Comparison of Branching CORDIC Implementations *

Abhishek Singh, Dhananjay S Phatak, Tom Goff, Mike Riggs, James Plusquellic and Chintan Patel
Computer Science and Electrical Engineering Department
University of Maryland Baltimore County (UMBC)
Baltimore, MD 21250
phatak@umbc.edu

Abstract

In this paper we compare implementations of Duprat and Muller's Branching CORDIC and Phatak's Double Step Branching (DSB)–CORDIC algorithms for Sine and Cosine evaluation. For reference we also report on classical CORDIC implementations for the same wordlengths. We have also implemented Double Stepping in the classical algorithm and report on the performance of this method.

CORDIC evaluation of Sine and Cosine includes two parts, the Zeroer and the Rotator. We discuss implementation issues related to the minimization of the delay of each iteration of the algorithm (including delays for both the Zeroer as well the Rotator). We then examine hybrid methods that select the components from different algorithms (such as a DSB Zeroer together with a classical rotator or vice versa).

Index terms : Branching CORDIC, Implementations, Comparison

1 Introduction

In this paper our main goal is to compare the relative performance of implementations of two Branching CORDIC algorithms, viz., Duprat and Muller's original method [1] and the Double Step Branching CORDIC [2] for Sine and Cosine evaluation. Right at the outset, we would like to point out that we do not attempt an exhaustive comparison of methods for evaluating forward trigonometric functions, rather we focus on a relative comparison of the two specific CORDIC methods mentioned above. We have also included results on implementations of classical CORDIC methods for gaining more insight regarding the advantages and drawbacks of each of the integral components of the Branching CORDIC methods.

We begin with a brief explanation of the circular rotations which are based on the iteration [1]

$$X_{i+1} = X_i - s_i Y_i 2^{-i} \quad (1)$$

$$Y_{i+1} = Y_i + s_i X_i 2^{-i} \quad (2)$$

$$Z_{i+1} = Z_i - s_i \arctan 2^{-i} \quad \text{where} \quad s_i \in \{-1, 0, +1\} \quad (3)$$

This work was supported in part by NSF grants ECS-9875705, ECS-0196362 and CISE 0098300

Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03)
ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE

In this paper, the module(s) that implement iteration (3) are together referred to as the "Zeroer" and the module(s) that implement the cross-coupled iterations (1)-(2) constitute the "Rotator".
Using complex numbers, equations (1) and (2) can be rewritten as [3]

$$\mathcal{W}_{i+1} = \mathcal{W}_i \cdot (1 + j s_i 2^{-i}) \quad \text{where} \tag{4}$$

$$\text{complex variable} \quad \mathcal{W}_i = (X_i + j \cdot Y_i) \quad \text{and } j = \sqrt{-1} \tag{5}$$

Using the polar form of complex numbers,

$$\mathcal{W}_{i+1} = \mathcal{W}_i \sqrt{1 + (s_i 2^{-i})^2} \cdot e^{j\theta_i} = \mathcal{W}_i \cdot K_i \cdot e^{j\theta_i} \quad \text{where} \tag{6}$$

$$\theta_i = \arctan(s_i 2^{-i}) \quad \text{and} \quad K_i = \sqrt{1 + (s_i 2^{-i})^2} \tag{7}$$

Starting with $\mathcal{W}_0$, if m iterations are carried out, then

$$\mathcal{W}_m = \mathcal{W}_0 \cdot K \cdot e^{j\theta} \quad \text{where} \tag{8}$$

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

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

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

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

$$\mathcal{W}_m \ \dashrightarrow \ \mathcal{W}_0 \cdot K(cos\theta_0 + jsin\theta_0) = (X_0 + jY_0) \cdot K(cos\theta_0 + jsin\theta_0) \quad \text{or} \tag{12}$$

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

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

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

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

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).

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

$$X_m \dashrightarrow \cos\theta_0 \qquad \text{and} \qquad Y_m \dashrightarrow \sin\theta_0 \tag{17}$$

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

Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03)
ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE

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 driven 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 [3, 4, 5] 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, 6]) 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 [6] 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).

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 (this is done in the Double Step Branching CORDIC method). In fact, in the original method the two modules do

Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03)
ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE

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 when in branching. DSB–CORDIC enhances 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 reasonable, while the hardware utilization is better and there is a potential for speedup. In other words, at the algorithm level, it appears that DSB-CORDIC could yield substantial speed enhancement over Branching CORDIC. (A detailed description of the algorithms can be found in [1] and [2]. The algorithms are fairly involved and reproducing their flowcharts or state diagrams has been avoided for the sake of brevity).

In order to see how much of that speed-enhancement potential is actually realized in hardware, we have implemented the algorithms using synthesis tools. We would like to point out that describing the algorithms in a combination of structural and behavioral Verilog modules and verifying the functional correctness of the designs is itself a non-trivial task. We have implemented 16 and 32 bit versions of the algorithms using ASIC synthesis tools and report the area and delay estimates generated by the tools. Besides the experimental results, some fundamental issues related to the implementations are discussed. Finally we examine hybrid methods that combine components of different algorithms, such as a DSB-CORDIC zeroer with a classical rotator or vice versa.

2 Double Stepping Version of Classical CORDIC

This is similar to DSB–CORDIC method in [2], except it is a lot simpler. We developed and implemented this method for the sake of gaining further insight when comparing the various implementations. We adopt the convention to label the steps from zero onwards, (i.e., step 0, step 1, ... etc.) so that step i utilizes $\arctan 2^{-2i}$ and $\arctan 2^{-(2i+1)}$ to generate new residual angle Z_{i+1} from Z_i. There are two modules (denoted as "Module α " and "Module β ") that operate in parallel. Assuming the sign of residue Z_i is $+$ (note that in 2's complement notation 0 would be deemed to have a + sign which does not cause a problem), the modules perform:

$$\text{Module } \alpha : Z^{\alpha}_{i+1} = Z^{\alpha}_{i} - \arctan 2^{-2i} - \arctan 2^{-(2i+1)} \quad \text{operation abbreviated } \langle - \ - \rangle \tag{19}$$

$$\text{Module } \beta : Z^{\beta}_{i+1} = Z^{\beta}_{i} - \arctan 2^{-2i} + \arctan 2^{-(2i+1)} \quad \text{operation abbreviated } \langle - \ + \rangle \tag{20}$$

If $Z^{\alpha}_{i+1} > 0$, i.e., a $\langle - \ - \rangle$operation still leaves a positive residue then module α is correct. Likewise,

```
if ( Sign(Z_i) == −Sign(Z^β_{i+1}) ) then
module β is correct
```

otherwise we compare the magnitudes of Z^{α}_{i+1} and Z^{β}_{i+1} and pick the smaller one. Note that such a magnitude comparison implies a full word length long addition (It is possible to look at a (small) fixed window of leading bits that slides to the right by two bit positions each iteration as in the DSB–CORDIC method in order to avoid a full magnitude comparison. However, in this round of experiments we did not pursue this extension). The results (value and sign) of the correct module are copied by the both modules in preparation for the next iteration. The method is summarized by the flowchart in Figure 1. In order to achieve an error $< 2^{-n}$, $n+3$ angles $\arctan 2^{-0}, \arctan 2^{-1} \cdots \arctan 2^{-(n+2)}$ have to be used (note that the angles are used in pairs, so in general, there is no option of stopping after using only $\arctan 2^{-(n+1)}$, the next angle $\arctan 2^{-(n+2)}$ will get used if n is odd).
Hence the number of iterations = $\lceil \frac{n+3}{2} \rceil$ exactly as in DSB–CORDIC.

Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03)
ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE

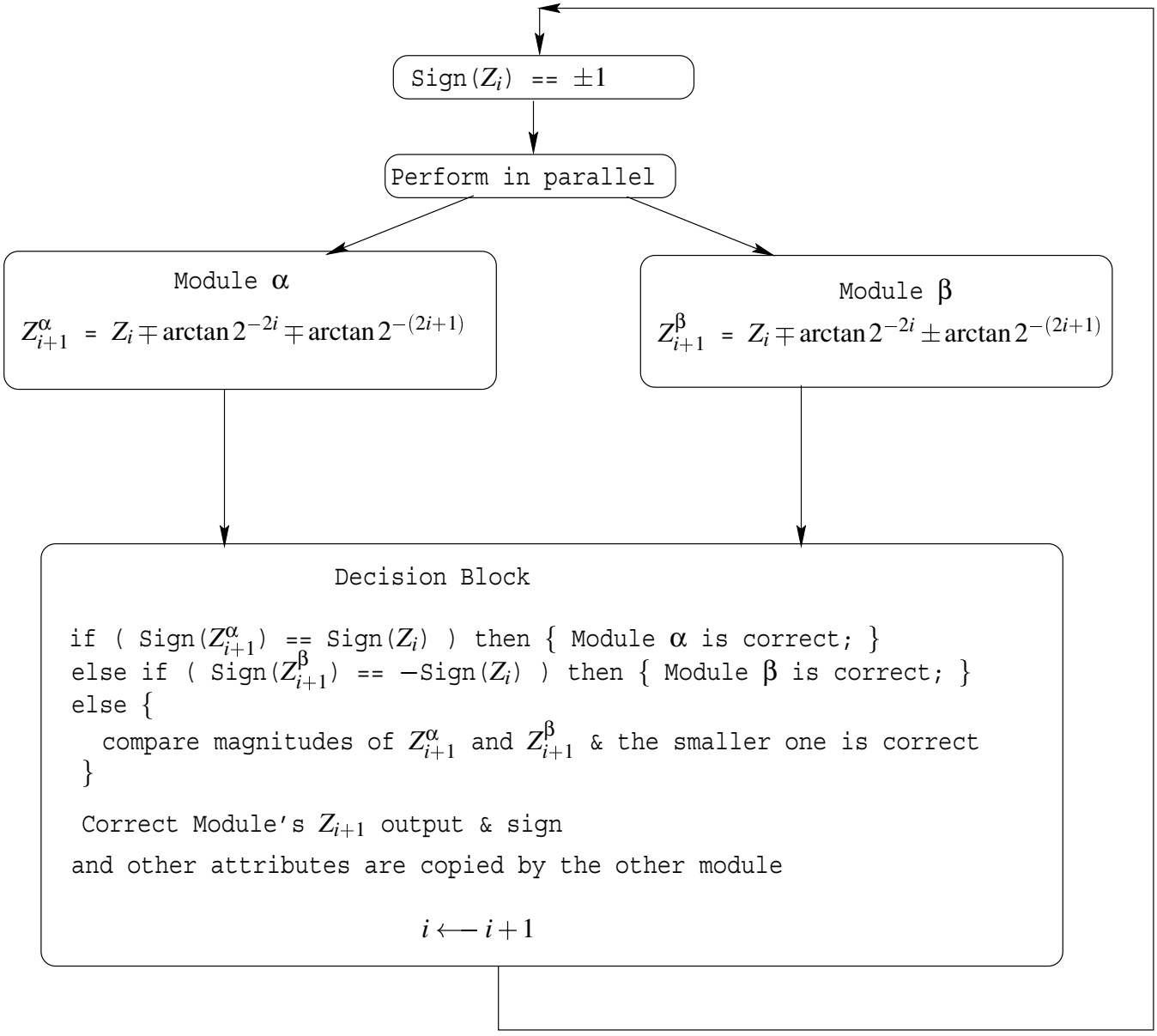

Figure 1. Flowchart summarizing the operation of the Zeroer in the i th (each) iteration of the Double Step Classical (DSC)–CORDIC.

Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03)
ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE

Instead of storing $\arctan 2^{-2i}$ and $\arctan 2^{-(2i+1)}$ their sum and difference are stored in the ROM, so that the total ROM size is (almost) the same as in classical method (it has two extra rows and 2 extra columns which is not a significant increase in the area).

Note that this method never needs branching, the explicit magnitude comparison avoids the need to branch by selecting the correct result. If ROM delay is significant, then this method of zeroing could be faster than the classical single stepping method. (the classical method would perform { `lookup, add, latch, lookup, add, latch`} while the double stepper would perform { `lookup, add, add, latch` } in the worst case (note that the decision logic amounts to examining 3 bits and simply muxing-in the right values and is therefore assumed to contribute negligible additional delay).

The main motivation behind implementing this method is to try to isolate the effect of using signed digits and associated "sign-evaluate" operations (that take non-trivial delay) that the branching CORDIC methods must follow.

3 Implementation and Results

3.1 Fundamental Issues

The whole point behind using signed-digits (or any other redundant representation) for the intermediate operands is to speed-up the operation. Redundancy helps by rendering the addition carry-free [5, 7], so that two operands can be added in a (small) constant time *independent* of the wordlength ($O(1)$delay). To preserve/fully exploit the advantage afforded by signed digits, other operations in each iteration should also be executed in constant-time (if this is feasible at all).

It turns out that all the operations in the Zeroer can be done in $O(1)$delay if $O(n^2)$ hardware is employed (n refers to the word length). The Rotator poses a much harder problem: as seen in equations (1) and (2) variable length shifts are needed. In iteration i we need both the unshifted X_i, Y_i as well as these same variables shifted by i places. One way to render the delay of each iteration of the Rotator to be $O(1)$is to unroll all the iterations in hardware so that the shifts are pre-wired. The other possibility is a crossbar switch. Both these schemes require a large amount of (i.e., $O(n^2)$) hardware. In order to fit the design on FPGAs available to us, we used multi-stage shifters requiring $\log n$ stages and $O(n \log n)$ hardware to implement arbitrary shifts in our rotators.

3.2 Architecture

The overall organization for Branching CORDIC methods is shown in Figure 2. (we again re-iterate that the algorithms are fairly involved and due to length constraints, it is not possible to replicate their flowcharts of state diagrams in this paper. The details of the algorithms can be found in [1] and [2]. This paper compares their IMPLEMENTATIONS).

3.2.1 Branching CORDIC Algorithms

Note that Branching CORDIC methods perform two independent computations in parallel in each step. Two separate modules are required for this purpose. These modules are indicated by the dotted boxes labeled

Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03)
ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE

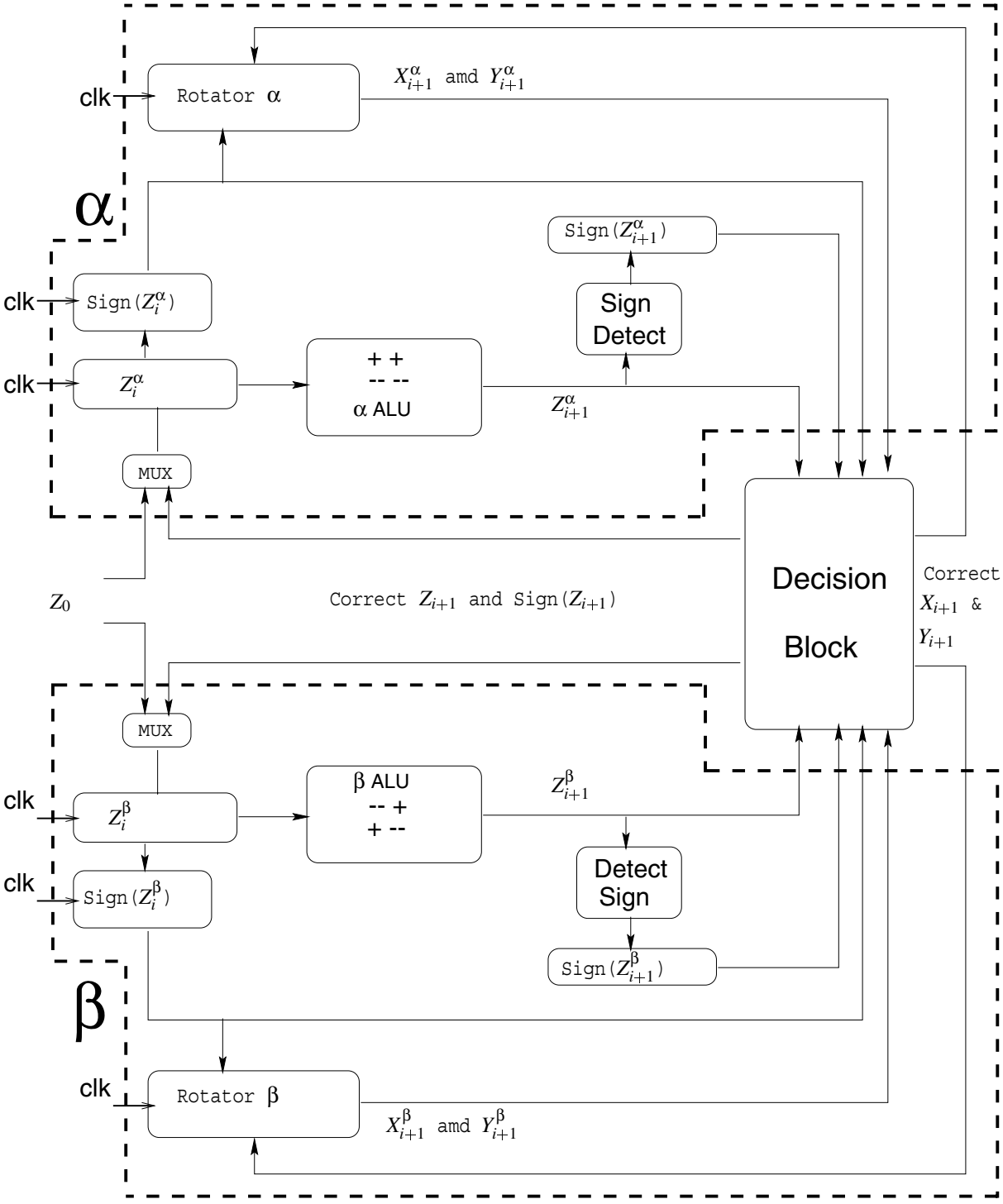

Figure 2. Overall Architecture for Branching CORDIC Algorithms

Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03)
ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE

α and β in Figure 2. Each module (α and β) itself includes a Zeroer and a Rotator.

The Zeroer consists of the Residue register (Z_i^{α} in module α and Z_i^{β} in module β), the sign latch, the ALU (which includes the signed digit adder), and the "Sign Detect" submodule.

Each rotator (in modules α and β) is a fairly big block in itself. The organization of rotators for the original branching and the DSB method is outlined in [2]. We have followed that organization. The main difference is that the rotators in the DSB method use 2 consecutive signs and directly evaluate $\{X,Y\}_{2i+2}$ from $\{X,Y\}_{2i}$. In other words, basic recursions (1) and (2) are unrolled once so that two iterations of the conventional method are accomplished in the double stepping method. The unrolling exposes some parallelism (as explained in [2]). To fully exploit the parallelism we have employed a lot of hardware. For instance each rotator module (module α as well as module β) uses 2 shifters and 3 (Signed Digit) adders. In contrast in the original Branching CORDIC method each rotator module needs 1 shifter and 1 adder.

It is possible to pipeline the shifter and the adder and cut down the amount of hardware in the DSB method to 1 shifter and 1 adder per module. However, the pipeline would have to be operated on a a different clock (faster than the overall clock cycle for each iteration) which is not that straightforward to implement with the synthesis tools we have used.

The rotator turns out to be the bottleneck in circular forward CORDIC and unless more hardware is thrown in to exploit the modest amount of parallelism exposed by recursion unrolling, there is no advantage to double stepping in the zeroing part at all. In other words, this critical path benefits the most from extra hardware. Furthermore, one of our main goals was to see the intrinsic capabilities of the methods. Hence, we decided to throw in the 2 shifters and 3 adders per rotator module. It should be noted that this is likely to be less expensive (in terms of transistors and routing area) than using a crossbar switch. Two iterations of the conventional rotator perform $\{$`shift, add, latch, shift, add, latch`$\}$ operations. In contrast the recursion unrolling and extra hardware cuts down the delay of each iteration of the Double stepping rotator to that of a sequence of $\{$`shift, add, add, latch`$\}$ operations (eliminating a shift and latch delay which can be significant since the shifter delay is word-length dependent and likely to be much longer than redundant addition delay which is small and independent of wordlength).

The ROM which stores the elementary angles can be shared by both modules and is omitted from the (already dense) diagram in Figure 2 for the sake of clarity. It supplies the elementary angles to the ALUs in the two modules.

The "Decision Block" shown in the Figure accepts results from both modules and determines which module performed the correct operation and provides the correct values of the "next residue", it's sign as well as the X,Y variables being updated by the rotators.

3.2.2 Architectures for Classical CORDIC Algorithms

These architectures are straightforward and a lot less involved than their Branching counterparts. The main difference is that there are no redundant representations (such as Signed Digit or Carry/borrow Save, etc.) used anywhere. As a result the adders employ carry-look-ahead designs. Because all intermediate results are in two's complement format, the "Sign Detect" operation is reduced to the trivial task of examining the sign bit (there are no "sliding windows" in the Zeroers).

Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03)
ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE

In single stepping classical CORDIC there is only one module and no decision block is necessary. This method simply implements equations (1)–(3) in each clock cycle (Equation (3) in Zeroer and equations (1) and (2) in the rotator).

Classical method with double stepping is explained in Section 2. It uses 2 elementary angles in one step but does not use signed digits. Here, two modules and a decision block are needed and the overall organization is similar to that in Figure 2. The rotators in this algorithm use multiple shifters and adders like its Branching counterpart.

3.3 Operation

Each iteration of the respective methods is implemented in 1 clock cycle (for that method). At the end of a cycle the correct residual angle and its sign are known so that the operations to be performed in the next step are determined. The Zeroer and the Rotator modules operate in tandem implementing the operations (determined at the end of previous cycle).

The Zeroer module adds the new elementary angle to the current residual angle to generate the new residual angle (which gets driven toward zero as the iterations progress). The "Sign Detect" module then evaluates the sign of the new residue. The new residues and their signs are then used to evaluate which module was correct and determine the sign and value of the next residue and bring it (the new residue) back to the input of the "Residue" registers to be latched-in for starting the next iteration at the next edge of the clock.

The Rotator works in tandem. At the beginning of the clock cycle the current values of X, Y variables (refer to (1) and (2)) are known along with s_i to be used in the current iteration. Using these values the rotator executes the cross-coupled iterations form (1) and (2). Note that in the Double Stepping methods, two iterations of recursions (1) and (2) are executed in one step (clock cycle) of the respective method.

If the "Decision Block" determines that one of the rotator modules performed incorrect operations, it sends the correct X, Y values from the other module on the same clock edge. Thus all the operations that logically belong to one iteration are implemented in 1 clock cycle.

3.4 Implementation

The designs were coded in Verilog and synthesized as an ASIC using the Cadence "Physical Knowledgeable Synthesis" (PKS) tool, version V4.0-S008. The cell library was designed by Artisan and derived from process parameters of the TMSC 0.18 micron technology with 6 metal layers.

We would like to emphasize that all the organizations, optimizations, etc were uniform across the board for both methods (DSB as well as original method). For instance, the same Signed Digit adder (from [8]) and the same shifter, were used (supplied as structural Verilog modules). Likewise the same ROM was used in all methods. The parameters and constraints given to the synthesis tools were identical (same level of logic compaction effort, same overall effort level, etc.). It is therefore reasonable to expect that any differences in the execution time, area, etc. are mainly due to the intrinsic differences in the underlying algorithms. Since

Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03)
ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE

we are interested in a *relative* comparison of the methods, the ratios of areas and delays are more important than individual raw nanoseconds (for delay) or microns (area) for a particular method.

3.5 Results and Discussion

The results of the PKS synthesis are shown in Table 1. We would like to point out that the timing estimates reported in the table were derived from a static timing analysis implemented by the tool. We believe that these estimates reflect the true delays within reasonable error bounds.

Algorithm		Clock Cycle Delay (ns)			Gate Count		
		Zeroer	Rotator	Overall	Zeroer	Rotator	Overall
Classical	16 bit	1.53	2.10	2.13	8688	21202	27256
	32 bit	2.29	2.53	2.59	21834	44574	60720
Classical with	16 bit	2.34	2.66	2.67	19885	47341	63564
Double Stepping	32 bit	2.60	3.06	3.09	110230	96395	201875
Branching	16 bit	1.67	1.89	2.10	26631	49772	69987
CORDIC [1]	32 bit	1.83	2.01	2.12	55787	100374	143384
Double Step	16 bit	2.29	2.34	2.44	32788	73853	94669
Branching CORDIC [2]	32 bit	2.35	2.49	2.58	58794	166596	198912

Table 1. Results of ASIC synthesis with the Cadence Physical Knowledgeable Synthesis (PKS) tool

We synthesized the Zeroer alone, Rotator alone as well as the whole system separately which enables a quick consistency check:
(i) The overall gate count should be about the sum of the gate counts for the Zeroer and the Rotator which can be verified in the last 3 columns of the table
(ii) The Zeroer and the Rotator work in parallel. We expect the delay of the rotator to be longer than the Zeroer. Accordingly the overall delay will be dominated by the rotator delay and the first 3 columns of the table corroborate the expected trend.

The table demonstrates that the Double stepping leads to significant speed advantage. For instance for $n = 16$ bits, 2 iterations of the classical method take 4.26 ns, Original Branching CORDIC takes 4.2 ns, DS-classical (which does not use signed digits) takes 2.67 ns, whereas DSB-CORDIC takes 2.44 ns (note that 1 iteration of Double stepping methods is equivalent to 2 iterations of other methods). This amounts to a speedup of about 42% (over Branching CORDIC, and Classical CORDIC, respectively). Likewise, for $n = 32$ bits, 2 iterations of the classical method take 5.18 ns, Original Branching CORDIC takes 4.24 ns whereas DSB CORDIC takes about 2.6 ns, a speedup of about 50% and 38% respectively (over classical and original branching CORDIC).

Classical methods do not use redundant representations and hence their delay can be expected to rise

Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03)
ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE

more substantially than that of methods that use redundant intermediate values. This is corroborated by the table. The benefits of branching methods will be more apparent at longer wordlengths and this is confirmed by the table. For example for 16 bits the classical double stepping came close to double stepping along with redundant digits. But at 32 bits the delay of DSB method does not go up much whereas the delay of classical method increases more substantially.

The ratios of overall gate counts for the original Branching and DSB methods are 1.35 and 1.38 (for 16 and 32 bits). The corresponding delay ratios are 1.72 and 1.64 which indicates that the speedup is linear (or better) as a function of hardware-added. In other words, increasing the hardware by a factor k also increases the speed by k or more. This indicates a very gainful, efficient use of the extra hardware DSB method adds to enhance the original Branching method. Speedups that are linearly proportional to hardware are not that common (this especially true in multi-processor systems: increasing the hardware by a factor k almost always results in a speedup by a factor substantially less than k).

Finally note that the there is no reason to force the Zeroer to work in tandem with the rotator. Since the Zeroer is faster, it can be allowed to race ahead, simply depositing the signs into a register and the rotator could then come along later and consume the signs. This implies two different clock cycles for the zeroer and the rotator. Once the zeroer is decoupled from the rotator, hybrid combinations are feasible. For instance, if the high hardware cost of double stepping rotators is not affordable it is possible to use the Zeroer from the original branching method together with the classical rotator.

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] D. S. Phatak, "Double Step Branching CORDIC: A New Algorithm for Fast Sine and Cosine Generation," *IEEE Transactions on Computers*, vol. TC–47, pp. 587–602, May 1998.

[3] I. Koren, *Computer Arithmetic Algorithms*. Brookside Court Publishers, Amherst, Massachusetts, 1998.

[4] 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.

[5] 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. (Unabridged version available at http://www.cs.umbc.edu/ phatak/pblications/hsdtrc.pdf).

[6] 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.

[7] D. S. Phatak, T. Goff, and I. Koren, "Constant-time Addition and Simultaneous Format Conversion Based on Redundant Binary Representations," *IEEE Trans. on Computers*, vol. TC–50, pp. 1267–1278, Nov. 2001.

[8] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Taniguchi, and N. Takagi, "Design of high speed MOS multiplier and divider using redundant binary representation," *Proc. of the 8th Symposium on Computer Arithmetic*, pp. 80–86, 1987.

Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP'03)
ISBN0-7695-1992-X/03 $17.00 © 2003 IEEE