CORDIC

Juan Carlos Llamas Núñez

Abril 2021

1. Introducción

CORDIC es un algoritmo para el cálculo de funciones trigonométricas e hiperbólicas. Toma su nombre del inglés (COordinate Rotation DIgital Computer) y también es conocido como el método dígito a dígito o el algoritmo de Volder. Tiene su origen en 1956, cuando Jack E. Volder [1], que trabajaba en el departamento de electrónica de la empresa de aviación Convair, buscaba remplazar un resolver[1] analógico en la computadora de uno de sus aviones bombarderos. La finalidad era mejorar el rendimiento y la precisión de las operaciones que se realizaban. En su investigación, Volder hizo uso de la siguiente fórmula publicada en *CRC Handbook of Chemestry and Physics*:

Si $tan(\varphi) = 2^{-n}$ y definimos $K_n = \sqrt{1 + 2^{-2n}}$ entonces

$$K_n sin(\theta \pm \varphi) = sin(\theta) \pm 2^{-n} cos(\theta)$$

$$K_n cos(\theta \pm \varphi) = sin(\theta) \mp 2^{-n} cos(\theta)$$

Este resultado se obtiene fácilmente aplicando las identidades del seno suma[2] y coseno suma[3] y la relación $\frac{1}{cos^2(\alpha)} = 1 + tan^2\alpha$.

Las técnicas empleadas por Volder fueron similares a otras publicadas por Henry Briggs algo más de tres siglos antes (1624) o por Robert Flower en 1771. Sin embargo, su principal aportación fue una implementación optimizada para máquinas de estados finitos de baja complejidad.

2. Conceptos previos

Antes de comenzar a explicar el algoritmo, conviene recordar algunos conceptos relacionados con las rotaciones en el plano. Si consideramos la circunferencia de centro $(0, 0)$ y radio 1, las coordenadas cartesianas (x, y) de todos sus puntos P cumplen la ecuación

[1] Un resolver es un dispositivo electromecánico que convierte el movimiento mecánico en una señal electrónica analógica.

[2] $sin(\alpha + \beta) = sin(\alpha)cos(\beta) + cos(\alpha)sin(\beta)$

[3] $cos(\alpha + \beta) = cos(\alpha)cos(\beta) - sin(\alpha)sin(\beta)$

$x^2 + y^2 = 1$. Esos puntos se pueden representar en coordenadas polares como $P = (1, \theta)$ donde θ es el ángulo[4] entre la recta definida por unir el punto P con el origen y el eje de abscisas, y 1 es el radio. Si expresamos el punto en coordenadas cartesianas en función del ángulo tenemos que $P = (cos(\theta), sin(\theta))$. Si queremos mandar el punto P a otro punto Q de la circunferencia, lo podemos hacer mediante una rotación, es decir, al punto $P = (cos(\theta), sin(\theta))$ enviarlo a $Q = (cos(\theta + \omega), sin(\theta + \omega))$. Por las fórmulas del seno suma y coseno suma, esto se puede expresar de forma matricial como:

$$\begin{pmatrix} cos(\omega) & -sin(\omega) \\ sin(\omega) & cos(\omega) \end{pmatrix} \begin{pmatrix} cos(\theta) \\ sin(\theta) \end{pmatrix} = \begin{pmatrix} cos(\theta + \omega) \\ sin(\theta + \omega) \end{pmatrix}.$$

Es decir, podemos codificar una rotación como la aplicación de una función lineal a un vector, que consiste en multiplicarlo por una matriz R_ω por la izquierda. Sacando el coseno fuera podemos descomponer la matriz R_ω como

$$R_\omega = \begin{pmatrix} cos(\omega) & -sin(\omega) \\ sin(\omega) & cos(\omega) \end{pmatrix} = cos(\omega) \begin{pmatrix} 1 & -tan(\omega) \\ tan(\omega) & 1 \end{pmatrix} =$$

$$\frac{1}{\sqrt{1 + tan^2(\omega)}} \begin{pmatrix} 1 & -tan(\omega) \\ tan(\omega) & 1 \end{pmatrix}$$

3. Modos de operación

El algoritmo CORDIC cuenta con dos modos de operación que nos permiten calcular los valores de la gran mayoría de funciones trigonométricas.

3.1. Modo *rotación*

El objetivo de este modo es calcular el seno y el coseno de un determinado ángulo θ dado. El algoritmo es iterativo y en cada paso construye un vector de módulo 1 (que se puede identificar con un punto de la circunferencia goniométrica). Cada nuevo vector va a ser el resultado de rotar el vector construido en la iteración anterior, en sentido horario o antihorario, un ángulo de amplitud ω_i. Estos ángulos van a ser elegidos de manera específica para que, como vamos a ver, las operaciones a realizar se simplifiquen lo máximo posible. Comenzaremos con $v_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ y v_{i+1} se calcula a partir de v_i como

$$v_{i+1} = R_{\omega_i} v_i = \frac{1}{\sqrt{1 + tan^2(\omega_i)}} \begin{pmatrix} 1 & -tan(\omega_i) \\ tan(\omega_i) & 1 \end{pmatrix} v_i =$$

$$\frac{1}{\sqrt{1 + 2^{-2i}}} \begin{pmatrix} 1 & -\delta_i 2^{-i} \\ \delta_i 2^{-i} & 1 \end{pmatrix} v_i = K_i \begin{pmatrix} 1 & -\delta_i 2^{-i} \\ \delta_i 2^{-i} & 1 \end{pmatrix} v_i$$

donde $K_i = \frac{1}{\sqrt{1+2^{-2i}}}$, δ_i es $+1$ o -1, según el ángulo de rotación sea positivo o negativo, y hemos particularizado para los ángulos $\omega_i \in (-\pi/2, \pi/2)$ con módulo $|\omega_i| = arctan(2^{-i})$, es decir, $tan(|\omega_i|) = 2^{-i}$.

[4]Los ángulos los vamos a medir siempre en radianes salvo mención expresa de lo contrario.

Para determinar el valor de δ_i necesitamos almacenar de alguna manera el ángulo asociado al vector v_i. Como la composición de rotaciones es una rotación de ángulo suma, es decir, $R_\alpha R_\beta = R_{\alpha+\beta}$, se sigue que hacer una serie de rotaciones de ángulo ω_i con $i = 0, 1, \dots n$ equivale a hacer una única rotación de ángulo $\phi_n = \sum_{i=0}^{n} \omega_i$. Por tanto, decidir si la rotación se hace en sentido horario o antihorario consiste en comprobar si ϕ_i es mayor o menor que θ, o equivalentemente, si $\theta - \phi_i$ es menor o mayor que 0. Se puede expresar esto de manera explícita con las recurrencias:

$$z_0 = \theta, \quad z_{i+1} = z_i - \delta_i arctan(2^{-i}), \quad \delta_i = \begin{cases} +1, & \text{si} \quad z_i \geq 0 \\ -1, & \text{si} \quad z_i < 0 \end{cases}.$$

De esta forma z_i almacena la diferencia entre el ángulo del que queremos calcular su seno y coseno y el ángulo asociado al vector v_i. Nuestro objetivo es hacer tender z_i a 0. Se podría pensar que z_i es difícil de calcular porque involucra el cómputo de arcotangentes. Sin embargo, en la práctica se guardan en una lookup table (LUT) los valores de los ángulos que nos interesan, es decir, $arctan(2^{-i})$ para $i = 0, 1, \dots n$. Si estos valores, que son siempre los mismos, ya los tenemos precalculados, lo único que tenemos que hacer es sumarlos o restarlos según corresponda. Además, cuando x es suficientemente pequeño podemos aproximar $arctan(x)$ por x, lo que, traducido a nuestro problema, supone que si i es suficientemente grande $arctan(2^{-i}) \approx 2^{-i}$. Esto implica que, en la práctica, es suficiente con guardar en la LUT solamente "unos pocos" ángulos.

Por tanto, si realizamos n iteraciones del algoritmo se tiene que

$$v_n = \left(\prod_{i=0}^{n-1} R_{\omega_i} \right) v_0 = \prod_{i=0}^{n-1} \begin{pmatrix} 1 & -\delta_i 2^{-i} \\ \delta_i 2^{-i} & 1 \end{pmatrix} \left[\left(\prod_{i=0}^{n-1} K_i \right) v_0 \right],$$

donde hemos sacado todas las K_i fuera porque son escalares. Ahora llamamos $K(n) = \prod_{i=0}^{n-1} K_i$ y M_n a la matriz 2x2 resultante de hacer el producto de las matrices. Como el número de iteraciones a realizar es conocido de antemano, podemos inicializar el algoritmo, en lugar de con v_0, con $\hat{v}_0 = K(n)v_0 = \begin{pmatrix} K(n) \\ 0 \end{pmatrix}$. Como hacíamos con los ángulos, estos valores de $K(n)$ pueden estar precalculados para cada n y guardados en una LUT. También se suele almacenar el valor $K = \lim_{n\to\infty} K(n) \approx 0{,}607252925008881256 1694$ para valores de n suficientemente grandes luego, nuevamente, solo tenemos que guardar "unos pocos" valores de $K(n)$. Otras versiones del algoritmo renuncian a la multiplicación de v_0 por K y recuerdan que el resultado aparece multiplicado por un factor de escala $A = \frac{1}{K} \approx 1{,}6467602025812107$. La interpretación geométrica de esta versión es que hacemos rotaciones y homotecias, luego los vectores dejan de tener módulo 1, pero conservan el ángulo, que es lo que verdaderamente nos interesa.

Nos queda todavía multiplicar M_n por v_0, lo cual se hace asociando por la derecha, multiplicando matriz por vector iterativamente. Estas operaciones son todas multiplicaciones de la forma

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \end{pmatrix} = \begin{pmatrix} 1 & -\delta_i 2^{-i} \\ \delta_i 2^{-i} & 1 \end{pmatrix} \begin{pmatrix} x_i \\ y_i \end{pmatrix} = \begin{pmatrix} x_i - \delta_i 2^{-i} y_i \\ y_i + \delta_i 2^{-i} x_i \end{pmatrix}.$$

Aquí se puede apreciar el gran poder de este algoritmo: las operaciones a realizar son sumas, restas y divisiones entre potencias de 2, que no es otra cosa que un desplazamiento

de bits. No hace falta efectuar ningún tipo de multiplicación que requiera un módulo multiplicador externo.

La ejecución de este algoritmo nos lleva al resultado deseado. Según aumenta i, el ángulo de la composición de las rotaciones se acerca cada vez más a θ o, alternativamente, z_i tiende a 0. Por construcción, en cada iteración nos aproximamos más a 0 (en valor absoluto) ya que decidimos rotar en sentido horario o antihorario según $\theta - \phi_i$ sea mayor o menor que 0, respectivamente. Por tanto, el resultado final es un vector $v_n = \begin{pmatrix} x_n \\ y_n \end{pmatrix}$ de módulo 1 y de ángulo en coordenadas polares $\sum_{i=0}^{n-1} \omega_i \approx \theta$, luego sus componentes serán $x_n = cos(\sum_{i=0}^{n-1} \omega_i) \approx cos(\theta)$ e $y_n = sin(\sum_{i=0}^{n-1} \omega_i) \approx sin(\theta)$.

3.2. Modo *vectorización*

En este modo alternativo, el *input* no es un ángulo, sino un vector, no necesariamente unitario, en el primer o el cuarto cuadrante, es decir, con primera coordenada positiva. El objetivo va a ser realizar rotaciones, de manera análoga a como se hacían en el modo *rotación*, para llevar el vector al eje X, es decir, haciendo tender a 0 el valor y_i según aumenta i. Si esta vez inicializamos el valor de z_0 a 0, obtendremos que z_i almacena el valor de las rotaciones que vayamos efectuando. Para decidir en que sentido rotar, en esta ocasión nos fijamos en el signo de y_i y las recurrencias quedan como:

$$v_0 = \begin{pmatrix} x_0 \\ y_0 \end{pmatrix}, \quad z_0 = 0, \quad z_{i+1} = z_i - \delta_i arctan(2^{-i}), \quad \delta_i = \begin{cases} +1, & \text{si} \quad y_i \leq 0 \\ -1, & \text{si} \quad y_i > 0 \end{cases}$$

y v_i calculado como antes, con la salvedad de que no hace falta multiplicar en ningún momento por los K_i. Para un valor de n suficientemente grande se tiene que $y_n \approx 0$, $z_n \approx arctan(\frac{y_0}{x_0})$ y $x_n = \frac{1}{K(n)}\sqrt{x_0^2 + y_0^2}$.

Nótese que el cálculo de x_{i+1}, y_{i+1} y z_{i+1} no cambia respecto al modo rotación. La única diferencia es la manera de calcular el sentido del giro, es decir δ_i, y los valores iniciales del algoritmo x_0, y_0 y z_0.

4. Aplicaciones

Los dos modos de funcionamiento anteriormente descritos nos presentan las siguientes posibilidades de cálculo:

1. **Senos y cosenos.** Para calcular el seno o el coseno de un ángulo θ dado basta aplicar el modo de operación *rotación* (n iteraciones, $x_0 = K(n)$, $y_0 = 0$, $z_0 = \theta$). El valor obtenido en x_n será el coseno y en y_n aparecerá el seno. Nótese que se calcula simultáneamente el seno y el coseno, lo cual puede suponer una mejora en rendimiento.

2. **Cambio de coordenadas.** Podemos pasar de coordenadas polares a cartesianas y viceversa. Para pasar de coordenadas polares a cartesianas se tiene que si $P = (r, \theta)$ en polares, entonces $P = (rcos(\theta), rsin(\theta))$ en coordenadas cartesianas. Basta entonces aplicar el modo *rotación* (n iteraciones, $x_0 = rK(n)$, $y_0 = 0$, $z_0 = \theta$) para obtener $x_n \approx rcos(\theta)$, $y_n \approx rsin(\theta)$. Para pasar de coordenadas cartesianas (x, y) a polares podemos utilizar el modo *vectorización* (n iteraciones, $x_0 = x$, $y_0 = y$, $z_0 = 0$). Entonces en z_n obtendremos el ángulo y en x_n tendremos r multiplicado por el factor de escalado $\frac{1}{K(n)}$. Multiplicar el resultado por $K(n)$ va en contra de la filosofía de este algoritmo, que intenta simplificar al máximo las operaciones de multiplicación con el objetivo de no usar módulos multiplicadores externos. Sin embargo, en este caso es necesario. Aun así, solo necesitamos una multiplicación, por lo que la mejora puede ser considerable. También hay que hacer notar que el vector introducido en el modo *vectorización* ha de tener la primera componente positiva, lo cual no supone ninguna restricción en la práctica porque el cálculo de las razones trigonométricas puede reducirse a ángulos del primer cuadrante simplemente cambiando algunos signos de manera apropiada.

3. **Arcotangentes y módulo de vectores.** El cálculo de la arcotangente de un valor y se deduce inmediatamente de la aplicación del modo *vectorización* (n iteraciones, $x_0 = 1$, $y_0 = y$, $z_0 = 0$). El resultado se obtiene en z_n. Para el módulo, basta recurrir a lo que hacíamos en el cambio de coordenadas cartesianas a polares.

5. Generalizaciones

En 1971, John Stephen Walther [2][3], de Hewlett-Packard (HP), propuso una generalización del algoritmo CORDIC que fue implementada en la calculadora HP-35, la primera calculadora de bolsillo fabricada por la empresa y primera calculadora científica de bolsillo del mundo con funciones trigonométricas, logarítmicas y exponenciales. Introdujo dos parámetros $f(x)$ y m que permitieran un mayor rango de cálculos. Las ecuaciones recursivas quedan:

$$\begin{cases} x_{i+1} = x_i - m\delta_i 2^{-i} y_i \\ y_{i+1} = y_i + \delta_i 2^{-i} x_i \\ z_{i+1} = z_i - \delta_i f(2^{-i}) \end{cases} .$$

Nos damos cuenta de que si $m = 1$ y $f(x) = arctan(x)$, las ecuaciones resultan ser:

$$\begin{cases} x_{i+1} = x_i - \delta_i 2^{-i} y_i \\ y_{i+1} = y_i + \delta_i 2^{-i} x_i \\ z_{i+1} = z_i - \delta_i arctan(2^{-i}) \end{cases} ,$$

es decir, las mismas que para el CORDIC básico. Estudiamos los casos en los que $m = 0$ y $f(x) = x$ (Rotación Lineal) y $m = -1$ y $f(x) = arctanh(x)$ (Rotación Hiperbólica).

5.1. Rotación Lineal

En este caso, las ecuaciones quedan:

$$\begin{cases} x_{i+1} = x_i \\ y_{i+1} = y_i + \delta_i 2^{-i} x_i \\ z_{i+1} = z_i - \delta_i 2^{-i} \end{cases} .$$

Para el modo *rotación*, en el que el sentido de giro era determinado como

$$\delta_i = \begin{cases} +1, & \text{si} \quad z_i \geq 0 \\ -1, & \text{si} \quad z_i < 0 \end{cases}$$

para hacer tender z_i a 0, entonces $x_n = x_0$, $z_n \approx 0$ y se puede probar por inducción que $y_n = y_0 + (z_0 - z_n)x_0 \approx y_0 + z_0 x_0$. Tomando $y_0 = 0$, $x_0 = a$ y $z_0 = b$, tras un número suficientemente grande de iteraciones, obtenemos que $y_n \approx ab$. Análogamente, para el modo *vectorización*, donde hacíamos tender y_i a 0 eligiendo

$$\delta_i = \begin{cases} +1, & \text{si} \quad y_i \leq 0 \\ -1, & \text{si} \quad y_i > 0 \end{cases},$$

podemos concluir que $x_n = x_0$, $y_n \approx 0$ y $z_n = z_0 + \frac{y_0 - y_n}{x_0} \approx z_0 + \frac{y_0}{x_0}$. Eligiendo $z_0 = 0$, $y_0 = a$ y $x_0 = b$, y para un número grande de iteraciones, obtenemos en z_n un valor próximo a a/b.

Con esta generalización hemos conseguido calcular también multiplicaciones y divisiones sin la necesidad de añadir ningún módulo extra, más allá de incluir posibles multiplexores y/o señales de control en la ruta de datos. Seguimos utilizando únicamente sumas, restas y desplazamientos de bits.

5.2. Rotación Hiperbólica

En este caso, las ecuaciones quedan:

$$\begin{cases} x_{i+1} = x_i + \delta_i 2^{-i} y_i \\ y_{i+1} = y_i + \delta_i 2^{-i} x_i \\ z_{i+1} = z_i - \delta_i arctanh(2^{-i}) \end{cases} .$$

Para el modo *rotación* se puede probar [2] que

$$\begin{cases} x_n = B_n(x_0 cosh(z_0) + y_0 sinh(z_0)) \\ y_n = B_n(y_0 cosh(z_0) + x_0 sinh(z_0)) \\ z_n \approx 0 \end{cases} , \text{con } B_n = \prod_{i=0}^{n-1} \sqrt{1 - 2^{-2i}}$$

y para el modo *vectorización* se tiene que

$$\begin{cases} x_n = B_n \sqrt{x_0^2 - y_0^2} \\ y_n \approx 0 \\ z_n = z_0 + arctanh(\frac{y_0}{x_0}) \end{cases} \quad \text{con } B_n \text{ definido como antes.}$$

Con este nuevo modo y elecciones convenientes de los valores de entrada (x_0, y_0, z_0 y modo de operación) podemos calcular las funciones $cosh(x)$, $sinh(x)$ y $arctanh(x)$. Si también podemos combinar los modos de rotación básico, lineal e hiperbólico, podemos generar otras funciones usando igualdades conocidas como:

$$tan(x) = \frac{sin(x)}{cos(x)}, \quad cotan(x) = \frac{cos(x)}{sin(x)}, \quad tanh(x) = \frac{sinh(x)}{cosh(x)},$$

$$cotanh(x) = \frac{cosh(x)}{sinh(x)}, \quad sec(x) = \frac{1}{cos(x)}, \quad cosec(x) = \frac{1}{sin(x)},$$

$$sech(x) = \frac{1}{cosh(x)}, \quad cosech(x) = \frac{1}{sinh(x)}, \quad exp(x) = sinh(x) + cosh(x),$$

$$ln(w) = 2arctanh(y/x) \quad \text{con} \quad x = w + 1 \quad \text{e} \quad y = w - 1 \quad \text{o}$$

$$\sqrt{w} = \sqrt{x^2 - y^2} \ \text{ para } \ x = w + 1/4 \quad \text{e} \quad y = w - 1/4 \ .$$

No debemos olvidar que todo esto lo podemos conseguir simplemente con sumas, restas, desplazamientos y LUTs, necesarias para almacenar los valores de $arctanh(2^{-i})$ para $i = 0, 1, \ldots n$.[5] No necesitamos añadir hardware adicional, salvo las modificaciones obvias en la ruta de datos y la unidad de control del sistema algorítmico para elegir el modo de funcionamiento, tipo de rotación y valores iniciales.

No es muy difícil imaginar que las aplicaciones de este algoritmo van más allá del cálculo de funciones trigonométricas, hiperbólicas, divisiones, productos y raíces cuadradas. Además, se pueden realizar multiplicaciones de números complejos y operaciones con matrices, como calcular soluciones a sistemas lineales, estimar valores propios, descomposición en valores singulares, factorización QR y muchas otras. Algunos campos en los que es usado este algoritmo son el procesamiento de imágenes y señales, sistemas de comunicacción, robótica o diseño gráfico, además de la computación científica y técnica de propósito general.

6. Iteraciones, precisión y convergencia

Una vez conocemos el funcionamiento del algoritmo es conveniente aclarar algunos asuntos que hemos ido dejando en el aire durante la explicación del funcionamiento del mismo. Hemos planteado los resultados de convergencia como ciertos para cualesquiera parámetros de entrada, lo cual no siempre es cierto, y no hemos concretado cómo de grande debe ser n para obtener resultados suficientemente satisfactorios. En esta sección pretendemos despejar las dudas sobre estos aspectos técnicos.

En primer lugar, comenzamos hablando sobre convergencia. Es natural esperar de un algoritmo iterativo que, a medida que se hacen más iteraciones, el resultado obtenido se acerque más al resultado real. Sin embargo, advertimos que esto no siempre es así. Es por tanto conveniente establecer un rango de valores o dominio de los parámetros en el que se cumple esta propiedad. Así, asociada a cada uno de los modos de operación y tipos

[5] También se cumple que $arctanh(x) \approx x$ para x pequeños.

de CORDIC, tendremos unas restricciones en los parámetros. Muchas veces, estos rangos se pueden extender haciendo uso de identidades conocidas. En la práctica, los cálculos del seno, coseno, tangente y arcotangente no presentan ningún problema. Sin embargo, los cálculos relacionados con las funciones hiperbólicas, la exponencial, el logaritmo, la raiz cuadrada y productos y divisiones presentan un rango de valores más restrictivo y requieren del uso de fórmulas, en algunos casos complejas, para extender dichos rangos.

Alternativamente al uso de identidades para ampliar el rango de parámetros, los rusos Vladimir Baykov y Vladimir Smolov propusieron la realización de dobles iteraciones para los cálculos en los que intervinieran estas últimas funciones que resultaban más problemáticas [4]. Al contrario que el método orginial del CORDIC, donde cada iteración se realiza una sola vez, con las iteraciones dobles se repetía cada iteración dos veces. Fueron aún más lejos [5], ya que probaron que para un sistema numérico de base arbitraria B el número de repeticiones de cada iteración para las funciones:

- seno, coseno y arcotangente debe ser $B - 1$.
- logaritmo, exponencial, seno, coseno y arcotangente hiperbólicos debe ser B.
- arcoseno y arcocoseno debe ser $2B - 2$.
- arcoseno y arcocoseno hiperbólicos debe ser $2B$.

Queda por saber que valores de n nos garantizan una precisión suficientemente buena. En la práctica tomando $n = 40$ se obtienen un resultado correcto hasta la décima cifra decimal (en base 10 y unas 30 en base 2). Otros experimentos empíricos [6] nos arrojan que para valores de $n = 16$ y $n = 32$ las cifras decimales correctas son 3 y 7 (del orden de 10 y 20 en base 2), respectivamente, lo que concuerda con la idea que teníamos de que al aumentar n aumentaba la precisión del resultado. Tras estas intuiciones sobre convergencia podemos comprender por qué este algoritmo recibe el nombre de método de dígito a dígito, ya que la convergencia se produce aproximadamente a razón de un bit por iteración.

7. Implementación

Para la implementación nos vamos a restringir al caso del CORDIC básico, ya que los demás casos no aportan nada nuevo, en el sentido de que la estructura del diseño sigue siendo la misma, y solo dificultan su comprensión. Presentamos el algoritmo del CORDIC, el diagrama ASM en la Figura 1, el diagrama de transición de estados en la Figura 2, la ruta de datos en la Figura 3 y la tabla de señales de control en la Figura 4, siguiendo el estándar de diseño algorítmico.

Algorithm 1: CORDIC

$N \leftarrow n$;
$X \leftarrow x_0$;
$Y \leftarrow y_0$;
$Z \leftarrow z_0$;
$i \leftarrow 0$;
$MODE \leftarrow modeOper$;
while $i < N$ **do**
    if $MODE == 1$ **then**
        $\delta_i \leftarrow sign(Z)$;
    else
        $\delta_i \leftarrow -sign(Y)$;
    end
    $Z \leftarrow Z + \delta_i arctan(2^{-i})$;
    //Asignaciones simultáneas;
    $X \leftarrow X - (\delta_i Y >> i)$;
    $Y \leftarrow Y + (\delta_i X >> i)$;
    $i \leftarrow i + 1$;
end

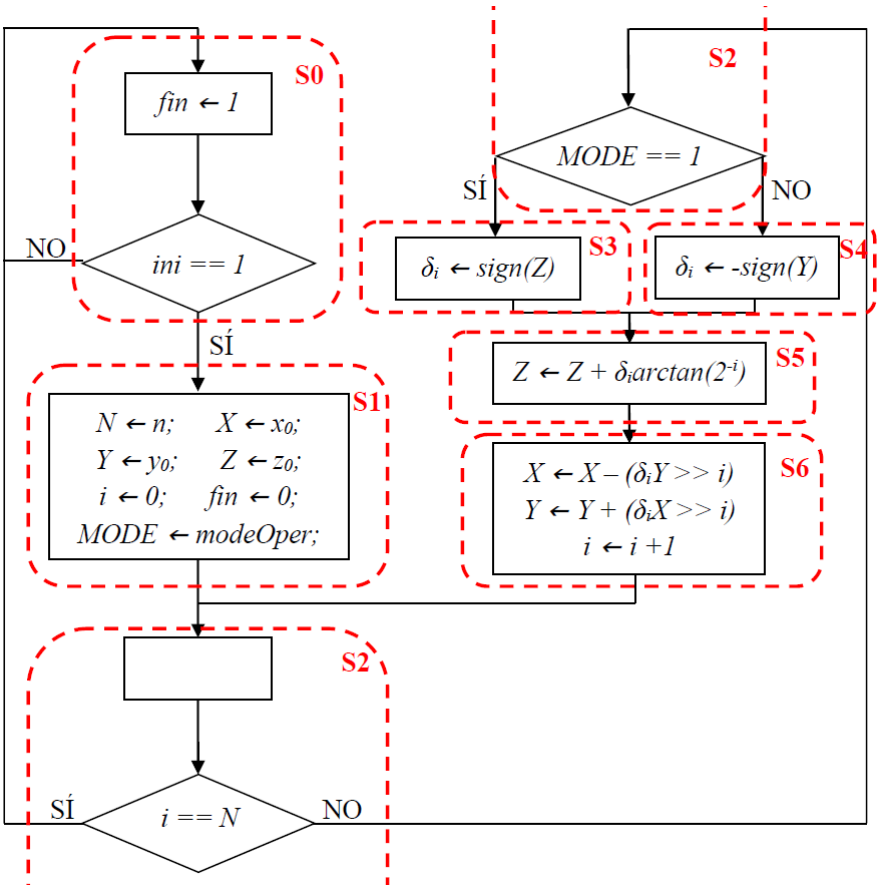

Figura 1: Diagrama ASM

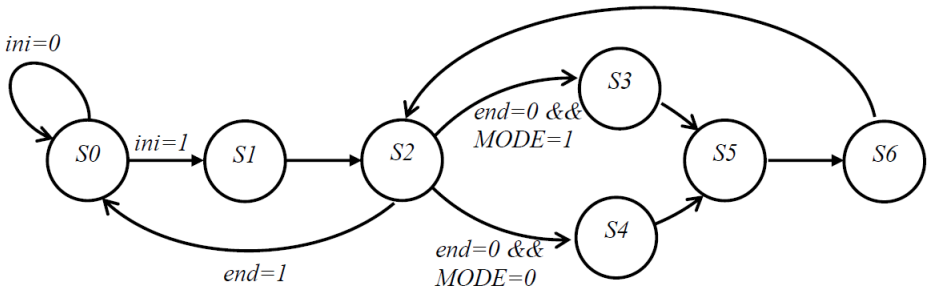

Figura 2: Diagrama de transición de estados

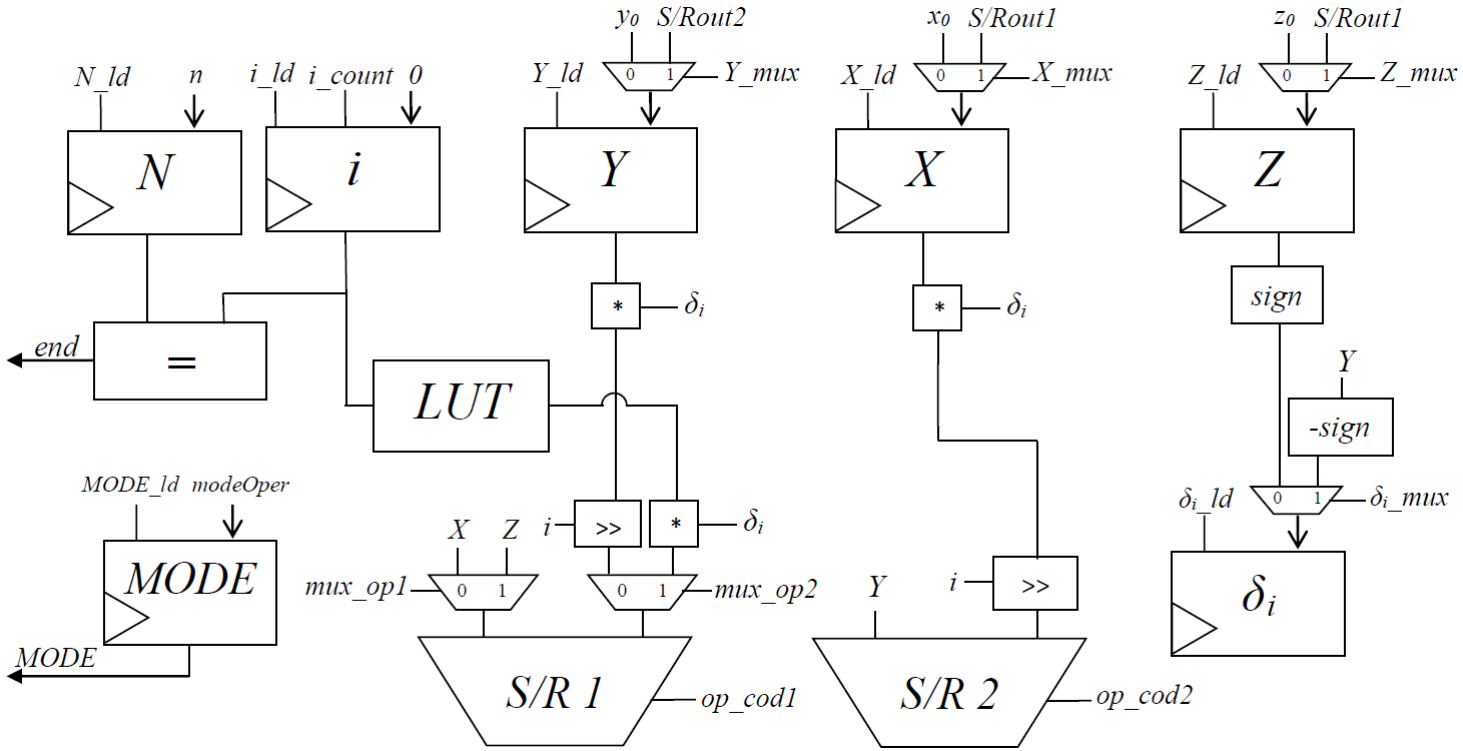

Figura 3: Ruta de datos

Estado	N_ld	i_ld	i_count	MODE_ld	Y_ld	Y_mux	X_ld	X_mux	Z_ld	Z_mux	δ_i_ld	δ_i_mux	mux_op1	mux_op2	op_cod1	op_cod2
S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S1	1	1	0	1	1	0	1	0	1	0	0	0	0	0	0	0
S2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S3	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
S4	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
S5	0	0	0	0	0	0	0	0	1	1	0	0	1	1	+	0
S6	0	0	1	0	1	1	1	1	0	0	0	0	0	0	−	+

Figura 4: Tabla de señales de control

Nótese que en la ruta de datos aparece un módulo etiquetado como "*" que no es un multiplicador. Simplemente opera el signo del número introducido con δ_i, lo que requerirá el uso de puertas lógicas y dependerá de la forma de representación elegida, pero nunca de un módulo multiplicador que es lo que pretendemos evitar a toda costa. De igual manera, los módulos "sign" y "-sign" calculan el signo y menos el signo de manera abstracta, huyendo de las implementaciones concretas. Por otro lado, en la tabla de señales de control tomamos el convenio de que, cuando nos es indiferente el valor que toma la señal, la ponemos a 0, y en el caso de los códigos de operación para los sumadores restadores, que nos dicen si se debe sumar o restar, indicamos la operación a realizar como + o − abstrayéndolo de cada codificación particular.

Referencias

[1] J. Volder. "The CORDIC Computing Technique". En: *Managing Requirements Knowledge, International Workshop on*. Vol. 1. Los Alamitos, CA, USA: IEEE Computer Society, mar. de 1959, pág. 257. DOI: 10.1109/AFIPS.1959.57. URL: https://doi.ieeecomputersociety.org/10.1109/AFIPS.1959.57.

[2] J. S. Walther. "A Unified Algorithm for Elementary Functions". En: *Proceedings of the May 18-20, 1971, Spring Joint Computer Conference*. AFIPS '71 (Spring). Atlantic City, New Jersey: Association for Computing Machinery, 1971, págs. 379-385. ISBN: 9781450379076. DOI: 10.1145/1478786.1478840. URL: https://doi.org/10.1145/1478786.1478840.

[3] John Stephen Walther. "The Story of Unified Cordic". En: *J. VLSI Signal Process. Syst.* 25.2 (jun. de 2000), págs. 107-112. ISSN: 0922-5773.

[4] Vladimir Baykov y Vladimir Somolov. "Hardware implementation of the elementary functions in computers". En: *http://baykov.de/CORDIC1972.htm* (1975).

[5] Vladimir Baykov y Vladimir Somolov. "Special-purpose processors: iterative algorithms and structures". En: *http://baykov.de/Cordic1985.htm* (1985).

[6] Robert Joachim Schweers. "Descripción en VHDL de arquitecturas para implementar el algoritmo CORDIC". En: *http://sedici.unlp.edu.ar/handle/10915/3835* (2002).