Dennis Lang

ECE 111 Winter 2024

Professor John Eldon

10 March 2024

Lab 8

Part A: UART Receiver

design.sv | uart_rx.sv | uart_top.sv | uart_tx.sv

```
// UART RX RTL Code
module uart_rx #(parameter NUM_CLKS_PER_BIT=16)
(input        clk, rstn, 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  count <= count + 1;
        end
```

```
        RX_DATA_BIT0: begin
            // sample start bit value at mid-point
        // for each databit to get midpoint count value is 16
        // counting starts from midpoint of previous bit and ends at midpoint
        // of current data bit
          if(count == (NUM_CLKS_PER_BIT-1)) begin
            state   <= RX_DATA_BIT1;
            count   <= 0;
            dout[0] <= rx;
          end
          else
            count   <= count + 1;
        end
        RX_DATA_BIT1: begin
          if(count == (NUM_CLKS_PER_BIT-1)) begin
            state   <= RX_DATA_BIT2;
            count   <= 0;
            dout[1] <= rx;
          end
          else
            count   <= count + 1;
        end
        RX_DATA_BIT2: begin
          if(count == (NUM_CLKS_PER_BIT-1)) begin
            state   <= RX_DATA_BIT3;
            count   <= 0;
            dout[2] <= rx;
          end
          else
            count   <= count + 1;
        end
        RX_DATA_BIT3: begin
          if(count == (NUM_CLKS_PER_BIT-1)) begin
            state   <= RX_DATA_BIT4;
            count   <= 0;
            dout[3] <= rx;
          end
          else
            count   <= count + 1;
        end
        RX_DATA_BIT4: begin
          if(count == (NUM_CLKS_PER_BIT-1)) begin
            state   <= RX_DATA_BIT5;
            count   <= 0;
            dout[4] <= rx;
          end
          else
            count <= count + 1;
        end
        RX_DATA_BIT5: begin
          if(count == (NUM_CLKS_PER_BIT-1)) begin
            state   <= RX_DATA_BIT6;
            count   <= 0;
            dout[5] <= rx;
          end
          else
            count <= count + 1;
        end
        RX_DATA_BIT6: begin
          if(count == (NUM_CLKS_PER_BIT-1)) begin
            state   <= RX_DATA_BIT7;
            count   <= 0;
            dout[6] <= rx;
          end
          else
            count <= count + 1;
        end
        RX_DATA_BIT7: begin
          if(count == (NUM_CLKS_PER_BIT-1)) begin
            state   <= RX_STOP_BIT;
            count   <= 0;
            dout[7] <= rx;
            done    <= 0;
          end
```

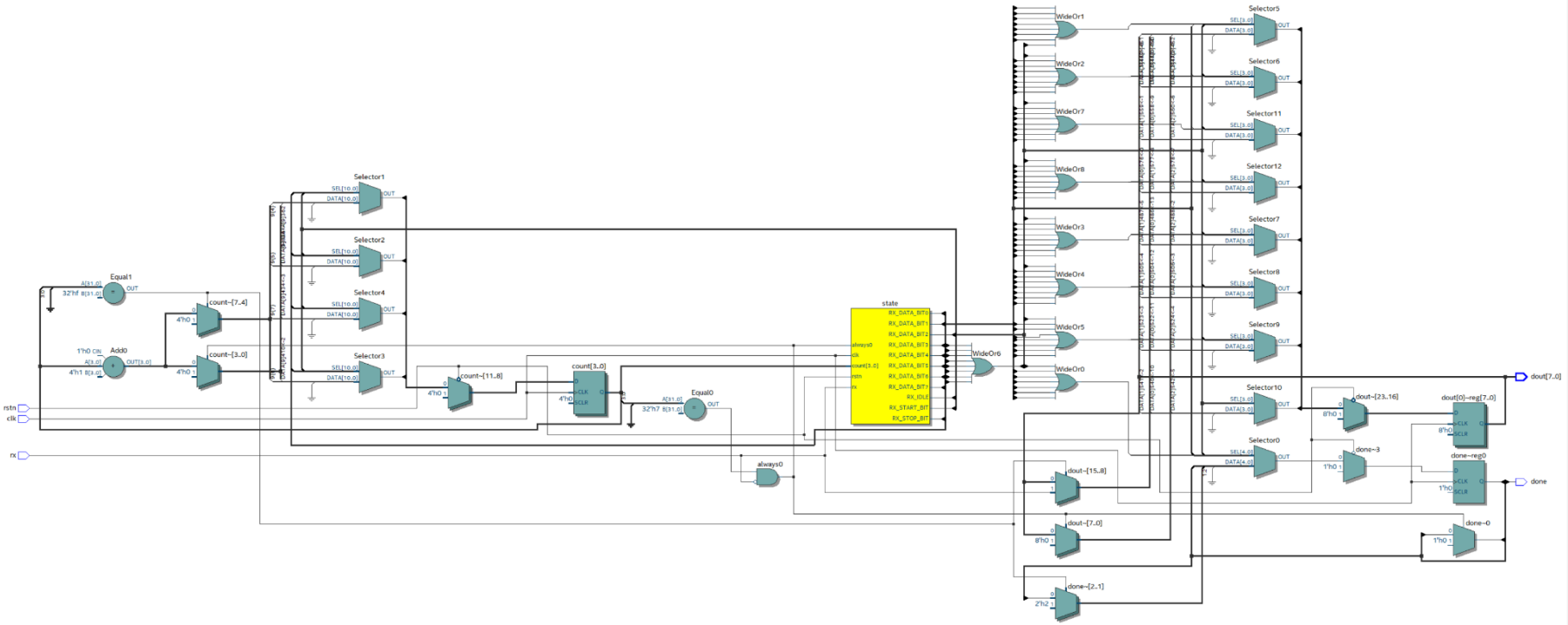
Part A: Synthesis Resource Usage + RTL Netlist Schematic

Compilation Report - uart_rx

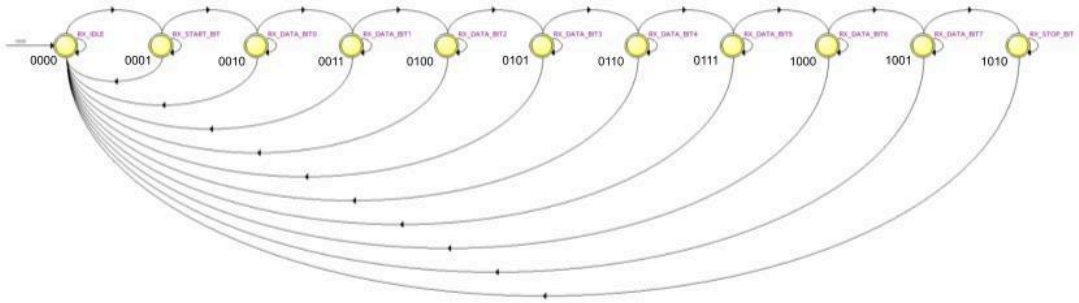
Analysis & Synthesis Resource Usage Summary

<<Filter>>

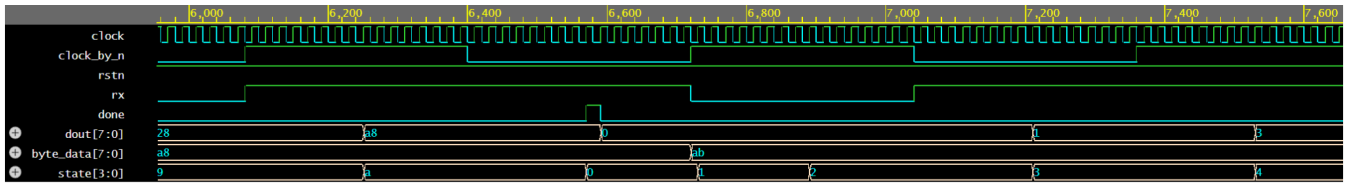
	Resource	Usage
1	Estimated ALUTs Used	30
1	-- Combinational ALUTs	30
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	24
3		
4	Estimated ALUTs Unavailable	14
1	-- Due to unpartnered combinational logic	14
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	30
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	1
2	-- 6 input functions	13
3	-- 5 input functions	4
4	-- 4 input functions	9
5	-- <=3 input functions	3
8		
9	Combinational ALUTs by mode	
1	-- normal mode	29
2	-- extended LUT mode	1
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	44
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	229
23	Average fan-out	2.94

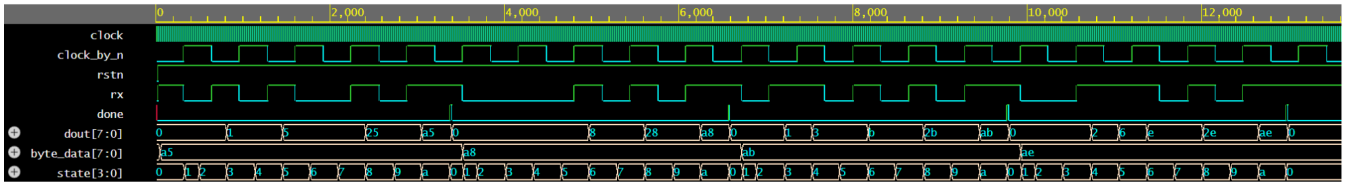

rstn
clk
rx
Equal1
32'hf
Add0
4'h1
count~[7..4]
count~[3..0]
Selector1
Selector2
Selector3
Selector4
count~[11..8]
count[3..0]
Equal0
32'h7
always0
state
RX_DATA_BIT0
RX_DATA_BIT1
RX_DATA_BIT2
RX_DATA_BIT3
RX_DATA_BIT4
RX_DATA_BIT5
RX_DATA_BIT6
RX_DATA_BIT7
RX_IDLE
RX_START_BIT
RX_STOP_BIT
WideOr6
WideOr1
WideOr2
WideOr7
WideOr8
WideOr3
WideOr4
WideOr5
WideOr0
Selector5
Selector6
Selector11
Selector12
Selector7
Selector8
Selector9
Selector10
Selector0
dout~[23..16]
dout[0]~reg[7..0]
done~3
done~reg0
done~0
dout~[15..8]
dout~[7..0]
done~[2..1]
dout[7..0]
done

Part A: State Machine Diagram



Part A: Simulation + Explanation





We can see Moore FSM behavior when looking at the state machine diagram, and we can verify the UART receiver is working by the sampling of every 16 clocks for every rx bit, and the first half (7 bits) for the start bit behavior. Along with the output of our testbench, we can say this design meets the requirements when given the RTL diagram. We can see output relies on the states, like the state value in the waveform and not the input variable. There is some uncombined logic but that is to be expected when given the design constraints.

Part B: UART Receiver

design.sv | uart_rx.sv | uart_top.sv | uart_tx.sv

```
// UART TX RTL Code
module uart_top #(parameter NUM_CLKS_PER_BIT=16)
(input              tx_clk, tx_rstn, rx_clk, rx_rstn,
 input [7:0]        tx_din,
 input              tx_start,
 output logic       tx_done, rx_done,
 output logic[7:0] rx_dout);


// wire to connect output of uart_tx "tx" signal to
// uart_rx "rx" signal
wire serial_data_bit;

// Instantiate uart transmitter module
// student to add code
  uart_tx #(NUM_CLKS_PER_BIT) u_trans (
    .clk(tx_clk),
    .rstn(tx_rstn),
    .din(tx_din),
    .start(tx_start),
    .done(tx_done),
    .tx(serial_data_bit)
  );


// Instantiate uart receiver module
// student to add code
  uart_rx #(NUM_CLKS_PER_BIT) u_rec (
    .clk(rx_clk),
    .rstn(rx_rstn),
    .rx(serial_data_bit),
    .done(rx_done),
    .dout(rx_dout)
  );

endmodule: uart_top
```

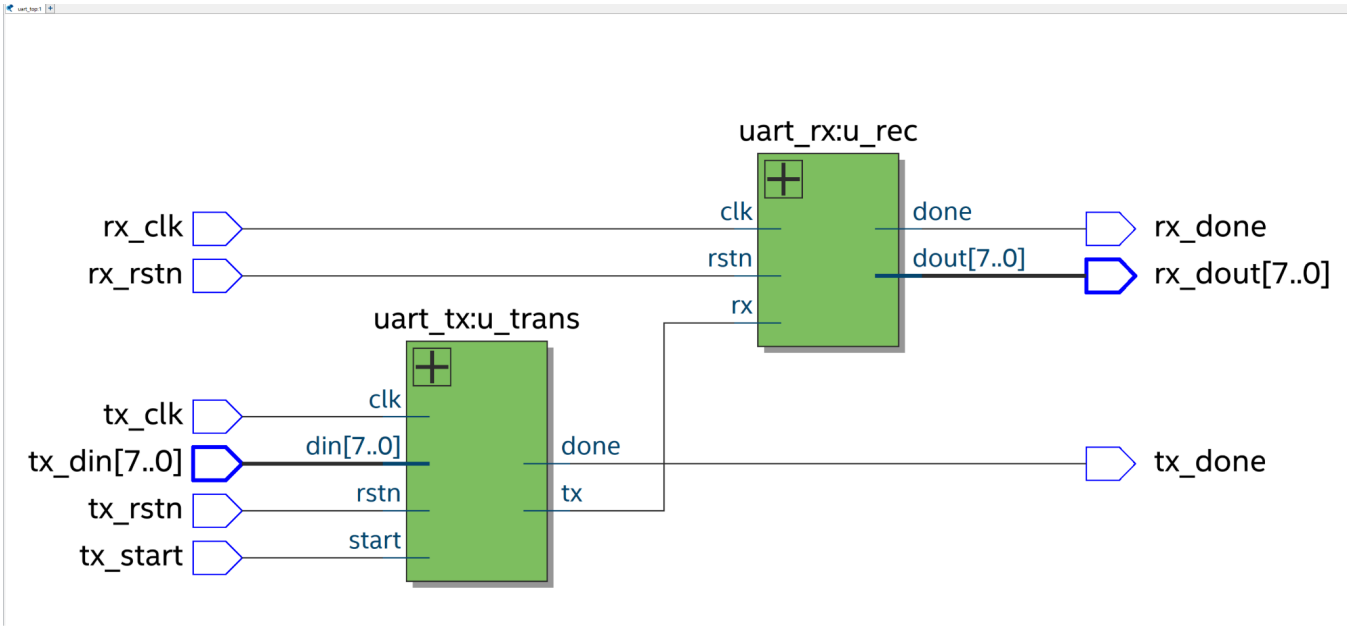
Part B: Synthesis Resource Usage + RTL Netlist Schematic

Compilation Report - uart_top

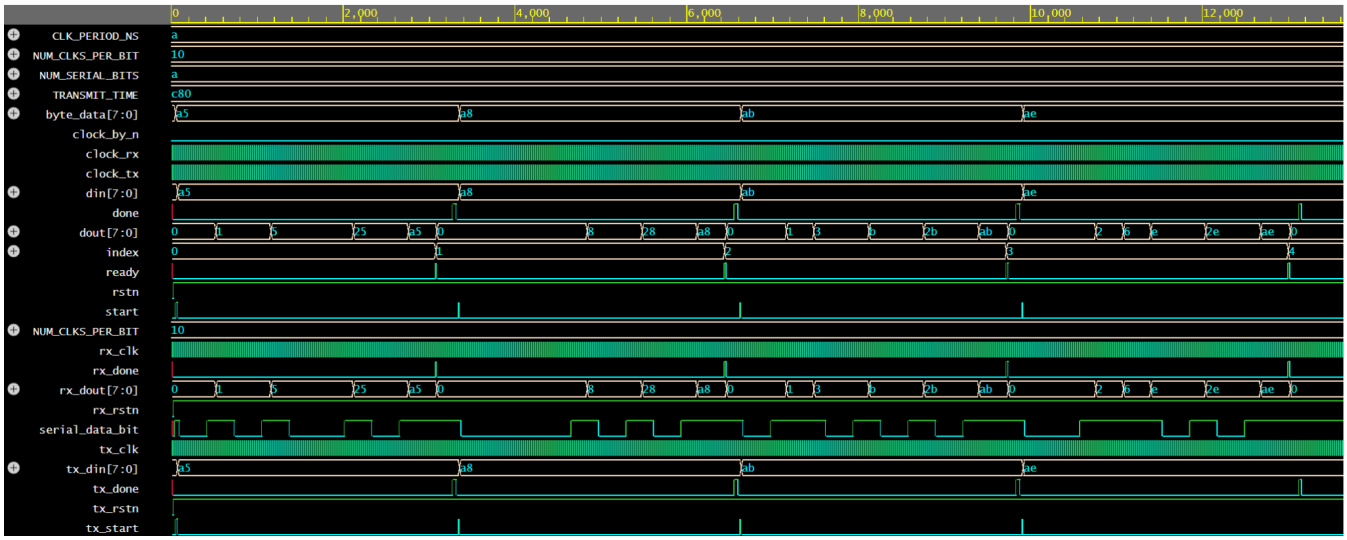
Analysis & Synthesis Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Estimated ALUTs Used	50
1	-- Combinational ALUTs	50
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	38
3		
4	Estimated ALUTs Unavailable	22
1	-- Due to unpartnered combinational logic	22
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	50
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	3
2	-- 6 input functions	19
3	-- 5 input functions	10
4	-- 4 input functions	13
5	-- <=3 input functions	5
8		
9	Combinational ALUTs by mode	
1	-- normal mode	47
2	-- extended LUT mode	3
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	72
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	374
23	Average fan-out	2.79


uart_rx:u_rec
rx_clk
rx_rstn
clk
rstn
rx
done
dout[7..0]
rx_done
rx_dout[7..0]
uart_tx:u_trans
tx_clk
tx_din[7..0]
tx_rstn
tx_start
clk
din[7..0]
rstn
start
done
tx
tx_done

Part B: Simulation + Explanation



We know the value is transferred properly between the two UARTs, the transmitter and the receiver based on the value of byte_data and dout above. Given the other waves match up and the output from the testbench below matches up, we can also use our RTL netlist to verify the behavior we designed is indeed being displayed with the input RX and output TX.

```
# Loading sv_std.std
# Loading work.uart_top_testbench(fast)
#
# 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: $stop    : uart_top_testbench.sv(98)
#    Time: 13650 ns  Iteration: 0  Instance: /uart_top_testbench
# Break in Module uart_top_testbench at uart_top_testbench.sv line 98
# exit
# End time: 02:35:07 on Mar 11,2024, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
# *** Summary ********************************************
#     qrun: Errors:   0, Warnings:   0
#     vlog: Errors:   0, Warnings:   0
#     vopt: Errors:   0, Warnings:   0
#     vsim: Errors:   0, Warnings:   0
#   Totals: Errors:   0, Warnings:   0
Finding VCD file...
./dump.vcd
[2024-03-11 06:35:08 UTC] Opening EPWave...
Done
```