`timescale 1ns / 1ps

module Rs232\_Tx\_nbytes(clk,reset,tx\_start\_flag,data\_in,tx\_done\_flag,Rs232\_Txd);

// Parameters

parameter n = 8; // No of bytes

parameter N = 8; // No of bits

parameter mlb = 0; // mlb=0 MSB first, mlb=1 LSB first

parameter BAUD\_RATE = 16'h28B0; // Baud rate on which UART is working

input clk;

input reset;

input [(N\*n)-1:0] data\_in;

output reg tx\_start\_flag = 1'b0;

output reg tx\_done\_flag = 1'b0;

output wire Rs232\_Txd;

// Internal signals

reg [15:0] clk\_count = 0; // Clk\_count to match baud rate

reg [N-1:0] data\_buffer = (n\*N) - N; // Data buffer to send bit by bit

reg [N-1:0] data[n-1:0]; // Array to store n bytes of data

reg start = 1'b1; // To indicate start bit

reg [3:0] state = 4'b0000; // No of states

reg tx\_data; // To store tx data

reg [n-1:0] byte\_counter = n-1; // To track index of data bytes

reg [n-1:0] count; // To assign data\_in into data by using count index

always @(posedge clk) begin

if (reset == 1'b0)

begin

tx\_data <= 1'b1; // Initial tx\_data

state <= 4'b0000; // Initial IDLE state

clk\_count <= 0; // clk\_count is 0

start <= 1'b1; // start bit is 1

count <= n-1; // Initial count is n-1 to send nth data

tx\_start\_flag <= 1'b0;

tx\_done\_flag <= 1'b0;

end

else

begin

case(state)

4'b0000: // IDLE state

begin

if (start)

begin

state <= 4'b0001; // START state

// Toggle tx\_start\_flag on every byte transmission start

data[count] <= data\_in[(count\*N) +: N]; // Assign data\_in to data for transmission

if (count >= 0)

begin

count <= count-1; // Until count becomes 0 decrementing count

end

end

else

begin

state <= 4'b0000; // IDLE state

end

end

4'b0001: // START state

begin

start <= 1'b0; // Start is 0

tx\_data <= 0; // Start bit

tx\_start\_flag <= 1'b1;

if ((BAUD\_RATE - 1) == clk\_count) // If baud\_rate and clk\_count matches

begin

state <= 4'b0010; // TRANSMIT state

clk\_count <= 0;

if (mlb == 0) // mlb for selecting msb or lsb to transfer first

begin

data\_buffer <= N-1; // bit\_counter to get msb

end

else

begin

data\_buffer <= 0; // bit\_counter to get lsb

end

end

else

begin

state <= 4'b0001; // START state

clk\_count <= clk\_count + 1; // If baud\_rate is not matches with clk\_count increment until it matches

end

end

4'b0010: // TRANSMIT state

begin

tx\_start\_flag <= 1'b0;

tx\_data <= data[byte\_counter][data\_buffer]; // if mlb=0 sending MSB of nth byte data else if mlb=1 sending LSB of the nth byte

if ((BAUD\_RATE - 1) == clk\_count) // if baud\_rate and clk\_count matches

begin

if (mlb == 0)

begin

data\_buffer <= data\_buffer-1; // if MSB first data\_buffer is decremented

end

else

begin

data\_buffer <= data\_buffer + 1; // if LSB first data\_buffer is incremented

end

if (data\_buffer == 0 && mlb == 0) // if data\_buffer reaches to min length of data\_in

begin

state <= 4'b0011; // state goes to stop

end

else if (data\_buffer == N-1 && mlb == 1) // if data\_buffer reaches to maxlength of data\_in

begin

state <= 4'b0011; // state goes to stop

end

else

begin

state <= 4'b0010; // else state will remain in transmit

end

clk\_count <= 0;

end

else

begin

state <= 4'b0010; // TRANSMIT state // if baud\_rate and clk\_count not matches

clk\_count <= clk\_count + 1; // Increment clk\_count until it matches with baud\_rate

end

end

4'b0011: // STOP state

begin

tx\_start\_flag <= 0;

tx\_data <= 1'b1; // Stop bit

if ((BAUD\_RATE - 1) == clk\_count) // if baud\_rate and clk\_count matches

begin

state <= 4'b0100; // REPEAT state for sending next byte of data

clk\_count <= 0;

end

else

begin

state <= 4'b0011; // STOP state

clk\_count <= clk\_count + 1;

end

end

4'b0100: // REPEAT state

begin

state <= 4'b0000; // IDLE state

clk\_count <= 0;

start <= 1'b1; // Generate start bit for next byte

byte\_counter <= byte\_counter - 1; // Move to next byte

if (byte\_counter == 0)

begin

byte\_counter <= n-1; // Reset counter after transmitting all bytes

state <= 4'b1101;

end

end

4'b1101: // END state

begin

tx\_done\_flag <= 1'b1;

clk\_count <= 0; // After transmitting all bytes of data reset clk\_count to 0

end

default:

begin

state <= 4'b1101; // Default state

end

endcase // end of the case statements

end

end

assign Rs232\_Txd = tx\_data; // Assigning tx\_data to output Rs232\_Txd

endmodule // End of the Design

//rx reciver

`timescale 1ns / 1ps

module Rs232\_Rx\_nbytes(clk,reset,Rs232\_Rxd,data\_out);

// parameters//

parameter n = 8; // No of bytes

parameter N = 8; // No of bits

parameter mlb=0; // mlb=0 MSB first, mlb=1 LSB first

parameter BAUD\_RATE = 16'h28B0; // Baud rate on which UART is working /\*(100000000/9600)=10416 == 16'h28B0\*/

//Inputs//

input clk; // input clock

input reset;// input reset

input Rs232\_Rxd; // input serial rxd coming from txd

output reg [(N\*n)-1:0] data\_out = 0; // output parallel data\_out

// Define the internal registers of the module

reg [15:0] clk\_count = 0; // clock count to match baud rate

reg [N+1:0] shift\_reg[n-1:0]; // shift register for received data

reg start\_bit ; // start bit for UART

reg [3:0] state = 4'b0000; // states defining

reg [(N\*n)-1:0] bit\_counter=0 ; // counter for received bits

reg [n-1:0] byte\_counter = n-1; // byte\_counter to keep track of how many bytes are sent

reg [n-1:0] count; // count to get byte by byte data

// Define the clocked logic for the module

always @(posedge clk)

begin

if (reset == 1'b0) // during reset is 0, reception doesn't take place

begin

state <= 4'b0000; // state is 0 means IDLE

clk\_count <= 0; // clock count to match baud rate is 0

start\_bit <= 1; // start bit is low

if(mlb==0) // if MSB first

begin

count <= n-1; // Count to get byte by byte data

end

else // if LSB first

begin

count<=n; // Count to get byte by byte data

end

end

else

begin

case (state) // starting of case statements

4'b0000: // IDLE state

begin

if (start\_bit== 1) // start bit detected

begin

state <= 4'b0001; // START state

end

else

begin

state <= 4'b0000; // IDLE state

end

end

4'b0001: // START state

begin

if ((BAUD\_RATE - 1) == clk\_count) // if baud\_rate and clk\_count matches

begin

state <= 4'b0010; // RECEIVE state

clk\_count <= 0; //reset clk\_count

if(mlb==0) // mlb for selecting msb or lsb to transfer first

begin

bit\_counter<=N+1; // bit \_counter to get msb

end

else begin

bit\_counter<=0; // bit \_counter to get lsb

end

end

else

begin

state <=4'b0001; // START state

clk\_count <= clk\_count + 1;

end

end

4'b0010: // RECEIVE state

begin

if ((BAUD\_RATE - 1) == clk\_count) //if baud\_rate and clk\_count matches

begin

shift\_reg[byte\_counter][bit\_counter] <= Rs232\_Rxd; // if mlb=0 receiving MSB of nth byte data else if mlb=1 receiving LSB of the nth byte

if(mlb==0)

begin

bit\_counter <= bit\_counter -1; // if MSB first data\_buffer is decremented

end

else begin

bit\_counter <= bit\_counter +1; // if MSB first data\_buffer is incremented

end

if(mlb==0)

begin

data\_out[(count\*N) +: N]<= shift\_reg[byte\_counter][9:2]; // first assigning nth byte of data by removing start and stop bit

end

else begin

data\_out[(count\*N-1) -: N]<= shift\_reg[byte\_counter][7:0]; // first assigning nth byte of data by removing start and stop bit

end

if (bit\_counter ==0 &&mlb==0)begin // received all bits of one byte

state <= 4'b0100; // REPEAT state

end

else if( bit\_counter==N+1 && mlb==1) begin // received all bits of one byte

state <= 4'b0100; // REPEAT

end

else

begin

state <= 4'b0010; // RECEIVE state

end

clk\_count <= 0;

end

else

begin

state <= 4'b0010; // RECEIVE state

clk\_count <= clk\_count + 1;

end

end

4'b0100: // REPEAT state

begin

if(byte\_counter>=0) // until byte\_counter becomes 0

begin

if((count >= 0&& count<=n-1) &&mlb==0) // if count is greater than zero and less than max

begin

byte\_counter <= byte\_counter - 1;// byte\_counter is decremented to get next byte

bit\_counter<=N+1; // bit \_counter to get msb

count <= count-1; // count is decremented to get next byte

end

else if((count >= 0&& count<=n) && mlb==1)

begin

byte\_counter <= byte\_counter - 1;//byte\_counter is decremented to get next byte

bit\_counter<=0; // bit \_counter to get lsb

count <= count-1; // count is decremented to get next byte

end

clk\_count <= 0;

state<=4'b0010; // RECEIVE state

end

else

begin

byte\_counter <= n-1; // Reset counter after transmitting all bytes

state <= 4'b1101; // STOP state

end

end

4'b0011: // STOP state

begin

state <= 2'b0011; // STOP state

clk\_count <= 0;// After transmitting all bytes of data reset clk\_count to 0

end

endcase// end of the case statements

end

end

endmodule // End of the Design

controller//

`timescale 1ns / 1ps

module Rs232\_Controller(clk,reset,data\_in,start\_flag,data\_out,done\_flag);

parameter N=8;

parameter n=8;

input clk; // Input clk

input reset;// Input reset

input [(N\*n)-1:0] data\_in; // Input data\_in

output start\_flag;

output [(N\*n)-1:0] data\_out;//output data\_out

output done\_flag;

wire Rs232\_Txd,Rs232\_Rxd;

assign Rs232\_Rxd = Rs232\_Txd;

Rs232\_Tx\_nbytes tx(.clk(clk),

.reset(reset),

.tx\_start\_flag(start\_flag),

.data\_in(data\_in),

.tx\_done\_flag(done\_flag),

.Rs232\_Txd(Rs232\_Txd));

Rs232\_Rx\_nbytes Rx(.clk(clk),

.reset(reset),

.Rs232\_Rxd(Rs232\_Rxd),

.data\_out(data\_out));

endmodule // END of Module

test bench

//

`timescale 1ns / 1ps

module tb\_Rs232\_Controller;

parameter N = 8;

parameter n = 8;

reg clk, reset;

reg [(N \* n) - 1: 0] data\_in;

wire start\_flag;

wire done\_flag;

wire [(N \* n) - 1: 0] data\_out;

Rs232\_Controller uut (

.clk(clk),

.reset(reset),

.start\_flag(start\_flag),

.data\_in(data\_in),

.done\_flag(done\_flag),

.data\_out(data\_out)

);

initial

begin

#10 clk = 1'b0; reset = 1'b0;

data\_in = 64'b000010101010101000001010101010100000101010101010;

#50 reset = 1'b1;

end

always #0.005 clk = ~clk;

endmodule