NASA Technical Memorandum 4095

High-Speed Real-Time Animated Displays on the ADAGE® RDS 3000 Raster Graphics System

William M. Kahlbaum, Jr., and Katrina L. Ownbey
Langley Research Center
Hampton, Virginia

NASA
National Aeronautics and
Space Administration
Office of Management
Scientific and Technical
Information Division
1989

Introduction

Realistic simulation of an aircraft cockpit in a flight simulator requires dynamic instrument panel displays that provide flight information to the pilot. These displays, consisting of dynamic lines, polygons, and alphanumeric characters, must be updated at a sufficiently high rate to realistically simulate actual flight displays. The displays for the Advanced Concepts Simulator (ACS) at NASA Langley Research Center are generated on the ADAGE Raster Display System 3000 (RDS 3000). An overall description of the RDS 3000 can be found in reference 1. Originally the display programs were written using straightforward sequential programming techniques in the IKONAS Display Language (IDL2). (See ref. 2.) IKONAS was the former designation of the RDS 3000. The resulting update rate of 4 frames per second for the primary flight display was insufficient for a realistic simulation. To correct this situation, the programming techniques, language implementation, and hardware characteristics were extensively studied. Improved programming techniques were developed and the language implementation was revised to take better advantage of the high-speed characteristics of the RDS 3000 hardware. The result was a fourfold increase in the update rate to 16 frames per second.

Each of the three processors in the RDS 3000 is designed to perform certain specialized tasks. The main processor, the Bipolar Processor System (BPS), is the master processor of the system, which is normally used to draw lines, polygons, and characters as well as to perform system control functions. The matrix multiplier (MA1024) is a slave processor that is designed to perform coordinate axis transformations at high speed. The Advanced Graphics Generator (AGG4) (ref. 3) is a second slave processor that may be used to draw characters at a higher speed than the BPS. Speed has been improved primarily by operating the processors and certain hardware functions in parallel with each other whenever possible and by revising the system microcode. The microcode revisions mainly focused on increasing the speed of character generation. Speed improvements here resulted from (1) using the parallel processing capabilities of the AGG4 and (2) using the AGG4 to take advantage of certain high-speed characteristics of the display memory that were not previously used. This paper describes techniques used to increase animation update rates by parallel processing and microcode improvements. A sample program illustrating the use of these techniques is included as the appendix.

Symbols and Acronyms

k	integer specifying increasing horizontal displacement, dimensionless
n	integer specifying the multiple of 32 pixels of the vertical displacement, dimensionless
X	vertical screen displacement, pixels
Y	horizontal screen displacement, pixels

Acronyms:

AGG4	Advanced Graphics Generator, Version 4
BPS	Bipolar Processor System
CHAR	character command in the IDL2 display language
CXBS	channel crossbar switch
FBC	frame buffer controller
FBM	frame buffer memory
GM	graphics memory
IDL2	IKONAS Display Language, Version 2
IKONAS	previous designation of the RDS 3000 system
IOR	inserter output register
LUVO	color map and video output module
MA1024	matrix multiplier
OBFM	onboard font memory on the AGG4 processor board
PING	buffer area in memory
PONG	buffer area in memory
RDS 3000	Raster Display System (current designation of the graphics system)
SR8	system scratch memory
XBS	crossbar switch

Display Generation

Description of the Displays

The primary flight display for the Advanced Concepts Simulator is shown in figure 1. The upper half of the display is the attitude display indicator (ADI), which provides the pilot with his primary flight information. Roll and pitch information is presented by the artificial horizon and pitch ladder at the center. Airspeed and altitude are displayed on the dial indicators at the upper left and right. Directional command information is presented by the circular track ball at the center and a speed error is presented by the vertical bar to the left of center. Various command and alert messages are presented by the character strings; and horizontal and vertical errors, by the pointers labeled "H" and "V." The lower half of the display presents navigational information in which the circular compass rose presents heading information and various special symbols such as the small triangle and hexagon present navigational aids. Additional navigational information is presented by the character strings. In both displays some character strings are stationary and some move about the screen; some characters change and some do not. Tracking indicators are displayed above the compass rose in the form of a diamond and a double square.

The modules of the RDS 3000 that generate all the displays are shown in figure 2. All intermodule data communication is over the 32-bit IKONAS bus and several auxiliary buses. Video information passes over the video bus. All images are stored in the frame buffer memory (FBM), which has a maximum display resolution of 1024 by 1024 pixels by up to 24 bits per pixel depending on the amount of memory available. Images are read from the frame buffer memory by the frame buffer controller (FBC) and start toward the video output. The crossbar switch (XBS) and the channel crossbar switch (CXBS) work together in a manner analogous to a telephone switchboard to allow selected individual and groups of frame buffer bit planes to be displayed. The color maps (LUVO) convert the pixel information selected by the crossbar switches to red, green, and blue (RGB) analog video signals for display on the video monitor. The host interface controls communication between the RDS 3000 system and the host machine, which is a Digital Equipment Corp. VAX 8650 superminicomputer. The main processor (BPS) processes program instructions which are stored in the scratch pad memory (SR8). Communication between the BPS and the scratch pad memory is over the IKONAS bus. The matrix multiplier (MA1024) performs matrix operations required for coordinate transformations. Vector data to be transformed are stored in the scratch pad memory and communication between the MA1024 and the scratch pad memory is over an auxiliary bus. This auxiliary bus allows the transformations to proceed without interfering with other data transfers taking place over the IKONAS bus. The parallel processor (AGG4) is used for character generation. All data needed by the AGG4 to draw the characters are stored in a small memory area on the AGG4 board known as the onboard font memory (OBFM). All communication between the processor and this memory area is over an onboard bus, so the only time that it must access the IKONAS bus is when a character or part of a character is ready to be written to the frame buffer memory. In contrast, the BPS must access the IKONAS bus to get vector and polygon vertex information and to write the resulting image information to the frame buffer memory.

Display programs are written in an assembly level language known as IDL2. Commands in this language perform operations such as drawing lines, polygons, and character strings as well as two- and three-dimensional coordinate transformations, clipping, and perspective projection. Display commands are stored in the scratch pad memory and interpreted sequentially by a dispatcher program which runs in the BPS. As each display command is interpreted, this dispatcher directs the execution of a block of microcode resident in the microcode memory (MCM). Communication between the BPS and the microcode memory is by way of an auxiliary bus to prevent the loss of speed that would occur if the IKONAS bus were used. Several microcode files are included with the system and the one used is determined by the hardware configuration. New commands may be created and existing ones modified by using tools that are included with the system. These tools were used to revise the microcode to improve character generation speed. A compiler for IDL2 and a microcode assembler are used to create the executable files.

Theory of Operation

Animation of a display is accomplished by repeatedly drawing new images and presenting them to the observer. These images are created as bit patterns in the frame buffer memory. Smooth animation is accomplished by repeating this process at high speed, typically at least 16 times per second. To prevent the observer from seeing the image creation process, a technique known as double buffering is used. The frame buffer memory is partitioned into two identically sized areas, or buffers, commonly referred to as PING and PONG, and while the new animation frame is being created in one buffer, the current completed frame is being displayed to the observer from the other. In the example of figure 3, bit planes 0 through 3 constitute the PING buffer and bit planes 8 through 11 constitute the PONG buffer. Only dynamic parts of the image are drawn to these bit planes. Static imagery that does not change on the screen is drawn to the static bit planes (4 through 7) when the program is initialized and never changed. The software-controlled write mask is used to select the bit planes to be written into, and the software-controlled crossbar switches (XBS and CXBS) select the bit planes to be passed onto the LUVO's for output to the video. The software alternately selects either PING or PONG to be combined with the static bit planes for output. The case shown in the figure will display the PONG buffer while the next dynamic frame imagery is being written to the PING buffer by the system processors. The 8-bit value coming out of the crossbar switches is applied to the input of the three LUVO's and is a pointer to the same location in each of the color maps. Each color map in the LUVO is a 256-word 10-bit table of memory. As each pixel is addressed, the resulting pointer value output from the crossbar switches points to the same location in each LUVO and the values stored in each of these determine the amount of red, green, and blue that will be displayed at that pixel location.

Animation Process

Four steps must be carried out to create the new animation frame in either buffer:

1. Erasure of the previous animation frame from the buffer
2. Coordinate transformation of vectors, polygons, and character locations to create coordinate information for the next frame
3. Drawing the new animation frame vectors, polygons, and character strings into the newly erased buffer
4. Displaying the new animation frame buffer to the observer

These steps must be executed at maximum speed for a smoothly animated display. Speed is maximized by minimizing the number of instructions and by executing them in parallel whenever possible. Also, microcode may be created or revised to optimize speed.

One more item must be considered in the image generation process. Step 4 of the display generation process described above is performed by the XBS and must be synchronized with the video sweep of the display monitor. This step determines what the observer sees and normally occurs during the vertical blanking interval of the video sweep. This synchronization is accomplished by using an IDL2 instruction called WAITB, which simply delays the changeover of the XBS until the start of the vertical blanking interval. Since the changeover requires less time than the blanking interval, it will be invisible to the observer. Also since steps 1 through 3 are being applied to the nondisplayed buffer, there is no need to synchronize them.

Erase Function

Currently the erase function can be performed in two ways, and selection of the method can have an impact on update rate. The two methods are known as autoclear and selected area erase. The first method, autoclear, is a hardware function in the frame buffer controller (FBC). Two steps are required: (1) select the bit planes to be erased by setting an erase mask and (2) set a bit in one of the FBC control registers. Once these steps are completed, the bits of each pixel selected by the erase mask are erased as each pixel is selected for display by the video output circuitry of the FBC. Autoclear is normally set up right after the XBS is changed during the vertical blanking interval and is left on for two full video field periods. The main advantage of this method is that the BPS is free to do other things during the erase time. The main disadvantage is that it requires one complete video frame interval which is 32 msec in the high-resolution mode used in these displays. During the erase time no drawing may be safely done to the frame buffer memory. The second method, selected area erase, consists of drawing a filled rectangle in the background color over the entire area that needs to be erased. The upgraded version of the IDL2 rectangle fill command is capable of erasing the entire high-resolution screen in 28 msec. The main disadvantage of this method is that the BPS is tied up during the entire erase time. However, if the area to be erased is small relative to the entire screen, then selected area erase requires less time than autoclear. This can be an advantage even though the BPS is busy during the erase time. There is no straightforward method for determining which method is best, and the choice depends on the nature of the display and is best made by experimentation.

Speed Improvements by Programming Techniques

The display program was originally written using sequential in-line code to implement steps 1 through 4 described above. Autoclear was used for erasure, but nothing was done by the BPS during the 32-msec erase time. This resulted in an animation update rate of 4 frames per second. Figure 4 shows the timing for this method. During the erase interval no new imagery may be drawn into the buffer. If imagery were created at a screen location that had not yet been erased, it would disappear when the erase did occur. However, nondrawing functions such as coordinate transformations, clipping, and perspective projection may be performed during the erase interval. The display program was therefore adjusted as illustrated in the timing diagram of figure 5. This change increased update rate from 4 to 8 frames per second. Further increase in update rate was prevented because of the slowness of the character generator which ran on the BPS processor. Installation of the Advanced Graphics Generator (AGG4) was expected to increase the drawing speed of the characters, but did not. The remainder of this paper deals with the extensive analysis of this problem which led to significant revisions of the microcode associated with the character generation command.

Speed Improvements by Microcode Revision

The original microcode provided with the AGG4 did not take full advantage of several advanced design features of the processor and the frame buffer display memory. These problems fell into three categories: (1) not taking advantage of the parallel processing capabilities of the AGG4 processor, (2) not taking advantage of the multiple pixel write capabilities of the GM-type FBM boards which were installed in the system, and (3) drawing characters of only one color and one size in each call of the command.

The original character draw command (CHAR) for the AGG4-based system consisted of microcode files that ran on both the BPS and the AGG4. The BPS code passed screen coordinates and string information for each character string to the AGG4 and then started the AGG4 microcode which did the actual character rendition. As originally designed, the BPS code passed the parameters for one string at a time to the AGG4 and then waited for it to complete its rendition before sending the parameters for the next string. This prevented any parallel operations. Also, the AGG4 microcode processed the characters one pixel at a time. While this did have the advantage of allowing rotation and magnification of characters, it did not take advantage of the multiple pixel write capabilities of the GM-type frame buffer memory boards installed in the system, that is, being able to write 32 pixels at a time to the frame buffer memory rather than only 1. Multiple pixel writes do not easily allow for rotation and magnification, but the speed increase far outweighs this loss of flexibility.

The revised microcode for the two processors corrects the problems described in the previous paragraph and adds several other features. To achieve parallel processing and drawing different sized and color characters in a single call, the BPS microcode is now designed to pass to the AGG4 all the parameters (coordinates and character string information) for all strings in a single call. These parameters are stored in the onboard font memory (OBFM) and up to 500 characters distributed over 100 strings may be handled with a single CHAR command. After the data transfer is complete, the BPS sends a command to the AGG4 to start drawing. Then the BPS is free to do other things such as drawing vectors. After receiving the draw command from the BPS, the AGG4 proceeds to draw the characters. A status bit is set by the AGG4 so that the BPS does not attempt to start another CHAR command until the AGG4 is finished. Figure 6 shows the activity of the three processors when operating in this parallel mode when using the autoclear erase mode and figure 7 shows it when using the selected area erase mode. Rotation and magnification of the characters are designed into the character font data, which also reside in the OBFM. To correct for this loss of the zoom and rotate capabilities, the character fonts are designed with a 90° rotation in the plane of the screen. This compensates for the monitor orientation in the cockpit. Different fonts are provided for the different character sizes. Also, some special symbols such as the track ball indicator in the attitude display indicator and the navigational symbols are implemented as special characters. Several other features built into the command include the capability to draw characters of different sizes and colors in a single CHAR command and the ability to disable the erase function of the AGG4 microcode.

With the parallel processing capabilities of the revised microcode illustrated in figures 6 and 7, the limiting speed is determined by the processor that takes the longest to complete its tasks. If the BPS completes its work at time t_2 when the AGG4 is still busy, then the AGG4 sets the time limit. Then the time required to draw the vectors is essentially free because the BPS would otherwise be idle. If the time for the BPS to complete extends out to t_{2m}, the situation is reversed and the BPS processing time determines the overall speed.

High-Speed Character Generation

Image Memory Mapping

An understanding of image mapping in the frame buffer memory (FBM) is necessary to understand how characters are drawn. Figure 8 illustrates this mapping for the orientation of the monitor used in this project (turned 90° on its side). Each small rectangle (referred to as an "FBM slice") representing 32 pixels in the vertical direction is mapped into one 32-bit word of memory. Thirty-two of these FBM slices are required to define one raster line, which is 1024 pixels high by 1 pixel wide, and these 32 FBM slices are mapped into 32 contiguous words of the

FBM. Moving vertically in figure 8 involves the combination of changing the bit number within a word and changing the word address in memory by a value denoted as n in the figure. Moving horizontally involves holding the bit number within the word constant and changing the word address by increments of 32 which is designated by k in the figure.

The next step is to show how the pixel pattern representing a character is written into the FBM. This process for the letter "R" is illustrated in figure 9. The character is represented by a series of vertical pixel patterns superimposed on the pixel array of the FBM. The vertical slices of the character are stored as bit patterns in a series of contiguous words in the onboard font memory (OBFM). In this case the character slices are 16 bits high so that 2 slices are stored in each 32-bit OBFM word and 5 OBFM words are required to store the 10 slices of the example character. Figure 9 illustrates the manner in which the 16-bit slices are stored in the 32-bit words. The five words of memory required for each character are known as a font entry and each slice of the character is called a font slice. The collection of five-word entries for the entire character set is called the font table. The problem to be solved is how the AGG4 hardware transfers the font slice entries to the FBM. Special hardware illustrated in figure 10 is required to accomplish this transfer, which is done 32 bits at a time. All the modules in figure 10 are located on the AGG4 circuit board.

Positional Computations for the Character

The font array for each character is transferred from the OBFM to the FBM one font slice at a time. The data flow follows the dashed arrows in figure 10. Figure 11 shows the condition of the font entry as it passes through each hardware component of the barrel shifter and of the inserter output register (IOR). The example shown is for the first font slice of the character "R" shown in figure 9 with the character positioned vertically four pixels, or bits, beyond a word boundary of the FBM memory slices. The processing steps are as follows:

1. A font entry word (representing the first two font slices) is transferred from the OBFM to the 32-bit slice register. Note that the upper and lower 16 bits in the slice register of figure 11 correspond to the odd and even font slices of figure 9.
2. The first font slice located in the lower 16 bits of the slice register is rotated from lower to higher bit positions by up to 15 bit positions with the higher bits being wrapped around to the lower bit positions. The amount of rotation is determined by how far the desired vertical position of the character differs from an integer multiple of 32 (which is the vertical separation between FBM slices). The barrel shifter accomplishes this rotation operation in one microcode instruction time to save processing time. In the example, the rotation is four bits which will position the character with a four-pixel offset beyond an FBM slice word boundary.
3. The rotated font entry (16 bits) is then written to the lower and the upper 16 bits of the IOR.
4. There are now two copies of the rotated font entry. The choice of the bits to be output is controlled by the left and right mask values, which are in turn determined by the offset value. The location of the left mask is equal to the offset (in this case four bits). All bits numbered less than the position of the left mask are turned off. The position of the right mask is 16 bits greater than the left mask and all bits numbered greater than it are turned off. When the offset exceeds 16 bits, the rotation is

$$\text{Rotation} = \text{Offset} - 16 \text{ bits}$$

Also two FBM writes are necessary as shown in figure 12 (for an X offset of 20 pixels) to include the portion of the IOR output that spills over into the next FBM slice. The location of the left mask for the FBM write to slice n is still equal to the offset and the right mask is at bit 31. The left mask for the FBM write to slice $(n+1)$ is at bit 0 and the location of the right mask is

$$\text{Right mask} = \text{Offset} - 16 \text{ bits}$$

Note that the barrel shifter rotation is the same for the offset of 4 and 20 bits and that the appropriate contents of the IOR is selected by the setting of the left and right masks.

Once the output for the first 16-bit font slice is complete, steps 1 through 4 are then repeated for the upper 16 bits of the contents of the slice register. Then this entire procedure is repeated for the remaining font slices in the character. For characters that are 32 pixels high, only one font slice is contained in each 32-bit font word in the OBFM. Since the barrel shifter can handle only 16 bits at a time, two passes through the procedures described above are necessary to process one font slice. Also, the spillover to the next FBM slice with its associated additional processing occurs whenever the offset is 1 bit or greater.

The special case of zero offset which occurs when the X coordinate is an integer multiple of 32 is worth noting. In this case no rotation is necessary and the mask values are fixed at

Left mask = 0
Right mask = 15

for the 16-pixel font and

Left mask = 0
Right mask = 31

for the 32-pixel font. The procedures described above reduce to simply writing the font slices out via the IOR. This requires less processing time and may be used advantageously to speed up character generation.

Figure 13 shows the font bit manipulation for the whole character "R" for several offset values ranging from 0 to 24 pixels.

Frame Buffer Memory Writes

The results of the operations described in the previous section are output from the inserter output register where either the font bit pattern or its complement is available for output. This output may go to either the FBM for display or back to the OBFM to be available for further processing. The complemented form of the IOR output must be written to the FBM because of the manner in which the GM-type FBM boards work. In the FBM write, the pixel value as found in the IOR is not written directly to the FBM. Instead the contents of the IOR is written to the FBM mask register and the FBM mask register controls the write operation to the FBM according to the following rules:

1. If the FBM mask register value is 1, then the pixel in the FBM is unchanged.
2. If the FBM mask register value is 0, then the contents of the FBM shade register is written to the pixel. This shade register resides on the GM memory boards and is normally preloaded during program initialization.

In order for a pixel bit to be turned on, two conditions must be met: (1) the bit must be a 1 in the shade register and (2) the corresponding bit of the write mask must be a 1. This write mask is also preloaded, normally at program initialization. Figure 14 illustrates this point for 2 pixel locations of the 32 available from the IOR. The upper half of the figure shows the state of two adjacent pixels prior to a write from the IOR. The pixel value at location 0 of the IOR is 1. Therefore, that pixel remains unchanged no matter what the shade and write mask values are set to and that pixel value in the lower portion of the figure is unchanged. The next pixel in the IOR is set to a value of 0. Now the shade value as modified by the write mask is written to the second pixel location at the bottom of the figure. This process is repeated for the remaining pixels of the FBM slice as shown by bits 2 through 31. The value in a particular bit location of the shade register is written to the FBM only if the corresponding write mask bit is 1.

The data writing characteristics of the FBM boards described above mean that a character is not automatically erased by setting the appropriate bits to a value of 1 in the IOR output. Therefore, the erasure of characters requires a separate operation. Specifically, all pixels that are on in a character and need to be turned off must be written with a shade value of 0 and all write mask bits turned on. The most time-consuming way of accomplishing this would be to redraw all characters with the shade value set to 0. A more efficient way, which works equally well, is to draw a rectangle at least as big as the character with the shade set to 0. This is accomplished by knowing the size and location of the character string to be erased and then drawing the rectangle. This erasure must be done for each animation frame since it is possible that each and every character might change and/or move from frame to frame. As a result, it is necessary to know the location of each character string from frame to frame and this must be done separately for each buffer (PING or PONG). Figure 15 illustrates this. To relieve the applications program from this bookkeeping task, the microcode has been written to handle it.

If the autoclear erase function is used, there is no need for the AGG4 to erase any individual strings. However, if the selected area erase mode is used, any strings outside of the selected area must be erased. The AGG4 microcode has been written to handle the erase function automatically and this erase function may be enabled or disabled for individual strings. The contents and location of the character strings for each buffer (PING and PONG) which are to be erased must be saved when they are drawn so that they are available for erasing prior to drawing the next animation frame. The data transfer is illustrated in figure 16. During the generation of each animation frame, the new string data for the current animation frame and the old string data for the previous frame in the same buffer are passed from the scratch pad memory to the OBFM. These data transfers are indicated by the solid arrows which terminate at the OBFM in figure 16. After the AGG4 is started, the BPS microcode transfers the current frame new string data to the old string data area in the scratch pad memory where it will be available for the next frame. This data transfer is indicated by the dashed arrow in figure 16 and relieves the applications program from this bookkeeping task. For each succeeding animation frame the source of the old character string data for both of these transfers alternates between the PING and PONG storage areas in the figure.

Limitations of the New Microcode

The increase in character drawing speed has its price. The zoom and rotation capabilities of the original character command were lost. This minor problem was corrected by creating font tables with the character size and rotation built in. Three different character sizes were needed: 10 by 14, 20 by 28, and 32 by 32 pixels. The 14 and 28 pixel dimensions were in the vertical direction so that each font slice threw away 2 or 4 bits, respectively. To have made use of these unused bits by data compaction would have added a significant processing burden on the microcode with its attendant increase in processing time. The task of building the font table was simplified by creating several font editor programs on a personal computer which had the character size and rotation built in. The output format of these programs was identical to the data entry format required by the RDS 3000. Several special symbols were created in the 32- by 32-pixel character set. They were the filled track ball and the numbers 1 and 2 with circles around them found in the airspeed meter at the upper left corner of the display (fig. 1). Also, several unused characters such as left and right parentheses were used for the navigation aid symbols. Using characters for these symbols was faster than building them up from graphics primitives such as lines and circles.

Character Draw Speed Measurements

The time required to draw the different sized characters has been measured. The method used was to repeatedly draw a real-time frame containing a known number of characters. This program was run for 30 seconds and the time to draw a single character was computed as follows:

$$T_f = T_{\mathrm{tot}}/N$$

where

T_f time per frame

N number of frames

T_{tot} measured time (normally 30 seconds)

and the time required to draw one character is

$$T_{\mathrm{ch}} = T_f/N_{\mathrm{ch}}$$

where

T_{ch} time per character

N_{ch} number of characters per frame

During this test, all synchronizing wait states were eliminated from the program to give a true time measurement.

Table 1 shows the results of these timing measurements for the new microcode. Note that there is a time penalty when character strings are not positioned vertically on 32 pixel boundaries. Also, there is a significant time penalty for the automatic erase function. If there are areas of the bit map image which are erased by use of the selected area erase function or if autoclear is used, then time may be saved by bypassing the character erase function. Table 2 shows a comparison of character draw times between the new and original character commands. Note that vertical position on the screen has no effect on the speed of the original character command.

Concluding Remarks

The problem of slow animation update rates has been resolved for the primary flight display of the Advanced Concepts Simulator at the Langley Research Center. The original update rate of 4 animation frames per second has been increased to 16 frames per second, which is as fast as the simulation program can currently run. This was accomplished by rewriting the character generator microcode to take better advantage of the hardware capabilities and by structuring the display program to take advantage of the parallel processing capabilities of the ADAGE RDS 3000 system.

NASA Langley Research Center
Hampton, VA 23665-5225
February 6, 1989

Appendix

Example of a Double-Buffered Display Program

The program contained in this appendix is a double-buffered display (see fig. A1) utilizing all the techniques described in this paper. It is written in IDL2 and runs on the RDS 3000 utilizing the revised microcode. The update rate is 27 frames per second and the entire picture is drawn for each frame. The special symbols (filled circles and the circled 1 and 2) are single 32- by 32-pixel characters. The pattern-filled polygon is made up of eight 32- by 32-pixel characters (four for the boundary and four for the pattern fill). All characters change color: the special characters and the polygon change for each frame and the alphanumeric characters change once every 30 frames. The special characters and polygons move around inside the two boxes at the bottom of the display. The three wheels rotate like a set of meshed gears and the selected area erase is used to erase the region containing the wheels by drawing in the background color. The BPS transforms and renders the wheels while the AGG4 is drawing the characters (see fig. 7). The color changes cycle through a set of 16 colors with the boundary and pattern fill of the polygon cycling separately.

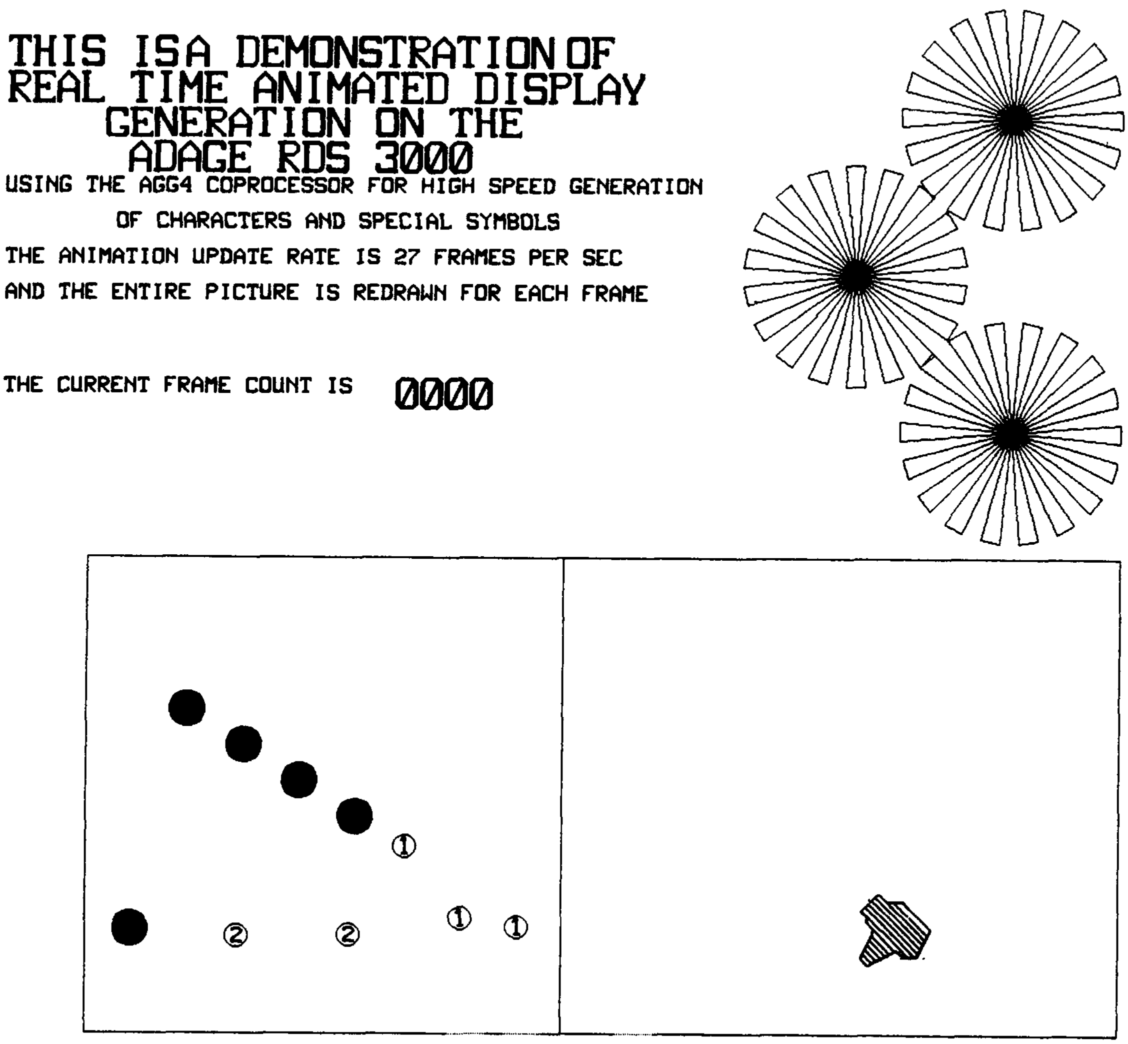

Figure A1. AGG4 demonstration display.

```
;                              APPENDIX 1
;
;            THIS IS AN EXAMPLE OF A DOUBLE BUFFERED DISPALY WHICH
;        WILL RUN ON THE ADAGE RDS 3000 USING THE MODIFIED MICROCODE
;        WHICH IS DESCRIBED IN THIS PAPER.  THE AGG4 PARALLEL PROCESSOR
;        IS USED TO DRAW THE STANDARD AND SPECIAL CHARACTERS.  THE SPECIAL
;        CHARACTERS INCLUDE THE FILLED CIRCLE, THE CIRCLED NUMBERS 1 AND 2
;        AND A CROSS HATCHED POLYGON.  THE POLYGON BOUNDARY IS MADE UP
;        OF 4 SPECIAL CHARACTERS AND THE INTERNAL CROSS HATCHING IS MADE
;        UP OF 4 MORE.  THE SPECIAL CHARACTER AND THE POLYGON MOVE AROUND
;        INSIDE OF AND BOUNCE AGAINST THE SIDES OF TWO BOXES WHICH ARE DRAWN
;        AT THE BOTTOM OF THE SCREEN.  THE CHARACTER COMMAND IS EXECUTED AT
;        THE BEGINNING OF EACH ANIMATION FRAME AND WHILE THE AGG4 IS
;        RENDERING THE CHARACTERS, THE BPS AND MA1024 PROCESSORS ARE
;        PERFORMING THE TRANSFORMATIONS FOR THE CHARACTER POSITIONS ON THE
;        SCREEN AND THE TRANSFORMATION AND RENDITION OF THREE WHEELS WHICH
;        ARE LOCATED AT THE UPPER RIGHT CORNER OF THE DISPLAY.  THE BPS AND
;        MA1024 ARE PROCESSING IN PARALLEL WITH THE AGG4. ERASURE OF
;        THE AREA CONTAINING THE THREE WHEELS IS ACCOMPLISHED BY DRAWING
;        A FILLED RECTANGLE IN THE BACKGROUND COLOR.  THE MICROCODE FOR
;        THE RECTANGLE FILL COMMAND HAS BEEN ENHANCED TO RUN 50 TIMES
;        FASTER THAN THE ORIGINAL.  THE SPECIAL  CHARACTERS CHANGE COLOR
;        THROUGH A PRESCRIBED SEQUENCE OF COLORS FROM ONE ANIMATION FRAME
;        TO THE NEXT.  THE POLYGON BOUNDARY GOES THROUGH ONE SEQUENCE
;        OF COLORS WHILE THE CROSS HATCHING GOES THROUGH A SECOND.
;        THE STANDARD CHARACTERS WHICH ARE FIXED ON THE SCREEN CHANGE
;        COLORS IN A SIMILAR SEQUENCE AFTER EVERY 30 ANIMATION FRAMES.
;        THIS WAS DONE TO GET A SMOOTHER TRANSITION OF COLORS.
;
;
;        DEFINITIONS OF COMMONLY USED CONSTANTS.
;
EQ=1.
NE=2.
LT=3.
LE=4.
GT=5.
GE=6.
AND=7.
OR=8.
XOR=9.
NAND=10.
NOR=11.
XNOR=12.
X3D=0.
X3DP=1.
X3DC=2.
X3D2=3.
X3DP2=4.
COLOR=377
DRAW=100000
RED=^X11
GREEN=^X22
BLUE=^X33
YELLOW=^X44
PINK=^X55
WHITE=^X66
BLACK=^X00
EOL=40000
CHXB=^O203\^O2000
DELTA=3.
```

```
DELTAM=-3.
LUVO=^O203\^OO
AMBER32=^X3333\^X3333
PING=^XO\^XOOOF
PONG=^XO\^XOOFO
;
;       DEFINE CHARACTER SIZE CODES
;
STD=2000                ;STANDARD SIZE 10 X 14
WID=4000                ;DOUBLE SIZE 20 X 28
SPC=6000                ;SPECIAL CHARACTERS 32 X 32
NOERASE=10000           ;TAG FIELD BIT FOR NO ERASE OF CHARACTER
                        ;THIS IS NOT USED IN THIS EXAMPLE PROGRAM
                        ;OBF MEMORY
        INIT    #1
        RESOLU  #1

        MOVE    #0,100\10020            ;INITIALLY CLEAR AGG4 BUSY BIT
        MOVE    #0,CHXB                 ;SET THE CHANNEL X BAR TO RED
;
;       INITIALIZE THE X AND Y INCREMENT VALUES. THESE WILL INCREMENT
;       THE POSITION OF THE MOVING SPECIAL CHARACTERS ON THE SCREEN
;
        MOVE    #DELTA,DELX1
        MOVE    #DELTAM\0,DELY1
        MOVE    #DELTA,DELX2
        MOVE    #DELTAM\0,DELY2
        MOVE    #DELTA,DELX3
        MOVE    #DELTAM\0,DELY3
        MOVE    #DELTAM,DELX4
        MOVE    #DELTAM\0,DELY4
        MOVE    #DELTA,DELX5
        MOVE    #DELTA\0,DELY5
        MOVE    #DELTA,DELX6
        MOVE    #DELTAM\0,DELY6
        MOVE    #DELTA,DELX7
        MOVE    #DELTA\0,DELY7
        MOVE    #DELTA,DELX8
        MOVE    #DELTAM\0,DELY8
        MOVE    #DELTAM,DELX9
        MOVE    #DELTAM\0,DELY9
        MOVE    #DELTA,DELX10
        MOVE    #DELTAM\0,DELY10
        MOVE    #DELTA,DELX11
        MOVE    #DELTA\0,DELY11
;
;       DRAW THE BOUNDARY BOXES FOR THE BOUNCING THINGS
;
        WMASK   #0\400          ;WRITE MASK FOR STATIC BIT PLANE
                                ;WHICH CONTAINS THE BOUNCING THING
                                ;BOUNDARIES
        VECT    VLBOUNCE        ;DRAW THE BOX

        MOVE    #PING,PINGPONG  ;INITIALIZE PINGPONG TO #PING
;
;       PING BUFFER IS BIT PLANES 0 - 3
;       PONG BUFFER IS BIT PLANES 4 - 7
;       STATIC BUFFER IS BIT PLANE 8 AND CONTAINS
;       THE BOUNDARY BOX FOR THE BOUNCING THINGS
;
;       LOAD THE SHADE TABLE WITH SOME COLORS IN THE FIRST 17 LOCATIONS;
```

```
        CMAPLD  SHADE,#0,#77777\0,LUVO,#256.
;
;
;
;       LOCATION "LOOP" DENOTES THE START OF THE NEW ANIMATION FRAME
;       COMPUTATIONS
;
LOOP:
;
;       SET UP THE AGG4 TO DRAW ALL CHARACTERS AT THE BEGINNING OF
;       EACH FRAME
;
        JMPC DPONGCH,PINGPONG,#EQ,#PONG
;
;       DRAW CHARACTERS IN PING BUFFER
;
        CHAR    COOR1_NEW,STRAD1_NEW,ATTAD1,OVLPING,OSTRPING,PINGPONG
;
;       FROM HERE ON THE BPS IS PROCESSING VECTORS WHILE THE AGG4 IS
;       SIMULTANEOUSLY PROCESSING CHARACTERS
;
;       TRANSFORM THE VECTORS FOR THE ROTATING WHEELS
;
        SUB     ROT1+2,#1777\0,ROT1+2
        SUB     ROT2+2,#1777\0,ROT2+2
        ADD     #1777\0,ROT3+2,ROT3+2
        COEFF   #0,#0,#0,ROT1,TRAN1,SCALE1
        XFORM   VL1,VL2,#0,#X3D
        COEFF   #1,#0,#0,ROT2,TRAN2,SCALE2
        XFORM   VL1,VL4,#1,#X3D
        COEFF   #1,#0,#0,ROT3,TRAN3,SCALE3
        XFORM   VL1,VL6,#1,#X3D
        SENDERID        #1
        WMASK   #0\17
        FILREC  RECLIST         ;ERASE THE REGION WHERE THE
                                ;WHEELS ARE DRAWN BY DRAWING
                                ;A RECTANGLE IN THE BACKGROUND
                                ;COLOR USING THE NEW HIGH SPEED
                                ;FILREC COMMAND
        VECT    VL2             ;DRAW THE NEW VERSION OF THE WHEELS
        VECT    VL4
        VECT    VL6
        JMP     LOOPC
DPONGCH:
;
;       DRAW CHARACTERS IN PONG BUFFER
;

        CHAR    COOR1_NEW,STRAD1_NEW,ATTAD1,OVLPONG,OSTRPONG,PINGPONG
;
;       TRANSFORM THE VECTORS FOR THE ROTATING WHEELS
;
        SUB     ROT1+2,#1777\0,ROT1+2
        SUB     ROT2+2,#1777\0,ROT2+2
        ADD     #1777\0,ROT3+2,ROT3+2
        COEFF   #0,#0,#0,ROT1,TRAN1,SCALE1
        XFORM   VL1,VL2,#0,#X3D
        COEFF   #1,#0,#0,ROT2,TRAN2,SCALE2
        XFORM   VL1,VL4,#1,#X3D
        COEFF   #1,#0,#0,ROT3,TRAN3,SCALE3
        XFORM   VL1,VL6,#1,#X3D
```

```
        SENDERID        #1
        WMASK   #0\360
        FILREC  RECLIST         ;ERASE THE REGION WHERE THE
                                ;WHEELS ARE DRAWN BY DRAWING
                                ;A RECTANGLE IN THE BACKGROUND
                                ;COLOR USING THE NEW HIGH SPEED
                                ;FILREC COMMAND
        VECT    VL2             ;DRAW THE NEW VERSION OF THE WHEELS
        VECT    VL4
        VECT    VL6
LOOPC:

;
;       INCREMENT FRAME COUNTER DISPLAY
;
;
        ADD     I3,#1,I3                ; INCREMENT UNITS
        JMPC    INCPL,I3,#LT,#10.
        MOVE    #0,I3
        ADD     I2,#1,I2                ; INCREMENT TENS
        JMPC    INCPL,I2,#LT,#10.
        MOVE    #0,I2
        ADD     I1,#1,I1                ; INCREMENT HUNDREDS
        JMPC    INCPL,I1,#LT,#10.
;
;               RESET ALL DIGITS TO ZERO IF LAST COUNT WAS 999
;
        MOVE    #0,I3
        MOVE    #0,I2
        MOVE    #0,I1
INCPL:  GTABLE  GS1,DIGIT,I1
        SHIFT   GS1,#8.,GS1
        GTABLE  GS2,DIGIT,I2
        SHIFT   GS2,#16.,GS2
        GTABLE  GS3,DIGIT,I3
        SHIFT   GS3,#24.,GS3
;
;       UPDATE THE STRING FOR THE FRAME COUNTER
;
        MOVE    #40,STR22
        OR      GS1,STR22,STR22
        OR      GS2,STR22,STR22
        OR      GS3,STR22,STR22
;
;       FRAME COUNTER UPDATE COMPLETE
;
;       UPDATE THE NEW BUFFER
;
;
;
;       UPDATE MID SIZE AND STANDARD SIZE CHAR COLORS
;
;
;       SOME OF THE STRINGS MOVE AND CHANGE COLORS AT THE SAME TIME
;       THEY WILL BE UPDATED FIRST START THE CHANGES FOR THESE
;       STRINGS BY UPDATING COOR1 THROUGH COOR18
;
;       SET THE MAXIMUM AND MINIMUM VALUES FOR THE BOUNCING BALLS
;
        MOVE    #500.,XMIN
        MOVE    #890.,XMAX
```

```
        MOVE    #535.\0,YMIN
        MOVE    #925.\0,YMAX
;
;       NOW DO THE INCREMENTS AND COLOR CHANGES TO THESE STRINGS
;
        MOVE    #COOR1,REG1
        MOVE    #DELX1,REG2
        MOVE    #DELY1,REG3
        JMPSUB  CHANGE
        MOVE    #COOR2,REG1
        MOVE    #DELX2,REG2
        MOVE    #DELY2,REG3
        JMPSUB  CHANGE
        MOVE    #COOR3,REG1
        MOVE    #DELX3,REG2
        MOVE    #DELY3,REG3
        JMPSUB  CHANGE
        MOVE    #COOR4,REG1
        MOVE    #DELX4,REG2
        MOVE    #DELY4,REG3
        JMPSUB  CHANGE
        MOVE    #COOR5,REG1
        MOVE    #DELX5,REG2
        MOVE    #DELY5,REG3
        JMPSUB  CHANGE
;
;       SET THE MAXIMUM AND MINIMUM VALUES FOR THE BOUNCING SPECIAL
;       CHARACTERS "CIRCLED 1 AND CIRCLED 2"
;
        MOVE    #500.,XMIN
        MOVE    #903.,XMAX
        MOVE    #522.\0,YMIN
        MOVE    #925.\0,YMAX

        MOVE    #COOR6,REG1
        MOVE    #DELX6,REG2
        MOVE    #DELY6,REG3
        JMPSUB  CHANGE
        MOVE    #COOR7,REG1
        MOVE    #DELX7,REG2
        MOVE    #DELY7,REG3
        JMPSUB  CHANGE
        MOVE    #COOR8,REG1
        MOVE    #DELX8,REG2
        MOVE    #DELY8,REG3
        JMPSUB  CHANGE
        MOVE    #COOR9,REG1
        MOVE    #DELX9,REG2
        MOVE    #DELY9,REG3
        JMPSUB  CHANGE
        MOVE    #COOR10,REG1
        MOVE    #DELX10,REG2
        MOVE    #DELY10,REG3
        JMPSUB  CHANGE

;
;       NOW CHANGE THOSE STATIONARY STRINGS WHICH ONLY CHANGE COLOR
;       AND ONLY DO IT AFTER EVERY 30 FRAMES HAVE BEEN DRAWN
;
        ADD     #1,FCT,FCT
        JMPC    NOCOLUPD,FCT,#NE,#30.
```

```
        MOVE    #0,FCT
        MOVE    #COORL1,REG1
        JMPSUB  CHGCOL
        MOVE    #COORL2,REG1
        JMPSUB  CHGCOL
        MOVE    #COORL3,REG1
        JMPSUB  CHGCOL
        MOVE    #COORL4,REG1
        JMPSUB  CHGCOL
        MOVE    #COORL5,REG1
        JMPSUB  CHGCOL
        MOVE    #COORL6,REG1
        JMPSUB  CHGCOL
        MOVE    #COORL7,REG1
        JMPSUB  CHGCOL
        MOVE    #COORL8,REG1
        JMPSUB  CHGCOL
        MOVE    #COORL9,REG1
        JMPSUB  CHGCOL
        MOVE    #COORL10,REG1
        JMPSUB  CHGCOL
        MOVE    #COORL11,REG1
        JMPSUB  CHGCOL
        MOVE    #COORL12,REG1
        JMPSUB  CHGCOL
NOCOLUPD:
;
;       UPDATE THE COLOR AND COORDINATES
;       OF THE SPECIAL CHARACTERS WHICH MAKE UP THE
;       POLYGON THIS IS DONE EVERY FRAME
;
;
;       SET THE MAXIMUM AND MINIMUM VALUES FOR THE POLYGON FILL AND
;       POLYGON BOUNDARY
;
        MOVE    #500.,XMIN
        MOVE    #865.,XMAX
        MOVE    #60.\0,YMIN
        MOVE    #500.\0,YMAX
;
;       NOW UPDATE IT
;
        MOVE    #COOR11,REG1
        MOVE    #DELX11,REG2
        MOVE    #DELY11,REG3
        JMPSUB  CHANGE                  ;CHANGE COLOR AND COORDINATE
        MOVE    #COOR15,REG1
        JMPSUB  CHGCOL                  ;CHANGE ONLY THE COLOR
        MOVE    COOR11+1,COOR12+1
        MOVE    COOR11+1,COOR13+1
        MOVE    COOR11+1,COOR14+1
        MOVE    COOR15+1,COOR16+1
        MOVE    COOR15+1,COOR17+1
        MOVE    COOR15+1,COOR18+1
;
;
;       UPDATE THE COORD OF THE SPECIAL SYMBOLS WHICH MAKE UP
;       CROSS HATCHED POLYGON BY ADDING OR SUBTRACTING THE APPROPRIATE
;       CONSTANTS TO COOR11 X AND Y COMP
;
        ADD     COOR11,#-32.\0.,COOR12
```

```
        ADD     COOR11,#0.\32.,COOR13
        ADD     COOR11,#-32.\32.,COOR14
;
;       DUPLICATE THE COORDINATES OF THE POLYGON CROSSHATCH FILL
;       BY COPYING THE CORRESPONDING COORDINATES OF THE SPECIAL
;       CHARACTERS WHICH MAKE UP THE POLYGON BOUNDARY
;
        MOVE    COOR11,COOR15
        MOVE    COOR12,COOR16
        MOVE    COOR13,COOR17
        MOVE    COOR14,COOR18
;
;       CHANGE THE CROSSBAR SWITCH
;
LOOP1:  JMPC DPONG,PINGPONG,#EQ,#PONG
        ADD     #0\1,COUNT,COUNT
        MOVE    #PONG,PINGPONG
        WAITB
        BLKMOVE PINGTB,302\0,#34.       ;SET XBAR TO DISPLAY PING
        JMP     LOOP                    ;GO BACK TO "LOOP" TO START
                                        ;THE NEXT ANIMATION FRAME
DPONG:
        ADD     #1,COUNT,COUNT
        MOVE    #PING,PINGPONG
        WAITB
        BLKMOVE PONGTB,302\0,#34.       ;SET XBAR TO DISPLAY PONG
        JMP     LOOP                    ;GO BACK TO "LOOP" TO START
                                        ;THE NEXT ANIMATION FRAME
;
;       THIS SUBROUTINE INCREMENTS COORDINATES AND CHANGES COLOR
CHANGE:
;       THIS IS THE ENTRY POINT TO CHANGE COORDINATES AND COLORS BOTH
;       FIRST INCREMENT THE COORDINATE AS POINTED TO BY REG1
;
        MOVE    @REG2,DELX
        MOVE    @REG3,DELY
        AND     @REG1,#0\177777,TEMP1           ;TEMP1 HAS THE X COORD
        AND     @REG1,#177777\0,TEMP2           ;TEMP1 HAS THE Y COORD
;
;       INCREMENT THE COORDINATES
;
        JMPC    PX1,TEMP1,#LT,XMAX      ;TEST FOR MAXIMUM X
        MOVE    #DELTAM,DELX
        JMP     PY1
PX1:    JMPC    PY1,TEMP1,#GT,XMIN      ;TEST FOR MINIMUM X
        MOVE    #DELTA,DELX
PY1:    JMPC    PY2,TEMP2,#LT,YMAX      ;TEST FOR MAXIMUM Y
        MOVE    #DELTAM\0,DELY
        JMP     PFIN1
PY2:    JMPC    PFIN1,TEMP2,#GT,YMIN    ;TEST FOR MINIMUM Y
        MOVE    #DELTA\0,DELY
PFIN1:  ADD     DELX,TEMP1,TEMP1
        ADD     DELY,TEMP2,TEMP2
;
;       INCREMENT COMPLETE
;
        OR      TEMP1,TEMP2,@REG1
        MOVE    DELX,@REG2
        MOVE    DELY,@REG3
;
;       INCREMENT REG1 TO POINT TO THE SHADE LOCATION
```

```
;
CHGCOL:
;
;       THIS IS THE ENTRY POINT TO ONLY CHANGE COLOR
;
        ADD     #1,REG1,REG1
;
;       NOW INCREMENT THE COLOR AS POINTED TO BY REG1
;
        AND     @REG1,#17\0,TEMP1               ;ISOLATE PING COLOR
        AND     @REG1,#360\0,TEMP2              ;ISOLATE PONG COLOR
        ADD     TEMP1,#1\0,TEMP1
        AND     TEMP1,#17\0,TEMP1
        JMPC    COLA,TEMP1,#NE,#0
        MOVE    #1\0,TEMP1
COLA:   ADD     TEMP2,#20\0,TEMP2
        AND     TEMP2,#360\0,TEMP2
        JMPC    COLB,TEMP2,#NE,#0
        MOVE    #20\0,TEMP2
COLB:
        OR      TEMP1,TEMP2,TEMP3
        AND     @REG1,#177400\0,@REG1
        OR      TEMP3,@REG1,@REG1
        RETURN
;
;
;
        .SETORG ^0202\^07000
;
;
;
;THIS IS THE COORDINATE LIST FOR THE NEW ANIMATION FRAME CHARACTER BUFFER
;
COOR1_NEW:
;
;       NEW COORDINATES FOR BOUNCING BALLS AND CIRCLED 1 AND 2
;
COOR1:  .WORD   900.\816.
        .WORD   (RED!SPC)\0
COOR2:  .WORD   850.\620.
        .WORD   (GREEN!SPC)\0
COOR3:  .WORD   800.\652.
        .WORD   (BLUE!SPC)\0
COOR4:  .WORD   750.\684.
        .WORD   (YELLOW!SPC)\0
COOR5:  .WORD   700.\716.
        .WORD   (PINK!SPC)\0
COOR6:  .WORD   650.\748.
        .WORD   (WHITE!SPC)\0
COOR7:  .WORD   600.\812.
        .WORD   (PINK!SPC)\0
COOR8:  .WORD   550.\820.
        .WORD   (YELLOW!SPC)\0
COOR9:  .WORD   800.\828.
        .WORD   (YELLOW!SPC)\0
COOR10: .WORD   700.\827.
        .WORD   (GREEN!SPC)\0
;
;       NEW COORDINATES FOR CROSS HATCHED POLYGON
;
COOR11: .WORD   232.\800.
```

```
        .WORD   (GREEN!SPC)\0
COOR12: .WORD   200.\800.
        .WORD   (GREEN!SPC)\0
COOR13: .WORD   232.\832.
        .WORD   (GREEN!SPC)\0
COOR14: .WORD   200.\832.
        .WORD   (GREEN!SPC)\0
COOR15: .WORD   232.\800.
        .WORD   (WHITE!SPC)\0
COOR16: .WORD   200.\800.
        .WORD   (WHITE!SPC)\0
COOR17: .WORD   232.\832.
        .WORD   (WHITE!SPC)\0
COOR18: .WORD   200.\832.
        .WORD   (WHITE!SPC)\0
;
;       NEW COORDINATES FOR THE WIDE CHARACTERS
;
COORL1: .WORD   1000.\32.
        .WORD   (RED!WID)\0
COORL2: .WORD   840.\32.
        .WORD   (WHITE!WID)\0
COORL3: .WORD   500.\32.
        .WORD   (YELLOW!WID)\0
COORL4: .WORD   1000.\64.
        .WORD   (WHITE!WID)\0
COORL5: .WORD   1000.\96.
        .WORD   (BLUE!WID)\0
COORL6: .WORD   1000.\128.
        .WORD   (RED!WID)\0
;
;       NEW COORDINATES FOR THE FRAME COUNTER AND ITS HEADER
;
COORL7: .WORD   1000.\340.
        .WORD   (BLUE!STD)\0
COORL8: .WORD   650.\340.       ;LOCATION OF COUNTER
        .WORD   (RED!WID)\0
;
;       NEW COORDINATES FOR STD CHARS
;
COORL9:         .WORD   1000.\160.
                .WORD   (WHITE!STD)\0
COORL10:        .WORD   900.\192.
                .WORD   (GREEN!STD)\0
COORL11:        .WORD   1000.\224.
                .WORD   (YELLOW!STD)\0
COORL12:        .WORD   1000.\256.
                .WORD   (RED!STD!EOL)\0
;
;
;       THIS IS THE OLD COORDINATE LIST FOR ERASING THE CHARACTERS
;       IN THE PING BUFFER FOR THE PREVIOUS ANIMATION FRAME
;       IT IS INITIALLY THE SAME AS THE NEW BUFFER
;
;       OLD PING BUFFER COORDINATES FOR BOUNCING BALLS
;       AND CIRCLED 1 AND 2
OVLPING:
        .WORD   900.\816.
        .WORD   (RED!SPC)\0
        .WORD   850.\620.
        .WORD   (GREEN!SPC)\0
```

```
        .WORD   800.\652.
        .WORD   (BLUE!SPC)\0
        .WORD   750.\684.
        .WORD   (YELLOW!SPC)\0
        .WORD   700.\716.
        .WORD   (PINK!SPC)\0
        .WORD   650.\748.
        .WORD   (WHITE!SPC)\0
        .WORD   600.\912.
        .WORD   (PINK!SPC)\0
        .WORD   550.\920.
        .WORD   (YELLOW!SPC)\0
        .WORD   500.\928.
        .WORD   (BLUE!SPC)\0
        .WORD   450.\927.
        .WORD   (GREEN!SPC)\0
;
;       OLD PING BUFFER COORDINATES FOR CROSS HATCHED POLYGON
;
        .WORD   232.\800.
        .WORD   (GREEN!SPC)\0
        .WORD   200.\800.
        .WORD   (GREEN!SPC)\0
        .WORD   232.\832.
        .WORD   (GREEN!SPC)\0
        .WORD   200.\832.
        .WORD   (GREEN!SPC)\0
        .WORD   232.\800.
        .WORD   (WHITE!SPC)\0
        .WORD   200.\800.
        .WORD   (WHITE!SPC)\0
        .WORD   232.\832.
        .WORD   (WHITE!SPC)\0
        .WORD   200.\832.
        .WORD   (WHITE!SPC)\0
;
;       OLD PING BUFFER COORDINATES FOR THE WIDE CHARACTERS
;
        .WORD   1000.\32.
        .WORD   (RED!WID)\0
        .WORD   840.\32.
        .WORD   (WHITE!WID)\0
        .WORD   500.\32.
        .WORD   (YELLOW!WID)\0
        .WORD   1000.\64.
        .WORD   (WHITE!WID)\0
        .WORD   1000.\96.
        .WORD   (BLUE!WID)\0
        .WORD   1000.\128.
        .WORD   (RED!WID)\0
;
;       OLD PING BUFFER COORDINATES FOR LAST TWO ROWS OF LARGE CHARACTERS
;
        .WORD   1000.\288.
        .WORD   (BLUE!STD)\0
        .WORD   800.\340.        ;LOCATION OF COUNTER
        .WORD   (RED!WID)\0
;
;       OLD PING BUFFER COORDINATES FOR STD CHARS
;
                .WORD   1000.\160.
```

```
		.WORD	(WHITE!STD)\0
		.WORD	900.\192.
		.WORD	(GREEN!STD)\0
		.WORD	1000.\224.
		.WORD	(YELLOW!STD)\0
		.WORD	1000.\256.
		.WORD	(RED!STD!EOL)\0
;
;	THIS IS THE OLD COORDINATE LIST FOR ERASING THE CHARACTERS
;	IN THE PONG BUFFER FOR THE PREVIOUS ANIMATION FRAME
;	IT IS INITIALLY THE SAME AS THE NEW BUFFER
;
;	OLD PONG BUFFER COORDINATES FOR BOUNCING BALLS AND
;	CIRCLED 1 AND 2
OVLPONG:
	.WORD	900.\816.
	.WORD	(RED!SPC)\0
	.WORD	850.\620.
	.WORD	(GREEN!SPC)\0
	.WORD	800.\652.
	.WORD	(BLUE!SPC)\0
	.WORD	750.\684.
	.WORD	(YELLOW!SPC)\0
	.WORD	700.\716.
	.WORD	(PINK!SPC)\0
	.WORD	650.\748.
	.WORD	(WHITE!SPC)\0
	.WORD	600.\912.
	.WORD	(PINK!SPC)\0
	.WORD	550.\920.
	.WORD	(YELLOW!SPC)\0
	.WORD	500.\928.
	.WORD	(BLUE!SPC)\0
	.WORD	450.\927.
	.WORD	(GREEN!SPC)\0
;
;	OLD PONG BUFFER COORDINATES FOR CROSS HATCHED POLYGON
;
	.WORD	232.\800.
	.WORD	(GREEN!SPC)\0
	.WORD	200.\800.
	.WORD	(GREEN!SPC)\0
	.WORD	232.\832.
	.WORD	(GREEN!SPC)\0
	.WORD	200.\832.
	.WORD	(GREEN!SPC)\0
	.WORD	232.\800.
	.WORD	(WHITE!SPC)\0
	.WORD	200.\800.
	.WORD	(WHITE!SPC)\0
	.WORD	232.\832.
	.WORD	(WHITE!SPC)\0
	.WORD	200.\832.
	.WORD	(WHITE!SPC)\0
;
;	OLD PONG BUFFER COORDINATES FOR THE WIDE CHARACTERS
;
:	.WORD	1000.\32.
	.WORD	(RED!WID)\0
	.WORD	840.\32.
	.WORD	(WHITE!WID)\0
```

```
        .WORD   500.\32.
        .WORD   (YELLOW!WID)\0
        .WORD   1000.\64.
        .WORD   (WHITE!WID)\0
        .WORD   1000.\96.
        .WORD   (BLUE!WID)\0
        .WORD   1000.\128.
        .WORD   (RED!WID)\0
;
;       OLD PONG BUFFER COORDINATES FOR LAST TWO ROWS OF LARGE CHARACTERS
;
        .WORD   1000.\288.
        .WORD   (BLUE!STD)\0
        .WORD   800.\340.       ;LOCATION OF COUNTER
        .WORD   (RED!WID)\0
;
;       OLD PONG BUFFER COORDINATES FOR STD CHARS
;
                .WORD   1000.\160.
                .WORD   (WHITE!STD)\0
                .WORD   900.\192.
                .WORD   (GREEN!STD)\0
                .WORD   1000.\224.
                .WORD   (YELLOW!STD)\0
                .WORD   1000.\256.
                .WORD   (RED!STD!EOL)\0
;
;       STRING POINTERS FOR THE SPECIAL SYMBOLS AND POLYGON
;
STRAD1_NEW:
STRAD2: .WORD   STR10,STR10,STR10,STR10,STR10
        .WORD   STR11,STR11,STR11,STR12,STR12
STRAD3: .WORD   STR13,STR14,STR15,STR16,STR17,STR18,STR19,STR20
;
;       STRING POINTER FOR FIRST 3 ROWS OF LARGE CHARACTERS
;
STADL1: .WORD   STR1,STR2,STR3,STR4,STR5,STR6,STR21,STR22
;
;       STRING POINTER FOR THE FOUR ROWS OF STANDARD SIZE CHARACTERS
;
STADL2: .WORD   STR7,STR8,STR9,STRA
;
;       STRING POINTERS FOR OLD STRINGS PING BUFFER
;
;
;       THIS IS THE OLD STRING POINTER LIST FOR ERASING THE CHARACTERS
;       IN THE PING BUFFER FOR THE PREVIOUS ANIMATION FRAME
;       IT IS INITIALLY THE SAME AS THE NEW BUFFER

OSTRPING:
        .WORD   STR10,STR10,STR10,STR10,STR10
        .WORD   STR11,STR11,STR11,STR12,STR12
        .WORD   STR13,STR14,STR15,STR16,STR17,STR18,STR19,STR20
;
;       STRING POINTER FOR FIRST 3 ROWS OF LARGE CHARACTERS
;
        .WORD   STR1,STR2,STR3,STR4,STR5,STR6,STR21,STR22
;
;       STRING POINTER FOR THE FOUR ROWS OF STANDARD SIZE CHARACTERS
;
        .WORD   STR7,STR8,STR9,STRA
```

```
;
;        THIS IS THE OLD STRING POINTER LIST FOR ERASING THE CHARACTERS
;        IN THE PONG BUFFER FOR THE PREVIOUS ANIMATION FRAME
;        IT IS INITIALLY THE SAME AS THE NEW BUFFER

OSTRPONG:
         .WORD   STR10,STR10,STR10,STR10,STR10
         .WORD   STR11,STR11,STR11,STR12,STR12
         .WORD   STR13,STR14,STR15,STR16,STR17,STR18,STR19,STR20
;
;        STRING POINTER FOR FIRST 3 ROWS OF LARGE CHARACTERS
;
         .WORD   STR1,STR2,STR3,STR4,STR5,STR6,STR21,STR22
;
;        STRING POINTER FOR THE FOUR ROWS OF STANDARD SIZE CHARACTERS
;
         .WORD   STR7,STR8,STR9,STRA
;
;        STRINGS FOR SPECIAL SYMBOLS THIS IS THE SAME FOR BOTH BUFFERS
;
STR10:   .BYTE   'A',0
STR11:   .BYTE   'B',0
STR12:   .BYTE   'C',0
;
;        STRINGS FOR CROSS HATCHED POLYGON
;
STR13:   .BYTE   'D',0
STR14:   .BYTE   'E',0
STR15:   .BYTE   'F',0
STR16:   .BYTE   'G',0
STR17:   .BYTE   'H',0
STR18:   .BYTE   'I',0
STR19:   .BYTE   'J',0
STR20:   .BYTE   'K',0
;
;        STRINGS FOR FIRST 3 ROWS OF LARGE CHARACTERS
;
STR1:    .BYTE   'THIS IS ',0
STR2:    .BYTE   'A DEMONSTRATION ',0
STR3:    .BYTE   'OF',0
STR4:    .BYTE   'REAL TIME ANIMATED DISPLAY',0
STR5:    .BYTE   '    GENERATION ON THE ',0
STR6:    .BYTE   '     ADAGE RDS 3000 ',0
;
;        STRINGS FOR THE FOUR ROWS OF STANDARD SIZE CHARACTERS
;
STR7:    .BYTE   'USING THE AGG4 COPROCESSOR FOR HIGH SPEED GENERATION',0
STR8:    .BYTE   'OF CHARACTERS AND SPECIAL SYMBOLS ',0
STR9:    .BYTE   'THE ANIMATION UPDATE RATE IS 27 FRAMES PER SEC',0
STRA:    .BYTE   'AND THE ENTIRE PICTURE IS REDRAWN FOR EACH FRAME',0
;
;         STRINGS FOR THE LAST TWO ROWS OF LARGE CHARACTERS
;
STR21:   .BYTE   'THE CURRENT FRAME COUNT IS',0
STR22:   .BYTE   '0000',0
;
;        CROSSBAR SETTINGS FOR PING BUFFER
;
PINGTB:  .WORD   0,1,2,3,10,77,77,77
         .WORD   77,77,77,77,77,77,77,77
         .WORD   77,77,77,77,77,77,77,77
```

```
        .WORD   77,77,77,77,77,77,77,77
        .WORD   77,77
;
;       CROSSBAR SETTINGS FOR PONG BUFFER
;
PONGTB: .WORD   4,5,6,7,10,77,77,77
        .WORD   77,77,77,77,77,77,77,77
        .WORD   77,77,77,77,77,77,77,77
        .WORD   77,77,77,77,77,77,77,77
        .WORD   77,77
;
;       STORAGE FOR FRAME COUNTER DIGITS
;
DIGIT:  .BYTE   '0',0,0,0
        .BYTE   '1',0,0,0
        .BYTE   '2',0,0,0
        .BYTE   '3',0,0,0
        .BYTE   '4',0,0,0
        .BYTE   '5',0,0,0
        .BYTE   '6',0,0,0
        .BYTE   '7',0,0,0
        .BYTE   '8',0,0,0
        .BYTE   '9',0,0,0
GS1:    .WORD   0\0
GS2:    .WORD   0\0
GS3:    .WORD   0\0
GS4:    .WORD   0\0
I1:     .WORD   0\0
I2:     .WORD   0\0
I3:     .WORD   0\0
I4:     .WORD   0\0
IC:     .WORD   0\0
INCT:   .WORD   0\0
ZERO:   .WORD   0\0
TEMP1:  .WORD   0\0
TEMP2:  .WORD   0\0
TEMP3:  .WORD   0\0
FCT:    .WORD   0\0
ICT:    .WORD   0\0
PINGPONG:       .WORD   0\0
DELX:   .WORD   0\0
DELY:   .WORD   0\0
DELX1:  .WORD   0\0
DELY1:  .WORD   0\0
DELX2:  .WORD   0\0
DELY2:  .WORD   0\0
DELX3:  .WORD   0\0
DELY3:  .WORD   0\0
DELX4:  .WORD   0\0
DELY4:  .WORD   0\0
DELX5:  .WORD   0\0
DELY5:  .WORD   0\0
DELX6:  .WORD   0\0
DELY6:  .WORD   0\0
DELX7:  .WORD   0\0
DELY7:  .WORD   0\0
DELX8:  .WORD   0\0
DELY8:  .WORD   0\0
DELX9:  .WORD   0\0
DELY9:  .WORD   0\0
DELX10: .WORD   0\0
```

```
DELY10: .WORD    0\0
DELX11: .WORD    0\0
DELY11: .WORD    0\0
DEL:    .WORD    0\0
DEL2:   .WORD    0\0
DEL3:   .WORD    0\0
DEL4:   .WORD    0\0
DEL5:   .WORD    0\0
DEL6:   .WORD    0\0
DEL7:   .WORD    0\0
DEL8:   .WORD    0\0
DEL9:   .WORD    0\0
DEL10:  .WORD    0\0
TEST1:  .WORD    0\0
TST1:   .WORD    0\0
XMIN:   .WORD    0\0
XMAX:   .WORD    0\0
YMIN:   .WORD    0\0
YMAX:   .WORD    0\0
;
;       TABLES OF ROTATION, TRANSLATION AND SCALING FOR THE
;       TRANSFORMATIONS
;
ROT1:   .WORD    0\0
        .WORD    0\0
        .WORD    0\0
TRAN1:  .WORD    0\105.
        .WORD    0\100.
        .WORD    0\0
SCALE1: .WORD    0\77777
        .WORD    0\77777
        .WORD    0\77777
ROT2:   .WORD    0\0
        .WORD    0\0
        .WORD    0\0
TRAN2:  .WORD    0\387.
        .WORD    0\100.
        .WORD    0\0
SCALE2: .WORD    0\77777
        .WORD    0\77777
        .WORD    0\77777
ROT3:   .WORD    0\0
        .WORD    0\0
        .WORD    0\0
TRAN3:  .WORD    0\246.
        .WORD    0\241.
        .WORD    0\0
SCALE3: .WORD    0\77777
        .WORD    0\77777
        .WORD    0\77777
;
;       VECTORS LIST FOR THE CONTAINING BOXES
;
VLBOUNCE:
        .WORD    500.\500.
        .WORD    0\0
        .WORD    500.\925.
        .WORD    (DRAW!RED)\0
        .WORD    925.\925.
        .WORD    (DRAW!RED)\0
        .WORD    925.\500.
```

```
        .WORD   (DRAW|RED)\0
        .WORD   500.\500.
        .WORD   (DRAW|RED)\0
        .WORD   0.\500.
        .WORD   0\0
        .WORD   0.\925.
        .WORD   (DRAW|RED)\0
        .WORD   500.\925.
        .WORD   (DRAW|RED)\0
        .WORD   500.\500.
        .WORD   (DRAW|RED)\0
        .WORD   0.\500.
        .WORD   (EOL|DRAW|RED)\0
;
;       VECTOR DEFINITIONS FOR THE ROTATING WHEELS
;
VL1:    .WORD   0\100.
        .WORD   0\0
        .WORD   17.\98.
        .WORD   (DRAW|GREEN)\0
        .WORD   0\0
        .WORD   (DRAW|RED)\0
        .WORD   34.\94.
        .WORD   (DRAW|WHITE)\0
        .WORD   50.\87.
        .WORD   (DRAW|PINK)\0
        .WORD   0\0
        .WORD   (DRAW|YELLOW)\0
        .WORD   64.\77.
        .WORD   (DRAW|BLUE)\0
        .WORD   77.\64.
        .WORD   (DRAW|WHITE)\0
        .WORD   0\0
        .WORD   (DRAW|GREEN)\0
        .WORD   87.\50.
        .WORD   (DRAW|RED)\0
        .WORD   94.\34.
        .WORD   (DRAW|PINK)\0
        .WORD   0\0
        .WORD   (DRAW|WHITE)\0
        .WORD   98.\17.
        .WORD   (DRAW|BLUE)\0
        .WORD   100.\0
        .WORD   (DRAW|YELLOW)\0
        .WORD   0\0
        .WORD   (DRAW|WHITE)\0
        .WORD   98.\-17.
        .WORD   (DRAW|GREEN)\0
        .WORD   94.\-34.
        .WORD   (DRAW|RED)\0
        .WORD   0\0
        .WORD   (DRAW|GREEN)\0
        .WORD   87.\-50.
        .WORD   (DRAW|YELLOW)\0
        .WORD   77.\-64.
        .WORD   (DRAW|PINK)\0
        .WORD   0\0
        .WORD   (DRAW|RED)\0
        .WORD   64.\-77.
        .WORD   (DRAW|WHITE)\0
        .WORD   50.\-87.
```

```
.WORD    (DRAW|RED)\0
.WORD    0\0
.WORD    (DRAW|GREEN)\0
.WORD    34.\-94.
.WORD    (DRAW|YELLOW)\0
.WORD    17.\-98.
.WORD    (DRAW|WHITE)\0
.WORD    0\0
.WORD    (DRAW|BLUE)\0
.WORD    0.\-100.
.WORD    (DRAW|GREEN)\0
.WORD    -17.\-98.
.WORD    (DRAW|RED)\0
.WORD    0\0
.WORD    (DRAW|WHITE)\0
.WORD    -34.\-94.
.WORD    (DRAW|YELLOW)\0
.WORD    -50.\-87.
.WORD    (DRAW|GREEN)\0
.WORD    0\0
.WORD    (DRAW|PINK)\0
.WORD    -64.\-77.
.WORD    (DRAW|WHITE)\0
.WORD    -77.\-64.
.WORD    (DRAW|RED)\0
.WORD    0\0
.WORD    (DRAW|BLUE)\0
.WORD    -87.\-50.
.WORD    (DRAW|WHITE)\0
.WORD    -94.\-34.
.WORD    (DRAW|GREEN)\0
.WORD    0\0
.WORD    (DRAW|RED)\0
.WORD    -98.\-17.
.WORD    (DRAW|PINK)\0
.WORD    -100.\0
.WORD    (DRAW|WHITE)\0
.WORD    0\0
.WORD    (DRAW|YELLOW)\0
.WORD    -98.\17.
.WORD    (DRAW|GREEN)\0
.WORD    -94.\34.
.WORD    (DRAW|WHITE)\0
.WORD    0.\0.
.WORD    (DRAW|PINK)\0
.WORD    -87.\50.
.WORD    (DRAW|YELLOW)\0
.WORD    -77.\64.
.WORD    (DRAW|GREEN)\0
.WORD    0\0
.WORD    (DRAW|RED)\0
.WORD    -64.\77.
.WORD    (DRAW|WHITE)\0
.WORD    -50.\87.
.WORD    (DRAW|YELLOW)\0
.WORD    0\0
.WORD    (DRAW|PINK)\0
.WORD    -34.\94.
.WORD    (DRAW|GREEN)\0
.WORD    -17.\98.
.WORD    (DRAW|WHITE)\0
```

```
        .WORD   0\0
        .WORD   (DRAW!RED)\0
        .WORD   0.\100.
        .WORD   (DRAW!GREEN!EOL)\0
VL2:    .BLKW   120.,0
VL4:    .BLKW   120.,0
VL6:    .BLKW   120.,0
REG1:   .WORD   0\0
REG2:   .WORD   0\0
REG3:   .WORD   0\0
REGLIST:
        .WORD   0\1
        .WORD   0\0
        .WORD   410.\499.
        .WORD   (BLACK!EOL)\0
;
;       ATTRIBUTE TABLES FOR CHARACTERS
;
ATTAD1: .WORD   0
        .WORD   0\0
        .WORD   0\0
        .WORD   0\100000
        .WORD   100\10200               ;START ADDRESS OF FONT TABLE
        .WORD   0                       ;CHAR WIDTH\HEIGHT

;
;       SHADE TABLE
;
SHADE:  .WORD   0\0
        .WORD   0\377           ;RED
        .WORD   0\31714
        .WORD   0\63231
        .WORD   0\114546
        .WORD   0\146063
        .WORD   0\177400        ;GREEN
        .WORD   63\146000
        .WORD   146\114400
        .WORD   231\63000
        .WORD   314\31400
        .WORD   314\0           ;BLUE
        .WORD   314\63
        .WORD   231\146
        .WORD   146\231
        .WORD   63\314
        .WORD   0\377
        .BLKW   239.,0
;
;
;       FRAME COUNTER MEMORY LOCATION
;
COUNT:  .WORD   0\0
        .END
```

References

1. Montoya, R. Jorge; England, J. Nick; Hatfield, Jack J.; and Rajala, Sarah A.: An Advanced Programmable/ Reconfigurable Color Graphics Display System for Crew Station Technology Research. *A Collection of Technical Papers—4th AIAA/IEEE Digital Avionics Systems Conference,* Nov. 1981, pp. 486–498. (Available as AIAA-81-2314.)
2. *IDL2 Reference Manual.* Ikonas Graphics System, Inc., Subsidiary of ADAGE, Inc., Mar. 1983.
3. *The ADAGE AGG4 Programming Reference Manual, Rev. A.* ADAGE, Inc., July 1984.

Table 1. Character Draw Times for New Microcode

Character size, pixels	Character draw time, μsec, with erase on		Character draw time, μsec, with erase off	
	Character on 32-pixel boundary	Character offset from 32-pixel boundary	Character on 32-pixel boundary	Character offset from 32-pixel boundary
10×14	46	53	24	34
20×28	147	222	108	195
32×32	217	366	165	295

Table 2. Comparison of Character Draw Times Between New and Original Character Commands Using Autoclear To Erase

New character size	Draw time, μsec, with new character command		Draw time, μsec, with original character command	
	Character on 32-pixel boundary	Character offset from 32-pixel boundary	Original character size	Character at any position
10×14	24	34	10×16	280
20×28	108	195	20×32	680
32×32	165	295	32×32	900

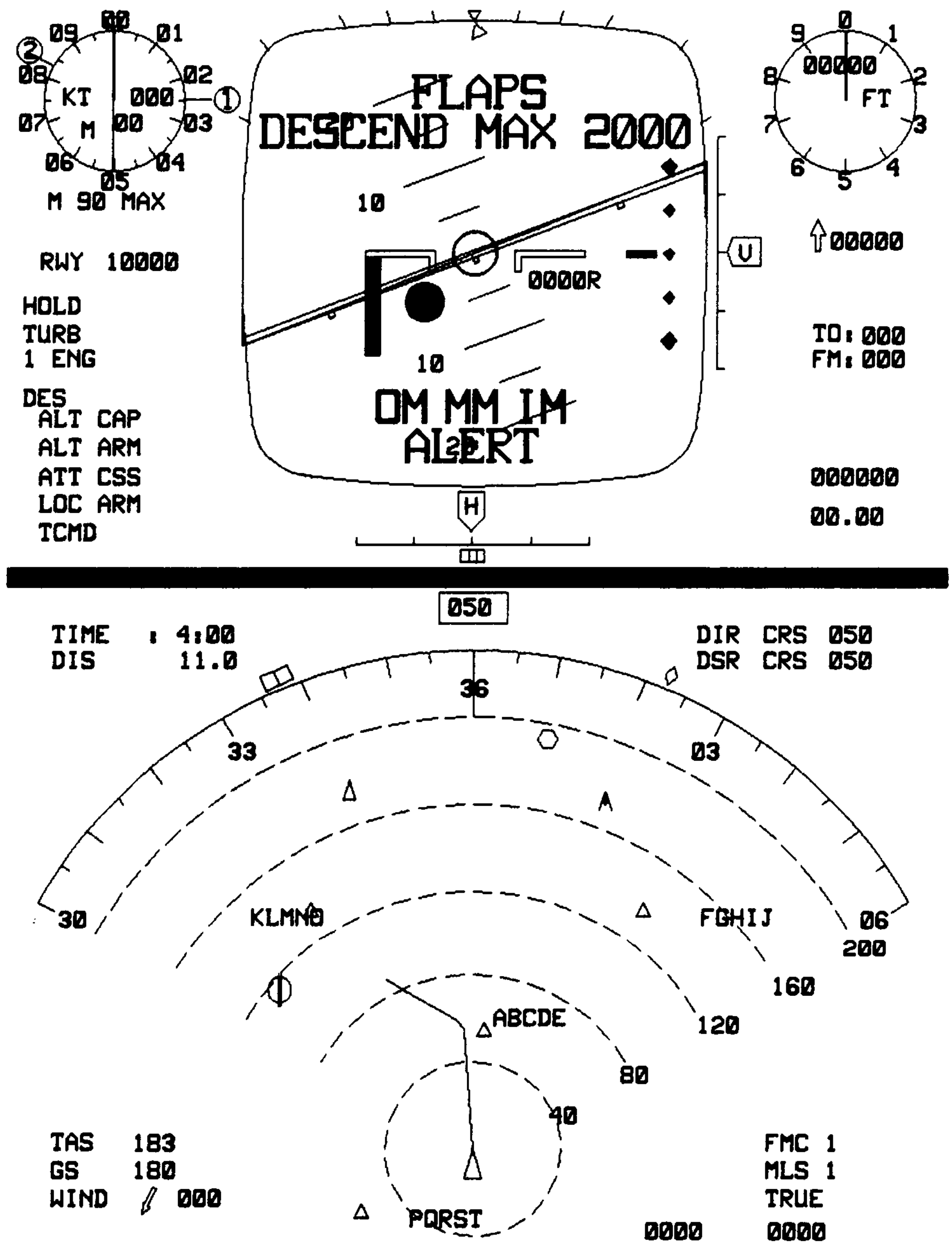

Figure 1. Primary flight display of Advanced Concepts Simulator.

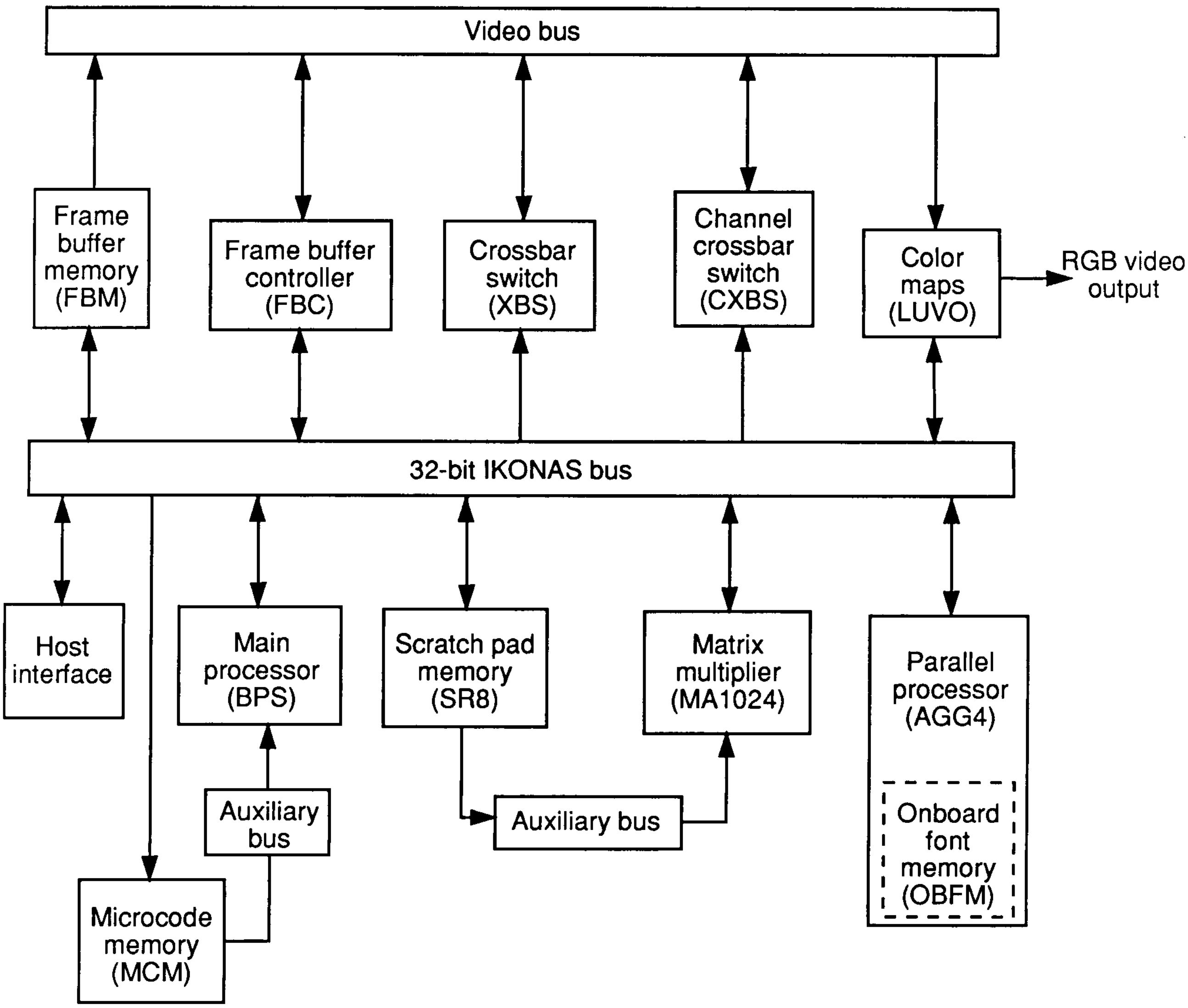

Figure 2. Block diagram of RDS 3000.

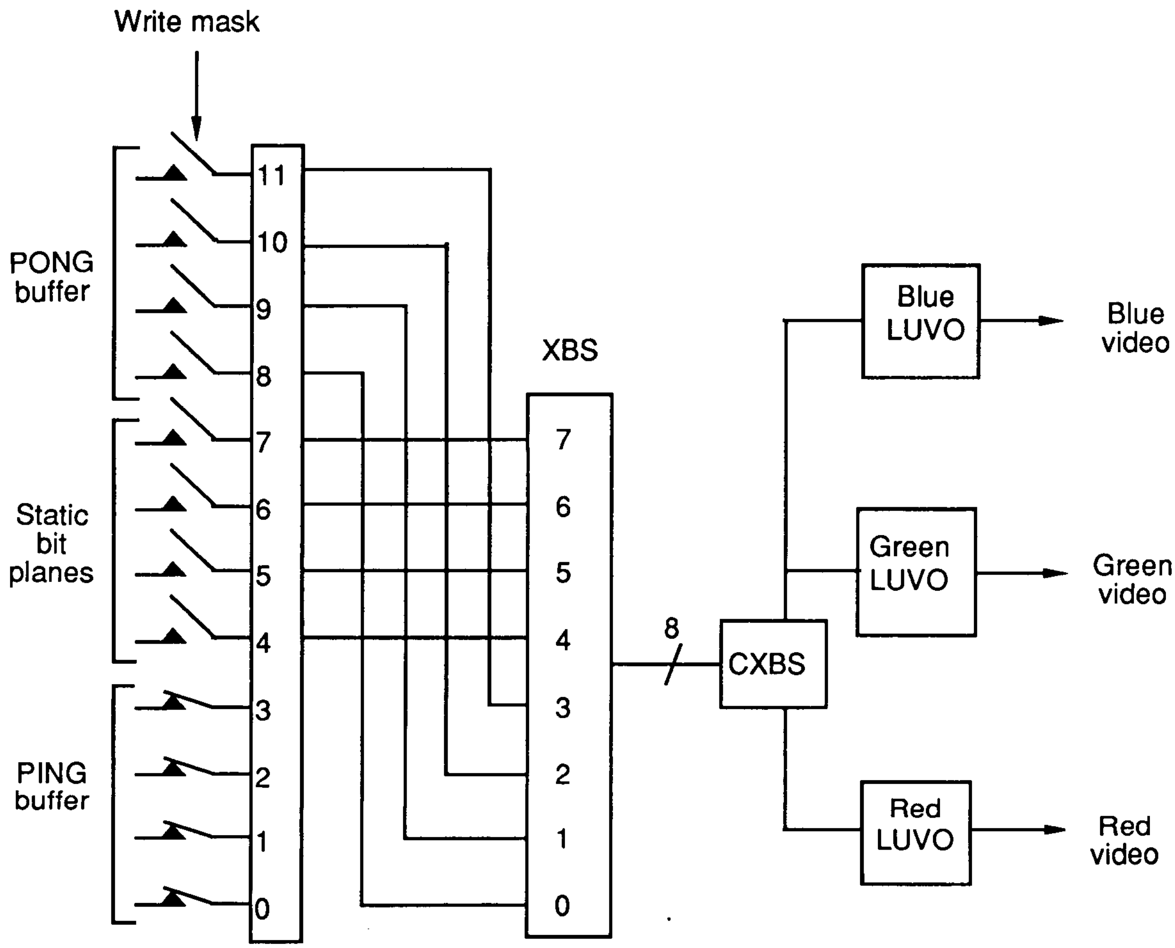

Figure 3. Double-buffering, crossbar switches, and masks.

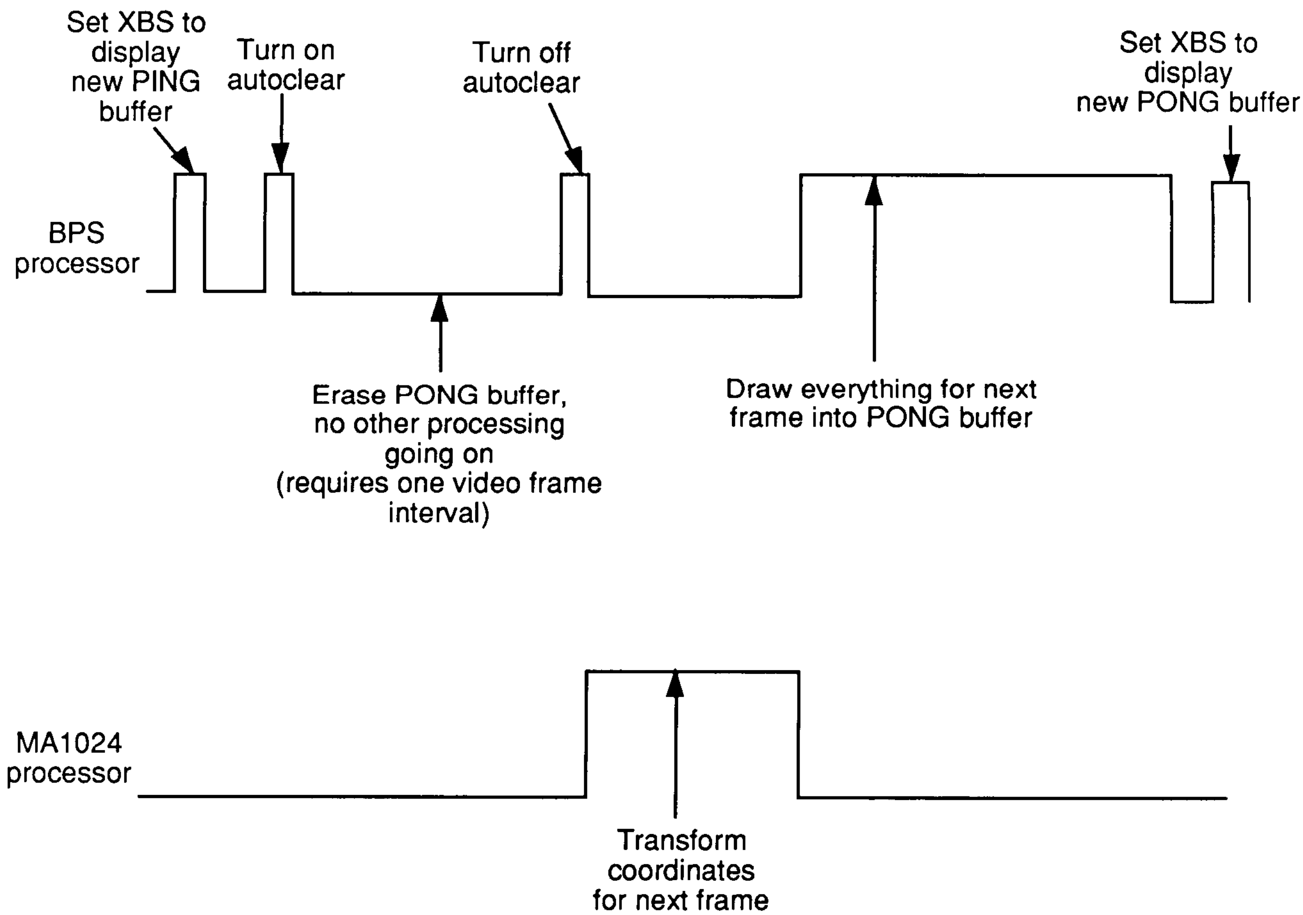

Figure 4. Processor activity during frame update using original sequential code and autoclear.

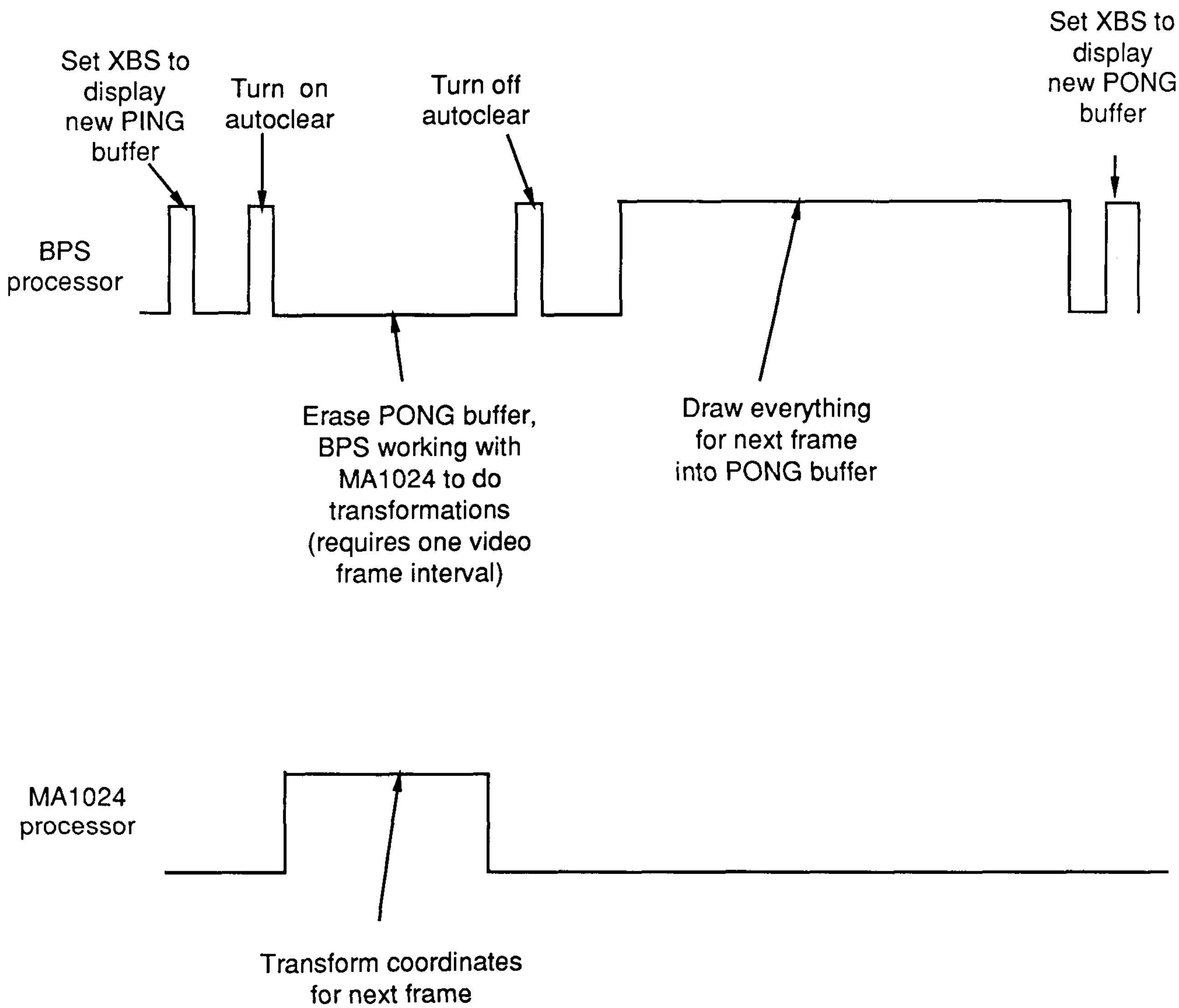

Figure 5. Processor activity during frame update with coordinate transformations computed during autoclear interval.

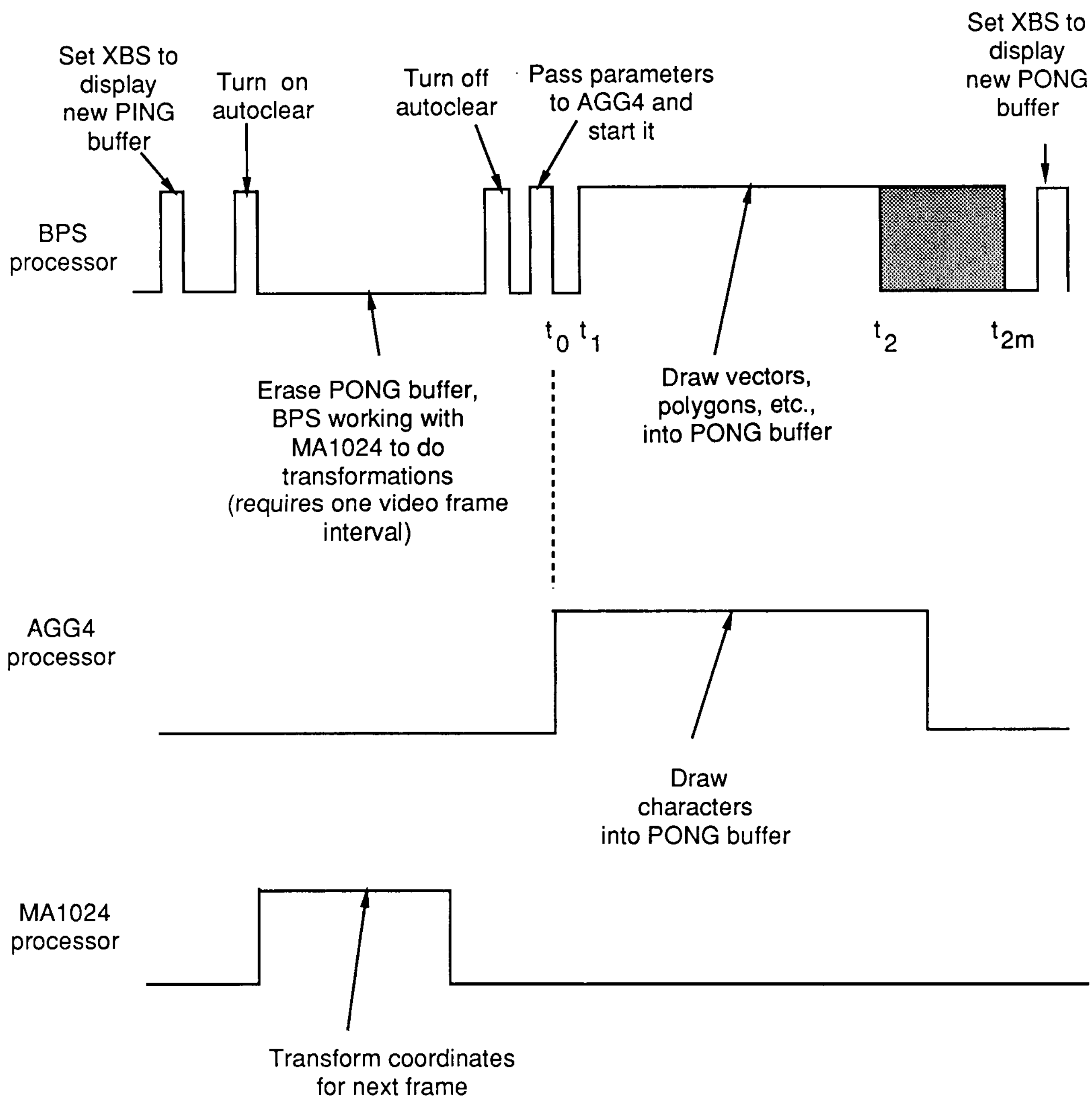

Figure 6. Processor activity during frame update with coordinate transformations computed during autoclear interval and AGG4 drawing characters.

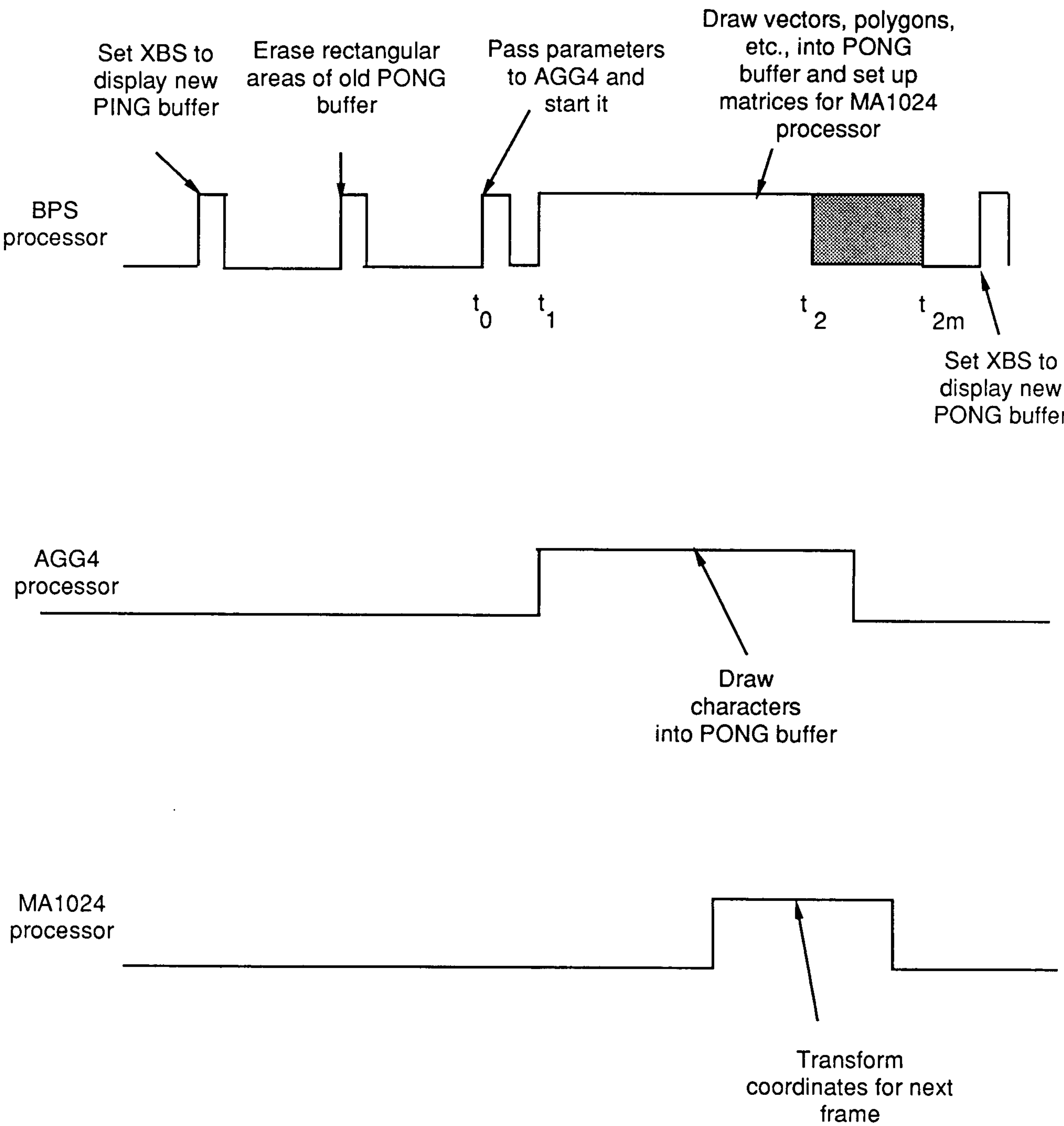

Figure 7. Processor activity during frame update using selected area erase, with coordinate transformations for next frame computed simultaneously with character generation by AGG4.

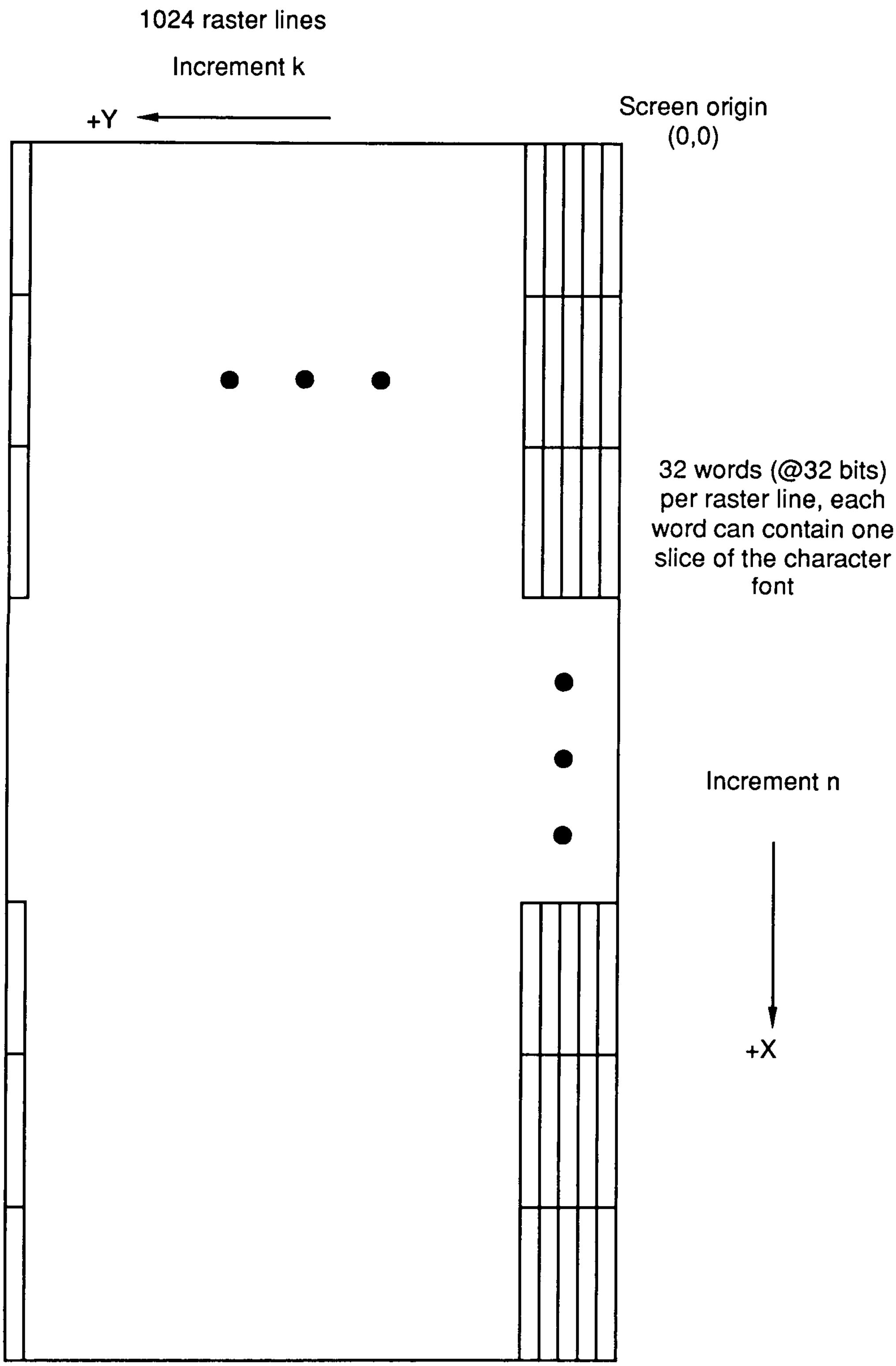

Figure 8. Frame buffer memory mapping.

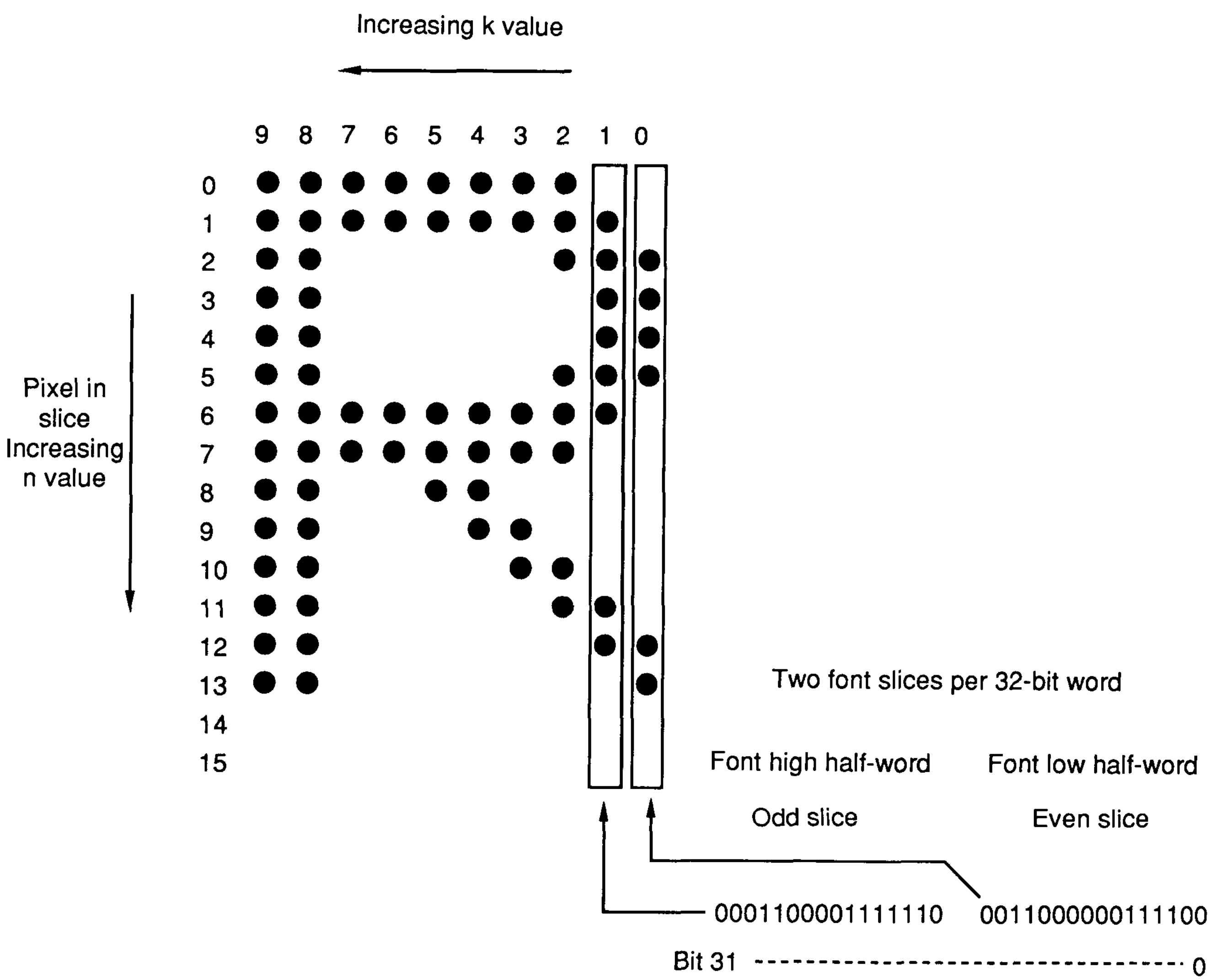

Figure 9. Bit-mapped character generation.

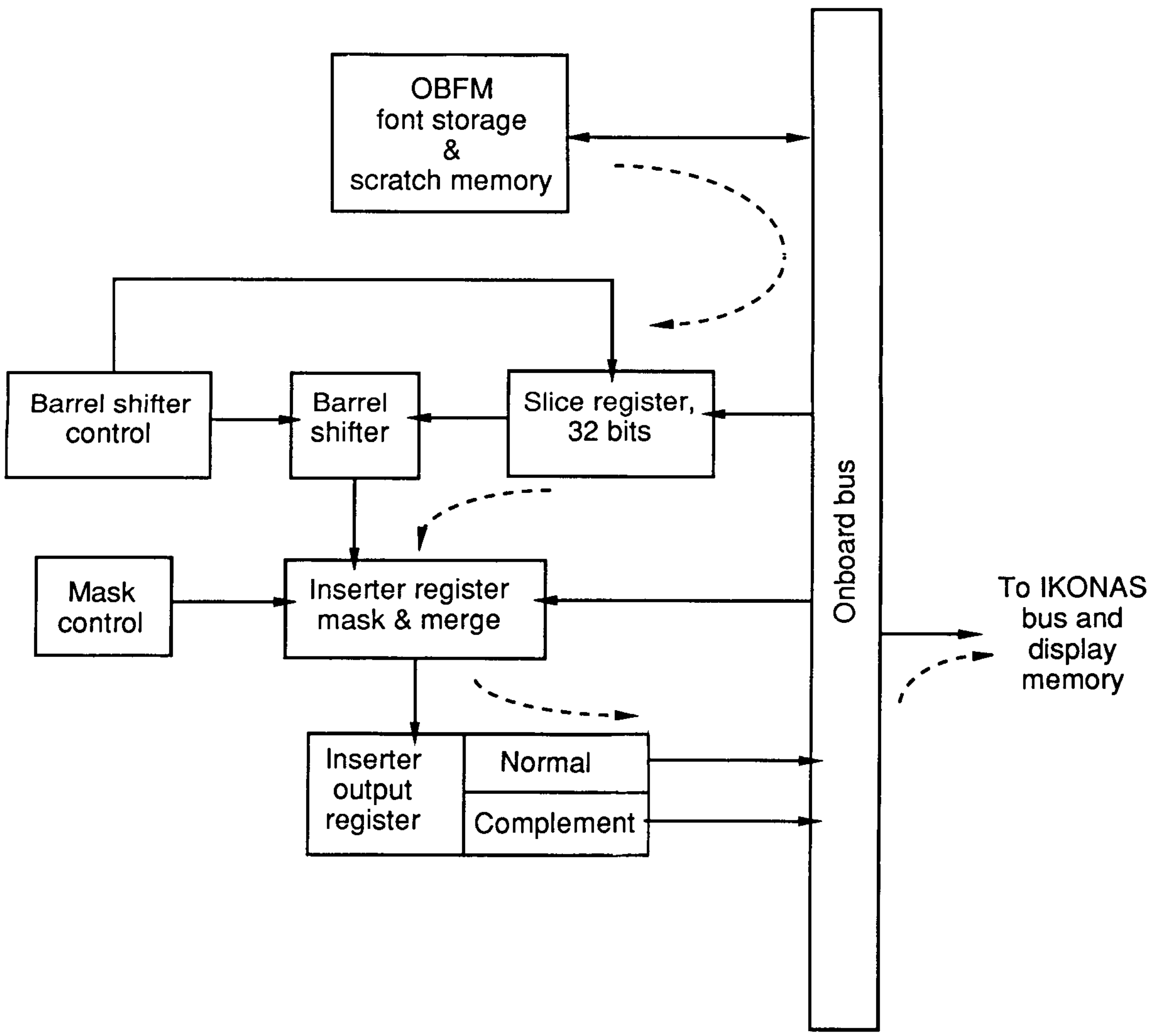

Figure 10. Barrel shifter and inserter output register.

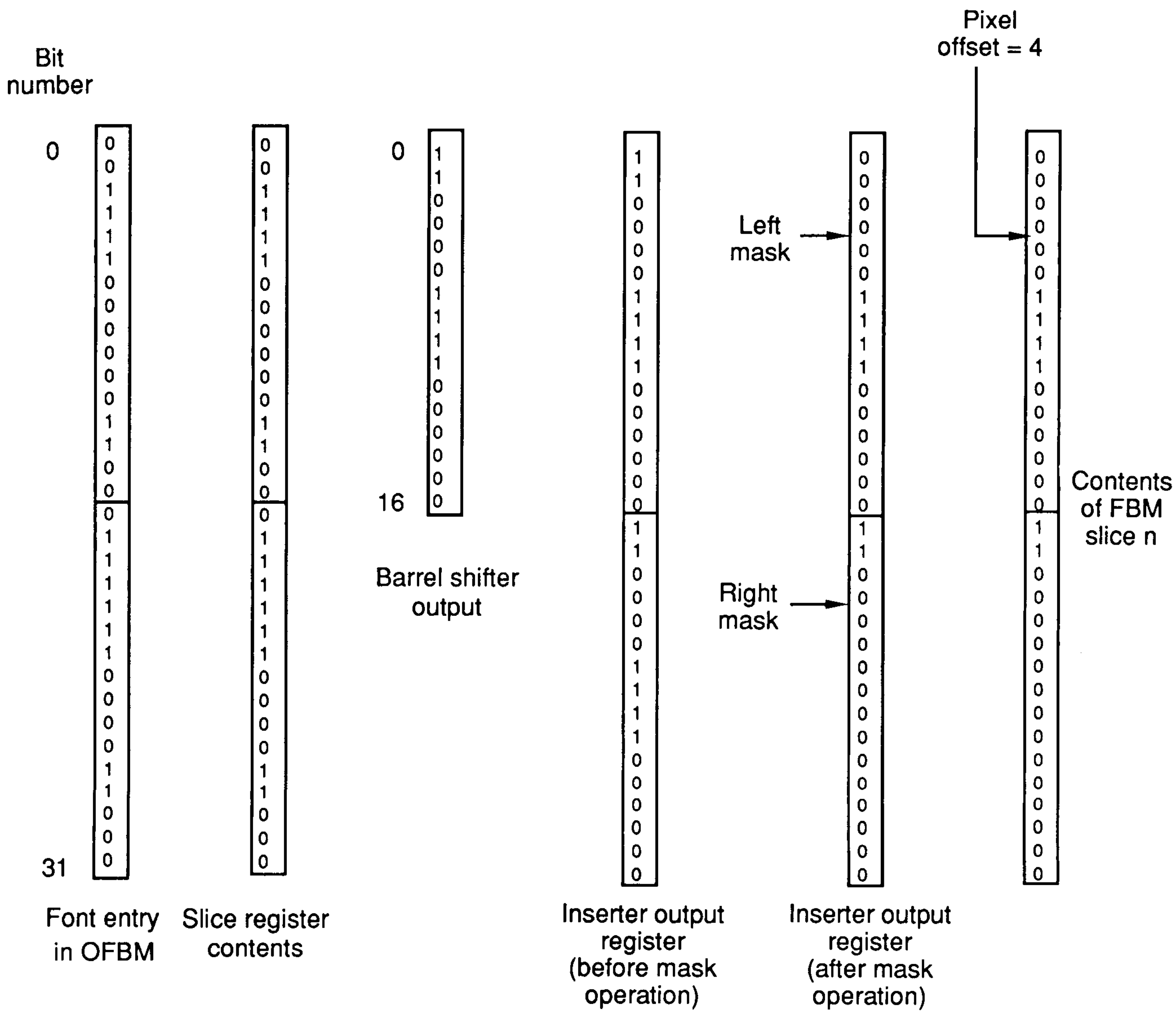

Figure 11. Data flow through the barrel shifter with offset of 4 bits.

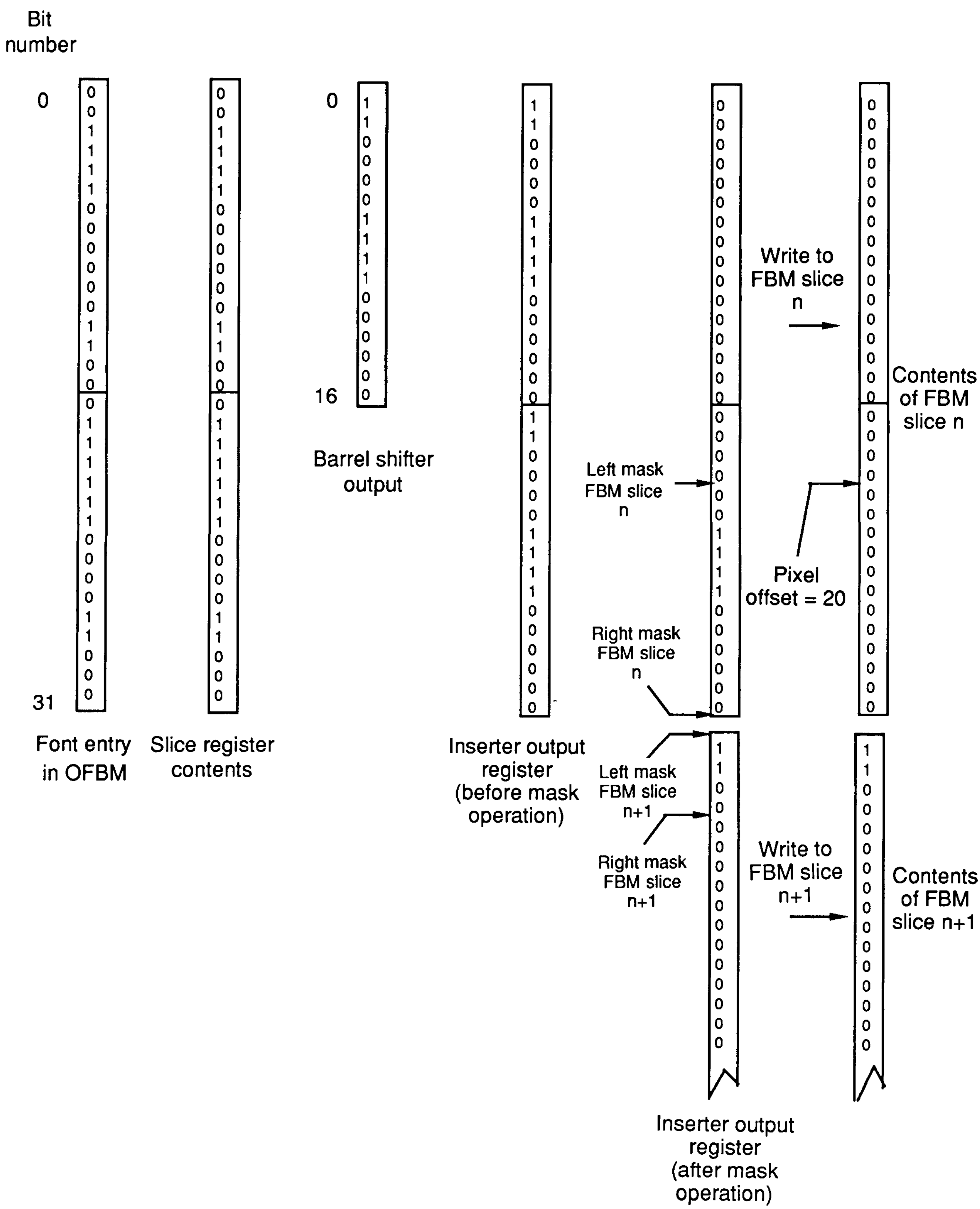

Figure 12. Data flow through the barrel shifter with offset of 20 bits.

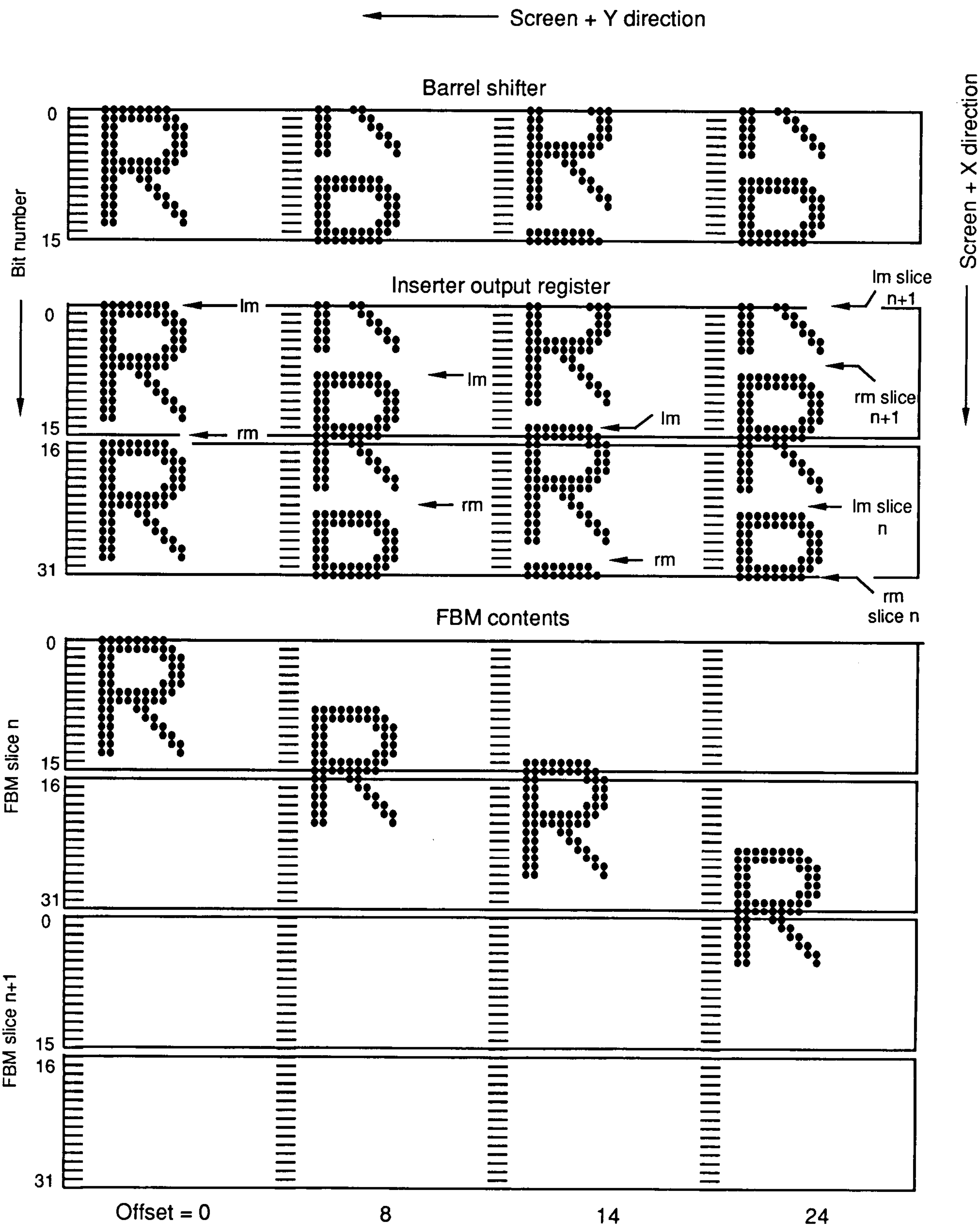

Figure 13. Font bit manipulation for several FBM offsets. Locations of inserter output register left masks (lm) and right masks (rm) are also shown.

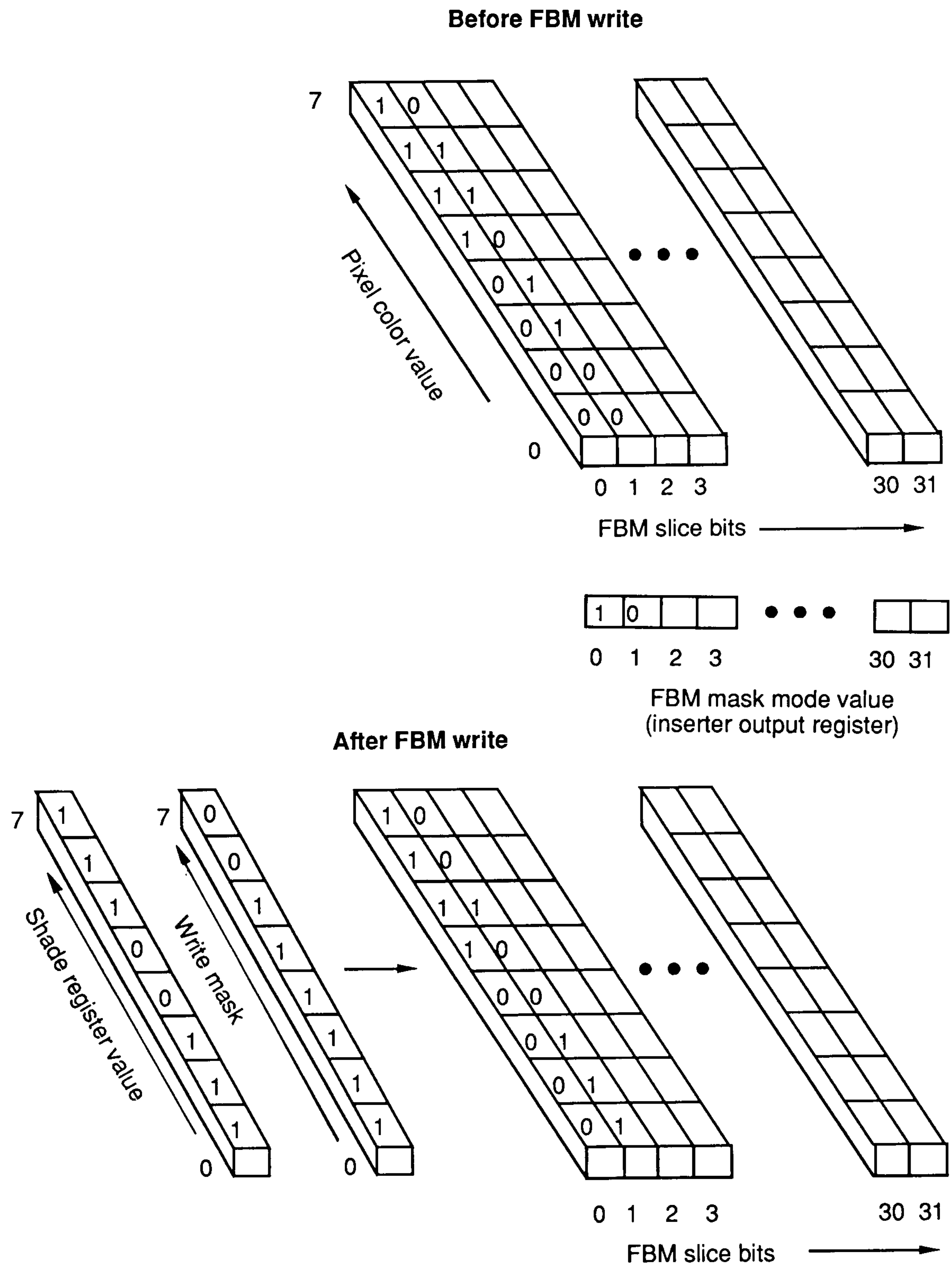

Figure 14. FBM writes for first two pixel locations.

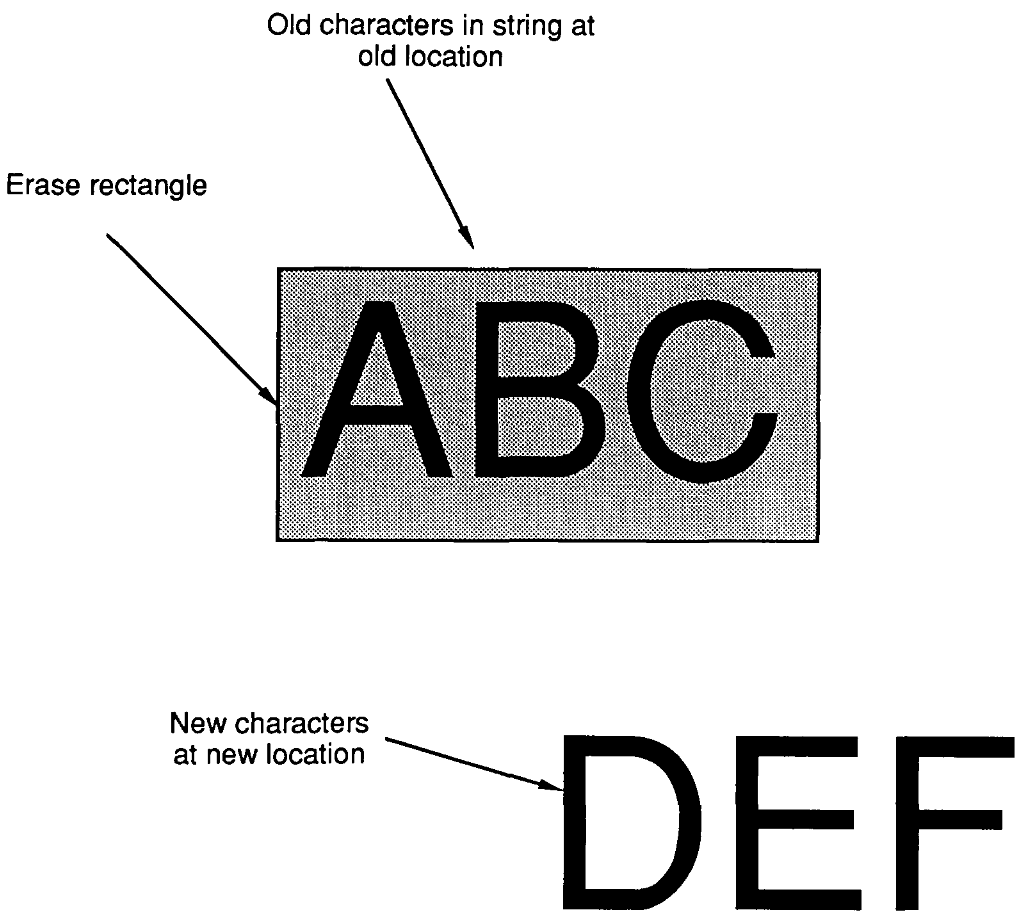

Figure 15. Character erase and draw.

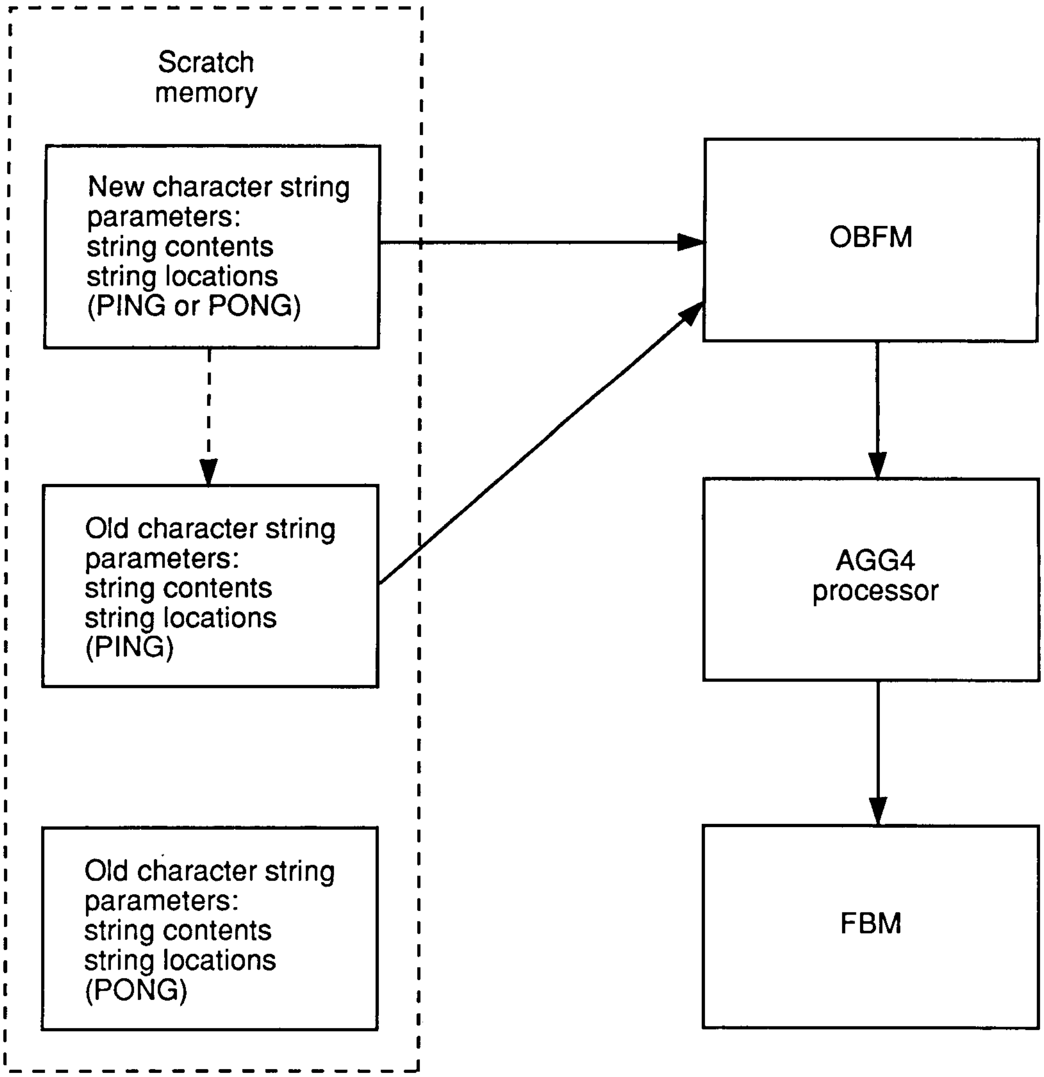

Figure 16. Data transfer.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-4095		

4. Title and Subtitle High-Speed Real-Time Animated Displays on the ADAGE® RDS 3000 Raster Graphics System	5. Report Date April 1989
	6. Performing Organization Code
7. Author(s) William M. Kahlbaum, Jr., and Katrina L. Ownbey	8. Performing Organization Report No. L-16504
9. Performing Organization Name and Address NASA Langley Research Center Hampton, VA 23665-5225	10. Work Unit No. 505-66-41-05
	11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, DC 20546-0001	13. Type of Report and Period Covered Technical Memorandum
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This paper describes techniques to increase the animation update rate of real-time computer raster graphics displays. They were developed on the ADAGE® RDS 3000 graphics system in support of the Advanced Concepts Simulator at the NASA Langley Research Center. These techniques involve the use of a special purpose parallel processor for high-speed character generation. The parallel processor includes the barrel shifter, which is a part of the hardware and is the key to the high-speed character rendition. The final result of this total effort was a fourfold increase in the update rate of an existing primary flight display from 4 to 16 frames per second.

17. Key Words (Suggested by Authors(s))	18. Distribution Statement
Computer graphics Computer animation Flight simulation	Unclassified—Unlimited Subject Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	45	A03

NASA FORM 1626 OCT 86 NASA-Langley, 1989

For sale by the National Technical Information Service, Springfield, Virginia 22161-2171