FPGA BASED IMAGE PROCESSING WITH R-FUNCTIONS AND THE CURVELET TRANSFORM

A Thesis

by

JOHN L. WISINGER JR.

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2003

Major Subject: Computer Engineering

FPGA BASED IMAGE PROCESSING WITH *R*-FUNCTIONS AND THE CURVELET TRANSFORM

A Thesis

by

JOHN L. WISINGER JR.

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

[illegible]
Rabi Mahapatra
(Chair of Committee)

[illegible]
Hank Walker
(Member)

[illegible]
Karen Butler-Purry
(Member)

[illegible]
Donald Friesen
(Head of Department)

May 2003

Major Subject: Computer Engineering

ABSTRACT

FPGA Based Image Processing with R-functions and the Curvelet Transform. (May 2003)

John L. Wisinger Jr., B.S., Texas A&M University

Chair of Advisory Committee: Dr. Rabi Mahapatra

In the past few years, image processing has begun to make its way into many new areas, both academic and commercial. One of the most popular areas is in computer generated animation. This includes films, video games, medical imaging, and various other multimedia systems for both entertainment and more serious applications. Two fairly recent independant developments in this field are R-functions and the curvelet transform. R-functions were developed to make it possible to represent complex objects by using a collection of simpler primitives. The curvelet transform was designed to extract specific features from complex objects.

Although impressive performance can be achieved with R-functions and curvelets, the complexity of their implementation is quite a drain on standard microprocessors. It is for this reason that an FPGA implementation was developed. By offloading some of the processing work into a properly configured FPGA, speeds can be achieved in excess of one hundred times faster than current high end servers.

This increase in processing speed and image representation ability combine to have some useful applications. Now, highly complex image processing can be done in small areas allowing for the design of systems that were previously not feasible to develop. By using the concepts presented in this thesis, ideas have come about for the development of a large scale Boltzman equation solver, and a satellite hyperspectral imaging system. The Boltzman equation solver has been developed before, but only by using very costly and space consuming servers. Design of the satellite hyperspectral imaging system has been hindered by the low data transmission rate of the communication system. By processing some of the data on the system itself this problem is removed.

This thesis proves that *R*-functions and numeric transforms can be done in an FPGA to give far better performance than regular microprocessors. It also shows the power of the *R*-function and the curvelet and ridgelet transforms. With further development, this could yield some amazing results.

To

Bob

ACKNOWLEDGMENTS

Thanks to Brenna, Brian, Ian, Joe, Junyi and Siddharth for making my life in grad school fun. Thanks to Texas A&M for making my life in grad school. Thanks to my parents for making my life

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

Image processing has become a very popular field in the past few years. It consist of two main areas. One involves breaking complex images into smaller parts. The other consists of combining these smaller parts into complex images. Both of these are important in many modern high end computing tasks. The best example would be computer generated animation. To create the characters, complex images are built up from much simpler shapes. Once the scene has been created, it must often be decomposed in a different form in order to compress it into a digital video. These tasks are very complex and require a large amount of computation to be completed. Creating specialized hardware would greatly reduce the time consumed by these processes. Also, the use of advanced techniques in image construction and decomposition would greatly increase the speed and effectiveness of the overall process. It is for this reason that an FPGA implementation of R-functions and the curvelet transform are proposed and demonstrated.

A. Motivation

1. Why R-functions

A popular field known as computational solid geometry (CSG) involves the modeling of complex objects through the combination of simpler functions such as lines and circles. When this is done incrementally, there is repeated floating point rounding that causes a loss of accuracy as well as the obvious delay of computing the location of a point among so many functions. R-functions offer a solution to this problem.

The journal model is *IEEE Transactions on Automatic Control.*

Through the use of R-functions, the many small functions can be combined into one large function that represents the entire complex image. This reduces the floating point error and greatly increases the speed in which the locations of points within the object can be calculated.

2. Why the curvelet transform?

It has long been known that the wavelet transform has many limitations when it comes to representing straight lines and edges in image processing. Not long ago, researchers at Stanford developed a solution to this problem[1, 2]. They created the curvelet transform, a transform that uses wavelets, but handles edges and lines much better.

3. The correlation between R-functions and curvelets

R-functions are used for object modeling. The curvelet transform is used for image processing. On the surface, these may seem unrelated. On further inspection, it can be seen that the curvelet transform is used for image decomposition and reconstruction. R-functions are used for object construction. In many cases this object will be converted into an image.

To model any object, the basic shapes that make it up need to be known. Once this is done, R-functions can be used to create the model. To get these basic shapes, the curvelet transform can be used for edge finding and similar techniques to decompose an image into its basic parts. These parts can then be used to model the object with R-functions.

4. Why implement R-functions and curvelets in hardware?

One of the primary advantages to FPGAs is their reconfigurable architecture. This opens up a whole world of possibilities unavailable in microprocessors, digital signal processors (DSPs), application specific integrated circuits (ASICs), or any other chip with a specific architecture.

One of the most obvious possibilities is the reduction of the total number of chips (which reduces cost and board area). In many cases, it is useful to have a device that performs two different tasks. One example would be a device that implements several different transforms and their inverses. The hardwired architecture of standard microprocessors and DSPs gives the designer limited opportunities to change they way these things are done. Obviously it is possible to store any number of transform algorithms in memory and then run them on these processors, but the hardware is unable to adapt to the different transforms, and therefore the designer is forced to make due with what is given to them.

One solution to this would be an ASIC. This would give the designer the freedom to modify the hardware in the most optimal way for each transform. Unfortunately, a separate ASIC would be required for every transform that was desired. This is the advantage of an FPGA. Several transforms can be designed and stored in ROM on the board. A single FPGA can be reprogrammed on the fly to perform any of these stored transforms when needed. Take the example of designing a board that needs to perform five different transforms. One could design a board with ten parallel DSPs or microprocessors that can be used in any way to calculate the transforms. Instead, that same area and money can be used for ten FPGAs to achieve an increase in speed. A board could be filled with ten different ASICs (two for each transform), which would give an even higher speed than FPGAs. The advantage to FPGAs is that if the application needed ten of one transform and none of the others, eight ASICs would sit idle, while other data would be waiting. The FPGAs could be reconfigured very quickly to handle this problem.

Another major advantage to FPGAs is the fact that they can be configured as dedicated processors. That means that there is no overhead for an operating system. This is a large time advantage. The other speedup is in stages of the data path within the processor. Many instructions do not need to go through each stage of the data path, but in order to keep the pipeline running smoothly, they must sit in that stage and wait. With an FPGA design, this is unnecessary since those other instructions would not be included.

By implementing the curvelet transform and the R-function concept as an FPGA coprocessor, these complex transforms could be done on large amounts of data faster than any current method. By using highly parallel hardware and the most advanced transforms, large amounts of image data can be processed in very small amounts of time.

B. Related Work

1. R-functions

R-functions were developed by Vladimir Rvachev in the mid 1960s [3]. Since then they have been used in many applications. Most of these are related to the modeling of solid objects [4]. Many involve computer graphics applications, but some are also used for system modeling and equation solving. There have been numerous implementations of R-functions on these different applications [5, 6]. However, as far as we know, there has never been an attempt to implement R-functions in hardware.

2. Curvelet

Due to the relative newness of the ridgelet and curvelet transforms, there have been very few implementations of each. The majority of the published work is from those who originally developed the transforms[1]. All of these implementations have been done in software only. Most of them were Matlab implementations of the equations themselves. Both transforms were used several times to demonstrate their various abilities such as denoising and compression[7, 8]. To our knowledge, this is the only hardware implementation of the radon and ridgelet transforms as well as their inverses.

3. FPGA based implementation

The wavelet transform has been implemented rather exhaustively[9, 10]. It has been done in hardware, software and every combination of the two. Because of this fact, implementing the wavelet transform will be the smallest part of this work. The radon transform has many software implementations, and even a one-sided hardware implementation, but we have not seen any complete implementations in any form of hardware.

4. Codesign

The codesign methodology (the combination of hardware and software) is becoming increasingly more popular[11, 12]. Its use often involves FPGAs, and it has helped to show the superior performance that FPGAs can provide. There have been a few instances of using codesign techniques in image processing[13, 14]. These focus primarily on the wavelet transform. We have never seen any attempts at using codesign for radon, ridgelet or curvelet transforms.

C. Thesis contribution

The primary contribution of this thesis is the combination of R-functions, the curvelet transform, and the FPGA. By using the three of these together, this thesis proves that impressive results can be obtained. Large amounts of image data can be decomposed and reconstructed in a small amount of time while taking up a small amount of area.

One possible application of the curvelet in an FPGA would be in the development of a hyperspectral imager. This device would orbit the earth and take pictures of the sky over time. What would then be left, are three-dimensional pictures of stars that become represented as lines (since the star moves over time). This data then needs to be sent back to stations on the earth for processing. The problem is that the slow communication system from space to earth, does not allow this much data to be sent fast enough. Obviously the best solution would be to compress this data.

This is where the curvelet transform becomes useful. The curvelet is the best at compressing straight lines in images, so it would be the optimum choice. The next problem is speed. There needs to be a way to compress this data as fast as the camera can take pictures. Other features that the hyperspectral imaging device might need are navigation, power management and communication control. These features will not constantly be needed, so using dedicated processing power for them would be somewhat wasteful. By having FPGAs in the system, they could be reconfigured to control the device's movement, power, etc. when needed, or be used to do transforms when the device is located in the proper spot. This means that the FPGAs that are in the system to do image processing can also be used for totally different purposes. This would be totally impossible if using an ASIC. Since DSPs are designed specifically for signal processing, tasks that require different capabilities (such as bus communication or power electronics control) could not be done by them, while an FPGA would be perfect.

There is some overhead in using an FPGA in a system. In order for an FPGA to be reconfigured, the configuration must be stored in memory chips in the system. Every time an FPGA is reconfigured, it takes a certain amount of time to do this. This time can range from 1.2 ms for the smallest Virtex chip (XCV50) to 31ms for the largest Virtex-E (XCV3200E). For some FPGAs, partial reconfiguration is also an option if the entire chip does not need to be modified. This can happen in as little as 4 μs to change a tiny fraction of the chip. FPGAs also consume more power than most ASICs and are usually less dense then ASICs. These things may point against FPGAs, but the advantages definitely outweigh the disadvantages.

Another advantage of FPGAs is the ability to find a radiation hardened version. Radiation hardness is a very important quality for electronics devices that will be used in space (as well as some other areas). FPGAs are widely used as radiation hardened devices, and it is also common to see DSPs implemented in an FPGA in order to gain radiation hardness. Radiation hardened DSPs are not easy to find on the market (something which would make their cost very high). FPGAs are also known to consume less power than DSPs.

CHAPTER II

BACKGROUND

A. R-functions

1. The purpose of R-functions

Although R-functions have many uses, the most interesting in modern research is probably the ability to describe geometric objects. Previously, complex objects needed to be described by a system of functions (or, for solid objects, inequalities would be used). With the use of R-functions, these complex objects can now easily be described with only one inequality. If chosen properly, these functions can even have a very useful set of logic properties as shown in Tables I and II. R-functions have been used in many applications including medical diagnostics and computer graphics (for movie-making and video games).

2. R-functions defined

An R-function is a real function that has some property that is entirely dependent upon the corresponding property of its inputs. An example would be a function in which the sign was determined only by the sign of the input arguments. In simpler words, a real function can be described as an R-function if a certain property (like sign) can only change when some of its inputs change that property. Sign allows only two levels of partitioning, positive or negative (or three if you count zero as its own category). However there are some R-functions in which the space of real numbers is partitioned into k subsets allowing more in depth analysis. For the purposes of this work, we will concentrate only on the two subset cases.

3. R-function mathematics

There are three R-functions defined for this work, conjunction, disjunction and negation. It can be shown that all other logic functions can be derived from these three basics (technically, disjunction is not needed for completeness, but it is included for convenience).

a. Conjunction

The R-function for conjunction is defined as

$$x_1 \wedge_\alpha x_2 \equiv \frac{1}{1+\alpha}\left(x_1 + x_2 - \sqrt{x_1^2 + x_2^2 - 2\alpha x_1 x_2}\right).$$

For the application described in this thesis, the value of α turns out to be unimportant (it is only important when the R-functions need to be differentiable). If α is chosen to be 1, the equation becomes

$$x_1 \wedge_1 x_2 \equiv \frac{1}{2}\left(x_1 + x_2 - \sqrt{(x_1 - x_2)^2}\right).$$

This should be recognized as the exact definition of the $min(x_1, x_2)$ operator. At this point all of the mathematics becomes simply a comparison.

b. Disjunction

The R-function for disjunction is defined as

$$x_1 \vee_\alpha x_2 \equiv \frac{1}{1+\alpha}\left(x_1 + x_2 + \sqrt{x_1^2 + x_2^2 - 2\alpha x_1 x_2}\right).$$

Of course if we again choose α to be 1, we get

$$x_1 \vee_1 x_2 \equiv \frac{1}{2}\left(x_1 + x_2 + \sqrt{(x_1 - x_2)^2}\right),$$

Table I. Basic Logic Properties of R-functions

$\overline{\overline{x}}$	$\equiv$	x
$x \wedge x$	$\equiv$	x
$x \vee x$	$\equiv$	x
$x \vee \overline{x}$	$\equiv$	$\|x\|$
$x \wedge \overline{x}$	$\equiv$	$-\|x\|$
$x_1 \wedge x_2$	$\equiv$	$x_2 \wedge x_1$
$x_1 \vee x_2$	$\equiv$	$x_2 \vee x_1$
$\overline{x_1 \wedge x_2}$	$\equiv$	$\overline{x_1} \vee \overline{x_2}$
$\overline{x_1 \vee x_2}$	$\equiv$	$\overline{x_1} \wedge \overline{x_2}$

which is the definition of the $max(x_1, x_2)$ operator.

c. Negation

The R-function for negation is the simplest of all being defined as

$$\overline{x} \equiv -x.$$

It should be clear that each of these three operators can be implemented very efficiently with little calculation involved.

d. Logic properties of R-functions

R-functions have many of the same properties as their companion Boolean functions. Tables I and II show a list of several of these properties. For a complete list of all of the properties, consult the references [5].

Table II. Advanced Logic Properties of R-functions

$(x_1 \wedge x_2) + (x_1 \vee x_2)$	$\equiv$	$x_1 + x_2$
$(x_1 \wedge x_2)(x_1 \vee x_2)$	$\equiv$	$x_1 x_2$
$x_1 \wedge (x_2 \wedge x_3)$	$\equiv$	$(x_1 \wedge x_2) \wedge x_3$
$x_1 \vee (x_2 \vee x_3)$	$\equiv$	$(x_1 \vee x_2) \vee x_3$
$x_1 \wedge (x_2 \vee x_3)$	$\equiv$	$(x_1 \wedge x_2) \vee (x_1 \wedge x_3)$
$x_1 \vee (x_2 \wedge x_3)$	$\equiv$	$(x_1 \vee x_2) \wedge (x_1 \vee x_3)$
$(x_1 \wedge x_2) \vee x_1$	$\equiv$	x_1
$(x_1 \vee x_2) \wedge x_1$	$\equiv$	x_1

4. An example usage of R-functions

The power of R-functions can best be shown in an example. Figure 1 shows a two dimensional object alongside the simple functions that compose its borders.

The simple functions shown in the second object are represented by the following mathematical inequalities[5]:

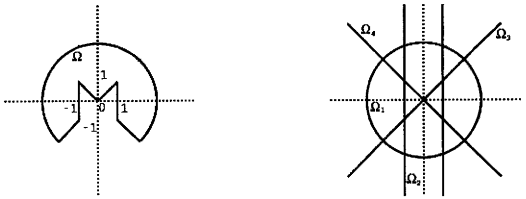

Fig. 1. Complex object and its basic components

$$\begin{aligned}
\Omega_1 &= (9 - x^2 - y^2 \geq 0) && \text{circle;}\\
\Omega_2 &= (x^2 - 1 \geq 0) && \text{vertical strip;}\\
\Omega_3 &= (y - x \geq 0) && \text{line;}\\
\Omega_4 &= (y + x \geq 0) && \text{line;}
\end{aligned}$$

By shading in the area in the first object of Figure 1 on the second image, it can be seen that the proper logical combination of the simple functions is

$$\Omega = [(\Omega_3 \wedge \Omega_4) \vee \Omega_2] \wedge (\Omega_3 \vee \Omega_4) \wedge \Omega_1.$$

By inserting the four simple inequalities into the above function, the original object is created perfectly with the use of only one function. This will hold true for even the most complex of objects.

B. Transforms

The curvelet transform is a combination of several other transforms. To understand how curvelets work, a certain knowledge of the transforms that comprise the curvelet is required.

1. Wavelets

a. The purpose of the wavelet transform

Wavelets are designed to hierarchically decompose a function. This function can be an image, signal, surface, etc. First, we label the vector space that includes all possible functions that can be contained in an image with $j-1$ pixels as V^j. Next, we define a new vector space W^j as the orthogonal complement of V^j in $V^{(j+1)}$.

Any set of linearly independent functions ψ_i^j that span W^j are called wavelets. The particular basis functions chosen determine the type of wavelet decomposition that can be performed.

Using the wavelet basis functions, we can recursively break a function into its course shape and a set of detail functions. If this is done enough, eventually what is left is a very simple course function and a large set of (hopefully similar) detail coefficients that correspond to the wavelet basis functions[9, 15].

b. The Haar wavelet

The one dimensional Haar basis is the simplest wavelet basis. Its functions are given by

$$\psi_i^j(x) := \psi(2x^i - i), i = 0, ..., 2^j - 1,$$

where

$$\psi(x) := \begin{cases} 1 & \text{for } 0 \leq x < 1/2 \\ -1 & \text{for } 1/2 \leq x < 1 \\ 0 & \text{otherwise.} \end{cases}$$

The implementation of the Haar wavelet decomposition can most easily be shown with an example. Begin with a one dimensional "image" containing four pixels:

$$[\ 3\ \ 7\ \ 9\ \ 5\ ]$$

If we average the two pairs of pixels, we obtain

$$[\ 5\ \ 7\ ]$$

The detail coefficients are the differences between the original numbers and their average. In this case since 3 is 2 less than 5 and 9 is 2 greater than 7, the detail coefficients are

$$[\ -2 \quad 2\ ]$$

If we repeat this process until only one average is left, we get

Iteration	Averages	Detail Coefficients	Image
0	$[\ 3\ 7\ 9\ 5\ ]$		$[\ 3\ 7\ 9\ 5\ ]$
1	$[\ 5\ 7\ ]$	$[\ -2\ 2\ ]$	$[\ 5\ 7\ -2\ 2\ ]$
2	$[\ 6\ ]$	$[\ -1\ ]$	$[\ 6\ -1\ -2\ 2\ ]$

In the end, a large average is left, but the detail coefficients tend to be small and similar numbers. This is the feature that makes the wavelet transform good for image compression.

c. The inverse Haar wavelet

Once the Haar wavelet is understood, its inverse becomes very simple. By reversing all the steps in the original transform, the inverse is created. The inverse in the above example can be shown as

Iteration	Averages	Detail Coefficients	Image
0	$[\ 6\ ]$	$[\ -1\ ]$	$[\ 6\ -1\ -2\ 2\ ]$
1	$[\ 5\ 7\ ]$	$[\ -2\ 2\ ]$	$[\ 5\ 7\ -2\ 2\ ]$
2	$[\ 3\ 7\ 9\ 5\ ]$		$[\ 3\ 7\ 9\ 5\ ]$

If the values are stored to one decimal place perfect reconstruction can be obtained.

d. The "á trous" wavelet

The "á trous" wavelet begins by selecting a low-pass filter H that satisfies

$$h_{2k} = \frac{\delta(k)}{\sqrt{2}},$$

where $\delta_{k,m}$ is the Kronecker delta[10].

In this case, the Lagrange interpolation filter was used. It is given as

$$h = \frac{1}{2} * \frac{1}{\sqrt{2}}(1,1) * \frac{1}{\sqrt{2}}(1,1) = \left(\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\right).$$

The wavelet functions are defined as

$$\psi(x) = \phi(x) - \frac{1}{2}\phi(\frac{x}{2}).$$

When the image is decomposed, it gives the course approximation and the detail functions (wavelet coefficients). By taking the scalar product of the function $f(x)$ with the scaling function $\phi(x)$ the first approximation is given as

$$c_0(k) = \langle f(x), \phi(x-k) \rangle.$$

Subsequent approximations are therefore given by the direct "á trous" decomposition

$$c_i(k) = \frac{1}{2^i}\langle f(x), \phi\left(\frac{x-k}{2^i}\right)\rangle.$$

The recursive "á trous" decomposition is defined as

$$c_i(k) = \sum_l h_l c_{i-1}(k + 2^{i-1}l).$$

The recursive formula for the wavelet coefficients is

$$w_i(k) = c_{i-1}(k) - c_i(k).$$

The "á trous" algorithm can best be explained with the following C code

```
for(j = 1; j <= MAX_LEVEL; j++) {
  for(l = 0; l < YRANGE; l++) {
    for(k = 0; k < XRANGE; k++) {
      c[j][k][l] = 0;
      for(y = 0; y < FILTER_DEGREE; y++) {
        for(x = 0; x < FILTER_DEGREE; x++) {
          if(k-offx < 0) offx -= XRANGE;
          if(l-offy < 0) offy -= YRANGE;
          c[j][k][l] += c[j-1][(k-offx)%XRANGE][(l-offy)%YRANGE]
                        / filter_mask(x,y);
        }
      }
      w[j][k][l] = c[j-1][k][l] - c[j][k][l];
    }
  }
}
```

where FILTER_DEGREE is the number of coefficients in the filter, XRANGE and YRANGE are the dimensions of the image and offx and offy are given by

$$offx = 2^j \times \left(x - \frac{FILTER\_DEGREE}{2}\right).$$

"filter_mask(x,y)" is the Lagrange interpolation filter given above convoluted with itself.

e. The inverse “á trous” wavelet

As with all of the wavelet transforms, the inverse of the “á trous” is just the sum of its parts. Formally, it is given by the following equation

$$I(x,y) = c_J(x,y) + \sum_{j=1}^{J} w_j(x,y),$$

where I is a two dimensional image.

2. Radon

a. The purpose of the radon transform

The radon transform has several purposes. Most of them involve reconstructing images. Its inverse, called back projection, is especially effective for this. The radon transform basically looks for lines in the image and tries to compress or reconstruct (depending on the application) based on those lines.

Unfortunately, the obvious discrete version of this transform is not effective for digital images. Since digital images have finite boundaries, the lines from the radon transform have different lengths depending on where they cross through the image. When this is reversed by the back projection equation, the unequal lengths are falsely interpreted and the recovered image looks very unlike the original.

The most popular solution to this problem involves the use of the Fourier transform. However, the newly developed finite radon transform was chosen in this case due to its more efficient FPGA implementation. Since an understanding of the radon transform aids in the explanation of the finite radon transform, both are included here.

b. The radon transform

The continuous radon transform is a collection of line integrals over a function (or in this case an image). These integrals are given by

$$R_f(\theta, t) = \int\int f(x_1, x_2)\delta(x_1 \cos\theta + x_2 \sin\theta - t)\, dx_1 dx_2.$$

The discrete version of this transform would simply replace the integral with a sum,

$$R_f(\theta, t) = \sum_{x_1 \in f} \sum_{x_2 \in f} f(x_1, x_2)\delta(x_1 \cos\theta + x_2 \sin\theta - t).$$

At this point, implementation becomes fairly straightforward. Each point in the radon transformed image is simply the addition of all the points along a line in the image. The values θ and t control the angle and offset of the line.

c. The inverse radon transform

The inverse radon transform is also known as back projection. By itself, it has several uses[16, 17, 18]. The continuous version is given by

$$B_g(x, y) = \int g(x \cos\theta + y \sin\theta, \theta)\, d\theta,$$

with the discrete being given by

$$B_g(x, y) = \sum_{\theta \in g} g(x \cos\theta + y \sin\theta, \theta).$$

d. The digital radon transform using the Fourier domain

There is a fundamental property of the Radon transform known as the projection-slice formula

$$Ff(\lambda\cos\theta, \lambda\sin\theta) = \int Rf(t,\theta)e^{-\imath\lambda t}\,dt.$$

This formula is the basis for the approximation of the radon transform using the Fourier domain. For an image $f(i_1, i_2)$ the following steps will produce the approximate radon transform

1. *2D-FFT.* Compute the two-dimensional FFT of f.
2. *Cartesian to Polar Conversion.* Use an interpolation scheme to convert to a polar coordinate system.
3. *1D-IFFT.* Compute the IFFT for each value of the angular parameter.

The inverse of this is basically each step backwards. Due to the fact that this method was not chosen for the FPGA implementation, further details will be omitted. Consult the references for more information[1, 19, 8, 7].

e. The finite radon transform

To combat the previously mentioned "wrapping" problem in the digital radon transform, the finite radon transform (FRAT) was developed[20]. If we say $Z_p = \{0, 1, ..., p-1\}$, where p is a prime number, then the FRAT of any real function f is defined as

$$FRAT_f(k,l) = \frac{1}{\sqrt{p}} \sum_{(\imath,\jmath)\in L_{k,l}} f(i,\jmath).$$

In this case $L_{k,l}$ is defined as

$$L_{k,l} = \{(i,j) : j = ki + l \pmod{p}, \imath \in Z_p\}, k \in Z_p.$$

$$L_{p,l} = \{(l,j) : j \in Z_p\}$$

The primary things that make the FRAT different than the original radon transform are the normalization factor $\frac{1}{\sqrt{p}}$ and the modulus in the definition of the line $L_{k,l}$. This modulus has the effect of making all lines equal length, by "wrapping" them around the image, and therefore making the inversion much easier[21, 22].

f. The inverse finite radon transform

Just as with the regular radon transform, the inverse finite radon transform is known as finite back projection (FBP). It is given by

$$FBP_r(i,j) = \frac{1}{\sqrt{p}} \sum_{(k,l)\in P_{i,j}} r(k,l),$$

where $P_{i,j}$ is the set of indexes for lines crossing the point (i,j) or

$$P_{i,j} = \{(k,l) : l = j - ki \pmod{p}, k \in Z_p\} + \{(p,i)\}.$$

It can easily be shown that this equation combined with the FRAT gives back the original function. The only real disadvantage to the FRAT and FBP combination (instead of the regular radon transform) is that the sides of the image must be prime numbers[21, 22].

3. Ridgelet

a. The purpose of the ridgelet transform

A singularity is when several pixels are very similar in color. A point singularity would be several pixels in a cluster that have similar color. The wavelet transform is very good with images that have point singularities. However, it has poor performance on images with many straight lines in them. The ridgelet transform is very similar to the wavelet, except that it performs better with line singularities. Put more simply, the wavelet transform would work well on a night sky image, but not vertical bars. The ridgelet performs well on both [23, 16].

b. The continuous ridgelet transform

Just as the wavelet transform breaks an image into a wavelet basis (called wavelets), the ridgelet transform breaks an image into a ridgelet basis (called ridgelets). The ridgelets are defined as

$$\psi_{a,b,\theta}(x) = a^{-\frac{1}{2}}\psi\left(\frac{x_1\cos\theta + x_2\sin\theta - b}{a}\right).$$

Using the given ridgelets, an equation can be derived to give the ridgelet coefficients. This ridgelet transform equation is

$$\Re_f(a,b,\theta) = \int \psi_{a,b,\theta}(x)f(x)\,dx.$$

For reference, the formal definition of the wavelet transform is similarly given as

$$W_f(a_1,b_1,a_2,b_2) = \int \psi_{a_1,b_1,a_2,b_2}(x)f(x)\,dx.$$

It was also discovered that the ridgelet transform is exactly the application of a 1-dimensional wavelet transform to the slices of the radon transform,

$$\Re_f(\theta,t) = \int Rf(\theta,t)a^{-\frac{1}{2}}\psi\left(\frac{t-b}{a}\right)\,dt.$$

This means that the ridgelet transform can be implemented simply by implementing the radon and wavelet transforms [17, 18].

c. The inverse continuous ridgelet transform

By solving the ridgelet transform equation for $f(x)$, the inverse ridgelet transform can be obtained.

$$f(x) = \int_0^{2\pi} \int_{-\infty}^{\infty} \int_0^{\infty} |\Re_f(a,b,\theta)|^2 \frac{da}{a^3} db \frac{d\theta}{4\pi}$$

Of course this can also be implemented by performing an inverse one dimensional wavelet followed by the inverse radon transform.

d. The finite ridgelet transform

Since the ridgelet transform is going to be implemented using a radon transform, a change in the radon transform requires a change in the ridgelet. Fortunately, just by using the FRAT and a one dimensional wavelet, the finite ridgelet transform (FRIT) is created.

$$FRIT_f(k,m) = \langle FRAT_f(k,.), w_m^{(k)}(.) \rangle$$

Interestingly, it can be shown that the FRIT (when using the Haar wavelet) is orthonormal[21, 22].

e. The inverse finite ridgelet transform

The inverse finite ridgelet transform (IFRIT) is simply the inverse wavelet followed by the finite back projection algorithm.

4. Curvelet

a. The purpose of the curvelet transform

As mentioned before, the wavelet transform performs well on point singularities and the ridgelet performs well on line singularities. However, most photographic images contain few straight lines and points. If an image contains several objects, along their edges will be curve singularities. The curvelet transform is specifically designed to handle these curve singularities.

By using the curvelet transform, it has been shown that a large amount of an image can be recovered with very few of the curvelet coefficients[1, 8]. This makes it excellent for image compression. The transform has also been used to clean up noisy images with surprising accuracy[7].

b. The curvelet transform

The curvelet transform is a combination of all the previously mentioned transforms. It consist of breaking an image into subbands and then applying the ridgelet transform to each of these subbands. The steps in the algorithm are

1. Apply the "á trous" algorithm with a degree of three.

2. Apply the FRAT to each of the three subbands.

3. Apply the one dimensional Haar wavelet to the rows of the subbands.

In the end, three new images remain that store the data from the original image. Depending on the application these can be modified as needed.

c. The inverse curvelet transform

The inverse curvelet transform is the reverse of the previous.

1. Apply the inverse one dimensional Haar wavelet to the rows of the subbands.

2. Apply the FBP to the subbands.

3. Add all of the images together (as in the inverse "á trous").

Since each individual part of the curvelet transform is fully invertable, the curvelet transform does have the property of exact reconstruction.

CHAPTER III

R-FUNCTION HARDWARE DESIGN AND IMPLEMENTATION

A. Hardware Platform

In order to test the *R*-function architecture, it was programmed on a Xilinx XCV1000E FPGA. The input and output image were sized at 200 x 200. There was no particular reason for this size other than the fact that some size needed to be chosen. The FPGA was connected to a PC through a parallel port. The PC sent a clock and reset signal to the FPGA. The FPGA sent data to the PC one pixel at a time. In practical usage, the FPGA would be reprogrammed every time a new object was to be modeled. The entire design was done asynchronously in order to give maximum speed (at the cost of area).

B. Basic Shapes for Building Objects

In order to construct complex objects, a library of simpler functions is required. For this work, four useful functions were chosen. Obviously, any other functions would work, but only a limited amount can be implemented. With the functions chosen for this project, any two dimensional object can be modeled.

1. Circle (x and y second order)

The definition of the inside of a circle is given by the well known inequality

$$Ax^2 + By^2 + C \leq 0.$$

The asynchronous architecture for this is shown in Figure 2.

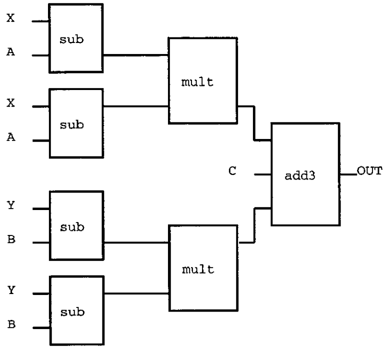

Fig. 2. Asynchronous circle calculation

2. Line (x and y first order)

A line is needed to define a half plane (the line is the point where the plane is cut in half). The inequality for the upper half of a plane is

$$mx + b - y \leq 0.$$

This architecture is shown in Figure 3.

3. Polynomial in x (x second order, y zeroth)

Although circles and lines are enough to draw a figure, having polynomials is convenient. The inequality for the "inner" portion of a second order polynomial in x is defined as

$$x^2 + xa + b \leq 0.$$

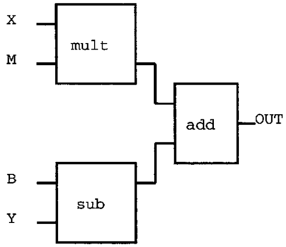

Fig. 3. Asynchronous line calculation

The FPGA design is displayed in Figure 4.

4. Polynomial in y (y second order, x zeroth)

Similar to the polynomial in x, a polynomial in y is also very useful. Of course, the inequality for the "inner" portion of a second order polynomial in y is defined just as it was in x

$$y^2 + ya + b \leq 0.$$

The hardware is also similar as shown in Figure 5.

C. *R*-functions

The *R*-functions are even easier to implement than the inequalities themselves. Conjunction and disjunction are created simply by implementing a max and min operator in hardware. Negation just changes the sign of its input. These operations are shown in Figure 6

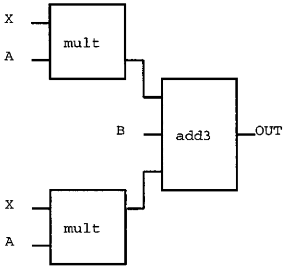

Fig. 4. Asynchronous polynomial in x calculation

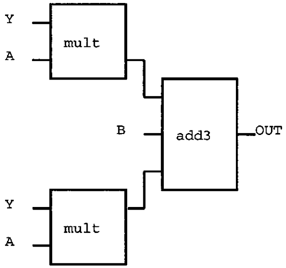

Fig. 5. Asynchronous polynomial in y calculation

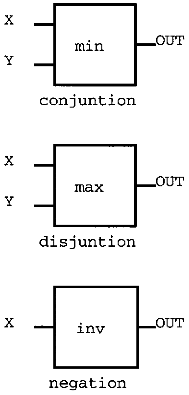

Fig. 6. *R*-function FPGA architecture

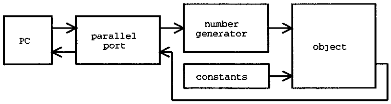

Fig. 7. FPGA architecture for R-function design

D. Combining the Pieces

Once the R-functions and the primitives have been implemented, the rest is easy. The first step is to connect all the primitives and R-functions in the proper way to produce the desired object. Once this has been done, the constants in the primitive equations need to be set. A seperate module was created to hold these constants. The last step is to generate the input coordinates. If the entire image is desired, the best solution is to simply have counters that go through every pixel and input them into the object equations. For some applications, not all pixels are needed, so a random number generator or some semi-random method may be preferred. This generator is controlled by an external clock and the results are returned through the parallel port. The result is a logic one if the pixel is within the image and a logic zero otherwise. These connections are shown in Figure 7.

CHAPTER IV

RIDGELET HARDWARE DESIGN AND IMPLEMENTATION

A. Hardware Platform

In order to test the ridgelet architecture, it was programmed on a Xilinx XCV1000E FPGA. The input image was sized at 17 x 17 to fill the prime number requirement of the FRAT. The FPGA was connected to a PC through a parallel port. Since the parallel port is limited to sending eight data bits and receiving four, a separate control block was added in the FPGA to convert the sixteen bits used in the transform to and from the size handled by the parallel port. The FPGA can be quickly reprogrammed to switch between the ridgelet transform and its inverse.

B. Architecture 1

This architecture focuses on parallelism. The time of the entire transform is equal to the time it takes to place the image onto the FPGA plus the time it takes to read it back. There is no time necessary for computation since this is done during the writing.

1. Wavelet

The Haar wavelet is basically a repeated sequence of adds, subtracts and shifts. The standard way to do this in software would require many steps. There would be one clock cycle for each add and subtract all throughout the process. Obviously this would be very time consuming.

In an FPGA, this can be sped up enormously. Since the entire transform is simply a large cascade of adders and subtractors, everything can be done in one cycle. The

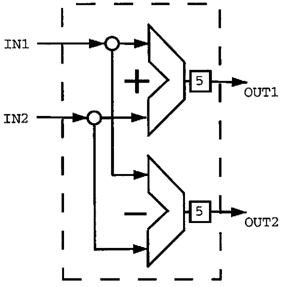

Fig. 8. Two input Haar wavelet

shifters can even be removed by simply connecting the output bus of one part one bit higher on the part it feeds.

a. Haar wavelet architecture

The basic step in the Haar transform is the averaging and differencing of two numbers. This simply means the two numbers have to be added, subtracted and divided by two. The divide by two is handled simply by dropping the lowest bit. The addition and subtraction are performed using a small adder in parallel with a small subtractor as shown in Figure 8.

By connecting several of these two input Haar wavelet boxes in a pyramid style, the Haar wavelet can be performed on any size one dimensional image desired. For this implementation, a sixteen input transform was needed. It is constructed as shown in Figure 9.

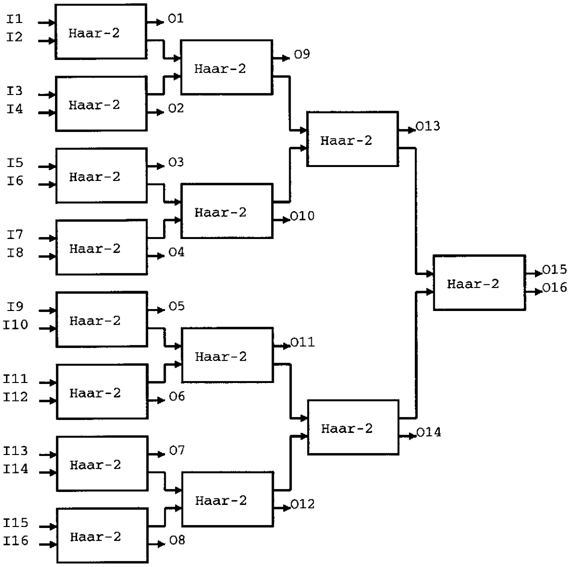

Fig. 9. Sixteen input Haar wavelet

b. Inverse Haar wavelet architecture

The basic step in the inverse Haar transform (Figure 10) is simply an addition and a subtraction. This is the same as the Haar transform without the divide by two. Because of this, the two input inverse Haar transform block in Figure 11 looks very similar to the Haar transform block in Figure 8.

As would be expected, the sixteen input inverse Haar transform looks like a reversed form of the pyramid for the Haar transform. By comparing Figures 9 and 10 the inverse can easily be seen. The inputs are simply fed backwards through the pyramid starting with just two inputs and doubling until the full sixteen is reached. Since this is all done at once, a noticeable speedup is achieved compared to doing each addition and subtraction separately.

2. Generic transform

In many transforms, each output pixel is simply the addition of a certain set of input pixels. Examples include the radon, back projection and Hough transforms. Because of this, a generic architecture was developed that would take in a square image and return an identically sized image with the output pixels equaling the additive combinations of the input pixels based on the look up tables. These look up tables could then be loaded with the proper values for whichever transform was desired.

a. Transform architecture

The architecture is basically a matrix of accumulators the size of the output image. Each input pixel is presented individually to the matrix, and the look up tables control the enable signals of the accumulators. This allows the values in the look up tables

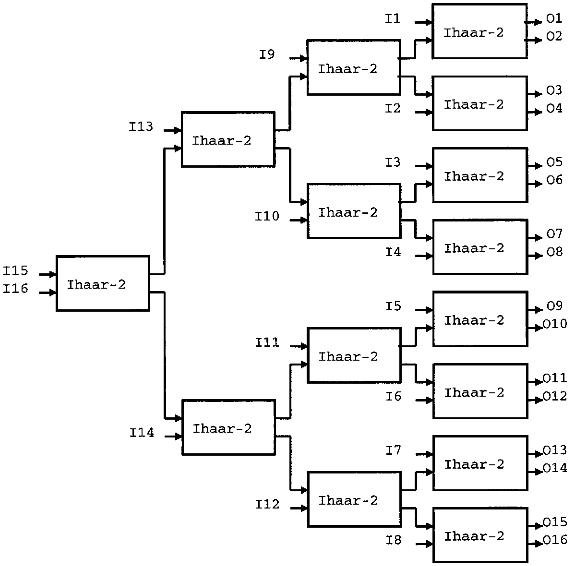

Fig. 10. Sixteen input inverse Haar wavelet

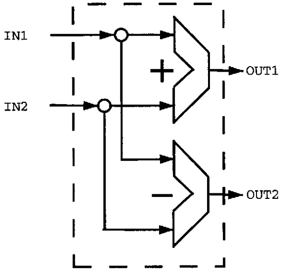

Fig. 11. Two input inverse Haar wavelet

to control which accumulators add in the pixel value as it is presented and which accumulators will ignore it. A control block consisting mainly of a counter is used to switch the output values of the look up table for each incoming pixel.

A simple block of four accumulators with the controlling look up table is shown in Figure 12. For this particular design, a 17 x 17 image was used. This means the forward transform would have an input of 17 x 17 and an output of 17 x 18. The inverse transform would have an input of 17 x 18 and an output of 17 x 17. Figure 13 shows the full accumulator matrix for all accumulators (289 for the inverse transform 306 for the forward). It should be understood that acc32 is simply 32 accumulators connected in the fashion shown in Figure 12.

When in output mode, the control block begins to switch the multiplexers so that the pixels in the accumulator matrix are sent out one at a time. The output is shifted right to normalize the pixel values for the radon and back projection transforms.

Part of the goal of the architecture is to take full advantage of the features the FPGA has to offer. The Virtex FPGA used in this implementation contains built

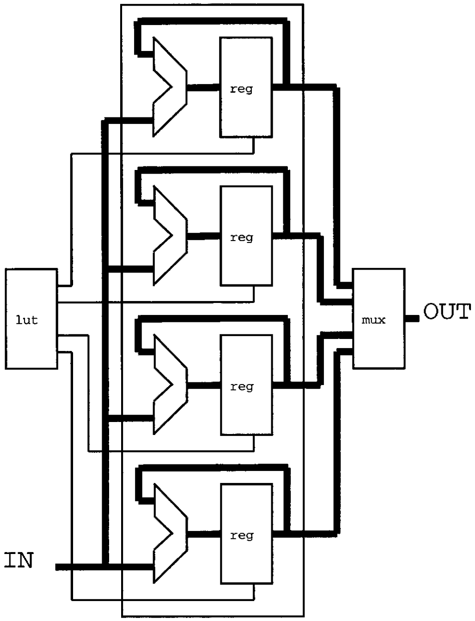

Fig. 12. Four accumulator block

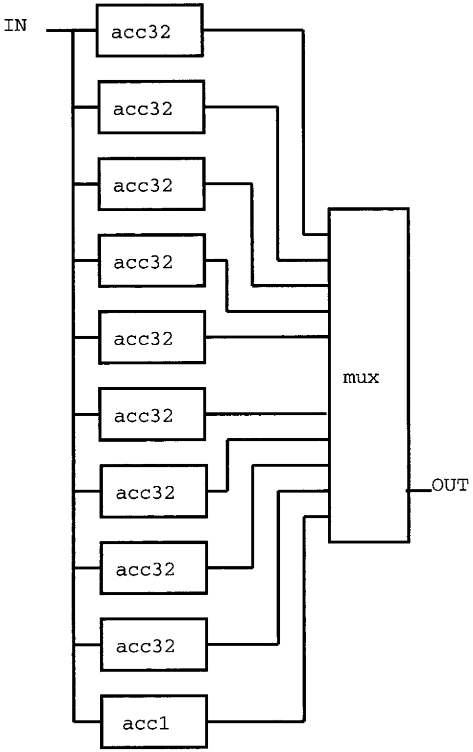

Fig. 13. Full 289 accumulator matrix

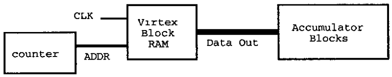

Fig. 14. Control system for the accumulator matrix

in Block RAM (BRAM) that can be pre-programmed with certain values. These BRAMs were used as the look up tables in the design. Their address was controlled by a counter as shown in Figure 14.

b. Finite radon transform data

The look up table data for the finite radon transform (FRAT) and inverse finite radon transform (IFRIT) was created using a C program. The program calculated a list of which pixels in the output image are affected by certain pixels in the input image. It then converted this data into a proper form to store in the FPGA Block RAM so that the accumulators would be properly controlled.

3. Ridgelet

As mentioned before, the ridgelet transform is simply the radon transform followed by the wavelet transform. To create the ridgelet on an FPGA, all that needs to be done is to connect its two component transforms together.

a. Finite ridgelet transform architecture

Once the radon and wavelet transforms have been implemented, the ridgelet is straightforward. Each output row of the radon is simply passed through the wavelet transform before it reaches the final output multiplexer. The primary advantage to this method is that it does not require any extra clock cycles to perform the ridgelet than it does

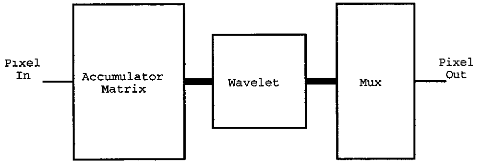

Fig. 15. Combining the radon and the wavelet to make the ridgelet

to perform the radon. The combination of the radon with the wavelet is shown in Figure 15.

b. Inverse finite ridgelet transform architecture

The inverse ridgelet requires that the inverse wavelet happen before the inverse radon. In this case, each input row is passed through the inverse wavelet before it reaches the inverse radon. Since pixels enter one at a time, there is a need for registers to store an entire row before the inverse wavelet takes place. This register setup is shown in Figure 16. Unfortunately, this adds extra clock cycles (equal to the number of pixels in a row). For a 17 x 17 pixel image, this changes the number of clock cycles from (289+289)=578 to (289+289)+17=595. The connections are shown in Figure 17.

C. Architecture 2

This architecture focuses on flexibility. Instead of the massive parallelism in architecture 1, a more compact and iterative approach is used. Each function is broken into its own module and a primary control block directs the flow of data. This allows for a higher clock rate, and blocks can be rearranged or added to increase parallelism as desired.

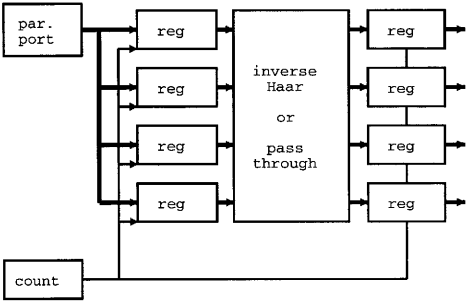

Fig. 16. Registers for inverse transform

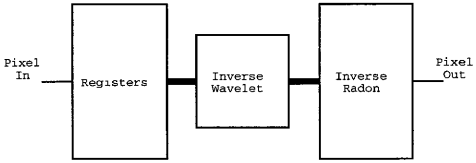

Fig. 17. Combining the inverse radon and the wavelet to make the inverse ridgelet

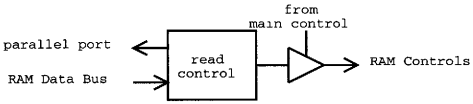

Fig. 18. RAM reading module for the second ridgelet architecture

1. Read and load blocks

This architecture is based on the Block RAM inside the FPGA. The image is loaded into the RAM, and then processed. When the transform is complete, the image is then returned to the host PC. In order for this to work, modules had to be developed to load data into the Block RAM and read the data from the Block RAM.

The read and load modules are fairly straightforward. The read module, pictured in Figure 18, consists of a block that connects the output of the RAM to the parallel port as well as providing some control signals to the RAM. A tristate buffer is used to control which module has access to the RAM's data and control busses.

The load module is very similar. As can be seen in Figure 19, the only difference in the read and load modules is that the load module connects to the input data of the RAM instead of the output data (this is for obvious reasons). Another tristate buffer is pictured, since all RAM inputs must have these to avoid signal contention.

2. Radon transform

The radon transform in this architecture is noticeably different than the one in the first. As mentioned before, all image data is stored in the Block RAM. The radon transform block uses a block called "frat calculator" to generate the list of which points in the input image affect which points in the output image. In the first architecture this was stored on chip (which takes quite a bit of storage space). After

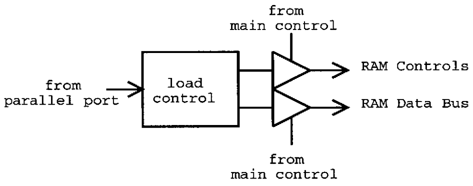

Fig. 19. RAM loading module for the second ridgelet architecture

the point list is calculated, "address generator" (Figure 20) converts the pixel values into RAM locations and switches the RAM address input. The accumulator is used to add the entire group of pixels chosen by "frat calculator" (Figure 21). The local control block organizes the flow of this process with input from the main controller. The diagram can be seen in Figure 22.

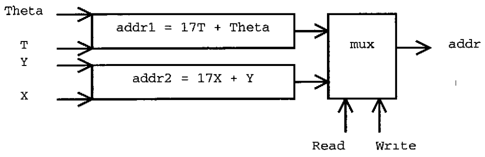

Fig. 20. Address generation module for the second ridgelet architecture

3. Wavelet transform

The wavelet transform in this design has much in common with that of the previous architecture. The usage of Block RAM makes an obvious difference, but the values are still read into a block of registers, transformed and then fed into a second block of

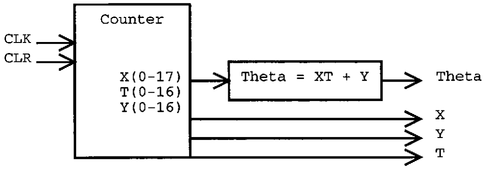

Fig. 21. FRAT calculation module for the second ridgelet architecture

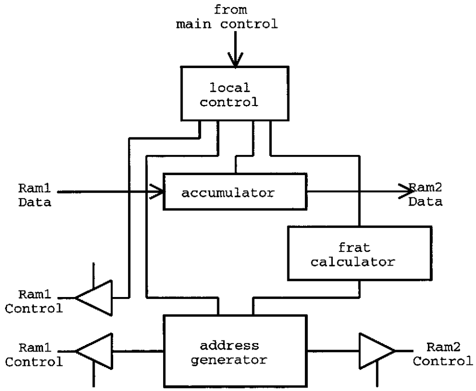

Fig. 22. Radon transform module for the second ridgelet architecture

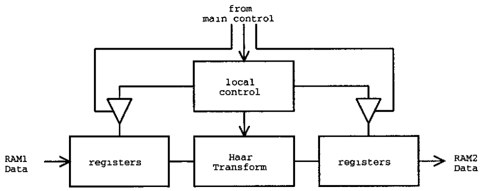

Fig. 23. Wavelet transform module for the second ridgelet architecture

registers so that they can be transferred into RAM again. Just as in the first architecture, this process is pipelined to increase speed. One change to the Haar transform block itself is the addition of a clock to aid pipelining. The flow is demonstrated in Figure 23.

4. Control block

Now that all the pieces have been described, the overall combination can be explained. Data comes in from the PC's parallel port. At this point it goes through the RAM loader and into the RAM. The control block decides how much data it will accept and uses the RAM loader to accomplish this. Next, the control block runs the radon transform block, and then the Haar transform (or their inverses). At the end, the data is transferred back to the parallel port through the RAM reader.

The advantage of this design is the ease of adding more modules. To double the speed of the radon transform, another radon block can be added with no other changes (except making the control block aware of the new addition). With the exception of the RAM loader and reader (which are limited by the size of the FPGA input bus), all modules can be duplicated to increase speed at the cost of area. The design (with only one of each module) is pictured in Figure 24.

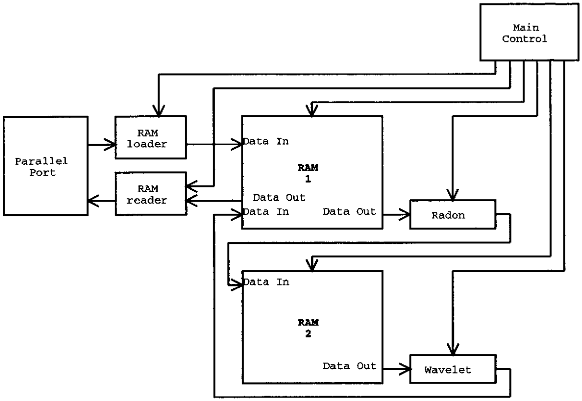

Fig. 24. Connections of the main control module for the second ridgelet architecture

CHAPTER V

CURVELET SOFTWARE DESIGN AND IMPLEMENTATION

A. Software Platform

The PC used was a Dual AMD Athlon MP 1800 running at 1.5 GHz. A C program running on this PC reads in a 289 x 289 image file (in PNM format) and runs the "á trous" decomposition on it. Next, it sends this image through the parallel port with the proper control signals for the FPGA. It then reads the image back and stores it in another PNM file. The PC side is identical for the ridgelet transform and its inverse.

B. Individual Programs

There are two separate activities that are performed on the PC. One is the "á trous" wavelet decomposition. The other is the control of the FPGA (which does the ridgelet transform).

1. "Á trous" decomposition

The first step in the codesigned curvelet transform is the PC implementation of the "á trous" wavelet decomposition. The process begins with a 289 x 289 image on the PC. This file is stored in PNM format due to the ease of converting between PNM images and human readable number matrices. When the program is run, it reads in this image and performs the "á trous" decomposition. This leaves 3 new 289 x 289 images. The steps that this program follows are best shown in the flowchart in Figure 25. The inverse is also performed by adding the three images together to produce the original.

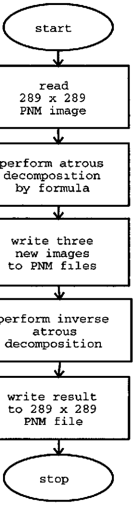
start
read 289 x 289 PNM image
perform atrous decomposition by formula
write three new images to PNM files
perform inverse atrous decomposition
write result to 289 x 289 PNM file
stop

Fig. 25. "Á trous" decomposition program flowchart

2. FPGA control

Since the FPGA acts as a coprocessor, there needs to be a way for the processor within the PC to control it. For a high speed implementation, a PCI bus or some form of direct connection would be the best. For demonstration purposes, this design uses the parallel port. A 289 x 289 image is taken in and broken into 289 separate 17 x 17 images. These images are then sent one by one to the FPGA and the ridgelet transform or its inverse is performed. For the inverse, the image would be 306 x 306 and would be broken into 17 x 18 blocks. A summary is shown in Figure 26.

C. Combined Overall Process

Though all the pieces have been explained, it may not be obvious how the entire process fits together. Figure 27 shows the flow from original image to the final. It should be noted that this shows only the transform and inverse transform. In practical use, it would be more likely to make some modification to the image (such as thresholding) in between transforming it and inverse transforming it.

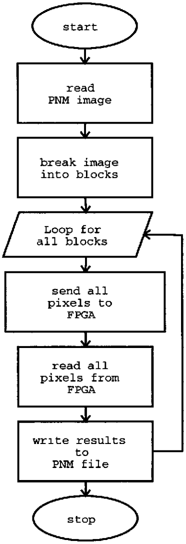

Fig. 26. FPGA control program flowchart

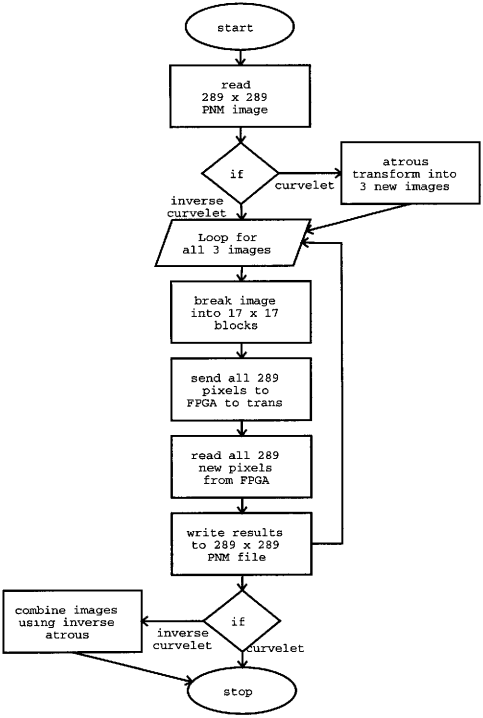
start
read 289 x 289 PNM image
if
curvelet
atrous transform into 3 new images
inverse curvelet
Loop for all 3 images
break image into 17 x 17 blocks
send all 289 pixels to FPGA to trans
read all 289 new pixels from FPGA
write results to 289 x 289 PNM file
if
combine images using inverse atrous
inverse curvelet
curvelet
stop

Fig. 27. Overall process flowchart

CHAPTER VI

RESULTS

A. *R*-function Performance

1. A practical example

When Vladimir Rvachev originally proposed R-functions, he came up with the simple example of a chess pawn to demonstrate their power. This same pawn object was implemented in the FPGA to show the performance of the system described in this thesis.

The pawn consist of five circles and two polynomials. They are given by the following inequalities:

$$
\begin{aligned}
&(D_1) \quad 1 - x^2 - (y-7)^2 \geq 0;\\
&(D_2) \quad 4 - x^2 - (y-7)^2 \geq 0;\\
&(D_3) \quad 64 - (x-8)^2 - (y-7)^2 \geq 0;\\
&(D_4) \quad 64 - (x+8)^2 - (y-7)^2 \geq 0;\\
&(D_5) \quad 4 - x^2 - (y-4)^2 \geq 0;\\
&(D_6) \quad 9 - x^2 \geq 0;\\
&(D_7) \quad (7-y)y \geq 0;
\end{aligned}
$$

These combine according to the following equation:

$$D = D_1 \vee (D_2 \wedge D_5) \vee (\overline{D_2} \wedge \overline{D_3} \wedge \overline{D_4} \wedge D_6 \wedge D_7)$$

When implemented in the FPGA, this object gave speed and area results as shown in Table III. The area is shown in terms of SLICEs (the units inside a Xilinx FPGA), and in percentage of the total area of a XCV1000E chip. The objects per second value describes how many 200 x 200 pixel pawns can be drawn per second. The visual result from the FPGA can be seen in Figure 28.

Table III. Speed and Area of Pawn R-functions

Max clock Speed	65 MHz
Area (in SLICEs)	3673
Area (in %)	29 %
200 x 200 pixel frames per second	1625
Max size of image for 30 frames/sec	1471 x 1471

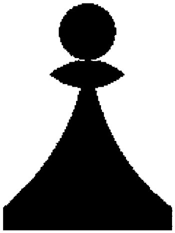

Fig. 28. Pawn generated with R-functions

2. Tradeoffs

After the pawn was created, some experiments were run to see how the performance was affected by the addition of primitives to the object. Figures 29 and 30 show how the speed and area of the design are affected by the addition of more lines and circles. Figure 29 shows the size for a design with one line (or circle), two lines (or circles), etc. Figure 30 shows similar data for speed. It is worth noting that due to the Xilinx routing tools, the speed can fluctuate (depending on how well the tool routed), but overall speed is not largely affected after the first couple of primitives. This means that although more primitives take more area, their affect on speed is not consistent enough to worry about.

Based on this data, it can be seen that circles take up approximately twice the FPGA area as lines do. It can also be seen that the maximum size object on a XCV1000E is one containing 20 circles, 40 lines, or some combination thereof. Due to the number of multipliers, second order polynomials are the same size as circles.

B. Ridgelet Architecture 1 Performance

The parallel port described in the design is just for the test setup. The speed of that port is much lower than the speed of the FPGA. The entire ridgelet transform takes approximately 1.6 seconds with the parallel port. In a real implementation, something similar to a PCI bus would be a much better communication system. The Xilinx synthesis tools do give the performance of any compiled design. Based on those numbers, we can determine the speed of the transform when the PCI (or some other high speed bus) is used. The FRIT architecture would take 289x289 send cycles plus another 289x306 for reading. This would all be at a clock rate of 33MHz. The IFRIT would have (306+17)x289 send cycles and 289x289 for the read. The maximum clock

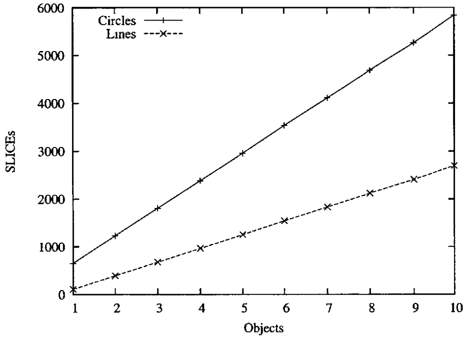

Fig. 29. Area data for R-function lines and circles

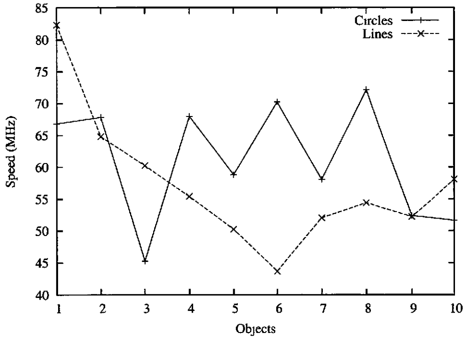

Fig. 30. Speed data for R-function lines and circles

Table IV. Common Image Sizes and Numbers of Pixels

Image Size	Number of Pixels
800 x 600	480000
1024 x 768	786432

speed for the inverse is 18MHz.

By multiplying these numbers out, the total time for the FRIT is 5.2 ms. Total for the inverse FRIT is 9.8 ms. This means the entire process takes 15 ms. For a comparison, this same process took 1.5s on a dual Athlon MP 1800. That gives a speed increase factor of 100.

Figure 31 shows the completion time of the first ridgelet architecture for various image sizes. The y-axis shows the amount of time (in milliseconds) for the entire transform to take place. The x-axis shows the number of pixels in the image. The graph shows the results for both the finite radon transform (FRIT) and its inverse (IFRIT). The maximum size of pixels that can be processed in 33 ms (30 frames per second) in the case of FRIT is 534285. For the IFRIT it is 283333. For reference, a short list of common image sizes and the number of pixels in them can be seen in Table IV.

C. Ridgelet Architecture 2 Performance

The second architecture has several more cycles, but the clock rate is higher. For the 289x289 image, there are 17x18x289 read and 17^2x289 write cycles. For the entire radon transform (or its inverse), 17^2x18x289 clock cycles are necessary. For the Haar transform, an additional (17+4)x17x289 clock cycles are needed. The time for this

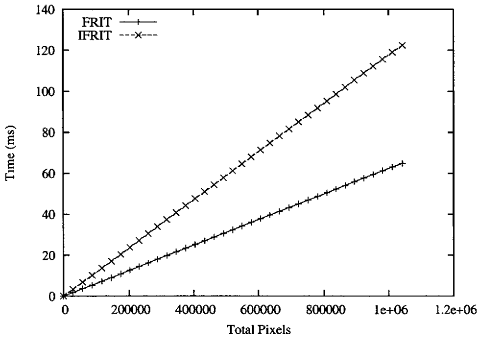

Fig. 31. Speed data for the first ridgelet architecture

entire process, at its speed of 60MHz, is 30ms (34 frames per second). This is almost six times the first forward architecture and triple the first inverse architecture. The advantage is in size. This architecture is one eighth the size of the first ridgelet architecture. Another advantage is the fact that some blocks can be repeated to increase speed (or pipelining can be utilized). Due to the large amount of clock cycles used by the radon transform relative to the Haar, it turns out not to be very useful to increase the number of Haar blocks, but increasing the number of radon blocks has a noticeable effect (up to a point).

The reconfigurability advantages of the second ridgelet architecture are shown in Figure 32. More blocks of the radon or Haar type can be added to increase the speed of the transform at the cost of area. It should be obvious that increasing the number of radon blocks has a much larger affect on the speed than increasing the number of Haar blocks. The y-axis shows the time (in milliseconds) that it takes to

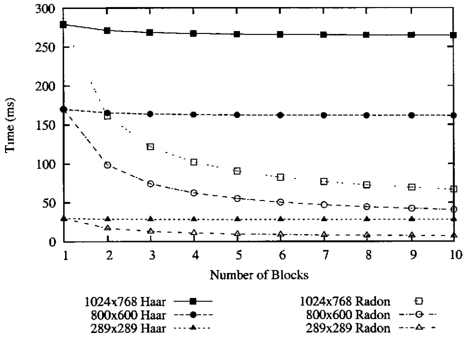

Fig. 32. Speed data for the second ridgelet architecture

complete the entire FRIT or IFRIT. The x-axis shows the number of radon or Haar blocks used in the FPGA. Each line shows the size of the image and the type of block that was repeated inside the FPGA. The lines with darkened shapes represent an architecture with one radon block and the number of Haar blocks shown on the x-axis. The lines with white shapes represent an architecture with one Haar block and the number of radon blocks shown on the x-axis. These numbers can be compared to rough estimations of the time to complete the same transform performed on a Texas Instruments DSP TMS320VC5502 running at 200MHz. These estimates are 1.35s (1024 x 768), 828ms (800 x 600) and 114ms (289 x 289).

It can also be shown that the addition of radon blocks has a fairly small effect on the overall area taken within the FPGA. The addition of Haar blocks has a much larger effect. Combined with the previous speed results, this works out very well since the radon is the block that should be added anyway. The results of this experiment

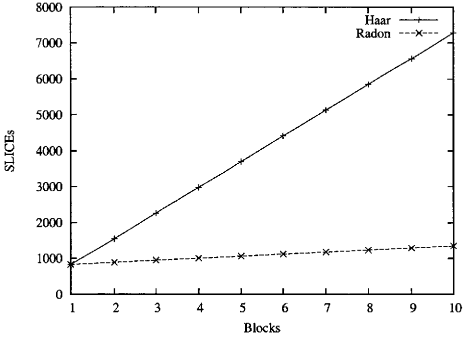

Fig. 33. Area data for the second ridgelet architecture

are shown in Figure 33 (these are the same for all image sizes).

Table V shows the minimum amount of each of the four types of blocks needed to reach video quality imaging (30 frames/sec), for each of the given image sizes. The last column shows the amount of slices in the FPGA what would be taken up by that particular design.

One of the previously stated advantages of FPGAs is their in field reprogrammability. From Figure 32, it can be seen that as more blocks are added, the time decreases asymptotically to a certain point. This point varies with the number of FPGAs used. In a system consisting of several FPGAs that could be reconfigured, the more FPGAs used, the lower the asymptotical point would become. This is shown in Figure 34. The y-axis shows the asymptotical minimum time, and the x-axis shows the number of FPGAs. The lines are the same as Figure 32. From Figure 34, it can be seen that two FPGAs is twice as fast as one, three FPGAs are thrice as fast as one, etc.

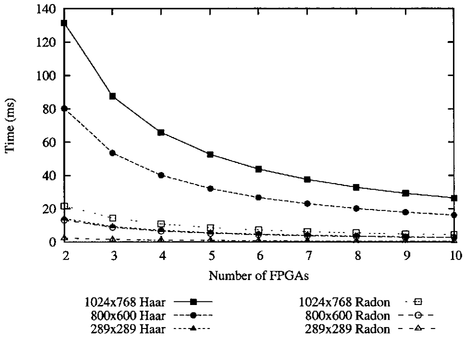

Fig. 34. Speed data for multiple FPGAs with the second ridgelet architecture

Table V. Minimum Ridgelet 2 Blocks Needed for Video Quality

	radon blocks	Haar blocks	load RAM	read RAM	total slices
289 x 289	1	1	1	1	828
800 x 600	20	1	1	1	1968
1024 x 768	20	2	2	2	2736

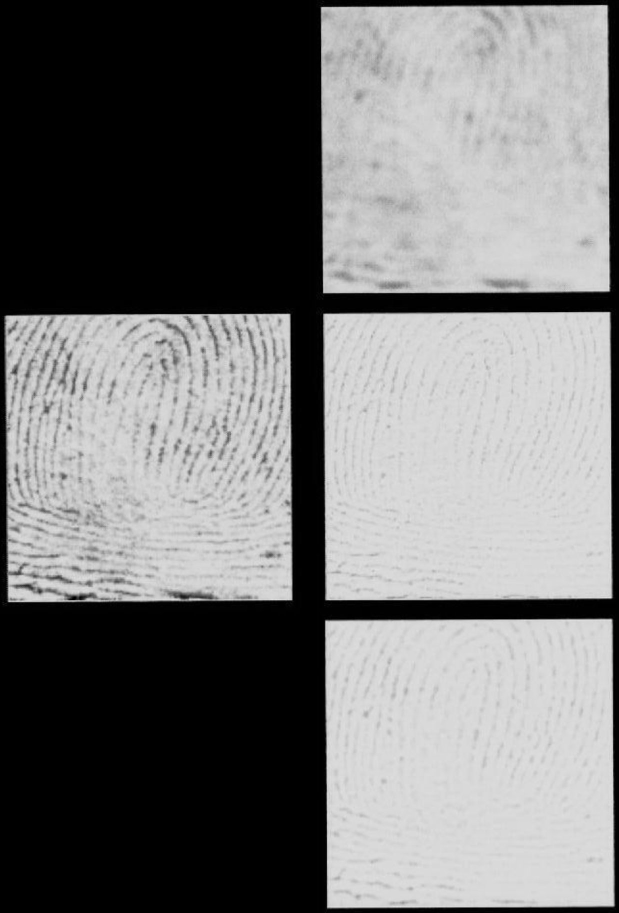

Fig. 35. A fingerprint image and its subbands

D. Software “Á Trous” Performance

The part of the curvelet transform that was implemented in software was the “’a trous” algorithm. On the dual Athlon machine, the transform took .77s, and its inverse took .04s. The visual results of the image being broken into three subbands can be seen in Figure 35. The results after the inverse finite ridgelet transform and the recombination are in Figure 36.

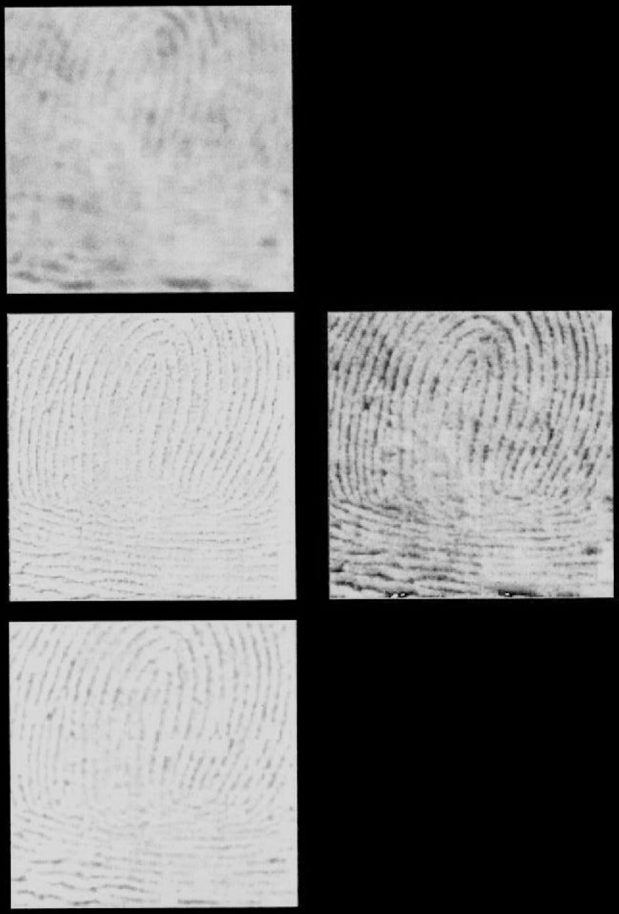

Fig. 36. Subbands after FRIT and IFRIT and the recombination

CHAPTER VII

CONCLUSIONS AND FUTURE WORK

A. Summary

1. *R*-functions

- An asynchronous parallel architecture was developed to represent the equations of a circle, line and polynomial as well as the *R*-functions for conjunction, disjunction and negation.
- These were connected in the proper order to create the image of a pawn on the FPGA, and speed and area measurements were taken. This design took 3673 slices (29% of the Virtex XCV1000E) and ran at 65MHz. It took 0.62ms to draw the entire image.
- A tradeoff study was done comparing size and speed of the design when more circles and lines were added to the image to be modeled. Circles take twice the space of lines, and on the XCV1000E, a total of 40 lines, 20 circles or some combination thereof would fit.

2. Curvelet

a. Ridgelet 1 (Full Parallel Scheme)

- An array of accumulators was put on the FPGA. This array is controlled by data calculated by a C program and stored in the FPGA Block RAM to perform the finite radon transform and its inverse.
- An array of adders and subtractors was put together in the proper order to create the Haar and inverse Haar wavelet transforms. These were connected

inline with the accumulators that due the finite radon transform and its inverse.

- For a 17x17 image, the finite ridgelet transform architecture uses 306 accumulators and takes in an image one pixel at a time. These pixels are passed into the accumulators (controlled by the Block RAM) creating a 17x18 image inside the FPGA. Once all pixels have been passed in, they are passed out one at a time (controlled by multiplexers) returning the transformed image.

- The inverse finite ridgelet transform is similar, except that it takes in a 17x18 image and has 289 accumulators so that it can return a 17x17 image. The architecture is valid for any square block size with the side equal to a prime number that will fit on an FPGA.

b. Ridgelet 2 (Shared Module Architecture)

- Separate modules were created to perform the many activities of the transform. One is for loading an image into RAM and one is for reading an image from RAM. Two other blocks perform the finite radon transform (or its inverse) and the Haar wavelet transform (or its inverse).

- In the radon transform, only certain pixels from the input image affect certain pixels in the output image. In the first architecture, this data is pre-calculated and stored in RAM.. In this architecture, it is calculated during operation by a module specifically for this purpose.

- Unlike the first architecture, this one is not limited to a 17x17 block. It is limited only by the size of the RAM on the FPGA. Obviously making the image smaller would increase the speed of the transform.

- The other primary advantage to this architecture is the ability to put more blocks of certain types on the FPGA in order to increase speed at the cost of area.

- A tradeoff study was done for this architecture showing what the speed area tradeoff would be for various image sizes. It was discovered that large speed increases were achieved with little area costs when the radon block was replicated. Replicating the Haar block gave little speed increase at a fairly large area overhead. Details are given in the results chapter.
- Based on the data collected, to reach video quality imaging with the first architecture, 4792 slices would be needed for any image size 543 x 543 for the first architecture. The second architecture would use the exact same area to do a 1132 x 1132 image for either the FRIT or IFRIT in the same amount of time.

c. Software

- The trous algorithm was implemented in software to complete the curvelet transform.
- The ridgelet was completed in software in order to make speed comparisons.
- A parallel port control was written in software (as well as a small addition to the hardware) to demonstrate the transforms on the FPGA.

B. Conclusions

Based on the results presented in this thesis it is quite obvious that the FPGA is far more efficient (in speed and area) at implementing R-functions and the various transforms. It is also clear that R-functions and the curvelet transform are two of the most powerful methods of modeling and processing images. The use of these two together is shown to make a powerful team that can be used to accomplish tasks that have previously been unimplementable.

C. Future Work

There are several future possibilities that can be based on this work. One improvement could involve replacing the Haar wavelet with one more suited to certain types of images. Another could involve replacing the finite radon transform with the digital radon transform involving the fast fourier transform and its inverse.

Besides improvements to the design, other developments can be built on top of this. The primary ones are the hyperspectral imager and equation solver mentioned in the introduction. Both of these devices are things that could never have been done or were extremely costly in money, human time and processing time. With the new techniques discussed in this thesis the possibility of creating these devices and similar ones can be seen.

REFERENCES

[1] E. Candès and D. Donoho, "Curvelets: A surprisingly effective nonadaptive representation of objects with edges," Unpublished manuscript. Available at http://www-stat.stanford.edu/~donoho/Reports/1999/curveletsurprise.pdf, 1999.

[2] E. Candès and D. Donoho, "Ridgelets: A key to higher-dimensional intermittency? " Unpublished manuscript. Available at http://www-stat.stanford.edu/~donoho/Reports/1999/RoySoc.pdf, 1999.

[3] V.I. Rvachev, *Theory of R-functions and Some Applications (in Russian)*, Kiev: Naukova Dumka, Ukraine, 1982.

[4] V. Shapiro, "Maintenance of geometric representations through space decompositions," *International Journal of Computational Geometry and Applications*, vol. 7, no. 1/2, pp. 21–56, 1997.

[5] V. Shapiro, "Theory of R-functions and applications: A primer," Tech. Rep. CPA88-3, Cornell University Mechanical Engineering Department, New York, New York, 1988.

[6] V. Shapiro and I. Tsukanov, "Implicit functions with guaranteed differential properties," in *SOLID MODELING '99*, 1999, pp. 258–269.

[7] J. Starck, E. Candès, and D. Donoho, "The curvelet transform for image denoising," Unpublished manuscript. Available at http://citeseer.nj.nec.com/starck00curvelet.html, 2000.

[8] D. Donoho and M. Duncan, "Digital curvelet transform: Strategy, implementation and experiments," Unpublished manuscript. Available at http://citeseer.nj.nec.com/donoho99digital.html, 1999.

[9] E. Stollnitz, T. DeRose, and D. Salesin, "Wavelets for computer graphics: A primer, part 1," *IEEE Computer Graphics and Applications*, vol. 15, no. 3, pp. 76–84, 1995.

[10] M. Feil and A. Uhl, "Real-time image analysis using wavelets: The'á trous' algorithm on MIMD architectures," *Real-Time Imaging IV*, vol. 3645, pp. 56–65, 1999.

[11] R. K. Gupta and G. De Michelli, "Hardware-software cosynthesis for digital systems," *IEEE Design and Test of Computers*, vol. 10, no. 3, pp. 29–41, 1993.

[12] R. K. Gupta, C. Coelho Jr., and G. De Micheli, "Program implementation schemes for hardware-software systems," *IEEE Computer*, vol. 27, no. 1, pp. 48–55, 1994.

[13] J. Wilberg, "Codesign for real-time video applications," Ph.D. dissertation, Brandenburg University of Technology Cottbus, Brandenburg, Germany, 1996.

[14] R. Janka and L. M. Wills, "A novel codesign methodology for real-time embedded COTS multiprocessor based signal processing systems," in *CODES '00*, 2000, pp. 157–162.

[15] E. Stollnitz, T. DeRose, and D. Salesin, "Wavelets for computer graphics: A primer, part 2," *IEEE Computer Graphics and Applications*, vol. 15, no. 4, pp. 75–85, 1995.

[16] E. Candès, "Ridgelets and their derivatives: Representation of images with edges," Unpublished manuscript. Available at http://citeseer.nj.nec.com/384752.html, 1999.

[17] D. Donoho, "Ridge functions and orthonormal ridgelets," Unpublished manuscript. Available at http://citeseer.nj.nec.com/145161.html, 1999.

[18] E. Candès, "Monoscale ridgelets for the representation of images with edges," Unpublished manuscript. Available at http://www-stat.stanford.edu/~emmanuel/papers/Monoscale.pdf, 1999.

[19] E. Candès and D. Donoho, "Curvelets and curvilinear integrals," Unpublished manuscript. Available at http://citeseer.nj.nec.com/383973.html, 1999.

[20] F. Matús and J. Flusser, "Image representations via a finite radon transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 10, pp. 996–1006, 1993.

[21] M. Do and M. Vetterli, "The finite ridgelet transform for image representation," Unpublished manuscript. Available at http://citeseer.nj.nec.com/519266.html, 2001.

[22] M. Do and M. Vetterli, "Orthonormal finite ridgelet transform for image compression," Unpublished manuscript. Available at http://citeseer.nj.nec.com/do00orthonormal.html, 2000.

[23] E. Candès, "Ridgelets: Estimating with ridge functions," Unpublished manuscript. Available at http://citeseer.nj.nec.com/386438.html, 1999.

VITA

John L. Wisinger Jr.

3712 Dauterive Dr.

Chalmette, LA 70043

John L. Wisinger Jr. was born in New Orleans, LA in 1978. He was raised in the nearby suburb Chalmette and went to Brother Martin High School. In 1996, he graduated and headed to College Station, Texas to begin his life as a computer engineer (and become the butt of so many Aggie jokes).

At Texas A&M, John began work on a Bachelor of Science degree in Computer Engineering in the Department of Electrical Engineering. During this endeavor, John developed an interest in embedded systems design. The ever growing collection of John's new found wisdom and knowledge shone brightly forth in the design of IDAPIC, an Internet data acquisition device that served as his senior design project.

In May of 2000, John entered Reed Arena and, four hours later, walked out the proud owner of a diploma. In his effort to postpone entrance into the real world, he returned to A&M that fall to begin work on a master's degree. There he worked on different embedded systems designs as well as being a teaching assistant (and sometimes a teacher) for senior design courses. He also found a home on the university's solar race car team.

This thesis is a combination of the knowledge that John has learned and developed in his many years of schooling. He hopes that by the time this thesis is published, he will have found a happy life designing little computers to make the world a better place (or at least make it easier to watch television and get some munchies from the fridge).