The Dissertation Committee for Chang Yong Kang
certifies that this is the approved version of the following dissertation:

CORDIC-Based High-Speed Direct Digital Frequency Synthesis

Committee:

Earl E. Swartzlander, Jr., Supervisor

Mircea D. Driga

Raghunath K. Rao

Nur A. Touba

Martin D. F. Wong

CORDIC-Based High-Speed Direct Digital Frequency Synthesis

by

Chang Yong Kang, B.E., M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2003

To my parents.

Acknowledgments

I am deeply indebted to Dr. Earl E. Swartzlander, Jr. for the guidance and patience he rendered throughout my graduate study at the University of Texas at Austin.

I am also grateful to Dr. Mircea D. Driga, Dr. Raghunath K. Rao, Dr. Nur A. Touba, and Dr. Martin D. F. Wong for serving on my dissertation committee. Their insightful comments consistently helped me sharpen my research.

I wish to express my sincere appreciation to Cirrus Logic, Inc. for allowing me to use industrial-strength tools as well as for its generous financial support.

My special thanks are due to my wife, Sun Ryung Suh, for her support, patience, sacrifice, and love. In times of difficulties, she has always been wise and never lost her confidence in me.

Finally, I would like to thank my son, Jae Young, for being the strongest motivation of my life.

CORDIC-Based High-Speed Direct Digital Frequency Synthesis

Publication No. _____

Chang Yong Kang, Ph.D.
The University of Texas at Austin, 2003

Supervisor: Earl E. Swartzlander, Jr.

The circular-mode CORDIC (coordinate rotation digital computer) algorithm is analyzed in the context of direct digital frequency synthesis (DDFS). It is shown how the CORDIC parameters should be selected to meet given DDFS parameters, and, through simulations, it is demonstrated that jamming outperforms truncation as a datapath quantization scheme, making it an attractive alternative to rounding. It is also shown that the CORDIC output can be made exact to the digits by an additional rounding process, which is especially useful for DDFS applications where the CORDIC output is truncated to the final digital-to-analog converter (DAC) width.

Variations of the basic circular-mode CORDIC algorithm are investigated, and previous works for fast DDFS implementation based on those algorithms are discussed. A new DDFS architecture based on the differential CORDIC (DCORDIC) algorithm is proposed. The proposed architecture allows a bit-level pipelining in the angle path by implementing a two-dimensional

systolic array. Unlike other fast DDFS architectures, it incorporates the phase accumulator in the bit-level pipelining framework so that a bottleneck-free datapath throughout the whole system is achieved. The sequential implementation and the hybrid implementation of the architecture for area-sensitive and low-latency designs, respectively, are proposed for further research.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Direct digital frequency synthesis, or DDFS, is a digital technique for generating sinusoids. Unlike conventional analog oscillator structures, DDFS can be applied where fast frequency switching, fine tuning, and a coherent phase relationship among sinusoids are required [1][2][3]. Most traditional DDFS systems follow the general architecture shown in Figure 1.1 [4].

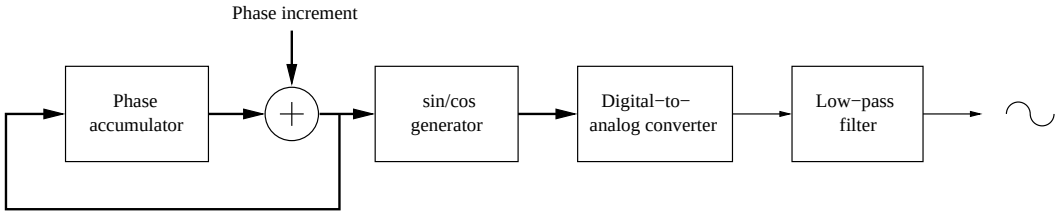


Figure 1.1: General DDFS architecture.

The sine/cosine generator in the DDFS architecture generally assumes one of the two forms: a ROM table for table-lookup or an arithmetic unit to actually evaluate the function. The former can be found in most of the early implementation [3][5][6]. The major difficulty with this approach is that the size of the ROM table increases exponentially with the width (i.e., the resolution) of the output. Even though there have been several techniques proposed

to mitigate this problem [3][7][8], which basically partition the ROM table to reduce the size, the exponential relationship still persists and a resolution higher than 14 bits is currently considered impractical [8].

Using an arithmetic unit is an alternative to the ROM table approach that has been getting attention recently. Especially, the CORDIC algorithm [9] has been dominant in this approach because it only requires shift-and-add operations while other approaches based on functional approximations generally involve more complicated arithmetic operations such as multiplication [10]. However, there are two major complications in using CORDIC for DDFS applications. First, the impact of the algorithm on the precision may appear implicit due to the iterative nature of the algorithm. This is in contrast to the ROM table approach where pre-calculated sine/cosine values are stored in a ROM, and therefore, the only error that is introduced by the sine/cosine generator is the rounding error. In ROM-based configurations, it has been shown that, as the word size of the phase quantization (i.e., the width of the ROM table address input) and the word size of the amplitude quantization (i.e., the width of the ROM table output) are increased by 1 bit each, there is a 6-dB reduction in the spurs (i.e., undesired frequency components) of the final output sinusoid [7][11]. Spurious-free dynamic range (SFDR) is defined as the ratio of the amplitude of the desired frequency component to that of the largest spur, and considered one of the most important performance measures of any DDFS system. Figure 1.2 shows an example of the SFDR characteristic of a commercial DDFS VLSI from Analog Devices [12]. Second, the sequential

nature of the CORDIC algorithm makes it hard to achieve high throughput.

In the following chapters, a brief introduction to the CORDIC algorithm is presented and the finite-precision effects of the algorithm are analyzed in depth in the context of DDFS. A brief survey of the variations of the CORDIC architecture for high-speed operation is also presented. Finally, a novel DDFS architecture based on the differential CORDIC algorithm is proposed and related experimental results are presented.

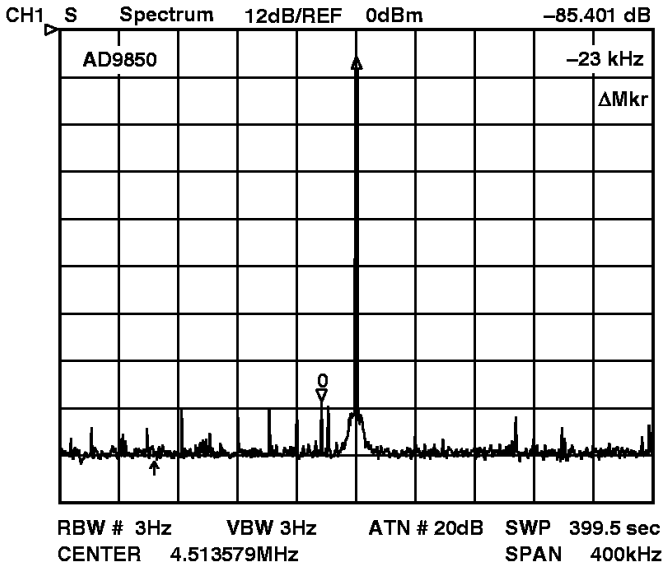


Figure 1.2: Spurious-free dynamic range characteristic of Analog Devices AD9850. Input clock at 20.5MHz and output sinusoid at 4.5MHz.

Chapter 2

Analysis of the CORDIC algorithm for DDFS*

2.1 Circular-mode CORDIC

The CORDIC algorithm was first introduced by Volder [9]. When used as the sine/cosine generation engine in DDFS, CORDIC uses the circular mode. In circular-mode CORDIC, a vector is rotated by a predetermined sequence of angles so that the overall summation of the rotated angles approaches the given angle argument. More precisely, circular-mode CORDIC is specified by the following set of equations that describes an iterative relationship.

$$\boldsymbol{V}_{i+1} = \boldsymbol{R}_i \boldsymbol{V}_i$$

$$z_{i+1} = z_i - \sigma_i \alpha_i$$

where:

$$i = 0, 1, 2, \ldots, n-1,$$

$$\boldsymbol{V}_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix},$$

$$\boldsymbol{R}_i = \begin{pmatrix} 1 & -\sigma_i \cdot 2^{-i} \\ \sigma_i \cdot 2^{-i} & 1 \end{pmatrix} = k_i \begin{pmatrix} \cos\alpha_i & -\sigma_i \sin\alpha_i \\ \sigma_i \sin\alpha_i & \cos\alpha_i \end{pmatrix},$$

$$\alpha_i = \arctan 2^{-i}, \text{ and } k_i = \sqrt{1 + 2^{-2i}}$$

*This chapter is based upon a paper previously published by the author [13].

In the forward circular mode, z_0 is set to the angle value for which the sine/cosine is to be evaluated. $\sigma_i = \text{sign}\,(z_i)$.

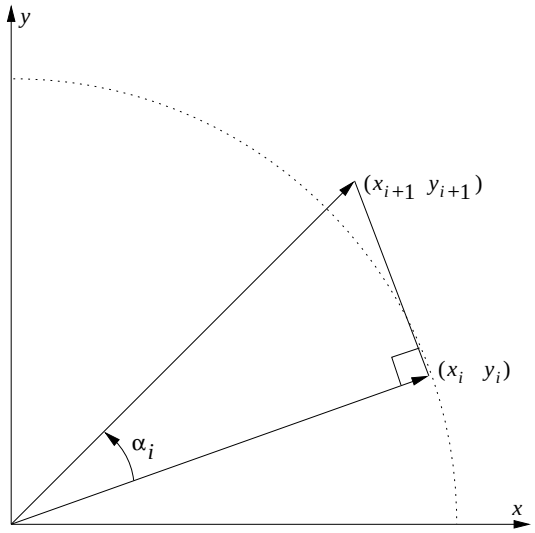


Figure 2.1: A CORDIC rotation in the forward circular mode.

After n iterations, there have been n pseudo-rotations, and $\boldsymbol{V}_n$ is the true rotation result multiplied by the factor K_n.

$$K_n = \prod_{i=0}^{n-1} k_i$$

Therefore, to get the true rotation result, it is necessary either to multiply $\boldsymbol{V}_n$ by $1/K_n$ (postscaling), or to use $1/K_n$ as the initial value of x_0 (prescaling).

The actual implementation of the CORDIC algorithm only requires adders, shifters, and a ROM table to hold α_i values. Figure 2.2 shows an example of the sequential implementation of the circular-mode CORDIC al-

gorithm that only requires three adders, two shifters, and one ROM table. In the figure, bold lines represent word-wide data paths.

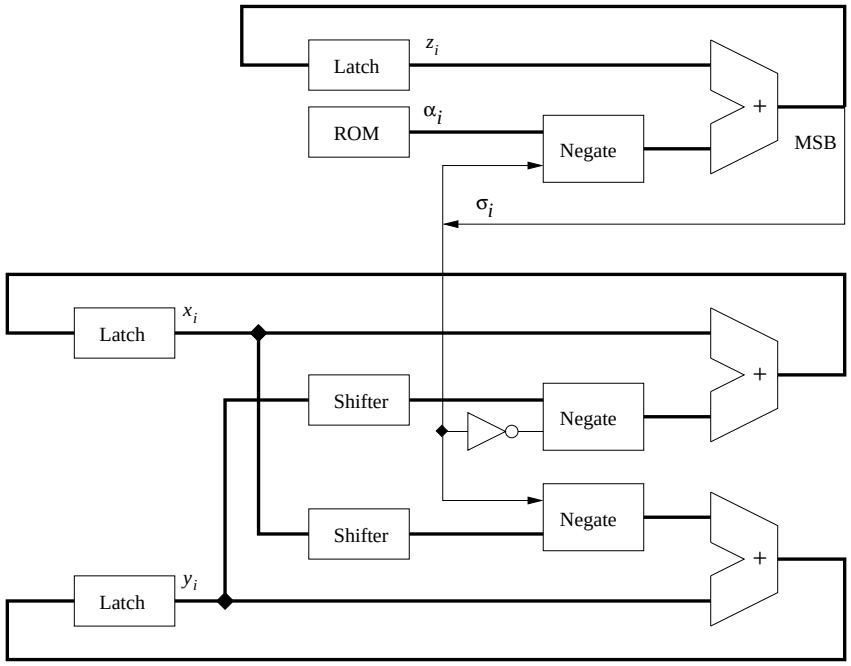


Figure 2.2: Sequential implementation of circular-mode CORDIC.

2.2 Angle precision, number of iterations, and datapath width

In applying circular-mode CORDIC to DDFS, it is crucial to understand how seemingly independent parameters can affect each other. First, it is necessary to determine how many iterations are necessary to properly resolve a radian angle in b fractional bits. The unit in the least precision (ulp) 2^{-b} in the angle representation is meaningful only when it can be resolved by

the iteration. That is, at the final iteration where $i = n - 1$, α_{n-1} should be less than or equal to 2^{-b}. This ensures that the final error in angle is less than 2^{-b}. The value $\alpha_i = \arctan 2^i$ can be approximated as 2^{-i} for large values of i. Therefore, $2^{-(n-1)}$ should be less than or equal to 2^{-b}. So, the minimum n is $b + 1$, which is the minimum number of iterations to correctly resolve the given angle.

In most DDFS applications, however, the angle is given in a normalized format such that $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right)$ is scaled to $\left[-\frac{1}{2}, \frac{1}{2}\right)$, because angle accumulator overflow is awkward to deal with in the radian format. When the normalized format is used, the angle table should also contain normalized entries. Suppose the angle is given by b fractional bits. Then, the normalized angle's *ulp* is 2^{-b} and it should be resolved by α_{n-1}.

$$\alpha_{n-1} \leq 2^{-b}$$

But,

$$\alpha_{n-1} = \frac{\arctan 2^{-(n-1)}}{\pi} \approx \frac{2^{-(n-1)}}{\pi}$$

Therefore,

$$\frac{2^{-(n-1)}}{\pi} \leq 2^{-b},$$

which means that the minimum n is b.

Finally, if n iterations are to be made, the datapath width w should be at least $n - 1$ (fractional). This is because, for n iterations, the largest shift is $n - 1$, and only when $w \geq n - 1$, can the largest shift contribute to the iteration.

2.3 Finite precision effects in CORDIC

In the framework shown in Figure 1.1, the CORDIC sine/cosine generator is another source of error because of its arithmetic nature, in addition to the phase quantization error, which is due to the finite precision that represents the angle, and the amplitude quantization errors, which is due to the finite precision that represents the sine/cosine value. This is in contrast to the ROM lookup table sine/cosine generator where there is no computation involved in sine/cosine generation since pre-computed values are utilized. It is generally known that the CORDIC algorithm approximately adds one bit of precision at each iteration, and it has been observed that the error that occurs from CORDIC arithmetic can be further decomposed into two parts; the quantization error in the datapath and the angle approximation error in the representation of the given angle by the summation of the prescribed angles [14]. The quantization error can be shown to have the bound [14]

$$|\boldsymbol{E}_n| \leq |\boldsymbol{e}_n| + \sum_{j=0}^{n-1} \Big(\prod_{i=j}^{n-1} k_i \Big) |\boldsymbol{e}_j| \tag{2.1}$$

where: $\boldsymbol{E}_n$ is the total error vector due to the quantization at iteration n, and $\boldsymbol{e}_n$ is the error vector incurred from iteration n. The key observation here is the total error passed to the next iteration stage is the sum of the quantization error at the given stage and the error passed from the previous stage. Note that the bound for $|\boldsymbol{E}_n|$ is also a bound for the errors of both sine/cosine components, but the inverse is not necessarily true because the magnitude of a vector is always greater than or, at least, equal to its quadrature and

in-phase components.

In addition to the quantization error, there also is an angle approximation error because any given angle is being represented by a sum of angles that has a resolution of α_{n-1}. The relative error $\boldsymbol{A}_n$ due to the angle approximation has been shown to have the bound [14]

$$|\boldsymbol{A}_n| = \frac{|\dot{\boldsymbol{V}}_n - \bar{\boldsymbol{V}}_n|}{|\bar{\boldsymbol{V}}_n|} \leq 2\sin(\frac{\alpha_{n-1}}{2})$$

where $\dot{\boldsymbol{V}}_n$ is the exact CORDIC value without the error and $\bar{\boldsymbol{V}}_n$ is the CORDIC result that contains the angle approximation error. Because

$$|\bar{\boldsymbol{V}}_n| = \left| K_n \cdot \sqrt{\sin^2\left(\sum_{i=0}^{n-1} \sigma_i \cdot \alpha_i\right) + \cos^2\left(\sum_{i=0}^{n-1} \sigma_i \cdot \alpha_i\right)} \right| = K_n \ ,$$

the bound for the absolute angle approximation error becomes:

$$\text{Absolute angle approximation} = |\dot{\boldsymbol{V}}_n - \bar{\boldsymbol{V}}_n| \leq 2\sin(\frac{\alpha_{n-1}}{2}) \cdot K_n \qquad (2.2)$$

2.4 Error bounds for rounding, truncation, and jamming

For the datapath quantization, three techniques (rounding, truncation, and jamming) are considered. Of these strategies, rounding gives the smallest error bound. In general, rounding involves adding LSB/2 and truncating the resulting sum to LSB. Some more sophisticated rounding schemes, such as convergent rounding ("round to nearest even") and R* rounding ("round to nearest odd"), may also be employed to reduce the bias of the resulting

errors [15][16]. In the convergent and R* rounding schemes, the treatment for the case where the digits to be rounded away have bit values $1000\cdots$ is different in that it is rounded to the nearest even and odd values, respectively. In any case, when there are w fractional bits available in the datapath, the largest error that is introduced by rounding is $2^{-(w+1)}$. If truncation is employed instead, the bound becomes 2^{-w}. The jamming strategy is to truncate and then force the LSB to 1 [16]. The bound in this case is also 2^{-w}.

In rounding, despite many desirable characteristics, a complication arises from the fact that the extra addition can require a significant amount of time for carry propagation. Because of this reason, truncation has been more widely used. In truncation, w bits are kept without any additional operation. However, truncation not only doubles the error bound, but, in case of two's complement representation, also causes a DC bias in the output because it always truncates toward negative infinity. In this regard, jamming, in general, is a better strategy. It can be shown that jamming produces results which are sufficiently unbiased, though the error bound remains the same as truncation [17]. Because the error at each iteration is unbiased, it is expected that jamming produces a final CORDIC output with a smaller total error than that produced by truncation.

Once a quantization scheme is selected, Equations (2.1) and (2.2) are used to evaluate the error bound of the CORDIC sine/cosine generator. However, no matter which quantization scheme is used, rounding can be used for x_0 in case of prescaling. In general, prescaling is preferred over postscaling due

to the simplicity of implementation, even though postscaling tends to produce smaller errors [14][18]. Also note that, at the second iteration when quantizing x_0 and y_0, there can be no error introduced because x_0 and y_0 are valid within the datapath and just added to yield x_1 and y_1. Considering these observations, bounds of $|\boldsymbol{e}_j|$ can be obtained as follows. For rounding,

$$|\boldsymbol{e}_0^{round}| = \text{absolute error in } \frac{1}{K_n} \tag{2.3}$$

$$|\boldsymbol{e}_1^{round}| = 0 \tag{2.4}$$

$$|\boldsymbol{e}_j^{round}| = \sqrt{2} \times 2^{-(w+1)},\ j = 2, 3, \ldots, n \tag{2.5}$$

For truncation, the number of bits truncated depends on the index j.

$$|\boldsymbol{e}_0^{trunc}| = \text{absolute error in } \frac{1}{K_n} \tag{2.6}$$

$$|\boldsymbol{e}_1^{trunc}| = 0 \tag{2.7}$$

$$|\boldsymbol{e}_j^{trunc}| = \sqrt{2} \sum_{k=w+1}^{w+j-1} 2^{-k},\ j = 2, 3, \ldots, n \tag{2.8}$$

For jamming, the errors are given as below.

$$|\boldsymbol{e}_0^{jam}| = \text{absolute error in } \frac{1}{K_n} \tag{2.9}$$

$$|\boldsymbol{e}_1^{jam}| = 0 \tag{2.10}$$

$$|\boldsymbol{e}_j^{jam}| = \sqrt{2} \times 2^{-w},\ j = 2, 3, \ldots, n \tag{2.11}$$

2.5 Choosing the number of iterations and the datapath width

When the number of iterations n is given, the angle approximation error bound can be obtained by Equation (2.2). The number of iterations n also requires that the datapath width w be greater than or equal to $n-1$. The total error bound is the summation of the angle approximation error bound, which is a function of n, and the quantization error bound, which is a function of w, and as w approaches infinity, the total error bound becomes the angle approximation error bound. Based on this analysis, it is evident that there should be two steps involved in determining n and w; first determine n so that the angle approximation error bound is lower than the desired total error bound, and then select w so that the summation of the angle approximation and the quantization error bounds is also lower than the desired total error bound. In general, above a certain minimum n, there is a trade-off between n and w.

2.6 Number of effective bits versus number of exact bits

The overall error can be re-interpreted as the number of effective bits in the output.

$$\text{Number of effective bits in fraction} = -\log_2\left(|\boldsymbol{E}_n + \boldsymbol{A}_n|\right) \tag{2.12}$$

This interpretation reveals how accurate the output is in terms of the number of bits. However, it should be noted that the number of effective bits, in

general, is different from the number of accurate bits. The following examples can help to convey an insight into this situation. Let $X = -x_0 + \sum_{i=1}^{\infty} x_i \cdot 2^{-i}$ be a reference number that has an infinite precision, and $Y = -y_0 + \sum_{j=1}^{5} x_i \cdot 2^{-j}$ be a quantized number that has only 5 fractional bits. First, suppose $X = 0.10011111\cdots$ and $Y = 0.10010$. In this case, the error between the two numbers is $\sum_{k=5}^{\infty} 2^{-k} = 2^{-4}$, and therefore the number of effective bits of Y is 4, which is the same as the number of exact bits. As a second example, consider $X = 0.10001000\cdots$ and $Y = 0.01111$. In this case, the error between the two numbers is $0.0001 = 2^{-4}$, and therefore the number of effective bits is again 4, but this time the number of exact bits is 0.

Unfortunately, this is not a desirable feature in CORDIC-based DDFS applications because the CORDIC engine requires a datapath width that is wider than the required output precision (i.e., number of effective bits) and the DAC generally only has a width that is the same as the designed precision. In many cases, the output of the CORDIC engine is simply truncated to the DAC width, but, as illustrated in the second example above, this will spoil the originally intended precision. The only way to preserve the precision even after truncation is to make the CORDIC output exact, bit by bit, up to the desired precision, which calls for an additional rounding process on the CORDIC output. Suppose the error in the CORDIC output is $2^{-(b+1)}$. If the number is rounded at the $2^{-(b+1)}$ digit, this rounding may cause another error also bounded by $2^{-(b+1)}$. So, the overall error bound becomes 2^{-b}, and the number of bits required to represent this number is b. Note that to achieve an

error bound of 2^{-b} after rounding, it is necessary to start with an error bound of $2^{-(b+1)}$. In other words, this additional step of rounding requires that the error bound of the CORDIC output be smaller than the desired error limit by a factor of 2.

2.7 Simulations

To verify the analyses made so far, a set of simulations is performed. Specifically, for angle resolutions of 8, 12, and 16 bits (angles in the normalized format), different numbers of iterations, datapath widths, and quantization methods are tried to evaluate the effects of the angle approximation error and the quantization error. For each simulation, the bound in terms of the number of effective bits is calculated using Equations (2.1) through (2.12), and compared with the minimum number of effective bits (of the vector, not of either sine or cosine, to ensure a fair comparison). The minimum number of effective bits is obtained by applying the $-\log_2$ operation to the maximum error of all the possible 2^n samples, where n is the angle resolution. The simulation program is written in the C++ language [19], and uses double-precision floating point for the reference numbers in the error calculation. Tables 2.1 through 2.3 summarize the simulation results. The tables are formatted in such a way that each row has a set of associated attributes that relates the row to a specific quantization method, number of iterations, and whether it has calculated bounds or simulation results.

The following conclusions can be drawn from the simulation results. First, the overall error is the summation of the angle approximation error and the quantization error so that reducing the error by increasing the datapath width eventually saturates, making the overall error approach the angle approximation error. Second, the number of iterations must be greater, at least by one, than the desired output precision. For example, with 8 iterations, the maximum number of effective bits achievable is 7. Third, the jamming strategy for datapath quantization consistently performs better than truncation. In fact, its performance is rather close to rounding, and sometimes it even outperforms rounding. Note, however, that the calculated bounds are generally pessimistic because error cancellation is not accounted for in the calculation.

Table 2.1: Number of effective bits in fraction for 8-bit angle resolution.

Quantization	Iterations	Bound/ Result	Datapath width								
			8	9	10	11	12	13	14	15	30
Rounding	8	Bound	4.824	5.450	5.795	6.005	6.136	6.207	6.243	6.262	6.280
		Result	5.987	6.418	6.933	6.939	6.982	7.034	7.040	7.049	7.049
	9	Bound	4.983	5.809	6.349	6.719	6.975	7.117	7.199	7.237	7.280
		Result	6.130	6.966	7.838	7.807	7.948	8.019	8.072	8.098	8.095
Truncation	8	Bound	4.346	5.068	5.541	5.853	6.051	6.161	6.220	6.250	6.280
		Result	5.021	5.771	6.361	6.674	6.860	6.954	7.022	7.031	7.049
	9	Bound	4.374	5.258	5.928	6.434	6.798	7.017	7.145	7.209	7.280
		Result	4.897	5.943	6.781	7.314	7.553	7.859	7.975	8.021	8.095
Jamming	8	Bound	4.190	4.938	5.450	5.795	6.018	6.143	6.210	6.245	6.280
		Result	5.689	6.337	6.579	6.841	6.992	7.037	7.067	7.044	7.049
	9	Bound	4.214	5.110	5.809	6.349	6.742	6.984	7.127	7.199	7.280
		Result	5.689	6.616	7.234	7.628	7.808	7.985	8.103	8.079	8.095

Table 2.2: Number of effective bits in fraction for 12-bit angle resolution.

Quantization	Iterations	Bound/ Result	Datapath width								
			12	13	14	15	16	17	18	19	30
Rounding	12	Bound	8.462	9.090	9.582	9.874	10.078	10.174	10.225	10.251	10.280
		Result	9.689	10.152	10.620	10.868	10.929	10.985	10.984	10.994	11.000
	13	Bound	8.583	9.362	10.046	10.511	10.871	11.059	11.162	11.219	11.280
		Result	9.819	10.506	11.215	11.748	11.830	11.968	11.966	11.991	12.001
Truncation	12	Bound	7.831	8.580	9.206	9.632	9.934	10.095	10.183	10.230	10.280
		Result	8.429	9.193	9.908	10.403	10.684	10.835	10.923	10.956	11.000
	13	Bound	7.850	8.713	9.504	10.115	10.605	10.901	11.076	11.173	11.280
		Result	8.380	9.271	10.140	10.904	11.370	11.653	11.829	11.906	12.001
Jamming	12	Bound	7.719	8.486	9.133	9.582	9.903	10.078	10.174	10.225	10.280
		Result	9.015	9.920	10.406	10.798	10.892	10.964	10.975	10.989	11.000
	13	Bound	7.737	8.609	9.414	10.046	10.557	10.871	11.059	11.164	11.280
		Result	9.015	10.125	11.017	11.483	11.800	11.912	11.937	11.963	12.001

Table 2.3: Number of effective bits in fraction for 16-bit angle resolution.

Quantization	Iterations	Bound/ Result	Datapath width								
			16	17	18	19	20	21	22	23	30
Rounding	16	Bound	12.187	12.873	13.398	13.764	14.009	14.136	14.205	14.243	14.280
		Result	13.360	14.158	14.465	14.735	14.832	14.957	14.966	14.978	15.000
	17	Bound	12.287	13.104	13.797	14.343	14.754	14.988	15.124	15.203	15.280
		Result	13.550	14.412	14.847	15.373	15.687	15.889	15.914	15.956	16.000
Truncation	16	Bound	11.461	12.263	12.934	13.449	13.815	14.026	14.146	14.213	14.280
		Result	12.003	12.933	13.606	14.135	14.481	14.736	14.876	14.929	15.000
	17	Bound	11.475	12.368	13.176	13.865	14.420	14.782	15.008	15.140	15.279
		Result	11.947	13.005	13.861	14.544	15.089	15.483	15.742	15.854	16.000
Jamming	16	Bound	11.373	12.187	12.873	13.405	13.786	14.009	14.137	14.208	14.280
		Result	12.677	13.530	14.045	14.484	14.772	14.889	14.955	14.969	15.000
	17	Bound	11.387	12.287	13.104	13.807	14.377	14.754	14.991	15.131	15.279
		Result	12.677	13.745	14.254	15.022	15.548	15.794	15.878	15.936	16.000

While the tables summarize the worst case performance of the different quantization schemes, it is also important to observe how the quantization schemes behave in a statistical sense. The variance is an appropriate measure for this purpose since it is a measure of the noise power in the signal. When the number of observed samples is limited, as in the case under consideration, the

sample variance $\bar{\mathbf{v}}$ can be used as an unbiased estimate of the true variance [20].

$$\bar{\mathbf{v}} = \frac{1}{n-1} \cdot \sum_{i=1}^{n} (x_i - \bar{\mathbf{x}})^2$$

where x_i is the observed set of sample values, and $\bar{\mathbf{x}}$ is the sample mean of the values given by:

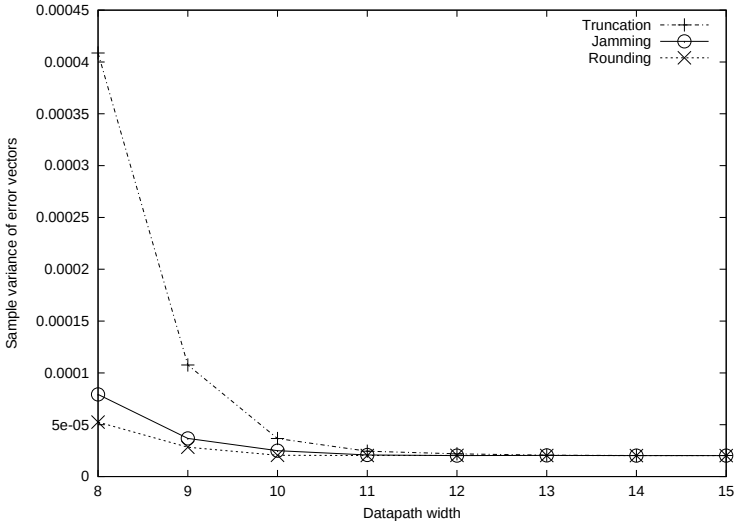
$$\bar{\mathbf{x}} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$$

Note, however, that CORDIC only evaluates over the $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right)$ range and uses the symmetry of the trigonometric functions to deal with other angles that fall outside the range. Therefore, the sample mean of the resulting errors, when calculated over one cycle, is always zero because the errors also appear in symmetry. Considering this, and the fact that the variance is for the in-phase/quadrature (i.e., cosine/sine) error vectors, the sample variance can be calculated by the following formula:

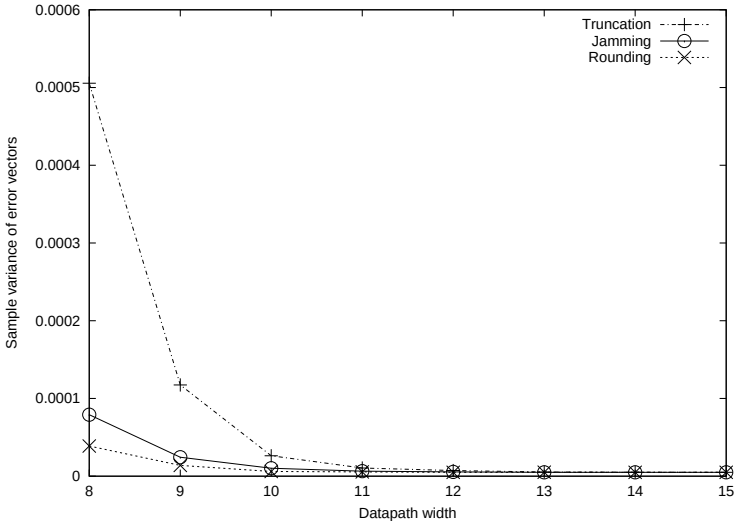
$$\bar{\mathbf{v}} = \frac{1}{n-1} \cdot \sum_{i=1}^{n} \left\{ (\text{cosine error in } i\text{-th sample})^2 + (\text{sine error in } i\text{-th sample})^2 \right\}$$

where n is to cover one cycle of the cosine/sine wave.

Figure 2.3 through Figure 2.5 compare the sample variances of the different quantization schemes, which again demonstrates that jamming consistently performs better than truncation in a statistical sense, too. It becomes obvious especially when the quantization error is dominant over the angle approximation error, i.e., when the datapath width is small relative to the number of iterations. Overall, jamming exhibits a comparable variance characteristic to rounding.

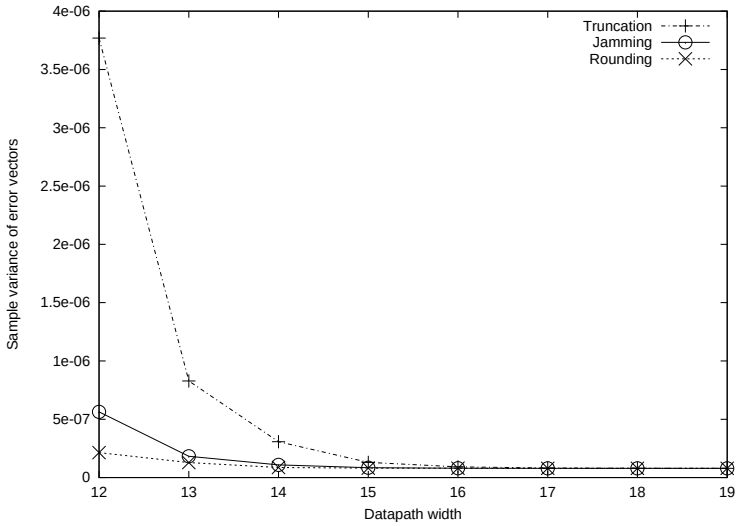


(a)

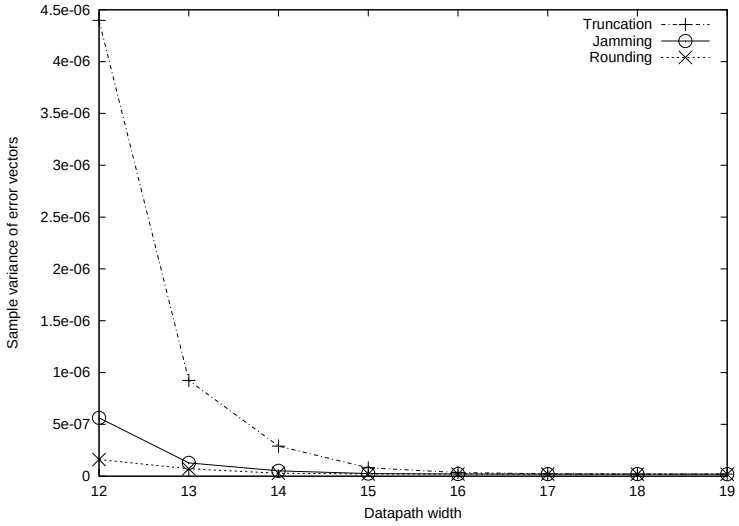


(b)

Figure 2.3: Error variance comparison for 8-bit angle resolution. With 8 iterations (a), and with 9 iterations (b).

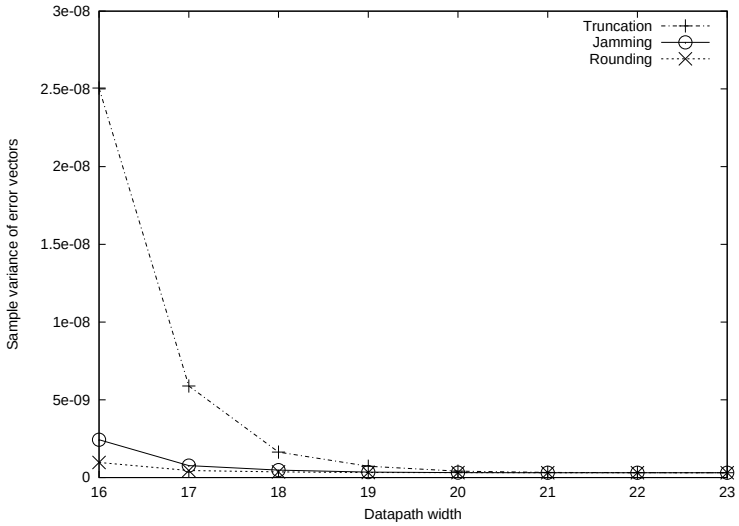


(a)

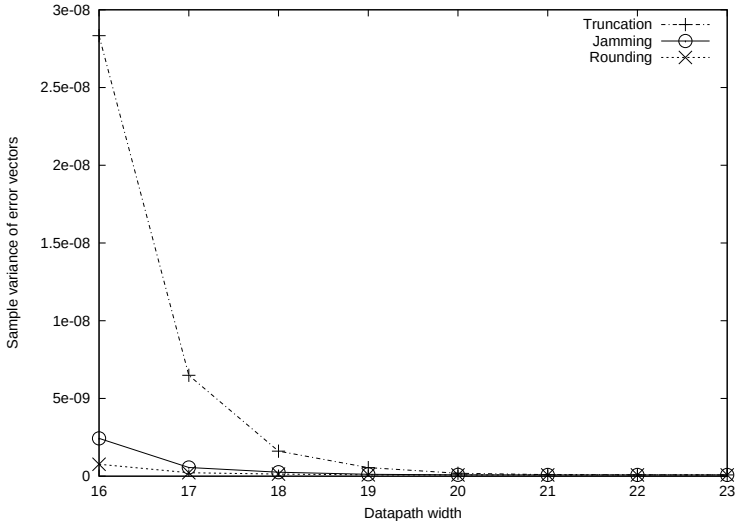


(b)

Figure 2.4: Error variance comparison for 12-bit angle resolution. With 12 iterations (a), and with 13 iterations (b).



(a)



(b)

Figure 2.5: Error variance comparison for 16-bit angle resolution. With 16 iterations (a), and with 17 iterations (b).

Finally, it is important to recognize how the output of the CORDIC engine can be properly truncated to the DAC width. As mentioned in the earlier analysis, for example, when Table 2.1 shows that the number of effective bits is 8.019, it does not necessarily mean that the output is exact in the first 8 fractional bits because the output has 13 fractional bits. To illustrate this point, additional simulations are performed for this example, and Table 2.4 summarizes the result. The table shows how the original CORDIC output with 8 effective bits in 13 fractional bits can be quantized to have 7 effective bits in 7 fractional bits. As shown in the table, this is only possible when the CORDIC output is rounded from the 8th digit, resulting in a number that has an error bound twice as big as the error bound in the CORDIC output before rounding, which again confirms the analysis presented. Note that, when rounded from the 8th digit, the number of effective bits of the *vector* can become smaller than 7, as in the example presented, because the magnitude of the error vector introduced by the quantization is generally greater than its quadrature and in-phase components.

Table 2.4: Number of bits versus number of effective bits.

		CORDIC output	Truncated	Rounded from 9th digit	Rounded from 8th digit
No. of bits (fractional)		13	8	8	7
No. of effective digits	Vector	8.019	6.959	7.441	6.789
	Cosine	8.11	7.237	7.5	7
	Sine	8.063	7.23	7.5	7.013

Chapter 3

CORDIC-based DDFS architectures

3.1 High-speed CORDIC algorithms

A major drawback of the CORDIC algorithm is that it is a decision-directed algorithm: to perform the next iteration, the sign of the angle at the current iteration needs to be known. This makes it difficult to employ a fast redundant number system in the angle representation, requiring instead a full carry-propagation addition in the angle calculation at each iteration. Considering that DDFS applications are generally more focused on high throughput than low latency, one may argue that it is still possible to realize high-throughput implementation by simply pipelining the angle and the data calculation paths. However, pipelined adders inherently introduce a skew in the output [21], and to compensate for the skew at each iteration stage, the implementation would require a huge number of flip-flops, leading to an almost impractical design. Also, low latency is desirable for such DDFS applications as frequency-hopping where a real-time on-the-fly frequency switching capability is necessary.

Since the inherent serialism in the decision-making process imposes such a severe penalty on performance, despite the elegance of the algorithm,

CORDIC didn't get much attention and was not considered mainstream. However, recent interests in developing application-specific signal processing systems have given CORDIC renewed attention since it is capable of evaluating a wide variety of elementary functions in a unified framework [22], and a number of papers have been published that deal with techniques to expedite the decision-making process [23][24][25][26][27][28]. One such effort is to allow a redundant number representation in the angle calculation but avoid the full carry-propagation in determining the sign of the number. Ercegovac and Lang [23] proposed this approach where only several most-significant digits (MSD) are inspected to determine the sign of the residual angle in a redundant form. In this approach, however, there are cases where the sign can not be determined, meaning that a rotation can not be made in the next iteration. This inability to determine the sign means that the residual angle is small enough at the iteration, and therefore the rotation may be skipped without hampering convergence. A complication, however, arises from the fact that the scale factor is no longer a constant because the number of rotations is not fixed. To address this problem, the authors proposed a method to calculate the scale factor in parallel with the CORDIC iterations. However, there are two major drawbacks in this approach. First of all, it requires a significant hardware overhead, which defeats the simplicity of the original CORDIC algorithm. Secondly, it mandates use of postscaling. Even though postscaling generally exhibits smaller computation errors than prescaling [14][18], prescaling is generally preferred where only the absolute angle calculation rather than a gen-

eral rotation capability, as in DDFS, is required because the implementation can be made without any multiplier by fixing the initial vector to the proper value. To improve this situation, Takagi, *et al.* proposed the double-rotating CORDIC method [24]. In this approach, each rotation is decomposed into two sub-rotations so that no-rotation can be performed by the two sub-rotations in opposite directions. With this modification, every iteration is guaranteed to have two sub-rotations so that the scale factor is a constant. However, this method essentially doubles the amount of hardware in the datapath, and the cycle time is also significantly increased. In the same paper, the authors also introduced an alternative method, called "correcting rotation method." In the correcting rotation method, only a certain number of digits are inspected to determine the sign, and even when it is not possible to decide, it rotates to one of the directions. To compensate for the potential error, a correcting rotation is inserted once every m-th iteration, where m depends on how many digits are inspected for sign determination. Later, this correcting rotation concept was further extended to the vectoring mode [25]. Another approach that utilizes correcting rotations in a different way was proposed by Timmerman, *et al.* [26] In this approach, the angle is directly recoded to rotation directions, and some of the rotation steps are repeated to compensate for the errors introduced in the recoding process. The authors also noticed that the second half of the iterations can be replaced by two parallel multiplications because the approximation $\arctan\theta = \theta$ holds for the given precision. A more general discussion regarding this potential use of the parallelism appeared in [27] and was later

extended by Wang, *et al.* to yield a hybrid architecture [28]. In the hybrid architecture, the circular-mode CORDIC implementation is divided into two parts: a front-end that computes the first $n/3$ iterations in the conventional sequential manner and a back-end part that processes the remaining $2n/3$ iterations in parallel. The authors noted the possibility of employing a ROM table for the front-end processor when the number of iterations is reasonably small and only a sine/cosine computation, as opposed to a general rotator, is necessary. While most of the approaches are focused, as discussed so far, on expediting the sign decision process of the residual angle in a redundant format or exploiting the arc-tangent approximation for later iterations, one unique approach, called differential CORDIC (DCORDIC), was proposed by Dawid and Meyr [29][30]. The DCORDIC algorithm transforms the original CORDIC angle calculation recursion in such a way that only absolute values of the angle are involved, and this algorithm transformation eventually leads to the implementation of the angle path using on-line arithmetic [31]: each angle calculation emits the result in the MSD-first manner and the next iteration stage processes data as the digits arrive. This implementation yields a two-dimensional systolic architecture [32][33] that is suitable for a low-latency and high-throughput design and highly regular for VLSI realization.

3.2 CORDIC-based DDFS architectures

One of the most noticeable features of DDFS is that it inevitably involves a feedback loop in the system. In the architecture shown in Figure 1.1,

the feedback loop is in the phase accumulator where the current angle output is recursively added to the phase increment to produce the next angle output. For ROM-based architectures, this is the most convenient way of introducing feedback. Unlike the ROM-based architectures, however, CORDIC-based DDFS systems can assume an alternative architecture that does not have a phase accumulator, as shown in Figure 3.1. In this architecture, the CORDIC engine performs as a rotator and the output is fed back to the input to be rotated by the given phase increment. However, this approach has several drawbacks. First of all, it becomes very difficult to pipeline the CORDIC datapath due to the feedback loop, so that a high-throughput design is not readily achievable. Second, it suffers from error build-up that can cause a decaying or growing waveform. Finally, implementing a rotator generally requires more hardware. A variant of this approach was proposed by Grayver and Daneshrad [34], where a CORDIC engine is used off line to compute sine/cosine values of the phase increment and then four multipliers are used to implement a rotator with feedback. However, this approach is not free from any of the drawbacks mentioned.

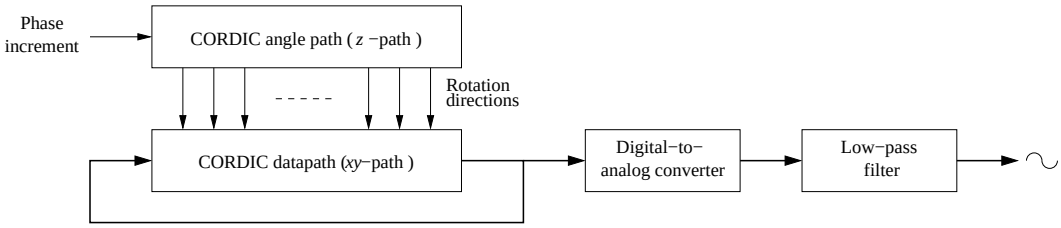


Figure 3.1: An alternative DDFS architecture with feedback in the datapath.

For the reasons explained, more successful CORDIC DDFS realizations have generally employed phase accumulators. Gielis, *et al.* reported a CORDIC DDFS that generates 10-bit samples at 540MHz rate [35]. This sample rate is achieved by means of a rather advanced process technology, a 1-μm 13-GHz triple-level interconnect bipolar process, without employing any special algorithm transformation. Madisetti, *et al.* reported a 16-bit 100MHz CORDIC DDFS system [36] based on the partitioned-hybrid CORDIC architecture [28] that enables it to perform the last $2n/3$ iterations in parallel. It also employs a small ROM table to avoid the first $n/3$ of the iterations as suggested in [28]. Note that, for some DDFS applications that need to rotate arbitrary inputs, this ROM table approach becomes impractical and the first $n/3$ sequential iterations are unavoidable [37][38]. One of the unique features of Madisetti's scheme is the angle recoding that is similar to the method of Takagi, *et al.* [24], which converts the angle representation from the radix set $\{0, 1\}$ to the radix set $\{-1, 1\}$. With this scheme, when the initial vector is properly pre-rotated by a fixed angle, each digit of the residual angle after the first $n/3$ rotations directly reflects the direction of the rotation for the rest of the iterations. The benefit of the recoding, however, is not immediately apparent. Without the recoding, the residual angle is available for parallel multiplication, so a lower-latency design would be achievable, or, if high throughput is the main concern, the multipliers can be pipelined to the desired level of granularity. Another hybrid approach proposed by Torosyan, *et al.* [39] supports this argument. As in Madisetti's scheme, the proposed architecture

employs a ROM table that contains sine/cosine values for the first $n/3$ bits of the angle. However, it does not use the CORDIC algorithm: instead it employs two stages of multipliers: the first of which performs a coarse rotation by the ROM table output, and the second of which performs a fine rotation by the remaining $2n/3$ bits of the angle. Note that the second rotation essentially performs the same task that is done by the CORDIC iterations in Madisetti's scheme. The first stage multiplier would not be necessary if only a DDFS were to be implemented instead of a mixer: the ROM table could directly contain pre-rotated values of an initial vector as in Madisetti's scheme.

To apply the tangent (or sine/cosine) approximation as in the hybrid approaches, the angle should be in radians. However, it is difficult to design a phase accumulator in the radian format since the overflow becomes awkward to deal with. This fact brings about a unique feature of the hybrid DDFS architectures: a π multiplier to convert the phase accumulator output from the range $[-1, 1)$ to the range $[-\pi, \pi)$. Note that in pure CORDIC rotations this is not necessary because the angle table entries can be normalized to the $[-1, 1)$ range. It is also noteworthy that, as the throughput requirement increases, the challenge is rather in the design of the phase accumulator. While the actual sine/cosine calculation datapath can generally be pipelined to the desired level of granularity, to increase the throughput of the phase accumulator is more difficult due to its feedback. In [40], it was reported that the phase accumulator limited the maximum system clock frequency. In [6] and [39], the phase accumulator was implemented in a conditional-sum adder form, which is

one of the most area-consuming adders [41], to achieve 200MHz and 300MHz throughput, respectively [42]. The phase accumulator may be implemented using a carry-save adder (CSA) with feedback, as suggested in [43]. This way, the throughput of the phase accumulator can be made very high. However, the vector-merging adder that follows the CSA should be pipelined to yield the same level of throughput, and the pipelining necessitates a large number of flip-flops to compensate for the time-skew inherent in pipelined adders [21]. Alternatively, the phase accumulator itself may directly be pipelined at the bit-level [44] or at some group-carry level [45]. Unlike the CSA approach, the pipelined phase accumulators require that the phase increment be supplied in an LSB-first time-skewed manner, and generates the output also in an LSB-first time-skewed manner, which generally mandates time-skewing latch blocks at both input and output sides of the phase accumulator. In [44], however, an approach that avoids the input-side latches by loading the phase increment in a time-skewed manner was proposed.

Chapter 4

DDFS based on the differential CORDIC algorithm

The differential CORDIC (DCORDIC) algorithm was first introduced by Dawid and Meyr [29][30]. While other CORDIC speed-up techniques are based either on fast sign determination of the angle in a redundant form or on the use of the small angle approximation to avoid sequential iterations, the DCORDIC algorithm is unique in that it uses on-line arithmetic to perform the computations for the angle path through an algorithm transformation. On-line arithmetic was first proposed by Ercegovac and Lang [31] and can be characterized by the following features. First, it utilizes a redundant number system. Secondly, by utilizing a redundant number system, it can serially process and output data in an MSD-first fashion. This means that, when stages are cascaded, the processing time between two stages is overlapped in a one-digit time-skewed manner, thereby achieving high throughput. In the following sections, the DCORDIC algorithm is introduced in depth and the application of the algorithm to DDFS is discussed. In the discussion, a new phase accumulator architecture that is compatible with on-line arithmetic is proposed and related algorithm modifications are introduced.

4.1 Introduction to the differential CORDIC algorithm

The transformation of the circular-mode CORDIC algorithm into the DCORDIC algorithm is accomplished by introducing an intermediate variable in the z-path (angle path) as follows.

$$\hat{z}_{i+1} = \sigma_i \cdot z_{i+1} \tag{4.1}$$

Then, the following hold.

$$|\hat{z}_{i+1}| = |z_{i+1}| \tag{4.2}$$

$$\sigma_{i+1} = \sigma_i \cdot \hat{\sigma}_{i+1} \tag{4.3}$$

Using this intermediate variable, the original angle recursion of the circular-mode CORDIC algorithm can be transformed as follows. From the original recursion $z_{i+1} = z_i - \sigma_i \cdot \alpha_i$,

$$\sigma_i \cdot z_{i+1} = \sigma_i \cdot z_i - \alpha_i$$

Using Equation (4.1),

$$\hat{z}_{i+1} = |z_i| - \alpha_i$$

Using Equation (4.2),

$$\hat{z}_{i+1} = |\hat{z}_i| - \alpha_i$$

Taking absolute values,

$$|\hat{z}_{i+1}| = \big||\hat{z}_i| - \alpha_i\big| = \big||\hat{z}_i| + \tilde{\alpha}_i\big| \tag{4.4}$$

where $\tilde{\alpha}_i$ is a W-bit fractional two's complement number:

$$\tilde{\alpha}_i = -\alpha_i = -\tilde{\alpha}_{i0} + \sum_{j=1}^{W-1} \tilde{\alpha}_{ij} \cdot 2^{-j}, \quad \tilde{\alpha}_{ij} = 0 \text{ or } 1$$

Equations (4.4) and (4.3) define a new recursion relationship.

The beauty of this algorithm transformation is that it allows the use of on-line arithmetic in the angle path. The on-line implementation of Equation (4.4) is accomplished through two stages. In the first stage, $|\hat{z}_i| + \tilde{\alpha}_i$ is computed. With a proper redundant number representation, such as binary signed-digit (BSD) numbers [16][46][47], this step can be implemented effortlessly. Here the role of the redundancy is that it confines the carry propagation to within one digit so that the MSD-first addition can produce the output of the digit with an on-line delay of one, without having to worry about the future carries that might propagate from the later digits [48]. The second stage of Equation (4.4) is to implement an on-line absolute value calculation algorithm. In some redundant number representations, this also can be easily achieved. For example, in the BSD representations, the sign of a number corresponds to the sign of the first non-zero MSD, and negation of the number can be performed by flipping signs of non-zero digits [16][46][47]. Figure 4.1 illustrates the on-line implementation of Equation (4.4) as explained. The implementation also can be considered a two-dimensional systolic array.

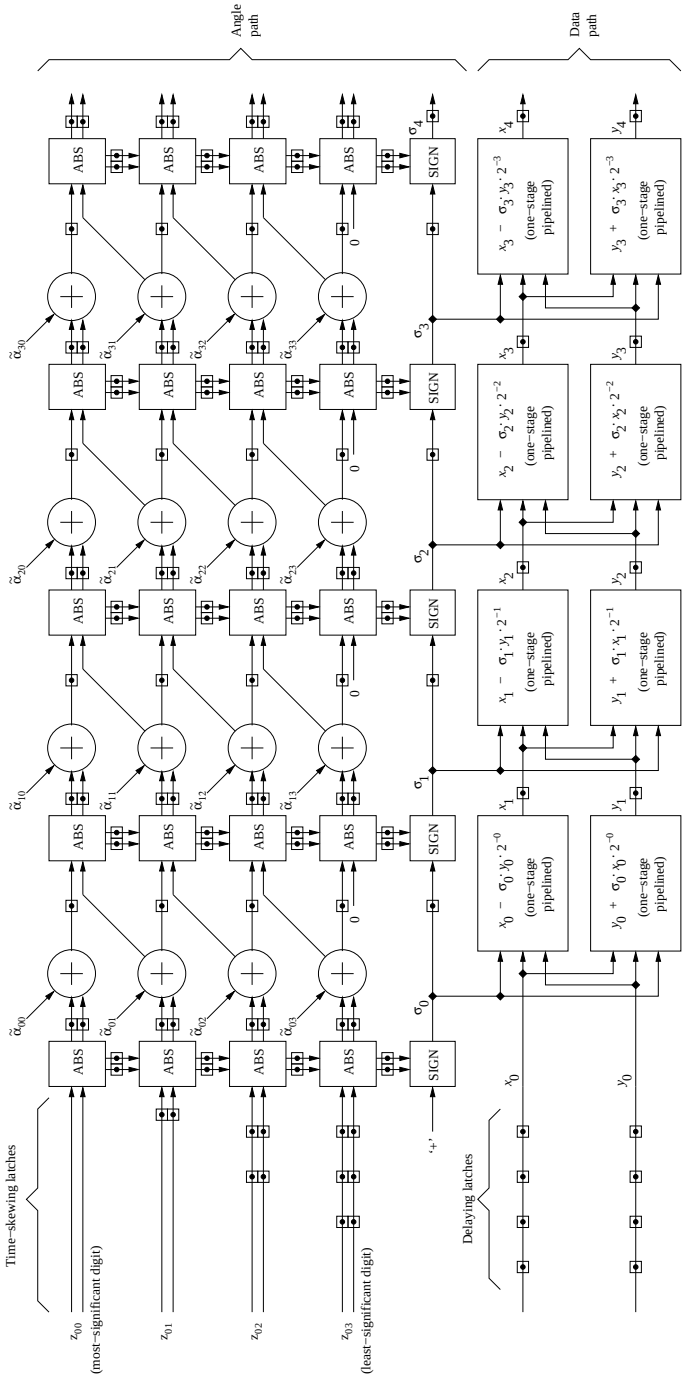


Figure 4.1: On-line implementation of the DCORDIC algorithm.

4.2 Using carry-save number representation

The DCORDIC algorithm requires a redundant number representation because it involves on-line arithmetic. In [30], the authors also indicated that the carry-save representation could be used for the same purpose. However, it is no longer on-line arithmetic in a strict sense when the carry-save number system is used. The complication arises because the negation of a carry-save number can require a full carry propagation that is caused by adding two *ulp*'s to the bit-wise inverted output, as in the negation of a two's complement number. Because of this reason, the authors indicated their preference for the BSD representations because they are well suited to the on-line arithmetic process in both sign determination and negation. However, for DDFS applications, the carry-save number system is preferred by far because the phase accumulator exploits the overflow behavior of the carry-save number representation, which is the same as in the two's complement number system. In [30], the authors introduced an MSD-first absolute value calculation algorithm for carry-save numbers, which has two fundamental limitations. First, it assumes that the carry-save number has not overflowed. Second, when it needs to negate, it defers the addition of *ulp*'s to the next stage. In general, the first limitation is not a big concern for other applications with a proper design of the word-length. However, it is incompatible with the output of the carry-save phase accumulator that actively exploits the overflow behavior. To circumvent this difficulty, first it is necessary to gain an insight into the overflow behavior of the carry-save number representation.

A fractional carry-save number X is represented by two vectors, a carry vector $C = -c_0 + \sum_{j=1}^{W-1} c_j \cdot 2^{-j}$ and a sum vector $S = -s_0 + \sum_{j=1}^{W-1} s_j \cdot 2^{-j}$, and the value of the number X is

$$X = -(c_0 + s_0) + \sum_{j=1}^{W-1} (c_j + s_j) \cdot 2^{-j} \tag{4.5}$$

From the expression, the range is clearly $-2 \leq X \leq 2 - 2^{-(W-2)}$. However, values outside of the range $-1 \leq X \leq 1 - 2^{-(W-1)}$ are considered to have overflowed because they are beyond the range that a W-bit two's complement number can represent. To determine if a carry-save number has overflowed or not may require a full-digit inspection. Specifically, the condition can be detected by adding C and S vectors and observing that the carry out of the MSD is different from the carry into the MSD. This is only possible when $c_0 = s_0$. When $c_0 = s_0 = 0$, the number has overflowed if and only if the first MSD where $c_N = s_N$, $N > 0$, has the value 2 ($c_N = s_N = 1$). In this case, the value that X represents, which is negative, is

$$X = -1 + \sum_{j=N+1}^{W-1} (c_j + s_j) \cdot 2^{-j} \tag{4.6}$$

On the other hand, when $c_0 = s_0 = 1$, the number has overflowed if and only if the first MSD where $c_N = s_N$, $N > 0$, has the value 0 ($c_N = s_N = 0$). In this case, the value that X represents, which is positive, is

$$X = \sum_{j=1}^{N-1} 2^{-j} + \sum_{j=N+1}^{W-1} (c_j + s_j) \cdot 2^{-j} \tag{4.7}$$

In all other cases where the number has not overflowed, the value that X represents is expressed by Equation (4.5).

Based on these observations, an MSD-first absolute value calculation and sign determination algorithm for a possibly overflowed carry-save number is derived as follows. When the MSD is 0 ($c_0 = s_0 = 0$), if the number has not overflowed, it is positive and there is no need for negation. On the other hand, if the number has overflowed, it is negative and should be negated. From Equation (4.6),

$$
\begin{aligned}
|X| &= -X && (4.8)\\
&= 1 - \sum_{j=N+1}^{W-1} (c_j + s_j) \cdot 2^{-j} && (4.9)\\
&= \sum_{j=1}^{N-1} 2^{-j} + 2^{-N} - \sum_{j=N+1}^{W-1} c_j \cdot 2^{-j} + 2^{-N} - \sum_{j=N+1}^{W-1} s_j \cdot 2^{-j} && (4.10)\\
&= \sum_{j=1}^{N-1} 2^{-j} + \sum_{j=N+1}^{W-1} (1 - c_j) \cdot 2^{-j} + 2^{-(W-1)} \\
&\quad + \sum_{j=N+1}^{W-1} (1 - s_j) \cdot 2^{-j} + 2^{-(W-1)} && (4.11)\\
&= \sum_{j=1}^{N-1} 2^{-j} + \sum_{j=N+1}^{W-1} \overline{c_j} \cdot 2^{-j} + \sum_{j=N+1}^{W-1} \overline{s_j} \cdot 2^{-j} + 2^{-(W-2)} && (4.12)
\end{aligned}
$$

It is interesting to note that, no matter whether the number has overflowed or not, the MSD-first absolute value calculation can emit digits without any modification until it encounters the first digit where $c_N = s_N$, and, from the digit and on, it inverts bits if $c_N = s_N = 1$. Likewise, when the MSD is 2 ($c_0 = s_0 = 1$), if the number has not overflowed, it is negative and needs to be negated. In this case, from Equation (4.5), $|X| = -X = 2 - \sum_{j=1}^{W-1}(c_j + s_j) \cdot 2^{-j}$, which is essentially equivalent to Equation (4.9) because $2 - \sum_{j=1}^{N}(c_j + s_j) = 1$.

On the other hand, if the number has overflowed, Equation (4.7) is valid. The MSD-first absolute value calculation for this case is the same as the case where the MSD is 0, except that the MSD output is the inversion of the input.

The algorithm for the cases where the MSD is 1 ($c_0 \neq s_0$) is derived in [30]. Basically, in this case, the number has not overflowed, and $X = -1 + \sum_{j=1}^{W-1}(c_j + s_j) \cdot 2^{-j}$. The number is positive if $c_N = s_N = 1$. In this case,

$$
\begin{aligned}
|X| &= X && (4.13)\\
&= -1 + \sum_{j=1}^{N-1} 2^{-j} + 2 \cdot \ 2^{-N} + \sum_{j=N+1}^{W-1} (c_j + s_j) \cdot 2^{-j} && (4.14)\\
&= \sum_{j=N+1}^{W-1} (c_j + s_j) \cdot 2^{-j} && (4.15)
\end{aligned}
$$

If $c_N = s_N = 0$, the number is negative, and the following hold.

$$
\begin{aligned}
|X| &= -X && (4.16)\\
&= 1 - \sum_{j=1}^{N-1} 2^{-j} - \sum_{j=N+1}^{W-1} (c_j + s_j) \cdot 2^{-j} && (4.17)\\
&= 2^{-(N-1)} - \sum_{j=N+1}^{W-1} (c_j + s_j) \cdot 2^{-j} && (4.18)\\
&= \sum_{j=N+1}^{W-1} (1 - c_j) \cdot 2^{-j} + 2^{-(W-1)} \\
&\quad + \sum_{j=N+1}^{W-1} (1 - s_j) \cdot 2^{-j} + 2^{-(W-1)} && (4.19)\\
&= \sum_{j=N+1}^{W-1} \overline{c_j} \cdot 2^{j} + \sum_{j=N+1}^{W-1} \overline{s_j} \cdot 2^{j} + 2^{-(W-2)} && (4.20)
\end{aligned}
$$

Therefore, the MSD-first absolute value calculation outputs 0 until it encounters a digit where $c_N = s_N$, and from *the next* digit on, it inverts bits if $c_N = s_N = 0$.

One special case of all the fractional digits being 1 should still be addressed to consider the algorithm complete. In this case, the sign of the number being processed will not be decided all the way down to the least-significant digit (LSD). This special case can further be divided into two different cases. First, if the MSD is 0 or 2, the number is positive and the algorithm may be terminated as usual because all the output digits are valid without any corrective actions. In the second case, where the MSD is 1, the sign of the number is negative. However, the output in this case is not correct until a *ulp* $2^{-(W-1)}$ is added to the output. This is in contrast to the other negation cases, which require the addition of two *ulp*'s, or equivalently, $2^{-(W-2)}$.

To summarize, the MSD-first absolute value and sign determination algorithm for the carry-save numbers that are possibly overflowed can be expressed by the following behavioral Verilog [49] code segment.

Algorithm [Absolute value calculation for carry-save numbers]

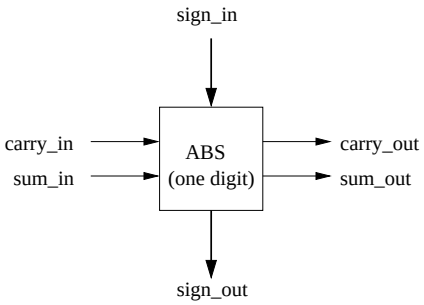
```
input [W-1:0] C,S;
output [W-1:0] A,B;
output sign; // 0 for plus, 1 for minus

reg [W-1:0] A,B;
reg [1:0] sign_temp;
integer sign,index;

always @(C or S)
 begin
```

```
sign = 0;
A[W-1] = 0;
B[W-1] = 0;

if (C[W-1]==S[W-1])
 begin

  sign_temp = 2'b00;

  for (index=W-2; index>=0; index=index-1)
   begin

    if (sign_temp[1]==0)
     if (C[index]==0 && S[index]==0) sign_temp = 2'b10;
     else if (C[index]==1 && S[index]==1) sign_temp = 2'b11;

    if (sign_temp[1]==0)
     begin
      A[index] = 1;
      B[index] = 0;
     end
    else if (sign_temp[0]==0)
     begin
      A[index] = C[index];
      B[index] = S[index];
     end
    else if (sign_temp[0]==1)
     begin
      A[index] = ~C[index];
      B[index] = ~S[index];
     end

   end

  end

else
 begin

  sign_temp = 2'b01;

  for (index=W-2; index>=0; index=index-1)
   begin

    if (sign_temp[1]==0)
     begin
      A[index] = 0;
      B[index] = 0;
     end
    else if (sign_temp[0]==0)
     begin
      A[index] = C[index];
      B[index] = S[index];
     end
    else if (sign_temp[0]==1)
     begin
```

```
        A[index] = ~C[index];
        B[index] = ~S[index];
       end

      if (sign_temp[1]==0)
       if (C[index]==0 && S[index]==0) sign_temp = 2'b11;
       else if (C[index]==1 && S[index]==1) sign_temp = 2'b10;

     end

    end

  if (sign_temp==2'b01)
   begin
    sign = 1;
    A = A + 1;
   end
  else if (sign_temp==2'b11)
   begin
    sign = 1;
    A = A + 1;
    B = B + 1;
   end
  else sign = 0;

end
```

The on-line, or more precisely, digit-pipelined realization of the algorithm involves designing of one absolute value calculation digit cell, to be called "ABS cell" hereafter, and cascading it multiple times to form a column, as illustrated in Figure 4.1. Although the algorithm may appear quite involved at first, the ABS cell can be implemented with a relatively simple logic circuitry as shown in Figure 4.2 and Figure 4.3.

The addition of $2^{-(W-1)}$ in the case of all the input digits being 1 can easily be incorporated into the LSD ABS cell. That is, the LSD ABS cell can directly output 1 for this particular case, thereby avoiding the correcting addition. This requires that the LSD cell be slightly different from the other cells. Also, the LSD cell need not output the 2-bit encoded sign; instead, a



(a)

sign_in*	carry_in	sum_in	carry_out	sum_out	sign_out *
MSD02	0	0	0	0	PLUS
	0	1	0	1	MSD02
	1	0	1	0	MSD02
	1	1	0	0	MINUS
MSD1	0	0	0	0	MINUS
	0	1	0	0	MSD1
	1	0	0	0	MSD1
	1	1	0	0	PLUS
PLUS	0	0	0	0	PLUS
	0	1	0	1	PLUS
	1	0	1	0	PLUS
	1	1	1	1	PLUS
MINUS	0	0	1	1	MINUS
	0	1	0	1	MINUS
	1	0	1	0	MINUS
	1	1	0	0	MINUS

* Sign is two−bit encoded. MSD02 : MSD=0 or 2, and sign undecided.
MSD1: MSD=1, and sign undecided. PLUS: sign is plus.
MINUS : sign is minus.

(b)

Figure 4.2: Symbol (a) and truth table (b) of the ABS cell.

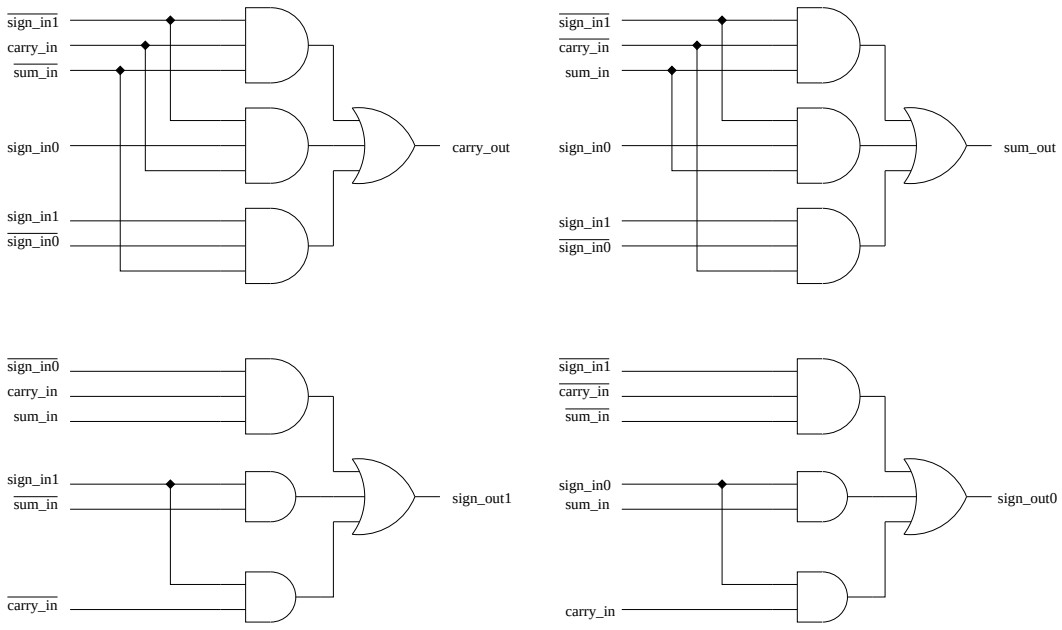


Figure 4.3: Possible circuit realization of the ABS cell.

separate sign decoder is required to output the final rotation direction.

While the addition of $2^{-(W-1)}$ for the special case can easily be handled as explained, the addition of $2^{-(W-2)}$ calls for a different treatment because, whereas the addition of $2^{-(W-2)}$ is equivalent to adding two *ulp*'s, there is only one carry input available at the LSD ABS cell. Therefore, normally, it will require another CSA column before the next ABS column. The addition of the new CSA column will not only increase the hardware overhead, but also increase the latency by one clock cycle per iteration. The extra CSA columns, however, can be avoided by adding an extra row instead at the LSD position, as explained in the following. Assume that the required angle resolution is $2^{-(W-1)}$, meaning that the angle can be represented by a W-digit fractional carry-save number. However, if $W+1$ bits are allowed in the representation, the angle recursion becomes:

$$\begin{aligned}
\hat{z}_{i+1} &= |\hat{z}_i| + \tilde{\alpha}_i && (4.21)\\
&= -(c_0 + s_0) + \sum_{j=1}^{W} (c_j + s_j) \cdot 2^{-j} \\
&\quad -\tilde{\alpha}_{i0} + \sum_{j=1}^{W-1} \tilde{\alpha}_{ij} \cdot 2^{-j} + \hat{\sigma}_i \cdot 2^{-(W-1)} && (4.22)\\
&= -(c_0 + s_0 + \tilde{\alpha}_{i0}) + \sum_{j=1}^{W-1} (c_j + s_j + \tilde{\alpha}_{ij}) \cdot 2^{-j} \\
&\quad +(c_W + s_W + \hat{\sigma}_i) \cdot 2^{-W} + \hat{\sigma}_i \cdot 2^{-W} && (4.23)
\end{aligned}$$

where:

$$
\begin{aligned}
|\hat{z}_i| &= -(c_0 + s_0) + \sum_{j=1}^{W} (c_j + s_j) \cdot 2^{-j} + \hat{\sigma}_i \cdot 2^{-(W-1)} \\
\tilde{\alpha}_i &= -\alpha_i = -\tilde{\alpha}_{i0} + \sum_{j=1}^{W-1} \tilde{\alpha}_{ij} \cdot 2^{-j}, \text{and} \\
\hat{\sigma}_i &= \begin{cases} 1, & \hat{z}_i < 0 \\ 0, & \text{otherwise} \end{cases}
\end{aligned}
$$

Note that, in Equation (4.23), all the terms are grouped by three and can be input to one CSA vector, except for the last term, which can be connected to the carry input of the $(i+1)$-th digit ABS cell. Therefore, the addition of $2^{-(W-2)}$ can be accomplished with a minimal overhead, by using an extra row with one-clock extra latency, as shown in Figure 4.4.

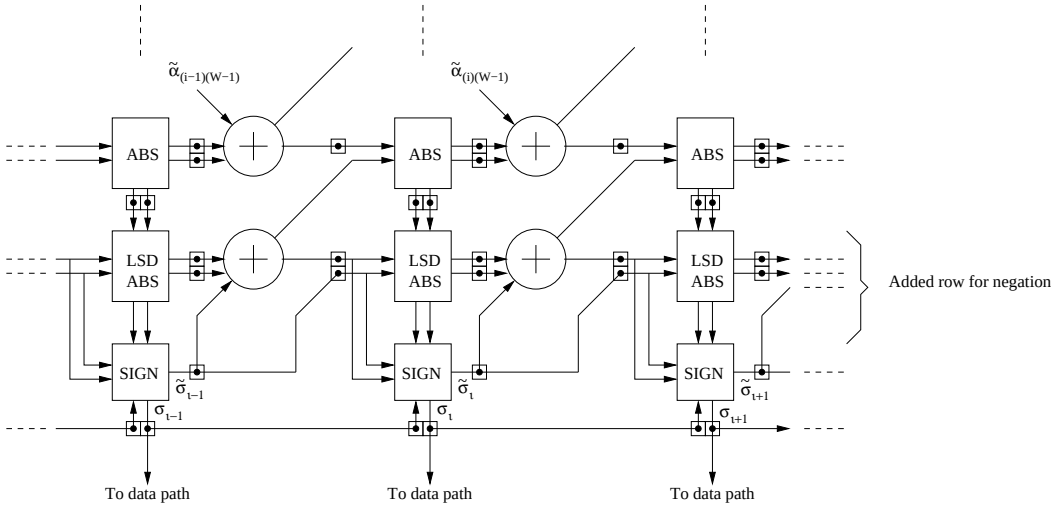


Figure 4.4: Adding an additional row for negating carry-save angle values.

Note that the sign decoder should also output the signal $\tilde{\sigma}_i$, which signifies the addition of $2^{-(W-2)}$. This signal is equivalent to $\hat{\sigma}_i$ except that it is zero for the special case of all digits being 1. Figures 4.5 through 4.7 show the symbol, the truth table, and a possible circuit realization of the LSD cell and the sign decoder. Even though the sign decoder seems to have a critical path longer than the others, it does not affect the system clock period because the LSD ABS cell and the sign decoder can exclusively use the whole clock period unlike the other digit cells.

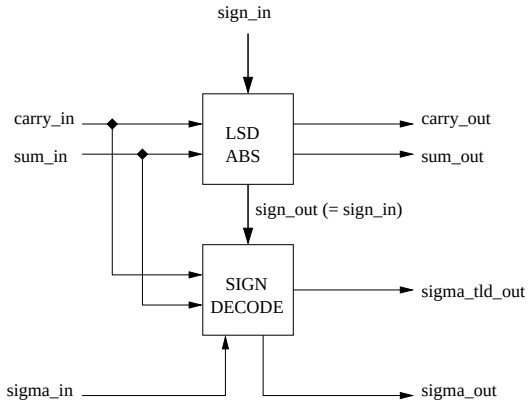


Figure 4.5: Symbol of the LSD ABS cell and the sign decoder.

sign_in*	carry_in	sum_in	carry_out	sum_out	sign_out *	sigma_hat	sigma_tld _out
MSD02	0	0	0	0	MSD02	0	0
	0	1	0	1	MSD02	0	0
	1	0	1	0	MSD02	0	0
	1	1	0	0	MSD02	1	1
MSD1	0	0	0	0	MSD1	1	1
	0	1	0	1	MSD1	1	0
	1	0	1	0	MSD1	1	0
	1	1	0	0	MSD1	0	0
PLUS	0	0	0	0	PLUS	0	0
	0	1	0	1	PLUS	0	0
	1	0	1	0	PLUS	0	0
	1	1	1	1	PLUS	0	0
MINUS	0	0	1	1	MINUS	1	1
	0	1	0	1	MINUS	1	1
	1	0	1	0	MINUS	1	1
	1	1	0	0	MINUS	1	1

* Sign is two−bit encoded. MSD02 : MSD=0 or 2, and sign undecided.
MSD1: MSD=1, and sign undecided. PLUS: sign is plus. MINUS : sign is minus.

Figure 4.6: Truth table of the LSD ABS cell and the sign decoder.

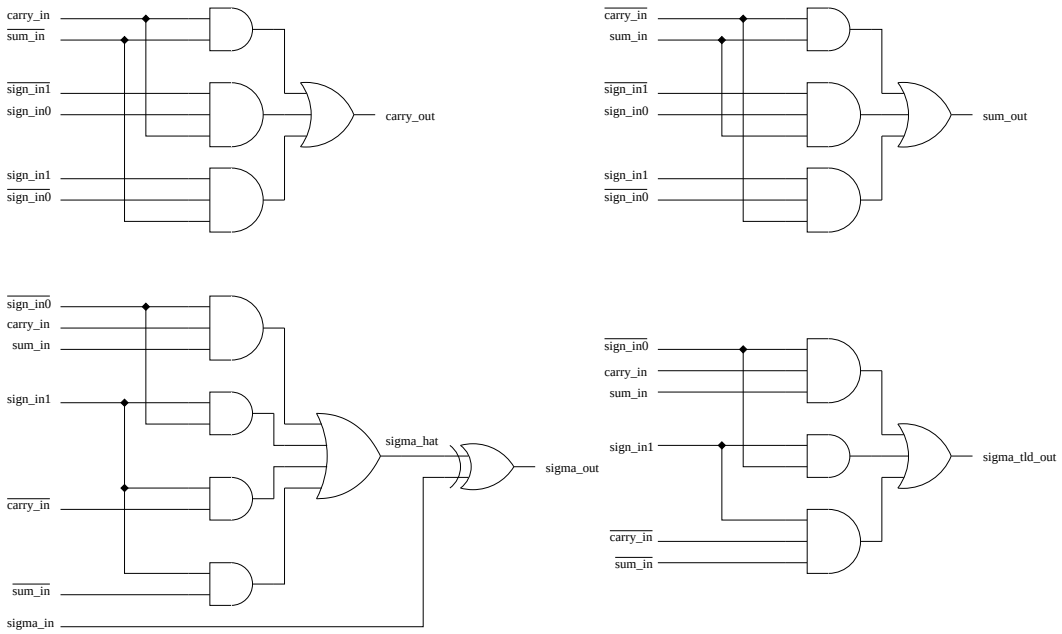


Figure 4.7: Possible circuit realization of the LSD ABS cell and the sign decoder.

4.3 Carry-save on-line phase accumulator

One major disadvantage of the DCORDIC architecture is that it requires a large number of latches. In fact, all the latches that are internal to the angle path are essential to the bit-level pipelining and can only be reduced by changing the pipeline granularity. The pipeline granularity can be systematically adjusted by grouping cells in a square form at the expense of throughput. Another source of latch-consumption, which appears more wasteful, is the input skewing block. The task of the input skewing block is simply to generate a time-skew so that the input to the angle path is compatible with the MSD-first format of the on-line absolute value calculation. When the phase accumulator is implemented with parallel feedback, as shown in Figure 4.8, this time-skewing block is unavoidable, so that an alternative scheme that skews the phase increment $P = -p_0 + \sum_{j=1}^{M-1} p_j \cdot 2^{-j}$ at the loading time is devised: loading the phase increment argument in the MSB-first time-skewed manner essentially creates the same effect as if the initial angle in the accumulator were skewed to begin with (Figure 4.9).

The alternative method proposed in Figure 4.9 is somewhat similar to the scheme introduced in [44] but different in that it skews the argument in the MSD-first manner, which complicates the feedback process because it introduces a one-stage pipeline within the loop. Note that this pipeline is inherent to carry-save arithmetic because the output of the current digit is not complete until the next digit outputs its carry. To show how to circumvent this complication, one-digit slice of the accumulator with feedback is illustrated in

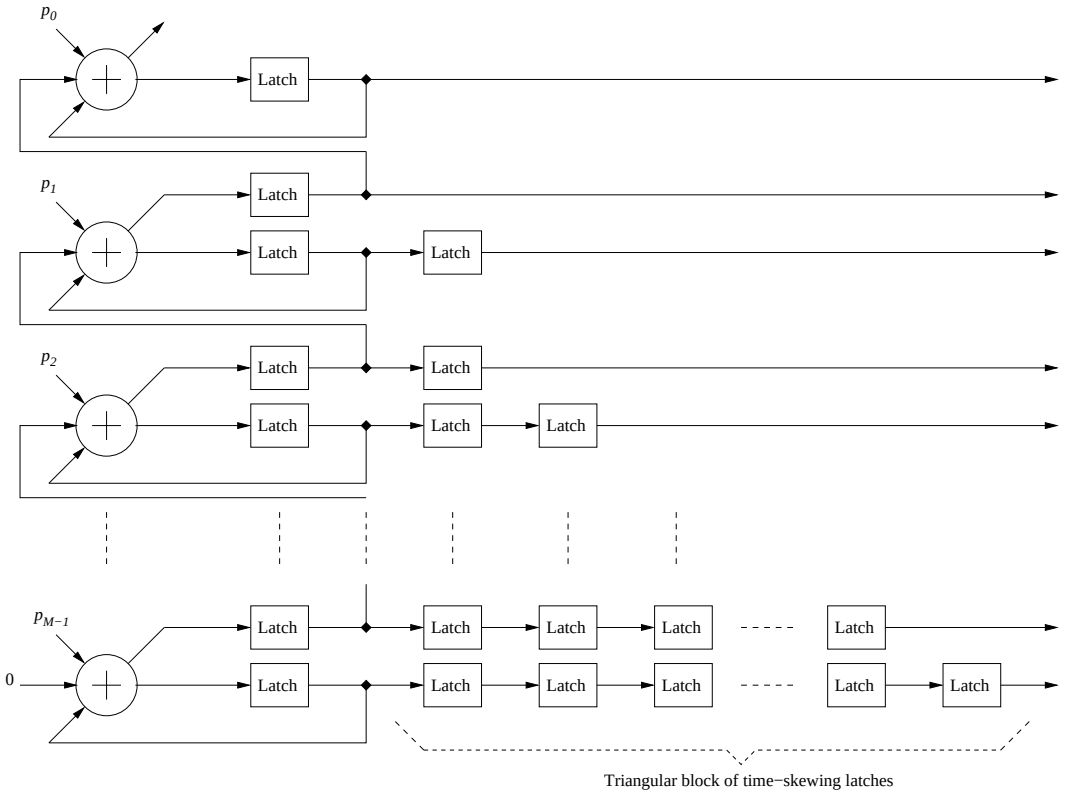


Figure 4.8: Phase accumulator with parallel feedback.

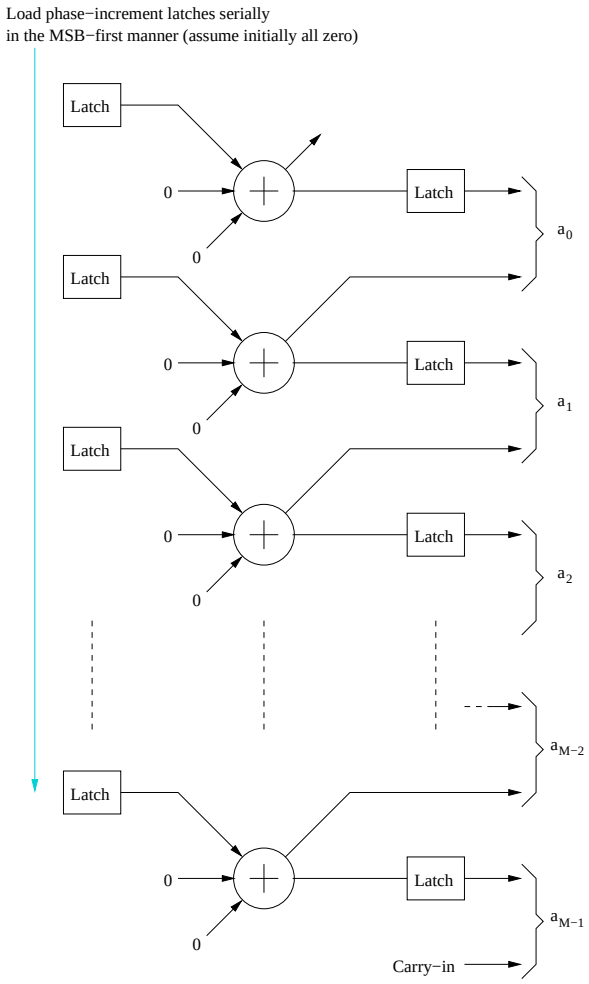


Figure 4.9: Generating time-skew by serially loading phase increment.

Figure 4.10. Note that the phase increment argument is $2P$ instead of P. With this small change, as long as the difference between the initial values across the feedback latches is the intended phase increment P, the accumulator behaves as desired, outputting angles in the increment of P. The validity of the scheme can be proven by induction. At time index i, if the output of the phase accumulator $a[i]$ is $x + P$, then the feedback latch value $f[i]$, by the hypothesis, is x. At time index $i + 1$,

$$\begin{aligned} a[i+1] &= f[i] + 2P \\ &= (x + P) + P \\ &= a[i] + P \\ f[i+1] &= a[i] \\ &= x + P \end{aligned}$$

Therefore, it is shown that, at each iteration, the initial difference of P across the feedback latches is maintained and the output is incremented by P.

The initial difference of P can easily be introduced by the scheme shown in Figure 4.11, which first loads the phase increment latch with P and then $2P$.

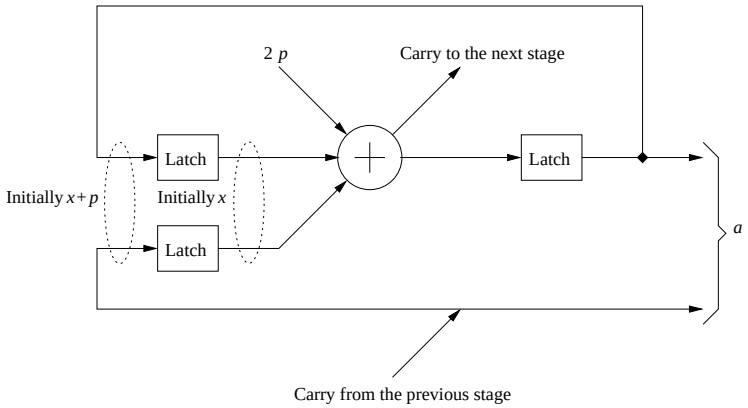


Figure 4.10: One-digit slice of the on-line phase accumulator.

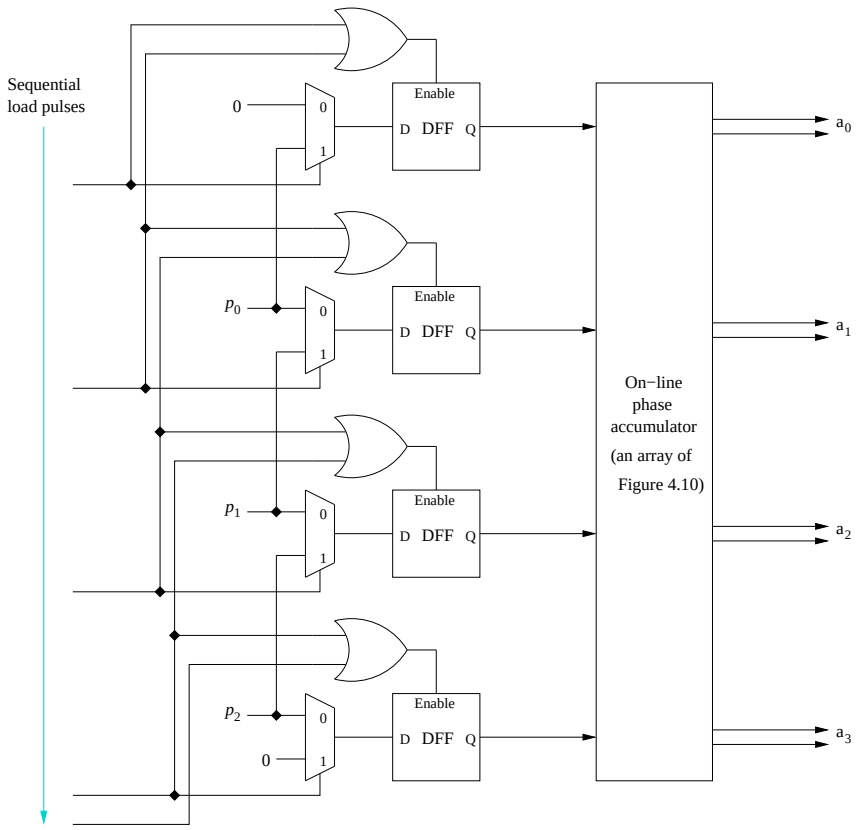


Figure 4.11: Phase increment loading for the on-line phase accumulator.

The on-line phase accumulator scheme shown in Figure 4.10 and Figure 4.11 is valid only when the phase increment latches are initially all zero. In this regard, it can be considered an off-line frequency switching scheme. However, it is often necessary to change frequencies on the fly without going off-line. On-the-fly frequency switching can be achieved by loading the latches with $P^0 + P^1$ and then with $2P^1$, where P^0 is the current phase increment value in the latches and P^1 is the new phase increment to be loaded. Comparing to the off-line scheme, where it loads $2P^1$ and then P^1, this on-line loading scheme requires an extra adder to compute P^0+P^1. This addition consequently results in a delay that increases the system latency. Note that $2P^1$ can be obtained by merely shifting the argument. An alternative on-the-fly frequency switching scheme that does not cost an additional adder is shown in Figure 4.12. This scheme basically follows the framework of the off-line scheme: loading $2P^1$ and then P^1. The difference is that, when it loads $2P^1$, it jams the feedback latches so that they are not updated. One drawback of this scheme is that there is one invalid sample at the switching boundary. Specifically, the output sequence when switching from P^0 to P^1 is as follows.

$$\cdots, x + P^0, x + 2P^0, x + 3P^0, x + 2P^0 + P^1, x + 2P^0 + 2P^1, x + 2P^0 + 3P^1, \cdots$$

Here the sample $x+3P^0$ does not fall within the continuous sequence. In most applications, this should be negligible and represent a reasonable trade-off.

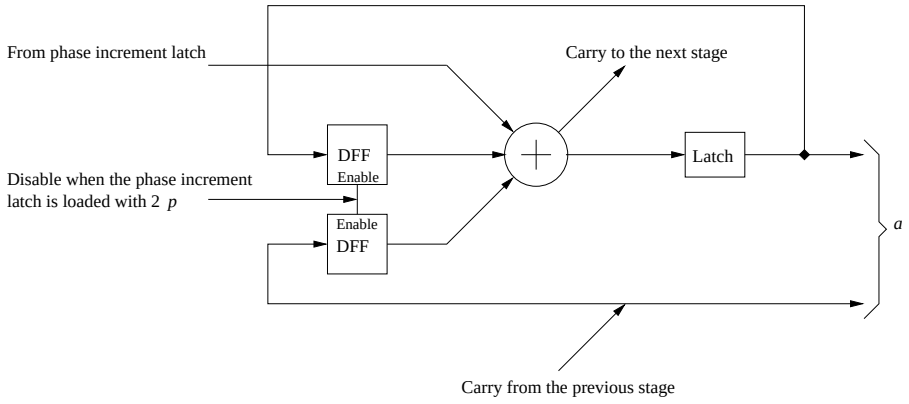


Figure 4.12: One-digit slice of the on-line phase accumulator for on-the-fly frequency switching.

4.4 Interfacing the phase accumulator to the DCORDIC engine

As is, the proposed phase accumulator cycles through the range $[-1, 1)$. Even though this is the intended behavior for a general phase accumulator, it is not compatible with the CORDIC algorithm which has a domain of convergence of, approximately, $[-99.7\ °, 99.7\ °]$ [16]. Therefore, it is necessary to limit the angle argument into the CORDIC engine to be in the range $\left[-\frac{1}{2}, \frac{1}{2}\right]$ and, instead, use the symmetry of the sine/cosine function to properly negate the final output. Note that the negative region $(-1, 0)$ is mapped to the positive region $(0, 1)$ immediately after the first ABS column. Therefore, to ensure convergence, it is sufficient, except for the case where the angle is -1 which

maps back to itself, to map the upper-half positive region, $\left(\frac{1}{2}, 1\right)$, into the lower-half positive region, $\left(0, \frac{1}{2}\right)$, after the first ABS column.

The proposed mapping can be achieved by performing another absolute value calculation on the fractional digits of the first absolute value calculation output. This can easily be accomplished by inserting an extra ABS column after the first one as shown in Figure 4.13. Because the operation is on the fractional digits, the sign digit is simply dropped and not input to the extra vector. More precisely, it is equivalent to multiplying and dividing the carry-save number by 2 before and after, respectively, the extra ABS column. This is possible since the sign digit is always zero after the first absolute value calculation. An extra CSA column is necessary, as shown in the figure, for the addition of two *ulp*'s in case of negation, as explained previously. With this extra iteration, when the datapath uses the rotation direction starting from σ_1, the correct overall mapping, $\left[-1, -\frac{1}{2}\right]$ to $\left[0, \frac{1}{2}\right]$ and $\left[\frac{1}{2}, 1\right)$ to $\left[-\frac{1}{2}, 0\right)$, is achieved. Note that the proposed scheme properly maps -1 to 0.

The information regarding whether the original angle argument was within the range $\left(-\frac{1}{2}, \frac{1}{2}\right)$ needs to be carried over through the datapath so that the final CORDIC output can be properly negated. Figure 4.14 shows the mapped cosine and sine waveforms in the normalized range $[-1, 1)$ and identifies the regions that require output negation. Essentially, this one-bit information that indicates the need for output negation is the same as $\hat{\sigma}_1$, the sign result of the extra ABS column. However, there are two singular cases to this generalization due to the asymmetry of the two's complement num-

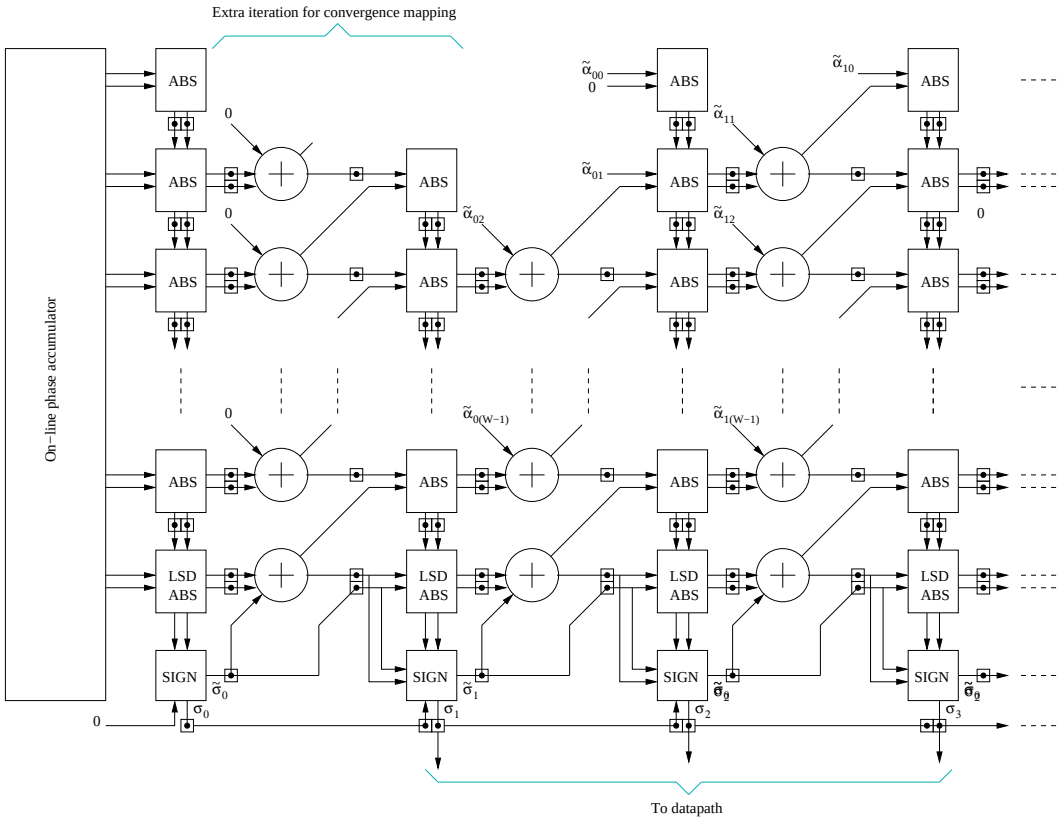


Figure 4.13: Extra iteration for angle mapping.

ber representation. The first case occurs when the phase accumulator output is -1, which can cause a problem in cosine evaluation. Because there is no positive counterpart for -1 in the fractional two's complement number representation, the absolute value calculation on this input results in -1 again, and multiplying the number by 2 results in 0 due to overflow. This is correct, at least in the sense that the sine/cosine evaluation is valid in terms of magnitude. The complication, however, arises from the fact that the absolute

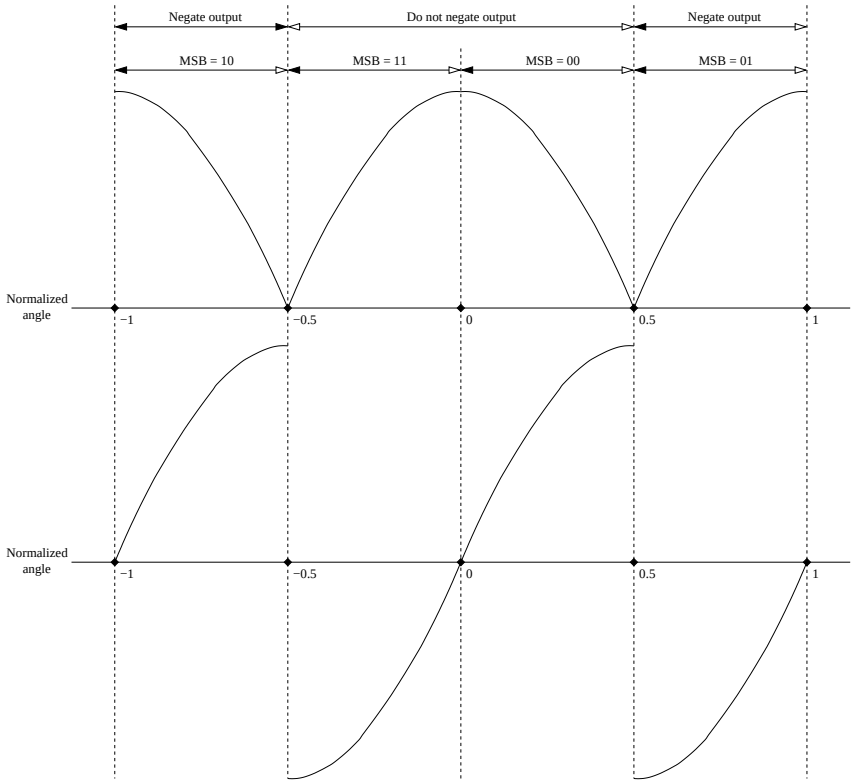


Figure 4.14: Mapped cosine and sine waveforms and regions that require output negation.

value calculation algorithm outputs a positive sign to this input so that the sine/cosine output will not be properly negated. The second case, which can cause a problem in sine evaluation, is similar to the first one, and occurs when the phase accumulator output is -0.5. In this case, the output of the first ABS column is 0.5, so that multiplication by 2 results in -1, again, due to overflow. Consequently, the extra absolute value calculation algorithm will decide on a negative sign, while its output, when divided by 2, finally becomes

0.5. Therefore, it is necessary to decode the sign information slightly differently to accommodate these singular cases. Fortunately, these two conditions can easily be identified by looking only at the inputs of the sign decoder block of the extra ABS column, because, in those cases, the sign input to the sign decoder block must be "undecided" and the digit input must be 2. This can be understood by first observing that the LSD input is non-zero for those cases because the proposed architecture always puts 1 or 2 in the LSD in case of negation. If the LSD is non-zero, then it must be 2 because the number is 0 or -1. With the LSD being 2, the only possibility that the number can be 0 or -1 is that all the fractional digits are 1 so that the carry from the LSD propagates all the way up to the sign digit, making all the fractional digits zero, which in turn implies that the on-line absolute value calculation algorithm can not determine the sign of the number until the LSD.

The extra iteration for angle mapping costs not only extra hardware but also increases the latency by two extra clocks. One way to avoid this extra overhead is to treat the phase accumulator output as being in the $\left[-\frac{1}{2}, \frac{1}{2}\right)$ range, and have a overflow detection circuit to alternatively mark negation every time an overflow occurs. This is certainly more efficient in terms of hardware and latency, but requires that the normalized phase increment be smaller than 1/2 to ensure that the overflow is detected. Even though the sampling theory dictates that the normalized phase increment may be as large as 1, allowing the output frequency of the sinusoids to be up to the half of the system clock frequency [50], it is a general practice to keep the increment

smaller than 2/3 because of the the complexity involved in the analog filter design [36]. In applications that can sacrifice some of the output frequency dynamic range, further restricting the increment under 1/2 can be a good trade-off since it also relaxes the analog filter specification.

Chapter 5

Implementation of the DCORDIC-based DDFS architecture

The DDFS architecture based on the DCORDIC algorithm, as presented in Chapter 4, is implemented in Verilog [49]. More precisely, a generic environment is created that generates fully synthesizable Verilog codes which implement the proposed DDFS architecture, given a set of the required parameters such as the number of iterations, the datapath width, and so on. In the following sections, implementation details of the environment are discussed. Also, some simulation results on the resulting DDFS systems are presented. The chapter concludes with a section that compares the physical attributes of the implementation with other published works.

5.1 Implementation

The overall work flow of the generalized environment that generates the proposed DCORDIC-based DDFS system is illustrated in Figure 5.1. Basically, there are pre-defined Verilog templates for the DDFS components,

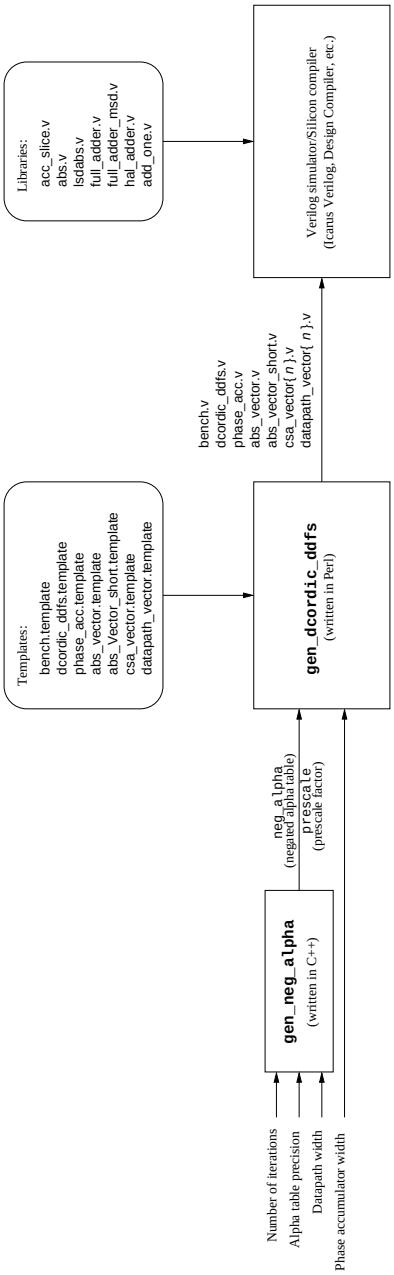


Figure 5.1: Work flow of the generalized DCORDIC-based DDFS generation environment.

and, using those templates, a Perl script [51] generates a complete DDFS system in Verilog for the given parameters. Figure 5.2 shows the symbol of the DDFS system generated by the environment.

There are several system-level choices made in the implementation: for the phase accumulator, the on-the-fly frequency switching scheme as in Figure 4.12 is employed, and, for mapping the phase accumulator output into the domain of convergence, the extra ABS column insertion as in Figure 4.13 is performed. Also, the carry-save representation is used for the sine/cosine datapath. In using the carry-save representation in the datapath, it is necessary to make each stage's output a non-overflowed carry-save number so that the shift-right operation on the carry-save number may be performed with sign-extension, as explained in [43]. This requires a modified adder structure in the MSD as shown in Figure 5.3. The modified structure ensures that each vector of the carry-save output is a valid two's complement number so that scaling may be performed properly with MSD-extension. Also, a guard-digit is added

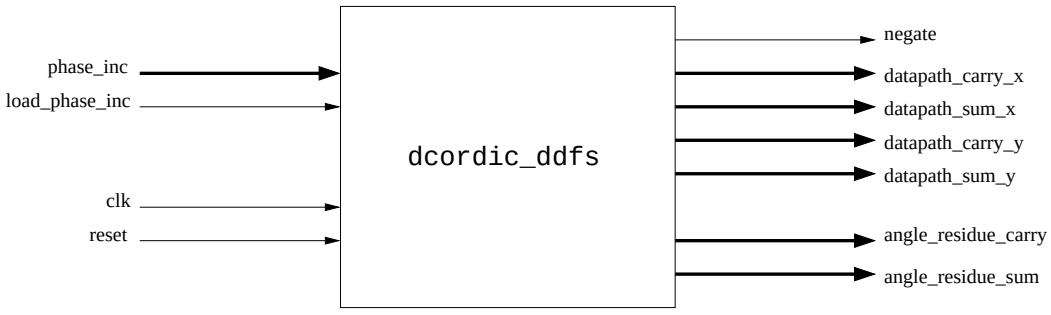


Figure 5.2: Symbolic representation of the DCORDIC-based DDFS system.

to deal with the true overflow that can temporarily occur during calculation. (That is, the case where datapath value becomes equal to or larger than 1, or smaller than -1.)

The datapath uses jamming as suggested in Chapter 2. The error *bound*, however, is doubled because there are effectively two two's complement numbers being quantized by jamming one carry-save number. The actual errors, however, should not be as severe as the bound because of the cancellation of errors due to the unbiasedness of jamming, as explained in Chapter 2.

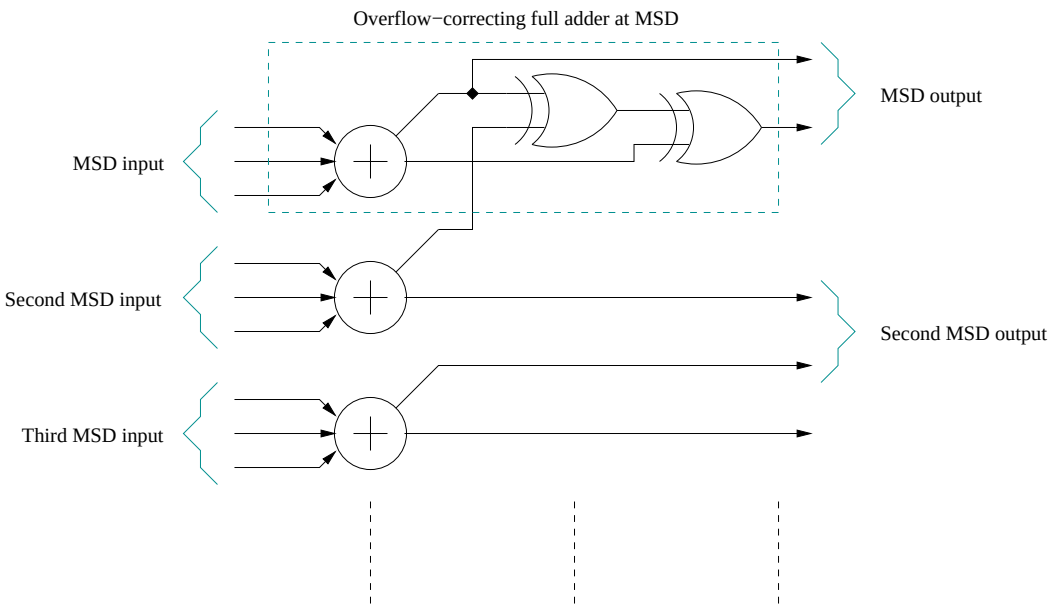


Figure 5.3: Overflow-correcting full adder at the MSD position of the datapath carry-save adders.

The datapath is two-stage pipelined per iteration stage. The input-delaying latches shown in Figure 4.1 are not necessary because the datapath inputs are constants.

One architectural consequence of the DCORDIC-based DDFS system, or, more generally, fully parallel CORDIC implementation, is that it does not require an explicit ROM table for storing the alpha values. That is, each full adder in Figure 4.1, one input of which is tied to a constant alpha bit, can actually be implemented with a half adder or a modified half adder that has two inputs and adds one internally, depending on whether the alpha bit is 0 or 1, respectively. This not only eliminates the ROM table, but also significantly reduces the hardware necessary to implement the CSA's and makes the critical path shorter.

5.2 Alpha table quantization

In a fixed-point realization of the proposed architecture, the alpha table must be quantized to a finite precision. This quantization can be another source of error in CORDIC arithmetic, and, therefore, calls for some analysis. To begin with, it can easily be inferred that the alpha table precision should be at least be $2^{-(n+1)}$, where n is the number of iteration, because the last entry in the table is

$$\alpha_{n-1} = \frac{\arctan 2^{-(n-1)}}{\pi} \approx \frac{2^{-(n-1)}}{\pi}$$

However, this is only a necessary condition because the quantization errors may break the convergence condition [52]

$$\alpha_i - \sum_{j=i+1}^{n-1} \alpha_j \leq \alpha_{n-1}$$

Even when it converges after quantization, the angle approximation error will have a different characteristic because the last entry in the table, the quantized version of α_{n-1}, will be bigger or smaller than the original α_{n-1} due to the quantization error.

5.3 Simulations

A set of register-transfer-level (RTL) simulations has been performed on the generated Verilog codes, using the Icarus Verilog simulator [53] under a Linux environment running on an Intel Pentium III 700MHz platform. The purpose of the simulations is three-fold. First of all, the correctness of the implementation is verified by the simulations. Second, the effect of the alpha table quantization can be observed through the simulations. Third, the effect of jamming on the carry-save datapath is observed and compared with the previous results of jamming the two's complement datapath.

The simulations are performed in two groups. In the first group, the effect of the alpha table quantization is observed by varying the alpha table precision with a fixed datapath width. The datapath width used is 30, which is a value large enough to emphasize only the angle approximation error, and

rounding is used as the alpha table quantization scheme. The simulations are performed for different numbers of iterations (8, 9, 12, 13, 16, and 17) and the alpha table precision is increased until the number of effective bits becomes the same as the value obtained by the previous simulation (Section 2.7) with the same number of iterations and datapath width. The alpha table precision obtained by this procedure can be considered equivalent to the precision achieved by the floating-point representation. Tables 5.1 through 5.3 summarize the results.

Table 5.1: Number of effective bits in fraction for 8-bit angle resolution with varying α table resolution. The α table is rounded. The datapath uses 30-digit (fractional) fixed-point carry-save format and is jammed at each iteration.

	No. of fractional bits in α representation					
	9	10	11	12	13	14
With 8 iterations	6.196	6.417	6.705	6.735	6.953	7.049
With 9 iterations		7.002	7.460	7.511	7.907	8.095

Table 5.2: Number of effective bits in fraction for 12-bit angle resolution with varying α table resolution. The α table is rounded. The datapath uses 30-digit (fractional) fixed-point carry-save format and is jammed at each iteration.

	No. of fractional bits in α representation								
	13	14	15	16	17	18	19	20	21
With 12 iterations	9.605	9.954	10.560	10.847	10.905	10.964	11.000	10.988	11.000
With 13 iterations		10.353	11.223	11.709	11.816	11.928	11.999	11.976	12.001

Table 5.3: Number of effective bits in fraction for 16-bit angle resolution with varying α table resolution. The α table is rounded. The datapath uses 30-digit (fractional) fixed-point carry-save format and is jammed at each iteration.

	No. of fractional bits in α representation								
	17	18	19	21	23	25	27	29	30
With 16 iterations	13.362	13.775	14.376	14.841	14.964	14.995	14.998	14.999	15.000
With 17 iterations		14.122	14.726	15.628	15.929	15.978	15.996	15.998	16.000

In the second group of simulations, the effect of jamming the carry-save datapath is observed by varying the datapath width while the alpha table precision is fixed to the values obtained in the first group of simulations. Tables 5.4 through 5.6 summarize the results.

Table 5.4: Number of effective bits in fraction for 8-bit angle resolution with varying datapath width. The α table resolution is 14 bits. The datapath uses carry-save format and is jammed at each iteration.

	Datapath width								
	8	9	10	11	12	13	14	15	30
With 8 iterations	5.318	6.077	6.661	6.625	6.905	6.992	7.034	7.044	7.049
With 9 iterations	5.300	6.114	6.934	7.325	7.749	7.977	8.026	8.079	8.095

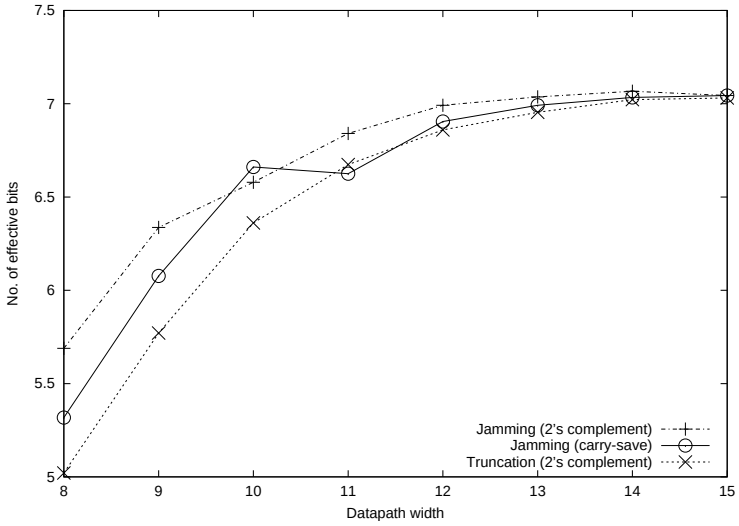
Table 5.5: Number of effective bits in fraction for 12-bit angle resolution with varying datapath width. The α table resolution is 21 bits. The datapath uses carry-save format and is jammed at each iteration.

	Datapath width								
	12	13	14	15	16	17	18	19	30
With 12 iterations	8.775	9.650	10.105	10.582	10.774	10.895	10.972	10.991	11.000
With 13 iterations	8.717	9.791	10.386	11.172	11.478	11.782	11.924	11.979	12.001

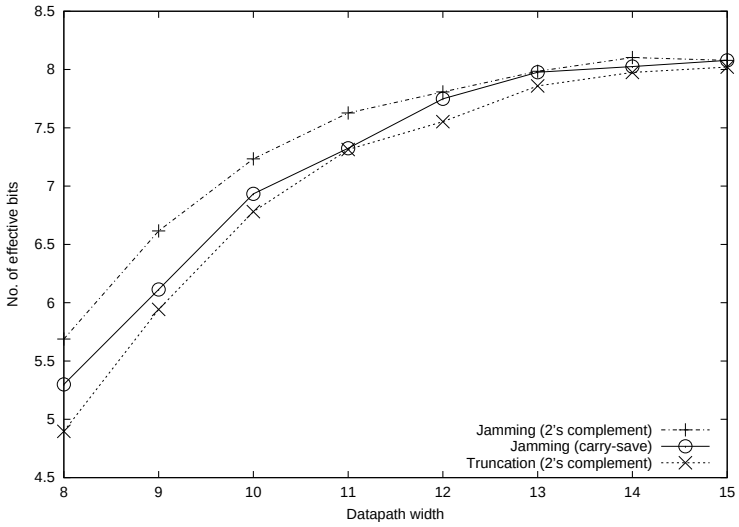
Table 5.6: Number of effective bits in fraction for 16-bit angle resolution with varying datapath width. The α table resolution is 30 bits. The datapath uses carry-save format and is jammed at each iteration.

	Datapath width								
	16	17	18	19	20	21	22	23	30
With 16 iterations	12.365	13.256	13.840	14.381	14.739	14.829	14.931	14.966	15.000
With 17 iterations	12.318	13.304	14.164	14.875	15.468	15.618	15.809	15.921	16.000

The effect of jamming in the carry-save datapath is also compared with the previous simulation results that used the two's complement datapath. Figures 5.4 through 5.6 compare the number of effective bits, and Figures 5.7 through 5.9 compare the error variance. In the figures, it is generally seen that jamming the carry-save datapath still performs better than truncating the two's complement datapath, though it is worse than jamming the two's complement datapath, for both the worst case error (the number of effective bits) and the noise power (the variance). Again, the unbiasedness of jamming errors plays an important role limiting the errors far under the bound.

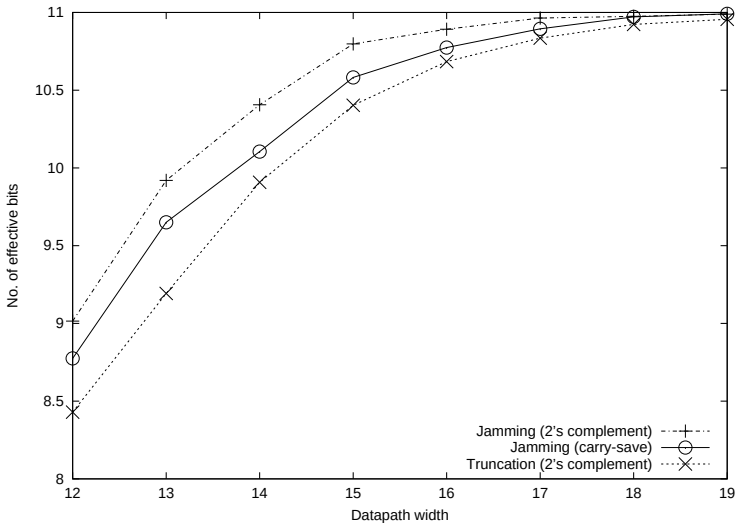


(a)

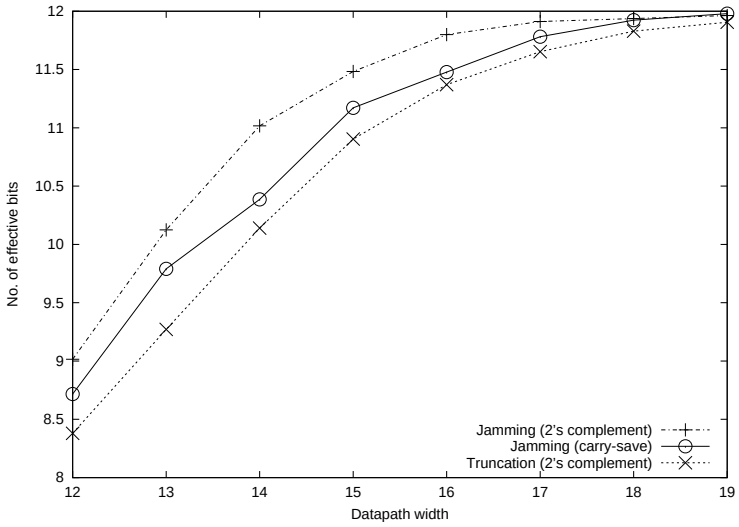


(b)

Figure 5.4: Number of effective bits comparison for 8-bit angle resolution. With 8 iterations (a), and with 9 iterations (b).

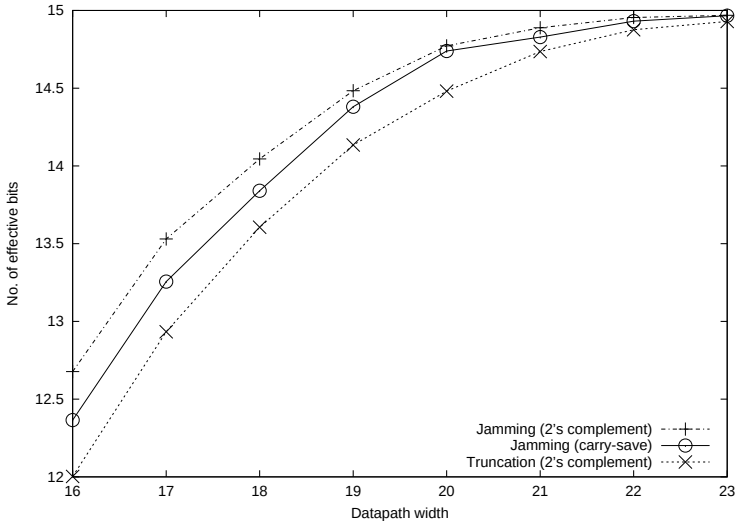


(a)

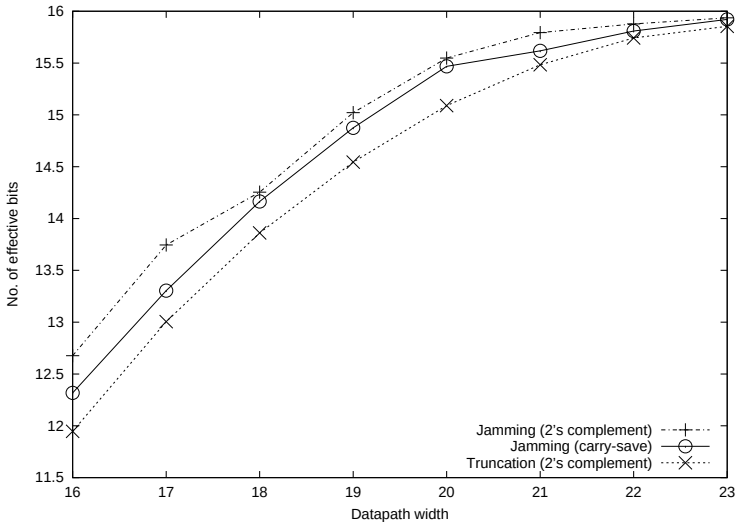


(b)

Figure 5.5: Number of effective bits comparison for 12-bit angle resolution. With 12 iterations (a), and with 13 iterations (b).

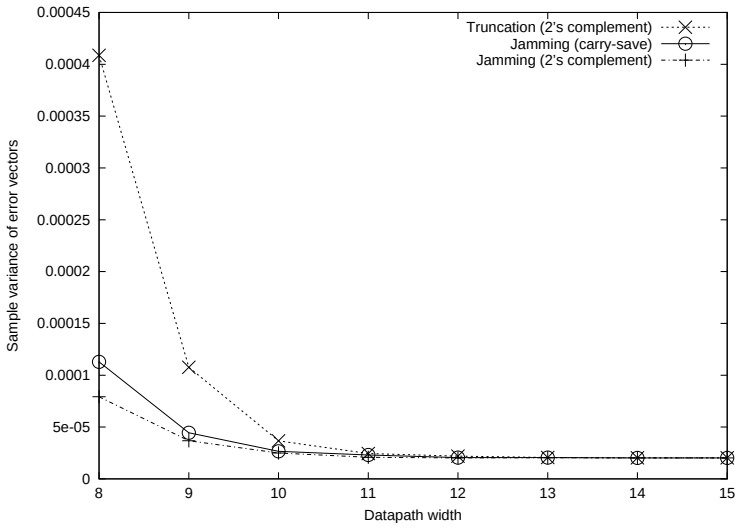


(a)

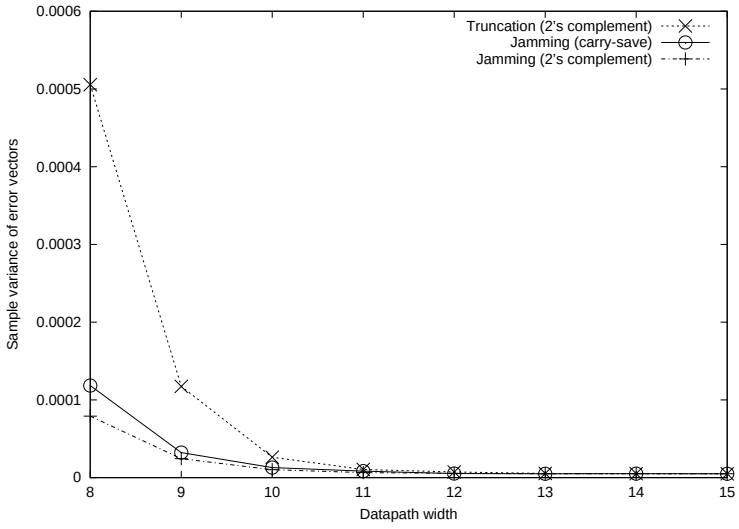


(b)

Figure 5.6: Number of effective bits comparison for 16-bit angle resolution. With 16 iterations (a), and with 17 iterations (b).

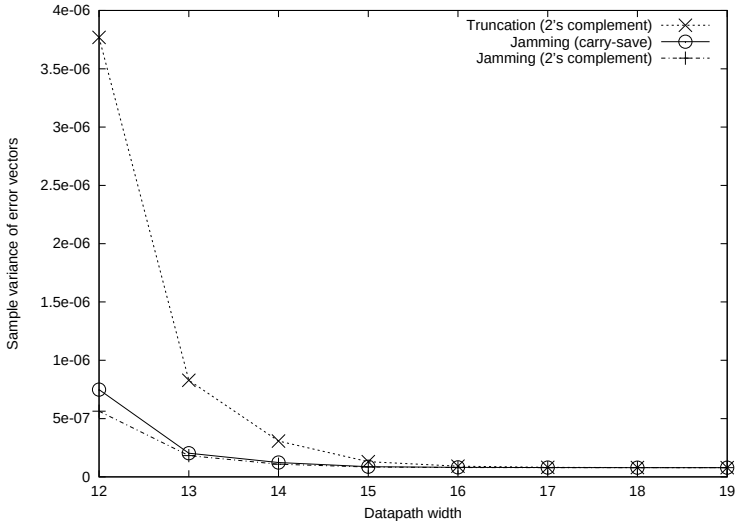


(a)

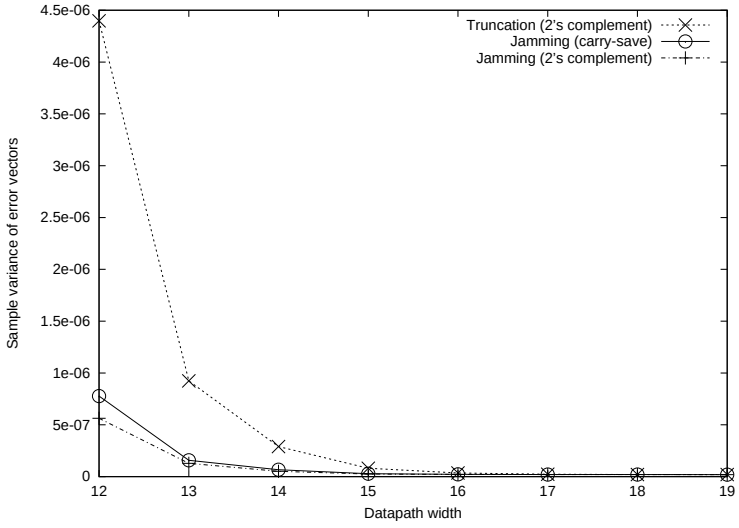


(b)

Figure 5.7: Error variance comparison for 8-bit angle resolution. With 8 iterations (a), and with 9 iterations (b).

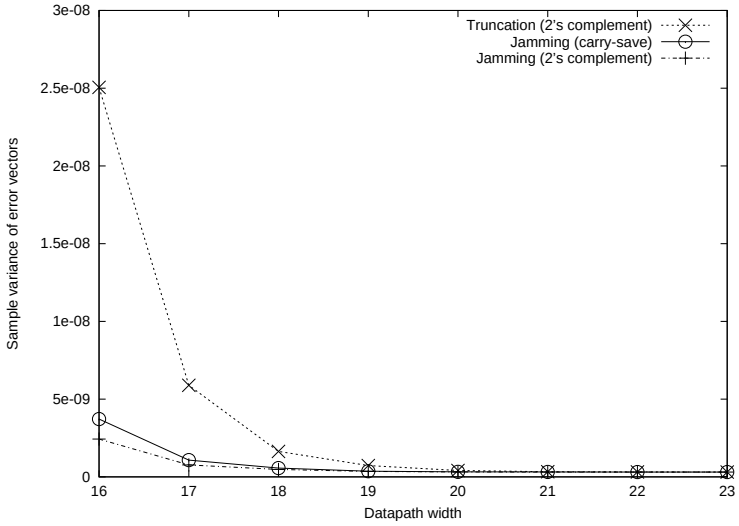


(a)

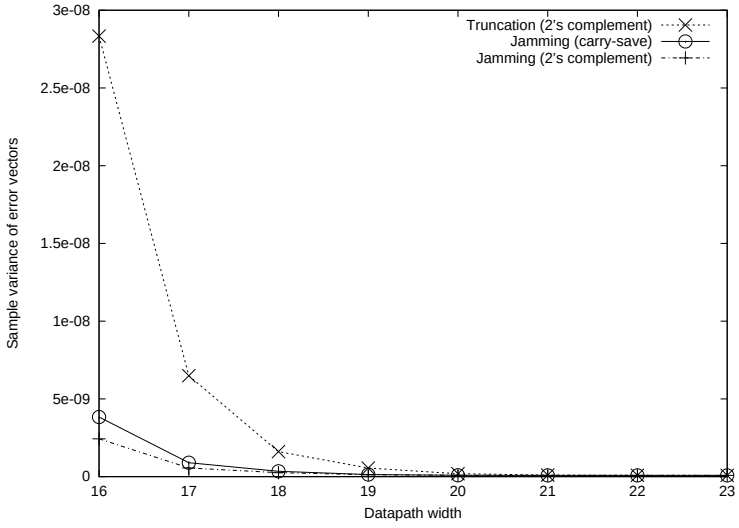


(b)

Figure 5.8: Error variance comparison for 12-bit angle resolution. With 12 iterations (a), and with 13 iterations (b).



(a)



(b)

Figure 5.9: Error variance comparison for 16-bit angle resolution. With 16 iterations (a), and with 17 iterations (b).

5.4 Comparison of physical attributes

To evaluate the physical attributes of the implementation, two specific instances of the resulting system are chosen and synthesized under a standard-cell design flow. In a standard-cell design flow, a few pre-laid-out cells, such as NAND and NOR gates, are used as basic building blocks to implement the entire system [54]. Normally, the mapping of the overall design into the pre-defined building blocks, a process more generally known as *synthesis*, can be performed automatically by a computer-aided design (CAD) tool. Though the resulting hardware is usually not optimal in terms of area and speed, the standard-cell design flow is widely adopted because it enables the rapid development of a complex system with relative ease.

The system parameters of the instances chosen for the experiment are as shown in Table 5.7. The parameters are chosen in such a way that the proposed instances are comparable to the specified comparison targets in terms of the SFDR so that the comparison of area and speed can be meaningful.

Table 5.7: System parameters of the instances chosen for synthesis.

	Phase accumulator width	No. of iterations	α table width (fractional)	Datapath width (fractional)	Comparison target
Instance 1	15	15	16	17	Torosyan [39]
Instance 2	12	10	11	10	Gielis [35]

The synthesis tool used is Design Compiler [55] from Synopsis, Inc. under the Solaris operating system running on a Ultra 10 workstation from

Sun Microsystems, Inc. The standard cells used are the SAGE library from Artisan Components, Inc., a 2.5-volt standard-cell library for the UMC 0.25-μm CMOS process [56]. The result of the synthesis run and the static timing analysis that is based on a conservative wire load model, together with the summary of the target systems for comparison, is presented in Table 5.8.

Table 5.8: Physical attributes of the synthesized instances and the comparison targets.

	Technology	Max. sample rate	Area	Worst-case SFDR
Instance 1	0.25μm CMOS, standard cell	625MHz	1.246mm^2	~90dB
Torosyan [39]	0.25μm CMOS, standard cell	300MHz	0.36mm^2	90.36dB
Instance 2	0.25μm CMOS, standard cell	621MHz	0.57mm^2	~60dB
Gielis [35]	1μm 13GHz bipolar, custom	540MHz	24.96mm^2	~60dB

The work of Torosyan, *et al.* is one of the most recent published designs. Comparison of this work with Instance 1 is meaningful in many aspects because it is based on a similar 0.25μm CMOS standard-cell design methodology. A rough look at the table indicates that Instance 1 achieves an output sample rate that is more than double that of Torosyan, *et al.* at the expense of extra hardware that is about 3.5 times more. Torosyan, *et al.* indicate that their approach can achieve 600MHz output sample rate by duplicating the hardware because, in its original implementation, the hardware actually uses a 600MHz master clock to time-multiplex the multipliers for the sine and cosine computations. In this case, the overall sample rate becomes comparable, but Instance 1 would occupy about 73% more hardware. Before taking

these numbers as they are, note that the work of Torosyan, *et al.* implements a general rotator, instead of a plain DDFS system, which makes Instance 1 a functional subset of the implementation. Despite this discrepancy, however, the comparison should still be indicative of the physical attributes of the proposed system in relation to the published work.

Table 5.8 also shows that the proposed architecture, as realized in Instance 2, outperforms Gielis' design, one of the fastest DDFS systems reported. However, a direct comparison of the numbers between these two, especially for the area, is not very useful because the technology and the design methodology used are very different. However, it is chosen for comparison mainly because it is one of the fastest DDFS systems reported. The fact that it also featured a direct CORDIC realization makes it a good candidate for comparison, too.

According to these comparisons, the major performance advantage of the proposed DDFS system is its high speed of operation. Even in the standard-cell design flow used, the analysis shows that it will result in throughput that is higher than 600MHz, which is still a conservative estimation since the wire load model used for the static timing analysis is a conservative one whereas the highly-regular architecture ensures that the majority of the signal paths will be short local interconnections.

While Torosyan, *et al.*, as mentioned previously, claim that their approach can achieve comparable throughput by duplicating the hardware, which would be with significantly less hardware, the proposed DCORDIC-based DDFS architecture has the advantage of being scalable across a wide range

of design parameters. More specifically, here the scalability means that the system can scale up to a desired precision by repeating only the basic building blocks in a highly regular manner, while maintaining largely the same throughput across different design parameters. The scalability is a direct consequence of the highly regular architecture, and demonstrated by showing that Instance 1 has approximately the same throughput as Instance 2 and does not incur any additional design effort.

The scalability also suggests that the proposed architecture will lend itself to very efficient custom circuit implementation. While design and characterization complexities are among the major hurdles that limit the general use of custom circuit design methodologies, the proposed architecture only requires that a few basic cells be created and thoroughly verified, after which the rest of the design process should resemble the standard-cell based design flow. The use of a custom circuit design methodology has two positive impacts on the resulting system. First of all, the size of the overall hardware will be significantly reduced. While this is generally a valid statement for custom-designed circuits, the proposed architecture will benefit even more due to its heavy use of flip-flop components. In standard-cell based designs, flip-flops are one of the most area-consuming components because there is no provision for forming an efficient vector of flip-flops.

The operation speed will also benefit from the custom circuit implementation. The static timing analysis for both Instance 1 and Instance 2 shows that the critical path of the design is through the MSD of the overflow-

correcting CSA in the sine/cosine datapath. The implementation of the overflow-correcting CSA shown in Figure 5.3 clearly confirms the timing analysis; unlike other digits, the MSD of the overflow-correcting CSA has two extra XOR gates in its longest path, which is approximately equivalent to one full-adder delay. While it is not generally possible to avoid this penalty in the standard-cell based implementation of the circuit, it is shown that the function can be realized by a custom-designed special adder block that costs neither the area nor the speed [43]. This means that, in addition to the proportional speed improvement that is generally expected, the custom circuit implementation can specifically improve the critical path of the proposed system, making the overall gain even more significant.

Chapter 6

Discussions and plans for further research

In this research, CORDIC-based DDFS architectures are studied. The finite-precision effects of the CORDIC algorithm are analyzed in the context of DDFS, and accompanying simulations show that the use of jamming in the CORDIC datapath results in errors that are comparable to those obtained with rounding. Also, it is shown that the CORDIC output can be made exact to the digits by an additional rounding process.

After the basic algorithm analysis, fast CORDIC algorithms are investigated and compared. The variations of DDFS architecture that utilize those algorithms are also investigated. The investigation reveals that most previous approaches focused only on the datapath, which is the CORDIC engine, and the integration of other system components did not get proper attention. For example, when the arc-tangent approximation is used to process later iterations in parallel, the angle argument should be in radians while most phase accumulators are in a normalized format, so that there should be a π multiplier in between. This multiplier can be a significant source of additional area and latency. Also, designing a high-speed phase accumulator in a con-

ventional manner can drastically increase the area and latency, too. In this regard, the DCORDIC algorithm is studied as an alternative. Specifically, the DCORDIC implementation using the carry-save number representation is introduced. Also, a new absolute value calculation algorithm is derived that can deal with possibly overflowed carry-save numbers. The new algorithm is required to interface the angle path with a carry-save phase accumulator. An architectural modification of the DCORDIC implementation is also introduced that enables the addition of two *ulp*'s, which is necessary for the negation of carry-save numbers, to be performed in the on-line arithmetic framework. On-line carry-save phase accumulator architectures that eliminate the time-skewing latch block inherent to the DCORDIC algorithm are another contribution of the research. The proposed on-line carry-save phase accumulators smoothly fit into the DCORDIC's systolic architecture through a novel angle mapping interface, which resembles the other iteration stages, so that a seamless, bottleneck-free datapath is realized throughout the system.

A generic environment is created that generates fully synthesizable Verilog codes that implement the proposed DDFS architecture when given a set of design parameters. Through a set of register-transfer-level simulations on the implemented Verilog codes, the effect of quantizing the alpha table is identified. The effectiveness of jamming in the carry-save datapath is also demonstrated through the simulations. Some physical attributes, such as area and speed, of the implemented system are investigated by synthesizing two specific instances of the implementation in a standard-cell design flow and comparing them with

two published designs. The comparison reveals that the implemented system can achieve high throughput in a scalable manner across a wide range of the design parameters, although it requires more area. It is noted that the highly-regular construction of the implementation is well suited to custom-design methodologies that can lead to further improvements in both area and speed.

There still exist possibilities of improvement to the current implementation of the architecture. It may be possible to further optimize the current sine/cosine datapath implementation. The DCORDIC algorithm requires that each iteration stage of the datapath be two-stage pipelined, the first of which is internal to the iteration and the second is at the output of the iteration stage, because the rotation directions become available in a two-clock time-skewed manner. Currently, each iteration stage, which is essentially two CSA's cascaded, is evenly divided so that each pipeline stage contains one CSA that incurs one full-adder delay. Though it may seem plausible at a glance, this provision for pipelining would not be able to achieve the optimal performance. This is because routing between iteration stages involves a cross-addition between the sine and cosine paths which also involves a hard-wired shifting operation, whereas the coupling of the two CSA's within each iteration stage is very tight. As a consequence, with the current pipelining scheme, pipelining between iteration stages will suffer a longer delay when it is physically implemented. Therefore, it would be wise to put more circuit elements in the pipeline stage that is internal to the iteration. Also note that the sine/cosine datapath is not perfectly scalable because the length of the wire that is required for the cross-

addition and the shifting operation depends on the datapath precision and the number of iterations, while the angle datapath is perfectly scalable, within physical limitations such as the clock skew, due to its systolic architecture. In this regard, it will be optimal to make the first pipeline stage delay the same as the critical path delay in the angle path, which is approximately a delay for two full-adders, leaving as much headroom as possible for routing between iteration stages. This way, the overall system can be made fully scalable until the routing delay between the iteration stages uses up the allowed headroom. A datapath layout scheme proposed in Figure 6.1 will also help maximize the routing headroom, which shows that alternating sine/cosine stages along the datapath leads to an improved critical path when compared to implementing a straight datapath.

The sine/cosine datapath may further require some additional post-processing such as negating, converting to the conventional two's complement format, and dithering. Negating may be necessary because the output has a "negate" flag that signifies that the output should be negated to correctly represent the current value. For the carry-save representation currently implemented for the datapath, an additional CSA stage is required to accomplish this task. Therefore, for applications that require properly negated outputs, it will be more beneficial to use the BSD representation instead, because negating BSD numbers only requires that the sign of each digit be flipped [46]. There also exist efficient limited-carry adder architectures for the BSD representation [47][57]. The output may also have to be converted to the conventional

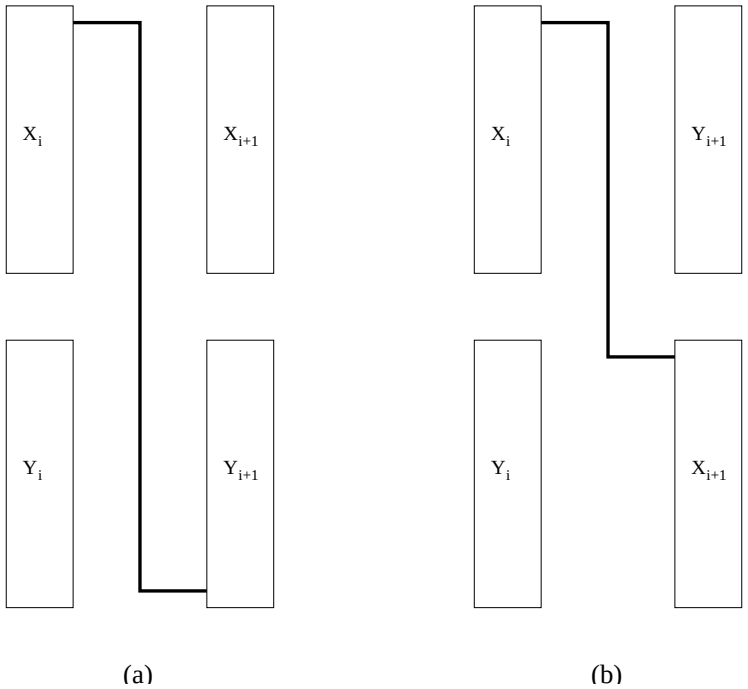


Figure 6.1: Worst-case routing between iteration stages in sine/cosine datapath. With a straight datapath (a), and with an alternating datapath (b).

two's complement number, for example, if it is to be interfaced to a commercially available DAC. The conversion will involve a parallel addition, which is significantly slower than the main DCORDIC DDFS engine. Therefore, it will require some form of pipelined adder, most likely at some group-carry level, with deskewing latches. The conversion process may also perform the truncation of the output to the precision of the DAC preserving the intended number of effective bits, if necessary, as mentioned in Chapter 2.

Dithering is a signal processing technique that deliberately introduces

a small level of noise to the signal to spread out the noise power in the signal over a wide range of frequencies. For DDFS, it has the effect of improving the SFDR of the output by spreading out the energy at the largest spur. Dithering generally involves generating a pseudo random number sequence and adding it to the output, and sometimes requires filtering to further "shape" the noise spectra so the unwanted energy is pushed out of the frequency band of interest [58]. Dithering can also be applied to phase truncation as it is customary to allow more bits for the phase accumulator than the sine/cosine engine can resolve [11].

The throughput of the DCORDIC architecture can readily be adjusted in a systematic way by grouping cells in a square form. This way, a trade-off can be made between throughput and area. While this approach saves area mainly by removing pipeline latches, however, the sequential implementation of the algorithm can save significantly more hardware by removing the actual datapath components, provided that a finer clock than the required throughput is available.

Another natural extension of the research is the hybrid implementation of the algorithm. Basically, this approach exploits arc-tangent approximation for small angles to perform the last $2n/3$ iterations by parallel multiplications, as suggested in [28]. While using the DCORDIC architecture together with the on-line phase accumulator for the first $n/3$ iterations will retain all the benefits of the systolic implementation as discussed, it is also expected that other back-end parts will benefit from this framework as well: especially the π multiplier

is likely to be realized in the on-line arithmetic framework, thereby imposing no additional burden on the latency. Overall, the hybrid implementation is expected to improve the latency of the system by performing the last $2n/3$ iterations in parallel by fast multipliers, at the expense of scalability.

Bibliography

[1] W. A. Chren, Jr., "RNS-Based enhancements for direct digital frequency synthesis," *IEEE Trans. Circuits and Systems-II:Analog and Digital Signal Processing*, vol. 42, pp. 516–524, 1995.

[2] L. E. Larson, *RF and Microwave Circuit Design for Wireless Communications.* Norwood, MA: Artech House, 1996.

[3] H. T. Nicholas, III and H. Samueli, "A 150-MHz direct digital frequency synthesizer in 1.25-μm CMOS with –90-dBc spurious performance," *IEEE J. Solid-State Circuits*, vol. 26, pp. 1959–1969, 1991.

[4] J. Tierney, C. M. Rader, and B. Gold, "A digital frequency synthesizer," *IEEE Trans. Audio Electroacoust.*, vol. AU-19, pp. 48–56, 1971.

[5] R. Jain, H. Samueli, P. T. Yang, C. Chien, G. G. Chen, L. K. Lau, B. Chung, and E. G. Cohen, "Computer-aided design of a BPSK spread-spectrum chip set," *IEEE J. Solid-State Circuits*, vol. 27, pp. 44–58, 1992.

[6] L. K. Tan and H. Samueli, "A 200 MHz quadrature digital synthesizer/mixer in 0.8 μm CMOS," *IEEE J. Solid-State Circuits*, vol. 30, pp. 193–200, 1995.

[7] H. T. Nicholas, III, H. Samueli, and B. Kim, "The optimization of direct digital frequency synthesizer performance in the presence of finite word length effects," *42nd Annu. Frequency Contr. Symp. (USERACOM)*, pp. 357–368, 1988.

[8] J. Vankka, "Methods of mapping from phase to sine amplitude in direct digital synthesis," *1996 IEEE International Frequency Contr. Symp.*, pp. 942–950, 1996.

[9] J. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. EC-8, pp. 330–334, 1959.

[10] D. De Caro, E. Napoli, and A. G. M. Strollo, "Direct digital frequency synthesizers using high-order polynomial approximation," *Int'l Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 134–135, 2002.

[11] M. J. Flanagan and G. A. Zimmerman, "Spur-reduced digital sinusoid synthesis," *IEEE Trans. Communications*, vol. 43, pp. 2254–2262, 1995.

[12] AD9850 Rev. E Datasheet, Analog Devices, 1999.

[13] C. Y. Kang and E. E. Swartzlander, Jr., "An analysis of the CORDIC algorithm for direct digital frequency synthesis," *Proc. 2002 IEEE Int'l Conf. on Application-Specific Systems, Architectures, and Processors (ASAP 2002)*, pp. 111–119, 2002.

[14] Y. H. Hu, "The quantization effects of the CORDIC algorithm," *IEEE Trans. Signal Processing*, vol. 40, pp. 834–844, 1992.

[15] D. J. Kuck, D. S. Parker, Jr., and A. H. Sameh, "Analysis of rounding methods in floating-point arithmetic," *IEEE Trans. Computers*, vol. 26, pp. 643–650, 1977.

[16] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. New York, NY: Oxford University Press, 2000.

[17] R. F. Shaw, "Arithmetic operations in a binary computer," *Rev. Sci. Instrum.*, vol. 21, pp. 687–693, 1950.

[18] M. Park, S. Choi, S. Kim, K. Kim, and J. Lee, "A digital sinusoid synthesis based on the postscaled CORDIC," *Proc. APCC/OECC'99*, pp. 948–951, 1999.

[19] S. B. Lippman and J. Lajoie, *C++ Primer*. Reading, MA: Addison-Wesley, 3rd ed., 1998.

[20] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*. New York, NY: McGraw-Hill, 3rd ed., 1991.

[21] L. Dadda and V. Piuri, "Pipelined adders," *IEEE Trans. Computers*, vol. 45, pp. 348–356, 1996.

[22] Y. H. Hu, "CORDIC-based VLSI architectures for digital signal processing," *IEEE Signal Processing Magazine*, pp. 16–35, July 1992.

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

[24] N. Takagi, T. Asada, and S. Yajima, "Redundant CORDIC methods with a constant scale factor for sine and cosine computation," *IEEE Trans. Computers*, vol. 40, pp. 989–995, 1991.

[25] J. Lee and T. Lang, "Constant-factor redundant CORDIC for angle calculation and rotation," *IEEE Trans. Computers*, vol. 41, pp. 1016–1025, 1992.

[26] D. Timmermann, H. Hahn, and B. J. Hosticka, "Low latency time CORDIC algorithms," *IEEE Trans. Computers*, vol. 41, pp. 1010–1015, 1992.

[27] H. M. Ahmed, "Efficient elementary function generation with multipliers," *Proc. 9th Symp. on Computer Arithmetic*, pp. 52–59, 1989.

[28] S. Wang, V. Piuri, and E. E. Swartzlander, Jr., "Hybrid CORDIC algorithms," *IEEE Trans. Computers*, vol. 46, pp. 1202–1207, 1997.

[29] H. Dawid and H. Meyr, "VLSI implementation of the CORDIC algorithm using redundant arithmetic," *Proc. IEEE Int'l Symp. Circuits and Systems (ISCAS)*, pp. 1089–1092, 1992.

[30] H. Dawid and H. Meyr, "The differential CORDIC algorithm: constant scale factor redundant implementation without correcting iterations," *IEEE Trans. Computers*, vol. 45, pp. 307–318, 1996.

[31] M. D. Ercegovac and T. Lang, "On-line arithmetic: a design methodology and applications in digital signal processing," *VLSI Signal Processing, III*, pp. 252–263, 1988.

[32] H. T. Kung, "Why systolic architectures?," *Computer*, pp. 37–46, Jan. 1982.

[33] E. E. Swartzlander, Jr., ed., *Systolic Signal Processing Systems*. New York, NY: Marcel Dekker, 1987.

[34] E. Grayver and B. Daneshrad, "Direct digital frequency synthesis using a modified CORDIC," *Proc. 1998 IEEE International Symp. on Circuits and Systems*, pp. V–241–V–244, 1998.

[35] G. C. Gielis, R. van de Plassche, and J. van Valburg, "A 540-MHz 10-b polar-to-cartesian converter," *IEEE J. Solid-State Circuits*, vol. 26, pp. 1645–1650, 1991.

[36] A. Madisetti, A. Y. Kwentus, and A. N. Willson, Jr., "A 100-MHz, 16-b, direct digital frequency synthesizer with a 100-dBc spurious-free dynamic range," *IEEE J. Solid-State Circuits*, vol. 34, pp. 1034–1043, 1999.

[37] Y. Ahn, S. Nahm, and W. Sung, "VLSI design of a CORDIC-based derotator," *Proc. IEEE Int'l Symp. on Circuits and Systems*, pp. 449–452, 1998.

[38] S. Nahm and W. Sung, "A fast direction sequence generation method for CORDIC processors," *IEEE Int'l Conf. on Acoustics, Speech, and Signal Processing*, pp. 635–638, 1997.

[39] A. Torosyan, D. Fu, and A. N. Willson, Jr., "A 300-MHz quadrature direct digital synthesizer/mixer in 0.25μm CMOS," *Int'l Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 132–133, 2002.

[40] A. N. Mohieldin, A. A. Emira, and E. Sánchez-Sinencio, "A 100-MHz 8-mW ROM-less quadrature direct digital frequency synthesizer," *IEEE J. Solid-State Circuits*, vol. 37, pp. 1235–1243, 2002.

[41] J. Sklansky, "Conditional-sum addition logic," *IRE Trans. Electron. Comput.*, vol. EC-9, pp. 226–231, 1960.

[42] Private communications with the author at ISSCC 2002.

[43] T. G. Noll, "Carry-save architectures for high-speed digital signal processing," *J. VLSI Signal Processing*, vol. 3, pp. 121–140, 1991.

[44] F. Lu, H. Samueli, J. Yuan, and C. Svensson, "A 700-MHz 24-b pipelined accumulator in 1.2μm CMOS for application as a numerically controlled oscillator," *IEEE J. Solid-State Circuits*, vol. 28, pp. 878–886, 1993.

[45] J. Vankka, M. Waltari, M. Kosunen, and K. A. I. Halonen, "A direct digital synthesizer with an on-chip D/A-converter," *IEEE J. Solid-State Circuits*, vol. 33, pp. 218–227, 1998.

[46] B. Parhami, "Generalized signed-digit number system: a unifying framework for redundant number representations," *IEEE Trans. Computers*, vol. 39, pp. 89–98, 1990.

[47] D. E. Atkins, "Design of the arithmetic units of ILLIAC III: use of redundancy and higher radix methods," *IEEE Trans. Computers*, vol. C-19, pp. 720–733, 1970.

[48] D. E. Atkins, "Introduction to the role of redundancy in computer arithmetic," *Computer*, pp. 74–77, June 1975.

[49] D. E. Thomas and P. R. Moorby, *The Verilog Hardware Description Language*. Norwell, MA: Kluwer Academic Publishers, 3rd ed., 1996.

[50] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

[51] R. L. Schwartz and T. Phoenix, *Learning Perl*. Sebastopol, CA: O'Reilly & Associates, 3rd ed., 2001.

[52] J. S. Walther, "A unified algorithm for elementary functions," *Proc. Spring Joint Computer Conf.*, pp. 379–385, 1971.

[53] Icarus Verilog home page at www.icarus.com/eda/verilog.

[54] N. H. E. Weste and K. Eshraghian, *Principles of CMOS VLSI Design: a Systems Perspective*. Reading, MA: Addison-Wesley, 2nd ed., 1993.

[55] Synopsys Online Documentation (SOLD) Version 2000.11-SP1, Synopsys, 2000.

[56] SAGE Library Databook Release 2.0, Artisan Components, 1999.

[57] C. Y. Chow and J. E. Robertson, "Logical design of a redundant binary adder," *Proc. 4th Symp. on Computer Arithmetic*, pp. 109–115, 1978.

[58] S. P. Lipshitz, J. Vanderkooy, and R. A. Wannamaker, "Minimally audible noise shaping," *J. Audio Eng. Soc.*, vol. 39, pp. 836–852, 1991.

Vita

Chang Yong Kang, the son of Jin Hwan Kang and Young Ji Yi, was born in Seoul, Korea, on January 15, 1969. In 1991, he received the degree of Bachelor of Engineering in Electronic Engineering from Hong-Ik University, Seoul, Korea, with support from Korea Telecommunication Authority Scholarship and Hong-Ik University Scholarship. In 1993, he received the degree of Master of Science in Electrical Engineering from Texas Tech University, Lubbock, Texas, with support from Electrical Engineering Departmental Fellowship. In 1993, he joined Samsung Electronics, Seoul, Korea, where he worked as a senior engineer to develop wireless communication base stations. In 2000, he joined Cirrus Logic, Austin, Texas, as a senior design engineer, where he is currently engaged in developing audio processing integrated circuits. Since 1999, he has been pursuing his Doctorate in Computer Engineering at the University of Texas at Austin.

Permanent address: 3477 Lake Austin Blvd., Apt. D
Austin, Texas 78703

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.