CHAPTER 27

SPIHT IMAGE COMPRESSION

Thomas W. Fry
Samsung, Global Strategy Group

Scott Hauck
Department of Electrical Engineering
University of Washington

This chapter describes the process of mapping the image compression algorithm SPIHT onto a reconfigurable logic architecture. A discussion of why adaptive logic is required, as opposed to an ASIC, is provided, along with background material on SPIHT. Several discrete wavelet transform hardware architectures are analyzed and evaluated. In addition, two major modifications to the original image compression algorithm, which are required in order to build a reconfigurable hardware implementation, are presented: (1) the storage elements necessary for each wavelet coefficient, and (2) a modification to the original SPIHT algorithm created to parallelize the computation. Also discussed are the effects these modifications have on the final compression results and the trade-offs involved.

The chapter then describes how the updated SPIHT algorithm is mapped onto the Annapolis Microsystems WildStar reconfigurable hardware system. This system is populated with three Virtex-E field-programmable gate array (FPGA) parts and several memory ports. The issues of how the modified algorithm is divided between individual FPGA parts and how data flows through the memories are discussed. Lastly, final results and speedups are presented and evaluated against a comparable microprocessor solution from the time the Annapolis Microsystems WildStar was released.

27.1 BACKGROUND

As NASA deploys each new generation of satellites with more sensors, capturing an ever-larger number of spectral bands, the volume of data being collected begins to outstrip a satellite's ability to transmit data back to Earth. For example, the Terra satellite contains five separate sensors, each collecting up to 36 individual spectral bands. The Tracking and Data Relay Satellite System (TDRSS) ground terminal in White Sands, New Mexico, captures data from these sensors at a limited rate of 150 Mbps [19]. As the number of sensors on a satellite grows and the transmission rates increase, this bandwidth limitation became a driving force for NASA to study methods of compressing images prior to downlinking.

FPGAs are an attractive implementation medium for such a system. Software solutions suffer from performance limitations and power requirements. At the same time, traditional hardware platforms lack the required flexibility needed for postlaunch modifications. After launch, such fixed hardware systems cannot be modified to use newer compression schemes or even to implement bug fixes. In the past, modification of fixed systems in satellites proved to be very expensive [4].

By implementing an image compression kernel in a reconfigurable system, we overcame these shortcomings. Because such a system may be reprogrammed after launch, it does not suffer from conventional hardware's inherit inflexibility. At the same time, the algorithm is computing in custom hardware and can perform at the required processing rates while consuming less power than a traditional software implementation.

This chapter describes the work performed as part of a NASA-sponsored investigation into the design and implementation of a space-bound FPGA-based hyperspectral image compression machine. For this work, the Set Partitioning in Hierarchical Trees (SPIHT) routine was selected as the image compression algorithm. First, we describe the algorithm and discuss the reasons for its selection. Then we describe how the algorithm was optimized for implementation in a specific hardware platform and we present the results.

27.2 SPIHT ALGORITHM

SPIHT is a wavelet-based image compression coder. It first converts an image into its wavelet transform and then transmits information about the wavelet coefficients. The decoder uses the received signal to reconstruct the wavelet and then performs an inverse transform to recover the image. SPIHT was selected because both it and its predecessor, the embedded zerotree wavelet coder, were significant breakthroughs in still-image compression. Both offered significantly improved quality over other image compression techniques such as vector quantization, JPEG, and wavelets combined with quantization, while not requiring training that would have been more difficult to implement in hardware. In short, SPIHT displays exceptional characteristics over several properties all at once [15]:

- Good image quality with a high peak-signal-to-noise ratio (PSNR).
- Fast coding and decoding.
- A fully progressive bitstream.
- Can be used for lossless compression.
- May be combined with error protection (useful in satellite transmissions).
- Ability to code for an exact bitrate or PSNR.

In addition, since the SPIHT algorithm processes an image in two distinct steps—the discrete wavelet transform phase and the coding phase—it provides a natural point at which a hardware implementation may be divided. (The advantage of this property will be seen in Section 27.4.) The rest of this section

describes the basics of wavelets, the discrete wavelet transform, and the SPIHT coding engine.

27.2.1 Wavelets and the Discrete Wavelet Transform

The wavelet transform is a reversible transform on spatial data. The discrete wavelet transform (DWT) is a form appropriate to discrete data, such as the individual points or pixels in an image. DWT runs a high-pass and low-pass filter over the signal in one dimension. This produces a low-pass ("average") version of the data and a high-pass (rapid changes within the average) version. Every other result from each pass is then sampled, yielding two subbands, each of which is one-half the size of the input stream. The result is a new image comprising of a high- and a low-pass subband. These two subbands can be used to fully recover the original image. In the case of a multidimensional signal such as an image, this procedure is repeated in each dimension (Figure 27.1).

The vertical and horizontal transformations break up the image into four distinct subbands. The wavelet coefficients that correspond to the fine details are the LH, HL, and HH subbands. Lower frequencies are represented by the LL subband, which is a low-pass filtered version of the original image [17].

The next wavelet level is calculated by repeating the horizontal and vertical transformations on the LL subband from the previous level. Four new subbands are created from the transformations. The LH, HL, and HH subbands in the next level represent coarser-scale coefficients and the new LL subband is an even smoother version of the original image. It is possible to obtain coarser and coarser scales of the LH, HL, and HH subbands by iteratively repeating the wavelet transformation on the LL subband of each level. Figure 27.2 displays the subband components of an image with three scales of wavelet transformation.

The reverse transformation uses an inverse filter on the final LL subband and the LH, HL, and HH subbands at the same level to recreate the LL subband of the previous level. By iteratively processing each level, the original image may be restored. Figure 27.3 displays a satellite image of San Francisco and its corresponding 3-level DWT. By processing either the wavelet transform or the inverse wavelet transform, these two images may be converted from one into the other and thus may be viewed as equivalent.

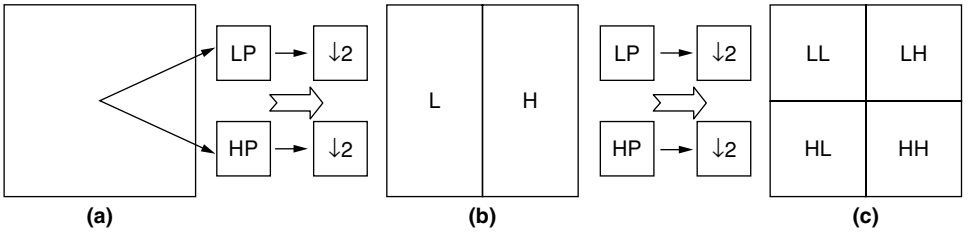

FIGURE 27.1 ■ A 1-level wavelet built by two one-dimensional passes: (a) original image, (b) horizontal pass, and (c) vertical pass.

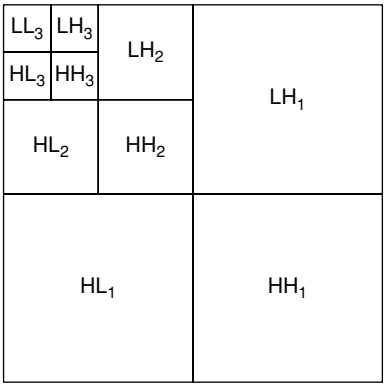

FIGURE 27.2 ■ A 3-level wavelet transform.

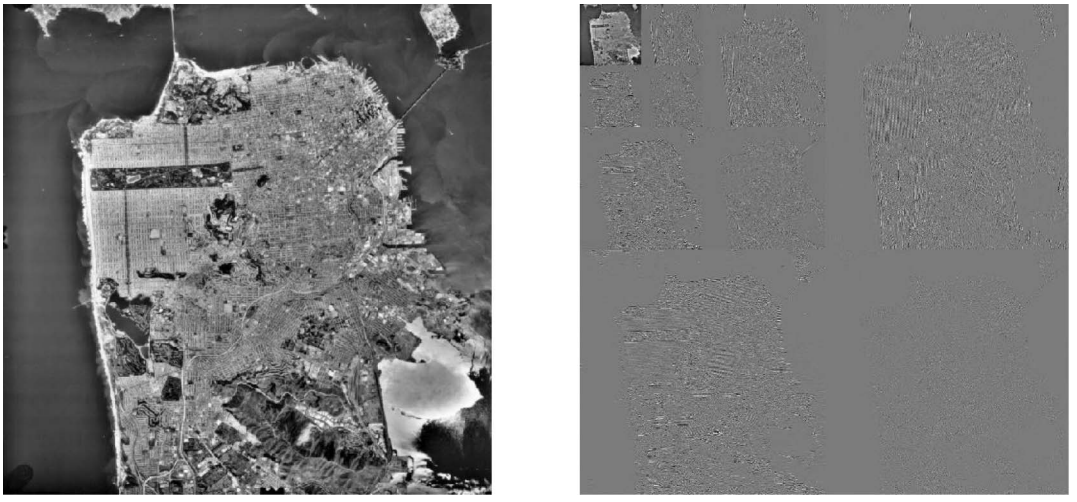

FIGURE 27.3 ■ An image of San Francisco (a) and the resulting 3-level DWT (b).

27.2.2 SPIHT Coding Engine

SPIHT is a method of coding and decoding the wavelet transform of an image. As discussed in the previous section, by coding and transmitting information about the wavelet coefficients, it is possible for a decoder to perform an inverse transformation on the wavelet and reconstruct the original image. A useful property of SPIHT is that the entire wavelet does not need to be transmitted in order to recover the image. Instead, as the decoder receives more information about the original wavelet transform, the inverse transformation yields a better-quality reconstruction (i.e., a higher PSNR) of the original image. SPIHT generates excellent image quality and performance due to three properties of the coding algorithm: partial ordering by coefficient value, taking advantage

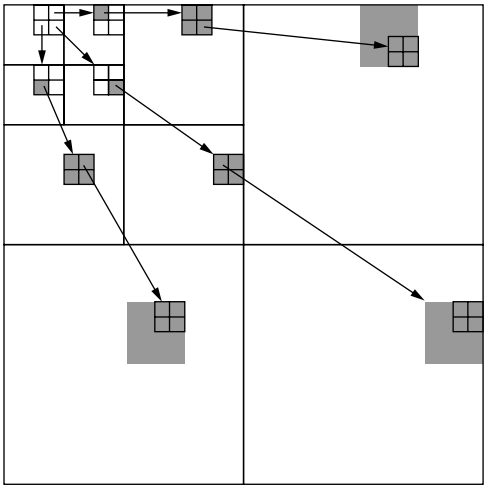

FIGURE 27.4 ■ Spatial orientation trees.

of the redundancies between different wavelet scales, and transmitting data in bit-plane order [14].

Following a wavelet transformation, SPIHT divides the wavelet into *spatial orientation trees* (Figure 27.4). Each node in a tree corresponds to an individual pixel. The offspring of a pixel are the four pixels in the same spatial location of the same subband at the next finer scale of the wavelet. Pixels at the finest scale of the wavelet are the leaves of the tree and have no children. Every pixel is part of a 2×2 block with its adjacent pixels. Blocks are a natural result of the hierarchical trees because every pixel in a block shares the same parent pixel. Also, the upper-left pixel of each 2×2 block at the root of the tree has no children since there are only three subbands at each scale and not four. Figure 27.4 shows how the pyramid is defined. Arrows point to the offspring of an individual pixel and the grayed blocks show all of the descendents for a specific pixel at every scale.

SPIHT codes a wavelet by transmitting information about the significance of a pixel. By stating whether or not a pixel is above some threshold, information about that pixel's value is implied. Furthermore, SPIHT transmits information stating whether a pixel or any of its descendents are above a threshold. If the statement proves false, all of the pixel's descendants are known to be below that threshold level and they do not need to be considered during the rest of the current pass. At the end of each pass, the threshold is divided by two and the algorithm continues. In this manner, information about the most significant bits of the wavelet coefficients will always precede information on lower-order significant bits, which is referred to as *bit-plane ordering*.

Information stating whether or not a pixel is above the current threshold or is being processed at the current threshold is contained in three lists: the *list of insignificant pixels* (LIP), the *list of insignificant sets* (LIS) and the *list of significant pixels* (LSP). The LIP are pixels that are currently being processed

but are not yet above the threshold. The LIS are pixels that are currently being processed but none of their descendents are yet above the current threshold and so they are not being processed. Lastly, the LSP are pixels that were already stated to be above a previous threshold level and whose value at each bit plane is now transmitted.

Figure 27.5 is the algorithm from the original SPIHT paper [14], modified to reflect changes (discussed later in the chapter) referring to 2×2 block information. $S_n(i, j)$ represents if the pixel (i, j) is greater than the current threshold, and $S_n(D(i, j))$ states if any of the pixel's (i, j) descendents are greater than the current threshold.

There are three important concepts to take from the SPIHT algorithm. First, as the encoder sequentially steps through the image, it inserts or deletes pixels from the three lists. All of the information required to keep track of the lists is output to the decoder, allowing the decoder to generate and maintain an identical list order as the encoder. For the decoder to reproduce the steps taken by the encoder we merely need to replace the `output` statements in the encoder's algorithm with `input` for the decoder's algorithm.

Second, the bitstream produced is naturally progressive. A progressive bitstream is one that can be cut off at any point and still be valid. As the decoder steps through the coding algorithm, it gathers finer and finer detail about the original wavelet transform. The decoder can stop at any point and perform an inverse transform with the wavelet coefficients it has currently reconstructed. Progressive bitstreams can also be reduced to an arbitrary size or be cut off during transmission and still produce a valid image. Such a property is very useful in satellite transmissions.

---

1. **Initialization: output** n = floor[$\log_2(\max_{(i,j)}\{|c_{i,j}|\})$]; clear the LSP list, add the root pixels to the LIP list and root pixels with descendants to LIS.
2. **Sorting Pass:**
   2.1 for each entry (i,j) in the LIP:
      2.1.1 **output** $S_n(i,j)$;
      2.1.2 If $S_n(i,j) = 1$, move (i,j) to the LSP list and **output** its sign
   2.2 for each entry (i,j) in the LIS:
      2.2.1 If one of the pixels in (i,j)'s block is not in LIP but all are in LIS:
         output S_n(all descendants of the current block);
         if none are significant, skip 2.2.2.
      2.2.2 **Output** $S_n(D(i,j))$
         if $S_n(D(i,j)) = 1$, then
         for each of (i,j) immediate children (k,l):
            output $S_n(k,l)$;
            add (k,l) to the LIS for the current pass
            if $S_n(k,l) = 1$, add (k,l) to the LSP and **output** its sign
                  else add (k,l) to the LIP
3. **Refinement Pass:** for each entry (i,j) in LSP, except ones inserted in the current pass, **output** the n^{th} most significant bit of (i,j).
4. **Quantization-step Update:** decrement n by 1 and go to Step **2**.

FIGURE 27.5 ■ SPIHT coding algorithm.

Third, and the concept that has the largest impact on building a hardware platform, the SPIHT algorithm develops an individual list order to transmit information within each bit plane. This ordering is implicitly created from the threshold information discussed before—the order in which each pixel enters each list determines the transmission order for each image. As a result, each image will transmit wavelet coefficients in an entirely different order. Slightly better PSNRs are achieved with this dynamic ordering of the wavelet coefficients.

The SPIHT algorithm in Figure 27.5, which creates the individual list ordering, is inherently sequential. As a result, SPIHT cannot be significantly parallelized in hardware. This drawback greatly limits the performance of any SPIHT implementation in hardware. To get around this limitation and improve performance, it was necessary to parallelize the SPIHT algorithm and essentially create a new image compression algorithm. These changes and the trade-offs involved are described in Section 27.3.3.

27.3 DESIGN CONSIDERATIONS AND MODIFICATIONS

To fully take advantage of the high performance a custom hardware implementation of SPIHT could yield, the software specifications had to be examined and adjusted where they either performed poorly in hardware or did not make the most of the resources available. Here we review the three major factors taken under consideration while evaluating how to create a hardware implementation of the SPIHT algorithm on an adaptive computing platform.

The first factor was to determine what discrete wavelet transform architecture to use. Section 27.3.1 provides a summary of the DWTs considered, showing how memory and communication requirements helped dictate the structure chosen. Section 27.3.2 describes the fixed-point precision optimization performed for each wavelet coefficient and the final data representation employed. Section 27.3.3 explains how the SPIHT algorithm was altered to vastly speed up the hardware implementation.

27.3.1 Discrete Wavelet Transform Architectures

One of the benefits of the SPIHT algorithm is its use of the discrete wavelet transform, which had existed for several years prior to this work. As a result, numerous studies on how to create a DWT hardware implementation were available for review. Much of this work on DWTs involved parallel platforms to save both memory access and computations [5, 12, 16].

The most basic architecture is the basic folded architecture. The one-dimensional DWT entails demanding computations, which involve significant hardware resources. Since the horizontal and vertical passes use identical finite impulse response (FIR) filters, most two-dimensional DWT architectures implement folding to reuse logic for each dimension [6]. Figure 27.6 illustrates how folded architectures use a one-dimensional DWT to realize a two-dimensional DWT.

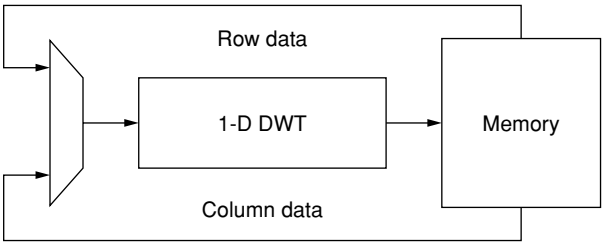

FIGURE 27.6 ■ A folded architecture.

Although the folded architecture saves hardware resources, it suffers from high memory bandwidth. For an $N \times N$ image there are at least $2N^2$ read-and-write cycles for the first wavelet level. Additional levels require rereading previously computed coefficients, further reducing efficiency.

To lower the memory bandwidth requirements needed to compute the DWT, we considered several alternative architectures. The first was the Recursive Pyramid Algorithm (RPA) [21]. RPA takes advantage of the fact that the various wavelet levels run at different clock rates. Each wavelet level requires one-quarter of the time that the previous level needed because at each level the size of the area under computation is reduced by one-half in both the horizontal and vertical dimensions. Thus, it is possible to store previously computed coefficients on-chip and intermix the next level's computations with the current level's. A careful analysis of the runtime yields $(4^*N^2)/3$ individual memory load and store operations for an image. However, the algorithm has huge on-chip memory requirements and demands a thorough scheduling process to interleave the various wavelet levels.

Another method to reduce memory accesses is the partitioned DWT, which breaks the image into smaller blocks and computes several scales of the DWT at once for each block [13]. In addition, the algorithm made use of wavelet lifting to reduce the DWT's computational complexity [18]. By partitioning an image into smaller blocks, the amount of on-chip memory storage required was significantly reduced because only the coefficients in the block needed to be stored. This approach was similar to the RPA, except that it computed over sections of the image at a time instead of the entire image at once. Figure 27.7, from Ritter and Molitor [13], illustrates how the partitioned wavelet was constructed.

Unfortunately, the partitioned approach suffers from blocking artifacts along the partition boundaries if the boundaries were treated with reflection.[1] Thus, pixels from neighboring partitions were required to smooth out these boundaries. The number of wavelet levels determined how many pixels beyond a subimage's boundary were needed, since higher wavelet levels represent data

[1] An FIR filter generally computes over several pixels at once and generates a result for the middle pixel. To calculate pixels close to an image's edge, data points are required beyond the edge of the image. Reflection is a method that takes pixels toward the image's edge and copies them beyond the edge of the actual image for calculation purposes.

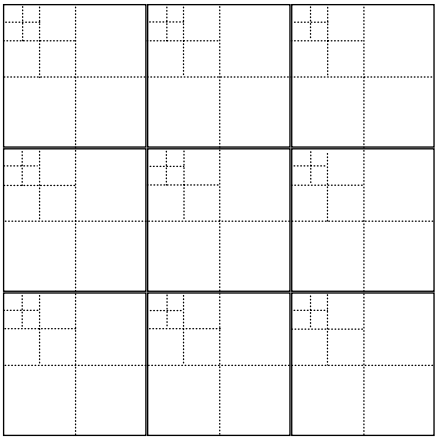

FIGURE 27.7 ■ The partitioned DWT.

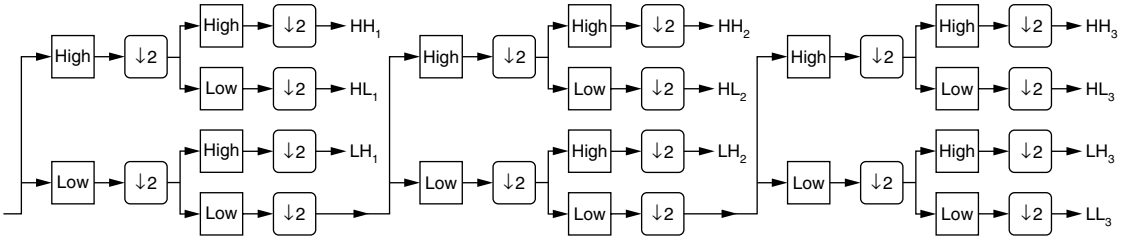

FIGURE 27.8 ■ A generic 2D biorthogonal DWT.

from a larger image region. To compensate for the partition boundaries, the algorithm processed subimages along a single row to eliminate multiple reads in the horizontal direction. Overall data throughputs of up to 152 Mbytes/second were reported with the partitioned DWT.

The last architecture we considered was the generic 2D biorthogonal DWT [3]. Unlike previous designs, the generic 2D biorthogonal DWT did not require FIR filter folding or on-chip memories as the Recursive Pyramid design. Nor did it involve partitioning an image into subimages. Instead, the architecture created separate structures to calculate each wavelet level as data were presented to it, as shown in Figure 27.8. The design sequentially read in the image and computed the four DWT subbands. As the LL_1 subband became available, the coefficients were passed to the next stage, which calculated the next coarser level subbands, and so on.

For larger images that required several individual wavelet scales, the generic 2D biorthogonal DWT architecture consumed a tremendous amount of on-chip resources. With SPIHT, a 1024×1024 pixel image computes seven separate wavelet scales. The proposed architecture would employ 21 individual high- and low-pass FIR filters. Since each wavelet scale processed data at different rates, some control complexity would be inevitable. The advantage of the architecture

was much lower on-chip memory requirements and full utilization of the memory's bandwidth, since each pixel was read and written only once.

To select a DWT, each of the architectures discussed before were reevaluated against our target hardware platform (discussed below). The parallel versions of the DWT saved some memory bandwidth. However, additional resources and more complex scheduling algorithms became necessary. In addition, some of the savings were minimal since each higher wavelet level is one-quarter the size of the previous wavelet level. In a 7-level DWT, the highest 4 levels compute in just 2 percent of the time it takes to compute the first level. Other factors considered were that the more complex DWT architectures simply required more resources than a single Xilinx Virtex 2000E FPGA (our target device) could accommodate, and that enough memory ports were available in our board to read and write four coefficients at a time in parallel.

For these reasons, we did not select a more complex parallel DWT architecture, but instead designed a simple folded architecture that processes one dimension of a single wavelet level at a time. In the architecture created, pixels are read in horizontally from one memory port and written directly to a second memory port. In addition, pixels are written to memory in columns, inverting the image along the 45-degree line. By utilizing the same addressing logic, pixels are again read in horizontally and written vertically. However, since the image was inverted along its diagonal, the second pass will calculate the vertical dimension of the wavelet and restore the image to its original orientation.

Each dimension of the image is reduced by half, and the process iteratively continues for each wavelet level. Finally, the mean of the LL subband is calculated and subtracted from itself. To speed up the DWT, the design reads and writes four rows at a time. Figure 27.9 illustrates the architecture of the DWT phase.

Since every pixel is read and written once and the design processes four rows at a time, for an $N \times N$-size image both dimensions in the lowest wavelet level compute in $2^*N^2/4$ clock cycles. Similarly, the next wavelet level processes the image in one-quarter the number of clock cycles as the previous level. With an infinite number of wavelet levels, the image processes in:

$$\sum_{l=1}^{\infty} \frac{2 \cdot N^2}{4^l} = \frac{3}{4} \cdot N^2 \tag{27.1}$$

Thus, the runtime of the DWT engine is bounded by three-quarters of a clock cycle per pixel in the image. This was made possible because the memory ports in the system allowed four pixels to be read and written in a single clock cycle.

It is very important to note that many of the parallel architectures designed to process multiple wavelet levels simultaneously run in more than one clock cycle per image. Also, because of the additional resources required by a parallel implementation, computing multiple rows at once becomes impractical. Given more resources, the parallel architectures discussed previously could process multiple rows at once and yield runtimes lower than three-quarters of a clock cycle per pixel. However, the FPGAs available in the system used, although state of the art at the time, did not have such extensive resources.

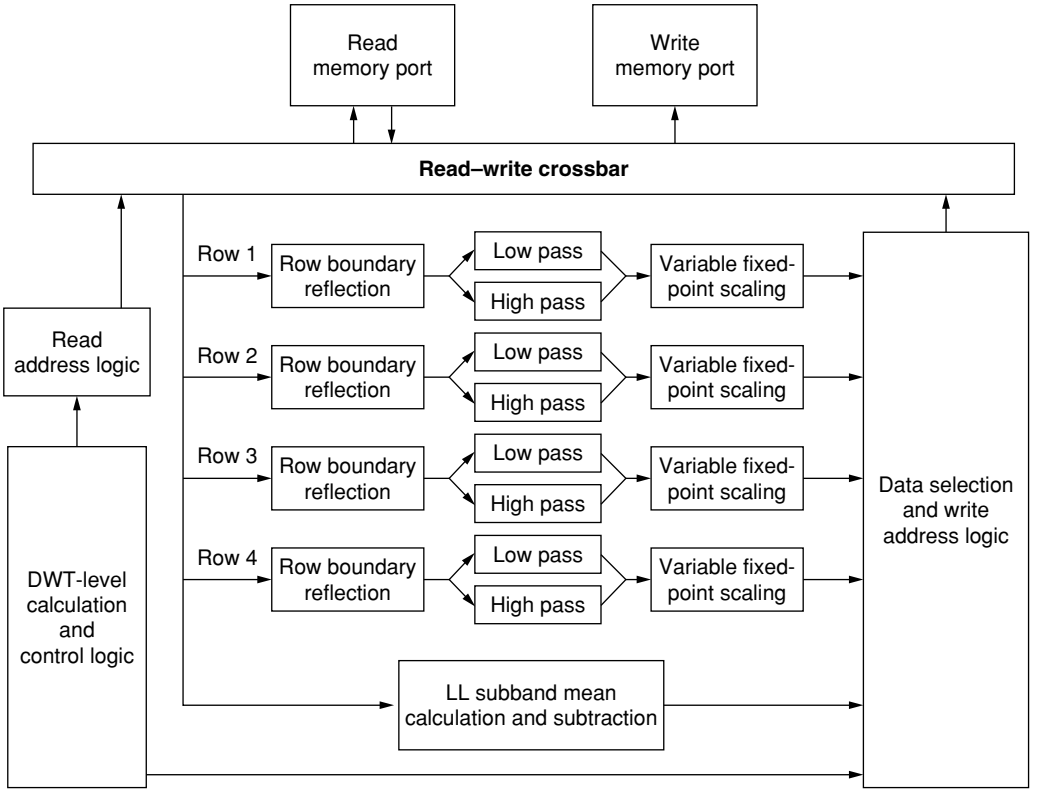

FIGURE 27.9 ■ A discrete wavelet transform architecture.

By keeping the address and control logic simple, there were enough resources on the FPGA to implement 8 distributed arithmetic FIR filters [23] from the Xilinx Core library. The FIR filters required significant FPGA resources, approximately 8 percent of the Virtex 2000E FPGA for each high- and low-pass FIR filter. We chose the distributed arithmetic FIR filters because they calculate a new coefficient every clock cycle, and this contributed to the system being able to process an image in three-quarters of a clock cycle per pixel.

27.3.2 Fixed-point Precision Analysis

The next major consideration was how to represent the wavelet coefficients in hardware. The discrete wavelet transform produces real numbers as the wavelet coefficients, which general-purpose computers realize as floating-point numbers. Traditionally, FPGAs have not employed floating-point numbers for several reasons:

- Floating-point numbers require variable shifts based on the exponential description, and variable shifters perform poorly in FPGAs.

- Floating-point numbers consume enormous hardware resources on a limited-resource FPGA.
- Floating point is often unnecessary for a known dataset.

At each wavelet level of the DWT, coefficients have a fixed range. Therefore, we opted for a fixed-point numerical representation—that is, one where the decimal point's position is predefined. With the decimal point locked at a specific location, each bit contributes a known value to the number, which eliminates the need for variable shifters. However, the DWT's filter bank was unbounded, meaning that the range of possible numbers increases with each additional wavelet level.

We chose to use the FIR filter set from the original SPIHT implementation. An analysis of the coefficients of each filter bank showed that the two-dimensional low-pass FIR filter at most increases the range of possible numbers by a factor of 2.9054. This number is the increase found from both the horizontal and the vertical directions. It represents how much larger a coefficient at the next wavelet level could be if the previous level's input wavelet coefficients were the maximum possible value and the correct sign to create the largest possible filter output. As a result, the coefficients at various wavelet levels require a variable number of bits above the decimal point to cover their possible ranges.

Table 27.1 illustrates the various requirements placed on a numerical representation for each wavelet level. The Factor and Maximum Magnitude columns demonstrate how the range of possible numbers increases with each level for an image starting with 1 byte per pixel. The Maximum Bits column shows the maximum number of bits (with a sign bit) necessary to represent the numeric range at each wavelet level. The Maximum Bits from Data column represents the maximum number of bits required to encode over one hundred sample images obtained from NASA. These numbers were produced via software simulation on this sample dataset.

In practice, the magnitude of the wavelet coefficients does not grow at the maximum theoretical rate. To maximize efficiency, the Maximum Bits from Data values were used to determine what position the most significant bit must stand for. Since the theoretical maximum is not used, an overflow situation may occur.

TABLE 27.1 ■ Fixed-point magnitude calculations

Wavelet level	Factor	Maximum magnitude	Maximum bits	Maximum bits from data
Input image	1	255	8	8
0	2.9054	741	11	11
1	8.4412	2152	13	12
2	24.525	6254	14	13
3	71.253	18170	16	14
4	207.02	52789	17	15
5	601.46	153373	19	16
6	1747.5	445605	20	17

To compensate, the system flags overflow occurrences as an error and truncates the data. However, after examining hundreds of sample images, no instances of overflow occurred, and the data scheme used provided enough space to capture all the required data.

If each wavelet level used the same numerical representation, they would all be required to handle numbers as large as the highest wavelet level to prevent overflow. However, since the lowest wavelet levels never encounter numbers in that range, several bits at these levels would not be used and therefore wasted.

To fully utilize all of the bits for each wavelet coefficient, we introduced the concept of *variable fixed-point* representation. With variable fixed-point we assigned a fixed-point numerical representation for each wavelet level optimized for that level's expected data size. In addition, each representation differed from one another, meaning that we employed a different fixed-point scheme for each wavelet level. Doing so allowed us to optimize both memory storage and I/O at each wavelet level to yield maximum performance.

Once the position of the most significant bit was found for each wavelet level, the number of precision bits needed to accurately represent the wavelet coefficients had to be determined. Our goal was to provide enough bits to fully recover the image and no more. Figure 27.10 displays the average PSNRs for several recovered images from SPIHT using a range of bit widths for each coefficient.

An assignment of 16 bits per coefficient most accurately matched the full-precision floating-point coefficients used in software, up through perfect reconstruction. Previous wavelet designs we looked at focused on bitrates less than 4 bits per pixel (bpp) and did not consider rounding effects on the wavelet transformation for bitrates greater than 4 bpp. These studies found this lower bitrate acceptable for lossy SPIHT compression [3].

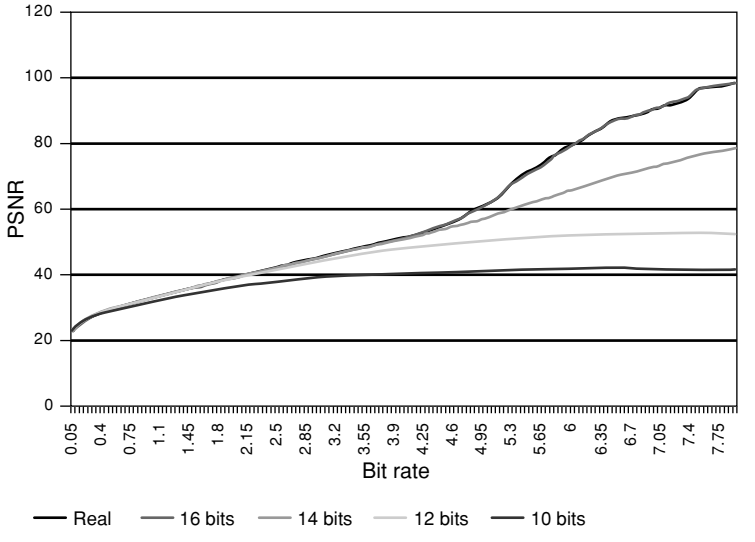

FIGURE 27.10 ■ PSNR versus bitrate for various coefficient sizes.

TABLE 27.2 ■ Final variable fixed-point representation

Wavelet level	Integer bits	Fractional bits
Input image	10	6
0	11	5
1	12	4
2	13	3
3	14	2
4	15	1
5	16	0
6	17	−1

Instead, we chose a numerical representation that retains the equivalent amount of information as a full floating-point number during wavelet transformation. By doing so, it was possible to perfectly reconstruct an image given a high enough bitrate. In other words, we allowed for a lossless implementation. Table 27.2 provides the number of integer and fractional bits allocated for each wavelet level. The number of integer bits also includes 1 extra bit for the sign value. The highest wavelet level's 16 integer bits represent positions 17 to 1, with no bit assigned for the 0 position.

27.3.3 Fixed Order SPIHT

The last major factor we took under consideration was how to parallelize the SPIHT algorithm for use in hardware. As discussed in Section 27.2, SPIHT computes a dynamic ordering of the wavelet coefficients as it progresses. By always adding pixels to the end of the LIP, LIS, and LSP, coefficients most critical to constructing a valid wavelet are generally sent first, while less critical coefficients are placed later in the lists. Such an ordering yields better image quality for bitstreams that end in the middle of a bit plane. The drawback of this ordering is that every image has a unique list order determined by the image's wavelet coefficient values.

By analyzing the SPIHT algorithm, we were able to conclude that the data a block of coefficients contributes to the final SPIHT bitstream is fully determined by the following set of localized information:

- The 2×2 block of coefficients
- Their immediate children
- The maximum magnitude of the four subtrees

As a result, we were able to show that every block of coefficients could be calculated independently and in parallel of one another. We were also able to determine that, if we could parallelize the computation of these coefficients, the final hardware implementation would operate at a much higher throughput. However, we were not able to take advantage of this parallelism because in SPIHT

the order in which a block's data is inserted into the bitstream is not known, since it depends on the image's unique ordering. Only once the order is determined is it possible to produce a valid SPIHT bitstream from the information listed previously.

Unfortunately, the algorithm employed to calculate the SPIHT ordering of coefficients is sequential. The computation steps over the coefficients of the image multiple times within each bit plane and dynamically inserts and removes coefficients from the LIP and LIS lists. Such an algorithm is not parallelizable in hardware. As a result, many of the speedups a custom hardware implementation may produce would be lost. Instead, any hardware implementation we could develop would need to create the lists in an identical manner as the software implementation. This process would require many clock cycles per block of coefficients, which would significantly limit the throughput of any SPIHT implementation in hardware.

To remove this limitation and design a faster system, we created a modification to the original algorithm called *Fixed Order SPIHT*. Fixed Order SPIHT is similar to the SPIHT algorithm shown in Figure 27.5, except that the order of the LIP, LIS, and LSP lists is fixed and known beforehand. Instead of inserting blocks of coefficients at the end of the lists, they are inserted in a predetermined order. For example, block A will always appear before block B, which is always before block C, regardless of the order in which A, B, and C were added to the lists. The order of Fixed Order SPIHT is based upon the Morton scan ordering discussed in Algazi and Estes [1].

Fixed Order SPIHT removed the need to calculate the ordering of coefficients within each bit plane and allowed us to create a fully parallel version of the original SPIHT algorithm. Such a modification increased the throughput of a hardware encoder by more than an order of magnitude at the cost of a slightly lower PSNR within each bit plane. Figure 27.11 outlines the new version of SPIHT we created. The final bitstream generated is precisely the same as the bitstream generated from the original SPIHT algorithm except that data will appear in a different order within each bit plane.

By using the algorithm in Figure 27.11 instead of the original sequential algorithm in Figure 27.8, the final datastream can be computed in one pass through the image instead of multiple passes. In addition, each pixel block is coded in parallel, which yields significantly faster compression times with FPGAs.

The advantage of this method is that at the end of each bit plane, the exact same data will have been transmitted, just in a different order. Thus, at the end of each bit plane the PSNR of Fixed Order SPIHT will match that of the original SPIHT algorithm, as shown in Figure 27.12. Since the length of each bitstream is fairly short within the transmitted datastream, the PSNR curve of Fixed Order SPIHT very closely matches that of the original algorithm. The maximum loss in quality between Fixed Order SPIHT and the original SPIHT algorithm found was 0.2 dB. This is the maximum loss any image in our sample set displayed over any bitrate from 0.05 to 8.00 bpp.

For a more complete discussion on Fixed Order SPIHT, refer to Fry [8].

```
1. Bit-plane calculation: for each 2×2 block of pixels (i,j) in a Morton
   Scan Ordering
   1.1 for each threshold level n from the highest level to the lowest
       1.1.1 if (i,j) is a root and Max((i,j)) >= n
              add all four pixels to the LIP
       1.1.2 if (i,j) is not a root and Max((i,j)) >= previous n
              for each pixel p in the block
                if p < previous n
                  add p to the LIP
                else
                  add p to the LSP
       1.1.3 if (i,j) is not a leaf and Max((i,j) ) >= n
              add all four pixel to the LIS unless (i,j) is a root, then
              just add the three with children
       1.1.4 if all four pixels are in LIS and at least one is not in the LIP
              if at least one pixel will be removed from the LIS at this level
                output a '0' to the LIS stream
              else
                output a '1' to the LIS stream
       1.1.5 for each pixel p in the LIP
               if p >= n
                 output a '1' and the sign of p to the LIP stream
                 remove p from the LIP and add it to the LSP
               else
                 output a '0' to the LIP stream
       1.1.6 for each pixel p in the LIS
               if child max(p) >= n
                 output a '1' to the LIS stream
                 remove p from the LIS
                 for each child (k,l) of p
                   if (k,l) >= n
                     output a '1' and the sign of (k,l) to the LIS stream
                   else
                     output a '0' to the LIS stream
               else
                 output a '0' to the LIS stream
       1.1.7 for each pixel p in the LSP
              output the value of p at the bit plane n to the LSP stream
2. Grouping phase: for each threshold level n from the highest level to
   the lowest
   2.1 output the LIP stream at threshold level n to the final data stream
   2.2 output the LIS stream at threshold level n to the final data stream
   2.3 output the LSP stream at threshold level n to the final data stream
```

FIGURE 27.11 ■ Fixed Order SPIHT.

27.4 HARDWARE IMPLEMENTATION

In the following subsections we first describe the target hardware platform that the SPIHT algorithm was mapped onto. Next, we present an overview of the implementation and a detailed description of the three major steps of the

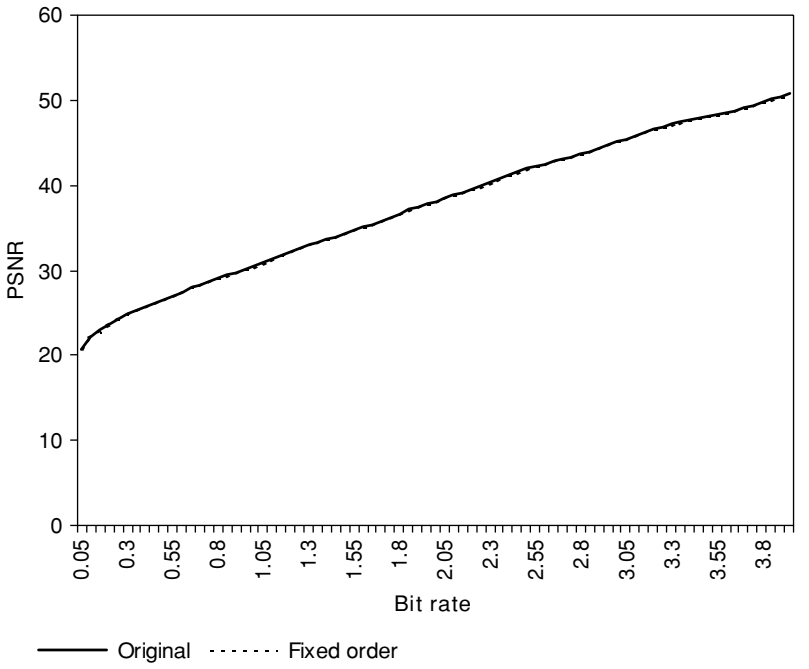

FIGURE 27.12 ■ A comparison of original SPIHT and Fixed Order SPIHT.

computation. A thorough understanding of the target platform is required because it strongly influenced the SPIHT implementation created.

27.4.1 Target Hardware Platform

The target platform was the WildStar FPGA processor board developed by Annapolis Microsystems [2]. Shown in Figure 27.13, it consists of three Xilinx Virtex 2000E FPGAs—PE 0, PE 1, and PE 2—and operates at rates of up to 133 MHz. The board makes available 48 MBytes of memory through 12 individual memory ports, between 32 and 64 bits wide, yielding a throughput of up to 8.5 GBytes/sec. Four shared memory blocks connect the Virtex chips through a crossbar. By switching a crossbar, several MBytes of data are passed between the chips in just a few clock cycles.

The Xilinx Virtex 2000E FPGA allows for 2 million gate designs [22]. For extra on-chip memory, the FPGAs contain 160 asynchronous dual-ported BlockRAMs. Each BlockRAM stores 4096 bits of data and is accessible in 1-, 2-, 4-, 8-, or 16-bit-wide words. Because they are dual ported, the BlockRAMs function well as first in, first outs (FIFOs). A PCI bus connects the board to a host computer.

27.4.2 Design Overview

The architecture constructed consisted of three phases: wavelet transform, maximum magnitude calculation, and Fixed Order SPIHT coding. Each phase

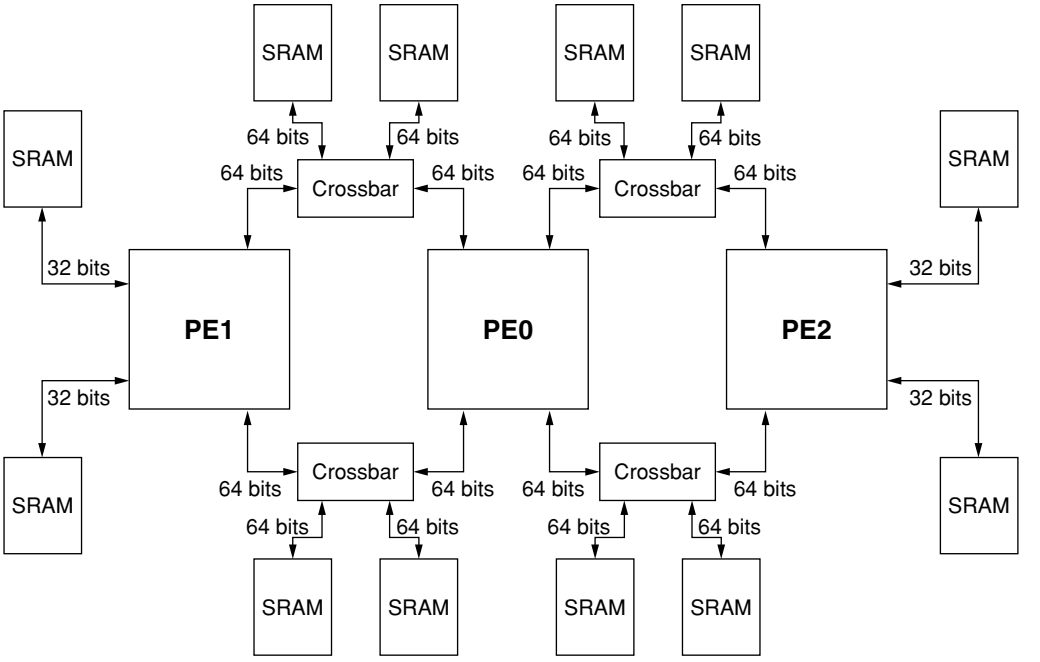

FIGURE 27.13 ■ A block diagram of the Annapolis Microsystems WildStar board.

was implemented in one of the three Virtex chips. By instantiating each phase on a separate chip, separate images could be operated on in parallel. Data was transferred from one phase by the next through the shared memories. The decision on how to break up the phases came naturally from the resources available in each FPGA and the requirements of each section. The DWT and the SPIHT coding phases each required close to the full resources of a single FPGA, and the maximum magnitude phase needed to be completed prior to the SPIHT coding phase. These characteristics of the algorithm and system naturally lead to placing the three phases on the three separate FPGAs.

The architecture was also designed in this manner because once processing in a phase is complete, the crossbar mode could be switched and the data calculated would be accessible to the next chip. By coding a different image in each phase simultaneously, the throughput of the system is determined by the slowest phase, while the latency of the architecture is the sum of the three phases. Figure 27.14 illustrates the architecture of the system.

27.4.3 Discrete Wavelet Transform Phase

As discussed in Section 27.3.1, after implementing each algorithm in hardware we chose a simple folded architecture, which matched the bandwidth, memory, and chip capacities of the target board well. The results of this phase are stored into memory and passed to the maximum magnitude phase.

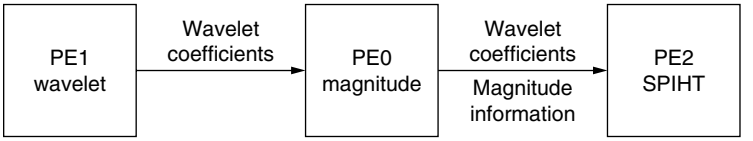

FIGURE 27.14 ■ An overview of the architecture.

27.4.4 Maximum Magnitude Phase

Once the DWT is complete, the next phase prepares and organizes the image into a form easily readable by the parallel version of the SPIHT coder. Specifically, the maximum magnitude phase calculates and rearranges the following information for the next phase:

- The maximum magnitude of each of the four child trees
- The absolute value of the 2×2 block of coefficients
- A sign value for each coefficient in the block
- The threshold level when the block is first inserted into the LIS by its parent
- Threshold and sign data of each of the 16 child coefficients
- Reorder the wavelet coefficients into a Morton Scan Ordering

The SPIHT coding phase shares two 64-bit memory ports with the maximum magnitude phase, allowing it to read 128 bits on each clock cycle. The data just listed can fit into these two memory ports. By doing so on every clock cycle the SPIHT coding phase will be able to read and process an entire block of data. The data that the maximum magnitude phase calculates is shown in Figure 27.15.

To calculate the maximum magnitude of all coefficients below a node in the spatial orientation trees, the image must be scanned in depth-first search order [7]. With a depth-first search, whenever a new coefficient is read and considered, all of its children will have already been read and the maximum coefficient so far is known. On every clock cycle the new coefficient is compared to and updates the current maximum. Because PE 0 (the maximum magnitude phase) uses 32-bit-wide memory ports, it can read half a block at a time.

The state machine, which controls how the spatial orientation trees are traversed, reads one-half of a block as it descends the tree, and the other half as it ascends the tree. By doing so all of the data needed to compute the maximum magnitude for the current block is available as the state machine ascends back up the spatial orientation tree. In addition, the four most recent blocks of each level are saved onto a stack so that all 16 child coefficients are available to the parent block.

Figure 27.16 demonstrates the algorithm. The current block, maximum magnitude for each child, and 16 child coefficients are shown on the stack. Light gray blocks are coefficients previously read and processed. Dark gray blocks are coefficients currently being read. In this example, the state machine has just finished reading the lowest level and has ascended to the second wavelet level.

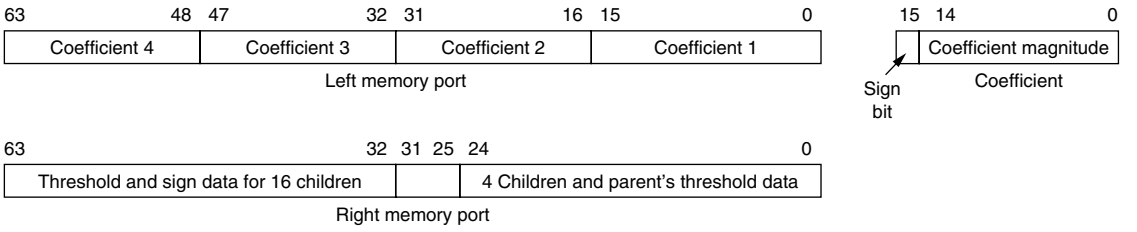

FIGURE 27.15 ■ Data passed to the SPIHT coder to calculate a single block.

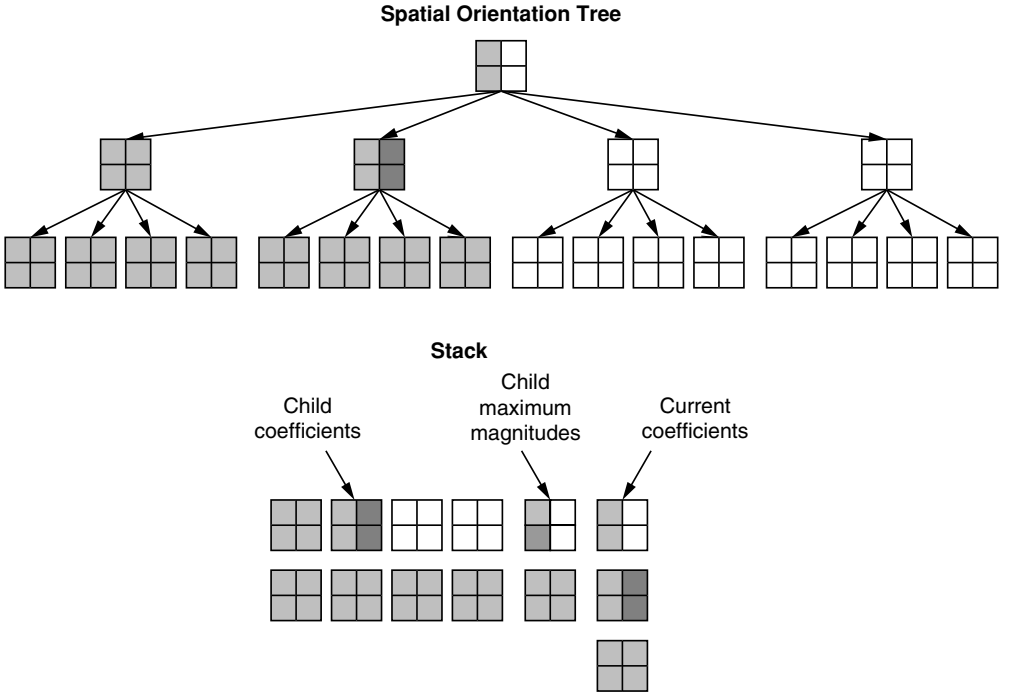

FIGURE 27.16 ■ A depth-first search of the spatial orientation trees.

The second block in the second level is now complete, and its maximum magnitude can now be calculated, shown as the dark gray block in the stack's highest level. In addition, the 16 child coefficients in the lowest level were saved and are available. There are no child values for the lowest level since there are no children.

Another benefit of scanning the image in a depth-first search order is that Morton Scan Ordering is naturally realized within each level, although it is intermixed between levels. By writing data from each level to a separate area of memory and later reading the data from the highest wavelet level to the lowest, the Morton

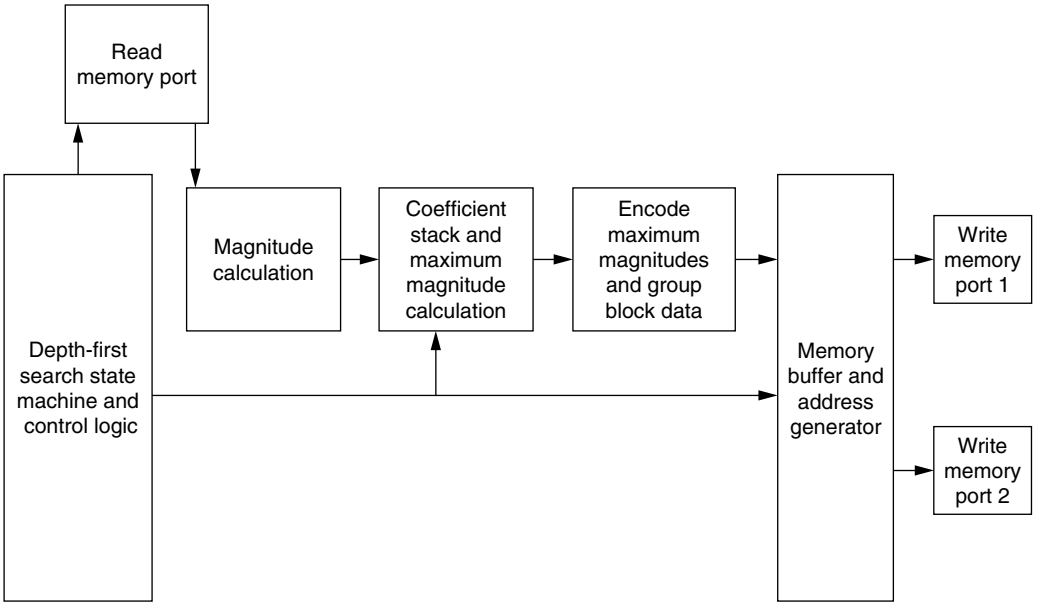

FIGURE 27.17 ■ A block diagram of the SPIHT maximum magnitude phase.

Scan Ordering is naturally realized. A block diagram of the maximum magnitude phase is provided in Figure 27.17. Since two pixels are read together and the image is scanned only once, the runtime of this phase is half a clock cycle per pixel. Because the maximum magnitude phase computes in less time than the wavelet phase, the throughput of the overall system is not affected.

27.4.5 The SPIHT Coding Phase

The final SPIHT coding phase performs the Fixed Order SPIHT encoding in parallel, based on the data from the maximum magnitude phase. Coefficient blocks are read from the highest wavelet level to the lowest. As information is loaded from memory it is shifted from the variable fixed-point representation to a common fixed-point representation for every wavelet level. Once each block has been adjusted to an identical numerical representation, the parallel version of SPIHT is used to calculate what information each block will contribute to each bit plane.

The information is grouped and counted before being added to three separate variable FIFOs for each bit plane. The data that the variable FIFO components receive range in size from 0 to 37 bits, and the variable FIFOs arrange the block data into regular sized 32-bit words for memory access. Care is also taken to stall the algorithm if any of the variable FIFOs becomes too full.

Data from each buffer is output to a fixed location in memory and the number of bits in each bitstream is output as well. Given that data is added dynamically to each bitstream, there needs to be a dynamic scheduler to select which buffer

should be written to memory. Since there are a large number of FIFOs that all require a BlockRAM, the FIFOs are spread across the FPGA, and some type of staging is required to prevent a signal from traveling too far. The scheduler selects which FIFO to read based on both how full a FIFO is and when it was last accessed.

Our studies showed that the LSP bitstream is roughly the same size of the LIP and LIS streams combined. Because of this the LSP bitstreams transfer more data to memory than the other two lists. In our design the LIP and LIS bitstreams share a memory port while the LSP stream writes to a separate memory port. Since a 2×2 block of coefficients is processed every clock cycle, the design takes one-quarter of a clock cycle per pixel, which is far less than the three-quarters of a clock cycle per pixel for the DWT. The block diagram for the SPIHT coding phase is given in Figure 27.18.

With 22 total bit planes to calculate, the design involves 66 individual data grouping and variable FIFO blocks. Although none consume a significant amount of FPGA resources individually, 66 blocks do. The entire design required 160 percent of the resources in a Virtex 2000E, and would not fit in the target system. However, by removing the lower bit planes, less FPGA resources are needed, and the architecture can easily be adjusted to fit the FPGA being used. Depending on the size of the final bitstream required, the FPGA size used in the SPIHT phase can be varied to handle the number of intermediate bitstreams generated.

Removing lower bit planes is possible since the final bitstream transmits data from the highest bit plane to the lowest. In our design the lower 9-bit planes

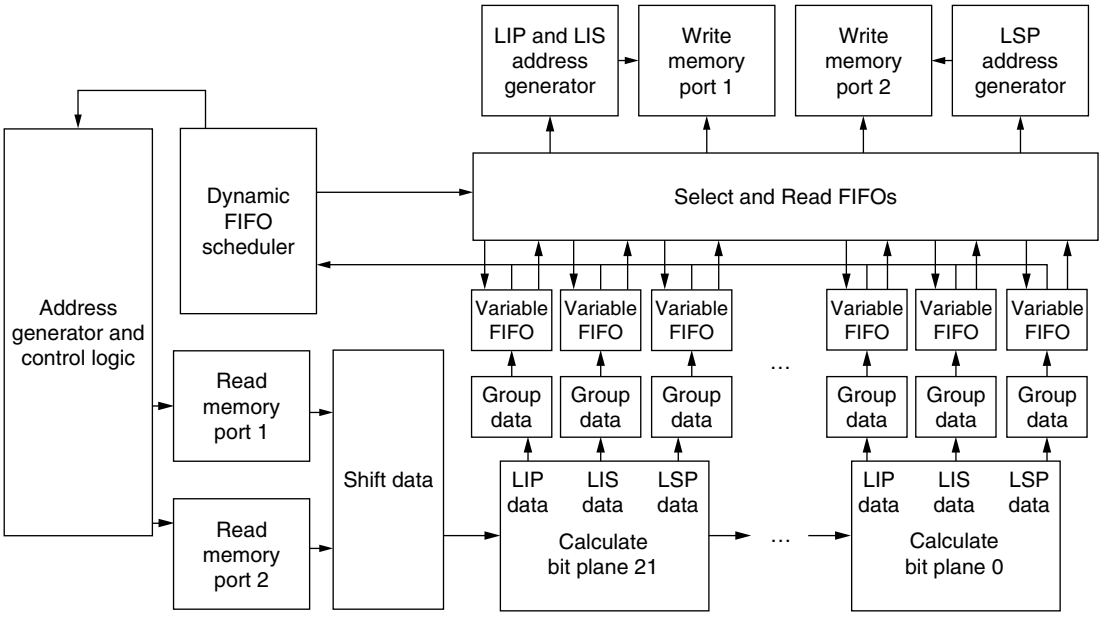

FIGURE 27.18 ■ A block diagram of the SPIHT coding phase.

were eliminated. Yet, without these lower planes, bitrates of up to 6 bpp can still be achieved. We found the constraint to be acceptable because we are interested in high compression ratios using low bitrates, and 6 bpp is practically a lossless signal. Since SPIHT is optimized for lower bitrates, the ability to calculate higher bitrates was not considered necessary. Alternatively, the use of a larger FPGA would alleviate the size constraint.

27.5 DESIGN RESULTS

The system was designed using VHDL with models provided by Annapolis Micro Systems to access the PCI bus and memory ports. Simulations for debugging purposes were carried out with ModelSim EE 5.4e from Mentor Graphics. Synplify 6.2 from Synplicity was used to compile the VHDL code and generate a netlist. The Xilinx Foundation Series 3.1i tool set was used to place and route the design. Lastly, the `peutil.exe` utility from Annapolis Micro Systems generated the FPGA configuration streams.

Table 27.3 shows the speed and runtime specifications of the final architecture. All performance numbers are measured results from the actual hardware implementation. Each phase computes on separate memory blocks, which can operate at different clock rates. The design can process any square image where the dimensions are a power of 2: 16×16, 32×32, up to 1024×1024.

Since the WildStar board is connected to the host computer by a relatively slow PCI bus, the throughput of the entire system we built is constrained by the throughput of the PCI bus. However, since the study is on how image compression routines could be implemented on a satellite, such a system would be designed differently, and would not contain a reconfigurable board connected to some host platform though a PCI bus. Instead, the image compression routines would be inserted directly into the data path and the data transfer times would not be the bottleneck of the system. For this reason we analyzed the throughput of just the SPIHT compression engine and analyzed how quickly the FPGAs can process the images.

The throughput of the system was constrained by the discrete wavelet transform at 100 MPixels/sec. One method to increase this rate is to compute more rows in parallel. If the available memory ports accessed 128 bits of data instead of the 64 bits with our WildStar board, the number of clock cycles per pixel could be reduced by half and the throughput could double.

TABLE 27.3 ■ Performance numbers

Phase	Clock cycles per 512×512 image	Clock cycles per pixel	Clock rate	Throughput	FPGA area (%)
Wavelet	182465	3/4	75 MHz	100 MPixels/sec	62
Magnitude	131132	1/2	73 MHz	146 MPixels/sec	34
SPIHT	65793	1/4	56 MHz	224 MPixels/sec	98

Assuming the original image consists of 8 bpp, images are processed at a rate of 800 Mbits/sec.

The entire throughput of the architecture is less than one clock cycle for every pixel, which is lower than parallel versions of the DWT. Parallel versions of the DWT used complex scheduling to compute multiple wavelet levels simultaneously, which left limited resources to process multiple rows at a time. Given more resources though, they would obtain higher data rates than our architecture by processing multiple rows simultaneously. In the future, a DWT architecture other than the one we implemented could be selected for additional speed improvements.

We compared our results to the original software version of SPIHT provided on the SPIHT web site [15]. The comparison was made without arithmetic coding since our hardware implementation does not perform any arithmetic coding on the final bitstream. Additionally, in our testing on sample NASA images, arithmetic coding added little to overall compression rates and thus was dropped [11]. An IBM RS/6000 Model 270 workstation was used for the comparison, and we used a combination of standard image compression benchmark images and satellite images from NASA's web site. The software version of SPIHT compressed a 512×512 image in 1.101 seconds on average without including disk access. The wavelet phase, which constrains the hardware implementation, computes in 2.48 milliseconds, yielding a speedup of 443 times for the SPIHT engine. In addition, by creating a parallel implementation of the wavelet phase, further improvements to the runtimes of the SPIHT engine are possible.

While this is the speedup we will obtain if the data transfer times are not a factor, the design may be used to speed up SPIHT on a general-purpose processor. On such a system the time to read and write data must be included as well. Our WildStar board is connected to the host processor over a PCI bus, which writes images in 13 milliseconds and reads the final datastream in 20.75 milliseconds. Even with the data transfer delay, the total speedup still yields an improvement of 31.4 times.

Both the magnitude and SPIHT phases yield higher throughputs than the wavelet phase, even though they operate at lower clock rates. The reason for the higher throughputs is that both of these phases need fewer clock cycles per pixel to compute an image. The magnitude phase takes half a clock cycle per pixel and the SPIHT phase requires just a quarter. The fact that the SPIHT phase computes in less than one clock cycle per pixel, let alone a quarter, is a striking result considering that the original SPIHT algorithm is very sequential in nature and had to consider each pixel in an image multiple times per bit plane.

27.6 SUMMARY AND FUTURE WORK

In this chapter we demonstrated a viable image compression routine on a reconfigurable platform. We showed how by analyzing the range of data processed by each section of the algorithm, it is advantageous to create optimized memory

structures as with our variable fixed-point work. Doing so minimizes memory usages and yields efficient data transfers. Here each bit transferred between memory and the processor board directly impacted the final results. In addition, our Fixed Order SPIHT modifications illustrate how by making slight adjustments to an existing algorithm, it is possible to dramatically increase the performance in a custom hardware implementation and simultaneously yield essentially identical results. With Fixed Order SPIHT the throughput of the system increased by over an order of magnitude while still matching the original algorithm's PSNR curve.

This SPIHT work was part of a development effort funded by NASA.

References

[1] V. R. Algazi, R. R. Estes. Analysis-based coding of image transform and subband coefficients. *Applications of Digital Image Processing XVIII, SPIE Proceedings* 2564, 1995.

[2] Annapolis Microsystems. *WildStar Reference Manual*, Annapolis Microsystems, 2000.

[3] A. Benkrid, D. Crookes, K. Benkrid. Design and implementation of generic 2D biorthogonal discrete wavelet transform on an FPGA. *IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2001.

[4] M. Carraeu. Hubble Servicing Mission: Hubble is fitted with a new "eye." *http://www.chron.com/content/interactive/space/missions/sts-103/hubble/archive/931207.html*, December 7, 1993.

[5] C. M. Chakrabarti, M. Vishwanath. Efficient realization of the discrete and continuous wavelet transforms: From single chip implementations to mappings in SIMD array computers. *IEEE Transactions on Signal Processing* 43, March 1995.

[6] C. M. Chakrabarti, M. Vishwanath, R. M. Owens. Architectures for wavelet transforms: A survey. *Journal of VLSI Signal Processing* 14, 1996.

[7] T. Cormen, C. Leiserson, R. Rivest. *Introduction to Algorithms*, MIT Press, 1997.

[8] T. W. Fry. *Hyper Spectral Image Compression on Reconfigurable Platforms*, Master's thesis, University of Washington, Seattle, 2001.

[9] R. C. Gonzalez, R. E. Woods. *Digital Image Processing*, Addison-Wesley, 1993.

[10] A. Graps. An introduction to wavelets. *IEEE Computational Science and Engineering* 2(2), 1995.

[11] T. Owen, S. Hauck. *Arithmetic Compression on SPITH Encoded Images*, Technical report UWEETR-2002–2007, Department of Electrical Engineering, University of Washington, Seattle, 2002.

[12] K. K. Parhi, T. Nishitani. VLSI architectures for discrete wavelet transforms. *IEEE Transactions on VLSI Systems* 1(2), 1993.

[13] J. Ritter, P. Molitor. A pipelined architecture for partitioned DWT based lossy image compression using FPGAs. *ACM/SIGDA Ninth International Symposium on Field-Programmable Gate Arrays*, February 2001.

[14] A. Said, W. A. Pearlman. A new fast and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology* 6, June 1996.

[15] A. Said, W. A. Pearlman. SPIHT image compression: Properties of the method. *http://www.cipr.rpi.edu/research/SPIHT/spiht1.html*.

[16] H. Sava, M. Fleury, A. C. Downton, A. Clark. Parallel pipeline implementations of wavelet transforms. *IEEE Proceedings Part 1 (Vision, Image and Signal Processing)* 144(6), 1997.

[17] J. M. Shapiro. Embedded image coding using zero trees of wavelet coefficients. *IEEE Transactions on Signal Processing* 41(12), 1993.
[18] W. Sweldens. The Lifting Scheme: A new philosophy in biorthogonal wavelet constructions. *Wavelet Applications in Signal and Image Processing* 3, 1995.
[19] NASA. TERRA: The EOS flagship. The EOS Data and Information System (EOSDIS). *http://terra.nasa.gov/Brochure/Sect_5-1.html*.
[20] C. Valens. A really friendly guide to wavelets. *http://perso.wanadoo.fr/polyvalens/clemens/wavelets/wavelets.html*.
[21] M. Vishwanath, R. M. Owens, M. J. Irwin. VLSI architectures for the discrete wavelet transform. *IEEE Transactions on Circuits and Systems, Part II*, May 1995.
[22] Xilinx, Inc. *The Programmable Logic Data Book*, Xilinx, Inc., 2000.
[23] Xilinx, Inc. *Serial Distributed Arithmetic FIR Filter*, Xilinx, Inc., 1998.