4.1.5 Using the serial port: Example program TMTM.COM

The following section contains the listing for an interrupt driven terminal emulation program. This program shows how the serial port can be used in an applications program. The serial port routines (TMIO.ASM) contain comments showing how the same thing could be performed on an IBM PC. This will allow users familier with the IBM PC to see how to modify existing software.

The program consists of several files:

TM.INC	Equates
TMTM.ASM	Main routine
TMKY.ASM	Keyboard routines
TMDP.ASM	Display routines
TMIO.ASM	Serial port routines

TMIO.ASM will be of most interest to those developing software for the serial port. The other files have been included for completeness. TMTM.ASM should be linked in as the first module create TMTM.COM.

The program will set the serial port to 1200 baud, 8 data bits, 1 stop bits and no parity. The top data bit will be cleared. ALT Q can be used to exit from the program.

```
;***********************************************************;
;       tm                                                  ;
;       Include file for Demo terminal emulator for         ;
;***********************************************************;


; Definitions for accessing 82C50 on serial port
SER_BASE        equ     400h    ; serial base address in ROM
; Offsets from base address of 82C50 control registers
RBR             equ     0h      ; receiver buffer register
THR             equ     0h      ; transmitter holding register
IER             equ     1h      ; interrupt enable register
IIR             equ     2h      ; interrupt identification register
LCR             equ     3h      ; line control register
MCR             equ     4h      ; modem control register
LSR             equ     5h      ; line status register
MSR             equ     6h      ; modem status register
; Interrupt Controller
INT_REG         equ     807fh   ; address of serial vector reg (SIVR)
INT_ON          equ     01h     ; enable interrupt on char in
INT_OFF         equ     00h     ; disable all serial interrupts
INT_NUM         equ     0ch     ; interrupt number for serial port
; Control bytes

DTR             equ     01h     ; bit in MCR for DTR
```

```
        RTS             equ     02h     ; bit in MCR for RTS
        THRE_MASK       equ     20h     ; bit in LSR for transmitter ready

        ; Memory allocation blocks
        BUF_LEN         equ     100h    ; length of serial input buffer
        STK_LEN         equ     200h    ; length of program stack

        ; Miscellaeous definitions
        CR              equ     0dh     ; carriage return character
        LF              equ     0ah     ; line feed character
        PORT_DEFAULT    equ     83h     ; serial port defaults
        STRP_TOP        equ     7fh     ; clear top bit

name        tmtm
;**************************************************************************;
;       tmtm_main                                                          ;
;       Terminal emulator for Pocket PC Serial Port                        ;
;                                                                          ;
;       This terminal emulator is fully interrupt                          ;
;       driven and shows how serial port applications                      ;
;       can be written for the Pocket PC                                   ;
;                                                                          ;
;       This module should appear at the start of                          ;
;       linked objects                                                     ;
;       tmtm_main is the entry point                                       ;
;                                                                          ;
;**************************************************************************;
            extrn       tmio_inon: near
            extrn       tmio_init: near
            extrn       tmky_gtky: near
            extrn       tmio_char: near
            extrn       tmio_intc: near
            extrn       tmio_offc: word
            extrn       tmio_segc: word

include tm.inc
code        segment byte public
            org         100h
code        ends
; pgroup allows the linking of several modules in such a way that the
; total code size can be determined
pgroup      group       code, endseg
            assume      cs:pgroup, ds:pgroup
code        segment byte public
tmtm_main proc near
; Free unused memory to allow applications/hotkeys to work
            mov         ah, 4ah         ; modify memory allocation
            mov         bx, offset pgroup:last_byte + STK_LEN + 0fh
            mov         cx, 4
```

```
; terminal emulator
endseg      segment byte public
last_byte:                               ;end of the program
endseg      ends
            end         tmtm_main


name        tmky
;*****************************************************************
;     tmky
;     Terminal keyboard handler
;
;     This module controls the terminal keyboard
;     Will allow emulator to quit on ALT Q
;
;*****************************************************************
            public`     tmky_gtky
            extrn       tmdp_prbf: near
            extrn       tmio_exit: near
include tm.inc
code        segment byte public
            assume      cs:code, ds:code
;*****************************************************************
;     tmky_gtky
;     terminal keyboard handler
;
;     wait and process key from keyboard
;     returns valid ASCII character in AL
;
;     ALT will call command key
;     ALT Q will leave program
;
;     Parameters:
;           NONE
;     Returns:
;           al          ASCII character code
;     Destroys:
;           NONE
;*****************************************************************
tmky_gtky proc near
gtky_wtky:
            call        tmdp_prb f      ; check and display input buffer
            mov         ah, 1           ; check key status for key stroke
            int         16h             ; ready
            jz          gtky_wtk y      ; wait for a key (no power down!)
            mov         ah, 0           ; key ready so get it
            int         16h             ; from keyboard buffer
            or          al, al          ; extended code?
            jz          gtky_test       ; use extended codes as special
            ret
```

```
          shr       bx, cl          ; divide by 10h; bx has paragraphs
          int       21h             ; do it!
          jnc       tmtm_mmok ; jump if modified ok
; memory modification failed: print message and exit
          mov       ah, 9h          ; display message
          mov       dx, offset tmtm_fail ; failed on allocation
          int       21h
          mov       ax, 4c00h       ; terminate program
          int       21h
; memory modification succeeded: continue starting up
tmtm_mmok:
; set up stack in allocated space
          mov       sp, offset pgroup:last_byte + STK_LEN
; intialise Pocket PC LCD screen using DIP specific services
          mov       ax, 0e01h       ; set external screen mode
          mov       dl, 02          ; to 80*25 tracked
          int       61h
          mov       ax, 1001h       ; set screen position
          mov       dx, 0           ; to top lh corner of display
          int       61h
          mov       ah, 9           ; display start up message
          mov       dx, offset tmtm_strt
          int       21h
; grab interrupt 0ch (COM1 interrupt service routine)
          cli                       ; disable interrupts
          push      bx
          push      es
          mov       ax, 350ch       ; get current int 0ch vector
          int       21h
          mov       tmio_offc, bx ; save offset
          mov       tmio_segc, es ; save segment
          pop       es
          pop       bx
          mov       dx, offset tmio_intc ; Set up our own 0ch service
          mov       ax, 250ch       ; routine as tmio_intc
          int       21h
          sti
          call      tmio_init       ; initialise terminal emulator
          call      tmio_inon       ; enable serial interrupts
; main emulator routine: exit from program is via tmky_gtky
main_next:
          call      tmky_gtky       ; ASCII key in al from keyboard
          call      tmio_char       ; send it to serial port
          jmp       main_next
tmtm_main endp
; Message table
tmtm_fail db 'Failed To Allocate Memory', CR, LF, '$'
tmtm_strt db 'DIP PPC Terminal Emulator Demo Program', CR, LF, '$'
code      ends
; endseg is a dummy segment that will appear at the end of the
```

```
;check for ALT codes
gtky_test:
            cmp         ah, 10h         ; check for ALT Q
            jne         gtky_wtky       ; jump if not ALT Q
            call        tmio_exit       ; prepare to leave terminal emulator
            int         20h             ; leave it
tmky_gtky endp
code        ends
            end


name        tmdp
;*******************************************************;
;     tmdp                                              ;
;     This module handles screen output                 ;
;                                                       ;
;*******************************************************;
            public      tmdp_prbf
            public      tmdp_bptr
include tm.inc
code        segment byte public
            assume      cs:code, ds:code
;*******************************************************;
;     tmdp_prbf                                         ;
;     Display serial input buffer contents              ;
;                                                       ;
;     The interrupt can place additional characters     ;
;     in the buffer, except when the buffer is being    ;
;     modified.                                         ;
;                                                       ;
;     Parameters:                                       ;
;           NONE                                        ;
;     Returns:                                          ;
;           NONE                                        ;
;     Destroys:                                         ;
;           NONE                                        ;
;                                                       ;
;*******************************************************;
tmdp_prbf proc near
            push        ax
            push        dx
            push        si
prbf_next:
; are we at the beginning of the serial input buffer?
            cmp         tmdp_bptr, offset tmdp_cbuf
            jne         prbf_char       ; if not then print contents
            pop         si
            pop         dx
            pop         ax
            ret
```

```
; at least one character needs to be printed
prbf_char:
            mov       di, offset tmdp_cbuf ; start of buffer
            mov       dl, [di]          ; move first character
            mov       ah, 2             ; into AH
            int       21h               ; display character
; shift serial buffer along
; first disable interrupts to prevent new charcters being added while
; buffer is being altered
            cli                         ;disable interrupts
            cld                         ;direction up
            mov       cx, tmdp_bptr ;end of buffer+1
            dec       cx                ;last character of buffer
            sub       cx, offset tmdp_cbuf ;no. bytes to move in CX
            mov       si, offset tmdp_cbuf+1 ;start of string to move
; at this point, es:di points to the start of the buffer and
; ds:si points to one character in.  The buffer will be shifted down one
; character by the use of movsb.
      rep   movsb                       ;[ds:si] --> [es:di] CX times
            dec       tmdp_bptr         ;new end of buffer
            sti                         ;allow interrupts again
; buffer may receive characters again
            jmp       prbf_next         ; loop for next character
tmdp_prbf endp
; Buffer storage
tmdp_cbuf db BUF_LEN dup (00) ;serial input buffer
tmdp_bptr dw offset tmdp_cbuf ;pointer to top input buffer
code        ends
            end


name        tmio
;*********************************************************************
;     tmio
;     This module interfaces with serial port
;
;     The interrupt routine assumes that an interrupt
;     signifies the presence of a serial input
;     character
;
;     No handshaking is performed by the emulator
;     A baud rate of 1200 is assumed
;     8 data bits/no parity is assumed
;     Top data bit is stripped off
;
;*********************************************************************
            public    tmio_char
            public    tmio_init
            public    tmio_inon
            public    tmio_inof
            public      tmio_exit
            public      tmio_intc
            public      tmio_offc
            public      tmio_segc
            extrn       tmdp_bptr: word
include tm.inc
code        segment byte public
            assume      cs:code, ds:code
;*************************************************************
;     tmio_char
;     Sends a character to the serial port
;
;     Parameters:
;         al:         ASCII character to send
;     Returns:
;         NONE
;     Destroys:
;         NONE
;*************************************************************
tmio_char proc near
            push        dx
            push        di
            push        ax
            mov         di, tmio_base ; get base address of COM1 82C50
            mov         dx, LSR       ; line status register
            add         dx, di
char_wthr:
            in          al, dx        ; wait for transmitter ready
            test        al, THRE_MASK
            jz          char_wthr     ; loop if not ready
            pop         ax
            mov         dx, THR       ; address of transmitter holding
            add         dx, di        ; register
            out         dx, al        ; send character to serial
            pop         di
            pop         dx
            ret
tmio_char endp
;*************************************************************
;     tmio_init
;     performsl initialisation of serial port
;
;     Port is intialised to 1200 baud, 8 bits,
;     no parity.
;     DTR is set high:  I'm always ready
;     Interrupt register on port setup as INT_REG
;
;     Parameters:
;         NONE
;     Returns:
```

```
;           NONE                                                ;
;       Destroys:                                               ;
;           NONE                                                ;
;***************************************************************;
tmio_init proc near
            push        ax
            push        si
            push        dx
            push        di

            xor         ax, ax
            push        ds
            mov         ds, ax          ; segment zero
            mov         di, ds:[SER_BASE] ; get base of com1
            pop         ds              ; restore ds to local
            mov         tmio_base, di   ; save base address
            call        tmio_inof       ; disable serial interrupts
            mov         al, PORT_DEFAULT ; set up port as in header
            call        tmio_inpt       ; set up 80c50
; Set up interrupts for the serial port
; On an IBM PC the following code could be used
;
;           in          al, 21h         ; access 82C59 PIC
;           and         al, 0efh        ; enable int 0ch
;           out         21h, al
;
; This will not work on the Pocket PC, but the following code can be used
            mov         ax, INT_NUM     ; interrupt number
            call        tmio_sint       ; set up serial interrupt
; set up modem control register
            mov         dx, MCR         ; Tell the world we are ready
            add         dx, di
            mov         al, DTR or RTS ; set RTS/DTR
; On an IBM PC the interrupt line needs to be enabled:
;
;           mov         al, DTR or RTS or 8
;
            out         dx, al          ; set up modem control register
            call        tmio_inon       ; enable serial interrupts
            mov         dx, di          ; clear input buffer on 82C50
            in          al, dx
            pop         di
            pop         dx
            pop         si
            pop         ax
            ret
tmio_init endp
;***************************************************************;
;       tmio_inon                                               ;
;       Enables serial interrupts                               ;
```

```
;
;       Parameters:
;           NONE
;       Returns:
;           NONE
;       Destroys:
;           al, dx
;
;*************************************************************;
tmio_inon proc near

        mov     dx, IER         ; interrupt enable register
        add     dx, cs:tmio_base
        mov     al, INT_ON      ; interrupt enabled
        out     dx, al
        ret
tmio_inon endp
;*************************************************************;
;       tmio_inof
;       Disable serial interrupts
;
;       Parameters:
;           NONE
;       Returns:
;           NONE
;       Destroys:
;           al, dx
;
;*************************************************************;
tmio_inof proc near

        mov     dx, IER         ; interrupt enable register
        add     dx, cs:tmio_base
        mov     al, INT_OFF     ; disable interrupts
        out     dx, al
        ret
tmio_inof endp
;*************************************************************;
;       tmio_intc
;       Serial read interrupt service
;
;       Invoked by serial input register being full
;       Places character in buffer and returns
;
;       Parameters:
;           NONE
;       Returns:
;           NONE
;       Destroys:
;           NONE
```

```
;**************************************************************;
tmio_intc proc near
          push      ax
          push      dx
          push      di
          mov       dx, RBR       ; address of receiver buffer
          add       dx, cs:tmio_base
          in        al, dx        ; get received character into al
          and       al, STRP_TOP  ; strip top bit
          mov       di, cs:tmdp_bptr ; place character at top
          mov       cs:[di], al   ; of buffer
          inc       cs:tmdp_bptr  ; advance buffer pointer
; On an IBM PC the interrupt must be acknowledged by the following code:
;
;         mov       al, 20h
;         out       20h, al
;
; On the Pocket PC this is unnecessary
          pop       di
          pop       dx
          pop       ax
          iret
tmio_intc endp
;*************************************************************;
;    tmio_exit
;    Ensures safe exit from terminal emulator
;
;    Parameters:
;         NONE
;    Returns:
;         NONE
;    Destroys:
;         NONE
;*************************************************************;
tmio_exit proc near
          push      ax
          push      bx
          push      dx
          call      tmio_inof     ; Disable interrupts
; put old interrupt service routine back
          push      ds
          mov       ds, tmio_segc ; get old segment
          mov       dx, tmio_offc ; get old offset
          mov       ax, 250ch
          int       21h           ; redirect serial interrupt
          pop       ds
          mov       al, 48h       ; reset default interrupt vector
          call      tmio_sint
          pop       dx
          pop       bx
```

```
            pop         ax
            ret
tmio_exit endp
;************************************************************
;     tmio_sint
;     Set interrupt vector register
;
;     Will replace existing entry if possible
;     This routine uses int 61h service 1ch to ensure
;     that power down will not corrupt serial port
;     vector register
;
;     Parameters:
;           al: interrupt number
;     Returns:
;           NONE
;     Destroys:
;           NONE
;************************************************************
tmio_sint proc near
            push        ax
            push        bx
            push        cx
            push        dx
; check for vector already being set up
            push        ax
            mov         cl, 3           ; first non-reserved entry
sint_srch:
            inc         cl
            cmp         cl, 11          ; max table entry+1
            je          sint_seti       ; if got here then entry no exist
            mov         ax, 1c01h       ; return table entry
            mov         bh, cl          ; table entry number
            int         61h             ; return table entry
; check if SIVR has been set up before
            cmp         dx, INT_RE G    ; have we found location in table ?
            jne         sint_srch       ; no than always replace
; have found location in table for interrupt vector number
sint_wral:
            pop         ax              ; interrupt number back
            mov         bl, al          ; put value to write into bl
            mov         bh, cl          ; table entry to use
            mov         dx, INT_REG     ; address of SIVR
            mov         ax, 1c00h       ; write entry number
            int         61h
            jmp         sint_exit
; find an empty entry table to use
sint_seti:                              ; find empty table entry
            mov         cl, 3           ; first entry to check
sint_sr00:
        inc     cl
        cmp     cl, 11          ; max table entry+1
        je      sint_bodg       ; if got here then entry no exist
        mov     ax, 1c01h       ; return table entry
        mov     bh, cl          ; table entry number
        int     61h             ; return table entry
        cmp     dx, 0           ; have we found empty location in table ?
        jne     sint_sr00       ; no than always replace
        jmp     sint_wral       ; yes go and write it
sint_bodg:                      ; no table entry
; no table entry has been found to do it the bad way
        pop     ax
        mov     dx, INT_REG     ; corruption of SIVR may occur
        out     dx, al          ; on power down
sint_exit:
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
tmio_sint endp
;*************************************************************************
;
;     tmio_inpt
;     Initialise 80c50 (based on int 14h service 0)
;
;     Parameters:
;           al: port parameters (as int 14h)
;                   Bits 7, 6, 5 BAUD RATE
;                        0 0  0   110
;                        0 0  1   150
;                        0 1  0   300
;                        0 1  1   600
;                        1 0  0   1200
;                        1 0  1   2400
;                        1 1  0   4800
;                        1 1  1   9600
;
;                   Bits 4, 3 PARITY
;                        x 0      none
;                        0 1      odd
;                        1 1      even
;
;                   Bit   2       STOP BITS
;                         0       1 bit
;                         1       2 bits
;
;                   Bits 1, 0 WORD LENGTH
;                        1 0      7 bits
;                        1 1      8 bits
;     Returns:
```

```
;           NONE                                                        ;
;       Destroys:                                                       ;
;           NONE                                                        ;
;***********************************************************************;
tmio_inpt proc near
            push        ax              ; Preserve parameters
            mov         cl, 5           ; Set up shift count
            shr         al, cl          ; Get bits to shift
            jz          init_spec       ; Special case of 110 baud
            mov         cl, al          ; Get count in CL
            mov         ch, 06h
            shr         cx, cl          ; Get divisor in CX
            jmp         short init_norm
init_spec:
            mov         cx, 417h        ; Divisor for 110 baud
init_norm:
            mov         dx, tmio_base ; Base address
            add         dx, LCR         ; Get line control reg port
            mov         al, 80h         ; Access divisor regs
            out         dx, al
            mov         dx, tmio_base ; Lower divisor latch
            mov         al, cl          ; Get low divisor
            out         dx, al          ; Write divisor
            inc         dx              ; Upper divisor latch
            mov         al, ch          ; Get high divisor
            out         dx, al          ; Write divisor
            pop         ax              ; Restore parameters
            and         al, 1fh         ; Get bits 4 to 0
            mov         dx, tmio_base ; Base address
            add         dx, LCR         ; Line control register port
            out         dx, al          ; Write data
            ret
tmio_inpt endp
tmio_base dw 0              ; base address
tmio_offc dw 0              ; offset of old int 0ch
tmio_segc dw 0              ; segment of old int 0ch
code        ends
            end
```

RUN FILES GREATER THAN 64K

In order to build a .RUN file with a code size greater than 64k, it is necessary to have more than one code segment. One way of achieving this is to build the program using the MEDIUM memory model. In this way the code size is only limited to the available space on a CCM (up to 128k).

Unlike an .EXE file, which has fixups resolved at run time, a .RUN file must have the fixups resolved before the program is commited to a ROM card. Therefore it is necessary to resolve the fixups based upon an absolute memory address for the file, and it must be known in advance where the file will reside on the card. If the program is the first file on the card, its position can be calculated as follows :-

Fixup Address (in paragraphs) =

C000H + (Boot sectors + FAT sectors + Root Dir sectors)* (sector size in paragraphs)

The number of sectors used can be found by using a disk utility program (such as Norton Utilities).

Example :-

For a 128k card formatted with 512 bytes per sector, 1 sector for the Boot Record, 1 sector for the FAT, and 8 sectors for the Root Dir, the address (in paragraphs) of the first file on the card will be C140H.

This value should then be used for the fixup segment address, before the program is copied to the ROM card.

Due to the mechanism used by the operating system to execute .RUN files, the file must have an apparent size less than 64k. Therefore after the program has been copied to the card, the file size entry in the Root Dir must be set to a value less than 64k.

Since data fixups must be resolved at run time, it is not possible to have more than 64k of data. This means that the HUGE memory model cannot be used.

Atari Corp., Sunnyvale, CA 94086
© Atari Corp.
Alle Rechte vorbehalten.

Atari Computer GmbH
Frankfurter Str. 89-91
6096 Raunheim