**NAME: PULLETIGURTHI DINESH**

**EMAIL:** [**pulletigurthidinesh@gmail.com**](mailto:pulletigurthidinesh@gmail.com)

**SPECIFICATION:**

**BOARD: Aritx-7 AC701**

**Question 1: Design a digital system which accepts parallel data packets and transmits them serially over a UART-like interface. The digital logic design of the system shall include the RTL development of the following modules.**

**a) Design VHDL/Verilog RTL code for the modules.**

module uart\_packetizer\_top (

input wire clk,

input wire rst,

input wire wr\_clk,

input wire rd\_clk,

input wire [7:0] data\_in,

input wire data\_valid,

output wire serial\_out,

output wire fifo\_full,

output wire tx\_busy

);

wire [7:0] fifo\_data;

wire fifo\_empty, fifo\_rd\_en, tx\_start, tx\_ready;

wire [9:0] tx\_packet;

async\_fifo fifo\_inst (

.wr\_clk(wr\_clk),

.rd\_clk(rd\_clk),

.rst(rst),

.wr\_en(data\_valid),

.rd\_en(fifo\_rd\_en),

.din(data\_in),a

.dout(fifo\_data),

.fifo\_full(fifo\_full),

.fifo\_empty(fifo\_empty),

.data\_out\_valid()

);

packetizer\_fsm fsm\_inst (

.clk(clk),

.rst(rst),

.tx\_ready(tx\_ready),

.fifo\_empty(fifo\_empty),

.fifo\_data(fifo\_data),

.fifo\_rd\_en(fifo\_rd\_en),

.tx\_start(tx\_start),

.tx\_packet(tx\_packet),

.tx\_busy(tx\_busy)

);

uart\_serializer uart\_inst (

.clk(clk),

.rst(rst),

.tx\_start(tx\_start),

.tx\_packet(tx\_packet),

.serial\_out(serial\_out),

.tx\_ready(tx\_ready)

);

endmodule

module uart\_serializer (

input wire clk,

input wire rst,

input wire tx\_start,

input wire [9:0] tx\_packet,

output reg serial\_out,

output reg tx\_ready

);

reg [3:0] bit\_count;

reg [9:0] shift\_reg;

reg sending;

always @(posedge clk or posedge rst) begin

if (rst) begin

serial\_out <= 1;

tx\_ready <= 1;

bit\_count <= 0;

shift\_reg <= 0;

sending <= 0;

end else begin

if (tx\_start && ~sending) begin

sending <= 1;

tx\_ready <= 0;

shift\_reg <= tx\_packet;

bit\_count <= 0;

end else if (sending) begin

serial\_out <= shift\_reg[0];

shift\_reg <= shift\_reg >> 1;

bit\_count <= bit\_count + 1;

if (bit\_count == 9) begin

sending <= 0;

tx\_ready <= 1;

serial\_out <= 1;

end

end

end

end

endmodule

module packetizer\_fsm (

input wire clk,

input wire rst,

input wire tx\_ready,

input wire fifo\_empty,

input wire [7:0] fifo\_data,

output reg fifo\_rd\_en,

output reg tx\_start,

output reg [9:0] tx\_packet,

output reg tx\_busy

);

reg [1:0] state;

parameter IDLE = 2'b00,

READ\_FIFO = 2'b01,

LOAD\_PACKET = 2'b10,

TRANSMIT = 2'b11;

always @(posedge clk) begin

if (rst) begin

state <= IDLE;

fifo\_rd\_en <= 0;

tx\_start <= 0;

tx\_packet <= 10'b0;

tx\_busy <= 0;

end else begin

fifo\_rd\_en <= 0;

tx\_start <= 0;

case (state)

IDLE: begin

tx\_busy <= 0;

if (~fifo\_empty && tx\_ready)

state <= READ\_FIFO;

end

READ\_FIFO: begin

fifo\_rd\_en <= 1;

state <= LOAD\_PACKET;

end

LOAD\_PACKET: begin

tx\_packet <= {1'b1, fifo\_data, 1'b0};

tx\_start <= 1;

tx\_busy <= 1;

state <= TRANSMIT;

end

TRANSMIT: begin

tx\_busy <= 1;

if (~tx\_ready)

state <= IDLE;

end

endcase

end

end

endmodule

module async\_fifo (

input wire wr\_clk,

input wire rd\_clk,

input wire rst,

input wire wr\_en,

input wire rd\_en,

input wire [7:0] din,

output reg [7:0] dout,

output wire fifo\_full,

output wire fifo\_empty,

output wire data\_out\_valid

);

reg [7:0] mem [15:0];

reg [3:0] wr\_ptr = 0;

reg [3:0] rd\_ptr = 0;

reg [4:0] count = 0;

assign fifo\_full = (count == 16);

assign fifo\_empty = (count == 0);

assign data\_out\_valid = ~fifo\_empty;

always @(posedge wr\_clk) begin

if (rst) begin

wr\_ptr <= 0;

count <= 0;

end else if (wr\_en && ~fifo\_full) begin

mem[wr\_ptr] <= din;

wr\_ptr <= wr\_ptr + 1;

count <= count + 1;

end

end

always @(posedge rd\_clk) begin

if (rst) begin

rd\_ptr <= 0;

end else if (rd\_en && ~fifo\_empty) begin

dout <= mem[rd\_ptr];

rd\_ptr <= rd\_ptr + 1;

count <= count - 1;

end

end

endmodule

**b) Develop a simulation testbench and provide the following results.**

**1. FIFO write and read.**

**2. Serial output waveform along with fifo\_full and tx\_busy**

**signals.**

**3. At least 2 valid packets with proper framing.**

**4. FSM state transitions.**

`timescale 1ns / 1ps

module tb\_uart\_packetizer;

reg clk;

reg wr\_clk;

reg rd\_clk;

reg rst;

reg [7:0] data\_in;

reg data\_valid;

wire serial\_out;

wire fifo\_full;

wire tx\_busy;

uart\_packetizer\_top uut (

.clk(clk),

.rst(rst),

.wr\_clk(wr\_clk),

.rd\_clk(rd\_clk),

.data\_in(data\_in),

.data\_valid(data\_valid),

.serial\_out(serial\_out),

.fifo\_full(fifo\_full),

.tx\_busy(tx\_busy)

);

always #10 clk = ~clk;

always #15 wr\_clk = ~wr\_clk;

always #20 rd\_clk = ~rd\_clk;

initial begin

clk = 0;

wr\_clk = 0;

rd\_clk = 0;

rst = 1;

data\_in = 8'b00000000;

data\_valid = 0;

#50;

rst = 0;

#50;

send\_data(8'hA5);

#200;

send\_data(8'h3C);

#1000;

$finish;

end

task send\_data(input [7:0] byte);

begin

@(posedge wr\_clk);

data\_in = byte;

data\_valid = 1;

@(posedge wr\_clk);

data\_valid = 0;

end

endtask

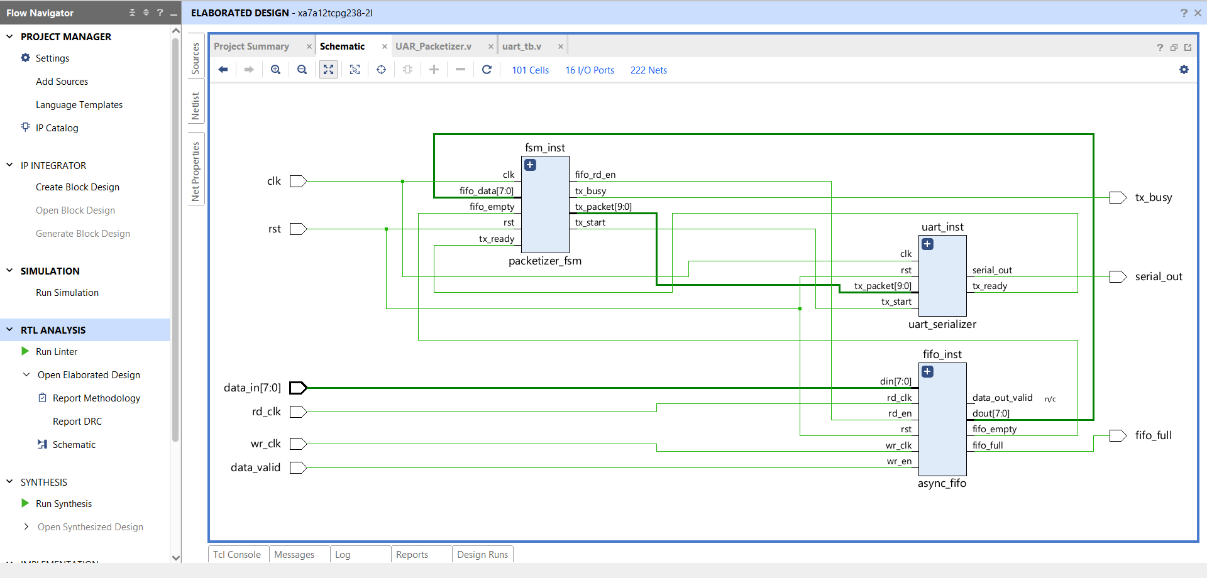
endmodule

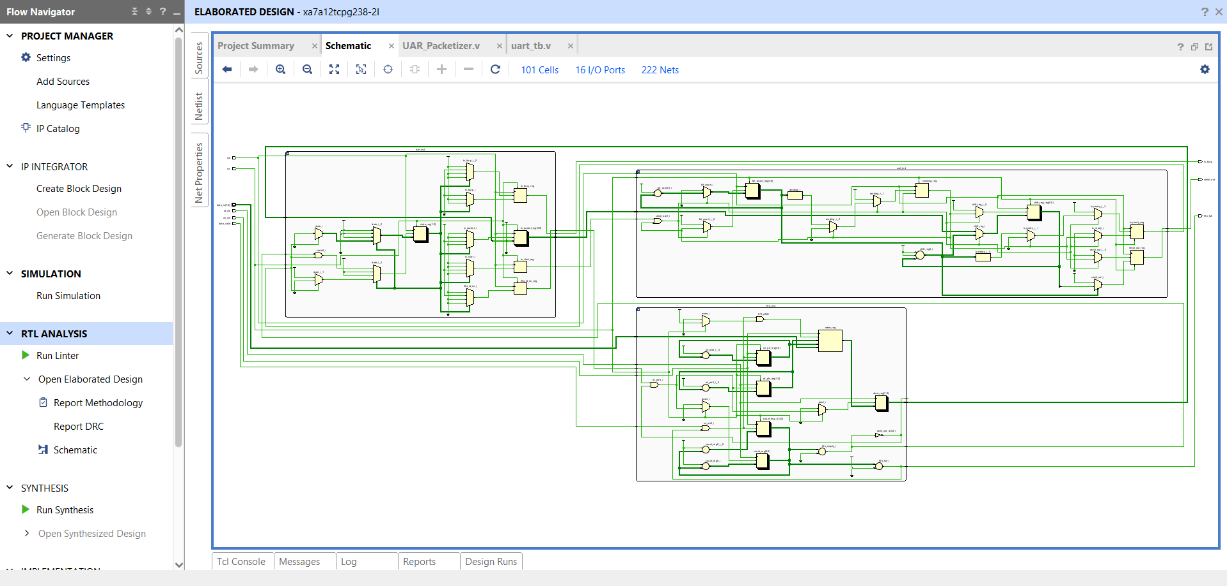
**c) Generate documentation with following details**

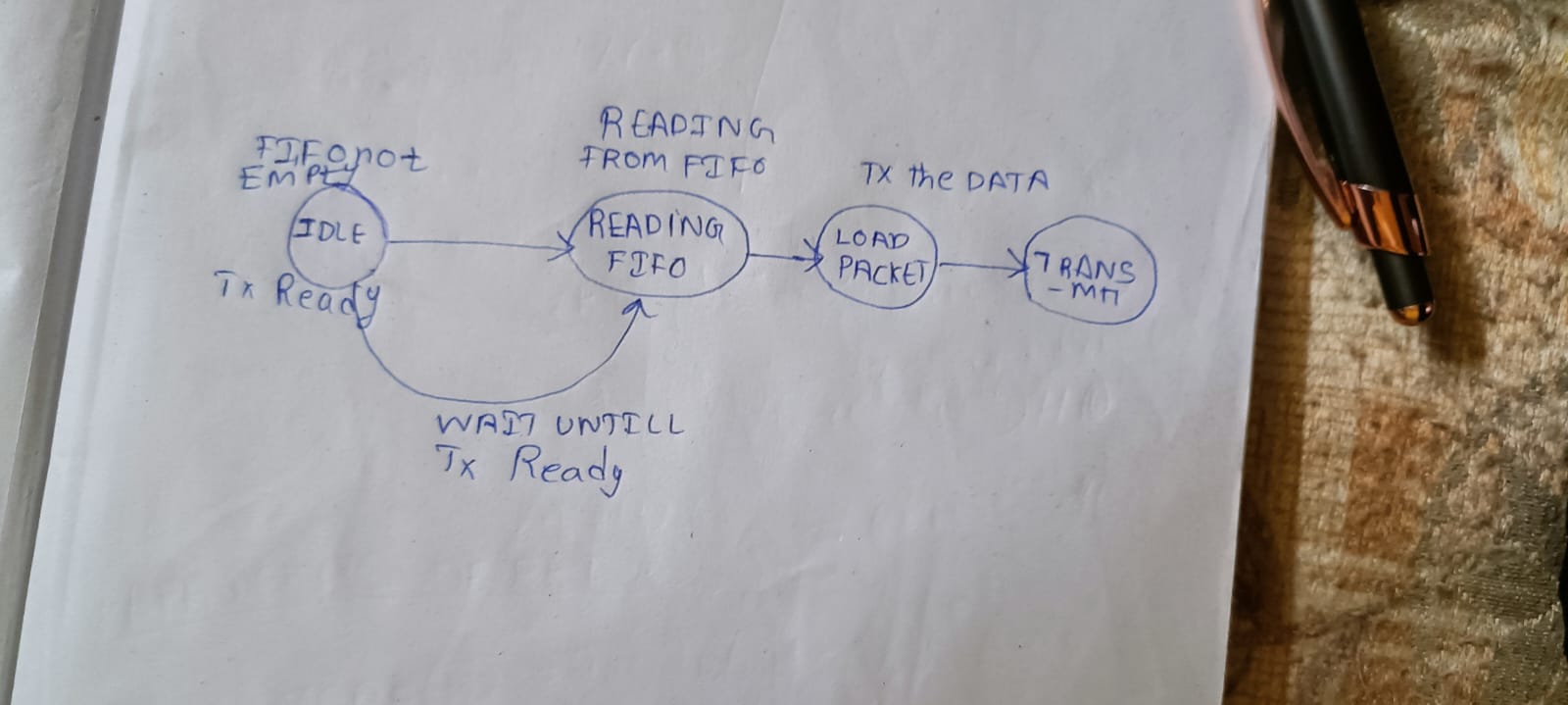
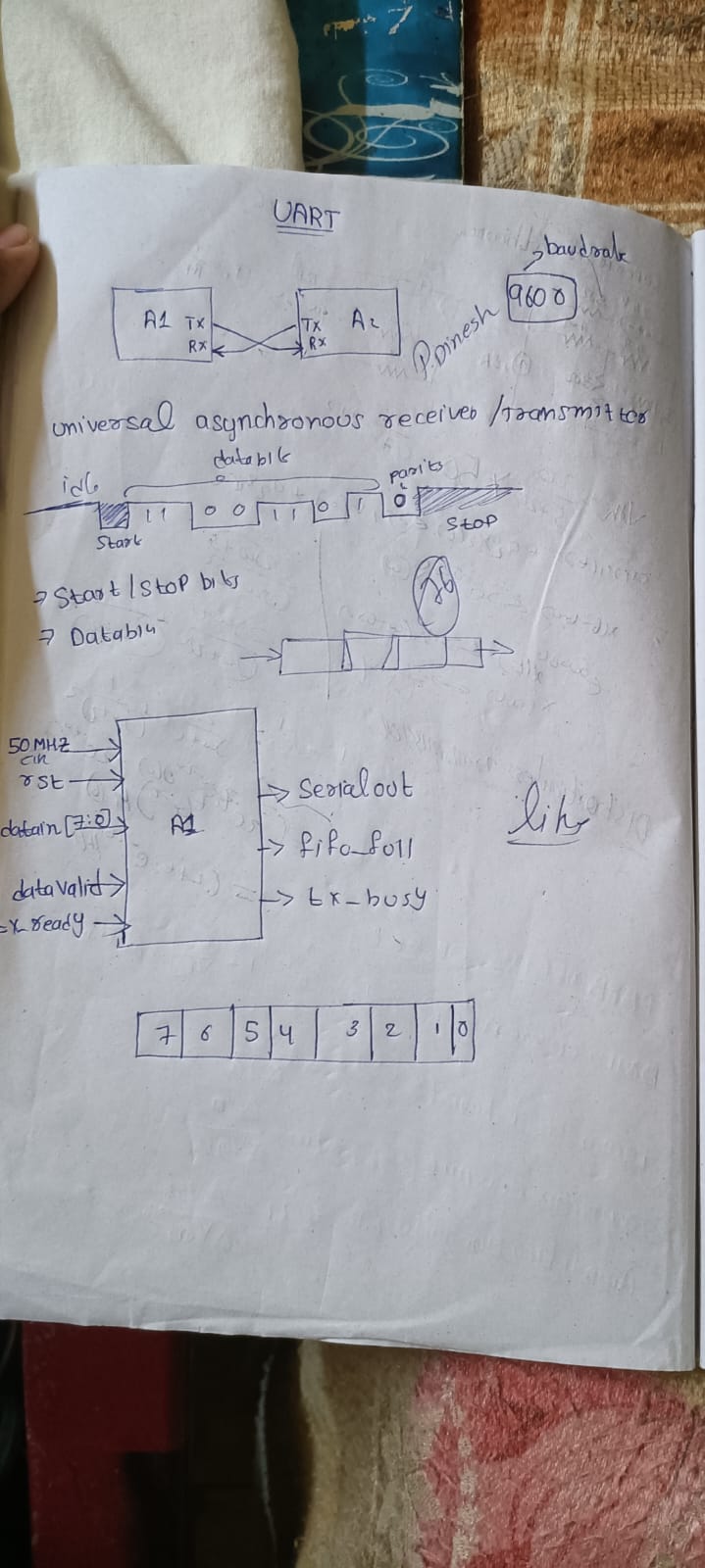
**1. FSM state diagram along with state transitions.**

**2. FIFO usage explanation.**

**3. Rational for synchronous/asynchronous FIFO**

FIFO stands for first in first out whenever any element or data store in a array or Queue which element is first in it goes first just like the order wise when it comes to the RTL code with the help of FSM RTL code is designed with the help of the HDL language like Verilog by understanding the UART protocol and FSM design process final schematic is generated.



****

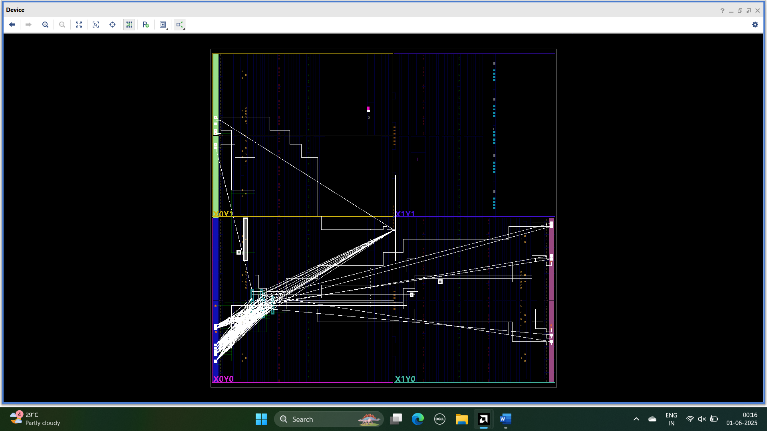
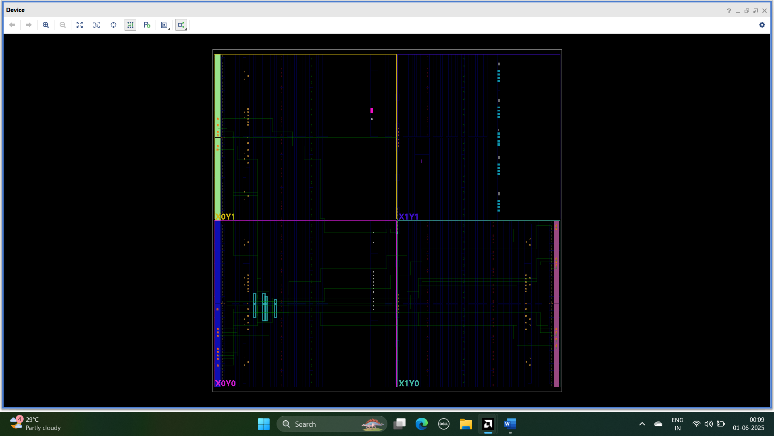
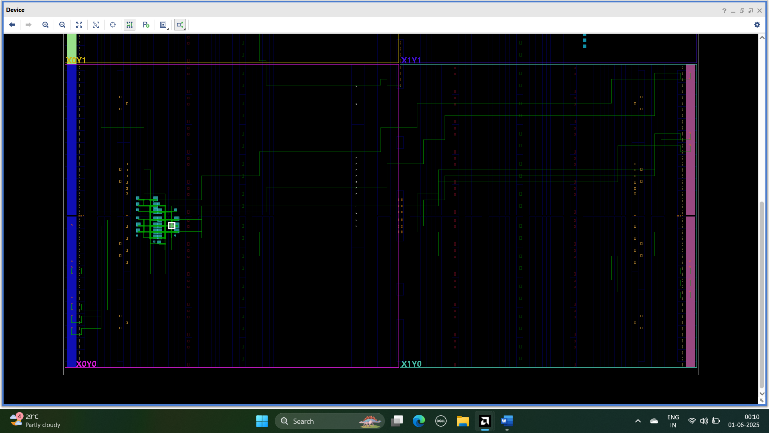
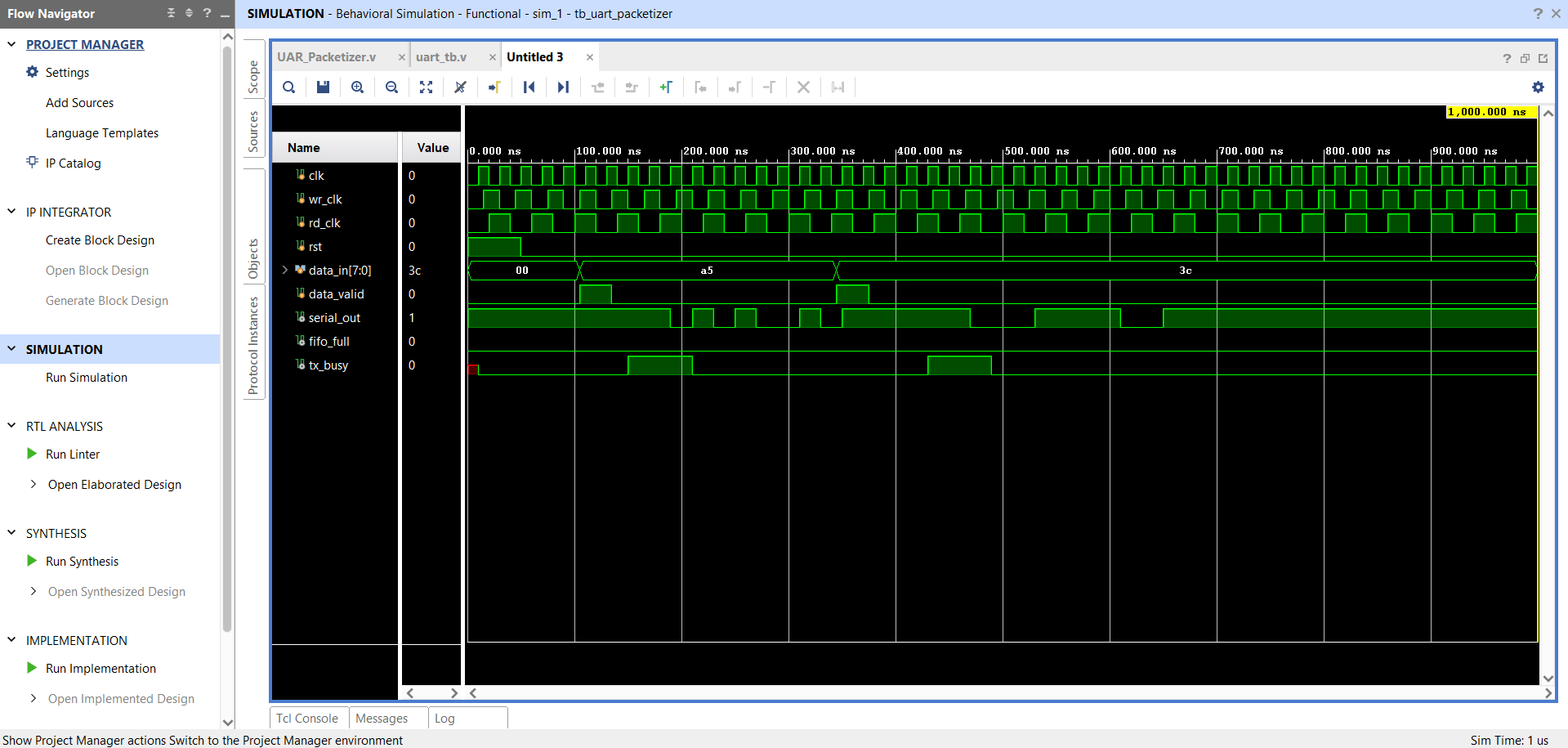
**Question 2**

**a) Carry out synthesis, place and route on AMD-Xillinx Vivado tool with a target FPGAfrom chosen from Kintex/Virtex family of FPGAs?**

**b) Provide the synthesis and timing reports generated by AMD-Xillinx Vivado tool.**

**c) Report the resource utilization and achieved maximum operating clock frequencyat Synthesis level and post-place and route level?**

**d) Describe the encoding technique is used in the FSM implementation. Provide a tool generated report indicating the type of encoding technique used for FSM implementation.**

****