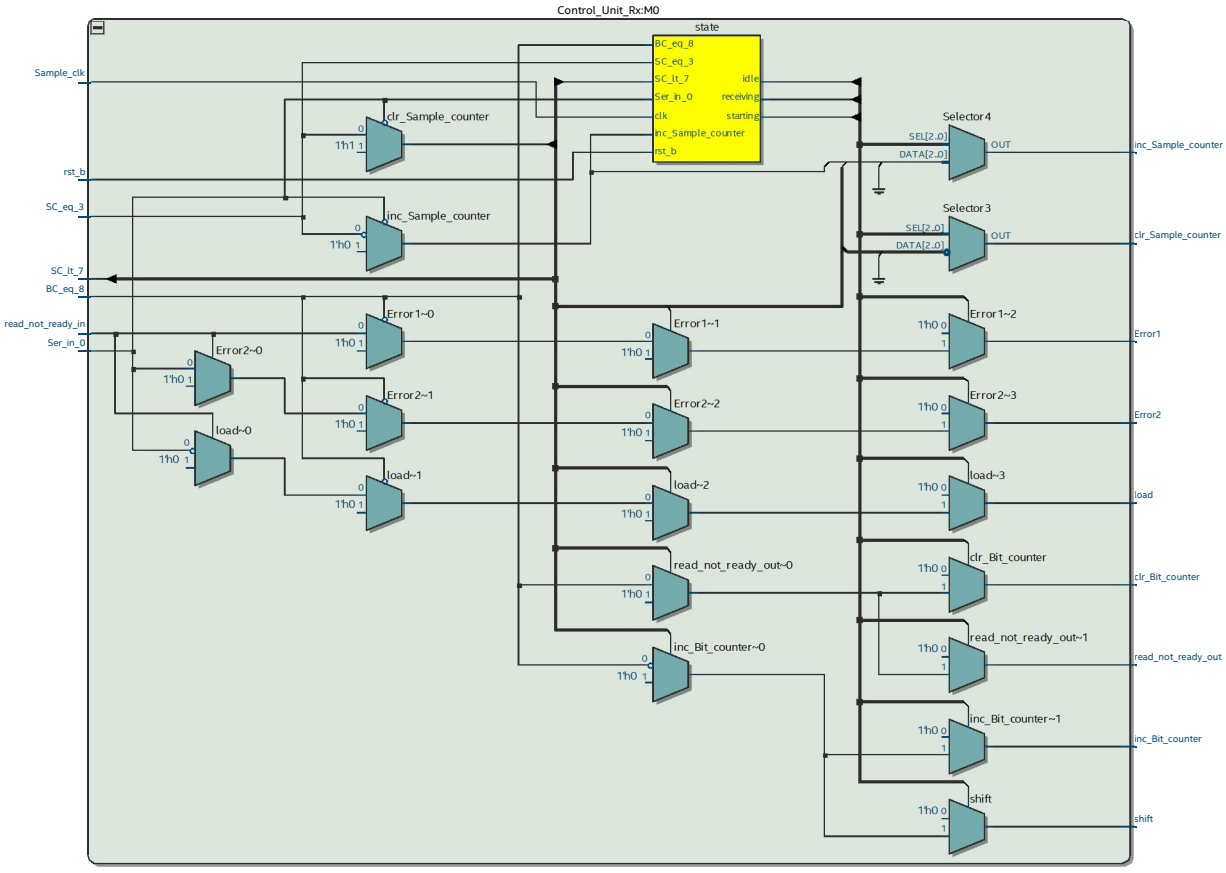
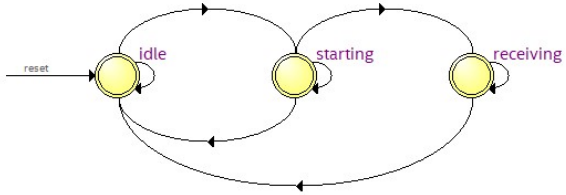
```
/*-------------------------------------------------------------------------------
Name Lamin Jammeh
CLass: EE417 Summer 2024
Lesson 08 HW Question 2
Group: Ron Kalin/ Lamin Jammeh
Project Description: Control_Unit for Reciever (Rx) will control the datapath for the
UART Rx. Once the Data is Transmitted by the Host processor, the Control_unit will
perform several checks before allowing the Datapath to load the recieved data
-------------------------------------------------------------------------------*/

module Control_Unit_Rx  #(parameter word_size = 8,
                          parameter half_word = word_size/2,
                          parameter Num_state_bits = 2
                         )
                        //state the outputs of the COntroller Unit
                        (output reg read_not_ready_out,
                         output reg Error1,
                         output reg Error2,
                         output reg clr_Sample_counter,
                         output reg inc_Sample_counter,
                         output reg clr_Bit_counter,
                         output reg inc_Bit_counter,
                         output reg shift,
                         output reg load,
                       //define the inputs of the Controller Unit
                         input read_not_ready_in,
                         input Ser_in_0,
                         input SC_eq_3,
                         input SC_lt_7,
                         input BC_eq_8,
                         input Sample_clk,
                         input rst_b
                         );

//Parameterize the different states of the Controller Unit as a 2-bit one-hot counter
parameter idle      = 2'b00;
parameter starting  = 2'b01;
parameter receiving = 2'b10;

//define the internal registers for state, next_stae and shift_reg
reg   [word_size-1:0]   Rx_shftreg;             //creates a temp register of 8-bits to
store the received data
reg   [Num_state_bits-1:0]  state, next_state;     //creates temp register of 2-bits for
the stae transition

//state Transition logic
always @(posedge Sample_clk)
   if   (rst_b == 1'b0)
        state <= idle;
   else state <= next_state;
   /*---------------   for the State Transition--------------------------
   ---if reset is low (0) the controller unit will remain in idle
   ---if reset is high (1) the controller will move to the next-state----*/

//create a initial or default condition for the controller
always @(state, Ser_in_0, SC_eq_3, SC_lt_7, read_not_ready_in)
   begin
      read_not_ready_out = 0;
      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;
      //create the Next State logic
      case (state)
```

```
68            idle: if   (Ser_in_0 == 1'b1)
69                         next_state = starting;
70                  else next_state = idle;
71          /*-----------------@ idle------------------------------
72          --- if Ser_in_0 is high (1) move to starting state------
73          --- if Ser_in_0 is low (0) remain idle-----------------*/
74            starting: if (Ser_in_0 ==1'b0)
75                        begin
76                          next_state = idle;
77                          clr_Sample_counter = 1;
78                        end                          //Keep receiving the Ser_in_0 = 1'b0 until
    the next else if condition is met
79                      else if (SC_eq_3 ==1'b1)
80                        begin
81                          next_state = receiving;
82                          clr_Sample_counter = 1;
83                        end                          //once Sample_counter = 3 then move to the
    receiving state and clr_Sample_counter
84                       else
85                          begin
86                             inc_Sample_counter = 1;
87                             next_state = starting;
88                          end                  //  when the if conditons are false remain @
    starting and increment the Sample_counter
89            receiving: if (SC_lt_7 == 1'b1)
90                          begin
91                             inc_Sample_counter = 1;
92                             next_state = receiving;
93                          end                  //Keep receiving data as long as SC less than
    7 is high (1) and remain the receiving state
94                        else                   //once SC_lt_7 drops low the current data
    package is completed
95                          begin
96                             clr_Sample_counter = 1;    //clr the Sample_counter and check
    the next if condition
97                             if (!BC_eq_8)              //if BC_eq_8 is not true or low
98                                begin
99                                   shift = 1;           //send shift high to the Datapath
100                                  inc_Bit_counter = 1;
101                                  next_state = receiving;
102                               end             //keep shift as ong as BC_eq_8 is false and
    remain in receiving state
103                            else
104                               begin
105                                  next_state = idle;      //if BC_eq_8 id high move to idle
106                                  read_not_ready_out = 1; //and send read_not_ready_out to
    the Datapath
107                                  clr_Bit_counter = 1;
108                            //check the integrity of the data received and status of the
    host
109                                  if (read_not_ready_in == 1'b1)   //host not ready to
    receive the data
110                                     Error1 = 1;
111                                  else if (Ser_in_0 ==1'b1)        //stop bit not received
112                                     Error2 = 1;
113                                  else              // the both Error1 and 2 are false the
    data integrity is correct and send load signal
114                                     load = 1;         //send load signal to the Datapath
115                               end
116                            end
117               default: next_state = idle;
118           endcase
119      end
120   endmodule
121
122
123
124
125
```

126


Control_Unit_Rx:M0
state
BC_eq_8
SC_eq_3
SC_lt_7
Ser_in_0
clk
inc_Sample_counter
rst_b
idle
receiving
starting
Sample_clk
rst_b
SC_eq_3
SC_lt_7
BC_eq_8
read_not_ready_in
Ser_in_0
clr_Sample_counter
inc_Sample_counter
Error1~0
Error2~0
Error2~1
load~0
load~1
Error1~1
Error2~2
load~2
read_not_ready_out~0
inc_Bit_counter~0
Selector4
Selector3
SEL[2..0]
DATA[2..0]
OUT
Error1~2
Error2~3
load~3
clr_Bit_counter
read_not_ready_out~1
inc_Bit_counter~1
shift
inc_Sample_counter
clr_Sample_counter
Error1
Error2
load
clr_Bit_counter
read_not_ready_out
inc_Bit_counter
shift

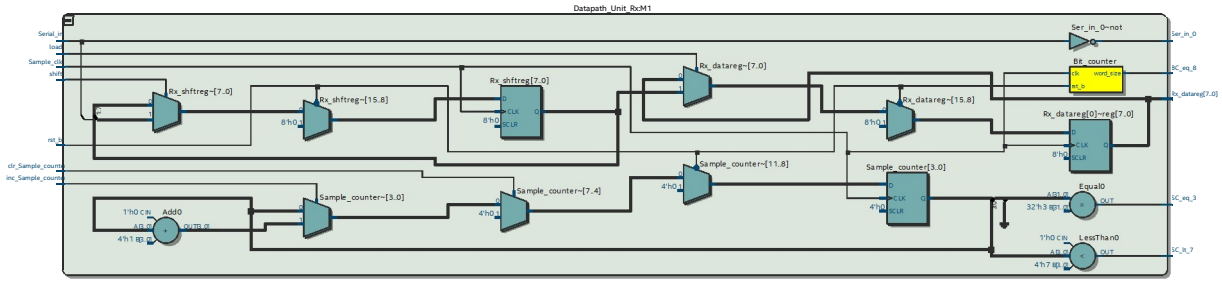


Table

	Source State	Destination State	Condition
1	idle	starting	(Ser_in_0).(rst_b)
2	idle	idle	(!Ser_in_0) + (Ser_in_0).(!rst_b)
3	receiving	receiving	(!BC_eq_8).(rst_b) + (BC_eq_8).(SC_lt_7).(rst_b)
4	receiving	idle	(!BC_eq_8).(!rst_b) + (BC_eq_8).(!SC_lt_7) + (BC_eq_8).(SC_lt_7).(!rst_b)
5	starting	receiving	(SC_eq_3).(Ser_in_0).(rst_b)
6	starting	starting	(inc_Sample_counter).(rst_b)
7	starting	idle	(!Ser_in_0) + (Ser_in_0).(!rst_b)

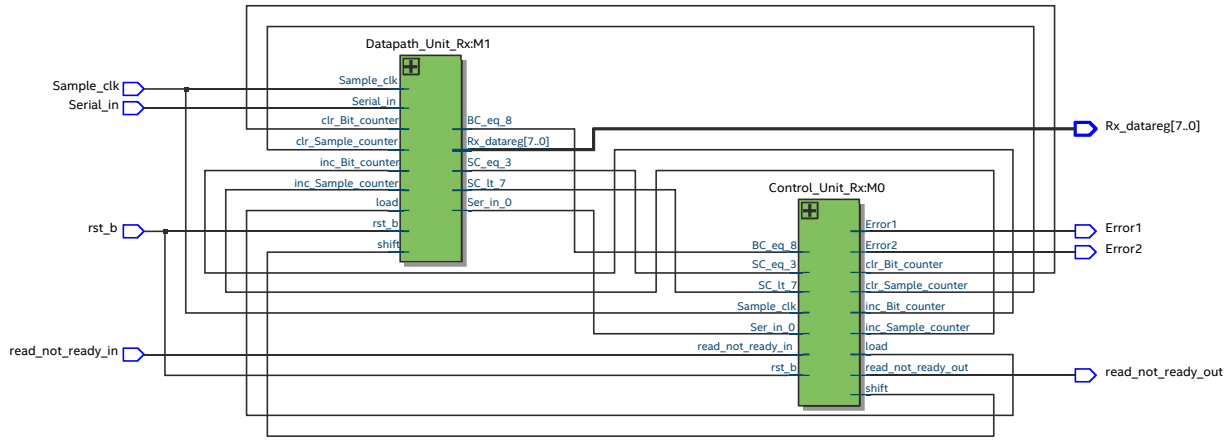
```
/*-------------------------------------------------------------------------------
Name Lamin Jammeh
CLass: EE417 Summer 2024
Lesson 08 HW Question 2
Group: Ron Kalin/ Lamin Jammeh
Project Description: DataPath_Unit for Reciever (Rx) will receive input command from the Control_unit.
The DataPath wiil check for input command from the control_Unit and it will do the following
load: the databus to the data_reg
shift: the data received in the shift_reg
-------------------------------------------------------------------------------*/

module Datapath_Unit_Rx #(parameter word_size = 8,
                          parameter half_word = word_size/2,
                          parameter Num_counter_bits = 4
                         )
                        //output of the DataPath Unit
                         (output reg [word_size-1:0]  Rx_datareg,
                          output                      Ser_in_0,
                          output                      SC_eq_3,
                          output                      SC_lt_7,
                          output                      BC_eq_8,
                        //define the inputs of the DataPath Unit
                          input                       Serial_in,
                          input                       clr_Sample_counter,
                          input                       inc_Sample_counter,
                          input                       clr_Bit_counter,
                          input                       inc_Bit_counter,
                          input                       shift,
                          input                       load,
                          input                       Sample_clk,
                          input                       rst_b
                         );

//create a temp register to store the shift_register, Sample_counter, and Bit_counter
reg   [word_size-1:0]         Rx_shftreg;
reg   [Num_counter_bits-1:0]  Sample_counter;
reg   [Num_counter_bits:0]    Bit_counter;

//assign an internal output to the inputs of the DataPath (define the the internal Probes)
assign Ser_in_0 = (Serial_in == 1'b0);
assign BC_eq_8  = (Bit_counter == word_size);
assign SC_lt_7  = (Sample_counter < word_size-1);
assign SC_eq_3  = (Sample_counter == half_word-1);

//initial condition of the DataPath
always   @(posedge Sample_clk)
   if (rst_b == 1'b0)            //synchronous rst_b
      begin
         Sample_counter <= 0;
         Bit_counter    <= 0;
         Rx_datareg     <= 0;
         Rx_shftreg     <= 0;
      end            //when rst_b is low (0) alway internal registers will be 0
   else
   // if rst_b is high check the following if condtions and start moving data through the DataPath
      begin
         if (clr_Sample_counter == 1)
            Sample_counter <= 0;  //sample_counter register is cleared after clr_Sample_counter bit goes high (1)

         else if (inc_Sample_counter == 1)
            Sample_counter <= Sample_counter + 1; //increment Sample_counter once inc_sample_counter bit goes high (1) and clr_sample_counter bit is low

         if (shift == 1)
            Rx_shftreg <= {Serial_in, Rx_shftreg[word_size-1:1]}; //concat the Serial_in with content of shft_reg[7:1] (shift right or toward LSB)
```

```
65
66            if (load ==1)
67               Rx_datareg <= Rx_shftreg; //load the entire content of shft_reg into datareg
    without any change (parallel load)
68          end
69   endmodule
70
71
72
73
```

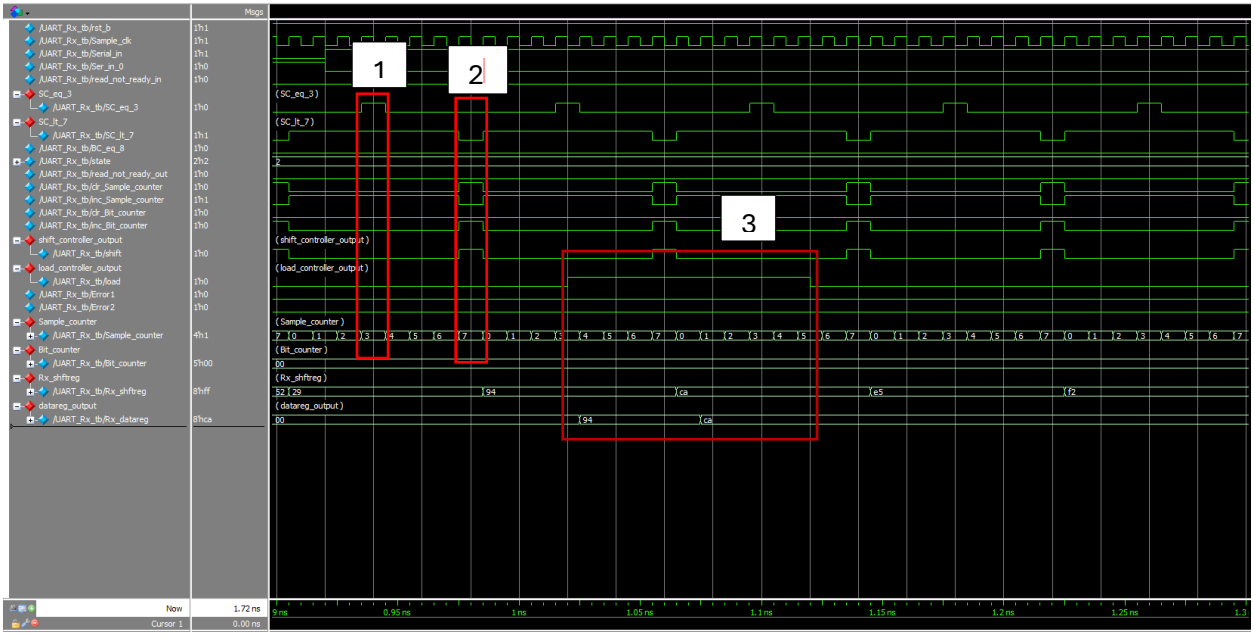

Datapath_Unit_RxM1
Ser_in_0~not
BR_counter
Rx_shftreg~[7..0]
Rx_shftreg~[15..8]
Rx_shftreg[7..0]
Rx_datareg~[7..0]
Rx_datareg~[15..8]
Rx_datareg[0]~reg[7..0]
Sample_counter~[3..0]
Sample_counter~[7..4]
Sample_counter~[11..8]
Sample_counter[3..0]
Add0
Equal0
LessThan0

```
/*-----------------------------------------
Name Lamin Jammeh
CLass: EE417 Summer 2024
Lesson 08 HW Question 2
Group: Ron Kalin/ Lamin Jammeh
Project Description: TestBench for teh UART_Rx
-----------------------------------------*/

module UART_Rx_tb ();

//define the registers and wires
reg   Serial_in, read_not_ready_in, Sample_clk, rst_b;
wire  [7:0]    Rx_datareg;
wire  read_not_ready_out;
wire  Error1, Error2;

//add wires to monitor the inputs and outputs on the submodules
wire        Ser_in_0;
wire        clr_Sample_counter;
wire        inc_Sample_counter;
wire        clr_Bit_counter;
wire        inc_Bit_counter;
wire [7:0]  Rx_shftreg;
wire [3:0]  Sample_counter;
wire [4:0]  Bit_counter;
wire        SC_eq_3;
wire        SC_lt_7;
wire [1:0]  state;
wire        shift;
wire        load;
wire        BC_eq_8;

//define the unit under test UUT
UART_Rx #(8, 4) UUT (
                     .Rx_datareg(Rx_datareg),
                     .read_not_ready_out(read_not_ready_out),
                     .Error1(Error1),
                     .Error2(Error2),

                     .Serial_in(Serial_in),
                     .read_not_ready_in(read_not_ready_in),
                     .Sample_clk(Sample_clk),
                     .rst_b(rst_b)
                     );

//assign internal probe monitor
assign Ser_in_0             = UUT.M1.Ser_in_0;
assign clr_Sample_counter   = UUT.M1.clr_Sample_counter;
assign inc_Sample_counter   = UUT.M1.inc_Sample_counter;
assign clr_Bit_counter      = UUT.M1.clr_Bit_counter;
assign inc_Bit_counter      = UUT.M1.inc_Bit_counter;
assign Rx_shftreg           = UUT.M1.Rx_shftreg;
assign Sample_counter       = UUT.M1.Sample_counter;
assign Bit_counter          = UUT.M1.Bit_counter;
assign SC_eq_3              = UUT.M0.SC_eq_3;
assign SC_lt_7              = UUT.M0.SC_lt_7;
assign state                = UUT.M0.state;
assign shift                = UUT.M0.shift;
assign load                 = UUT.M0.load;
assign BC_eq_8              = UUT.M0.BC_eq_8;

//clock cycle
always
   begin
      Sample_clk = 0;
      forever #5 Sample_clk = ~Sample_clk;
   end

//initialie reset and run enough clk cycle to get all desired counts
```

```
70   initial
71      begin
72         // Initialize Inputs
73       Serial_in = 0;
74       read_not_ready_in = 0;
75       Sample_clk = 0;
76       rst_b = 0;
77
78       // Apply reset
79       rst_b = 1;
80       #10;
81       rst_b = 0;
82       #10;
83       rst_b = 1;
84
85       // Test Case 1: Transmit a byte (e.g., 8'b10101010)
86       Serial_in = 1; // Start bit
87       #100;
88
89       Serial_in = 1; // Bit 0
90       #100;
91
92       Serial_in = 0; // Bit 1
93       #100;
94
95       Serial_in = 1; // Bit 2
96       #100;
97
98       Serial_in = 0; // Bit 3
99       #100;
100
101      Serial_in = 1; // Bit 4
102      #100;
103
104      Serial_in = 0; // Bit 5
105      #100;
106
107      Serial_in = 1; // Bit 6
108      #100;
109
110      Serial_in = 0; // Bit 7
111      #100;
112
113      Serial_in = 1; // Stop bit
114      #100;
115
116
117       // Force load signal to load the data
118      force UUT.M0.load = 1;
119      #100;
120      force UUT.M0.load = 0;
121      //relese load signal
122      release UUT.M0.load;
123      // Wait for a few cycles
124      #500;
125      #100;
126      $stop;
127        begin
128           $monitor ($time ,, "Serial_in = %h  Rx_datareg = %h", Serial_in, Rx_datareg);
129        end
130     end
131  endmodule
132
133
```


Datapath_Unit_Rx:M1
Sample_clk
Serial_in
clr_Bit_counter
clr_Sample_counter
inc_Bit_counter
inc_Sample_counter
load
rst_b
shift
BC_eq_8
Rx_datareg[7..0]
SC_eq_3
SC_lt_7
Ser_in_0
Control_Unit_Rx:M0
BC_eq_8
SC_eq_3
SC_lt_7
Sample_clk
Ser_in_0
read_not_ready_in
rst_b
Error1
Error2
clr_Bit_counter
clr_Sample_counter
inc_Bit_counter
inc_Sample_counter
load
read_not_ready_out
shift
Sample_clk
Serial_in
rst_b
read_not_ready_in
Rx_datareg[7..0]
Error1
Error2
read_not_ready_out

Bitwave



1. The SC_eq_3 only goes high when the Sample_counter is 3
2. SC_lt_7 goes low when the Sample_counter is 7
3. The Rx_datareg is loaded with the Rx_shftreg when load signa is received from the control_unit