**Logo

Description automatically generated San Francisco Bay University**

**EE461 Digital Design and HDL**

**Week#8 UART Design**

**VI. Exercises**

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

**UART Receiver**

module UARTRx(

input wire clk,

input wire reset,

input wire rx,

output reg [7:0] data,

output reg valid

);

// Declare variables

reg [2:0] state;

reg [3:0] bitCounter;

reg [7:0] shiftReg;

// Declare parameters for FSM

parameter IDLE = 3'b000;

parameter START = 3'b001;

parameter RECEIVE = 3'b010;

parameter STOP = 3'b011;

// FSM: Synchronous Sequential block & Combinational block

always @(posedge clk, posedge reset) begin

if (reset) begin

state <= IDLE;

bitCounter <= 4'd0;

shiftReg <= 8'd0;

valid <= 1'b0;

end else begin

case(state)

IDLE: begin

if(!rx) begin

state <= START;

bitCounter <= 4'd1;

shiftReg <= 8'd0;

end

end

START: begin

shiftReg <= {shiftReg[6:0], rx};

bitCounter <= bitCounter + 1;

if(bitCounter == 4) begin

state <= RECEIVE;

bitCounter <= 4'd1;

end

end

RECEIVE: begin

shiftReg <= {shiftReg[6:0], rx};

bitCounter <= bitCounter + 1;

if(bitCounter == 8) begin

state <= STOP;

bitCounter <= 4'd1;

end

end

STOP: begin

shiftReg <= {shiftReg[6:0], rx};

bitCounter <= bitCounter + 1;

if(bitCounter == 9) begin

state <= IDLE;

bitCounter <= 4'd0;

data <= shiftReg[7:0];

valid <= 1'b1;

end

end

endcase

end

end

// Implementation

always @(\*) begin

case(state)

IDLE: begin

valid <= 1'b0;

end

START: begin

valid <= 1'b0;

end

RECEIVE: begin

valid <= 1'b0;

end

STOP: begin

valid <= 1'b0;

end

endcase

end

endmodule

**TestBench**

module UARTRx\_tb;

reg clk;

reg reset;

reg rx;

wire [7:0] data;

wire valid;

UARTRx uartrx(

.clk(clk),

.reset(reset),

.rx(rx),

.data(data),

.valid(valid)

);

initial begin

$dumpfile("UARTRx\_tb.vcd");

$dumpvars(0, UARTRx\_tb);

clk = 0;

reset = 1;

rx = 1;

#10 reset = 0;

#10 rx = 0;

#5 rx = 1;

#5 rx = 0;

#5 rx = 1;

#5 rx = 0;

#5 rx = 1;

#5 rx = 0;

#5 rx = 1;

#5 rx = 0;

#5 $finish;

end

always #5 clk = ~clk;

endmodule

**UART Transmitter**

module UARTTx(

clk,

rst\_i,

byteReady\_i,

load\_i,

TxByte\_i,

busData\_i,

serialOut\_o

);

input clk;

input rst\_i;

input byteReady\_i;

input load\_i;

input TxByte\_i;

input[7:0] busData\_i;

output serialOut\_o;

parameter pIdle=2'b00;

parameter pWait=2'b01;

parameter pSend=2'b10;

reg[1:0] curSt\_r;

reg[1:0] nxtSt\_r;

reg loadData\_r;

reg start\_r;

reg clr\_r;

reg shift\_r;

reg[3:0] cnt\_r;

reg[7:0] busReg\_r;

reg[8:0] dataReg\_r;

//Sequential Logic

always@(posedge clk)begin

if(rst\_i)begin

curSt\_r <= pIdle;

end

else begin

curSt\_r <= nxtSt\_r ;

end

end

// Combo Logic

always@(\*)begin

loadData\_r=1'b0;

start\_r=1'b0;

clr\_r=1'b0;

shift\_r=1'b0;

nxtSt\_r= curSt\_r;

case(curSt\_r)

pIdle: begin

if(byteReady\_i)begin

loadData\_r=1'b1;

nxtSt\_r=pWait;

end

end

pWait: begin

if(TxByte\_i)begin

start\_r = 1'b1;

nxtSt\_r= pSend;

end

end

pSend: begin

if(cnt\_r ==9)begin

clr\_r = 1'b1;

nxtSt\_r= pIdle;

end

else begin

shift\_r = 1'b1;

end

end

default: begin

nxtSt\_r= pIdle;

end

endcase

end

//Implementation block- bit counter

always@(posedge clk)begin

if(rst\_i)begin

cnt\_r<=4'b0000;

end

else if(clr\_r)begin

cnt\_r<=4'b0000;

end

else if(shift\_r)begin

cnt\_r<= cnt\_r + 1'b1;

end

end

always@(posedge clk)begin

if(rst\_i)begin

busReg\_r <=8'b0000\_0000;

dataReg\_r<=9'b1\_1111\_1111;

end

else if(load\_i)begin

busReg\_r <= busData\_i;

end

else if(loadData\_r)begin

dataReg\_r <= {busReg\_r, 1'b1};

end

else if(start\_r)begin

dataReg\_r[0]<=1'b0; //output startbit ==0;

end

else if(shift\_r)begin

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

end

end

assign serialOut\_o = dataReg\_r[0];

endmodule

**Testbench**

`timescale 1ns/10ps

module UARTTx\_tb;

// Inputs

reg clk;

reg rst\_i;

reg byteReady\_i;

reg load\_i;

reg TxByte\_i;

reg [7:0] busData\_i;

// Outputs

wire serialOut\_o;

// Instantiate the UARTTx module

UARTTx dut (

.clk(clk),

.rst\_i(rst\_i),

.byteReady\_i(byteReady\_i),

.load\_i(load\_i),

.TxByte\_i(TxByte\_i),

.busData\_i(busData\_i),

.serialOut\_o(serialOut\_o)

);

// Clock generator

always begin

clk = 1'b0;

#5;

clk = 1'b1;

#5;

end

// Stimulus

initial begin

$dumpfile("UARTTx\_tb.vcd");

$dumpvars(0, UARTTx\_tb);

rst\_i = 1'b1;

byteReady\_i = 1'b0;

load\_i = 1'b0;

TxByte\_i = 1'b0;

busData\_i = 8'b0000\_0000;

#50;

rst\_i = 1'b0;

byteReady\_i = 1'b1;

busData\_i = 8'b0101\_0101;

#100;

byteReady\_i = 1'b0;

load\_i = 1'b1;

#20;

load\_i = 1'b0;

#20;

TxByte\_i = 1'b1;

#100;

$finish;

end

endmodule