Double Step Branching CORDIC : A New Algorithm for Fast Sine and Cosine Generation

Dhananjay S. Phatak
Electrical Engineering Department
State University of New York, Binghamton, NY 13902-6000

(*IEEE Transactions on Computers, vol 47, No. 5, May 1998, pp 587–602.*

Algorithm and Architecture being patented;
Application number 09/287,281, filing date 04/07/99.)

ABSTRACT

Duprat and Muller [1] introduced the ingenious "Branching CORDIC" algorithm. It enables a fast implementation of CORDIC algorithm using signed digits and requires a constant normalization factor. The speedup is achieved by performing two basic CORDIC rotations in parallel in two separate modules. In their method, both modules perform identical computation except when the algorithm is in a "branching" [1].

We have improved the algorithm and show that it is possible to perform two circular mode rotations in a single step, with little additional hardware. In our method, both modules perform distinct computations at each step which leads to a better utilization of the hardware and the possibility of further speedup over the original method. Architectures for VLSI implementation of our algorithm are discussed.

Index Terms : Double Step, Branching CORDIC, Constant Scale Factor, Redundant Signed-Digit CORDIC

I Introduction

The CORDIC algorithm was introduced by Volder [2] for fast computations of trigonometric functions and their inverses. In the classic reference [2], he also showed that the CORDIC method can be used for multiplication/division, as well as for conversion between binary and mixed radix number systems. Walther then demonstrated a "unified" CORDIC algorithm [3] that could be used to calculate trigonometric, exponential and square root functions along with their inverses; as well as to carry out multiply/divide operations. In essence, the CORDIC method evaluates elementary functions merely by table-look-up, shift and add operations. A small number (of the order of n, where n bits of precision is required in the evaluation of the functions) of pre-calculated fixed constants is all that is required to be stored in the look-up table. The CORDIC algorithm has nice geometrical interpretations [2, 3]: trigonometric, exponential, multiply functions are evaluated via rotations in the circular, hyperbolic and linear coordinate systems, respectively. Their inverses (i.e., inverse trigonometric functions, logarithm and division) can be implemented in a "vectoring" mode in the appropriate coordinate system.

Since its inception, CORDIC methods have been investigated by many researchers and have been employed in software and/or hardware for a variety of applications such as performing

(i) Fourier and related Transforms (FFT/DFT, Discrete Sine/Cosine Transforms, etc.) [4];
(ii) Householder Transformations [5], Singular Value Decomposition (SVD) and other Matrix Operations [6, 7];
(iii) Filtering and Array Processing [8, 9, 10].

Indeed CORDIC has evolved into a very rich and useful area. A good introduction to the subject can be found in [11]. For a fairly detailed list of CORDIC related references, tutorials, and simulation programs, the reader is referred to the CORDIC bibliography site [12] (http://devil.ece.utexas.edu).

In this paper we demonstrate a fast double-stepping method applicable in the CIRCULAR ROTATION mode (used to evaluate Sine/Cosine/Tangent) functions. We begin with an explanation of the circular rotations which are based on the iteration [1]

$$X_{i+1} = X_i - s_i\, Y_i 2^{-i} \tag{1}$$
$$Y_{i+1} = Y_i + s_i\, X_i 2^{-i} \tag{2}$$
$$Z_{i+1} = Z_i - s_i\, \arctan 2^{-i} \quad \text{where} \quad s_i \in \{-1, 0, +1\} \tag{3}$$

Using complex numbers, equations (1) and (2) can be rewritten as [11]

$$W_{i+1} = W_i \cdot (1 + j s_i 2^{-i}) \quad \text{where} \tag{4}$$
$$\text{complex variable} \quad W_i = (X_i + j \cdot Y_i) \quad \text{and } j = \sqrt{-1} \tag{5}$$

Using the polar form of complex numbers,

$$W_{i+1} = W_i \sqrt{1 + (s_i 2^{-i})^2} \cdot e^{j\theta_i} = W_i \cdot K_i \cdot e^{j\theta_i} \quad \text{where} \tag{6}$$
$$\theta_i = \arctan(s_i 2^{-i}) \quad \text{and} \quad K_i = \sqrt{1 + (s_i 2^{-i})^2} \tag{7}$$

Starting with W_0, if m iterations are carried out, then

$$W_m = W_0 \cdot K \cdot e^{j\theta} \quad \text{where} \tag{8}$$

$$K = \prod_{i=0}^{m-1} K_i = \prod_{i=0}^{m-1} \sqrt{1+(s_i 2^{-i})^2} \quad \text{and} \tag{9}$$

$$\theta = \sum_{i=0}^{m-1} \theta_i = \sum_{i=0}^{m-1} \arctan(s_i\, 2^{-i}) \tag{10}$$

If $s_i,\ i = 0, \cdots, m-1,$ are selected so that

$$\sum_{i=0}^{m-1} \arctan(s_i\, 2^{-i}) \longrightarrow \theta_0 \quad \text{then} \tag{11}$$

$$W_m \longrightarrow W_0 \cdot K(cos\theta_0 + jsin\theta_0) = (X_0 + jY_0) \cdot K(cos\theta_0 + jsin\theta_0) \quad \text{or} \tag{12}$$

$$X_m \longrightarrow K(X_0 \cos\theta_0 - Y_0 \sin\theta_0) \quad \text{and} \tag{13}$$

$$Y_m \longrightarrow K(X_0 \sin\theta_0 + Y_0 \cos\theta_0) \tag{14}$$

If the initial values X_0 and Y_0 are set to 1 and 0, respectively, then

$$X_m \longrightarrow K\cos\theta_0 \quad \text{and} \quad Y_m \longrightarrow K\sin\theta_0 \tag{15}$$

In general, the coefficients s_i at each step of the CORDIC iteration can take any of the three values $\{-1, 0, +1\}$. If $s_i = 0$ is allowed, then the scaling factor K is not a constant, but depends on the actual sequence of s_i values. On the other hand, if s_i can be restricted to ± 1, then K is a constant (since the number of iterations m that are to be executed for a given precision are known ahead of time).

$$\text{In this case, selecting } X_0 = \frac{1}{K} \quad \text{and} \quad Y_0 = 0 \quad \text{yields} \tag{16}$$

$$X_m \longrightarrow \cos\theta_0 \quad \text{and} \quad Y_m \longrightarrow \sin\theta_0 \tag{17}$$

This method falls under the category of "additive normalization" since the initial angle $Z_0 = \theta_0$ gets zeroed out (i.e., it has at least $m-2$ leading zeroes in a non-redundant representation, if m iterations are executed) by adding $\pm\arctan 2^{-i},\ i = 0, \cdots, (m-1)$. At step i if the residual angle $Z_i > 0$ then the algorithm selects $s_i = +1$ and if $Z_i < 0$ then $s_i = -1$ so that the *magnitude* of the residual angle is constantly being reduced toward zero. For this method to work, the initial angle must satisfy

$$|Z_0| \leq \sum_{k=0}^{\infty} \arctan 2^{-k} = 1.74328662 \tag{18}$$

This range covers all angles of practical interest since $\frac{\pi}{2} \approx 1.5708 < \sum_{k=0}^{\infty} \arctan 2^{-k}$.

With the introduction of (Redundant) Signed-Digit representations [11, 13, 14] the addition becomes carry-free, (i.e., the addition takes a small fixed amount of time, irrespective of the wordlength) thus offering a potential for significant speedup. To fully exploit the speed advantage gained by using signed digits, the sign detection of the residual angle also must be done in a constant (and small) time delay (note that the next action depends on whether the current residual angle is positive or negative). This in turn implies that only a fixed number of leading digits can be looked at to determine the sign of the residual angle. In most methods (for example, those in [1, 15]) a window of 3 (leading) digits turns out to be sufficient to determine the sign. At each iteration, the window shifts right by one digit position. If at least one of the digits in the window of interest is non-zero, the sign of the residual

angle can be determined to be ± 1. If the sign is +1, the next elementary angle ($\arctan 2^{-i}$ at step i) should be subtracted, if the sign is -1, the next elementary angle should be added. The problem occurs when all the digits in the window of interest are zero or in other words the residual angle has many leading zeroes, so that just by looking at the window of 3 (leading) digits, it is not possible to tell whether its sign is +1 or -1. Ideally, in this case, one should select $s_i = 0$ and neither add nor subtract the elemental angle for that step. However, if the coefficients s_i are restricted to $\{-1, +1\}$ (to render the scaling factor K to be a constant) then sign detection of the angle being zeroed becomes the bottleneck: it could require the examination of a large number of digits (possibly the entire word length). In essence, if the residual angle Z_i has a lot of leading zeroes, the decision whether to add or subtract $\arctan 2^{-i}$ can be made only after examining the first non-zero digit. In general this might imply scanning the entire word length, in which case the advantage due to constant time addition using signed digits is lost. Takagi, Asada and Yajima proposed two different methods [15] viz., the method of "double rotation" and the method of "correcting rotations" to overcome this problem. However, these methods need to perform extra rotations which makes them slow.

Duprat and Muller proposed the Branching CORDIC algorithm [1] that lets the coefficients s_i to be restricted to ± 1, without the need for extra rotations. This is achieved by performing two CORDIC rotations in parallel at each step and retaining the correct result at the end of each step. Thus, extra hardware is needed, but the speed improvement is significant. When the residual angle Z_i has a lot of leading zeroes (so that its sign cannot be determined by looking only at a fixed number of most significant digits), Duprat and Muller's algorithm initiates two sequences of computations, one assuming $s_i = +1$ and the other assuming $s_i = -1$. It might appear that this branching could in turn lead to further branchings down the line. The ingenuity of their method essentially lies in realizing that further branchings are not possible and that a branching either terminates eventually (with both computation sequences converging) or if it does not terminate till the end, then *both* the modules have the correct result (within the tolerance specified).

We demonstrate that the basic idea of executing two sequences of computation in parallel can be exploited even further, making it possible to determine two coefficients (s_i and s_{i+1}) at each step of the CORDIC iteration (hence the name Double Step Branching CORDIC). In fact, in the original method the two modules do different computations only when in a branching. Otherwise they perform identical computations, which means one of the modules is not being utilized at all, except in branching. We enhance the algorithm by making the modules perform distinct computations at each step and retaining the correct result (just as in the original method). The additional hardware overhead is minimal, while the hardware utilization is better and the potential speedup could be significant.

The rest of the paper is organized as follows. The following section presents the algorithm via flowcharts and explains motivations behind various procedures therein. It is intended to provide a good overall picture without getting into rigorous mathematics which is deferred to the following sections. Section III derives some identities that are repeatedly used later on in proving the convergence properties of the algorithm. Building on this background, the next section presents the proof of correctness and convergence properties of the algorithm. Section V discusses architectures to implement our algorithm in hardware. Section VI presents conclusions.

The author would like to point out that all the algorithms (the original algorithm of Duprat and Muller, as well as the double step algorithm) have been implemented in software. Correctness of the analytical proofs in this manuscript was double checked independently via *exhaustive* simulation of all possible initial angles with a word length of 16 bits that lie in the range specified by (18) (≈ 57125 cases). In addition, thousands of randomly generated initial angles in the range specified by equation (18) with word lengths up to 49 bits have been simulated and verified for correctness.

II The Double Step Branching CORDIC method

We perform two computations in parallel in two separate modules at every step. The modules are referred to as "Module α" and "Module β". The inputs and outputs of Module α (as well as any other variables/attributes associated with it) are designated with superscript α while those of Module β are designated with superscript β. The subscript indicates the step or iteration number. A variable without the module designator superscript (α or β) indicates the correct value of that variable at that step. (for instance Z_i without any superscripts refers to the correct residual angle at step i).

For lucidity and ease of understanding, we adopt the convention to label the steps from zero onwards, (i.e., step 0, step 1, ... etc.) where step i utilizes $\arctan 2^{-2i}$ and $\arctan 2^{-(2i+1)}$ to generate Z_{i+1} from Z_i. With this convention, step 0 generates Z_1^α and Z_1^β from initial angle Z_0 by using $\arctan 2^{-0}$ and $\arctan 2^{-1}$; step 1 generates Z_2^α and Z_2^β from Z_1 by using $\arctan 2^{-2}$ and $\arctan 2^{-3}$ and so on.

At step i, each module performs one of the four possible computations (by selecting the + or $-$ signs)

$$\text{Module } \alpha : Z_{i+1}^\alpha = Z_i^\alpha \pm \arctan 2^{-2i} \pm \arctan 2^{-(2i+1)} \tag{19}$$

$$\text{Module } \beta : Z_{i+1}^\beta = Z_i^\beta \pm \arctan 2^{-2i} \pm \arctan 2^{-(2i+1)} \tag{20}$$

Note that $\arctan 2^{-0}$ through $\arctan 2^{-(2i-1)}$ have **already** been used in prior steps $0, 1 \cdots, (i-1)$ and step i uses $\arctan 2^{-2i}$ and $\arctan 2^{-(2i+1)}$. If the algorithm is not in a branching, then the input residual angle at step i is the same for both modules, i.e., $Z_i^\alpha = Z_i^\beta = Z_i$. When the algorithm is in a branching $Z_i^\alpha \neq Z_i^\beta$.

To understand the main idea behind the algorithm, assume that at the end of step $i-1$ we are not in a branching, i.e., $Z_i^\alpha = Z_i^\beta = Z_i$. For the purpose of illustration, also assume that sign of Z_i can be unambiguously determined to be positive. In that case, $\arctan 2^{-2i}$ must be subtracted. In the original method of Duprat and Muller this subtraction is performed by both modules (i.e., one of the two modules is not utilized) and the sign is determined again to figure out whether to add or subtract the next angle, i.e., $\arctan 2^{-(2i+1)}$. In our method, the two modules available are put to full use as follows:

$$\text{Module } \alpha : Z_{i+1}^\alpha = Z_i - \arctan 2^{-2i} - \arctan 2^{-(2i+1)} \quad \text{operation abbreviated } \langle - \ - \rangle \tag{21}$$

$$\text{Module } \beta : Z_{i+1}^\beta = Z_i - \arctan 2^{-2i} + \arctan 2^{-(2i+1)} \quad \text{operation abbreviated } \langle - \ + \rangle \tag{22}$$

Now both modules determine the signs of Z_{i+1}^α and Z_{i+1}^β in parallel. The sign is determined by looking at a fixed number of leading digits of the residue. As a result, the sign can be ± 1 if at least one or

more of the digits (in the window of digits used to evaluate the sign) is non zero; otherwise the sign evaluates to 0 if all the digits in the window are 0. It should be noted that the sign of the most significant digit in the "window" is not necessarily the sign of the residue (rules for inferring the sign of the residue by looking at the digits in the current window are summarized in Table 1 and explained later in this section). The sign of residue Z_k is henceforth denoted as "Sign(Z_k)". Note that if Sign(Z_k) = 1, then $Z_k > 0$. Similarly if Sign(Z_k) = -1, then $Z_k < 0$. When Sign(Z_k) = 0, however, there is insufficient information to decide whether Z_k is positive or negative.

If Sign(Z^{α}_{i+1}) is positive, i.e., a $\langle - \ - \rangle$ operation still leaves a positive residue, then Module α is correct. In this case a $\langle - \ + \rangle$ operation that Module β performed must also yield a positive residue, which is higher in magnitude than the residue of Module α which is deemed to be correct and copied by Module β prior to the next iteration. Similarly if a $\langle - \ - \rangle$ operation leads to a "zero sign" (i.e., the residue Z^{α}_{i+1} has leading zeroes which implies it has a small magnitude), then that operation was the correct one. Finally when a $\langle - \ - \rangle$ operation leads to a negative residue, then the next action depends on the sign of the residue Z^{β}_{i+1} in the other module obtained by a $\langle - \ + \rangle$ operation: if Sign(Z^{β}_{i+1}) is -1 or 0 then it is correct, otherwise, both modules are correct and the algorithm enters a branching. This is summarized by the flowchart in Figure 2 (which is further explained a bit later). Similarly, the flowchart in Figure 3 summarizes the procedure when Sign(Z_i) = 0, and the flowchart in Figure 4 summarizes the procedure when the algorithm is in branching.

Note that the initial angle Z_0 must satisfy condition (18). Similarly, at the end of step $(i-1)$, having used $\{\arctan 2^0, \arctan 2^{-1}, \cdots, \arctan 2^{-(2i-2)}, \arctan 2^{-(2i-1)}\}$, the residue Z_i must satisfy

$$|Z_i| \leq \sum_{k=2i}^{\infty} \arctan 2^{-k} \tag{23}$$

If this condition is not satisfied, no combination of remaining angles $\{\arctan 2^{-2i}, \arctan 2^{-(2i+1)}, \cdots\}$ can force the residue magnitude arbitrarily close toward zero (as more iterations are executed). Thus, the magnitude bound specified by condition (23) must be satisfied at every step (of CORDIC algorithm) and is henceforth referred to as the "**tighter**" bound. If the algorithm is in a branching, then two possibilities are being tried and least one of the computation sequences must generate residual angles that satisfy the *"tighter"* bound .

Directly verifying condition (23) would require a magnitude comparison or a full subtraction with a delay which depends on the word length and would defeat the purpose behind using signed digits. The evaluation of the residue sign must be done by looking at a *fixed* number of leading digits to make the sign evaluation delay independent of the word length. In most methods (for example, those in [1] and [15]) 3 leading digits of the residue are examined to determine the sign. In our case, since we need to do a double step, it turns out that 6 digits need to be examined to determine the sign. This does not mean that the delay required to determine the sign is double (as compared with the methods that examine only 3 digits), because the 6 digits are divided into 2 sub groups of 3 digits each and each subgroup is handled separately, *in parallel* to generate the required signals. The control logic that integrates signals from these subgroups is slightly more complex than the case where only 3 digits are examined (this is explained in detail later on).

What makes it possible to look at only a fixed number of leading digits is the fact that at step *i* (i.e., having used $\{\arctan 2^0, \arctan 2^{-1}, \cdots, \arctan 2^{-(2i-2)}, \arctan 2^{-(2i-1)}\}$; when the sign is determined prior to using $\{\arctan 2^{-2i}$ and $\arctan 2^{-(2i+1)}\}$, both the residual angles Z_i^{α} and Z_i^{β} satisfy

$$\{|Z_i^{\alpha}|, |Z_i^{\beta}|\} \leq 3 \cdot 2^{-(2i-1)} \tag{24}$$

Note that $\sum_{k=2i}^{\infty} \arctan 2^{-k} < 2^{-(2i-1)} < 3 \cdot 2^{-(2i-1)}$, so that the bound specified by equation (24) is "**coarser**" than that of equation (23) above, and is referred to by that name throughout the rest of the manuscript. If this *"coarser"* bound is violated, then the sign that is evaluated can be incorrect, possibly leading to a wrong result at the end. Note the distinction between the *"tighter"* bound and *"coarser"* bound : only one of $\{Z_n^{\alpha}, Z_n^{\beta}\}$ needs to satisfy the *"tighter"* bound at all steps while both of them must satisfy the *"coarser"* bound , irrespective of whether or not the algorithm is in a branching.

With this background, we now present the algorithm and give the details of how the sign evaluation is done. Convergence properties of the algorithm (i.e., the fact that conditions (23) and (24) are satisfied at all steps) are proved in Section IV. The algorithm is summarized by the flowcharts in Figures 1, 2, 3 and 4.

Figure 1 shows the overall flowchart of the algorithm. In CORDIC, $n+1$ angles $(\arctan 2^0, \cdots, \arctan 2^{-n})$ need to be utilized for *n* bits of precision. This can be explained using the following identities (please refer to [1] for their derivation)

$$\arctan 2^{-n} < \sum_{k=n+1}^{\infty} \arctan 2^{-k} < 2^{-n} \tag{25}$$

$$\arctan 2^{-n} - \sum_{k=n+1}^{n+p} \arctan 2^{-k} \leq \sum_{k=n+p+1}^{\infty} \arctan 2^{-k} < \sum_{k=n+p+1}^{\infty} 2^{-k} = 2^{-(n+p)} \tag{26}$$

Equation (25) together with the fact that the residual angle in at least one of the two modules satisfies *"tighter"* bound at each step implies that $n+1$ angles mentioned above suffice to render the required precision. However, the algorithm might terminate in a branching in which case both modules satisfy the *"coarser"* bound . Using the *"coarser"* bound (instead of the *"tighter"* bound) indicates that $n+3$ angles $(\arctan 2^0, \cdots, \arctan 2^{-(n+2)})$ should be used to ensure that outputs of both modules satisfy the condition [(absolute value of residual angle) $< 2^{-n}$]. Since we use two angles at each step,

$$\text{number iterations required} = \lceil \frac{n+3}{2} \rceil \qquad \text{where } \lceil x \rceil = \text{smallest integer} \geq x \tag{27}$$

This is the loop-iteration-index appearing in Figure 1.

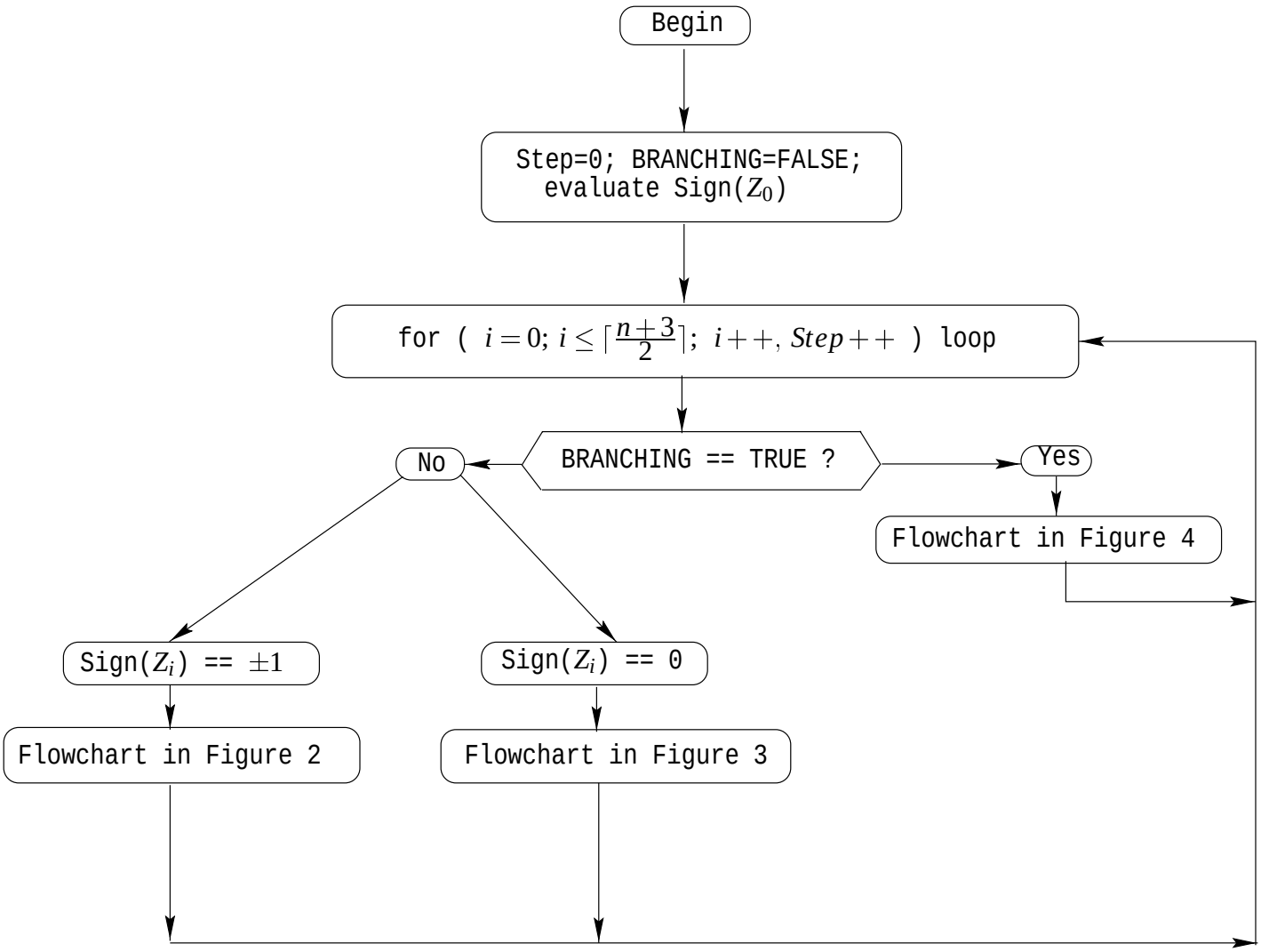

Figure 1 : Flowchart for the algorithm. Detailed flowcharts for specific cases when residue Sign evaluation returns ± 1, 0 and when the algorithm is in a branching are illustrated in Figures 2, 3 and 4, respectively.

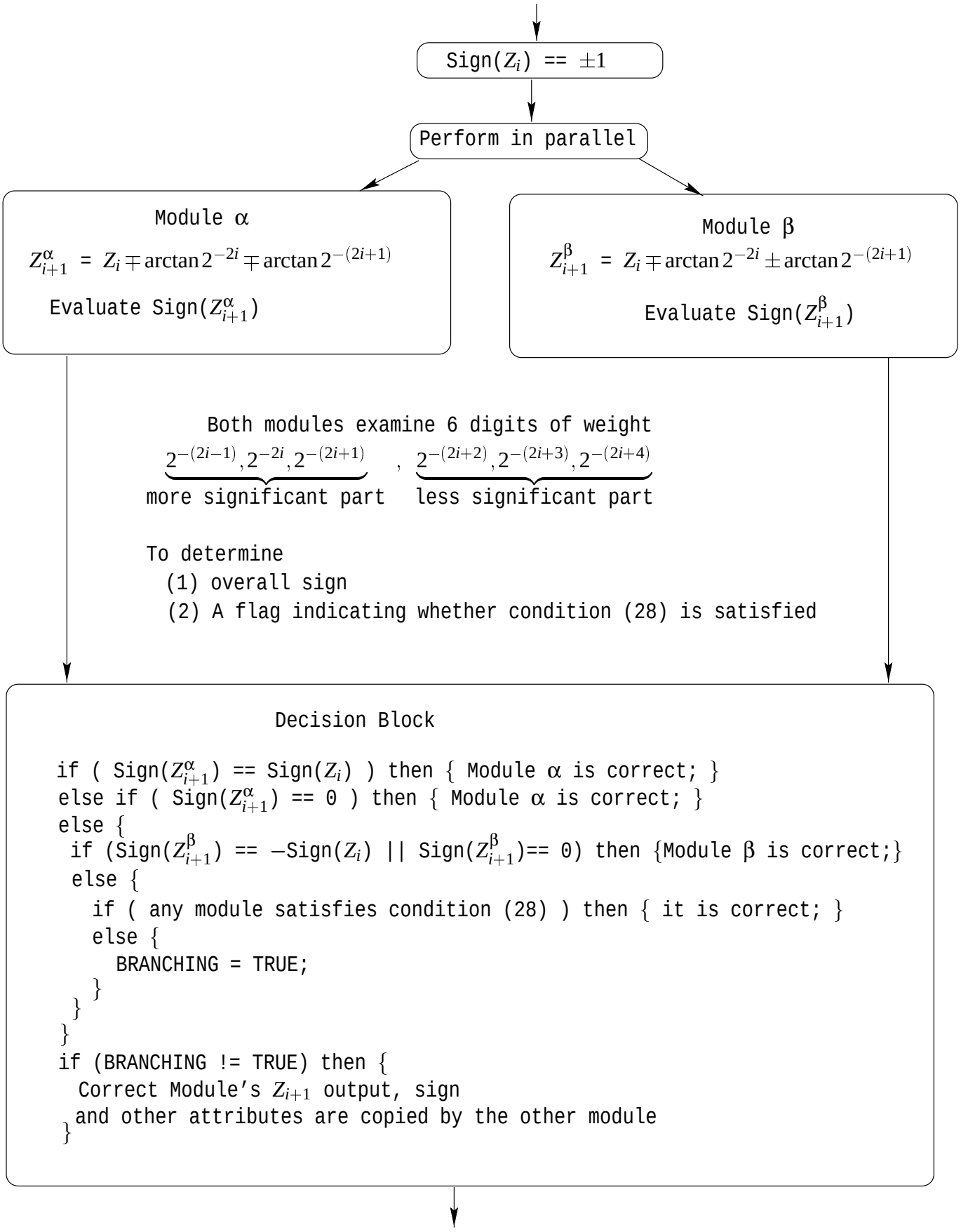

Figure 2 : Flowchart for step *i* for the case when residue Sign evaluation returns ± 1.

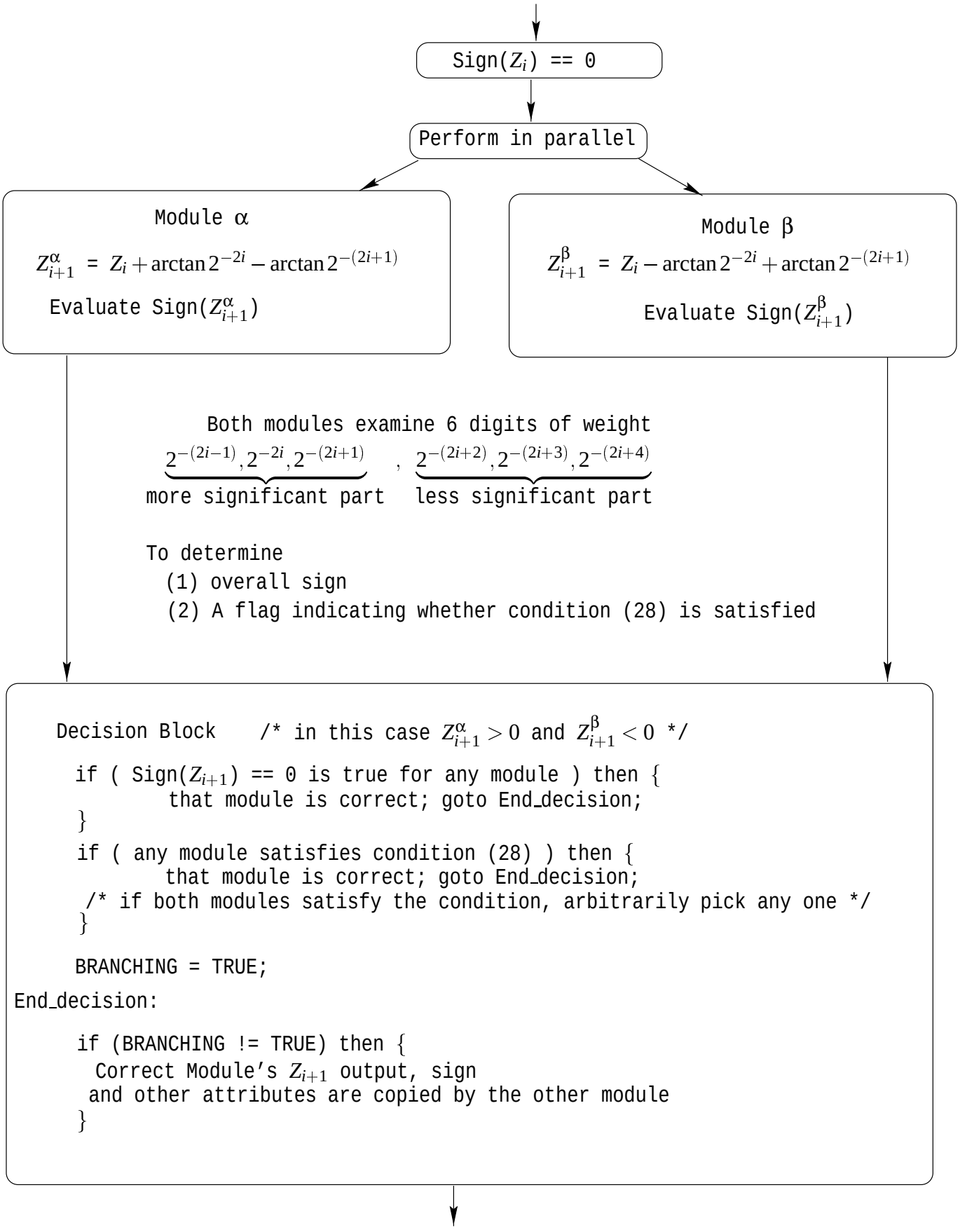

Figure 3 : Flowchart for step *i* for the case when residue Sign evaluation returns 0.

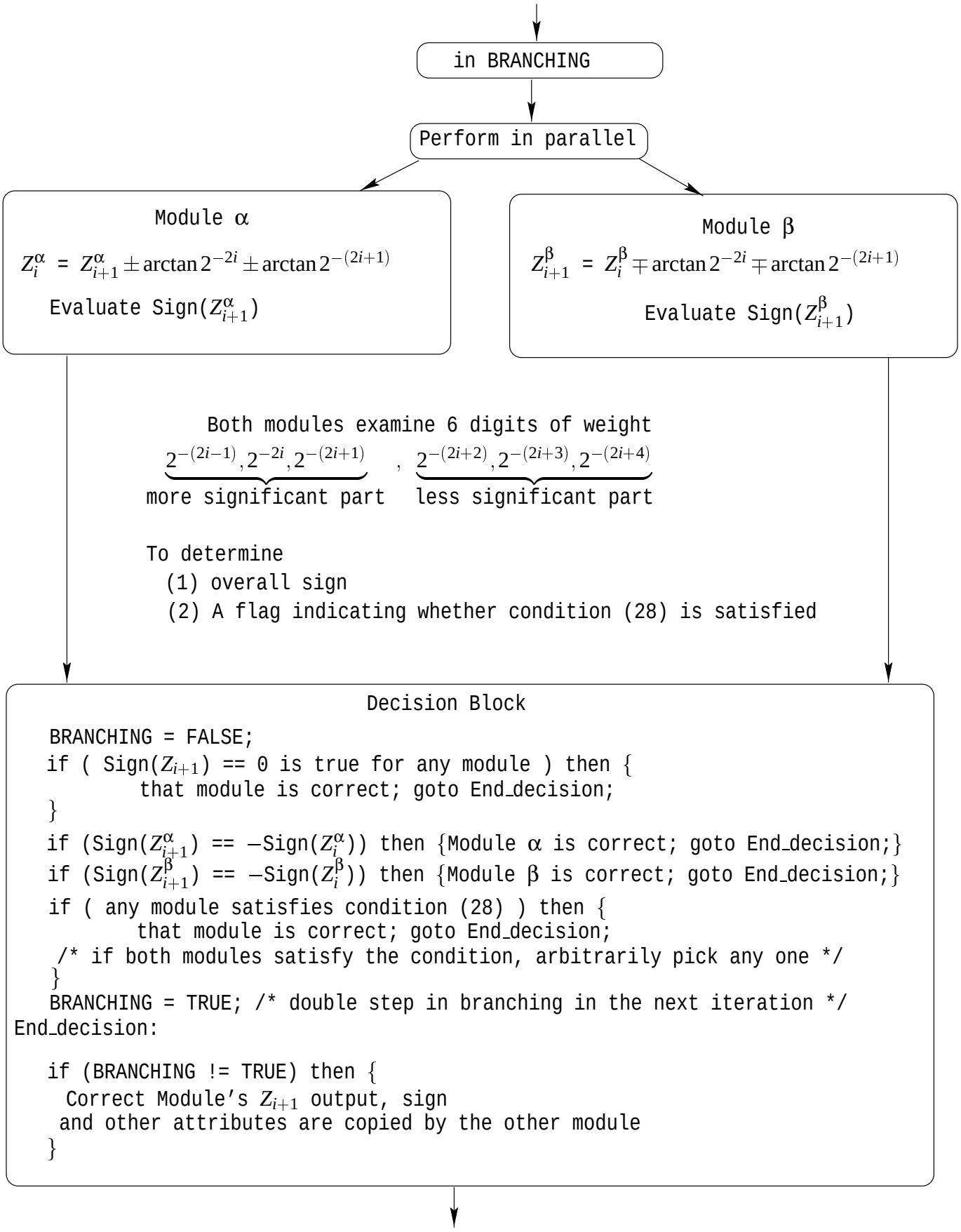

Figure 4 : Flowchart for step *i* when the algorithm is in a branching.

Figure 2 illustrates the procedure followed when the sign can be determined to be ± 1. In the blocks titled "Module α " and "Module β ", the upper and lower signs (in the stacks $\pm$ or $\mp$) correspond to Sign(Z_i) = +1 and Sign(Z_i) = -1, respectively.

Next, we explain the sign detection procedure. Because the residues always satisfy the *"coarser"* bound , digits of weight higher than $2^{-(2i-3)}$ need not be examined when determining the sign of Z_i^{α} (or Z_i^{β}, the sign detection operation is identical in both modules). This is a consequence of using signed digits to represent the residue. If only one angle is used at every step (as in Duprat and Muller's original method) then 3 digits of weights $2^{-(k-3)}, 2^{-(k-2)}, 2^{-(k-1)}$ suffice to evaluate the sign, **prior to using** $\arctan 2^{-k}$. Note that $\arctan 2^{-k}$ can have its leading "1" (i.e., its first non-zero digit) at bit position 2^{-k}. Hence, merely examining two digits of weight $2^{-(k-3)}, 2^{-(k-2)}$ does not suffice even in the original method. For example, consider the case when the three digits in question are 001. If a branching is initiated just by looking at the first two zeroes, one module would do an add and the other would do a subtract. In this case the module that did an add could exceed the *"coarser"* bound for the next step. That's why at least 3 digits are required.

In the double stepping method, evaluation of residue sign is done by examining a window of 6 digits as illustrated in Figures 5 and 6.

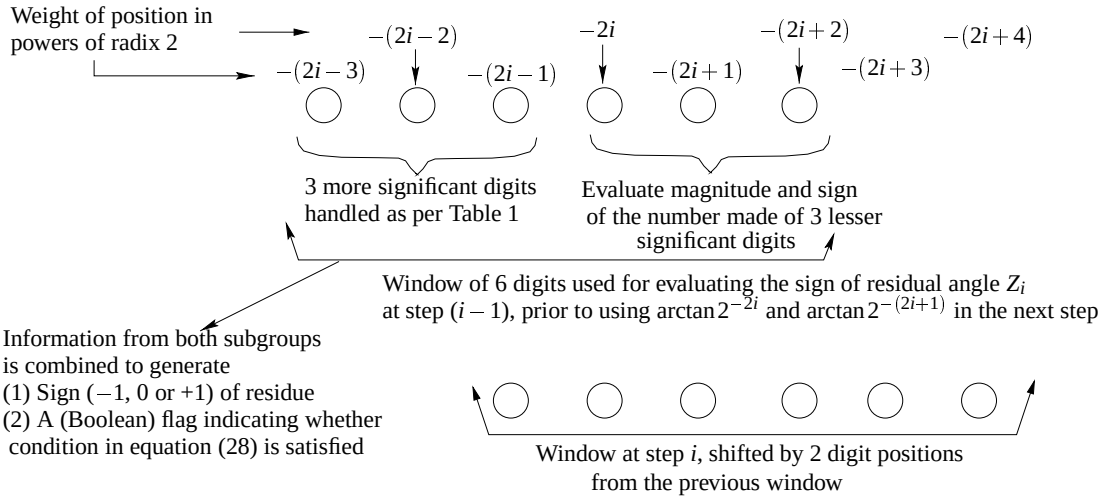

Figure 5: Window of 6 digits used to determine the sign of residue.

First of all, note that the 3 leading digits can be interpreted in a manner identical to the original method as indicated in Table 1. This table is derived from the fact that the residue satisfies *"coarser"* bound at all times.

In the flowcharts in Figures 2 and 3 if the sign of the residue can be determined to be +1 or -1 by examining only 3 (more significant) digits, then the lower 3 digits are inconsequential. If however, the leading 3 digits are zero, then there are 4 distinct possibilities for the double stepping: $\langle - \; - \rangle$; $\langle - \; + \rangle$; $\langle + \; - \rangle$; $\langle + \; + \rangle$. Note that double stepping implies $\arctan 2^{-(2i+1)}$ also gets used along with $\arctan 2^{-2i}$ (in fact they are not stored individually, only their sum and difference,i.e., $[\arctan 2^{-2i} + \arctan 2^{-(2i+1)}]$ and $[\arctan 2^{-2i} - \arctan 2^{-(2i+1)}]$ are stored in the look-up table). There is no "corrective" rotation (any back-tracking would defeat the purpose of trying to double step in the

first place), which implies that the four possibilities must be narrowed down to two, because only two modules are to be used. More digits need to be examined for this purpose; the natural question being "how many more digits ?"

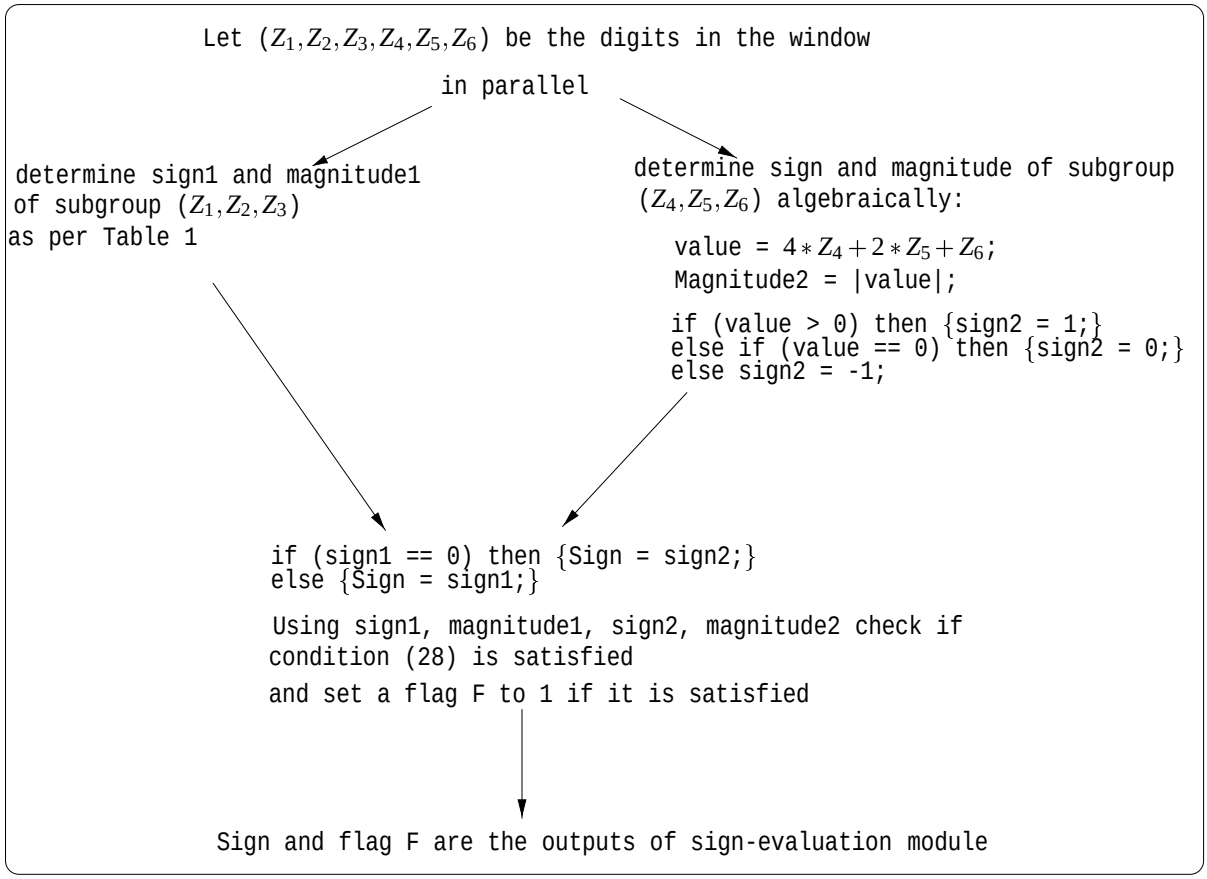

Figure 6: Sign detection procedure.

The answer turns out to be 3 extra digits, as indicated next. For the purpose of illustration, assume that the leading 3 digits in the window are all 0. Suppose that we start examining more digits one by one. Any time a non-zero digit (+1 or -1) is encountered, the sign of the residue is known and we can utilize the procedure in the flow chart in Figure 2 again. The problem happens when the following (few) digits are also 0 (which means that the *magnitude* of the residue is small). The question is: how many additional zero digits (following the 3 leading digits in the window that are all 0) must be encountered to ensure that $\langle - \ + \rangle$; $\langle + \ - \rangle$ are the only possibilities and rule out the other two, viz., $\langle - \ - \rangle$; $\langle + \ + \rangle$? The answer is 3 extra zeroes.

There is one more case that needs further scrutiny: when the algorithm is in a branching. In the original method, when in a branching, one module keeps adding (and the other keeps subtracting) the angles one at a time as long as the signs of next residues continue to be same as those of the previous ones (please refer to [1] for details). In the double stepping method, the analogous operation is adding (subtracting in the other module) two angles at a time. If the sign of the next residue remains the same as the previous one despite adding (subtracting) two angles, then that module is still indicating a continuation of branching.

Z_{2i-3}	Z_{2i-2}	Z_{2i-1}	Implied leading digit	Sign	Magnitude	comment
$\overline{1}$	$\overline{1}$	$\overline{1}$	1	+	1	
$\overline{1}$	$\overline{1}$	0	1	+	2	$\overline{1}$ denotes -1
$\overline{1}$	$\overline{1}$	1	1	+	3	
$\overline{1}$	0	$\overline{1}$	1	+	3	
$\overline{1}$	0	0	$\times$	$\times$	4	(impossible, violates *"coarser"* bound)
$\overline{1}$	0	1	$\overline{1}$	$-$	3	
$\overline{1}$	1	$\overline{1}$	$\overline{1}$	$-$	3	
$\overline{1}$	1	0	$\overline{1}$	$-$	2	
$\overline{1}$	1	1	$\overline{1}$	$-$	1	
0	$\overline{1}$	$\overline{1}$	$\overline{1}$	$-$	3	
0	$\overline{1}$	0	$\overline{1}$	$-$	2	
0	$\overline{1}$	1	$\overline{1}$	$-$	1	
0	0	$\overline{1}$	$\overline{1}$	$-$	1	
0	0	0	0	0	0	(need to look at next 3 digits)
0	0	1	1	$+$	1	
0	1	$\overline{1}$	1	$+$	1	
0	1	0	1	$+$	2	
0	1	1	1	$+$	3	
1	$\overline{1}$	$\overline{1}$	1	+	1	
1	$\overline{1}$	0	1	+	2	
1	$\overline{1}$	1	1	+	3	
1	0	$\overline{1}$	1	+	3	
1	0	0	$\times$	$\times$	4	(impossible, violates *"coarser"* bound)
1	0	1	$\overline{1}$	$-$	3	
1	1	$\overline{1}$	$\overline{1}$	$-$	3	
1	1	0	$\overline{1}$	$-$	2	
1	1	1	$\overline{1}$	$-$	1	

Table 1: Determination of Sign and Magnitude of the subgroup of 3 more significant digits at step $(i-1)$. A "$\times$" indicates a don't care, when the corresponding digit combination cannot occur in the residue.

However, if the sign does change, indicating a termination of branching, there is no quick way of telling whether
(i) the sign switched as a result of adding (subtracting) the first angle or
(ii) it remained the same after using the first angle and switched only after using the second angle.
If (i) happens to be true then the second angle has been used incorrectly. There is no way except to back track if this was allowed to happen. Hence, before taking a double step when in branching, we detect whether the first step itself would terminate the branching. The 6 digits in the window have all the information needed to make this prediction. As seen in the flow chart of the algorithm, a branching is terminated (i.e., not continued) if the following condition

$$\begin{aligned}&\texttt{[}\{\texttt{(sign of more significant part == 0) \&\& (magnitude of less sig. part} \leq C)\}\\ &\texttt{||}\{\texttt{(magnitude of more sig. part == 1) \&\& (magnitude of less sig. part} \geq (8-C)\texttt{)}\\ &\quad\texttt{\&\&(sign of less sig. part == }-\texttt{(sign of more sig. part))}\}\ \texttt{] == TRUE}\end{aligned} \tag{28}$$

is satisfied by any of the two modules, where

the constant C in equation (28) above can take only one of the two values 5 or 6
i.e., $C \in \{5,6\}$ (29)

Mathematically, the algorithm works correctly with both values of C. Whichever value leads to simpler circuits should be selected in the final VLSI implementation.

The above condition essentially terminates branching if

$$|Z_i^{\alpha}| \leq C \cdot 2^{-(2n+2)} \quad \text{or} \quad |Z_i^{\beta}| \leq C \cdot 2^{-(2n+2)} \tag{30}$$

If only one of Z_i^{α} or Z_i^{β} satisfies the above condition, then it is the correct residue (and the current branching terminates). If both Z_i^{α} or Z_i^{β} satisfy the above condition, then any of the two modules can be arbitrarily deemed to be correct and the current branching terminates (The proof of its correctness is in case II.1.a in Section IV).

Having presented the algorithm, we now prove its convergence. The next section derives some identities repeatedly used later on. Section IV presents the detailed proof of convergence.

III Background

In the following it is assumed that all indices are non-negative integers (i.e., $k,n,i,\cdots \geq 0$) unless stated otherwise. The most fundamental identities are (25) and (26) and are derived in [1]. The following relationship is also well-known [11]

$\arctan 2^{-(n+1)} > \frac{1}{2}\arctan 2^{-n}$ This relation immediately leads to the following (31)

useful identity: $\arctan 2^{-n} - \arctan 2^{-(n+1)} < \arctan 2^{-(n+1)}$ (32)

From the above, we drive the following identities

$$\sum_{k=n}^{\infty} \arctan 2^{-k} > 2^{-n} + 2^{-(n+1)} \tag{33}$$

$$\arctan 2^{-n} > 2^{-(n+1)} + 2^{-(n+2)} \tag{34}$$

$$2\arctan 2^{-n} > \sum_{k=n+1}^{\infty} \arctan 2^{-k} \tag{35}$$

Proof of 33 : By induction;

$$\text{base case : } \sum_{k=0}^{\infty} \arctan 2^{-k} = 1.74329 > 2^0 + 2^{-1} = 1.5 \tag{36}$$

$$\text{Assume (33) holds for } n = i. \text{ Then, } \quad \arctan 2^{-i} + \sum_{k=i+1}^{\infty} \arctan 2^{-k} > 2^{-i} + 2^{-(i+1)} \tag{37}$$

Invoking the first inequality in (25)

$$(\sum_{k=i+1}^{\infty} \arctan 2^{-k}) + (\sum_{k=i+1}^{\infty} \arctan 2^{-k}) > \arctan 2^{-i} + \sum_{k=i+1}^{\infty} \arctan 2^{-k} > 2^{-i} + 2^{-(i+1)} \tag{38}$$

Dividing both sides of the above inequality by 2, we get

$$\sum_{k=i+1}^{\infty} \arctan 2^{-k} > \frac{1}{2}[2^{-i} + 2^{-(i+1)}] = 2^{-(i+1)} + 2^{-(i+2)} \tag{39}$$

which shows that (33) holds for $(n+1)$ and completes the proof.

Proof of 34 : By induction;

$$\text{base case : } \arctan 2^{-0} = 0.78\cdots > 2^{-1}+2^{-2} = 0.75 \tag{40}$$

Assume (34) holds for *n*. Then dividing both sides by 2 and invoking (31)

$$\arctan 2^{-(n+1)} > \frac{1}{2}\arctan 2^{-n} > 2^{-(n+2)} + 2^{-(n+3)} \qquad \text{which completes the proof.} \tag{41}$$

Finally to prove (35), invoke (34) and (25):

$$2\arctan 2^{-n} > 2[2^{-(n+1)} + 2^{-(n+2)}] = 2^{-n} + 2^{-(n+1)} > 2^{-n} > \sum_{k=n+1}^{\infty} \arctan 2^{-k} \tag{42}$$

IV Proof of Convergence

Theorem : The algorithm generates the sequences Z_n^{α} and Z_n^{β} which satisfy the following property: at step *i* (Note that $\arctan 2^{-0}$ through $\arctan 2^{-(2i-1)}$ have **already** been used in prior steps $0, 1\cdots, (i-1)$ and step *i* uses $\arctan 2^{-2i}$ and $\arctan 2^{-(2i+1)}$)

$$|Z_i^{\alpha}| \le \sum_{k=2i}^{\infty} \arctan 2^{-k} \qquad \text{or} \qquad |Z_i^{\beta}| \le \sum_{k=2i}^{\infty} \arctan 2^{-k} \tag{43}$$

$$|Z| \le 3\cdot 2^{-(2i-1)} \qquad \text{where} \qquad Z \in \{Z_i^{\alpha}, Z_i^{\beta}\} \tag{44}$$

The above relations state that at least one of $|Z_i^{\alpha}|$ and $|Z_i^{\beta}|$ satisfies the "*tighter*" bound while both satisfy the "*coarser*" bound.

Proof : We prove the correctness of the algorithm by induction, i.e., assume that it holds at step *i* and show that it holds at step $i+1$. This, together with the base case (i.e., $i=0$) where the theorem holds (as seen from relation (18)) completes the proof.

There are two main cases to be considered:

(I) At step *i*, there is no on-going branching, i.e., one of Z_i^{α} or Z_i^{β} is determined to be the correct output and both modules start off with this value (Z_i) as the starting residue for step *i*.

(II) At step *i*, the algorithm is in a branching with distinct starting residues (Z_i^{α} for module α and Z_i^{β} for module β).

Case I : No on-going branching This is further subdivided into 2 cases

(I.1) $\text{Sign}(Z_i) = \pm 1$ and

(I.2) $\text{Sign}(Z_i) = 0$

We consider case (I.1) first and illustrate the proof assuming $\text{Sign}(Z_i) = +1$. The proof for the case when $\text{Sign}(Z_i) = -1$ is identical and is omitted for the sake of brevity.

(I.1) $\text{Sign}(Z_i) = +1$

As seen in the flow chart, the modules perform

$$\text{Module } \alpha : Z_{i+1}^{\alpha} = Z_i - \arctan 2^{-2i} - \arctan 2^{-(2i+1)} \tag{45}$$

$$\text{Module } \beta : Z_{i+1}^{\beta} = Z_i - \arctan 2^{-2i} + \arctan 2^{-(2i+1)} \tag{46}$$

Induction hypothesis and the fact that Sign(Z_i) = +1 yields: $\quad 0 < Z_i < \sum_{k=2i}^{\infty} \arctan 2^{-k}$ (47)

Hence, $\quad -\arctan 2^{-2i} - \arctan 2^{-(2i+1)} < Z_{i+1}^{\alpha} \leq \sum_{k=2i+2}^{\infty} \arctan 2^{-k}$ (48)

Using the fact that $\quad +\arctan 2^{-2i} + \arctan 2^{-(2i+1)} < 2^{-2i} + 2^{-(2i+1)} = 3 \cdot 2^{-(2i+1)}$ (49)

and invoking relation (25), we get

$$-3 \cdot 2^{-(2i+1)} < -\arctan 2^{-2i} - \arctan 2^{-(2i+1)} < Z_{i+1}^{\alpha} \leq \sum_{k=2i+2}^{\infty} \arctan 2^{-k} < 2^{-(2i+1)} \quad (50)$$

$$\text{or } -3 \cdot 2^{-(2i+1)} < Z_{i+1}^{\alpha} < 3 \cdot 2^{-(2i+1)} \implies |Z_{i+1}^{\alpha}| < 3 \cdot 2^{-(2i+1)} \quad (51)$$

Similarly, $\quad -\arctan 2^{-2i} + \arctan 2^{-(2i+1)} < Z_{i+1}^{\beta} \leq 2\arctan 2^{-(2i+1)} + \sum_{k=2i+2}^{\infty} \arctan 2^{-k}$ (52)

Using basic identities (25) and (26), we get

$$-\sum_{k=2i+2}^{\infty} \arctan 2^{-k} < Z_{i+1}^{\beta} < 2 \cdot 2^{-(2i+1)} + 2^{-(2i+1)} \quad \text{or} \quad |Z_{i+1}^{\beta}| < 3 \cdot 2^{-(2i+1)} \quad (53)$$

Identities (51) and (53) demonstrate that outputs of *both* modules satisfy the coarser bound as required.

Next we prove that at least one of Z_{i+1}^{β} and Z_{i+1}^{α} satisfies the tighter bound. When the algorithm does not enter branching, the residual angle which is determined to be "correct" by the algorithm is shown to satisfy the tighter bound. When the algorithm enters branching, at least one of the two is shown to satisfy the *"tighter"* bound . There are 3 cases to be considered:

(I.1.a) Sign(Z_{i+1}^{α}) = +1

(I.1.b) Sign(Z_{i+1}^{α}) = 0

(I.1.c) Sign(Z_{i+1}^{α}) = −1

(I.1.a) Sign(Z_{i+1}^{α}) = +1 : Here module α is correct. From relation (48) and the fact that Sign(Z_{i+1}^{α}) = +1, it follows that

$$0 < Z_{i+1}^{\alpha} < \sum_{k=2i+2}^{\infty} \arctan 2^{-k} \implies Z_{i+1}^{\alpha} \quad \text{satisfies the "}\textit{tighter}\text{" bound} \quad (54)$$

(I.1.b) Sign(Z_{i+1}^{α}) = 0 : Module α is correct. Note that 6 digits of weight

$2^{-(2i-1)}$, 2^{-2i}, $2^{-(2i+1)}$, $2^{-(2i+2)}$, $2^{-(2i+3)}$ and $2^{-(2i+4)}$ are examined to determine the signs, and Sign(Z_{i+1}^{α}) = 0 implies all 6 digits are zero. Hence

$$|Z_{i+1}^{\alpha}| < 2^{-(2i+4)} < 2^{-(2i+2)} < \sum_{k=2i+2}^{\infty} \arctan 2^{-k} \implies Z_{i+1}^{\alpha} \quad \text{satisfies "}\textit{tighter}\text{" bound} \quad (55)$$

(I.1.c) Sign(Z_{i+1}^{α}) = −1 : 3 subcases need to be considered here:

(I.1.c.i) Sign(Z_{i+1}^{β}) = −1: in this case module β is correct.

(I.1.c.i) Sign(Z_{i+1}^{β}) = 0: again module β is correct.

(I.1.c.i) Sign(Z^{β}_{i+1}) $= +1$: both modules are correct and the algorithm has entered branching.

(I.1.c.i) Sign(Z^{β}_{i+1}) $= -1$: This along with relations (46) and (47) implies that

$$-\arctan 2^{-2i} + \arctan 2^{-(2i+1)} < Z^{\beta}_{i+1} < 0 \quad \text{or} \tag{56}$$

$$|Z^{\beta}_{i+1}| < +\arctan 2^{-2i} - \arctan 2^{-(2i+1)} < \arctan 2^{-(2i+1)} < \sum_{k=2i+2}^{\infty} \arctan 2^{-k} \tag{57}$$

Identities (32) and (25) were used to obtain the last relation which demonstrates that Z^{β}_{i+1} satisfies *"tighter"* bound .

(I.1.c.ii) Sign(Z^{β}_{i+1}) $= 0$: In this case module β is correct and the proof that Z^{β}_{i+1} satisfies *"tighter"* bound is exactly same as case (I.1.b) above.

(I.1.c.iii) Sign(Z^{β}_{i+1}) $= 1$: In this case the algorithm has entered branching. From the fact that $Z^{\alpha}_{i+1} < 0$ and $Z^{\beta}_{i+1} > 0$ and relations (45), (46) and (47) it follows that

$$+\arctan 2^{-2i} - \arctan 2^{-(2i+1)} < Z_i < +\arctan 2^{-2i} + \arctan 2^{-(2i+1)} \tag{58}$$

From equations (58), (45) and (46) we get

$$-2\arctan 2^{-(2i+1)} < Z^{\alpha}_{i+1} < 0 \quad \text{and} \quad 0 < Z^{\beta}_{i+1} < 2\arctan 2^{-(2i+1)} \tag{59}$$

From equations (45) and (46), it follows that $Z^{\beta}_{i+1} - Z^{\alpha}_{i+1} = 2\arctan 2^{-(2i+1)}$ (60)

This, along with $Z^{\alpha}_{i+1} < 0$ and $Z^{\beta}_{i+1} > 0$ (as seen from (59)) implies

$$|Z^{\beta}_{i+1}| + |Z^{\alpha}_{i+1}| = 2\arctan 2^{-(2i+1)} \tag{61}$$

Hence, at least one of $|Z^{\alpha}_{i+1}|$ and $|Z^{\beta}_{i+1}|$ is $\leq \arctan 2^{-(2i+1)} < \sum_{k=2i+2}^{\infty} \arctan 2^{-k}$ (62)

which proves that at least one of Z^{α}_{i+1} and Z^{β}_{i+1} satisfies the tighter bound.

Case I.2: No on going branching and Sign(Z_i) = 0

This implies that digits of weight $2^{-(2i-3)}$, $2^{-(2i-2)}$, $2^{-(2i-1)}$, 2^{-2i}, $2^{-(2i+1)}$ and $2^{-(2i+2)}$ are all zero. Note that nothing is known about the remaining digits or in other words it is not known whether $Z_i \overset{?}{>} 0$ or $Z_i \overset{?}{=} 0$ or $Z_i \overset{?}{<} 0$. It might appear that all 4 possibilities $[\pm\arctan 2^{-2i} \pm \arctan 2^{-(2i+1)}]$ need to be examined. However, only two of the possibilities viz: $[+\arctan 2^{-2i} - \arctan 2^{-(2i+1)}]$ and $[-\arctan 2^{-2i} + \arctan 2^{-(2i+1)}]$ suffice as demonstrated next.

As per the flowchart, the the modules perform

$$\text{Module } \alpha : Z^{\alpha}_{i+1} = Z_i + \arctan 2^{-2i} - \arctan 2^{-(2i+1)} \tag{63}$$

$$\text{Module } \beta : Z^{\beta}_{i+1} = Z_i - \arctan 2^{-2i} + \arctan 2^{-(2i+1)} \tag{64}$$

$$\text{Note that } \text{Sign}(Z_i) = 0 \Longrightarrow |Z_i| < 2^{-(2i+2)} < \sum_{k=2i+2}^{\infty} \arctan 2^{-k} \tag{65}$$

Both Z^{α}_{i+1} and Z^{β}_{i+1} satisfy the *"coarser"* bound as shown by the following identities:

$$|Z^{\alpha}_{i+1}| \leq |Z_i| + |\arctan 2^{-2i} - \arctan 2^{-(2i+1)}| < 2^{-(2i+2)} + 2^{-(2i+1)} < 3 \cdot 2^{-(2i+1)} \quad (66)$$

$$|Z^{\beta}_{i+1}| \leq |Z_i| + |-\arctan 2^{-2i} + \arctan 2^{-(2i+1)}| < 2^{-(2i+2)} + 2^{-(2i+1)} < 3 \cdot 2^{-(2i+1)} \quad (67)$$

Next we show that: $\quad Z^{\alpha}_{i+1} > 0 \qquad$ and $\qquad Z^{\beta}_{i+1} < 0 \quad (68)$

Using relations (63) and (65) we get

$$\begin{aligned} Z^{\alpha}_{i+1} = [Z_i] &+ (\arctan 2^{-2i} - \arctan 2^{-(2i+1)}) > [-2^{-(2i+2)}] + (\arctan 2^{-2i} - \arctan 2^{-(2i+1)}) \\ &> \arctan 2^{-2i} - 2^{-(2i+1)} - 2^{-(2i+2)} > 0 \end{aligned} \quad (69)$$

where identity (34) was used to arrive at the last inequality. The fact that $Z^{\beta}_{i+1} < 0$ can be shown in an identical manner.

From equations (63) and (64), we have $\quad Z^{\alpha}_{i+1} - Z^{\beta}_{i+1} = 2(\arctan 2^{-2i} - \arctan 2^{-(2i+1)}) \quad (70)$

This, along with (68) yields $\quad |Z^{\alpha}_{i+1}| + |Z^{\beta}_{i+1}| = 2(\arctan 2^{-2i} - \arctan 2^{-(2i+1)}) \quad (71)$

$$\begin{aligned} \text{Hence, at least one of } |Z^{\alpha}_{i+1}| \text{ and } |Z^{\beta}_{i+1}| &\leq \arctan 2^{-2i} - \arctan 2^{-(2i+1)} \\ &< \arctan 2^{-(2i+1)} < \sum_{k=2i+2}^{\infty} \arctan 2^{-k} \end{aligned} \quad (72)$$

Thus, at least one of Z^{α}_{i+1} and Z^{β}_{i+1}satisfies the *"tighter"* bound .

Case II. Branching on–going

(II.1) Proof that at least one satisfies the *"tighter"* bound

Note that the algorithm can enter a branching only via one of two ways

(1) Case (I.1.c.iii): Sign(Z_i) = ± 1, Sign(Z^{α}_{i+1}) = ∓ 1 and Sign(Z^{β}_{i+1}) = ± 1 or

(2) Case (I.2): Sign(Z_i) = 0, which can lead to branching.

Identity (72) and the proof of case (I.1.c.iii) above demonstrate that whenever the algorithm enters branching, at least one of Z^{α} and Z^{β} satisfies the *"tighter"* bound . Without loss of generality assume that

$$\text{the branching started at step } i-1 \quad (73)$$

and that Sign(Z^{α}_i) = +1 (which implies that Sign(Z^{β}_i) = -1). (74)

We consider two subcases

(II.1.a) Branching immediately terminates

(II.1.b) Branching does not immediately terminate

(II.1.a) Branching immediately terminates :

Note that 6 digits of weight $2^{-(2i-3)}, 2^{-(2i-2)}, 2^{-(2i-1)}, 2^{-2i}, 2^{-(2i+1)}$ and $2^{-(2i+2)}$ are utilized to determine the sign of Z^{α}_i (prior to using $\arctan 2^{-2i}$ and $\arctan 2^{-(2i+1)}$). As seen in the flow charts, a branching is terminated (i.e., not continued) if condition (28) is satisfied by any of the two modules.

As mentioned before, condition (28) translates into checking whether

$$(\text{any of } \{|Z_i^{\alpha}|, |Z_i^{\beta}|\}) \leq 6 \cdot 2^{-(2n+2)} \tag{75}$$

(note that the constant C in identities (28) and (30) can take any of the two values 5 or 6. We demonstrate the proof for $C = 6$ only, as the proof for $C = 5$ is identical).

If only one of Z_i^{α} or Z_i^{β} satisfies the above condition, then it is the correct residue (and the current branching terminates). If both Z_i^{α} or Z_i^{β} satisfy the above condition, then any of the two modules can be arbitrarily deemed to be correct and the current branching terminates.

Once again, without loss of generality assume that Z_i^{α} satisfies this condition on magnitude which implies that Module α is deemed to be correct and the current branching is terminated. Hence we need to show that Z_i^{α} satisfies the *"tighter"* bound which is done using identity (33):

$$\sum_{k=2i}^{\infty} \arctan 2^{-k} > 2^{-2i} + 2^{-(2i+1)} = 6 \cdot 2^{-(2i+2)} \quad \text{hence} \quad Z_i^{\alpha} \leq 6 \cdot 2^{-(2i+2)} \Longrightarrow Z_i^{\alpha} < \sum_{k=2i}^{\infty} \arctan 2^{-k} \tag{76}$$

(II.1.b) Branching does not immediately terminate:

This happens when

$$\text{Sign}(Z_i^{\alpha}) = \pm 1 \text{ and } \text{Sign}(Z_i^{\beta}) = \mp 1 \text{ and } |Z_i^{\alpha}| > C \cdot 2^{-(2i+2)} \text{ and } |Z_i^{\beta}| > C \cdot 2^{-(2i+2)} \tag{77}$$

However, since the branching started at step $i-1$, then as per proofs of cases (I.1.c.iii) and (I.2) above at least one of Z_i^{α} and Z_i^{β} does satisfy the *"tighter"* bound .

Here, in step i, in the "continue-branching" mode, the modules execute

$$\text{Module } \alpha : Z_{i+1}^{\alpha} = Z_i^{\alpha} - \arctan 2^{-2i} - \arctan 2^{-(2i+1)} \tag{78}$$

$$\text{Module } \beta : Z_{i+1}^{\beta} = Z_i^{\beta} + \arctan 2^{-2i} + \arctan 2^{-(2i+1)} \tag{79}$$

The decisions for next step $(i+1)$ are based on $\text{Sign}(Z_{i+1}^{\alpha})$ and $\text{Sign}(Z_{i+1}^{\beta})$. We illustrate the proofs assuming Z_{i+1}^{α} satisfies the conditions for the case being considered without loss of generality (proofs are identical when the output of Module β satisfies the conditions instead). the following cases need to be considered.

(II.1.b.i) branching continues into step $i+2$:

This happens when

$$\text{Sign}(Z_{i+1}^{\alpha}) = \pm 1 \text{ and } \text{Sign}(Z_{i+1}^{\beta}) = \mp 1 \text{ and } |Z_{i+1}^{\alpha}| > C \cdot 2^{-(2i+4)} \text{ and } |Z_i^{\beta}| > C \cdot 2^{-(2i+4)} \tag{80}$$

To show that at least one of Z_{i+1}^{α} and Z_{i+1}^{β} satisfy the *"tighter"* bound , consider the case when $\text{Sign}(Z_i^{\alpha}) = +1$, (which implies $\text{Sign}(Z_i^{\beta}) = -1$, $\text{Sign}(Z_{i+1}^{\alpha}) = +1$ and $\text{Sign}(Z_{i+1}^{\beta}) = -1$. Other cases can be proved identically and are omitted for the sake of brevity).

Without loss of generality, assume that Z_i^{α} satisfied the *"tighter"* bound when branching started at the end of step $(i-1)$. Then $0 < Z_i^{\alpha} < \sum_{k=2i}^{\infty} \arctan 2^{-k}$, which, together with the fact that $\text{Sign}(Z_{i+1}^{\alpha})$

= +1 implies that

$$0 < Z^{\alpha}_{i+1} \leq \sum_{k=2i}^{\infty} \arctan 2^{-k} - \arctan 2^{-2i} - \arctan 2^{-(2i+1)} = \sum_{k=2i+2}^{\infty} \arctan 2^{-k} \tag{81}$$

which proves the desired result.

(II.1.b.ii) One of the modules terminates branching as per condition (28):
In this case, the proof is identical to that of case (II.1.a) above

(II.1.b.iii) One of the modules terminates branching via a change of sign:
Two sub cases need to be considered here

(A) One of the modules generates a zero sign: In this case the output of that module satisfies a bound similar to that stated in (65). Proof for the current case is based on this fact and is identical to that of case (I.2) above

(B) One of the modules changes sign from +1 to -1 or vice versa: For the purpose of illustration, assuming (73) and (74), consider the case when $\text{Sign}(Z^{\alpha}_{i+1}) = -1$.

In this case Z^{α}_{i} satisfies: $\quad C \cdot 2^{-(2i+2)} < Z^{\alpha}_{i} \leq \sum_{k=2i}^{\infty} \arctan 2^{-k} \quad$ where $\quad C = \{5,6\}$ (82)

This, together with the fact that $\text{Sign}(Z^{\alpha}_{i+1}) < 0$ yields

$$C \cdot 2^{-(2i+2)} - \arctan 2^{-2i} - \arctan 2^{-(2i+1)} < Z^{\alpha}_{i+1} < 0 \quad \text{or} \tag{83}$$

$$|Z^{\alpha}_{i+1}| = -Z^{\alpha}_{i+1} < +\arctan 2^{-2i} + \arctan 2^{-(2i+1)} - C \cdot 2^{-(2i+2)} < \sum_{k=2i+2}^{\infty} \arctan 2^{-k} \tag{84}$$

Where identity (25) and the fact that $C = \{5,6\}$ was used to arrive at the last inequality.

This completes proof (II.1) by showing that in branching, at least one of Z^{α} and Z^{β} satisfy the *"tighter"* bound .

(II.2) Proof that both Z^{α} and Z^{β} satisfy *"coarser"* bound in branching

Assume that the current step is i and that the branching started at step p. 3 sub cases need to be considered:
(II.2.a) $\text{Sign}(Z_p) = 0$
(II.2.b) $\text{Sign}(Z_p) = +1$ and
(II.2.c) $\text{Sign}(Z_p) = -1$

(II.2.a) $\text{Sign}(Z_p) = 0$ Then, there exists a Z_p (equal to Z^{α}_{p} or Z^{β}_{p}) such that

$$Z^{\alpha}_{i} = Z_p - \arctan 2^{-2p} + \sum_{k=2p+1}^{2i-1} \arctan 2^{-k} \quad \text{and} \tag{85}$$

$$Z^{\beta}_{i} = Z_p + \arctan 2^{-2p} - \sum_{k=2p+1}^{2i-1} \arctan 2^{-k} \tag{86}$$

$$\text{which implies} \quad Z^{\beta}_{i} - Z^{\alpha}_{i} = 2(\arctan 2^{-2p} - \sum_{k=2p+1}^{2i-1} \arctan 2^{-k}) \tag{87}$$

Equation (87) together with the fact that at least one of Z_i^{α} and Z_i^{β} satisfies the *"tighter"* bound implies that (using the triangle inequality $|x \pm y| \leq |x| + |y|$)

$$\begin{aligned}(\text{both } |Z_i^{\alpha}| \text{ and } |Z_i^{\beta}|) &\leq |2(\arctan 2^{-2p} - \sum_{k=2p+1}^{2i-1} \arctan 2^{-k})| + |\sum_{k=2i}^{\infty} \arctan 2^{-k}| \\ &< 2|\sum_{k=2i}^{\infty} \arctan 2^{-k}| + |\sum_{k=2i}^{\infty} \arctan 2^{-k}| < 3 \cdot 2^{-(2i-1)} \end{aligned} \tag{88}$$

where identities (25) and (26) were used to arrive at the last two inequalities.

(II.2.b) Sign(Z_p) = 1 In this case, by an argument similar to the one in case (II.2.a) above, it can be shown that

$$Z_i^{\beta} - Z_i^{\alpha} = 2(\arctan 2^{-(2p+1)} - \sum_{k=2p+2}^{2i-1} \arctan 2^{-k}) \tag{89}$$

From here on, the proof for this sub case is identical to that of case (II.2.a) above

(II.2.c) Sign(Z_p) = -1 The proof is similar to case II.2.b above

At this point, all possible cases have been exhausted and the proof is complete.

V Overview of Hardware Implementations

(a) Organization : CORDIC hardware for circular rotations consists of two main parts
(i) "Zeroing" module(s) to implement the Z recursion (equation (3)). These modules incorporate a look-up-table storing the elementary angles $\arctan 2^{-i}$, $0 \leq i \leq n+1$; and determine the "signs" s_i that decide whether to add or subtract each of the elementary angles $\arctan 2^{-i}$
(ii) $X - Y$ rotator modules which implement the cross coupled recursions in equations (1) and (2), utilizing the signs s_i determined by the zeroing modules.
Each of these parts is discussed below.

(i) *Zeroing Module(s)* : The "double stepping" discussed so far concentrated only on the zeroing part, i.e., on equation (3). Organization of the hardware required to implement the zeroing part follows from the flow charts, diagrams and tables presented in the previous sections. It consists of the two modules (Module α and Module β), each of which includes a full-wordlength-long signed-digit adder and the sign evaluation block as the main constituents. A decision block and some storage registers (for instance, to store the s_i values determined) are also required in the zeroing part.

(ii) *Rotator Modules* : In the original method, at step i, after the sign s_i is determined, an i positions-shifted Y_i is added (includes the operation of subtraction as well) to X_i and vice versa (please refer to equations (1) and (2)). To fully exploit the speed advantage gained by using two modules in the zeroing part, the branching method of Duprat and Muller should compute X_{i+1} and Y_{i+1} *in parallel* in two separate rotator modules. Each rotator module needs one shifter and one adder as the main building blocks. Possible hardware organization of the rotator modules required by the original scheme is illustrated in Figure 7-a.

We now outline hardware organization of the $X - Y$ rotator modules in the double step branching

CORDIC scheme. To this end, unroll the recursion in equations (1) and (2) one more time (i.e., express X_i and Y_i in terms of X_{i-1} and Y_{i-1}) to obtain

$$\begin{aligned} X_{i+1} &= (X_{i-1} - s_{i-1}2^{-(i-1)}Y_{i-1}) - s_i 2^{-i}(Y_{i-1} + s_{i-1}2^{-(i-1)}X_{i-1}) \\ &= X_{i-1}(1 - s_i s_{i-1}2^{-(2i-1)}) - [Y_{i-1}(s_i + 2s_{i-1})]2^{-i} \qquad (90) \\ Y_{i+1} &= (Y_{i-1} + s_{i-1}2^{-(i-1)}X_{i-1}) + s_i 2^{-i}(X_{i-1} - s_{i-1}2^{-(i-1)}Y_{i-1}) \\ &= Y_{i-1}(1 - s_i s_{i-1}2^{-(2i-1)}) + [X_{i-1}(s_i + 2s_{i-1})]2^{-i} \qquad (91) \end{aligned}$$

The above equations express X_{i+1} and Y_{i+1} in terms of X_{i-1} and Y_{i-1} and can be implemented as shown in Figure 7-b. In the double stepping method, s_i and s_{i-1} (which determine whether to add or subtract $\arctan 2^{-(i-1)}$ and $\arctan 2^{-i}$, respectively) get determined at step $\lfloor \frac{i-1}{2} \rfloor$. Note that Z_i^{α} and Z_i^{β} denoted the outputs of the "zeroing" modules at the end of step $(i-1)$ whereas X_i and Y_i in the above equations denote the outputs of rotator modules after

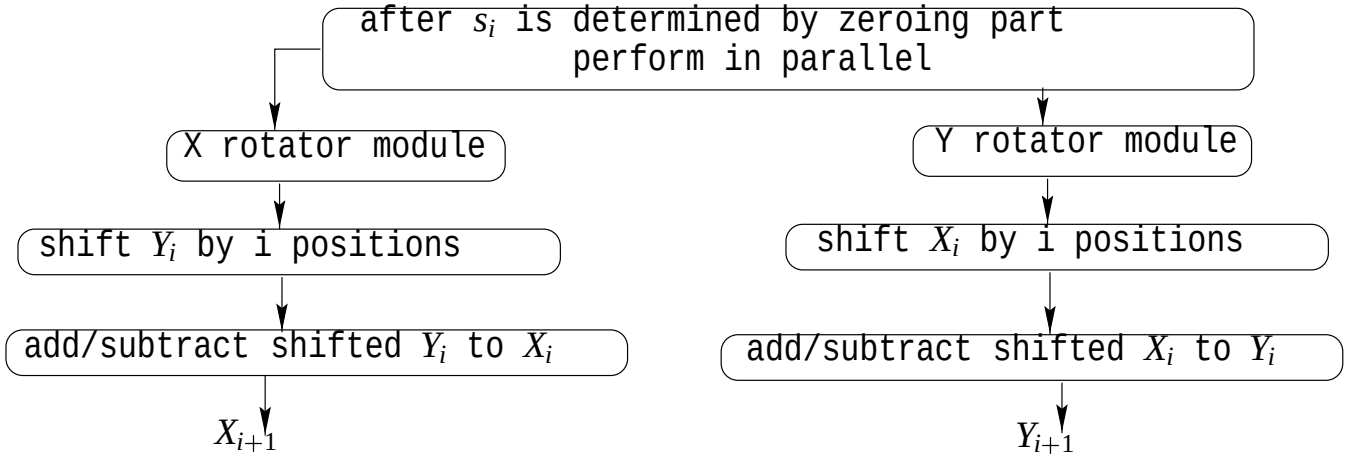

Figure 7-a : X–Y rotator schematic for Duprat and Muller's method
X_{i+1} and Y_{i+1} are generated from X_i and Y_i
as per equations (1) and (2).

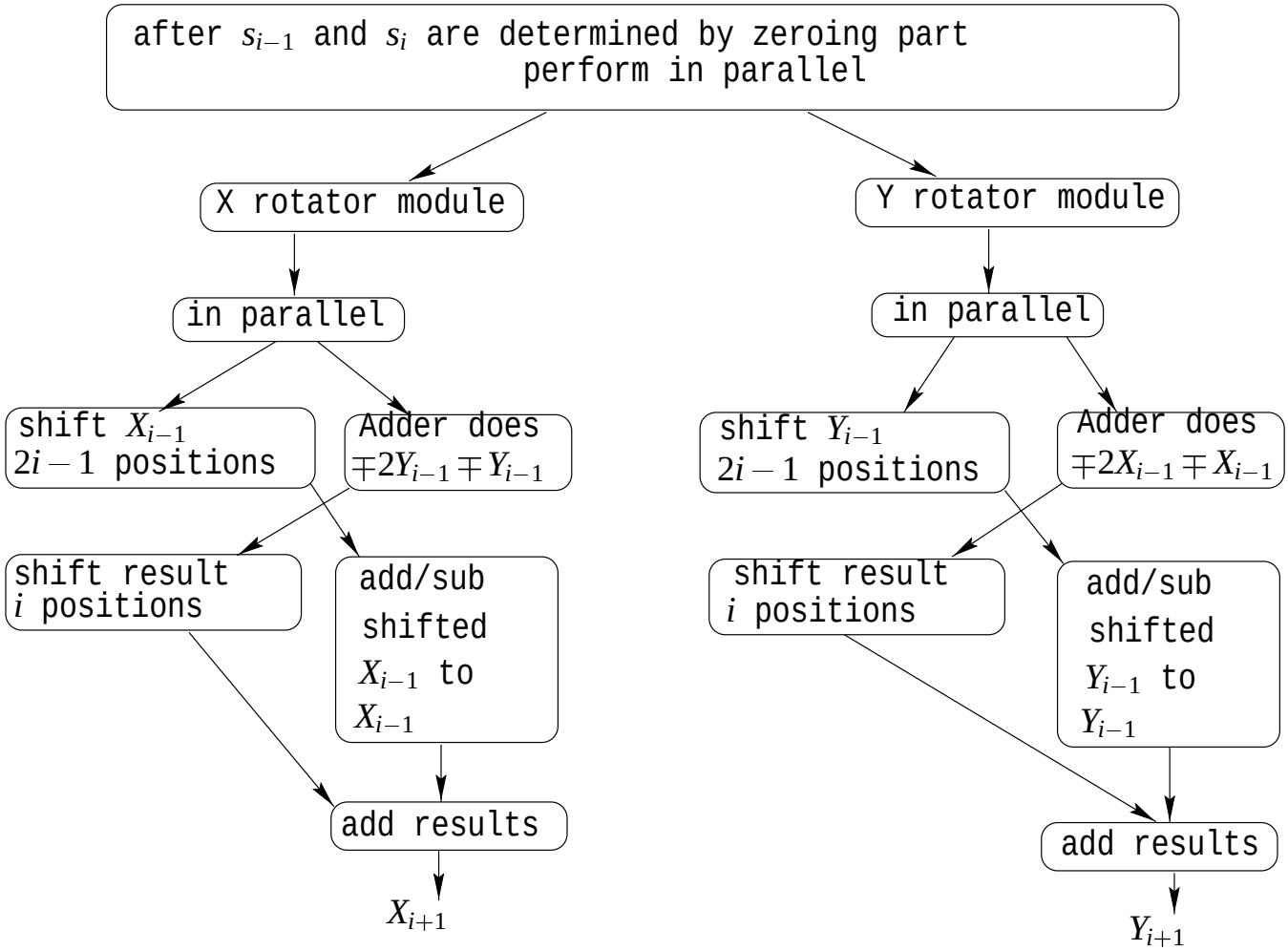

Figure 7-b : X–Y rotator schematic for Double Step Branching CORDIC
X_{i+1} and Y_{i+1} are generated from X_{i-1} and Y_{i-1}
as per equations (90) and (91).

having used angles upto and including $\arctan 2^{-(i-2)}$, i.e., at the end of step $\lfloor \frac{i-3}{2} \rfloor$ (The Z notation was selected to make the proofs and flowcharts more readable. That notation leads to a non-uniformity between the index subscripts of outputs of the zeroing modules and the rotator modules).

Next we compare the execution delay and hardware requirements of the double stepping method vis-a-vis the original method.

(b) Execution Delay

To begin with, note that the shift required in CORDIC rotations is *variable*; i.e., the number of positions to be shifted at each step is different, because the shift is a function of the iteration index. Hence, a general purpose shifter that can shift from 1 position all the way through n positions (n being the word length) is required. Barrel shifters [16] or crossbar switches have such capabilities. However, these shifters are complex and are likely to require a delay dependent on the word length (typically $O(\log n)$). The main point is that the shifter delay is likely to be comparable to (if not longer than) the adder delay, since a signed digit addition takes a (small) fixed amount of time delay independent of the wordlength.

(i) *Zeroing modules* : The delay required for executing two steps in the zeroing modules in the original method [1] is equivalent to a sequence of {`sign-detect, add, sign-detect, add`} operations. The {`sign-detect`} operation in [1] utilizes a window of 3 leading digits. The delay required by the decision block; copy operations (that are performed whenever the output of one of the modules can be detected to be wrong in which case that module copies the output of the correct module) and latching is also lumped along with the delay of the {`sign-detect`} operation for the sake of simplicity.

In comparison, the delay required for executing one "double stepping" iteration in the zeroing part of our method is equivalent to that of {`sign-evaluate, add`} operations. Note that instead of storing individual angles $\arctan 2^{-2i}$ and $\arctan 2^{-(2i+1)}$ only their sum and difference, viz., $(\arctan 2^{-2i} + \arctan 2^{-(2i+1)})$ and $(\arctan 2^{-2i} - \arctan 2^{-(2i+1)})$ need to be stored for our algorithm. Hence, two additions in the original method get replaced by a single addition in our double stepping method.

The {`sign-evaluate`} operation in our method is more complex than that in the original method since our window size is 6 while the original method uses a window of 3 digits. However, the additional delay required for the sign evaluation in our method is likely to be small, since the two subgroups of 3 digits are handled in parallel. The only extra delay is that of deciding the overall sign and determining whether condition (28) is satisfied after the two subgroups in the window have generated their signals (independently).

(ii) *Rotator modules* : The delay required for executing two $X-Y$ rotations the original method is tantamount to a sequence of {`shift, add, latch, shift, add, latch`} operations (each rotation needs a shift, an addition and latching as per equations (1) and (2)).

The delay required for two $X-Y$ rotations (that are performed in a single step in our method) can be *approximated* to that of a sequence of
{`add, latch, add, latch, add, latch`} operations
(or {`shift, latch, shift, latch, add, latch`} operations if the shifter takes a longer time

than the adder). Assuming that
(i) the shifter delay is about the same as the adder delay $\Delta_{\text{shift}} \approx \Delta_{\text{add}}$, and
(ii) the delay required for addition Δ_{add} is longer than delay required for latching Δ_{latch},
it can be seen that the $X-Y$ rotator part in the double stepping method saves $[\Delta_{\text{add}} - \Delta_{\text{latch}}]$ for every iteration (of the double stepping method, or equivalently, for every 2 rotations in the original method). It should be noted that this is an approximate estimate. If the shifter or adder can be pipelined, the results might look different.

Finally, note that in the original method, both the zeroing and rotator blocks work in tandem, with the rotator modules utilizing the sign s_i determined by the zeroing module(s). If the {`sign-detect`} operation in the original method (which includes the decision block delay, along with module-to-module copying and latching delays) takes a longer delay than the shift operation, then it is possible that the $X-Y$ rotator modules could get stuck, waiting for the zeroing module to finish its part.

In contrast, in the double stepping method, the delay of the zeroing part is likely to be smaller than the delay of the $X-Y$ rotator part, so that the rotator modules would never idle. Thus, the double stepping method could lead to significantly faster execution time depending on the actual implementation.

(c) Area and Hardware Utilization
Next we look at the hardware (Area) requirements of and hardware utilization in both methods.

(i) *Look–Up Table Size* : First, note that the look–up table size remains identical to that of the original method. Instead of storing individual angles $\arctan 2^{-2i}$ and $\arctan 2^{-(2i+1)}$ only their sum and difference, viz., $(\arctan 2^{-2i} + \arctan 2^{-(2i+1)})$ and $(\arctan 2^{-2i} - \arctan 2^{-(i2+1)})$ need to be stored for our algorithm.

(ii) *Rotator Block* : In the $X-Y$ rotator part, the double stepping scheme needs only a little more hardware compared with the original method. As in the original method of Duprat and Muller, the main constituents of each of the X and Y rotator modules in our method are one adder and one shifter. In the original method the shifters are idle when the adders are active and vice versa, while in the double stepping method, the adders and shifters work in parallel, virtually eliminating idle time (thereby enhancing hardware utilization). The small additional hardware overhead in our method arises because the adders and shifters need selectors at their inputs and latches to hold intermediate results.

(iii) *Zeroing Block* : Implementation of the zeroing part of the double stepping method also needs two modules as in the original case. Note that both modules (in particular, the adders which are full word-length long) are doing distinct (and useful) operations in every iteration in the double stepping method. In comparison, in the original method, the modules in the zeroing part perform distinct operations only when the algorithm is in a branching. One of the modules in the zeroing part essentially remains unutilized whenever the algorithm (in [1]) is not in a branching.

In our method, 6 digits need to be examined instead of 3, to evaluate the sign of the residual angle. Hence, the sign detection hardware will become more complex than that in the original method.

Similarly, the decision block is slightly more complex than that in the original method. The additional circuits (in the sign-evaluation and decision blocks in our method), however, act on a fixed and small number of extra digit positions (3 digits) and are therefore likely to be small (in comparison with the full wordlength long adders, shifters, etc.)

It should be noted the decision blocks in the flowcharts in Figures 1–4 probably appear more complicated in software: the hardware complexity of implementing these blocks is likely to be somewhat higher but comparable to the hardware complexity of decision blocks in the original method [1]. For example, the evaluation of the overall sign of the entire group of 6 digits in the window can be simply done with a simple multiplexor once the signs of the individual subgroups are evaluated (in parallel). Thus, the decision block and the sign evaluation circuits together should constitute only a small fraction of the total hardware required which is likely to be dominated by (full word length) adders, shifters, latches and the look-up table. Hence, the additional hardware overhead of the double stepping method is likely to be small, but worthwhile, since in return, the utilization of large blocks (such as adders, shifters ...) can be substantially increased.

In summary, the double stepping method has the potential to reduce the execution time at the cost of a small hardware overhead. It leads to a significant improvement in hardware utilization.

VI Conclusion

We have proposed the Double Step Branching CORDIC algorithm and shown that it is possible to perform two rotations in a single step, with little additional hardware (compared to Duprat and Muller's Branching CORDIC method). In our method, both modules perform distinct computations at each step which leads to a better utilization of the hardware and the possibility of further speedup over the original method. Architectures for hardware implementation of our algorithm are discussed.

A natural question is why not try to triple-step ? A preliminary investigation quickly reveals that the hardware complexity increases exponentially, and the decision blocks would become so complex that the delay of such a method is likely to be *higher* than the original method. Double stepping appears to be the optimum when speed, hardware cost and utilization are considered.

Acknowledgment

The author would like to thank Prof. Israel Koren from the University of Massachusetts at Amherst for

(i) getting him interested in CORDIC, and showing him Duprat and Muller's ingenious work.

(ii) several important discussions related to this manuscript.

(iii) suggesting the idea of a software simulator to independently verify the algorithm.

The anonymous reviewers are also thanked for their constructive comments which helped to improve the quality of the final manuscript.

References

[1] J. Duprat and J. Muller, "The CORDIC algorithm: new results for fast VLSI implementation," *IEEE Trans. on Computers*, vol. TC–42, pp. 168–178, Feb. 1993.

[2] J. E. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Trans. on Electronic Computers*, vol. EC–8, pp. 330–334, Sep. 1959.

[3] J. S. Walther, "A unified Algorithm for Elementary Functions," in *Proceedings of the 38th Spring Joint Computer Conference*, pp. 379–385, 1971.

[4] A. M. Despain, "Fourier Transform Computers Using CORDIC Iterations," *IEEE Transactions on Computers*, vol. C-30, pp. 993–1001, Oct. 1974.

[5] S.-F. Hsiao and J.-M. Delosme, "The CORDIC Householder Algorithm," in *Proc. of the 10th Symp. on Computer Arithmetic*, pp. 256–263, 1991.

[6] J. R. Cavallaro and F. T. Luk, "CORDIC Arithmetic for a SVD processor," *Journal of Parallel and Distributed Computing*, vol. 5, pp. 271–290, 1988.

[7] M. D. Ercegovac and T. Lang, "Redundant and on-line CORDIC: application to matrix triangularization and SVD," *IEEE Transactions on Computers*, vol. C-39, pp. 725–740, Jun. 1990.

[8] E. Deprettere, P. Dewilde, and R. Udo, "Pipelined CORDIC Architecture for Fast VLSI Filtering and Array Processing," in *Proceedings of ICASSP'84*, pp. 41.A.6.1–41.A.6.4, 1984.

[9] P. Strobach, "The Square-Root Schur RLS Adaptive Filter," in *Proceedings of ICASSP'91, Toronto, May 1991*, pp. 1845–1848, 1991.

[10] M. Terre and M. Bellanger, "Systolic QRD-Based Algorithm For Adaptive Filtering and Its Implementation," in *Proceedings of the 1993 IEEE International Conference on Acoustics, Speech and Signal Processing, Minneapolis, MN*, vol. III, pp. III.296–III.298, 1993.

[11] I. Koren, *Computer Arithmetic Algorithms*. Prentice-Hall Inc., Englewood Cliffs, NJ, 1993.

[12] CORDIC Bibliography site including tutorials and simulation code, accessible through the URL http://devil.ece.utexas.edu.

[13] B. Parhami, "Generalized signed-digit number systems: a unifying framework for redundant number representations," *IEEE Transactions on Computers*, vol. C-39, pp. 89–98, Jan. 1990.

[14] D. S. Phatak and I. Koren, "Hybrid Signed–Digit Number Systems: A Unified Framework for Redundant Number Representations with Bounded Carry Propagation Chains," *IEEE Trans. on Computers, Special issue on Computer Arithmetic*, vol. TC–43, pp. 880–891, Aug. 1994. (An unabridged version is available on the web via the URL http://www.ee.binghamton.edu/faculty/phatak).

[15] N. Takagi, T. Asada, and S. Yajima, "Redundant CORDIC methods with a constant scale factor for Sine and Cosine computation," *IEEE Trans. on Computers*, vol. 40, pp. 989–999, Sep. 1991.

[16] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design, A Systems Perspective*. Addison Wesley, 1988.