NORTHEASTERN UNIVERSITY

Graduate School of Engineering

Thesis Title: Parallel-Beam Backprojection: an FPGA Implementation Optimized for Medical Imaging

Author: Srdjan Coric

Department: Electrical and Computer Engineering

Approved for Thesis Requirement of the Master of Science Degree

_____ _____

Thesis Advisor Date

_____ _____

Thesis Reader Date

_____ _____

Thesis Reader Date

_____ _____

Department Chair Date

Graduate School Notified of Acceptance:

_____ _____

Director of the Graduate School Date

Copy Deposited in Library:

_____ _____

Reference Librarian Date

PARALLEL-BEAM BACKPROJECTION: AN FPGA IMPLEMENTATION OPTIMIZED FOR MEDICAL IMAGING

A Thesis Presented

by

Srdjan Coric

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements

for the degree of

Master of Science

in

Electrical Engineering

in the field of

Computer Engineering

Northeastern University

Boston, Massachusetts

December 2001

Acknowledgements

I would like first to thank my advisors Prof. Miriam Leeser and Prof. Eric Miller for their friendship, support and guidance. The ideas provided by Prof. Eric Miller and Prof. Miriam Leeser were instrumental in making theoretical foundation for this work. They consistently contributed useful insights and helped me develop my research in the right direction.

This thesis would not have been possible without Mercury Computer Systems, Inc. which initiated the project and funded my research. I owe special gratitude to Marc Trepanier and Iain Goddard from Mercury for their expertise and constructive suggestions that helped guide this research to a successful end.

I would also like to thank my colleagues from the Rapid Prototyping Group at Northeastern University for creating friendly atmosphere that made long hours in our lab more enjoyable.

Thanks to Heather Quinn for allowing me to use "Field Programmable Gate Arrays" section from her thesis.

A special thanks goes to my parents and my brother for their support and constant encouragement.

Abstract

Medical image processing in general and computerized tomography (CT) in particular can benefit greatly from hardware acceleration. This application domain is marked by computationally intensive algorithms requiring the rapid processing of large amounts of data. To date, reconfigurable hardware has not been applied to this important area. For efficient implementation and maximum speedup, fixed-point implementations are required. The associated quantization errors must be carefully balanced against the requirements of the medical community. Specifically, care must be taken so that very little error is introduced compared to floating-point implementations and the visual quality of the images is not compromised.

An FPGA implementation of the parallel-beam backprojection algorithm used in CT for which all of the above requirements are met is presented in this thesis. A number of quantization issues arising in backprojection are explored with concentration on minimizing error while maximizing efficiency. Presented implementation shows significant speedup over software versions, and is more flexible than an ASIC implementation. Furthermore, this FPGA implementation can easily be adapted to both medical sensors with different dynamic ranges as well as tomographic scanners employed

in a wider range of application areas including nondestructive evaluation and baggage inspection in airport terminals.

Contents

List of Figures

Chapter 1

Introduction

Presently, digital signal processing (DSP) algorithms are most common candidates for hardware implementation. This trend is driven by constant expansion of the DSP applications, such as multimedia computing and high-speed wired and wireless communications, and also by increasing demand for smaller and faster devices. Up to recently, the implementation mechanisms of choice for many DSP applications were the general-purpose processors, application-specific integrated circuits (ASICs), programmable digital signal processors (PDSPs) or multiprocessing systems. As the high capacity field-programmable gate arrays (FPGAs) have emerged, increasingly new system implementations based on reconfigurable computing are being considered.

The are several reasons that render reconfigurable devices to stand out among other implementation choices. The most obvious one is their ability to be dynamically reconfigured and thus accommodate the need of various DSP algorithms to adapt to changing data sets and computing conditions, offering at the same time compromise

between the high computational speed of ASICs and the programmability of PDSPs. Furthermore, the fine-grained parallelism inherent to FPGAs is the main feature that can be used to capture data parallelism found in many DSP functions. Equally important are the facts that FPGAs provide simpler design cycle than ASIC based implementations making time to market shorter, and have greater customization capabilities than PDSPs making hardware more efficient.

Computerized tomography, as one of DSP or more specifically image processing applications, can also benefit from the features of reconfigurable devices. Image construction algorithms used in tomography require a large amount of data to be processed. In the case of medical imaging, this processing has to be accurate and fast. A high-capacity FPGA device configured with the hardware architecture presented in this thesis, which implements parallel-beam backprojection image reconstruction algorithm, can satisfy these requirements. Furthermore, it can be easily reconfigured to accommodate different geometrical properties and data acquisition methods of various scanners applied in medical imaging or other nondestructive evaluation applications.

1.1 Thesis Outline

The outline of the reminder of the thesis is as follows.

Chapter 2 describes the theory of line integrals and parallel-beam backprojection algorithm. It further expands on this by describing practical approaches for

implementation of reprojection and backprojection procedures. A review of relevant information regarding fixed-point numbers is also given. A brief outline of reconfigurable hardware technology and the design process for implementation of algorithms using this technology follows. The final section in this chapter gives an overview of the related work.

Chapter 3 describes procedures applied for finding proper quantizations of relevant data sets used for the implementation of backprojection. A hardware architecture implementing parallel-beam backprojection is presented. This is accompanied with the discussion of different methods for parallelism extraction and their trade-offs.

Chapter 4 presents results showing comparisons of floating-point and quantized reconstructions for different quantization schemes. This chapter also contains a summary of performance of the implemented hardware.

Chapter 5 draws conclusions and gives suggestions for future work.

Chapter 2

Background

Tomography refers to the process that generates a cross-sectional or volumetric image of an object from a series of projections collected by scanning the object from many different directions [3]. It is applied in diagnostic medicine, baggage inspection, industry, astronomy, and geology. Projection data acquisition can utilize X-rays, magnetic resonance, radioisotopes, or ultrasound. The discussion presented here pertains to the case of two-dimensional X-ray absorption tomography. In this type of tomography, projections are obtained by a number of sensors that measure the intensity of X-rays travelling through a slice of the scanned object (see Figure 2.1). The radiation source and the sensor array rotate around the object in small increments. One projection is taken for each rotational angle. The image reconstruction process uses these projections to calculate the average X-ray attenuation coefficient in cross-sections of a scanned slice. If different structures inside the object induce different levels of X-ray attenuation, they are discernible in the reconstructed image.

The most commonly used approach for image reconstruction from dense projection data (many projections, many samples per projection) is filtered

backprojection (FBP). Depending on the type of X-ray source, FBP comes in parallel-beam and fan-beam variations [3]. In this thesis, the focus is on parallel-beam backprojection, but methods and results presented here can be extended to the fan-beam case.

FBP is a computationally intensive process. For an image of size $n \times n$ being reconstructed with n projections, the complexity of the backprojection algorithm is $O(n^3)$. There is another algorithm whose complexity is on the order of $n^2 \log_2 n$ [6]; however, its suitability for hardware implementation has not yet been investigated. Image reconstruction through backprojection is a highly parallelizable process. Such applications are good candidates for implementation in FPGA devices since they provide fine-grained parallelism and the ability to be customized to the needs of a particular implementation. In this thesis backprojection was implemented by making use of these principles and shown approximately 20 times speedup over a software implementation on a 1GHz Pentium. Presented architecture can easily be expanded to newer and larger FPGA devices further accelerating image generation by extracting more data parallelism.

Another difficulty of implementing FBP is that producing high-resolution images with good resemblance to internal characteristics of the scanned object requires that both the density of each projection and their total number be large. This represents a considerable challenge for hardware implementations, especially in terms of required data transfer rates. Therefore, it can be beneficial for fixed-point implementations to optimize the bit-width of a projection sample to the specific needs of the targeted

application domain. This is shown for medical imaging, which exhibits distinctive properties in terms of required fixed-point precision.

Finally, medical imaging requires high precision reconstructions since visual quality of images must not be compromised. Here, special attention has been paid to this requirement by carefully analyzing the effects of quantization on the quality of reconstructed images. One of the findings in this thesis is that a fixed-point implementation with properly chosen bit-widths can give high quality reconstructions and, at the same time, make hardware implementation fast and area efficient. Conducted quantization analysis investigates algorithm specific and also general data quantization issues that pertain to input data. Algorithm specific quantization deals with the precision of spatial address generation including the interpolation factor, and also investigates bit reduction of intermediate results for different rounding schemes.

2.1 Line Integrals and Projections

A line integral represents the integral of some parameter of the object along a line. A typical example is the attenuation of X-rays as they propagate through biological tissue. In this case, the object is modeled as a two-dimensional (or three-dimensional) distribution of the X-ray attenuation constant and a line integral represents the total attenuation suffered by a beam of X-rays as it travels in a straight line through the object.

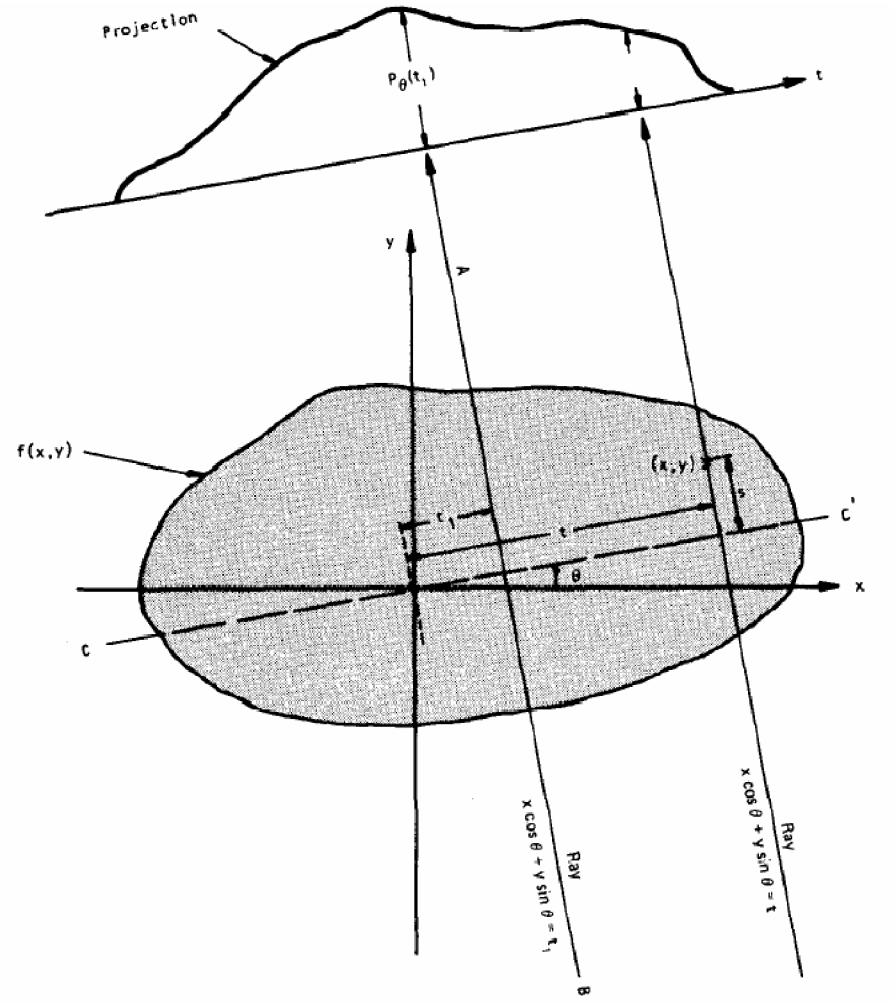


Figure 2.1: An object, *f*(*x*, *y*), and its projection, $P_\theta(t)$, are shown for an angle of *θ*. (From [Kak 87].)

Figure 2.1 shows the coordinate system used to describe line integrals and projections. In this example the object is represented by a two-dimensional function *f*(x, *y*) and each line integral by the (θ, *t*) parameters.

The equation of line AB in Figure 2.1 is

$$x\cos\theta + y\sin\theta = t \tag{2.1}$$

and the line integral along this line is defined as

$$P_\theta(t) = \int_{(\theta, t)\,\text{line}} f(x, y)\, ds\,. \tag{2.2}$$

Using a delta function, this can be rewritten as

$$P_\theta(t) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y)\delta(x\cos\theta + y\sin\theta - t)\, dx\, dy\,. \tag{2.3}$$

The function $P_\theta(t)$ is known as the Radon transform of the function $f(x, y)$.

A projection is formed by combining a set of line integrals. The simplest projection is a collection of parallel ray integrals as is given by $P_\theta(t)$ for a constant θ. It could be measured, for example, by moving an X-ray source and detector along parallel lines on opposite sides of an object.

Another type of projection is possible if a single source is placed in a fixed position relative to a line of detectors. This is shown in Figure 2.2 and is known as a fan-beam projection because the line integrals are measured along fans.

2.2 The Fourier Slice Theorem

The key to tomographic imaging is the Fourier Slice Theorem which relates the measured projection data to the two-dimensional Fourier transform of the object cross section.

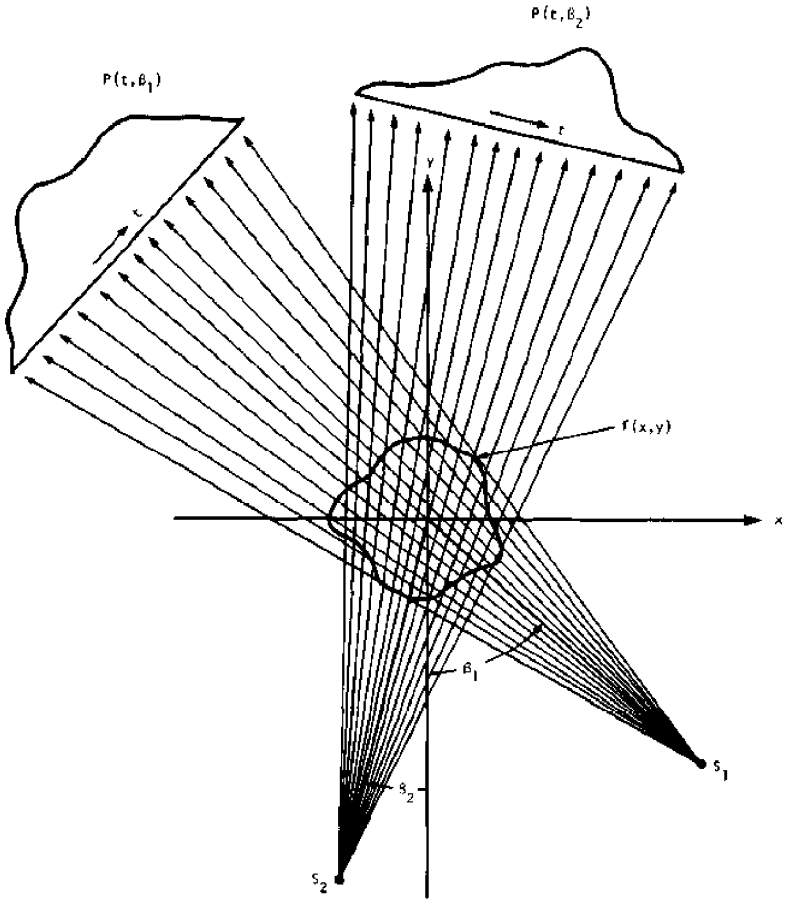


Figure 2.2: A fan beam projection is collected if all the rays meet in one location. (From [Kak87].)

The Fourier Slice Theorem can be derived by taking the one-dimensional Fourier transform of a parallel projection and noting that it is equal to a slice of the two-dimensional Fourier transform of the original object. It follows that given the projection data, it should then be possible to estimate the object by simply performing a two-dimensional inverse Fourier transform.

The first step is to define the two-dimensional Fourier transform of the object function as

$$F(u,v) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-j2\pi(ux+vy)}\,dx\,dy\,, \tag{2.4}$$

and the Fourier transform of a projection at an angle θ as

$$S_\theta(w) = \int_{-\infty}^{\infty} P_\theta(t)e^{-j2\pi wt}dt\,. \tag{2.5}$$

The simplest example of the Fourier Slice Theorem is given for a projection at θ = 0. First, consider the Fourier transform of the object along the line in the frequency domain given by v = 0. The Fourier transform integral now simplifies to

$$F(u,0) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)e^{-j2\pi ux}\,dx\,dy\,. \tag{2.6}$$

but because the phase factor is no longer dependent on y we can split the integral into two parts,

$$F(u,0) = \int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty} f(x,y)\,dy\right]e^{-j2\pi ux}\,dx\ . \tag{2.7}$$

From the definition of a parallel projection, the term in brackets can be recognized as the equation for a projection along lines of constant x or

$$P_{\theta=0}(x) = \int_{-\infty}^{\infty} f(x,y)\,dy\,. \tag{2.8}$$

Substituting this in (2.7) we find

$$F(u,0) = \int_{-\infty}^{\infty} P_{\theta=0}(x) e^{-j2\pi ux}\, dx\,. \tag{2.9}$$

The right-hand side of this equation represents the one-dimensional Fourier transform of the projection $P_{\theta\,=\,0}$; thus we have the following relationship between the vertical projection and the 2-D transform of the object function:

$$F(u,0) = S_{\theta=0}(u)\,. \tag{2.10}$$

This is the simplest form of the Fourier Slice Theorem. Clearly this result is independent of the orientation between the object and the coordinate system. If, for example, as shown in Figure 2.3 the (t, s) coordinate system is rotated by an angle θ, the Fourier transform of the projection defined in (2.8) is equal to the two-dimensional Fourier transform of the object along the line rotated by θ. This leads to the Fourier Slice

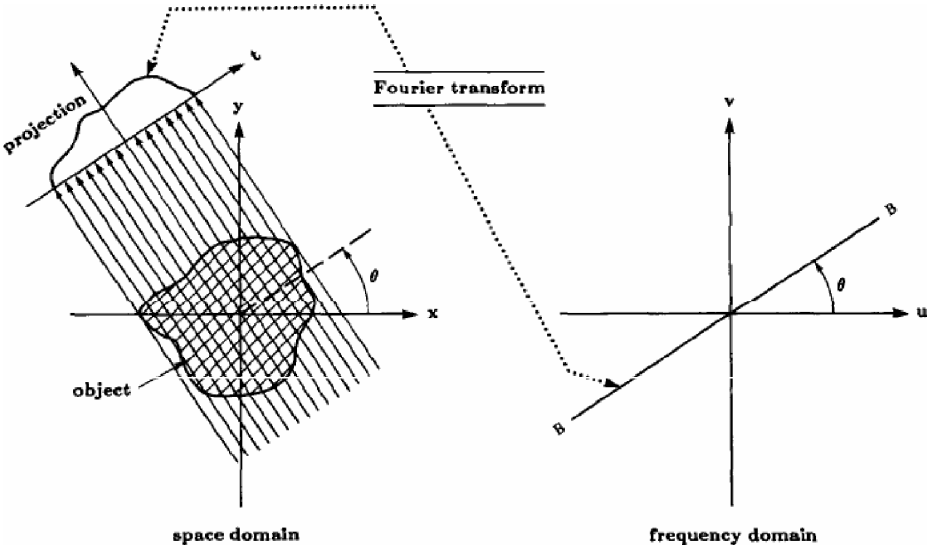


Figure 2.3: The Fourier Slice Theorem relates the Fourier transform of a projection to the Fourier transform of the object along a radial line. (From [Kak87].)

Theorem which is stated as [Kak87]:

> The Fourier transform of a parallel projection of an image $f(x, y)$ taken at angle θ gives a slice of the two-dimensional transform, $F(u, v)$, subtending an angle θ with the u-axis. In other words, the Fourier transform of $P_\theta(t)$ gives the values of $F(u, v)$ along line BB in Figure 2.3.

The Fourier Slice Theorem indicates that by taking projections of an object function at angles $\theta_1, \theta_2, \ldots, \theta_k$ and Fourier transforming each of these, we can determine the values $F(u, v)$ on radial lines as shown in Figure 2.3. If an infinite number of projections are taken, then $F(u, v)$ would be known at all points in the uv-plane. Knowing $F(u, v)$, the object function $f(x, y)$ can be recovered by using the inverse Fourier transform.

In practice only a finite number of projections of an object can be taken. In that case it is clear that the function $F(u, v)$ is only known along a finite number of radial lines. In order to use inverse Finite Fourier transform, one must then interpolate from these radial points to the points on a square grid. Theoretically, one can exactly determine the N^2 coefficients required for this operation provided as many values of the function $F(u, v)$ are known on some radial lines. This calculation involves solving a large set of simultaneous equations often leading to unstable solutions. It is more common to determine the values on the square grid by some kind of nearest neighbor or linear interpolation from the radial points. Since the density of the radial points becomes sparser as one gets farther away from the center, the interpolation error also becomes

larger. This implies that there is greater error in the calculation of the high frequency components in an image than in the low frequency ones, which results in some image degradation.

While this provides a simple conceptual model of tomography, practical implementations require a different approach. The algorithm that is currently being used in almost all applications of straight ray tomography is the filtered backprojection algorithm. It has been shown to be extremely accurate and amenable to fast implementation and will be derived by using the Fourier Slice Theorem.

In the next section, derivation will be presented for the backprojection algorithm for the parallel-beam scanning geometry.

2.3 Parallel-Beam Filtered Backprojection

A parallel-beam CT scanning system uses an array of equally spaced unidirectional sources of focused X-ray beams. Generated radiation, not absorbed by the object's internal structure, reaches a collinear array of detectors (Figure 2.4). Spatial variation of the absorbed energy in the two-dimensional plane through the object is expressed by the attenuation coefficient $f(x, y)$. The logarithm of the measured radiation intensity is proportional to the integral of the attenuation coefficient along the straight line traversed by the X-ray beam. A set of values given by all detectors in the array comprises a one-dimensional projection of the attenuation coefficient, $P(t, \theta)$, where t is

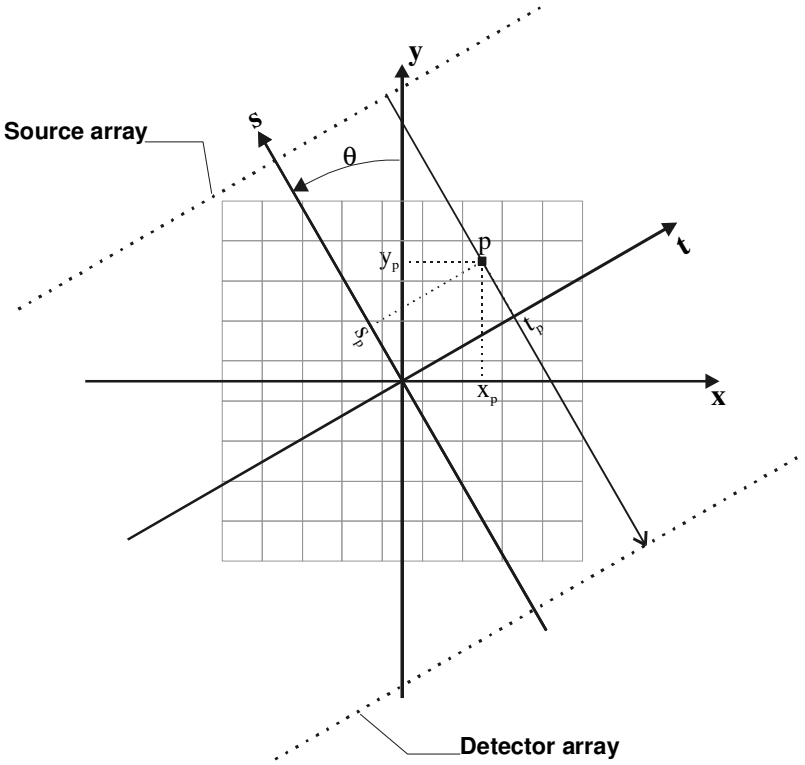


Figure 2.4: Illustration of the coordinate system used in parallel-beam backprojection

the detector distance from the origin of the array, and θ is the angle at which the measurement is taken. A collection of projections for different angles over 180° can be visualized in the form of an image in which one axis is position t and the other is angle θ. This is called a sinogram or Radon transform of the two-dimensional function f, and it contains information needed for the reconstruction of an image $f(x, y)$. The Radon transform can be formulated as

$$\log_e \frac{I_0}{I_d} = \iint f(x, y)\delta(x\cos\theta + y\sin\theta - t)\,dx\,dy \equiv P(t, \theta), \tag{2.11}$$

where I_o is the source intensity, I_d is the detected intensity, and $\delta(\cdot)$ is the Dirac delta function. Equation (2.11) is actually a line integral along the path of the X-ray beam,

which is perpendicular to the t axis (see Figure 2.4) at location $t = x\cos\theta + y\sin\theta$. The Radon transform represents an operator that maps an image $f(x, y)$ to a sinogram $P(t, \theta)$. Its inverse mapping, called the inverse Radon transform, applied to a sinogram results in an image. The filtered backprojection (FBP) algorithm performs this mapping [3].

The mathematical derivation of the filtered backprojection algorithm for parallel-beam projections is presented next. From the formula for the inverse Fourier transform, the object function, $f(x, y)$, can be expressed as

$$f(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u, v) e^{j2\pi(ux + vy)}\, du\, dv\,. \tag{2.12}$$

Exchanging the rectangular coordinate system in the frequency domain, (u, v), for a polar coordinate system, (w, θ), by making the substitutions

$$u = w\cos\theta \tag{2.13}$$

$$v = w\sin\theta \tag{2.14}$$

and then changing the differentials by using

$$du\, dv = w\, dw\, d\theta \tag{2.15}$$

we can write the inverse Fourier transform of a polar function as

$$f(x, y) = \int_{0}^{2\pi} \int_{0}^{\infty} F(w, \theta) e^{j2\pi w(x\cos\theta\, +\, y\sin\theta)}\, w\, dw\, d\theta\,. \tag{2.16}$$

This integral can be split into two by considering θ from 0° to 180 and then from 180 to 360 ,

$$f(x, y) = \int_0^{\pi} \int_0^{\infty} F(w, \theta) e^{j2\pi w(x\cos\theta + y\sin\theta)}\, w\, dw\, d\theta + \int_0^{\pi} \int_0^{\infty} F(w, \theta + 180^\circ) e^{j2\pi w[x\cos(\theta+180^\circ) + y\sin(\theta+180^\circ)]}\, w\, dw\, d\theta \tag{2.17}$$

and then using the property

$$F(w, \theta + 180^\circ) = F(-w, \theta) \tag{2.18}$$

the above expression for $f(x, y)$ may be written as

$$f(x, y) = \int_0^{\pi} \left[\int_{-\infty}^{\infty} F(w, \theta) |w| e^{j2\pi wt}\, dw \right] d\theta . \tag{2.19}$$

Here we have simplified the expression by setting

$$t = x\cos\theta + y\sin\theta \tag{2.20}$$

If we substitute the Fourier transform of the projection at angle θ, $S_\theta(w)$, for the two-dimensional Fourier transform $F(w, \theta)$, we get

$$f(x, y) = \int_0^{\pi} \left[\int_{-\infty}^{\infty} S_\theta(w) |w| e^{j2\pi wt}\, dw \right] d\theta . \tag{2.21}$$

This integral in (33) may be expressed as

$$f(x, y) = \int_0^{\pi} Q_\theta(x\cos\theta + y\sin\theta)\, d\theta \tag{2.22}$$

where

$$Q_\theta(t) = \int_{-\infty}^{\infty} S_\theta(w)\,|w|\, e^{j2\pi wt}\, dw. \tag{2.23}$$

This estimate of $f(x, y)$, given the projection data transform $S_\theta(w)$, has a simple form. Equation (2.23) represents a filtering operation, where the frequency response of the filter is given by $| w |$; therefore $Q_\theta(w)$ is called a "filtered projection." The resulting

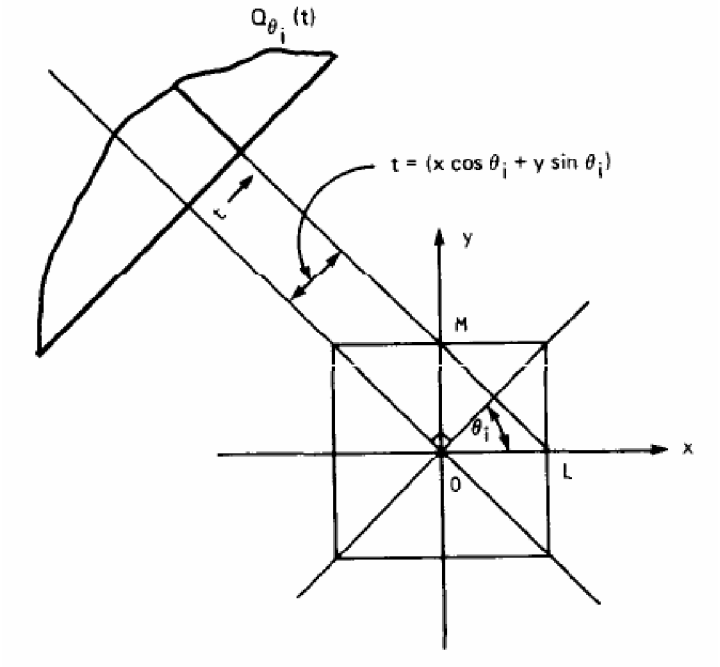


Figure 2.5: Reconstructions are often done using procedure known as backprojection. Here a filtered projection is smeared back over the reconstruction plane along lines of constant *t*. (From [Kak87].)

projections for different angles θ are then added to form the estimate of $f(x, y)$.

Equation (2.22) calls for each filtered projection, Q_θ, to be "backprojected. " This can be explained as follows. To every point (x, y) in the image plane there corresponds a value of $t = x \cos\theta + y \sin\theta$ for a given value of θ, and the filtered projection Q_θ

contributes to the reconstruction its value at t (= $x \cos \theta + y \sin \theta$). This is further illustrated in Fig. 2.5. It is easily shown that for the indicated angle θ, the value of t is the same for all (x, y) on the line LM. Therefore, the filtered projection, Q_θ, will make the same contribution to the reconstruction at all of these points. Therefore, one could say that in the reconstruction process each filtered projection, Q_θ, is smeared back, or backprojected, over the image plane.

The parameter w has the dimension of spatial frequency. The integration in (2.23) must, in principle, be carried out over all spatial frequencies. In practice the energy contained in the Fourier transform components above a certain frequency is negligible, so for all practical purposes the projections may be considered to be bandlimited. If W is a frequency higher than the highest frequency component in each projection, then by the sampling theorem the projections can be sampled at intervals of

$$T = \frac{1}{2W} \tag{2.24}$$

without introducing any error. If we also assume that the projection data are equal to zero for large values of $| t |$ then a projection can be represented as

$$P_\theta(mT), \qquad m = \frac{-N}{2}, \ldots, 0, \ldots, \frac{N}{2} - 1 \tag{2.25}$$

for some (large) value of N.

Let' s assume that the projection data are sampled with a sampling interval of τ cm. If there is no aliasing, this implies that in the transform domain the projections don' t contain any energy outside the frequency interval (- W, W) where

$$W = \frac{1}{2\tau} \text{ cycles/cm.} \tag{2.26}$$

Let the sampled projections be represented by $P_\theta(k\tau)$ where k takes integer values. The theory presented in the preceding subsection says that for each sampled projection $P_\theta(k\tau)$ we must generate a filtered $Q_\theta(k\tau)$. When the highest frequency in the projections is finite (as given by (2.26)), (2.23) may be expressed as

$$Q_\theta(t) = \int_{-\infty}^{\infty} S_\theta(w) H(w)\, e^{j2\pi wt}\, dw \tag{2.27}$$

where

$$H(w) = |w|\, b_w(w) \tag{2.28}$$

where, again,

$$b_w(w) = \begin{cases} 1 & |w| < W \\ 0 & \text{otherwise}. \end{cases} \tag{2.29}$$

$H(w)$, shown in Fig. 2.6, represents the transfer function of a filter with which the projections must be processed. The impulse response, $h(t)$, of this filter is given by the inverse Fourier transform of $H(w)$ and is

$$\begin{aligned} h(t) &= \int_{-\infty}^{\infty} H(w)\, e^{j2\pi wt}\, dw \\ &= \frac{1}{2\tau^2} \frac{\sin 2\pi\, t/2\tau}{2\pi\, t/2\tau} - \frac{1}{4\tau^2} \left(\frac{\sin \pi\, t/2\tau}{\pi\, t/2\tau} \right)^2 \end{aligned} \tag{2.30}$$

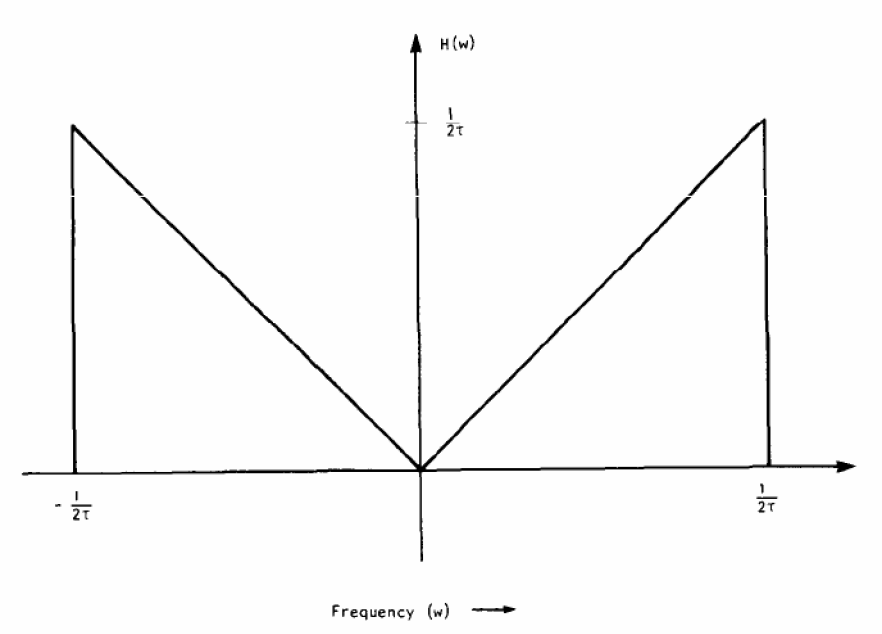


Figure 2.6: The ideal filter response for the filtered backprojection algorithm is shown here. It has been bandlimited to 1/(2τ). (From [Kak87].)

where we have used (2.26). Since the projection data are measured with a sampling interval of τ, for digital processing the impulse response need only be known with the same sampling interval. The samples, $h(\mathrm{n}\,\tau)$, of $h(t)$ are given by

$$h(n\tau) = \begin{cases} \dfrac{1}{4\tau^2}, & \mathrm{n} = 0 \\ 0, & \mathrm{n\ even} \\ -\dfrac{1}{n^2\pi^2\tau^2} & \mathrm{n\ odd} \end{cases} \tag{2.31}$$

This function is shown in Fig. 2.7.

By the convolution theorem the filtered projection at the sampling points can be written as

$$Q_\theta(n\tau) = \tau \sum_{k=-\infty}^{\infty} h(n\tau - k\tau)\, P_\theta(k\tau)\,. \tag{2.32}$$

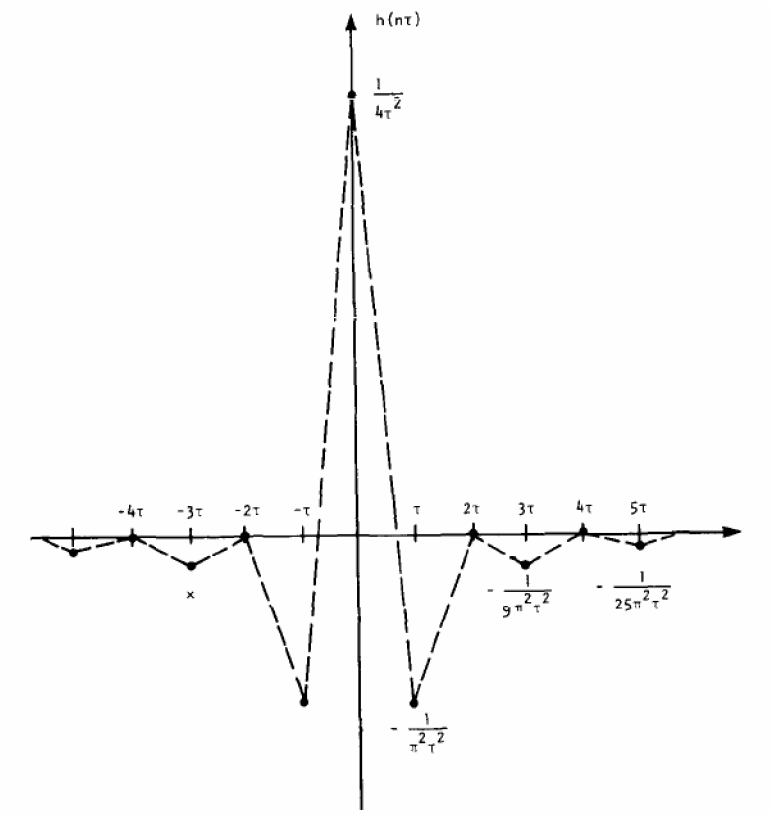


Figure 2.7: The impulse response of the filter shown in Fig. 2.6 is shown here. (From [Kak87].)

In practice each projection is of only finite extent. Suppose that each $P_\theta(k\tau)$ is zero outside the index range $k = 0, \ldots, N - 1$. We may now write the following two equivalent forms of (2.32):

$$Q_\theta(n\tau) = \tau \sum_{k=0}^{N-1} h(n\tau - k\tau)\, P_\theta(k\tau)\,, \qquad n=0, 1, 2, \ldots, N-1 \tag{2.33}$$

or

$$Q_\theta(n\tau) = \tau \sum_{k=-(N-1)}^{N-1} h(k\tau)\, P_\theta(n\tau - k\tau), \qquad n=0, 1, 2, \ldots, N\text{-}1 \tag{2.34}$$

These equations imply that in order to determine $P_\theta(k\tau)$ the length of the sequence $h(n\tau)$ used should be from $l = -(N - 1)$ to $l = (N - 1)$. It is important to realize that the discrete Fourier transform of the sequence $h(n\tau)$ with n taking values in a finite range [such as when n ranges from $-(N - 1)$ to $(N - 1)$] has nonzero dc component which is not the case for the ideal filter response. This is illustrated in Fig. 2.8.

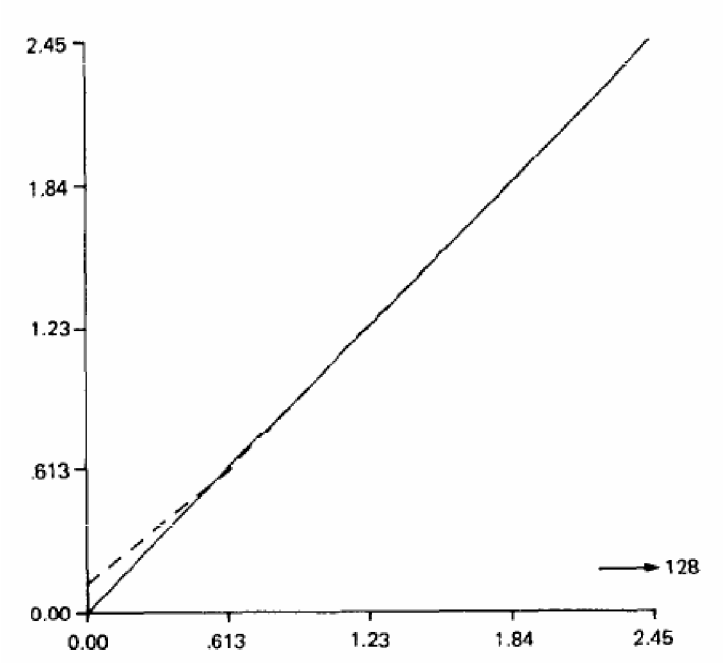


Figure 2.8: The DFT of the bandlimited filter (broken line) and that of the ideal filter (solid line) are shown here. Notice the primary difference is in the dc component. (From [Kak87].)

The reconstructed picture $f(x, y)$ may then be obtained by the discrete approximation to the integral in (2.22), i.e.,

$$f(x, y) = \frac{\pi}{K} \sum_{i=1}^{K} Q_{\theta_i} \left(x \cos \theta_i + y \sin \theta_i \right), \tag{2.35}$$

where the K angles θ_i are those for which the projections $P_\theta(t)$ are known. Note that the value of $x \cos \theta_i + y \sin \theta_i$ in (2.35) may not correspond to one of the $k\tau$ at which Q_{θ_i} is known. However, Q_{θ_i} for such t may be approximated by suitable interpolation; often linear interpolation is adequate.

2.4 Algorithms

2.4.1 Reprojection – Square Pixel Method

The reprojection algorithm is essentially the inverse of the backprojection procedure. It is encountered in a variety of areas, e.g., iterative correction of CT images for beam hardening affects, iterative algorithms for image reconstructions from a limited number of views, 3-D reconstruction algorithms in nuclear medicine and 3-D projection and stereoscopic visualization of CT images.

There are various algorithms for calculating projections. They differ in complexity of required calculations and accuracy. In this thesis, reprojection is needed for generation of sinograms of input images that are used as inputs to the simulation process. The simulation process is used to evaluate the effects of quantization on the

quality of reconstructed images by comparing these images with non-quantized (floating-point) reconstructed images. Since for both quantized and non-quantized reconstructions input sinograms are generated by using the same reprojection method, the error introduced by the reprojection process itself exists in both types of compared images and because of that it does not affect result of the comparison in any significant extent. For this reason the algorithm chosen to be used for the implementation of reprojection does not need to be highly accurate. It is more important that it can be implemented quickly and that it is not computationally complex. The Square Pixel Method, which was ultimately used for the purpose of this thesis, is relatively easy to implement and gives satisfying accuracy of generated sinograms.

The basic assumption of the Square Pixel Method is that the object considered truly consists of an array of $N \times N$ square pixels, with the image function $f(x, y)$ assumed to be constant over the domain of each pixel. The method proceeds (ideally) by evaluating the length of intersection of each ray with each pixel, and multiplying by the value of the pixel

$$S(K) = \sum_{r,c} L_{r,c}\, P_{r,c}\,, \tag{2.36}$$

where the indices r, c specify the row and column of the pixel, whose value is $P_{r,c}$. That is, $P_{r,c} = f(x_r, y_c)$, so that x_r, y_c are the coordinates of the center of the pixel. $L_{r,c}$ is the length of intersection of ray K with pixel (r, c). In practice, for any given ray K, most of

the pixels contribute zero intersection, so some sort of efficient method to select out those pixels with $L_{r,c} > 0$ is needed.

Appendix A contains a MatLab function that implements reprojection for parallel-beam, collinear detector fan-beam and equiangular fan-beam configurations. This script practically simulates a CT scanner for each of these configurations. Geometrical properties of a simulated scanner like detector spacing, pixel size, total number of pixels in a reprojected image, number of projections, number of detectors, and distance between the detector array and the X-ray emitter(s), all represent inputs to this function.

2.4.2 Backprojection – Incremental Algorithm

Section 2.3 described the theoretical foundation of the filtered parallel-beam backprojection algorithm. This section presents a practical approach for the efficient hardware implementation of the backprojection. The key feature of this approach is the incremental calculation of the spatial address representing the location where the X-ray going through currently reconstructed pixel intersects the detector array.

Equation (2.35) is the mathematical definition of the backprojection process. For its practical implementation, it is important to determine how many projections and how many samples per projection one should use to obtain highly accurate reconstructions. The number of samples (detectors) depends on the image size. In case of $n \times n$ pixel images, $N = \sqrt{2}nD$ detectors are required. The ratio $D = d/\tau$, where d is the distance

between adjacent pixels and τ is the detector spacing, is a critical factor for the quality of a reconstructed image and it obviously should satisfy $D > 1$. In this implementation, it is $D \approx 1.4$ and $N = 1024$, which is typical for real world systems. Higher values do not significantly increase the image quality. As for the number of projections, since the summation in Equation (2.35) represents an approximation of the integral, obviously the higher the number of projections is the more accurate this approximation is going to be. Still, for all practical purposes 1024 projections used here are sufficiently enough. This implies that projections are taken at angles 0°, 180°/1024, 2·180°/1024, . . ., 1023·180°/1024.

Algorithmically, Eq. (2.35) is implemented as a triple nested "for" loop. The outermost loop is over projection angle, θ. For each θ, incremental algorithm updates every pixel in the image in raster-scan order: starting in the upper left corner and looping first over columns, c, and next over rows, r. Thus, from Eq. (2.35), pixel at location (r, c) is incremented by the value of $Q_\theta(t)$ where t is a function of r and c. The issue here is that the X-ray going through the currently reconstructed pixel, in general, intersects the detector array between detectors. This is solved by linear interpolation. The point of intersection is calculated as an address corresponding to detectors numbered from 0 to 1023. The fractional part of this address (spatial address) is the interpolation factor. The equation that performs linear interpolation is given by

$$Q_\theta^{\text{int}}(i) = \left[Q_\theta(i+1) - Q_\theta(i)\right] \cdot IF + Q_\theta(i), \tag{2.37}$$

where *IF* denotes interpolation factor, $Q_\theta(\cdot)$ is the 1024-element-long array containing filtered projection data at angle θ, and i is the integer part of the calculated address. The interpolation can be performed beforehand in software, or it can be a part of the backprojection hardware itself. Solution presented here implements interpolation in hardware for two reasons. First, this substantially reduces the amount of data that must be transmitted to the reconfigurable board. Second, interpolation in hardware is much faster, thus speeding up the reconstruction process.

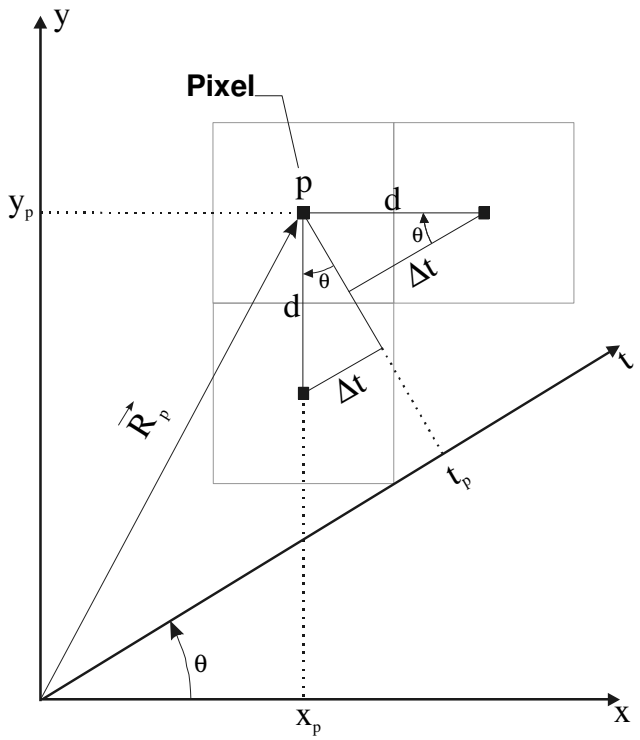


Figure 2.9: Geometrical explanation of the incremental spatial address calculation

The key to an efficient implementation of Equation (2.35) is shown in Figure 2.9. It shows how a distance d between square areas that correspond to adjacent pixels can be converted to a distance Δt between locations where X-ray beams that go through the

centers of these areas hit the detector array. This is also derived from the equation $t = x\cos\theta + y\sin\theta$. Assuming that pixels are processed in raster-scan fashion, then $\Delta t = d\cos\theta$ for two adjacent pixels in the same row ($x2 = x1 + d$) and similarly $\Delta t = d\sin\theta$ for two adjacent pixels in the same column ($y2 = y1 - d$). Presented implementation is based on pre-computing and storing these deltas in look-up tables. Three LUTs are used corresponding to the nested "for" loop structure of the backprojection algorithm. LUT 1 stores the initial address along the detector axis (i.e. along t) for a given θ required to update the pixel at row -1, column -1. This pixel is actually not part of the image, it is diagonally next to the pixel in the upper-left corner of the image and its coordinates are (x_0, y_0). Choosing this pixel to be a starting one makes hardware more regular (see Fig. 2.10). LUT 2 (resp. LUT 3) stores the increment in t required as algorithm increments columns (resp. rows). The contents of the look-up tables is given in Equations (2.38), (2.39) and (2.40), where Δ_{detector} is detector spacing and $N_{\text{detectors}}$ is the number of detectors used for projection sampling.

$$\text{LUT\_1}(i) = \frac{x_0 \cos(\theta_i)}{\Delta_{\text{detector}}} + \frac{y_0 \sin(\theta_i)}{\Delta_{\text{detector}}} + \frac{N_{\text{detectors}}\text{-}1}{2} + \frac{d \sin(\theta_i)}{\Delta_{\text{detector}}} - \frac{d \cos(\theta_i)}{\Delta_{\text{detector}}} \qquad (2.38)$$

$$\text{LUT\_2}(i) = \frac{d \sin(\theta_i)}{\Delta_{\text{detector}}} \qquad (2.39)$$

$$\text{LUT\_3}(i) = \frac{d \cos(\theta_i)}{\Delta_{\text{detector}}} \qquad (2.40)$$

Figure 2.10 shows a simplified data path of the incremental algorithm for backprojection. The data path consists of two major blocks. The first one implements

incrementa spatial address generation. Its input is variable *prj_num* which represents the number of currently processed projection, and it serves as address into all three look-up tables. The output of this block is designated as *loc_ij[n+m-1..0]* and it represents the spatial address (location on axis t corresponding to pixel (i, j)) consisting of $n+m$ bits. Lower m bits represent the fractional part of this address which is used as interpolation factor. So, as the pixels are being processed in the raster-scan fashion, the spatial address is generated by accumulating entries from LUTs 2 and 3 with the corresponding entry in LUT 1. The second block in Fig. 2.10 consists of a subtracter, multiplier and adder, and it implements linear interpolation given in Equation (2.37). The output of this block is

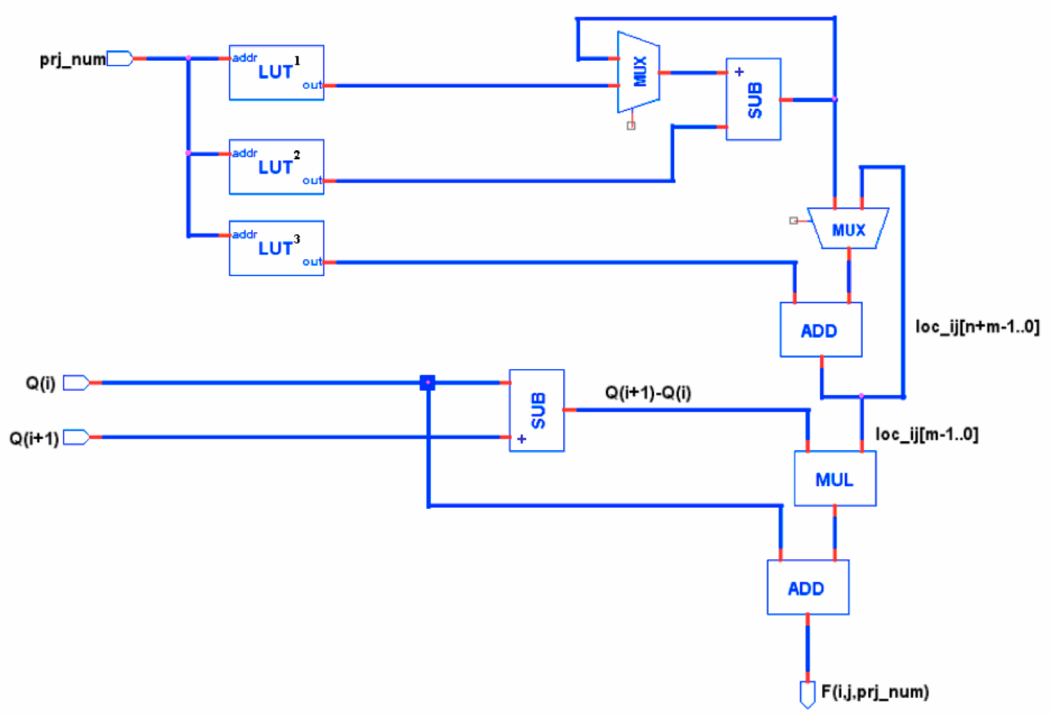


Figure 2.10: Block diagram of the data path implementing incremental algorithm for backprojection

designated as $F(i, j, prj\_num)$ and it represents reconstructed value of the pixel (i, j) from projection *prj_num*.

2.5 Fixed-Point Numbers

Within digital hardware, numbers are represented as either fixed-point or floating-point data types. For both these data types, word sizes are fixed at a set number of bits. However, the dynamic range of fixed-point values is much less than floating-point values with equivalent word sizes. Therefore, in order to avoid overflow or unreasonable quantization errors, fixed-point values must be scaled. Since floating-point processors can greatly simplify the real-time implementation of a control law or digital filter, and floating-point numbers can effectively approximate real-world numbers, then why use a microcontroller or processor with fixed-point hardware support? The answer to this question in many cases is cost and size:

- Cost – Fixed-point hardware is more cost effective where price/cost is an important consideration. When using digital hardware in a product, especially mass-produced products, fixed-point hardware, costing much less than floating-point hardware, can result in significant savings.
- Size – The logic circuits of fixed-point hardware are much less complicated than those of floating-point hardware. This means the fixed-point chip size is smaller with less power consumption when compared with floating-point hardware. For example,

consider a portable telephone where one of the product design goals is to make it as portable (small and light) as possible. If one of today's high-end floating-point, general purpose processors is used, a large heat sink and battery would also be needed resulting in a costly, large, and heavy portable phone.

When developing a dynamic system using floating-point arithmetic, you generally don't have to worry about numerical limitations since floating-point data types have high precision and range. Conversely, when working with fixed-point arithmetic, you must consider these factors when developing dynamic systems:

- **Overflow**

  Adding two sufficiently large negative or positive values can produce a result that does not fit into the representation.

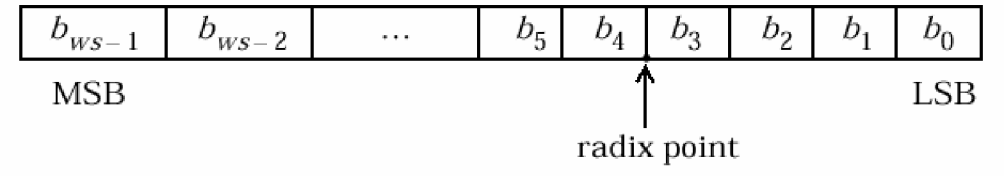
- **Quantization**

  Fixed-point values are rounded. Therefore, they contain quantization error.

- **Computational noise**

  The accumulated errors that result from the rounding of individual terms within the realization introduces noise into the control signal.

A common representation of a binary fixed-point number (either signed or unsigned) is shown below.



where:

- b_i are the binary digits (bits).
- The size of the word in bits is given by ws.
- The most significant bit (MSB) is the leftmost bit, and is represented by location b_{ws-1}.
- The least significant bit (LSB) is the rightmost bit, and is represented by location b_0.
- The radix (binary) point is shown four places to the left of the LSB.

Computer hardware typically represent the negation of a binary fixed-point number in three different ways: sign/magnitude, one's complement, and two's complement. Two's complement is the preferred representation of signed fixed-point numbers and is used in this thesis.

Whether a fixed-point value is signed or unsigned is usually not encoded explicitly within the binary word (i.e., there is no sign bit). Instead, the sign information is implicitly defined within the computer architecture.

The radix point is the means by which fixed-point numbers are scaled. It is usually the software that determines the radix point. When performing basic math functions such as addition or subtraction, the hardware uses the same logic circuits regardless of the value of the scale factor. In essence, the logic circuits have no knowledge of a scale factor. They are performing signed or unsigned fixed-point binary algebra as if the radix point is to the right of b_0.

Scaling

The dynamic range of fixed-point numbers is much less than that of floating-point numbers with equivalent word sizes. To avoid overflow conditions and minimize quantization errors, fixed-point numbers must be scaled.

A fixed-point number can be represented by a general slope/bias encoding scheme

$$V \approx V_a = S\,Q + B\,, \tag{2.41}$$

where:

- V is an arbitrarily precise real-world value.
- V_a is the approximate real-world value.
- Q is an integer that encodes V.
- $S = F \cdot 2^E$ is the slope.
- B is the bias.

The slope is partitioned into two components:

- 2^E specifies the radix point. E is the fixed power-of-two exponent.
- F is the fractional slope. It is normalized such that $1 \leq F < 2$.

Note that S and B are constants and do not show up in the computer hardware directly – only the quantization value Q is stored in computer memory.

The scaling modes available within this encoding scheme are described below.

Radix Point-Only Scaling

As the name implies, radix point-only (or "powers-of-two") scaling involves moving only the radix point within the generalized fixed-point word. The advantage of this scaling mode is the number of processor arithmetic operations is minimized. With radix point-only scaling, the components of the general slope/bias formula have these values:

- $F = 1$
- $S = 2^E$
- $B = 0$

That is, the scaling of the quantized real-world number is defined only by the slope S, which is restricted to a power of two. For example, $E = -10$ defines a scaling such that the radix point is at a location 10 places to the left of the least significant bit.

Slope/Bias Scaling

When scaling by slope and bias, the slope S and bias B of the quantized real-world number can take on any value. The slope must be a positive number.

Quantization

The quantization Q of a real-world value V is represented by a weighted sum of bits. Within the context of the general slope/bias encoding scheme, the value of an unsigned fixed-point quantity is given by

$$V_a = S \cdot \left[\sum_{i=0}^{ws-1} b_i \, 2^i \right] + B \,, \tag{2.42}$$

while the value of a signed fixed-point quantity is given by

$$V_a = S \cdot \left[-b_{ws-1} 2^{ws-1} + \sum_{i=0}^{ws-2} b_i \, 2^i \right] + B \,, \tag{2.43}$$

where:

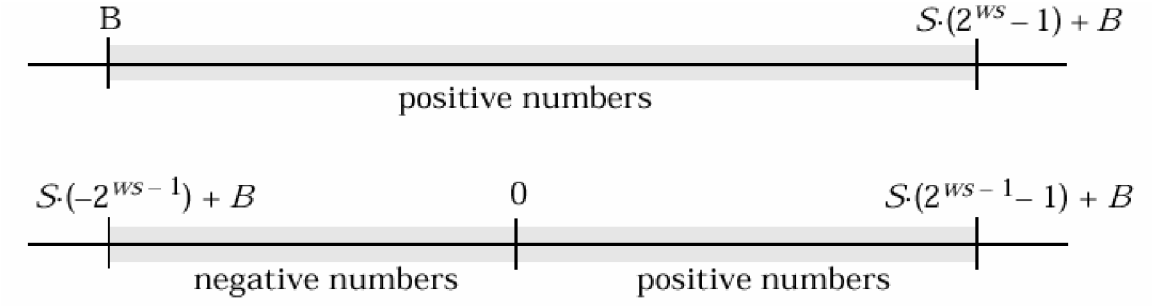
- b_i are binary digits, with $b_i = 1, 0$.
- The word size in bits is given by ws.
- S is given by $F \cdot 2^E$, where the scaling is unrestricted since the radix point does not have to be contiguous with the word.

b_i are called bit multipliers and 2^i are called the weights.

Range and Precision

The range of a number gives the limits of the representation while the precision gives the distance between successive numbers in the representation. The range and precision of a fixed-point number depends on the length of the word and the scaling.

The range of representable numbers for an unsigned and two's complement fixed-point number of size ws, scaling S, and bias B is illustrated below.



For both the signed and unsigned fixed-point numbers of any data type, the number of different bit patterns is 2^{ws}. For example, if the fixed-point data type is an integer with scaling defined as S = 1 and $B = 0$, then the maximum unsigned value is $2^{ws} - 1$ since zero must be represented. In two's complement, negative numbers must be represented as well as zero so the maximum value is $2^{ws-1} - 1$. Additionally, since there is only one representation for zero, there must be an unequal number of positive and negative numbers. This means there is a representation for -2^{ws-1} but not for 2^{ws-1} .

Limitations on the range of a fixed-point word occur for the same reason as limitations on its precision. Namely, fixed-point words have limited size. The range depends on the word's size and scaling. In binary arithmetic, a processor may need to take an n-bit fixed-point number and store it in m bits, where . If $m < n$, the range of the number has been reduced and an operation can produce an overflow condition, which can be delt with by using saturation or wrapping. Saturation sets an overflow condition to the maximum positive or negative value allowed. Conversely, wrapping sets an overflow condition to the appropriate value within the range of the representation.

Computer words consist of a finite numbers of bits. This means that the binary encoding of variables is only an approximation of an arbitrarily precise real-world value. Therefore, the limitations of the binary representation automatically introduce limitations on the precision of the value.

The precision of a fixed-point word depends on the word size and radix point location. For generalized fixed-point data types, the scaling is explicitly defined as either

slope/bias or radix point-only. In either case, the precision is given by the slope. Extending the precision of a word can always be accomplished with more bits although you face practical limitations with this approach. Instead, you must carefully select the data type, word size, and scaling such that numbers are accurately represented. Rounding is typical method implemented on processors to deal with the precision of binary words.

The result of any operation on a fixed-point number is typically stored in a register that is longer than the number's original format. When the result is put back into the original format, the extra bits must be disposed of. That is, the result must be rounded. Rounding involves going from high precision to lower precision and produces quantization errors and computational noise.

Round Toward Zero

The computationally simplest rounding mode is to drop all digits beyond the number required. This mode is referred to as rounding toward zero, and it results in a number whose magnitude is always less than or equal to the more precise original value.

Rounding toward zero introduces a cumulative downward bias in the result for positive numbers and a cumulative upward bias in the result for negative numbers. That is, all positive numbers are rounded to smaller positive numbers, while all negative numbers are rounded to smaller negative numbers.

Rounding to zero and truncation or chopping are sometimes thought to mean the same thing. However, the results produced by rounding to zero and truncation are different for unsigned and two's complement numbers.

To illustrate this point, consider rounding a 5-bit unsigned number to zero by dropping (truncating) the two least significant bits. For example, the unsigned number 100.01 = 4.25 is truncated to 100 = 4. Therefore, truncating an unsigned number is equivalent to rounding to zero or rounding to floor.

Now consider rounding a 5-bit two's complement number by dropping the two least significant bits. For example, dropping the last two digits of -3.75 yields -3.00. However, digital hardware performing two's complement arithmetic yields a different result. Specifically, the number 100.01 = -3.75 truncates to 100 = -4, which is rounding to floor.

Therefore, rounding to zero for a two's complement number is not the same as truncation when the original value is negative.

Round Toward Nearest

When rounding toward nearest, the number is rounded to the nearest representable value. This mode has the smallest errors associated with it and these errors are symmetric. As a result, rounding toward nearest is the most useful approach for most applications.

Round Toward Ceiling

When rounding toward ceiling, both positive and negative numbers are rounded toward positive infinity. As a result, a positive cumulative bias is introduced in the number.

Rounding toward ceiling and rounding toward floor are sometimes useful for diagnostic purposes. For example, after a series of arithmetic operations, you may not know the exact answer because of word-size limitations, which introduce rounding. If every operation in the series is performed twice, once rounding to positive infinity and once rounding to negative infinity, you obtain an upper limit and a lower limit on the correct answer. You can then decide if the result is sufficiently accurate or if additional analysis is required.

Maximizing Precision

Precision is limited by slope. To achieve maximum precision, the slope should be made as small as possible while keeping the range adequately large. The bias will be adjusted in coordination with the slope.

Assume the maximum and minimum real-world value is given by max(V) and min(V), respectively. These limits may be known based on physical principles or engineering considerations. To maximize the precision, you must decide upon a rounding scheme and whether overflows saturate or wrap. To simplify matters, this example assumes the minimum real-world value corresponds to the minimum encoded value, and the maximum real-world value corresponds to the maximum encoded value. Using the encoding scheme described in “Scaling”, these values are given by

$$\max(V) = F\,2^{E}\,[\max(Q)] + B\,, \tag{2.44}$$

$$\min(V) = F\,2^{E}\,[\min(Q)] + B\,. \tag{2.45}$$

Solving for the slope and bias, you get

$$S = \frac{\max(V) - \min(V)}{\max(Q) - \min(Q)} = \frac{\max(V) - \min(V)}{2^{ws} - 1}, \tag{2.46}$$

$$B = \max(V) - S \cdot \max(Q) \quad \text{or} \quad B = \min(V) - S \cdot \min(Q), \tag{2.47}$$

This formula is independent of rounding and overflow issues, and depends only on the word size, *ws*.

Once slope and bias are determined, quantization of the real-word values with maximal precision is done according to following formula

$$Q = round\left(\frac{V - B}{S}\right), \tag{2.48}$$

where *round* denotes rounding toward nearest.

Arithmetic and Scaling

This section describes the relationship between arithmetic operations and fixed-point scaling. Scaling choices are based on:

- Minimizing the number of arithmetic operations of the result.
- Maximizing the precision of the result.

The second consideration has been explained in the above text. In this section, the first consideration is described in more detail by using the multiplication example.

Consider the multiplication of two real-world values.

$$V_a = V_b \times V_c \tag{2.49}$$

These values are represented by the general slope/bias encoding scheme described in "Scaling" section.

$$V_i = F_i \, 2^{E_i} \, Q_i + B_i \tag{2.50}$$

In a fixed-point system, the multiplication of values results in finding the variable Q_a.

$$\begin{aligned} Q_a = {} & \frac{F_b F_c}{F_a} \cdot 2^{E_b + E_c - E_a} Q_b Q_c + \frac{F_b F_c}{F_a} \cdot 2^{E_b - E_a} Q_b + \frac{F_c F_b}{F_a} \cdot 2^{E_c - E_a} Q_c \\ & + \frac{B_b B_c - B_a}{F_a} \cdot 2^{-E_a} \end{aligned} \tag{2.51}$$

This formula shows:

- In general, Q_a a is not computed through a simple multiplication of Q_b and Q_c.
- In general, there is one multiply of a constant and two variables, two multiplies of a constant and a variable, three additions, and some additional bit shifting.

Implementing in hardware formula (2.51) is unacccaptable. Obviosly, some simplifications of the scaling for these three numbers has to be made. Lets consider the case where $B_b = 0$, and $B_c = 0$, $F_c = 1$. This means that number V_b might be obtained as difference of two numbers with general slope/bias scaling from (2.41) so that bias is canceled, while number V_c is using radix point-only scaling. Also, lets assume that the scaling of the result V_a is the same as scaling of the input V_b. The multiplication of these values results in finding the variable Q_a according to:

$$Q_a = 2^{E_c} Q_b Q_c \tag{2.52}$$

In this case, Q_a is computed through a simple multiplication of Q_b and Q_c, and bit-shifting of the result.

Choosing scaling method so that hardware implementation of arithmetic operations is as simple, and fast as possible is important consideration when using fixed-point numbers. This will have direct implications on the design presented in this thesis.

2.5 Reconfigurable Hardware - FPGAs[1]

The most common reconfigurable device is the Field Programmable Gate Array (FPGA). FPGAs grew out of earlier devices such as Programmable Array Logic (PAL) that implemented combinational logic functions with fixed inputs and outputs as look-up tables (LUTs) [Old1995]. The advantage of using FPGAs is that they can be as flexible as software without losing the speed, as illustrated in Figure 2.11.

The most common class of FPGA devices is channel-type FPGAs, which has configurable logic blocks (CLBs) separated by wiring channels. The CLBs are the function elements that implement the algorithm [Xil2000]. The wiring channels provide for data movement around the chip.

[1] Adapted from masters thesis "Image Processing Desgins in JHDL, a Java-based Hardware Design language" by Heather Quinn

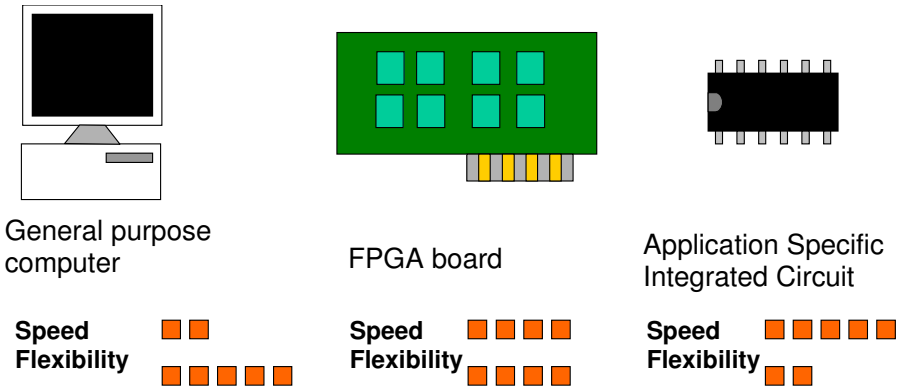


Figure 2.11: Speed vs. flexibility of software, FPGA and ASICs

FPGA chips can be placed into different devices or boards for use. The designs in this thesis have been implemented for the Annapolis WildStar board. This board has:

- 3 Xilinx® VIRTEX™ 1000 FPGAs,
- Processing clock rates up to 100MHz,
- 1.6 GB/s I/O bandwidth,
- 6.4 GB/s memory bandwidth,
- 40MB of 100MHz Synchronous ZBT SRAM

A picture of the board is shown in Figure 2.12, and a schematic diagram of the board in Figure 2.13. The 3 Xilinx chips are called processing elements (PEs). The three PEs can be used in conjunction or separately. There are also four mezzanine memory units that the PEs have access to. The mezzanine memories can be used to hold the image data while being processed. Many of the designs in this thesis use PE0 to interface with the mezzanine memory and the host PC to handle the input image data, PE1 to do

the processing on the image and PE2 to handle the output image by interfacing with the mezzanine memory and the host.

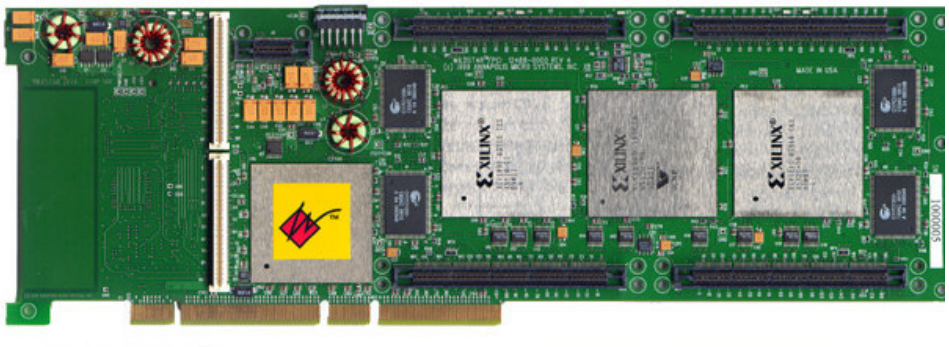


Figure 2.12: Annapolis WildStar

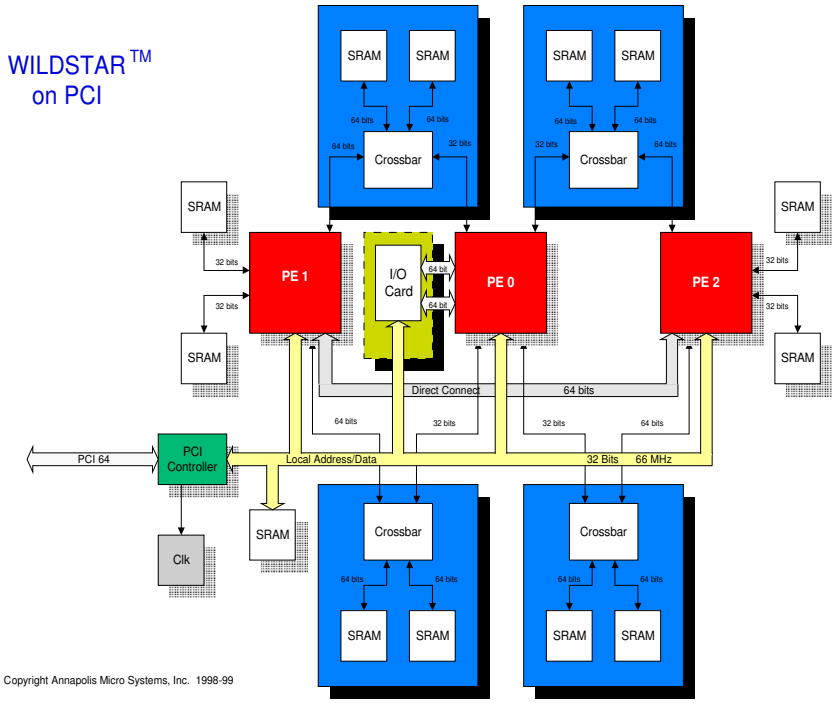


Figure 2.12: WildStar Schematic Diagram

The FPGA used in this thesis is Xilinx Virtex™ 1000 FPGA chip. It is made up of configurable logic blocks, input/output blocks and interconnect. Each of these pieces are important to consider when describing circuits for FPGAs.

2.5.1 Configurable Logic Blocks

The CLB in a Xilinx Virtex™ 1000 chip has three function generators. The function generators can be used for logic or for RAM. The F and G function generators have 4-inputs, and the H function generator has 3-inputs. The F and G function generators can only take input data from outside the CLB; the H function generator can take up to 2 inputs from each of the F and G function generators, as well as outside inputs. For example, the F and G function generators can be used to implement an eight-bit comparator with the upper nibble in the F function generator and the lower nibble in the G function generator. Then the H function generator can combine the results for output. Each CLB has two storage elements that can be used independent of or in conjunction with the function generators. There are four output signals on each CLB. The function generators can drive two of these outputs without latching and the other two outputs are driven by the storage elements. The storage elements are driven by the F and G function generators or by independent data.

There are very obvious constraints on what a CLB can implement. With the F, G, H function generator structure, these functions can be done:

- Any function of up to four variables, plus any second function of up to four unrelated variables, plus any third function of up to three unrelated variables
- Any single function of five variables
- Any function of four variables together with some functions of six variables
- Some functions of up to nine variables
- It is also possible to use the F and G function generators as RAM instead of logic. The function generators can be configured in the following ways for RAM:
  - Two 16x1 RAMs: two data inputs and two data outputs with identical or, if preferred, different addressing for each RAM
  - One 32x1 RAM: one data input and one data output.
  - One of the function generators can be used to implement a function and the other used as RAM.

2.5.2 Input/Output Blocks

Input/Output Blocks (IOBs) are the interfacing circuits for external chip data and internal signal lines. There is one IOB per package pin and the IOB can be programmed to allow the pin to be an input, output or bi-directional pin. They can also be programmed to either register or not register input data before outputting to the signal lines.

2.5.3 Routing Data

Routing data between CLBs can be the hardest area in FPGA algorithm implementation, unlike traditional ASIC design. There are a complex variety of ways to move data around the chip. Each CLB in a V1000 chip has access to these routing resources: singles, doubles, longlines, globals and carry logic. The routing has horizontal and vertical routing methods, although some routing methods do not work in both directions. Both doubles and singles have an equivalent number of lines in horizontal and vertical directions. There are programmable switch matrices (PSMs) that allow data to be routed from a vertical line to horizontal line or vice versa.

Single length lines are used to connect a CLB to its nearest neighbors. Since they are connected via PSMs, they only route data efficiently for short distances. For slightly longer distances, double lines can be used. Double lines pass through every other PSM so they can travel twice the distance as the single line with the same amount of delay. Some FPGA chips have quad lines that are twice the length of the doubles and pass through only one in four PSMs. For data that needs to travel the length or the width of the chip there are longlines that are controlled by tristate buffers. Another routing possibility that some chips have are direct interconnects. The direct interconnects allow one CLB to directly pass data to adjacent CLBs without passing through a PSM. A final routing method is global lines that are used for the clock and reset lines.

There is also routing associated with the IOBs. In general, there is routing directly from pin to IOB to CLB. Should the pins change where data is being handled, some FPGA chips can reroute the pins without changing the design.

2.5.4 Algorithm Implementation on FGPAs

To be able to use the FPGA, one must be able to implement the algorithm on the chip. There are many ways to achieve this process. Most often people use Hardware Description Languages (HDLs). The advantage of using an HDL is the ability to rapidly develop algorithm descriptions that can be used on the chip.

Implementing algorithms on an FGPA chip is not the same as with ASICs. While the method of describing the algorithm might be the same, the design processes split once the algorithm is ready for fabrication. In ASIC fabrication, the design process involves designing masks for silicon and getting chips made from the masks. For FPGA, fabrication involves moving the design to FPGA chip. The process of mapping the design is often not straightforward. As a result, there are many tools to help this process. This thesis examines automatical mapping of image processing algorithms to reconfigurable hardware.

The Design Process and Hardware Description Languages

The hardware design process involves transforming a circuit from concept to manufacture. This process usually involves many steps as outlined in Figure 2.13:

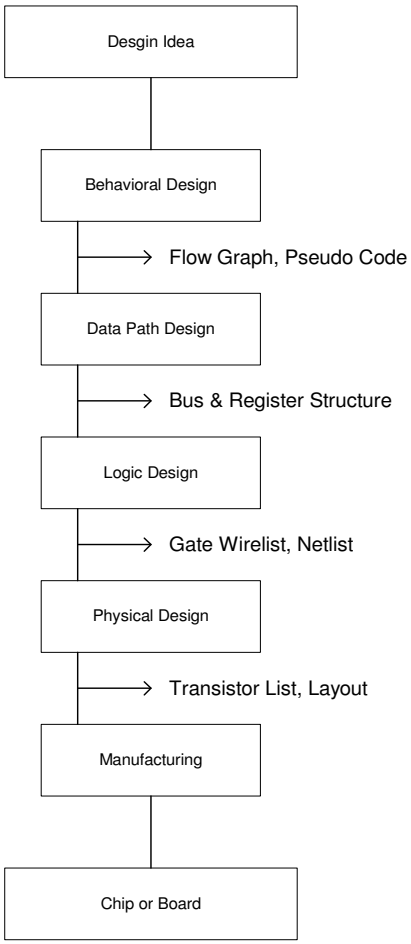


Figure 2.13: Design Process [Nav1998]

Much of the design process involves taking one design abstraction and transforming it to another design abstraction, e.g. Flow Graph to Register Structure. The transformation to different design abstractions can be considered "tedious and repetitive", which after the Data Path Design phase "a machine can do much faster than a talented engineer" [Nav1998]. Therefore, design automation is needed to make design work less arduous for the engineer. Design automation tools help the designer do design entry, transformation from one abstraction level to another, and design verification. Therefore,

design automation tools need to include editors and simulators that can work with designs described at any of the most common abstraction levels. Design automation tools also reduce human error in transforming designs from different abstraction levels.

Most designs are specified using textual descriptions. There are a few methods available for text description. Traditional methods are state diagrams, timing diagrams, state assignments and truth tables. Hardware Description Languages (HDLs) represent one method of achieving a textual method of hardware design [Arm2000]. HDLs are particularly good for describing the algorithms of a circuit.

HDLs represent a way to handle some of the layers of abstraction necessary in circuit design. HDLs are an efficient method to design circuits as the circuit can be quickly described and synthesized without spending time involved in extensive design layouts. The better-known languages, such as Verilog and VHDL, have many tools to make the design process easy. Many vendors give the designer tools for editing, simulating and synthesizing circuit descriptions. Besides making the design cycle shorter for hardware engineers, HDLs also allow for easily reusable circuit units with design parameterizations.

2.5 Related Work

Chapter 3

Experimental Setup