CORDIC-Augmented Sandbridge Processor for Channel Equalization

Mihai Sima¹, John Glossner^{2,3}, Daniel Iancu², Hua Ye²,
Andrei Iancu^{4,2}, and Joe Hoane²

¹ University of Victoria, Department of Electrical and Computer Engineering,
P.O. Box 3055 Stn CSC, Victoria, B.C. V8W 3P6, CANADA
msima@ece.uvic.ca

² Sandbridge Technologies, Inc., 1 North Lexington Avenue, White Plains, NY 10601, U.S.A.
{JGlossner, DIancu, HuaYe, AIancu, JHoane}@sandbridgetech.com

³ Delft University of Technology, Department of E.E.M.C.S., Delft, The Netherlands

⁴ Rochester Institute of Technology, Computer Science Department, Rochester, NY

Abstract. In this paper we analyze an architectural extension for a Sandbridge processor which encompasses a CORDIC functional unit and the associated instructions. Specifically, the first instruction is `CFG_CORDIC` that configure the CORDIC unit in one of the rotation and vectoring modes for circular, linear, and hyperbolic coordinate systems. The second instruction is `RUN_CORDIC` that launches CORDIC operations into execution. As case study, we consider channel estimation and correction of the Orthogonal Frequency Division Multiplexing (OFDM) demodulation. In particular, we propose a scheme to implement OFDM channel correction within the extended instruction set. Preliminary results indicate a performance improvement over the base instruction set architecture of more than 80% for doing channel correction, which translates to an improvement of 50% for the entire channel estimation and correction task.

1 Introduction

A common trade-off in the design of computing engines involves the balance between efficiency and flexibility. Although Application-Specific Integrated Circuits (ASIC's) are highly efficient, they are often not flexible enough to support the variations of today's rapidly evolving standards. On the other hand, DSP processors, although fully programmable, may not achieve the high performance required for future generations of wireless systems. An architectural solution, Application-Specific Instruction set Processors, combines the efficiency of ASIC's and flexibility of DSP's. Typically, ASIP's are heterogenous platforms composed of programmable processor cores and customized hardware modules. Considering the Sandbridge processor [1–3] and CORDIC algorithm [4, 5], two general questions may be raised:

- What is the influence of a CORDIC functional unit on the performance of a Sandbridge processor?
- What are the architectural changes needed for incorporating a CORDIC unit into a Sandbridge processor core?

In order to evaluate the potential performance of the CORDIC-augmented Sandbridge processor, we address as an example channel equalization, which is one of the most challenging baseband wireless algorithms for an Orthogonal Frequency Division Multiplexing (OFDM) demodulation task. In particular, we present the implementation strategy of the channel correction on an ASIP comprising a Sandbridge processor and a CORDIC unit.

The extension of the Sandbridge Instruction Set Architecture (ISA) encompasses a CORDIC functional unit and two associated instructions: CFG_CORDIC that configures the CORDIC unit, and RUN_CORDIC that launches CORDIC operations into execution. With these instructions, a large number of transcendental functions can be computed in pipelined and SIMD fashion, which translates into a significant reduction in cycle count. In particular, modulus, division, sine, cosine, and arctangent will benefit from CORDIC support when doing channel equalization.

The paper is organized as follows. For background purposes, we outline the CORDIC algorithm in Section 2 and OFDM channel equalization in Section 3. Section 4 describes briefly the architecture of the Sandbridge processor. The architectural extension including CORDIC instructions is introduced in Section 5. The execution scenario of channel correction within the extended instruction set is discussed in Section 6, while experimental results are presented in Section 7. Section 8 completes the paper with some conclusions and closing remarks.

2 Coordinate Rotation Digital Computer

A Givens transformation [6] is a 2-by-2 orthogonal matrix $R(\theta)$ of the form described in Equation (1). It can be observed that multiplication by $R(\theta)$ of a vector $[x, y]^T$ amounts to a counterclockwise rotation of θ radians in plane.

$$R(\theta) \cdot \begin{bmatrix} x \\ y \end{bmatrix} \equiv \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad (1)$$

Historically, the Givens transformation has been used in QR factorization [7], since it can zero matrix elements selectively. Clearly, by setting

$$\cos \theta = \frac{x}{\sqrt{x^2 + y^2}}, \quad \sin \theta = \frac{y}{\sqrt{x^2 + y^2}} \quad (2)$$

it is possible to force the second entry in the vector $[x, y]^T$ to zero:

$$\begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \sqrt{x^2 + y^2} \\ 0 \end{bmatrix} \quad (3)$$

The Givens transformation is computationally demanding. For example, given an arbitrary angle θ , the direct evaluation of the rotation (Equation (1)) requires four multiplications, two additions, and a large memory storing the cosine and sine tables. Also, finding the angle θ which satisfies the trigonometric Equations (2), translates to a sequence of multiplications, additions, and memory look-up operations if the common Taylor series expansion is employed.

COordinate Rotation Digital Computer (CORDIC) is an iterative method performing vector rotations by arbitrary angles using only shifts and additions. The main idea is to first split the rotation angle θ into a sequence of subrotations of angles $\theta(n)$, where the rotation for iteration n is

$$\begin{bmatrix} x(n+1) \\ y(n+1) \end{bmatrix} = \begin{bmatrix} \cos \theta(n) & \sin \theta(n) \\ -\sin \theta(n) & \cos \theta(n) \end{bmatrix} \cdot \begin{bmatrix} x(n) \\ y(n) \end{bmatrix} \quad (4)$$

Then, the rotation matrix $R(\theta(n))$ is written as

$$R(\theta(n)) = \cos \theta(n) \cdot \begin{bmatrix} 1 & \tan \theta(n) \\ -\tan \theta(n) & 1 \end{bmatrix} \quad (5)$$

and the rotation angles are restricted so that $\tan \theta(n) = \pm 2^{-n}$. This way, the multiplication by the tangent factor is reduced to simple shift operations.

Arbitrary rotation angles can be obtained by performing a series of successively smaller elementary rotations. If the decision at each iteration, n , is which direction to rotate rather than whether or not to rotate, then the factor $\cos \theta[n]$ becomes a constant for the current iteration (since $\cos \theta[n] = \cos(-\theta[n])$). Then, the product of all these cosine values is also a constant and can be applied anywhere in the system or treated as system processing gain.

The angle of a composite rotation is uniquely defined by the sequence of the directions of the elementary rotations. That sequence can be represented by a decision vector. The set of all possible decision vectors is an angular measurement system based on binary arctangents. Conversions between this angular system and any other can be accomplished using an additional adder-subtractor that accumulates the elementary rotation angles at each iteration. The elementary angles are supplied by a small look-up table (one entry per iteration), or are hardwired, depending on the implementation. The angle accumulator adds a third difference equation to the CORDIC algorithm.

$$z(n+1) = z(n) - d(n) \arctan(2^{-n}) \quad (6)$$

The CORDIC rotator is operated in one of two modes: rotation or vectoring [4].

- In **rotation mode**, the angle accumulator is initialized with the desired rotation angle. The rotation decision at each iteration is made to diminish the magnitude of the residual angle in the angle accumulator.
- In **vectoring mode**, the CORDIC unit rotates the input vector through whatever angle is necessary to align the result vector with the x axis. The result of the vectoring operation is a rotation angle and the scaled magnitude of the original vector (the x component of the result).

Using CORDIC, a large number of transcendental functions, e.g., polar to cartesian or cartesian to polar transformations, can be calculated with the latency of a serial multiplication. By providing an additional parameter, the basic CORDIC method can be generalized to perform rotations in a linear or hyperbolic coordinate system [5], thus providing a more powerful tool for function evaluation. Of particular importance for this paper is CORDIC operating in vectoring mode in the linear coordinate system, since it provides a method for evaluating ratios.

3 OFDM Channel Equalization

Due to its robustness to multi-path propagation conditions and support for high data rates, coded Orthogonal Frequency Division Multiplexing (OFDM) has become one of the most popular modulation techniques for indoor and outdoor broadband wireless data transmission [8]. OFDM has been adopted in many wireless worldwide standards such as wireless LAN 802.11a/g, HIPERLAN/2, Digital Audio Broadcasting (DAB), Digital Video Broadcasting Terrestrial (DVB-T), Digital Video Broadcasting for Handheld (DVB-H), WirelessMAN 802.16, and Broadband Wireless Access.

Consider DVB-T/H [9]: the transmitted signal is organized in frames, and each frame consists of 68 OFDM symbols. Each symbol contains both data and reference information, and is constituted by a set of 6817 carriers in the 8K mode and 1705 carriers in the 2K mode. Due to phase and amplitude variations in the channel transfer function, received complex data symbols appear not only rotated in the complex domain, but also attenuated or enhanced. Under these conditions the amplitude and phase of each carrier is distorted. If the receiver is to coherently demodulate the signal, it needs to equalise the phase and amplitude of each carrier. This process, which is known as Channel Equalisation, is comprised of an estimation phase and a correction phase.

Channel estimators usually employ pilot information as a point of reference. A fading channel requires constant tracking, so pilot information has to be transmitted continuously. An efficient way of allowing a continuously updated channel estimate is to transmit pilot symbols instead of data at certain locations of the OFDM time-frequency lattice. This technique is referred to as Pilot-Assisted Transmission (PAT).

Most OFDM receivers, such as DVB-T/H [9], are PAT systems. The channel estimation and correction is performed for the current OFDM symbol using a set of pilot carriers [10–12]. In the following, we describe at a high level the channel estimation we used for our DVB-T/H implementation. For each OFDM symbol, the scattered pilots are spaced 12 carriers apart. The first step is the estimate of the Channel Transfer Function (CTF) samples on the scattered pilot positions for the current OFDM symbol. The second step in channel estimation is to perform interpolation in time domain for three virtual pilot groups [8]. The third step in channel estimation is to perform interpolation in frequency domain. This is done to estimate the remaining two CTF samples in between each virtual pilot pairs for the current symbol. Once CTF samples for the carriers are estimated, channel correction can be readily performed to get the corrected received carriers that are ready to be fed to further processing such as QAM demapping and TPS decoding [9].

The correction algorithm is a so called *derotation*. Assuming the channel estimation yields an error vector $e_I + je_Q$ for a particular carrier, the corrected vector for that particular carrier is obtained by the complex division

$$c_I + jc_Q = \frac{r_I + jr_Q}{e_I + je_Q} = \frac{(r_I + jr_Q)(e_I - je_Q)}{e_I^2 + e_Q^2} \quad (7)$$

where c_I and c_Q are the corrected values for the real and imaginary part of a particular carrier, and r_I and r_Q are the real and imaginary parts of the received carrier. Each carrier must go through the computationally-intensive derotation process described in Equation 7.

4 Overview of the Sandbridge processor

In this section we describe the most important issues of the Sandbridge architecture and microarchitecture. In particular, our emphasis will be on the multi-threading capability and SIMD-style Vector Unit.

4.1 Sandbridge processor

Sandbridge Technologies has designed a multithreaded processor capable of executing DSP, embedded control, and Java code in a single compound instruction set optimized for handset radio applications [1–3]. The Sandbridge Sandblaster design overcomes the deficiencies of previous approaches by providing substantial parallelism and throughput for high-performance DSP applications, while maintaining fast interrupt response, high-level language programmability, and low power dissipation.

The Sandbridge processor [1–3] is partitioned into three units; an instruction fetch and branch unit, an integer and load/store unit, and a SIMD-style vector unit. The design utilizes a unique combination of techniques including hardware support for multiple threads, SIMD vector processing, and instruction set support for Java code. Program memory is conserved through the use of powerful compounded instructions that may issue multiple operations per cycle. The resulting combination provides for efficient execution of DSP, control, and Java code. The instructions to speed up CORDIC operations are executed in the Sandbridge Vector Unit described in Subsection 4.4.

4.2 Sandbridge pipeline

The pipelines are different for various operations as shown in Figure 1. The Load/Store (Ld/St) pipeline has nine stages. The integer and load/store unit has two execute stages for Arithmetic and Logic Unit (ALU) instructions and three execute stages for integer multiplication (LMUL) instructions. A *Wait* stage for the ALU and LMUL instructions causes these instructions to read from the general-purpose register file one cycle later than Ld/St instructions. This helps reduce the number of register file read ports. The vector multiplication (V_MUL) has four execute stages – two for multiplication and two for addition. It should be noted that once an instruction from a particular thread enters the pipeline, it runs to completion. It is also guaranteed to write back its result before the next instruction from the same thread reads the result.

4.3 Sandbridge multithreading

The Sandblaster architecture supports multiple concurrent program execution by the use of hardware thread units. Multiple copies (e.g., banks and/or modules) of memory are available for each thread to access. The Sandblaster processor uses the Token Triggered Threading (T^3) form of interleaved multithreading [2], in which each thread is allowed to simultaneously execute an instruction, but only one thread may issue an instruction on a cycle boundary. The microarchitecture currently supports up to eight concurrent hardware threads. Multi-threading effectively hides true dependencies which typically occur in connection with long-latency operations.

Ld/St	Inst Dec	RF Read	AGEN	XFer	Int Ext	Mem 0	Mem 1	Mem 2	WB
ALU	Inst Dec	Wait	RF Read	Exec 1	Exec 2	XFer	WB		
L_Mul	Inst Dec	Wait	RF Read	Exec 1	Exec 2	Exec 3	XFer	WB	
V_Mul	Inst Dec	VRF Read	Mpy1	Mpy2	Add1	Add2	XFer	VRF WB	

Fig. 1. Sandbridge pipeline.

4.4 The vector processing unit

The Vector Processing Unit (VPU) consists mainly of four Vector Processing Elements (VPEs), which 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 PE to load or store 16 bits of data each cycle in SIMD fashion. Support for SIMD execution significantly reduces code size, as well as power consumption, since multiple sets of data elements are processed with a single instruction [13].

Most SIMD vector instructions go through eight pipeline stages. For example, a vector MAC (V_MAC) instruction goes through the following stages: Instruction Decode, Vector Register File (VRF) Read, Mpy1, Mpy2, Add1, Add2, Transfer, and Write Back. The Transfer stage is needed due to the long wiring delay between the bottom of the VPU and the VRF. Since there are eight cycles between when consecutive instructions issue from the same thread, results from one instruction in a thread are guaranteed to have written their results back to the VRF by the time the next instruction in the same thread is ready to read them. Thus, the long pipeline latency of the VPEs is effectively hidden, and no data dependency checking or bypass hardware is needed. This is illustrated in Figure 2, where two consecutive vector multiply instructions issue from the same thread. Even if there is a data dependency between the two instructions, there is no need to stall the second instruction, since the first instruction has completed the Write Back stage before the second instruction enters the VRF Read stage.

Fig. 2. Two consecutive Vector Multiply instructions that issue from the same thread.

5 An architectural extension for Sandbridge processor

The instructions investigated are `CFG_CORDIC` that configures the CORDIC unit in one of the execution modes (rotation, vectoring) and one of the coordinate systems (circular, linear, hyperbolic), and `RUN_CORDIC` which launches the configured CORDIC operation. Assuming that 16-bit precision is needed (it is usually the case in OFDM demodulation), then the CORDIC algorithm reads in two 16-bit arguments and produces two 16-bit results. If not all the CORDIC iterations can be performed by a single `RUN_CORDIC` call, then the angle and iteration number must be saved between successive `RUN_CORDIC` calls.

The proposed `RUN_CORDIC` instruction is a vector instruction that goes through eight pipeline stages; that is, the execution itself has a latency of 4 thread cycles. The CORDIC functional unit can perform 2 CORDIC iterations in a thread cycle (two additions and two shifts), and is shared by four SIMD units. Consequently, `RUN_CORDIC` will execute 8 times, i.e., it will take up 8 instruction cycles, for a 16-bit precision, and will perform 4 conversions in SIMD style.

This is the result where the CORDIC unit is added to the vector unit by adding one adder, one shifter, a comparator and some control logic to the existing pipeline. Deploying an autonomous CORDIC unit for each SIMD unit of each thread will translate into both a memory bandwidth problem and a hardware problem, i.e., too much added hardware (32 adders, 32 shifters, 32 comparators), and the operands cannot be fetched from, or the results cannot be stored back to memory anyway.

The CORDIC instructions are defined as follows.

- `CFG_CORDIC`
for(i=0; i<4; i++) {
 - Read 8 bits of configuration data
 - Configure the CORDIC unit:
 - * Mode (1 bit): rotation or vectoring
 - * Coordinate system (2 bits): circular, linear, or hyperbolic
 - * Iteration Identifier (5 bits): ranges from 0 to 31}

- `RUN_CORDIC`
for(i=0; i<4; i++) {
 - Reads in the first 32-bit vector register packing:
 - * 16-bit modulus and 16-bit angle for rotation mode
 - * 16-bit x-value and 16-bit y-value for vectoring mode
 - Reads in the second 32-bit vector register storing:
 - * 16-bit angle for vectoring mode
 - Performs two CORDIC iterations (two additions and shiftings)
 - Writes back one 32-bit vector register packing:
 - * 16-bit x-value and 16-bit y-value for rotation mode
 - * 16-bit modulus and 16-bit angle for vectoring mode}

6 Channel correction execution scenario

As mentioned in Section 6, the channel correction involves essentially a complex division. The strategy we implemented for channel correction is to express complex numbers in trigonometric form and to use the CORDIC algorithm to perform the computation. Briefly, this strategy can be summarized as follows:

1. Express the numerator of Equation 7 in trigonometric form by using CORDIC (vectoring mode, circular rotations).
2. Express also the denominator of Equation 7 in trigonometric form by using CORDIC (vectoring mode, circular rotations).
3. Perform the division using CORDIC (vectoring mode, linear rotations).
4. Express the result back in algebraic form also using CORDIC (rotation mode, circular rotations)

Assume 16-bit precision: it requires 16 vector instructions to complete a CORDIC rotation. Therefore, 64 vector instructions are used to perform the derotation. The computing performance according to this scenario has been evaluated for a pure software solution and also when CORDIC operation benefits from customized instruction set. The experimental results are presented in the next section.

7 Experimental results

The Sandbridge integer ALU (non-vectorized) takes $9 \times 16 = 144$ instructions to do a rotate. This implies $144 \times 4 = 576$ instructions are needed to do a derotate. The Vectorized loop takes $8 \times 16 = 128$ instructions to do 4 rotates, which implies $128 \times 4 = 512$ instructions for 4 derotates. The ALU takes 1 instruction longer because the ALU has no MAC instruction and must use conditional jump. If the ALU had a CORDIC instruction, it would take 0.5×16 instructions to do 1 rotate. If the Vector Unit had a CORDIC instruction, it would take 0.5×16 instructions to do 4 rotates. The figures are presented in Table 1.

Implementation style	1 thread (instructions)	8 threads (cycles)
ALU using emulated division	96	96
ALU using emulated CORDIC	$4 \times 9 \times 16 = 576$	576
ALU using hardware CORDIC	$4 \times 16 = 64$	64
Vector using emulated CORDIC	$4 \times (8 \times 16)/4 = 128$	128
Vector using hardware CORDIC	$(4 \times 16)/4 = 16$	16

Table 1. Derotation figures per carrier.

Although CORDIC is essentially a sequential algorithm (it can compute a number of functions in a serial way, one bit per iteration), it has the very important property of

being vectorizable and pipelineable. This explains the very good performance provided by the 4-way CORDIC unit when doing derotation (16 cycles per carrier) over the non-vectorized ALU solution (96 cycles per carrier).

Experiments which have been carried out on a cycle-accurate simulator provide for the following numerical figures. Channel equalization total cycle count per OFDM symbol is 319634 cycles, out of which the complex division (that is, the derotation) in Equation 7 counts for 209388 cycles. Given the fact that the CORDIC-based solution provides for a cycle count reduction of $(96 - 16) \times 100/96 = 83\%$, the global improvement for channel estimation is $0.83 \times 209388 \times 100/319634 = 54\%$. Given the fact that Sandbridge is a multi-threaded DSP-oriented processor, such an improvement within wireless processing domain indicates that extending the Sandbridge instruction set with CORDIC instructions is a promising approach.

8 Conclusions

We have proposed an architectural extension for the Sandbridge processor which encompasses a CORDIC functional unit and the associated instructions: CFG_CORDIC and RUN_CORDIC. Configuring the CORDIC unit in one of the two modes and three coordinate systems is performed under the command of the CFG_CORDIC, while the RUN_CORDIC instruction launches into execution CORDIC operations. Preliminary results indicate a performance improvement over the base instruction set architecture of more than 80% for doing channel correction, which translates to an improvement of more than 50% for the entire channel estimation and correction task. As future work, we intend to address the entire DVB-T processing chain and to evaluate the overall system improvement from the CORDIC-augmented Sandbridge processor.

References

1. Glossner, J.C., Hokenek, E., Moudgill, M.: Multithreaded Processor for Software Defined Radio. In: Proceedings of the 2002 Software Defined Radio Technical Conference. Volume I., San Diego, California (2002) 195–199
2. Schulte, M.J., Glossner, J.C., Mamidi, S., Moudgill, M., Vassiliadis, S.: A Low-Power Multithreaded Processor for Baseband Communication Systems. In Pimentel, A.D., Vassiliadis, S., eds.: Proceedings of the Third and Fourth International Annual Workshops on Systems, Architectures, MOdeling, and Simulation (SAMOS). Volume 3133 of Lecture Notes in Computer Science., Samos, Greece, Springer (2004) 393–402
3. Glossner, J.C., Schulte, M.J., Moudgill, M., Iancu, D., Jinturkar, S., Raja, T., Nacer, G., Vassiliadis, S.: Sandblaster Low-Power Multithreaded SDR Baseband Processor. In: Proceedings of the 3rd Workshop on Applications Specific Processors (WASP'04), Stockholm, Sweden (2004) 53–58
4. Volder, J.E.: The CORDIC trigonometric computing technique. IRE Transactions on Electronic Computers **EC-8** (1959) 330–334
5. Walther, J.: A unified algorithm for elementary functions. In: Proceedings of the Spring Joint Computer Conference of the American Federation of Information Processing Societies (AFIPS). Volume 38., Arlington, Virginia, AFIPS Press (1971) 379–385

6. Golub, G.H., van Loan, C.F.: Matrix Computations. 3rd edn. The Johns Hopkins University Press, 2715 North Charles Street, Baltimore, Maryland 21218-4363 (1996) ISBN: 0-8018-5414-8.
7. Strang, G.: Introduction to Linear Algebra. 3rd edn. Wellesley-Cambridge Press, Box 812060, Wellesley, MA 02482 (2003) ISBN: 0-9614088-9-8.
8. van Nee, R.D., Prasad, R., eds.: OFDM for Wireless Multimedia Communications. Artech House Publishers, Norwood, MA 02062, U.S.A. (2000)
9. European Telecommunications Standards Institute: (Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television)
10. Speth, M., Fechtel, S., Fock, G., Meyr, H.: Optimum Receiver Design for OFDM-Based Broadband Transmission – Part II: A Case Study. IEEE Transactions on Communications **49** (2001) 571–578
11. Frescura, F., Pielmeier, S., Reali, G., Baruffa, G., Cacopardi, S.: DSP-Based OFDM Demodulator and Equalizer for Professional DVB-T Receivers. IEEE Transactions on Broadcasting **45** (1998) 323–332
12. Tong, L., Sadler, B.M., Dong, M.: Pilot-Assisted Wireless Transmissions: General Model, Design Criteria, and Signal Processing. IEEE Signal Processing Magazine **21** (2004) 12–25
13. Sebot, J., Drach, N.: SIMD ISA Extensions: Reducing Power Consumption on a Superscalar Processor for Multimedia Applications. In: IEEE Symposium on Low-Power and High-Speed Chips (Cool Chips) IV, Tokyo, Japan (2001)