A Display Architecture for Driving Two Different BitMapped Displays From One Frame Buffer

Robert P. Colwell
Dept. of Electrical and Computer Engineering
Carnegie-Mellon University
Pittsburgh, Pa. 15213

Perq Systems Corporation
2600 Liberty Ave
Pittsburgh, Pa. 15230

Abstract

There are two styles of bitmapped-display personal workstation system architectures, those with the display frame buffers in main memory, and those with separate frame buffer memories. Machines with frame buffers resident in main memory (frame-buffer resident) must provide enough memory, and large enough memory bandwidth, to meet the demands of the displays as well as to maintain satisfactory CPU performance. A workstation with two independent bitmapped displays, one color and one black-and-white, appears to offer many advantages to users performing engineering or design tasks. Although the cost/performance tradeoffs of these two design styles have not been quantified, this paper will show that the cost and complexity of a frame-buffer resident machine simultaneously driving two independent bitmapped displays is substantial. A prototype of such a system has been built and tested. This paper discusses the cost and performance implications of the frame-buffer-resident design style in some detail.

1. Introduction

The current popularity of personal workstations utilizing bit-mapped raster-scan displays is testimony to both the usefulness of these displays as well as their relative economy of implementation. First-generation workstations such as the Perq Systems Corporation Perq 1 [3], Xerox's Alto [7], Apollo's Domain [4], and Sun Microsystem's Sun workstation [6], all provide a bitmapped high-resolution black-and-white display as the primary output device, and in each case the display influences system design in significant ways. For instance, to manage the screen constructively, a pointing device (such as a mouse) is required and often special hardware mechanisms such as RasterOp [5] are also provided. More importantly, a bitmapped display is sufficiently difficult to design into a system that it can have a profound influence on the overall system architecture [8].

The Perq, the Alto, and the more recent Syte [2] workstations refresh their displays directly from main memory (frame-buffer resident, or FBR), while the Apollo and Sun machines provide separate frame buffer memories for screen refresh. This architectural decision is of singular importance and has profound implications on such system aspects as performance, cost, expandability, and operating system design. A thorough discussion of the ramifications of this choice would be of great value, but that topic is too large to be adequately dealt with here. In this paper it is assumed that all frame buffers must reside in main memory, maximizing compatibility of the new design with the existing Perq CPU, RasterOp, and operating system.

The basic design goal of the dual-display Perq was to have a machine with two bit-mapped displays: a standard portrait B&W display (768 x 1024 pixels, 1 bit/pixel, 60 Hz noninterlaced), and a high-resolution color display (1280 x 1024 pixels, 8 bits/pixel, 60 Hz interlaced refresh[1]). This configuration appears to offer many advantages to workstation users performing engineering or design tasks, for the B&W screen is naturally suited to text and command-level interactions, while the color screen can maintain a constant image of the design in progress [1].

From these and other constraints this paper will show how a feasible system architecture was derived. We conclude that FBR systems present a formidable challenge to the system architect in terms of memory bandwidth, the cost of an FBR system capable of driving two high-resolution bitmapped displays is high, and that the primary problem is probably one of future expandability.

2. Design Constraints

To understand the system design implications of FBR machines we will review the requirements of raster-scan bitmapped displays. Typical of most workstation displays, a Perq B&W portrait monitor displays one frame, 1024 lines of 768 pixels each, 60 times each second. A display frame consists of the visible portion plus horizontal retraces (one per line) and vertical retrace (one per frame). Retrace times are specified by the manufacturers of the cathode ray tube drive electronics. Equation 1 describes the interaction of all of these parameters [8].

$$t_f = n_l(n_p t_p + t_{hrt}) + t_{vrt} \quad (1)$$

t_f : time per frame (1/refresh rate)

[1] Interlacing is a technique whereby a display frame (the set of all visible pixels) is split into two "fields": one field consists of all even lines, and the other consists of the odd lines. A given refresh scan from the top of the display to the bottom will refresh one of the fields, and the next scan will refresh the other.

n_l : number of visible lines
n_p : number of pixels per line
t_{vrt} : time per vertical retrace
t_p : time per pixel
t_{hrt} : time per horizontal retrace

For the Perq B&W portrait monitor, t_{hrt} = 3.76 μS, t_{vrt} = 672 μS, so t_p = 15.4 nS. This implies that, in order to keep the screen stable, a standard Perq memory system must provide at least one bit of video data every 15.4 nS (a pixel rate of 64.788 MHz). To ensure this, the Perq main memory is dual-cycled: every other 340 nS memory cycle is dedicated to the display. Remaining memory cycles are available for the CPU or DMA devices. The widest data word available per fetch is 64 bits, and since 680 nS/64 bits = 10.625 nS/bit, enough memory bandwidth exists to keep the screen stable.

The memory bandwidth required for the color system described above is much higher. Equation 1 still holds, but now n_l = 1024/2 (due to interlacing), n_p = 1280, t_{hrt} = 7 μS, and t_{vrt} = 700 μS[2], so the time per pixel, t_p, is 18.9 nS (a pixel rate of 53 MHz). However, the color screen requires eight bits for each pixel, not just one bit as for the B&W screen.

Since the color and B&W screens are to have the same number of lines, and the color screen is interlaced, two B&W lines will be refreshed for every color line. A B&W line takes 768 pixels×15.4 nS/pixel + 3.76 μS = 15.62 μS/line. So a color line will take 2×15.62μS = 31.24 μS.

What do these numbers imply about FBR memory architecture? It is already obvious that 64 bits/fetch is insufficient to drive the color screen. The next choice, 128 bits/fetch, is also too small: 680 nS/128 bits = 5.3 nS/bit, so a pixel's worth of data will take 5.3 nS × 8 = 42 nS to collect. Since color pixels are only 18.9 nS

[2] These specifications are for the Hitachi HM3619 color monitor.

long that is too slow. But, as will be seen, 256 bits/fetch is sufficient not only to drive the color screen but the B&W simultaneously.

256 bits is 32 color pixels, so the number of memory fetches required per color line is (1280 pixels/line) / (32 pixels/fetch) = 40 memory fetches. The B&W screen requires 768/256 = 3 fetches/B&W line, or 6 fetches per color line. So the total number of memory fetches to refresh both screens is 46. At this point, two observations can be made.

1. There is not enough time to fetch color data "on the fly". 680 nS/32 pixels per fetch = 21.25 nS/pixel. Since the color pixel period is 18.9 nS, we would fall behind a little more on each fetch.
2. But there are enough memory fetches per line to get all the data out of memory for both B&W and color displays. 46 fetches × 680 nS/fetch = 31.28 μS: exactly enough memory fetches to retrieve all the video data needed for both screens.

This implies that some prefetching is required (hence temporary storage is necessary), but also shows that 256 bits/fetch is a high enough bandwidth to meet the requirements. Another implication is that B&W cursor data, color cursor data, and colormap data cannot be fetched during frame refresh since all memory accesses, including those occurring during horizontal refresh, are being used to keep up with the displays. As a consequence, temporary buffers must be provided for the cursors, which are loaded during vertical retrace, and unloaded as required during visible scan. The color map is also reloaded during vertical retrace.

Another important result of the design specification is the amount of memory required. A Perq portrait B&W screen requires 768 × 1024 / 8 = 100K bytes of memory. The color display requires 1280 × 1024 = 1.31 MBytes, so the frame buffers alone require 1.4 MBytes. Since 256 bits/fetch are needed, and dynamic RAMs are one bit wide, at least 256 RAMs must be provided in the memory system (not counting parity RAMs). 256 64K RAMs are 2 MBytes, so that is the amount installed in the prototype, although at least twice as much is needed to make system performance competitive.

Note that the memory bandwidth required for the color display is so much higher than the B&W that the color frame buffer design would have been the same even if the B&W display had not been included. This is because the design constraint that the color frame buffer reside in main memory forces a quantization of memory bandwidth that leaves some of the memory cycles unused by the color display. Given this design constraint, the additional cost of supporting a B&W screen beside the color display is almost nil. On the other hand, if the B&W screen were of higher resolution, say 1280 pixels per line instead of 768, the basic drawback of the FBR style is strongly manifested: the overall memory bandwidth would probably have to be at least doubled, so that more bits per lookup would be available.

Fetching 256 bits for every memory read necessitates some additional memory data registers for CPU accesses. A decoder enables the appropriate registers onto a bus to the CPU based on the width of the CPU's requested access (16-bit word, double word, or quad word read) and the value of two low-order address bits. Since the memory is never written by the video display system, but only by the CPU, the data path into the memory needs to be only 64 bits wide, not 256.

On the other hand, there are some good reasons why the memory write data path ought to be as wide as possible. Writing many contiguous pixels to the same value is a common operation for bitmapped displays (e.g., clearing the screen).

It would be trivial to include a new finite-state

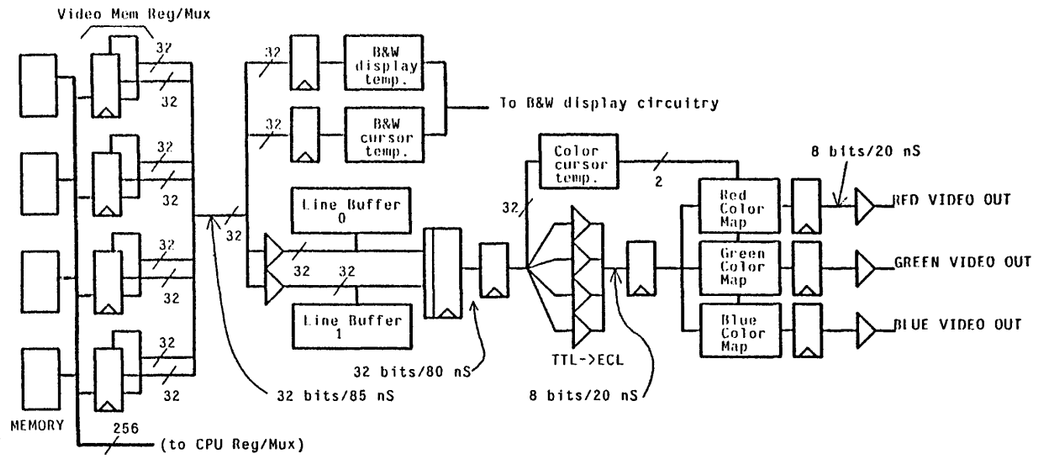

Figure 2-1: System Block Diagram

machine (FSM), controlled by the CPU, which could load up a 256-bit register with the desired pattern and then write large blocks of memory with that pattern. In an FBR system this same mechanism could be used to initialize or clear out user memory such as data arrays very inexpensively. Such a mechanism was not incorporated in this prototype.

3. Architecture

Figure 2-1 shows the overall display system architecture for the dual-display Perq.

The 256 bits retrieved per video fetch are multiplexed, 32 bits at a time, onto a synchronous backplane bus running at 85 nS (twice the CPU clock rate). This data can be B&W video, color video, cursor data, or colormap data (as discussed earlier, during visible refresh this data will either be B&W video or color video). For color video data, two operations must be performed: the data must be stored temporarily in FIFO order, and it must be read out of temporary storage at a different clock rate than that used in loading. The reason is that the color display pixel rate is fixed by physical constants such as retrace times, number of visible lines, and refresh rates, while the CPU and memory clocks are related to the B&W display rates determined when the Perq 1 was designed. Thus the timebases of these two subsystems are inherently unequal and asynchronous. The line buffers in Figure 2-1 perform both of these required functions.

3.1. Line Buffers and Queueing

The line buffers are small (1K x 32) memories, sufficient to store enough color pixels for one display line. While one line buffer is being loaded with the next line's pixel values, the current line is being read out of the other buffer at a multiple of the color pixel clock (18.9 nS/pixel × 4 is approximately 80 nS).

Even if this color system memory were not required to reside in main memory, queueing would still be necessary for transfers of data between main memory and the frame buffer. The FBR architecture makes this queueing much more difficult, since the screen refresh depends on the timing of the transfer.

Once the price has been paid for this complexity, however, a useful function becomes feasible: real-time conversion from one display

format to another. A brief description of the mechanism for performing this conversion follows. A PAL-based finite-state-machine generates addresses for the static RAM arrays in the line buffers. These addresses are not necessarily sequential, nor are they necessarily the same for each RAM in the array for a given access. In fact, by suitably scrambling these addresses upon loading, and by using a 32-bit barrel shifter in the datapath, the video data can be converted in real time from one format to another.

For shaded images, the most natural data format is a packed representation, where a 16-bit word in memory contains two adjacent 8-bit display pixels. This format is preferred when a program manipulating the color image is operating on groups of contiguous pixels, such as a rectangular region. It is also the easiest format to use in a machine which shares the main memory between the CPU and the displays, since the CPU is fetching its instructions from this same memory, and instruction opcodes are designed for contiguous bit encodings.

The bit-plane format is another popular representation. In this format, a 16-bit memory word constitutes one bit each of 16 consecutive display pixels, and each plane must be fetched and their data combined to form the pixels displayed. The bit-plane format is preferred in applications where the display image is organized as planes stacked parallel to the screen's surface, or where hardware constrains the design. For images which easily separate into planes, the bit-plane format can offer significant speed improvements to image manipulations, since the image is then likely to be operated on a plane at a time. RasterOp is already optimized to operate on this type of data.

Since the optimum data format depends on the application, real-time conversion can make the workstation much more flexible. It is important to realize, however, that this conversion requires significant prefetching in order to work. A single video memory fetch yields 256 bits, which in bit-plane format belong to 256 adjacent pixels. Thus, eight separate memory fetches must transpire before enough data has been collected to display even one pixel. The design discussed here prefetches an entire color line, so enough time is available to construct the pixels.

Note that this data conversion mechanism is not restricted to use in FBR systems. If it is included in FBR systems it can be done in real time with only a small amount of additional complexity, but non-FBR systems can provide this operation in their video data queueing or as a co-processor.

When a line buffer is loading the next display line's data, it is clocked at a rate synchronized to the memory system's clock. At the same time, the other line buffer is unloading the current display data at a rate synchronized to the color display rate. This is accomplished by clocking the line buffer controllers through multiplexers which are switched only during a (rare) otherwise unused memory cycle, once per color display line.

A pipeline multiplexer/register selects the appropriate line buffer to read, and its outputs traverse the backplane and are clocked into a register. This data stream is used to load the color cursor during vertical retrace, but normally the data are converted to ECL levels, multiplexed down to a pixel stream of 8 bits arriving every 18.9 nS, and converted through the color map to red, green, and blue output values with fast video digital-to-analog converters.

3.2. The Color Cursor

The color cursor buffer is a temporary store capable of holding a color cursor of size 64 x 64 x 2 bits (8K bits). This buffer is required because the only free memory cycles available to perform color cursor fetches are during vertical retrace, while color cursor data is required at visible display lines for which the cursor is to appear. The need for a temporary store for the cursor is not just due to the FBR design style, however. Cursor information must be available in a high-bandwidth form, since this data is required at the high pixel clock rates.

The FBR style does not appreciably complicate the cursor store controller The FSM controlling the cursor store is implemented in the same way as those controlling the line buffers. The moderately difficult challenge represented by the color cursor is that it must be visibly locatable to within a pixel. Since pixels occur every 18.9 nS, a fast pixel counter is included in the cursor controller, which is loadable via a double buffer by the operating system's mouse-handling code. This pixel counter is loaded during horizontal retrace from the first buffer register. When the underflow condition is reached the FSM is allowed to proceed and color cursor data begins to flow out of the cursor store.

For each pixel located under the cursor, the color cursor buffer supplies two bits of data. These bits are used as address bits <9:8> into the color map. Bits <7:0> are, of course, the pixel value. By convention, when the color cursor bits are 00 (quadrant 0 of the color map) the cursor is not visible. If the user desired a solid red cursor, for example, the color cursor could be loaded with 01 for all 64x64 cursor pixels, and quadrant 1 of the color map written to red for all 256 locations of that quadrant. Thus, no matter what the pixel value is for pixels under the cursor, they would still be mapped into red in the color map.

For suitable choice of color cursor data bits, any 3-color pattern can be superimposed on the underlying image. At the cost of real-time computation, the CPU can even compute a function between the display and the cursor overlay, for cursors of arbitrary complexity and coloring. Since the cursor is reloaded during vertical retrace (60 times per second) it is possible to switch cursor patterns or colors dynamically while moving the mouse.

3.3. The Color Map

Using a color lookup table to map a few bits of pixel data into 24 bits of color data (8 bits each for red, green, and blue) is a standard technique in color display systems, and is employed here. Since the color map must be fast enough to keep up with the pixel stream of 18.9 nS, it is implemented in ECL. For the same reasons as before, the map is reloaded only during vertical retrace, so it can be reloaded as often as 60 times per second.

3.4. The Bitslice Video State Machine

A bitslice microprogrammed controller generates video addresses for both B&W and color displays, both cursors, and the color map, as well as control signals such as blanking, retrace, and line buffer enables. This bitslice machine is known as the Video State Machine, or VSM, and is composed of a 16-bit ALU and a microstore of 1K x 40 bits. The ALU only needs to be 16 bits because the Perq generates 20-bit physical addresses, and video fetches only occur on 256-bit boundaries. The VSM is programmed in micro-assembly code. Since the VSM is clocked at the same rate as the Perq CPU (170 nS), the control bits are only available at time increments of 170 nS. This necessitates a non-trivial amount of local queueing for time-critical signals such as blanking or retrace. A high speed phase-locked-loop generates a color video clock that is synchronized to a multiple of the CPU clock. Control signals that have to be passed from one timebase to the other are constrained to occur only at the points where these different clocks coincide. This also makes local delays for time-critical signals necessary.

This VSM represents a large amount of circuitry and cost, and is provided for several reasons.

Due to the FBR design style, there are multiple sources of addresses into main memory (CPU, B&W display, B&W cursor, color display, color cursor, and color map). All of these accesses must occur in their proper order. In the standard Perq, video addresses are generated by HW counters, with a small FSM governing their memory accesses. The Color Perq has an interlaced screen, so at the end of a display line the color display address must be adjusted to skip the next line's data (which will not be displayed until the next display field is refreshed). The VSM regulates the memory accesses and performs the necessary address manipulations to make the interlacing work correctly.

In the bitplane display mode the VSM generates eight separate addresses, one for each plane. These addresses are kept in the VSM's external register file. In addition, copies of the base address for each plane is kept so that the address counters can be re-initialized once per frame. The VSM interrupts the CPU at the beginning of each vertical retrace, and then enters a quiescent period during which the operating system can alter any of the VSM's registers. This provides the means to switch cursors or the color map very quickly.

The VSM also makes it possible to support other color display screen formats, as long as the required video bandwidth does not exceed that discussed earlier. The reprogrammability of the VSM was of critical importance in debugging the overall system.

4. Conclusions

The major drawback of the frame-buffer resident architecture is that all memory bandwidth requirements in the system are cumulative. To meet the design specification for the FBR Color Perq, the memory had to be designed so that it could fetch 256 bits every 340 nS: a very high memory bandwidth of 753 MBits/second. While meeting that requirement was non-trivial, neither was it an overwhelming obstacle (the cost is approximately 100 integrated circuits system-wide).

The harder challenge was to synchronize the interleaved fetches from B&W and color video systems, and to provide the temporary storage they require. For example, a standard Perq fetches cursor data as required, on a line-by-line basis, so much less temporary storage is needed (only a few FIFO chips). Since memory bandwidth is being saturated here, the entire cursors must be prefetched, so much larger, faster, more expensive, and more capable temporary stores had to be provided.

Perhaps the biggest drawback to an FBR architecture is that it cannot easily accommodate additional demands on memory bandwidth. Adding the color system to the Perq, for example, necessitated a redesign of the Perq's memory boards and a substantial amount of complex clock synchronization. If a new system were proposed, say an array co-processor, the additional memory bandwidth required might have to come out of the CPU's allocation, degrading its performance. (As it is, the CPU must wait to access memory if its requests conflict with a scheduled video fetch, although this loss in CPU performance has not been quantified.)

The original design goal of having RasterOp work unchanged in the color system was met, and the performance loss of having to move eight times more bits for the color display is not as bad as it might seem at first, since all pixels are now byte-aligned and no shifting or masking is ever required. The high degree of architectural compatibility achieved has made software conversion from the B&W Perq to the dual-display Perq quite straightforward.

Although the cumulative memory bandwidth requirements of an FBR system are stringent, they can still be met with existing technology for high performance display systems. Nevertheless, the additional expense and complexity of the temporary storage needed, and especially the problems of future expandability, are serious disadvantages to the frame-buffer resident system design style. Future research should be directed at the performance tradeoffs of this architectural style.

5. Acknowledgements

The substantial contributions of the following individuals to the design presented here is gratefully acknowledged: Dave Stoner, Stan Kriz, Pradeep Reddy, Dave Hile, John Strait, and Brian Rosen.

References

1. Clifford Barney. "Work station integrates design, layout". *Electronics 56*, 11 (May 31 1983).

2. B.E. Hamilton, M.A. Fischer. "A High-Performance Workstation Using a Closely Coupled Architecture". *IEEE Computer Graphics and Applications 4*, 4 (1984), 67.

3. F.R.A. Hopgood, R.W. Witty. "PERQ and Advanced Raster Graphics Workstations". *IEEE Computer Graphics and Applications 2*, 7 (1982).

4. D.L. Nelson, P.J. Leach. "The Architecture and Applications of the Apollo Domain". *IEEE Computer Graphics and Applications 4*, 4 (1984), 58.

5. W.M. Newman, R.F. Sproull. *Principles of Interactive Computer Graphics.* McGraw-Hill, 1979. (chapter on RasterOp).

6. Andy Rappaport. "Workstations feature UNIX-optimized architectures". *EDN* (November 24 1983).

7. D. Siewiorek, G. Bell, A. Newell. *Computer Structures: Principles and Examples.* McGraw-Hill, 1982. (Chapter on the Alto personal computer).

8. Mary C. Whitton. "Memory Design for Raster Graphics Displays". *IEEE Computer Graphics and Applications 4*, 3 (1984), 48.