Übertragungsrate bei Modems

- Analoges Telefonnetz hat “Bandpasscharakteristik”: Bandbreite von 300 Hz - 3400 Hz
  - ausreichend für verständliche Sprache

- Informationstheoretisches Theorem von Shannon:

$$C = B \log_2(1 + S/N)$$

mit C = Kanalkapazität (in b/s)
B = Kanalbandbreite (in Hz)
S/N = Signal-Geräuschabstand (“Rauschabstand”)

- Theorem gilt unabhängig von der Codierung!

- Beim (analogen) Telefonsystem ist B = 3100 Hz und typw. S/N = 1000 (d.h. 30 dB)

Dezibel als Masseinheit entspricht $10 \log_{10}$ S/N

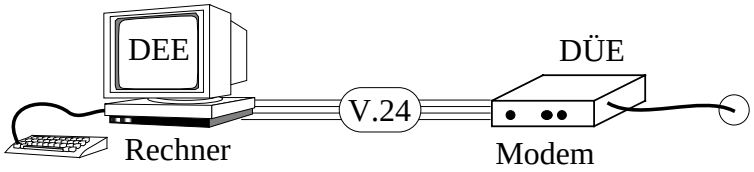
$C = 3100 \log_2(1+1000) = 30894$ b/s

das wäre immerhin ca. 10 Bits pro Hz!

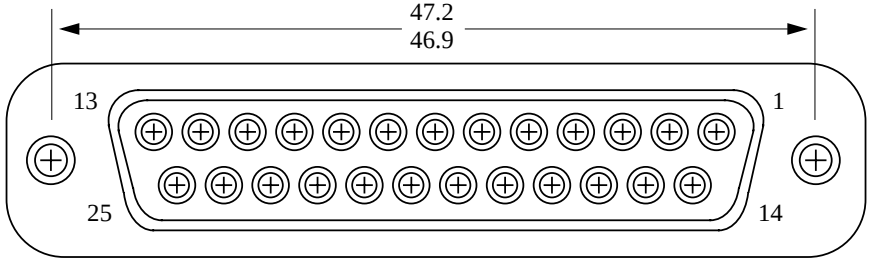
- D.h es kann nicht mehr als ca. 30Kb/s über analoge Telefonleitungen übertragen werden!
  - ohne Berücksichtigung von Störungen (z.B. Übersprechen) und Dämpfung!
  - 28.8Kb/s- bzw. 33.6kb/s-Modem ist also kaum noch zu übertreffen!
  - ausser der Rauschabstand verbessert sich wesentlich (aber wieso?)
  - oder man wendet Tricks wie Datenkompression an
  - bei digitaler Telefonie (ISDN etc.) wird die Übertragungsrate durch die verwendete Abtastnorm für Sprache (ITU G.711) begrenzt ==> 34882 b/s in der Praxis (nur theoretisch bis 39365 b/s)
  - die V.90-Modems umgehen die G.711-Quantisierung in einer Richtung (“downstream” vom Internet-Provider zum Kunden): Internet-Provider schiebt seine Daten “direkt” in das ISDN-Netz

V.24 bzw. RS-232-C

- Beispiel für ein (klassisches) Protokoll des physical layers

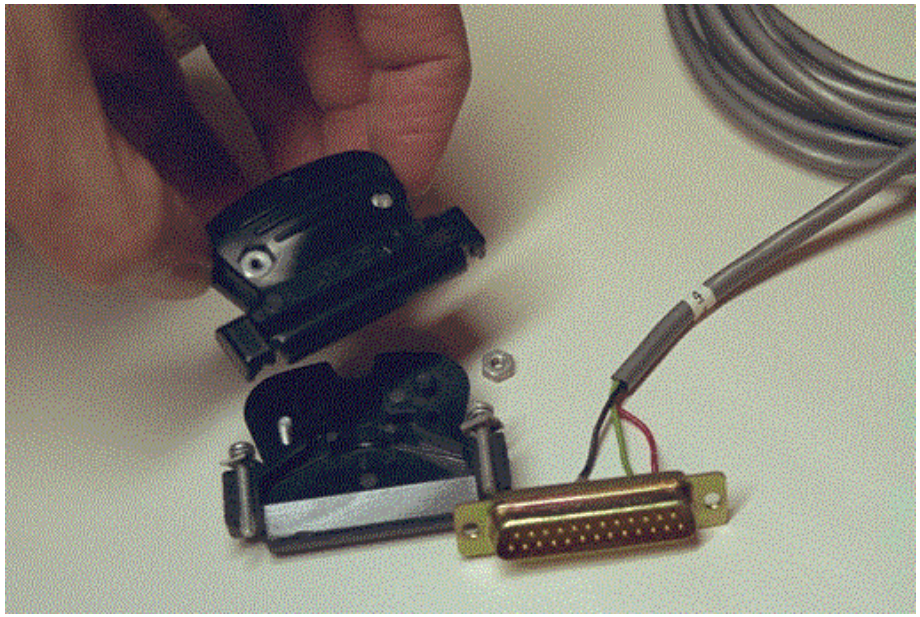


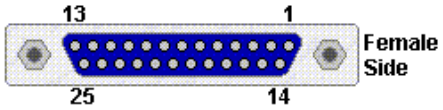
- Spezifikation der elektrischen Signale zwischen DEE und DÜE (Modem) sowie der funktionellen und prozeduralen Einzelheiten
- auch oft für sonstige Datenverbindungen (max 15m) geringer Bitrate (max. 20kb/s) zwischen Rechner und zeichenorientiertem Peripheriegerät
- Spannungsgesteuert: Signalwert = bestimmte Spannung auf einer Leitung (-15 bis -3V = “1”; +3V bis +15V = “0”); festgelegt in V.28



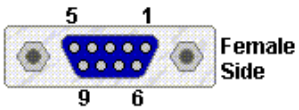
- 25-poliger Stecker (selten mehr als 9 Leitungen verwendet, daher oft auch 9-polige Stecker, insbes. bei PCs!)
- der Stecker wird oft auch für andere Zwecke eingesetzt
- i.a. Flachband- bzw. Mehraderkabel; bitserielle Übertragung (manchmal nur Zweidrahtleitung trotz 25-poligem Stecker!)

RS-232-Stecker



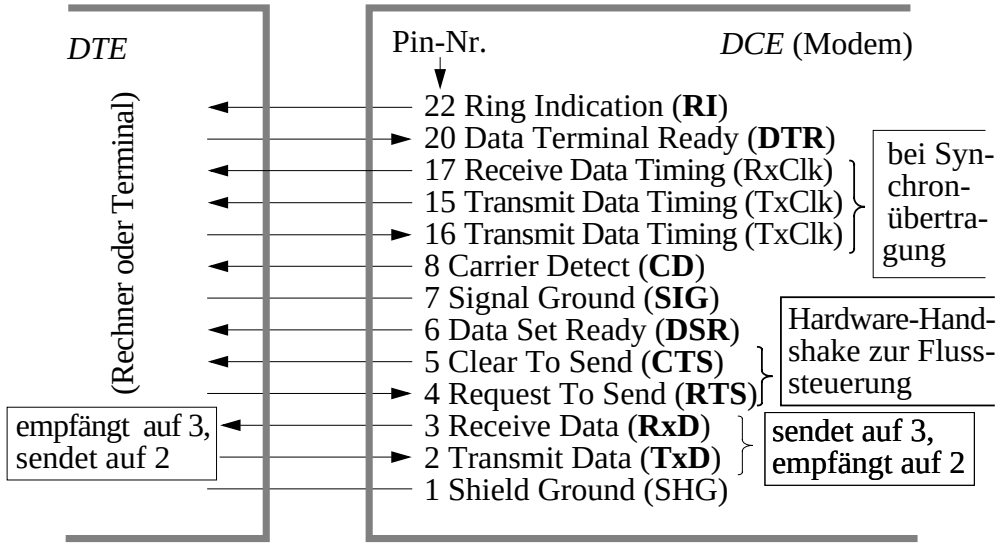


Sub-D 25 (“DB 25”)



Sub-D 9 (“DB 9”)

V.24 / RS-232-C: Signaldefinitionen



- Wichtigste Signalleitungen (andere selten benutzt):
  - DTR = 1, wenn Rechner bzw. Terminal initialisiert wird
  - DSR = 1, wenn Modem initialisiert wird
  - CD = 1, wenn Modem einen Träger auf der Leitung entdeckt
  - RTS = 1, wenn DTE etwas senden möchte
  - CTS = 1, wenn Modem Daten von DTE entgegennehmen kann
  - TxD = gesendete Daten
  - RxD = empfangene Daten
  - SIG = Erdung
  - RI = 1 bei ankommendem Ruf

Diese 9 Leitungen sind auf den 9-Pin-Steckern realisiert; oft genügen sogar noch weniger

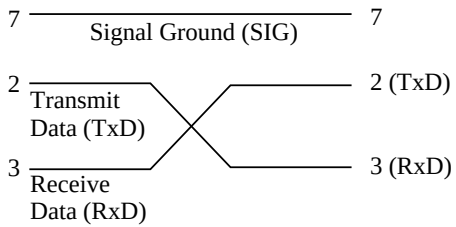
---

- Prozedurale Spezifikation legt erlaubte Abfolge von Ereignissen fest (oft Aktion/Reaktion)
  - z.B.: Sendeanforderung (RTS) von DTE wird durch CTS beantwortet
  - Aktions-/Reaktionsdiagramm bzw. Zustandsdiagramm zur Spezifikation

Nullmodem

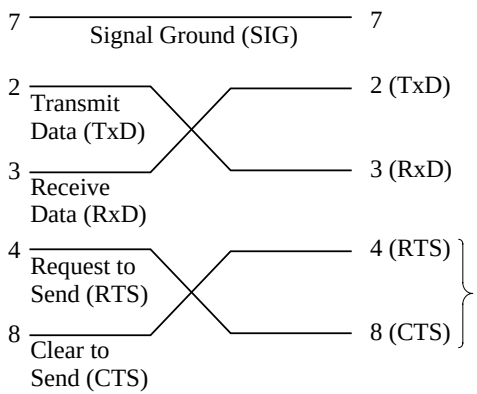
- Koppelung zweier Rechner (z.B. PCs) *ohne* dazwischenliegendes Modem über die RS-232-Schnittstelle
  - durch “Emulation” eines Modems
  - Problem: Ist ein PC ein DEE (DTE) oder ein DÜE (DCE)?

  - Nullmodem: Kurzes Drahtstück, bei dem gewisse Leitungen gekreuzt sind z.B. (Sendeleitung des einen --> Empfangsleitung des anderen)



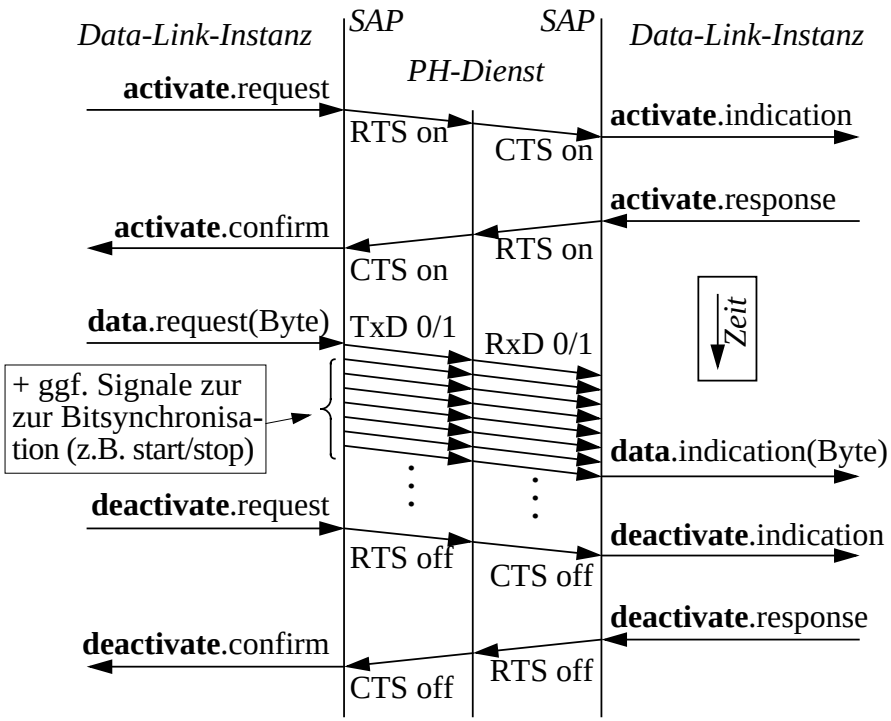
  - RS-232-C-Schittstelle in Rechnern zum direkten Anschluss von Druckern etc. “enthält” oft bereits ein Nullmodem

Protokoll bei Nullmodem-Betrieb



Exemplarisch ein Protokoll mit Diensten “activate”, “deactivate” und “data” der Ebene 2 (Data Link) unter Nutzung von Diensten der Ebene 1 (Physical Layer: “PH”)

Hier verwendet als “handshake” zum Auf-/Abbau einer Verbindung



Digitale Übertragungscodes

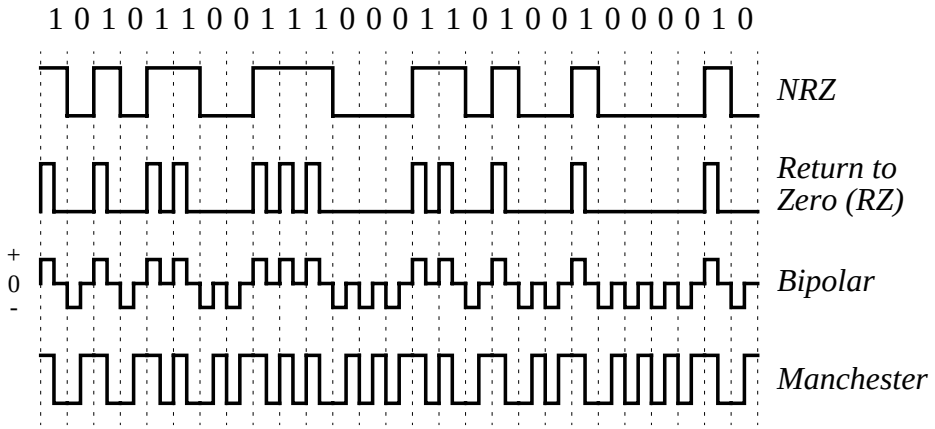
Auch bei digitalen Daten und digitaler Signalübertragung muss i.a. die *Codierung der Daten* in eine für die Übertragung zweckmässige *Codierung des Signals* umgewandelt werden (“Leitungscodierung” oder “Signalcodierung”), z.B. Unempfindlichkeit gegenüber Störungen

---

- Code = Abbildung zwischen Zeichenwert (z.B. “0” oder “1”) und Signalwert (z.B. “-3V” oder “+3V”)

- Codes haben unterschiedliche Eigenschaften, z.B.
  - Fähigkeit zur Resynchronisation oder Taktrückgewinnung aus der ankommenden Signalfolge
  - Redundanz (zur Fehlerekennung / -korrektur auf Signalebene)
  - Signalwechsel auch ohne Information (--> Leitungsüberwachung)

- Hier einige Beispiele:



- Es gibt noch viel mehr Übertragungscodes...
  - z.B. *Differential Manchester* als Variante von Manchester

Übertragungscodes (2)

- Non-Return-to-Zero (NRZ)
  - Einfacher Code: Jedem Bit ein binäres Signal (z.B. versch. Spannung)
  - Viele gleiche Bits --> u.U. hoher Gleichspannungsanteil
  - Keine Taktrückgewinnung möglich

konstante Spannung lässt sich i.a. schlecht übertragen (Hochpasseigenschaft des Mediums)

- Return-to-Zero (RZ)
  - Signal kehrt stets wieder zum Pegel für das 0-Signal zurück
  - Pulsbreiten von halber Bitdauer (--> halbe Bandbreite “verschenkt”!)
  - Signal ändert sich nicht bei längeren 0-Folgen
  - Keine sichere Taktrückgewinnung möglich (ggf. Variante, bei der nach einigen Nullen eine 1 erzwungen wird, die vom Empfänger wieder entfernt wird)

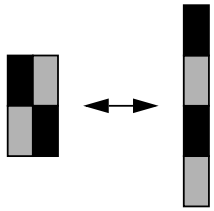
- Bipolar
  - Drei Signalwerte (z.B. ‘+’, ‘0’, ‘-’) notwendig
  - Pulsbreiten von halber Bitdauer
  - Taktrückgewinnung möglich (“self clocking”)

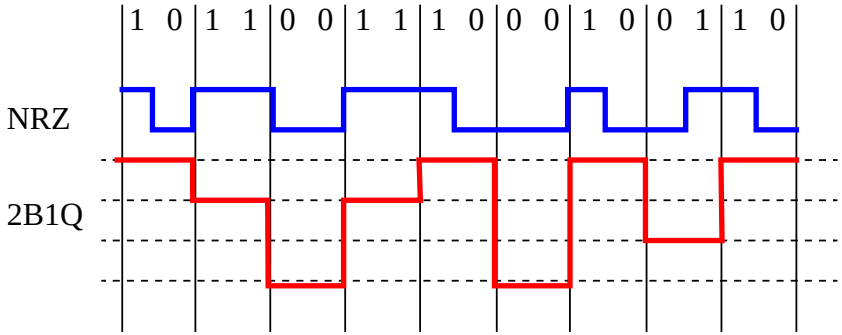
- Manchester
  - Bitperiode in zwei Hälften aufspalten
  - ‘1’: Übergang ‘high’ --> ‘low’ in der Mitte; bei ‘0’ umgekehrt
  - Taktrückgewinnung dadurch möglich
  - Gleich lange ‘high’ / ‘low’-Anteile --> kein Gleichspannungsanteil
  - Verwendung in lokalen Netzen (Ethernet)

Gruppencodierung

- Eine Gruppe von g *Zeichen* wird auf eine Gruppe mit k *Codesymbolen* abgebildet
  - zweistufig: Gruppencodierung wird vor Leitungscodierung angewendet
  - ggf. überzählige Symbole, die keine Zeichen kodieren, können z.B. zur Leitungsüberwachung oder -Steuerung eingesetzt werden (z.B. als Begrenzungssymbole für Datenpakete)
  - auch magnetische Speichermedien und CDs verwenden Gruppencodes

- Beispiel: *2B1Q-Codierung*
  - 2 Binärstellen codieren
  - 1 quaternäres Zeichen übertragen





- Weitere gebräuchliche Gruppencodes, z.B. 4B5B
  - 4 Bitblöcke --> 5 Bitblöcke mit 80% Effizienz

    (32 verschiedene Symbole für 16 Zeichen --> einige der restlichen Symbole verwendet für die Übertragungssteuerung; nicht mehr als 3 konsekutive Nullen in den Symbolen --> Taktrückgewinnung; Verwendung z.B. bei FDDI sowie 100 Base TX und FX-Ethernet)

Quellencodierung

- Auf höherer Ebene (Darstellungsebene) angesiedelt
  - von Leitungscodierung (und Gruppencodierung) zu unterscheiden!

- Ziele:
  - kompakte Datendarstellung (Komprimierung)
  - Hinzufügen von Redundanz

  } z.T. antagonistisch

- Beispiele:
  - Huffman-Codierung
  - Lauflängencodierung
  - Lempel-Ziv-Kompression
  - diskrete Kosinustransformation (DCT)
  - Bildkompression mit JPEG
  - wavelet-Bildkompression
  - fraktale Bildkompression

Synchroner / asynchroner Datenverkehr

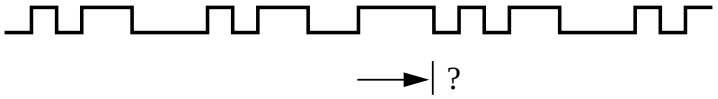
- *Synchronisation* = zeitliche Abstimmung von Sende- / Empfangsstationen zur geregelten Signalübertragung
  - "Gleichlauf" von Sender und Empfänger
  - nach einer Synchronisation sollen Sender und Empfänger die gleiche Vorstellung vom Zustand der Kommunikation haben
  - *Taktsynchronisation* notwendig zur korrekten Interpretation übertragener Signale (Zeitpunkt zum Abtasten des Signals)
  - hierzu verschiedene Techniken: Taktsignal über getrennte Leitung oder Regenerierung des Taktes aus dem Übertragungssignal (dann genügend viele Wertänderungen im Signalstrom notwendig!) und punktuelle Resynchronisation eines unabhängigen Taktgenerators
  - mehrmaliges Abtasten pro Bit, um Abweichungen zu kompensieren
  - Sender und Empfänger müssen sich über Taktrate verständigen

---

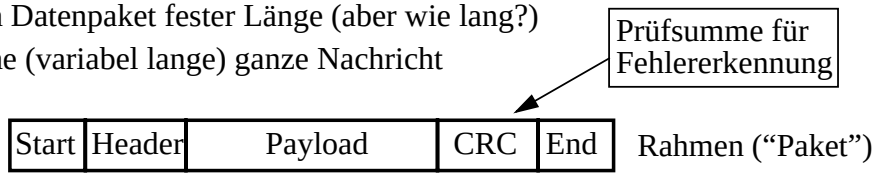
- *Asynchroner Datenverkehr:* Zu übertragende Daten fallen in zufälligen Mengen zu zufälligen Zeiten an
  - "zufällig" = nicht vorhersehbar; unregelmässig
- *Synchroner Datenverkehr:* Daten fallen mehr regulär an
  - "Steigerung": *isochroner* Datenverkehr = gleichmässige Datenrate

Rahmenbildung ("framing")

- Wo beginnt / endet eine Informationseinheit?



- Informationseinheit kann sein
  - ein einzelnes Zeichen (z.B. ASCII)
  - ein Datenpaket fester Länge (aber wie lang?)
  - eine (variabel lange) ganze Nachricht

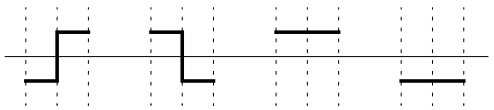


- Header enthält typischerweise die Länge des Pakets, Adressinformation und Steuerinformation (Flags etc.)
- Wozu Informationseinheit bzw. Rahmenbildung?
  - Bündelung für Bitfehler-Prüfung mit CRC
  - Einheit für Quittierung und Flusskontrolle ("bis Rahmen n alle Daten korrekt erhalten!" oder "bitte maximal 7 Rahmen auf einmal!)
  - selektives Wiederholen bei Übertragungsfehlern ("alle Daten ab Rahmen k wiederholen!")
  - Rahmengrenze als Aufsetzpunkt nach Störung

Erkennung von Rahmengrenzen

- Längenangabe im Header
  - Problem: Bei Übertragungsfehlern kann das Längenfeld verfälscht werden; eine Bitfehlererkennung ist ohne Blockgrenzen aber kaum möglich (ggf. in Kombination mit anderen Methoden sinnvoll)

- Verwendung illegaler Codezeichen



z.B. Manchestercodierung: die beiden rechten Zeichen kommen in den Daten nicht vor und können so Paketgrenzen markieren (da "selten": kein Synchronisationsverlust)

- Verwendung von speziellen Steuerzeichen
  - z.B. STX ("start of text") bzw. ETX ("end of text")
  - in den Nutzdaten ("payload") darf kein Steuerzeichen vorkommen
  - "character stuffing" als Lösung für "Datentransparenz":
    - Voranstellen eines Fluchtzeichens ("escape"); i.a. DLE
    - DLE ("data link escape") selbst im Datenteil wird dann verdoppelt
    - dadurch "transparenter Übertragungsmodus"
    - Variante: DLE STX schaltet Transparenzmodus ein; DLE ETX aus

- Verwendung eines speziellen Bitmusters
  - z.B. 01111110 und "bit stuffing" als Lösung für Datentransparenz

- Bei character stuffing oder bit stuffing verlängert sich allerdings die "Nutzlast"
  - insbesondere keine festen Blockgrössen möglich

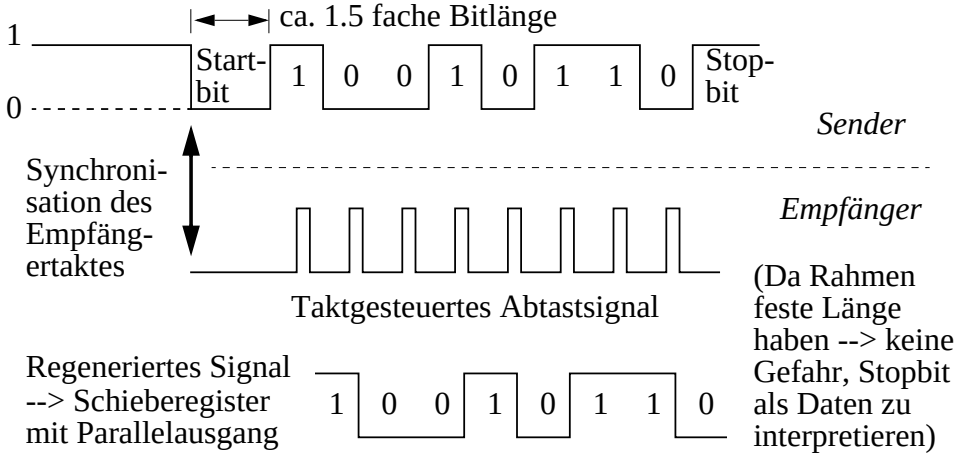
Asynchrone Datenübertragung

- Einzelne Zeichen werden unabhängig voneinander übertragen (jedes Byte "einzeln" in einem Rahmen)
  - beliebig lange Pausen zwischen den Byte-Rahmen ("asynchron")
  - i.a. bitserielle Übertragung; ggf. Paritätsbit am Zeichenende angefügt

- Rahmen ("frame") hat am Anfang bzw. Ende ein Start- bzw. Stopbit zur Synchronisation ("Start-Stop-Verfahren")
  - Synchronisationsoverhead ca. 30% bei einzelnen Bytes!

- Da Rahmen klein, sind die Gleichlaufbedingungen nicht strikt --> einfache, preiswerte Realisierungen
  - alte Technik (Telex-Netze: Rahmen mit ca. 10 Bits)

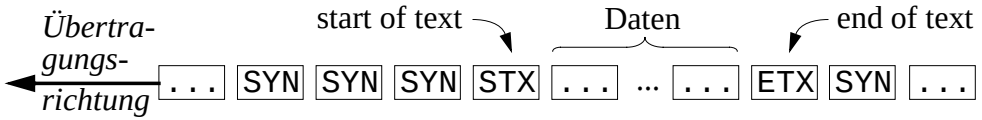
- Klassischerweise eingesetzt u.a. bei Datenübertragung von der Tastatur bzw. ASCII-Terminal zum Rechner
  - lange idle-Zeiten --> Empfänger muss sich bei Beginn des nächsten Zeichens synchronisieren (dazu dient das "Startbit")
  - während einer Zeichenübertragung laufen Sender / Empfänger synchron
  - typische Übertragungsraten: 9600 b/s oder 19200 b/s; z.B. über V.24



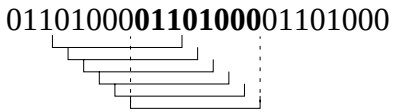
(Da Rahmen feste Länge haben --> keine Gefahr, Stopbit als Daten zu interpretieren)

Synchrone Datenübertragung

- Gleichlauf Sender / Empfänger über lange Zeit garantiert
  - Rückgewinnung des Taktes aus dem Signalwert (oder eigene Taktleitung)
- Damit auch grosse Datenblöcke “am Stück” übertragbar
  - Maximalgrösse von Blöcken notwendig, da Blöcke beim Empfänger zwischengespeichert werden und Fehlererkennung blockweise erfolgt
  - ein Block (auch oft als *Rahmen* oder “*frame*” bezeichnet) kann unabhängig vom letzten Block gesendet werden (d.h. “blockweise asynchron”!)
  - damit Synchronisation aufrecht erhalten werden kann, sollten auch zwischen Blöcken Signale gesendet werden
  - daher Anfang / Ende von Blöcken markieren (“framing”), z.B. so:



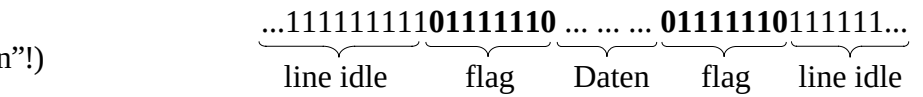
- Vor einem Datenblock z.B. zwei oder mehr SYN-Zeichen
  - ggf. auch zwischen zwei Blöcken ständig SYN-Zeichen senden
  - SYN = “synchronous idle”; Bitmuster **01101000**
  - erhält *Bitsynchronisation* über Blockgrenzen (ggf. wie bei asynchroner Übertragung anhand dieser Bitsignale neu synchronisieren)
  - ermöglicht (nach Bitsynchronisation) die *Zeichensynchronisation*
    - “hunt mode”: Jedes Fenster von 8 Bits auf SYN überprüfen:



    - damit die Byte-Grenzen gefunden (=Zeichensynchronisation)!
  - *Blocksynchronisation* mit STX- und ETX-Zeichen zwischen Folgen ganzer Zeichen (daher auch “zeichenorientiertes Verfahren”)

Bitorientierte Übertragung

- Obiges Verfahren war *zeichenorientiert*
  - spezielle Kontrollzeichen für Blockbegrenzung
- Daneben existiert die *bitorientierte* Übertragung:

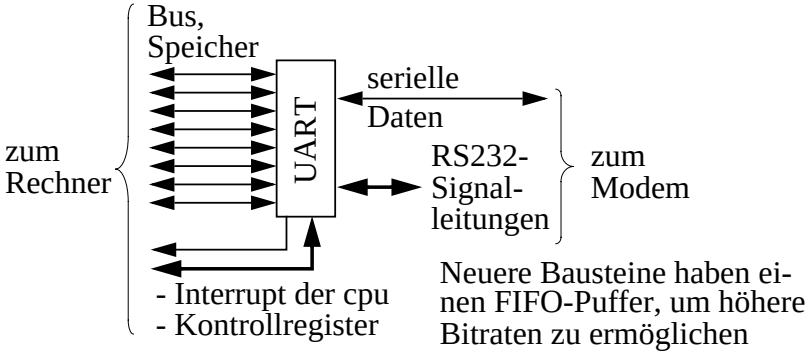


  - Daten werden durch ein flag 01111110 eingerahmt
  - Empfänger sucht (“bitweise”) das flag im Bitstrom
  - Sender fügt in die Nutzdaten nach 5 aufeinanderfolgenden Einsen eine Null ein (“bit stuffing”)
  - Empfänger macht Bitfolge 111110 zu 11111 (löscht die 0)
  - kann (effizient) durch Hardware erledigt werden
  - garantiert Transparenz
- Bitorientierte Übertragung i.a. effizienter als zeichenorientierte
  - beliebige (Binär-)Daten, nicht an Bytegrenzen gebunden, codeunabhängig
  - wegen bit stuffing ist allerdings keine festeBlockgrösse möglich!

UART-Baustein

- “Universal Asynchronous Receiver/Transmitter”
- Schnittstellenbaustein (z.B. für RS232) auf einem IC, der die meisten Funktionen serieller Kommunikation realisiert
- Wesentliche Funktion: Parallel/Seriell-Umwandlung einzelner Zeichen (mit Bit- und Zeichensynchronisation)

  - Bsp.: 8250, 8251, 8450, 16450, 16550A, 16650, 16750...
    (Intel, National Semiconductor...)



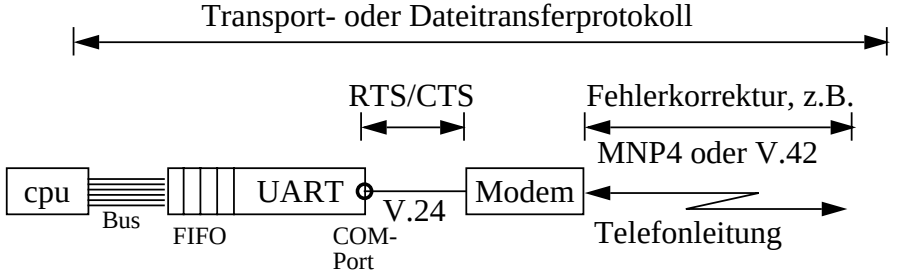
- verwendet zur Ansteuerung von Modems bzw. des “COM-Ports” bei PCs (Signalleitungen TxD, RxD, DSR, DTR, CTS, RTS...) etc.

- “universal” heisst programmierbar für verschiedene Funktionen, i.a. realisiert durch Setzen von Bits in Steuerregistern, z.B.:
  - synchron / asynchron
  - Übertragungsrate
  - Anzahl der Bits pro Zeichen (z.B. 5 oder 8)
  - parity Bit (ja / nein bzw. even / odd)
  - Stop-Bit (ja / nein bzw. Länge von 1, 1.5 oder 2 Bits)

- gemeldet werden (i.a. über ein Statusregister) auch Fehlerbedingungen wie z.B. parity error oder framing error (z.B. ungültiges Stop-Bit, was auf eine falsche Datenrate hindeutet)

“UART Overrun Errors”

- Symptome bei der Modembenutzung:
  - Rechner meldet “CRC-Fehler”, oder
  - Übertragung ist exterm langsam, da durch übergeordnetes Protokoll CRC-Fehler durch Wiederholungen automatisch maskiert werden

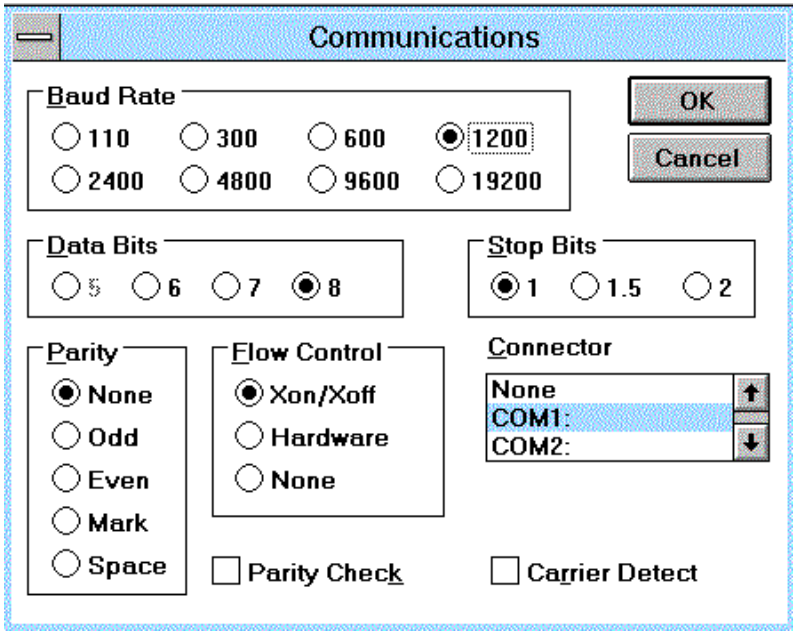


- Mögliche Ursache:
  - Prozessor reagiert zu spät auf Interrupt (z.B. bei Überlastung oder Deaktivierung des Interrupts durch ein Anwendungsprogramm)
  - nächstes ankommendes Zeichen überschreibt alte, noch nicht aus dem Puffer des UART abgeholte Zeichen

- Beispiel: UART-Chip 16550 mit 16-Byte-FIFO-Puffer
  - kann über Software (Treiber des Betriebssystems) konfiguriert werden
  - z.B. Interrupt erzeugen, wenn FIFO mit 14 Byte gefüllt ist
  - bei V.90-Modem und Kompressionsfaktor 4 --> ca. 200kb/s --> restliche 2 Bytes sind schon nach 80 µs verbraucht!
  - UART 16650: 32-Byte-Puffer, 16750: 64-Byte-Puffer

- Lösung: Interrupt früher generieren, FIFO vergrössern, Bus verbreitern, cpu beschleunigen, Flusssteuerung verbessern...

COM - Der “Communication Port”



This is the communications configuration panel from Windows Terminal. It raises a number of interesting questions. Why would someone use 5 bits per character? Because a 5-bit code was used by very early Teletype equipment that was already obsolete in the 1950’s. What is the right number of Stop Bits? Well, if you have a Teletype Model 33, the right answer is 2. If you have a Teletype Model 35, the right answer is 1.5. However, no device built in the last 20 years has needed more than 1 stop bit. What is Xon/Xoff Flow Control? XON and XOFF are byte values. The Teletype had a device to read punched paper tape. The XON character turned the tape reader on, and the XOFF character turned it off. Long after the last paper tape was burned, computers have maintained the tradition that XOFF can optionally mean “stop sending data,” in which case XON means “begin sending again.” What is parity? Before modems did error correction, parity provided a simple mechanism to detect characters corrupted by phone line noise. Today it is unnecessary and is typically disabled.

So in current use, the correct setting for the COM port is always 8-bit characters, no parity, 1 stop bit, hardware pacing and some speed faster that the native transmission speed of the modem. The panel to configure the COM port is left around because everyone is scared to get rid of it.