Giao Tran

ECE 111 Homework 8 report

Homework 8a: SystemVerilog FSM code for Uart Receive

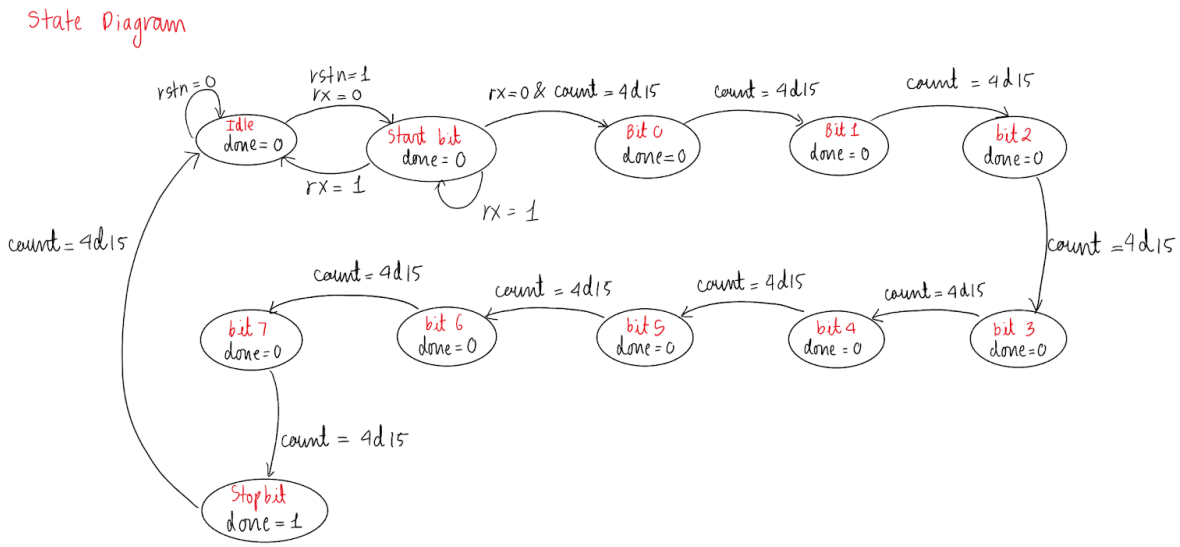
a) Design code

```
// UART RX RTL Code
module uart_rx #(parameter NUM_CLKS_PER_BIT=16)
(input logic clk, rstn,
 input logic rx, // input serial incoming data
 output logic done, // indicates 8-bit serial data is converted into 8-bit parallel data and available on dout port
 output logic [7:0] dout // 8-bit parallel data output
);

// count variable
logic [$clog2(NUM_CLKS_PER_BIT)-1:0] count;

// state encoding and state variable
enum logic[3:0]{
  RX_IDLE       = 4'b0000,
  RX_START_BIT  = 4'b0001,
  RX_DATA_BIT0  = 4'b0010,
  RX_DATA_BIT1  = 4'b0011,
  RX_DATA_BIT2  = 4'b0100,
  RX_DATA_BIT3  = 4'b0101,
  RX_DATA_BIT4  = 4'b0110,
  RX_DATA_BIT5  = 4'b0111,
  RX_DATA_BIT6  = 4'b1000,
  RX_DATA_BIT7  = 4'b1001,
  RX_STOP_BIT   = 4'b1010} state;


// FSM with single always block for next state,
// present state flipflop and output logic
always_ff@(posedge clk) begin
 if(!rstn) begin
   done <= 0;
   count <= 0;
   dout <= 0;
   state <= RX_IDLE;
 end
 else begin
    case(state)
       RX_IDLE: begin
           done <= 0;
           count <= 0;
           dout <= 0;
           // Wait for rx = 0 indicating start bit
       if(rx == 0) state <= RX_START_BIT;
       else state <= RX_IDLE;
       end
       RX_START_BIT: begin
           // sample start bit value at mid-point, for start bit counter
       // value = 7 is midpoint
           // wait for rx to transition from 1 to 0
           if(rx == 0 && count == ((NUM_CLKS_PER_BIT-1)/2)) begin
               done <= 0;
               state <= RX_DATA_BIT0;
               count <= 0;
               dout <= 0;
           end else begin
               count <= count + 1;
           end
    end
        RX_DATA_BIT0: begin
            if (count==NUM_CLKS_PER_BIT-1)
                begin
                    count<=0;
                    dout[0]<=rx;
                    state<=RX_DATA_BIT1;
                    done<=0;
                end
            else
                begin
```

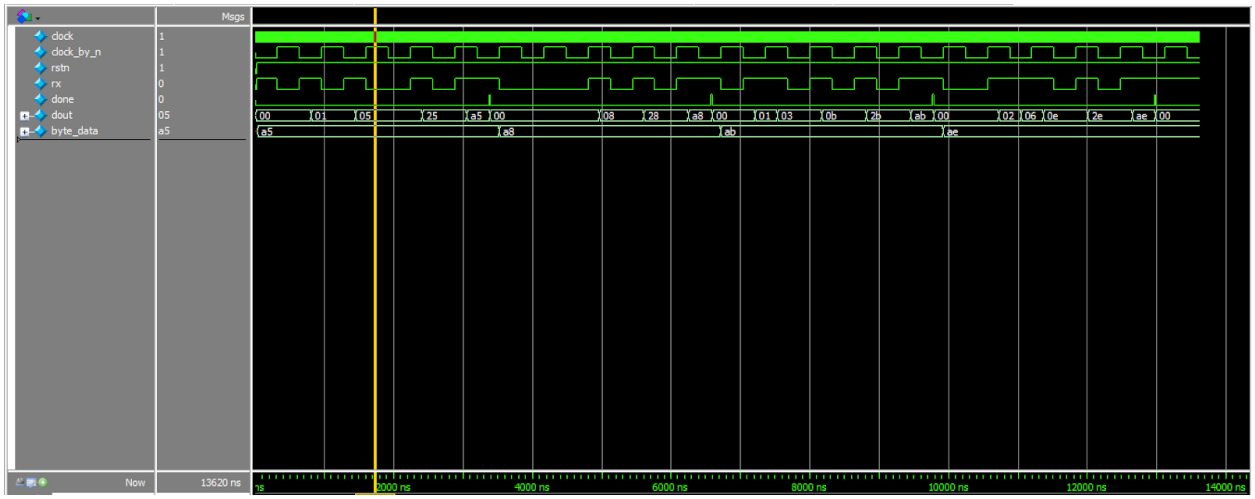
```
69                      count<=count+1;
70                  end
71              end
72
73              RX_DATA_BIT1: begin
74                  if (count==NUM_CLKS_PER_BIT-1)
75                      begin
76                          count<=0;
77                          dout[1]<=rx;
78                          state<=RX_DATA_BIT2;
79                          done<=0;
80                      end
81                  else
82                      begin
83                          count<=count+1;
84                      end
85              end
86
87              RX_DATA_BIT2: begin
88                  if (count==NUM_CLKS_PER_BIT-1)
89                      begin
90                          count<=0;
91                          dout[2]<=rx;
92                          state<=RX_DATA_BIT3;
93                          done<=0;
94                      end
95                  else
96                      begin
97                          count<=count+1;
98                      end
99              end
100
101             RX_DATA_BIT3: begin
102                 if (count==NUM_CLKS_PER_BIT-1)
103                     begin
104                         count<=0;
105                         dout[3]<=rx;
106                         state<=RX_DATA_BIT4;
107                         done<=0;
108                     end
109                 else
110                     begin
111                         count<=count+1;
112                     end
113             end
114
115             RX_DATA_BIT4: begin
116                 if (count==NUM_CLKS_PER_BIT-1)
117                     begin
118                         count<=0;
119                         dout[4]<=rx;
120                         state<=RX_DATA_BIT5;
121                         done<=0;
122                     end
123                 else
124                     begin
125                         count<=count+1;
126                     end
127             end
128
129             RX_DATA_BIT5: begin
130                 if (count==NUM_CLKS_PER_BIT-1)
131                     begin
132                         count<=0;
133                         dout[5]<=rx;
134                         state<=RX_DATA_BIT6;
135                         done<=0;
136                     end
137                 else
```

```
                begin
                    count<=count+1;
                end
        end

        RX_DATA_BIT6: begin
            if (count==NUM_CLKS_PER_BIT-1)
                begin
                    count<=0;
                    dout[6]<=rx;
                    state<=RX_DATA_BIT7;
                    done<=0;
                end
            else
                begin
                    count<=count+1;
                end
        end

        RX_DATA_BIT7: begin
            if (count==NUM_CLKS_PER_BIT-1)
                begin
                    count<=0;
                    dout[7]<=rx;
                    state<=RX_STOP_BIT;
                    done<=0;
                end
            else
                begin
                    count<=count+1;
                end
        end

         RX_STOP_BIT: begin
            if (count==NUM_CLKS_PER_BIT-1)
                begin
                    count<=0;
                    state<=RX_IDLE;
                    done<=1;
                end
            else
                begin
                    count<=count+1;
                end
        end

        default: begin
            state<=RX_IDLE;
        end


    endcase
 end
end
endmodule: uart_rx
```

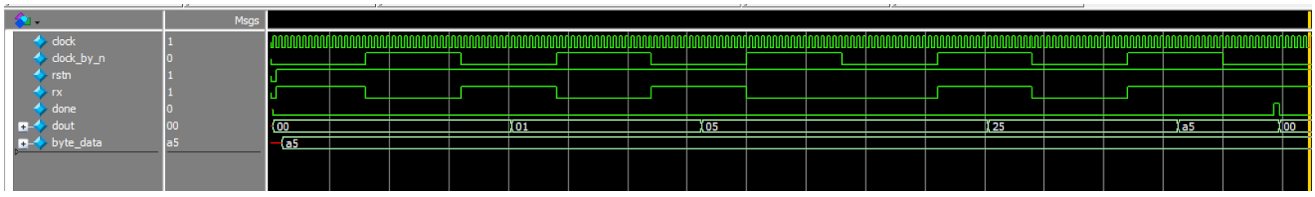
b) FSM diagram



c) Simulation waveform and explanations

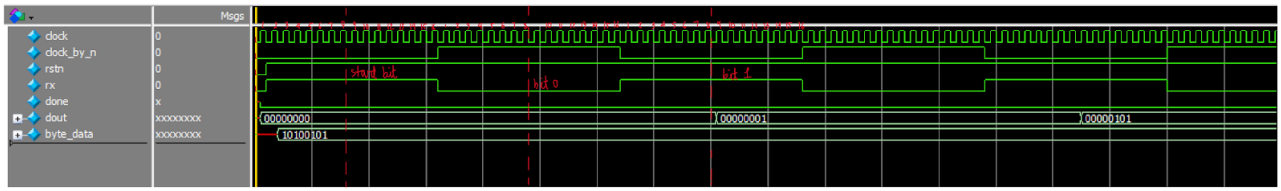


Close up waveform is below

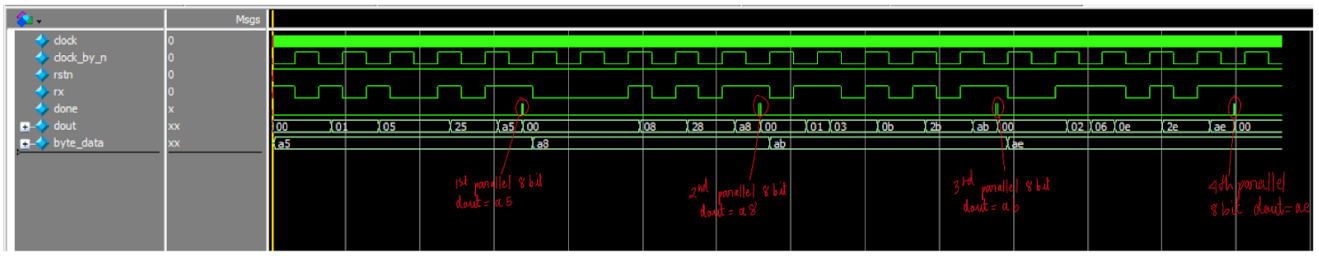


Explanations:

When the receiving UART detects the “start bit”, which is indicated when the serial incoming data “rx” goes from 1 to 0. When a “start bit” is detected, the UART begins to read the incoming bits. Since the UART doesn’t have a clock signal to synchronize the transmitter and receiver, the receiver uses its internal clock signal to sample the data in the middle of each bit period. In this case the receiver waits 8 clock cycles to establish a sampling point at the middle of the start bit, then another 16 clock cycles to bring it at the middle of the first data bit “bit 0” as shown below.



When all 8 bits are sampled and stored, “done” is high to indicate that the serial data is converted into parallel data and available at “dout”. A “stop bit” is signaled to put the sampling of data at the “idle” state again. The “dout” of the receiver should match the converted parallel bytes.



Resource summary and transcript

	Resource	Usage
1	Estimated ALUTs Used	32
1	-- Combinational ALUTs	32
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	24
3		
4	Estimated ALUTs Unavailable	15
1	-- Due to unpartnered combinational logic	15
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	32
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	1
2	-- 6 input functions	14
3	-- 5 input functions	4
4	-- 4 input functions	9
5	-- <=3 input functions	4
8		
9	Combinational ALUTs by mode	
1	-- normal mode	31
2	-- extended LUT mode	1
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	47
12		
13	Total registers	24
1	-- Dedicated logic registers	24
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
15		
16	I/O pins	12
17		
18	DSP block 18-bit elements	0
19		
20	Maximum fan-out node	clk~input
21	Maximum fan-out	24
22	Total fan-out	237

	Resource	Usage
23	Average fan-out	2.96

- Number of ALUT: 32 (12 I/O pins)
- Number of functions: 32
    - 1 (7 input functions)
    - 14 (6 input functions)
    - 4 (5 input functions)
    - 9 (4 input functions)
    - 4 (3 input functions)
- Total registers: 24

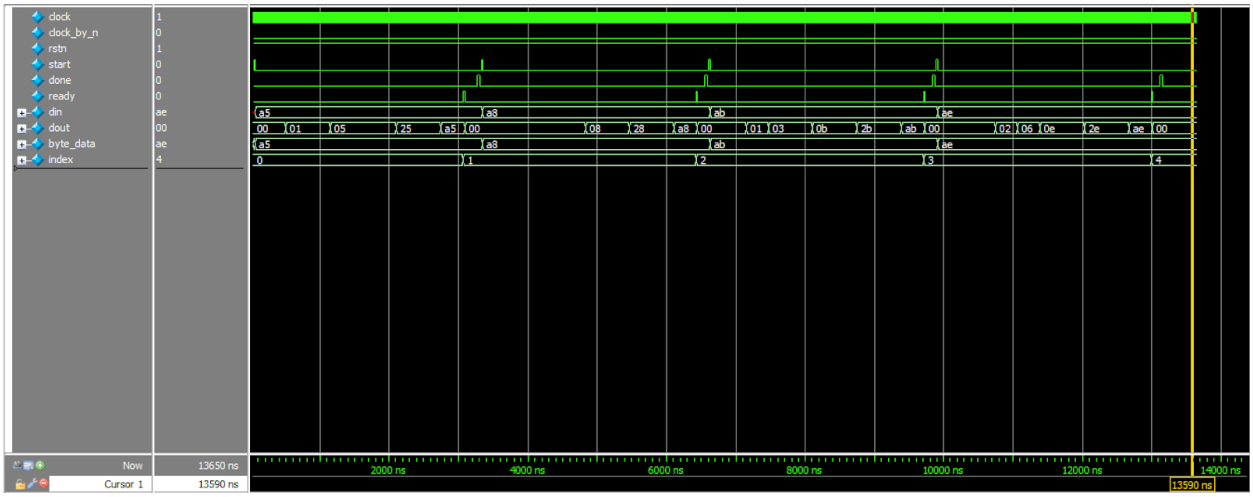
```
# Top level modules:
#       uart_rx_testbench
# End time: 01:35:11 on Mar 07,2021, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
ModelSim> vsim work.uart_rx_testbench
# vsim work.uart_rx_testbench
# Start time: 01:35:17 on Mar 07,2021
# Loading sv_std.std
# Loading work.uart_rx_testbench
# Loading work.uart_rx
add wave -position insertpoint sim:/uart_rx_testbench/*
VSIM 6> run -all
# Test Passed - Correct Byte Received time=                3200  expected=a5   actual=a5
# Test Passed - Correct Byte Received time=                6400  expected=a8   actual=a8
# Test Passed - Correct Byte Received time=                9600  expected=ab   actual=ab
# Test Passed - Correct Byte Received time=               12800  expected=ae   actual=ae
# ** Note: $finish    : //amznfsx7umcv4bw.AD.UCSD.EDU/share/users/ghtran/Desktop/Homework8/Homework8/Lab8/uart_top/uart_rx/uart_rx_testbench.sv(91)
#    Time: 13620 ns  Iteration: 0  Instance: /uart_rx_testbench
# 1
# Break in Module uart_rx_testbench at //amznfsx7umcv4bw.AD.UCSD.EDU/share/users/ghtran/Desktop/Homework8/Homework8/Lab8/uart_top/uart_rx/uart_rx_testbench.sv line 91
```

Homework 8b: Create Uart Tx-Rx communication system

a) **Design code**

```
1   // UART TX RTL Code
2   module uart_top #(parameter NUM_CLKS_PER_BIT=16)
3   (input logic tx_clk, tx_rstn, rx_clk, rx_rstn,
4    input logic[7:0] tx_din,
5    input logic tx_start,
6    output logic tx_done, rx_done,
7    output logic[7:0] rx_dout);
8
9
10  // wire to connect output of uart_tx "tx" signal to
11  // uart_rx "rx" signal
12  logic serial_data_bit;
13
14  // Instantiate uart transmitter module
15  uart_tx inst0 (
16      .clk(tx_clk),
17      .rstn(tx_rstn),
18      .start(tx_start),
19      .din(tx_din),
20      .tx(serial_data_bit),
21      .done(tx_done)
22  );
23
24
25  // Instantiate uart receiver module
26  uart_rx inst1 (
27      .clk(rx_clk),
28      .rstn(rx_rstn),
29      .rx(serial_data_bit),
30      .dout(rx_dout),
31      .done(rx_done)
32  );
33
34  endmodule: uart_top
35
36
37
```

b) Simulation waveform and explanations



Explanations:

The UART tx-rx system takes bytes of data and transits individual bits sequentially in which the output of the tx is fed into the input of the rx. Then the rx re-assembles the bits into complete parallel bytes. Thus, “din” the input of tx should match “byte_data” which is the converted 8 bit parallel data from the output of rx, “dout”. Notice that when every 8 bits is sampled and stored, indicated when “done=1”, the output of rx “dout” matches the byte value of tx and data bytes, this is how we know the datas have been sampled and matched correctly.

c) Resource summary and transcript

	Resource	Usage
1	Estimated ALUTs Used	52
1	-- Combinational ALUTs	52
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	38
3		
4	Estimated ALUTs Unavailable	23
1	-- Due to unpartnered combinational logic	23
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	52
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	3
2	-- 6 input functions	20
3	-- 5 input functions	10
4	-- 4 input functions	13
5	-- <=3 input functions	6
8		
9	Combinational ALUTs by mode	
1	-- normal mode	49
2	-- extended LUT mode	3
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	75
12		
13	Total registers	38
1	-- Dedicated logic registers	38
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
15		
16	I/O pins	23
17		
18	DSP block 18-bit elements	0
19		
20	Maximum fan-out node	rx_clk~input
21	Maximum fan-out	24
22	Total fan-out	382

	Resource	Usage
23	Average fan-out	2.81

- Number of ALUT: 52 (23 I/O pins)
- Number of functions: 52
    - 3 (7 input functions)
    - 20 (6 input functions)
    - 10 (5 input functions)
    - 13 (4 input functions)
    - 6 (3 input functions)
- Total registers: 38

```
# Top level modules:
#       uart_top_testbench
# End time: 21:53:19 on Mar 07,2021, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
ModelSim> vsim work.uart_top_testbench
# vsim work.uart_top_testbench
# Start time: 21:53:24 on Mar 07,2021
# Loading sv_std.std
# Loading work.uart_top_testbench
# Loading work.uart_top
# Loading work.uart_tx
# Loading work.uart_rx
add wave -position insertpoint sim:/uart_top_testbench/*
VSIM 6> run -all
# Test Passed - Correct Byte Received time=                3070  expected=a5   actual=a5
# Test Passed - Correct Byte Received time=                6430  expected=a8   actual=a8
# Test Passed - Correct Byte Received time=                9710  expected=ab   actual=ab
# Test Passed - Correct Byte Received time=               12990  expected=ae   actual=ae
# ** Note: $finish    : //amznfsx7umcv4bw.AD.UCSD.EDU/share/users/ghtran/Desktop/Homework8/Homework8/Lab8/uart_top/uart_top_testbench.sv(109)
#    Time: 13650 ns  Iteration: 0  Instance: /uart_top_testbench
# 1
# Break in Module uart_top_testbench at //amznfsx7umcv4bw.AD.UCSD.EDU/share/users/ghtran/Desktop/Homework8/Homework8/Lab8/uart_top/uart_top_testbench.sv line 109
```