**Q1.**

`timescale 1ns/1ps

// Top-Level UART Transmitter Module

module UARTTx(

input clk, rst, // Clock and reset

input load\_i, // Load signal

input [7:0] busData\_i, // 8-bit input data

input byteReady, TxByte, // Signals to indicate byte readiness and transmission trigger

output serial\_out // Serial output

);

// Internal signals

wire cnt\_9; // Counter reaches 9 signal

wire loadData; // Load data signal

wire start; // Start transmission signal

wire shift; // Shift signal

wire clr; // Clear signal

// Instantiate Counter

Counter uCnt (

.clk(clk),

.rst(rst),

.shift(shift),

.clr(clr),

.cnt\_9(cnt\_9)

);

// Instantiate Control logic (Ctl)

Ctl uCtl (

.clk(clk),

.rst(rst),

.byteReady(byteReady),

.TxByte(TxByte),

.cnt\_9(cnt\_9),

.loadData(loadData),

.start(start),

.shift(shift),

.clr(clr)

);

// Instantiate Load Register

LoadReg uLoadReg (

.clk(clk),

.rst(rst),

.load\_i(load\_i),

.loadData(loadData),

.start\_r(start),

.shift\_r(shift),

.busData\_i(busData\_i),

.serial\_out(serial\_out)

);

endmodule

// Counter Module

module Counter(

input clk, rst,

input shift, clr,

output cnt\_9

);

reg [3:0] cnt;

always @(posedge clk or posedge rst) begin

if (rst)

cnt <= 4'b0; // Reset counter

else if (clr)

cnt <= 4'b0; // Clear counter

else if (shift)

cnt <= cnt + 1; // Increment counter

end

assign cnt\_9 = (cnt == 9) ? 1'b1 : 1'b0; // Signal when counter reaches 9

endmodule

// Control (Ctl) Module

module Ctl(

input clk, rst,

input byteReady, TxByte, cnt\_9,

output reg loadData, start, shift, clr

);

parameter pIdle = 3'b001;

parameter pWait = 3'b010;

parameter pSend = 3'b100;

reg [2:0] curSt, nxtSt;

// State register

always @(posedge clk or posedge rst) begin

if (rst)

curSt <= pIdle; // Reset to idle state

else

curSt <= nxtSt; // Transition to next state

end

// Next state logic and output control

always @(\*) begin

// Default signal values

loadData = 1'b0;

start = 1'b0;

shift = 1'b0;

clr = 1'b0;

case (curSt)

pIdle: begin

if (byteReady) begin

nxtSt = pWait;

loadData = 1'b1; // Load data

end else

nxtSt = pIdle;

end

pWait: begin

if (TxByte) begin

nxtSt = pSend;

start = 1'b1; // Start transmission

end else

nxtSt = pWait;

end

pSend: begin

if (cnt\_9) begin

nxtSt = pIdle;

clr = 1'b1; // Clear counter after transmission

end else begin

nxtSt = pSend;

shift = 1'b1; // Shift data

end

end

default: nxtSt = pIdle;

endcase

end

endmodule

// Load Register (LoadReg) Module

module LoadReg(

input clk, rst,

input load\_i, loadData, start\_r, shift\_r,

input [7:0] busData\_i,

output serial\_out

);

reg [7:0] busReg\_r; // Data register for incoming bus data

reg [8:0] dataReg\_r; // Shift register for serial output

always @(posedge clk or posedge rst) begin

if (rst) begin

busReg\_r <= 8'b0; // Reset bus register

dataReg\_r <= 9'b111111111; // Idle state (all stop bits)

end else if (load\_i) begin

busReg\_r <= busData\_i; // Load data from bus

end else if (loadData) begin

dataReg\_r <= {busReg\_r, 1'b1}; // Append stop bit

end else if (start\_r) begin

dataReg\_r[0] <= 1'b0; // Start bit

end else if (shift\_r) begin

dataReg\_r <= {1'b1, dataReg\_r[8:1]}; // Shift data

end

end

assign serial\_out = dataReg\_r[0]; // Serial output from LSB

endmodule

**//tb**

`timescale 1ns/1ps

module UARTTx\_tb;

// Testbench signals

reg clk;

reg rst;

reg load\_i;

reg byteReady;

reg TxByte;

reg [7:0] busData\_i;

wire serial\_out;

// Instantiate the UARTTx module

UARTTx uut (

.clk(clk),

.rst(rst),

.load\_i(load\_i),

.busData\_i(busData\_i),

.byteReady(byteReady),

.TxByte(TxByte),

.serial\_out(serial\_out)

);

// Clock generation

initial clk = 0;

always #5 clk = ~clk; // 10 ns clock period

// Test sequence

initial begin

// Initialize inputs

rst = 1; // Assert reset

load\_i = 0;

byteReady = 0;

TxByte = 0;

busData\_i = 8'b0;

// Reset the system

#50; // Hold reset for sufficient time

rst = 0;

// Wait for the system to settle

#10;

// Load data into the UART transmitter

load\_i = 1;

busData\_i = 8'b10101010; // Example data

#10;

load\_i = 0;

// Indicate that a byte is ready to send

byteReady = 1;

#10;

byteReady = 0;

// Start the transmission

TxByte = 1;

#10;

TxByte = 0;

// Allow time for the transmission to complete

#300;

// Load another byte and repeat the process

load\_i = 1;

busData\_i = 8'b11001100; // New data

#10;

load\_i = 0;

byteReady = 1;

#10;

byteReady = 0;

TxByte = 1;

#10;

TxByte = 0;

// Allow time for the second transmission to complete

#300;

// End simulation

#50; // Optional additional delay before finish

$finish;

end

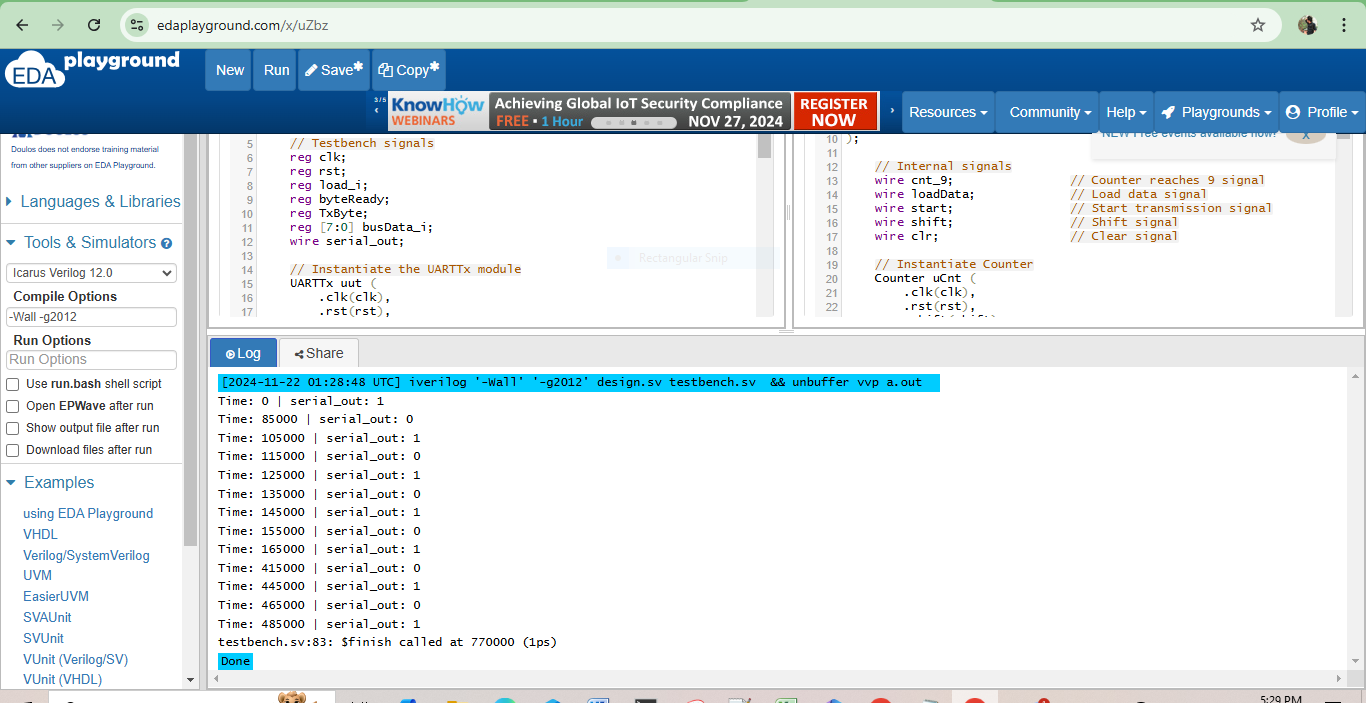
// Monitor outputs

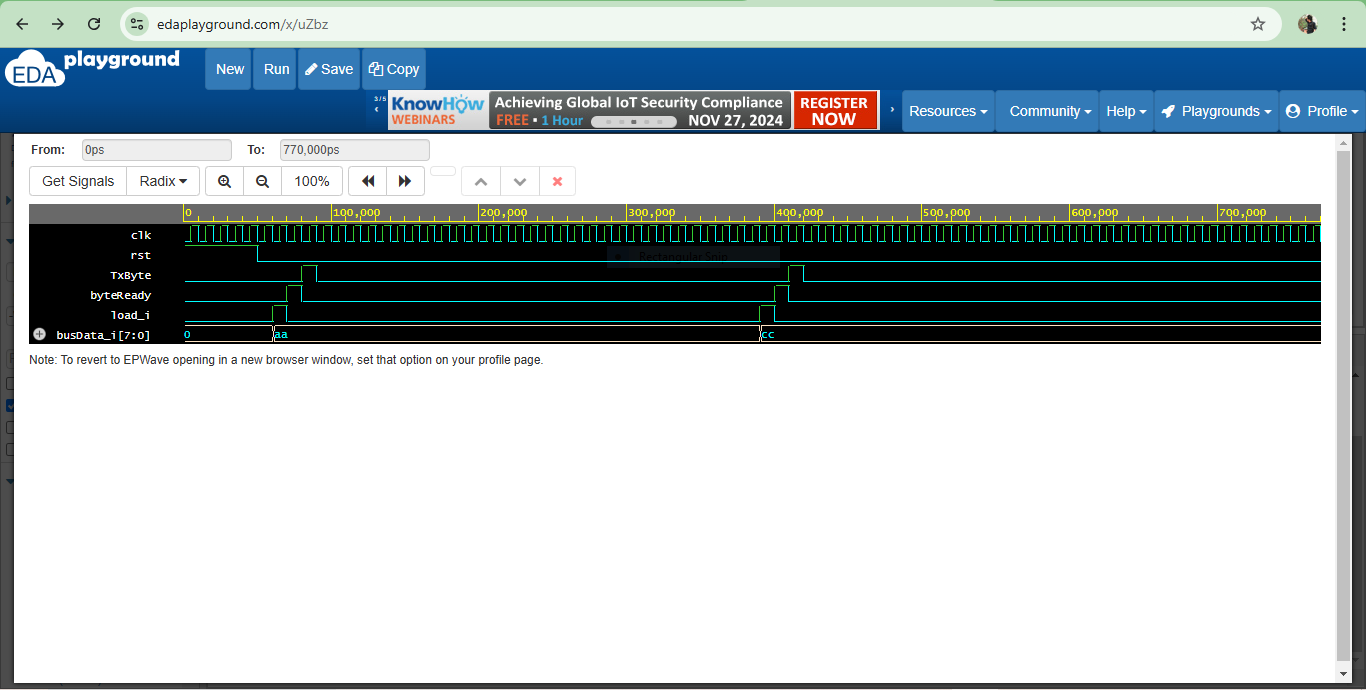
initial begin

$monitor("Time: %0t | serial\_out: %b", $time, serial\_out);

end

endmodule





**//Receiver**

`timescale 1ns/1ps

module rcv (

input clk, // System clock

input rst, // Active-high reset

input wire rx, // Serial input

output reg [7:0] rcv\_data, // Received data byte

output reg byte\_received // Byte received flag

);

// Internal signals

reg [3:0] bit\_cnt; // Bit counter

reg [7:0] rcv\_shiftreg; // Shift register for serial data

reg [3:0] sample\_cnt; // Sample counter for baud rate alignment

reg sample\_en; // Enable sampling

reg [1:0] state; // State machine for UART

// State encoding

localparam IDLE = 2'b00, START = 2'b01, DATA = 2'b10, STOP = 2'b11;

// Sampling Logic

always @(posedge clk or posedge rst) begin

if (rst) begin

sample\_cnt <= 0;

sample\_en <= 0;

end else if (sample\_cnt == 15) begin // Adjust for baud rate as needed

sample\_cnt <= 0;

sample\_en <= 1;

end else begin

sample\_cnt <= sample\_cnt + 1;

sample\_en <= 0;

end

end

// State Machine for UART Reception

always @(posedge clk or posedge rst) begin

if (rst) begin

state <= IDLE;

bit\_cnt <= 0;

rcv\_shiftreg <= 0;

rcv\_data <= 0;

byte\_received <= 0;

end else begin

case (state)

IDLE: begin

byte\_received <= 0;

if (~rx) begin // Start bit detected

state <= START;

$display("Time=%0t: START bit detected", $time);

end

end

START: begin

if (sample\_en) begin // Sample the start bit

if (~rx) begin

state <= DATA; // Valid start bit

bit\_cnt <= 0;

end else begin

state <= IDLE; // Invalid start bit, return to IDLE

end

end

end

DATA: begin

if (sample\_en) begin

rcv\_shiftreg <= {rx, rcv\_shiftreg[7:1]}; // Shift in serial data

$display("Time=%0t: Shift Register=%b", $time, rcv\_shiftreg);

if (bit\_cnt == 7) begin

state <= STOP; // All data bits received

end else begin

bit\_cnt <= bit\_cnt + 1;

end

end

end

STOP: begin

if (sample\_en) begin

if (rx) begin // Validate stop bit

rcv\_data <= rcv\_shiftreg; // Transfer received byte

byte\_received <= 1; // Indicate byte received

$display("Time=%0t: STOP bit valid, Data=%b", $time, rcv\_data);

end else begin

$display("Time=%0t: STOP bit error", $time);

end

state <= IDLE; // Return to IDLE state

end

end

endcase

end

end

endmodule

`timescale 1ns/1ps

module tb\_rcv;

// Inputs

reg clk, rst, rx;

// Outputs

wire [7:0] rcv\_data;

wire byte\_received;

// Instantiate the DUT (Device Under Test)

rcv dut (

.clk(clk),

.rst(rst),

.rx(rx),

.rcv\_data(rcv\_data),

.byte\_received(byte\_received)

);

// Clock generation

initial clk = 0;

always #5 clk = ~clk;

// Reset generation

initial begin

rst = 1;

#15 rst = 0;

end

// Stimulus generation

initial begin

rx = 1; // Idle state

// Wait for reset to complete

#20;

// Transmit Start Bit

rx = 0; #160;

// Transmit 8 Data Bits (10101010)

rx = 1; #160; // Bit 0

rx = 0; #160; // Bit 1

rx = 1; #160; // Bit 2

rx = 0; #160; // Bit 3

rx = 1; #160; // Bit 4

rx = 0; #160; // Bit 5

rx = 1; #160; // Bit 6

rx = 0; #160; // Bit 7

// Transmit Stop Bit

rx = 1; #160;

// Wait for processing

#500 $finish;

end

// Monitor outputs

initial begin

$monitor("Time=%0t: rx=%b, rcv\_data=%b, byte\_received=%b", $time, rx, rcv\_data, byte\_received);

end

// Waveform generation

initial begin

$dumpfile("rcv.vcd");

$dumpvars(0, tb\_rcv);

end

endmodule

