A Wavelet Analyzer[1]
An alternative to the FFT-based spectrum analyzer

Mac A. Cody[2]

The ubiquitous (and indispensable) oscilloscope and its cousin, the spectrum analyzer, are the basic tools of the trade for electronic engineers. The oscilloscope is used for viewing signals; its display shows the signal as a function of voltage vs. time. The spectrum analyzer allows you to view the signal as a function of voltage vs. frequency. Some versions of the spectrum analyzer use a frequency-adjustable, narrow-band filter to extract the spectral content of the signal. As the filter's center frequency is swept from lower to higher frequencies, the output-signal energy is traced upon an oscilloscope screen. The resulting plot is the time-averaged spectrum of the input signal. More modern spectrum analyzers use the fast Fourier transform (FFT) to directly calculate the spectrum of the sampled signal.

An alternative to the Fourier transform is the wavelet transform (discussed in my article "The Fast Wavelet Transform," *Dr. Dobb's Journal*, April 1992), which is capable of simultaneous analysis of both time and frequency in signals. In addition, an infinite variety of wavelets enables you to tailor that analysis according to the wavelet chosen. The fast wavelet transform provides an efficient implementation of the wavelet transform for use on digital computers.

This article presents an implementation of a *wavelet analyzer* (a variation of the spectrum analyzer), which uses the fast wavelet transform rather than the FFT to analyze the signal. The fast wavelet transform is implemented as a recursive routine on a digital signal processing (DSP) board based on an AT&T DSP32 or DSP32C DSP IC. The input is a voice-grade audio signal sampled and digitized by a "codec" (a type of analog-to-digital converter) on the DSP board. The wavlet analyzer uses the features and capabilities of a standard VGA controller to display the input signal and wavelet transform coefficients as they are generated in real time.

No Recourse But to Recurse

My previous article provided a complete description of the fast wavelet transform algorithm; here I'll focus on one variation. The tree or pyramid structure of the fast wavelet transform algorithm naturally leads to a recursive implementation; see Figure 1. The input of each level of the tree or pyramid algorithm depends upon the output of the previous approximation level. This is true even for the first level, since the input samples are themselves an approximation of the continuous function which the samples represent. Note, also, that the filter pairs for all levels are identical. Each level of "filtering" is followed by a down-sampling (or decimation) by a factor of 2 prior to sending a sample to the next-lower level.

Therefore, the recursion for each level acts upon the same filter pair with the same set of wavelet filter coefficients. Each level of recursion occurs half as often as each higher level. Recursion terminates under two conditions. The first (and most common) condition is that, at any level, two samples have not been generated, so decimation cannot occur. The decimation operation occurs as a result of sliding the samples through the filters two at a time, generating one output. Before proceeding to the next-lower level, recursion must occur twice, generating two approximation coefficient samples. The other condition for termination of recursion is execution of the lowest (and last) level of approximation and detail filters. When this happens, the approximation and detail-coefficient samples are both output. Depending upon the level of recursion performed, there can be from three to nine signal, detail-coefficient, and approximation-coefficient samples output from each recursion for a six-level fast wavelet transform, see Table 1.

In my previous article, the program listings for the transform contain control statements that handle the ends of data arrays. This is necessary because the data arrays have finite length in both directions. In the case of real signals, however, once sampling of data begins, it can go on endlessly (or as long as the hardware and power supply hold out). As a result, the algorithm must now deal with a nonterminating data sequence, which actually makes the algorithm and code implementation simpler, since new data will always be available.

[1] Originally appeared in Dr. Dobb's Journal, April, 1993. Copyright ©1993, M&T Publishing Inc., Miller Freeman Publishing, Inc.; preprint created by permission. Corrections made by the author.

[2] Mac is the owner of Mac A. Cody Associates, Richardson, TX. He has published Tsunami Plus, a wavelet software library written in C. He can be at contacted by phone at 214.437.5761 and on the Internet at mcody@dfw.net.

Signal sample	Detail level sample 5	4	3	2	1	0	Approximation sample	Samples to plot
0, 1	0							3
2, 3	1	0						4
4, 5	2							3
6, 7	3	1	0					5
8, 9	4							3
10, 11	5	2						4
12, 13	6							3
14, 15	7	3	1	0				6
16, 17	8							3
18, 19	9	4						4
20, 21	10							3
22, 23	11	5	2					5
24, 25	12							3
26, 27	13	6						4
28, 29	14							3
30, 31	15	7	3	1	0			7
32, 33	16							3
34, 35	17	8						4
36, 37	18							3
38, 39	19	9	4					5
40, 41	20							3
42, 43	21	10						4
44, 45	22							3
46, 47	23	11	5	2				6
48, 49	24							3
50, 51	25	12						4
52, 53	26							3
54, 55	27	13	6					5
56, 57	28							3
58, 59	29	14						4
60, 61	30							3
62, 63	31	15	7	3	1	0	0	9

Table 1: The recursive fast wavelet transform generates a quantity of detail samples for every two signal samples it is supplied.

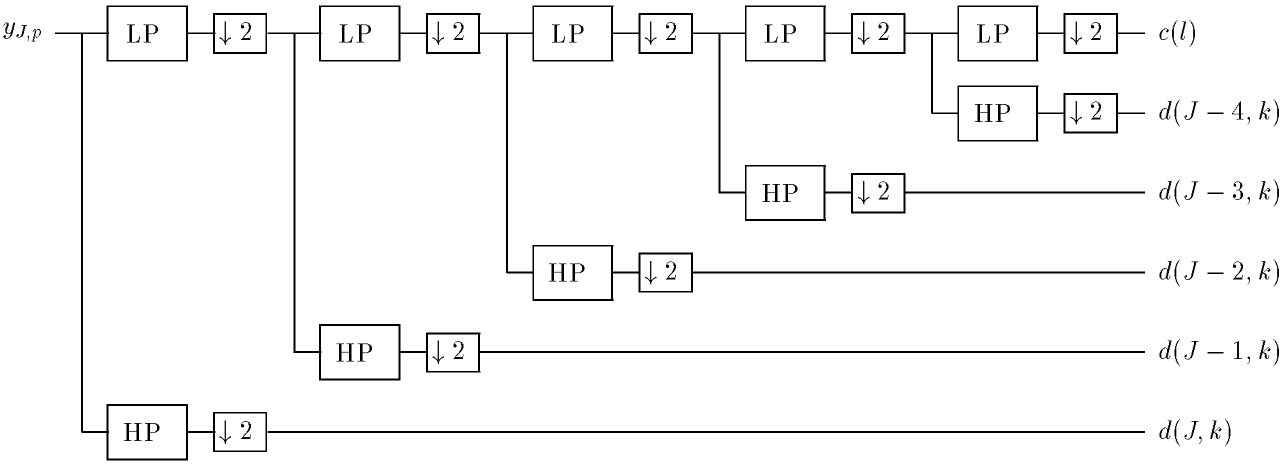

Figure 1: The tree or pyramid algorithm of the forward fast wavelet transform lends itself to a recursive implementation.

DSP Implementation

The recursive fast wavelet transform is implemented on the AT&T DSP32 and DSP32C digital signal processors. I chose the DSP32 and DSP32C were chosen because of their support of floating-point math at high speeds (8 MegaFLOPs and 24 MegaFLOPs, respectively), pointer addressing, and their C-like assembly language. These DSPs are described more fully in Jim Bittman's article "Adding the Power of DSP to Your Application" (*Dr. Dobb's Journal*, May 1991).

The fast wavelet transform algorithm borrows heavily from the field of DSP. As a result, the algorithm can easily be taken to a full DSP implementation. Four source-code modules, written in AT&T DSP32 and DSP32C assembly languages, form the DSP component of the wavelet analyzer. Listing One (page 82) is the recursive transform; Listing Two (page 82) is the pixel plotting routine; Listing Three (page 87) the main loop; Listing Four (page 90) storage allocator for variables and constants; and Listing Five (page 90) the make and header files.

Recursive Fast Wavelet Transform

The routine DECOMP (see Listing One) implements the recursive fast wavelet transform, or signal decomposition. The routine first retrieves pointers to the input- and output-approximation storage arrays. The input-approximation coefficients are supplied by the next higher level of the transform (or recursion). In the case of the first level of recursion, the input consists of the signal samples from the codec. The output-approximation coefficients are those generated by the current level of the transform.

The routine then performs two data convolutions between the input-approximation coefficients and the high-pass and low-pass wavelet filters (the wavelet coefficients). This operation is coded in-line to avoid the execution-time penalty of control loops. A jump vector supplied by the calling function causes DECOMP to execute the necessary number of convolution instructions for the length of the wavelet filters.

The first convolution produces one wavelet-detail coefficient sample. This sample is stored in an array for eventual display on the host system's VGA. The second convolution produces one wavelet-approximation coefficient sample. At the same time, the input-approximation coefficients are shifted by two storage locations, thereby producing the decimation factor of 2 and opening up two free storage locations for new approximation samples from the next higher transform level. What happens to the newly calculated approximation-coefficient sample depends on the current level of recursion of DECOMP and the number of samples previously generated at this level of recursion.

As DECOMP executes each level of recursion, a recursion counter is decremented. When the count reaches 0, DECOMP has reached the desired lowest level of the wavelet transform, level 0. At this point the new approximation coefficient sample is placed into the array with the other detail-coefficient samples for display. This coefficient is the level-0 approximation coefficient for the wavelet transform.

If the recursion count has not reached zero, the approximation-coefficient sample is destined for input into the next lower level of the wavelet transform. If the sample is the first of a pair of samples for the next lower level, then it is stored in the appropriate input-approximation coefficient array, and DECOMP then returns to the next higher level of recursion. If this sample is the second sample of the pair, then it is stored and DECOMP calls itself to proceed down to the next level of the transform (and the recursion). At the same time, the output pointer is realigned to allow two more approximation samples to be placed into the output array during future recursions.

Pixel Plotting Routine

The DRAWIMAG routine plots each sample generated by the codec or DECOMP into one of two sets of image bit maps in the DSP memory; see Listing Two. Each set consists of eight bit maps corresponding to the displays of the input signal, the six detail levels, and the level-0 approximation. Each map is designed to hold eight unit intervals of sample data or wavelet coefficients. A unit interval is the number of samples required to generate a wavelet transform to a desired number of levels. In this application, a six-level wavelet transform has a unit interval of 2^6, or 64, samples at the input.

Each map consists of 50 rows of bytes, which correspond to the 50 rows assigned for each signal trace on the VGA display. The number of bytes per row depends upon the number of samples in eight unit intervals for the input signal or wavelet-coefficient level displayed. For example, the level-2 detail bitmap, in this application, has four samples per unit interval. With eight unit intervals, this bitmap need four bytes per row. Exceptions to this general rule are the level-0 detail and approximation-display bitmaps, which require two, rather than one byte per row. This is necessary because the DSP32 and DSP32C are restricted to 16 and 32-bit DMA transfers through the parallel data register. The additional byte in each row is unused and set to 0.

Plotting a sample value into the bitmap involves translating the sample value into a count (either up or down) from the baseline in the bitmap; see Figure 2. The count determines the length of a line of pixels drawn from the baseline. The byte-column pointer and bit pointer determine in which pixel column the line will be drawn. There is a bit pointer, byte-column pointer, and baseline offset value for each of the eight bit maps of the image.

Depending upon the value of the bit pointer, the drawing operation is either a direct write of the pixel values to the memory or a logical OR with the current contents of the bitmap prior to the write. When the bit pointer selects the most significant bit in the byte, all 50 bytes in the currently selected byte column are written to so that the previously displayed image is overwritten while the new image is written. The write operation proceeds from the upper blank portion of the image to the trace pixels, and then the lower blank portion of the image. If the sample value is full scale in the positive, the upper blank fill operation is skipped; if the sample value is full scale in the negative, the lower blank fill operation is skipped. Trace pixels are not drawn if the signal level is below the scale of the least significant pixel.

When the bit pointer points to a bit other than the most significant, the pixel-drawing routine uses logical ORing of the bit pointer and the bytes in the column to turn on the trace pixels. This process takes longer than the direct write operation described previously, but it is performed only on the bytes where the trace image will reside. The baseline offset determines the starting byte for the line of pixels drawn. While calculating the pixel draw count, the direction of the draw operation from the baseline is established. A draw count of 0 results in no pixels being drawn on that bit column.

Once the column of pixels is drawn, the bit pointer is shifted down one bit and saved for the next call of DRAWIMAG for that bitmap. If the bit pointer is 0 after the shift, then it is set to point to the most-significant bit, and the byte column pointer is incremented and saved. The pointers move through all the bit columns of their respective bit maps until all samples of the eight unit intervals are plotted.

Main DSP Control Loop

ANALYZER is the main routine for the DSP portion of the wavelet-analyzer program; see Listing Three. The routine initializes the DSP registers, memory, and codec prior to entering the main processor loop. Within

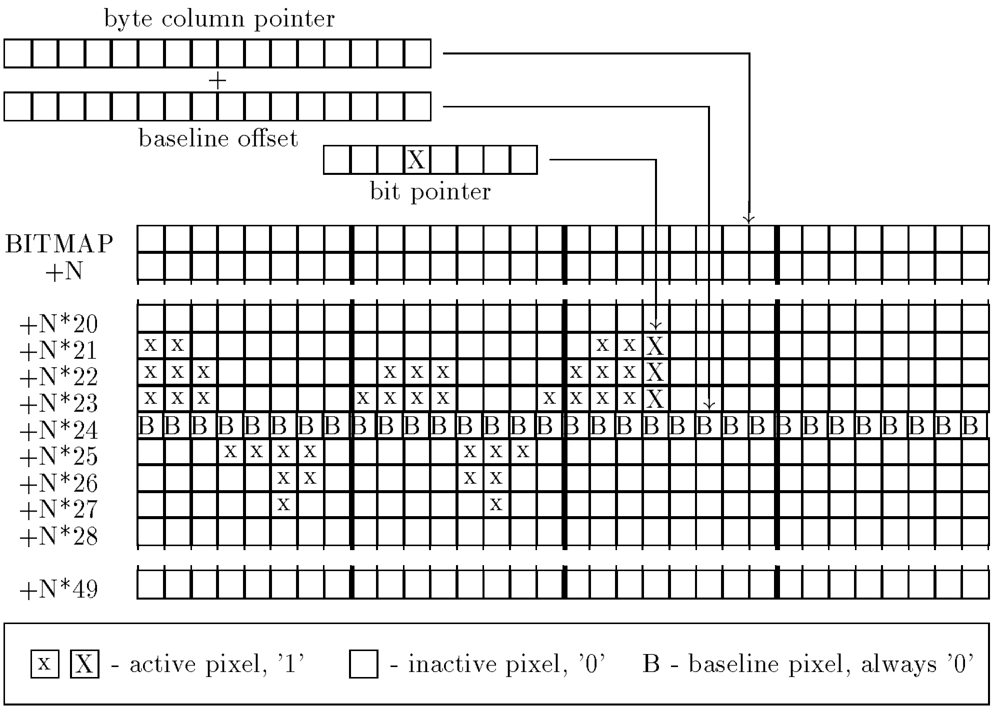

Figure 2: DRAWIMAG plots each signal and wavelet transform coefficient sample into a bitmap within the DSP memory.

the main processor loop, signal samples are taken from the codec and passed to DECOMP for processing to generate the next group of wavelet coefficients. After returning from DECOMP, DRAWIMAG is called several times to plot the group of signal samples and wavelet-coefficient samples into the various image bitmaps for eventual display on the VGA screen. The plotting operation can exceed the time of arrival of the next sample from the codec. To prevent loss of a sample during plotting, the serial-input buffer flag is checked after each plot until the sample arrives. Once the sample arrives, it is acquired and the remainder of the plotting operations are completed.

The ANALYZER main loop processes eight unit intervals (512 data samples) in order to fill the bitmaps with new information to display. Then the signal processor flags the host processor and informs the host program which set of bitmaps (IMAGE_0 or IMAGE_1) holds the latest set of display data. The signal processor then reinitializes the pointers for DRAWIMAG to plot to the other set of bit maps. Before proceeding with processing the next pair of signal samples, the DSP checks that the host processor has acknowledged the flag and waits if it hasn't.

Watchin' the Waves Roll In ...

Once the DSP is running, the host PC essentially acts as an I/O server for the DSP board. Whenever a new block of image data is plotted and ready for display, the DSP board flags the host PC to retrieve the image and display it on the VGA screen. This process is extremely time critical, since new image data is ready to be displayed every 64 milliseconds. The PC must shift the pixels of previously displayed images to make room for the new image and then transfer the image from the DSP memory to the display memory. Thus, the display device must be able to receive, manipulate, and display the image data as quickly as it becomes available. A standard VGA controller and display are used for this task. The VGA provides the graphics resolution, color, and control features necessary to display and manipulate images at fairly rapid rates. It is also a well-established display standard and is now quite affordable.

Smooth update of the display requires page flipping. Michael Abrash ("Graphics Programming," *DDJ*, December 1991) suggested that this is possible on a 640x480 VGA display. However, Michael's technique allows only the upper two-thirds of the display area to be flipped; more area is needed for the wavelet-analyzer display. Since only 512 columns are needed for the display of data, a compromise between horizontal display resolution and page size can be made.

A display with 592 columns and 480 rows is created, with the top 400 rows capable of being page flipped; see Figure 3. The horizontal display dimension is altered by setting the Horizontal Display Enable End and the Offset registers in the VGA's CRTC to display 74 bytes (or 592 pixels) on each horizontal scan. The size of the flip page is determined by setting the Line Compare Count register in the VGA's CRTC to reset the line count after 400 lines have been displayed. Page flipping is controlled by setting the CRTC Start Address High and Low registers to point to the specific page. The routine *Set592x480*, see Listing Six (page 90), performs the setup of the VGA display by initially setting the display to mode 16H (640x480 pixels, 16 colors) and modifying the registers described earlier.

The image data currently displayed on the VGA must be shifted to make room for new image data coming from the DSP memory. The backfields of the latest image alternate between dark and light gray to delineate between the eight unit intervals. These backfields need to be changed to a blue background to indicate past wavelet coefficients when they are shifted. Therefore, a trick is played with the VGA color palette to produce multiple mappings of signal and backfield colors; see Table 2. Write Mode 1 and the Map Mask register are used to enable pixel color changes to occur simultaneously with the movement of image data; see Figure 4.

When the CPU reads the VGA memory, all four bytes in the pixel planes are placed in latch registers. When the CPU writes to the VGA memory with Write Mode 1 enabled, the contents of the latch registers are written back into the pixel planes. Writing latch contents to memory can be blocked by setting the corresponding bits of the Map Mask register to 0. Setting bit 2 of the Map Mask register to 0 and all others to 1 causes the pixels' color registers values be remapped to other values at the time of writing. The remapping causes the light gray and dark gray backfields to be changed to blue while the traces remain yellow.

The routine *ShiftWaveTraces* in Listing Six, performs the operations of image shifting and color modification. The routine moves eight rectangles of the screen image representing the input signal and wavelet-transform coefficient traces. Each rectangle consists of 50 rows of pixel bytes. Although the routine is written in C, an assembly language loop (written using Borland Turbo C 2.0 __emit__ statements) is used to move

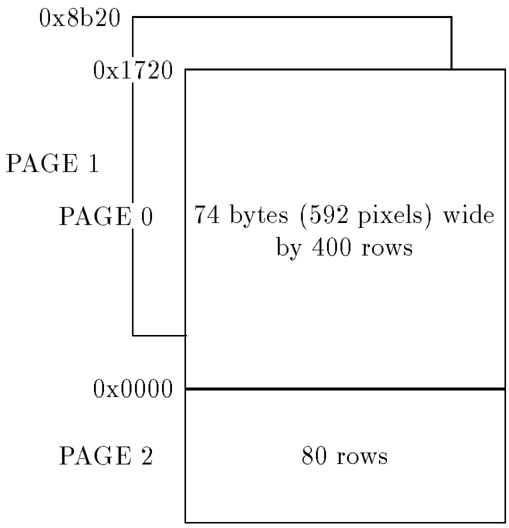

Figure 3: The VGA display map of the wavelet analyzer has three pages, two of which can be flipped.

Palette register	DAC palette register number		Pixel shift and mask operation maps pixel to register:
0	0	(black)	0
1	63	(white)	1
2	60	(light red)	2
3	3	(cyan)	3
4	4	(red)	0
5	5	(magenta)	1
6	6	(olive)	2
7	55	(pale yellow)	3
8	1	(blue)	8
9	62	(yellow)	9
10	1	(blue)	10
11	62	(yellow)	11
12	56	(dark gray)	8
13	62	(yellow)	9
14	7	(light gray)	10
15	62	(yellow)	11

Table 2: The colors of the VGA palette registers can be selected to remap the colors of the signal traces during a data-shift operation so that the background colors are changed while the signal itself appears unaffected.

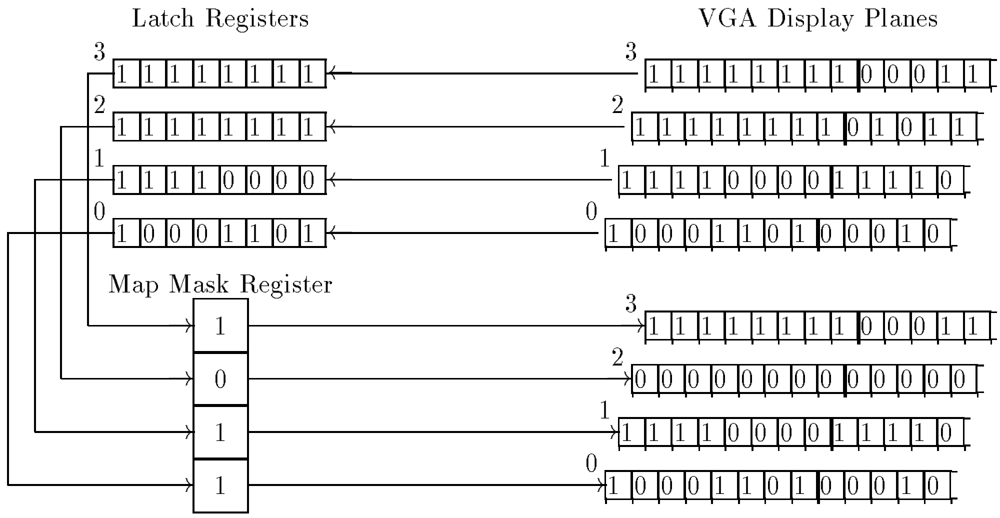

Figure 4: Signal traces displayed on the VGA are shifted and have their colors modified simultaneously through use of the VGA Write Mode and Map Mask registers.

each block of pixel bytes a row at a time. The REP MOVSB instruction is used to move the bytes of each row quickly. The data in the array shiftblocks describes the number of bytes to move in each rectangle and the screen wrap index to keep the source and destination pointers aligned with the block boundaries. The source and destination addresses for the shift are chosen so that the image blocks are moved from the page being displayed to the other page. In this fashion, the screen can be updated with the new image from the DSP without disturbing the screen image.

With the pixel images shifted in the VGA's memory, the routine *GetDSPimage* can now be transfer the pixel image from the DSP's memory to that of the VGA; see Listing Five. The design of the image bitmaps in the DSP memory allow for rapid transfer of the images. The bytes in the DSP image bitmaps correspond directly to the bytes in pixel plane 0 memory of the VGA.

To transfer the pixel data, the Write Mode register is set to mode 0 to allow the external data to be written into the VGA memory. The Bit Mask register is set so that all bits in a byte of VGA memory can be written to. The Map Mask register is set so that only pixel plane 0 (and the least significant bit of the color-register value) is affected by the write. Transfers can now proceed by performing word transfers from the DSP's memory directly into the VGA memory using the DMA capabilities DSP32 and DSP32C; see Figure 5. As in *ShiftWaveTraces*, *GetDSPimage* uses an assembly language loop to optimize the data transfers. The REP INSW instruction is used to move individual rows of pixel bytes.

Making Waves, Again ...

The code presented here is only a portion of a complete wavelet-analyzer package called ANALYZER. The user interface of ANALYZER allows the operator to select the coefficients to be used by the analyzer and view their values and the recursive scaling and wavelet functions defined by those coefficients. The user can also run and halt the wave-transform operation on the incoming signal and pause the screen display at any time. The vertical scale of the input signal and wavelet-transform coefficient traces can be independently adjusted to allow detailed viewing of the data.

The code for the host PC is written in Borland's Turbo C 2.0 with the small memory model. Although the assembly language components of the VGA graphics code are written using _emit_ statements, it should not be too difficult to convert the code over to full assembly language. Assembling the DSP code requires the AT&T WE DSP32-SL Support Software Library. The batch files supplied with the analyzer source code are sufficient to control the library's make utility. Because of its size, the program is available electronically;

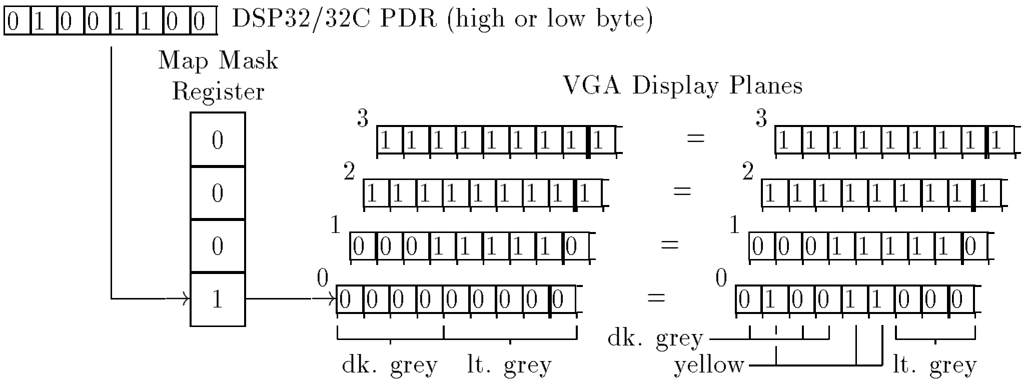

Figure 5: Completed signal-trace bitmaps are moved from the DSP board to the VGA display using the Intel 80x86 REP INSW instruction.

see page 5[3]. The electronic version includes the ANALYZER executable and source, the DSP source and executables, and documentation.

A PC equipped with an 80286 (or higher) processor running at 12 MHz or higher and a VGA card is required. To run the DSP software, the user must supply either a Burr-Brown ZPB32 (DSP32) or ZPB34 (DSP32C) DSP board and a ZPB100 codec board. Other DSP boards can be supported by modifying the DSP I/O control software.

Wave Goodbye (Again)!

The ANALYZER program turns the PC and a DSP board into a test instrument that uses the recursive fast wavelet transform (much as the spectrum analyzer uses the Fourier transform) to analyze signals in real time. With this tool, you can study interactively how different wavelet and scaling functions can interpret a signal. Someday, the wavelet transform analyzer may even reside alongside the oscilloscope and spectrum analyzer as an essential tool in every engineer's toolbox!

Bibliography

Abrash, Michael. "Catching Up." *Dr. Dobb's Journal* (December, 1991).

Bittman, Jim. "Adding the Power of DSP to Your Application." *Dr. Dobb's Journal* (May, 1991).

Cody, Mac A. "The Fast Wavelet Transform." *Dr. Dobb's Journal* (April, 1992).

iAPX 286 Programmer's Reference Manual, Intel Corporation, 1983.

Kliewer, Bradley Dyck. *EGA/VGA: A Programmer's Reference Guide*, second edition. Berkeley, CA: McGraw-Hill, 1990.

[3]The listings, source code, and programs described in this article are available via anonymous ftp from ftp.mv.com in the /pub/ddj/1993.04 directory. The file to download is wavelet.arc.