NAME: PULLETIGURTHI DINESH

EMAIL: 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
```

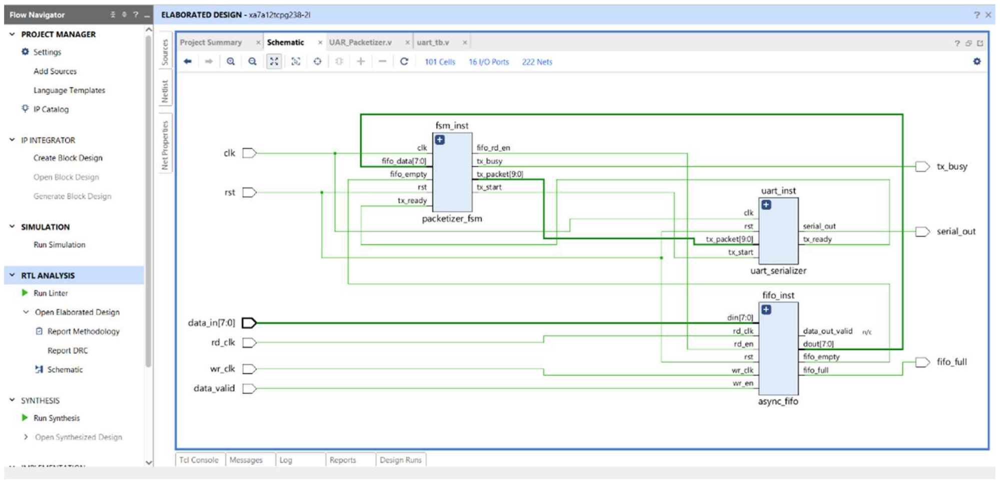
c) Generate documentation with following details

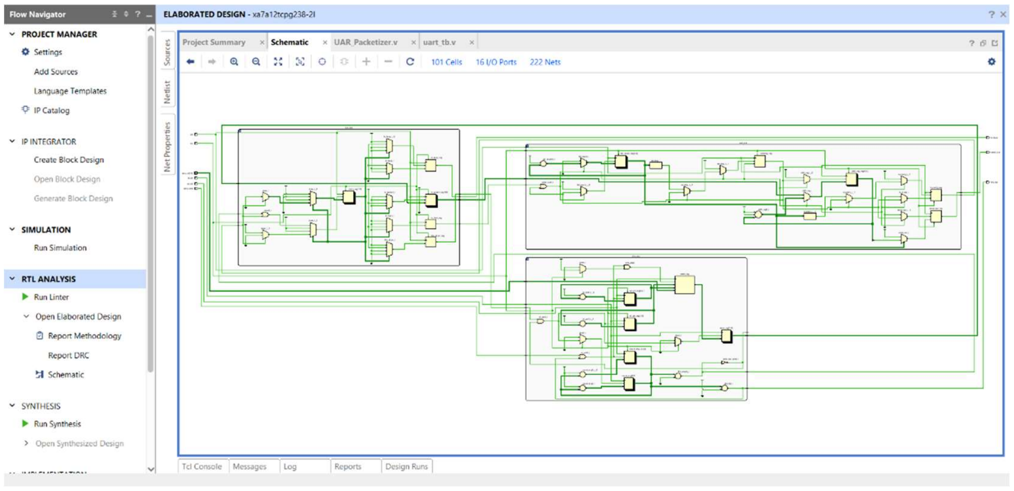
1. FSM state diagram along with state transitions.

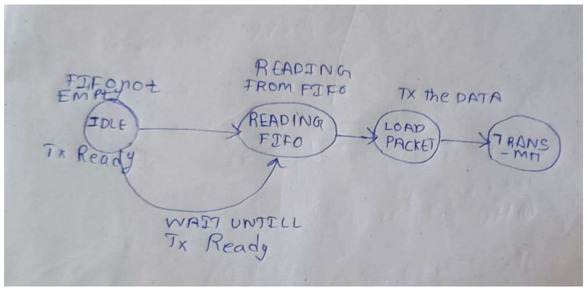
2. FIFO usage explanation.

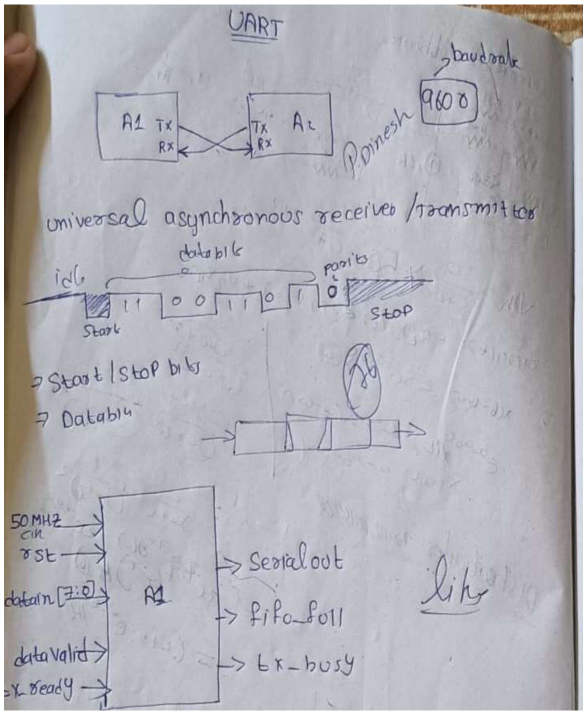
3. Rational for synchronous/asynchronous FIFO

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.









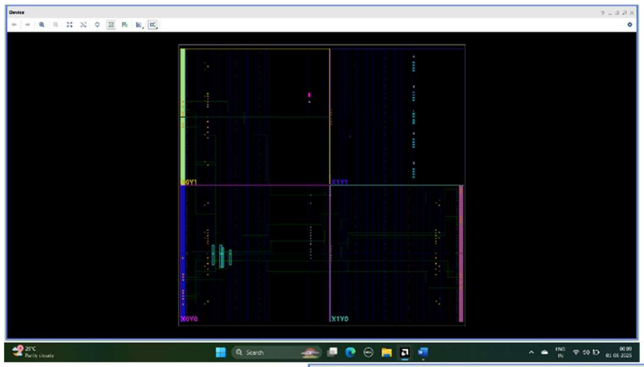
Question 2

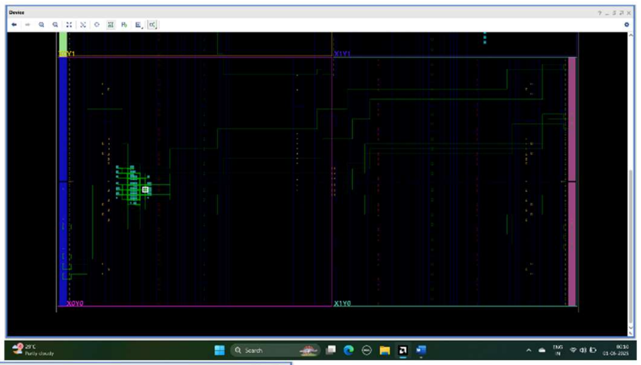
a) Carry out synthesis, place and route on AMD-Xillinx Vivado tool with a target FPGAfrom chosen from Kintex/Virtex family of FPGAs?

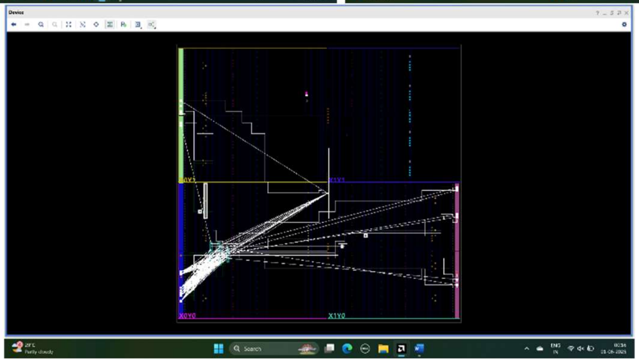
b) Provide the synthesis and timing reports generated by AMD-Xillinx Vivado tool.

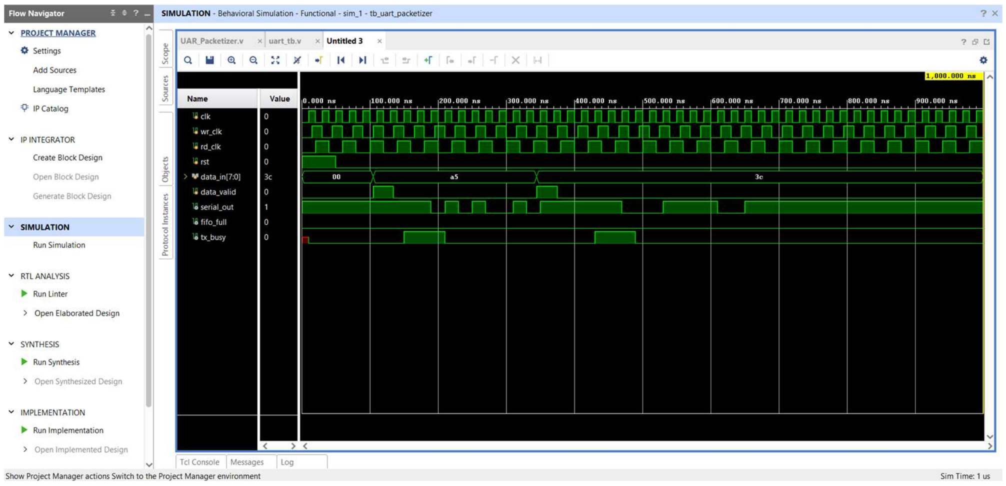
c) Report the resource utilization and achieved maximum operating clock frequencyat Synthesis level and post-place and route level?

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.


Device
X0Y1
X1Y1
X0Y0
X1Y0


Device
X0Y1
X1Y1
X0Y0
X1Y0


Device
X0Y1
X1Y1
X0Y0
X1Y0


Flow Navigator
PROJECT MANAGER
Settings
Add Sources
Language Templates
IP Catalog
IP INTEGRATOR
Create Block Design
Open Block Design
Generate Block Design
SIMULATION
Run Simulation
RTL ANALYSIS
Run Linter
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
Open Implemented Design
SIMULATION - Behavioral Simulation - Functional - sim_1 - tb_uart_packetizer
UAR_Packetizer.v
uart_tb.v
Untitled 3
Name
Value
clk 0
wr_clk 0
rd_clk 0
rst 0
data_in[7:0] 3c
data_valid 0
serial_out 1
fifo_full 0
tx_busy 0
1,000.000 ns
0.000 ns
100.000 ns
200.000 ns
300.000 ns
400.000 ns
500.000 ns
600.000 ns
700.000 ns
800.000 ns
900.000 ns
00
a5
3c
Tcl Console
Messages
Log
Show Project Manager actions Switch to the Project Manager environment
Sim Time: 1 us