FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

Sistemas Digitales

Trabajo Práctico N3

Cordic

Denise Gayet 100828 dgayet@fi.uba.ar

11 de diciembre de 2023

Sistemas Digitales

Trabajo Práctico N3: Cordic

Denise Gayet 100828 dgayet@fi.uba.ar

Índice

1. Enunciado

El presente Trabajo Práctico tiene como objetivo aprender a especificar, diseñar, describir una arquitectura, simular, sintetizar e implementar en FPGA el algoritmo CORDIC.

2. Introducción

El algoritmo de CORDIC (Coordinate Rotation Digital Computer) se utiliza para calcular funciones trigonométricas e hiperbólicas de una manera simple, dado que solamente utiliza **sumas**, **restas**, **desplazamiento de bits** y **look-up tables**, por lo que es ideal para sistemas que no cuentan con una unidad de multiplicación. Este algoritmo cuenta con dos modos de operación: rotación y vectorización.

El modo rotación toma un vector $[x_1,\ y_1]$ y lo rota en un ángulo β para obtener el vector $[x_2,\ y_2]$, como se observa en la siguiente ecuación:

$$\begin{aligned} x_2 &= \cos(\beta)(x_1 - \mathrm{tg}(\beta)y_1), \\ y_2 &= \cos(\beta)(y_1 + \mathrm{tg}(\beta)x_1) \end{aligned} \tag{1}$$

Obviando la constante multiplicativa $\cos(\beta)$ y restringiendo los ángulos de rotación a $\mathrm{tg}(\beta) = 2^{-i}$ se puede reducir la multiplicación por la tangente a un desplazamiento hacia la izquierda.

Luego, reemplazando en la ecuación original queda:

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

donde $d_i = \pm 1$ indica si rotación es horario o anti-horario.
De esta manera, las únicas operaciones que se deben realizar pasan a ser sumas, restas y desplazamientos. Además, si se llega a cierta cantidad de rotaciones (iteraciones), la constante multiplicativa converge a $G = 1{,}64676026$ y no depende del ángulo de rotación.

Por último, la acumulación angular se puede describir como:

$$z_{i+1} = z_i - d_i \,\mathrm{tg}^{-1}(2^{-i}) \tag{3}$$

Esta ecuación se utiliza para saber si la rotación en la iteración i debe ser positiva o negativa, según el signo de la acumulación angular en dicha iteración. Para realizar el cómputo se necesita tener los valores del arco-tangente almacenados.

En el modo vectorial se rota un vector hacia el eje de coordenadas x, almacenando los ángulos requeridos para lograrlo. Una de las aplicaciones de este modo es realizar un cambio de coordenadas de sistema cartesiano a sistema polar.

Las ecuaciones de rotación de las componentes x e y son las mismas, pero difiere la variable que define si la rotación es positiva o negativa. En este caso, como se desea llevar el vector al eje de abscisas, se decide el valor de d_i en función del signo de y_i.

Utilizando este algoritmo en cualquiera de sus modos de operación se llega a que la máxima rotación posible está dada por:

$$\beta_n = \sum_{i=0}^{n} tg^{-1}(2^{-i}) \tag{4}$$

Esta sumatoria converge a un valor de 1,74329 radianes, o aproximadamente 100 grados.
En el caso de la rotación, si se desea rotar un ángulo mayor, se debe realizar el siguiente pre-procesamiento:

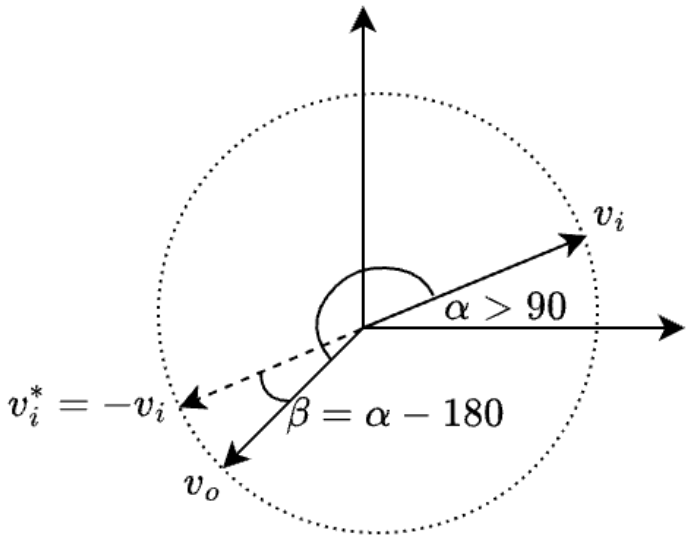

Figura 1: Pre-cordic rotacional

Sobre el vector original v_i se aplica una rotación de 180 grados, obteniendo v_i^* y al ángulo de rotación se le resta 180 grados obteniendo: $\beta = \alpha - 180$.

En el caso de la vectorización, como se quiere llegar al eje de abscisas, el pre procesamiento se debe realizar cuando el vector de entrada se encuentra en el segundo o tercer cuadrante ($x_{in} < 0$).

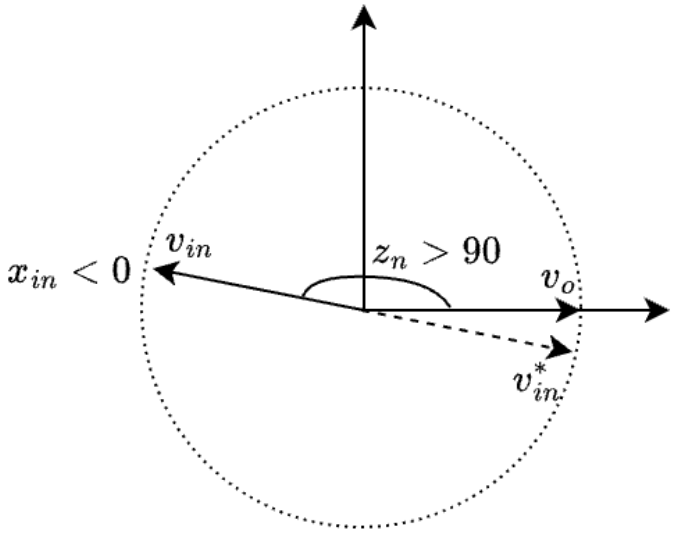

Figura 2: Pre-cordic para vectorización

En conclusión, el pre-procesamiento es el mismo en ambos casos, pero la condición necesaria para aplicarlo difiere.

En este trabajo se implementan ambos algoritmos (de rotación y de vectorización) y para cada uno se describen dos arquitecturas: la iterativa y la desenrollada. Además, en la forma desenrollada está disponible el sistema de *pipelining*, para poder cambiar los valores de entrada *on the fly*.

2.1. Diagrama esquemático

En la Fig. 3 se muestra el diagrama esquemático de la arquitectura iterativa del algoritmo de cordic.

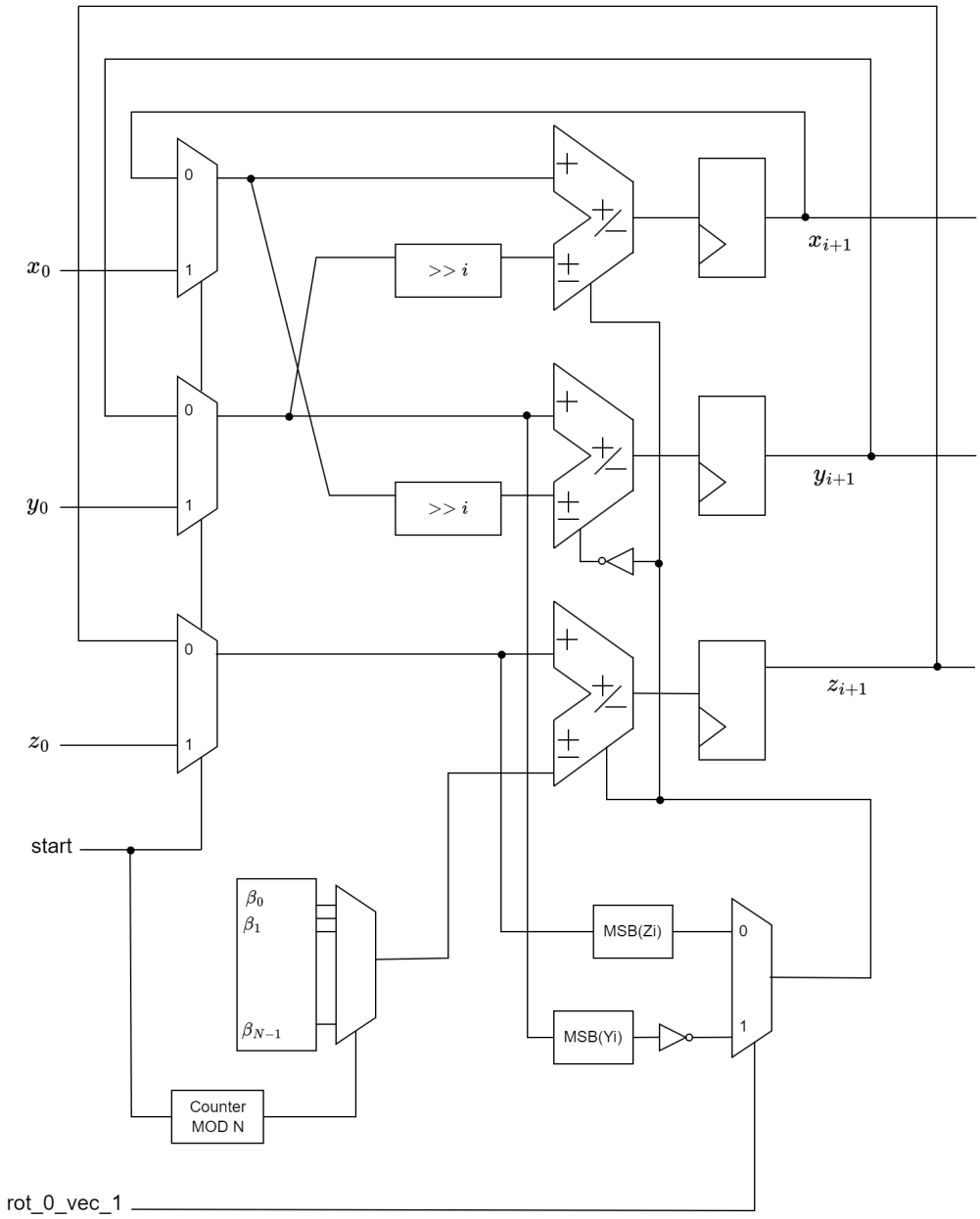

Figura 3: Esquemático de cordic iterativo

El modo iterativo toma N ciclos de reloj (donde N corresponde a la cantidad de rotaciones realizadas) para dar la salida final del circuito. A este esquemático se le agregó un flag que indica en qué momento la salida es válida, dado que está disponible en cada ciclo de reloj. De otra forma, también se podría haber agregado una capa más que disponga la salida únicamente al terminar el cómputo.

Por otro lado, en la Fig. 4 se muestra la arquitectura desenrollada. Como se puede observar, se concatenan N etapas de un mismo componente conectando la salida de una etapa a la entrada de la siguiente. La salida que lee el usuario es la de la última etapa. En este caso, si no se desea utilizar *pipelining* el circuito es puramente combinacional. Por el contrario, si se registran las salidas de cada etapa, también se necesitan N ciclos de reloj en computar el valor de salida.

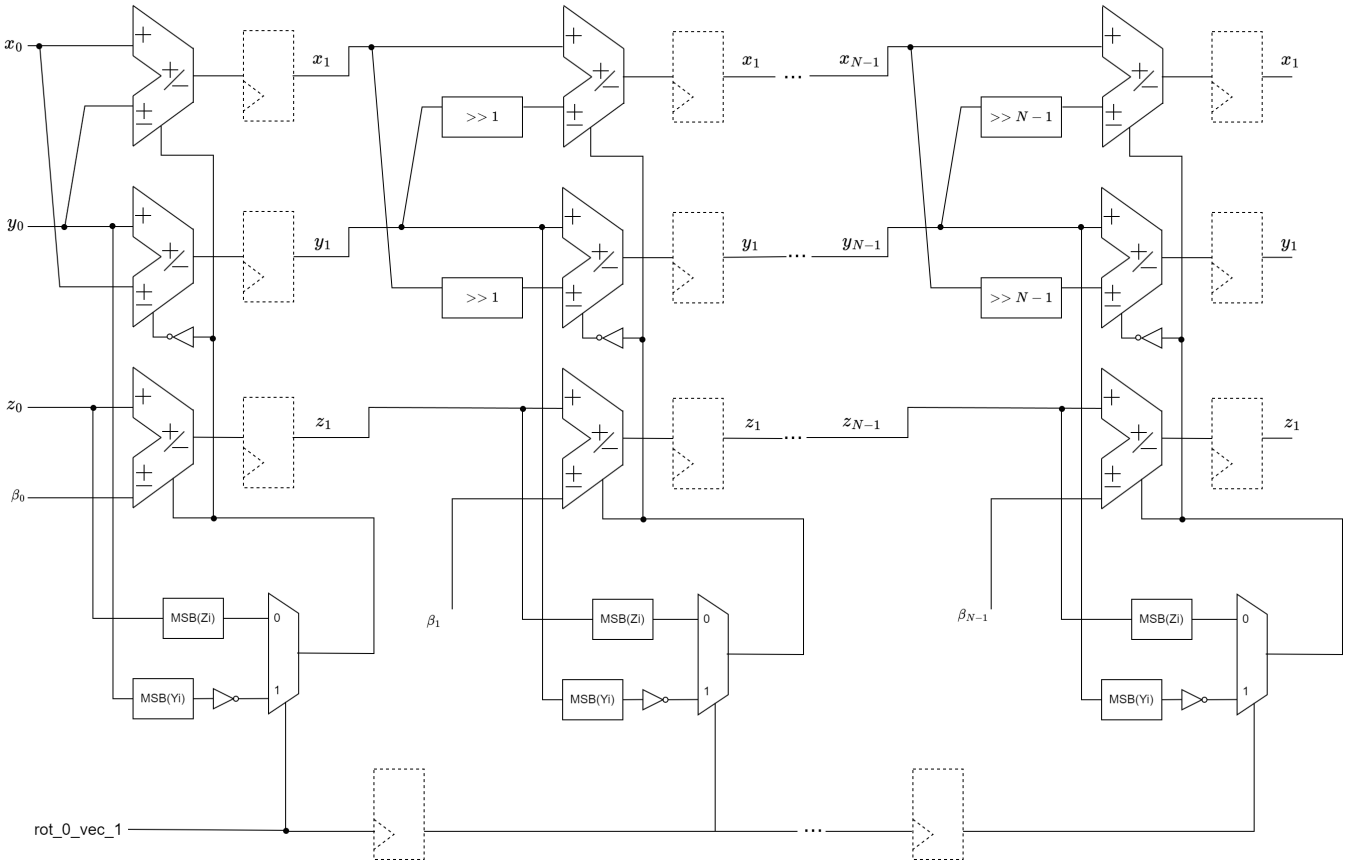

Figura 4: Esquemático de cordic desenrollado

En la Fig. 5 se observa el pre-procesamiento que se debe realizar antes de entrar al algoritmo.

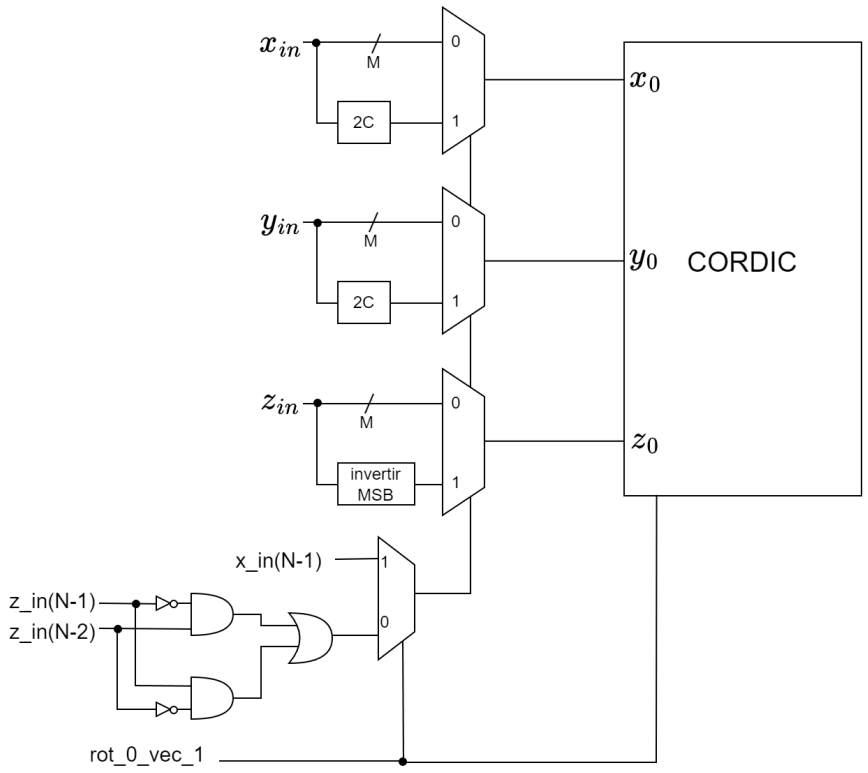

Figura 5: Esquemático del pre-cordic

Por último, en la Fig. 6 se observa el diagrama en bloques de los 3 grandes bloques de procesamiento: PRE-CORDIC: el pre-procesamiento de los datos de entrada, CORDIC: el algoritmo CORDIC propiamente dicho, y POST-CORDIC: la post-multiplicación por la ganancia del algoritmo.

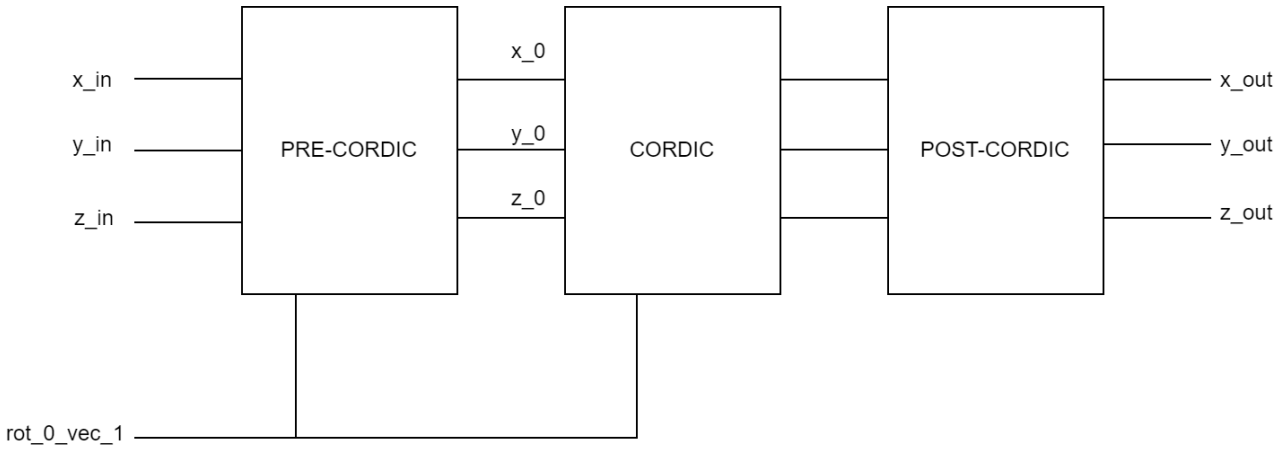

Figura 6: Diagrama en bloques del sistema

3. Descripción en VHDL

3.1. Generalidades

Si bien el algoritmo está planteado para ser genérico en cuanto a la cantidad de iteraciones que puede realizar, se optó por definir un arreglo con los valores de los ángulos a rotar (`BETAS`), con lo cual se tiene un máximo de iteraciones posibles.

Para la implementación del la arquitectura desenrollada se reutilizaron los bloques de suma/resta y el pre-cordic, por lo que no se incluirán nuevamente en dicha sección, sino únicamente en la sección de la arquitectura iterativa.

3.2. Arquitectura Iterativa

En primer lugar se muestra la descripción en VHDL de la arquitectura iterativa. Para implementar esta arquitectura se utilizaron distintas entidades: un contador de N ciclos, una entidad que contiene la lógica de suma/resta de las entradas, una entidad que implementa el algoritmo iterativo propiamente dicho, un componente encargado de realizar el pre-procesamiento, y la entidad superior.

3.2.a. Contador N ciclos

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  entity counter_N is
6      generic (
7          N : natural := 16
8      );
9      port (
10         clk_i: in std_logic;
11         rst_i: in std_logic;
12         restart : std_logic;
13         count: out integer range 0 to N-1
14     );
15 end;
16
17 architecture behavioral of counter_N is
18     signal aux_count: integer := 0;
19 begin
```

```
    count <= aux_count;
    process(clk_i, rst_i)
    begin
        if (rst_i = '1') then
            aux_count <= 0;
        elsif clk_i='1' and clk_i'event then
            if (aux_count = N-1 or restart='1') then
                aux_count <= 0;
            else
                aux_count <= aux_count + 1;
            end if;
        end if;
    end process;
end;
```

3.2.b. Pre-cordic

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity pre_cordic is
    generic (
        N : natural := 18
    );
    port (
        x_in : in std_logic_vector(N-1 downto 0);
        y_in : in std_logic_vector(N-1 downto 0);
        z_in : in std_logic_vector(N-1 downto 0);
        rot_0_vec_1 : in std_logic;
        x_out : out std_logic_vector(N-1 downto 0);
        y_out : out std_logic_vector(N-1 downto 0);
        z_out : out std_logic_vector(N-1 downto 0)
    );
end;

architecture behavioral of pre_cordic is
    signal selec : std_logic;
    signal z_sat : std_logic;

begin
    z_sat <= '1' when (z_in(N-1 downto N-2)="01" OR z_in(N-1 downto N-2)="10") else '0';
    selec <= z_sat when rot_0_vec_1='0' else x_in(N-1);

    x_out <= x_in when selec='0' else std_logic_vector(not signed(x_in) + 1);
    y_out <= y_in when selec='0' else std_logic_vector(not signed(y_in) + 1);
    z_out <= z_in when selec='0' else ((not z_in(N-1)) & z_in(N-2 downto 0));

end;
```

3.2.c. Bloque suma/resta

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity add_sub is
    generic (
        N : natural := 18
    );
    port (
        x_in : in signed(N-1 downto 0);
        y_in : in signed(N-1 downto 0);
        z_in : in signed(N-1 downto 0);
        shift : in integer;
```

```
        beta_i : in signed(N-1 downto 0);
        rot_0_vec_1 : in std_logic;
        x_out : out signed(N-1 downto 0);
        y_out : out signed(N-1 downto 0);
        z_out : out signed(N-1 downto 0)
    );
end;

architecture behavioral of add_sub is
    signal x_shift : signed(N-1 downto 0);
    signal y_shift : signed(N-1 downto 0);
    signal selec : std_logic;
begin
    x_shift <= shift_right(x_in, shift);
    y_shift <= shift_right(y_in, shift);

    -- selector para el bloque suma/resta
    selec <= z_in(N-1) when rot_0_vec_1 = '0' else (not y_in(N-1));

    x_out <= x_in - y_shift when (selec='0') else x_in + y_shift;
    y_out <= y_in + x_shift when (selec='0') else y_in - x_shift;
    z_out <= z_in - beta_i when (selec='0') else z_in + beta_i;
end;
```

3.2.d. Cordic

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity cordic_stage is
    generic (
        ITERS : natural := 16
    );
    port (
        clk : in std_logic;
        rst : in std_logic;
        rot_0_vec_1 : in std_logic;
        start : in std_logic;
        x_in : in std_logic_vector(ITERS+1 downto 0);
        y_in : in std_logic_vector(ITERS+1 downto 0);
        z_in : in std_logic_vector(ITERS+1 downto 0);
        x_out : out std_logic_vector(ITERS+1 downto 0);
        y_out : out std_logic_vector(ITERS+1 downto 0);
        z_out : out std_logic_vector(ITERS+1 downto 0);
        ready : out std_logic
    );
end;

architecture behavioral of cordic_stage is
    constant N : natural := ITERS+2;

    type t_array_mux is array(0 to ITERS-1) of signed(N-1 downto 0);
    constant BETAS : t_array_mux := (
        to_signed(32768,N),
        to_signed(19344,N),
        to_signed(10221,N),
        to_signed(5188,N),
        to_signed(2604,N),
        to_signed(1303,N),
        to_signed(652,N),
        to_signed(326,N),
        to_signed(163,N),
        to_signed(81,N),
        to_signed(41,N),
```

```
        to_signed(20,N),
        to_signed(10,N),
        to_signed(5,N),
        to_signed(3,N),
        to_signed(1,N)
    );

    signal x_reg : std_logic_vector(N-1 downto 0);
    signal y_reg : std_logic_vector(N-1 downto 0);
    signal z_reg : std_logic_vector(N-1 downto 0);

    signal x_aux : signed(N-1 downto 0) := (others => '0');
    signal y_aux : signed(N-1 downto 0) := (others => '0');
    signal z_aux : signed(N-1 downto 0) := (others => '0');

    signal x_i :  signed(N-1 downto 0);
    signal y_i :  signed(N-1 downto 0);
    signal z_i :  signed(N-1 downto 0);
    signal beta_i : signed(N-1 downto 0);

    signal iteration : integer range 0 to ITERS-1 := 0;


begin

    beta_i <= BETAS(iteration);

    process(clk, rst)
    begin
        if rst='1' then
            x_i <= (others=>'0');
            y_i <= (others=>'0');
            z_i <= (others=>'0');
        elsif rising_edge(clk) then
            if start='1' then
                x_i <= signed(x_in);
                y_i <= signed(y_in);
                z_i <= signed(z_in);
            else
                x_i <= x_aux;
                y_i <= y_aux;
                z_i <= z_aux;
            end if;
        end if;
    end process;

    counter : entity work.counter_N(behavioral2)
                    generic map (
                        N => ITERS
                    )
                    port map (
                        clk_i => clk,
                        rst_i => rst,
                        restart => start,
                        count => iteration
                    );

    add_sub_logic: entity work.add_sub(behavioral)
                        generic map (
                            N => N
                        )
                        port map (
                            x_in => x_i,
                            y_in => y_i,
```

```
                                z_in => z_i,
                                shift => iteration,
                                beta_i => beta_i,
                                rot_0_vec_1 => rot_0_vec_1,
                                x_out => x_aux,
                                y_out => y_aux,
                                z_out => z_aux
                            );

    x_out <= std_logic_vector(x_aux);
    y_out <= std_logic_vector(y_aux);
    z_out <= std_logic_vector(z_aux);
    ready <= '1' when iteration=ITERS-1 else '0';
end;
```

3.2.e. Entidad superior

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity cordic_top is
    generic (
        ITERS : natural := 16
    );
    port (
        clk : in std_logic;
        rst : in std_logic;
        rot_0_vec_1 : in std_logic;
        start : in std_logic;
        x_in : in std_logic_vector(ITERS+1 downto 0);
        y_in : in std_logic_vector(ITERS+1 downto 0);
        z_in : in std_logic_vector(ITERS+1 downto 0);
        x_out : out std_logic_vector(ITERS+1 downto 0);
        y_out : out std_logic_vector(ITERS+1 downto 0);
        z_out : out std_logic_vector(ITERS+1 downto 0);
        ready : out std_logic
    );
end;

architecture behavioral of cordic_top is
signal x_i_ap : std_logic_vector(ITERS+1 downto 0);
signal y_i_ap : std_logic_vector(ITERS+1 downto 0);
signal z_i_ap : std_logic_vector(ITERS+1 downto 0);

signal x_o_bp : std_logic_vector(ITERS+1 downto 0);
signal y_o_bp : std_logic_vector(ITERS+1 downto 0);
signal z_o_bp : std_logic_vector(ITERS+1 downto 0);

signal x_o_ap : std_logic_vector(2*(ITERS+1)+1 downto 0);
signal y_o_ap : std_logic_vector(2*(ITERS+1)+1 downto 0);

signal GAIN : signed(ITERS+1 downto 0) := "010011011010111101"; -- 79594



begin
    PRE: entity work.pre_cordic(behavioral)
        generic map(
            N => ITERS+2
        )
        port map (
            x_in => x_in,
            y_in => y_in,
            z_in => z_in,
```

```
            rot_0_vec_1 => rot_0_vec_1,
            x_out => x_i_ap,
            y_out => y_i_ap,
            z_out => z_i_ap
        );

    CORDIC: entity work.cordic_stage(behavioral)
        generic map(
            ITERS => ITERS
        )
        port map (
            clk => clk,
            rst => rst,
            rot_0_vec_1 => rot_0_vec_1,
            start => start,
            x_in => x_i_ap,
            y_in => y_i_ap,
            z_in => z_i_ap,
            x_out => x_o_bp,
            y_out => y_o_bp,
            z_out => z_o_bp,
            ready => ready
        );

    POST:

    x_o_ap <= std_logic_vector(signed(x_o_bp)*GAIN);
    y_o_ap <= std_logic_vector(signed(y_o_bp)*GAIN);

    x_out <= x_o_ap(2*ITERS+2 downto ITERS+1);
    y_out <= y_o_ap(2*ITERS+2 downto ITERS+1);
    z_out <= z_o_bp;

end;
```

3.2.f. Testbench

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.math_real;
use std.textio.all;

entity testbench is
end;

architecture behavioral of testbench is
    constant ITERS: natural := 16;

    signal clk : std_logic := '0';
    signal rst : std_logic := '1';

    signal rot_0_vec_1 : std_logic_vector(1 downto 0);
    signal start : std_logic;
    signal ready : std_logic;

    signal x_in : std_logic_vector(ITERS+1 downto 0);
    signal y_in : std_logic_vector(ITERS+1 downto 0);
    signal z_in : std_logic_vector(ITERS+1 downto 0);
    signal x_out : std_logic_vector(ITERS+1 downto 0);
    signal y_out : std_logic_vector(ITERS+1 downto 0);
    signal z_out : std_logic_vector(ITERS+1 downto 0);


    signal cycle_count : integer := 0;
```

```
29
30      file data : text open read_mode is "D:\Documentos\Fiuba\2C2023\SistemasDigitales\TP3\testbench\t
31
32  begin
33
34      clk <= not clk after 10 us;
35      rst <= '0' after 2 us;
36
37      Test_Sequence: process
38          variable l : line;
39          variable ch : character := ' ';
40          variable aux : integer;
41      begin
42          while not(endfile(data)) loop
43              wait until rising_edge(clk);
44              cycle_count <= cycle_count +1;
45              -- se lee una linea del archivo
46              readline(data, l);
47              -- se extrae un entero de la linea
48              read(l,aux);
49              -- se carga el valor del operando X
50              x_in <= std_logic_vector(to_unsigned(aux,ITERS+2));
51              -- se lee el espacio
52              read(l, ch);
53              -- se lee otro entero de la linea
54              read(l, aux);
55              -- se carga el valor del operando Y
56              y_in <= std_logic_vector(to_unsigned(aux,ITERS+2));
57              -- se lee otro espacio
58              read(l, ch);
59              -- se lee otro entero de la linea
60              read(l, aux);
61              -- se carga el operando z
62              z_in <= std_logic_vector(to_unsigned(aux,ITERS+2));
63
64              -- se lee otro espacio
65              read(l, ch);
66              -- se lee otro entero de la linea
67              read(l, aux);
68              -- se carga el operando z
69              rot_0_vec_1 <= std_logic_vector(to_unsigned(aux,2));
70
71
72
73              start <= '1';
74              wait until rising_edge(clk);
75              start <= '0';
76              wait until ready='1';
77              wait until rising_edge(clk);
78
79            end loop;
80
81            file_close(data); -- cierro el archivo
82
83            -- abort
84            assert false report
85                 "Fin de la simulacion" severity failure;
86
87      end process Test_Sequence;
88
89     DUV: entity work.cordic_top(behavioral)
90                  generic map (
91                      ITERS => ITERS
92                  )
```

```
                port map(
                    clk => clk,
                    rst => rst,
                    rot_0_vec_1 => rot_0_vec_1(0),
                    start => start,
                    x_in => x_in,
                    y_in => y_in,
                    z_in => z_in,
                    x_out => x_out,
                    y_out => y_out,
                    z_out => z_out,
                    ready => ready
                );

end;
```

3.3. Arquitectura Desenrollada

3.3.a. Registro

Como se explicó previamente, al ralizar *pipelining* se necesita registrar la salida de cada etapa. A continuación se muestra la descripción en VHDL de la entidad de registro.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity reg is
    generic(N: integer := 4);
    port (
        clk : in std_logic;
        rst : in std_logic;
        ena : in std_logic;
        D : in std_logic_vector(N-1 downto 0);
        Q : out std_logic_vector(N-1 downto 0)
    );
end;

architecture behavioral of reg is
begin
    process(clk, rst)
    begin
        if rst='1' then
            Q <= (others => '0');
        elsif rising_edge(clk) then
            if ena='1' then
                Q <= D;
            end if;
        end if;
    end process;
end;
```

3.3.b. Cordic

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity cordic_unrolled_stage is
    generic (
        ITERS : natural := 16;
        PIPELINE : boolean := True
    );
    port (
        clk : in std_logic;
```

```
        rst : in std_logic;
        rot_0_vec_1_in : in std_logic;
        x_in : in std_logic_vector(ITERS+1 downto 0);
        y_in : in std_logic_vector(ITERS+1 downto 0);
        z_in : in std_logic_vector(ITERS+1 downto 0);
        x_out : out std_logic_vector(ITERS+1 downto 0);
        y_out : out std_logic_vector(ITERS+1 downto 0);
        z_out : out std_logic_vector(ITERS+1 downto 0);
        rot_0_vec_1_out : out std_logic
    );
end;

architecture behavioral of cordic_unrolled_stage is
    constant N : natural := ITERS+2;
    type signed_matrix is array(integer range <>) of signed(N-1 downto 0);
    type t_matrix is array(integer range <>) of std_logic_vector;

    constant BETAS : signed_matrix(0 to ITERS-1) := (
        to_signed(32768,N),
        to_signed(19344,N),
        to_signed(10221,N),
        to_signed(5188,N),
        to_signed(2604,N),
        to_signed(1303,N),
        to_signed(652,N),
        to_signed(326,N),
        to_signed(163,N),
        to_signed(81,N),
        to_signed(41,N),
        to_signed(20,N),
        to_signed(10,N),
        to_signed(5,N),
        to_signed(3,N),
        to_signed(1,N)
    );

    signal x_i :  t_matrix(ITERS downto 0)(N-1 downto 0);
    signal y_i :  t_matrix(ITERS downto 0)(N-1 downto 0);
    signal z_i :  t_matrix(ITERS downto 0)(N-1 downto 0);

    signal x_o :  t_matrix(ITERS-1 downto 0)(N-1 downto 0);
    signal y_o :  t_matrix(ITERS-1 downto 0)(N-1 downto 0);
    signal z_o :  t_matrix(ITERS-1 downto 0)(N-1 downto 0);

    signal r0_v1 : std_logic_vector(ITERS downto 0);
    signal rv_aux : t_matrix(ITERS-1 downto 0)(1 downto 0);

begin
    x_i(0) <= x_in;
    y_i(0) <= y_in;
    z_i(0) <= z_in;
    r0_v1(0) <=  rot_0_vec_1_in;

    x_out <= x_i(ITERS);
    y_out <= y_i(ITERS);
    z_out <= z_i(ITERS);
    rot_0_vec_1_out <= r0_v1(ITERS);


    cordic: for i in 0 to ITERS-1 generate
        cs: entity work.add_sub(behavioral)
                generic map (
                    N => N
                )
```

```
76                  port map (
77                      x_in => x_i(i),
78                      y_in => y_i(i),
79                      z_in => z_i(i),
80                      shift => i,
81                      beta_i => std_logic_vector(BETAS(i)),
82                      rot_0_vec_1 => r0_v1(i),
83                      x_out => x_o(i),
84                      y_out => y_o(i),
85                      z_out => z_o(i)
86                  );
87
88      pipeline_condition: if PIPELINE generate
89      begin
90          regX: entity work.reg(behavioral)
91              generic map(N)
92              port map(
93                  clk => clk,
94                  rst => rst,
95                  ena => '1',
96                  D => std_logic_vector(x_o(i)),
97                  Q => x_i(i+1)
98              );
99
100         regY: entity work.reg(behavioral)
101             generic map(N)
102             port map(
103                 clk => clk,
104                 rst => rst,
105                 ena => '1',
106                 D => std_logic_vector(y_o(i)),
107                 Q => y_i(i+1)
108             );
109
110
111         regZ: entity work.reg(behavioral)
112         generic map(N)
113         port map(
114             clk => clk,
115             rst => rst,
116             ena => '1',
117             D => std_logic_vector(z_o(i)),
118             Q => z_i(i+1)
119         );
120
121         regRV: entity work.reg(behavioral)
122         generic map(2)
123         port map(
124             clk => clk,
125             rst => rst,
126             ena => '1',
127             D => '0' & r0_v1(i),
128             Q => rv_aux(i)
129         );
130         r0_v1(i+1) <= rv_aux(i)(0);
131
132         else generate
133             x_i(i+1) <= x_o(i);
134             y_i(i+1) <= y_o(i);
135             z_i(i+1) <= z_o(i);
136             r0_v1(i) <= rot_0_vec_1_in;
137         end generate;
138
139     end generate;
```

```
140
141 end;
```

Cabe aclarar que la sintaxis:

```
if PIPELINE generate

else generate
```

está disponible unicamente para versiones de VHDL superiores a la 2008.

3.4. Entidad Superior

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  entity cordic_ur_top is
6      generic (
7          ITERS : natural := 16;
8          PIPELINE : boolean := True
9      );
10     port (
11         clk : in std_logic;
12         rst : in std_logic;
13         rot_0_vec_1_in : in std_logic;
14         x_in : in std_logic_vector(ITERS+1 downto 0);
15         y_in : in std_logic_vector(ITERS+1 downto 0);
16         z_in : in std_logic_vector(ITERS+1 downto 0);
17         x_out : out std_logic_vector(ITERS+1 downto 0);
18         y_out : out std_logic_vector(ITERS+1 downto 0);
19         z_out : out std_logic_vector(ITERS+1 downto 0);
20         rot_0_vec_1_out : out std_logic
21     );
22 end;
23
24 architecture behavioral of cordic_ur_top is
25 signal x_i_ap : std_logic_vector(ITERS+1 downto 0);
26 signal y_i_ap : std_logic_vector(ITERS+1 downto 0);
27 signal z_i_ap : std_logic_vector(ITERS+1 downto 0);
28
29 signal x_o_bp : std_logic_vector(ITERS+1 downto 0);
30 signal y_o_bp : std_logic_vector(ITERS+1 downto 0);
31 signal z_o_bp : std_logic_vector(ITERS+1 downto 0);
32
33 signal x_o_ap : std_logic_vector(2*(ITERS+1)+1 downto 0);
34 signal y_o_ap : std_logic_vector(2*(ITERS+1)+1 downto 0);
35
36 signal GAIN : signed(ITERS+1 downto 0) := "0100110110101011101"; -- 79594
37
38
39
40 begin
41
42     PRE: entity work.pre_cordic(behavioral)
43         generic map(
44             N => ITERS+2
45         )
46         port map (
47             x_in => x_in,
48             y_in => y_in,
49             z_in => z_in,
50             rot_0_vec_1 => rot_0_vec_1_in,
51             x_out => x_i_ap,
52             y_out => y_i_ap,
```

```
            z_out => z_i_ap
        );

    CORDIC: entity work.cordic_unrolled_stage(behavioral)
        generic map(
            ITERS => ITERS,
            PIPELINE => PIPELINE
        )
        port map (
            clk => clk,
            rst => rst,
            rot_0_vec_1_in => rot_0_vec_1_in,
            x_in => x_i_ap,
            y_in => y_i_ap,
            z_in => z_i_ap,
            x_out => x_o_bp,
            y_out => y_o_bp,
            z_out => z_o_bp,
            rot_0_vec_1_out => rot_0_vec_1_out
        );

    POST:

    x_o_ap <= std_logic_vector(signed(x_o_bp)*GAIN);
    y_o_ap <= std_logic_vector(signed(y_o_bp)*GAIN);

    x_out <= x_o_ap(2*ITERS+2 downto ITERS+1);
    y_out <= y_o_ap(2*ITERS+2 downto ITERS+1);
    z_out <= z_o_bp;

end;
```

3.5. Generación de delay

Se utilizó este componente para poder alinear las entradas con las salidas durante la simulación.

```
library ieee;
use ieee.std_logic_1164.all;

entity delay_gen is
    generic(
            N: natural:= 18;
            DELAY: natural:= 0
        );
    port(
        clk: in std_logic;
        rst: in std_logic;
        A: in std_logic_vector(N-1 downto 0);
        B: out std_logic_vector(N-1 downto 0)
    );
end;


architecture behavioral of delay_gen is
    type std_logic_matrix is array(DELAY-1 downto 0) of std_logic_vector(N-1 downto 0);
    signal sr : std_logic_matrix;
begin
    process(clk,rst)
    begin
        if rst='1' then
            sr <= (others=>(others=>'0'));
        elsif clk = '1' and clk'event then
            sr(DELAY-1 downto 1) <= sr(DELAY-2 downto 0);
            sr(0) <= A;
        end if;
    end process;
```

```
    B <= sr(DELAY-1);
end;
```

3.5.a. Testbench

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity tb_cordic_unrolled is
end;

architecture behavioral of tb_cordic_unrolled is
    constant N : natural := 16;
    signal clk : std_logic := '0';
    signal rst : std_logic := '1';
    signal x_in : std_logic_vector(N+1 downto 0) := std_logic_vector(to_signed(10000,N+2));
    signal y_in : std_logic_vector(N+1 downto 0) := std_logic_vector(to_signed(10000,N+2));
    signal z_in : std_logic_vector(N+1 downto 0) := std_logic_vector(to_signed(32768,N+2));

    signal x_aux : std_logic_vector(N+1 downto 0);
    signal y_aux : std_logic_vector(N+1 downto 0);
    signal z_aux : std_logic_vector(N+1 downto 0);

    signal x_out : std_logic_vector(N+1 downto 0);
    signal y_out : std_logic_vector(N+1 downto 0);
    signal z_out : std_logic_vector(N+1 downto 0);

    signal rot_0_vec_1_in : std_logic := '0';
    signal rot_0_vec_1_out : std_logic;
begin
    clk <= not clk after 10 us;
    rst <= '0' after 2 us;

    x_in <= std_logic_vector(to_signed(-10000,N+2)) after 150 us;
    y_in <= std_logic_vector(to_signed(10000,N+2)) after 150 us;
    z_in <= std_logic_vector(to_signed(65536,N+2)) after 30 us, std_logic_vector(to_signed(98304,N+2)
            std_logic_vector(to_signed(163840,N+2)) after 90 us, std_logic_vector(to_signed(0,N+2)) a
    rot_0_vec_1_in <= '1' after 120 us;

    cordic: entity work.cordic_ur_top(behavioral)
                generic map (
                    ITERS => N,
                    PIPELINE => True
                )
                port map(
                    clk => clk,
                    rst => rst,
                    rot_0_vec_1_in  => rot_0_vec_1_in,
                    x_in => x_in,
                    y_in => y_in,
                    z_in => z_in,
                    x_out => x_out,
                    y_out => y_out,
                    z_out => z_out,
                    rot_0_vec_1_out => rot_0_vec_1_out
                );
         -- Instanciacion de la linea de retardo
    delz: entity work.delay_gen(behavioral)
        generic map(N+2, N)
        port map(clk, '0', std_logic_vector(z_in), z_aux);
    -- Verificacion de la condicion

        -- Instanciacion de la linea de retardo
```

```
60      delx: entity work.delay_gen(behavioral)
61          generic map(N+2, N)
62          port map(clk, '0', std_logic_vector(x_in), x_aux);
63      -- Verificacion de la condicion
64
65          -- Instanciacion de la linea de retardo
66      dely: entity work.delay_gen(behavioral)
67          generic map(N+2, N)
68          port map(clk, '0', std_logic_vector(y_in), y_aux);
69      -- Verificacion de la condicion
70  end;
```

4. Simulación

En esta sección se muestran los resultados obtenidos de realizar la simulación de comportamiento sobre los *testbenchs*.

Para ambas arquitecturas los valores *testeados* son los siguientes:

Xin	Yin	Zin	Xout	Yout
10000	10000	32768	0	14142
10000	10000	65563	-10000	10000
10000	10000	98304	-14142	0
10000	10000	163840	0	- 14142

Tabla 1: Valores del testbench, modo rotación

Xin	Yin	Zin	Xout	Zout
10000	10000	0	14142	32768
-10000	10000	0	14142	98304

Tabla 2: Valores del testbench, modo vectorización

En ambos casos se eligieron valores que permitan ver tanto el algoritmo como el pre-procesamiento.

4.1. Arquitectura iterativa

A continuación se muestran las simulaciones de las señales para el modo rotación.

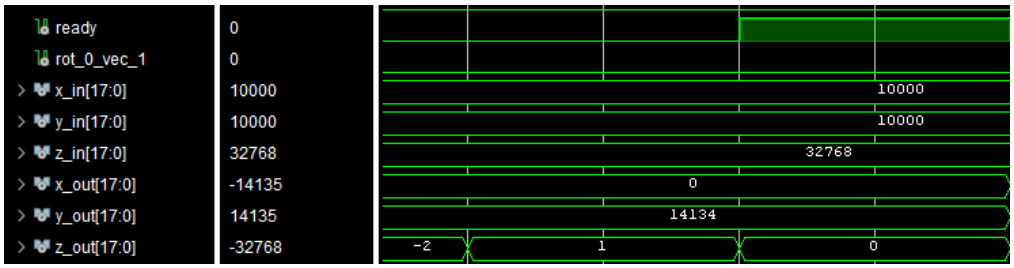

Figura 7: Simulación del modo rotación para $z_{in} = 32768$

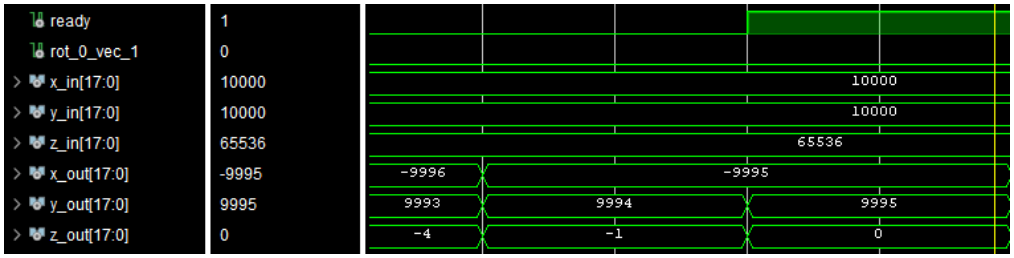

Figura 8: Simulación del modo rotación para $z_{in} = 65563$

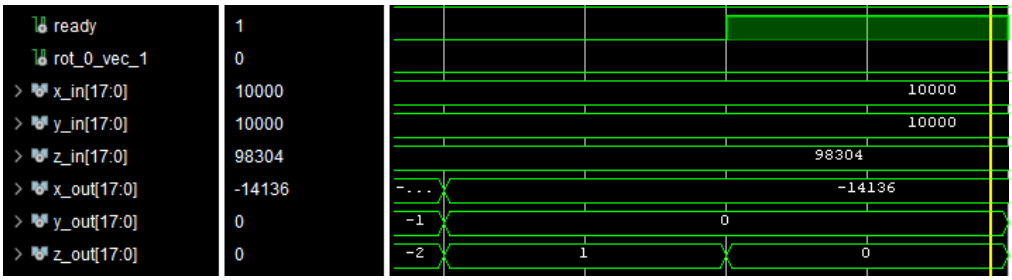

Figura 9: Simulación del modo rotación para $z_{in} = 98304$

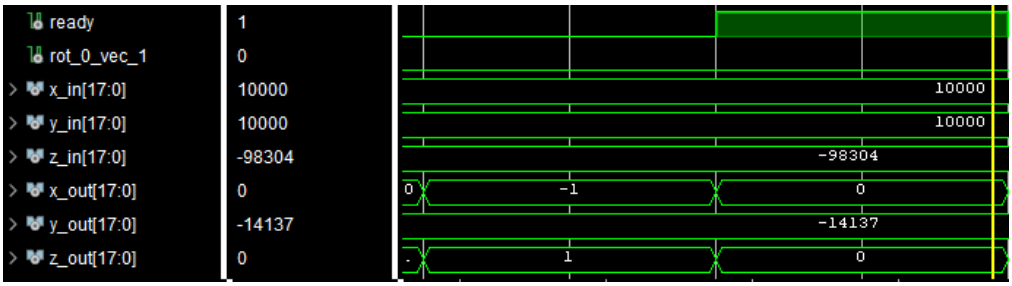

Figura 10: Simulación del modo rotación para $z_{in} = 163840$

Como se puede observar, hay una leve diferencia entre los valores esperados y los obtenidos, debida al redondeo producido por la precisión del algoritmo.

En las siguientes figuras se muestran las simulaciones para el modo vectorización.

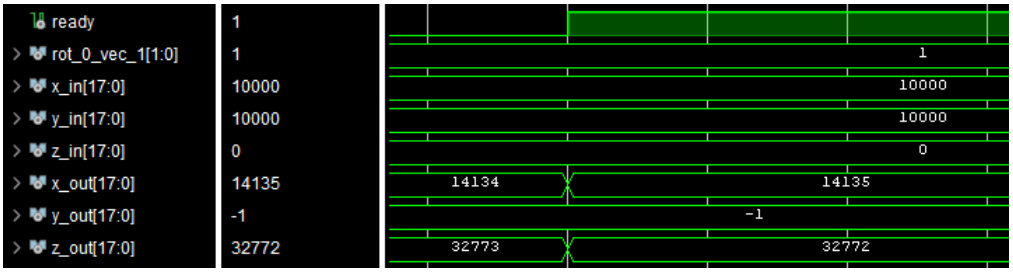

Figura 11: Simulación del modo vectorización para $x_{in} = 10000$, $y_{in} = 10000$

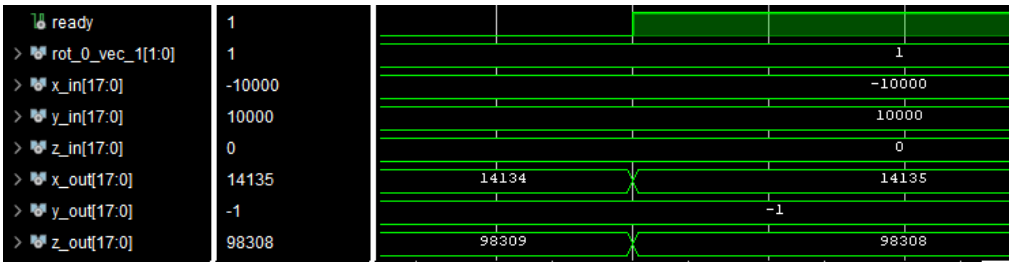

Figura 12: Simulación del modo vectorización para $x_{in} = -10000$, $y_{in} = 10000$

Nuevamente se observan leves diferencias, pero se puede concluir que los valores obtenidos son correctos.

4.2. Arquitectura desenrollada

A continuación se muestran los resultados obtenidos de la simulación de la arquitectura desenrollada, tanto para el modo de rotación como el modo de vectorización.

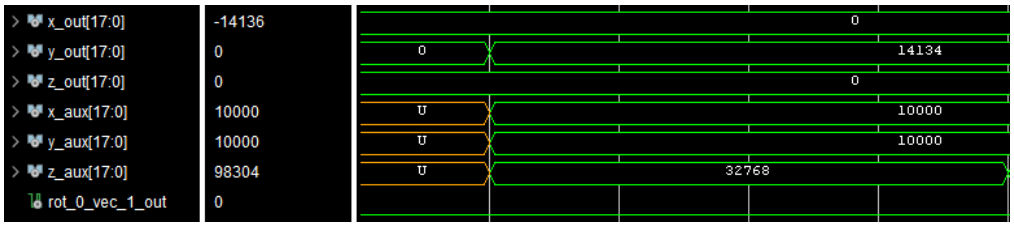

Figura 13: Simulación del modo rotación para $z_{in} = 32768$

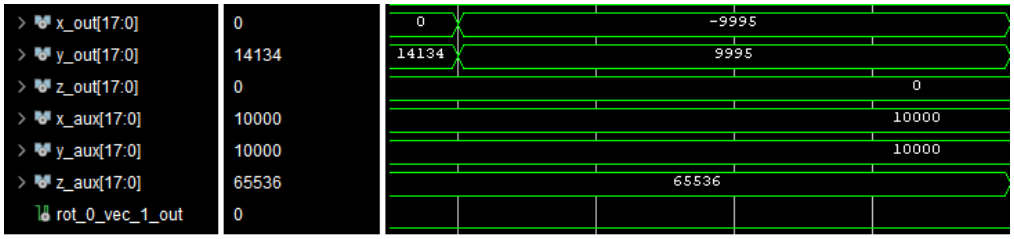

Figura 14: Simulación del modo rotación para $z_{in} = 65563$

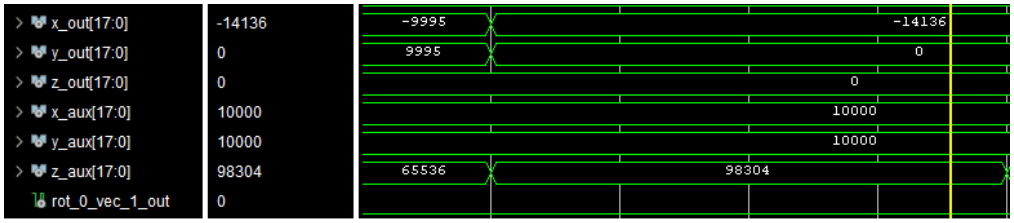

Figura 15: Simulación del modo rotación para $z_{in} = 98304$

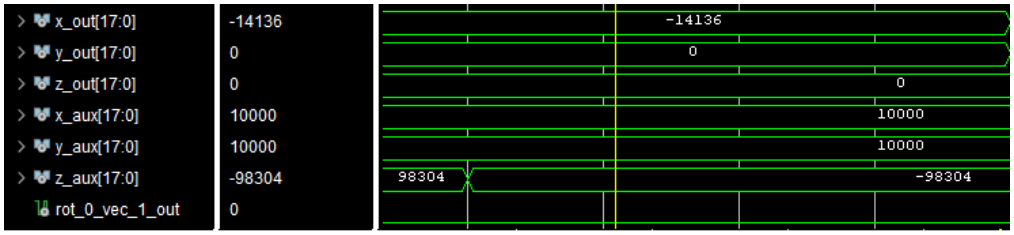

Figura 16: Simulación del modo rotación para $z_{in} = 163840$

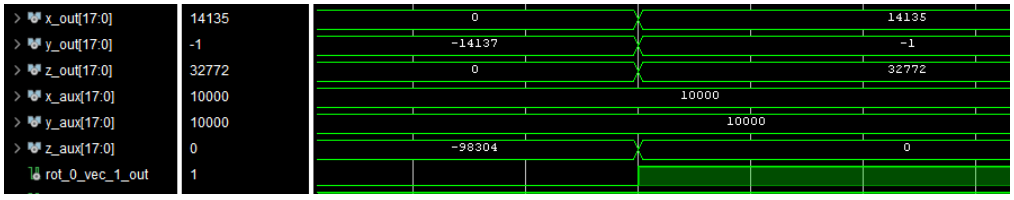

Figura 17: Simulación del modo vectorización para $x_{in} = 10000$, $y_{in} = 10000$

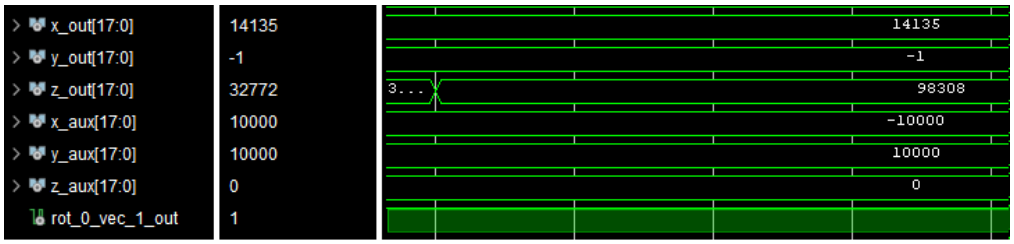

Figura 18: Simulación del modo vectorización para $x_{in} = -10000$, $y_{in} = 10000$

Como se puede observar, se obtuvieron resultados similares para ambas arquitecturas.

5. Síntesis

La síntesis del diseño se hizo sobre la FPGA xc7a15tftg256-1, utilizando Vivado. En primer lugar se muestra el RTL de cada arquitectura y luego se muestran los recursos utilizados en cada caso.

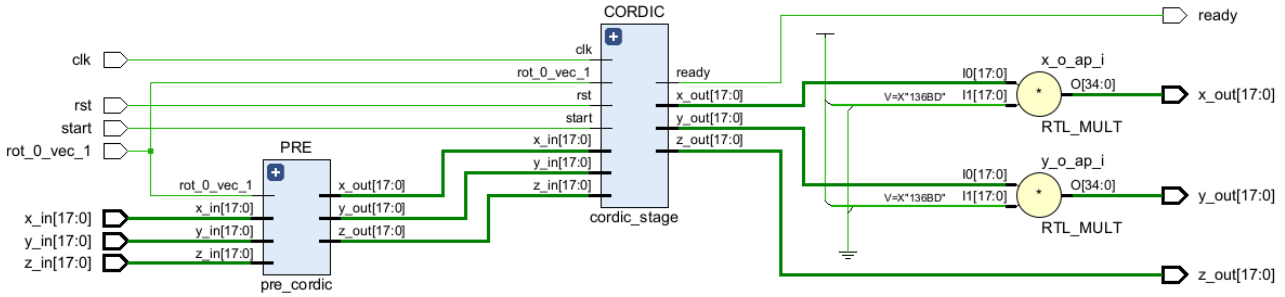

Figura 19: RTL para la Arquitectura Iterativa

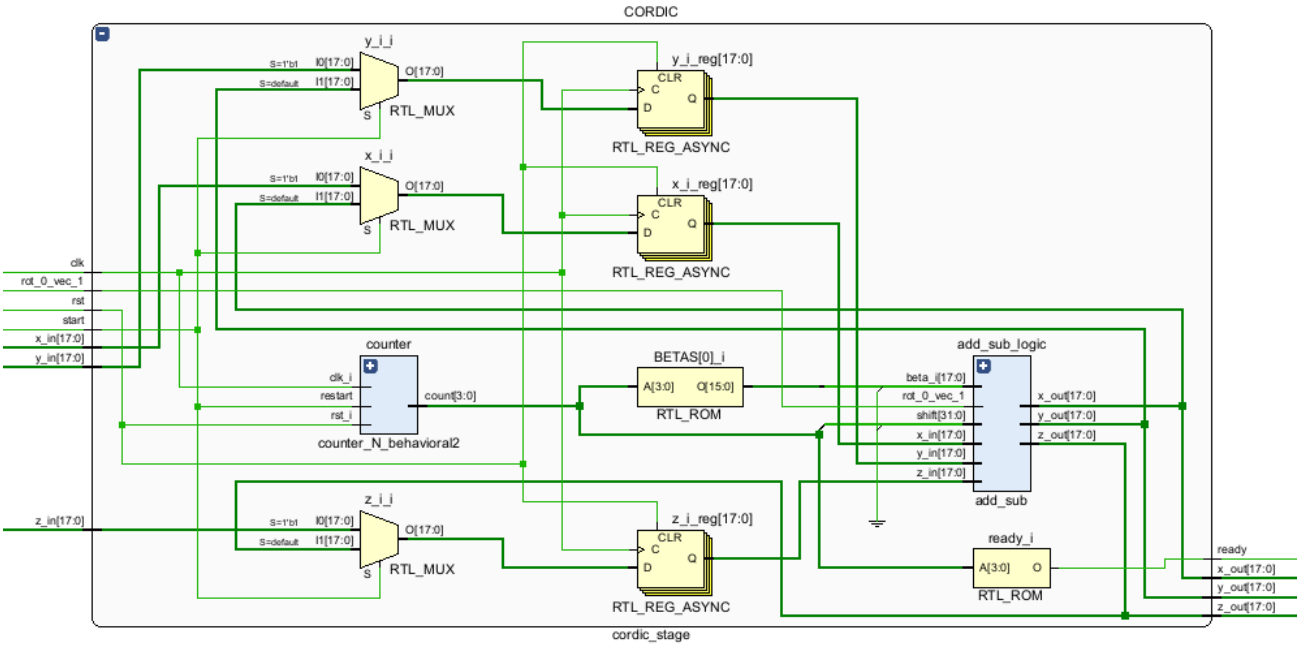

Figura 20: RTL del componente CORDIC para la Arquitectura Iterativa

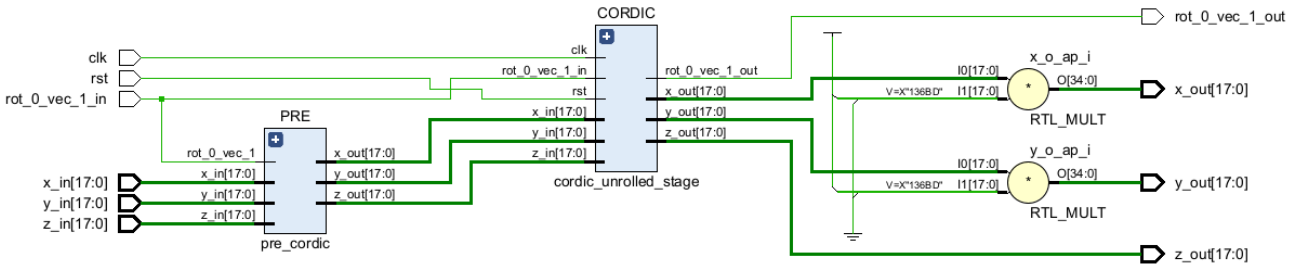

Figura 21: RTL para la Arquitectura Desenrollada

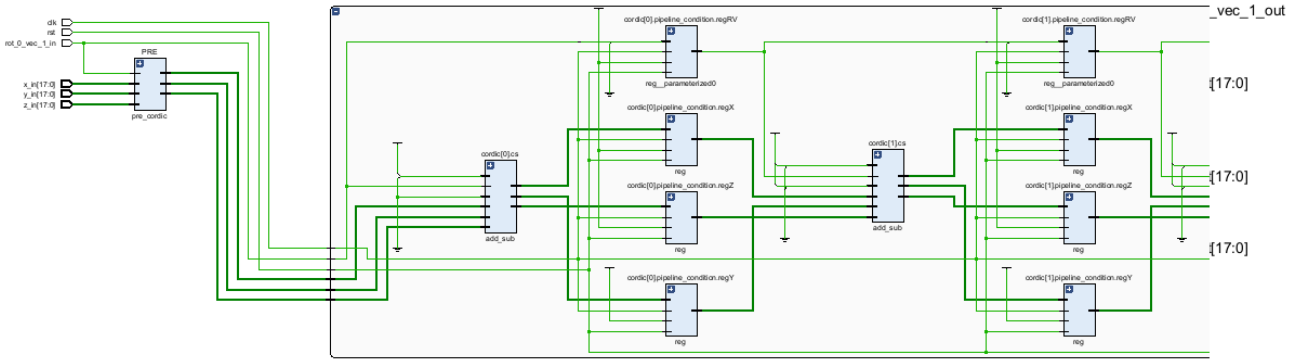

Figura 22: Parte del RTL del componente CORDIC para la Arquitectura Iterativa

En el caso de la arquitectura desenrollada no es posible visualizar de manera completa la entidad CORDIC dado que son componentes concatenados N veces, pero de todos modos se muestra una parte para poder observar cómo la salida de una etapa se conecta a la entrada de la próxima.

A continuación se muestra la utilización de recursos para cada arquitectura

Resource	Utilization	Available	Utilization %
LUT	318	10400	3.06
FF	86	20800	0.41
DSP	2	45	4.44
IO	113	170	66.47

Figura 23: Uso de recursos de la Arquitectura Iterativa

Resource	Utilization	Available	Utilization %
LUT	965	10400	9.28
FF	880	20800	4.23
DSP	2	45	4.44
IO	112	170	65.88

Figura 24: Uso de recursos de la Arquitectura Desenrollada

La arquitectura desenrollada presenta un uso claramente mayor de los recursos de la FPGA, dado que se generaron N etapas de la unidad de CORDIC, en vez de reutilizar una única etapa como ocurre en la arquitectura iterativa.

6. Conclusión

En este trabajo se cumplió el objetivo de describir, simular y sintetizar el algoritmo CORDIC. Se pudo apreciar su utilidad y su simpleza, dado que se puede implementar mediante operaciones del tipo suma/resta/desplazamiento y logra

la producción de funciones trigonómetricas, hiperbólicas y cambios de coordenadas de un sistema polar a uno cartesiano y viceversa.

Además, se pudo realizar una comparación entre ambas arquitecturas implementadas. Por un lado, la arquitectura iterativa utiliza menos recursos físicos en la FPGA. Sin embargo, la arquitectura desenrollada con el uso de *pipeline* permite realizar cambios en la entrada sobre la marcha, evitando la necesidad de esperar a que se termine de computar un valor para que se empiece a calcular el próximo. Esto es especialmente útil cuando se tiene un gran set de datos a calcular. La desventaja de esta arquitectura se presenta en un mayor uso de recursos de la placa.