```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date: 19.02.2016 09:45:52
7  // Design Name:
8  // Module Name: CORDIC_Coprocessor
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22
23 module CORDIC_Coprocessor #(parameter W=32 , E=8, M=23)
24 (
25 //Input Signals
26 input wire clk,                         //  Reloj del sistema.
27 input wire rst_cordic,                  //  Señal de reset del sistema.
28 input wire beg_fsm_cordic,              //  Señal de inicio de la maquina de estados ↵
   del módulo CORDIC.
29 input wire ack_cordic,                  //  Señal de acknowledge proveniente de otro ↵
   módulo que indica que ha recibido el resultado del modulo CORDIC.
30 input wire operation,                  //  Señal que indica si se realiza la ↵
   operacion seno(1'b1) o coseno(1'b0).
31 input wire ready_add_subt,             //  Señal que indica que se ha realizado la ↵
   operacion de suma/resta en punto flotante.
32
33 input wire [W-1:0] data_in,            //  Dato de entrada, contiene el angulo que ↵
   se desea calcular en radianes.
34 input wire [1:0] shift_region_flag,    //  Señal que indica si el ángulo a calcular ↵
   esta fuera del rango de calculo del algoritmo CORDIC.
35 input wire [W-1:0] result_add_subt,    //  Dato de entrada, contiene el resultado ↵
   del módulo de suma/resta.
36
37 //Output Signals
38 output reg ready_cordic,               //  Señal de salida que indica que se ha ↵
   completado el calculo del seno/coseno.
39 output reg beg_add_subt,               //  Señal de salida que indica que se debe ↵
   de iniciar el modulo de suma/resta.
40 output reg ack_add_subt,               //  Señal que le indica al modulo de ↵
   suma/resta que se recibio el resultado de este modulo correctamente.
41 output reg op_add_subt,                //  Señal hacia el módulo de suma/resta que ↵
   indica si se va a realizar una suma(1'b0) o una resta(1'b1).
42
```

```
43  output reg [W-1:0] add_subt_dataA,      //  Bus de datos hacia el modulo de suma/resta con el valor al que se le desea aplicar dicha operacion.
44  output reg [W-1:0] add_subt_dataB,      //  Bus de datos hacia el modulo de suma/resta con el valor al que se le desea aplicar dicha operacion.
45  output reg [W-1:0] data_output          //  Bus de datos con el valor final del angulo calculado.
46  );
47
48  /*generate
49      if(W==32)
50      begin*/
51          parameter x0 = 32'h3f1b74ee;            //  x0 = 0.607252935008881, valor inicial de la variable X.
52          parameter y0 = 32'h00000000;            //  y0 = 0, valor inicial de la variable Y.
53          parameter up = 1'b0;                    //  Valor por defecto para que el contador realize la cuenta hacia abajo.
54          parameter syn_clr = 1'b0;               //
55          parameter d_var = 2'b10;                //  Valor por defecto que se le carga al contador de variables.
56          parameter d_iter = 5'b11111;            //  Valor por defecto que se le carga al contador de iteraciones.
57      /*end
58
59      else
60      begin
61          parameter x0 = 64'h3fe36e9db5086bc9;    //  x0 = 0.607252935008881, valor inicial de la variable X.
62          parameter y0 = 64'h0000000000000000;    //  y0 = 0, valor inicial de la variable Y.
63          parameter up = 1'b0;                    //  Valor por defecto para que el contador realize la cuenta hacia abajo.
64          parameter syn_clr = 1'b0;               //
65          parameter d_var = 2'b10;                //  Valor por defecto que se le carga al contador de variables.
66          parameter d_iter = 5'b11111;            //  Valor por defecto que se le carga al contador de iteraciones.
67      end
68  endgenerate*/
69
70
71  //-------------------------------------------------------------------------------------------------
72
73  //Signal declaration
74
75  //ENABLE
76  wire enab_d_ff1_RB1;                                    //  Enable de la primera linea de registros.
77  wire enab_d_ff2_RB2;                                    //  Enable de la segunda linea de registros.
78  wire enab_d_ff3_sh_exp_x, enab_d_ff3_sh_exp_y;          //  Enable de los registros que guardan el valor desplazado de X y Y.
```

```
79  wire enab_d_ff3_LUT;                                    //  Enable del registro
    que guarda el valor obtenido de la LUT
80  wire enab_d_ff3_sign;                                   //  Enable del registro
    que guarda el valor del signo, dependiendo del modo del algoritmo.
81  wire enab_d_ff4_Xn, enab_d_ff4_Yn, enab_d_ff4_Yn;       //  Enable de los
    registros que guardan los datos provenientes del modulo de suma/resta.
82  wire enab_d_ff5;                                        //  Enable del registo
    que guarda el valor de salida antes de pasar por el moduo de cambio de signo.
83  wire enab_d_ff5_data_out;                               //  Enable del registo
    que guarda el valor de salida final, listo para enviarse al procesador.
84  wire enab_cont_iter, enab_cont_var;                     //  Enable de los
    contadores de variable e iteracion
85  wire load_con_iter, load_cont_var;                      //  Señal de carga de un
    valor en los contadores de variable e iteraciones.
86
87
88
89  //SELECTION
90  wire sel_mux_1, mode, sel_mux_3;                        //  Señales de seleccion
    provenientes de la maquina de estados.
91  wire [1:0] sel_mux_2;                                   //  Señal de seleccion
    que se activa dependiendo de la variable que se este calculando.
92
93  //DATA WIRES
94  wire d_ff1_operation_out;                               //  Salida del registro
    que guarda el dato de entrada de la operacion a realizar, coseno(1'b0) o seno(1'b1)
95  wire [1:0] d_ff1_shift_region_flag_out;                 //  Salida del registro
    que guarda el dato de entrada que indica si el ángulo a calcular esta fuera del
    rango de calculo del algoritmo CORDIC.
96  wire [W-1:0] d_ff1_X, d_ff1_Y, d_ff1_Z;                 //  Salidas de los
    registros que guardan los valores iniciales de las variables X, Y y Z.
97  wire [W-1:0] d_ff_Xn, d_ff_Yn, d_ff_Zn;                 //  Salidas de los
    registros que guardan los valores de las variables X, Y y Z despues de cada iteracion.
98  wire [W-1:0] first_mux_X, first_mux_Y, first_mux_Z;     //  Salidas de los mux
    que escogen entre un valor inicial y el valor obtenido en una iteracion.
99  wire [W-1:0] d_ff2_X, d_ff2_Y, d_ff2_Z;                 //  Salidas de los
    registros que guardan los valores provenientes de la primera linea de mux.
100 wire sign;                                              //  Salida del mux que
    escoge entre el signo de Y o Z, dependiendo del modo, ya sea rotacion o vectorizacion.
101 reg [W-1:0] data_out_LUT;                               //  Salida del modulo
    generate que genera la LUT necesaria dependiendo del ancho de palabra.
102 wire [4:0] cont_iter_out;                               //  Salida del contador
    que cuenta las iteraciones realizadas.
103 wire [E-1:0] sh_exp_x, sh_exp_y;                        //  Salidas de los
    sumadores de punto fijo que realizan los desplazamientos.
104 wire [W-1:0] d_ff3_sh_x_out, d_ff3_sh_y_out;            //  Salida del registro
    que guarda el valor de X y Y luego de realizar los desplazamientos.
105 wire [W-1:0] d_ff3_LUT_out;                             //  Salida del registro
    que guarda el valor de la LUT.
106 wire d_ff3_sign_out;                                    //  Salida del registro
    que guarda el valor del signo.
107 wire [1:0] cont_var_out;                                //  Salida del contador
    que cuenta las variables calculadas.
```

```
108   wire [W-1:0] mux_sal;                                    //  Salida del mux final para colocar en la salida el valor deseado.
109   wire [W-1:0] data_output2;                               //  Salida del registro antes del cambio de signo.
110   wire [W-1:0] sign_inv_out;                               //  Salida del modulo de inversion de signo, dependiendo de si se el angulo de entrada estaba fuera del rango de calculo del algoritmo CORDIC.
111   wire min_tick_iter,max_tick_iter;                        //  Señales que indican cuando se ha alcanzado el valor mas bajo y masalto de cuenta, correspondientemente en el contador de iteraciones.
112   wire min_tick_var,max_tick_var;                          //  Señales que indican cuando se ha alcanzado el valor mas bajo y masalto de cuenta, correspondientemente en el contador de variables.
113
114   //-------------------------------------------------------------------------------------------------------------------
115
116   //Instanciacion
117   ////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
118   //Primera Etapa
119
120   //FF_D para guardar la señal de entrada proveniente del procesador que define si se quiere realizar el calculo de un seno o un coseno.
121   d_ff_en # (.W(1)) d_ff_operation
122   (
123   .clk(clk),
124   .rst(rst_cordic),
125   .enable(enab_d_ff_RB1),
126   .D(operation),
127   .Q(d_ff1_operation_out)
128   );
129
130   //FF_D para guardar el dato de entrada que define si hay un desplazamiento hacia el rango de cálculo del alg. CORDIC.
131   d_ff_en # (.W(2)) d_ff_shift_region_flag
132   (
133   .clk(clk),
134   .rst(rst_cordic),
135   .enable(enab_d_ff_RB1),
136   .D(shift_region_flag),
137   .Q(d_ff1_shift_region_flag_out)
138   );
139
140   //FF_D para guardar el dato de entrada que define el valor inicial de la variable X.
141   d_ff_en # (.W(W)) d_ff1_x
142   (
143   .clk(clk),
144   .rst(rst_cordic),
145   .enable(enab_d_ff_RB1),
146   .D(x0),
147   .Q(d_ff1_X)
148   );
```

```
149
150   //FF_D para guardar el dato de entrada que define el valor inicial de la variable Y.
151   d_ff_en # (.W(W)) d_ff1_y
152   (
153   .clk(clk),
154   .rst(rst_cordic),
155   .enable(enab_d_ff_RB1),
156   .D(y0),
157   .Q(d_ff1_Y)
158   );
159
160   //FF_D para guardar el dato de entrada que define el valor inicial de la variable Z.
161   d_ff_en # (.W(W)) d_ff1_z
162   (
163   .clk(clk),
164   .rst(rst_cordic),
165   .enable(enab_d_ff_RB1),
166   .D(data_in),
167   .Q(d_ff1_Z)
168   );
169
170   ////////////////////////////////////////////////////////////////////////////////
      ////////////////////////////////////////////////////////////////////////////////
171   //Segunda etapa
172
173   //Mux de 2x1 para regular cual valor de la variable X se ocupa en el calculo.
174   Mux_2x1 #(.W(W)) mux_2x1_x
175   (
176   .select(sel_mux_1),
177   .ch_0(d_ff1_X),
178   .ch_1(d_ff_Xn),
179   .data_out(first_mux_X)
180   );
181
182   //Instanciacion de un mux de 2x1 para regular cual valor de la variable Y se ocupa
      en el calculo
183   Mux_2x1 #(.W(W)) mux_2x1_y
184   (
185   .select(sel_mux_1),
186   .ch_0(d_ff1_Y),
187   .ch_1(d_ff_Yn),
188   .data_out(first_mux_Y)
189   );
190
191   //Instanciacion de un mux de 2x1 para regular cual valor de la variable Z se ocupa
      en el calculo.
192   Mux_2x1 #(.W(W)) mux_2x1_z
193   (
194   .select(sel_mux_1),
195   .ch_0(d_ff1_Z),
196   .ch_1(d_ff_Zn),
197   .data_out(first_mux_Z)
198   );
```

```
199
200 //FF_D para guardar el dato de salida de los mux 2x1 de la variable X.
201 d_ff_en # (.W(W)) d_ff2_x
202 (
203 .clk(clk),
204 .rst(rst_cordic),
205 .enable(enab_d_ff2_RB2),
206 .D(first_mux_X),
207 .Q(d_ff2_X)
208 );
209
210 //FF_D para guardar el dato de salida de los mux 2x1 de la variable Y.
211 d_ff_en # (.W(W)) d_ff2_y
212 (
213 .clk(clk),
214 .rst(rst_cordic),
215 .enable(enab_d_ff2_RB2),
216 .D(first_mux_Y),
217 .Q(d_ff2_Y)
218 );
219
220 //FF_D para guardar el dato de salida de los mux 2x1 de la variable Z.
221 d_ff_en # (.W(W)) d_ff2_z
222 (
223 .clk(clk),
224 .rst(rst_cordic),
225 .enable(enab_d_ff2_RB2),
226 .D(first_mux_Z),
227 .Q(d_ff2_Z)
228 );
229
230 ////////////////////////////////////////////////////////////////////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////////////////////////
231 //Third Stage
232
233 //Instanciacion de un mux de 2x1 para escoger entre el signo de la variable Y y Z.
234 Mux_2x1 #(.W(1)) mux_2x1_signo
235 (
236 .select(mode),
237 .ch_0(d_ff2_Z[W-1]),
238 .ch_1(d_ff2_Y[W-1]),
239 .data_out(sign)
240 );
241
242 // Sintetiza las LUT de 32 o 64 bits, dependiendo del ancho de palabra que se
    especifica al inicio a la hora de la sintesis.
243 /*generate
244     if(W==32)
245     begin*/
246         always @*//LUT de 32 bits
247         begin
248             case (cont_iter_out)
249                 5'b00000: data_out_LUT <= 32'h3f490fdb;
```

```
250                     5'b00001: data_out_LUT <= 32'h3eed6338;
251                     5'b00010: data_out_LUT <= 32'h3e7adbb0;
252                     5'b00011: data_out_LUT <= 32'h3dfeadd5;
253                     5'b00100: data_out_LUT <= 32'h3d7faade;
254                     5'b00101: data_out_LUT <= 32'h3cffeaae;
255                     5'b00110: data_out_LUT <= 32'h3c7ffaab;
256                     5'b00111: data_out_LUT <= 32'h3bfffeab;
257                     5'b01000: data_out_LUT <= 32'h3b7fffab;
258                     5'b01001: data_out_LUT <= 32'h3affffeb;
259                     5'b01010: data_out_LUT <= 32'h3a7ffffb;
260                     5'b01011: data_out_LUT <= 32'h39ffffff;
261                     5'b01100: data_out_LUT <= 32'h39800000;
262                     5'b01101: data_out_LUT <= 32'h39000000;
263                     5'b01110: data_out_LUT <= 32'h38800000;
264                     5'b01111: data_out_LUT <= 32'h38000000;
265                     5'b10000: data_out_LUT <= 32'h37800000;
266                     5'b10001: data_out_LUT <= 32'h37000000;
267                     5'b10010: data_out_LUT <= 32'h36800000;
268                     5'b10011: data_out_LUT <= 32'h36000000;
269                     5'b10100: data_out_LUT <= 32'h35800000;
270                     5'b10101: data_out_LUT <= 32'h35000000;
271                     5'b10110: data_out_LUT <= 32'h34800000;
272                     5'b10111: data_out_LUT <= 32'h34000000;
273                     5'b11000: data_out_LUT <= 32'h33800000;
274                     5'b11001: data_out_LUT <= 32'h33000000;
275                     5'b11010: data_out_LUT <= 32'h32800000;
276                     5'b11011: data_out_LUT <= 32'h32000000;
277                     5'b11100: data_out_LUT <= 32'h31800000;
278                     5'b11101: data_out_LUT <= 32'h31000000;
279                     5'b11110: data_out_LUT <= 32'h30800000;
280                     5'b11111: data_out_LUT <= 32'h30000000;
281                 /*      5'b100000: data_out_LUT <= 32'h3df0000000000000;
282                     5'b100001: data_out_LUT <= 32'h3de0000000000000;
283                     5'b100010: data_out_LUT <= 32'h3dd0000000000000;
284                     5'b100011: data_out_LUT <= 32'h3dc0000000000000;
285                     5'b100100: data_out_LUT <= 32'h3db0000000000000;
286                     5'b100101: data_out_LUT <= 32'h3da0000000000000;
287                     5'b100110: data_out_LUT <= 32'h3d90000000000000;
288                     5'b100111: data_out_LUT <= 32'h3d80000000000000;
289                     5'b101000: data_out_LUT <= 32'h3d70000000000000;
290                     5'b101001: data_out_LUT <= 32'h3d60000000000000;
291                     5'b101010: data_out_LUT <= 32'h3d50000000000000;
292                     5'b101011: data_out_LUT <= 32'h3d40000000000000;
293                     5'b101100: data_out_LUT <= 32'h3d30000000000000;
294                     5'b101101: data_out_LUT <= 32'h3d20000000000000;
295                     5'b101110: data_out_LUT <= 32'h3d10000000000000;
296                     5'b101111: data_out_LUT <= 32'h3d00000000000000;
297                     5'b110000: data_out_LUT <= 32'h3cf0000000000000;
298                     5'b110001: data_out_LUT <= 32'h3ce0000000000000;
299                     5'b110010: data_out_LUT <= 32'h3cd0000000000000;
300                     5'b110011: data_out_LUT <= 32'h3cc0000000000000;
301                     5'b110100: data_out_LUT <= 32'h3cb0000000000000;
302                     5'b110101: data_out_LUT <= 32'h3ca0000000000000;
```

```
303                     5'b110110: data_out_LUT <= 32'h3c90000000000000;
304                     5'b110111: data_out_LUT <= 32'h3c80000000000000;
305                     5'b111000: data_out_LUT <= 32'h3c70000000000000;
306                     5'b111001: data_out_LUT <= 32'h3c60000000000000;
307                     5'b111010: data_out_LUT <= 32'h3c50000000000000;
308                     5'b111011: data_out_LUT <= 32'h3c40000000000000;
309                     5'b111100: data_out_LUT <= 32'h3c30000000000000;
310                     5'b111101: data_out_LUT <= 32'h3c20000000000000;
311                     5'b111110: data_out_LUT <= 32'h3c10000000000000;
312                     5'b111111: data_out_LUT <= 32'h3c00000000000000;*/
313                     default:   data_out_LUT <= 32'h00000000;
314               endcase
315             end
316         /*end
317
318         else
319         begin
320             always @* //LUT de 64 bits
321             begin
322                 case (cont_iter_out)
323                     5'b00000: data_out_LUT <= 64'h3fe921fb54442d18;
324                     5'b00001: data_out_LUT <= 64'h3fddac670561bb4f;
325                     5'b00010: data_out_LUT <= 64'h3fcf5b75f92c80dd;
326                     5'b00011: data_out_LUT <= 64'h3fbfd5ba9aac2f6e;
327                     5'b00100: data_out_LUT <= 64'h3faff55bb72cfdea;
328                     5'b00101: data_out_LUT <= 64'h3f9ffd55bba97625;
329                     5'b00110: data_out_LUT <= 64'h3f8fff555bbb729b;
330                     5'b00111: data_out_LUT <= 64'h3f7fffd555bbba97;
331                     5'b01000: data_out_LUT <= 64'h3f6ffff5555bbbb7;
332                     5'b01001: data_out_LUT <= 64'h3f5ffffd5555bbbc;
333                     5'b01010: data_out_LUT <= 64'h3f4fffff55555bbc;
334                     5'b01011: data_out_LUT <= 64'h3f3fffffd55555bc;
335                     5'b01100: data_out_LUT <= 64'h3f2ffffff555555c;
336                     5'b01101: data_out_LUT <= 64'h3f1ffffffd555556;
337                     5'b01110: data_out_LUT <= 64'h3f0fffffff555555;
338                     5'b01111: data_out_LUT <= 64'h3effffffffd55555;
339                     5'b10000: data_out_LUT <= 64'h3eeffffffff55555;
340                     5'b10001: data_out_LUT <= 64'h3edfffffffffd5555;
341                     5'b10010: data_out_LUT <= 64'h3ecfffffffff5555;
342                     5'b10011: data_out_LUT <= 64'h3ebfffffffffd555;
343                     5'b10100: data_out_LUT <= 64'h3eafffffffffff555;
344                     5'b10101: data_out_LUT <= 64'h3e9fffffffffffd55;
345                     5'b10110: data_out_LUT <= 64'h3e8fffffffffff55;
346                     5'b10111: data_out_LUT <= 64'h3e7fffffffffffd5;
347                     5'b11000: data_out_LUT <= 64'h3e6fffffffffff5;
348                     5'b11001: data_out_LUT <= 64'h3e5fffffffffffd;
349                     5'b11010: data_out_LUT <= 64'h3e4fffffffffffff;
350                     5'b11011: data_out_LUT <= 64'h3e40000000000000;
351                     5'b11100: data_out_LUT <= 64'h3e30000000000000;
352                     5'b11101: data_out_LUT <= 64'h3e20000000000000;
353                     5'b11110: data_out_LUT <= 64'h3e10000000000000;
354                     5'b11111: data_out_LUT <= 64'h3e00000000000000;
355                     /*5'b100000: data_out_LUT <= 64'h3df0000000000000;
```

```
356                     5'b100001: data_out_LUT <= 64'h3de0000000000000;
357                     5'b100010: data_out_LUT <= 64'h3dd0000000000000;
358                     5'b100011: data_out_LUT <= 64'h3dc0000000000000;
359                     5'b100100: data_out_LUT <= 64'h3db0000000000000;
360                     5'b100101: data_out_LUT <= 64'h3da0000000000000;
361                     5'b100110: data_out_LUT <= 64'h3d90000000000000;
362                     5'b100111: data_out_LUT <= 64'h3d80000000000000;
363                     5'b101000: data_out_LUT <= 64'h3d70000000000000;
364                     5'b101001: data_out_LUT <= 64'h3d60000000000000;
365                     5'b101010: data_out_LUT <= 64'h3d50000000000000;
366                     5'b101011: data_out_LUT <= 64'h3d40000000000000;
367                     5'b101100: data_out_LUT <= 64'h3d30000000000000;
368                     5'b101101: data_out_LUT <= 64'h3d20000000000000;
369                     5'b101110: data_out_LUT <= 64'h3d10000000000000;
370                     5'b101111: data_out_LUT <= 64'h3d00000000000000;
371                     5'b110000: data_out_LUT <= 64'h3cf0000000000000;
372                     5'b110001: data_out_LUT <= 64'h3ce0000000000000;
373                     5'b110010: data_out_LUT <= 64'h3cd0000000000000;
374                     5'b110011: data_out_LUT <= 64'h3cc0000000000000;
375                     5'b110100: data_out_LUT <= 64'h3cb0000000000000;
376                     5'b110101: data_out_LUT <= 64'h3ca0000000000000;
377                     5'b110110: data_out_LUT <= 64'h3c90000000000000;
378                     5'b110111: data_out_LUT <= 64'h3c80000000000000;
379                     5'b111000: data_out_LUT <= 64'h3c70000000000000;
380                     5'b111001: data_out_LUT <= 64'h3c60000000000000;
381                     5'b111010: data_out_LUT <= 64'h3c50000000000000;
382                     5'b111011: data_out_LUT <= 64'h3c40000000000000;
383                     5'b111100: data_out_LUT <= 64'h3c30000000000000;
384                     5'b111101: data_out_LUT <= 64'h3c20000000000000;
385                     5'b111110: data_out_LUT <= 64'h3c10000000000000;
386                     5'b111111: data_out_LUT <= 64'h3c00000000000000;
387                     default:   data_out_LUT <= 64'h0000000000000000;
388             endcase
389         end
390 endgenerate*/
391
392
393 //Modulo de resta en punto fijo que le resta al exponente de x el valor de la
    iteracion actual, y con esto se realiza el desplazamiento en punto flotante.
394 Simple_Subt #(.W(E)) shift_x
395 (
396 .A(d_ff2_X[W-2:M]),
397 .B(cont_iter_out),
398 .Y(sh_exp_x)
399 );
400
401 ////Modulo de resta en punto fijo que le resta al exponente de y el valor de la
    iteracion actual, y con esto se realiza el desplazamiento en punto flotante.
402 Simple_Subt #(.W(E)) shift_y
403 (
404 .A(d_ff2_Y[W-2:M]),
405 .B(cont_iter_out),
406 .Y(sh_exp_y)
```

```
407   );
408
409   //FF_D que guarda el nuevo valor de x en punto flotante despues de realizarse el
      desplazamiento.
410   d_ff_en #(.W(W)) d_ff3_x_shift
411   (
412   .clk(clk),
413   .rst(rst_cordic),
414   .enable(enab_d_ff3_sh_exp_x),
415   .D({d_ff2_X[W-1],sh_exp_x,d_ff2_X[M-1:0]}),
416   .Q(d_ff3_sh_x_out));
417
418   ////FF_D que guarda el nuevo valor de y en punto flotante despues de realizarse el
      desplazamiento.
419   d_ff_en #(.W(W)) d_ff3_y_shift
420   (
421   .clk(clk),
422   .rst(rst_cordic),
423   .enable(enab_d_ff3_sh_exp_y),
424   .D({d_ff2_Y[W-1],sh_exp_y,d_ff2_Y[M-1:0]}),
425   .Q(d_ff3_sh_y_out)
426   );
427
428   //FF_D que guarda el valor obtenido de la LUT
429   d_ff_en #(.W(W)) d_ff3_LUT
430   (
431   .clk(clk),
432   .rst(rst_cordic),
433   .enable(enab_d_ff3_LUT),
434   .D(data_out_LUT),
435   .Q(d_ff3_LUT_out)
436   );
437
438   //FF_D que guarda el valor del signo de la variable Y o Z, dependiendo del modo del
      algoritmo CORDIC.
439   d_ff_en #(.W(1)) d_ff3_sign
440   (
441   .clk(clk),
442   .rst(rst_cordic),
443   .enable(enab_d_ff3_sign),
444   .D(sign),
445   .Q(d_ff3_sign_out)
446   );
447
448   ////////////////////////////////////////////////////////////////////////////////
      ////////////////////////////////////////////////////////////////////////////////
449   //Fourth Stage
450
451   //Mux que regula su salida dependiendo de la variable que se este calculando.
452   Mux_3x1 #(.W(W)) mux_3x1_var1
453   (
454   .select(sel_mux_2),
455   .ch_0(d_ff2_X),
```

```
456   .ch_1(d_ff2_Y),
457   .ch_2(d_ff2_Z),
458   .data_out(add_subt_dataA)
459   );
460
461   //Mux que regula su salida dependiendo de la variable que se este calculando.
462   Mux_3x1 #(.W(W)) mux_3x1_var2
463   (
464   .select(sel_mux_2),
465   .ch_0(d_ff3_sh_y_out),
466   .ch_1(d_ff3_sh_x_out),
467   .ch_2(d_ff3_LUT_out),
468   .data_out(add_subt_dataB)
469   );
470
471   //Modulo que decide dependiendo de sus entradas si la operacion que se debe realizar
      es una suma o resta.
472   Op_Select   op_select_mod
473   (
474   .variable(cont_var_out[0]),
475   .sign(d_ff3_sign_out),
476   .operation(op_add_subt)
477   );
478
479   ////////////////////////////////////////////////////////////////////////////////
      ////////////////////////////////////////////////////////////////////////////////
480   //Fifth Stage
481
482   //Registro que guarda el valor que proviene del modulo de suma/resta, y que es el
      valor de la variable X en dicha iteracion.
483   d_ff_en #(.W(W)) d_ff4_Xn
484   (
485   .clk(clk),
486   .rst(rst_cordic),
487   .enable(enab_d_ff4_Xn),
488   .D(result_add_subt),
489   .Q(d_ff_Xn)
490   );
491
492   //Registro que guarda el valor que proviene del modulo de suma/resta, y que es el
      valor de la variable Y en dicha iteracion.
493   d_ff_en #(.W(W)) d_ff4_Yn
494   (
495   .clk(clk),
496   .rst(rst_cordic),
497   .enable(enab_d_ff4_Yn),
498   .D(result_add_subt),
499   .Q(d_ff_Yn)
500   );
501
502   //Registro que guarda el valor que proviene del modulo de suma/resta, y que es el
      valor de la variable Z en dicha iteracion.
503   d_ff_en #(.W(W)) d_ff4_Zn
```

```
504   (
505   .clk(clk),
506   .rst(rst_cordic),
507   .enable(enab_d_ff4_Zn),
508   .D(result_add_subt),
509   .Q(d_ff_Zn)
510   );
511
512   ////////////////////////////////////////////////////////////////////////////////
      ////////////////////////////////////////////////////////////////////////////////
513   //Sixth Stage
514
515   // Mux de salida que controla cual valor se coloca en la salida, dependiendo de la
      operacion que se especifico al inicio.
516   Mux_2x1 #(.W(W)) mux_2x1_sal
517   (
518   .select(sel_mux_3),
519   .ch_0(d_ff_Xn),
520   .ch_1(d_ff_Yn),
521   .data_out(mux_sal)
522   );
523
524   //Registro que guarda el valor proveniente del mux ante explicado.
525   d_ff_en #(.W(W)) d_ff5
526   (
527   .clk(clk),
528   .rst(rst_cordic),
529   .enable(enab_d_ff5),
530   .D(mux_sal),
531   .Q(data_output2)
532   );
533
534   //Modulo que invierte el signo del resultado dependiendo de si el angulo de entrada
      estaba fuera del rango del calculo del algoritmo CORDIC.
535   sign_inverter #(.W(W)) sign_inverter_mod
536   (
537   .data(data_output2),
538   .shift_region_flag(d_ff1_shift_region_flag_out),
539   .operation(d_ff1_operation_out),
540   .data_out(sign_inv_out)
541   );
542
543   //Registro que guarda el valor de salida final, listo para enviarse al procesador.
544   d_ff_en #(.W(W)) d_ff5_data_out
545   (
546   .clk(clk),
547   .rst(rst_cordic),
548   .enable(enab_d_ff5_data_out),
549   .D(sign_inv_out),
550   .Q(data_output)
551   );
552
553   ////////////////////////////////////////////////////////////////////////////////
```

```
   ////////////////////////////////////////////////////////////////////////////////
554 //FSM and counters
555
556 //Contador que maneja cuantas iteraciones se deben realizar, activa una bandera
    cuando se alcanza la minima y maxima cuenta.
557 univ_bin_counter #(.N(5)) cont_iter
558 (
559 .clk(clk),
560 .reset(rst_cordic),
561 .syn_clr(syn_clr),
562 .load(load_con_iter),
563 .en(enab_cont_iter),
564 .up(up),
565 .d(d_iter),
566 .max_tick(max_tick_iter),
567 .min_tick(min_tick_iter),
568 .q(cont_iter_out)
569 );
570
571 //Contador que maneja cual variable se calcula, activa una bandera cuando se alcanza
    la minima y maxima cuenta.
572 univ_bin_counter #(.N(2)) cont_var
573 (
574 .clk(clk),
575 .reset(rst_cordic),
576 .syn_clr(syn_clr),
577 .load(load_con_var),
578 .en(enab_cont_var),
579 .up(up),
580 .d(d_var),
581 .max_tick(max_tick_var),
582 .min_tick(min_tick_var),
583 .q(cont_var_out)
584 );
585
586 //Maquina de estados que controla los procesos de enable, carga y controla los
    tiempos en que se activan cada etapa del calculo.
587 CORDIC_FSM fsm_cordic
588 (
589 .clk(clk),
590 .reset(rst_cordic),
591 .beg_FSM_CORDIC(beg_fsm_cordic),
592 .ACK_FSM_CORDIC(ack_cordic),
593 .operation(d_ff1_operation_out),
594 .shift_region_flag(d_ff1_shift_region_flag_out),
595 .cont_var(cont_var_out),
596 .ready_add_subt(ready_add_subt),
597 .max_tick_iter(max_tick_iter),
598 .min_tick_iter(min_tick_iter),
599 .max_tick_var(max_tick_var),
600 .min_tick_var(min_tick_var),
601
602 .ready_CORDIC(ready_cordic),
```

```
603   .beg_add_subt(beg_add_subt),
604   .ack_add_subt(ack_add_subt),
605   .sel_mux_1(sel_mux_1),
606   .sel_mux_3(sel_mux_3),
607   .sel_mux_2(sel_mux_2),
608   .mode(mode),
609   .enab_cont_iter(enab_cont_iter),
610   .load_cont_iter(load_con_iter),
611   .enab_cont_var(enab_cont_var),
612   .load_cont_var(load_cont_var),
613   .enab_RB1(enab_d_ff1_RB1),
614   .enab_RB2(enab_d_ff2_RB2),
615   .enab_d_ff_Xn(enab_d_ff4_Xn),
616   .enab_d_ff_Yn(enab_d_ff4_Yn),
617   .enab_d_ff_Zn(enab_d_ff4_Zn),
618   .enab_d_ff_out(enab_d_ff5_data_out),
619   .enab_dff5(enab_dff5),
620   .enab_dff_shifted_x(enab_d_ff3_sh_exp_x),
621   .enab_dff_shifted_y(enab_d_ff3_sh_exp_y),
622   .enab_dff_LUT(enab_d_ff3_LUT),
623   .enab_dff_sign(enab_d_ff3_sign)
624   );
625
626   endmodule
627
```