Universidad de Buenos Aires
Facultad de Ingeniería

Sistemas Digitales (86.41)

Trabajo practico Nº3:

CORDIC

Integrantes:

Puy Gonzalo 99784 gpuy@fi.uba.ar

1er cuatrimestre 2022

1. Introducción

El presente trabajo tiene como objetivo describir e implementar en FPGA la arquitectura CORDIC. Dicha arquitectura sera descripta en lenguaje VHDL, simulada mediante *GTKwave* y sintetizada sobre el dispositivo `xc7a15tftg256-1`.

El algoritmo CORDIC, es utilizado para la rotación de vectores, el calculo de funciones trigonométricas, la transformación de coordenadas, entre otras.

Para el caso de este trabajo, solo se lo utilizará para rotar vectores.

2. Desarrollo

El algoritmo CORDIC en este trabajo será utilizado para la rotación de vectores en el plano xy. Este algoritmo tiene dos modos de operación, los cuales son rotación y vectorización.

El modo rotación rota el vector en un ángulo especificado. El acumulador angular se inicializa con el ángulo a rotar y la decisión de rotación en cada iteración se lleva a cabo de tal manera de disminuir el ángulo residual en el acumulador angular (se utiliza su signo).

Se puede resumir este modo con las siguientes ecuaciones:

$$\begin{cases} x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i} \\ y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i} \\ z_{i+1} = z_i - d_i \cdot \arctan\left(2^{-i}\right) \end{cases} \tag{1}$$

Donde: $d_i = -1$ si $z_i < 0$, $+1$ en otro caso.

Y finalmente se tiene

$$\begin{cases} x_n = A_n(x_0 \cdot \cos\left(z_0\right) - y_0 \cdot \sin\left(z_0\right)) \\ y_n = A_n(y_0 \cdot \cos\left(z_0\right) + x_0 \cdot \sin\left(z_0\right)) \\ z_n = 0 \end{cases} \tag{2}$$

En cambio en el modo vectorización se rota un vector hacia el eje de coordenadas x, guardando los ángulos requeridos para lograrlo. Busca minimizar la componente y del vector residual y la dirección de rotación se decide por el signo de la componente y residual.

Se puede resumir este modo con las siguientes ecuaciones:

$$\begin{cases} x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i} \\ y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i} \\ z_{i+1} = z_i - d_i \cdot \arctan\left(2^{-i}\right) \end{cases} \tag{3}$$

Donde: $d_i = +1$ si $y_i < 0$, -1 en otro caso.

Y finalmente se tiene

$$\begin{cases} x_n = A_n \cdot \sqrt{x_0^2 + y_0^2} \\ y_n = 0 \\ z_n = z_0 + \arctan\left(\frac{y_0}{x_0}\right) \end{cases} \tag{4}$$

El A_n mencionado en las ecuaciones (2) y (4) es la ganancia de CORDIC, la cual vale aproximadamente 1,647 cuando la cantidad de iteraciones es lo suficientemente grande.

2.1. Especificaciones de diseño

Se implementaron en lenguaje descriptor de hardware VHDL dos arquitecturas CORDIC trabajando en ambos modos: rotación y vectorización. En la siguiente figura se puede observar un diagrama en bloques aproximado del diseño

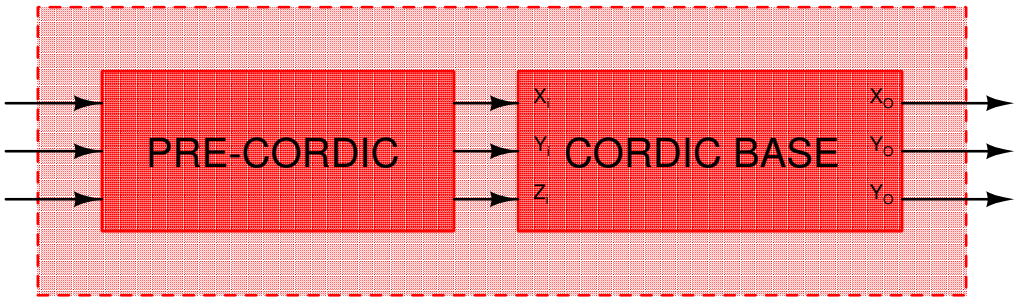

Figura 1: Diagrama en bloques

El bloque pre-cordic es una lógica combinacional que se encarga de (en modo rotación) negar las coordenadas xy originales para poder rotar ángulos mayores a 90°. Para el modo vectorización, este bloque no tiene funcionalidad alguna.

La primer arquitectura es iterativa y la segunda desenrrollada. A esta ultima se le agregó *pipelining* en las etapas intermedias.

En las siguientes figuras se muestra en diagrama de estas arquitecturas

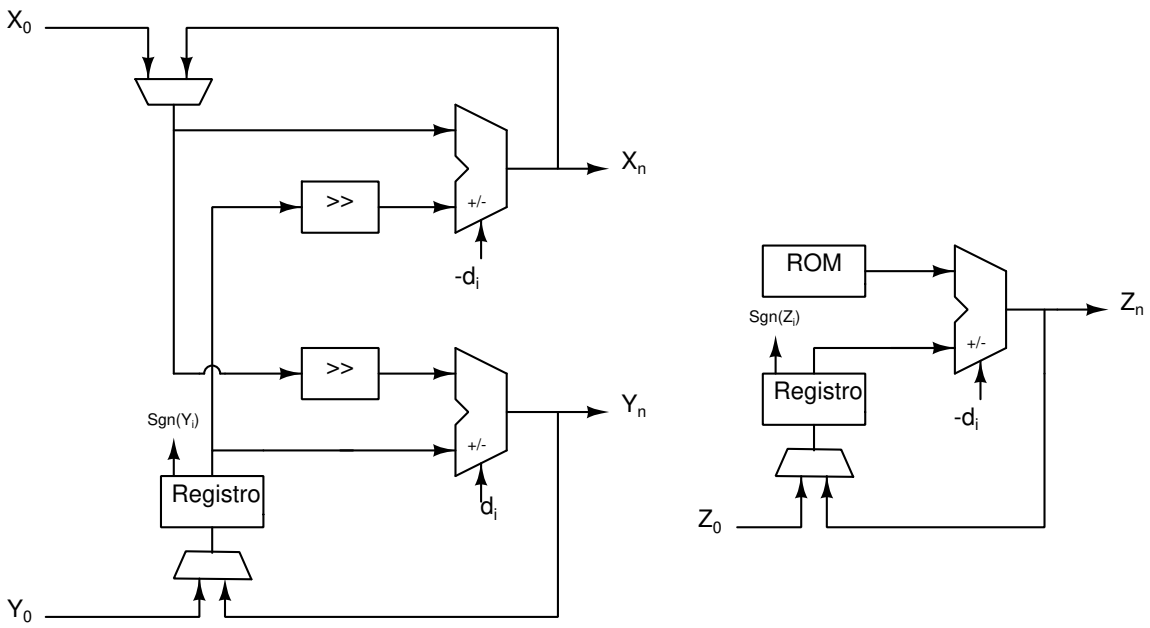

Figura 2: Arquitectura iterativa.

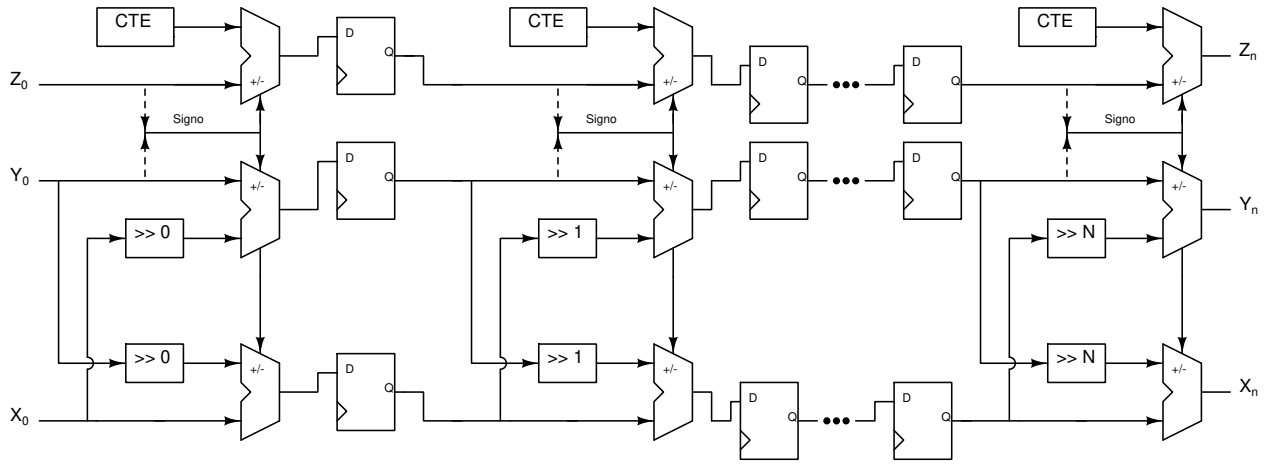

Figura 3: Arquitectura desenrrollada con *pipelining*.

2.2. Aclaraciones sobre el diseño

Se eligió realizar 16 iteraciones del algoritmo, por lo que tendremos que usar 16 bits para los datos. Además de esto se eligió un fondo de escala de 2^{13}, por lo que tanto los ángulos como las coordenadas de los vectores estarán escaladas por este valor. Es decir,

$$\begin{cases} \text{Coordenadas:} & x = x \cdot 2^{13} \\ \text{Ángulos:} & \text{Ang} = \dfrac{\text{Ang}}{\arctan(1)} \cdot 2^{13} \end{cases} \tag{5}$$

También se decidió no utilizar un bloque post-cordic. Este estaría encargado de quitar de las coordenadas finales, la ganancia de cordic. Pero al no requerirse esta funcionalidad se decidió no implementar este bloque.

3. Implementación en VHDL

A continuación se muestra el código utilizado para implementar ambas arquitecturas en VHDL

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use ieee.math_real.all;
5
6  entity cordic is
7      generic(
8          N : natural := 16
9      );
10     port(
11         clk             : in std_logic;
12         rst             : in std_logic;
13         req             : in std_logic;
14         rot0_vec1       : in std_logic;
```

```
15          x_i, y_i, z_i   : in std_logic_vector(N-1 downto 0);
16          ack             : out std_logic;
17          x_o, y_o, z_o   : out std_logic_vector(N-1 downto 0)
18      );
19  end cordic;
20
21
22  architecture behavioral_iter of cordic is
23
24      -- Entradas y salidas de Pre - Cordic
25      signal x_i_precordic, y_i_precordic, z_i_precordic : std_logic_vector(N-1 downto 0);
26      signal x_o_precordic, y_o_precordic, z_o_precordic : std_logic_vector(N-1 downto 0);
27
28      -- Entradas y salidas de Cordic_iterativo
29      signal x_i_cordic, y_i_cordic, z_i_cordic : std_logic_vector(N-1 downto 0);
30      signal ack_aux : std_logic;
31
32
33  begin
34
35      x_i_precordic <= x_i;
36      y_i_precordic <= y_i;
37      z_i_precordic <= z_i;
38      ack <= ack_aux;
39
40      PRE_CORDIC: entity work.pre_cordic(behavioral)
41      generic map(N => N)
42      port map(
43          x_i => x_i_precordic,
44          y_i => y_i_precordic,
45          z_i => z_i_precordic,
46          rot0_vec1 => rot0_vec1,
47          x_o => x_o_precordic,
48          y_o => y_o_precordic,
49          z_o => z_o_precordic
50      );
51
52      CORDIC_I: entity work.cordic_iter(behavioral)
53      generic map(N => N)
54      port map(
55          clk       => clk,
56          rst       => rst,
57          req       => req,
58          ack       => ack_aux,
59          rot0_vec1 => rot0_vec1,
60          x_0       => x_o_precordic,
61          y_0       => y_o_precordic,
62          z_0       => z_o_precordic,
63          x_nm1     => x_o,
```

```
64          y_nm1     => y_o,
65          z_nm1     => z_o
66      );
67
68  end behavioral_iter;
69
70
71  architecture behavioral_unrolled of cordic is
72
73      -- Entradas y salidas de Pre - Cordic
74      signal x_i_precordic, y_i_precordic, z_i_precordic : std_logic_vector(N-1 downto 0);
75      signal x_o_precordic, y_o_precordic, z_o_precordic : std_logic_vector(N-1 downto 0);
76
77      -- Entradas y salidas de Cordic_iterativo
78      signal x_i_cordic, y_i_cordic, z_i_cordic : std_logic_vector(N-1 downto 0);
79      signal ack_aux : std_logic;
80
81
82
83  begin
84
85      x_i_precordic <= x_i;
86      y_i_precordic <= y_i;
87      z_i_precordic <= z_i;
88      ack <= ack_aux;
89
90
91      PRE_CORDIC: entity work.pre_cordic(behavioral)
92      generic map(N => N)
93      port map(
94          x_i       => x_i_precordic,
95          y_i       => y_i_precordic,
96          z_i       => z_i_precordic,
97          rot0_vec1 => rot0_vec1,
98          x_o       => x_o_precordic,
99          y_o       => y_o_precordic,
100         z_o       => z_o_precordic
101     );
102
103     CORDIC_UP: entity work.cordic_unrolled(behavioral)
104     generic map(
105         N => N
106     )
107     port map(
108         clk => clk,
109         rst => rst,
110         req => req,
111         ack => ack_aux,
112         rot0_vec1 => rot0_vec1,
```

```
113         x_0 => x_o_precordic,
114         y_0 => y_o_precordic,
115         z_0 => z_o_precordic,
116         x_nm1 => x_o,
117         y_nm1 => y_o,
118         z_nm1 => z_o
119     );
120 
121 end behavioral_unrolled;
```

3.1. Pre-Cordic

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  
5  entity pre_cordic is
6      generic(
7          N : natural := 16 -- cantidad de iteraciones que va a hacer el algoritmo
8      );
9      port(
10 
11         x_i        : in std_logic_vector(N-1 downto 0);
12         y_i        : in std_logic_vector(N-1 downto 0);
13         z_i        : in std_logic_vector(N-1 downto 0);
14         rot0_vec1  : in std_logic;
15         x_o        : out std_logic_vector(N-1 downto 0);
16         y_o        : out std_logic_vector(N-1 downto 0);
17         z_o        : out std_logic_vector(N-1 downto 0)
18     );
19 
20 end pre_cordic;
21 
22 architecture behavioral of pre_cordic is
23 
24     -- 2**(N-1) --> 180
25     -- 2**(N-2) --> 90
26     -- 2**(N-3) --> 45
27     constant ANG_180 : signed := to_signed(2**(N-1), N);
28     constant ANG_90 : signed := to_signed(2**(N-2), N);
29     constant ANG_45 : signed := to_signed(2**(N-3), N);
30     signal   x_o_rot0, y_o_rot0, z_o_rot0 : std_logic_vector(N-1 downto 0);
31     signal   x_o_vec1, y_o_vec1, z_o_vec1 : std_logic_vector(N-1 downto 0);
32 
33 begin
34 
35     x_o_vec1 <= x_i;
```

```
36      y_o_vec1 <= y_i;
37      z_o_vec1 <= z_i;
38
39
40      x_o_rot0 <= std_logic_vector(signed(not(x_i)) + 1) when std_logic_vector(abs(signed(z_i))) > std_lo
41                  x_i;
42
43      y_o_rot0 <= std_logic_vector(signed(not(y_i)) + 1) when std_logic_vector(abs(signed(z_i))) > std_lo
44                  y_i;
45
46      z_o_rot0 <= std_logic_vector( signed(z_i) - ANG_180 ) when signed(z_i) > ANG_90 else
47                  std_logic_vector( signed(z_i) + ANG_180 ) when signed(z_i) < -ANG_90 else
48                  z_i;
49
50
51      x_o <= x_o_rot0 when rot0_vec1 = '0' else
52             x_o_vec1;
53
54      y_o <= y_o_rot0 when rot0_vec1 = '0' else
55             y_o_vec1;
56
57      z_o <= z_o_rot0 when rot0_vec1 = '0' else
58             z_o_vec1;
59
60  end behavioral;
```

3.2. Arquitectura iterativa

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4   use ieee.math_real.all;
5
6   entity cordic_iter is
7       generic(
8           N : natural := 16 -- cantidad de iteraciones que va a hacer el algoritmo
9       );
10      port(
11          clk: in std_logic;
12          rst: in std_logic;
13          req: in std_logic;
14          ack: out std_logic;
15          rot0_vec1: in std_logic;
16          x_0 : in std_logic_vector(N-1 downto 0);
17          y_0 : in std_logic_vector(N-1 downto 0);
18          z_0 : in std_logic_vector(N-1 downto 0);
19          x_nm1 : out std_logic_vector(N-1 downto 0);
```

```
20          y_nm1 : out std_logic_vector(N-1 downto 0);
21          z_nm1 : out std_logic_vector(N-1 downto 0)
22      );
23  end cordic_iter;
24
25  architecture behavioral of cordic_iter is
26
27      constant CLOG2N : natural := natural(ceil(log2(real(N))));
28
29      signal count : unsigned(CLOG2N-1 downto 0);
30
31      signal x_i,y_i,z_i : std_logic_vector(N-1 downto 0);
32      signal x_ip1,y_ip1,z_ip1 : std_logic_vector(N-1 downto 0);
33      signal atan_2mi : std_logic_vector(N-3 downto 0);
34      signal ack_aux : std_logic;
35
36  begin
37
38      ROM:entity work.atan_rom(behavioral)
39          generic map(
40              ADD_W  => CLOG2N,
41              DATA_W => N-2
42          )
43          port map(
44              addr_i => std_logic_vector(count),
45              data_o => atan_2mi
46          );
47
48
49      CORDIC_BASE: entity work.cordic_base(behavioral)
50      generic map(
51          N => N,
52          CLOG2N => CLOG2N
53      )
54      port map(
55          num_iter => count,
56          rot0_vec1 => rot0_vec1,
57          x_i => x_i,
58          y_i => y_i,
59          z_i => z_i,
60          atan_2mi => atan_2mi,
61          x_ip1 => x_ip1,
62          y_ip1 => y_ip1,
63          z_ip1 => z_ip1
64      );
65
66      process(clk,rst)
67      begin
68          if rst = '1' then
```

```
69              count <= (others => '0');
70          elsif clk'event and clk = '1' then
71              if req = '1' then
72                  count <= (others => '0');
73              elsif count /= N-1 then
74                  count <= count + 1;
75              end if;
76          end if;
77      end process;
78
79      ack_aux <= '1' when  count = N-1 else
80                 '0';
81
82      ack <= ack_aux;
83
84      process(clk,rst)
85      begin
86          if rst = '1' then
87              x_i <= (others => '0');
88              y_i <= (others => '0');
89              z_i <= (others => '0');
90          elsif clk'event and clk = '1' then
91              if req = '1' then
92                  x_i <= x_0;
93                  y_i <= y_0;
94                  z_i <= z_0;
95              else
96                  x_i <= x_ip1;
97                  y_i <= y_ip1;
98                  z_i <= z_ip1;
99              end if;
100         end if;
101     end process;
102
103     x_nm1 <= x_ip1 when ack_aux = '1' else (others => '0');
104     y_nm1 <= y_ip1 when ack_aux = '1' else (others => '0');
105     z_nm1 <= z_ip1 when ack_aux = '1' else (others => '0');
106
107 end behavioral;
```

3.3. Arquitectura desenrrollada

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use ieee.math_real.all;
5
```

```
6  entity cordic_unrolled is
7      generic(
8          N : natural := 16
9      );
10     port(
11         clk: in std_logic;
12         rst: in std_logic;
13         req: in std_logic;
14         ack: out std_logic;
15         rot0_vec1: in std_logic;
16         x_0 : in std_logic_vector(N-1 downto 0);
17         y_0 : in std_logic_vector(N-1 downto 0);
18         z_0 : in std_logic_vector(N-1 downto 0);
19         x_nm1 : out std_logic_vector(N-1 downto 0);
20         y_nm1 : out std_logic_vector(N-1 downto 0);
21         z_nm1 : out std_logic_vector(N-1 downto 0)
22     );
23 end cordic_unrolled;
24
25 architecture behavioral of cordic_unrolled is
26
27     type matrix_type is array (natural range <>) of std_logic_vector(N-1 downto 0);
28
29     constant CLOG2N : natural := natural(ceil(log2(real(N))));
30
31     signal count : unsigned(CLOG2N-1 downto 0) := (others => '0');
32     signal x_vect, y_vect, z_vect : matrix_type(N downto 0);
33     signal x_reg, y_reg, z_reg : matrix_type(N-1 downto 0);
34     signal ack_aux : std_logic;
35     signal atan_2mi : std_logic_vector(N-3 downto 0);
36
37
38 begin
39
40     -- Mi idea era utilizar esto, pero vivado no me dejo sintetizar porque los valores reales no consta
41     --atan_2mi <= std_logic_vector(to_unsigned(integer(round( (arctan(real(2)**real(-1*to_integer(count
42     -- Asi que volvi a utilizar la ROM
43
44     ROM: entity work.atan_rom(behavioral)
45         generic map(
46             ADD_W  => CLOG2N,
47             DATA_W => N-2
48         )
49         port map(
50             addr_i => std_logic_vector(count),
51             data_o => atan_2mi
52         );
53
54     CORDIC_UNR: for i in 0 to N-1 generate
```

```
55
56          CORDIC_BASE: entity work.cordic_base(behavioral)
57          generic map(
58              N => N,
59              CLOG2N => CLOG2N
60          )
61          port map(
62              num_iter => count,
63              rot0_vec1 => rot0_vec1,
64              x_i => x_vect(i),
65              y_i => y_vect(i),
66              z_i => z_vect(i),
67              atan_2mi => atan_2mi,
68              x_ip1 => x_reg(i),
69              y_ip1 => y_reg(i),
70              z_ip1 => z_reg(i)
71          );
72
73      end generate CORDIC_UNR;
74
75      REGISTER_X: for i in 0 to N-1 generate
76          REG_X: entity work.reg(behavioral)
77          generic map(
78              N => N
79          )
80          port map(
81              d   => x_reg(i),
82              rst => rst,
83              clk => clk,
84              q   => x_vect(i+1)
85          );
86      end generate REGISTER_X;
87
88      REGISTER_Y: for i in 0 to N-1 generate
89          REG_Y: entity work.reg(behavioral)
90          generic map(
91              N => N
92          )
93          port map(
94              d   => y_reg(i),
95              rst => rst,
96              clk => clk,
97              q   => y_vect(i+1)
98          );
99      end generate REGISTER_Y;
100
101     REGISTER_Z: for i in 0 to N-1 generate
102         REG_Z: entity work.reg(behavioral)
103         generic map(
```

```
104             N => N
105         )
106         port map(
107             d   => z_reg(i),
108             rst => rst,
109             clk => clk,
110             q   => z_vect(i+1)
111         );
112     end generate REGISTER_Z;
113 
114     x_vect(0) <= x_0;
115     y_vect(0) <= y_0;
116     z_vect(0) <= z_0;
117     x_nm1 <= x_reg(N-1);
118     y_nm1 <= y_reg(N-1);
119     z_nm1 <= z_reg(N-1);
120 
121     process(clk,rst)
122     begin
123         if rst = '1' then
124             count <= (others => '0');
125         elsif clk'event and clk = '1' then
126             if req = '1' then
127                 count <= (others => '0');
128             elsif count /= N-1 then
129                 count <= count + 1;
130             end if;
131         end if;
132     end process;
133 
134     ack_aux <= '1' when  count = N-1 else
135                '0';
136 
137     ack <= ack_aux;
138 
139 
140 end behavioral;
```

3.4. Cordic Base

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  
5  entity cordic_base is
6      generic(
7          N : natural := 16; -- cantidad de iteraciones que va a hacer el algoritmo
```

```
8           CLOG2N : natural := 4 -- tamaño del punto fijo
9       );
10      port(
11          num_iter : in unsigned(CLOG2N-1 downto 0);
12          rot0_vec1: in std_logic;
13          x_i : in std_logic_vector(N-1 downto 0);
14          y_i : in std_logic_vector(N-1 downto 0);
15          z_i : in std_logic_vector(N-1 downto 0);
16          atan_2mi : in std_logic_vector(N-3 downto 0);
17          x_ip1 : out std_logic_vector(N-1 downto 0);
18          y_ip1 : out std_logic_vector(N-1 downto 0);
19          z_ip1 : out std_logic_vector(N-1 downto 0)
20      );
21  end cordic_base;
22
23  architecture behavioral of cordic_base is
24
25      signal x_aux,y_aux,z_aux : std_logic_vector(N-1 downto 0);
26      signal x_shifted,y_shifted : std_logic_vector(N-1 downto 0);
27      signal x_o,y_o,z_o : std_logic_vector(N-1 downto 0);
28      signal di: std_logic;
29      signal not_di: std_logic;
30      signal sx_atan_2mi: std_logic_vector(N-1 downto 0);
31
32  begin
33
34      x_aux <= x_i;
35      y_aux <= y_i;
36      z_aux <= z_i;
37
38      x_shifted <= std_logic_vector(shift_right(signed(x_aux), to_integer(num_iter)));
39      y_shifted <= std_logic_vector(shift_right(signed(y_aux), to_integer(num_iter)));
40
41      di <= z_i(N-1) when rot0_vec1 = '0' else
42            not(y_i(N-1));
43
44      not_di <= not(di);
45
46      sumador_x: entity work.add_sub(behavioral)
47      generic map(N => N)
48      port map(
49          x => x_aux,
50          y => y_shifted,
51          z => x_o,
52          add0_sub1 => not_di
53      );
54
55      sumador_y: entity work.add_sub(behavioral)
56      generic map(N => N)
```

```
57      port map(
58          x => y_aux,
59          y => x_shifted,
60          z => y_o,
61          add0_sub1 => di
62      );
63
64      sx_atan_2mi <= "00" & atan_2mi;
65
66      sumador_z: entity work.add_sub(behavioral)
67      generic map(N => N)
68      port map(
69          x => z_aux,
70          y => sx_atan_2mi,
71          z => z_o,
72          add0_sub1 => not_di
73      );
74
75      x_ip1 <= x_o;
76      y_ip1 <= y_o;
77      z_ip1 <= z_o;
78
79  end behavioral;
```

3.4.1. Componentes extras utilizados

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   use IEEE.numeric_std.all;
4   use IEEE.math_real.all;
5
6   entity atan_rom is
7           generic(
8                   ADD_W  : natural:= 10;
9                   DATA_W : natural:= 9
10      );
11          port(
12                  addr_i : in std_logic_vector(ADD_W-1 downto 0);
13                  data_o : out std_logic_vector(DATA_W-1 downto 0)
14      );
15  end entity atan_rom;
16
17
18  architecture behavioral of atan_rom is
19
20          type rom_type is array (natural range <>) of std_logic_vector(DATA_W-1 downto 0);
21
22          function atan (constant i : natural) return std_logic_vector is
```

```
23
24      begin
25          return std_logic_vector(to_unsigned(integer(round( (arctan(real(2)**real(-1*i)) / arctan(real(1
26          end;
27
28          signal rom : rom_type(0 to 2**ADD_W-1);
29
30
31  begin
32          Load_ROM : for i in 0 to DATA_W-1 generate
33                  rom(i) <= atan(i);
34          end generate Load_ROM;
35
36      data_o <= (others => '0') when unsigned(addr_i) > to_unsigned(DATA_W-1,ADD_W)  else
37                rom(to_integer(unsigned(addr_i)));
38
39  end behavioral;
```

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   use IEEE.numeric_std.all;
4
5   entity add_sub is
6       generic(N : natural := 5);
7       port(
8           x         : in std_logic_vector(N-1 downto 0);
9           y         : in std_logic_vector(N-1 downto 0);
10          add0_sub1 : in std_logic;
11          z : out std_logic_vector(N-1 downto 0)
12      );
13  end add_sub;
14
15  architecture behavioral of add_sub is
16
17  begin
18      z <= std_logic_vector(signed(x) + signed(y)) when add0_sub1 = '0' else
19           std_logic_vector(signed(x) - signed(y));
20
21  end behavioral;
```

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity reg is
5       generic(
6           N : natural := 16
7       );
```

```
    port(
        d   : in std_logic_vector(N-1 downto 0);
        rst : in std_logic;
        clk : in std_logic;
        q   : out std_logic_vector(N-1 downto 0)
    );

end reg;

architecture behavioral of reg is

begin

    process(clk,rst)
    begin
        if rst = '1' then
            q <= (others => '0');

        elsif clk = '1' and clk'event then
            q <= d;

        end if;
    end process;

end behavioral;
```

4. Simulación

Para simular estas arquitecturas se utilizo el software *Excel* para comparar los resultados arrojados en la simulación, con las ecuaciones (2) y (4). Tanto para la arquitectura iterativa como desenrrollada se decidió utilizar los siguientes parámetros, teniendo en cuenta que las coordenadas en realidad se consideran a fondo de escala, es decir, en este caso la coordenada 0,5 seria 4096.

$$\begin{cases} \text{Coordenadas 1:} \quad (x_0, y_0) = \{0{,}5\,;\ 0\} \quad \text{Ángulo a rotar:} \quad \beta = 45^\circ \\ \\ \text{Coordenadas 2:} \quad (x_0, y_0) = \{0{,}5\,;\ 0{,}5\,\} \quad \text{Ángulos a rotar:} \quad \beta_1 = 150^\circ \ \ \beta_2 = 200^\circ \ \ \beta_3 = -128^\circ \end{cases}$$

Además, para ambas arquitecturas se utilizó el siguiente banco de pruebas

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

```

```
7  entity cordic_testbench is
8  end entity cordic_testbench;
9
10 architecture cordic_testbench_arq of cordic_testbench is
11
12     constant N : natural := 16;
13     constant FILE_PATH  : string := "..\Datos.txt";
14
15     signal clk : std_logic := '0';
16     signal rst : std_logic := '1';
17     signal req : std_logic := '0';
18     signal ack : std_logic;
19     signal rot0_vec1 : std_logic := '0';
20     signal x_in,y_in,z_in : std_logic_vector(N-1 downto 0);
21     signal x_out,y_out,z_out : std_logic_vector(N-1 downto 0);
22
23     file datos: text open read_mode is FILE_PATH;
24
25
26 begin
27
28     clk <= not clk after 10 us;
29     rst <= '0' after 2 us;
30
31     Test_Sequence: process
32
33         variable l   : line;
34         variable ch  : character := ' ';
35         variable aux : integer;
36         variable z_file: integer;
37         variable ANG_RAD: real;
38         variable ANG_Z : integer;
39
40     begin
41
42         while not(endfile(datos)) loop
43             wait until rising_edge(clk);
44             -- Se lee una linea del archivo de valores de prueba
45             readline(datos, l);
46             -- Se extrae un entero de la linea
47             read(l, aux);
48             -- Se carga el valor de la coordenada X (en fondo de escala)
49             x_in <= std_logic_vector(to_unsigned(aux, N));
50             -- Se lee un caracter (el espacio)
51             read(l, ch);
52             -- Se lee otro entero de la linea
53             read(l, aux);
54             -- Se carga el valor de la coordenada Y (en fondo de escala)
55             y_in <= std_logic_vector(to_signed(aux, N));
```

```
56              -- Se lee otro caracter (el espacio)
57              read(l, ch);
58              -- Se lee otro entero
59              read(l, aux);
60              -- Se carga el valor del angulo a rotar (en grados)
61              z_file := aux;
62
63              -- Opero con el angulo a rotar.
64              ANG_RAD := (real(z_file)*MATH_PI)/real(180); -- Lo paso a radianes
65              ANG_Z := integer( round( (ANG_RAD/arctan(real(1))) * real(2**(N-3)) ) ); -- Lo escalo
66
67              -- Se carga el valor correspondiente del angulo a rotar
68              z_in <= std_logic_vector(to_signed(ANG_Z,N));
69
70              req <= '1';
71              wait until rising_edge(clk);
72              req <= '0';
73              wait until ack = '1';
74              wait until rising_edge(clk);
75          end loop;
76
77          file_close(datos); -- Se cierra el archivo
78
79          -- Se aborta la simulacion (fin del archivo)
80          assert false report
81              "Fin de la simulacion" severity failure;
82
83      end process Test_Sequence;
84
85      DUT: entity work.cordic(behavioral_unrolled)
86      generic map(
87          N => N
88      )
89      port map(
90          clk => clk,
91          rst => rst,
92          req => req,
93          rot0_vec1 => rot0_vec1,
94          x_i => x_in,
95          y_i => y_in,
96          z_i => z_in,
97          ack => ack,
98          x_o => x_out,
99          y_o => y_out,
100         z_o => z_out
101
102     );
103
104 end architecture cordic_testbench_arq;
```

4.1. Arquitectura iterativa

Los resultados de excel para el primer caso donde los parámetros son $(x_0, y_0) = \{0{,}5\,;\ 0\}$ y Ángulo a rotar $\beta = 45°$, pueden verse en la siguiente figura

CORDIC - ALGORITMO PARA ROTACION DE VECTORES

Xin	0,5	4096
Yin	0	0
Zin [º]	45	16384
Zin [rad]	0,785398163	

Modo ROT	
Xout	4769,352649
Yout	4769,352649
Zout [º]	0

Modo Vec	
Xout	6744,8832
Yout	0
Zout [º]	8192

G Cordic	1,6467	
Unidad	1	8192

Figura 4: Resultados Excel

E resultado (para modo rotación) de la simulación, además de la simulación completa en el software *GTKwave* pueden verse en las figuras 5 y 6 respectivamente. El resto de los resultados se pueden comprobar corriendo la simulación con el banco de pruebas indicado anteriormente.

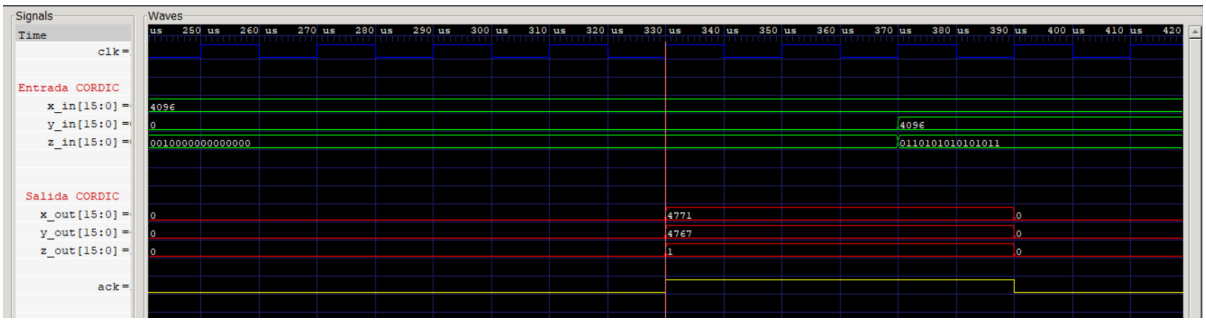

Figura 5: Resultados simulación

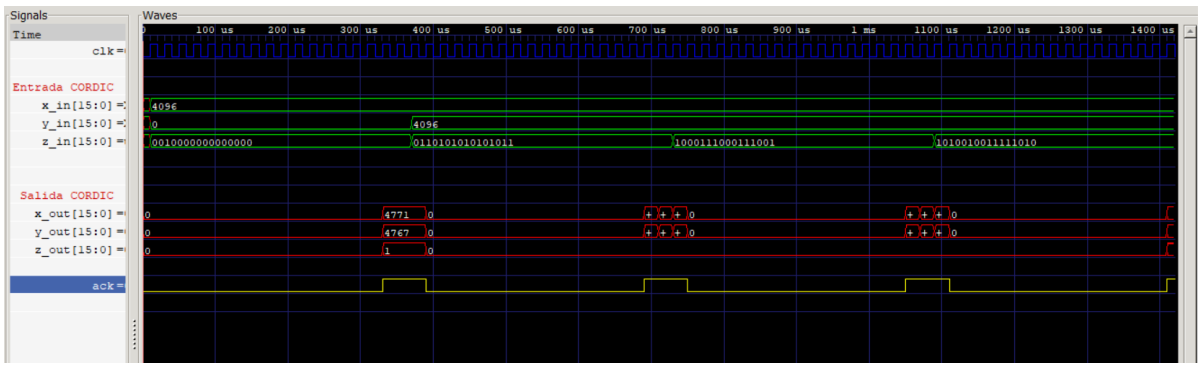

Figura 6: Simulación completa

4.2. Arquitectura desenrrollada

Los resultados de excel para el caso donde los parámetros son $(x_0, y_0) = \{0{,}5\,;\ 0,5\}$ y Ángulo a rotar: $\beta = -128°$, pueden verse en la siguiente figura

CORDIC - ALGORITMO PARA ROTACION DE VECTORES

Xin	0,5	4096
Yin	0,5	4096
Zin [°]	-128	-46603
Zin [rad]	-2,234021443	

Modo ROT		
	Xout	1162,475752
	Yout	-9467,605235
	Zout [°]	0

Modo Vec		
	Xout	9538,705298
	Yout	0
	Zout [°]	-15109,68889

G Cordic	1,6467	
Unidad	1	8192

Figura 7: Resultados Excel

El resultado (para modo rotación) de la simulación, además de la simulación completa en el software *GTKwave* puede verse en las figuras 8 y 9 respectivamente. El resto de los resultados se pueden comprobar corriendo la simulación con el banco de pruebas indicado anteriormente.

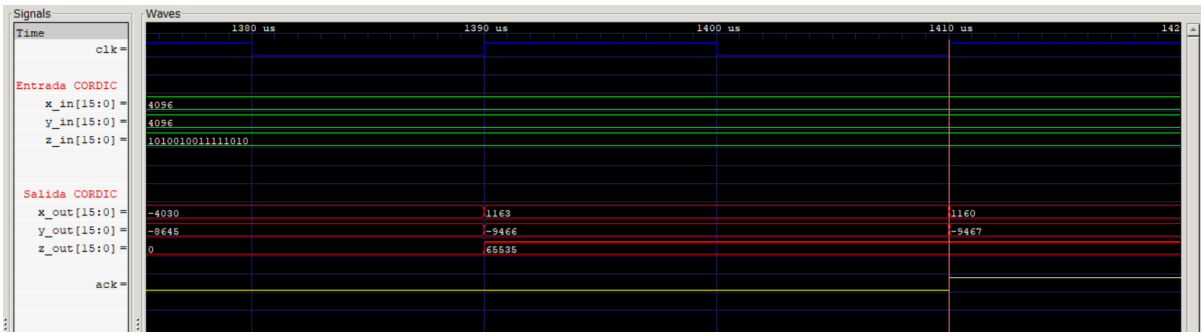

Figura 8: Resultados simulación

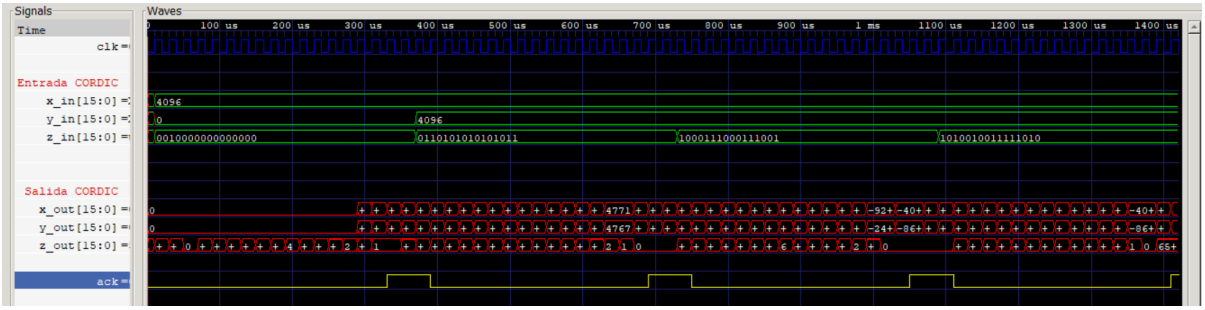

Figura 9: Simulación completa

5. Síntesis

Para esta sección, se realizo una síntesis sobre el dispositivo FPGA `xc7a15tftg256-1` mediante el *software* Vivado. En esta herramienta, se muestra como resulta el circuito descripto mediante VHDL implementado en el dispositivo.

5.1. Arquitectura iterativa

El esquemático RTL se muestra en las figura 10 y 11, donde esta ultima es un acercamiento para mostrar con mas exactitud la entidad `CORDIC_I`. Para el caso del circuito de implementación, este resulto muy grande como para mostrar en este informe. Por lo tanto, se adjuntará como un anexo con el presente trabajo.

Otro de los resultados de interés, es el resumen de recursos utilizados en el diseño. El cual muestra la cantidad utilizada de Flip-Flops, LUT y puertos IO que se serian necesarios en la implementación en el dispositivo. Este resumen y el resumen de tiempos se muestra en las figuras 12 y 13 respectivamente.

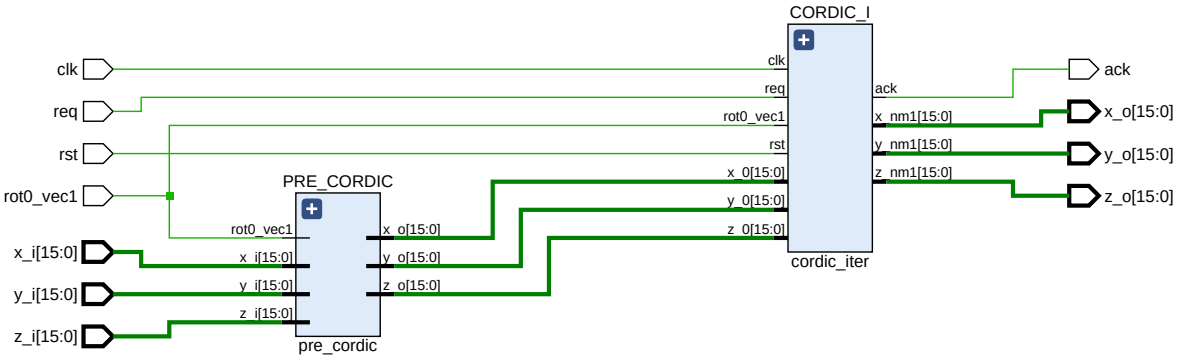

Figura 10

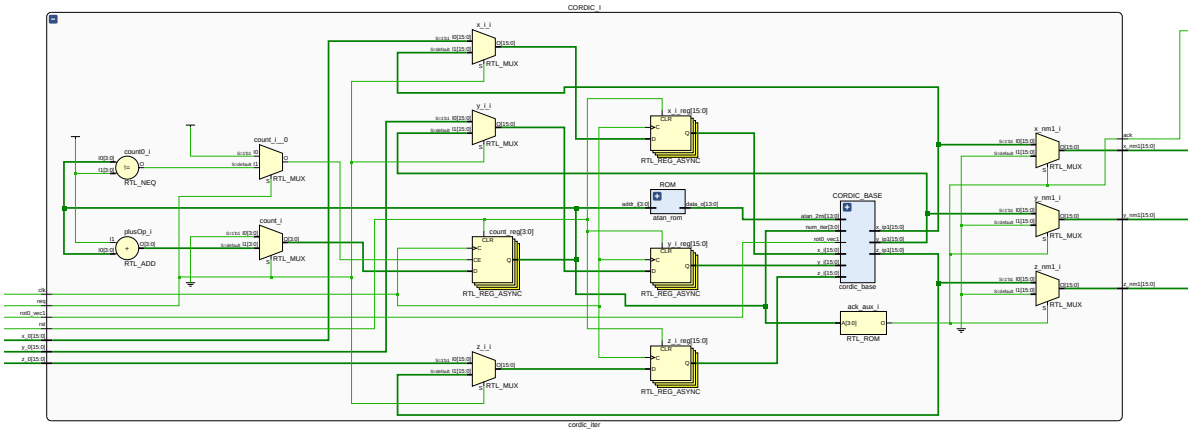

Figura 11

Resource	Utilization	Available	Utilization %
LUT	251	10400	2.41
FF	52	20800	0.25
IO	101	170	59.41

Figura 12

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	inf	Worst Hold Slack (WHS):	inf	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	NA
Total Number of Endpoints:	157	Total Number of Endpoints:	157	Total Number of Endpoints:	NA

Figura 13

5.2. Arquitectura desenrrollada

El esquemático RTL se muestra en las figura 14 y 15, donde esta ultima es un acercamiento para mostrar con mas exactitud la entidad `CORDIC_UP`. Si bien esta entidad queda muy grande para mostrar en una sola imagen, se hizo zoom sobre la primera parte para poder ver el conexionado y el *pipelining*.

Para el caso del circuito de implementación, este resulto muy grande como para mostrar en este informe. Por lo tanto, se adjuntará como un anexo con el presente trabajo.

Y finalmente, se muestra el resumen de los recursos utilizados y el resumen de tiempos en las figuras 16 y 17 respectivamente.

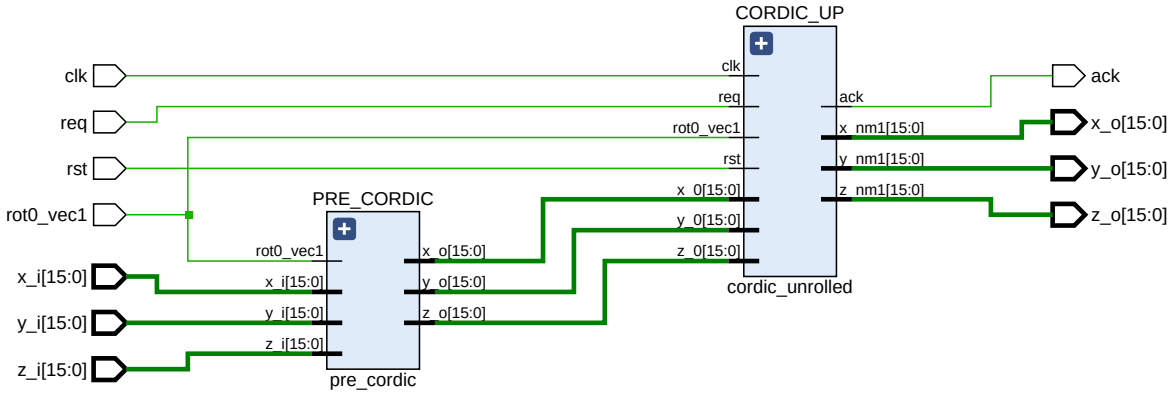

Figura 14

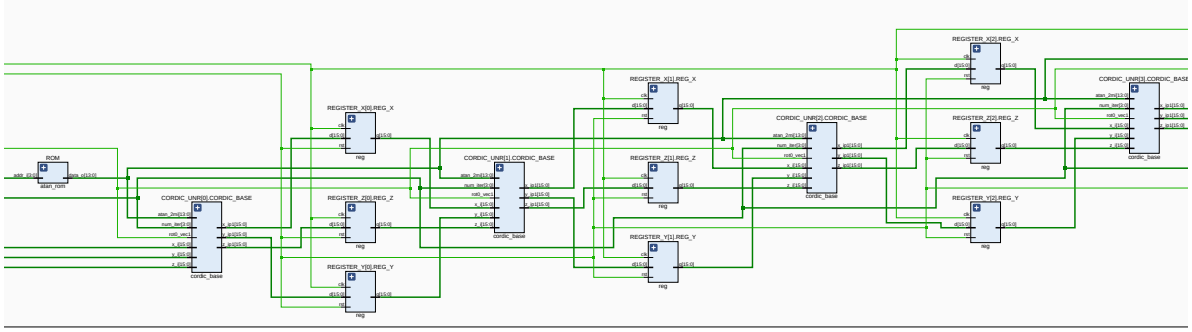

Figura 15

Resource	Utilization	Available	Utilization %
LUT	1660	10400	15.96
FF	730	20800	3.51
IO	101	170	59.41

Figura 16

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	inf	Worst Hold Slack (WHS):	inf	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	NA
Total Number of Endpoints:	1519	Total Number of Endpoints:	1519	Total Number of Endpoints:	NA

Figura 17

6. Conclusiones

En conclusión, se puede aclarar que se comprendió el funcionamiento del algoritmo de cordic y se pudo ver la ventaja que este presenta en dispositivos en los que no hay disponibilidad de multiplicadores, ya que solo utiliza las operaciones de suma, resta y desplazamiento.

Por otro lado, también se pudo notar que los resultados arrojados por la simulación presentan un error bajo, el cual puede mejorarse incluso más, si se toman mas iteraciones y se utilizan unidades de punto flotante para las sumas y restas.

Por ultimo, es importante destacar las diferencias encontradas en las dos arquitecturas implementadas.

La arquitectura iterativa ocupa mucho menos espacio físico (menos recursos), ya que la salida del cordic base se realimenta a la entrada, pero resulto mas lenta que la desenrrollada con pipeline.

Como aspecto negativo, la arquitectura desenrrollada utiliza mas espacio, esto pudiéndose ver, en la utilización de recursos provista por el software *Vivado* en la sección de síntesis.