

Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica

**Diseño e Implementación de una Unidad de Punto Flotante
FPU.**

Informe de Proyecto de Graduación para optar por el título de
Ingeniero en Electrónica con el grado académico de Licenciatura

Jeffry Ignacio Quirós Fallas

Borrador de 14 de marzo de 2016

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Jeffry Ignacio Quirós Fallas

Cartago, 14 de marzo de 2016

Céd: 1-1439-0476

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Proyecto de Graduación
Tribunal Evaluador

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

Lic. Luis Carlos Rosales
Profesor Lector

Lic. Jose Alberto Díaz García
Profesor Lector

Dr. Alfonso Chacón Rodríguez
Profesor Asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica.

Cartago, 14 de marzo de 2016

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Proyecto de Graduación
Tribunal Evaluador
Acta de Evaluación

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisito para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura, del Instituto Tecnológico de Costa Rica.

Estudiante: Jeffry Ignacio Quiróa Fallas

Nombre del Proyecto: *Diseño e Implementación de una Unidad de Punto Flotante FPU*

Miembros del Tribunal

Lic. Luis Carlos Rosales
Profesor Lector

Lic. Jose Alberto Díaz García
Profesor Lector

Dr. Alfonso Chacón Rodríguez
Profesor Asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica.

Nota final del Proyecto de Graduación: _____

Cartago, 14 de marzo de 2016

Resumen

Palabras clave: CORDIC, seno, coseno, FPU, precisión simple, precisión doble, FPGA

Abstract

Keywords: CORDIC, sine, cosine, FPU, simple precision, double precision, FPGA.

a mis queridos padres

Agradecimientos

Jeffry Ignacio Quirós Fallas

Cartago, 14 de marzo de 2016

Índice general

Índice de figuras	iii
Índice de tablas	v
Revisar	vii
1 Introducción	1
1.1 Entorno del Proyecto.	1
1.2 Descripción del problema y justificación.	2
1.3 Síntesis del problema.	2
1.4 Enfoque de la solución.	3
1.5 Meta.	3
1.6 Objetivos y estructura.	3
1.6.1 Objetivo General.	3
1.6.2 Objetivos Específicos.	3
1.6.3 Estructura de la tesis.	4
2 Marco teórico	5
2.1 Estándar IEEE 754	5
2.1.1 Formatos de precisión en punto flotante.	7
2.2 Algoritmo CORDIC para el cálculo de operaciones trigonométricas.	10
2.2.1 Fundamento Teórico.	10
2.3 Arquitecturas de implementación del algoritmo CORDIC	14
2.3.1 Arquitectura Bit-Paralela Iterativa.	14
2.3.2 Arquitectura Bit-Paralela Desplegada.	15
2.3.3 Arquitectura Bit-Serie Iterativa.	16
3 Solución propuesta	19
4 Resultados y análisis	21
5 Conclusiones	23
Bibliografía	25
A Demostración del teorema de Nyquist	27

Índice de figuras

2.1	Acomodo de bits para la representación binaria de un número en punto flotante utilizando el estándar IEEE 754-2008.	8
2.2	Ordenamiento de bits en el formato de Precisión Simple.	8
2.3	Ordenamiento de bits en el formato de Precisión Doble.	9
2.4	Vector bidimensional antes y después de ser rotado.	10
2.5	Pseudorotación de un vector bidimensional.	11
2.6	Arquitectura Bit-Paralela Iterativa para la implementación en hardware del algoritmo CORDIC	15
2.7	Arquitectura Bit-Paralela Desplegada para la implementación en hardware del algoritmo CORDIC	16
2.8	Arquitectura Bit-Serie Iterativa para la implementación en hardware del algoritmo CORDIC	17

Índice de tablas

2.1	Valores de los parámetros que definen los formatos básicos de números en punto flotante.7	6
2.2	Interpretación de patrones de bits del estándar IEEE 754 para el formato de precisión simple y doble.	9

Revisar

Capítulo 1

Introducción

1.1 Entorno del Proyecto.

El Laboratorio de Diseño de Circuitos Integrados, DCILab, es un laboratorio que es parte de la Escuela de Ingeniería Electrónica, del Instituto Tecnológico de Costa Rica, el cual está liderado por el Dr. Ing. Alfonso Chacón Rodríguez. En este laboratorio se han impulsado y desarrollado diversos proyectos, entre los cuáles se encuentra el Sistema de Reconocimiento de Patrones Acústicos, SiRPA.

El SiRPA forma parte de un proyecto de la Escuela de Ingeniería en Electrónica denominado "Sistema Electrónico Integrado en Chip (SoC)", cuya finalidad es detectar disparos de armas de fuego y de motosierras, a partir del análisis de patrones de audio en una red inalámbrica de sensores, para monitorizar zonas protegidas a lo largo del territorio nacional. El SiRPA, entonces, es básicamente la implementación algorítmica software-hardware del proceso de clasificación de señales, una vez que este sistema ha sido entrenado para reconocer señales específicas [6]. Debido a que el SiRPA se encarga de la extracción e identificación de las características de la señal, el cálculo de probabilidades y la toma de decisiones, es que se necesita la implementación una unidad de punto flotante, dado que el algoritmo matemático, Modelos Ocultos de Markov [3] [7] (HMM por sus siglas en inglés), que se escogió para la implementación del SiRPA, requiere procesar datos en punto flotante.

En el contexto de este proyecto se va a implementar un procesador de aplicación específica (ASP por sus siglas en inglés) cuya arquitectura está basada en el set de instrucciones (ISA, por sus siglas en inglés) RISC-V [11], este ISA fue desarrollado por la División de Ciencias de Computación de la Universidad de California, Berkeley. El objetivo del procesador es ejecutar el HMM, para esto es necesario que sea capaz de manejar aritmética en punto flotante. Cabe mencionar que el ASP es un proyecto de maestría a cargo del Ing. Carlos Salazar García. La implementación de la FPU dependerá meramente de adecuarse a una arquitectura que disminuya el consumo de potencia a la hora de ejecutar el HMM.

El desarrollo de la Unidad de Punto Flotante, es un proyecto que se viene implementando

desde el primer semestre del 2015, y en el que en la primera etapa del mismo, se implementaron las operaciones suma, resta y multiplicación, así como el manejo de excepciones cuando se presentan resultados que tiendan al infinito, y el redondeo. Debido a las limitantes de tiempo que conlleva el realizar un proyecto de graduación en el lapso de un semestre, y también la complejidad que conlleva el desarrollo de dicha unidad, es que se procede a continuar con su implementación.

Un aspecto importante que se debe tener en cuenta, es que la FPU que se está implementando toma como referencia el estándar IEEE 754-2008 [4] [8] y el ISA RISC-V, por lo que estos dos temas se deberán de investigar y estudiar para la realización del proyecto.

1.2 Descripción del problema y justificación.

En una primera etapa, se han diseñado e implementado algunas operaciones aritméticas como lo son la suma, resta y multiplicación, así como formatos de redondeo y excepciones de operaciones inválidas.

Dada la necesidad de desarrollar una FPU, debido a falta de presupuesto para adquirir un coprocesador numérico de este tipo, es que se procede a continuar con la implementación de esta unidad.

La FPU se utilizará para trabajar con el ASP, así como para futuros proyectos que requieran procesar aritmética de punto flotante. Es por eso que esta unidad quedará una vez finalizada, a disposición del DCILab y de la Escuela de Ingeniería en Electrónica, por lo que deberá quedar debidamente documentada, para poder ser utilizada por un usuario que requiera el desarrollo de algún proyecto con la FPU.

Al ser una continuación de un proyecto, se considera desarrollar aspectos como la optimización del timing de algunos bloques desarrollados en la etapa anterior, el manejo de la excepción NaN (Not a Number), así como también la implementación de las operaciones trigonométricas seno y coseno, además de la escalabilidad de la unidad para etapas posteriores de desarrollo.

Para la implementación de la FPU, se tomará el estándar IEEE 754-2008 como referencia para el manejo de operaciones en punto flotante, pero este no se va a implementar como tal, sino que se tomará como punto de partida para definir un estándar propio para el módulo a diseñar. Esta unidad tendrá una arquitectura de 64 bits y se diseñó para poder ejecutar instrucciones del ISA RISC-V.

1.3 Síntesis del problema.

Inexistencia de operaciones aritméticas básicas seno y coseno en la primera etapa de implementación de la FPU, así como la operación de manejo de excepción *NaN*, *Not a*

Number.

1.4 Enfoque de la solución.

En primer lugar, estudiara acerca de la aritmética de punto flotante basada en el estándar IEEE 754-2008. Seguidamente, se investigará posibles formas o algoritmos que permitan implementar las operaciones trigonométricas que se desean desarrollar.

Una vez investigado lo anterior, se procederá desarrollar los módulos utilizando una estrategia de diseño modular, a través de la cual se definirá la interfaz de entrada-salida, y la estrategia de interconexión de los bloques que constituyen las diferentes operaciones trigonométricas.

Seguidamente, los módulos se implementaran utilizando el lenguaje de descripción de hardware Verilog, y se verificarán utilizando un proceso de comparación entre los resultados obtenidos de simulaciones Post Place and Route, y los resultados obtenidos del software GNU Octave.

Por último se integrarán los módulos diseñados y desarrollados al procesador de aplicación específica, sobre el cual se ejecutarán rutinas matemáticas complejas para corroborar el funcionamiento conjunto de ambas unidades.

1.5 Meta.

Desarrollar un procesador de aplicación específica con soporte de operaciones de coma flotante en formato basado en IEEE 754-2008, compatible con la arquitectura de juego de instrucciones abierto (ISA) del proyecto RISC-V, para resolver problemas de procesamiento de sonido aplicados a la protección ambiental.

1.6 Objetivos y estructura.

1.6.1 Objetivo General.

Incorporar al mínimo dos operaciones aritméticas complejas y capacidades de tratamiento de la excepción NaN a la versión actual de unidad de coma flotante (FPU) compatible con un procesador de aplicación específica (ASP) basado en el ISA RISC-V.

1.6.2 Objetivos Específicos.

- Optimizar el temporizado de los módulos suma y multiplicación de la versión preliminar de la FPU.

- Implementar las funciones lógico-aritméticas seno y coseno basadas en el formato de representación de números flotantes IEEE 754-2008 y el ISA RISC-V.
- Implementar la excepción *NaN* de una operación aritmética inválida, basada en el formato de representación de números flotantes IEEE 754-2008 y el ISA RISC-V.

1.6.3 Estructura de la tesis.

Capítulo 2

Marco teórico

En este capítulo se presentan los conceptos en los que se basa el sistema propuesto para la solución de operaciones trigonométricas, en el formato de representación estándar de números en coma flotante, IEEE 754-2008, en concreto las operaciones seno y coseno. A continuación se brinda información acerca del formato de palabra para la representación binaria de números en punto flotante, ya sea en precisión simple o precisión doble, 32 y 64 bits respectivamente, así como para el manejo de la excepción de resultados *Not a Number*, *NaN*.

Se presenta además las bases teóricas de la rotación de vectores, fundamento teórico en el que se sustenta el algoritmo CORDIC, investigado e implementado para la resolución de las operaciones trigonométricas antes mencionadas.

2.1 Estándar IEEE 754

El estándar IEEE 754 especifica formatos y métodos para aritmética en punto flotante en sistemas computacionales, permitiendo obtener el mismo resultado, dado el mismo dato de entrada, sin importar si el proceso es llevado a cabo en hardware, software, o una combinación de ambos [1]. Para lograr esto, los diseñadores del estándar utilizaron el método de notación científica, permitiendo representar un número factorizándolo en dos partes, la magnitud y la potencia [5].

Los formatos de representación de números en punto flotante del estándar, son usados para representar un subconjunto finito de los números reales. Los formatos se caracterizan por: su base (binaria, decimal), precisión (simple, doble, extendida) y por el rango de su exponente, y cada uno puede representar un conjunto único de datos en punto flotante [1].

El estándar define 5 formatos básicos:

- Tres formatos binarios, con codificaciones en longitud de 32, 64 y 128 bits.

- Dos formatos binarios, con codificaciones en longitud de 64 y 128 bits.

Las representaciones de un dato en punto flotante en un formato consiste en:

- Triples (signo, exponente y significando); en una base b , el numero en punto flotante representado por un tripe tiene la siguiente forma:

$$(-1)^{signo} \cdot b^{exponente} \cdot significando$$

- $+\infty, -\infty$
- $qNaN, sNaN$

El conjunto finito de números en punto flotante representables dentro de un formato en particular es determinado por determinados parámetros enteros, los cuales son:

- b = la base, 2 (binaria) o 10 (decimal).
- p = el número de dígitos en el significando, es decir, a precisión.
- $emax$ = el máximo exponente.
- $emin$ = el mínimo exponente.

Donde $emin = 1 - emax$ para cada uno de los formatos.

En la Tabla 2.1 se presentan los valores de cada uno de los parámetros para cada uno de los formatos básicos, en el cuál cada formato está identificado por su base y el número de bits de su codificación.

Tabla 2.1: Valores de los parámetros que definen los formatos básicos de números en punto flotante.⁷

parametro	Formato Binario (b=2)			Formato Decimal (b=10)	
	binario(32 bits)	binario(64 bits)	binario(128 bits)	decimal64	decimal 128
p, digitos	24	53	113	16	34
$emax$	+127	+1023	+16383	+384	+6144

Sin importar el formato escogido, los siguientes datos deberán poder ser representados [1]:

- Cualquier número en punto flotante igual o diferente a cero y con signo de la forma:

$$(-1)^s \cdot b^e \cdot m$$

, donde

- s es igual a 1 o 0, donde 1 representa a los números negativos y el 0 a los números positivos.
- e corresponde a cualquier numero entero entre $emin \leq e \leq emax$.
- m es un número representado por una cadena de dígitos de la forma $d0 * d1d2...d_{p-1}$, donde d_i es un dígito entero entre $0 \leq d_i \leq b$, por lo tanto m puede ser un valor entre $0 \leq m \leq b$.
- Dos valores infinitos, $+\infty, -\infty$.
- Dos representaciones NaN , $qNaN$ y $sNaN$.

El número positivo más pequeño representable en cada formato es b^{emin} , mientras que el más grande es $b^{emax} * (b - b^{1-p})$. Los números diferentes a cero con magnitudes menores a b^{emin} son llamados subnormales, debido a que sus magnitudes caen en valores entre cero y el número normal más pequeño.

En la notación científica, los números pueden tener exponentes tanto negativos como positivos. En la representación binaria, el exponente debe ser normalizado para ser utilizado, ya que en un sistema computacional un exponente negativo debería ser almacenado utilizando una representación en complemento a 2, pero para fines de comparación entre valores, la comparación de valores en complemento a 2 es más compleja que comparar valores que no posean signo, por lo que al exponente se le suma un valor de sesgo para normalizar dicho exponente, para ubicarlo dentro de los valores positivos siempre. Este valor de sesgo tiene un valor igual a $emin$, por lo que el valor del exponente normalizado vendría dado por (2.1) [9].

$$Exponente_normalizado = Exponente - emin \quad (2.1)$$

2.1.1 Formatos de precisión en punto flotante.

Como se menciona anteriormente, en el estándar IEEE 754 existen varios formatos de representación de números en punto flotante, ya sean binarios, decimales o formatos extendidos.

Los formatos de precisión, o almacenamiento, determinan la cantidad de espacio en memoria que un valor en punto flotante necesita en memoria para ser almacenado, y son utilizados para representar un subconjunto definido finito de números reales. Cada formato se diferencia de los demás por la ubicación del punto decimal, la precisión del dato y el rango de su exponente, así como el hecho de que cada uno puede representar solo un determinado conjunto de números en punto flotante [9].

Los formatos que se desarrollan en este documento tienen una distribución definida de bits, para representar un valor en punto flotante. En la Figura 2.1 se observa el acomodo de los diferentes campos que conforman la representación según el estándar IEEE 754 [1].

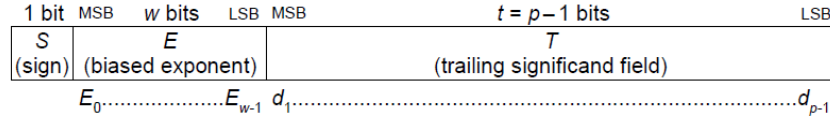


Figura 2.1: Acomodo de bits para la representación binaria de un número en punto flotante utilizando el estándar IEEE 754-2008.

En este documento solo se tratan dos formatos de precisión binarios, el formato de precisión simple y el de precisión doble, y al igual que los formatos decimales y extendidos, se caracterizan por tener un ancho de palabra definido.

Según el patrón de bits presente en la representación binaria de un número en punto flotante, así será su interpretación. En la Tabla 2.2 se presenta la interpretación de dichos patrones según los valores en los campos de la representación, ya sea para precisión simple o doble [9].

Formato de Precisión Simple.

Este formato de precisión tiene un ancho de palabra definido de 32 bits, de los cuales el campo de signo ocupa 1 bit, el bit más significativo de la cadena de bits, y puede tomar dos valores únicamente, 0 que representa a los números positivos, y 1 que representa a los números negativos. El campo de exponente ocupa los 8 bits después del bit de signo, siguiendo la dirección del MSB al LSB, por lo que el exponente puede tomar cualquier valor desde 0 a 255, donde los valores desde 0 a 127 representan los exponentes negativos, y desde 128 a 255 representan los exponentes positivos. Por último, el campo de mantisa ocupa 23 bits.

En la Figura 2.2 se muestra la distribución de bits de cada campo para el formato de precisión simple.

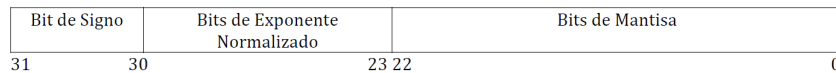


Figura 2.2: Ordenamiento de bits en el formato de Precisión Simple.

La normalización del exponente para este formato se lleva a cabo con un valor de sesgo de 127.

Formato de Precisión Doble.

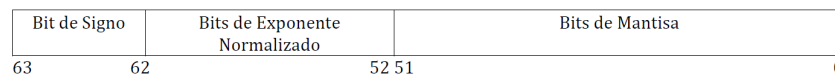
En el formato de precisión doble se tiene un ancho de palabra definido de 64 bits, de los cuales el campo de signo, al igual que en el formato de precisión simple, ocupa 1 bit, el

Tabla 2.2: Interpretación de patrones de bits del estándar IEEE 754 para el formato de precisión simple y doble.

Tipo de Número	Patrón de bits	
	Precisión Simple	Precisión Doble
Números Normales	Signo = 0 ó 1 $00_h < \text{Exponente Normalizado} < ff_h$ Fracción = Cualquier combinación de bits	Signo = 0 ó 1 $000_h < \text{Exponente Normalizado} < 7ff_h$ Fracción = Cualquier combinación de bits
Números Subnormales	Signo = 0 ó 1 Exponente Normalizado = 00_h Fracción $\neq 00000_h$	Signo = 0 ó 1 Exponente Normalizado = 000_h Fracción $\neq 0000000000000000_h$
Caso Especial Cero con Signo	Signo = 0 ó 1 Exponente Normalizado = 00_h Fracción = 000000_h	Signo = 0 ó 1 Exponente Normalizado = 000_h Fracción = 0000000000000000_h
Caso Especial Infinito Positivo	Signo = 0 Exponente Normalizado = ff_h Fracción = 00_h	Signo = 0 Exponente Normalizado = $7ff_h$ Fracción = 0000000000000000_h
Caso Especial Infinito Negativo	Signo = 1 Exponente Normalizado = ff_h Fracción = 00_h	Signo = 1 Exponente Normalizado = $7ff_h$ Fracción = 0000000000000000_h
Caso Especial Números no representables (Not a Number)	Signo = 0 ó 1 Exponente Normalizado = ff_h Fracción $\neq 00000_h$	Signo = 0 ó 1 Exponente Normalizado = $7ff_h$ Fracción $\neq 0000000000000000_h$

bit más significativo de la cadena de bits, y puede tomar dos valores únicamente, 0 que representa a los números positivos, y 1 que representa a los números negativos. El campo de exponente ocupa los once bits después del bit de signo, siguiendo la dirección del MSB al LSB.. Por último, el campo de mantisa ocupa 52 bits.

En la Figura 2.3 se muestra la distribución de bits de cada campo para el formato de precisión doble.

**Figura 2.3:** Ordenamiento de bits en el formato de Precisión Doble.

La normalización del exponente para este formato se lleva a cabo con un valor de sesgo de 1023.

2.2 Algoritmo CORDIC para el cálculo de operaciones trigonométricas.

El algoritmo de Computación Digital para Rotación de Coordenadas, CORDIC por sus siglas en inglés (**CO**ordinate **R**otation **D**igital **C**omputer), fue desarrollado especialmente para usarse en computadoras digitales en tiempo real, donde la mayoría de la computación involucraba la resolución de las relaciones trigonométricas de las ecuaciones de navegación y una alta tasa de exactitud para las relaciones trigonométricas de las transformaciones de coordenadas [10].

Este algoritmo fue originalmente propuesto por Jack Volder en el año 1959, con el propósito de calcular funciones trigonométricas mediante la rotación de vectores. La rotación de vectores es usada también para realizar la conversión de sistemas coordenados (polar a cartesiano y viceversa), y como parte de funciones matemáticas más complejas como lo son la Transformada Rápida de Fourier (FFT), y la transformada Coseno Discreta (DCT).

El CORDIC es un algoritmo iterativo para rotar vectores a ciertos ángulos determinados, que efectúa operaciones matemáticas a alta velocidad en sistemas coordenados lineales, circulares e hiperbólicos y que solo utiliza operaciones de suma y desplazamiento para efectuar operaciones trigonométricas, exponenciales, logarítmicas, y otras funciones trascendentales [12].

2.2.1 Fundamento Teórico.

El algoritmo originalmente propuesto [10], describe la rotación de un vector bidimensional en el plano cartesiano, como el mostrado en la Figura 2.4. El funcionamiento del algoritmo se deduce de la formula general de rotación de vectores [2]

$$\begin{aligned} x' &= x \cos \theta - y \sin \theta \\ y' &= y \cos \theta + x \sin \theta \end{aligned} \quad (2.2)$$

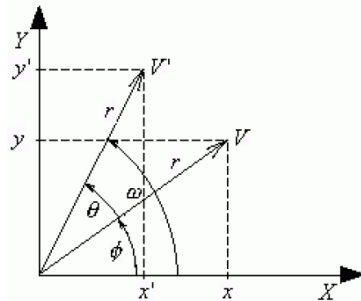


Figura 2.4: Vector bidimensional antes y después de ser rotado.

Sacando a factor común el $\cos \theta$ en la Ecuación (2.2), asumiendo que $\cos \theta \neq 0$, se obtiene

[2]

$$\begin{aligned} x' &= \cos\theta(x - y\tan\theta) \\ y' &= \cos\theta(y + x\tan\theta) \end{aligned} \quad (2.3)$$

Y tomando en cuenta que,

$$\cos\theta = \frac{1}{\sqrt{1 + \tan^2\theta}} \quad (2.4)$$

Entonces la Ecuación (2.3) se puede expresar como

$$\begin{aligned} x' &= \frac{x - y\tan\theta}{\sqrt{1 + \tan^2\theta}} \\ y' &= \frac{y + x\tan\theta}{\sqrt{1 + \tan^2\theta}} \end{aligned} \quad (2.5)$$

En el algoritmo CORDIC, las rotaciones son substituidas por pseudorotaciones vectoriales, tal y como se muestra en la Figura 2.5.

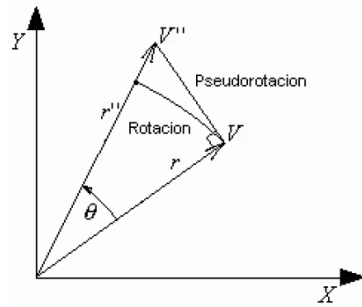


Figura 2.5: Pseudorotación de un vector bidimensional.

En una rotación real no se cambia la magnitud del vector V una vez realizada la rotación, pero en una pseudorotación esto no aplica, ya que incrementa su magnitud en

$$V'' = V\sqrt{1 + \tan^2\theta} \quad (2.6)$$

La rotación de un ángulo se puede descomponer en una sumatoria de rotaciones más pequeñas. Asumiendo inicialmente que $x = x_0$, $y = y_0$ y $z = z_0$, después de realizadas n iteraciones para una rotación real, se obtiene

$$x'_n = x\cos\theta\left(\sum_{i=0}^{n-1}\theta_i\right) - y\sin\theta\left(\sum_{i=0}^{n-1}\theta_i\right) \quad (2.7)$$

$$y'_n = y\cos\theta\left(\sum_{i=0}^{n-1}\theta_i\right) + x\sin\theta\left(\sum_{i=0}^{n-1}\theta_i\right) \quad (2.8)$$

$$z'_n = z - \left(\sum_{i=0}^{n-1} \theta_i \right) \quad (2.9)$$

En cambio para una pseudorotación se obtiene

$$x''_n = (x \cos \theta \left(\sum_{i=0}^{n-1} \theta_i \right) - y \sin \theta \left(\sum_{i=0}^{n-1} \theta_i \right)) \prod_{i=0}^{n-1} \sqrt{1 + \tan^2 \theta_i} \quad (2.10)$$

$$y''_n = (y \cos \theta \left(\sum_{i=0}^{n-1} \theta_i \right) + x \sin \theta \left(\sum_{i=0}^{n-1} \theta_i \right)) \prod_{i=0}^{n-1} \sqrt{1 + \tan^2 \theta_i} \quad (2.11)$$

$$z''_n = z - \left(\sum_{i=0}^{n-1} \theta_i \right) \quad (2.12)$$

, en donde

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

Las Ecuaciones (2.9) y (2.12) son conocidas como el parámetro *acumulador angular*, ya que incluye las rotaciones realizadas hasta el momento.

Si se restringen los ángulos de rotación de tal manera que $\tan \theta = \pm 2^{-i}$, $i \in N$, la operación multiplicación, que es tan costosa para un sistema computacional en términos de área y tiempo, se ve reducida a una operación de desplazamiento. Además, los diversos ángulos se pueden obtener al realizar una sucesión de rotaciones elementales cada vez mas pequeñas, y en cada iteración, en vez de de determinar si se debe rotar o no, lo que hace es escoger el sentido de rotación.

Tomando como punto de partida las Ecuaciones (2.3), sustituyendo $\tan \theta = \pm 2^{-i}$, y además teniendo en cuenta que $\cos(\theta) = \cos(-\theta)$, las iteraciones se pueden expresar como

$$\begin{aligned} x_{i+1} &= K_i(x_i - y_i d_i 2^{-i}) \\ y_{i+1} &= K_i(y_i + x_i d_i 2^{-i}) \end{aligned} \quad (2.13)$$

en donde $K_i = \frac{1}{\sqrt{1 + 2^{-2i}}}$ y $d_i = \pm 1$ dependiendo del sentido de rotación [2].

Esta valor K_i , debe ser multiplicado por las ecuaciones correspondientes a las pseudorotaciones, para obtener la rotación real, ya que como se menciono anteriormente, las pseudorotaciones aumentan el tamaño del vector, por lo que el valor K_i contrarrestaría este efecto.

Eliminando el valor K_i de las ecuaciones iterativas, se obtiene un algoritmo basado en sumas y desplazamientos, y factor K_i puede aplicarse al final o al principio del proceso como una constante K_n , definida como

$$K_n = \lim_{n \rightarrow \infty} \prod_{i=0}^n K_i \cong 0.607253 \quad (2.14)$$

El valor exacto de la constante K_n es determinado por la cantidad de iteraciones realizadas.

En cada paso de iteración, el algoritmo, en vez de decidir si se rota o no, lo que decide es el signo o sentido de rotación que se efectuara. por esto mismo, cada ángulo final se puede representar mediante un vector de signos, en donde cada componente corresponde a un ángulo de la secuencia de ángulos elementales. Estos ángulos se almacenan en una Look-Up Table(LUT). Dependiendo del sistema angular, se almacenan en la tabla las arcotangentes correspondientes a ese sistema. Con esta último punto, se puede modificar el acumulador angular, obteniendo

$$z_{i+1} = z_i - d_i \arctan(2^{-i}) \quad (2.15)$$

El algoritmo CORDIC puede trabajar u operar en dos modos, *rotación*(rotation) o *vectorización*(vectoring). En primero rota el vector de entrada un ángulo específico que se introduce como parámetro. El segundo modo rota el vector de entrada hacia el eje X, acumulando el ángulo necesario para efectuar dicha rotación.

En el caso del modo de *rotación*, el acumulador angular es inicializado con el ángulo a rotar, y la decisión sobre el sentido de rotación en cada iteración, se realiza para minimizar la magnitud del ángulo acumulado, hasta llegar a cero, con lo que el signo que determina el sentido de rotación, se obtiene del valor del acumulador angular en cada iteración [12].

Para el modo de *rotación*, las ecuaciones iterativas son

$$\begin{aligned} x_{i+1} &= x_i - y_i d_i 2^{-i} \\ y_{i+1} &= y_i + x_i d_i 2^{-i} \\ z_{i+1} &= z_i - d_i \arctan(2^{-i}) \end{aligned} \quad (2.16)$$

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

Para el modo de *vectorización*, el ángulo que ingresa se rota hasta quedar alineado con el eje X. Este resultado se logra minimizando la magnitud del componente y , en vez de la magnitud del acumulador angular. a diferencia del modo de *rotación*, se utiliza el signo de la variable y para definir la dirección de rotación de la siguiente iteración. Si se inicia el acumulador angular con el valor cero, al finalizar el proceso esta variable contendrá el ángulo de rotación adecuado.

Para el modo de *vectorización*, las ecuaciones iterativas son

$$\begin{aligned}
x_{i+1} &= x_i - y_i d_i 2^{-i} \\
y_{i+1} &= y_i + x_i d_i 2^{-i} \\
z_{i+1} &= z_i - d_i \arctan(2^{-i})
\end{aligned} \tag{2.17}$$

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

Cabe destacar que el algoritmo CORDIC está restringido a rango de ángulos $\frac{-\pi}{2} \leq \theta \leq \frac{\pi}{2}$, esto debido a que los ángulos elementales convergen sólo dentro de este rango, y para tratar ángulos por fuera de ese rango, estos se deben de reducir al primer o cuarto cuadrante.

2.3 Arquitecturas de implementación del algoritmo CORDIC.

2.3.1 Arquitectura Bit-Paralela Iterativa.

La arquitectura bit-paralela es una de las más utilizadas a la hora de implementar el algoritmo CORDIC en hardware. Se denomina paralela por la forma en que se opera con las componentes X , Y y Z . Cada etapa de esta implementación consiste en un registro para almacenar los valores de salida, una unidad del desplazamiento y un sumador algebraico.

Cuando se inicia el cálculo, los valores iniciales para x_0, y_0 y z_0 ingresan de manera paralela a los registros a través de los multiplexores de 2x1. El bit más significativo de la variable X o Y , dependiendo del modo, en cada paso iterativo determina la operación a efectuar por el sumador. Seguidamente, las señales de las variables X y Y son desplazadas, y luego sumadas o restadas a las mismas señales sin desplazar, correspondiente a la variable opuesta.

La variable Z suma o resta los valores almacenados en el registro con los valores almacenados en una Look-Up Table, (LUT), de arcotangentes precalculadas con una cantidad de entradas proporcional a la cantidad de iteraciones. Dicha LUT puede ser implementada en una memoria ROM. Cuando se alcanza la iteración n , el valor a calcular se puede obtener en la salida.

Esta arquitectura tiene como ventaja el uso eficiente de hardware, debido a que los recursos son reutilizados en cada iteración. En la Figura 2.6 se muestra el diagrama correspondiente a la arquitectura antes descrita. A esta arquitectura además, se le debe implementar una máquina de control que controle el flujo de los datos, además de controlar las iteraciones realizadas, el canal de mux a activar, entre otras cosas.

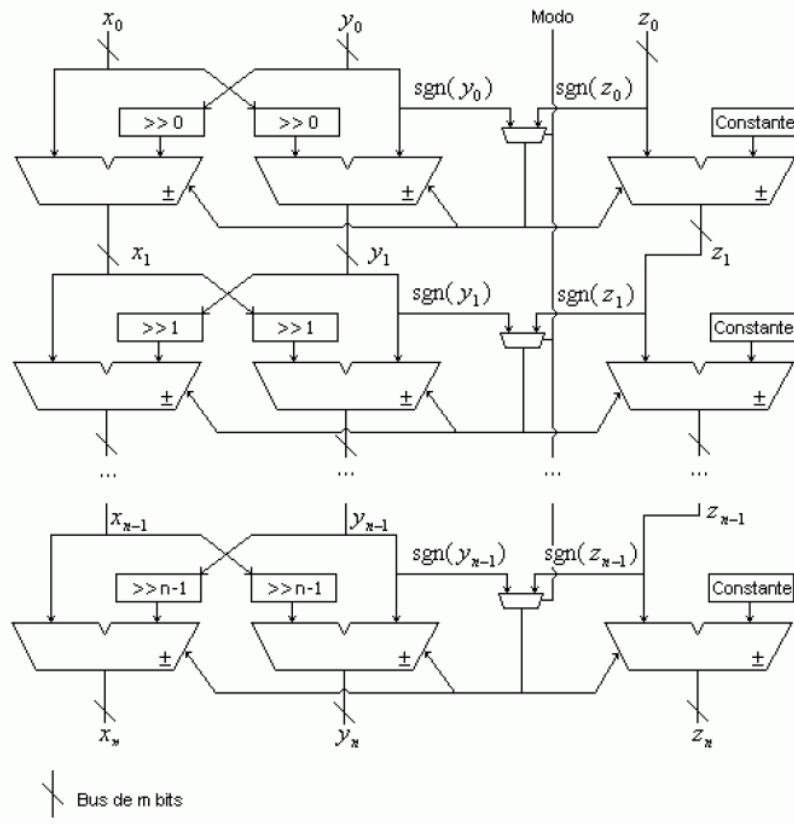


Figura 2.7: Arquitectura Bit-Paralela Desplegada para la implementación en hardware del algoritmo **CORDIC**.

2.3.3 Arquitectura Bit-Serie Iterativa.

En esta arquitectura, se procesa un bit a la vez, con lo que las conexiones se reducen a un bit de ancho. En este diseño, cada etapa del algoritmo está conformada por tres multiplexores, dos registros de desplazamiento y 3 sumadores algebraico bit-serie. Estos sumadores se implementan como un sumador completo, en el que la resta se lleva a cabo sumando el complemento a dos del valor a restar.

El desempeño de esta arquitectura puede ser calculado como

$$\delta = \frac{f}{n \cdot W} \quad (2.18)$$

,donde f es la frecuencia de reloj del sistema, n la cantidad de iteraciones a realizar y W el ancho de palabra. En la Figura 2.8 se observa el esquema de esta arquitectura.

La ventaja que presenta esta arquitectura es que los buses de interconexión son del ancho de un bit y que el registro de desplazamiento está integrado al registro intermedio que se utilizaba en la arquitectura Bit-Paralela Iterativa, lo que minimiza el espacio a la hora de la implementación, en comparación con dicha arquitectura.

Pero por otro lado presenta dos grandes desventajas. Primero que introduce un retardo

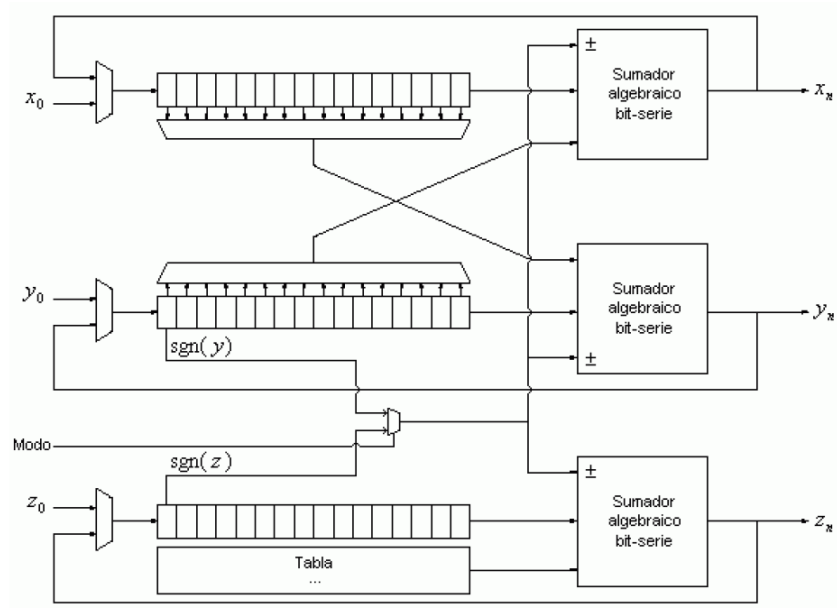


Figura 2.8: Arquitectura Bit-Serie Iterativa para la implementación en hardware del algoritmo **CORDIC**.

proporcional al ancho de palabra en cada etapa, esto ocasionado por los registros de desplazamiento, y segundo, es que esta arquitectura requeriría una maquina de control más compleja que la que se necesita en la arquitectura Bit-Paralela Iterativa.

Capítulo 3

Solución propuesta

Capítulo 4

Resultados y análisis

Capítulo 5

Conclusiones

Bibliografía

- [1] Ieee standard for floating-point arithmetic. *IEEE Std 754-2008*, pages 13–21, Aug 2008. 5, 6, 8
- [2] T. Adiono and R. S. Purba. Scalable pipelined cordic architecture design and implementation in fpga. In *Electrical Engineering and Informatics, 2009. ICEEI '09. International Conference on*, volume 02, pages 646–649, 2009. ID: 1. 10, 11, 12
- [3] Matthew Butler. Hidden markov model clustering of acoustic data [online]. 2003 [visitado el 15 de Octubre del 2015]. URL <http://www.inf.ed.ac.uk/publications/thesis/online/IM030057.pdf>. 1
- [4] ORACLE Help Center. Chapter 2. ieee arithmetic [online]. 2015 [visitado el 14 de marzo de 2016]. URL http://docs.oracle.com/cd/E19957-01/806-3568/ncg_math.html. 2
- [5] The Oxford Math Center. Ieee 754 format [online]. 2015 [visitado el 14 de marzo de 2016]. URL <http://www.oxfordmathcenter.com/drupal7/node/43>. 5
- [6] Alvarado Moya J.P Chacón Rodríguez, A. Sistema electrónico integrado en chip (soc) para el reconocimiento de patrones de disparos y motosierras en una red inalámbrica de sensores para la protección ambiental (documento i) [online]. 2014 [visitado el 11 de Octubre del 2015]. URL <http://www.ie.itcr.ac.cr/achacon/VIE/sirpa-doc1.pdf>. 1
- [7] Z Ghahramani. An introduction to hidden markov models and bayesian networks [online]. 2001 [visitado el 13 de Octubre del 2015]. URL <http://mlg.eng.cam.ac.uk/zoubin/papers/ijprai.pdf>. 1
- [8] W Kahan. Ieee standard 754 for binary floating-point arithmetic [online]. 1997 [visitado el 15 de Octubre del 2015]. URL <http://www.eecs.berkeley.edu/wkahan/ieee754status/IEEE754.pdf>. 2
- [9] Diego Andrés Rodríguez Valverde. Diseño e implementación de una unidad aritmético-lógica de coma flotante para un procesador de aplicación específica., 2016. 7, 8
- [10] J. E. Volder. The cordic trigonometric computing technique. *IRE Transactions on Electronic Computers*, EC-8(3):330–334, 1959. 10

- [11] Lee Y. Patterson D. Asanovic K Waterman, A. The risc-v instruction set manual, volume i: User- level isa, version 2.0 [online]. 2014 [visitado el 10 de Octubre del 2015]. URL <http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-54.pdf>.
1
- [12] Jie Zhou, Yong Dou, Yuanwu Lei, Jinbo Xu, and Yazhuo Dong. Double precision hybrid-mode floating-point fpga cordic co-processor. In *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, pages 182–189, 2008. ID: 1. 10, 13

Apéndice A

Demostración del teorema de Nyquist

