

```
1  `timescale 1ns / 1ps
2  ///////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  //
6  // Create Date: 08/28/2016 01:49:26 AM
7  // Design Name:
8  // Module Name: KOA_2
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 KOA_2    //(parameter SW = 24)
24 // #(parameter SW = 54)
25 #(parameter SW = 12)
26 (
27   input wire clk,
28   input wire rst,
29   input wire load_b_i,
30   input wire [SW-1:0] Data_A_i,
31   input wire [SW-1:0] Data_B_i,
32   output wire [2*SW-1:0] sgf_result_o
33 );
34
35   //wire [SW-1:0] Data_A_i;
36   //wire [SW-1:0] Data_B_i;
37
38
39   //wire [2*(SW/2)-1:0] result_left_mult;
40   //wire [2*(SW/2+1)-1:0] result_right_mult;
41   wire [SW/2+1:0] result_A_adder;
42   //wire [SW/2+1:0] Q_result_A_adder;
43   wire [SW/2+1:0] result_B_adder;
44   //wire [SW/2+1:0] Q_result_B_adder;
45   //wire [2*(SW/2+2)-1:0] result_middle_mult;
46
47   wire [SW-1:0] Q_left;
48   wire [SW+1:0] Q_right;
49   wire [SW+3:0] Q_middle; ///Modificación J: Le he agregado dos bits al largo del puerto, para acomodar los 2 cero el resultado de una multiplicacion
50
51   wire [2*(SW/2+2)-1:0] S_A;
52   wire [2*(SW/2+2)-1:0] S_B;
```

```

53
54     wire [4*(SW/2)+2:0] Result;
55     //////////////////////////////////////////////////
56     wire [1:0] zero1;
57     wire [3:0] zero2;
58     assign zero1 =2'b00;
59     assign zero2 =4'b0000;
60     //////////////////////////////////////////////////
61     wire [SW/2-1:0] rightside1;
62     wire [SW/2-3:0] leftside1;
63     wire [SW/2:0] rightside2;
64
65     wire [4*(SW/2)-1:0] sgf_r;
66
67     assign rightside1 = (SW/2) *1'b0;
68     assign rightside2 = (SW/2+1)*1'b0;
69     assign leftside1 = (SW/2-2) *1'b0;
70
71     localparam half = SW/2;
72     localparam full_port = SW - 1;
73     //localparam level1=4;
74     //localparam level2=5;
75
76     //////////////////////////////////////////////////
77     //generate
78     //     case (SW%2)
79     //         0:begin
80             //////////////////////////////////////////////////even////////////////////////////////////
81             //Multiplier for left side and right side
82
83             KOA_1 #(.SW(SW/2)*,.level(level1)*/) left(
84                 .Data_A_i(Data_A_i[full_port:half]/*P=SW/2*/),
85                 .Data_B_i(Data_B_i[full_port:half]/*P=SW/2*/),
86                 .sgf_result_o(Q_left) /*P=SW*//*result_left_mult*/
87             );
88
89
90             KOA_1 #(.SW(SW/2)) right(                                     /*,.level(level1)*/
91                 .Data_A_i(Data_A_i[half-1:0]/*P=SW/2*/),
92                 .Data_B_i(Data_B_i[half-1:0]/*P=SW/2*/),
93                 .sgf_result_o(Q_right[full_port:0]/*P=SW*/)
94                 /*result_right_mult[2*(SW/2)-1:0]*/
95             );
96
97             //Adders for middle
98
99             adder #(.W(SW/2)) A_operation (
100                 .Data_A_i(Data_A_i[SW-1:SW-SW/2]/*P=SW/2*/),
101                 .Data_B_i(Data_A_i[SW-SW/2-1:0]/*P=SW/2*/),
102                 .Data_S_o(result_A_adder[SW/2:0]/*P=SW/2+1*/)
103             );
104
105             adder #(.W(SW/2)) B_operation (

```

```

105         .Data_A_i(Data_B_i[SW-1:SW/2]/*P=SW/2*/),
106         .Data_B_i(Data_B_i[SW/2-1:0]/*P=SW/2*/),
107         .Data_S_o(result_B_adder[SW/2:0]/*P=SW/2+1*/)
108     );
109
110     //multiplication for middle
111     //Introducimos un par de ceros, ya que esta multiplicacion es siempre impar,
112     //Modificación: Le agregué otro bit al puerto de este multiplicador, para
113     KOA_1 #(.SW(SW/2+2)/* Port length = SW/2 + 2*/) middle (
114
115         .Data_A_i({1'b0 /*P=1*/, result_A_adder[SW/2:0]/*P=SW/2+1**/),
116         /*Q_result_A_adder[SW/2+1:0]*/
117         .Data_B_i({1'b0 /*P=1*/, result_B_adder[SW/2:0]/*P=SW/2+1**/),
118         /*Q_result_B_adder[SW/2+1:0]*/
119         .sgf_result_o(Q_middle[SW+3:0])
120         /*result_middle_mult[2*(SW/2)+2:0]*/
121
122         //Vamos a truncar este resultado en la siguiente etapa del puerto
123     );
124
125     // wire [SW-1:0] Q_left;
126     // wire [SW+1:0] Q_right;
127     // wire [SW+3:0] Q_middle;
128
129     ///Subtractors for middle
130
131     subtractor #(.W(SW+2)) Subtr_1 (
132         .Data_A_i(Q_middle[SW+1:0] /*P=SW+2*/),
133         /*result_middle_mult*/
134         .Data_B_i({zero1/*P=2*/, Q_left /*P=SW**/),
135         /*result_left_mult*/
136         .Data_S_o(S_A[SW+1:0]/*P=SW+2*/)
137     );
138
139     subtractor #(.W(SW+2)) Subtr_2 (
140         .Data_A_i(S_A[2*(SW/2)+1:0] /*P=SW+2*/),
141         .Data_B_i({zero1 /*P=2*/, Q_right[SW-1:0] /*P=SW**/),
142         /*result_right_mult*/
143         .Data_S_o(S_B[2*(SW/2)+1:0]) //Port width is SW+1
144     );
145
146     //wire [SW-1:0] Q_left; /*P=SW*/
147     //wire [SW+1:0] Q_right; /*P=SW+2*/
148
149     //Final adder
150     adder #(.W(2*SW)) Final(
151         .Data_A_i({Q_left /*P=SW*/, Q_right[SW-1:0]/*P=SW**/), //Port
152         width is 2*SW /*result_left_mult,result_right_mult*/

```

```

149         .Data_B_i({leftside1,S_B[2*(SW/2)+1:0],rightside1}),
150         .Data_S_o(Result[4*(SW/2):0]/*P=2*SW+1*/);
151     );
152
153     //Final Register
154
155     RegisterAdd #(.W(2*SW)) finalreg ( //Data X input register
156         .clk(clk),
157         .rst(rst),
158         .load(load_b_i),
159         .D(Result[4*(SW/2)-1:0]),
160         .Q({sgf_result_o})
161     );
162
163     //      end
164     //      1:begin
165     //      //////////////////////////////////odd////////////////////////////////////
166     //      //Multiplier for left side and right side
167
168     //      KOA_1 #(.SW(SW/2)/*,.level(level2)*/) left_high(
169
170     //          .Data_A_i(Data_A_i[SW-1:SW/2]),
171     //          .Data_B_i(Data_B_i[SW-1:SW/2]),
172     //          .sgf_result_o(/*result_left_mult*/Q_left)
173     //      );
174
175     //      /*RegisterAdd #(.W(2*(SW/2))) leftreg( //Data X input register
176     //          .clk(clk),
177     //          .rst(rst),
178     //          .load(1'b1),
179     //          .D(result_left_mult),
180     //          .Q(Q_left)
181     //      );//*/
182
183     //      KOA_1 #(.SW((SW/2)+1)/*,.level(level2)*/) right_lower(
184
185     //          .Data_A_i(Data_A_i[SW/2-1:0]),
186     //          .Data_B_i(Data_B_i[SW/2-1:0]),
187     //          .sgf_result_o(/*result_right_mult*/Q_right)
188     //      );
189
190     //      /*RegisterAdd #(.W(2*((SW/2)+1))) rightreg( //Data X input register
191     //          .clk(clk),
192     //          .rst(rst),
193     //          .load(1'b1),
194     //          .D(result_right_mult),
195     //          .Q(Q_right)
196     //      );//*/
197
198     //      //Adders for middle
199
200     //      adder #(.W(SW/2+1)) A_operation (
201     //          .Data_A_i({1'b0,Data_A_i[SW-1:SW-SW/2]}),

```

```

202 //      .Data_B_i(Data_A_i[SW-SW/2-1:0]),
203 //      .Data_S_o(result_A_adder)
204 //  );
205
206 //      adder #(.W(SW/2+1)) B_operation (
207 //          .Data_A_i({1'b0,Data_B_i[SW-1:SW-SW/2]}),
208 //          .Data_B_i(Data_B_i[SW-SW/2-1:0]),
209 //          .Data_S_o(result_B_adder)
210 //  );
211
212 //      //segmentation registers for 64 bits
213
214 //      /*RegisterAdd #(.W(SW/2+2)) preAreg ( //Data X input register
215 //          .clk(clk),
216 //          .rst(rst),
217 //          .load(1'b1),
218 //          .D(result_A_adder),
219 //          .Q(Q_result_A_adder)
220 //      );//*/
221
222 //      RegisterAdd #(.W(SW/2+2)) preBreg ( //Data X input register
223 //          .clk(clk),
224 //          .rst(rst),
225 //          .load(1'b1),
226 //          .D(result_B_adder),
227 //          .Q(Q_result_B_adder)
228 //      );//*/
229 //      //multiplication for middle
230
231 //      K0A_1 #(.SW(SW/2+2)/*,.level(level2)*/) middle (
232
233 //          .Data_A_i(/*Q_result_A_adder*/result_A_adder),
234 //          .Data_B_i(/*Q_result_B_adder*/result_B_adder),
235 //          .sgf_result_o(/*result_middle_mult*/Q_middle)
236 //      );
237
238 //      //segmentation registers array
239
240
241 //      /*RegisterAdd #(.W(2*((SW/2)+2))) midreg ( //Data X input register
242 //          .clk(clk),
243 //          .rst(rst),
244 //          .load(1'b1),
245 //          .D(result_middle_mult),
246 //          .Q(Q_middle)
247 //      );//*/
248
249 //      ///Subtractors for middle
250
251 //      subtractor #(.W(2*(SW/2+2))) Subtr_1 (
252 //          .Data_A_i(/*result_middle_mult*//*Q_middle),
253 //          .Data_B_i({zero2, /*result_left_mult*//*Q_left}),
254 //          .Data_S_o(S_A)

```

```
255 // );
256
257 // subtractor #(.W(2*(SW/2+2))) Subtr_2 (
258 //     .Data_A_i(S_A),
259 //     .Data_B_i({zero1, /*result_right_mult/*/*Q_right}),
260 //     .Data_S_o(S_B)
261 // );
262
263 // //Final adder
264
265 // adder #(.W(4*(SW/2)+2)) Final(
266 //     .Data_A_i({/*result_left_mult,result_right_mult*/Q_left,Q_right}),
267 //     .Data_B_i({S_B,rightside2}),
268 //     .Data_S_o(Result[4*(SW/2)+2:0])
269 // );
270
271 // //Final Register
272
273
274
275 // RegisterAdd #(.W(4*(SW/2)+2)) finalreg ( //Data X input register
276 //     .clk(clk),
277 //     .rst(rst),
278 //     .load(load_b_i),
279 //     .D(Result[2*SW-1:0]),
280 //     .Q({sgf_result_o})
281 // );
282 //     end
283 // endcase
284 //endgenerate
285
286
287
288
289 endmodule
290
```