SPIE—The International Society for Optical Engineering

PROCEEDINGS

Advanced Signal Processing Algorithms, Architectures, and Implementations IV

Franklin T. Luk
Chair/Editor

11–13 July 1993
San Diego, California

Sponsored and Published by
SPIE—The International Society for Optical Engineering

Volume 2027

SPIE (The Society of Photo-Optical Instrumentation Engineers) is a nonprofit society dedicated to the advancement of optical and optoelectronic applied science and technology.

CMOS processor element for a fault-tolerant SVD array

Kishore Kota and Joseph R. Cavallaro

Dept. of Electrical and Computer Engineering
Rice University, Houston, TX 77251

ABSTRACT

This paper describes the VLSI implementation of a CORDIC based processor element for use in a fault-reconfigurable systolic array to compute the Singular Value Decomposition (SVD) of a matrix. The chip implements a time redundant fault tolerance scheme, which allows processors adjacent to a faulty processor to act as computation backup during the systolic idle time. Also, processors around a fault collaborate to reroute data around the faulty processor. This form of time redundancy is attractive when tolerance to a few faults needs to be achieved with little hardware overhead.

Many of the proposed systolic array architectures for SVD are made of slightly dissimilar processors. We show that a physically uniform structure of the array simplifies the design, especially for fault-reconfigurable processor arrays. Our implementation required the addition of a number of architectural features to ease custom VLSI design. We eliminated the special physical edge connections proposed by earlier mesh architectures by adding extra programmability to the chip and embedding the original array in a regular toroidal structure. This allows undersized problems to be mapped onto the same physical array without padding the matrix with rows or columns of zeros. In addition, an entire row or column may be bypassed without the need for external switches, thus providing an extra degree of fault tolerance.

The chip was designed in a CMOS double-metal 2μ process and is $8954\mu \times 7840\mu$. The area overheads incurred in adding the time redundancy were an increase of about 40% in the number of controller states and a backup set of registers to store the faulty neighbor's data. The array was initially simulated at a higher level using VHDL descriptions and schematic capture software. This was then mapped to a custom chip using rapid-prototyping techniques.

1. INTRODUCTION

This paper discusses some details of the implementation of a systolic array to compute the Singular Value Decomposition (SVD) of a matrix. Given a $p \times p$ matrix A , the SVD is defined as $A = U\Sigma V^T$, where, Σ is a $p \times p$ diagonal matrix of what are termed *singular values*, U and V are $p \times p$ orthogonal matrices, the columns of which are called the left and right *singular vectors* respectively. Computation of the SVD requires $O(p^3)$ operations on a sequential computer. With a square array of processors it is possible to reduce the time complexity of the problem to $O(p \log p)$. Many real-time signal processing, image processing and robotics applications require fast computation of the SVD. Our research provides the framework for the design of a special-purpose processor array for the SVD to meet the demands of real-time processing applications, in particular the inverse kinematics engine of a robot.¹³ A complex mathematical problem has been mapped to relatively simple hardware through the use of special arithmetic techniques, extensive pipelining and parallelism. The design, fabrication and testing of a prototype custom VLSI processor for such an array has been completed. Construction of a 4×4 array of these processors has been completed and is currently being tested.

A secondary goal of the design was to determine the feasibility of a time-redundancy type of fault-tolerance scheme in the systolic array. To this end we discovered that it is essential to design the array to be as uniform as possible to avoid handling too many special cases. We avoided the complications of too many special cases by configuring a non-uniform array on a uniform toroidal mesh interconnection. This artificially introduced uniformity eased the design considerably. We were able to achieve the desired fault-tolerance with little hardware overhead.

2. BACKGROUND

2.1. CORDIC algorithm

In special-purpose VLSI processors, an efficient design with reduced area and time complexity often results from the use of special arithmetic techniques that map the desired computations to hardware. The principal computations involved in the SVD algorithm are vector rotations and inverse tangent calculations, and hence the choice of CORDIC as the arithmetic technique seems appropriate. Secondary operations such as addition, subtraction and divide by two required in the SVD computation, could re-utilize the functional units in CORDIC.

The COordinate Rotation DIgital Computer (CORDIC) technique was initially developed by Volder¹² as an algorithm to solve the trigonometric relationships that arise in navigation problems. Involving only a fixed sequence of additions or subtractions, and binary shifts, this scheme was used to quickly and systematically approximate the value of a trigonometric function or its inverse. This algorithm was later unified for elementary functions by Walther.¹⁴

The basic CORDIC iterations for circular mode of operation, as specified by Volder,¹² are:

$$\begin{aligned}x_{i+1} &= x_i + \delta_i y_i 2^{-i} \\ y_{i+1} &= y_i - \delta_i x_i 2^{-i} \\ z_{i+1} &= z_i + \delta_i \alpha_i,\end{aligned}\tag{1}$$

where x_i , y_i and z_i are the states of the variables x , y and z at the start of the i^{th} iteration, $0 \leq i < n$, and $\delta_i \in \{-1, +1\}$. The angles, $\alpha_i = \tan^{-1}(2^{-i})$, are pre-computed and stored in a table of angles. A choice of δ_i to reduce the initial angle z_0 to zero results in vector rotation or *z-reduction*. Inverse tangents are computed by choosing δ_i to reduce the initial value of y_0 to zero. This CORDIC operation is termed *y-reduction*. For a detailed description of CORDIC the reader is referred to Walther.¹⁴

2.2. Brent-Luk-van Loan SVD algorithm

In our system, the Brent-Luk-van Loan algorithm² (BLV) forms the basis for computation of the SVD of a matrix. In this paper, we will sketch the basic algorithm from the point of view of a hardware designer. For a detailed description of the algorithm the reader is referred to the original paper. We first introduce the terms *sweep*, *iteration* and *2-sided Jacobi transformation* in the current context. To compute the SVD of a $p \times p$ matrix, the algorithm requires $O(\log p)$ *sweeps* for convergence. Each *sweep* consists of $(p - 1)$ *iterations*. Each *iteration* consists of the parallel computation of $\frac{p}{2}$ *2-sided Jacobi transformations* and their application. Each *2-sided Jacobi transformation* selectively zeros two elements of the matrix.

$$R(\theta) = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}; \quad R(\theta_l)^T \begin{bmatrix} a & b \\ c & d \end{bmatrix} R(\theta_r) = \begin{bmatrix} \psi_1 & 0 \\ 0 & \psi_2 \end{bmatrix},\tag{2}$$

The computation of a *2-sided Jacobi transformation* primarily involves inverse tangent computations and a few secondary operations such as additions and shifts, on the elements of a 2×2 submatrix, to obtain two angles. These angles, called the *left angle* and the *right angle*, completely characterize the *2-sided Jacobi transformation*. Application of the *2-sided Jacobi transformation* requires performing vector rotations on vectors of size two. By choosing a specific ordering called *parallel ordering* it is possible to reduce the above algorithm to the following:

- The $p \times p$ matrix, A , is partitioned into 2×2 submatrices. A square array of $\frac{p}{2} \times \frac{p}{2}$ processors, each storing a 2×2 submatrix results from this partition (see Fig. 1).
- All processors on the main diagonal simultaneously compute the *2-sided Jacobi transformations* using only the four data elements available in their respective registers. This results in the computation of $\frac{p}{2}$ *2-sided Jacobi transformations*, one computed in each main diagonal processor.

Figure 1: A 4×4 array of processors to compute the SVD of an 8×8 matrix.

- Each processor systolically propagates the left and right angles along its row and column respectively, to result in diagonal waves of computations originating at the main diagonal. Each off-diagonal processor receives the two angles from its predecessors along the row and column and applies them to the two vectors stored in its registers. It then systolically passes these angles to the next processor. When the angles reach the edge of the array, the simultaneous application of $\frac{\pi}{2}$ Jacobi transformations is complete.
- At the earliest possible time, the neighboring processors exchange data in a particular fashion so that a new set of data is available at the main diagonal to start a new *iteration*.
- The permutation used to exchange data ensures that all the matrix elements return to their respective places in the array at the end of every *sweep*.
- For hardware simplicity, a pre-determined number of sweeps are performed to ensure convergence of the algorithm.

2.3. Cavallaro-Luk CORDIC-SVD architecture

The SVD algorithm described in the previous section may be mapped to a systolic array using CORDIC. Cavallaro and Luk,³ Delosme,⁵ and Yang and Böhme¹⁵ have demonstrated the use of CORDIC for Jacobi transformations required in the computation of the SVD of real matrices. Recently, redundant and on-line CORDIC based approaches have been suggested by Ercegovic and Lang⁶ and Lee and Lang.⁹ We use the Cavallaro-Luk method to map the SVD algorithm to hardware. In the Cavallaro-Luk method, by providing two independent CORDIC units, computation of the *2-sided Jacobi transformation* in each processor requires approximately the time required to perform one inverse tangent computation or a CORDIC *y-reduction*, which requires one CORDIC cycle. Application of the transformation requires two CORDIC cycles to perform four vector rotations using CORDIC *z-reduction*. Multiple *iterations* in the array may be pipelined, separated in time by seven and a half CORDIC cycles and the time to move data both within and between the processors.

Figure 2: The time-redundancy approach to tolerate the presence of a faulty processor in the array. Processor P12 is faulty and processor P11 acts as its computation backup.

2.4. Fault tolerant SVD array

Cavallaro-Near-Uyar described a time-redundancy fault-tolerance scheme⁴ for the above CORDIC-SVD array. The scheme is expected to result in very little hardware overhead while providing tolerance to multiple non-adjacent faults. The basic idea is that processors surrounding a faulty processor collaborate to route data around the fault to one of the processors that acts as a backup during its systolic idle time. Since approximately a third of the processors are idle at any given time even when the pipeline is full, this backup scheme may be implemented in the array with very little overhead. Figure 2 shows a 4×4 array with an interior super-diagonal fault, namely processor P12. In this figure, processor P11 acts as the computation backup. P11 has an extra set of registers, which store the four matrix elements that ought to be in P12. The other 7 processors surrounding P12, which communicate with P12 in a non-faulty array, now need to communicate with the backup processor P11. The communication links shown using thicker lines are used to achieve this communication. Processors P01, P21, P02 and P22 can communicate directly with processor P11 using just one communication link. Processors P03, P23 and P13 however, need to communicate through the intermediary processors, either P02 or P22 as shown in Fig. 2.

3. DESIGN OF CAPE

The CORDIC Array Processor Element (CAPE) is an implementation of the architecture proposed by Cavallaro and Luk³ with some fault-tolerance features.⁴ The primary computational ability of the chip is SVD of a 2×2 matrix.

Figure 3: (a) A 4×4 toroidal mesh with four way connections, (b) an alternate layout of the same interconnection

The BLV algorithm requires different portions of the 2×2 SVD computation performed in the processors at different positions in the array. These small differences add to the operating modes of the chip. We have identified 3 different computation modes, i.e. positions in the array that vary in the way the data in the registers is operated upon, and 11 different communication modes, differing in the way data is exchanged with neighboring processors. To provide flexibility in testing the chip stand-alone or in a capacity other than a SVD processor, we allow access to sub-portions of the chip, resulting in 3 additional modes.

The addition of fault-tolerance significantly affected the physical and logical structure of the array. In the presence of a fault, each chip around the fault is required to either act as a backup or route data in a different fashion. This translates to eight separate modes, one corresponding to each position relative to the fault. In a heterogeneous array, such as the one proposed by Brent-Luk-van Loan, to tolerate a single fault at *any* location in the array, each separate mode would add 8 additional modes. This results in an explosion of modes which is neither elegant nor simple to design. The basic problem is that time-redundancy based fault-tolerance adds heterogeneity to the array. Hence if the fault-free array is itself heterogeneous, the resulting fault-tolerant processor needs the ability to reconfigure into a number of different configurations equal to roughly the product of the number of modes in the fault-free array and the number of modes added by each fault. The mesh interconnection is inherently heterogeneous because the edge processors see a different interconnection from the interior processors. A toroidal mesh interconnection (Figure 3), however, avoids this problem. We alleviate the problem of explosion in the number of modes by configuring the heterogeneous BLV array (Fig. 1) on top of a physically homogeneous 8-connected toroidal mesh interconnection. Since the homogeneity of the toroidal interconnection allows the logical array to be mapped with the processor P00 mapped to any physical processor, tolerance to a single type of fault achieves tolerance to a fault in any location of the array. In our array we chose to tolerate a super-diagonal interior fault in the BLV array. A fault in any other location in the array is tolerated by *remapping* the logical BLV array on the physical array, such that the faulty processor is mapped to a superdiagonal interior position. Thus we could design the processor with only 8 additional modes. Mapping the logical BLV array onto a physical toroidal mesh has some additional properties. The ability to configure the array edges implies the SVD of undersized square matrices may be computed without any padded zeros. The resulting array has additional fault-tolerance properties, which allow remapping the problem to a smaller contiguous processor array, effectively bypassing an entire row or column. Hence traditional fault-tolerance schemes may be implemented on the same array with no added cost.

Figure 4: Block diagram of CAPE.

3.1. Major blocks on the chip

The major blocks in CAPE processor are shown in Figure 4. The processor consists of two CORDIC units, two register banks for storing intermediate values and two shift-register banks to store the values to be communicated between processors. When a complex algorithm is implemented in hardware, it is necessary to assign subtasks to separate controllers, to save chip area and to ensure that the controllers are not slow. In CAPE, the controllers are arranged in a hierarchy with the *intra-controller* at the lowest level controlling the CORDIC units and using the temporary register banks to perform calculations. At the highest level the *inter-controller* co-ordinates the computations and communication both within and between processors. A custom communication exchange redirects data between the serial communication links and the shift registers. Asynchronous handshake with the external host is performed by the *port-controller*. Moving data between the external host and the internal registers is performed as a subtask by the *read-write controller*. A primary concern in dividing the entire sequence of controls necessary to accomplish the different modes is that any given resource should be controlled by only one controller.

3.2. Design of the CORDIC units

The CORDIC units are designed to implement the iterations described by Equation 1. A CORDIC controller implements the basic CORDIC operations, *z-reduction* and *y-reduction*, and allows access to single CORDIC iterations to perform additions and shifts as special cases. These operations are available to the controller at the next level in the hierarchy, namely the *intra-controller*. The datapaths which implement the CORDIC iterations consist of registers, barrel shifters, adders and a ROM table of angles to store the angles α_i . Figure 5 shows a block diagram of the CORDIC units. This diagram may be seen as a straightforward implementation of Equations 1. The blocks named *normalization shifters* are additional hardware to correct some numerical errors⁸ we observed in the *y-reduction* mode of traditional CORDIC.

Figure 5: Block diagram of CORDIC modules with normalization for inverse tangent computation.

3.3. Intra-controller and intra-processor communication

In an initial version of the chip, we observed that the time to compute the 2×2 SVD was much larger than the time for three CORDIC cycles. We attributed this to the time it takes to move data within the chip between the various registers and CORDIC units. In CAPE we reduced this time by using a two bus architecture within the chip. Each bus is connected to one CORDIC unit. Data movement between the CORDIC units is achieved through the dual-ported register banks *regbank1* and *regbank2* (See Fig. 4). Each register bank consists of three registers and may be accessed from either bus. Only one register from each bank may be accessed at any time, but the two banks may be accessed in parallel. An additional register in each bank is hardwired to zero. Thus a value of zero may be driven onto either bus to be used as an argument in certain calculations. Comparators are provided on each bus to compare the arguments with zero and handle special cases. The *intra-controller* provides all the control signals for the CORDIC units, *regbank1*, *regbank2* and the zero comparators. The functions provided by the intra-controller are a complete 2×2 SVD, application of a *2-sided Jacobi transformation* on a 2×2 matrix, z-reduction and y-reduction.

3.4. Inter-controller and inter-processor communication

The inter-controller is the highest level controller in the hierarchy. Its function is to sequence all the operations in the chip utilizing the other controllers to perform subtasks. Some of the operations involved are loading the shift register banks with data from the external host using the *port-controller*, performing the desired computation using the *intra-controller* and exchanging data with other chips in the array. In addition, it implements the different modes necessary to operate in the presence of a fault in the array.

Figure 6: Design methodology

The time-redundancy approach to fault-tolerance results in additional communication along some of the links. It is necessary to schedule the communication along the links such as to avoid conflicts and to use a uniform protocol throughout. In our array the additional communication to tolerate a fault does not conflict with the communication along the links in the fault-free array. Hence, the timing of the data exchange remained the same even when redirected through a different set of links. We were able to complete the communication without affecting the time for the overall SVD computation. Hence the processors far from the fault never get affected and continue operations as in a fault-free array. Hence all faults that are at least two away from each other can be tolerated by this array provided they can all be mapped to interior super-diagonal processors. The host needs to run a diagnostic test to determine the cause of the fault and reconfigure the array for all future SVD calculations.

3.5. Array loading and the parallel bus

In CAPE, we implemented an asynchronous loading scheme that allows interface with a host running at an independent clock rate. Data are transferred from the host using a common 8-bit bus. The data transfer uses a typical handshake protocol. We implemented two different protocols to allow interface with either an Intel 8255 parallel I/O device or a TMS320C40 DSP processor.

Care had to be taken to allow array loading in the presence of faults. A daisy chain approach to control transfer from one processor to another is not possible, since it would not work in the presence of a fault. Instead we opted for polling, assuming nonmalicious faults. The particular processor that is to be accessed is selected through an 8-bit poll bus. Each processor has a hardwired 8-bit chip address which is used to match the poll. In the presence of a fault, the host selects the backup and loads the data of the faulty chip into it, when loading a new problem. In addition, it reconfigures all the processors around the fault to their new modes, to allow the next SVD computation to be performed without the faulty processor.

3.6. Design methodology

Figure 7: Die photograph of CAPE.

The design of CAPE posed a number of problems, which we were able to solve using the appropriate CAD tools to rapidly prototype the chip. A constant problem posed in the design of this chip was that a number of interactions took place at the array level rather than the chip level. A number of errors crept into the design due to the number of special cases that needed to be handled correctly for correct array operation. Hence it was necessary to be able to define and simulate the array at a higher level to resolve problems in array interaction. Schematic capture tools provide a convenient framework to design at this level. It was possible to define schematics for the chip as well as the array, with some of the lowest level blocks described in VHDL. The *Workview*¹¹ system was used at this level. Simulation at this level was performed using *viewsim*. After the array design was complete, it was imperative that we exported the schematics and VHDL descriptions directly to a layout system. The *Vanda* set of tools allowed this transfer of data from the schematic software to the layout tools. This enabled numerous changes to be made in the high-level description and propagated to the layout system, to automatically generate the layout in a short time, typically less than an hour. The interaction of the various tools in our design environment is presented in Fig. 6.

The leaf cells were described in the schematic capture system using a subset of VHDL. Care was taken to use only such constructs as are synthesizable. Layout for some of the cells was initially available from our previous versions of the non-fault-tolerant chip.⁷ In such cases, the VHDL was just a tool for simulation and hence the VHDL was functional in nature and not synthesizable. Most random logic blocks and controllers were written in synthesizable VHDL. The random logic blocks could be compiled by a synthesis tool provided with the schematic capture system, using cells provided by the layout system, in our case the UC Berkeley CAD system, *Lager*.¹⁰ The controllers were too large to be synthesized into standard cell design. Hence, they were mapped into PLAs. The random logic blocks were coalesced into a single block, optimized and then routed using the *TimberWolf* package from the *Octtools* suite. The various blocks within the chip were routed using the *Flint* package available in the *Lager* set of tools.

The chip was designed in a CMOS double-metal 2μ nwell process offered by the MOSIS fabrication service. The

chip was designed conservatively using static CMOS techniques. The die size is $8954\mu \times 7840\mu$. The chip consists of approximately 35000 transistors. CAPE was fabricated in March 1993. It is packaged in an 84 pin PGA. A die photograph of the chip is shown in Fig. 7.

4. DESIGN OF THE SVD ARRAY

4.1. Packaging Goals

The goal of the array design was to design the fewest number of identical circuit boards which could be put together to form a processor array of arbitrary size. This was motivated by the technique devised by Bhatt and Leiserson¹ to package a tree machine. Our goal was to determine if a torus could be packaged in a similar fashion. To meet this goal we had to first determine if there exists a layout of a torus in 2-dimensions, such that all the wire lengths are a constant independent of the size of the array.

4.2. Layout of a torus in 2-dimensions

The interconnection graph of the SVD array is an 8-connected mesh. Every processor communicates with its eight nearest neighbors. A way to layout a 4-connected mesh with north-south-east-west connections is shown in Figure 3(b). This layout requires each row and column of the original conventional layout to be placed in two rows and columns respectively. With a little book-keeping a constant wire length, independent of the array size is obtained.

An 8-connected mesh with constant wire lengths is obtained by making the following observation: any diagonal communication can be thought of as two communications, one to the neighbor along the x-axis and the other to the neighbor along the y-axis. Alternately, the connection can be thought of as two wires that travel along the above mentioned directions. Since, each wire is a constant length, the diagonal wire is also a constant length, twice the north-south-east-west connections.

Design of an expandable board involves partitioning the layout, such that the partitions obey the property of regularity i.e. all partitions are similar in structure, and simultaneously minimizing the number of interconnections between partitions. An aesthetically pleasing design would partition the array such that the communication between partitions is a constant independent of the size of the partition (alternately, the size of a partition is the number of processors that can be conveniently placed and routed on a single circuit board). We partitioned the array using cuts parallel to the x and the y axes to form subarrays. Such a partitioning results in inter-partition communication that grows linearly as the size of the partition. To demonstrate the expandable nature of our design, we designed a circuit board with a 2×2 processor array on each, and used four of these boards to form a 4×4 SVD array.

Figure 8 shows a photograph of the completed 4×4 array. The layout of the torus may be viewed as a mesh layout with special connections at the edges to wrap the wires of adjacent rows or columns around. We achieved this objective using jumpers which short the wires from adjacent rows or columns only at the edges. The interface to the array is through the parallel ports on a personal computer acting as a host. In our tests the main program runs on the host and uses the array as a coprocessor to accelerate the computation of SVD.

5. SUMMARY AND CURRENT WORK

We implemented a systolic array to compute the SVD of a matrix using the BLV algorithm. The array also incorporates a time-redundancy fault-tolerance scheme. Initially we designed a custom processor to be used in the array. Numerous architectural changes were made to the original proposed designs to accommodate implementation details and fault-tolerance capabilities. The most important change is that the array was made uniform to conveniently implement the fault-tolerance scheme. Uniformity appears to be the key in the implementation of this kind of fault-tolerance. A way to layout the array such that an arbitrary sized array may be built using almost identical building blocks was investigated. We designed a board for a 2×2 array of processors, four of which were used to construct a 4×4 array. The array is currently being tested.

Figure 8: Photograph of a working 4×4 CAPE processor array.

The method that we used to package the array achieves constant wire lengths when laid out in 2-dimensions. However, we were unable to determine if the array can indeed be packaged in 3-dimensions with wire lengths that are constants independent of the size of the array. Results from 3-dimensional VLSI layout theory might provide keen insights on wire lengths, which can be used to guide the search for a compact layout in 3-dimensions.

The design of this array was an interesting experience which revealed that a number of minor details overlooked at the higher level, play a significant role in proper system functioning. The design cycle was quite large, but this time included the time to learn the numerous CAD tools used in the system design. The design used multiple CAD systems at different abstractions using simulations at every stage to eliminate the design errors. The effort involved in the design of a special purpose system is quite large. The use of high-level synthesis tools moves the onus of the design from the designer to the machine and considerably reduces the design cycle. The designer just needs to concentrate on the design at the schematics level. However, to achieve high speeds still a lot of effort is needed. Some design principles, such as near-neighbor communication, allow a fast system to be designed with very little effort. The ability to incorporate such general principles applied uniformly seems to considerably ease the designer's task and eliminate the special cases which is where most of the design errors creep in.

In conclusion, we feel design of a custom array should include specification, design and simulation in a top-down fashion starting at the specification of the array and gradually descending the hierarchy to design the cells in the chip. Use of high-level synthesis tools is a necessity and it allows a working system to be designed with a quick turnaround.

6. ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation under awards MIP-8909498 and DDM-9202639. Thanks are also due to N. D. Hemkumar for suggesting the partitioning method used in packaging the array.

References

- [1] S. N. Bhatt and C. E. Leiserson. How to assemble tree machines. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, pages 77–84, May 1982.
- [2] R. P. Brent, F. T. Luk, and C. F. Van Loan. Computation of the Singular Value Decomposition Using Mesh-Connected Processors. *Journal of VLSI and Computer Systems*, 1(3):242–270, 1985.
- [3] J. R. Cavallaro and F. T. Luk. CORDIC Arithmetic for an SVD Processor. *Journal of Parallel and Distributed Computing*, 5(3):271–290, June 1988.
- [4] J. R. Cavallaro, C. D. Near, and M. Ü. Uyar. Fault-Tolerant VLSI Processor Array for the SVD. *IEEE Int. Conf. on Computer Design*, pages 176–180, October 1989.
- [5] J. M. Delosme. A Processor for Two-Dimensional Symmetric Eigenvalue and Singular Value Arrays. *IEEE 21th Asilomar Conf. on Circuits, Systems, and Computers*, pages 217–221, November 1987.
- [6] M. D. Ercegovic and T. Lang. Redundant and On-Line CORDIC: Application to Matrix Triangularization and SVD. *IEEE Trans. Computers*, 39(6):725–740, June 1990.
- [7] N. D. Hemkumar, K. Kota, and J. R. Cavallaro. CAPE - VLSI Implementation of a Systolic Processor Array: Architecture, Design and Testing. *Proc. 9th Biennial University/Government/Industry Microelectronics Symposium*, pages 64–69, June 1991.
- [8] K. Kota and J. R. Cavallaro. Numerical accuracy and hardware tradeoffs for CORDIC arithmetic for special-purpose processors. *IEEE Transactions on Computers*, accepted for publication.
- [9] J. Lee and T. Lang. Constant-Factor Redundant CORDIC for Angle Calculation and Rotation. *IEEE Trans. on Computers*, 41(8):1016–1025, August 1992.
- [10] C. B. Shung, R. Jain, K. Rimey, E. Wang, M. B. Srivastava, B. C. Richards, E. Lettang, S. K. Azim, L. Thon, P. N. Hilfinger, J. M. Rabaey, and R. W. Brodersen. An Integrated CAD System for Algorithm-Specific IC Design. *IEEE Trans. on CAD*, 10(4):447–463, April 1991.
- [11] Viewlogic Systems, Inc., Marlboro, MA. *Schematic Design User's Guide*, December 1989.
- [12] J. Volder. The CORDIC Trigonometric Computing Technique. *IRE Trans. Electronic Computers*, EC-8(3):330–334, Sept. 1959.
- [13] I. D. Walker and J. R. Cavallaro. Parallel VLSI Architectures for Real-Time Kinematics of Redundant Robots. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 870–877, Atlanta, GA, May 1993.
- [14] J. S. Walther. A Unified Algorithm for Elementary Functions. *AFIPS Spring Joint Computer Conf.*, pages 379–385, 1971.
- [15] B. Yang and J. F. Böhme. Reducing the Computations of the SVD Array given by Brent and Luk. *SIAM J. Matrix Analysis Appl.*, 12(4):713–725, 1991.