

# Chapter 1

## CORDIC

### 1.1 Algoritmo CORDIC

El algoritmo CORDIC propuesto por Volder realizaba rotaciones en coordenadas circulares. Partiendo de esa base es facil extender su funcionamiento para que realice rotaciones en coordenadas hiperbólicas y lineales. Para lograrlo se agrega una variables que modifica las ecuaciones y además se eligen diferentes angulos para el acumulador de angule (variable z del algoritmo). Las ecuaciones del CORDIC completo son:

$$\begin{aligned}x_{n+1} &= x_n - m \cdot d_n \cdot 2^{-s_{m,n}} \\y_{n+1} &= y_n + d_n \cdot 2^{-s_{m,n}} \\z_{n+1} &= z_n - d_n \cdot \alpha_{m,n}\end{aligned}\tag{1.2}$$

Donde  $N$  representa la cantidad de pasos del algoritmo y se cumple que  $n = 0, 1, 2, \dots, N-1$ .  $s_{m,n}$  es una secuencia de números enteros no decreciente llamada *shift sequence* y  $\alpha_{m,n}$  representa los angulos rotados para las diferentes coordenadas.  $d_n$  es una variable de control que maneja los sumadores/restadores.

$$\alpha_{m,n} = \frac{1}{\sqrt{m}} \cdot \tan^{-1}(\sqrt{m} \cdot 2^{-s_{m,n}})\tag{1.3}$$

Eligiendo correctamente  $d_n$  y  $s_{m,n}$  el algoritmo converge (ver tabla 1.1). La tabla 1.2 muestra las diferentes salidas del algoritmo. En la table ?? podemos ver las salidas del algoritmo para valores de entrada particulares, estos valores son de especial interés ya que representan operaciones matematicas dificiles de calcular.

Sistema de coordenadas	Shift sequence	Convergencia	Factor de escala
$m$	$s_{m,n}$	$ A_0 $	$K_m(n \rightarrow \inf)$
1	0,1,2, ... , n	1.74	1.64676
0	1,2,3, ... , n+1	1.0	1.0
-1	1,2,3,4,4, ...	1.13	0.82816

Table 1.1: CORDIC shift sequences

$m$	Modo	Entradas	Salidas
1	rotation	$x_0 = x$ $y_0 = y$ $z_0 = \theta$	$x_N = K_1 \cdot (x \cos \theta - y \sin \theta)$ $y_N = K_1 \cdot (y \cos \theta + x \sin \theta)$ $z_N = 0$
1	vectoring	$x_0 = x$ $y_0 = y$ $z_0 = \theta$	$x_N = K_1 \cdot \text{sign}(x) \cdot \sqrt{x^2 + y^2}$ $y_N = 0$ $z_N = \theta + \tan^{-1}(y/x)$
0	rotation	$x_0 = x$ $y_0 = y$ $z_0 = \theta$	$x_N = x$ $y_N = y + x \cdot z$ $z_N = 0$
0	vectoring	$x_0 = x$ $y_0 = y$ $z_0 = z$	$x_N = x$ $y_N = 0$ $z_N = z + y/x$
-1	rotation	$x_0 = x$ $y_0 = y$ $z_0 = \theta$	$x_N = K_{-1} \cdot (x \cosh \theta + y \sinh \theta)$ $y_N = K_{-1} \cdot (y \cosh \theta + x \sinh \theta)$ $z_N = 0$
-1	vectoring	$x_0 = x$ $y_0 = y$ $z_0 = \theta$	$x_N = K_{-1} \cdot \text{sign}(x) \cdot \sqrt{x^2 - y^2}$ $y_N = 0$ $z_N = \theta + \tanh^{-1}(y/x)$

Table 1.2: Salidas del algoritmo CORDIC.

## Chapter 2

# Algoritmo de Briggs para $\ln(x)$

Si se encuentra una secuencia  $d_k$  tal que la productoria de  $x$  con  $(1 + d_k 2^{-k})$  es cercana a 1 entonces vale que:

$$\begin{aligned} x \prod_{k=1}^n (1 + d_k 2^{-k}) &\approx 1 \\ \ln(x) &\approx - \sum_{k=1}^n \ln(1 + d_k 2^{-k}) \end{aligned} \tag{2.2}$$



## Chapter 3

# BKM

### 3.1 El algoritmo

#### 3.1.1 Orígenes

Consideremos el paso básico del algoritmo CORDIC en modo trigonométrico (con  $m=1$ ). Si definimos el número complejo  $E_n = x_n + j y_n$  con  $j = \sqrt{-1}$ , obtenemos  $E_{n+1} = E_n(1 + j d_n 2^{-n})$ , esta relación es similar al paso básico del algoritmo de Briggs. Esta similitud nos lleva a una generalización de ese algoritmo: podríamos realizar multiplicaciones por términos  $(1 + d_n 2^{-n})$ , donde los  $d_n$ s son números complejos elegidos de tal manera que la multiplicación por  $d_n$  pueda reducirse a unas pocas sumas. Entonces se define el algoritmo BKM de la siguiente manera:

$$\begin{cases} E_{n+1} = E_n \cdot (1 + d_n 2^{-n}) \\ L_{n+1} = L_n - \ln(1 + d_n 2^{-n}) \end{cases} \quad (3.1)$$

con  $d_n = d_n^r + j d_n^i$  y  $d_n^r, d_n^i \in \{0, \pm 1\}$  y  $\ln z = t$  es el número complejo  $t$  tal que  $\exp t = z$  y cuya parte imaginaria está entre  $-\pi$  y  $\pi$ .

#### 3.1.2 Número de constantes

Este algoritmo requiere que se almacenen las constantes referidas a  $\ln(1 + d_n 2^{-n})$ .

$$\begin{cases} \Re\{\ln(1 + d_n 2^{-n})\} = \frac{1}{2} \ln[1 + d_n^r 2^{-n+1} + (d_n^r{}^2 + d_n^i{}^2) 2^{-2n}] \\ \Im\{\ln(1 + d_n 2^{-n})\} = d_n^i \arctan\left(\frac{2^{-n}}{1 + d_n^r 2^{-n}}\right) \end{cases} \quad (3.2)$$

En la ecuación ?? podemos observar que la parte real de las constantes posee simetría con respecto a  $d_n^i$  y que la parte imaginaria sólo depende del valor de  $d_n^r$  (su signo puede invertirse en el hardware). Siguiendo estas deducciones llegamos a que se necesitan  $8 \neq 9$  términos por cada valor de  $n$ . Por lo tanto, para obtener aproximadamente  $N$  bits de precisión necesitamos almacenar  $8 \neq 9N$  constantes. Este resultado puede mejorarse observando que si  $n > N/2$  entonces  $\ln(\cdot)$  y  $\arctan(\cdot)$  pueden reemplazarse por  $d_n^r 2^{-n+1}$  y  $2^{-n}$  con una exactitud de  $2^{-N}$  bits. Entonces sólo necesitamos almacenar  $4N \neq 9/2N$  constantes.

## 3.2 E-mode

Encontrar una secuencia de  $d_n$  tal que  $L_n \rightarrow 0$  entonces  $E_n \rightarrow E_1 e^{L_1}$ .

$$\begin{cases} E_n \rightarrow E_1 e^{L_1} \\ L_n \rightarrow 0 \end{cases} \quad (3.3)$$

### 3.2.1 Canal de datos

$$\begin{cases} E_{n+1}^r = E_n^r + (d_n^r E_n^r - d_n^i E_n^i) 2^{-n} \\ E_{n+1}^i = E_n^i + (d_n^r E_n^i + d_n^i E_n^r) 2^{-n} \end{cases} \quad (3.4)$$

### 3.2.2 Canal de control

$$\begin{cases} l_n = 2^n L_n \text{ con 4 dígitos de precisión decimal} \\ l_{n+1} = 2l_n - 2^{n+1} \ln(1 + d_n 2^{-n}) \\ l_{n+1}^r = 2l_n^r - 2^n \ln[1 + d_n^r 2^{-n+1} + (d_n^{r^2} + d_n^{i^2}) 2^{-2n}] \\ l_{n+1}^i = 2l_n^i - 2^{n+1} d_n^i \arctan\left(\frac{2^{-n}}{1 + d_n^r 2^{-n}}\right) \end{cases} \quad (3.5)$$

$$2^n \ln(1 + d_n 2^{-n}) \rightarrow 1 \quad \text{as } n \rightarrow +\infty \quad (3.6)$$

### 3.2.3 Rango de convergencia

$$\begin{aligned} L_1 &\in R_1 \\ R_n &= [-s_n^r; r_n^r] + j[-r_n^i; r_n^i] \\ r_n^r &= \sum_{k=n}^{\infty} \ln(1 + 2^{-k}) \\ s_n^r &= -\frac{1}{2} \sum_{k=n}^{\infty} \ln(1 - 2^{-k+1} + 2^{-2k+1}) \\ r_n^i &= \sum_{k=n}^{\infty} \arctan\left(\frac{2^{-k}}{1 + 2^{-k}}\right) \\ s_1^r &= 0.82980237... \\ r_1^r &= 0.86887665... \\ r_1^i &= 0.749780302... \end{aligned} \quad (3.7)$$

### 3.2.4 Reducción del rango de entrada

Los algoritmos para calcular funciones elementales generalmente convergen en un dominio acotado. Para calcular  $f(x)$  dado un valor arbitrario  $x$ , usualmente tenemos que encontrar un valor  $x'$

perteneciente al dominio de convergencia del algoritmo que calcula  $f$ , luego  $f(x)$  puede deducirse de  $f(x')$ . Este proceso se denomina *reducción de rango* de entrada (del inglés *range reduction*).

Si  $I$  es un intervalo que contienen al cero y su longitud es mayor que  $\rho$ , podemos computar para cualquier número real  $x$  un número entero  $k$  tal que  $x - k\rho \in I$ . Esto puede lograrse aplicando unos pocos pasos de un algoritmo similar a una división SRT. Supongamos que queremos calcular  $\exp(x + jy)$ . El algoritmo BKM nos permite evaluar la función exponencial compleja dentro del rectángulo  $R_1 = [-s_1^r; r_1^r] + j[-r_1^i; r_1^i]$ . La reducción del rango de entrada puede lograrse de la siguiente manera cuando la entrada es  $z_{in} = x_{in} + jy_{in}$ :

1. Calcular  $k_y$  tal que  $y_{in} - k_y \frac{\pi}{4}$  pertenezca a  $[-r_1^i; r_1^i]$ . Definir  $y' = y_{in} - k_y \frac{\pi}{4}$ .
2. Tenemos:  $e^{z_{in}} = e^{x_{in} + jy_{in}} = e^{j(k_y \bmod 8) \frac{\pi}{4}} e^{x_{in} + jy'}$ . La multiplicación por  $e^{j(k_y \bmod 8) \frac{\pi}{4}}$  puede parecer difícil de reducir a una pequeña cantidad de sumas y shifts. Afortunadamente, este problema se puede solucionar fácilmente. Como ejemplo consideremos el caso  $k_y \bmod 8 = 1$ . Allí el término  $e^{j\frac{\pi}{4}}$  es igual a  $\frac{\sqrt{2}}{2}(1 + j)$ . Una multiplicación por este término puede evitarse si se le suma  $-\frac{1}{2}\ln(2) = \ln(\frac{\sqrt{2}}{2})$  a  $x_{in}$ , que nos da un valor  $x'$  y luego obtenemos:  $e^{x_{in} + jy_{in}} = (1 + j)e^{x' + jy'}$ . Una multiplicación por  $(1 + j)$  puede reducirse muy fácilmente a dos sumas. Un truco similar puede usarse para otros posibles valores de  $k_y \bmod 8$ . Entonces definimos  $K_p$  y  $\gamma_p$  de la siguiente manera:

$$\left\{ \begin{array}{lll} K_0 = 1 & \text{and} & \gamma_0 = 0 \\ K_1 = 1 + j & \text{and} & \gamma_1 = -\frac{1}{2}\ln(2) \\ K_2 = j & \text{and} & \gamma_2 = 0 \\ K_3 = -1 + j & \text{and} & \gamma_3 = -\frac{1}{2}\ln(2) \\ K_4 = -1 & \text{and} & \gamma_4 = 0 \\ K_5 = -1 - j & \text{and} & \gamma_5 = -\frac{1}{2}\ln(2) \\ K_6 = -j & \text{and} & \gamma_6 = 0 \\ K_7 = 1 - j & \text{and} & \gamma_7 = -\frac{1}{2}\ln(2) \end{array} \right. \quad (3.8)$$

Con  $p = k_y \bmod 8$  y  $x' = x_{in} + \gamma_p$  tenemos  $e^{z_{in}} = K_p e^{x' + jy'}$ .

3. Calcular  $k_x$  tal que  $x'' = x' - 2k_x \ln(2)$  pertenezca a  $[-s_1^r; r_1^r]$ . Con ese valor definimos  $y'' = y'$  y  $z_{BKM} = x'' + jy''$ .

El resultado final es  $e^{z_{in}} = 2^{2k_x} K_p e^{z_{BKM}}$

### 3.3 L-mode

Encontrar una secuencia de  $d_n$  tal que  $E_n \rightarrow 1$  entonces  $L_n \rightarrow L_1 + \ln(E_1)$ .

$$\left\{ \begin{array}{l} E_n \rightarrow 1 \\ L_n \rightarrow L_1 + \ln(E_1) \end{array} \right. \quad (3.9)$$

### 3.3.1 Canal de datos

$$\begin{cases} L_{n+1}^r = L_n^r - \frac{1}{2} \ln[1 + d_n^r 2^{-n+1} - (d_n^r)^2 + d_n^i)^2] 2^{-2n}] \\ L_{n+1}^i = L_n^i - d_n^i \arctan\left(\frac{2^{-n}}{1 + d_n^r 2^{-n}}\right) \end{cases} \quad (3.10)$$

### 3.3.2 Canal de control

$$\begin{cases} e_n = 2^n(E_n - 1) \text{ con 4 dígitos de precisión decimal} \\ e_{n+1} = 2(e_n + d_n) + d_n e_n 2^{-n+1} \\ e_{n+1}^r = 2(e_n^r + d_n^r) + (d_n^r e_n^r - d_n^i e_n^i) 2^{-n+1} \\ e_{n+1}^i = 2(e_n^i + d_n^i) + (d_n^i e_n^r + d_n^r e_n^i) 2^{-n+1} \end{cases} \quad (3.11)$$

### 3.3.3 Rango de convergencia

Seguendo el algoritmo original de 94 el rango de convergencia es:

$$E1 \in T = \{x + j y : 0.5 \leq x \leq 1.3, -\frac{1}{2}x \leq y \leq \frac{1}{2}x\} \quad (3.12)$$

Seguendo el algoritmo del 99 el rango de convergencia es:

$$E1 \in T = \{x + j y : 0.64 \leq x \leq 1.4, -\frac{2}{5}x \leq y \leq \frac{2}{5}x\} \quad (3.13)$$

### 3.3.4 Reducción del rango de entrada

Entro con una variable  $z_{in} = x_{in} + j y_{in}$

$$\ln(z) = \ln(+x + jy) = \frac{1}{2} \ln(x^2 + y^2) + j \arctan\left(\frac{y}{x}\right) \quad (3.14)$$

1. Pasar de los cuadrantes II, III y IV al cuadrante I definiendo  $z = |x_{in}| + j |y_{in}|$ . En caso que se pase de los cuadrantes II o IV entonces se tiene que conjugar el resultado final.

$$\begin{aligned} \ln(-z) &= \ln(-x - jy) = \ln(x + jy) = \ln(z) \\ \ln(z^*) &= \ln(+x - jy) = \ln(x + jy)^* = \ln(z)^* \\ \ln(-z^*) &= \ln(-x + jy) = \ln(x + jy)^* = \ln(z)^* \end{aligned} \quad (3.15)$$

2. Si me encuentro en la parte superior del semicuartante I ( $y > x$ ) entonces definir  $z' = z * e^{-j\pi/4} = z * (1 - j)$  y luego  $\ln(z') = \ln(z) - \frac{1}{2} \ln(2) - j \frac{\pi}{4}$ . De caso contrario  $z' = z$

$$\begin{cases} z' = z * (1 - j) & \text{si } y > x \text{ luego } \ln(z') = \ln(z) - \frac{1}{2} \ln(2) - j \frac{\pi}{4} \\ z' = z & \text{si } y \leq x \text{ luego } \ln(z') = \ln(z) \end{cases} \quad (3.16)$$

De esta manera  $z'$  cumple que  $y \leq x$ , osea que esta siempre en la parte inferior del semicuartante I.



3. La primera linea convierte el rango  $[45; 26.565]$  en  $[0; -18.435]$ . La segunda linea convierte el rango  $[14.035; 26.565]$  en  $[-12.529; 0]$ . La tercera linea mantiene el rango  $[0; 14.035]$ . Por lo tanto  $z''$  pertenece al rango de  $[-12.529; 14.036]$  en grados.

$$\begin{cases} z'' = z' * (1 - j) & \text{si } \frac{x'}{2} \leq y' \leq x' & \text{luego } \frac{-x''}{3} \leq y'' \leq 0 & y \quad \ln(z'') = \ln(z') - \frac{1}{2} \ln(2) - j \frac{\pi}{4} \\ z'' = z' * (1 - \frac{j}{2}) & \text{si } \frac{x'}{4} \leq y' \leq \frac{x'}{2} & \text{luego } \frac{-2x''}{9} \leq y'' \leq 0 & y \quad \ln(z'') = \ln(z') - \frac{1}{2} \ln(2) - j \frac{\pi}{8} \\ z'' = z' & \text{si } 0 \leq y' \leq \frac{x'}{4} & \text{luego} & \ln(z'') = \ln(z') \end{cases} \quad (3.17)$$

Finalmente  $\frac{-x''}{3} \leq y'' \leq \frac{x''}{4}$ . De esta manera queda dentro del rango especificado en el paper de 99.

4. Elegir un numero entero  $k$  que verifique que  $0.6 \leq x_{BKM} \leq 1.4$  y luego definir  $z_{BKM} = 2^k z''$ . Luego  $\ln(z_{BKM}) = \ln(z'') + k \ln(2)$ .

El resultado final es de la forma  $\ln(z_{in}) = \ln(z_{BKM}) + a \ln(2) + jb\pi$ .



## Chapter 4

# Architecture

### 4.1 BKM Float

### 4.2 BKM Fixed

#### 4.2.1 Función

Este bloque toma las entradas  $z_1 = x_1 + jy_1$  y  $z_2 = x_2 + jy_2$  y calcula  $z_3 = f_{\text{op}}(z_1, z_2)$  siendo  $f_{\text{op}}(z_1, z_2)$  una operación definida por la variable op. A continuación se detalla la lista de operaciones posibles y sus códigos de operación.

Código de operación	Operación	Descripción	Tiempo de ejecución
000000	$\exp(z_1)$		N
000000	$\ln(z_1)$		N
000000	$\cosh(z_1)$	$\frac{e^{z_1} + e^{-z_1}}{2}$	2N
000000	$\sinh(z_1)$	$\frac{e^{z_1} - e^{-z_1}}{2}$	2N
000000	$\tanh(z_1)$	$\frac{e^{2z_1} - 1}{e^{2z_1} + 1}$	2N
000000	$\text{arctanh}(z_1)$	doble	2N
000000	$\cos(z_1)$	doble	2N
000000	$\sin(z_1)$	doble	2N
000000	$\arctan(z_1)$	doble	2N
000000	$\tan(z_1)$	doble	2N

Table 4.1: Operaciones del bkm.fixed.

#### 4.2.2 BKM Pre

Este bloque se encarga de

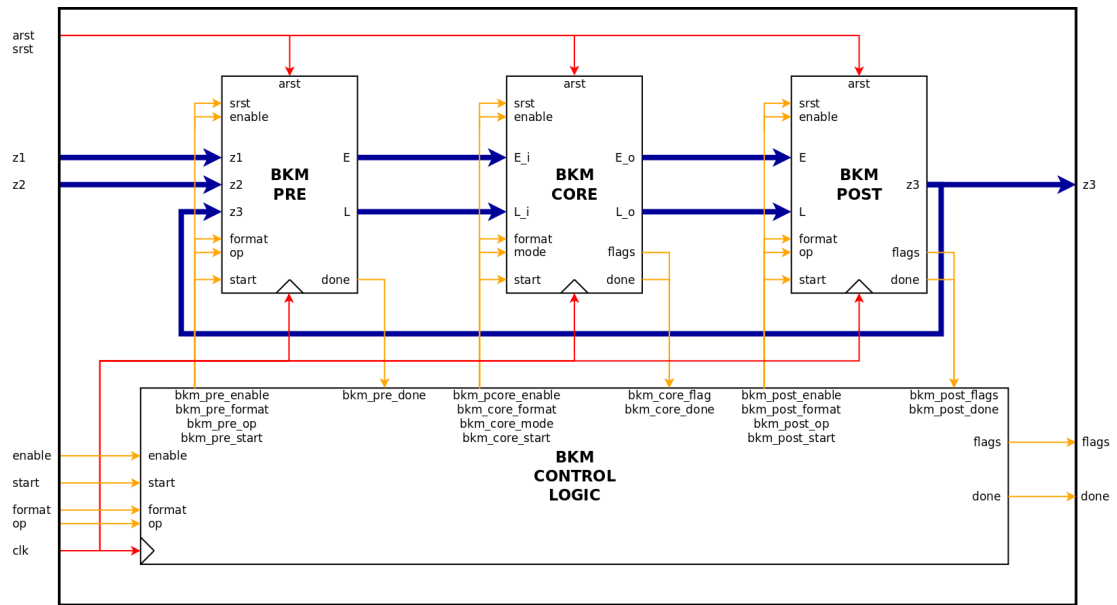


Figure 4.1: Architecture del bloque BKM fixed.

#### 4.2.3 BKM Post

#### 4.2.4 BKM Fixed control logic

### 4.3 BKM Core

Puerto	Tamaño	Dirección	Descripción
<b>clk</b>	1 bit	Input	Clock signal
<b>arst</b>	1 bit	Input	Active high asynchronous reset signal
<b>srst</b>	1 bit	Input	Active high synchronous reset signal
<b>enable</b>	1 bit	Input	Active high enable signal
<b>start</b>	1 bit	Input	Active high start signal
<b>format</b>	2 bit	Input	Format specifier (0/1): MSB for complex/real, LSB for 64/32 bits
<b>op</b>	5 bits	Input	Operation code
<b>x_1</b>	64 bits	Input	Real part of input variable z1
<b>y_1</b>	64 bits	Input	Imaginary part of input variable z1
<b>x_2</b>	64 bits	Input	Real part of input variable z2
<b>y_2</b>	64 bits	Input	Imaginary part of input variable z2
<b>x_3</b>	64 bits	Output	Real part of output variable z3
<b>y_3</b>	64 bits	Output	Imaginary part of output variable z3
<b>flags</b>	1 bit	Output	Active high invert output signal
<b>done</b>	1 bit	Output	Active high done signal: Stays asserted until start strobe

Table 4.2: Puertos del bloque op decoder.

Puerto	Tamaño	Dirección	Descripción
<b>clk</b>	1 bit	Input	Clock signal
<b>arst</b>	1 bit	Input	Active high asynchronous reset signal
<b>srst</b>	1 bit	Input	Active high synchronous reset signal
<b>enable</b>	1 bit	Input	Active high enable signal
<b>start</b>	1 bit	Input	Active high start signal
<b>format</b>	2 bit	Input	Format specifier (0/1): MSB for complex/real, LSB for 64/32 bits
<b>op</b>	5 bits	Input	Operation code
<b>x_1</b>	64 bits	Input	Real part of input variable z1
<b>y_1</b>	64 bits	Input	Imaginary part of input variable z1
<b>x_2</b>	64 bits	Input	Real part of input variable z2
<b>y_2</b>	64 bits	Input	Imaginary part of input variable z2
<b>x_3</b>	64 bits	Output	Real part of output variable z3
<b>y_3</b>	64 bits	Output	Imaginary part of output variable z3
<b>flags</b>	1 bit	Output	Active high invert output signal
<b>done</b>	1 bit	Output	Active high done signal: Stays asserted until start strobe

Table 4.3: Puertos del bloque op decoder.