```verilog
// ee417 lesson 8 Assignment 2 L8A2
// Name: Ron Kalin, Date: 07-07-24  Group: Kalin/Jammeh
// Design: UART Receiver (Rx)
// top level module
module UART_RCVR #(parameter word_size = 8, half_word = word_size /2)
(output [word_size -1: 0] RCV_datareg,
 output read_not_ready_out, Error1, Error2,
 input Serial_in,  read_not_ready_in,  Sample_clk,  rst_b );

Control_Unit M0 (
  read_not_ready_out,
  Error1,
  Error2,
  clr_Sample_counter,
  inc_Sample_counter,
  clr_Bit_counter,
  inc_Bit_counter,
  shift,
  load,
  read_not_ready_in,
  Ser_in_0,
  SC_eq_3,
  SC_lt_7,
  BC_eq_8,
  Sample_clk,
  rst_b  );

Datapath_Unit M1 (
  RCV_datareg,
  Ser_in_0,
  SC_eq_3,
  SC_lt_7,
  BC_eq_8,
  Serial_in,
  clr_Sample_counter,
  inc_Sample_counter,
  clr_Bit_counter,
  inc_Bit_counter,
  shift,
  load,
  Sample_clk,
  rst_b  );

endmodule

//control unit..set parameters and output registers
module Control_Unit #(
  parameter  word_size = 8, half_word = word_size /2, Num_state_bits = 2,  idle = 2'b00,
starting = 2'b01, receiving = 2'b10// one-hot assignment
)(
  output reg read_not_ready_out,
         Error1, Error2,
         clr_Sample_counter,
         inc_Sample_counter,
         clr_Bit_counter,
         inc_Bit_counter,
         shift,
         load,
         input read_not_ready_in,
         Ser_in_0,
         SC_eq_3,
         SC_lt_7,
         BC_eq_8,
         Sample_clk,
         rst_b  );
//internal registers
  reg [word_size-1: 0]       RCV_shftreg;
  reg [Num_state_bits -1: 0] state, next_state;
  always @ (posedge Sample_clk)  //setup idle state
    if (rst_b == 1'b0) state <= idle; else state <= next_state;
  always @ (state, Ser_in_0, SC_eq_3, SC_lt_7, read_not_ready_in) begin
   read_not_ready_out = 0;  //initialize registers to zero
   clr_Sample_counter = 0;
   clr_Bit_counter    = 0;
   inc_Sample_counter = 0;
   inc_Bit_counter    = 0;
   shift              = 0;
   Error1             = 0;
   Error2             = 0;
   load               = 0;
   next_state = idle;
   case (state)  //set up states from state diagram
     idle: if (Ser_in_0 == 1'b1) next_state = starting;
           else next_state = idle;
     starting:   if (Ser_in_0 == 1'b0) begin  //ser_in_0 is deasserted means ready to start
                  next_state = idle;
                  clr_Sample_counter  = 1;
                 end
                 else if (SC_eq_3 == 1'b1) begin  //when sample count equal to 3 move to receiving
                  next_state = receiving;
                  clr_Sample_counter  = 1;
                 end else begin inc_Sample_counter = 1; next_state = starting; end  //else continue with starting
     receiving:  if (SC_lt_7 == 1'b1) begin  //sample count less than 7, continue receiving
                   inc_Sample_counter  = 1;
                   next_state = receiving;
                 end
                 else begin
                  clr_Sample_counter  = 1;
                  if (!BC_eq_8) begin       //bit count not equal to 8, continue receiving
                   shift  = 1;
                   inc_Bit_counter  = 1;
                   next_state = receiving;
                  end
                  else begin
                   next_state = idle;       //sample count =7, and bit count = 8, then move back to idle
                   read_not_ready_out  = 1;
                   clr_Bit_counter  = 1;
                   if (read_not_ready_in == 1'b1) Error1 = 1;  //read_not_ready_in error, send error1 high
                   else if (Ser_in_0 == 1'b1) Error2 = 1;      //ser_in_0 error, send error2 high
                   else load = 1;                              //if no errors then load goes high to load data
                  end
                 end
     default:    next_state = idle;         //default case is idle
    endcase
   end
endmodule

module Datapath_Unit #(parameter
 word_size = 8, half_word = word_size /2, Num_counter_bits = 4
)(
 output reg [word_size-1: 0] RCV_datareg, //the function of each of these outputs and inputs is listed in the block diagram
 output                        Ser_in_0,
                               SC_eq_3,
                               SC_lt_7,
                               BC_eq_8,
 input                         Serial_in,
                               clr_Sample_counter,
                               inc_Sample_counter,
                               clr_Bit_counter,
                               inc_Bit_counter,
                               shift,
                               load,
                               Sample_clk,
                              rst_b  );
 reg [word_size-1: 0]         RCV_shftreg;
 reg [Num_counter_bits-1: 0] Sample_counter;
 reg [Num_counter_bits: 0]   Bit_counter;
 assign Ser_in_0 = (Serial_in == 1'b0);             //when serial_in =0, ser_in_0 = 0
 assign BC_eq_8 = (Bit_counter == word_size);       //bit counter equal to 8
 assign SC_lt_7 = (Sample_counter < word_size -1);  //shift counter less than 7
 assign SC_eq_3 = (Sample_counter == half_word -1); //shift counter equal to 3

 always @ (posedge Sample_clk)
 if (rst_b == 1'b0) begin // synchronous rst_b, reset is low means system reset, system inactive
   Sample_counter <= 0;   // registers get the value of zero
   Bit_counter <= 0;
   RCV_datareg <= 0;
   RCV_shftreg <= 0;
 end
 else begin                 //reset low
  if (clr_Sample_counter == 1) Sample_counter <= 0;  //clear sample counter
  else if (inc_Sample_counter == 1) Sample_counter <= Sample_counter + 1;  //increment sample counter

  if (clr_Bit_counter == 1) Bit_counter <= 0;        //clear bit counter
  else if (inc_Bit_counter == 1) Bit_counter <= Bit_counter + 1;  //increment bit counter
  if (shift == 1) RCV_shftreg <= {Serial_in, RCV_shftreg[word_size-1:1]};  //shift high causes shift register to shift towards least significant bit
  if (load == 1) RCV_datareg <= RCV_shftreg;  //load high means data register gets the value of shift register
 end
endmodule
```