Myrelia I. Villa

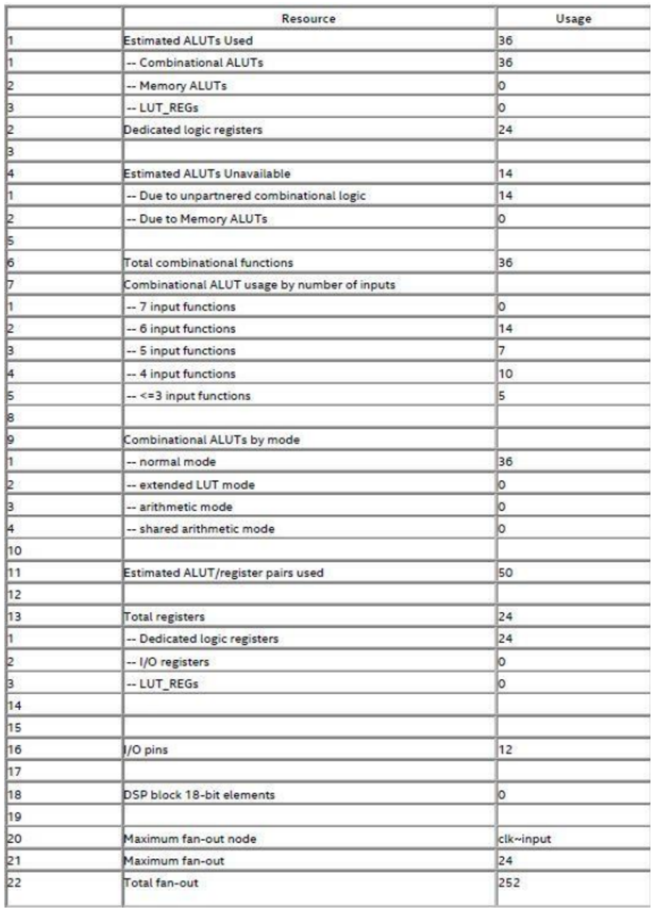
- Code

```
// 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
```

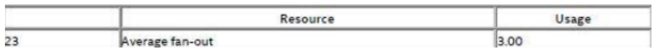
```
count <= count + 1;
end
end
RX_DATA_BIT0: begin
// sample start bit value at mid-point
// for each databit to get midpoint count value is 16
// counting starts from midpoint of previous bit and ends at midpoint
// of current data bit
// Student to fill rest of the code
if(count == (NUM_CLKS_PER_BIT-1)) begin
done <= 0;
state <= RX_DATA_BIT1;
count <= 0;
dout[0] <= rx;
end
else begin
state <= RX_DATA_BIT0;
count <= count + 1;
end
end
// Student to fill rest of the code for all remaining data bits and stop bit
RX_DATA_BIT1: begin
if(count == (NUM_CLKS_PER_BIT-1)) begin
done <= 0;
state <= RX_DATA_BIT2;
count <= 0;
dout[1] <= rx;
end
else begin
state <= RX_DATA_BIT1;
count <= count + 1;
end
end
RX_DATA_BIT2: begin
if(count == (NUM_CLKS_PER_BIT-1)) begin
done <= 0;
state <= RX_DATA_BIT3;
count <= 0;
dout[2] <= rx;
end
else begin
state <= RX_DATA_BIT2;
count <= count + 1;
end
```

```
end
RX_DATA_BIT3: begin
if(count == (NUM_CLKS_PER_BIT-1)) begin
done <= 0;
state <= RX_DATA_BIT4;
count <= 0;
dout[3] <= rx;
end
else begin
state <= RX_DATA_BIT3;
count <= count + 1;
end
end
RX_DATA_BIT4: begin
if(count == (NUM_CLKS_PER_BIT-1)) begin
done <= 0;
state <= RX_DATA_BIT5;
count <= 0;
dout[4] <= rx;
end
else begin
state <= RX_DATA_BIT4;
count <= count + 1;
end
end
RX_DATA_BIT5: begin
if(count == (NUM_CLKS_PER_BIT-1)) begin
done <= 0;
state <= RX_DATA_BIT6;
count <= 0;
dout[5] <= rx;
end
else begin
state <= RX_DATA_BIT5;
count <= count + 1;
end
end
RX_DATA_BIT6: begin
if(count == (NUM_CLKS_PER_BIT-1)) begin
done <= 0;
state <= RX_DATA_BIT7;
count <= 0;
dout[6] <= rx;
end
```

```
else begin
state <= RX_DATA_BIT6;
count <= count + 1;
end
end
RX_DATA_BIT7: begin
if(count == (NUM_CLKS_PER_BIT-1)) begin
done <= 0;
state <= RX_STOP_BIT;
count <= 0;
dout[7] <= rx;
end
else begin
state <= RX_DATA_BIT7;
count <= count + 1;
end
end
RX_STOP_BIT: begin
if(rx == 1 && count == (NUM_CLKS_PER_BIT -1)) begin
done <= 1;
state <= RX_IDLE;
count <= 0;
end
else begin
state <= RX_STOP_BIT;
count <= count + 1;
end
end
endcase
end
end
```

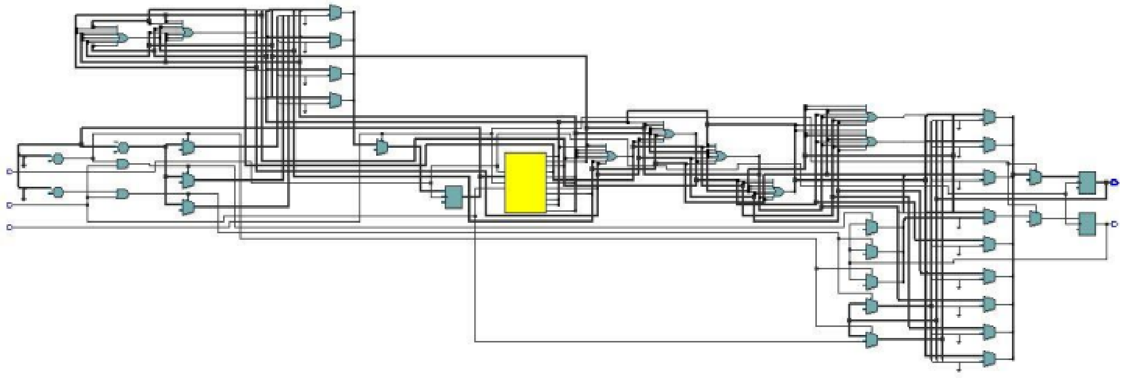


	Resource	Usage
1	Estimated ALUTs Used	36
1	-- Combinational ALUTs	36
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	24
3		
4	Estimated ALUTs Unavailable	14
1	-- Due to unpartnered combinational logic	14
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	36
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	14
3	-- 5 input functions	7
4	-- 4 input functions	10
5	-- <=3 input functions	5
8		
9	Combinational ALUTs by mode	
1	-- normal mode	36
2	-- extended LUT mode	0
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	50
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	252

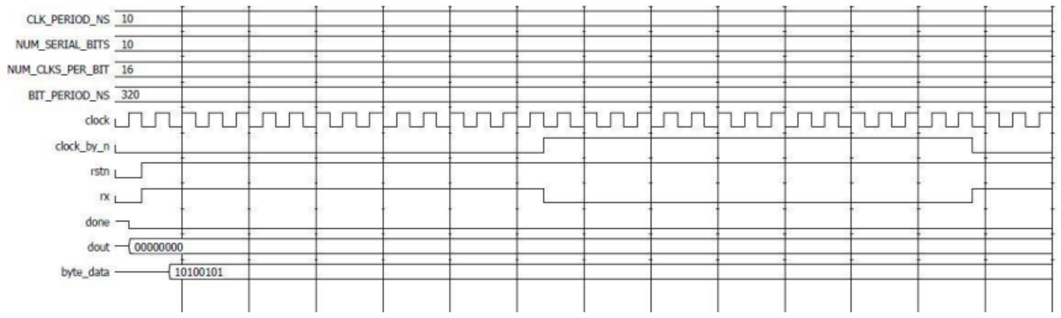


	Resource	Usage
23	Average fan-out	3.00

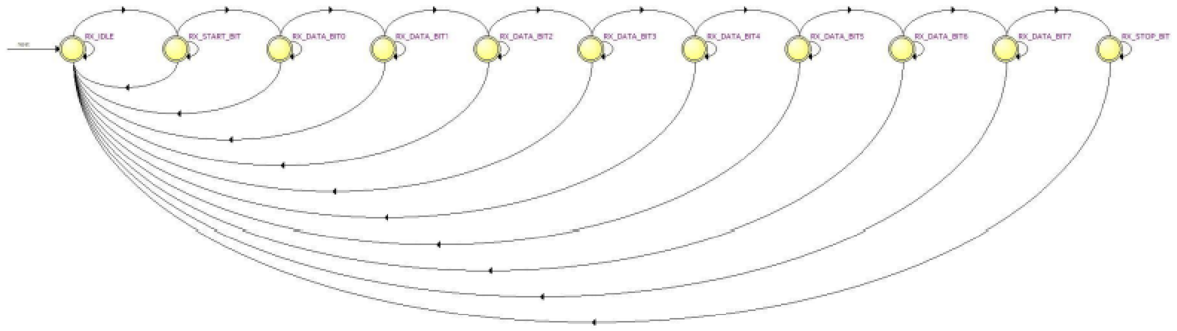
- resource usage
- RTL schematic

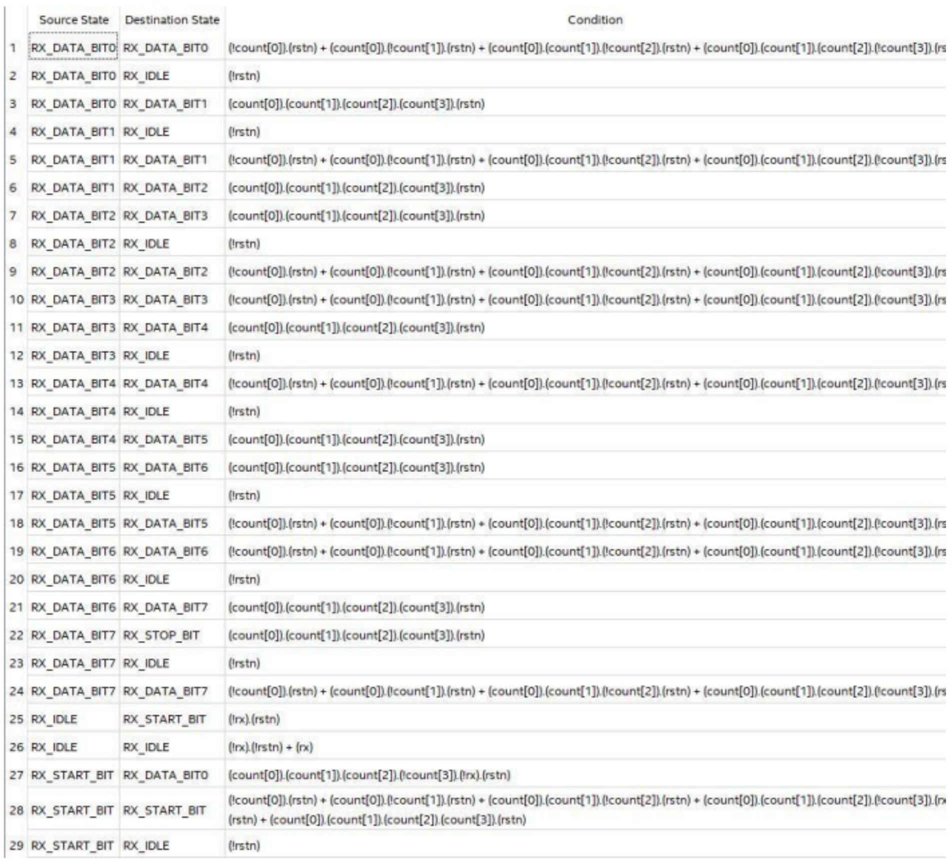


- Modelsim

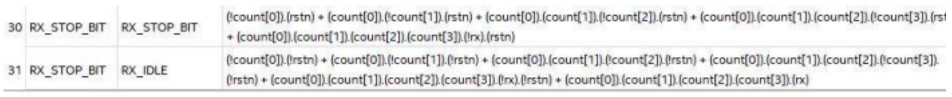


- FSM Diagram

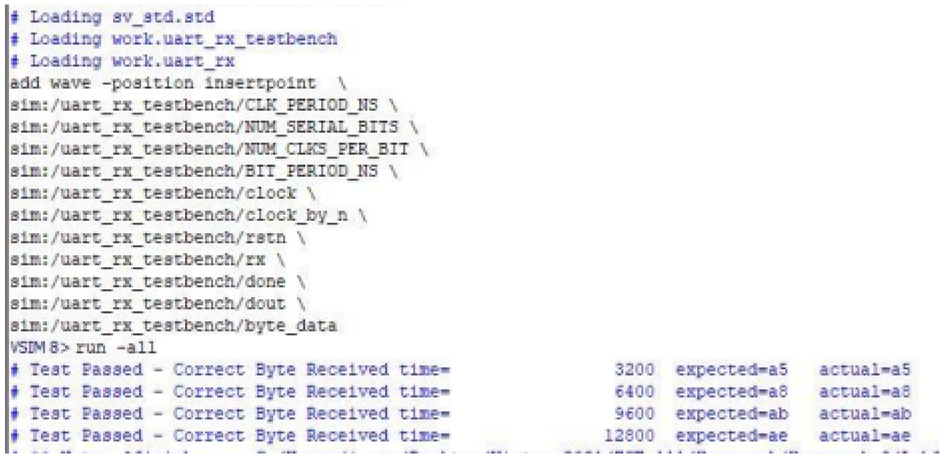




	Source State	Destination State	Condition
1	RX_DATA_BIT0	RX_DATA_BIT0	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rs
2	RX_DATA_BIT0	RX_IDLE	(!rstn)
3	RX_DATA_BIT0	RX_DATA_BIT1	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
4	RX_DATA_BIT1	RX_IDLE	(!rstn)
5	RX_DATA_BIT1	RX_DATA_BIT1	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rs
6	RX_DATA_BIT1	RX_DATA_BIT2	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
7	RX_DATA_BIT2	RX_DATA_BIT3	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
8	RX_DATA_BIT2	RX_IDLE	(!rstn)
9	RX_DATA_BIT2	RX_DATA_BIT2	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rs
10	RX_DATA_BIT3	RX_DATA_BIT3	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rs
11	RX_DATA_BIT3	RX_DATA_BIT4	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
12	RX_DATA_BIT3	RX_IDLE	(!rstn)
13	RX_DATA_BIT4	RX_DATA_BIT4	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rs
14	RX_DATA_BIT4	RX_IDLE	(!rstn)
15	RX_DATA_BIT4	RX_DATA_BIT5	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
16	RX_DATA_BIT5	RX_DATA_BIT6	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
17	RX_DATA_BIT5	RX_IDLE	(!rstn)
18	RX_DATA_BIT5	RX_DATA_BIT5	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rs
19	RX_DATA_BIT6	RX_DATA_BIT6	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rs
20	RX_DATA_BIT6	RX_IDLE	(!rstn)
21	RX_DATA_BIT6	RX_DATA_BIT7	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
22	RX_DATA_BIT7	RX_STOP_BIT	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
23	RX_DATA_BIT7	RX_IDLE	(!rstn)
24	RX_DATA_BIT7	RX_DATA_BIT7	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rs
25	RX_IDLE	RX_START_BIT	(!rx).(rstn)
26	RX_IDLE	RX_IDLE	(!rx).(!rstn) + (rx)
27	RX_START_BIT	RX_DATA_BIT0	(count[0]).(count[1]).(count[2]).(!count[3]).(!rx).(rstn)
28	RX_START_BIT	RX_START_BIT	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(r (rstn) + (count[0]).(count[1]).(count[2]).(count[3]).(rstn)
29	RX_START_BIT	RX_IDLE	(!rstn)
30	RX_STOP_BIT	RX_STOP_BIT	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rst + (count[0]).(count[1]).(count[2]).(count[3]).(!rx).(rstn)
31	RX_STOP_BIT	RX_IDLE	(!count[0]).(!rstn) + (count[0]).(!count[1]).(!rstn) + (count[0]).(count[1]).(!count[2]).(!rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(!rstn) + (count[0]).(count[1]).(count[2]).(count[3]).(!rx).(!rstn) + (count[0]).(count[1]).(count[2]).(count[3]).(rx)



-
- testbench

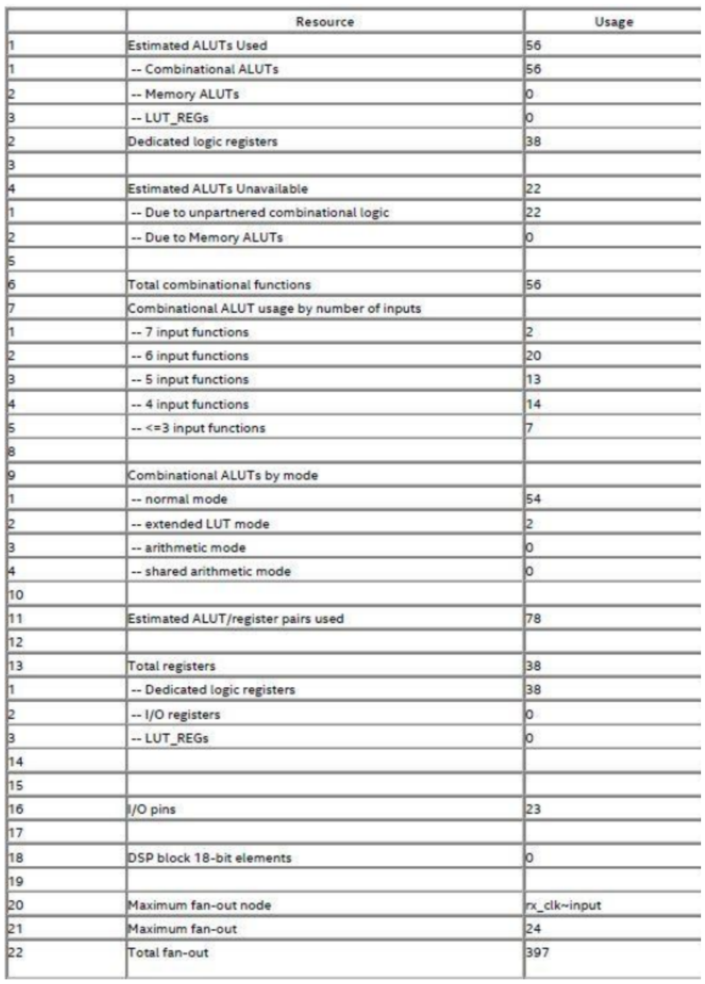


```
# Loading sv_std.std
# Loading work.uart_rx_testbench
# Loading work.uart_rx
add wave -position insertpoint  \
sim:/uart_rx_testbench/CLK_PERIOD_NS \
sim:/uart_rx_testbench/NUM_SERIAL_BITS \
sim:/uart_rx_testbench/NUM_CLKS_PER_BIT \
sim:/uart_rx_testbench/BIT_PERIOD_NS \
sim:/uart_rx_testbench/clock \
sim:/uart_rx_testbench/clock_by_n \
sim:/uart_rx_testbench/rstn \
sim:/uart_rx_testbench/rx \
sim:/uart_rx_testbench/done \
sim:/uart_rx_testbench/dout \
sim:/uart_rx_testbench/byte_data
VSIM 8> 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
```

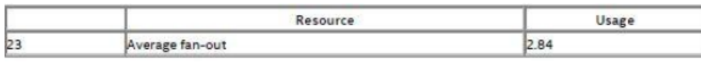
-

- Uart Tx-Rx Code

```
// UART TX RTL Code
module uart_top #(parameter NUM_CLKS_PER_BIT=16)
(input logic tx_clk, tx_rstn, rx_clk, rx_rstn,
input logic[7:0] tx_din,
input logic 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
logic serial_data_bit;
// Instantiate uart transmitter module
// student to add code
uart_tx #(.NUM_CLKS_PER_BIT(NUM_CLKS_PER_BIT)) inst1(
.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(NUM_CLKS_PER_BIT)) inst2(
.clk(rx_clk),
.rstn(rx_rstn),
.rx(serial_data_bit),
.done(rx_done),
.dout(rx_dout)
);
endmodule: uart_top
```

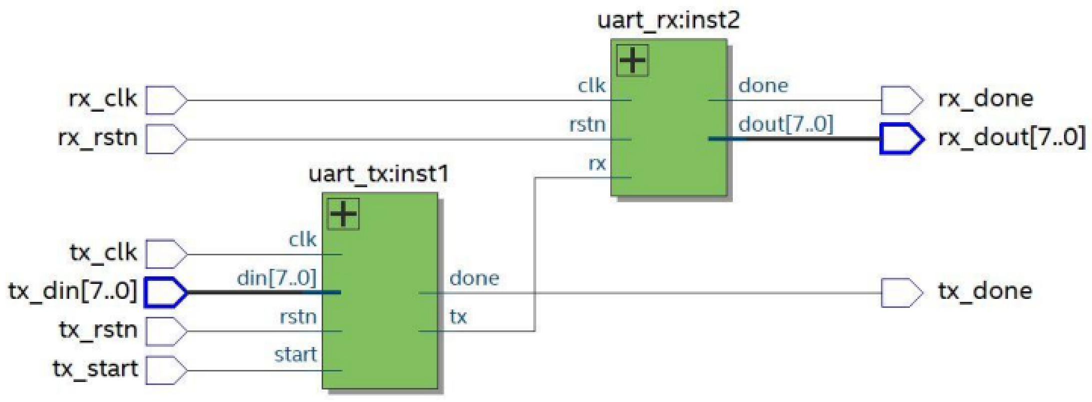


	Resource	Usage
1	Estimated ALUTs Used	56
1	-- Combinational ALUTs	56
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	38
3		
4	Estimated ALUTs Unavailable	22
1	-- Due to unpartnered combinational logic	22
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	56
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	2
2	-- 6 input functions	20
3	-- 5 input functions	13
4	-- 4 input functions	14
5	-- <=3 input functions	7
8		
9	Combinational ALUTs by mode	
1	-- normal mode	54
2	-- extended LUT mode	2
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	78
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	397

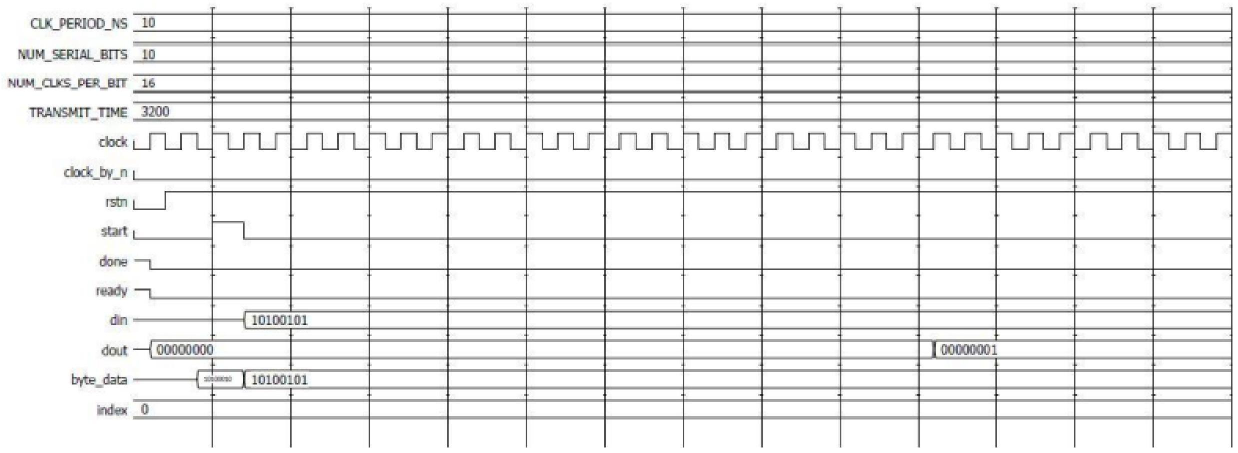


	Resource	Usage
23	Average fan-out	2.84

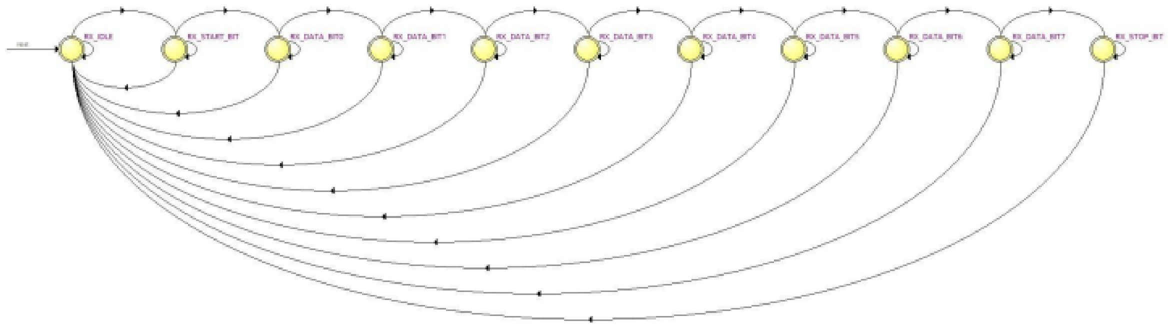
- Resource Usage
- RTL Schematic

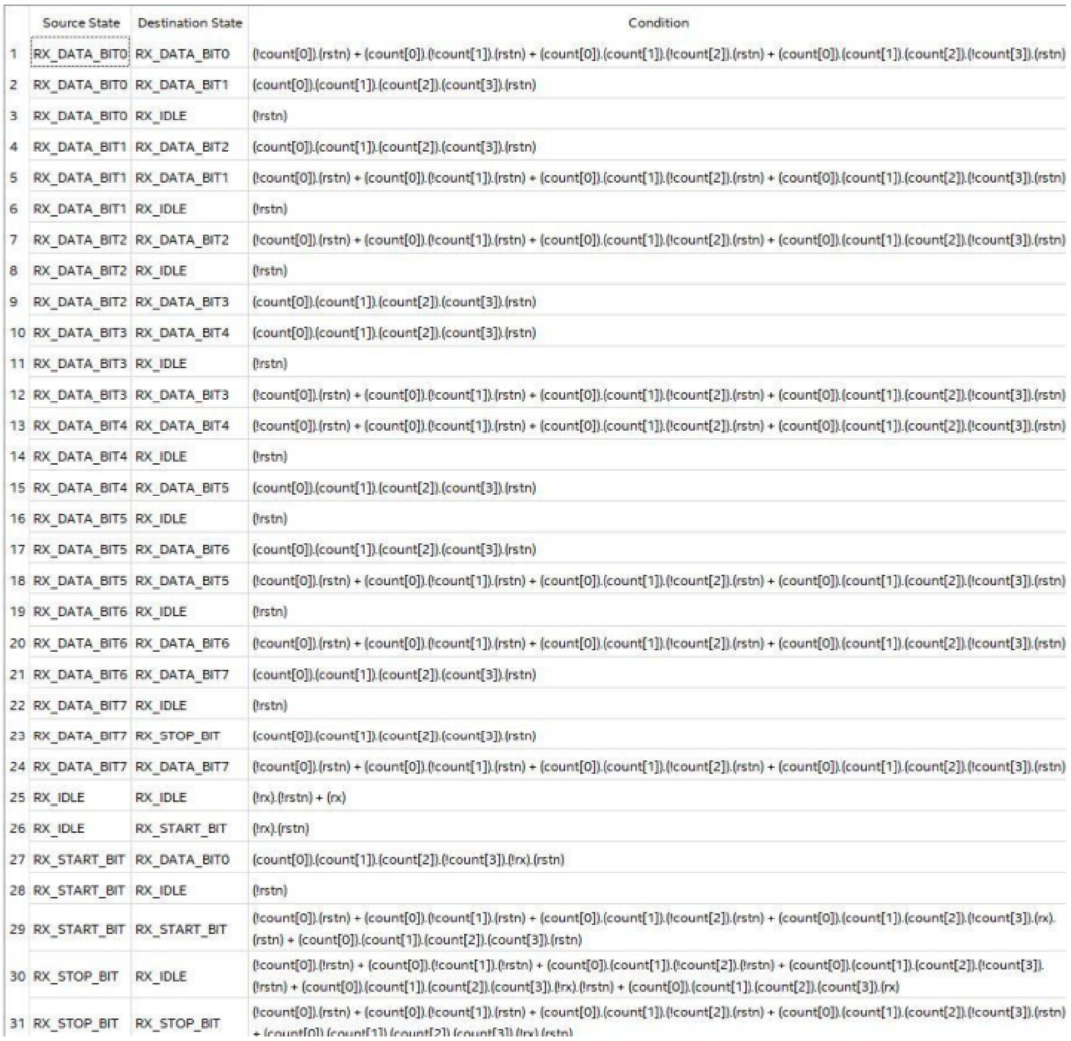


- Waveform

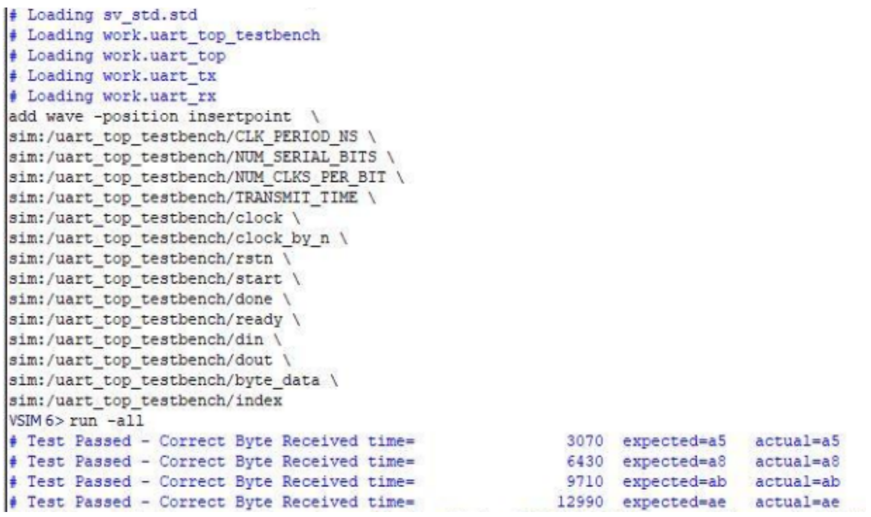


- FSM Diagram



- 

	Source State	Destination State	Condition
1	RX_DATA_BIT0	RX_DATA_BIT0	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
2	RX_DATA_BIT0	RX_DATA_BIT1	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
3	RX_DATA_BIT0	RX_IDLE	(!rstn)
4	RX_DATA_BIT1	RX_DATA_BIT2	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
5	RX_DATA_BIT1	RX_DATA_BIT1	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
6	RX_DATA_BIT1	RX_IDLE	(!rstn)
7	RX_DATA_BIT2	RX_DATA_BIT2	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
8	RX_DATA_BIT2	RX_IDLE	(!rstn)
9	RX_DATA_BIT2	RX_DATA_BIT3	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
10	RX_DATA_BIT3	RX_DATA_BIT4	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
11	RX_DATA_BIT3	RX_IDLE	(!rstn)
12	RX_DATA_BIT3	RX_DATA_BIT3	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
13	RX_DATA_BIT4	RX_DATA_BIT4	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
14	RX_DATA_BIT4	RX_IDLE	(!rstn)
15	RX_DATA_BIT4	RX_DATA_BIT5	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
16	RX_DATA_BIT5	RX_IDLE	(!rstn)
17	RX_DATA_BIT5	RX_DATA_BIT6	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
18	RX_DATA_BIT5	RX_DATA_BIT5	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
19	RX_DATA_BIT6	RX_IDLE	(!rstn)
20	RX_DATA_BIT6	RX_DATA_BIT6	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
21	RX_DATA_BIT6	RX_DATA_BIT7	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
22	RX_DATA_BIT7	RX_IDLE	(!rstn)
23	RX_DATA_BIT7	RX_STOP_BIT	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
24	RX_DATA_BIT7	RX_DATA_BIT7	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
25	RX_IDLE	RX_IDLE	(!rx).(!rstn) + (rx)
26	RX_IDLE	RX_START_BIT	(!rx).(rstn)
27	RX_START_BIT	RX_DATA_BIT0	(count[0]).(count[1]).(count[2]).(!count[3]).(!rx).(rstn)
28	RX_START_BIT	RX_IDLE	(!rstn)
29	RX_START_BIT	RX_START_BIT	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rx).(rstn) + (count[0]).(count[1]).(count[2]).(count[3]).(rstn)
30	RX_STOP_BIT	RX_IDLE	(!count[0]).(!rstn) + (count[0]).(!count[1]).(!rstn) + (count[0]).(count[1]).(!count[2]).(!rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(!rstn) + (count[0]).(count[1]).(count[2]).(count[3]).(!rx).(!rstn) + (count[0]).(count[1]).(count[2]).(count[3]).(rx)
31	RX_STOP_BIT	RX_STOP_BIT	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn) + (count[0]).(count[1]).(count[2]).(count[3]).(!rx).(rstn)

- Transcript 

```
# 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/CLK_PERIOD_NS \
sim:/uart_top_testbench/NUM_SERIAL_BITS \
sim:/uart_top_testbench/NUM_CLKS_PER_BIT \
sim:/uart_top_testbench/TRANSMIT_TIME \
sim:/uart_top_testbench/clock \
sim:/uart_top_testbench/clock_by_n \
sim:/uart_top_testbench/rstn \
sim:/uart_top_testbench/start \
sim:/uart_top_testbench/done \
sim:/uart_top_testbench/ready \
sim:/uart_top_testbench/din \
sim:/uart_top_testbench/dout \
sim:/uart_top_testbench/byte_data \
sim:/uart_top_testbench/index
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
```