

```

1  // ee417 lesson 8 Assignment 2 L8A2
2  // Name: Ron Kalin, Date: 07-07-24 Group: Kalin/Jammeh
3  // Design: UART Receiver (Rx)
4  // top level module
5  module UART_RCVR #(parameter word_size = 8, half_word = word_size /2)
6  (output [word_size -1: 0] RCV_datareg,
7   output read_not_ready_out, Error1, Error2,
8   input Serial_in, read_not_ready_in, Sample_clk, rst_b );
9
10 Control_Unit M0 (
11     read_not_ready_out,
12     Error1,
13     Error2,
14     clr_Sample_counter,
15     inc_Sample_counter,
16     clr_Bit_counter,
17     inc_Bit_counter,
18     shift,
19     load,
20     read_not_ready_in,
21     Ser_in_0,
22     SC_eq_3,
23     SC_lt_7,
24     BC_eq_8,
25     Sample_clk,
26     rst_b );
27
28 Datapath_Unit M1 (
29     RCV_datareg,
30     Ser_in_0,
31     SC_eq_3,
32     SC_lt_7,
33     BC_eq_8,
34     Serial_in,
35     clr_Sample_counter,
36     inc_Sample_counter,
37     clr_Bit_counter,
38     inc_Bit_counter,
39     shift,
40     load,
41     Sample_clk,
42     rst_b );
43
44 endmodule
45
46 //control unit..set parameters and output registers
47 module Control_Unit #(
48     parameter word_size = 8, half_word = word_size /2, Num_state_bits = 2, idle = 2'b00,
49     starting = 2'b01, receiving = 2'b10// one-hot assignment
50 )(
51     output reg read_not_ready_out,
52             Error1, Error2,
53             clr_Sample_counter,
54             inc_Sample_counter,
55             clr_Bit_counter,
56             inc_Bit_counter,
57             shift,
58             load,
59     input read_not_ready_in,
60           Ser_in_0,
61           SC_eq_3,
62           SC_lt_7,
63           BC_eq_8,
64           Sample_clk,
65           rst_b );
66 //internal registers
67 reg [word_size-1: 0] RCV_shftreg;
68 reg [Num_state_bits -1: 0] state, next_state;
69 always @ (posedge Sample_clk) //setup idle state
70     if (rst_b == 1'b0) state <= idle; else state <= next_state;

```

```

70  always @ (state, Ser_in_0, SC_eq_3, SC_lt_7, read_not_ready_in) begin
71      read_not_ready_out = 0; //initialize registers to zero
72      clr_Sample_counter = 0;
73      clr_Bit_counter = 0;
74      inc_Sample_counter = 0;
75      inc_Bit_counter = 0;
76      shift = 0;
77      Error1 = 0;
78      Error2 = 0;
79      load = 0;
80      next_state = idle;
81      case (state) //set up states from state diagram
82          idle: if (Ser_in_0 == 1'b1) next_state = starting;
83              else next_state = idle;
84          starting: if (Ser_in_0 == 1'b0) begin //ser_in_0 is deasserted means ready to start
85                      next_state = idle;
86                      clr_Sample_counter = 1;
87                  end
88              else if (SC_eq_3 == 1'b1) begin //when sample count equal to 3 move to
receiving
89                      next_state = receiving;
90                      clr_Sample_counter = 1;
91                  end else begin inc_Sample_counter = 1; next_state = starting; end //else
continue with starting
92          receiving: if (SC_lt_7 == 1'b1) begin //sample count less than 7, continue receiving
93                      inc_Sample_counter = 1;
94                      next_state = receiving;
95                  end
96                  else begin
97                      clr_Sample_counter = 1;
98                      if (!BC_eq_8) begin //bit count not equal to 8, continue receiving
99                          shift = 1;
100                         inc_Bit_counter = 1;
101                         next_state = receiving;
102                     end
103                     else begin
104                         next_state = idle; //sample count =7, and bit count = 8, then move
back to idle
105                         read_not_ready_out = 1;
106                         clr_Bit_counter = 1;
107                         if (read_not_ready_in == 1'b1) Error1 = 1; //read_not_ready_in error,
send error1 high
108                         else if (Ser_in_0 == 1'b1) Error2 = 1; //ser_in_0 error, send
error2 high
109                         else load = 1; //if no errors then load
goes high to load data
110                     end
111                     end
112          default: next_state = idle; //default case is idle
113      endcase
114  end
115  endmodule
116
117  module Datapath_Unit #(parameter
118      word_size = 8, half_word = word_size /2, Num_counter_bits = 4
119  )(
120      output reg [word_size-1: 0] RCV_datareg, //the function of each of these outputs and
inputs is listed in the block diagram
121      output
122          Ser_in_0,
123          SC_eq_3,
124          SC_lt_7,
125          BC_eq_8,
126      input
127          Serial_in,
128          clr_Sample_counter,
129          inc_Sample_counter,
130          clr_Bit_counter,
131          inc_Bit_counter,
132          shift,
133          load,
134          sample_clk,

```

```
133         rst_b );
134     reg [word_size-1: 0] RCV_shftreg;
135     reg [Num_counter_bits-1: 0] Sample_counter;
136     reg [Num_counter_bits: 0] Bit_counter;
137     assign Ser_in_0 = (Serial_in == 1'b0); //when serial_in =0, ser_in_0 = 0
138     assign BC_eq_8 = (Bit_counter == word_size); //bit counter equal to 8
139     assign SC_lt_7 = (Sample_counter < word_size -1); //shift counter less than 7
140     assign SC_eq_3 = (Sample_counter == half_word -1); //shift counter equal to 3
141
142     always @ (posedge Sample_clk)
143     if (rst_b == 1'b0) begin // synchronous rst_b, reset is low means system reset, system
inactive
144         Sample_counter <= 0; // registers get the value of zero
145         Bit_counter <= 0;
146         RCV_datareg <= 0;
147         RCV_shftreg <= 0;
148     end
149     else begin //reset low
150         if (clr_Sample_counter == 1) Sample_counter <= 0; //clear sample counter
151         else if (inc_Sample_counter == 1) Sample_counter <= Sample_counter + 1; //increment
sample counter
152
153         if (clr_Bit_counter == 1) Bit_counter <= 0; //clear bit counter
154         else if (inc_Bit_counter == 1) Bit_counter <= Bit_counter + 1; //increment bit counter
155         if (shift == 1) RCV_shftreg <= {Serial_in, RCV_shftreg[word_size-1:1]}; //shift high
causes shift register to shift towards least significant bit
156         if (load == 1) RCV_datareg <= RCV_shftreg; //load high means data register gets the
value of shift register
157     end
158 endmodule
```