Design and Implementation of a Floating-point Quasi-Systolic General Purpose CORDIC Rotator for High-rate Parallel Data and Signal Processing

Alfons A.J. de Lange and Ed F. Deprettere
Department of Electrical Engineering
Delft University of Technology
2628 CD Delft, The Netherlands

Abstract

We describe the design and implementation of an algorithm and a processor which can be used to accelerate computations in which large amounts of rotations (circular as well as hyperbolic) are involved. The processor is a low-cost high-throughput vlsi implementation of the algorithm. With 10^7 *rotations per second, many real-time and interaction-time applications in scientific computation become feasible.*

1. Introduction

A central problem in image processing (HDTV, video phone), document processing (digital copiers), imaging (medical, acoustic radar), dynamic system modeling, simulation and visualization (computational sciences), computer graphics (photo realism, animation) and other time critical computational intensive applications [Kun88]) is : how can the algorithm be so (re-)designed that it can meet the time/space requirements. A solution to this problem is ideally obtained by deriving a parallel algorithm in which the basic functions have been carrefully converted into opimally designded software and/or hardware operations. In applications from time-critical domains such as computer graphics, image processing etc. fast hardware functions are usally paramount. In such cases, the performance of multi-processor or processor array implementations will heavily depend on the performance of its basic software functions (in programmable processors) or hardware functions (in application specific hardware). In case the basic functions are simple, say of type multiplication, the corresponding software/hardware algorithms will not degrade performance when they are also kept simple. However, when basic functions become more complex, it will not in general be acceptable to realize them in terms of a composition of the simple functions since their overall efficiency, in terms of time and/or space, will be much lower than for newly designed operations.
For example, in many signal processing and matrix computation applications, i.e. speech processing, radar, computer graphics and antenna array processing [Jai89, Vee88, Dep89]. Algorithms in these domains *can most advantageously* be expressed in terms of basic plane rotations, yielding much more stable and often also more structured algorithms than the conventional algorithms. For a typical example, see [Jai89]. However, the advantage may quickly turn into a disadvantage when the basic plane rotation itself is not implemented in an optimal way. The latter will usually mean non-iteratively and unconditionally (if possible), minimal number of evaluation steps, inherent accuracy etc.

To pursue the plane rotation example, it is well known that circular implemented using Taylor or Chebychev expansions, which requires a number of multiplication and addition/subtraction steps. See also [Ahm85, Sch83, Nav83] and[Mul85]. Another solution that is more appealing for applications where rotation evaluation time must be a constant is the bit-recursive shift and add *Coordinate Digital Computer* (CORDIC) approach. See [Vol59]. In application specific algorithm design, Volder's algorithm is easily implemented in both software and hardware. However, in case one would like to go a bit more general, in the sense that one wants a bit-recursive shift-and-add implementation of Walter's unified plane rotation

$$R_m(\alpha) = \begin{bmatrix} \cos(m^{1/2}\alpha) & -m^{1/2}\sin(m^{1/2}\alpha) \\ m^{-1/2}\sin(m^{1/2}\alpha) & \cos(m^{1/2}\alpha) \end{bmatrix}$$

which is circular, linear or hyperbolic in case m=1, m=0 and m=-1 respectively, the task is much more involved. The reason is that the hyperbolic rotation can not be implemented in the same (minimal bit-recursive) way as the circular rotation, whence it is not at all obvious wheter a unified algorithm for the unified function exists and if so, what the optimal implementation will be. See e.g.[Coc72, Hav70], and [Hwa79].

Several implementations of the CORDIC algorithm in hardware are described in [Hav70, Coc72]. AT&T, Motorola and Intel have released products using CORDIC arithmetic. These are respectively the WE32206, the 68881 and the 80x87. HP built a CORDIC floating point processor box for their 2100 series of minicomputers, however it was never sold. The CORDIC hardware in these products is meant to compute transcendental functions like sin, cos, arcsin, arccos, sinh, cosh, arcsinh, arccosh, exp, log and sqrt. Currently, CORDIC hardware is only used in desk-calculators. This is because it is generally felt that a CORDIC co-processor adds much

CH3015-5/91/0000/0272$01.00 © 1991 IEEE

hardware and communication links to the multiplier-adder unit of a standard computer. It is cheaper to add some simple sequencing logic or micro-programs to a multiplier-adder unit, which allows polynomial approximation of transcendental functions. In short, the application of a CORDIC processor for the computation of transcendental functions is not very efficient. This is not true for the application of fast CORDICs in massive-parallel algorithms where they can significantly accelerate system performance. An example of a more useful CORDIC processor that was designed recently is the TMC2330 of TRW LSI Products. It is a pipelined implementation that is used to accelerate circular coordinate transformations in computer graphics. See[Wil89]. Another useful CORDIC processor is the word-level pipelined core processor applied in the Plessey (Plessey Research Caswell Ltd, UK) PDPSP 163XX series products. However, no vectorization mode has been built in this core processor, which makes it useless in most of the previously mentioned applications. We even believe that a CORDIC processor is only useful if *both vectorization and rotation are possible and can be executed simultaneously*. With the latter we moreover mean *also working on the same angle*.

In this paper, we present a CORDIC algorithm and processor which allow a substantial acceleration of many known and novel algorithms in the application domains of computational algebra and physics, in particular time critical applications which must be implemented in parallel machines or arrays of processors.

In the *next section* a modified version of the CORDIC algorithm as originally proposed by Volder [Vol59] and Walther [Wal71] is presented. This modification enables a low-cost simultaneous implementation of the various rotation functions in several coordinate systems. Tthis section also describes the derivation of optimal parameters for the proposed CORDIC algorithm. *Section 3* briefly discusses several CORDIC (hardware) architectures with different speed and hardware complexity properties. High-speed architectures are carefully considered, because a CORDIC processor implementation must be applicable in real-time signal processing computations. *Section 4* describes a word-level floating point extension to the CORDIC algorithm of section 2 and 3. This extention is necessary to meet a true 16-bit accuracy which is minimally required in most of the applications we are referring to in this paper. *Section 5* gives the accuracy analyses of the CORDIC algorithm. In this section all parameters are determined that influence the accuracy and dynamic range of computations. In *section 6*, we present the specification of a prototype CORDIC which has been fabricated and used by the authors. Finally, in *sections 7*, conclusions about the novelty of the work are made.

2. The CORDIC Algorithm

In this section we briefly review the CORDIC concept in so far as our own needs go. Then we proceed with the the modifications we are proposing and the motivations for doing so.

The original CORDIC algorithm is a bit-recursive shift-and-add implementation of the forward and backward Givens rotation, that is, if we put

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}, \tag{2.1}$$

then in the forward rotation, called *rotation*, the algorithm computes the rotation of a vector $[x\ y]^t$ over an angle α to a new vector $[x'\ y']^t$, and in the backward rotation, called *vectoring*, it computes the length x' and the inclination α of a vector $[x,\ y]^t$. The 2 procedures are eccentially the same. The function (2.1) can be easily generalized as follows[Wal71]:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(m^{½}\alpha) & -m^{½}\sin(m^{½}\alpha) \\ m^{-½}\sin(m^{½}\alpha) & \cos(m^{½}\alpha) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}. \tag{2.2}$$

which is the original Givens rotation for m=1, and includes, moreover, the hyperbolic rotation (m=-1) and the linear rotation (m=0). Figure 2.1 shows the three rotations. With appropriate definitions of m-norms (lengths) and m-arguments (angles) the 3 rotations (forward and backward) are vector norm preserving.

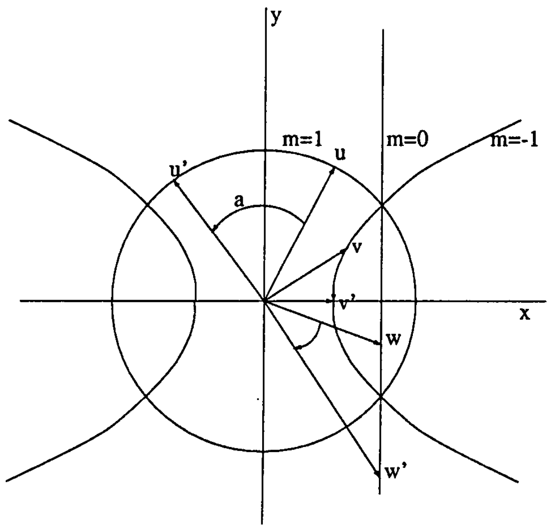

Figure 2.1. Rotation in the Circular, Hyperbolic and Linear Coordinate Systems. Shown is (1) rotation of a vector u to u′ over an angle α in the circular coordinate system, (2) vectorization of a vector v to v′ in the hyperbolic coordinate system, and (3) rotation of vector w to w′ in the linear coordinate system over an angle β.

The original CORDIC algorithm essentially decomposes the rotation in a minimal product of rotations, each one embedding an angle of fixed magnitude and variable sign such that the signed sum of angles can be any angle in the range of angles of resolution dependent on the number of elementary rotations. However the "minimal" number is not independent of the coordinate system (m) and the choice of elementary angle values is neither unique nor unifirm over the m-values. In traditional algorithms, the sets of fixed positive angles (called basic angles) are selected appropriately to each coordinate system and are denoted by $\alpha_{m,i}$, with $i = 0, 1, \cdots p(m)$, wher $p(m)+1$ is the (minimal) number of elementary rotations. Hence the angle α, the number of basic rotations $p(m)$, and the set of basic angles $\alpha_{m,i}$ satisfy the following relation :

$$\sum_{i=0}^{p(m)} \sigma_i \alpha_{m,i} = \alpha, \quad \text{and} \quad \sigma_i \in [-1\ ,\ 1]. \tag{2.3}$$

So, equation 2.2 can be rewritten as :

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \prod_{i=0}^{p(m)} \begin{bmatrix} \cos(m^{½}\sigma_i\alpha_{m,i}) & -m^{½}\sin(m^{½}\sigma_i\alpha_{m,i}) \\ m^{-½}\sin(m^{½}\sigma_i\alpha_{m,i}) & \cos(m^{½}\sigma_i\alpha_{m,i}) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$
$$= K \prod_{i=0}^{p(m)} \begin{bmatrix} 1 & -m^{½}\tan(m^{½}\sigma_i\alpha_{m,i}) \\ m^{-½}\tan(m^{½}\sigma_i\alpha_{m,i}) & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}. \tag{2.4}$$

where

$$K = \prod_{i=0}^{p(m)} \cos(m^{½}\alpha_{m,i})\ .$$

The right hand side decomposition form in eqn. (2.4) is now inteerpreted as follows:

1. Choose the angles $\alpha_{m,i}$ in such a way that the term $m^{-½}\tan(m^{½}\alpha_{m,i})$ equals $2^{-S_{m,i}}$, where $S_{m,i}$ is a non-negative integer.
2. Get the scaling factor $\prod_{i=0}^{p(m)}\cos(m^{½}\alpha_{m,i})$ included in pre-operations or post-operations, or distribute it over the elementary rotations, using (if possible) approximations $\cos(m^{½}\alpha_{m,i}) \approx (1-m\varepsilon_{m,i}2^{-2S_{m,i}})$, with either $\varepsilon_{m,i} = 0$, or $\varepsilon_{m,i} = 1$.

The CORDIC algorithms will then approximately implement the functions (2.2) in the following sense:

Let:

$$m^{½}\tan^{-1}(m^{½}\ \sigma_i\ \alpha_{m,i}) = 2^{-S_{m,i}}),\quad S_{m,i} \in N, \tag{2.5}$$

$$½\alpha_{m,i} \le \alpha_{m,i+1} \le \alpha_{m,i}, \tag{2.6}$$

$$|\,\alpha - \sum_{i=0}^{p(m)} \sigma_i\alpha_{m,i}\,| \le \alpha_{m,p(m)}, \tag{2.7}$$

and:

$$\prod_{i=0}^{p(m)}(1-m\varepsilon_{m,i}2^{-S_{m,i}}) \approx \prod_{i=0}^{p(m)}\cos(m^{½}\alpha_{m,i}) \tag{2.8}$$

Then eq. 2.2 can approximately be rewritten as :

for i=0,1,...p(m)

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} = (1-m\varepsilon_{m,i}2^{-S_{m,i}}) \begin{bmatrix} 1 & m\,\sigma_i\,2^{-S_{m,i}} \\ -\sigma_i\,2^{-S_{m,i}} & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}, \tag{2.9}$$

with either $\varepsilon_{m,i} = 0$ or $\varepsilon_{m,i} = 1$, and either $\sigma_i = -1$ or $\sigma_i = 1$[†].
In case of rotation, the σ_i $(i = 0,1,..,p(m))$ are computed by evaluating

$$\sigma_i = \text{sign}(\alpha - \sum_{j=0}^{i-1}\sigma_j\alpha_{m,j}). \tag{2.10}$$

In case of vectoring, σ_i $(i = 0,1,..,p(m))$ are computed by inverting the signs of y_i $(i = 0,1,..,p(m))$:

$$\sigma_i = -\text{sign}(y_i). \tag{2.11}$$

Due to the dependency of all coefficients of the parameter m, the set of algorithms as given above is much less unified as the function (2.2) is. For example, if we require an accuracy of M e.g 2^{-16} — and an angle range condition (2.7) — e.g. $-\pi \le \alpha \le \pi$ — and accuracy — e.g. 2^{-16} —, then the number of basic angles, hence of *micro rotations*, will be different for different values of m, since the convergence condition (2.6) depends on the parameter m, See[Dep84], and all shift exponents in the set m=1 will be different from those in the set m=-1, this is expensive, both in software and hardware implementations. The range $[-\pi\ ,\ \pi]$ is very realistic; it is *the* range for the circular rotation and it is an *acceptable* range for the hyperbolicrotation (having an infinite range in theory). On the other hand; if the condition would be that execution times must be independent of the papprameter m — this is necessary in many applications that use rotations in both circular and hyperbolic coordinate systems —, then different conditions hold for the accuracy and range of angles α (eq. 2.7, accuracy is determined by $\alpha_{m,p(m)}$) in different coordinate system. This is unacceptable if both circular and hyperbolic rotations are used in the same algorithm (see e.g. speech coding[Jai86]). Another drawback of the original CORDIC algorithms as described above are the auxiliary angle *decomposer* (eq. 2.10) and a result angle *composer* : $\sum_{i=0}^{p(m)}\sigma_i\alpha_{m,i}$. These operations are neither necessary nor useful. For example, in many applications one has to rotate a set of vectors over an angle that is first to be computed from a given vector. In such cases the decomposer succeeds the composer creating a perfect identity that is not only useless but also prevents the sequence of rotations to be started almost concurrently with the vectorization. If, on the other hand encoded angle

† Some authors allow the σ_i's to become zero. This will speed up the determination of the sign of y_i and the sign of $\alpha-\sum_{j=0}^{i-1}\sigma_j\alpha_{m,j}$ if α and x/y_i have a binary redundant representation [Dup89], which will be explained in section 3.2. A serious drawback however is that the scaling factor $\prod_{i=0}^{p(m)}\cos(m^{½}\sigma_i\alpha_{m,i})$ (eq 2.10) will then depend on σ_i. This problem can be solved by either repeating the iteration serially[Tag87] (one for σ=-1 and one for σ=+1), or performing two iterations in parallel [Dup89]. See also page 11.

inputs/outpus are used, (σ_i), rotations can be initiated immediately after the first step in the vectorization has been completed.

Most of the drawbacks can be overcome by using a slightly more complex representation of the term "$m^{½}\tan(m^{½}\alpha_{i,m})$". Namely,

$$m^{½}\tan(m^{½}\alpha_{i,m}) = 2^{-S_{m,i}} - \eta_{m,i}2^{-S_{m,i}'}, \qquad (2.12)$$

where $\eta_{m,i}$ is either -1,0, or 1 for i=0,1,..,p. This modification allows us to impose the following constraints on the CORDIC algorithm :

1. The number of iterations p+1 is independent of the coordinate system.
2. Accuracy and range conditions are the same for each coordinate system.
3. The normalization (scaling) factors $K_m = \prod_{i=0}^{p(m)}\cos(m^{½}\alpha_{m,i})$ are simple powers of 2 :

$$K_m = \prod_{i=0}^{p(m)}\cos(m^{½}\alpha_{m,i}) \approx 2^{-S(m)}. \qquad (2.13)$$

A complete set of basic angles $\alpha_{m,i}$, η's, scaling constants $K_{m,i}$ and shift factors $S_{m,i}$, $S_{m,i}'$ has been deduced for the case m=−1,+1 (circular/hyperbolic coordinate systems) with an absolute error smaller than 2^{-15} and an angle range of [−π,+π]. This table is depicted in table 2.1.

TABLE 2.1. Optimal CORDIC parameters for 16-bits accuracy

index i	S,S',η_{-1}	S,S',η_1	α_{-1}	α_1
1	0 4 -1	0 3 +1	1.716994	0.844154
2	1 8 -1	1 8 +1	0.544111	0.466768
3	1 6 -1	1 6 +1	0.528685	0.476069
4	2 14 -1	2 14 +1	0.255348	0.245036
5 .	2 4 -1	2 4 -1	0.189745	0.185348
6	4 6 +1	4 6 +1	0.078285	0.077967
7	4 10 -1	4 10 -1	0.061601	0.061446
8	5	5	0.031260	0.031240
9	6	6	0.015626	0.015624
10	7	7	0.007813	0.007812
11	8	8	0.003906	0.003906
12	9	9	0.001953	0.001953
13	10	10	0.000977	0.000977
14	11	11	0.000488	0.000488
15	12	12	0.000244	0.000244
16	13	13	0.000122	0.000122
17	14	14	0.000061	0.000061
18	15	15	0.000031	0.000031
19	16	16	0.000015	0.000015
K_{-1}≈4.0000058891≈2^2, K_1≈0.5000096618≈2^{-1}				

Note that for circular rotation, an additional initial rotation over $\alpha_{m,i} = \alpha_{1,0} = \pm½\pi$ is necessary, to cover the required angle range of [−π,+π]. This has not been depicted in the table.

The papmeters in the above table are slightly different than those bupIished in [Bu86] where the number of micro-rotation p(m)+1 was stilll different for circular and hyperbolic coordinate systems. The parameters given here have been given for the first time in [Lan88].

Notice that almost all shift exponents S and S′ are independent of the parameter m. The problem of finding such set of exponents is not trivial. It is an NP complete problem and, moreover, the solution space is a very small intersection of the two solution spaces, one for m=1 and one for m=-1. The solution given here has been obtained by means of a simplified exhaustive search. It is not the only solution although the number of solutions quickly decreases with increasing accuracy. Only one solution was found for a 32 bit accuracy. It is not clear whether such a solution (such a set of constraints) is optimal or not. In fact, the *VLSI implementation* of *the* algorithm, based on the data of table 2.1, is not optimal in terms of silicon area used. Notice also that for most of the micro-rotations (i=7,8,..p(m)) $\eta_{m,i}$ equals zero. This is not hard to understand, since for small basic angles the cordic becomes linear in α_m and

$$\cos(\alpha_{m,i}) = 1 - \frac{\alpha_{m,i}^2}{2} + O(\alpha_{m,i}^4) \qquad (2.14)$$
$$= 1 + O(2^{-24}), \qquad \text{for } \alpha_{m,i} < 2^{-12}.$$

This has several consequences, one being that the price to be payed for the "double rotations" is negligible, another being that increasing the accuracy (number of rotations) will only have a minor effect on the scaling factor. However, the price to be payed for one more bit of accuracy is a complete micro-rotation : the price is in the tail.

A detailed analysis of all different errors sources in our modified CORDIC algorithm is given in section 5. This will explain the need for additional micro-rotations to enhance the accuracy in hyperbolic rotations.

3. CORDIC Architectures

Many different architectures can be deduced for the CORDIC algorithm described in the previous section. These vary from bit-serial implementations to word parallel pipelined architectures. Which choice is made depends on the requirements for computing throughput and constraints that hold for area usage, latency and dissipation. At each level of abstraction of the CORDIC algorithm, such a tradeoff must be made. Two important levels of abstraction can be identified in the CORDIC algorithm :

1. *the micro-rotation level*
   This level describes the basic shift-add operation,
2. *the CORDIC top-level*
   This level describes the CORDIC operation as a sequence of micro-rotation operations.

3.1 Top Level CORDIC Architectures

Table 3.1 summarizes the main properties of some alternative top-level CORDIC architectures.

For any technology with a given T_c, table 3.1 can be used to

TABLE 3.1. Properties of Top Level CORDIC Architectures. T_c is the minimum clock period, T_μ is the delay of a micro-rotation, p+1 is the number of CORDIC iterations, L_μ is the latency of a micro-rotation (for pipelined CORDIC only) and area$^+$ indicates the additional amount of area due to multiplexers, rams, barrel-shifters, registers, wiring and control.

properties	Top Level CORDIC Implementations sequential	ripple	pipeline
delay (sec)	$2(p+1)\max(T_c,T_\mu)$	$(p+1)T_\mu$	$L_\mu(p+1)\max(T_c,T_\mu)$
thru-put (rot/sec)	$\frac{1}{2(p+1)\max(T_c,T_\mu)}$	$\frac{1}{(p+1)T_\mu}$	$\frac{1}{\max(T_c,\frac{T_\mu}{L_\mu})}$
latency (# cycles)	0	0	$L_\mu(p+1)$
area (# μRot's)	½	(p+1)	(p+1)
area$^+$ (# μRot's)	16	$\frac{(p+1)}{4}$	$\frac{(p+1)}{2}$

determine the optimal top-level CORDIC architecture.

If we compare the performance figures for the different architectural alternatives (table 3.1), it is clear that the pipelined implementation has the highest throughput, however at the cost of more hardware. The amount of extra hardware is less than would be expected, because a number of simplifications in both top-level and micro-rotation architecture are possible.

1. The full-pipeline implementation does not need a barrel-shifter for shift-factors, because the shifts can be hard-wired.
2. No memory for shift-factors and η_i parameters is needed, because these are constant for each different micro-rotation.
3. The terms $m_i\sigma_i\eta_{m,i}$ and $\sigma_i\eta_{m,i}$ (eq. 3.5–3.12) can be simplified since $\eta_{m,i}$ only depends on m for a given i, hence reduces the complexity of the control-unit of each micro-rotation (identified by i).
4. A double shift factor $2^{-S_{m,i}}$ can be omitted for a large number of micro-rotations (see eq. 3.5–3.12) and table 2.1) when $\eta_{m,i}$ is zero. This reduces the micro-rotation complexity with a factor 2.

We conclude that a full pipeline implementation leads to a regular architecture with very few control. A compromise such as the usage of a limited number of micro-rotations, which are executed sequentially, looses all the advantages listed above.

3.2 Micro-Rotation Architectures

At the next level down we identify the micro-rotation operation. This operation can be implemented by add/subtract devices, control unit and shifter. Figure 3.1 shows two structures of a micro-rotation that implement a micro rotation operation given by eq. 3.5–3.12 : one with $\eta_{m,i} \neq 0$ and the other with $\eta_{m,i} = 0$. Both structures can be used in a ripple or pipeline CORDIC implementation.

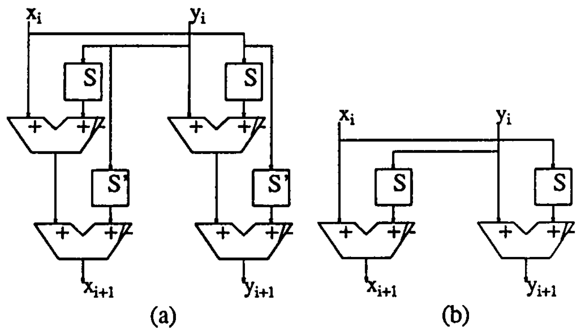

Figure 3.1. Micro-Rotation Structures for (a) $\eta_{m,i} \neq 0$ and (b) $\eta_{m,i} = 0$

High throughput with at the same time a limited latency and area occupation is a very important property of the CORDIC processor we want to design. We are therefore most interested in fast parallel add/subtract devices as far as area usage and dissipation do not create a problem.

Figure 3.2 shows the architecture of a parallel pipeline micro-rotation device for $\eta_{m,i} = 0$. The shift factors are hardwired and therefore not shown.

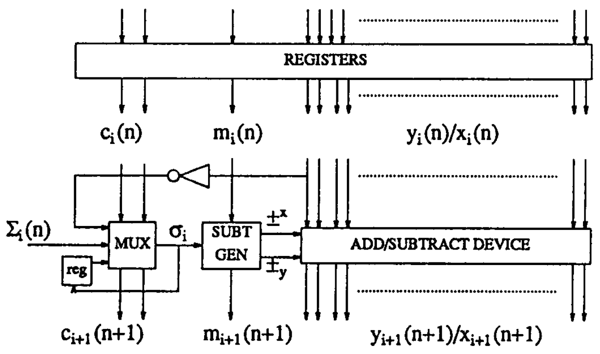

Figure 3.2. Architecture of a Word Parallel Pipeline Single Section Micro-Rotation. Σ_i is the encoded angle input signal, $c_i(n)$ the control signal that selects the input σ_i : $\Sigma_i(n)$, the previous σ_i : $\sigma_i(n-1)$, or the sign of y_i. The other signals are as defined in section 2

The best known fast devices are the carry-select adder/subtractor, the carry-look ahead adder/subtractor and its dynamic implementation in CMOS called the manchester chain adder/subtractor. See pages 322-325 of Weste and Eshraghian[Wes85]. Another very fast addition/subtraction device that is worth considering is the 'Binary Redundant' device See e.g.[Tak85]. A comparison of the transistor-count and gate-delay between the different types of add devices is depicted in table 3.2. The data for this table is obtained from Weste[Wes85] and Takagi[Tak85]. The numbers depicted are approximate figures for devices in static and dynamic CMOS. The extension to subtract devices involves only a small O(n) number of extra gates.

TABLE 3.2. Comparison of n-bit Adder Devices, consisting of k-bit digits

name	static		dynamic	
	# transistors	carry delay (# gates)	# transistors	carry delay (#gates)
carry ripple	28 n	2 n	22 n	2 n
carry save	34 n	2 k	28 n	2 k
carry select	28 (2 n - k)	2 k + 2 n / k	22 (2 n - k)	2 k + 2 n / k
static carry look ahead	$< 38 \cdot \frac{4}{3} n$ †	$2 \cdot {}^4\log(n)$	$< 32 \cdot \frac{4}{3} n$ †	$2 \cdot {}^4\log(n)$
manchester chain	-	-	30 n	$\frac{n}{2}$
binary redundant	116 n	3	103 n	3

Discussion.
Before going into the pros and cons of the different add/subtract devices, the following remark is in order. The CORDIC algorithm presented in this paper is nearly regular, meaning that it can be easily generated using repetition of (parametrized) bit-level as well micro-rotor level units. This means that recently proposed *systolic* or *Quasi-systolic* architecture design methodologies can be invoked to design such algorithms (and their implementations) as the CORDIC algorithm. See [Kun88, Dep91]. In this context, the implementation of the CORDIC algorithm using adders "such" or "so" is simply a matter of degree of systolization. See e.g., [McC90]. Therefore, the usage of a particular full-adder in the VLSI implementation of the algorithm is just a parameter in the design process. The implementation considered here uses carry-ripple adders only because such adders have been available in the cell library and not because of architecture-design level considerations. Of course, the performance and price of the ultimate implementation will depend on the particular adders used. We therefore include the following additional remarks.

- *carry-ripple*
  The slowest device is the carry-ripple adder, however its area consumption is low. Even more important is its simple and regular structure.
- *carry-save*
  The carry save adder has relatively low area consumption, paired with a high throughput. The main drawback of such a device, is its high latency. In the CORDIC algorithm, the latency will be increased with a factor n/k. This is undesirable in some filtering applications which require low computation latency[Dep89]. However, in case of a high computation latency, several filter applications can be executed in pipeline on the same CORDIC processor.
- *carry-select*
  The carry-select adder is a fast device, but occupies nearly twice as much area. Application of this device in a micro-rotation can be considered in case small latency and high speed is paramount, while area consumption is of less importance.
- *carry look ahead*
  The area and delay of this device are logarithmically proportional to the number of bits. A drawback of this device is its less regular structure. This causes lower area efficiency of VLSI implementations, and complicates the task of constructing automatic module generators for this device (this is necessary if different accuracy requirements must be fulfilled). If only one level of carry-lookahead sections is used, the delay is equivalent to the delay of the manchester-chain, but the hardware of the device is reduced (30 % smaller for large devices : n > 32) and its structure is more regular.
- *manchester chain*
  This device is a more compact implementation of the carry-lookahead device. It is a dynamic implementation. This results in smaller area and its architecture is more regular, but the cascading of several sections cannot be done without clocking. However the applied pre/discharging scheme allows correct carry ripple propagation. It can be equipped with several levels of carry-lookahead, but this complicates the design of error free dynamic behavior. Like all dynamic devices, it must be designed carefully to ensure correct operation under all circuit conditions.
- *binary-redundant adder*
  Evidently, the fastest device is the binary-redundant adder. A binary redundant adder is a device that can perform additions in constant time, because there is no carry ripple. This is possible since two bits are used to encode one redundant bit, hence is sufficient to retain both sum and carry bit or an encoded version of these two.
  The area consumption of such a device is however very high. This is not only due to the larger basic add-cells, but also because of the additional wiring (twice as much, because each redundant bit consists of 2 standard bits). Finally, an area consuming conversion unit from binary redundant to standard two's complement encoding is necessary.

An additional problem arises in CORDIC vectoring operation. Here, the sign of the y-operand must be determined before the next addition/subtraction can take place. However, the sign of a binary redundant number cannot be determined in constant time, because there is no reserved sign bit. At best a O(log n) time complexity can be achieved. We can solve this problem for the CORDIC algorithm, by limiting the number of most significant bits (e.g. 1 to 4 bits) to be examined for sign determination. This means that the correct sign can not be determined if y_i has a too small magnitude. In this case however, vectorization (rotation over the next basic angle) is not necessary, because the accuracy of y_i will not be improved in this micro-rotation. In the following micro-rotation, a

† The number of bits that have carry look-ahead is 4 (k=4), because a larger number increases the complexity of the logic (and delay) dramatically. For 4 of such blocks (16-bits), a new block is added that performs carry lookahead for for the 16 th bit. This is repeated each time ${}^4\log(n)$ is integer. Hence the number of transistors = $38n\,(1 + 1/4 + 1/4^2 + \ldots) = 38\,n\,(1 + 1/3\,(1 - 1/4^m)) < 57\,n$. Here m is the number of levels of carry-lookahead devices are added = ceiling(${}^4\log(n) - 1$).

few bits of y_i of lower weight are examined to determine the sign. If once again the magnitude of y_i is too small to allow correct sign determination, again no vectorizing micro-rotation is necessary. However, if the sign of y_i can be determined by examining these few bits, a micro-rotation must be performed. This way, in each micro-rotation, a few number of different bits are examined : the first micro-rotation examines the most significant bits, the last micro-rotation the least significant bits. A fast determination of the sign of y_i was proposed in [Tag87]. It can be shown that it is possible to combine Tagaki's approach with ours to obtain both fast determination of y_i signs and simple scaling factors.

The final problem that remains, is that if no micro-rotation is performed, this corresponds to $\sigma_i = 0$, hence eq. 3.8/12/13 are no longer valid. I.e., the scaling factor K_m is no longer constant. This can be solved by performing both a positive and negative micro-rotation (namely, $\sigma_i = 1$ and $\sigma_i = -1$). For this, the micro-rotation hardware must be doubled. We have now obtained, a very fast device. However, the maximum clock frequency will be smaller than the rate at which an adder device can execute. The only way to to retain performance, is to execute several microrotations within the same clock period. This is obtained by implementing the CORDIC as a rippling device.

Considering the previous discussion, most devices are either too costly in area, or too slow. A reasonable alternative that allows a trade-off between latency and throughput is the carry-save adder. This device has a very regular and simple structure, hence can easily be described as a parameterized module. Instead of using a static or dynamic fulladder in a carry-save device, the application of a manchester chain device seems attractive. This is because it is 4 times faster than the fulladder, hence less pipeline stages are needed in the carry save device.

4. Floating Point Extension of the CORDIC Processor

In many numerical applications, it is required that the CORDIC processor can operate on numbers with a large dynamic range. E.g. simulations of a speech coding application [Jai86] demonstrate the need for a CORDIC processor with an accuracy of 16 bits and a dynamic range of $\pm 2^{\pm 16}$. We therefore need to make a floating point version of the fixed-point CORDIC algorithm of section 2. Roughly speaking there are two options.

1. *Local floating point normalization*
   Each micro-rotation performs a floating point shift-add/subtract operation and normalizes the result.
2. *global floating point normalization*
   The CORDIC algorithm remains fixed-point. Only at the inputs and outputs of the algorithm floating to fixed point and fixed to floating point conversions are done.

The first approach is very costly in terms of hardware and speed in case several pipelined micro-rotations are used. Moreover, this approach is not very useful, because every micro rotation performs an addition or subtraction operation, which requires equal weights for the exponents of both operands. Hence, prior to each micro-rotation the floating point normalization operation (placing the binary point before the most significant bit of the mantissa and correcting the exponent accordingly) performed by the previous micro rotation has to be reversed.

The equalization of exponents before an add/subtract operation results in loss of accuracy because some bits of one of the operands will be shifted out of the data path. This problem also occurs in the global floating point normalization case. This is significant for the case of hyperbolic rotations/vectorizations where large input vectors (both x and y are large) become subsequently smaller during each micro rotation (this case is treated in section 5). The cases where the loss of accuracy leads to serious problems are very rare. Even then the relative error still remains acceptably small : ($e^{-\pi}$, see section 5).

The opposite case, where rotations can give rise to overflow, is easily prevented with local floating point normalizations. However, if we know the maximum possible overflow that can occur in hyperbolic and circular rotations, we can extend the data path to account for this. All these considerations are described in more detail in section 5.

Floating Point Input Processor
The floating point input processor adapts the mantissa whose associated exponent is the smallest exponent of the x/y vectors according to the difference between the x/y exponent values, such that the resulting x/y mantissas have equal exponents. The largest exponent is outputted. n and m are the sizes of respectively the mantissa and exponent data path. The architecture of the floating point preprocessor is shown in figure 4.1.

Floating Point Output Processor
The floating point output processor takes care of the output scaling (factor K_m) and the selection of significant bits. n denotes the size of the external datapath, F is the number of overflow bits, m the datapath size of the exponents and K_m the scaling factor in the CORDIC algorithm (see eq. 4.13). The generic architecture of the unit is given in figure 4.2.

† We would like to thank one of the reviewers for bringing this to our attention.

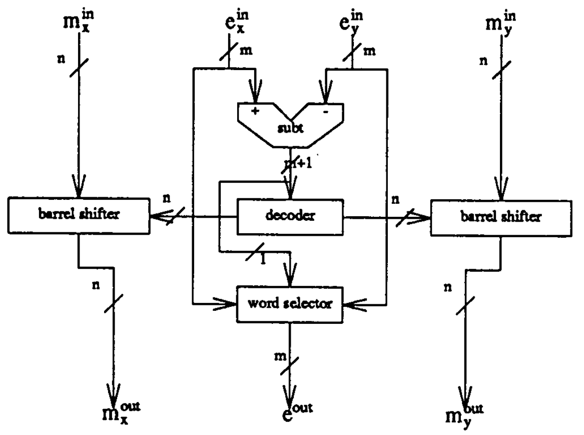

Figure 4.1. Floating Point Preprocessor Architecture. The parameter expressions n, m, m+1 indicate the number of bits of interconnections

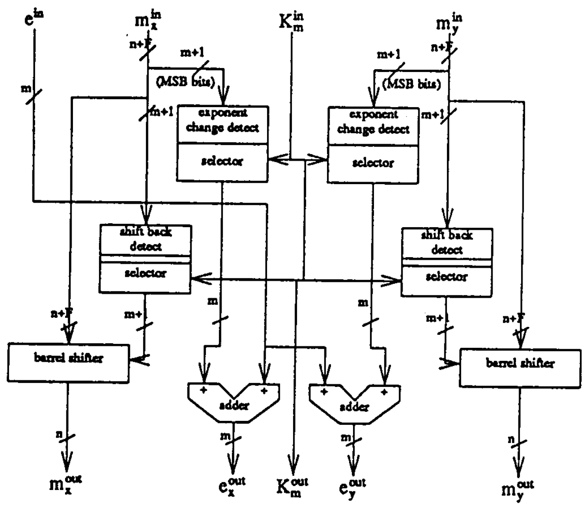

Figure 4.2. Generic Floating Point Postprocessor Structure

5. Accuracy Analysis of the Floating Point CORDIC Algorithm

In this section, the datapath widths of mantissas, exponents and control signals as present in floating point pre/post processors and CORDIC pipe are computed from the requirements for accuracy and dynamic range. The following items influence the computation accuracy.

5.1 Floating Point Input Normalization

The datapath size of the mantissa at the input and output of the floating point preprocessor are n-bits to obtain n bits (adder/subtractor) accuracy. In contrast to a floating point multiplier, no guaranteed accuracy improvement can be obtained by an extension of the internal datapath (micro-rotations perform add/subtract operations). See[Lac88].

5.2 Floating Point Output Normalization

The floating point postprocessor performs a "back-shift" of x and y mantissas. The maximum shift size is determined by the number of overflow bits in the internal datapath in the CORDIC pipe and the scaling factor K_m that is to be performed at the output for circular ($K_1 = 2^{-1}$) and hyperbolic ($K_{-1} = 2^2$) rotations. Circular rotations are norm preserving according to $x^2 + y^2 = C$, with $C \geq 0$ is constant, hence x and y are always bounded by $\sqrt{C}$. On the other hand, hyperbolic rotations are norm preserving according to $x^2 - y^2 = C$, with $C \in Z$ and constant. This implies that the magnitude of x and y can be very large in hyperbolic rotations. Because the magnitude of x and y depends on the angle α, the number of overflow bits is determined by the maximum rotation angle α_{max} in hyperbolic rotations. For this, we write out equation 5.2 for x' with $m = -1$ (hyperbolic coordinate system) and $\alpha = \alpha_{max}$:

$$\begin{aligned} x' &= \cos j\alpha_{max} \cdot x + \sin j\alpha_{max} \cdot y. \\ &= \underline{\cosh\alpha}_{max} \cdot x + \sinh\alpha_{max} \cdot x. \\ & j = \sqrt{-1}. \end{aligned}$$

Then

$$\begin{aligned} |x'| &= |x\cosh(\alpha_{max}) + y_{in}\sinh(\alpha_{max})| \\ &\leq \max(|x|,|y|)\, 2\cosh(\alpha_{max}). \end{aligned}$$

In the CORDIC algorithm that we have described in section 2, hyperbolic and circular rotations have the same domain for angles α. This is required, because a variety of algorithms (e.g. speech coding[Jai86]) apply both circular and hyperbolic and circular rotation/vectorization operations on the same (angle) data. In the circular case, an angle interval $[-\pi,+\pi]$ is sufficient to cover all possible rotations, hence we have chosen this angle range for both the circular and hyperbolic coordinate systems. The angle parameters of table 2.1 constitute an angle interval $|\alpha_{max}| = \pi + O(2^{-16})$, hence $2\cosh(\alpha_{max}) < 2^5$, which means that a 5 bits overflow extension is sufficient. In case of hyperbolic vectoring, the angle α of an input vector (x, y) may be larger than $|\alpha_{max}|$. Then, this vector is rotated to the x-axes over an angle α_{max}, which introduces a relative error of $e^{-\pi}$ [Lan88].

The output scaling factor for hyperbolic rotations is : $K_{-1} \approx 2^2$ and is performed in the floating point postprocessor. Therefore, inside the fixed point CORDIC pipeline, only (5 - K_{-1} =) 3 **MSB overflow bits** are needed, while 2 **LSB extension bits** are needed to prevent the loss of accuracy due to the multiplication factor of $K_{-1} = 2^2$ in hyperbolic rotations.

5.3 Number of Micro Rotations

In the linear part of the CORDIC algorithm (see table 2.1) only single shift/add operations are performed. In this part of the pipe, the angles have become small, hence the micro-rotations can be approximated by the following equations (see eq. 5.9 and 5.8) :

$x_{i+1} = x_i - m\alpha_i y_i, \quad y_{i+1} = y_i + m\alpha_i x_i$

When $\alpha_i < 2^{-(n-1)}$ (n is the number of bits of the input/output mantissas), then the terms $m\alpha_i x_i$ and $m\alpha_i y_i$ can be neglected. This is because the selection of the n most significant bits from the mantissa x_i/y_i results the same string of bits as the selection of the n most significant bits from the mantissa x_{i+1}/y_{i+1}. If we denote the selection of the bits 0 to n-1 of a binary word z by z[0..n-1], then

$x_{i+1}[0..n-1] = x_i[0..n-1]$, and

$y_{i+1}[0..n-1] = y_i[0..n-1]$, because

$2^{-(n-1)} x_i$ and $2^{-(n-1)} y_i$ are the LSB (and bits of lower weight) of x_i and y_i.

Therefore, the angle in the last useful micro-rotation satisfies : $\alpha_i = 2^{-(n-1)}$. However, the hyperbolic scaling factor $K_{-1} = 2^2$ propagates the error 2 bits towards the MSB, hence we require for the smallest angle α_i^{min} : $\alpha_i^{min} = 2^{-(n-1)-2} = 2^{-(n+1)}$. To reduce the cost of area we choose $\alpha_i^{min} = 2^{-n}$. This introduces a relative error of $2^{-(n-1)}$ for x_i/y_i numbers whose MSB equals one (negative number in Two's Complement), and an error of 2^{-n} otherwise.

5.4 Errors due to the Scaling Factors

In our CORDIC algorithm, the scaling factor K_m was determined for a total of p+1 = 20 micro rotors (see table 2.1), and an angle resolution of 2^{-n} (S = n). For hyperbolic scaling then holds :

$m = -1$, and $K_m = 4.0000058891 \approx 4 + 2^{-(n+1)}$, where n = 16

and for circular scaling :

$m = 1$, and $K_m = 0.5000096618 \approx 0.5 + 2^{-(n+1)}$

The normalization factor K_m (see eq. 5.13) hence introduces a relative error of 2^{-n}.

5.5 Accumulation of Truncation Errors

Every step in the CORDIC algorithm truncates the binary representation of the numbers at the least significant bit (LSB). The average error is equal to ½ the weight of the bit. For the total of p + 1 = 20 micro rotors, 26 shift/add operations are performed to achieve an angle resolution of 2^{-n} (Smax = n). Then the average error may have propagated over 4 bits towards the MSB, because $2^3 < ½\,26 < 2^4$. Therefore, another **4 bits must be added** to the data path. The total amount of bits in the CORDIC pipe now reaches 16 + 2 + 3 + 4 = 25, 21 of which are accurate at the output. Figure 5.1 shows a simplified schematic for the floating point pipeline CORDIC architecture.

Detailed information considering the circuit and layout design of the CORDIC processor can be found in[Hoe88].

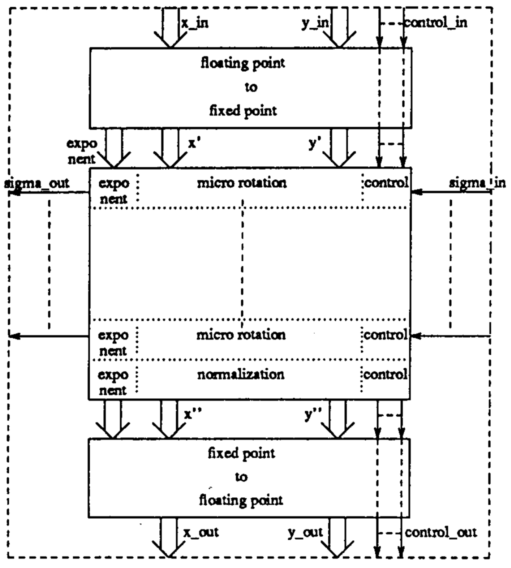

Figure 5.1. Floating Point Pipeline CORDIC Architecture

6. Implementation Results

A 21 bits floating point pipeline CORDIC processor has been designed and manufactured in a 1.5 μ CMOS process. It was originally designed in a 3 μ CMOS process[Phi82] and was later scaled down by a factor 0.72 to fit on a 1.2cm² chip. The CORDIC algorithm has been implemented as a pipeline of micro-rotations, while the micro-rotations consist of carry-ripple adders/subtractors (latency is zero), 2 → 1 multiplexers, hardwired shifts, word-pipeline registers and a small control unit that takes care of σ_i selection and coordinate system selection. See figure 3.2. Furthermore, floating point input and output convertors, a handshake module and clock buffers are integrated on the chip. The chip has a hold mode, in which the current state of pipeline registers is frozen, a scan-test mode, in which test vectors can be shifted in and out of the pipeline registers via the sigma-input/output pins, and a synchronous/asynchronous operation mode. The features of the chip are depicted in table 6.1.

TABLE 6.1. Implementation Results

accuracy	range	thru-put	latency	transistors
16-bits	$\pm 2^{16}$	5×10^6 rot/sec	4.4μs	70 000
# inputs	# outputs	dissipation	size	process
71	69	0.13 Watt	1.2cm²	CMOS(1.5μ)

7. Conclusions

In this paper we have described a novel CORDIC algorithm and processor. It is new with respect to its algorithm which is optimized to allow for both circular and hyperbolic rotations with low complexity, both in terms of software implementation and harware implementation. Moreover it is a very efficient algorithm when used in applications in which large amount of rotation and vectorizations are used, in any possible coordinate system and in any order. The required storage and/or silicon area is low and the execution time is independent of the particular operation performed. Another new feature of our CORDIC design is its pipelined architecture and floating point extension. It is angle pipelinable at the bit-level and has an execution time which is independent of any possible operation that can be executed. Its complexity is almost unaffected by the fact that a set of functions are implemented instead of just one. We are currently re-designing our floating point pipeline CORDIC to provide a professional very fast, small and high accuracy CORDIC core for signal processing applications. The novel design will be considerably smaller, mainly because the area consuming linear part of the cordic function can be implemented in a way which is much more efficient than has been done in our prototype CORDIC which is merely a straightforward map of the VLSI algorithm on silicon.

This architecture will enables the (real-time) application of CORDIC arithmetic in 2-dimensional high-speed systolic/wavefront arrays for parallel/pipelined signal processing algorithms and matrix computation applications. For an overview of applications see[Dep90]

References

Ahm85. H.M. Ahmed, "Alternative arithmetic unit architectures for VLSI digital signal processors", in *VLSI and modern signal processing*, ed. S.Y. Kung et.aL., Prentice Hall, Inc., Englewood Cliffs, NJ (1985).

Bu86. J.C. Bu, E.F. Deprettere, and F. (A.A.J.) de Lange, "On the Optimization of a Pipelined Silicon Cordic Algorithm", *Proceedings European Signal Processing Conference*, (2) pp. 1227-1230 (September 1986).

Coc72. D.S. Cochran, "Algorithms and Accuracy in the HP-35", *Hewlett-Packard Journal* 10(11)(1972).

Dep84. E.F. Deprettere, P. Dewilde, and R. Udo, "Pipelined cordic architectures for fast VLSI filtering and array processing", *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, pp. 41A6.1-41A6.4 (March 1984).

Dep89. E.F. Deprettere and P.M. Dewilde, "Orthogonal Filter Design and VLSI Implementation", *Proc. International Conference on Computer Systems and Signal Processing*, pp. 779-790 (Dec. 1989).

Dep90. E.F. Deprettere and A.A.J. de Lange, "The Synthesis and Implementation of Signal Processing Application Specific VLSI CORDIC Arrays", *proc. International Symposium on Circuits and Signal Processing (ISCAS)*, (May 1-3. 1990).

Dep91. Ed F. Deprettere and P. Dewilde, "Architectural synthesis of large, nearly regular algorithms: design trajectory and environment", *Annales des telecommunications*, p. (to appear) (1991).

Dep89. J. Bu and E.F. Deprettere, "A VLSI Architecture for High Speed Radiative Transfer 3D Image Synthesis", *the VISUAL COMPUTER*, (5) pp. 121-133 (1989).

Dup89. Jean Duprat and Jean-Michel Muller, "The Cordic Algorithm : new results for fast VLSI implementation", Internal Report, CNRS, Laboratoire LIP-IMAG 69364 Lyon Cedex 07, France (1989).

Hav70. G.L. Haviland and A.A. Tuszynski, "A CORDIC Arithmetic Processor Chip", *IEEE trans. Computers Vol. C-29(2)* , pp. 68-79 (1970).

Hoe88. A.J. van der Hoeven and A.A.J. de Lange, "Synthesis and Verification of the Pipelined Floating Point Cordic Processor", pp. 109-125 in *Lecture Notes of the Nelsis Project*, ed. O.E. Hermann and B.J.F. van Beijnum, University Twente, Enschede (March 9-11 1988).

Hwa79. Kai Hwang, "Computer Arithmetic, Principles, Architecture and Design", *John Wiley & Sons*, (1979).

Jai86. K. Jainandunsing and E.Deprettere, "Design and VLSI Implementation of a Concurrent Solver for N Coupled Least-Squares Fitting Problems", *IEEE journal on SELECTED AREAS IN COMMUNICATIONS* , pp. 39-48 (Jan. 1986).

Jai89. K. Jainandunsing and E.F. Deprettere, "A New Class of Highly Structured Algorithms for Solving Systems of Linear Equations", *SIAM journal on Scient. and Stat. Computations*, pp. 880-912 (September 1989).

Kun88. S.Y. Kung, *VLSI Array Processors*, Prentice-Hall International, Englewood-Cliffs, NJ 07632 (1988).

Lac88. Arild Lacroix, "Floating-Point Signal Processing - Arithmetic, Roundoff-Noise, and Limit Cycles", *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, pp. 2023-2030 (1988).

Lan88. A.A.J. de Lange, A.J. van der Hoeven, E.F Deprettere, and J. Bu, "An Optimal Floating Point Pipelined CMOS CORDIC Processor", *Proceedings International Symposium on Circuits and Systems (ISCAS)*, pp. 2043-2047 (June, 1988).

McC90. J.V. McCanny, J.G. McWhirter, and S.-Y. Kung, "The Use of Data Dependence Graphs in the Design of Bit-Level Systolic Arrays", *IEEE Transactions Acoustics, Speech, Signal Processing* ASSP-38(5) pp. 787-793 (May 1990).

Mul85. J.M. Muller, "Discrete Basis and Computation of Elementary Functions", *IEEE Transactions on Computers* C-34(9) pp. 857-862 (Sept. 1985).

Nav83. R. Nave, "Implementation of Transcendental Functions on a Numeric Processor", *Microprocessing and Microprogramming* 11 pp. 221-225 (1983).

Phi82. Philips, "Process Description - Philips C5th", USER's MANUAL, Philips Research Laboratries, Eindhoven (1982).

Sch83. C.W. Schelin, "Calculator Function Approximation", *American Mathematical Monthly*, pp. 317-325 (May 1983).

Tag87. N. Tagaki, T. Asada, and S. Yajima, "A hardware algorithm for computing sine and cosine using redundant binary representation", *Systems and Computers in Japan* 18(8) pp. 1-9 (Aug. 1987).

Tak85. Naofumi Takagi, Hiroto Yasuura, and Shuzo Yajima, "High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree", *IEEE Transactions on Computers* c-34(9) pp. 789-796 (September 1985).

Vee88. Alle-Jan van der Veen and Ed F. Deprettere, "A parallel VLSI direction finding algorithm", *Proc. SPIE Conf. on Advanced Algorithms & Architecures* III(975) pp. 289-299 (1988).

Vol59. J.E. Volder, "The CORDIC trigonometric computing technique", *IRE Trans. Electronic Computers* EC-8 pp. 330-334 (Sep. 1959).

Wal71. J.S. Walther, "An Unified Algorithm for Elementary Functions", *Proc. Spring Joint Computer Conference Vol. 38* , p. 397 AFIPS press, (1971).

Wes85. Neil Weste and Kamran Eshraghian, *Principles of CMOS VLSI DESIGN, A Systems Perspective*, Addison Wesley Publishing Company (1985).

Wil89. F. Williams, "The CORDIC Algorithm - Cast in Silicon", *Electronic Engineering*, pp. 47-50 (Sept. 1989).