Inter-Office Memorandum

To	Notetaker Working Group	Date	April 19, 1978
From	Larry Tesler	Location	Palo Alto
Subject	Distributed BitBlt	Organization	SSL

XEROX

Filed on: <Tesler>dBitBlt.press **For Xerox Internal Use Only**

Problem

There has been a question of whether the Notetaker I should do its BitBlts in the emulation processor or in an I/O processor. If it is done in an I/O processor, then there can be some overlap between emulation and character generation. If it is done in the emulation processor, then it will be compatible with the Notetaker II, which can do faster BitBlts in custom LSI than Notetaker I could do on an 8086.

Alternatives

It is possible to involve two processors in the BitBlt, one fetching the source and one storing into the destination rectangle. The two processors would communicate over three extra lines on the motherboard. It would not be necessary to gain access to the system bus to use these lines. The three lines would be: one serial data line; two handshake lines to control the flow. (Maybe these lines could be multiplexed at a cost of logic at both ends, using phase-encoding or other tricks.)

If two I/O processors were employed, the emulation processor would be free during the BitBlt. However, this would require the presence of both I/O boards at all times, or two 8086's on the same board with separate access to the system bus, requirements we don't otherwise have. An alternative is to use the emulation processor and only one I/O processor.

There is still a question of which processor should be source and which should be destination. Since both the source and the destination processors must worry about the cursor, the situation seems entirely symmetrical in Notetaker I. However, in Notetaker II, the emulation processor will be faster. Therefore, it should have the larger share of the work, which is the destination side.

The emulation processor would start a BitBlt by setting up a control block in main memory and signalling the I/O processor by the interrupt mechanism. It would then start its own store loop going, being sure to wait for each word of the source stream to be available before doing the logical operation and the store. Hardware and software details of each side are presented below. A block diagram of the hardware is appended.

The Source

The source processor would output 16-bit words into a 16-bit bidirectional shift register (e.g., two 8-bit MSI TTL DM54198's, which shift at 35 MHz and dissipate 360 mW apiece, or something better). Associated with the shifter would be a four-bit counter, a one-bit direction register, and a decoder that maps this hardware into the 8086 address space.

The source rectangle would be serialized a horizontal line at a time. For example, to send a line of a rectangle in left-to-right order whose x was 2 and whose width was 40, the source processor would:

(1) Set the shifter mode to Left Shifts;
(2) Set the counter to 14;
(3) Load the first word (x = 0 to 15) shifted left 2 into the shifter;
(4) Wait until the shifter is empty (the counter is zero);
(5) Load the next word (x = 16 to 31) into the shifter;
(6) Wait until the shifter is empty (the counter is zero);
(7) Set the counter to 10;
(8) Load the last word (x = 32 to 47) into the shifter;
(9) Wait until the shifter is empty (the counter is zero).

Then, to provide zero padding needed by the destination processor, it would:

(10) Set the counter to the padding amount from the control block;
(11) Load zero into the shifter;
(12) Wait until the shifter is empty (the counter is zero).

When bits must be supplied right-to-left, the shifter mode would be Right Shifts. Note that the counter automatically resets itself to 16 (0) each time.

To accomplish the above-listed steps, the source processor would output shifter data and control words into memory-mapped locations. The control word would have a 4-bit count and a 1-bit shift-direction. The act of storing a data word would set the shifter's mode from the shift-direction register and would start the counting and shifting.

The inner loop of a scan-line (steps 4-5 above) would thus be:

```
        RPT MOVW                    ; (DI)←(SI), step, loop
```

The loop would be preceded by a special case for left of line and by initialization of CX (count), SI (source address), DI (shifter address), DF (inc/dec flag). It would be followed by a special case for right of line. The loop assumes that the shifter has been assigned at least 1024 consecutive memory addresses, since DI is incremented each time.

Execution time per word = (6+10) clocks or 2 microseconds, plus .8 microseconds waiting for main memory, plus wait states added when the destination processor falls behind. Note that interrupts can not happen during the wait states, so we have to guarantee that wait states don't last too long (say, 12 us).

The source processor can pump out the full bit map (19,200 words) in about 55 ms, less than 2 frame times. However, on Notetaker I, BitBlt is destination-bound, so this figure is irrelevant. What is relevant is that up to .35 of the main memory bandwidth is consumed.

For constant sources, STOW is used instead of MOVW. The speed is just 2 us per word because no main memory bandwidth is consumed.

The Destination

The destination processor would input 16-bit words from a 16-bit bidirectional shift register with counter etc. as above.

The source rectangle would be deserialized a horizontal line at a time. For example, to receive a line of a rectangle in left-to-right order whose width was 40 and whose destination x will be 7, the destination processor would:

(1) Set the shifter mode to Left Shifts and clear the shifter;
(2) Set the counter to 9 and initiate shift-in;
(3) Wait until the shifter is full (the counter is zero);
(4) Load the first snip (x = 7 to 15) from the shifter;
(5) [Load the destination word, and/or/xor with snip,] Store the result;
(6) Wait until the shifter is full (the counter is zero);
(7) Load the next snip (x = 16 to 31) from the shifter;
(8) [Load the destination word, and/or/xor with snip,] Store the result;
(9) Wait until the shifter is full (the counter is zero).
(10) Load the last snip (x = 32 to 46 plus padding) from the shifter;
(11) [Load the destination word, and/or/xor with snip,] Store the result.

When bits must be supplied right-to-left, the shifter mode would be Right Shifts. Note that the counter automatically resets itself to 16 (0) each time.

To accomplish the above-listed steps, the destination processor would input shifter data and output control words into memory-mapped locations as above. Shifting would be initiated automatically after each snip is loaded.

The inner loop of a scan-line (steps 6-8 above) would thus be:

```
L:  LODW            ; AX←(SI) and inc SI while shift finishes
    AND Shifter     ; AX←AX AND Shifter
    MOV AX,(SI)     ; (SI)←AX
    LOOP L          ; dec CX and loop
```

The loop would be preceded by a special case for left of line and by initialization of CX (count), SI (source address), DF (inc/dec flag). It would be followed by a special case for right of line.

Execution time per word = (12+15+14+17) clocks or 7.25 microseconds, plus 1.6 us waiting for main memory, plus wait states added when the source processor falls behind. This is within the 12 us limit suggested above.

The destination processor can operate on the full bit map (19,200 words) in about 170 ms, or 5 frame times. Only .2 of the main memory bandwidth is consumed on Notetaker I.

For store mode, the inner loop is:

```
    RPT MOVW
```

Execution time per word = (6+10) clocks or 2 microseconds, plus .8 us waiting for main memory, plus wait states. Up to .35 of main memory bandwidth is consumed. The bit map can be covered in less than 2 frame times.

Character Generation

The source processor executes something like the following code for each character on a line (DX is the raster width of the font, DI points at the shifter, DF is set to increment, CH is 0, the shifter is ready to shift left, font and string are in main memory):

```
CLOOP:  MOV CharPtr, BP              ;14 BP ← Addr of next char
        INC CharPtr                  ;21 inc string ptr
        MOV 0(BP), BL                ;23 BL ← next character (BH already 0)
        MOV Font, SI                 ;14 SI ← Strike font
        MOV Xtable(BX)(SI), AX       ;26 AX ← starting X of character
        MOV Xtable+1(BX)(SI), BX     ;26 BX ← X of char+1
        SUB AX, BX                   ; 3 BX ← width of character
        MOV BX, Counter              ;14 Initialize counter modulo 4
        ADD AX, BX                   ; 3 restore BX
        DEC BX                       ; 2 BX ← X of end of char
        SR4, AX                      ; 2 word offset of start of char
        SR4, BX                      ; 2 word offset of end of char
        SUB AX, BX                   ; 3 BX ← words to transfer -1
        INC BX                       ; 2 BL ← words to transfer
        ADD Glyphs(SI), AX           ;24 AX ← addr of first source word
        MOV Padding, BH              ;14 Get padding ready
        MOV FontHeight, BP           ;14 Get font height ready

VLOOP:  MOV AX,SI                    ; 2 SI ← BitBlt source addr
        MOV BL,CL                    ; 2 CX ← words to transfer ("n")
        RPT MOVW                     ; 2+22n   Move n words to shifter
        MOV BH, Counter              ;15 Supply padding
        MOV =0,Shifter               ;16
        ADD DX,AX                    ; 3 AX ← next scan line address
        DEC BP                       ; 2 if any scan lines left
        JNZ VLOOP                    ;16 loop

        DEC CharCount                ;21 any characters left
        JNZ CLOOP                    ;16 loop
```

The time to send an h high by 15-or-fewer wide character (n=1 or 2, avg. 1.5) is thus:

244 + 88h clocks

or 162 us for a 12 point character.

Although I haven't written the code, I think the destination processor takes a comparable amount of time. Since the loops are concurrent, the total character generation rate is about 6000 characters per second. This is three times faster than the Alto (a factor of two from multi-processing and the rest from the special hardware). Since the Notetaker will display fewer characters than the Alto, the apparent rate should be even faster.

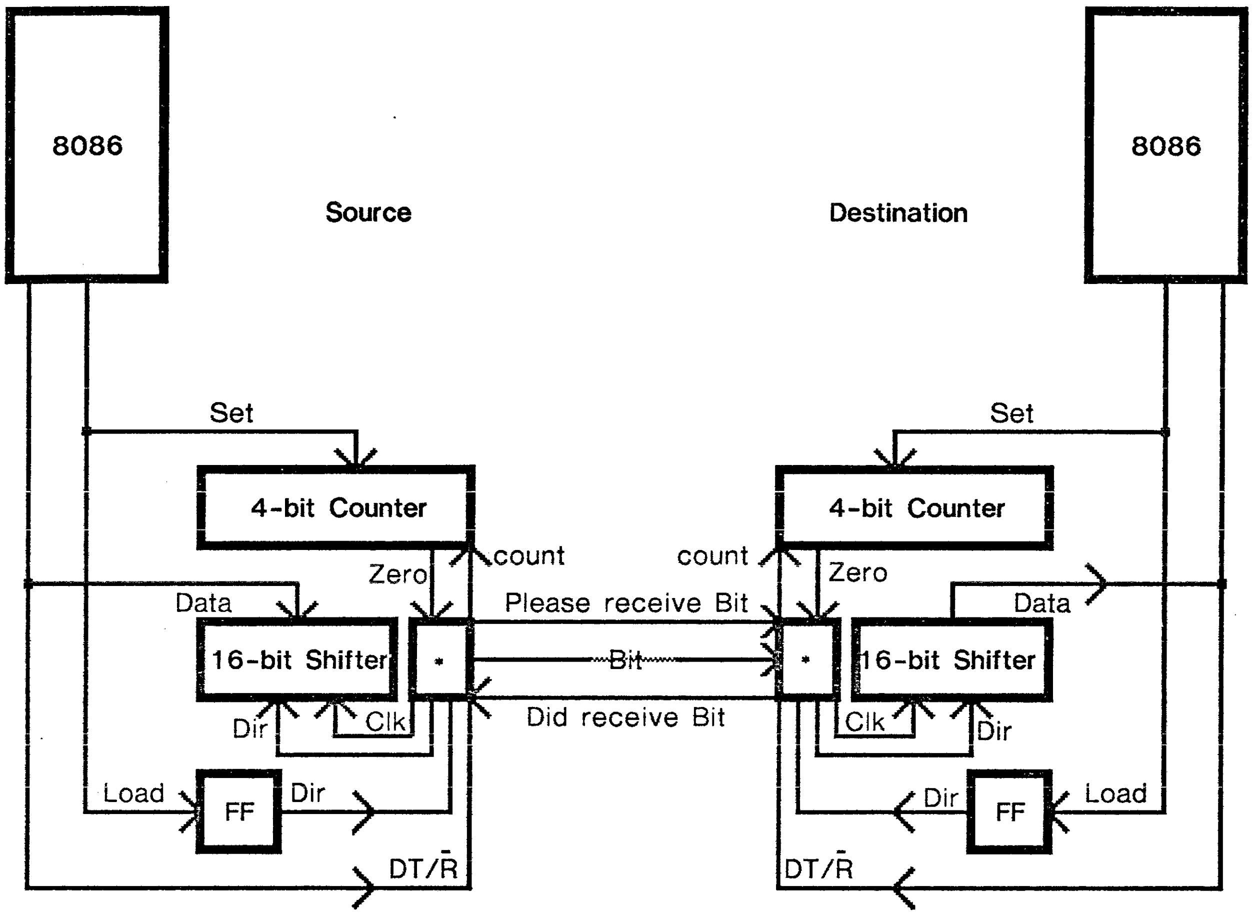
Proposal

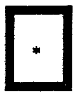
Because of the board space, power requirements, and multi-processor requirements of the above scheme, it seems wisest in the beginning to implement a one-processor BitBlt. When we have an expansion box, we could experiment with multi-processor schemes in it.

Distributed BitBlt

Block Diagram

April 19, 1978




is control logic

driven off a 16 or 32 MHz clock

It starts shifting when Shifter is read or written

It decrements the counter at each shift

It handshakes with the other side

It stops shifting when the Counter is zero

It inhibits access to Shifter & Counter during above

Also can be reset and can report 1 bit of ready status

LT