CORDIC-Based LMMSE Equalizer for Software Defined Radio

Murugappan SENTHILVELAN †‡, Javier HORMIGO ‡, Joon Hwa CHUN €, Mihai SIMA *, Daniel IANCU ¶, Michael SCHULTE §, John GLOSSNER ¥

†Optimum Semiconductor Technologies, Inc., Tarrytown, NY, U.S.A.
‡University of Malaga, Department of Computer Architecture, Malaga, Spain.
* University of Victoria, Department of Electrical and Computer Engineering, Victoria, BC, Canada.
§University of Wisconsin-Madison, Department of Electrical and Computer Engineering, Madison, WI, U.S.A.
¶Tampere University of Technology, Korkeakoulunkatu 1, FIN-33720 Tampere, Finland.
¥Sandbridge Technologies, Inc., Tarrytown, NY, U.S.A.
e-mail: msenthilvelan@optimumsemi.com, fjhormigo@uma.es, jchun@i-a-i.com
msima@ece.uvic.ca, schulte@engr.wisc.edu, diancu@optimumsemi.com, jglossner@sandbridgetech.com

Abstract — In Code Division Multiple Access (CDMA) systems, the orthogonality of the spreading codes used to achieve multiple access over a channel is severely degraded due to multi-path interference. Expensive equalization techniques are needed to recover the transmitted signal. The Linear Minimum Mean Square Error (LMMSE) equalizer is a sub-optimal equalizer that is a good compromise between computational complexity and communication system performance. It uses computationally-intensive matrix inversion operations to perform equalization. In this paper, we address the computational challenges of implementing the LMMSE equalizer on Software Defined Radio (SDR) platforms. SDR platforms are favored by the wireless industry due to their significant benefits of reduced development costs and accelerated time-to-market. We present COrordinate Rotation Digital Computer (CORDIC) Instruction Set Architecture (ISA) extensions that speed up the LMMSE equalization algorithm. The costs and benefits of the ISA extensions are evaluated on the Sandbridge Sandblaster 3000 (SB3000) low-power, multithreaded SDR processor. The proposed ISA extensions provide significant performance improvements with little hardware overhead, while improving the accuracy of the LMMSE Equalizer.

Keywords – LMMSE equalization, CORDIC, SDR, ISA extensions, QR decomposition, Givens rotation

I. INTRODUCTION

Code Division Multiple Access (CDMA) systems [1] exploit orthogonality between vectors to support multiple users over a single channel. Each user is assigned one orthogonal vector, called the spreading code, which is a chip sequence \( c \) from the Walsh matrix, for spreading the data to be transmitted over the channel. The transmitter sends either the chip sequence \( c \) or \(-c\) depending on whether the user wants to communicate a 1 or 0 symbol over the channel. Under ideal channel conditions, in the absence of multi-path and channel interference, and with synchronized transmission of blocks over the channel, multiple users can be multiplexed over the same channel by using a different spreading code for each user. Even though signals from multiple users interfere with each other in the channel, due to the orthogonal spreading codes, each user can extract the transmission intended for them from the received signal by despreading it using their spreading code.

While the spreading codes are perfectly orthogonal to each other, they are not orthogonal to time-shifted versions of each other, resulting in Multiple Access Interference (MAI), or time shifted-versions of themselves, resulting in Multi-Path Interference (MPI). Any means to suppress or avoid MAI and MPI increases the capacity of the CDMA system. Hence, extensive research has been performed on enhanced receiver techniques that combat MPI and MAI.

Expensive equalization techniques (both linear and nonlinear) are used at the receiver to nullify the effects of MAI and MAI, and to recover the transmitted signal. Linear interference suppression equalizers such as Linear Minimum Mean Square Error (LMMSE) equalizers have proven to be a promising approach to enhance the performance of downlink channels in CDMA-based systems. They provide a good balance between performance and complexity, result in simple adaptive implementations, and exhibit reasonable robustness.

Compute-intensive wireless techniques, such as LMMSE equalizers, traditionally have been implemented using Application Specific Integrated Circuits (ASICs). ASICs achieve high performance at the expense of flexibility. Software Defined Radio (SDR) [5] is an alternative programmable platform that is being increasingly adopted by the wireless industry due to dramatically reduced development and hardware costs, accelerated time-to-market, increased flexibility, and upgradeability.

In this paper, we address the computational challenges of implementing the LMMSE equalizer on Software Defined Radio (SDR) platforms. We present and analyze a QR
Decomposition (QRD) [3-4] based back-substitution algorithm to perform the matrix inversion of a 15x15 complex-valued matrix for LMMSE equalization. When profiling the LMMSE algorithm on the SB3000 platform [6-7], a state-of-the-art SDR processor, we observed that over 97% of the computation time is spent on vector rotation operations. Convenient ways to perform complicated matrix operations, transcendental functions and division using the iterative CORDIC (COordinate Rotation Digital Computer) algorithms have been proposed previously [8-11]. However, the sequential CORDIC algorithm is inefficient to implement completely in software using conventional SDR processors. Consequently, Instruction Set Architecture (ISA) extensions and hardware designs based on the CORDIC algorithms can enable SDR architectures to meet the performance requirements of LMMSE equalization.

We propose, discuss and evaluate different design choices for CORDIC ISA extensions when implementing the CORDIC algorithm on a SDR architecture. We evaluate the proposed CORDIC ISA extensions on the SB3000 platform by augmenting the Sandblaster tool chain with the proposed CORDIC ISA extensions. Our investigations demonstrate a speedup of more than 4.5x on 15x15 QRD and LMMSE equalization algorithms when using the CORDIC ISA extensions compared to a non-CORDIC baseline software implementation on the SB3000, which uses powerful Single Instruction Multiple Data (SIMD) DSP instructions. The CORDIC-based implementations also have better numerical accuracy than the non-CORDIC baseline software implementation. This paper makes the following contributions:

- It addresses the high computational complexity of QR decomposition, complex-valued matrices, and LMMSE equalization on SDR platforms. It presents dramatic speedups when using the CORDIC ISA extensions.
- It investigates the class of architectures extended with CORDIC functional units [15][16], and performs an analysis to determine the set of CORDIC ISA extensions that provide appropriate performance in terms of computing speed and numerical accuracy when operating on large complex-valued matrices.

The remainder of the paper is structured as follows. Section II provides background information on CDMA Rake receivers, LMMSE equalization, the QRD algorithm, CORDIC, and the SB3000 SDR platform. Section III discusses various design considerations for CORDIC ISA extensions and describes our proposed ISA extensions. Section IV describes the simulation methodology and compares performance and numerical accuracy results from CORDIC-based implementations with those from the non-CORDIC baseline software implementation. It also provides hardware synthesis estimates for functional units that implement the proposed CORDIC ISA extensions. Section V summarizes our observations.

II. BACKGROUND

A. CDMA Rake Receiver

The Rake receiver [2] is the most commonly used receiver in CDMA systems due to its simplicity and reasonable performance. It is utilized to achieve path diversity in a multipath fading channel by effectively collecting the signal energy from each received multipath using a delay line at the receiver. A correlation with the spreading code $c$ is performed on each delay path. Due to the orthogonal nature of the spreading codes, they yield highly peaked autocorrelation properties, effectively separating the received signal into its component paths. The different signal paths are combined using a Maximal Ratio Combining (MRC) scheme, whereby the signal on each path is weighted according to its received power prior to combining the signals. The Rake receiver performs reasonably well with a limited number of users and long spreading codes, but its performance is severely degraded by MPI and MAI in the presence of a large number of users and/or short spreading codes. The length of the spreading code is defined by the communication standard. The longer the spreading code, the more resilient it is to MPI and MAI, but longer spreading codes result in reduced bit rates. Shortening the spreading code increases the bit rate, at the cost of a loss of orthogonality in the spreading code. For example, Wideband CDMA supports spreading code lengths from 4 to 512 with data rates up to 2 Mbps, while its evolution, HSPA, allows a fixed spreading code length of 16, but supports higher data rates of up to 14.4 Mbps in downlink and 7.2 Mbps in uplink. The higher data rates in HSPA are achieved through high modulation schemes (16 QAM) and short spreading codes.

Equalizers are used in the receiver to suppress the effects of MPI and MAI. The optimum equalization scheme is Maximum Likelihood (ML) based estimation. However, since the complexity of the ML estimator grows exponentially with the number of users, sub-optimal equalizers, such as the Zero Forcing (ZF) equalizer and Linear Minimum Mean Square Error (LMMSE) equalizer, have been considered for practical implementation. The ZF equalizer completely restores the orthogonality of the user signals, which could have been partially destroyed by MPI. However, in the presence of noise, the ZF equalizer will also boost the noise level and hence it has been regarded as an unreliable solution. The LMMSE equalizer, in comparison, factors in the presence of noise in the system as well as MPI and MAI. Therefore, the LMMSE equalizer has been proposed as a compromise between computational complexity and communication system performance.

B. LMMSE Equalization

The observation $Y$ of a transmitted sequence $X$ in a wireless system can be modeled as shown in Equation 1.

$$Y = XH + n$$

where $Y$ is the received chip sequence $[y_1, y_2, \ldots, y_N]^T$ with observation length $N$, $H$ is the $N \times M$ channel matrix, $X$ is the
transmitted chip sequence \( [x_1, x_2, \ldots, x_M]^T \), and \( n \sim N(0, \sigma_n^2) \) is Additive White Gaussian Noise (AWGN) in a system with zero mean and variance \( \sigma_n^2 \). The LMMSE solution provides a sub-optimal solution for the estimated transmitted chip sequence \( \hat{X} \), as shown in Equation \( 2 \). The LMMSE equalizer produces an equalized vector of transmitted symbols deduced from the received signal under the Minimum Mean Square Error criteria, which gives:

\[
\hat{X} = H^H \left[ HH^H + \frac{\sigma_n^2}{\sigma_s^2} I_N \right]^{-1} Y
\]  

(2)

where \( X \) is the transmitted chip sequence \( [\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_M]^T \), \( HH^H \) is the \( N \times N \) channel correlation Toeplitz matrix, \( \sigma_n^2 \) is the noise variance, \( \sigma_s^2 \) is the signal power, and \( I_n \) is the \( N \times N \) identity matrix.

The LMMSE equalizer is modeled as a Finite Impulse Response (FIR) filter with the filter tap coefficients optimized using the LMMSE criteria. The output of the LMMSE equalizer can be expressed as shown in Equation \( 3 \).

\[
\hat{X} = WY
\]  

(3)

\[
\hat{x}_k = \sum_{k=0}^{K-1} w_k y_k
\]

(4)

where \( W = [w_1, w_2, \ldots, w_K] \) is the channel tap coefficients of the FIR filter, \( \hat{X} \) is the estimated transmitted chip sequence, \( Y \) is the received chip sequence, and \( K \) is the number of FIR filter taps. The number of taps in the FIR structure is established through an empirical iterative method, i.e. the length of the filter is increased until the performance dictated by the specific communication system is achieved. For this paper, the number of filter taps is assumed to be equal to the number of channel response coefficients \( (K=N) \). In this case, the vector of FIR filter tap coefficients \( W \) is shown in Equation \( 4 \).

In Equation \( 7 \), \( h^H \), \( Q \), and \( R \) are known. Next, the classical back-substitution technique is performed to compute \( WQ \). Since \( Q \) is a unitary matrix, such that \( QQ^H = I \), \( WQ \) is multiplied by \( Q^H \) to find \( W \), the vector of FIR filter coefficients.

C. QR Decomposition

QR decomposition is a commonly used factorization technique in linear algebra to factor a matrix into the product of an orthogonal matrix and a triangular matrix. The QR decomposition of an \( M \times N \) matrix, \( A \), is given by Equation \( 8 \).

\[
A = QR
\]  

(8)

where \( Q \) is a \( M \times M \) unitary matrix, \( R \) is a \( N \times N \) upper triangular matrix, and \( \theta \) is an \( (M-N) \times N \) all-zero matrix. QR decomposition algorithms are computationally expensive and their computation complexity is on the order of \( O(N^3) \) for a \( N \times N \) matrix.

The Givens rotation algorithm for QR decomposition is chosen for this paper since it exhibits a regular computation pattern and can be easily vectorized on SIMD architectures. The Givens rotation technique is used to selectively introduce a zero into a matrix. A real Givens rotation to introduce a zero into a matrix at the position of \( b \) is shown in Equation \( 9 \).

\[
\begin{bmatrix}
\cos \theta & \sin \theta \\
\-\sin \theta & \cos \theta
\end{bmatrix}
\begin{bmatrix}
a \\
b
\end{bmatrix}
= 
\begin{bmatrix}
r \\
0
\end{bmatrix}
\]  

(9)

where \( r = \sqrt{a^2 + b^2} \), \( \theta = \tan^{-1} \left( \frac{b}{a} \right) \).

The real Givens rotation can be generalized to the complex number system. A complex-valued Givens rotation can be described in terms of two rotation angles, as formulated by Cavallaro et al. [12].

When the Givens rotation is used to perform QR decomposition on the \( M \times N \) matrix, \( A \), the rotation angles are chosen to zero out the sub-diagonal elements one at a time to obtain the upper triangular matrix \( R \). At each step, the rotation angles used to zero out a sub-diagonal element are also
propagated across the corresponding rows. The unitary matrix $Q$ is obtained by beginning with an identity matrix and performing all rotations propagated across the rows of the $R$ matrix on the corresponding rows of the $Q$ matrix. The CORDIC algorithm described in the next section is a convenient way to perform these rotations.

### D. CORDIC

CORDIC is an iterative algorithm to perform vector rotations in a 2-dimensional plane using simple shift and add/subtract operations. The CORDIC algorithm was introduced by Volder [8] for the circular and linear coordinate systems. It is used for rotation of vectors, determination of a vector’s magnitude and phase, computation of trigonometric and transcendental functions, multiplication, division and data type conversion. Later, Walther generalized CORDIC to the hyperbolic coordinate system to compute hyperbolic functions [9].

Equation 10 presents a generalized set of equations that defines the CORDIC algorithm and is applicable to multiple coordinate systems as:

$$
\begin{align*}
    x[i+1] &= x[i] - m \cdot \sigma[i] \cdot 2^{-i} \cdot y[i] \\
    y[i+1] &= y[i] + \sigma[i] \cdot 2^{-i} \cdot x[i] \\
    z[i+1] &= z[i] - \sigma[i] \cdot \alpha[i] 
\end{align*}
$$

(10)

where $x$ and $y$ are the vector coordinates, $z$ is the angle accumulator, $\sigma$ is the direction of rotation, $m = 1$ and $a[i] = \tan^{-1}(2^{-i})$ for the circular coordinate system, $m = 0$ and $a[i] = 2^i$ for the linear coordinate system, and $m = -1$ and $a[i] = \tanh^{-1}(2^i)$ for the hyperbolic coordinate system.

There are two modes of operation defined by the CORDIC algorithms. The rotation mode is used to rotate a vector by a specified rotation angle. The rotation decision made in each iteration decreases the magnitude of the residual angle in the angle accumulator, $z$. The vectoring mode is used to rotate the input vector to align the result vector with the X axis. The result of the vectoring operation is the angle and scaled magnitude of the original vector. The rotation decision at each iteration is made to decrease the magnitude of the $y$ coordinate.

Considerable research has been done on hardware implementations of the CORDIC algorithms for different applications [10-11]. Unlike related work in this area, this is the first time CORDIC ISA extensions have been used to perform LMMSE equalization on a programmable DSP for SDR.

#### E. The Sandbridge Sandblaster 3000

The SB3000 SDR platform [6-7] is designed to exploit parallelism inherent in emerging communication applications at different levels of granularity. It has four Sandblaster DSP cores and an ARM core. Each Sandblaster DSP core supports multithreading and has eight hardware threads. The Sandblaster DSP features compound instructions that can issue three parallel operations in a single cycle. It has a powerful instruction set architecture that supports saturating arithmetic, SIMD vector dot products, and SIMD vector multiply-accumulate operations.

The Sandblaster DSP microarchitecture is partitioned into three units: a program flow control unit, an integer and load/store unit, and a SIMD Vector Processing Unit (VPU). The SIMD VPU consists of four vector processing elements (VPEs), a shuffle unit, a reduction unit, and an accumulator register file. The four VPEs perform arithmetic and logic operations in SIMD fashion on 16-bit, 32-bit, and 40-bit fixed-point data types. High-speed 64-bit data busses allow each VPE to load or store 16 bits of data each cycle in SIMD fashion. Vector instructions have four execute stages in the pipeline. Since there is considerable data-level parallelism in the algorithms under investigation, SIMD vector CORDIC ISA extensions are proposed in this paper.

### III. PROPOSED CORDIC ISA EXTENSIONS

This section discusses the various considerations that affect the design of CORDIC ISA extensions and describes our proposed ISA extensions. In the SB3000, our CORDIC ISA extensions are implemented as SIMD vector operations, but they can easily be modified to provide scalar CORDIC operations.

#### A. Convergence

The CORDIC algorithm, summarized in Equation 10, performs vector rotations by splitting the rotation angle $\theta$ into a sequence of fixed micro-rotation angles as shown in Equation 11.

$$
\theta = \pm \alpha[0] \pm \alpha[1] \pm \alpha[n-1] = \sum_{i=0}^{n-1} \alpha[i] \cdot \sigma[i]
$$

(11)

where $n$ is the total number of CORDIC iterations, $\alpha[i]$ is the fixed micro-rotation angle and $\sigma[i]$ is the direction of rotation, -1 or +1, at iteration $i$. In order to guarantee convergence, the chosen set of fixed micro-rotation angles must satisfy the two convergence criteria shown in Equation 12 and 13.

$$
\alpha[i] - \sum_{j=i+1}^{n-1} \alpha[j] \leq \alpha[n-1]
$$

(12)

Equation 12 enforces the constraint that if, at any iteration $i$, the remaining rotation angle, $\alpha$ is zero, it will be changed to $\pm \alpha[i+1]$ in the next iteration. Then, the sum of the remaining fixed micro-rotation angles has to be large enough to bring the remaining rotation angle to within $\alpha[n-1]$ after the last iteration ($n-1$).

$$
\theta \leq \sum_{i=0}^{n-1} \alpha[i] + \alpha[n-1]
$$

(13)
For the circular coordinate system, with 16 CORDIC iterations, if the iteration sequence is chosen as \( i = 0 \) to 15, the sum of all fixed micro-rotation angles approaches ±100°. If the rotation angle \( \theta \) for any circular CORDIC operation is restricted to be within ±100° then this iteration sequence is suitable. Since the circular coordinate system is periodic about ±90°, the vector rotation angles can be limited to ±90°. For rotations greater than ±90°, we can apply Equation 14 to bring the angles within the convergence range.

\[
\begin{align*}
x' &= -d \cdot y \\
y' &= d \cdot x \\
z' &= z + d \cdot (\pi/2)
\end{align*}
\]

where \( d = \begin{cases} 1 & \text{if } y < 0 \\ -1 & \text{otherwise} \end{cases} \)

B. Precision Requirements and Operand Representation

While designing the CORDIC ISA extensions, precision and computational accuracy are critical because, if the error introduced by the use of CORDIC ISA extensions exceeds the allocated system error budget, it can adversely affect the reliability of the wireless system.

The finite word length also results in accumulation of rounding errors while computing the coordinates \( x \) and \( y \). The impact of these rounding errors can be reduced by using additional fraction guard bits. If \( n \) is the number of CORDIC iterations, \( W \) is the number of accurate fractional bits desired in the output, and \( C \) is the number of additional fractional guard bits used in intermediate computations, the rounding error can be considered to have a minor effect on the output accuracy if Equation 15 is satisfied.

\[ n \cdot 2^{-(W+C)} < 2^{-W} \]  

which yields \( C \geq \log_2(n) \). Hence, at least \( \log_2(n) \) additional fractional bits should be provided to reduce the impact of rounding errors to at most one unit in the last place (ulp) [13]. For this study, 20-bit internal precision is maintained within the CORDIC functional unit to produce 16-bit outputs.

The coordinates \( x \) and \( y \) are represented as two’s complement numbers. The angle accumulator, \( z \), that tracks the residual angle of rotation after each CORDIC iteration can be represented in radian format as a two’s complement number. The disadvantage of the radian format is that it does not provide wrap around capability for the angle accumulator. We keep the angle representation in radian format because angles are typically represented in the radian format in communication protocols.

In fixed-point formats, the CORDIC iterations for the \( x \) and \( y \) coordinate data paths return the same number of integer and fractional bits (Q format) as the input. In the \( z \) data path, pre-computed fixed micro-rotation angles have to match the Q format of the input angle. It is beneficial to provide configurability for the Q format in the \( z \) data path to accommodate varying precision requirements from different communication protocols. We implement this configurability by keeping more bits for extra precision of the fixed micro-rotation angles within the CORDIC unit and choosing the right number of bits to match the input Q format using a configurable CORDIC angle precision register.

C. CORDIC Iterations

The CORDIC iteration is a four-operand operation, with coordinates \( x \) and \( y \), angle accumulator \( z \), and iteration counter \( i \), as shown in Equation 10. Each CORDIC iteration reads all four operands and updates the value of all four operands. The final values of two operands (coordinate \( y \) or angle \( z \), and the iteration counter \( i \)) are not needed after the fixed number of CORDIC iterations. Consequently, if all required CORDIC iterations can be performed in a single DSP instruction, then only two operands need to be written back. However, it is unlikely that all CORDIC iterations can be performed within one DSP instruction given the limited number of execute stages in a typical DSP pipeline. For example, in the SB3000 platform, four CORDIC iterations fit in the four available execute pipeline stages of a vector instruction. If not all required CORDIC iterations can be performed by a single DSP instruction, then all four operands, including the iteration counter, must be written and reread between successive CORDIC instructions.

DSP instructions typically have one to three input operands and one output operand. If all four operands of the CORDIC iteration need to be read from and written back to registers, it poses a challenge. In typical DSP architectures with 32-bit registers, two 16-bit CORDIC operands \( x, y \) and \( z \) are 16 bits and \( i \) is at most 4 bits) can be packed into one register. They can be unpacked into 20-bit zero padded registers in the CORDIC unit before being used. Two 32-bit input registers can thus be used to pass the four input CORDIC operands. The write back of all four CORDIC operands (two output registers) still remains a challenge. We have explored a couple of design options to alleviate this problem.

D. Full-CORDIC Approach

The full-CORDIC approach augments the CORDIC unit with an auxiliary register that is read and written implicitly by each CORDIC instruction. This auxiliary register holds two CORDIC operands between CORDIC instructions. The other two operands are packed into one 32-bit register that is used as both a source and target register for the CORDIC instruction. The auxiliary register is setup using a special instruction before the CORDIC iterations. The two CORDIC operands that are not needed as results; \( y \) or \( z \), and \( i \), are assigned to these auxiliary registers. This choice is logical because these two operands need not be written back after all iterations are complete and \( i \) requires fewer bits than the other operands.

Four kinds of instructions are proposed for the full-CORDIC approach: Configure Set CORDIC (CFG_SET_CORDIC), Configure Read CORDIC (CFG_READ_CORDIC), CORDIC Rotate (ROT_CORDIC), and CORDIC Vector (VEC_CORDIC).
CFG_SET_CORDIC instruction is be used to initialize the auxiliary register and the CFG_READ_CORDIC instruction is used to read the auxiliary register in the CORDIC unit. The ROT_CORDIC and the VEC_CORDIC instructions can be used in Circular, Linear or Hyperbolic modes to perform four CORDIC iterations in rotation mode or vectoring mode.

The full-CORDIC approach is straight forward to implement in a typical DSP architecture. However, the hardware overhead due to multiple copies of the auxiliary register becomes substantial compared to the hardware cost of the entire CORDIC functional unit when considering a multithreaded SIMD architecture. The other consideration is saving and restoring this additional state of the CORDIC unit during interrupt processing.

E. Semi-CORDIC Approach

The semi-CORDIC approach splits each CORDIC operation into two semi-CORDIC operations, each executed using separate semi-CORDIC instructions. The semi-CORDIC instructions read x, y, z, and i packed in two 32-bit registers. One semi-CORDIC instruction writes back x and y packed into one 32-bit register while the other semi-CORDIC instruction writes back z and i packed into a second 32-bit register. XY CORDIC Rotation (XY_ROT_CORDIC), and ZI CORDIC Rotation (ZI_ROT_CORDIC) are the two semi-CORDIC rotation instructions. XY_ROT_CORDIC computes the new value of x and y using i and the sign of z. ZI_ROT_CORDIC then updates the value of z and i. Since the XY_ROT_CORDIC instruction depends on the current values of z and i, it has to be executed before the ZI_ROT_CORDIC instruction. ZI CORDIC Vectoring (ZI_VEC_CORDIC) and XY CORDIC Vectoring (XY_VEC_CORDIC) are the two semi-CORDIC vectoring instructions. ZI_VEC_CORDIC computes the new value of z and i, based on i and the sign of y. XY_VEC_CORDIC computes the new value of x and y, using the previous value of i and the sign of y. Since the ZI_VEC_CORDIC instruction depends on the sign of the current y value, it has to be executed before XY_VEC_CORDIC. Even though the dependency on i is violated in this case, the previous value of the iteration counter can be determined easily by subtracting the number of iterations per instruction.

All four instructions can be used in Circular, Linear or Hyperbolic modes. Depending on the type of CORDIC operation, the respective sequential semantics between the semi-CORDIC instructions have to be maintained. The semi-CORDIC approach readily fits into a traditional RISC architecture with two source operands and one destination operand per instruction. This approach does not have the drawbacks of the full-CORDIC approach, but it almost doubles the number of CORDIC instructions needed to implement a CORDIC operation.

F. Refined CORDIC Approach

While performing the QR decomposition using Givens rotations, multiple elements in the R matrix and Q matrix are rotated by the same angles. If the 16-bit direction vectors, σ, for the rotation angles are stored while performing the corresponding CORDIC vectoring operation, they can be reused in the CORDIC rotation operations, eliminating the need to re-compute them in every CORDIC rotation operation. The iteration counter, i, uses at most four bits of the available 16 bits to count 16 iterations. The most significant bits of the register that holds the iteration counter can be used to store the direction vectors. The last instruction of the CORDIC operation would, however, overwrite the least significant bits. This does not pose a problem because the final value of the iteration counter is not needed. We also store the direction vector in the CORDIC rotation operations, because it has low hardware overhead and can be utilized when multiple vectors are rotated by the same angle.

In the full-CORDIC approach, 16 bits are added to the auxiliary register to store the direction vectors. After all instructions of the full-CORDIC vectoring operation, an additional CFG_READ_CORDIC instruction is executed to read the direction vectors from the auxiliary register. Reusing these direction vectors for subsequent CORDIC rotation operation can result in energy savings by turning off the z data path.

In the semi-CORDIC approach, the ZI_VEC_CORDIC and ZI_ROT_CORDIC instructions are modified to capture the direction vector. An additional semi-CORDIC rotation instruction, XYS_ROT_CORDIC, is introduced. The inputs to this instruction are the coordinates, x and y, and the direction vector, σ. Computations on the XY datapath are performed using the direction vector. The ZI datapath is inactive. The iteration counter, i still needs to be incremented and stored after every CORDIC instruction. This can either be done in software or multiple instruction opcodes can be used that implicitly specify the starting iteration number. This refinement reduces the number of semi-CORDIC instructions and increases energy efficiency. It is referred to as the refined semi-CORDIC approach in the rest of this paper.

IV. SIMULATION METHODOLOGY AND RESULTS

A. Simulation Methodology

As stated in Section II, the LMMSE Equalizer is modeled as a FIR filter with 15 taps. The computation of these 15 FIR filter coefficients involves a 15x15 complex-valued matrix inversion. The QRD based back-substitution technique is used for the LMMSE equalizer coefficient computation. Since the Sandblaster has the capability for 4-way SIMD processing, the code for QRD using Givens rotations is re-written to take advantage of SIMD processing. The first step of zeroing out a sub-diagonal matrix element can be performed simultaneously on multiple sets of rows, but this involves a lot of data movement to accommodate SIMD processing. Since the first step contributes to less than 10% of the computation time, we zero out one matrix-element at a time. The second step of propagating the rotations across the remaining elements in the affected rows and in the corresponding rows of the Q matrix, which is about 85% of the computation time, is performed in parallel using SIMD processing. Since the SB3000 supports
40-bit SIMD elements, all 20 bits of the x, y and z CORDIC operands are retained between CORDIC instructions.

The non-CORDIC baseline software implementation (with no ISA extensions) for QRD and LMMSE equalizer coefficient computation use polynomial approximations and table lookups to evaluate trigonometric functions and perform division in fixed point. An optimized fixed-point assembly implementation of the CORDIC algorithm using the SB3000 instructions (with no ISA extensions) was also used to implement QRD and LMMSE equalizer to provide another software-only implementation.

To study the effects of the proposed CORDIC ISA extensions on the LMMSE equalizer, they were added to the SB3000 tool chain. The CORDIC instructions were simulated with four CORDIC iterations per instruction. The proposed refinement applied to the semi-CORDIC approach was also investigated. Since the compiler is not able to automatically generate the proposed CORDIC instructions from the C application code, the 15 x 15 complex-valued matrix QRD and LMMSE equalizer coefficient computation under investigation were re-written using intrinsic functions that the compiler replaces with CORDIC instructions. CORDIC instructions are also used in the division operations in the back-substitution step. The functional correctness, precision, and performance of the modified QRD and LMMSE algorithms were studied with the help of the modified SB3000 DSP cycle accurate simulator and compared against the baseline implementation.

The performance is represented as a speedup normalized to the non-CORDIC baseline implementation. The computational speedup is computed as shown in Equation 16, since each instruction take one thread cycle.

\[
\text{Speedup} = \frac{\text{Dynamic instruction count of non-CORDIC baseline implementation}}{\text{Dynamic instruction count of new implementation}}
\]

The arithmetic error is computed by comparing all fixed-point implementations against a floating-point implementation. The arithmetic error is represented as a Normalized Root Mean Square percentage error and is computed using Equation 17.

\[
\text{Error} = \frac{\sum (\text{Value}_{\text{float}} - \text{Value}_{\text{fixed to float}})^2}{\sum \text{Value}_{\text{float}}^2} \times 100
\]

### B. Performance Results

Figure 1 shows the speedup of various CORDIC algorithm implementations normalized to the non-CORDIC baseline software implementation. The optimized software-only fixed-point assembly implementation of the CORDIC algorithm with no ISA extensions is slower than the non-CORDIC baseline software implementation for both QRD and LMMSE equalization. This is because the sequential CORDIC algorithm is slow when implemented completely in software. All implementations of the QRD and LMMSE equalization algorithms that use CORDIC ISA extensions exhibit significant speedup compared to the non-CORDIC baseline software implementation. The full-CORDIC, semi-CORDIC and refined semi-CORDIC implementations demonstrate more than 4.5x speedups while performing the QRD of a 15 x 15 complex-valued matrix and the LMMSE equalizer coefficient computation. As expected the semi-CORDIC approach is slower than the full-CORDIC approach, because it almost doubles the number of CORDIC instructions. As seen in Figure 1, the refined semi-CORDIC improves the performance by storing the direction vector and re-using it.

### C. Arithmetic Error

All CORDIC-based implementations demonstrated similar accuracy and are better than the standard Sandblaster implementation. With the non-CORDIC baseline software implementation, the arithmetic error for the computed FIR coefficients was 2.14%, while with the CORDIC implementations the arithmetic error reduces to 0.75%. The accuracy of the baseline non-CORDIC implementation can be improved by adding higher order terms to the polynomial approximations, but this increases computational complexity.

### D. Hardware Synthesis Estimates

Hardware synthesis for CORDIC functional units was performed using Synopsys Design Compiler and TSMC’s tcbn65gplus 65nm CMOS standard cell library. All environmental and process parameters were set to their nominal or typical conditions. In this technology, a fan-out-of-four (FO4) inverter’s delay is 31.3 ps.

All CORDIC functional units are designed to fit within the SB3000 SIMD vector pipeline with four SIMD elements and four execute pipeline stages. Additional details regarding the CORDIC functional units and the impact of the CORDIC ISA extensions on power dissipation of wireless algorithms are presented by Senthilvelan [17]. The designs are synthesized with a clock constraint of 600 MHz to match the clock speed of the SB3000. For comparison, the SB3000 SIMD-Vector-Multiply-Accumulate (VMAC) instruction data path is also synthesized in the same environment. Figure 2 presents the synthesis area, delay and power estimates for CORDIC functional units that implement full-CORDIC, semi-CORDIC and refined semi-CORDIC approaches normalized to the SB3000 VMAC datapath.
The combinational areas of the CORDIC units are similar to the combinational area of the SB3000 VMAC units. All designs contain pipeline registers to support four execute pipeline stages. The CORDIC units contain three 250-bit pipeline registers. In addition to the pipeline registers, the full-CORDIC approach also has auxiliary registers for storing the two CORDIC operands in between instructions. The auxiliary registers account for 47% of the non-combinational area in the full-CORDIC approach making its CORDIC functional units much larger than with the other two approaches.

The delays of all approaches are similar because the synthesis was performed to meet a clock frequency of 600MHz. The critical paths for all three CORDIC approaches pass through the z data path. The semi-CORDIC and the refined semi-CORDIC approaches add a 3-bit constant adder and multiplexer that are used to re-generate the value of the iteration counter to this critical path, which causes a small increase in delay.

V. SUMMARY

The CORDIC-augmented Sandblaster implementations exhibit significant speed-ups and provide better arithmetic accuracy than the standard Sandblaster non-CORDIC implementation when performing QRD on a 15 x 15 complex-valued matrix and computing the LMMSE equalizer coefficients. The LMMSE algorithm is also used in next generation Multiple-Input Multiple-Output (MIMO) systems. Improving the computational performance on the LMMSE algorithm will help SDR platforms meet the performance requirements of future generation wireless protocols. The CORDIC algorithms can also be used to efficiently implement trigonometric functions, transcendental functions, and division, which are commonly used in wireless protocols. The penalty is the additional silicon area consumed by the CORDIC functional units.

Of the ISA extensions presented, the refined semi-CORDIC approach provides significant performance benefits with a reasonable hardware overhead. Other variations of the CORDIC algorithm proposed in the literature can potentially yield further performance improvements. The general techniques presented in this paper are applicable to other high-performance DSP systems.

REFERENCES