Application of Reconfigurable CORDIC Architectures

Oskar Mencer*, Luc Séméria*, Martin Morf*, Jean-Marc Delosme**

Abstract

Reconfiguration enables the adaption of Coordinate Rotation DIgital Computer (CORDIC) units to the specific needs of sets of applications, hence creating application specific CORDIC-style implementations.

Reconfiguration can be implemented at a high level, taking the entire CORDIC unit as a basic cell (CORDIC-cells) implemented in VLSI, or at a low level such as Field-Programmable Gate Arrays(FPGAs). We suggest a design methodology and analyze area/time results for coarse(VLSI) and fine-grain(FPGA) reconfigurable CORDIC units. For FPGAs we implement CORDIC units in Verilog HDL and our object-oriented design environment, PAM-Blox. For CORDIC-cells, multiple reconfigurable CORDIC modules are synthesized with state-of-the-art CAD tools.

At the algorithm level we present a case study combining multiple CORDICs based on a geometrical interpretation of a normalized ladder algorithm for adaptive filtering to reduce latency and area of a fully pipelined CORDIC implementation. Ultimately, the goal is to create automatic tools to map applications directly to reconfigurable high-level arithmetic units such as CORDICs.

I. Introduction

Reconfigurable computing spans the space between programmable microprocessors and static Application Specific Integrated Circuits (ASICs). Reconfigurable architectures offer the flexibility of ASIC design and the programmability of microprocessors. The overhead of reconfigurability depends on the complexity of the reconfigurable cell. Bit-level cells in Field-Programmable Gate Arrays(FPGAs) offer high flexibility with a high overhead in latency and area. ASICs with programmable arithmetic units, "chunky architectures"[1], in our case Coordinate Rotation DIgital Computer (CORDIC)-cells, have a lower overhead with much less flexibility. In this research we are not going to end the debate about which level of reconfigurability is best. Instead, we show how CORDIC arithmetic units can be implemented on FPGAs and on ASICs. Given multiple CORDIC arithmetic units, we show a case study on how applications could be mapped onto a set of CORDIC arithmetic units by using a geometric interpretation of computation.

CORDIC arithmetic units use shift-and-add primitives to compute fixed-point elementary functions on relatively small silicon area. For a more detailed introduction to CORDIC algorithms see [2]. For advanced CORDIC techniques see for example [3][4]. CORDIC units are known to be highly pipelineable, very small, with linear convergence towards the correct result. Linear convergence means that we can guarantee at least one bit of precision per shift-add iteration. The internal structure of CORDICs, consisting of adders and wired shifts in the case of parallel CORDICs, makes them well suited for FPGA implementation[5].

CORDIC functional units compute up to two elementary functions at the same time. Given three arguments x, y, z, basic CORDIC arithmetic units compute function pairs such as shown in Table I.

The fundamental principles behind the CORDIC algorithms of Volder[6] and Walther[7] can be found in their scalar form in the work of Chen[8]. Ahmed showed in [4] how scale factor compensation can be avoided by choosing an appropriate shift-sequence to automatically compensate for the scale factor. A refinement of this idea, in order to minimize overhead, was presented in [9]. All CORDIC implementations in this paper are therefore correctly scaled with minimal overhead.

Ahmed also showed in [4] that if Chen's convergence computation technique is applied to complex numbers instead of real numbers (as assumed by Chen) one obtains the class of CORDIC algorithms. The method of formally "replacing" real by complex numbers was extended in [10],[3] to obtain CORDIC algorithms for quaternions and pseudo-quaternions. When the CORDIC functions, especially the higher order functions, are matched to applications–a system design issue–the real power of CORDICs and related algorithms can be exploited.

One alternative to CORDICs are multiplication based algorithms. The major drawback of fast multipliers is their large size and irregularity of wiring for a logarithmic reduction of terms[11]. CORDICs compute two elementary functions on approximately the area and latency of 1-2 multipliers.

Reconfiguration of CORDICs enables effective hardware support of such complex functions, similar to micro-code or firmware (library functions). It becomes possible to hide the complexity involved from a typical application-level programmer. Custom design of CORDIC units for individual applications is a complex task, requiring both specialized low-level design tools and symbolic computing tools that support a domain expert. Sophisticated tools that can support

*Computer Systems Laboratory, Department of Electrical Engineering, Stanford, CA 94305, USA, email: {oskar,semeria,morf}@stanford.edu, http://umunhum.stanford.edu/PAM-Blox/

**Département Informatique, Université d'Evry, Cours Monseigneur Romero, 91025 Evry, France. email: delosme@lami.univ-evry.fr

a typical programmer will eventually become available. In the mean-time domain experts will have to use today's tools to create winning designs using these ideas in advanced applications.

As a first step towards an automatic CORDIC compiler for FPGAs, we introduce the hardware object as an intermediate level of abstraction[12]. We define a set of CORDIC module generators that could be targeted by a compiler similar to the instruction set of a microprocessor. Most of todays efforts at direct compilation from a high-level language to FPGAs target very simple arithmetic units such as adders, multipliers, shifters, etc. Generally, by targeting such simple modules, most of the power of reconfigurable computing is lost. Instead, more complex arithmetic units such as CORDICs coupled with various alternatives of number representations should be targeted by higher-level compilers to exploit the full potential of reconfigurable computing. We are at the beginning of the development process of special purpose compilers for complex arithmetic units such as CORDICs. One objective of this paper is to show a possible direction for high-level compilation to CORDICs.

Section 2 describes the methodology of this research. Section 3 presents the results at the module level for a "chunky" CORDIC architecture, and CORDICs on FPGAs. Section 4 presents a case study at the algorithm level: mapping an adaptive ladder filter to *fixed-point* CORDIC arithmetic units.

II. Methodology

Ultimately, the goal is to create automatic algorithms to map applications directly to reconfigurable high-level arithmetic units. As a starting point we split the problem into 2 parts:

- **module-generation level (Section III):** on this level we create building blocks based on CORDIC arithmetic units. Module generation implies that we can create application-specific CORDIC units given parameters such as data bit-width, precision, or number of stages. We consider two options at the module level:
  1. fine-grain reconfiguration: CORDICs on FPGAs
  2. coarse-grain reconfiguration: VLSI CORDIC-cells
- **algorithm level (Section IV):** we use the CORDIC modules generated at the lower level, and combine them to compute entire applications.

Module-generation is well understood, based on past research on CORDIC arithmetic units. We implement CORDICs for FPGAs with our module-generation environment, PAM-Blox, described in the next section, and Synopsys FPGA Express. Coarse-grain CORDIC cells are synthesized with Synopsys Behavioral Compiler.

The algorithm level poses similar challenges as compilation to a very complex instruction set. The process of combining complex building blocks to optimally compute a specific algorithm is still not very well understood. We propose a geometrical interpretation of computation to create a unified approach for combining multiple CORDICs given a specific algorithm.

III. Module Level

At the module level, our task is to map a CORDIC architecture to gates or look-up tables. In this section, we look at two approaches. The first approach is a fine-grain reconfiguration on FPGAs. The CORDIC modules are implemented using the PAM-Blox environment or commercial synthesis tools for FPGAs. The second approach is a coarse-grain reconfiguration in which the CORDIC module itself represents the basic reconfigurable cell implemented on an ASIC. Synthesis tools map CORDICs efficiently to hardware. By changing the constraints on the latency of the design, different implementations of the cell can be explored.

We implement fully parallel CORDIC modules. Figure 1 shows the parallel architecture of a generic CORDIC unit.

A. Fine-Grain Reconfiguration: FPGAs

We compare the implementation of CORDICs with PAM-Blox and a state-of-the-art synthesis tool for FPGAs.

A.1 PAM-Blox: Object-Oriented Module Generation

Traditional hardware synthesis is based on a top-down approach; starting from a high-level description, CAD tools synthesize and optimize the hardware level by level, until the final layout. Initial FPGA synthesis tools have taken the same approach, adding a last step of technology mapping at the end of the CAD hierarchy.

We propose a bottom-up approach to the design of synthesis tools/compilers for FPGAs. The main reason behind building FPGA circuits bottom up, is that the architecture and interconnect is limited to the resources on the FPGA, making the traditional top-down approach less optimal.

By creating a parameterizable repository of module generators, PAM-Blox[12], we add a level of abstraction that preserves optimal area and performance while simplifying the design process (compared to state-of-the-art high-performance FPGA tools). In terms of reconfigurable computing, these modules constitute the instruction set that could be targeted by the compiler.

Figure 2 shows an overview of the PAM-Blox system. We use PAM-Blox as the name for the entire design environment. PamBlox(see Figure 2) stands for templates of hardware objects while the more complex PaModules are objects with a fixed size. PAM-Blox simplifies the design of datapaths for FPGAs by implementing an object-oriented hierarchy described in C++. With PAM-Blox, hardware designers can benefit from some of the advantages of object-oriented system design that the software industry has learned to cherish during the last decade. Efficient use of function overloading, virtual functions, and templates makes PAM-Blox a competitive and yet simple to use design environment.

A major question is which modules are required. This question is actually similar to defining a hardware-software interface for hardware-software co-design. We believe that by providing higher-level modules such as specialized multipliers (see [13] for IDEA encryption), state-machines (see [14] for boolean satisfiability), arithmetic units for advanced number representations, etc., we can explore the benefits of reconfigurable arithmetic. In this study, we implement CORDICs and study how higher-level compilers might target such modules.

Currently, the PAM-Blox CORDICs are implemented as PaModules with a fixed bitwidth. A floorplan for a parallel CORDIC is shown in Figure 3. The 8-bit parallel CORDIC requires 131 CLBs while a bit-serial CORDIC, with 23 bit-serial adders requires substantially more area, due to the inherent dependency structure of the CORDIC algorithm. In contrast, a CORDIC iterating with only 3 parallel ADD/SUB modules on the CORDIC equations would have very low throughput, and an area penalty for the z look-up table which is hardwired in the parallel case.

Although serial arithmetic usually takes less area, the bit-serial CORDIC occupies 30% more area than the parallel CORDIC. This counter-intuitive result is due to dependencies between the stages. A stage needs to know the sign of z of the previous stage in order to select the sign for its own computation. The resulting overhead of storing the intermediate values while waiting for the sign to compute and the increased overhead for control logic, making the bit-serial CORDIC a less desirable CORDIC solution.

The parallel CORDIC achieves a throughput of 33 million rotations per second at 33 MHz PCI clock speed. The results are summarized in Table II. With current FPGA technology the throughput would scale up easily to 100 MHz, hence 100 million rotations per second.

A.2 Synthesis for FPGA

We compare PAM-Blox module-generation to Synopsys FPGA Express synthesis. We try to optimize a CORDIC architecture for Xilinx XC4000 FPGAs using Synopsys FPGA Express[15]. The results after optimization are comparable to the PAM-Blox results found in the previous section: after place&route the area of the circuit is 133 CLBs with a clock cycle latency of 25.4 ns.

As we will see in the next section, synthesis tools can be used to effectively optimize CORDIC modules for ASICs. However, for FPGAs, the possible optimizations are restricted by the internal architecture of the CLBs – especially the fast carry-chains. The advantage of FPGA synthesis over PAM-Blox, a structural bottom-up approach, for complex arithmetic units is therefore limited. As a consequence at the module level it is preferable to use module generation to create CORDIC units for FPGAs, and a compiler to optimize the application-level structure using reconfigurable CORDICs as elementary building blocks.

B. Coarse-grain Reconfiguration: ASICs

In this section we analyze how much a CORDIC unit can be optimized by state-of-the-art VLSI synthesis.

For the implementation of a CORDIC arithmetic unit in hardware, many of the operators (adders, subtracters) can be optimized. In particular, optimization can be performed when one of the operands is a constant (calculation of the z factors) or when some input bits have the same value (calculation of the x and y factors after shifting). We apply logic and architectural optimization for a non-pipelined version of the parallel CORDIC and synthesize the design for ASIC.

In general, the behavior of circuits can be represented by abstract models such as boolean functions and finite state machines which can be derived from higher-level models. In the case of combinational logic (i.e. circuits without feed-back), the abstract model is a set of boolean functions and relations on the circuit's inputs and outputs. These functions can be simplified for a given target architecture by employing logic synthesis and optimization[16]. Very powerful optimization can be performed under both area and/or time constraints.

For arithmetic operations, further optimization can also be performed at the architectural level by looking at different architectures of operators (e.g. ripple carry adder, carry save adder, etc.), trying to increase bit-level parallelism. In the past few years, such techniques have also been integrated within commercial tools[17] and allow quick estimation of the performance of many candidate architectures.

For ASIC synthesis we use the Synopsys Design Compiler to synthesize the circuit and the Synopsys Behavioral Compiler for the arithmetic optimization[17]. The target technology is the 'tsms 0.35 micron' logic process. We study the area/latency trade-off by changing the constraints on the optimizations. Figure 4 presents the area-time curves with and without architectural and logic optimizations. We observe that after optimization the circuit is at least 20% smaller for a given latency and at least 17% faster for a given area. The smallest design (with optimization) has a total area of 57K library units and a latency of 41ns, compared to an area of 75K library units and a latency of 43ns without optimization. Minimal latency with optimization is 17.94ns for an area of 153K library units. Without optimization the latency is 23.45ns for the same area as the optimized design.

To increase the throughput for a given latency, we pipeline the implementation by inserting registers into the datapath. This can be done automatically by the synthesis tool. For any given clock frequency, an area/latency trade-off similar to Figure 4 can also be identified with pipelined modules.

IV. Algorithm Level: Combining multiple CORDICs

Combining multiple CORDICs to an entire application is currently more an art than a science. In order to illustrate some of the reasoning and manipulations involved when deriving CORDIC-style implementations for specific applications, we revisit an algorithm of Lee and Morf, summed up in [10] and detailed in Section 7 of the survey [18].

A. Adaptive ladder filter

The are many ways of developing adaptive ladder filters. A very typical case is represented by the recursive exact least-squares filters, such as discussed in [19], [20]. One of the most efficient implementations is based on adaptive ladder or lattice filters. The adaptive ladder filter is an FIR filter used for the prediction of stochastic processes, e.g. for channel equalization or speech encoding. The following development is the scalar version corresponding to a single channel filter. The corresponding multi-channel version could be derived from [21]. The filter is composed of n cascaded feed-forward stages, n being the order of the filter. Each stage has two outputs, the so-called forward and backward innovation, which are sent on to the next stage (the backward innovation being delayed by one sample period before being used). Each stage is parameterized by a "gain", the partial correlation between the forward and backward innovation. This gain varies with time and is updated whenever new values of the innovations are computed, i.e. each time there is a new sample; this is the "adaptive" part of the filter. Within each stage a time update consists of 3 equations (the stage and time indices are not shown here)

$$\left\{\begin{array}{lcl} \rho_+ & = & \rho\bar{\nu}\bar{\eta} + \nu\eta \\ \nu_+ & = & (\nu - \rho_+\eta)/(\bar{\rho}_+\bar{\eta}) \\ \eta_+ & = & (\eta - \rho_+\nu)/(\bar{\rho}_+\bar{\nu}) \end{array}\right. \tag{1}$$

where ρ denotes the normalized partial correlations, ν and η denote the normalized forward and backward innovations respectively, ρ_+, ν_+ and η_+ are the updated variables, and $\bar{x} = \sqrt{1-x^2}$, the complement of x.

Usually adaptive filter implementations with Given's rotations require floating point arithmetic. The selected filter algorithm (equations above) has a built-in variance and magnitude-normalization property that allows us to use fixed-point arithmetic, which is more suitable for FPGAs. For more details on VLSI implementations and associated block diagrams of these equations see [19],[20].

B. Geometrical interpretation of the ladder filter

The relations (1) are normalized versions of "Schur complement" identities relating the covariances of random variables. Since the Schur complement identities essentially capture the theorem of Pythagoras in Euclidean space, normalization, which amounts to projecting the objects from Euclidean space onto the unit sphere, yields identities of spherical geometry.

As a result the relations (1) have an elegant interpretation in terms of spherical trigonometry. Considering the triangle *FRB* in Figure 5, and measuring both the angles R, F, B and the sides r, f, b in radians, we can write three identities from spherical trigonometry:

$$\left\{\begin{array}{lcl} \cos r & = & \cos R \cdot \sin f \cdot \sin b \; + \; \cos f \cdot \cos b \\ \cos F & = & (\cos f \; - \; \cos r \cdot \cos b)/(\sin r \cdot \sin b) \\ \cos B & = & (\cos b \; - \; \cos r \cdot \cos f)/(\sin r \cdot \sin f) \end{array}\right. \tag{2}$$

These identities enable the determination of information to the left of the dashed line (L) in Figure 5 in terms of information to the right of (L).

With the correspondence

$$\left\{\begin{array}{lll} \rho = \cos R \;, & \nu = \cos f \;, & \eta = \cos b \;, \\ \rho_+ = \cos r \;, & \nu_+ = \cos F \;, & \eta_+ = \cos B \;, \end{array}\right. \tag{3}$$

relations (1) are seen as relations providing the solution of a spherical triangle given two sides and the included angle. Such equations are found in navigation on the Earth's surface. Volder [6] developed the CORDIC procedure precisely to solve such problems digitally and showed how to link CORDIC rotations for that purpose. Following a similar vein, Lee et al. [10] proposed a way for linking the three types of CORDIC operations of Walther [7] to evaluate the expressions (1) (this way is also presented in [18], with a slight modification). Is this way optimal? Can our geometrical insight enable us to improve on it? Since the work [3] on quaternion CORDIC algorithms we know how to perform 3-D rotations in a CORDIC-like fashion by working simultaneously on all 3 components. Can this be exploited?

Geometrically, we are interested in the result of the composition of the "backward" rotation from F to R along b and the "forward" rotation from R to B along f; the cosine of the angle R between the sides b and f corresponds naturally to the normalized partial correlation. That result corresponds to the rotation from F to B along r, whose parameters are what we seek. Appendix A details this composition of 3-D rotations and actually obtains as a result a decomposition of the rotations in terms of 2-D CORDICs.

C. Implementation

Putting some flesh on the skeleton obtained in Appendix A, the computations are cast in terms of *pairs* of 2-D CORDIC operators (represented below by ◉ *and* ◎), where the operators do not require the "Z-factor" part that computes the angles explicitly. We obtain the architecture shown in figure 6:

$\boxed{\text{Step 1}}$

◉ rotate $\begin{bmatrix} \rho \\ \bar\rho \end{bmatrix}$ to force the 2nd component to 0 and thus obtain the encoding (sign sequence) of the angle R,

◎ simultaneously apply the rotation $\mathcal{R}(R) = \begin{bmatrix} \rho & -\bar\rho \\ \bar\rho & \rho \end{bmatrix}$, determined by the sign sequence for the angle R, to the vector $\begin{bmatrix} \bar\eta \\ 0 \end{bmatrix}$ to obtain $\begin{bmatrix} \rho\bar\eta \\ \bar\rho\bar\eta \end{bmatrix}$.

$\boxed{\text{Step 2}}$

◉ apply the rotation $\mathcal{R}(f) = \begin{bmatrix} \nu & -\bar\nu \\ \bar\nu & \nu \end{bmatrix}$, determined by the encoding of f (obtained in Step 5 of previous update) to $\begin{bmatrix} \bar\rho\bar\eta \\ 0 \end{bmatrix}$ to obtain $\begin{bmatrix} \nu\bar\rho\bar\eta \\ \bar\nu\bar\rho\bar\eta \end{bmatrix} = \begin{bmatrix} \nu^\dagger \\ \bar\eta \cdot \bar\rho\bar\nu \end{bmatrix}$.

◎ apply the rotation $\mathcal{R}(f)$ to the vector $\begin{bmatrix} \eta \\ -\rho\bar\eta \end{bmatrix}$ to obtain $\begin{bmatrix} \rho_+ \\ \eta^* \end{bmatrix}$.

$\boxed{\text{Step 3}}$

◉ apply $\mathcal{R}(R)$ to $\begin{bmatrix} \eta^* \\ \nu^\dagger \end{bmatrix}$ to get $\begin{bmatrix} -\nu^* \\ \eta^\dagger \end{bmatrix}$, where $\eta^\dagger = \eta \cdot \bar\rho\bar\nu$,

◎ employ the hyperbolic CORDIC mode and force to 0 the 2nd component of the vector $\begin{bmatrix} 1 \\ \rho_+ \end{bmatrix}$ to obtain $\bar\rho_+$ as 1st component.

$\boxed{\text{Step 4}}$

◉rotate $\begin{bmatrix} \eta^\dagger \\ \bar\eta \cdot \bar\rho\bar\nu \end{bmatrix}$ to force the 2nd component to 0 and thus obtain as 1st component $\bar\rho\bar\nu = \bar\rho_+\bar\nu_+$ (and, as a byproduct, a not very accurate—when $\bar\rho\bar\nu$ is small—encoding of b, that we shall not use),

◎ compute, in the linear mode, the encoding of $1/\bar\rho_+$ (non-restoring division) and, simultaneously, apply this sign sequence to $\bar\rho\bar\eta$ to get $\bar\eta_+$.

$\boxed{\text{Step 5}}$

◉ rotate $\begin{bmatrix} \nu^* \\ \bar\rho_+\bar\nu_+ \end{bmatrix}$ to force the 2nd component to 0 and thus obtain the encoding (sign sequence) of the angle F to be used as encoding of the "angle" f in Step 2 for the next update,

◎ apply in the linear mode the sign sequence encoding $1/\bar\rho_+$ both to ν^*, to get ν_+, and to η^*, to get η_+.

The second CORDIC at Step 4 is a modified version of the standard CORDIC architecture. It computes the sign sequence encoding of $1/\bar\rho_+$ and, simultaneously, applies this sign sequence to $\bar\rho\bar\nu$ to get $\bar\nu_+$. With $x_0 = 1$ and $y_0 = 0$, the recurrence is of the form (assuming $1/\bar\rho_+$ does not exceed 8, i.e., $|\rho_+|$ does not exceed 0.992):

$$\begin{cases} x_{i+1} = x_i - sign(x_i) \cdot 2^{2-i} \cdot \bar\rho_+ \\ y_{i+1} = y_i + sign(x_i) \cdot 2^{2-i} \cdot \bar\rho\bar\nu \end{cases} \tag{4}$$

From the recurrence relation, one can see that equation (4) fits on the shift-and-add resources of a CORDIC architecture.

The accuracy d needed for the computations will typically be about 16 to 20 bits. However, scaling plus additional iterations for convergence (in the hyperbolic case) impose slightly more than d pipeline stages within a CORDIC unit [4], [9].

Using the pipelined, parallel CORDIC presented before, we distinguish 4 basic architectures based on the number of CORDICs used:
1. minimal: $1-2$ CORDICs
2. based on the 5 steps of computing 1 stage of the filter: $2 \cdot 5$ CORDICs
3. based on the number of stages in the filter: $2 \cdot n$ CORDICs
4. fully pipelineable, maximal performance: $2 \cdot 5 \cdot n$ CORDICs

All cases require some amount of memory, or shift-registers (FIFOs), to store intermediate values of the computation. Xilinx CLBs can be configured to 16-bit FIFOs enabling a very efficient implementation of the intermediate shift registers. Note that we do not require all three CORDIC equations, using only the x and y pipes, we save 33% of area. Also each case has different requirements on reconfigurability on the CORDICs.

In all cases the delay for 1 result is $5 \cdot n \cdot d$ clock cycles. However, throughput differs with pipeline depth. In case 4, with $10 \cdot n$ CORDICs, throughput is 1 clock cycle between results. Case 3 with $2 \cdot n$ CORDICs results in 5 clock cycles between results. Case 2 with 10 CORDICs results in n clock cycles between results. Finally, case 1 requires $(1-2) \cdot 5 \cdot n$ clock cycles between results.

D. Discussion

In order to understand the advantages of the geometrical interpretation we compare the above implementation to an earlier implementation with CORDICs (see [10], [18]). Both implementations require 10 CORDICs. The earlier implementation employs all 3 pipelines (x, y, and z) as opposed to only (x, y) CORDICs in the proposed implementation. Thus, the geometrical interpretation gives us 17% reduction of latency with a 45% reduction of area for the fully pipelined case 4 (see above). However the earlier implementation could be modified to also use sign encodings of the z-quantities thus giving about the same latency and area. The geometrical approach has the advantage of being more systematic, less empirical, and therefore more apt to be used to create compilers for reconfigurable computing that can target CORDIC arithmetic units. While being a good starting point, our geometric viewpoint has probably not been fully exploited here and we still have hope for a more parallel computational scheme, operating on 3-D vectors.

An alternative way of using the geometrical insight could be derived from the general update equation(32 in [21]) which is not only valid in the scalar case but also in the multi-channel case. The idea in [21] is based on a block diagonalization of the singular-value decomposition type. This could be done for instance using an extension of the CORDIC idea to quaternion representation as in [3].

V. Conclusions

We have implemented high-throughput CORDICs for reconfigurable computing in our object-oriented hardware design environment – PAM-Blox – and optimized generic parallel CORDICs with state-of-the-art synthesis tools.

While commercial synthesis tools are very efficient in optimizing CORDICs for ASICs, FPGAs do not seem to lend themselves to these types of optimizations. At a higher level, in order to give an idea of what is involved in the automatic generation of CORDIC-like units for specific applications, we have decomposed the computations of an adaptive filter in terms of CORDIC operations, using geometric insights that could lead to high-level compilation to CORDICs.

For PAM-Blox, the natural extension is to integrate behavioral synthesis into module generation e.g. using synthesis from C [22], [23]. A behavioral method within the hardware object could be automatically transformed into multiple structural C++ methods.

CORDICs map well onto FPGAs. Due to their small area requirement, CORDICs – especially parallel forms – are most useful for certain highly parallelizeable and pipelineable applications which can take advantage of a large number of CORDIC units on a chip.

Appendix A

Compositions of rotations in 3-D space are best represented in terms of quaternions. Simply, and to facilitate the relation with [3], rotation by an angle u around an axis $\mathbf{u} = [u_x, u_y, u_z]^T$ with $u_x^2 + u_y^2 + u_z^2 = 1$ is evaluated by means of a multiplication by the matrix

$$Q = \begin{bmatrix} w & -x & -y & -z \\ x & w & -z & y \\ y & z & w & -x \\ z & -y & x & w \end{bmatrix} \qquad \begin{array}{l} \text{where} \quad w = \cos u \\ \text{and} \quad \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \cdot \sin u. \end{array} \tag{A1}$$

The product of the rotation by b around $\mathbf{b}$ (matrix Q_b) followed by the rotation by f around $\mathbf{f}$ (matrix Q_f) is given by the first column of $Q_f \cdot Q_b$, hence, in order to determine the resulting rotation angle r and direction $\mathbf{r}$, it is sufficient to multiply the first column of Q_b by Q_f. Exploiting the structure of Q_f and denoting by $\times$ the cross-product of two vectors, the evaluation of the first column of the product yields

$$\begin{bmatrix} \cos r \\ \mathbf{r} \cdot \sin r \end{bmatrix} = \begin{bmatrix} \cos f \cos b \ - \ (\mathbf{f} \sin f) \cdot (\mathbf{b} \sin b) \\ (\mathbf{f} \sin f) \cos b \ + \ \cos f (\mathbf{b} \sin b) + (\mathbf{f} \sin f) \times (\mathbf{b} \sin b) \end{bmatrix}$$

$$= \left[\begin{array}{c} \cos f \cos b - (\mathbf{f} \cdot \mathbf{b}) \sin f \sin b \\ \mathbf{f} \sin f \cos b + \mathbf{b} \cos f \sin b + (\mathbf{f} \times \mathbf{b}) \sin f \sin b \end{array} \right] . \tag{A2}$$

Specifically (see equations (3) and Figure 5)

$$\left\{ \begin{array}{llll} \cos f = \nu \ , & \sin f = \bar{\nu} \ , & \mathbf{f} = [\ 1,\ 0,\ 0\ ]^T \ , & \mathbf{f} \cdot \mathbf{b} = -\cos R = -\rho \ , \\ \cos b = \eta \ , & \sin b = \bar{\eta} \ , & \mathbf{b} = [\ -\cos R,\ 0,\ \sin R\ ]^T = [\ -\rho,\ 0,\ \bar{\rho}\ ]^T \ , & \\ \cos r = \rho_+ \ , & \sin r = \bar{\rho}_+ \ , & \mathbf{f} \cdot \mathbf{r} = \cos B = \eta_+ \ , & \mathbf{b} \cdot \mathbf{r} = \cos F = \nu_+ \ . \end{array} \right. \tag{A3}$$

Thus, expressing the vectors **f**, **b** and **r** in terms of their components, to compose the rotations we compute the product

$$\left[\begin{array}{cccc} \nu & -\bar{\nu} & 0 & 0 \\ \bar{\nu} & \nu & 0 & 0 \\ 0 & 0 & \nu & -\bar{\nu} \\ 0 & 0 & \bar{\nu} & \nu \end{array} \right] \left[\begin{array}{c} \eta \\ -\rho\bar{\eta} \\ 0 \\ \bar{\rho}\bar{\eta} \end{array} \right] = \left[\begin{array}{c} \eta\nu + \rho\bar{\eta}\bar{\nu} \\ \eta\bar{\nu} - \rho\bar{\eta}\nu \\ -\bar{\rho}\bar{\eta}\bar{\nu} \\ \bar{\rho}\bar{\eta}\nu \end{array} \right] . \tag{A4}$$

This equation provides the skeleton of the decomposition of the adaptive filter computations into CORDIC operations. Since Q_f is the composition of two independent plane rotations, this decoupling should be exploited: it is preferable here to employ 2-D CORDICs rather than a quaternion CORDIC (We outline an alternative approach that leads to the use of quaternion CORDIC at the end of Section 4.D), and apply two 2-D CORDICs in parallel for speed.
The equation implies that $\rho_+ = \cos r$ may be obtained as the first component of

$$\left[\begin{array}{cc} \nu & -\bar{\nu} \\ \bar{\nu} & \nu \end{array} \right] \left[\begin{array}{c} \eta \\ -\rho\bar{\eta} \end{array} \right] .$$

The second component of that vector is equal to the first component of $\mathbf{r} \sin r$, i.e., $\mathbf{f} \cdot (\mathbf{r} \sin r) = (\mathbf{f} \cdot \mathbf{r}) \sin r = \eta_+ \bar{\rho}_+$. We denote η^* this second component.
Similarly ν_+ can be obtained according to $\nu_+ \bar{\rho}_+ = (\mathbf{b} \cdot \mathbf{r}) \sin r = \mathbf{b} \cdot (\mathbf{r} \sin r) = -\rho\eta^* + \bar{\rho}\nu^\dagger$, where $\nu^\dagger$ is the second component of the vector

$$\left[\begin{array}{cc} \nu & -\bar{\nu} \\ \bar{\nu} & \nu \end{array} \right] \left[\begin{array}{c} 0 \\ \bar{\rho}\bar{\eta} \end{array} \right] .$$

Thus, denoting $\nu^* = \nu_+ \bar{\rho}_+$, $-\nu^*$ may be obtained as the first component of

$$\left[\begin{array}{cc} \rho & -\bar{\rho} \\ \bar{\rho} & \rho \end{array} \right] \left[\begin{array}{c} \eta^* \\ \nu^\dagger \end{array} \right] .$$

Hence, first we compute

$$\left[\begin{array}{c} \rho_+ \\ \eta^* \end{array} \right] = \left[\begin{array}{cc} \nu & -\bar{\nu} \\ \bar{\nu} & \nu \end{array} \right] \left[\begin{array}{c} \eta \\ -\rho\bar{\eta} \end{array} \right] \quad \text{and} \quad \left[\begin{array}{c} \nu^\dagger \\ \bar{\nu}\bar{\rho}\bar{\eta} \end{array} \right] = \left[\begin{array}{cc} \nu & -\bar{\nu} \\ \bar{\nu} & \nu \end{array} \right] \left[\begin{array}{c} \bar{\rho}\bar{\eta} \\ 0 \end{array} \right] \tag{A5}$$

then we evaluate

$$\left[\begin{array}{c} -\nu^* \\ \eta^\dagger \end{array} \right] = \left[\begin{array}{c} -\nu^* \\ \eta \cdot \bar{\rho}\bar{\nu} \end{array} \right] = \left[\begin{array}{cc} \rho & -\bar{\rho} \\ \bar{\rho} & \rho \end{array} \right] \left[\begin{array}{c} \eta^* \\ \nu^\dagger \end{array} \right] \tag{A6}$$

and finally we obtain

$$\nu_+ = \nu^* / \bar{\rho}_+ \quad \text{and} \quad \eta_+ = \eta^* / \bar{\rho}_+ \tag{A7}$$

This way of decomposing the equations (1), guided by our geometric interpretation, leads to a computational structure different from that of [10], [18].

Proceeding with our geometrical approach we shall also use the relation between the sines of the angles and sides of a spherical triangle ("the law of sines"),

$$\frac{\bar{\rho}}{\bar{\rho}_+} = \frac{\bar{\nu}_+}{\bar{\nu}} = \frac{\bar{\eta}_+}{\bar{\eta}}$$

to update the sines $\bar{\nu}$ and $\bar{\eta}$:

$$\bar{\nu}_+ = \bar{\nu} \cdot (\bar{\rho}/\bar{\rho}_+) \quad \text{and} \quad \bar{\eta}_+ = \bar{\eta} \cdot (\bar{\rho}/\bar{\rho}_+) \tag{A8}$$

Acknowledgments

This research is supported by DARPA Grant No. DABT63-96-C-0106 and CNET under Grant 941B098. We thank M. Flynn and G. De Micheli for support and encouragement. We would also like to thank Synopsys and Xilinx for the synthesis tools used in this paper, and Compaq Systems Research Center for support of the PAM-Blox project.

References

[1] W.H. Mangione-Smith, B. Hutchings, D. Andrews, A. DeHon, C. Ebeling, R. Hartenstein, O. Mencer, J. Morris, K. Palem, V. Prasanna, H. Spaanenburg, "Seeking Solutions in Configurable Computing", *IEEE Computer Magazine*, Dec. 1997.
[2] J.M. Muller, *"Elementary Functions: Algorithms and Implementation"*, Birkhauser, 1997.
[3] J.-M. Delosme and S.-F. Hsiao, "CORDIC Algorithms in Four Dimensions, Advanced Signal Processing Algorithms, Architectures and Implementations", *Proc. SPIE 1348*, San Diego, CA, pp. 349-360, July 1990.
[4] H.M. Ahmed, *"Signal Processing Algorithms and Architectures"*, PhD thesis, E.E. Dept., Stanford, June 1982.
[5] R. Andraka, "A Survey of CORDIC algorithms for FPGA based computers", *Sixth International Symposium on Field Programmable Gate Arrays*, Monterey, CA, 1998.

[6] J.E. Volder, "The CORDIC Trigonometric Computing Technique", *IRE Trans. on Electronic Computers*, Vol. EC-8, No. 3, pp. 330-334, Sept. 1959.
[7] J.S. Walther, "A Unified Algorithm for Elementary Functions", *AFIPS Conf*, Proc., Vol. 38, pp. 379-385, 1971.
[8] T.C. Chen, "Automatic Computation of Exponentials, Logarithms, Ratios and Square Roots", *IBM Journal of Research and Development*, July 1972.
[9] J.-M. Delosme, "VLSI Implementation of Rotations in Pseudo-Euclidean Spaces", *Proc. 1983 Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, Boston, MA, pp. 927-930, Apr. 1983.
[10] D.T.L. Lee and M. Morf, "Generalized Cordic for Digital Signal Processing", *Proc. 1982 Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, Paris, France, pp. 1748-1751, May 1982. See also PhD Thesis, E.E. Dept., Stanford, Aug. 1980.
[11] H. Al-Twaijry, *"Area and Performance Optimized CMOS Multipliers"*, PhD thesis, Stanford, August 1997.
[12] O. Mencer, M. Morf, M. J. Flynn, "PAM-Blox: High Performance FPGA Design for Adaptive Computing", *IEEE Symp. on FPGAs for Custom Computing Machines*, Napa Valley, CA, 1998, http://umunhum.stanford.edu/PAM-Blox/.
[13] O. Mencer, M. Morf, M.J. Flynn, "Hardware Software Tri-Design of Encryption for Mobile Communication Units", *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, Seattle, May 1998.
[14] O. Mencer, M. Platzner, "Dynamic Circuit Generation for Boolean Satisfiability in an Object-Oriented Design Environment", *Hawaii Int. Conf. on System Sciences (ConfigWare Track)*, Jan. 1999.
[15] *Synopsys FPGA Express*, http://www.synopsys.com/products/fpga_solution/fpga_express.html.
[16] G. De Micheli, *"Synthesis and Optimization of Digital Circuits"*, Mc Graw Hill, Highstown, NJ, 1994.
[17] *Synopsys products*, http://www.synopsys.com/products.
[18] J. Turner, chapter Chapter 5, Prentice-Hall Signal Processing Series, 1985.
[19] H.M. Ahmed, P.H. Ang and M. Morf, "A VLSI Speech Analysis Chip Set Utilizing Co-ordinate Rotation Arithmetic", *Proc. 1981 Int. Symp. on Circuits and Systems (ISCAS)*, Chicago, Illinois, pp. 737-741, Apr. 1981.
[20] H.M. Ahmed, P.H. Ang and M. Morf, "A VLSI Speech Analysis Chip Set Based on Square-Root Normalized Ladder Forms", *Proc. 1981 Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, Atlanta, GA, pp. 648-653, Mar.-Apr. 1981.
[21] M. Morf and D.T.Lee, "State-space structure of ladder canonical forms", *Proc. 18th Conf. on Control and Design*, pp. 1221-1224., Dec. 1980.
[22] L. Séméria, G. De Micheli, "Synthesis of Pointers in C: Application of Pointer Analysis to the Behavioral Synthesis from C", *Proceedings of the International Conference on Computer-Aided Design (ICCAD 1998)*, San Jose, November 98.
[23] T.J. Callahan, J. Wawrzynek, "Instruction Level Parallelism for Reconfigurable Computing", *FPL'98*, Tallinn, Estonia, Published in Springer-Verlag LNCS 1482, pp. 248-258, Sept. 1998.

List of Figures

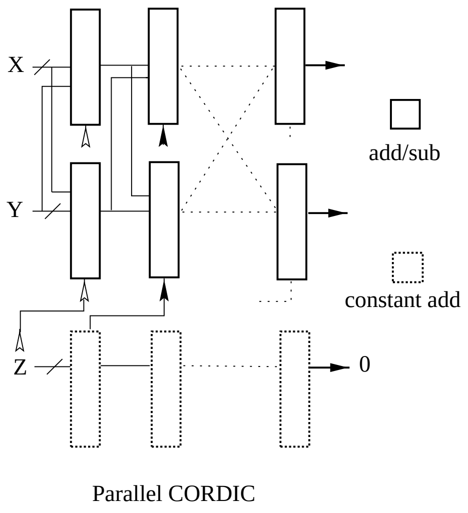

Fig. 1. *The figure shows a block diagram of a parallel CORDIC architecture. The table for the z-pipe is coded implicitly in the constant adders. All adders include a wired shift of the operands. The shift amount is chosen to eliminate the scaling factor. For a bit-serial CORDIC simply replace parallel adders with bit-serial adders, add delay elements and a table for z-values.*

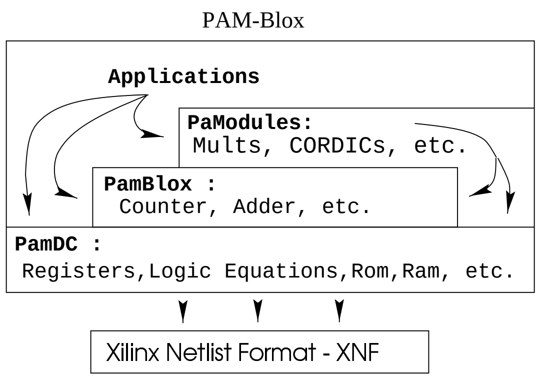

Fig. 2. *Layers of the PAM-Blox design environment: DIGITAL PamDC compiles the design to the Xilinx Netlist Format XNF; PamBlox are interacting with PamDC objects, PaModules interact with PamBlox and PamDC, and the application can access features from all three layers below.*

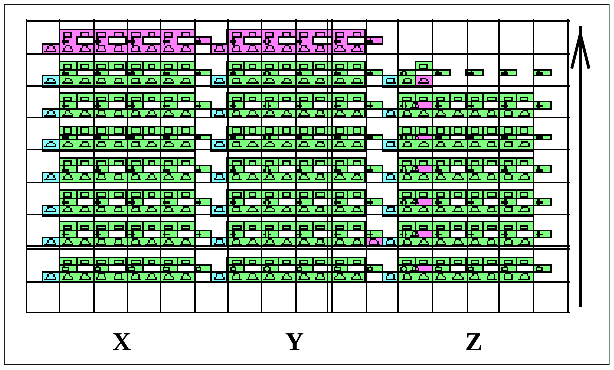

Fig. 3. *Layout of the PP-CORDIC, placed with PAM-Blox.*

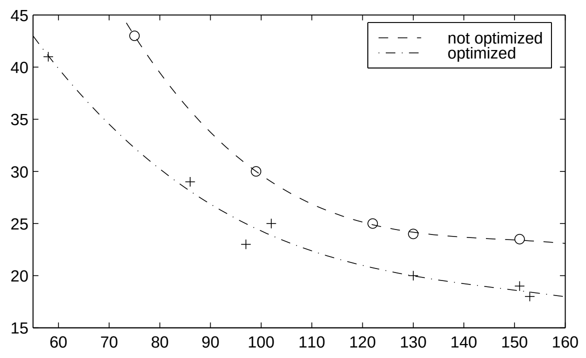

Fig. 4. *Area-time curves for synthesized CORDICs*

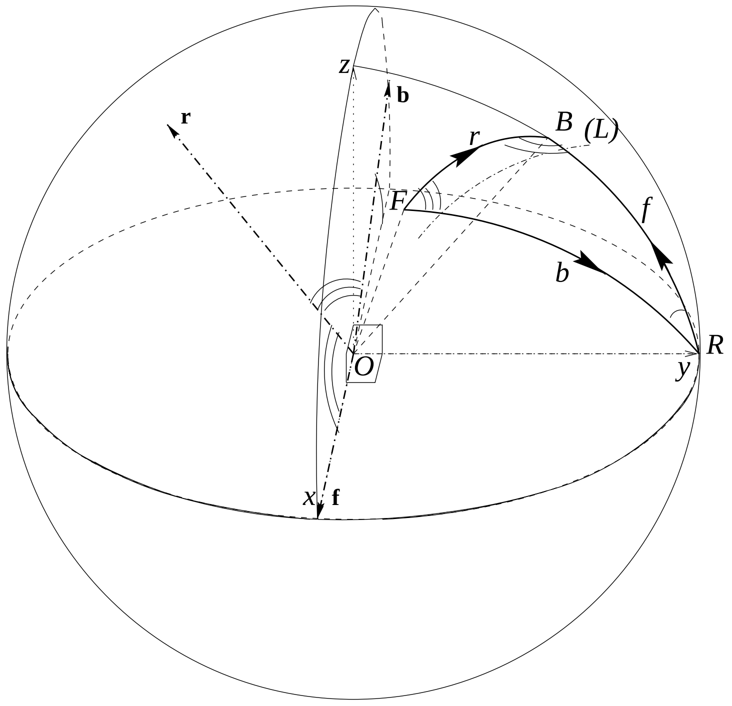

Fig. 5. *Geometric interpretation of the normalized ladder algorithm in terms of spherical trigonometry. From the information bRf, to the right of (L), we deduce the information FrB to the left of (L). This amounts to computing a rotation, from F to B, as the composition of two rotations, from F to R and from R to B.*

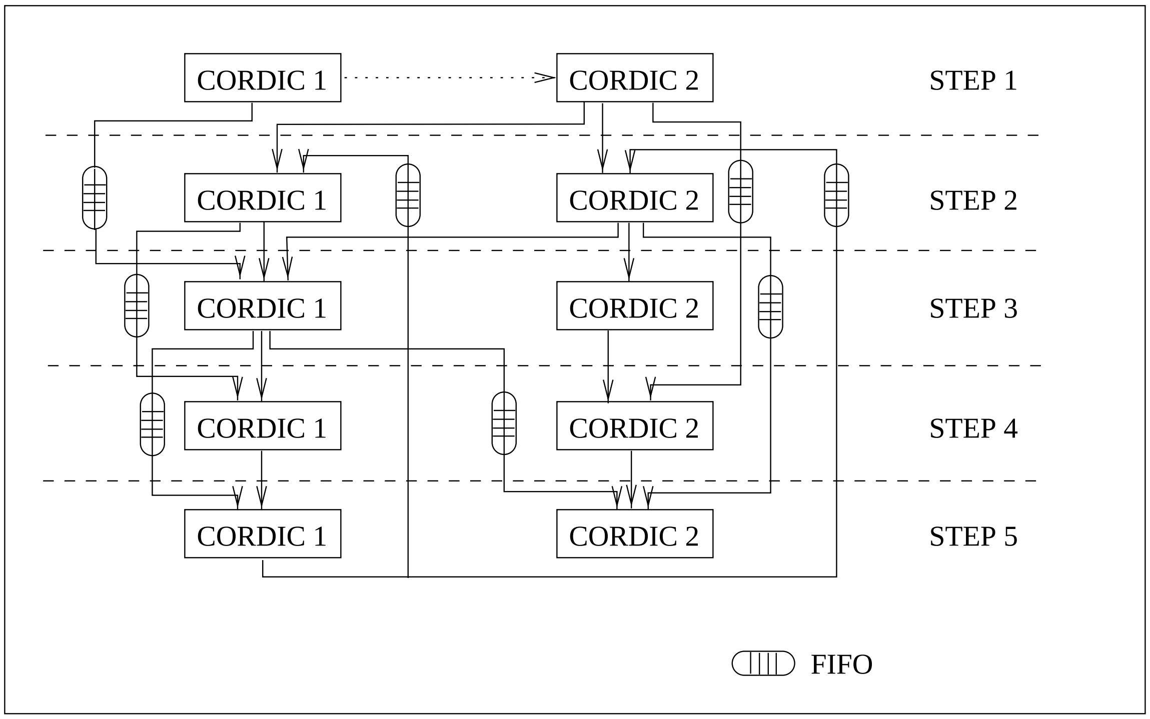

Fig. 6. *General dataflow of the proposed filter stage implementation.*

List of Tables

$x \cdot \cos(z) - y \cdot \sin(z)$	$y \cdot \cos(z) + x \cdot \sin(z)$
x	$y + x \cdot z$
$x \cdot \cosh(z) + y \cdot \sinh(z)$	$y \cdot \cosh(z) + x \cdot \sinh(z)$
$\sqrt{x^2 + y^2}$	$z + \tan^{-1}(y/x)$
x	$z + (y/x)$
$\sqrt{x^2 - y^2}$	$z + \tanh^{-1}(y/x)$

TABLE I
Functions computed by CORDIC

CORDIC on FPGAs

PP-CORDIC	CYCLE TIME	AREA
FE-II	25.4 ns	133 CLBs
PAM-Blox	23.7 ns	131 CLBs

TABLE II
The table shows area and cycle time for PAM-Blox and Synopsys FPGA Express II (FE-II) for Xilinx XC4000 FPGAs at 0.5μ technology (speedgrade -3), after Xilinx place-and-route. The area of the parallel CORDIC (PP-CORDIC) is given in Configurable Logic Blocks, CLBs(cells).