```
`timescale 1ns / 1ps
/****************************************************************************
 * File Name: transmitEngine.v
 * Project: Tx Machine
 * Designer: Marc Cabote
 * Email: marcdominic011@gmail.com
 * Rev. Date: 10 March, 2018
 *
 * Purpose:The transmit engine has the logic needed to communicate with a
 *         computer using UART. For the terminal to have a stable connection
 *         in order to output the proper characters, the program—Realterm—
 *         should know what the setting is for the transmit engine—it should
 *         know the baud rate, odd or even parity and number of bits.
 *         If the program settings match the FPGA, it will then show
 *         "CSULB CECS 460 - [LINECOUNTER] <CR> <LF>".
 *
 * Notes:   -  This module has an asynchronous reset input.
 *
 ****************************************************************************/
module transmitEngine(input clk, rst, eight, pen, ohel, load,
                      input [7:0] out_port,
                      input [3:0] baud,
                      output reg TxRdy, Tx);

    wire ep, op, btu, done;

    reg doit, load_d1, bit10, bit9;
    reg [7:0] ldata;
    reg [3:0] bit_count, bit_counter;
    reg [10:0] shift_out;
    reg [18:0] bit_time_count, bit_time_counter, k;


    //=================================================================
    // TxRdy RS Flop
    //=================================================================
    always @(posedge clk, posedge rst)
       begin
         if(rst)
            TxRdy <= 1; else // high at reset
         if(done==1 && load==1)
            TxRdy <= TxRdy; else
         if(done)
            TxRdy <= 1; else
         if(load)
            TxRdy <= 0;
         else
            TxRdy <= TxRdy;
       end

   //=================================================================
   // Doit RS Flop
   //=================================================================

   always @(posedge clk, posedge rst)
       begin
         if(rst)
```

```
               doit <= 0; else
            if(done==1 && load_d1==1)
               doit <= doit; else
            if(done)
               doit <= 0; else
            if(load_d1)
               doit <= 1;
            else
               doit <= doit;
          end

      //===================================================================
      // 8-bit Loadable Register
      //===================================================================
      always @(posedge clk, posedge rst)
         begin
            if(rst)
               ldata <= 0;
            else
               ldata <= out_port;
         end


      //===================================================================
      // Baud Decoder
      //===================================================================
      always @(*) begin
            case (baud)
               4'b0000: k = 333333; //300
               4'b0001: k = 83333;  //1200
               4'b0010: k = 41667;  //2400
               4'b0011: k = 20833;  //4800
               4'b0100: k = 10417;  //9600
               4'b0101: k = 5208;   //19200
               4'b0110: k = 2604;   //38400
               4'b0111: k = 1736;   //57600
               4'b1000: k = 868;    //115200
               4'b1001: k = 434;    //230400
               4'b1010: k = 217;    //460800
               4'b1011: k = 109;    //921600
               default: k = 333333;
            endcase
      end

      //===================================================================
      //Bit Time Counter
      //===================================================================
      assign btu = (bit_time_count == k);

      always @(posedge clk, posedge rst)begin
            if(rst)
               bit_time_count <= 0;
            else
               bit_time_count <= bit_time_counter;
         end

      always @(*)begin
```

```
            case ({doit,btu})
               0: bit_time_counter = 0;
               1: bit_time_counter = 0;
               2: bit_time_counter = bit_time_count + 1;
               3: bit_time_counter = 0;
            endcase
         end

      //==================================================================
      //Bit Counter
      //==================================================================
      assign done = (bit_count == 11);

      always @(posedge clk, posedge rst)begin
            if(rst)
               bit_count <= 0;
            else
               bit_count <= bit_counter;
         end

      always @(*)begin
            case ({doit,btu})
               0: bit_counter = 0;
               1: bit_counter = 0;
               2: bit_counter = bit_count;
               3: bit_counter = bit_count + 1;
            endcase
         end

      //==================================================================
      // Parity Decoder
      //==================================================================
      assign ep = eight ? (^ldata) : ^(ldata[6:0]);
      assign op = eight ? !(^ldata): !(^(ldata[6:0]));

      always @*
       begin
         case ({eight,pen,ohel})
            0: {bit10,bit9} = {1'b1, 1'b1};
            1: {bit10,bit9} = {1'b1, 1'b1};
            2: {bit10,bit9} = {1'b1, ep};
            3: {bit10,bit9} = {1'b1, op};
            4: {bit10,bit9} = {1'b1, ldata[7]};
            5: {bit10,bit9} = {1'b1, ldata[7]};
            6: {bit10,bit9} = {ep, ldata[7]};
            7: {bit10,bit9} = {op, ldata[7]};
            default: {bit10,bit9} = {1'b1,1'b1};
         endcase
       end

      //==================================================================
      // LoadD1 D Flop
      //==================================================================
      always @(posedge clk, posedge rst)
         begin
            if(rst)
               load_d1 <= 0;
```

```verilog
            else
               load_d1 <= load;
         end

      //==================================================================
      // Shift Register
      //==================================================================
      always @(posedge clk, posedge rst)begin
         if(rst)
            shift_out <= 11'b11111111111; else
         if(load_d1)
            shift_out <= {bit10, bit9, ldata[6:0], 1'b0, 1'b1}; else
         if(btu)
            shift_out <= {1'b1, shift_out[10:1]};
       end

       always @(*)begin
         Tx = shift_out[0];
       end


   endmodule

```