A VLSI Systolic Array Processor for Complex Singular Value Decomposition

by

Christopher Charles Niessen

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Electrical Engineering

and

Master of Science in Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1994

Author . . . .
Department of Electrical Engineering and Computer Science
May 6, 1994

Certified by . . . .
Srinivas Devadas
Associate Professor of Electrical Engineering
Thesis Supervisor

Certified by . . . .
Steven R. Broadstone
Staff, MIT Lincoln Laboratory
Company Supervisor

Certified by . . . .
H. T. Kung
Professor, Harvard University

Accepted by . . . .
Frederic R. Morganthaler
Chairman, Departmental Committee on Graduate Students

Eng.

A VLSI Systolic Array Processor for Complex Singular Value Decomposition

by

Christopher Charles Niessen

Submitted to the Department of Electrical Engineering and Computer Science
on May 6, 1994, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Electrical Engineering
and
Master of Science in Electrical Engineering

Abstract

The singular value decomposition is one example of a variety of more complex routines that are finding use in modern high performance signal processing systems. In the interest of achieving the maximum possible performance, a systolic array processor for computing the singular value decomposition of an arbitrary complex matrix was designed using a silicon compiler system. This system allows for ease of design by specification of the processor architecture in a high level language, utilizing parts from a variety of cell libraries, while still benefiting from the power of custom VLSI. The level of abstraction provided by this system allowed more complex functional units to be built up from existing simple library parts. A novel fast interpolation cell for computation of square roots and inverse square roots was designed, allowing for a new algebraic approach to the singular value decomposition problem. The processors connect together in a systolic array to maximize computational efficiency while minimizing overhead due to high communication requirements.

Thesis Supervisor: Srinivas Devadas
Title: Associate Professor of Electrical Engineering

Company Supervisor: Steven R. Broadstone
Title: Staff, MIT Lincoln Laboratory

Reader: H. T. Kung
Title: Professor, Harvard University

Acknowledgments

I would like to extend my deepest gratitude to Dr. Steven Broadstone for all of the help he has given me throughout this thesis. His help and guidance have proved to be absolutely essebtial.

I would also like to thank Professor Srinivas Devadas for supervising this thesis and for his helpful input, and Professor Kung of Harvard Univeristy for reading and commenting on my thesis. In addition, I would like to thank the members of MIT Lincoln Laboratory Group 44 for the assistance they have provided.

I would like to thank my father for believing in me all of these years, and for convincing me that I could handle anything the Institute could throw at me. I also would like to thank Gen for making everything just a little bit better.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The singular value decomposition (SVD) of an arbitrary complex matrix is a computationally intensive problem. However, this matrix factorization is useful in a wide variety of signal processing applications. As signal processing algorithms become more complex, new and different implementations need to be devised. Most SVD algorithms make assumptions about the input matrix which limit their flexibility. This thesis will develop a method for computing the SVD of a complex matrix that makes no assumptions about the input data. In addition, a VLSI design of this implementation on a systolic processor array will be presented.

1.1 Parallel Implementations of the SVD

The problem of computing the SVD factorization of a complex matrix requires a large number of computations. However, it is also an algorithm with a high level of inherent parallelism. This parallelism, which comes from the independent nature of the computations, allow it to be realized very efficiently on a massively parallel computation engine. Algorithms for calculation of the SVD in an orderly fashion have been around for a long time [6], but only since the advent of high performance, low cost specialized processor subsystems has the problem been addressed with a parallel implementation in mind [3]. These new parallel algorithms have given rise to several proposed designs for systems tailored for SVD computation.

Several implementations of parallel architectures for SVD have been suggested [3, 7, 4]. However, there may be difficulties with existing designs. [3] is intended for real matrices, and [7] and [4] make use of the CORDIC (COordinate Rotation Digital Computer) [20] engine, which is inefficient for longer word widths. In this thesis, the general methods detailed in [3] are expanded to accommodate complex matrices, and the computations are pipelined to increase efficiency. In addition, the SVD problem will be solved algebraically, without the need for direct implementation of trigonometric functions. The CORDIC engine used in [7] will be replaced by a novel fast interpolation cell that is capable of performing complicated functions in a time much less dependent on output word width, while maintaining accuracy. In addition, this design will be implemented entirely in a high level structural description language used as input to a silicon compiler system. This allowed the entire design to be parameterized and altered with minimal effort. The processor elements are designed to utilize the high level of abstraction afforded by the silicon compiler system; specifically, they may be used as frameworks for application specific digital signal processors. These nature of these designs allow additional functional units to be added or removed easily, making for high performance systems while maintaining an extremely low design time.

1.2 Systolic Processor Array

The nature of the SVD algorithm requires very little data for each calculation, and it can be implemented without the ability to move data quickly over large distances across the system. These qualities of the algorithm make it ideal for implementation in a systolic processor array. A systolic processor array is a mesh connected set of processors. The only data interconnections are to immediately adjacent nodes. The array is termed a systolic array because the data "pumps" through the system as if blood were pumping through a body. The array performance has no dependence on the overall size, due to the locality of the interconnects. Given multiple processors all connected to one central resource, as in a shared memory or a data bus, the overall size should be limited, as any increase necessitates an alteration in the design. Using a systolic array, all processor interconnection is to the nearest neighbor. Isolating each processor from the rest of the array allows the system to grow nearly without bound, with little or no change in the overall design.

1.3 Overview of the Thesis

In chapter 2, a two step process for computing the singular value decomposition of an arbitrary complex matrix is developed, including details on how it will be carried out in a parallel system. In chapter 3, the design of the systolic processor array is discussed, detailing the data flow through the array and necessary design issues and trade-offs. Chapter 4 will discuss the individual processor elements, including their functional units and architecture. Chapter 5 will examine the design of the fast interpolation functional unit and its benefits. Chapter 6 will discuss the LAGER silicon assembly system used for the realization of the design. The trade-offs of design time and performance will be considered. Chapter 7 will discuss the specific implementation of this system, and finally, chapter 8 will discuss possible improvements and future work.

Chapter 2

The Singular Value Decomposition

The singular value decomposition (SVD) is a matrix factorization technique that is of use in many areas of signal processing. SVD can be used to help reconstruct the electrical activity inside the brain using non-invasive electrodes placed on the scalp [14]. It is also used to decrease the data rate needed to transmit images [18]. For adaptive array signal processing, SVD can be used to separate the signal subspace from the noise subspace [15]. All of these applications involve reducing a given data matrix to one of smaller effective rank. The amount of information that is retained from the original matrix is determined by a chosen threshold level imposed upon the singular values of the matrix. Having chosen an acceptable cutoff threshold, the elements of the matrix that are below the threshold value can be discarded with little or no information loss. This method is extremely effective for distinguishing the important information in the matrix.

2.1 SVD of a Real, Symmetric Matrix

To begin, the Singular Value Decomposition of a real, symmetric matrix will be examined, because it is the simplest case. The singular value decomposition for a $n \times n$ real, symmetric matrix A is

$$A = U\Sigma V^T \tag{2.1}$$

where U and V are $n \times n$ orthogonal matrices, and Σ is a diagonal matrix of the singular values of A in non-increasing order down the diagonal. In the following, only 2×2 matrices will be considered. The cyclic Jacobi method detailed in [6] suggests a method for calculating the SVD of a $n \times n$ matrix by breaking it into a series of 2×2 matrices, so the SVD of only 2×2 matrices can be discussed without a loss of generality. Given that U and V are orthogonal matrices, (2.1) can be re-written as

$$U^T AV = \Sigma.$$

U and V need to be chosen such that $U^T AV$ yields a diagonal matrix Σ of the singular values of A. For a real, symmetric matrix, U and V are chosen to be real and equal. They are rotation matrices of the form

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}.$$

The problem is to choose a real angle, θ_{SVD}, such that

$$R(\theta_{SVD})^T AR(\theta_{SVD}) = \Sigma. \tag{2.2}$$

[3] gives a method of selecting θ_{SVD} given A.

However, what is really of interest is $\sin\theta_{SVD}$ and $\cos\theta_{SVD}$, not θ_{SVD} itself. If the matrix multiplication of (2.2) is carried out, with

$$A = \begin{bmatrix} p & q \\ q & r \end{bmatrix},$$

where $q \neq 0$, the result is

$$\begin{bmatrix} \cos\theta_{SVD} & \sin\theta_{SVD} \\ -\sin\theta_{SVD} & \cos\theta_{SVD} \end{bmatrix}^T \begin{bmatrix} p & q \\ q & r \end{bmatrix} \begin{bmatrix} \cos\theta_{SVD} & \sin\theta_{SVD} \\ -\sin\theta_{SVD} & \cos\theta_{SVD} \end{bmatrix} = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix},$$

where σ_1 and σ_2 are the singular values of A. Expanding this through multiplication, it is found for the bottom left element and top right element of Σ that

$$q(\cos^2\theta_{SVD} - \sin^2\theta_{SVD}) + (p-r)(\cos\theta_{SVD}\sin\theta_{SVD}) = 0. \tag{2.3}$$

To solve for $\cos\theta_{SVD}$ and $\sin\theta_{SVD}$, the substitution

$$t = \frac{\cos\theta_{SVD}}{\sin\theta_{SVD}},$$

is made, allowing equation (2.3) to b rewritten as

$$qt^2 + (p-r)t - q = 0.$$

Making the additional substitution

$$\rho = \frac{p-r}{2q}$$

allows (2.3) be further simplified,

$$t^2 + 2\rho t - 1 = 0.$$

Using the quadratic formula and trigonometric identities [3] to solve this eequation yields

$$t = -\rho \pm \sqrt{\rho^2+1} \quad \cos\theta_{SVD} = \frac{1}{\sqrt{1+t^2}}, \quad \sin\theta_{SVD} = t\cos\theta_{SVD}.$$

The two possible values for t correspond to the two possible rotation angles, one positive, one negative.

Now, the SVD of A is complete, with

$$U = V = \begin{bmatrix} \cos\theta_{SVD} & \sin\theta_{SVD} \\ -\sin\theta_{SVD} & \cos\theta_{SVD} \end{bmatrix}$$

and

$$\Sigma = UAV^T.$$

2.2 SVD of an arbitrary real matrix

In the SVD of an arbitrary real 2×2 matrix, U and V are no longer equal, however, U and V are still orthogonal rotation matrices. This means that two separate angles, θ_{left} and θ_{right}, must be chosen to satisfy

$$R(\theta_{left})^T A R(\theta_{right}) = \Sigma. \tag{2.4}$$

θ_{left} and θ_{right} are chosen directly, or, alternately, a one-sided rotation by a real angle θ_{sym} is chosen such that

$$R(\theta_{sym})^T A = B, \tag{2.5}$$

where B is real and symmetric. This allows the SVD algorithm to be performed on B as in (2.2). θ_{sym} is chosen such that the angle between the rows and the angle between the columns of A become equal. (2.5) can be rewritten as

$$R(\theta_{sym})^T A = \begin{bmatrix} \cos\theta_{sym} & \sin\theta_{sym} \\ -\sin\theta_{sym} & \cos\theta_{sym} \end{bmatrix}^T \begin{bmatrix} w & x \\ y & z \end{bmatrix} = \begin{bmatrix} p & q \\ q & r \end{bmatrix}.$$

Solving for q

$$q = x\cos\theta_{sym} - z\sin\theta_{sym} = y\cos\theta_{sym} + w\sin\theta_{sym},$$

and rearranging terms finds that

$$x\cos\theta_{sym} - y\cos\theta_{sym} = w\sin\theta_{sym} + z\sin\theta_{sym}.$$

By making the subsitutions

$$\sin\theta_{sym} = \sqrt{1 - \sin^2\theta_{sym}}$$

and

$$\rho = \frac{w+z}{x-y},$$

it becomes simple to find $\sin\theta_{sym}$, and therefore $\cos\theta_{sym}$.

$$\sin\theta_{sym} = \frac{1}{\sqrt{1+\rho^2}}, \quad \cos\theta_{sym} = \frac{\rho}{\sqrt{1+\rho^2}} \ .$$

This yields the one sided Givens rotation necessary to make A symmetric. So the SVD of an arbitrary, real 2×2 matrix can be expressed as

$$R(\theta_{SVD})^T R(\theta_{sym})^T A R(\theta_{SVD}) = \Sigma.$$

This can be rewritten as in (2.4), if $\theta_{left} = \theta_{SVD} + \theta_{sym}$, and $\theta_{right} = \theta_{SVD}$.

2.3 SVD of an arbitrary complex matrix

An approach similar to that suggested in [7] and [4] for computing the SVD of an arbitrary complex matrix is often chosen. Initially, a complex matrix

$$M = \begin{bmatrix} Ae^{i\theta_A} & Be^{i\theta_B} \\ Ce^{i\theta_C} & De^{i\theta_D} \end{bmatrix}$$

is first multiplied by a modified complex Givens rotation to zero out the bottom left element. The method of [7] has been slightly modified such that the top left element becomes real after the Givens rotation. To accomplish this, the modified complex Givens rotation is separated into a unitary transform, followed by a real Givens rotation. First, M is pre-multiplied by the unitary transform

$$T_\alpha^H = \begin{bmatrix} e^{i\theta_\alpha} & 0 \\ 0 & e^{i\theta_\gamma} \end{bmatrix},$$

where $\theta_\alpha = -\theta_A$, and $\theta_\gamma = -\theta_C$, yielding

$$T^H M = \begin{bmatrix} A & Be^{i(\theta_B - \theta_A)} \\ C & De^{i(\theta_D - \theta_C)} \end{bmatrix},$$

then a real Givens rotation, $G(\psi)$

$$G(\psi)_\alpha^T = \begin{bmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{bmatrix},$$

with $\psi = \tan^{-1}(\frac{C}{A})$, is performed. Given A and C, $\cos\psi$ and $\sin\psi$ are easily computed. They are

$$\cos\psi = \frac{C}{\sqrt{A^2+C^2}}, \sin\psi = \frac{A}{\sqrt{A^2+C^2}}.$$

The Givens rotation yields

$$M' = G(\psi)^T T_\alpha^H M = \begin{bmatrix} A' & B'e^{i\theta_{B'}} \\ 0 & D'e^{i\theta_{D'}} \end{bmatrix},$$

where

$$A' = A\cos\psi + C\sin\psi,$$

which is real. These two steps can be combined into one left-sided unitary transformation matrix $L(\psi, \theta_\alpha, \theta_\gamma)$

$$L(\psi,\theta_\alpha,\theta_\gamma)^T = G(\psi)T_\alpha = \begin{bmatrix} e^{i\theta_\alpha}\cos\psi & e^{i\theta_\gamma}\sin\psi \\ -e^{i\theta_\alpha}\sin\psi & e^{i\theta_\gamma}\cos\psi \end{bmatrix}.$$

$L(\psi, \theta_\alpha, \theta_\gamma)$ is considered to be a modified complex Givens rotation [4]. A standard complex Givens rotation takes this modified complex Givens rotation and premultiplies it by another unitary transform to produce real diagonal elements. For the purposes of making the matrix M entirely real, this step can be omitted.

Next, the right column of the matrix is rotated to make the top right element, $B'e^{i\theta_{B'}}$, real. The bottom right element remains complex, and only its angle is affected. Post-multiplying by a simple transformation matrix

$$T_\beta = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta_\beta} \end{bmatrix},$$

where $\theta_\beta = -\theta_{B'}$, yields

$$M'' = M'T_\beta = \begin{bmatrix} A' & B' \\ 0 & D'e^{i(\theta_{D'}-\theta_{B'})} \end{bmatrix},$$

Finally, the bottom row of M'' is rotated to make the bottom right element real. The bottom left element is unaffected, because it already equals 0. This is accomplished by pre-multiplying M'' by

$$T_\delta^H = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\theta_\delta} \end{bmatrix},$$

where $\theta_\delta = -(\theta_{D'} - \theta_{B'})$. This gives

$$T_\delta^H M'' = \begin{bmatrix} A' & B' \\ 0 & D' \end{bmatrix},$$

with A', B', and D' all real. Explicitly,

$$T_\delta^H G(\psi)^T T_\alpha^H M T_\gamma = \begin{bmatrix} A' & B' \\ 0 & D' \end{bmatrix}.$$

Then the SVD of an arbitrary real matrix proceeds as outlined above. Starting with

$$A = R_\delta^H g(\psi)^T T^H M R_\gamma,$$

section 2.2 shows how to find two more rotation angles, θ_{left} and θ_{right}, that will make A diagonal. All that remains is to arrange the singular values of A in non-increasing order. The resulting matrix, after $R(\theta_{left})$ and $R(\theta_{right})$ have been applied is of the form

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}.$$

If $\sigma_1 < \sigma_2$, then the matrix needs to be rearranged. This can be accomplished by pre-

and post-multiplying Σ by a transformation matrix

$$N = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

If $\sigma_1 \geq \sigma_2$, then N can be set to the identity matrix.

The complete SVD algorithm can be expressed as

$$R(\theta_{left_2})^H R(\theta_{left_1})^H M R(\theta_{right_1}) R(\theta_{right_2}) = \Sigma$$

where

$$\begin{aligned} R(\theta_{left_1}) &= T_\alpha G(\psi) T_\delta, \\ R(\theta_{left_2}) &= R(\theta_{sym}) R(\theta_{SVD}) N, \\ R(\theta_{right_1}) &= T_\gamma, \\ R(\theta_{right_2}) &= R(\theta_{SVD}) N. \end{aligned}$$

Using this method, the SVD of the complex matrix can be thought of as a pair of two-sided transformations. The first transform produces a real matrix, while the second set makes it diagonal. It should be noted that because all of the individual transformation matrices are unitary, the two-sided transformation matrices are also unitary, and therefore their products are unitary transformation matrices. U and V are therefore

$$R(\theta_{left_1}) R(\theta_{left_2}) = U,$$

and

$$R(\theta_{right_1}) R(\theta_{right_2}) = V,$$

giving the complete SVD as in (2.1).

2.4 SVD of an $n \times n$ Matrix

The singular value decomposition has been known for over 100 years, but only recently have efficient, stable algorithms for computation of the SVD of general matrices been introduced. One algorithm that has been examined is the cyclic Jacobi method [6]. The original algorithm was intended for finding the eigenvalues of symmetric real matrices. It was later modified to perform for the SVD of real, square matrices. Kogbetliantz [11] showed that it was straightforward to extend Jacobi's method to arbitrary square complex matrices.

The algorithm begins with an $n \times n$ matrix A. First, the pair of off-diagonal elements, a_{ij} and a_{ji}, that have the largest combined magnitude, with $i < j < n$, are chosen. Indices i and j are chosen such that

$$|a_{ij}|^2 + |a_{ji}|^2 = max\{|a_{pq}|^2 + |a_{pq}|^2\},$$

where p and q represent the index into the matrix for all possible combinations of p and q, with $i \neq j$ and $p \neq q$. This allows us to select largest off-diagonal matrix elements for elimination. To eliminate these elements, the matrix

$$\begin{bmatrix} a_{ii} & a_{ji} \\ a_{ij} & a_{jj} \end{bmatrix}$$

is considered. The SVD of this 2×2 matrix is performed, which creates zeros at a_{ij} and a_{ji}. In order to propagate the effects of the SVD of this submatrix, larger $n \times n$ rotation matrices are constructed from smaller 2×2 transformation matrices T, with

$$R_{pq} = \begin{cases} 1, & \text{if } p = q \text{ and } p, q \neq i, j, \\ T_{11}, & \text{if } p = q = i, \\ T_{12}, & \text{if } p = i \text{ and } q = j, \\ T_{21}, & \text{if } p = j \text{ and } q = i, \\ T_{22}, & \text{if } p = q = j, \\ 0, & \text{otherwise.} \end{cases}$$

The entire matrix A is then multiplied by the larger transformation matrices R. Then the next largest pair of off diagonal elements is chosen, and the process is repeated. The total magnitude of the off diagonal elements, given by

$$\sum_{p=1}^{n} \sum_{q=p+1}^{n} |a_{pq}|^2 + |a_{qp}|^2 \tag{2.6}$$

is reduced at each step. When this sum has fallen to below some preset threshold, the matrix is considered to be diagonal and the process stops.

There are a few problems with actually implementing this algorithm. First, it is necessary to examine $n(n-1)/2$ possible pairs of off-diagonal elements to select the best pair for elimination. In addition, since this algorithm iterates until the sum of the off-diagonals, as given in (2.6), has fallen below some acceptable threshold, the off-diagonal matrix elements must each be examined at every iteration to compute the sum of their magnitudes.

The first concern is dealt with by [3], where efficiency is sacrificed for completeness. What is suggested is to simply not consider which off-diagonal pair reduces the total sum the most. Every pair of off diagonals gets eliminated in turn. Because of the fact that after each pair of off-diagonals is eliminated, the sum of the magnitudes of the off-diagonal elements as given by (2.6) is either reduced or stays the same, the extra computations required using approach do not corrupt the matrix, however, more than the minimum number of computations will be performed. When this is compared to the extra complexity required to inspect the entire matrix, the suggested modification becomes an acceptable tradeoff. Each pass through the matrix eliminating every possible off-diagonal pair is considered to be a complete *sweep*.

The second concern is dealt with in [7]. While no proof for the exact rate of convergence of the off-diagonals has been offered, extensive simulations of the algorithm reveals that the required number of sweeps through the matrix is nearly constant for a large variety of input matrices. For instance, when matrices of sizes ranging from $n = 4$ up to $n = 100$ were simulated, the difference in the average number of sweeps and the maximum number of sweeps required was less than 1. It was also found that average number of sweeps required for convergence with $n = 100$ was only six more than the average number required with $n = 4$. This data shows that even for varying values of n, a fixed number of sweeps can be chosen for the array. By examining the number of sweeps needed for convergence through simulation with an upper bound for n, a fixed value can be chosen such that the SVD will complete nearly all the time. Due to the invariance of the number of sweeps required for convergence for different values of n, n can be allowed to vary, within limits, with convergence nearly guaranteed. In addition, if the matrix has not finished converging to the preset threshold when the fixed number of sweeps is complete, the off-diagonal elements will have been steadily reduced througout the computations, and the values on the diagonal elements will be close approximations of their final values. That is, they will not be completely unrelated interim results. This means that if this approach is employed, a near constant processing time is achieved in exchange for possibly having a slight error introduced in certain cases, making it useful in real-time processing applications.

2.5 Parallel Computation of the SVD

An advantage of the cyclic Jacobi method is that with each elimination of an off-diagonal pair, only the rows and columns that contain the pair are affected. By exploiting this fact, the modification suggested by [3], where every possible pair is eliminated, becomes significantly less time consuming to implement than if each elimination is performed sequentially. Because only the two rows and two columns that contain the pair are affected by each elimination, up to $n/2$ eliminations can be performed simultaneously. The time for a complete sweep through the matrix is now, at best, $n - 1$ time steps, where each elimination requires one time step. This reduces the time required from $O(n^2)$ to $O(n)$.

A parallel ordering scheme is suggested in [2] that can be used to maximally exploit the parallelism possible through this algorithm, insuring that it only takes $n - 1$ time steps to complete a sweep. This parallel ordering is illustrated for the $n = 8$ case:

(1,2)	(3,4)	(5,6)	(7,8)
(1,4)	(2,6)	(3,8)	(5,7)
(1,6)	(4,8)	(2,7)	(3,5)
(1,8)	(6,7)	(4,5)	(2,3)
(1,7)	(5,8)	(3,6)	(2,4)
(1,5)	(3,7)	(2,8)	(4,6)
(1,3)	(2,5)	(4,7)	(6,8)

where (i, j) means that an elimination of a_{ij} and a_{ji} takes place.

This ordering naturally suggests an implementation on a parallel array of $n \times n$ processing elements, however, the amount of communication required to realize this implementation is enormous. At some point, each processor has to communicate with every other processor. It is unwieldy to implement an array in which each processor has a direct communication link to every other processor for any reasonably sized array. The alternative is fixed communication paths, but this leads to long communication latencies, requiring excessive times for messages to pass from one side of the array to the other.

This implementation also requires that every processor element have equal computation ability. However, only a small fraction of the time will any given processor element have to perform difficult computations, such as those required to generate the rotation angles. The majority of the time, a processor is applying an angle calculated by some other processor in its row or column. As a consequence, complex functional units are required for all processors, but are idle most of the time. An alternative architecture has been suggested that moves the matrix in the array to simulate the new selection of (i, j) pairs, while allowing the actual processor elements that calculate the rotation angles to be the same in each time step [3]. Therefore, only $2n$ processor elements need the complex functional units required for computation of the rotation parameters, and the remaining elements only need simple multiplication and addition units to apply the rotation angles. The drawback of this arrangement is that it becomes difficult to arrange the eliminations such that the singular values of the larger matrix are in non-increasing order down the diagonal when computation stops because it is difficult to predict where a given data element will be after each shift of the data. It becomes easier to reorder the matrix after the computation of the singular values has finished.

A possible implementation of this algorithm is described in [3]. A more thorough implementation is described in [7]. An approach similar to that of [7] is taken, with modifications to increase the efficiency of the algorithm and some changes to increase the performance of the individual processor elements. Many parallel implementations of this architecture have been suggested [2, 16, 13], but most attempt to map the algorithm onto existing hardware, similar to the simulations run in [7] on the Connection Machine CM-5. To maximize the performance using a real time system, specialized hardware must be designed, and the algorithm must be modified by shifting the data to perform the complex angle calculations using a few, fixed processor elements.

Chapter 3

Systolic Processor Array

The systolic array represents a unique balance between the possible limitations of finite processing power and finite communications capability. Too often, a large problem is tackled by simply throwing more computational units at it, with little heed to the problem of moving data around. However, at this point in time, processing power has become inexpensive enough that it is not the limiting factor. No matter how fast a processing element is, it cannot realize its full potential if it does not have any data to work on. The systolic array takes both factors into account, and attempts to achieve an optimal balance, so that the system's resources can be efficiently used.

3.1 The Systolic Array

The notion of the systolic array was first developed at Carnegie-Mellon University in the late 1970's [12]. The systolic array is a number of relatively simple processing elements connected together, usually in a linear or mesh-like pattern. Each processing element communicates only with its nearest neighbor. This arrangement means that the communications load is distributed throughout the array. As a consequence, it also means that only certain types of problems are amenable to implementation on a systolic array. The problem must be one where a fixed amount of data is applied against another fixed amount of data repetitively. An example of this type of problem would be one-dimensional convolution [12]. This problem is well suited for implementation using a linear array of processors. For this problem, a vector of data, X is convolved with a vector W of weights, and the result Y is given by

$$y_n = \sum_{m=0}^{n} x_m w_{n-m}.$$

Each element of X is eventually multiplied by every element of W. As a consequence, a relatively small amount of data goes into the array, and a small amount of data goes out of the array, but in order to perform the calculations, a large amount of data must flow throughout the array.

One possible way to implement the convolution is to move the weights and the data vectors linearly through the array in opposite directions, multiplying them when they meet, and to have the resulting sums stay in each systolic processor element. Each weight sees each data point as they pass by each other, and each processor element accumulates the sum of each datum multiplied by each weight. After the weights and the data are moved through the array, the convolution vector Y remains to be shifted out of the processor elements.

It is important to note that only certain problems can be mapped efficiently into systolic arrays; they are meant to balance communication bandwidth with processing power. If the problem requires more of one or the other, then a systolic array will be inefficient. Fortunately, singular value decomposition can be mapped very efficiently onto a systolic array. To implement the singular value decomposition on a systolic array, the problem must be broken down into a series of smaller problems. Using the method for computation of the SVD given in the previous chapter, the problem can be implemented as a two-step process. First, the rotation angles needed to perform the SVD must be calculated, and then applied to the rows and columns of the particular submatrix being diagonalized. To calculate the rotation angles, the processor only needs access to the four elements of the submatrix being worked on. After calculating the rotation angles, it can then pass these values on to other processors that are handling the problem of applying the rotation matrices to the rest of the rows and columns. Remembering that the problem can be arranged such that only a few processors need to be able to calculate rotation parameters, this would suggest a possible architecture in which the diagonal processor elements can calculate the rotation parameters, and the off-diagonal processors apply them.

3.2 The Brent-Luk-Van Loan Systolic Array

A systolic array for computation of real SVD has been suggested in [3]. This approach starts with an array of processors that is $n/2 \times n/2$ in size. Each processor, P_{ij}, operates on a small submatrix

$$\begin{bmatrix} \alpha_{ij} & \beta_{ij} \\ \gamma_{ij} & \delta_{ij} \end{bmatrix}.$$

Initially, the processor element P_{ij} is loaded with a submatrix of A

$$\begin{bmatrix} a_{2i-1,2j-1} & a_{2i-1,2j} \\ a_{2i,2j-1} & a_{2i,2j} \end{bmatrix}.$$

The diagonal processor elements contain the necessary hardware to calculate the rotation parameters, and the off-diagonal elements only contain enough processing capability to apply these parameters.

When computation begins, the processors on the diagonals of the array, $P_{ii}, 1 \leq i \leq n/2$, compute the rotation angles necessary for diagonalization of their data matrix. These rotation matrices affect only the data values in the same row and column of the array as the submatrix being diagonalized. By utilizing the communication links suggested in [3], it is possible to move the data through the array to maintain this special relationship between the data in the diagonal processor elements and that in the off-diagonal elements, while carrying out the suggested parallel ordering to maximize computational efficiency. These data flow interconnections can be seen in figure 3-1. Recalling that parallel ordering coupled with the suggested data flow paths allowed all angle computations to be carried out in the diagonal processor el-

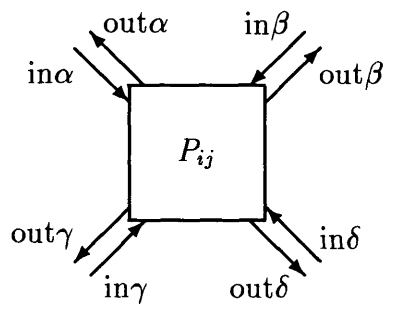

Figure 3-1: Dataflow Connections for Each Processor Element

ements, the processors on the diagonal of the array are more important than those above and below the diagonal. In fact, the off-diagonal processor elements become slave processors, applying values generated elsewhere to the data values they contain. The angle parameter connections for the off-diagonal processors are illustrated in figure 3-2, and the connections for the diagonal processor elements are depicted in figure 3-3. As shown, the angle connections become very different for the off-diagonal processor elements and the diagonal processor elements. It should be noted that the sub-diagonal processor elements, P_{ij} with $i > j$, have just the mirror image of the super-diagonal processor element, P_{ij} with $i < j$, connections. This set of diagonal

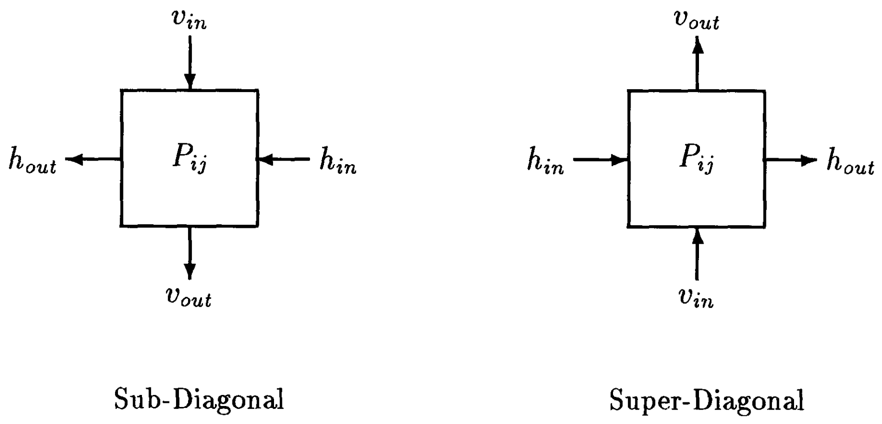

Figure 3-2: Rotation Parameter Flow for Off-Diagonal Processor Elements

connections represent the complete set of data pathways needed in the array. When combined, a schematic representation of the data connections for the off-diagonal pro-

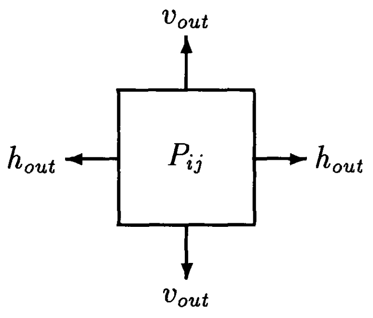

Figure 3-3: Rotation Parameter Flow for Diagonal Processor Elements

cessor elements is produced, as in figure 3-4 and for the diagonal processor elements in figure 3-5. These connections are assembled into a representation of the complete

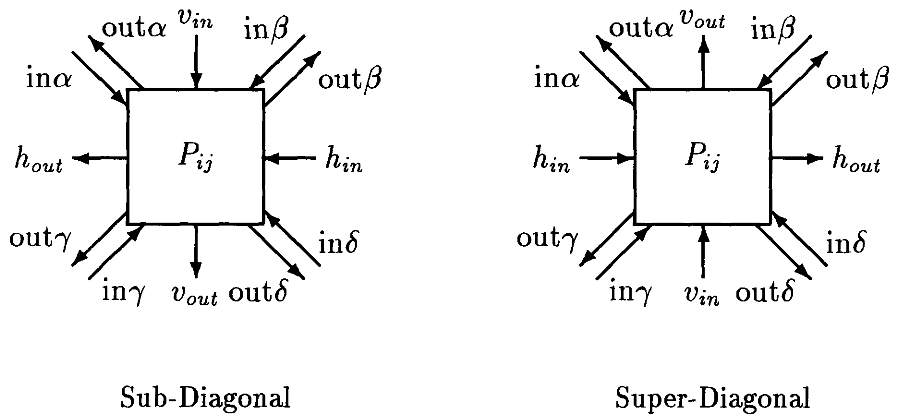

Figure 3-4: Complete Interconnections for Off-Diagonal Processor Elements

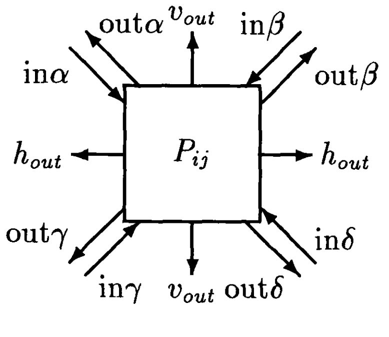

Figure 3-5: Complete Interconnections for Diagonal Processor Elements

array. The processor interconnections at the edges of the array were handled in a slightly different way to tie up all loose connections. Formally, these interconnections can be represented by

$$\text{out}\alpha_{y,x} \Leftrightarrow \begin{cases} \text{in}\alpha_{y,x}, & \text{if } y=1,\ x=1; \\ \text{in}\beta_{y,x}, & \text{if } y=1,\ x>1; \\ \text{in}\gamma_{y-1,x}, & \text{if } y>1,\ x=1; \\ \text{in}\delta_{y-1,x-1}, & \text{if } y>1,\ x>1. \end{cases} \qquad \text{out}\beta_{y,x} \Leftrightarrow \begin{cases} \text{in}\alpha_{y,x+1}, & \text{if } y=1,\ x<\frac{n}{2}; \\ \text{in}\beta_{y,x}, & \text{if } y=1,\ x=\frac{n}{2}; \\ \text{in}\gamma_{y-1,x+1}, & \text{if } y>1,\ x<\frac{n}{2}; \\ \text{in}\delta_{y-1,x}, & \text{if } y>1,\ x=\frac{n}{2}. \end{cases}$$

$$\text{out}\gamma_{y,x} \Leftrightarrow \begin{cases} \text{in}\alpha_{y+1,x}, & \text{if } y<\frac{n}{2},\ x=1; \\ \text{in}\beta_{y-1,x+1}, & \text{if } y<\frac{n}{2},\ x>1; \\ \text{in}\gamma_{y,x}, & \text{if } y=\frac{n}{2},\ x=1; \\ \text{in}\delta_{y,x+1}, & \text{if } y=\frac{n}{2},\ x>1. \end{cases} \qquad \text{out}\delta_{y,x} \Leftrightarrow \begin{cases} \text{in}\alpha_{y+1,x+1}, & \text{if } y<\frac{n}{2},\ x<\frac{n}{2}; \\ \text{in}\beta_{y+1,x}, & \text{if } y<\frac{n}{2},\ x=\frac{n}{2}; \\ \text{in}\gamma_{y,x+1}, & \text{if } y=\frac{n}{2},\ x<\frac{n}{2}; \\ \text{in}\delta_{y,x}, & \text{if } y=\frac{n}{2},\ x=\frac{n}{2}. \end{cases}$$

A complete 4×4 array utilizing these interconnections can be seen in figure 3-6. For

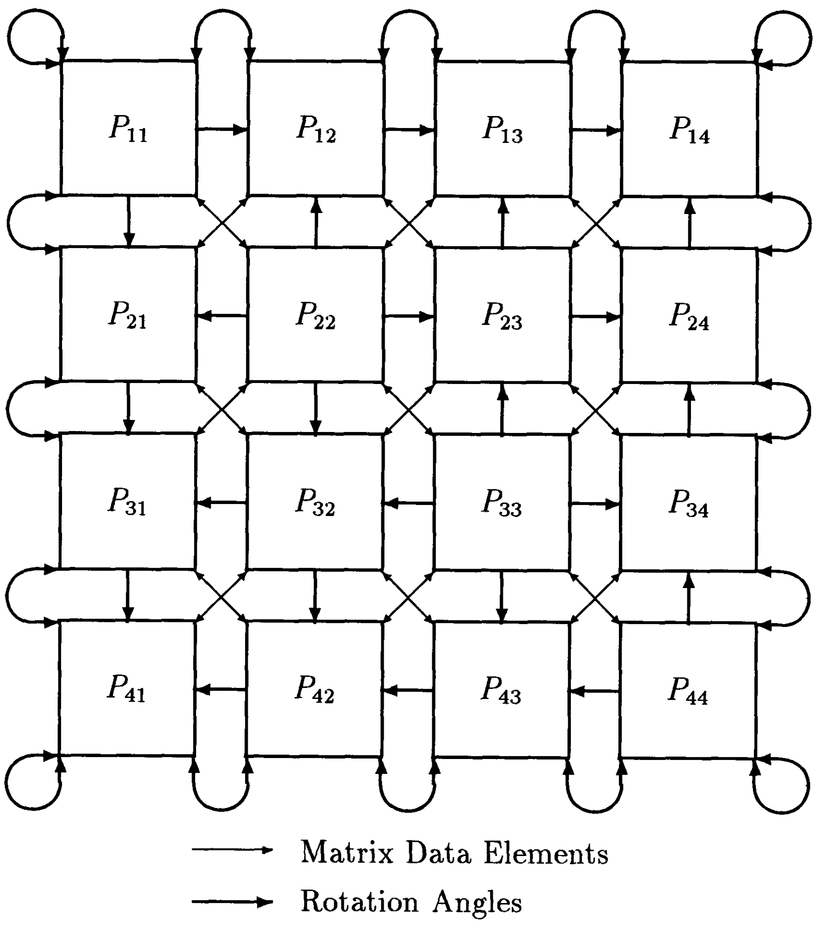

Figure 3-6: Example Systolic Array

the array with $n = 8$, one sweep through the array requires 7 time steps to complete.

In order to move the data to take advantage of the parallel ordering scheme being used, data must be exported out of the processing nodes according to the interchange algorithm presented in [3]. At the end of each time step, the data in each processing element is made available on its outputs for the adjacent processor elements according to:

$$\begin{array}{cc}
\text{if } y = 1 \text{ and } x = 1 \text{ then} & \left[\begin{array}{cccccc} \text{out}\alpha & \leftarrow & \alpha & \text{out}\beta & \leftarrow & \beta \\ \text{out}\gamma & \leftarrow & \gamma & \text{out}\delta & \leftarrow & \delta \end{array}\right] \\
\text{else if } y = 1 \text{ then} & \left[\begin{array}{cccccc} \text{out}\alpha & \leftarrow & \beta & \text{out}\beta & \leftarrow & \alpha \\ \text{out}\gamma & \leftarrow & \delta & \text{out}\delta & \leftarrow & \gamma \end{array}\right] \\
\text{else if } x = 1 \text{ then} & \left[\begin{array}{cccccc} \text{out}\alpha & \leftarrow & \gamma & \text{out}\beta & \leftarrow & \delta \\ \text{out}\gamma & \leftarrow & \alpha & \text{out}\delta & \leftarrow & \beta \end{array}\right] \\
\text{else} & \left[\begin{array}{cccccc} \text{out}\alpha & \leftarrow & \delta & \text{out}\beta & \leftarrow & \gamma \\ \text{out}\gamma & \leftarrow & \beta & \text{out}\delta & \leftarrow & \alpha \end{array}\right]
\end{array} . \tag{3.1}$$

Then, after the outputs have propagated to adjacent processors,

$$\left[\begin{array}{cccccc} \alpha & \leftarrow & \text{in}\alpha & \beta & \leftarrow & \text{in}\beta \\ \gamma & \leftarrow & \text{in}\gamma & \delta & \leftarrow & \text{in}\delta \end{array}\right] .$$

This details how data flows in the fully systolic Brent-Luk-Van Loan systolic array.

3.3 Changes to the Brent-Luk-Van Loan Array

The systolic array for real-valued SVD presented in [3] has been used a starting point and extensive modifications have been applied to it. Such modifications were necessary because the original design was intended for real-valued matrices. Most importantly, by expanding this array to handle complex-valued data elements, the SVD algorithm has changed from a single cycle to a two cycle algorithm. Originally, the complete SVD of the 2×2 submatrix contained in each processor was completed prior to any data interchange. In the array for complex SVD, the angles used to make the submatrix real can be made available prior to the entire SVD completing. In addition, the original array was fully systolic; that is, each processor was connected *only* to its nearest neighbors. This is not absoultely required, as long as care is taken when making compromises to not destroy the overall systolic nature of the array.

To improve performance, a compromise was made in that a fully systolic implementation, in this case, would introduce unnecessary delays. It was decided that the rotation angles generated by the diagonal processor elements would be broadcast throughout their rows and columns. In the original array, there was a long delay associated with propagating rotation parameters to off-diagonal nodes, applying those rotations in the off-diagonals, and then propagating the resultant data back to the diagonal element. Only after this could new rotations be started. As a consequence, each processor element was active for only one in three cycles, with each step in a sweep of the array being three cycles long. This is a major inefficiency of the Brent-Luk-Van Loan array.

Complex SVD is a two-stage process which allows some pipelining to take place in the interest of streamlining computations and increasing efficiency. In [7], the two-step SVD is arranged so that each processing element is active for two cycles out of four, with each step of a sweep being comprised of 4 cycles. Even with this advancement, the efficiency of the array is 50%. It is this efficiency, coupled with external design considerations, that necessitated the broadcasting of the rotation parameters.

The major disadvantage of a partially systolic architecture to a fully systolic one is that the partially systolic array has some limitations on the maximum size to which it can grow. When real-world external design considerations are taken into account, other, possibly more severe limitations may be placed upon the array size. Without any external hardware, a fan-out of 10 from the diagonal processors to the off-diagonal processors is acceptable. Thus, an 11×11 array would be possible without the need for any external logic. This processor array could handle data matrices up to 22×22 in size, having 121 processing elements. Assuming that each processing element costs approximately \$500 yeilds a total array cost of over \$60,000. With the addition of one additional buffer per diagonal element, effectively quadrupling the limiting size of the array, the cost for the complete system grows to nearly \$250,000. In addition, as the data arrays become large, the amount of time needed to complete each sweep, and therefore to diagonalize the entire matrix, grows to be prohibitively long for use in real-time systems. It is these real-world design limitations, in addition to the increased performance, that makes the decision to sacrifice the fully systolic nature of the array suitable.

It should be noted, however, that the individual processor elements have been designed in such a way that they do contain all of the necessary hardware for a fully systolic implementation. Only the internal programming of the processors would need to be changed, in addition to the timing control and the wiring of the array. The fully systolic implementation requires a different number of cycles per sweep step, and a different amount of time per cycle. The details of the data interchange in a two-step, complex-valued, fully systolic array are given in [7].

Knowing that the rotation parameters will be broadcast throughout the array, the processor cycles can be optimized in such a way that the overall efficiency of the array increases to each processor being active for two cycles out of three, with each step in a sweep through the matrix being comprised of three cycles. The actions of each processor during the three cycles are summarized in table 3.1. The off-diagonal

Cycle	Diagonal processor	Off-Diagonal processor
1	Calculate angles to make matrix real	Idle
2	Calculate angles for real SVD	Apply angles that made submatrix in diagonal processor real
3	Finish applying angles for real SVD	Apply angles from diagonal processor real SVD

Table 3.1: Processor Cycle Usage

processors are only active for 2 out of 3 cycles. Because there are $n(n-2)/4$ more off-diagonal than diagonal processors, this limits the efficiency of the array.

The algorithm used in the processor elements is a direct adaptation of the algorithm given in section 2.3. For the first cycle, the diagonal processor elements calculate the component angles of θ_{left1} and θ_{right1}, which are the angles needed to make the matrix real and upper triangular. Next, these angles are made available to the off-diagonal processors, which were idle during the first cycle. During the second cycle, the diagonal processor elements first compute the angles needed to make the real, upper diagonal matrix symmetric, and then the angle needed to make it diagonal. These angles are then broadcast. During this cycle, the off-diagonal processors apply the rotation angles previously received. For the third cycle, the diagonal processor elements finish applying the rotation angles needed to make the matrix diagonal. To make all of the rotation angles available as early as possible, use of the rotation angles in the diagonal elements is delayed until the third cycle. At this point, the off-diagonal processors also make use of these angles, applying them to the rest of the matrix. At the end of the third cycle, the data matrix is interchanged between processors as given in the interchange algorithm in (3.1).

Following the data interchange, the process repeats for a specified number of sweeps through the array. At this time the off-diagonals have converged to values below a certain preset threshold. The data is available to be unloaded from the array. To rapidly load and unload the array, a second set of interconnections, along the columns of the array, have been inserted. When the array is to be initially loaded, the data enters the top of the array and is passed down the array by rows until the array is full. After the array is loaded, the data connections down the columns are no longer used, and remain dormant until the processing is complete, at which time the data is unloaded out the bottom of the array. Separate communications channels were provided for loading and processing due to the fact that the method by which the data is inserted into and extracted from the array is implementation dependent, and the flexibility of the array is compromised by assuming that certain pathways always exist. Therefore, having extra channels for loading and unloading was considered an acceptable design tradeoff because this allows the core processor elements to remain unchanged for a variety of implementations.

Although the unitary matricies U and V from (2.1) are being generated, they are not being saved. Only the final result matrix, Σ, is retained. It would be simple to construct additional hardware that attaches to the edges of the array and collects the rotation matrices being generated by the diagonal processors. These matrices can be multiplied together to recover U and V.

Chapter 4

Systolic Processor Elements

The highest possible computational throughput while maintaining a reasonable cost are the primary design goals of the processor elements used in the systolic array. When considering the compute engines used in the array, the assumption made is that each node has a specified number of communication channels for transmission of the matrix data rotation angles. The functionality of the processors has been dictated by the mathematics of the algorithm, however, the implementation details have not been restricted beyond this criteria.

4.1 Processor Element Design

There are two major design methods that might have been used to implement the individual processors: they could have been constructed from off-the-shelf parts like commercial digital signal processors, or they could be designed using custom VLSI parts. In order to meet the design goals of the system, it was found that implementation of the individual nodes in commonly available commercial parts would have been complicated and expensive. The individual nodes would have needed a DSP in addition to external hardware to handle the I/O requirements and program store. This would have increased the cost and complexity. In addition, the algorithm calls for complicated functions, in the form of trigonometric routines or square roots. Existing DSP's do not implement these algorithms in hardware, so their computation is accomplished using a multi-cycle iterative routine, which adversely affects performance. In addition, a DSP includes some features that would go unused in the systolic architecture.

By implementing the design in custom VLSI, the processor can be made algorithm specific. Functional units can be added or removed depending on whether or not they are needed. In addition, different functional units that offer varying levels of performance while maintaining the same functionality can be used, allowing for performance tradeoffs in the interest of reducing cost. With the common availability of high-level design tools, and accessibility to fabrication resources, the total cost for a VLSI implementation is lower than for a DSP-based system.

In addition to the price/performance advantage that VLSI implementations can offer, there is a great deal of flexibility in the design details of the processor. For example, the processor can be designed to use an arbitrarily large word size. The algorithm in question can be simulated to determine the amount of round-off error introduced through the computations, and the processor word size can be adjusted accordingly. Commercial DSP's are available in a few word sizes, usually 16 or 24 bit, and beyond that, they typically use floating point representations. In a design such as this, to have 24 bits of accuracy in the final result may be require a 26 bit internal representation. In custom VLSI, this is simple to realize. In a commercial DSP, it is often necessary to use double length representation, which invariably requires more computational time to implement. The VLSI implementation allows a previously unavailable match between the hardware and the algorithm.

4.2 Functional Unit Design

To calculate the singular value decomposition, it is necessary to calculate certain trigonometric functions. The algorithm presented in section 2.3 presents an implementation that trades these trigonometric functions for square roots and inverse square roots. This tradeoff was made in the interest of simplicity. Making this change allows the algorithm to deal explicitly with the sine and cosine of the rotation angles and not the angles themselves. The angles are not needed, so it would be inefficient to calculate them. Also, by broadcasting the sine and cosine of the rotation angles instead of just the angles, the off-diagonal processor elements do not need complicated functional units. They implement a matrix multiplication, which only requires multiplication and addition. Simplicity in the off-diagonal processors has been traded for communication bandwidth, since twice as much data is now being sent from each diagonal processor.

The decision to calculate the SVD algebraically, as opposed to trigonometrically, dictated a change in the functional units. There have been many designs presented for efficient computation of the inverse and the square root [10, 8, 5, 9]. Expanding on these works, a fast interpolation cell was constructed for computing either the square root or the inverse square root of a value in 6 clock cycles, which is noticeably faster than any other multi-cycle method. Extensive research has gone into numerical methods for these computations. Iterative routines require a non-constant time for completion, making instruction scheduling difficult. Having a constant-time functional unit for these calculations allows the processor central controller to be simpler, and therefore smaller.

Another advantage to this cell is that most trigonometric functions can be implemented in a reasonable time with the addition of external hardware that would be present in almost any design, such as a multiplier. The interpolation cell adds to the flexibility of the design; it could be re-used in another system that had no need for trigonometric functions, or, by making use of the inverse square root capability and a multiplier, it could be used as a divider. One of the design goals of this system was to allow the functional units to be useful in other designs, and the interpolation cell was found to be the most flexible.

Other implementations of the SVD on systolic processor arrays [7, 4] make use of CORDIC [20] functional units. The CORDIC algorithm is primarily used for direct implementation of geometric problems. It can calculate sines, cosines, tangents, arc sines, arc cosines, arc tangents, products, square roots, and quotients. However, to implement each function, a particular CORDIC unit must include hardware specific to the computation. To calculate a value, the CORDIC algorithms iteratively converge to the desired result at a linear rate of one bit of output per time step. This means that for each angle calculation using 24-bit internal word representations, it takes a full 24 clock cycles to complete. As the word size increases, the amount of time required for the computations increases linearly. While the CORDIC algorithms represent a common set of algorithms for direct implementation of a wide variety of functions, improved performance is achieved by using a smaller number of optimized cells that implement specific functions.

In comparison, the functions that the CORDIC engines calculate can be simulated by a few more cells in less time, and the performance benefit of the interpolation cell grows as the desired word size increases. The amount of time required for computation of functions in the interpolation cell increases at a rate significantly less than linear as word size increases. The interpolation cell is discussed in chapter 5. The CORDIC implementations may be less complicated, using more regular design, but the amount of area they require is comparable to the area required for the interpolation cell.

While the CORDIC engine can perform direct computation of the square root, however, it is much slower due to the performance limitations inherent in its design. The amount of time required to perform the square root of a reasonably large word can be greater than the time required for a general-purpose DSP which uses an iterative method. While the CORDIC algorithms present methods for making very powerful functional units, the flexibility they offer is outweighed by the amount of time they require for processing.

4.3 Processor Element Architecture

One of the key issues in the design of the processor elements was for the implementation to be useful as a flexible framework for a variety of algorithm specific processors. They were designed to make no assumptions about the data flow through the processor that might limit their flexibility. This regular design of the processor elements also makes the inclusion or exclusion of functional units simple, yielding an open architecture.

The architecture that was used required that all functional units be connected to at least one of two operand busses, and one result bus. Data storage is handled through a register file. A functional unit works by reading one or two parameters off of the operand busses, and presenting its result on the result bus. By enforcing a standard interface for all functional units, it becomes easy to make changes in the set of functional units contained in a given processor. Although some efficiency may be lost through requiring every result to be written back into the register file, any performance gain achieved by providing short cut paths between the functional units would be outweighed by this limitation on the design's flexibility and generality.

The basic blocks used in these processors included complex multipliers, complex adders, register files, interpolation cells, barrel shifters, and off-chip input/output blocks. The block-level design for the off-diagonal processor can be seen in figure 4-1 and for the diagonal processor in figure 4-2, where the A and B busses represent the operand busses and the E bus represents the results bus. The only part of the processors that depends on which functional units are installed is the central controller. This unit must be designed with every change to the processor. However, by using a high-level description of the controller's functionality in a form closely resembling assembly language, the effort required to implement changes is minimal.

As can be seen in figures 4-1 and 4-2, the functional units of the off-diagonal processor are a subset of those in the diagonal processor. The difference between the two processors is in the controller. This similarity can be used in an attempt to keep prototyping costs down. One chip can be designed that implements the functionality of both processors. The controller unit could be modified to examine some external state input to tell the processor whether or not it is on a diagonal of the array. If it is, it runs the diagonal processor routines, and if it is not, it behaves as an off-diagonal processor, and does not use its interpolation cell or its barrel shifter. This single part is useful for small prototyping runs, as only one processor needs to be fabricated.

Having the controller for the processor elements as a separate unit, as opposed to being distributed throughout the design, allows more freedom in the design of

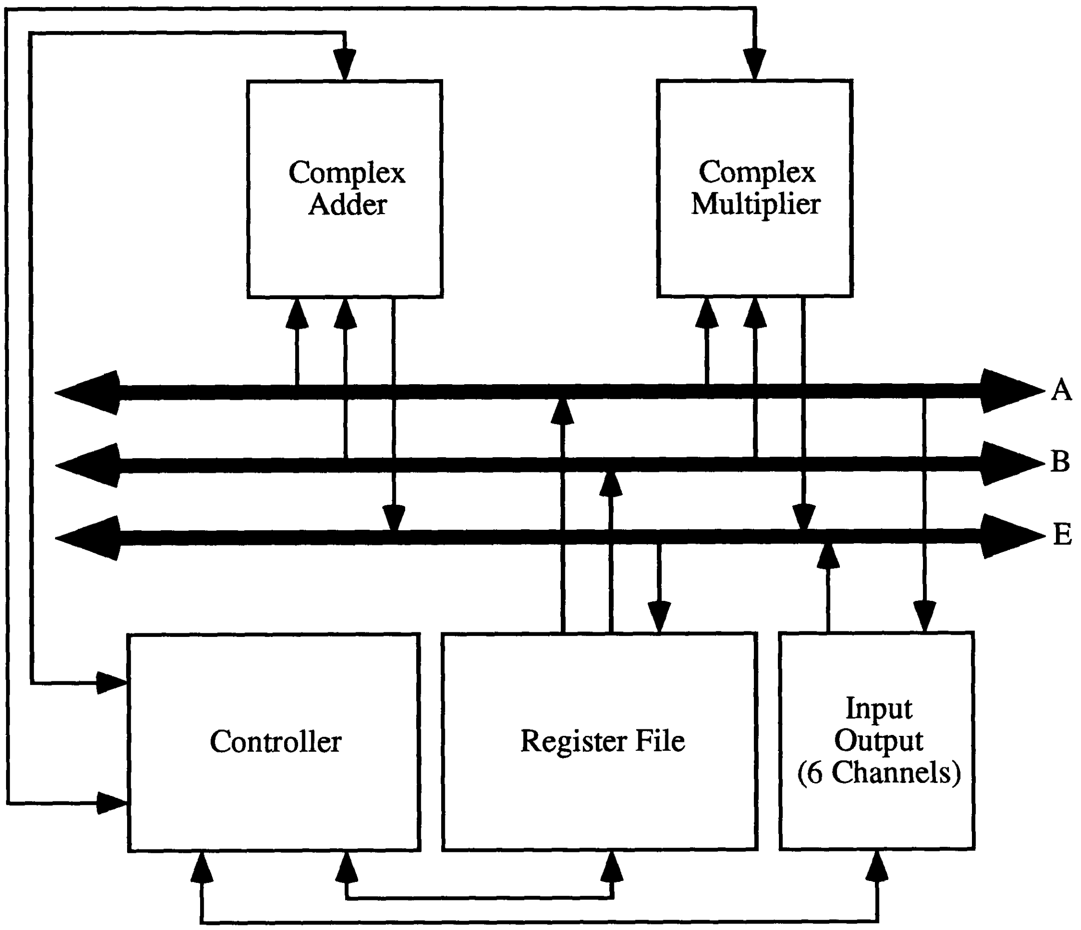

Figure 4-1: Block Diagram of Off-Diagonal Processor Element

the controller. It is currently implemented as a finite state machine, using a PLA lattice to store the program. However, the program can be implemented in the form of an internal or external program store that could be dynamically loaded. In this configuration, which would not require extensive changes, the processor could function as a small, application-specific digital signal processor. The particular algorithm could be loaded at run time, and could be changed at any time, even when the processor is running. Thus, a new class of custom DSP's could be created where the designer is free to specify the capabilities necessary for a particular application. It would not be limited to an application-specific integrated circuit, because it would be programmable.

Additional flexibility is gained by having all of the functional units share inputs and outputs. This insures that only the necessary control information was required

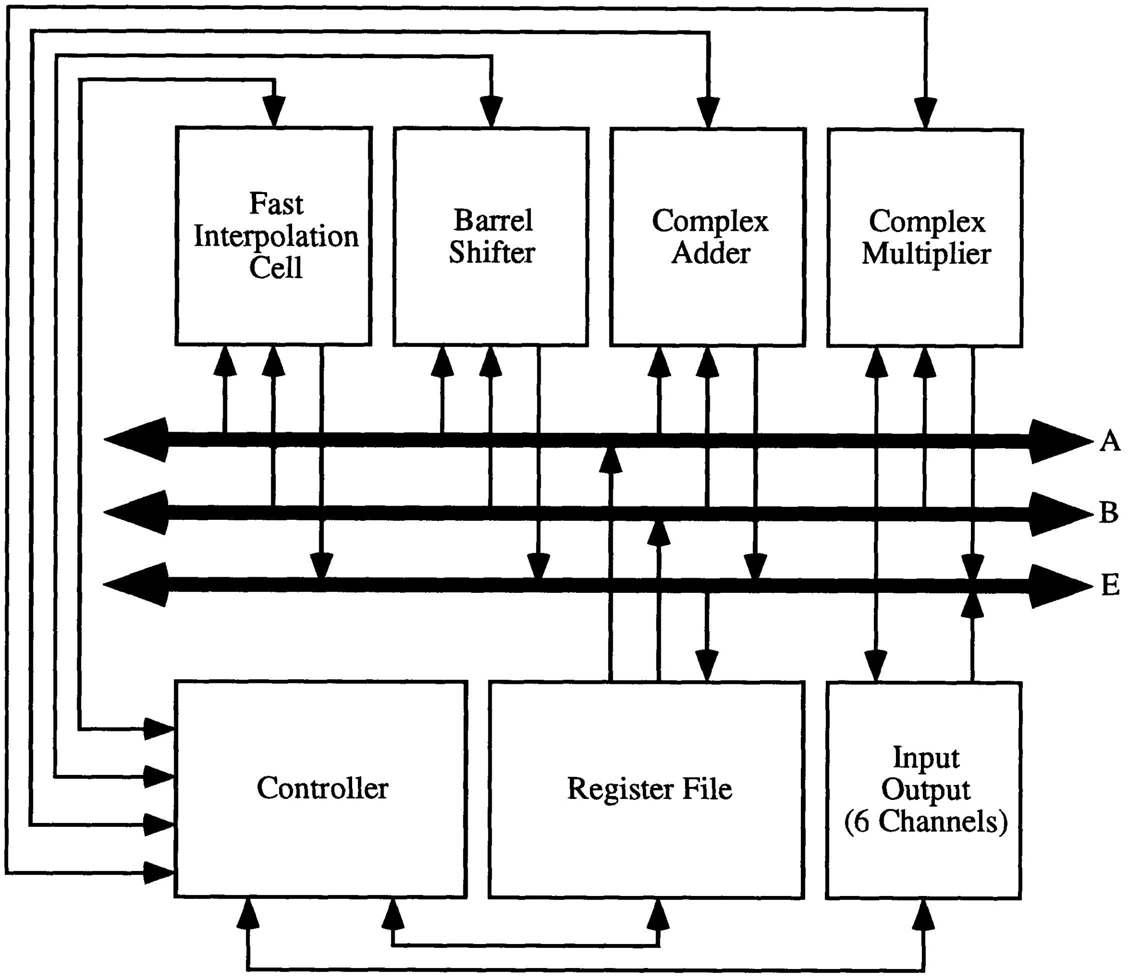

Figure 4-2: Block Diagram of Diagonal Processor Element

at the inputs to the cell. For example, if true and complement values of a control signal were required, only the true signal is propagated; the complementary signal is generated locally. While this may have made for more complicated functional units, with each unit typically requiring a small section of random logic used for control signal generation, it made for a more standard interface. The flexibility afforded by this is that multiple versions of the same functional unit can be created, with differing levels of performance and size. For instance, a complex multiplier might have a `start` control input, a `busy` control output, two operand inputs `A` and `B`, and one result output `E`.

Three different multipliers were created in the course of this design. One design used the fastest available cells, which used large numbers of parallel carry paths and other speed optimizations. The second model used more space conservative design techniques, but was essentially the same architecture, with the first two performing complex multiplication by using four real multipliers and two real adders, allowing for a complete complex multiplication in a single cycle. The third design, made further tradeoffs in an attempt to minimize area. It used a single multiplier of the type used in the second model, and two real adders. It had registers to hold the temporary values, and it required four clock cycles to complete a computation. It multiplied the real part of `A` first by the real part of `B` and saved it, then by the imaginary part of `B` and saved that. Next it repeated these steps with the imaginary part of `A`. Finally, it summed the real parts and the imaginary parts.

By requiring a standard interface with `start` and `busy` inputs, multi-cycle computations are provided for. In the first two multipliers, the `busy` input could just be tied to the `start` input. In the third multiplier, the controller that handled the multiplications would also have to generate the `busy` signal given the `start` signal. Requiring a least common denominator set of interface signals can cause some inefficiencies, but at the same time, it provides a level of abstraction for the designer that makes it much simpler to meet design objectives about performance and cost. Since the three multipliers all shared a common interface, they could be used interchangeably, depending on the area and time requirements of the system. The first multiplier was an order of magnitude faster and larger than the third, which shows that there can be a wide variance in performance and size, with identical functionality. The layout area and computation times for these three multipliers can be seen in table 4.1.

Multiplier	Multiply Time (nsec)	Area (mm^2)
1	40	124.98
2	80	45.90
3	350	13.67

Table 4.1: Comparison of Three Different Complex Multipliers

Every attempt has been made to provide hierarchical levels of abstraction throughout the design of the processors. Individual functional units can be hand optimized for the best performance, with no change required in the overall design. As much of the design as possible has been automated, so that the designer can focus on what is needed, as opposed to how to get it. However, some of this abstraction can cause inefficiencies in the system. In the example implementation, discussed in chapter 7, the provision for multi-cycle addition and multiplication has been removed, because this feature greatly increases the complexity of the controlling program, as status lines must be monitored after each instruction to determine when the functional unit has completed its computation. The size and performance of the single cycle implementations were found to be acceptable, removing the need for the multi-cycle implementations.

There is a tradeoff of simplicity versus performance. As was the case with the multi-cycle provision, by limiting some of the functionality of the individual units, the levels of abstraction start to disappear; the designer is no longer insulated from implementation specific details. However, with small modifications at the highest levels of the system, large performance gains can be achieved. As the modifications affect the inner workings of elements in the design, the performance benefits of optimizing the design for a specific algorithm decrease, to the point where the benefits that the abstraction provide to the ease of design outweigh the potential improvements. Exactly where this threshold lies depends on the performance requirements of the processor.

Chapter 5

Fast Interpolation Cell

There exist a number of classical methods for designing hardware for computation of only a few mathematical functions. Typically, any digital signal processing hardware will contain some combination of multipliers and adders. For algorithms that require more complicated functions, there are few options. For some cases, the function can be implemented in the form of a look-up table. However, for an input word size of more than about 16 bits, the memory required becomes prohibitively large. For some functions, iterative routines, in which an initial guess is successively refined until it is considered to be a match to the desired value, can be used. In order to be able to realize a function in an iterative algorithm, there must exist a way to evaluate the error in the estimate. An example of such a function would be the square root. While it is not possible to find the square root of a number x using only multiplications and additions, it is possible to evaluate how close an estimate y is to $\sqrt{x}$. This is accomplished by examining the error function, given by $e = x - y^2$. Based on this error, y is modified to minimize e.

Even though certain functions can be implemented using iterative routines, the execution time required depends on the goodness of the initial estimate, how quickly the chosen minimization routine converges, and on the rate of change of the function itself. To make the abstraction that the functional unit requires a fixed amount of time to complete, which makes instruction scheduling significantly less difficult, the amount of time allocated to the algorithm needs to be long enough that it will converge even in the worst case. This leads to inefficient implementations where the processor sits idle as the routine may complete early. This inefficiency can be reduced by either allowing for variable processing time, or by making use of an entirely different method that works in fixed time. The latter approach was chosen because it offers increased efficiency without additional complexity in the processor controller.

5.1 Interpolation

An alternative to iteration will be considered. For interpolation, the problem is to make an educated guess of the value of the function at any point, given only the values of the function at selected points. This is accomplished by making assumptions about the behavior of the function in between the given points. The simplest method assumes that the function is linear. For a slowly varying function, with the selected data points fairly closely spaced, this is an acceptable approximation. In this case, the value of the function $f(x)$ at a given point x is given by

$$f_i(x) = \frac{x - x_i}{x_{i+1} - x_i} f(x_i) + \frac{x_{i+1} - x}{x_{i+1} - x_i} f(x_{i+1}), \tag{5.1}$$

where x_i and x_{i+1} are the points at which the corresponding values of the function $f(x)$ are given, and $f_i(x)$ is the interpolated value of $f(x)$. This is a linear interpolation, where the desired function is modeled as a straight line, and the model is matched to the actual function at the two points $(x_i, f(x_i))$ and $(x_{i+1}, f(x_{i+1}))$.

The error in the interpolated value can be reduced by choosing a better approximation to the desired function $f(x)$. It only takes two points to completely determine a line, so using this as our model for $f(x)$ only allows an exact match of the desired function at two points. To match more points, a more complicated model must be choosen. One approach is to expand $f(x)$ into its Taylor series. If $x = x_i + a$ then

$$f(x_i + a) = f(x_i) + af'(x_i) + \frac{a^2}{2!} f''(x_i) + \cdots, \tag{5.2}$$

where $f(x_i)$, $f'(x_i)$, $f''(x_i)$, ...are known, allowing an accurate approximation for

$f(x)$. It is important that a is small, and $0 \leq a < 1$, so that a^{n+1} is less than than a^n. Choosing a according to these rules allows the subsequent terms in (5.2) to become less important. This important result allows an extra degree of freedom in choosing the desired accuracy for the approximation of $f(x)$. Using the linear model, the only way to increase accuracy is to choose more closely spaced values for $f(x_i)$. A design parameter h can be selected, with $h = x_{i+1} - x_i$ that specifies the spacing in the lookup table. In addition, the substitution $a = hp$, with a as in (5.1) can be made. p now represents the distance between points given in the look-up table and the desired point in terms of the interpolation point spacing h, and $0 \leq p < 1$. The accuracy can be improved by having multiple lookup tables supplying values for $f^{(n)}(x)$. Another design parameter, m, can be chosen to provide lookup tables for $f^{(n)}$ with $0 \leq n \leq m$.

For descrete computations, it is more appropriate to store the values in tables of finite differences. The finite difference of $f(x)$ evaluated at point x is given by

$$\Delta_m^n f(x) = f(x + m + nh) - f(x + m).$$

The derivative of $f(x)$ can be approximated by

$$\begin{aligned} f(x) &= \tfrac{f(x+h)-f(x)}{h} \\ &= \tfrac{\Delta f(x)}{h}. \end{aligned}$$

Defining

$$\delta_m^n f(x) = f(x + m + \frac{n}{2}h) - f(x + m - \frac{n}{2}h),$$

[19] allows (5.2) to be rewritten as

$$\begin{aligned} f_i(x_i + hp) &= \{(1-p)f(x_i) + pf(x_i + h)\} + \\ &\quad \{E_2\delta_0^2 f(x_i) + F_2\delta_1^2 f(x_i)\} + \\ &\quad \{E_4\delta_0^4 f(x_i) + F_4\delta_1^4 f(x_i)\} + \cdots \end{aligned} \tag{5.3}$$

where

$$\begin{aligned} E_2(p) &= F_2(1-p) &= -p(p-1)(p-2)/3!, \\ E_4(p) &= F_4(1-p) &= -(p+1)p(p-1)(p-2)(p-3)/5!, \ldots . \end{aligned}$$

If only the first two terms in (5.3) are used, a two point linear interpolation is performed. Use of the first four terms corresponds to four point cubic interpolation, and so on [19]. For the four and higher point interpolations, the goal is to match the interpolation function to the original function and the first and higher derivatives at the end points to provide a better approximation. This can be verified by differentiating (5.3) successively with respect to p, while observing

$$\begin{aligned} f^{(n)}(a+ph) &= f^{(n)} + pf^{(n+1)}(a) \\ &= f^{(n)} + p[f^{(n)}(a+h) - f^{(n)}(a)] \\ &= (1-p)f^{(n)}(a) + pf^{(n)}(a+h). \end{aligned}$$

Considering $0 \leq p < 1$, the ratio

$$\frac{E_4}{E_2} = \frac{p^2 - 2p - 3}{20}$$

does not vary greatly over the domain of p [19]. We can find a mean value for this ratio, k,

$$k = \int_0^1 \frac{p^2 - 2p - 3}{20} = -.18333.$$

However, it has been shown that a more effective value is obtained if the mean value is not weighted equally over the entire range. The accepted value is $k = -.18393$ [1]. By calculating this value for k,the substitution can be made

$$\delta_m^2 f(x) = \delta^2 f(x) + k\delta^4 f(x),$$

which realizes the benefits of a six point interpolation, at the expense of placing limitations on the choice of parameters in the look-up tables. For this substitution to be allowed, the look-up table spacing must be slightly reduced. This substitution is known as Comrie's Throwback. Specifically, these equations allow for the most accurate interpolation when h is several orders of magnitude smaller than x, and the differences, δ, are kept reasonably small.

With this modification,a complete six point interpolation can be written as function as

$$f_i(x_i + hp) = \{(1-p)f(x_i) + pf(x_i + h)\} + \{E_2\delta^2_{m_0}f(x_i) + F_2\delta^2_{m_1}f(x_i)\}, \qquad (5.4)$$

with E_2 and F_2 as previously defined. It has been shown that limiting the size of the values in the look-up table constrains the error in (5.4) to be proportional to h^6, which allows for very little memory consumption. Substituting $g(x_i) = f(x_i + h) - f(x_i)$, allows (5.4) to be rewritten in slightly simpler form,

$$f_i(x_i + hp) = \{f(x_i) - pg(x_i)\} + \{E_2\delta^2_{m_0}f(x_i) + F_2\delta^2_{m_0}f(x_i + h)\}, \qquad (5.5)$$

with $g = f(x_i+h)-f(x_i)$, and E_2 and F_2 as previously defined. This format suggests a possible implementation. Given the input, it is easy to calculate p, E_2, and F_2. This implies only $f(x_i)$, $g(x_i)$, $\delta^2_{m_0}f(x_i)$, and $\delta_{m_1} = \delta^2_{m_0}f(x_i + h)$ need to be looked up. $\delta^2_{m_0}f(x_i)$ and $\delta^2_{m_0}f(x_i + h)$ are just sequential values in the same look-up table. Thus only three look-up tables are required: $f(x_i)$, $g(x_i)$, and $\delta^2_{m_0}f(x_i)$.

An important attribute of this implementation is that the only items specific to any given function are these three lookup tables. The remainder of the algorithm makes few assumptions on the function to be implemented, other than the fact that the function and its first few derrivatives must be continuous. While these restrictions may limit this algorithm's approach for estimating arbitrary functions, it is an acceptable trade-off for the speed with which it can calculate these specific values. The size of the lookup tables varies with the function being implemented, but they are usually quite small.

To keep the lookup tables as compact as possible, the input to the interpolator must be normalized. For the square root and inverse square root, the input is normalized to be $.25 < x \leq 1$, making the output for the square root function $.5 < f(x) \leq 1$, and for the inverse square root $1 \leq f(x) < 2$. One advantage to the square root and inverse square root functions is that one half the number of shifts applied to the input value should be applied to the output in the appropriate direction to undo the effects of the normalization stage. Functions that do not share this property, such as trigonometric functions where simple binary normalization is not possible, require larger lookup tables. The normalization, in addition to the lookup tables, must be modified for each function implemented.

This approach provides a simple, fast way to compute a large class of functions in fixed time that otherwise require iterative functions or large lookup tables. These difficult functions are now implemented using only a few table look-ups, multiplications, and additions. In addition to being able to implement functions that previously were too difficult to compute in reasonable time and space, the algorithm separates out the computations that are specific to a given function, that is the structure of any implementation independent of the required function. To implement multiple functions, the only additional hardware needed are look-up tables, making this cell flexible and useful for a wide variety of applications.

5.2 Interpolation Cell

The functional element that performs the interpolation can be implemented in a variety of ways. If layout area were of no concern, equation (5.5) could be implemented directly, with 3 real multipliers and three real adders. However, the layout would be large. Instead, the compromise was made that only one multiplier and one adder were needed. In addition, there were 3 lookup tables, 3 registers to hold interim results, and a barrel shifter used to normalize the input. There was a small finite state machine that controlled the process. The complete computation required six clock cycles to complete. The activities performed in each cycle can be seen in table 5.1. A block diagram of the cell data path can be seen in figure 5-1.

To keep all subcells as small as possible, extensive simulation of the algorithm was performed to determine how many bits were needed for internal representations to

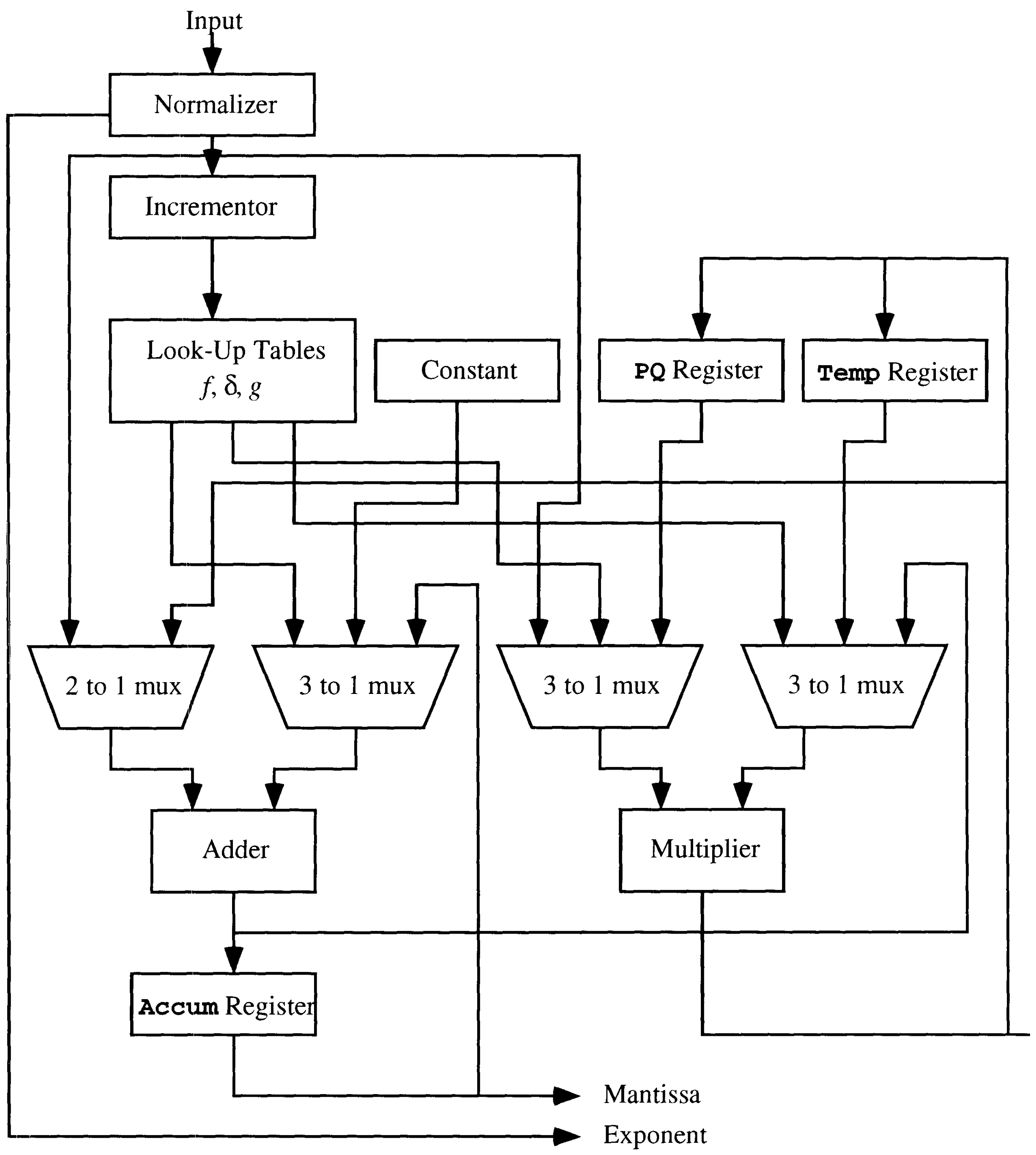

Figure 5-1: Block Diagram of Fast Interpolation Cell Data Paths

Cycle	Calculation
1	Compute $p(1-p)$ and save in `PQ` register
2	Calculate $(2-p)$`PQ` and save in `Temp` register.
3	Calculate $\delta_m f(0)$`Temp` and save in `Accum` register.
4	Calculate $(p+1)$`PQ` and save in `Temp` register.
5	Calculate $\delta_m f(1)$`Temp` and add to and save in `Accum` register.
6	Calculate pg, add to and save final value in `Accum` register.

Table 5.1: Interpolator Cycle Computations

insure that the output was accurate to within one least significant bit. The calculation of (5.5) was constrained to allow at most one bit of error in each of the three terms, so that the output could have at most three bits of error. The input word was chosen to be 24 bits wide, which, for the square root and inverse square root, necessitated the $f(x)$, $g(x)$, and $\delta_m(x)$ to be 25 bits, 18 bits, and 18 bits wide respectively. This allowed an output word width of 27 bits, of which 24 were always valid. After any necessary truncation, the output would be 24 bits wide, with an average of 1/2 bit of error. The values in the look-up tables were adjusted in an attempt to minimize this error.

The input word width was also 24 bits. To reduce the latency required for computation, the input value was normalized, in a single cycle, to within $.25 \leq x < 1$, with the most significant bit having the value of .5. The input was assumed to be unsigned; otherwise, the output would be complex. However, the algorithm does not require this. This normalization is necessary to keep the error to within the desired bounds. After computations on the output of this cell are complete, the final value is shifted by the inverse of the amount that the original input required for normalization to reduce errors because even simple computations between two numbers could always introduce at least one bit of error. This approach attempted to insure that any error that was introduced later would be in unused bits. To delay the inverse shift until later, the cell output was kept in a mantissa-exponent form; the cell output consisted of the 24 bits of mantissa, and 5 bits of exponent, making the output resemble a block-floating point value rather than a fixed-point value.

To implement two functions using a single interpolation cell, a single bit was brought into the lookup tables that allowed for the selection of one of two output values for any input. In addition, this function select line was extended to the normalizer, as this subcell must be altered for each function implemented. Fortunately, for the square root versus inverse square root, the only change required was to invert the sign of the exponent when the inverse square root is computed. To calculate the exponent, the number of shifts applied to the input was divided by 2 using a right shift by one bit. This operation requires almost no hardware to translate the input shifts to the output exponent.

It was found that the algorithm itself, making use of the modifications suggested by Comrie, was capable of providing computations to over 30 bits of accuracy with less than one bit of error. The only modification that is required is tpo scale up the subcells to the new word size, until values much larger than 30 bits need to be computed. In addition, care was taken to design using in a high-level language, so that the word width was parametric and cell generation was automated. Separate programs were written to generate the appropriately sized lookup tables given the word width. This cell is uniquely versatile because additional functions may be added with little effort, and the accuracy of the design can be quickly and easily altered to the requirements of the overall processor algorithm.

Chapter 6

LAGER Silicon Assembly System

Until recently, digital integrated circuit design has been dominated by few individuals with the necessary training to learn all of the subtle issues involved with VLSI layouts. Recently, however, this has started to change. One example of a system that attempts to make VLSI more usable is the LAGER silicon assembly system. It consists of a suite of tools that allow for the specification of a system design in a high-level language. It also allows for hierarchical designs that minimize the required design time, while achieving the best possible performance. In addition, it attempts to shield the designer from the subtle details of integrated circuit design. While this ease of use comes at the price of less than optimal layouts, the results are very usable, and are available in a fraction of the time.

6.1 The LAGER Tools

The LAGER system is a complete set of silicon design tools created by many institutions [17]. Each tool performs a small part of the overall design process. Because the system is constructed as a framework of tool sets, new capabilities can easily be added to the system. For example, design data can be input in the form of a structure description language (SDL) file or a schematic representation from Viewlogic System's Viewlogic or LAGER's VEM. All of these supported input formats translate the design information into a standard intermediate format that can be passed as input to the next level of tools.

The blocks that comprise every design come from a variety of libraries. The libraries are specialized for several different types of designs: one is specialized for fixed width datapath driven designs, another for designs based on individual logic elements, and a variety of other design criteria. To use a library in a design, each library requires a specialized structure processor and layout generator, which are the tools that create layouts from the desing specification. The structure processor determines the overall structure of the design, performs any translations between logical and physical groupings, as in expanding the specification of a mulibit unit into multiple occurances of a single bit wide cell. The layout generator actually assembles the layout cells from the library into a complete design that meets preset input and output specification. This standard interface makes it possible to use a variety of libraries in a design. However, only one library may be used at any level of the hierarchy. This is because the inner workings of each library may be different, so the layout generators and structure processors only need to be capable of manipulating cells from the associated library.

While this limitation may seem restrictive, the hierarchy is the strength of the system. Certain libraries may be better for certain functions than others. Having multiple levels of hierarchy allows one level of a design to be modified, possibly by making use of different libraries, without affecting the overall desing. This permits changes at the low levels of the design, such as improved performance or reduced area, to not require changes throughout the design. Certain libraries may be able to generate more efficient versions of logically equivalent cells. This gives the designer added flexibility in creating the design. Certain parts may be constructed in a variety of ways to determine the best design solution. In the interest of performance, the system attempts to recreate only the parts of the design that have changed when compiling from design specification to silicon layout.

Although each library can have its own layout generator and structure processor, the final output from these routines must be a standard file format, so that a single layout generator can assemble all of the blocks of the system. This program, Flint, performs all of the routing between the subcells in any design. It attempts to gather all of the blocks in a design and arrange them so that the routing between them is feasible, and as efficient as possible. This is an enormous task, which would take excessive computation time and complexity to automatically compile large designs. For this reason, the placement and routing operations can be performed interactively. Often, simply by rearranging the locations of the subcells, the efficiency of the routing can be significantly improved.

While one of the overall design goals of the LAGER system was to allow for system specification through behavioral descriptions, this is still not possible. There are, however, utilities that allow for the creation of complicated finite state machines and combinatorial logic using a simple behavioral descriptions. The desired logic is described in the behavioral description system language (BDS), which is processed by standard logic optimization routines, and can be used to generate a design based on standard logic cell parts or the AND-OR planes for programmable logic arrays or finite state machines. This allows for FSM-based controllers to be generated using a behavioral description language that closely resembles assembly code.

One of the greatest benefits of the system is that, given the input in the form of a structural description language, there are facilities provided for parameterization of the design. It is possible to create designs where important parameters, like the datapath width, are required only at compilation time as the layout generation is occurring. Because SDL is implemented using LightLisp, most of the standard lisp operators are available. Using these operators, it is possible to specify, for example, control line widths in terms of the logarithm of the datapath width. This high-level processing allows very complicated designs to be constructed, with many different parameters being internally calculated from a few inputs. The input values can be passed down through the hierarchy, allowing the same subcell to be used in different parts of the design using different parameter values in each instance.

The entire system provides an unprecedented level of flexibility in design and implementation. The layout generators and structure processors are sophisticated programs, capable of laying out complex, non-uniform cells. For example, the most significant and least significant bits of a subcell often need different connections than the other bits. These routines can handle this type of variation. This allows for very complex library cells to be constructed. Multi-bit cells, like adders, registers, and multipliers, only need to have bit slice representations laid out by hand in the library. The processors can assemble them, using an instruction file, into a larger, full-width representation. This arrangement helps to keep the libraries small, while still maintaining design flexibility.

6.2 Design Hierarchy

The design of the diagonal and off-diagonal processor elements made extensive use of the hierarchy available through this system. The subcells were designed to be as general as possible, while requiring only the minimum number of control connections. The only design parameters required for the overall system were the width of the internal data bus, and the number of registers in each processor. All other design parameters were derived from these. Some assumptions were made at certain levels of the design about the width of certain busses; internal requirements necessitated that they have minimum width, but these limitations were easily adhered to.

A block-level diagram of the diagonal processor can be seen in figure 4-2 and for the off-diagonal processor in figure 4-1. These block diagrams show the top-level of the design. The file dependencies for the top level of this design can be seen in figure 6-1. Only the file breakdown for the diagonal processor will be shown, because the off-diagonal processor contains a subset of the function units in the diagonal processor. The files used for this design are listed in appendix A. Each of the top-level cells is a functional unit with logic around it to provide the bus input/output functions. The tristate buffers and control signal generation logic is included at this level. The names of files taken from the supplied cell libraries are underlined.

The individual functional units, the multiplier, the adder, and the top level of the interpolator, can be further broken down as shown in figure 6-2. The design structure of these functional units is predominantly vertical, as seen in this figure. The tree

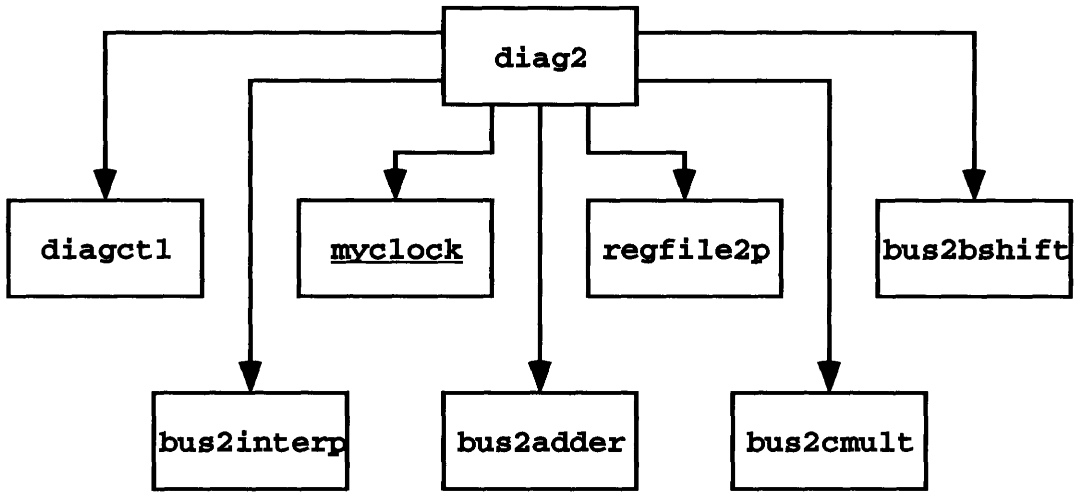

Figure 6-1: Design Hierarchy for the Top Level of the Diagonal Processor Element

does not branch frequently because an extra level of hierarchy is often needed to use a single part from a cell library. One level is needed to translate the part from the library, with library specific interfaces, into a part with standard interfaces. This extra level of hierarchy does not have significant overhead, but its presence is one of the drawbacks of the system.

The system controller can be seen in figure 6-3. This cell is very simple because the functionality of the controller is realized using the behavioral description file, `diagctl.bds`. For the off-diagonal controller, this file is replaced by `offdiagctl.bds`.

The hierarchy of for the barrel shifter and the register file are shown in figures 6-4 and 6-5, respectively. One of the basic cells of the register file, `scanreg2t`, is a modified version of a library cell. This specialized cell was constructed because the layout efficiency of the register file functional unit increased by an order of magnitude when the tristate buffer outputs were incorporated into each of the registers as opposed to being added as an additional level of hierarchy. This illustrates the tradeoff of design time verses layout efficiency. The easiest solution, from a design standpoint, to just tack the buffers onto the registers in the form of additional parts from the library. However, by creating hand optimized parts, thereby expanding the library, dramatic reductions in size are achieved.

The most complicated functional unit in the entrie design was the interpolator. The hierarchy for this unit is shown in figures 6-6, 6-7, 6-8, and 6-9. The look-up table ROM's were automatically generated by translateing a file of data values into a complete SDL file to be used as a PLA lookup table. The numerous subcells in the top level of the interpolator presented a problem for the placement and routing routines. The number of wires that needed to be routed for this design surpassed the limitations of the Flint automatic router. To complete the design, it was necessary to arrange the subcells by hand to minimize the routing distance.

One of the shortcomings of the LAGER system is the compatibility between the different tools. Because the system has been constructed using programs from a wide variety of institutions, subtle incompatibilities are evident. For instance, different libraries may assign different names to power supply connections, which prevent them from being connected to cells from other libraries. The different origins of the system are most evident in the cell libraries. The level of duplication between the libraries, and the incompatibility between them, can lead to layout inefficiencies, such as the extra levels of hierarchy that may be required. The restrictions on library compatibility often require extra, unwanted levels of hierarchy. It may be preferable to have a single library, with compatible layout generators and structure processors that allow parts from different libraries to be combined.

One of the most severe limitations for placement and routing is the fact that each subcell was represented by the smallest rectangle that encloses the subcell. This means that a significant area was left unused if the subcell had a non-rectangular shape. In addition, when the subcells were laid out using Flint, very little care was taken make the layouts rectangular. To achieve any reasonable level of efficiency in the layout required that the automation that is available in Flint could not be used. The majority of the layout had to be done interactively.

Despite these problems, the amount of time required to lay out a design using this system is significantly less than traditional methods. If this tool set is viewed as a starting point, then the next generation of tools should produce much more flexible and efficient designs. The system as a whole is extremely functional; the next step is to optimize it. With the increased processing power of modern workstations,improved routing methods can be employed for better designs. The time required to lay out a design by hand will continue to increase as circuits become more complicated. Soon, the benefits obtained from hand designs will be outweighed by the extra time required for layout. This system allows for fast prototyping of very complicated parts. When combined with the increasing availability of fabrication facilities, this will help to make integrated circuit system prototyping the most cost and time efficient was to design, prototype, and implement custom hardware.

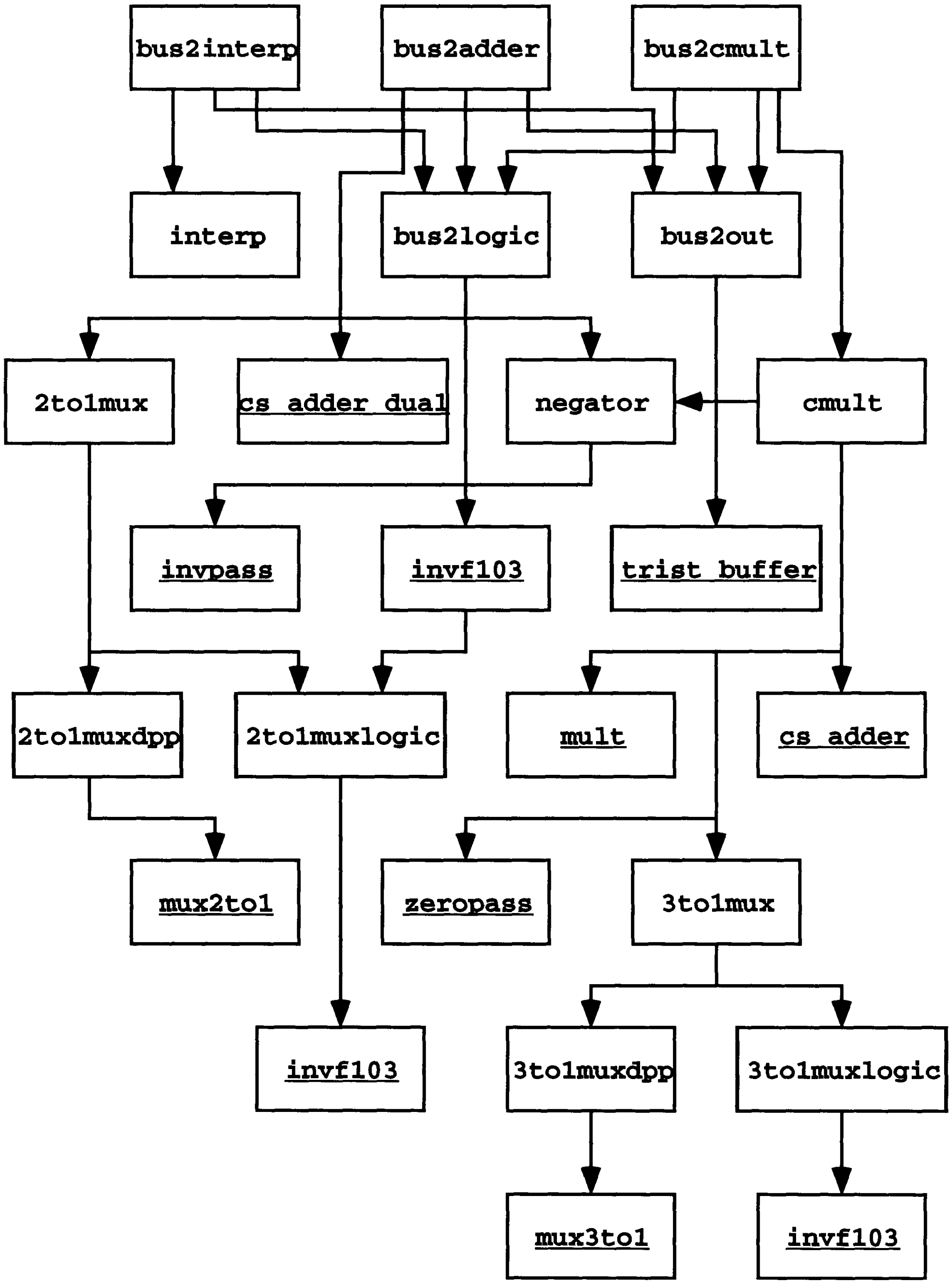

Figure 6-2: Design Hierarchy for Adder, Multiplier, and Top Level of Interpolator

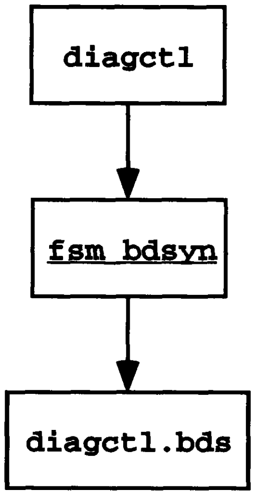

Figure 6-3: Design Hierarchy for Processor Controller

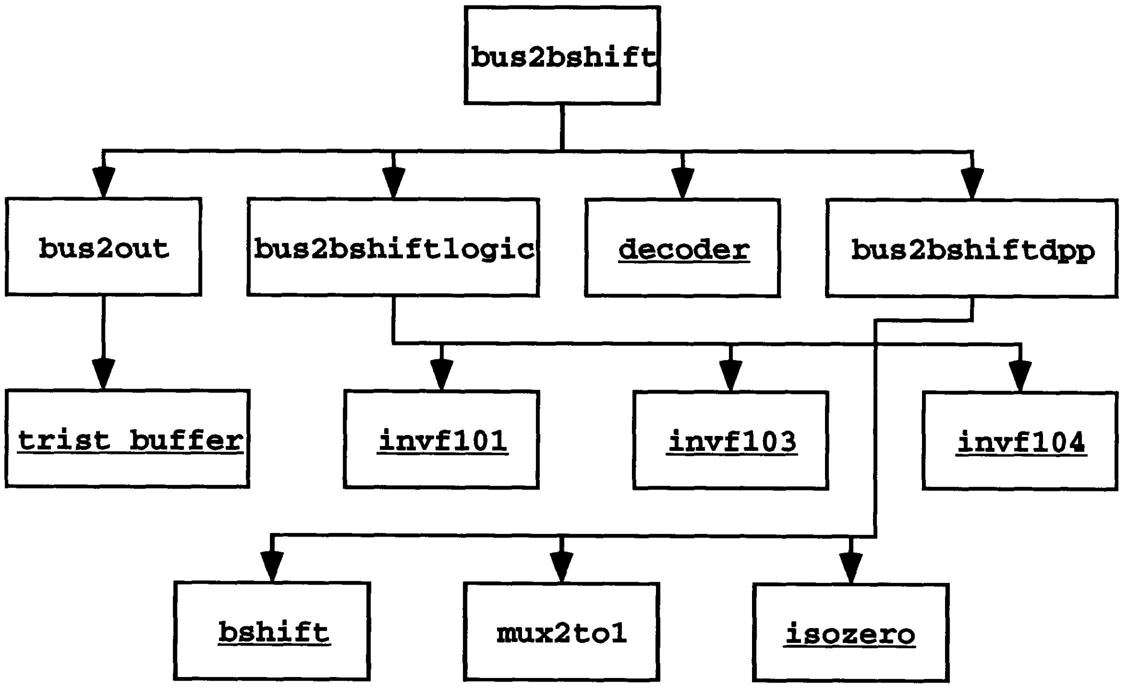

Figure 6-4: Design Hierarchy for Barrel Shifter

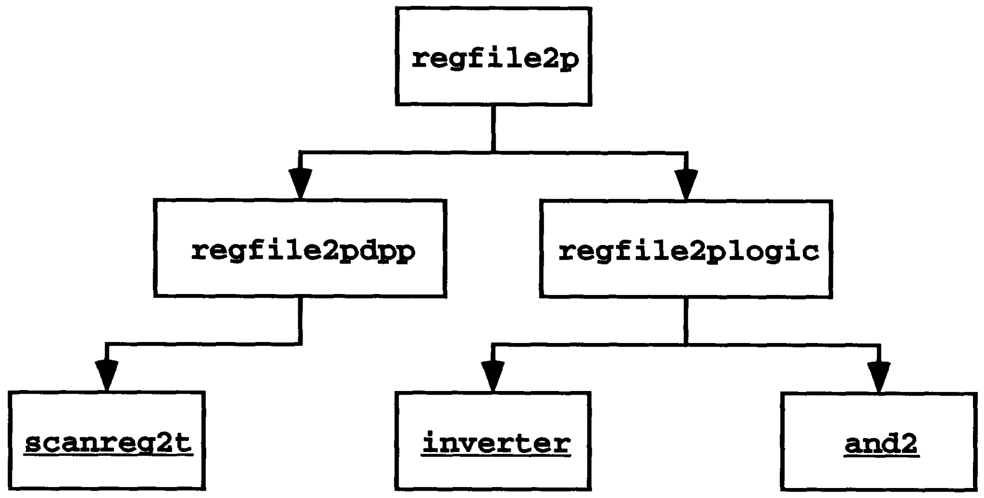

Figure 6-5: Design Hierarchy for Register File

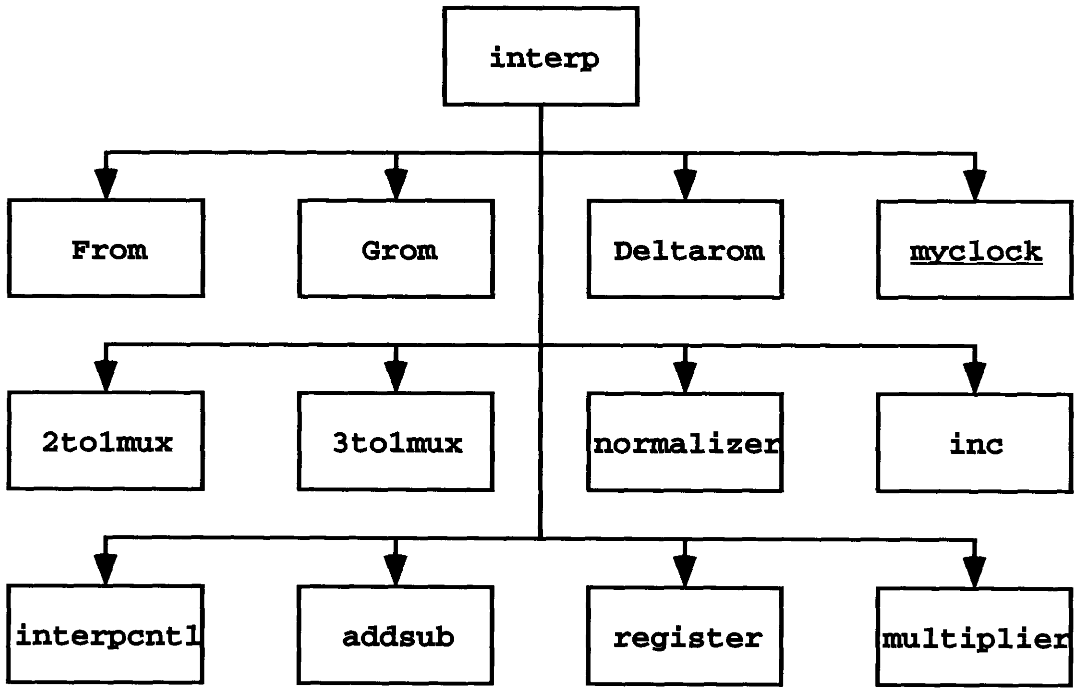

Figure 6-6: Top Level Design Hierarchy for Fast Interpolation Cell

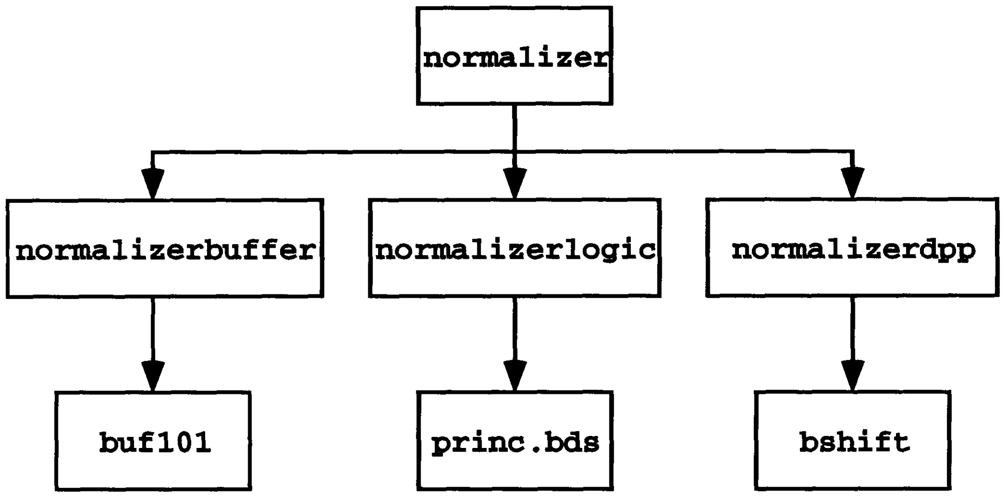

Figure 6-7: Design Hierarchy for Fast Interpolation Normalization Subcell

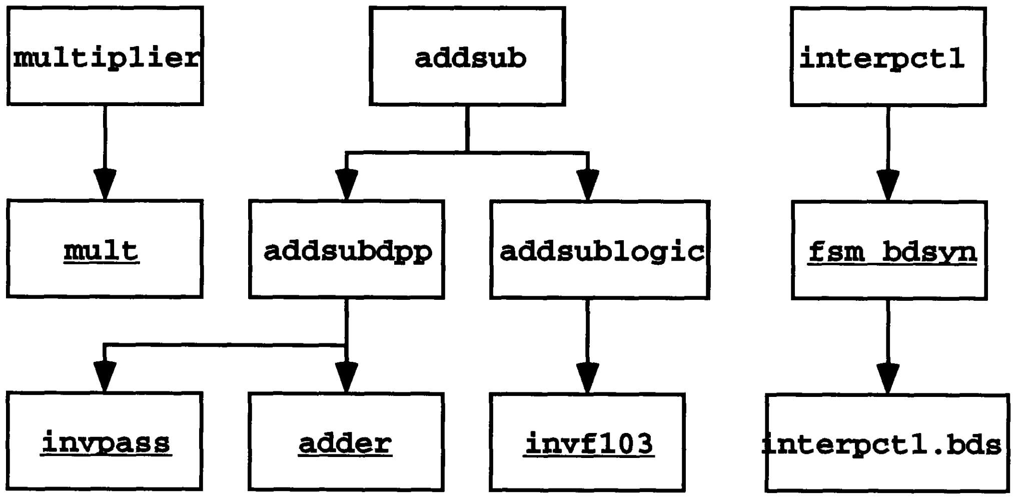

Figure 6-8: Design Hierarchy for Fast Interpolation Cell Multiplier, Adder, and Controller

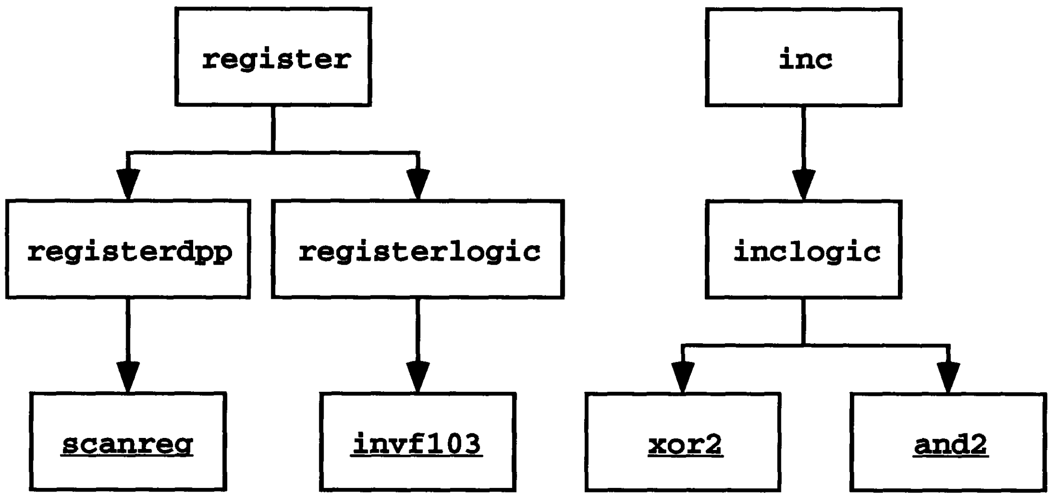

Figure 6-9: Hierarchy for Fast Interpolation Cell Incrementor and Register

Chapter 7

Realization

The approach presented here has been tailored to allow for flexibility in the specific realization. However, this design was created with a specific purpose in mind. It was designed to be used for direction of arrival calculations for an adaptive array system. Because it is to become part of a larger system, the specifications for input and output word size are determined by the other components. With this information in mind, the architecture was tailored to these values when compromises on the generality of the system had to be made.

7.1 Specifications

The data format for used in the realization of this system is in the form of 32-bit complex numbers, with 16 bits of real data and 16 bits of imaginary. Data words of this size are adequate to provide an at least 90 dB signal-to-noise ratio for the entire system. Internally, there were two supported data formats; 32-bit complex and 32-bit block floating point real. Because large portion of the computations for complex SVD are performed on real values, it was considered advantageous to support an extended precision real data type.

The 32-bit real representation was driven by the input and output of the interpolator. The interpolator inputs a 24-bit real number with no exponent, and outputs a 24-bit number, with an 8-bit exponent. Thus, the output of the interpolator cannot be fed directly back into the interpolator, since the exponent would be discarded. This would be useful for computation of the fourth root of a number, but as that is not needed for this algorithm, it was considered to be an acceptable tradeoff. The reason that this data type was supplied was that in the diagonal processor, all the complex data becomes real early on. In an attempt to limit the error in every computation, 24 bits was allowed upon translation to being fully real. Most of the time, a value became real through multiplication by its conjugate, as in magnitude calculations. Because of this, the multiplier was modified to allow a mode where the input was a complex number and its conjugate, and the output was a 24-bit real value. To deal with these values, the complex adder had to be modified to allow for a mode that could add two 24-bit real numbers.

For the most part, the exponent was just carried through the system, without being affected by many of the functional units. In fact, no provision was made to handle operations on two 24-bit numbers with different exponents. If the need arose, the a barrel shifter is available in the system to apply the exponent to the mantissa. This was used when two real numbers had to be manipulated, and their exponents were not identical, and for conversion from real data to the 32-bit complex data used by the rest of the system.

The addition of this data type did not require excessive additional space. Due to the fact that this data type was not fully supported for all operations, most units could be modified by adding only a small amount of control logic. Specifically, the exponent was seldom used given that this would have significantly increased the size and complexity of the functional units. In order to limit the additional logic required, the real data type used the same data paths as the complex data type. Instead of providing connections for 24 bits of real data, 16 bits of complex data, and 8 bits of exponent, the added 8 bits of real data and 8 bits of exponent were sent along the data paths normally used for the 16 bit imaginary part of a complex value. Thus, the destinction between a real and complex number is entirely in software. There is no physical way to tell the difference.

As the additional data type was added to the existing system, care was taken to insure that the functional units would be backward compatible with simpler designs. The units that were modified to allow the real data computations were altered so that if this ability is not needed in other designs, the additional control inputs can be a fixed value, and the parts will behave exactly as they had before the modification. Designs that will not use these modifications will not require any increase in complexity.

The real data type will certainly help to reduce the error through the system, thereby justifying the extra design effort required. The cycle time for the completed parts is expected to be between 5 and 10 MHz, possibly faster. This should allow for for use in most real-time systems.

7.2 Implementation Difficulties

The largest problem with the complete design was design efficiency. Due to the operation of Flint, the floorplanner, there was unused space between the cells in the final layout. Flint requires all subcells to be rectangular, which can lead to inefficiencies for irregularly shaped subcells. The impact of this layout problem can be reduced through careful interactive operation of Flint. By recognizing where the blank spots are, and re-arranging the cells by hand into the most rectangular a region as possible can generate the best results. However, even the best interactive layout generated by Flint is still inefficient. For a 100,000 transistor implementation, supporting 32-bit data type, the overall area for the diagonal processor was found to be approximately $11mm$ per side, for a total area of $121mm^2$, when implemented in a process with $\lambda = 0.6\mu m$. In the same process, the off-diagonal processor was found to be approximately $9.5mm$ on a side, for a total area of $90.25mm^2$. The actual layouts can be seen in figure 7-1 for the diagonal processor, and in figure 7-2 for the off-diagonal processor.

Additional performance can be achieved as fabrication technologies improve. However, this design can only be implemented in a technology supported by the LAGER system, and at the current release, the most advanced process supported is $\lambda = 0.6\mu m$.

One way to decrease layout area and increase overall performance is by hand

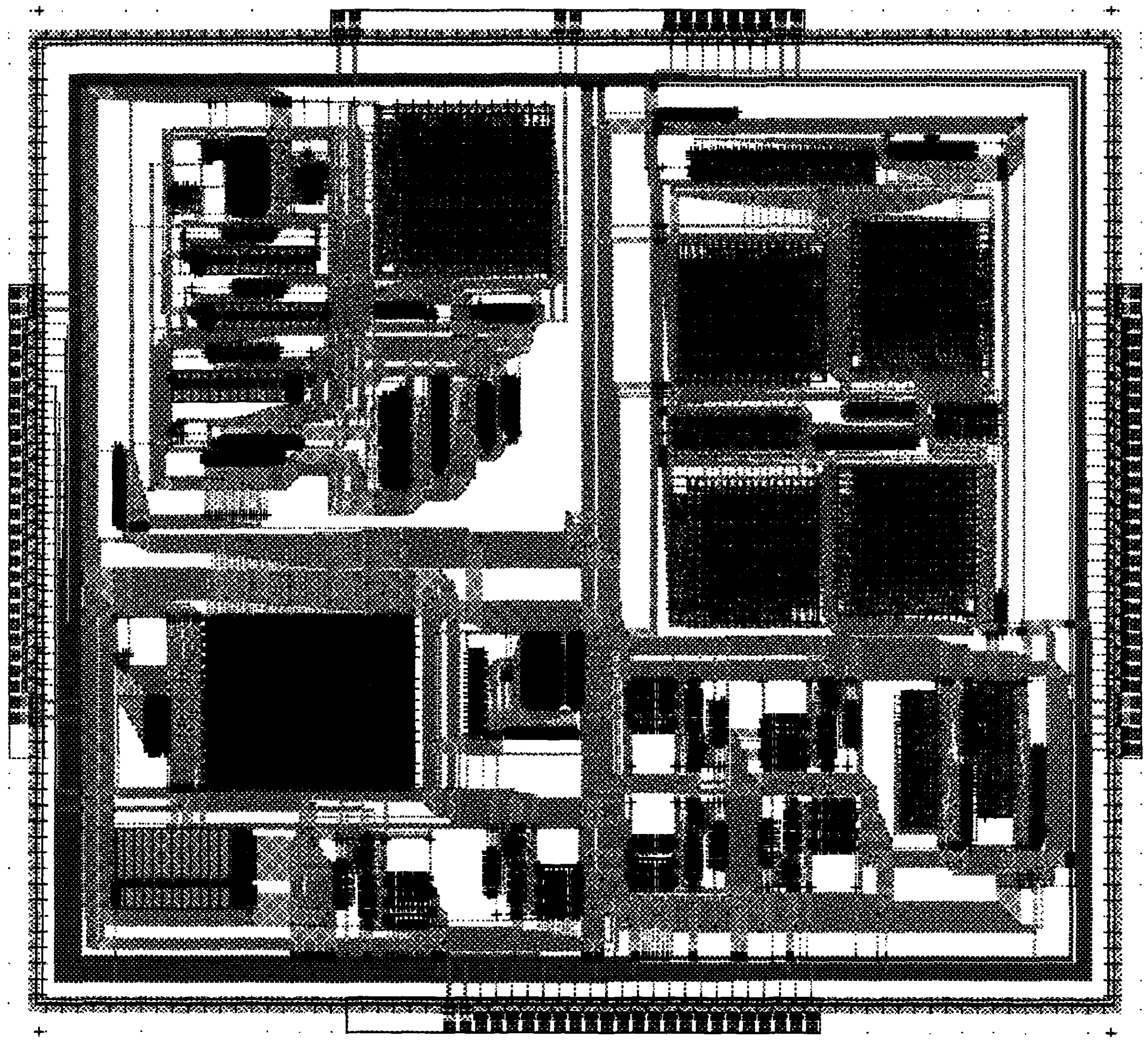

Figure 7-1: Layout of Diagonal Processor

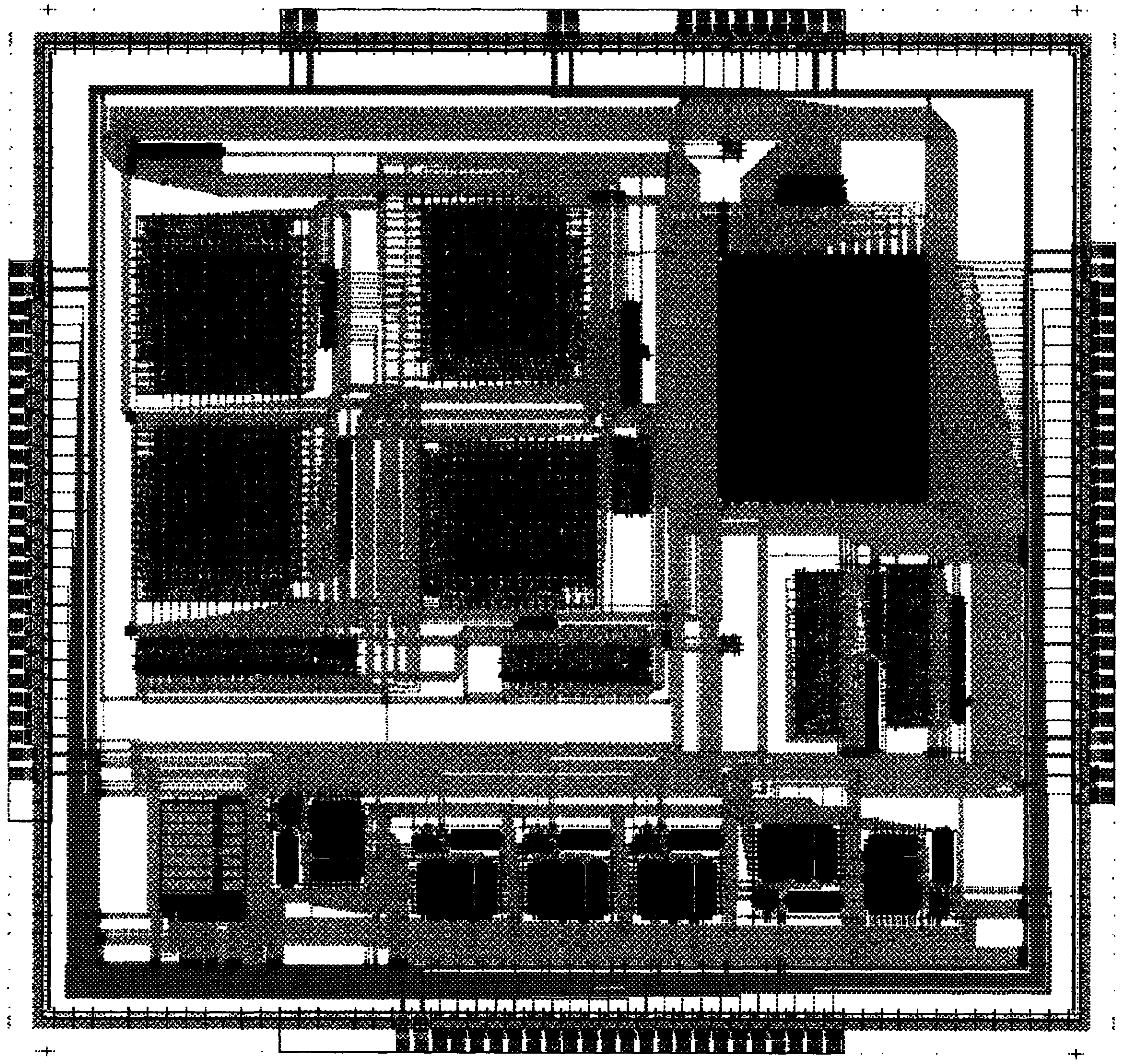

Figure 7-2: Layout of Off-Diagonal Processor

optimization of the library parts. By examining the design and determining what groupings of parts seem to occur frequently, new library parts that contain the functionality of two or more parts can be constructed. As mentioned in the previous example, the combination of a register and a tristate buffer allowed for an order of magnitude decrease in size of the register file. The same customizations could be performed on a number of other subcells in the interpolator.

To maximize performance, the original design used two separate pairs of operand busses and two result busses, so that computations could go on in parallel. The algorithm can be re-ordered in such a way as to exploit this parallelism if it is available. However, due to the feature size limitations imposed by existing fabrication facilities, the two operand bus design is all that can be supported using the current technology.

As it stands, the design, offers reasonable performance without requiring additional parallelism. The effort required to make changes to the overall design now that the framework exists is minimal. Through hand optimizations, the size, and therefore cost of fabrication can be decreased, while at the same time increasing the overall performance. Despite the implementation difficulties presented, the benefits from having a short design time, high performance VLSI solution available makes this approach a viable solution to a difficult problem.

Chapter 8

Conclusions

8.1 Future Work

To maximize the performance of the LAGER system, the efficiency of the routing tool needs to be improved. This can be accomplished by either creating a new tool to replace or augment the performance of Flint, or using a more customized library allowing for design realization in fewer subcells. By having a more complete set of cells to work from, the number of cells needed would decrease. This would decrease the load on Flint, making the overall routing simpler.

Additionally, support for the new sub-micron processes needs to be included. These smaller, faster processes will allow for more functionality on a chip. By decreasing the minimum feature size, for example, a processor could have two multipliers to increase the performance on algorithms that support parallelism. The power of a given processor could be increased even further.

As the power of these designs increases, the need for specialization increases even more. This means that the optimal solution to a problem can no longer be an implementation in off-the-shelf components. As the power increases, the cost will inevitably decrease, making this type of solution even more attractive. With the inclusion of more powerful, specialized units, custom processors become a more attractive solution.

Specifically, for this processor, the interpolation cell should be optimized, as it takes up approximately one third of the silicon area. If the number of subcells could be reduced, the overhead associated with subcell interconnections could be reduced. With 32-bit wide data paths, each extra data connection requires a large amount of space.

With an optimized interpolation cell, and careful hand placement of the subcells, this design could become much more compact, allowing for better overall performance. The design itself should be extensively simulated to determine where the performance bottlenecks are and to gauge the overall performance. Once the design has been thoroughly tested and refined, it will be ready to be fabricated for prototyping purposes. To keep initial costs down, one chip that allows for the functionality of both processor types should be designed and fabricated.

Following verification of the prototypes, the design framework can be taken and used to build custom processors for a variety of uses. A new controller should be designed that allows for external program storage, or at least some means of downloading a set of instructions to be executed. This would allow individual processors to perform different tasks at different points in the system, while still allowing for the flexibility of having processors optimized to the particular specifications of that system.

8.2 Concluding Remarks

In this thesis, a framework for a pair of custom processors designed to implement the singular value decomposition of a complex matrix was developed. The systolic array suggested allows for a unique match between the computation and communication requirements of this algorithm. The algorithm was designed to be executed in a fixed time, so that the inclusion of this processor array in a real-time system is feasible.

Additionally, the processors were created using a new silicon compiler system, allowing for design specification in a high-level language, while maintaining the performance advantage of customized VLSI. The framework can be used as a starting point for a new class of signal processors whose operation can be optimized for a given algorithm. The decreased design time and increased performance will improve as tools become more capable, to the point where custom VLSI is considered to be the rule, rather than the exception.

Appendix A

Source Code

A.1 SDL Code

A.1.1 2to1mux.sdl

This file is a shell to combine `2to1muxdpp.sdl` and `2to1muxlogic.sdl`. It is required for combining parts from the standard cell library and the datapath library. It is intended to implement a 2 to 1 multiplexer. If `SEL` is low, then `A` is routed to `OUT`. If `SEL` is low, then `B` is routed to `OUT`.

```
; 2 to 1 mux
; sel = 0, out = A
; sel = 1, out = B
; 5 october 1993 ccn
(parent-cell 2to1mux)
;
(parameters N)
;
(structure-processor SIVcheck)
(layout-generator Flint b)
;
(subcells (2to1muxdpp MUX ((N N)))
        (2to1muxlogic LOGIC)
)
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE SUPPLY))
;
(instance MUX (
        (IN0 A (width N))
        (IN1 B (width N))
        (S0 SEL)
        (S0_ S0_)
        (OUT OUT (width N))
        (Vdd Vdd)
        (GND GND)
))
;
(instance LOGIC (
        (SEL SEL)
        (S0_ S0_)
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal A (DIRECTION INPUT)) A (width N))
        ((terminal B (DIRECTION INPUT)) B (width N))
        ((terminal OUT (DIRECTION OUTPUT)) OUT (width N))
        ((terminal SEL (DIRECTION INPUT)) SEL)
        ((terminal Vdd (TERMTYPE SUPPLY) (TERM_EDGE TOP)) Vdd)
        ((terminal GND (TERMTYPE GROUND) (TERM_EDGE BOTTOM)) GND)
))
;
(end-sdl)
```

A.1.2 2to1muxdpp.sdl

This file comprises the actual multiplexer used in the file `2to1mux.sdl`. However, it requires both true and complement of the selection lines, so those will be generated elsewhere. The library cell it uses, `mux2to1`, comes from the datapath library.

```
(parent-cell 2to1muxdpp)
;
(parameters N)
;
(structure-processor dpp)
(layout-generator Flint a)
;
(subcells (mux2to1 MUX ((N N))))
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
;
(instance MUX (
        (IN1 IN0)
        (IN2 IN1)
        (SEL1 S0_)
        (SEL2 S0)
        (OUT OUT)
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal IN0 (DIRECTION INPUT)) IN0)
        ((terminal IN1 (DIRECTION INPUT)) IN1)
        ((terminal OUT (DIRECTION OUTPUT)) OUT)
        ((terminal S0 (DIRECTION INPUT)) S0)
        ((terminal S0_ (DIRECTION INPUT)) S0_)
        ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
        ((terminal GND (TERMTYPE GROUND) ) GND)
))
;
(end-sdl)
```

A.1.3 2to1muxlogic.sdl

The logic used to generate the complement of the data select line SEL for the 2 to 1 multiplexer is implemented using an inverter from the standard cell library.

```
(parent-cell 2to1muxlogic)
;
(layout-generator Stdcell)
;
(subcells (invf103 INV))
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE SUPPLY))
;
(instance INV (
        (A1 SEL )
        (O S0_)
))
;
(instance parent (
        ((terminal SEL (DIRECTION INPUT)) SEL)
        ((terminal S0_ (DIRECTION OUTPUT)) S0_)
        ((terminal Vdd (TERMTYPE SUPPLY) (TERM_EDGE TOP)) Vdd)
        ((terminal GND (TERMTYPE GROUND) (TERM_EDGE BOTTOM)) GND)
))
;
(end-sdl)
```

A.1.4 3to1mux.sdl

This file is a shell to combine `3to1muxdpp.sdl` and `3to1muxlogic.sdl`. It is required for combining parts from the standard cell library and the datapath library. It is intended to implement a 3 to 1 multiplexer. If `SEL[1:0]=00`, then `A` is routed to `OUT`. If `SEL[1:0]=01`, then `B` is routed to `OUT`, and if `SEL[1:0]=1X`, then `C` is routed to `OUT`.

```
; 3 to 1 mux
; sel = 0, out = A
; sel = 1, out = B
; sel = 3, out = C
; (sel = 2, out = A)
; 5 october 1993 ccn
(parent-cell 3to1mux)
;
(parameters N)
;
(structure-processor SIVcheck)
(layout-generator Flint v b)
;
(subcells (3to1muxdpp MUX ((N N)))
          (3to1muxlogic LOGIC)
)
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE SUPPLY))
;
(instance MUX (
          (IN0 A (width N))
          (IN1 B (width N))
          (IN2 C (width N))
          (S0 SEL (net-base 0))
          (S1 SEL (net-base 1))
          (S0_ S0_)
          (S1_ S1_)
          (OUT OUT (width N))
          (Vdd Vdd)
          (GND GND)
))
;
(instance LOGIC (
          (SEL SEL (width 2))
          (S0_ S0_)
          (S1_ S1_)
          (Vdd Vdd)
          (GND GND)
))
;
(instance parent (
          ((terminal A (DIRECTION INPUT)) A (width N))
```

```
        ((terminal B (DIRECTION INPUT)) B (width N))
        ((terminal C (DIRECTION INPUT)) C (width N))
        ((terminal OUT (DIRECTION OUTPUT)) OUT (width N))
        ((terminal SEL (DIRECTION INPUT)) SEL (width 2))
        ((terminal Vdd (TERMTYPE SUPPLY) (TERM_EDGE TOP)) Vdd)
        ((terminal GND (TERMTYPE GROUND) (TERM_EDGE BOTTOM)) GND)
))
;
(end-sdl)
```

A.1.5 2to1muxdpp.sdl

This file comprises the actual multiplexer used in the file `3to1mux.sdl`. However, it requires both true and complement of the selection lines, so those will be generated elsewhere. The library cell it uses, `mux3to1`, comes from the datapath library.

```
(parent-cell 3to1muxdpp)
;
(parameters N)
;
(structure-processor dpp)
(layout-generator Flint a)
;
(subcells (mux3to1 MUX ((N N))))
;(subcells (mux2to1 (MUXA MUXB) ((N N))))
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
;
(instance MUX (
        (IN0 IN0)
        (IN1 IN1)
        (IN2 IN2)
        (S0 S0)
        (S0_ S0_)
        (S1 S1)
        (S1_ S1_)
        (OUT OUT)
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal IN0 (DIRECTION INPUT)) IN0)
        ((terminal IN1 (DIRECTION INPUT)) IN1)
        ((terminal IN2 (DIRECTION INPUT)) IN2)
        ((terminal OUT (DIRECTION OUTPUT)) OUT)
        ((terminal S0 (DIRECTION INPUT)) S0)
        ((terminal S0_ (DIRECTION INPUT)) S0_)
        ((terminal S1 (DIRECTION INPUT)) S1)
        ((terminal S1_ (DIRECTION INPUT)) S1_)
        ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
        ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.6 3to1muxlogic.sdl

The logic used to generate the complement of the data select lines `SEL[1:0]` for the 3 to 1 multiplexer is implemented using inverters from the standard cell library.

```
(parent-cell 3to1muxlogic)
;
(layout-generator Stdcell)
;
(subcells (invf103 (INV0 INV1))
)
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE SUPPLY))
;
(instance INV0 (
        (A1 SEL (net-base 0))
        (O S0_)
))
;
(instance INV1 (
        (A1 SEL (net-base 1))
        (O S1_)
))
;
(instance parent (
        ((terminal SEL (DIRECTION INPUT)) SEL (width 2))
        ((terminal S0_ (DIRECTION OUTPUT)) S0_)
        ((terminal S1_ (DIRECTION OUTPUT)) S1_)
        ((terminal Vdd (TERMTYPE SUPPLY) (TERM_EDGE TOP)) Vdd)
        ((terminal GND (TERMTYPE GROUND) (TERM_EDGE BOTTOM)) GND)
))
;
(end-sdl)
```

A.1.7 addsub.sdl

This file is a shell to combine `3to1muxdpp.sdl` and `3to1muxlogic.sdl`. It is required for combining parts from the standard cell library and the datapath library. It is intended to implement an adder/subtracter. If SUB is low, then the unit adds A and B, and presents the sum on SUM. If SUB is high, then B − A is presented at C. This cell is used in the fast interpolation cell.

```
; adder/subtractor
; if sub = 0, sum = a + b
; if sum = 1, sum = b − a
; ccn 5 october 1993
(parent−cell addsub)
;
(parameters width)
;
(structure−processor SIVcheck)
(layout−generator Flint v b)
;
(subcells (addsubdpp ADDER ((N width)))
        (addsublogic LOGIC)
)
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE SUPPLY))
;
(instance ADDER (
        (A A (width width))
        (B B (width width))
        (SUB SUB)
        (SUBINV SUBINV)
        (OUT SUM (width width))
        (Vdd Vdd)
        (GND GND)
))
;
(instance LOGIC (
        (SUB SUB)
        (SUBINV SUBINV)
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal A (DIRECTION INPUT)) A (width width))
        ((terminal B (DIRECTION INPUT)) B (width width))
        ((terminal SUM (DIRECTION OUTPUT)) SUM (width width))
        ((terminal SUB (DIRECTION INPUT)) SUB)
        ((terminal Vdd (TERMTYPE SUPPLY) (TERM_EDGE TOP)) Vdd)
        ((terminal GND (TERMTYPE GROUND) (TERM_EDGE BOTTOM)) GND)
))
;
(end−sdl)
```

A.1.8 addsubdpp.sdl

This is the actual adder used in the `addsub.sdl` cell. In addition, each bit of the `A` input is XOR'ed with the `SUB` signal, and the `SUB` signal is fed into the carry input of the adder. This takes the 2's complement inverse of `A` when `SUB` is high. Each bit of `A` is inverted, and 1 is added to this value, making it negative. These subcells come from the datapath library.

```
(parent-cell addsubdpp)
;
(parameters N)
;
(structure-processor dpp)
(layout-generator Flint a)
;
(subcells (invpass NEG ((N N)))
          (adder ADDER ((N N)))

)
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
;
(instance ADDER (
          (IN1 INVOUT)
          (IN2 B)
          (CIN SUB)
          (CININV SUBINV)
          (COUT COUT)
          (COUTINV COUTINV)
          (OUT OUT)
          (Vdd Vdd)
          (GND GND)
))
;
(instance NEG (
          (IN A)
          (OUT INVOUT)
          (CNTL SUB)
          (Vdd Vdd)
          (GND GND)
))
;
(instance parent (
          ((terminal A (DIRECTION INPUT)) A)
          ((terminal B (DIRECTION INPUT)) B)
          ((terminal OUT (DIRECTION OUTPUT)) OUT)
          ((terminal SUB (DIRECTION INPUT)) SUB)
          ((terminal SUBINV (DIRECTION INPUT)) SUBINV)
```

```
        ((terminal COUT (DIRECTION INPUT)) COUT)
        ((terminal COUTINV (DIRECTION INPUT)) COUTINV)
        ((terminal Vdd (TERMTYPE SUPPLY) ) Vdd)
        ((terminal GND (TERMTYPE GROUND) ) GND)
))
;
(end-sdl)
```

A.1.9 addsublogic.sdl

The inverse of the control signal `SUB` for the `addsubdpp.sdl` cell is generated using an inverter from the standard cell library. This cell is combined with `addsubdpp.sdl` in the `addsub.sdl` file.

```
(parent-cell addsublogic)
;
(layout-generator Stdcell)
;
(subcells (invf103 INV))
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE SUPPLY))
;
(instance INV (
        (A1 SUB)
        (O SUBINV)
))
;
(instance parent (
        ((terminal SUB (DIRECTION INPUT)) SUB)
        ((terminal SUBINV (DIRECTION OUTPUT)) SUBINV)
        ((terminal Vdd (TERMTYPE SUPPLY) (TERM_EDGE TOP)) Vdd)
        ((terminal GND (TERMTYPE GROUND) (TERM_EDGE BOTTOM)) GND)
))
;
(end-sdl)
```

A.1.10 bus2adder.sdl

This is a complex adder/subtracter that has been modified to be able to work on real block floating point values. The complex data type has an equal number of real and complex bits. The real data type is fixed to 24 bits in the interest of design simplicity. The exponents of the addends are not handled when performing real adds. The exponent of the `B` input is just carried through to the output. This is acceptable if the addends have the same exponent. If **REALADD** is high, an addition using operands of the extended precision real data type is performed. If it is low, the operands are taken to be complex. If **FUNC** is low, the operands are added. If it is high, then **B** is subtracted from **A**. This cell is designed to fit into an architecture with two operand busses and one result bus. This cell is used in the top level hierarchy of both the diagonal and off-diagonal processor elements.

```
; sdl file for width bits real, width bits complex adder that sits in a one bus arch.

; if REALADD, then out = IN + IN, where in is assumed to be 24 bits real, 8 bits

; exponent.  Exponent bits are assumed to be the same, so B's bits are used.
; NOTE: DO NOT USE subtract function with REAL INPUTS..
; 8 april 1994
(parent−cell bus2adder)
;
(parameters inwidth outwidth (realBits 24) (exponentBits (− (* 2 outwidth)
 realBits)))
; note: outwidth <= inwidth
;
(structure−processor SIVcheck)
(layout−generator Flint b)
;
;
(subcells
        (cs_adder_dual REALADDER ((n inwidth) (csindex
(make−carry−select inwidth)) (g (length (make−carry−select inwidth)))))
        (cs_adder_dual IMAGADDER ((n inwidth) (csindex
(make−carry−select inwidth)) (g (length (make−carry−select inwidth)))
(FEEDTHRUS 2) (BUFFER 2) (BITHEIGHT 150) (CELLHEIGHT 118)))
        (negator (REALINV IMAGINV) ((N inwidth)))
        (bus2out BUSOUT ((N (* 2 outwidth))))
        (bus2logic LOGIC )
        (2to1mux BMUX ((N (+ exponentBits 1))))
)
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE SUPPLY))
;
(instance REALINV (
                (IN BBUS (width inwidth) (net−base inwidth))
                (OUT BBUSP (width inwidth) (net−base inwidth))
                (CNTL FUNC)
                (Vdd Vdd)
                (GND GND)
))
;
(instance IMAGINV (
                (IN BBUS (width inwidth) (net−base 0))
```

```
                    (OUT BBUSP (width inwidth) (net-base 0))
                    (CNTL FUNC)
                    (Vdd Vdd)
                    (GND GND)
))
;
(instance REALADDER (
          (a ABUS (net-base inwidth) (width inwidth))
          (b BBUSP (net-base inwidth) (width inwidth))
          (s OUT (term-base (- inwidth outwidth)) (net-base outwidth)
(width outwidth))
          (cin REALCIN)
          (Vdd Vdd)
          (GND GND)
))
;
(instance BMUX (
          (A BBUSP (net-base 0) (width exponentBits))
          (B GND (merge 0 (- exponentBits 1)))
          (OUT BMUXOUT (width exponentBits))
          (A FUNC (term-base exponentBits))
          (B IMAGCOUT (term-base exponentBits))
          (OUT REALCIN (term-base exponentBits))
          (SEL REALADD)
          (Vdd Vdd)
          (GND GND)
))
(instance IMAGADDER (
          (a ABUS (net-base 0) (term-base 0) (width inwidth))
          (b BMUXOUT (net-base 0) (term-base 0) (width exponentBits))
          (b BBUSP (net-base (- inwidth exponentBits)) (term-base
(- inwidth exponentBits)) (width (- inwidth exponentBits)))
          (s OUT (term-base (- inwidth outwidth)) (net-base 0) (width outwidth))
          (cin FUNC)
          (cout IMAGCOUT)
          (Vdd Vdd)
          (GND GND)
))
;
(instance BUSOUT (
          (OUT OUT (width ( * 2 outwidth)))
          (EBUS EBUS (width (* 2 outwidth)))
          (OUTEN OUTEN)
          (OUTENINV OUTENINV)
          (Vdd Vdd)
          (GND GND)
))
;
(instance LOGIC (
          (OUTEN OUTEN)
          (OUTENINV OUTENINV)
          (Vdd Vdd)
          (GND GND)
))
```

```
;
(instance parent (
        ((terminal ABUS (DIRECTION INPUT)) ABUS (width (* 2 inwidth)))
        ((terminal BBUS (DIRECTION INPUT)) BBUS (width (* 2 inwidth)))
        ((terminal EBUS (DIRECTION OUTPUT)) EBUS (width (* 2 outwidth)))
        ((terminal OUTEN (DIRECTION INPUT)) OUTEN )
        ((terminal FUNC (DIRECTION INPUT)) FUNC)
        ((terminal REALADD (DIRECTION INPUT)) REALADD)
        ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
        ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.11 bus2bshift.sdl

This is a barrel shifter that sits in a two operand, one result bus architecture. This barrel shifter operates on the 24-bit extended precision real data type. It accepts the data to be shifted in the upper 24 bits of the A input, and the amount to be shifted is in the lower 8 bits. This cell is used in the top level of the diagonal processor element.

```
; sdl file for width bit barrel shifter, with exponent bits of exp. data
; 8 april 1994
(parent-cell bus2bshift)
;
(parameters busWidth (realBits 24) (exponentBits (- busWidth realBits))
(realExpBits (ceiling (log2 realBits))))
;
(structure-processor SIVcheck)
(layout-generator Flint b)
;
;
(subcells
        (bus2bshiftdpp BSHIFT ((N realBits)))
        (bus2out BUSOUT ((N busWidth)))
        (bus2bshiftlogic LOGIC )
        (decoder DECODER ((address realExpBits)))
)
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE SUPPLY))
;
(instance DECODER (
        (address ABUS (net-base 0) (term-base 0) (width realExpBits))
        (dec_out DECOUT (width realBits))
        (Vdd Vdd)
        (GND GND)
))
```

```
;
(instance BSHIFT (
        (IN ABUS (net-base exponentBits) (width realBits))
        (OUT OUT (net-base exponentBits) (width realBits))
        (CONTROL DECOUT (width realBits))
        (LEFTRIGHT ABUS (net-base realExpBits))
        (INVLEFTRIGHT INVLEFTRIGHT)
        (MSB MSB (merge 0 (- realBits 1)))
        (Vdd Vdd)
        (GND GND)
))
;
(instance BUSOUT (
        (OUT OUT (width realBits) (net-base exponentBits)
(term-base exponentBits))
        (OUT GND (merge 0 (- realExpBits 1)))
        (OUT ABUS (width (- exponentBits realExpBits)) (net-base realExpBits)
 (term-base realExpBits))
        (EBUS EBUS (width busWidth))
        (OUTEN OUTEN)
        (OUTENINV OUTENINV)
        (Vdd Vdd)
        (GND GND)
))
;
(instance LOGIC (
        (OUTEN OUTEN)
        (OUTENINV OUTENINV)
        (LEFTRIGHT ABUS (net-base realExpBits))
        (INVLEFTRIGHT INVLEFTRIGHT)
        (MSBIN ABUS (net-base (- busWidth 1)))
        (MSB MSB)
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal ABUS (DIRECTION INPUT)) ABUS (width busWidth))
        ((terminal EBUS (DIRECTION OUTPUT)) EBUS (width busWidth))
        ((terminal OUTEN (DIRECTION INPUT)) OUTEN )
        ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
        ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.12 bus2bshiftdpp.sdl

This is the actual barrel shifter from the `bus2bshift.sdl` cell. It is taken from the datapath library. The `CONTROL` input selects how many bits the input should be shifted by, with only one bit of CONTROL high at a time. if LEFTRIGHT is low, the data is shifted left, and if it is high, the data is shifted right.

```
(parent-cell bus2bshiftdpp)
;
(parameters N)
;
;
(structure-processor dpp)
(layout-generator Flint a)
;
(subcells
 (bshift BSHIFT ((N N)))
 (mux2to1 MUX ((N N)))
 (isozero ZERO ((N N))) ; A
)
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
;
(instance BSHIFT (
                    (A A)
                    (B B)
                    (S CONTROL)
                    (O OUT)
))
;
(instance ZERO (
                  (IN IN)
                  (OUT A)
                  (ZERO INVLEFTRIGHT)
                  (Vdd Vdd)
                  (GND GND)
))
;
(instance MUX (
              (IN1 IN)
              (IN2 MSB)
              (SEL1 INVLEFTRIGHT)
              (SEL2 LEFTRIGHT)
              (OUT B)
              (Vdd Vdd)
              (GND GND)
))
;
(instance parent (
                    ((terminal IN (DIRECTION INPUT)) IN)
                    ((terminal OUT (DIRECTION OUTPUT)) OUT)
                    ((terminal MSB (DIRECTION INPUT)) MSB)
                    ((terminal CONTROL (DIRECTION INPUT)) CONTROL)
                    ((terminal LEFTRIGHT (DIRECTION INPUT)) LEFTRIGHT)
                    ((terminal INVLEFTRIGHT (DIRECTION INPUT)) INVLEFTRIGHT)
```

```
                ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
                ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.13 bus2bshiflogic.sdl

This file implements inverters for the control signals for the `bus2bshiftdpp.sdl` subcell. The inverters used are from the standard cell library. The primary difference between the inverters used is the drive capability. The cells `invf10X` are inverters with high drive capability for higher values of `X`.

```
(parent-cell bus2bshiftlogic)
;
(layout-generator Stdcell)
;
(subcells
        (invf103 INVEOUTEN)
        (invf101 MSBINV)
        (invf104 MSBINVINV)
        (invf103 LEFTRIGHTINV)
)
;
(instance INVEOUTEN (
        (A1 OUTEN)
        (O OUTENINV)
))
;
(instance MSBINV (
        (A1 MSBIN)
        (O MSBINV)
))
(instance MSBINVINV (
        (A1 MSBINV)
        (O MSB)
))
;
(instance LEFTRIGHTINV (
        (A1 LEFTRIGHT)
        (O INVLEFTRIGHT)
))
;
(instance parent (
        ((terminal OUTEN (DIRECTION INPUT)) OUTEN)
        ((terminal OUTENINV (DIRECTION OUTPUT)) OUTENINV)
        ((terminal MSBIN (DIRECTION INPUT)) MSBIN)
        ((terminal MSB (DIRECTION OUTPUT)) MSB)
        ((terminal LEFTRIGHT (DIRECTION INPUT)) LEFTRIGHT)
        ((terminal INVLEFTRIGHT (DIRECTION OUTPUT)) INVLEFTRIGHT)
        ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
        ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.14 bus2cmult.sdl

This is a complex multiplier designed for use in a two operand, one result bus architecture. This file is just a shell to but the bus interface logic, `bus2logic.sdl` onto the complex multiplier cell `cmult.sdl`. This cell is used in the top level of both the diagonal and off-diagonal processor elements.

```
(parent-cell bus2bshiftlogic)
;
(layout-generator Stdcell)
;
(subcells
        (invf103 INVEOUTEN)
        (invf101 MSBINV)
        (invf104 MSBINVINV)
        (invf103 LEFTRIGHTINV)
)
;
(instance INVEOUTEN (
        (A1 OUTEN)
        (O OUTENINV)
))
;
(instance MSBINV (
        (A1 MSBIN)
        (O MSBINV)
))
(instance MSBINVINV (
        (A1 MSBINV)
        (O MSB)
))
;
(instance LEFTRIGHTINV (
        (A1 LEFTRIGHT)
        (O INVLEFTRIGHT)
))
;
(instance parent (
        ((terminal OUTEN (DIRECTION INPUT)) OUTEN)
        ((terminal OUTENINV (DIRECTION OUTPUT)) OUTENINV)
        ((terminal MSBIN (DIRECTION INPUT)) MSBIN)
        ((terminal MSB (DIRECTION OUTPUT)) MSB)
        ((terminal LEFTRIGHT (DIRECTION INPUT)) LEFTRIGHT)
        ((terminal INVLEFTRIGHT (DIRECTION OUTPUT)) INVLEFTRIGHT)
        ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
        ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.15 bus2interp.sdl

This is a wrapper file used to put bus interface logic, `bus2logic.sdl` onto the fast interpolation cell discussed in chapter 5. It is designed for use in a two operand, one result bus architecture.

```
; sdl file for width bit fast interpolator that sits in a one bus arch.
; 23 feb 1994
(parent-cell bus2interp)
;
(parameters (width 24) busWidth)
;
;(structure-processor SIVcheck)
(layout-generator Flint b)
;
;
(subcells
        (interp INTERP ((width width)))
        (bus2out BUSOUT ((N busWidth)))
        (bus2logic LOGIC )
)
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE SUPPLY))
(net PHI (NETTYPE CLOCK))
;
(instance INTERP (
        (IN ABUS (width width))
        (OUT OUT (net-base (- busWidth width)) (width width))
        (SHIFTS OUT (net-base 0) (width (+ (ceiling (log2 width)) 1)))
        (FUNC FUNC)
        (GO GO)
        (BUSY BUSY)
        (RESET RESET)
        (PHI PHI)
        (Vdd Vdd)
        (GND GND)
))
;
(instance BUSOUT (
        (OUT OUT (net-base (- busWidth width)) (width width) (term-base
(- busWidth width)))
        (OUT OUT (width (+ (ceiling (log2 width)) 1)))
        (OUT GND (merge (+ 1 (ceiling (log2 width))) (- (- busWidth width) 1)
))
        (EBUS EBUS (width busWidth))
        (OUTEN OUTEN)
        (OUTENINV OUTENINV)
        (Vdd Vdd)
        (GND GND)))
;
(instance LOGIC (
        (OUTEN OUTEN)
        (OUTENINV OUTENINV)
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal ABUS (DIRECTION INPUT)) ABUS (width width) (term-base
(- busWidth width)))
        ((terminal EBUS (DIRECTION OUTPUT)) EBUS (width busWidth))
        ((terminal GO (DIRECTION INPUT)) GO)
        ((terminal RESET (DIRECTION OUTPUT)) RESET)
        ((terminal BUSY (DIRECTION OUTPUT)) BUSY)
        ((terminal FUNC (DIRECTION INPUT)) FUNC)
        ((terminal RESET (DIRECTION INPUT)) RESET)
        ((terminal OUTEN (DIRECTION INPUT)) OUTEN )
        ((terminal PHI (TERMTYPE CLOKC)) PHI)
        ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
        ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.16 bus2logic.sdl

This is just an inverter used for generating the complement of the bus output enable signal used by the tri-state inverter in the `bus2out.sdl` file.

```
(parent-cell bus2logic)
;
(layout-generator Stdcell)
;
(subcells
        (invf103 INVEOUTEN)
)
```

```
;
(instance INVEOUTEN (
        (A1 OUTEN)
        (O OUTENINV)
))
;
(instance parent (
        ((terminal OUTEN (DIRECTION INPUT)) OUTEN)
        ((terminal OUTENINV (DIRECTION OUTPUT)) OUTENINV)
        ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
        ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.17 bus2out.sdl

A tristate buffer used to control whether or not a functional unit presents its value on the result bus at any given time. This cell is a standard tri-state buffer to be attached to each functional unit to implement the bus interface logic. The cell requires both true and complement versions of the output enable signal `OUTEN`. The complement is generated in the `bus2logic.sdl` cell.

```
(parent-cell bus2out)
;
(parameters N)
;
(structure-processor dpp)
(layout-generator Flint a)
;
(subcells
        (trist_buffer EBUF ((N N)))
)
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
;
(instance EBUF (
        (IN OUT)
        (OUT EBUS)
        (CNTL OUTEN)
        (CNTLINV OUTENINV)
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal EBUS (DIRECTION OUTPUT)) EBUS)
        ((terminal OUT (DIRECTION INPUT)) OUT)
        ((terminal OUTEN (DIRECTION INPUT)) OUTEN)
        ((terminal OUTENINV (DIRECTION INPUT)) OUTENINV)
        ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
        ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.18 busserio.sdl

Data communication between the processors is implemented using a reduced width semi-serial communication path. In this case, 4 bits are allowed per communication channel. So it takes 8 clock cycles to transmit a 32-bit value. This is done by have 4 scan registers, and the data is shifted in or out of the register. A fifth register that is used for controlling the shifters is added. The data can come from one of two places; a second set of data connections in created for the purpose of loading and unloading the array.

```
(parent-cell busserio)
;
(parameters ioLines busWidth (N (floor (/ busWidth ioLines))))
;
(structure-processor SIVcheck)
(layout-generator Flint b v)
;
(subcells (busseriologic LOGIC ((ioLines ioLines)))
          (busseriodpp TIMER ((N N)))
)
(dotimes (i ioLines)
        (subcells (busseriodpp REGISTER ((N N)))
))

(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
(net PHI1 (NETTYPE CLOCK))
(net PHI2 (NETTYPE CLOCK))
(net PHI1INV (NETTYPE CLOCK))
(net PHI2INV (NETTYPE CLOCK))
;
(instance REGISTER (
        (REGIN ABUS (width N) (net-base (* i N)))
        (REGOUT EBUS (width N) (net-base (* i N)))
        (SCANIN INPUT (net-base i))
        (SCANOUT OUTPUT (net-base i))
        (LOAD LOAD)
        (LOADINV LOADINV)
        (SCAN SCAN)
        (SCANINV SCANINV)
        (KEEP KEEP)
        (KEEPINV KEEPINV)
        (BUSOE BUSOE)
        (BUSOEINV BUSOEINV)
        (PHI1 PHI1)
        (PHI2 PHI2)
        (PHI1INV PHI1INV)
        (PHI2INV PHI2INV)
        (Vdd Vdd)
        (GND GND)
))
;
(instance TIMER (
        (REGIN RESETINV (merge 0 (- N 1)))
        (SCANIN GND)
        (SCANOUT TIMEROUT)
        (LOAD TLOAD)
        (LOADINV TLOADINV)
        (SCAN SCAN)
        (SCANINV SCANINV)
        (KEEP TKEEP)
        (KEEPINV TKEEPINV)
        (BUSOE GND)
        (BUSOEINV Vdd)
        (PHI1 PHI1)
        (PHI2 PHI2)
        (PHI1INV PHI1INV)
        (PHI2INV PHI2INV)
        (Vdd Vdd)
        (GND GND)
))

(instance LOGIC (
                (LOAD LOAD)
                (LOADINV LOADINV)
                (TLOAD TLOAD)
                (TLOADINV TLOADINV)
                (SCAN SCAN)
                (SCANINV SCANINV)
                (KEEP KEEP)
                (KEEPINV KEEPINV)
                (TKEEP TKEEP)
                (TKEEPINV TKEEPINV)
                (RESET RESET)
                (RESETINV RESETINV)
                (BUSLD BUSLD)
                (BUSOE BUSOE)
                (BUSOEINV BUSOEINV)
                (START START)
                (TIMEROUT TIMEROUT)
                (INPUTA INPUTA (width ioLines))
                (INPUTB INPUTB (width ioLines))
                (INPUT INPUT (width ioLines))
                (IOSEL IOSEL)
                (Vdd Vdd)
                (GND GND)
))
;
(instance parent (
        ((terminal ABUS (DIRECTION INPUT)) ABUS (width busWidth))
        ((terminal EBUS (DIRECTION OUTPUT)) EBUS (width busWidth))
        ((terminal INPUTA (DIRECTION INPUT)) INPUTA (width ioLines))
        ((terminal OUTPUT (DIRECTION OUTPUT)) OUTPUT (width ioLines))
        ((terminal INPUTB (DIRECTION INPUT)) INPUTB (width ioLines))
        ((terminal IOSEL (DIRECTION INPUT)) IOSEL)
        ((terminal BUSLD (DIRECTION INPUT)) BUSLD)
        ((terminal BUSOE (DIRECTION INPUT)) BUSOE)
        ((terminal START (DIRECTION INPUT)) START)
        ((terminal LOAD (DIRECTION OUTPUT)) LOAD)
        ((terminal LOADINV (DIRECTION OUTPUT)) LOADINV)
        ((terminal SCAN (DIRECTION OUTPUT)) SCAN)
        ((terminal KEEP (DIRECTION OUTPUT)) KEEP)
        ((terminal KEEPINV (DIRECTION OUTPUT)) KEEPINV)
        ((terminal READY (DIRECTION OUTPUT)) SCANINV)
        ((terminal RESET (DIRECTION INPUT)) RESET)
        ((terminal PHI1 (TERMTYPE CLOCK) (DIRECTION INPUT)) PHI1)
        ((terminal PHI2 (TERMTYPE CLOCK) (DIRECTION INPUT)) PHI2)
        ((terminal PHI1INV (TERMTYPE CLOCK)(DIRECTION INPUT)) PHI1INV)
        ((terminal PHI2INV (TERMTYPE CLOCK)(DIRECTION INPUT)) PHI2INV)
        ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
        ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.19 busseriodpp.sdl

This file creates the scan registers used in the `busserio.sdl` cell. The number of available data connections to the other chips in the array control the number of these subcells that are created. An additional cell is always created for timing purposes.

```
(parent-cell busseriodpp)
;
(parameters N)
;
;
(structure-processor dpp)
(layout-generator Flint a)
;
(subcells (scanreg1t REGISTER ((N N))))
```

```
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
(net PHI1 (NETTYPE CLOCK))
(net PHI2 (NETTYPE CLOCK))
(net PHI1INV (NETTYPE CLOCK))
(net PHI2INV (NETTYPE CLOCK))
;
(instance REGISTER (
        (IN REGIN )
        (BUS REGOUT)
        (SCANIN SCANIN)
        (SCANOUT SCANOUT)
        (LOAD LOAD)
        (LOADINV LOADINV)
        (SCAN SCAN)
        (SCANINV SCANINV)
        (KEEP KEEP)
        (KEEPINV KEEPINV)
        (BUSOE BUSOE)
        (BUSOEINV BUSOEINV)
        (PHI1 PHI1)
        (PHI2 PHI2)
        (PHI1INV PHI1INV)
        (PHI2INV PHI2INV)
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal REGIN (DIRECTION INPUT)) REGIN )
        ((terminal REGOUT (DIRECTION OUTPUT)) REGOUT)
        ((terminal SCANIN (DIRECTION INPUT)) SCANIN)
        ((terminal SCANOUT (DIRECTION OUTPUT)) SCANOUT)
        ((terminal LOAD (DIRECTION INPUT)) LOAD)
        ((terminal LOADINV (DIRECTION INPUT)) LOADINV)
        ((terminal SCAN (DIRECTION INPUT)) SCAN)
        ((terminal SCANINV (DIRECTION INPUT)) SCANINV)
        ((terminal KEEP (DIRECTION INPUT)) KEEP)
        ((terminal KEEPINV (DIRECTION INPUT)) KEEPINV)
        ((terminal BUSOE (DIRECTION INPUT)) BUSOE)
        ((terminal BUSOEINV (DIRECTION INPUT)) BUSOEINV)
        ((terminal PHI1 (TERMTYPE CLOCK)(DIRECTION INPUT)) PHI1)
        ((terminal PHI2 (TERMTYPE CLOCK)(DIRECTION INPUT)) PHI2)
        ((terminal PHI1INV (TERMTYPE CLOCK)(DIRECTION INPUT)) PHI1INV)
        ((terminal PHI2INV (TERMTYPE CLOCK)(DIRECTION INPUT)) PHI2INV)
        ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
        ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.20 busseriologic.sdl

The logic needed to generate the control signals for the registers `busseriodpp.sdl` in the communications cell `busserio.sdl` are implemented using standard cell library parts.

```
(parent-cell busseriologic)
;
(parameters ioLines)
;
(layout-generator Stdcell)
;
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
;
(subcells (norf211 (LOADGATE TLOADGATE))
          (nanf211 (KEEPGATE TKEEPGATE))
          (nanf311 SCANGATE)
          (invf103 (RESETINV BUSOEINV IOSELINV))
)
(dotimes (i ioLines)
         (subcells (aof2201 INMUX)
                   )
)
;
(instance LOADGATE (
                    (A1 BUSLD)
                    (B1 RESET)
                    (O1 LOAD)
                    (O2 LOADINV)
))
;
(instance TLOADGATE (
                    (A1 START)
                    (B1 RESET)
                    (O1 TLOAD)
                    (O2 TLOADINV)
))
;
(instance SCANGATE (
                    (A1 TIMEROUT)
                    (B1 LOADINV)
                    (C1 TLOADINV)
                    (O2 SCAN)
                    (O1 SCANINV)
))
;
(instance KEEPGATE (
                    (A1 LOADINV)
                    (B1 SCANINV)
                    (O2 KEEP)
                    (O1 KEEPINV)
))
;
(instance TKEEPGATE (
                    (A1 TLOADINV)
                    (B1 SCANINV)
                    (O2 TKEEP)
                    (O1 TKEEPINV)
))
;
(instance RESETINV (
                    (A1 RESET)
                    (O RESETINV)
))
;
(instance BUSOEINV (
                    (A1 BUSOE)
                    (O BUSOEINV)
))
;
(instance IOSELINV (
                    (A1 IOSEL)
                    (O IOSELINV)
))
;
(instance INMUX (
              (A1 INPUTA (net-base i))
              (B1 IOSELINV)
              (C2 INPUTB (net-base i))
              (D2 IOSEL)
              (O INPUT (net-base i))
))
;
(instance parent (
                   ((terminal LOAD (DIRECTION OUTPUT)) LOAD)
                   ((terminal LOADINV (DIRECTION OUTPUT)) LOADINV)
                   ((terminal TLOAD (DIRECTION OUTPUT)) TLOAD)
                   ((terminal TLOADINV (DIRECTION OUTPUT)) TLOADINV)
                   ((terminal SCAN (DIRECTION OUTPUT)) SCAN)
                   ((terminal SCANINV (DIRECTION OUTPUT)) SCANINV)
                   ((terminal KEEP (DIRECTION OUTPUT)) KEEP)
                   ((terminal KEEPINV (DIRECTION OUTPUT)) KEEPINV)
                   ((terminal TKEEP (DIRECTION OUTPUT)) TKEEP)
                   ((terminal TKEEPINV (DIRECTION OUTPUT)) TKEEPINV)
                   ((terminal BUSLD (DIRECTION INPUT)) BUSLD)
                   ((terminal START (DIRECTION INPUT)) START)
                   ((terminal TIMEROUT (DIRECTION INPUT)) TIMEROUT)
                   ((terminal RESET (DIRECTION INPUT)) RESET)
                   ((terminal RESETINV (DIRECTION OUTPUT)) RESETINV)
                   ((terminal BUSOE (DIRECTION INPUT)) BUSOE)
                   ((terminal BUSOEINV (DIRECTION OUTPUT)) BUSOEINV)
                   ((terminal INPUTA (DIRECTION INPUT)) INPUTA
(width ioLines))
```

```
                    ((terminal INPUTB (DIRECTION INPUT)) INPUTB
(width ioLines))
                    ((terminal INPUT (DIRECTION OUTPUT)) INPUT
(width ioLines))
                    ((terminal IOSEL (DIRECTION INPUT)) IOSEL)
                    ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
                    ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.21 cmult.sdl

The complex multiplier used in the top level of the diagonal and off-diagonal processor elements is capable of multiplying both complex and extended precision data types. If `COMPLEXOUT` is high, then a complex multiplication is performed. If `COMPB` is high, then the complement of the `B` input is multiplied by the `A`. If `REALA`, then the data on the `A` bus is taken to be real, but not extended precision, and the imaginary part is set to zero.

```
; sdl file for width x width complex multiplier (a[(- (* 2 width) 1)
:0] = real:imag))
; assumes that a,b <= 1.
; ie 010 * 010 = 010
;
; COMPLEXOUT COMPB  OUT
; ------------------------------------------------
;    0      0    AI * BI (24 bits, exponent = exp(a) + exp(b))
;    0      1    AI * AI (24 bits, exponent = 0)
;    1      0    AI * AI - AQ * AQ (16 bits), AQ * BI + AI * BQ (16 bits)
;    1      1    AI * AI + AQ * BQ (16 bits), AQ * BI - AI * BQ (16 bits)
;
; if realA, imag(A)=0
; Sunday 8 April 1994
;
(parent-cell cmult)
;
(parameters inwidth outwidth (realBits 24) (exponentBits (- (* 2 outwidth)
 realBits)))
; note: outwidth <= (- (* 2 inwdith) 1)
; note: inwidth => 6
;
(structure-processor SIVcheck)
(layout-generator Flint b)
;
;
(subcells
        (mult II_MULT ((m inwidth) (n inwidth) (s (length (make-carry-select
(- (* inwidth 2) 3)))) (csindex (make-carry-select (- (* inwidth 2) 3)))))
            (mult IQ_MULT ((m inwidth) (n inwidth) (s (length
(make-carry-select (- (* inwidth 2) 3)))) (csindex (make-carry-select (-
(* inwidth 2) 3)))))
            (mult QI_MULT ((m inwidth) (n inwidth) (s (length
(make-carry-select (- (* inwidth 2) 3)))) (csindex (make-carry-select (-
(* inwidth 2) 3)))))
            (mult QQ_MULT ((m inwidth) (n inwidth) (s (length
(make-carry-select (- (* inwidth 2) 3)))) (csindex (make-carry-select (-
(* inwidth 2) 3)))))
            (cs_adder REAL_ADDER ((N realBits) (NUM_GROUPS (length
(make-carry-select realBits)))  (GROUPING (make-carry-select realBits))
(CELLHEIGHT 118) (BITHEIGHT 100) (FEEDTHRUS 1) (BUFFER 2)))
            (cs_adder IMAG_ADDER ((N outwidth) (NUM_GROUPS (length
(make-carry-select outwidth)))  (GROUPING (make-carry-select outwidth))
(CELLHEIGHT 118) (BITHEIGHT 100) (FEEDTHRUS 1) (BUFFER 2)))
            (negator QQNEG ((N realBits)))
            (zeropass AZERO ((N inwidth)))
            (3to1mux OUTMUX ((N (* 2 outwidth))))
            (cs_adder EXP_ADDER ((N exponentBits) (NUM_GROUPS (length
(make-carry-select exponentBits))) (GROUPING (make-carry-select exponentBits))
 (CELLHEIGHT 118) (BITHEIGHT 100) (FEEDTHRUS 1) (BUFFER 2)))
            (negator COMPNEG ((N inwidth)))
)


;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE SUPPLY))
;
(instance COMPNEG (
                        (IN BQ (width inwidth))
                        (OUT COMPBQ (width outwidth))
                        (CNTL COMPB)
                        (Vdd Vdd)
                        (GND GND)
))

(instance II_MULT (
            (X AI (width inwidth))
            (Y BI (width inwidth))
            (P II_PROD (width realBits) (term-base (- (- (* inwidth 2) 2)
 realBits)))
            (Vdd Vdd)
            (GND GND)
))
;
(instance IQ_MULT (
            (X AI (width inwidth))
            (Y COMPBQ (width inwidth))
            (P IQ_PROD (width outwidth) (term-base (- (- (* inwidth 2) 2)
 outwidth)))
            (Vdd Vdd)
```

```
          (GND GND)
))
;
(instance AZERO (
          (IN AQ (width inwidth))
          (OUT AQP (width inwidth))
          (CNTL REALA)
          (Vdd Vdd)
          (GND GND)
))
(instance QI_MULT (
          (X AQP (width inwidth) )
          (Y BI (width inwidth) )
          (P QI_PROD (width outwidth) (term-base (- (- (* inwidth 2) 2)
 outwidth)))
          (Vdd Vdd)
          (GND GND)
))
;
(instance QQ_MULT (
          (X AQP (width inwidth) )
          (Y BQ (width inwidth) )
          (P QQ_PROD (width realBits) (term-base (- (- (* inwidth 2) 2)
 realBits)))
          (Vdd Vdd)
          (GND GND)
))
;
(instance QQNEG (
          (IN QQ_PROD (width realBits))
          (OUT NEGQQ_PROD (width realBits))
          (CNTL Vdd)
          (Vdd Vdd)
          (GND GND)
))
;
(instance REAL_ADDER (
          (AIN II_PROD (width realBits) (net-base (- realBits 1)) (net-incr
(- 0 1)))
          (BIN NEGQQ_PROD (width realBits) (net-base (- realBits 1))
(net-incr (- 0 1)))
          (CIN Vdd)
          (SUM COMPREAL (term-base (- realBits 1)) (term-incr (- 0 1))
(width realBits))
          (Vdd Vdd (merge 0 (- realBits 1)))
          (GND GND (merge 0 (- realBits 1)))
))
;
(instance IMAG_ADDER (
          (AIN IQ_PROD (width outwidth) (term-base (- outwidth 1)) (term-incr
(- 0 1)))
          (BIN QI_PROD (width outwidth) (term-base (- outwidth 1)) (term-incr
(- 0 1)))
          (CIN GND)
```

```
        (SUM COMPIMAG  (term-base (- outwidth 1)) (term-incr (- 0 1))
(width outwidth))
        (Vdd Vdd (merge 0 (- outwidth 1)))
        (GND GND (merge 0 (- outwidth 1)))
))
;
(instance EXP_ADDER (
                    (AIN AQ (term-base (- exponentBits 1)) (term-incr
(- 0 1)) (width exponentBits))
                    (BIN BQ (term-base (- exponentBits 1)) (term-incr
(- 0 1)) (width exponentBits))
                    (CIN GND)
                    (SUM EXPONENT (width exponentBits))
                    (Vdd Vdd (merge 0 (- exponentBits 1)))
                    (GND GND (merge 0 (- exponentBits 1)))
))
;
(instance OUTMUX (
                  (C COMPIMAG (term-base 0)  (width outwidth))
                  (C COMPREAL (net-base (- realBits outwidth))
(width outwidth) (term-base outwidth))
                  (A II_PROD (term-base exponentBits) (width realBits))
                  (A EXPONENT (width exponentBits))
                  (B COMPREAL (term-base exponentBits) (width realBits))
                  (B GND (merge 0 (- exponentBits 1)))
                  (OUT REAL (term-base outwidth) (width outwidth))
                  (OUT IMAG (term-base 0) (width outwidth))
                  (SEL COMPB (term-base 0))
                  (SEL COMPLEXOUT (term-base 1))
                  (Vdd Vdd)
                  (GND GND)
))
;
(instance parent(
                  ((terminal AI (DIRECTION INPUT)) AI (width inwidth))
                  ((terminal AQ (DIRECTION INPUT)) AQ (width inwidth))
                  ((terminal BI (DIRECTION INPUT)) BI (width inwidth))
                  ((terminal BQ (DIRECTION INPUT)) BQ (width inwidth))
                  ((terminal II_PROD (DIRECTION OUTPUT)) II_PROD
(width outwidth))
                  ((terminal IQ_PROD (DIRECTION OUTPUT)) IQ_PROD
(width outwidth))
                  ((terminal QI_PROD (DIRECTION OUTPUT)) QI_PROD
(width outwidth))
                  ((terminal QQ_PROD (DIRECTION OUTPUT)) QQ_PROD
(width outwidth))
                  ((terminal REAL (DIRECTION OUPUT)) REAL (width outwidth))
                  ((terminal IMAG (DIRECTION OUPUT)) IMAG (width outwidth))
                  ((terminal COMPB (DIRECTION INPUT)) COMPB)
                  ((terminal COMPLEXOUT (DIRECTION INPUT)) COMPLEXOUT)
                  ((terminal REALA (DIRECTION INPUT)) REALA)
                  ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
                  ((terminal GND (TERMTYPE GROUND)) GND)
))
```

```
;
(end-sdl)
```

A.1.22 deltarom.sdl

The lookup table used in the interpolation cell is generated from a file of data value. It has been arranged so that the most significant bit of the input to this lookup table selects the function being calculated. If it is a zero, then the data for the square root calculation is provided. If it is a one, then the data for the inverse square root is provided. This file contains the values for $\delta_{m_0}(x_i)$.

```
; Automatically Generated PLA cell file to be used as ROM.  DO NOT EDIT!
; Genereated Monday, 11 April 1994, 06:39:53 AM.
;
(parent-cell deltarom)
;
(layout-generator Flint b)
;
(subcells
        (latch LATCH ((width 19)))
        (pla ROM (
                        (inwidth 9)
                        (outwidth 19)
                        (input-plane '(
                                "010000000"
                                "101001001"
                                "000101011"
                                "001011000"
                                "001010000"
                                "001000000"
                                "101010010"
                                "10011011-"
                                "00101011-"
                                "0001-1110"
                                "001011101"
                                "101111011"
                                "000111100"
                                "101100011"
                                "001011100"
                                "10-100110"
                                "001000001"
                                "000110100"
                                "0011111-0"
                                "000110000"
                                "-00101010"
                                "001001-01"
                                "1001111-0"
                                "0010-1110"
```

```
"-00110001"
"0001110-0"
"001011001"
"001110111"
"-010-1110"
"000111101"
"10100--11"
"101101011"
"000100101"
"101100101"
"-0-110111"
"00100-010"
"-01-11110"
"00-11-010"
"1011111-1"
"-00101000"
"00-1000-0"
"1011-1111"
"00011-000"
"001100-11"
"000101-10"
"00010000-"
"0010110-0"
"001010-00"
"0010-0101"
"001001-00"
"00011101-"
"00010-000"
"100110-01"
"00100-00-"
"000110011"
"000110-0-"
"101110100"
"-011-0011"
"001-00000"
"00-100001"
"101101001"
"001000011"
"001010011"
"001110000"
"001011011"
"001010001"
"001000111"
"000100011"
"00110111-"
"00100-101"
"001-011-0"
"001000100"
"000100100"
"0010-0110"
"00110110-"
"001110-10"
"0010101-1"
"101011-10"
```

```
"1011100-1"
"000101-11"
"-00110111"
"00101111-"
"001100-10"
"00110010-"
"-0-11-111"
"001-001--"
"-011-0111"
"001101001"
"10111-1-1"
"001110001"
"001111001"
"-011011-1"
"001111000"
"0010011--"
"000101101"
"000111001"
"1011001-0"
"101111001"
"001001011"
"001101000"
"000100010"
"0011-0100"
"000100110"
"000110010"
"0011-0101"
"00110-010"
"00111110-"
"00110001-"
"0010-0010"
"00-101111"
"0011-001-"
"000101001"
"000101100"
"000110110"
"00111101-"
"00111111-"
"00011111-"
"001101011"
"110000000"
"000100111"
"100100-00"
"100100111"
"100110001"
"100100110"
"100100001"
"100100010"
"100101000"
"100111101"
"100110010"
"101001011"
"100111011"
"100111010"
```

```
"100110000"
"100100101"
"100101011"
"100101100"
"100100011"
"1010-1010"
"101010011"
"101000111"
"101000100"
"101010100"
"101000101"
"10-111110"
"101001101"
"100101111"
"100111111"
"101000000"
"100101101"
"101001100"
"100101010"
"10-100100"
"100101110"
"10100111-"
"100111001"
"100110-11"
"1010000-1"
"101011101"
"101011011"
"101010110"
"100111-00"
"101010101"
"101011001"
"10101-111"
"100101001"
"100110110"
"10-11010-"
"10101111-"
"1010100-0"
"10100001-"
"101101110"
"101011100"
"101010001"
"101101000"
"101011000"
"101100-10"
"101101100"
"101111000"
"101110010"
"101101010"
"101110000"
"101000110"
"10100100-"
"101111100"
"101100000"
"10111-110"
```

```
            "1011----1"
            "101111-1-"
))
(output-plane '(
            "000000000010101011"
            "000000000000010010"
            "000000000101000000"
            "000000000000001010"
            "000000000000011000"
            "000000000001100000"
            "000000000000011000"
            "000000000010000000"
            "000000000000010000"
            "000000000000010000"
            "000000000100010100"
            "000000000000000011"
            "000000001000010100"
            "000000000000000101"
            "000000000100011000"
            "000000000000001000"
            "000000000001011000"
            "000000000000010011"
            "000000000000000010"
            "000000000000100101"
            "000000000010000100"
            "000000000000001100"
            "000000000010100000"
            "000000000000000101"
            "000000000001010000"
            "000000000000001100"
            "000000000100100110"
            "000000000010011000"
            "000000000000000010"
            "000000001000000111"
            "000000000000010000"
            "000000000000111000"
            "000000010001001010"
            "000000000000011001"
            "000000000000000010"
            "000000000110000100"
            "000000000000001000"
            "000000000000000010"
            "000000000000011000"
            "000000001010010000"
            "000000000000000100"
            "000000000000001001"
            "000000001001000010"
            "000000000001001000"
            "000000001100001000"
            "000000010000010100"
            "000000000100100001"
            "000000000101000001"
            "000000000000001011"
            "000000000000010101"
```

```
"0000000001000100001"
"0000000000101000001"
"0000000000000111000"
"0000000000110000000"
"0000000001010100110"
"0000000001010000000"
"0000000010000011100"
"0000000000000001000"
"0000000000100000011"
"0000000000100000011"
"0000000000000101110"
"0000000000111000011"
"0000000000101000111"
"0000000000011010001"
"0000000000100011101"
"0000000000101010011"
"0000000000100011101"
"0000000010010101001"
"0000000000010010010"
"0000000000100100110"
"0000000000001000100"
"0000000000100111001"
"0000000010001111000"
"0000000000100100110"
"0000000000010011000"
"0000000000011000001"
"0000000000100110001"
"0000000000011000011"
"0000000000001010001"
"0000000001000101100"
"0000000001001011000"
"0000000000100001011"
"0000000000001110000"
"0000000000000110001"
"0000000000000000100"
"0000000000010000000"
"0000000000000100100"
"0000000000011100110"
"0000000000001100000"
"0000000000011001110"
"0000000000010111010"
"0000000000001000001"
"0000000000010111100"
"0000000000101100000"
"0000000001100110011"
"0000000001000111110"
"0000000101100010000"
"0000000000001101101"
"0000000000101111100"
"0000000000011101001"
"0000000010011011010"
"0000000000011000110"
"0000000010000011111"
"0000000001010111001"
```

```
"000000000000011000011"
"000000000000011100010"
"000000000000010110001"
"000000000000000110011"
"000000000000101001101"
"000000000000011010011"
"000000000000011001000"
"000000000001110101101"
"000000000001101001111"
"000000000001001101111"
"000000000000010110101"
"000000000000010101101"
"000000000000111101010"
"000000000000011011111"
"111111111111110000000"
"000000000001111110111"
"111111100000000000100"
"111111101100100000000"
"111111111010000000101"
"111111101010110010000"
"111111100010011001 0"
```

```
                                        "1111111111011010100"
                                        "1111111111010100110"
                                        "1111111110000001101"
                                        "1111111111010011100"
                                        "1111111111011000011"
                                        "1111111111010110000"
                                        "1111111011101100101"
                                        "1111111101100101101"
                                        "1111111101101000000"
                                        "1111111111001100010"
                                        "1111111111001100010"
                                        "1111111110101100010"
                                        "1111111111101000101"
                                        "1111111111011011100"
                                        "1111111111001101110"
                                        "1111111111100101001"
                                        "1111111111010111001"
                                        "1111111111100000110"
                                        "1111111111100111100"
                                        "1111111111101101010"
                                        "1111111111101010101"
                                        "1111111111100110011"
                                        "1111111111101001101"
                                        "1111111110110111101"
                                        "1111111110111100101"
                                        "1111111111101110101"
                                        "1111111111011111001"
                                        "1111111111101100011"
                                        "1111111111100000000"
                                        "1111111111101110000"
                                ) )
                                (minterm 188)
            ) )
)
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
(net PHI (NETTYPE CLOCK ))
(net PHI1 (NETTYPE CLOCK ))
(net PHI2 (NETTYPE CLOCK ))
;
(instance ROM (
            (IN ADDRESS (width 9))
            (OUT PLAOUT (width 19))
            (CLOCK PHI)
            (Vdd Vdd)
            (GND GND)
))
;
(instance LATCH (
            (IN PLAOUT (width 19))
            (OUT DATA (width 19))
            (PHI1 PHI1)
            (PHI2 PHI2)
```

```
        (LD Vdd)
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal ADDRESS (DIRECTION INPUT)) ADDRESS (width 9))
        ((terminal DATA (DIRECTION OUTPUT)) DATA (width 19))
        ((terminal PHI (TERMTYPE CLOCK) (DIRECTION INPUT)) PHI )
        ((terminal PHI1 (TERMTYPE CLOCK) (DIRECTION INPUT)) PHI1 )
        ((terminal PHI2 (TERMTYPE CLOCK) (DIRECTION INPUT)) PHI2 )
        ((terminal Vdd (TERMTYPE SUPPLY) (TERM_EDGE TOP)) Vdd)
        ((terminal GND (TERMTYPE GROUND)(TERM_EDGE BOTTOM)) GND)
))
;
(end-sdl)
```

A.1.23 diag2.sdl

This file is the core of the diagonal processor elements. It takes as parameters the number of registers to be tiled into the register file `regfile2p.sdl`, and the width to be used for all of the subcells. The data comes in through 4 bit wide serial links. The core also has connections to determine whether or not the processor is on an edge of the array. This cell implements a barrel shifter `bus2bshift.sdl`, a fast interpolation cell `bus2interp.sdl`, a complex multiplier `bus2cmult.sdl`, a complex adder `bus2adder.sdl`, a register file `regfile2p.sdl`, and several semi-serial input output registers `busserio.sdl`.

```
(parent-cell diag2)
;
(parameters width numOfRegs (ioPerChannel 4) (ioChannels 6) (realBits 24)
(exponentBits (- (* width 2) realBits)))
; Note: Width must be even, and is total number of real+imag bits.
;
(layout-generator Flint b v)
;
(subcells
;
; note: floor is used for in/outwidth to force an error if width was odd.
; if width was odd, floor will round down, so that (* 2 (floor (/ width 2)))
; will be less than width, and errors will result due to unknown terminals.
        (bus2adder ADDER ((inwidth (floor (/ width 2))) (outwidth (floor
(/ width 2)))))
        (bus2cmult MULT  ((inwidth (floor (/ width 2))) (outwidth (floor
(/ width 2)))))
```

```
        (regfile2p REGFILE ((numOfRegs numOfRegs) (width width)))
        (bus2interp INTERP ((busWidth width)))
        (bus2bshift BSHIFT ((realBits realBits) (busWidth width)))
        (myclock CLKGEN)
        (diagctl FSM)
)
;
(dotimes (i ioChannels)
        (subcells
         (busserio BUSIO ((ioLines ioPerChannel) (busWidth width)))
        )
)

;
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE Vdd))
(net PHI (NETTYPE CLOCK))
;
(instance REGFILE (
        (ABUS ABUS (width width))
        (BBUS BBUS (width width))
        (EBUS EBUS (width width))
        (AOUTEN AOUTEN (width numOfRegs))
        (BOUTEN BOUTEN (width numOfRegs))
        (LOAD LOAD (width numOfRegs))
        (SCAN GND (merge 0 (- numOfRegs 1)))
        (PHI PHI)
        (Vdd Vdd)
        (GND GND)
))
;
(instance ADDER (
        (ABUS ABUS (width width))
        (BBUS BBUS (width width))
        (EBUS EBUS (width width))
        (OUTEN ADDEROUTEN)
        (FUNC ADDERFUNC)
        (REALADD ADDERREALADD)
        (Vdd Vdd)
        (GND GND)
))
;
(instance MULT (
        (ABUS ABUS (width width))
        (BBUS BBUS (width width))
        (EBUS EBUS (width width))
        (COMPB MULTCOMPB)
        (COMPLEXOUT MULTCOMPLEXOUT)
        (REALA MULTREALA)
        (OUTEN MULTOUTEN)
        (Vdd Vdd)
        (GND GND)
))
```

```
;
(instance BUSIO (
                    (EBUS EBUS (width width))
                    (ABUS ABUS (width width))
                    (INPUTA INPUTA (net-base (* i ioPerChannel))
(width ioPerChannel))
;                   (OUTPUTA OUTPUTA (net-base (* i ioPerChannel))
(width ioPerChannel))
                    (INPUTB INPUTB (net-base (* i ioPerChannel))
(width ioPerChannel))
;                   (OUTPUTB OUTPUTB (net-base (* i ioPerChannel))
(width ioPerChannel))
                    (OUTPUT OUTPUT (net-base (* i ioPerChannel))
(width ioPerChannel))
                    (READY IOREADY (net-base i))
                    (BUSOE IOBUSOE (net-base i))
                    (BUSLD IOBUSLD (net-base i))
                    (START IOSTART (net-base i))
                    (IOSEL IOSEL)
                    (RESET RESET)
                    (PHI1 PHI1)
                    (PHI2 PHI2)
                    (PHI1INV PHI1INV)
                    (PHI2INV PHI2INV)
                    (Vdd Vdd)
                    (GND GND)
))
;
(instance CLKGEN (
                    (CLK PHI)
                    (PHI1 PHI1)
                    (PHI2 PHI2)
                    (PHI1INV PHI1INV)
                    (PHI2INV PHI2INV)
                    (Vdd Vdd)
                    (GND GND)
))
;
(instance INTERP (
                    (ABUS ABUS (width realBits) (term-base (- width realBits)
))
                    (EBUS EBUS (width width))
                    (GO INTERPGO)
                    (RESET RESET)
                    (FUNC INTERPFUNC)
                    (BUSY INTERPBUSY)
                    (OUTEN INTERPOUTEN)
                    (PHI PHI)
                    (Vdd Vdd)
                    (GND GND)
))
;
(instance BSHIFT (
                    (ABUS ABUS (width width))
                    (EBUS EBUS (width width))
                    (OUTEN BSHIFTOUTEN)
                    (Vdd Vdd)
                    (GND GND)
))
;
(instance FSM (
                (RESET RESET)
                (IOREADY IOREADY (width ioChannels))
                (INCCYCLE INCCYCLE)
                (FINALCYCLE FINALCYCLE)
                (NEXTDATA NEXTDATA)
                (LEFTEDGE LEFTEDGE)
                (TOPEDGE TOPEDGE)
                (INTERPBUSY INTERPBUSY)
                (IOBUSOE IOBUSOE (width ioChannels))
                (IOBUSLD IOBUSLD (width ioChannels))
                (IOSTART IOSTART (width ioChannels))
                (IOSEL IOSEL)
                (ADDERFUNC ADDERFUNC)
                (ADDEROUTEN ADDEROUTEN)
                (ADDERREALADD ADDERREALADD)
                (MULTOUTEN MULTOUTEN)
                (MULTCOMPB MULTCOMPB)
                (MULTCOMPLEXOUT MULTCOMPLEXOUT)
                (MULTREALA MULTREALA)
                (INTERPGO INTERPGO)
                (INTERPFUNC INTERPFUNC)
                (INTERPOUTEN INTERPOUTEN)
                (BSHIFTOUTEN BSHIFTOUTEN)
                (AOUTEN AOUTEN (width numOfRegs))
                (BOUTEN BOUTEN (width numOfRegs))
                (REGLOAD LOAD (width numOfRegs))
                (PHI1 PHI1)
                (PHI2 PHI2)
                (Vdd Vdd)
                (GND GND)
))
;
(instance parent (
                    ((terminal ABUS (DIRECTION OUTPUT)) ABUS (width width))
                    ((terminal BBUS (DIRECTION OUTPUT)) BBUS (width width))
                    ((terminal EBUS (DIRECTION OUTPUT)) EBUS (width width))
                    ((terminal INPUTA (DIRECTION INPUT)) INPUTA (width
(* ioPerChannel ioChannels)))
                    ((terminal INPUTB (DIRECTION INPUT)) INPUTB (width
(* ioPerChannel ioChannels)))
                    ((terminal OUTPUT (DIRECTION OUTPUT)) OUTPUT (width
(* ioPerChannel ioChannels)))
                    ((terminal IOBUSOE (DIRECTION OUTPUT)) IOBUSOE
(width ioChannels))
                    ((terminal IOBUSLD (DIRECTION OUTPUT)) IOBUSLD
(width ioChannels))
                    ((terminal IOREADY (DIRECTION OUTPUT)) IOREADY
(width ioChannels))
                    ((terminal IOSTART (DIRECTION OUTPUT)) IOSTART
(width ioChannels))

                    ((terminal AOUTEN (DIRECTION OUTPUT)) AOUTEN
(width numOfRegs))
                    ((terminal BOUTEN (DIRECTION OUTPUT)) BOUTEN
(width numOfRegs))
                    ((terminal LOAD (DIRECTION OUTPUT)) LOAD
(width numOfRegs))
                    ((terminal ADDEROUTEN (DIRECTION OUTPUT)) ADDEROUTEN)
                    ((terminal MULTOUTEN (DIRECTION OUTPUT)) MULTOUTEN)
                    ((terminal RESET (DIRECTION INPUT)) RESET)
                    ((terminal INCCYCLE (DIRECTION INPUT)) INCCYCLE)
                    ((terminal FINALCYCLE (DIRECTION INPUT)) FINALCYCLE)
                    ((terminal NEXTDATA (DIRECTION INPUT)) NEXTDATA)
                    ((terminal LEFTEDGE (DIRECTION INPUT)) LEFTEDGE)
                    ((terminal TOPEDGE (DIRECTION INPUT)) TOPEDGE)
                    ((terminal PHI (TERMTYPE CLOCK) (DIRECTION INPUT)) PHI)
                    ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
                    ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.24 diag2chip.sdl

This file puts the pad frame around the `diag2.sdl` core of the diagonal processor. The chip inputs are put on the left side, the outputs are along the right, and the control connections come in on the top of the chip.

```
(parent-cell diag2chip)
;
(structure-processor Padgroup)
(layout-generator Padroute lpadsl_2)
;
(parameters width numOfRegs (ioPerChannel 4) (diagChannels 4) (hvChannels 2)
(ioChannels (+ hvChannels diagChannels)) (ioLines (* ioPerChannel ioChannels))
            (max_pads_per_side 30)
            (pads (list
                    (list "top" 1 max_pads_per_side)
                    (list "left" (+ max_pads_per_side 1)
(* max_pads_per_side 2))
                    (list "bottom" (+ (* max_pads_per_side 2) 1)
(* max_pads_per_side 3))
                    (list "right" (+ (* max_pads_per_side 3) 1)
(* max_pads_per_side 4))
                    )
                    )
```

```
              )
;
(subcells
 (diag2 CORE ((width width) (numOfRegs numOfRegs)))
 (vdd1_2 (VDD_TOP_1 VDD_TOP_2  VDD_RIGHT_1
          VDD_RIGHT_2  VDD_LEFT_1 VDD_LEFT_2 )
 )
 (gnd1_2
(GND_TOP_1 GND_TOP_2 GND_TOP_3 GND_TOP_4 GND_LEFT_1 GND_LEFT_2
           GND_LEFT_3 GND_LEFT_4 GND_RIGHT_1 GND_RIGHT_2))

 (in1_2
(RESET_PAD INCCYCLE_PAD FINALCYCLE_PAD NEXTDATA_PAD
          LEFTEDGE_PAD TOPEDGE_PAD PHI_PAD))

 )
(dotimes (i (* ioPerChannel ioChannels))
           (subcells
            (in1_2 (INPUTA_PAD INPUTB_PAD))
            (out1_2 OUTPUT_PAD)
            )
)
(dotimes (dummy_top_left (- (round (/ max_pads_per_side 2)) (+ 7 3)))
           (subcells
            (space1_2 DUMMY_PADS_TOP_LEFT)
            ))
(dotimes (dummy_top_right (- max_pads_per_side (+ (round
(/ max_pads_per_side 2)) 3)))
           (subcells
            (space1_2 DUMMY_PADS_TOP_RIGHT)
            ))
(dotimes (dummy_bottom (- (- (* max_pads_per_side 4) (+ (* 2
(* ioPerChannel ioChannels)) 6)) (+ max_pads_per_side (+
(* ioPerChannel ioChannels)) 4)))
           (subcells
            (space1_2 DUMMY_PADS_BOTTOM)
            ))

;
(net VDD (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
(net PHI (NETTYPE CLOCK))
;
(instance CORE (
                    (INPUTA INPUTA (width (* ioPerChannel ioChannels)))
                    (INPUTB INPUTB (width (* ioPerChannel ioChannels)))
                    (OUTPUT OUTPUT (width ioLines))
                    (RESET RESET)
                    (INCCYCLE INCCYCLE)
                    (FINALCYCLE FINALCYCLE)
                    (NEXTDATA NEXTDATA)
                    (LEFTEDGE LEFTEDGE)
                    (TOPEDGE TOPEDGE)
                    (Vdd Vdd)
                    (GND GND)
                    (PHI PHI)
))
;
(instance INPUTA_PAD (PAD (- (- (* max_pads_per_side 4) 2) i)) (

(padin INPUTA_p (net-base i))

(in INPUTA (net-base i))
                                                    )
)
;
(instance INPUTB_PAD (PAD (- (- (* max_pads_per_side 4) (+ ioLines 4)) i)) (

(padin INPUTB_p (net-base i))

(in INPUTB (net-base i))
                                                    )
)
;
(instance OUTPUT_PAD (PAD (+ (+ max_pads_per_side 3) i)) (

(pado OUTPUT_p (net-base i))

(out OUTPUT (net-base i))
))
;
(instance RESET_PAD (PAD '3) (
                              (padin RESET_p)
                              (in RESET)
                              ))
;
(instance INCCYCLE_PAD (PAD '4) (
                              (padin INCCYCLE_p)
                              (in INCCYCLE)
                              ))
;
(instance FINALCYCLE_PAD (PAD '5) (
                              (padin FINALCYCLE_p)
                              (in FINALCYCLE)
                              ))
;
(instance NEXTDATA_PAD (PAD '6) (
                              (padin NEXTDATA_p)
                              (in NEXTDATA)
                              ))
;
(instance LEFTEDGE_PAD (PAD '7) (
                              (padin LEFTEDGE_p)
                              (in LEFTEDGE)
                              ))
;
(instance TOPEDGE_PAD (PAD '8) (
                              (padin TOPEDGE_p)
```

```
                                        (in TOPEDGE)
                                        ))
;
(instance PHI_PAD (PAD '9) (
                                        (padin PHI_p)
                                        (in PHI)
                                        ))
;
(instance VDD_TOP_1 (PAD (+ (round (/ max_pads_per_side 2)) 0)) (

(padvdd Vdd_p)

(Vdd Vdd)
                                                                           ))
(instance VDD_TOP_2 (PAD (+ (round (/ max_pads_per_side 2)) 1)) (

(padvdd Vdd_p)

(Vdd Vdd)
                                                                           ))
(instance VDD_LEFT_1 (PAD (- (* max_pads_per_side 4) (+ ioLines 2))) (
                                                                                  (padvdd Vdd_p)

                                                                                  (Vdd Vdd)

                                                                                  ))

(instance VDD_LEFT_2 (PAD (- (* max_pads_per_side 4) (+ ioLines 3))) (
                                                                                  (padvdd Vdd_p)

                                                                                  (Vdd Vdd)

                                                                                  ))

(instance VDD_RIGHT_1 (PAD (+ max_pads_per_side (+ ioLines 3))) (
                                                                              (padvdd Vdd_p)

                                                                              (Vdd Vdd)

                                                                              ))

(instance VDD_RIGHT_2 (PAD (+ max_pads_per_side (+ ioLines 4))) (
                                                                              (padvdd Vdd_p)

                                                                              (Vdd Vdd)

                                                                              ))

;
(instance GND_TOP_1 (PAD '1) (
                                        (padgnd GND_p)
                                        (GND GND)
                                        ))
(instance GND_TOP_2 (PAD '2) (
                              (padgnd GND_p)
                              (GND GND)
                              ))
(**instance** GND_TOP_3 (PAD (– max_pads_per_side 1)) (
                                        (padgnd GND_p)
                                        (GND GND)
                                        ))
(**instance** GND_TOP_4 (PAD max_pads_per_side) (
                                        (padgnd GND_p)
                                        (GND GND)
                                        ))
(**instance** GND_LEFT_1 (PAD (– (* max_pads_per_side 4) 1)) (
                                           (padgnd GND_p)
                                           (GND GND)
                                           ))
(**instance** GND_LEFT_2 (PAD (* max_pads_per_side 4)) (
                                           (padgnd GND_p)
                                           (GND GND)
                                           ))
(**instance** GND_LEFT_3 (PAD (– (* max_pads_per_side 4) (+ (* 2
(* ioPerChannel ioChannels)) 4))) (
                                           (padgnd GND_p)
                                           (GND GND)
                                           ))
(**instance** GND_LEFT_4 (PAD (– (* max_pads_per_side 4) (+ (* 2
(* ioPerChannel ioChannels)) 5))) (
                                           (padgnd GND_p)
                                           (GND GND)
                                           ))
(**instance** GND_RIGHT_1 (PAD (+ max_pads_per_side 1)) (
                                           (padgnd GND_p)
                                           (GND GND)
                                           ))
(**instance** GND_RIGHT_2 (PAD (+ max_pads_per_side 2)) (
                                           (padgnd GND_p)
                                           (GND GND)
                                           ))
;
(**instance** DUMMY_PADS_TOP_LEFT (PAD (+ (+ 7 3) dummy_top_left)))
(**instance** DUMMY_PADS_TOP_RIGHT (PAD (+ (+ (round (/ max_pads_per_side 2)) 2)
 dummy_top_right)))
(**instance** DUMMY_PADS_BOTTOM (PAD (+ (+ max_pads_per_side (+
(* ioPerChannel ioChannels) 5)) dummy_bottom)))
;
(**instance parent** (
                    ((**terminal** INPUTA_p (**DIRECTION** INPUT)) INPUTA_p (**width**
(* ioPerChannel ioChannels)))
                    ((**terminal** INPUTB_p (**DIRECTION** INPUT)) INPUTB_p (**width**
(* ioPerChannel ioChannels)))
                    ((**terminal** OUTPUT_p (**DIRECTION** OUTPUT)) OUTPUT_p (**width**
(* ioPerChannel ioChannels)))
                    ((**terminal** RESET_p (**DIRECTION** INPUT)) RESET_p)
                    ((**terminal** INCCYCLE_p (**DIRECTION** INPUT)) INCCYCLE_p)
                    ((**terminal** FINALCYCLE_p (**DIRECTION** INPUT)) FINALCYCLE_p)
```

```
                ((terminal NEXTDATA_p (DIRECTION INPUT)) NEXTDATA_p)
                ((terminal LEFTEDGE_p (DIRECTION INPUT)) LEFTEDGE_p)
                ((terminal TOPEDGE_p (DIRECTION INPUT)) TOPEDGE_p)
                ((terminal PHI_p (TERMTYPE CLOCK) (DIRECTION INPUT))
 PHI_p)
                ((terminal Vdd_p (TERMTYPE SUPPLY) (DIRECTION INPUT))
 Vdd_p)
                ((terminal GND_p (TERMTYPE GROUND) (DIRECTION INPUT))
 GND_p)
))
;
(end-sdl)
```

A.1.25 diagctl.sdl

This is the controller for the diagonal processor element. It takes as input a global synchronization signal `INCCYCLE`, a reset signal `RESET`, a completion signal, `FINALCYCLE`, and all of the status lines from the functional units. The operation is controlled by the behavioral description file `diagctl.bds`.

```
(parent-cell diagctl)
;
(parameters (ioChannels 6) (stateBits 6) (cycleBits 3) (subStateBits 2)
(numOfRegs 16))
(layout-generator Flint b v)
;
(subcells
        (fsm_bdsyn FSM ((inwidth 24) (outwidth 89)
(bdsyn "~/sdl/diagctl/diagctl.bds")))
)
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
(net PHI1 (NETTYPE CLOCK))
(net PHI2 (NETTYPE CLOCK))
;
(instance FSM (
             (IN reset (term-base 0))
             (IN ioReady (term-base 1) (width ioChannels))
             (IN incCycle (term-base (+ ioChannels 1)))
             (IN finalCycle (term-base (+ ioChannels 2)))
             (IN nextData (term-base (+ ioChannels 3)))
             (IN leftEdge (term-base (+ ioChannels 4)))
             (IN topEdge (term-base (+ ioChannels 5)))
             (IN interpBusy (term-base (+ ioChannels 6)))
             (IN cycle (term-base (+ ioChannels 7)) (width cycleBits))
             (IN state (term-base (+ (+ ioChannels 7) cycleBits))
(width stateBits))
```

```
                (IN subState (term−base (+ (+ (+ ioChannels 7) cycleBits)
 stateBits))  (width subStateBits))
                (OUT iobusoe (term−base 0) (width ioChannels))
                (OUT iobusld (term−base ioChannels) (width ioChannels))
                (OUT iostart (term−base (* 2 ioChannels)) (width ioChannels))
                (OUT ioSel (term−base (* 3 ioChannels)))
                (OUT adderFunc (term−base (+ (* 3 ioChannels) 1)))
                (OUT adderOutEn (term−base (+ (* 3 ioChannels) 2)))
                (OUT adderRealAdd (term−base (+ (* 3 ioChannels) 3)))
                (OUT multOutEn (term−base (+ (* 3 ioChannels) 4)))
                (OUT multCompB (term−base (+ (* 3 ioChannels) 5)))
                (OUT multComplexOut (term−base (+ (* 3 ioChannels) 6)))
                (OUT multRealA (term−base (+ (* 3 ioChannels) 7)))
                (OUT interpGo (term−base (+ (* 3 ioChannels) 8)))
                (OUT interpFunc (term−base (+ (* 3 ioChannels) 9)))
                (OUT interpOutEn (term−base (+ (* 3 ioChannels) 10)))
                (OUT bshiftOutEn (term−base (+ (* 3 ioChannels) 11)))
                (OUT aOutEn (term−base (+ (* 3 ioChannels) 12))
(width numOfRegs))
                (OUT bOutEn (term−base (+ (+ (* 3 ioChannels) 12) numOfRegs))
 (width numOfRegs))
                (OUT regLoad (term−base (+ (+ (* 3 ioChannels) 12)
(* 2 numOfRegs))) (width numOfRegs))
                (OUT cycle (term−base (+ (+ (* 3 ioChannels) 12)
(* 3 numOfRegs))) (width cycleBits))
                (OUT state (term−base (+ (+ (+ (* 3 ioChannels) 12)
(* 3 numOfRegs)) cycleBits)) (width stateBits))
                (OUT subState (term−base (+ (+ (+ (+ (* 3 ioChannels) 12)
(* 3 numOfRegs)) cycleBits) stateBits)) (width subStateBits))
                (PHI1 PHI1)
                (PHI2 PHI2)
                (Vdd Vdd)
                (GND GND)
))
;
(instance parent (
                   ((terminal RESET (DIRECTION INPUT)) reset)
                   ((terminal IOREADY (DIRECTION INPUT)) ioReady
(width ioChannels))
                   ((terminal INCCYCLE (DIRECTION INPUT)) incCycle)
                   ((terminal FINALCYCLE (DIRECTION INPUT)) finalCycle)
                   ((terminal NEXTDATA (DIRECTION INPUT)) nextData)
                   ((terminal LEFTEDGE (DIRECTION INPUT)) leftEdge)
                   ((terminal TOPEDGE (DIRECTION INPUT)) topEdge)
                   ((terminal INTERPBUSY (DIRECTION INPUT)) interpBusy)
                   ((terminal IOBUSOE (DIRECTION OUTPUT)) iobusoe
(width ioChannels))
                   ((terminal IOBUSLD (DIRECTION OUTPUT)) iobusld
(width ioChannels))
                   ((terminal IOSTART (DIRECTION OUTPUT)) iostart
(width ioChannels))
                   ((terminal IOSEL (DIRECTION OUTPUT)) ioSel)
                   ((terminal ADDERFUNC (DIRECTION OUTPUT)) adderFunc )
                   ((terminal ADDEROUTEN (DIRECTION OUTPUT)) adderOutEn)
```

```
                        ((terminal ADDERREALADD (DIRECTION OUTPUT)) adderRealAdd)
                        ((terminal MULTOUTEN (DIRECTION OUTPUT)) multOutEn)
                        ((terminal MULTCOMPB (DIRECTION OUTPUT)) multCompB)
                        ((terminal MULTCOMPLEXOUT (DIRECTION OUTPUT))
 multComplexOut)
                        ((terminal MULTREALA (DIRECTION OUTPUT)) multRealA)
                        ((terminal INTERPGO (DIRECTION OUTPUT)) interpGo)
                        ((terminal INTERPFUNC (DIRECTION OUTPUT)) interpFunc)
                        ((terminal INTERPOUTEN (DIRECTION OUTPUT)) interpOutEn)
                        ((terminal BSHIFTOUTEN (DIRECTION OUTPUT)) bshiftOutEn)
                        ((terminal AOUTEN (DIRECTION OUTPUT)) aOutEn
(width numOfRegs))
                        ((terminal BOUTEN (DIRECTION OUTPUT)) bOutEn
(width numOfRegs))
                        ((terminal REGLOAD (DIRECTION OUTPUT)) regLoad
(width numOfRegs))
                        ((terminal CYCLE (DIRECTION OUTPUT)) cycle
(width cycleBits))
                        ((terminal STATE (DIRECTION OUTPUT)) state
(width stateBits))
                        ((terminal PHI1 (DIRECTION INPUT)
                                (TERMTYPE CLOCK)) PHI1)
                        ((terminal PHI2 (DIRECTION INPUT)
                                (TERMTYPE CLOCK)) PHI2)
                        ((terminal Vdd (DIRECTION INPUT)
                                (TERMTYPE SUPPLY)) Vdd)
                        ((terminal GND (DIRECTION INPUT)
                                (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.26 From.sdl

The lookup table used in the interpolation cell is generated from a file of data value. It has been arranged so that the most significant bit of the input to this lookup table selects the function being calculated. If it is a zero, then the data for the square root calculation is provided. If it is a one, then the data for the inverse square root is provided. This file contains the values for $f(x_i)$.

```
; Automatically Generated PLA cell file to be used as ROM.  DO NOT EDIT!
; Genereated Monday, 11 April 1994, 06:35:32 AM.
;
(parent-cell From)
;
(layout-generator Flint b)
;
(subcells
(latch LATCH ((width 27)))
(pla ROM (
              (inwidth 9)
              (outwidth 27)
              (input–plane '(
                     "010000000"
                     "110000000"
                     "-00100000"
                     "001111111"
                     "00-11--1-"
                     "000111110"
                     "101111110"
                     "-01111111"
                     "0011---1-"
                     "001-01001"
                     "001--1---"
                     "000100101"
                     "001001101"
                     "101111100"
                     "001101001"
                     "101011001"
                     "001001100"
                     "001100110"
                     "001100011"
                     "001011100"
                     "001011000"
                     "000111010"
                     "001110001"
                     "101100101"
                     "101001101"
                     "001100100"
                     "101110111"
                     "001101011"
                     "000101111"
                     "101111010"
                     "101111000"
                     "001010000"
                     "101001010"
                     "001100111"
                     "100110110"
                     "101101000"
                     "101100100"
                     "100111000"
                     "101100010"
                     "000110011"
                     "001110101"
                     "101111001"
                     "001011110"
                     "101010110"
                     "101111011"
                     "000111001"
                     "000110110"
                     "101111101"
                     "101000010"
```

```
"001101000"
"101010111"
"000101011"
"000110000"
"000101010"
"001001111"
"001010011"
"100110000"
"100101100"
"101101101"
"101110110"
"001000110"
"101100111"
"101101100"
"001000011"
"001001110"
"000110100"
"101010101"
"100110111"
"101011101"
"101010100"
"101110011"
"001101100"
"001110010"
"001000000"
"000100011"
"000101101"
"001101010"
"001111-11"
"100100010"
"00111-111"
"101110001"
"001000100"
"000110001"
"000111101"
"101110100"
"0011-1111"
"100101110"
"100110001"
"100110100"
"101011010"
"101100110"
"101000000"
"101000110"
"101011111"
"101110010"
"101010000"
"100111001"
"101100011"
"001001011"
"101010010"
"101011110"
"101001110"
"001010101"
```

```
"000100110"
"100100001"
"000100111"
"000100001"
"000101100"
"000101001"
"001110011"
"001010110"
"001111010"
"100101111"
"000111111"
"101101110"
"100101001"
"000100010"
"001011010"
"101101111"
"101101010"
"101000101"
"001010010"
"100110010"
"000110101"
"100100011"
"101001111"
"100110011"
"101001001"
"001010001"
"001000001"
"101000011"
"100111111"
"101010011"
"100111011"
"101100001"
"000110111"
"001001010"
"001101101"
"001100101"
"101110101"
"101001000"
"101011011"
"101011100"
"100101010"
"100100101"
"001011011"
"001110110"
"100101000"
"000100100"
"001010100"
"101000001"
"101110000"
"001011101"
"101001011"
"101100000"
"001000111"
"100110101"
```

```
        "001011001"
        "101010001"
        "101101001"
        "001110100"
        "101001100"
        "001111001"
        "101011000"
        "001-11111"
        "000101000"
        "000111100"
        "100111101"
        "000111011"
        "101000111"
        "001101110"
        "100111100"
        "001010111"
        "000111000"
        "101000100"
        "100111010"
        "100101101"
        "001000101"
        "001000010"
        "100111110"
        "000101110"
        "100101011"
        "001110000"
        "100100110"
        "100100100"
        "001100001"
        "100100111"
        "0011111-1"
        "001111110"
        "101101011"
        "001111000"
        "001100000"
        "001111100"
))
(output-plane '(
        "010000000000000000000000000"
        "000100000000000000000000000"
        "001000000000000000000000000"
        "000000000000000000000010010"
        "001010000000000000000000000"
        "000001001000101011001000000"
        "000100000010000001100001010"
        "000100000001000000011000001"
        "001110000000000000000000000"
        "000000000101010100001010000"
        "001100000000000000000000000"
        "001000100110100011001000000"
        "000000011010001110000001100"
        "000100000100000110001010010"
        "000010011010001000100001100"
        "000100110011000000100001100"
```

```
"000000010101000010111000010"
"000000010010000110100101010"
"000000000100100011110101000"
"000001100100001000110110010"
"000001010001000011100101001"
"000000110001010011010001010"
"001111000010001000011000100"
"000100100000001100011000110"
"000101001010000100001010101"
"001110001001000110001100000"
"000100000100110000001000101 1"
"000000101000001111010100001"
"001001101100100000001100011"
"000100000110001110000011010"
"000100000100001100101010010 1"
"001100101001100010110000100"
"000101010000101100000110101"
"000000010110100100101010010"
"000110001010001000110100011"
"000100011100000000011010101"
"000100100001101000011000011"
"000110000011000010010001111"
"000100100100100100100100101"
"000000000110010111100100101"
"001111010011000000110000001"
"000100000111010011010000010"
"000001101101100001100001100"
"000100111000010100010010110"
"000100000101001001101100010"
"001010101011010101010100000"
"000000011001000110111000011"
"000100000011000011011100010"
"000101100100100000101101010"
"000010011011000001010110100"
"000100110110100001000101100"
"001001010001100000101111100"
"001001110011000100011100001"
"001001001010100100011101100"
"000000100100011101111011000"
"001100111000100101010001001"
"000110100010000010111101100"
"000110110100101000101001010"
"000100010101011010101000100"
"000100000101010100000011111 0"
"001011110101010000100001101"
"000100011101011000011100000"
"000100010110101100101001000"
"001011100100110110101000010"
"000000011111010111000001100"
"001010001100101011001010110"
"000100111010001001100001110"
"000110000110100010011100110"
"000100101100010101010101001"
"000100111110000000110110011"
```

```
"0001000011100001010100000111"
"0000101011001001101010100010"
"0000010001100110000001111101"
"0010110101000001001111001110"
"0010000101110111011001110000"
"0010010111110010100001000011"
"0000001000111101101010100010"
"0000011010111110011010000000"
"0001111100001011011010000010"
"0000010110110101100000000101"
"0001000100000111011000111110"
"0010111010100101110010100001"
"0010011110011001000101010100"
"0010110000101110011100110001"
"0001000011001110101001100100"
"0000001110011001010000011011"
"0001101010110000100110011011"
"0001100111011100001000101100"
"0001100100011010010101010111"
"0001001100010100110000111111"
"0001000111101100011100000001"
"0001011010100000100111100111"
"0001010110100010110011011000"
"0001001010010010011110110010"
"0001000011110100001110100100"
"0001010000111101000100110111"
"0001011111111010000000100100"
"0001001000110001011100001111"
"0000000011111101011000110100"
"0001001111111101100000001000"
"0001001010101011101101000100"
"0001010001111111000101000100"
"0011010000100111010100111011"
"0010001011011111000001100111"
"0001111111000001011101100100"
"0010001101010011101110010011"
"0010000000111111100000011111"
"0010010110000101111110001100"
"0010010000111000101101101111"
"0000010010101001101110101011"
"0011010001110101101000100111"
"0000011001111011011001000011"
"0001101001100111100001011100"
"0000010011100110010111110011"
"0001000101000010011011111011"
"0001110001000101001111011011"
"0010000001111110000011110110"
"0000010110101010011001101111"
"0001000100101110011111010000"
"0001000110010101000000111111"
"0001010111001010110010111000"
"0011001100111001100110010100"
"0001100110011001100110011011"
"0010100100101110101110011011"
```

```
"0001111010011001000011001111"
"0001010001011101110000111101"
"0001100101011001000000111111"
"0001010100101111110011000100"
"0011001011101001011001000011"
"0010110110011011011001011000"
"0001011000011101011100101100"
"0001011011001110011010010100"
"0001001111011110100101001000"
"0001011110010001000100111110"
"0001001001100001001101011110"
"0000000111110011110011011000"
"0000000010101001011111110110"
"0000101100001111001011011100"
"0011100011011001110001100100"
"0001000001011110000111000110"
"0001010101010101010101010110"
"0001001011111001110110100000"
"0001001011011111011000001100"
"0001101111101110100100000110"
"0001110111000010011001111100"
"0000010111110110100000111110"
"0000010101110010111111001000"
"0001110010011111001001011100"
"0010000111110000111011011010"
"0011001111011000100011101010"
"0001011001110011111000110010"
"0001000100011010110011101110"
"0000011010001101011111111000"
"0001010011100110111111100000"
"0001001001111001101001110100"
"0010111110101010010111100100"
"0001100011011101011010110100"
"0000010101011101110111010010"
"0001010000011100111111101010"
"0001000110101010011010011110"
"0011110011101101000110101000"
"0001010011000011101010111110"
"0000111000111001101100111010"
"0001001101001011111101100100"
"0000011100100010110111011010"
"0010001111000110111011110100"
"0010101111010001010110111010"
"0001011100101101010110011100"
"0000001101110011011111001100"
"0001010101111011101010001100"
"0000001101010100010111111110"
"0001011101011110100101110100"
"0011010011000011011111001110"
"0010101001010100111111110110"
"0001010111110011101010100110"
"0001011111000100110111010110"
"0001101011111100000110011110"
"0010111011111101010001101100"
```

```
                                "001011011111010011011101010"
                                "000101101111111010100111010 0"
                                "001001100101110111011100111"
                                "000110111001101011101101110"
                                "001110111101110111010100001"
                                "000111010101110101111110101"
                                "000111100010101101111101111"
                                "001101111011011010101011011"
                                "000111001111110001111101101"
                                "000011110011111011011100101"
                                "000001110111111101111111000"
                                "000100010111111111110010111"
                                "000011011111011110111101011"
                                "001101110110110011110101111"
                                "000011101111110111110111111"
                        ))
                        (minterm 193)
        ))
)
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
(net PHI (NETTYPE CLOCK ))
(net PHI1 (NETTYPE CLOCK ))
(net PHI2 (NETTYPE CLOCK ))
;
(instance ROM (
        (IN ADDRESS (width 9))
        (OUT PLAOUT (width 27))
        (CLOCK PHI)
        (Vdd Vdd)
        (GND GND)
))
;
(instance LATCH (
        (IN PLAOUT (width 27))
        (OUT DATA (width 27))
        (PHI1 PHI1)
        (PHI2 PHI2)
        (LD Vdd)
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal ADDRESS (DIRECTION INPUT)) ADDRESS (width 9))
        ((terminal DATA (DIRECTION OUTPUT)) DATA (width 27))
        ((terminal PHI (TERMTYPE CLOCK) (DIRECTION INPUT)) PHI )
        ((terminal PHI1 (TERMTYPE CLOCK) (DIRECTION INPUT)) PHI1 )
        ((terminal PHI2 (TERMTYPE CLOCK) (DIRECTION INPUT)) PHI2 )
        ((terminal Vdd (TERMTYPE SUPPLY) (TERM_EDGE TOP)) Vdd)
        ((terminal GND (TERMTYPE GROUND)(TERM_EDGE BOTTOM)) GND)
))
;
```

(end–sdl)

A.1.27 Grom.sdl

The lookup table used in the interpolation cell is generated from a file of data value. It has been arranged so that the most significant bit of the input to this lookup table selects the function being calculated. If it is a zero, then the data for the square root calculation is provided. If it is a one, then the data for the inverse square root is provided. This file contains the values for $g(x_i) = f(x_{i+1}) - f(x_i)$.

```
; Automatically Generated PLA cell file to be used as ROM.  DO NOT EDIT!
; Genereated Monday, 11 April 1994, 06:37:36 AM.
;
(parent–cell Grom)
;
(layout–generator Flint b)
;
(subcells
        (latch LATCH ((width 19)))
        (pla ROM (
                        (inwidth 9)
                        (outwidth 19)
                        (input–plane '(
                                "-01000100"
                                "10-100001"
                                "001111101"
                                "001001100"
                                "-01001010"
                                "-01010-11"
                                "101001111"
                                "001010101"
                                "001111001"
                                "10111111-"
                                "001000001"
                                "101110100"
                                "101110010"
                                "101100111"
                                "10111100-"
                                "101000101"
                                "001001000"
                                "001110110"
                                "001111--1"
                                "001010000"
                                "00100-10-"
                                "001110000"
                                "001011111"
                                "100100111"
                                "100100011"
```

```
"010000000"
"001111110"
"101000001"
"001110011"
"001101111"
"0010101-1"
"000110110"
"000111000"
"00111-011"
"001000101"
"100100001"
"100100000"
"101-10010"
"001000011"
"1011-0100"
"101000100"
"001000000"
"001101100"
"001111100"
"001100100"
"001001010"
"001101101"
"001001111"
"101010111"
"101010101"
"001100110"
"100110101"
"001011001"
"001011100"
"0010-0001"
"001100011"
"110000000"
"001011000"
"100110011"
"000101010"
"001011010"
"001010100"
"100101001"
"100100010"
"001001110"
"001000111"
"101100010"
"001110101"
"001110010"
"100111110"
"001101110"
"100111111"
"000101001"
"101011000"
"001110111"
"1010100-1"
"101000000"
"100101100"
"001001011"
```

```
"101000010"
"100101110"
"000110111"
"001110100"
"100110110"
"001100010"
"001100001"
"100111000"
"001010010"
"000110011"
"101011010"
"100111100"
"001011110"
"100111010"
"100101000"
"001101010"
"001000010"
"001111010"
"000100110"
"001100111"
"101001011"
"000100011"
"000101011"
"101010100"
"101011111"
"000100111"
"001001001"
"000101100"
"000111101"
"000101101"
"001011101"
"001111000"
"000111110"
"001101011"
"100111011"
"101011011"
"100110000"
"101010110"
"000101111"
"101011110"
"000111100"
"001011011"
"000110010"
"000101110"
"001010110"
"001010011"
"100110100"
"100100101"
"101-11001"
"100101010"
"101110110"
"101111000"
"101101110"
"101000111"
```

```
"10110011-"
"1010011-1"
"101000110"
"101101001"
"000111010"
"101100011"
"101111100"
"000101000"
"000100010"
"000100101"
"100100100"
"001100000"
"000110001"
"001101000"
"000110000"
"0010001-0"
"100111001"
"100101111"
"100101011"
"101111010"
"101101010"
"101101111"
"101001001"
"000100001"
"10111-111"
"101101000"
"1011111-1"
"000100000"
"101110000"
"101111011"
"000111111"
"101101011"
"101100101"
"101110001"
"001100101"
"001110001"
"101011100"
"101000011"
"001101001"
"101011101"
"000110101"
"1010010-0"
"100110111"
"100100110"
"000100100"
"101101101"
"101110011"
"101111110"
"000111001"
"101001100"
"100110010"
"100101101"
"000110100"
"100110001"
```

```
        "100111101"
        "101010000"
        "101001110"
        "10110000-"
        "000111011"
        "101110101"
        "101101100"
))
(output-plane '(
        "0000000100000000000"
        "0000000001100000000"
        "0000000010000010010"
        "0000000010001001010"
        "0000000110000000010"
        "0000000100100000000"
        "0000000101000110000"
        "0000001000000110100"
        "0000000110010000100"
        "0000000000100000101"
        "0000100101000011000"
        "0000010010010010010"
        "0000110000000010101"
        "0000000001010010101"
        "0000101000010000100"
        "1101100000000010001"
        "0101010100001010000"
        "0100001010000100001"
        "0100000000100000001"
        "0101000010110100000"
        "0101001001000000000"
        "0100010001000100010"
        "0100101000011000001"
        "1010001010101000001"
        "1001001001110001000"
        "0011111111100000001"
        "0100000001100000111"
        "1101010001001010000"
        "0100001001011000001"
        "0100010010010010100"
        "0100110001001000010"
        "0110001000010101001"
        "0110000001010100101"
        "0000000100000111110"
        "0000010010011010111"
        "1000100000011011110"
        "1000001011101100100"
        "1110000100010100000"
        "0101100000100001111"
        "1110100100000000100"
        "1101011000100001001"
        "0101101000101000101"
        "0100010110000011100"
        "0100000011100100110"
        "0100100000111010010"
```

```
"0101001001100011100"
"0100010100110010001"
"0101000100110101100"
"1110001010010000110"
"1110001010110001000"
"0100011110000101000"
"1100010011001001001"
"0100110010001001101"
"0100101101001001010"
"0101000000110100110"
"0100100010010111000"
"1111000000010111111"
"0100110011111000000"
"1100000101001101100"
"0110111100010010000"
"0100110000011101001"
"0100111011000101000"
"1010100101010010110"
"1000110110100100101"
"0101000110111001100"
"0101010110100001110"
"1110100001001100010"
"0100001011001100011"
"0100001110101011000"
"1101000100011010110"
"0100010011100001110"
"1101001000110101010"
"0111000001100110101"
"1110010000101011010"
"0100001000111100110"
"1110000010000001111"
"1101001101000100110"
"1011000111110000101"
"0101001101010101001"
"1101010101000101100"
"1011011011101100000"
"0110000100110001111"
"0100001100010101101"
"1100011001101000011"
"0100100011110101000"
"0100100101010100101"
"1100100101110000011"
"0100111110111000000"
"0110010011100110001"
"1110010100010110001"
"1100111011000010100"
"0100101001111100010"
"1100110000110110100"
"1010011000010111110"
"0100011000101001111"
"0101100011001010111"
"0100000101101011110"
"0111010010110010110"
"0100011100101100011"
```

```
"11011100101011110000"
"01111001100001111100"
"01101101110010010 10"
"11100010001010 11011"
"11100111001011 00001"
"0111001100110110001"
"0101010001110101011"
"0110110010001011101"
"0101110001010100111"
"0110101101011000101"
"0100101011100001111"
"0100000111110110010"
"0101101110010111000"
"0100010111010110001"
"1100110110000011011"
"1110010110000110110"
"1011101101100101010"
"1110001100110010110"
"0110100100001111110"
"1110011011000111000"
"0101110100010111100"
"0100101110110010011"
"0110010111100100101"
"0110101000101111100"
"0100110111011010100"
"0100111000011101011"
"1100001100010101111"
"1001101100010110111"
"1110010010100010011"
"1010110001011101011"
"1110111000001001101"
"1110010001101011001"
"1110110000001101011"
"1101100110101100101"
"1110100110101100000"
"1101111000001001110"
"1101100011011011001"
"1110101010011110000"
"0101111010101011100"
"1110100010100111100"
"1110111010100010000"
"0111000111000111101"
"0111101101000111010"
"0111011000111110100"
"1001011011101001111"
"0100100110110101101"
"0110011011101010110"
"0100011011010100111"
"0110011111111001000"
"0101011000111100101"
"1100101011011011001"
"1011100100110111110"
"1010111100111011100"
"1110111011101001000"
```

```
                                        "11101010111010110000"
                                        "11101100010100011111"
                                        "11011011001110100111"
                                        "01111101000110101111"
                                        "11101110010000110101"
                                        "11101010010011110101"
                                        "11101111100001010000"
                                        "01111111000000111111"
                                        "11101100100101001111"
                                        "11101111000111100000"
                                        "01011010110111011100"
                                        "11101011001101100011"
                                        "11101001010101110100"
                                        "11101100110101101000"
                                        "01000111110111110000"
                                        "01000011111101110011"
                                        "11100101111101000111"
                                        "11010110001101111011"
                                        "01000110011111101100"
                                        "11100110010111110011"
                                        "01100010111111101100"
                                        "11011010011101110000"
                                        "11000111111101010011"
                                        "10011110111111110000"
                                        "01110111110110100111"
                                        "11101011110001110100"
                                        "11101101010101010111"
                                        "11101111100101100100"
                                        "01011111011111010100"
                                        "11011101010111101011"
                                        "10111111011011100111"
                                        "10110100011111111111"
                                        "01100011111011101111"
                                        "10111101011101101111"
                                        "11001111111101001100"
                                        "11011111111010110100"
                                        "11011110101011110111"
                                        "11100111100011101011"
                                        "01011101110111101111"
                                        "11101101110011101111"
                                        "11101011011111111000"
                                ) )
                                (minterm 194)
                ) )
)
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
(net PHI (NETTYPE CLOCK ))
(net PHI1 (NETTYPE CLOCK ))
(net PHI2 (NETTYPE CLOCK ))
;
(instance ROM (
        (IN ADDRESS (width 9))
        (OUT PLAOUT (width 19))
        (CLOCK PHI)
        (Vdd Vdd)
        (GND GND)
))
;
(instance LATCH (
        (IN PLAOUT (width 19))
        (OUT DATA (width 19))
        (PHI1 PHI1)
        (PHI2 PHI2)
        (LD Vdd)
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal ADDRESS (DIRECTION INPUT)) ADDRESS (width 9))
        ((terminal DATA (DIRECTION OUTPUT)) DATA (width 19))
        ((terminal PHI (TERMTYPE CLOCK) (DIRECTION INPUT)) PHI )
        ((terminal PHI1 (TERMTYPE CLOCK) (DIRECTION INPUT)) PHI1 )
        ((terminal PHI2 (TERMTYPE CLOCK) (DIRECTION INPUT)) PHI2 )
        ((terminal Vdd (TERMTYPE SUPPLY) (TERM_EDGE TOP)) Vdd)
        ((terminal GND (TERMTYPE GROUND)(TERM_EDGE BOTTOM)) GND)
))
;
(end-sdl)
```

A.1.28 inc.sdl

In the interpolation cell, to access the next point in the lookup tables, the address to index the tables in incremented by one. This is used to have access to both δ_{m_0} and δ_{m_1}. It is essential an adder that just adds the `INC` signal to the address presented to the lookup tables in `deltarom.sdl`, `From.sdl`, and `Grom.sdl`.

```
(parent-cell inc)
;
(parameters width)
;
(layout-generator Flint b)
;
(subcells (inclogic LOGIC ((N width))))
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
;
(instance LOGIC (
        (IN IN (width width))
        (OUT OUT (width width))
        (CARRYIN CARRYOUT (width (– width 1)) (term–base 1))
        (CARRYIN INC (term–base 0))
        (CARRYOUT CARRYOUT (width (– width 1)) (term–base 0))
        (CARRYOUT CARRY (term–base (– width 1)))
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal IN (DIRECTION INPUT)) IN (width width))
        ((terminal OUT (DIRECTION OUTPUT)) OUT (width width))
        ((terminal INC (DIRECTION INPUT)) INC)
        ((terminal CARRY (DIRECTION OUTPUT)) CARRY)
        ((terminal CARRYOUT (DIRECTION OUTPUT))
                CARRYOUT (width (– width 1)))
        ((terminal Vdd (TERMTYPE SUPPLY)
                (TERM_EDGE TOP)) Vdd)
        ((terminal GND (TERMTYPE GROUND)
                (TERM_EDGE BOTTOM)) GND)
))
;
(end–sdl)
```

A.1.29 incdpp.sdl

This file contains the actual logic used to implement the simple adder needed for the `inc.sdl` cell. The add is performed by way of an XOR and an ADD cell. This is possible because the input to any bit of the adder is only the address value being incremented and a carry in signal, because the cell only performs addition by one.

```
; incremnetor – increments input by one when inc=1
; ccn 4 october 1993
(parent–cell inclogic)
;
(parameters N)
;
(structure–processor dpp)
(layout–generator Flint a)
;
(subcells (xor2 XOR ((N N)))
        (and2 AND ((N N)))
)
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
;
(instance XOR (
        (A IN)
        (B CARRYIN)
        (XOR OUT)
        (Vdd Vdd)
        (GND GND)))
;
(instance AND (
        (A IN)
        (B CARRYIN)
        (AND2 CARRYOUT)
        (Vdd! Vdd)
        (GND! GND)
))
;
(instance parent (
        ((terminal IN (DIRECTION INPUT)) IN)
        ((terminal OUT (DIRECTION OUTPUT)) OUT)
        ((terminal CARRYIN (DIRECTION INPUT)) CARRYIN)
        ((terminal CARRYOUT (DIRECTION OUTPUT)) CARRYOUT)
        ((terminal Vdd (TERMTYPE SUPPLY) ) Vdd)
        ((terminal GND (TERMTYPE GROUND) ) GND)
))
;
(end-sdl)
```

A.1.30 interp.sdl

This is the top level of the interpolation cell discussed in chapter 5. The data is accepted at the `IN` input, where it is normalized. Depending on the value of the `FUNC` signal, either the square root or the inverse square root will be performed when the `GO` signal is taken high. When the cell is calculating, the `BUSY` signal is held high. Upon completion, the mantissa of the output is available at `OUT`, and the exponent is available at `SHIFTS`.

```
;
; top level for cubic interpolation cell using Comrie's throwback
;
; ccn 4 october 1993
; new version using muxadder mux/addsub combination to reduce routing load
; Flint couldn't handle the routing
; ccn 8 oct 1993
; Added it back in.  Probelms are deeper in Flint, but unrelated to size.
; ccn 2 November 1993
; added support for 2 functions: (func=0, sqrt) ( func=1, inverse)
; ccn 9 April 1994
;
(parent-cell interp)
```

```
;
(parameters (width 24)
          (M 7)
)
;
(structure-processor SIVcheck)
(layout-generator Flint v w b)
;
(subcells
          (3to1mux MULTAMUX ((N (+ (- width M) 2))))
          (3to1mux MULTBMUX ((N (+ (- width M) 2))))
          (3to1mux ADDERBMUX ((N (+ width 3 ))))
          (2to1mux ADDERAMUX ((N (+ width 3 ))))
          (normalizer NORMALIZER ((width width)))
          (inc INC ((width M)))
          (From FROM)
          (deltarom DELTAROM)
          (Grom GROM)
          (register TEMP ((width (- (+ (- width M) 2) 3))) )
          (register PQ ((width (- (+ (- width M) 2) 2))) )
          (register ACCUM ((width (+ width 3))))
          (addsub ADDER ((width (+ width 3))))
          (multiplier MULT ((width (+ (- width M) 2))))
          (myclock CLKGEN)
          (interpcntl FSM)
)
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE SUPPLY))
(net PHI1 (NETTYPE CLOCK))
(net PHI2 (NETTYPE CLOCK))
(net NOTPHI1 (NETTYPE CLOCK))
(net NOTPHI2 (NETTYPE CLOCK))
(net PHI (NETTYPE CLOCK))
;
(instance NORMALIZER (
          (IN IN (width width))
          (OUT NORMOUT (width width))
          (SHIFTS SHIFTS (width (+ (ceiling (log2 width)) 1)))
          (FUNC FUNC)
          (Vdd Vdd)
          (GND GND)
))
;
(instance INC (
          (IN NORMOUT (width M) (net-base (- (- width M) 0)))
          (OUT ROMADD (width M))
          (CARRY ROMADD (net-base M))
          (INC INC)
          (Vdd Vdd)
          (GND GND)
))
;
(instance FROM (
        (ADDRESS ROMADD (width (+ M 1)))
        (ADDRESS FUNC (term-base (+ M 1)))
        (DATA FROMOUT (width (+ width 3)))
        (PHI PHI)
        (PHI1 PHI1)
        (PHI2 PHI2)
        (Vdd Vdd)
        (GND GND)
))
;
(instance DELTAROM (
        (ADDRESS ROMADD (width (+ M 1)))
        (ADDRESS FUNC (term-base (+ M 1)))
        (DATA DELTAROMOUT (width (+ (- width M) 2)))
        (PHI PHI)
        (PHI1 PHI1)
        (PHI2 PHI2)
        (Vdd Vdd)
        (GND GND)
))
;
(instance GROM (
        (ADDRESS ROMADD (width (+ M 1)))
        (ADDRESS FUNC (term-base (+ M 1)))
        (DATA GROMOUT (width (+ (- width M) 2)))
        (PHI PHI)
        (PHI1 PHI1)
        (PHI2 PHI2)
        (Vdd Vdd)
        (GND GND)
))
;
(instance MULTAMUX (
        ; -2 <= A < 2, so top 2 bits > 1, rest < 1
        (A SUM (width (+ (- width M) 2)) (net-base (- (+ width 3) (+ (+
(- width M) 2) 1)) ) )
        (B TEMPOUT (width (- (+ (- width M) 2) 3) ) )
        (B GND (merge (- (+ (- width M) 2) 3) (- (+ (- width M) 2) 1)))
 ; add leading zeros since temp < .5, and sign extend.
        (C GROMOUT (width (+ (- width M) 2)))
        (SEL MULTASEL (width 2))
        (OUT MULTAIN (width (+ (- width M) 2)))
        (Vdd Vdd)
        (GND GND)
))
;
(instance MULTBMUX (
        ; -1 <= B < 1, so only MSB > 1
        (A NORMOUT (width (- width M)) (net-base 0) (term-base 1))
        (A GND (term-base 0)) ; pad lsb with zero
        (A GND (term-base (- (+ (- width M) 2) 1))) ; sign extension, just MSB
        (B DELTAROMOUT (width (+ (- width M) 2)))
        (C PQOUT (width (- (+ (- width M) 2) 2)))
        (C GND (merge (- (+ (- width M) 2) 2) (- (+ (- width M) 2) 1) ) )
 ; add leading zeros since pq <= .25, and sign extend
        (SEL MULTBSEL (width 2))
        (OUT MULTBIN (width (+ (- width M) 2)))
        (Vdd Vdd)
        (GND GND)
))
;
(instance MULT (
        (A MULTAIN (width (+ (- width M) 2 )))
        (B MULTBIN (width (+ (- width M) 2 )))
        (PROD PROD (width (- (+ ( - width M) 2) 1)) (term-base (- (- (* 2
(+ (- width M) 2)) 1) (+ (- width M) 2)) ) )
        (PROD PROD_MSB (term-base (- (- ( * 2 (+ (- width M) 2)) 1) 1) ) )
        (Vdd Vdd)
        (GND GND)
))
;
(instance TEMP (
        (D PROD (width (- (+ (- width M) 2) 3)) )
; don't need to sign extend, becuase temp is always positive
        (Q TEMPOUT (width (- (+ (- width M) 2) 3) ) )
        (LOAD LOADTEMP)
        (PHI1 PHI1)
        (PHI1INV PHI1INV)
        (PHI2 PHI2)
        (PHI2INV PHI2INV)
        (Vdd Vdd)
        (GND GND)
))
;
(instance PQ (
        (D PROD (width (- (+ (- width M) 2) 3) ) (term-base 1))
        (D GND (term-base 0))
 ; needed because LSB of PQ = 2^-18, ok because delta < 2 ^(-(M+1))
, so we'll drop this bit anyway.
        (Q PQOUT (width (- (+ (- width M) 2) 2) ) )
        (LOAD LOADPQ)
        (PHI1 PHI1)
        (PHI1INV PHI1INV)
        (PHI2 PHI2)
        (PHI2INV PHI2INV)
        (Vdd Vdd)
        (GND GND)
))
;
(instance ADDERAMUX (
        (A NORMOUT (width (- width M)) (net-base 0) (term-base   M ))
        (A GND (merge width (+ width 2))) ; sign extension
        (A GND (merge 0 (- M 1))) ; pad end with 0
        (B PROD (width (- (+ (- width M) 2) 1)) (term-base 0) (net-base 0))
        (B PROD_MSB (merge (- (+ (- width M) 2) 1) (+ width 2)))
        (OUT ADDERAIN (width (+ width 3) ))
        (SEL ADDERASEL)
        (Vdd Vdd)
        (GND GND)
))
;
(instance ADDERBMUX (
        (A ACCUMOUT (width (+ width 3)))
        (B FROMOUT (width (+ width 3)) (term-base 0))
        (C CONSTSEL (width 2) (term-base width))
        (C GND (merge 0 (- width 1)))
        (C GND (term-base (+ width 2))) ; sign extension
        (OUT ADDERBIN (width (+ width 3)))
        (SEL ADDERBSEL (width 2))
        (Vdd Vdd)
        (GND GND)
))
;
(instance ADDER (
        (A ADDERAIN (width (+ width 3 )))
        (B ADDERBIN (width (+ width 3 )))
        (SUB ADDERSUB)
        (SUM SUM (width (+ width 3)))
        (Vdd Vdd)
        (GND GND)
))
;
(instance ACCUM (
        (D SUM (width (+ width 3)))
        (Q ACCUMOUT (width (+ width 3)))
        (LOAD LOADACCUM)
        (PHI1 PHI1)
        (PHI1INV PHI1INV)
        (PHI2 PHI2)
        (PHI2INV PHI2INV)
        (Vdd Vdd)
        (GND GND)
))
;
(instance CLKGEN (
        (CLK PHI)
        (PHI1 PHI1)
        (PHI2 PHI2)
        (PHI1INV PHI1INV)
        (PHI2INV PHI2INV)
        (Vdd Vdd)
        (GND GND)
))
;
(instance FSM (
        (GO GO)
        (BUSY BUSY)
        (INC INC)
        (RESET RESET)
        (MULTASEL MULTASEL (width 2))
        (MULTBSEL MULTBSEL (width 2))
        (ADDERASEL ADDERASEL)
        (ADDERBSEL ADDERBSEL (width 2))
        (ADDERSUB ADDERSUB)
        (LOADACCUM LOADACCUM)
        (LOADPQ LOADPQ)
        (LOADTEMP LOADTEMP)
        (CONSTSEL CONSTSEL (width 2))
        (PHI1 PHI1)
        (PHI2 PHI2)
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal IN (DIRECTION INPUT)) IN (width width))
        ((terminal GO (DIRECTION INPUT)) GO)
        ((terminal RESET (DIRECTION INPUT)) RESET)
        ((terminal BUSY (DIRECTION OUTPUT)) BUSY)
        ((terminal OUT (DIRECTION OUTPUT)) ACCUMOUT (width (+ width 2)))
        ((terminal SHIFTS (DIRECTION OUTPUT)) SHIFTS (width (+ (ceiling
(log2 width)) 1)))
        ((terminal FUNC (DIRECTION INPUT)) FUNC)
        ((terminal Vdd (TERMTYPE SUPPLY) (TERM_EDGE TOP)) Vdd)
        ((terminal GND (TERMTYPE GROUND) (TERM_EDGE BOTTOM)) GND)
        ((terminal PHI (TERMTYPE CLOCK)) PHI)
;
(end-sdl)
```

A.1.31 interpcntl.sdl

This is a simple finite state machine that creates a PLA grid of instructions based on the contents of the `interpcntl.bds` behavioral description language file. It is designed to control the operation of the fast interpolation cell.

```
(parent-cell interpcntl)
;
(layout-generator Flint b)
;
(subcells
        (fsm_bdsyn FSM ((inwidth 5) (outwidth 18)
(bdsyn "~/sdl/interp/interpcntl/interpcntl.bds")))
)
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
(net PHI1 (NETTYPE CLOCK))
(net PHI2 (NETTYPE CLOCK))
;
(instance FSM (
        (IN STATE (term-base 0) (width 3))
        (IN GO (term-base 3))
        (IN RESET (term-base 4))
        (OUT STATE (term-base 0) (width 3))
        (OUT BUSY (term-base 3))
        (OUT INC (term-base 4))
        (OUT MULTASEL (term-base 5) (width 2))
        (OUT MULTBSEL (term-base 7) (width 2))
        (OUT ADDERASEL (term-base 9))
        (OUT ADDERBSEL (term-base 10) (width 2))
        (OUT ADDERSUB (term-base 12))
        (OUT LOADACCUM (term-base 13))
        (OUT LOADPQ (term-base 14))
        (OUT LOADTEMP (term-base 15))
        (OUT CONSTSEL (term-base 16) (width 2))
        (PHI1 PHI1)
        (PHI2 PHI2)
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal GO (DIRECTION INPUT)) GO)
        ((terminal RESET (DIRECTION INPUT)) RESET)
        ((terminal STATE (DIRECTION OUTPUT)) STATE (width 3))
        ((terminal BUSY (DIRECTION OUTPUT)) BUSY)
        ((terminal INC (DIRECTION OUTPUT)) INC)
        ((terminal MULTASEL (DIRECTION OUTPUT)) MULTASEL (width 2))
        ((terminal MULTBSEL (DIRECTION OUTPUT)) MULTBSEL (width 2))
        ((terminal ADDERASEL (DIRECTION OUTPUT)) ADDERASEL)
        ((terminal ADDERBSEL (DIRECTION OUTPUT)) ADDERBSEL (width 2))
        ((terminal ADDERSUB (DIRECTION OUTPUT)) ADDERSUB)
        ((terminal LOADACCUM (DIRECTION OUTPUT)) LOADACCUM)
        ((terminal LOADPQ (DIRECTION OUTPUT)) LOADPQ)
        ((terminal LOADTEMP (DIRECTION OUTPUT)) LOADTEMP)
        ((terminal CONSTSEL (DIRECTION OUTPUT)) CONSTSEL (width 2))
        ((terminal PHI1 (TERMTYPE CLOCK)) PHI1)
        ((terminal PHI2 (TERMTYPE CLOCK)) PHI2)
        ((terminal Vdd (TERMTYPE SUPPLY) (TERM_EDGE TOP)) Vdd)
        ((terminal GND (TERMTYPE GROUND) (TERM_EDGE BOTTOM)) GND)
))
;
(end-sdl)
```

A.1.32 multiplier.sdl

This cell is a real multiplier for use in the fast interpolation cell. It performs a single cycle multiply of `A` and `B`.

```
(parent-cell multiplier)
```

```
;
(parameters width)
;
(layout-generator Flint b)
;
(subcells
        (mult MULT ((m width) (n width) (s (list-length (make-carry-select
(- (* 2 width) 3)))) (csindex (make-carry-select (- (* 2 width) 3)))))
)
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
;
(instance MULT (
        (X A (width width))
        (Y B (width width))
        (P PROD (width (- (* 2 width) 1)))
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal A (DIRECTION INPUT)) A (width width))
        ((terminal B (DIRECTION INPUT)) B (width width))
        ((terminal PROD (DIRECTION OUTPUT)) PROD (width (- (* 2 width) 1)))
        ((terminal Vdd (TERMTYPE SUPPLY) (TERM_EDGE TOP)) Vdd)
        ((terminal GND (TERMTYPE GROUND) (TERM_EDGE BOTTOM)) GND)
))
;
(end-sdl)
```

A.1.33 negator.sdl

This cell performs an XOR of each bit of the input IN with the value of CNTL. This is used for taking the complement of the B input to the cmult.sdl complex multiplier cell.

```
(parent-cell negator)
;
(parameters N)
;
(structure-processor dpp)
(layout-generator Flint a)
;
(subcells (invpass INVPASS ((N N))))
;
(instance INVPASS (
                (IN IN)
                (OUT OUT)
                (CNTL CNTL)
                (Vdd Vdd)
                (GND GND)
))
;
(instance parent (
                ((terminal IN (DIRECTION INPUT)) IN)
                ((terminal OUT (DIRECTION OUTPUT)) OUT)
                ((terminal CNTL (DIRECTION INPUT)) CNTL)
                ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
                ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.34 normalizer.sdl

The input to the fast interpolation cell is normalized at the input. The cell brings together the priority encode used to determine the number of shifts required and the barrel shifter to actually implement them.

```
(parent-cell normalizer)
;
(parameters width )
;
(structure-processor SIVcheck)
(layout-generator Flint b)
;
(subcells (normalizerlogic LOGIC ((width width)))
                (normalizerdpp BSHIFT ((N width)))
                (normalizerbuffer BUFFER ((width width)))
)
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
;
(instance LOGIC (
        (bits IN (width (- width 0)))
        (shifts SHIFTS (width (ceiling (log2 width))))
        (shiftcntl shiftcntl (width width))
        (Vdd Vdd)
        (GND GND)
))
;
(instance BSHIFT (
        (DATAIN IN (width width))
        (SHIFTCNTL shiftcntl (width width))
        (DATAOUT OUT(width width))
        (ZERO GND (merge 0 (- width 1)))
))
;
(instance BUFFER (
                    (IN FUNC)
                    (OUT SHIFTS (net-base (ceiling (log2 width))))
                    (Vdd Vdd)
                    (GND GND)
))
;
(instance parent (
        ((terminal IN (DIRECTION INPUT)) IN (width (- width 0)))
        ((terminal SHIFTS (DIRECTION OUTPUT)) SHIFTS (width (+ (ceiling
(log2 width)) 1)))
        ((terminal FUNC (DIRECTION INPUT)) FUNC)
        ((terminal OUT (DIRECTION OUTPUT)) OUT (width (- width 0)))
        ((terminal Vdd (TERMTYPE SUPPLY) (TERM_EDGE TOP)) Vdd)
        ((terminal GND (TERMTYPE GROUND) (TERM_EDGE BOTTOM)) GND)
))
(end-sdl)
```

A.1.35 normalizerbuffer.sdl

This is just a buffer from the standard cell library. It is used to assign a different name to the `FUNC` input for use in the output. LAGER does not permit having two names for the same net, so this cell works around this limitation.

```
(parent-cell normalizerbuffer)
;
(layout-generator Stdcell)
;
(subcells          (buff101 BUFFER))
;
(instance BUFFER (
        (A1 IN)
        (O OUT)
))
;
(instance parent (
        ((terminal IN (DIRECTION INPUT)) IN)
        ((terminal OUT (DIRECTION OUTPUT)) OUT)
        ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
        ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.36 normalizerdpp.sdl

This is the actual barrel shifter used to perform the shift up required for normalization at the input to the interpolation cell. The `ZERO` input is tied to `GND` one level up in the hierarchy because the datapath library does not permit connections between the data inputs and the power supply rails. `S` the shift control signal generated by the priority encoder, `normalizerlogic.sdl`.

```
(parent-cell normalizerdpp)
;
(parameters N)
;
(structure-processor dpp)
(layout-generator Flint b)

;
(subcells (bshift BSHIFT ((N N) )))
;
(instance BSHIFT (
        (A DATAIN )
        (B ZERO )
        (S SHIFTCNTL)
        (O DATAOUT )
))
;
(instance parent (
        ((terminal DATAIN (DIRECTION INPUT)) DATAIN )
        ((terminal DATAOUT (DIRECTION OUTPUT)) DATAOUT)
        ((terminal SHIFTCNTL (DIRECTION INPUT)) SHIFTCNTL)
        ((terminal ZERO (DIRECTION INPUT)) ZERO )
))
;
(end-sdl)
```

A.1.37 normalizerlogic.sdl

The priority encoder used to normalize the input to the interpolation cell is implemented using a behavioral description file `prienc.bds` that is translated into standard logic cells. This cell generates the exponent for the interpolator output and the control signals for the `normalizerdpp.sdl` cell.

```
(parent-cell normalizerdpp)
;
(parameters N)
```

```
;
(structure-processor dpp)
(layout-generator Flint b)

;
(subcells (bshift BSHIFT ((N N) )))
;
(instance BSHIFT (
        (A DATAIN )
        (B ZERO )
        (S SHIFTCNTL)
        (O DATAOUT )
))
;
(instance parent (
        ((terminal DATAIN (DIRECTION INPUT)) DATAIN )
        ((terminal DATAOUT (DIRECTION OUTPUT)) DATAOUT)
        ((terminal SHIFTCNTL (DIRECTION INPUT)) SHIFTCNTL)
        ((terminal ZERO (DIRECTION INPUT)) ZERO )
))
;
(end-sdl)
```

A.1.38 offdiag2.sdl

This file is the core of the off-diagonal processor elements. It takes as parameters the number of registers to be tiled into the register file `regfile2p.sdl`, and the width to be used for all of the subcells. The data comes in through 4 bit wide serial links. The core also has connections to determine whether or not the processor is on an edge of the array. This cell implements a complex multiplier `bus2cmult.sdl`, a complex adder `bus2adder`, a register file `regfile2p.sdl`, and several semi-serial input output registers `busserio.sdl`.

```
(parent-cell offdiag2)
;
(parameters width numOfRegs (ioPerChannel 4) (ioChannels 6))
; Note: Width must be even, and is total number of real+imag bits.
;
(layout-generator Flint b v)
;
(subcells
;
; note: floor is used for in/outwidth to force an error if width was odd.
; if width was odd, floor will round down, so that (* 2 (floor (/ width 2)))
; will be less than width, and errors will result due to unknown terminals.
        (bus2adder ADDER ((inwidth (floor (/ width 2))) (outwidth (floor
(/ width 2)))))
          (bus2cmult MULT  ((inwidth (floor (/ width 2))) (outwidth (floor
(/ width 2)))))
          (regfile2p REGFILE ((numOfRegs numOfRegs) (width width)))
          (myclock CLKGEN)
          (offdiagctl FSM)
)
;
(dotimes (i ioChannels)
          (subcells
           (busserio BUSIO ((ioLines ioPerChannel) (busWidth width)))
          )
)


;
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE Vdd))
(net PHI (NETTYPE CLOCK))
;
(instance REGFILE (
          (ABUS ABUS (width width))
          (BBUS BBUS (width width))
          (EBUS EBUS (width width))
          (AOUTEN AOUTEN (width numOfRegs))
          (BOUTEN BOUTEN (width numOfRegs))
          (LOAD LOAD (width numOfRegs))
          (SCAN GND (merge 0 (− numOfRegs 1)))
          (PHI PHI)
          (Vdd Vdd)
          (GND GND)
))
;
(instance ADDER (
          (ABUS ABUS (width width))
          (BBUS BBUS (width width))
          (EBUS EBUS (width width))
          (OUTEN ADDEROUTEN)
          (FUNC ADDERFUNC)
          (REALADD GND)
          (Vdd Vdd)
          (GND GND)
))
;
(instance MULT (
          (ABUS ABUS (width width))
          (BBUS BBUS (width width))
          (EBUS EBUS (width width))
          (COMPB GND)
          (COMPLEXOUT Vdd)
          (OUTEN MULTOUTEN)
          (REALA GND)
          (Vdd Vdd)
          (GND GND)
```

```
))
;
(instance BUSIO (
                    (EBUS EBUS (width width))
                    (ABUS ABUS (width width))
                    (INPUTA INPUTA (net-base (* i ioPerChannel))
(width ioPerChannel))
                    (INPUTB INPUTB (net-base (* i ioPerChannel))
(width ioPerChannel))
                    (OUTPUT OUTPUT (net-base (* i ioPerChannel))
(width ioPerChannel))
                    (READY IOREADY (net-base i))
                    (BUSOE IOBUSOE (net-base i))
                    (BUSLD IOBUSLD (net-base i))
                    (START IOSTART (net-base i))
                    (IOSEL IOSEL)
                    (RESET RESET)
                    (PHI1 PHI1)
                    (PHI2 PHI2)
                    (PHI1INV PHI1INV)
                    (PHI2INV PHI2INV)
                    (Vdd Vdd)
                    (GND GND)
))
;
(instance CLKGEN (
                     (CLK PHI)
                     (PHI1 PHI1)
                     (PHI2 PHI2)
                     (PHI1INV PHI1INV)
                     (PHI2INV PHI2INV)
                     (Vdd Vdd)
                     (GND GND)
))
;
(instance FSM (
               (RESET RESET)
               (IOREADY IOREADY (width ioChannels))
               (INCCYCLE INCCYCLE)
               (FINALCYCLE FINALCYCLE)
               (NEXTDATA NEXTDATA)
               (LEFTEDGE LEFTEDGE)
               (TOPEDGE TOPEDGE)
               (IOBUSOE IOBUSOE (width ioChannels))
               (IOBUSLD IOBUSLD (width ioChannels))
               (IOSTART IOSTART (width ioChannels))
               (IOSEL IOSEL)
               (ADDERFUNC ADDERFUNC)
               (ADDEROUTEN ADDEROUTEN)
               (MULTOUTEN MULTOUTEN)
               (AOUTEN AOUTEN (width numOfRegs))
               (BOUTEN BOUTEN (width numOfRegs))
               (REGLOAD LOAD (width numOfRegs))
               (PHI1 PHI1)
```

```
            (PHI2 PHI2)
            (Vdd Vdd)
            (GND GND)
))
;
(instance parent (
                ((terminal ABUS (DIRECTION OUTPUT)) ABUS (width width))
                ((terminal BBUS (DIRECTION OUTPUT)) BBUS (width width))
                ((terminal EBUS (DIRECTION OUTPUT)) EBUS (width width))
                ((terminal INPUTA (DIRECTION INPUT)) INPUTA (width
(* ioPerChannel ioChannels)))
                ((terminal INPUTB (DIRECTION INPUT)) INPUTB (width
(* ioPerChannel ioChannels)))
                ((terminal OUTPUT (DIRECTION OUTPUT)) OUTPUT (width
(* ioPerChannel ioChannels)))
                ((terminal IOBUSOE (DIRECTION OUTPUT)) IOBUSOE
(width ioChannels))
                ((terminal IOBUSLD (DIRECTION OUTPUT)) IOBUSLD
(width ioChannels))
                ((terminal IOREADY (DIRECTION OUTPUT)) IOREADY
(width ioChannels))
                ((terminal IOSTART (DIRECTION OUTPUT)) IOSTART
(width ioChannels))

                ((terminal AOUTEN (DIRECTION OUTPUT)) AOUTEN
(width numOfRegs))
                ((terminal BOUTEN (DIRECTION OUTPUT)) BOUTEN
(width numOfRegs))
                ((terminal LOAD (DIRECTION OUTPUT)) LOAD
(width numOfRegs))
                ((terminal ADDEROUTEN (DIRECTION OUTPUT)) ADDEROUTEN)
                ((terminal MULTOUTEN (DIRECTION OUTPUT)) MULTOUTEN)
                ((terminal RESET (DIRECTION INPUT)) RESET)
                ((terminal INCCYCLE (DIRECTION INPUT)) INCCYCLE)
                ((terminal FINALCYCLE (DIRECTION INPUT)) FINALCYCLE)
                ((terminal NEXTDATA (DIRECTION INPUT)) NEXTDATA)
                ((terminal LEFTEDGE (DIRECTION INPUT)) LEFTEDGE)
                ((terminal TOPEDGE (DIRECTION INPUT)) TOPEDGE)
                ((terminal PHI (TERMTYPE CLOCK) (DIRECTION INPUT)) PHI)
                ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
                ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.39 offdiag2chip.sdl

This file puts the pad frame around the `offdiag2.sdl` core of the off-diagonal processor. The chip inputs are put on the left side, the outputs are along the right, and the control connections come in on the top of the chip.

```
(parent-cell offdiag2chip)
;
(structure-processor Padgroup)
(layout-generator Padroute lpads1_2)
;
(parameters width numOfRegs (ioPerChannel 4) (diagChannels 4) (hvChannels 2)
(ioChannels (+ hvChannels diagChannels)) (ioLines (* ioPerChannel ioChannels))
          (max_pads_per_side 30)
          (pads (list
                 (list "top" 1 max_pads_per_side)
                 (list "left" (+ max_pads_per_side 1)
(* max_pads_per_side 2))
                 (list "bottom" (+ (* max_pads_per_side 2) 1)
(* max_pads_per_side 3))
                 (list "right" (+ (* max_pads_per_side 3) 1)
(* max_pads_per_side 4))
                 )
                )
          )
;
(subcells
 (offdiag2 CORE ((width width) (numOfRegs numOfRegs)))
 (vdd1_2 (VDD_TOP_1 VDD_TOP_2  VDD_RIGHT_1
     VDD_RIGHT_2  VDD_LEFT_1 VDD_LEFT_2 )
 )
 (gnd1_2
(GND_TOP_1 GND_TOP_2 GND_TOP_3 GND_TOP_4 GND_LEFT_1 GND_LEFT_2
     GND_LEFT_3 GND_LEFT_4 GND_RIGHT_1 GND_RIGHT_2))

 (in1_2
(RESET_PAD INCCYCLE_PAD FINALCYCLE_PAD NEXTDATA_PAD
     LEFTEDGE_PAD TOPEDGE_PAD PHI_PAD))

 )
(dotimes (i (* ioPerChannel ioChannels))
      (subcells
       (in1_2 (INPUTA_PAD INPUTB_PAD))
       (out1_2 OUTPUT_PAD)
       )
)
(dotimes (dummy_top_left (- (round (/ max_pads_per_side 2)) (+ 7 3)))
      (subcells
       (space1_2 DUMMY_PADS_TOP_LEFT)
       ))
(dotimes (dummy_top_right (- max_pads_per_side (+ (round
(/ max_pads_per_side 2)) 3)))
      (subcells
       (space1_2 DUMMY_PADS_TOP_RIGHT)
       ))
(dotimes (dummy_bottom (- (- (* max_pads_per_side 4) (+ (* 2
(* ioPerChannel ioChannels)) 6)) (+ max_pads_per_side (+
(* ioPerChannel ioChannels)) 4)))
```

```
            (subcells
             (space1_2 DUMMY_PADS_BOTTOM)
             ))

;
(net VDD (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
(net PHI (NETTYPE CLOCK))
;
(instance CORE (
                  (INPUTA INPUTA (width (* ioPerChannel ioChannels)))
                  (INPUTB INPUTB (width (* ioPerChannel ioChannels)))
                  (OUTPUT OUTPUT (width ioLines))
                  (RESET RESET)
                  (INCCYCLE INCCYCLE)
                  (FINALCYCLE FINALCYCLE)
                  (NEXTDATA NEXTDATA)
                  (LEFTEDGE LEFTEDGE)
                  (TOPEDGE TOPEDGE)
                  (Vdd Vdd)
                  (GND GND)
                  (PHI PHI)
))
;
(instance INPUTA_PAD (PAD (- (- (* max_pads_per_side 4) 2) i)) (

(padin INPUTA_p (net-base i))

(in INPUTA (net-base i))
                                                                      )
)
;
(instance INPUTB_PAD (PAD (- (- (* max_pads_per_side 4) (+ ioLines 4)) i)) (

(padin INPUTB_p (net-base i))

(in INPUTB (net-base i))
                                                                      )
)
;
(instance OUTPUT_PAD (PAD (+ (+ max_pads_per_side 3) i)) (

(pado OUTPUT_p (net-base i))

(out OUTPUT (net-base i))
))
;
(instance RESET_PAD (PAD '3) (
                                 (padin RESET_p)
                                 (in RESET)
                                 ))
;
(instance INCCYCLE_PAD (PAD '4) (
                                 (padin INCCYCLE_p)
```

```
                                        (in INCCYCLE)
                                        ))
;
(instance FINALCYCLE_PAD (PAD '5) (
                                        (padin FINALCYCLE_p)
                                        (in FINALCYCLE)
                                        ))
;
(instance NEXTDATA_PAD (PAD '6) (
                                        (padin NEXTDATA_p)
                                        (in NEXTDATA)
                                        ))
;
(instance LEFTEDGE_PAD (PAD '7) (
                                        (padin LEFTEDGE_p)
                                        (in LEFTEDGE)
                                        ))
;
(instance TOPEDGE_PAD (PAD '8) (
                                        (padin TOPEDGE_p)
                                        (in TOPEDGE)
                                        ))
;
(instance PHI_PAD (PAD '9) (
                                        (padin PHI_p)
                                        (in PHI)
                                        ))
;
(instance VDD_TOP_1 (PAD (+ (round (/ max_pads_per_side 2)) 0)) (

(padvdd Vdd_p)

(Vdd Vdd)
                                                                          ))
(instance VDD_TOP_2 (PAD (+ (round (/ max_pads_per_side 2)) 1)) (

(padvdd Vdd_p)

(Vdd Vdd)
                                                                          ))
(instance VDD_LEFT_1 (PAD (- (* max_pads_per_side 4) (+ ioLines 2))) (
                                                                          (padvdd Vdd_p)

                                                                          (Vdd Vdd)

                                                                          ))

(instance VDD_LEFT_2 (PAD (- (* max_pads_per_side 4) (+ ioLines 3))) (
                                                                          (padvdd Vdd_p)

                                                                          (Vdd Vdd)

                                                                          ))
```

```
(instance VDD_RIGHT_1 (PAD (+ max_pads_per_side (+ ioLines 3))) (
                                                               (padvdd Vdd_p)

                                                               (Vdd Vdd)

                                                               ))
(instance VDD_RIGHT_2 (PAD (+ max_pads_per_side (+ ioLines 4))) (
                                                               (padvdd Vdd_p)

                                                               (Vdd Vdd)

                                                               ))

;
(instance GND_TOP_1 (PAD '1) (
                        (padgnd GND_p)
                        (GND GND)
                        ))
(instance GND_TOP_2 (PAD '2) (
                        (padgnd GND_p)
                        (GND GND)
                        ))
(instance GND_TOP_3 (PAD (- max_pads_per_side 1)) (
                                     (padgnd GND_p)
                                     (GND GND)
                                     ))
(instance GND_TOP_4 (PAD max_pads_per_side) (
                                     (padgnd GND_p)
                                     (GND GND)
                                     ))
(instance GND_LEFT_1 (PAD (- (* max_pads_per_side 4) 1)) (
                                         (padgnd GND_p)
                                         (GND GND)
                                         ))
(instance GND_LEFT_2 (PAD (* max_pads_per_side 4)) (
                                         (padgnd GND_p)
                                         (GND GND)
                                         ))
(instance GND_LEFT_3 (PAD (- (* max_pads_per_side 4) (+ (* 2
(* ioPerChannel ioChannels)) 4))) (
                                         (padgnd GND_p)
                                         (GND GND)
                                         ))
(instance GND_LEFT_4 (PAD (- (* max_pads_per_side 4) (+ (* 2
(* ioPerChannel ioChannels)) 5))) (
                                         (padgnd GND_p)
                                         (GND GND)
                                         ))
(instance GND_RIGHT_1 (PAD (+ max_pads_per_side 1)) (
                                         (padgnd GND_p)
                                         (GND GND)
                                         ))
(instance GND_RIGHT_2 (PAD (+ max_pads_per_side 2)) (
                                        (padgnd GND_p)
                                        (GND GND)
                                        ))
;
(instance DUMMY_PADS_TOP_LEFT (PAD (+ (+ 7 3) dummy_top_left)))
(instance DUMMY_PADS_TOP_RIGHT (PAD (+ (+ (round (/ max_pads_per_side 2)) 2)
 dummy_top_right)))
(instance DUMMY_PADS_BOTTOM (PAD (+ (+ max_pads_per_side (+
(* ioPerChannel ioChannels) 5)) dummy_bottom)))
;
(instance parent (
            ((terminal INPUTA_p (DIRECTION INPUT)) INPUTA_p (width
(* ioPerChannel ioChannels)))
            ((terminal INPUTB_p (DIRECTION INPUT)) INPUTB_p (width
(* ioPerChannel ioChannels)))
            ((terminal OUTPUT_p (DIRECTION OUTPUT)) OUTPUT_p (width
(* ioPerChannel ioChannels)))
            ((terminal RESET_p (DIRECTION INPUT)) RESET_p)
            ((terminal INCCYCLE_p (DIRECTION INPUT)) INCCYCLE_p)
            ((terminal FINALCYCLE_p (DIRECTION INPUT)) FINALCYCLE_p)
            ((terminal NEXTDATA_p (DIRECTION INPUT)) NEXTDATA_p)
            ((terminal LEFTEDGE_p (DIRECTION INPUT)) LEFTEDGE_p)
            ((terminal TOPEDGE_p (DIRECTION INPUT)) TOPEDGE_p)
            ((terminal PHI_p (TERMTYPE CLOCK) (DIRECTION INPUT))
 PHI_p)
            ((terminal Vdd_p (TERMTYPE SUPPLY) (DIRECTION INPUT))
 Vdd_p)
            ((terminal GND_p (TERMTYPE GROUND) (DIRECTION INPUT))
 GND_p)
))
;
(end-sdl)
```

A.1.40 offdiagctl.sdl

This is the controller for the off-diagonal processor element. It takes as input a global synchronization signal `INCCYCLE`, a reset signal `RESET`, a completion signal, `FINALCYCLE`, and all of the status lines from the functional units. The operation is controlled by the behavioral description file `offdiagctl.bds`.

```
(parent-cell offdiagctl)
;
(parameters (ioChannels 6) (stateBits 6) (cycleBits 3) (numOfRegs 16))
(layout-generator Flint b v)
;
(subcells
        (fsm_bdsyn FSM ((inwidth 21) (outwidth 79)
(bdsyn "~/sdl/offdiagctl/offdiagctl.bds")))
```

```
)
;
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
(net PHI1 (NETTYPE CLOCK))
(net PHI2 (NETTYPE CLOCK))
;
(instance FSM (
                (IN reset (term-base 0))
                (IN ioReady (term-base 1) (width ioChannels))
                (IN incCycle (term-base (+ ioChannels 1)))
                (IN finalCycle (term-base (+ ioChannels 2)))
                (IN nextData (term-base (+ ioChannels 3)))
                (IN leftEdge (term-base (+ ioChannels 4)))
                (IN topEdge (term-base (+ ioChannels 5)))
                (IN cycle (term-base (+ ioChannels 6)) (width cycleBits))
                (IN state (term-base (+ (+ ioChannels 6) cycleBits))
(width stateBits))
                (OUT iobusoe (term-base 0) (width ioChannels))
                (OUT iobusld (term-base ioChannels) (width ioChannels))
                (OUT iostart (term-base (* 2 ioChannels)) (width ioChannels))
                (OUT ioSel (term-base (* 3 ioChannels)))
                (OUT adderFunc (term-base (+ (* 3 ioChannels) 1)))
                (OUT adderOutEn (term-base (+ (* 3 ioChannels) 2)))
                (OUT multOutEn (term-base (+ (* 3 ioChannels) 3)))
                (OUT aOutEn (term-base (+ (* 3 ioChannels) 4))
(width numOfRegs))
                (OUT bOutEn (term-base (+ (+ (* 3 ioChannels) 4) numOfRegs))
(width numOfRegs))
                (OUT regLoad (term-base (+ (+ (* 3 ioChannels) 4)
(* 2 numOfRegs))) (width numOfRegs))
                (OUT cycle (term-base (+ (+ (* 3 ioChannels) 4)
(* 3 numOfRegs))) (width cycleBits))
                (OUT state (term-base (+ (+ (+ (* 3 ioChannels) 4)
(* 3 numOfRegs)) cycleBits)) (width stateBits))
                (PHI1 PHI1)
                (PHI2 PHI2)
                (Vdd Vdd)
                (GND GND)
))
;
(instance parent (
                    ((terminal RESET (DIRECTION INPUT)) reset)
                    ((terminal IOREADY (DIRECTION INPUT)) ioReady
(width ioChannels))
                    ((terminal INCCYCLE (DIRECTION INPUT)) incCycle)
                    ((terminal FINALCYCLE (DIRECTION INPUT)) finalCycle)
                    ((terminal NEXTDATA (DIRECTION INPUT)) nextData)
                    ((terminal LEFTEDGE (DIRECTION INPUT)) leftEdge)
                    ((terminal TOPEDGE (DIRECTION INPUT)) topEdge)
                    ((terminal IOBUSOE (DIRECTION OUTPUT)) iobusoe
(width ioChannels))
                    ((terminal IOBUSLD (DIRECTION OUTPUT)) iobusld
(width ioChannels))
```

```
                ((terminal IOSTART (DIRECTION OUTPUT)) iostart
(width ioChannels))
                ((terminal IOSEL (DIRECTION OUTPUT)) ioSel)
                ((terminal ADDERFUNC (DIRECTION OUTPUT)) adderFunc )
                ((terminal ADDEROUTEN (DIRECTION OUTPUT)) adderOutEn)
                ((terminal MULTOUTEN (DIRECTION OUTPUT)) multOutEn)
                ((terminal AOUTEN (DIRECTION OUTPUT)) aOutEn
(width numOfRegs))
                ((terminal BOUTEN (DIRECTION OUTPUT)) bOutEn
(width numOfRegs))
                ((terminal REGLOAD (DIRECTION OUTPUT)) regLoad
(width numOfRegs))
                ((terminal CYCLE (DIRECTION OUTPUT)) cycle
(width cycleBits))
                ((terminal STATE (DIRECTION OUTPUT)) state
(width stateBits))
                ((terminal PHI1 (DIRECTION INPUT)
                        (TERMTYPE CLOCK)) PHI1)
                ((terminal PHI2 (DIRECTION INPUT)
                        (TERMTYPE CLOCK)) PHI2)
                ((terminal Vdd (DIRECTION INPUT)
                        (TERMTYPE SUPPLY)) Vdd)
                ((terminal GND (DIRECTION INPUT)
                        (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.41 regfile2p.sdl

This file generates an array of instances of the datapath library cell scanreg2t, which is a scan register with two tristate buffers on its output. It is designed to sit in a two-operand, one result bus architecture.

```
(parent-cell regfile2p)
;
(parameters width numOfRegs )
;
(structure-processor SIVcheck)
(layout-generator Flint b v)

(subcells
        (regfile2pdpp REGFILE ((N width) (numOfRegs numOfRegs)))
        (regfile2plogic LOGIC ((N numOfRegs)))
        (myclock CLKGEN)
)
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE Vdd))
```

```
(net PHI (NETTYPE CLOCK))
(net PHI1 (NETTYPE CLOCK))
(net PHI2 (NETTYPE CLOCK))
(net PHI1INV (NETTYPE CLOCK))
(net PHI2INV (NETTYPE CLOCK))
;
(instance REGFILE (
                    (ABUS ABUS (width width))
                    (BBUS BBUS (width width))
                    (EBUS EBUS (width width))
                    (AOUTEN AOUTEN (width numOfRegs))
                    (AOUTENINV AOUTENINV (width numOfRegs))
                    (BOUTEN BOUTEN (width numOfRegs))
                    (BOUTENINV BOUTENINV (width numOfRegs))
                    (LOAD GLOAD (width numOfRegs))
                    (LOADINV GLOADINV (width numOfRegs))
                    (SCAN SCAN (width numOfRegs))
                    (SCANINV SCANINV (width numOfRegs))
                    (SCANIN GND (merge 0 ( – numOfRegs 1)))
                    (PHI1 PHI1)
                    (PHI1INV PHI1INV)
                    (PHI2 PHI2)
                    (PHI2INV PHI2INV)
                    (Vdd Vdd)
                    (GND GND)
))
;
(instance LOGIC (
                    (AOUTEN AOUTEN (width numOfRegs))
                    (AOUTENINV AOUTENINV (width numOfRegs))
                    (BOUTEN BOUTEN (width numOfRegs))
                    (BOUTENINV BOUTENINV (width numOfRegs))
                    (LOAD LOAD (width numOfRegs))
                    (GLOAD GLOAD (width numOfRegs))
                    (GLOADINV GLOADINV (width numOfRegs))
                    (SCAN SCAN (width numOfRegs))
                    (SCANINV SCANINV (width numOfRegs))
                    (Vdd Vdd)
                    (GND GND)
))
;
(instance CLKGEN (
          (CLK PHI)
          (PHI1 PHI1)
          (PHI2 PHI2)
          (PHI1INV PHI1INV)
          (PHI2INV PHI2INV)
          (Vdd Vdd)
          (GND GND)
))
;
(instance parent (
                    ((terminal ABUS (DIRECTION OUTPUT)) ABUS (width width))
                    ((terminal BBUS (DIRECTION OUTPUT)) BBUS (width width))
                    ((terminal EBUS (DIRECTION INPUT)) EBUS (width width))
                    ((terminal AOUTEN (DIRECTION INPUT)) AOUTEN
(width numOfRegs))
                    ((terminal BOUTEN (DIRECTION INPUT)) BOUTEN
(width numOfRegs))
                    ((terminal LOAD (DIRECTION INPUT)) LOAD (width numOfRegs))
                    ((terminal SCAN (DIRECTION INPUT)) SCAN (width numOfRegs))
                    ((terminal PHI (TERMTYPE CLOCK)) PHI)
                    ((terminal PHI1 (TERMTYPE CLOCK)) PHI1)
                    ((terminal PHI1INV (TERMTYPE CLOCK)) PHI1INV)
                    ((terminal PHI2 (TERMTYPE CLOCK)) PHI2)
                    ((terminal PHI2INV (TERMTYPE CLOCK)) PHI2INV)
                    ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
                    ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.42 regfile2pdpp.sdl

This file contains the actual instances of the registers for the `regfile2p.sdl` cell. The registers come from the datapath library.

```
(parent-cell regfile2pdpp)
;
(parameters N numOfRegs)
;
(structure-processor dpp)
;
(layout-generator Flint a)
;
(dotimes (i numOfRegs)
          (subcells (scanreg2t REGISTER ((N N)))
                    )
          )
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE Vdd))
(net PHI1 (NETTYPE CLOCK))
(net PHI2 (NETTYPE CLOCK))
(net PHI1INV (NETTYPE CLOCK))
(net PHI2INV (NETTYPE CLOCK))
;
(instance REGISTER (
                    (IN EBUS)
                    (ABUS ABUS)
                    (BBUS BBUS)
                    (SCANIN SCANIN (net-base i))
                    (SCANOUT SCANOUT (net-base i))
                    (LOAD LOAD (net-base i))
```

```
                        (LOADINV LOADINV (net-base i))
                        (SCAN SCAN (net-base i))
                        (SCANINV SCANINV (net-base i))
                        (KEEP LOADINV (net-base i))
                        (KEEPINV LOAD (net-base i))
                        (AOUT AOUTEN (net-base i))
                        (AOUTINV AOUTENINV (net-base i))
                        (BOUT BOUTEN (net-base i))
                        (BOUTINV BOUTENINV (net-base i))
                        (PHI2 PHI2)
                        (PHI2INV PHI2INV)
                        (PHI1 PHI1)
                        (PHI1INV PHI1INV)
                        (Vdd Vdd)
                        (GND GND)
))
;
(instance parent (
                      ((terminal ABUS (DIRECTION OUTPUT)) ABUS)
                      ((terminal BBUS (DIRECTION OUTPUT)) BBUS)
                      ((terminal EBUS (DIRECTION INPUT)) EBUS)
                      ((terminal AOUTEN (DIRECTION INPUT)) AOUTEN
(width numOfRegs))
                      ((terminal AOUTENINV (DIRECTION INPUT)) AOUTENINV
(width numOfRegs))
                      ((terminal BOUTEN (DIRECTION INPUT)) BOUTEN
(width numOfRegs))
                      ((terminal BOUTENINV (DIRECTION INPUT)) BOUTENINV
(width numOfRegs))
                      ((terminal LOAD (DIRECTION INPUT)) LOAD (width numOfRegs))
                      ((terminal LOADINV (DIRECTION INPUT)) LOADINV
(width numOfRegs))
                      ((terminal SCAN (DIRECTION INPUT)) SCAN (width numOfRegs))
                      ((terminal SCANINV (DIRECTION INPUT)) SCANINV
(width numOfRegs))
                      ((terminal SCANIN (DIRECTION INPUT)) SCANIN
(width numOfRegs))
                      ((terminal SCANOUT (DIRECTION INPUT)) SCANOUT
(width numOfRegs))
                      ((terminal PHI1 (TERMTYPE CLOCK)) PHI1)
                      ((terminal PHI1INV (TERMTYPE CLOCK)) PHI1INV)
                      ((terminal PHI2 (TERMTYPE CLOCK)) PHI2)
                      ((terminal PHI2INV (TERMTYPE CLOCK)) PHI2INV)
                      ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
                      ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.43 regfile2plogic.sdl

This file implements the random logic needed for generation of complements of control signals for `regfile2pdpp.sdl` in `regfile2p.sdl`.

```
(parent-cell regfile2plogic)
;
(parameters N)
;
(structure-processor dpp)
(layout-generator Flint a)

(subcells
        (inverter ACNTLINV ((N N)))
        (inverter BCNTLINV ((N N)))
        (inverter LDCNTLINV ((N N)))
        (inverter SCANCNTLINV ((N N)))
        (and2 LOADGATE ((N N)))
)
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE SUPPLY))
;
(instance ACNTLINV (
        (IN AOUTEN)
        (OUTINV AOUTENINV)
        (Vdd Vdd)
        (GND GND)
))
(instance BCNTLINV (
        (IN BOUTEN)
        (OUTINV BOUTENINV)
        (Vdd Vdd)
        (GND GND)
))
(instance LDCNTLINV (
        (IN GLOAD)
        (OUTINV GLOADINV)
        (Vdd Vdd)
        (GND GND)
))
(instance SCANCNTLINV (
        (IN SCAN)
        (OUTINV SCANINV)
        (Vdd Vdd)
        (GND GND)
))
(instance LOADGATE (
        (A LOAD)
        (B SCANINV)
        (AND2 GLOAD)
))
;
(instance parent (
                ((terminal AOUTEN (DIRECTION INPUT)) AOUTEN)
                ((terminal AOUTENINV (DIRECTION OUTPUT)) AOUTENINV)
                ((terminal BOUTEN (DIRECTION INPUT)) BOUTEN)
                ((terminal BOUTENINV (DIRECTION OUTPUT)) BOUTENINV)
                ((terminal LOAD (DIRECTION INPUT)) LOAD)
                ((terminal GLOAD (DIRECTION OUTPUT)) GLOAD)
                ((terminal GLOADINV (DIRECTION OUTPUT)) GLOADINV)
                ((terminal SCAN (DIRECTION INPUT)) SCAN)
                ((terminal SCANINV (DIRECTION OUTPUT)) SCANINV)
                ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
                ((terminal GND (TERMTYPE GROUND)) GND)
))
;
(end-sdl)
```

A.1.44 register.sdl

This is a simple loadable register used to store temporary values during computation in the fast interpolation cell.

```
(parent-cell register)
;
(parameters width)
;
(structure-processor SIVcheck)
(layout-generator Flint v b)
;
(subcells (registerdpp REGISTER ((N width)))
        (registerlogic LOGIC)
)
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE SUPPLY))
(net PHI1 (NETTYPE CLOCK))
(net PHI1INV (NETTYPE CLOCK))
(net PHI2 (NETTYPE CLOCK))
(net PHI2INV (NETTYPE CLOCK))

;
(instance REGISTER (
        (D D (width width))
        (Q Q (width width))
        (SCANIN GND)
        (LOAD LOAD)
        (LOADINV LOADINV)
        (SCAN GND)
        (SCANINV Vdd)
        (PHI2 PHI2)
        (PHI2INV PHI2INV)
        (PHI1 PHI1)
        (PHI1INV PHI1INV)
        (Vdd Vdd)
        (GND GND)
))
(instance LOGIC (
        (LOAD LOAD)
        (LOADINV LOADINV)
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal D (DIRECTION INPUT)) D (width width))
        ((terminal Q (DIRECTION OUTPUT)) Q (width width))
        ((terminal LOAD (DIRECTION INPUT)) LOAD )
        ((terminal PHI1 (TERMTYPE CLOCK)) PHI1)
        ((terminal PHI1INV (TERMTYPE CLOCK)) PHI1INV)
        ((terminal PHI2 (TERMTYPE CLOCK)) PHI2)
        ((terminal PHI2INV (TERMTYPE CLOCK)) PHI2INV)
        ((terminal Vdd (TERMTYPE SUPPLY) (TERM_EDGE TOP)) Vdd)
        ((terminal GND (TERMTYPE GROUND) (TERM_EDGE BOTTOM)) GND)
))
;
(end-sdl)
```

A.1.45 registerdpp.sdl

This is the actual scan register used by the `register.sdl` cell. It is implemented as a scan register from the datapath library.

```
(parent-cell registerdpp)
;
(parameters N)
;
(structure-processor dpp)
(layout-generator Flint a)
;
(subcells (scanreg REGISTER ((N N)))
)
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE SUPPLY))
(net PHI1 (NETTYPE CLOCK))
(net PHI1INV (NETTYPE CLOCK))
(net PHI2 (NETTYPE CLOCK))
(net PHI2INV (NETTYPE CLOCK))
```

```
;
(instance REGISTER (
        (IN D)
        (OUT Q)
        (SCANIN SCANIN)
        (LOAD LOAD)
        (LOADINV LOADINV)
        (SCAN SCAN)
        (SCANINV SCANINV)
        (KEEP LOADINV)
        (KEEPINV LOAD)
        (PHI2 PHI2)
        (PHI2INV PHI2INV)
        (PHI1 PHI1)
        (PHI1INV PHI1INV)
        (Vdd Vdd)
        (GND GND)
))
;
(instance parent (
        ((terminal D (DIRECTION INPUT)) D)
        ((terminal Q (DIRECTION OUTPUT)) Q)
        ((terminal LOAD (DIRECTION INPUT)) LOAD)
        ((terminal LOADINV (DIRECTION INPUT)) LOADINV)
        ((terminal SCAN (DIRECTION INPUT)) SCAN)
        ((terminal SCANINV (DIRECTION INPUT))SCANINV)
        ((terminal SCANIN (DIRECTION INPUT)) SCANIN)
        ((terminal PHI1 (TERMTYPE CLOCK)) PHI1)
        ((terminal PHI1INV (TERMTYPE CLOCK)) PHI1INV)
        ((terminal PHI2 (TERMTYPE CLOCK)) PHI2)
        ((terminal PHI2INV (TERMTYPE CLOCK)) PHI2INV)
        ((terminal Vdd (TERMTYPE SUPPLY) ) Vdd)
        ((terminal GND (TERMTYPE GROUND) ) GND)
))
;
(end-sdl)
```

A.1.46 registerlogic.sdl

This file contains the logic needed for control signal generation in the `register.sdl` cell. The logic is implemented from the standard cell library.

```
(parent-cell registerlogic)
;
(layout-generator Stdcell)
;
(subcells (invf103 INV))
;
(net GND (NETTYPE GROUND))
(net Vdd (NETTYPE SUPPLY))
```

```
;
(instance INV (
        (A1 LOAD )
        (O LOADINV)
))
;
(instance parent (
        ((terminal LOAD (DIRECTION INPUT)) LOAD)
        ((terminal LOADINV (DIRECTION OUTPUT)) LOADINV)
        ((terminal Vdd (TERMTYPE SUPPLY) (TERM_EDGE TOP)) Vdd)
        ((terminal GND (TERMTYPE GROUND) (TERM_EDGE BOTTOM)) GND)
))
;
(end-sdl)
```

A.2 BDS Files

A.2.1 diagctl.bds

This file contains the behavioral description for performing the calculations of the diagonal processor as presented in chapter 2 for computation of the SVD on a systolic processor array as a three cycle computation. This file is used to generate the PLA grid for a finite state machine.

```
! io2reg(serio, r, state, nextstate) : take io from serial i/o serio and loads into reg r

! reg2io(r, serio, state, nextstate) : take io from serial i/o serio and loads into reg r

! mul(r1, r2, r3, state, nextstate): r1 * r2 -> r3; all complex

! mag(r1, r2, state, nextstate): r1 * comp(r1) -> r2; r1 complex, r2 real

! cmul(r1, r2, r3, state, nextstate): r1 * comp(r2) -> r3; r1, r2, r3 complex

! rmul (r1, r2, r3, state, nextstate): r1 * r2 -> r3; all real

! crmul(r1, r2, r3, state, nextstate): r1 * r2 -> r3; r1 real, r2, r3 complex

! ccrmul(r1, r2, r3, state, nextstate): r1 * conj(r2) -> r3; r1 real, r2, r3 complex

! add(r1, r2, r3, state, nextstate): r1 + r2 -> r3, all complex

! sub(r1, r2, r3, state, nextstate): r1 - r2 -> r3, all complex

! radd(r1, r2, r3, state, nextstate): r1 + r2 -> r3, all real

! sqrt(r1, r2, state, nextstate): sqaure_root(r1) -> r2; r1, r2 real: NOTE: take 3 steps
! invsqrt(r1, r2, state, nextstate): 1/square_root(r1) -> r2; r1, r2 real: NOTE: take 3 steps


model offdiagctl
OUT<88:0> = IN<23:0>;
        synonym iobusoe<5:0>   = out<5:0>,
                iobusld<5:0>   = out<11:6>,
                iostart<5:0>   = out<17:12>,
                ioSel          = out<18>,
                adderFunc      = out<19>,
                adderOutEn     = out<20>,
                adderRealAdd   = out<21>,
                multOutEn      = out<22>,
                multCompB      = out<23>,
                multComplexOut = out<24>,
                multRealA      = out<25>,
                interpGo       = out<26>,
                interpFunc     = out<27>,
                interpOutEn    = out<28>,
                bshuftOutEn    = out<29>,
                aOutEn<15:0>   = out<45:30>,
                bOutEn<15:0>   = out<61:46>,
                regLoad<15:0>  = out<77:62>,
                nextCycle<2:0> = out<80:78>,
                nextState<5:0> = out<86:81>,
                nextSubState<1:0> = out<88:87>,
                reset          = in<0>,
                alphaReady     = in<1>,
                betaReady      = in<2>,
                gammaReady     = in<3>,
                deltaReady     = in<4>,
                hReady         = in<5>,
                vReady         = in<6>,
                incCycle       = in<7>,
                finalCycle     = in<8>,
                nextData       = in<9>,
                leftEdge       = in<10>,
                topEdge        = in<11>,
                interpBusy     = in<12>,
                presentCycle<2:0>= in<15:13>,
                presentState<5:0>        = in<21:16>,
                presentSubState<1:0>     = in<23:22>;
        constant        alpha  = 1,
                        beta   = 2,
                        gamma  = 4,
                        delta  = 8,
                        abgd   = 15,
                        horiz  = 16,
                        vert   = 32,
                        handv  = 48,
                        r0     = 1,
                        r1     = 2,
                    r2          = 4,
                    r3          = 8,
                    r4          = 16,
                    r5          = 32,
                    r6          = 64,
                    r7          = 128,
                    r8          = 256,
                    r9          = 512,
                    r10         = 1024,
                    r11         = 2048,
                    r12         = 4096,
                    r13         = 8192,
                    r14         = 16384,
                    r15         = 32768,
                    none        = 0,
                    ADD         = 1,
                    SUB         = 0,
                    MULT        = 1,
                    PassInputs  = 0,
                    LoadInput   = 1,
                    idle        = 0,
                    load        = 1,
                    cyc1        = 2,
                    cyc2        = 3,
                    cyc3        = 4,
                    pass        = 5,
                    unload      = 6,
                    true        = 1,
                    false       = 0,
                    squarert = 0,
                    invsquarert = 1;

routine main;
        ! defaults
        nextState = 0;               ! reset by default
        nextSubState = 0;            ! no substate
        iobusoe = none;              ! nobody on the bus
        iobusld = none;              ! nobody loading off bus
        iostart = none;              ! nobody starting io
        ioSel = PassInputs;          ! use internal connections
        adderOutEn = none;           ! adder not on bus
        adderFunc = ADD;             ! adder adds
        multOutEn = none;            ! mult not on bus either
        multComplexOut = true;       ! complex multiply by default
        multCompB = false;           ! do not complement b in multiplies
        adderRealAdd =false;         ! complex adds
        interpGo = false;            ! no interp
        interpOutEn = false;         ! no interp on bus
        interpFunc = squarert;       ! square root by default
        aOutEn = none;               ! no regs on abus
        bOutEn = none;               ! no regs on bbus
        regLoad = none;              ! no regs loading
        nextCycle = presentCycle;    ! waiting to go
        ! now lets do something...
select presentCycle from
        [idle]: if incCycle then
                        begin
                                nextState = 0;          ! reset state
                                nextCycle = load;       ! first load
                        end;
        [load]: begin
                ioSel = LoadInput;
                if incCycle then
                        begin
                                nextState = 0;          ! reset state
                                nextCycle = cyc1;       ! done loading
                        end
                else begin
                 select presentState from
                        [0]: begin
                            if nextData then begin
                                iostart = abgd;         ! start r/w in col
                                nextState = 1;          ! proceed
                                end
                          else begin
                                nextState = 0;          ! still waitint
                                end;
                         end;
                        [1]: begin
                            if alphaReady then begin
                                nextState = 0;          ! done w/this one
                             end else begin
                                nextState = 1;          ! not done
                                end;
                            end;
                  endselectselect; ! presentState
                 end; !else
                end; !cycle
        [cyc1]: begin
                if incCycle then
                        begin
                                nextState = 0;          ! reset state
                                nextCycle = cyc2;       ! next cycle
                        end
                else begin
                 select presentState from
                        [0]: begin
                                iobusoe = alpha;                ! alpha on bus
                                regload = r0;           ! r0 <- alpha
                                nextState = 1;
                        end;
                        [1]: begin
                                iobusoe = beta;         ! beta on bus
                                regload = r1;           ! r1 <- beta
                                nextState = 2;
                        end;
```

```
              [2]: begin
                        iobusoe = gamma;                  ! gamma on bus
                        regload = r2;           ! r2 <-- gamma
                        nextState = 3;
              end;
              [3]: begin
                        iobusoe = delta;                  ! delta on bus
                        regload = r3;           ! r3 <-- delta
                        nextState = 4;
              end;
              [4]: begin
                        nextState = 4;    ! wait here
              end;
! register map for this cycle:  r0 = alpha    r1 = beta          r2 = gamma
!                               r3 = delta    r4 = magSqAlpha r5 = magSqGamma
!                               r6 = betaP    r7 = cTg           r8 = sTg
!                               r9 = Ta       r10 = Tb           r11 = Tg
!                               r12 = Td      r13 = Temp1        r14 = Temp2
!                               r15 = =AddTemp
              [5]: begin
                        aOutEn = r0;
                        bOutEn = r0;
                        multOutEn = MULT;
                        multComplexOut = false;
                        multCompB = true;
                        regLoad = r4;
                        nextState = 6;
              end;                    ! alpha *c(alpha)-->magSqAlpha
              [6]: begin
                        aOutEn = r2;
                        bOutEn = r2;
                        multOutEn = MULT;
                        multComplexOut = false;
                        multCompB = true;
                        regLoad = r5;
                        nextState = 7;
              end;                    ! gamma *c(gamma)-->magSqGamma
              [7]: begin
                        aOutEn = r4;
                        bOutEn = r5;
                        adderOutEn =ADD;
                        regLoad = r15;
                        nextState = 8;
              end;          ! magSqAlpha+magSqGamma-->AddT

              [8]: begin
                        select presentSubState from
                         [0]: begin
                                  aOutEn = r4;
                                  interpGo = true;
                                  interpFunc = invsquarert;
                                  nextState = 8;
                                  nextSubState = 1;
                         end;
```

```
            [1]: begin
                    aOutEn = r4;
                    interpFunc = invsquarert;
                    if interpBusy then begin
                            nextSubState = 1;
                    end else begin
                            nextSubState = 2;
                    end;
                    nextState = 8;
            end;
            [2]: begin
                    interpFunc = invsquarert;
                    interpOutEn = true;
                    regLoad = r9;
                    nextSubState = 0;
                    nextState = 11;
            end;
            endselectselect;
    end;            ! Ta = 1/magSqAlpha
    [11]: begin
            aOutEn = r9;
            bOutEn = r0;
            multOutEn =MULT;
            multRealA = true;
            multCompB = false;
            regLoad = r9 ;
            nextState = 12;
    end;            ! Ta = Ta * c(beta)
    [12]: begin
            select presentSubState from
            [0]: begin
                    aOutEn = r5;
                    interpGo = true;
                    interpFunc = invsquarert;
                    nextState = 12;
                    nextSubState = 1;
            end;
            [1]: begin
                    aOutEn = r5;
                    interpFunc = invsquarert;
                    if interpBusy then begin
                            nextSubState = 1;
                    end else begin
                            nextSubState = 2;
                    end;
                    nextState = 12;
            end;
            [2]: begin
                    interpFunc = invsquarert;
                    interpOutEn = true;
                    regLoad = r11;
                    nextSubState = 0;
                    nextState = 15;
            end;
```

```
          endselectselect;
  end;             ! Tg = 1/magSqGamma
  [15]: begin
          aOutEn = r11;
          bOutEn = r2;
          multOutEn =MULT;
          multRealA = true;
          multCompB = false;
          regLoad = r11;
          nextState = 16;
  end;    ! Tg = Tg * c(gammma)

  [16]: begin
          select presentSubState from
           [0]: begin
                  aOutEn = r4;
                  interpGo = true;
                  nextState = 16;
                  nextSubState = 1;
          end;
          [1]: begin
                  aOutEn = r4;
                  if interpBusy then begin
                          nextSubState = 1;
                  end else begin
                          nextSubState = 2;
                  end;
                  nextState = 16;
          end;
          [2]: begin
                  interpOutEn = true;
                  regLoad = r4;
                  nextSubState = 0;
                  nextState = 19;
          end;
          endselectselect;
  end;                       ! magAlpha = magSqAlpha
  [19]: begin
          select presentSubState from
           [0]: begin
                  aOutEn = r5;
                  interpGo = true;
                  nextState = 19;
                  nextSubState = 1;
          end;
          [1]: begin
                  aOutEn = r5;
                  if interpBusy then begin
                          nextSubState = 1;
                  end else begin
                          nextSubState = 2;
                  end;
                  nextState = 19;
          end;
```

```
            [2]: begin
                    interpOutEn = true;
                    regLoad = r5;
                    nextSubState = 0;
                    nextState = 22;
            end;
          endselectselect;
  end;             ! magGamma = magSqGamma
!

  [22]: begin
          select presentSubState from
            [0]: begin
                    aOutEn = r15;
                    interpGo = true;
                    interpFunc = invsquarert;
                    nextState = 22;
                    nextSubState = 1;
            end;
            [1]: begin
                    aOutEn = r15;
                    interpFunc = invsquarert;
                    if interpBusy then begin
                            nextSubState = 1;
                    end else begin
                            nextSubState = 2;
                    end;
                    nextState = 22;
            end;
            [2]: begin
                    interpFunc = invsquarert;
                    interpOutEn = true;
                    regLoad = r13;
                    nextSubState = 0;
                    nextState = 25;
            end;
          endselectselect;
  end;             ! Temp1 = 1/SQRT(addTemp)
  [25]: begin
          aOutEn = r13;
          bOutEn = r4;
          multOutEn =MULT;
          multComplexOut = false;
          regLoad = r7;
          nextState = 26;
  end;             ! cTg = magAlpha * Temp1
  [26]: begin
          aOutEn = r13;
          bOutEn = r5;
          multOutEn =MULT;
          multComplexOut = false;
          regLoad = r8;
          nextState = 27;
  end;             ! sTg = magGamma * Temp1
!
```

```
[27]: begin
        aOutEn = r7;
        bOutEn = r9;
        multOutEn =MULT;
        multRealA = true;
        regLoad = r13;
        nextState = 28;
end;            ! cTg * Ta -> Temp1
[28]: begin
        aOutEn = r13;
        bOutEn = r1;
        multOutEn =MULT;! multiply
        regLoad = r13;
        nextState = 29;
end;            ! Temp1 * beta -> Temp1
[29]: begin
        aOutEn = r8;
        bOutEn = r11;
        multOutEn =MULT;
        multRealA = true;
        regLoad = r14;
        nextState = 30;
end;            ! sTg * Tg -> Temp2
[30]: begin
        aOutEn = r14;
        bOutEn = r3;
        multOutEn =MULT;! multiply
        regLoad = r14;
        nextState = 31;
end;            ! Temp2 * delta -> Temp2
[31]: begin
        aOutEn = r13;
        bOutEn = r14;
        adderOutEn =ADD;
        regLoad = r6;
        nextState = 32;
end;            ! Temp1 + Temp2 -> betaP
[32]: begin
        aOutEn = r6;
        bOutEn = r6;
        multOutEn = MULT;
        multComplexOut = false;
        multCompB = true;
        regLoad = r13;
        nextState = 33;
end;            ! Temp1 = magsq(betaP)
[33]: begin
        select presentSubState from
         [0]: begin
                aOutEn = r13;
                interpGo = true;
                interpFunc = invsquarert;
                nextState = 33;
                nextSubState = 1;
                end;
                [1]: begin
                        aOutEn = r13;
                        interpFunc = invsquarert;
                        if interpBusy then begin
                                nextSubState = 1;
                        end else begin
                                nextSubState = 2;
                        end;
                        nextState = 33;
                end;
                [2]: begin
                        interpFunc = invsquarert;
                        interpOutEn = true;
                        regLoad = r14;
                        nextSubState = 0;
                        nextState = 36;
                end;
                endselectselect;
        end;            ! Temp2 = 1/SQRT(Temp1)
        [36]: begin
                aOutEn = r14;
                bOutEn = r6;
                multOutEn =MULT;
                multRealA = true;
                multCompB = false;
                regLoad = r10;
                nextState = 37;
        end;    ! Temp2 * C(betaP) -> Tb
!
        [37]: begin
                aOutEn = r8;
                bOutEn = r9;
                multOutEn =MULT;
                multRealA = true;
                regLoad = r13;
                nextState = 38;
        end;            ! sTg * Ta -> Temp1
        [38]: begin
                aOutEn = r13;
                bOutEn = r3;
                multOutEn =MULT;! multiply
                regLoad = r13;
                nextState = 39;
        end;            ! Temp1 * delta -> Temp1
        [39]: begin
                aOutEn = r7;
                bOutEn = r11;
                multOutEn =MULT;
                multRealA = true;
                regLoad = r14;
                nextState = 40;
        end;            ! cTg * Tg -> Temp2
        [40]: begin
```

```
            aOutEn = r14;
            bOutEn = r1;
            multOutEn =MULT;! multiply
            regLoad = r14;
            nextState = 41;
    end;              ! Temp2 * beta -> Temp2
    [41]: begin
            aOutEn = r13;
            bOutEn = r14;
            adderOutEn =ADD;
            adderFunc = SUB;
            regLoad = r15;
            nextState = 42;
    end;              ! Temp1 - Temp2 -> addTemp
    [42]: begin
            aOutEn = r15;
            bOutEn = r15;
            multOutEn = MULT;
            multComplexOut = false;
            multCompB = true;
            regLoad = r13;
            nextState = 43;
    end;              ! addTemp^2 = Temp1
    [43]: begin
            select presentSubState from
             [0]: begin
                    aOutEn = r13;
                    interpGo = true;
                    interpFunc = invsquarert;
                    nextState = 43;
                    nextSubState = 1;
             end;
             [1]: begin
                    aOutEn = r13;
                    interpFunc = invsquarert;
                    if interpBusy then begin
                            nextSubState = 1;
                    end else begin
                            nextSubState = 2;
                    end;
                    nextState = 43;
             end;
             [2]: begin
                    interpFunc = invsquarert;
                    interpOutEn = true;
                    regLoad = r14;
                    nextSubState = 0;
                    nextState = 46;
             end;
            endselectselect;
    end;              ! 1/SQRT(Temp1) -> Temp2
    [46]: begin
            aOutEn = r14;
            bOutEn = r15;
```

```
                        multOutEn =MULT;
                        multRealA = true;
                        multCompB = false;
                        regLoad = r12;
                        nextState = 47;
                end;    ! Temp2 * C(AddTemp)->Td
                [47]: begin
                        aOutEn = r10;
                        bOutEn = r12;
                        multOutEn =MULT;! multiply
                        regLoad = r12;
                        nextState = 48;
                end;            ! Tb * Td -> Td
!
                [48]: begin
                        iobusld = vert;         ! r11 on bus
                        aOutEn  = r11;          ! vert <- r11
                        nextState = 49;
                end;            ! Tg -> vert
                [49]: begin
                        iobusld = horiz;                ! r8 on bus
                        aOutEn  = r8;           ! horiz <- r8
                        nextState = 50;
                end;            ! sTg -> horiz
                [50]: begin
                        nextState = 50;  ! wait here
                end;
!
         endselectselect; ! presentState
        end; ! else
end; ! cyc1
[cyc2]: begin
        if incCycle then
                begin
                        nextState = 0;          ! reset state
                        nextCycle = cyc3;       ! next cycle
                end
        else begin
         select presentState from
                [0]: begin
                        iostart = handv;        ! send Tg on vert,
                                                ! sTg on horiz
                        nextState = 1;
                end;
                [1]: if hReady then begin
                        aOutEn = r7;
                        iobusld = horiz;
                        nextState = 3;
                end else begin
                        nextState = 1;          ! waiting for io
                end;
                [3]: begin
                        iostart = horiz;        ! send cTg
                        nextState = 4;
```

```
                                end;
                                [4]: if hReady then begin
                                        aOutEn = r9;
                                        regLoad = horiz;
                                        nextState = 5;
                                end else begin
                                        nextState = 4;          ! waiting for io
                                end;
                                [5]: begin
                                        iostart = horiz;        ! send Ta
                                        nextState = 6;
                                end;
                                [6]: if hReady then begin
                                        aOutEn = r10;
                                        regLoad = horiz;
                                        nextState = 7;
                                end else begin
                                        nextState = 6;          ! waiting for io
                                end;
                                [7]: begin
                                        iostart = horiz;        ! send Tb
                                        nextState = 8;
                                end;
                                [8]: if hReady then begin
                                        aOutEn = r12;
                                        regLoad = horiz;
                                        nextState = 9;
                                end else begin
                                        nextState = 8;          ! waiting for io
                                end;
                                [9]: begin
                                        iostart = horiz;        ! send Td
                                        nextState = 10;
                                end;
                                [10]: if hReady then begin
                                        nextState = 11;
                                end else begin
                                        nextState = 10;         ! waiting for io
                                end;
! finish up applying values from cyc1
                                [11]: begin
                                        aOutEn = r7;
                                        bOutEn = r4;
                                        multOutEn =MULT;! multiply
                                        regLoad = r0;
                                        nextState = 12;
                                end;    ! cTg * magAlpha -> alpha
                                [12]: begin
                                        aOutEn = r8;
                                        bOutEn = r5;
                                        multOutEn =MULT;! multiply
                                        regLoad = r2;
                                        nextState = 13;
                                end;    ! sTg * magGamma -> gamma
```

```
                  [13]: begin
                        aOutEn = r0;
                        bOutEn = r2;
                        adderOutEn =ADD;
                        regLoad = r0;
                        nextState = 14;
                  end;  ! alpha + gamma -> alpha
                  [14]: begin
                        aOutEn = r2;
                        bOutEn = r2;
                        adderOutEn =ADD;
                        adderFunc = SUB;
                        regLoad = r2;
                        nextState = 15;
                  end;  ! gamma - gamma -> gamma = 0
                  [15]: begin
                        aOutEn = r6;
                        bOutEn = r10;
                        multOutEn =MULT;! multiply
                        regLoad = r1;
                        nextState = 16;
                  end;  ! betaP * Tb -> beta
                  [16]: begin
                        aOutEn = r15;
                        bOutEn = r12;
                        multOutEn =MULT;
                        multCompB = false;
                        regLoad = r3;
                        nextState = 17;
                  end;  ! deltaP * conj(Td) -> delta
!
!
! register map for this cycle:  r0 = alpha,p   r1 = beta,q     r2 = gamma
!                               r3 = delta,r   r4 = sTv        r5 = cTv
!                               r6 = d1        r7 = d2         r8 = sTs
!                               r9 = cTs       r10 = rho       r11 = t
!                               r12 =          r13 = Temp1     r14 = Temp2
!                               r15 = Temp3
!
                  [17]: begin
                        aOutEn = r0;
                        bOutEn = r3;
                        adderOutEn =ADD;
                        regLoad = r13;
                        nextState = 18;
                  end;  ! alpha + delta -> Temp1
                  [18]: begin
                        aOutEn = r13;
                        bOutEn = r13;
                        multOutEn = MULT;
                        multComplexOut = false;
                        multCompB = true;
                        regLoad = r15;
                        nextState = 19;
```

```
end;    ! Temp1*Temp1 -> Temp3
[19]: begin
        aOutEn = r1;
        bOutEn = r1;
        multOutEn = MULT;
        multComplexOut = false;
        multCompB = true;
        regLoad = r14;
        nextState = 20;
end;    ! beta * beta -> Temp2
[20]: begin
        aOutEn = r15;
        bOutEn = r14;
        adderOutEn =ADD;
        adderRealAdd = true;
        regLoad = r14;
        nextState = 21;
end;    ! Temp3 + Temp2 -> Temp2
[21]: begin
        select presentSubState from
        [0]: begin
                aOutEn = r14;
                interpGo = true;
                interpFunc = invsquarert;
                nextState = 21;
                nextSubState = 1;
        end;
        [1]: begin
                aOutEn = r14;
                interpFunc = invsquarert;
                if interpBusy then begin
                        nextSubState = 1;
                end else begin
                        nextSubState = 2;
                end;
                nextState = 21;
        end;
        [2]: begin
                interpFunc = invsquarert;
                interpOutEn = true;
                regLoad = r14;
                nextSubState = 0;
                nextState = 24;
        end;
        endselectselect;
end;    ! 1/SQRT(Temp2) -> Temp2
[24]: begin
        aOutEn = r14;
        bOutEn = r1;
        multOutEn =MULT;
        multRealA = true;
        regLoad = r8;
        nextState = 25;
end;    ! Temp2 * beta -> sTs
```

```
[25]: begin
          aOutEn = r14;
          bOutEn = r13;
          multOutEn =MULT;
          multRealA = true;
          regLoad = r9;
          nextState = 26;
end;      ! Temp2 * Temp1 -> cTs
!

[26]: begin
          aOutEn = r1;
          bOutEn = r8;
          multOutEn =MULT;! multiply
          regLoad = r13;
          nextState = 27;
end;      ! beta * sTs -> Temp1
[27]: begin
          aOutEn = r3;
          bOutEn = r9;
          multOutEn =MULT;! multiply
          regLoad = r14;
          nextState = 28;
end;      ! delta * cTs -> Temp2
[28]: begin
          aOutEn = r13;
          bOutEn = r14;
          adderOutEn =ADD;
          regLoad = r3;
          nextState = 29;
end;      ! Temp1 + Temp2 -> r
[29]: begin
          aOutEn = r8;
          bOutEn = r0;
          multOutEn =MULT;! multiply
          regLoad = r1;
          nextState = 30;
end;      ! sTs * alpha -> q
[30]: begin
          aOutEn = r9;
          bOutEn = r0;
          multOutEn =MULT;! multiply
          regLoad = r0;
          nextState = 31;
end;      ! cTs * alpha -> p
!

[31]: begin
          aOutEn = r3;
          bOutEn = r0;
          adderOutEn =ADD;
          adderFunc = SUB;
          regLoad = r10;
          nextState = 32;
end;      ! r - p -> rho
[32]: begin
```

```
        aOutEn = r1;
        bOutEn = r1;
        adderOutEn =ADD;
        regLoad = r13;
        nextState = 33;
end;    ! q + q -> Temp1
[33]: begin
        aOutEn = r10;
        bOutEn = r10;
        multOutEn = MULT;
        multComplexOut = false;
        multCompB = true;
        regLoad = r14;
        nextState = 34;
end;    ! p * p -> Temp2
[34]: begin
        aOutEn = r13;
        bOutEn = r13;
        multOutEn =MULT;! multiply
        regLoad = r15;
        nextState = 35;
end;    ! Temp1 * Temp2 -> Temp3
[35]: begin
        aOutEn = r13;
        bOutEn = r15;
        adderOutEn =ADD;
        regLoad = r14;
        nextState = 36;
end;    ! Temp1 + Temp3 -> Temp2
[36]: begin
        select presentSubState from
         [0]: begin
                aOutEn = r14;
                interpGo = true;
                nextState = 36;
                nextSubState = 1;
         end;
         [1]: begin
                aOutEn = r14;
                if interpBusy then begin
                        nextSubState = 1;
                end else begin
                        nextSubState = 2;
                end;
                nextState = 36;
         end;
         [2]: begin
                interpOutEn = true;
                regLoad = r14;
                nextSubState = 0;
                nextState = 37;
         end;
        endselectselect;
end;    ! SQRT(Temp2) -> Temp2
```

```
 [39]: begin
           aOutEn = r10;
           bOutEn = r13;
           adderOutEn =ADD;
           adderFunc = SUB;
           regLoad = r11;
           nextState = 40;
 end;      ! rho - Temp1 -> t
 [40]: begin
           aOutEn = r14;
           bOutEn = r15;
           adderOutEn =ADD;
           regLoad = r14;
           nextState = 41;
 end;      ! temp2 + Temp3 -> Temp2
 [41]: begin
           select presentSubState from
            [0]: begin
                     aOutEn = r14;
                     interpGo = true;
                     interpFunc = invsquarert;
                     nextState = 41;
                     nextSubState = 1;
            end;
            [1]: begin
                     aOutEn = r14;
                     interpFunc = invsquarert;
                     if interpBusy then begin
                               nextSubState = 1;
                     end else begin
                               nextSubState = 2;
                     end;
                     nextState = 41;
            end;
            [2]: begin
                     interpFunc = invsquarert;
                     interpOutEn = true;
                     regLoad = r14;
                     nextSubState = 0;
                     nextState = 44;
            end;
           endselectselect;
 end;      ! 1/SQRT(Temp2) -> Temp2
 [44]: begin
           aOutEn = r14;
           bOutEn = r13;
           multOutEn =MULT;
           multRealA = true;
           regLoad = r4;
           nextState = 45;
 end;      ! Temp2 * Temp1 -> cTv
 [45]: begin
           aOutEn = r14;
           bOutEn = r11;
```

```
                        multOutEn =MULT;
                        multRealA = true;
                        regLoad = r5;
                        nextState = 46;
                end; ! Temp2 * t -> sTv
!
                [46]: begin
                        iobusld = horiz;                 ! r5 on bus
                        aOutEn  = r5;           ! horiz <- r5
                        nextState = 47;
                end;    ! sTv -> horiz
                [47]: begin
                        iobusld = vert;         ! r5 on bus
                        aOutEn  = r5;           ! vert <- r5
                        nextState = 48;
                end;    ! sTv -> vert
                [48]: begin
                        nextState = 48;    ! wait here
                end;
             endselectselect; ! presetnState
            end; ! else
           end; ! cyc2
! We are now done with cycle 2.
! on to cycle 3, then haha! the world
!
      [cyc3]: begin
           if incCycle then
                begin
                 if finalCycle then begin
                        nextState = 0;          ! reset state
                        nextCycle = unload;     ! we're done
                 end else begin
                        nextState = 0;          ! reset state
                        nextCycle = pass;       ! exch data and restart
                 end;
           end else begin
            select presentState from
                [0]: begin
                        iostart = handv;        ! send sTvr on vert
                                                ! sTvl on horiz
                        nextState = 1;
                end;
                [1]: if hReady then begin
                        aOutEn = r5;
                        regLoad = horiz;
                        nextState = 2;
                end else begin
                        nextState = 1;
                end;
                [2]: begin
                        aOutEn = r5;
                        regLoad = vert;
                        nextState = 3;
                end;
```

```
[3]: begin
        iostart = handv;            ! send cTvr on vert
                                    ! cTvl on horiz
        nextState = 4;
end;
[4]: if hReady then begin
        aOutEn = r8;
        regLoad = horiz;
        nextState = 6;
end else begin
        nextState = 4;
end;
[6]: begin
        iostart = horiz;            ! send sTs from horiz
        nextState = 7;
end;
[7]: if hReady then begin
        aOutEn = r9;
        regLoad = horiz;
        nextState = 8;
end else begin
        nextState = 7;
end;
[8]: begin
        iostart = horiz;            ! send cTs from horiz
        nextState = 9;
end;
[9]: if hReady then begin
        nextState = 10;
end else begin
        nextState = 9;
end;


[10]: begin
        aOutEn = r5;
        bOutEn = r5;
        multOutEn =MULT;! multiply
        regLoad = r13;
        nextState = 11;
end;    ! cTv * cTv -> Temp1
[11]: begin
        aOutEn = r13;
        bOutEn = r0;
        multOutEn =MULT;! multiply
        regLoad = r6;
        nextState = 12;
end;    ! Temp1 * p -> d1
[12]: begin
        aOutEn = r5;
        bOutEn = r5;
        multOutEn =MULT;! multiply
        regLoad = r14;
        nextState = 13;
```

```
end;     ! cTv * cTv -> Temp2
[13]: begin
          aOutEn = r14;
          bOutEn = r0;
          multOutEn =MULT;! multiply
          regLoad = r7;
          nextState = 14;
end;     ! Temp2 * P -> d2
[14]: begin
          aOutEn = r4;
          bOutEn = r5;
          multOutEn =MULT;! multiply
          regLoad = r15;
          nextState = 15;
end;     ! sTv * cTv -> Temp3
[15]: begin
          aOutEn = r15;
          bOutEn = r15;
          adderOutEn =ADD;
          regLoad = r15;
          nextState = 16;
end;     ! Temp3 + Temp3 -> Temp3
[16]: begin
          aOutEn = r1;
          bOutEn = r15;
          multOutEn =MULT;! multiply
          regLoad = r13;
          nextState = 17;
end;     ! q * Temp3 -> Temp1
[17]: begin
          aOutEn = r6;
          bOutEn = r13;
          adderOutEn =ADD;
          adderFunc = SUB;
          regLoad = r6;
          nextState = 18;
end;     ! d1 - Temp1 -> d1
[18]: begin
          aOutEn = r7;
          bOutEn = r14;
          adderOutEn =ADD;
          regLoad = r7;
          nextState = 19;
end;     ! d2 - Temp2 -> d2
[19]: begin
          aOutEn = r3;
          bOutEn = r13;
          multOutEn =MULT;! multiply
          regLoad = r15;
          nextState = 20;
end;     ! r * Temp1 -> Temp3
[20]: begin
          aOutEn = r3;
          bOutEn = r13;
```

```
                multOutEn =MULT;! multiply
                regLoad = r14;
                nextState = 21;
        end;    ! r * Temp2 -> Temp2
        [21]: begin
                aOutEn = r6;
                bOutEn = r15;
                adderOutEn =ADD;
                regLoad = r0;
                nextState = 22;
        end;    ! d1 + Temp3 -> d1
        [22]: begin
                aOutEn = r7;
                bOutEn = r14;
                adderOutEn =ADD;
                regLoad = r3;
                nextState = 23;
        end;    ! d2 + Temp2 -> d2
        [23]: begin
                aOutEn = r1;
                bOutEn = r1;
                adderOutEn =ADD;
                adderFunc = SUB;
                regLoad = r1;
                nextState = 24;
        end;    ! 0 -> r1
        [24]: begin
                aOutEn = r2;
                bOutEn = r2;
                adderOutEn =ADD;
                adderFunc = SUB;
                regLoad = r2;
                nextState = 25;
        end;    ! 0 -> r2




!
!
! now, get ready for data exchange, as per BLVL— algorithm interchange
!
        [25]: begin
                if topEdge then begin
                 if leftEdge then begin
                   aOutEn = r0;  ! alpha
                 end else begin
                   aOutEn = r1;  ! beta
                 end;
                end else begin
                 if leftEdge then begin
                   aOutEn = r2;  ! gammaP
                 end else begin
```

```
            aOutEn = r3;  ! deltaP
          end;
        end;
        iobusld = alpha;! alphaOut to io
        nextState = 26;
end;
[26]: begin
        if topEdge then begin
          if leftEdge then begin
            aOutEn = r1;  ! betaP
          end else begin
            aOutEn = r0;  ! alphaP
          end;
        end else begin
          if leftEdge then begin
            aOutEn = r3;  ! deltaP
          end else begin
            aOutEn = r2;  ! gammaP
          end;
        end;
        iobusld = beta;! betaOut to io
        nextState = 27;
end;
[27]: begin
        if topEdge then begin
          if leftEdge then begin
            aOutEn = r2;  ! gammaP
          end else begin
            aOutEn = r3;  ! deltaP
          end;
        end else begin
          if leftEdge then begin
            aOutEn = r0;  ! alphaP
          end else begin
            aOutEn = r1;  ! betaP
          end;
        end;
        iobusld = gamma;! gammaOut to io
        nextState = 28;
end;
[28]: begin
        if topEdge then begin
          if leftEdge then begin
            aOutEn = r3;  ! deltaP
          end else begin
            aOutEn = r2;  ! gammaP
          end;
        end else begin
          if leftEdge then begin
            aOutEn = r1;  ! betaP
          end else begin
            aOutEn = r0;  ! alphaP
          end;
        end;
```

```
                    iobusld = delta;! deltaOut to io
                    nextState = 29;
                end;
                [29]: begin
                    nextState = 29;   ! wait here until cycle inc
                end;
            endselectselect; ! presentState
            end; ! else
        end; ! cyc3
!
! end of computation cycles; alpha—delta now updated.
! now they must be exchanged with adjacent nodes
!
        [pass]: begin
            ioSel = PassInputs;
            if incCycle then
                begin
                    nextState = 0;          ! reset state
                    nextCycle = cycl;       ! go back and compute
                end
            else begin
            select presentState from
                [0]: begin
                    iostart = abgd;         ! start sending all
                    nextState = 1;
                end;
                [1]: if alphaReady then begin
                    nextState = 2;
                end else begin
                    nextState = 1;
                end;
                [2]: begin
                    nextState = 1;          ! wait for next cycle
                end;
            endselectselect; ! presentState
            end; ! else
        end; ! pass
!
! end of data intercahnge
!
        [unload]: begin
            if incCycle then
                begin
                    nextState = 0;          ! reset state
                    nextCycle = idle;       ! done unloading
                end
            else begin
             select presentState from
                [0]: begin
                  if nextData then begin
                    iostart = abgd;         ! start r/w in col
                    nextState = 1;          ! proceed
                    end
                 else begin
```

```
                        nextState = 0;            ! still waitint
                        end;
                    end;
                  [1]: begin
                      if alphaReady then begin
                        nextState = 0;            ! done w/this one
                      end else begin
                        nextState = 1;            ! not done
                        end;
                      end;
                endselectselect; ! presentState
              end; !else
      end; !unload
  endselectselect; ! presentCycle
  if reset then begin
      nextState = 0;          ! reset state
      nextSubState = 0;       ! no substate
      nextCycle = idle;       ! idle cycle
      iobusoe = none;         ! nobody on the bus
      iobusld = none;         ! nobody loading off bus
      iostart = none;         ! nobody starting io
      adderOutEn = none;      ! adder not on bus
      adderFunc = ADD;        ! adder adds
      multOutEn = none;       ! mult not on bus either
      multComplexOut = true;  ! complex multiply by default
      multCompB = false;      ! do not complement b in multiplies
      adderRealAdd =false;    ! complex adds
      interpGo = false;       ! no interp
      interpOutEn = false;    ! no interp on bus
      interpFunc = squarert;  ! square root by default
      aOutEn = none;          ! no regs on abus
      bOutEn = none;          ! no regs on bbus
      regLoad = none;         ! no regs loading
  end;
endroutineroutine;
endmodelmodel;
```

A.2.2 interpcntl.bds

This file contains the behavioral description for control of the fast interpolation cell presented in chapter 5. It requires six clock cycles to complete. This file is used to generate the PLA grid for a finite state machine.

```
model interpcntl
OUT<17:0> = in<4:0>;
        synonym presentState<2:0> = in <2:0>,
                go = in<3>,
                reset = in<4>,
                nextState<2:0> = out<2:0>,
        busy = out<3>,
        inc = out<4>,
        multASel<1:0> = out<6:5>,
        multBSel<1:0> = out<8:7>,
        adderASel = out<9>,
        adderBSel<1:0> = out<11:10>,
        adderSub = out<12>,
        loadAccum = out<13>,
        loadPQ = out<14>,
        loadTemp = out<15>,
        constSel<1:0> = out<17:16>;

constant A=0,
        B=1,
        C=3,
        ISBUSY = 1,
        NOTBUSY = 0,
        INCREMENT = 1,
        DONTINCREMENT = 0,
        ADD = 0,
        SUB = 1,
        LOAD = 1,
        KEEP = 0,
        ZERO = 0,
        ONE = 1,
        TWO = 2,
        THREE = 3;

        ! ADDER A MUX:
        !       A = P (lower width-M bits of input)
        !       B = output of multiplier
        !
        ! ADDER B MUX:
        !       A = Accumulation register
        !       B = F rom look up table
        !       C = Constant (selectable 0,1,2,3)
        ! MULTIPLIER A MUX:
        !       A = output of adder
        !       B = Temporary register
        !       C = G rom look up table
        ! MULTIPLIER B MUX:
        !       A = P
        !       B = delta rom look up table
        !       C = PQ register
        !

routine main;
        ! defaults...
        nextState = 0;          ! reset by default
        busy = NOTBUSY;         ! not doing anything
        inc = DONTINCREMENT; ! dont inc rom address
        multASel = A;           ! just so its something
        multBSel = A;           ! again
        adderASel = A;          ! again
```

```
adderBSel = A;           ! again
adderSub = ADD;          ! add by default
loadAccum = KEEP;        ! dont load
loadPQ = KEEP;           ! ditto
loadTemp = KEEP;         ! ditto
constSel = ZERO;         ! just so its something

! now the real work...
select presentState from
        [0]: if go then
                begin
                        nextState = 1;          ! start
                        busy = ISBUSY;          ! go!
                        constSel = ONE;         ! (1
                        adderBSel = C;          ! choose it
                        adderSub = SUB;         ! minus
                        adderASel=A;            ! P)
                        multASel = A;           ! times
                        multBsel = A;           ! P
                        loadPQ = LOAD;          ! and save
                end;
        [1]:    begin
                        nextState = 2;          ! keep going
                        busy = ISBUSY;          ! running
                        constSel = TWO;         ! (2
                        adderBSel = C;          ! choose it
                        adderSub = SUB;         ! minus
                        adderASel = A;          ! P)
                        multASel = A;           ! times
                        multBSel = C;           ! PQ
                        loadTemp = LOAD;        ! and save
                end;
        [2]:    begin
                        nextState = 3;          ! keep going
                        busy = ISBUSY;          ! running
                        multASel = B;           ! pq(2-p)
                        multBSel = B;           ! times delta
                        adderASel = B;          ! plus
                        adderBSel = B;          ! fi
                        loadAccum = LOAD;       ! and save
                end;
        [3]:    begin
                        nextState = 4;          ! keep going
                        busy = ISBUSY;          ! running
                        constSel = ONE;         ! (1
                        adderBSel = C;          ! choose it
                        adderASel = A;          ! plus P)
                        multASel = A;           ! times
                        multBSel = C;           ! PQ
                        loadTemp = LOAD;        ! and save
                        inc = INCREMENT;        ! increment rom
                end;
        [4]:    begin
                        nextState = 5;          ! keep going
```

```
                    busy = ISBUSY;              ! running
                    multASel = B;               ! pq(1+p)
                    multBSel = B;               ! * nextdelta
                    adderASel = B;              ! plus
                    adderBSel = A;              ! accum
                    loadAccum = LOAD;           ! and save
                end;
        [5]:    begin
                    nextState = 6;              ! keep going
                    busy = ISBUSY;              ! running
                    multASel = C;               ! g
                    multBSel = A;               ! times p
                    adderASel = B;              ! plus
                    adderBSel = A;              ! accum
                    loadAccum = LOAD;           ! and save
                end;
        [6]: nextState = 0;                     ! unused
        [7]: nextState = 0;                     ! unused
    endselectselect;
    if reset then
        begin
            nextState = 0;              ! reset by default
            busy = NOTBUSY;             ! not doing anything
            inc = DONTINCREMENT;        ! dont inc rom address
            multASel = A;               ! just so its something
            multBSel = A;               ! again
            adderASel = A;              ! again
            adderBSel = A;              ! again
            adderSub = ADD;             ! add by default
            loadAccum = KEEP;           ! dont load
            loadPQ = KEEP;              ! ditto
            loadTemp = KEEP;            ! ditto
            constSel = ZERO;            ! just so its something
        end;
  endroutineroutine;
endmodelmodel;
```

A.2.3 offdiagctl.bds

This file contains the behavioral description for performing the calculations of the off-diagonal processor as presented in chapter 2 for computation of the SVD on a systolic processor array as a three cycle computation. This file is used to generate the PLA grid for a finite state machine.

```
          model offdiagctl
OUT<78:0> = IN<20:0>;
          synonym iobusoe<5:0>   = out<5:0>,
                  iobusld<5:0>   = out<11:6>,
                  iostart<5:0>   = out<17:12>,
                  ioSel          = out<18>,
                  adderFunc      = out<19>,
                  adderOutEn     = out<20>,
                  multOutEn      = out<21>,
                  aOutEn<15:0>   = out<37:22>,
                  bOutEn<15:0>   = out<53:38>,
                  regLoad<15:0>  = out<69:54>,
                  nextCycle<2:0> = out<72:70>,
                  nextState<5:0> = out<78:73>,
                  reset          = in<0>,
                  alphaReady     = in<1>,
                  betaReady      = in<2>,
                  gammaReady     = in<3>,
                  deltaReady     = in<4>,
                  hReady         = in<5>,
                  vReady         = in<6>,
                  incCycle       = in<7>,
                  finalCycle     = in<8>,
                  nextData       = in<9>,
                  leftEdge       = in<10>,
                  topEdge        = in<11>,
                  presentCycle<2:0>= in<14:12>,
                  presentState<5:0>          = in<20:15>;

          constant         alpha  = 1,
                           beta   = 2,
                           gamma  = 4,
                           delta  = 8,
                           abgd   = 15,
                           horiz  = 16,
                           vert   = 32,
                           handv  = 48,
                           r0     = 1,
                           r1     = 2,
                           r2     = 4,
                           r3     = 8,
                           r4     = 16,
                           r5     = 32,
                           r6     = 64,
                           r7     = 128,
                           r8     = 256,
                           r9     = 512,
                           r10    = 1024,
                           r11    = 2048,
                           r12    = 4096,
                           r13    = 8192,
                    r14        = 16384,
                    r15        = 32768,
                    none       = 0,
                    add        = 1,
                    sub        = 0,
                    mult       = 1,
                    PassInputs = 0,
                    LoadInput  = 1,
                    idle       = 0,
                    load       = 1,
                    cyc1       = 2,
                    cyc2       = 3,
                    cyc3       = 4,
                    pass       = 5,
                    unload     = 6;


routine main;
        ! defaults
        nextState = 0;              ! reset by default
        iobusoe = none;             ! nobody on the bus
        iobusld = none;             ! nobody loading off bus
        iostart = none;             ! nobody starting io
        ioSel = PassInputs;         ! use internal connections
        adderOutEn = none;          ! adder not on bus
        adderFunc = add;            ! adder adds
        multOutEn = none;           ! mult not on bus either
        aOutEn = none;              ! no regs on abus
        bOutEn = none;              ! no regs on bbus
        regLoad = none;             ! no regs loading
        nextCycle = presentCycle;   ! waiting to go

        ! now lets do something...
select presentCycle from
        [idle]: if incCycle then
                    begin
                            nextState = 0;          ! reset state
                            nextCycle = load;       ! first load
                    end;
        [load]: begin
                ioSel = LoadInput;
                if incCycle then
                    begin
                            nextState = 0;          ! reset state
                            nextCycle = cyc1;       ! done loading
                    end
               else begin
                select presentState from
                    [0]: begin
                        if nextData then begin
                            iostart = abgd;         ! start r/w in col
                            nextState = 1;          ! proceed
                            end
                      else begin
```

```
                        nextState = 0;            ! still waitint
                        end;
                    end;
                  [1]: begin
                      if alphaReady then begin
                        nextState = 0;            ! done w/this one
                       end else begin
                        nextState = 1;            ! not done
                        end;
                      end;
              endselectselect; ! presentState
             end; !else
           end; !cycle
[cyc1]: begin
           if incCycle then
                  begin
                        nextState = 0;            ! reset state
                        nextCycle = cyc2;         ! next cycle
                  end
           else begin
            select presentState from
                  [0]: begin
                        iobusoe = alpha;          ! alpha on bus
                        regLoad = r0;             ! r0 <- alpha
                        nextState = 1;
                  end;
                  [1]: begin
                        iobusoe = beta;           ! beta on bus
                        regLoad = r1;             ! r1 <- beta
                        nextState = 2;
                  end;
                  [2]: begin
                        iobusoe = gamma;          ! gamma on bus
                        regLoad = r2;             ! r2 <- gamma
                        nextState = 3;
                  end;
                  [3]: begin
                        iobusoe = delta;          ! delta on bus
                        regLoad = r3;             ! r3 <- delta
                        nextState = 4;
                  end;
                  [4]: begin
                        nextState = 4;            ! stay here until
                  end;                            ! next cycle
            endselectselect; ! presentState
           end; ! else
end; ! cyc1
[cyc2]: begin
           if incCycle then
                  begin
                        nextState = 0;            ! reset state
                        nextCycle = cyc3;         ! next cycle
                  end
           else begin
```

```
select presentState from
        [0]: begin
                iostart = handv;            ! get Tg from vert,
                                            ! sTg from horiz
                nextState = 1;
        end;
        [1]: if hReady then begin
                iobusoe = horiz;            ! sTg on bus
                regLoad = r4;               ! r4 <- sTg
                nextState = 2;
        end else begin
                nextState = 1;              ! waiting for io
        end;
        [2]: begin
                iobusoe = vert;             ! Tg on bus
                regLoad = r8;               ! r8 <- Tg
                nextState = 3;
        end;
        [3]: begin
                iostart = horiz;            ! get cTg
                nextState = 4;
        end;
        [4]: if hReady then begin
                iobusoe = horiz;            ! cTg on bus
                regLoad = r5;               ! r5 <- cTg
                nextState = 5;
        end else begin
                nextState = 4;              ! waiting for io
        end;
        [5]: begin
                iostart = horiz;            ! get Ta
                nextState = 6;
        end;
        [6]: if hReady then begin
                iobusoe = horiz;            ! Ta on bus
                regLoad = r6;               ! r6 <- Ta
                nextState = 7;
        end else begin
                nextState = 6;              ! waiting for io
        end;
        [7]: begin
                iostart = horiz;            ! get Tb
                nextState = 8;
        end;
        [8]: if hReady then begin
                iobusoe = horiz;            ! Tb on bus
                regLoad = r7;               ! r7 <- Tb
                nextState = 9;
        end else begin
                nextState = 8;              ! waiting for io
        end;
        [9]: begin
                iostart = horiz;            ! get Td
                nextState = 10;
```

```
                        end;
                        [10]: if hReady then begin
                                iobusoe = horiz;           ! Td on bus
                                regLoad = r9;              ! r9 <- Td
                                nextState = 11;
                        end else begin
                                nextState = 10;            ! waiting for io
                        end;
! register map for this cycle:  r0 = alpha      r1 = beta       r2 = gamma
!                               r3 = delta      r4 = sTg        r5 = cTg
!                               r6 = Ta         r7 = Tb         r8 = Tg
!                               r9 = Td         r10 = Temp1     r11 = Temp2
!                               r12 = Temp3     r13 = alphaP    r14 = betaP
!                               r15 = unused
!               Temp[123], alphaP, betaP are undefined at this point.
! so now we have all four data values (alpha-delta), and all the rot. angles
! we can start applying them.
!
! first the right sided rotation
                        [11]: begin
                                aOutEn = r1;      ! beta
                                bOutEn = r8;      ! Tg
                                multOutEn =mult;! multiply
                                regLoad = r1;     ! beta = beta * Tg
                                nextState = 12;
                        end;
                        [12]: begin
                                aOutEn = r3;      ! delta
                                bOutEn = r8;      ! Tg
                                multOutEn =mult;! multiply
                                regLoad = r3;     ! delta = delta * Tg
                                nextState = 13;
                        end;
                        [13]: begin
                                aOutEn = r6;      ! Ta
                                bOutEn = r5;      ! cTg
                                multOutEn =mult;! multiply
                                regLoad = r10;    ! Temp1 = Ta * cTg
                                nextState = 14;
                        end;
                        [14]: begin
                                aOutEn = r7;      ! Tb
                                bOutEn = r4;      ! sTg
                                multOutEn =mult;! multiply
                                regLoad = r11;    ! Temp2 = Tb * sTg
                                nextState = 15;
                        end;
                        [15]: begin
                                aOutEn = r0;      ! alpha
                                bOutEn = r10;     ! Temp1
                                multOutEn =mult;! multiply
                                regLoad = r13;    ! alphaP = alpha * Temp1
                                nextState = 15;
                        end;
```

```
[15]: begin
        aOutEn = r1;      ! beta
        bOutEn = r11;     ! Temp2
        multOutEn =mult;! multiply
        regLoad = r14;    ! betaP = beta * Temp2
        nextState = 16;
end;
[16]: begin
        aOutEn = r2;      ! gamma
        bOutEn = r11;     ! Temp2
        multOutEn =mult;! multiply
        regLoad = r10;    ! Temp1 = Temp2*gamma
        nextState = 17;
end;
[17]: begin
        aOutEn = r3;      ! delta
        bOutEn = r11;     ! Temp2
        multOutEn =mult;! multiply
        regLoad = r12;    ! Temp3 = delta * Temp2
        nextState = 18;
end;
[18]: begin
        aOutEn = r13;     ! alphaP
        bOutEn = r10;     ! temp1
        adderOutEn =add;! plus
        regLoad = r13;    ! alphaP = alphaP + temp1
        nextState = 19;
end;
[19]: begin
        aOutEn = r14;     ! betaP
        bOutEn = r12;     ! Temp3
        adderOutEn =add;! plus
        regLoad = r14;    ! betaP = betaP + temp3
        nextState = 20;
end;
[20]: begin
        aOutEn = r9;      ! Td
        bOutEn = r7;      ! Tb
        multOutEn =mult;! multiply
        regLoad = r10;    ! Temp1 = Td * Tb
        nextState = 21;
end;
[21]: begin
        aOutEn = r10;     ! Temp1
        bOutEn = r5;      ! cTg
        multOutEn =mult;! multiply
        regLoad = r10;    ! Temp1 = Temp1 * cTg
        nextState = 22;
end;
[22]: begin
        aOutEn = r9;      ! Td
        bOutEn = r6;      ! Ta
        multOutEn =mult;! multiply
        regLoad = r11;    ! Temp2 = Td * Ta
```

```
            nextState = 23;
    end;
    [23]: begin
            aOutEn = r11;     ! Temp2
            bOutEn = r4;      ! sTg
            multOutEn =mult;! multiply
            regLoad = r11;    ! Temp2 = Temp2 * sTg
            nextState = 24;
    end;
    [25]: begin
            aOutEn = r2;      ! gamma
            bOutEn = r10;     ! Temp1
            multOutEn =mult;! multiply
            regLoad = r2;     ! gamma = gamma * Temp1
            nextState = 26;
    end;
    [26]: begin
            aOutEn = r3;      ! delta
            bOutEn = r10;     ! Temp1
            multOutEn =mult;! multiply
            regLoad = r3;     ! delta = delta * Temp1
            nextState = 27;
    end;
    [27]: begin
            aOutEn = r0;      ! alpha
            bOutEn = r11;     ! Temp2
            multOutEn =mult;! multiply
            regLoad = r10;    ! Temp1 = alpha * Temp2
            nextState = 28;
    end;
    [28]: begin
            aOutEn = r2;      ! gamma
            bOutEn = r10;     ! Temp1
            adderOutEn =add;!
            adderFunc = sub;! minus
            regLoad = r2;     ! gamma = gamma − Temp1
            nextState = 29;
    end;
    [29]: begin
            aOutEn = r1;      ! beta
            bOutEn = r11;     ! Temp2
            multOutEn =mult;! multiply
            regLoad = r12;    ! Temp3
            nextState = 30;
    end;
    [30]: begin
            aOutEn = r3;      ! delta
            bOutEn = r12;     ! Temp3
            adderOutEn =add;!
            adderFunc = sub;! minus
            regLoad = r3;     ! delta = delta − Temp3
            nextState = 31;
    end;
    [31]: begin
```

```
                        aOutEn = r13;     ! alphaP
                        iobusld = alpha;! temp holding place
                        nextState = 32;
                end;
                [32]: begin
                        iobusoe = alpha;! alphaP
                        regLoad = r0;     ! alpha = alphaP
                        nextState = 33;
                end;
                [33]: begin
                        aOutEn = r14;     ! betaP
                        iobusld = beta;   ! temp holding place
                        nextState = 34;
                end;
                [34]: begin
                        iobusoe = beta;   ! betaP
                        regLoad = r1;     ! beta = betaP
                        nextState = 35;
                end;
                [35]: begin
                        nextState = 35;   ! hold
                end;
            endselectselect; ! presetnState
          end; ! else
         end; ! cyc2
! We are now done with cycle 2.
! on to cycle 3, then haha! the world
!
    [cyc3]: begin
        if incCycle then
                begin
                 if finalCycle then begin
                        nextState = 0;              ! reset state
                        nextCycle = unload;         ! we're done
                 end else begin
                        nextState = 0;              ! reset state
                        nextCycle = pass;           ! exch data and restart
                 end;
        end else begin
         select presentState from
                [0]: begin
                        iostart = handv;            ! get sTvr from vert
                                                    ! sTvl from horiz
                        nextState = 1;
                end;
                [1]: if hReady then begin
                        iobusoe = horiz;            ! sTvl on bus
                        regLoad = r4;               ! r4 <- sTvl
                        nextState = 2;
                end else begin
                        nextState = 1;
                end;
                [2]: begin
                        iobusoe = vert;             ! sTvr on bus
                        regLoad = r6;           ! r6 <- sTvr
                        nextState = 3;
                end;
                [3]: begin
                        iostart = handv;        ! get cTvr from vert
                                                ! cTvl from horiz
                        nextState = 4;
                end;
                [4]: if hReady then begin
                        iobusoe = horiz;        ! cTvl on bus
                        regLoad = r5;           ! r5 <- cTvl
                        nextState = 5;
                end else begin
                        nextState = 4;
                end;
                [5]: begin
                        iobusoe = vert;         ! cTvr on bus
                        regLoad = r7;           ! r7 <- cTvr
                        nextState = 6;
                end;
                [6]: begin
                        iostart = horiz;        ! get sTs from horiz
                        nextState = 7;
                end;
                [7]: if hReady then begin
                        iobusoe = horiz;        ! sTs on bus
                        regLoad = r8;           ! r8 <- sTs
                        nextState = 8;
                end else begin
                        nextState = 7;
                end;
                [8]: begin
                        iostart = horiz;        ! get sTs from horiz
                        nextState = 9;
                end;
                [9]: if hReady then begin
                        iobusoe = horiz;        ! cTs on bus
                        regLoad = r9;           ! r9 <- cTs
                        nextState = 10;
                end else begin
                        nextState = 9;
                end;
! register map for this cycle:  r0 = alpha      r1 = beta       r2 = gamma
!                               r3 = delta      r4 = sTvl       r5 = cTvl
!                               r6 = sTvr       r7 = cTvr       r8 = sTs
!                               r9 = cTs        r10 = Temp1     r11 = Temp2
!                               r12 = alphaP    r13 = betaP     r14 = gammaP
!                               r15 = deltaP
!       Temp[12], alphaP, betaP, deltaP, gammaP are undefined at this point.
! Now we have all the rotation angles and alpha-delta.  Lets begin.
!
                [10]: begin
                        aOutEn = r0;    ! alpha
                        bOutEn = r7;    ! cTvr
```

```
        multOutEn =mult;! multiply
        regLoad = r12;   ! alphaP = alpha * cTvr
        nextState = 11;
end;
[11]: begin
        aOutEn = r1;     ! beta
        bOutEn = r6;     ! sTvr
        multOutEn =mult;! multiply
        regLoad = r10;   ! Temp1 = beta * sTvr
        nextState = 12;
end;
[12]: begin
        aOutEn = r12;    ! alphaP
        bOutEn = r10;    ! Temp1
        adderOutEn =add;!
        adderFunc = sub;! minus
        regLoad = r12;   ! alphaP = alphaP − Temp1
        nextState = 13;
end;
[13]: begin
        aOutEn = r1;     ! beta
        bOutEn = r7;     ! cTvr
        multOutEn =mult;! multiply
        regLoad = r13;   ! betaP = beta * cTvr
        nextState =14;
end;
[14]: begin
        aOutEn = r0;     ! alpha
        bOutEn = r6;     ! sTvr
        multOutEn =mult;! multiply
        regLoad = r11;   ! Temp2 = alpha * sTvr
        nextState = 15;
end;
[15]: begin
        aOutEn = r13;    ! betaP
        bOutEn = r11;    ! Temp2
        adderOutEn =add;!
        regLoad = r13;   ! betaP = betaP + Temp2
        nextState = 16;
end;
[16]: begin
        aOutEn = r2;     ! gamma
        bOutEn = r7;     ! cTvr
        multOutEn =mult;! multiply
        regLoad = r14;   ! gammaP = gamma * cTvr
        nextState = 17;
end;
[17]: begin
        aOutEn = r3;     ! delta
        bOutEn = r6;     ! sTvr
        multOutEn =mult;! multiply
        regLoad = r10;   ! Temp1 = delta * sTvr
        nextState = 18;
end;
```

```
                [18]: begin
                        aOutEn = r14;     ! gammaP
                        bOutEn = r10;     ! Temp1
                        adderOutEn =add;!
                        adderFunc = sub;! minus
                        regLoad = r14;    ! gammaP = gammaP − Temp1
                        nextState = 19;
                end;
                [19]: begin
                        aOutEn = r3;      ! delta
                        bOutEn = r7;      ! cTvr
                        multOutEn =mult;! multiply
                        regLoad = r15;    ! deltaP = delta * cTvr
                        nextState =20;
                end;
                [20]: begin
                        aOutEn = r2;      ! gamma
                        bOutEn = r6;      ! sTvr
                        multOutEn =mult;! multiply
                        regLoad = r11;    ! Temp2 = gamma * sTvr
                        nextState = 21;
                end;
                [21]: begin
                        aOutEn = r15;     ! deltaP
                        bOutEn = r11;     ! Temp2
                        adderOutEn =add;!
                        regLoad = r15;    ! deltaP = deltaP + Temp2
                        nextState = 22;
                end;
!
! right sided rotation by ThetaSVDright complete.
!
                [22]: begin
                        aOutEn = r12;     ! alphaP
                        bOutEn = r9;      ! cTs
                        multOutEn =mult;! multiply
                        regLoad = r0;     ! alpha = alphaP * cTs
                        nextState = 23;
                end;
                [23]: begin
                        aOutEn = r14;     ! gammaP
                        bOutEn = r8;      ! sTs
                        multOutEn =mult;! multiply
                        regLoad = r10;    ! Temp1 = gammaP * sTs
                        nextState = 24;
                end;
                [24]: begin
                        aOutEn = r0;      ! alpha
                        bOutEn = r10;     ! Temp1
                        adderOutEn =add;!
                        adderFunc = sub;! minus
                        regLoad = r0;     ! alpha = alpha − Temp1
                        nextState = 25;
                end;
```

```
[25]: begin
        aOutEn = r13;    ! betaP
        bOutEn = r9;     ! cTs
        multOutEn =mult;! multiply
        regLoad = r13;   ! betaP = betaP * cTs
        nextState =26;
end;
[26]: begin
        aOutEn = r15;    ! deltaP
        bOutEn = r8;     ! sTs
        multOutEn =mult;! multiply
        regLoad = r11;   ! Temp2 = deltaP * sTs
        nextState = 27;
end;
[27]: begin
        aOutEn = r1;     ! beta
        bOutEn = r11;    ! Temp2
        adderOutEn =add;!
        adderFunc = sub;! minus
        regLoad = r1;    ! beta = beta = Temp2
        nextState = 28;
end;
[28]: begin
        aOutEn = r14;    ! gammaP
        bOutEn = r9;     ! cTs
        multOutEn =mult;! multiply
        regLoad = r2;    ! gamma = gammaP * cTs
        nextState = 29;
end;
[29]: begin
        aOutEn = r12;    ! alphaP
        bOutEn = r8;     ! sTs
        multOutEn =mult;! multiply
        regLoad = r10;   ! Temp1 = alphaP * sTs
        nextState = 30;
end;
[30]: begin
        aOutEn = r2;     ! gamma
        bOutEn = r10;    ! Temp1
        adderOutEn =add;!
        regLoad = r2;    ! gamma = gamma + Temp1
        nextState = 31;
end;
[31]: begin
        aOutEn = r15;    ! deltaP
        bOutEn = r9;     ! cTs
        multOutEn =mult;! multiply
        regLoad = r3;    ! delta = deltaP * cTs
        nextState =32;
end;
[32]: begin
        aOutEn = r13;    ! betaP
        bOutEn = r8;     ! sTs
        multOutEn =mult;! multiply
```

```
                        regLoad = r11;     ! Temp2 = betaP * sTs
                        nextState = 33;
                end;
                [33]: begin
                        aOutEn = r3;       ! delta
                        bOutEn = r11;      ! Temp2
                        adderOutEn =add;!
                        regLoad = r3;      ! delta = delta + Temp2
                        nextState = 34;
                end;
!
! left sided rotation by thetaSym complete.
!
                [34]: begin
                        aOutEn = r0;       ! alpha
                        bOutEn = r5;       ! cTvl
                        multOutEn =mult;! multiply
                        regLoad = r12;     ! alphaP = alpha * cTvl
                        nextState = 35;
                end;
                [35]: begin
                        aOutEn = r2;       ! gamma
                        bOutEn = r4;       ! sTvl
                        multOutEn =mult;! multiply
                        regLoad = r10;     ! Temp1 = gamma * sTvl
                        nextState = 36;
                end;
                [36]: begin
                        aOutEn = r12;      ! alphaP
                        bOutEn = r10;      ! Temp1
                        adderOutEn =add;!
                        adderFunc = sub;! minus
                        regLoad = r12;     ! alphaP = alphaP - Temp1
                        nextState = 37;
                end;
                [37]: begin
                        aOutEn = r1;       ! beta
                        bOutEn = r9;       ! cTvl
                        multOutEn =mult;! multiply
                        regLoad = r1;      ! beta = beta * cTvl
                        nextState =38;
                end;
                [38]: begin
                        aOutEn = r3;       ! delta
                        bOutEn = r8;       ! sTvl
                        multOutEn =mult;! multiply
                        regLoad = r11;     ! Temp2 = delta * sTvl
                        nextState = 39;
                end;
                [39]: begin
                        aOutEn = r13;      ! betaP
                        bOutEn = r11;      ! Temp2
                        adderOutEn =add;!
                        adderFunc = sub;! minus
                regLoad = r13;    ! betaP = betaP - Temp2
                nextState = 40;
        end;
        [40]: begin
                aOutEn = r2;      ! gamma
                bOutEn = r5;      ! cTvl
                multOutEn =mult;! multiply
                regLoad = r14;    ! gammaP = gamma * cTvl
                nextState = 41;
        end;
        [41]: begin
                aOutEn = r0;      ! alpha
                bOutEn = r4;      ! sTvl
                multOutEn =mult;! multiply
                regLoad = r10;    ! Temp1 = alpha * sTs
                nextState = 42;
        end;
        [42]: begin
                aOutEn = r14;     ! gammaP
                bOutEn = r10;     ! Temp1
                adderOutEn =add;!
                regLoad = r14;    ! gammaP = gammaP + Temp1
                nextState = 43;
        end;
        [43]: begin
                aOutEn = r3;      ! delta
                bOutEn = r5;      ! cTvl
                multOutEn =mult;! multiply
                regLoad = r15;    ! deltaP = delta * cTvl
                nextState =44;
        end;
        [44]: begin
                aOutEn = r1;      ! beta
                bOutEn = r4;      ! sTvl
                multOutEn =mult;! multiply
                regLoad = r11;    ! Temp2 = beta * sTvl
                nextState = 45;
        end;
        [45]: begin
                aOutEn = r15;     ! deltaP
                bOutEn = r11;     ! Temp2
                adderOutEn =add;!
                regLoad = r15;    ! deltaP = deltaP + Temp2
                nextState = 46;
        end;
!
! new values for alpha, beta, gamma, delta in r12-r15.
!
! now, get ready for data exchange, as per BLVL- algorithm interchange
!
        [46]: begin
                if topEdge then begin
                 if leftEdge then begin
                   aOutEn = r12;            ! alphaP
```

```
             end else begin
               aOutEn = r13;            ! betaP
             end;
            end else begin
             if leftEdge then begin
               aOutEn = r14;            ! gammaP
             end else begin
               aOutEn = r15;            ! deltaP
             end;
            end;
            iobusld = alpha;! alphaOut to io
            nextState = 47;
end;
[47]: begin
            if topEdge then begin
             if leftEdge then begin
               aOutEn = r13;            ! betaP
             end else begin
               aOutEn = r12;            ! alphaP
             end;
            end else begin
             if leftEdge then begin
               aOutEn = r15;            ! deltaP
             end else begin
               aOutEn = r14;            ! gammaP
             end;
            end;
            iobusld = beta;! betaOut to io
            nextState = 48;
end;
[48]: begin
            if topEdge then begin
             if leftEdge then begin
               aOutEn = r14;            ! gammaP
             end else begin
               aOutEn = r15;            ! deltaP
             end;
            end else begin
             if leftEdge then begin
               aOutEn = r12;            ! alphaP
             end else begin
               aOutEn = r13;            ! betaP
             end;
            end;
            iobusld = gamma;! gammaOut to io
            nextState = 49;
end;
[49]: begin
            if topEdge then begin
             if leftEdge then begin
               aOutEn = r15;            ! deltaP
             end else begin
               aOutEn = r14;            ! gammaP
             end;
```

```
                    end else begin
                     if leftEdge then begin
                       aOutEn = r13;           ! betaP
                     end else begin
                       aOutEn = r12;           ! alphaP
                     end;
                    end;
                    iobusld = delta;! deltaOut to io
                    nextState = 50;
                end;
                [50]: begin
                    nextState = 50;    ! wait here until cycle inc
                end;
             endselectselect; ! presentState
            end; ! else
        end; ! cyc3
!
! end of computation cycles; alpha-delta now updated.
! now they must be exchanged with adjacent nodes
!
        [pass]: begin
            ioSel = PassInputs;
            if incCycle then
                begin
                    nextState = 0;             ! reset state
                    nextCycle = cycl;          ! go back and compute
                end
            else begin
             select presentState from
                [0]: begin
                    iostart = abgd;            ! start sending all
                    nextState = 1;
                end;
                [1]: if alphaReady then begin
                    nextState = 2;
                end else begin
                    nextState = 1;
                end;
                [2]: begin
                    nextState = 1;             ! wait for next cycle
                end;
             endselectselect; ! presentState
            end; ! else
        end; ! pass
!
! end of data intercahnge
!
        [unload]: begin
             if incCycle then
                begin
                    nextState = 0;             ! reset state
                    nextCycle = idle;          ! done unloading
                end
            else begin
```

```
                    select presentState from
                            [0]: begin
                                 if nextData then begin
                                      iostart = abgd;          ! start r/w in col
                                      nextState = 1;           ! proceed
                                      end
                                 else begin
                                      nextState = 0;           ! still waitint
                                      end;
                               end;
                            [1]: begin
                                 if alphaReady then begin
                                      nextState = 0;           ! done w/this one
                                   end else begin
                                      nextState = 1;           ! not done
                                      end;
                                 end;
                     endselectselect; ! presentState
                    end; !else
          end; !unload
 endselectselect; ! presentCycle
 if reset then begin
          nextState = 0;           ! reset state
          nextCycle = idle;        ! idle cycle
          iobusoe = none;          ! nobody on the bus
          iobusld = none;          ! nobody loading off bus
          iostart = none;          ! nobody starting io
          adderOutEn = none;       ! adder not on bus
          adderFunc = add;         ! adder adds
          multOutEn = none;        ! mult not on bus either
          aOutEn = none;           ! no regs on abus
          bOutEn = none;           ! no regs on bbus
          regLoad = none;          ! no regs loading
 end;
endroutineroutine;
endmodelmodel;
```

A.2.4 prienc.bds

This file is translated into standard cell logic to determine the amount that the input to the interpolated needs to be shifted in order to normalize it. It also produces the value to be used as the exponent of the cell output.

```
MODEL prienc shifts<5:0>,shiftcntl<23:0> = bits<23:0>;

ROUTINE main;
          shifts = 0;
          shiftcntl = 1;
```

```
if bits<23> then begin
        shifts = 0;
        shiftcntl=1;
end
else if bits<22> then begin
        shifts = 0;
        shiftcntl=1;
end
else if bits<21> then begin
        shifts = 1;
        shiftcntl=4;
end
else if bits<20> then begin
        shifts = 1;
        shiftcntl=4;
end
else if bits<19> then begin
        shifts = 2;
        shiftcntl=16;
end
else if bits<18> then begin
        shifts = 2;
        shiftcntl=16;
end
else if bits<17> then begin
        shifts = 3;
        shiftcntl=64;
end
else if bits<16> then begin
        shifts = 3;
        shiftcntl=64;
end
else if bits<15> then begin
        shifts = 4;
        shiftcntl=256;
end
else if bits<14> then begin
        shifts = 4;
        shiftcntl=256;
end
else if bits<13> then begin
        shifts = 5;
        shiftcntl=1024;
end
else if bits<12> then begin
        shifts = 5;
        shiftcntl=1024;
end
else if bits<11> then begin
        shifts = 6;
        shiftcntl=4096;
end
else if bits<10> then begin
        shifts = 6;
```

```
                shiftcntl=4096;
        end
        else if bits<9> then begin
                shifts = 7;
                shiftcntl=16384;
        end
        else if bits<8> then begin
                shifts = 7;
                shiftcntl=16384;
        end
        else if bits<7> then begin
                shifts = 8;
                shiftcntl=65536;
        end
        else if bits<6> then begin
                shifts = 8;
                shiftcntl=65536;
        end
        else if bits<5> then begin
                shifts = 9;
                shiftcntl=262144;
        end
        else if bits<4> then begin
                shifts = 9;
                shiftcntl=262144;
        end
        else if bits<3> then begin
                shifts = 10;
                shiftcntl=1048576;
        end
        else if bits<2> then begin
                shifts = 10;
                shiftcntl=1048576;
        end
        else if bits<1> then begin
                shifts = 11;
                shiftcntl=4194304;
        end
        else if bits<0> then begin
                shifts = 11;
                shiftcntl=4194304;
        end;
ENDROUTINE;
ENDMODEL;
```

Bibliography

[1] M. Abramowitz. *Tables of Bessel Functions of Fractional Order*, volume 10 of *Columbia University Press Series*, chapter Note on Modified Second Differences for Use with Everett's Interpolation Formula, pages xxxiii–xxxvi. Nation Bureau of Standards, 1948.

[2] R. P. Brent and F. T. Luk. The solution of singular value and symmetric eigenvalue problems on multiprocessor arrays. *SIAM Journal on Scientific and Statistical Computing*, 6:69–84, 1985.

[3] Richard P. Brent, Franklin T. Luk, and Charles Van Loan. Computation of the singular value decomposition using mesh-conn ected processors. *Journal of VLSI and Computer Systems*, 1(3):242–290, 1985.

[4] Joseph R. Cavallaro and Anne C. Elster. A CORDIC processor array of the SVD of a complex matrix. In Richard J. Vaccaro, editor, *SVD and Signal Processing, II*, pages 227–239. Elsevier Science Publishers, 1991.

[5] M. D. Ercegovac, J. G. Nash, and L. P. Chow. An area-time efficient binary divider. In *International Conference on Computer Design*, pages 645–648. IEEE, Oct 1987.

[6] G. E. Forsythe and P. Henrici. The cyclic Jacobi method for computing the principal values o f a complex matrix. *Transactions of the Americal Mathematical Society*, 94(1):1–23, January 1960.

[7] Nariankadu D. Hemkumar. A systolic VLSI architecture for complex SVD. Master's thesis, Rice University, 1991.

[8] V. K. Jain. Statistical error analysis of VLSI architectures: Methodology and a case study. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 1089–1092. IEEE, 1989.

[9] V. K. Jain, D. L. Landis, and G. E. Alvarez. Systolic L-U decomposition array with a new reciprocal cell. In *International Conference on Computer Design*, pages 645–648. IEEE, 1989.

[10] V. K. Jain, G. E. Perez, and J. M. Wills. Novel reciprocal and square-root VLSI cell: Architecture and application to signal processing. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 1201–1204. IEEE, 1993.

[11] E. G. Kogbetliantz. Solution of linear equation by diagonalization of coefficients matrix. *Quarterly of Applied Mathematics*, 13:123–132, 1955.

[12] H. T. Kung. Why systolic architectures? *IEEE Computer*, 15(1):37–46, Jan 1982.

[13] F. T. Luk. Computing the singular-value decomposition on the ILIIAC IV. *ACM Transactions on Mathematical Software*, 6:524–539, 1980.

[14] Diana J. Major and Robert Sidman. A new use of the singular value decomposition in bioelectric imaging of the brain. In Richard J. Vaccaro, editor, *SVD and Signal Processing, II*, pages 497–512. Elsevier Science Publishers, 1991.

[15] Manuel D. Ortigueira and Miguel-Angel Lagunas. Eigendecomposition versus singular value decomposition in adaptive array signal processing. *Signal Processing*, 25(1):35–49, 1991.

[16] A. H. Sameh. On Jacobi and Jacobi-like algorithms for a parallel computer. *Mathematical Computing*, 25:579–590, 1971.

[17] C. Bernard Shung, Rajeev Jain, Ken Rimey, Edward Wang, Mani B. Srivastava, Brian C. Richrads, Erik Lettang, S. Khalid Azim, Lars Thon, Paul N. Hilfinger, Jan M. Rabaey, and Robert W. Bordsen. An integrated CAD system for algorithm-specific IC design. *IEEE Transactions on Computer Aided Design*, 10(4):447–463, April 1991.

[18] Gilbert Strang. *Linear Algebra and its Applications.* Harcourt Brace Jovanovich, Inc., third edition, 1988.

[19] John Todd, editor. *Survey of Numerical Analysis.* McGraw Hill Book Co., Inc., 1962.

[20] J. Volder. The CORDIC trigonometric computing technique. *IRE Transactions on Electronic Computers*, EC-8(3):330–334, September 1959.