Digital Electronics Verilog Lab 4

Submitted By:
Saurabh Kumar
SC ID: SC22B146
Course: DIGITAL ELECTRONICS AND VLSI DESIGN LAB (AV232)

LAB SHEET

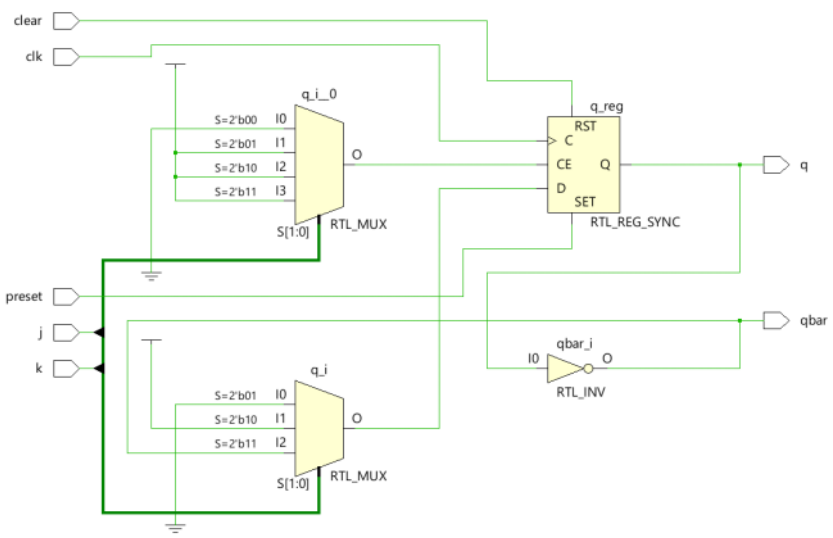
Question No. 1 Write a Verilog code for the following flip flops using behavioural modelling with preset and clear inputs.

a) Simple JK Flip Flop with
i. Synchronous and

Sol: Code:

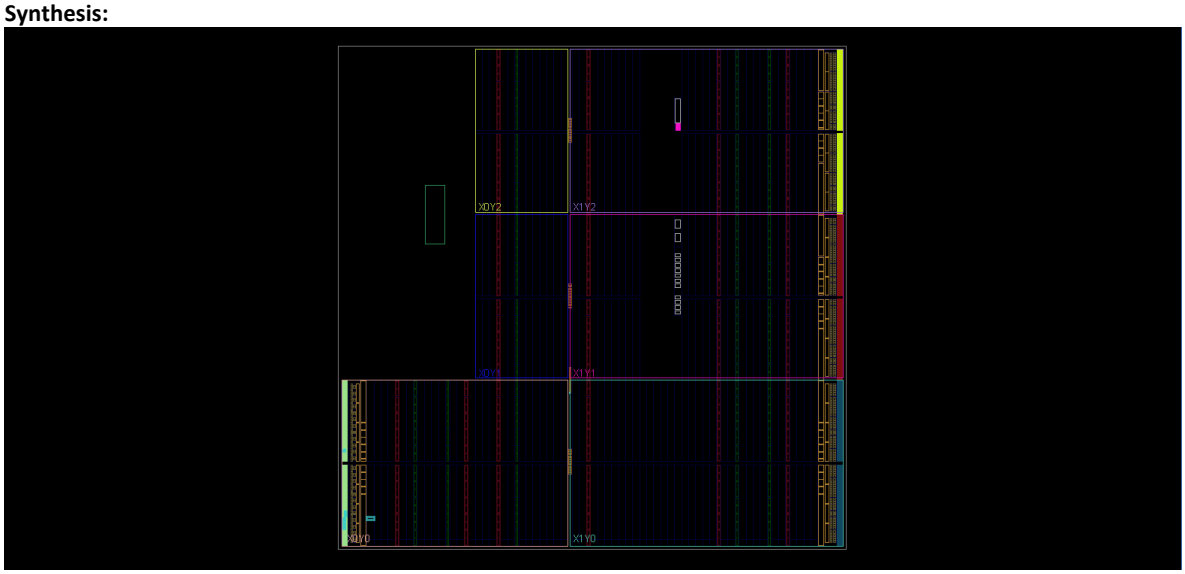
```
module jksync(clk,preset,clear,j,k,q,qbar);
  input clk,preset,clear,j,k;
  output reg q;
  output qbar;

always@(posedge clk)
begin
  if(clear==1)
    q <= 0;
  else
    begin
    if(preset==1)
      q <= 1;
    else
      begin
       case({j,k})
        2'b00: q <= q;
        2'b01: q <= 1'b0;
        2'b10: q <= 1'b1;
        2'b11: q <= ~q;
        default: q <= q;
       endcase
      end
    end
end
assign qbar = ~q;
endmodule
```

Schematic:



Simulation:

Synthesis:



Question No. 1. a) ii. Asynchronous reset port. Discuss the advantages of synchronous designs over asynchronous circuits.
Sol: Advantages of synchronous designs over asynchronous circuits:

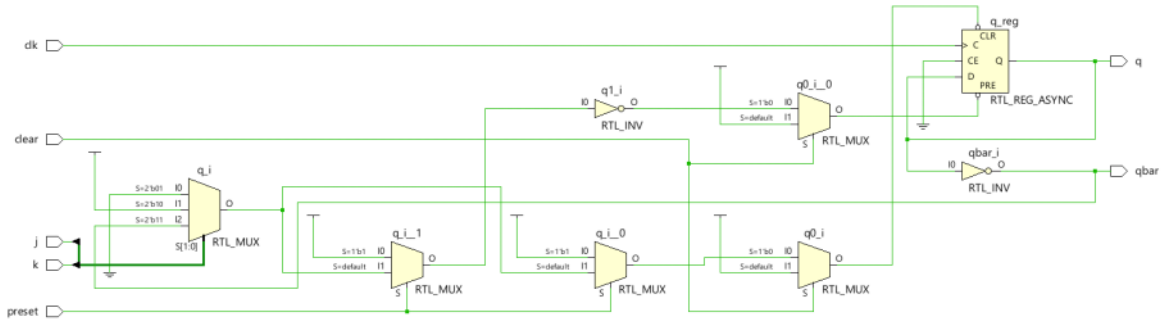
Synchronous Sequential Circuits are digital sequential circuits in which the feedback to the input for the next output generation is governed by a universal clock signal, while Asynchronous Sequential Circuits are digital sequential circuits in which the feedback to the input for the next output generation is not governed by a universal clock signal.

Synchronous designs are more reliable. They are deterministic in their behaviour, due to the fact that all signals are sampled at a well-defined time interval. Synchronous designs rely on very few timing parameters to guarantee operation, namely, the maximum frequency of operation of a device (fmax), the register setup and hold times (tSU and tH), and the register propagation delay (tPD).
Synchronous designs can be tested more easily and run statically, with the clock input driven by a test signal.
They can be made virtually immune to noise. Synchronous designs attain performance levels easily.
Finally, synchronous designs are easier to code in a hardware description language (HDL), and are also easier to read.

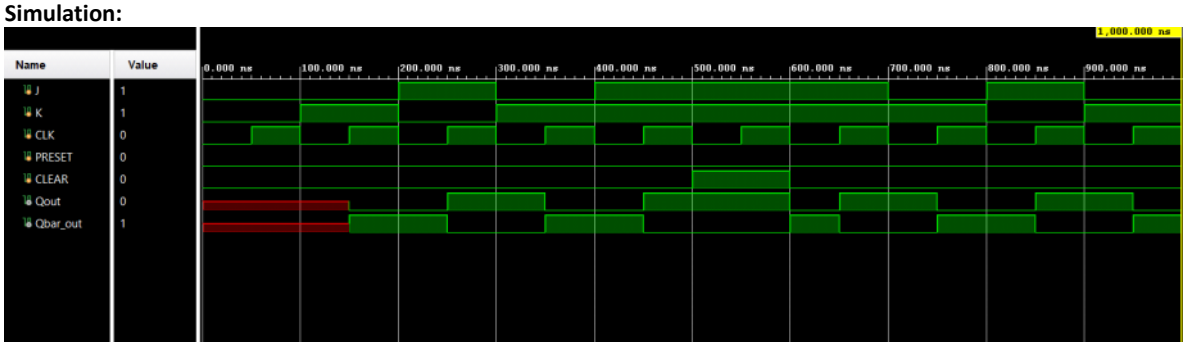
Code:

```
module jkasync(clk,preset,clear,j,k,q,qbar);
   input clk,preset,clear,j,k;
   output reg q;
   output qbar;

always@(posedge clk or negedge clear)
begin
   if(clear==1)
      q <= 0;
   else
      begin
      if(preset==1)
         q <= 1;
      else
         begin
          case({j,k})
            2'b00: q <= q;
            2'b01: q <= 1'b0;
            2'b10: q <= 1'b1;
            2'b11: q <= ~q;
            default: q <= q;
          endcase
         end
      end
end
assign qbar = ~q;
endmodule
```

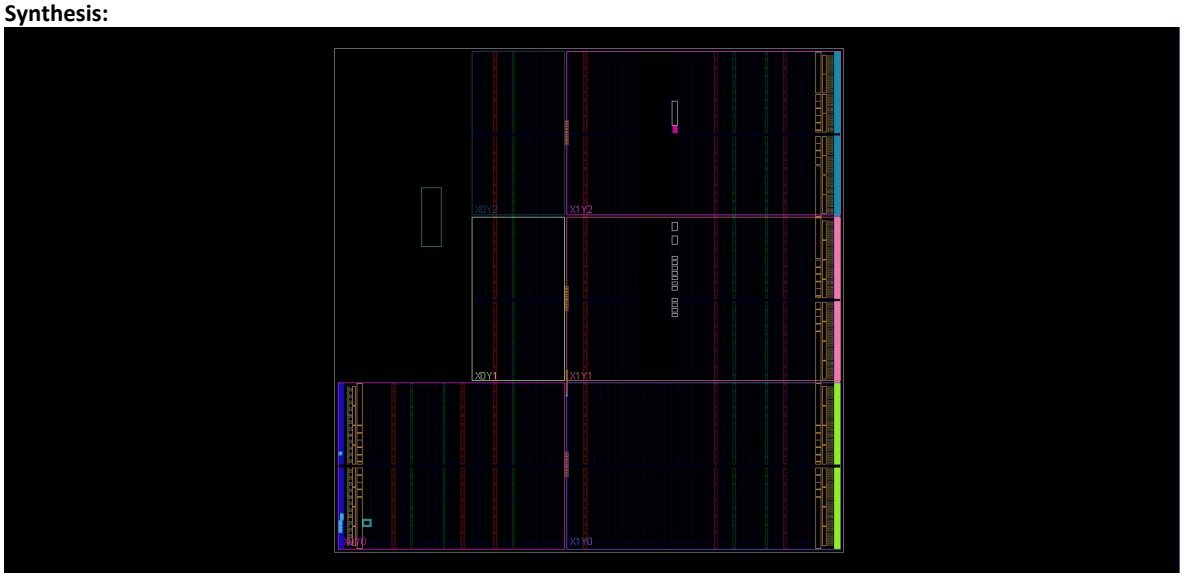
Schematic:



Simulation:



Synthesis:



Question No. 1. b) Master Slave JK FLIP FLOP with their advantages over simple JKFF

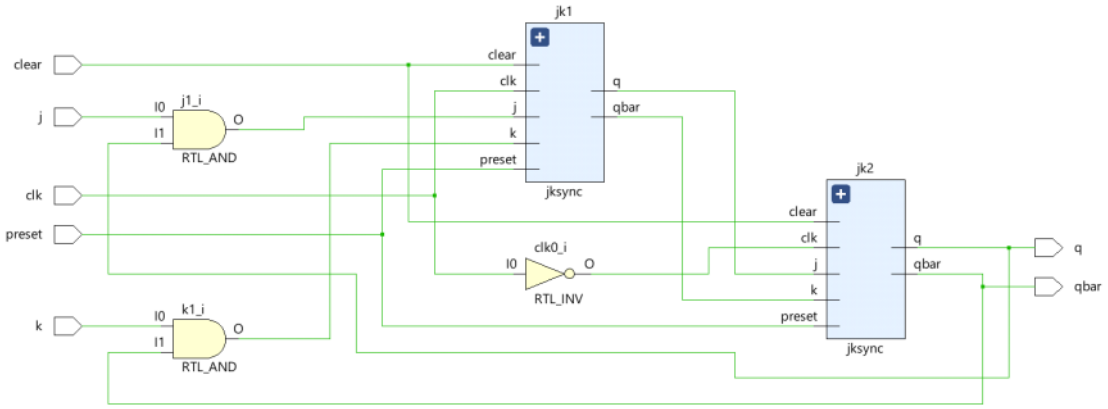
Sol: Advantage of master JK FLIP FLOP over simple JK FLIP FLOP:

Master slave configuration removes the possibility of race around condition from JK flip flop when we put both J and K as 1.
In normal JK flip flop when we put both J and K as 1 and as clock level becomes high the output changes continuously from 0 to 1 and 1 to 0 till the clock is high.
However in master slave configuration when we put both J & K as 1 the present output will be compliment of past output i.e., the race around is changes to toggling. The output in master slave changes when negative edge of clock reaches i.e., negative triggers at negative edge which also removes problem due to propagation delay.
This toggling effect helps in designing counters.

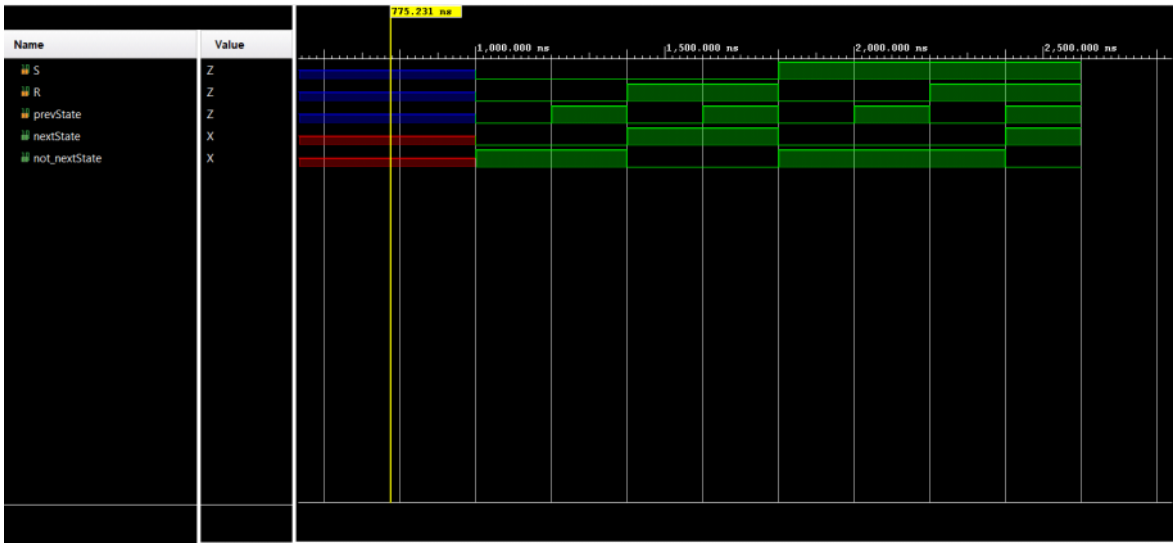
Code:

```
module jk_masterSlave(clk,preset,clear,j,k,q,qbar);
   input clk,preset,clear,j,k;
   output q,qbar;

   wire j1,k1,j2,k2;

   and(j1,j,q);
   and(k1,k,qbar);

   jksync jk1(clk,preset,clear,j1,k1,j2,k2);
   jksync jk2(~clk,preset,clear,j2,k2,q,qbar);

endmodule
```

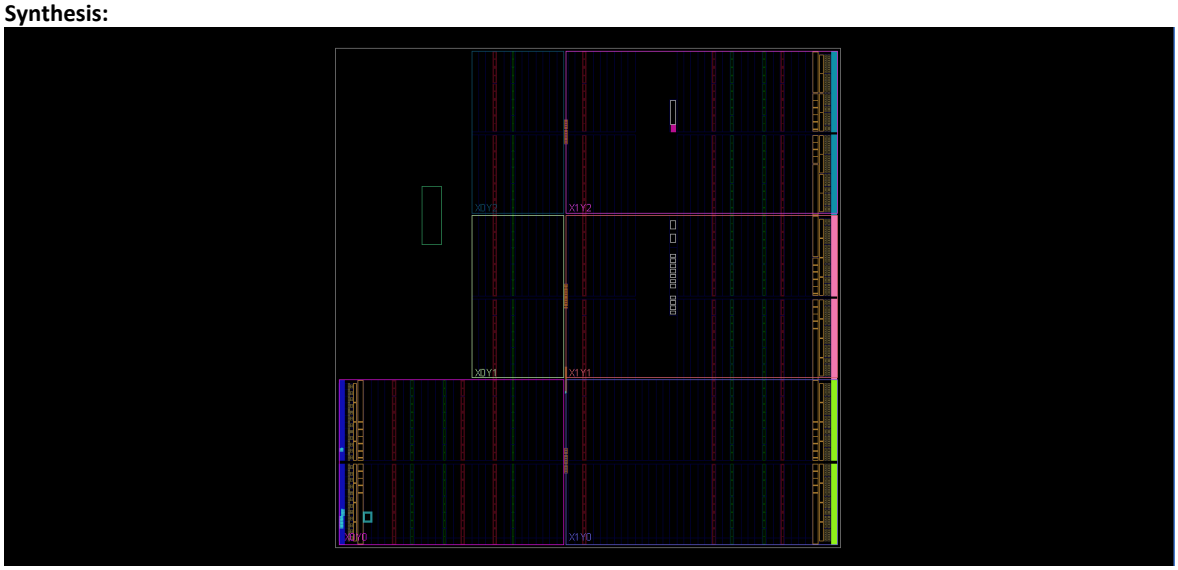
Schematic:



Simulation:



Synthesis:



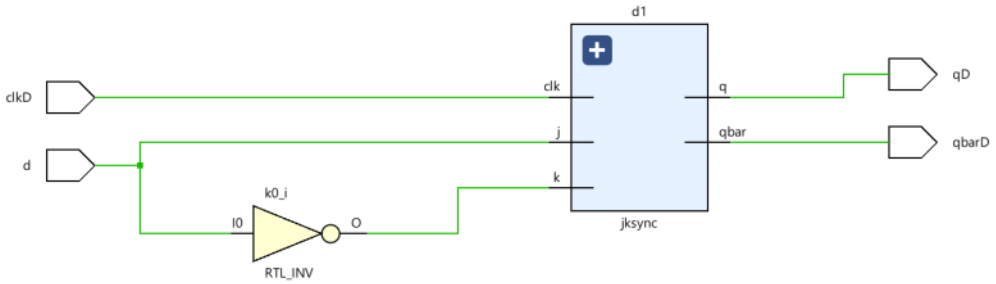
Question No. 1. c) D FLIP FLOP from a Simple JK FF

Sol: Code:

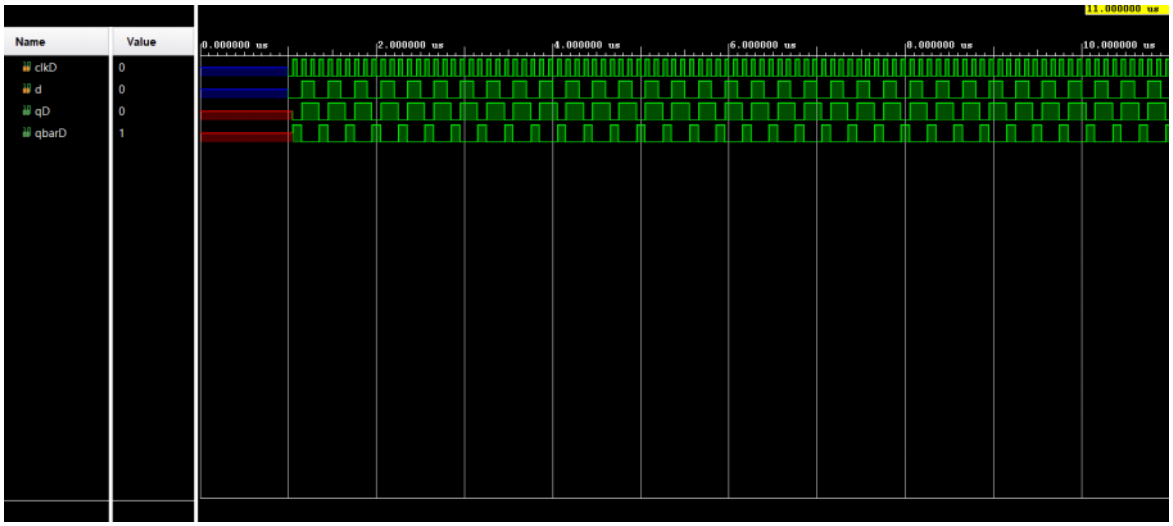
```
module DfromJK(clkD,d,qD,qbarD);
  input clkD,d;
  output qD,qbarD;
```

```
    jksync d1(.clk(clkD), .j(d), .k(~d), .q(qD), .qbar(qbarD));
endmodule
```

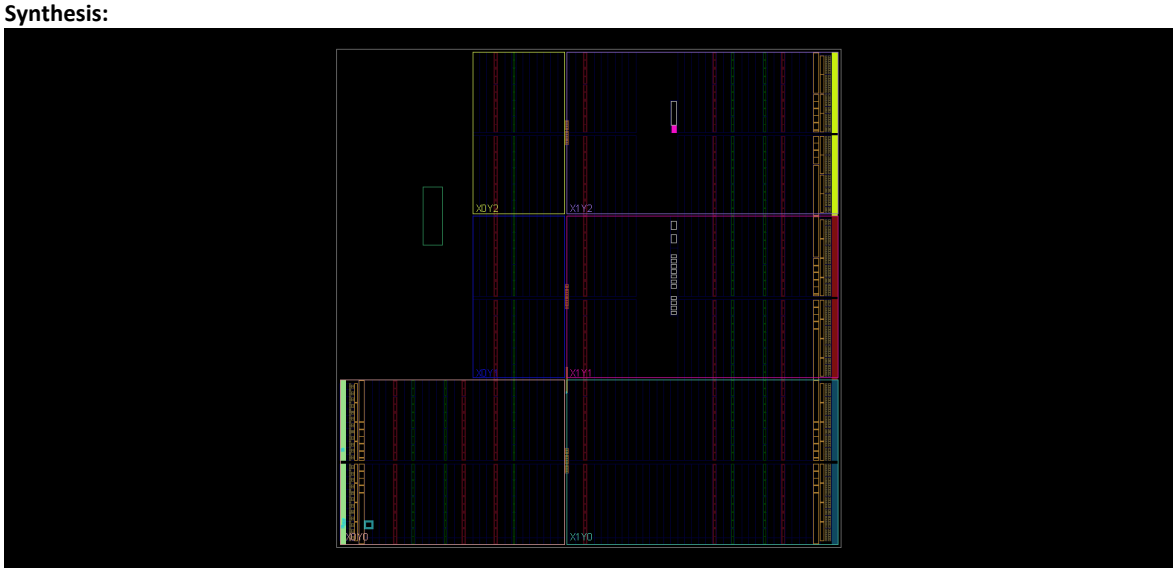
Schematic:



Simulation:



Synthesis:



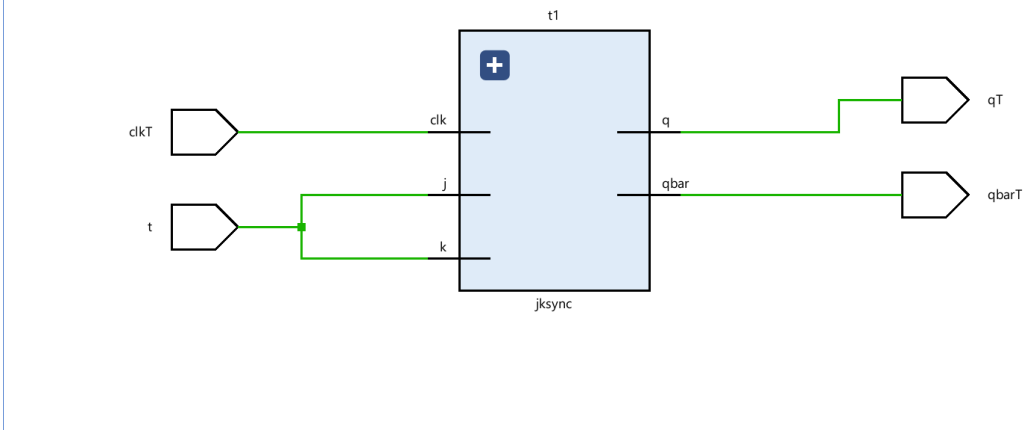
Question No. 1. d) T FLIP FLOP from a Simple JK FF

Sol: Code:

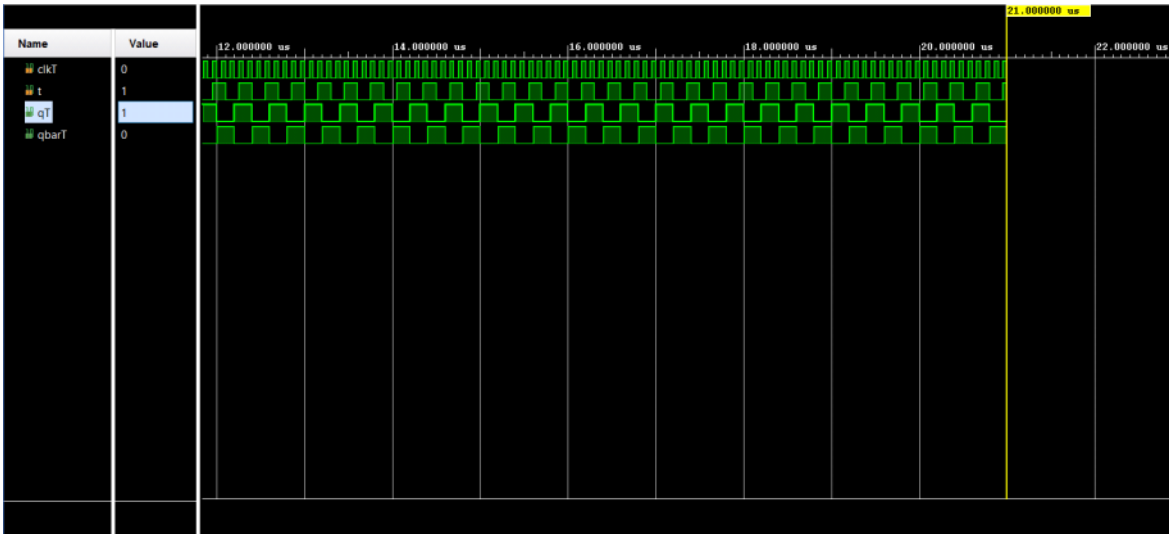
```
    module TfromJK(clkT,t,qT,qbarT);
      input clkT,t;
      output qT,qbarT;
```

```
    jksync t1(.clk(clkT), .j(t), .k(t), .q(qT), .qbar(qbarT));
endmodule
```

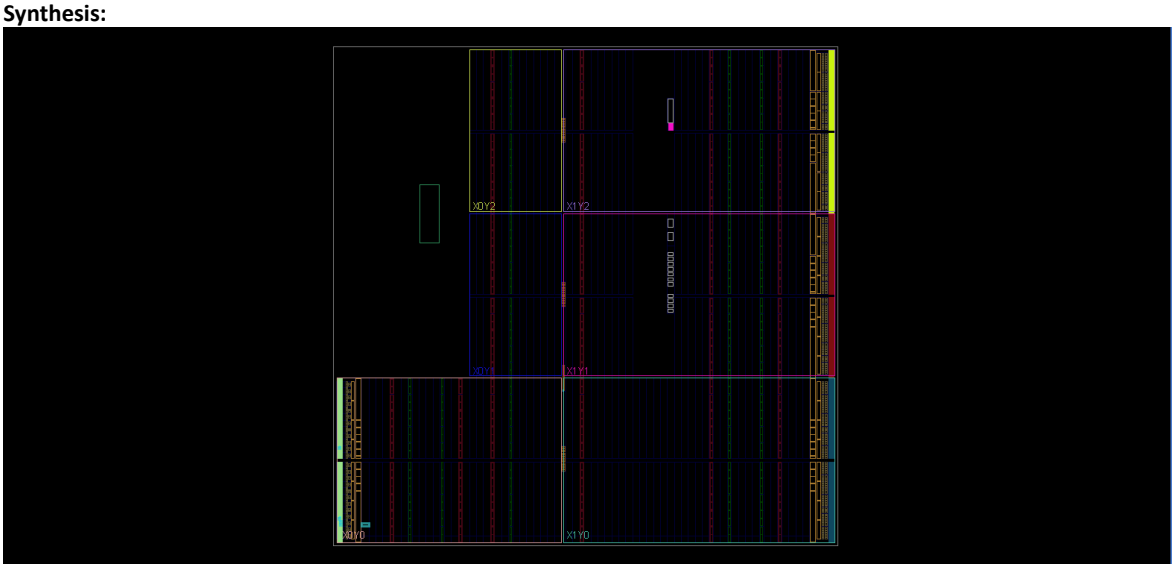
Schematic:



Simulation:



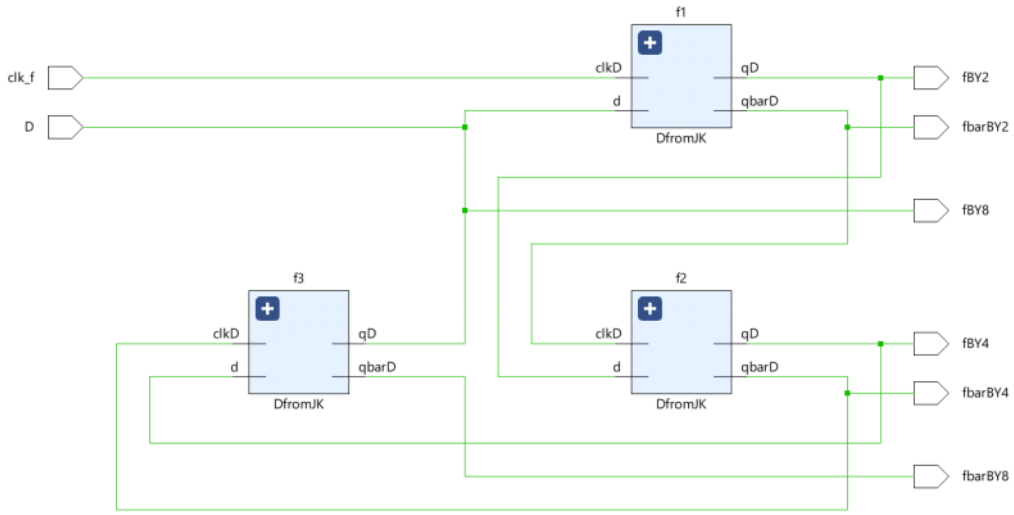
Synthesis:



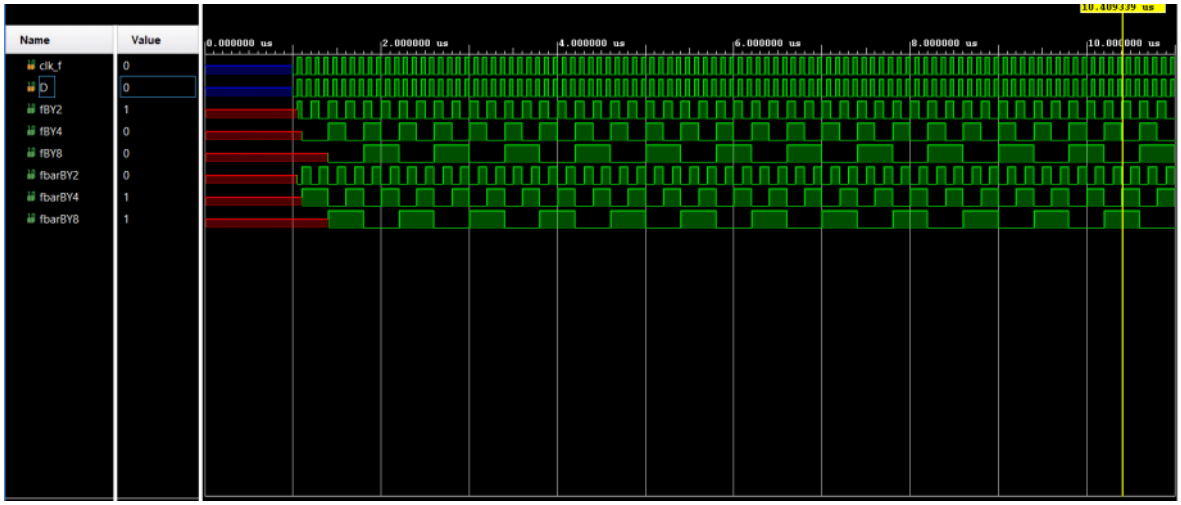
Question No. 2 Design a frequency divider to divide the input clock by 2, 4 and 8 using the DFF designed in the first question and basic gat es.
Sol: Code:

```
module fDivider(clk_f,D,fBY2,fBY4,fBY8,fbarBY2, fbarBY4,fbarBY8);
input clk_f,D;
output fBY2,fBY4,fBY8,fbarBY2, fbarBY4,fbarBY8;
assign D = fBY8;

DfromJK f1(.clkD(clk_f), .d(D), .qD(fBY2), .qbarD(fbarBY2));
DfromJK f2(.clkD(fbarBY2), .d(fBY2), .qD(fBY4), .qbarD(fbarBY4));
DfromJK f3(.clkD(fbarBY4), .d(fBY4), .qD(fBY8), .qbarD(fbarBY8));
endmodule
```

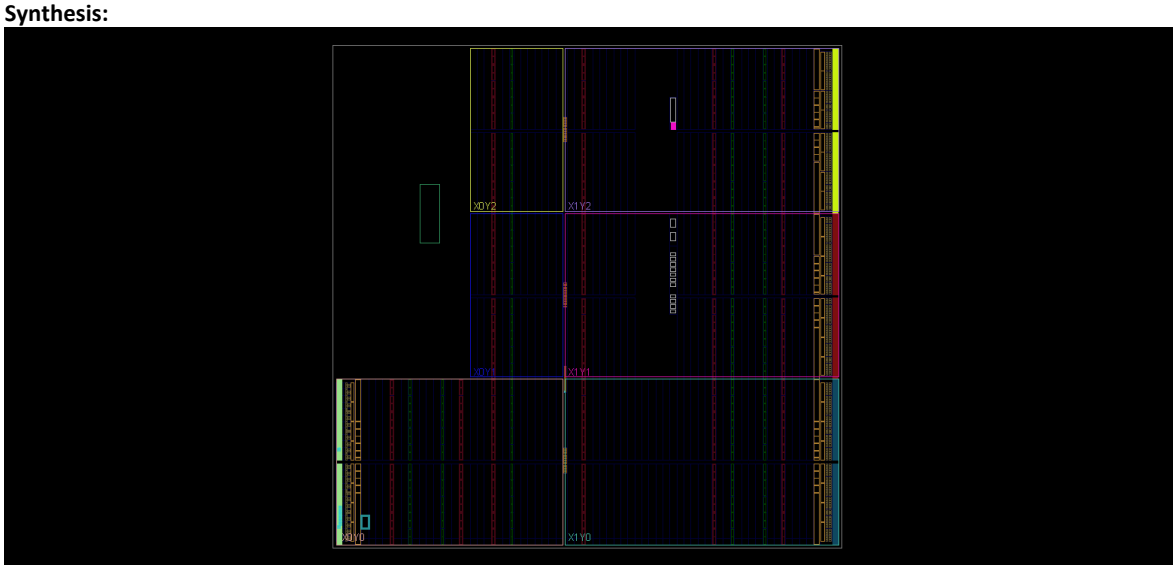
Schematic:



Simulation:



Synthesis:



ASSIGNMENT

Question. Design a 4-bit synchronous shift register? Use user defined input clock.
Sol: Code:

```
// Implementation using DFF
module DFF(clk,reset,D,Q,Qbar);
   input D,clk,reset;
   output reg Q, Qbar;

always@(posedge clk)
begin
   if(reset)
      begin
      Q <= 0;
      Qbar <= 1;
      end
   else
      begin
      Q <= D;
      Qbar <= ~D;
      end

end
endmodule


// 4-bit synchronous shift register
module ShiftRegister(clk,clear,Out,QA,QB,QC,QD,QAbar,QBbar,QCbar,QDbar);
   input clk,clear;
   output Out;
   output QA,QB,QC,QD,QAbar,QBbar,QCbar,QDbar;
   wire qa,qb,qc,qd;

   assign QA = qa, QB = qb, QC = qc, QD = qd;
   assign Out = (QAbar & QDbar);
   DFF reg1(clk,clear,QDbar,qa,QAbar);
   DFF reg2(clk,clear,qa,qb,QBbar);
   DFF reg3(clk,clear,qb,qc,QCbar);
   DFF reg4(clk,clear,qc,qd,QDbar);
endmodule
```

Schematic: