Application-Specific Instruction Set Processor for SoC Implementation of Modern Signal Processing Algorithms

Zhaohui Liu, Kevin Dickson, and John V. McCanny, *Fellow, IEEE*

***Abstract*—A novel application-specific instruction set processor (ASIP) for use in the construction of modern signal processing systems is presented. This is a flexible device that can be used in the construction of array processor systems for the real-time implementation of functions such as singular-value decomposition (SVD) and QR decomposition (QRD), as well as other important matrix computations. It uses a coordinate rotation digital computer (CORDIC) module to perform arithmetic operations and several approaches are adopted to achieve high performance including pipelining of the micro-rotations, the use of parallel instructions and a dual-bus architecture. In addition, a novel method for scale factor correction is presented which only needs to be applied once at the end of the computation. This also reduces computation time and enhances performance. Methods are described which allow this processor to be used in reduced dimension (i.e., folded) array processor structures that allow tradeoffs between hardware and performance. The net result is a flexible matrix computational processing element (PE) whose functionality can be changed under program control for use in a wider range of scenarios than previous work. Details are presented of the results of a design study, which considers the application of this decomposition PE architecture in a combined SVD/QRD system and demonstrates that a combination of high performance and efficient silicon implementation are achievable.**

***Index Terms*—Application-specific instruction set processor (ASIP), coordinate rotation digital computer (CORDIC) processors, modern signal processing, QR decomposition (QRD), singular-value decomposition (SVD), system on chip (SoC).**

I. Introduction

OVER THE PAST decade or more, extensive research has been devoted to the development of modern signal processing algorithms and methods which have widespread potential provided these can be implemented in real-time using cost effective hardware/software solutions. Examples of the algorithms are ones that involve matrix algebraic methods such as QR decomposition (QRD) and singular-value decomposition (SVD). Such methods are increasingly used in applications such as direction estimation, spectrum analysis and recursive least squares (RLS) filtering. The real-time, computational complexity of such algorithms tends to be high, usually significantly greater than more conventional [e.g., fast Fourier transform (FFT), finite-impulse response (FIR) filter based] techniques, in many cases reaching the limits of what is achievable with current technology. As a consequence, considerable research has also been undertaken into parallel architectures and systematic methodologies for mapping matrix algorithms onto such architectures. Examples include the early work by Gentleman and Kung [1] and by McWhirter *et al.* [2], [3] on triangular systolic array architectures for QRD. Considerable effort has also been devoted to SVD architectures, including the work done by Luk *et al.* [4]–[6]. Indeed, many modern signal processing algorithms are heavily dependent on these two Jacobi-rotation based matrix factorizations, i.e., QRD and SVD.

Research has also been undertaken on more detailed architectural aspects relating to the design of processor elements suitable for constructing such systems. An important contribution has been the work of Van Dijk *et al.* [7]. They recognized that a Jacobi processor based on a coordinate rotation digital computer (CORDIC) arithmetic element can be used in a variety of such applications and described systems based around the concept of a board containing processor chips connected to a PC/workstation. This involves the use of a host program to control this multiprocessor system and reconfigurable interconnections to allow the underlying array to be configured to suit different applications.

Related research undertaken in this laboratory and in collaboration with others [8]–[10] has been concerned with the detailed design and implementation of system-on-chip (SoC) architectures, the prime focus being array processors for QRD. Designs described based on both application-specific integrated circuits (ASIC) and field-programmable gate array (FPGA) technologies have been reported. This has been motivated by the realization that the use of algorithms based on such methods will start to see much more widespread practical application once they can be implemented cost effectively in silicon. Our recent research has therefore involved finding efficient methods for mapping triangular QR array processors onto linear arrays [8], the development of generic SoC architectures for this, the establishment of generic timing methods and the demonstration of these concepts through SoC design and implementation [9], [10]. This has been based on the use of more conventional arithmetic based processors. The generic approach developed has the attraction of allowing QR processor designs with different specifications (i.e., matrix size, wordlengths) to be created very rapidly through synthesis from a structured very

Manuscript received September 22, 2003; revised August 23, 2004. This work was supported in part by the Northern Ireland Special Universities Research (SPUR 2) programme as well as funding for a studentship from the Department of Employment and Learning. This paper was recommended by Associate Editor K. Chakrabarty.

The authors are with the Institute of Electronics, Communications and Information Technology (ECIT), Queen's University Belfast, Belfast BT3 9DT, Northern Ireland, U.K. (e-mail: z.liu@ecit.qub.ac.uk; k.dickson@ecit.qub.ac.uk; j.mccanny@ecit.qub.ac.uk).

Digital Object Identifier 10.1109/TCSI.2005.844109

high-speed integrated circuit (VHSIC) Hardware Description Language (VHDL) description. However, if fabricated in an ASIC technology, these implementations become fixed and are not particularly flexible.

The purpose of this paper is to extend the research described by combining various elements to create a new application-specific instruction set processor (ASIP) for implementing SoC processor blocks that can be used to construct real-time systems for modern signal processing computations. This is much more flexible than the generic approach previously described [9], [10]. In doing so, and consistent with the work of Van Dijk *et al.* [7], this exploits similarities in the QRD and SVD algorithms, including the fact that both are based on Givens rotations and uses a CORDIC approach to implement the internal computational unit. This combination, incorporating programmability, provides a flexible alternative to more dedicated solutions. As will be discussed, the processor presented incorporates facilities for the issue of parallel instructions and a dual-bus architecture designed to achieve high performance. In addition, a new method for scale factor correction (SFC) within the CORDIC module is introduced which needs only to be applied once at the end of the SVD/QRD computation. This reduces computation time and enhances performance. Pipelining of the internal recursive loop within the CORDIC arithmetic module is also exploited to allow two independent micro-rotations to be naturally multiplexed onto a single piece of hardware, with ensuing benefits in terms of higher performance for the same silicon area.

The paper also describes how well-known systolic structures for SVD/QRD can be constructed using arrays of these ASIP processors, including mappings to reduced array sizes. The paper also describes methods for the redistribution of computations between neighboring processors to balance computational load and also reduce computation time. RTL-based silicon design studies have also been undertaken, which clearly demonstrate the viability of the approach in terms of modern SoC implementation.

The structure of the paper is as follows. Section II provides a summary of SVD and QRD computations and the use of floating-point CORDIC algorithms for implementing functions such as Givens rotations. The new SFC method is presented in Section III with details of the proposed ASIP architecture created described in Section IV. Section V then shows how SVD and QRD computations can be mapped onto arrays of these ASIPs, including methods for balancing computational load. The results of design studies are then given in Section VI, with a discussion of the work and the conclusions that can be drawn provided in Section VII.

II. CORDIC Algorithm for SVD and QRD

It is well known that both SVD and QRD are related to Jacobi-type methods for parallel computations. For SVD, the Kogbetliantz method (involving two-sided rotation) is favored because it is highly suitable for mapping onto a regular rectangular systolic array architecture [4], [5]. In the Kogbetliantz method, the norm of the off-diagonal elements of the matrix is successively reduced by a sequence of two-sided Givens transformations; this requires the computation of a rotation angle and subsequent operations. As with SVD, the QRD algorithm uses a sequence of Givens rotations to transform the incoming data matrix into an upper triangular matrix. The CORDIC algorithm [11], [12] provides an attractive means for implementing the arithmetic units required in typical SVD/QRD processing elements (PEs) as these enable the efficient implementation of plane rotation and phase computation [13]–[17].

Using the CORDIC algorithm allows a rotation to be decomposed into a sequence of micro-rotations over the angles α_i, where α_i is usually chosen as $\tan\alpha_i = 2^{-i}$. The corresponding micro-rotation, iterative equations [11] are then

$$x_{i+1} = x_i + \delta_i y_i 2^{-i} \tag{2.1a}$$

$$y_{i+1} = y_i - \delta_i x_i 2^{-i} \tag{2.1b}$$

$$z_{i+1} = z_i - \delta_i \tan^{-1}(2^{-i}) \tag{2.1c}$$

where $\delta_i = \pm 1$.

The inverse tangent is a primitive operation in the CORDIC algorithm. Therefore, the rotation angles can be solved explicitly in (2.1) by setting $\delta_i = sign(x_i y_i)$. Moreover, in the CORDIC rotation mode (2.1) can be used to perform matrix-vector multiplication (apart from the constant scale factor) as shown in (2.2)

$$\begin{pmatrix} \widetilde{x} \\ \widetilde{y} \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}. \tag{2.2}$$

If $\delta_i = sign(z_i)$ and x_n and y_n are the final outputs from (2.2), then the desired outputs $\widetilde{x}$ and $\widetilde{y}$ in (2.2) are given by

$$\widetilde{x} = k_n x_n \tag{2.3a}$$

$$\widetilde{y} = k_n y_n \tag{2.3b}$$

where scale factor k_n is given by

$$k_n = \prod_{i=0}^{N-1} \cos\alpha_i = \prod_{i=0}^{N-1} \frac{1}{\sqrt{1+2^{-2i}}} \tag{2.3c}$$

and N is the word length.

In this paper, we consider the case of a hybrid system in which x and y are floating-point values and z is a fixed-point number. This scheme can lead to a reduction in computation time while maintaining the dynamic range; albeit at the expense of floating-to-fixed point and fixed-to-floating point conversion (see Fig. 1). This is due to the fact that the required number of CORDIC iterations is dependent on the wordlength used, and in this case the mantissa wordlength is typically significantly less than the fixed-point equivalent. For example, a typical QRD-based beamformer used in an advanced RADAR application requires a 26-bit wordlength to maintain the dynamic range if fixed-point arithmetic is used. This compares with the use of a 6-bit exponent and a 14-bit mantissa floating-point system to achieve the same performance [18]. The latter values have therefore been used as the basis for the case study described later.

III. Scale Factor Correction

As shown by (2.3), an implicit feature of the CORDIC algorithm is that rotations include a scale factor, which needs to be

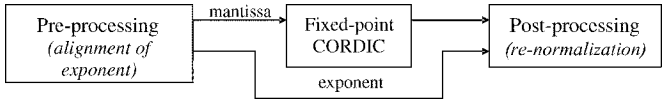


Fig. 1. Floating-point CORDIC.

corrected in order to achieve "perfect" rotations. First, we describe a novel method to derive this for SVD and then extend it to QRD.

A. SFC for SVD

Existing methods [6] carry out this SFC after each two-sided rotation. The time required is in the region of $T_c/4$, where T_c is the total time required to complete a CORDIC iteration. This therefore can impede overall performance. The new method described below postpones this until after the last sweep of the SVD algorithm (a sweep is defined as each off-diagonal element being eliminated once). This therefore achieves an important reduction in total computation time. This can be illustrated by an example.

The basis of the Kogbetliantz method [19] is to apply a two-sided Givens rotation to each 2×2 submatrix to nullify the two off-diagonal elements i.e.,

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}^T \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix} = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix}. \tag{3.1}$$

This can also be written as

$$\mathbf{U}(\theta)\mathbf{A}\mathbf{V}(\phi) = \widetilde{\mathbf{A}} \tag{3.2}$$

where $\mathbf{A}$ is a 2×2 matrix and $\widetilde{\mathbf{A}}$ is the desired output, and $\mathbf{U}(\theta)$ and $\mathbf{V}(\phi)$ are unitary matrices corresponding to θ and ϕ. The rotation parameters θ and ϕ are obtained using the following:

$$\begin{cases} \phi + \theta = \tan^{-1}\frac{c+b}{d-a} \\ \phi - \theta = \tan^{-1}\frac{c-b}{d+a.} \end{cases} \tag{3.3}$$

If implemented using CORDIC rotations, then, (3.2) becomes

$$\mathbf{U}(\theta)\mathbf{A}\mathbf{V}(\phi) \Rightarrow \frac{\widetilde{\mathbf{A}}}{k_n^2} \tag{3.4}$$

where k_n is the scale factor given in (2.3c). Consider the example of a 4×4 matrix, which can be partitioned into four $2 \times$ submatrices, of the form

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{pmatrix}. \tag{3.5}$$

After the rotation angles for $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ have been calculated, i.e., (θ_1, ϕ_1) and (θ_2, ϕ_2), respectively, then CORDIC rotations can be applied to the four submatrices. This is shown in Fig. 2. It is clear that all submatrices are scaled by the same value and hence the SFC can be postponed so that it only needs to be applied once at the end.

However, one issue that must be addressed when doing this is that the magnitude of these values can grow significantly, thus requiring very large wordlengths to represent them. For instance, in the case of a 14-bit mantissa, $k_{14}^2 = 0.3688$. If we

1. Calculating angles in the CORDIC vectoring (angle calculation) mode:
   θ_1 and ϕ_1 for A_{11}
   θ_2 and ϕ_2 for A_{22},
2. Applying two-sided rotations in the CORDIC rotation mode:
   Applying θ_1 and ϕ_1 to A_{11}: $U(\theta_1)A_{11}V(\phi_1) \Rightarrow \widetilde{A}_{11}/k_n^2$
   Applying θ_2 and ϕ_2 to A_{22}: $U(\theta_2)A_{22}V(\phi_2) \Rightarrow \widetilde{A}_{22}/k_n^2$
   Applying θ_1 and ϕ_2 to A_{12}: $U(\theta_1)A_{12}V(\phi_2) \Rightarrow \widetilde{A}_{12}/k_n^2$
   Applying θ_2 and ϕ_1 to A_{21}: $U(\theta_2)A_{21}V(\phi_1) \Rightarrow \widetilde{A}_{21}/k_n^2$

Fig. 2. SVD of a 4×4 matrix using the CORDIC algorithm.

then consider a 20×20 matrix involving 5 SVD sweeps then the accumulated scale factor is 7×10^{-42}. This indicates that the dynamic range of the data and thus the wordlength of the exponent would need to be extended significantly to cope with this.

This issue can be addressed by ensuring that the scale factor is normalized to a value between 0.5 and 1.0 after each two-sided rotation so as to maintain the dynamic range. It can be proven (see Appendix) that for any n-bit mantissa, $2^{-1/2} > k_n > 2^{-1}$, and so $2^{-1} > k_n^2 > 2^{-2}$, i.e., after each two-sided rotation, the dynamic range can be maintained by dividing by either 2 or 4. This can be implemented by subtracting the value d (= 1 or 2) from the x (or y) exponent value. The subtraction of this value d can be determined in advance, from the mantissa word length, matrix dimensions, and number of sweeps.

This can be illustrated using the following example.

- After the first two-sided rotation the scale factor $\varsigma_1 = k_{14}^2 = 0.3688 \equiv 2^{-1}(0.7376) \equiv 2^{-1}\varsigma_1'$ i.e., ς_1' is in the range 0.5 to 1.0, with the multiplication by 2^{-1} achieved by subtracting 1 from the exponent.
- After the second two-sided rotation the scale factor $\varsigma_2 = k_{14}^2\varsigma_1' = 0.3688(0.7376) \equiv 2^{-1}(0.5441) \equiv 2^{-1}\varsigma_2'$ i.e., ς_2' is again in the range 0.5 to 1.0, with 1 again subtracted from the exponent.
- However, after the third two-sided rotation the scale factor $\varsigma_3 = k_{14}^2\varsigma_2' = 0.3688(0.5441) \equiv 2^{-2}(0.8027) \equiv 2^{-2}\varsigma_3'$ and thus in order to maintain ς_3' within the range 0.5 to 1.0, the value 2 must be subtracted from the exponent.

The final SFC can then be performed as a series of shifts and adds, at the last step of an SVD computation. The time required to do this is generally negligible compared with the total computation time.

B. SFC for QRD

QRD performs a set of Givens rotations on an incoming data matrix transforming it into an equivalent upper triangular matrix, i.e.,

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} \beta r_i & \beta r_k \\ x_i & x_k \end{pmatrix} = \begin{pmatrix} \widetilde{r}_i & \widetilde{r}_k \\ 0 & \widetilde{x}_k \end{pmatrix}. \tag{3.6}$$

In the case of QRD-RLS based adaptive beamforming, it is usually the case that the desired output is the posterior residual

$$\widetilde{e}(t) = \widetilde{\gamma}(t)\tilde{\eta}(t) \tag{3.7}$$

where

$$\widetilde{\gamma}(t) = \gamma_0(t) \prod_{i=1}^{p-1} \cos\theta_i(t) \tag{3.8}$$

and $\widetilde{\eta}(t)$ is the desired output of $PE_{p-1,p}$ where p is array size. If the CORDIC algorithm is used for rotations, then

$$\widetilde{\gamma}(t) = k_n^{p-1}\gamma(t) \tag{3.9}$$

and due to an accumulation of the scale factor in the $\mathbf{R}$-matrix over time, the input data vector $\mathbf{x}$ must be appropriately scaled before processing by $1/k_n^t$, hence

$$\widetilde{\eta}(t) = k_n^{p-1}\eta(t) \tag{3.10}$$

where $\widetilde{\gamma}(t)$ and $\widetilde{\eta}(t)$ are the outputs of the CORDIC rotations. Thus, the posterior residual is

$$\widetilde{e}(t) = k_n^{2(p-1)}\gamma(t)\eta(t). \tag{3.11}$$

Setting

$$\gamma_0 = \frac{1}{k_n^{2(p-1)}} \tag{3.12}$$

then $\widetilde{e}(t)$ can be corrected without any extra operations. However, to prevent the $\mathbf{R}$ from growing in an unlimited fashion, it is still necessary to subtract from the exponent value in order to maintain the dynamic range. In this case, because QRD involves one-sided rotations, the subtraction of the value d (= 1 or 2) from the exponent value can be executed every other set of new sample data.

IV. Processor Architecture

A. CORDIC Module Architecture

The floating-point CORDIC module we have developed comprises of a pre-processing unit (alignment of exponents), a post-processing unit (re-normalization) and a fixed-point CORDIC unit. To achieve a high throughput rate, this is pipelined, as shown in Fig. 3. The fixed-point CORDIC unit is constructed using a shift and add circuit.

Assuming a clock period T and mantissa wordlength N, then, for SVD computation (ignoring the time for data preparation i.e., $d \pm a$ and $c \pm b$ etc.), the total time required, within a single CORDIC module, to compute the two-sided rotation in (3.1) and the angle solving operation of (3.2) is $6NT$. If higher performance is required, the fixed-point CORDIC unit becomes a bottleneck since it is based on iterations. However, performance can be enhanced by also pipelining each micro-rotation using two clock cycles. Since the $(i+1)$th iteration is dependent on the output of the ith iteration, the introduction of two levels of pipelining naturally allows two independent computations to be directly multiplexed onto the same piece of hardware, thus making it suitable for the parallel computation of two different CORDIC rotations. For SVD, the left-sided rotation is

$$\begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{pmatrix} a' & b' \\ c' & d' \end{pmatrix} \tag{4.1}$$

Fig. 3. Pipelined floating-point CORDIC module.

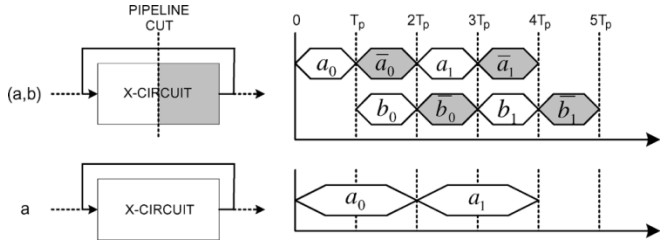


Fig. 4. Throughput rate achieved by pipelining.

which can be decomposed into two independent CORDIC rotations that can be computed in parallel, i.e.,

$$\begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} a \\ c \end{pmatrix} = \begin{pmatrix} a' \\ c' \end{pmatrix} \tag{4.2}$$

and

$$\begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} b \\ d \end{pmatrix} = \begin{pmatrix} b' \\ d' \end{pmatrix} \tag{4.3}$$

The right-sided rotation can also be performed in a similar manner, with the two rows $(a\ b)$ and $(c\ d)$ being computed in parallel. Thus hardware sharing can be achieved in accordance with the following data input schedule.

For left-multiplication

$$(a_i, c_i), (b_i, d_i), (a_{i+1}, c_{i+1}), (b_{i+1}, d_{i+1}), \cdots$$

For right-multiplication

$$(a_i, b_i), (c_i, d_i), (a_{i+1}, b_{i+1}), (c_{i+1}, d_{i+1}), \cdots$$

Here, the times required to compute both the left- and right-multiplication are both $2NT_p$, where T_p is the clock cycle for the case of pipelined micro-rotation. In a similar manner, the angles $\phi + \theta$ and $\phi - \theta$ can also be calculated in parallel. The total time required to perform the computation given by (3.1) is therefore $6NT_p$, which is the same for the architecture in Fig. 3. However, because T_p is nearly half of T, the throughput rate is nearly doubled. Thus, by exploiting this parallel approach a higher performance can be achieved but using roughly the same hardware. Fig. 4 illustrates the performance improvement.

In the case of QRD $r_{i,j}$ and $r_{i,j+1}$ can be rotated in parallel in the same way. Hence, the method described applies to both SVD and QRD.

Fig. 5 presents a high-level schematic of the CORDIC architecture developed, with each micro-rotation pipelined over two clock cycles. Interaction of data between the blocks can also be seen. The existence of the two angle registers (labeled 'Reg Z1' and 'Reg Z2') correspond to the two independent computations.

As mentioned in Section III-A, the final SFC can be performed as a series of shifts and adds. Therefore, the CORDIC architecture, which is based on shifts and adds, can be reused to carry out this SFC. The subtraction of the value d (= 1 or 2)

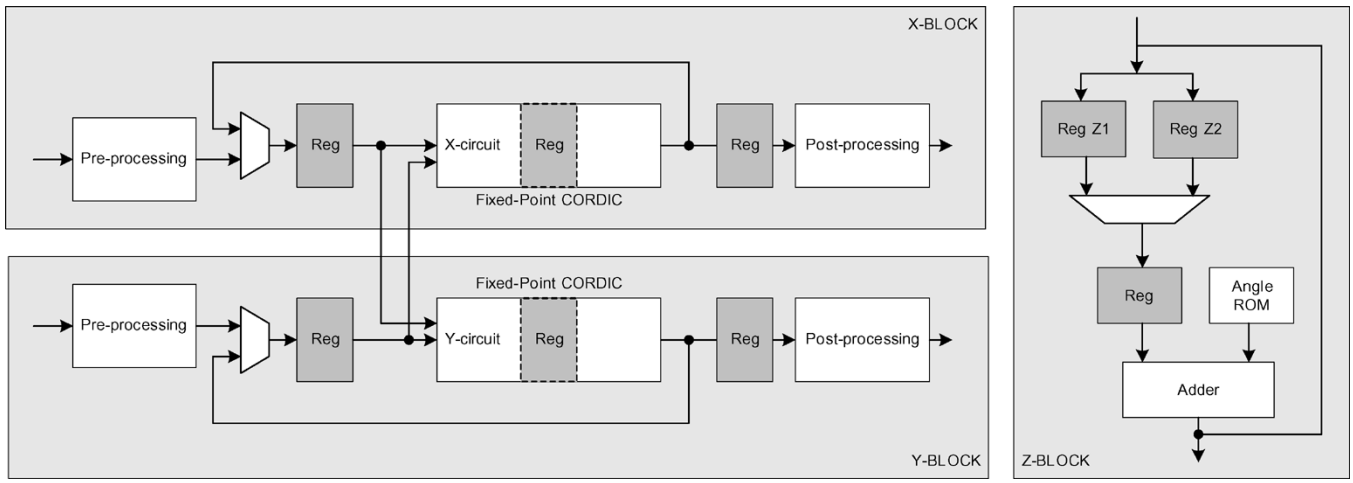


Fig. 5. CORDIC architecture.

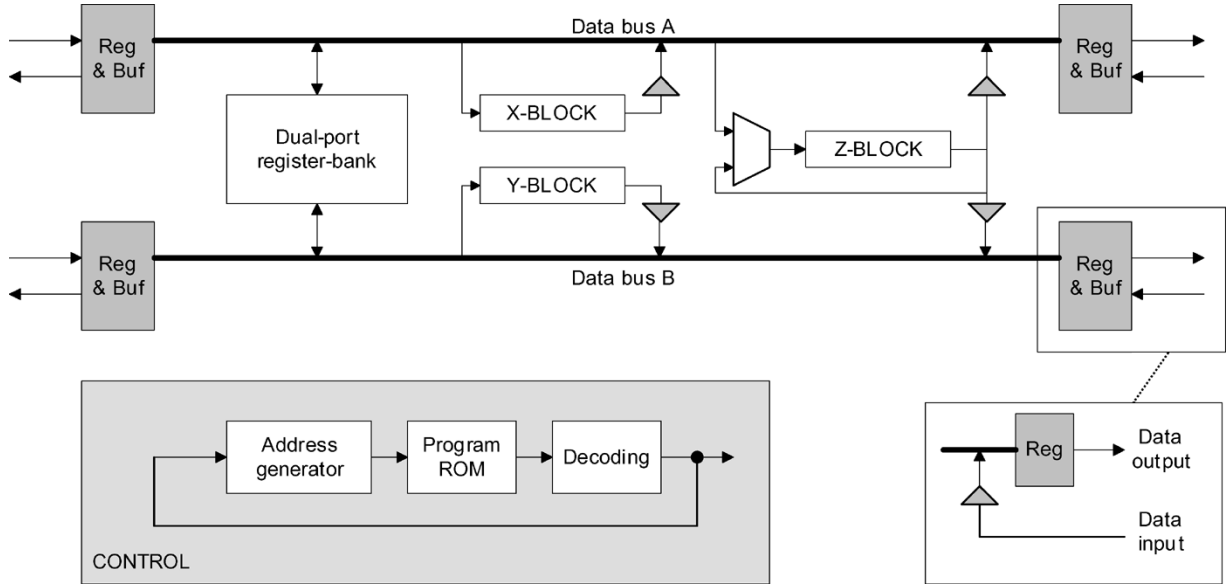


Fig. 6. Programmable processor architecture.

from the exponent values required after each two-sided rotation is carried out in the post-processing unit.

B. Programmable Processor

A programmable processor architecture, which can be used for both QRD and SVD computations, which incorporates the aforementioned techniques, is shown in Fig. 6. Here, each PE contains a CORDIC module of the type described in Fig. 5 with this used for angle solving and rotation. The overall operation of the processor is controlled through a program. Program and data memory are separated to provide parallel instruction and data streams. The processor also incorporates two data-buses and a dual-port register file (data memory) to allow concurrent operations to be performed i.e., two operands can be fed to the data stream in one cycle. The detailed operations performed by the CORDIC module are controlled by using different operation modes. These are defined by assembly code. Data is exchanged between processors using the registers and buffers ("Reg & Buf") with details as illustrated. Here data is held in the registers, ready to be accessed by a neighboring processor. It is also input to the bus via the buffer.

One of the main challenges that had to be addressed with this system is loop control. Usually, a jump instruction takes a single clock cycle. This degrades processor efficiency because each micro-rotation requires such a jump. However, it will be noted that only one instruction needs to be looped and thus a specific counter has been incorporated. This allows an instruction to be executed in a predefined number of clock cycles, thus avoiding redundant cycles. To illustrate this we use the following example.

$$\mathrm{mov\ Reg\_XY, mem\_ac : mod\_vector, 14.}$$

This implies moving the values a and c in the dual-port memory into registers X and Y via data buses A and B, whilst setting the CORDIC module to vector mode with the loop number being 14 (in this case corresponding to a mantissa wordlength of 14). This number, which also corresponds to the number of rotation or vector mode iterations, is programmable allowing mantissa wordlengths to be varied. This feature also provides the facility

for trading off speed and numerical accuracy. In some applications a short mantissa may be acceptable. For example, the QRD-RLS algorithm remains stable for mantissa wordlengths as low as 5 bits [20]. This is due to its high numerical stability.

As discussed above (Fig. 5), the CORDIC module incorporates three levels of pipelining to provide a high clock speed. However, this makes assembly coding challenging because of uncertainty in determining the cycle on which the output value of the CORDIC unit becomes available. For example, this takes $2+2N$ cycles if running in vector or rotation mode but 4 cycles for other operations. To deal with this the compiler firstly parses the assembly code and determines which cycle the CORDIC output is available on. A dedicated register is then set with this value.

It will be noted that data scheduling and arithmetic operations can be separated. Due to the loop path in the CORDIC unit, the data buses are not occupied for part of the time and so data exchange can occur during these periods. Other operations performed by the CORDIC unit include, for example

$$\text{mov Reg\_XY}, \text{mem\_ac} : \text{mod\_xSuby\_xAddy}.$$

This statement implies moving the values a and c in the dual-port memory into registers X and Y via the data buses A and B, with the calculation of $x-y$ and $x+y$ performed in parallel in the X and Y blocks of the CORDIC unit, respectively. Therefore by careful design and definition of the instructions sets it is possible to perform most operations in parallel.

V. ASIP-Based SVD/QRD

The relevance of the QRD and SVD algorithms as separate entities has been discussed previously. However, many modern signal processing algorithms use these in conjunction with one another, a good example being SVD updating. In many real-time applications, it is necessary to continuously update the SVD as new input data is continuously added to the system. The SVD updating algorithm described in [21], [22] achieves this by combining QR updating with a Jacobi-type SVD algorithm. Such a combined SVD/QRD system can be implemented using a square processor array (Fig. 7) with each PE implemented as a single ASIP. This type of application illustrates the flexibility of the ASIP architecture described.

A. SVD Computation

In terms of implementation, each 2×2 matrix is mapped onto an ASIP PE. Each sweep of an SVD computation then requires the following schedule of operations.

1) Solve $c \pm b$ and $d \pm a$.
2) Solve $\phi \pm \theta$ using the CORDIC module and then solve θ and ϕ (diagonal cells only)
3) Solve the left- and right-rotation using the CORDIC module.
4) Exchange data with neighboring processors.
5) If the pre-defined sweeps have been finished, perform the final SFC (by shifts and adds), else go to step 1.

The values $c+b$ and $c-b$ can be calculated simultaneously in the X- and Y-blocks. The derived angles, ϕ and θ, are stored in the register-bank and the rotated data values are stored in their original location within the register-bank. After each two-sided rotation, data is exchanged with neighboring PEs using "Reg & buf," as shown in Fig. 6. The required register-bank memory contains only eight cells: four cells to store the block elements a, b, c, d, and four cells to store the temporal variable and angle values. A 16-cell register-bank is sufficient if both left- and right- singular vectors are required (i.e., an additional eight cells needed).

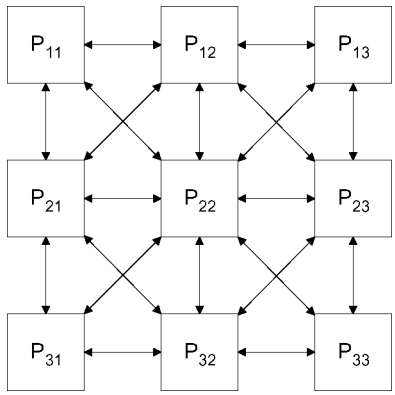


Fig. 7. Square array of ASIPs.

```
mov Reg_XY, mem_da : mod_xSuby_xAddy   // x = d − a and y = d + a
mov Reg_XY, mem_cb : mod_xSuby_xAddy   // x = c − b and y = c + b
mov mem_45, out_XY                     // mem4 = d − a and mem5 = d + a
mov mem_67, out_XY                     // mem6 = c − b and mem7 = c + b

mov Reg_XY, mem_47 : mod_vector, 14    //solve φ+θ
mov Reg_XY, mem_56 : mod_vector, 14    //solve φ−θ
mov RegX, outZ1
mov RegY, outZ2  : mod_xSuby_xAddy : half  //(φ ± θ) / 2
mov Reg_Z12_mem4, out_XY               // z1 = z2 = θ and mem4 = φ

mov Reg_XY, mem_ac : mod_rotation, 14  //left rotation for (a, c)
mov Reg_XY, mem_bd : mod_rotation, 14  //left rotation for (b, d)
mov mem_ac, out_XY
mov mem_bd, out_XY
```

Fig. 8. Assembly code of the operation of a diagonal PE.

Here, the angle solving (step 2) requires $2NT$, and the left- and right-rotation (step 3) requires $4NT$, the data preparation and data exchange (steps 1 and 4, respectively) require approximately $20T$, so each two-sided rotation requires a computation time of $(6N+20)T$. A section of the assembly code describing the operation of a diagonal PE is shown in Fig. 8.

In some applications, the implementation of full systolic arrays offers data rates far in excess of those required [8], [9]. In these situations, a hardware reduction can be achieved by mapping this to an array of reduced dimensions. In such a case, where it is desirable to use fewer PEs, i.e., number of PEs less than $M^2/4$ (for $M \times M$ matrix), multiple 2×2 submatrices can be mapped to a single PE, provided enough cells of the register-bank exist. Fig. 9 gives an example where a 12×12 matrix is mapped to a 3×3 PE array, where each PE contains four 2×2 submatrices. It will be noted that in the first two steps each diagonal PE has to calculate two sets of (θ, ϕ), while the off-diagonal PEs are idle. Therefore, the computation of one of the sets of (θ, ϕ) in each PE can be offloaded to a neighbor PE. This is shown in Fig. 10. The time for four two-sided rotations in each PE is approximately $2NT+4\times(4NT+20T) = (18N+80)T$.

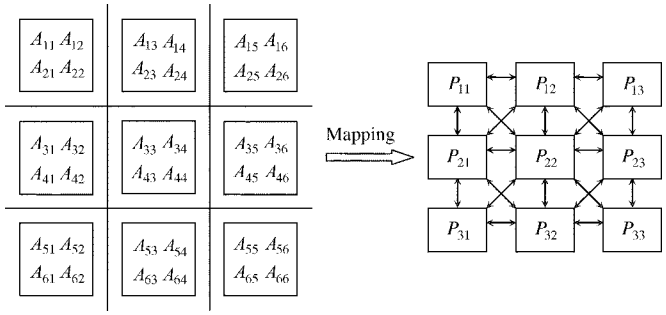
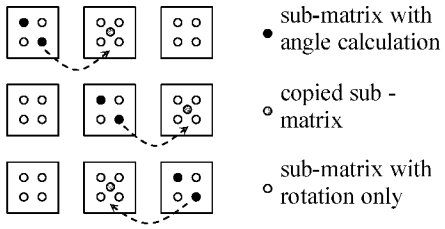


Fig. 9. Mapping of a 12 × 12 matrix to a 3 × 3 PE array (Each A_{ij} is a 2 × 2 submatrix).



1. Data preparation: copy A_{22} to P_{12}, A_{44} to P_{23}, A_{66} to P_{32}
2. Calculating angles: (θ_1,ϕ_1) in P_{11}, (θ_2,ϕ_2) in P_{12}, (θ_3,ϕ_3) in P_{22}, (θ_4,ϕ_4) in P_{23}, (θ_5,ϕ_5) in P_{33}, (θ_6,ϕ_6) in P_{32}
3. Applying four two-sided rotations in each PE

Fig. 10. SVD calculation for the case of mapping four 2 × 2 submatrices to one PE.

B. QRD Computation

A QRD computation then requires the following schedule of operations.

1) $r_{ij}(t)$ is multiplied by the forgetting factor β.
2) Solve the rotation angle $\alpha_i(t)$ which eliminates $x_i(t)$ and calculate $\gamma_{i+1}(t) = \gamma_i(t)\cos\alpha_i(t)$ (diagonal cell only).
3) Apply a left-rotation to $[r_{ij}(t)\ x_j(t)]^T$.

Usually, the forgetting factor β is close to unity and can be written as $\beta = 1 - 2^{-\varepsilon}$, where ε is an integer. Thus, the multiplication by β can be replaced by shift and add/subtract operations. Therefore, the computation time is dominated by the angle solving and rotation operations. In the case of QRD, the typical triangular systolic architecture can then be mapped to the rectangular processor array, as shown in Fig. 11.

From this, it will be observed that the computational load on the diagonal PEs is high compared with the internal cells. For example, P_{11} has to compute the rotation angle $\alpha_1(t)$ and $\alpha_2(t)$ as well as apply rotations to $[r_{11}(t)\ x_1^{(0)}(t)]^T$, $[r_{12}(t)\ x_2^{(0)}(t)]^T$ and $[r_{22}(t)\ x_2^{(1)}(t)]^T$. In addition, P_{11} has to derive the value $\gamma_{\text{out}} = \gamma_{\text{in}}\cos\alpha_1\cos\alpha_2$. Therefore, in Fig. 11, each internal cell performs four rotations and each diagonal cell performs two angle solving operations and five rotations i.e., the equivalent of seven rotations.

One approach that can be used to reduce overall computation time is to offload some of these computations to otherwise idle PEs below the diagonal as illustrated in Fig. 12.

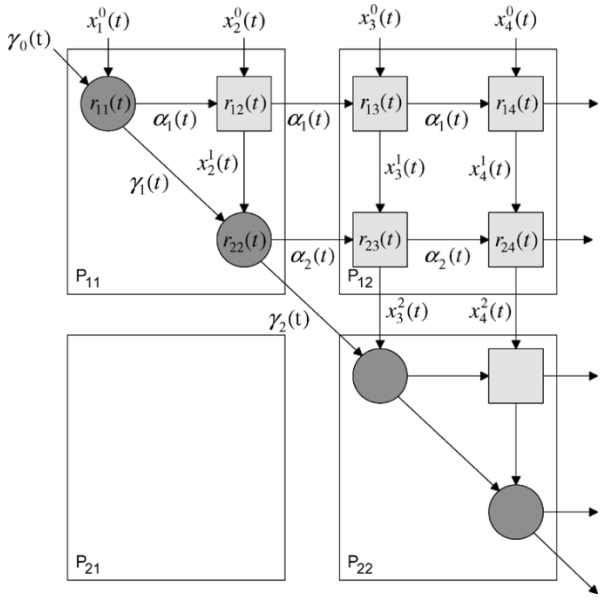


Fig. 11. QR array mapped to processor array.

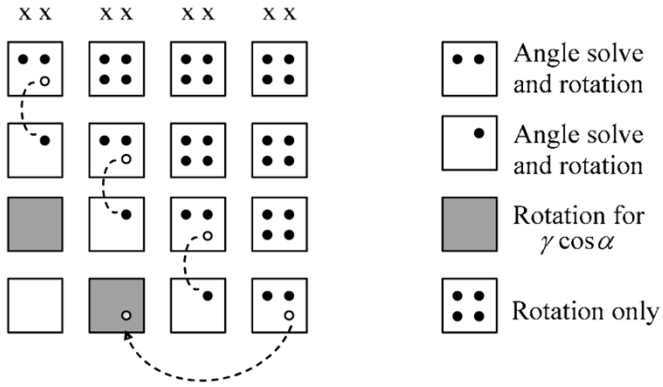


Fig. 12. ASIP array mapping for QRD.

(a) P_{11} operations
1. $r'_{11} \Leftarrow \beta r_{11}$ and $r'_{12} \Leftarrow \beta r_{12}$
2. solve angle α_1
3. rotation for $[r'_{11}\ \ x_1]^T$ and $[r'_{12}\ \ x_2]^T$ in parallel

(b) P_{12} operations
1. $r'_{13} \Leftarrow \beta r_{13}$ and $r'_{14} \Leftarrow \beta r_{14}$
2. rotation for $[r'_{13}\ \ x_3]^T$ and $[r'_{14}\ \ x_4]^T$ in parallel
3. $r'_{23} \Leftarrow \beta r_{23}$ $\quad r'_{24} \Leftarrow \beta r_{24}$
4. rotation for $[r'_{23}\ \ x'_3]^T$ and $[r'_{24}\ \ x'_4]^T$ in parallel

(c) P_{21} operations
1. $r'_{22} \Leftarrow \beta r_{22}$
2. solve angle α_2
3. rotation for $[r'_{22}\ \ x'_2]^T$ and $\gamma_1 = \gamma_0\cos\alpha_1$ in parallel

(d) P_{31} operations
1. rotation for $\gamma_2 = \gamma_1\cos\alpha_2$
2. rotation for $\gamma_4 = \gamma_3\cos\alpha_4$

Fig. 13. Operations of P_{11}, P_{12}, P_{21}, and P_{31}.

Fig. 13 provides a summary of the operations carried out in PEs P_{11}, P_{12}, P_{21}, and P_{31}. It will be noted that P_{21} is used to solve the angle $\alpha_2(t)$ and carry out the vector rotation $[r_{22}(t)\ \ x_2^{(1)}(t)]^T$ in parallel with the calculation of $\gamma_1(t) = \gamma_0(t)\cos\alpha_1(t)$. Effectively, then the operations of the lower diagonal cell in PE, P_{11} are computed in the neighboring PE, P_{21}. Similarly, P_{31} is used to perform the calculations $\gamma_2(t) =$

	0	T'	$2T'$	$3T'$	$4T'$	$5T'$	$6T'$	$7T'$	$8T'$	$10T'$
P_{11}	COMP $\alpha_1(0)$	ROT $r_{11}(0)$ ROT $r_{12}(0)$	COMP $\alpha_1(1)$	ROT $r_{11}(1)$ ROT $r_{12}(1)$	COMP $\alpha_1(2)$	ROT $r_{11}(2)$ ROT $r_{12}(2)$	COMP $\alpha_1(3)$	ROT $r_{11}(3)$ ROT $r_{12}(3)$	COMP $\alpha_1(4)$	
P_{12}			ROT $r_{13}(0)$ ROT $r_{14}(0)$	ROT $r_{23}(0)$ ROT $r_{24}(0)$	ROT $r_{13}(1)$ ROT $r_{14}(1)$	ROT $r_{23}(1)$ ROT $r_{24}(1)$	ROT $r_{13}(2)$ ROT $r_{14}(2)$	ROT $r_{23}(2)$ ROT $r_{24}(2)$	ROT $r_{13}(3)$ ROT $r_{14}(3)$	
P_{21}			COMP $\alpha_2(0)$	ROT $r_{22}(0)$ ROT $\gamma_0(0)$	COMP $\alpha_2(1)$	ROT $r_{22}(1)$ ROT $\gamma_0(1)$	COMP $\alpha_2(2)$	ROT $r_{22}(2)$ ROT $\gamma_0(2)$	COMP $\alpha_2(3)$	
P_{31}					ROT $\gamma_1(0)$		ROT $\gamma_1(1)$		ROT $\gamma_1(2)$ ROT $\gamma_3(0)$	

(COMP = computation and ROT = rotation)

Fig. 14. Schedule of PEs P_{11}, P_{12}, P_{21}, and P_{31}.

```
mov RegX, mem_a : mod_xSubShift, 7     // r − 2^−7 r ≈ 0.992r
mov RegX, mem_b : mod_xSubShift, 7
mov mem_a, OutX                        // mem_a ⇐ βr1
mov mem_b, OutX                        // mem_b ⇐ βr2
mov mem_4, RegU                        // input x from upper register
mov Reg_XY, mem_a4 : mod_vector, 14    //solve α1,
mov Reg_Z12, OutZ                      //set angle as α1
mov Reg_RD, OutZ                       //pass angle to right and down
mov Reg_XY, mem_a4 : mod_rotation, 14  //rotation for r1
mov Reg_XY, mem_b5 : mod_rotation, 14  //rotation for r2
mov mem_a, OutX                        //save r1' in mem_a
mov mem_b_Reg_D, Out_XY                //save r2' in mem_b and pass x2' to the lower PE
```

Fig. 15. Section of assembly code for P_{11}.

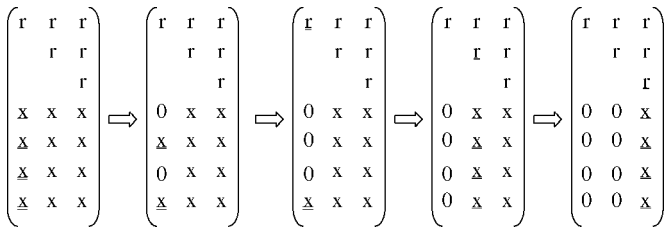


Fig. 16. Block-processing QR update with the annihilation reordering (look-ahead factor is 4).

$\gamma_1(t)\cos\alpha_2(t)$ and $\gamma_4(t) = \gamma_3(t)\cos\alpha_4(t)$. Therefore, P_{31} is used to carry out computations that would have been carried out in PEs P_{11} and P_{22}.

If the time for $\beta r(t)$ and data exchange is small compared with the time for angle solving and rotation, then it follows that each PE takes $4NT$ for angle solving and/or rotation operations. This is because a pipelined CORDIC unit enables two rotation operations to be carried out in parallel.

Let $2T' = 4NT + mT$, where mT is the time for βr and data exchanging operations. Then the schedule of operations for the four cells is shown in Fig. 14, where only angle solving and rotation operations are given for simplicity.

For other diagonal PEs, computations can be assigned and distributed in a similar manner. Therefore, this mapping means that each PE takes almost the same number of cycles for processing, and most PEs are fully utilized, in effect balancing the computational load across the hardware array.

The assembly code for P_{11} is shown in Fig. 15, and has duration of approximately $4NT + 10T$, and so the data input rate can be up to $1/(4N + 10)T$.

An alternative approach, which allows a higher data throughput rate to be achieved, is presented below. In this an ASIP based array can be configured as an "annihilation-reordering look-ahead" RLS filter [23]. The block data processing involved is illustrated schematically in Fig. 16 for a system in which the look-ahead factor is four.

Fig. 16 shows that each block of four new inputs (e.g., $\{x_{n+1}, x_{n+2}, x_{n+3}, x_{n+4}\}$) requires four angle solutions and subsequent rotations. Since the pipelined CORDIC module allows two angle solutions or rotations to be computed in parallel, each PE can finish four such operations within $2T'$ provided data is scheduled properly.

Fig. 16 also shows that although the first two rotations can be applied in parallel, the remaining two rotations are dependent, i.e., they cannot be calculated in parallel. However, they can be calculated in parallel with the previous and subsequent input block. This can be shown as follows.

1) Apply rotations $[x_{n+1}, x_{n+2}]^T \Rightarrow [0, x'_{n+2}]^T$ and $[x_{n+3}, x_{n+4}]^T \Rightarrow [0, x'_{n+4}]^T$ in parallel.
2) Apply rotations $[x'_{n+2}, x'_{n+4}]^T \Rightarrow [0, x''_{n+4}]^T$ and $[x_n, r_{n-4}]^T \Rightarrow [0, r_n]^T$ in parallel.
3) Apply rotations $[x''_{n+4}, r_n]^T \Rightarrow [0, r_{n+4}]^T$ and $[x'_{n+6}, x'_{n+8}]^T \Rightarrow [0, x''_{n+8}]^T$ in parallel.

The procedure for angle solving follows that described above.

In this case, the function of each internal cell can be mapped to a single PE and the function of a boundary cell can be mapped to two separate PEs, one for angle solving and another for performing rotations. This is shown in Fig. 17. It will be noted that each PE can support four data samples in $2T'$ cycles, and thus the data throughput is $4/(4N + 10)T$ (assuming the same time for data preparation). This is almost four times the throughput of Fig. 12. In this case, the maximum number of sensors that the ASIP array can support is equal to the ASIP array size.

VI. RTL Implementation Results

Initial design studies have been carried out based on a TSMC 0.13 micron, 1.2 V CMOS technology. Synthesized RTL results indicate that for a wordlength of $14 + 6$ bits (mantissa

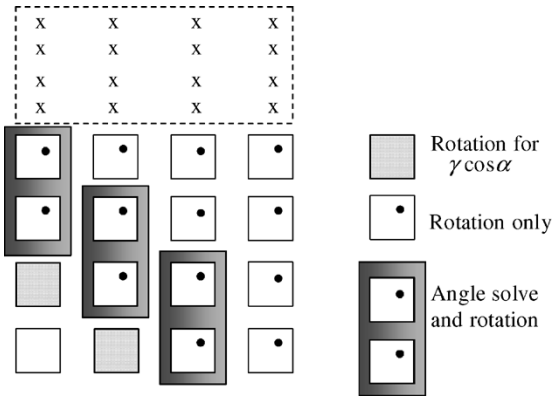


Fig. 17. ASIP array mapping for block-processing QR update.

TABLE I
AREA OF EACH BLOCK IN A PE

Each PE Wordlength (14+6)	CORDIC module	Control logic	Register-bank (32×20 bits)	Program RAM (128×24 bits)	Total
Area (gates)	8650	2820	3060	3000 (estimation)	17.5K

and exponent), a clock rate of up to 300 MHz is achievable, with each PE requiring around 17.5 K gates. Details of the area requirements for each part of the PE are listed in Table I. Therefore, a system comprising 20 × 20 CORDIC modules requires around 7 M gates and can therefore be accommodated on a modern System-on-Chip (SoC) device. If implemented as described above then this can perform SVD on a matrix of dimensions up to 40 × 40 with the overall SVD computations requiring less than 70 μs (assuming five sweeps), or less than 512 μs (assuming five sweeps) on a 80 × 80 matrix (employing the mapping technique described). A QRD array of different sizes (i.e., different number of input sensors) can also be constructed from such PEs. For QRD based adaptive beamforming, the results obtained indicate that a sampling rate up to $1/(4N+10)T \approx 4.5$ MHz for 40 sensors can be supported. Alternatively, if in "look ahead" mode, a 20-sensor array with a sampling rate of up to 18 MHz can be supported.

The authors have been made aware of the work of Glokler and Meyr [24] who have recently undertaken an ASIP case study for eigenvalue decomposition (EVD) as part of a broader investigation into ASIPs for embedded DSP applications with performance and energy constraints. This specific design, referred to as ICORE-II, has been developed to study the relative merits of using a fixed instruction set processor with parameterized units such as multiplier, adders and memories and an alternative which uses a dedicated CORDIC datapath for hardware acceleration. A detailed comparison with this is difficult as insufficient detail is presented, for example on the level of pipelining used and hence the number of clock cycles required to compute the algorithm. In addition, as the authors themselves point out, multiple ICORE-II processors are required for high computational requirements but this is not something they have addressed. This requires extra data storage making direct area comparisons difficult. Other differences such as variations in target technology, in wordlengths (which affects number of CORDIC iterations required and hence computation time) and matrix dimensions also hinder a straightforward comparison. What comparisons can be made are summarized in Table II.

TABLE II
PERFORMANCE COMPARISON OF ICORE-II [24] AND PROPOSED QRD/SVD ASIP

	EVD ASIP (ICORE-II) [24]	QRD/SVD ASIP
Wordlength	8-bit exponent and 24-bit mantissa	6-bit exponent and 14-bit mantissa
Technology	0.18 micron	0.13 micron
Critical Path	6.8ns	3.3ns
Area (gates)	34.5K	17.5K
PE number	Not specified	400
Total Area (gates)	Not specified	7M

VII. DISCUSSION AND SUMMARY

An ASIP suitable for real-time SVD, QRD, and combined SVD/QRD computation has been described. This takes the form of a programmable processor in which the arithmetic operations are performed using an internal CORDIC module. This processor employs a novel SFC method, which reduces overall computation time and incorporates pipelining with ensuing benefits in terms of higher performance for the same silicon area. It also incorporates mechanisms for parallel instruction issue and uses dual data buses, which act in a similar manner. It has been shown how, through careful scheduling and off-loading of computations onto neighboring processors, the SVD array can be mapped onto an ASIP array of reduced dimensions in an efficient manner. The use of similar techniques have also been described to balance the computational load across an ASIP array configured as an annihilation-reordering look-ahead RLS filter, hence supporting an increased throughput rate for QRD based adaptive beamforming.

The ASIP described can also be used to implement a range of other modern signal processing algorithms including ones involving Jacobi rotations. Applications include, for example, SVD updating for subspace tracking [21], [22]. In [25], an algebraic transformation approach is used to eliminate the critical loop path making the delay unrelated to the problem size. Here, SVD updating is achieved by combining QR updating with a single sweep of the SVD algorithm.

Many other similar examples also exist. For example, the concept of algorithmic engineering, proposed by McWhirter *et al.* [26], provides a framework for describing and manipulating the type of building blocks commonly used to define a wide range of parallel algorithms and associated architectures for many modern digital signal processing applications. It is the belief of the authors that the ASIP described can be used to provide physical (i.e., silicon) instantiations for algorithms captured and developed in this way i.e., as a building block, suitable for Jacobi rotation based matrix computation in modern signal processing. It can of course also be used for more general processors e.g., ones for performing matrix-by-matrix multiplication.

As discussed above, the results of design studies have shown that with such processors, it is now possible to implement a wide range of real-time systems SVD/QRD systems covering

different specifications on a single SoC device with these being programmable in terms of the wordlengths used. It is also possible to use these to implement systems covering array sizes up to $N \times N$ thus allowing a single chip system to be created, which is applicable across many applications and specification requirements.

APPENDIX

In order to compute the bounds of the scale factor (2.3c) is considered as follows:

$$k_n = \prod_{i=0}^{n-1} \cos \alpha_i = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1+2^{-2i}}}. \tag{A-1}$$

It can be seen that $k_1 = 1/\sqrt{2}$ and $k_{n+1}/k_n < 1$, so for any $n > 1$,

$$k_n < k_1 = 2^{-\frac{1}{2}}$$

On the other hand, if applying logarithms to (A-1) then

$$\ln(k_n) = -\frac{1}{2} \sum_{i=0}^{n-1} \ln(1+2^{-2i}). \tag{A-2}$$

It can be noted that $\ln(1+x)$ can be substituted with Maclaurin's series

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots \tag{A-3}$$

and it is clear that $\ln(1+x) < x \; for \; \forall |x| < 1$, and then (A-2) becomes

$$\begin{aligned} \lim_{n->\infty} \ln(k_n) &= -\frac{1}{2} \lim_{n->\infty} \sum_{i=0}^{n-1} \ln(1+2^{-2i}) \\ &> -\frac{1}{2} \lim_{N->\infty} \sum_{i=0}^{N-1} 2^{-2i} = -\frac{2}{3} \end{aligned} \tag{A-4}$$

i.e., $\lim_{n->\infty} k_n > e^{-2/3} = 0.5134 > 2^{-1}$.

Therefore, the bounds of the scale factor are $2^{-1/2} > k_n > 2^{-1}$.

REFERENCES

[1] W. M. Gentleman and H. T. Kung, "Matrix triangularization by systolic arrays," in *Proc. SPIE Real-Time Signal Processing IV*, vol. 298, 1981, pp. 19–26.

[2] J. G. McWhirter, "Recursive least squares minimization using a systolic array," in *Proc. SPIE Real-Time Signal Processing IV*, vol. 431, 1983, pp. 105–112.

[3] T. J. Shepherd and J. G. McWhirter, "Systolic adaptive beamforming," in *Array Signal Processing*, S. Haykin, J. Litva, and T. J. Shepherd, Eds. New York: Springer-Verlag, 1993, ch. 5, pp. 153–243.

[4] R. P. Brent and F. T. Luk, "The solution of singular-value problems using systolic arrays," in *Proc. SPIE Real-Time Signal Processing VII*, vol. 495, 1984, pp. 7–12.

[5] R. P. Brent, F. T. Luk, and C. Van Loan, "Computation of the singular-value decomposition using mesh-connected processors," *J. VLSI Comp. Syst.*, vol. 1, no. 3, pp. 242–270, 1985.

[6] J. R. Cavallaro and F. T. Luk, "CORDIC arithmetic for an SVD processor," *J. Parallel Distrib. Comput.*, vol. 5, no. 3, pp. 271–290, 1988.

[7] H. W. van Dijk, G. J. Hekstra, and E. F. Deprettere, "Scalable parallel processor array for Jacobi-type matrix computations," *Integration*, vol. 20, pp. 41–61, 1995.

[8] G. Lightbody, R. Walke, R. Woods, and J. McCanny, "Linear QR architecture for a single chip adaptive beamformer," *J. VLSI Signal Process. Syst.*, vol. 24, pp. 67–81, 2000.

[9] G. Lightbody, R. Woods, and R. Walke, "Design of a parameterizable silicon intellectual property core for QR-based RLS filtering," *IEEE Trans. VLSI Syst.*, vol. 11, no. 4, pp. 659–678, 2003.

[10] Z. Liu, J. McCanny, G. Lightbody, and R. Walke, "Generic SoC QR array processor for adaptive beamforming," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 50, no. 4, pp. 169–175, 2003.

[11] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. EC-9, no. 3, pp. 330–334, 1959.

[12] J. S. Walther, "A unified algorithm for elementary functions," in *Proc AFIPS Conf.*, vol. 38, 1971, pp. 379–385.

[13] K. Kota and J. R. Cavallaro, "Numerical accuracy and hardware tradeoffs for CORDIC arithmetic for special purpose processors," *IEEE Trans. Comput.*, vol. 42, no. 7, pp. 769–779, Jul. 1993.

[14] S.-F. Hsiao and J. M. Delosme, "Parallel singular-value decomposition of complex matrices using multidimensional CORDIC algorithms," *IEEE Trans. Signal Process.*, vol. 44, no. 3, pp. 685–697, Mar. 1996.

[15] A. A. J. de Lange, A. van der Hoeven, E. F. Deprettere, and J. Bu, "An optimal floating-point pipeline CMOS CORDIC processor," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 3, 1988, pp. 2043–2047.

[16] J. R. Cavallaro and F. T. Luk, "Floating-point CORDIC for matrix computations," in *Proc. IEEE Int. Conf. Computer Design*, 1988, pp. 40–42.

[17] G. J. Hekstra and E. F. Deprettere, "Floating point CORDIC," in *Proc. 11th IEEE Symp. Computer Arithmetic*, 1993, pp. 130–137.

[18] R. L. Walke, R. W. M. Smith, and G. Lightbody, "Architectures for adaptive weight calculation on ASIC and FPGA," in *Proc. IEEE 33rd Asilomar Conf. Signals, Systems and Computers*, vol. 2, 1999, pp. 1375–1380.

[19] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed. Baltimore, MD: Johns Hopkins Univ. Press, 1989, pp. 444–459.

[20] B. Yang and J. F. Bohme, "Rotation-based RLS algorithms: unified derivations, numerical properties, and parallel implementations," *IEEE Trans. Signal Processing*, vol. 40, no. 5, pp. 1151–1167, May 1992.

[21] M. Moonen, P. VanDooren, and J. Vanderwalle, "A SVD updating algorithm for subspace tracking," *SIAM J. Matrix Anal. Appl.*, vol. 13, pp. 1015–1038, 1992.

[22] M. Moonen, E. Deprettere, I. Proudler, and J. McWhirter, "On the derivation of parallel filter structures for adaptive eigenvalue and singular-value decompositions," in *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, 1995, pp. 3247–3250.

[23] L. Gao and K. Parhi, "Hierarchical pipelining and folding of QRD-RLS adaptive filters and its application to digital beamforming," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 47, no. 12, pp. 1503–1519, Dec. 2000.

[24] T. Glokler and H. Meyr, *Design of Energy-Efficient Application-Specific Instruction Set Processors*. Norwell, MA: Kluwer, 2004.

[25] J. Ma, K. K. Parhi, and E. F. Deprettere, "A unified algebraic transformation approach for parallel recursive and adaptive filtering and SVD algorithms," *IEEE Trans. Signal Processing*, vol. 49, no. 2, pp. 434–437, Feb. 2001.

[26] J. McWhirter and I. Proudler, "Algorithmic engineering: a worked example," *Signal Process. VI*, pp. 5–12, 1992.

Zhaohui Liu received the B.Eng., M.Eng., and Ph.D. degrees in electrical engineering from Beijing Institute of Technology, Beijing, China, in 1991, 1994, and 1999, respectively.

From 1994 to 1996, he was a Software Engineer. Currently he is a Research Fellow with the Institute of Electronics, Communications and Information Technology (ECIT), Queen's University, Belfast, Northern Ireland, U.K. His research interests include the design of very large-scale integration architectures and circuits for digital signal processing, with emphasis on adaptive beamforming and subspace based signal analysis.

Kevin Dickson was born in Ballymoney, County Antrim, Northern Ireland, U.K., in 1980. He received the M.Eng. degree in electrical and electronic engineering from Queen's University, Belfast, Northern Ireland, U.K., in 2002. He is currently working toward the Ph.D. degree at Queen's University, Belfast, Northern Ireland, U.K.

His main research interests are in very large-scale integration architectures and digital signal processing.

John V. McCanny (M'86–SM'95–F'99) received the Bachelor's degree in physics from the University of Manchester, Manchester, U.K., the Ph.D.degree in physics from the University of Ulster, Ulster, U.K., and the D.Sc. (higher doctorate) degree in electrical and electronics engineering from Queen's University, Belfast, Northern Ireland, U.K., in 1973, 1978, and 1998, respectively.

He is an international authority in the design of silicon integrated circuits for Digital Signal Processing; having made many pioneering contributions to this field. He has co-founded two successful high technology companies, Audio Processing Technology Ltd., and Amphion Semiconductor Ltd. a leading supplier of SoC cores for video compression. He is currently Director of the Institute of Electronics, Communications and Information Technology at Queen's University, Belfast. He has published 250 major journal and conference papers, holds 25 patents and has published five research books.

Prof. McCanny is a Fellow of the Royal Society, the Royal Academy of Engineering, the Institution of Electrical Engineers ,and the Institute of Physics. He has won numerous awards, including a Royal Academy of Engineering Silver Medal for outstanding contributions to U.K. engineering leading to commercial exploitation (1996), an IEEE Third Millennium medal and the Royal Dublin Society/Irish Times Boyle Medal (2003). In 2002, he was awarded a CBE for his contributions to engineering and higher education.