```
// ee417 lesson 7 Assignment 1 L7A1
// Name: Ron Kalin, Date: 06-25-24  Group: Kalin/Jammeh
// Design: Sequential_multiplier
// top level module raises a ready flag when ready to load new input words
// user should activate a start input to indicate a new multiplication operation starts
module Sequential_multiplier  (product, final_product,
                               Ready, start,
                               word1, word2,
                               clk, rst);
parameter                   L_WORD =4;  //Datapathsize
output    [2*L_WORD-1: 0] product, final_product;
output                    Ready;
input     [L_WORD -1: 0]  word1, word2;
input                     start, clk, rst;

reg [L_WORD-1:0] mcand, mult;

wire multiplier_LSB, Load_words, shift, Add, latch, zero_flag;
wire [L_WORD-2:0] state;

Datapath M1   (product, final_product,
               Ready, multiplier_LSB, zero_flag,
               word1, word2,
               Load_words, shift, Add, latch,
               clk, rst);

controller M2 (Load_words, shift, Add, latch, state,
               Ready, multiplier_LSB, start, zero_flag,
               clk, rst);

endmodule

//Datapath
module Datapath (product, final_product,
                 Ready, multiplier_LSB, zero_flag,
                 word1, word2,
                 Load_words, shift, Add, latch,
                 clk, rst);

parameter L_WORD= 4;  //declare parameter values

//declare outputs and input
output reg [2*L_WORD-1:0] product, final_product;
output reg                Ready;
output                    multiplier_LSB, zero_flag;
input      [ L_WORD-1:0]  word1, word2;
input                     Load_words, shift, Add, latch;
input                     clk, rst;
//declare internal wires
reg    [2*L_WORD-1: 0]    multiplicand;
reg    [  L_WORD-1: 0]    multiplier;
//assign values
assign multiplier_LSB = multiplier[0];    //least significant bit of multiplier
assign zero_flag      = (multiplier == 0);//if multiplier=all zeros, zero_flag=1

//create always block
always @ (posedge clk)
 begin
       if (rst)        begin multiplier    <= 0;//if reset =1 then zero
                             multiplicand  <= 0;
                             product       <= 0;
                             final_product <= 0;
                             Ready         <= 1; end//ready high= accept input words
   else if (Load_words) begin multiplicand  <= word1;//Load_words=1
                             multiplier    <= word2;//then m..cand gets value of word1
                             product       <= 0;    //mult gets value word2
                             final_product <= 0;    //prod and final_prod=0
                             Ready         <= 0; end//ready low means calculate
   else if (shift)      begin multiplier    <= multiplier >> 1;   //shift right 1
                             multiplicand  <= multiplicand << 1; //shift left 1
```

```
71                                  Ready          <= 0; end
72      else if (Add)         begin product        <= product + multiplicand;
73                                  Ready          <= 0; end
74      else if (latch)       begin final_product  <= product; //adding done
75                                  Ready          <= 1; end   //ready for new input
76      else                  begin end
77     end
78
79    endmodule
80
81    //controller
82    module controller (Load_words, shift, Add, latch, state,
83                Ready, multiplier_LSB, start, zero_flag,
84                clk, rst);
85    //parameter L_WORD= 4;  //declare parameter values
86
87    //declare outputs/inputs, control unit only handles single bit inputs and outputs
88    output reg Load_words, shift, Add, latch;
89    output reg [2:0] state;
90    input      Ready, multiplier_LSB, start, zero_flag;
91    input      clk, rst;
92
93    reg [2:0]  next_state; //3 bits for up to 8 states
94    //build code from FSM diagram
95
96    //declare states from FSM
97    parameter idle    = 3'b000;
98    parameter loading = 3'b001;
99    parameter loaded  = 3'b010;
100   parameter add     = 3'b011;
101   parameter shft    = 3'b100;
102   parameter buff    = 3'b101;
103   parameter ltch    = 3'b110;
104
105   always @ (posedge clk)
106   if (rst) state <= idle;
107      else  state <= next_state;
108
109   always @ *
110     //assign probe <= state;
111     case (state)
112       idle: begin
113               Load_words = 1'b0;
114               latch      = 1'b0;
115               Add        = 1'b0;
116               shift      = 1'b0;
117               if (Ready && start) next_state = loading;
118                           else    next_state = idle;    end
119       loading: begin
120               Load_words = 1'b1;
121               latch      = 1'b0;
122               Add        = 1'b0;
123               shift      = 1'b0;
124               next_state = loaded;   end
125       loaded: begin  //2nd load stage is needed because input data changes
126               Load_words = 1'b0;  //1 cycle is needed to look at the change
127               latch      = 1'b0;
128               Add        = 1'b0;
129               shift      = 1'b0;
130               if (multiplier_LSB) next_state = add;
131                           else    next_state = shft;    end
132       add: begin
133               Load_words = 1'b0;
134               latch      = 1'b0;
135               Add        = 1'b1;    //output changes
136               shift      = 1'b0;
137               next_state = shft;     end  //shift always follows after an add
138       shft: begin
139               Load_words = 1'b0;
140               latch      = 1'b0;
```

```
141                 Add        = 1'b0;
142                 shift      = 1'b1;
143                 next_state = buff;     end
144         buff: begin //buffer state needed after shift because input data changes
145                 Load_words = 1'b0; //1 cycle is needed to take a look at the change
146                 latch      = 1'b0;
147                 Add        = 1'b0;
148                 shift      = 1'b0;
149                 if (multiplier_LSB) next_state = add;
150                 else if (zero_flag) next_state = ltch;
151                                     else   next_state = shft;
    end
152         ltch: begin
153                 Load_words = 1'b0;
154                 Add        = 1'b0;
155                 shift      = 1'b0;
156                 latch      = 1'b1;
157                 next_state = loading;    end  //cycle back to loading stage
158
159   //below is initial pseudocode
160   //if rst=1 then load_words=0, shift=add=latch=0
161   //if rst=0 and start=1 and ready=1 then load_words=1, shift=0, add=0, latch=0
162   //if rst=0 and start=1 and ready=0 then
163     //if multiplier_LSB=0 then load_words=0, shift=1, add=0, latch=0
164     //
165     //else if multiplier_LSB=1 then load_words=0, shift=0, add=1, latch=0
166     //
167     //else if zero_flag=1      then load_words=0, shift=0, Add=0, latch=1, ready=1 ...done
168     endcase
169   endmodule
170
171
```