All points addressable raster display memory

by Richard Matick
Daniel T. Ling
Satish Gupta
Frederick Dill

This paper discusses display designs which store the image point by point in random access memory, so that independent update of every pixel is possible. A frequent bottleneck in the design of high performance displays of this type is the available bandwidth of the memory subsystem. In this paper, we focus on this issue and present features of a customized dynamic RAM chip which can readily provide the necessary bandwidth and thus greatly simplify the design of very high performance APA raster scan displays. The customized RAM chip is quasi-two-ported. After briefly introducing APA raster displays, we discuss display memory system design and the design of the proposed custom memory chip. We describe the second port for the video refresh, which makes the primary port available for update almost continuously. We also discuss modifications to the existing primary port to make it easily usable for the parallel update required for high update performance as well as for other applications.

1. Introduction

Raster displays create images by repeatedly scanning a CRT from left to right and top to bottom. The electron beam's intensity is appropriately modified at discrete points (or pixels) on the screen, thus presenting the image information as intensity samples at each pixel. This paper discusses display designs which store the image point by point in a random access memory, called a frame buffer, so that independent update of every pixel is possible. Such displays are capable of producing arbitrarily complex, flicker-free images. We call such displays APA raster scan displays (APA stands for All Points Addressable).

A frequent bottleneck in the design of high performance displays of this type is the available bandwidth of the memory subsystem. In this paper, we focus on this issue and present features of a customized dynamic RAM chip which can readily provide the necessary bandwidth and thus greatly simplify the design of very high performance APA raster scan displays. The customized RAM chip is quasi-two-ported. After briefly introducing APA raster displays in Section 2 to more specifically identify the problem, we discuss display memory system design in Section 3, and the design of the proposed custom memory chip in Sections 4 and 5. Section 4 describes a second port for the video refresh, hence making the primary port available for update almost all of the time. Section 5 discusses modifications to the existing primary port to make it easily usable for the parallel update required for high update performance. Section 6 provides some trade-off considerations between the primary and secondary port, whereas Section 7 discusses other applications of the custom memory chip.

©Copyright 1984 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

2. APA raster displays

The development of APA raster displays has been primarily limited by the cost of random access memory. The cost of random access memory for a 512 × 512 bit-map display where each pixel can be only off or on has dropped from $2500 in 1971 to $30 in 1983. Early frame buffers used disks and drums for storage due to the prohibitive cost of random

Table 1 Time taken to update a screen of 10 000 characters.

Update time per pixel (μs)	*Total time for 10 000 characters* (s)	*Total bandwidth* (Mpixels/s)
10	8	0.1
1	0.8	1
1/16	0.05	16

access memory (Terlet [1], Ophir et al. [2]). When semiconductors became economical, some designs made use of LSI shift registers (McCracken et al. [3]), but since these memories were not very fast, the displays tended to have low resolution. The resolution of Terlet's disk display was 320 × 192 pixels, and McCracken's shift register had a resolution of 256 × 256 pixels. It was not until recently that random access memory has become cheap enough to be used for inexpensive raster displays.

The image on a raster scan display must be repeatedly scanned, which requires continual memory access. This process is usually referred to as *video refresh*. To achieve a flicker-free image, the refresh rate must be extremely fast. For example, a 1024 × 1024-pixel display refreshed 60 times per second displays a pixel every 12 nanoseconds. As a result, the memory must typically access several pixels in parallel in order to maintain video refresh rate. If the display were implemented using 64-kilobit chips, 16 chips could be read in parallel with a read cycle time of 192 nanoseconds.[a] However, the memory would have to be accessed on every cycle during actual pixel refresh time, leaving only the idle time during horizontal and vertical retrace for updating the memory. For this example, the idle time is a maximum of 4 ms or 24 percent memory availability for updating. This is insufficient time for most applications if one pixel per cycle is updated.

Since rapid picture update is desired, parallel pixel update is also necessary. Update speed is crucial for interactive uses of the display because large amounts of information may have to be changed even for conceptually simple operations. This problem is best illustrated by some examples.

A 1024 × 1024-pixel display can show approximately 10 000 characters, occupying an area of approximately 800 000 pixels on the display. **Table 1** shows the time taken to generate a new screen full of characters in the frame buffer assuming various update times for individual pixels. Frame buffer systems in which the host computer has to update (write) separately every pixel of every character might take 10 microseconds/pixel, in which case the total time of 8 seconds required to generate a new image is unacceptable. Even at an update speed of 1 microsecond/pixel, the total time still leaves much to be desired. To speed up the process, new characters can be copied into the frame buffer by writing several pixels in parallel. If 16 pixels can be written in parallel every microsecond, then an update speed of 1/16 microsecond/pixel can be achieved, giving a satisfactory screen update time. However, this update time is still too long to be done during the 4-ms retrace idle time, hence additional bandwidth improvements are still required.

The operation of copying pixels from one part of the image memory to another is a useful technique when scrolling a window across the frame buffer. For the scrolling of the entire 1024 × 1024-pixel display to appear smooth, it should occur in less than one frame time (e.g., 1/60 second). This requires an update bandwidth of 60 megapixels/second, which can be achieved if 64 pixels are copied every microsecond.

Typical calligraphic displays can draw several thousand lines during each refresh period. If we assume each vector to be approximately 100 pixels long (about 1/10 of the display), then a frame buffer display should update at least 3 megapixels/second in order to emulate a vector display. Table 1 shows that this cannot be done unless several pixels are updated in parallel.

In addition to the memory bandwidth requirements imposed by the display system, there are additional constraints imposed by the memory chips themselves. Dynamic memory cells require refreshing about every 2 ms, which can lead to contention problems with the CRT display refresh demands on the memory. Hence this must be taken into account in the design. Since dynamic memory refresh is a common problem for all bit-buffered displays using dynamic chips, we do not include it except to mention that under certain fortuitous circumstances, which do not often occur, the accessing of memory to refresh the CRT screen can also provide the dynamic cell refresh.

3. Display memory mapping

The display memory has two primary functions. First, it can be updated to change the data contained in it and hence produce new images. Second, it has to be accessed repeatedly to display the image on an output device. As discussed earlier, both the update and the output operations require parallel access to achieve satisfactory performance. To provide this parallelism, the display memory is organized into words, with each word containing the data for more than one pixel. The *memory mapping* determines how the pixels in each word map onto the display (Sproull et al. [4], Gupta [5]). If N pixels are to be accessed in parallel, then the display can be designed using N random access memory chips, where each chip can read or write one pixel per memory access.[b]

[a] Since the 64K-bit chips available today are typically not fast enough, page mode access, a lower refresh rate, or a smaller screen format has to be used, all of which are undesirable.

[b] If each pixel contains more than one bit, then $g \times N$ memory chips are required, where g is the number of bits per pixel. If more than one bit could be accessed in parallel from each chip, then fewer memory chips would be required.

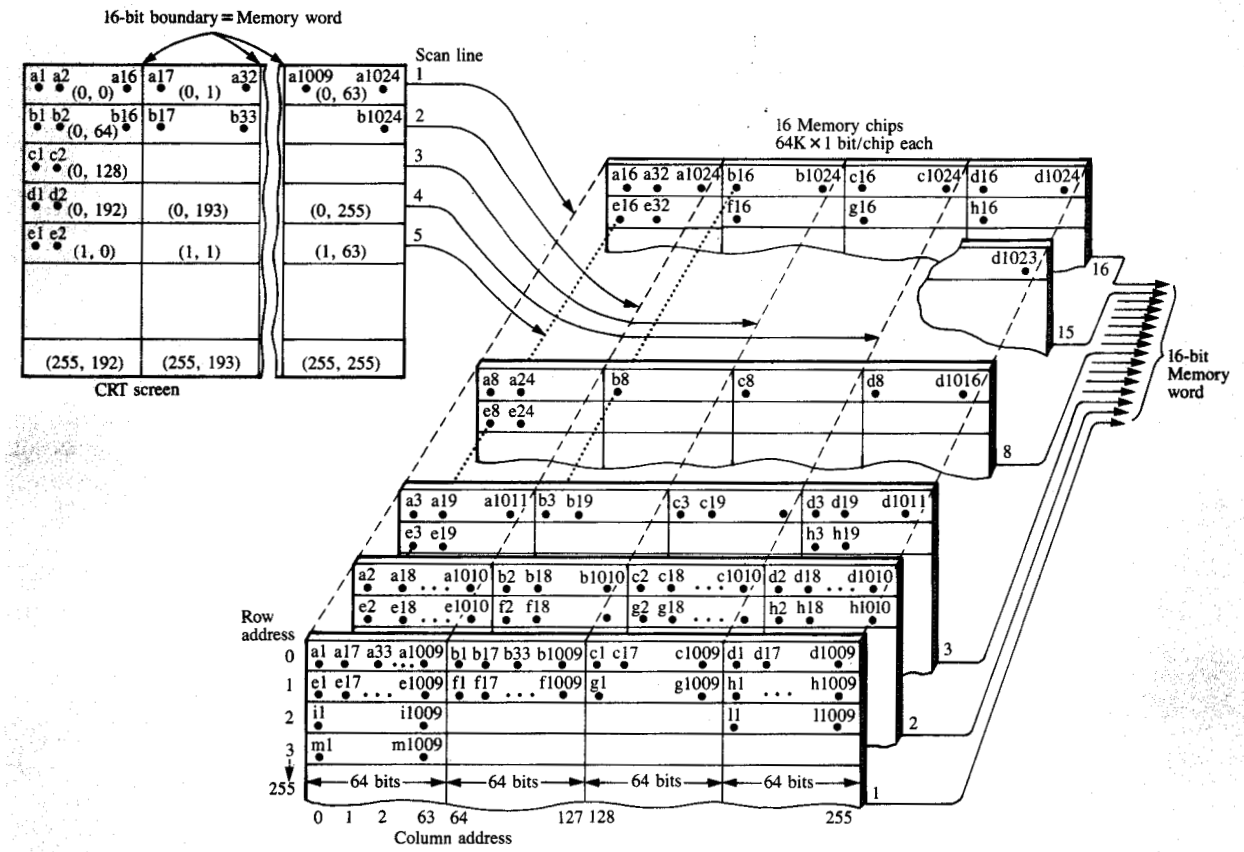

Figure 1

Memory mapping for a 1024 × 1024-pixel display with 1 × 16 mapping.

Scan-line mapping

For the most conventional display memory mapping, all pixels in each word are mapped along a scan line on the display. This organization is motivated by the high data rate required by the video refresh of CRTs. In a 1024 × 1024-pixel, 60-Hertz, noninterlaced display, the screen is refreshed at the rate of 12 nanoseconds/pixel. Assuming a memory cycle time of 192 nanoseconds, each memory access has to provide at least the next 16 pixels along the scan line for the video refresh controller to maintain the data rate required. If this display were implemented with 16 memory chips, then the only time available to update it would be during the horizontal and vertical retrace intervals.

Figure 1 shows the scan-line mapping for a 1024 × 1024-pixel display using sixteen 64-kilobit random access memory chips, each providing one bit per chip. All the pixels in each 1 × 16 box on the screen can be accessed by providing all the memory chips with the address (row address, column address) specified in parentheses in each box. Notice that the mapping shown allows the access of 16-bit sequential words along the scan line using sequential column addresses, hence allowing the use of fast page mode read for video refresh.

A disadvantage of this kind of mapping is that it introduces boundaries into the display corresponding to the word boundaries in memory. Parallel updates can take place only on the fixed 1 × 16 grid aligned to the column address boundaries of the memory and hence of the screen. If it is necessary to write 16 pixels into the display memory starting at any bit position which is not aligned to the fixed 16-bit word boundary, two separate writes to adjoining words of memory are needed. For instance, if we wish to write 16 pixels starting at pixel a2 in Fig. 1, the first 15 bits, a2 through a16, would be written on chips 2 through 16, respectively, on the first cycle with row-column address (0, 0), whereas the 16th bit, a17, would be written on chip 1 on the second cycle with address (0, 1). The second cycle

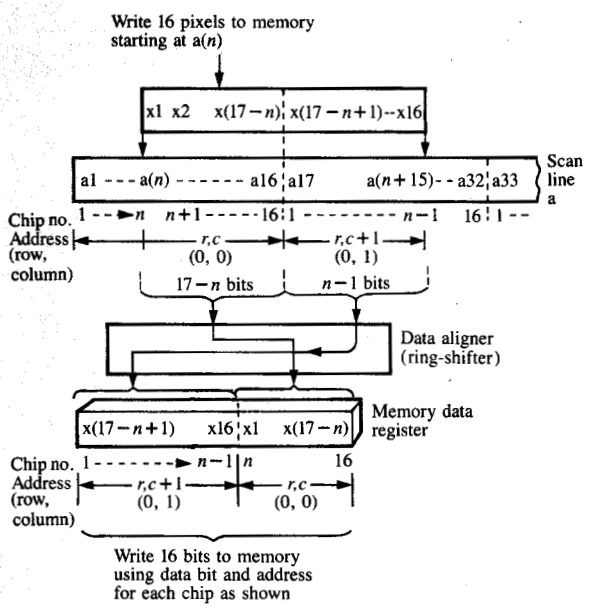

Figure 2

Updating an arbitrarily located 1 × 16-bit memory word starting at screen position a(n) through a(n + 15) showing data alignment and memory chip addresses required.

only requires writing to a chip which was not accessed on the first cycle and only needs the column address to be incremented by one. The data alignment and addressing can be seen more clearly in **Figure 2**, where for this example n = 2. To load a 20 × 16-pixel character would probably take 40 memory cycles, and 20 only in the rare case where the character is aligned to the 16-bit-word boundary.

Scan-line mapping can be modified to provide single cycle access to any horizontal span of 16 pixels starting at any arbitrary pixel position. This is accomplished by simultaneously addressing different memory chips with separate addresses, as shown in Fig. 2. An arbitrary span of 16 pixels can cross only one word boundary, and two addresses are sufficient to access the span. In addition, the two addresses differ only by one.

Symmetric mapping

Scan-line mapping imposes an inherent asymmetry on update operations to the memory chips—horizontal updates are easier than the vertical ones. For example, a horizontal line can be drawn very quickly, but only one pixel of a vertical line can be drawn in one memory cycle. This can be improved by changing the memory-to-screen scan-line mapping of Fig. 1 to a new mapping wherein the 16 bits written into memory represent a 4 × 4 square array on the screen, as shown in **Figure 3(a)**. The memory addresses (row address, column address) are shown inside the arrays, in parentheses. As in the case of the scan-line mapping, the symmetric mapping also has the property that sequential squares along the scan-line direction require sequential column addresses, as shown. Whenever 16 bits are appropriately written into memory, they will now appear on the screen as a square array rather than a straight horizontal line. By the use of appropriate masks, any vector, horizontal or otherwise, or any pattern within the square array can be written in one memory cycle. It was for these reasons that the designers of the 8 × 8 display (Sproull et al. [4], Gupta [5]) chose a symmetric 8 × 8 organization. In an 8 × 8-pixel display, 64 pixels can be read or written in each memory cycle; the 64 pixels read or written lie on an 8 × 8 square on the screen. The line drawing operations are now symmetrical with respect to the x and y axes of the screen.

Figure 3(b) shows how the scan lines now map to the same sixteen 64K-bit memory chips of Fig. 1 for a 1024 × 1024-pixel display using 4 × 4 square array mapping. Since a scan line is now contained within only 4 chips rather than 16, for video refresh the memory chips must be accessed four times as fast as for scan-line mapping. A multiplexer or selector of some sort is required to select sequential groups of four chips for sequential scan lines. The addressing scheme shown in Fig. 3 has the same boundary problem that we discussed before. It does not allow access to an arbitrarily positioned 4 × 4 square on the screen in one memory cycle. By providing incremented row and column addresses to different chips in the memory chip array, we can access an arbitrarily positioned 4 × 4 square. For instance, **Figure 4** shows the data alignment and row-column addresses needed to write the nonaligned square array shown in Fig. 3(b) starting on the screen at pixel a12 in Fig. 3(a). At most, an increment of one is required in the row, the column, or both.

The symmetric memory organization is also appropriate for driving a multiple-beam CRT, in which several beams simultaneously scan the CRT face. Such a CRT can provide a larger number of pixels, as well as higher refresh rates and brightness levels, but it requires data from several consecutive scan lines simultaneously. Symmetrical 8 × 8 mapping allows eight beams to be scanned together on a multiple-beam CRT.

4. Dual-ported memory

A dual-ported memory chip can effectively double the available bandwidth to and from the memory system and can be used to decouple the accesses required for picture update from those for CRT video refresh. However, a full dual-ported memory is very expensive, both in terms of silicon area and pin count, and is also unnecessary for display memory systems. We can use the nature of video refresh and the characteristics of dynamic RAMs to provide

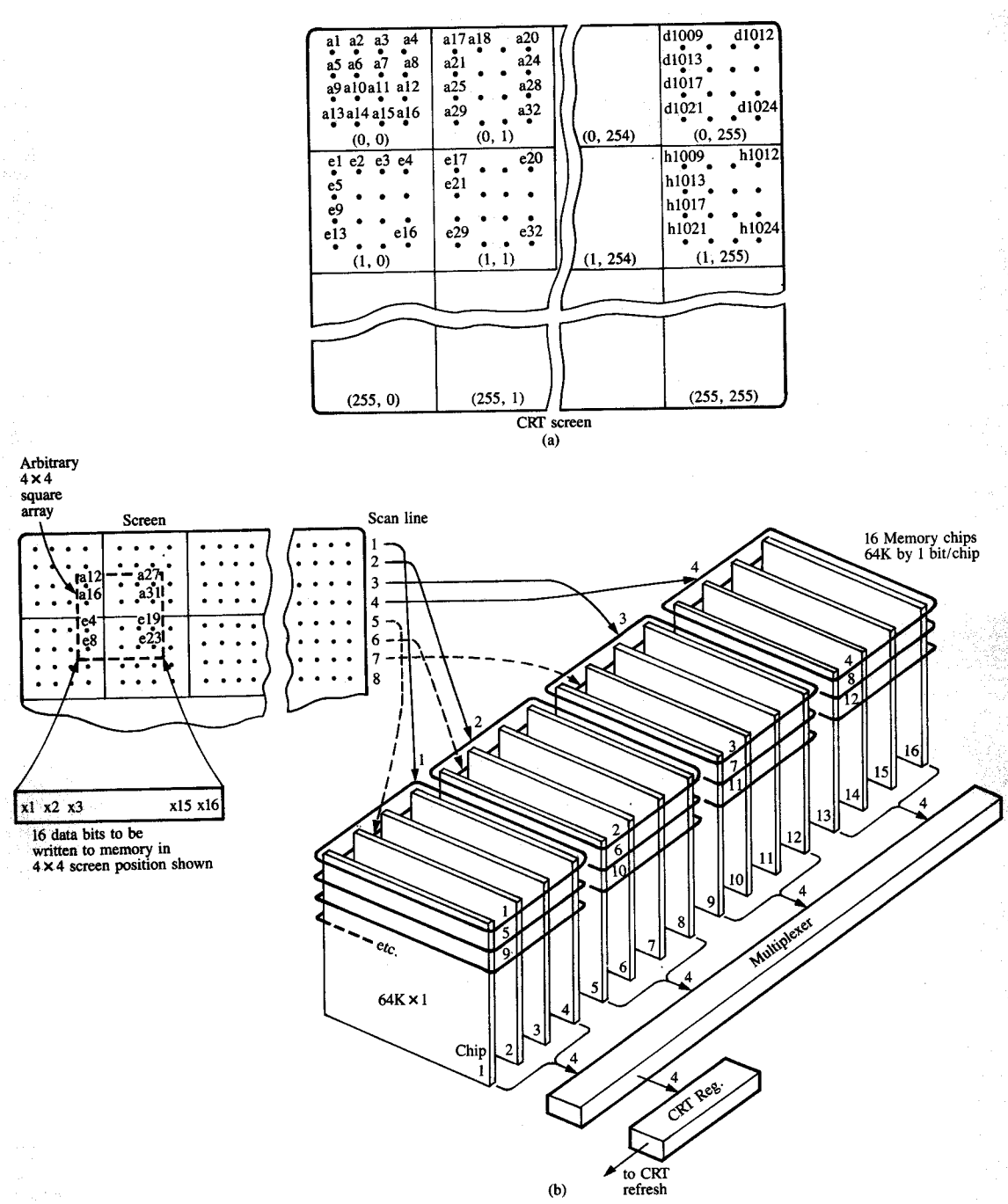

Figure 3

Symmetric memory mapping using a 4 × 4 square array. (a) Bit designation on 1024 × 1024-pixel screen; (b) mapping of screen to sixteen 64K-bit by 1-bit/chip memory chips, showing an arbitrarily aligned 4 × 4 square array.

a simpler quasi-second port to the memory chip, which can more than double (e.g., triple or quadruple) the effective bandwidth for many applications. As we show in this section, the quasi-second port is much easier and cheaper to implement because only small changes to existing dynamic RAMs are needed.

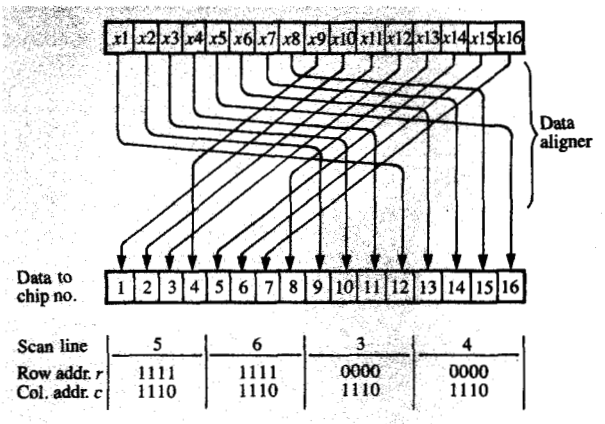

Figure 4

Data alignment and row-column address required for writing into memory the arbitrary square array of Fig. 3(b).

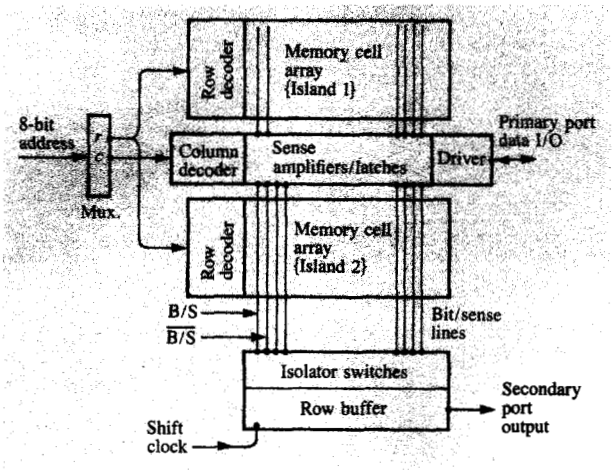

Figure 5

Schematic of quasi-dual-ported memory chip showing a row buffer implemented as a shift register and directly connected to the far end of a folded bit/sense line.

The video refresh port of the display memory is a serial port, clocked and addressed in a regular and predictable fashion. Also, in both the scan-line and symmetric memory organizations, successive words in the scan-line direction can be accessed at successive column addresses in a given row, as can be seen in Figs. 1 and 3.

Random access memory chips usually store the bits in a large array of memory cells. A 64K-bit RAM will hence probably contain 256 words of 256 bits each.[c] Addresses for such a RAM are provided on eight multiplexed address lines. The row address selects which word is driven onto the bit lines. The subsequent column address is used to gate one sense amplifier to the data output.

We can add a quasi-second port to such memory chips by providing a row of registers alongside the sense amplifiers into which the contents of the sense amplifiers can be gated, as shown in **Figure 5**. We refer to this set of registers as the row buffer, because of its ability to buffer one row of the memory array. The row buffer can be viewed as a separate, largely independent 256-bit static memory which can be connected into the main array through the sense amplifiers or totally isolated from the array by the isolator switches, as shown. It is only during transfer of data between the sense amplifiers and the row buffer that there is any need for synchronism between the two memories (Ling [6]).

If the row buffer is configured as a serial shift register, with its output provided at an independent data output pin (secondary port), then successive column addresses on the row can be rapidly shifted out. As shown in Fig. 5, this provides a serial quasi-second port which can be used for video refresh in both the scan-line and the symmetric memory mappings.

In the scan-line mapping of Section 2, the row buffer shift registers of each memory chip contain every sixteenth pixel along a scan line (and hence all the pixels required from this memory chip for four scan lines). The serial outputs of the 16 shift registers (one from each memory chip) are serialized by a 16-bit high speed shift register to form the output video **(Figure 6).**

In the symmetric mapping of Section 2, if each chip of Fig. 3(b) contains a row buffer of the type shown in Fig. 5, then the loaded row buffers of 16 memory chips contain one pixel for all 4 × 4 squares of four scan lines. One scan line is thus contained in a group of four chips, as shown. To produce the video output, the four row buffer shift registers containing the current scan line are serialized using a 4-bit high speed shift register through some multiplexer or selector logic, as indicated in Fig. 3(b). It is assumed that the serial output from the memory chips are tristate. Note that the 4 × 4 square mapping requires the row buffer to shift at four times the speed required by the 1 × 16 scan-line mapping.

The symmetric 4 × 4 mapping can also be used to drive a 4-beam CRT, in which case all 16 memory chips will be shifting their shift registers simultaneously using four 4-bit serializers to create the four video streams.

Connecting the row buffer as a shift register is the least flexible mode of operation, but it is adequate for many applications, such as video refresh. The quasi-second port can be made bidirectional by gating the row buffer in order to write a particular row of the memory. The second port

[c] Some chips are implemented using several islands of fewer words or fewer bits per word. They can however be conceptually thought of in the canonical 256 × 256 organization.

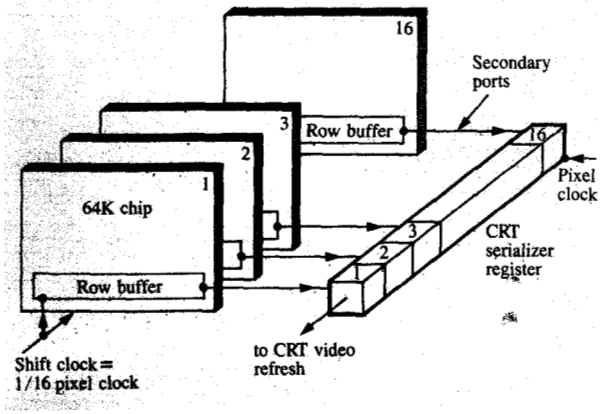

Figure 6

Schematic of the use of the row-buffer secondary ports to provide CRT video refresh for the case of scan-line mapping.

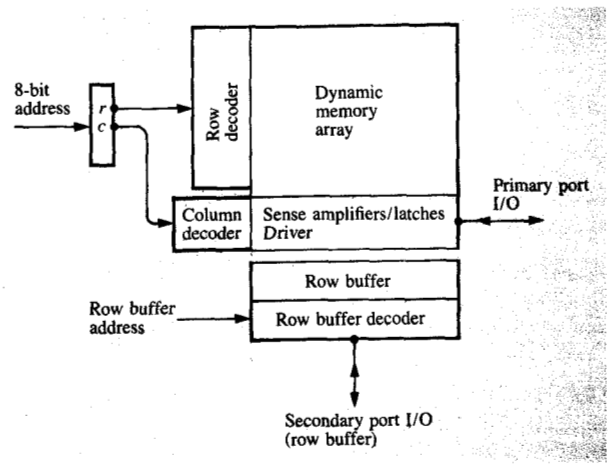

Figure 7

Memory chip with row buffer configured as a static, fully decoded array.

could be connected in alternative configurations, e.g., an 8-bit parallel output might be desired when the chip is used as the character refresh buffer for a character-based display.

Another more general arrangement for handling the input/output of the row buffer is to use a decode tree much like that used to decode from the sense amplifiers for the primary port. This allows a much more flexible addressing of the information into and out of the register row. Again, this does not have to be arranged with a bit decode organization unless a one-bit-wide output is desired. The modified memory is shown in **Figure 7.**

5. Primary port

The quasi-second port increases the bandwidth of the memory chip to the level needed for highly interactive displays. This section discusses a few changes to the primary port which increase its capabilities as well as reduce the overall cost of the memory system.

• *Address incrementer*

Both the scan-line and the symmetric mappings presented in Section 2 required a subset of the memory chips to receive incremented addresses to provide bit-addressability of multiple-pixel words (see Figs. 2 and 4). The easiest way to implement this is to provide an on-chip address increment which can increment either the row or the column addresses. A separate pin would control whether the address is incremented or not. External logic would now determine which memory chips increment their addresses and which do not. The increments can be determined from the bit address within the word in both the scan-line and the symmetric memory mappings (Gupta [5], Matick et al. [7]).

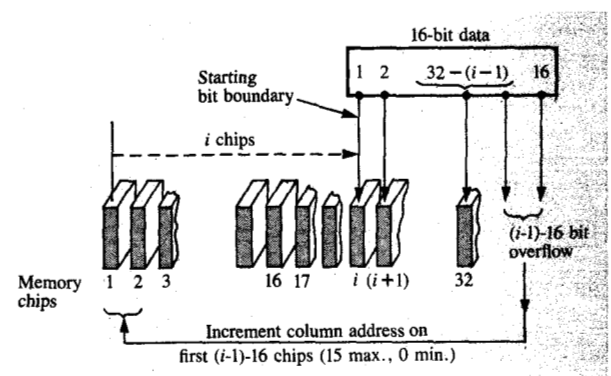

Figure 8

Schematic of a memory bank consisting of 32 chips at 1 bit/chip and having a 16-bit I/O word, showing that only the left half of the memory chips need incremented addresses.

The address incrementer can be implemented using any of several techniques. The easiest is to add combinatorial logic just after the address input pins to increment the address. Another possibility is to shift the decoded outputs of both the row and column decoders when the increment signal is asserted. The design trade-offs for these possibilities are discussed in Section 6.

If the number of memory chips used to implement the display memory is more than twice the width of the data path, then each memory chip does not have to independently increment its address. As shown in **Figure 8,**

if 32 memory chips are used to implement a 1 × 16 memory mapping, then only the leftmost 16 memory chips have to increment their addresses for the case shown. This kind of bank incrementing can simplify system design when an on-chip incrementer is not available.

The lack of an on-chip address incrementer forces the choice between an off-chip incrementer and an address multiplexer which selects between the incremented and the unincremented address. This results in the use of a large number of external parts if the number of memory chips is not greater than twice the width of the data path. A solution to this problem could be to time multiplex the two addresses (the incremented and the unincremented one) and strobe only the chips which require the particular address. This solution leads to a slower system.

• *Pixel aligner*

The ability to access a bit-addressable word introduces an alignment problem. Figures 2 and 4 show that the bit accessed from the leftmost chip is not necessarily the leftmost pixel of the desired word. The word read has to be aligned such that the processor can manipulate it in the desired order. The alignment can be performed by circularly shifting the word read from memory by the offset from the alignment boundary.

The external aligner can be eliminated if each memory chip can be augmented with several input and/or output pins, only one of which is used in any given memory cycle. The augmentation takes the following form: Replace the single data input pin with 2^n multiplexed data input pins, replace the output pin with 2^m demultiplexed data output pins, and add the maximum of n or m control pins. The control pins either select one of the 2^n input pins on data input or one of the 2^m output pins on data output.

The purpose of the additional data pins is to permute the data among the memory chips. The wiring shown in **Figure 9** allows an arbitrary circular shift of the eight data bits. In this case there are eight output pins, one input pin, and three control pins. The operation of this circuit can be understood in the following way: Assign to each memory chip input pin one of the eight wires in the bus below the memory chips; in this case assign the topmost wire to the leftmost chip, and so on, as indicated by the connections of the input pins. Then, by virtue of the wiring, the simultaneous selection of the same output pin on all chips, via the control lines, causes the data read from the memory to be placed on the bus, and also on the input pins, in some permuted fashion, depending on which output pin has been selected. In this case, for example, simultaneously selecting output pin 0 on all chips causes no permutation, selecting pin 1 causes an end-around shift by one, and so on.

6. Implementation considerations and trade-offs

The foregoing discussion was intended to illustrate the various types of mappings between frame buffer and screen, and the types of additional functions which can be used to advantage. Two additional memory chip hardware features would greatly facilitate the various mappings while providing high bandwidth for simultaneous screen refresh and frame buffer updating—an on-chip secondary port supplied by a row buffer, and an address incrementer. These features can be implemented in numerous ways with various trade-offs which are discussed here.

One of the main considerations in the design of a memory chip with these special features is the cost, as reflected in the additional silicon area needed. These features are added to every memory chip of the frame buffer. Thus, for large frame buffers which are a significant part of the overall cost, a large percentage cost increase is not justified unless the components displaced by the features are larger than the increased memory cost. Such detailed cost comparisons are highly system dependent and cannot be done for a general case. As memory costs continue to decrease, these additional features look quite attractive, even at a larger percentage increase in chip cost, but cheap memory encourages systems with larger frame buffers and/or more bits per pixel. Thus the added cost tends to increase. Hence the objective is to add the smallest possible amount of additional circuitry to the memory chip, commensurate with being able to do the screen refresh without high speed shifting of the row buffer and with high availability of the primary port for frame buffer updating. These two additional features are functions which can be added to the periphery of a dynamic memory array. The input to the row buffer is essentially the output of the sense amplifiers, so that the internal parts of the array and sense circuits need not be disturbed. Address incrementing can, in principle, be done at the address receivers, prior to decoding. Then, the incrementer could also be on the periphery of the array and need not disturb the internal circuits. Thus, both the cost and design time can be limited by using a suitable, existing memory design and modifying it to include these features.

Typical dynamic memory arrays contain a column of sense amplifiers imbedded between two arrays of memory cells. For sensing, a true and a complement signal are required, with the complement provided by a "dummy cell" connected to the opposite side of a cross-coupled sense latch. If this dummy cell were physically on the opposite side of the sense amplifier from the bit/sense line being interrogated, then connection of the row buffer to the sense latch would be very difficult. In fact, a separate line would have to be placed over the array for each bit of the row buffer. This is extremely undesirable for many reasons—cell density on the array is compromised, the sense circuit may become unbalanced. However, if a "folded" bit sense/line is an inherent part of the memory design, then the row buffer can just be connected to the far end of the bit/sense line pair, as shown in Fig. 5. Thus, a folded bit sense/line, or equivalent, is highly desirable in the basic array design to permit

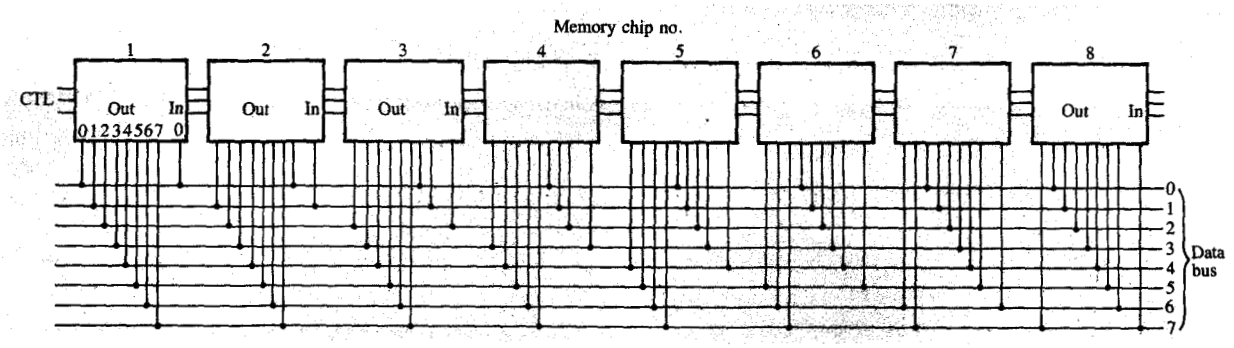

Figure 9

On-chip circular shift obtained using eight output pins.

connection of the row buffer on the periphery of the array with no modification to the array itself. After the memory array choice has been made, the next decision involves the row buffer itself, its architecture and bit capacity.

• *Row buffer architecture*

If there were no cost constraints, the most useful row buffer would be a single stage register of such a capacity that, for any given memory plane configuration which maps to one full screen, one load of all the row buffers on the chips of that plane would contain at least one scan line for the largest screen format of interest. For example, suppose the largest screen of interest is 1K × 1K pixels, refreshed by a 1-megabit frame buffer. If the frame buffer were made up of 64K-bit chips providing one bit (one row buffer) per chip, then the 16 row buffers on the 16 chips should hold a total of 1K bits, which requires a minimum of 64 bits per row buffer. If the row buffers were all, say, 32 bits each, then the 16 row buffers would only hold 512 bits, which is half of the assumed scan line on the screen. In order to get uninterrupted refresh of the screen, some special off-chip buffering of the data out of the row buffers would be required to hold the data while a (slow) reload of the row buffer took place. This special buffering can be avoided, and the application of the chip extended to allow uninterrupted screen refresh for all possible screen formats by the use of a two-stage (master/slave) row buffer organization in which the master and slave have independent load-enable signals. The master is loaded from the memory array, and the slave is loaded from the master. The data are then serially shifted out of the slave to the CRT register. After a slave has been loaded and is being used to refresh the screen, the master can be independently loaded with the next sequential data at any convenient time. If all the slave row buffers do not hold one scan line, then, when the last bit of the slave is being shifted out, the master can be loaded back to back into the slave so that an uninterrupted flow of bits appears out of the slave to the CRT register. This avoids all critical timing problems as well as the need for any additional buffering.

The conditions under which a master/slave organization is useful depend on scan line size, bit capacity of the memory chip, and numbers of row buffers per chip providing separate data paths to the CRT register, as follows: Assume that

L = number of scan lines on the screen,
S_b = bits (pixels) per scan line,
F_b = bits per frame buffer,
B_c = total number of bits/chip,
N_{RB} = number of row buffers per chip providing separate data paths to the CRT register, and
B_{RB} = total number of bits for each of the N_{RB} row buffers.

The minimum number of bits per row buffer, assuming one scan line per load of row buffers, is found as follows. For a frame buffer of F_b bits, using chips of capacity B_c, the total number of chips required, N_c, is

$$N_c = \frac{F_b \; chips}{B_c \; frame}. \tag{1a}$$

But the frame buffer size is set by the number of lines L and bits per scan line, S_b, or

$$F_b = LS_b. \tag{1b}$$

Substituting Eq. (1b) into (1a) gives the required number of chips as

$$N_c = \frac{LS_b}{B_c} \frac{chips}{frame}. \tag{1c}$$

With this number of chips, the total number of row buffers available to hold the scan line is

$$\frac{Total\ no.\ row\ buffers}{frame} = N_c N_{RB}. \tag{2}$$

The total number of bits held in these row buffers is B_{RB} times Eq. (2), and this must be equal to (or greater than) one scan line; thus,

$$S_b \leq B_{RB} N_c N_{RB} . \quad (3)$$

Substituting Eq. (1a) and solving for B_{RB} gives the size of each on-chip row buffer needed to hold the full scan line as

$$B_{RB} \geq \frac{B_c}{N_{RB} L} \frac{bits}{row\ buffer} . \quad (4)$$

As an example, if the frame buffer is F_b = 1M pixels, the scan line size is 1K pixels, the chips are 64K bits each, and there is one row buffer per chip, then Eq. (4) gives $B_{RB} = 64$ bits per row buffer to hold one scan line. If the number of bits per chip increased to, say, 512K, all other parameters remaining the same, then B_{RB} must increase to 512 bits per row buffer to contain one scan line. Note that this value increases directly with the chip capacity, all other factors remaining constant. This is the case because, as the chip capacity increases, the number of chips required in the frame buffer decreases directly, and the total number of row buffers therefore decreases, also directly. Hence the row buffer capacity per chip must be increased directly.

Again, if all other factors remain constant, the row buffer size is independent of scan line size since more chips (and more row buffers) must be added to each frame as the scan line length increases. The additional row buffers exactly match the capacity required for the increased scan line length.

A row buffer of 512 bits is rather excessive since considerable area is required. One might consider building a master/slave type row buffer of fewer bits to circumvent the problem. The point at which this becomes feasible is a function of the physical size of the master and slave. Assume that the master consumes exactly the same amount of silicon area as the slave. Further assume that the chips are 512K bits, requiring a 512-bit row buffer to contain one scan line. In such a case, it would be much more economical to build a master/slave type of row buffer, each of 64 bits, rather than one row buffer of 512 bits.

The 64-bit master/slave configuration would require a reload from the main array four times as often as a 512-bit row buffer, but the impact on the primary port availability for updating with new information would be very small. Besides, the master could be loaded when convenient (e.g., when there was no updating) and would not be limited to a fixed time slot. The only requirement is that it be loaded prior to the slave being totally shifted out to the CRT register.

The master/slave would provide uninterrrupted refreshing for any size screen format and any number of bits per chip. One could even consider building a master/slave of, say, 32 bits, which would be very economical and serve a wide range of screen formats. The frequency at which the master would have to be reloaded from the main array would increase for the above case, but for many applications, especially with smaller size screens, this would be negligible.

The major impact of a small row buffer is on the availability of the primary port for updating (writing) new information into the frame buffer and the increased shift rate to supply refresh data. Obviously, if the row buffer must be reloaded every main memory cycle, it is of no value. However, for nearly all cases of practical interest, a master/slave row buffer of 32 bits would give an average primary port availability in the range of 90 percent or more. These values are taken from **Table 2** for the typical case of using chips of either 64K or 256K bits per chip with one row buffer. The average primary port availability, (b) in Table 2, is taken as the total percentage of time that the primary port is free during the 16.7-ms screen refresh time, assuming a 1M-bit frame buffer having a 200-ns cycle time, and a 1K × 1K-pixel screen. This is the number of interest. Also given is the minimum instantaneous primary port availability (a), which is the percentage of time the primary port is free during the time period of one horizontal scan line (assumed to be 12 ns per pixel times 1K pixels) without including the retrace time. These availabilities (a and b) are tabulated, as a function of row buffer size, for various memory chip capacities. For the assumed screen size and memory cycle time, a single 32-bit row buffer would begin to have a significant effect on the average primary port availability at 512K and 1M bits per chip. However, it is not likely that a single 1M-bit chip would be used with a 1K × 1K-pixel screen at any row buffer size since the shift rate of the row buffer would have to be 12 ns per bit. This is unreasonable. A more desirable configuration for chip capacities above 256K bits is to essentially make multiple islands of the 256K-bit configuration with a 32-bit master/slave row buffer. Thus a 512K-bit chip would have two 32-bit row buffers, each providing one bit to the CRT register, and a 1M-bit chip would have four such row buffers. Under these circumstances, the maximum shift rate required of the row buffer would be once every 48 ns. This is commensurate with the likely speed of such circuits. Primary port availability is thus maintained in the range of 90 percent, while the additional circuitry needed for the row buffer is quite reasonable.

While a small row buffer capacity is desirable in terms of the silicon area consumed, it introduces additional complexity into the chip design. Dynamic memory chips typically have 128, 256, or even more sense amplifier/latches. It is a very simple matter to just latch each of these into the row buffer whenever a row is accessed. However, this tends to make the row buffer rather large. If it were decided to use a smaller row buffer, then some additional decoding and additional addressing would have to be done. For instance, suppose there were two islands on a given chip, each having 256 sense latches. If we wished to provide one row buffer of 32 bits, then each of the two groups of 256

Table 2 Primary port availability vs frame buffer and row buffer parameters for an assumed 1K × 1K-pixel screen, 1M-bit frame buffer at 200 ns cycle time. Screen refresh occurs at 12 ns/pel, 60 Hz.

B_b bits/chip	N_c No. chips / frame	N_{RB} No. row buffers / chip	B_{RB} row buffer capacity / chip (bits)	No. row buffers accessed per scan line	Primary port availability (%)	
					a	*b*
64K	16	1	32	2	96.8	97.5
			64	1	98.4	98.8
			128	1/2	99.2	99.4
			256	1/4	99.6	99.7
256K	4	1	32	8	87.2	90.2
			64	4	93.6	95.1
			128	2	96.8	97.5
			256	1	98.4	98.8
512K	2	2	32	8	87.2	90.2
			64	4	93.6	95.1
			128	2	96.8	97.5
			256	1	98.4	98.8
1024K	1	4	32	8	87.2	90.2
			64	4	93.6	95.1
			128	2	96.8	97.5
			256	1	98.4	98.8
{The following are for one row buffer per chip.}						
512K	2	1	32	16	74.4	80.3
			64	8	87.2	90.2
			128	4	93.6	95.1
			256	2	96.8	97.5
1024K	1	1	32	32	48.8	60.6
			64	16	74.4	80.3
			128	8	87.2	90.2
			256	4	93.6	95.1

[a] Minimum instantaneous primary port availability during one scan line refresh excluding retrace time.
[b] Primary port average availability over 16.7 ms screen refresh time.

sense lines would have to be decoded into eight groups of 32 bits. Furthermore, the 32 bits coming out of each island must be multiplexed into the row buffer. While all of this decoding circuitry could be placed on the periphery of the chip without disturbing the original array design, it nevertheless drives the design toward a larger row buffer.

Thus the final design of the row buffer must include three components—the row buffer master, row buffer slave, and any additional decoding circuitry necessary to fill the row buffer. Once a decision has been made concerning the use of a master/slave vs just a slave organization, the problem is reduced to finding the minimum area required for the buffer and additional decoding circuitry. This can only be determined by actual design in a given technology, using a given array as a starting point.

• *Address incrementer*

The address incrementer is used only for reading or writing through the primary port. In principle, address incrementing is a simple function which can be performed on the addresses as they come onto the chip. For a chip which uses multiplexed row and column addresses, either one or both of these can be incremented by the same adder circuit before being latched. This minimizes the additional circuitry. However, there are several problems. First, this adds time delay directly into the critical path of the chip access. Second, the timing chain of the original chip may, or may not, have to be changed, depending on the type of incrementer used and how the delay is factored into the system. Typical dynamic chips require that the row address be valid at the time the strobe pulse ($\overline{RAS}$) becomes valid. The ($\overline{RAS}$) strobe acts as a chip select and starts a timing chain sequence. The row addresses must typically be valid for some minimum time, just long enough to be "received" and latched into the row address buffer. If the address incrementer were a dynamic circuit and logically appeared prior to the latch, then its delay would have to be added to the minimum address hold time. This requires a change in

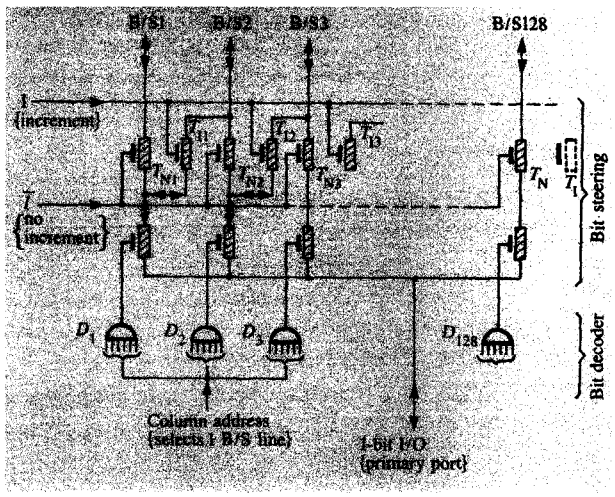

Figure 10

Bit address incrementing using bit steering on bit/sense lines with existing decoder.

the timing sequence on the chip to accommodate the additional hold time. This could be avoided by using a static incrementer together with an external requirement that the row addresses be valid for at least a time equal to the incrementer delay, before the ($\overline{RAS}$) signal becomes valid. This would eliminate changes to the timing circuits for row addresses but require a static incrementer.

Since the incrementer is a relatively small circuit compared to the row buffer, the additional power for a static circuit might be tolerable. This would require detailed design in the given technology before being decided. For either a static or a dynamic incrementer, an additional delay is incurred for the total access time to the chip.

For column address incrementing, using the same incrementer, the circuit delay must also be added to the column address hold time. However, all or at least part of this extra delay can usually be overlapped with the row address decoding and sensing, for the following reason. After the row address is latched, the column address input can be started. Before the actual column address is needed to select one of the bit-sense latches, the row address must go through a long chain of driving row address lines, decoding (1 out of 128 or 256) word (row) lines, sensing the signal (a slow process), and setting the sense latches. While this is proceeding, the column address must be incremented, latched, and driven to the column decoder. The decoder is timed so as not to select a sense line until the sense latches are set. The exact amount of the total column incrementing that can be overlapped depends on the chip design and technology.

If the column incrementing time delay cannot be avoided in the total chip access time, there is a completely separate way to circumvent this, but at a cost. The column incrementing can be done on the sense line side by a simple bit steering circuit as follows. The logical meaning of incrementing a column address is to just select the bit which follows the one specified by the column address. For instance, in **Figure 10**, if the given column address normally would select, say, bit/sense line 1 (B/S1), then the incremented address should select B/S2. If the bit decoders are to be left unchanged, then the incrementing can be achieved by adding two switch devices, T_N and T_I for each bit/sense line, as shown. When no incrementing is to be done, signal $\bar{I}$ is valid and all T_N devices are on. This allows the normal decoding sequence to take place. When incrementing is to take place, signal I is on and $\bar{I}$ is off, thereby turning on all devices T_I. If the given column address still selects the first position, decoder D_1, then device T_{I1} will steer the signal from B/S2 into decoder D_1, as desired. This bit steering avoids the need for an adder circuit, uses the existing decoder circuits, and can be very fast. Note that these steering devices can, and must, be turned on before the decoding is completed. In fact, as soon as the I or $\bar{I}$ signal is valid on the appropriate pin, the steering devices can be allowed to switch without concern for timing sequences. Thus the incrementing delay can be completely overlapped and hidden under the normal access delays. Unfortunately, this method introduces two devices per B/S line. Also, these devices must be physically inserted in between the bit/sense lines and the decoders.

Since the decoders are typically imbedded within the array structure, this requires disturbing the original physical design. This method of address incrementing is, therefore, not very attractive if the intent is to just add features to an existing array design. On the other hand, if the intent is to design an optimal display memory chip, bit steering is worth considering.

If address incrementing is required on the row address, a similar type of steering can be used on the word (row) lines. The output of a word line driver-decoder would be directed into the usual line for no increment or to the logically adjacent line if row incrementing were desired.

7. Other applications of display memory chips

This section discusses nondisplay applications of the quasi-dual-ported memory chip. These discussions also result in a few minor modifications to its architecture, which make it a much more generally applicable part.

- *Printers: fast clear*

APA printer systems present memory problems identical to those of APA display systems. Fast printer systems require the quasi-dual-ported memory for exactly the same reasons as do fast display systems. The memory organization

arguments used for display memory system designs are identically applicable to printer memory systems.

But printers have one property which requires a small modification to the quasi-dual-ported memory chip. Typically, a printer clears the frame buffer memory before it prints the next page, whereas displays are repeatedly refreshed. The clearing requirement implies that once the data in a given row have been latched into the second port's row buffer, it should be cleared in the memory array. Without such a feature, all the bits in that row would have to be cleared one bit at a time, and the bandwidth advantage provided by the second port would be lost.

We propose to add one extra input pin (which we call the clear input) to the memory chips such that when the signal on that pin is asserted during a row address strobe, all the bits in that row are set to the value asserted on the primary port data input pin. Conventional memory chip designs select a row specified by the row address during the row address strobe; the selected output is amplified by the sense amplifiers and written back into the selected row, hence refreshing the contents of the whole row. In our proposal, the data written back are either all ones or all zeros if the clear input signal is asserted during the row address strobe. Using conventional memory design, 64K memory cycles would be required to clear/set all the bits. With the proposed design, only 256 memory cycles (each with a unique row address) would be required.

A printer memory system could make use of this fast clear by clearing the row in the memory array immediately after loading it into the row buffer.

• *Bit/byte addressable memory systems*

The IBM System/370 architecture allows the CPU to manipulate eight-byte words from main memory which are not aligned to word boundaries in main memory (i.e., their addresses do not have to be a multiple of eight). This feature is typically implemented by making the CPU access two words from main memory and then rotating and masking them to extract the desired word. Using the address incrementer and the multiple input/output pins, the memory system can be made to provide words directly at arbitrary byte boundaries.

Features similar to System/370 byte addressability are found in numerous other computer architectures. With the use of the enhanced memory system, they can be extended to provide bit addressability. Data fields can be placed in memory without worrying about bit or byte alignment, and hence higher data density can be achieved.

• *DMA from/to second port*

Using such dual-ported memory chips, we can implement a radically different computer system in which the traditional system bus has an additional serial path which increases the bandwidth of the bus and services high speed I/O devices such as APA displays, printers, networks, and secondary storage systems.

We assume a computer system with the following general design: The system board contains the main processer, the Direct Memory Access (DMA) controller, the address relocate tables for virtual memory, and bus arbitration logic to arbitrate the system bus between the main processor and the DMA controllers. Connected to the system bus are memory cards which contain the main system memory (implemented using the quasi-dual-ported memory chips), and the I/O controllers which control individual I/O devices.

All memory cards would use the quasi-dual-ported memory chips and would provide both a randomly addressable primary port and a serial secondary port which reads and writes data serially. The secondary port is driven by an external clock supplied by the device which is writing into (or reading from) the serial port at its own individual clock rate. These two ports, primary and secondary, can be independently and simultaneously accessed. For instance, a serial I/O device can be filling the row buffer via the secondary port, using its own clock to serially shift the row buffer at its individual rate, while the CPU is randomly accessing instructions or data via the primary port, at the primary port rate. The primary port would be fully available during this period, and many accesses could be made to this port before the serial row buffer was filled. When the buffer was full, one primary cycle would be required to load (write) the buffer into the memory array starting at the row address supplied on the address bus. Hence, totally asynchronous, simultaneous block transfers could be made with minuscule effect on the primary port availability.

In order to load the row buffer into the memory array, the primary port address bus must be used as well as some additional control signal. We would use a separate signal for this, which we call *serial select*. When the serial select signal is on, the read or write memory cycle occurs to or from the shift register (row buffer) from or to the addressed row. The lower order eight bits of the address (i.e., the column address) are not used in such a memory cycle because 256 bits are simultaneously loaded into each chip's shift register.

Both the processor and the DMA controllers would use the randomly addressable port to address main memory. Either of them could read or write the block accessible through the serial port by asserting the serial select signal. The I/O device controllers would request the DMA controllers to read or write the serial block by asserting the usual DMA request signals. So, in fact, the DMA controllers perform the usual function, except that their intervention is required once for each block transfer, rather than for each word or byte transfer. In a simplified and cheaper system design, the block transfer could be controlled by the processor by using interrupt requests and acknowledges.

All device controllers would benefit from the serial bus by being able to match the speed of their transfers with the

speed of the serial port. The clock control for such devices should hence be distributed. All I/O devices could then transfer data to and from anywhere in main memory, obviating the need for a separate buffer. An example of I/O subsystem designs which relied on such buffers is the use of frame buffers for display memory. With the use of the serial port, the display image can be located anywhere in the system memory and the CRT refreshed therefrom. Other fast I/O devices like disk and network controllers also rely on large buffers to allow for the speed mismatch between the I/O devices and the system buses.

8. Conclusions

This paper has presented an alternative architecture for dynamic random access memory chips, an architecture mainly motivated by displays but useful for other devices as well as main system memories.

9. Acknowledgments

We would like to acknowledge the contributions of William Bogholtz, Paul Chung, Dennis McBride, and Daniel Ostapko in this work.

10. References

1. J. H. Terlet, "The CRT Display Subsystem of the IBM 1500 Instructional System," *AFIPS Conf. Proc.* **31**, Fall Joint Computer Conference, 1967.
2. D. Ophir, S. Rankovitz, B. J. Shepherd, and R. J. Spinard, "BRAD: The Brookhaven Raster Display," *Commun. ACM* **11**, 415 (1968).
3. T. E. McCracken, B. W. Sherman, and S. J. Dwyer III, "An Economical Tonal Display for Interactive Graphics and Image Analysis Data," *Comput. Graph.* **1**, 79–94 (1975).
4. R. F. Sproull, I. E. Sutherland, A. Thompson, S. Gupta, and C. Minter, "The 8 × 8 Display," *ACM Trans. Graphics* **2**, 32–56 (January 1983).
5. S. Gupta, "Architectures and Algorithms for Parallel Updates of Raster Scan Displays," *Technical Report,* Computer Science Department, Carnegie-Mellon University, Pittsburgh, 1981.
6. D. T. Ling, "Mapping from Bits in Memory to Pels on Screen: Design Considerations," file memo available from author, 1981.
7. R. E. Matick, D. T. Ling, D. J. McBride, and F. H. Dill, "General Bit Manipulator: VLSI Memory and Logic Chips for Graphic/Text Applications," internal report available from author, May 1980.

Received September 30, 1983; revised March 19, 1984

Frederick H. Dill *IBM General Technology Division, East Fishkill Facility, Hopewell Junction, New York 12533.* Dr. Dill is manager of the Information Architecture Department at East Fishkill. He received a B.S. in physics and an M.S. and a Ph.D. in electrical engineering from Carnegie Institute of Technology, Pittsburgh, Pennsylvania, in 1954, 1956, and 1958, respectively. He joined the IBM Research Division in 1958 and has worked on solid state devices and technologies ranging from tunnel diodes and injection lasers to photolithography and automated measurement techniques. In 1968 to 1969 he was a Mackay visiting lecturer at the University of California in Berkeley and in 1974 he was a visiting lecturer at the Massachusetts Institute of Technology, Cambridge. From 1976 to 1983, he was involved in research on information display devices and VLSI design as the manager of the Display Technology Department. In 1983 he left the Research Division to work on manufacturing of semiconductor devices. Dr. Dill is a Fellow of the Institute of Electrical and Electronics Engineers and a past president of the IEEE Electron Devices Society.

Satish Gupta *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Gupta is the manager of the display architecture group in the Computer Science Department at the Thomas J. Watson Research Center. His research interests are currently focused on computer displays and the study of both software and hardware techniques to provide higher-performance graphics displays. He received a B.Tech. degree from the Indian Institute of Technology, Kanpur, in 1977, and the M.S. and Ph.D. degrees in computer science from Carnegie-Mellon University, Pittsburgh, Pennsylvania, in 1979 and 1981, respectively. Dr. Gupta is a member of the Association for Computing Machinery and its Special Interest Group in Graphics.

Daniel T. Ling *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Ling joined IBM at the Thomas J. Watson Research Center in 1979. He is currently manager of exploratory VLSI design and is involved in the design of a high performance microprocessor. His interests are in the areas of microprocessor and display design, as well as VLSI design tools and methods. Dr. Ling received his B.S., M.S., and Ph.D. in electrical engineering from Stanford University. While at Stanford, Dr. Ling was a Fannie and John Hertz Foundation Fellow. He is a member of the American Physical Society, the Institute of Electrical and Electronics Engineers, and Tau Beta Pi.

Richard E. Matick *IBM Research Division, P.O. Box 218, Yorktown Heights, New York 10598.* Dr. Matick received his B.S., M.S., and Ph.D. in electrical engineering from Carnegie-Mellon University, Pittsburgh, Pennsylvania, in 1955, 1956, and 1958, respectively. He joined IBM in 1958 and worked in the areas of thin magnetic films, memories, and ferroelectrics. As manager of the magnetic film memory group from 1962 to 1964, he received an IBM Outstanding Invention Award for the invention and development of the thick film read-only memory. Dr. Matick spent six months at IBM United Kingdom in Hursley developing a read-only memory for the System/360 applications. He joined the technical staff of the IBM Director of Research in 1965 and remained until 1972, serving in various staff positions which included responsibility for the Research Division plans and technical assistance to the Director of Research. In 1972 he took a sabbatical to teach at the University of Colorado and at IBM in Boulder, Colorado. In the summer of 1973, he taught at Stanford University. Dr. Matick is currently working in the areas of VLSI functional memory chip and microprocessor design. He is the author of the books *Transmission Lines for Digital and Communication Networks* and *Computer Storage Systems and Technology.* He is a member of Eta Kappa Nu and the Institute of Electrical and Electronics Engineers.