

FPGA-Based Co-processor for Singular Value Array Reconciliation Tomography

Jack Coyne, David Cyganski, and R. James Duckworth
Worcester Polytechnic Institute, Worcester, MA, USA

Abstract—We present an FPGA-based co-processor for accelerating computations associated with Singular Value Array Reconciliation Tomography (SART), a recently developed method for RF source localization. The co-processor allows this relatively complex computational task to be performed using less hardware and less power than would be required by a microprocessor-based computing cluster with comparable throughput and accuracy. The architecture exploits parallelism of the SART algorithm at many levels in order to efficiently map it into the FPGA platform. The system has been developed in VHDL, and implemented on a Virtex-4 SX55 FPGA. Compared to a Pentium 4 CPU running at 3 GHz, use of the co-processor system provides a speed-up of about 6 times for the current signal matrix size of 128-by-16. Greater speed-ups will be obtained when larger matrices are processed because the co-processor reduces the complexity of a m-by-n SVD from $O(mn^2)$, to $O(mn)$. Even larger speed-ups may be obtained by using multiple devices in parallel. The system is capable of computing the SART metric to an accuracy of about -145 dB (0.2 ppm) with respect to its true value. This level of accuracy, which is better than that obtained using single precision floating point arithmetic, allows even relatively weak signals to make a meaningful contribution to the final SART solution.

I. INTRODUCTION

The goal of the Worcester Polytechnic Institute (WPI) Precision Personnel Locator (PPL) project [1] is to develop a system capable of providing sub-meter accuracy in the location and tracking of personnel situated within indoor environments. In general, radio based tracking systems attempt to extract time-of-flight information from radio signals that have propagated through a building, from the object being tracked to externally located reference units, or visa versa. The major challenge posed by this technique results from the fact that radio-frequency electromagnetic waves are reflected and diffracted by metallic structural members and metallic objects within the building. These phenomena result in a received signal that can be defined as the sum of the direct-path component and the reflected, “multi-path”, components. Solving for the range to the object being tracked requires isolating the direct-path component. Separating this component from the undesired portion of the received signal presents a significant challenge to research teams developing radio-based location systems [2]. Faculty and student researchers at WPI have developed an algorithm known as Singular Value Array Reconciliation Tomography (SART) [3] [4] in order to overcome this challenge.

One of the most evident obstacles on the path towards deployment of the PPL system is computational complexity of the SART algorithm. The amount of computation associated with the SART algorithm is so great that scanning a 10m by

10m room for a single target takes approximately 1 second using a Pentium 4 CPU with a clock frequency of 3GHz. Using this figure, and assuming that a structure consists of many rooms and many floors, and that there could potentially be dozens of personnel that need to be located or tracked, a complete scan could easily take many minutes.

Acceleration of the SART algorithm could be accomplished using a cluster of PCs working in parallel, but this solution would be physically too large for a mobile application, and have needlessly high power consumption. A smaller, more efficient, solution would be one that uses custom hardware instead of the generalized architecture of a personal computer. For flexible and customizable high-speed signal processing, Field Programmable Gate Arrays (FPGAs) are the industry standard hardware. An FPGA-based system was chosen as the platform for the SART accelerator system described here.

II. SINGULAR VALUE ARRAY RECONCILIATION TOMOGRAPHY

The location process begins with an RF transmission by the device that is to be located. The transmitted signal is a wide-band multi-tone waveform that can be represented in a discrete-frequency domain by an array of complex values,

$$S_0 = [s_1 \quad s_2 \quad s_3 \quad \dots \quad s_m] \quad (1)$$

which describe the phase and magnitude of the individual tones (or sub-carriers). In our system, we have commonly used $m \approx 100$.

The transmission is recorded at multiple receiving elements, which ideally surround the transmitter. For the PPL project application of SART, these receiving elements correspond to antennas located on fire-trucks and other vehicles located around the outside of a building in which personnel are being tracked. The frequency domain representation of the received signal has the form

$$S = D + M \quad (2)$$

where

$$D = \alpha_0 \begin{bmatrix} s_1 e^{-jk d_0} & s_2 e^{-j2k d_0} & \dots & s_m e^{-j m k d_0} \end{bmatrix} \quad (3)$$

and

$$M = \sum_{i=1}^q \alpha_i \begin{bmatrix} s_1 e^{-j k d_i} & s_2 e^{-j 2 k d_i} & \dots & s_m e^{-j m k d_i} \end{bmatrix} \quad (4)$$

with

$$\dot{k} = 2\pi \Delta f c^{-1} \quad (5)$$

Row vector \mathbf{D} represents the direct path component of the received signal. Row vector \mathbf{M} represents the contributions of q different multipath components. The α factors represent the complex signal attenuation along the corresponding path. The exponential factors represent the phase shifts of individual sub-carriers. These phase shifts are due to propagation delay, and depend on the propagation distances, d_i , between the transmitter and receiver along individual paths, the sub carrier frequency spacing, Δf , the sub-carrier index, $[1, \dots, m]$, and the propagation velocity, c , which is equal to the speed of light.

Signals from multiple receivers are used to form a signal matrix, where the columns represent different receivers, and the rows represent different sub-carriers (frequencies) in the multi-tone signal. For a system with n receive antennas, the signal matrix has the form

$$\mathbf{S} = [S_1^T \quad S_2^T \quad \dots \quad S_n^T], \quad (6)$$

where S_k^T represents the transpose of row vector S_k , and contains the sub-carrier phase and amplitude information gathered at the k^{th} receive element. The signal matrix, \mathbf{S} , acts as an input to the SART algorithm. The remaining inputs are the relative coordinates of the receive antennas, which are assumed to be known.

A. Scanning

The SART algorithm is an exhaustive imaging approach. Given the algorithm inputs at any given instance, the physical area of interest is “scanned” at regular spatial intervals, where it is assessed in terms of the SART metric. This set of scan locations is known as the “scan-grid”. The location of maximum metric value indicates the estimated location of the transmitter. An example SART image is shown in Figure 1, rendered as a contour plot. The center of the innermost contour indicates the estimated transmitter location.

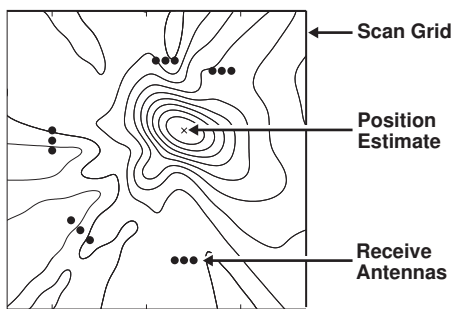


Fig. 1. Example of a SART image

Scanning is accomplished by applying negative time delays to the received signals. These delays correspond to the light-speed travel time between the current scan location and each receive element. Because the locations of the receive elements are known, the time delays can easily be calculated for each location on the scan-grid. Time delay application is accomplished by imposing a linear (with frequency) phase

shift to the signals from each receive element. For example, the rephased version of element S_{jk} from the signal matrix is

$$S'_{jk} = S_{jk} e^{jk d_{kp}}, \quad (7)$$

where d_{kp} is the distance from the k^{th} receive element, to a location, p , on the scan grid, and k is from (5).

If the exact location of the transmitter is scanned, then the application of the negative time delays will counter-act the actual time delays associated with the direct path propagation distance, d_{kt} . That is, if $d_{kp} = d_{kt}$, then

$$D'_{jk} = D_{jk} e^{jk d_{kp}} = (s_j \alpha_k e^{-jk d_{kt}}) e^{jk d_{kp}} = s_j \alpha_k \quad (8)$$

for all signal matrix elements $S_{jk} = D_{jk} + M_{jk}$. The result of this will be alignment of the direct-path portions of the signals, to within a constant phase. That is, they will differ by only their complex attenuation factors, such that,

$$D'_{j1} \alpha_1^{-1} = D'_{j2} \alpha_2^{-1} = \dots = D'_{jn} \alpha_n^{-1} = s_j \quad (9)$$

This alignment increases the level of linear dependence between the direct path components of the rephased signals. The level of linear dependence between the rephased signals is measured by performing singular value decomposition of the signal matrix, and observing the first singular value[5].

B. Singular Value Decomposition

Using the process of Singular Value Decomposition (SVD) a matrix, \mathbf{A} , can be broken down into a set of unitary column and row vectors, \mathbf{U} and \mathbf{V} , and a diagonal matrix Σ containing the singular values, σ_i , such that

$$\mathbf{U} \Sigma \mathbf{V}^H = \mathbf{A}$$

where \mathbf{V}^H indicates the complex conjugate transpose of \mathbf{V} . The singular vectors serve as an orthonormal basis for the matrix, while the singular values contain information about the amplitudes of these vectors in their role as components within the original matrix. Together, the singular values provide an indication of matrix rank [6]. For example, if a matrix is composed of multiple linearly dependent columns (or rows), then the matrix is said to have a rank of one (i.e. all but the first singular value will be zero). If there are small dissimilarities between columns then the SVD will reveal more non-zero singular values, with magnitudes corresponding to the amplitudes of the components that impart these dissimilarities. Larger dissimilarities will be reflected in larger rank supporting singular values. Leveraging the properties of singular values, the SART algorithm uses singular value decomposition to obtain a measure of linear dependence between columns of the signal matrix in the form of the first singular value, σ_1 . This is the SART metric.

The process of singular value decomposition can be divided into three main stages: QR decomposition, bidiagonalization, and diagonalization.

scan-grid location. Assuming each complex multiplication requires 6 operations, the complexity of this stage is $O(6Gmn)$. The amount of data in is mn . The amount of data out is Gmn .

C. QR Decomposition

Using a figure from [6], the complexity of QR decomposition using Givens rotations is $O(12n^2(m - n/3))$. Multiplying by the number of locations in the SART scan-grid gives a total complexity of $O(12Gn^2(m - n/3))$. The amount of data in is Gmn . The amount of data out is less, due to the elimination of the lower portion of the matrix. The amount of data flowing out of this stage is G multiplied by number of non-zero elements in an n -by- n upper triangular matrix, $G(n^2 + n)/2$.

D. Bidiagonalization

The number of operations required to compute bidiagonalization using Givens rotations is approximately twice the complexity of QR decomposition of an equally sized matrix. The total complexity of the bidiagonalization stage is $O(16Gn^3)$. The amount of data in is $G(n^2 + n)/2$. The amount of data out is equal to one diagonal and one super diagonal for each scan-grid location, or $G(2n - 1)$.

E. Diagonalization

According to [6], the complexity of diagonalization is $O(54n)$ per iteration, assuming single precision, and approximately 12 operations per square-root (Intel processors generate 2 bits of precision each clock cycle for square root calculations [10]). For a scan-grid with G scan locations, and assuming a conservative 20 iterations, the total complexity is $O(1000Gn)$. The amount of data in is $G2n - 1$. The amount of data out is Gn .

F. Totals

In the current implementation of SART, sixteen receive elements are used, and 103 sub-carrier tones are transmitted. The signal matrix therefore has dimensions m -by- n equals 103-by-16. A moderately sized scan-grid, perhaps for a small building, consists of $G = 10,000$ points. The number of samples collected for each receive element is $N = 8192$. Using these figures, the operation and data counts from above can be calculated. Table I summarizes these values.

TABLE I
OPERATION AND DATA OUTPUT COUNTS FOR SART PROCESSING STAGES

Processing Stage Name	Operations [millions]	Data Out [MB]
Fast Fourier Transform	0.1065	0.0132
Rephasing	19.78	26.37
QR Decomposition	600.1	2.176
Bidiagonalization	131.1	0.496
Diagonalization	32.00	0.256

G. Partitioning Decisions

The SART algorithm was partitioned such that a portion of the associated computations are conducted using custom co-processor hardware, while the remaining portion are conducted in software on the host PC. Three factors were considered when choosing the partition boundaries:

- the number of computations associated with each processing stage
- the amount of data flowing in and out of each stage
- the suitability to parallel implementation of each stage

Using the results from the previous section, it was clear that the QR decomposition and bidiagonalization stages represent the bulk of the SART computational burden, and were therefore the prime targets for custom hardware implementation. The rephasing stage, though not computationally intensive, produces many rephased signal matrices for each input signal matrix. In order to reduce the amount of data transferred from the host to the co-processor system, the rephasing stage was also selected for hardware implementation.

The remaining portions of the SART algorithm were assigned to the host PC, but could be assigned to a general purpose DSP type processor for a more compact implementation. The FFT stage was assigned to the host in order to allow for manipulation of the frequency domain signal data before SART processing. This allows for various calibration and synchronization corrections to be applied by the host. The diagonalization stage was assigned to the host PC because it involves algorithms that are both sequential and iterative, and therefore better suited to a sequential processor.

Using this partitioning, the number of operations assigned to the host PC was reduced by more than 95%. This corresponds to a potential speed-up of more than 20, without modifications to the host. The following section describes the architecture of the co-processor.

IV. CO-PROCESSOR ARCHITECTURE

The SART co-processor was designed in VHDL, and implemented on Xilinx Virtex-4 SX55 FPGA. For the prototype system, a ADM-XRC-4 development board from Alpha-Data was used. The ADM-XRC-4 module consists of the SX55 device, SRAM, and a PCI interface IC. When used in conjunction with a ADC-PMC adapter card, it may be installed into any PCI compatible host PC. A top-level diagram of the SART co-processor is shown in Figure 3. The main components of the co-processor are the:

- Host PC interface
- Rephasing stage
- QR decomposition stage
- Bidiagonalization stage

A. Host Interface

The host PC interfaces with the SART co-processor system through a PCI bridge IC. A state machine and output logic were instantiated inside the FPGA, for controlling the bridge

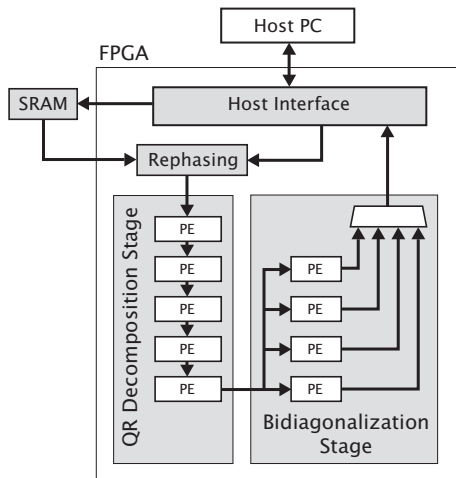


Fig. 3. Top-level structure of the SART co-processor system

IC via its local bus signals. These components reduce the interface to simple address and data buses, with additional signals to indicate read and write transactions, and acknowledge data availability. Memory and control elements within the FPGA are mapped into the memory space of the host PC. Device drivers allow the SART co-processor system to be controlled using a C program or MATLAB script.

B. Rephasing Stage

Rephasing of the SART signal matrix is accomplished by element-wise multiplication with a “phase reference” matrix. The phase reference matrix is simply the set of complex exponentials that describe the frequency-dependent phase shifts for the current scan grid location. In order to reduce the amount of memory occupied by the SART phase reference data, a simple compression scheme was adopted.

Because the current PPL system employs a signal with sub-carrier tones that are placed on evenly-spaced frequency intervals, the phase shifts undergone by these sub-carriers are also evenly spaced. For example, if the first sub-carrier undergoes a phase shift of θ_0 , then the second and third sub-carriers will undergo phase shifts of $\theta_0 + \Delta\theta$ and $\theta_0 + 2\Delta\theta$. Furthermore, the common phase shift, θ_0 , does not effect the singular values of the signal matrix because it does not effect the linear dependence between columns. Ignoring θ_0 , the phase shift of the k^{th} sub-carrier may be expressed as

$$\Theta_k = k\Delta\theta \quad (18)$$

the phase reference unit vector that encodes this phase shift is

$$U_k = e^{jk\Delta\theta} = \prod_{i=0}^k e^{j\Delta\theta} \quad (19)$$

therefore the phase reference vector for any one sub-carrier may be obtained by multiplying the phase reference vector of the previous sub-carrier with the constant phase step vector, $e^{j\Delta\theta}$. This can be expressed as

$$U_k = U_{k-1}e^{j\Delta\theta}, \quad \text{where } U_0 = 1 \quad (20)$$

By storing only this phase step, and generating the phase reference vectors as needed, the amount of memory consumed by the phase reference data may be reduced by a factor equal to the number of sub-carriers. For the current signal matrix, the storage space is reduced by a factor of 100. By employing this method, the current SART co-processor system supports a scan grid of up to 16,384 points using a single 1 Mbit SRAM bank.

The rephasing matrices are decompressed using the iterative technique described in Equation 20. One phase step vector, $e^{j\Delta\theta}$, is retrieved from SRAM for each column of the matrix, and stored in a FIFO buffer. An equal number of unit length, zero-phase vectors are loaded into a second FIFO, these are the rephasing vectors for the first row in the signal matrix. In subsequent cycles, calculation of the rephased signal matrix and decompression of the rephasing matrix happen simultaneously, with one element of the rephased signal matrix and one element of the rephasing matrix being generated on each clock cycle.

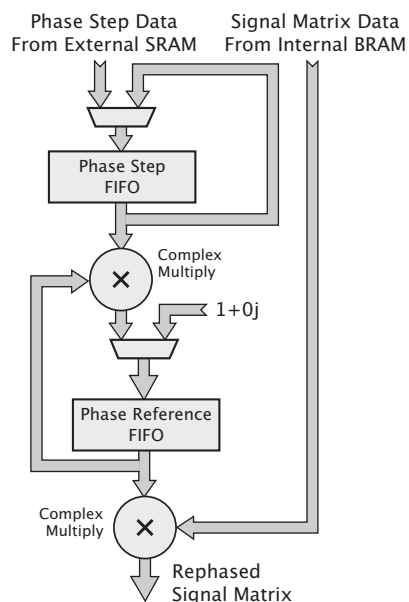


Fig. 4. Rephasing stage decompression and application circuit

The signal matrix is processed beginning with the bottom row and first column. On each clock cycle the column number is advanced by one. After the last element in the current row has been processed, the row number is decremented, such that processing of the next row above may begin. The rephased signal matrix is stored in a FIFO buffer so it may be used when the QR decomposition stage is ready for new data. A diagram of the rephasing processing circuit is shown in Figure 4. It is pipelined such that an m -by- n matrix may be rephased in mn cycles after n phase step vectors have been retrieved from RAM.

C. QRD Stage

The QR decomposition stage moves energy from the sub-diagonal elements of the input matrix to its diagonal elements,

while preserving its singular values. The pseudo-code for this procedure is shown in Algorithm 1. For each column of the matrix, the row that intersects that column at the diagonal is multiplied by the complex exponential that causes the diagonal element's phase-angle to become zero. Then, starting at the bottom of the matrix all rows are rotated in the same manner, such that their elements which lie in the target column have a phase of zero. After each sub-diagonal element is rotated, its magnitude is compared to the magnitude of the diagonal element, using the arctangent operation. Energy from the sub-diagonal element is then moved into the diagonal element using a unitary matrix rotation involving the two rows. These are the operations described in Equations 13-17.

Algorithm 1 Annihilate Sub-diagonal Elements in **A**

```

M = height(A)
N = width(A)
for n = 1 to (N - 1) do
    ϕ = angle(A(n, n))
    A(n, :) = A(n, :)e-jϕ
    for m = M down to (n + 1) do
        ϕ = angle(A(m, n))
        A(m, :) = A(m, :)e-jϕ
        θ = atan( $\frac{|A(m, n)|}{|A(n, n)|}$ )
        L = A(m, :)
        U = A(n, :)
        A(n, :) = Lsin(θ) + Ucos(θ)
        A(m, :) = Usin(θ) - Lcos(θ)
    end for
end for

```

In order to implement this process in a linear array, the algorithm was reformulated so that data may flow through the array. This requires that each processing stage only access matrix elements that are close together, and that computations in each stage occur independently. Fortunately, the sub-diagonal element annihilation task can be localized to two adjacent rows if the elimination processing begins at the bottom of the matrix, row m , and if the k^{th} Givens rotation operation is performed such that it involves moving energy from row $m - (k - 1)$ to row $m - k$. Each processing element may annihilate sub-diagonal elements in a single column, as the matrix passes through it row-by-row, without inter-element dependency. The pseudo-code for a processing element that annihilates elements in column n is described in Algorithm 2.

Matrix rows are processed starting with the bottom row, and progressing upwards. The bottom row is multiplied by a complex exponential, such that its target element has a phase of zero. It is then used to initialize the "feedback" row, F . Each subsequent row is also rephased to zero degrees. After being rephased, a unitary Givens rotation is performed between the new row and the feedback row. The result of the rotation is two output rows. One becomes the feedback row. The other row is part of the output matrix, starting from the bottom, and progressing upwards. When the final target element (which lies on the matrix diagonal) is reached, the last Givens rotation is

Algorithm 2 Annihilate Sub-diagonal Elements in Column n

```

M = height(A)
for m = M down to n do
    if m = M then
        ϕ = angle(A(m, n))
        F = A(m, :)e-jϕ
    else
        ϕ = angle(A(m, n))
        A(m, :) = A(m, :)e-jϕ
        θ = atan( $\frac{|A(m, n)|}{|F(1, n)|}$ )
        L = A(m, :)
        U = F
        F = Lsin(θ) + Ucos(θ)
        A(m + 1, :) = Usin(θ) - Lcos(θ)
    end if
end for
A(n, :) = F

```

performed and the feedback result is deposited on the diagonal. Rows above the final row are part of the upper-triangular result and need not be processed. An input matrix may be streamed, row-by-row, through the processing element that performs this algorithm. The output of the processing element will be a matrix with the same singular values, but in which all sub-diagonal elements in one column have been annihilated. For a matrix with N columns, a linear array of N processing elements may be used to eliminate all sub-diagonal elements.

Because one column is eliminated by each processing element, the number of non-zero elements that must be processed by subsequent processing elements is reduced. This means that some processing elements have a lower processing load than others, which leads to less efficient use of resources in processing elements located towards the end of the array. In order to minimize this effect, processing elements with complementary sized loads share a vector processing unit (see Section IV-C5). For example, if the width of the matrix is 16, then the first processor may be considered fully loaded, with 16 non-zero elements. The second processor, however, is only loaded with 15 non-zero elements. The last processor is loaded very lightly, with 1 non-zero element. Loads of 15 and 1 are complementary because they sum to 16. Therefore, processing elements 2 and 16 share a vector processing unit (VPU). Elements 3 and 15 have complementary loads of 14 and 2, and also share a VPU. The same is true for processing elements 3 and 14, 4 and 13, 5 and 12, et cetera. If the processing elements are arranged properly within the FPGA then the shared resources may be placed close to related logic, and routing distance may be kept short. The appropriate arrangement is a U shape, as shown in Figure 5. Rader [11] called this configuration a folded linear systolic array.

In the SART co-processor, processing elements that share a VPU are grouped into a single "combined element" for implementation. A block diagram of the combined processing element is shown in Figure 6. An additional level of paral-

leism is obtained by pipelining Algorithm 2. That is, the phase and arctangent calculations are performed for the newest input row, while the phase and Givens rotations are applied to the previous row. For this reason, each element consists of multiple stages, which operate in parallel. Rows that need not be processed, such as those above the diagonal, flow through a bypass buffer. The latency of the bypass buffer matches the latency of the pipeline, such that proper data alignment is maintained.

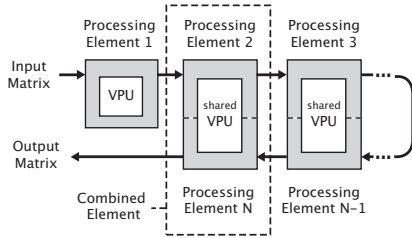


Fig. 5. Elements in the QR decomposition processing array share vector processing unit (VPU) resources in order to improve efficiency. The elements are arranged in a U shape so that elements that share a VPU remain close to each other. This helps to avoid long-distance signal routing.

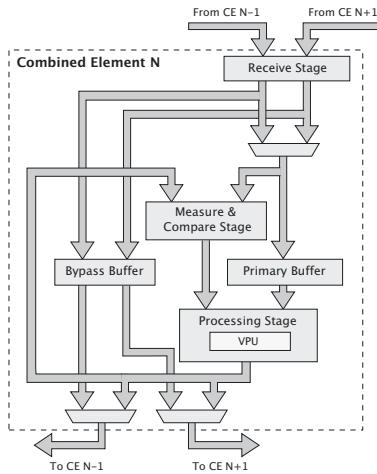


Fig. 6. QR decomposition processing element top-level diagram

1) *Receive Stage*: The receive stage buffers incoming data, from neighboring processing elements, while previously received data are being processed in other stages. By taking advantage of abundant block RAM resources, the receive stage eases scheduling requirements and allows for variable transport latencies so that single chip, or multi-chip processing arrays can be constructed without scheduling changes.

The receive stage also counts each incoming row and—using information about the size of the matrix and the index of the target column—classifies each row by type. Subsequent stages use this type information to determine what kind of processing is required for each row. The row type classifications are: sub-diagonal, diagonal, and super-diagonal. For a processing element that seeks to annihilate subdiagonal elements in column

k , the classification for row j is

$$\text{rowtype} = \begin{cases} \text{subdiagonal} & : j > k \\ \text{diagonal} & : j = k \\ \text{superdiagonal} & : j < k \end{cases} \quad (21)$$

2) *Measure and Compare Stage*: The measure and compare stage consists of two CORDIC modules (see the next section) which compute the unit vectors that are used in the phase rotation, and Givens rotation operations. A simplified diagram of the measure and compare stage is shown in Figure 7. Not shown is the state machine and output logic, which allows the measure and compare stage to control the unloading of new row data, from receive stage, and into the primary buffer. As the first non-zero element from each neighboring processing element is unloaded, it is latched by the measure and compare stage such that its phase and magnitude can be measured. Likewise, the corresponding elements in the feedback row, which are generated by the processing stage, also are latched so that their magnitudes can be compared to the magnitudes of the newly arrived target elements, i.e., so that the arctangent operation from Equation (15) can be performed.

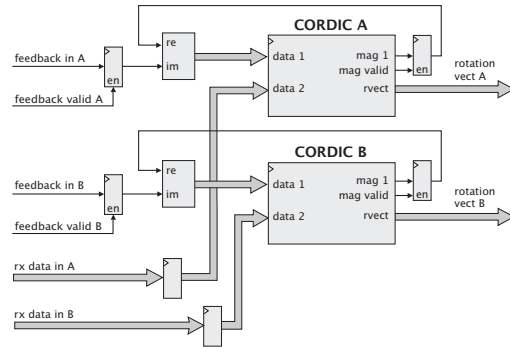


Fig. 7. The measure and compare stage consists of two CORDIC modules that compute the unit vectors used during phase rotations and Givens rotations.

3) *CORDIC Module*: A special CORDIC [12] module was developed to suit the coarse-grain architecture of the Xilinx SX55 FPGA, which contains many compact, high-speed multiply-accumulate (DSP) modules, but a relatively low amount of general-purpose (CLB) fabric. The structure of the CORDIC implementation is shown in Figure 8. Four DSP blocks are used to form two 35-bit shifting circuits. The output of each shifting circuit is fed back to the C_IN port of its lower DSP block. This loop in the pipeline allows for accumulation of the current result with the result of the next shift operation. Addition of the feedback is incorporated into the addition of the partial products from the shift result using the DSP block 3-input adders. The feedback is left-shifted by 17 in its connection to the C_IN port; this compensates for the right shift that occurs as part of the partial product summation. The output of each shifter is also connected to the

input of the opposite shifter, in order to supply the operand of the next shift/accumulate operation. The subtraction control signals are generated from the signs of the accumulation results, and used to specify rotation direction. The latency of the shifter/add operation is 4 cycles, however, a micro-rotation direction decision cannot be made until the result of the previous micro-rotation exits the pipe. Therefore, the pipe must be filled with 4 independent operands in order to be used efficiently.

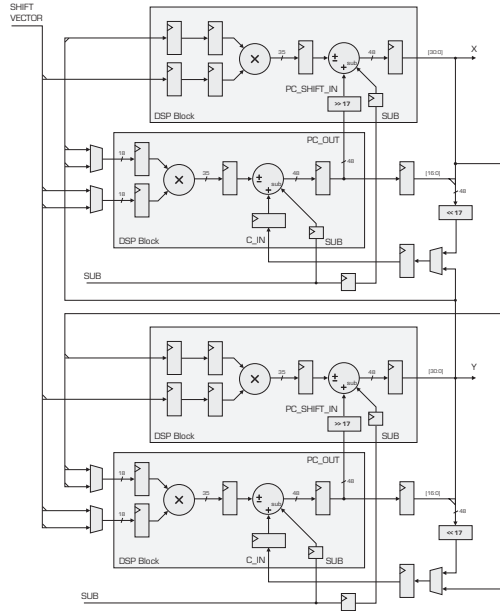


Fig. 8. CORDIC rotation circuit implemented using four DSP blocks.

Each CORDIC module accepts two input vectors. Two of the pipeline slots are used to rotate these vectors to a phase-angle of zero degrees. The operands for the remaining two processing slots are two constant vectors, whose initial phases are zero. The lengths of these vectors are equal to the inverse of the CORDIC gain factor, so that they will have unit-length after rotation. As the arbitrary input vectors are rotated to zero, the constant vectors are rotated in the same direction, and by the same amount. The result is two unit length vectors, whose angles are equal and opposite to those of the arbitrary input vectors. These unit vectors are used to perform the phase and Givens rotations in the processing stage of the QR decomposition processing element.

Two additional DSP slices are used as wide multiplexers for loading operands into each of the shifters. This configuration is shown in Figure 9. These blocks are also used to apply sign changes to the vector component inputs, as part of the coarse rotation stage in the CORDIC algorithm. Swapping of vector components (also part of coarse rotation) is accomplished using the C_IN input multiplexer shown in Figure 8.

4) *Processing stage (VPU stage)*: The processing stage is responsible for performing the vector operations associated with phase rotations and Givens rotations. These operations have been designated as

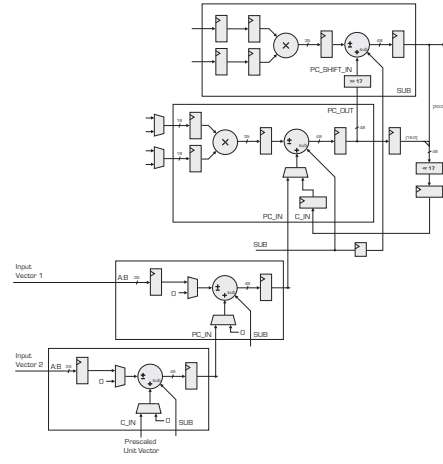


Fig. 9. Inputs to the CORDIC circuit are selected using a wide multiplexer implemented with two DSP slices.

- Rotation:

$$\Re\{\vec{a}_j'\} = \Re\{u_j\}\Re\{\vec{a}_j\} - \Im\{u_k\}\Im\{\vec{a}_j\} \quad (22)$$

$$\Im\{\vec{a}_j'\} = \Re\{u_j\}\Im\{\vec{a}_j\} + \Im\{u_k\}\Re\{\vec{a}_j\} \quad (23)$$

- Output:

$$\Re\{\vec{a}_j''\} = \Re\{u_0\}\Re\{\vec{a}_j'\} + \Im\{u_0\}\Re\{\vec{a}_k'\} \quad (24)$$

$$\Im\{\vec{a}_j''\} = \Re\{u_0\}\Im\{\vec{a}_j'\} + \Im\{u_0\}\Im\{\vec{a}_k'\} \quad (25)$$

- Feedback:

$$\Re\{\vec{a}_k''\} = \Re\{u_0\}\Re\{\vec{a}_k'\} - \Im\{u_0\}\Re\{\vec{a}_j'\} \quad (26)$$

$$\Im\{\vec{a}_k''\} = \Re\{u_0\}\Im\{\vec{a}_k'\} - \Im\{u_0\}\Im\{\vec{a}_j'\} \quad (27)$$

$\Re\{\vec{a}_j\}$ indicates the real parts of elements in row vector \vec{a}_j , $\Im\{\vec{a}_j\}$ indicates imaginary components, and u_0 represents the unit vector that describes the rotation angle. Note that the output and feedback operations together comprise a Givens rotation.

Within the processing stage, the results of these operations are computed sequentially. That is, to process a row with n elements, $3n$ clock cycles are required. This splitting of the Givens rotation into two parts results in a better balance with the processing requirements of the phase rotation operation, which requires half as many computations as the Givens rotation. The structure of the processing stage is shown in Figure 10. The vector processing unit (VPU) performs the arithmetic operations listed above, and is described in the next section.

Computations within the processing stage are triggered by the assertion of the “newrow” signal, which causes the state machine to move from the idle state to the feedback state. For every cycle that the state is not equal to the idle state, the column counter is incremented. When the count reaches the number of columns in the signal matrix, *width*, the counter rolls over, and the state is updated according to the state-flow diagram in Figure 11.

In order to achieve the resource sharing of the VPU, each of the processing states is divided into two time slots, A and

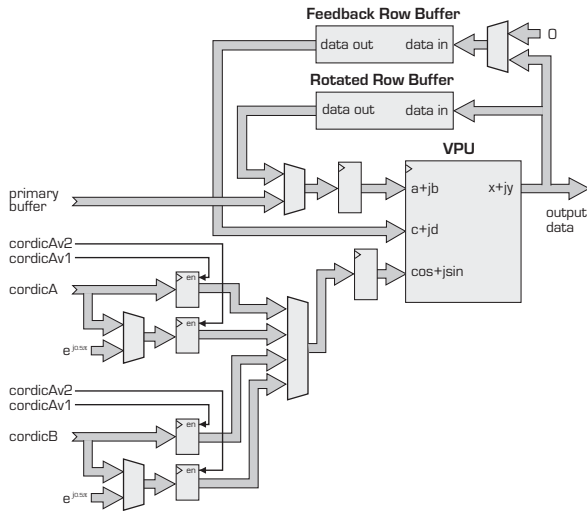


Fig. 10. Processing stage of the QR decomposition processing element

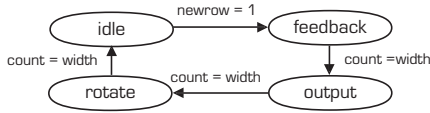


Fig. 11. State-flow diagram for the processing stage

B, during which data from the two neighboring processing elements are processed. This is illustrated in Figure 12.

When the processing stage is in the feedback state, partial Givens rotation operations are performed using the row data contained in the “rotated row” and “feedback row” buffers connected to the $a + jb$ and $c + jd$ input ports of the VPU. The $\cos + jsin$ port is connected to the Givens rotation unit vectors, which have been generated by the measure and compare stage, and registered internally. For time slot A, the result produced by CORDIC module A is used. For time slot B, the result from CORDIC module B is used. The VPU is set to compute Equations 26 and 27, and the results are queued in the feedback buffer. When a diagonal row for A or B has been processed, the $\cos + jsin$ port is connected to a unit vector with phase angle of 90 degrees during the appropriate time slot. This causes the feedback row to be deposited on the matrix diagonal as described in Algorithm 2.

The output state is very similar to the feedback state. The same unit vectors are used, and the remaining portion of the Givens rotation is computed. The only change is the operation of the VPU, which is set to compute Equations 24 and 25. The target elements of the resulting A and B rows are zero. These rows are passed directly to subsequent elements in the QR decomposition processing array.

When the processing stage is in the rotate state, the newly arrived row data are rephased such that the A and B target elements have phase angles of zero. This is accomplished by selecting the unit vectors generated by the A and B CORDIC module in the measure stage. The rephased data is stored in the “rotated row” buffer as

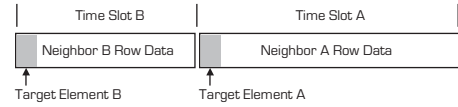


Fig. 12. Resource sharing schedule for the processing stage. Row data from neighboring processing elements, A and B, present complementary loads, which are combined to achieve better load balancing across all processing elements.

it exits the VPU, and will be used for the next Givens rotation.

5) *Vector Processing Unit (VPU)*: The sum-of-products operations described in Equations 22-27 are implemented using the structure shown in Figure 13. Each 35-bit product is implemented using four DSP blocks, with each block generating a partial product, which is summed with other partial products using the dedicated cascade path in the Virtex device. To avoid use of a ninth adder, for summing the two full products, this summation is performed as part of the partial product summation. The signs of each term in the sum are controlled using the DSP block subtract signals. Two of these structures comprise the VPU. Multiplexers are used to select the appropriate input operands and signs, given the current op-code input.

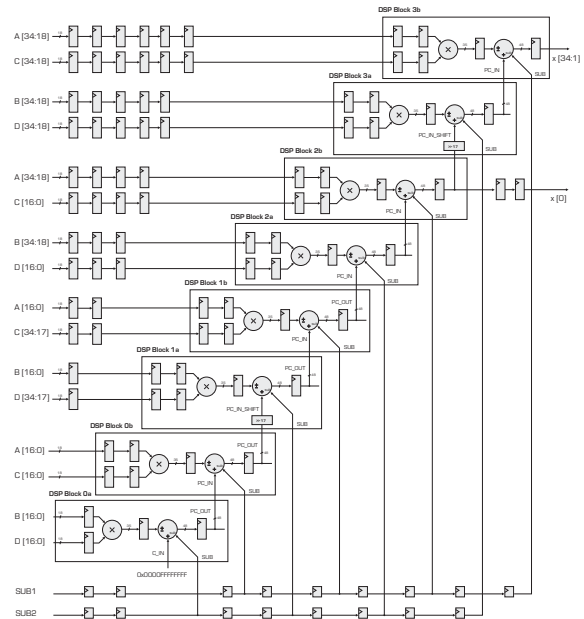


Fig. 13. Sum of products implemented using eight DSP blocks

D. Bidiagonalization Stage

The final stage in the SART co-processor is the bidiagonalization stage, which further reduces the size of matrices that have been upper-triangularized by the QR decomposition stage. It is similar to the QR decomposition stage in that it uses the same CORDIC and VPU modules. However, due to flow dependencies, the bidiagonalization stage is not implemented as a linear processing array. Instead, multiple independent bidiagonalization modules are used. Each module

stores matrix data locally, and conducts column and row annihilations sequentially. Like the QR decomposition stage, each bidiagonalization module annihilates matrix elements by moving energy to elements on the primary diagonal, or to elements just above the diagonal. Figure 2 shows the order in which sub-diagonal and super-diagonal elements are annihilated, and to which element their energy is moved. Figure 14 shows how matrix data flow between memory structures within the bidiagonalization module, for each operation type. Each time a element that lies on either of the two diagonals is calculated, it is saved to a result buffer. The host PC may read the results generated by all bidiagonalization modules, and conduct the final diagonalization using LAPACK routines.

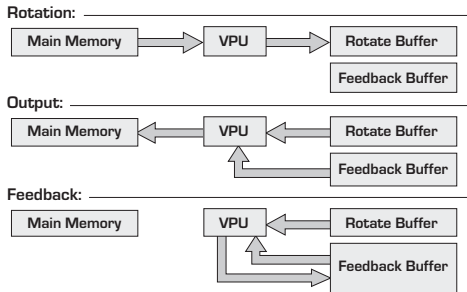


Fig. 14. Flow of data between memory structures within the bidiagonalization module

V. PERFORMANCE AND FUTURE WORK

The SART co-processor architecture described here was synthesized, placed, and routed using the Xilinx toolchain (ISE v9.1.03i). The clock frequency for the system was 200 MHz. For the current implementation, at a matrix size of 128-by-16, this results in a throughput of about 41.67 thousand matrices per second. For comparison, a Pentium 4 CPU running at 3 GHz, executing LAPACK routines under Windows XP Pro, is capable of processing about 7 thousand matrices without the coprocessor. Therefore, use of the co-processor results in a speedup of about 6. It is also important to note that the size of the processing array scales with the matrix width, n ; greater speedups will be obtained when larger matrices are processed because the processor reduces the $O(mn^2)$ SVD task into a $O(mn)$ task. Still higher performance can be obtained by using multiple FPGAs in parallel. The resource usage of the current implementation is detailed in Table II. The accuracy of the

TABLE II
RESOURCE USAGE OF SART CO-PROCESSOR

Resource Type	Usage	Percent of SX55 FPGA
Logic Slices	20,028	81%
Flip-Flops	26,378	53%
DSP Blocks	384	75%
BRAM	214	67%

system is 25-bit fixed point, and is limited by accumulated truncation errors in the CORDIC module [13]. This is an area for future improvement. However, this level of accuracy is

actually slightly better than that obtained with single precision floating point calculation because precise calculation of the first singular is limited by mantissa precision, which limits the representation of small fluctuations in a large value.

Future implementations of the co-processor will be fully parameterized, so that any matrix dimensions may be supported. This will not be trivial, because larger processing arrays must span multiple FPGAs. Fortunately the system was designed with this in mind; the linear nature of the array results in minimal requirements for inter-IC data transfer rates.

VI. CONCLUSION

The architecture of an FPGA-based co-processor, developed for accelerating the singular value array reconciliation tomography (SART) algorithm, was described. The SART algorithm was mapped into a Xilinx SX55 device in order to make efficient use of all available resources. Pipelining techniques were used at multiple levels of granularity in order to allow processing stages within the SART and SVD algorithms operate concurrently without violating dependencies. Buffering between stages was used to provide tolerance to changes in data transport latency, such that a multi-IC processing array can be created without scheduling changes, regardless of interconnect latency. It was shown that a single FPGA running at 200 MHz can provide a processing throughput that is six times greater than that of a 3 GHz Pentium 4 machine with no co-processor. This co-processor will enable the development of a small, low power, SART-based tracking system for mobile search and rescue applications.

REFERENCES

- [1] The Precision Personnel Locator Project Website. <http://www.ece.wpi.edu/research/ppl>, WPI, 2008.
- [2] B. Alavi and K. Pahlavan. Modeling of the TOA-based Distance Measurement Error Using UWB Indoor Radio Measurements. *IEEE Communications Letters*, 10, April 2006.
- [3] D. Cyganski; et al. Patent application 60/934,880 - Singular Value Array Reconciliation Tomography, 2007.
- [4] D. Cyganski; et al. WPI Precision Personnel Location (PPL) System. In *Institute of Navigation, 63rd Annual Meeting*, Cambridge, MA, April 2007.
- [5] V. T. Amendolare. Synchronization in an Indoor Precision Location System. Master's thesis, Worcester Polytechnic Institute, Worcester, MA, 2007.
- [6] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, MD, 1996.
- [7] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224, 1965.
- [8] M. Gu and S. C. Eisenstat. A divide-and-conquer algorithm for the bidiagonal svd. *SIAM J. Matrix Anal. Appl.*, 16(1):79–92, 1995.
- [9] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297–301, April 1965.
- [10] M. Cornea; J. Harrison; P. T. P. Tang. Intel itanium floating-point architecture. White paper, Intel Corporation, 2003.
- [11] C. M. Rader. Patent 4972361- folded linear systolic array, 1988.
- [12] J. E. Volder. The cordic trigonometric computing technique. *IRE Transactions on Electronic Computers*, EC-8:330–334, 1959.
- [13] J. W. Coyne. FPGA-Based Co-processor for Singular Value Array Reconciliation Tomography. Master's thesis, Worcester Polytechnic Institute, Worcester, MA, 2007.