Emulation of a Stripchart Recorder with the Split Screen and Smooth Scrolling Capabilities of the VGA

David MacGibbon

Transonic Systems, Inc.
34 Dutch Mill Road
Ithaca, NY 14580

Introduction

We produce a medical flowmeter equipped with a RS232 computer interface. The interface board was completely described in last year's proceedings **[Mac90]**, but it's basic function is to digitize a medium speed analog signal (100 samples per second) and to transmit the data to the computer for storage and display. We soon found that even our moderate data acquisition rate was enough to severely burden conventional computer display techniques. We tried a plot and clear screen type of display, but users complained that it was hard to follow and that important events at the edge of the screen were immediately erased. We also explored the technique in which an erasing cursor moves across the screen just in front of the drawing cursor. Our company critics felt that this was an improvement but that viewing this display was still fatiguing because of the constant eye movement required. At this point, our chief critic challenged us to produce a smooth scrolling stripchart type display without any hardware. It was also requested that half of the screen be available for use as a help menu.

After a couple of weeks of vehemently insisting that these requirements could not be met with software alone I discovered an excellent article discussing the smooth scrolling capabilities of the EGA and VGA **[Wil88]**. The article also describes the screen splitting potential of the VGA. With this additional information, we were able to build a software package that converts any VGA equipped personal computer into a smooth scrolling split-screen stripchart emulator.

The source code listed was written for Tom Zimmer's FPC3 **[Zim90]** dialect of FORTH and has several components that are specific for a VGA equipped IBM clone. The program described will provide for stripchart playback of a recorded disk file at 100 samples per second. It can easily be adapted for multichannel, multicolor operation and for data streams from sources other than disk.

The Timer

The first item necessary to make an accurate stripchart emulator is a precision timer. If we don't have an external hardware timer, then we must do the best we can with components already in the IBM PC. The first segment of the source code describes the reprogramming of the IBM clone's 8253 counter-timer chip to generate a timer tick at 100 samples per second. Please note that the method used does disable the normal DOS timer functions. You may wish to refer to references **[Dun86]** and **[Dun88]** for more sophisticated ways to accomplish this, chaining to the normal DOS timer interrupt vector on every 5th time tick would provide approximately correct DOS timing functions.

Drawing on a moving target

We are going to rearrange the way the screen maps to the video memory slightly. Figure 1 shows the normal relationship of screen to memory. We are going to change this relationship to that shown in figure 2, where the screen is split into two window that are mapped separately. Note that the video memory from 6000H to AFFFH isn't initially in use. To avoid rewriting the entire video map when we display the next data point, we will put the new data in the unused memory and readjust the graphics controller registers that determine which memory is displayed in the top window. The net effect of this approach is that the memory map for the top window slides down as the data pans across the screen. This is shown in figure 3, where the beginning corner of the top window memory map slides down from AFFFH to 6000H.

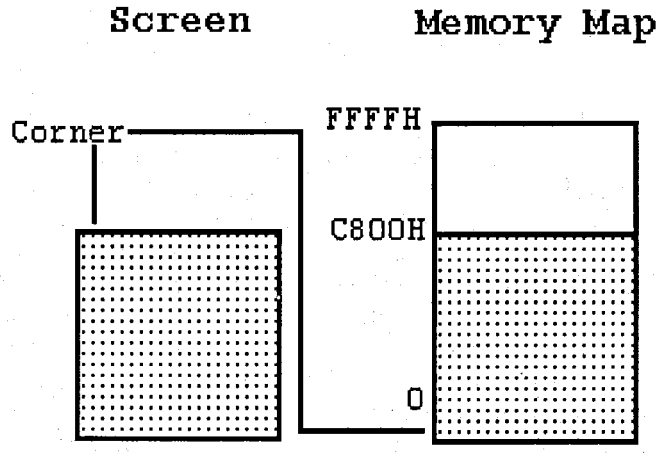

Figure 1.

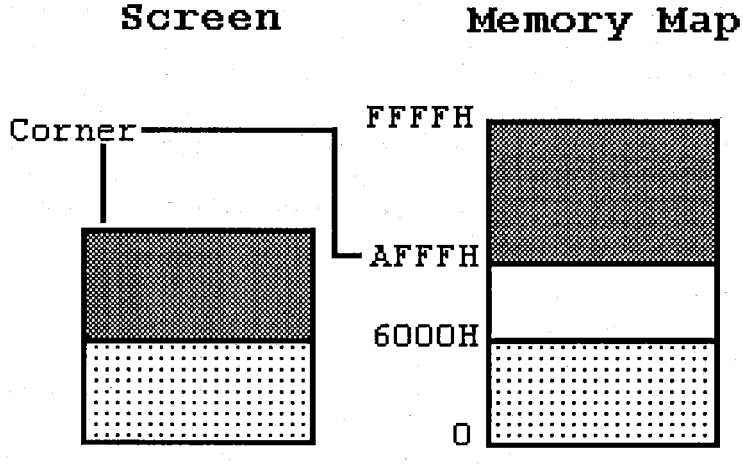

Figure 2.

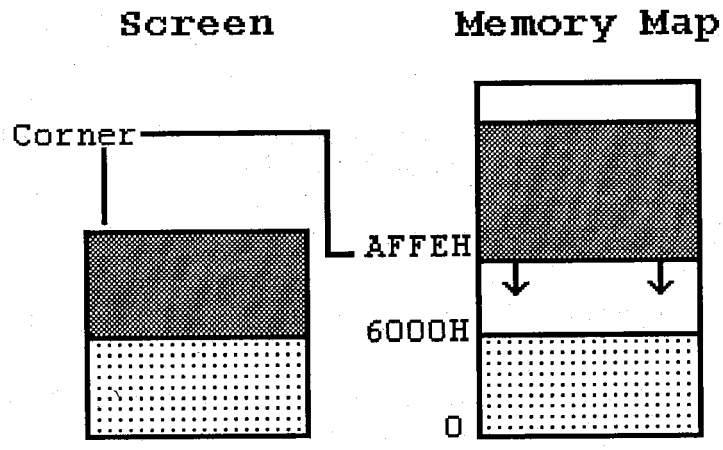

Figure 3.

Unfortunately, the graphics BIOS routines in ROM were not designed to function properly on a moving target. The next segment of source code contains our replacement for the BIOS pixel writing routine and words to draw a line. Our first requirement is for a routine to set the appropriate pixels of the moving memory map. This is accomplished by the routine `(VSET)`, an adaptation of routines described in reference **[Wil87]**. Once pixel setting worked properly, the line drawing routines **[Smi90]** provided with FPC were easily adapted to draw lines on the moving screen.

Panning the screen

The third section of source code contains words to manipulate the graphics registers. Initially, we must set the crt controller registers so that the screen is split into two separate windows with the word `SPLITLINE!`. Next we disable scrolling on the bottom menu window with the word `ENABLEHALFSCROLL`.

Once this is accomplished we can reposition the top window anywhere in the memory map with the words `CORNER_BIT!` and `CORNER_BYTE!`. `CORNER_BYTE!` sets the memory map byte that corresponds to the top left corner of the screen. For precision placement the word `CORNER_BIT!` may be used to set the bit within the chosen corner byte. The word `SMOOTH.SCROLL` uses these two words together to move the top window of the screen across the memory map a pixel at a time. This section of code also contains words to synchronize register changes with the horizontal and vertical retrace intervals. I must admit that my choice of the synchronization methods used in `SMOOTH.SCROLL` doesn't really agree with my understanding of how the hardware works.

Finally, since we will eventually reach the end of the memory map, we need a word `(MOVE.MAPS)` to periodically copy the current memory image to the beginning of the memory map.

Point and play

The fourth and final section of source code is a simplified version of our actual application. The word `!GRAPH` adds a data point to the left side of the scrolling window. This words includes factors to change the display's offset and gain. To invoke the stripchart scroller, execute the word

PLAY, place the cursor on the ramp function data file "TEST.DAT" created by **FAKE_DATA** and let it scroll.

References

[Mac90] MacGibbon,D.R., *A Three Chip Design for Analog to RS232 Data Acquisition* Proceedings of the 1990 Rochester Forth Conference, pp. 99-100, 1990.

[Dun86] Duncan, R.,*Advanced MS DOS*, pp. 208-220. MicroSoft Press, 1986.

[Dun88] Duncan, R., *The MS DOS Encyclopedia* pp. 425-426. Microsoft Press, 1988.

[Mac90] MacGibbon, D.R.,*A Three Chip Design for Analog to RS232 Data Acquisition*, Proceedings of the 1990 Rochester Forth Conference, pp99-100, 1990

[Smi90] Smiley, M. *FPC user routine VGA.SEQ*, FORTH INTEREST GROUP, 1990., San Jose

[Wil87] Wilton, R. *Programmer's Guide to PC & PS/2 Video Systems*, pp. 96,144-145. Microsoft Press 1987.

[Wil88] Wilton,R., *Pixel panning and split screens* PC Tech Journal, volume 6, issue 11, Nov, 1988, p62(8)

[Zim90] Zimmer, T. *FPC*, FORTH INTEREST GROUP, 1990., San Jose

Appendix

```
11930     CONSTANT 100HZ
8         CONSTANT TIMER-INT       \ IRQ 8
2         CONSTANT NUMBYTES \ bytes per packet
VARIABLE  TIMERCOUNT
2VARIABLE PREV_TIMER_VEC     \ old int vector
CREATE    INPUTBUFFER NUMBYTES ALLOT
HANDLE    DATAFILE         \ file handle

CODE INT_VECTOR! ( seg addr int_# -- )
\ store int vector
      POP AX        POP DX          POP BX
      PUSH DS       MOV DS, BX      MOV AH, # $25
      INT $21       POP DS          NEXT END-CODE

CODE INT_VECTOR@ ( int_# -- seg addr )
  \ fetch int vector
      POP AX        PUSH ES         MOV AH, # $35
      INT $21       MOV DX, ES      POP ES
      PUSH DX       PUSH BX         NEXT END-CODE

: TIMERSTART       ( count -- )       \ start timer
      $34 $43 PC!    DUP $40 PC!    FLIP $40 PC! ;

CODE TIMER_
RTN              \ timer int routine
      PUSH AX        PUSH DX
      CS: INC TIMERCOUNT WORD \ increments variable
      MOV DX, # $20   MOV AL, # $20   OUT DX,  AL
      POP DX         POP AX         STI IRET
END-CODE

: TIMER-ON  ( -- )              \ reset timer and
  PREV_TIMER_VEC 2@ OR 0= IF \ initialize int vec.
     TIMER-INT INT_VECTOR@  \ save previous vector
     PREV_TIMER_VEC 2!
  THEN
  ?CS:  ['] TIMER_RTN  TIMER-INT  INT_VECTOR!
  100HZ TIMERSTART  TIMERCOUNT OFF;
                    \ interrupt @ 100 hz

: TIMER-OFF  ( -- )         \ restore to DOS state
  PREV_TIMER_VEC 2@ OR IF
    PREV_TIMER_VEC 2@  TIMER-INT  INT_VECTOR!
    0. PREV_TIMER_VEC 2!  $FFFE TIMERSTART
  THEN ;                \ old DOS timer rate
                        \ about 18 hz.

: ?TIMER  ( -- data flg )   \ returns data from file
   TIMERCOUNT @ 1 U> IF    \ if time for it
     INPUTBUFFER NUMBYTES DATAFILE HREAD IF
       TIMERCOUNT DECR   INPUTBUFFER @   TRUE
     ELSE
       TRUE FALSE          \ late enough, but no
     THEN                  \ more data
   ELSE
      FALSE FALSE          \ too early not time yet
   THEN ;
```

```
\ Adapted from routines distributed or published
\ by M. Smiley, E.T. Smith and R. Wilton
DECIMAL PREFIX

CODE MODE  ( N -- )  \ set video mode
  POP AX      INT $10    NEXT  END-CODE

: TEXT ( -- )  \ return to normal fast text mode
   3 MODE DARK  FAST  STATON ;

VARIABLE COLOR  11 COLOR !
VARIABLE X1'    VARIABLE X1
VARIABLE X2'    VARIABLE Y1
VARIABLE Y1'    VARIABLE X2
VARIABLE Y2'    VARIABLE Y2
CREATE LTBL     8 ALLOT
80 CONSTANT     BYTESPERROW
VARIABLE       CORNER_BYTE
$A000 CONSTANT  GSEG

LABEL  (VSET)  \ subroutine to set pixel on scrolling screen
   PUSH DS              PUSH AX
   PUSH BX              PUSH CX
   PUSH DX              MOV AX, DX
   MOV BX, CX            MOV CL, BL
   \ compute byte address of pixel  AX = #rows * bytes per row
   MOV DX, # BYTESPERROW  MUL DX
   SHR BX              SHR BX   \ BX = # cols * 8
   SHR BX              ADD BX, AX
   ADD BX, CORNER_BYTE     IN  \ add pos for corner
   MOV DX, # GSEG          MOV DS, DX
   \ compute bit mask of pixel
   AND CL, # 7           XOR CL, # 7
   MOV AH, # 1           SHL AH, CL
   \ select bit mask register of controller
   MOV DX, # $3CE         MOV AL, # 8
   OUT DX, AX
   MOV AL, 0 [BX]         MOV 0 [BX], # 0 BYTE
   MOV DX, # $3C4
   \ select map mask register of sequencer to select the color
   CS: MOV AH, COLOR       MOV AL, # 2
   OUT DX, AX            MOV 0 [BX], # $FF BYTE
   \ restore default map mask and default bit mask
   MOV AH, # $0F          OUT DX, AX
   MOV DX, # $3CE         MOV AX, # $FF08
   OUT DX, AX            POP DX
   POP CX               POP BX
   POP AX               POP DS RET END-CODE

LABEL STEEP(V)
   MOV AX, LTBL 2 +       SHR AX
   MOV LTBL 4 + AX        MOV CX,  X1'
   MOV DX, Y1'           MOV BX, # 0
   MOV AX, LTBL 2 +       MOV LTBL 6 + AX
   BEGIN
      CALL (VSET)
      ADD DX, SI         ADD BX, LTBL
```

```
      CMP BX, LTBL 4 + > IF
          SUB BX, LTBL 2 +    ADD  CX, DI
      THEN
  DEC  LTBL 6 + WORD < UNTIL
  RET END-CODE

LABEL EASY(V)
  MOV  AX, LTBL          SHR AX
  MOV LTBL 4 +  AX       MOV CX, X1'
  MOV DX, Y1'            MOV BX, # 0
  MOV AX, LTBL           MOV LTBL 6 + AX
  BEGIN
      CALL (VSET)
      ADD CX, DI        ADD BX, LTBL 2 +
      CMP BX, LTBL 4 + > IF
        SUB BX, LTBL    ADD DX, SI
     THEN
  DEC LTBL 6 + WORD < UNTIL
  RET END-CODE

LABEL  STORE-X(V)
  MOV LTBL AX
  CMP AX, LTBL 2 +       JL STEEP(V)
  CALL EASY(V)           RET END-CODE

LABEL STORE-Y(V)
  MOV LTBL 2 +  AX       MOV AX, X2'
  SUB AX, X1'            MOV DI, # 1
  JGE STORE-X(V)
  MOV DI, # -1          NEG AX
  JMP STORE-X(V)          END-CODE

LABEL ((VLINE))
  MOV AX, X1            MOV X1' AX
  MOV AX, Y1            MOV Y1' AX
  MOV AX, X2            MOV X2' AX
  MOV AX, Y2            MOV Y2' AX
  SUB AX, Y1'
  MOV SI, # 1           JGE STORE-Y(V)
  MOV SI, # -1          NEG AX
  JMP STORE-Y(V)        END-CODE

CODE (VLINE) ( -- )
  PUSH SI              PUSH BP
  CALL ((VLINE))          POP BP
  POP SI              NEXT END-CODE

\ draw line segment between 2 points
: VLINE  ( x1 y1 x2 y2 -- )
  Y2 ! X2 ! Y1 ! X1 !  (VLINE) ;

\ draw line segment from last pt. to this pt.
: VDRAW ( x1 y1 -- )
  2DUP Y1 ! X1 !  (VLINE)  Y2 ! X2 ! ;

$afff CONSTANT  FIRST.FRAME
$6000 CONSTANT  LAST.FRAME
$5000 CONSTANT  SCROLL.MEM.LENGTH
VARIABLE      CORNER_BIT
```

```
: CRTC!   ( value register -- )
     $3D4 PC!      $3D5 PC! ;

: CRTC@   ( register  -- value )
     $3D4 PC!      $3D5 PC@ ;

: SPLITLINE! ( line -- ) \ line # ranges from 0 to 479
   2/ 2*                  \ force line # even
   DUP $FF AND $18 CRTC!
   DUP $100 AND $10 /
   7 CRTC@ $10 $FF XOR AND OR
   7 CRTC! $200 AND 8 /
   9 CRTC@ $40 $FF XOR AND OR
   9 CRTC! ;

CODE  CORNER_BYTE! ( n -- )
    \ n ranges from 6000H to AFFFH
   CLI                    MOV DX, # $3d4
   MOV AX, # $0d          OUT DX, AL
   INC DX                 POP AX
   MOV CORNER_BYTE AX     OUT DX, AL
   DEC DX                MOV Al, # $0c
   OUT DX, AL            INC DX
   MOV AL, AH            OUT DX, AL
  STI
NEXT               END-CODE

CODE  CORNER_BIT!  ( N -- )  \ n ranges from 0 to 7
   CLI                     MOV AX, # $33
   MOV DX, # $3C0          OUT DX, AL
   POP AX                  OUT DX, AL
   STI                     MOV CORNER_BIT AX
   NEXT                    END-CODE

CODE WAITFORH  ( -- )    \ wait for horizontal synch
   BEGIN
        MOV DX, # $3DA   IN AL, DX
   AND AL, # 1 0 UNTIL
   NEXT END-CODE

CODE WAITFORNOV  ( -- )  \ wait for not vertical synch
   BEGIN
        MOV DX, # $3DA   IN AL, DX
   TEST AL, # 8  0=   UNTIL
   CLI
   NEXT END-CODE

CODE WAITFORV  ( -- )       \ wait for vertical synch
   BEGIN
        MOV DX, # $3DA   IN AL, DX
   TEST AL, # 8  0   UNTIL
   NEXT END-CODE

\ allow scrolling on half screen only
CODE ENABLEHALFSCROLL  (  -- )
   MOV AX, # $1007 MOV BL, # $10
    \ read paette reg 16
   INT $10       OR  BH, # $20
   MOV AX, # $1000  MOV BL, # $10
    \ set bit 5 of palette
```

```
   INT $10        NEXT END-CODE   \ register 16

: INITSCREEN  ( -- )
   STATOFF  SLOW  18 MODE
   CORNER_BYTE OFF
   0 1 631 1 VLINE  7 X2 !  255 Y2 !
   255 SPLITLINE!  ENABLEHALFSCROLL
   FIRST.FRAME CORNER_BYTE!  7 CORNER_BIT! ;

: MOVE.MAP ( -- )
   GSEGLAST.FRAME GSEG FIRST.FRAME SCROLL.MEM.LENGTH
   CMOVEL ;

: MOVE.MAPS  ( -- )
   $105 $3CE P!      \ set mode to read X write Y
   $802 $3C4 P! MOVE.MAP
      \ move 4 bitplanes for 16 colors
   $402 $3C4 P! MOVE.MAP
   $202 $3C4 P! MOVE.MAP
   $102 $3C4 P! MOVE.MAP
   $005 $3CE P!  \ restore mode to read Z write B

: SMOOTH.SCROLL ( -- )
  CORNER_BIT @ 0 = IF
      CORNER_BYTE @ LAST.FRAME = IF
      MOVE.MAPS  FIRST.FRAME CORNER_BYTE!
      THEN
  256 0 DO
       0 GSEG CORNER_BYTE @ BYTESPERROW I * 1- + C!L
     LOOP
      CORNER_BYTE @ 1-  WAITFORNOV  CORNER_BYTE!
      WAITFORV  7 CORNER_BIT!
    ELSE
     CORNER_BIT  @ 1-  WAITFORH  CORNER_BIT!
   THEN ;
\ stripchart smooth scrolling demonstration
FLOAD TIMERASC  FLOAD VLINE    FLOAD SCROLL
VARIABLE OFF-SET            VARIABLE MULT  1 MULT !
VARIABLE DIV  4 DIV !        VARIABLE FINISHED

: !GRAPH   ( n -- )   \ add data pt to the scrolling graph
    OFF-SET @ +  MULT @ DIV @ */    \ adjust gain & offset
    255 MIN     1 MAX                \ clip to fit on screen
    224 +      479 SWAP -            \ adjust to screen coords
    CORNER_BIT @ SWAP VDRAW          \ draw new segment
    CORNER_BIT @ 0 = if              \ if last bitwise scroll
     8 X2 !                         \ adjust last pt for
    THEN                            \ byte scroll
    SMOOTH.SCROLL ;

: OPENSTOREDFILE ( -- flg )  \ select file with Tom Zimmers
    GETFILE 0= IF
        FALSE EXIT               \ point and pick utility
    THEN
    DATAFILE $HANDLE
    DATAFILE HOPEN 0= NOT IF
        ." FILE DOESN'T EXIST " KEY DROP   FALSE EXIT
    THEN
    TRUE ;
```

```
: PLAY ( -- ) \ play back a recorded data file
     OPENSTOREDFILE 0= IF  EXIT  THEN
     INITSCREEN
     20 2 AT ." THIS HALF OF SCREEN IS FOR THE MENU "
     FINISHED OFF  TIMER-ON  TIMERCOUNT OFF
     BEGIN
        ?TIMER IF
           !GRAPH
        ELSE
           IF
              FINISHED ON
           THEN
        THEN
        KEY? IF
             KEY CASE
              $1B ( ESC )  OF  FINISHED ON  ENDOF
             DROP
             ENDCASE
           THEN
     FINISHED @ UNTIL
     TEXT  TIMER-OFF  DATAFILE HCLOSE DROP ;

: FAKE_DATA    \ create a data file with a ramp function
     " TEST.DAT" "$ DATAFILE $HANDLE
     DATAFILE HCREATE IF  ." FILE CREATION ERROR "  THEN
     5 0 DO 500 0 DO
        I INPUTBUFFER !  INPUTBUFFER 2 DATAFILE HWRITE DROP
     LOOP LOOP
     DATAFILE HCLOSE DROP ;

FAKE_DATA     \ make a fake data file
```