A COMPLEX ARITHMETIC DIGITAL SIGNAL PROCESSOR USING CORDIC ROTATORS

S. Freeman and M. O'Donnell
Electrical Engineering and Computer Science Department
and Bioengineering Program
University of Michigan
Ann Arbor, MI 48109

ABSTRACT

A versatile signal processor has been designed that can perform multiple rotations, multiplications and additions within one clock cycle. The computational elements of this processor include four pipelined CORDIC rotators, two pipelined fast multipliers and two adders. A combination of register files, SRAM, and ROM provides on chip storage for coefficients, running sums and programs. The chip architecture and its applicability to complex valued signal processing tasks are discussed.

1. INTRODUCTION

Digital signal processors are typically designed for fixed or floating point real number calculations. Consequently, complex number operations often suffer a 4:1 throughput penalty due to the 4 multiplies and 4 adds needed for a single, full complex multiply accumulate. Many applications, including real-time baseband filtering in medical ultrasound color flow Doppler processors, require complex arithmetic. Hence, alternate computational structures are needed to maximize throughput in these systems. This paper explores an alternate computational architecture based on a geometric interpretation of complex multiplication.

Complex multiplication can be viewed either as 4 real number operations or as a scaling and phase rotation of one complex number by the magnitude and phase of another complex number. That is, complex multiplication can be viewed as a phase rotation followed by scaling of the resultant vector. Consequently, the efficiency of complex multiplication may be increased by integrating vector rotation hardware into a typical digital signal processor. Moreover, phase sensitive processing, such as complex correlation, may be easily implemented with efficient vector rotation structures.

There are several possible methods for efficient vector rotation. Because of its simplicity, the most commonly used method is based on the CORDIC (COordinate Rotation Digital Computer) algorithm. This algorithm can perform vector rotations along a line, circle, or hyperbola [1,2,3,4,5,6,7,8]. Several different monolithic implementations have been realized using both fixed point and floating point computations [1,3,4,6,9]. These chips were developed either as application specific rotator units or as general-purpose stand-alone processors. This paper presents a monolithic chip that includes several CORDIC rotators along with the basic computational elements found in commercial digital signal processors to create a powerful yet versatile general-purpose digital signal processor.

Section 2 of this paper briefly describes the CORDIC algorithm and its implementation. The numerous modes in which the four CORDIC rotators can be used are also discussed. The following section presents other circuit elements on the chip, including computational and storage units, as well and the ways in which these elements are interconnected. In section 4, examples are presented illustrating the processor's ability to perform complex valued arithmetic efficiently. Also discussed here are possible pipelining and parallelization schemes maximizing throughput. Section 5 discusses other modes of operation, including self-test, external test, and single execution modes.

2. THE CORDIC ALGORITHM

The CORDIC algorithm was proposed by Volder[7] in 1959 as an iterative method to transform rectangular to polar coordinates by successive circular rotations of a vector through a fixed set of angles. Others have extended the algorithm to permit rotations of a vector along a line or hyperbola [1,2,8]. We have implemented only a circular rotator, since FFTs and most complex number operations use only this type of rotation. A circular rotation can be written as:

$$I_{out}=I_{in}\cos(\Theta)-Q_{in}\sin(\Theta) \qquad (1)$$
$$Q_{out}=Q_{in}\cos(\Theta)+I_{in}\sin(\Theta),$$

0-7803-2431-5/95 $4.00

where Θ is the angle of rotation, I represents the real part and Q the imaginary part of a complex signal. Any arbitrary rotation can be decomposed into a set of smaller rotations performed iteratively according to the relations:

$$I_{n+1}=I_n\cos(\Theta_n)-Q_n\sin(\Theta_n) \qquad (2)$$
$$Q_{n+1}=Q_n\cos(\Theta_n)+I_n\sin(\Theta_n),$$

where n is the iteration index. Volder noticed that by choosing $\tan(\Theta_n)=1/2^n$ (therefore Θ_n=45,26.5,14,7.12°...) the rotation operation can be transformed into a series of simple arithmetic operations, written specifically as:

$$I_{n+1}=k_n(I_n-\sigma_n Q_n 2^{-n}) \qquad (3)$$
$$Q_{n+1}=k_n(Q_n+\sigma_n I_n 2^{-n}).$$

The k_n scaling factor is constant for a fixed number of rotations (iterations) and so can be omitted within each iteration. These equations are easily implemented with binary shift and add/subtract circuits. One approach uses pipelined adders with fixed shifts between them. Another, (the one we adopted), uses a recursive cell with barrel shift circuits and adders to iteratively perform the rotations.

The σ_n in the above equations represents the direction of each rotation; σ_n=1 is a positive rotation and σ_n=-1 is a negative rotation. By varying the direction during each iteration, a vector is rotated through any arbitrary angle subject to the quantization of the algorithm[10]. Practically, σ_n is actually a sign bit, s_n, which controls the adder/ subtractor circuits such that s_n=1 is a positive rotation and s_n=0 is a negative rotation.

The CORDIC rotator shown in Figure 1 can be operated in either angle accumulation or rotation modes. For angle accumulation, the sign bit (s_n) equals the MSB of the imaginary component. If the Q component is positive (MSB=0 in 2's complement notation), then a negative rotation will occur. The effect is to iteratively force the vector toward the real (I) axis. In rotation mode, the sign bits are supplied from outside the rotator and perform some arbitrary rotation through a given angle Θ.

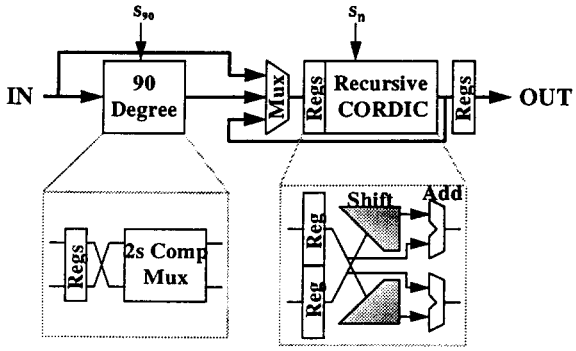

Figure 1. Block diagram of CORDIC rotator

The specific complex number processing unit we have developed is based on a four CORDIC rotator structure illustrated in Figure 2. This structure consists of one master, two slaves, and a master/slave rotator. A master cell can export its internal sign bit to the slave rotators. This allows the master rotation to be mirrored in each of the slave CORDICs.

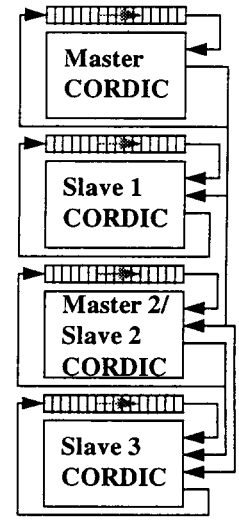

Figure 2. Sign Bit Distribution

Figure 2 shows the sign bit distribution scheme among the CORDIC rotators. This scheme allows the processor to perform a full complex multiply by decoupling phasor rotation from magnitude scaling:

$$(|A|e^{j\Theta_A})(|B|e^{j\Theta_B}) = (|A|e^{j0})(|B|e^{j(\Theta_A+\Theta_B)}) \qquad (4)$$
$$= |A||B|\,e^{j(\Theta_A+\Theta_B)}$$

Here, the master CORDIC rotates its vector toward the real axis, while the slave mirrors this rotation, thereby performing the phase shift involved in the complex multiply. The magnitude multiply is simplified since the master vector magnitude (i.e. |A|) is available at the I output and can be used to multiply the I and Q components of the slave. A full complex multiply using this method requires one rotation and two multiplies, whereas a normal complex multiply requires four multiplies and two additions.

Each CORDIC rotator contains a shift register serving as local storage for the sign bits. Therefore, every CORDIC can either be operated independently to rotate through different angles, or coupled with other CORDICs in many ways for different tasks. Complex conjugate operations are performed simply by inverting the desired sign bits.

3. OTHER CIRCUIT ELEMENTS

The four CORDIC cells occupy roughly 1/3 of the available chip area (section 6); the remainder is other computational and storage elements which vastly extend the chip's suitability to DSP applications. Two high-speed, two-stage multipliers can perform the magnitude multiply operations following vector rotations, and two adders can perform running sum calculations. Running sums and filter coefficients are stored in two separate register files. One of these has two read ports for updating coefficients and running sums simultaneously. A 824 byte SRAM

holds 64 microcoded program instructions that can be executed continuously using simple looping procedures. Finally, an FFT ROM contains the angle information for computing up to 32 point FFTs as well as self-test information for that mode (section 5).

Communication between the various computational and storage elements was designed to be as flexible as possible permitting programmable optimization for each application. FFTs, for instance, use the rotators and adders, whereas complex filtering requires rotators, multipliers, and adders, and coordinate transformations only require rotators. On-chip bus structures, shown in Figure 3, provide communication between the various circuit elements.

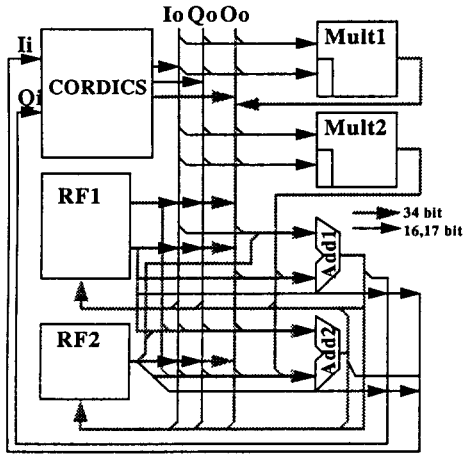

Figure 3. On-chip bus structures; combinations of multiplexers and tristate buffers control bus interconnections.

4. PROGRAMMING AND EXAMPLES

The chip uses a 103 bit microcode instruction during each clock cycle to explicitly control all bus interfaces and computational elements within the chip. This allows the programmer to use all buses and major elements of the chip to optimize performance. These instructions also control the 34 input and 34 bi-directional pins on the chip, as well as regulate their connection to the internal buses and computational elements.

For efficient programming, the pipelining and parallelization features of the various circuit elements must be exploited. In particular, the following functions can be performed simultaneously if care is taken to control the buses properly:

1) Load CORDICs with new data.
2) Rotate four vectors through same or different Θ.
3) Multiply rotated vectors by magnitude of filter tap.
4) Add scaled vectors to running sum and output result.

Only these four operations are needed to implement a complex valued digital filter. Consequently, this chip has a maximum throughput of one complex filter tap per cycle. FFTs are calculated using functions 1,2, and 4 to obtain the same throughput. For longer filters or FFTs, the throughput drops simply because the running sums have more terms in them.

5. OTHER MODES

In addition to normal operation, a finite state machine (FSM) can be executed that loads a program into the on-chip SRAM and initializes both register files. Another FSM can reload just the register files, for instance, to change filter coefficients on the fly. There is also a test-mode scan chain that can shift known data through all on-chip registers. This chain is used as part of a self-test FSM that shifts FFT coefficients through the registers. After all registers have been scanned, the processor runs for one cycle, and then shifts the data out of the registers to compare with known results stored in ROM. If the self test passes, an "OK" signal is asserted. For applications that do not completely fit in the on chip SRAM, a "Single Execution Mode" has been developed that allows a single instruction to be loaded into the instruction register (two cycles) and then executed (one cycle). The user can also stall the chip, where all operations are suspended indefinitely. This may be useful for processing signals with non-uniform data rates.

6. CHIP SPECIFICATIONS

The chip was designed and simulated using a Verilog hardware description language where layout was later generated by the Epoch silicon compiler from Cascade Design Automation for a .8μm three-metal CMOS process. The 84 pin chip, whose core is shown in Figure 4, contains more than 240,000 transistors on a 8.2 x 8.1mm die and is expected to operate at clock frequencies up to 33MHz.

7. CONCLUSION

A digital integrated circuit has been designed integrating multiple CORDIC rotators, multipliers, adders, and storage units into a general-purpose digital signal processor. This device is well suited to any digital signal processing application using complex valued arithmetic, including complex correlation and filtering and FFT computation.

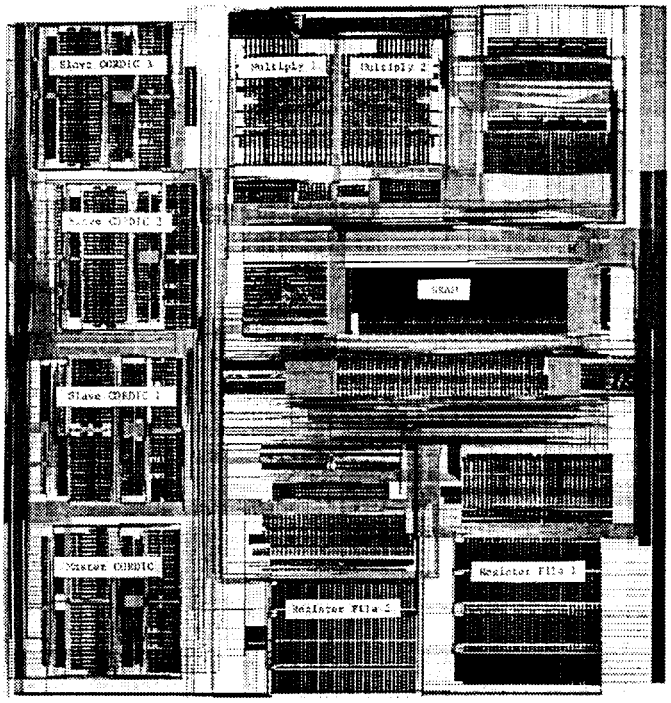

Figure 4. Complex number DSP chip core

References:

[1] A.A.J. de Lange, E.F. Deprettere, A. van der Veen, J. Bu, "Real Time Applications of the Floating Point Pipeline CORDIC Processor in Massive-Parallel Pipelined DSP Algorithms", Proc. of ICASSP, pp. 1013-16, April 1990.

[2] A.M. Despain, "Fourier Transform Computers Using CORDIC Iterations", IEEE Trans. on Computers, Vol. C-23, No. 10, pp. 993-1001, Oct, 1974.

[3] M.D. Ercegovac, T. Lang, "Implementation of Fast Angle Calculation and Rotation Using On-Line CORDIC", Int'l Symp. on Circuits and Systems, pp. 2703-6, June 1988.

[4] G.L. Haviland, A.A. Tuszynski, "A CORDIC Arithmetic Processor Chip", IEEE Trans. on Computers, Vol. C-29, No. 2, pp. 68-79, February 1980.

[5] Y.H. Hu, "CORDIC-Based VLSI Architectures for Digital Signal Processing", IEEE Signal Processing Magazine, pp. 16-35, July 1992.

[6] D. Timmerman, et.al, "A Programmable CORDIC Chip for Digital Signal Processing Applications", IEEE J. of Solid-State Circuits, Vol. 26, No. 9, pp. 1317-21, Sept. 1991.

[7] J.E. Volder, "The CORDIC Trigonometric Computing Technique", IRE Transactions on Electronic Computers, Vol. EC-8, No. 3, pp. 330-4, Sept. 1959.

[8] J.S. Walther, "A Unified Algorithm for Elementary Functions", Spring Joint Computer Conference, pp. 379-385,1971.

[9] M. O'Donnell, W.E. Engeler, "Correlation-Based Aberration Correction in the Presence of Inoperable Elements", IEEE Trans. on Ultrasonics, Ferroelectrics, and Frequency Control, Vol. UFFC-39, pp. 700-707, 1992.

[10] Y.H. Hu, "The Quantization Effects of the CORDIC Algorithm", IEEE Trans. on Signal Processing, Vol. 40, No. 4, pp. 834-44, April 1992.