```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date:    22:06:01 08/27/2015
7  // Design Name:
8  // Module Name:    FPU_Add_Subtract_Function
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 module FPU_Add_Subtract_Function
22 //Add/Subtract Function Parameters
23
24    /*#(parameter W = 32, parameter EW = 8, parameter SW = 23,
25         parameter SWR=26, parameter EWR = 5)  //Single Precision */
26
27     #(parameter W = 64, parameter EW = 11, parameter SW = 52,
28         parameter SWR = 55, parameter EWR = 6) //-- Double Precision */
29     (
30         //FSM Signals
31         input wire clk,
32         input wire rst,
33         input wire beg_FSM,
34         input wire ack_FSM,
35
36         //Oper_Start_in signals
37         input wire [W-1:0] Data_X,
38         input wire [W-1:0] Data_Y,
39         input wire add_subt,
40
41         //Round signals signals
42         input wire [1:0] r_mode,
43
44         //OUTPUT SIGNALS
45         output wire overflow_flag,
46         output wire underflow_flag,
47         output wire ready,
48         output wire [W-1:0] final_result_ieee
49     );
50
51
52
53 ////////////op_start_in//////////////
```

```
54  wire FSM_op_start_in_load_a,FSM_op_start_in_load_b;
55  wire [W-2:0] DMP, DmP;
56  wire real_op;
57  wire sign_final_result;
58  ///////////Mux S-> exp_operation OPER_A_i/////////
59
60  wire FSM_selector_A;
61  //D0=DMP_o[W-2:W-EW-1]
62  //D1=exp_oper_result
63  wire [EW-1:0] S_Oper_A_exp;
64
65  ///////////Mux S-> exp_operation OPER_B_i/////////
66
67  wire [1:0] FSM_selector_B;
68  //D0=DmP_o[W-2:W-9/W-12]
69  //D1=LZA_output
70  wire [EW-1:0] S_Oper_B_exp;
71
72  ///////////exp_operation/////////////////////////
73  wire FSM_exp_operation_load_diff, FSM_exp_operation_load_OU ,FSM_exp_operation_A_S;
74  //oper_A= S_Oper_A_exp
75  //oper_B= S_Oper_B_exp
76  wire [EW-1:0] exp_oper_result;
77
78  ///////////Mux S-> Barrel shifter shift_Value//////
79
80  //ctrl = FSM_selector_B;
81  //D0=exp_oper_result
82  //D1=LZA_output
83  wire [EWR-1:0] S_Shift_Value;
84
85  ///////////Mux S-> Barrel shifter Data_in//////
86
87  wire FSM_selector_C;
88  //D0={1'b1,DmP [SW-1:0], 2'b0}
89  //D1={Add_Subt_Data_output}
90  wire [SWR-1:0]S_Data_Shift;
91
92  ///////////Barrel_Shifter/////////////////////////
93
94  wire FSM_barrel_shifter_load, FSM_barrel_shifter_L_R, FSM_barrel_shifter_B_S;
95  //Shift_Value=S_Shift_Value
96  //Data_in=S_Data_Shift
97  wire [SWR-1:0] Sgf_normalized_result;
98
99  /////////Mux S-> Add_Subt_Sgf op/////////////////
100 wire FSM_selector_D;
101 //D0=real_op
102 //D1= 1'b0
103 wire S_A_S_op;
104
105 /////////Mux S-> Add_Subt_Sgf OPER A/////////////////
106 //wire FSM_selector_D
```

```
107   //D0={1'b1, DMP[SW-1:0], 2'b00}
108   //D1= Norm_Shift_Data
109   wire [SWR-1:0] S_A_S_Oper_A;
110
111   /////////Mux S-> Add_Subt_Sgf OPER B/////////////////
112   //wire FSM_selector_D
113   //D0= Norm_Shift_Data
114   //D1= SWR'd1;
115   wire [SWR-1:0] S_A_S_Oper_B;
116
117   ////////ADD_Subt_sgf////////////////////////////////////
118
119
120   wire FSM_Add_Subt_Sgf_load, add_overflow_flag;
121   //Add_Subt_i=S_A_S_op
122   //Oper_A_i=S_A_S_Oper_A
123   //Oper_B_i=S_A_S_Oper_B
124   wire [SWR-1:0] Add_Subt_result;
125   wire [SWR-1:0] A_S_P;
126   wire [SWR-1:1] A_S_C;
127   //FSM_C_o=add_overflow_flag
128
129
130   /////////LZA/////////////////////////////////////////
131
132   wire FSM_LZA_load;
133   //P_i=A_S_P
134   //C_i=A_S_C
135   //A_S_op_i=S_A_S_op
136   wire [EWR-1:0] LZA_output;
137
138   ////////Deco_round//////////////////////////////////////
139
140   //Data_i=Sgf_normalized_result
141   //Round_Type=r_mode
142   //Sign_Result_i=sign_final_result
143   wire round_flag;
144
145   ///////Final_result/////////////////////////////////////
146
147   wire FSM_Final_Result_load;
148
149   ////////////////////////////////////////////////////////////////////////////////
150
151   wire rst_int;
152
153
154   ////////////////////////////////////////////////////////////////////////////////
155   wire selector_A;
156   wire [1:0] selector_B;
157   wire load_b;
158   wire selector_C;
159   wire selector_D;
```

```
160
161 ////////////////////////////////////FSM/////////////////////////////////////////
162
163 FSM_Add_Subtract FS_Module(
164     .clk(clk),                                                          //
165     .rst(rst),                                                          //
166     .rst_FSM(ack_FSM),                                                  //
167     .beg_FSM(beg_FSM),                                                  //
168     .zero_flag_i(zero_flag),                                            //
169     .norm_iteration_i(FSM_selector_C),                                  //
170     .add_overflow_i(add_overflow_flag),                                 //
171     .round_i(round_flag),                                               //
172     .load_1_o(FSM_op_start_in_load_a),                                  //
173     .load_2_o(FSM_op_start_in_load_b),                                  //
174     .load_3_o(FSM_exp_operation_load_diff),                                  //
175     .load_8_o(FSM_exp_operation_load_OU),
176     .A_S_op_o(FSM_exp_operation_A_S),                                   //
177     .load_4_o(FSM_barrel_shifter_load),                                 //
178     .left_right_o(FSM_barrel_shifter_L_R),                              //
179     .bit_shift_o(FSM_barrel_shifter_B_S),                               //
180     .load_5_o(FSM_Add_Subt_Sgf_load),                                     //
181     .load_6_o(FSM_LZA_load),                                              //
182     .load_7_o(FSM_Final_Result_load),                                     //
183     .ctrl_a_o(selector_A),                                            //
184     .ctrl_b_o(selector_B),                                            //
185     .ctrl_b_load_o(load_b),
186     .ctrl_c_o(selector_C),                                            //
187     .ctrl_d_o(selector_D),                                            //
188     .rst_int(rst_int),                                                  //
189     .ready(ready)                                                       //
190     );
191
192
193
194
195 ////////////////////////////Selector's registers//////////////////////////////
196
197 RegisterAdd #(.W(1)) Sel_A ( //Selector_A register
198     .clk(clk),
199     .rst(rst_int),
200     .load(selector_A),
201     .D(1'b1),
202     .Q(FSM_selector_A)
203     );
204
205 RegisterAdd #(.W(1)) Sel_C ( //Selector_C register
206     .clk(clk),
207     .rst(rst_int),
208     .load(selector_C),
209     .D(1'b1),
210     .Q(FSM_selector_C)
211     );
212
```

```
213  RegisterAdd #(.W(1)) Sel_D ( //Selector_D register
214          .clk(clk),
215          .rst(rst_int),
216          .load(selector_D),
217          .D(1'b1),
218          .Q(FSM_selector_D)
219          );
220
221  RegisterAdd #(.W(2)) Sel_B ( //Selector_B register
222                  .clk(clk),
223                  .rst(rst_int),
224                  .load(load_b),
225                  .D(selector_B),
226                  .Q(FSM_selector_B)
227                  );
228
229  ////////////////////////////////////////////////////
230
231  //MODULES////////////////////////////
232
233  ////////////////////Oper_Start_in/////////////////7
234
235  //This Module classify both operands
236  //in bigger and smaller magnitude, Calculate the result' Sign bit and calculate the real
237  //operation for the execution//////////////////////////////
238
239  Oper_Start_In #(.W(W)) Oper_Start_in_module (
240      .clk(clk),
241      .rst(rst_int),
242      .load_a_i(FSM_op_start_in_load_a),
243      .load_b_i(FSM_op_start_in_load_b),
244      .add_subt_i(add_subt),
245      .Data_X_i(Data_X),
246      .Data_Y_i(Data_Y),
247      .DMP_o(DMP),
248      .DmP_o(DmP),
249      .zero_flag_o(zero_flag),
250      .real_op_o(real_op),
251      .sign_final_result_o(sign_final_result)
252      );
253
254  ////////////////////////////////////////////////////
255
256
257  ///////////Mux exp_operation OPER_A_i/////////
258
259  Multiplexer_AC #(.W(EW)) Exp_Oper_A_mux(
260          .ctrl(FSM_selector_A),
261          .D0 (DMP[W-2:W-EW-1]),
262          .D1 (exp_oper_result),
263          .S (S_Oper_A_exp)
264      );
```

```
265
266 ///////////Mux exp_operation OPER_B_i//////////
267 wire [EW-EWR-1:0] Exp_oper_B_D1;
268 wire [EW-1:0] Exp_oper_B_D2;
269
270 Mux_3x1 #(.W(EW)) Exp_Oper_B_mux(
271                 .ctrl(FSM_selector_B),
272                 .D0 (DmP[W-2:W-EW-1]),
273                 .D1 ({Exp_oper_B_D1,LZA_output}),
274                 .D2 (Exp_oper_B_D2),
275                 .S(S_Oper_B_exp)
276             );
277
278
279 generate
280     case(EW)
281         8:begin
282             assign Exp_oper_B_D1 =3'd0;
283             assign Exp_oper_B_D2 = 8'd1;
284         end
285         default:begin
286             assign Exp_oper_B_D1 =5'd0;
287              assign Exp_oper_B_D2 = 11'd1;
288         end
289     endcase
290 endgenerate
291
292 ///////////exp_operation//////////////////////////
293
294 Exp_Operation #(.EW(EW)) Exp_Operation_Module(
295     .clk(clk),
296     .rst(rst_int),
297     .load_a_i(FSM_exp_operation_load_diff),
298     .load_b_i(FSM_exp_operation_load_OU),
299     .Data_A_i(S_Oper_A_exp),
300     .Data_B_i(S_Oper_B_exp),
301     .Add_Subt_i(FSM_exp_operation_A_S),
302     .Data_Result_o(exp_oper_result),
303     .Overflow_flag_o(overflow_flag),
304     .Underflow_flag_o(underflow_flag)
305     );
306
307
308 ///////////Mux Barrel shifter shift_Value////////////////
309
310 wire [EWR-1:0] Barrel_Shifter_S_V_D2;
311
312 Mux_3x1 #(.W(EWR)) Barrel_Shifter_S_V_mux(
313                 .ctrl(FSM_selector_B),
314                 .D0 (exp_oper_result[EWR-1:0]),
315                 .D1 (LZA_output),
316                 .D2 (Barrel_Shifter_S_V_D2),
317                 .S  (S_Shift_Value)
```

```
318             );
319 
320 generate
321     case(EW)
322         8:begin
323             assign Barrel_Shifter_S_V_D2 = 5'd1;
324         end
325         default:begin
326             assign Barrel_Shifter_S_V_D2 = 6'd1;
327         end
328     endcase
329 endgenerate
330 
331 ////////////Mux Barrel shifter Data_in//////
332 
333 Multiplexer_AC #(.W(SWR)) Barrel_Shifter_D_I_mux(
334     .ctrl(FSM_selector_C),
335     .D0 ({1'b1,DmP[SW-1:0],2'b00}),
336     .D1 (Add_Subt_result),
337     .S (S_Data_Shift)
338     );
339 
340 ////////////Barrel_Shifter//////////////////////////
341 
342 Barrel_Shifter #(.SWR(SWR),.EWR(EWR)) Barrel_Shifter_module (
343     .clk(clk),
344     .rst(rst_int),
345     .load_i(FSM_barrel_shifter_load),
346     .Shift_Value_i(S_Shift_Value),
347     .Shift_Data_i(S_Data_Shift),
348     .Left_Right_i(FSM_barrel_shifter_L_R),
349     .Bit_Shift_i(FSM_barrel_shifter_B_S),
350     .N_mant_o(Sgf_normalized_result)
351     );
352 
353 
354 ////////////Mux Add_Subt_Sgf op//////////////////
355 
356 Multiplexer_AC #(.W(1)) Add_Sub_Sgf_op_mux(
357     .ctrl(FSM_selector_D),
358     .D0 (real_op),
359     .D1 (1'b0),
360     .S (S_A_S_op)
361     );
362 
363 ////////////Mux Add_Subt_Sgf oper A//////////////////
364 
365 Multiplexer_AC #(.W(SWR)) Add_Sub_Sgf_Oper_A_mux(
366     .ctrl(FSM_selector_D),
367     .D0 ({1'b1,DMP[SW-1:0],2'b00}),
368     .D1 (Sgf_normalized_result),
369     .S (S_A_S_Oper_A)
370     );
```

```
371
372 //////////Mux Add_Subt_Sgf oper B/////////////////
373
374 wire [SWR-1:0] Add_Sub_Sgf_Oper_A_D1;
375
376 Multiplexer_AC #(.W(SWR)) Add_Sub_Sgf_Oper_B_mux(
377                 .ctrl(FSM_selector_D),
378                 .D0 (Sgf_normalized_result),
379                 .D1 (Add_Sub_Sgf_Oper_A_D1),
380                 .S (S_A_S_Oper_B)
381                 );
382 generate
383     case (W)
384         32:begin
385             assign Add_Sub_Sgf_Oper_A_D1 = 26'd4;
386             end
387         default:begin
388              assign Add_Sub_Sgf_Oper_A_D1 =55'd4;
389             end
390        endcase
391 endgenerate
392
393 /////////ADD_Subt_sgf///////////////////////////////////
394
395 Add_Subt #(.SWR(SWR)) Add_Subt_Sgf_module(
396     .clk(clk),
397     .rst(rst_int),
398     .load_i(FSM_Add_Subt_Sgf_load),
399     .Add_Sub_op_i(S_A_S_op),
400     .Data_A_i(S_A_S_Oper_A),
401     .PreData_B_i(S_A_S_Oper_B),
402     .Data_Result_o(Add_Subt_result),
403     //.P_o(A_S_P),
404     //.Cn_o(A_S_C),
405     .FSM_C_o(add_overflow_flag)
406     );
407
408 /*
409 //Test Comb LZA//
410
411 Test_comb_LZA #(.SWR(SWR)) comb(
412         .clk(clk),
413         .rst(rst),
414         .Op_A_i(S_A_S_Oper_A),
415         .Pre_Op_B_i(S_A_S_Oper_B),
416         .op(S_A_S_op), //Carry in
417         .Cn_o(A_S_C),
418         .P_o(A_S_P) //Propagate (for LZA)
419     );
420
421
422 //////////LZA/////////////////////////////////////////
423
```

```
424   LZA #(.SWR(SWR),.EWR(EWR)) Leading_Zero_Anticipator_Module (
425       .clk(clk),
426       .rst(rst_int),
427       .load_i(FSM_LZA_load),
428       .P_i(A_S_P),
429       .C_i(A_S_C),
430       .A_S_op_i(S_A_S_op),
431       .Shift_Value_o(LZA_output)
432       );
433   */
434   wire [SWR-1:0] Add_Subt_LZD;
435   assign Add_Subt_LZD = ~Add_Subt_result;
436
437   LZD #(.SWR(SWR),.EWR(EWR)) Leading_Zero_Detector_Module (
438       .clk(clk),
439       .rst(rst_int),
440       .load_i(FSM_LZA_load),
441       .Add_subt_result_i(Add_Subt_LZD),
442       .Shift_Value_o(LZA_output)
443       );
444
445   ////////Deco_round//////////////////////////////////////
446
447   Round_Sgf_Dec Rounding_Decoder(
448       .clk(clk),
449       .Data_i(Sgf_normalized_result[1:0]),
450       .Round_Type_i(r_mode),
451       .Sign_Result_i(sign_final_result),
452       .Round_Flag_o(round_flag)
453       );
454
455   ////////Final_result//////////////////////////////////////
456
457   Tenth_Phase #(.W(W),.EW(EW),.SW(SW)) final_result_ieee_Module(
458       .clk(clk),
459       .rst(rst_int),
460       .load_i(FSM_Final_Result_load),
461       .sel_a_i(overflow_flag),
462       .sel_b_i(underflow_flag),
463       .sign_i(sign_final_result),
464       .exp_ieee_i(exp_oper_result),
465       .sgf_ieee_i(Sgf_normalized_result[SWR-2:2]),
466       .final_result_ieee_o(final_result_ieee)
467       );
468
469   endmodule
470
```