

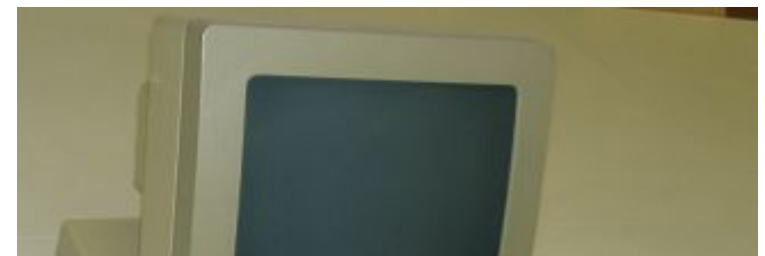
# Terminal-Kodes

1. [Wozu braucht man das?](#)
  1. [Was ist ein Terminal](#)
  2. [Wozu heute noch](#)
2. [Terminals, Emulatoren und Emulationen](#)
  1. ["Dumme" Terminals](#)
  2. ["Richtige" Terminals](#)
  3. [Terminal-Emulatoren](#)
3. [Übersicht der Steuerkommandos VT100 und ANSI](#)
  1. [Was ist überhaupt drin?](#)
  2. [Abfragen](#)
  3. [Schalter](#)
  4. [Steuern und Löschen](#)
  5. [Kursorsteuerung](#)
  6. [Attribute](#)
  7. [Tastatur-Kodes](#)
  8. [Der Grafikzeichensatz](#)
4. [Die Verwendung & das XON/XOFF-Protokoll](#)
  1. [Die Initialisierung](#)
  2. [Das laufende Programm](#)
  3. [Die lästigen Unterschiede](#)
  4. [Das XON/XOFF-Protokoll](#)
5. [Binärdatenübertragung mit XMODEM](#)
6. [Alles zusammen im Mikrocontroller](#)

## 1 Wozu braucht man das?

### 1.1 Was ist ein Terminal

Ein *Terminal* besteht aus Tastatur und Bildschirm sowie einer seriellen Schnittstelle. Tastendrucke gehen zur seriellen Schnittstelle hinaus, Zeichen von außen landen auf dem Bildschirm. Zumindest grundsätzlich.



Es sieht also so ähnlich aus wie ein Computer, ist es aber nicht. (In aller Regel befindet sich darin ein fest programmierter Computer, aber man kann darauf kein beliebiges Programm ausführen.)

Die Erfindung reicht in die Computersteinzeit zurück (die ich *nicht* erlebt habe), als man mit dem Mainframe über ein Modem (ja, genau so wie heutzutage fürs Internet) und über eine Telefonleitung als einer von 'zig Benutzern (etwas von der teuren) Rechenzeit angeknabbert hatte.

Terminals gab es mit Druckerschnittstelle (zum Protokollieren, ein Diskettenlaufwerk gab es nicht) und später auch mit Farbbildschirm. Stets präsentieren sie sich mit einem Zeichenraster aus 80 Spalten und 24 Zeilen (nicht 25 wie unter DOS) und weißer Schrift auf schwarzem Grund, neuere auch umgekehrt. Sie sind so gut wie nie vollgrafikfähig.

Heutzutage sind sie allesamt museumsreif, denn die sog. **Terminal-Emulationen** sind viel verbreiteter und auch einfacher zu handhaben. Als Vorteil von echten Terminals kann man aber herausstellen, dass sie absolut idiotensicher sind, weil es keine Vireninfectionsgefahr gibt.

Übrigens, jeder TELNET-Client ist gleichzeitig eine Terminal-Emulation; die Funktion der seriellen Schnittstelle nimmt nun der TCP/IP-Socket ein (und ist viel schneller).

## 1.2 Wozu heute noch

Mainframes sind so gut wie ausgestorben. Terminal-Emulatoren nicht. In der Ära vor dem Internet brauchte man sie zur Bedienung von sog. Mailboxen (der Name irritiert, sie sind eher mit FTP-Servern vergleichbar). Aber auch diese sind so gut wie tot.

Sehr gerne benutzt man einen Terminal-Emulator für **Mikrocontroller-Projekte** zum Anzeigen interner Werte und (wenn man weiß wie) zu ihrer Fernbedienung. Im Folgenden liegt daher das Schwergewicht auf Mikrocontroller. Sie sollten groß und komfortabel genug sein für C-Programmierung, denn in Assembler artet eine interaktive serielle Schnittstelle in richtig viel Arbeit aus.

Richtig ausprogrammiert kann man eine Mikrocontroller-Lösung basteln, die zur Diagnose nur die serielle Schnittstelle braucht; Setup-Werte u. ä. können damit ebenfalls eingesehen und verändert werden, ohne den Mikrocontroller neu zu brennen oder ein spezielles Kommunikationsprogramm (für welches Betriebssystem?) zu schreiben.

Ähnlich verhalten sich auch Laserdrucker der gehobenen Preisklasse, die über ihren Ethernet-Anschluss ein TELNET-Port anbieten, mit dem man alle Setup-Werte viel klarer einstellen kann als mit dem Mini-Display und den wenigen Tasten am Drucker. (Noch schöner ist es mit einem HTML-



Interface.)

Die Arbeit mit Terminals ist nichts für Leute, die ohne Maus nicht auskommen. Im Zweifelsfall prüfen Sie sich bitte im Selbstversuch und ziehen die Maus vom PC ab und legen los. Wenn Sie 10 Minuten durchhalten, können wir weitermachen.

## 2 Terminals, Emulatoren und Emulationen

### 2.1 "Dumme" Terminals

Vor dem Bildschirmterminal gab es das *Schreibmaschinenterminal*: Tastenanschläge gingen zum seriellen Port, und Daten vom seriellen Port wurden ausgedruckt. Eine Bimmel (für den Kode 07h) gab es damals schon. (Für das Echo bspw. bei der Zeicheneingabe muss der Server sorgen.)

Die ersten *Bildschirmterminals* sparten Papier, aber verhielten sich genau so: Zeichenausgabe nur von links nach rechts und von oben nach unten sowie "Rollbetrieb", was oben herausrollte, war weg. Man nennt solche Terminals "dumm" ("dumb terminal").

Die meisten Unix-Kommandos arbeiten mit "dummen" Terminals, denn die Ausgaben lassen sich hervorragend weiterverarbeiten (filtern) und problemlos ausdrucken.

Es ist aber offensichtlich, dass man so keinen vernünftigen Text-Editor realisieren kann. Dazu bedarf es Steuerkommandos, die auf einen Bildschirm zugeschnitten sind.

### 2.2 "Richtige" Terminals

Offensichtlich benötigt man für das Beispiel Text-Editor in etwa folgende Steuersequenzen:

- Cursor-Positionierung,
- Zeichen einfügen und löschen,
- Zeilen einfügen und löschen (= gezieltes Rollen).

Ohne Zeilen löschen oder einfügen zu können würde der Bildaufbau beim sonst fälligen (fast) kompletten Neuzeichnen unerträglich lange dauern.

Weitere nützliche Steuersequenzen sind:

- Abfrage der Cursor-Position (gleichzeitig Erkennung, dass es kein "dummes" Terminal ist),
- Zeichen-Attribute, wie **fett**, unterstrichen und **Farbe** (je nach Fähigkeiten der Hardware!),
- Zeichensatz-Umschaltung (Rahmensymbole, Umlaute usw.),
- Unterstützung variabler Zeilen- und Spaltenzahl.

Zum Verdruss der Programmierer hat aber jeder Mainframe-Hersteller sein eigenes Süppchen gekocht. Mal sind die Terminals verschieden fähig, und auch bei gleichen Fähigkeiten gibt es völlig verschiedene Steuerkodes.

Zwei Terminal-Systeme sind jedoch (zumindest in ihrer Emulation) vergleichsweise weit verbreitet:

- VT100 - ein Schwarzweiß-Terminal, das als Extra doppelt breite und hohe Schrift darstellen kann,
- ANSI - *kein* reales Terminal, eher ein Standard für Emulationen mit dem Extra von bunter Schrift.

Ihre Steuerkodes sind sogar einander ähnlich, sodass für ein Mikrocontroller-Projekt nur diese beiden Steuersätze interessant sind.

Erwähnenswert ist, dass DOS mit geladenem ANSI-Treiber und umgelenkten Ein- und Ausgabeströmen einem "richtigem" ANSI-Terminal recht nahe kommt, jedoch die Tastaturunterstützung der Sondertasten ist komplett verkehrt.

## 2.3 Terminal-Emulatoren

Man braucht kein echtes Terminal zur Kommunikation mit seinem Mikrocontroller. (Obwohl das natürlich geht!) Dazu gibt es Terminal-Emulatoren. In der Regel können sie verschiedene (reale) Terminals emulieren.

Sowohl erstere als auch letztere verhalten sich unterschiedlich, und so ist die Komplexität quadratisch! Beispielsweise kann ein zeichenbasierter (DOS-) Emulator auch im VT100-Modus keine doppelt hohen oder breiten Zeichen, die VGA-Karte kann das nicht. Manche Steuerkodes werden von dem einen oder anderen Programm nicht oder nur in einem Modus verstanden. Auto-detect-fähige Programme wiederum erwarten bestimmte Sequenzen, bevor sie sich auf einen Modus festlegen.

Die "feinen Unterschiede" herauszufinden und ein Rezept für folgende Mikrocontroller-Programmierungen zu erarbeiten ist Hauptanliegen dieser Arbeit.

Aus Aufwandsgründen wurde die Zahl der getesteten Terminal-Emulatoren ziemlich eingeschränkt durch die Auswahl von:

- Windows 9x/NT/XP HyperTerminal (HYPERTRM.EXE)
- Windows 3.x Terminal (TERMINAL.EXE)
- Norton Commander TERM90.EXE bzw. TERM95.EXE

Als Emulationen kamen nur VT100 und ANSI in Frage. Das Windows-3.x-Programm unterstützt kein ANSI, während HyperTerminal ein Auto-Detect anbietet.

Beiläufig erwähnt werden muss, dass es sich bei diesen Programmen allesamt um **wahre Krücken** handelt, die Exaktheit aller Emulationen lässt sehr zu wünschen übrig. Nur der allgemeinen Verfügbarkeit wegen wurden sie ausgewählt; ein extra Programm herunterladen zu müssen sollte ja gerade

vermieden werden.

## 3 Übersicht der Steuerkommandos VT100 und ANSI

### 3.1 Was ist überhaupt drin?

Bevor es zu den Steuerkommandos geht, erst mal die Grenzen von VT100 und ANSI (in den untersuchten Emulatoren):

- Das Vorhandensein von Farbe kann nicht garantiert werden.
- Die Farb-Standardeinstellung (meist schwarz auf weiß/hellgrau oder **weiß/hellgrau auf schwarz**) ist benutzerspezifisch und kann nicht abgefragt werden; daher sind zur Hervorhebung nur **grün (grün)** und **rot (rot)** zulässig, es sei denn, man übernimmt komplett die Farb-Kontrolle, auch über die Hintergrundfarbe. Denn: **Gelb auf Weiß** sowie **(Dunkel-)Blau auf Schwarz** sind schwer lesbar.
- Der Zeichensatz für die Zeichen  $\geq 80h$  wird oft DOS-kompatibel angenommen; besser fährt man mit einer zuschaltbaren Umkodierung im Mikrocontroller.
- HyperTerminal sorgt schlauerweise dafür, bei ausgewähltem OEM-Zeichensatz auch bei den Umlaut-Tasten OEM-Kodes zur Gegenseite zu schicken.
- Als Sondertasten funktionieren nur **F1..F4**, Pfeiltasten, **Pos1** und **Ende**; die Taste **Entf** geht nur teilweise, und insbesondere fehlen die Tasten BildAuf und BildAb,
- Das VT100-Feature mit den doppelt breiten und hohen Zeichen sollte man besser vergessen,
- Die Vorzüge beider lassen sich in der Emulation **VT220** verbinden, diese ist aber nicht verbreitet.

com-1 - HyperTerminal

Datei Bearbeiten Ansicht Anrufen Übertragung ?

Doppelte Höhe  
Doppelte Breite  
Normale Zeile  
Obere und untere Hälfte:  
Zeichensatz mit ESC ( 0:  
'abcdefghijklmnopqrstuvwxyz{|}~  
♦HFRL°±NV 7 4 ——— H 4 3/4=ö-£. Nur VT100, nicht VT100J  
normal fett unterstrichen+fett unterstrichen invers (unsichtbar)  
rote Schriftfarbe (geht nicht!)  
Lfschen nach links:  
Umlaut-Problem: VT100J: ÄÖÜäöüß  
VT100: (also nur 7-bit-Übertragung)  
Attribut speichern und hier weilerschreiben  
—

vermutlich Emulationsfehler, doppelte Größe wirkt stets zeilenweit  
weilerschreiben  
ESC 7 ESC 3;40H  
ESC 8

Verbunden 01:14:28 VT100J 9600 8-N-1 RF GROSS NUM Aufzeichnen Druckerecho

HyperTerminal im VT100- und VT100J-Modus. Die Maus zeigt die Sequenzen.

```

normale Schrift
Fettschrift
unterstrichen
Blinken (nicht zusammen mit unterstrichen)
invers  invers+fett
          dunkelgrau
dunkelrot rot
dunkelgrün grün
braun gelb
dunkelblau blau
blaurot hell-blaurot
zyan hellzyan
hellgrau weiß
schwarze Schrift auf braunem Grund
gelbe Schrift auf blauem Grund, invers normal
(diese Zeile wurde vorher mit ESC [ K gelöscht)
(Wirkung der Farben auf hellgrauem Grund:)
schwarz dunkelgrau    dunkelrot rot
dunkelgrün grün      braun gelb
dunkelblau blau       blaurot hell-blaurot
zyan hellzyan         weiß

```

Verbunden 00:22:25    ANSIW    9600 8-N-1    RF    GROSS    NUM    Aufzeichnen    Druckerecho

*HyperTerminal im ANSI- und ANSIW-Modus. Die Maus zeigt die Sequenzen.*

Der Kode **ESC** ist hexadezimal **1Bh**, dezimal **27**, als Control-Code **^[** geschrieben und kann auch über die Esc-Taste direkt eingegeben werden.

Weitere Eingabemöglichkeiten bestehen über die Tastenkombination **Ctrl+[** (auf der deutschen Tastatur **Strg+Ü**, weil dort die eckige Klammer der

amerikanischen Tastatur liegt - und deshalb diese Control-Schreibweise), sowie über den numerischen Tastenblock durch Festhalten der linken **Alt**-Taste, Drücken der Tasten **2** und **7** und Loslassen der **Alt**-Taste. (Das sind eigentlich Binsenweisheiten, die es am PC schon seit 20 Jahren gibt, aber kaum jemand weiß das heute noch!)

Die Groß- und Kleinschreibung ist bei den Steuersequenzen unbedingt einzuhalten! Numerische Parameter sind stets dezimal.

An Stelle von "ESC [" kann bei einer 8-bit-Verbindung auch ein "ESC" mit gesetztem Bit 7 ( CSI = 9Bh, 155) übertragen werden; die getesteten Emulatoren interpretieren solche Codes jedoch als Zeichen.

Die wirklich wichtigen und nützlichen Kommandos sind **fett** hervorgehoben. Es bedeuten:

- **HT** = HyperTerminal (unter Windows 95 und höher) in den Modi VT100, VT100J, **ANSI**, **ANSIW** und Auto-Detect
- **W3** = Windwos 3.x TERMINAL.EXE im Modus **VT100**
- **NC** = Norton Commander 4 TERM95.EXE in den Modi **VT100** und ANSI

### 3.2 Abfragen

Befehl	Kode	Antwort *	Emul:	HT	W3	NC	Bemerkung
Identifizierung	ESC [ c ESC Z	ESC [ ? 1 ; x c	VT100 ANSI	OK nein	OK -	OK nein	<b>Auto-Detect</b> zu VT100J $x=2$ unter <b>HT</b> und <b>W3</b> , $x=0$ unter <b>NC</b>
	ESC [ > c		VT100 ANSI	OK nein	nein -	nein nein	
Anwesenheit testen	ESC [ 5 n	ESC [ ? 0 n	VT100 ANSI	OK OK	OK -	nein nein	Auch im Auto-Detect-Modus
<b>Kursorposition abfragen</b>	ESC [ 6 n	ESC [ y ; x R	VT100 ANSI	OK OK	OK -	nein OK	(x:y) = Kursorposition, (1:1) = linke obere Ecke
Kennung ???	ESC [ x	ESC [2;1;1;112;112;1;0x	VT100 ANSI	OK nein	nein -	nein nein	<b>Auto-Detect</b> zu VT100J
Drucker bereit?	ESC [ ? 15 n	ESC [ ? 10 n	VT100 ANSI	nein nein	nein -	nein nein	Offenbar nirgends Drucker-Unterstützung
Benutzerdefinierte Tasten gesperrt?	ESC [ ? 25 n	ESC [ ? 20 n	VT100 ANSI	OK OK	nein -	nein nein	Auch im Auto-Detect-Modus. Antwort: »Tasten nicht gesperrt«
Tastatursprache	ESC [ ? 26 n	ESC [ ? 27 ; 1 n	VT100 ANSI	OK OK	nein -	nein nein	Auch im Auto-Detect-Modus. Antwort: »US-Tastatur«

\* *HyperTerminal sendet nach jeder Antwort einige Bytes Schrott!*



### 3.3 Schalter

Befehl	Kode	Emul:	HT	W3	NC	Bemerkung
Tastatur-Sperre	ESC [ 2 h	VT100	OK	nein	nein	Bei Sperre werden Tastendrücke ohne Pieps ignoriert
Tastatur freigegeben *	ESC [ 2 l	ANSI	nein	-	nein	
Einfüge-Modus	ESC [ 4 h	VT100	OK	OK	OK	Bei Zeichenausgabe rutscht Rest der Zeile nach rechts, aber nie in nächste Zeile
Überschreib-Modus *	ESC [ 4 l	ANSI	OK	-	nein	
Zeilenvorschub (0Ah) bei Zeilenende (0Dh)	ESC [ 20 h	VT100	OK	nein	nein*4	= Schalter »Beim Empfang Zeilenvorschub am Zeilenende anhängen«
kein Zeilenvorschub anhängen *	ESC [ 20 l	ANSI	nein	-	nein*4	
Kursortasten im Application-Modus *2	ESC [ ? 1 h	VT100	OK	OK	nein	<a href="#">Siehe unten</a>
Kursortasten im Cursor-Modus *	ESC [ ? 1 l	ANSI	nein	-	nein	
VT100-Modus *	ESC <	VT100	OK	OK	nein	Wirkung insbesondere auf <a href="#">Kursortasten</a>
VT52-Modus	ESC [ ? 2 l	ANSI	nein	-	nein	
132 Spalten	ESC [ ? 3 h	VT100	OK	nein*4	nein*5	
80 Spalten *	ESC [ ? 3 l	ANSI	nein	-	nein	
Sanftes (langsames) Rollen	ESC [ ? 4 h	VT100	nein	nein	nein	
Springendes (schnelles) Rollen *	ESC [ ? 4 l	ANSI	nein	-	nein	
Schrift schwarz auf weiß *8	ESC [ ? 5 h	VT100	OK	nein	ja*6	Betrifft ganzen Bildschirm. Häufig genutzt als »sichtbare Klingel«. <b>Bug?</b> Die oberen zwei Zeilen bei HyperTerminal bleiben unbeeinflusst
Schrift weiß auf schwarz *	ESC [ ? 5 l	ANSI	nein	-	nein	
Relative Cursor-Platzierung *7	ESC [ ? 6 h	VT100	OK	OK	OK	Cursor springt auf (1:1) = linke obere Ecke (absolut) oder linke obere Rollbereichs-Ecke (relativ)
Absolute Cursor-Platzierung *	ESC [ ? 6 l	ANSI	nein	-	nein	
<b>Umbruch:</b> Bei voller Zeile rutscht der Cursor auf nächste Zeile *	ESC [ ? 7 h	VT100	OK	OK	nein!	Die Standardeinstellung gibt alle Zeichen einer zu langen Zeile auf der rechten Position übereinander aus wie eine dumme Schreibmaschine
<b>kein Umbruch:</b> Bei voller Zeile bleibt Cursor rechts stehen *3	ESC [ ? 7 l	ANSI	OK	-	OK	
Automatische Tastenwiederholung *	ESC [ ? 8 h	VT100	nein	nein	nein	
keine Tastenwiederholung	ESC [ ? 8 l	ANSI	nein	-	nein	
Bildschirm im Zeilensprungverfahren (Interlaced)	ESC [ ? 9 h	VT100	nein	nein	nein	

kein Zeilensprungverfahren *	ESC [ ? 9 1	ANSI	nein	-	nein	Unsinn bei Windows-Emulatoren
Kursor sichtbar *	ESC [ ? 25 h	VT100	OK	nein	nein	Vermisste Funktion!
Kursor unsichtbar	ESC [ ? 25 l	ANSI	nein	-	nein*5	
Grafik-Option(?)	ESC 1	VT100	nein	nein	nein	
normaler Betrieb *	ESC 2	ANSI	nein	-	nein	
Num. Tastenblock im Application-Modus	ESC =	VT100	OK	OK	nein	Siehe <a href="#">Numerischer Tastenblock</a>
Num. Tastenblock im Numeric-Modus *	ESC >	ANSI	nein	-	nein	
Zeichensatz G0 - Vorwahl	ESC ( x	VT100	OK*9	OK**	OK	x: A = britisch (#=£), B = amerik. *, 0 od. 2 = grafisch <a href="#">Siehe unten</a>
Zeichensatz G1 - Vorwahl	ESC ) x	ANSI	nein	-	nein	
Zeichensatz G0 nehmen *	SI (=0Fh, 15, ^O)	VT100	OK*9	OK	OK	Wer hat sich bloß diesen Müll ausgedacht? Siehe <a href="#">Grafik</a>
Zeichensatz G1 nehmen	SO (=0Eh, 14, ^N)	ANSI	nein	-	nein	

\* Standardeinstellung (bei den übrigen Emulatoren)

\*2 Standardeinstellung in **HT** im VT100J-Modus

\*3 Standardeinstellung in **HT**

\*4 Obwohl Schalter im Menü oder Konfigurationsdialog verfügbar

\*5 Sichtbare Steuerzeichenausgabe

\*6 Lässt sich nicht zurückstellen, Wirkung nur auf neue Zeichen (Bug!)

\*7 Standardeinstellung in **NC** (nur relevant im VT100-Modus)

\*8 Standardeinstellung in **HT** und **W3**

\*9 Nicht im VT100J-Modus! Nur im VT100-Modus. (VT100J ist der 8-bit-Modus und braucht keine Zeichensatzumschaltung)

\*\* Keine Funktion der Amerikanisch/Britisch-Umschaltung; diese ist im Konfigurationsdialog festgelegt

### 3.4 Steuern und Löschen

Befehl	Kode	Emul:	HT	W3	NC	Bemerkung
Terminal rücksetzen	ESC c	VT100 ANSI	OK nein	OK* -	nein nein	<b>Auto-Detect</b> zu VT100J
Zeile doppelt groß und obere Hälfte darstellen	ESC # 3	VT100 ANSI	OK nein	OK*2 -	nein nein*3	Das Programm sollte die gleichen Buchstaben ausgeben!
Zeile doppelt groß und untere Hälfte darstellen	ESC # 4					
Zeile normal	ESC # 5					Voreinstellung

Zeile doppelt breit	ESC # 6					Doppelt groß ist stets auch doppelt breit Die 40 (66) ersten Zeichen sind sichtbar, die anderen fallen heraus
Bildschirm mit "E"s füllen (Bildschirm-Test)	ESC # 8	VT100 ANSI	OK nein	OK -	nein nein	
ESC-Sequenz abbrechen	CAN (=18h, 24, ^X) SUB (=1Ah, 26, ^Z)	VT100 ANSI	? ?	? -	OK OK	ungetestet
<b>Piep</b> ausgeben	BEL (=07h, 7, ^G)	VT100 ANSI	OK OK	OK -	OK OK	Ton ist oft abschaltbar
Tastatur-Leuchtdioden schalten	ESC [ x q	VT100 ANSI	nein nein	nein -	nein nein	$x$ = Nummer der Leuchtdiode, 0 = aus
Einfügen von Zeilen, Runter-Rollen	ESC [ n L	VT100 ANSI	OK OK	OK*4 -	nein nein	$n=0$ oder weggelassen = eine Zeile Es wird nach bzw. von unten gerollt
Löschen von Zeilen, Hoch-Rollen	ESC [ n M	VT100 ANSI	OK OK	nein -	nein nein	
<b>Löschen</b> von Zeichen	ESC [ n P	VT100 ANSI	OK OK	OK -	OK nein	$n=0$ oder weggelassen = ein Zeichen Zeichen rutschen von rechts nach
<b>Löschen</b> vom Cursor zum Zeilenende	ESC [ K	VT100 ANSI	OK OK	OK -	OK OK	Kursorposition bleibt unverändert. Das Zeichen unter dem Cursor wird stets mit gelöscht
<b>Löschen</b> vom Zeilenanfang zum Cursor	ESC [ 1 K		OK OK	-	OK	
<b>Löschen</b> der Zeile, die Cursor enthält	ESC [ 2 K	VT100 ANSI	OK OK	OK -	OK OK*4	
<b>Löschen</b> vom Cursor nach rechts und bis zum Bildende	ESC [ J	VT100 ANSI	OK OK	OK -	OK OK	
<b>Löschen</b> vom Bildanfang und von links bis zum Cursor	ESC [ 1 J		OK OK	-	OK	
<b>Löschen</b> des Bildschirms	ESC [ 2 J	VT100	OK	OK*5	OK*5	Kursorposition danach (1:1), LF = wie LineFeed
		ANSI	OK	-	OK	
	FF (=0Ch, 12, ^L)	VT100 ANSI	LF OK	LF -	OK OK	

\* Bildschirm wird gelöscht

\*2 Fehldarstellung: Die darunterliegende Zeile wird auch bemalt

\*3 Sichtbare Steuerzeichenausgabe

\*4 Kursorposition danach links

\*5 *Kursorposition danach* (1:1)

### 3.5 Kursorsteuerung

Befehl	Kode	Emul:	HT	W3	NC	Bemerkung
<b>Position und Attribute speichern</b>	ESC 7	VT100 ANSI	OK OK	OK* -	OK nein	
	ESC [ s	VT100 ANSI	OK OK	nein -	nein OK*	Nur diese Kombination geht mit DOS' ANSI.SYS
<b>Position und Attribute wiederherstellen</b>	ESC 8	VT100 ANSI	OK OK	OK* -	OK nein	
	ESC [ u	VT100 ANSI	OK OK	nein -	nein OK*	Nur diese Kombination geht mit DOS' ANSI.SYS
<b>Roll-Bereich</b> setzen (immer volle Breite!)	ESC [ yI ; y2 r	VT100 ANSI	OK OK	OK -	OK nein	yI, y2 = Roll-Bereich (Standard 1,24), setzt Kursor auf (1:1) (linke obere Ecke oder Roll-Ecke je nach <a href="#">Relativ-Modus</a> )
Roll-Bereich löschen	ESC [ r	VT100 ANSI	OK OK	OK -	nein nein	
<b>Kursor positionieren</b>	ESC [ y ; x H ESC [ y ; x f	VT100 ANSI	OK OK	OK -	OK OK	Linke obere Ecke (bzw. Roll-Rand im <a href="#">Relativ-Modus</a> ) ist (1:1) Weglassen von x führt manchmal zu (1:y)
<b>Kursor nach links oben</b>	ESC [ H ESC [ f					Erforderlich nach VT100' ESC [ 2 J
Tabulator an Kursorpalte setzen	ESC H					Standardmäßig ist alle 8 Spalten ein Tabulator.
Tabulator an Kursorpalte löschen	ESC [ g	VT100 ANSI	OK nein	OK -	OK nein	
Alle Tabulatoren löschen	ESC [ 3 g					Auch alle Standard-Tabulatoren werden gelöscht!
Kursor y Zeilen hoch	ESC [ y A					Niemals rollen. Wenn x/y = 0 ist oder weggelassen wird, wird dennoch um eine Zeile/Spalte bewegt. Beim Erreichen von Bildschirmrändern bleibt der Kursor dort stehen.
Kursor y Zeilen runter	ESC [ y B					
Kursor x Spalten rechts	ESC [ x C					
Kursor x Spalten links	ESC [ x D					
Kursor zum <b>Zeilenanfang</b>	CR (=0Dh, 13, ^M)	VT100 ANSI	OK OK	OK -	OK OK	Text bleibt stehen! Ein Terminal-Programm kann automatisch Zeilenvorschub (LF) ausführen.
Kursor <b>1 Zeichen nach links</b>	BS (=08h, 8, ^H)					Text bleibt stehen - Nur zur Erinnerung! Bei TAB ohne Tabulator springt der Kursor ans

Kursor zum <b>nächsten Tabstopp</b>	HT (=09h, 9, ^I)					Zeilenende wie bei einer dummen Schreibmaschine, bei NC ist TAB dann wirkungslos.
Nächste Zeile, <b>Hoch-Rollen</b> beim Erreichen des Roll-Randes	LF (=0Ah, 10, ^J) VT (=0Bh, 11, ^K)					Kursor-Spalte unverändert; ein Terminal-Programm kann automatisch Wagenrücklauf (CR) ausführen.
	ESC D					Kursor-Spalte unverändert
	ESC E	VT100 ANSI	OK OK	OK -	OK nein	Kursor-Spalte danach stets = 1
Vorherige Zeile, <b>Runter-Rollen</b> beim Erreichen des Roll-Randes	ESC M					

\* *Speichert Attribute nicht*

### 3.6 Attribute

Modus	Kode	Emul:	HT	W3	NC	Bemerkung
<b>Zeichenattribute</b> setzen	ESC [ x ; y ; z m	<b>siehe unten</b>				x,y,z sind (beliebig viele) Parameter, siehe nachfolgend. <b>Auto-Detect</b> zu ANSIW bei >=30
<b>alles</b> AUS	ESC [ 0 m ESC [ m	VT100 ANSI	OK OK	OK -	OK OK	stellt hellgraue Schrift auf schwarzem Grund ein
<b>fett</b> EIN	ESC [ 1 m					VT100: fette Schrift ANSI: hellere (bspw. weiße) Schrift
<b>unterstrichen</b> EIN	ESC [ 4 m					NC: dunkelblaue Schrift wie ESC [ 34 m
<b>Blinken</b> EIN	ESC [ 5 m					W3: Fettschrift wie ESC [ 1 m
<b>Invers</b> EIN	ESC [ 7 m					Vertauscht Vorder- und Hintergrundfarbe, vergisst Intensität
unsichtbar	ESC [ 8 m	VT100 ANSI	OK OK	nein -	nein nein	Vordergrundfarbe = Hintergrundfarbe, vergisst Intensität
<b>fett</b> AUS	ESC [ 22 m	VT100 ANSI	OK OK	nein -	OK OK	ANSI: dunkle Schrift
<b>unterstrichen</b> AUS	ESC [ 24 m					
<b>Blinken</b> AUS	ESC [ 25 m					
<b>Invers</b> AUS	ESC [ 27 m					
sichtbar	ESC [ 28 m	VT100 ANSI	OK OK	nein -	nein nein	
	ESC [ 30 m					Vordergrund schwarz / dunkelgrau
	ESC [ 31 m					Vordergrund dunkelrot / rot

<b>Textfarbe</b>	ESC [ 32 m	VT100 ANSI	nein OK	nein -	OK OK	Vordergrund dunkelgrün / grün
	ESC [ 33 m					Vordergrund braun / gelb
	ESC [ 34 m					Vordergrund dunkelblau / blau
	ESC [ 35 m					Vordergrund blaurot / hell-blaurot
	ESC [ 36 m					Vordergrund zyan / hellzyan
<b>Hintergrundfarbe der Zeichenzelle</b>	ESC [ 37 m					Vordergrund hellgrau* / weiß
	ESC [ 40 m					Hintergrund schwarz*
	ESC [ 41 m					Hintergrund dunkelrot
	ESC [ 42 m					Hintergrund dunkelgrün
	ESC [ 43 m					Hintergrund braun
	ESC [ 44 m					Hintergrund dunkelblau
	ESC [ 45 m					Hintergrund blaurot
	ESC [ 46 m					Hintergrund zyan
	ESC [ 47 m					Hintergrund hellgrau

\* Standardeinstellung, jedoch unter **HT** nicht Vorgabe. Dort ist der Hintergrund hellweiß und kann, einmal verstellt, mit keiner Sequenz wieder so eingestellt werden.

### 3.7 Tastatur-Kodes

In der umgekehrten Richtung muss das Mikrocontroller-Programm nicht nur die Antwort-Strings der Abfragen (Kursor-Position) verarbeiten, sondern auch die Sondertasten, die es auf einem Schreibmaschinenterminal noch nicht gab. Die »höheren« Funktionstasten sind hier aufgeführt, weil Tasten bisweilen belegt und so VT200-kompatibel gemacht werden können. Für 8-bit-Mikrocontroller ist eine sinnvolle und häufig genutzte Transkodierung in der Spalte **8bit** angegeben.

Taste	Kode*				Emul:	HT	W3	NC	Bemerkung
	Cursor Mode	App Mode	VT52	8bit					Man tut gut daran, einfach alle Modi zu unterstützen
<b>Pfeil hoch</b>	ESC [ A	ESC O A	ESC A	^P	VT100 ANSI	OK OK	OK -	OK OK	Wirken bei Echo (= Zurücksendung zum Terminal) schlauerweise genau so wie erwartet.
<b>Pfeil runter</b>	ESC [ B	ESC O B	ESC B	^N					
<b>Pfeil rechts</b>	ESC [ C	ESC O C	ESC C	^F					
<b>Pfeil links</b>	ESC [ D	ESC O D	ESC D	^B					

Pos1	ESC [ H			^A	VT100 ANSI	nein OK	? -	OK OK	Als Bildschirmsteuerkode geht der Cursor nach (1:1)
Ende	ESC [ K			^E	VT100 ANSI	nein OK	? -	ESC [ J ESC [ J	Als Bildschirmsteuerkode wird der Rest der Zeile ab Cursor gelöscht
Rückschritt	BS (=08h, 8, ^H)				VT100 ANSI	OK OK	OK -	OK OK	Nur zur Erinnerung! Das Echo hat andere Wirkung als unter DOS/Windows!
	ESC [ 1 ~			^H	VT200			Nie gesehen	
Tabulator	TAB (=09h, 9, ^I)				VT100 ANSI	OK OK	OK -	OK OK	Nur zur Erinnerung!
Enter	CR (=0Dh, 13, ^M)								
Strg+Enter	LF (=0Ah, 10, ^J)								
Esc	ESC (=1Bh, 27, ^[]		2x: ^C						
Find	ESC [ 2 ~		7Fh, 127		VT200				Nicht auf der PC-Tastatur
Einfg	ESC [ 2 ~								Ggf. mit diesem Kode belegen
Entf	ESC [ 3 ~								
	DEL (=7Fh, 127)				VT100 ANSI	OK nein	OK -	OK OK	
Select	ESC [ 4 ~		^U		VT200				Nicht auf der PC-Tastatur
BildAuf	ESC [ 5 ~								Ggf. mit diesem Kode belegen
BildAb	ESC [ 6 ~								^V
F1	ESC O P		ESC P	^\\ = 1Ch	VT100 ANSI	OK OK	OK -	nein nein	
F2	ESC O Q		ESC Q	^] = 1Dh					
F3	ESC O R		ESC R	^^ = 1Eh					
F4	ESC O S		ESC S	^_ = 1Fh					
F5	ESC [ 15 ~								
F6	ESC [ 17 ~								
F7	ESC [ 18 ~								

F8	ESC [ 19 ~			VT200	Ggf. mit diesem Kode belegen						
F9	ESC [ 20 ~										
F10	ESC [ 21 ~										
F11	ESC [ 23 ~										
F12	ESC [ 24 ~										
	Num Mode	App Mode	VT52								
Num0	0	ESC O p	ESC ? p		VT100 ANSI	OK OK	OK - OK OK	Umschaltung in App Mode mit "ESC =", in VT52 mit "ESC [ ? 2 l" Zurück mit "ESC >" bzw. "ESC [ ? 2 h"			
Num1	1	ESC O q	ESC ? q								
Num2	2	ESC O r	ESC ? r								
Num3	3	ESC O s	ESC ? s								
Num4	4	ESC O t	ESC ? t								
Num5	5	ESC O u	ESC ? u								
Num6	6	ESC O v	ESC ? v								
Num7	7	ESC O w	ESC ? w								
Num8	8	ESC O x	ESC ? x								
Num9	9	ESC O y	ESC ? y								
Num/	/										
Num*	*	ESC O l	ESC ? l								
Num-	-	ESC O m	ESC ? m								
Num+	+ *2	ESC O M	ESC ? M								
NumEnter	CR	CR	CR								
Num,	,	ESC O n	ESC ? n								
Gängige Tasten				Freie Tasten: ^K, ^O, ^T, ^W, ^X, ^Y							
Eingabe-Abbruch											
Bild neu zeichnen											
Mikrocontroller-Reset								Vorzugsweise in Empfangs-ISR verarbeiten			
XOFF = Ausgabe anhalten								Zwingend in Empfangs-ISR verarbeiten			
XON = Ausgabe fortsetzen											



Dateiende, manchmal Entf	^D (^Z)		
Bimmel-Test	^G		

\* *HyperTerminal sendet nach jeder Sondertaste einige Null-Bytes an Müll!*

\*<sup>2</sup> *In W3 liefert diese Taste irritierenderweise CR*

### 3.8 Der Grafikzeichensatz

\*\* Zeichensatzumschaltung \*\*

ESC ( 0 VT100, nicht VT100J

ESC ( B zurück

Der Grafikzeichensatz enthält statt der Kleinbuchstaben die beliebten Rahmensymbole sowie einige wenige mathematische Symbole (VT220):

`	Diamant	p	waag. Linie halb-oben
a	Schachbrett	q	waag. Linie Mitte
b	H/T (klein)	r	waag. Linie halb-unten
c	F/F	s	waag. Linie unten
d	C/R	t	Abzweig links (nach rechts)
e	L/F	u	Abzweig rechts (nach links)
f	° (Grad)	v	Abzweig unten (nach oben)
g	± (Plusminus)	w	Abzweig oben (nach unten)
h	N/L	x	senkr. Strich
i	V/T	y	Kleiner-Gleich
j	Ecke rechts unten	z	Größer-Gleich
k	Ecke rechts oben	{	pi
l	Ecke links oben		Ungleich
m	Ecke links unten	}	£ (Pfund-Symbol)
n	Ramenkreuz	~	• (Punkt in Mitte)
o	waag. Linie oben	DEL	DEL

Font hergibt

Der "deutsche Zeichensatz" ersetzt in bekannter Manier "@[\]\{\|\}~" durch "§ÄÖÜäöüß".

nur was der Terminal-

Siehe auch [www.vt100.net](http://www.vt100.net).

## 4 Die Verwendung & das XON/XOFF-Protokoll

Die folgenden Darlegungen gehen davon aus, dass alle serielle Kommunikation komplett interruptgesteuert über Puffer abläuft. Ansonsten würde der

Mikrocontroller wechselweise bei printf() arbeiten und dann einzelzeichenweise an der Schnittstelle warten. Daher sollte die Puffergröße dem längsten printf()-Ergebnis entsprechen.

```
bool esc;
bool control;
char inchar;
int ansi_arg;
// eine Zeichen-Eingabe-Routine
bool peekchar(void) {
    char c;
    while (getc(&c)) {
        if (esc) {
            switch (c&0x7F){
                case '[':           // bei VT52 kommt kein [
                case 'O':           // im App Mode
                case '?': continue; // VT52 NumPad
            }
            if ('0'<=c && c<='9') {
                ansi_arg=ansi_arg*10+c-'0';
                continue;
            }
            if (c==';') {
                ansi_arg=0;
                continue;
            }
            if (c>' ') {
                esc=0; control=1;
                inchar=c;
                return TRUE;
            }
            continue;           // Doppel-Escapes etwa
        }
        switch (c) {
            case 27: esc=1; ansi_arg=0; continue;
        }
        control=0;
        inchar=c;
        return TRUE;
    }
    return FALSE;           // kein weiteres Zeichen in Warteschlange
}
```

```
}
```

## 4.1 Die Initialisierung

Der ANSI-Modus ist wegen seiner Farbfähigkeit im (wohl am meisten verwendeten) HyperTerminal am geeignetsten. Deshalb ist die optimale Sequenz:

```
printf( "\033[37;40m" "\033[2J" "\033[H" "\033[6n" );
```

Sie schaltet auf ANSI um, löscht den Bildschirm schwarz (für bessere Brillanz der Farben zu empfehlen) und fragt die Cursorposition ab. Die Darstellung in Einzelzeichenketten pro Escape-Sequenz verbessert die Lesbarkeit. Der C-Compiler setzt sie beim Kompilieren zusammen. Für das Escape-Zeichen wird die Oktal-Kodierung "\033" verwendet; das Metazeichen "\e" ist nicht portabel.

Denkbar ist auch folgende Schreibweise:

```
#define ESC "\033"          /* das Escape-Zeichen */
#define CSI ESC "["         /* die sog. CSI-Sequenz */
printf(CSI"37;40m" CSI"2J" CSI"H" CSI"6n");
```

Danach muss folgender String zurück kommen:

```
"\033[?1;1R"
```

Nun hat HyperTerminal einen Fehler und sendet grundlos weitere sinnlose Bytes (einige Null-Bytes, Steuerzeichen und Zeichen > 80h) hintendrein. (Die anderen getesteten Programme tun das nicht.) Deshalb muss man etwas warten und dann den Eingabepuffer leeren, sonst bleibt unweigerlich Müll im Eingabepuffer.

Nun kann das Terminal-Programm bereits im VT100-Modus sein. Das prüft man durch folgende Sequenz:

```
printf( "\033[c" );
```

Das führt zu "\033[?1;2c", wobei die zweite Ziffer anders sein kann. Bei HyperTerminal kommt danach wieder ein Rattenschwanz aus Müll. Im ANSI-Modus bleibt die Antwort aus.

Üblicherweise wird man einige Werte auf feste Positionen (bspw. oben) ausgeben und einen anderen Bereich des Bildschirms (bspw. unten, stets in voller Breite) rollbar für die Ausgabe von Fehlermeldungen u. ä. verwenden. Dazu nützt die Festlegung des rollbaren Bereichs; der Cursor muss »zu Fuß« hineingesetzt werden:

```
printf( "\033[%d;%dr" "\033[%d;1H", start, ende, start );
```

Natürlich können die "%d" durch Konstanten ersetzt werden.

## 4.2 Das laufende Programm

Da das Abfragen der Cursorposition lästig ist, sollte sie im Programm mitgeführt werden.


Per Schalter sollte eine Zeichensatz-Auswahl erfolgen, **DOS** (OEM CP437) oder **Windows/Linux** (»ANSI«-CP1252 bzw. ISO-Latin1). Je nachdem, womit das Programm erstellt wird, braucht man *eine der beiden* folgenden Umkodierungs-Tabellen für Zeichenkodes >=80h:

```
unsigned char oem2ansi[0x80]={
0xC7,0xFC,0xE9,0xE2,0xE4,0xE0,0xE5,0xE7,0xEA,0xEB,0xE8,0xEF,0xEE,0x3C,0xC4,0xC5,
0xC9,0xE6,0xC6,0xF4,0xF6,0xF2,0xFB,0xF9,0xFF,0xD6,0xDC,0xA2,0xA3,0xA5,0x80,0x81,
0xE1,0xED,0xF3,0xFA,0xF1,0xD1,0xAA,0xBA,0xBF,0x82,0xAC,0xBD,0xBC,0xA1,0xAB,0xBB,
0x83,0x84,0x85,0xA6,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,
0x92,0x93,0x94,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F,0xA8,0xA9,
0xAD,0xAE,0xAF,0xB3,0xB4,0xB6,0xB8,0xB9,0xBE,0xC0,0xC1,0xC2,0xC3,0xC8,0xCA,0xCB,
0xCC,0xDF,0xCD,0xCE,0xCF,0xD0,0xB5,0xD2,0xA4,0xD3,0xD4,0xA7,0xD5,0xD7,0xD8,0xD9,
0xDA,0xB1,0xDB,0xDD,0xDE,0xE3,0xF7,0xF0,0xB0,0xF5,0xB7,0xF8,0xFD,0xB2,0xFE,0xA0};
```

```
unsigned char ansi2oem[0x80]={
0x9E,0x9F,0xA9,0xB0,0xB1,0xB2,0xB4,0xB5,0xB6,0xB7,0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,
0xBE,0xBF,0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,
0xFF,0xAD,0x9B,0x9C,0xE8,0x9D,0xB3,0xEB,0xCE,0xCF,0xA6,0xAE,0xAA,0xD0,0xD1,0xD2,
0xF8,0xF1,0xFD,0xD3,0xD4,0xE6,0xD5,0xFA,0xD6,0xD7,0xA7,0xAF,0xAC,0xAB,0xD8,0xA8,
0xD9,0xDA,0xDB,0xDC,0x8E,0x8F,0x92,0x80,0xDD,0x90,0xDE,0xDF,0xE0,0xE2,0xE3,0xE4,
0xE5,0xA5,0xE7,0xE9,0xEA,0xEC,0x99,0xED,0xEE,0xEF,0xF0,0xF2,0x9A,0xF3,0xF4,0xE1,
0x85,0xA0,0x83,0xF5,0x84,0x86,0x91,0x87,0x8A,0x82,0x88,0x89,0x8D,0xA1,0x8C,0x8B,
0xF7,0xA4,0x95,0xA2,0x93,0xF9,0x94,0xF6,0xFB,0x97,0xA3,0x96,0x81,0xFC,0xFE,0x98};
```

Die übliche Methode, nur die Umlaute umzusetzen, ist für Mikrocontroller kein Armutszeugnis, sondern bisweilen aus Platzmangel nicht anders möglich.

Der Verzicht auf Umlaute erspart Aerger, aber es sieht haesslich aus!

Im Hinblick auf die Zukunft ist als weitere Kodierung  [UTF-8](#) dringend angeraten!! Für den Mikrocontroller genügt es aber, intern mit OEM oder ANSI oder [Hitachi-Display-Kodes](#) zu arbeiten. Ungültige UTF-8-Kodes sollte der Mikrocontroller schlauerweise (und entgegen der RFC2279) als OEM oder ANSI verarbeiten, um fehlertolerant zu bleiben. Bestimmt wird auch in zehn Jahren nicht jeder Anwender wissen, was UTF-8 ist. Obwohl es wirklich wichtig ist.

Ein besonders hübsches Programm kann sich an **wechselnde Fenstergrößen** anpassen. Dazu positioniert es den Cursor regelmäßig nach »janz weit draußen« (=jwd), bspw. mit:

```
printf( "\033[99;99H" "\033[6n" );
```

Das Terminal wird (hoffentlich!) den Cursor in die rechte untere Ecke stellen, und die nachfolgende Abfrage liefert dann mit:

```
if ( scanf( "\033[?%d;%dR" ,&y,&x)==2 ) ...;
```

in  $x$  und  $y$  die Koordinaten. Sicherheitshalber stellt man den Cursor vorher in die Standard-Ecke, oder ignoriert Rückgabewerte kleiner als 24 bzw. 80.

Ein Stückchen Kode, den man sonst ständig neu erfinden müsste, ist die Eingabezeile mit Editierfunktionen. Das Verhalten wurde weitestgehend an Windows-Editfelder angelehnt, insbesondere das Markieren der Vorgabe. Die globale Variable *inputstring* erspart für Mikrocontroller lästige Schaufelarbeiten.

```
static void deleteright(void) {printf(CSI"K");}          // löschen rechts
static void gotoxy(int x, int y) {printf(CSI"%d;%dH",y,x);} //wie Pascal

static char inputstring[80];    // einzige statische Variable
static void inputclear(int x) {
    gotoxy(x,21);
    deleteright();
    inputstring[0]=0;
}

// Eingabe -- während der Eingabe steht der Rest des Bildschirms still
// Angezeigt und zurückgegeben wird -- hier feste Zeile 21
static bool input(const char *prompt) {
    int edleft=strlen(prompt)+3;
    int inlen=strlen(inputstring);
    int x=0, nx=inlen;        // Momentaner und neuer Cursor
    SaveCurs();
    printf(XY(1,21) CSI"30;43m" CSI"K" "%s: ",prompt);
    high=1;                  // Zunächst alles markieren (wie Windows)
rep:
    StartHi();
    printf("%s",inputstring+x);
    EndHi(); if (high) printf(CSI"30;43m");           // Farbkode wiederherstellen
```

```

    if (nx<=x) putc(' ');          // 1 Zeichen dahinter löschen (für BS und DEL)
repl:
    x=nx;
    gotoxy(x+edleft,21);          // Cursor positionieren
    for(;;) {
        if (!peekchar()) _idle(); else switch (UART_Inchar) {
            case 'C'+256: nx++; goto move;          //RIGHT
            case 'D'+256: nx--; goto move;          //LEFT
            case 'H'+256: pos1: nx=0; goto move;     //HOME
            case 'K'+256: ende: nx=inlen; move:{ //END
                nx=limit(nx,0,inlen);
                if (_testclear_(high)) goto rep;     // komplett zeichnen
            }goto repl;
            case '~'+256: switch(argv[0]){           // VT200-Tasten
                case 1: goto bs;
                case 3: goto del;
            }goto b;
            case 27+256:                          // 2x Escape (sehr intuitiv)
            case 3:                               // ^C, Programm-Ende = Abbruch
            case 4:                               // ^D, Eingabe-Ende Unix = Abbruch
            case 0x1A:                             // ^Z, Eingabe-Ende DOS = Abbruch
            case 13: goto ret;                      // ENTER, übernehmen
            case 8: bs: if (!x) goto b;             // Löschen nach links, ignorieren am Anfang
                nx=-x; putc(8); nobreak;           // 1 Zeichen nach links rücken fürs Löschen
            case 0x7F: del:{                        // Löschen nach rechts, ignorieren am Ende
                if (_testclear_(high)) {
                    x=nx=inlen=0; inputclear(edleft); // Cursor positionieren, rechts löschen
                }
                if (x==inlen) goto b;
                memmove(inputstring+x,inputstring+x+1,inlen-x);
                inlen--;
            }goto rep;                             // rechts neuzeichnen
            case 1: goto pos1;                      // ^A, Zeilenanfang
            case 5: goto ende;                      // ^E, Zeilenende
            default: if (UART_Inchar>=' ') {
                if (_testclear_(high)) {
                    x=nx=inlen=0; inputclear(edleft); // Cursor positionieren, rechts löschen
                }
                if (inlen+edleft<79) inlen++;        // neue Länge
                memmove(inputstring+x+1,inputstring+x,inlen-x);
            }
        }
    }

```

```

        // Platz machen ODER letztes Zeichen herausschieben
        inputstring[x]=UART_Inchar; // Zeichen einfügen
        nx++;                        // Cursor nach rechts
        goto rep;                    // rechts neuzeichnen
    }else b: putc(7); break;        // ^G, Bimmel-Test
    }
}
ret:
    high=0;
    printf(CSI"0m" CSI"2K");        // Zeile mit Normalattributen löschen
    RestoreCurs();
    return UART_Inchar==13;
}

```

Keine Panik bei **goto**! Ohne wäre der Quelltext und der Maschinenkode dreimal so lang... Pardon, *high* ist noch eine globale Bitvariable, die in den Funktionen `StartHi()` und `EndHi()` wirksam wird.

Die Funktion `peekchar()` verwaltet alle ANSI-Buchstaben und erzeugt ggf. Zeichenkodes  $\geq 100h$  für eine einfache Fallunterscheidung. (Bei 8-bit-Prozessoren sollte man solche "Optimierungen" tunlichst unterlassen!)

## 4.3 Die lästigen Unterschiede

Der Leser mag bitte diesen unfertigen Abschnitt überspringen...

### HyperTerminal ###

Das dumme HyperTerminal kann eigentlich nur im ANSI-Modus betrieben werden und unterstützt nur folgende Sondertasten:

Pfeiltasten, Pos1, Ende, F1..F4; insbesondere KEIN PgUp, PgDn!

Im VT100-Modus entfallen zudem Pos1 und Ende, Grau+ ergibt Komma.

Der VT52-Modus kann genauso viel, mit kürzeren Codes.

Keine Funktion der Entf-Taste. Backspace liefert standardmäßig 08h

ANSIW brachte keine sichtbaren Veränderungen (von wegen Unicode).

Kermit: 01 2D 20 53 7E 25 20 40 2D 23 59 31 7E 28 43 0D

01 "- S~% @-#Y1~(C" 0D

### Windows 3.1 Terminal ###

Im VT100-Modus nur Pfeiltasten und F1..F4; muss man aber erst aktivieren

(="Funktions-, Richtungs- und Strg-Tasten für Windows" AUS), kein Pos1/Ende!

Erstaunlicherweise funktioniert die Entf-Taste und liefert 7Fh,

Backspace liefert 08h, 20h, 08h

```
XMODEM: 18 18 18 18 18 08 08 08 ...
```

```
### NC Term90 ###
```

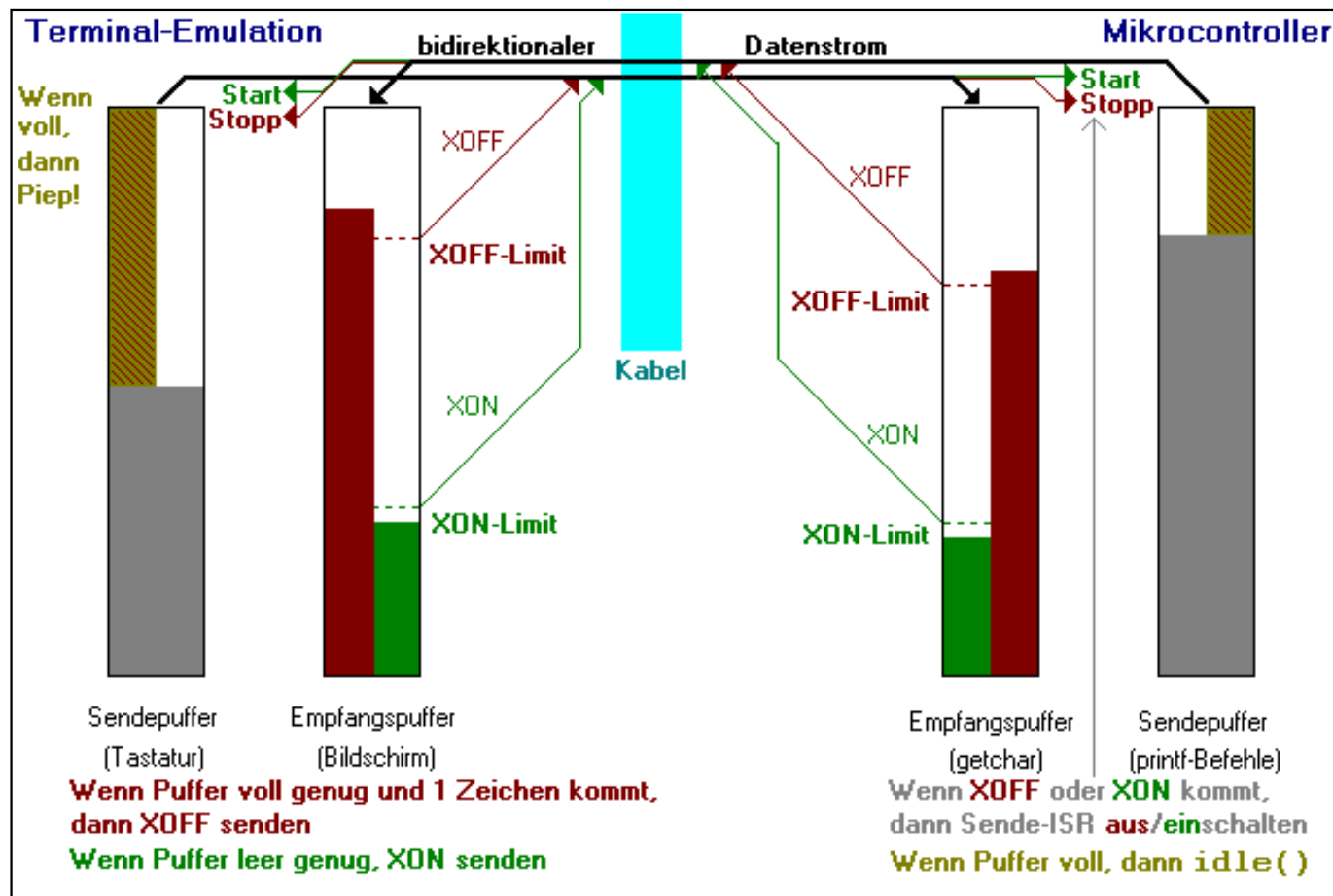
Auch hier gibt es keine Unterstützung für PgUp/PgDn; das ist (sinnvollerweise) mit Upload/Download belegt! Entf liefert 7Fh, Backspace 08h

```
Kermit: 01 29 20 53 7E 2A 20 40 2D 23 57 0D
```

```
01 "9 S~* @-#W" 0D
```

## 4.4 Das XON/XOFF-Protokoll

Für den Fall, dass der Mikrocontroller das Terminal mit Daten überschüttet, ist zur Synchronisierung das XON/XOFF-Protokoll das geeignetste, weil es keine weitere Leitung benötigt.



Ist der Datenpuffer des Terminals fast voll, sendet es (statt eines Tastendrucks) den Code XOFF (=13h, 19, ^S). Daraufhin sollte der Mikrocontroller



den Datenausstoß anhalten; am einfachsten, wenn bei Empfang von XOFF der Interrupt "Sendehalteregister frei" abgeschaltet wird. Etwas »**Kopffreiheit**« (= Abstand zwischen *XOFF-Limit* und *Puffer voll*) von mindestens zwei Zeichen benötigt der Empfangspuffer, weil das Aussenden von XOFF und das Verarbeiten in der Gegenstelle etwas Zeit erfordert.

Bei genügend leerem Datenpuffer sendet das Terminal den Kode XON (=11h, 17, ^Q). Das Mikrocontrollerprogramm sollte nun den vorher gesperrten Sendehalteregister-Interrupt freigeben.

Das *XON-Limit* darf mit *Puffer leer* zusammenfallen; es schadet nur dem Durchsatz.

**Wichtiges implementatorisches Detail:** Das Aussenden von XOFF und XON darf keinesfalls über den Sendepuffer laufen, sondern geht priorisiert an diesem vorbei, es sind sogenannte OOB-(out-of-band-)Daten. Diese Zeichen werden auch dann gesendet, wenn die Gegenstelle selbst per XOFF-Befehl (in Gegenrichtung) die Datenannahme verweigert. Auch beim Zeichenempfang gehen diese Zeichen niemals in den Empfangspuffer, sondern werden sofort in der ISR verarbeitet.

Von der Funktion des XON/XOFF-Protokolls im Mikrocontroller kann man sich ganz einfach durch Betätigen von **Strg+S** im Terminalprogramm überzeugen: Die Ausgaben müssen anhalten, und - richtig programmiert - verbraucht der im Schlafmodus "hängende" Controller auch weniger Strom als sonst. Aufwecken (hier im wahrsten Sinne des Wortes) erfolgt dann mit **Strg+Q**.

Für die umgekehrte Richtung ist das XON/XOFF-Protokoll zwar auch vorgesehen, aber der Eingabestrom von einer Tastatur sollte nicht so riesig werden. (Ausnahme: Eingabeumleitung! Dazu muss die Sende-ISR so gestaltet werden, dass ein Zeichen an der Warteschlange vorbei bevorzugt gesendet werden kann.)

Zu beachten ist, dass XON/XOFF für die Dauer einer (im folgenden Abschnitt beschriebenen) Binärdatenübertragung abzuschalten ist!

Für einen AT90S4433 oder ATmega kann es aus Aufwandsgründen bei "halbem" XON/XOFF-Protokoll bleiben, wie [in diesem funktionierenden Assembler Quelltext](#) dargestellt.

Der Quelltext für "volles" XON/XOFF auf dem 80C167:

```
#define BAUDRATE 19200 // Datenbits fest 8, Stopbits fest 1
#define UART_OBUFLN 64 // Puffer für interruptgesteuerte Zeichenausgabe
#define UART_IBUFLN 32 // dito für Zeichen-Eingabe
#define XON_THRESHOLD (UART_IBUFLN/2) // Schwelle für Senden von XON
#define XOFF_THRESHOLD (UART_IBUFLN-8) // Schwelle für Senden von XOFF
#define XON 0x11 // ^Q
#define XOFF 0x13 // ^S
sbit P3_10= P3^10; // Port-Pin für TxD (muss HIGH sein)
sbit DP3_10=DP3^10; // Port-Richtung für TxD (muss 1 = Output sein)
sbit DP3_11=DP3^11; // Port-Richtung für RxD (muss 0 = Input sein)
```

```

bool UART_Tx-Allow;      // true wenn XON und false wenn XOFF empfangen wurde
bool UART_XonXoff;       // XON/XOFF-Flusssteuerung (AUS während XMODEM)
bool UART_Xoff_sent;     // XON nur senden, wenn auch XOFF gesendet wurde
bool UART_Xon_sent;      // XON-Stopp, wenn XON gesendet + Zeichen eintreffen
char UART_Bypass;        // XON/XOFF-Zeichen, hochpriorisiert einzufügen
word UART_Inchar;        // low=letztes Zeichen von peekc() (optimiert 16bit)

static char obuf[UART_OBUFLLEN];
static int obufwr,obufrd; // zunächst Null
static char ibuf[UART_IBUFLLEN];
static int ibufwr,ibufrd; // zunächst Null

void UART_Init(void) {
    P3_10=1;                // Port 3.10 auf High (TxD)
    DP3_10=1;               // Port 3.10 als Ausgang (TxD)
                           // Port 3.11 bleibt Eingang (RxD)
    S0BG =CPUCLK/32/BAUDRATE-1;
    S0CON=0x8011;           // set serial mode
//      ^-- 1 Stopbit (0), 8bit Daten (001)
//      ^--- Überlauf, Rahmenfehler, Parity AUS (000), Empfänger EIN (1)
//      ^---- (keine Fehler-Flags setzen)
//      ^----- Baudratengenerator EIN (1), Loopback, Baudrate/3, Odd AUS (000)
    S0TBIC=MKIC(1,0,2,0);   // niedrige Priorität, Interrupts noch gesperrt
    S0RIC =MKIC(0,1,2,1);   // ebenso niedrige Priorität
    UART_Tx-Allow=1;
    UART_XonXoff=1;
}

void UART_Transmit(void) interrupt S0TBINT=0x47 {
    ISR_TIC();
    if (UART_Bypass) {      // Notfall-Zeichen (XON oder XOFF)
        S0TBUF=UART_Bypass;
        UART_Bypass=0;
    }else if (obufrd!=obufwr) {
        S0TBUF=obuf[obufrd];
//    _bflld_(S0TBUF,255,obuf[obufrd]); // besser, aber crasht den Compiler!!
        RING_INC(obufrd,UART_OBUFLLEN);
    }else{
        S0TBIE=0;          // keine Interrupts = Puffer leer
        S0TBIR=1;          // aber bei EI sofort ISR aufrufen
    }
}

```

```

}
ISR_TOC();
}

void UART_Receive(void) interrupt S0RINT=0x2B {
    char c;
    int fill;
    ISR_TIC();
    c=S0RBUF;
    if (UART_XonXoff) switch (c) {          // XON und XOFF nicht in Empfangspuffer
        case XON: S0TBIE=UART_Tx-Allow=1; goto toc;
        case XOFF: S0TBIE=UART_Tx-Allow=0; goto toc;
    }
    fill=ibufwr-ibufrd; if (fill<0) fill+=UART_IBUFLEN;
    if (UART_XonXoff && fill>=XOFF_THRESHOLD) {
        UART_Bypass=XOFF;          // Notfalls bei jedem Zeichen einfordern
        UART_Xoff_sent=1;
        S0TBIE=1;
    }
    if (_testclear_(UART_Xon_sent)) UART_Xoff_sent=0;
        // wenn ein Zeichen kommt, XON nicht mehr wiederholen lassen
    if (fill==UART_IBUFLEN-1) goto toc;    // Puffer voll, Zeichen ignorieren
    ibuf[ibufwr]=c;
    RING_INC(ibufwr, UART_IBUFLEN);
toc:
    ISR_TOC();
}

char putc(char c) {          // binäre Zeichenausgabe auf 1. serielle Schnittstelle
    if (IEN) {              // interruptgetrieben wenn globale Interrupts ein
        int o_next=obufwr;
        obuf[o_next]=c;      // diese Stelle ist stets frei
        RING_INC(o_next, UART_OBUFLEN);
        for (;;) _idle_() if (o_next!=obufrd) break;
        obufwr=o_next;
        S0TBIE=UART_Tx-Allow; // ISR anschubsen, wenn nicht per XOFF blockiert
    } else {                // klassisch (mit Handbremse, OHNE Xon/Xoff)
        while (!_testclear_(S0TBIR));
        S0TBUF=c;
    }
}

```

```

    return c;
}

bool peekc(void) {           // Zeichen im Empfangspuffer?
    int fill=ibufwr-ibufrd;
    if (fill<0) fill+=UART_IBUFLEN;
    if (UART_Xoff_sent && fill<=XON_THRESHOLD) {
        UART_Bypass=XON;    // Notfalls bei jedem peekc() einfordern
        UART_Xon_sent=1;
        S0TBIE=1;
    }
    if (!fill) return false; // wenn Puffer leer
    UART_Inchar=(byte)ibuf[ibufrd];
    RING_INC(ibufrd, UART_IBUFLEN);
    return true;
}

```

Das Arbeiten mit einem globalen Zeichenpuffer (UART\_Inchar) erweist sich als wesentlich zweckmäßiger, als das Zeichen ständig als Funktions-Rückgabewert herumzuschleppen. Der Rückgabotyp **bool** (dasselbe wie **bit**) ist beim C166er Compiler recht günstig implementiert.

## 5 Binärdatenübertragung mit XMODEM

Bisweilen benötigt man die Möglichkeit, Binärdaten aus dem Mikrocontroller zu lesen bzw. hinein zu schreiben.

Beispielsweise die nächste Programmversion. Dazu musste man immer zu einem Brennprogramm wechseln. Darauf kann man künftig verzichten, wenn man nur einen Urlader im Mikrocontroller behält, der sogar eine *Entschlüsselung* der (geheimen) Brenndaten vornehmen kann.

Umständlich ist auch, dass das Brennprogramm oftmals die gleiche Schnittstelle braucht, sodass man das Terminal-Emulationsprogramm derweil beenden muss.

Auch das Auslesen des RAM-Bereichs als eine Art "Arme-Leute-Debugger" oder bei der Fehlersuche im Feldeinsatz (Stichwort: Bananensoftware, reift beim Anwender) ist sicher irgendwann nützlich.

Jede Terminal-Emulation bietet mehrere Binärdatenübertragungen an. Am einfachsten zu implementieren ist das XMODEM-Protokoll:

1. Der Empfänger beginnt und sendet ein ^U = 15h
2. Der Sender schickt einen Block aus 132 Bytes mit:
  - 1 Byte ^A = 01h (SOH)
  - 1 Byte Blocknummer (mit 01h beginnend und von FFh nach 00h umschaltend)
  - 1 Byte Einerkomplement der Blocknummer

- 128 Byte Nutzdaten (am Ende der Datei mit ^Z = 1Ah aufgefüllt)
  - 1 Byte Prüfsumme = Summe über die Nutzdaten
3. Der Empfänger bestätigt den korrekten Empfang mit ^F = 06h, weiter bei 2. oder 5.
  4. Der Empfänger bestätigt den falschen Empfang mit ^U = 15h, weiter bei 2.
  5. Der Sender sendet ein Ende-Byte ^D = 04h (EOT), wenn fertig
  6. Der Empfänger quittiert durch Senden von ^F = 06h

Von Nachteil ist die feste Blockgröße von 128 Bytes (jaja, die CP/M-Zeiten).

Es gibt auch eine CRC-Variante mit besserer Fehlerabsicherung; diese überträgt 133-Byte-Blöcke mit einer 2-Byte-CRC. Das CRC-fähige Empfangsprogramm signalisiert sein Extra durch Senden von 'C' = 43h an Stelle von ^U = 15h, nach einem TimeOut sendet es dann ^U = 15h und fällt in den einfachen Prüfsummen-Modus zurück.

Die Norton Commander TERM9x.EXE unterstützen keine CRC. Auf der anderen Seite hat HyperTerminal den Fehler, beim Empfang über eine Minute lang CRC zu erwarten und dann auf Prüfsumme zurückzuschalten; bis dahin hat der Mikrocontroller (laut Vorschrift) längst aufgegeben.

Auch gibt es vergrößerte Blöcke; diese haben als erstes Byte ein ^B = 02h und sind 1 KByte lang. In Verbindung mit Mikrocontrollern genügen die kurzen Blöcke.

Während einer Datenübertragung ist keine Bildschirmausgabe möglich!

Einmal in den Sende- oder Empfangsmodus gebracht kann der Mikrocontroller nur durch Druck aufs richtige Knöpfchen vorzeitig erlöst werden:

- Strg+X (CAN = 18h, 24, ^X) beim Datei-Empfang
- Strg+D (EOT = 4, ^D) beim Datei-Versand

Hier folgt eine (getestete!) Implementierung für den SAB80C167-Controller:

```
enum {SOH=1,EOT=4,ACK=6,NAK=0x15,CAN=0x18};

static bool crcmode;
static word crc;
static void new_crc(word c) {
    if (crcmode) {
        int j;
        c<=8; crc^=c; j=8;do{crc<=1; if(C)crc^=0x1021;}while(--j);
    }else crc+=c;           // C (=Carry) enthält das soeben herausgeschobene Bit
}

word xmodem_send(const char huge* buf, word buflen) {
```

```

bool eot;
word t,bytes,bn,i;      // Compiler stellt sich bei CHAR blöd an!

UART_XonXoff=0;
bn=1; bytes=i=0; eot=0; crcmode=0;
rep:
t=t8h; do{
  if (peekc()) switch(UART_Inchar) {
    case ACK: {              // 6, ^F
      if (eot) goto ret;
      if (i>buflen) i=buflen;
      bytes+=i; buf+=i; bn++;
      if (buflen==i) goto sendblock;
      putc(EOT);              // innerhalb 60s das ^D bestätigt absenden
      eot=1;
    }break;
    case 'C': crcmode=1; nobreak; // Bug in HyperTerminal: geht nicht ohne CRC
    case NAK: {              // 0x15, ^U; nur Prüfsumme
sendblock:
      RED_LED=~RED_LED;      // Bei jedem ACK oder NAK: EIN oder AUS
      putc(SOH);              // 1, ^A
      putc(bn); putc(~bn);    // Blocknummer, inverse Blocknummer
      crc=0; i=0; do{
        word c=0x1A;          // Terminator (Dateiendekennung) ^Z
        if (i<buflen) c=buf[i]; // sonst "richtiges" Zeichen
        putc(c);
        new_crc(c);
      }while (++i<128);
      if (crcmode) putc(crc>>8); // High-Teil zuerst
      putc(crc);
    }goto rep;              // wieder 1 Minute Timeout
    case CAN: goto ret;      // Abbruchwunsch des Empfängers
  }else _idle();
}while (t8h-t<60*T8H_TICPERSEC); // 1 Minute Timeout
ret:
UART_XonXoff=1;
return bytes;
}

static bool getchls(void) { // Zeichen mit 1 Sekunde Timeout empfangen

```

```

word t=t8h;
do{
    if (peekc()) return true;
    _idle_();
}while (t8h-t<T8H_TICPERSEC);
return false;
}

word xmodem_rcv(char huge *buf, word buflen) {
    word bytes,bl,i,t,rcs;
    byte ak,bn,rbn,ibn;

    UART_XonXoff=0;
    bytes=0; crcmode=1; ak='C'; bn=1;
rep:
    i=10; do{
        putc(ak);                // versucht XMODEM/CRC
        t=t8h; do{
            bl=128;                // Standard-Blocklänge
            if (getc1s()) switch (UART_Inchar) {
                case 2: bl=1024; nobreak;    // akzeptiert XMODEM/1K
                case SOH: goto soh;
                case EOT: ak=ACK; goto eot;
            }
        }while (t8h-t<6*T8H_TICPERSEC);    // 6 "innere" Sekunden
        if (i==7 && _testclear_(crcmode)) ak=NAK; // nach 4 Versuchen zurückschalten
    }while (--i);                // 1 Minute lang
    goto ret;
soh:
    RED_LED=~RED_LED;            // Bei jedem Blockanfang: EIN oder AUS
    ak=NAK;
    if (!getc1s()) goto rep;      // Blocknummer
    rbn=UART_Inchar;
    if (!getc1s()) goto rep;      // Inverse Blocknummer
    ibn=UART_Inchar;              // Tests erst am Blockende!
    crc=0; i=0; do{
        byte c;                    // besserer Kode
        if (!getc1s()) goto rep;
        c=UART_Inchar;
        new_crc(c);
    }while (i++<buflen);
    if (rcs) *buf+=bufen;
    return rcs;
}

```

```

    if (i<buflen) buf[i]=c;          // gleich abspeichern
}while (++i<bl);
if (!getc1s()) goto rep;
rcs=UART_Inchar;                    // rcs = gelesene Prüfsumme
if (crcmode) {
    if (!getc1s()) goto rep;
    rcs=rcs<<8 | UART_Inchar;
}else crc&=0xFF;                    // Block empfangen, jetzt kommen die Tests:
if (rbn!=~ibn) goto rep;            // ungültige Blocknummer
if (crc!=rcs) goto rep;             // ungültige Daten
if (rbn==(byte)(bn-1)) goto ack;    // ACK wiederholen
if (!buflen                         // keine Daten mehr erwartet (nicht fatal)
|| rbn!=bn) {ak=CAN; goto eot;}    // fataler Sync-Fehler
if (i>buflen) i=buflen;            // Minimum
bytes+=i; buf+=i; ++bn;  buflen-=i;
ack:
    ak=ACK;
    goto rep;
eot:
    putc(ak);
ret:
    UART_XonXoff=1;
    return bytes;
}

```

Der Quelltext ist leidlich handoptimiert, deshalb einige merkwürdige Konstrukte und die vielen **gotos**.

Die **volatile** Variable *t8h* ist der Interruptzähler für den Zeitgeber T8 und macht bei 20 MHz Takt und Vorteiler 8 38 Schritte pro Sekunde; hervorragend geeignet für solche Zeitmessungen.

## 6 Alles zusammen im Mikrocontroller

Wenn wir schon mal bei dem SAB80C167 sind, dann könnte man die Daten auch gleich wegbrennen (AMD-Flash auf Phyttec MiniModul-167):

```

static void preamble(word huge *adr, word third) {
// Interrupts sperren und AMD-Präambel-Bytes ausgeben
disable();
*(word*)&adr=0xAAAA;    // Low-Teil setzen
*adr=0xAAAA;            // beide Flash-Chips via EXTS ansprechen

```



```

*(word*)&adr=0x5554;
*adr=0x5555;
if (third) {
    *(word*)&adr=0xAAAA;
    *adr=third;
}
}

static bool flashreset(word huge *adr) {
// AMD-Flash-Reset ausgeben und Interrupts freigeben
*adr=0xF0F0;           // Flash-RESET auslösen (= Array-Lese-Modus)
enable();
return false;
}

static bool polling(word huge* addr, word data) {
word r,r2;
for(;;) {               // DATA# Polling
    r=*addr;
    if (r==data) {
        enable();
        return true;
    }
    r2=*addr;
    if (r&0x0020 && (r2^data)&0x00FF) break;    // Low-Teil defekt
    if (r&0x2000 && (r2^data)&0xFF00) break;    //High-Teil defekt
}
return flashreset();    // Flash-RESET auslösen
}

bool far ProgWord(word huge* addr, word data) { // addr muss gerade sein!
preamble(0xA0A0);      // Kommando: Byte schreiben
*addr=data;
return polling(addr,data);
}

bool far EraseSector(word huge* addr) { // addr sollte durch 32K teilbar sein!
preamble(0x8080);      // Kommando: Sektor löschen
preamble(0);
*addr=0x3030;

```

```

    return polling(addr,0xFFFF);
}

bool far IsAMD256K(void) {
    word MID,DID;                // Hersteller- und Chip-ID
    preamble(0x9090);            // Kommando: Autoselect-Modus
    MID=*(word huge*)0;
    DID=*(word huge*)2;
    flashreset();
    if (MID==0x0101 && DID==0x2020) return true;
    return false;
}

bool ProgSector(const word huge* src, word huge* dst, word len) {
    // 1 Sektor löschen und programmieren; dst sollte am Sektoranfang liegen
    // (durch 32K teilbar sein), len darf bis zu 16K (=32KByte) groß sein.
    // Prozeduren in RAM kopieren und diese anspringen!
    // Da alle CALLs und JMPs absolut sind(?), verbietet sich dabei eine
    // Adressverschiebung, also müssen o.g. Prozeduren in ein extra Segment,
    // bspw. ab 400h, und im RAM braucht man eine ebenso große Lücke.
    word huge *cp=preamble;
    word plen=(ProgSector-preamble+1)/2;
    do { *(cp|RAM_BASE)=*cp; cp++; } while (--plen);
    if (!(IsAMD256K|RAM_BASE)()) return false;
    if (!(EraseSector|RAM_BASE)(dst)) return false;
    if (len) do{
        if (!(ProgWord|RAM_BASE)(dst,*src)) return false;
        dst++;
        src++;
    }while (--len);
    return true;
}

```

---

EMail: [Henrik Haftmann](mailto:Henrik.Haftmann@tu-chemnitz.de)

Soforthilfe: [talk\\_henni@wombat.infotech.tu-chemnitz.de](mailto:talk_henni@wombat.infotech.tu-chemnitz.de)

Chemnitz, letzte Änderung: 11.10.2004