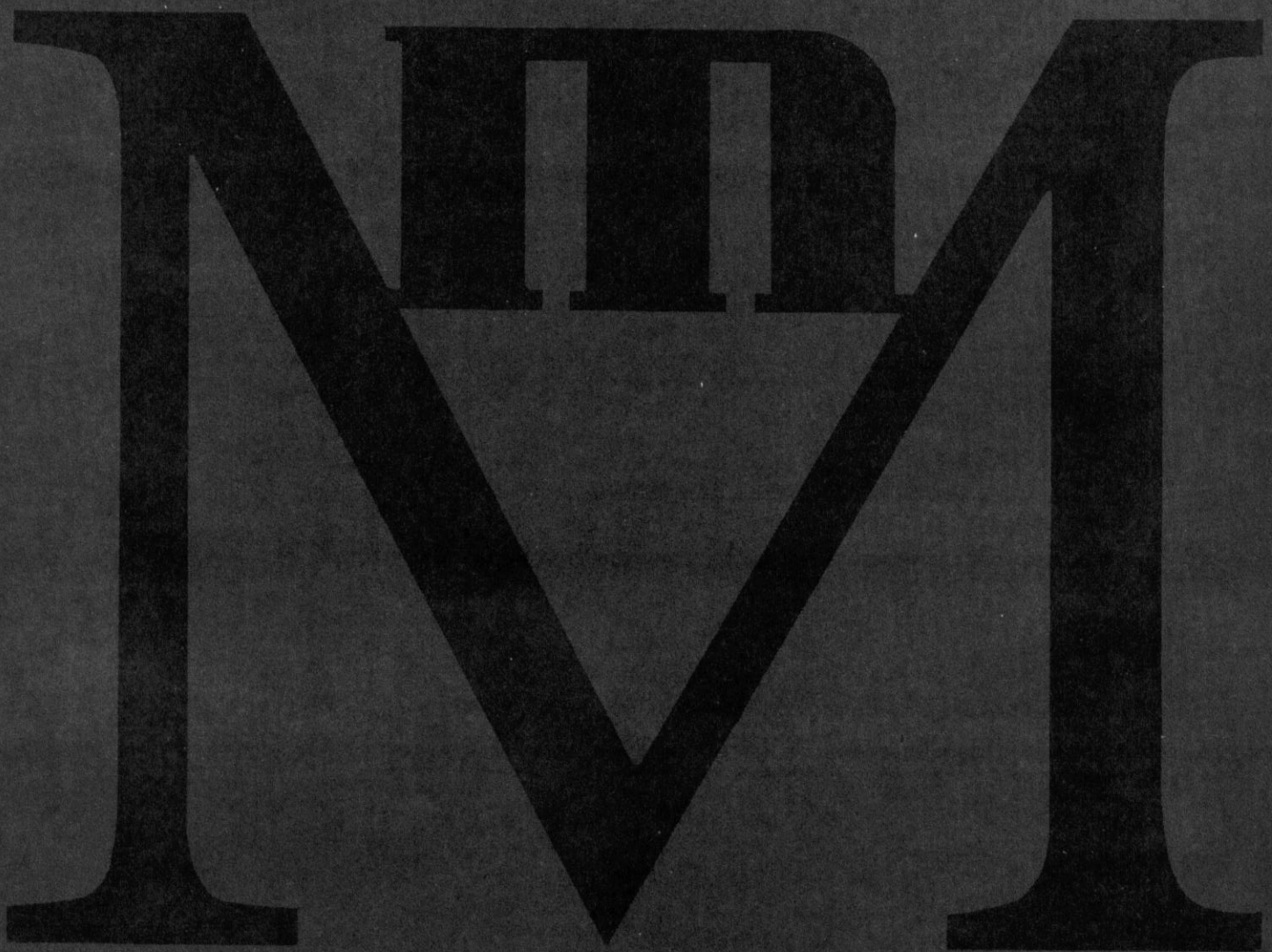
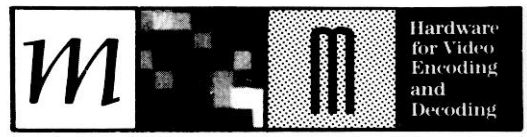
The i750® Video Processor: A Total Multimedia Solution

Kevin Harney,
Mike Keith,
Gary Lavelle,
Lawrence D. Ryan
and Daniel J. Stark





The term *multimedia* means different things to different people. For some, it is the simplest combination of text and graphics on a personal computer. For others, it is the combination of text, graphics, and audio. Still others think of multimedia as including video still images. With the introduction of digital motion video, the definition of multimedia computing becomes even more clouded and confusing.

For the purposes of this article, we offer the following working definition of multimedia computing: Multimedia combines motion and still video, special effects, synthetic video, fast graphics and text with the interactive capabilities of a personal desktop computer.

The i750® video processor was developed to deliver all these multimedia elements cost-effectively to the end user. This article will provide a video subsystem overview, a detailed description of the video processor itself, a sidebar on various aspects of its custom IC design and development, and a discussion of programming the video processor.

First, A Little History . . .

The dominant user interface for personal computers during the mid-to-late 1970s was a keyboard, a monochrome text display, and the command line. A number of these early machines had graphics and simple audio capabilities. However, it was not until 1984 with the introduction of the Macintosh™ with bitmapped monochrome graphics as a fundamental feature, that user interfaces began to shift toward mouse-driven windows and icons.

It has been only in the last year or so that application developers have taken advantage of the opportunity to incorporate audio into mainstream applications, such as computer-driven desktop presentations.

It is interesting to note that the basic graphical user interface has been only modestly improved upon in the seven years since its introduction on the Macintosh. We now have color graphics displays but usually with a disappointingly small number of colors to choose from. Even more interesting, considering the advances in VLSI technology during the same time, is the fact that virtually all personal computers still rely on the host microprocessor to create and format the display.

Technology Advances Set The Stage . . .

Consider some technology advances during the seven years since the introduction of the Macintosh:

- VLSI transistor densities increased 16-fold.
- VLSI speeds increased 8-fold.
- The CD-ROM is available as a low-cost, high-capacity, read-only computer peripheral.
- Dual-port dynamic RAMs were introduced.
- Networks are pervasive.

These five major advances in basic computer technology, coupled with equally major advances in digital video compression algorithms and video processor architectures, transform the simple monochrome text/graphics display subsystem of seven years ago into today's rich and highly dynamic video subsystem. Displays can now feature full-screen, full-motion video, high-resolution video stills, video special effects, 3-D real-time synthetic video, and very fast, true color graphics.

The presence of motion video also pulls audio onto center stage as an integral data element. Because good quality audio requires only 10% or so of video's bandwidth, storage capacity, and processing power, it can be included with motion video with little, if any, additional cost.

The I750 Video Processor

The motivating force behind the design of the i750 video processor architecture was the desire to integrate all processing functions required for multimedia into a single programmable chip set. This involved the design and development of basic video and logic building blocks that would find application over a broad range of algorithms. While a hardwired implementation may at first appear to offer better performance for a particular algorithm, a programmable approach is ultimately superior since it can be reprogrammed to track the evolution of an algorithm or standard, and at the same time, support each of the diverse algorithms required in a full multimedia system.

An example of this flexibility is the ability to process images encoded within the JPEG still image compression standard. While not explicitly tailored for performing the multitude of JPEG-specific calculations needed to reconstruct these images, the processor still performs admirably—decoding a 640×480 JPEG-encoded image requires less than one second.

Programmability has provided a straightforward path to upward compatibility with the i750 A-series video processor (Intel's previous generation i750 is comprised of the 82750PA pixel and 82750DA display processors used in the Intel/IBM ActionMedia™ PC-AT and PS/2-compatible boards).

System Overview

A DVI® display subsystem is shown in Figure 1. Its engine is the 82750PB pixel processor (PB) [4] and the 82750DB display processor (DB) [2].

Separate from the "host" processor, PB is responsible for most of the classical data processing and control functions in the subsystem. The pixel processor compresses and decompresses image data from memory, generates fast graphics and special effects, and acts as the master arbitrator for subsystem memory and interdevice transactions.

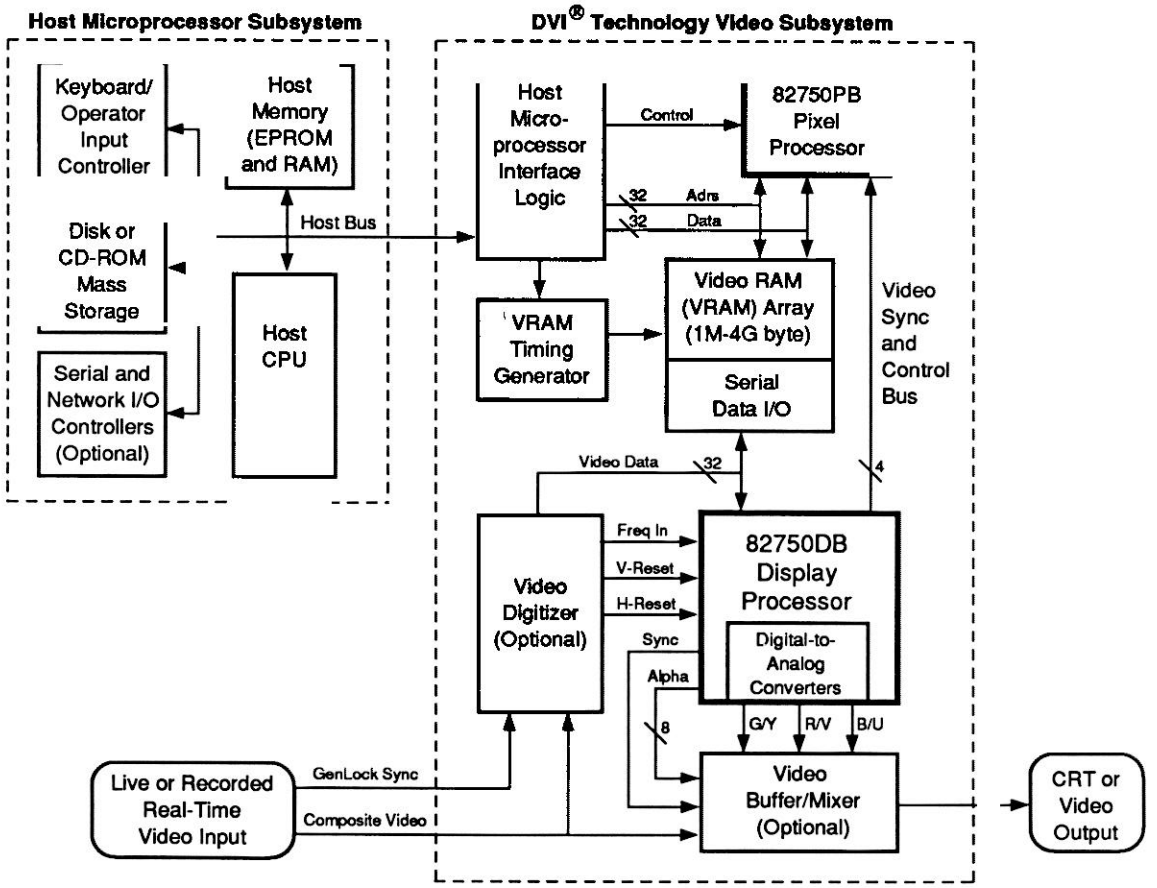


FIGURE 1.
DVI® Technology Display Subsystem

The display processor performs real-time display functions including data-format transformations, color translation, pixel-value interpolation, and provides outputs which support image capture and video synchronization. The DB generates all the timing signals required to drive display devices including digital and analog RGB or YUV pixel outputs and an 8-bit digital word of alpha data. An alpha channel is provided to obtain a fractional mix of DB outputs with another video source to achieve video effects such as titling and graphic overlays.

DVI technology performs its operations in YUV color space. Each YUV component is represented as a series of 8-bit samples. In this color system the luminance, or brightness information (as given by the Y information), is separated from the chrominance, or color information (as given by the U and V components).

Working in this domain allows various psychovisual effects to be exploited, most notably, color information need not be stored at the same resolution as the luminance data [5]. For this reason, color information is stored at one-half or one-quarter the luminance density (subsampled) in each dimension, without significant loss of image quality. Since luma and chroma information are stored at different resolutions, they are placed in different bitmaps.

Generating Displays From Bitmaps

During display blanking intervals, DB reads in programming information and two lines of chrominance information. During the programmed active portion of the display, DB reads in luminance information and, at the same time, interpolates the chroma information up to the resolution of the luminance data.

At the earliest possible point in the blanking interval preceding an active display line, DB requests PB to load a line of the luma bitmap into the serial shift register of the VRAM. At the start of an active display, DB will access the luma information at the programmed transfer rate.

Since each data word is 32-bits wide, each access yields up to four 8-bit luma samples allowing the luma data rate to be one-fourth of the active display rate. The display rate is determined by two factors: the operating frequency of DB; and the ratio of the horizontal resolution of the source bitmap to the display resolution (the pixel rate).

It is possible to overlay DB output with the output of another display device, such as a VGA graphics card (or NTSC video) to combine screens of DVI motion video and an external source. This is made possible by "gen-locking" DB to the external source. Since all of DB's video timing and display parameters are programmable, almost any video rate may be gen-locked.

Selection of the source image stream to be displayed or overlaid is accomplished by "tagging" bits in either the luminance or chrominance data streams stored in video memory. Since the luminance channel is of a higher bandwidth, it can be "channel switched" on a pixel-by-pixel basis. Because the chroma information is subsampled, channel switching is restricted to the subsampled resolution described by either a 2 × 2 or 4 × 4 block of pixels.

The PB offers hardware specifically designed to tag any portion of the input stream of pixels to create windows and overlays from arbitrarily selected sources. The ability to efficiently tag pixel data allows the application to optimize system resources and to produce a high-resolution display that includes text overlays and windows from multiple sources.

The 82750PB Pixel Processor

The 82750PB was designed as a cost-effective, real-time video and graphics processor for multimedia applications. Its unique architecture, with fully static, 25 MHz, single-cycle instruction execution, supports real-time full-screen encoding and decoding of digital motion video at 30 frames/second. The PB can simultaneously perform a wide range of video effects, including scaling motion video into windows, "warping" (texture mapping) video onto surfaces, and many screen transition effects.

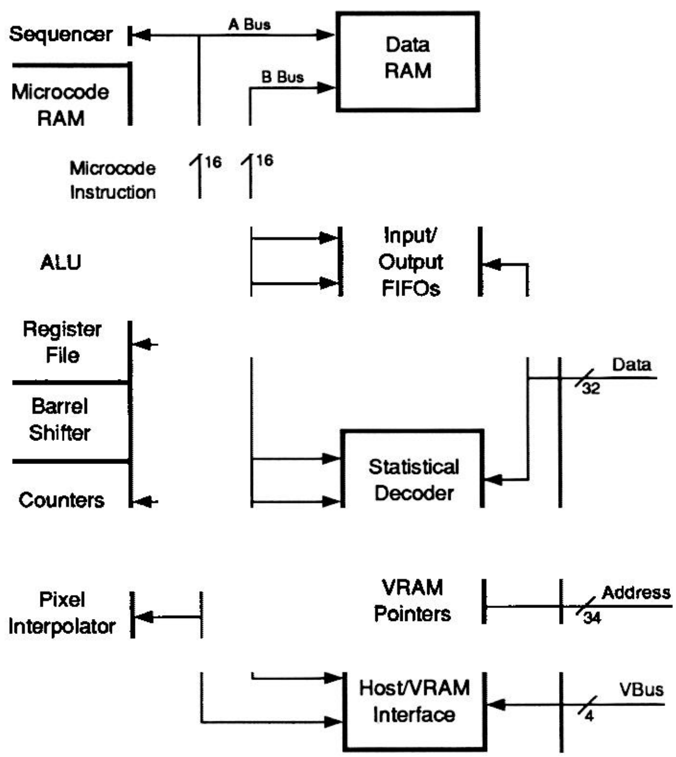
Architectural Overview

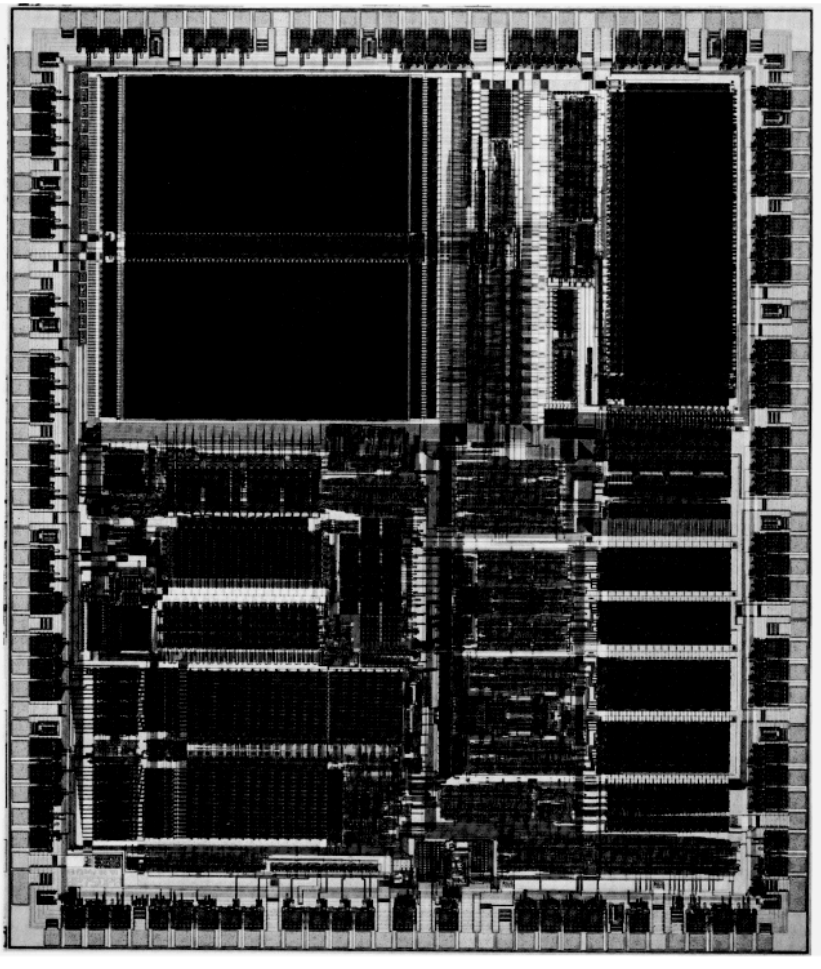
The PB architecture was designed to be simple, efficient, and highly parallel. Figure 2 is a simplified diagram of the 82750PB.

A 32-bit, linear, byte-addressable, external address bus and 32-bit data bus are used to communicate with external memory and external devices. Internal datapaths are either 16- or 32-bits wide and designed to allow the simultaneous processing of two or more 8-bit pixels. Functional blocks are connected to one another using one or more of these buses and the PB can transfer used to perform most arithmetic, logical, and control operations. The ALU supports 32 types of operations specific to processing motion video, such as operations on dual "side-by-side" 8-bit pixel magnitudes.

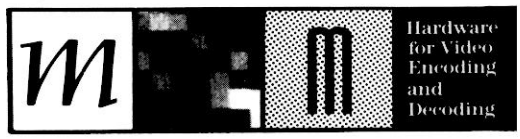
The ALU and barrel shifter (as well as most other functional blocks) communicate with a multiport 16-word × 16-bit register file and a 512-word × 16-bit data RAM. The RAM can be accessed from two 16-bit buses via four independent pointers. These "C-style" pointers can be autoincremented and decremented, as well as partici-

FIGURE 2.





The 82750PB Pixel Processor
This device compresses and decompresses video data and builds screen images pixel-by-pixel. The processor must perform its tasks in less time than it takes each dot to appear on the display. Its standard speed is 25 MHz—higher speeds will be offered. It contains over 310,000 transistors in an area measuring 309 × 261 mils.



pate in general-purpose ALU operations.

The PB also contains two specialized, semi-independent structures that are optimized for image-processing tasks.

First, the pixel interpolator calculates fractional spatial movement of pixels, scales groups of pixels, and is used for various filtering operations.

Second, the statistical decoder decompresses video data in real time. It is a specialized input FIFO that reads variable-length serial bit sequences from memory and decodes them into fixed-length 16-bit values.

Because of the voluminous quantities of memory data that must continually pass through the pixel processor, four independent FIFO channels allow PB to efficiently read, process, and write data. Each FIFO packs and unpacks data words and bytes, buffers data, and initiates transfers to and from the memory array.

Two FIFO channels are dedicated to supply continuous streams of input from memory, containing compressed and uncompressed images as well as the microcode routines required to operate on them.

Two output FIFO channels are used to store intermediate pixel values required during the construction of complete displayable images.

Communication with the "host" processor and other devices on the DVI system bus, as well as PD/DB requests for memory, are arbitrated by a state-machine within PB. This logic is responsible for generating the external timing required to load or read PB internal registers, generate and acknowledge interrupts, and to provide a rotating priority scheme to control the four FIFOs and statistical decoder requests for memory.

Instruction Cache and Execution

Instructions are stored and executed from a pipelined 512-word × 48-bit static cache and can be executed in a single clock cycle.

A microprogrammable cache control algorithm is used to guarantee maximum cache "hit" efficiency and can be tailored for different types of system- and device-level operations.

By coordinating the operation of the i750 assembler with dual-word cache accesses and appropriate PB instruction control fields, zero-delay branches can be performed without conscious programmer intervention. The ability to perform any logical operation and then to effectively perform a simultaneous two-way branch based on the results of that operation yields a significant performance increase over typical microprogrammable processors.

The 48-bit PB instruction word is divided into a total of 11 different control fields designed to allow highly parallel operations.

For example, ALU operations on pixels fetched from memory can be performed simultaneously with iteration counter updates while the pixel interpolator is operating on pixels decoded by the statistical decoder—and then branch—all within a single clock cycle.

Pixel Interpolator

As long as the only objective of video data decompression is the faithful reproduction of the source image, decompression itself is relatively straightforward. If any additional transformations are required (the image is to be enlarged or reduced, scrolled, rotated, or intentionally distorted) a simple correspondence no longer exists between the original stored pixels and those to be displayed.

Straightforward attempts to adjust the image often produce undesirable artifacts known generally as "aliasing effects."

There exist a number of "anti-aliasing" algorithms that eliminate these artifacts at the expense of processing cycles and, as such, are not appropriate for use in real-time systems.

As an example, when the size of an image is reduced, the space between each stored pixel will exceed the space between those to be displayed. This means that each displayed pixel falls somewhere between two rows and two columns of pixels in the original image. In principle, the color and intensity of each displayed pixel can be derived by averaging the values of the four stored pixels that surround it, weighted according to how far the displayed pixel is from the rows and columns to the top, bottom, left, and right.

If TL, TR, BL, and BR define the four corner magnitudes of the pixels of the surrounding block, and h and v define the horizontal and vertical displacement fractions to move a pixel within the block, the weighted average of each color and intensity component is defined by the following equation:

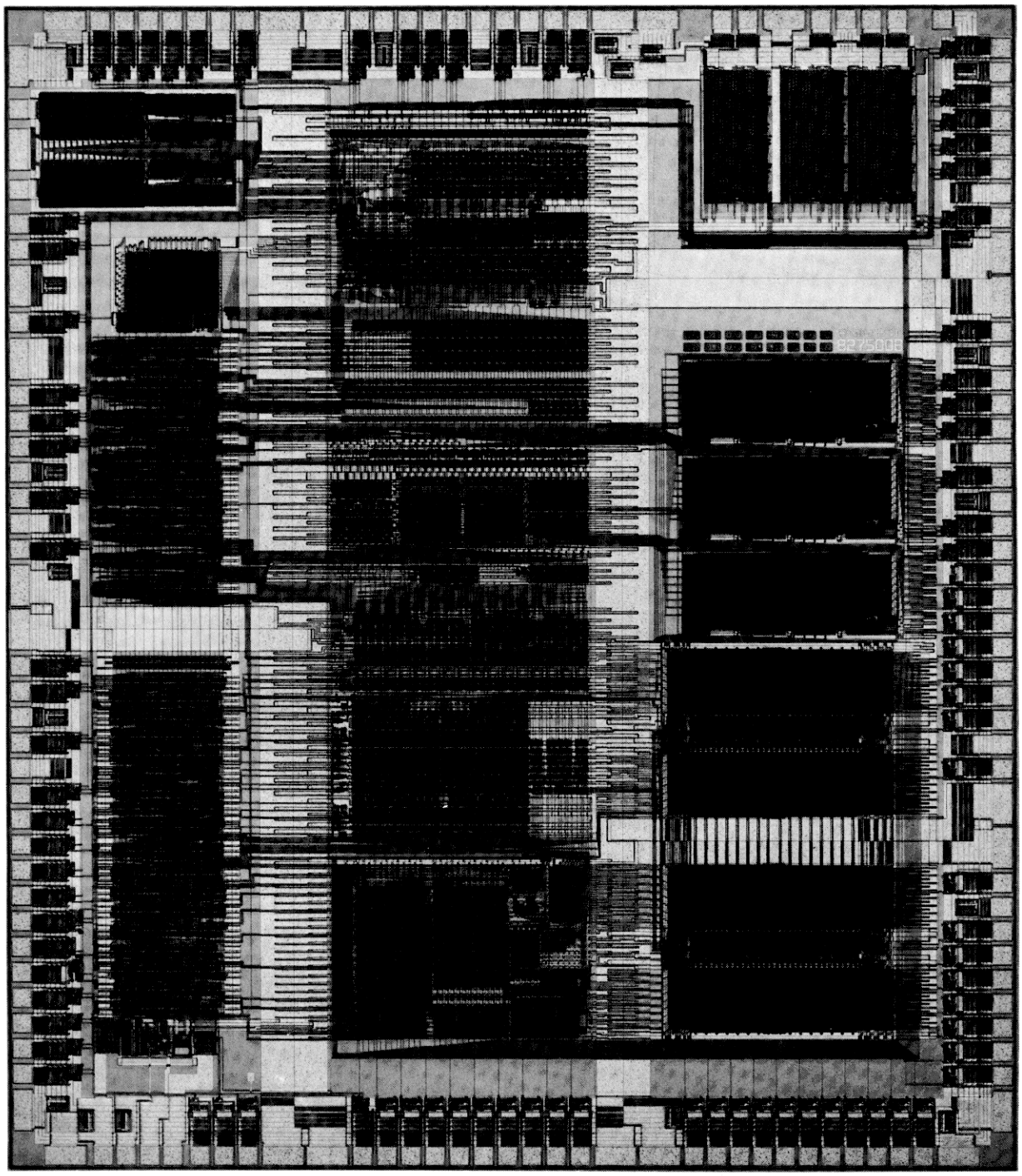
$$W = TL*(1-h)(1-v) + TR*h(1-v) + BL*(1-h)v + BR*hv$$

The horizontal and vertical weights are specified in multiples of 1/16th of the space between individual pixels.

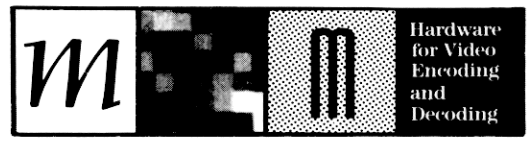
The pixel interpolator is designed to operate in several different modes. Most common is a pipelined sequential mode that allows interpolated pixels to be generated at a maximum rate of one pixel every other clock cycle. Additional modes optimized for interpolating over random pixel "quads," different horizontal and vertical weights, and selectable output "phase" have been included.

Statistical Decoder

The statistical decoder is a specialized input channel that reads encoded bit streams containing variable-length symbols and decodes each symbol into fixed-length values.



The 82750DB Display Processor
This device retrieves expanded image data from memory, converts it to the format needed for display, and produces the color, timing, and control signals needed to drive various displays. Its standard speed is 28 MHz—higher speeds will be offered. It contains over 200,000 transistors in an area measuring 321 × 276 mils.



In image compression, certain values occur more frequently than others. One means of compressing such data is to use fewer bits to encode the more frequently occurring values and more bits to encode less frequently occurring ones. This type of encoding scheme is called statistical or Huffman encoding because it depends on the frequency of occurrence of any particular value to be encoded.

The statistical decoder receives long series of variable-length symbols from memory as a sequence of 32-bit data words. Each symbol encodes both its length and value and ignores the natural boundaries between memory words or bytes. The decoder must concatenate these variable-length sequences, examine the initial bit patterns, extract suitable substrings, expand the substrings, and then realign the remaining data and repeat the process for the next symbol.

Each of the above operations is performed automatically for a variety of codes (code books) and coding conventions. The PB's decoding hardware includes a RAM that allows the encoding and decoding algorithms to optimize themselves for different populations of values to be encoded/decoded.

To use the decoder, a starting address and code book are loaded. Thereafter, programs may simply read an indefinite series of expanded symbols in parallel with other operations.

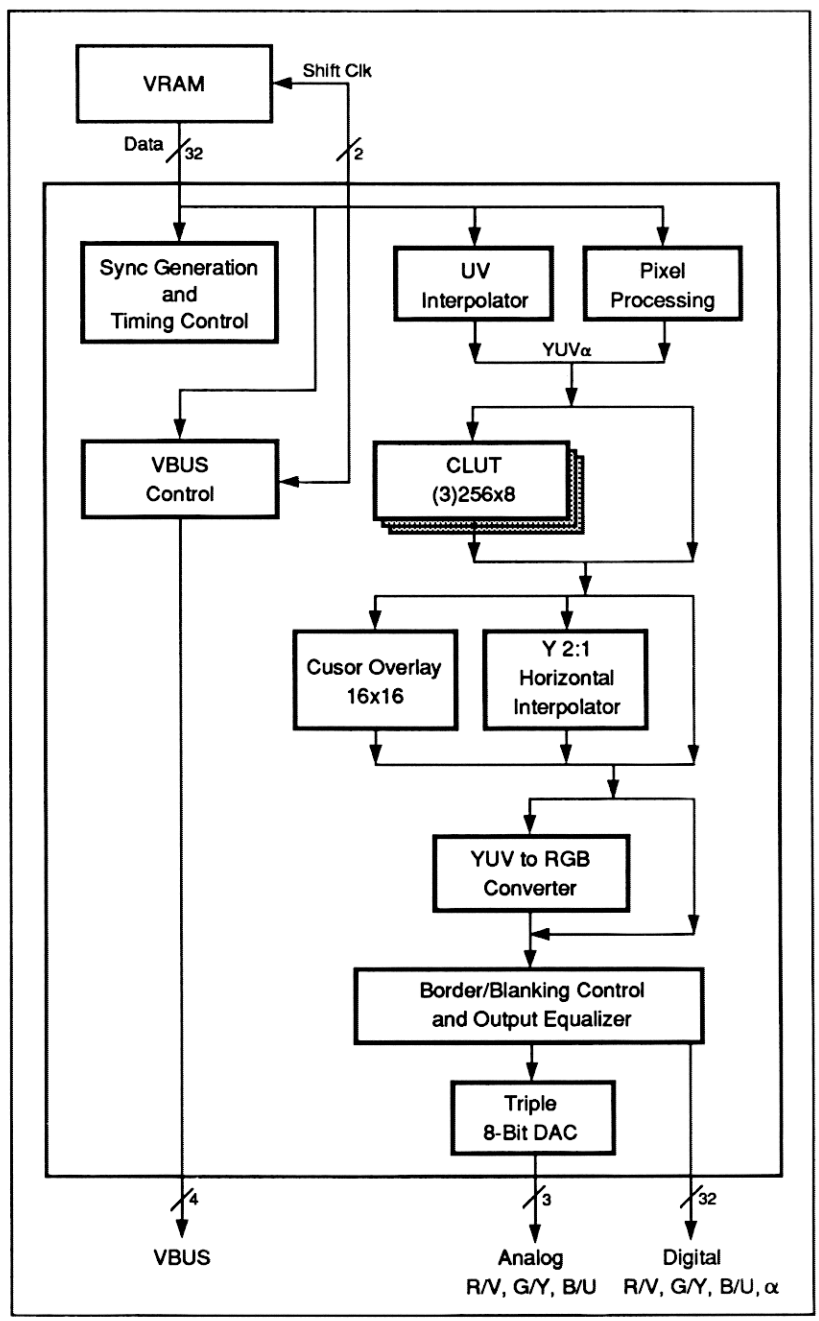
Input And Output FIFOs

The PB contains two input and two output FIFO channels that are used to read and write pixel and program data to external memory. Each FIFO has its own 32-bit address pointer that can be programmed to sequentially increment or decrement through either 8- or 16-bit data types. Each FIFO double buffers 32-bit data words to allow more efficient program execution and to allow hardware memory controllers to take advantage of the sequential nature of FIFO memory accesses.

The 82750DB Display Processor

The 82750DB incorporates all of the digital and analog processing elements necessary to form the basis of a low-cost display subsystem. By programming internal control registers, video timing can be modified to accommodate a wide variety of scanning frequencies and display characteristics. A large selection of bits-per-pixel, pixels-per-line, and pixel heights are available that allow designers a wide latitude in selecting display resolution,

FIGURE 3.
82750DB Display Processor



frame rates and memory requirements.

Architectural Overview

The 82750DB uses a single phase, 1X clock system and a heavily pipelined architecture. The 82750DB incorporates the following major functional units:

- Pixel data path
- Chrominance interpolator
- YUV-to-RGB color space conversion
- VBUS control
- Pixel equalization
- Triple 8-bit DACs

Pixel Data Path

The main purpose of the pixel data path is to read coded pixel bitmaps out of memory, reconstruct bitmaps into Red/Green/Blue pixels, and output them to a video mixer or directly to the display device.

Depending on memory size, timing requirements, and the desired final pixel resolution, application designers will choose from a number of different bits-per-pixel formats. Coded pixel bitmaps are written into a video memory shift register by PB, and read out as 32-bit words by DB during the active display time.

The pixel data path can take different sizes and resolutions of pixel bitmaps and zoom them under program control to whatever final image size is desired on the display screen. The actual positioning of these video and graphics display windows is also fully programmable, down to individual pixel resolutions.

When the display device beam is retracing, DB reads program register data out of memory. Since retrace occurs on a line-by-line basis, DB can be reconfigured (programmed) on virtually every horizontal scan line. This allows multiple "strips" of different video or graphics sources, different window sizes, or different resolutions to be displayed on different portions of the screen.

The DB incorporates a reconfigurable color lookup table (CLUT) that normally is organized as either a single 24-bit or a triple 8-bit 256-entry color map. Because each map can be loaded for every scan line, the programmer may select and display 256 colors from a palette of over 16 million on a line-by-line basis.

These modes allow DB to display color graphics or to perform nonlinear pixel transformations, such as gamma correction. [5]

The CLUT may also be split into two 128-entry color maps that, when used in conjunction with DB's overlay capability, allow completely independent video and graphics pixel transformations.

Another unique functional block in the pixel data path is an alpha channel processor. With this feature, a weighted mix of DB outputs can be combined with other video or graphics sources. This allows not only multiple sources to be displayed in different portions of the display device, but also the ability to perform high-end video processing effects, such as edge feathering.

Two other special-purpose blocks in the pixel data path are the hardware cursor and the horizontal luminance interpolator. The hardware cursor gives application writers the ability to overlay the cursor without editing bitmaps whenever the mouse position is updated. The luminance interpolator generates an averaged Y value that is inserted between each pair of source Y values, softening the edges within reconstructed images. The luminance interpolator can be turned off for "graphics" pixels.

Chrominance Interpolator

The task of the chrominance interpolator is to expand the subsampled color information up to the corresponding luminance resolution.

During programmed blanking time, subsampled chrominance data is read from bitmaps into color information line storage RAMs within DB. The interpolator determines when new chrominance information is needed based on the current display position of the raster scan.

During active display time when subsampled color video is being shown, DB performs a bilinear interpolation to expand color information to the resolution currently being displayed. To accomplish this, the chrominance interpolator expands compressed color data in both vertical and horizontal directions by a ratio of 2:1 or 4:1 in each dimension.

After interpolation, DB can mix chrominance and luminance on a pixel-by-pixel basis. This ability to mix video and graphics on the same screen at any position, with any size, and with any aspect ratio, is one of the distinctive features of the i750 architecture.

YUV-To-RGB Conversion

The YUV-to-RGB color space conversion matrix is compatible with the CCIR 601 standard. The conversion is done fairly late in the processing cycle, after things such as color lookup table operations and chrominance interpolation have been completed.

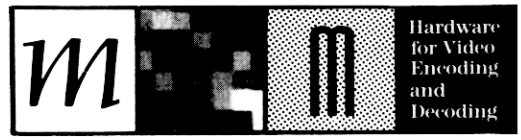
For source data which is coded in RGB format, or for systems that draw only in RGB color space, the programmer can choose to bypass this function without disturbing pipeline timing.

The color space conversion is accomplished by performing a matrix multiplication on the YUV pixels. Since the coefficients are all constants, the multiplications may be done with lookup ROMs and adders.

Saturation circuitry is included to ensure that any arbitrary YUV pixel converts to an RGB value which is within allowed ranges. The accuracy of the conversion process is $\pm 1/2$ LSB.

VBUS Control

The VBUS control unit is responsible for much of DB's internal pixel timing and external system communication tasks, as well as coordinating DB requests for memory



and all communication with PB.

Communication with PB is done via requests over a dedicated asynchronous bus. The VBUS unit is responsible for arbitrating among the various requests that can be sent to PB; these include bitmap display requests, program load information, memory refresh, and frame synchronization information.

Actual communication with VRAM is initiated via DB's serial shift clock. The DB must anticipate when data is available from the VRAM output shift register as a function of PB request processing and memory access time. Programmable delay parameters eliminate the need for DB-to-PB or DB-to-VRAM handshaking.

When gen-locked to an external video source, the VBUS handles the communications necessary to capture and frame buffer the received digitized video.

Pixel Equalization

A significant DB feature is its ability to take source image data of any size and resize it for different windows. Designers program how long each active pixel is held at the device outputs while the display beam is tracing. The selection is made in terms of number of frequency periods. The range of programmable values is not only integer, but half-integer as well. This quantizes the pixel size selection choices to a finer granularity resulting in a greater range of horizontal resolution choices.

The pixel equalizer ensures that the output pixels are all of the same duration, and their widths are of the programmed size. As a system integration feature, the on-chip pixel equalizer eliminates the need for external circuitry for pixel duration equalization.

Digital-To-Analog Converter

Incorporating a triple 8-bit D/A converter on DB represents the single largest savings in overall system cost. The triple DAC delivers Red/Green/Blue analog outputs to a display device. The DAC conversion is monotonic and can operate at the maximum chip input clock frequency. The DAC integral non-linearity (INL) is 0.3% and differential non-linearity (DNL) is 1/4 LSB.

The 82750DB provides both analog and 24-bit digital RGB outputs. The digital outputs can be disabled as a power reduction feature in systems requiring only analog outputs from the DACs.

Programming The i750 Video Processor

Like most microcoded processors, the i750 video processor does not have an "instruction set" in the conventional sense. Rather, its instruction set can be said to consist of a single instruction with a multitude of options. The instruction with its options is shown in Figure 4.

FIGURE 4.
i750PB Instruction Syntax

adst = asrc	bdst = bsrc	alu = aval OP bval	<< >> <>	cnt[2]--	jCOND label
A-Bus Transfer	**B-Bus Transfer**	**ALU Operation**	**Shifter Operation**	**Loop Counters**	**Jump Condition**
adst, asrc, bdst, bsrc = 82750PB internal registers; e.g., rN, dramN, *dramN[++][--], inN-c, inN-lo, inN-hi, *inN, outN-c, outN-lo, outN-hi, *outN, stat-c, stat-lo, stat-hi, *stat, cnt[2], pixint		aval = alatch or a bval = blatch or b ALU opcodes = a, b, ~a, ~b, -a, -b, a++, b++, a--, b--, &, |, ^, +, ++, +], -], ~&, &~, - , -+, +<, -<, ~|, |~	<< = 1-bit left >> = 1-bit right <> = byte swap	cnt-- cnt2--	jmp = always COND = c nc, z nz, n nn, o no, rs nrs, ls nls, cz ncz, cz2 ncz2

Instruction Field Definitions

Each of the various instruction fields will be described in preparation for examining actual coding examples of motion video decompression, memory-to-memory block transfers, and other common routines.

Bus transfer operations are of the form:

adst = asrc, bdst = bsrc

These two fields specify register transfers using two internal data buses. The first transfer uses the "A" bus and the second uses the "B" bus. Most of the registers on PB are connected to both buses, but some are connected to just one.

The term "register" here has a rather broad meaning, referring to any of the bus-addressable objects of PB. These include the 16 general-purpose and a number of special-purpose registers which provide access to the PB's special hardware units (the pixel interpolator, FIFOs, and statistical decoder, for example). Registers available for bus transfers include:

rN (N = 0 to 15)
(16 general-purpose registers)

dramN (N = 1 to 4)
(data RAM (DRAM) pointer registers)

These registers are used to access on-chip DRAM. First, a DRAM address is loaded into one of the pointer registers. Then, DRAM itself can be written or read by using one of the "DRAM contents" registers, which are denoted "*dramN[++][--]." Doing a bus transfer using these names causes the DRAM entry currently pointed to by the pointer register to be read or written with an optional post-increment or -decrement operation.

inN-c, inN-lo, inN-hi, *inN
outN-c, outN-lo, outN-hi, *outN
These registers are used for input and output to external memory using the PB external memory FIFOs. The "lo" and "hi" registers are loaded with a 32-bit memory address, after which the *in or *out registers can be specified to input or output a value to external memory. The "c" register is a control register which specifies byte or word mode, post-increment or post-decrement, and several other options.

stat-c, stat-hi, stat-lo, stat-ram, *stat
These registers read from external memory using the statistical decoder. The "c," "hi," and "lo" registers are similar to PB's FIFOs; the stat-ram register is used to load a small RAM in the statistical decoder that specifies the variable-length code book. Reading the *stat register causes the statistical decoder to decode the next variable-length bit string and return its index in the code book.

pixint
(the pixel interpolator)

To perform a 2-D bilinear interpolation between four 8-bit pixel values (or values of any kind), two writes are performed to the 16-bit "pixint" register (each write containing two 8-bit values). A few cycles later, the "pixint" register can be read and will return the interpolated result. The pixel interpolator can also be programmed to return its results in pairs, so that all processing can be performed on pairs of pixels for maximum efficiency.

cnt, cnt2
(two loop counters)

These can be decremented and tested in parallel with instruction execution, to produce a zero-overhead looping capability.

alu = aval OP bval
This instruction field specifies an ALU operation. There are only two possibilities for aval (and similarly for bval). The symbol "a" tells the ALU to get its input by "eavesdropping" on the "A" bus and extracting whatever value is being transferred there on this instruction. The word "alatch" tells it to reuse the value in its A input latch. The OP specifies one of the ALU opcodes shown in Figure 4. These include standard arithmetic and logical operations plus some special opcodes such as the dual-add-with-saturate operation: +].

<< >> <>
These operations specify a single-bit left shift, single-bit right shift, or byte swap using a special shifter register. The 82750PB also has a barrel shifter for multibit shifts, but the advantage of the single-bit shifter is that its shifts can be done in parallel.

cnt[2]—
This operation decrements one of the two loop counters. In combination with the conditional jump field, this provides the zero-overhead looping feature of PB.

jmp label jCOND label
These fields specify an unconditional or conditional jump. Various conditions can be used for conditionals, including ALU condition codes (zero, overflow, etc.), shifter conditions, and loop counter conditions.

Microprogramming

A program consists of a series of these instructions. Each instruction executes in a single clock cycle, no matter how many fields are used. In other words, all the available options execute in parallel.

Figure 4 also illustrates several additional features of the PB programming language. Although PB is programmed in microcode—an even lower-level language than assembly code—the programming language has a high-level, C-like syntax. The motivation for this approach is to make complex microcode programs as readable and maintainable as possible. For example, instead of writing:

```
MOV Y,X MOV Z, NUL INA INB
ADD DCNT
```

as might be done in some microcode-level languages, simply write:

```
x = y,    nul = z,    alu = A + B,
cnt--;
```

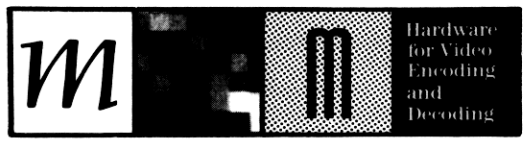
which gives a much clearer picture of what is going on. The PB microcode assembler works in conjunction with a C preprocessor, which provides the usual macros, conditional assembly, and other features found in the preprocessor.

Microprogram Performance

The 82750PB is typically 5-to-10 times faster than a conventional microprocessor at the same clock speed (even faster for operations such as image decompression and manipulation which take advantage of its special hardware assists). There are two reasons for its efficiency: single-cycle instruction execution and the inherent parallelism of the dual-bus, multifield instruction. Single-cycle execution means that instructions execute faster and parallelism means that fewer instructions are required.

The dual-bus nature of PB permits elegant constructs such as the following instruction to swap the values in two registers: x = y, y = x;

Or, consider the "butterfly," a key component of Fast Fourier



Transform and Fast Cosine Transform algorithms; two variables "x" and "y" are replaced with the values x + y and x − y, respectively. This can be accomplished in three instructions on PB, which is fewer than required on most conventional CISC processors:

```
nul = x, nul = y, alu = A + B;
x = alu, alu = Alatch − Blatch;
y = alu;
```

Decoding Motion Video

Now consider a more complex example of PB coding, such as decoding digital motion video using vector quantization and motion compensation.

One of the algorithms used for creating full-screen motion video from compressed data files is called Production Level Video (PLV). In this algorithm, most images are encoded as difference images; that is, the encoded data represents the difference between the current image and the previous image in the video sequence. To provide further compression, the image is subdivided into rectangular regions.

The compressed data contains information for each region; this tells the decoder where, in the previous image, a region can be found to serve as the basis for constructing the region in the image being created. This location will usually be spatially near the location of the region in the current image, but it need not be exactly coincident.

Furthermore, the best compression is achieved when the location of the previous-image region is specified in fractional pixel coordinates. This means that the decoder must interpolate between pixel values in the previous image to derive the starting region—a prime application for the PB pixel interpolator. This process of deriving an array of initial values from a spatially shifted region in the previous image is referred to as motion compensation.

After deriving the motion-compensated array, the decoder must somehow construct an array of (signed) difference pixels, which are then added to the motion-compensated array to produce the final reconstructed region. In PLV, the difference array for a given region can be constructed using one of several algorithms. The algorithm illustrated here is known as "2 × 1 vector quantization."

In 2 × 1 vector quantization, each horizontally adjacent pair of pixels in the difference array is quantized to one of a finite set of number pairs (vectors). Each vector in the set is identified by a unique integer which is then statistically encoded. (Of course, the vectors in the vector table are sorted by expected frequency of occurrence.) The decoder simply decodes a Huffman code and does a lookup into the current table of vectors. The pair of numbers retrieved from the table is then written to the difference array.

Figure 5 illustrates the preceding process diagrammatically. A region in the previous image (located at fractional pixel coordinates) is added to a difference array produced by statistical decoding and vector table lookup to produce a region of the current image being constructed.

Motion Video Decode Example

Listing 1 presents the actual PB code to perform the inner loop of this algorithm. It uses a line buffer in on-chip DRAM. The reason for this is that two lines must be read from the previous image in order to perform pixel interpolation. The straightforward implementation requires reading each line from external memory twice (once as the

i750 Video Processor

Implementation

The major design goal of the i750 video processor was to deliver multimedia functionality and performance at a cost that would make broad acceptance possible in desktop machines. Following are some of the key parameters that were considered in making performance/cost trade-offs.

Performance

The key performance requirement was to compress, decode, and display motion video with special effects in popular high-resolution formats such as NTSC, PAL, VGA, and SVGA. To meet the requirements of these and future display formats, such as XGA, innovative circuit design techniques were employed that fully utilize Intel's unique design tools and advanced CHMOS*-IV technology and fabrication capabilities.

System Cost

In today's desktop machines, there is a high cost associated with component count and the resultant use of board area. Reducing system component count is as important as reducing chip cost. A conscious effort was made to integrate system-level and peripheral functions on the chip to reduce overall system cost. Some of these functions include a YUV-RGB converter, triple video-rate digital-to-analog converters, an on-chip bilinear interpolator, and a statistical decoder.

Component cost is directly related to die size and packaging options. Die size was significantly reduced by using Intel's custom layout tools and CHMOS-IV one-micron "design rules." A plastic, 132-pin leaded quad-flat package was chosen to provide the lowest cost surface-mount capability.

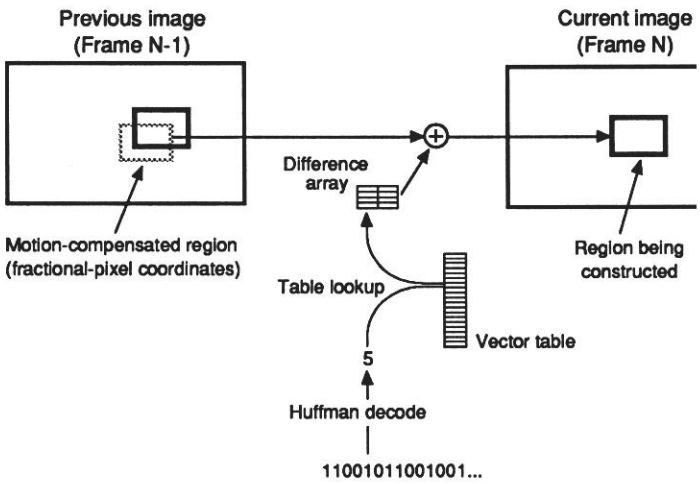


FIGURE 5.
Interframe Motion-Compensated Vector Quantization

```
/* Define some names for PB hardware objects */
#define input in1          /* inputFIFO for reading pixels */
#define output out1        /* output FIFO for writing pixels */
#define readouf dram1      /* Points to previous-line buffer in DRAM */
#define writebuf dram2     /* Points to same buffer; for reading */
#define vector dram3       /* Points to vector table at DRAM address 0 */
/* Here is the actual code */
LOOP:
pixint = *readbuf++,       *output = alu, cnt -;
pixint = *input,                          alu = A;
vector = *stat,            *writebuf++ = alu;
nul = pixint,              nul = *vector,    alu = A +] B,   jcp loop;
```

LISTING 1.
Interframe Motion-Compensated Vector Quantization Decoding

bottom line of a pair of lines and once as the top line of the next pair of lines). By buffering each input line in on-chip DRAM, extra memory reads are saved and execution time is reduced.

The first line of code in Listing 1

pixint = *readbuf++, *output = alu, cnt−−;

loads a 16-bit value, containing two pixels, from the line buffer into the pixel interpolator. Simultaneously, it outputs the final result (also two pixels) from the last time through the loop to memory. The loop counter, cnt, is also decremented.

The second instruction

pixint = *input, alu = A;

loads two pixels from the next line of the source region into the pixel interpolator, and also passes it through the ALU. This is done so it can be saved in the line buffer on the next instruction.

The third instruction

vector = *stat, *writebuf++ = alu;

reads the statistical decoder into a DRAM pointer. This puts the decoded code book index for the vector into the DRAM pointer. The vector table has been located in DRAM starting at address 0, so the required table lookup can be performed on the next instruction by simply reading the value of *vector. Also on this instruction, the pixels read on the previous instruction are put into the line buffer.

The final instruction

nul = pixint, nul = *vector, alu = A +] B, jcp loop

reads the output of the pixel interpolator (which produces 2 values in the motion-compensated array) and adds this to the difference-image vector (which is obtained by the table lookup through *vector). The ALU +] opcode is used, which independently adds the upper and lower bytes in the two 16-bit words, and also restricts each result to the limits 0 and 255. Finally, this instruction performs a zero-overhead conditional branch based on the loop counter.

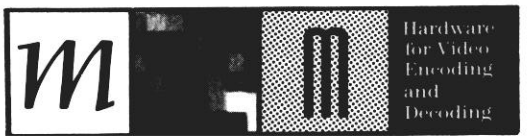
Decode Performance

Since this loop processes two pixels every four instructions, at 25 MHz it can process about 200,000 pixels in 1/60 of a second. Decompression of an entire image includes other processing steps; total execution speed is typically about 70,000 pixels per 1/60 of a second, which is sufficient for good quality full-screen video. Note that for standard 30 frame/second video, this means that only about one-half the PB cycles are used for reconstructing each image. The remaining cycles can be used to apply video effects, graphics overlays, and other operations which are useful in an interactive environment.

Other Applications and Performance

In addition to motion video compression and decompression, PB can be programmed to do image manipulation, video special effects, 2-D and 3-D graphics, text, and digital audio algorithms [1].

For example, a piece of an image can be copied in external memory (such as a "raster-op" or "bit-blit" operation) by the following inner loop:




```
LOOP:
tmp = *in1, cnt--
*out1 = tmp, jcp loop
```

This requires two instructions rather than one because only one memory access per instruction is permitted in PB. For an 8-bit bitmap (or bitmap plane), this provides a "blit" speed of 12.5M pixels/second.

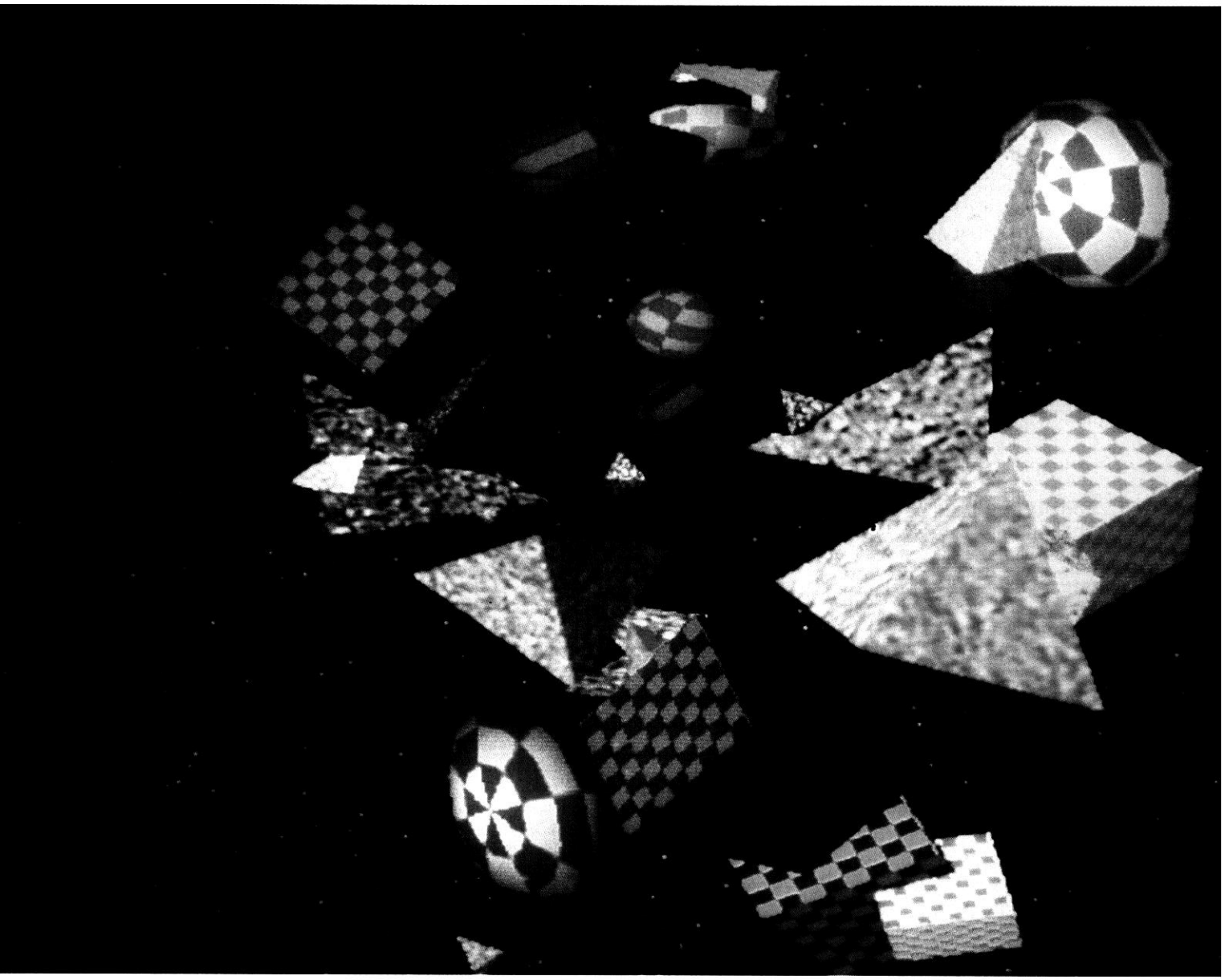
Now, an arbitrary transformation of the pixel values can be achieved at no cost, using a lookup table in DRAM, as follows:

```
LOOP:
dram1 = *in1, cnt--
*out1 = *dram1, jcp loop
```

This can be used for example, for real-time adjustment of image brightness, contrast, or tint, by applying the appropriate transformation of values for each color component.

Standard 2-D and 3-D graphics primitives have also been implemented. Figure 6 shows an example of 3-D texture-mapped solids drawn with PB, including Z-buffer-

FIGURE 6.
An Example of Three-Dimensional Texture-Mapped Solids Drawn with PB



ing for hidden-surface removal. Drawing speed for this picture is approximately 1 million pixels/second.

Although PB does not have a hardware multiplier, algorithms involving multiplication can often be implemented efficiently, using combinations of the two shifters available (single-bit parallel shifter and multibit barrel shifter). For example, a recent implementation of the CCITT/ISO JPEG image-compression algorithm based on the Discrete Cosine Transform (DCT) was found to be very multiply intensive.

Nevertheless, PB is able to decompress a 640 × 480 JPEG encoded image in less than 1 second—about 10 times faster than typical 25MHz microprocessors (even those with a hardware multiply capability). This illustrates the power and flexibility of programming PB at the microcode level.

The PB can also be used for digital audio decompression. Algorithms such as CD-ROM/XA, which are based on time-domain ADPCM techniques, can be decoded in 5% to 10% of PB's available cycles. This permits the design of very low-cost multimedia systems in which a single PB does both the video and audio processing.

Conclusion, Prediction, and Challenge

This article has described a low-cost VLSI solution for delivering full multimedia to personal computers and workstations. The i750 video processor's substantial increase in multimedia performance results from a new video-rate DSP architecture that has been well matched to key video-oriented processing requirements.

Programmability at the microcode level offers hardware OEM's and application developers complete flexibility in the face of uncertain and evolving end-user requirements without compromising system cost or performance.

The new elements of multimedia—motion and still video, special effects, synthetic video, fast graphics—will result in a new paradigm for personal computers of greater impact than the 1980s shift from the "command-line" to the graphical user interface.

Along with the advantages and benefits of multimedia comes the challenge to system and application developers to create new, friendlier and more effective user interfaces that fully leverage all the multimedia elements now possible with the i750 video processor.

Acknowledgments

The authors would like to acknowledge the contributions made to the design and development of the i750 video processor and this article: Tuan Bui, Ken Caviasca, Alfred Frim, Don Garrett, Judi Goldstein, Ei-Ichi Kowashi, Joseph Krauskopf, Kai Lee, Michael Patti, Hy Rao, Mark Ross, David Sprague, Sanjay Vinekar and Gini Volini. ◻

References

1. Foley and Van Dam, A. *Fundamentals of Interactive Computer Graphics.* Addison-Wesley, Reading, Mass.
2. Lavelle, G.J. and Harney, K., et al. The 82750DB Display Processor. *Intel Tech. J.* (Jan. 1991).
3. Luther, A.C. *Digital Video in the PC Environment.* McGraw-Hill, Second Ed, 1990.
4. Patti, M.F. and Sprague, D.L. The 82750PB Pixel Processor. *Intel Tech. J.* (Jan. 1991).
5. Pritchard, D. Worldwide color television standards—Similarities and differences. *J. Soc. of Motion Picture and Television Eng.* (Feb. 1980).

CR Categories and Subject Descriptors: B.4.2 [**Input/Output and Data Communications**]: Input/Output Devices—*Image display*; C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures—*Parallel processors*; C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Microprocessor/microcomputer applications*; I.4.2 [**Image Processing**]: Compression (coding)—*Approximate methods*

General Terms: Design

Additional Key Words and Phrases: DVI, digital video, multimedia

About the Authors:

KEVIN HARNEY is a chip designer at Intel Corporation.
His research interests include image processing.

MIKE KEITH is a software engineer at Intel Corporation.
His research interests include digital video algorithms and applications.

GARY LAVELLE is a chip designer at Intel Corporation. His research interests include computer architectures and display processors.

LAWRENCE D. RYAN is a design manager at Intel Corporation.
His research interests include DSPs, parallel processing and multimedia applications.

DANIEL J. STARK is a chip designer at Intel Corporation.
His research interests include computer architectures.

Authors' Present Address: Intel Corporation, 313 Enterprise Drive, Plainsboro, NJ 08536