```
1   `timescale 1ns / 1ps
2   /**********************************************************************
3    * File Name: transmitEngine.v
4    * Project: Tx Machine
5    * Designer: Marc Dominic Cabote
6    * Email: marcdominic011@gmail.com
7    * Rev. Date: 10 March, 2018
8    *
9    * Purpose:The transmit engine has the logic needed to communicate with a
10   *         computer using UART. For the terminal to have a stable connection
11   *         in order to output the proper characters, the program—Realterm—
12   *         should know what the setting is for the transmit engine—it should
13   *         know the baud rate, odd or even parity and number of bits.
14   *         If the program settings match the FPGA, it will then show
15   *         "CSULB CECS 460 - [LINECOUNTER] <CR> <LF>".
16   *
17   * Notes:   -  This module has a synchronous reset input.
18   *
19   **********************************************************************/
20  module transmitEngine(input clk, rst, eight, pen, ohel, load,
21                        input [7:0] out_port,
22                        input [18:0] k,
23                        output reg TxRdy, Tx);
24
25      wire ep, op, btu, done;
26
27      reg doit, load_d1, bit10, bit9;
28      reg [7:0] ldata;
29      reg [3:0] bit_count, bit_counter;
30      reg [10:0] shift_out;
31      reg [18:0] bit_time_count, bit_time_counter;
32
33
34      //=================================================================
35      // TxRdy RS Flop
36      //=================================================================
37      always @(posedge clk, posedge rst)
38         begin
39           if(rst)
40              TxRdy <= 1; else // high at reset
41           if(done==1 && load==1)
42              TxRdy <= TxRdy; else
43           if(done)
44              TxRdy <= 1; else
45           if(load)
46              TxRdy <= 0;
47           else
48              TxRdy <= TxRdy;
49         end
50
51      //=================================================================
52      // Doit RS Flop
53      //=================================================================
54
55      always @(posedge clk, posedge rst)
56          begin
57            if(rst)
```

```
58                doit <= 0; else
59             if(done==1 && load_d1==1)
60                doit <= doit; else
61             if(done)
62                doit <= 0; else
63             if(load_d1)
64                doit <= 1;
65             else
66                doit <= doit;
67          end
68
69       //=====================================================================
70       // 8-bit Loadable Register
71       //=====================================================================
72       always @(posedge clk, posedge rst)
73          begin
74             if(rst)
75                ldata <= 0;
76             else
77                ldata <= out_port;
78          end
79
80       //=====================================================================
81       //Bit Time Counter
82       //=====================================================================
83       assign btu = (bit_time_count == k);
84
85       always @(posedge clk, posedge rst)
86          begin
87             if(rst)
88                bit_time_count <= 0;
89             else
90                bit_time_count <= bit_time_counter;
91          end
92
93       always @(*)
94          begin
95             case ({doit,btu})
96                0: bit_time_counter = 0;
97                1: bit_time_counter = 0;
98                2: bit_time_counter = bit_time_count + 1;
99                3: bit_time_counter = 0;
100            endcase
101         end
102
103      //=====================================================================
104      //Bit Counter
105      //=====================================================================
106      assign done = (bit_count == 11);
107
108      always @(posedge clk, posedge rst)
109         begin
110            if(rst)
111               bit_count <= 0;
112            else
113               bit_count <= bit_counter;
114         end
```

```
115
116      always @(*)
117         begin
118            case ({doit,btu})
119               0: bit_counter = 0;
120               1: bit_counter = 0;
121               2: bit_counter = bit_count;
122               3: bit_counter = bit_count + 1;
123            endcase
124         end
125
126      //=================================================================
127      // Parity Decoder
128      //=================================================================
129      assign ep = eight ? (^ldata) : ^(ldata[6:0]);
130      assign op = eight ? !(^ldata): !(^(ldata[6:0]));
131
132      always @*
133       begin
134         case ({eight,pen,ohel})
135            0: {bit10,bit9} = {1'b1, 1'b1};
136            1: {bit10,bit9} = {1'b1, 1'b1};
137            2: {bit10,bit9} = {1'b1, ep};
138            3: {bit10,bit9} = {1'b1, op};
139            4: {bit10,bit9} = {1'b1, ldata[7]};
140            5: {bit10,bit9} = {1'b1, ldata[7]};
141            6: {bit10,bit9} = {ep, ldata[7]};
142            7: {bit10,bit9} = {op, ldata[7]};
143            default: {bit10,bit9} = {1'b1,1'b1};
144         endcase
145       end
146
147      //=================================================================
148      // LoadD1 D Flop
149      //=================================================================
150      always @(posedge clk, posedge rst)
151         begin
152            if(rst)
153               load_d1 <= 0;
154            else
155               load_d1 <= load;
156         end
157
158      //=================================================================
159      // Shift Register
160      //=================================================================
161      always @(posedge clk, posedge rst)
162         begin
163            if(rst)
164               shift_out <= 11'b11111111111; else
165            if(load_d1)
166               shift_out <= {bit10, bit9, ldata[6:0], 1'b0, 1'b1}; else
167            if(btu)
168               shift_out <= {1'b1, shift_out[10:1]};
169            else
170               shift_out  <= shift_out;
171         end
```

```
172
173       always @(*)begin
174         Tx = shift_out[0];
175       end
176
177
178   endmodule
179
```