# Hyperspectral Image Compression on Reconfigurable Platforms

Thomas W. Fry
Department of Electrical Engineering
University of Washington
Seattle, WA 98195
tom@tomfry.com

Scott Hauck
Department of Electrical Engineering
University of Washington
Seattle, WA 98195
hauck@ee.washington.edu

## ABSTRACT

In this paper we present an implementation of the image compression routine SPIHT in reconfigurable logic. A discussion on why adaptive logic is required, as opposed to an ASIC, is provided along with background material on the image compression algorithm. We analyzed several Discrete Wavelet Transform architectures and selected the folded DWT design. In addition we provide a study on what storage elements are required for each wavelet coefficient.

The paper uses a modification to the original SPIHT algorithm needed to parallelize the computation. The architecture of the SPIHT engine is based upon Fixed-Order SPIHT, developed specifically for use within adaptive hardware. For an N x N image Fixed-Order SPIHT may be calculated in $N^2/4$ cycles. Square images which are powers of 2 up to 1024 x 1024 are supported by the architecture. Our system was developed on an Annapolis Microsystems WildStar board populated with Xilinx Virtex-E parts.

## 1. Introduction

Satellites deployed by NASA currently do not make use of lossy image compression techniques during transmission. There have been a few driving reasons behind NASA's decision to transmit raw data. First, the downlink channels have provided enough bandwidth to handle all of the data a satellite's sensors collected in real time. Second, there has been a lack of viable platforms with which a satellite could process data. Lastly, transmitting raw data reduces the risk of corrupting the data-stream.

As NASA deploys satellites with more sensors, capturing an ever-larger number of spectral bands, the volume of data being collected is beginning to outstrip a satellite's ability to transmit it back to Earth. NASA's most recent satellite Terra 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 all of these sensors at a rate of 150Mbps [15]. As the number of sensors on a satellite grows and thus the transmission rates increase, they are providing a driving force for NASA to study methods of compressing images prior to down linking.

Current technologies have been unable to provide NASA with a viable platform to process data in space. Software solutions suffer from performance limitations and power requirements. At the same time traditional hardware platforms lack the required flexibility needed for post-launch modifications. After launch they cannot be modified to use newer compression schemes or even implement bug fixes. In the past, a modification to fixed systems in satellites has proven to be very expensive. The correction to the Hubble telescope's flawed 94-inch-wide primary mirror approached $50 million [3].

By implementing an image compression kernel in a reconfigurable system, it is possible to overcome these shortcomings. Since 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 rates, while consuming less power than a traditional software implementation.

Our work is part of a NASA-sponsored investigation into the design and implementation of a space-bound FPGA-based Hyperspectral Image Compression algorithm. We have selected the Set Partitioning in Hierarchical Trees (SPIHT) compression routine and optimized the algorithm for implementation in hardware. This thesis

describes our work towards this effort and provides a description of our results.

## 2 Description of the Algorithm

SPIHT is a wavelet-based image compression coder. It first converts the image into its wavelet transform and then transmits information about the wavelet coefficients. The decoder uses the received signal to reconstruct the wavelet and performs an inverse transform to recover the image. We selected SPIHT because it displays exceptional characteristics over several properties all at once [11]. They include:

- Good image quality with a high PSNR
- Fast coding and decoding
- A fully progressive bit-stream
- Can be used for lossless compression
- May be combined with error protection
- Ability to code for exact bit rate or PSNR

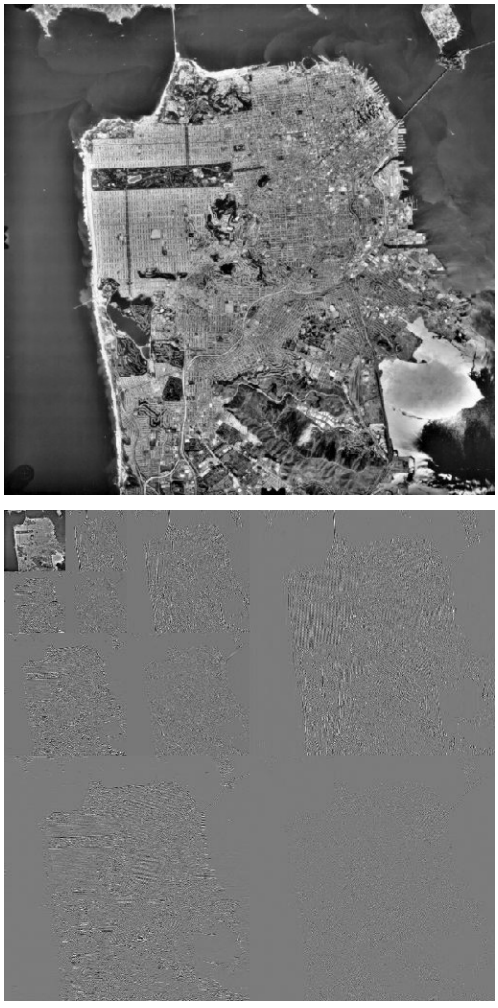

**Figure 1: A Three-level DWT**

The Discrete Wavelet Transform (DWT) runs a high and low-pass subband over the signal in one dimension. The result is a new image comprising of a high and low-pass subband. This procedure is then repeated in the other dimension yielding four subbands, three high-pass components and one low-pass component. The next wavelet level is calculated by repeating the horizontal and vertical transformations on the low-pass subband from the previous level. The DWT repeats this procedure for however many levels are required. Each procedure is fully reversible so that each wavelet level is later recovered to obtain the low-pass result used in the previous wavelet level. Figure 1 displays a satellite image of San Francisco and its corresponding three level DWT.

SPIHT is a method of coding and decoding the wavelet transform of an image. 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. The entire wavelet does not need to be transmitted in order to recover the image. Instead, as the decoder receives more information about the wavelet, the inverse-transformation will yield a better quality reconstruction of the original image. SPIHT generates excellent image quality and performance due to several properties of the coding algorithm. They are partial ordering by coefficient value, taking advantage of the redundancies between different wavelet scales and transmitting data in bit plane order [10].

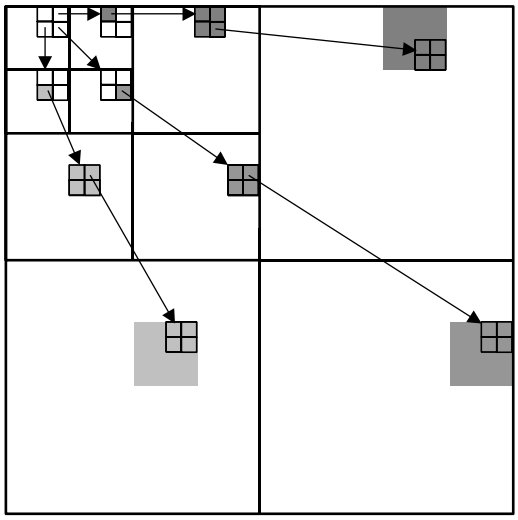

**Figure 2: Spatial-orientation trees**

Following a wavelet transformation, SPIHT divides the wavelet into *Spatial Orientation Trees*. Each node in the 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 x 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. Also, the upper left pixel of each 2 x 2 block at the root of the tree has no children since there only 3 subbands at each scale and not four. Figure 2 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 descendants are above a threshold. If the statement proves false, then all of its 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. By proceeding 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.

In addition to transmitting wavelet coefficients in a bit plane ordering, the SPIHT algorithm develops an individual order to transmit information within each bit plane. The ordering is implicitly created from the threshold information discussed above and by a set of rules which both the encoder and decoder agree upon. Thus each image will transmit wavelet coefficients in an entirely different order. Slightly better Peak Signal to Noise Ratios (PSNR) are achieved by using this dynamic ordering of the wavelet coefficients. The trade-off for the improvement are increased run-times for both the encoder and decoder since the order must be calculated.

# 3. Prior Work

## 3.1 Wavelet Architectures

As wavelets have gained popularity over the past several years there has been growing interest in implementing the discrete wavelet transform in hardware. Much of the work on DWTs involves parallel platforms to save both memory access and computations [4][8][12]. Here we will provide a review of four individual DWT architectures and their performance where available.

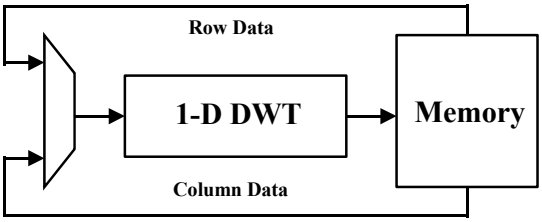


**Figure 3: Illustration of a folded architecture**

The one-dimensional DWT entails demanding computations, which involve significant hardware resources. Most two-dimensional DWT architectures have implemented folding to reuse logic for each dimension, since the horizontal and vertical passes use identical FIR filters [5]. Figure 3 illustrates how a 1-D DWT is used to realize a 2-D DWT.

Such an architecture suffers from high memory bandwidth. For an N x N image there are at least $2N^2$ read and write cycles for the first wavelet level. Additional levels require re-reading previously computed coefficients.

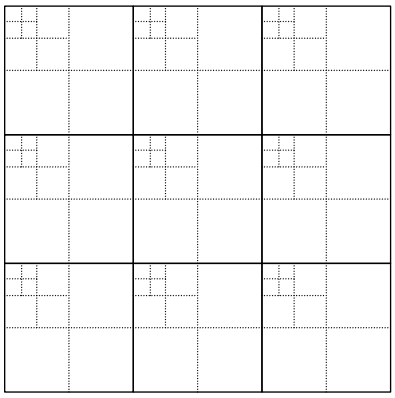

**Figure 4: The Partitioned DWT**

In order to address these superfluous memory accesses the Partitioned DWT was designed. The Partitioned DWT partitions the image into smaller blocks and computes several scales of the DWT at once for each block [9]. In addition, the algorithm makes use of wavelet lifting to reduce the computational complexity of the DWT [14]. By partitioning an image into smaller blocks, the amount of on-chip memory storage required is significantly reduced since only the coefficients in the block need to be stored. The approach is similar to the Recursive Pyramid Algorithm except that it computes over sections of the image at a time instead of the entire image at once. Figure 4 from Ritter et al. [9] illustrates how the partitioned wavelet is constructed.

Nevertheless the partitioned approach suffers blocking artifacts along the partition boundaries if the boundaries are treated with reflection[1]. Thus pixels from neighboring partitions are required to smooth out these boundaries. The number of wavelet levels determines how many pixels beyond a sub-image's boundary are needed. Higher wavelet levels represent data from a greater region of the image. To compensate for the partition boundaries the algorithm processes sub-image rows at a time to eliminate multiple reads in the horizontal direction. Overall data throughputs of up to 152Mbytes have been achieved with the Partitioned DWT.

Another method to reduce memory accesses is the Recursive Pyramid Algorithm (RPA) [16]. RPA takes advantage of the fact that the various wavelet levels run at different clock rates. Each wavelet level requires ¼ the amount of time as the previous level. Thus it is possible to store previously computed coefficients on-chip and intermix the next level's computations with the current calculations. A careful analysis of the runtime yields $(4*N^2)/3$ computations for an image. However the algorithm has significant on chip memory requirements and requires a thorough scheduling process to interleave the various wavelet levels.

The last unique architecture to discuss is the Generic 2-D Biorthogonal DWT shown in Benkrid et al. [2]. Unlike previous design methodologies, the Generic 2-D Biorthogonal DWT does not require filter folding or large on chip memories as the Recursive Pyramid design. Nor does it involve partitioning an image into sub-images. Instead, the architecture proposed creates separate structures to calculate each wavelet level as data is presented to it, as shown in Figure 5. The design sequentially reads in the image and computes the four DWT subbands. As the $LL_1$ subband becomes available, the coefficients are passed off to the next stage, which will calculate the next coarser level subbands and so on.

For larger images that require several individual wavelet scales, the Generic 2-D Biorthogonal DWT architecture consumes a tremendous amount of on-chip resources. With SPIHT, a 1024 by 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 processes data at different rates, a separate clock signal is also needed for each scale. The advantage of the architecture is much lower on-chip memory requirements and full utilization of the memory's bandwidth since each pixel is only read and written once.

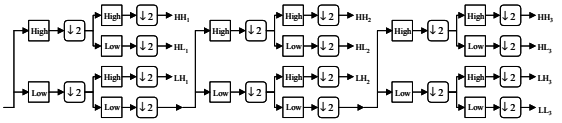


**Figure 5: Generic 2-D Biorthogonal DWT**

## 3.2 SPIHT Architectures

To date the literature contains very little on hardware implementations of SPIHT since the algorithm was developed so recently. Singh et al. [13] briefly describes a direct implementation of the SPIHT software algorithm. The paper is a brief on work done and provides a high level overview of the architecture.

Their design calls for one processing phase to calculate the image's wavelet transformation and another for the SPIHT coding. The SPIHT coding is performed using Content Addressable Memories to keep track of the order in which information about the wavelet is sent for each image.

The algorithm sequentially steps through the wavelet coefficients multiple times in the same order as the original software program. No optimizations or modifications were made to the algorithm to take into account that the design would compute on a hardware platform as opposed to a software platform. The design was simulated over 8 by 8 sized images for functional verification. Since the design has only been simulated no performance numbers were given.

# 4 Design Considerations and Modifications

In order to fully take advantage of the high performance a custom hardware implementation of SPIHT can yield, the software specifications must be examined and adjusted where they either perform poorly in hardware or do not make the most of the resources available. Here we discuss both memory storage considerations and optimizations to the original SPIHT algorithm for use in hardware.

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

## 4.1 Variable Fixed-Point

The discrete wavelet transform produces real numbers as wavelet coefficients. Traditionally FPGAs have not employed the use of floating-point numbers for several reasons. Some of these reasons are that floating-point numbers:

- Require variable shifts based on the exponential description and variable shifters in FPGAs perform poorly.
- Consume enormous hardware resources on a limited resource FPGA.
- Are unnecessary for a known data set.

At each wavelet level of the DWT, coefficients have a fixed range. Therefore we opted for a fixed-point numerical representation. A fixed-point number 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 is unbounded, meaning that the range of possible numbers increases with each additional wavelet level.

An analysis of the coefficients of each filter bank shows that a 2-D 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 all of the previous level's coefficients were both the maximum found at that level and of the correct sign. As a result, the coefficients at various wavelet levels require a variable number of bits above the decimal point to cover their possible ranges, as shown in Table 1.

**Table 1: Fixed-Point Magnitude Calculations**

| Wavelet Level | Factor | Maximum Magnitude | Maximum Bits | Maximum Bits from Data |
|---|---|---|---|---|
| Input | 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 |

Figure 6 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 and the final result for a 1 byte per pixel image. The next column shows the maximum number of bits (with a sign bit) that are necessary to represent the numeric range at each wavelet level. The maximum number of bits we found by evaluating the actual range observed over many sample images is displayed in the last column. These values determine what position the most significant bit must stand for.

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. Yet the lowest wavelet levels will never encounter numbers in that range. As a result, several bits at these levels would not be employed and therefore wasted.

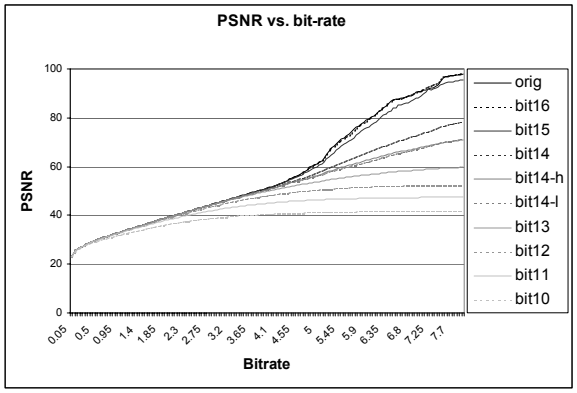


**Figure 6: PSNR vs. bit-rate for various coefficient sizes**

To fully utilize all of the bits for each wavelet coefficient, we introduce the concept of *Variable Fixed-Point* representation. With Variable Fixed-Point we assign a fixed-point numerical representation for each wavelet level optimized for the expected data. In addition, each representation differs from one another, meaning we employ a different fixed-point scheme for each wavelet level. Doing so allows 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 is found for each wavelet level, the number of precision bits to accurately represent the wavelet coefficients needs to be determined. Our goal is to provide enough bits to fully recover the image and at the same time use only as many are necessary to do so. Figure 6 displays the average Peak Signal to Noise ratios for several recovered images from SPIHT using a range of bit widths for each coefficient.

An assignment of 16 bits per coefficient most accurately matches the full precision floating-point

coefficients used in software, up through perfect reconstruction. Previous wavelet designs have focused on bit rates less than 4 bpp. Their studies found that fewer pixels are necessary for SPIHT [2]. Instead we elected a numerical representation which retains an equivalent amount of information as a floating-point number. By doing so, it is possible to perfectly reconstruct an image given a high enough bit rate. Table 2 provides the number of integer and decimal bits[2] allocated for each wavelet level. The number of integer bits also includes one 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.

**Table 2: Variable Fixed-Point Representation**

| Wavelet Level | Integer Bits | Decimal Bits |
|---|---|---|
| image | 10 | 6 |
| 0 | 11 | 5 |
| 1 | 12 | 4 |
| 2 | 13 | 3 |
| 3 | 14 | 2 |
| 4 | 15 | 1 |
| 5 | 16 | 0 |
| 6 | 17 | -1 |

## 4.2 Fixed Order SPIHT

As discussed within Section 3 the SPIHT algorithm computes a dynamic ordering of the wavelet coefficients as it progresses. Such an ordering will yield better image quality for bit-streams which end within the middle of a bit-plane. The drawback of this ordering is that every image will have a unique list order determined by the image's wavelet coefficient values.

Yet, the data that a block of coefficients contributes to the final SPIHT bit-stream is fully determined by the following localized information.

- The 2x2 block of coefficients
- Their immediate children
- The maximum value within the sub-tree.

Thus, every block of coefficients may be calculated independently and in parallel of one another. However, the order that a block's data will be inserted into the bit-stream is not known since this order is dependent upon the image's unique ordering. Once the order is determined it is possible to produce a valid SPIHT bit-stream from the above information.

However the algorithm employed to calculate the SPIHT ordering of coefficients is sequential in nature. The computation steps over the coefficients of the image a couple of times within each bit-plane and dynamically inserts and removes coefficients from multiple lists. Such an algorithm is not parallizable in hardware and significantly limits the throughput of any implementation.

We propose a modification to the original SPIHT algorithm called *Fixed Order SPIHT*. In Fixed Order SPIHT the order in which blocks of coefficients are transmitted are fixed before hand. Doing so removes the need to calculate the ordering of coefficients within each bit-plane and allows us to create a fully parallel version of the original SPIHT algorithm. Such a modification increases the through put of a hardware encoder by greater than an order of magnitude, at the cost of a slightly lower PSNR within each bit-plane (approximately 0.1 – 0.2 dB).

The advantage of such a method is 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 regular SPIHT algorithm. Since the length of each bit-stream is fairly short within the transmitted data stream, the PSNR curve of Fixed Order SPIHT very closely matches that of the original algorithm. For a more complete discussion on Fixed Order SPIHT refer to Fry et al. [7].

# 5 Architecture

## 5.1 Target Platform

Our target platform is the WildStar FPGA processor board developed by Annapolis Micro Systems [1]. The board, shown in Figure 7, consists of three Xilinx Virtex 2000E FPGAs: PE0, PE1 and PE2. It operates at rates up to 133MHz. 48 MBytes of memory is available 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 is passed between the chips in just a few clock cycles.

## 5.2 Design Overview

Our architecture consists of three phases: Wavelet Transform, Maximum Magnitude Calculation and

[2] Integer bits refer to bits above the decimal point. Decimal bits refer to bits following the decimal point.

Fixed Order SPIHT Coding. Each phase is implemented in one of the three Virtex chips. By instantiating each phase on a separate chip, separate images can be operated upon in parallel. Data are transferred from one phase to the next through the shared memories.

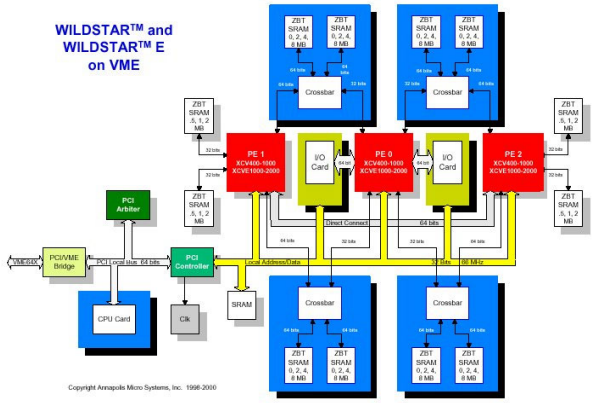


**Figure 7: Annapolis Micro Systems WildStar board block diagram**

Once processing in a phase is complete, the crossbar mode is switched and the data calculated is 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 8 illustrates the architecture of the system.

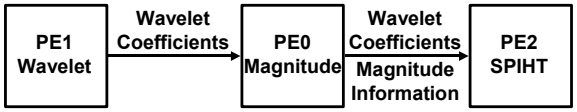


**Figure 8: Overview of the architecture**

## 5.3 DWT Phase

We selected a form of the folding architecture to calculate the DWT. Previous parallel versions of the DWT saved some memory bandwidth. However, additional resources and a more complex scheduling algorithm become necessary. In addition the savings becomes minimal since each higher wavelet level is ¼ the size of the previous wavelet level. In a seven level DWT, the highest 4 levels compute in just 2% of the time it takes to compute the first level.

For this reason we designed a folded architecture which processes one dimension of a single wavelet level. 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 form.

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. Given 16 bit coefficients and a 64-bit wide memory port, four rows is the maximum that can be transferred in a clock cycles. Figure 9 illustrates the architecture of the discrete wavelet transform phase.

Since every pixel is read and written once and the design processes four rows at a time, for an N x N size image both dimensions in the lowest wavelet level will compute in N/4 clock cycles. Similarly, the next wavelet level will process the image in ¼ the number of clock cycles as the previous level. With an infinite number of wavelet levels the image will process in:

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

Thus the runtime of the DWT engine is bounded by ¾$^{th}$ a clock cycle per pixel in the image. Many of the parallel architectures designed to process multiple wavelet levels simultaneously run in more than one clock cycle per image. Because of the additional resources required by a parallel implementation, computing multiple rows at once becomes impractical. Given more resources, the parallel architectures discussed above could process multiple rows at once and yield runtimes lower than ¾$^{th}$ a clock cycle per pixel. However, our FPGAs do not have such extensive resources.

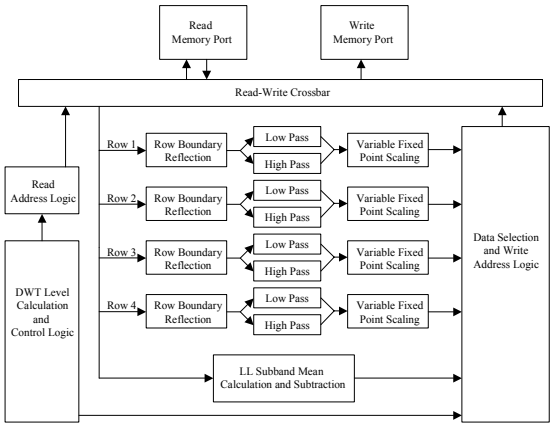


**Figure 9: DWT Architecture**

## 5.4 Maximum Magnitude Phase

The maximum magnitude phase calculates and rearranges the following information for the SPIHT phase.

- The maximum magnitude of each of the 4 child trees.
- The maximum magnitude of the current tree.
- Threshold and Sign data of each of the 16 child coefficients.
- Re-order the wavelet coefficients into a Morton Scan ordering.

To calculate the maximum magnitude of all coefficients below a node in the spatial orientation trees, the image must be scanned in a depth-first search order [6]. By scanning the trees of the image in a depth-first search order, whenever a new coefficient is read and being 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. Since PE0 (the 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.

## 5.5 SPIHT Phase

The final SPIHT Coding phase essentially computes the parallel algorithm. 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 the same 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 which the variable FIFO components receive varies in size, ranging from zero bits to thirty-seven bits. The variable FIFOs are used to arrange the block data into regular sized 32-bit sized words for memory accesses. Care is also taken to stall the algorithm if anyone of the variable FIFOs becomes too full. The block diagram for the SPIHT coding phase is given in Figure 10.

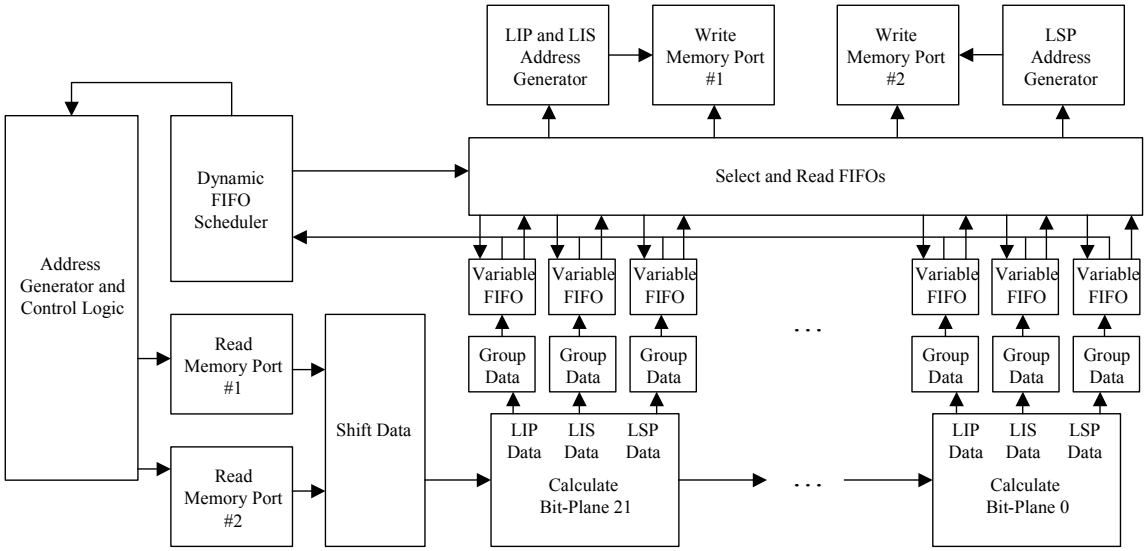


**Figure 10: SPIHT Coding Phase Block Diagram**

# 6 Design Results

Our system was designed using VHDL with models provided from Annapolis Micro Systems to access the PCI bus and memory ports. Simulations for debugging purposes were done with ModelSim EE 5.4e from Mentor Graphics. Synplify 6.2 from Synplicity was used to compile the VHDL code and

generate a net list. The Xilinx Foundation Series 3.1i tool set was used to both place and route the design. Lastly the peutil.exe utility from Annapolis Micro Systems generated the FPGA configuration streams.

Table 3 shows the speed and runtime specifications of our architecture. All performance numbers are measured results from the actual hardware implementations. 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 by 16, 32 by 32 up to 1024 by 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, our 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. Rather 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.

**Table 3: Performance Numbers**

| Phase | Clock Cycles per 512x512 image | Clock Cycles per Pixel | Clock Rate | Throughput | FPGA Area |
|---|---|---|---|---|---|
| Wavelet | 182465 | 3/4 | 75 MHz | 100 MBytes/sec | 62% |
| Magnitude | 131132 | 1/2 | 73 MHz | 146 MBytes/sec | 34% |
| SPIHT | 65793 | 1/4 | 56 MHz | 224 MBytes/sec | 98% |

The throughput of the system is constrained by the discrete wavelet transform at 100 MBytes/second. One method to increase its 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. Assuming the original image consists of 8 bpp, images are processed at a rate of 800Mbits/sec.

In addition 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 by processing multiple rows simultaneously than our architecture could. In the future another DWT architecture than the one we implemented could be selected for further speed improvements.

We compared our results to the original software version of SPIHT provided on the SPIHT website [11]. The comparison was made without arithmetic coding since our hardware implementation currently does not perform any arithmetic coding on the final bit-stream. An Ultra 5 SPARC workstation was used for the comparison and we used a combination of satellite images from NASA's website and standard image compression benchmark images. The software version of SPIHT compressed a 512 x 512 image in 1.14 seconds on average. The wavelet phase, which constrains the hardware implementation, computes in 2.48 milliseconds, yielding a speedup of **457** 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 data stream in 20.75 milliseconds. With the data transfer delay, the total speedup still yields an improvement of 31.4 times.

# 7 Conclusions

In this paper we have 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 usage and yields the utmost usefulness of transferred data. (i.e. each bit transferred between memory and the processor board directly impacts the final result.) In addition

our Fixed Order SPIHT work illustrates how by making slight adjustments to an existing algorithm, it is possible to dramatically increase the performance of a custom hardware implementation and simultaneously yield essentially identical results. With Fixed Order SPIHT the throughput of the system increases by roughly an order of magnitude while still matching the original algorithm's PSNR curve.

Our SPIHT work is part of an ongoing development effort funded by NASA. Future work will to address how lossy image compression will affect downstream processing. The level of lossy image compression that is tolerable before later processing begins to yield false results needs to be analyzed and dealt with. Lastly improvements to SPIHT and the consequences to a hardware implementation will be studied. Modifications to Fixed Order SPIHT including adding error protection to the bit-stream and region of interest coding will be considered.

# 8 References


[1] Annapolis Microsystems. *WildStar Reference Manual*, Maryland: Annapolis Microsystems, 2000.

[2] A. Benkrid, D. Crookes, K. Benkrid, "Design and Implementation of Generic 2-D Biorthogonal Discrete Wavelet Transform on and FPGA," *IEEE Symposium on Field Programmable Custom Computing Machines*, pp 1 – 9, April 2001.

[3] M. Carraeu, "Hubble is fitted with a new 'Eye'", Houston Chronicle, December 7, 1993.

[4] 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*, Vol. 43, pp 759 – 771, March 1995.

[5] C. M. Chakrabarti, M. Vishwanath, Owens R.M, "Architectures for Wavelet Transforms: A Survey," *Journal of VLSI Signal Processing*, Vol. 14, pp 171-192, 1996.

[6] T. Cormen, C. Leiserson, R. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusetts, 1997.

[7] T. W. Fry, *Hyper Spectral Image Compression on Reconfigurable Platforms*, Master Thesis, University of Washington, Seattle, Washington, 2001.

[8] K. K. Parhi, T. Nishitani, "VLSI Architectures for Discrete Wavelet Transforms," *IEEE Transactions on VLSI Systems*, pp 191 – 201, June 1993.

[9] J. Ritter, P. Molitor, "A Pipelined Architecture for Partitioned DWT Based Lossy Image Compression using FPGA's," *ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays*, pp 201 – 206, February 2001.

[10] 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*, Vol. 6, pp 243 - 250, June 1996.

[11] A. Said, W. A. Pearlman, "SPIHT Image Compression: Properties of the Method", http://www.cipr.rpi.edu/research/SPIHT/spiht1.html

[12] H. Sava, M. Fleury, A. C. Downton, Clark A, "Parallel pipeline implementations of wavelet transforms." *IEEE Proceedings Part 1 (Vision, Image and Signal Processing)*, Vol. 144(6), pp 355 – 359, December 1997.

[13] J. Singh, A. Antoniou, D. J. Shpak, "Hardware Implementation of a Wavelet based Image Compression Coder," *IEEE Symposium on Advances in Digital Filtering and Signal Processing*, pp 169 – 173, 1998.

[14] W. Sweldens, "The Lifting Scheme: A New Philosophy in Biorthogonal Wavelet Constructions," *Wavelet Applications in Signal and Image Processing*, Vol. 3, pp 68 – 79, 1995.

[15] "The EOS Data and Information System (EOSDIS)", http://terra.nasa.gov/Brochure/Sect_5-1.html

[16] M. Vishwanath, R. M. Owens, M. J. Irwin, "VLSI Architectures for the Discrete Wavelet Transform," *IEEE Transactions on Circuits and Systems, Part II*, pp 305-316, May 1995.