MÁQUINA CALCULADORA DE FUNCIONES TRASCENDENTALES BASADA EN EL ALGORTIMO CORDIC CON NIOS II.

Juliana Basurto [(1)], Jorge Crespín [(2)], Ing. Ronald Ponguillo [(3)]
Facultad de Ingeniería en Electricidad y Computación [(1)(2)(3)]
Escuela Superior Politécnica del Litoral (ESPOL) [(1)(2)(3)]
Campus Gustavo Galindo, Km 30.5 Vía Perimetral, Apartado 09-01-5863. Guayaquil, Ecuador [(1)(2)(3)]
jabasurt@espol.edu.ec [(1)], joxacres@espol.edu.ec [(2)], rponguil@espol.edu.ec [(3)]

Resumen

El proyecto tiene como finalidad la implementación de un sistema embebido en una arquitectura de hardware basada en el procesador configurable Nios II y programado en un FPGA, que tendrá inmerso un módulo ip core cuya función es realizar el cálculo de funciones trascendentales.

El módulo IP CORE está basado en el algoritmo CORDIC y juntos a otros bloques embebidos en un dispositivo ALTERA CYCLONE II FPGA se interconectan al procesador NIOS II, por medio de la implementación de un bus Avalon que es generado y configurado de manera automática por la herramienta de construcción SOPC Builder. Este proyecto está limitado al cálculo de 3 funciones trigonométrica que son Seno, Coseno y Tangente, el usuario ingresara por teclado la función a calcular y el valor del ángulo, las dos primeras obtenidas del core CORDIC y la Tangente como resultado de la relación de Seno y Coseno. La finalidad de usar un IP Core radica en mejorar la eficiencia del sistema medida por su tiempo de respuesta.

Palabras Claves: *IP CORE, FPGA, NIOS II, SOPC Builder.*

Abstract

The Project has as its goal the implementation of a configurable NIOS II hardware based embedded System and programmed into a FPGA that will contain an IP core module which function is to execute transcendental operations.

The IP CORE is based in CORDIC algorithm and together with others embedded blocks into an ALTERA CYCLONE II FPGA device, interfaces with the NIOS II processor by an Avalon bus generated and configured automatically by the SOPC builder construction tool. This Project is limited to calculate 3 trigonometric functions which are Sine, Cosine and Tangent; the user inputs by the keyboard the function to execute and the angle value, the first two functions are gotten from the CORDIC core and the Tangent function as a result of the Sine and Cosine relation. The finality of using an IP CORE resides into improve the system efficiency measuring its time response.

Keywords*: IP CORE, FPGA, NIOS II, SOPC Builder.*

1. Introducción

Uno de los tantos problemas presentes en la ingeniería es el cálculo de funciones trascendentales (funciones trigonométricas, logarítmicas y exponenciales) debido a su complejidad y a los recursos que se requiere para la implementación, en la actualidad existen grandes sistemas que usan aplicaciones que requieren de este tipo de cálculo, como ejemplo podemos mencionar a los detectores de fase, donde es necesario el cálculo de la tangente inversa para sincronización de portadora en sistemas de Radio Software (SR) o los sistemas de aviación donde debido a diferentes situaciones climáticas es necesario calcular la rapidez y velocidad de vuelo realizando funciones senos y cosenos.

El uso de un FPGA nos brinda una excelente vía para el desarrollo de aplicaciones que requieran del cálculo de funciones matemáticas (inclusive complejas) sobre ambientes de software donde no se posee unidades aritméticas complejas o multiplicadores, y sobre todo a bajo costo, es por eso que el algoritmo CORDIC es adecuado para este proyecto, ya que está basado en desplazamientos, sumas y tablas de búsqueda, que no requiere de unidades aritméticas específicas para ser implementado, lo que lo hace muy efectivo en nuestro hardware.

2. Algoritmo CORDIC

2.1. Fundamento teórico

[1] Propuesto por Jack Volder en el año 1959 y es un método para calcular funciones elementales usando lo mínimo en hardware, como registros de desplazamientos, sumadores/restadores y comparadores. Existen dos modos de operación: Modo rotación y el modo vectorial.

2.1.1. Modo de rotación del CORDIC

En éste modo, el acumulador del ángulo es inicializado con el ángulo deseado. La decisión de la rotación en cada iteración es realizada para disminuir la magnitud del ángulo residual en el acumulador. Por lo tanto, dicha decisión está basada en el signo del ángulo residual después de cada paso.

Las ecuaciones de recurrencia utilizadas en éste modo son:

$X_{i+1} = X_i - Y_i \cdot d_i \cdot 2^{-i}$

$Y_{i+1} = Y_i - X_i \cdot d_i \cdot 2^{-i}$

$z_{i+1} = Z_i - \cdot d_i \cdot \tan^{-1}(2^{-i})$

Donde $di = \begin{cases} -1, Z_i < 0 \\ -1, Z_i \geq 0 \end{cases}$

Y los resultados finales para cada parámetro son:

$X_n = A_n (X_0 \cos Z_0 - Y_0 \operatorname{sen} Z_0)$

$Y_n = A_n (Y_0 \cos Z_0 - X_0 \operatorname{sen} Z_0)$

$Z_n = 0$

Donde $A_n = \Pi_n \sqrt{1 + 2^{-2i}}$. Nótese que al hacer las condiciones iniciales $Y_0=0$ y $X_0=1/A_n$, los resultados serían el seno y el coseno de Z_0.

El algoritmo CORDIC original solamente funciona para ángulos entre $-\pi/2$ y $\pi/2$, por lo que para ángulos fuera de éste intervalo hay que tener en cuenta ciertas consideraciones de signo en los parámetros *X* y *Y*, además se debe sumar un desplazamiento en el ángulo inicial *Z*.

2.1.2. Modo vectorial del CORDIC

En modo vectorial, el vector de entrada es rotado cierto ángulo con el fin de alinearlo con el eje *x*. Esta operación da como resultado el ángulo de rotación y la magnitud escalada del vector original, en este caso la componente *x* del resultado. Lo que esta función vectorial hace es disminuir al máximo la componente *y* del resultado que se obtiene con cada rotación, este valor de componente en *y* es usado para determinar la dirección para la cual se debe hacer la rotación. En caso de que el valor inicial del acumulador sea cero, el resultado arrojado por la operación será el del ángulo recorrido durante la rotación total, es decir, cuando terminen las iteraciones. Las ecuaciones para este modo están definidas de la siguiente forma:

$x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i}$

$y_{i+1} = y_i - x_i \cdot d_i \cdot 2^{-i}$

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

Donde

$d_i = +1$ Si $y_i < 0$, -1 para otros valores

Por lo tanto:

$$x_n = A_n \sqrt{{x_0}^2 + {y_0}^2}$$

$$y_n = 0$$

$$z_n = z_n + d_i . tan^{-1}\left(\frac{y_0}{x_0}\right)$$

$$A_n = \Pi_n \sqrt{(1 + 2^{-2i})}$$

Para obtener la función arco tangente, es decir, $\theta = tan^{-1}\left(\frac{y}{x}\right)$, es necesario usar el modo vectorial del módulo CORDIC inicializando el ángulo del acumulador con cero. Con el fin de calcular valores infinitos

$(x = 0)$, el argumento de entrada debe ser ingresado como un vector (x, y). Una vez el acumulador guarde

el valor del ángulo calculado, este valor no se verá afectado por el incremento rotacional.

$$z_n = z_0 + tan^{-1}\left(\frac{y_0}{x_0}\right)$$

2.2. Arquitectura de implementación del algoritmo CORDIC

La arquitectura que se usa es la arquitectura Bit-Paralela Desplegada
La denominación de paralela se debe a la forma en que se opera con las componentes X, Y y Z. El diseño se separa en etapas correspondientes a cada iteración.

Cada etapa está compuesta por los mismos componentes, dos unidades de desplazamiento y dos sumadores algebraicos. Por consiguiente la salida de una etapa corresponde a la entrada de la siguiente etapa. Los valores iniciales para X0, Y0 y Z0 se ingresan en paralelo a la primera etapa.
En el esquema se supone que las componentes tienen un ancho de m bits. La señal Modo indica el modo de Operación (Rotación o Vectorización).

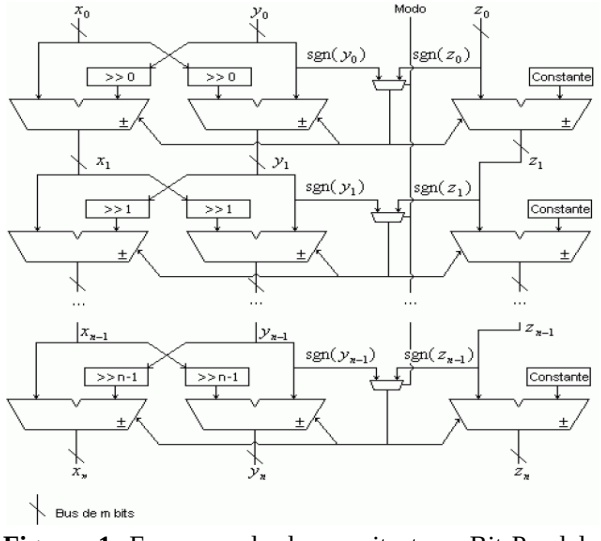

Figura 1 Esquema de la arquitectura Bit-Paralela Desplegada

2.3. Ventajas del algoritmo CORDIC

Debido a que dicho algoritmo funciona en base a sumas, restas y desplazamientos, el hardware requerido para su implementación no requiere de unidades aritméticas sofisticadas que son costosas he aquí una ventaja del algoritmo CORDIC es que el costo de su implementación es bajo, y debido al hardware que usa consume menos energía en comparación con un sistema cuyo hardware trabaje realizando cálculos en su ALU. La existencia de este algoritmo permite que sea posible poder implementar sistemas que hagan el cálculo de funciones trascendentales sobre hardware básico, que no posea unidades aritméticas complejas como por ejemplo un microcontrolador PIC.

3. Plataformas de desarrollo.

3.1. Tarjeta DE2 de Altera

Es la parte tangible de este proyecto, esta tarjeta nos proporciona una serie de módulos q pueden ser administrados desde un software por el desarrollador, los módulos usados en este proyecto son [2]:

- FPGA Cyclone II EP2C3F672C6
- Puertos de expansión (40 pines)
- USB-Blaster para la configuración de la FPGA.
- Display LCD de 16x2
- 3 LEDs verdes
- Reloj de 50 MHz

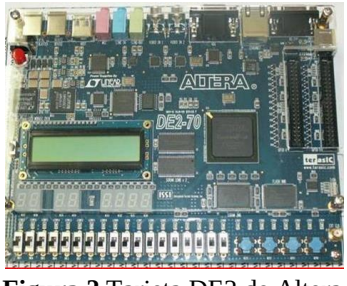

Figura 2 Tarjeta DE2 de Altera

[4] El FPGA es la parte más importante porque es allí donde va a reposar nuestro sistema basado en NIOSII. Pero para poder administrar los recursos de esta tarjeta se tiene que estableces una conexión entre el host-tarjeta, el host se refiere al computador que contiene el software para el diseño del sistema, esta conexión es posible mediante el USB-Blaster.

3.2. Quartus II

Este software ofrece un completo entorno de desarrollo multiplataforma que se adapta fácilmente a las necesidades específicas de diseño de SOPC[3]. Esta herramienta permite el análisis y la síntesis de diseños realizados en lenguaje HDL, examinar diagramas RTL [5].Dentro de Quartus II se encuentra una herramienta de desarrollo de sistema llamada SOPC Builder.

3.3. SOPC Builder

Es una herramienta de propósito general para la creación de sistemas que pueden o no pueden contener un procesador. Permite definir y generar un completo SOPC de alto rendimiento en mucho menos tiempo que el método tradicional.

Con esta herramienta es posible diseñar la estructura de un sistema de hardware, que por medio de

una interfaz gráfica de usuario, que permite agregar componentes a un sistema, configurar los componentes, y genera la lógica de interconexión de forma automática [6].

3.4. NIOS II SBT para Eclipse

El Nios II SBT para Eclipse es un conjunto de plugins basados en la infraestructura de Eclipse, nos permite crear interfaces usuario-hardware que se pueden escribir en varios lenguajes de programación, por ejemplo: Lenguaje Assembler, C, C++, Java.

Esta es la herramienta que nos permitirá controlar el sistema embebido en la FPGA, por medio de instrucciones descritas en leguaje C, que es el lenguaje que se eligió para el desarrollo de este proyecto.

4. Diseño e implementación.

Antes de proceder a realizar cualquier diseño, es necesario saber cómo debe funcionar el sistema, en el siguiente diagrama de flujo se muestra el proceso del funcionamiento de la máquina calculadora.

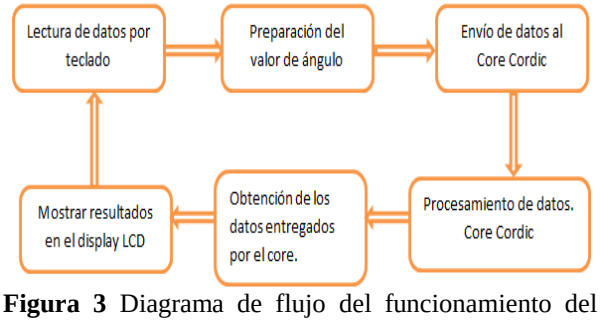

Figura 3 Diagrama de flujo del funcionamiento del sistema

4.1. Diseño del sistema basado en NIOS II

El diseño se lo realiza por medio de la herramienta SOPC Builder, en la Figura se muestra los módulos o IP`s añadidos, cada módulo corresponde al hardware que tendrá el proyecto:

Use	C...	Name	Description	Clock
☑		sysid	System ID Peripheral	clk_0
☑		cpu	Nios II Processor	clk_0
☑		memoria	On-Chip Memory (RAM or ROM)	clk_0
☑		jtag	JTAG UART	clk_0
☑		timer	Interval Timer	clk_0
☑		lcd	Character LCD	clk_0
☑		leds	Parallel Port	clk_0
☑		jp2	Parallel Port	clk_0
☑		salida	PIO (Parallel I/O)	clk_0
☑		entrada	PIO (Parallel I/O)	clk_0
☑		seno	PIO (Parallel I/O)	clk_0
☑		coseno	PIO (Parallel I/O)	clk_0
☑		zeta	PIO (Parallel I/O)	clk_0

Figura 4 Hardware agregado con SOPC Builder

En la segunda línea de la Figura 4 vemos que el sistema tiene un procesador NIOSII. Luego de haber añadido los IP`s necesarios para nuestro sistema, presionamos el botón GENERAR. Este proceso creará una descripción en lenguaje HDL de todos los IP que se agregaron al sistema SOPC Builder. En la Figura 5 podemos observar en un bloque esquemático el sistema basado en NIOSII.

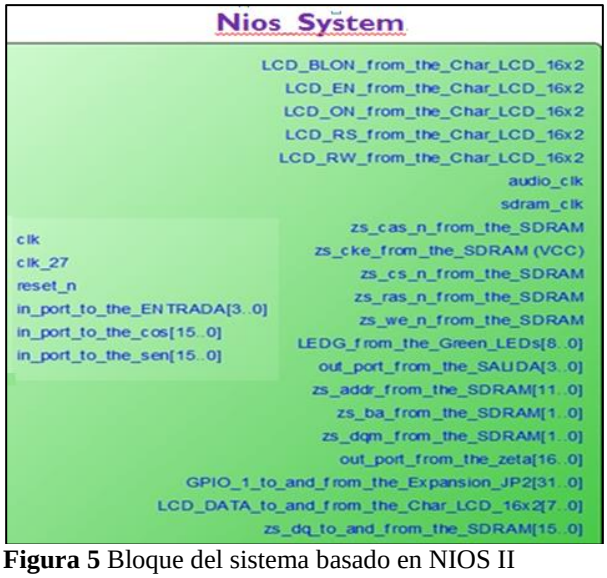

Figura 5 Bloque del sistema basado en NIOS II

4.2. Inclusión de un ip core al sistema basado en NIOS II

Nuestro sistema debe realizar el cálculo de funciones senos, cosenos y tangentes, para lo cual vamos a utilizar un CORE basado en el algoritmo CORDIC. El core está escrito en lenguaje VHDL (la creación de CORES utilizando lenguajes de descripción de hardware está fuera del alcance de este proyecto) y fue obtenido de la web www.opencores.org.

El core está formado por 3 archivos que deben ser agregados a nuestro proyecto en QuartusII.
Para añadir los archivos del core, seguiremos la siguiente ruta en QuartusII:

Project-> Add/Remove Files on a Project

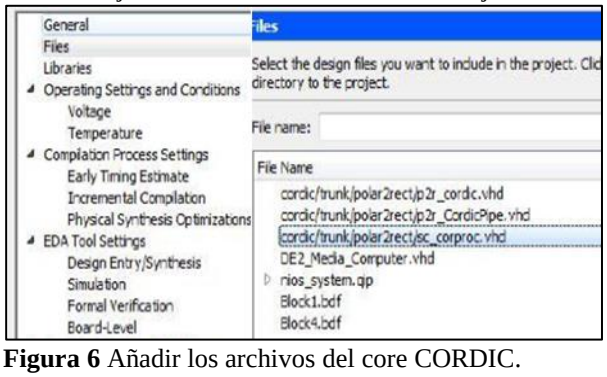

Figura 6 Añadir los archivos del core CORDIC.

Ahora el core es un bloque independiente como se muestra en la Figura 7 a nuestro sistema, hay que conectarlo con nuestro sistema (Figura 5).

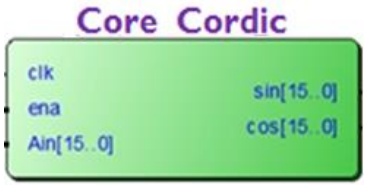

Figura 7 Bloque del core CORDIC.

Para realizar dicha conexión crearemos una nueva entidad con 2 componentes: nios_system (nuestro sistema) y sc_corproc (CORDIC core) y luego debemos unir ambos componentes con señales internas en los puertos correspondientes, debe quedar como se muestra en la siguiente figura.

```
-------------------------------------------------------
--   Internal Wires and Registers Declarations
-------------------------------------------------------
-- Internal Wires
-- Used to concatenate some SDRAM control signals
signal          BA : STD_LOGIC_VECTOR(1 DOWNTO 0);
signal          DQM : STD_LOGIC_VECTOR(1 DOWNTO 0);

signal         wzeta: signed(16 downto 0);
signal         wsen: signed(15 downto 0);
signal         wcos: signed(15 downto 0);
```

Figura 8 Conexión del core con el sistema.

Una vez hecho la conexión, podemos visualizar en un diagrama esquemático nuestro sistema completo, como se muestra en la Figura 9

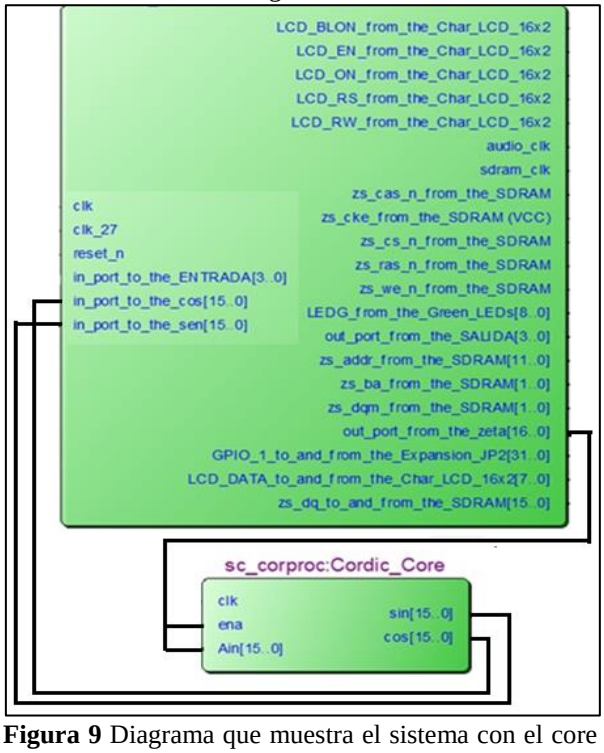

Figura 9 Diagrama que muestra el sistema con el core añadido.

4.3. Carga del sistema basado en NIOS II al FPGA

Ya diseñado nuestro sistema, ahora hay que embeberlo en la FPGA, para esto primero hay que hacer la asignación de pines físicos de la tarjeta a las terminales de nuestro diseño. Esto permite encaminar de manera correcta las señales generadas de manera interna del FPGA con sus correspondientes generadores /receptores fuera del dispositivo.

Tabla 1 Tabla con asignación de pines usados

Nombre de la señal	Dirección	Ubicación de PINES	Descripción
ENTRADA[3]	Input	PIN_M21	Señales de salida del teclado, que son leídas por el procesador
ENTRADA[2]	Input	PIN_N20	
ENTRADA[1]	Input	PIN_M20	
ENTRADA[0]	Input	PIN_M19	
LEDS[3]	Output	PIN_V18	Señales indicadores a mostrarse en los leds verdes
LEDS[2]	Output	PIN_W19	
LEDS[1]	Output	PIN_AF22	
LEDS[0]	Output	PIN_AE22	
SALIDA[3]	Output	PIN_M23	Señales de entrada el teclado, enviadas desde el procesador
SALIDA[2]	Output	PIN_M22	
SALIDA[1]	Output	PIN_K26	
SALIDA[0]	Output	PIN_K25	
CLOCK_27	Input	PIN_D13	Reloj de 27Mhz
CLOCK_50	Input	PIN_N2	Reloj de 50Mhz

Ya realizada la asignación de pines, se procede a compilar el proyecto en QuartusII donde son generados los archivos necesarios para programar y configurar la FPGA.
Antes de iniciar la carga del diseño debemos revisar:

- Que se esté aplicando alimentación a la tarjeta DE2.
- La conexión USB entre el PC y el puerto USB Blaster de la tarjeta.
- Configuración de la tarjeta DE2 esté en modo JTAG.
- El archivo de diseño que va a ser volcado debe tener extensión .sof

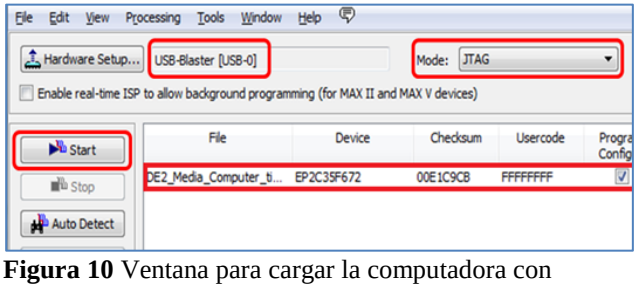

Figura 10 Ventana para cargar la computadora con NIOS II en el dispositivo.

Presionamos el botón START. Al terminar, la barra de progreso muestra 100%. En este momento la computadora basada en NIOS II está cargada en el

dispositivo pero aún no es capaz de realizar ninguna tarea sin el software apropiado.

4.4. Ingreso – obtención de datos

El ingreso de datos se logra por medio de un teclado hexadecimal que es conectado al puerto de expansión JP2 de la Tarjeta DE2 usando como medio un BUS de datos.

A la computadora basada en NIOS2 se agregó el PIO SALIDA y el PIO ENTRADA (ver Figura4) que son los encargados del transporte de los datos transmitidos/recibidos por el teclado desde el procesador al teclado y viceversa. Cada bits de estos PIO son asignados a una ubicación de pines en la FPGA (ver la Tabla2).

Figura 11 Teclado hexadecimal

5. Pruebas

Prueba 1: medición del tiempo de respuesta del core CORDIC

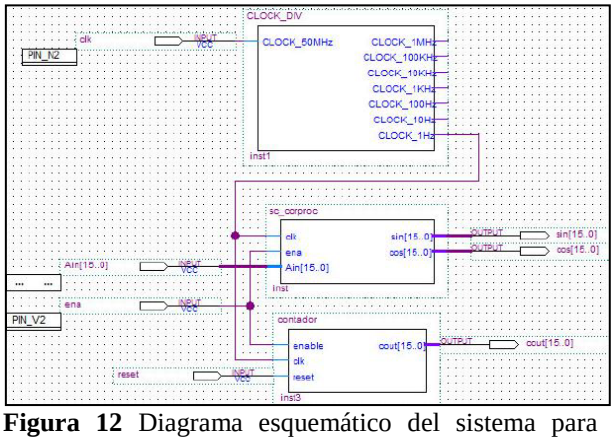

Figura 12 Diagrama esquemático del sistema para contar los flancos mediante un bloque contador.

Esta prueba consiste en medir el tiempo de respuesta que toma el core CORDIC en procesar arrojar un resultado, para esto usamos un contador que será el que nos de la cantidad de flancos que transcurran, tanto el core como el contador tendrán la misma habilitadora para que ambos se activen e manera simultánea, y la frecuencia a la que trabajaran ambos será de 1HZ que esta dado por un divisor de frecuencia. El contador mostrara su respuesta por medio de los LEDs rojos. Al finalizar la prueba, en valor que dio el contador es de 15 flancos de reloj.

Prueba 2: Medición del tiempo de respuesta del algoritmo CORDIC implementado en lenguaje C.

Esta prueba se basa en colocar el algortimo CORDIC descrito en lenguaje C en el código principal, y mediante el uso del temporizaor alt_timestamp medir el tiempo que se toma en realizar el cálculo, el algoritmo CORDIC en C.

```
int main(){
while(1){
int rr=alt_ticks_per_second();
int t1=0,t2=0,t3=0,p1,p2;
int angulo1=0, seno, coseno;
printf("Ingrese angulo: ");
scanf("%d",&angulo1);
int angulo2=angulo1*to_radianes*MUL;
int k, d, tx, ty, tz;
int x=cordic_1K,y=0,z=angulo2;
t1=alt_timestamp_start();
for (k=0; k<CORDIC_NTAB; ++k)
{
d = z>>(CORDIC_NTAB-1);
tx = x - (((y>>k) ^ d) - d);
ty = y + (((x>>k) ^ d) - d);
tz = z - ((cordic_ctab[k] ^ d) - d);
x = tx; y = ty; z = tz;
}
t2=alt_timestamp();
coseno = x; seno = y;
}
return 0;
}
```

alt_timestamp_start() inicia el contador y alt_timestamp() detiene el contador, entonces el valor del contador está almacenado en la variable t2. Al final de la prueba el temporizador dio por resultado el valor de 9671 flancos de reloj.

6. Resultados

La siguiente tabla resume los resultados obtenidos en las pruebas realizadas:

Tabla 2 Comparación de la medición de tiempo de respuesta en diferentes escenarios.

Prueba	Números de flancos
Contador de flacos – Core CORDIC	15
Implementacion en C	9671 (aprox)

Podemos observar que el core CORDIC retorna sus salidas más rápido que su implementación en C. Esta

diferencia se debe porque que el core CORDIC recibe el dato a calcular de manera directa (no pasa por el procesador NIOS), lo procesa y arroja una respuesta, pero al usar el algoritmo CORDIC en lenguaje C, para procesar un dato el código tiene que ser cargado en memoria y luego al procesador que es donde se va a ejecutar el código, lo que toma mas tiempo obtener una respuesta a diferencia del core CORDIC.

7. Conclusiones

- El desarrollo de sistemas embebidos posee enorme versatilidad frente a cambios en el diseño de hardware o software, ya que pueden ser modificados en cualquier momento sin alterar o añadir hardware extra en el PBC a diferencia de los métodos tradicionales.

- La precisión del resultado entregado por el algoritmo puede mejorarse aumentando el número de iteraciones con las cuales opera.
- El core CORDIC agregado al SOC entrega resultados mucho más rápido que la implementación del algoritmo escrita en lenguaje C dentro del código principal del sistema.

8. Recomendaciones

- La interconexión entre el core CORDIC y el Sistema Nios puede realizarse instanciando ambas entidades en un nuevo proyecto, añadir señales internas y sentencias de port map.

- Utilizar las herramientas de depuración incluidas en NIOS II SBT para la resolución de problemas que aparecieran durante el desarrollo del código.

- Para realizar mediciones de tiempo sobre la ejecución de código se recomienda utilizar un temporizador TIMESTAMP.
  -Nuestro hardware debe incluir al menos un temporizador en su diseño.
  -Debemos incluir en el código la librería que contiene las funciones necesarias de dicha herramienta:

  #include "sys/alt_timestamp.h"
  Funciones a utilizar:
  - alt_timestamp_start()
  - alt_timestamp()
  - alt_timestamp_freq()

9. Bibliografía

[1] Richard Herveille. Algoritmo CORDIC. http://cutler.eecs.berkeley.edu/classes/ee225c/Papers/cordic.pdf

[2] Tarjeta de desarrollo de2 de altera, https://paruro.pe/productos/tarjeta-altera-de2

[3] Sistemas en chip, http://es.wikipedia.org/wiki/System_on_a_chip

[4] Enciclopedia libre, FPGA. http://es.wikipedia.org/wiki/Field_Programmable_Gate_Array

[5] Enciclopedia libre, Quartus II. http://es.wikipedia.org/wiki/Quartus_II

[6] SOPC Builder guía del usuario http://www.altera.com/literature/ug/ug_sopc_builder.pdf