Universidad de Buenos Aires
Facultad de Ingeniería

Sistemas Digitales (86.41)

Trabajo Práctico Final (Versión C):

Diseño de un motor de rotación gráfico en 2D Basado en el algoritmo CORDIC

Integrantes:

Puy Gonzalo	99784	gpuy@fi.uba.ar

2 do cuatrimestre 2023

Índice

1. Objetivo **2**

2. Desarrollo **2**
2.1. UART . . . 4
2.2. RX Control . . . 4
2.3. Cordic Control . . . 8
2.4. CORDIC . . . 11
2.4.1. Pre-Cordic . . . 14
2.4.2. Cordic Base . . . 15
2.4.3. Post-Cordic . . . 15

3. Resultados **17**
3.1. Simulación . . . 17
3.1.1. Rotación de la posición inicial $-45°$ . . . 17
3.1.2. Rotación continua en sentido horario . . . 18
3.1.3. Código VHDL del *test bench* utilizado en la sección 3.1.1 . . . 19
3.2. Prueba en FPGA del servidor remoto . . . 27

4. Conclusiones **31**

1. Objetivo

El presente trabajo se desarrollará una arquitectura de rotación de un vector en 2D, basada en el algoritmo CORDIC. El objetivo principal es desarrollar tanto la unidad aritmética de cálculo como así también el controlador de video asociado a la visualización del movimiento.

Para la realización completa del trabajo práctico se utilizará una interfaz serie UART por medio de la cual se comandará el giro del vector. A partir de los valores de las componentes, se rotará el vector en el plano xy según el valor que adquieran las entradas del sistema, y por último las componentes rotadas serán presentadas en un monitor VGA. En la Figura 1 puede observarse un diagrama en bloques del sistema completo. Se determinará la cantidad mínima de bits de ancho de palabra (bits de precisión) para alcanzar las especificaciones requeridas.

2. Desarrollo

En la Figura 1 se muestra un esquema general del sistema. El diseño será implementado en lenguaje VHDL para luego ser simulado, sintetizado y finalmente, probado en una FPGA que es encuentra en un servidor remoto provisto por la cátedra. Se mostrará tanto la simulación como las pruebas en la sección 3 con un correspondiente análisis.

Para lograr la rotación de las coordenadas 2D se deberán enviar comandos a través de la UART, respetando el siguiente formato

`ROT A ang`: indica la rotación del vector en un ángulo `ang`
`ROT C N`: indica la rotación continua del vector en sentido del parámetro `N`, donde `N` puede ser `H` para sentido horario o `D` para sentido antihorario

En el modo continuo, se eligió que el ángulo de rotación sea de 10°. Además, se eligió como posición inicial las coordenadas $(0; 0, 5)$. Y se cuenta con las siguientes especificaciones dadas por el enunciado del trabajo:

- Resolución de video: 640 x 480 1 bit monocromo 50 Hz.
- Velocidad angular de rotación mínima: 35,156 25 $\frac{^\circ}{s}$.
- Paso angular $\Delta\phi$: 0,703 125°.
- Dispositivo: Spartan 3E-500 (Kit Nexys 2 Board o Starter Kit Board).

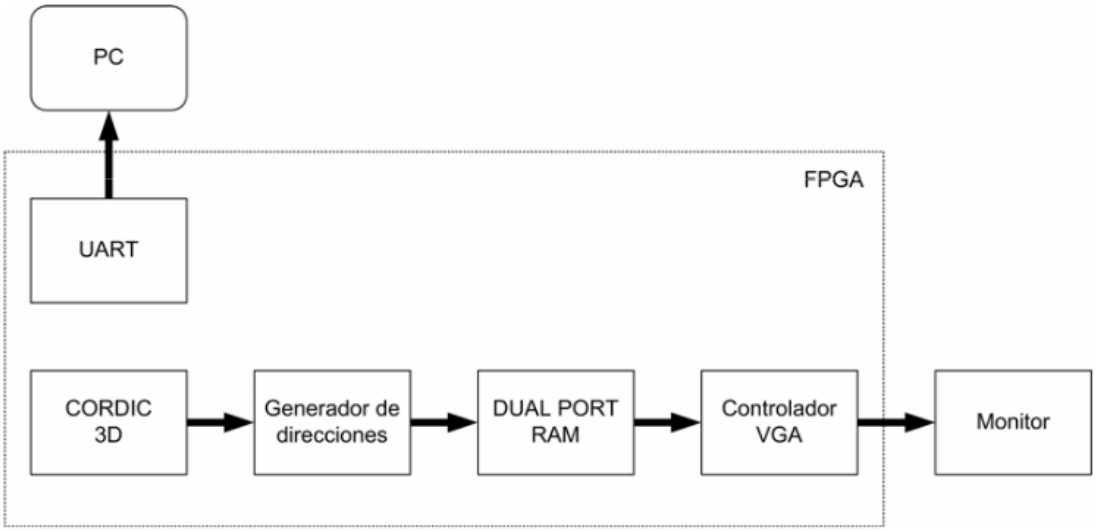

Figura 1: Esquema general del sistema

Se deben tener en cuenta un par de aclaraciones importantes. Primero, el cordic en este caso será 2D (dos dimensiones, es decir, plano xy) y no 3D como se muestra en la Figura 1. Segundo, por falta de acceso a una placa de desarrollo FPGA, solo fue posible realizar la arquitectura de rotación y el manejo del sistema vía UART. Y por último, dada la segunda aclaración, no se pudo usar el dispositivo dado por las especificaciones. Para las pruebas en una FPGA real se utilizó la placa `Arty Z7-10` que está alojada en un servidor remoto que es provisto por la cátedra. Entonces, la síntesis de la arquitectura desarrollada para este trabajo se realizó para dicha placa.

En la Figura 2 se puede ver un esquema de la arquitectura diseñada teniendo en cuenta las aclaraciones mencionadas. En las secciones posteriores se mostrarán más detalles sobre esta.

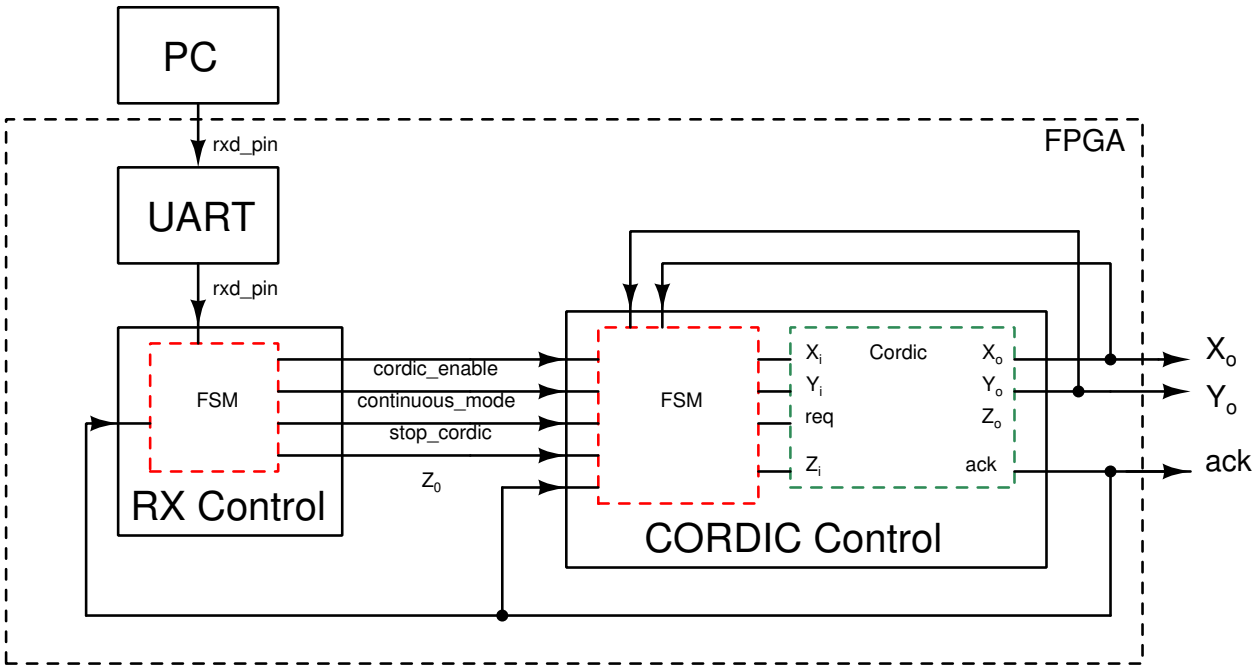

Figura 2: Esquema del sistema diseñado

2.1. UART

El bloque UART fue provisto por la cátedra, sin más que agregar que se eligió un baudrate igual a 115200. El código VHDL con la implementación de este bloque se adjunta como anexo con este informe.

2.2. RX Control

Este bloque es el que se encarga de recibir los comandos vía UART para poder indicarle al bloque `CORDIC CONTROL` que accione al bloque `CORDIC` (donde se realizara la rotación) según los comandos recibidos. Va a proporcionarle al bloque a `CORDIC CONTROL` el modo elegido y el ángulo a rotar.

Está diseñado con base en una máquina de estados que simplemente va avanzando según los caracteres recibidos por UART. En caso de recibir algo que tenga sentido para el sistema, este continúa avanzando y en caso de recibir algo que no corresponde simplemente se vuelve al estado `IDLE` en espera de la próxima recepción.

Al recibir un comando que tenga sentido, este bloque proporciona el modo elegido (continuo [C] o rotación simple [A]) con la señal de control ***continuos_mode***, una señal de control para activar la rotación ***enable_cordic*** y se quedará iterando en alguno de los estados `PROCESS_CORDIC_C` o `PROCESS_CORDIC_A` (según corresponda) hasta que (en el caso de rotación simple) se reciba la señal ***ack*** que indica que se procesó la rotación o (en el caso de rotación continua) se reciba vía UART el caracter "`S`" en cuyo caso se activará la señal del control ***stop_cordic*** que parará toda rotación que se esté haciendo, volviendo finalmente al estado `IDLE`.

Como se puede ver en el código de este bloque, se tienen la máquina de estados (FSM) que fue realizada con un `process()` el cual tiene en su lista de sensibilidad a las señales `clk` y `rst`. Una vez se dé un flanco positivo de reloj, se verifica si ocurrió un flanco ascendente de la señal `rx_data_rdy` o de `ack_cordic` (por eso se hace el seguimiento de estas con la señal original y con la señal de delay).

Otra aclaración importante es que no se pueden recibir números de más de 3 dígitos. Esto es, luego del primer dígito el sistema espera otro dígito o un `ENTER` (carriage return), y así hasta llegar al tercero donde **solo se espera un** `ENTER` (carriage return). De lo contrario, se pasa al estado `IDLE` y se queda a la espera del siguiente envío. Además, los ángulos enviados deben ser números enteros. También se aceptan números negativos.

```
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 rx_control is
7      generic(
8          N : natural := 16;
9          BAUD_RATE : integer := 115200;
10         CLOCK_RATE: integer := 50E6
11     );
12     Port (
13         clk :   in  std_logic;
14         rst :   in  std_logic;
15         rxd_pin :   in std_logic;
16         cordic_ack : in std_logic;
17         cordic_enable : out std_logic;
18         continuous_mode : out std_logic;
```

```
19          stop_cordic : out std_logic;
20          z_o    : out std_logic_vector(N-1 downto 0)
21      );
22  end rx_control;
23
24
25  architecture behavioral of rx_control is
26
27      type fsm_state is (IDLE, R, 0, T, A, C, SIGN, NUM_1, NUM_2, NUM_3, COUNTER_CLOCKWISE, CLOCKWISE, SPACE_1,
        ↪  SPACE_2, PROCESS_A_CORDIC, PROCESS_C_CORDIC);
28
29      constant ZERO       : signed(N-9 downto 0) := (others => '0'); --Recordar que N >= 9
30      constant C_ANG      : signed(N-1 downto 0) := to_signed(10,N);
31
32      signal rx_data_rdy : std_logic;
33      signal rx_data_rdy_delay : std_logic := '0';
34      signal rx_data_rdy_re : std_logic := '0';
35      signal cordic_ack_delay : std_logic := '0';
36      signal cordic_ack_re : std_logic := '0';
37      signal rx_data      : std_logic_vector(7 downto 0);
38      signal state        : fsm_state;
39      signal digit_1, digit_2, digit_3       : signed(N-1 downto 0) := (others => '0');
40      signal ang          : signed(N-1 downto 0) := (others => '0');
41      signal ang_rad      : signed(N-1 downto 0) := (others => '0');
42      signal continuous_mode_aux    : std_logic := '0';
43      signal cordic_enable_aux      : std_logic := '0';
44      signal sign_en      : std_logic := '0';
45      signal stop_cordic_aux : std_logic := '0';
46
47      function AngConverter (constant i : signed(N-1 downto 0) ) return signed is
48      begin
49          return resize( i * to_signed( integer(round( MATH_PI/real(180) * real(2**(N-3)) *
            ↪  (real(1)/arctan(real(1))) )),N) ,N);
50      end;
51
52      -- Para poder ver los estados de interes:
53      signal idle_p, process_a_cordic_p, process_c_cordic_p : std_logic := '0';
54
55  begin
56
57  -- Para poder ver los estados de interes:
58  idle_p <= '1' when state = IDLE else '0';
59  process_a_cordic_p <= '1' when state = PROCESS_A_CORDIC else '0';
60  process_c_cordic_p <= '1' when state = PROCESS_C_CORDIC else '0';
61  --
62
63
64  stop_cordic <= stop_cordic_aux;
65  continuous_mode <= continuous_mode_aux;
66  cordic_enable <= cordic_enable_aux;
67
68
69  UART: entity work.uart_top(uart_top_arq)
70  generic map (
71      BAUD_RATE => BAUD_RATE,
72      CLOCK_RATE => CLOCK_RATE
73  )
74  port map(
75      clk_pin => clk,
76      rst_pin => rst,
77      rxd_pin => rxd_pin,
78      rx_data_rdy => rx_data_rdy,
79      rx_data => rx_data
80  );
81
82
83  FSM: process(rst, clk)
84  begin
85      if rst = '1' then
86          stop_cordic_aux <= '0';
87          ang <= (others => '0');
```

```
88          digit_1 <= (others => '0');
89          digit_2 <= (others => '0');
90          digit_3 <= (others => '0');
91          sign_en <= '0';
92          rx_data_rdy_delay <= '0';
93          cordic_ack_delay <= '0';
94          state <= IDLE;
95
96      elsif clk = '1' and clk'event then
97          if ( rx_data_rdy_re = '1' ) or ( cordic_ack_re = '1' ) then
98              case state is
99
100                 when IDLE =>
101                     ang <= (others => '0');
102                     sign_en <= '0';
103                     stop_cordic_aux <= '0';
104                     if rx_data = "01010010" then --Si el sigiuente dato por UART es la R
105                         state <= R;
106                     else
107                         state <= IDLE;
108                     end if;
109
110                 when R =>
111                     if rx_data = "01001111" then --Si el sigiuente dato por UART es la O
112                         state <= O;
113                     else
114                         state <= IDLE;
115                     end if;
116
117                 when O =>
118                     if rx_data = "01010100" then --Si el sigiuente dato por UART es la T
119                         state <= T;
120                     else
121                         state <= IDLE;
122                     end if;
123
124                 when T =>
125                     if rx_data = "00100000" then --Si el sigiuente dato por UART es un ESPACIO
126                         state <= SPACE_1;
127                     else
128                         state <= IDLE;
129                     end if;
130
131                 when SPACE_1 =>
132                     if rx_data = "01000011" then --Si el sigiuente dato por UART es la C
133                         state <= C;
134                     elsif rx_data = "01000001" then --Si el sigiuente dato por UART es la A
135                         state <= A;
136                     else
137                         state <= IDLE;
138                     end if;
139
140                 when C =>
141                     if rx_data = "00100000" then --Si el sigiuente dato por UART es un ESPACIO
142                         state <= SPACE_2;
143                     else
144                         state <= IDLE;
145                     end if;
146
147                 when A =>
148                     if rx_data = "00100000" then --Si el sigiuente dato por UART es un ESPACIO
149                         state <= SPACE_2;
150                     else
151                         state <= IDLE;
152                     end if;
153
154                 when SPACE_2 =>
155                     if (rx_data >= "00110000" and rx_data <= "00111001") then --Si el sigiuente dato es mayor o =
                        ↪  que 48 y menor o = que 57 entonces es un numero y es el primer digito.
156                         digit_1 <= (ZERO & signed(rx_data)) - to_signed(48,N);
157                         state <= NUM_1;
```

```
158
159                     elsif rx_data = "00101101" then --Si el siguiente dato por UART es un '-'
160                         state <= SIGN;
161
162                     elsif rx_data = "01000100" then --Si el siguiente dato por UART es una D
163                         state <= COUNTER_CLOCKWISE;
164
165                     elsif rx_data = "01001000" then --Si el siguiente dato por UART es una H
166                         state <= CLOCKWISE;
167
168                     else
169                         state <= IDLE;
170                     end if;
171
172                 when SIGN =>
173                     sign_en <= '1';
174                     if rx_data >= "00110000" and rx_data <= "00111001" then --Si el sigiuente dato por UART es un
                    ↪  numero.
175                         digit_1 <= (ZERO & signed(rx_data)) - to_signed(48,N);
176                         state <= NUM_1;
177
178                     else
179                         state <= IDLE;
180                     end if;
181
182                 when NUM_1 =>
183                     if rx_data >= "00110000" and rx_data <= "00111001" then --Si el sigiuente dato por UART es un
                    ↪  numero.
184                         digit_2 <= (ZERO & signed(rx_data)) - to_signed(48,N);
185                         state <= NUM_2;
186
187                     elsif rx_data = "00001101" then --Si el siguiente dato por UART es un ENTER.
188                         if sign_en = '1' then
189                             ang <= not(resize((digit_1 * to_signed(1,N)),N)) + 1;
190                         else
191                             ang <= resize((digit_1 * to_signed(1,N)),N);
192                         end if;
193                         state <= PROCESS_A_CORDIC;
194
195                     else
196                         state <= IDLE;
197                     end if;
198
199                 when NUM_2 =>
200                     if rx_data >= "00110000" and rx_data <= "00111001" then
201                         digit_3 <= (ZERO & signed(rx_data)) - to_signed(48,10);
202                         state <= NUM_3;
203
204                     elsif rx_data = "00001101" then
205                         if sign_en = '1' then
206                             ang <= not(resize((digit_1 * to_signed(10,N)),N) + resize((digit_2 *
                            ↪  to_signed(1,N)),N)) + 1;
207                         else
208                             ang <= resize((digit_1 * to_signed(10,N)),N) + resize((digit_2 * to_signed(1,N)),N);
209                         end if;
210                         state <= PROCESS_A_CORDIC;
211
212                     else
213                         state <= IDLE;
214                     end if;
215
216                 when NUM_3 =>
217                     if rx_data = "00001101" then
218                         if sign_en = '1' then
219                             ang <= not(resize((digit_1 * to_signed(100,N)),N) + resize((digit_2 *
                            ↪  to_signed(10,N)),N) + resize((digit_3 * to_signed(1,N)),N)) + 1;
220                         else
221                             ang <= resize((digit_1 * to_signed(100,N)),N) + resize((digit_2 * to_signed(10,N)),N)
                            ↪  + resize((digit_3 * to_signed(1,N)),N);
222                         end if;
223                         state <= PROCESS_A_CORDIC;
```

```
224                     else
225                         state <= IDLE;
226                     end if;
227
228                 when COUNTER_CLOCKWISE =>
229                     ang <= C_ANG;
230                     if rx_data = "00001101" then
231                         state <= PROCESS_C_CORDIC;
232                     else
233                         state <= IDLE;
234                     end if;
235
236                 when CLOCKWISE =>
237                     ang <= not(C_ANG) + 1;
238                     if rx_data = "00001101" then
239                         state <= PROCESS_C_CORDIC;
240                     else
241                         state <= IDLE;
242                     end if;
243
244                 when PROCESS_A_CORDIC =>
245                     if cordic_ack = '0' then
246                         state <= PROCESS_A_CORDIC;
247                     else
248                         state <= IDLE;
249                     end if;
250
251                 when PROCESS_C_CORDIC =>
252                     if rx_data = "01010011" or rx_data = "01110011" then --Si el siguiente dato por UART es 's' o
                        ↪  'S' (Stop)
253                         stop_cordic_aux <= '1';
254                         state <= IDLE;
255                     else
256                         state <= PROCESS_C_CORDIC;
257                     end if;
258             end case;
259
260         end if;
261         rx_data_rdy_delay <= rx_data_rdy;
262         cordic_ack_delay  <= cordic_ack;
263     end if;
264 end process;
265
266 rx_data_rdy_re <= '1' when (rx_data_rdy = '1' and rx_data_rdy_delay = '0') else '0';
267 cordic_ack_re <= '1' when (cordic_ack = '1' and cordic_ack_delay = '0') else '0';
268
269 cordic_enable_aux <= '1' when (state = PROCESS_A_CORDIC) or (state = PROCESS_C_CORDIC) else
270                      '0';
271
272 continuous_mode_aux <= '1' when state = PROCESS_C_CORDIC else
273                        '0';
274
275 ang_rad <= AngConverter(ang) when ang /= 0 else
276            (others => '0');
277
278 z_o <= std_logic_vector(to_unsigned(0,N)) when cordic_enable_aux = '0' else
279        std_logic_vector(ang_rad) when cordic_enable_aux = '1' else
280        std_logic_vector(ang) when continuous_mode_aux = '1';
281
282 end behavioral;
```

2.3. Cordic Control

Para cumplir con las especificaciones de velocidad mostradas en la sección 2 en este bloque se diseñó un generador de pulsos de baja frecuencia el cual controla la velocidad de rotación del cordic.

Como se desea una velocidad angular mínima de $35{,}156\,25\,\frac{^\circ}{s}$ y un paso de $0{,}703\,125^\circ$ se

requiere una frecuencia de 50 Hz. Dado que se utiliza una placa que tiene por defecto un clock de 125 MHz se deberá generar un pulso de `cordic_clk` cada 2,5 millones de ciclos de clock.

Luego, también se diseñó con base en una máquina de estados que cambia de estado según las señales de control que envíe el bloque `RX CONTROL` y se encarga de elegir las coordenadas x e y a la entrada del bloque cordic. Notar en el código adjunto que se tiene un estado que se dominó `INITIAL` el cual es utilizado al inicio del sistema y se utiliza para hacer la primera conversión del cordic y dejarlo en las coordenadas iniciales (se ingresan las coordenadas iniciales y se rotan 0 grados). Esta inicialización es independiente del bloque `RX CONTROL`, **por lo que es importante esperar a que el sistema esté en las coordenadas iniciales antes de enviar comandos por UART**. De lo contrario, el comportamiento será errático.

La máquina de estados (FSM) fue realizada con un `process()` el cual tiene en su lista de sensibilidad a las señales `cordic_clk` (clock de baja frecuencia) y `rst` como puede observarse en el siguiente código.

```
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_control is
7      generic(
8          N : natural := 16;
9          N_COUNT : natural := 22
10     );
11     Port (
12         clk  : in std_logic;
13         rst  : in std_logic;
14         ena  : in std_logic;
15         mode : in std_logic;
16         stop_cordic : in std_logic;
17         ang  : in std_logic_vector(N-1 downto 0);
18         ack  : out std_logic;
19         x_o, y_o   : out std_logic_vector(N-1 downto 0)
20     );
21 end cordic_control;
22
23 architecture behavioral of cordic_control is
24
25     constant NC : natural := 2499;
26     --constant NC : natural := 2499999;
27
28     type fsm_state is (INITIAL, CONVERSION, C_CONVERSION, IDLE, CORDIC_A, CORDIC_C);
29
30     signal count        : integer := 0;
31     signal state        : fsm_state;
32     signal cordic_clk   : std_logic := '0';
33     signal req          : std_logic := '0';
34     signal ack_aux      : std_logic := '0';
35     signal x_i, y_i, z_i : std_logic_vector(N-1 downto 0) := (others => '0');
36     signal x_initial, y_initial : std_logic_vector(N-1 downto 0) := (others => '0');
37     signal x_o_aux, y_o_aux, z_o_aux : std_logic_vector(N-1 downto 0) := (others => '0');
38
39     -- Para probar:
40     signal initial_p, idle_p, cordic_a_p, cordic_c_p, conversion_p, c_conversion_p   : std_logic := '0';
41
42 begin
43
44 -- ++++++++++++++++++++++++++++++++++++++++++++++
45     -- Para poder ver los estados de la FSM:
46     initial_p <= '1' when state = INITIAL else
47                  '0';
48
49     idle_p <= '1' when state = IDLE else
50               '0';
```

```
51
52      cordic_a_p <= '1' when state = CORDIC_A else
53                    '0';
54
55      cordic_c_p <= '1' when state = CORDIC_C else
56                    '0';
57
58      conversion_p <= '1' when state = CONVERSION else
59                      '0';
60
61      c_conversion_p <= '1' when state = C_CONVERSION else
62                        '0';
63  -- ++++++++++++++++++++++++++++++++++++++++++++++++
64
65      x_initial <= std_logic_vector(to_signed(0,N));
66      y_initial <= std_logic_vector(to_signed(4096,N));
67
68      ack <= ack_aux;
69      x_o <= x_o_aux;
70      y_o <= y_o_aux;
71
72      CORDIC: entity work.cordic(behavioral_iter)
73      generic map(N => N)
74      port map(
75          clk => cordic_clk,
76          rst => rst,
77          req => req,
78          rot0_vec1 => '0', --Siempre en modo rotacion
79          x_i => x_i,
80          y_i => y_i,
81          z_i => z_i,
82          ack => ack_aux,
83          x_o => x_o_aux,
84          y_o => y_o_aux,
85          z_o => z_o_aux
86      );
87
88  FSM:
89      process (cordic_clk,rst)
90      begin
91          if rst = '1' then
92              state <= INITIAL;
93          elsif cordic_clk = '1' and cordic_clk'event then
94              case state is
95
96                  when INITIAL =>
97                      x_i <= x_initial;--Coordenada X inicial.
98                      y_i <= y_initial;-- Coordenada Y inicial.
99                      z_i <= (others => '0');
100                     req <= '1';
101                     state <= CONVERSION;
102
103                 when CONVERSION =>
104                     req <= '0';
105                     if ack_aux = '1' then
106                         state <= IDLE;
107                     else
108                         state <= CONVERSION;
109                     end if;
110
111                 when C_CONVERSION =>
112                     req <= '0';
113                     if ack_aux = '1' then
114                         state <= CORDIC_C;
115                     elsif stop_cordic = '1' then
116                         state <= IDLE;
117                     else
118                         state <= C_CONVERSION;
119                     end if;
120
121                 when IDLE =>
```

```
122                     x_i <= x_o_aux;
123                     y_i <= y_o_aux;
124                     z_i <= (others => '0');
125                     req <= '1';
126                     if ena = '1' and mode = '0' then
127                         state <= CORDIC_A;
128                     elsif ena = '1' and mode = '1' then
129                         state <= CORDIC_C;
130                     else
131                         state <= IDLE;
132                     end if;
133
134                 when CORDIC_A =>
135                     x_i <= x_o_aux;
136                     y_i <= y_o_aux;
137                     z_i <= ang;
138                     state <= CONVERSION;
139
140                 when CORDIC_C =>
141                     x_i <= x_o_aux;
142                     y_i <= y_o_aux;
143                     z_i <= ang;
144                     req <= '1';
145                     state <= C_CONVERSION;
146
147
148             end case;
149         end if;
150     end process;
151
152     clk_cordic_gen: process(clk, rst)
153     begin
154         if rst = '1' then
155             count <= 0;
156         elsif (clk = '1' and clk'event) then
157             count <= count + 1;
158             if count = NC then
159                 cordic_clk <= not cordic_clk;
160                 count <= 0;
161             end if;
162         end if;
163     end process;
164
165
166 end behavioral;
```

2.4. CORDIC

Se utilizó para este bloque el motor cordic implementado en el trabajo práctico número 3 durante la cursada de esta materia. El motor diseñado cuenta con dos modos de operación (rotación y vectorización), pero para el caso de este trabajo se utilizará **solo** el modo rotación.

Este modo 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

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

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

En la Figura 3 se puede observar como está conformado el motor cordic.

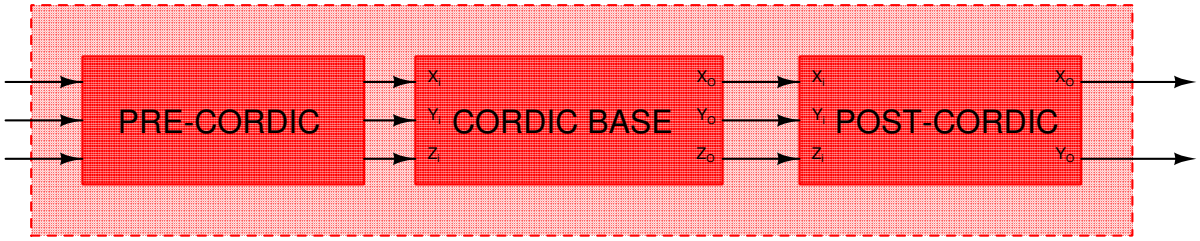

Figura 3: Motor cordic.

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,

$$\begin{cases} \text{Coordenadas:} & x = x \cdot 2^{13} \\ \text{Ángulos:} & \text{Ang} = \dfrac{\text{Ang}}{\arctan(1)} \cdot 2^{13} \end{cases} \tag{3}$$

A continuación se adjunta el código del bloque `CORDIC`

```
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      -- Entradas de post_cordic
33      signal x_i_postcordic, y_i_postcordic : std_logic_vector(N-1 downto 0);
34
35
36  begin
37
38      x_i_precordic <= x_i;
39      y_i_precordic <= y_i;
40      z_i_precordic <= z_i;
41      ack <= ack_aux;
42
43      PRE_CORDIC: entity work.pre_cordic(behavioral)
44      generic map(N => N)
45      port map(
46          x_i => x_i_precordic,
47          y_i => y_i_precordic,
48          z_i => z_i_precordic,
49          rot0_vec1 => rot0_vec1,
50          x_o => x_o_precordic,
51          y_o => y_o_precordic,
52          z_o => z_o_precordic
53      );
54
55      CORDIC_I: entity work.cordic_iter(behavioral)
56      generic map(N => N)
57      port map(
58          clk       => clk,
59          rst       => rst,
60          req       => req,
61          ack       => ack_aux,
62          rot0_vec1 => rot0_vec1,
63          x_0       => x_o_precordic,
64          y_0       => y_o_precordic,
65          z_0       => z_o_precordic,
66          x_nm1     => x_i_postcordic,
67          y_nm1     => y_i_postcordic,
68          z_nm1     => z_o
69      );
70
71      POST_CORDIC: entity work.post_cordic(behavioral)
72      generic map (N => N)
73      port map(
74          x_i_postcordic => x_i_postcordic,
75          y_i_postcordic => y_i_postcordic,
76          x_o_postcordic => x_o,
77          y_o_postcordic => y_o
78      );
79
```

```
80 end behavioral_iter;
```

2.4.1. Pre-Cordic

El bloque pre-cordic es una lógica combinacional que se encarga de negar las coordenadas xy originales para poder rotar ángulos mayores a 90°.

A continuación se adjunta el código VHDL de este bloque.

```
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 := 10 -- 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_logic_vector(abs(ANG_90)) else
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_logic_vector(abs(ANG_90)) else
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;
```

2.4.2. Cordic Base

En el trabajo práctico número 3 realizado durante el cuatrimestre, se realizaron 2 arquitecturas para el motor cordic (iterativa y desenrollada). Dentro de las conclusiones del trabajo se encontró que la arquitectura iterativa tiene la ventaja de utilizar menos recursos de la FPGA y que, en cambio, la arquitectura desenrollada es más veloz a comparación de la iterativa.

Como no se tiene especificación sobre estos aspectos, se decidió utilizar la arquitectura iterativa. En la figura 4 se muestra el diagrama de dicha arquitectura.

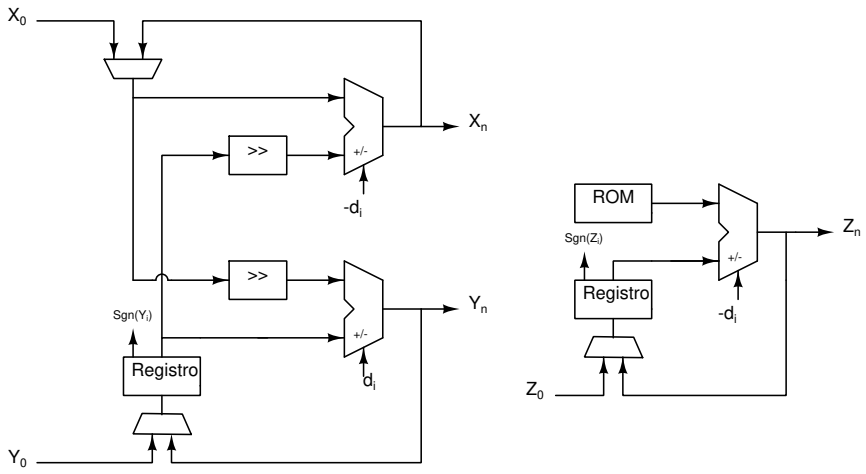

Figura 4: Arquitectura iterativa.

2.4.3. Post-Cordic

Este bloque también es una lógica combinacional que se encarga de quitar a las coordenadas finales la ganancia de cordic. Al elegir utilizar 16 iteraciones del algoritmo, sabemos que la ganancia será de 1,646 760 258 057 16 (calculado mediante hoja de cálculo provista por la cátedra que se adjunta con el trabajo), dado esto se eligió utilizar un script realizado en *python* para que calculara múltiples valores para multiplicar y luego *shiftear* (con el correspondiente error) las coordenadas originales para lograr la división.

A continuación se adjunta el código VHDL del bloque post-cordic, junto con el código de python utilizado. Finalmente, se decidió multiplicar por 2487 y shiftear 12 veces a derecha para lograr un error del 1,23 %.

```
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 post_cordic is
7      generic(
8          N : natural := 16 -- cantidad de iteraciones que va a hacer el algoritmo CORDIC.
9      );
10     port(
11         x_i_postcordic       : in std_logic_vector(N-1 downto 0);
12         y_i_postcordic       : in std_logic_vector(N-1 downto 0);
13         x_o_postcordic       : out std_logic_vector(N-1 downto 0);
14         y_o_postcordic       : out std_logic_vector(N-1 downto 0)
15     );
16 end post_cordic;
17
18 architecture behavioral of post_cordic is
19     constant MULTP : signed := to_signed(2487, N);
20     signal   x_o_rot, y_o_rot : std_logic_vector(N-1 downto 0);
21     signal   mult_i : std_logic_vector(N-1 downto 0);
22     signal   mult_o_x, mult_o_y, x_o_postcordic_shifted, y_o_postcordic_shifted :
       ↪  std_logic_vector(2*N-1 downto 0);
23
24 begin
25
26     x_o_rot <= x_i_postcordic;
27     y_o_rot <= y_i_postcordic;
28
29     mult_i <= std_logic_vector(MULTP);
30
31     MULT_X: entity work.Nbits_Mult(behavioral)
32     generic map(N => N)
33     port map(
34         x0 => x_o_rot,
35         x1 => mult_i,
36         y  => mult_o_x
37     );
38
39     MULT_Y: entity work.Nbits_Mult(behavioral)
40     generic map(N => N)
41     port map(
42         x0 => y_o_rot,
43         x1 => mult_i,
44         y  => mult_o_y
45     );
46
47     x_o_postcordic_shifted <= std_logic_vector(shift_right(signed(mult_o_x),12));
48     y_o_postcordic_shifted <= std_logic_vector(shift_right(signed(mult_o_y),12));
49
50 -- Como multiplico por ~ 0.6, no puedo tener un numero mas grande del que tenia
   ↪  originalmente de N bits, por lo que vuelvo a poner el resultado en N bits.
```

```
    x_o_postcordic <= x_o_postcordic_shifted(N-1 downto 0);
    y_o_postcordic <= y_o_postcordic_shifted(N-1 downto 0);

end behavioral;
```

```
# Imports
import pandas as pd
from fxpmath import Fxp

row_i = []
slope = 1/1.64676025805716

for e in range(1,25):
    a = Fxp(slope, 0, e, e-1)
    diff = slope - a.real
    diff_percentage = diff * 100 / slope
    mult = a.real / a.precision
    row = [e-1, slope, a.real, diff, diff_percentage, mult]
    row_i.append(row)

df = pd.DataFrame(row_i, columns = ['WL', 'Gain OSR 1', 'FP', 'Difference', 'Diff %',
↪  'MULT'])

print(df)
```

3. Resultados

3.1. Simulación

Para llevar a cabo la simulación se realizaron 2 *test bench*. Uno para la rotación continua y otro en donde se solicita una sola rotación. Como los bancos de prueba son similares (solo cambia lo que se envía por UART), solo se adjunta el código de uno de ellos para evitar repeticiones innecesarias. Se simuló con el software *GTKWave* para poder visualizar las señales de interés y sobre todo las salidas.

El clock de baja frecuencia que se debe utilizar para el motor cordic mencionado en la subsección 2.3, hacía los tiempos de simulación demasiado altos, por lo que se tuvo que dividir ese número por 1000 para poder simular los resultados. Para estas pruebas se utiliza un generador de pulsos cada 2500 ciclos de clock. Esto permitió tiempos de simulación menores.

A continuación se muestran los resultados para ambos bancos de prueba.

3.1.1. Rotación de la posición inicial $-45°$

Se simuló una rotación de $-45°$, por lo que el comando que se envió al sistema luego de esperar la conversión inicial fue:

```
ROT A -45
```

Los resultados se pueden ver en la siguiente captura.

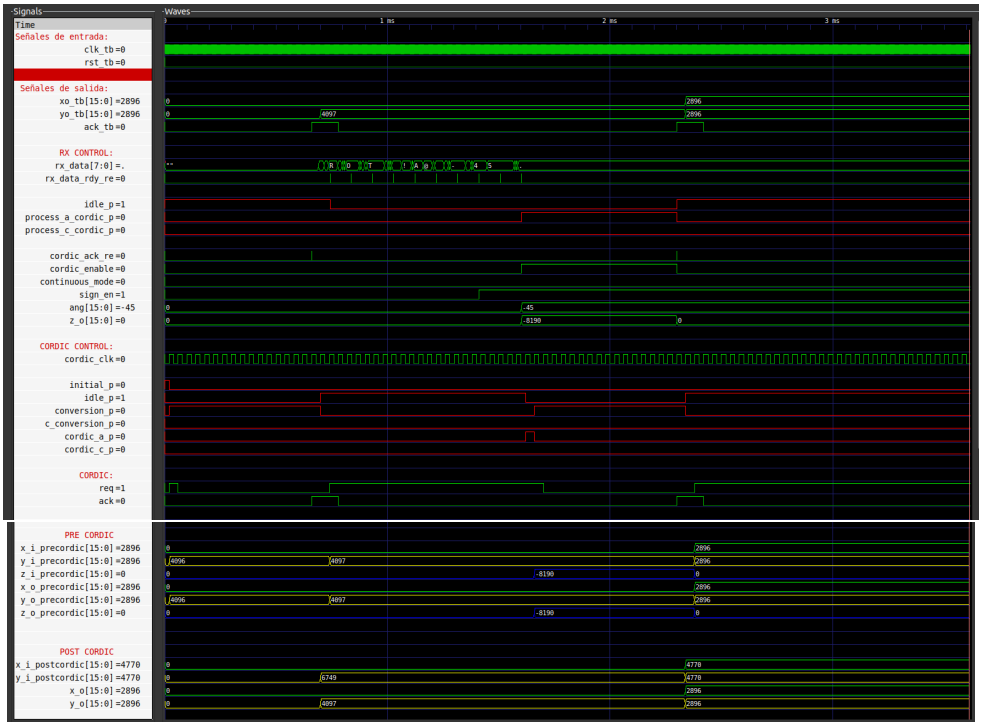

Figura 5: Simulación de rotación de las coordenadas iniciales en $-45°$

Como puede verse en la Figura 5, la simulación se produjo correctamente. Una vez iniciado el sistema, y luego de recibir el comando, la rotación fue exitosa y los distintos cambios de estado (se omitieron algunos estados no tan importantes del bloque `RX Control`) fueron los esperados. Si bien se ve que la señal `sign_en` queda en `'1'`, esto se debe a que todavía no se dio una condición para que se entre en el `process()` del bloque `RX Control` donde la próxima vez que se entre, el estado será `IDLE` y dicha señal valdrá `'0'`.

Las cuentas de rotación fueron corroboradas con la hoja de cálculo que se adjunta con este trabajo.

3.1.2. Rotación continua en sentido horario

Se simuló una rotación continua en sentido horario, por lo que el comando que se envió al sistema luego de esperar la conversión inicial fue:

`ROT C H`

Después de 4 ms se envió:

`S`

Los resultados se pueden ver en las siguientes capturas.

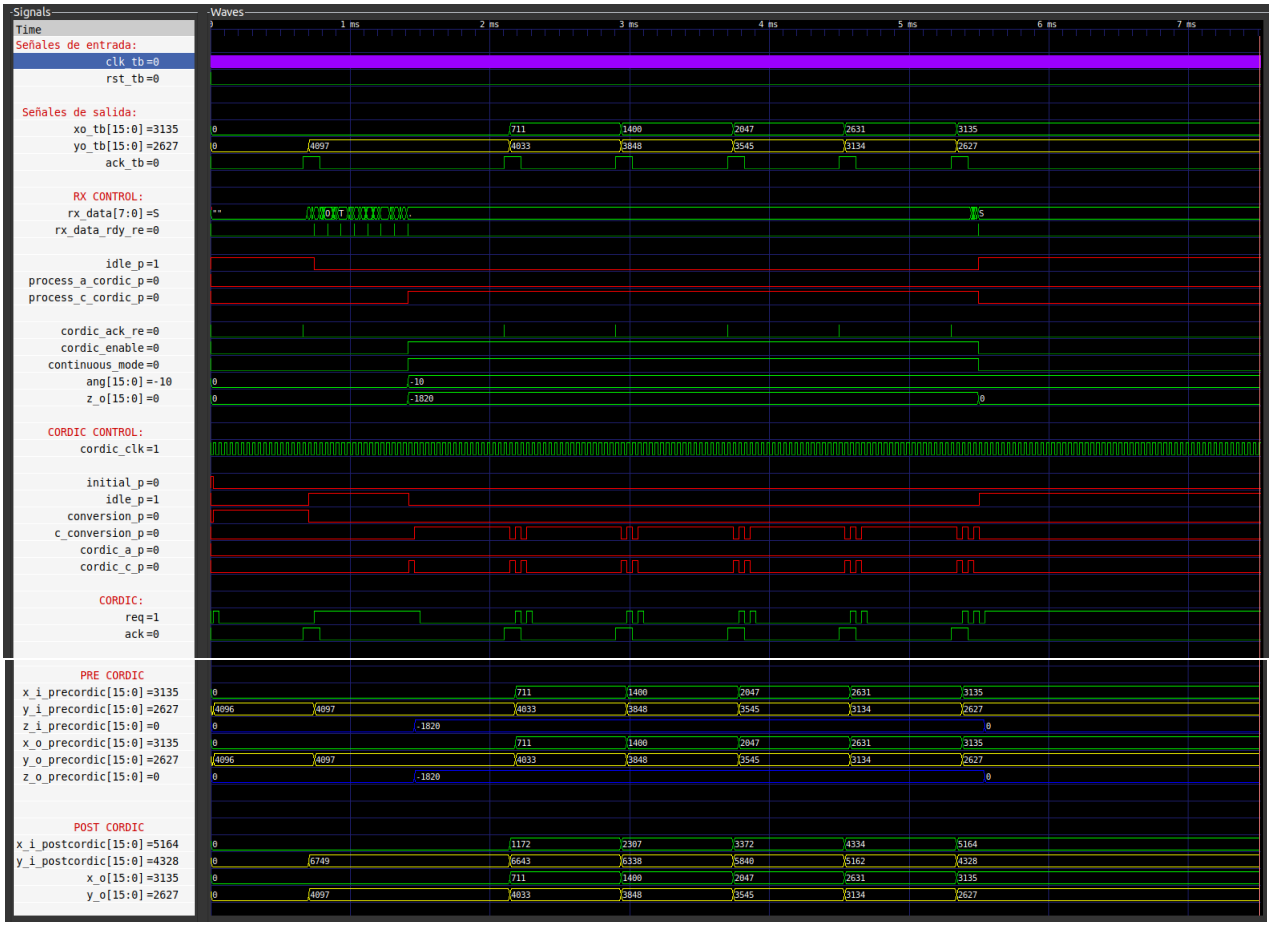

Figura 6: Simulación de rotación continua en sentido horario

Analizando la Figura 6, vemos que la simulación se realzó correctamente. Viendo que una vez que se recibe el comando, el ángulo a rotar se coloca en $-10°$ y comienza a realizar las rotaciones continuamente. Los cambios de estados también siguen la lógica adecuada y se puede ver como al parar la rotación (con el comando `S`) se deja de rotar y se conserva el valor de la última conversión.

Nuevamente, los resultados de las rotaciones fueron corroborados con la hoja de cálculo ya mencionada a lo largo de este trabajo.

3.1.3. Código VHDL del *test bench* utilizado en la sección 3.1.1

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4
5  entity tp_testbench_1 is
6  end tp_testbench_1;
7
8  architecture behavioral of tp_testbench_1 is
9
10 constant N       : natural := 16;
11 signal clk_tb    : std_logic := '0';
12 signal rst_tb    : std_logic := '1';
```

```
13  signal rxd_tb   : std_logic := '1';
14  signal ack_tb   : std_logic := '0';
15  signal xo_tb, yo_tb : Std_logic_vector(N-1 downto 0) := (others => '0');
16
17  constant FRECUENCIA : integer := 125E6; --En MHz
18  constant PERIODO    : time    := 1 sec/FRECUENCIA; -- En us.
19  constant BAUD_RATE  : integer := 115200;
20  signal   detener    : boolean := false;
21  constant nb_ack     : integer := 2;
22
23
24  begin
25
26
27  clk_tb <= not clk_tb after 4 ns;
28  rst_tb <= '1', '0' after 9 ns;
29
30  TEST:
31  process is begin
32      report "Se inicia la prueba"
33      severity note;
34
35      wait until rst_tb = '0';
36      wait until ack_tb = '1';
37      wait for 3 ns;
38
39
40      rxd_tb <= '1';  -- IDLE
41      wait for 8681 ns; -- Tiempo de un bit: 1/BAUD_RATE. En este caso BAUD_RATE = 115200 =>
        ↪  8680,555 => 8681 redondeando
42
43      rxd_tb <= '0'; -- START
44      wait for 8681 ns;
45
46      -- Envio letra R
47      rxd_tb <= '0';  --bit(0) de la R = "01010010", se envia de derecha a izquierda.
        ↪  Empezando por el LSB
48      wait for 8681 ns;
49
50      rxd_tb <= '1';  --bit(1) de la R
51      wait for 8681 ns;
52
53      rxd_tb <= '0';  --bit(2) de la R
54      wait for 8681 ns;
55
56      rxd_tb <= '0';  --bit(3) de la R
57      wait for 8681 ns;
58
59      rxd_tb <= '1';  --bit(4) de la R
60      wait for 8681 ns;
61
62      rxd_tb <= '0';  --bit(5) de la R
63      wait for 8681 ns;
64
65      rxd_tb <= '1';  --bit(6) de la R
66      wait for 8681 ns;
```

```
67
68      rxd_tb <= '0';  --bit(7) de la R
69      wait for 8681 ns;
70
71      rxd_tb <= '1';   -- STOP
72      wait for 8681 ns;
73
74      rxd_tb <= '1';   -- IDLE
75      wait for 8681 ns;
76
77      -- Envio letra O
78
79      rxd_tb <= '0';   -- START
80      wait for 8681 ns;
81
82      rxd_tb <= '1';  --bit(0) de la O = "01001111"
83      wait for 8681 ns;
84
85      rxd_tb <= '1';  --bit(1) de la O
86      wait for 8681 ns;
87
88      rxd_tb <= '1';  --bit(2) de la O
89      wait for 8681 ns;
90
91      rxd_tb <= '1';  --bit(3) de la O
92      wait for 8681 ns;
93
94      rxd_tb <= '0';  --bit(4) de la O
95      wait for 8681 ns;
96
97      rxd_tb <= '0';  --bit(5) de la O
98      wait for 8681 ns;
99
100     rxd_tb <= '1';  --bit(6) de la O
101     wait for 8681 ns;
102
103     rxd_tb <= '0';  --bit(7) de la O
104     wait for 8681 ns;
105
106     rxd_tb <= '1';   -- STOP
107     wait for 8681 ns;
108
109     rxd_tb <= '1';   -- IDLE
110     wait for 8681 ns;
111
112     -- Envio letra T
113
114     rxd_tb <= '0';   -- START
115     wait for 8681 ns;
116
117     rxd_tb <= '0';  --bit(0) de la T = "01010100"
118     wait for 8681 ns;
119
120     rxd_tb <= '0';  --bit(1) de la T
121     wait for 8681 ns;
122
```

```
123     rxd_tb <= '1';  --bit(2) de la T
124     wait for 8681 ns;
125
126     rxd_tb <= '0';  --bit(3) de la T
127     wait for 8681 ns;
128
129     rxd_tb <= '1';  --bit(4) de la T
130     wait for 8681 ns;
131
132     rxd_tb <= '0';  --bit(5) de la T
133     wait for 8681 ns;
134
135     rxd_tb <= '1';  --bit(6) de la T
136     wait for 8681 ns;
137
138     rxd_tb <= '0';  --bit(7) de la T
139     wait for 8681 ns;
140
141     rxd_tb <= '1';   -- STOP
142     wait for 8681 ns;
143
144     rxd_tb <= '1';   -- IDLE
145     wait for 8681 ns;
146
147     -- Envio ESACIO
148
149     rxd_tb <= '0';   -- START
150     wait for 8681 ns;
151
152     rxd_tb <= '0';  --bit(0) de ESPACIO = "00100000"
153     wait for 8681 ns;
154
155     rxd_tb <= '0';  --bit(1) de ESPACIO
156     wait for 8681 ns;
157
158     rxd_tb <= '0';  --bit(2) de ESPACIO
159     wait for 8681 ns;
160
161     rxd_tb <= '0';  --bit(3) de ESPACIO
162     wait for 8681 ns;
163
164     rxd_tb <= '0';  --bit(4) de ESPACIO
165     wait for 8681 ns;
166
167     rxd_tb <= '1';  --bit(5) de ESPACIO
168     wait for 8681 ns;
169
170     rxd_tb <= '0';  --bit(6) de ESPACIO
171     wait for 8681 ns;
172
173     rxd_tb <= '0';  --bit(7) de ESPACIO
174     wait for 8681 ns;
175
176     rxd_tb <= '1';   -- STOP
177     wait for 8681 ns;
178
```

```
179     rxd_tb <= '1';   -- IDLE
180     wait for 8681 ns;
181
182     -- Envio letra A
183
184     rxd_tb <= '0';   -- START
185     wait for 8681 ns;
186
187     rxd_tb <= '1';  --bit(0) de la A = "01000001"
188     wait for 8681 ns;
189
190     rxd_tb <= '0';  --bit(1) de la A
191     wait for 8681 ns;
192
193     rxd_tb <= '0';  --bit(2) de la A
194     wait for 8681 ns;
195
196     rxd_tb <= '0';  --bit(3) de la A
197     wait for 8681 ns;
198
199     rxd_tb <= '0';  --bit(4) de la A
200     wait for 8681 ns;
201
202     rxd_tb <= '0';  --bit(5) de la A
203     wait for 8681 ns;
204
205     rxd_tb <= '1';  --bit(6) de la A
206     wait for 8681 ns;
207
208     rxd_tb <= '0';  --bit(7) de la A
209     wait for 8681 ns;
210
211     rxd_tb <= '1';   -- STOP
212     wait for 8681 ns;
213
214     rxd_tb <= '1';   -- IDLE
215     wait for 8681 ns;
216
217     -- Envio ESACIO
218
219     rxd_tb <= '0';   -- START
220     wait for 8681 ns;
221
222     rxd_tb <= '0';  --bit(0) de ESPACIO = "00100000"
223     wait for 8681 ns;
224
225     rxd_tb <= '0';  --bit(1) de ESPACIO
226     wait for 8681 ns;
227
228     rxd_tb <= '0';  --bit(2) de ESPACIO
229     wait for 8681 ns;
230
231     rxd_tb <= '0';  --bit(3) de ESPACIO
232     wait for 8681 ns;
233
234     rxd_tb <= '0';  --bit(4) de ESPACIO
```

```
235     wait for 8681 ns;
236
237     rxd_tb <= '1';  --bit(5) de ESPACIO
238     wait for 8681 ns;
239
240     rxd_tb <= '0';  --bit(6) de ESPACIO
241     wait for 8681 ns;
242
243     rxd_tb <= '0';  --bit(7) de ESPACIO
244     wait for 8681 ns;
245
246     rxd_tb <= '1';   -- STOP
247     wait for 8681 ns;
248
249     rxd_tb <= '1';   -- IDLE
250     wait for 8681 ns;
251
252     -- Envio numeros
253     -- SIGNO MENOS
254     rxd_tb <= '0';   -- START
255     wait for 8681 ns;
256
257     rxd_tb <= '1';  --bit(0) de - = "00101101"
258     wait for 8681 ns;
259
260     rxd_tb <= '0';  --bit(1) de -
261     wait for 8681 ns;
262
263     rxd_tb <= '1';  --bit(2) de -
264     wait for 8681 ns;
265
266     rxd_tb <= '1';  --bit(3) de -
267     wait for 8681 ns;
268
269     rxd_tb <= '0';  --bit(4) de -
270     wait for 8681 ns;
271
272     rxd_tb <= '1';  --bit(5) de -
273     wait for 8681 ns;
274
275     rxd_tb <= '0';  --bit(6) de -
276     wait for 8681 ns;
277
278     rxd_tb <= '0';  --bit(7) de -
279     wait for 8681 ns;
280
281     rxd_tb <= '1';   -- STOP
282     wait for 8681 ns;
283
284     rxd_tb <= '1';   -- IDLE
285     wait for 8681 ns;
286
287     --4
288     rxd_tb <= '0';   -- START
289     wait for 8681 ns;
290
```

```
291     rxd_tb <= '0';  --bit(0) de 4 = "00110100"
292     wait for 8681 ns;
293
294     rxd_tb <= '0';  --bit(1) de 4
295     wait for 8681 ns;
296
297     rxd_tb <= '1';  --bit(2) de 4
298     wait for 8681 ns;
299
300     rxd_tb <= '0';  --bit(3) de 4
301     wait for 8681 ns;
302
303     rxd_tb <= '1';  --bit(4) de 4
304     wait for 8681 ns;
305
306     rxd_tb <= '1';  --bit(5) de 4
307     wait for 8681 ns;
308
309     rxd_tb <= '0';  --bit(6) de 4
310     wait for 8681 ns;
311
312     rxd_tb <= '0';  --bit(7) de 4
313     wait for 8681 ns;
314
315     rxd_tb <= '1';   -- STOP
316     wait for 8681 ns;
317
318     rxd_tb <= '1';   -- IDLE
319     wait for 8681 ns;
320
321     --5
322     rxd_tb <= '0';   -- START
323     wait for 8681 ns;
324
325     rxd_tb <= '1';  --bit(0) de 5 = "00110101"
326     wait for 8681 ns;
327
328     rxd_tb <= '0';  --bit(1) de 5
329     wait for 8681 ns;
330
331     rxd_tb <= '1';  --bit(2) de 5
332     wait for 8681 ns;
333
334     rxd_tb <= '0';  --bit(3) de 5
335     wait for 8681 ns;
336
337     rxd_tb <= '1';  --bit(4) de 5
338     wait for 8681 ns;
339
340     rxd_tb <= '1';  --bit(5) de 5
341     wait for 8681 ns;
342
343     rxd_tb <= '0';  --bit(6) de 5
344     wait for 8681 ns;
345
346     rxd_tb <= '0';  --bit(7) de 5
```

```
347      wait for 8681 ns;
348
349      rxd_tb <= '1';   -- STOP
350      wait for 8681 ns;
351
352      rxd_tb <= '1';   -- IDLE
353      wait for 8681 ns;
354
355      -- Envio ENTER (Carriage Return)
356
357      rxd_tb <= '0';   -- START
358      wait for 8681 ns;
359
360      rxd_tb <= '1';  --bit(0) de ENTER = "00001101"
361      wait for 8681 ns;
362
363      rxd_tb <= '0';  --bit(1) de ENTER
364      wait for 8681 ns;
365
366      rxd_tb <= '1';  --bit(2) de ENTER
367      wait for 8681 ns;
368
369      rxd_tb <= '1';  --bit(3) de ENTER
370      wait for 8681 ns;
371
372      rxd_tb <= '0';  --bit(4) de ENTER
373      wait for 8681 ns;
374
375      rxd_tb <= '0';  --bit(5) de ENTER
376      wait for 8681 ns;
377
378      rxd_tb <= '0';  --bit(6) de ENTER
379      wait for 8681 ns;
380
381      rxd_tb <= '0';  --bit(7) de ENTER
382      wait for 8681 ns;
383
384    rxd_tb <= '1'; -- STOP
385    wait for 8681 ns;
386
387    rxd_tb <= '1'; -- IDLE
388    wait for 2 ms;
389    --wait for 8681 ns;
390
391    report "Se envio comando de rotacion"
392    severity note;
393
394    --wait until ack_tb = '1';
395
396    report "Termina Prueba"
397    severity note;
398
399    detener <= true;
400
401    -- Se aborta la simulacion
402     assert false report
```

```
403         "Fin de la simulacion" severity failure;
404
405 end process TEST;
406
407 DUT: entity work.top(behavioral)
408 generic map (N => 16, BAUD_RATE => BAUD_RATE, CLOCK_RATE => FRECUENCIA)
409 port map (
410     clk => clk_tb,
411     rst => rst_tb,
412     rxd_pin => rxd_tb,
413     ack => ack_tb,
414     x_o => xo_tb,
415     y_o => yo_tb
416 );
417
418 end behavioral;
```

3.2. Prueba en FPGA del servidor remoto

Para poder probar el código implementado en la placa `Arty Z7-10` alojada en el servidor remoto, se tuvo que implementar un bloque VIO (disponible dentro del *IP Catalog* de Vivado) el cual tendrá 3 entradas (salidas `ack`, coordenada x y coordenada y). El bloque VIO quedó configurado como se muestra en la Figura 7.

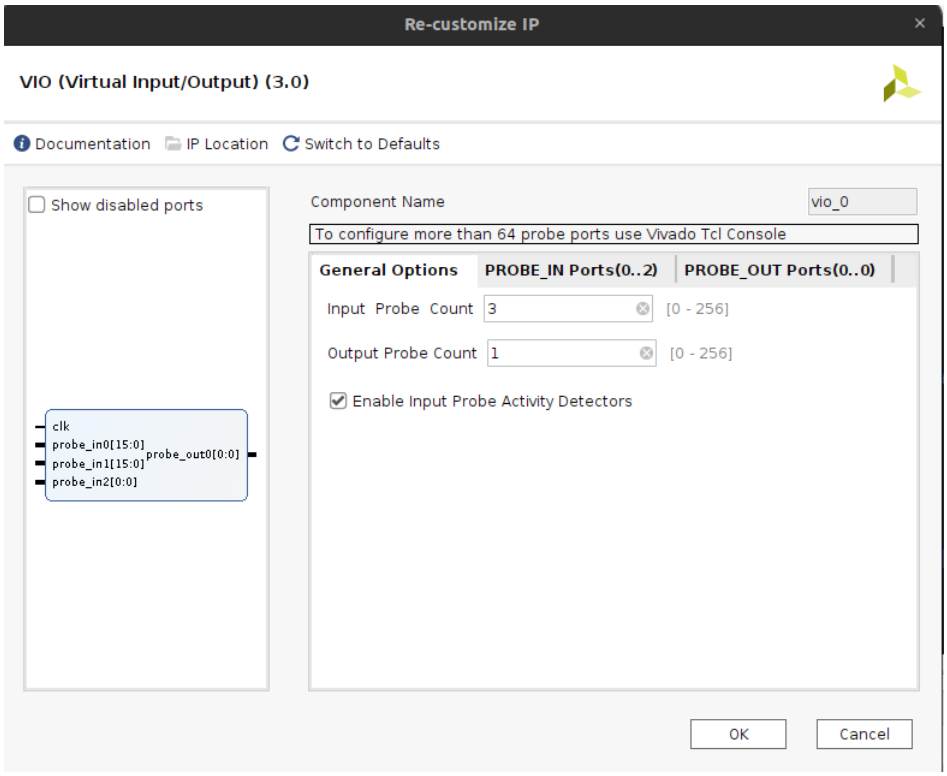

Figura 7: Configuración del bloque VIO

Se modificó acordemente el archivo de restricciones específico para la placa `Arty Z7-10` (obtenido del github de Xiling). Para poder incluir el bloque VIO dentro del bloque `top level`, se tuvo que modificar el código como se muestra en el código adjunto a continuación.

```
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 top is
7      generic(
8          N : natural := 16;
9          BAUD_RATE : integer := 115200;
10         CLOCK_RATE: integer := 125E6
11     );
12     Port (
13         clk      : in  std_logic;
14         --rst      : in  std_logic;
15         rxd_pin  : in std_logic
16         --ack      : out std_logic;
17         --x_o, y_o : out std_logic_vector(N-1 downto 0)
18     );
19 end top;
20
21 architecture behavioral of top is
22
23     constant N_COUNT : natural := 22;
24
25     signal cordic_enable   : std_logic := '0';
26     signal continuous_mode : std_logic := '0';
27     signal ang             : std_logic_vector(N-1 downto 0) := (others => '0');
28     signal ack_aux         : std_logic := '0';
29     signal stop_cordic     : std_logic := '0';
30
31     -- Señales para bloque VIO
32     signal rst_i : std_logic_vector(0 downto 0);
33     signal ack_o : std_logic_vector(0 downto 0);
34     signal x_o_vio, y_o_vio : std_logic_vector(15 downto 0);
35     signal rst : std_logic;
36
37     COMPONENT vio_0
38       PORT (
39         clk : IN STD_LOGIC;
40         probe_in0 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
41         probe_in1 : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
42         probe_in2 : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
43         probe_out0 : OUT STD_LOGIC_VECTOR(0 DOWNTO 0)
44       );
45     END COMPONENT;
46
47 begin
48
49     --ack <= ack_aux;
50     -- Hago esto para operar de la misma forma que antes sin cambiar el codigo (Ahora rst y
       ↪  ack son std_logic_vector !!).
51     ack_o <= "1" when ack_aux = '1' else "0";
52     rst <= '1' when rst_i = "1" else '0';
53
54 top_vio : vio_0
```

```
55    PORT MAP (
56      clk => clk,
57      probe_in0 => x_o_vio,
58      probe_in1 => y_o_vio,
59      probe_in2 => ack_o,
60      probe_out0 => rst_i
61    );
62
63      RX_CONTROL: entity work.rx_control(behavioral)
64      generic map(N => N,
65                  BAUD_RATE => BAUD_RATE,
66                  CLOCK_RATE => CLOCK_RATE
67      )
68      port map(
69          clk => clk,
70          rst => rst,
71          rxd_pin => rxd_pin,
72          cordic_ack => ack_aux,
73          cordic_enable => cordic_enable,
74          continuous_mode => continuous_mode,
75          stop_cordic => stop_cordic,
76          z_o => ang
77      );
78
79      CORDIC_CONTROL: entity work.cordic_control(behavioral)
80      generic map(N => N, N_COUNT => N_COUNT)
81      port map(
82          clk => clk,
83          rst => rst,
84          ena => cordic_enable,
85          mode => continuous_mode,
86          stop_cordic => stop_cordic,
87          ack => ack_aux,
88          ang => ang,
89          x_o => x_o_vio,
90          y_o => y_o_vio
91      );
92
93  end behavioral;
```

Una vez estuvo todo configurado, se generó el *BitStream* (archivo `.bin`) y se accedió al *Hardware Manager* de Vivado, se conectó a la placa del servidor remoto y se programó con el *BitStream* generado.

Luego se conectó vía *ssh* al servidor remoto y se utilizó *minicom* para comunicarse con la placa vía UART. Este procedimiento se puede ver en la Figura 8.

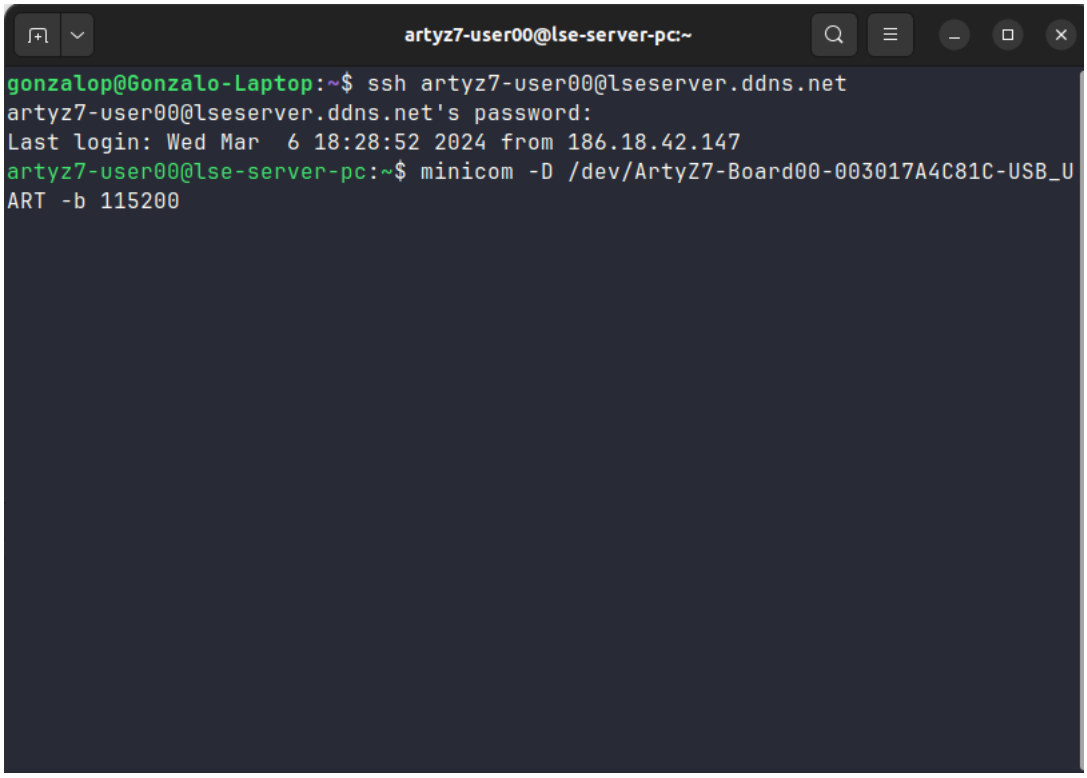

Figura 8: Conexión *ssh* con la placa en el servidor remoto

En las siguientes imágenes se puede ver la prueba del programa. En la Figura 9, se observa el accionar del reset y en la Figura 10 se observa el comportamiento luego del comando introducido.

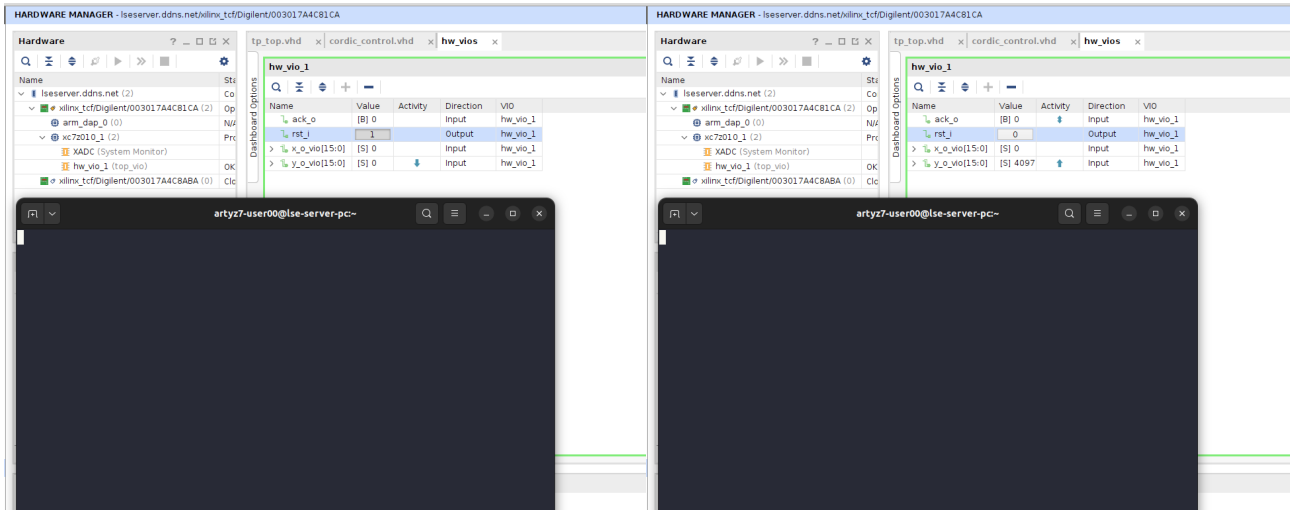

Figura 9: Comportamiento luego de accionar el `rst`

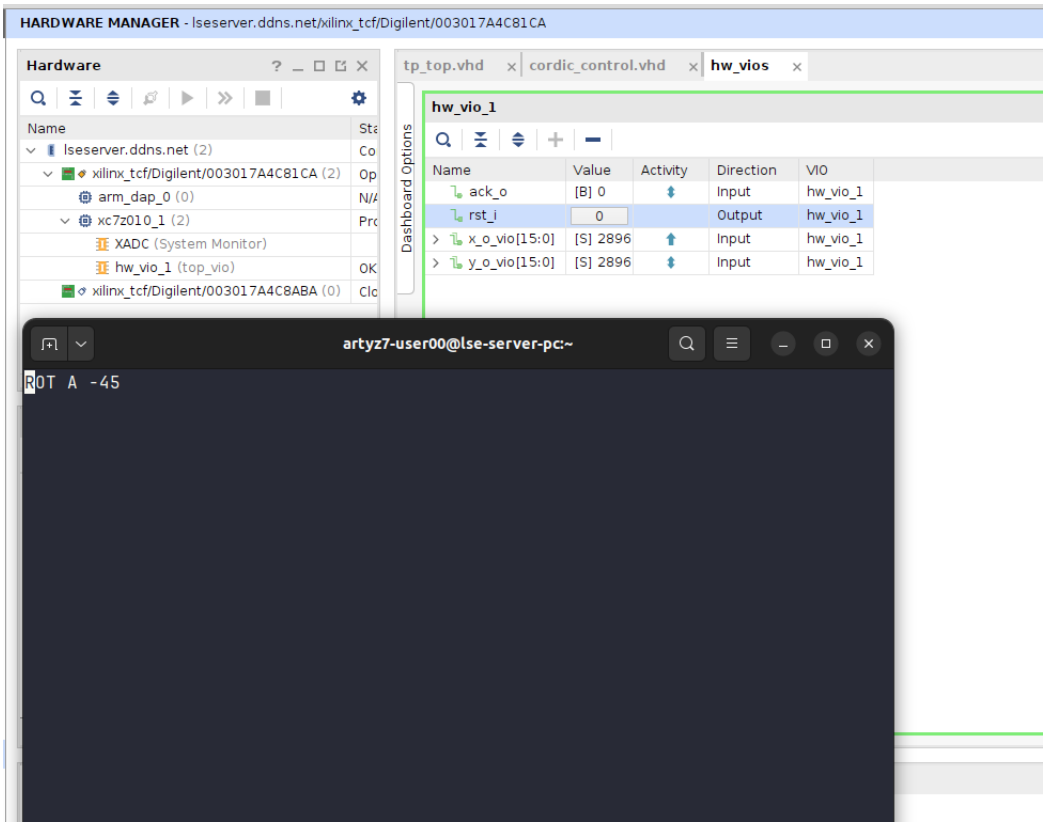

Figura 10: Comportamiento luego de enviar comando de rotación

Como se puede observar, el comportamiento es el esperado y también es acorde a los visto en las simulaciones mostradas en la sección 3.1.1.

4. Conclusiones

Dado lo hecho en este trabajo y teniendo en cuenta todos los contratiempos dados por la falta de una placa física con la que trabajar, se puede afirmar que se cumplieron con los objetivos planteados.

La dificultad de este trabajo estuvo en la implementación de los bloques `RX Control` y `Cordic Control`. Con los cuales tienen la tarea de controlar lo recibido mediante UART y de anexar el resto de los bloques. Finalmente, se pudo implementar todo, sin problemas con la síntesis e implementación del código realizada en el *software* Vivado. Sin tener, además, *warnings* significativos.

Como punto a mejorar, se puede mencionar que sería de interés poder sincronizar los bloques `RX Control` y `Cordic Control`, para evitar comportamientos erráticos cuando se envía algo por UART y todavía no terminó de procesarse la inicialización del sistema (primera rotación para las coordenadas iniciales). Sin embargo, esta inicialización es lo suficientemente rápida y el comportamiento actual tiene sentido si se lo piensa en conjunto con la parte faltante del trabajo (controlador de video). Ya que es razonable esperar a ver el vector ubicado en el monitor antes de empezar a enviar comandos. De todas formas, parece interesante resolverlo como medida de seguridad y para aportar más robustez al programa.

Por otro lado, es obvio que en un futuro sería ideal poder contar con placas físicas con las

cuales trabajar y poder implementar la parte gráfica (Controlador de video VGA) que en el presente trabajo quedo inconcluso. Dicho controlador resulta de mucho interés, y lograría que los resultados mostrados sean mejor visualizados que de la forma en que se mostraron en el trabajo.