```
From - Sun Nov 30 16:25:48 1997
Received: from dosbuster.home.dd (root@dial8.rz.fh-heilbronn.de [141.7.42.8])
        by mx1.eskimo.com (8.8.8/8.8.8) with ESMTP id NAA29015
        for <voyager@eskimo.com>; Sun, 30 Nov 1997 13:59:10 -0800
Received: (from poldi@localhost) by dosbuster.home.dd (8.7.5/8.7.3) id RAA01078 for voyager@eskimo.com; Sun, 30 Nov 1997
17:21:37 +0100
From: Daniel Dallmann <Daniel.Dallmann@studbox.uni-stuttgart.de>
Message-Id: <199711301621.RAA01078@dosbuster.home.dd>
Subject: Re: Novaterm9.6
To: voyager@eskimo.com (Nick Rossi)
Date: Sun, 30 Nov 1997 17:21:37 +0100 (MET)
In-Reply-To: <199711301228.EAA22736@eskimo.com> from "Nick Rossi" at Nov 30, 97 04:28:37 am
Reply-To: Daniel.Dallmann@studbox.uni-stuttgart.de
X-Mailer: ELM [version 2.4 PL24]
MIME-Version: 1.0
Content-Type: text/plain; charset=iso-8859-1
Content-Transfer-Encoding: 8bit
X-UIDL: 25148198b8c6b2dd16493fb473a6f131
X-Mozilla-Status: 8011
```

Hello,

> > Did you made any changes to Novaterm's interface to serial-device-drivers ?
> > Isn't it time for a new UP9600 driver ?

> As a matter of fact, I am working on a new version of Novaterm and
> everything is completely different. It now uses relocatable code to load
> modules into any place in memory, and allocates the memory ahead of time. I
> wrote an assembler to specifically generate the code plus the tables needed
> to relocate it. There are also lots of rules for using zero page pointers,
> initialization functions, etc. In order to adapt UP9600 to this version, I
> would have to assemble the source code here with my assembler and make sure
> there are no conflicts.

sounds very interesting. did you know, that i've also written an assembler
that is able to generate relocatable code ? :)

> > Did you get any feedback, related to the UP9600 driver ?

> I'm pretty sure that some people are using it. I've heard mention of it
> from time to time.

> > i just want to make sure, that i'm still here and ready to do some work
> > for NT9.6 in case of drivers! :-)

> Perhaps you could send me the source for adaptation. I'll send you back a
> beta copy of the new version (still a long way off from finished).

i have just written a (new) sample implementation of my driver, that can
even be used from BASIC V2.0.
It uses both receive and send buffers (each of 256 bytes).

I'll just append the source.

```
==============================================================================
Contents:

 1) source code (luna format)
 2) uuencoded binary
 3) how to interface with BASIC V2.0

==============================================================================
1) source code

        ;; rewritten (based on LUnix' getty code)
        ;;  UP9600
        ;;    (universal) device dirver for RS232 userport interface with
        ;;    special wiring.

        ;; Nov 23 1997 by Daniel Dallmann

        org $c000

        ;; provided functions

.global install                 ; install and (probe for) UP9600 (c=error)
.global enable                  ; (re-)enable interface
.global disable                 ; disable interface (eg. for floppy accesses)

        ;; rsout and rsin both modify A and X register

.global rsout                   ; put byte to RS232 (blocking)
.global rsin                    ; read byte from RS232 (c=try_again)


        jiffies equ $a2         ; lowest byte of system's jiffie counter

        original_irq equ $ea31  ; (must incease jiffie-counter !)
        original_nmi equ $fe47

        nmi_vect equ 792
```

```
         irq_vect equ 788

         ;; NMI part
         ;; nmi_startbit and nmi_bytrdy must be in the same code page !!!

 nmi_startbit:
         pha
         bit  $dd0d              ; check bit 7 (startbit ?)
         bpl  +                  ; no startbit received, then skip

         lda  #$13
         sta  $dd0f              ; start timer B (forced reload, signal at PB7)
         sta  $dd0d              ; disable timer and FLAG interrupts
         lda  #<nmi_bytrdy       ; on next NMI call nmi_bytrdy
         sta  nmi_vect           ; (triggered by SDR full)

     + - pla                     ; ignore, if NMI was triggered by RESTORE-key
         rti

 nmi_bytrdy:
         pha
         bit  $dd0d              ; check bit 7 (SDR full ?)
         bpl  -                  ; SDR not full, then skip (eg. RESTORE-key)

         lda  #$92
         sta  $dd0f              ; stop timer B (keep signalling at PB7!)
         sta  $dd0d              ; enable FLAG (and timer) interrupts
         lda  #<nmi_startbit     ; on next NMI call nmi_startbit
         sta  nmi_vect           ; (triggered by a startbit)
         txa
         pha
         lda  $dd0c              ; read SDR (bit0=databit7,...,bit7=databit0)
         cmp  #128               ; move bit7 into carry-flag
         and  #127
         tax
         lda  revtab,x           ; read databits 1-7 from lookup table
         adc  #0                 ; add databit0
         ldx  wr_rptr            ; and write it into the receive buffer
         sta  recbuf,x
         inx
         stx  wr_rptr
         sec
         txa
         sbc  rd_rptr
         cmp  #200
         bcc  +
         lda  $dd01              ; more than 200 bytes in the receive buffer
         and  #$fd               ; then disbale RTS
         sta  $dd01
     +   pla
         tax
         pla
         rti

         ;; IRQ part

 new_irq:
         lda  $dc0d              ; read IRQ-mask
         lsr  a
         lsr  a                  ; move bit1 into carry-flag (timer B - flag)
         and  #2                 ; test bit3 (SDR - flag)
         beq  +                  ; SDR not empty, then skip the first part
         ldx  outstat
         beq  +                  ; skip, if we're not waiting for an empty SDR
         dex
         stx  outstat
         bne  +                  ; skip, if we're not waiting for an empty SDR

         php
         jsr  send_nxtbyt        ; send the next databyte
         plp

     +   bcs  +                  ; skip if there was no timer-B-underflow
         jmp  $ea81              ; return from IRQ

     +   ; keyscan IRQ

         sec                     ; (a lost SDR-empty interrupt, would
         lda  jiffies            ; totally lock up the sender. So i've added
         sbc  stime              ; a timeout)
         cmp  #16                ; (timeout after 16/64 = 0.25 seconds)
         bcc  +                  ; no timeout jet
         jsr  send_nxtbyt        ; send the next databyte
     +   jmp  original_irq

         ;; send next byte from buffer

 send_nxtbyt:
         lda  jiffies            ; remember jiffie counter for detecting
         sta  stime              ; timeouts
         lda  $dd01              ; check CTS line from RS232 interface
         and  #$40
         beq  +                  ; skip (because CTS is inactive)
         ldx  rd_sptr
         cpx  wr_sptr
         beq  +                  ; skip (because buffer is empty)
         lda  sndbuf,x
         inx
         stx  rd_sptr
         cmp  #128               ; move bit7 into carry-flag
         and  #127               ; get bits 1-7 from lookup table
         tax
         lda  revtab,x
         adc  #0                 ; add bit0
         lsr  a
         sta  $dc0c              ; send startbit (=0) and the first 7 databits
         lda  #2                 ; (2 IRQs per byte sent)
         sta  outstat
         ror  a
         ora  #127               ; then send databit7 and 7 stopbits (=1)
         sta  $dc0c              ; (and wait for 2 SDR-empty IRQs or a timeout
     +   rts                     ; before sending the next databyte)

         ;; get byte from serial interface

 rsin:   ldx  rd_rptr
         cpx  wr_rptr
         beq  ++                 ; skip (empty buffer, return with carry set)
         lda  recbuf,x
         inx
         stx  rd_rptr
         pha
         txa
         sec
         sbc  wr_rptr
         cmp  #256-50
         bcc  +
         lda  #2                 ; enable RTS if there are less than 50 bytes
         ora  $dd01              ; in the receive buffer
         sta  $dd01
         clc
     +   pla
     +   rts

         ;; put byte to serial interface

 rsout:  ldx  wr_sptr
         sta  sndbuf,x
         inx
     -   cpx  rd_sptr            ; wait for free slot in the send buffer
         beq  -
         stx  wr_sptr
         lda  outstat
         bne  +
         lda  jiffies
         eor  #$80
         sta  stime              ; force timeout on next IRQ
     +   rts

         ;; install (and probe for) serial interface
         ;; return with carry set if there was an error

 inst_err:
         cli
         sec
         rts

 install:
         sei
         lda  irq_vect
         cmp  #<original_irq
         bne  inst_err           ; IRQ-vector already changed
         lda  irq_vect+1
         cmp  #>original_irq
         bne  inst_err           ; IRQ-vector already changed
         lda  nmi_vect
         cmp  #<original_nmi
         bne  inst_err           ; NMI-vector already changed
         lda  nmi_vect+1
         cmp  #>original_nmi
         bne  inst_err           ; NMI-vector already changed

         ldy  #0
         sty  wr_sptr
         sty  rd_sptr
         sty  wr_rptr
         sty  rd_rptr

         ;; probe for RS232 interface

         cli
         lda  #$7f
         sta  $dd0d              ; disable all NMIs
         lda  #$80
         sta  $dd03              ; PB7 used as output
         sta  $dd0e              ; stop timerA
         sta  $dd0f              ; stop timerB
         bit  $dd0d              ; clear pending interrupts
         ldx  #8
     -   stx  $dd01              ; toggle TXD
         sta  $dd01              ; and look if it triggers an
         dex                     ; shift-register interrupt
         bne  -
         lda  $dd0d              ; check for bit3 (SDR-flag)
         and  #8
         beq  inst_err           ; no interface detected

         ;; generate lookup table

         ldx  #0
     -   stx  outstat            ; outstat used as temporary variable
         ldy  #8
     -   asl  outstat
         ror  a
         dey
         bne  -
         sta  revtab,x
         inx
         bpl  --

         ;; enable serial interface (IRQ+NMI)

 enable: sei

         ldx  #<new_irq          ; install new IRQ-handler
         ldy  #>new_irq
         stx  irq_vect
         sty  irq_vect+1

         ldx  #<nmi_startbit     ; install new NMI-handler
         ldy  #>nmi_startbit
         stx  nmi_vect
         sty  nmi_vect+1

         ldx  $2a6               ; PAL or NTSC version ?
         lda  ilotab,x           ; (keyscan interrupt once every 1/64 second)
         sta  $dc06              ; (sorry this will break code, that uses
         lda  ihitab,x           ; the ti$ - variable)
         sta  $dc07              ; start value for timer B (of CIA1)
         txa
         asl  a

         eor  #$33               ; ** time constant for sender **
         ldx  #0                 ; 51 or 55 depending on PAL/NTSC version
         sta  $dc04              ; start value for timerA (of CIA1)
         stx  $dc05              ; (time is around 1/(2*baudrate) )

         asl  a                  ; ** time constant for receiver **
         ora  #1                 ; 103 or 111 depending on PAL/NTSC version
         sta  $dd06              ; start value for timerB (of CIA2)
         stx  $dd07              ; (time is around 1/baudrate )

         lda  #$41               ; start timerA of CIA1, SP1 used as output
         sta  $dc0e              ; generates the sender's bit clock
         lda  #1
         sta  outstat
         sta  $dc0d              ; disable timerA (CIA1) interrupt
         sta  $dc0f              ; start timerB of CIA1 (generates keyscan IRQ)
         lda  #$92               ; stop timerB of CIA2 (enable signal at PB7)
         sta  $dd0f
         lda  #$98
         bit  $dd0d              ; clear pending NMIs
         sta  $dd0d              ; enable NMI (SDR and FLAG) (CIA2)
         lda  #$8a
         sta  $dc0d              ; enable IRQ (timerB and SDR) (CIA1)
         lda  #$ff
         sta  $dd01              ; PB0-7 default to 1
         sta  $dc0c              ; SP1 defaults to 1
         sec
         lda  wr_rptr
         sbc  rd_rptr
         cmp  #200
         bcs  +                  ; don't enable RTS if rec-buffer is full
         lda  #2                 ; enable RTS
         sta  $dd03              ; (the RTS line is the only output)
     +   cli
         rts

         ;; table of timer values for PAL and NTSC version

 ilotab:
         .byte $95
         .byte $25
 ihitab:
         .byte $42
         .byte $40
         ;; disable serial interface
 disable:
         sei
         lda  $dd01              ; disable RTS
         and  #$fd
         sta  $dd01
         lda  #$7f
         sta  $dd0d              ; disable all CIA interrupts
         sta  $dc0d
         lda  #$41               ; quick (and dirty) hack to switch back
         sta  $dc05              ; to the default CIA1 configuration
         lda  #$81
         sta  $dc0d              ; enable timer1 (this is default)

         lda  #<original_irq     ; restore old IRQ-handler
         sta  irq_vect
         lda  #>original_irq
         sta  irq_vect+1

         lda  #<original_nmi     ; restore old NMI-handler
         sta  nmi_vect
         lda  #>original_nmi
         sta  nmi_vect+1
         cli
         rts

         ;; static variables

         stime:          .buf 1  ; copy of $a2=jiffies to detect timeouts
         outstat:        .buf 1

         wr_sptr:        .buf 1  ; write-pointer into send buffer
         rd_sptr:        .buf 1  ; read-pointer into send buffer
         wr_rptr:        .buf 1  ; write-pointer into receive buffer
         rd_rptr:        .buf 1  ; read-pointer into receive buffer

         revtab:         .buf 128
 .newpage
         recbuf:         .buf 256
         sndbuf:         .buf 256

         .global recbuf, sndbuf

 ==============================================================================
 2) c64-binary (uuencoded)
```

```
begin 644 up9600.c64
M`,!(+`W=$`VI$XT/W8T-W:D5C1@#:$!(+`W=$/BIDHT/W8T-W:D`C1@#BDBM
M#-W)@"E_JKT1PFD`K@_"G0##Z(X/PCB*[1#"R<B0"*T!W2G]C0'=:*IH0*T-
MW$I**0+P$*X,PO`+RHX,PM`%"""$P"BP`TR!ZCBEHNT+PLD0D`,@A,!,,>JE
MHHT+PJT!W2E`\"BN#L+L#<+P(+T`Q.B.#L+)@"E_JKT1PFD`2HT,W*D"C0S"
M:@E_C0S<8*X0PNP/PO`;O0##Z(X0PDB*..T/PLG.D`FI`@T!W8T!W1AH8*X-
MPIT`Q.CL#L+P^XX-PJT,PM`'I:))@(T+PF!8.&!XK10#R3'0]:T5`\GJT.ZM
M&`/)1]#GK1D#R?[0X*``C`W"C`["C`_"C!#"6*E_C0W=J8"-`]V-#MV-#]TL
M#=VB"(X!W8T!W<K0]ZT-W2D(\*RB`(X,PJ`(#@S":HC0^9T1PN@0[GBB5J#`
MCA0#C!4#H@"@P(X8`XP9`ZZF`KW6P8T&W+W8P8T'W(H*23.B`(T$W(X%W`H)
M`8T&W8X'W:E!C0[<J0&-#,*-#=R-#]RIDHT/W:F8+`W=C0W=J8J-#=RI_XT!
MW8T,W#BM#\+M$,+)R+`%J0*-`]U88)4E0D!XK0'=*?V-`=VI?XT-W8T-W*E!
>C07<J8&-#=RI,8T4`ZGJC14#J4>-&`.I_HT9`UA@
`
end
```

==

3) how to interface with BASIC V2.0

```
10 fl=fl+1
20 if fl=1 then load"up9600.c64",8,1
30 sys 49404 : rem install up9600 driver
40 if peek(783)and1 then print "can't detect rs232 interface": end

100 sys 49337
110 if peek(783)and1 goto 100 : rem nothing received jet
120 b=peek(780) : rem b holds the received byte

130 poke 780,b:sys 49373 : rem send byte b
140 goto 100
```

you can disable the interface with "sys 49626"
and enable it again with "sys 49505"
(a must, when you want to access your floppy or printer!)