

```
1  `timescale 1ns / 1ps
2  /*****
3   * File Name: transmitEngine.v
4   * Project: Tx Machine
5   * Designer: Marc 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 an asynchronous reset input.
18  *
19  *****/
20  module transmitEngine(input clk, rst, eight, pen, ohel, load,
21                       input [7:0] out_port,
22                       input [3:0] baud,
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, k;
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         //=====
82         // Baud Decoder
83         //=====
84         always @(*) begin
85             case (baud)
86                 4'b0000: k = 333333; //300
87                 4'b0001: k = 83333;  //1200
88                 4'b0010: k = 41667;  //2400
89                 4'b0011: k = 20833;  //4800
90                 4'b0100: k = 10417;  //9600
91                 4'b0101: k = 5208;   //19200
92                 4'b0110: k = 2604;   //38400
93                 4'b0111: k = 1736;   //57600
94                 4'b1000: k = 868;    //115200
95                 4'b1001: k = 434;    //230400
96                 4'b1010: k = 217;    //460800
97                 4'b1011: k = 109;    //921600
98                 default: k = 333333;
99             endcase
100         end
101
102         //=====
103         //Bit Time Counter
104         //=====
105         assign btu = (bit_time_count == k);
106
107         always @(posedge clk, posedge rst)begin
108             if(rst)
109                 bit_time_count <= 0;
110             else
111                 bit_time_count <= bit_time_counter;
112         end
113
114         always @(*)begin
```

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

```
172         else
173             load_d1 <= load;
174         end
175
176         //=====
177         // Shift Register
178         //=====
179         always @(posedge clk, posedge rst)begin
180             if(rst)
181                 shift_out <= 11'b1111111111; else
182                 if(load_d1)
183                     shift_out <= {bit10, bit9, ldata[6:0], 1'b0, 1'b1}; else
184                     if(btu)
185                         shift_out <= {1'b1, shift_out[10:1]};
186             end
187
188             always @(*)begin
189                 Tx = shift_out[0];
190             end
191
192
193 endmodule
194
```