

1. Introduction

In recent years memory prices have fallen to the point that it is economical to represent text and graphics images using a [bitmap]. A bitmap represents a two-dimensional image by dividing it into a rectangular matrix of pixels, each represented by a fixed number of bits. All bitmaps must have at least one bit per pixel assigned to convey color information. Additional bits may be used for more color information, but may also be used to convey intensity, depth, priority and a broad range of other application dependent information. Because of this flexibility and decreasing cost, raster displays have gained broad popularity. Systems that use more than one bit per pixel are somewhat more complex but are not substantially different in principle than those that use a single bit.

Decreasing memory prices and increasing user demand for display quality have also caused an increase in the resolution of raster displays. This increased resolution has placed significant demands on display hardware. In this article we will examine the origin of these demands and a few cost-effective techniques for meeting them.

2. Raster Display Hardware

A basic raster display system is shown in Figure 1. The heart of this system is the [frame buffer]. The frame buffer is a block of memory with storage assigned for each pixel of the displayed image. This memory is accessed by the [sweeper] and the [graphics processor]. The sweeper accesses the frame buffer periodically to obtain the data necessary to update the display device. The most common raster display device is the cathode ray tube (CRT). Other examples include bit-mapped impact printers and laser printers. Since the horizontal and vertical timing of a raster display system is usually fixed, the sweeper must provide new data at precise intervals. For this reason the sweeper is given priority of access to the frame buffer. On a high resolution system, this decision can have a significant impact on performance. Consider a 1024 X 1024 pixel display that is refreshed by the sweeper at 60 frames per second. The sweeper must obtain 62.9 million pixels per second from the frame buffer!

Figure 1 Basic Raster Display System

The graphics processor also accesses the frame buffer, either to read its current contents or to write new information. The graphics processor must synchronize its requests for access to the frame buffer so that it does not interfere with sweeper access. Failure to perform this synchronization would result in corruption of the displayed image during graphics processor access to the frame buffer. Since the sweeper must always access the frame buffer, that portion of the frame buffer's bandwidth not used by the sweeper is the time allotted to the graphics processor to manipulate graphical data to be displayed. With conventional memory components, this percentage approaches zero as display resolution increases much beyond 1K X 1K. Later we will describe a new memory component called a [video ram] that allows the practical construction of much higher resolution displays without severely impacting the percentage of time available for graphics processor update of the frame buffer.

Raster displays are not the panacea for all graphics applications. Line drawing and point-plotting display systems also serve in several application areas. What is perhaps unique about raster display systems is their ability to effectively emulate other technologies. This

ability, coupled with the relative simplicity of raster-based systems make them an attractive choice for many graphics applications.

3. Using Raster Displays

Raster Displays are used to display and manipulate graphic images and text. Although text is conceptually just another form of graphical data, in practice it is often convenient (and more efficient) to treat text as a distinct entity. Non-text graphic images are composed from one or more primitive elements. The most primitive element is the [point]. A point is a pixel that is displayed with some color and/or intensity. Points are used to compose [lines] and [curves]. Lines are used to compose polylines and polygons (an open or closed sequence of lines). Curves may be simple (e.g., a circle) or complex (e.g., a parametric cubic surface). Lines and curves need not be only a single pixel in width; they may be of arbitrary width. Lines and curves may also be [textured]. A textured line or curve is created by replacing each pixel with a bitmap. Bitmaps are themselves primitive graphic elements. Figure 2 shows textured lines created using a bitmap composed of a circle. Bitmaps may be much more complex, for example, a LANDSAT image.

Figure 2 Some Textured Lines

Closed regions created with other primitive elements may be [filled]. Regions may be filled with a solid color or with a pattern. The pattern is simply another bitmap. By combining these primitive elements virtually any graphical image can be created.

Because text occurs so frequently it is usually treated as a distinct graphical element. A character of text may in reality be just a rectangular bitmap of some size. Characters may also be created from lines (called [stroke] characters). Examples of stroke characters are character strings displayed on pen-plotters or vector displays.

Raster displays customarily use bit-mapped character strings. Characters are stored in a [font]. A font contains the bitmaps of the letters of the alphabet and any special characters that may be displayed. These bit maps are usually densely packed in a safe area of the address space (such as ROM). The study and design of character fonts is an interesting subject in itself. We will confine our discussion of fonts to only their most basic properties.

Characters in a font are stored in a rectangular bitmap sometimes called a cell. Within a particular font, cell sizes may be fixed or variable. If all cells have the same width the font is said to be [monospace]. If cell widths vary according to the character being displayed, the font is said to be [proportionally spaced]. Strings built from proportionally spaced fonts have characters starting at arbitrary pixel boundaries with the line. Monospace fonts usually have characters aligned at particular pixel intervals. If the character cells are word-aligned within the raster (for example, they might correspond to byte boundaries within the frame buffer), we have the simplest (and least flexible) mechanism for text display.

4. Integrating Text and Graphics

Many applications require that text and graphics be displayed simultaneously. Bitmaps are a natural choice for this environment. Even the lowest-priced personal computers now provide bit-mapped displays and some level of support for interactive graphic editing. High quality document preparation systems often have high resolution displays emulating photo-typesetters and laser printers for what-you-see is what-you-get document preparation.

With the exception of single-pixel-width lines, nearly all graphics primitives involve the display, manipulation, and combination of bitmaps. One of the most popular and complete tools for manipulating bitmaps is [BitBlt] or [RasterOp]. BitBlt is a general-purpose abstraction for copying, moving, and combining portions of bitmaps. BitBlt got its name from an instruction called "Bit-Boundary Block Transfer" on the Alto personal computer developed by the Xerox Palo Alto Research Center in the early '70's. Newman and Sproull first coined the term "Raster-OP" in their text on interactive graphics. BitBlt is an integral part of the user interface of the Smalltalk-80 system also developed at XEROX PARC.

BitBlt is general enough to perform a wide variety of graphics operations including text display using arbitrary fonts, scrolling, window management and highlighting. Successive applications of BitBlt can perform operations such as scaling, area fill, rotation by multiples of 90 degrees and textured line drawing.

Bitmaps are usually stored in physical memory in what is called "raster order". Assuming the unit of storage is a 16 bit word and there is one bit of information per pixel, the leftmost 16 bits of the top row of the bitmap would be stored in the first word, followed by zero or more words containing the remainder of that row followed by zero or more additional words containing subsequent rows from top to bottom. Ordering of pixels within words is implementation dependent.

The left and right edges of bitmaps may fall on arbitrary pixel boundaries. Arbitrary pixel boundaries imply arbitrary bit positions within the underlying physical memory. One of the most useful properties of BitBlt is its ability to mask the details of source-to-destination bitmap alignment from the user. All direct handling of bitmaps and the word boundaries inherent in them is encapsulated by the BitBlt operation. The user specifies desired operations with dimensions given in pixels.

BitBlt performs the following operation: An arbitrarily-aligned rectangular region of a destination bitmap is replaced on a pixel by pixel basis with one of the 256 boolean functions of three variables: the previous contents of the destination bitmap, the corresponding pixel from an arbitrarily aligned rectangular region of a source bitmap, and the corresponding bit from a [halftone] bitmap. The halftone bitmap is typically a 16 X 16 pixel array interpreted as one tile of a pattern covering the entire destination bitmap. Examples of typical functions are shown below.

- Clear all pixels to 0
- Set all pixels to 1
- Invert destination
- Erase (Clear where Source = 1)
- Copy Inverse of Source
- XOR (Invert Destination where Source = 1)
- Copy Source
- Copy Halftone
- Paint Halftone (Source = "brush", halftone = "paint")
- Paint (Source OR Destination)

With the exception of single-pixel-width vectors and arcs, BitBlt provides a natural and powerful tool for manipulating text and graphics. Although BitBlt can be performed one pixel at a time, such an implementation would be quite slow. It is therefore highly desirable to devise a means of performing the BitBlt operation on multiple pixels simultaneously. We will examine this and other techniques for improving the performance of BitBlt in the next section. **5. Improving Performance**

As resolution and capability increase in a raster display system, the sweeper and graphics processor will be placed in increasing conflict for frame buffer access. Since the sweeper bandwidth requirements are usually inflexible, two alternatives exist for reducing this conflict:

- 1) allow the sweeper to obtain more pixels per frame buffer cycle.
- 2) provide a means to dual-port the frame buffer so that sweeper and graphics processor accesses may occur concurrently.

The first alternative requires increasing the width of the data path between the frame buffer and sweeper. This approach is usually only practical for widths up to 64 bits, where component cost and layout considerations tend to make further increases infeasible. Because of this limitation, increasing data path width can only provide a four-to-eight-fold improvement in bandwidth.

Further improvements require dual-porting the frame buffer. Since standard memory devices are inherently single ported, special hardware is required. Recently, Texas Instruments, NEC, and AMD have announced products that fulfill this need. These so-called "video rams" consist of a conventional memory element and a large shift register. The shift register is loaded with a number of bits (on the order of 256) with a [single] memory cycle. These bits may then be shifted out independently of other access to the memory element. Video rams have other capabilities (for example, shifting in data to the shift register and copying the register to the memory element with one memory cycle) that may also be useful in graphics applications.

In a frame buffer constructed from video rams, the graphics processor accesses the frame buffer in the usual manner. The sweeper, however, only needs to access the memory from time to time when the shift register requires new data. When video rams are employed in a system with a wide data path to the sweeper, extremely high data rates are possible. Put in simple terms, a 2K X 2K 60Hz non-interlaced display is nearly impossible with conventional memory devices, with video rams it is comparatively easy.

When we reduce sweeper demand on frame buffer bandwidth, the graphics processor is potentially able to access the frame buffer at a higher rate. This will not be the case, however, if the graphics processor is compute-bound, i.e., overwhelmed attempting to perform the computation necessary to generate the graphic elements of interest. In this case the raw performance of the graphics processor must be improved before the benefits of higher bandwidth can be exploited. **6. A VLSI Solution**

As an example of this situation, consider BitBlt. Because BitBlt is a powerful abstraction, it is difficult to efficiently implement. Microcoded implementations have proven successful. Other combinations of hardware support and clever software have met with varying success.

Since BitBlt is a repetitive process, an obvious mechanism for improving the performance of BitBlt is to optimize the inner loop. Within the inner loop, the following actions occur:

- 1) Data from a region in the source bitmap is shifted and aligned to produce words of source pixels aligned with destination words. This is necessary since, in general, word boundaries of source bitmap will not line up with word boundaries in the destination bitmap. Two words of source bitmap may be required.
- 2) Combining source, destination and halftone values according to the specified function. Recall that the function can be any of the 256 possible functions of three boolean variables.
- 3) Clipping destination modifications so that only pixels within the designated destination region are affected. The left and right boundaries of this region will not usually fall on word boundaries.

At PMR, we studied several techniques for improving performance of BitBlt and decided that a hardware accelerator for the inner loop was the most cost-effective solution. We first built an MSI TTL version of the accelerator to demonstrate the validity of our ideas. Encouraged by excellent results, we developed a full custom CMOS VLSI device that embodied what we had learned from the TTL version. Some key design criteria of the device were as follows:

The device should function as a general purpose data path so as to not constrain the user to a particular microprocessor family or memory technology. Control should be kept external to ensure this flexibility.

We should support all of the 256 possible boolean functions of source, destination, and half-tone. Several implementations had restricted the set of functions, thereby limiting the flexibility of the device.

The performance of the device should not be the limiting system design criterion. The device should support current and foreseen memory speeds.

The device should be low power to allow inclusion in battery-operated equipment.

The device should have TTL inputs and outputs and require only a single 5 volt supply.

The device should have minimal size.

The result of these design decisions and some careful simulation and layout by Kent Shimasaki of Seattle Silicon Technology was the PMR 96016. We call it the "Blt Chip". A block diagram of the 96016 is shown in Figure 3.

Figure 3 PMR 96016 Block Diagram

The PMR 96016 contains a data path and a number of configuration registers. The data path performs in hardware the functions that consume the most time in a software-only implementation of BitBlt, namely shifting, masking, and boolean combination.

As shown in the Block Diagram, the Blt Chip consists of two source registers, a destination register, a data path, and five configuration registers that control the data path. All data path elements are 16 bits wide. The three data path registers are the most frequently used and accordingly they are controlled by dedicated interface pins. These registers are:

Source Holds a word of source pixels. The Source register is loaded from the data bus using the LDSRC- pin.

Previous Source Holds the former contents of the Source register, so that two adjacent words of source pixels are available in the device at one time. The Previous Source register is loaded from the Source register whenever the Source register is loaded.

Destination Holds a word of destination pixels as they appear prior to modification. The Destination register is loaded from the data bus using the LDDEST- pin.

The configuraton registers control the operaton of the data path. The CS- interface pin causes the configuration register addressed by A1-A3 to be loaded from the data bus. Except where otherwise specified, all configuraton registers are 16 bits wide.

Skew Mask Controls the source merge unit, that forms one word from portions of two source words. For each of the 16 bit positions, a "1" selects the corresponding bit of the Previous Source Register, while a "0" selects the corresponding bit of the Source Register.

Skew Bits 3-0 specify a left-rotate amount for the rotator. For example, Skew=2 moves bit positions 0-2,1-3,...,15-1.

Halftone Row of halftone region appropriate to current row of destination. If used, the halftone register is rewritten at the beginning of each row.

Function Bits 7-0 specify the boolean function for combining the rotated source word, Halftone register, and Destination register. Each of these 8 bits specifies the desired result (0 or 1) for one of the eight possible combinations of source, halftone and destination. There are 256 possible functions.

Merge Mask Specifies the portion of the destination word to be modified. For each of the 16 bit positions, a "0" selects the corresponding bit of the function unit output, while a "1" selects the corresponding bit of the Destination Register (i.e., leaves that bit unmodified).

The pinout of the PMR 96016 is shown in Figure 4.

Figure 4 PMR 96016 Pinout

6. Using the Blt Chip

In performing a BitBlt operation the PMR 96016 acts as an accelerator for a controlling microprocessor. The BitBlt algorithm for which the Blt Chip design is optimized operates in scan-line order. It can be viewed as two nested loops, with the outer loop moving from one scan line to the next and the inner loop moving from one word to the next along a scan line. Setup of the Blt Chip configuration registers is done outside of the inner loop. In fact, most of the configuration registers need be written only once for the entire BitBlt.

There are four different types of cycles used to access different portions of the Blt Chip circuitry. All cycle types use the same 16-bit data bus. The cycle type is specified by asserting one of four control pins. The CS- pin is used to access configuration registers while the other three control pins access portions of the data path. All four cycle types are summarized below:

Control Pins

CS- Load (write) one of the configuraton registers (specified by A1-A3) with the data on the bus.

LDSRC- Load the Source register with the data on the bus and simultaneously transfer its previous contents to the Previous Source register.

LDDEST- Load the Destination register from the data bus.

OE- Read the data path result to the data bus.

The sequence of actions that constitute the body of the inner loop, and therefore must be performed the most frequently, is as follows:

- (1) Read a word from the source region in memory and load it into the Source register using LDSRC-.
- (2) Read a word from the destinaton region in memory and load it into the Destination register using LDDEST-.
- (3) Use OE- to obtain the new destination word and write it to the same memory address that was read in step 2.
- (4) Increment both the source and destination memory addresses so that they point to the next word along the scan line.

It is possible to design the Blt Chip into a system in such a way that all of the actions in steps 1 through 4 above can be performed with a single microprocessor instruction. If the microprocessor has a special instruction for moving a string of words, the entire inner loop becomes one string move.

The system components relevant to BitBlt operation include a microprocessor, a PMR 96016 Blt Chip, dynamic RAM, and a RAM controller. Whether or not the RAM has a second port for generating video to refresh a raster display device has little effect on the portion of the design described here. All data busses are 16 bits wide, although the microprocessor might have 32-bit wide data paths internally. The block diagram in Figure 5 shows these components and the primary busses and control lines interconnecting them.

Figure 5 Typical Application Block Diagram

The microprocessor serves as the BitBlt controller in addition to its task of CPU for the system. It accesses the Blt Chip configuration registers directly as individual words in its I/O address space (that might be memory-mapped). The three types of Blt Chip data path access cycles, however, are implemented as side-effects of accessing memory within certain address ranges. This type of memory cycle, further described below, we will call "blt-special."

In a typical application, the 96016 performs its task by intercepting data transferred from memory on read cycles (blt-special read) and substituting its own data on write cycles (blt-special write). The memory controller participates in this task by causing BitBlt memory write cycles to be converted into read-modify-write cycles. The read portion of the RMW cycle reads the previous destination contents. During the write portion of the cycle, the 96016 outputs the new destination contents. The host processor is used as an address generator and for initial setup. The result is that a horizontal row of pixels can be read from the source, combined with the destination and halftone, and written back to the destination, all requiring only two bus cycles per 16 pixels, plus a small amount of set up.

Previously, we outlined the sequence of memory and Blt Chip cycles that occur most frequently during a BitBlt operation and therefore should execute as fast as possible. The sequence involves two memory addresses, one for a source word and one for a destination word, and these addresses must be incremented before the next loop iteration. Most microprocessors can generate this type of alternating sequence of addresses rather quickly, using either a single string-move instruction or a sequence of move instructions where both source and destination addressing modes are register auto-increment.

Outside the microprocessor, at the bus level, this type of instruction manifests itself as a read cycle followed by a write cycle, with both cycles addressed in appropriate ranges to cause the cycles to be blt-special. By using these blt-special cycles we are able to transfer data between memory and the Blt Chip in a single bus cycle. By encoding this special type of cycle in the address, we get a sort of cycle-by-cycle context switch between blt-special and normal bus cycles.

In this example we will assume a microprocessor that generates a 24-bit address (for example, an MC68000) and we will use the highest-order address bit, A23, as a blt-special indicator. This divides the 16MB address space into two 8MB spaces. The lower half is allocated among memory-mapped I/O, ROM, and RAM, including Ram used for program and data storage and any dual-ported Ram used as video frame buffers. The upper half is then used to access the same resources, but with blt-special action. Blt-special action is not appropriate for all system resources, but it is not necessarily limited to frame buffer memory.

A blt-special read cycle is identical to a normal read cycle except for the side-effect of asserting LDSRC- so that the read data from memory will be loaded into the Blt Chip Source register. The resource addressed (regardless of A23) might be RAM or it might be ROM containing fixed images. The circuitry to generate LDSRC-, known as the "blt-special read controller" (BSRC), need only look at the CPU's status or strobe, A23, and the acknowledge or ready line to the CPU. The timing of LDSRC- should be such that its rising edge occurs at the time when the CPU captures the read data. This is shown in Figure 6.

Figure 6 Blt-special read cycle

A blt-special write cycle is quite different from an ordinary write cycle. It is orchestrated by a "blt-special write controller" (BSWC), that may be combined with the dynamic memory controller in some systems. Upon detecting that the microprocessor has initiated a write cycle with a bit-special address, the BSWC in fact performs a read-modify-write cycle. First the CPU data buffer is turned off in order to keep the CPU's write data from being driven onto the bus on which the Blt Chip resides, and the memory is accessed for reading. When the read data becomes available, LDDEST- is asserted to load it into the Blt Chip Destination register. Finally the Blt Chip OE- pin is asserted and the data path result is written to memory at the current address. The microprocessor "ready" or "acknowledge" line is used to stretch the bus cycle until the read-modify-write process is complete.

The write data provided by the microprocessor in a blt-special write cycle is not used. The only reason that the cycle is a write is because this is a convenient way to indicate that the address is a destination address rather than a source address, and therefore requires a different type of interaction between the memory and the Blt Chip.

The timing diagram in Figure 7 shows the actions of the BSWC during a blt-special write (read-modify-write).

Figure 7 Blt-special write (read-modify-write) cycle

Clearly blt-special write cycles are available only within RAM that has this special kind of controller. If possible, it is desirable to have such control for more RAM than just the frame buffer. This allows images to be built in non-displayed memory and then transferred to display memory later.

Figure 8 shows simplified but functionally correct code for implementing the complete BitBlt operation. For clarity of presentation, this version does no clipping and operates from top to bottom and left to right. The full version of the code clips to account for different size rectangles and rectangles that go outside of their bitmaps. It also correctly handles overlapping source and destination rectangles for operations such as scrolling down and/or to the right.

Reads with the BLTSPECIAL bit set cause the Blt Chip to load the data (from the memory location addressed by the other address bits) into its Source register. The previous source word is transferred to the Previous Source register.

A write operation with the BLTSPECIAL bit set results in a read-modify-write operation to the memory location addressed by the other address bits. The previous contents of the memory word are loaded into the Blt Chip's Destination register; the new destination word is computed combinatorially by the Blt Chip and is written to the same memory location.

The HALFTONE register is written once per scan line with a word from a 16 X 16 pixel bitmap.

Further optimization is possible, for example unrolling loops or creating dedicated code for special cases such as small rectangles, e.g., characters. **8. Where Do We Go From Here?**

Encouraged by our success with the Blt Chip, we are turning our attention to an application area in which the BitBlt operation provides little help: fast display of vectors. By employing some of the same techniques used in the development of the Blt Chip (i.e., operate on as many pixels as possible concurrently and optimize inner loops with cost-effective hardware support) we anticipate low-cost vector emulation on raster displays that competes effectively with the highest performance display technology available. Much work remains. The problems involved and the potential rewards are both significant.

The PMR 96016 is conceptually a simple device, but is perhaps more difficult to grasp on first examination than many other microprocessor peripherals. We believe the Blt Chip falls on the knee of the price performance curve and provides maximum performance for minimum investment. Extensive test data supports this conclusion. Additional data and application information may be obtained from the factory:

Pacific Mountain Research, Inc.
8026 35th N.E.
Seattle, Wa. 98115