Anschluß an den Rest der Welt

Serielle Schnittstelle für Atari XL/XE in der Assemblerecke

Der kleine Atari ist nicht mit sehr vielen (normgerechten) Schnittstellen ausgestattet. So kommt es, daß er ein einsames Dasein ohne jede Verbindung zu anderen Computern fristen muß. Das gebräuchlichste Interface mit dessen Hilfe man Kontakt zu fast allen Computern, Akkustikkopplern und fremden Datenverarbeitungsanlagen aufnehmen kann, ist die RS-232-Schnittstelle. Diese muß schnell große Datenmengen übertragen können, so daß sie über den Expansions-Port anzuschließen ist. In dieser Assemblerecke liefern wir die Hard-

8 Bit

ware-Schaltung und erklären die notwendige Software zur Einbindung in das Atari-Betriebssystem. Außerdem zeigen wir anhand eines kleinen Basic-Programms, wie man grundsätzlich ein Terminal-Programm zur Datenübertragung erstellt.

Aufbau der RS-232-Schaltung

Die Schaltung kann problemlos auf einer Lochrasterplatine aufgebaut werden. Etwas Sorgfalt verlangt aber der Anschluß der 50poligen Federleiste für die Verbindung mit dem Atari XL. Am besten verwendet man eine Federleiste mit gewinkelten Anschlüssen und lötet sie direkt auf der Platine an. Da am Expansions-Port der XL-Computer leider keine Betriebsspannung von +5 V zur Verfügung steht, muß man sich mit einem Extrastecker an einem der beiden Joystickports behelfen.

Für Besitzer eines Atari XE haben die Entwickler an die Betriebsspannung gedacht. Der Preis dafür ist aber ein völlig geänderter Expansions-Port. Es gibt nun das ECI, das erst in Verbindung mit dem Modul-Port einen vollständigen Expansions-Port bildet. Außerdem wird jetzt keine Federleiste mehr gebraucht, sondern zwei Platinenrandstecker. Auf dem abgedruckten Schaltplan sind die Pin-Belegungen für die Ports beider Computer verzeichnet. Standardmäßig wird für die RS-232-Schnittstelle eine 25polige SUB-D-Buchse verwendet. Da das Raster der Anschlußbeinchen nicht mit dem einer Lochrasterplatine übereinstimmt, müssen Sie die Beinchen beim Einlöten entsprechend verbiegen oder die Buchse mit einem Kabel anschließen.

Die Bauteile der RS-232-Schnittstelle

Das Herzstück der Schaltung ist der ACIA (Asynchronouzs Communitations Interface Adapter) 6551. Seine Steuerleitungen sind kompatibel zur 6502-CPU des Atari. Der ACIA übernimmt die gesamte serielle Datenübertragung und kümmert sich um alle Formalitäten wie Baud-Rate, Parität und Start-/Stopbits. Er liefert an seinen Ausgängen aber nur standardmäßige TTL-Pegel (0 bzw. 5 V), während der RS-232-Standard höhere Spannungsdifferenzen (ca. 12 V bzw. ca. −12 V) verlangt. Deshalb sind zwei Treiber-ICs ICL232 (baugleich mit MAX232) nötig. Jedes von ihnen enthält zwei Eingangstreiber zur Umwandlung des RS-232-Pegels in den TTL-Pegel und zwei Ausgangstreiber für den umgekehrten Vorgang.

Adresse	Name	Funktion beim Lesen	Funktion beim Schreiben
$D100	DATA-Register	enthält empfangenes Daten-Byte	übergibt zu sendendes Daten-Byte an ACIA
$D101	STATUS-Register	enthält Status-Byte	löst Programm-Reset aus
$D102	COMMAND-Register	enthält Kommando-Byte	übergibt Kommando-Byte an ACIA
$D103	CONTROL-Register	enthält Kontroll-Byte	übergibt Kontroll-Byte an ACIA

Außerdem benötigt man noch zwei TTL-ICs zur Speicherdekodierung. In unserer Schaltung werden dem ACIA die Speicherzellen $D100 bis $D103 zugewiesen. Die Speicherdekodierung ist in unserer Schaltung, um Bauteile zu sparen, nicht vollständig, d.h., die vier ACIA-Register wiederholen sich 256 Bytes lang periodisch (bis $D1FF).

Über das COMMAND- und das CONTROL-Register kann man dem ACIA alle für die Datenübertragung wichtigen Parameter mitteilen. Auch die Interruptsteuerung des ACIA läßt sich über diese Register beeinflussen. Das STATUS-Register kann nur gelesen werden und enthält alle wichtigen Informa-

CONTROL-Register

Bit 7	Anzahl der Stopbits
0	1 Stopbit
1	2 Stopbits
1	1.5 Stopbits, wenn Wortlänge = 5 und keine Parität
1	1 Stopbit, wenn Wortlänge = 8 und Parität

Bit 6 5	Wortlänge
0 0	8 Bits
0 1	7 Bits
1 0	6 Bits
1 1	5 Bits

Bit 4	Taktgeber für Empfänger
0	externer Takt
1	eingestellte interne Baudrate

Bit 3 2 1 0	Baud-Rate
0 0 0 0	16mal externer Takt
0 0 0 1	50
0 0 1 0	75
0 0 1 1	109.92
0 1 0 0	134.58
0 1 0 1	150
0 1 1 0	300
0 1 1 1	600
1 0 0 0	1200
1 0 0 1	1800
1 0 1 0	2400
1 0 1 1	3600
1 1 0 0	4800
1 1 0 1	7200
1 1 1 0	9600
1 1 1 1	19200

COMMAND-Register

Bit 7 6	Paritätsmodus
0 0	ungerade Parität beim Senden und Empfangen
0 1	gerade Parität beim Senden und Empfangen
1 0	High-Parität nur beim Senden, Paritätsüberprüfung aus
1 1	Low-Parität nur beim Senden, Paritätsüberprüfung aus

Bit 5	Parität
0	keine Parität
1	Paritätsmodus einschalten

Bit 4	Empfänger Echomodus
0	normaler Modus
1	Echomodus, Bit 2 und 3 müssen 0 sein. RTS geht auf low.

Bit 3 2	Sender-Interrupt-Kontrolle
0 0	RTS-high, Sender-Interrupt aus
0 1	RTS-low, Sender-Interrupt ein
1 0	RTS-low, Sender-Interrupt aus
1 1	RTS-low, Sender-Interrupt aus Break senden über TxD

Bit 1	Interrupt-Kontrolle
0	IRQ ermöglicht
1	IRQ aus

Bit 0	Daten-Terminal (ACIA) bereit
0	nicht bereicht (DTR high)
1	bereit (DTR low)

STATUS-Register (mit * gekennzeichnete Ereignisse sind Gründe für das Auslösen eines IRQ)

Bit 7	Interrupt-Steuerung (IRQ-Ausgang) wird nach Lesen des STATUS-Registers gelöscht
0	kein IRQ
1	IRQ aufgetreten*

Bit 6	Datengegenstelle (DSR-Eingang)
0	bereit – DRS low*
1	nicht bereit – DSR high*

Bit 5	Trägersignal (DCD-Eingang)
0	liegt vor – DCD low*
1	liegt nicht vor – DCD high*

Bit 4	STATUS-Senderregister
0	nicht leer
1	leer*

Bit 3	STATUS-Empfangsregister
0	nicht voll
1	voll*

Bit 2	Überlauffehler
0	kein Fehler
1	Fehler

Bit 1	Fehler bei Start- oder Stopbit
0	kein Fehler
1	Fehler

Bit 0	Paritätsfehler
0	kein Fehler
1	Fehler

tionen über den Zustand des ACIA. Die Bit-Belegung aller Register entnehmen sie bitte dem Kasten.

Ansonsten sind auf der Platine nur noch acht Elkos, ein Quarz und eine LED mit Widerstand zu finden. Die Daten-, Steuer-, Adreß- und Datensteuerleitungen des ACIA sind kompatibel zu der CPU und können daher direkt mit dem Expansions-Port verbunden werden. Hervorzuheben ist noch der IRQ-Ausgang, über denm der ACIA mit Low meldet, daß er etwas vom Computer will. Doch dazu später mehr.

Die Programmierung des ACIA 6551

Der ACIA stellt vier Register zur Verfügung, mit deren Hilfe man alle Funktionen im Griff hat (s. Kasten).

Wie man erkennen kann, ist das CONTROL-Register nur zur Steuerung des Übertragungsprotokols da und muß deshalb nur einmal vor der Datenübertragung richtig gesetzt, d.h. auf das andere Daten-Terminal eingestellt werden. Das COMMAND-Register hat wichtige Funktionen während der Datenübertragung. Die Bits 5 bis 7 müssen nur einmal richtig gesetzt werden, um eine eventuelle Paritätsüberprüfung bzw. -generierung auszuwählen. Über Bit 4 bestimmt man, ob der ACIA die ankommenden Daten sofort (um ein halbes Bit zeitvergrößert) wieder zum Sender zurückschicken soll. In diesem Fall muß der normale Sender des ACIA ausgeschaltet sein (Bit 2 und 3) gleich 0.

Die Bits 2 und 3 haben zwei Aufgaben. Einmal steuern sie das Hardware-Handshaking über den RTS auf high, bedeutet dies, daß der ACIA nicht bereit ist, weitere Daten zu empfangen. Außerdem kann man mit ihnen den Sende-Interrupt des ACIA ein- bzw. ausschalten. Ebenfalls über diese beiden Bits kann der ACIA veranlaßt werden, ein

Break-Zeichen zu schicken, das den Sender stoppt. Über Bit 1 kann man dem ACIA mitteilen, ob er Interrupts erzeugen darf oder nicht. Bit 0 meldet die Bereitschaft der Schnittstelle. Auf die genaue Benutzung dieser beiden Register gehen wir gleich noch näher ein.

Doch nun zum prinzipiellen Ablauf einer Datenübertragung. Zuerst wollen wir uns mit dem Senden von Daten beschäftigen. Sobald der ACIA dazu bereit ist, meldet er dies mit einem Low-Signal am IRQ-Ausgang. Voraussetzung dafür ist aber, daß das Erzeugen von Interrupts generell (COMMAND-Register Bit 1 = 0) und auch speziell des Sender-Interrupts (Bit 3 und 2) erlaubt ist. Die CPU reagiert auf das IRQ Signal und sucht nach dessen Verursacher. Dabei liest sie auch das STATUS-Register des ACIA. Dort kann man feststellen, ob der IRQ vom ACIA ausgelöst wurde und, wenn ja, aus welchem Grund (s. Bit-Belegung STATUS-Register). In diesem FAll müssen Bit 4 und 7 gesetzt sein. Nun schreibt man das nächste zu sendende Byte in das DATA-Register, und den Rest übernimmt der ACIA.

Das Empfangen von Daten läuft im Prinzip genauso ab. Wieder meldet der ACIA per IRQ, daß er ein Daten-Byte empfangen hat. Auch nun gibt die Abfrage des STATUS-Registers Aufschluß über den Grund des IRQ. Ist der Status in Ordnung, ließt man das Daten-Byte im DATA-Register. Dies muß natürlich vor dem nächsten Empfangs-Interrupt geschehen, da sonst ein Überlauffehler auftritt.

Installation des RS-232-Handlers

Zur Kommunikation des Atari mit dem ACIA benötigen wir ein Programm, das einige Forderungen erfüllen muß:

- Reset-Festigkeit
- automatisches Laden des Programms beim Booten des DOS
- einfache Handhabung der Schnittstelle über herkömmliche Befehle (GET, PUT usw.)
- einfaches Einstellen der Parameter (Parität, Baud-Rate usw.)

Dazu wollen wir einen Handler programmieren, der über den Namen R: anzusprechen ist. Da dieses Programm automatisch geladen werden soll, wird es als AUTORUN.SYS auf der DOS-Diskette abgespeichert. Der so gebootete Handler ist nun zu schützen, d.h., dem Betriebssystem muß mitgeteilt werden, daß der freie Speicher (z.B. des Basic) erst nach dem Handler beginnt. Zunächst wird DOS 2.5 im Bereich von $700 bis $1C6C geladen, direkt danach der R:-Treiber von $1EE0 bis $21CD. Die Adresse des untersten freien RAM-Bytes ($2400) kommt in MEMLO.

Natürlich soll der Treiber auch noch nach jedem Reset aktiv sein. Dazu verbiegen wir den Vektor DOSINI, über den normalerweise das DOS nach jedem Reset initialisiert wird, auf eine eigene Routine, die den Handler, die Handler-Tabelle, aber auch den ACIA initialisiert. Den ursprünglichen Wert von DOSINI müssen wird uns natürlich merken, um den normalen Ablauf weiterzuführen.

Nun ist unser Programm sozusagen nicht mehr kleinzukriegen. Aber schon taucht das nächste Problem auf. Der ACIA macht sich ja immer mit einem IRQ bemerkbar, wenn er der CPU etwas mitteilen will. Natürlich ist im Betriebssystem der Ataris ein IRQ, der vom ACIA kommt, zunächst nicht vorgesehen. Zum Glück haben die Entwickler bei Atari weitergedacht, so daß beim Auftreten eines IRQ durch den Vektor VIMIRQ gesprungen wird. Nun ist alles ganz einfach. Wir merken uns die Adresse in VIMIRQ, schreiben die unserer eigenen IRQ-Routine ein und springen (nachdem unsere Routine jetzt vom Betriebssystem aufgerufen wurde) danach zur gemerkten Adresse.

Die Interrupt-Routine des Handlers

Diese Routine wird immer dann aufgerufen, wenn ein IRQ auftritt. Zunächst müssen wir also mit Hilfe des STATUS-Registers (Bit 7) feststellen, ob der IRQ vom ACIA ausgelöst wurde. Falls dies der Fall war, wird dieses Register ausgewertet, um herauszufinden, wie darauf zu reagieren ist. Dabei gibt es drei Möglichkeiten.

1. Der ACIA möchte das nächste zu sendende Byte entgegennehmen.

   In diesem Fall wird das nächste Byte aus dem Sende-Buffer in das DATA-Register des ACIA geschrieben. Dieser Buffer (OBUF, im Handler mit 256 Byte Länge angelegt) hat den Vorteil, daß sich kürzere Datensätze blitzschnell vom Basic aus übertragen lassen. Sie landen dabei zunächst im Buffer, von wo sie dann nach und nach ausgegeben werden. Zur Verwaltung dieses Buffers gibt es zwei Pointer, PPOI und TPOI. PPOI gibt an, an welcher Stelle im Buffer das nächste Byte bei einem PUT- oder PRINT-Befehl geschrieben werden muß. TPOI besagt, welches Byte vom Buffer als nächstes an den ACIA zu übergeben ist. Da der Buffer nur 256 Bytes lang ist, kann es natürlich sein, daß beim schnellen Printen PPOI so schnell erhöht wird, daß er TPOI überholen würde. In diesem Fall muß der Handler warten, bis wieder Platz im Buffer ist. Dann kommt es im (Basic-Programm) zu Wartezeiten.
   Natürlich ist noch zu berücksichtigen, daß kein Byte ausgegeben wird, wenn der Buffer leer ist (PPOI = TPOI). Es muß aber auch darauf geachtet werden, ob die Gegenstelle vielleicht ein Xoff-Zei-

chen gesendet hat. In diesem Fall dürfen wir keine weiteren Daten senden, bis die Gegenstelle wieder ein Xon meldet. Diese Art des Übertragungsprotokolls (mit Xon und Xoff) nennt man Softare Handshaking. Das Abfragen des CTS-Eingangs für das Hardware-Handshaking übernimmt der ACIA selbst.

2. Der ACIA hat ein Daten-Byte empfangen.

   Auch hier existiert ein Buffer (IBUF, 256 Bytes). Es wird also das Byte im DATA-Register des ACIA in den Empfangs-Buffer an die Stelle des Pointers RPOI geschrieben. Falls der Buffer nun voll ist, muß der Handler dies der Gegenstelle mitteilen, damit diese zu senden aufhört. Das kann über die Leitung RTS geschehen oder durch Senden des Wertes in STOP (Xoff). Wenn wieder Platz im Buffer ist, ist entweder RTS auf low zu setzen oder der Wert in CONT (Xon) zu senden. Mit GET oder INPUT werden dann Daten aus dem Buffer an der Stelle IPOI ausgelesen. Diesmal ist zu beachten, daß der Buffer leer sein kann (IPOI = RPOI), so daß kein Byte gelesen werden kann. Dann wartet der GET-Befehl im Normalfall, bis ein Byte empfangen wird.

3. Der Status von DSR oder DCD hat sich geändert.

   Je nach Status des Akustikkopplers oder Computers können diese Meldugen entfallen oder verschieden sein. Deshalb läßt sich im Handler über CHECK bestimmen, ob und welches der beiden Signale beachtet wird. Näheres folgt später.

PROT	$6F0	0 = kein Handshake, 1 = Software-Handshake, 2 = Hardware-Handshake Hardware-Handshake bedeutet: – Der ACIA sendet über RTS-Ausgang seine Bereitschaft (RTS low). – Der ACIA empfängt über CTS-Eingang Bereitschaftssignale der Gegenstelle. Software-Handshake bedeutet: – Der ACIA sendet von Xon/Xoff-Zeichen als Bereit-/Nichtbereitmeldung. Der ACIA empfängt Xon/Xoff von der Gegenstelle.
STOP	$6F1	Zeichen für Xoff bei Softhandshake (norm. 19)
CONT	$6F2	Zeichen für Xon bei Softhandshake (norm. 17)
FULLFL	$6F3	0 = im Input-Buffer ist Platz, 1 = Input-Buffer voll
DATABRK	$6F4	0 = Normalzustand, 1 = Handler wartet auf Gegenstelle
ERR	$6F5	0 = alles o.k. sonst Fehler aufgetreten
STAT	$6F6	ACIA-Status (ACSTAT)
CHECK	$6F7	0= weder DSR noch DCD abfragen, 32 = nur DCD abfragen, 64 = nur DSR abfragen, 96 = beide abfragen
OFF	$6F8	0 = Normalzustand, 1 = nur noch Output-Buffer senden, dann Übertragung stoppen.
RWFL	$6F9	12 = bei GET warten, wenn Input-Buffer leer, 13 = nicht warten
NOCHR	$6FA	0 = Zeichen empfangen, 1 = kein Zeichen empfangen (nach GET-Befehl)

Die Variablen des Treiberprogramms.

Die einzelnen Handler-Routinen

Zu jedem richtigen Handler gehören die verschiedenen Treiberroutinen OPEN, CLOSE, PUT, GET und bei uns auch SPECIAL. Doch fallen diese für unsere Schnittstelle überraschenderweise recht kurz aus, denn die Interrupt-Routine nimmt uns schon viel Arbeit ab. Doch nun zu den Programmteilen.

OPEN

Hier wird eigentlich nur der Gegenstelle mitgeteilt, daß unser Computer zur Datenübertragung bereit ist. Außerdem wird das Flag für den Wait-Modus (s. später) entsprechend dem ersten Parameter des OPEN-Befehls gesetzt. Wie bei jeder Handler-Routine erfolgt vor dem Rücksprung im Y-Register die Übergabe der Fehlernummer (1 = Status OK).

CLOSE

Es wird nun das Stop-Flag (OFF) gesetzt. Dieses teilt der IRQ-Routine mit, daß nur noch der Buffer gesendet und dann die Gegenstelle ausgeschaltet wird.

GET

Falls kein Fehler aufgetreten ist und die BREAK-Taste nicht gedrückt wurde, wird das aktuelle Zeichen aus dem Buffer gelesen. Ist der Input-Buffer leer, hängt es davon ab, ob der Wait-Modus aktiviert ist oder nicht. Im ersten Fall wartet die GET-Routine so lange, bis ein Zeichen empfangen wurde, im zweiten wird die GET-Routine mit dem Rückgabewert 0 verlassen und das Flag NOCHR ($6FA) gesetzt. War der Input-Buffer voll, wird auch FULLFL zurückgesetzt, um der IRQ-Routine mitzuteilen, daß nun wieder Platz für Zeichen ist. Dies geschieht aber nur, wenn mindestens sechs Zeichen im Input-Buffer frei sind.

PUT

Zuerst wird der ACIA-Sende-Interrupt eingeschaltet und die BREAK-Taste abgefragt. Sofern noch Platz im Output-Buffer ist, wird ein Zeichen in diesen geschrieben, andernfalls gewartet, bis wieder Platz frei ist.

SPECIAL

Diese Routine ist für alle XIO-Befehle zuständig. Je nach XIO-Nummer wird zu verschiedenen kleinen Unterroutinen gesprungen. Eine Beschreibung der ein-

zelnen XIOs folgt später.

Sollten Ihnen noch verschiedene Teile des Handlers unverständlich sein, schauen Sie sich doch das abgedruckte Listing an, denn es ist ausführlich kommentiert. Eine Liste aller vom Handler verwendeten (eigenen) Variablen finden Sie im Kasten.

Bedienung des Handlers

Folgende Basic-Befehle werden vom R:-Handler unterstützt:

OPEN #Kanal, Wait-Modus (12 oder 13), 0, "R:" GET #Kanal, Byte

PUT #Kanal, Byte

PRINT #Kanal, Text

INPUT #Kanal, Text

XIO 40, #Kanal, Wert für COMMAND-Register des ACIA, Wert für CONTROL-Reg., "R":

XIO 41, #Kanal für PROT, O, "R:"

XIO 42, #, Wert für STOP, Wert für CONT, "R:"

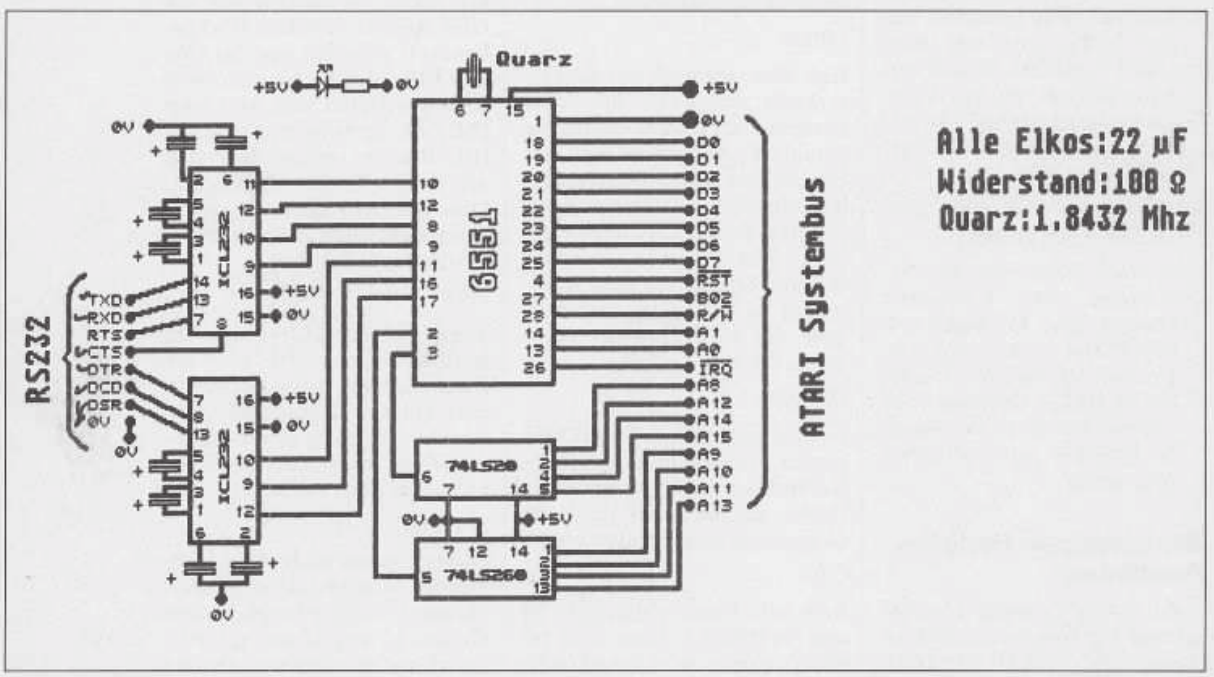
Nach RESET wird der Handler bzw. der ACIA auf 9600 Baud, keine Parität, 1 Stopbit und Software-Handshake initialisiert. STOP erhält den Standardwert 19 (CTRL-S), CONT den Wert 17 (CTRL-Q) und CHECK den Wert 32.

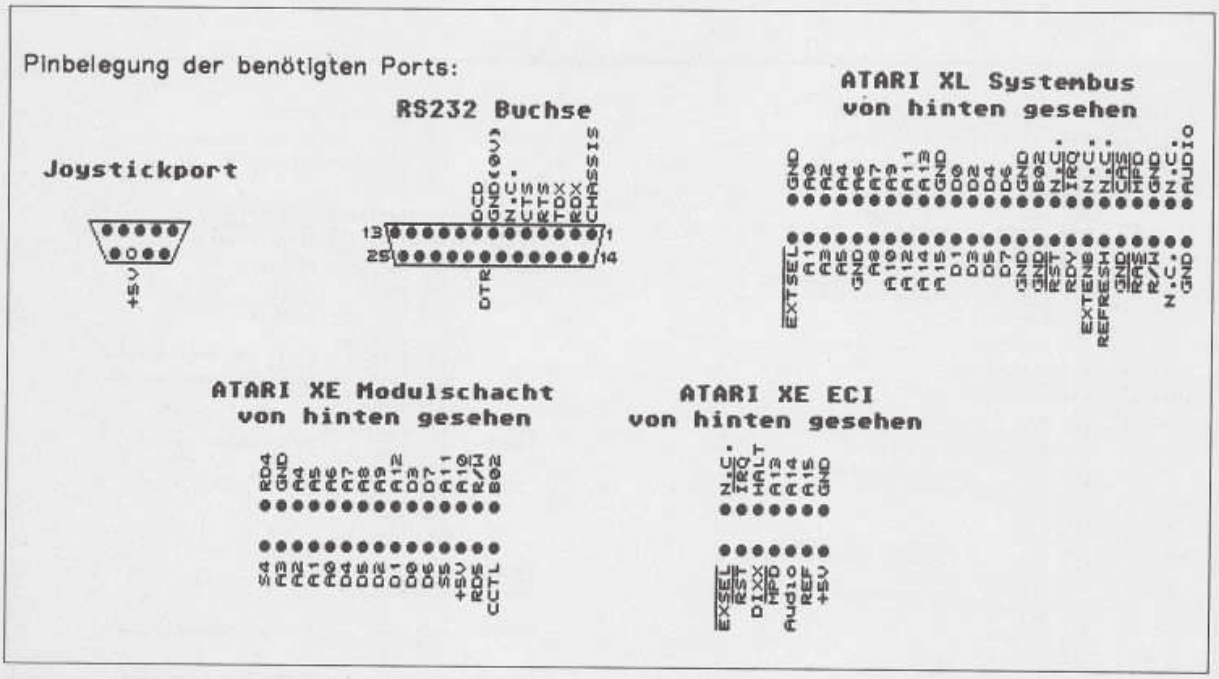
Die Programme

Listing 1 zeigt den R:-Handler als Atmas-2-Quellcode und ist für alle Assembler-Kundigen gedacht. Da Hardware-Erweiterungen auch für Basic-Programmierer interessant sind, haben wir in Listing 2 den gleichen Handler als "AMD"-File abgedruckt. Listing 3 ist ein kleines Basic-Programm, mit dem man schon erste Kontaktversuche unternehmen kann. Wir haben mit diesem Miniterminal-Programm mit dem VT52-Emulator des Atari ST kommuniziert, und das klappte einwandfrei.

Wenn Sie den Handler in Basic benutzen wollen, müssen Sie das Handlerfile als AUTORUN.SYS auf ihrer DOS-2.5-Diskette legen. Dazu tippen Sie das "AMD"-Listing ab (Name AUTORUN.SYS), oder Sie geben den Quellcode mit Atmas ein. In diesem Fall müssen Sie Ihr fertiges (assembliertes) Programm mit dem Monitor abspeichern ($1EE0 bis $21CD) und die Startadresse $1EE0 an dieses File anhängen (s. Assemblerecke 6/88).

Sie können den Handler auch in Ihre eigenen Assembler-Programme einbinden oder nachladen. In diesem Fall müssen Sie ihn aber erst initialisieren. Das geschieht mit einem Sprung zu (JSR) $1EE0. Dadurch wird der Handler eingetragen und resetfest gemacht. Damit der Handler nur eingetragen wird, ist ein Sprung zu $1EF2 erforderlich. Sie können das Handlerfile auch im ATMAS-(Monitor) oder BIBO-Assembler (BLOAD) laden. In beiden Fällen ist der Handler über $ 1EF2 zu initialisieren. Eventuell müssen Sie ihn dann an eine andere Stelle assemblieren. Wenn er nicht mehr direkt nach dem DOS liegt, entfällt die Änderung von MEMLO im Unterprogramm VECSET! Auch können Sie die Adressen für die Handler-Variablen (PROT, FULLFL usw.) beliebig abwandeln.





Schaltplan der RS-232-Schnittstelle

Beim Verbindungskabel der RS-232-Schnittstelle zu einem anderen Gerät ist zu beachten, ob es sich um eine Dateneinrichtung (z.B. Computer) handelt oder nicht. Dann müssen nämlich die Ausgänge unserer Schaltung mit den Eingängen des anderen Rechners verbunden werden (und umgekehrt). Das bedeutet, daß das Kabel RTS mit CTS, TxD mit RxD und DCD mit DTR vertauschen muß.

Der Handler erzeugt folgende Fehlermeldungen, die erst beim nächsten GET-Befehl ausgegeben werden:

137: Overrun (Handshake fehlgeschlagen oder falsche Baud-Rate)

142: Framing Error (Stopbits falsch, wird nicht ausgegeben.)

143: Parity Error (falsche Pairät oder Fehler bei der Übertragung)

Manchmal kann es sein, daß die Fehlermeldung erst nach RESET verschwinden. Man kann auch einen ACIA-RESET durch POKE $D101,0 auslösen. Dann muß man aber alle ACIA-Parameter (Baud-Rate usw.) neu setzen.

Andreas Binner

```
*****************************************
*          ASSEMBLERECKE                *
*  HANDLER FUER RS232 SCHNITTSTELLE     *
* ANDREAS BINNER & HARALD SCHOENFELD    *
*****************************************

          ORG $1EE0

IBUF      EQU $2200     ;Inputbuffer
OBUF      EQU $2300     ;Outputbuffer
ACDATA    EQU $D100     ;ACIA Datareg.
ACSTAT    EQU $D101     ;ACIA Statusreg.
ACCOMM    EQU $D102     ;ACIA Commandreg.
ACCONT    EQU $D103     ;ACIA Controlreg.

PROT      EQU $6F0      ;siehe Text
STOP      EQU $6F1
CONT      EQU $6F2
FULLFL    EQU $6F3
DATABRK   EQU $6F4
ERR       EQU $6F5
STAT      EQU $6F6
CHECK     EQU $6F7
OFF       EQU $6F8
RWFL      EQU $6F9
NOCHR     EQU $6FA

RPOI      EQU $6FC
IPOI      EQU $6FD
TPOI      EQU $6FE
PPOI      EQU $6FF

DOSINI    EQU $C
DOSVEC    EQU $A
VIMIRQ    EQU $216
COLOR1    EQU 709
COLOR2    EQU 710
COLOR4    EQU 712
MEMLO     EQU 743
ICAX1Z    EQU $2A
ICAX2Z    EQU $2B
ICCOM2    EQU $22
BRKKEY    EQU 17

STARTUP   LDA DOSINI    ;DOS-Initvektor
          STA MEM       ;merken
          LDA DOSINI+1
          STA MEM+1
          LDA #INIT     ;Neue Initadresse
          STA DOSINI
          LDA #INIT/256
          STA DOSINI+1
START2    LDA VIMIRQ    ;OS-Interruptvektor
          STA IRQ       ;merken
          LDA VIMIRQ+1
          STA IRQ+1
          LDA DOSVEC    ;DUP-Vektor
          STA DOS       ;merken
          LDA DOSVEC+1
          STA DOS+1
          JSR VECSET    ;->eigene Vektoren
*         RTS           ;setzen

**********************************
*Initalisierungsroutine nach jedem*
*Reset                           *
**********************************

INIT      DFB 32        ;Code "JSR" Befehl
MEM       DFW 0         ;"Platzhalter" fuer
*                       ;Sprungadresse
          JSR VECSET    ;->Vektoren setzen
          RTS
```

```
************************************
*Routine nach BASIC-Befehl DOS     *
************************************

DUP     LDA MEM         ;Gesemkte Werte
        STA DOSINI      ;wieder zurueck
        LDA MEM+1       ;schreiben
        STA DOSINI+1
        LDA DOS
        STA DOSVEC
        LDA DOS+1
        STA DOSVEC+1
        SEI             ;Interrupt aus
        LDA IRQ         ;OS-Interruptvek.
        STA VIMIRQ      ;wieder setzten
        LDA IRQ+1
        STA VIMIRQ+1
        CLI
        JMP (DOSVEC)      ;->zum DUP

**********************
*Vektorinitalisierung*
**********************

VECSET  LDA #DUP        ;Eigene Aufruf-
        STA DOSVEC      ;routine fuer DUP
        LDA #DUP/256    ;eintragen
        STA DOSVEC+1
        LDA #TAB        ;"R:"-Handler
        STA $32D        ;eintragen
        LDA #TAB/256
        STA $32E
        LDA #'R
        STA $32C
        LDA #0          ;Handler vor BASIC
        STA MEMLO       ;Zugriff schuetzen
        LDA #$24
        STA MEMLO+1
        LDA #52         ;Neue Farben
        STA COLOR2
        LDA #14
        STA COLOR1
        LDA #6
        STA COLOR4
        SEI             ;IRQ's aus
        LDA #INT        ;Vektor auf
        STA VIMIRQ      ;eigene Routine
        LDA #INT/256    ;setzen
        STA VIMIRQ+1
        CLI             ;IRQ's an
        LDA #9          ;kein Parity
        STA ACCOMM      ;1 Stopbit
        LDA #30         ;9600 Baud
        STA ACCONT
        LDA #19         ;Codes fuer
        STA STOP        ;Softhandshake
        LDA #17
        STA CONT
        LDA #32         ;32=DCD abfragen
        STA CHECK
        LDA #1          ;Softhandshake
        STA PROT
        LDA #0          ;Pointer und
        STA RPOI        ;Flags auf Null
        STA IPOI
        STA TPOI
        STA PPOI
        STA FULLFL
        STA ERR
        STA OFF
        STA RWFL
        STA DATABRK
        TAY             ;Einschalttext
TOP     LDA TEXT,Y      ;ausgeben
        STA (88),Y
        INY
        CPY #40
        BNE TOP
        RTS             ;...das war's

TEXT    ASC %       RS232-Treiber installiert       %

IRQ     DFW 0           ;zum merken...
DOS     DFW 0

TAB     DFW OPEN-1      ;Handlertabelle
        DFW CLOSE-1
        DFW GET-1
        DFW PUT-1
        DFW STATUS-1
        DFW SPECIAL-1
        JMP INIT

******************
*Interruptroutine*
******************

INT     PHA             ;alle Reg. retten
        TYA
        PHA
        TXA
        PHA
        LDA ACSTAT      ;ACIA-IRQ?
        STA STAT
        AND #128
        BNE ACIA        ;->Ja
BACK    PLA             ;alle Reg. holen
        TAX
        PLA
        TAY
        PLA
        JMP (IRQ)       ;->OS-Routine
ACIA    LDA CHECK       ;evtl.DCD und
        AND STAT        ;DSR abfragen
        BEQ OK
        JMP BACK
OK      LDA STAT        ;Daten angekommen?
        AND #8
        BEQ NOTFULL     ;->Nein
        JSR REC
NOTFULL LDA STAT        ;Daten ausgeben?
        AND #16
        BEQ NOTEMPT     ;->Nein
        JMP TRANS
NOTEMPT JMP BACK

************************************
*Angekommenes Byte in Inputbuffer*
************************************

REC     LDA STAT        ;Fehler?
        AND #5
        BNE ERROR       ;->Ja
        LDA PROT        ;Handshake?
        BEQ NSH         ;->Nein

        LDA ACDATA
        CMP STOP        ;Xoff von Gegenstelle?
        BEQ ANHALT      ;->Ja
        CMP CONT        ;Xon von Gegenstelle?
        BEQ WEITER      ;->Ja
        JMP INBUF

NSH     LDA ACDATA
INBUF   LDY RPOI        ;Byte in Buffer
        STA IBUF,Y
        INC RPOI
        LDA IPOI        ;Buffer voll?
        SEC
        SBC RPOI
        CMP #8
        BEQ VOLL        ;->Ja
        RTS

VOLL    JSR AUS
        LDA #1
        STA FULLFL
        RTS

ERROR   LDA STAT        ;Fehler merken
        STA ERR
        JMP VOLL

ANHALT  LDA #1          ;alle Sendevorgaenge
        STA DATABRK     ;des ACIA stoppen       RTS

WEITER  LDA #0          ;alle Sendevorgaenge
        STA DATABRK     ;wieder freigeben
        RTS

**********************************
*Byte aus Outputbuffer ausgeben*
**********************************

TRANS   LDA DATABRK     ;Auf Gegenstelle
*                       ;warten?
        BNE NOTRANS     ;->Ja

        LDA TPOI        ;Buffer leer?
        CMP PPOI
        BNE AUSG        ;->Nein

NOTRANS LDA ACCOMM      ;ACIA Sendeinterrupt
        AND #243        ;sperren
        ORA #8
        STA ACCOMM
        LDA OFF         ;wurde Kanal ge-
*                       ;schlossen?
        BNE ZU          ;->Ja
        JMP BACK
ZU      JSR AUS         ;Uebertragung Ende
        JMP BACK

AUSG    TAY             ;Byte senden
        LDA OBUF,Y
        STA ACDATA
        INC TPOI
        JMP BACK

****************************
*"Ready" an Sender melden*
****************************

AN      LDA PROT
        BNE HAND
        RTS
HAND    CMP #2          ;Hardwarehandshake
        BNE AN1
        LDA ACCOMM
        AND #243
        ORA #8
        STA ACCOMM
        RTS
AN1     LDA ACSTAT      ;Softwarehandshake
        AND #16
        BEQ AN1
        LDA CONT
        STA ACDATA
        RTS

**********************************
*"Nicht Ready" an Sender melden*
**********************************

AUS     LDA PROT
        BNE HAND1
        RTS
HAND1   CMP #2          ;Hardwarehandshake
        BNE AUS1
        LDA ACCOMM
        AND #243
        STA ACCOMM
        RTS
AUS1    LDA ACSTAT      ;Softwarehandshake
        AND #16
```

▶

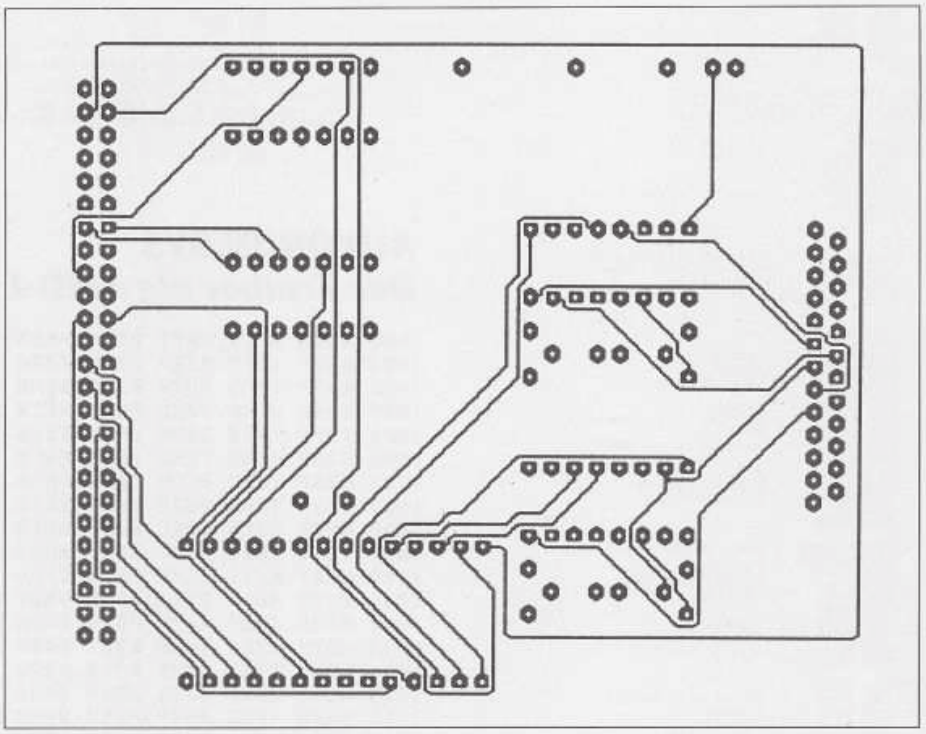
```
        BEQ AUS1
        LDA STOP
        STA ACDATA
AUS1    RTS

*********************************
*Die einzelnen Handlerroutinen:*
*********************************

OPEN    LDY #1          ;Status OK
        JSR AN          ;Sender an
        LDA ICAX1Z
        STA RWFL
        LDA #0          ;keine Fehler
        STA ERR
        RTS

CLOSE   LDA #1          ;Sender ausschalten
        STA OFF         ;wenn Buffer leer
        LDY #1          ;Status OK
        RTS

GET     LDA #0
        STA NOCHR
        LDA BRKKEY      ;BREAK ?
        BEQ BREAK       ;->Ja
        LDA IPOI        ;ein Byte da?
        CMP RPOI
        BNE BUFCHK      ;->Ja
        LDA ERR         ;Fehler aufgetreten?
        BNE FEHLER      ;->Ja
        LDA RWFL        ;auf Byte warten?
        CMP #12
        BEQ GET         ;->Ja
        LDA #1          ;"Kein Zeichen"-
        STA NOCHR       ;Flag setzen
        LDA #0
        JMP NEXT

BUFCHK  LDA FULLFL      ;war Buffer voll?
        BEQ GETCHR      ;->Nein
        LDA IPOI        ;wieder Platz
        SEC             ;fuer min. 10
        SBC RPOI        ;Zeichen?
        CMP #10
        BNE GETCHR      ;->Nein
        JSR AN          ;Gegenstelle an
        LDA #0          ;Full#flag auf 0
        STA FULLFL

GETCHR  LDY IPOI
        INC IPOI        ;Byte aus Buffer
        LDA IBUF,Y      ;holen
NEXT    LDY #1          ;Status OK
        RTS
BREAK   LDY #128        ;Status Break
        RTS

FEHLER  LDA ERR         ;Fehler auswerten
        AND #1          ;und entsprechende
        BNE PARITY      ;Meldung erzeugen
        LDA ERR
        AND #3
        BNE OVERRUN
        LDY #142
        RTS
PARITY  LDY #143
        RTS
OVERRUN LDY #137
        RTS

PUT     PHA
        LDA ACCOMM      ;ACIA Sendeinterrupt
        AND #243        ;ermoeglichen
        ORA #4
        STA ACCOMM
PUTL    LDA BRKKEY      ;BREAK ?
        BEQ BREAK       ;->Ja
        LDA PPOI        ;Buffer schon voll?
        CLC
        ADC #2
        CMP TPOI
        BEQ PUTL        ;->Ja
        LDY PPOI        ;Byte in Output-
        INC PPOI        ;buffer schreiben
        PLA
        STA OBUF,Y
STATUS  LDY #1          ;Status OK
        RTS

SPECIAL LDA ICCOMZ
        CMP #40         ;XIO 40?
        BEQ COMMAND     ;->Ja
        CMP #41         ;XIO 41?
        BEQ PROTOKOL    ;->Ja
        CMP #42         ;XIO 42?
        BEQ SOFTCHR     ;->Ja
        LDY #146        ;Nicht, dann Fehler
        RTS
COMMAND LDA ICAX1Z      ;XIO-Werte in
        STA ACCOMM      ;ACIA-Reg.
        LDA ICAX2Z      ;schreiben
        STA ACCONT
        LDY #1          ;Status OK
        RTS
PROTOKOL LDA ICAX1Z     ;wie oben...
        STA PROT
        LDY #1
        RTS
SOFTCHR LDA ICAX1Z
        STA STOP
        LDA ICAX2Z
        STA CONT
        LDY #1
        RTS

**************************************
*       Startadresse eintragen       *
*Achtung! Auch abspeichern im Monitor*
*SAVE mit Append von $2E0 bis $2E1    *
**************************************

        ORG $2E0
        DFW $1EE0
```

AUTORUN.SYS
Der Treiber als AMD-Listing

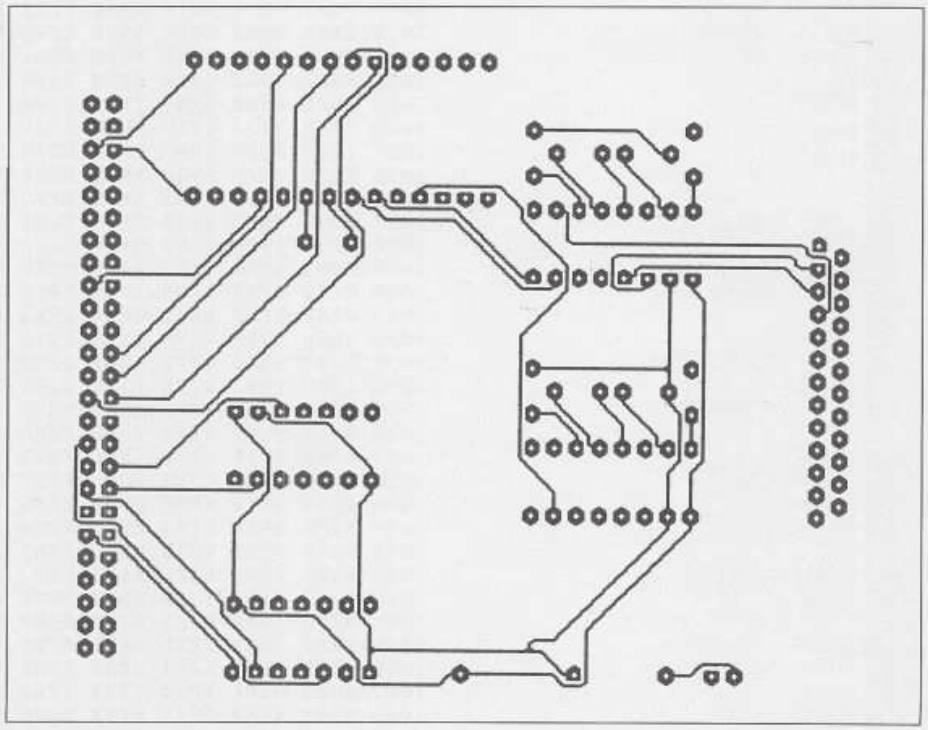
```
1000 MMMM NRTN VBYT KDRV HBRB TMKD 30892
1001 RBHB RNTM KJRV HDRV KJTM HDRB 30800
1002 KBTF RYHB NUTM KBTG RYHB NITM 31161
1003 KDRK HBND TMKD RCHB NFTM YRUH 31020
1004 TMFR YRRR RRYR UHTM FRKB RBTM 31514
1005 HDRV KBRN TMHD RBKB NDTM HDRK 30340
1006 KBNF TMHD RCGH KBNU TMHB TFRY 30985
1007 KBNI TMHB TGRY DHFV RKRR KJTU 31638
1008 HDRK KJTM HDRC KJNG HBYB RUKJ 30586
1009 TMHB YNRU KJDY HBYV RUKJ RRHB 31350
1010 NGRY KJYI HBNH RYKJ UIHB VFRY 31390
1011 KJRN HBVD RYKJ RFHB VHRY GHKJ 30867
1012 MFHB TFRY KJTM HBTG RYDH KJRJ 30940
1013 HBRY BTKJ TNHB RUBT KJTU HBMT 31151
1014 RFKJ TTHB MYRF KJYR HBMG RFKJ 30705
1015 RTHB MRRF KJRR HBMV RFHB MBRF 30300
1016 HBMN RFHB MMRF HBMU RFHB MDRF 30073
1017 HBMH RFHB MJRF HBMI RFKH CJCC 29394
1018 TMJT DHVH VRYH BRMF FRRR RRRR 31871
1019 RRRR RRRR UYUU TYTU TYRB UIGY 32923
1020 FDFJ FYFD GYRR FJFN GUGI FTFV 30818
1021 FVFJ FDGY GIRR RRRR RRRR RRRR 32445
1022 RRRR RRRR RRMD YRRD YTRB YTGY 32465
1023 YTJF YTJJ YTIV RVTM IHJH IHHK 31031
1024 IHKB RTBT HBMF RFYJ HRBR RHFH 30495
1025 KKFH KHFH FVNU TMKB MGRF YBMF 30600
1026 RFMR RUIV RDYR KBMF RFYJ RHMR 31408
1027 RUYR YMYR KBMF RFYJ TRMR RUIV 32177
1028 HRYR IVRD YRKB MFRF YJRD BRUF 31012
1029 KBMR RFMR TRKB RRBT VBMT RFMR 31429
1030 UYVB MYRF MRUY IVIN YRKB RRBT 31806
1031 KVMV RFJJ RRYY NNMV RFKB MBRF 31057
1032 UHNB MVRF VJRH MRRT FRYR BDYR 31826
1033 KJRT HBMU RFFR KBMF RFHB MDRF 30017
1034 IVFU YRKJ RTHB MIRF KJRR HBMI 30636
1035 RFFR KBMI RFBR RHKB MNRF VBMM 30574
1036 RFBR THKB RYBT YJMU RJRH HBRY 31568
1037 BTKB MHRF BRRU IVRD YRYR BDYR 31886
1038 IVRD YRKH CJRR YUHB RRBT NNMN 31427
1039 RFIV RDYR KBMR RFBR RTFR VJRY 31915
1040 BRRC KBRY BTYJ MURJ RHHB RYBT 31404
1041 FRKB RTBT YJTR MRMJ KBMY RFHB 30939
1042 RRBT FRKB MRRF BRRT FRVJ RYBR 31598
1043 RJKB RYBT YJMU HBRY BTFR KBRT 31336
1044 BTYJ TRMR MJKB MTRF HBRR BTFR 30913
1045 KRRT YRCY YRKD YKHB MJRF KJRR 31283
1046 HBMD RFFR KJRT HBMH RFKR RTFR 31109
1047 KJRR HBMK RFKD TTMR IYKB MBRF 30724
1048 VBMV RFBR TFKB MDRF BRUH KBMJ 30253
```

▶

Die Oberseite der Platine...



...und hier die Unterseite



```
1049 RFVJ RVMR NUKJ RTHB MKRF KJRR 31307
1050 IVDF YTKB MURF MRTU KBMB RFUH 31033
1051 NBMV RFVJ RKBR RHYR CYYR KJRR 32078
1052 HBMU RFKV MBRF NNMB RFCJ RRYY 31480
1053 KRRT FRKR HRFR KBMD RFYJ RTBR 31151
1054 RKKB MDRF YJRU BRRF KRHN FRKR 31197
1055 HMFR KRHJ FRIH KBRY BTYJ MURJ 31481
1056 RIHB RYBT KDTT MRBG KBMM RFTH 30743
1057 FJRY VBMN RFMR MTKV MMRF NNMM 31508
1058 RFFH JJRR YUKR RTFR KDYY VJYH 32282
1059 MRRC VJYJ MRTI VJYK MRTH KRJY 32130
1060 FRKD YKHB RYBT KDYC HBRU BTKR 31007
1061 RTFR KDYK HBMR RFKR RTFR KDYK 31306
1062 HBMT RFKD YCHB MYRF KRRT FRRR 31610
1063 MMMM NRRY NTRY NRTN  15106 *
```

RS232-Terminal

```
10 REM *** ASSEMBLERECKE "RS232" ***          ßS:ZP
20 REM ***  Mini-Terminalprogramm  ***        ßS:VK
30 REM                                        ßS:GO
40 REM CHR$(155) WIRD IN CHR$(13) UND         ßS:HK
50 REM CHR$(10) UMGEWANDELT (UND UM-          ßS:SZ
60 REM GEKEHRT).                              ßS:EW
70 KEYTAB=PEEK(121)+256*PEEK(122)             ßS:TQ
80 NOCHR=1786                                 ßS:VD
90 POKE 82,0                                  ßS:QR
100 ? CHR$(125)                               ßS:XG
110 XIO 40,#1,9,22,"R:"                       ßS:BO
120 XIO 41,#1,1,0,"R:"                        ßS:WA
130 XIO 42,#1,19,17,"R:"                      ßS:HB
140 OPEN #1,13,0,"R:"                         ßS:TH
150 POKE 764,255                              ßS:CZ
160 GET #1,A                                  ßS:QP
170 IF A=13 THEN A=155                        ßS:ET
180 IF A=10 THEN 210                          ßS:TH
190 IF PEEK(NOCHR)=1 THEN 210                 ßS:XQ
200 ? CHR$(A);                                ßS:WP
210 IF PEEK(764)=255 THEN 160                 ßS:RN
220 KEY=PEEK(KEYTAB+PEEK(764))                ßS:HM
230 IF KEY=155 THEN PUT #1,13:KEY=10          ßS:ZB
240 PUT #1,KEY                                ßS:ZI
250 POKE 764,255                              ßS:CA
260 GOTO 160                                  ßS:RX
```