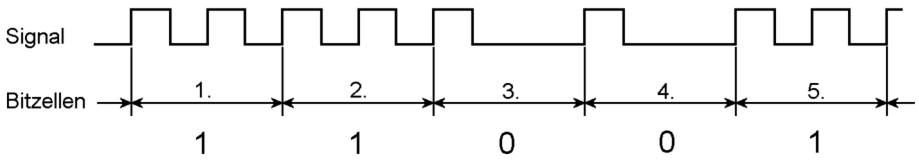

Abbildung 6.17 Frequenzmodulation (FM)

Erklärung des Namens (FM): Folgen von Einsen haben die doppelte Impulsfolgefrequenz ($^2/_T$), verglichen mit Folgen von Nullen ($^1/_T$).

Das Verfahren war bei den ersten Platten- und Diskettenlaufwerken üblich. Es ist im PC-Bereich nicht in Gebrauch. Der Vorteil: Zum Codieren und zur Taktrückgewinnung kommt man mit einfachen Zeitstufen aus. In den 70er Jahren hatten etliche der ersten Mikrocomputer solche Sparlösungen. In den oberen Preisbereichen hat man aber auch hier PLLs eingesetzt. Der Vorteil der PLL: sie läuft auch über gelegentlich fehlende Taktimpulse gleichsam hinweg. Somit liefert der Serdes auf jeden Fall einen vollständigen Datenstrom ab (auf den dann die Steuereinheit oder die Software ggf. Fehlerkorrekturverfahren anwenden kann). Eine Sparlösung mit Zeitstufen liefert hingegen vom ersten fehlenden Taktimpuls an nur Unsinn, der sich beim besten Willen nicht mehr korrigieren läßt.

6.3.2. Modifizierte Frequenzmodulation (MFM)

Bei FM beginnt jede Bitzelle mit einem Taktimpuls. Eine Eins wird durch einen weiteren Impuls in der Mitte der Bitzelle codiert, eine Null durch Weglassen dieses Impulses. Der entscheidende Nachteil: da je Bit ein Taktimpuls mitgeführt wird, brauchen wir für eine bestimmte Datenrate oder Speicherkapazität jeweils die doppelte Impulsfolgefrequenz oder Flußwechseldichte (anders herum gesehen: wir nutzen eine gegebene technisch mögliche Flußwechseldichte bzw. Grenzfrequenz f_g nur zur Hälfte aus). Dieser Nachteil wird beseitigt, indem in einer FM-Bitzelle zwei Nutzbits codiert werden. Wir halbieren also die für FM erforderliche Bitzelle (Abbildung 6.18). Ein Taktbit wird am Anfang der Bitzelle angeordnet, ein Datenbit in deren Mitte. Ein Taktimpuls wird nur dann vorgesehen, wenn in der vorhergehenden und der aktuellen Bitzelle kein Datenimpuls vorhanden ist (wenn diese Bitzellen also Nullen enthalten). Aufeinanderfolgende Einsen werden durch aufeinanderfolgende Datenimpulse dargestellt, aufeinanderfolgende Nullen durch aufeinanderfolgende Taktimpulse. Folgt auf eine "1"-Bitzelle eine "0"-Bitzelle, so enthält die erste einen Datenimpuls und die zweite gar keinen Impuls. Folgt eine "1"-Bitzelle auf eine "0"-Bitzelle, so enthält die "1"-Bitzelle wiederum einen Datenimpuls; die "0"- Bitzelle ist entweder leer oder sie enthält einen Taktimpuls. Die Abstände zwischen den Impulsen entsprechen also:

- T (der Bitzellenlänge) bei Folgen von Nullen oder Einsen,
- $1\ ^1/_2$ T bei Folgen "1001..." oder "0110..",
- 2 T bei Folgen "10101..".

Das entspricht den Impulsfolgefrequenzen f, $^2/_3$ f und $^1/_2$ f (mit $f = {}^1/_T$).

Anwendung: u. a. bei Disketten- und (sehr alten) Festplatten-Laufwerken.

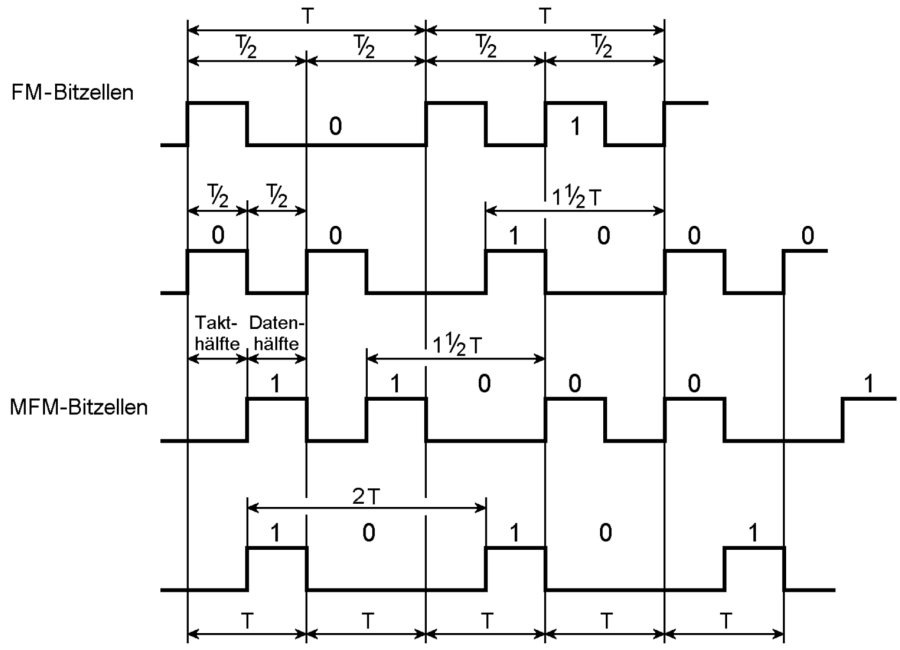

Abbildung 6.18 Modifizierte Frequenzmodulation (MFM)

6.4. Manchester-Codierung

Die kennzeichnende Eigenart der Manchester-Codierung besteht darin, daß in der Mitte einer jeden Bitzelle garantiert eine Signalflanke vorkommt (Abbildung 6.19). Anwendungsbeispiel: 10Base-X-Ethernet.

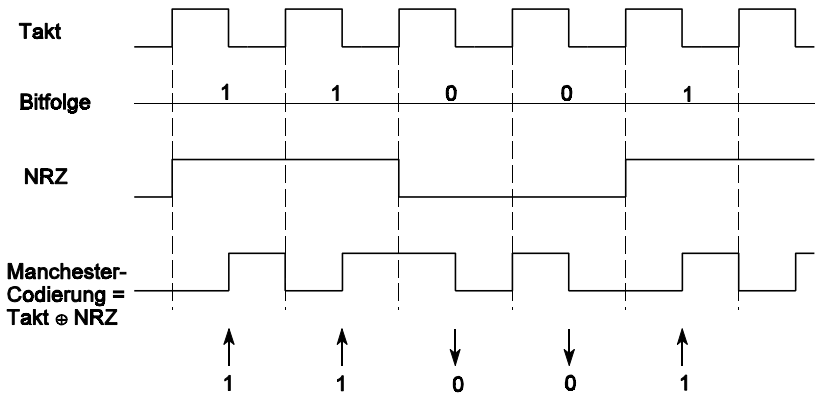

Abbildung 6.19 Manchester-Codierung

Erklärung zu Abbildung 6.19:
Jede Bitzelle wird in zwei Hälften eingeteilt. Die erste Hälfte wird mit dem negierten Wert (Komplement) des jeweiligen Bits belegt, die zweite mit dem Wert des Bits direkt. Die Manchester-Codierung ergibt sich aus der Antivalenzverknüpfung (XOR) der NRZ-Codierung und des zugehörigen Taktes (der 50 % Dury Cycle haben muß). Die Richtung der Signalflanke in Bitzellenmitte ergibt die Signalbelegung: steigende Flanken signalisieren Einsen, fallende Flanken signalisieren Nullen.

Differentielle Manchester-Codierung
Dieses Prinzip (Abbildung 6.20) wird u. a. in Token-Ring-Netzwerken verwendet.

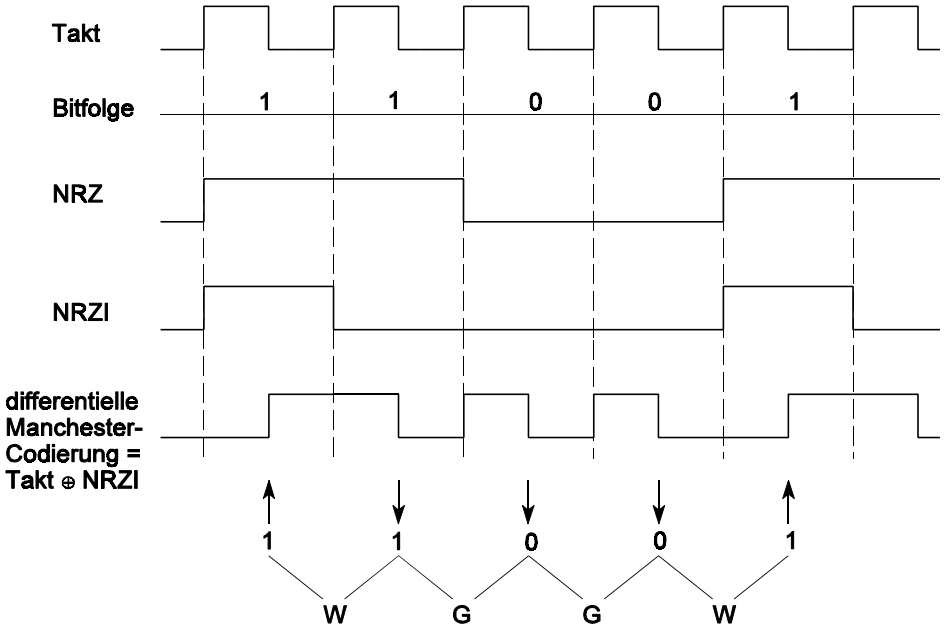

Abbildung 6.20 Differentielle Manchester-Codierung

Erklärung:
Der einzige Unterschied zur gewöhnlichen Manchester-Codierung besteht darin, daß statt des NRZ-Signals ein NRZI-Signal mit dem Takt XOR-verknüpft wird (im Beispiel: Invert on One). Die Signalflanken in Bitzellenmitte wechseln ihre Richtung, sofern Einsen, und behalten ihre Richtung bei, sofern Nullen übertragen werden (W = wechselnde, G = gleiche Richtung).

Weil Taktflanken nicht nur in Bitzellenmitte vorkommen, sondern auch an den Grenzen der Bitzellen auftreten können, haben beide Verfahren der Manchester-Codierung den Nachteil, daß bei einer maximalen Impulsfolgefrequenz f_g die Datenrate höchstens f_g Bits/s betragen kann (Einheitsintervall = $1/f_g$).

6.5. Daten-Strobe-Codierung (IEEE 1394)

Das Interface enthält gesonderte Leitungen für ein Datensignal (D) und ein Strobesignal (S). Die Daten werden in NRZ-Codierung übertragen (also praktisch so, wie sie aus dem Serializer herauskommen (Abbildung 6.21)). Das Strobesignal wechselt seine Belegung dann, wenn sich in zwei aufeinanderfolgenden Taktperioden die Datenbelegung nicht ändert. Damit ist gewährleistet, daß in jeder Bitzelle (t_p) auf jeweils einer der beiden Leitungen eine Signalflanke ($0 \rightarrow 1$ oder $1 \rightarrow 0$) auftritt. Durch Antivalenzverknüpfung beider Signale ($D \oplus S$) kann man den Takt wiederherstellen (es werden beide Taktflanken ausgenutzt). Der Vorteil: während herkömmlicherweise (d. h. bei NRZ-Daten mit begleitenden Taktimpulsen)*) in einer Taktperiode beide Signale schalten können, schaltet hier jedesmal nur eines. Das erlaubt es, deutlich größere Werte für Skew und Jitter zuzulassen. Zudem ist die Codierung (im Sender) und die Decodierung (im Empfänger) recht einfach (Abbildung 6.22).

*): vgl. I^2C, PCI, AGP, SDRAM, Rambus usw.

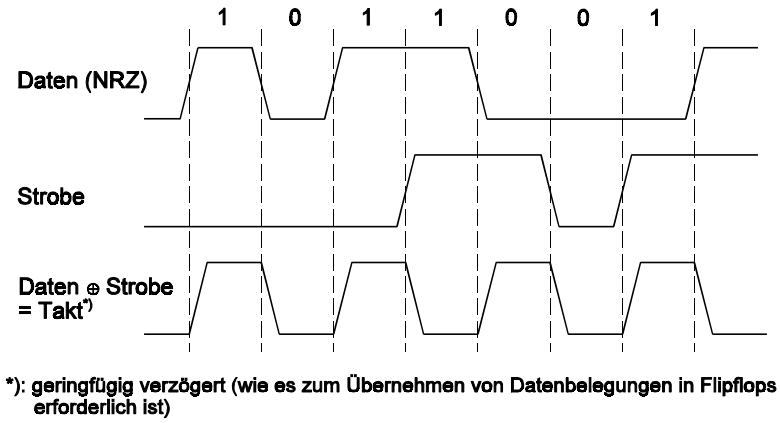

Abbildung 6.21 Daten-Strobe-Codierung

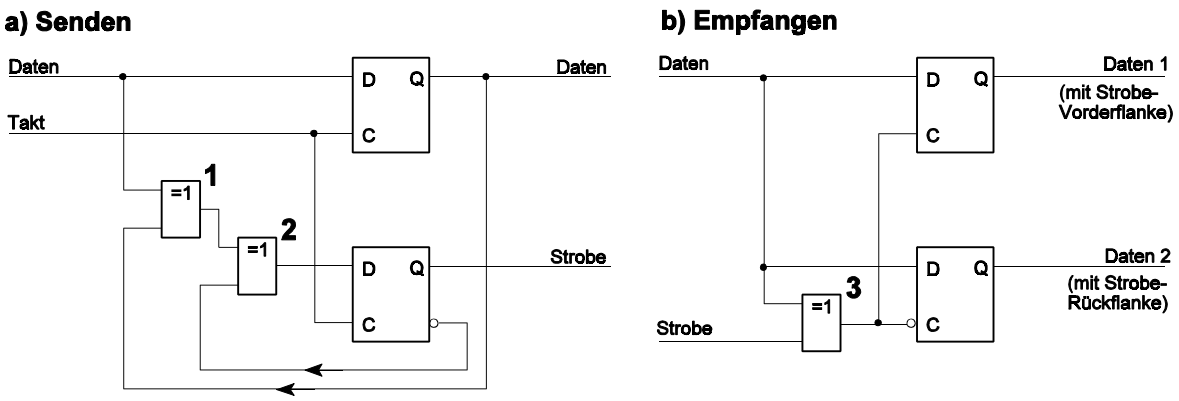

Abbildung 6.22 Die Schaltungstechnik der Daten-Strobe-Codierung (nach: Apple Computer, Inc.). Erklärung der Bezugszeichen und der Wirkungsweise in Kapitel 8, Aufgabe 21

6.6. Gruppencodes

6.6.1. Die RLL-Codes der Plattenspeicher

Von einem magnetischen Speichermedium erhalten wir nur dann ein Lesesignal, wenn zwei verschieden gepolte Elementarmagnete einander abwechseln (Flußwechsel). Herkömmliche Schreibverstärker wirken so, daß sie auf jede High-Low-Flanke des Schreibsignals hin den Schreibstrom umpolen und so im Speichermedium einen Flußwechsel hervorrufen.

FM erzeugt einen oder zwei Flußwechsel je Bitzelle, MFM einen oder gar keinen. Aber auch MFM nutzt das Speichermedium nicht optimal aus, weil das Schreibsignal Daten- und Taktimpulse enthält. Beide Codierungen (FM und MFM) liefern aber beim Lesen auf einfachste Weise Impulse, die der PLL Gelegenheit zum Nachregeln geben (bei FM in jeder, bei MFM schlimmstenfalls in jeder zweiten Bitzelle - der ungünstigste Fall (vgl. Abbildung 6.18) ist die Datenfolge 10101...).

Die RLL-Codierung ermöglicht es, bei gleicher Flußwechseldichte mehr Bits zu speichern als mit MFM. Der Grundgedanke: wir sparen die Taktimpulse ein, fassen die Datenbits in Gruppen zusammen und ergänzen sie durch zusätzliche Bits so, daß von Zeit zu Zeit garantiert Signalflanken (= Flußwechsel) auftreten. RLL bedeutet Run Length Limited. Run Length ("Lauflänge") ist dabei die Anzahl der Nullen (bzw. der ohne Flußwechsel gespeicherten Bits), die aufeinander folgen dürfen. Eine Bezeichnung wie "RLL 2,7" besagt, daß wenigstens 2 und höchstens 7 Nullen aufeinander folgen. Die herkömmlichen RLL-Codes der Festplatten - wir wollen uns hier auf die ursprüngliche (= durchaus noch überschaubare) Auslegung beschränken - beruht auf einer variablen Gruppenaufteilung, wobei die einzelne Gruppe von den Werten der aufeinanderfolgenden Bits bestimmt wird. Um eine solche Codierung abzuleiten, werden die ankommenden Bits nacheinander analysiert (Abbildung 6.23). Dabei wird auf Bytegrenzen keine Rücksicht genommen (es wird beispielsweise ein gesamter Platten-Sektor hintereinanderweg codiert).

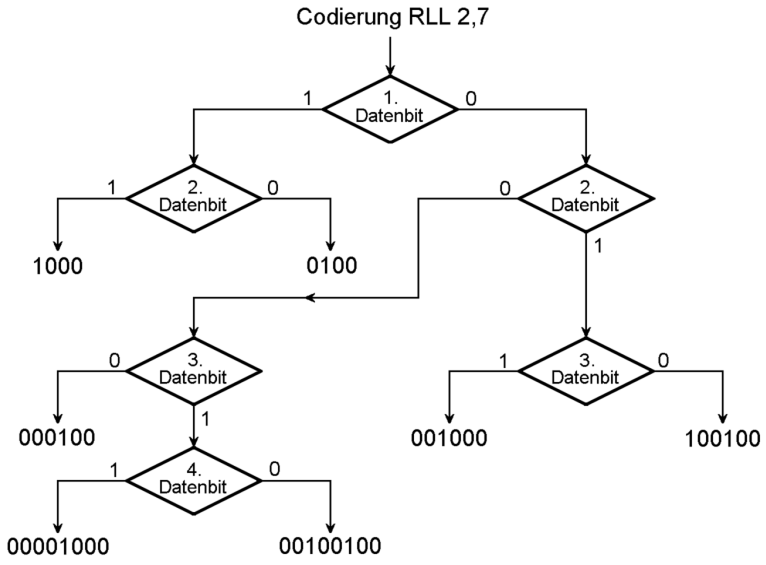

Abbildung 6.23 Codierung gemäß RLL 2,7

Die Anzahl der Code-Bits ist doppelt so groß wie die Anzahl der ursprünglichen Nutzbits. Ist also die Effektivität nicht lediglich genauso gut (oder schlecht) wie bei FM? - Abbildung 6.24 soll veranschaulichen, welche Bedeutung diese Codierung für die magnetische Informationsspeicherung hat.

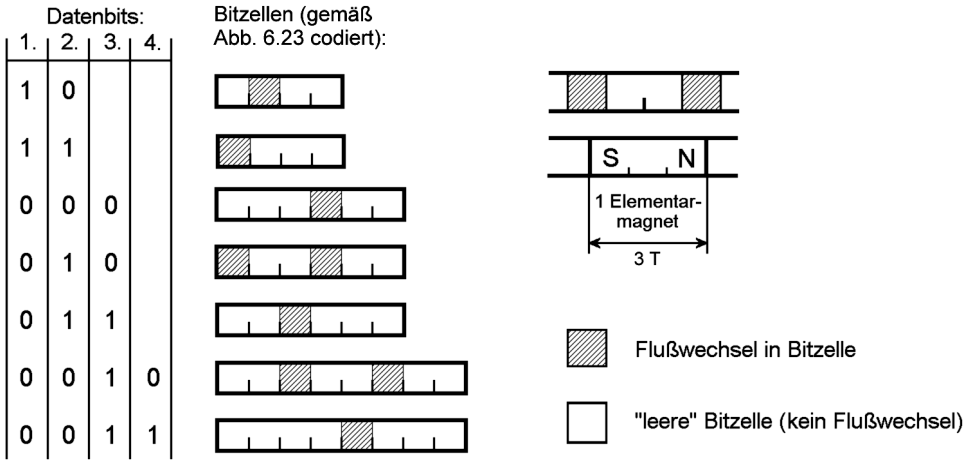

Abbildung 6.24 Flußwechsel bei RLL 2,7

Erklärung:
In der Codierung gemäß Abbildung 6.23 steht eine Eins nicht für einen Impuls, sondern für einen Flußwechsel (mit anderen Worten: für das Umpolen des Schreibstroms). Aus Abbildung 6.24 ist ersichtlich, daß wenigstens zwei weitere Bitzellen zwischen zwei Flußwechseln liegen. Der Abstand zwischen aufeinanderfolgenden Flußwechseln beträgt also insgesamt wenigstens 3 Bitzellen (3 T) Das heißt - anders herum gesehen - ein Elementarmagnet darf sich über wenigstens 3 Bitzellen erstrecken; er entspricht also praktisch 3 Bits.

Vergleichen wir: bei MFM entspricht eine Bitzelle einem Flußwechsel bzw. einem kleinstmöglichen Elementarmagneten (gemäß der Speicherdichte des magnetischen Mediums), bei RLL entspricht ein Flußwechsel bzw. ein solcher Elementarmagnet drei Bits. Kann man auf einer Magnetspur gegebener Länge n kleinstmögliche Elementarmagnete unterbringen, so lassen sich mit MFM n Bits speichern, mit RLL 3 n Bits. Das Verhältnis beträgt also 1 : 3. Bei RLL handelt es sich aber um gruppenweise umcodierte Bits, wobei ein Nutzbit zwei RLL-Bits entspricht. Also müssen wir durch 2 dividieren. Das Ergebnis: RLL erlaubt $^3/_2$ n = 1,5 n Bits zu speichern, ist also um 50 % effektiver als MFM.

6.6.2. Codierung auf CD-ROMs und DVDs: EFM und EFMPlus

Die Information auf CD- und DVD-Datenträgern wird durch das Aufeinanderfolgen von reflektierenden und nichtreflektierenden Flächen (Lands und Pits) codiert. Grundlage der Codierung ist das NRZI-Prinzip; eine Änderung (von Pit zu Land oder umgekehrt) ist eine Eins, keine Änderung - bezogen auf das jeweilige Taktraster - eine Null. Die Pit-Land-Wechsel sind näherungsweise mit den Flußwechseln der magnetischen Speicherung vergleichbar. Und auch hier gibt es das Problem der Taktrückgewinnung: der Lesedatenstrom muß in gewissen Abständen Einsen enthalten, um der PLL Gelegenheit zum Synchronisieren zu bieten. Es sind

aber andere physikalische und technologische Gegebenheiten zu berücksichtigen, so daß die Codes der magnetischen Signalspeicherung nicht einfach übernommen werden können. Die RLL-Codes der CDs und DVDs sind dadurch gekennzeichnet, daß stets 8 Nutzbits (also ganze Bytes) umgeschlüsselt werden (Tabelle 6.2).

Speichermedium	**CD**	**DVD**
Code-Bitpositionen für 8 Nutzbits (insgesamt)	17	16
Codierung	8 auf 14 Bits (Eight-to-Fourteen Modulation EFM)	8 auf 16 Bits (8/16 Modulation (EFMPlus))
Ergänzung des Codes	3 *Merging*-Bits	-
RLL-Codeschema	2, 10 (2...10 Nullen zwischen 2 Einsen)	

Tabelle 6.2 Codierung auf CD- und DVD-Speichermedien

CDs: 8/14-Modulation (EFM)
Jeweils 8 Informationsbits werden auf 14 Bits erweitert, wobei wenigstens zwei und höchstens 10 Nullen zwischen zwei Einsen liegen (EFM = Eight-to-Fourteen Modulation). Die Wandlung erfolgt anhand einer Tabelle, die in den einschlägigen Standards zu finden ist. Eine naheliegende technische Lösung (vgl. Abbildung 6.1): die dem jeweiligen Schieberegister vor- bzw. nachgeordneten Codewandler sind als ROM-Matrizen ausgelegt, die die jeweilige Codetabelle enthalten. Jeder 14-Bit-Struktur werden 3 sog. Merging-Bits vorangestellt. Deren Belegung wird nach einem in den Standards beschriebenen Verfahren so ermittelt, das die folgenden Anforderungen erfüllt werden:

- auch beim Übergang von einem14-Bit-Codewort zum nächsten muß die RLL-Bedingung erfüllt sein (2...10 Nullen zwischen aufeinanderfolgenden Einsen). Die 2,10-Codierung wurde u. a. aus zwei Gründen gewählt: (1) - wie bekannt, um der PLL Gelegenheit zum Vergleichen und Nachregeln zu geben, (2) damit die Pits nicht allzu klein werden. (Die Lands entsprechen der blanken Oberfläche - und die entsteht bei der Fertigung des Rohlings gleichsam von selbst. Die Pits hingegen müssen eingeprägt werden. Es ist offensichtlich, daß man sehr kleine Pits nicht ausreichend genau fertigen kann und daß mit stärkerem Werkzeugverschleiß zu rechnen wäre. (Diese Überlegung gilt sinngemäß für das “Brennen” der Pits im CD-RW-Laufwerk.)
- ein bestimmtes Bitmuster (nämlich 801004H) darf nur als Synchronisationsvorspann (Sync Header) vorkommen, sonst aber nirgends. Wichtig, damit die anfängliche Synchronisation funktioniert.
- der sog. digitale Summenwert muß möglichst gleich, aber wenigstens nahe bei Null sein (dieser Wert wird als Summe aus allen Bits des codierten Datenstromes berechnet, wobei eine Eins als + 1 und eine Null als -1 eingeht). Mit anderen Worten: die Anzahl der Einsen und der Nullen soll - über alles gesehen - möglichst gleich sein. Wichtig für die zuverlässige Auswertung der Lesesignale (wobei wir die elektrotechnischen Spitzfindigkeiten hier übergehen wollen).

DVDs: 8/16-Modulation (EFMPlus)

Jeweils 8 Informationsbits werden auf 16 Bits erweitert, wobei wenigstens zwei und höchstens 10 Nullen zwischen zwei Einsen liegen (EFMPlus = Warenzeichen, das die Weiterentwicklung aus dem Code der CDs andeuten soll). Die RLL-Bedingung (2,10) wurde von der CD übernommen. Man verzichtet aber auf die Merging-Bits. Dafür ist der Code selbst nicht nur um 2 Bits länger, sondern es gibt auch 4 verschiedene Codetabellen, die vier Zuständen (States 1...4) entsprechen. Die Codewandlung erfolgt über eine State Machine mit diesen 4 Zuständen (Abbildungen 6.25, 6.26).

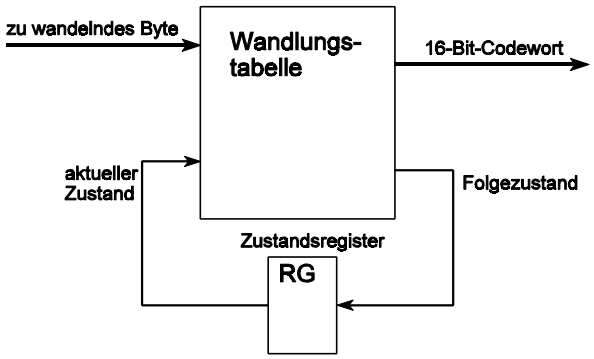

Abbildung 6.25 State Machine zur Codewandlung beim Schreiben (nach: ECMA)

8-bit byte	State 1		State 2		State 3		State 4	
	Code Word msb lsb	Next State	Code Word msb lsb	Next State	Code Word msb lsb	Next State	Code Word msb lsb	Next State
0	0010000000001001	1	0100000100100000	2	0010000000001001	1	0100000100100000	2
1	0010000000010010	1	0010000000010010	1	1000000100100000	3	1000000100100000	3
2	0010000100100000	2	0010000100100000	2	1000000000010010	1	1000000000010010	1
3	0010000001001000	2	0100010010000000	4	0010000001001000	2	0100010010000000	4
4	0010000010010000	2	0010000010010000	2	1000000100100000	2	1000000100100000	2
5	0010000000100100	2	0010000000100100	2	1001001000000000	4	1001001000000000	4
6	0010000000100100	3	0010000000100100	3	1000100100000000	4	1000100100000000	4
7	0010000001001000	3	0100000000010010	1	0010000001001000	3	0100000000010010	1
8	0010000010010000	3	0010000010010000	3	1000010010000000	4	1000010010000000	4
9	0010000100100000	3	0010000100100000	3	1001001000000001	1	1001001000000001	1
10	0010010010000000	4	0010010010000000	4	1000100100000001	1	1000100100000001	1

Abbildung 6.26 Ein kleiner Auszug aus einer Codetabelle (ECMA)

Erklärung:

Die Anordnung von Abbildung 6.25 ist dem Schieberegister des Serializers (vgl. Abbildung 6.21) vorgeschaltet. Gemäß Wandlungstabelle hängt es vom jeweils vorher codierten Byte ab, wie das aktuelle Byte codiert wird (die Codes wurden so gewählt, daß auch zwischen den Codeworten die RLL-Bedingung 2,10 erfüllt ist). Beispiel: das zu wandelnde Byte hat den Wert 10 (die Bytes sind hier dezimal von 0 bis 255 durchnumeriert). Befindet sich die State Machine von Abbildung 6.25 im Zustand 1, so ergeben sich das Codewort 0010010010000000 und der Folgezustand 4.

6.6.3. 100Base-T-Ethernet u. a.: 4B/5B

Die z. B. bei 10Base-X verwendete Manchester-Codierung nutzt die gegebene Bandbreite des Kabels vergleichsweise schlecht aus (Einheitsintervall = $1/f_g$), so daß sie für höhere Geschwindigkeiten nicht mehr in Frage kommt. Die Alternative: NRZ-Signalisierung auf Grundlage von Gruppencodes, die beliebige Nutzbitkombinationen in längere Bitfelder so umsetzen, daß stets die erforderliche Dichte an Signalflanken gewährleistet ist. Bei Netzwerken handelt es sich "nur" um die Ausbreitung elektrischer Signale, so daß man mit weniger Zusatzbits (= Overhead) auskommt als bei den Massenspeichern (es gibt keine mechanische Bewegung und keine Wandlungen zwischen elektrischen und magnetischen bzw. optischen Signalen, es schadet nicht, wenn - in gewissem Maße - Einsen unmittelbar aufeinander folgen usw.).

Es gibt verschiedene Codes: 4B/5B, 5B/6B, 8B/10B usw. 4B/5B eignet sich deshalb gut als Beispiel, weil die Codetabelle vergleichsweise kurz ist (Tabelle 6.3).

Symbol	5-Bit-Code	Bedeutung
0	1110	Datenmuster 0H
1	01001	Datenmuster 1H
2	10100	Datenmuster 2H
3	10101	Datenmuster 3H
4	01010	Datenmuster 4H
5	01011	Datenmuster 5H
6	01110	Datenmuster 6H
7	01111	Datenmuster 7H
8	10010	Datenmuster 8H
9	10011	Datenmuster 9H
A	10110	Datenmuster AH
B	10111	Datenmuster BH
C	11010	Datenmuster CH
D	11011	Datenmuster DH
E	11100	Datenmuster EH
F	11101	Datenmuster FH

Symbol	5-Bit-Code	Bedeutung
I	1111	Füllzeichen, das zwischen den Paketen übertragen wird
J	11000	1. Steuerzeichen in Anfangskennung*)
K	10001	2. Steuerzeichen in Anfangskennung
T	01101	1. Steuerzeichen in Endekennung**)
R	00111	2. Steuerzeichen in Endekennung
V	alle anderen Belegungen	ungültig

*): Start-of-Packet Delimiter; **): End-of-Packet Delimiter

Tabelle 6.3 Codetabelle zur 4B/5B-Codierung

Jeweils 4 Datenbits werden in 5 zu übertragende Bits umgeschlüsselt, um die Taktrückgewinnung im Empfänger zu ermöglichen. Von den insgesamt 32 möglichen Bitkombinationen dienen 16 zur Codierung der 4-Bit-Daten ("Nibbles "). Weitere 5 Kombinationen werden als Füll- oder Steuerzeichen verwendet.

6.6.4. Gigabits/s: 8B/10B

Diese Codierung - aus 8 Nutzbits werden 10 zu übertragende Bits - ist gleichsam Industriestandard, wenn es um höchste Datenraten geht (Escon, Ficon, Fibre Channel, Gigabit Ethernet, ATM, InfiniBand usw.).

Der Code betrifft alle 256 Bitkombinationen eines Datenbytes (Data Code Groups) sowie mehrere (12...ca. 20 sind typisch) zu Steuer- und Synchronisationszwecken verwendete Bitkombinationen (Special Code Groups) und Zeichenfolgen (Sequences). Einsen dürfen unmittelbar aufeinander folgen. Zwei Einsen sind durch höchstens 5 Nullen voneinander getrennt (RLL-Schema 0,5).

Einzelheiten brauchen wir in der Praxis kaum. Wegen der Bedeutung dieses Codes wollen wir aber einige Grundlagen kurz vorstellen (Abbildungen 6.27 bis 6.29).

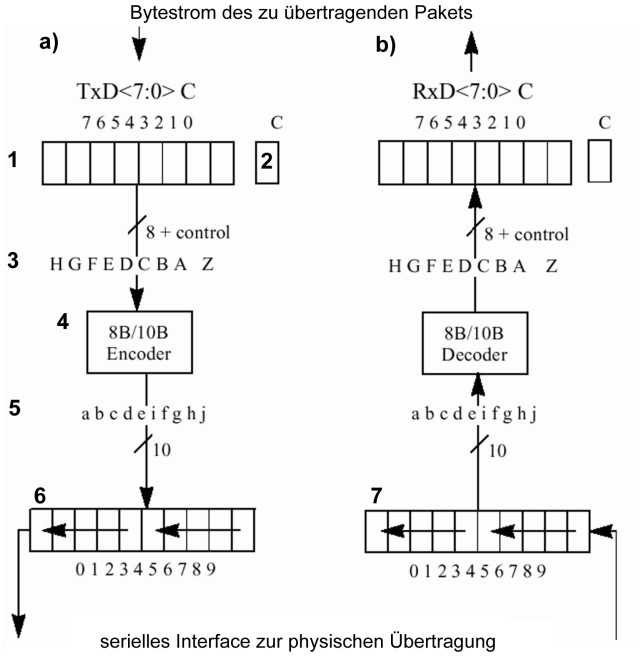

Abbildung 6.27 8B/10B im Übertragungsweg (InfiniBand TA)

Erklärung:
a) - Sendeseite; b) - Empfangsseite; 1 - Byte (z. B. im Datenpaket); 2 - Unterscheidung zwischen Daten und Steuerangaben (Data Codes/Special Codes); 3 - so werden die Bitpositionen des Bytes bezeichnet*); 4 - Wandlungshardware; 5 - so werden die Bitpositionen des 10-Bit-Codewortes bezeichnet*); 6 - Serializer; 7- Deserializer. Das Schema entspricht offensichtlich Abbildung 6.1 - also nichts grundsätzlich Neues. Achten Sie aber auf die Bezeichnung der Bitpositionen und auf gewisse Spitzfindigkeiten (z. B. HGFE..., aber abcd...).

*): in den Codetabellen, Standards, Handbüchern usw.

1	2	3	4	5
Data Byte Name	**Data Byte Value**	**Bits HGF EDCBA**	**Current RD - abcdei fghj**	**Current RD + abcdei fghj**
D0.0	00	000 00000	100111 0100	011000 1011
D1.0	01	000 00001	011101 0100	100010 1011
D2.0	02	000 00010	101101 0100	010010 1011
D3.0	03	000 00011	110001 1011	110001 0100
D4.0	04	000 00100	110101 0100	001010 1011
D5.0	05	000 00101	101001 1011	101001 0100
D6.0	06	000 00110	011001 1011	011001 0100
D7.0	07	000 00111	111000 1011	000111 0100
D8.0	08	000 01000	111001 0100	000110 1011

Abbildung 6.28 Ein kleiner Auszug aus der Codetabelle (InfiniBand TA)

Erklärung:
1 - Bezeichnung der Datenbytes; 2 - Datenbytebelegung (hexadezimal); 3 - die Bitpositionen des Bytes; 4, 5 - beide Varianten des zugehörigen 10-Bit-Codes; RD - Running Disparity.

Die Bitkombinationen sind in Gruppen eingeteilt:

- jedes Byte in eine Gruppe zu 3 Bits (HGF) und in eine zu 5 Bits (EDCBA),
- jedes 10-Bit-Codewort in eine Gruppe zu 6 Bits (abcdei) und in eine zu 4 Bits (fghj).

Die Datenbytebezeichnung in der Codetabelle (1) entspricht dieser Gruppeneinteilung. Erst kommt die 5-Bit-Gruppe, dann die 3-Bit-Gruppe. Beispiel: D8.0: EDCBA = 01000 (= 8); HGF = 000 (= 0).

Die Gruppen werden paarweise getrennt umgeschlüsselt; wir können uns den 8B/10B-Code als Überlagerung zweier einfacherer Codes (Sub-Blocks) vorstellen:

- HGF wird zu fghj (3B/4B-Codierung),
- EDCBA wird zu abcdei (5B/6B-Codierung).

Man vermeidet so große ROM-Zuordner (die sendeseitig über 256 10-Bit-Worte und empfangsseitig 1024 Worte zu wenigstens 9 Bits*) haben müßten). Die kurzen Codetabellen kann man durchaus auch mit optimierten Gatternetzwerken oder mit PLA-Strukturen aufbauen (die schneller als ROMs sind und weniger Platz brauchen).

*): zwecks Unterscheidung zwischen Datenbyte, Steuerzeichen und fehlerhaftem Zeichen.

Weshab zwei Varianten des 10-Bit-Codes?
Wie bei CD und DVD (Seite 303) geht es darum, daß im Datenstrom die Nullen und die Einsen in möglichst gleicher Anzahl vorkommen sollen. Der einschlägige Fachbegriff: gleichstromfreie Übertragung. Die Hardware muß die Einsen und Nullen mitzählen und sich demgemäß für eine der Varianten entscheiden. Zum Rechengang:

- Running Disparity = positiv (RD +) , wenn mehr Einsen als Nullen,
- Running Disparity = negativ (RD -), wenn mehr Nullen als Einsen.

Kommen Nullen und Einsen gleich oft vor, so bleibt der bisherige Wert (RD + oder RD -) erhalten.

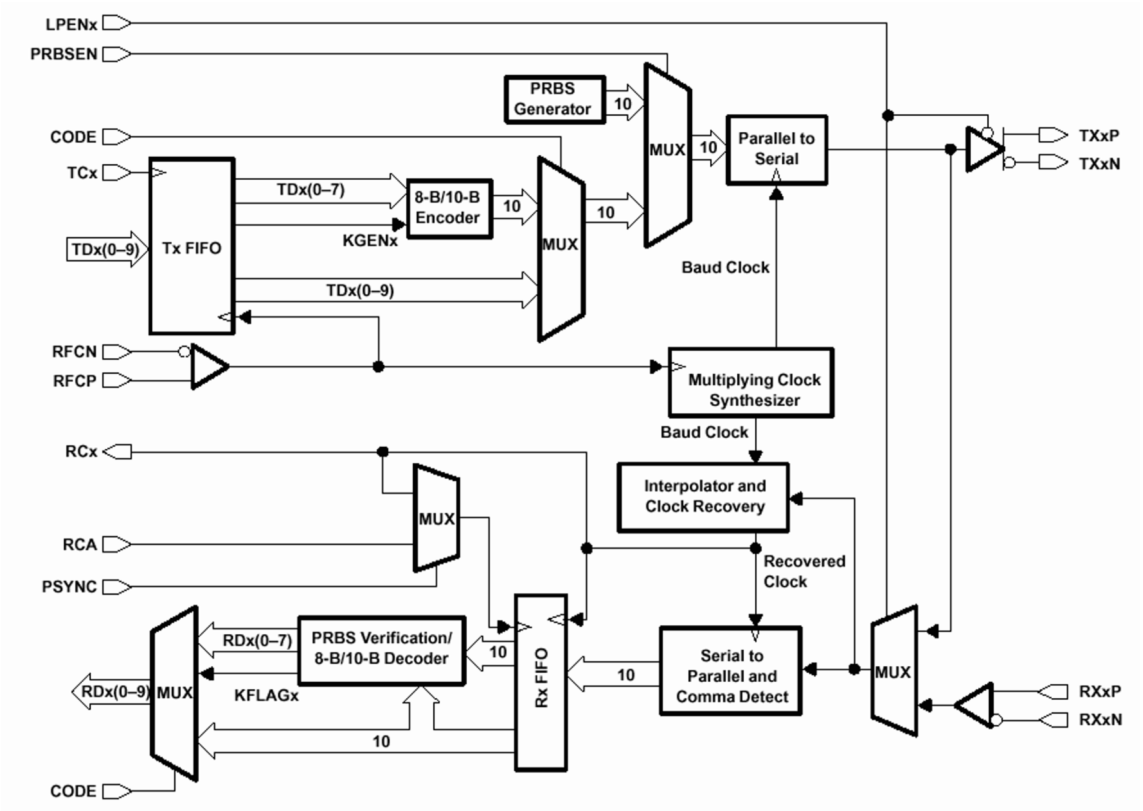

Abbildung 6.29 Eine Serdes-Konfiguration des obersten Leistungsbereichs (Texas Instruments)

Erklärung:
Der Schaltkreis TLK3104SA enthält vier derartige Serdes-Komplexe. Jeder ist für eine Datenrate von 2,5 bis 3,125 GBits/s vorgesehen. Einsatzgebiete: vor allem serielle Hochgeschwindigkeitsverbindungen auf Subsystemebene (Backplane-Verbindungen über Leiterplatten, Kupferkabel oder Glasfaserkabel), Transceiver für 10-Gigabit-Ethernet (medien-unabhängige Schnittstelle mit 4 seriellen Wegen je Richtung bis zum Schaltverteiler oder bis zum Glasfaseranschluß).

1 - Sendedatenweg mit FIFO; 2 - Serializer; 3 - Sendetakterzeugung; 4 - Taktrückgewinnung; 5 - Deserializer; 6 - Empfangsdatenweg mit FIFO. Beide Datenwege 1, 6 sind umschaltbar zwischen (1) Durchreichen der von außen kommenden Daten und (2) eingebauter 8B/10B-Codierung. Die FIFOs sind jeweils 4 Bytes tief.

Beachten Sie die Vorkehrungen zur Selbstprüfung (sehr notwendig, denn bei GHz ist mit Wald- und Wiesen-Prüfgeräten nichts mehr auszurichten): P1- Kurzschlußprüfung (ähnlich Abbildung 6.8); P2 - eingebauter Generator für pseudo-zufällige Prüfmuster; P3 - Soll-Ist-Vergleich für empfangene, von P2 erzeugte Prüfmuster.