San Francisco Bay University

EE461 Digital Design and HDL
Week#8 UART Design

1. Lab Outlines

- Introduction to UART
- Transmitter Design
- Receiver Design

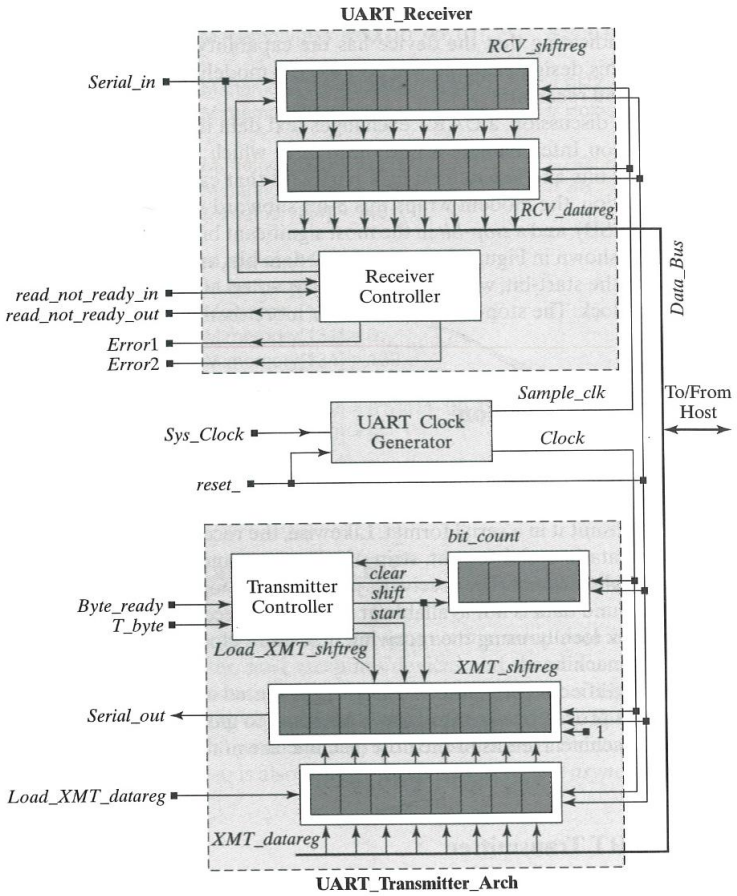
2. Lab Procedures

I. Introduction:

UART: Universal asynchronous receiver/transmitter. UARTs are commonly used in conjunction with communication standards such as EIA RS-232, RS-422 or RS-485. Bit0-bit7 are ASCII code and "Parity" is for error check. UART will add "start" and "stop" bits to for communication with outside device.

Stop Bit	Parity Bit	Data Bit 6	Data Bit 5	Data Bit 4	Data Bit 3	Data Bit 2	Data Bit 1	Data Bit 0	Start Bit

II. UART Architecture:



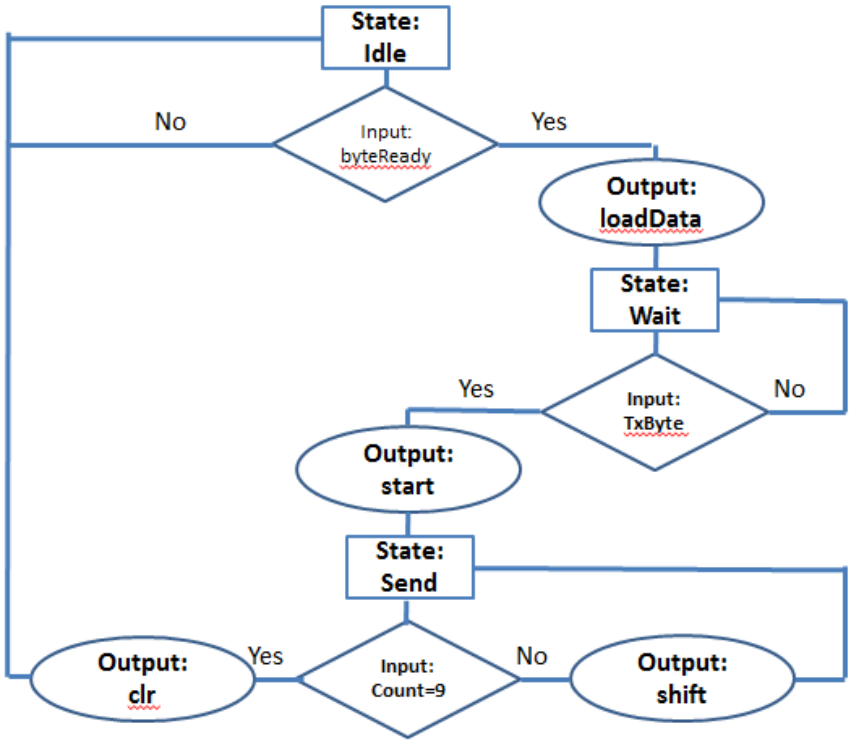
III. UART Transmitter Design

- UART Transmitter Signals:

- Input signals:
  1) Byte_ready from host computer is to indicate that the data in the bus is valid
  2) Load_XMT_datareg from host computer is to tell UART to load data in the bus
  3) T_byte from host computer is to ask UART to send data out.
  4) Data in the Bus
  5) Reset in UART
  6) Clock in UART

- Output signals:
  1) SeriesOut

- FSM's Algorithm diagram in Transmitter

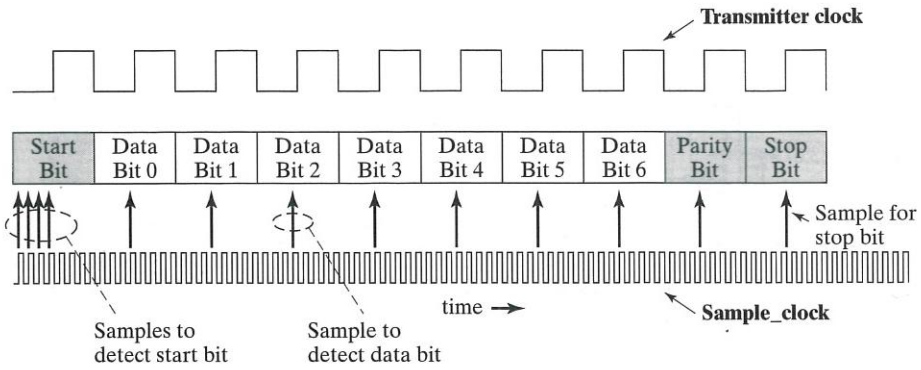


- Transmitter Verilog Code:

```
module UARTTx();
        declare variables;
        declare parameters for FSM;
        //FSM: Synchronous Sequential block & Combinational block
        always@(posedge clk)begin
                … …
        end
        always@(*)begin
                … …
        end
        //Implementation
        always@(posedge clk)begin
                … …
         end
endmodule
```

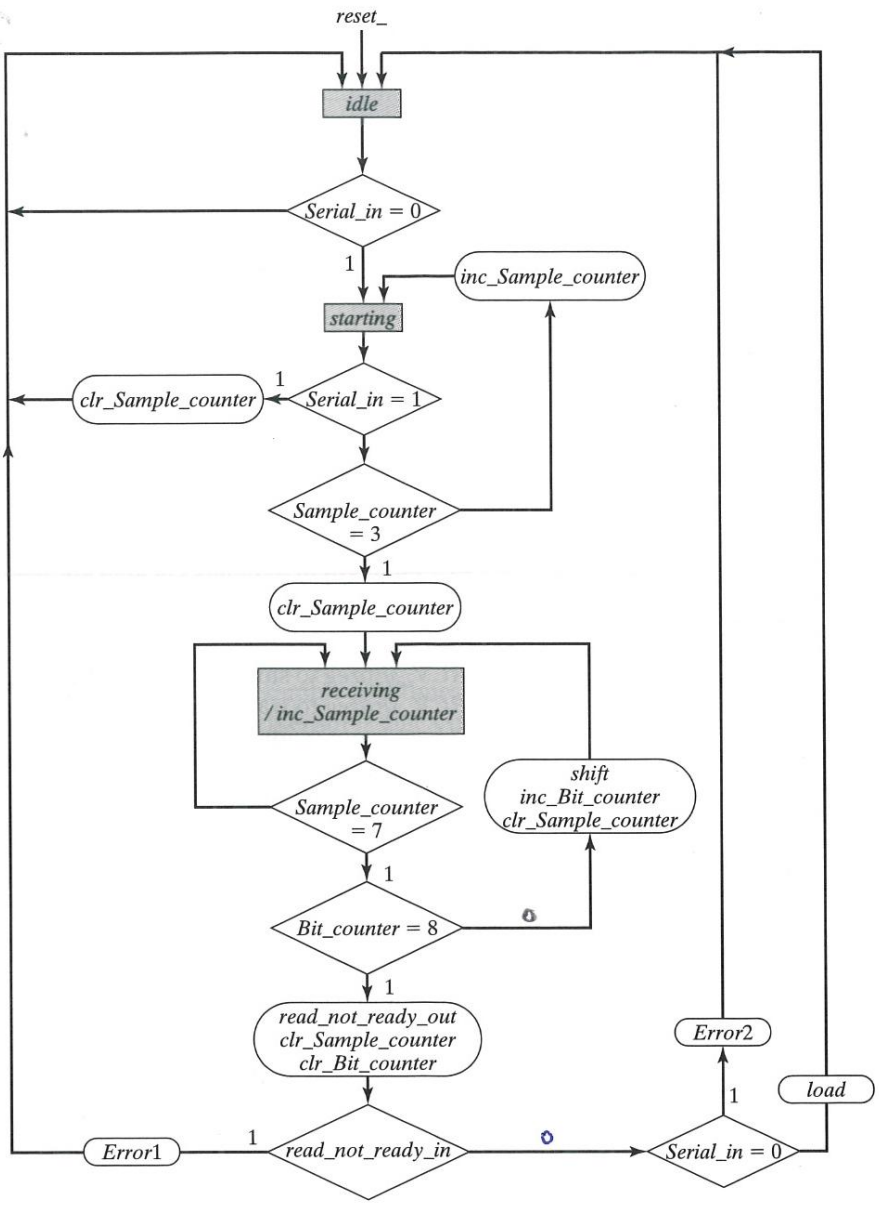
IV. UART Receiver Design

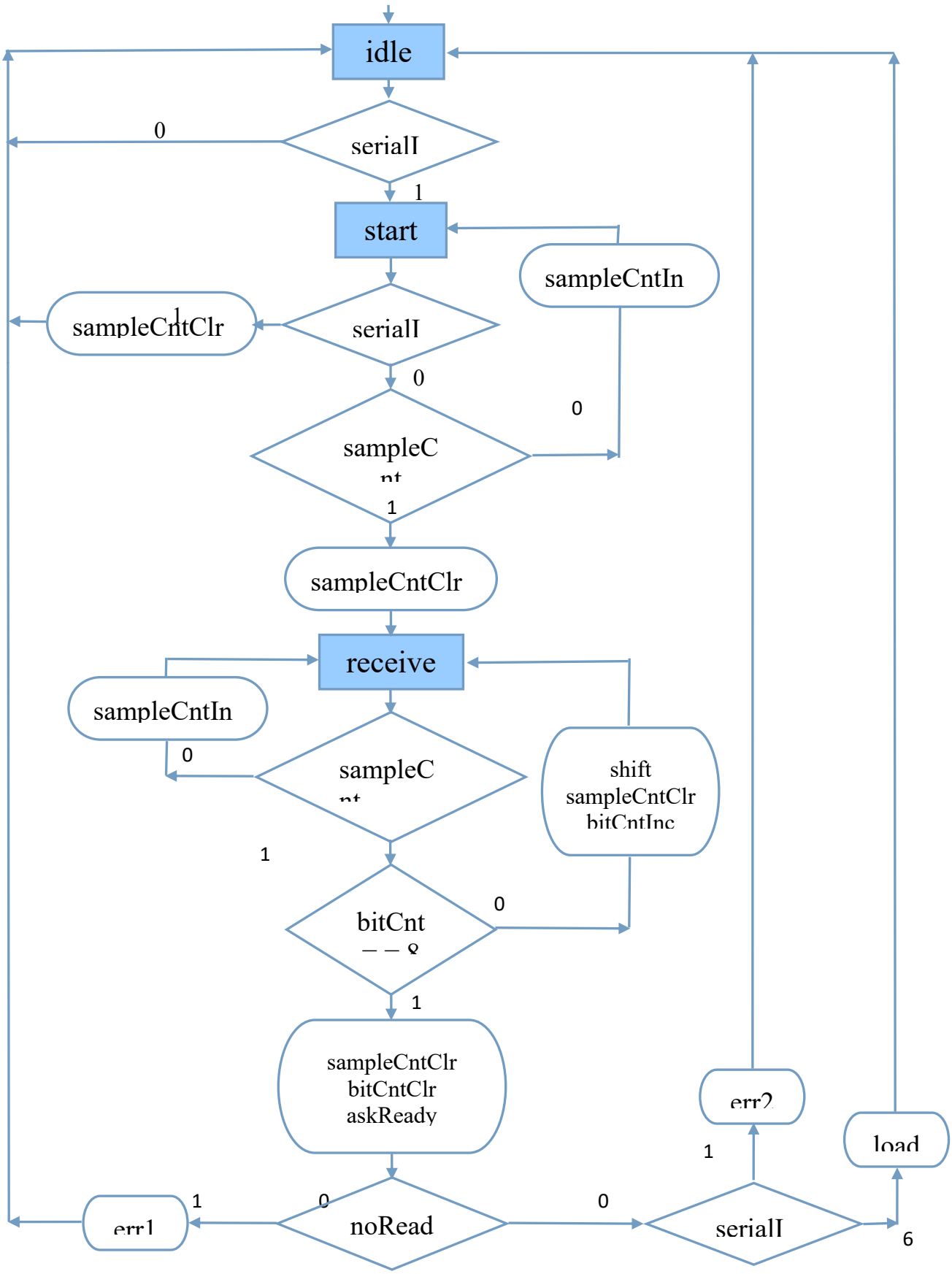
Data received by UART doesn't include clock signal. UART will use a higher frequency clock to sample each bit for input data. Usually, sampling frequency in receiver is 8 times frequency of input data.

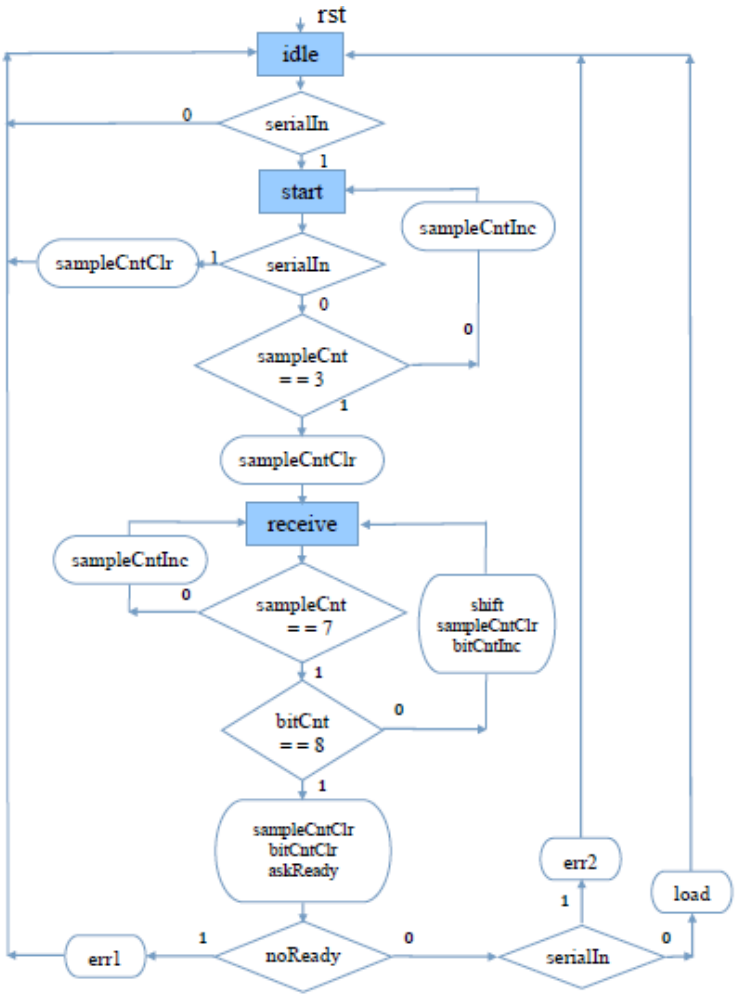


- Input signals
  1) notReady from host computer is to indicate that host is not ready to receive.
  2) serialIn from outside device
  3) sampleClk is 8 times frequency of transmitter clk
  4) rst
- Output signals
  1) dataOut is to send to host computer
  2) askReady is to ask if host computer is ready to receive or not
  3) err1 is to indicate the host is not ready to receive
  4) err2 is to send to outside device to indicate the missing stop bit

- FSM's algorithm diagram


reset_
idle
Serial_in = 0
1
inc_Sample_counter
starting
1
clr_Sample_counter
Serial_in = 1
Sample_counter = 3
1
clr_Sample_counter
receiving / inc_Sample_counter
shift inc_Bit_counter clr_Sample_counter
Sample_counter = 7
1
Bit_counter = 8
0
1
read_not_ready_out clr_Sample_counter clr_Bit_counter
Error2
1
load
1
Error1
read_not_ready_in
0
Serial_in = 0


idle
serialI
0
1
start
sampleCntIn
sampleCntClr
serialI
1
0
0
sampleC
nt
1
sampleCntClr
receive
sampleCntIn
0
sampleC
nt
shift
sampleCntClr
bitCntInc
1
bitCnt
0
1
sampleCntClr
bitCntClr
askReady
err2
load
1
err1
1
0
noRead
0
serialI
6

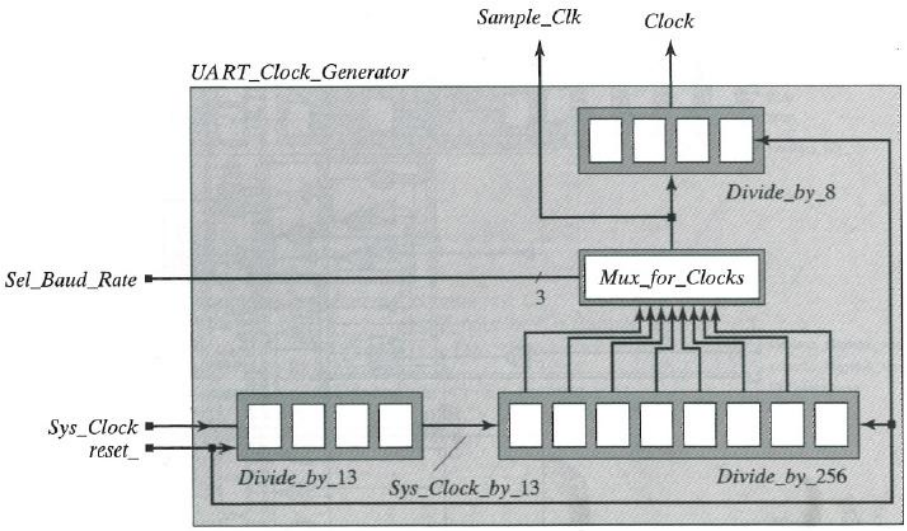


- Receiver Verilog Code

```
module UARTTx();
        declare variables;
        declare parameters for FSM;
        //FSM: Synchronous Sequential block & Combinational block
        always@(posedge clk)begin
                … …
        end
        always@(*)begin
                … …
        end
        //Implementation
        always@(posedge clk)begin
                … …
         end
endmodule
```

V. Clock Generator

In UART architecture diagram, there is clock generator block to output two clock signals sampleClk and Clk with the different frequencies, namely that sampleClk is 8 times of Clk. Assuming systemClk is 8MHz, design this block based on baud rate in the following table.



selBaud	sampleClk	Clk
000	307,696	38462
001	153,838	19231
010	76,920	9615
011	38,464	4808
100	18,232	2404
101	9,616	1202
110	4,808	601
111	2,404	300.5

```
module Div13(
             sysClk,
             rst_i,
             sysClkDiv13_o
```

```
);
	input  sysClk;
	input  rst_i;
	output sysClkDiv13_o;

	reg[3:0] cnt_r;

	assign  sysClkDiv13_o = cnt_r[3];
	// wire Clk1 = (cnt_r = = 12);
	// wire Clk2 = (cnt_r > 6);
	always@(posedge sysClk)begin
		if(rst_i)begin
			cnt_r <=4'b0000;
		end
		else if(cnt_r= =4'd12)begin
			cnt_r <=4'd0;
		end
		else begin
			cnt_r <= cnt_r + 1;
		end
	end
endmodule
```

VI. Exercises

Complete UART receiver and transmitter RTL design modules and verify them in the testbenches.

testbench.sv

```
`timescale 1ns/1ps

module UART_Integration_tb;
    reg sys_clk, reset, byteReady;
    reg [7:0] tx_data_in;
    wire tx_serialOut, tx_ready;

    wire [7:0] rx_data_out;
    wire rx_ready, rx_error;

    UART_Transmitter transmitter (
        .sys_clk(sys_clk),
        .reset(reset),
        .data_in(tx_data_in),
        .byteReady(byteReady),
        .serialOut(tx_serialOut),
        .ready(tx_ready)
    );

    UART_Receiver receiver (
        .sys_clk(sys_clk),
        .reset(reset),
        .serialIn(tx_serialOut),
        .data_out(rx_data_out),
        .ready(rx_ready),
        .error(rx_error)
    );

    always #5 sys_clk = ~sys_clk;

    initial begin
        $dumpfile("waveform.vcd");
        $dumpvars(0, UART_Integration_tb);

        sys_clk = 0;
        reset = 1;
        byteReady = 0;
        tx_data_in = 8'hA5;

        #20 reset = 0;

        #10 byteReady = 1;
        #10 byteReady = 0;

        wait (rx_ready);

        if (rx_data_out == tx_data_in) begin
            $display("Test Passed: Data received
correctly - %h", rx_data_out);
        end else begin
            $display("Test Failed: Expected %h but
got %h.", tx_data_in, rx_data_out);
        end

        #100 $finish;
    end
endmodule
```

design.sv

```
`timescale 1ns/1ps

// UART Transmitter
module UART_Transmitter (
    input wire sys_clk,
    input wire reset,
    input wire [7:0] data_in,
    input wire byteReady,
    output reg serialOut,
    output reg ready
);

    typedef enum logic [2:0] {IDLE, LOAD, START, SEND, STOP} state_t;
    state_t state;
    reg [8:0] shiftReg; // Start + 8 Data + Stop
    reg [3:0] bitCnt;

    always @(posedge sys_clk or posedge reset) begin
        if (reset) begin
            state <= IDLE;
            serialOut <= 1;
            ready <= 1;
            bitCnt <= 0;
        end else begin
            case (state)
                IDLE: if (byteReady) begin state <= LOAD; ready <= 0; end
                LOAD: begin shiftReg <= {1'b1, data_in, 1'b0}; state <= START; end
                START: begin serialOut <= shiftReg[0]; shiftReg <= shiftReg >> 1; state <= SEND; bitCnt <= 1; end
                SEND: begin
                    serialOut <= shiftReg[0];
                    shiftReg <= shiftReg >> 1;
                    bitCnt <= bitCnt + 1;
                    if (bitCnt == 8) state <= STOP;
                end
                STOP: begin serialOut <= 1; ready <= 1; state <= IDLE; end
                default: state <= IDLE;
            endcase
        end
    end
endmodule

// UART Receiver
module UART_Receiver (
    input wire sys_clk,
    input wire reset,
    input wire serialIn,
    output reg [7:0] data_out,
    output reg ready,
    output reg error
);

    typedef enum logic [2:0] {IDLE, START, RECEIVE, STOP, ERROR} state_t;
    state_t state;
    reg [3:0] sampleCnt;
    reg [2:0] bitCnt;
    reg [7:0] shiftReg;

    parameter SAMPLE_POINT = 7;

    always @(posedge sys_clk or posedge reset) begin
        if (reset) begin
```

Log Share

```
[2024-12-19 19:21:53 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && urbuffer vvp a.out
VCD info: dumpfile waveform.vcd opened for output.
Execution interrupted or reached maximum runtime.
Exit code expected: 0, received: 137
Done
```