
PONTIFICIA **UNIVERSIDAD CATÓLICA** DEL PERÚ

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

# IMPLEMENTACIÓN DE ARQUITECTURAS PARA EL CÁLCULO DE FUNCIONES TRASCENDENTALES EMPLEANDO EL ALGORITMO CORDIC EN UN FPGA.

**Tesis para Optar el Título de:**

**INGENIERO ELECTRÓNICO**

**Presentado por:**

**CARLA PAOLA AGURTO RÍOS**

**Lima – Perú**

**2006**

## RESUMEN

Al implementar un algoritmo de procesamiento digital de señales en hardware es muy común encontrarse con funciones matemáticas trascendentales las cuales, en principio, se pueden implementar usando la serie de Taylor o diseñando un hardware específico para cada función. A fin de mejorar su rendimiento se desarrolló el algoritmo Coordenado Circular, Hiperbólico y Lineal (CORDIC), el cual reduce tanto el uso de compuertas lógicas como el número de iteraciones empleadas al implementar una función trascendental.

Con este trabajo se diseñaron distintos tipos de arquitectura para implementar este algoritmo en el sistema CORDIC, las cuales son: Iterativa, Serial y Pipeline. Son parametrizables el tamaño de bus de datos (12, 16, 24 y 32 bits) y el modo de operación del algoritmo. De esta manera se puede escoger la arquitectura más óptima según la función que se necesite implementar.

Ejemplos de estas aplicaciones son: el diseño de filtros adaptativos, FFT (Transformada Rápida de Fourier), redes neuronales, demoduladores, DCT (Transformada Rápida del Coseno), tratamiento de imágenes, entre otros; de manera que estas aplicaciones puedan funcionar en tiempo real.

**Eliminado:** S

**Eliminado:** Siendo

**Eliminado:** en

**Eliminado:** en

El sistema digital descrito se implementó en su totalidad usando el lenguaje de descripción de hardware VHDL y se simuló usando el programa QUARTUS II 5.0 de ALTERA CORPORATION.

A Dios que siempre me ayuda.

A mi familia que siempre me apoya.

A mis amigos que siempre me animan.

GRACIAS!!!

# INTRODUCCIÓN

Es muy común encontrarse con funciones trascendentales al implementar en hardware un algoritmo de procesamiento digital de señales e imágenes, ya que esto implica el desarrollo de un hardware específico para cada función utilizando algoritmos iterativos clásicos como: el método de aproximación de Newton o el método de la serie de Taylor; métodos que son difíciles de implementar en dispositivos electrónicos debido a la gran carga computacional que generalmente exigen esa clase de algoritmos por lo que resulta muy complicado su implementación física. Han habido intentos exitosos de optimizar estos algoritmos como la transformada rápida de Fourier.

En 1959 se crea un algoritmo llamado CORDIC, el cual se basa en rotaciones y desplazamientos; estas operaciones que presenta el algoritmo son simples de implementar, así mismo se pueden parametrizar en precisión delimitando el número de iteraciones y el tamaño del bus de datos de los vectores. Estas modificaciones son muy útiles, debido a que en ciertos casos los resultados de las operaciones no requieren de la misma precisión. Es por ello que este algoritmo mejora el rendimiento del sistema a utilizar además de disminuir el tiempo de respuesta.

El presente trabajo implementa este algoritmo en hardware en tres sistemas coordenados: Circular, Hiperbólico y Lineal, y con tres tipos de arquitecturas: Serial, Iterativo y Pipeline que son parametrizables de acuerdo al tamaño del bus

de datos entre los valores de 12, 16, 24 y 32 bits; además de poder escoger el tipo de función a realizar.

La precisión obtenida de las operaciones empleando el algoritmo en sus diferentes arquitecturas fue comparado con los obtenidos en MATLAB con lo que se define el error relativo del sistema. Para obtener un bajo error relativo fue necesario hacer un análisis independiente de cada arquitectura sobre el formato de bits que le fue asignado a cada modo de operación en cada bus de datos. También se tuvieron en cuenta factores de desbordamiento al realizar las operaciones.

Para el sistema coordenado Hiperbólico y Lineal fue necesario, por su limitado rango de convergencia, expandirlo utilizando métodos alternativos propuestos por Hu [3]. Además se implementó como casos particulares las funciones exponencial y logaritmo natural en la arquitectura iterativa.

Es objetivo de este trabajo el proponer diferentes tipos de arquitectura, para diferentes tipos de buses de datos, de tal manera que se pueda escoger alguno de ellos en la implementación de otros algoritmos, por tal motivo se expondrán dichos resultados en el Capitulo 4. Estos resultados comprenden el número de compuertas lógicas utilizadas, la frecuencia máxima de operación y el error relativo de los resultados tomando como base de comparación, los hallados con el programa MATLAB.

# CAPÍTULO 1

## DESARROLLO DEL ALGORITMO CORDIC

## 1.1 DESCRIPCIÓN DEL ALGORITMO CORDIC

El algoritmo trigonométrico llamado CORDIC, basado en rotaciones de vectores; fue desarrollado en 1959 por Jack Volder [1] como una solución digital en tiempo real a los problemas de navegación; estas incluían operaciones trigonométricas derivadas de la ecuación de rotación de vectores para el sistema circular como son: seno, coseno, arco tangente, conversión de tipo de datos, etc. Años más tarde, en 1971, Walter [2] generalizó el algoritmo para operaciones con funciones hiperbólicas y las funciones obtenidas para la ecuación de rotación en el sistema lineal; estas funciones incluyen: multiplicación, división en el sistema lineal, seno hiperbólico, coseno hiperbólico, arco tangente hiperbólica, logaritmo natural, exponencial, entre otros.

Existen 2 modos de operación de este algoritmo llamados ROTATION y VECTORING, el primero tiene como datos de entrada el valor del vector y el ángulo que debe rotarse, de donde se obtienen las funciones de tipo seno, coseno, seno hiperbólico, coseno hiperbólico, multiplicación y las derivadas de éstas, mientras que en el segundo se requieren como datos de entrada la magnitud del vector y el argumento angular a ser rotado, de esta manera obtendremos las funciones del tipo arco tangente, división y las derivadas de éstas.

## 1.2 SISTEMAS UTILIZADOS EN EL ALGORITMO CORDIC

El algoritmo se implementó en los 3 sistemas coordenados posibles: Circular, Hiperbólico y Lineal

### 1.2.1 SISTEMA COORDENADO CIRCULAR

Este sistema es utilizado para realizar las siguientes operaciones: seno, coseno, arco tangente, arco coseno, arco seno, vector magnitud y transformación de coordenadas polares a cartesianas y de cartesianas a polares. A continuación se muestra las ecuaciones generales de rotación para un vector (1), cuyas coordenadas son representadas por las variables X e Y, y cuyo ángulo respecto a eje coordenado X, se encuentra en la variable Z.

$$\begin{aligned} X_{i+1} &= X_i.\cos(\alpha_i) - Y_i.sen(\alpha_i) \\ Y_{i+1} &= Y_i.\cos(\alpha_i) + X_i.sen(\alpha_i) \\ Z_{i+1} &= Z_i - \alpha_i \end{aligned} \tag{1}$$

Se divide a las dos primeras ecuaciones por el coseno del mismo ángulo para expresar éstas en función de la tangente multiplicada por un factor.

$$\begin{aligned} X_{i+1} &= \cos(\alpha_i).(X_i - Y_i.\tan(\alpha_i)) \\ Y_{i+1} &= \cos(\alpha_i).(Y_i + X_i.\tan(\alpha_i)) \\ Z_{i+1} &= Z_i - \alpha_i \end{aligned} \tag{2}$$

A su vez $tan(\alpha_i)$ se la reemplaza por $2^{-i}$, esta potencia de 2 hace que la multiplicación se transforme en un desplazamiento simple cuando se procesan los datos. Como la tangente de un ángulo puede ser negativa o positiva

dependiendo del cuadrante en que se encuentre el ángulo (I o IV), por ello se coloca la variable $d_i$ como se muestra en (3) que puede tomar valores de 1 o –1 dependiendo del modo de funcionamiento que está desarrollando el algoritmo.

$$\begin{aligned} X_{i+1} &= X_i - d_i.Y_i.2^{-i} \\ Y_{i+1} &= Y_i + d_i.X_i.2^{-i} \\ Z_{i+1} &= Z_i - d_i.\tan^{-1}(2^{-i}) \end{aligned} \tag{3}$$

Como se puede observar en (3), no se considera el factor del coseno multiplicado en cada iteración. Tomando como referencia que la $tan(\alpha_i)=2^{-i}$, entonces el valor del coseno es como se muestra en la ecuación.

$$\cos(\alpha_i) = \frac{1}{\sqrt{1+2^{-2i}}} \tag{4}$$

Esto se puede apreciar mejor gráficamente (figura 1.2), donde en cada rotación el tamaño del vector varía a falta del factor coseno expuesto en (4). Estas iteraciones comprenden los siguientes valores: 0,1,2,3 ….N, donde N es el número máximo de iteraciones.

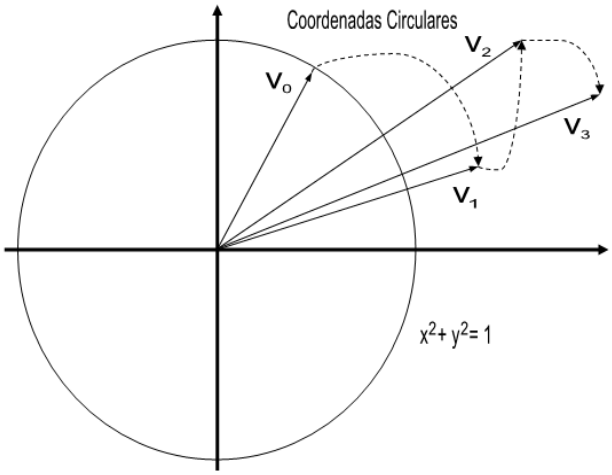



**Figura 1.2 Trayectoria de Rotación para el sistema Circular.**

Por ello es necesario considerar ese factor para todas las iteraciones realizadas cuando se procesa el algoritmo.

### 1.2.1.1 MODO *ROTATION* PARA EL CORDIC CIRCULAR.

En este modo se desea que el vector original de entrada ($X_0$ e $Y_0$) rote un ángulo determinado, este valor de ángulo ($Z_0$) también es un parámetro de entrada; y para completar ese giro se hacen pequeñas rotaciones angulares hasta que se complete el ángulo de entrada $Z_0$. Dado que los ángulos están definidos como potencias de 2, el mayor ángulo a rotar será π/4 (45°); es por eso que se debe de modificar el sentido de giro de la rotación si es que al rotar el sistema se pasa del ángulo $Z_0$, haciendo que el nuevo ángulo girado sea π/8 (22.5°). Este sentido de giro lo definirá la variable $d_i$ mencionada en (3), donde:

$$\begin{aligned} &\text{Si } Z_i \geq 0 \text{ entonces } d_i = 1 \\ &\text{caso contrario } d_i = -1 \end{aligned} \qquad (5)$$

De esta manera según (5) se va obteniendo una mejor aproximación del ángulo, ya que dependiendo de si el ángulo acumulado es negativo o positivo, se podrá cambiar de sentido de rotación. Después de la (n-1)-ésima iteración con sus correspondientes giros, los valores finales de $X_n$, $Y_n$ y $Z_n$ quedan definidos como:

$$\begin{aligned} X_n &= K.(X_0.\cos(Z_0) - Y_0.sen(Z_0)) \\ Y_n &= K.(Y_0.\cos(Z_0) + X_0.sen(Z_0)) \\ Z_n &= 0 \end{aligned} \qquad (6)$$

Vemos que el valor de $Z_n$ en (6) debe de tender a cero y que el valor de $X_n$ e $Y_n$ quedan multiplicados por una constante K. Esta constante viene descrita por:

$$K = \prod_{n=0}^{i} \sqrt{1 + 2^{-2i}} \tag{7}$$

Por los motivos expuestos anteriormente, para obtener el resultado de la operación se debe multiplicar $X_n$ e $Y_n$ por un factor $A_n$, el cual es la inversa de esta constante K, que es el producto de los cosenos considerados para cada iteración:

$$A_n = \frac{1}{K} \tag{8}$$

Y al multiplicar a $X_n$ por el factor $A_n$ se obtiene la magnitud real de este vector, además, para efectos prácticos, se coloca $Y_0$= 0 con lo que el vector de giro empieza en el eje x.

### 1.2.1.2 MODO *VECTORING* PARA EL CORDIC CIRCULAR.

En este modo el vector también se rota, pero a diferencia del modo *rotation,* el parámetro de entrada es la posición del vector ($X_0$ y $Y_0$), ya que lo que se halla es el ángulo de dicho vector. Al igual que en el modo *rotation* el $d_i$ también queda definido por una relación, en este caso será de la siguiente manera.

$$\text{Si } Y_i \geq 0 \text{ entonces } d_i = -1$$

$$\text{caso contrario } d_i = 1 \tag{9}$$

Después de realizar n iteraciones el valor final de $X_n$, $Y_n$ y $Z_n$ queda definido de la siguiente forma:

$$
\begin{aligned}
X_n &= K.\sqrt{X_0^{\,2} + Y_0^{\,2}} \\
Y_n &= 0 \\
Z_n &= Z_0 + \tan^{-1}\left(\frac{Y_0}{X_0}\right)
\end{aligned}
\tag{10}
$$

Como se observa en la ecuación (10), para utilizar este modo es conveniente que el ángulo inicial ($Z_0$) sea cero para que el ángulo final acumulado en $Z_n$ sea el ángulo del vector de entrada. Se observa además que $Y_n$ debe tender a cero y que si se desea obtener la magnitud del vector se tiene que multiplicar $X_n$ por la inversa del factor *K* al igual que en el modo *rotation.*

### 1.2.2 SISTEMA COORDENADO HIPERBÓLICO

Las operaciones de seno, coseno, arco tangente hiperbólico y sus operaciones inversas se obtienen a partir de este sistema. Además se puede obtener el valor de la exponencial, el logaritmo natural y la raíz cuadrada. A continuación se muestra las ecuaciones para este sistema en la (*i*+1)-ésima rotación.

$$
\begin{aligned}
X_{i+1} &= X_i.\cosh(\alpha_i) + Y_i.senh(\alpha_i) \\
Y_{i+1} &= Y_i.\cosh(\alpha_i) + X_i.senh(\alpha_i) \\
Z_{i+1} &= Z_i - \alpha_i
\end{aligned}
\tag{11}
$$

Al igual que en el Sistema Coordenado Circular, se factoriza el *cosh(αi)* y se reemplaza la *tanh(αi)* por $2^{-i}$, quedando la ecuación (11) definida de la siguiente manera.

$$
\begin{aligned}
X_{i+1} &= X_i - d_i.Y_i.2^{-i} \\
Y_{i+1} &= Y_i + d_i.X_i.2^{-i} \\
Z_{i+1} &= Z_i - d_i.\tanh^{-1}(2^{-i})
\end{aligned}
\tag{12}
$$

Y el factor coseno hiperbólico presente en cada iteración queda definido en la siguiente ecuación.

$$
\cosh(\alpha_i) = \frac{1}{\sqrt{1-2^{-2i}}} \tag{13}
$$

Estas rotaciones se pueden apreciar mejor en la figura 1.3, donde se observa ese cambio de magnitud del vector al no tener en cuenta el factor del coseno hiperbólico al implementar el algoritmo con la ecuación 12.

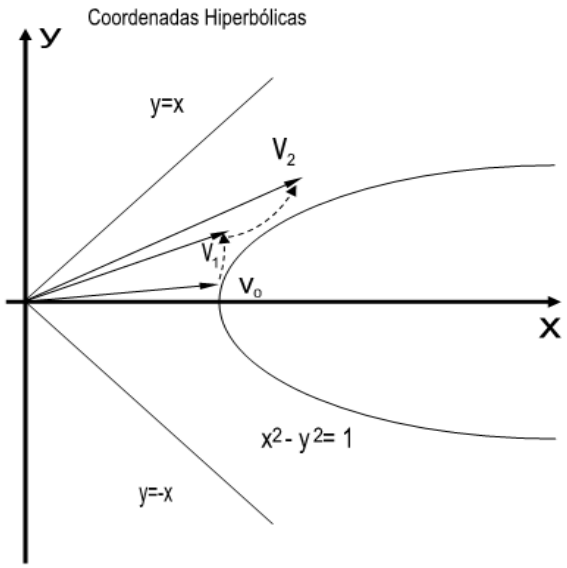



**Figura 1.3 Trayectoria de rotación en el Sistema Hiperbólico.**

Según (12) se debe tener en cuenta que la primera iteración debe darse con *i*=1, ya que con *i*=0, se hace que $tanh^{-1}(1)$ sea infinita. Además, para que el valor

obtenido con las iteraciones converjan, se necesita repetir ciertas iteraciones cuya forma se muestra a continuación:

$$i= 4, 13, 40, \ldots. k, 3k+1$$

En el caso del sentido de las rotaciones, dadas por *di*, se emplea, al igual que en el Sistema Coordenado Circular, la ecuación 5.

### 1.2.2.1 MODO *ROTATION* PARA EL CORDIC HIPERBÓLICO.

Al igual que en el sistema coordenado circular el vector de entrada también rotara con micro rotaciones angulares que estarán definidas en una tabla en la arquitectura.

Después de realizar las *n* iteraciones, la ecuación 12 queda definida de la siguiente forma:

$$\begin{aligned} X_n &= K.(X_0.\cosh(Z_0) + Y_0.senh(Z_0)) \\ Y_n &= K.(Y_0.\cosh(Z_0) + X_0.senh(Z_0)) \\ Z_n &= 0 \end{aligned} \tag{14}$$

Donde *K* es el factor acumulado al factorizar el coseno hiperbólico. Es por ello que al valor final de $X_n$ e $Y_n$ se le debe multiplicar por $A_n$,, la inversa del factor *K*, este valor se muestra en la ecuación 15. Además $Y_0$ debe ser cero para poder obtener el valor del coseno hiperbólico y seno hiperbólico para este modo. El pre-procesamiento es similar al sistema coordenado circular.

$$A_n = \prod_{n=0}^{i} \sqrt{1-2^{-i}} \tag{15}$$

### 1.2.2.2 MODO *VECTORING* PARA EL CORDIC HIPERBÓLICO.

Para este modo, tomando la ecuación 9 para definir el valor de $d_i$, y después de realizar las iteraciones con la ecuación 12 se obtienen las siguientes ecuaciones.

$$\begin{aligned} X_n &= A_n \cdot \sqrt{X_0^{\,2} + Y_0^{\,2}} \\ Y_n &= 0 \\ Z_n &= Z_0 + \tanh^{-1}\left(\frac{Y_0}{X_0}\right) \end{aligned} \tag{16}$$

En este modo también se toman las mismas consideraciones que en el sistema coordenado circular, ya que para hallar arco tangente hiperbólica se debe poner el valor de entrada $Z_0$=0.

### 1.2.3 SISTEMA COORDENADO LINEAL

Las operaciones multiplicación y división se obtienen empleando el Sistema Coordenado Lineal. La ecuación 17 muestra las rotaciones en este sistema. Se puede observar que la variable X se mantiene constante, por lo cual indica que las únicas variables que se modifican en las rotaciones son Y y Z.

$$\begin{aligned} X_{i+1} &= X_i \\ Y_{i+1} &= Y_i + X_i . \tan(\alpha_i) \\ Z_{i+1} &= Z_i - \alpha_i \end{aligned} \tag{17}$$

Reemplazando *tan(αi)* por una potencia de $2^{-i}$ se obtiene a partir de (17):

$$
\begin{aligned}
X_{i+1} &= X_i \\
Y_{i+1} &= Y_i + d_i * X_i * 2^{-i} \\
Z_{i+1} &= Z_i - d_i * (2^{-i})
\end{aligned}
\tag{18}
$$

En la figura 1.4 se puede apreciar cómo se realizan estas rotaciones, y en comparación con los otros dos sistemas, el vector conserva su valor en eje x (ecuación 18), por lo que no se necesita multiplicar a las señales iniciales por un factor adicional, porque la rotación en el sistema lineal implica un cambio en la magnitud del vector producto de la tangente.

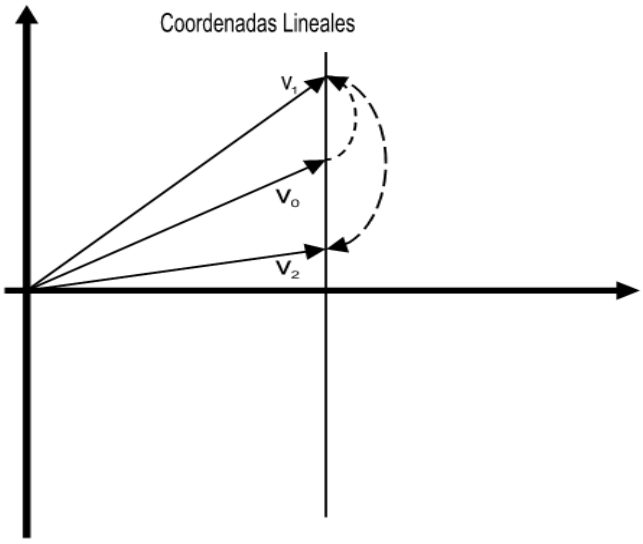



**Figura 1.4 Trayectoria de Rotación del Sistema Lineal**

Para realizar estas iteraciones *i* tomará los siguientes valores.

i=0,1,2,3 …N, donde N pertenece a los enteros positivos.

### 1.2.3.1 MODO *ROTATION* PARA EL CORDIC LINEAL.

Para este modo, los datos de entrada son los dos operandos de la multiplicación, los cuales estarán inicialmente en la variables X y Z (ecuación 19). Después de realizar n iteraciones en las variables X, Y y Z se obtienen:

$$\begin{aligned} Xn &= X_o \\ Yn &= Y_o + X_o.Z_o \\ Zn &= 0 \end{aligned} \tag{19}$$

Por lo que $Y_0$ debe ser cero al inicio, y con solo la respuesta de la multiplicación de los dos factores restantes se obtiene en la variable Y. Se debe de tener en cuenta que en este modo, la variable que se va comparando es Z y una tabla de valores debe ser construida para satisfacer los casos deseados.

### 1.2.3.2 MODO *VECTORING* PARA EL CORDIC LINEAL.

En este modo, los datos de entrada son los factores de la división los cuales ingresan por las variables X e Y. Después de *n* iteraciones el resultado de dicha división se obtiene en la variable Z (ecuación 20).

$$\begin{aligned} Xn &= X_o \\ Yn &= 0 \\ Zn &= Z_o + Y_o / X_o \end{aligned} \tag{20}$$

Como se explicó anteriormente, en este modo el algoritmo CORDIC, compara la variable Y, hasta obtener cero. Es por eso que el valor obtenido en la división se

obtendrá en la variable Z, pero para ello inicialmente la variable Z debe tener 0 como valor de entrada. Otra consideración a tener en cuenta es que es necesario comprobar el signo del resultado de la división para poder hacer el post procesamiento, o si los datos ingresados son de diferente signo, hacer que el dato que contiene el signo, que posteriormente definirá si el resultado es positivo o negativo, ingrese por la variable Y. Finalmente, debido a que en la división el resultado tendrá un rango muy amplio, con tendencia al infinito según sea el valor del dividendo, será necesario limitarlo de la manera más óptima.

# CAPÍTULO 2

## ANÁLISIS DEL FORMATO NUMÉRICO PARA CADA SISTEMA

El algoritmo CORDIC posee limitaciones debido a que es una aproximación de las funciones trascendentales con las que se desea trabajar, es por eso que este capítulo se subdivide en 2 tipos de análisis; en el primero definimos el rango básico de convergencia y se plantea un método para expandirlo, y en el segundo se analiza el formato numérico en cada caso según las recomendaciones obtenidas en el primero.

## 2.1 ANÁLISIS DEL RANGO DE CONVERGENCIA PARA CADA SISTEMA.

Dado que se implementan diferentes funciones para cada sistema, se hace un análisis por separado de cada uno de ellos.

### 2.1.1 ANÁLISIS DEL RANGO DE CONVERGENCIA PARA EL CORDIC CIRCULAR.

Es necesario analizar el rango de convergencia propio de la implementación del algoritmo en el sistema circular, ya que este nos indica los máximos valores que se puede obtener con esta forma básica de implementación. Para obtener el rango utilizamos la ecuación 21, en la que asumimos que el sentido de giro de todos los ángulos definidos por la expresión $atan(2^{-i})$ es para el mismo lado. En este caso, como se indicó en el capitulo anterior, las rotaciones empezarán desde $i$=0 hasta el número máximo de iteraciones definidas.

$$\theta_{mac} = \sum_{i=0}^{n} \tan^{-1}(2^{-i}) \qquad (21)$$

Considerando que *n* es 16 y reemplazando este valor en (21), el máximo ángulo acumulado hallado es:

$$\sum_{i=0}^{16} 2^{-i} \approx 1.74 rad. \tag{22}$$

Dado que en este rango se contemplan los valores de [-π /2  π /2], es decir que las rotaciones se ejecutan en el I y IV cuadrante, no es necesario expandir el rango, ya que las funciones trigonométricas son simétricas y con identidades trigonométricas se puede obtener el valor para cualquier ángulo con un pre procesamiento en los datos de entrada.

### 2.1.2 ANÁLISIS DEL RANGO DE CONVERGENCIA PARA EL CORDIC HIPERBÓLICO.

El algoritmo CORDIC para este sistema, descrito en las ecuaciones anteriores, es similar al CORDIC circular. Ambos reemplazan sus tangentes por desplazamientos de bits. Pero mientras que en el sistema Coordenado Circular las iteraciones comienzan de 0, en el hiperbólico comienzan de 1. Entonces el rango básico del valor de convergencia para n iteraciones viene dado por la ecuación 23.

$$\theta_{mac} = \tanh^{-1}(2^{-n}) + \sum_{i=1}^{n} \tanh^{-1}(2^{-i}) \tag{23}$$

Tomando como máximo número de iteraciones 16 y considerando que las ecuaciones 4 y 13 se repiten (ver capítulo 1), el máximo ángulo hiperbólico acumulado es:

$$|Zin| \leq \theta_{\max} \approx 1.1182 rad.$$

Como se puede observar con el valor máximo de θ, el valor de la tangente hiperbólica es 0.806941 (ver Figura 2.1). Este valor tiende 1 cuando el argumento tiende al infinito, y si se desea mayor precisión se debe expandir el rango, para de esta manera llegar a valores cercanos a 1.

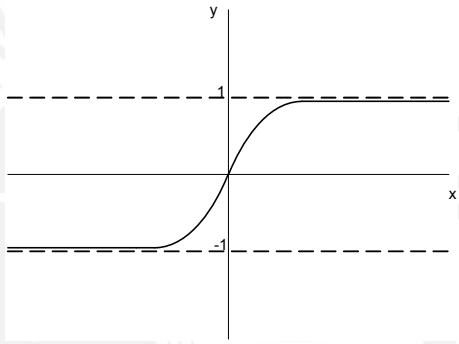



**Figura 2. 1 Tangente Hiperbólica**

Una estrategia para solucionar el problema del rango de convergencia es utilizar identidades matemáticas en el pre procesamiento de los valores de las entradas, sin embargo este va a ser diferente para cada función, lo cual dificulta la implementación de un hardware unificado para el CORDIC hiperbólico. Además del incremento en el uso de recursos lógicos y el tiempo de procesamiento.

Actualmente existen 2 métodos alternativos para expandir el rango de convergencia propuestos por Hu [3], el cual propone iteraciones adicionales en la implementación del algoritmo.

El primer método reemplaza la tangente hiperbólica por la expresión en (24) para las iteraciones adicionales, donde $i \leq 0$.

$$\tanh(\alpha_i) = 1 - 2^{i-2} \tag{24}$$

Al considerar iteraciones adicionales el $\theta_{max}$ se incrementa, con lo cual se obtiene un mayor rango de convergencia, esto se puede apreciar en la tabla 2.1. En (25) se muestra como hallar este $\theta_{max}$, donde –M es el máximo numero de iteraciones adicionales y N es el máximo número de iteraciones normales.

$$\theta_{mac} = \sum_{i=-M}^{0} \tanh^{-1}(1 - 2^{i-2}) + (\tanh^{-1}(2^{-n}) + \sum_{i=1}^{n} \tanh^{-1}(2^{-i})) \tag{25}$$

En el segundo método, la tangente hiperbólica se reemplaza por otra expresión (26) para los valores de $i \leq 0$.

$$\tanh(\alpha_i) = 1 - 2^{-2^{-i+1}} \tag{26}$$

Con este reemplazo, el rango de convergencia aumenta, con lo que para hallar esos valores utilizamos la ecuación 27.

$$\theta_{mac} = \sum_{i=-M}^{0} \tanh^{-1}(1 - 2^{-2^{-i+1}}) + (\tanh^{-1}(2^{-n}) + \sum_{i=1}^{n} \tanh^{-1}(2^{-i})) \tag{27}$$

En la tabla 2.1, se muestra $\theta_{max}$ hallado para cada método según el número de iteraciones adicionales.

| -i | $\theta_{max}$ – Ec.25 | $\theta_{max}$ – Ec.27 |
|---|---|---|
| 0 | 2.09113 | 2.09113 |
| 1 | 3.44515 | 3.80812 |
| 2 | 5.16215 | 6.92631 |
| 3 | 7.23371 | 12.81805 |
| 4 | 9.6581 | 24.25498 |
| 5 | 12.42644 | 46.78227 |
| 6 | 15.54462 | 91.49026 |
| 7 | 19.00987 | 180.55967 |
| 8 | 22.82194 | 358.35192 |
| 9 | 26.9807 | 713.58985 |
| 10 | 31.48609 | 1423.71914 |

**Tabla 2. 1 $\theta_{max}$ obtenido con el nuevo rango de convergencia**

El método empleado en este trabajo es el primero, porque implica menos recursos lógicos que el segundo; y además para los valores que se van a implementar en la tabla de datos para el tamaño de buses propuestos, no aporta más beneficios que el primer método. Finalmente la ecuación para cada rotación para la parte de expansión queda definida de esta manera:

$$\begin{aligned} X_{i+1} &= X_i - d_i.Y_i.(1-2^{i-2}) \\ Y_{i+1} &= Y_i + d_i.X_i.(1-2^{i-2}) \\ Z_{i+1} &= Z_i - d_i * \tanh^{-1}(1-2^{i-2}) \end{aligned} \qquad (28)$$

### 2.1.3 ANÁLISIS DEL RANGO DE CONVERGENCIA PARA EL CORDIC LINEAL.

El algoritmo para este caso presenta un limitado rango de convergencia, debido a que con este sistema se implementa las funciones multiplicación y división; y

ambas tienden al infinito. Por tal motivo, es necesario definir un límite en el rango para ambas funciones. En el análisis del rango básico de convergencia se utiliza la siguiente ecuación.

$$\theta_{mac} = \sum_{i=0}^{n} (2^{-i}) \tag{28}$$

Asumiendo que el valor máximo de iteraciones es 16, el $\theta_{max}$ en (28) es aproximadamente 2; con este valor solo podemos representar, en el caso de la división, resultados que se encuentran en el rango <-2  2>, es por ello que en este caso es necesario la expansión. En el caso de la multiplicación, dado que se van a realizar las iteraciones hasta que el valor en la variable Z sea cero, y Z es un operando de la multiplicación. La operación está restringida para multiplicaciones de un número por valores menores que 2, por lo que se requiere una expansión.

Para expandir el rango de convergencia para el CORDIC en el sistema lineal es necesario añadir algunas iteraciones. El método propuesto por Hu [3], añade a los índices de las iteraciones los valores negativos, al igual que en el CORDIC hiperbólico, de tal manera que i toma los valores –M, -M+1…0,1,…N.  En este caso, la expansión tiene la misma estructura que en las iteraciones normales, pero el cambio radica en que el desplazamiento va a ser para la izquierda, ya que las iteraciones adicionales son potencias de 2 positivas.

## 2.2 ANÁLISIS DEL FORMATO NUMÉRICO PARA EL SISTEMA CIRCULAR.

En este análisis, es necesario considerar que el tamaño del bus de datos tanto para la entrada como para la salida son iguales. Además se toma en consideración para el análisis los 4 tipos de buses propuestos, los cuales son 12, 16, 24 y 32 bits.

Debido a que el resultado es diferente para cada modo de procesamiento, es necesario analizar por separado cada uno de ellos. También es importante tener en cuenta que como X e Y realizan operaciones entre ellas, se considera que ambas variables tengan el mismo formato.

### 2.2.1 MODO ROTATION PARA EL CORDIC CIRCULAR.

En este modo las ecuaciones después de 'n' iteraciones se muestran en (6), en donde para obtener las funciones trigonométricas del coseno y seno se hace que la entrada de $Y_0 = 0$, $X_0 = An$ y $Z_0=\theta$, donde θ es el ángulo a rotar. Además que el ángulo de entrada ($Z_0$) tiene valores entre $-\pi/2$ y $\pi/2$, y en las salidas el resultado que se obtiene en $X_n$ e $Y_n$ será el valor del coseno y seno del ángulo de entrada, por lo cual estos valores están en el rango de [-1 1]. Tomando estas consideraciones, se realiza el siguiente análisis para hallar el formato numérico necesario para cada bus de datos.

#### 2.2.1.1 CASO 12 BITS

Por lo descrito anteriormente se determina que es necesario por lo menos 2 bits enteros (incluyendo el bit de signo) para representar los valores tanto de las entradas ($Z_0$) como las salidas ($X_0$ e $Y_0$). Por otro lado encontramos que para la entrada en $X_0 = A_n$ obtendremos el valor de 0.6072530, el cual no afecta al formato establecido. Finalmente el formato para cada variable es:

$$X = [12\ 10] \quad Y = [12\ 10] \quad Z = [12\ 10]$$

Para este formato en Z encontramos que el número de iteraciones necesarias es 11 (ver tabla 2.2).

| i | Valor | i | Valor |
|---|---|---|---|
| 0 | 324 | 6 | 010 |
| 1 | 1DB | 7 | 008 |
| 2 | 0FB | 8 | 004 |
| 3 | 07F | 9 | 002 |
| 4 | 040 | 10 | 001 |
| 5 | 020 | | |

**Tabla 2.2 Valores de la tabla de datos para 12 bits.**

Con los valores obtenidos se halla que θmax = 1.7421875, el cual es mayor que π/2 (1.5708rad.) por lo que se comprueba que para esos valores se cubre el ángulo requerido. Este algoritmo se ha implementado en MATLAB para el formato establecido y se comprobó que ningún valor intermedio, en la peor condición (*i*=0), excede lo establecido [12 10].

#### 2.2.1.2 CASO 16 BITS

Al igual que para 12 bits, se realizó el análisis del formato numérico posible para las variables y se halló que 2 bits de parte entera son necesarios para representar los valores de los resultados como las variables de entrada. Además se encontró que son necesarias 15 iteraciones (ver tabla 2.3) en donde el $\theta_{max}$ obtenido es 1.74322509.

Con este valor se cubre todo el rango de ángulos pre establecidos como entrada. Finalmente los formatos para las 3 variables son:

X = [16 14]  Y = [16 14]  Z = [16 14]

| i | Valor | i | Valor |
|---|---|---|---|
| 0 | 3244 | 8 | 0040 |
| 1 | 1DAC | 9 | 0020 |
| 2 | 0FAE | 10 | 0010 |
| 3 | 07F5 | 11 | 0008 |
| 4 | 03FF | 12 | 0004 |
| 5 | 0200 | 13 | 0002 |
| 6 | 0100 | 14 | 0001 |
| 7 | 0080 | | |

**Tabla 2.3 Valores de la tabla de datos para 16 bits.**

Se comprueba en MATLAB que ningún valor intermedio en X, Y y Z sobrepasa el formato numérico establecido [16 14].

#### 2.2.1.3 CASO 24 BITS

Para este caso, también se observa que serán necesarios 2 bits enteros, debido a las condiciones iniciales, ya que tanto el ángulo como las funciones se encuentran dentro del rango establecido para los posibles valores en este formato. Por esta razón se halló que para el caso de 24 bits el número de iteraciones posibles es 23 (ver tabla 2.4); pero al hacer un balance entre precisión y recursos de hardware solo se utilizan 16 iteraciones como máximo en la arquitectura interna. Con este número de iteraciones el θmax obtenido es 1.743256092. Finalmente los formatos establecidos para las 3 variables son:

X = [24 22]  Y = [24 22]  Z = [24 22]

En la tabla 2.4 se muestran los valores que toma la tabla de datos que se procesa con la variable Z.

| i | Valor | i | Valor | i | Valor |
|---|---|---|---|---|---|
| 0 | 3243F7 | 8 | 004000 | 16 | 000040 |
| 1 | 1DAC67 | 9 | 002000 | 17 | 000020 |
| 2 | 0FADBB | 10 | 001000 | 18 | 000010 |
| 3 | 07F56F | 11 | 000800 | 19 | 000008 |
| 4 | 03FEAB | 12 | 000400 | 20 | 000004 |
| 5 | 01FFD5 | 13 | 000200 | 21 | 000002 |
| 6 | 00FFFB | 14 | 000100 | 22 | 000001 |
| 7 | 007FFF | 15 | 000080 | | |

**Tabla 2.4 Valores de la tabla de datos para 24 bits.**

Se comprueba en MATLAB que para el formato [24 22] no se encontraron valores intermedios en las variables que no cumplan lo establecido.

#### 2.2.1.4 CASO 32 BITS

También para este caso se utilizarán 2 bits para la parte entera, con este formato se halló que el número de iteraciones posibles es 31 (ver tabla 2.5). Al igual que para 24 bits solo se utilizan las primeras 16 iteraciones. Finalmente los formatos para cada variable quedan definidos de la siguiente manera.

X = [32 30]  Y = [32 30]  Z = [32 30]

| i | Valor | i | Valor | i | Valor | i | Valor |
|---|---|---|---|---|---|---|---|
| 0 | 3243F6A9 | 8 | 003FFFEB | 16 | 00004000 | 24 | 00000040 |
| 1 | 1DAC6705 | 9 | 001FFFFD | 17 | 00002000 | 25 | 00000020 |
| 2 | 0FADBAFD | 10 | 00100000 | 18 | 00001000 | 26 | 00000010 |
| 3 | 07F56EA7 | 11 | 00080000 | 19 | 00000800 | 27 | 00000008 |
| 4 | 03FEAB77 | 12 | 00040000 | 20 | 00000400 | 28 | 00000004 |
| 5 | 01FFD55C | 13 | 00020000 | 21 | 00000200 | 29 | 00000002 |
| 6 | 00FFFAAB | 14 | 00010000 | 22 | 00000100 | 30 | 00000001 |
| 7 | 007FFF55 | 15 | 00008000 | 23 | 00000080 | | |

**Tabla 2.5 Valores de la tabla de datos para 32 bits.**

Se comprueba en MATLAB que para el formato [32 30] no se encontraron valores intermedios en las variables que no cumplan lo establecido.

### 2.2.2 MODO VECTORING PARA EL CORDIC CIRCULAR.

Para este modo, es necesario analizar la función arco tangente que se obtiene en (10). Debido a la naturaleza de la tangente, cuyo resultado abarca los valores en el rango de los números reales, es necesario que el formato para cada bus de

datos cubra el mayor rango posible para el ángulo de salida. Al igual que en el modo rotation, hay cierto factores que se tienen que considerar para determinar el formato a elegir. El primero es que $X_0$=1, para que el argumento de la arco tangente sea $Y_0$. Además se tiene que hacer que $Z_0$ = 0 para que en $Z_n$ se obtenga el valor del ángulo hallado. En los 4 tipos de bus de datos propuestos, la variable Z tendrá 2 bits de parte entera, ya que el rango del ángulo es de [-π/2  π /2]. De esta manera se analizó el formato para la variable Y en cada caso.

#### 2.2.2.1 CASO 12 BITS

Para este caso se consideró un formato inicial de 3 bits de parte entera, es decir [12 9] para la variable Y, con estos valores se obtiene el siguiente rango de valores en Y de [-4  3.99], aplicando la función arco tangente a estos valores se obtiene que el rango de salida para los ángulos es de [-0.417π  0.417π]. Debido a que el formato es el mismo para Z, por que los ángulos se encuentran en el mismo rango ([-π /2  π /2]), se utiliza la misma tabla de datos (ver tabla 2.2) para las iteraciones.

Se comparó con el programa MATLAB si existía algún valor intermedio que no cumpla con el formato de Y; en este análisis se halló que para el máximo valor negativo del rango de Y = - 4 se incrementa en una unidad su valor al sumarse en la variable X=1-(-4) en la iteración 0, por lo que se requiere de 1 bit adicional tanto para X como para Y en la arquitectura interna, ya que estas variables deben de tener el mismo formato, porque realizan operaciones conjuntamente, quedando como formato para las variables X e Y en la arquitectura interna [13 9].

Finalmente, los formatos definidos para las 3 variables son:

X = [12 9]  Y = [12 9]  Z = [12 10]

#### 2.2.2.2 CASO 16 BITS

Para este caso, se incrementa 1 bit más en la parte entera por que al tener más bits podemos incrementar el rango del argumento sin reducir los bits para la parte decimal. Entonces el formato para 16 bits para las variables X e Y es [16 12]. Con este formato utilizado se puede alcanzar valores en el siguiente rango: [-8  7.99]. Y el ángulo obtenido se encuentra aproximadamente en el rango de [-0.455π 0.455π]. Al igual que para 12 bits también se analizó en MATLAB si algún valor intermedio no cumplía con el formato, y se halló que en la iteración 0 se incrementaba X en 1 unidad para el máximo argumento negativo por lo que se requiere aumentar 1 bit más a la parte entera en la arquitectura interna. Finalmente, los formatos definidos para las 3 variables son:

X = [16 12]  Y = [16 12]  Z = [16 14]

#### 2.2.2.3 CASO 24 BITS

Para este caso, se incrementa 1 bit más en la parte entera, en comparación con el caso anterior; ya que al aumentar el número de bits en el bus de datos, debemos aumentar también el rango del argumento; por que de esta manera se obtiene un mayor rango de valores sin perder precisión en los resultados. Es por eso que X e Y tiene como formato [24 19], con este formato se pueden alcanzar valores en el

rango de [-16 15.99], con los cuales se obtiene valores para los ángulos en un rango de [-0.48π 0.48π].

Por lo que se explicó en los casos anteriores se le aumentó 1 bit más en la parte entera en la arquitectura interna.

#### 2.2.2.4 CASO 32 BITS

En este caso también se incrementará 1 bit más en la parte entera, de tal manera que se puede expandir el rango de convergencia para esta función. De esta manera con el formato [32 26] se obtiene que el argumento abarca valores en el rango de [-32 31.99], con estos valores se obtienen ángulos entre los rangos [-0.489π 0.489π]; como se puede observar los valores son muy cercanos a [-0.5π 0.5π].

Analizando en el MATLAB, también se observa que es necesario incrementar 1 bit más en la arquitectura interna para evitar el desbordamiento al realizar las operaciones entre las variables X e Y en cada iteración.

## 2.3 ANÁLISIS DEL FORMATO PARA EL SISTEMA HIPERBÓLICO.

Para este análisis, también se consideró que el tamaño del bus de datos tanto para entrada como para la salida sean iguales y que se analiza el formato numérico para cada uno de los 4 tipos de bus de datos independientemente.

### 2.3.1 MODO ROTATION PARA EL CORDIC HIPERBÓLICO.

Las funciones obtenidas en este modo son seno hiperbólico y coseno hiperbólico, por lo cual el formato de Z es fijado al inicio; mientras que con el valor máximo de Z se halla el formato para las otras variables. Además se considera que $X_0 = A_n$ (para el caso del sistema hiperbólico), por lo que se halla el valor An en cada caso según el número de iteraciones adicionales y normales que se implementan. De esta manera se puede comprobar si es que no existe ningún desborde y si se puede representar el valor de X, Y y Z con el formato establecido.

#### 2.3.1.1 CASO 12 BITS

En este caso, el formato escogido para Z es [12 10]. Observando que Z se encuentra en el rango de valores de [-2 1.9990234375]. Para definir el número de iteraciones adicionales a utilizar, es necesario tener en cuenta el máximo valor que puede tomar Z, ya que el algoritmo busca que Z sea cero. Para 12 bits, Z tomará como valor máximo -2, por lo que según la tabla 2.1, es necesario una iteración adicional con lo que se cubre este valor. Y en el caso del formato de X e Y se hallan los valores que toman las funciones implementadas seno y coseno hiperbólico para Z= -2.

$$\text{Sinh}(-2) = -3.626860$$

$$\text{Cosh}(-2) = 3.76219$$

Tomando en consideración el valor del coseno hiperbólico, que es el de mayor valor, se observa que son necesarios 3 bits de parte entera para cubrir todo el

rango de valores posibles en la entrada. Además se comprueba que no hay desborde en esas variables hallando $A_n$ = 0.5477760199 y $X_0$ = 1.82556366 para el total del número de iteraciones halladas con el formato de Z.

Al igual que para el Sistema Circular se ejecutó el algoritmo en MATLAB y no se encontró ningún desborde en los valores intermedios. Finalmente los formatos para las variables son:

X = [12 9] Y = [12 9] Z = [12 10]

A continuación se muestra la tabla de iteraciones que se usarán en este modo para este bus de datos.

| I | Valor | i | Valor |
|---|---|---|---|
| 0 | 3E4 | 6 | 010 |
| 1 | 232 | 7 | 008 |
| 2 | 106 | 8 | 004 |
| 3 | 081 | 9 | 002 |
| 4 | 040 | 10 | 001 |
| 5 | 020 | | |

**Tabla 2.6 Valores de la tabla de datos para 12 bits.**

### 2.3.1.2 CASO 16 BITS

El formato escogido para Z es [16 13] en este caso. Con este formato establecido se observó que Z se encuentra en el rango de valores de [-4 3.99987792968]. Para esos valores se halla que para el valor máximo de Z= - 4, el número de iteraciones adicionales es 3 para cubrir ese valor (ver tabla 2.1); además con el máximo valor de Z, X e Y quedan definidas con los siguientes valores.

Sinh(-4) = -27.289917197

Cosh(-4) = 27.3082328361

Tomando en consideración el valor del resultado para el coseno hiperbólico, se halla que son necesarios 6 bits de parte entera para cubrir todo el rango de valores posibles en la entrada, por lo tanto X e Y = [16 10]. Además se comprueba que no hay desborde para el valor inicial de X, hallando que $A_n = 9.2282521x10^{-2}$ y $X_0 = 10.83628128$ para el total del número de iteraciones establecidas para ese formato. Al igual que para los casos anteriores se probó el algoritmo en MATLAB y no se encontró ningún desborde en los valores intermedios. Finalmente los formatos para las variables son:

X = [16 10]  Y = [16 10]  Z = [16 13]

A continuación se muestra la tabla de iteraciones que se usarán en la arquitectura de este modo para este tipo de bus de dato.

| I | Valor | i | Valor |
|---|---|---|---|
| -2 | 36F2 | 6 | 0080 |
| -1 | 2B54 | 7 | 0040 |
| 0 | 1F22 | 8 | 0020 |
| 1 | 1194 | 9 | 0010 |
| 2 | 082C | 10 | 0008 |
| 3 | 0405 | 11 | 0004 |
| 4 | 0201 | 12 | 0002 |
| 5 | 0100 | 13 | 0001 |

**Tabla 2.7 Valores de la tabla de datos para 16bits.**

Se analizó para el formato de Z =[16 12] pero se pierde mucha precisión en los bits de la parte decimal, por lo que se optó por el anterior.

### 2.3.1.3 CASO 24 BITS

Para 24 bits el formato escogido para Z es [24 20]. En este caso se observa que Z se encuentra en el rango de valores de [-8 7.999999046325]. Considerando que el valor máximo a representar es – 8, se observó que son necesarias 5 iteraciones adicionales (ver tabla 2.1). Para ese mismo valor de Z se halla que X e Y quedan definidas con los siguientes valores.

$$\text{Sinh}(-8) = -1490.4788257$$

$$\text{Cosh}(-8) = 1490.47916$$

Tomando en consideración el valor del coseno hiperbólico se halla que son necesarios 12 bits de parte entera para cubrir todo el rango de valores posibles en la entrada, por lo tanto X e Y =[24 12]. Además se comprueba a que no hay desborde en esas variables hallando $A_n=4.0305251x10^{-3}$ y $X_0=248.1066$ para el total del número de iteraciones que se muestran en la tabla 2.8. También para este caso se probó el algoritmo en MATLAB y se encontró que para la iteración 0 el valor de X e Y sobrepasan el formato establecido (X=3081.0854 e Y=3081.085173) por lo que se requiere aumentar a X e Y en un bit más en la parte entera. Por lo tanto el formato de X e Y para la arquitectura interna es [25 12]. Finalmente los formatos para las variables son:

$$X = [24\ 12] \quad Y = [24\ 12] \quad Z = [24\ 20]$$

Se analizó para el formato de Z =[24 19] pero se necesitan todo los bits como parte entera siendo ineficiente para este diseño. A continuación se muestra la tabla de iteraciones que se implementa en la arquitectura.

| I | Valor | i | Valor |
|---|---|---|---|
| -4 | 26C0E5 | 7 | 002000 |
| -3 | 212524 | 8 | 001000 |
| -2 | 1B78CE | 9 | 000800 |
| -1 | 15AA16 | 10 | 000400 |
| 0 | 0F9139 | 11 | 000200 |
| 1 | 08C9F5 | 12 | 000100 |
| 2 | 04162C | 13 | 000080 |
| 3 | 0202B1 | 14 | 000040 |
| 4 | 010056 | 15 | 000020 |
| 5 | 00800B | 16 | 000010 |
| 6 | 004000 | | |

**Tabla 2.8 Valores de la tabla de datos para 24bits.**

### 2.3.1.4 CASO 32 BITS

Para 32 bits el formato escogido para Z es [32 27]. Con este formato establecido se observó que Z se encuentra en el rango de valores de [-16 15.999999992]. Tomando en cuenta que el máximo valor de Z es -16, la tabla 2.1 nos indica que son necesarias 8 iteraciones adicionales. Además para ese mismo valor se encuentra que X e Y toman los siguientes valores.

$$\text{Sinh}(-16) = -4.44305526025399 \times 10^{6}$$

$$\text{Cosh}(-16) = 4.44305526025388 \times 10^{6}$$

Considerando el valor del coseno hiperbólico, como en los demás casos, se halla que son necesarios 24 bits de parte entera para cubrir todo el rango de valores posibles en la entrada, por lo tanto X e Y = [32 8]. Además se comprobó que con ese formato se puede representar el valor inicial $X_0$, hallando que $A_n$=2.7737x10$^{-6}$ y $X_0$ = 3.625287519x10$^5$ para el total del número de iteraciones. Al ejecutar el algoritmo en MATLAB se encontró que para la iteración 0 el valor de X e Y sobrepasan el formato establecido (X = 20.32x10$^6$ e Y = 20.32x10$^6$) por lo que se debe aumentar a X e Y en 2 bits mas en la parte entera para corregir esto. Este cambio se hace internamente, por lo tanto el formato de X e Y para la arquitectura interna es [34 8]. Finalmente los formatos para las variables son:

X = [32 8]  Y = [32 8]  Z = [32 27]

A continuación se muestra la tabla de iteraciones que se usaran en este modo para 32 bits.

| i | Valor | i | Valor |
|---|---|---|---|
| -7 | 1BB8BAC7 | 5 | 00400556 |
| -6 | 18F2085B | 6 | 002000AB |
| -5 | 162A40FE | 7 | 00100015 |
| -4 | 13607294 | 8 | 00080003 |
| -3 | 109291E9 | 9 | 00040000 |
| -2 | 0DBC6724 | 10 | 00020000 |
| -1 | 0AD50B1D | 11 | 00010000 |
| 0 | 07C89CAC | 12 | 00008000 |
| 1 | 0464FA9F | 13 | 00004000 |
| 2 | 020B15DF | 14 | 00002000 |
| 3 | 01015892 | 15 | 00001000 |
| 4 | 00802AC4 | 16 | 00000800 |

**Tabla 2.9 Valores de la tabla de datos para 32bits.**

Se analizó para el formato de Z =[32 26] pero son necesarios 47 bits de parte entera y esto es imposible. No se opto por hacer [32 28] , ya que este formato no presenta ganancia en rango respecto a la de 24 bits.

### 2.3.2 MODO VECTORING PARA EL CORDIC HIPERBÓLICO.

Debido a que la función obtenida con este método es la arco tangente hiperbólico, cuyo argumento varia en el rango de <-1, 1> y dado que el valor del argumento depende tanto de la variable $X_0$ e $Y_0$ (ver ecuación 15), los valores que tomaran esas variables serán: $X_0$ = 1 e $Y_0$ = <-1, 1>, por lo que es necesario aprovechar la máxima cantidad de bits de la parte fraccionaria, y que X e Y tengan el mismo formato ya que estos realizan operaciones conjuntamente; es por ello que la parte entera para cada bus de datos en este modo será de 2 bits.

Para hallar el formato de Z será necesario calcular el valor de la arco tangente hiperbólica para cada caso y encontrar el número de iteraciones adicionales para hacer corresponder su valor en la variable Z.

#### 2.3.2.1 CASO 12 BITS

El formato para X e Y es de [12 10] en ambos casos, con lo que para el formato de Z es necesario considerar lo siguiente: El valor máximo de Y que se obtiene con este formato (10 bits de parte fraccionaria) es + 0.999023475, la arco tangente hiperbólica de este valor será θ = 3.812065. Debido a que con este valor no se puede alcanzar con las iteraciones normales como en el caso del modo rotation se hará uso de la expansión descrita en el capítulo 1; de la tabla 2.1 se

encuentra que el número de iteraciones necesarias es 3; con estas iteraciones el $\theta_{max}$= 5.162. Para representar el ángulo para $Y_{0max}$ será necesario 3 bits de parte entera. Sin embargo, analizando los valores obtenidos en todas las iteraciones en MATLAB, hallamos que para una iteración intermedia el valor obtenido para Z es 4.04, por lo que será necesario extender un bit más en la parte entera de Z. Este cambio se hará en la implementación de la arquitectura interna. Entonces, el formato para Z es de [12 9] y en el bus interno es de [13 9]. Finalmente los formatos para las 3 variables son:

X = [12 10] Y = [12 10] Z = [12 9]

Con este formato la tabla de iteraciones para Z se muestra en la siguiente tabla.

| i | Valor | I | Valor |
|---|---|---|---|
| -2 | 36F | 4 | 020 |
| -1 | 2B5 | 5 | 010 |
| 0 | 1F2 | 6 | 008 |
| 1 | 119 | 7 | 004 |
| 2 | 083 | 8 | 002 |
| 3 | 040 | 9 | 001 |

**Tabla 2.10 Valores de la tabla de datos para 12bits.**

### 2.3.2.2 CASO 16 BITS

El formato para la entrada X e Y es [16 14], y para hallar el formato de Z es necesario considerar que el valor máximo que se obtiene con este formato (14 bits de parte fraccionaria) es + 0.99993896484375, la arco tangente hiperbólica para ese valor es θ = 5.1985885952. Para este caso el número de iteraciones necesarias adicionales es 4 (ver tabla 2.1); ya que con este valor se obtiene un

$\theta_{max}$= 7.23 aproximadamente. Con estos valores se calculó que son necesarios 4 bits para cubrir ambos casos. Por lo que el formato de Z es: [16 12]. Finalmente los formatos para las 3 variables son:

X = [16 14]  Y = [16 14]  Z = [16 12]

En este caso, no se presenta desborde después de analizar los resultados con el programa MATLAB.

| i | Valor | I | Valor |
|---|---|---|---|
| -3 | 2125 | 5 | 0080 |
| -2 | 1B79 | 6 | 0040 |
| -1 | 15AA | 7 | 0020 |
| 0 | 0F91 | 8 | 0010 |
| 1 | 08CA | 9 | 0008 |
| 2 | 0416 | 10 | 0004 |
| 3 | 0203 | 11 | 0002 |
| 4 | 0100 | 12 | 0001 |

**Tabla 2.11 Valores para la tabla de datos de 16bits.**

### 2.3.2.3 CASO 24 BITS

Para este caso, el formato para la entrada X e Y es [24 22] y para hallar el formato de Z es necesario considerar que el valor máximo que se obtiene con este formato (22 bits de parte fraccionaria) es + 0.9999997615824209, la arco tangente hiperbólica para ese valor es θ =  7.9711925. De la tabla 2.1, se halló que el número de iteraciones adicionales necesarias es 5. Con estas iteraciones se obtiene que el $\theta_{max}$= 8.5346, además se observó que solo se requieren 4 bits de parte entera pero con MATLAB, se halló que para una iteración intermedia el valor que se acumula requiere 5 bits de la parte entera. Por lo tanto la arquitectura de la

parte interna tiene un formato para Z = [24 19]. Finalmente los formatos para las 3 variables son:

$$X = [24\ 22] \quad Y = [24\ 22] \quad Z = [24\ 20]$$

Con este formato la tabla de iteraciones para Z se muestra en la siguiente tabla. Para este caso, el número de iteraciones normales posibles es 20, pero dado que el número máximo establecido de iteraciones normales es 16, solo se considera hasta este valor.

| i | Valor | I | Valor |
|---|---|---|---|
| -4 | 26C0E5 | 7 | 002000 |
| -3 | 212524 | 8 | 001000 |
| -2 | 1B78CE | 9 | 000800 |
| -1 | 15AA16 | 10 | 000400 |
| 0 | 0F9139 | 11 | 000200 |
| 1 | 08C9F5 | 12 | 000100 |
| 2 | 04162C | 13 | 000080 |
| 3 | 0202B1 | 14 | 000040 |
| 4 | 010056 | 15 | 000020 |
| 5 | 00800B | 16 | 000010 |
| 6 | 004000 | | |

**Tabla 2.12 Valores para la tabla de datos de 24bits.**

### 2.3.2.4 CASO 32 BITS

El formato para la entrada X e Y es [32 30] y para hallar el formato de Z se consideró que el valor máximo que se obtiene con este formato (30 bits de parte fraccionaria) es + 0.999999999068677, el arco tangente hiperbólico para ese valor es θ = 10.743781. De la tabla 2.1, se halló que el número de iteraciones

adicionales necesarias es 6. Con estas iteraciones se obtiene que el $\theta_{max}$= 12.36 aproximadamente. Con estos valores se observó que solo se necesitan 5 bits de parte entera por lo que el formato de Z será [32 27]. Finalmente los formatos para las 3 variables son:

X = [32 30]  Y = [32 30]  Z = [32 27]

Con este formato la tabla de iteraciones para Z se muestra en la tabla 2.13. Para este caso, el número de iteraciones normales posibles es 27; pero al igual que para 24 bits solo se considera hasta 16.

| i | Valor | I | Valor |
|---|---|---|---|
| -5 | 162A40FE | 6 | 002000AB |
| -4 | 13607294 | 7 | 00100015 |
| -3 | 109291E9 | 8 | 00080003 |
| -2 | 0DBC6724 | 9 | 00040000 |
| -1 | 0AD50B1D | 10 | 00020000 |
| 0 | 07C89CAC | 11 | 00010000 |
| 1 | 0464FA9F | 12 | 00008000 |
| 2 | 020B15DF | 13 | 00004000 |
| 3 | 01015892 | 14 | 00002000 |
| 4 | 00802AC4 | 15 | 00001000 |
| 5 | 00400556 | 16 | 00000800 |

**Tabla 2.13 Valores para la tabla de datos de 32bits.**

## 2.4 ANÁLISIS DEL FORMATO NÚMERICO PARA EL SISTEMA LINEAL.

Para este análisis, es necesario considerar que debido a la limitación del algoritmo solo se puede obtener multiplicaciones y divisiones en un rango muy

pequeño. Al igual que en el sistema hiperbólico es necesario una expansión como se explicó anteriormente, la cual nos permite expandir el rango a medida que aumentemos el tamaño de los buses de datos. Tanto para el caso de la multiplicación y la división se escogió un óptimo formato. Además de considerar que el signo de la división dependerá del valor que tenga la variable Y para el caso de la división. A continuación se analizará para cada bus de datos por separado.

### 2.4.1 MODO ROTATION PARA EL CORDIC LINEAL.

Como se observa en la ecuación 19, en este modo es necesario que la variable $Y_0$ sea 0; de esta manera se podrá obtener el valor de la multiplicación entre $X_0$ y $Z_0$ en la variable Y. Al igual que en los sistemas anteriores se analizará el formato con cada tipo de bus. Es necesario que Z tenga el mismo formato que las variables X e Y, ya que el valor que se desea obtener, es la multiplicación de las variables iniciales X y Z. A continuación se analiza para cada tamaño de bus de datos.

#### 2.4.1.1 CASO 12 BITS

En este caso, se parte de un formato inicial [12 10] para las variables X, Y y Z. Al multiplicar 2 señales con el mismo formato se obtiene una respuesta con un formato de [24 20], pero al inicio se especificó que como salida debemos tener una señal de igual cantidad de bits que la señal de entrada; el formato de salida de la señal será de [12 8], ya que se tiene que conservar la parte entera. Es por

eso, que para obtener esa repuesta se necesita que a los datos de entrada se le asigne 2 bits más en la parte entera en la arquitectura interna y no perder precisión en la parte fraccionaria.

Con el formato de entrada definido se obtiene valores en el rango [-2 1.99], el máximo número obtenido para la multiplicación será -4, por lo que se necesita de 2 iteraciones adicionales (ver tabla 2.14). Las iteraciones normales quedan definidas por el mínimo valor que se pueda obtener con el formato interno [14 10]. Por lo cual la tabla de datos para las iteraciones necesarias en este caso es:

| i | Valor | i | Valor |
|---|---|---|---|
| -2 | 1000 | 5 | 0020 |
| -1 | 0800 | 6 | 0010 |
| 0 | 0400 | 7 | 0008 |
| 1 | 0200 | 8 | 0004 |
| 2 | 0100 | 9 | 0002 |
| 3 | 0080 | 10 | 0001 |
| 4 | 0040 | | |

**Tabla 3.13 Valores para la tabla de datos de 32bits.**

### 2.4.1.2 CASO 16 BITS

Para el caso de 16 bits, se amplia el rango de entrada utilizando el formato de las variables [16 13]. De esta manera, al multiplicar las variables X y Z con ese formato se obtiene una respuesta de [32 26], pero ya que solo se utilizan 13 bits, la respuesta de salida en la variables X, Y y Z tiene el formato [16 10]. Por lo que se observó que es necesario agregar 3 bits más de parte entera en la arquitectura interna para procesar los datos correctamente, quedando el formato [19 13] en la arquitectura interna para esas variables.

Para determinar el número de iteraciones adicionales, es necesario saber que el máximo número obtenido con ese formato [-4  3.99] en la multiplicación será -16. Entonces se necesita 4 iteraciones adicionales (ver tabla 2.15) y dado que el formato es [19 13], el número máximo de iteraciones normales es 14.

Finalmente la tabla de datos para 16 bits queda definida con los siguientes valores para las iteraciones.

| i | Valor | i | Valor |
|---|---|---|---|
| -4 | 20000 | 5 | 00100 |
| -3 | 10000 | 6 | 00080 |
| -2 | 08000 | 7 | 00040 |
| -1 | 04000 | 8 | 00020 |
| 0 | 02000 | 9 | 00010 |
| 1 | 01000 | 10 | 00008 |
| 2 | 00800 | 11 | 00004 |
| 3 | 00400 | 12 | 00002 |
| 4 | 00200 | 13 | 00001 |

**Tabla 2.15 Valores para la tabla de datos de 16bits.**

#### 2.4.1.3 CASO 24 BITS

Para este caso, se le agrega un bit más en la parte entera para ampliar el rango de entrada, obteniéndose con esto el formato [24 20] para las variables de entrada. Al igual que los casos anteriores al multiplicar las entradas con ese formato se obtiene un salida con el formato [48 40]. Limitando esta salida a 24 bits, el formato queda definido por [24 16], es por ello que internamente se tiene que agregar 4 bit más en la parte entera; por lo cual el  formato queda definido en [28 16] para la arquitectura interna.

El número de iteraciones adicionales es definido por el máximo valor que se obtiene con el formato [-8 7.99] para la multiplicación, el cual es 64. Para ello se encontró que son necesarias 6 iteraciones adicionales, y como máximo 20 iteraciones normales (ver tabla 2.16), pero al igual que en los sistemas coordenados anteriores, el número de iteraciones normales esta limitado a 16, quedando definidas en la siguiente tabla.

| i | Valor | i | Valor |
|---|---|---|---|
| -6 | 4000000 | 6 | 004000 |
| -5 | 2000000 | 7 | 002000 |
| -4 | 1000000 | 8 | 001000 |
| -3 | 800000 | 9 | 000800 |
| -2 | 400000 | 10 | 000400 |
| -1 | 200000 | 11 | 000200 |
| 0 | 100000 | 12 | 000100 |
| 1 | 080000 | 13 | 000080 |
| 2 | 040000 | 14 | 000040 |
| 3 | 020000 | 15 | 000020 |
| 4 | 010000 | 16 | 000010 |
| 5 | 008000 | | |

**Tabla 2.16 Valores para la tabla de datos de 24 bits.**

#### 2.4.1.4 CASO 32 BITS

Para 32 bits también se sigue el mismo procedimiento, se amplia el rango de valores enteros por lo que el formato de entrada queda definido en [32 27]. Al multiplicar señales del mismo formato se obtiene una respuesta con el siguiente formato [64 54]. Como se observa se necesitan 10 bits de parte entera, por lo cual

se tiene que agregar en la arquitectura interna 5 bit más para obtener la salida en el formato deseado.

Con el formato de entrada definido se obtienen valores en el rango de [-16 15.99], siendo el máximo valor conseguido en la multiplicación de X y Z es 256. Para este valor se necesitan 8 iteraciones adicionales, y el máximo número de iteraciones normales es 27, pero al igual que los casos anteriores este valor se limita a 16 iteraciones. A continuación se presenta la tabla de datos para 32 bits.

| I | Valor | i | Valor |
|---|---|---|---|
| -8 | 800000000 | 5 | 000400000 |
| -7 | 400000000 | 6 | 000200000 |
| -6 | 200000000 | 7 | 000100000 |
| -5 | 100000000 | 8 | 000080000 |
| -4 | 080000000 | 9 | 000040000 |
| -3 | 040000000 | 10 | 000020000 |
| -2 | 020000000 | 11 | 000010000 |
| -1 | 010000000 | 12 | 000008000 |
| 0 | 008000000 | 13 | 000004000 |
| 1 | 004000000 | 14 | 000002000 |
| 2 | 002000000 | 15 | 000001000 |
| 3 | 001000000 | 16 | 000000800 |
| 4 | 000800000 | | |

**Tabla 2.17 Valores para la tabla de datos de 32 bits.**

### 2.4.2 MODO VECTORING PARA EL CORDIC LINEAL.

En el modo vectoring, la función implementada es la división. Como se observó en (20), la operación de división depende del valor que tengan las variables X e Y en la entrada, mientras que la salida está definida en la variable Z. La consideración

que se debe tener para este caso es que $Z_0$= 0 con lo que se obtiene el resultado real en la variable Z.

En el caso de la división los datos de entrada deben estar limitados en el dividendo (ver Capítulo 1), ya que al ser este un valor muy pequeño, el resultado será muy grande, y no se puede representar con el tamaño del bus de datos designado. Es por eso que se hace un balance entre la salida y la entrada. A continuación se analiza cada caso por separado.

#### 2.4.2.1 CASO 12 BITS

Al igual que el modo rotation los datos iniciales de entrada X e Y tendrán el formato [12 10]. Mientras que el dato de salida queda fijado en [12 6], para que tenga 6 bits de parte fraccional y 6 bits de parte entera. Este formato se utiliza en la variable Z; de esta manera se obtiene un rango más amplio, hasta valores menores que 64. Por lo cual Y puede tomar todos los valores de entrada mientras que X toma valores mayores que 1.99/64 ([0.03123 1.99]). Con estos valores se observó que para representar valores de hasta 64, se necesitan 5 iteraciones adicionales y 7 iteraciones normales.

| I | Valor | i | Valor |
|---|---|---|---|
| -4 | 400 | 2 | 010 |
| -3 | 200 | 3 | 008 |
| -2 | 100 | 4 | 004 |
| -1 | 080 | 5 | 002 |
| 0 | 040 | 6 | 001 |
| 1 | 020 | | |

**Tabla 2.18 Valores para la tabla de datos de 12 bits.**

### 2.4.2.2 CASO 16 BITS

El rango de entrada también se amplia en 1 bit más para este caso, siendo el formato de entrada [16 13] para las variables X e Y; y para el resultado se define el formato de [16 8] para la variable Z. Con este formato se obtiene como valor máximo 256, dando como rango posible para el dividendo valores entre [0.01558 3.99]. Con el formato establecido para Z se halla que se necesitan 6 iteraciones adicionales, y además se utiliza 9 iteraciones normales. En la tabla 2.19 se muestra los valores para la tabla de datos para el caso de 16 bits.

| i | Valor | i | Valor |
|---|---|---|---|
| -6 | 4000 | 2 | 0040 |
| -5 | 2000 | 3 | 0020 |
| -4 | 1000 | 4 | 0010 |
| -3 | 0800 | 5 | 0008 |
| -2 | 0400 | 6 | 0004 |
| -1 | 0200 | 7 | 0002 |
| 0 | 0100 | 8 | 0001 |
| 1 | 0080 | | |

**Tabla 2.19 Valores para la tabla de datos de 16 bits.**

### 2.4.2.3 CASO 24 BITS

En este caso se partió de que el formato de entrada para las variables X e Y es [24 20], y que el formato de salida para Z es [24 12]. De esta manera el máximo valor que se obtiene como resultado es 1024, quedando limitado el rango de entrada para X en [0.0078 7.99]. Para representar esto valores de la división se

tiene que añadir 10 iteraciones en la expansión y 13 iteraciones normales, cuyos valores se muestran en la tabla 2.20.

| i | Valor | i | Valor |
|---|---|---|---|
| -10 | 400000 | 2 | 000400 |
| -9 | 200000 | 3 | 000200 |
| -8 | 100000 | 4 | 000100 |
| -7 | 080000 | 5 | 000080 |
| -6 | 040000 | 6 | 000040 |
| -5 | 020000 | 7 | 000020 |
| -4 | 010000 | 8 | 000010 |
| -3 | 008000 | 9 | 000008 |
| -2 | 004000 | 10 | 000004 |
| -1 | 002000 | 11 | 000002 |
| 0 | 001000 | 12 | 000001 |
| 1 | 000800 | | |

**Tabla 2.20 Valores para la tabla de datos de 24 bits.**

### 2.4.2.3 CASO 32 BITS

Para este caso de 32 bits, tambien le agregamos 1 bit más a la parte entera, con ello se obtiene un formato de [32 27] para las variables X e Y; y para el formato de salida se obtiene [32 16]. Con este formato de salida se obtuvo una respuesta de hasta 65536, con lo cual la entrada X queda limitada al rango [0.000244 15.99]. Como se observa el rango se amplia bastante, pero a costa de más iteraciones, ya que para lograr obtener valores en ese rango se necesitan 14 iteraciones adicionales más 16 iteraciones normales. Esto se puede apreciar en la tabla 2.21 que muestra los valores para la tabla de datos en 32 bits.

| I | Valor | I | Valor |
|---|---|---|---|
| -14 | 40000000 | 2 | 00004000 |
| -13 | 20000000 | 3 | 00002000 |
| -12 | 10000000 | 4 | 00001000 |
| -11 | 08000000 | 5 | 00000800 |
| -10 | 04000000 | 6 | 00000400 |
| -9 | 02000000 | 7 | 00000200 |
| -8 | 01000000 | 8 | 00000100 |
| -7 | 00800000 | 9 | 00000080 |
| -6 | 00400000 | 10 | 00000040 |
| -5 | 00200000 | 11 | 00000020 |
| -4 | 00100000 | 12 | 00000010 |
| -3 | 00080000 | 13 | 00000008 |
| -2 | 00040000 | 14 | 00000004 |
| -1 | 00020000 | 15 | 00000002 |
| 0 | 00010000 | 16 | 00000001 |
| 1 | 00008000 | | |

**Tabla 2.21 Valores para la tabla de datos de 32 bits.**

Como se puede observar para ampliar el rango de convergencia, es necesario incrementar los recursos lógicos y agregar más iteraciones, lo que incrementa el tiempo de procesamiento.

# CAPÍTULO 3

## IMPLEMENTACIÓN DEL ALGORITMO CORDIC PARA CADA SISTEMA

Para la implementación de las arquitecturas se tuvo en cuenta las siguientes consideraciones de diseño: el ancho de palabra de los datos de entrada (vector ($X_0$, $Y_0$) y el ángulo $Z_0$), la función que se desea desarrollar y la arquitectura que se va a utilizar (serial, iterativa, pipeline), así como ajustar los datos de entrada y salida al algoritmo con el cual se va a trabajar. Con ello, el sistema va a constar de cinco partes principales (ver figura 3.1): señales de entrada que son las señales de datos y control con las cuales se va a trabajar, bloque de pre procesamiento que se encarga de acomodar las señales de datos de entrada para que sean compatibles con el bloque de procesamiento, bloque de procesamiento del algoritmo CORDIC en el cual se desarrolla la operación deseada, bloque de post procesamiento que ajusta el tamaño de palabra de los resultados y estos puedan ser leído externamente y finalmente las señales de salida que muestran los resultados e indica cuando estos están listos para ser leídos.

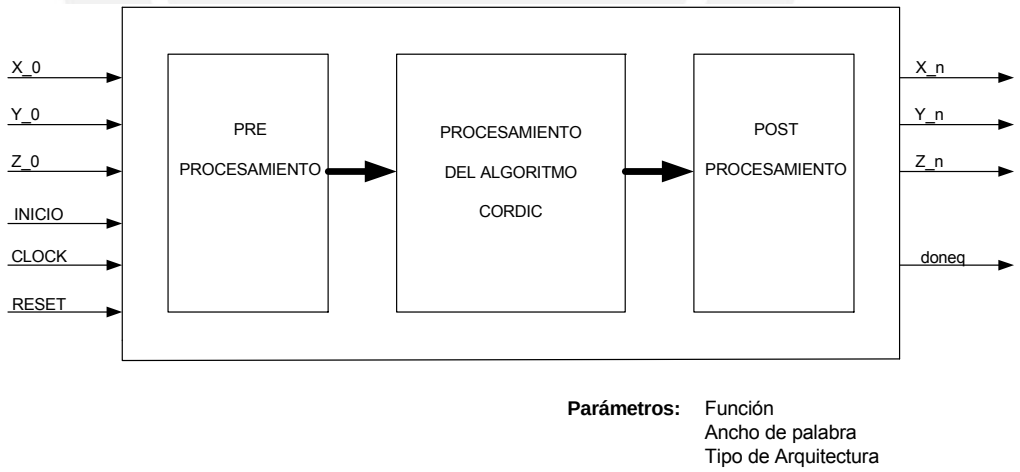



***Figura 3. 1 Diagrama de bloques general de la implementación del CORDIC***

Un criterio importante en la implementación del programa es el factor de precisión de datos para cada sistema [4]. Este factor, conocido como “rule of thumb”,

expresa lo siguiente: “Si ‘n’ bits se desean con precisión a la salida, los registros internos deben tener log2(n) bits de guarda en los bits de menor posición (LSB)” [4]. Esta consideración, aunque arbitraria, ha sido probada que trabaja muy bien para este caso. Otra consideración a tener en cuenta, es el análisis hecho en el capítulo 2, ya que este nos indica el número de iteraciones necesarias según el tamaño del bus. A continuación se analiza cada sistema por separado.

## 3.1 ARQUITECTURAS PARA EL SISTEMA CIRCULAR

Las arquitecturas presentes en este análisis implementan el algoritmo CORDIC Circular representado en la ecuación 6. Se emplearon registros, desplazadores y sumadores/restadores del tamaño hallado en el análisis anterior (ver capítulo 2). Los 3 tipos de arquitectura implementados son: Iterativa, Serial y Pipeline. Antes de describir cada tipo de arquitectura, es necesario definir la nomenclatura que es usada en el diagrama de bloques.

n= número de bits entrada/salida.

N= número de iteraciones normales.

ng= log2(n), número de bits de guarda.

nz= número de bits en la arquitectura interna para la variable Z

nr ≥ n+ng, número de bits en la arquitectura interna para las variables X e Y.

### 3.1.1 ARQUITECTURA ITERATIVA

En la arquitectura iterativa fue necesario definir una máquina de estados (ver figura 3.2); en la cual se observa que posee 2 estados de procesamiento. S1, definido como estado inicial, esta a la espera que se active en alta la señal de inicio (s). Este pulso en alta indica al sistema que los datos de entrada están listos para su captura colocando la señal s_xyz='1' y luego pasa al estado siguiente. En el estado S2, se realiza el procesamiento de las señales con la ejecución del algoritmo para la obtención de las funciones escogidas al inicio.

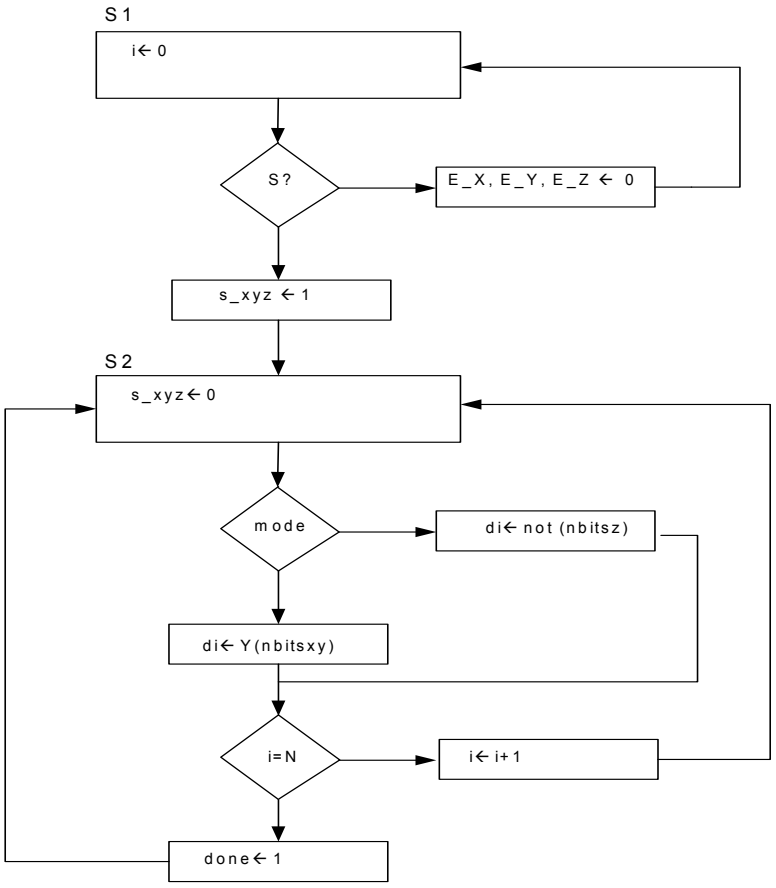



**Figura 3.2 Diagrama de estados de la arquitectura Iterativa**

La máquina de estados maneja las operaciones lógicas y aritméticas que se realizan en esta arquitectura (figura 3.3), donde se permite el ingreso de los datos

iniciales ($X_0$, $Y_0$, $Z_0$) si la señal s_xyz='0', caso contrario entran los datos realimentados. Dependiendo del modo de operación en que se encuentre el algoritmo, se escoge el valor de la variable $d_i$, ya sea con el signo de la variable Z si el modo es *Rotation* (5), o con el signo de la variable Y si el modo es *Vectoring* (9). Finalmente si el valor del número de iteraciones definido por *'n'* (11, 15 o 16) llega al valor deseado, se regresa al estado inicial; sino se incrementa el número de iteraciones y se realiza el procedimiento para la siguiente rotación.

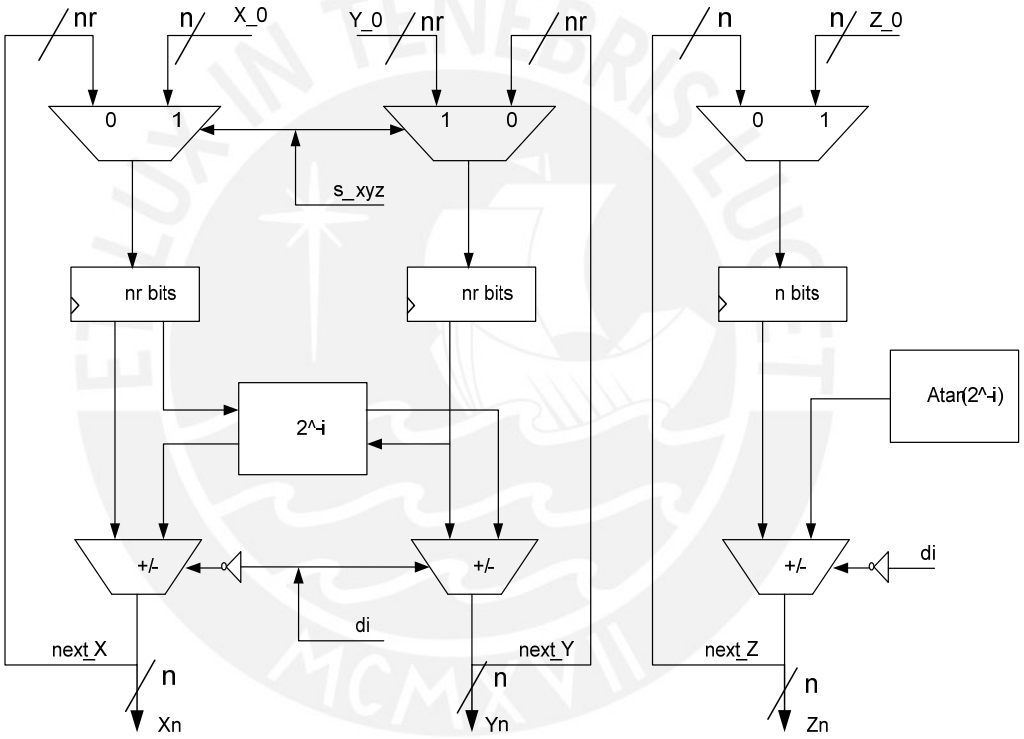



**Figura 3.3 Arquitectura Iterativa del CORDIC Circular**

Los registros utilizados son de 'nr' bits para la señales X e Y, ya que se necesita, para el modo rotation, añadir un bit mas a la parte entera (ver capítulo 2); mientras que para Z, el registro utilizado es de 'n' bits. En la operación de suma y/o resta con el registro Z se utilizó una tabla de datos que tiene la forma atan($2^{-i}$), la cual contiene los valores de las tablas 2.2, 2.3, 2.4 y 2.5.

A continuación se muestra el diagrama de tiempo para el caso del modo *rotation* para 12 bits, en el que se observa que el tiempo de procesamiento para obtener los resultados después de activar la señal *'s'*, depende del número de iteraciones.

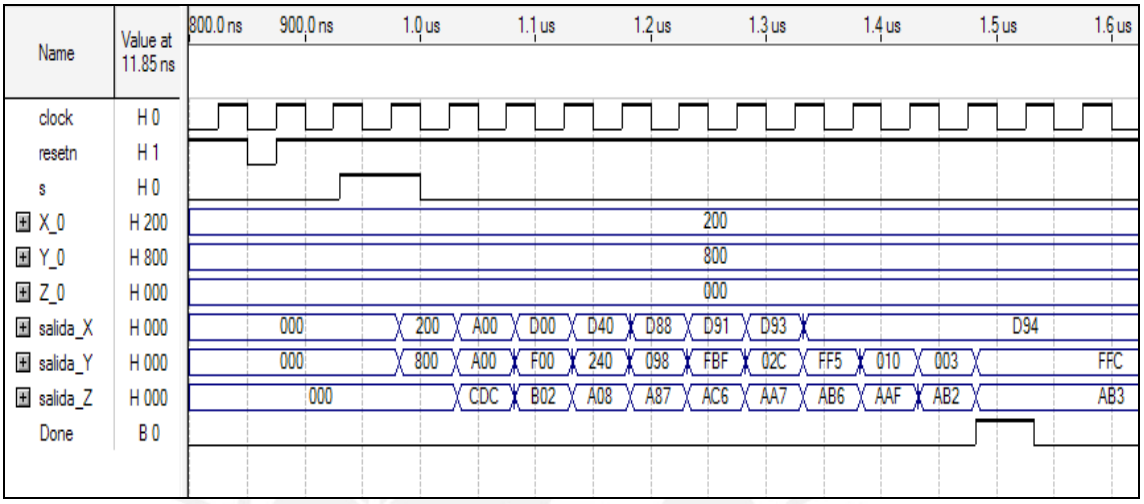



**Figura 3.4 Diagrama de tiempo para 12 bits en modo Vectoring.**

Se observa que el tiempo de obtención de datos validos desde que se presiona la tecla de inicio es de 11 ciclos de reloj. Esto se debe a que cada iteración toma un ciclo de reloj para ejecutarse. Por tal motivo, el resultado para 16, 24 y 32 bits es 15, 16 y 16 ciclos de reloj respectivamente para ambos modos de procesamiento.

### 3.1.2 ARQUITECTURA SERIAL

Al igual que en la arquitectura Iterativa, se implementa una máquina de estados (ver figura 3.5). En este caso se utiliza 3 estados, ya que los datos de entrada ingresados serialmente necesitan un estado más para poder ser almacenados. Al igual que el caso anterior, en el estado S1 se espera que se active la señal 's' para iniciar la ejecución del algoritmo. En el estado S2 los datos de entrada se cargan serialmente, por lo que se utiliza un contador para controlar que todos los

bits del dato se carguen correctamente. En el estado S3 se realiza el procesamiento de los datos para ello se usan 2 contadores, uno que indica el tamaño del bus (señal 'p'), el mismo que se utilizo para S2, y el otro q verifique el número de iteraciones realizadas (señal 'i').

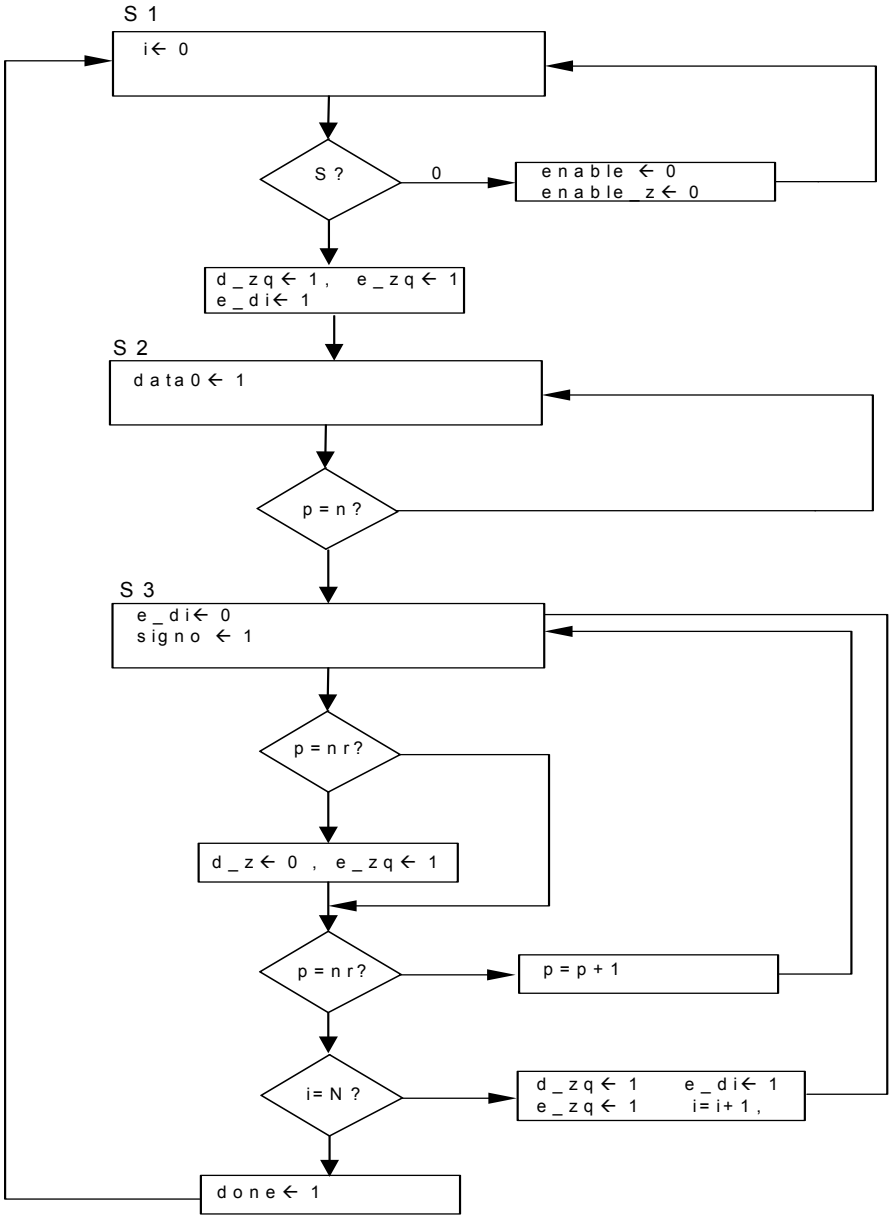



**Figura 3.5 Diagrama de estados de la Arquitectura serial.**

La arquitectura serial (ver figura 3.6) manejada por la máquina de estados descrita anteriormente cuenta con un registro de datos de entrada serial y salida

serial/paralela, además de utilizar multiplexores, ya que dependiendo del estado en que se encuentre, se escoge entre 2 tipos de entrada: el inicial o el realimentado.

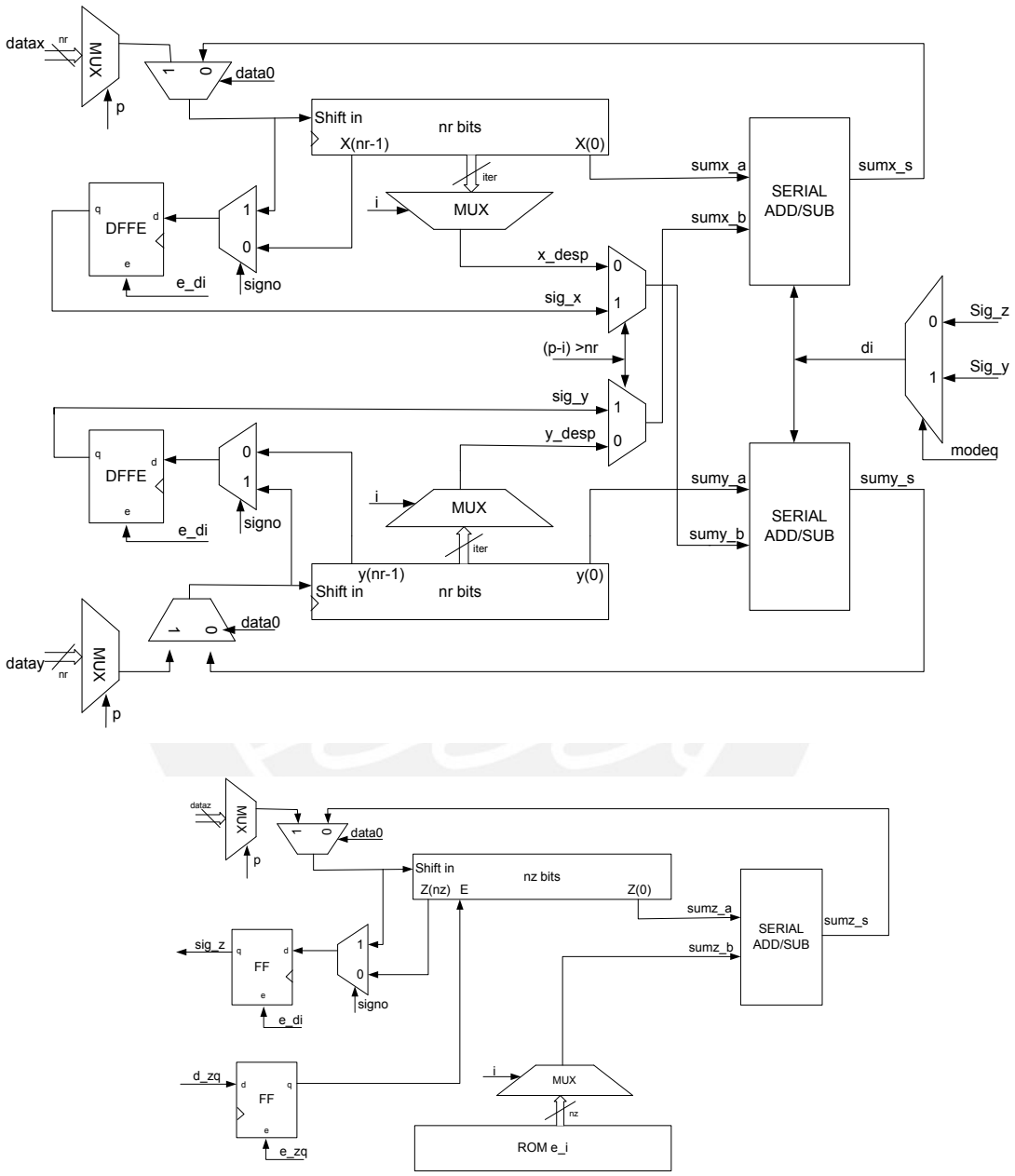



**Figura 3.6 Diagrama de bloques de la Arquitectura Serial.**

Los multiplexores sirven para que al desplazarse los datos en las iteraciones se utilicen los mismos registros para guardar la información. Otra característica de esta arquitectura es que los datos deben estar sincronizados, por lo cual se

utilizan flip flops adicionales que capturan el signo del dato procesado en cada iteración, sin que este sea modificado en el proceso.

Teniendo en cuenta que Z tiene menos bits que X e Y, es necesario una señal que avise el termino de procesamiento de toda la información en cada iteración; es por ello que se utiliza un flip flop como el habilitador del registro Z (ver figura 3.6). Con este tipo de arquitectura se obtuvo el siguiente resultado como se muestra en el diagrama de tiempos (figura 3.7).

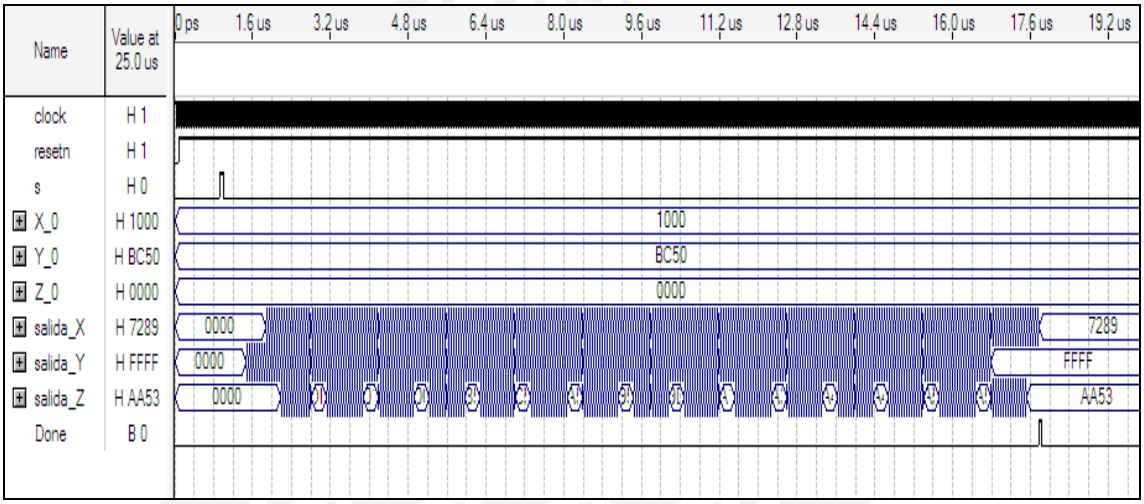



**Figura 3.7 Diagrama de tiempo para 16 bits en el modo Vectoring.**

Se observa en el diagrama de tiempos que la duración en cargar el dato inicial es de 20 ciclos de reloj, ya que debe cargarse los ‘nr’ bits de datos de las variables X e Y como se observa en el estado 2 (ver figura 3.6); mientras que la duración en ejecutar las operaciones de cada iteración es de (nr) x (N) ciclos de reloj, donde N es el número de iteraciones, para este caso es 300 ciclos de reloj. Finalmente para los casos de 12, 24 y 32 se obtiene como tiempo de procesamiento 176, 476 y 612 ciclos de reloj respectivamente para ambos modos de procesamiento.

### 3.1.3 ARQUITECTURA PIPELINE

Este tipo de arquitectura no necesita de máquina de estados, ya que contiene todo los registros, sumadores y demás componentes de hardware necesarios para el procesamiento en cada iteración (ver figura 3.8).

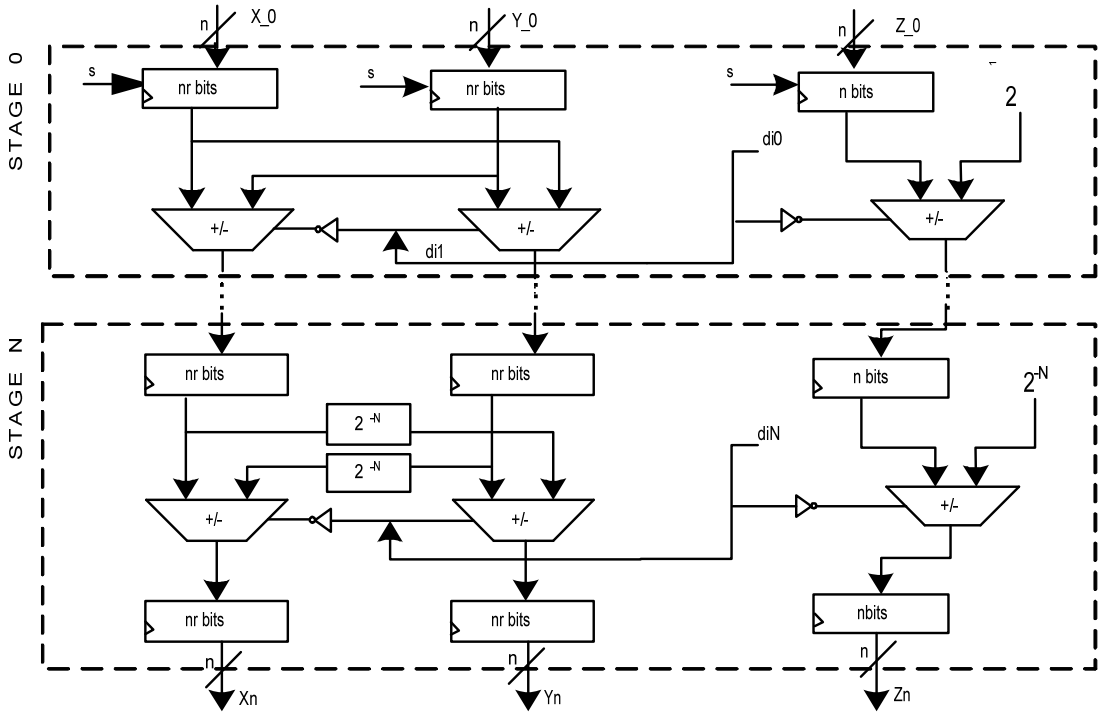



**Figura 3.8 Diagrama de bloques de la Arquitectura Pipeline**

En este caso no son necesarios multiplexores para el desplazamiento de las variables X e Y, ya que se asigna el dato desplazado como operando de sumador/restador. Así mismo los resultados de la simulación de esta arquitectura se muestran en el siguiente diagrama de tiempos.

Se puede observar que el resultado se obtiene después de 11 ciclos de reloj, al igual que en la arquitectura iterativa (ver figura 3.9), pero para este tipo de arquitectura se obtiene un nuevo resultado de operación en el siguiente ciclo de reloj.

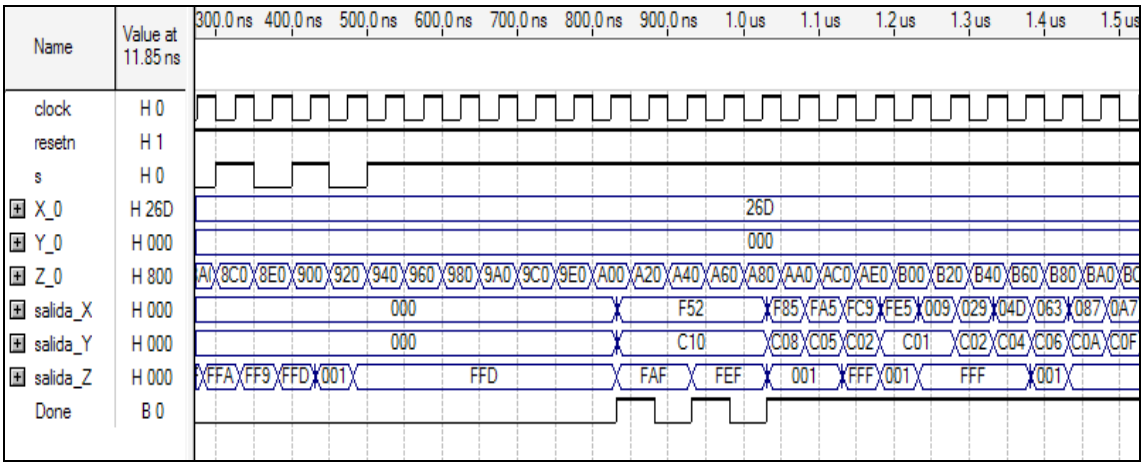



**Figura 3.9 Diagrama de tiempo para 12 bits en modo Vectoring.**

## 3.2 ARQUITECTURAS PARA EL CORDIC HIPERBÓLICO.

Debido a que en este sistema se implementan las arquitecturas con expansión de rango de convergencia, y estas implican un cambio en la ejecución de las ecuaciones, se divide el procesamiento del algoritmo en 2 etapas. En la primera etapa se implementa la expansión del algoritmo para valores de i≤0, donde la ecuación que se tiene que implementar (28) se divide en 2 operaciones consecutivas como se observa en la figura 3.10; y en la segunda etapa se implementa las operaciones para las iteraciones normales. Para este sistema también se considera los bits de guarda [4] y el análisis hecho en el capítulo 2. Las arquitecturas implementadas son las mismas que para el CORDIC Circular y la nomenclatura usada se define a continuación.

n= número de bits entrada/salida.

N= número de iteraciones normales.

M=número de iteraciones adicionales.

ng= log2(n), número de bits de guarda.

nz= número de bits en la arquitectura interna para Z.

nr≥ n+ng, número de bits en la aquitectura interna para X e Y.

### 3.2.1 ARQUITECTURA ITERATIVA.

En este diseño es necesario incorporar un estado más en la máquina de estados de los que usa el CORDIC Circular, ya que en este estado adicional se controla el procesamiento para las iteraciones adicionales. El estado para controlar la expansión es S2, mientras que en el estado S3 se controla las iteraciones normales teniendo en cuenta que se repiten las iteraciones 4 y 13 (ver capítulo 1).

En el diagrama de bloques (figura 3.11) se observa en la parte superior la implementación para la expansión i≤0. Esta necesita 2 multiplexores, 2 registros, 4 sumadores y 2 desplazadores. Esta parte es la más crítica en el diseño, ya que añade un considerable retardo, el cual reduce la frecuencia de operación. En la parte de inferior del diagrama de bloques se implementan las iteraciones i>0, esta implementación es clásica y se encuentra en varios trabajos y documentos [5].

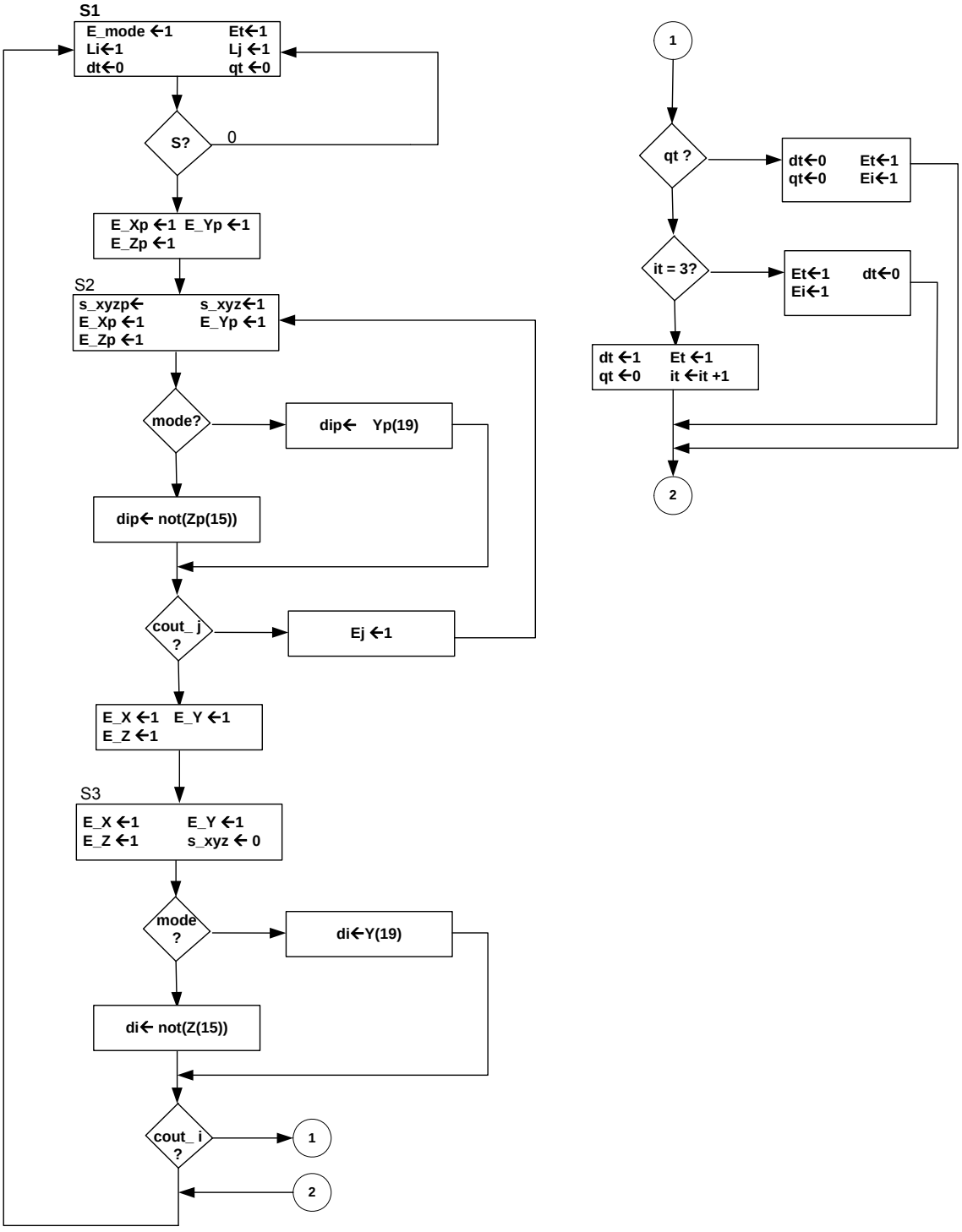



**Figura 3.10 Diagrama de estados de la Arquitectura Iterativa Hiperbólica.**

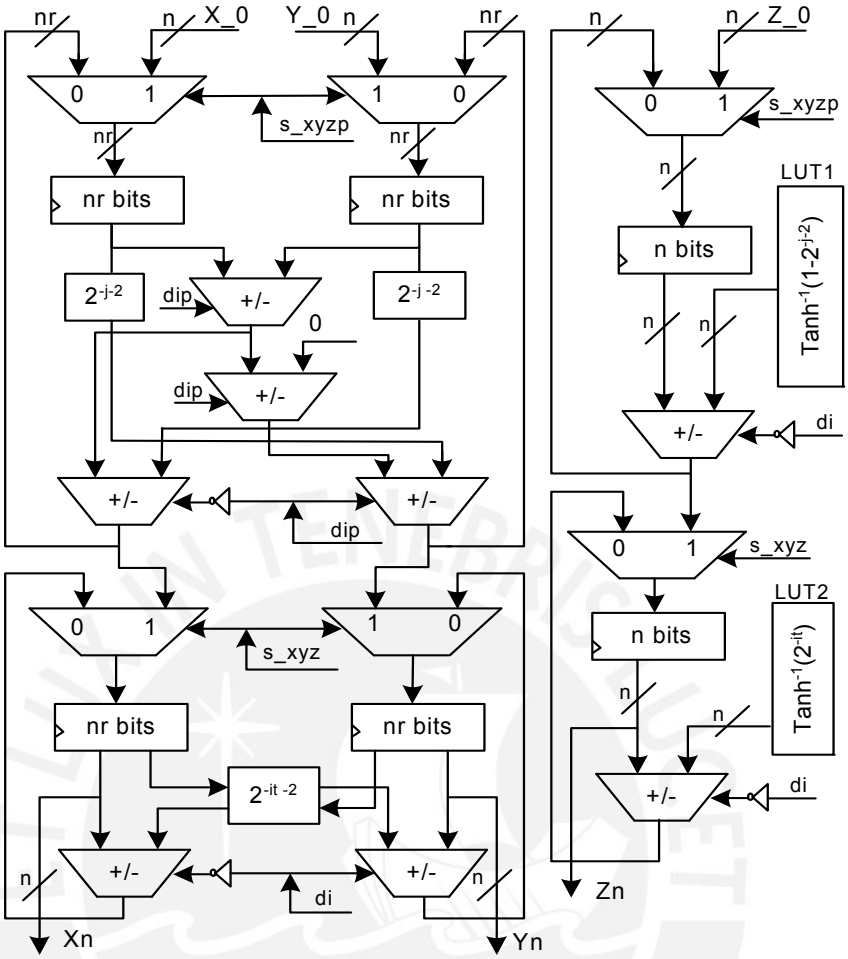



**Figura 3.11 Diagrama de bloques de la Arquitectura Iterativa Hiperbólico.**

En la figura 3.12 se muestra el diagrama de tiempos para el caso de 12 bits para el modo *vectoring.* En este caso, el tiempo de procesamiento total es de 12 ciclos de reloj, esto se debe a que aparte de las iteraciones normales (9) y la repetición de la cuarta iteración, se añade 2 iteraciones adicionales más (ver tabla 2.10). Finalmente el tiempo de procesamiento para la obtención del resultado es M+1+N+v ciclos de reloj, donde v es el número de iteraciones repetidas (1 o 2).

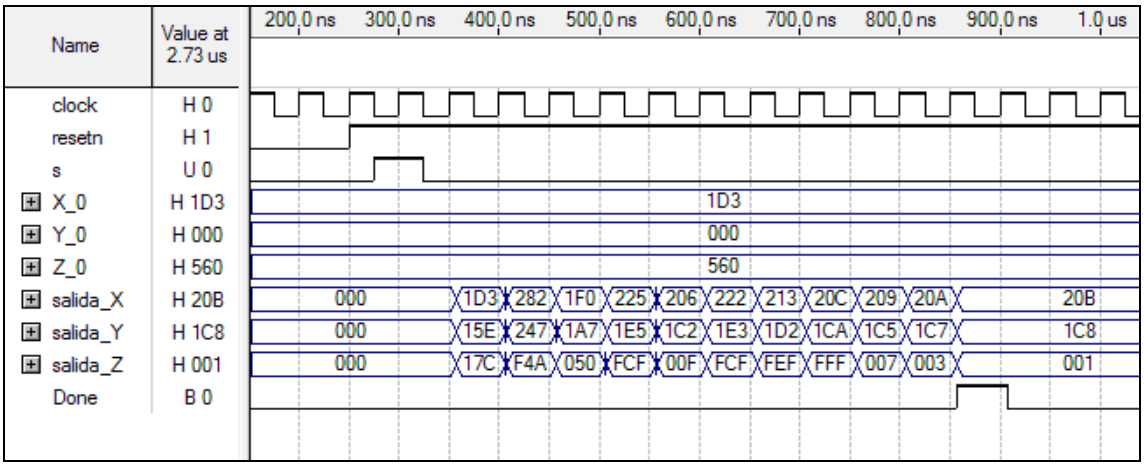



**Figura 3.12 Diagrama de tiempo para 12 bits en modo Rotation.**

### 3.2.2 ARQUITECTURA SERIAL

Para este tipo de arquitectura se debe tener en cuenta retrasar los datos cuando se realiza la parte de la expansión del rango de convergencia. Como se puede apreciar en la figura 3.13 se le agrega la parte de expansión a la arquitectura serial del CORDIC Circular. En este diagrama de estados se observan 4 estados, a diferencia de la arquitectura Iterativa, se le agrega un estado más (S2) para la carga inicial de las variables.

En el diagrama de bloques mostrado en la figura 3.14, se observa que para el desplazamiento de datos se utiliza un registro de desplazamiento con 2 multiplexores; uno de los multiplexores es para las iteraciones adicionales y el otro es para las iteraciones normales; de esta manera se reduce el número de registros desplazadores.

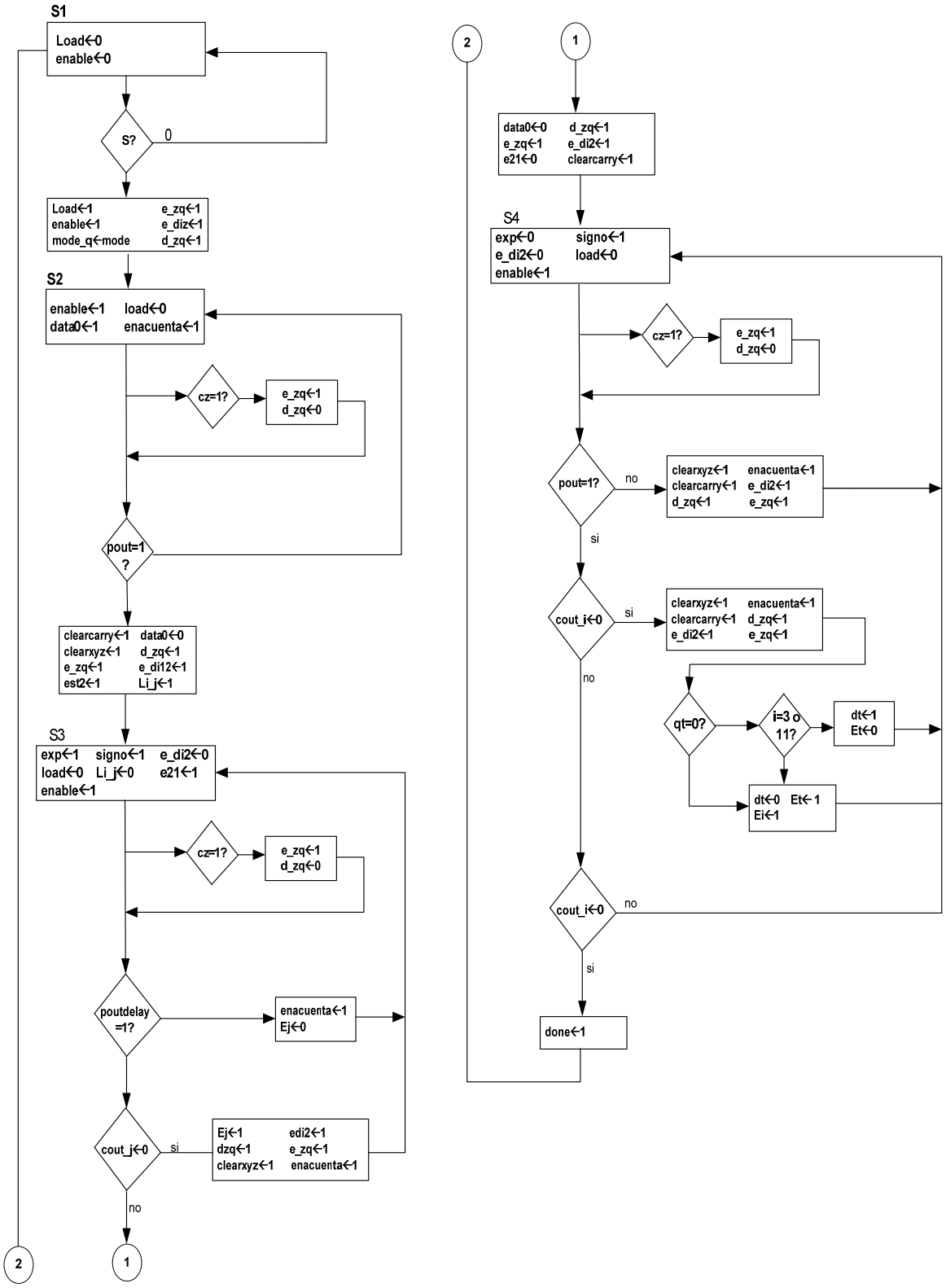



**Figura 3.13 Diagrama de bloques del CORDIC Hiperbólico.**

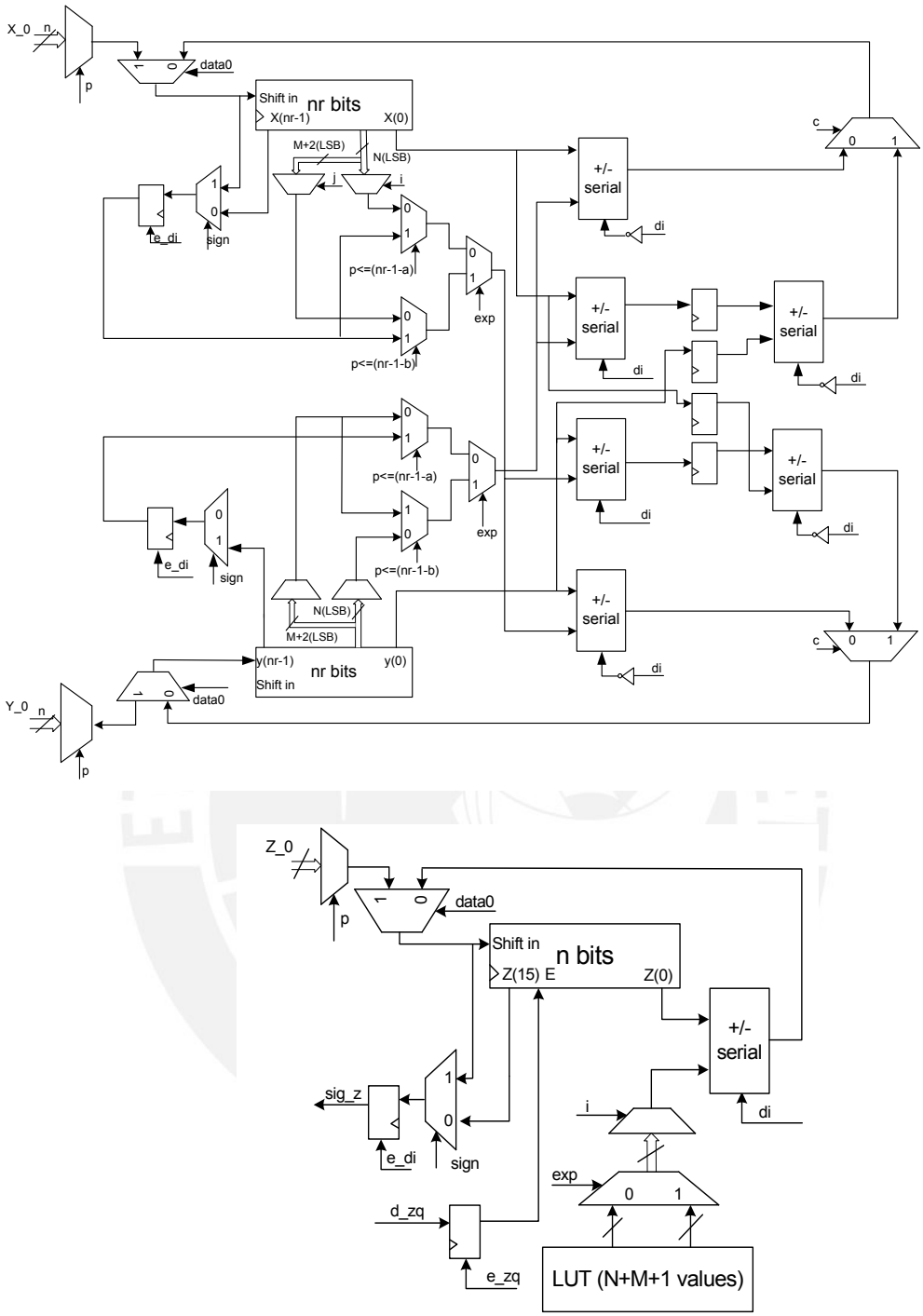



**Figura 3.14 Diagrama de bloques del CORDIC Hiperbólico.**

Como para el caso de la expansión se realizan 2 operaciones de suma/resta consecutivas se colocó flip flops después de la primera operación para poder sincronizar los datos. Esta operación extra que se realiza se considera en el estado S3, ya que aporta un ciclo más de retardo en comparación con la etapa

que ejecuta las iteraciones normales. A continuación se muestra el diagrama de tiempos para una operación en el modo *rotation* de 12 bits de bus de datos.

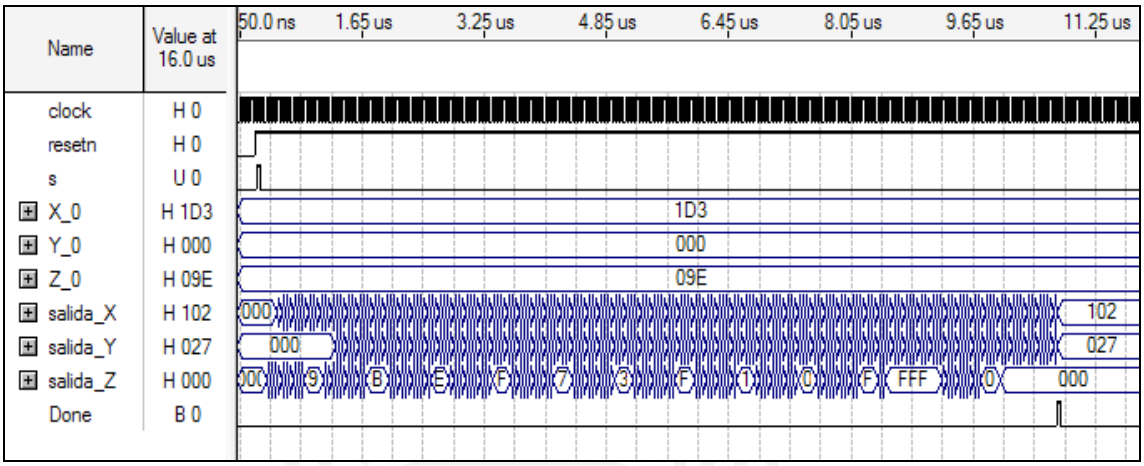



**Figura 3.15 Diagrama de tiempos del CORDIC Hiperbólico.**

El resultado para esta operación se obtiene después de 209 ciclos de reloj. Esto se debe a que se demora 16 ciclos de reloj en el estado S2, 17 ciclos de reloj para la iteración adicional y 11x16 ciclos de reloj para S3 (ver tabla 2.6). Finalmente el resultado para cada operación toma M.(nr+1)+(N+v +1).(nr) ciclos de reloj, donde v es el número de iteraciones que se repiten.

### 3.2.3 ARQUITECTURA PIPELINE

Al igual que para el caso del CORDIC Circular se implementa el hardware necesario para cada operación. Como se observa en la figura 3.16, para el caso de la expansión se implemento registros para cada una de las 2 operaciones que se mencionan anteriormente; es por eso que para la variable Z es necesario retrasar un ciclo más de reloj para la sincronización de las variables. Estos cambios se hacen para evitar retardos en el procesamiento, es por ello que según los parámetros descritos anteriormente se tiene 2(M+1)+N etapas.

Para obtener un resultado en esta arquitectura es necesario esperar un retardo inicial de 2(M+1)+N ciclos de reloj. Esto se observa en la figura 3.17, en la que para el caso de 24 bits en el modo *vectoring* son necesarios 28 ciclos de reloj.

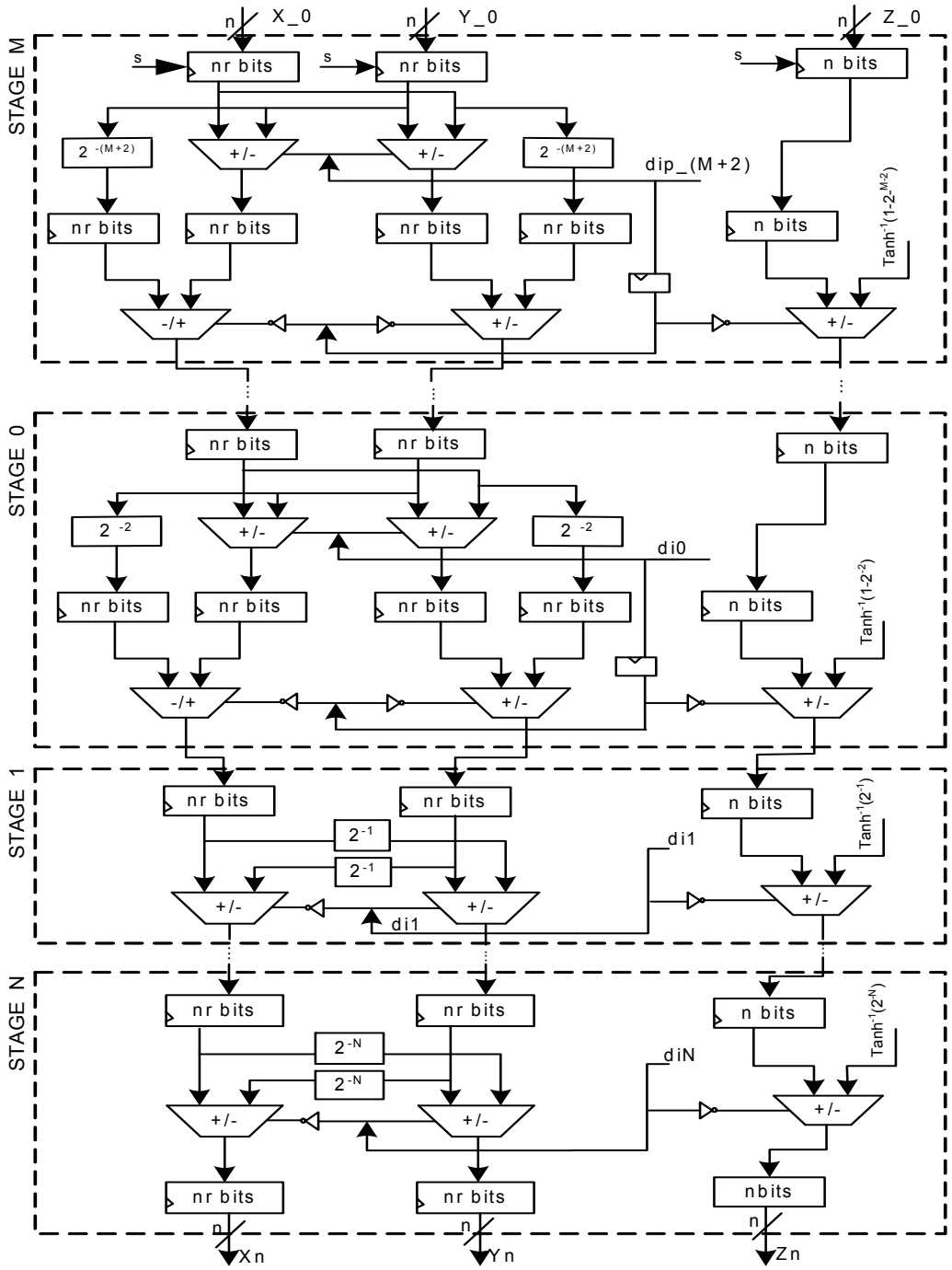



**Figura 3.16 Diagrama de bloques del CORDIC Hiperbólico.**

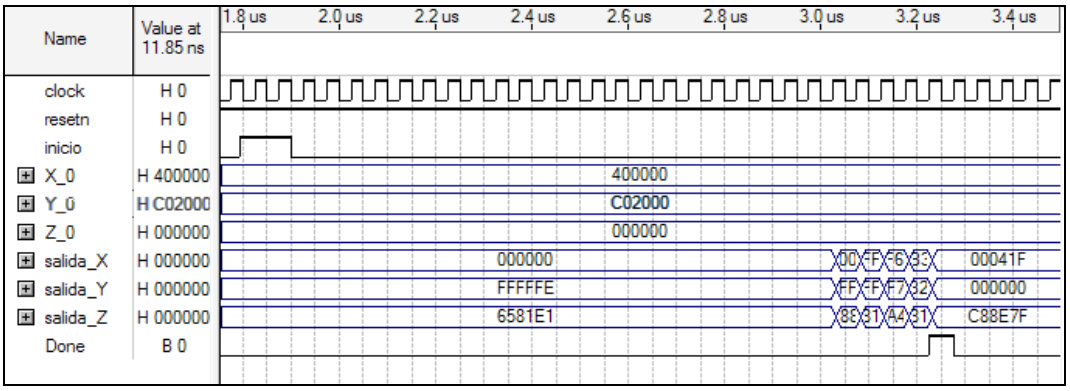



**Figura 3.17 Diagrama de tiempos para 24 bits modo Vectoring.**

## 3.3 ARQUITECTURAS PARA EL CORDIC LINEAL

Tambien en este caso se implemento los tres tipos de arquitectura mencionados previamente. Como se puede apreciar en la figuras siguientes existe un gran parecido a la implementación del CORDIC Circular, esto se debe a la gran similitud que tiene tanto lo que corresponde a las iteraciones normales como a las iteraciones adicionales. A continuación se explica el desarrollo para las 3 arquitecturas.

### 3.3.1 ARQUITECTURA ITERATIVA

Debido a la expansión del rango de convergencia para este sistema, se añadió un estado más en comparación con el sistema circular. Como se puede apreciar en el diagrama de estados (ver figura 3.18), en el estado S1 se espera que active en alta la señal inicio *'s'* para poder pasar al siguiente estado. En el estado S2 las operaciones realizadas son la de expansión, y como se indicó anteriormente, se le

indica al registro de desplazamiento utilizado, mediante la señal dirección, que el desplazamiento es para la izquierda. Para que ocurra el cambio del estado de S2 a S3, el contador de iteraciones adicionales debe llegar a su fin. En el estado S3, se realizan las iteraciones normales y el procedimiento seguido es básicamente el mismo que en el sistema circular, solo que en este caso la variable X no se modifica.

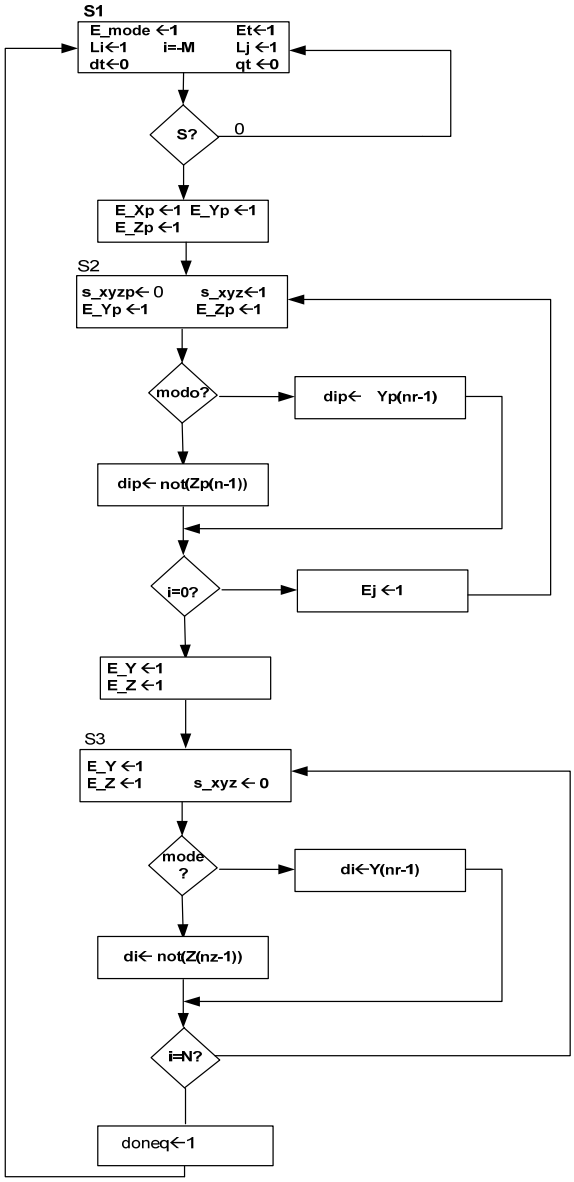



**Figura 3.18 Diagrama de estados para la arquitectura Iterativa.**

En la figura 3.19 vemos el diagrama de bloques propuesto para la arquitectura iterativa, el procedimiento que se sigue es muy parecido a los dos sistemas anteriores. Se dividió en un bloque la expansión, mientras que el otro bloque se desarrolla las iteraciones normales.

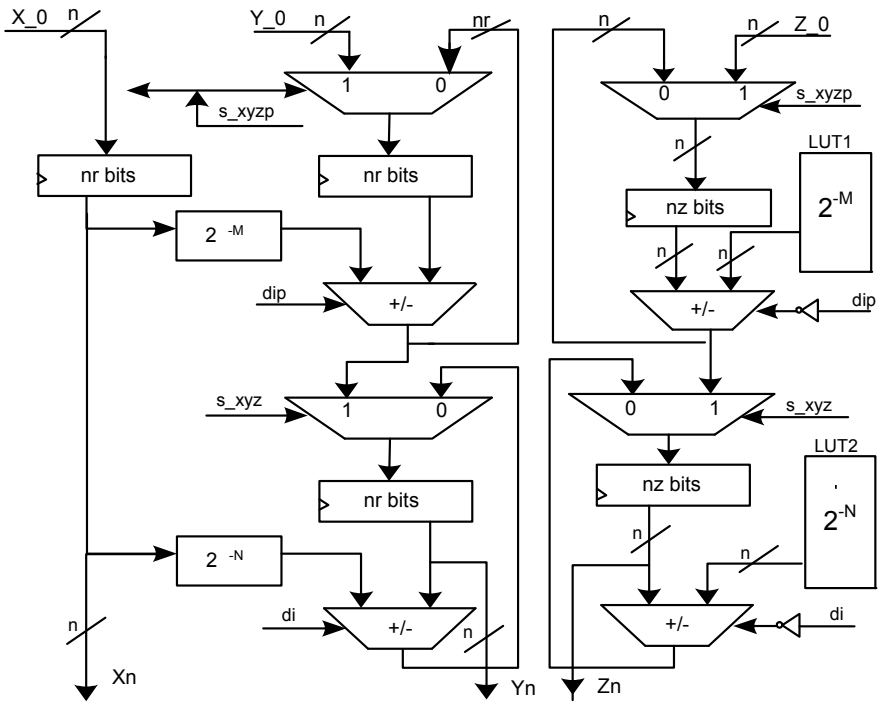



**Figura 3.19 Diagrama de bloques para la arquitectura Iterativa.**

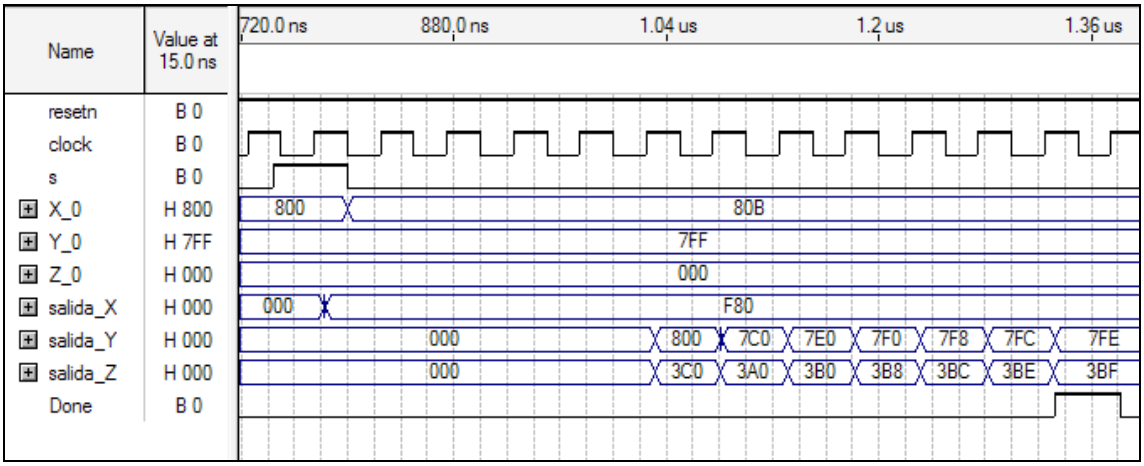



**Figura 3.20 Diagrama de tiempos para 12 bits en el modo Vectoring.**

En el diagrama de tiempo se observa que son necesarios 12 ciclos de reloj para obtener la respuesta en este modo, esto se debe a que se utilizan 7 iteraciones normales y 4 iteraciones adicionales. Es por ello que una operación con esta arquitectura demanda M+N ciclos de reloj para la obtención del resultado.

### 3.3.2 ARQUITECTURA SERIAL

Para el caso de la arquitectura serial, el diseño implica otro estado más, en el cual se cargan los datos serialmente. Entonces el primer estado S1, está a la espera de la señal inicio ‘s‘ para pasar al estado S2 y cargar los datos iniciales serialmente. Las operaciones para ejecutar las iteraciones normales y adicionales se realizan en el estado S3 y S4 respectivamente (ver figura 3.21).

En el diagrama de bloques se observa que se dividen en 2 etapas el procesamiento. Esta implementación es muy parecida en ambos casos.

El tiempo de procesamiento para obtener un resultado es de (M+N+1).nr ciclos de reloj (ver tabla 3.13), es por eso que para el caso de la figura 3.23 el resultado se obtiene después de 247 ciclos de reloj.

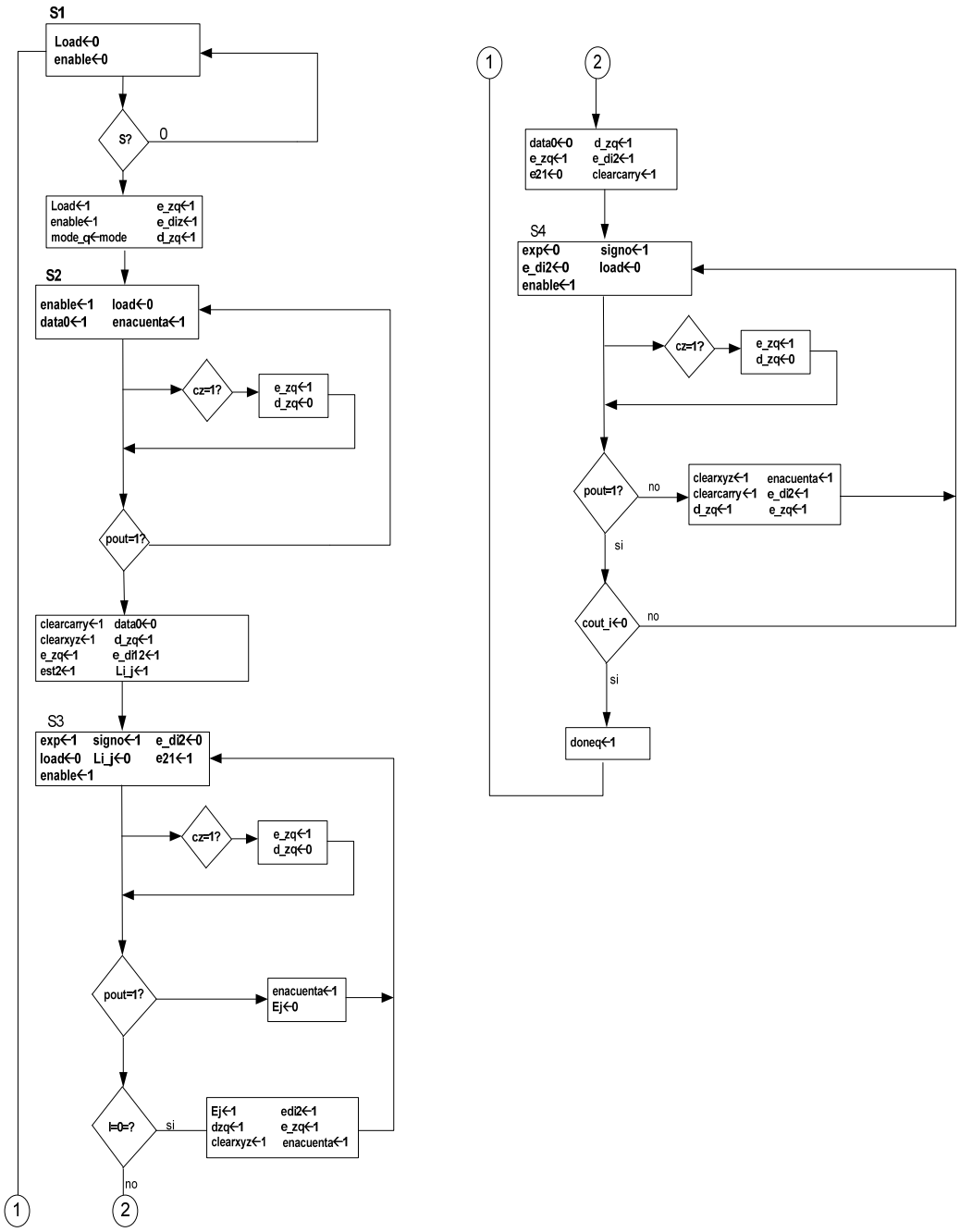



**Figura 3.22 Diagrama de estados para la arquitectura Serial.**

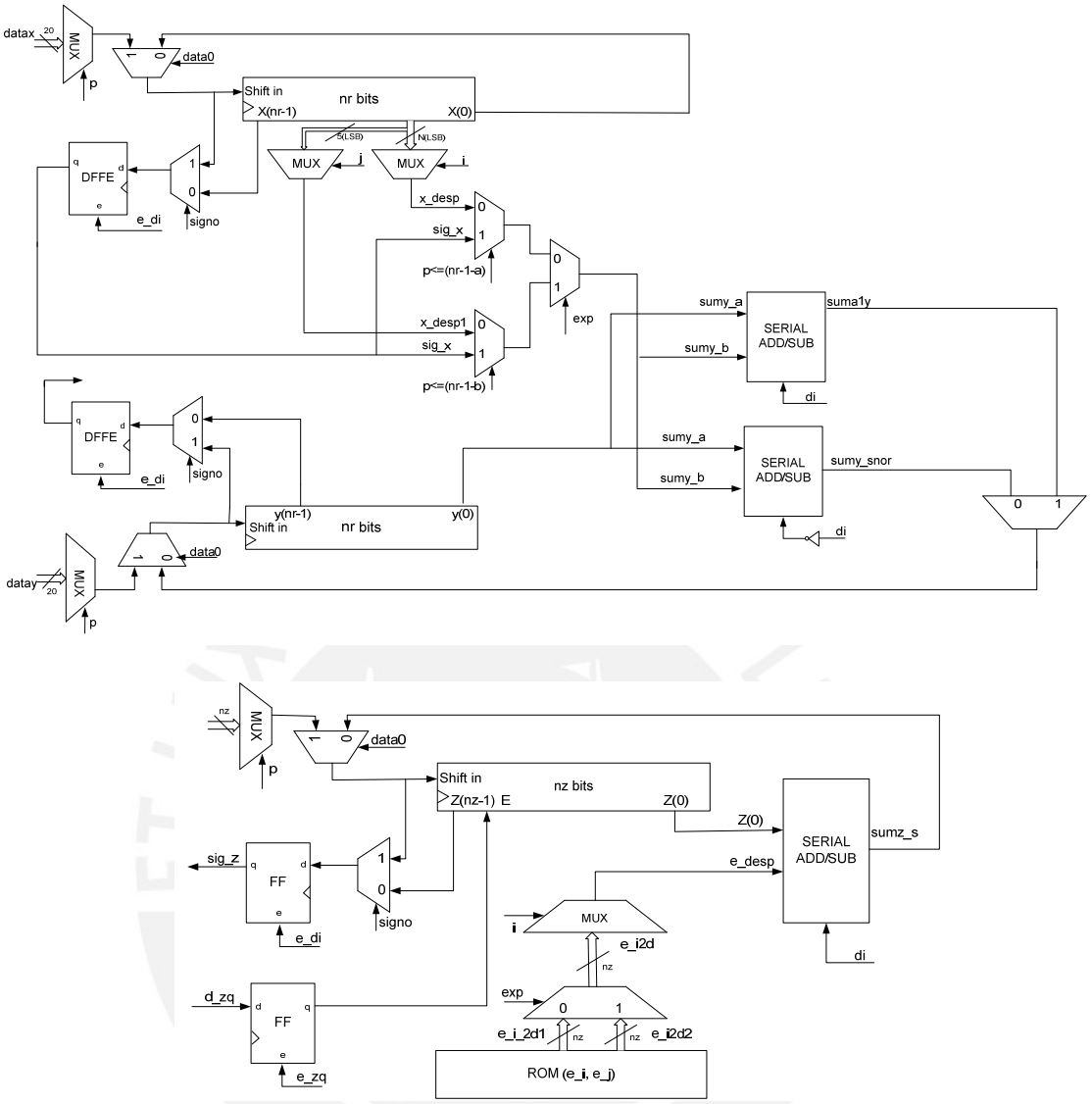



**Figura 3.21 Diagrama de bloques para la arquitectura Serial.**

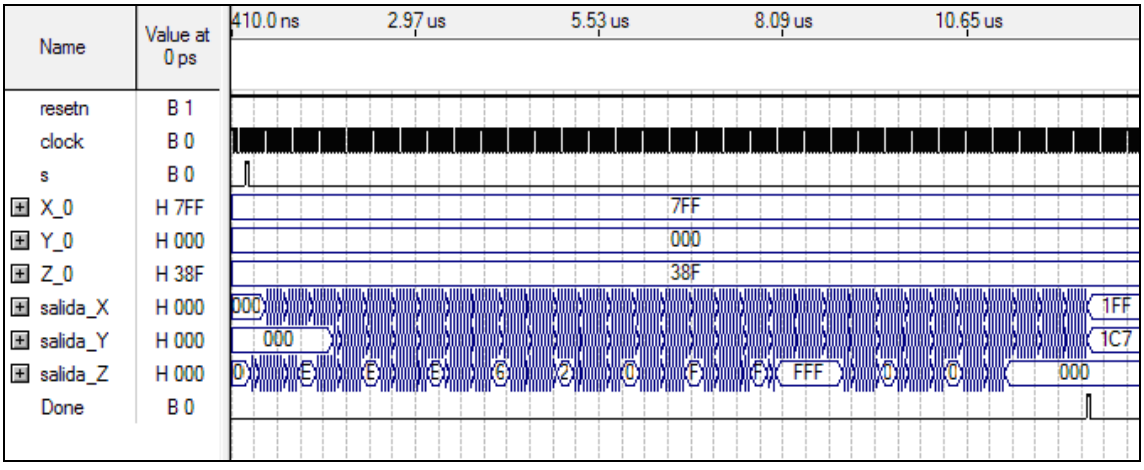



**Figura 3.23 Diagrama de tiempo de 12 bits para el modo Rotation.**

### 3.3.3 ARQUITECTURA PIPELINE

En esta arquitectura no es necesario máquina de estados, se puede apreciar que se utiliza un registro para cada operación de iteración. La implementación es la misma tanto para la expansión como para el rango normal, la única diferencia es que en el primero los datos se desplazan hacia la izquierda, mientras que en el segundo los datos se desplazan a la derecha. Se observa además que solo se utilizó un registro para la variable X, ya que es necesario guardar el valor inicial de dicha variable.

En el diagrama de tiempos (ver figura 3.25) se muestra que los resultados para el caso de 12 bits para el modo rotation se obtiene después de 10 ciclos de reloj. Este tiempo se debe a que se procesan 7 iteraciones normales y 4 iteraciones adicionales (ver tabla 2.18). Finalmente para cada caso el resultado se obtiene después de M+N ciclos de reloj.

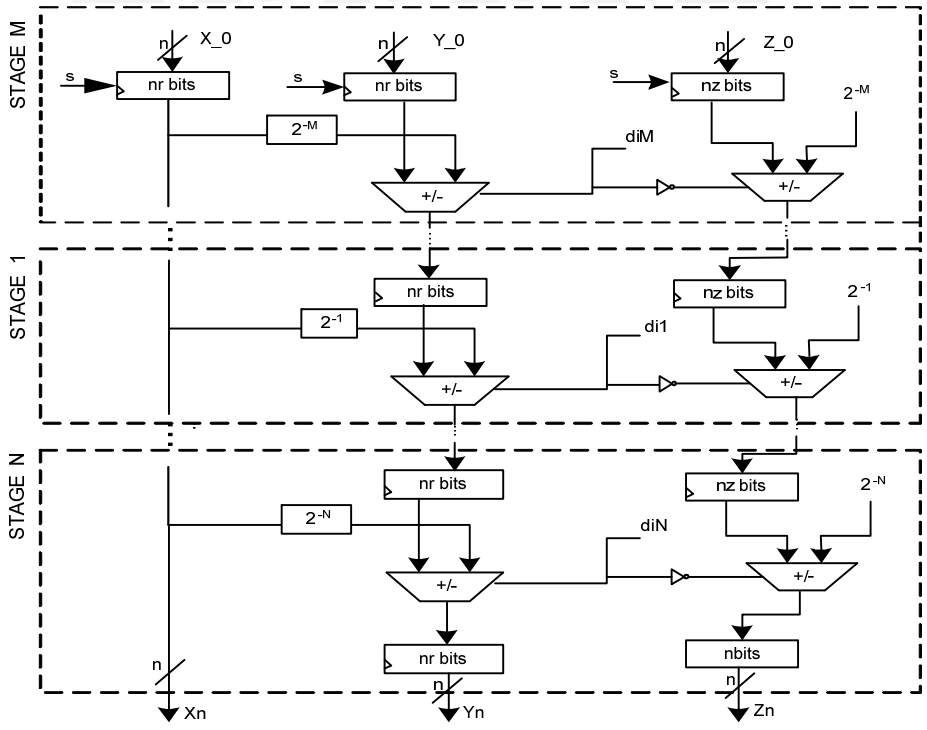



**Figura 3.24 Diagrama de bloques para la arquitectura Pipeline.**

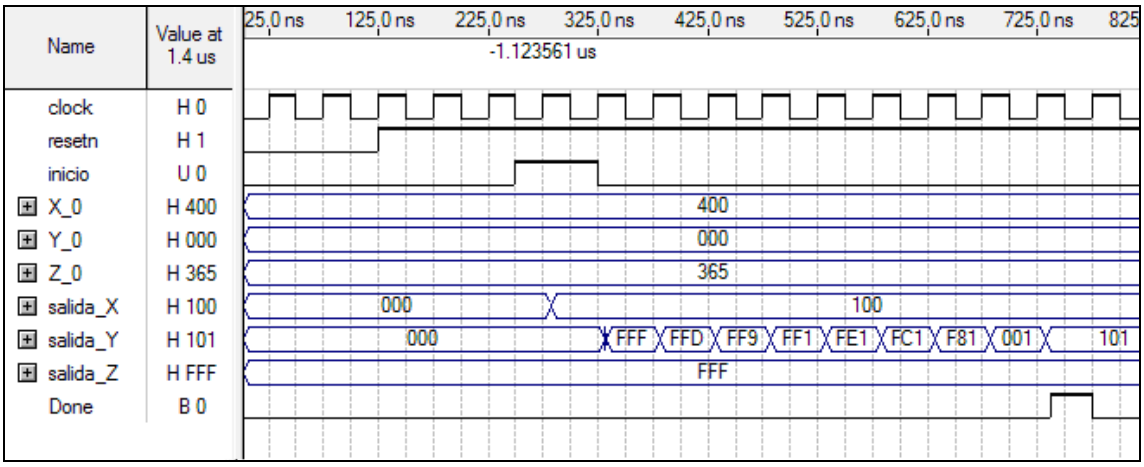



**Figura 3.25 Diagrama de tiempo de 12 bits para el modo Vectoring.**

# CAPÍTULO 4

## RESULTADOS Y ERROR RELATIVO

## 4.1 RESULTADOS DE LA IMPLEMENTACIÓN EN EL FPGA

El rendimiento de las arquitecturas implementadas (ver capítulo 3) se realizó para tres familias de FPGAs (STRATIX, STRATIX II, CYCLONE) y empleando el compilador del programa QUARTUS II v. 5.0 ambos del fabricante Altera. Se recopiló la frecuencia máxima de operación (Fmax), el uso de elementos lógicos (LEs) y el porcentaje que ocupa la implementación en el dispositivo para cada caso a partir de los tamaños de palabras (N° Bits) vistos (12, 16, 24 y 32 bits).

### 4.1.1 RESULTADOS PARA EL CORDIC CIRCULAR.

En el modo vectoring la función implementada es el arco tangente mientras que en el modo rotation se implementan las funciones seno y coseno. Viendo el tipo de arquitectura (ver tabla 4.1 y 4.2), el pipeline posee Fmax de más del doble del iterativo y mayores al serial, pero ocupa más LEs que el iterativo y el serial, siendo este último el de menor ocupación de recursos. Según el número de Bits, los mejores resultados se obtienen cuando el número de bits es pequeño con la desventaja del decaimiento en la precisión del resultado (ver capítulo 3). En el dispositivo de la familia Stratix II se tienen las más altas Fmax y la menor ocupación de LEs.

| TIPO | N BITS | STRATIX EP1S10F484C5 | | | CYCLONE EP1C4F324C6 | | | STRATIX II EP2S15F484C3 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Fmax | LEs | %LEs | Fmax | LEs | % LEs | Fmax | LEs | % LEs |
| ITERATIVO | 12 | 104.67 | 242 | 2% | 132.36 | 242 | 8% | 172.77 | 177/52 | 1% |
| | 16 | 103.24 | 309 | 2% | 129.79 | 309 | 7% | 154.73 | 215/64 | 1% |
| | 24 | 93.8 | 476 | 4% | 110.57 | 476 | 11% | 134.73 | 333/89 | 2% |
| | 32 | 79.47 | 602 | 5% | 108.08 | 602 | 15% | 131.79 | 419/113 | 3% |
| SERIAL | 12 | 167.39 | 148 | 1% | 205.34 | 148 | 5% | 235.4 | 112/64 | <1% |
| | 16 | 132.14 | 181 | 1% | 196.89 | 181 | 4% | 243.01 | 132/78 | 1% |
| | 24 | 129.53 | 237 | 2% | 175.01 | 237 | 5% | 191.5 | 176/102 | 1% |
| | 32 | 117.38 | 303 | 2% | 166.06 | 303 | 7% | 193.65 | 222/127 | 1% |
| PIPELINE | 12 | 279.33 | 507 | 4% | 286.53 | 505 | 12% | 324.89 | 534/497 | 4% |
| | 16 | 201.82 | 934 | 8% | 262.26 | 932 | 23% | 329 | 976/924 | 7% |
| | 24 | 218.91 | 1401 | 13% | 252.27 | 1339 | 34% | 275.94 | 1446/1391 | 11% |
| | 32 | 211.51 | 1809 | 17% | 226.5 | 1867 | 45% | 237.76 | 1854/1799 | 14% |

**Tabla 4.1 Resultados para el modo Vectoring- CORDIC Circular.**

| TIPO | N BITS | STRATIX EP1S10F484C5 | | | CYCLONE EP1C4F324C6 | | | STRATIX II EP2S15F484C3 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Fmax | LEs | %LES | Fmax | LEs | %LES | Fmax | LEs | %LES |
| ITERATIVO | 12 | 103.47 | 229 | 2% | 136.99 | 229 | 5% | 171.56 | 168/50 | 1% |
| | 16 | 104.44 | 295 | 2% | 127.84 | 295 | 7% | 146.03 | 207/62 | 1% |
| | 24 | 91.22 | 464 | 4% | 114.23 | 464 | 11% | 144.15 | 324/87 | 2% |
| | 32 | 79.87 | 600 | 5% | 101.29 | 600 | 15% | 131.23 | 410/111 | 3% |
| SERIAL | 12 | 138.05 | 140 | 1% | 200.4 | 140 | 4% | 205.38 | 112/64 | <1% |
| | 16 | 151.42 | 179 | 1% | 200.68 | 179 | 4% | 203.58 | 131/77 | 1% |
| | 24 | 132.45 | 229 | 2% | 198.77 | 229 | 5% | 181.39 | 170/101 | 1% |
| | 32 | 114.23 | 298 | 2% | 179.4 | 298 | 7% | 176.4 | 218/126 | 1% |
| PIPELINE | 12 | 267.67 | 489 | 4% | 283.37 | 487 | 12% | 390.02 | 516/475 | 4% |
| | 16 | 270.27 | 906 | 8% | 281.29 | 904 | 22% | 323.21 | 948/896 | 7% |
| | 24 | 221.68 | 1371 | 12% | 257.53 | 1309 | 34% | 275.86 | 1416/1361 | 11% |
| | 32 | 218.05 | 1779 | 16% | 215.15 | 1777 | 44% | 274.26 | 1824/1769 | 14% |

**Tabla 4.2 Resultados para el modo Rotation- CORDIC Circular.**

### 4.1.2 RESULTADOS PARA EL CORDIC HIPERBÓLICO.

Los resultados obtenidos para cada modo de procesamiento se muestran en las tablas 4.3 y 4.4.

| TIPO | N BITS | STRATIX EP1S10F484C5 | | | CYCLONE EP1C4F324C6 | | | STRATIX II EP2S15F484C3 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Fmax | LEs | %LES | Fmax | LEs | %LES | Fmax | LEs | %LES |
| ITERATIVO | 12 | 101.5 | 405 | 3% | 97.04 | 405 | 13% | 160.51 | 290/99 | 2% |
| | 16 | 103.49 | 515 | 4% | 95.88 | 515 | 12% | 132.96 | 367/123 | 2% |
| | 24 | 89.15 | 808 | 7% | 81.03 | 808 | 20% | 128.67 | 564/172 | 4% |
| | 32 | 84.86 | 1065 | 10% | 80.79 | 1065 | 26% | 121.17 | 742/220 | 5% |
| SERIAL | 12 | 142.69 | 187 | 1% | 132.96 | 187 | 6% | 187.58 | 156/77 | 1% |
| | 16 | 127.93 | 218 | 2% | 115.79 | 218 | 5% | 129.55 | 176/89 | 1% |
| | 24 | 122.29 | 288 | 2% | 123.95 | 288 | 7% | 148.1 | 235/115 | 1% |
| | 32 | 119.82 | 364 | 3% | 116.05 | 364 | 9% | 130.11 | 314/139 | 2% |
| PIPELINE | 12 | 248.57 | 759 | 7% | 220.56 | 759 | 18% | 346.02 | 800/751 | 6% |
| | 16 | 247.77 | 1248 | 11% | 217.86 | 1248 | 31% | 318.88 | 1302/1240 | 10% |
| | 24 | 228.21 | 2343 | 22% | 168.38 | 2343 | 58% | 274.65 | 2416/2335 | 19% |
| | 32 | 176.21 | 3249 | 30% | *171.66 | 3249 | *26% | 212.4 | 3330/3241 | 26% |

**Tabla 4.3 Resultados para el modo Vectoring- CORDIC Hiperbólico.**

| TIPO | N BITS | STRATIX EP1S10F484C5 | | | CYCLONE EP1C4F324C6 | | | STRATIX II EP2S15F484C3 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Fmax | LEs | %LES | Fmax | LEs | %LES | Fmax | LEs | %LES |
| ITERATIVO | 12 | 101.39 | 377 | 3% | 98.6 | 377 | 12% | 153.7 | 277/98 | 2% |
| | 16 | 99.49 | 541 | 5% | 98.43 | 541 | 13% | 144.45 | 382/127 | 3% |
| | 24 | 82.67 | 834 | 7% | 81.29 | 834 | 20% | 137.16 | 583/176 | 4% |
| | 32 | 79.83 | 1126 | 10% | 81.53 | 1126 | 28% | 112.11 | 790/228 | 6% |
| SERIAL | 12 | 141.3 | 184 | 1% | 138.43 | 184 | 6% | 224.42 | 149/76 | 1% |
| | 16 | 118.25 | 235 | 2% | 121.54 | 235 | 5% | 167.56 | 200/92 | 1% |
| | 24 | 117.58 | 295 | 2% | 111.21 | 295 | 7% | 135.74 | 240/117 | 1% |
| | 32 | 124.38 | 366 | 3% | 106.56 | 366 | 9% | 133.12 | 301/144 | 2% |
| PIPELINE | 12 | 271.15 | 624 | 5% | 250.82 | 624 | 15% | 361.14 | 660/616 | 5% |
| | 16 | 226.3 | 1288 | 12% | 217.34 | 1288 | 32% | 320.31 | 1340/1280 | 10% |
| | 24 | 226.65 | 2397 | 22% | 174.46 | 2397 | 59% | 283.05 | 2470/2389 | 19% |
| | 32 | 186.29 | 3801 | 35% | *164.45 | 3801 | *31% | 229.57 | 3890/3795 | 31% |

**Tabla 4.4 Resultados para el modo Rotation- CORDIC Hiperbólico.**

* Para el caso de 32 bits en ambos modos de procesamiento (ver * en tablas 4.3 y 4.4) se utilizó otro dispositivo de CYCLONE-EP1C12F324C6, ya que debido a el gran uso de recursos de hardware que se utilizan para este tipo de arquitectura, no se puede implementar en el dispositivo designado al inicio.

En este sistema la función implementada para el modo vectoring es la función arco tangente hiperbólico y para el modo rotation las funciones seno y coseno hiperbólico. Al igual que para el CORDIC Circular las Fmax más altas se obtienen con la familia Stratix II; además podemos apreciar que debido a la expansión se incrementa el usos de LEs en comparación con el caso anterior sin embargo la Fmax no disminuye, ya que el procesamiento se separó en 2 etapas. Se observa que la Arquitectura Serial presenta un balance entre los LEs usados y la Fmax obtenida respecto a las otras 2 arquitecturas pero como se observo anteriormente (capítulo 3) el resultado se obtiene en un tiempo mucho mayor que los otros casos.

### 4.1.3 RESULTADOS PARA EL CORDIC LINEAL.

Las funciones implementadas para este caso son la función división para el modo vectoring y la función multiplicación para el modo rotation. Los resultados de la compilación se muestran en las tablas 4.5 y 4.6. Se observa que para el caso del modo vectoring (tabla 4.5), cuya función implementada es la división, se utiliza el doble de elementos lógicos que para la implementación de la función multiplicación, esto se debe a que se ejecutan más iteraciones que el primero

(ver tabla 2.21). En la arquitectura pipeline se presentan las más altas Fmax en comparación con los otros sistemas llegando a los 424.27 KHz para 12 bits en la Stratix II. Para el caso de la multiplicación, modo rotation, se obtiene un mejor rendimiento en cuanto a LEs utilizados y Fmax obtenida. También se observa que para el dispositivo de CYCLONE se utiliza menos de la mitad de %Les. en el caso extremo (32bits – modo vectoring).

| TIPO | N BITS | STRATIX EP1S10F484C5 | | | CYCLONE EP1C4F324C6 | | | STRATIX II EP2S15F484C3 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Fmax | LEs | %LES | Fmax | Les | %LES | Fmax | Les | %LES |
| ITERATIVO | 12 | 125.77 | 237 | 2% | 158.13 | 237 | 5% | 251.64 | 163/86 | 1% |
| | 16 | 112.08 | 330 | 3% | 141.63 | 330 | 8% | 210.3 | 225/111 | 1% |
| | 24 | 91.11 | 521 | 4% | 125.66 | 521 | 13% | 189.18 | 333/160 | 2% |
| | 32 | 79.49 | 740 | 7% | 93.19 | 740 | 18% | 135.92 | 469/213 | 3% |
| SERIAL | 12 | 176.03 | 149 | 1% | 194.33 | 149 | 3% | 281.45 | 124/74 | <1% |
| | 16 | 156.76 | 180 | 1% | 204.75 | 180 | 4% | 313.68 | 148/90 | 1% |
| | 24 | 116.39 | 253 | 2% | 175.01 | 253 | 6% | 269.98 | 223/125 | 1% |
| | 32 | 97.07 | 307 | 3% | 163.96 | 307 | 7% | 295.42 | 266/161 | 2% |
| PIPELINE | 12 | 238.27 | 275 | 2% | 296.12 | 275 | 6% | 424.27 | 281/11 | 2% |
| | 16 | 255.68 | 463 | 4% | 292.91 | 463 | 11% | 395.41 | 471/345 | 4% |
| | 24 | 164.42 | 983 | 9% | 249.44 | 983 | 24% | 337.84 | 995/709 | 7% |
| | 32 | 139.7 | 1695 | 16% | 231.91 | 1695 | 42% | 279.7 | 1714/1201 | 13% |

**Tabla 4.5 Resultados para el modo Vectoring- CORDIC Lineal.**

| TIPO | N BITS | STRATIX EP1S10F484C5 | | | CYCLONE EP1C4F324C6 | | | STRATIX II EP2S15F484C3 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Fmax | LEs | %LES | Fmax | LEs | %LES | Fmax | Les | %LES |
| ITERATIVO | 12 | 121.12 | 222 | 2% | 148.17 | 222 | 5% | 248.32 | 154/85 | 1% |
| | 16 | 118.89 | 332 | 3% | 139.53 | 332 | 8% | 207.63 | 223/111 | 1% |
| | 24 | 94.26 | 491 | 4% | 121.54 | 491 | 12% | 192.9 | 334/156 | 2% |
| | 32 | 91.46 | 667 | 6% | 112.96 | 667 | 16% | 160.44 | 437/261 | 3% |
| SERIAL | 12 | 149.93 | 154 | 1% | 195.75 | 154 | 3% | 295.77 | 123/70 | <1% |
| | 16 | 132.05 | 194 | 1% | 180.47 | 194 | 4% | 254.78 | 163/87 | 1% |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **24** | 137.32 | 237 | 2% | 198.29 | 237 | 5% | 285.39 | 190/114 | 1% |
| | **32** | 112.83 | 315 | 2% | 149.59 | 315 | 7% | 234.19 | 257/143 | 2% |
| **PIPELINE** | **12** | 264.06 | 178 | 1% | 339.9 | 177 | 4% | 415.97 | 302/284 | 2% |
| | **16** | 240.67 | 350 | 3% | 296.91 | 349 | 8% | 352.49 | 620/589 | 4% |
| | **24** | 192.2 | 585 | 5% | 270.56 | 584 | 14% | 314.66 | 1064/1023 | 8% |
| | **32** | 180.08 | 748 | 7% | 243.66 | 747 | 18% | 273 | 1377/1328 | 11% |

**Tabla 4.6 Resultados de simulación para el modo Rotation- CORDIC Lineal.**

## 4.2 ERROR RELATIVO PARA CADA SISTEMA.

Se definió el error relativo para cada sistema (29) como los valores obtenidos en las simulaciones hechas en QUARTUS II 5.0 que se contrasta con los valores obtenidos en MATLAB. Es decir, los datos de entrada en la simulación son transformados según el formato que les corresponde (ver capítulo 2) en MATLAB y son procesados con la función que se desea comparar. Por otro lado los valores de los resultados obtenidos en la simulación para la función implementada son transformados según su formato de salida con el mismo programa y son comparados en la fórmula de error relativo, siendo el valor ideal, el hallado con el programa MATLAB.

$$Error\ \mathrm{Re}lativo = \left| \frac{ValorIdeal - ValorCORDIC}{ValorIdeal} \right| \qquad (29)$$

### 4.2.1 ERROR RELATIVO PARA EL CORDIC CIRCULAR

Para este caso se halló el error relativo para las funciones seno, coseno y arco tangente.

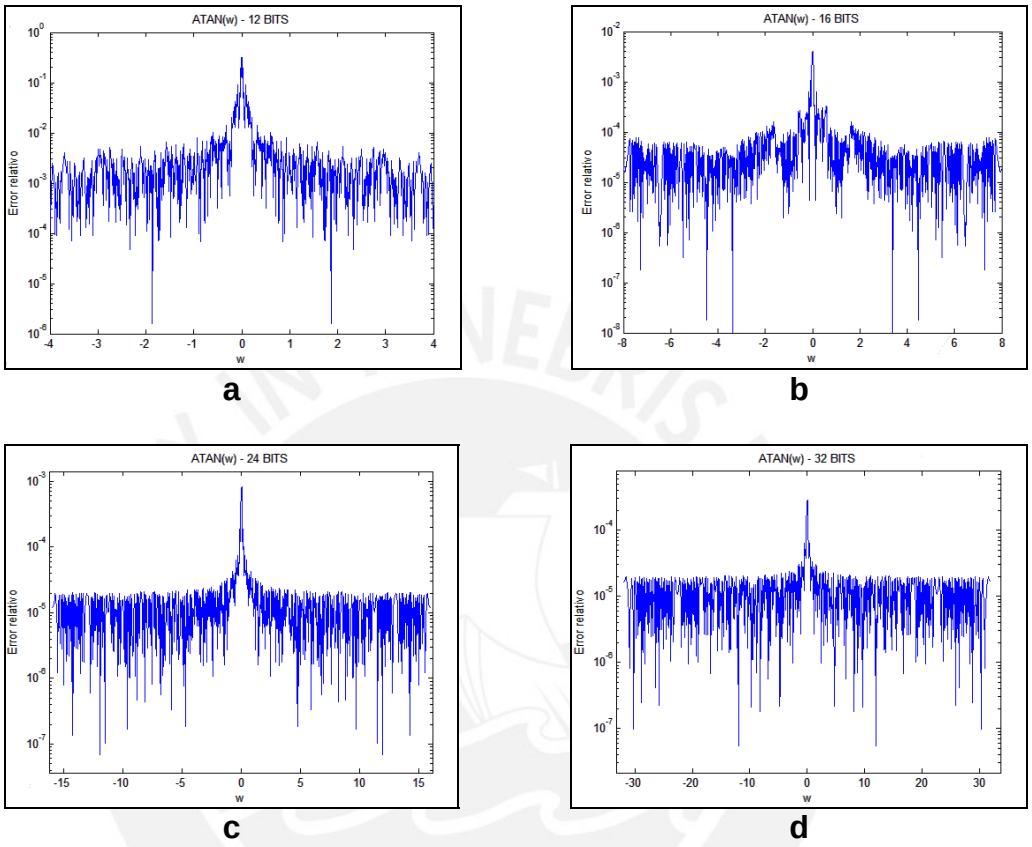



**Figura 4.1 Error Relativo para la función Arco Tangente**

**a) 12, b) 16, c) 24 y d) 32 bits**

Como se puede observar, se aprecia una disminución de error al aumentar el ancho del bus de datos. Otro detalle que se observa en el caso del arco tangente es que presenta un error relativo alto para valores cercanos a cero. Esto se debe a que los bits asignados para la parte fraccional no son los necesarios para obtener una buena precisión en cuanto a resultados muy pequeños. Además se

observa que no es apreciable la diferencia entre el error relativo para los valores hallados cuando el bus de datos es 24 bits y cuando es 32 bits, sin embargo el valor del error disminuye para valores cercanos, esto se debe al incremento de bits de parte fraccional para el caso de 32 bits.

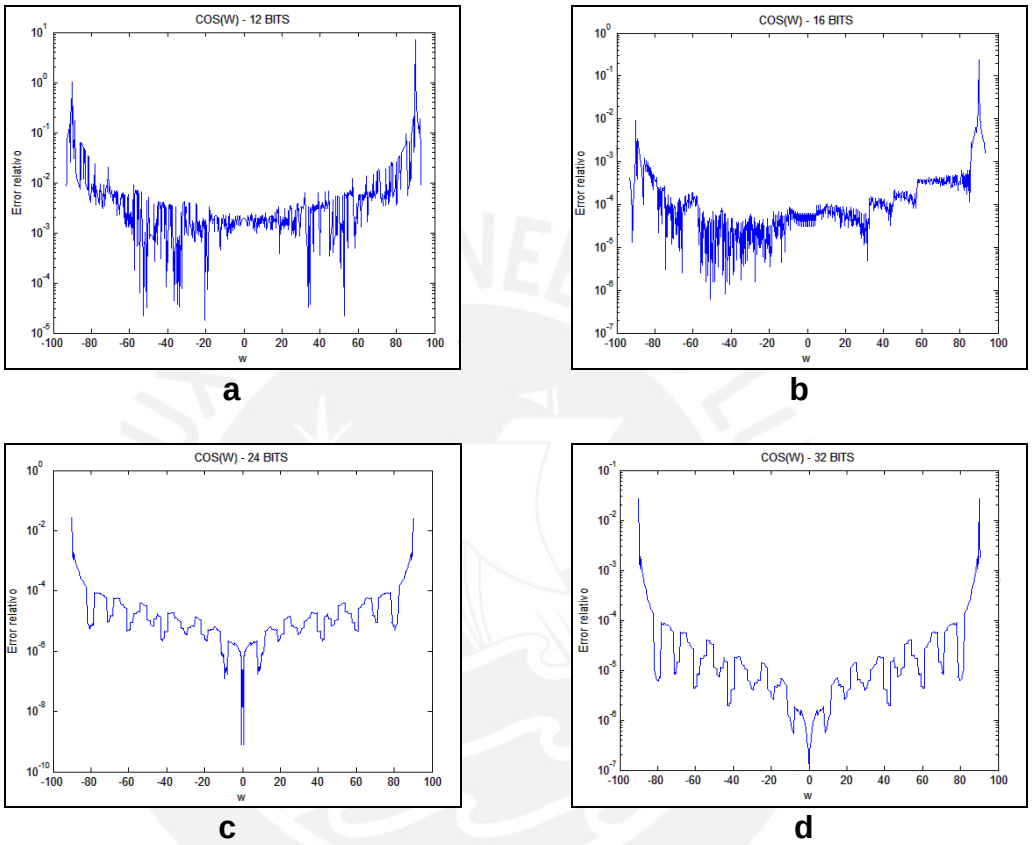



**Figura 4.2 Error Relativo para la función Coseno**

**a) 12, b) 16, c) 24 y d) 32 bits**

Como se puede apreciar, el error aumenta en los extremos de las mismas, esto se debe a que el coseno para los ángulos cercanos a ± π/2 es cero.

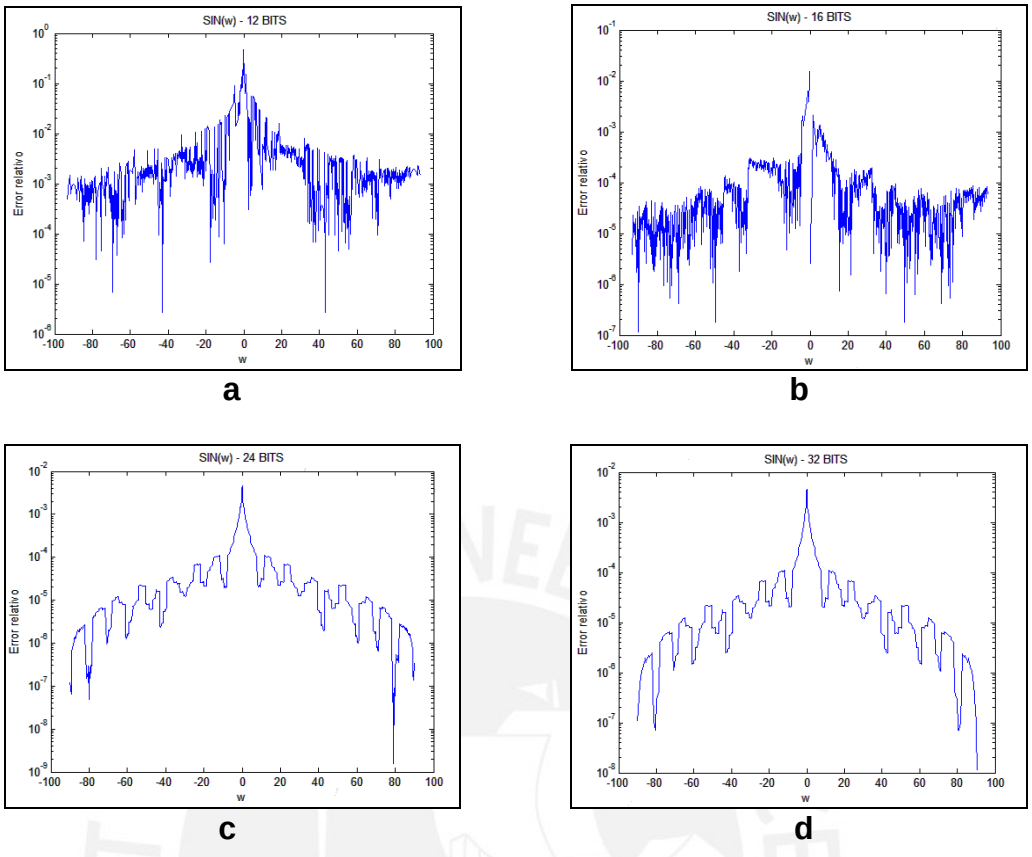



**Figura 4.3 Error Relativo para la función Seno**

**a) 12, b) 16, c) 24 y d) 32 bits**

### 4.2.2 ERROR RELATIVO PARA EL CORDIC HIPERBÓLICO.

A continuación se presenta los gráficos de error relativo de las funciones seno hiperbólico, coseno hiperbólico y arco tangente hiperbólico. Como se indico anteriormente, el error relativo aumenta para resultados cuyos valores son muy pequeños como en el caso del arco tangente hiperbólico, esto también se puede apreciar en las otras dos funciones. Se observa además que para el caso de 32 bits el error para valores pequeños aumenta en gran cantidad en comparación con el otro rango de valores implementado. Esto se debe a que solo se utilizó 8

bits de partes fraccional para aumentar el rango de convergencia; sin embargo en los resultados se observa una muy buena aproximación en los resultados.

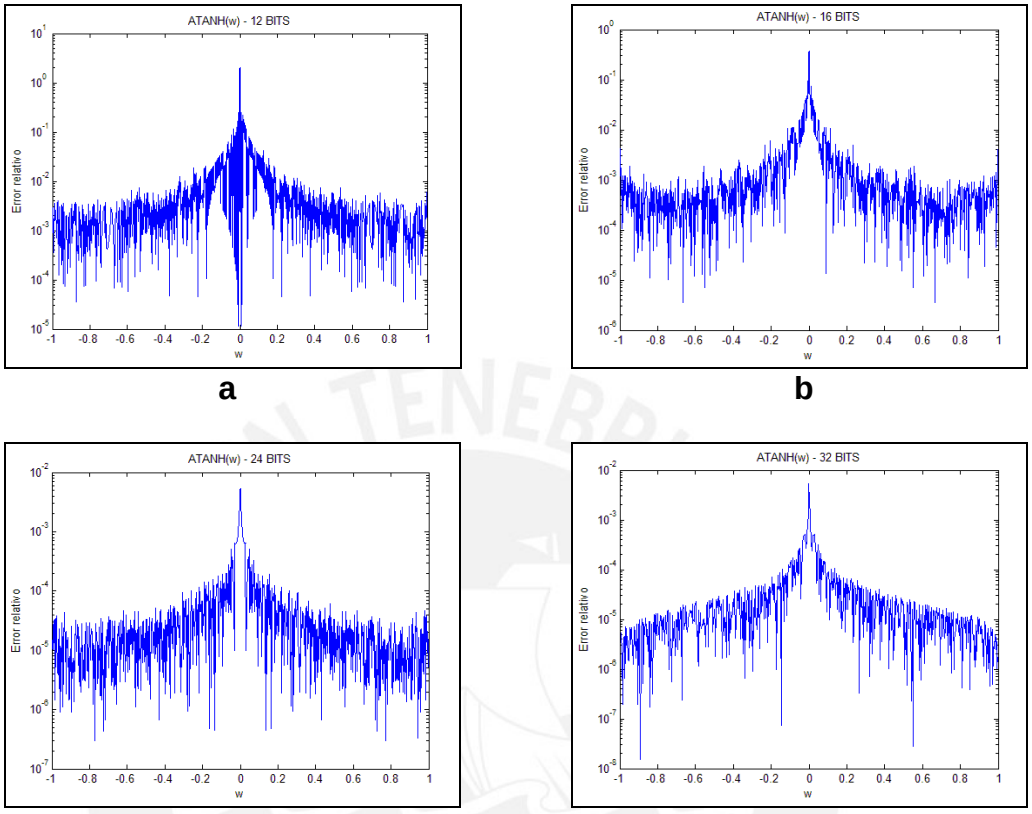
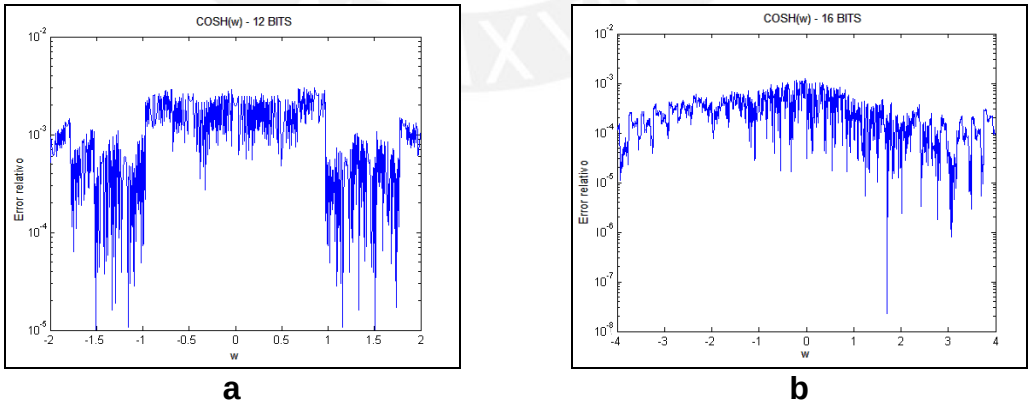
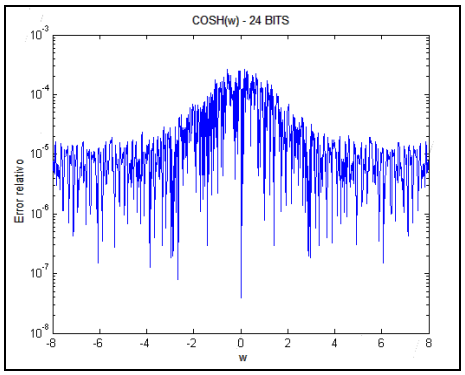
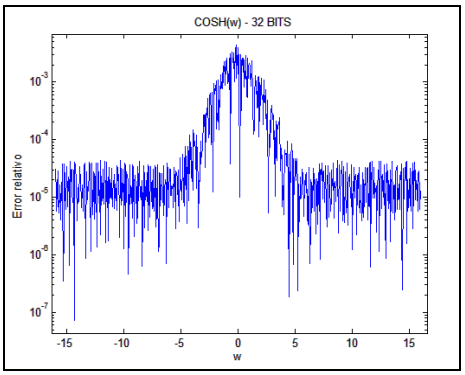



**Figura 4.4 Error Relativo para la función Arco Tangente Hiperbólico a) 12, b) 16, c) 24 y d) 32 bits**





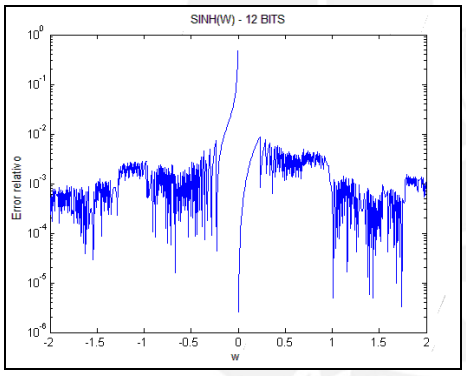

c



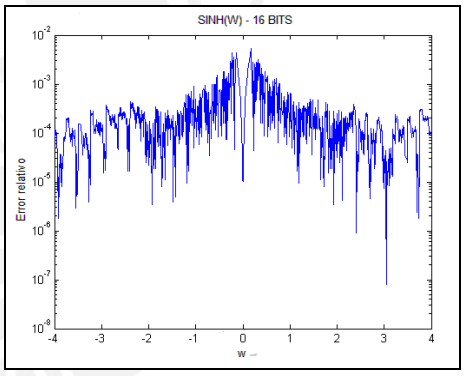

d

**Figura 4.5 Error Relativo para la función Coseno Hiperbólico**

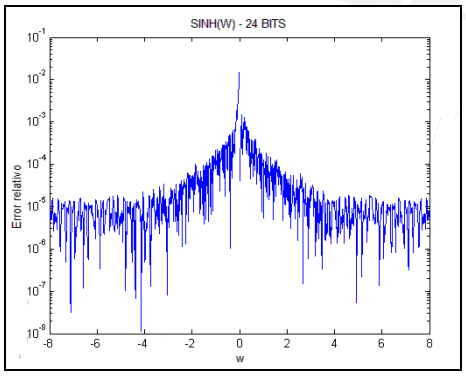
**a) 12, b) 16, c) 24 y d) 32 bits**



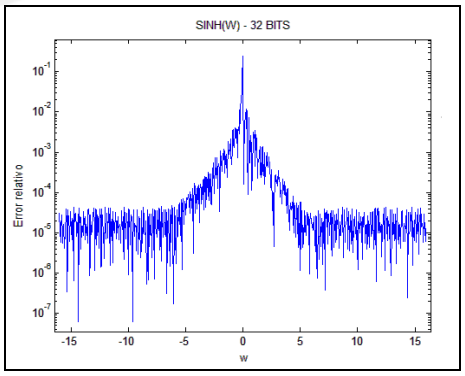

a




b




c



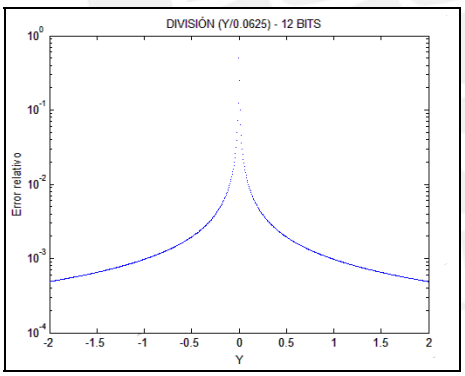

d

**Figura 4.6 Error Relativo para la función Seno Hiperbólico**

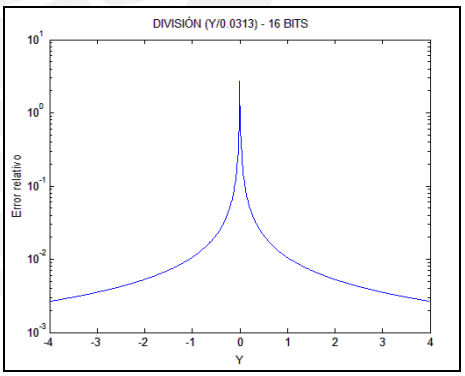
**a) 12, b) 16, c) 24 y d) 32 bits**

### 4.2.1 ERROR RELATIVO DEL CORDIC LINEAL.

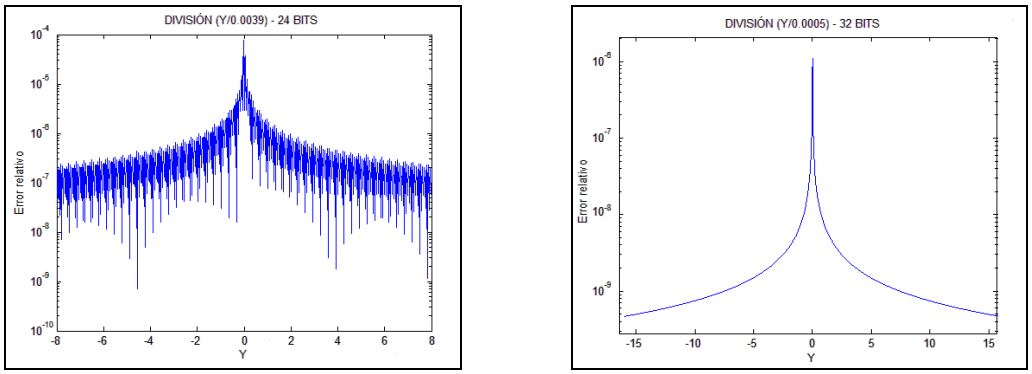
Para este caso se presentan las gráficas de error relativo para las funciones multiplicación y división. La linealidad que se observa en las figuras es propio de las funciones. El error relativo obtenido es muy bajo pero a costa de más hardware; además se observa que para el caso de la multiplicación en 32 bits se presenta el error relativo variando en un rango mayor esto se debe a que se pierde parte fraccional (ver capítulo 2).




**a**



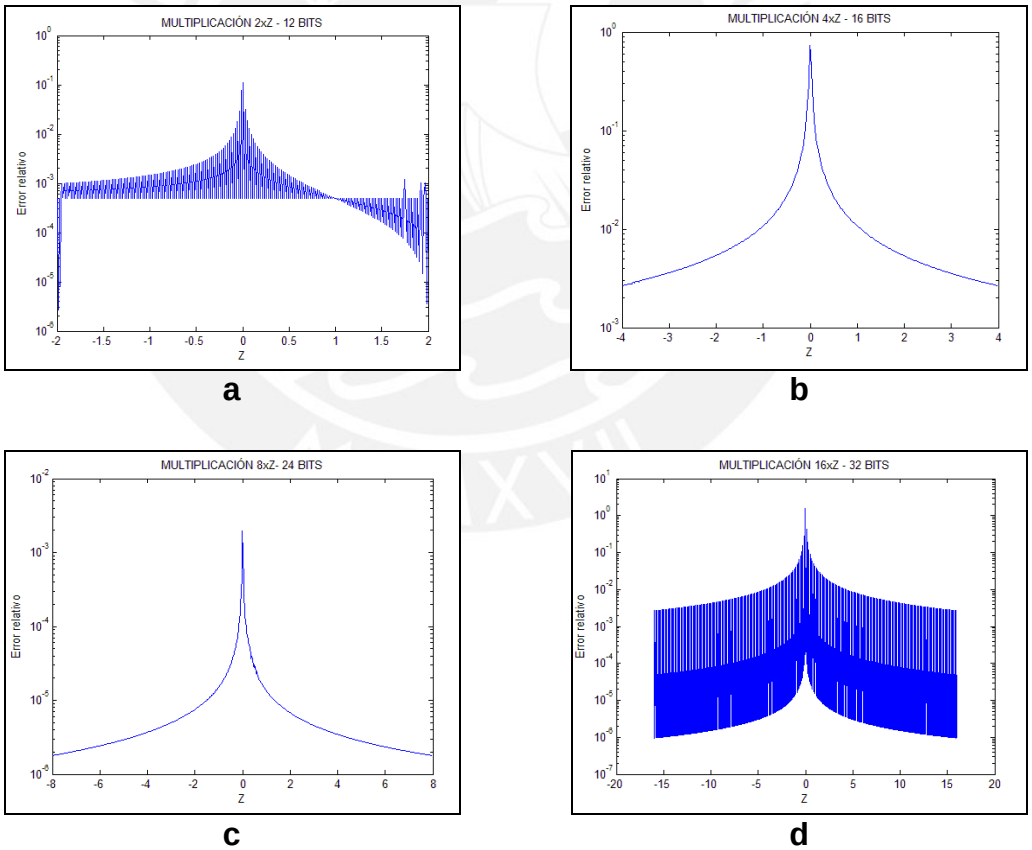

**b**




c d

**Figura 4.6 Error Relativo para la función División**

**a) 12, b) 16, c) 24 y d) 32 bits**




a b

c d

**Figura 4.8 Error Relativo para la función Multiplicación.**

**a) 12, b) 16, c) 24 y d) 32 bits**

# CONCLUSIONES

Las conclusiones de este trabajo son las siguientes:

- La expansión del rango de convergencia propuesta por Hu [3] ha probado ser eficiente para implementación en FPGAs a pesar que se utiliza hardware adicional para ello. Esto se puede observar claramente en el capítulo 4, donde los recursos de hardware y la Fmax de operación son óptimos comparados con otras implementaciones [6].

- En las gráficas de error relativo se muestran ciertas irregularidades, esto se debe al truncamiento de la parte fraccional en el procesamiento de los datos y la limitación de iteraciones especificadas al inicio. Además se debe tener en cuenta que para poder representar valores muy pequeños debemos contar con una mayor cantidad de bits para parte fraccionaria, es por esta razón que para los casos de 12 y 16 bits el error se muestra mayor para los valores pequeños.

- Como se indicó anteriormente, la base de referencia fue el programa MATLAB, el cual presenta un formato de 32 bits de punto flotante; y al comparar estos valores con los valores hallados para cada caso, siendo este trabajo presentado para trabajar con punto fijo, se han obtenido resultados óptimos. Esto se puede observar en las gráficas de error relativo, ya que se presenta un error muy lineal, sin mucha variación, el cual nos da una mayor precisión para trabajar en un rango fijo.

- Para el caso de las funciones implementadas en el sistema lineal (multiplicación y división), los resultados obtenidos son muy buenos, pero el rango es muy limitado, además consume muchos recursos lógicos y tiempo de procesamiento; es por ello que al compararse con otras implementaciones [7], no se considera adecuado utilizar este algoritmo para este tipo de funciones propuestas en el CORDIC Lineal.

- Esta implementación, comparada con otros COREs propuestos por otras compañías como XILINX [8], presenta más ventajas en ciertos aspectos como la expansión para el sistema hiperbólico y lineal lo cual disminuye el error relativo en gran cantidad; y la no limitación en los valores de las entradas, ya que se hizo el estudio necesario para todo el rango de entradas de los tipos, razón por la cual se escogió los 4 tipos de buses típicos para la implementación.

- Con el trabajo propuesto, además de los casos mencionados, se implementaron 2 funciones más: la función exponencial y el logaritmo natural, ya que estas son derivadas de las funciones hiperbólicas básicas implementadas [2]. Este trabajo se encuentra detallado en [9] [10].

# RECOMENDACIONES

- Se ha desarrollado el CORE para los 4 tipos de buses típicos, en ciertas ocasiones podría ser requerido un tamaño de bus diferente y en ese caso un trabajo futuro sería cubrir ese caso. El análisis a seguir será el mismo presentado en el capítulo 2.

- Se pueden implementar otros tipos de arquitecturas como Serial - Paralelo, con el fin de darle al usuario la posibilidad de escoger mediante un factor que tan serial o paralela será la arquitectura con la que se desea trabajar.

- Para la implementación de la mayoría de las funciones se ha utilizado la librería parametrizable de ALTERA; si se quiere transplantar a otro fabricante como XILINX, y como este no tiene esos componentes se tiene 2 opciones: la primera sería usar los propios componentes de XILINX que implica cambiar todo el código; y la segunda sería utilizar la librería funcional que provee ALTERA a XILINX, donde no se obtendrán mejores resultados ya que están optimizados para ALTERA pero funcionarán.

- Si se quiere tener más precisión en los resultados se podría incluir un análisis estadístico de error de cuantización para ver la tendencia que tendrán los datos al hacer el diseño de las arquitecturas.

- Debido a que cada caso presenta diferencias abruptas, es un desafío diseñar un CORDIC unificado; pero este sería de gran utilidad ya que en

ciertos casos se necesita los modos de procesamiento *rotation* y de *vectoring* como tal, además de poder controlar todas las funciones con un solo código [11].

# REFERENCIAS

[1] J. Volder, “The CORDIC trigonometric computing technique, IRE Transaction Electronic computers, Vol. 8, pp. 330-335, 1959

[2] J.S. Walther, “A unified algorithm for elementary functions”, in Proc. Spring Joint Comput. Conf., pp.379-385,1971

[3] X. Hu, R. Huber, S. Bass, “Expanding the Range of Convergence of the CORDIC Algorithm”, IEEE Transactions on Computers. Vol. 40, Nº 1, pp. 13-21, 1991.

[4] Y. Hu: “CORDIC-Based VLSI Architectures for Digital Signal Processing”, IEEE Signal Processing Magazine pp.16-35, 1992

[4] U. Meyer – Baese, Digital Signal Processing with Field Programmable Gate Arrays: Springer-Verlag Berlin Heidelberg, 2001.

[5] Ray Andraka, “A survey of CORDIC algorithm for FPGA based computers”,

[6]

[7]

[8] XILINX Logic CORE CORDIC V3.0, 2004.

[9] D. LLamocca, C. Agurto, “A Fixed-Point Implementation of the Expanded Hyperbolic CORDIC Algorithm”, XII WORKSHOP IBERCHIP, p.p. 179-182, 2006.

[10] D. LLamocca, C. Agurto, “A Fixed-Point Implementation of the Natural Logarithm Based on a Expanded Hyperbolic CORDIC Algorithm”, XII WORKSHOP IBERCHIP, p.p. 322 -323, 2006.

[11] Y. Hu: “The quantization Effects of the CORDIC Algorithm”, IEEE Transactions on Signal Processing Vol.40, N° 4, 1992.

[12] H. Dawid, H. Meyr: “Chapter 24- CORDIC Algorithms and Architectures” <http://www.eecs.berkeley.edu/~newton/Classes/EE290sp99/lectures/ee290aSp996_1/cordic_chap24.pdf>

[13] K. Turkowski, A. Computer, “Fixed-Point Trigonometry with CORDIC Iterations”, 1990.

[14] T. Vladivimora, H. Tiggeler, “FPGA Implementation of Sine and Cosine Generators Using the CORDIC Algorithm”, 2000.