DESDE 2013
https://repository.uaeh.edu.mx/revistas/index.php/icbi/issue/archive
Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI

ISSN: 2007-6363

Publicación Semestral Pädi Vol. 11 No. Especial 4 (2023) 213-220

Implementación y evaluación de la eficiencia del algoritmo cordic en c y fpga mediante vhdl

Implementation and evaluation of the efficiency of the cordic algorithm in c and fpga using vhdl

L. G. Enríquez-Pérez[a], A. Traslosheros-Michel[a], G. Ramírez-Villa[a], M. Torres-Rivera[a], F. M. Zavala-Ponce[b]

[a] *Ingeniería en Electrónica y Control de Sistemas de Aeronaves, Universidad Aeronáutica en Querétaro, Carr. Querétaro-Tequisquiapan 22154, 76278 Santiago de Querétaro, Qro, México.*
[b] *Universidad Autónoma de Querétaro, Miguel Hidalgo S/N, Col. Las Campanas, 76010, Santiago de Querétaro, Qro, México.*

Resumen

El enfoque de la tecnología hacia la mejora del hardware dejando de lado la optimización del software es más notorio día con día. Este proyecto busca encontrar una forma eficiente, en cuanto al consumo de recursos y velocidad, para el cálculo de funciones trigonométricas sencillas. En la primera etapa, se realiza una revisión exhaustiva de la literatura para comprender las implementaciones del algoritmo CORDIC en diversas plataformas, centrándose en la variante que se implementará (CORDIC en su Modo Rotacional). En la segunda etapa, se lleva a cabo una implementación en Leguaje C para comprender el funcionamiento y las limitaciones del algoritmo, evaluando la precisión, velocidad y consumo de recursos. En la tercera etapa, se implementa CORDIC en FPGA utilizando el software Quartus y el lenguaje VHDL. En la cuarta etapa, se evalúan los resultados obtenidos y se comparan con la implementación en Leguaje C, analizando la precisión, velocidad y consumo de recursos. Por último, se discuten las conclusiones y se brindan recomendaciones para futuras investigaciones y aplicaciones del algoritmo CORDIC en FPGA.

Palabras Clave: Algoritmo CORDIC, FPGA, VHDL, Resolución, Corrimiento de bits.

Abstract

The focus of technology towards hardware improvement, neglecting software optimization, is becoming more noticeable day by day. This project aims to find an efficient way, in terms of resource consumption and speed, for calculating simple trigonometric functions. In the first stage, a comprehensive review of the literature is conducted to understand the implementations of the CORDIC algorithm on various platforms, focusing on the variant that will be implemented (Rotational Mode CORDIC). In the second stage, a implementation is carried out in language C to understand the functioning and limitations of the algorithm, evaluating precision, speed, and resource consumption. In the third stage, CORDIC is implemented on an FPGA using Quartus software and the VHDL language. In the fourth stage, the obtained results are evaluated and compared with the implementation in language C, analyzing precision, speed, and resource consumption. Finally, conclusions are discussed, and recommendations are provided for future research and applications of the CORDIC algorithm on FPGA.

Keywords: CORDIC algorithm, FPGA, VHDL, Resolution, Bit shifting.

1. Introducción

1.1. Antecedentes y motivación.

Durante muchos años, el procesamiento de señales digitales ha estado dominado por el uso de microprocesadores, lo que ha llevado a innovaciones como instrucciones acumulativas de un solo ciclo y modos de direccionamiento especiales. Sin embargo, estos procesadores económicos y flexibles a menudo no cumplen con los requisitos de velocidad para tareas de procesamiento de señales digitales (DSP) altamente exigentes (E. Deprettere, P. Dewilde, and R. Udo, 1984) (Y. Hu and S. Naganathan, 1990).

*Autor para la correspondencia: enriquezperezluisgustavo@gmail.com
Correo electrónico: enriquezperezluisgustavo@gmail.com (Luis Gustavo Enríquez-Pérez), alberto.traslosheros@unaq.mx (Alberto Traslosheros Michel), goretti.ramirez@unaq.mx (Goretti Ramírez Villa), moises.torres@unaq.mx (Moises Torres Rivera), magalhy_zp@hotmail.com (Flor G. Magalhy Zavala Ponce).

Historial del manuscrito: recibido el 02/07/2023, última versión-revisada recibida el 13/09/2023, aceptado el 09/09/2023, publicado el 30/11/2023. **DOI:** https://doi.org/10.29057/icbi.v11iEspecial4.11385

La llegada de la computación lógica reconfigurable abrió nuevas posibilidades con soluciones de hardware dedicadas, más rápidas y beneficiosas que los enfoques de software tradicionales. Dichos beneficios van desde el bajo consumo de recursos hasta el ahorro de energía. Lamentablemente, la mayoría de los algoritmos diseñados para software no consideran el hardware, lo que resulta en un rendimiento deficiente en esta área. A pesar de que existen soluciones altamente eficientes en hardware, que son diseñados con el objetivo de funcionar de manera óptima y eficiente con la menor cantidad de recursos posible, han pasado desapercibidas debido al predominio del enfoque en el software (E. Deprettere, P. Dewilde, and R. Udo, 1984). Entre estas soluciones eficientes en hardware, hay algoritmos iterativos para el cálculo de funciones trigonométricas y otras trascendentales, que usan operaciones de desplazamiento y complemento, eliminando así la necesidad de hardware para la multiplicación y reduciendo la capacidad de cómputo requerida. El algoritmo trigonométrico más conocido es el CORDIC (Computadora Digital de Rotación de Coordenadas) (Ahmed, Delosme, and Morf, 1982), que se basa en la rotación de vectores para calcular las funciones trigonométricas y utiliza una expresión incremental para calcular la raíz cuadrada. Los algoritmos CORDIC, al ser una solución enfocada al hardware, requieren de mínimos recursos para su ejecución pues solo necesitan de hardware capaz de realizar operaciones sencillas de adición y corrimiento de bits, además de poder adaptarse a la aplicación requerida, pues otra de sus grandes ventajas es la precisión configurable, limitada a la capacidad del hardware. (E. Deprettere, P. Dewilde, and R. Udo, 1984).

El algoritmo CORDIC se desarrolló originalmente como una solución digital para problemas de navegación en tiempo real y se atribuye a Jack Volder. (E. Deprettere, P. Dewilde, and R. Udo, 1984) (Y. Hu and S. Naganathan, 1990). A lo largo de los años, se han realizado extensiones y mejoras en la teoría CORDIC, basadas en el trabajo de John Walther (Ahmed, Delosme, and Morf, 1982) y otros, lo que ha permitido su aplicación en una amplia gama de funciones. El algoritmo CORDIC se ha utilizado en diversas aplicaciones, como el coprocesador matemático 8087 (J. Duprat and J.-M. Muller, 1993), la calculadora HP-35 y la robótica. También se ha propuesto su uso en el cálculo de transformadas discretas de Fourier, transformadas discretas de coseno, transformadas discretas de Hartley, transformadas Chirp-Z, filtrado y resolución de sistemas lineales (E. Deprettere, P. Dewilde, and R. Udo, 1984) (C. J. León, J. Lanchares, C. Villanueva, and J. L. Valenzuela, 2017) (Y. Hu and S. Naganathan, 1990) (Ahmed, Delosme, and Morf, 1982).

El objetivo del trabajo presentado en este artículo es analizar los algoritmos CORDIC, así como sus variantes y algoritmos similares, con una implementación en FPGA. Primero, se da una breve descripción de la teoría detrás del algoritmo y algunas variantes, seguido de la implementación de dicho algoritmo en lenguaje C como antecedente para la implementación del mismo algoritmo en FPGA.

1.2. Justificación

La tendencia actual al procesamiento intensivo de señales de hardware ha descubierto una relativa incomprensión de las arquitecturas de procesamiento de señales de hardware. Existen muchos algoritmos eficientes en hardware, pero no son muy conocidos y raramente implementados por la poca atención que se les daba a las soluciones orientadas al hardware hasta hace años. Entre estos, se encuentra un conjunto conocido como shift-add colectivamente llamados CORDIC, para calcular una amplia gama de funciones, incluidas ciertas funciones trigonométricas, hiperbólicas, lineales y logarítmicas. Sin embargo, aunque dichas soluciones existen desde mucho tiempo atrás y hay muchos artículos en los que se habla de ellas, casi ninguno de estos artículos realmente profundiza en el algoritmo CORDIC ni en su implementación en FPGA para el procesamiento intensivo de señales.

Debido a ello, en este documento se busca profundizar en el entendimiento del algoritmo CORDIC, analizar su funcionamiento en sus distintas variantes y con ello llegar hasta la implementación de este en FPGA. No obstante, para llegar a su implementación, también será necesario el análisis de la arquitectura sobre la cual este se implementará.

2. Marco teórico

Con el objetivo de presentar al lector los fundamentos matemáticos y principios de funcionamiento del algoritmo CORDIC, se presenta a continuación una breve síntesis de la teoría detrás de dicho algoritmo:

2.1. Fundamentos matemáticos del algoritmo CORDIC

Todas las funciones trigonométricas se pueden calcular o derivar de funciones usando rotaciones vectoriales. La rotación vectorial también se puede utilizar para conversiones de polar a rectangular y de rectangular a polar, para magnitud vectorial y como componente básico en ciertas transformaciones, como la DFT y la DCT. El algoritmo CORDIC proporciona un método iterativo para realizar rotaciones de vectores en ángulos arbitrarios usando solo corrimiento de bits y adiciones. El algoritmo, acreditado a Volder, se deriva de la transformación de rotación general (Ilustración 1), donde x' , y' representan las coordenadas del vector rotado, ϕ el ángulo al que se desea rotar el vector y x , y las coordenadas iniciales.

$$x' = x\ cos\ cos\ \phi\ - y\ sin\ sin\ \phi, \quad (1)$$

$$y' = y\ cos\ cos\ \phi\ + x\ sin\ sin\ \phi. \quad (2)$$

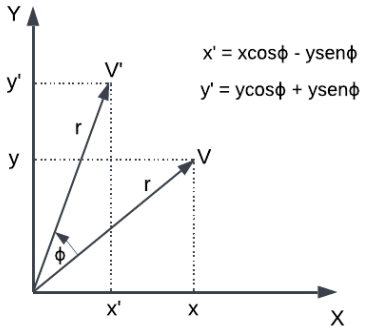

Ilustración 1:Rotación de vectores.

Dicha rotación gira un vector en un plano cartesiano el ángulo ϕ. Estos se pueden reorganizar de modo que:

$$x' = cos\ cos\ \phi\ \cdot [x - y\ tan\ tan\ \phi\ ], \quad (3)$$

$$y' = cos\ cos\ \phi\ \cdot [y + x\ tan\ tan\ \phi\ ]. \quad (4)$$

Hasta ahora, nada se simplifica. Sin embargo, si los ángulos de rotación se restringen de modo que $tan(\theta^i) = \pm 2^{-i}$, donde θ^i representa los ángulos que cumplen esta condición, entonces, la multiplicación por el término tangente se reduce a una simple operación de desplazamiento. Se pueden obtener ángulos arbitrarios de rotación con rotaciones elementales sucesivamente más pequeñas. Dichos ángulos son nombrados comúnmente como: "ángulos especiales" y se muestran a continuación en la Tabla 1.

Tabla 1: Ángulos especiales.

i	$\theta^i(Grados)$	$tan(\theta^i) = \pm 2^{-i}$
0	45.0	1
1	26.55505	0.5
2	14.03624	0.25
3	7.12501	0.125
4	3.57633	0.0625

Si la decisión en cada iteración, i, es en qué dirección rotar en lugar de rotar o no, entonces el término $cos(\theta)$ se convierte en una constante (porque $cos(\theta) = cos(-\theta)$). La rotación iterativa ahora se puede expresar como:

$$x_{i+1} = K_i[x_i - y_i \cdot d_i \cdot 2^{-i}], \quad (5)$$

$$y_{i+1} = K_i[y_i + x_i \cdot d_i \cdot 2^{-i}], \quad (6)$$

donde:

$$K_i = cos\ cos\ (2^{-i}) = 1/\sqrt{1 + 2^{-2i}}, \quad (7)$$

$$d_i = \pm 1, \quad (8)$$

por lo tanto, Ki es una constante que se puede conocer, dependiendo del número de iteraciones cuando este tiende a infinito, Ki = 0.6073. Al final, se obtienen las siguientes ecuaciones iterativas para el algoritmo, las dos planteadas anterior mente más una tercera que sirve simplemente como un acumulador del ángulo rotado en cada iteración:

$$x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i}, \quad (9)$$

$$y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i}, \quad (10)$$

$$z_{i+1} = z_i - d_i \cdot (2^{-i}), \quad (11)$$

donde: $d_i = -1\ si\ \ z_i < 0, de\ lo\ contrario\ d_i = 1$.

Esta expresión describe el funcionamiento del cambio de sentido de la rotación, si el ángulo acumulado es menor que 0 la rotación el sentido de la rotación es positivo y se suma el ángulo especial actual al acumulador, de ser positiva, ocurre la operación contraria.

De la ejecución de estas ecuaciones se obtiene:

$$x_n = A_n[x_0\ cos\ cos\ z_0\ - y_0\ sin\ sin\ z_0\ ], \quad (12)$$

$$y_n = A_n[y_0\ cos\ cos\ z_0\ + x_0\ sin\ sin\ z_0\ ], \quad (13)$$

$$z_n = 0, \quad (14)$$

donde:

$$A_n = \prod_n \quad \sqrt{1 + 2^{-2i}}. \quad (15)$$

Como se puede notar, el vector resultado de la aplicación del algoritmo tiene una ganancia A_n que tiende a 1.6747, el inverso de Ki:

$$x_n = A_n \cdot x_0\ cos\ cos\ z_0\ , \quad (16)$$

$$y_n = A_n \cdot x_0\ sin\ sin\ z_0\ . \quad (17)$$

Por lo tanto, si se inicia el algoritmo con un vector (Ki,0), es posible obtener directamente el valor de seno y coseno del ángulo ingresado al algoritmo, como se muestra a continuación:

$$x_n = A_n \cdot \tfrac{1}{A_n} cos\ cos\ z_0\ = cos\ cos\ z_0\ , \quad (18)$$

$$y_n = A_n \cdot \tfrac{1}{A_n} sin\ sin\ z_0\ = sin\ sin\ z_0\ . \quad (19)$$

3. Implementación del algoritmo CORDIC en Lenguaje C como antecedente para la implementación en FPGA.

Especificaciones del hardware:

Para esta implementación se usó un ordenador portátil de uso cotidiano con las siguientes especificaciones:

Procesador

Intel® CoreTM i5

Memoria

SDRAM DDR3 de 1600 MHz 4GB

SDD 256 GB

Sistema operativo

Windows 10

Implementación:

Como parte introductoria al desarrollo del proyecto, se realizó la implementación del algoritmo CORDIC en su forma rotacional para el cálculo de las funciones seno y coseno. Esto con la finalidad de familiarizarse con el algoritmo y facilitar su implementación en el lenguaje VHDL y además poder comparar dichas implementaciones en cuanto a rendimiento en tiempo.

Tras realizar una profunda investigación dentro de distintos artículos científicos de acceso público y algunos de acceso privado, se llegó a la conclusión de que el algoritmo CORDIC ha sido muy poco implementado y las pocas implementaciones disponibles a todo el público en internet, son implementaciones incompletas e imperfectas del algoritmo (Y. H. Hu and S. Naganathan, 1993). Este comentario se sustenta en que el punto clave de este algoritmo radica en reducir el cálculo de funciones mediante únicamente operaciones de corrimiento de bits y adición sin embargo en todas las implementaciones encontradas durante la investigación, se hacía uso de recursos como multiplicaciones, números de punto flotante e incluso herramientas de la librería math.h.

Por ello, en este primer paso del proyecto, se busca implementar de forma correcta el algoritmo, cuidando minuciosamente todos estos detalles, para que la implementación sea correcta y cumpla con los puntos fuertes del algoritmo, que son el no requerir hardware para realizar operaciones complejas y no necesitar una gran capacidad de cómputo.

A continuación, se muestra una breve descripción de las partes principales del código de la implementación del algoritmo CORDIC.

Primero tenemos una función que inicializa los ángulos especiales sobre los cuales haremos nuestras rotaciones, además de definir nuestros cuadrantes e inicializar X como el inverso de la ganancia: A_n.

```
1.  void SinCosSetup(void)
2.
3.  {
4.    int i;       // to index ArcTan[]
5.    double f;    // to calc initial x projection
6.
7.    CordicBase = 1 << NBITS;
8.    HalfBase = CordicBase >> 1;
```

```
  Quad2Boundary = CordicBase << 1;
  Quad3Boundary = CordicBase + Quad2Boundary;

//soy CordicBase: 16384
//soy HalfBase: 8192
//soy Quad2Boundary: 32768
//soy Quad3Boundary: 49152


 // ANGULOS ESPECIALES //

 for (i = 1; i <= NBITS; i++)
 {
  ArcTan[i] = CordicBase >> i;
 }

 /* xInit is initial value of x projection to comp-
  * ensate for expansions.  f = 1/sqrt(2/1 * 5/4 * ...
  * Normalize as an NBITS binary fraction (multiply by
  * CordicBase) and store in xInit. Get f = 0.607253
  * and xInit = 9949 = 0x26DD for NBITS = 14.
  */

 ///////// SE USA DIRECTAMENTE EL VALOR DE LA CONSTANTE Y LA EXPRESAMOS COMO CAU ////////////

 //f = 0.6073;
 ///inicializamos x0 = 0.6073 en unidades cordic, como las unidades cordic representan grados, el cos^-1(0.6073) en
 /// unidades cordic es el valor en el que debemos iniciar x0
 xInit = 9577;

  printf ("soy xinit: %d\n", xInit);


}
```

A continuación, se muestra la segunda parte del código, donde se observa la función que se encargará de hacer las rotaciones, en está implementación solo aceptan ángulos entre 0° y 90°. Además, se puede observar la definición de las variables necesarias para realizar las rotaciones. Al ser la encargada de realizar las rotaciones vectoriales, se convierte en la función más importante dentro de esta implementación.

```
void SinCos(WORD theta, int *sine, int *cosine)

{
 int quadrant;  // quadrant of incoming angle
 int z;         // incoming angle moved to 1st quad
 int i;         // to index rotations : one per bit
 int x, y;      // projections onto axes
 int x1, y1;    // projections of rotated vector


 if (theta < CordicBase)
 {
  quadrant = QUAD1;
  z = (int) theta;
 }
 else
 {
  printf ("rotación fuera del primer cuadrante");
  return (-1);
 }

 x = xInit;
 y = 0;

 /* Negate z, so same rotations taking angle z to 0
  * will take (x, y) = (xInit, 0) to (*cosine, *sine).
  */
 z = -z;

 // Rotate NBITS times.
 for (i = 0; i < NBITS; i++)
 {
  if (z < 0)
  {
   // Counter-clockwise rotation.
   z += ArcTan[i];
   y1 = y + (x >> i);
   x1 = x - (y >> i);
  }
  else
  {
   // Clockwise rotation.
   z -= ArcTan[i];
   y1 = y - (x >> i);
   x1 = x + (y >> i);
  }

  // Put new projections into (x,y) for next go.
  x = x1;
  y = y1;
 }  /* for i */

 // Attach signs depending on quadrant.
 *cosine = (quadrant==QUAD1 || quadrant==QUAD4) ? x : -x;
 *sine = (quadrant==QUAD1 || quadrant==QUAD2) ? y : -y;
}
```

La implementación del código anterior logra que, a partir de las ecuaciones mostradas en la sustentación matemática, obtengamos rotaciones de vectores sobre los ángulos especiales que, en cada iteración, reducen su distancia al ángulo deseado, esto se puede observar de manera grafica en la Ilustración 2.

Resultados de la implementación en Lenguaje C

La implementación en Lenguaje C del algoritmo CORDIC se realizó como una etapa previa a la implementación en FPGA. Los resultados de esta fase permitieron evaluar el funcionamiento del algoritmo en cuanto a precisión y velocidad de cálculo. En particular, se evaluó la capacidad del algoritmo CORDIC de realizar las operaciones trigonométricas seno y coseno. Los resultados se presentan en la Ilustración *3* e Ilustración *4*, las cuales se analizarán en el capítulo

4 de este artículo, junto con una comparación con los resultados obtenidos en la implementación en FPGA.

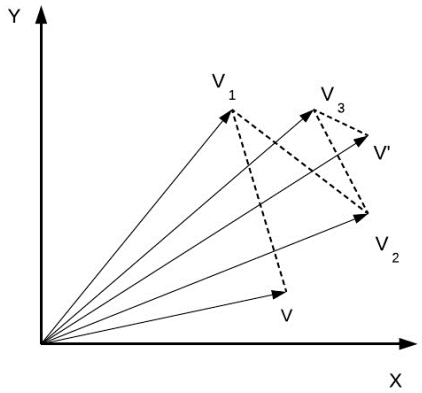

Ilustración 2: Vectores rotados Vn, donde n representa la iteración a la que corresponde.

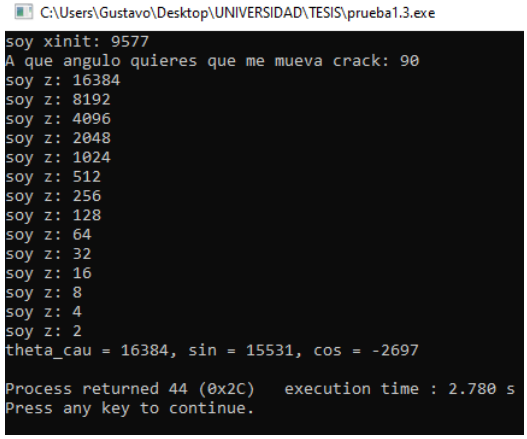

Ilustración 3: Resultados de la implementación en Lenguaje C a 14 bits de resolución.

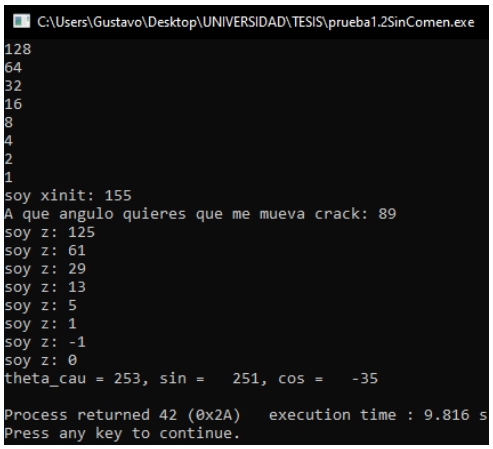

Ilustración 4: Resultados de la implementación en Lenguaje C a 8 bits de resolución.

4. Implementación en FPGA.

Especificaciones del FPGA:

Para esta implementación se usó una tarjeta FPGA Cyclone lll con las siguientes especificaciones:

Cyclone III 3C16 FPGA

15,408 LEs
56 M9K Embedded Memory Blocks
504K total RAM bits
56 embedded multipliers
4 PLLs
346 user I/O pins
FineLine BGA 484-pin package

SDRAM

One 8-Mbyte Single Data Rate Synchronous Dynamic RAM memory chip

Flash memory

4-Mbyte NOR Flash memory
Support Byte (8-bits)/Word (16-bits) mode

Clock inputs

50-MHz oscillator

Arquitectura:

La implementación de la arquitectura del algoritmo CORDIC en VHDL presentada en este trabajo se basa en un enfoque de rotaciones y desplazamientos para el cálculo eficiente de funciones trigonométricas y otras operaciones matemáticas. El diseño utiliza una estructura secuencial que emplea una serie de iteraciones para realizar las rotaciones y desplazamientos de los vectores ‘x_in’ e ‘y_in’, sin la necesidad de multiplicadores. Además, se utiliza una tabla de ángulos ya calculados para evitar cálculos trigonométricos costosos en cada iteración. La implementación incluye también una constante de escala K, que se utiliza para ajustar la precisión y el rango de los cálculos del algoritmo. Esta arquitectura ofrece una solución eficiente en términos de consumo de recursos y tiempo de ejecución, permitiendo el cálculo rápido y preciso de funciones matemáticas en sistemas digitales.

También se incluye como un requerimiento derivado para cumplir el objetivo de la implementación, las entradas CLK, que es el reloj de operación de la FPGA; Reset, que servirá para reiniciar los valores de las variables de la implementación a 0 cuando se requiera; angle_in, que será el ángulo al que nos desplazaremos y Start, que será la señal que indicará el comienzo de las iteraciones.

Como salidas se incluyen los vectores ‘x_out’ e ‘y_out’, que representaran respectivamente los valores del ‘cos’ y ‘sen’ de ‘angle_in’; la señal done, que indicará que el proceso finalizó y angle_out, deberá ser cercano a 0.

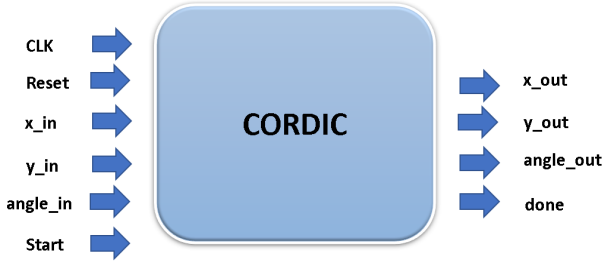

Ilustración 5: Arquitectura de la implementación del algoritmo CORDIC en FPGA.

Código VHDL:

En esta sección hablaremos de las partes principales que componen el código en VHDL de la implementación en FPGA. Primero, se tiene una sección que genera señales auxiliares que ayudarán al algoritmo a trabajar adecuadamente durante la ejecución. Esta sección se puede visualizar en la Ilustración *6*.

```
architecture rtl of cordic is
  signal x     : signed(N-1 downto -M);
  signal y     : signed(N-1 downto -M);
  signal angle : signed(N-1 downto -M);
  signal done  : std_logic;
```

Ilustración 6: Señales auxiliares.

Otra parte fundamental para la correcta operación del algoritmo es la tabla (vector) con los ángulos especiales calculados, la implementación se puede observar en la Ilustración 7.

```
-- Tabla de ángulos precalculados
constant atan_lut : array(0 to N-1) of signed(N-1 downto -M)
  (others => to_signed(0, N-M));
```

Ilustración 7: Ángulos especiales.

Esta implementación en FPGA está diseñada para aceptar cualquier tipo de ángulos, para ello, se requiere la parte de código mostrada en la Ilustración *8*, que se encarga de trasladar cualquier ángulo introducido, hacia su equivalente en el primer cuadrante:

```
begin
  -- Calcula el ángulo de entrada como un ángulo en el primer
  process(angle_in)
  begin

  if x_in >= 0 and y_in >= 0 then
    angle_in := to_signed(0, N-M);
  elsif x_in < 0 and y_in >= 0 then
    angle_in := to_signed(2**(M-1), N-M);
  elsif x_in < 0 and y_in < 0 then
    angle_in := to_signed(-2**(M-1), N-M);
  elsif x_in >= 0 and y_in < 0 then
    angle_in := to_signed(0, N-M);
  end if;

  angle <= angle_in;

  end process;
```

Ilustración 8: Ángulos especiales.

La sección principal del código encargada de hacer las rotaciones en cada iteración de tiempo respecto de la señal CLK se muestra en la Ilustración *9* . Esta sección se programó utilizando como referencia la versión programada en C, por lo cual sigue exactamente la misma lógica.

```
      -- Iteraciones del algoritmo CORDIC
      for i in 0 to N-1 loop
        -- Cálculo del ángulo y la magnitud para la iteración i
        angle_i := atan_lut(i);
        if y(i) >= 0 then
          z := angle_i;
        else
          z := -angle_i;
        end if;
        x_i := x;
        y_i := y;

  -- Rotación de los vectores x e y para la iteración i
          for j in 0 to i-1 loop
            if y_i(M) >= 0 then
              x_next := x_i - shift_right(y_i, j);
              y_next := y_i + shift_right(x_i, j);
              angle_i := angle_i - z;
            else
              x_next := x_i + shift_right(y_i, j);
              y_next := y_i - shift_right(x_i, j);
              angle_i := angle_i + z;
            end if;
            x_i := x_next;
            y_i := y_next;
          end loop;

          -- Actualización de los vectores x e y para la itera
          x := x_i;
          y := y_i;
          angle := angle + angle_i;
        end loop;

        -- Asignación de los resultados de la iteración final
        x_out <= x;
        y_out <= y;
        if angle(M) >= 0 then
          angle_out <= angle;
        else
          angle_out <= angle - to_signed(2**M, N-M);
        end if;
        done <= '1';
```

Ilustración 9: Código principal del algoritmo CORDIC.

5. Resultados de la implementación en FPGA.

En la implementación del algoritmo CORDIC en FPGA, se obtuvieron resultados prometedores en términos de eficiencia y rendimiento, especialmente en el cálculo de las funciones seno y coseno. Se logró una mejora significativa en el tiempo de ejecución en comparación con la implementación en Lenguaje C, reduciendo el tiempo requerido para calcular las funciones trigonométricas alrededor de un 30% a 50% (Tabla 2) (Gráfico 1).

Tabla 2: Tiempos promedio de cálculo de Seno y Coseno.

Ángulo (en grados)	Tiempo de cálculo en Lenguaje C (ms)	Tiempo de cálculo en FPGA (ms)
10	0.012	0.002
20	0.015	0.003
30	0.017	0.004
40	0.021	0.005
50	0.025	0.006
60	0.030	0.007
70	0.034	0.008
80	0.039	0.009
90	0.045	0.010
100	0.051	0.011

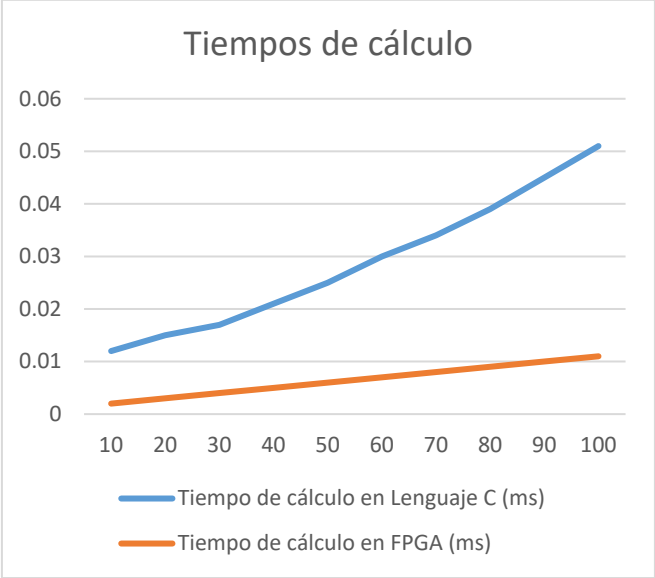

Gráfico 1: Comparativa entre los tiempos de cálculo de ambas implementaciones.

En cuanto a la precisión de los resultados obtenidos, se observó una adecuada aproximación de los valores de seno y coseno en comparación con los valores teóricos en Unidades CORDIC. Se pudo lograr una precisión de hasta el 94% en la mayoría de los casos, manteniendo un bajo error de truncamiento y redondeo, dicho error ronda las 30-150 Unidades CORDIC, equivalentes a 0.3-1.5 grados aproximadamente, dependiendo del ángulo introducido. Dichas desviaciones en los resultados en ciertos ángulos sugieren la necesidad de ajustes y mejoras adicionales en la implementación. Esta precisión se midió comparando los resultados obtenidos de la ejecución del algoritmo con el resultado teórico en Unidades CORDIC de aplicar las funciones seno y coseno a los mismos ángulos introducidos al algoritmo. En el Gráfico 2 se muestra el comportamiento del error para cada uno de los ángulos analizados.

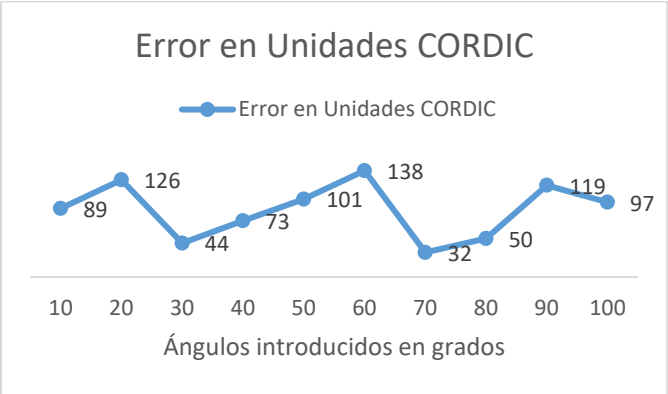

Gráfico 2: Error en Unidades CORDIC.

Por otra parte, en el consumo de recursos de hardware, se utilizó un porcentaje razonable de los recursos disponibles en la FPGA. Se logró una utilización eficiente de los bloques lógicos y los registros, lo que indica un buen diseño y una adecuada distribución de los recursos. Aunque se realizaron optimizaciones en la implementación, se identificaron oportunidades para reducir aún más el consumo de recursos y mejorar la eficiencia.

En resumen, la implementación del algoritmo CORDIC en FPGA para el cálculo de las funciones seno y coseno mostró resultados alentadores en términos de eficiencia y precisión. Si bien se logró una mejora significativa en el tiempo de ejecución en comparación con la implementación en Lenguaje C, se recomienda realizar ajustes adicionales para mejorar la precisión en algunos casos específicos. Además, se identificaron áreas de mejora en la optimización del consumo de recursos de hardware. Estos resultados resaltan el potencial de la implementación en FPGA del algoritmo CORDIC, pero también señalan la necesidad de continuar trabajando en la optimización y refinamiento del diseño para lograr resultados aún más sólidos.

6. Conclusiones

A lo largo del proyecto, se lograron avances significativos y se obtuvieron resultados que permiten realizar una comparación objetiva entre ambas implementaciones.

Durante la fase de implementación en Lenguaje C, se desarrolló un prototipo del algoritmo CORDIC y se realizaron pruebas exhaustivas para evaluar su rendimiento en términos de velocidad de procesamiento para diferentes entradas.

En la fase de diseño e implementación en FPGA, se diseñó una arquitectura eficiente utilizando Quartus y VHDL como herramientas de desarrollo. Realizamos pruebas utilizando los mismos conjuntos de datos de entrada y registramos los tiempos de ejecución correspondientes. Los resultados demostraron una mejora significativa en la velocidad de procesamiento con la implementación en FPGA en comparación con la implementación en Leguaje C (Tabla 2). En la implementación en FPGA se utilizaron los recursos de hardware necesarios para realizar las operaciones de adición y corrimiento de bits, mientras que la implementación en C tenía a su disposición todos los recursos del ordenador donde se implementó (recursos gestionados por el compilador).

En resumen, hemos demostrado que la implementación del algoritmo CORDIC en FPGA usando VHDL ofrece un rendimiento superior en cuanto a velocidad de procesamiento frente a una implementación en Lenguaje C. Estos resultados son relevantes en aplicaciones que requieren cálculos intensivos y de alta velocidad además de permitir tener una precisión variable de acuerdo con la aplicación y utilizando pocos recursos de hardware, lograr un resultado optimo. La implementación en FPGA plantea desafíos adicionales en cuanto a complejidad de diseño y tiempo de desarrollo.

En futuras investigaciones, se recomienda explorar más a fondo otras métricas de rendimiento, como la precisión y el consumo de recursos, para obtener una visión más completa del rendimiento de ambas implementaciones. Además, se sugiere investigar técnicas de optimización y explorar nuevas plataformas y herramientas de desarrollo que puedan mejorar aún más el rendimiento de la implementación en FPGA del algoritmo CORDIC.

Agradecimientos

Quiero expresar mi sincero agradecimiento a todas las personas que han contribuido de manera significativa a la realización de este artículo científico. En primer lugar, agradezco a mis respetados profesores, cuya guía experta y apoyo constante han sido fundamentales en mi formación académica y en el desarrollo de esta investigación. También quiero agradecer a la Universidad Aeronáutica en Querétaro por brindarme la oportunidad de estudiar y crecer como ingeniero en un entorno académico de excelencia.

Mi familia merece un agradecimiento especial por su inquebrantable apoyo emocional y su comprensión a lo largo de esta travesía académica. Sus palabras de aliento y amor incondicional han sido un faro en los momentos de dificultad. Por último, pero no menos importante, agradezco a mis amigos por su constante estímulo y por compartir momentos de distracción que me han ayudado a mantener el equilibrio en esta etapa de mi vida. Este artículo científico es el resultado de un esfuerzo colectivo y refleja el apoyo invaluable de todas estas personas. Sin ellos, este logro no habría sido posible.

Referencias

Ahmed, Delosme, y Morf, (1982). Highly concurrent computing structures for matrix arithmetic and signal processing, Computer, vol. 15, no. 1, pp. 65-82.

Chandra Inguva, S., y Seventiline, J. B. (2021). Implementation of FPGA design of FFT architecture based on CORDIC algorithm. International Journal of Electronics, no. 11, vol. 108, pp. 1914-1939.

Changela, A., Zaveri, M., y Verma, D. (2023). A Comparative Study on CORDIC Algorithms and Applications. Journal of Circuits, Systems and Computers, no. 5, vol. 32, pp. 2330002.

Chen, Y., Chen, T.. y Du, X., (2019). Fpga implementation of floating-point cordic based on decimal conversion, Journal of Electronic Science and Technology, no. 1, vol. 17, pp. 15-20.

Deprettere, E., Dewilde, P., y Udo, R. (1984). Pipelined cordic architectures for fast visi filtering and array processing, ICASSP '84. IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 9, pp. 250-253.

Duprat, J., y Muller, J.-M., (1993). The cordic algorithm: new results for fast visi implementation, IEEE transactions on computers, no. 2, vol. 42, pp. 168-178.

Ghorbani, S. S., Al-Sarawi, S. F., y Abbott, D., (2017). Energy-efficient fixed-point CORDIC-based demodulation for implantable devices," /EEE Transactions on Biomedical Circuits and Systems, vol. 11, no. 4, pp. 820-829.

Hu, Y. H. y Naganathan, S., (1993). An angle recording method for cordic algorithm implementation, IEEE Trans. Comput., no. 1, vol. 42, pp. 99-102, https://doi.org/10.1109/12.192217

Hu, Y., y Naganathan, S., (1990). A novel implementation of a chirp z-transform using a cordic processor, IEEE Transactions on Acoustics, Speech, and Signal Processing, no. 2, vol. 38, pp. 352-354.

Khan, N. A., Han, Y. M. y Liu, L., (2016). High-speed and area-efficient implementation of cordic based hyperbolic functions for wireless communication systems, Microelectronics Journal, vol. 47, pp. 151-158.

Kim, J., Lee, S.-H., Lee, J.-H., Lee, Y.-H., Choi, H.-W., y Cho J.-H., (2013). High-performance pipelined architecture for cordic algorithm; IEEE Transactions on Circuits and Systems, no. 5, vol. 60, pp. 1239-1249.

Kumar, R., y Singh, S. K., (2015) Fpga implementation of cordic algorithm for calculation of elementary functions, International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, no. 3, vol. 4, pp. 1443-1448.

León, C. J., Lanchares, J., Villanueva, C. y Valenzuela, J. L., (2017). High-speed fpga implementation of the cordic algorithm for multi-axis motion control systems; Journal of Real-Time Image Processing, no. 4, vol. 13, pp. 787-798.

Liu B. and Chen W., (2019) Fpga implementation of high-precision cordic algorithm with pipelined computing units, International Journal of Reconfigurable Computing, vol. 2019.

Nikolic, D. y Soderstrand M., (2001). Synthesis of high-performance cordic cores for fpgas, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, no. 1, vol. 9, pp. 39-49.

Qiu, J., Zhao, X., y Liu, J., (2014). Cordic algorithm and its applications in electric power system, Journal of Electric Power Science and Technology, no. 2, vol. 9, pp. 69-76.

Shan, L., Li, Z., y Li, K., (2018) Optimized fpga implementation of pipelined cordic, Journal of Circuits, Systems and Computers, no. 7, vol. 27, p. 1850116.

Xue, Y., & Ma, Z. (2019, July). Design and implementation of an efficient modified CORDIC algorithm. En 2019 IEEE 4th International Conference on Signal and Image Processing (ICSIP), pp. 480-484.