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 $n = 0, 1, 2, ..., N-1$. $s_{m,n}$ es una sequencia 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 difíciles de calcular.

Sistema de coordenadas	Shift sequence	Convergencia	Factor de escala
m	$s_{m,n}$	$\|A_0\|$	$K_m(n \to \inf)$
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	$x_0 = x$	$x_N = K_1 \cdot (x\cos\theta - y\sin\theta)$
		$y_0 = y$	$y_N = K_1 \cdot (y\cos\theta + x\sin\theta)$
		$z_0 = \theta$	$z_N = 0$
1	vectoring	$x_0 = x$	$x_N = K_1 \cdot sign(x) \cdot \sqrt{x^2 + y^2}$
		$y_0 = y$	$y_N = 0$
		$z_0 = \theta$	$z_N = \theta + \tan^{-1}(y/x)$
0	rotation	$x_0 = x$	$x_N = x$
		$y_0 = y$	$y_N = y + x \cdot z$
		$z_0 = \theta$	$z_N = 0$
0	vectoring	$x_0 = x$	$x_N = x$
		$y_0 = y$	$y_N = 0$
		$z_0 = z$	$z_N = z + y/x$
-1	rotation	$x_0 = x$	$x_N = K_{-1} \cdot (x\cosh\theta + y\sinh\theta)$
		$y_0 = y$	$y_N = K_{-1} \cdot (y\cosh\theta + x\sinh\theta)$
		$z_0 = \theta$	$z_N = 0$
-1	vectoring	$x_0 = x$	$x_N = K_{-1} \cdot sign(x) \cdot \sqrt{x^2 - y^2}$
		$y_0 = y$	$y_N = 0$
		$z_0 = \theta$	$z_N = \theta + \tanh^{-1}(y/x)$

Table 1.2: Salidas del algoritmo CORDIC.

Chapter 2

Método de Briggs para calculo de logaritmos

XXXXXXXXXXXXXXSi se encuentra una secuencia d_n tal que la productoria de x con $(1 + d_n 2^{-n})$ es cercana a 1 entonces vale que:XXXXXXXXXXXX

Briggs uso un metodo especial para calcular logaritmos con mucha precision, pero solo de ciertos valores. En particular calculo todos los valores de la forma $1 + q10^{-n}$ siendo 1 un valor entre 1 y 9. Luego desarollo un método para poder calcular el logaritmo de un numero cualquiera en base a tablas ya calculadas.

$$\begin{aligned} x &= x_0 * (1 + q_1 10^{-1}) * (1 + q_2 10^{-2}) * (1 + q_3 10^{-3}) * (1 + q_4 10^{-4})... \\ \ln(x) &= \ln(x_0) + \ln(1 + q_1 10^{-1}) + \ln(1 + q_2 10^{-2}) + \ln(1 + q_3 10^{-3})... \end{aligned} \tag{2.2}$$

$$\begin{aligned} x \prod_{n=1}^{N} (1 + d_n 2^{-n}) &\approx 1 \\ \ln(x) &\approx -\sum_{n=1}^{N} \ln(1 + d_n 2^{-n}) \end{aligned} \tag{2.4}$$

$$\begin{aligned} x_{n+1} &= x_n \cdot (1 + d_n 2^{-n}) \\ L_{n+1} &= L_n - \ln(1 + d_n 2^{-n}) \\ x_1 &= x \\ L_1 &= 0 \end{aligned} \tag{2.6}$$

Chapter 3

BKM

3.1 El algoritmo

3.1.1 Origenes

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 algorithmo de Briggs. Esta similitud nos lleva a una generalización de ese algoritmo: podriamos realizar multiplicaciones por terminos $(1 + d_n 2^{-n})$, donde los d_ns son números complejos elejidos de tal manera que la multiplicación por d_n pueda reducirce a unas pocas sumas. Entonces se define el algoritmo BKM de la siguiente manera:

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

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 π.

3.1.2 Número de constantes

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

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

En la equación **??** podemos observar que la parte real de las constantes posee simería 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). XXXXXXXXXXXXXXXXXXXXXXXXXXXX. Entonces para la parte real necesito 3 para $d_n^i = 0$ y 3 para $d_n^i = \pm 1$. Para la parte imaginaria solo necesito 3 valores ya que d_n^i no influye. XXXXXXXXXXXXXXXXXXXXXXXXXXXX. Siguiendo estas deducciones llegamos a que se necesitan $8 =! = 9$ términos por cada valor de n. Por lo tanto, para

obtener aproximadamente N bits de precisión necesitamos almacenar $8 =! = 9N$ 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 $4N =! = 9/2N$ constantes.

3.2 E-mode

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

$$\begin{cases} E_n \to E_1 e^{L_1} \\ L_n \to 0 \end{cases} \tag{3.3}$$

3.2.1 Canal de datos

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

3.2.2 Canal de control

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

$$2^n \ln(1 + d_n 2^{-n}) \to 1 \qquad as\ n \to +\infty \tag{3.6}$$

3.2.3 Rango de convergencia

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

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 ρ, 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: $e^{z_{in}} = e^{x_{in}+jy_{in}} = e^{j(k_y \mod 8)\frac{\pi}{4}} e^{x_{in}+jy'}$. La multiplicación por $e^{j(k_y \mod 8)\frac{\pi}{4}}$ 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 $k_y \mod 8 = 1$. Allí el término $e^{\frac{j\pi}{4}}$ 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 facilmente a dos sumas. Un truco similar puede usarse para otros posibles valores de $k_y \mod 8$. Entonces definimos K_p y γ_p de la siguiente manera:

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

   Con $p = k_y \mod 8$ 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 sequencia de d_n tal que $E_n \to 1$ entonces $L_n \to L_1 + \ln(E_1)$.

$$\begin{cases} E_n \to 1 \\ L_n \to L_1 + \ln(E_1) \end{cases} \tag{3.9}$$

3.3.1 Canal de datos

$$\begin{cases} L^r_{n+1} = L^r_n - \frac{1}{2}\ln[1 + d^r_n\, 2^{-n+1} - (d^{r\,2}_n + d^{i\,2}_n)\, 2^{-2n}] \\ L^i_{n+1} = L^i_n - d^i_n \arctan\left(\frac{2^{-n}}{1 + d^r_n 2^{-n}}\right) \end{cases} \tag{3.10}$$

3.3.2 Canal de control

$$\begin{cases} e_n = 2^n(E_n - 1) \text{ con 4 digitos de precisión decimal} \\ e_{n+1} = 2(e_n + d_n) + d_n e_n 2^{-n+1} \\ e^r_{n+1} = 2(e^r_n + d^r_n) + (d^r_n e^r_n - d^i_n e^i_n)2^{-n+1} \\ e^i_{n+1} = 2(e^i_n + d^i_n) + (d^i_n e^r_n + d^r_n e^i_n)2^{-n+1} \end{cases} \tag{3.11}$$

3.3.3 Rango de convergencia

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

$$E1 \in T = \{x + j\, y : 0.5 \le x \le 1.3, -\frac{1}{2}x \le y \le \frac{1}{2}x\} \tag{3.12}$$

Siguiendo el algoritmo del 99 el rango de convergencia es:

$$E1 \in T = \{x + j\, y : 0.64 \le x \le 1.4, -\frac{2}{5}x \le y \le \frac{2}{5}x\} \tag{3.13}$$

3.3.4 Reducción del rango de entrada

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

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

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.

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

2. Si me encuentro en la parte superior del semicuadrante 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 $z' = z$

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

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

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

   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 $z_3 = f_{\text{op}}(z1, z2)$ siendo $f_{\text{op}}(z1, z2)$ una operación definida por la varaiable op. A continuación se detalla la lista de operaciones posibles y sus códigos de operación.

Código de operación	Operación	Descripción	Tiempo de ejecución
000000	$\exp(z1)$		N
000000	$\ln(z1)$		N
000000	$\cosh(z1)$	$\frac{e^{z_1}+e^{-z_1}}{2}$	2N
000000	$\sinh(z1)$	$\frac{e^{z_1}-e^{-z_1}}{2}$	2N
000000	$\tanh(z1)$	$\frac{e^{2z_1}-1}{e^{2z_1}+1}$	2N
000000	$\text{arctanh}(z1)$	doble	2N
000000	$\cos(z1)$	doble	2N
000000	$\sin(z1)$	doble	2N
000000	$\arctan(z1)$	doble	2N
000000	$\tan(z1)$	doble	2N

Table 4.1: Operaciones del bkm_fixed.

4.2.2 BKM Pre

Este bloque se encarga de

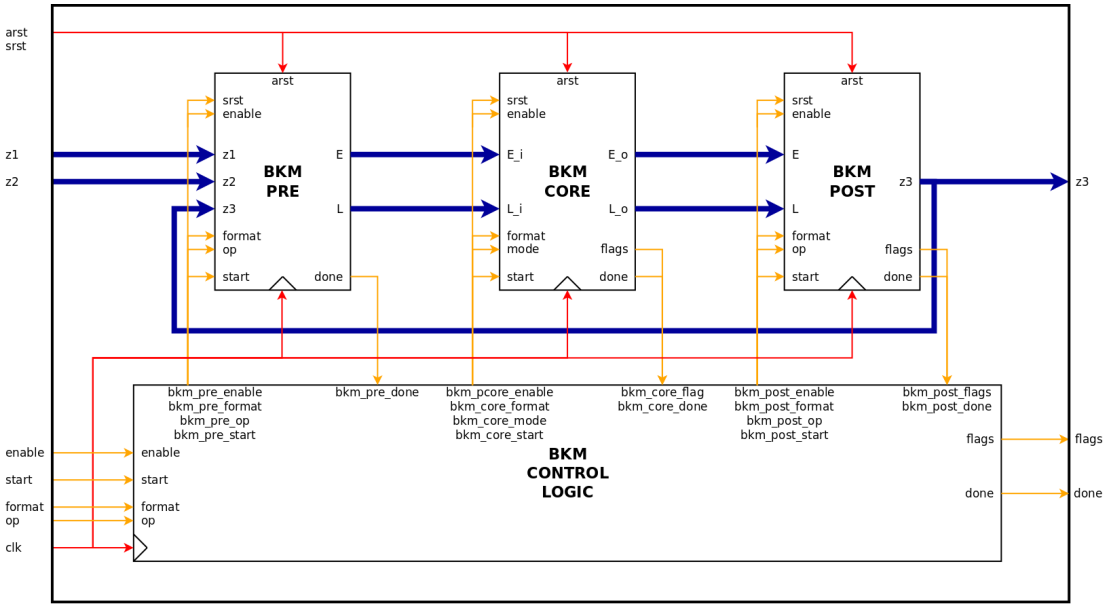

Figure 4.1: Arquitecture del bloque BKM fixed.

4.2.3 BKM Post

4.2.4 BKM Fixed control logic

4.3 BKM Core

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

Table 4.2: Puertos del bloque op decoder.

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

Table 4.3: Puertos del bloque op decoder.