

```
1  `timescale 1ns / 1ps
2  /*****
3   * File Name: receiveEngine.v
4   * Project:   FULL UART
5   * Designer: Marc Dominic Cabote
6   * Email: marcdominic011@gmail.com
7   * Rev. Date: 5 May, 2018
8   *
9   * Purpose: The receive engine has the logic needed to communicate with a
10  *           convert bits of signals received into data. Using a shift register,
11  *           The register is filled with data throu the Rx pin which either
12  *           goes high or low. Once the data has been received, It spits out
13  *           the data to the tramelblaze to be output in RealTerm.
14  *
15  * Notes:   - This module has a asynchronous reset input.
16  *
17  *****/
18  module receiveEngine(input clk, rst, Rx, eight, pen, reads0, even,
19                      input [18:0] k,
20                      output reg RxRdy, perr, ferr, ovf,
21                      output [7:0] data);
22
23  wire btu, done, shift, overflow, stop_bit, parity;
24
25  reg start, doit, parity_gen, parity_gen_sel, parity_bit_sel,
26     stop_bit_sel;
27  reg [1:0] pstate, nstate;
28  reg [3:0] bit_count, bit_counter, bits; // bit counter
29  reg [9:0] shift_out, combo_out;
30  reg [18:0] bit_time_count, bit_time_counter, bit_time; //bit-time counter
31
32  //=====
33  // State Machine
34  //=====
35  always @ (posedge clk, posedge rst)
36  begin
37      if (rst)
38          pstate <= 2'b00;
39      else
40          pstate <= nstate;
41  end
42
43  always @ (*)
44  begin
45      case (pstate)
46          //state 0
47          0: begin
48              {start, doit} = 2'b00;
49              //nstate = Rx ? 2'b00 : 2'b01; //if ~Rx go next
50              if(Rx)
51                  nstate = 0;
52              else
53                  nstate = 1;
54              end
55
56          //state 1
57          1: begin
```

```
58         {start, doit} = 2'b11;
59         //nstate = Rx ? 2'b00: btu ? 2'b10 : 2'b01; //if ~Rx & Btu go next
60         if (Rx)
61             nstate = 0;else
62         if (~Rx && btu)
63             nstate = 1;
64         else
65             nstate = 2;
66         end
67
68         //state 2
69     2: begin
70         {start, doit} = 2'b01;
71         //nstate = done ? 2'b00 : 2'b01; //if done go next
72         if (done)
73             nstate = 0;
74         else
75             nstate = 2;
76         end
77
78         //default state 0
79     default: begin
80         {start, doit} = 2'b00;
81         nstate = 2'b00;
82         end
83     endcase
84 end
85
86 //=====
87 //Bit Time Counter k select
88 //=====
89
90 always @ (*)
91     begin
92         if (start)
93             bit_time = k >> 1; //divide k in half
94         else
95             bit_time = k;      //else default baud
96         end
97
98 //=====
99 //Bit Time Counter
100 //=====
101 assign btu = (bit_time_count == bit_time);
102 assign shift = btu & ~start;
103
104 always @(posedge clk, posedge rst)
105     begin
106         if(rst)
107             bit_time_count <= 0;
108         else
109             bit_time_count <= bit_time_counter;
110         end
111
112 always @ (*)
113     begin
114         case ({doit,btu})
```

```
115         0: bit_time_counter = 0;
116         1: bit_time_counter = 0;
117         2: bit_time_counter = bit_time_count + 1;
118         3: bit_time_counter = 0;
119     endcase
120 end
121
122 //=====
123 //Bit Counter Number of Bits
124 //=====
125 always @ (*)
126     begin
127         case ({eight,pen})
128             0: bits = 9;
129             1: bits = 10;
130             2: bits = 10;
131             3: bits = 11;
132             default: bits = 9;
133         endcase
134     end
135
136 //=====
137 //Bit Counter
138 //=====
139 assign done = (bit_count == bits);
140
141 always @(posedge clk, posedge rst)
142     begin
143         if(rst)
144             bit_count <= 0;
145         else
146             bit_count <= bit_counter;
147     end
148
149 always @(*)
150     begin
151         case ({doit,btu})
152             0: bit_counter = 0;
153             1: bit_counter = 0;
154             2: bit_counter = bit_count;
155             3: bit_counter = bit_count + 1;
156         endcase
157     end
158
159 //=====
160 // Shift Register
161 //=====
162 always @(posedge clk, posedge rst)
163     begin
164         if (rst)
165             shift_out <= 10'b0; else
166             if (shift)
167                 shift_out <= {Rx, shift_out[9:1]};
168             else
169                 shift_out <= shift_out;
170     end
171
```

```
172 //=====
173 // Remap Combo
174 //=====
175 always @(*)
176     begin
177         case ({eight, pen})
178             0: combo_out = shift_out >> 2;           //7N1
179             1: combo_out = shift_out >> 1;           //7P1
180             2: combo_out = shift_out >> 1;           //8N1
181             3: combo_out = shift_out;                //8P1
182         endcase
183     end
184
185 assign data = combo_out [6:0]; //output to the TB
186
187 //=====
188 // Parity Gen Select
189 //=====
190 always @(*)
191     begin
192         if (eight)
193             parity_gen_sel = combo_out[7];
194         else
195             parity_gen_sel = 0;
196     end
197 //=====
198 // Parity Bit Select
199 //=====
200 always @(*)
201     begin
202         if (eight)
203             parity_bit_sel = combo_out[8];
204         else
205             parity_bit_sel = combo_out[7];
206     end
207
208 //=====
209 // Stop Bit Select
210 //=====
211 always @(*)
212     begin
213         case ({eight, pen})
214             0: stop_bit_sel = combo_out[7];
215             1: stop_bit_sel = combo_out[8];
216             2: stop_bit_sel = combo_out[8];
217             3: stop_bit_sel = combo_out[9];
218         endcase
219     end
220
221 //=====
222 // RxRdy RS Flop
223 //=====
224 always @(posedge clk, posedge rst)
225     begin
226         if(rst)
227             RxRdy <= 0;else
228             if(done)
```

```
229         RxRdy <= 1;else
230         if(reads0)
231             RxRdy <= 0;
232         else
233             RxRdy <= RxRdy;
234     end
235
236     //=====
237     // Parity Error RS Flop
238     //=====
239     always @(*)
240     begin
241         if (even)
242             parity_gen = parity_gen_sel;
243         else
244             parity_gen = ~parity_gen_sel;
245     end
246
247     assign parity = (parity_gen ^ parity_bit_sel) & pen & done;
248
249     always @ (posedge clk, posedge rst)
250     begin
251         if (rst)
252             perr <= 0; else
253             if (parity)
254                 perr <= 1; else
255             if (reads0)
256                 perr <= 0;
257             else
258                 perr <= perr;
259     end
260
261     //=====
262     // Framing Error RS Flop
263     //=====
264     assign stop_bit = done & ~stop_bit_sel;
265
266     always @ (posedge clk, posedge rst)
267     begin
268         if (rst)
269             ferr <= 0; else
270             if (stop_bit)
271                 ferr <= 1; else
272             if (reads0)
273                 ferr <= 0;
274             else
275                 ferr <= ferr;
276     end
277
278     //=====
279     // Overflow Error RS Flop
280     //=====
281     assign overflow = RxRdy & done;
282
283     always @ (posedge clk, posedge rst)
284     begin
285         if (rst)
```

```
286         ovf <= 0; else
287         if (overflow)
288         ovf <= 1; else
289         if (reads0)
290         ovf <= 0;
291         else
292         ovf <= ovf;
293     end
294
295 endmodule
296
```