Moppel Hardware

Ergänzungen
Serielles Interface

(Stand 17.09.2026)

Erstellt: Werner Römer
werner.roemer@t-online.de

Inhalt:

Seite 2 …..... Beschreibung, Übersichtsplan
Seite 3 …..... Portadressen, Modusregister
Seite 4 …..... UART-Controlregister, UART-Status/Datenregister
Seite 5 …..... Timer-Controlregister
Seite 6 …..... Timer-Datenregister, Init-Timer, Baudraten
Seite 7 …..... Daten empfangen mit Interrupt-Service-Routine
Seite 8 …..... Detailierte Softwarebeschreibung
Seite 12 …..... Besonderheiten MC6850
Seite 13 ……. Steckerbelegung DB9 zu DB25
Seite 14 …..... diverse Adapter

Beschreibung:

Diese Karte stellt die Schnittstelle zur Außenwelt dar und beinhaltet:

- Kasetteninterface
- Druckerschnittstelle
- 20mA Schnittstelle
- V24 Schnittstelle
- Summer (Bell)

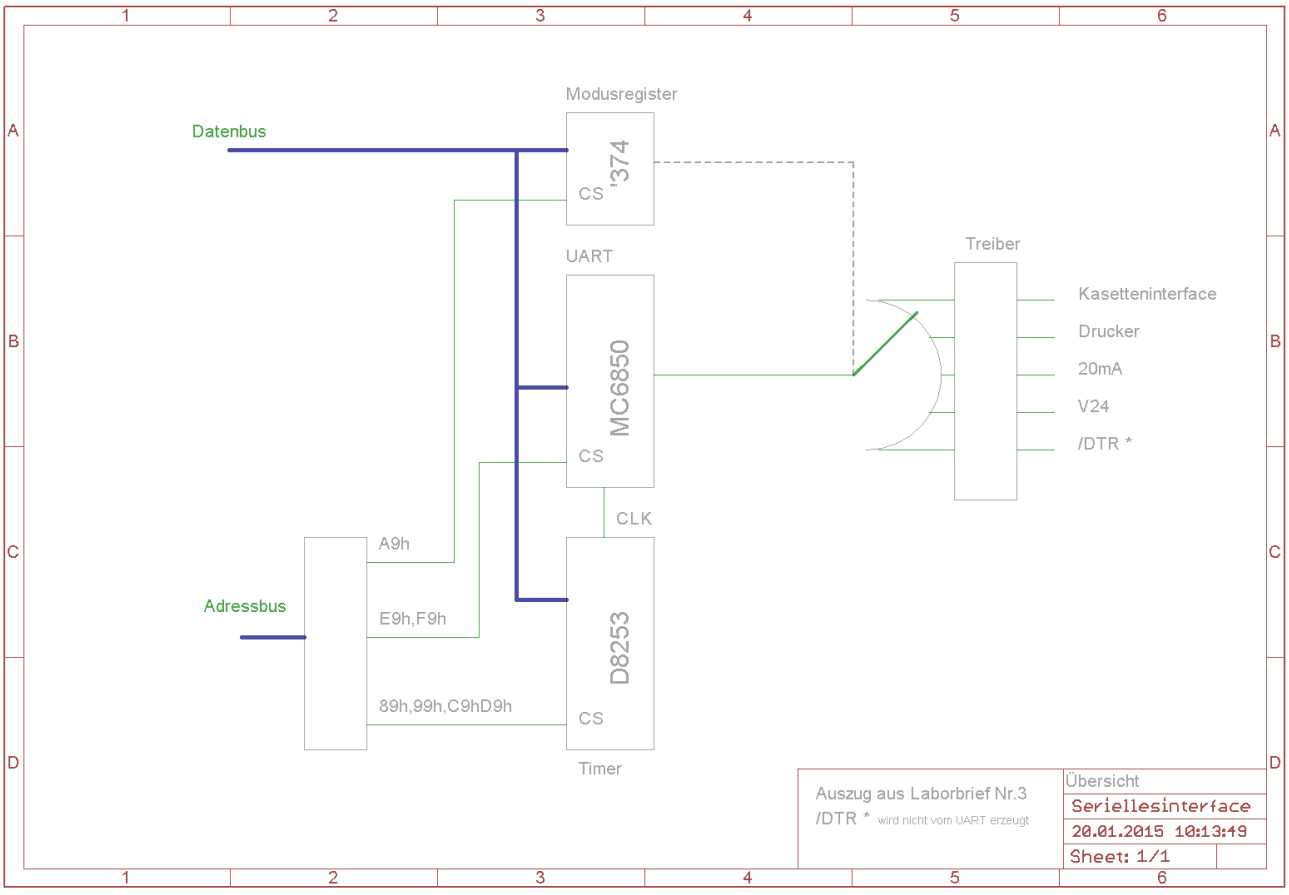
Diese können mit einem Umschalter (Modusregister) entsprechend angesteuert werden.
Die Baudrate wird vom Timer SAB8253 aus dem Systemtakt erzeugt, die der UART MC6850 für die Paralell/Seriell-Wandlung benötigt.
Schaltpläne findet Ihr im Laborbrief Nr.3 (ELO Sonderheft Nr.63)

Für den Betrieb muss die Karte entsprechend initialisiert werden:

- Gerät über das Modusregister auswählen
- Timer(0) mit der gewünschten Baudrate laden
- UART resetten und gewünschte Betriebsart laden (z.B. 8Daten, 1Stop, keine Parität)

Jetzt können die Daten geschrieben/gelesen werden.

Übersichtsplan:



verinfachte Darstellung aus ELO Sonderheft 63, bzw. Laborbrief 3

Portadressen:

Die Karte belegt 8 Adressen im IO-Bereich:

```
CS_MODE         EQU  0A9H   ; Modus (Eingangswahlschalter)
CS_BELL         EQU  0B9H   ; Bell  (Schnarre)
;
ACIA_CTL        EQU  0E9h   ; ACIA 6850 Control/Statusregister
ACIA_DAT        EQU  0F9H   ; Daten lesen/schreiben
;

TIMER_CTL       EQU  0D9H   ; Timer 8253 Controlregister
TIMER_0         EQU  89H    ;          Zähler 0 für Baudraten
TIMER_1         EQU  99h    ;          Zähler 1     * nicht benutzt
TIMER_2         EQU  0C9h   ;          Zähler 2     * nicht benutzt
;
```

Modusregister:

```
;
; Modusregister A9h
;
;       7 6 5 4 3 2 1 0
;       I I I I I I I I
;       I I I I I I I 0 = Relais 0
;       I I I I I I 0 x = Relais 1
;       0 0 0 0 0 1 x x = Modus 2 20mA
;       0 1 0 0 1 0 x x = Modus 3 V24
;       x x 0 1 0 0 x x = Modus 4 DTR
;       1 0 1 0 0 0 x x = Modus 5 Printer
;       1 1 x x x x x x = Kasette
;
;       0 0 0 0 0 1 1 1 = 07h Modus 2    (20mA)
;       0 1 0 0 1 0 1 1 = 4Bh Modus 3    (V24)
;       0 1 0 1 1 0 1 1 = 5Bh Modus + 4  (V24 mit DTR)
;       1 0 1 0 0 0 1 1 = A3h Modus 5    (Drucker)
;       1 1 0 0 0 0 1 1 = C3h Kasettenbetrieb
;
; für die Nutzung der Realais 0 u. 1 sind die Bits 0 u. 1entsprechend zu maskieren
; z.B. Das Kassettenlaufwerk zu starten
;
```

UART Controlregister:

```
;
; Controlregister ACIA E9h (aus Datenblatt Motorola M6850)
;
;       7 6 5 4 3 2 1 0
;       I I I I I I I I
;       I I I I I I 0 0 = clk/1
;       I I I I I I 0 1 = clk/16
;       I I I I I I 1 0 = clk/64
;       I I I I I I 1 1 = Reset
;       I I I 0 0 0     = ev 2Stop 7Datenbits
;       I I I 0 0 1     = od
;       I I I 0 1 0     = ev 1Stop      (gerade Parität)
;       I I I 0 1 1     = od            (ungerade Parität)
;       I I I 1 0 0     = -- 2Stop 8Datenbits
;       I I I 1 0 1     = -- 1Stop
;       I I I 1 1 0     = ev 1 Stop
;       I I I 1 1 1     = od
;       I 0 0           = RTS low
;       I 0 1           = RTS low IRQ enable
;       I 1 0           = RTS high
;       I 1 1           = RTS low
;       1               = IRQ enable
;
;       0 0 0 0 0 0 1 1 = 03h Baustein Reset
;       0 0 0 1 0 1 0 1 = 15h clk/16, 8Datenbit,1Stop,ohne Parität
;       0 1 0 1 0 1 0 1 = 55h ... mit RTS high
;       0 0 0 1 0 1 0 0 = 14h clk/1, 8Datenbit,1Stop,ohne Parität
;       0 1 0 1 0 1 0 0 = 54h wie oben mit RTS high
;
```

UART Statusregister:

```
;
; Statusregister ACIA E9h nur lesen
;
;       7 6 5 4 3 2 1 0
;       I I I I I I I I
;       I I I I I I I 1 = Empfangsregister voll
;       I I I I I I 1   = Senderegister leer
;       I I I I I 1     = DCD Eingang
;       I I I I x       = CTS Eingang
;       I I I 1         = Fehlerbits    -Frame
;       I I 1           =               -Overrun
;       I 1             =               -Parität
;       1               = IRQ-Ausgang   (ungerade Parität)
;
;
```

! Anmerkung:
Daten können nur richtig empfangen werden, wenn im UART der Takt durch 16 bzw. 64 geteilt wird (Bit 0), hierzu ist der Timmer entsprechend zu programmieren.

bei vollem Empfangsregister wird RTS nicht automatisch erzeugt,
die Datenflußsteuerung muss per Software nachgebildet werden.
Entweder DTR über das Modusregister oder RTS im Controlregister entsprechend setzen.

UART Datenregister:

Adresse F9h lesen/schreiben

TIMER Controlregister:

```
;
; Timer Controlregister D9h
;
;       7 6 5 4 3 2 1 0
;       I I I I I I I I
;       I I I I I I I 0 = Binärzähler
;       I I I I I I I 1 = BCD-Zähler
;       I I I I 0 0 0   = Modus 0 Impuls (einmalig, Softwareauslösung)
;       I I I I 0 0 1   = Modus 1        (einmalig, Hardwareauslösung G0-G2)
;       I I I I x 1 0   = Modus 2 Teiler durch n
;       I I I I x 1 1   = Modus 3 Rechteckgenerator
;       I I I I 1 0 0   = Modus 4 Triggerimpuls (Softwareauslösung)
;       I I I I 1 0 1   = Modus 5 Triggerimpuls (Hardwareauslösung G0-G2)
;       I I 0 0         = Zählerstand lesen
;       I I 0 1         = nur unteres Byte
;       I I 1 0         = nur oberes Byte
;       I I 1 1         = esrt unteres, dann oberes Byte
;       0 0             = Zähler 0
;       0 1             = Zähler 1
;       1 0             = Zähler 2
;       1 1             = unzulässig
;
```

TIMER Datenregister 0-2:

Im Moppel wird nur der Timer 0 für die Baudraten genutzt Adresse 89h

```
INIT:  mvi a,37h          ; Zähler 0 als BCD-Zähler
       out TIMER_CTL            ;
       mvi a,TIM_lo       ;
       out TIMER_0        ; Zähler untere hälfte
       mvi a,TIM_hi       ;
       out TIMER_0        ; Zähler obere hälfte
```

die anderen stehen zur freien Verfügung, Ausgänge und die Gate-Steuerung sind an der 25pol Sub-D Buchse herausgeführt – Spielwiese für eigene Experimente...

Baudraten:

Alle Baudraten im Moppel leiten sich aus dem halbe CPU-Takt ab, für die unterschiedlichen Datenübertragungsraten gibt es im Monitorprogramm Speicherstellen die einmal den CPU-Takt und die Baudrate darstellen:

0006h CPU-Quarz (4,6 oder 8Mhz)

000Eh Baudrate Drucker (4800Bd)

0016h Baudrate Kasetteninterface (1200Bd)

001Eh Baudrate V24 (4800Bd)

Anhand des CPU-Quarzes wird eine von drei Tabellen vorbereitet, die dann über die gewünschte Baudrate ausgelesen und im Timer0 geladen wird. Jede Tabelle enthält 8 Werte für die Baudraten 4800, 110, 300, 600, 1200, 2400, 4800 und 9600Bd. Die 4800Bd sind doppelt vorhanden, einmal als Standardwert am Tabellenanfang und einmal über Zeiger zugänglich.

Details können dem Listing im Anhang entnommen werden.

Datenempfang im IRQ:

Wenn der Moppel mit der neuen Welt verbunden werden soll (Window, etc.) gibt es beim Datenempfang erhebliche Probleme. Mit dem Hyperterminal funktioniert es, wenn die Datenübertragung aber per Batch läuft, gibt es nur noch Datenmüll.

Ursache sind die FIFO`s auf der PC Seite, dort sprudeln trotz gesetztem RTS noch weiter Daten, bis der FIFO geleert ist. Bei einigen PC's ist er zwar abschaltbar, trotzdem schiebt er noch ein zwei Bytes.

Lösung: Datenempfang mit einem „Software-FIFO“. Über eine InterruptService-Routine werden die Daten in einem Buffer abgelegt und wenn Zeit ist von dort dem Anwendungsprogramm zur Verfügung gestellt. Die Bufferverwaltung muss frühzeitig ein RST an den Sender schicken damit noch für die restlichen Bytes aus dem Sende-FIFO Platz bleibt.

```
;
;=====================================================================
;
; V24 mit IRQ Routine - Test fuer Datenempfang
;
; Parameter zur Uebersicht
;
; buffer  equ 9000h ; fuer BIN-Datei
; zbuff             equ 8400h ; Software FIFO
;
;            v-Schreibzeiger (wrz)
;            I
; 8400h I------------...------------------+-...--I 843fh (maxbuf)
;         ^                                ^
;         I- Lesezeiger (rdz)       842ah +------ RTS Setzen (rtsbuf)
;
; maxbuf  equ 3fh             ; Bufferlaenge
; rtsbuf  equ 2ah             ; Bufferposition fuer RTS
;
```

Die IS-Routine schreibt den Buffer solange voll, bis der RTS-Punkt erreicht ist und setzt dann das Stop-signal für den Sender, bleibt aber weiterhin Empfangsbereit um die restlichen Bytes einzusammeln. Die Anwendung kann nun, gesteuert über den Lesezeiger die Daten in Ruhe abholen. Wenn der Lesezeiger die Position vom Schreibzeiger erreicht hat, bedeutet dies keine Daten mehr vorhanden und stellt alles wieder auf Anfang. Damit der Moppel nicht bis zum Sankt-Nimmerleins-Tag wartet, beendet eine Zeitüberwachung nach ca. zwei Sekunden die Übertragung.

Speicherzuordnung:

```
;=====================================================================
;
fifo    EQU 2D10h       ; Softwarefifo
buffer  EQU 2D50h       ; Hex-Zeilenbuffer
baud    equ 0020h       ; fuer 9600Bd
timeout equ 0800h       ; ca 2s
;
fiforts equ 28h         ; Bufferposition fuer RTS
fifomax equ 40h         ; Fifo Groesse
;
; Zwischenspeicher fuer Buffer(FIFO) Steuerung
;
        org 2D00h
buffz   dw  0000h       ; Schreibzeiger fuer Zielbuffer
wrz     dw  0000h       ; Schreibzeiger fuer IRQ
rdz     dw  0000h       ; Lesezeiger
v24flag db  00h         ; 1000 0001
;                       ; :  : :  :
;                       ; :  : :  +--- RTS punkt erreicht
;                       ; :  : +------ Timeout
;                       ; :  +-------- INIT Flag 0=FIFO Zeiger setzen
;                       ; :                     1=FIFO Betrieb
;                       ; +---------- Buffer voll
;                       ;
;
```

Damit die beiden Programmteile, ISR und Leseroutine sich über den Ablauf verständigen können, bedarf es ein paar Absprachen. Neben der Buffergrösse der RTS-Position werden ein paar Speicherplätze im RAM benötigt. Ich habe den FIFO auf 64Byte gesetzt, da hier z.B. die Intelhex-Zeilen locker reingehen. Dort ist mit „Buffer“ nochmals ein Bereich für 64Bytes reserviert damit dem HEX-Interpreter eine Zeile komplett übergeben werden kann.

Darüber hinaus müssen ja irgendwo die Zeiger zwischengelagert werden und das von mir getaufte „V24flag“. Dies spiegelt die verschiedenen Betriebszustände vom Software-FIFO wieder.

Bit-0 wird mit Erreichen des RTS-Punktes von der ISR gesetzt
und vom Leseprogramm wieder zurückgestellt.

Bit-3 wird vom Leseprogramm nach ausbleiben von Daten gesetzt und von der Anwendung entsprechend berücksichtigt, also Programmende.

Bit-4 zeigt an ob das Leseprogramm schon mal benutzt wurde und wird von diesem beim ersten Durchlauf gesetzt, damit die Initialisierung nicht dauern alles wieder auf Null setzt.

Bit-7 ist der Notnagel falls mal irgendwas fürchterlich durcheinander kommt

Softwaredetails:

```
;
;---------------------------------------------------------------------
;
; IRQ vorbereiten
;
irqi:   lxi     h,int75         ; Spungtabelle
        mvi     m,0c3h          ; = JMP
        inx     h               ; IRQ Seviceroutine
        lxi     d,irqs          ; In Sprungtabelle
        mov     m,e             ;
        inx     h               ;
        mov     m,d             ; eintragen
        mvi     a,1bh           ; 0001 1011 irq7.5 freigeben
        sim                     ;
        ei                      ; IRQ freigeben
        ret                     ;
;
```

hier wird das Sprungziel für RST7.5 vorbereitet.

Hintergrund: Der 8085 springt beim Eintreffen von RST7.5 zur Adresse 003Ch, ist natürlich im EPROM und somit nicht ohne Umstand zu ändern. Hier steht in kluger Vorausicht schon in Sprungbefehl zum „System-RAM“ wo dann sehr einfach der eigentliche Sprung zur Interrupt-Service-Routine eingetragen wird.

Natürlich gilt es die V24-Schnittstelle mit Baudrate und Betriebsart sowie der Wahlschalter auf der 87erSchnittstellenkarte entsprechend einzustellen.

```
        ;
        call    iniv24          ; Karte initialisieren
        call    irqi            ; IRQ Serviceroutinen einrichten
        mvi     a,95h           ; 1001 0101 = RX IRQ und
        out     uart_c          ; RTS freigeben
        ;
```

Mit der Maske 1001 0101(bin) wird im UART der Interrupt für den Empfang freigeschaltet. Und natürlich RTS gelöscht, damit der Sender loslegen kann...

Interupt-ServiceRoutine:

```
;
; IRQ Serviceroutine
; Ein Byte uber V24 empfangen und
; im Empfangsbuffer (FIFO) ablegen
;
irqs:   push    psw             ; alles sichern
        push    h               ;
        push    d               ;
        push    b               ;
        in      uart_c          ; irq loeschen
        lhld    wrz             ; Zeiger auf Zeilenbuffer
        in      uart_d          ; Daten holen
        mov     m,a             ; und ablegen
        inx     h               ;
        shld    wrz             ; Zeiger sichern
        mvi     a,fiforts       ; Zeiger an RTS Grenze
        cmp     l               ;
        jz      setrts          ; RTS Setzen
        mvi     a,fifomax       ; Test auf Ende Zeilenbuffer
        cmp     l               ;
        jz      irqbs           ; Zeilenbuffer Ende erreicht

irqe:   pop     b               ; alles wieder zurueck
        pop     d               ;
        pop     h               ;
        pop     psw             ;
        ei                      ; IRQ freigeben
        ret                     ;

setrts: mvi     a,0d5h          ; RTS gesetzt
        out     uart_c          ; IRQ Freigabe
        lda     v24flag         ;
        ori     01h             ;
        sta     v24flag         ; RTS-Flag setzen
        jmp     irqe            ;

irqbs:  lxi     h,fifo          ; Buffer wieder auf Anfang
        shld    wrz             ;
        mvi     a,55h           ; RTS gesetzt
        out     uart_c          ; IRQ gesperrt
        lda     v24flag         ;
        ori     81h             ; 1000 0001 Error u. V24 Flag setzen
        sta     v24flag         ;
        jmp     irqe            ;
;
;=================================================================
```

Da der Interrupt zu einem beliebigem Zeitpunkt eintreffen kann, sind hier alle Register zu sichern und beim Verlassen so wieder herzustellen als sei nichts geschehen. Zunächst wird über „in uart_c“ der Interrupt gelöscht, damit kein Dauerfeuer entsteht, anschliessend der Schreibzeiger geladen und das Datenbyte in den FIFO abgelegt. Nun kommt die Prüfung ob der RTS-Punkt schon erreicht ist, wenn ja wird diese gesetzt über das V24Flag Bit-0 dem Leseprogramm angezeigt. Das Buffer-Ende wird noch überprüft, sollte eigendlich nie vorkommen und damit ist die Routine fertig und der Moppel hat wieder etwas Luft für andere Dinge.

FIFO auslesen:

```
;---------------------------------------------------------------------
;
; Rx    Daten aus FIFO lesen
;
; (A)=Zeichen
; (A)=0 und Z=1 Timeout V24-Flag 0000 1000b
;
rx:     push    b               ;
        push    d               ;
        push    h               ;
        lda     v24flag         ;
        ani     10h             ; Init-Flag ?
        cz      setirq          ; nein -> Zeiger auf Anfang
rxz_l1  lxi     b,timeout       ; Abbruchbedingung laden
rxz_l2: di                      ;
        lhld    wrz             ; Daten in FIFO ?
        xchg                    ;
        lhld    rdz             ;
        mov     a,l             ;
        cmp     e               ;
        jc      rxzrb           ; ja --> Daten lesen
        lda     v24flag         ;
        ani     01h             ; RTS gesetzt ?
        jnz     rtscl           ; RTS und IRQ freigeben
        ei                      ;
        lda     v24flag         ;
        ani     080h            ; FIFO Ende ?
        jnz     rxz_e           ; ja --> Ende mit Bufferueberlauf
        call    dely1           ;
        dcr     c               ; Timeout ?
        jnz     rxz_l2          ;
        dcr     b               ;
        jnz     rxz_l2          ; warten
        lda     v24flag         ;
        ori     08h             ; 0000 1000 Timeout
        sta     v24flag         ;
        xra     a               ; keine Daten im FIFO (A)=0 Z=1

rxz_e:  pop     h               ;
        pop     d               ;
        pop     b               ;
        ret                     ; Ende RX
;
;
; setirq        IRQ, FIFO, Flags setzen
;
setirq: lxi     h,fifo          ;
        shld    wrz             ;
        shld    rdz             ; auf Anfang
        mvi     a,10h           ;
        sta     v24flag         ; Init-Flag setzen
        ret                     ;
;
```

Im ersten Schritt wird nachgesehen ob die Leseroutine schon mal benutzt wurde, wenn nein werden ein paar Dinge richtig gerückt. Mit dem Vergleich der beiden Zeiger wird der Füllstand ermittelt, wenn ungleich Null wird ein Byte aus dem Buffer ausgelesen

```
;
; Buffer auslesen
;
rxzrb:  mov     a,m             ; Byte holen
        inx     h               ;
        shld    rdz             ;
        ei                      ;
        jmp     rxz_e           ; warten auf naechstes Byte

;
; RTS und IRQ freigeben
;
rtscl:  lxi     h,fifo          ; Buffer
        shld    wrz             ;
        shld    rdz             ; auf Anfang
        lda     v24flag         ;
        ani     0feh            ; 1111 1110b
        sta     v24flag         ; RTS-Flag loeschen
        mvi     a,95h           ; 1001 0101 = RX IRQ und
        out     uart_c          ; RTS freigeben
        ei
        jmp     rxz_l1          ;
;
;
;-----------------------------------------------------------------
```

wenn keine Daten mehr im Buffer stehen, gibt es zwei Möglichkeiten. Entweder ist alles fertig, dann greift der Timeout und das Programm wird beendet oder RTS steht noch an. Das muss natürlich wieder freigegeben werden und alles wieder auf Anfang gesetzt werde. So können dann die nächsten Bytes eintrudeln und verfrühstückt werden.

Besonderheiten:

Im MC6850 ist die Interruptbedingung nicht eindeutig!
/IRQ wird auch vom /DCD Eingang beeinflusst.

Auf der Multi-IO-Karte habe ich diesen Eingang auf GND gelegt, da ich keinen weiteren Treiber einbauen wollte und er für den RTS-Handshake nicht benötigt wird.
Dadurch legt er aber ein IRQ an, die über die Abfrage

```
in      uart_c      ; Statusregister
ani     01h         ; Empfangsbuffer voll ?
Jz      irq_end     ; nichts machen
;                   ;
in      uart_d      ; Datum lesen
.
.
```

nochmals eindeutig selektiert werden muss.

In der 87erV24-Interfacekarte ist der DCD-Eingang über den Umschalter auf DSR (Stift 6) gelegt und über die V24 zum PC geführt, wird also bei der Übertragung entsprechend gesetzt.

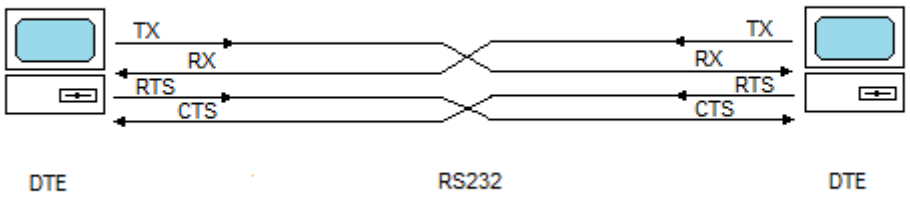
Steckerbelegung:

Gegenüberstellung DB25 (alt) nach DB9 (neu)

PIN 9pol	PIN 25pol	RS232	In/Out	Bezeichnung	Anmerkung
1	8	DCD	In	Data Caririer Detect	Signaltrger erkannt
2	3	RxD	in		Empfangsdaten
3	2	TxD	out		Sendedaten
4	20	DTR	out	Data Terminal Read	
5	7	GND		Ground	
6	6	DSR	in	Data Set Read	Endgert bereit
7	4	RTS	out	Request to send	Sendeanfrage
8	5	CTS	in	Clear to Send	Fertig zum Senden
9	22	RI	in	Ring Indicator	
-	12	-		Speed Mode Detector	
-	23	SPDS		Speed select	

Das RS232-Interface war ursprünglich nur für die Steuerung von Modems verwendet. Für den Datenaustauch zwischen zwei Rechnern genügen oft die Leitungen RxD, TxD und GND.

Für den Moppel sollte das RTS/CTS Protokoll genutzt werden!



Quelle: https://www.mikrocontroller.net/articles/Datei:RS232_dte.png

Anschlußbelegung:

