Hyperspectral Image Compression on Reconfigurable Platforms

Thomas W. Fry

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in Electrical Engineering

University of Washington

2001

Program Authorized to Offer Degree: Electrical Engineering

University of Washington

Abstract

Hyperspectral Image Compression on Reconfigurable Platforms

Thomas W. Fry

Chair of the Supervisory Committee:
Associate Professor, Scott Hauck
Electrical Engineering

NASA's satellites currently do not make use of image compression techniques during data transmission to earth because of limitations in the available platforms. Yet as satellites are built with an ever-larger number of sensors, the amount of data captured by a satellite is beginning to overwhelm the bandwidth capabilities of the transmission channels. Current software implementations are unable to meet the necessary computational and power requirements for use in a satellite. At the same time hardware platforms lack the required flexibility needed for post-launch modifications.

With the advent of Field Programmable Gate Arrays (FPGAs) and Adaptive Computing technologies it is now possible to construct a system, which compresses the data stream before down linking. By developing image compression routines on a reconfigurable platform, it is possible to obtain the computational performance required to compress a satellite's data in real time and at the same time retain the ability to modify the system post-launch. Our work is part of a NASA-sponsored study on the design and implementation of FPGA-based Hyperspectral Image Compression algorithms for use in space.

Table of Contents

Page

List of Figures

Page

List of Tables

Page

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 [19]. 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 [4].

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 Background

2.1 FPGAs

The most common type of computing system is the general-purpose processor. Under this model, the hardware of the system is limited to merely a few basic tasks. By combining and building off of these operations, a general-purpose computer can perform a much larger number of operations than it was originally designed to handle. Which is why the general-purpose computer is so flexible. However this flexibility comes with a price. For any specific application, the general-purpose processor will perform poorly when compared to a custom hardware implementation.

Traditionally computations that required the high performance of a custom hardware implementation needed the development and fabrication of an Application-Specific Integrated Circuit (ASIC). Development of an ASIC requires several steps. The circuit must be designed and tested. Once the circuit is designed it must be fabricated. Fabrication involves creating wafer masks for that specific design, fabricating the chips, packaging and finally testing. A modification to a design post-masking requires whole new wafer masks to be prepared. All of these factors contribute to making ASIC designs both expensive for low volume runs and intolerant to design errors or modifications once the fabrication process is started.

With the advent of Field Programmable Gate Arrays (FPGAs) and Reconfigurable Computing, designers may now develop custom hardware solutions without a separate fabrication run for each design. FPGAs are, as their name implies, an array of logic gates, which can be programmed to perform a variety of tasks. They consist of programmable logic structures distributed throughout the chip in one of four methods: Symmetrical Array, Sea-of-Gates, Row Based and Hierarchical PLD [10]. A routing interconnect is used to connect the logic structures. Like the array of logic gates, the routing interconnect is fully programmable.

By reprogramming the logic gates and the routing interconnect it is possible to configure the chip to perform any arbitrary computation. By utilizing their programmable nature,

FPGAs offer a low cost, flexible solution over traditional ASICs. Since a single FPGA design may be used for many tasks, it can be fabricated in higher volumes, lowering fabrication costs. Also, their ability to be reprogrammed allows for easy design modifications and bug fixes without the need to construct a new hardware system. FPGAs may be reprogrammed within milliseconds for no cost other than the designer's time, while ASICs require a completely new fabrication run lasting a month or two and costing hundreds of thousands of dollars.

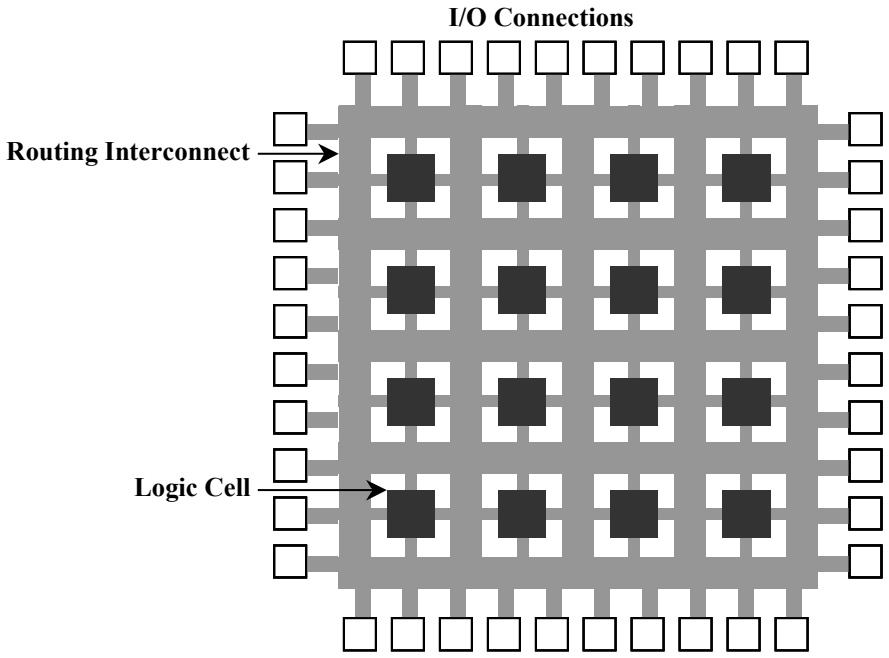

Figure 1: Typical FPGA Structure

2.2 SPIHT

For our system we selected the Set Partitioning in Hierarchical Trees (SPIHT) image compression 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 [14]. 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

2.2.1 Wavelet Transformation

The Fourier Transform (Figure 2) is used to convert an aperiodic signal from the time domain to the frequency domain [8]. The inverse Fourier Transform (Figure 3) may then be used to restore the original signal by converting the signal back to the time domain.

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t} dt$$

Figure 2: Fourier Transform

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{j\omega t} dt$$

Figure 3: Inverse Fourier Transform

Such a Fourier expansion yields full frequency resolution but no spatial/time resolution over a signal. Wavelet theory has been refined over the past 15 years to address the spatial resolution problem. Wavelets make use of scales to analyze data at various resolutions. By using a scalable window over a signal they provide a multi-resolution analysis of the signal [9]. Using these windows, it is possible to gather spatial-scale representations. Since the various scales correspond to different frequencies, wavelet analysis offers a spatial-frequency representation of a signal.

The typical wavelet uses a filter bank to process a high-pass and low-pass subband of the signal and down samples the results by a factor of two. By spatially compressing the wavelet by a factor of two, the frequency spectrum will stretch and shift by a factor of two as well, restoring the original frequency range. Thus by iteratively applying the filter bank to the low-pass result of the wavelet and down sampling the results, each wavelet covers half of the frequency components remaining (Figure 4). Lastly, a low-pass subband will represent all of the remaining frequency components [20].

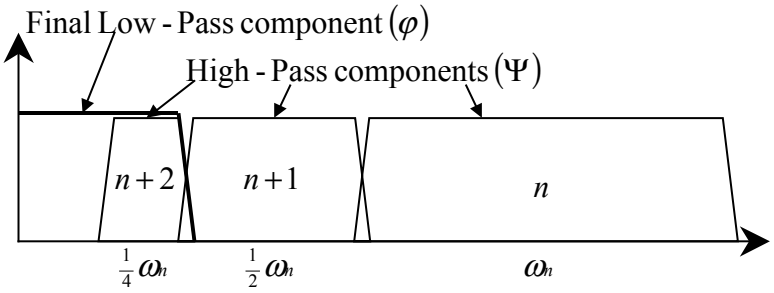

Figure 4: A set of Wavelet scales and one final low-pass function

2.2.2 Discrete Wavelet Transform

Since images are not continuous, but sampled at individual points or pixels, it is necessary to use a discrete form of the wavelet transform, as wavelets assume a continuous signal. The discrete wavelet transform runs a high and low-pass subband over the signal in one dimension. 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 low-pass subband. The 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 5).

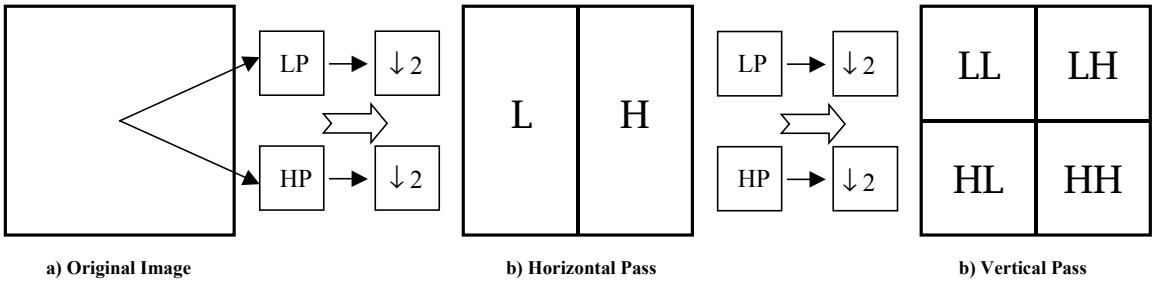

Figure 5: One Level Wavelet built by two one-dimensional passes

The vertical and horizontal transformations break up the image into four distinct subbands. The wavelet coefficients that correspond to the sharpest scale 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 [16].

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 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. Once the procedure is compete, the final LL subband is saved along with the other three subbands within the same level. Figure 6 displays an image with three scales of wavelet transformation.

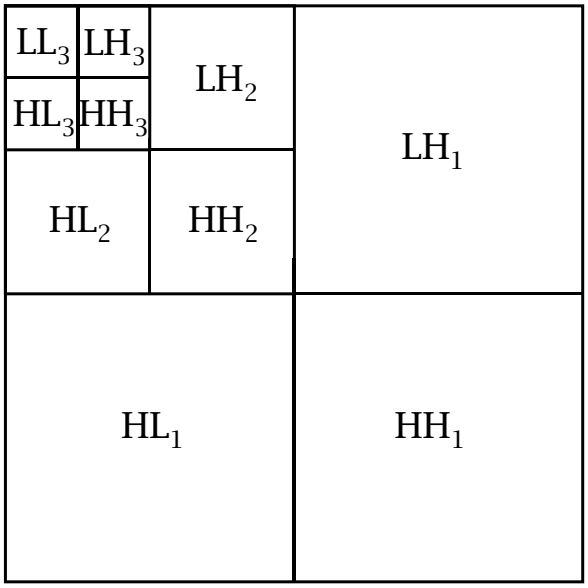

Figure 6: A three-level wavelet transform

2.2.3 SPIHT Coding

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 [13].

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 7 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.

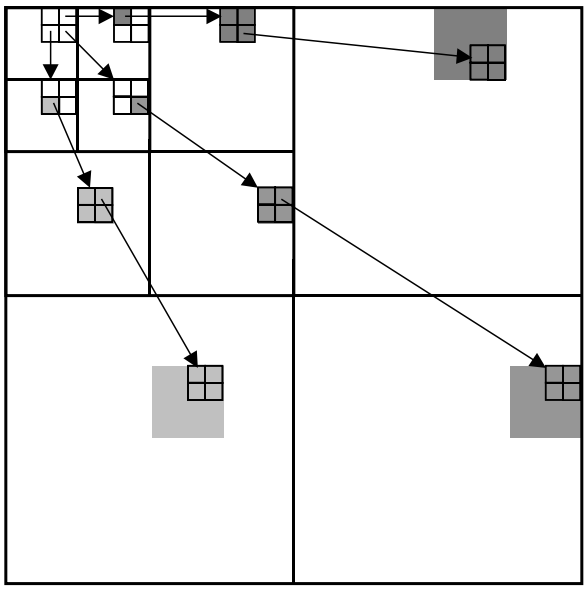

Figure 7: Spatial-orientation trees

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.

Information stating whether or not a pixel is above the current threshold or 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 is pixels that are currently being processed but have yet to be above the threshold. The LIS is pixels, which are currently being processed, but none of their descendants are yet above the threshold and are not being processed. Lastly the LSP is pixels, which were already stated to be above a previous threshold level, and now their value at each bit plane is transmitted.

Figure 8 is the algorithm copied from the original SPIHT paper [13]. It is modified to reflect changes discussed later in the paper referring to 2x2 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 pixel's (i,j) descendants are greater than the current threshold.

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 $S_n(i,j) = 1$ then 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 $S_n(k,l) = 1$ then 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 8: SPIHT Coding algorithm

There are two important concepts to take from the algorithm. First, as the encoder sequentially steps through the image and inserts/deletes pixels from the three lists, all of the information required to keep track of the lists is output to the decoder. In order for the decoder to reproduce the steps taken by the encoder you merely need to replace **output** in the encoder with **input** in the decoder. Second, the bit-stream produced is naturally progressive. A progressive bit-stream is one that can be cut off at any point and still is a valid bit-stream. As the decoder steps through the coding algorithm, it gathers finer and finer detail about the original wavelet transform. The decoder is able to stop at any point and perform an inverse transform with the wavelet coefficients it has currently reconstructed. Progressive bit-streams also have the advantage of being able to be reduced to an arbitrary size or be cut off during transmission and still produce a valid image.

3 Prior Work

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 [5][11][15]. Here we will provide a review of four individual DWT architectures and their performance where available.

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 [6]. Figure 9 illustrates how a 1-D DWT is used to realize a 2-D DWT.

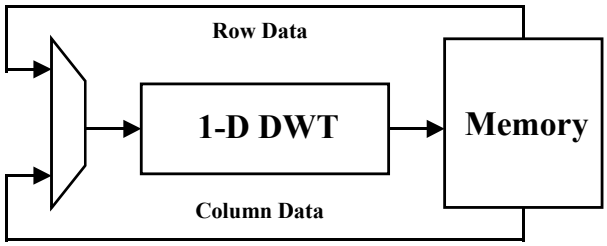

Figure 9: Illustration of a folded architecture

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.

In order to address these superfluous memory accesses the Recursive Pyramid Algorithm (RPA) was designed [21]. 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.

Another method to reduce memory accesses is the Partitioned DWT, which partitions the image into smaller blocks and computes several scales of the DWT at once for each block [12]. In addition, the algorithm makes use of wavelet lifting to reduce the computational complexity of the DWT [18]. 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 10 from Ritter et al. [12] illustrates how the partitioned wavelet is constructed.

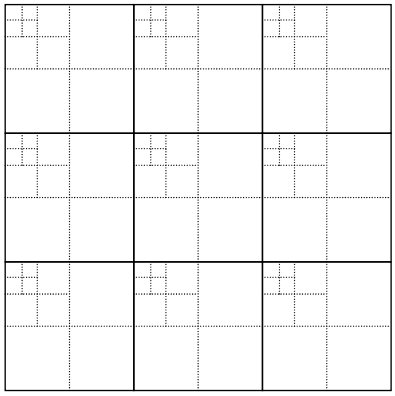

Figure 10: The Partitioned DWT

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.

[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.

The last unique architecture to discuss is the Generic 2-D Biorthogonal DWT shown in Benkrid et al. [3]. 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 11. 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.

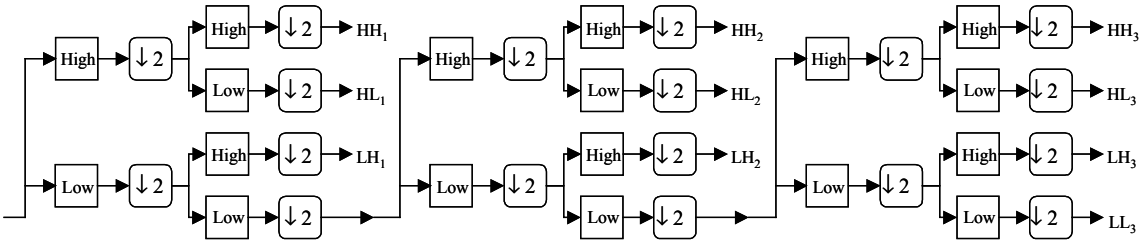

Figure 11: Generic 2-D Biorthogonal DWT

To date the literature contains very little on hardware implementations of SPIHT since the algorithm was developed so recently. Singh et al. [17] 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 in

what lists each pixel is active. The algorithm sequentially steps through the wavelet coefficients multiple times in the same order as the original software program. The design has been developed for 8 by 8 sized images and no performance numbers were given.

4 SPIHT 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.

4.1 Variable Fixed Point

The discrete wavelet transform produces real numbers as the wavelet coefficients. General-purpose computers realize real numbers as floating-point numbers. The representation of a floating-point number contains three parts: the sign bit, the exponent definition and the decimal description. A computer shifts the decimal description of the number by analyzing the exponential part. By utilizing these three parts, floating point numbers are able to represent a tremendous numerical range while still providing a high degree of precision.

Because of their inherit flexibility, floating-point numbers are well suited for a broad range of applications. Which is why general-purpose computers provide hardware support for floating-point numbers. However, floating-point numbers are not optimized for any specific application. For example, a data set that includes only numbers between −1000 and 1000 will waste many of the bits employed by an exponential description optimized for numbers in the range $\pm 10^{31}$. In order to conserve bits and optimize both memory storage and bandwidth, it is important to create a numerical representation specific to the data set under consideration.

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 often 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.

Table 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

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.

Figure 12 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.

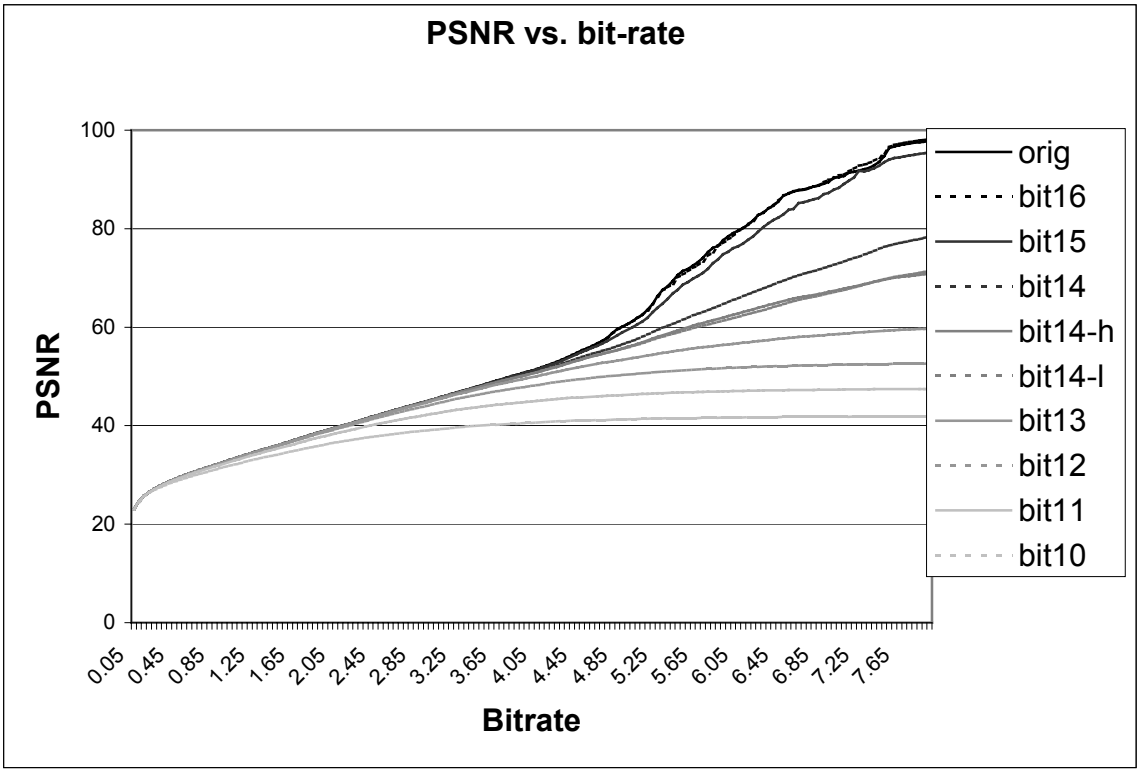

Figure 12: PSNR vs. bit-rate for various coefficient sizes

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. 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 12 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 [3]. 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: Final Variable Fixed-Point Representation

Wavelet Level	Integer Bits	Decimal 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

4.2 Fixed Order SPIHT

The SPIHT algorithm given in Figure 8 is designed to transmit data in the optimal order within each bit-plane. By always adding pixels to the end of the LIP, LIS and LSP lists, coefficients most critical to constructing a valid wavelet are generally placed first while less critical coefficients are placed last in the lists. Such an ordering will yield better

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

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.

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.
- The maximum magnitude of the four child trees.
- Some threshold and sign data of the 16 child coefficients when they are first inserted into the LIS list.

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 list order. Once the order is determined it is possible to produce a valid SPIHT bit-stream from the above information.

In order to determine an image's list order, it is necessary to sequentially step through the LIP and LIS lists for each bit-plane. By doing so each pixel must be considered multiple times and thus read from memory multiple times. Consequently, a hardware implementation of SPIHT is unable to calculate coefficient blocks in parallel. As a result, many of the speedups a custom hardware implementation may produce are lost. Instead, any hardware implementation must create the lists in the same manner as a software implementation. This process requires many clock cycles per block of coefficients.

We propose a modification to the original SPIHT algorithm called *Fixed Order SPIHT*. Fixed Order SPIHT is similar to the algorithm in Figure 8, 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 et al. [1].

Fixed Order SPIHT allows us to create a fully parallel version of the original SPIHT algorithm. Figure 13 outlines our new version of SPIHT. The final bit-stream generated is precisely the same as the bit-stream generated from the original SPIHT algorithm modified to use a Morton Scan Ordering. As a result the original decoder looping through the lists multiple times can decode the bit-stream generated by the parallel encoder.

```
1. Bit Plane Calculation: for each 2x2 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
```

(continued)

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 13: Fixed Order SPIHT

By using the algorithm in Figure 13 instead of the original sequential algorithm in Figure 8, the final data stream 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.

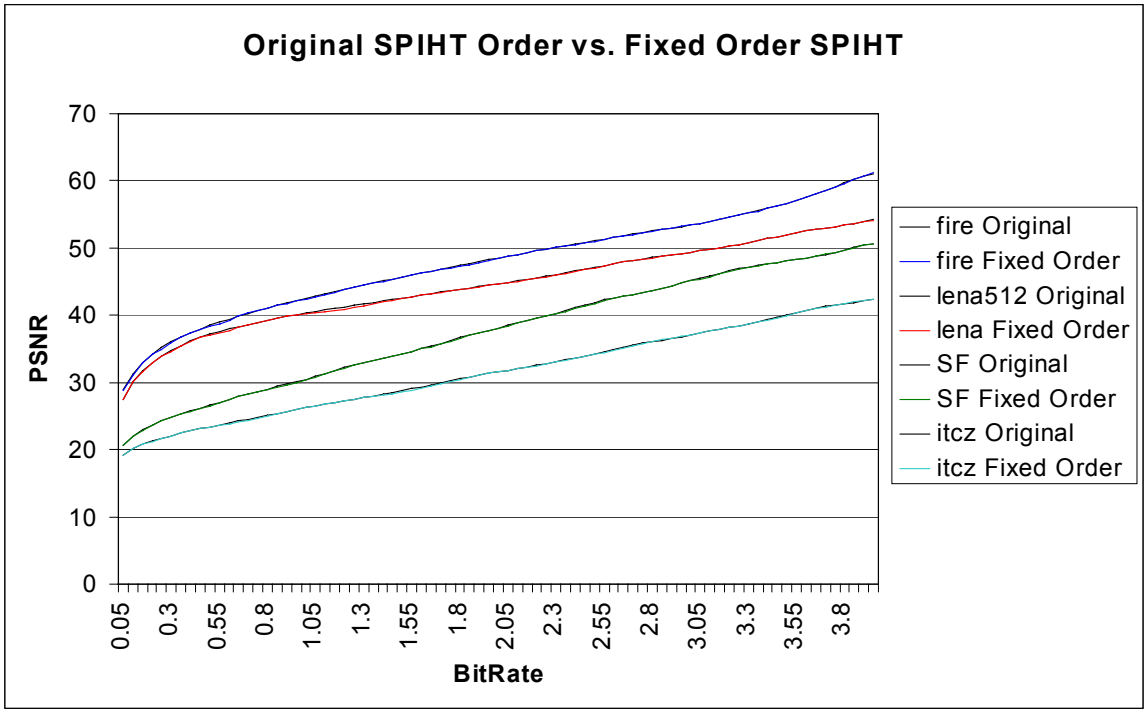

Figure 14: Comparison of Original SPIHT to Fixed Order SPIHT

An analysis between the original SPIHT algorithm and the new version using Morton Scan Ordering shows that the data streams are not identical. The original algorithm varies the order in which it transmits data to produce very high image quality within each pass. On the other hand Fixed Order SPIHT transmits the same data but in a fixed order. Therefore at the end of each LIP, LIS and LSP passes, the image quality of Fixed Order

SPIHT will match that of the original SPIHT algorithm. However since the new fixed order is not optimal for the image, within a pass there is a marginal drop in PSNR. Results show the loss of quality to be between at most 0.1 dB and 0.2 dB. Figure 14 provides an image quality comparison between the two algorithms. Because the data transmitted is the same at the end of each bit plane, the PSNR curves of Fixed Order SPIHT follow the original algorithm very closely.

5 Architecture

5.1 Target Platform

Our target platform is the WildStar FPGA processor board developed by Annapolis Micro Systems [2]. The board, shown in Figure 15, 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.

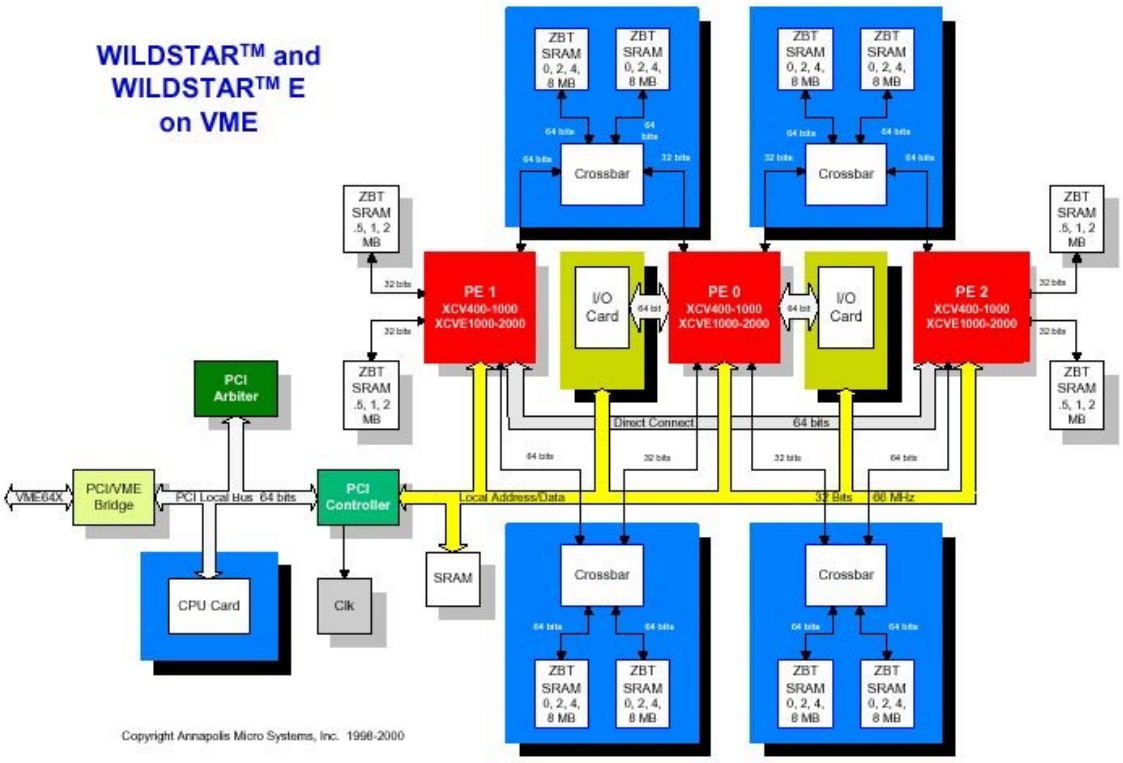

Figure 15: Annapolis Micro Systems WildStar board block diagram

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 FIFOs. A PCI bus connects the board to a host computer.

5.2 Design Overview

Our architecture consists of three phases: Wavelet Transform, Maximum Magnitude Calculation and 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. 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 16 illustrates the architecture of the system.

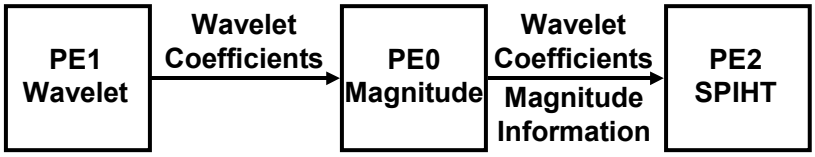

Figure 16: Overview of the architecture

5.3 Discrete Wavelet Transform 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 17 illustrates the architecture of the discrete wavelet transform phase.

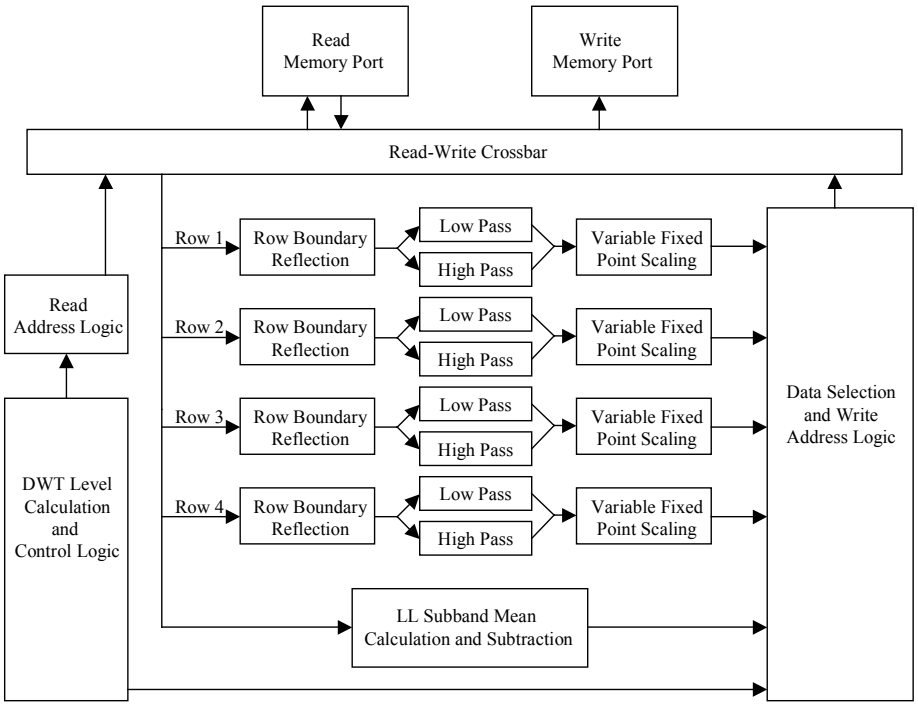

Figure 17: DWT Architecture

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 ¾th 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 ¾th a clock cycle per pixel. However, our FPGAs do not have such extensive resources.

By keeping the address and control logic simple, there are enough resources on the FPGA to implement 8 distributed arithmetic FIR filters [23] from the Xilinx Core library. The FIR filters use a bit-serial approach to compute the input stream and require significant FPGA resources, approximately 8% of the Virtex 2000E FPGA for each high and low-pass FIR filter. We chose the distributed arithmetic FIR filters because they produce a new result on every clock cycle.

5.4 Maximum Magnitude Phase

Once the DWT is complete the next phase must prepare and organize the image into a form easily readable by the parallel version of the SPIHT coder. Specifically the maximum magnitude phase needs to calculate and rearrange the following information for the next phase.

- The absolute value of the 2x2 block of coefficients.
- A sign value for each coefficient in the block.
- The maximum magnitude of each of the 4 child trees.
- 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 when they are first inserted into the LIS.
- Re-order the wavelet coefficients into a Morton Scan ordering.

The SPIHT coding phase shares two 64-bit memory ports with the magnitude phase allowing it to read 128 bits on each clock cycle. The data listed above 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 18. Since there are less than 32 threshold levels, a five-bit number can represent when a block is inserted into the lists.

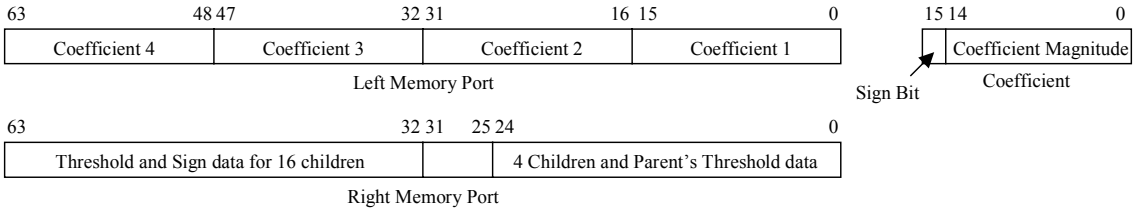

Figure 18: Data passed to the SPIHT coder to calculate a single block

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 [7]. 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.

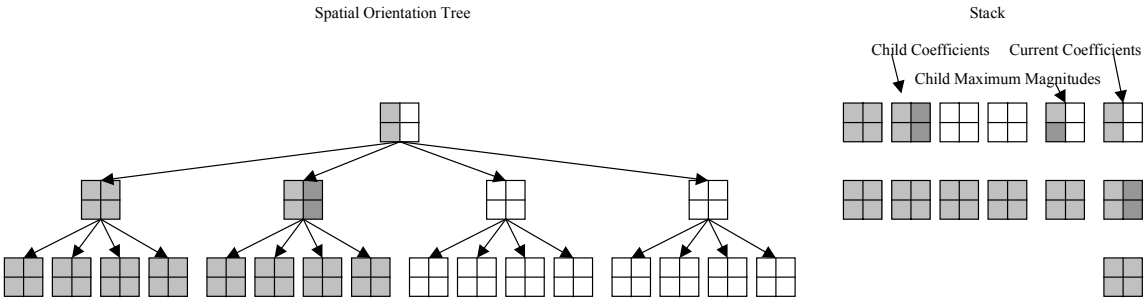

Figure 19: Depth-First Search of the Spatial Orientation Trees

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 19 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 that were previously read and processed. The dark gray blocks are the coefficients currently being read. In this example the state machine just finished reading the lowest level and has just ascended to the second wavelet level. The second block in the second level is now complete and its maximum magnitude can now be calculated, as shown as the dark gray block in the highest level of the stack. 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.

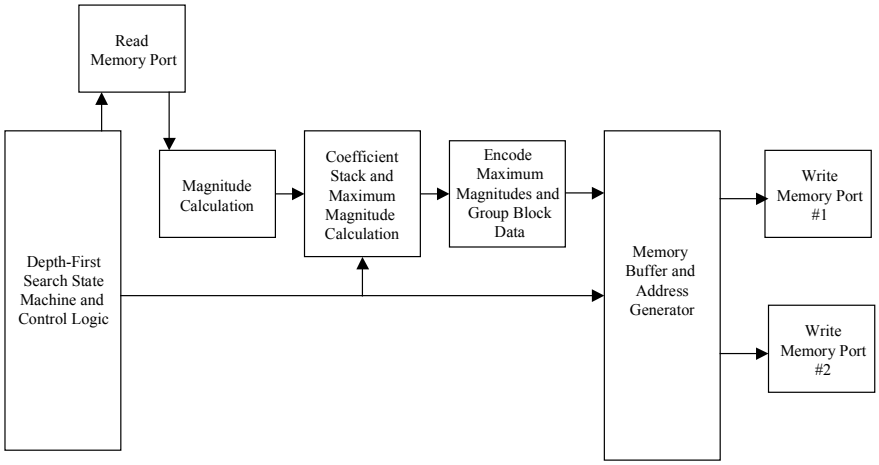

Figure 20: Maximum Magnitude Block Diagram

Another benefit of scanning the image in a depth-first search order is that the 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, data will be read in a Morton scan order. A block diagram of the maximum magnitude phase is provided in Figure 20. Since two pixels are read together and the image is scanned only once, the runtime of this phase

is ½ a clock cycle per pixel. Because the magnitude phase computes in less time than the wavelet phase, the throughput of the system is not affected.

5.5 SPIHT Coding Phase

The final SPIHT Coding phase essentially computes the parallel algorithm in Figure 13. 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 scheme 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 the 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 and the variable FIFOs are used to arrange the data into regular sized blocks. Separate FIFOs are used for each of the LIP, LIS and LSP bit-streams.

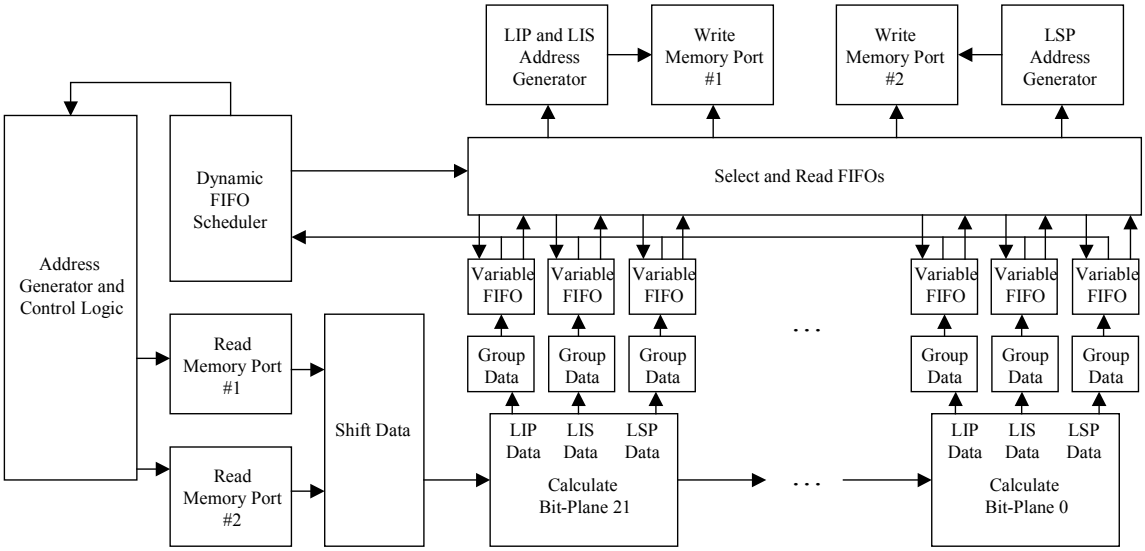

Figure 21: SPIHT Coding Phase Block Diagram

Lastly, data from each buffer is output to a fixed location in memory and the number of bits in each bit-stream is output as well. Given that data is added dynamically to each bit-stream, there needs to be a dynamic scheduler to select which buffer should be written to memory. Since there are a large number of FIFOs which all require a BlockRAM, the

FIFOs are spread apart on 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 upon both how full a FIFO is and when it was last accessed. Our studies show that the LSP bit-stream is roughly the size of the LIP and LIS streams combined. Because of this the LSP bit-streams transfer more data to memory than the other two lists. In our design the LIP and LIS bit-streams share a memory port while the LSP stream writes to a separate memory port. Since a 2x2 block of coefficients is processed every clock cycle, the design takes ¼ a clock cycle per pixel which is far less than the ¾ a clock cycle per pixel for the DWT. The block diagram for the SPIHT coding phase is given in Figure 21.

With 22 total bit-planes to calculate, the design involves 66 individual data grouping and variable FIFO blocks. Although none of the blocks consume a significant amount of FPGA resources individually, 66 blocks do. The entire design requires 160% of the resources in a Virtex 2000E and will not fit. 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 bit-stream required, the size of the FPGA used in the SPIHT phase can be varied to handle the number of intermediate bit-streams generated.

Removing lower bit-planes is possible since the final bit-stream transmits data from the highest bit-plane to the lowest. In our design the lower 9 bit-planes were eliminated. Yet, without these lower planes, bit-rates 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 bit-rates. Since SPIHT is optimized for lower bit-rates, the ability to calculate higher bit-rates is not considered necessary. In addition the use of a larger FPGA would alleviate the size constraint.

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 [14]. The comparison was made without arithmetic coding since our hardware implementation currently does not perform any arithmetic coding on the final bit-stream. A SPARC 5 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.

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 phases 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.

The entire system has been tested over many images of variable size and functions correctly. However, at the time of the writing of this paper there is one bit-stream generated by the SPIHT phase with a slight glitch. The first word written to memory of one stream is blank yet the stream is valid after the first 32 bits of data. Simulations in ModelSim show the design computing correctly and it appears to be a synthesis problem with Synplify. Part of our ongoing development effort is to both lower the size requirement of the final SPHIT phase and correct this one glitch.

7 Conclusions and Future Work

In this thesis 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. In addition our Fixed Order SPIHT design illustrates how by making slight adjustments to an existing algorithm it is possible to dramatically increase the performance in a custom hardware implementation.

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.

Bibliography

[1] V. R. Algazi, R. R. Estes. "Analysis based coding of image transform and subband coefficients," *Applications of Digital Image Processing XVIII*, volume 2564 of *SPIE Proceedings*, pages 11-21, 1995.

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

[3] 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.

[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 (Dec. 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*, Vol. 43, pp 759 – 771, March 1995.

[6] 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.

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

[8] R. C. Gonzalez, R. E. Woods, *Digital Image Processing*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1993.

[9] A. Graps, "An Introduction to Wavelets," *IEEE Computational Science and Engineering,* Vol. 2, Num. 2, pp. 50 - 61, Summer 1995.

[10] R. H. Katz, *Contemporary Logic Design*, The Benjamin/Cummings Publishing Company, Inc. Redwood City, CA. pp. 524 – 538, 1994.

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

[12] 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.

[13] 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.

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

[15] 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.

[16] J. M. Shapiro, "Embedded Image Coding Using Zerotrees of Wavelet Coefficients," *IEEE Transactions on Signal Processing*, Vol. 41, No. 12, pp 3445 - 3462, December 1993.

[17] 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.

[18] 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.

[19] 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*, pp 305-316, May 1995.

[22] Xilinx, Inc., *The Programmable Logic Data Book*, California: Xilinx, Inc., 2000.

[23] Xilinx, Inc., *Serial Distributed Arithmetic FIR Filter*, California: Xilinx, Inc., 1998.