Capítulo 1

El algoritmo CORDIC

CORDIC es un acrónimo de COordinate Rotation DIgital Computer que en castellano significa Computadora Digital para Rotación de Coordenadas. El algoritmo original fue propuesto por Jack Volder en el año 1959 [1], con el propósito de calcular funciones trigonométricas mediante la rotación de vectores. La rotación de vectores puede utilizarse a su vez para conversión de sistemas de coordenadas (cartesiano a polar y viceversa), magnitud de vectores y como parte de funciones matemáticas mas complejas como ser la Transformada Rápida de Fourier (Fast Fourier Transform, FFT) y la Transformada Coseno Discreta (Discrete Cosine Transform, DCT).

1.1 Fundamento teórico

El algoritmo CORDIC, proporciona un método iterativo para rotar vectores ciertos ángulos determinados y se basa exclusivamente en sumas y desplazamientos. El algoritmo original tal como fue propuesto inicialmente, describe la rotación de un vector bidimensional en el plano cartesiano. Su funcionamiento se deduce de la fórmula general para rotación de vectores como se muestra a continuación

$$\begin{aligned} x' &= x\cos\theta - y\,\mathrm{sen}\,\theta \\ y' &= y\cos\theta + x\,\mathrm{sen}\,\theta \end{aligned} \qquad (1.1)$$

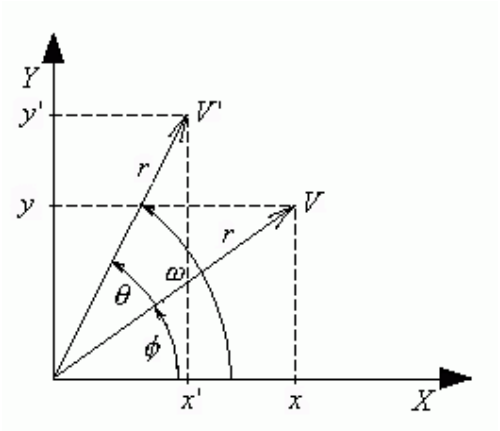

Figura 1.1

La fórmula (1.1) se deduce a partir de las identidades trigonométricas conocidas, en donde

$$V:\begin{cases} x = r\cos\phi \\ y = r\ \mathrm{sen}\,\phi \end{cases} \qquad (1.2.a) \qquad V':\begin{cases} x' = r\cos\omega \\ y' = r\,\mathrm{sen}\,\omega \end{cases} \qquad (1.2.b)$$

Si $r = 1$ y $\theta = \omega - \phi$ entonces $\omega = \theta + \phi$. Utilizando las identidades para el seno y el coseno de la suma de ángulos se tiene

$$\begin{aligned} \mathrm{sen}(\alpha+\beta) &= \mathrm{sen}\,\alpha\cos\beta + \cos\alpha\ \mathrm{sen}\,\beta \\ \cos(\alpha+\beta) &= \cos\alpha\cos\beta - \mathrm{sen}\,\alpha\ \mathrm{sen}\,\beta \end{aligned} \qquad (1.3)$$

Si se sustituye $\omega = \theta + \phi$ en (1.2.b), aplicando (1.3) y (1.2.a), se obtiene

$$\begin{aligned} x' &= \cos(\theta+\phi) = \cos\theta\cos\phi - \mathrm{sen}\,\theta\,\mathrm{sen}\,\phi = x\cos\theta - y\,\mathrm{sen}\,\theta \\ y' &= \mathrm{sen}(\theta+\phi) = \mathrm{sen}\,\theta\cos\phi + \cos\theta\,\mathrm{sen}\,\phi = x\,\mathrm{sen}\,\theta + y\cos\theta \end{aligned}$$

Con lo que se obtiene la fórmula general (1.1), la cual se puede reagrupar considerando que $\cos\theta \neq 0$ [4] de forma tal que

$$\begin{aligned} x' &= \cos\theta\left(x - y\,\mathrm{tg}\,\theta\right) \\ y' &= \cos\theta\left(y + x\,\mathrm{tg}\,\theta\right) \end{aligned} \qquad (1.4)$$

Si además

$$\cos\theta = \sqrt{\cos^2\theta} = \sqrt{\frac{1}{\frac{1}{\cos^2\theta}}} = \frac{1}{\sqrt{\frac{1}{\cos^2\theta}}} = \frac{1}{\sqrt{\frac{\mathrm{sen}^2\theta + \cos^2\theta}{\cos^2\theta}}} = \frac{1}{\sqrt{\frac{\cos^2\theta}{\cos^2\theta} + \frac{\mathrm{sen}^2\theta}{\cos^2\theta}}} =$$

$$= \frac{1}{\sqrt{1 + \mathrm{tg}^2\theta}}$$

entonces la ecuación (1.4) puede expresarse como

$$\begin{aligned} x' &= \frac{x - y\,\mathrm{tg}\,\theta}{\sqrt{1+\mathrm{tg}^2\theta}} \\ y' &= \frac{y + x\,\mathrm{tg}\,\theta}{\sqrt{1+\mathrm{tg}^2\theta}} \end{aligned} \qquad (1.5)$$

En el algoritmo CORDIC, las rotaciones de vectores son reemplazadas por pseudorotaciones [7] como se muestra en la figura 1.2.

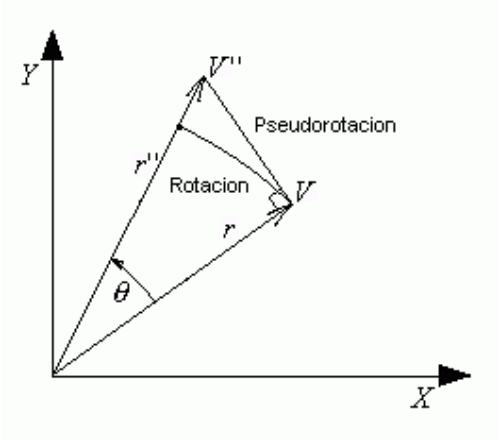

Figura 1.2

Mientras que una rotación real no cambia la magnitud del vector V una vez rotado, una pseudorotación incrementa su magnitud en

$$V'' = V\sqrt{1+\mathrm{tg}^2\,\theta} \tag{1.6}$$

El vector pseudorotado V'' incrementa su magnitud en $\sqrt{1+\mathrm{tg}^2\,\theta}$ respecto del vector original V. La pseudorotación por el ángulo θ está caracterizada por las ecuaciones

$$\begin{aligned} x'' &= x - y\,\mathrm{tg}\,\theta \\ y'' &= y + x\,\mathrm{tg}\,\theta \end{aligned} \tag{1.7}$$

La rotación de un ángulo puede descomponerse en una suma de rotaciones mas pequeñas. Si se asume inicialmente que $x = x_0$, $y = y_0$ y $z = z_0$, luego de n iteraciones para una rotación real se obtiene,

$$\begin{aligned} x_n' &= x\cos\left(\sum_{i=0}^{n-1}\theta_i\right) - y\,\mathrm{sen}\left(\sum_{i=0}^{n-1}\theta_i\right) \\ y_n' &= y\cos\left(\sum_{i=0}^{n-1}\theta_i\right) + x\,\mathrm{sen}\left(\sum_{i=0}^{n-1}\theta_i\right) \\ z_n' &= z - \left(\sum_{i=0}^{n-1}\theta_i\right) \end{aligned} \tag{1.8}$$

mientras que para una pseudorotación,

$$\begin{aligned} x_n'' &= \left(x\cos\left(\sum_{i=0}^{n-1}\theta_i\right) - y\,\mathrm{sen}\left(\sum_{i=0}^{n-1}\theta_i\right)\right)\prod_{i=0}^{n-1}\sqrt{1+\mathrm{tg}^2\,\theta_i} \\ y_n'' &= \left(y\cos\left(\sum_{i=0}^{n-1}\theta_i\right) + x\,\mathrm{sen}\left(\sum_{i=0}^{n-1}\theta_i\right)\right)\prod_{i=0}^{n-1}\sqrt{1+\mathrm{tg}^2\,\theta_i} \\ z_n'' &= z - \left(\sum_{i=0}^{n-1}\theta_i\right) \end{aligned} \tag{1.9}$$

, en donde $\sum_{i=0}^{n-1}\theta_i = \theta$

A las ecuaciones 1.8 y 1.9 se agrega una tercera ecuación para z_n que se denomina *acumulador angular* [7] porque incluye las rotaciones efectuadas hasta el momento.

En la figura 1.3 se muestra un vector *V* rotado sobre otro vector *V'* mediante rotaciones menores.

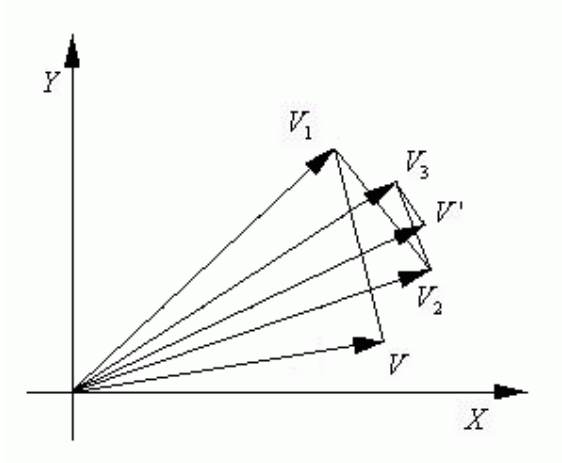

Figura 1.3

Si los ángulos de rotación se restringen de forma tal que $\text{tg}\,\theta = \pm 2^{-i}, i \in \text{N}$, la multiplicación por la tangente, se reduce a una operación de desplazamiento. Los diversos ángulos pueden obtenerse realizando una serie de rotaciones elementales cada vez mas pequeñas [Anexo A]. En cada iteración, en lugar de determinar si se debe rotar o no, se elige el sentido de rotación. Partiendo de las ecuaciones dadas en (1.4), y considerando que $\cos(\theta_i) = \cos(-\theta_i)$, una iteración puede expresarse como

$$\begin{aligned} x_{i+1} &= K_i\left(x_i - y_i d_i 2^{-i}\right) \\ y_{i+1} &= K_i\left(y_i + x_i d_i 2^{-i}\right) \end{aligned} \qquad (1.10)$$

en donde, $K_i = \cos\theta_i = \cos\left(\text{arctg}\, 2^{-i}\right) = \dfrac{1}{\sqrt{1+\text{tg}^2\,\theta_i}} = \dfrac{1}{\sqrt{1+2^{-2i}}}$ y $d_i = \pm 1$ dependiendo del sentido de rotación.

El factor de expansión K_i debe ser multiplicado por las ecuaciones correspondientes a las pseudorotaciones para obtener la rotación real.

El ángulo final de una rotación está determinado por la suma algebraica, según el sentido de rotación, de los ángulos de las rotaciones elementales. Si se elimina K_i de las ecuaciones iterativas, se obtiene un algoritmo basado en sumas y desplazamientos. El factor K_i puede aplicarse al final del proceso como una constante K_n, definida como sigue

$$K_n = \lim_{n\to\infty} \prod_{i=0}^{n} K_i \cong 0{,}6073$$

El valor exacto de K_n depende del número de iteraciones. Se define además

$$A_n = \frac{1}{K_n} = \prod_{i=0}^{n-1} \frac{1}{K_i} = \prod_{i=0}^{n-1} \sqrt{1+2^{-2i}} \cong 1{,}6467$$

Como se explicó anteriormente, en cada paso de iteración, en lugar de decidir si rotar o no, se decide el signo o sentido de la rotación a efectuar. Por lo tanto cada ángulo final se puede representar mediante un vector de signos, en donde cada componente corresponde a un ángulo de la secuencia de ángulos elementales. Dichos ángulos elementales se almacenan en una tabla de búsqueda (Lookup Table, LUT). Con esto último, se modifica el acumulador angular para dar lugar a la ecuación,

$$z_{i+1} = z_i - d_i \operatorname{arctg}\left(2^{-i}\right) \qquad (1.11)$$

Dependiendo del sistema angular con el que se trabaje, se almacenan en la tabla las arcotangentes correspondientes en ese sistema.

El algoritmo CORDIC se opera normalmente en dos modos. El primero, se denomina *rotación* (rotation) que rota el vector de entrada un ángulo específico que se introduce como parámetro. El segundo modo, denominado *vectorización* (vectoring), rota el vector de entrada hacia el eje *X*, acumulando el ángulo necesario para efectuar dicha rotación.

En el caso de una *rotación*, el acumulador angular se inicializa con el ángulo a rotar. La decisión sobre el sentido de rotación en cada paso de iteración, se efectúa para minimizar la magnitud del ángulo acumulado. Por ello, el signo que determina el sentido de rotación, se obtiene del valor de dicho ángulo en cada paso. Para el modo rotación, las ecuaciones son

$$\begin{aligned} x_{i+1} &= x_i - y_i d_i 2^{-i} \\ y_{i+1} &= y_i + x_i d_i 2^{-i} \\ z_{i+1} &= z_i - d_i \operatorname{arctg}\left(2^{-i}\right) \end{aligned} \qquad (1.12)$$

en donde, $d_i = \begin{cases} -1 & ,si\ z_i < 0 \\ 1 & ,si\ z_i \geq 0 \end{cases}$

partiendo de las ecuaciones (1.9), luego de *n* etapas, cuando $z_n \to 0$ [1] entonces $\sum_{i=0}^{n-1} \theta_i = z$ para $x = x_0, y = y_0, z = z_0$ [7] se obtiene

[1] De aquí en más se utiliza w_n para una variable de la forma w_n'', a los efectos de simplificar las expresiones.

$$
\begin{aligned}
x_n &= A_n\left(x_0 \cos z_0 - y_0 \operatorname{sen} z_0\right) \\
y_n &= A_n\left(y_0 \cos z_0 + x_0 \operatorname{sen} z_0\right) \\
z_n &= 0 \\
A_n &= \prod_{i=0}^{n-1} \sqrt{1 + 2^{-2i}}
\end{aligned}
\qquad (1.13)
$$

Para una *vectorización,* el ángulo ingresado se rota para alinearlo con el eje *X*. Para obtener este resultado, en lugar de minimizar la magnitud del acumulador angular, se minimiza la magnitud del componente *y*, ya que si *y = 0* entonces el vector se encuentra sobre el eje *X*. Asimismo se utiliza el signo del componente *y* para determinar la dirección de rotación. Si el acumulador angular se inicializa con cero, al final del proceso contendrá el ángulo de rotación adecuado. Por lo tanto se pueden deducir las siguientes ecuaciones

$$
\begin{aligned}
x_{i+1} &= x_i - y_i d_i 2^{-i} \\
y_{i+1} &= y_i + x_i d_i 2^{-i} \\
z_{i+1} &= z_i - d_i \operatorname{arctg}\left(2^{-i}\right)
\end{aligned}
\qquad (1.14)
$$

en donde, $d_i = \begin{cases} -1 & ,si\ y_i \geq 0 \\ 1 & ,si\ y_i < 0 \end{cases}$

partiendo de las ecuaciones (1.9), luego de *n* etapas cuando $y_n \to 0$ entonces $\operatorname{tg}\left(\sum_{i=0}^{n-1} \theta_i\right) = -\frac{y}{x}$ para $x = x_0,\ y = y_0, z = z_0$ [7] se obtiene

$$
\begin{aligned}
x_n &= A_n \sqrt{x_0^2 + y_0^2} \\
y_n &= 0 \\
z_n &= z_0 + \operatorname{arctg}\left(\frac{y_0}{x_0}\right) \\
A_n &= \prod_{i=0}^{n-1} \sqrt{1 + 2^{-2i}}
\end{aligned}
\qquad (1.15)
$$

El algoritmo CORDIC está restringido a ángulos θ tales que $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$ debido a que los ángulos elementales convergen sólo dentro de esos límites [Anexo A] [4]. Para tratar ángulos que caen fuera de ese rango, se puede reducir al primer o cuarto cuadrante.

1.2 Cálculo del seno y del coseno

El modo rotación del algoritmo CORDIC puede utilizarse para calcular el seno y el coseno de un ángulo en forma simultánea. Si se inicializa $y_0 = 0$, de la fórmula (1.13) se obtiene

$$\begin{aligned} x_n &= A_n x_0 \cos z_0 \\ y_n &= A_n x_0 \operatorname{sen} z_0 \\ z_n &= 0 \\ A_n &= \prod_{i=0}^{n-1} \sqrt{1+2^{-2i}} \end{aligned} \qquad (1.16)$$

Si además $x_0 = \frac{1}{A_n}$, de (1.16) se tiene que

$$\begin{aligned} x_n &= \cos z_0 \\ y_n &= \operatorname{sen} z_0 \\ z_n &= 0 \end{aligned} \qquad (1.17)$$

que calcula el seno y el coseno del ángulo inicial z_0.

1.3 Transformación de coordenadas polares a cartesianas y viceversa

La transformación *T*, que hace corresponder coordenadas polares a cartesianas, se define como

$$T: \begin{cases} x = r\cos\theta \\ y = r\operatorname{sen}\theta \end{cases} \qquad (1.18)$$

Si se reemplaza en (1.13), $x_0 = \frac{1}{A_n} r$ (módulo)

$y_0 = 0$

$z_0 = \theta$ (fase)

se obtiene la transformación *T*.

La transformación de coordenadas cartesianas a polares se puede llevar a cabo utilizando la transformación inversa *T'*

$$T': \begin{cases} r = \sqrt{x^2 + y^2} \\ \theta = \operatorname{arctg}\left(\frac{y}{x}\right) \end{cases} \qquad (1.19)$$

Utilizando el modo vectorización de CORDIC, se puede sustituir en (1.15)

$$\begin{aligned} x_0 &= x \\ y_0 &= y \\ z_0 &= 0 \end{aligned}$$

Sin embargo el algoritmo también computa la constante A_n que puede ser eliminada multiplicando por su inverso, lo cual agrega cierta complejidad al algoritmo inicial ya que se precisa de un multiplicador con el mismo ancho en bits que el resultado.

1.4 Arcotangente y módulo de un vector

El arcotangente se puede calcular utilizando el modo vectorización de CORDIC.
Si se inicializa $z_0 = 0$ y se sustituye en (1.15), se obtiene el término z_n

$$\begin{aligned} x_n &= A_n\sqrt{x_0^2 + y_0^2} \\ y_n &= 0 \\ z_n &= z_0 + \text{arctg}\left(\frac{y_0}{x_0}\right) \\ A_n &= \prod_{i=0}^{n-1}\sqrt{1+2^{-2i}} \end{aligned} \qquad (1.20)$$

El término x_n representa el módulo de un vector en el espacio bidimensional en donde x_0 e y_0 son sus componentes. Nuevamente para eliminar a la constante A_n se debe agregar al algoritmo un multiplicador.

1.5 CORDIC generalizado

El algoritmo CORDIC básico puede ser generalizado para proveer una herramienta aún mas potente para el cálculo de funciones. Esta generalización fue propuesta por Walther [5]. Se define

$$\begin{aligned} x_{i+1} &= x_i - \mu\ y_i d_i 2^{-i} \\ y_{i+1} &= y_i + x_i d_i 2^{-i} \\ z_{i+1} &= z_i - d_i f\left(2^{-i}\right) \end{aligned} \qquad (1.21)$$

La única diferencia que presenta este algoritmo con el básico, es la introducción del parámetro μ en la ecuación correspondiente al componente x y la función *f(.)* que adquiere significado según el sistema que se utilice. El parámetro μ y la función *f(.)* pueden asumir uno de los tres valores que se muestran a continuación

$\mu = 1$	y	$f(x) = \text{arctg}(x)$	Rotación Circular (CORDIC básico)
$\mu = 0$	y	$f(x) = x$	Rotación Lineal
$\mu = -1$	y	$f(x) = \text{tgh}^{-1}(x)$	Rotación Hiperbólica

1.5.1 Caso lineal

Si se parte de (1.21) con $\mu = 0$ y $f(x) = x$ se obtiene

$$\begin{aligned} x_{i+1} &= x_i \\ y_{i+1} &= y_i + x_i d_i 2^{-i} \\ z_{i+1} &= z_i - d_i 2^{-i} \end{aligned} \qquad (1.22)$$

Para el modo rotación, donde $d_i = \begin{cases} -1 & ,si\ z_i < 0 \\ 1 & ,si\ z_i \geq 0 \end{cases}$

la rotación lineal produce

$$\begin{aligned} x_n &= x_0 \\ y_n &= y_0 + x_0 z_0 \\ z_n &= 0 \end{aligned} \qquad (1.23)$$

Por lo tanto a partir de estas ecuaciones puede obtenerse el producto de dos valores *a.b*, sustituyendo

$$\begin{aligned} x_0 &= a \\ y_0 &= 0 \\ z_0 &= b \end{aligned}$$

En modo vectorización, donde $d_i = \begin{cases} -1 & ,si\ y_i \geq 0 \\ 1 & ,si\ y_i < 0 \end{cases}$

la rotación lineal produce

$$\begin{aligned} x_n &= x_0 \\ y_n &= z_0 - \frac{y_0}{x_0} \\ z_n &= 0 \end{aligned} \qquad (1.24)$$

Por lo tanto a partir de estas ecuaciones puede obtenerse el cociente de dos magnitudes *a/b*, sustituyendo

$$\begin{aligned} x_0 &= b \\ y_0 &= -a \\ z_0 &= 0 \end{aligned}$$

1.5.2 Caso hiperbólico

Si se parte de (1.21) con μ = *-1* y $f(x) = \text{tgh}^{-1}(x)$ se obtiene

$$\begin{aligned} x_{i+1} &= x_i + y_i d_i 2^{-i} \\ y_{i+1} &= y_i + x_i d_i 2^{-i} \\ z_{i+1} &= z_i - d_i \ \text{tgh}^{-1}\left(2^{-i}\right) \end{aligned} \qquad (1.25)$$

Para el modo rotación, donde $d_i = \begin{cases} -1 & ,si\ z_i < 0 \\ 1 & ,si\ z_i \geq 0 \end{cases}$

la rotación hiperbólica produce

$$\begin{aligned} x_n &= A_n\left(x_0 \cosh z_0 + y_0 \operatorname{senh} z_0\right) \\ y_n &= A_n\left(y_0 \cosh z_0 + x_0 \operatorname{senh} z_0\right) \\ z_n &= 0 \\ A_n &= \prod_{i=0}^{n-1} \sqrt{1 - 2^{-2i}} \cong 0{,}80 \end{aligned} \qquad (1.26)$$

Para el modo vectorización, donde $d_i = \begin{cases} -1 & ,si\ y_i \geq 0 \\ 1 & ,si\ y_i < 0 \end{cases}$

la rotación hiperbólica produce

$$\begin{aligned} x_n &= A_n \sqrt{x_0^2 - y_0^2} \\ y_n &= 0 \\ z_n &= z_0 + \operatorname{tgh}^{-1}\left(\frac{y_0}{x_0}\right) \\ A_n &= \prod_{i=0}^{n-1} \sqrt{1 - 2^{-2i}} \end{aligned} \qquad (1.27)$$

Las rotaciones elementales en el sistema hiperbólico no convergen. Sin embargo Walther [5] demostró que se pueden conseguir los resultados esperados si se repiten algunas de las iteraciones efectuadas [6].

Las funciones trigonométricas hiperbólicas equivalentes a las funciones trigonométricas circulares pueden obtenerse de manera similar a partir de la ecuación 1.26.

1.6 Arquitecturas para implementar el algoritmo CORDIC

1.6.1 La arquitectura Bit-Paralela Iterativa

Cuando el algoritmo CORDIC se implementa en hardware, una de las arquitectura correspondientes es la arquitectura bit-paralela iterativa [3]. La denominación de paralela se debe a la forma en que se opera con las componentes *X*, *Y* y *Z*. En esta arquitectura cada etapa del algoritmo consiste de un registro para almacenar la salida, una unidad de desplazamiento y un sumador algebraico. Al comenzar el cálculo, los valores iniciales para x_0, y_0 y z_0 ingresan en forma paralela a los registros a través del multiplexor. El bit más significativo del componente *Z* ó *Y* en cada paso de iteración determina la operación a efectuar por el sumador algebraico en modo rotación o vectorización respectivamente. Las señales correspondientes a los componentes *X* e *Y* son desplazadas y luego restadas o sumadas a las señales sin desplazar, correspondientes al componente opuesto. El componente *Z* combina los valores almacenados en el registro con valores que obtiene de una tabla de búsqueda (Lookup Table, LUT) de arcotangentes precalculadas con una cantidad de

entradas proporcional a la cantidad de iteraciones. Esta tabla de búsqueda puede ser implementada como una memoria ROM. La dirección de memoria cambia acorde al número de iteración. Para *n* iteraciones el valor a calcular puede ser obtenido en la salida. Se requiere de un controlador que puede ser implementado como una máquina de estados para controlar los multiplexores, la cantidad de desplazamientos y el direccionamiento de las constantes precalculadas. Esta representación tiene como ventaja el uso eficiente de hardware, debido a que los recursos (registros, multiplexores, unidades de desplazamiento y sumadores) son reutilizados en cada iteración. En la figura 1.4 se muestra el esquema correspondiente a la arquitectura descripta. La señal Modo especifica el modo de operación (Rotación o Vectorización). Se considera además en el esquema que las componentes tienen un ancho de *m* bits. Sin embargo se puede ampliar el ancho en bits de los buses internos para obtener una mejor exactitud [6].

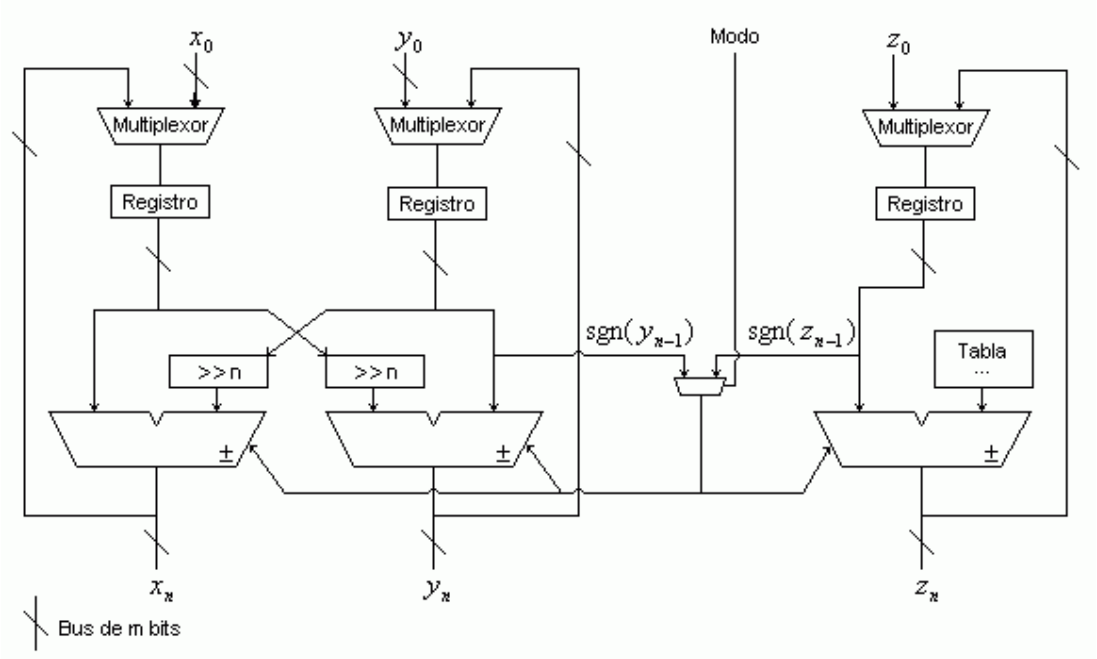

Figura 1.4

1.6.2 La arquitectura Bit-Paralela Desplegada

En lugar de almacenar el resultado de cada paso de iteración en registros y volver a utilizar los mismos recursos, el diseño puede desplegarse [3] como muestra la Figura 1.5. El diseño se separa en etapas correspondientes a cada iteración. Cada etapa está compuesta por los mismos componentes, dos unidades de desplazamiento y dos sumadores algebraicos. Por consiguiente la salida de una etapa corresponde a la entrada de la siguiente etapa. Los valores iniciales para x_0, y_0 y z_0 se ingresan en paralelo a la primera etapa.

Este diseño introduce dos ventajas importantes. Las unidades de desplazamiento así como las constantes correspondientes a cada iteración pueden ser cableadas. La segunda ventaja radica en que el circuito es puramente combinatorio, y por lo tanto no se necesita de una unidad de control, simplificando enormemente el diseño. La desventaja que presenta esta arquitectura es la enorme cantidad de espacio que requiere su implementación.

En el esquema se supone un ancho de palabra de *m* bits. La señal Modo indica el modo de operación al igual que en el caso iterativo.

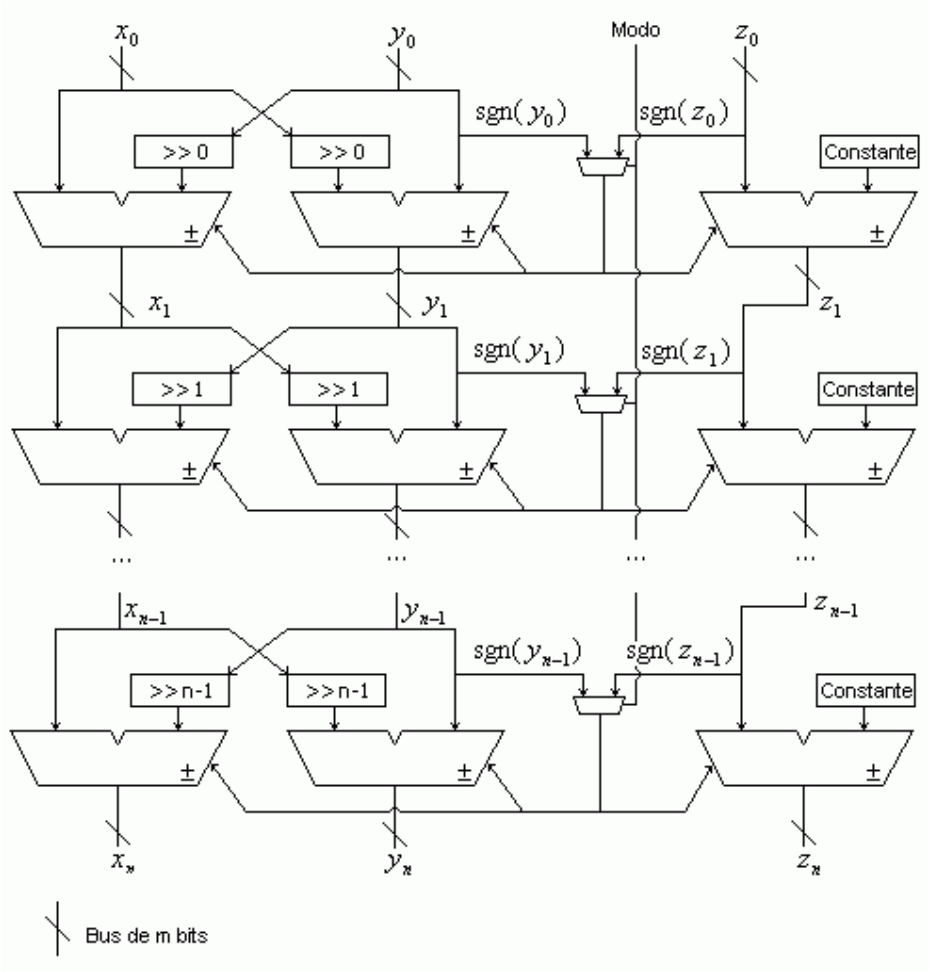

Figura 1.5

1.6.3 La arquitectura Bit-Serie Iterativa

En un diseño en serie, se procesa un bit por vez, con lo cual las interconexiones se reducen al ancho de un bit [1] [3]. El desempeño δ de esta arquitectura se puede calcular como

$$\delta = \frac{velocidad\ de\ reloj}{número\ de\ iteraciones \cdot ancho\ de\ palabra}$$

La arquitectura resultante se muestra en la Figura 1.6. Cada etapa del algoritmo consiste en un multiplexor, un registro de desplazamiento y un sumador algebraico bit-serie. El sumador algebraico se implementa como un sumador completo en el que una resta se lleva a cabo sumando el complemento a dos del valor a restar. La operación a efectuar por el sumador algebraico se obtiene del bit de signo del componente *z* para el modo rotación. Para el modo vectorización, del bit de signo se obtiene del componente *y*.

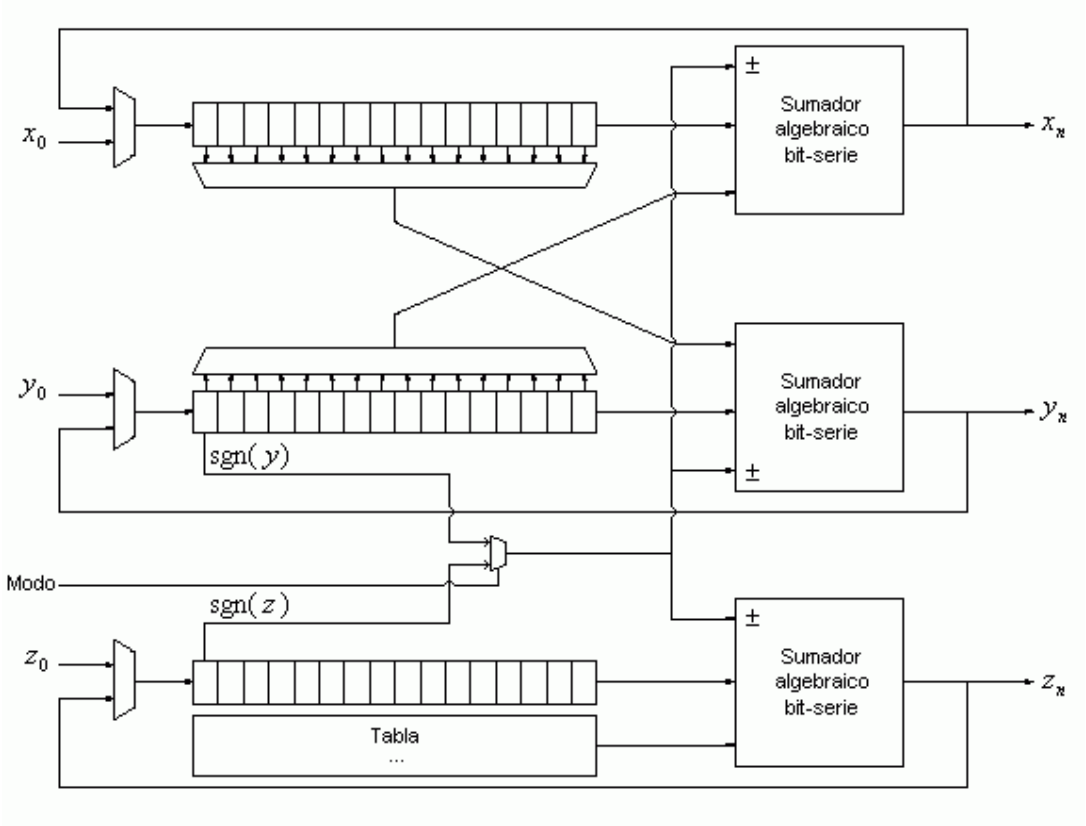

Figura 1.6

La operación de desplazamiento (multiplicación por 2^{-i}) se lleva a cabo leyendo el bit *i – 1* considerado a partir del extremo derecho del registro de desplazamiento. Se puede usar un multiplexor para cambiar la posición de acuerdo a la iteración actual. Los valores iniciales para x_0, y_0 y z_0 ingresan al registro de desplazamiento por el extremo izquierdo en el esquema. Cuando el primer bit que fue ingresado al registro es procesado por el sumador algebraico, el multiplexor permite nuevamente el ingreso de los bits sumados al registro de desplazamiento. Por último, cuando se han completado todas las iteraciones los multiplexores permiten el ingreso de un nuevo valor al registro de desplazamiento y el valor calculado se obtiene en la salida. Alternativamente la carga y lectura del registro de desplazamiento puede efectuarse en paralelo para simplificar el diseño. La desventaja que presenta esta arquitectura con respecto a las que procesan los vectores de entrada en forma paralela es que se introduce un retardo proporcional al ancho de palabra en cada etapa, ocasionado por los desplazamientos. Por otra parte se requiere de hardware de control mas complejo. La ventaja que presenta esta arquitectura es que los buses de interconexión entre los componentes son del ancho de un bit y el registro de desplazamiento está integrado con el registro intermedio, minimizando espacio al momento de su implementación.