Floating Point Cordic*

Gerben J. Hekstra

Department of Electrical Engineering
Delft University of Technology
2826 CD Delft, The Netherlands

Ed F.A. Deprettere

Department of Electrical Engineering
Delft University of Technology
2826 CD Delft, The Netherlands

Abstract

In this paper, we present a full precision floating-point Cordic algorithm, suitable for the implementation of a word-serial Cordic architecture.

The extension to existing block floating-point Cordic algorithms is in a floating-point representation for the angle. The angle is represented as a combination of exponent, micro-rotation bits and two bits to indicate pre-rotations over $\pi/2$ and π radians. Representing floating-point angles in this fashion maintains the accuracy that is present in the input data, which makes it ideally suited for implementing a floating-point Givens operator.

1 Introduction

The Cordic (COordinate Rotation DIgital Computer) algorithm was presented by Volder[8] as an elegant and cost-effective method to perform rotations on vectors in the 2-D plane. Walther extended the algorithm in [9] to rotations in circular, linear and hyperbolic coordinate systems. Since then, many implementations of the Cordic have been made, both for fixed-point [4, 7] and floating-point [2, 9] with respect to the input.

The main drawback in the computations on floating-point data with the classic Cordic algorithm lies in the inherent fixed-point resolution of the angle. The accuracy becomes unacceptable when calculating angles close to or smaller than the angle resolution.

In this paper we propose a full precision floating-point angle extension to the classic Cordic algorithm to overcome this problem. The floating-point angle representation that we use is such that it fully preserves the accuracy present in the input data. This accuracy would be lost if the angle were to be converted to radians.

The algorithm forms the basis to an IEEE 754 std. 32-bit single precision floating-point Cordic architecture for Givens rotations over true floating-point angles. A similar scheme for floating-point on-line computation of Givens rotations has been presented in [3], but not using Cordic arithmetic. The algorithm presented here has the advantage that it can easily be extended for full precision floating-point computations of transcendental functions.

The results presented in this paper are for circular rotations only, but the same techniques can be applied to derive floating-point algorithms for the hyperbolic and linear coordinate systems.

*This work has been supported by the Dutch National Science Foundation STW, under project STW DEL00.2331

2 Cordic background

The Cordic algorithm was first introduced by Volder in [8], as an efficient method to perform plane rotations. This algorithm was later generalised by Walther in [9] for rotations in circular, linear and hyperbolic systems.

The algorithm knows two modes of operation, namely *vectoring* and *rotation* which are equivalent to the so-called *backward* and *forward* Givens rotations.

In the vectoring mode, the vector $(x, y)_{\text{in}}$ is rotated to $(x, y)_{\text{out}} = ((x_{\text{in}}^2 + y_{\text{in}}^2)^{\frac{1}{2}}, 0)$ and thereby computing the angle of inclination α_{out}.

In the rotation mode, the vector $(x, y)_{\text{in}}$ is rotated over a given angle α_{in} to $(x, y)_{\text{out}}$.

The basic idea behind the Cordic algorithm is that a rotation is decomposed into a sequence of n so-called unnormalized micro-rotations over the base angles α_i, with $i \in \{0, \ldots, n-1\}$ and $0 < \alpha_i \leq \frac{\pi}{2}$. These base angles are chosen in such a way that the micro-rotations are easy to execute (implement) in hardware.

The general recursion for the micro-rotations is given by:

$$\begin{aligned} x_{i+1} &= x_i + (\sigma_i \tan\alpha_i) y_i \\ y_{i+1} &= -(\sigma_i \tan\alpha_i) x_i + y_i \end{aligned} \quad (1)$$

with the in- and output to the recursion:

$$\begin{aligned} x_0 &= x_{\text{in}} \\ y_0 &= y_{\text{in}} \end{aligned} \quad \text{and} \quad \begin{aligned} x_{\text{out}} &= K^{-1} \cdot x_n \\ y_{\text{out}} &= K^{-1} \cdot y_n \end{aligned} \quad (2)$$

The use of *un*normalized micro-rotations in the recursion (1) causes that the vector is lengthened or *scaled* by a factor $(\cos\alpha_i)^{-1}$ with every step. Hence the need for a division by K in equation (2), where K is the accumulative *scaling factor* given by:

$$K = \prod_{i=0}^{n-1} \frac{1}{\cos\alpha_i} \tag{3}$$

The σ_i in the recursion (1) indicate the direction of the micro-rotations. These "sigma-bits" σ_i can be either 1 or -1, signifying clockwise resp. counterclockwise rotations. The relationship between the angle of rotation α and the sequence of sigma-bits $\{\sigma_i\}$ is given by the summation:

$$\alpha = -\sum_{i=0}^{n-1} \sigma_i \alpha_i \tag{4}$$

In order to converge for any angle within a given domain of convergence, the sequence of base angles must form a *basis* and hence satisfy the conditions:

$$\alpha_i \leq \sum_{k>i} \alpha_k + \alpha_{\min} \tag{5}$$

$$\alpha_i \geq \alpha_{i+1} \tag{6}$$

We will refer to the sequence of base angles $\{\alpha_i\}$ as the *angular basis*. The domain of convergence r is given by the sum of the base angles:

$$r = \sum_{i=0}^{n-1} \alpha_i + \alpha_{\min} \tag{7}$$

For any angle α within the domain of convergence, with $|\alpha| \leq r$, the Cordic algorithm will converge with the resolution determined by the smallest angle, $\alpha_{\min} = \alpha_{n-1}$. For proofs and a more detailed discussion of this, the reader is referred to the paper by Walther [9].

For the actual implementation of the micro-rotations, we introduce

$$a_i = \tan\alpha_i \tag{8}$$

and require that the multiplication by a_i is of low complexity when implemented in hardware. We opt for the approach presented by Bu et al. [1], where

$$a_i = 2^{-s_i} + \eta_i \cdot 2^{-s_i'} \qquad \eta_i \in \{-1, 0, 1\} \tag{9}$$

Using this scheme, the micro-rotation in recursion (1) is either a pair on one ($\eta_i = 0$) or two ($\eta_i = -1, 1$) shift- and add operations.

3 Floating-Point Representation of Numbers

Since we are dealing with a floating-point Cordic, let us first take a look at the representation of floating-point numbers. A floating-point number x is represented by a tuple (s_x, e_x, m_x) with:

sign A sign bit s_x, indicating the sign of the number and taking values from $\{-1, 1\}$.

exponent An N_e-bit exponent e_x. The number of bits N_e determine the dynamic range of the representation.

mantissa An N_m-bit, unsigned, mantissa m_x. The number of bits N_m determine the *precision* of the representation. Assuming correct rounding, the mantissa m_x has an inherent accuracy of $\pm\frac{1}{2}$lsb, where lsb $= 2^{-N_m+1}$ is the weight of the least significant bit. In certain cases we will allow the use of a *signed*, 2's complement representation for the mantissa, eliminating the need for a separate sign bit.

The corresponding value of x is given by

$$x = s_x m_x 2^{e_x} \tag{10}$$

For the sake of discussion, we will assume that the input of the floating-point Cordic is *normalized*. This means that the mantissa lies within the range given by:

$$\begin{aligned} 1.00\ldots00_2 &\leq m_x \leq 1.11\ldots11_2 \\ 1 &\leq m_x \leq 2 - \text{lsb} \end{aligned} \tag{11}$$

The IEEE 754 Standard for Binary Floating-Point Arithmetic [6] has, in addition, representations for denormalized numbers, zero, infinity and "Not a Number" (NaN). It has to be mentioned here that all of these too can be elegantly handled by the floating-point Cordic implementation presented in [5]. To deal with this in detail, however, is beyond the scope of this paper.

Before explaining the details of the floating-point Cordic we will first make the following observation. The floating-point Cordic works on 2-D floating-point vectors (x, y). All the possible values for the vector (x, y) lie in the 2-D floating-point domain, as is visualised in figure 1. The boxed areas represent regions in the domain in which the x and y exponents are constant. The points within such a region represent all possible values for the floating-point numbers with *normalized* mantissa. In the example shown in figure 1, we have taken $N_m = 4$, giving $1.000_2 \leq m_x, m_y \leq 1.111_2$. The large box in the top-right corner of the figure represents the region with $e_x = 2, e_y = 2$. The figure only shows only the first quadrant, for positive vectors. It should be extended by mirroring in the x- and y-axes.

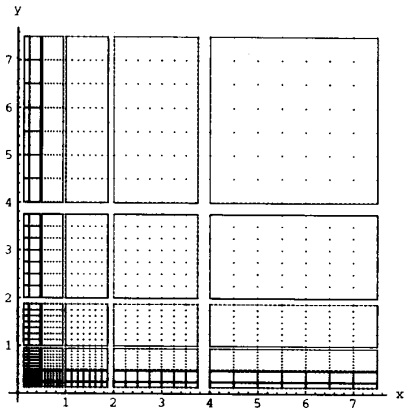

Figure 1: Part of the 2-D floating-point domain spanned by the values of (x, y)

4 The Floating-point Cordic Algorithm

In this section we derive the floating-point Cordic algorithm that can rotate a floating-point vector (x, y) over a floating-point angle α. First, we will define the floating-point Cordic and how the angle α is represented in an internal floating-point format. Next, we will explain how the two basic operations, *rotation* and *vectoring*, work for this floating-point Cordic.

4.1 Definition of the Floating-point Cordic

In the floating-point Cordic, a floating-point vector $(x, y)_{\text{in}}$ is rotated to $(x, y)_{\text{out}}$ over the floating-point angle of rotation α. How this angle α is derived depends on whether the mode of operation is vectoring or rotation.

The floating-point Cordic consists of three distinctive operations. They are:

1. Accuracy preserving pre-rotations over $-\pi/2$ and $-\pi$ radians.
2. The computation of the angle exponent ξ and the selection of the working Cordic $\mathcal{C}_\xi$.
3. The actual micro-rotations using a modified Cordic recursion for block floating-point computations.

The combined results of each of these operations form the representation of the floating-point angle.

4.1.1 Pre-rotations

The input data, $(x, y)_{\text{in}}$, is rotated to the region of convergence using *accuracy-preserving* pre-rotations over $-\pi/2$ and $-\pi$ radians. These pre-rotations take effect in changing the signs and exchanging the exponents and mantissas between x and y and therefore do not affect the precision of in the data. The algorithm is shown below.

if $(e_{y,\text{in}} > e_{x,\text{in}})$ **then**
  /* rotate over $-\pi/2$ */
  $\sigma_{\pi/2} := 1$
  $(s_x, e_x, m_x)_{\pi/2} := (s_y, e_y, m_y)_{\text{in}}$
  $(s_y, e_y, m_y)_{\pi/2} := (-s_x, e_x, m_x)_{\text{in}}$
else
  /* no rotation */
  $\sigma_{\pi/2} := 0$
  $(s_x, e_x, m_x)_{\pi/2} := (s_x, e_x, m_x)_{\text{in}}$
  $(s_y, e_y, m_y)_{\pi/2} := (s_y, e_y, m_y)_{\text{in}}$
end if
if $(s_{x,\pi/2} = -1)$ **then**
  /* rotate over π */
  $\sigma_\pi := 1$
  $(s_x, e_x, m_x)_{\text{pre}} := (-s_x, e_x, m_x)_{\pi/2}$
  $(s_y, e_y, m_y)_{\text{pre}} := (-s_y, e_y, m_y)_{\pi/2}$
else
  /* no rotation */
  $\sigma_\pi := 0$
  $(s_x, e_x, m_x)_{\text{pre}} := (s_x, e_x, m_x)_{\pi/2}$
  $(s_y, e_y, m_y)_{\text{pre}} := (s_y, e_y, m_y)_{\pi/2}$
end if

Algorithm 1: Pre-rotations over $-\pi/2$ and $-\pi$ radians

The first pre-rotation over $-\pi/2$ is performed if $e_{y,\text{in}} > e_{x,\text{in}}$, otherwise no rotation is performed. The result of this pre-rotation is an intermediate vector $(x, y)_{\pi/2}$.

The second pre-rotation, over π radians this time, is performed if the resulting $x_{\pi/2}$ is negative. The combination of these rotations bring the input data to the half-plane $x \geq 0$. The effect of the pre-rotations is visualised in figure 2.

The following properties now hold for the resulting $(x, y)_{\text{pre}}$ data:

$$e_{x,\text{pre}} \geq e_{y,\text{pre}} \tag{12}$$

$$s_{x,\text{pre}} = +1 \tag{13}$$

The pre-rotations serve two purposes. Firstly, they make it possible to accurately represent angles which are close to the x and y-axes, rather than just angles close to the positive x-axis. Secondly, the formulae and the calculations in the datapath can be greatly simplified due to the prior knowledge of condition (12).

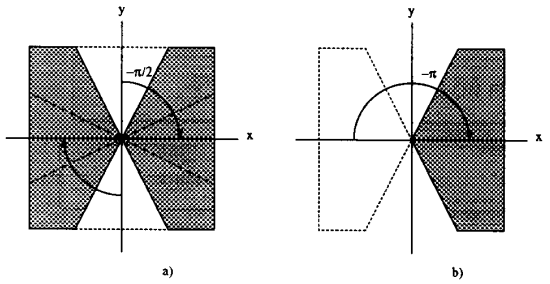

Figure 2: Pre-rotations of the data over a) $-\pi/2$ and b) $-\pi$ radians. The shaded areas indicate the region in in which the vectors reside after rotation.

4.1.2 Angle exponent

The next step in the derivation of the floating-point Cordic algorithm requires the definition of an *angle exponent.*

Looking at the floating-point domain in figure 1, we see that when two regions have the same difference in exponents $e_y - e_x$ they share the same domain of convergence and accuracy in angles. This is due to the equivalence:

$$\tan^{-1}\left(\frac{m_y \cdot 2^{e_y}}{m_x \cdot 2^{e_x}}\right) = \tan^{-1}\left(\frac{m_y}{m_x} \cdot 2^{e_y - e_x}\right) \tag{14}$$

We will use this exponent difference in the vectoring mode to form the angle exponent ξ.

$$\xi = e_{y,\text{pre}} - e_{x,\text{pre}} \tag{15}$$

Combining this with the property (12) placed upon the exponents of the rotated data, we have that:

$$\xi_{\min} \leq \xi \leq 0 \tag{16}$$

where $\xi_{\min} = -(2^{N_e} - 1)$ is the smallest value that the angle exponent can attain in equation (15).

This angle exponent, in turn, selects the proper Cordic $\mathcal{C}_\xi$, from a set of block floating-point Cordics, $\{\mathcal{C}_0, \mathcal{C}_{-1}, \ldots, \mathcal{C}_{\xi_{\min}}\}$. Each of these is specifically designed to operate on angles with given domain of convergence and accuracy as determined by the regions of the floating-point domain. Each $\mathcal{C}_\xi$ has:

1. an angular basis $\{\alpha_i\}_\xi$, consisting of n_ξ angles numbered $\alpha_{0,\xi}, \alpha_{1,\xi}, \ldots, \alpha_{n_\xi - 1,\xi}$,
2. a resulting domain of convergence, r_ξ,
3. a scaling factor K_ξ,
4. a given accuracy as determined by the minimum angle $\alpha_{\min,\xi} = \alpha_{n_\xi - 1,\xi}$.

For the moment, we will assume that these are all given. The actual computation of these values and related implementation details concerning the floating-point Cordic are deferred to section 5. Also, we shall see that the set of Cordics is limited in number: below a certain limit, when $\xi \leq L_{\text{gen}}$ we can revert to one *generic* Cordic, which functions as a template for the set $\{\mathcal{C}_{L_{\text{gen}}}, \ldots, \mathcal{C}_{\xi_{\min}}\}$.

4.1.3 Modified Floating-point Cordic recursion

For a given Cordic $\mathcal{C}_\xi$, selected by the angle exponent ξ, the angular basis is $\{\alpha_i\}_\xi$. What we need now is some scheme, capable of computing the micro-rotations for these angles.

We are placing the constraint on whatever scheme that the micro-rotations are performed in a fixed-precision, block floating-point datapath. This means that exponent remains fixed throughout the calculation and the mantissa is not re-normalized between consecutive additions. This is to prevent the use of expensive floating-point additions for the micro-rotations.

We name this datapath the *core*, and hence define the floating-point vectors $(x, y)_{\text{core}}$ and $(x, y)_{\text{core}'}$ to be the respective in- and output to the core.

In this section we will adhere to the simple micro-rotation model, with $a_{i,\xi}$ satisfying.

$$\tan \alpha_{i,\xi} = a_{i,\xi} = 2^{-S_{i,\xi}} \tag{17}$$

No generality is lost here. The choice is made to keep formulae simple and there is an easy generalisation for the more complex micro-rotations as in formula (9).

If we write out the recursion formula (1) in full, substituting the iteration variables x_i, y_i by their floating-point counterparts; [1]

$$\begin{aligned} x_i &= m_x[i] \cdot 2^{e_x} \\ y_i &= m_y[i] \cdot 2^{e_y} \end{aligned} \tag{18}$$

and re-write the formulae in terms of operations on mantissas, we arrive at the modified Cordic recursion for the mantissas:

$$\begin{aligned} m_x[i+1] &= m_x[i] + \sigma_i 2^{-S_x[i]} m_y[i] \\ m_y[i+1] &= m_y[i] - \sigma_i 2^{-S_y[i]} m_x[i] \end{aligned} \tag{19}$$

and hereby introducing $S_x[i], S_y[i]$ as the *local* shifts for the x, y datapaths.

$$\begin{aligned} S_x[i] &= S_{i,\xi} - (e_y - e_x) \\ S_y[i] &= S_{i,\xi} + (e_y - e_x) \end{aligned} \tag{20}$$

[1] Note well that the exponents are not indexed by the iteration variable i and remain constant through the micro-rotations. This is due to the condition placed on the datapath that computations are done in block floating-point arithmetic. Note also that the core's iteration variables $m_x[i], m_y[i]$ contain an implicit sign.

The initial input to the recursion is given by:

$$\begin{aligned} m_x[0] &= s_{x,\text{core}} \cdot m_{x,\text{core}} \cdot K_\xi^{-1} \\ m_y[0] &= s_{y,\text{core}} \cdot m_{y,\text{core}} \cdot K_\xi^{-1} \end{aligned} \tag{21}$$

Note that the initial values of the mantissas are *pre*scaled by K_ξ^{-1}. In the implementation we will force the scaling factor to be simple to compute, such as a power of two or a difference of two powers of two.

The result of the n_ξ micro-rotations is $m_x[n_\xi], m_y[n_\xi]$. This is converted back into sign and mantissa and rejoined with the exponents to form the floating-point counterpart $(x, y)_{\text{core}'}$.

$$\begin{cases} s_{x,\text{core}'} &= \text{sign}\,(m_x[n_\xi]) \\ e_{x,\text{core}'} &= e_{x,\text{core}} \\ m_{x,\text{core}'} &= |m_x[n_\xi]| \end{cases} \tag{22}$$

and similarly for $y_{\text{core}'}$.

4.1.4 Floating-point Angle representation

The floating-point angle, as produced by the vectoring operation, is a tuple

$$(\sigma_{\pi/2}, \sigma_\pi, \xi, \{\sigma_i\}) \tag{23}$$

of which the elements are:

1. the pre-rotation sigma bits, $\sigma_{\pi/2}, \sigma_\pi$,
2. the angle exponent, ξ,
3. the sigma bit string, $\{\sigma_i\}$ produced by the micro-rotations of the Cordic $\mathcal{C}_\xi$.

The relationship between the floating-point angle of rotation α and its representation as given by formula (23) is given by:

$$\alpha = -\left(\sigma_{\pi/2} \cdot \frac{\pi}{2} + \sigma_\pi \cdot \pi + \sum_{i=0}^{n_\xi - 1} \sigma_i \alpha_{i,\xi}\right) \tag{24}$$

These elements have their analogues in the conventional floating-point number representation as given in section 3. The pre-rotation sigma bits, indicating in which "quadrant" the angle is located are similar in function to the sign bit. The angle exponent has its analogue in the conventional exponent. And finally, the sigma bit string, $\{\sigma_i\}$, has its analogue in the conventional mantissa.[2]

[2]The full representation of a floating-point angle also has flags indicating whether the angle is zero or "Not an Angle"(NaA). An infinite angle cannot occur.

4.2 Vectoring mode of the Floating-point Cordic

4.2.1 In- and Output of Vectoring

The input to the vectoring Cordic operation is the floating-point vector $(x, y)_{\text{in}}$. The output is again a floating-point vector $(x, y)_{\text{out}}$, and a floating point angle α_{out} in the format as given by formula (23).

4.2.2 Execution

The first step is the pre-rotation of the input vector as described in algorithm 1 of the previous subsection. The sigma bits, $\sigma_{\pi/2}, \sigma_\pi$, produced form part of the angle.

Next, the angle exponent ξ is computed from the difference of the exponents of the pre-rotated input vector $(x, y)_{\text{pre}}$:

$$\xi = e_{y,\text{pre}} - e_{x,\text{pre}} \tag{25}$$

This angle exponent is also part of the angle representation and selects the proper Cordic $\mathcal{C}_\xi$ to continue vectoring $(x, y)_{\text{pre}}$.

This vectoring takes place in the core, with the modified Cordic recursion (19). The input to the core is the pre-rotated data, so $(x, y)_{\text{core}} = (x, y)_{\text{pre}}$. The vectoring is steered by the sigma bits produced by the vectoring equation (26).

$$\sigma_i = \begin{cases} +1 & \text{if} \quad m_y[i] \geq 0 \\ -1 & \text{if} \quad m_y[i] < 0 \end{cases} \tag{26}$$

To be consistent to the input/output conventions, the result of the core recursions must be normalized such that $m_{x,\text{out}}$ and $m_{y,\text{out}}$ satisfy the normalization condition (11).

$$\begin{aligned} (s_x, e_x, m_x)_{\text{out}} &= \text{normalize}\,((s_x, e_x, m_x)_{\text{core}'}) \\ (s_y, e_y, m_y)_{\text{out}} &= \text{normalize}\,((s_y, e_y, m_y)_{\text{core}'}) \end{aligned} \tag{27}$$

We will not go into the details of how the normalisation takes place as this is a well known and well documented operation.

The sigma bit string $\{\sigma_i\}$, produced by the vectoring equation (26), makes up the final part of the output angle α_{out}:

$$\alpha_{\text{out}} = (\sigma_{\pi/2}, \sigma_\pi, \xi, \{\sigma_i\}) \tag{28}$$

4.3 Rotation mode of the Floating-point Cordic

4.3.1 In- and Output of Rotation

The input to the rotation Cordic operation is the floating-point vector $(x, y)_{\text{in}}$ and a floating-point angle α_{in} as produced by a previous vectoring operation. The output is a floating-point vector $(x, y)_{\text{out}}$.

4.3.2 Execution

The first step is again the pre-rotation of the input vector $(x, y)_{\text{in}}$ to $(x, y)_{\text{pre}}$. This produces the pre-rotation bits $(\sigma_{\pi/2}, \sigma_{\pi})_{\text{pre}}$. Instead of directly applying the pre-rotation bits $(\sigma_{\pi/2}, \sigma_{\pi})_{\text{in}}$ of the input angle, the two pairs are combined and the result is used in a *post*-rotation later on.

The angle exponent produced by a previous vectoring operation is used to select the working Cordic $\mathcal{C}_\xi$. We will rotate in the same "angle domain" as in which the vectoring took place. This is logical as the sigma bit string $\{\sigma_i\}$ is only meaningful to *this* Cordic.

We do have to take precautions against the possible overflow of the y-mantissa datapath. This could occur when the angle exponent of the angle used for rotation is larger than that of the data. To prevent this we must align the mantissa and change its exponent accordingly. To assure that the overflow is no more than two bits [5] we must have that:

$$e_{y,\text{core}} - e_{x,\text{core}} \geq \xi \tag{29}$$

The alignment algorithm that satisfies the above inequality is given in algorithm 2. Due to the pre-rotations we know that $e_{y,\text{pre}} \leq e_{x,\text{pre}}$, and so only the y mantissa ever needs to be aligned.

```
/* adjust the y mantissa if ξ > e_y − e_x */
ps := max(0, e_y,pre − e_x,pre − ξ)
(s_x, e_x, m_x)_core := (s_x, e_x, m_x)_pre
(s_y, e_y, m_y)_core := (s_y, e_y + ps, m_y · 2^−ps)_pre
```

Algorithm 2: Alignment of the y mantissa to prevent overflow of the fixed-point datapath

Next, the sigma bit string $\{\sigma_i\}_{\text{in}}$ of the angle of rotation α_{in}, is applied in the modified recursion of equation (19).

Similar to vectoring, the result of the core iterations must be normalized in order to comply with the in- and output conditions, viz. equation (27).

Either before or after this normalization step, the data must be brought back to the correct "quadrant" by post-rotations. These rotations are executed in an identical fashion to the pre-rotations. The angle that $(x, y)_{\text{core}'}$ must be rotated over to produce $(x, y)_{\text{out}}$ is given by:

$$(\sigma_{\pi/2,\text{pre}} - \sigma_{\pi/2,\text{in}}) \cdot \frac{\pi}{2} + (\sigma_{\pi,\text{pre}} - \sigma_{\pi,\text{in}}) \cdot \pi \tag{30}$$

5 Implementation

In order to arrive at a practical implementation of the floating-point Cordic algorithm, some analysis must be done on the implementation parameters. These are requirements on the domain of convergence, the accuracy of the angles, and the scaling factor. Due to space limitations, we cannot include all the proofs involved in the derivations of these parameters. We direct the reader to a more detailed treatment of the matter in [5].

Once these parameters are known, the angular bases can be computed for the set of Cordics $\mathcal{C}_\xi$. For this purpose, a heuristic search program, `Bangles` has been written.

Furthermore, we have built a simulation model to validate the correct working of the algorithm.

5.1 Domain of convergence

The largest angle within a region with fixed exponents $(e_x, e_y)_{\text{pre}}$ occurs when:

$$\begin{cases} x &= (1.00\ldots00)_2 \cdot 2^{e_{x,\text{pre}}} \\ y &= (1.11\ldots11)_2 \cdot 2^{e_{y,\text{pre}}} \end{cases} \tag{31}$$

Since this angle determines the domain of convergence, a suitable bound r_ξ, for a Cordic $\mathcal{C}_\xi$ with $\xi = e_{y,\text{pre}} - e_{x,\text{pre}}$, is

$$r_\xi \geq \tan^{-1}(2^{\xi+1}) \tag{32}$$

5.2 Angle accuracy

The resolution in the computation of an angle is determined by the smallest angle $\alpha_{\min}$ in the angular basis. There are a number of criteria to determine a bound for the necessary accuracy in the angle. We will adopt one which takes into account the given accuracy of the input data, making sure that the error induced by the Cordic computation is less than the error which can be accounted to the N_m-bit accuracy of the mantissa representation. A bound for the minimum angle $\alpha_{\min,\xi}$ is derived in [5]. From this bound, and the requirement that a_i is easy to implement, say a power of two, a suitable $\alpha_{\min}$ is given by

$$\begin{cases} \alpha_{\min,\xi} &= \tan^{-1}(2^{\xi - N_m - 1}) \\ a_{\min,\xi} &= 2^{\xi - N_m - 1} \\ S_{\min,\xi} &= -\xi + N_m + 1 \end{cases} \tag{33}$$

5.3 Scaling factor

We would like to force the scaling factor K_ξ to be such that the multiplication with the inverse K_ξ^{-1} in the prescaling of formula (21) is cheap in hardware. In [1] the scaling factor is forced to 2, using complex or additional micro-rotations. However, this approach will not work for the floating-point Cordic. If we force $K_\xi \to 2$ for *every* Cordic $\mathcal{C}_\xi$, then the resulting domain of convergence r_ξ

and the number of base angles n_ξ will be unnecessary large. This is especially so for Cordics with a small angle exponent ξ.

In [5] we derive a suitable value for the scaling factor, that is optimal in the sense of combining an easy multiplication of the inverse scaling factor with a minimal number of micro-rotations. For a Cordic $\mathcal{C}_\xi$, with domain of convergence satisfying condition (32), the ideal scaling factor turns out to be:

$$K_\xi \approx (1 - 2^{2\xi-1})^{-1} \tag{34}$$

The multiplication with the inverse scaling factor then degenerates to a shift and subtract operation, similar to a Cordic micro-rotation.

5.4 The generic Cordic

As we can see from equation (34), the scaling factor quickly approaches 1 as the angle exponent ξ gets smaller. Hence, after a certain limit, we can revert to a *generic* Cordic which has a scaling factor of $K_\xi = 1$ to the required precision and its angular basis defined by the template:

$$\begin{cases} S_i &= -\xi + i \\ a_i &= 2^{\xi-i} \\ \alpha_i &= \tan^{-1}(2^{\xi-i}) \end{cases} , i \in \{0, 1, \ldots, N_m\} \tag{35}$$

It can be proven that this set of angles spans the required domain of convergence r_ξ as set in equation (32) and satisfies the required angle accuracy.

It remains to determine the limiting value of ξ at which the scaling factor of the Cordic, with the angular basis given by equations (35), becomes 1 to enough precision. We call this the *generic Cordic* limit L_{gen}. In [5] we derive this limit to be:

$$L_{\text{gen}} = \left\lfloor \frac{-N_m - 1}{2} \right\rfloor \tag{36}$$

In practice this means that for, say, a 12-bit floating-point Cordic with $L_{\text{gen}} = -7$, the 7 Cordics $\mathcal{C}_0 \ldots \mathcal{C}_{-6}$ must be implemented.

5.5 Computing the Cordic angle bases

With the requirements on the domain of convergence r_ξ, accuracy $\alpha_{\min,\xi}$ and scaling factor K_ξ known, the set of angles that make up the angular basis $\{\alpha_i\}_\xi$ can be computed.

For this purpose, we have written a heuristic search program `Bangles` that finds an optimal set of angles under the given constraints of domain of convergence, accuracy and scaling factor.

The result for a 12-bit floating-point Cordic is presented in table 1. Similar tables have been calculated for a 24-bit mantissa, as required by the IEEE 754 std for single precision floating-point numbers.

Cordic	$\mathcal{C}_0$	$\mathcal{C}_{-1}$	$\mathcal{C}_{-2}$	$\mathcal{C}_{-3}$	$\mathcal{C}_{-4}$	$\mathcal{C}_{-5}$	$\mathcal{C}_{-6}$	$\mathcal{C}_\xi$
ξ	0	-1	-2	-3	-4	-5	-6	≤ -7
k_{scale}	-1	-3	-5	-7	-9	-11	-13	—
K	2	$\frac{8}{7}$	$\frac{32}{31}$	$\frac{128}{127}$	$\frac{512}{511}$	$\frac{2048}{2047}$	$\frac{8192}{8191}$	1

Cordic iteration	$(S_i\ S'_i\ \eta_i)_\xi$							$s_{i,\xi}$
1	(0 4 1)	(1 —)	(3 4 1)	(4 5 1)	(5 7 1)	(6 —)	(6 —)	$-\xi + 0$
2	(1 3 1)	(3 —)	(5 7 1)	(4 —)	(5 —)	(6 —)	(7 —)	$-\xi + 1$
3	(1 —)	(3 —)	(4 6 1)	(5 7 1)	(5 —)	(6 —)	(8 —)	$-\xi + 2$
4	(2 7 −1)	(4 —)	(4 —)	(5 —)	(6 —)	(7 —)	(9 —)	$-\xi + 3$
5	(3 —)	(4 —)	(5 —)	(6 —)	(7 —)	(8 —)	(10 —)	$-\xi + 4$
6	(4 —)	(4 —)	(6 —)	(7 —)	(8 —)	(9 —)	(11 —)	$-\xi + 5$
7	(5 —)	(5 —)	(7 —)	(8 —)	(9 —)	(10 —)	(12 —)	$-\xi + 6$
8	(6 —)	(6 —)	(8 —)	(9 —)	(10 —)	(11 —)	(13 —)	$-\xi + 7$
9	(7 —)	(7 —)	(9 —)	(10 —)	(11 —)	(12 —)	(14 —)	$-\xi + 8$
10	(8 —)	(8 —)	(10 —)	(11 —)	(12 —)	(13 —)	(15 —)	$-\xi + 9$
11	(9 —)	(9 —)	(11 —)	(12 —)	(13 —)	(14 —)	(16 —)	$-\xi + 10$
12	(10 —)	(10 —)	(12 —)	(13 —)	(14 —)	(15 —)	(17 —)	$-\xi + 11$
13	(11 —)	(11 —)	(13 —)	(14 —)	(15 —)	(16 —)	(18 —)	$-\xi + 12$
14	(12 —)	(12 —)	(14 —)	(15 —)	(16 —)	(17 —)	(19 —)	—
15	(13 —)	(13 —)	(15 —)	(16 —)	(17 —)	(18 —)	—	—
16	—	(14 —)	—	—	—	—	—	—

Table 1: The series of floating-point Cordics $\mathcal{C}_\xi$

5.6 Validation of the Algorithm

A simulation model of the floating-point Cordic was built. This model is based on the assumption that calculations are performed in fixed-precision, block-floating-point datapath. Extensive simulations on all modes and input combinations have validated the correctness of the algorithm.

As an example we show the relative error in both the angle- and the x-components after vectoring of the input vector $(m_{\max}, m_{\max} \cdot 2^{-p})$, for the worst-case mantissa $m_{\max} = 1.11\ldots11_2$. We take the $^2\log$ of the relative error to give an indication of the precision in bits.

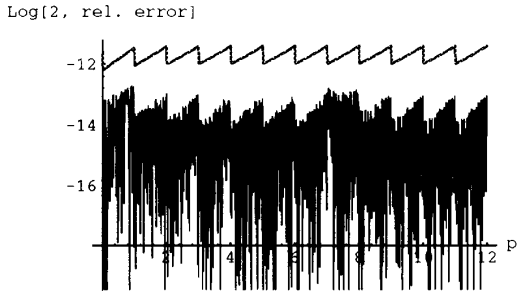

Figure 3: relative error in the approximation of the angle and the bound set by the precision of the input

The bottom curve in figure 3 shows the relative error in the approximation of the angle if it were to be converted back to radians in unlimited precision. The top curve is the maximum relative error which can be accounted to the 12-bit precision of the input mantissas. This sets a bound on the maximum allowable error of the angle approximation.

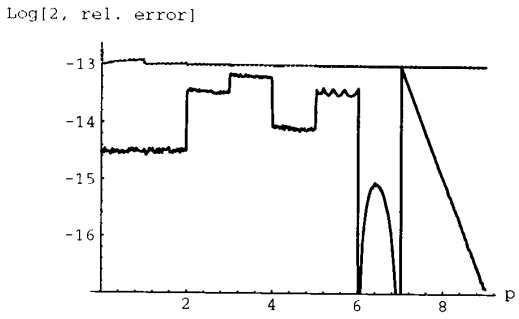

Figure 4: relative error in the x-component and the bound set by the precision of the input

Figure 4 shows the relative error in the x component. Again, the top curve represents the maximum allowable error level set by the precision of the data. The typical sandcastle curve of the relative error is caused by the differing precision for each Cordic in which the scaling factor is approximated. You can clearly see the generic Cordic set in at $-\xi = p \geq 7$.

6 Conclusions

We have presented a floating-point Cordic algorithm that calculates angles to full floating-point precision. We introduce a new floating-point angle representation that preserves the accuracy that is present in the input data. This representation is inherently more accurate than a fixed-precision floating-point representation in radians.

This algorithm serves as the basis to a word-serial Cordic architecture, which is presented in [5]. For the implementation, we have computed the angular bases that make up the series of Cordics $\mathcal{C}_\xi$ for both 12- and 24-bit mantissas.

We have validated the algorithm, architecture and implementation decisions by simulation of a model of the floating-point Cordic.

Currently, we are working on the implementation of the architecture for an IEEE 754 std. single precision Cordic functional unit for use in massively parallel DSP applications.

Our experience has been that for less "regular" binary arithmetic algorithms, such as the floating-point Cordic, the interplay between algorithm and architecture is quite strong. We will soon report on a derived floating-point Cordic algorithm which is suitable for high throughput pipelined computations.

References

[1] Jichun Bu, Ed F. A. Deprettere, and Fons de Lange. On the optimization of a pipelined silicon Cordic algorithm. In I.T. Young et al., editor, *Signal Processing III: Theories and Applications*, pages 1227–1230, 1986.

[2] A.A.J. de Lange, A.J. van der Hoeven, E.F. Deprettere, and J. Bu. An optimal floating-point pipeline CMOS Cordic processor. In *IEEE International Symposium on Circuits and Systems.*, 1988.

[3] Miloš Ercegovac and Tomàs Lang. On-line scheme for computing rotation factors. *Journal of Parallel and distributed computing*, 5:209–227, 1988. Year and volume unknown.

[4] Gene L. Haviland and Al A. Tuszynski. A CORDIC arithmetic processor chip. *IEEE journal of solid-state circuits*, SC-15(1):4–14, February 1980.

[5] Gerben J. Hekstra and Ed F.A. Deprettere. Floating-point Cordic: Algorithm and architecture for a word-serial implementation. Technical report, Technical University of Delft, 1992.

[6] The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, New York 10017. *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std. 754-1985*, July 1985.

[7] R. Künemund, H. Söldner, S. Wohlleben, and T. Noll. Cordic processor with carry-save architecture. In 16th *European Solid-state Circuit Conference*, pages 193–196, September 1990.

[8] Jack. E. Volder. The CORDIC trigonometric computing technique. *IRE Transactions on electronic computers*, pages 330–334, September 1959.

[9] J.S Walther. A unified algorithm for elementary functions. *proceedings of the AFIPS Spring Joint Computer Conference*, pages 379–385, 1971.