Application of Reconfigurable CORDIC Architectures

Oskar Mencer, Luc Semeria, Martin Morf

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

Jean-Marc Delosme

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

Abstract

Very high performance architectures can be designed for data intensive and latency tolerant applications by maximizing the parallelism and pipelining at the algorithm and bit level. This is achieved by combining such technologies as reconfigurable or adaptive computing and CORDIC style arithmetic, for computing (possibly hyperbolic) rotations, multiply, divide, and related higher order functions (e.g. square-root, multidimensional rotations). Reconfiguration allows adapting the implementation of such functions to the specific needs of individual or specific sets of applications, from multi-media to radar and sonar, hence creating application specific CORDIC-style implementations. We show a high-throughput CORDIC for reconfigurable computing, a low latency CORDIC, and discuss an application to adaptive filtering (normalized ladder algorithm).

1. Introduction

The fundamental principles behind the CORDIC algorithms of Volder [10] and Walther [11] can be found in their scalar form in the work of Chen [2]. Ahmed showed [1] 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 [4], [3] to obtain for instance 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. Field-Programmable Gate Arrays (FPGAs) and other fine grain architectural features, enable effective hardware support of such complex functions, similar to micro-code or firmware (library functions). This allows hiding the complexity involved from a typical applications 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 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.

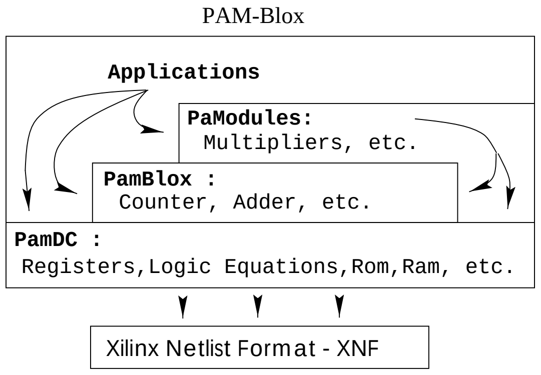

Figure 1: *Layers of the PAM-Blox design environment: DIGITAL PamDC compiles the design to the Xilinx Netlist Format XNF; PamBlox are interacting with PamDC objects, PaModules interact with PamBlox and PamDC, and the application can access features from all three layers below.*

2. PAM-Blox: Object-Oriented Hardware Design

For datapaths, hand layouts are typically more efficient than compiled behavioral descriptions. In order to exploit the efficiency of hand design while simplifying the design process, we propose a bottom-up approach to compilation for custom computing machines. By creating a powerful and highly optimized parameterizable repository of circuit generators, PAM-Blox [5], we add a level of abstraction that preserves optimal area and performance while simplifying the design process.

Figure 1 shows an overview of the PAM-Blox system. We use PAM-Blox as the name for the entire design environment. PamBlox stands for templates of hardware objects

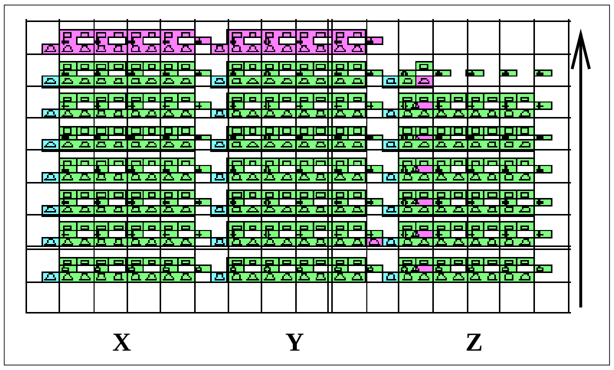

Figure 2: *Layout of the PP-CORDIC, placed with PAM-Blox.*

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 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 make PAM-Blox a competitive and yet simple to use design environment.

3. High Throughput CORDIC

We use the PAM-Blox environment to create the optimized, parallel CORDIC functional units. Currently the CORDICs are implemented as PaModules with a fixed bitwidth. A floorplan for a parallel-parallel (PP) CORDIC is shown in Figure 2. The 8-bit PP-CORDIC requires 123 CLBs while a serial-parallel (SP) CORDIC, with 23 serial adders requires 171 CLBs. In contrast, a PS-CORDIC iterating with 3 ADD/SUB modules on the CORDIC equations would have a very long latency of over 200 ns, and an area penalty for the z lookup table which is hardwired in the parallel case.

Although serial arithmetic usually takes less area, the SP-CORDIC occupies 30% more area than the PP-CORDIC. This is mainly 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, make the 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. With current FPGA technology the throughput would scale up easily to 100 MHz $\rightarrow$ 100 million rotations per second.

4. Optimization using Hardware Synthesis

For the implementation of CORDIC in hardware, many of the operators (adders, subtractors) 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 architectual optimization for a non-pipelined version of the CORDIC and synthesize the design for both FPGA and ASIC (Application Specific Integrated Circuit).

4.1. Logic Optimization

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 feedback), 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 [6]. Very powerful optimization can be performed under both area 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 [8] and allow quick estimation of the performance of multiple architectures.

4.2. Synthesis for ASIC

For ASIC synthesis we use Synopsys Design Compiler to synthesize the circuit and Synopsys Behavioral Compiler [8] for the arithmetic optimization. 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 3 presents the area-time curves with and without architectual and logic optimizations. We observe that the circuit after optimization is about 20% smaller for a given latency and can be as much as 20% faster. The smallest design (with optimization) had 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 on the same area as the optimized design.

4.3. Synthesis for FPGA

For the purpose of reconfigurable computing we try to optimize a CORDIC architecture for Xilinx XC4000 FPGAs. For FPGA synthesis we use Synopsys FPGA Express [7].

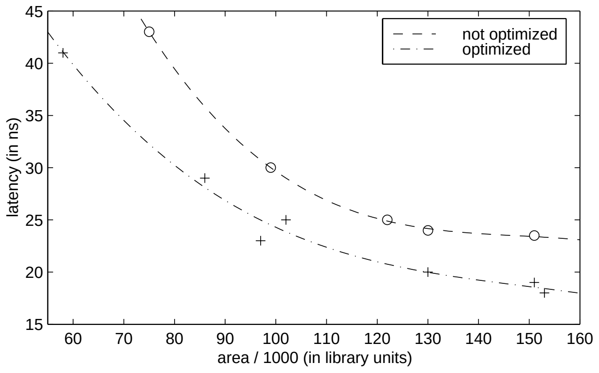

Figure 3: *The graph shows the area-time curves for the synthesized CORDICs*

The results after optimization are comparable to the PAM-Blox results found in section 3: after place&route the area of the circuit is 113 CLBs with a pure combinational latency (without registers) of 141 ns. We conclude that for FPGAs, the possible optimizations are restricted by the internal architecture of the CLBs – especially the fast carry-chains. This result matches the results from our initial study on PAM-Blox [5].

5. CORDIC for normalized ladder algorithms

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 [4] and detailed in Section 7 of the survey [9]. The adaptive ladder (or lattice) filter is an FIR filter used for the prediction of stochastic processes, e.g. for channel equalization. It 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 in 3 equations (the stage and time indices are not shown here)

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

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

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.

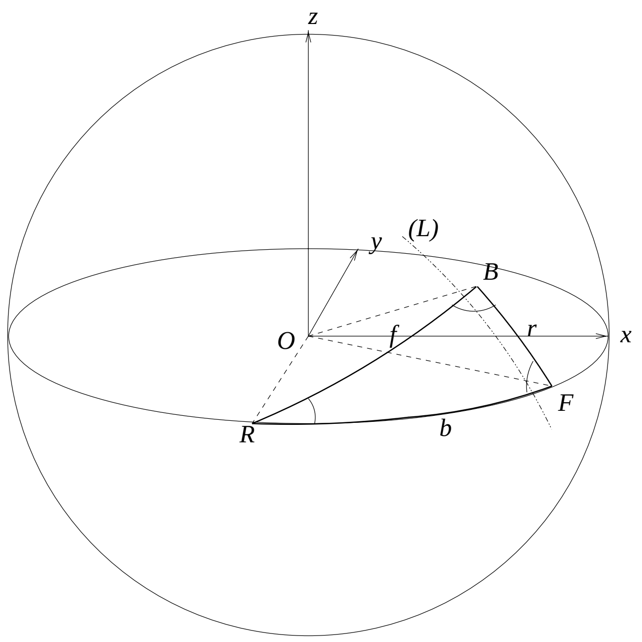

Figure 4: *Geometric interpretation of the normalized ladder algorithm in terms of spherical trigonometry. From the information fRb, to the left of (L), we deduce the information BrF to the right of (L). This amounts to computing a rotation, from B to F, as the composition of two rotations, from B to R and from R to F.*

Considering the triangle RBF in figure 4, and measuring both the angles R, B, F and the sides r, b, f in radians, we can write 3 identities from spherical trigonometry:

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

These identities enable the determination of information to the right of the dashed line (L) in Figure 4 in terms of information to the left of (L).

With the correspondence

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

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 [10] developed the CORDIC procedure precisely to solve digitally such problems and showed how to link CORDIC rotations for that purpose. Following a similar vein, Lee et al. [4] proposed a way for linking the 3 types of CORDIC operations of Walther [11] to evaluate the expressions (1) (this way is also presented in [9], 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 B to R along f and the "forward" rotation from R to F along b; the cosine of the angle R between the sides f and b corresponds naturally to the normalized partial correlation. That result corresponds to the rotation from B to F along r, whose parameters are what we seek.

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 θ around an axis $\vec{u} = [\alpha, \beta, \gamma]^T$ with $\alpha^2 + \beta^2 + \gamma^2 = 1$ is evaluated by means of a multiplication by the matrix

$$Q = \begin{bmatrix} w & -x & -y & -z \\ x & w & -z & y \\ y & z & w & -x \\ z & -y & x & w \end{bmatrix} \quad \text{where} \quad w = \cos\theta \quad \text{and} \quad \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix} \cdot \sin\theta. \tag{4a}$$

The product of the rotation by θ around $\vec{u}$ (matrix Q) followed by the rotation by θ' around $\vec{u}'$ (matrix Q') is given by the first column of $Q'Q$, hence, in order to determine the resulting rotation angle ϕ and direction $\vec{v}$, it is sufficient to multiply the first column of Q by Q'. The evaluation of the first column of the product yields

$$\begin{bmatrix} \cos\phi \\ \vec{v} \cdot \sin\phi \end{bmatrix} = \tag{4b}$$

$$\begin{bmatrix} \cos\theta' \cdot \cos\theta - (\vec{u}' \cdot \vec{u}) \cdot \sin\theta' \cdot \sin\theta \\ \vec{u}' \cdot \sin\theta' \cdot \cos\theta + \vec{u} \cdot \cos\theta' \cdot \sin\theta + (\vec{u}' \times \vec{u}) \cdot \sin\theta' \cdot \sin\theta \end{bmatrix}.$$

In our case $\cos\theta' = \cos b = \eta$, $\vec{u}' = [0, 0, 1]^T$; $\cos\theta = \cos f = \nu$, $\vec{u} = [\sin R, 0, -\cos R]^T = [\bar{\rho}, 0, -\rho]^T$; $\cos\phi = \cos r = \rho_+$, $\vec{u} \cdot \vec{v} = \cos B = \eta_+$, $\vec{u}' \cdot \vec{v} = \cos F = \nu_+$. Thus, specifically, to compose the rotations we compute the product

$$\begin{bmatrix} \eta & 0 & 0 & -\bar{\eta} \\ 0 & \eta & -\bar{\eta} & 0 \\ 0 & \bar{\eta} & \eta & 0 \\ \bar{\eta} & 0 & 0 & \eta \end{bmatrix} \begin{bmatrix} \nu \\ \bar{\rho}\bar{\nu} \\ 0 \\ -\rho\bar{\nu} \end{bmatrix} = \tag{4c}$$

$$\begin{bmatrix} \eta\nu + \rho\bar{\eta}\bar{\nu} \\ \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \cdot \bar{\eta}\nu + \begin{bmatrix} \bar{\rho} \\ 0 \\ -\rho \end{bmatrix} \cdot \eta\bar{\nu} + \begin{bmatrix} 0 \\ \bar{\rho} \\ 0 \end{bmatrix} \cdot \bar{\eta}\bar{\nu} \end{bmatrix} = \begin{bmatrix} \rho_+ \\ \bar{\rho}\eta\bar{\nu} \\ \bar{\rho}\bar{\eta}\bar{\nu} \\ \bar{\eta}\nu - \rho\eta\bar{\nu} \end{bmatrix}.$$

Hence $(\vec{u} \cdot \vec{v}) \cdot \bar{\rho}_+ = \bar{\rho}^2\eta\bar{\nu} - \rho(\bar{\eta}\nu - \rho\eta\bar{\nu}) = \eta\bar{\nu} - \rho\bar{\eta}\nu$ and $(\vec{u}' \cdot \vec{v}) \cdot \bar{\rho}_+ = \bar{\eta}\nu - \rho\eta\bar{\nu}$.

Since Q' is the composition of 2 independent plane rotations, it is preferable here to use standard 2-D CORDIC than a quaternion CORDIC algorithm, and apply the 2 standard 2-D CORDIC in parallel for speed. Hence, first we compute

$$\begin{bmatrix} \rho_+ \\ \nu^* \end{bmatrix} = \begin{bmatrix} \eta & -\bar{\eta} \\ \bar{\eta} & \eta \end{bmatrix} \begin{bmatrix} \nu \\ -\rho\bar{\nu} \end{bmatrix} \quad \text{and} \quad \eta^\dagger = \eta \cdot \bar{\rho}\bar{\nu} \tag{5a}$$

then we evaluate

$$\eta^* = (\bar{\rho} \cdot \eta^\dagger - \rho \cdot \nu^*) \tag{5b}$$

and finally we obtain

$$\eta_+ = \eta^* / \bar{\rho}_+ \quad \text{and} \quad \nu_+ = \nu^* / \bar{\rho}_+ \ . \tag{5c}$$

This way of decomposing the equations (1), guided by our geometric interpretation, leads to a computational structure different from that of [4], [9].

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"),

$$\frac{\bar{\rho}}{\bar{\rho}_+} = \frac{\bar{\eta}_+}{\bar{\eta}} = \frac{\bar{\nu}_+}{\bar{\nu}} \ , \tag{6}$$

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

$$\bar{\eta}_+ = \bar{\eta} \cdot (\bar{\rho}/\bar{\rho}_+) \quad \text{and} \quad \bar{\nu}_+ = \bar{\nu} \cdot (\bar{\rho}/\bar{\rho}_+) \ . \tag{5d}$$

Finally, casting these computations in terms of *pairs* of 2-D CORDIC operators (represented below by ◉ *and* ◎ *)*, where the operators do not have the "Z-factor" part that computes the angles explicitly since it is not needed here, we obtain:

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

◉ 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 to the vector $\begin{bmatrix} \bar{\nu} \\ 0 \end{bmatrix}$ 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 obtain $\begin{bmatrix} \rho\bar{\nu} \\ \bar{\rho}\bar{\nu} \end{bmatrix}$.

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

◉ use "linear" CORDIC iterations to find the representation of η as a sequence of signs (i.e. non-restoring division of η by 1) and simultaneously build up the product $\eta \cdot \bar{\rho}\bar{\nu} = \eta^\dagger$,

◎ apply to the vector $\begin{bmatrix} \nu \\ -\rho\bar{\nu} \end{bmatrix}$ the rotation $\mathcal{R}(b)$, determined by the encoding of b (obtained in Step 5 of previous update), to obtain $\begin{bmatrix} \rho_+ \\ \nu^* \end{bmatrix}$.

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

◉ apply $\mathcal{R}(R)$ to $\begin{bmatrix} \eta^\dagger \\ \nu^* \end{bmatrix}$ to get $-\eta^*$ as 1st component,

◎ 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.

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

◉ apply to the vector $\begin{bmatrix} \bar{\eta} \\ 0 \end{bmatrix}$ the rotation $\mathcal{R}(R)$ to get as 2nd component $\bar{\rho}\bar{\eta} = \bar{\rho}_+\bar{\eta}_+$,

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

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

◉ rotate $\left[\begin{array}{c}\eta^* \\ \bar{\rho}_+\bar{\eta}_+\end{array}\right]$ to force the 2nd component to 0 and thus obtain the encoding (sign sequence) of the angle B to be used as encoding of the "angle" b in Step 2 for the next update,
◎apply in the linear mode the sign sequence encoding $1/\bar{\rho}_+$ both to η^*, to get η_+, and to ν^*, to get ν_+.

This decomposition in terms of 2-D CORDIC operations has the same complexity as the solution presented in [4], [9], assuming that this older solution is modified to benefit from the fact that the angles do not have to be computed explicitly. It has however the advantage of having an underlying geometric interpretation. On one hand one may use the relations introduced here, such as (6), and depart from the geometrical interpretation in an attempt to simplify the computations, especially those for η_+ which, by symmetry, could be made as simple as the computations for ν_+. On the other hand 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.

The accuracy d needed for the computations will typically be about 16 or 20 bits.

A conceptually straightforward implementation of one stage would use two PS-CORDIC (without z-part) working in parallel to implement the 5 steps; a stage would therefore take about $5d$ iterations for a time update. Globally, the filter, which typically requires $n = 10$ to 20 or more stages, could be implemented as a pipeline of stages (hence taking about $5d$ iterations for the update of all the stages) or sequentially (computation time being then multiplied by the order n of the filter) or in any intermediate fashion.

An interesting alternative would use two PP-CORDIC (without z-part) working in parallel. The data from the stages would be pipelined so that Step 1 for all stages would be computed first, followed by Step 2 for all stages, etc. The depth of the pipeline would be d and the pipeline would process $5n$ sets of 4 input values per time update; if the order n exceeds the depth, the pipeline would take about $5n+d$ cycles to perform the time update, to be compared to the $5d$ to $5dn$ cycles of the PS-CORDIC solutions above. If $d/2 \leq n$, a one PP-CORDIC solution would take about $10n+d$ cycles to perform the time update.

6. Conclusions

We have implemented high throughput CORDICs for reconfigurable computing in our object-oriented hardware design environment – PAM-Blox – and synthesized generic parallel CORDICs with Synopsys Design Compiler for minimal area, and minimal latency.

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.

7. Acknowledgments

This research is supported by DARPA Grant No. DABT63-96-C-0106 and CNET under Grant 941B098. We would also like to thank Synopsys and Xilinx for the synthesis tools used in this paper.

8. References

[1] H.M. Ahmed, *Signal Processing Algorithms and Architectures,* PhD Thesis (with M. Morf), E.E. Dept., Stanford, June 1982.

[2] T.C. Chen, *Automatic Computation of Exponentials, Logarithms, Ratios and Square Roots,* IBM Journal of Research and Development, July 1972.

[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] 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, May 3-5, 1982, pp. 1748-1751. See also PhD Thesis, E.E. Dept., Stanford, August 1980.

[5] O. Mencer, M. Morf, M. J. Flynn, *PAM-Blox: High Performance FPGA Design for Adaptive Computing,* IEEE Symposium on FPGAs for Custom Computing Machines, Napa Valley, CA, 1998. *http://umunhum.stanford.edu/PAM-Blox/*

[6] G. De Micheli, *Synthesis and Optimization of Digital Circuits,* Mc Graw Hill, Highstown, NJ, 1994.

[7] Synopsys FPGA Express, *http://www.synopsys.com/products/fpga_solution/fpga_express.html*

[8] Synopsys products, *http://www.synopsys.com/products*

[9] J. Turner, Chapter 5 in *Adaptive Filters,* C.F.N. Cowan and P.M. Grant editors, Prentice-Hall Signal Processing Series, 1985.

[10] J.E. Volder, *The CORDIC Trigonometric Computing Technique,* IRE Trans. on Electronic Computers, Vol. EC-8, No. 3, pp. 330-334, Sept. 1959.

[11] J.S. Walther, *A Unified Algorithm for Elementary Functions,* AFIPS Conf. Proc., Vol. 38, pp. 379-385, 1971.