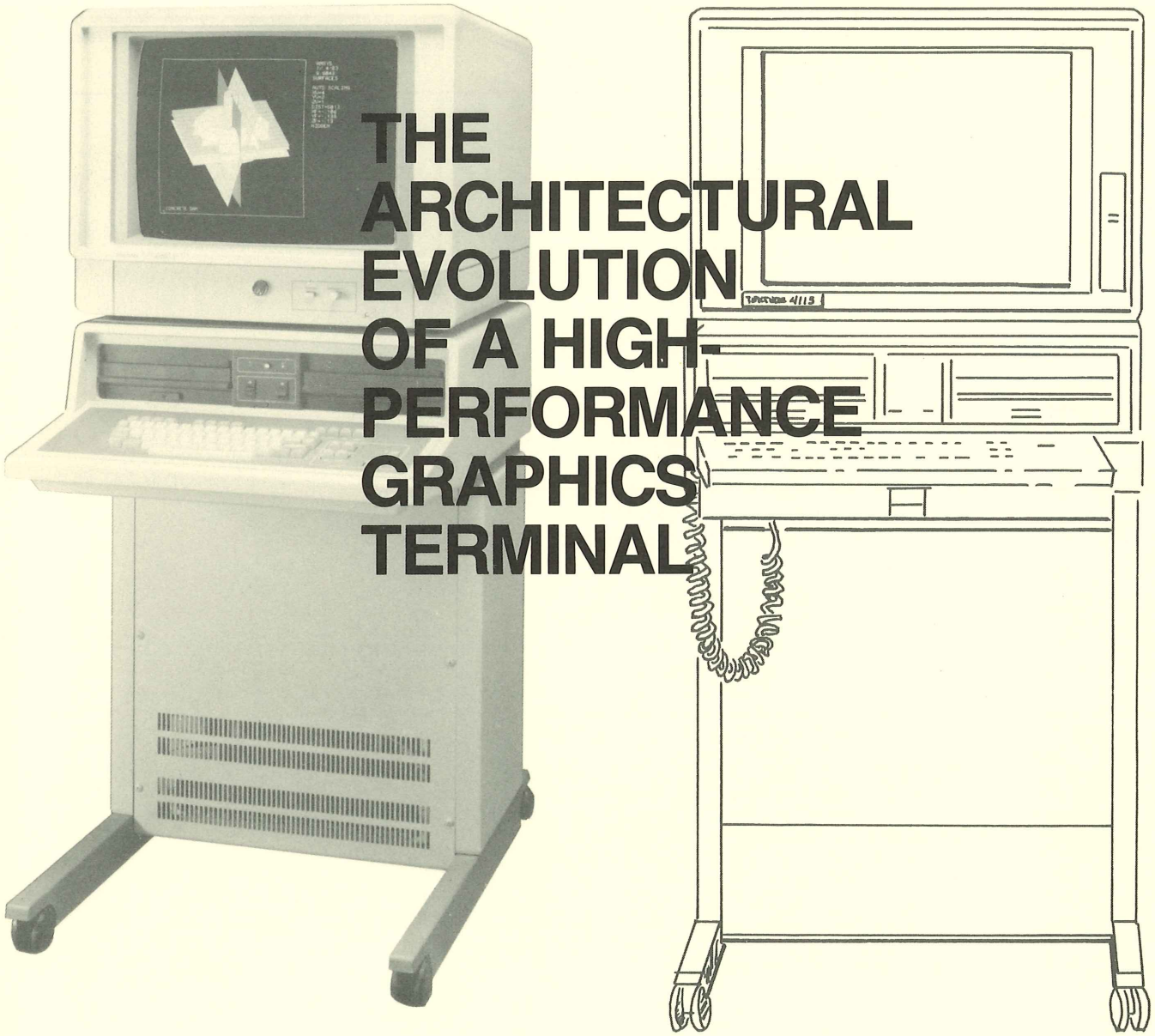
TEK INFORMATION FOR TECHNOLOGISTS

MARCH 1984

TECHNOLOGY report

COMPANY CONFIDENTIAL



THE ARCHITECTURAL EVOLUTION OF A HIGH-PERFORMANCE GRAPHICS TERMINAL

Tektronix®
COMMITTED TO EXCELLENCE

CONTENTS

Volume 6, No. 3, March 1984. Managing editor: Art Andersen, ext. MR-8934, d.s. 53-077. Cover: Jackie Miner; Graphic illustrator: Nancy Pearen. Composition editors: Jean Bunker and Sharlet Foster. Published for the benefit of the Tektronix engineering and scientific community.

Why TR?

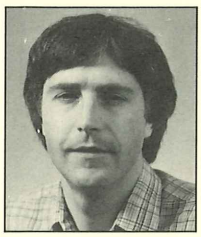
Technology Report serves two purposes. Long-range, it promotes the flow of technical information among the diverse segments of the Tektronix engineering and scientific community. Short-range, it publicizes current events (new services available and notice of achievements by members of the technical community).

CORRECTION

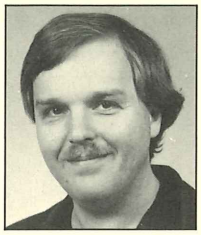
Micheal Cranford, Electronic Systems Lab.

In the February issue of *Technology Report,* we incorrectly identified Micheal Cranford, a co-author of "Perceived Brightness and Color Contrast of Color Displays," as a member of the Imaging Research Laboratory. Mike is a senior engineer in the Electronic Systems Laboratory, but he has made substantial contributions to the work of the Imaging Research Laboratory and to other organizations. □

THE 4115B:
THE ARCHITECTURAL EVOLUTION OF A HIGH-PERFORMANCE GRAPHICS TERMINAL



Douglas J. Doornink is a senior electronic engineer in Graphic Systems Products (GSP). He was the leader of the hardware development project for the 4115B. After joining Tek in 1974, he participated in the design of the 4112, 4027, 4025, 4024, and 4081. Doug holds an MSEE from Stanford University. His BSEE is from the University of Washington.



John C. Dalrymple is a senior electronic engineer in Graphic Systems Products. He lead the team that developed the picture processor for the 4115B. John joined Tek in 1976 and has done research in color anti-aliasing, display-processor architecture for directed-beam refreshed displays, and display-processor architecture for intelligent oscilloscopes. John's MSEE and BSEE are from Oregon State University.

This article is a case study of the design of the Tektronix 4115B. It details the design constraints, shows how the architecture evolved from the Tektronix 4113, and shows how performance was gained by adding a microcoded picture processor and some special-purpose hardware. This article was adapted from a paper given at Spring COMPCON, 1984.

The task given the designer is usually not to design the "best" or ultimate product but rather to squeeze the most out of a design – given many constraints. These constraints range all the way from how much the customer is willing to pay to how big the product can or must be. In 1981, the GSP engineering group was asked to design a "high-performance" graphics terminal.

High performance in computer graphics requires interactivity, and interactivity hinges on the speed at which an image can be produced or changed on the screen. The system must respond to the user in the appropriate amount of time, or productivity will suffer and frustration will increase. The challenge for the designers of a high-performance graphics terminal is to squeeze out the most speed given a set of real-world constraints.

This article reviews the design of the Tektronix 4115B and, in particular, the picture processor. First, we will discuss the background and motivation behind the design; this discussion will analyze the Tektronix 4113 and its performance bottlenecks. We will then discuss the system partitioning of the 4115B and the tradeoffs made, show how the right balance of hardware and microcode can substantially increase performance, and look at how the completed design performs.

Background and Motivation

Even though most engineers would like to start each new design from the ground up, this is not the job usually given. Because most products are not designed in a vacuum, product design means specific requirements and hard constraints. Products are designed within an environment that includes other products, customer desires, and technology limitations. This was very much the case with the design of the Tektronix 4115B graphics terminal.

Functional requirements

Since the 4115B was to be a member of the 4110 family, the most important functional requirement was family compatibility. The 4110 family was modeled after the *SIGGRAPH CORE,*[1] which includes the ability to store and transform picture segments. Compatibility with all the 4110 family commands was necessary. There were raster display commands implemented in the 4112 and 4113. These commands were similar to those in the *Raster Extension* to the *Core System.*[2] Compatibility with all raster and color features of the Tektronix 4113 was necessary because it was the only color-raster member of the 4110 family.

Because the 12-bit coordinate system in the then current 4110 family was inadequate for many applications being addressed by computer graphics, a 32-bit coordinate system requirement was added to the design goals. This extension had to be compatible with the old 12-bit system, of course.

In addition, the 4115B was to be compatibile with ANSI X3.64 alphanumeric control commands. This would allow the terminal to be used with the many alphanumeric-oriented programs available. This compatability was essential. Even though the 4115B is a high-performance graphics terminal, it is frequently used for alphanumeric data entry and editing. Most users expect their $20,000 terminal to be able to do what their $500 terminal can do.

Performance requirements

Interactivity is the primary requirement of a high-performance graphics terminal. Another word for interactivity is *speed*. To be interactive in a man-machine environment, the long-standing rule of thumb was "response within two seconds."[3] This rule has been superseded because exposure to personal computers and single-user systems has increased user expectations.[4] Functions done locally on a terminal are expected to be done quickly, if not instantaneously.

On the 4110 terminals, this meant that the re-draw and manipulation of picture segments, which are stored locally in the terminal, must be done quickly. Dragging and transforming of segments should happen instantly. Picture segments of several thousand vectors are not unusual; pictures with tens of thousands of vectors are common.

Another performance requirement was high display quality. 1280 × 1024 pixels is now mandatory for a high-resolution display. For picture stability, 60-Hz noninterlaced scanning is required; anything less tires users with flicker.

A third requirement was that the new terminal must support a display of 256 colors simultaneously. This meant at least eight bits per pixel, or eight planes of display memory.

Physical constraints

The 4115B was to be an addition to a family; this predetermined physical constraints. The 4110 family was pedestal based; therefore the 4115B had to fit into the pedestal. (See figure 1(A).) The established card cage had room for only 18 circuit cards; all of the display system plus all options had to fit. We knew six circuit cards would be needed for the display memory and controller and that nine would be needed for the options common to the 4110 family. We could use only three 8.5 × 11-inch boards for the picture processor (figure 1(B)). This definitely restricted the amount of circuitry that could be used for the picture processor.

The power for the system as a whole was fixed at 500 watts. This limit was important when it came to making tradeoffs between hardware, software, and microcode in the system architecture.

Technological constraints

Because the development schedule for the 4115B was to be short, we had to use off-the-shelf parts – no custom VLSI. We also had to use as many existing 4110 circuit cards as we could. This meant staying with the processor (8086), option cards, and bus structure used in the previous 4110 terminals. This limited system architecture to 1-M byte address space and 16-bit data paths.

Bottlenecks in the 4113

The 4113 was the functional model for the 4115B, but the 4113 had some shortcomings for high-performance graphics; the most important was its picture-segment re-draw speed. The 4113 outputs constant-time vectors to the screen at about 1000 fully transformed vectors per second.

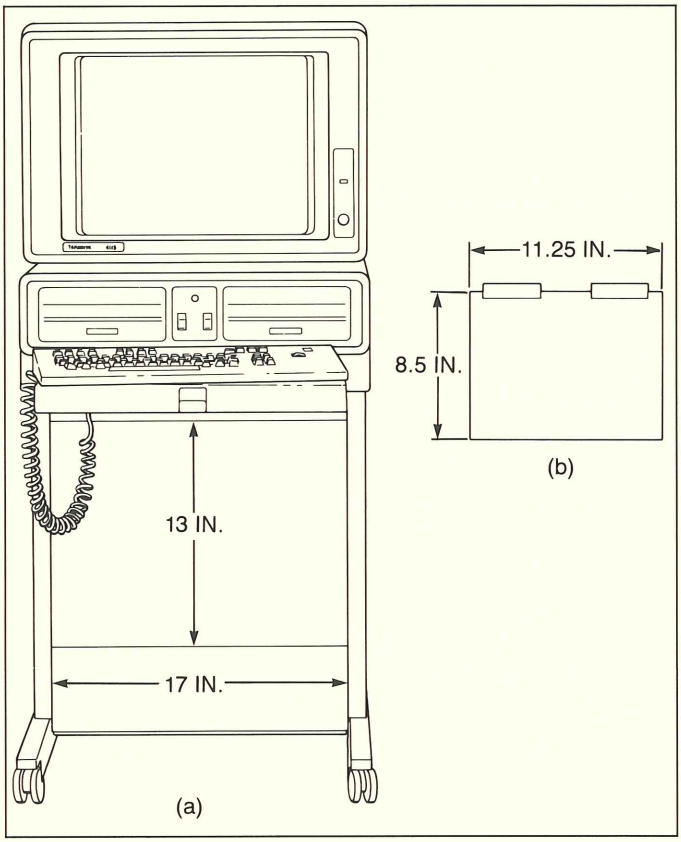


Figure 1. (a) Front view of the 4115B pedestal. The card-cage dimensions, and (b) circuit-card dimensions were the primary physical constraint for the new terminal.

Also, the 4113 couldn't keep up with sustained communications at 19.2 kilobaud. The 4113 could sustain only 9600 baud without overflowing the communications buffer.

8086 overloaded

In the 4113, there is a definite imbalance between hardware and firmware.

Although there was a hardware vector generator in the 4113, the firmware in a 8086 microprocessor did most of the work. Running in the 8086 was a message-based multitasking operating system, which also handled all host communications and peripheral management. In the graphic pipeline, 8086 firmware did all command parsing and decoding. It also did the display-list creation and traversal, 2D transformations, panel-scan conversion, and dot-matrix character setup.

The hardware vector generator was a slave to the 8086, doing only actual vector generation and controlling writing to the frame buffer. Also, the hardware performed block moves in the frame buffer for dialog-area scrolling since the dialog area in the 4113 was in the frame buffer. The hardware vector generator could do 2600 full-screen vectors per second and 100,000 short vectors per second; so it was not the bottle neck.

Data flow in the 4113

Figure 2 shows the data flow in the basic 4113. The data originates from one of three sources: the RS-232 interface, the keyboard, or the graphic input devices. After going through the command interpreter, the communication system, or the graphic input system, all data – except report data – goes to the display driver. The display driver is shown in detail in figure 3. The heart of this driver is the transform system: Ultimately, all of the data paths lead to the transform system, and the hardware is not invoked until the very end of the data flow.

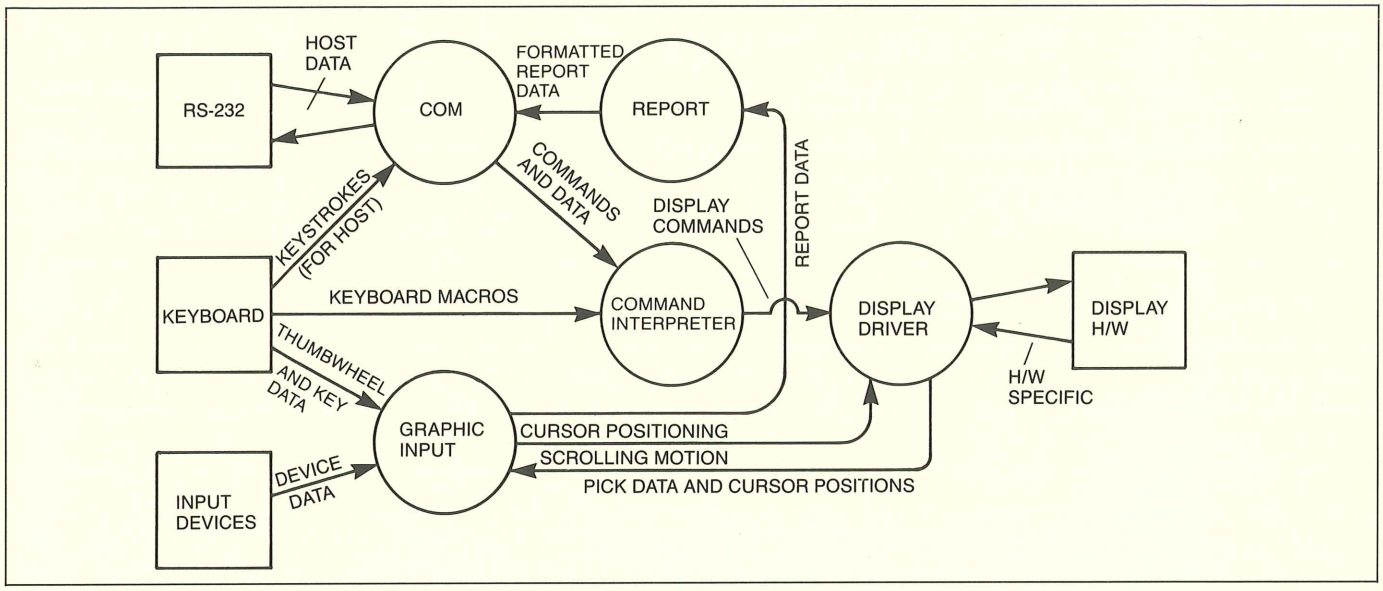


Figure 2. 4113 basic data flow. Data originates from three sources: the RS-232 interface, the keyboard, graphic input devices. The display driver is shown in detail in figure 3.

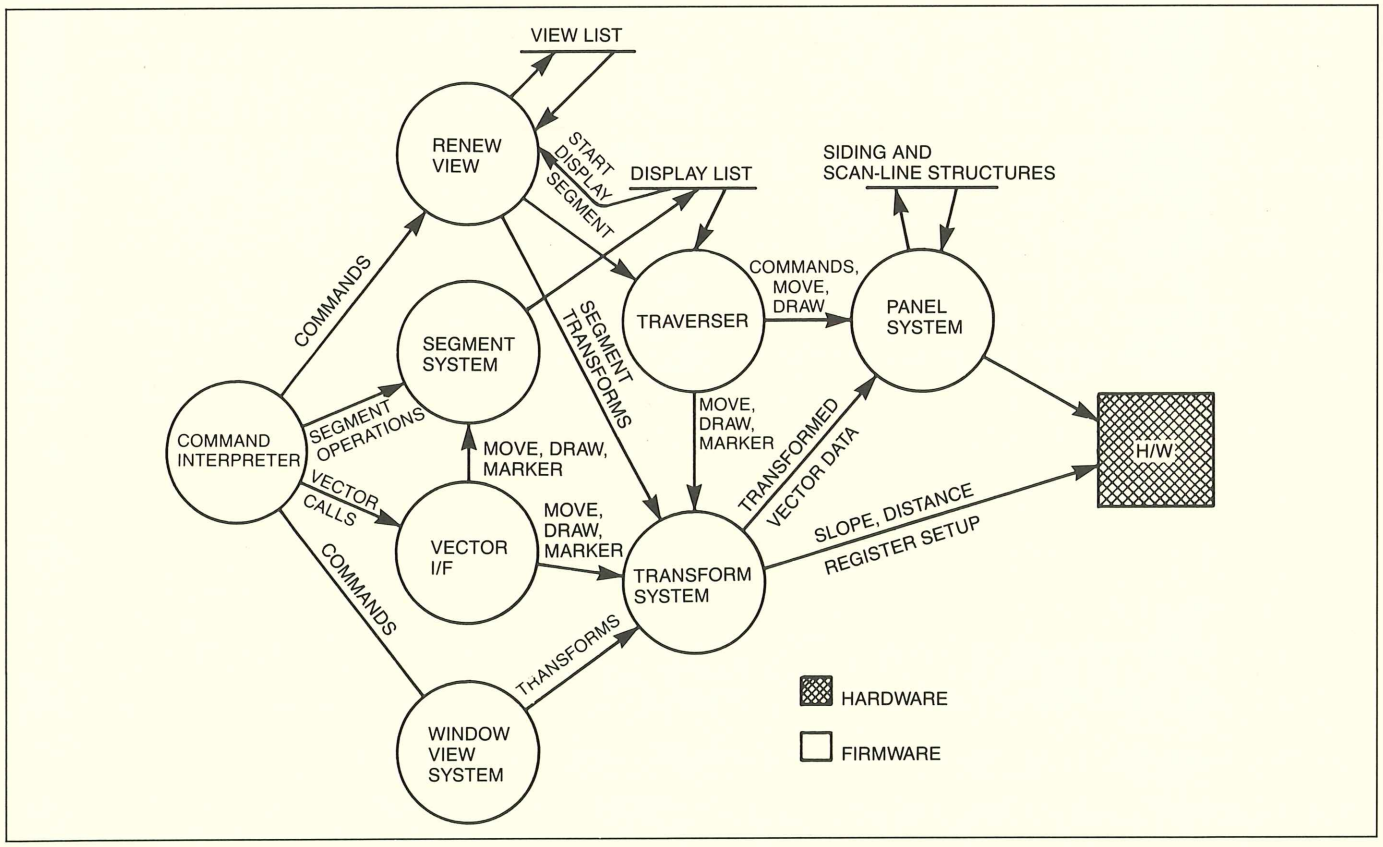
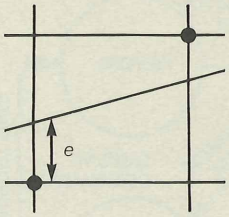


Figure 3. 4113 display-driver data flow. Ultimately, all paths lead to the heart of the system, the transform system. The hardware is a slave to the 8086, invoked only at end of data flow. There was a software/hardware unbalance.

BRESENHAM'S LINE-DRAWING ALGORITHM

Bresenham's algorithm is designed so that each iteration changes one of the coordinate values by ±1. The other coordinate may or may not change, depending on the value of an error term maintained by the algorithm. This error term records the distance, measured perpendicular to the axis of greatest movement, between the exact path of the line and the actual dots generated. In the example below, where the x axis is the axis of greatest movement, the error term *e* is shown measured parallel to the y axis. The following description of the algorithm assumes this particular orientation of the line.



At each iteration of the algorithm the slope of the line, Δy/Δx, is added to the error term e. Before this is done, the sign of e is used to determine whether to increment the y coordinate of the current point. A positive *e* value indicates that the path of the line lies above the current point; therefore the y coordinate is incremented, and 1 is subtracted from *e*. If *e* is negative the y coordinate value if left unchanged. Thus the basic algorithm is expressed by the following PASCAL program:

```
{Note:  e is real; x, y, deltay are integers}
e:=(deltay/deltax)-0.5;
for i:=1 to deltax do begin
   Plot(x,y);
   if e>0 then begin
      y:=y+1;
      e:=e-1
      end;
   x:=x+1;
   e:=e+(deltay/deltax)
end;
```

The weakness of this sequence of operations lies in the division required to compute the initial value and increment of *e*. This division can be avoided, however, since the algorithm is unaffected by multiplying *e* by a constant: only the sign of *e* is tested. Thus by multiplying e by 2Δx we produce the following program, requiring neither divisions nor multiplications:

```
{Note:  all variables are integers}
e:=2 * deltay -deltax;
for i:=1 to deltax do begin
   Plot(x,y);
   if e>0 then begin
      y:=y+1;
      e:=e+(2 * deltay -2 * deltax)
      end
      else e:=e+2 * deltay;
   x:=x+1
end;
```

A full implementation of Bresenham's algorithm involves allowing for other cases besides $0 \leqslant \Delta y \leqslant \Delta x$, the case discussed above. At the same time the algorithm can be somewhat simplified by using only integer arithmetic. Bresenham's algorithm avoids generating duplicate points. Because it also avoids multiplications and divisions, it is well suited to implementation in hardware or on simple microprocessors.

— From *Principles of Interactive Graphics,* Newman and Sproull

In addition to performing few processes by hardware, the 4113 used a low-level interface to the hardware vector generator. The firmware had to calculate all the parameters for a Bresenham vector algorithm[5] and load the parameters into registers on the vector generator. The Bresenham algorithm was then executed in the hardware.

Breaking The Bottlenecks

Adding a picture processor

To achieve the design goals of the 4115, we had to open the bottlenecks of the severely overloaded 8086 and mostly idle vector-generator hardware. We took a conventional approach:

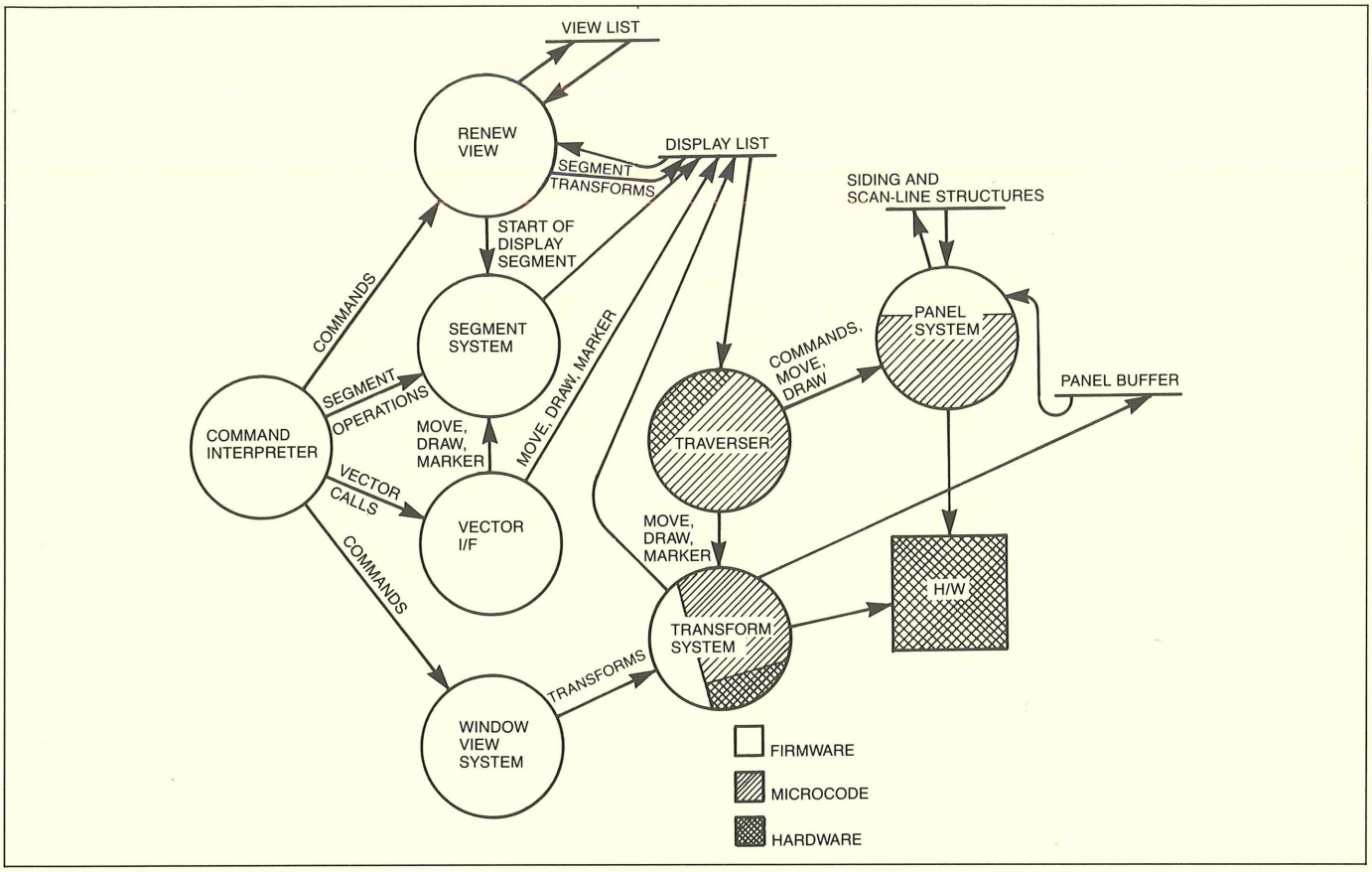


Figure 4. 4115B display-driver data flow. We offloaded lower-level graphics processing from the 8086 by replacing the vector-generator hardware with a microprogrammable bit-slice "picture processor."

replacing the vector-generator hardware with a microprogrammable bit-slice "picture processor," so that the lower-level graphics-processing functions could be offloaded from the 8086. This new division of labor can be seen in figure 4. Along with the bit slice, this processor contains several hardware accelerators for speed-critical tasks. It was constrained to fit on two standard-sized (11.25 × 8.5 in) cards.

The 4115B picture processor is an instruction-set processor that executes programs (display lists) built by code running on the 8086. The initial specification of the display-list format was done by software engineers who were designing the 8086 code and by microcoders who would be implementing the instruction set. The specification evolved as the implementation proceeded; however, task partitioning between the 8086 and the picture processor did not change drastically after the first specification.

8086 tasks

In the 4115B, the 8086 retains the multitasking operating system, host communication, peripheral management, and display-list management functions discussed in the description of the 4113. The code significantly differs from that of the 4113 in several key areas:

(1) Most data paths are now 32-bits wide to support the 32-bit coordinate space.

(2) New algorithms and data structures allow faster, more space-efficient creation of many small graphic segments.

(3) New code drives the hardware dialog overlay and cursor overlay (not present in the 4113).

(4) The numeric co-processor (8087) is used for the precise arithmetic operations needed to generate graphic-image transforms for the picture processor.

Additionally, the 8086 assists the picture processor in the scan conversion of panels (filled areas). For this task, the picture processor transforms coordinates from the display list and sends them to the 8086. The 8086 then builds temporary data structures, which it passes back to the picture processor. The picture processor uses these data structures to compute and fill the set of pixels interior to the specified area. Finally, the 8086 deletes the temporary structures.

Picture processor tasks

The picture processor executes commands from a display list resident in system memory. It transforms graphic primitives, described in a 32-bit integer coordinate space, into 1280 × 1024 pixel screen-coordinate space and clips the results to rectangular viewports on the screen. It scan converts the transformed primitives and writes pixels into the frame buffer. Using information from the display list, the picture processor controls the appearance parameters (primitive attributes such as line style, whether areas are to be filled or left hollow, background transparency of dot-matrix characters, etc.).

The picture processor can traverse a display list in one of three modes. In the *default* mode, all attribute commands are obeyed and all visible portions of primitives are drawn. In the *erase* mode, all visible portions of primitives are drawn in a solid color (fixed when the mode is entered) and the attribute-setting commands are ignored. In the *pick* mode, nothing is drawn; instead, the picture processor informs the 8086 about items that would have intersected the viewport. In this case, the 8086 has given the viewport the size and position of a very small pick aperture.

Adding hardware to the picture processor

When a user is locally panning and zooming on a retained picture, the entire picture must be re-transformed and re-drawn each time the "view" key on the terminal is pressed. A principal design goal for the 4115B was to both speed up local redraw to at least 20 times that of the 4113. It was clear that some hardware help for point transformations would be beneficial. Also, in increasing drawing speed, vectors would need much attention since terminal applications were vector intensive. Therefore, we optimized picture-processor hardware for vector-drawing speed. The following sections describe how we partitioned tasks between the 4115B picture-processor hardware and its microcode.

Display-list traversal

Since the picture processor is an independently executing processor, it must acquire the system bus and perform data transfers to and from system memory and I/O devices. In the 4115B, the details of these low-level operations are hidden from the microcode. Two hardware state machines (both resident in one registered PAL) implement the bus acquisition and data-transfer protocols.

The machines are activated by a single microinstruction. The microcode can then continue executing until it needs the results of a bus-read or until it tries to start another bus operation. At that time, a hardware "wait" mechanism temporarily halts the picture processor until the original cycle has been completed. Thus, microcode does not have to test any status flags to see if a transfer has been completed prior to starting another transfer.

The terminal bus has 20 address bits (referencing 1 Mbyte) and 16 data bits. Maximum bus bandwidth is obtained by performing 16-bit transfers on even-address boundaries. The picture-processor instruction set (display list) consists of one- and two-byte opcodes, with operand lengths ranging from zero to tens of bytes.

Most operands are immediate data following an opcode. Therefore, a display-list fetch-ahead system was built in microcode. Routines needing operands from the display list make subroutine calls to this system, which manages a three-byte fetch-ahead queue and always does 16-bit bus transfers at even addresses.

If the execution of an opcode references data from outside the instruction stream, another set of subroutines is used, which destroys information in the queue. These may be eight- or 16-bit transfers at arbitrary addresses. The queue state must be restored (by calling a subroutine) before the next opcode is fetched. Transfer-of-control opcodes flush and refill the queue.

2D point transformation

To assist in transforming points from 32-bit terminal-coordinate space to screen-coordinate space, we added low-cost serial/parallel multiplier hardware to the picture processor. This hardware consists of two 24-bit shift registers linked in a ring with two 25LS14 chips,[6] which are controlled by a hardwired state machine and a combinational PAL.

Like the bus-data transfer circuitry, this hardware is activated by, and can operate in parallel with, microcode. Data items are loaded by microcode into the shift registers and the multiplicand is input to the 25LS14s. Then, a control register is loaded with a shift-count value.

When the control register has been loaded, the state machine automatically switches the clock period for the 25LS14s and shift registers from 163 nsec/cycle to 65 nsec/cycle and begins shifting and decrementing the shift counter. During this time, any microinstruction that attempts to access or change the data in the 25LS14s, shift registers, or control register will be held off by the wait mechanism described previously.

The shift operation is completed when the shift counter reaches zero, and the result of the multiplication is read by microcode from the shift registers. Up to 48×48-bit multiplications are performed using multiple passes and partial-product accumulation in microcode.

Scan conversion

Scan conversion is the process of converting primitive descriptions (such as vector endpoints or polygon vertices) together with attribute information (such as line color or area-fill pattern) into the set of pixel addresses in the frame buffer to be modified and into the pixel data to be written at those addresses.

The 4115B includes special frame-buffer-interface (FBI) hardware, resident on one standard-sized card. It is designed to hide the details of frame-buffer memory organization from the picture processor. The frame buffer appears as a 2D array of 8-bit pixels. Also included in the FBI hardware are X- and Y-address registers (both registers can be incremented, decremented, held, or loaded on each cycle) and two sets of pixel-data registers. Either set of registers can be selected on each microcycle. A four-pixel cache with automatic swapping hardware contains a copy of the current frame-buffer region being accessed by the picture processor.

The FBI is augmented by additional hardware on the picture processor for boosting vector performance. The control signals that drive the FBI can come either directly from the current microinstruction (during vector setup) or from a 32-deep FIFO memory that queues up address-stepping and data-register-selection commands for the FBI. These commands are used during the actual drawing of vectors. The address-stepping commands trace out the trajectory of the vector, and the data-register-selection commands select between foreground and background colors for dashed lines.

During vector drawing, special hardware around the bit-slice processor allows the inner loop of Bresenham's algorithm[5] to execute in a single 163-nsec cycle. During each cycle of this loop, a bit is generated that selects one of two inputs to the FIFO pipeline: (1) step the FBI address along the long axis of the vector, or (2) step diagonally one unit in the direction of both axes.

Simultaneously, a bit from the dash pattern (stored in the same shift registers used for multiplication) is loaded into the FIFO pipeline, and the shift registers are rotated one bit position. As long as the FIFO is not full, it will accept input at the full 163-nsec/pixel rate. The FIFO is emptied by the FBI at an average rate of about 1 μsec/pixel.

When the FIFO fills up, the wait mechanism holds off further inputs until the FBI has unloaded the FIFO. Also, the wait mechanism prevents direct microcode access to the FBI as long as the FIFO is not empty. Because of the FIFO, the fetching and setup of a vector can be overlapped with the writing of the previous vector's pixels into the frame buffer.

For scan conversion of solid-filled areas, up to 80 pixels at a time (along a scan line) are written into the frame buffer. When an area is to be filled with a pattern consisting of two colors, the FIFO and the two pixel-data registers are used. When an area is filled with a general fill pattern (arbitrary size up to the limit of system memory and up to 8 bits per pixel), there is no hardware help.

Three area-fill algorithms are implemented in the microcode:

(1) A speed-optimized rectangle-fill algorithm with trivial rejection and clipping for rectangles whose sides are vertical and horizontal.

(2) A microcode-only algorithm for unclipped polygons with up to 16 sides.

(3) A general panel algorithm for areas that may be clipped, may have holes in them, and are unrestricted as to the number of edges.

In the general panel algorithm, the picture processor passes transformed vertices to the 8086 (through a shared buffer in system memory) and the 8086 builds data structures for the microcode to use when it fills the panel. Because of the handshaking overhead the general panel algorithm has the slowest panel-filling performance.

System Performance

It is not enough to say "the system must be fast" and use that statement as a design goal. To know if the design is good enough, one must have numbers against which design-goal achievement can be measured. Early in the design of the 4115B, we established performance metrics that we felt were appropriate in light of the application targets for the 4115B. Our metrics dealt with drawing speed: vectors per second, segments per second, and simple panels per second. All of these metrics assumed the application of a 2D transform as part of the drawing process.

After analyzing sample pictures from typical applications, we determined that the average vector length was 10 pixels on a 1280 x 1024 display. Pictures in this category averaged 30,000 vectors; pictures with longer vectors typically have fewer vectors. The worst-case application had only one vector per segment, while in the best case, all vectors would be in one segment. Both cases needed benchmarking because each picture segment incurs significant picture-processor overhead.

Another important picture type uses "simple panels" – simple panels have fewer than 16 edges. A prime example of simple-panel use is in a solids model that uses a mesh description and generates the image with a lot of quadrilaterals. For these images, the quadrilaterals have an area of about 100 pixels. In other applications, such as VLSI CAD, the areas to be filled are even simpler – rectangles whose sides are vertical and horizontal. For CAD applications, we benchmarked rectangle-fill performance using rectangles with an area of 100 pixels.

Comparing Performance: 4115B vs. 4113

Figure 5 compares the 4115B with the 4113 using the previously defined metrics. The performance gained by adding a microcoded picture processor is dramatic. The line-drawing performance gained by adding the FIFO is shown in figure 6.

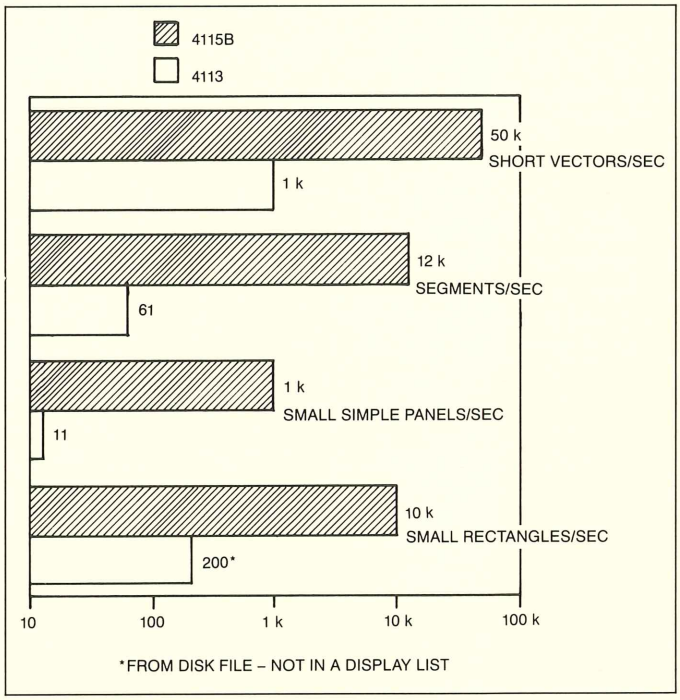


Figure 5. 4115B versus 4113 performance. Scale is logarithmic. Performance metrics assume that graphic primitives are repainted from retained segments.

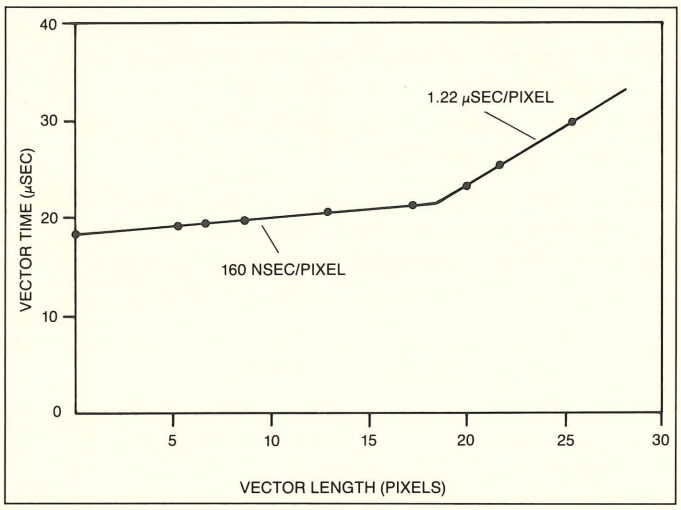


Figure 6. 4115B vector drawing performance. The effect of having a FIFO between the picture processor and the frame buffer can be clearly seen. The initial slope is 163 nsec per pixel (microcode instruction time). The final slope is 1.22 msec, the worst-case rate for writing pixels to the frame buffer.

From figure 6, one can see that the basic setup time for a vector, including the transform time, is about 18 μsec. The initial slope is 163 nsec per pixel, the microcode instruction time. As the FIFO becomes more than half full, the output port has priority and we see the knee in the curve. The final slope of the graph is 1.22 μsec per pixel, which is the worst-case rate at which the pixels can be written into the frame buffer.

It is also interesting to compare the performance of the 4115B with that of other architectures and implementations. Four architectures are compared in figure 7. The 4113 is at one end of the spectrum, an example in which a single microprocessor does most of the work. We included the Apollo DN420 as an example of a graphic system having a microprocessor and hardware optimized for BITBLT-type functions. The third example is the 4115B, in which the work is divided between a microprocessor and a microcoded picture processor. The final example is the Seillac-7 in which the work is done by a pipeline of a 32-bit bit-slice processor, two 16-bit bit-slice processors, two 16-bit microprocessors, a 4 × 4 matrix multiplier, a clip circuit, and a perspective circuit.

Remaining Issues

The 4115B project did not address several functional extensions and performance issues: we did not implement 3D transforms or picture-segment hierarchy, although these are natural extensions to the 4115B feature set. The main reason these features are not in the 4115B now is that our resources were limited. When we prioritized features, these two fell below the cut line.

Another functional issue is standardization, GKS[7] in particular. Tektronix supports GKS as a standard, but since the 4110 family was not an implementation of GKS, the 4115B could not implement GKS and still be a part of the 4110 family.

Some performance areas were not addressed by the 4115B project. Although one can do picture dynamics on the 4115B, it was never designed to do animation; the performance necessary for animation was beyond the scope of the 4115B.

We did not attempt more than eight planes, which would allow the display of more than 256 colors at a time. Restricting the 4115B to eight planes was necessary because of limits in the frame-buffer system – not because of picture processor capacity.

Finally, we did not optimize the system for BITBLT-type operations as has been done in some work-station architectures. The Apollo DN420 engineering work station is a good example of this architecture. Our application and system architecture required optimization toward line drawing.

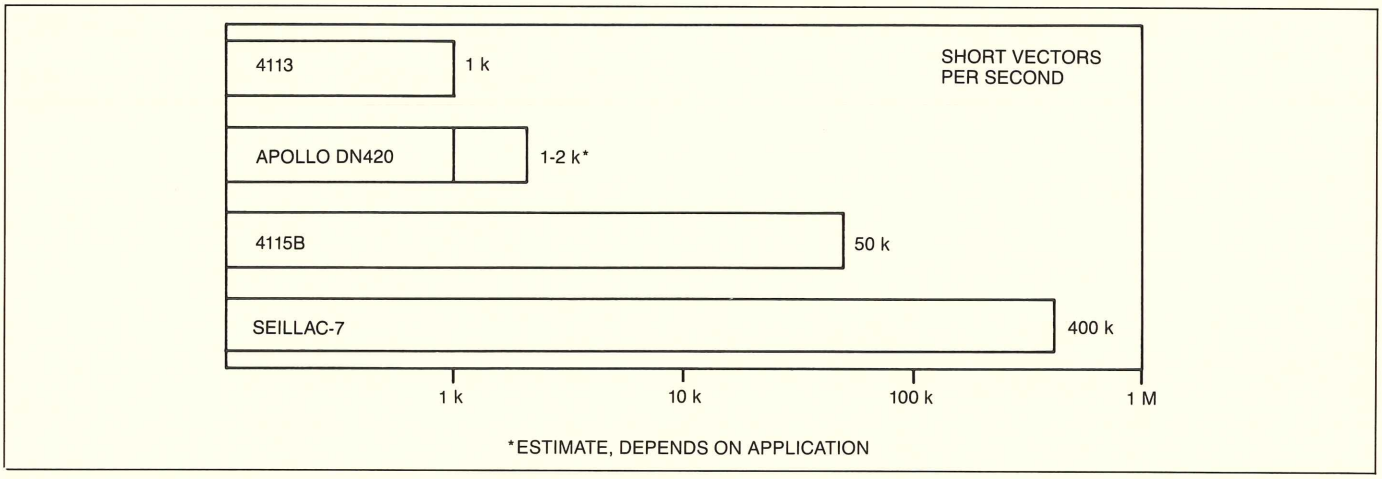


Figure 7. Vector-drawing performance of different architectures.

Conclusion

The architecture development of the 4115B was an evolutionary step rather than a revolutionary jump. The main reason for this was the set of constraints placed on the project and architecture by the product-family environment in which it was developed. In spite of this, dramatic performance improvements were made, and the requirements of interactivity were met by adding a microcoded picture processor and partitioning the work among firmware, microcode, and hardware. Because of this balancing, brute force was not necessary for high performance.

References

[1] "Status Report of the Graphics Standards Planning Committee of ACM/SIGGRAPH," published as *Computer Graphics,* vol. 11, no. 3 (Fall, 1977).

[2] "Status Report of the Graphics Standards Planning Committee of ACM/SIGGRAPH," published as *Computer Graphics,* vol. 13, no. 3 (August 1979).

[3] J. Martin, *Design of Man-Computer Dialogues,* Prentice-Hall: Englewood Cliffs, N.J. (1973).

[4] J.D. Foley and A. Van Dam, *Fundamentals of Computer Graphics,* Addison-Wesley: Reading, Mass. (1982).

[5] J.E. Bresenham, "Algorithm for Computer Control of Digital Plotter," *IBM System Journal,* vol. 4, no. 1 (1965).

[6] "Bipolar Microprocessor Logic and Interface," Advanced Micro Devices, Sunnyvale, CA (1983).

[7] Graphical Kernel Standard (GKS) International Standards Organization (ISO) Draft International Standard (DIS) 7942.

TECHNICAL STANDARDS

Technical Books Available

The Technical Standards library of technical volumes is growing. Here are some recent additions:

Mark's Standard Handbook for Mechanical Engineers, a dictionary.

IEEE Standard Dictionary of Electrical and Electronic Terms.

IEEE STD 141-Recommended Practice for Electric Power Distribution for Industrial Plants.

If your interest is in a field affected by standards, we may have the book you need. You don't have to buy; we'll lend it.

Books and Reports

IOOC '81 — Third International Conference on Integrated Optics & Optical Fiber Communication – April 27-29, 1981. This is a technical digest. A number of test procedure documents from this document are also available. May be purchased or borrowed.

NTIS — *Directory of Computer Software Applications on Energy* – Available for loan.

NASA — *A Catalog of Selected Computer Programs* – Title, number and a brief description of the program included. Available for loan.

New Standards

EIA-RS-505 — *Packaging for Return CRT Glass Component*

EIA-RS-455-47-1983, FOTP-47, *Output Far-Field Radiation Pattern Measurement,* $6.00

EIA-RS-455-51-1983, FOTP-51, *Pulse Distortion Measurement of Multimode Glass Optical Fiber Information Transmission Capacity,* $6.00

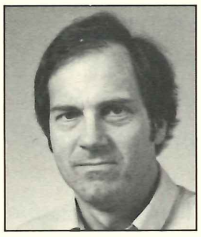
EIA-RS-455-87-1983, FOTP-87, *Fiber Optic Cable Knot,* $5.00

ANSI X3.99-1983, *Optical Character Recognition (OCR)* — Guidelines for OCR Print Quality, $6.00

ANSI X3.103-1983, *Unrecorded Magnetic Tape Minicassettes for Information Interchange,* Coplanar 3.81 mm (0.150 in) $6.00

Information regarding standards, publications, and workshops can be obtained by contacting Technical Standards, 627-1800, Leah D'Grey.

CIRCUIT SIMULATION ON A PERSONAL COMPUTER



Brian L. Biehl is a senior SW/HW engineer in Integrated Circuits Engineering (ICE), part of the Technology Group. Brian joined Tek last year from the Harry Diamond Laboratories in Washington, DC. Brian holds a PhD and BS in electrical engineering from the University of Maryland. His MSEE is from Catholic University.

A simulator program such as SPICE2 running on a desktop computer at IBM-mainframe speeds would be most desirable. Although such desktop technology is not yet here, even an 8-bit personal computer can perform SPICE-like simulations fast enough to be considered interactive. This article examines just what performance can be achieved with a circuit-simulation program running on an 8-bit-microprocessor-based system and compares that performance with simulations run on 16- and 32-bit computer systems.

Using small computer systems for circuit simulation is not new. Simulation programs such as BIAS-D[1] have been available for desktop calculators since 1972. The main disadvantage of these programs was not in the programs themselves but in the computer systems on which they executed. These computers were memory, speed, and language limited. The largest memories available were 16-Kbytes of RAM, with cassette tape as the storage media. Many tradeoffs were required just to get a 10-node circuit to execute in a reasonable time.

It was not until the mid 1970s that virtual-memory minicomputer systems such as the PRIME 350 were available, allowing more than 64-Kbytes of RAM memory to be addressed without overlay or segmentation. The virtual memory coupled with a fast cache memory increased throughput to where these $100K to $200K minicomputers were comparable to the large mainframe computers in speed and memory-address space. These systems have grown into the PRIME 750s and the DEC VAXs of the 1980s.

In the late 70s came the "home computer" – the personal computer or "PC." These computers contained 8-bit microprocessors and used audio cassettes for program storage. The cassettes were soon replaced by 80- to 100-Kbyte 5-1/4-inch floppy-disk drives. These systems were still memory and speed limited, but rather than costing $20,000, they cost less than $3,000. Another favorable aspect for simulation is that the PC systems are not language limited, working comfortably in high-level languages such as FORTRAN, Pascal and C.

Now we are seeing desktop "workstations" based on 16-bit microprocessors. These desktop systems have dual- or quad-density floppies or even 5-inch hard disks and 128 Kbyte or more of addressable RAM. This environment seems attractive for computer-aided design. Just what are the capabilities of these small systems?

Bias-D, a Small-System Circuit Simulation Program

Circuit simulation program BIAS-D was first written in BASIC for the HP 9830 desktop calculator.[1] Later it was converted to Fortran and executed on the HP 1000, DEC 11/45, PRIME 400 and IBM 370/168.[3] BIAS-D was then modified to run on a Z80-based CROMEMCO system[4] with 64-Kbytes of RAM.

This modified version of BIAS-D (called Micro-BIASD) executes on any CP/M system having 48 Kbytes of memory (excluding the operating system). Program overlays on disk are not required.

The capabilities and features of Micro-BIASD are listed in Table 1. The 50-node capability is based on a 90-percent-sparse circuit with less than 75 elements. The node capabilities and element count can be adjusted depending on the available memory. The circuit elements are stored in a linked-list integer array with 16 bytes allocated to store resistors, current sources and voltage sources; 24 bytes allocated to store capacitors and inductors; 32 bytes for transistors and 40 bytes for models. (The contents of each list can be found in reference 3, page 77.) Micro-BIASD also uses sparse-matrix storage, along with its associated pointer arrays.

Nodes: *50, not including voltage sources and ground*
Element Types: *75 (total) dynamically allocated*
- Capacitors
- Inductors
- Voltage sources
- Current sources
- Bipolar transistors (npn and pnp)

Model Types (built-in):
- PUL – pulse (vi, vf, td, tr, tw, period)
- SIN – sine wave (td, vp, freq, phi)
- EXT – Any arbitrary v(t)
- NPN – Ebers-Moll (bf, br, is, va, ir)
- PNP – Ebers-Moll
- TEM – Temperature (temp coefficients for R, C, and Beta)

Analysis Types:
- DC operating points
- Transient
- Small signal frequency response

Features:
- Insert elements
- Alter elements (single value or sweep)
- Load circuit file to or from disk
- Save dc, transient or ac analyses data to disk
- Print circuit data at any time
- Graphics interface built-in

Table 1. What Micro-BIASD can do on a personal computer.

The speed-improvement techniques used in Micro-BIASD include node reordering and sparse-matrix decomposition. But preprocessed array storage, a speedup technique used on many large simulation programs – including SPICE, isn't used. Instead, the admittance values are computed "as needed."

If the preprocessed array were used, it would contain computed admittance values to be entered into the linear admittance array. For example, a resistor R is stored in the processed array as 1/R. Also required are the four pointers to where this admittance is added to the admittance array. Using this processed array only slightly increases simulation speed for circuits with less than 50 nodes. More significant for small computer systems, the processed-array technique takes more than 2 Kbytes of memory.

In BIAS-D, NPN and PNP transistor models are like SPICE level-1-bjt models without voltage-dependent junction capacitors and series collector, base, and emitter resistors. But the model does have a low-current-beta-roll-off parameter not in the SPICE level-1 model.

The data is inputed using a semi-free format similar to SPICE. The element names are limited to two characters and the nodes limited to no more than 99. An example of the input format of Micro-BIASD is given in figure 1. This is the input representation of the 24-node five-transistor circuit shown in figure 2.

Micro-BIASD is interactive therefore it can (at anytime):

- Insert circuit elements or models using an INSERT command.
- Alter models or element values using an ALTER command. Element values may be swept using ALTER.
- Save modified circuits to a disk file using a SAVE command. Saved circuits and others can be re-entered later from the disk using a LOAD command.
- Print circuits to the screen or printer using a PRINT command.

Table 2 shows how Micro-BIASD uses memory space for its most-used functions and for the operating system and the Fortran library. The memory utilization by Micro-BIASD is compiler dependent. In this case, since the system memory available for use is 64 Kbytes, 10 Kbytes are available for adding more nodes or elements, graphics (about 6 Kbytes), a MOS-transistor model, and so forth. The data in table 2 was obtained using the Microsoft Fortran 80 compiler. Coding parts of the program in Z80 assembly language would reduce memory needs and speed execution time. However, the code would be less portable.

Computer System Requirements

To execute Micro-BIASD at least this is needed:

- A CPU with 64K of RAM
- CP/M or any operating system supporting Fortran
- One disk drive of at least 81 Kbyte
- A monitor or terminal

Compiling and linking Micro-BIASD requires either two 81-Kbyte disk drives or one 320-Kbyte drive. Such a system including Fortran and CP/M software could cost less than $850 if a home-brew system such as "BigBoard" is used. Turnkey systems such as a KAYPRO cost $1200 to $1800.

```
*TEST CIRCUIT CKT13 (24 NODES)
***INTEGRATED PREAMPLIFIER***
***RESISTORS
R1 6 1 12K
R2 7 3 7.5K
R3 4 0 680
R4 7 6 9K
R5 8 0 5K
*TRANSISTORS
Q1 32 11 23 M2
Q2 34 21 43 M2
Q3 62 51 44 M2
Q4 64 61 53 M2
Q5 72 31 83 M2
*VOLTAGE SOURCES
VB + 7 0 6.1
VS 9 0 1 M1
CS 9 1 1U
*BASE RESISTORS
RB1 1 11 100
RB2 2 21 100
RB3 5 51 100
RB4 6 61 100
RB5 3 31 100
*COLLECTOR RESISTORS
RC1 3 32 100
RC2 3 34 100
RC3 6 62 100
RC4 6 64 100
RC5 7 72 100
*EMITTER RESISTORS
RE5 83 8 10
RE4 53 5 10
RE3 44 4 10
RE2 43 4 10
RE1 23 2 10
*JUNCTION CAPACITORS
CE1 11 2 2P
CC1 11 3 2P
CE2 21 3 2P
CC2 21 4 2P
CE3 51 6 2P
CC3 51 4 2P
CE4 61 5 2P
CC4 61 6 2P
CE5 31 7 2P
CC5 31 8 2P
M1 PUL 0-1 .5U .5U 5U .5U
M2 NPN 100 1 5E-15
END
```

Figure 1. Twenty-four node benchmark circuit listing for the circuit shown in figure 2.

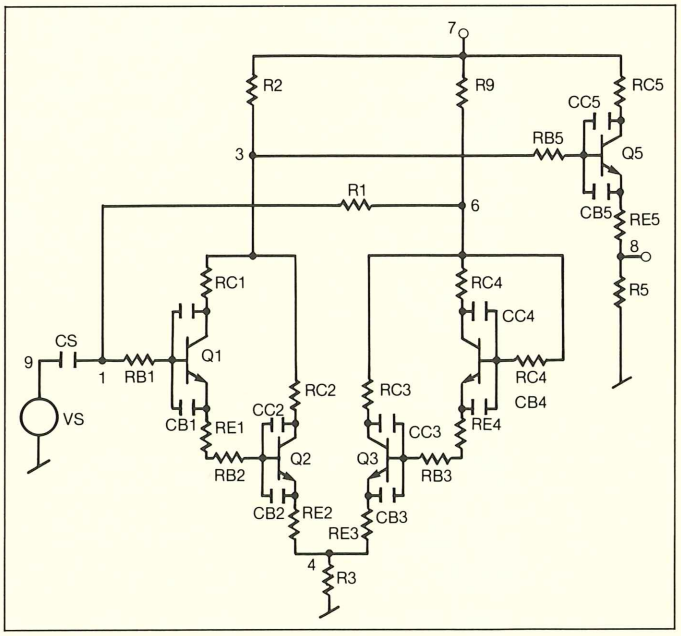


Figure 2. This five-transistor amplifier circuit with 24 nodes was simulated to evaluate the simulation possibilities of personal computers.

Function	Size (Kbytes)
Operating System (cp/m)	8.0
Common (no ac)	6.0
Ac Common	2.4
Input/output	9.8
Setup	4.9
Bjt Model	3.4
Analysis (no ac)	6.5
Ac Analysis	3.8
Fortran Library	9.2
TOTAL (Kbytes)	54.0

Table 2. Memory utilization for Micro-BIASD on CP/M Fortran. On a 64K system, 10 Kbytes are available for more nodes, graphics, and other models.

Speed Comparisons of Computer Systems Using BIASD

Since BIASD is in standard FORTRAN it should execute without modification – except for timing and disk I/O routines – on most 8-bit to 32-bit CPUs. Table 3 compares the execution times of 12 systems including a VAX 11/780 running VMS Fortran for a transient analysis of 101 timepoints for the 24-node circuit in figure 2. The total time on the VAX was 12 seconds versus 82 minutes for the CROMEMCO. The systems are listed in the approximate order of execution time. As expected, 8-bit CPUs are slower than 16-bit CPUs, which are slower than 32-bit CPUs.

Speed Differences

Table 3 shows some interesting speed-related items:

- Floating point hardware halved execution times, for both the APOLLO (16-bit CPU) and the VAX 11/780 (32-bit CPU).
- Fortran compiler efficiency significantly shortened execution speed for identical VAX 11/780s. One ran UNIX; the other ran VMS.
- SPICE2 executed about 60% slower than BIASD on a VAX 11/780 but executed about twice as fast (not shown) as BIAS-D on an IBM 370/168.[3]

The speed differences resulted from differences in hardware or software (or both). In the 8-bit CPUs, the primary limit is the double-precision multiply speed. The speed of a double-precision (64 bit) multiply on a 4-MHz Z80 (MicroSoft Fortran) is 10 ms; on the VAX 11/780 VMS Fortran, it is 2 μs.

The compiler determines how efficiently hardware interfaces with software. Efficiency also depends on the programmer's skills, how well the machine-instruction set utilizes the hardware, and other factors. Compiler-caused speed differences can range from a factor of two to as much as four or five[3,5] (see table 3).

Operating System & Features	Time per iteration	Speed relative to VAX* *
8-bit CPU		
CROMEMCO CR2D *(CDOS, 4 MHz, Z80)*	11 s	410
TARBELL *(CP/M, 4 MHz, Z80)*	11 s	410
16-bit CPU		
APOLLO *(virtual memory, cache, 68000)*		
a) 8 MHz clock	370 ms	14
b) 10 MHz clock, f.p. hdwr. ++	140 ms	5.2
HP-21MX *(RTE III; f.p. hdwr.; 4 8-bit double precision)*	230 ms	8.5
PDP 11/45 *(RSX 11D; f.p. hdwr.)*	94 ms	3.5
HP-1000F *(RTE VI; f.p. hdwr.; 4 8-bit double precision)*	61 ms	2.2
32-bit CPU		
VAX 11/750 *(VMS 3.1; cache; virtual memory)*		
a) without f.p. hdwr.	61 ms	2.2
b) with f.p. hdwr.	37 ms	1.4
SEL 32/27 *(no f.p. hdwr.)*	58 ms	2.1
PRIME 400 *(PRIMOS; cache; virtual memory, f.p. hdwr.)*	64 ms	2.4
VAX 11/780 *(cache; virtual memory; f.p. hdwr.)*		
a) Unix *(Berkeley version 4.2)*	43 ms	1.6
b) VMS *(rev 3.2)*	27 ms	1.0
IBM 370/168 *(MVS/TSO; cache; f.p. hdwr.)*	4.5 ms	0.16
SPICE2G.6 *(on a VAX 11/780 VMS)*	43 ms	1.6

**running CKT13 (24 nodes, 5 bipolar transistors)*
***relative to a VAX 11/780 VMS FORTRAN (with optimization)*
++f.p. hdwr. = floating point hardware

Table 3. Speed comparisons of computer systems running BIASD.

Program size affects execution speed. Large programs require more paging (disk I/O is slower than RAM). This is shown by the slower execution of SPICE on a mini compared to the faster execution on a mainframe.

Interactive circuit simulation requires execution speeds of one-iteration-per-second or faster – any slower and you bore the user. The 4-MHz Z80 took 82 minutes for a 101-point transient analysis – at 11-seconds-per-iteration, this is definitely not interactive (see table 3). At one-second-per-iteration the same analysis would take 7.5 minutes – marginal yes, but tolerable.

Conclusions

A circuit simulator with SPICE-like capabilities can run on 8-bit microprocessor based systems. The primary sacrifice is execution speed. The data indicate that with floating-point hardware 8-bit systems could be interactive (less than one-iteration-per-second). However, the data also show 16-bit systems, even without the floating-point hardware, are faster.

Other observations:

- Circuit simulation programs running on small systems need to be tuned for that system to speed up execution.
- Compilers need to take advantage of the hardware and the particular microprocessor's instruction set.
- Floating point hardware is needed to speed up double-precision multiplies.
- Simulator programs should be scaled to the size of the executing system.

For More Information

For more information, call Brian Biehl 627-4073 (59-316). □

References

[1] Brian L. Biehl, "BIAS-D: A Semi-Interactive Circuit Analysis Program for Desktop Calculators and Minicomputers," Eighth Annual Asilomar Conference on Circuits, Systems and Computers, December 1974.

[2] Larry W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," Electronics Research Lab, ERL-M520, University of California, Berkeley, May 1975.

[3] Brian L. Biehl, "Interactive Electronic Circuit Simulation on Small Computer Systems," Harry Diamond Labs, HDL-TM-79-30, 120 pps., November 1979. (Available from the Defense Documentation Center, Alexandria, VA.)

[4] Brian L. Biehl and Charles W. Cairns, "BIAS-D as a Microcomputer-Based Circuit Simulator," Twelfth Annual Asilomar Conference on Circuits, Systems and Computers, December 1978.

[5] Avram Tetewsky, "Benchmarking FORTRAN Compilers," *BYTE,* February 1984, page 217.

DEVELOPING A DISPLAY MEMORY FOR HIGH RESOLUTION RASTER GRAPHICS

Greg Thompson is an electronic design engineer in Graphics Systems Products (GSP), part of IDG. Greg joined Tek in 1977. He holds a BSEE from the University of Portland and an associate degree from Mount Hood Community College.

Raster graphic terminals are demanding increasingly denser memories. This article, after tracing the evolution of raster graphics, describes the decisions and the approaches used in designing a display memory for the 4115B, a high-resolution raster graphics terminal.

The Evolution of Raster Graphics

The display devices developed in the mid-sixties and still in use today are called vector stroke or calligraphic displays. They consist of a display processor, a display buffer memory, and a CRT with its associated electronics. The buffer stores the computer-produced display list or display program which contains points, line plotting commands, and character plotting commands.

These commands are interpreted by the display processor, which converts digital values to analog voltages that displace an electron beam writing on the phosphor coating of the CRT. Since the light output of the phosphor decays in tens or at most hundreds of microseconds, the display processor must cycle through the display list to refresh the phosphor at least 30 times a second to avoid flicker.

The buffer memory required to develop and display typical line drawings (8 to 32 kilobytes) and a processor fast enough to refresh at least 30 times a second were both very expensive in the sixties. Thus in the late sixties the advent of the DVST or direct-view storage tube (which obviated both the buffer and the refresh process) was a vital step in making interactive computer graphics affordable.

In the DVST, the image is stored by writing it once on a storage target carrying a bistable storage phosphor. This type of storage tube is still popular for applications that require large numbers (tens of thousands) of precise lines
and characters but do not need dynamic picture manipulation.

A mid-seventies achievement was an inexpensive raster display based on television technology. The development that made raster graphics possible was inexpensive solid-state refresh buffer memories, considerably larger than those of a decade ago at a fraction of the price of earlier systems.

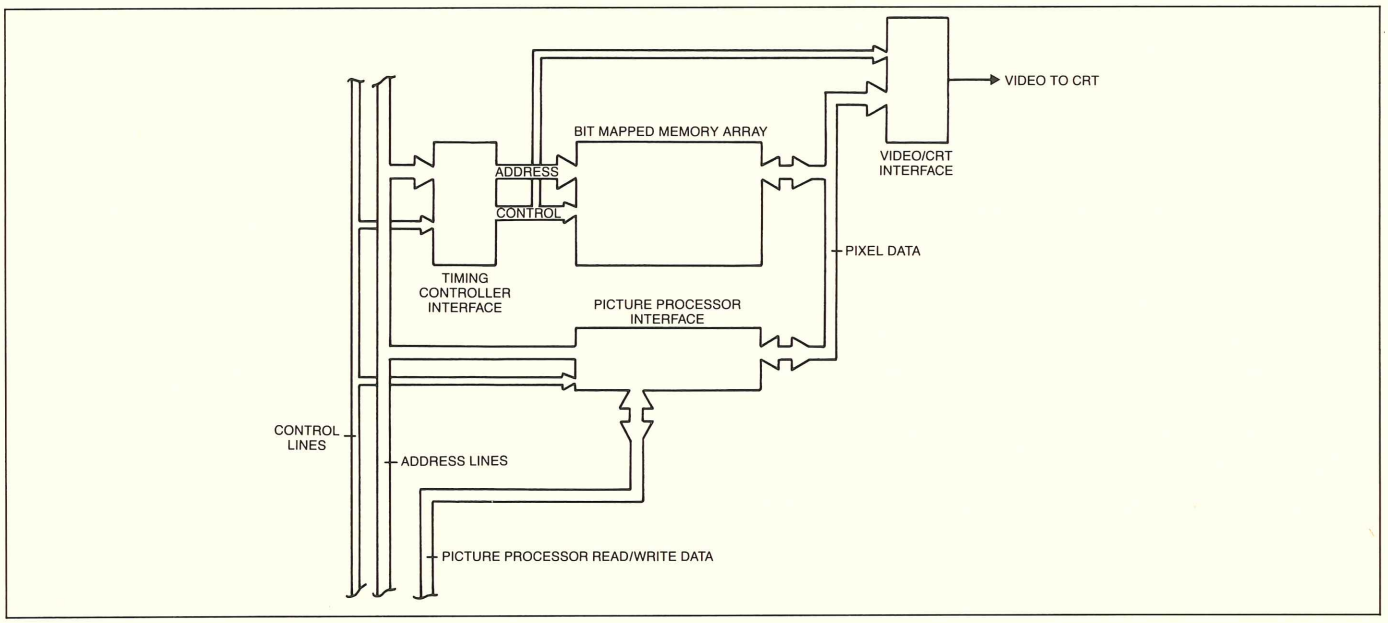


Figure 1. Fundamental bit-plane memory architecture.

Fundamental Bit-Plane Architecture

As shown in figure 1, basic bit-plane architecture consists of four blocks: a bit-mapped memory array (where the image is stored), a timing controller interface (that controls the scanning of memory array to the CRT screen), a picture processor interface (for reading and writing the image into the memory), and a video/CRT interface.

In raster displays, the display primitives such as lines, characters and solid areas (typically polygons and rectangles) are stored in a refresh buffer as individual picture elements called pixels. The image is formed from a set of horizontal raster lines, which are simply a matrix of pixels covering the entire screen.

Much more data storage is needed for raster scan displays than for refreshed vector or DVST displays. An entire image of say, 512 lines, each containing 512 pixels, must be stored in a "bit-map" memory, which contains all the points necessary to map, one for one, the points on the screen.

Timing controller interface

A raster display memory has an interface to a controller to generate the timing signals needed by both the display memory and the CRT (figure 2). This interface consists of the address lines and the memory read/write control lines. These address/control lines control the scanning sequence of the bit map and the loading and shifting of the parallel-to-serial shift register, and other control lines such as the RAS and CAS signals for dynamic RAMs.

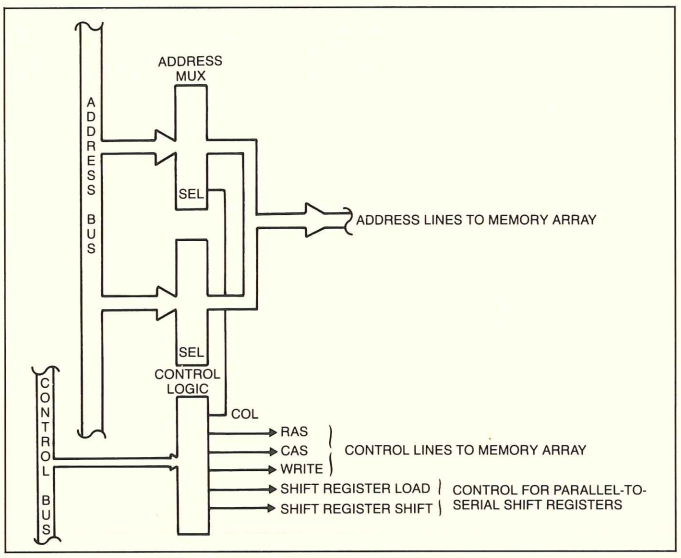


Figure 2. Address Bus interface.

Video/CRT interface

Although the frequency of access is limited by the bit-map memory's access time, pixel information in the memory is accessed fast enough to refresh the CRT at least 30 times a second. Enough pixels must be accessed and shifted from the memory to the CRT to allow time to get the next set of pixels from the display memory to be shifted out. This accessing and shifting is done by the parallel-to-serial interface.

Pixel information accessed from the memory is some *n* pixels wide (called "word width"). The *word* is latched into a parallel-to-serial shift register (figure 3). Pixel information is then either shifted out serially to the CRT as video – a sequence of pixels – or is processed further for color or gray-scale displays.

Picture processor interface

The bit-map memory also has an interface to the picture processor (figure 4). The picture-processor interface provides the path for pixels to be read or written from/to the bit map; so the picture processor can create the image that appears on the CRT. The picture processor accesses the bit map during horizontal and vertical blanking, when the CRT is not being refreshed.

The picture processor can be given more "accesses" to the bit map by cycling the display RAMs at twice the speed needed to refresh the screen. Although this increases word width, it significantly speeds memory access by the picture processor.

It is faster memory access that makes higher performance possible in a raster-scan graphics terminal.

4115 Design Goals

Design goals for the Tektronix 4115 Graphic Display Terminal were high performance, high resolution, and little or no flicker. To avoid flicker, we used a 60 Hz noninterlaced raster rather than the usual 30 Hz, interlaced format.

The resolution was to be 1280 by 1024 pixels with an ability to simultaneously display 256 different colors. We chose to go for high resolution and many colors to position the 4115 for CAD/CAM applications.

To reach the desired resolution, 1280 by 1024, 1,310,720 individual pixels per bit plane were needed. To display 256 colors simultaneously, eight bit planes would be required.

Eight nanosecond/pixel rate

Choosing the pixel rate for 60-Hz, noninterlaced operation was our first engineering decision. Scanning 1024 lines in less than 16.66 milliseconds means a time-per-line of about 16 microseconds. And 1280 pixels per line means about 12 nanoseconds per pixel. After allowing for horizontal and vertical blanking times, the actual values are 15.62 microseconds per horizontal line and 8.14 nanoseconds per pixel.

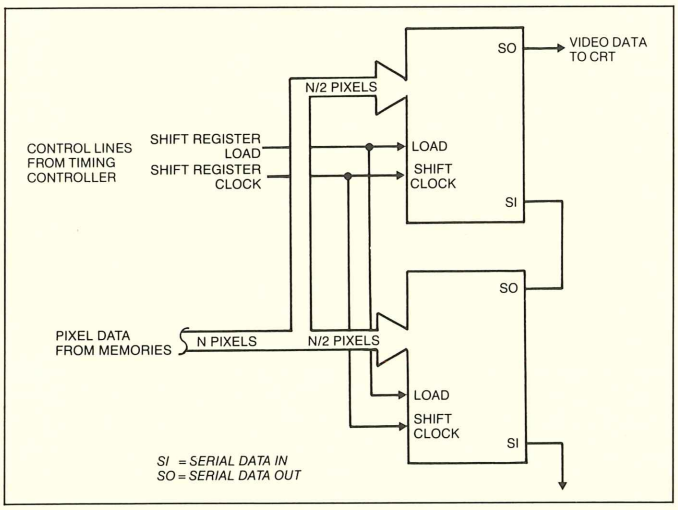


Figure 3. Parallel-to-serial shift registers, P/O Video/CRT interface.

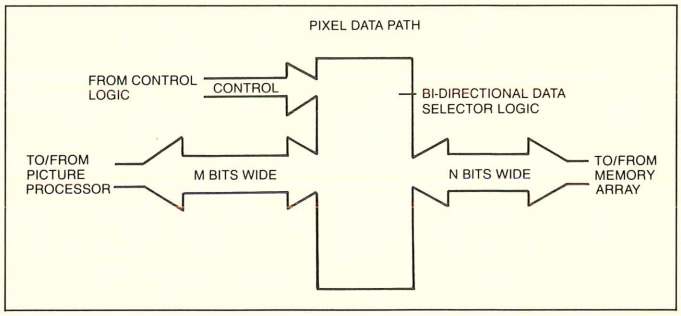


Figure 4. Picture processor interface.

Word Width

With the intended resolution of 1280 by 1024, our next step was deciding what dynamic RAMs to use. A few calculations indicated that the entire 1280 by 1024 pixels would map directly into eighty 16,384 by 1 dynamic RAMs. This choice set the word width to 80 pixels.

The 80 pixel word width and pixel rate of 8.14 nanoseconds required memory access within 651 nanoseconds. This access time meant that the memory cycle time could be cut in half to 325.4 nanoseconds – because most memories can be accessed in less than 300 nanoseconds – to give the picture processor more access time. Thus, we could achieve design goals without using faster, more expensive RAMs.

Dynamic RAM technology and the 80 pixel word width meant that each bit-plane would require 80 RAMs. Since one of the design goals was to display 256 colors at once, eight bit planes were required.

Because the 4115 was to be mounted in an existing terminal pedestal (with no change), space requirements for the eight bit planes complicated design. Finding space for eight bit planes meant fitting two planes on a single circuit board. With current dynamic RAM technology, putting 160, 16 pin ICs plus support logic on a 8.5 by 11 inch circuit board seemed impossible.

At the time of the design effort – December 1981 – the only RAMs available were 16 K by 1. However, Texas Instruments was talking about a 16 K by 4 device, but wasn't sure they could meet the necessary delivery dates.

Design solutions

The tentative design solution for the board-space problem was to develop a hybrid memory board using a ceramic, single inline-package with four 16 K by 1 RAMs. This approach created a 16 K by 4 device which solved the problem of board space – but this device cost too much to produce and did not meet reliability requirements.

So while one team worked to improve the reliability of this hybrid memory, a second design team developed an alternative memory board using the Texas Instruments TMS4416, 16 K by 4 dynamic RAM. We decided that if TI made its promised delivery dates, we would use TM4416 instead of the hybrid memory board.

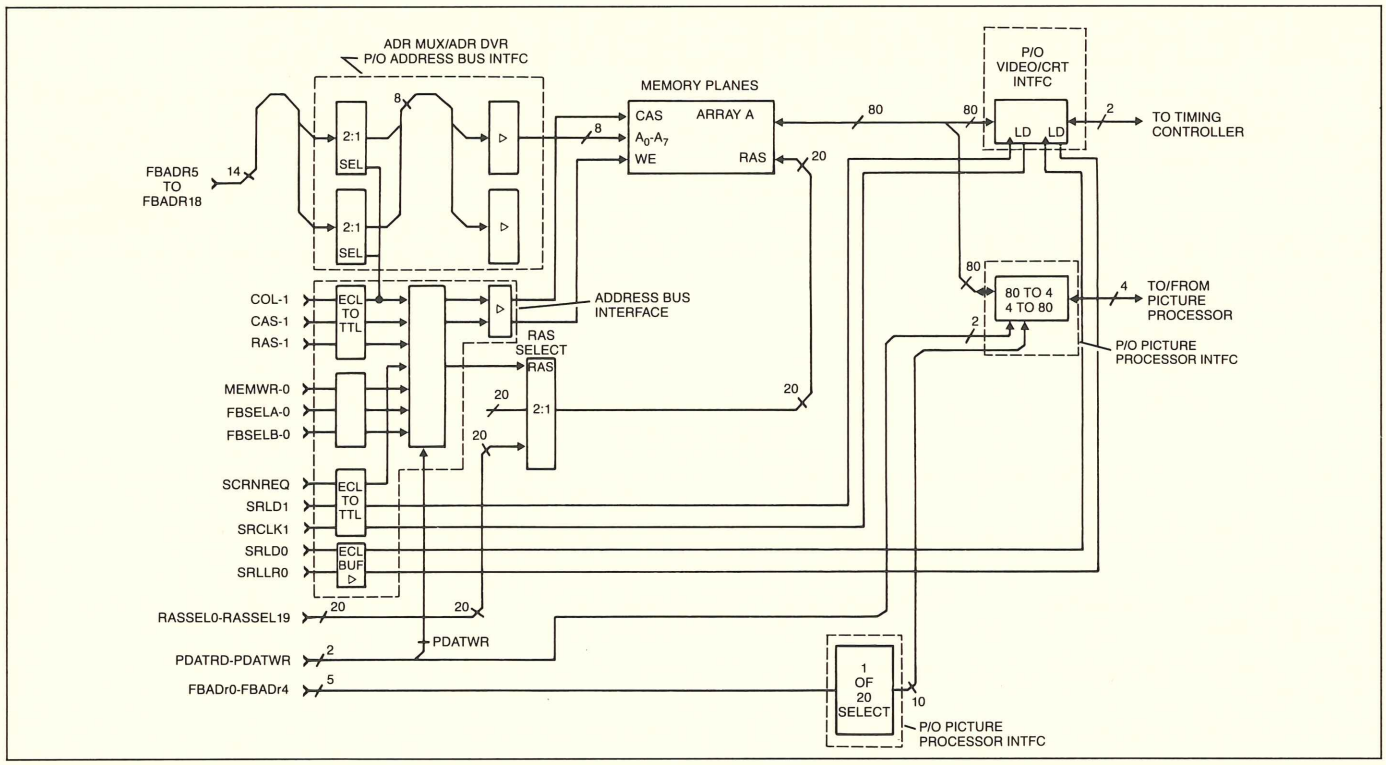


Figure 5. 4115 bit-plane memory architecture.

Well, TI came through. By using one TM4416 instead of four 16 K devices per single inline package, costs were cut more than half and memory-part count was cut by a factor of four. System costs were also lowered because fewer interconnections and support logic were required.

Also, by using the TMS4416, power consumption was significantly cut. Power supply margins were increased and electrical noise was reduced. And, with lower heat dissipation and fewer interconnections, system reliability would be improved.

4115 Bit-Plane Memory Architecture

The 4115 bit-plane memory includes three blocks described next and shown in figure 5.

Parallel to serial shift register

The parallel to serial shift register takes 80 pixels in parallel from the memories and shifts these bits two at a time to the controller.

The timing controller takes these two bits along with bits from the other bit planes and uses them as an index into two parallel color look-up tables, the output from the look-up tables is then multiplexed out to a digital to analog converter to become the video used by the color monitor.

Address-bus interface

The address-bus interface is shared by both the timing controller and the picture processor. It consists of the 19 address lines and the memory read/write control lines. The other control lines (RAS, COL, CAS) and the Shift register load and shift register clocks are generated by the timing controller. The frame-buffer-select lines are generated by the picture-processor interface to select the desired memory plane for reading or writing. This means any combination of memory planes can be read or written without disturbing any of the other memory planes.

Picture-processor interface

The picture-processor interface consists of an 80 bit to 4 bit bidirectional multiplexer. The reason for this 80 to 4 bit interface is that the picture processor reads or writes only a nibble. The nibble consists of four adjacent pixels on a horizontal line, from each of the eight planes.

Summary

In the seventies, semiconductor memory made raster graphics affordable. But as the resolution of raster graphic terminals increased so did the density of the memory array needed to support the display.

Increasingly denser memories are an on-going problem in designing today's raster graphic terminals. Higher cost per unit, lower reliability, and more circuit board space all create difficulties.

The development of an integrated circuit memory to support the raster graphics market will undoubtedly ease both design and cost constraints and contribute to the increased performance of future raster graphic terminals.

For More Information

For more information, call Greg Thompson, 685-3092, 63-205. □

PAPERS AND PRESENTATIONS

The table below is a list of papers published and presentations given during recent months.

While providing recognition for Tektronix engineers and scientists, the presentation of papers and articles contributes to Tektronix' technological leadership image.

If you plan to submit an abstract, outline, or manuscript to a conference committee or publication editor, take advantage of the services that Technology Communication Support (TCS) offers.

TCS provides editorial and graphic assistance to Tektronix engineers and scientists for papers and articles presented or published outside Tektronix and obtains patents and confidentiality reviews as required.

Call Eleanor McElwee on ext. MR-8924. ☐

DECEMBER

TITLE	AUTHOR	PUBLISHED	PRESENTED
Television – Why We Use It and How It Works	Rex Stevens	**International Television**	
A Process for Two-Layer Gold IC Metallization	Doug Summers	**Solid-State Technology**	
Recent EIA Phosphor Screen Registrations	Peter Keller	**Proceedings of SID**	
1976 CIE-UCS Chromaticity Diagram with Color Boundaries	Peter Keller	**Proceedings of SID**	
Development of a Computer-to-Computer Interface	Garey Fouts	**Proceedings of 5th International Computervision Conference**	
Graphic Workstation Requirements for Computer-Aided Design for Electrical Engineering	Robert Chew		Nikograph Conference, Tokyo, Japan
High Speed, Latchup-Free, 0.5 μm-Channel CMOS Technology Using Self-Aligned TiSi2 and Deep-Trench Isolation	Tad Yamaguchi		International Electron Devices Meeting (IEDM), Washington, D.C.
EMI Shielding with Electroless Copper	Larry Helton David Jordahl		Society of Mechanical Engineers Chapter Meeting, Portland, OR
Designing a Reconfigurable Automated Test System: A Common-Sense Approach	Thomas Gifford N.D. Gerbracht David Laib		Automated Manufacturing '83 and Test Instrumentation Conference, Brighton, England

Continued

Technology Report
MAILING LIST COUPON

☐ ADD
☐ REMOVE

Not available to field offices or outside the U.S.

MAIL COUPON TO 53-077

Name:_____ D.S.: _____
Payroll Code: _____
(Required for the mailing list)

For change of delivery station, use a directory change form.

JANUARY

TITLE	AUTHOR	PUBLISHED	PRESENTED
Spectrum Analyzer is Portable Lab Assistant	Dave Barnard	**Microwaves & RF**	
The Optical Time-Domain Reflectometer (OTDR)	Richard Osborn	**Laser Focus**	
BASIC Language Telecommunications	Bob Broughton	**Dr. Dobb's Journal**	
An Experimental Color Monitor for Vision Research	John McCormick Gerald Murch Lyle Leavitt		SPIE Conference, Los Angeles, CA
SQA Contributions to a Quality Software Product	George Tice		Reliability and Maintainability Symposium, San Francisco, CA