Low-density Parity-check Decoding on the Sandbridge Sandblaster® SDR Platform

Murugappan SENTHILVELAN [† §], Meng YU [†], Daniel IANCU [†¶], Mihai SIMA [*], Michael SCHULTE [§ ↓]

[†]Optimum Semiconductor Technologies, Inc., Tarrytown, N.Y., U.S.A.
[*] University of Victoria, Department of Electrical and Computer Engineering, Victoria, B.C., Canada.
[↓] Advanced Micro Devices, Research and Advanced Development Labs, Austin, Tex., U.S.A
[§] University of Wisconsin-Madison, Department of Electrical and Computer Engineering, Madison, Wis., U.S.A.
[¶] Tampere University of Technology, Korkeakoulunkatu 1, FIN-33720 Tampere, Finland.

e-mail: {msenthilvelan, myu, diancu}@optimumsemi.com, msima@ece.uvic.ca, schulte@engr.wisc.edu

***Abstract* — Wireless protocols strive to increase spectral efficiency and achieve high data throughput. Low-density parity-check (LDPC) codes are advanced forward error correction (FEC) codes that use iterative decoding techniques to achieve Shannon capacity. Due to their superior performance, state-of-art wireless protocols such as WiMAX and LTE Advanced are adopting LDPC codes. LDPC codes come with the high cost of drastically increased computational effort for decoding. Among the proposed decoding algorithms, the belief propagation (BP) algorithm leads to a good approximation of an optimal ideal decoder; however, it uses compute-intensive hyperbolic trigonometric functions. To reduce the computational complexity, typical LDPC decoder implementations use simplified algorithms such as the min-sum algorithm at the expense of reduced signal processing performance. Efficient and accurate ways of computing hyperbolic trigonometric functions can facilitate the use of the BP algorithm in real-time LDPC decoder implementations. This paper investigates hyperbolic COordinate Rotation DIgital Computer (CORDIC) instruction set architecture (ISA) extensions for software-defined radio (SDR) processors to efficiently compute the hyperbolic trigonometric functions. The CORDIC ISA extensions are evaluated on the low-power multi-threaded Sandbridge Sandblaster® SB3000 platform. The computational performance, numerical accuracy, hardware estimates, power consumption estimates, and memory requirements with the CORDIC ISA extensions are compared to a baseline implementation on the SB3000.**

Keywords – LDPC decoding, Belief propagation algorithm, Hyperbolic tangents, CORDIC, SDR, ISA extensions

I. INTRODUCTION

The International Telecommunications Union (ITU) recommendations for the next generation (4G – Fourth Generation) wireless protocols (ITU-R M.1645) advocates maximum data rates of 100 megabits per second (Mbps) for high-mobility situations and 1 gigabit per second (Gbps) for stationary and low-mobility situations [1]. Current and next-generation wireless protocols propose sophisticated techniques including low-density parity-check (LDPC) [2] [3] forward error correction (FEC) codes to increase the wireless channel capacity and spectral efficiency. The advent of these new techniques rapidly increases the algorithmic complexity and computational requirements of wireless systems.

The LDPC codes are FEC codes that exhibit excellent error-correcting capabilities, close to the Shannon capacity. Due to their superior performance, state-of-art wireless protocols such as WiMAX and LTE Advanced are adopting the LDPC codes. The improved error-correction capabilities of LDPC codes come with the high cost of drastically increased computational effort for decoding. The LDPC decoder is implemented as an iterative message-passing algorithm between data nodes and parity-check nodes with termination criteria. Among the proposed LDPC decoding algorithms, the belief propagation (BP) algorithm [2] exhibits the highest error-correction capability and is a good approximation of an optimal ideal decoder [4]. However, in the BP algorithm, message computation at the parity-check nodes involves the compute-intensive evaluation of hyperbolic trigonometric functions. Simplified decoding algorithms such as the min-sum algorithm trade error correction capability for reduced computational complexity, and are used in typical LDPC decoder implementations. Efficient and accurate ways of computing hyperbolic trigonometric functions can facilitate the use of the BP algorithm in real-time LDPC decoder implementations.

Compute-intensive wireless techniques such as LDPC decoders 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 BP algorithm for LDPC decoding on SDR platforms. When profiling the BP algorithm on the SB3000

platform [6-7], a state-of-the-art SDR processor, we observed that more than 32% of the computation time is spent performing hyperbolic tangent computations.

Convenient ways to compute transcendental functions including hyperbolic trigonometric functions using the iterative COordinate Rotation DIgital Computer (CORDIC) 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 efficient implementation of the BP LDPC decoding algorithm on SDR processors.

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 speed-up of more than 1.2x on the BP LDPC algorithm 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 when evaluating the hyperbolic trigonometric functions. This paper makes the following contributions:

- It addresses the high computational complexity of the BP algorithm in LDPC decoding on SDR platforms. It presents the speed-ups 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 to evaluate hyperbolic trigonometric functions.

In the rest of this paper, Section II provides background information on LDPC codes, the BP decoding 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 evaluation methodology and compares computational performance, numerical accuracy, power consumption estimates, and memory requirements of implementations that use the CORDIC ISA extensions 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. Low-density Parity-check (LDPC) Codes

The LDPC codes, first proposed by Robert Gallager [2], are some of the most promising FEC codes. They are being adopted by wireless protocols such as LTE and WiMAX. The LDPC codes exhibit excellent error-correcting capability; close to the Shannon theoretical limits [18]. They are a class of linear block codes whose code words satisfy a set of linear parity-check constraints. The LDPC codes can be expressed in matrix form using parity-check matrices [2] or in graphical representation using bipartite graphs [3].

Each LDPC code has a set of parity-check constraints that is defined by an (m x n) parity-check matrix $\boldsymbol{H}$, whose m rows define the m parity-check constraints, and n columns represent the length of the code word. An entry (i, j) in the parity-check matrix is 1 if and only if the jth element of the code word is connected to the ith parity-check constraint. Then, the LDPC code is defined by a set of equations satisfying Equation 1.

$$\mathbf{H} \cdot \mathbf{c}^{\mathbf{T}} = 0 \qquad (1)$$

where $\boldsymbol{c}$ = ($c1, c2, \ldots, cn$) is the set of n elements of the code word. Wireless protocols typically specify the LDPC parity-check matrices in the physical layer specifications.

For example, the parity-check matrix for a LDPC code is shown in Equation 2.

$$\mathbf{H} = \begin{array}{c} \begin{matrix} c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \end{array} \begin{matrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{matrix} \qquad (2)$$

Using this parity-check matrix $\boldsymbol{H}$, the LDPC code is defined by Equation 3.

$$\begin{aligned} f_0 &= c_1 \oplus c_3 \oplus c_4 \oplus c_7 = 0 \\ f_1 &= c_0 \oplus c_1 \oplus c_2 \oplus c_5 = 0 \\ f_2 &= c_2 \oplus c_5 \oplus c_6 \oplus c_7 = 0 \\ f_3 &= c_0 \oplus c_3 \oplus c_4 \oplus c_6 = 0 \end{aligned} \qquad (3)$$

Two numbers, w_r and w_c, can be used to further characterize the parity-check matrix $\boldsymbol{H}$; w_r is the number of 1s in each row and w_c is the number of 1s in each column. For a matrix to be called low-density, the two conditions $w_c \ll n$ and $w_r \ll m$ must be satisfied. The sparsity of the matrix is the key property that provides the algorithmic efficiency of the LDPC codes.

An LDPC code is called a regular LDPC code if w_c is constant for every column. In other words, all the parity-check nodes should have the same number of incoming edges and all the element nodes of the code word should have the same number of incoming edges. The example LDPC structure shown in Equation 2 is regular, with $w_c = 2$ and $w_r = 4$. If the number of 1s in each row or column is not constant in the parity-check matrix $\boldsymbol{H}$, then the LDPC code is called an irregular LDPC code.

Tanner [3] introduced an effective graphical representation of LDPC codes using bipartite graphs that contain two distinctive sets of nodes and edges that only connect two nodes of different types. The two distinctive types of nodes in the Tanner bipartite graph are the variable nodes and the parity-check nodes. Figure 1 is an example of a Tanner bipartite graph with eight variable nodes and four parity-check nodes corresponding to the parity-check matrix ***H*** shown in Equation 2.

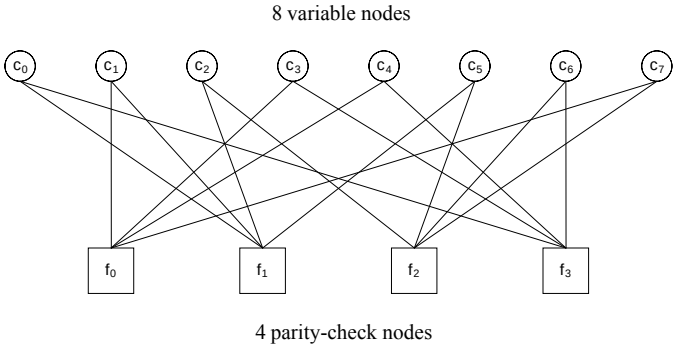

Figure 1. Tanner bipartite graph for a LDPC code.

The construction of the Tanner graph shown in Figure 1 is straightforward. A parity-check node f_i is connected to a variable node c_j if the element h_{ij} of the parity-check matrix ***H*** shown in Equation 2 is equal to 1.

LDPC codes are decoded iteratively using message-passing algorithms [2]. Message-passing algorithms involve passing of likelihood or belief messages from variable nodes to parity-check nodes and from parity-check nodes to variable nodes with termination criteria. The input to the decoding algorithm is a message vector giving the intrinsic likelihood of each bit being a 0 or 1. An iteration of LDPC decoding consists of a round of message passing from each variable node to all parity-check nodes connected to it, followed by another round of message passing from each parity-check node to all variable nodes connected to it. Decoding performance is achieved through repeated iterations of message passing along edges in the graph, with a termination criterion.

Messages from variable nodes to the parity-check nodes are computed based on the observed value at the variable node and messages that are passed on from other neighboring parity-check nodes in the previous iteration. An important aspect is that the message passed from a variable node *c* to a parity-check node *f* must not take into account the message sent from node *f* to node *c* in the previous iteration. The same is true for messages passed from the parity-check node to the variable nodes.

Among the proposed decoding algorithms, the original algorithm proposed by Gallager for LDPC decoding -- the BP algorithm -- leads to the best decoding performance [4]. However, the BP algorithm is computationally intensive because it uses hyperbolic tangents and arctangents. The Min-Sum algorithm is a simplified algorithm that trades computational complexity for reduced decoding performance.

B. Belief Propagation Algorithm

In the BP algorithm, the message passed from the variable node *c* to a parity-check node *f* is the likelihood *L* that *c* has a certain value based on the observed value at node *c* and input received in the previous iteration from other parity-check nodes connected to *c*. On the other hand, the message passed from parity-check node *f* to variable node *c* is the likelihood that *c* has a certain value based on the inputs received in the current iteration from other variable nodes connected to the parity-check node *f*.

The iteration steps in the BP algorithm are as follows:

Step 1: Messages are passed from each variable node *c* to all connected parity-check nodes *f* at the beginning of the iteration. The messages are represented as $m_{cf}[i]$, where *i* is the iteration number, and shown in Equation 4.

$$m_{cf}[i] = \begin{cases} m_c & \text{if } i = 0 \\ m_c + \sum_{f' \in (Fc - \{f\})} m_{f'c}[i-1] & \text{if } i \geq 1 \end{cases} \quad (4)$$

where m_c is the log-likelihood of the variable node *c,* conditioned on the observed value *x,* and is independent of any parity-check node *f*. $f' \in (F_c - \{f\})$ is the list of all parity-check nodes F_c connected to variable node *c,* except the parity-check node *f* for which the message $m_{cf}[i]$ is intended, and $m_{f'c}[i\text{-}1]$ is the message received from parity-check node *f'* in the previous iteration. The value of m_c is given by

$$m_c = \ln\left(L(x)\right) = \ln\left(\frac{\Pr[x=0]}{\Pr[x=1]}\right) \quad (5)$$

Pr[x = 0] is the probability that $x = 0$ and *Pr[x = 1]* is the probability that $x = 1$. For iterations in which the iteration counter *i* is greater than 0, the messages received from all parity-check nodes *f'* need to be considered. If y_k is the log likelihood message received from parity-check node $f_k \in f'$, then

$$\begin{aligned} \sum_{f' \in (Fc - \{f\})} m_{f'c}[i-1] &= \ln\left(L\left(x \middle| y_1, y_2, \ldots, y_{f'}\right)\right) \\ &= \sum_{k=1}^{f'} \ln\left(L\left(x \middle| y_k\right)\right) \end{aligned} \quad (6)$$

where $\ln\left(L\left(x \middle| y_k\right)\right) = \ln\left(\frac{\Pr[x=0 \middle| y_k]}{\Pr[x=1 \middle| y_k]}\right)$ and $\Pr[x=0 \middle| y_k]$ is the probability that $x = 0$ conditional on y_k.

Step 2: Messages are passed from each parity-check node *f* to all connected variable nodes. The messages are represented as $m_{fc}[i]$, as shown in Equation 7.

$$m_{fc}[i] = \ln\left(L\left(x_1 \oplus x_2 \oplus ... \oplus x_{c'} \mid y_1, y_2, ..., y_{c'}\right)\right)$$

$$= \ln\left(\frac{1+\prod_{c'\in(Cf-\{c\})} \tanh\left(m_{c'f}[i]/2\right)}{1-\prod_{c'\in(Cf-\{c\})} \tanh\left(m_{c'f}[i]/2\right)}\right) \quad (7)$$

$$= 2\tanh^{-1}\left(\prod_{c'\in(Cf-\{c\})} \tanh\left(m_{c'f}[i]/2\right)\right)$$

where i is the iteration number, $c'\in(C_f-\{c\})$ is the list of all variable nodes C_f connected to parity-check node f, except the variable node c for which that the message $m_{fc}[i]$ is intended, and $m_{c'f}[i]$ is the message received from variable node c' in the current iteration.

Steps 1 and 2 are repeated until convergence is obtained.

C. CORDIC

CORDIC is an iterative algorithm to perform vector rotations in a two-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 trigonometric functions [9].

Equation 8 presents a generalized set of equations that defines the CORDIC algorithm and is applicable to multiple coordinate systems as:

$$\begin{aligned} x[i+1] &= x[i] - m\cdot\sigma[i]\cdot 2^{-i}\cdot y[i] \\ y[i+1] &= y[i] + \sigma[i]\cdot 2^{-i}\cdot x[i] \\ z[i+1] &= z[i] - \sigma[i]\cdot\alpha[i] \end{aligned} \quad (8)$$

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

D. 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 multi-threading 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 ISA that supports saturating arithmetic, SIMD vector dot products, and SIMD vector multiply-accumulate operations.

The Sandblaster DSP micro-architecture 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 8, performs vector rotations by splitting the rotation angle θ into a sequence of fixed micro-rotation angles, as shown in Equation 9.

$$\theta = \pm\alpha[0] \pm \alpha[1] \ldots \pm \alpha[n-1] = \sum_{i=0}^{n-1} \sigma[i]\cdot\alpha[i] \quad (9)$$

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. To guarantee convergence, the chosen set of fixed micro-rotation angles must satisfy the two convergence criteria shown in Equation 10 and 11.

$$\alpha[i] - \sum_{j=i+1}^{n-1} \alpha[i] \quad \leq \quad \alpha[n-1] \quad (10)$$

Equation 10 enforces the constraint that if, at any iteration i, the remaining rotation angle z 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 $(n-1)$.

$$\theta \quad \leq \quad \sum_{i=0}^{n-1} \alpha[i] + \alpha[n-1] \quad (11)$$

Equation 11 ensures that the total rotation angle θ does not exceed the sum of all fixed micro-rotation angles plus the final micro-rotation angle.

The CORDIC algorithm produces one additional bit of accuracy each iteration. Since wireless algorithms typically operate on 16-bit data, 16 CORDIC iterations are performed. For the hyperbolic coordinate system, it has been shown [9] that certain iterations have to be repeated to satisfy the convergence criterion shown in Equation 10. In the basic CORDIC algorithm, the iteration sequence when executing 16 CORDIC iterations is chosen as i = 1, 2, 3, 4, 4, 5, … 13, 13, 14. For this iteration sequence, the sum of all fixed micro-rotation angles is ±64°. If the rotation angle θ for any hyperbolic CORDIC operation is restricted to be within ±64°, then this iteration sequence is suitable.

However, it was observed that the hyperbolic trigonometric functions in the BP algorithm require a convergence range close to ±180° for good decoding performance. Hence, the convergence range for the hyperbolic CORDIC algorithm was increased by performing two additional iterations with i = 0 and i = -1 (elementary rotation angles computed $\tanh^{-1}(1-2^{i-2})$) to yield a new convergence range of ±197°.

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 12 is satisfied.

$$n \cdot 2^{-(W+C)} < 2^{-W} \tag{12}$$

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 8. 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, 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 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:

1. Configure Set CORDIC (CFG_SET_CORDIC)
2. Configure Read CORDIC (CFG_READ_CORDIC)
3. CORDIC Rotate (ROT_CORDIC)
4. CORDIC Vector (VEC_CORDIC)

The CFG_SET_CORDIC instruction is 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 straightforward 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 multi-threaded 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.

IV. EVALUATION METHODOLOGY AND RESULTS

A. *Baseline Implementation*

The LDPC decoder specifications used for this evaluation are from the Mobile WiMAX IEEE 802.16e standard. The LDPC decoder data-block size considered is 2,304 bits with half-rate coding (1,152 data bits and 1,152 parity bits). The BP algorithm described in Section II is used for LDPC decoding. Four iterations of the BP algorithm are used to obtain a balance between decoding capability and computational complexity. The hyperbolic tangents (tanh) and hyperbolic arctangents (atanh) functions can be computed either using polynomial approximations or table lookup operations.

The polynomial approximations have low memory requirements, but are not very accurate. The table lookup operations produce results with better accuracy, but have a large memory footprint because of the pre-computed lookup tables. Since the BP algorithm was very sensitive to the accuracy of the hyperbolic functions, especially the atanh function, the hyperbolic functions are computed using table lookup operations. To meet the required numerical accuracy, the lookup tables are 8 KB and 32 KB for tanh and atanh, respectively. The large memory footprint for the table lookup approach is a disadvantage when using this technique on DSPs with small amounts of high-speed memory.

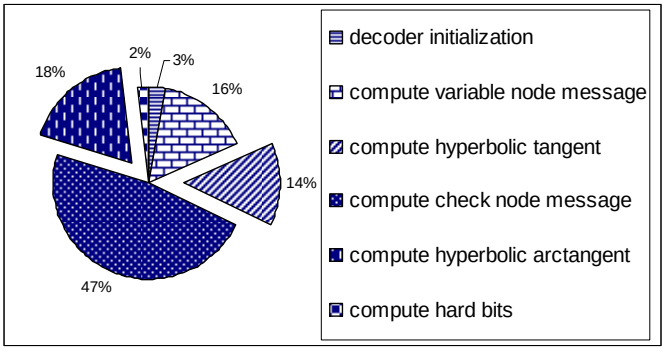

Figure 2. Computation time breakdown for the BP LDPC decoding algorithm on the Sandbridge Sandblaster SB3000.

The breakdown of the computation time utilized by various operations in the BP algorithm is shown in Figure 2. For a given LDPC code, the parity-check nodes and the

variables nodes appear to be connected at random, which is key to the superior error-correction performance of LDPC codes. However, this randomness results in a large memory overhead of reading and writing small chunks of data throughout memory. Hence, the computation of the messages in the parity-check nodes and variable nodes utilize nearly half the computation time. The computation of the tanh and atanh functions also contributes to a significant portion of the computation time, 14% and 18% respectively.

B. *Evaluation Methodology*

To study the effects of the proposed CORDIC ISA extensions on LDPC decoding, the ISA extensions were added to the SB3000 tool chain. The CORDIC instructions were simulated with four CORDIC iterations per instruction. Since the compiler is not able to automatically generate the proposed CORDIC instructions from the C application code, the BP algorithm for LDPC decoding under investigation was re-written using intrinsic functions that the compiler replaces with CORDIC instructions and optimizes. The functional correctness, numerical accuracy, computational performance, and power consumption of the modified BP algorithm was studied with the help of the modified SB3000 DSP cycle-accurate simulator and compared against the baseline implementation.

C. *Performance Results*

The performance is represented as a speed-up normalized to the non-CORDIC baseline implementation. The computational speed-up is computed as shown in Equation 13, since each instruction takes one thread cycle.

$$Speed\text{-}up = \frac{\left(\begin{array}{c}Dynamic\ instruction\ count\ of\\ non-CORDIC\ baseline\ implementation\end{array}\right)}{\left(\begin{array}{c}Dynamic\ instruction\ count\\ of\ new\ implementation\end{array}\right)} \qquad (13)$$

Figure 3 shows the speed-up of the full-CORDIC and semi-CORDIC algorithm implementations normalized to the non-CORDIC baseline software implementation.

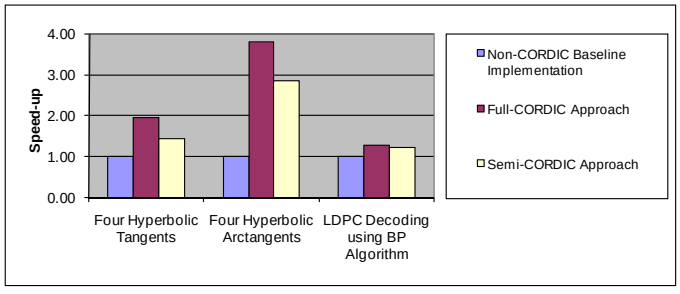

Figure 3. Speed-ups on tanh, atanh, and BP algorithm relative to the baseline non-CORDIC implementation.

Both CORDIC implementations of the BP algorithm that use CORDIC ISA extensions exhibit speed-up compared to the non-CORDIC baseline software implementation. The elementary hyperbolic tangent and arctangent operations demonstrate significant speed-ups when using the CORDIC ISA extensions. However, this speed-up is not translated to the BP algorithm completely because the speed-ups apply to 32% of the execution time of the BP algorithm, while the rest of the execution time remains unchanged. The full-CORDIC and semi-CORDIC implementations demonstrate speed-ups of 1.27x and 1.21x on the BP algorithm, respectively. As expected, the semi-CORDIC approach is slower than the full-CORDIC approach, because it almost doubles the number of CORDIC instructions. The LDPC decoder parity-check matrix used in the WiMAX standard has repetitive patterns that can be exploited to improve the memory performance of the algorithm. If these memory optimizations are utilized, then the speed-ups observed on the BP algorithm will be improved further.

D. *Arithmetic Error*

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 14.

$$Error = \sqrt{\frac{\sum\left(Value_{float} - Value_{fixed\_to\_float}\right)^2}{\sum\left(Value_{float}\right)^2}} \times 100 \qquad (14)$$

Since the output of the LDPC decoder is decoded bits, the arithmetic error is computed on the results of the tanh and atanh functions. Both 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 hyperbolic functions was 2.08%, while the CORDIC implementations reduced the arithmetic error to 0.05%. The accuracy of the baseline non-CORDIC implementation can be improved by increasing the resolution of the lookup tables, but this would result in the increase in memory requirements.

E. *Hardware Synthesis Estimates*

Hardware synthesis for CORDIC functional units was performed using Synopsys® Design Compiler and TSMC's tcbn65gplus 65-nm 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.

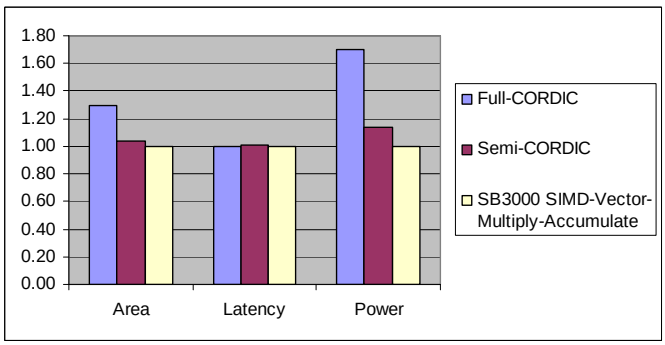

Figure 4. Hardware synthesis estimates for CORDIC functional units.

All CORDIC functional units are designed to fit in 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 4 presents the synthesis area, delay, and power estimates for CORDIC functional units that implement full-CORDIC and 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 600 MHz. 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 multi-plexer that are used to re-generate the value of the iteration counter to this critical path, which causes a small increase in delay.

F. Power Consumption Estimates

An analytical methodology using instruction type profiling and instruction power consumption measurements from real hardware is employed to estimate the power consumption of wireless algorithms [17]. Previous experiments indicate that this power model is accurate to within about 10% when modeling the power dissipation of compute-intensive wireless communication algorithms. The profiler in the Sandblaster simulator is used to perform the instruction type profiling on wireless algorithms.

The Sandblaster DSP's instruction set can be divided into the following instruction classes:

- Wide-vector instruction class (40-bit or 32-bit SIMD vector instructions)
- Vector instruction class (16-bit SIMD vector instructions)
- Scalar instruction class
- Control instruction class

These instruction classes are further divided into instruction sub-classes that have similar computational complexity in hardware. The power consumption for each of these instruction sub-classes is measured on hardware. The fraction of computation time each of these instruction sub-classes constitutes can be measured using simulator profiling. The power estimate for the entire application can be computed as a dot product of these two measurements.

To make a fair comparison when comparing power numbers computed from two different implementations and using this technique, both the implementations need to estimate power over the same amount of time. For the shorter implementation, after the necessary computations are completed, there can be three power scenarios for the remaining time:

1. Least-power scenario is one in which the processor is shut down after the computations are completed, yielding maximum power savings.
2. Nominal-power scenario is one in which the processor is clock gated to reduce the dynamic power.
3. Most-power scenario is one in which the processor is executing NOP instructions.

In practice, the type of power scenario would be decided based on the overhead of restoring normal functionality to the Sandblaster DSP after turning it OFF or putting it in stand-by mode.

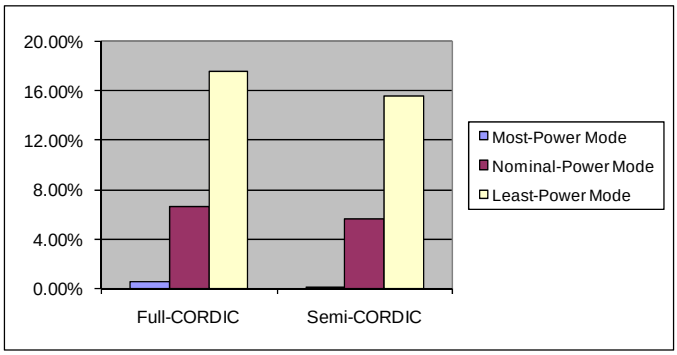

Figure 5. Percentage savings in power consumption on BP algorithm relative to the baseline non-CORDIC implementation.

Figure 5 shows the percentage savings in power consumption for the full-CORDIC and semi-CORDIC approaches relative to the non-CORDIC baseline implementation. Considerable power savings are obtained in the nominal- and least-power modes.

V. SUMMARY

The CORDIC-augmented Sandblaster implementations exhibit 1.2x speed-up over the non-CORDIC baseline implementation and provide better arithmetic accuracy and power savings with low memory overhead when performing LDPC decoding using the BP algorithm. Memory access optimizations based on the LDPC parity-check matrix patterns can further improve the observed speed-ups. The CORDIC algorithms can also be used to efficiently implement vector rotations, 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 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

[1] "International Telecommunications Union's (ITU) recommendation for the next generation (4G – Fourth Generation) wireless protocols," ITU-R M.1645, URL: http://www.ieee802.org/secmail/pdf00204.pdf

[2] R.G. Gallager, "Low Density Parity-Check Codes," MIT Press, Cambridge, Mass., 1963.

[3] M. Tanner, "A recursive approach to low complexity codes," in IEEE Transactions on Inform. Theory, vol. IT-27, pp. 533-547, 1981.

[4] F. Guilloud, E. Boutillon, and J. Danger, "λ-Min Decoding Algorithm of Regular and Irregular LDPC Codes," in Proc. 3rd International Symposium on Turbo Codes & Related Topics, Brest, France, Sept. 2003, pp. 451-454.

[5] W.H.W. Tuttlebee, editor, "Software Defined Radio," John Wiley & Sons, Ltd., March 2004.

[6] J. Glossner, D. Iancu, M. Moudgill, G. Nacer, S. Jinturkar, S. Stanley, and M. Schulte, "The Sandbridge SB3011 Platform," in the *EURASIP Journal on Embedded System*s, Special Issue on Embedded Digital Signal Processing Systems, vol. 2007, Article ID 56467, 2007.

[7] M.J. Schulte, J. Glossner, S. Jinturkar, M. Moudgill, S. Mamidi, S. Vassiliadis, "A Low-Power Multithreaded Processor for Software Defined Radio," in the *Journal of VLSI Signal Processing*, vol. 43, no. 2-3, pp. 143–159, June 2006.

[8] J.E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Transactions Electronic Computers*, vol. 8, no. 3, pp. 330-334, Sept. 1959.

[9] J.S. Walther, "A Unified Algorithm for Elementary Functions," *AFIPS Spring Joint Computer Conference*, vol. 38, pp. 379-385, 1971.

[10] R.J. Andraka, "A Survey of CORDIC Algorithms for FPGAs," *Proceedings of the 1998 ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, pp. 191-200, Feb. 1998.

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

[12] J.R. Cavallaro and A. Elster, "Complex Matrix Factorizations with CORDIC Arithmetic," *Technical Report 89-1071, Department of Computer Science, Cornell University*, Ithaca, N.Y., Dec. 1989.

[13] Y.H. Hu, "The Quantization Effects of the CORDIC Algorithm," *IEEE Transactions on Signal Processing*, vol. 40, pp. 834-844, July 1992.

[14] D.H. Daggett, "Decimal-binary Conversions in CORDIC," *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, pp. 335-339, Sept. 1959.

[15] M. Sima, M. Senthilvelan, D. Iancu, J. Glossner, M. Moudgill, M.J. Schulte, "Software Solutions for Converting a MIMO-OFDM Channel into Multiple SISO-OFDM Channels," *Third IEEE International Conference on Wireless and Mobile Computing, Networking and Communications, WiMOB 2007*, Oct. 2007.

[16] M. Senthilvelan, M. Sima, D. Iancu, J. Glossner, M. Moudgill, M. Schulte "Instruction Set Extensions for Matrix Decompositions on Software Defined Radio Architectures," *submitted to Journal of Signal Processing Systems*, Springer, Sept. 2009.

[17] M. Senthilvelan, "CORDIC Instructions for Software Defined Radio," *PhD dissertation,* University of Wisconsin-Madison, Aug. 2010.

[18] C.E. Shannon, "A mathematical theory of communication," in Bell System Technical Journal, vol. 27, pp. 379-423 / pp. 623-656, July 1948. URL: http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf