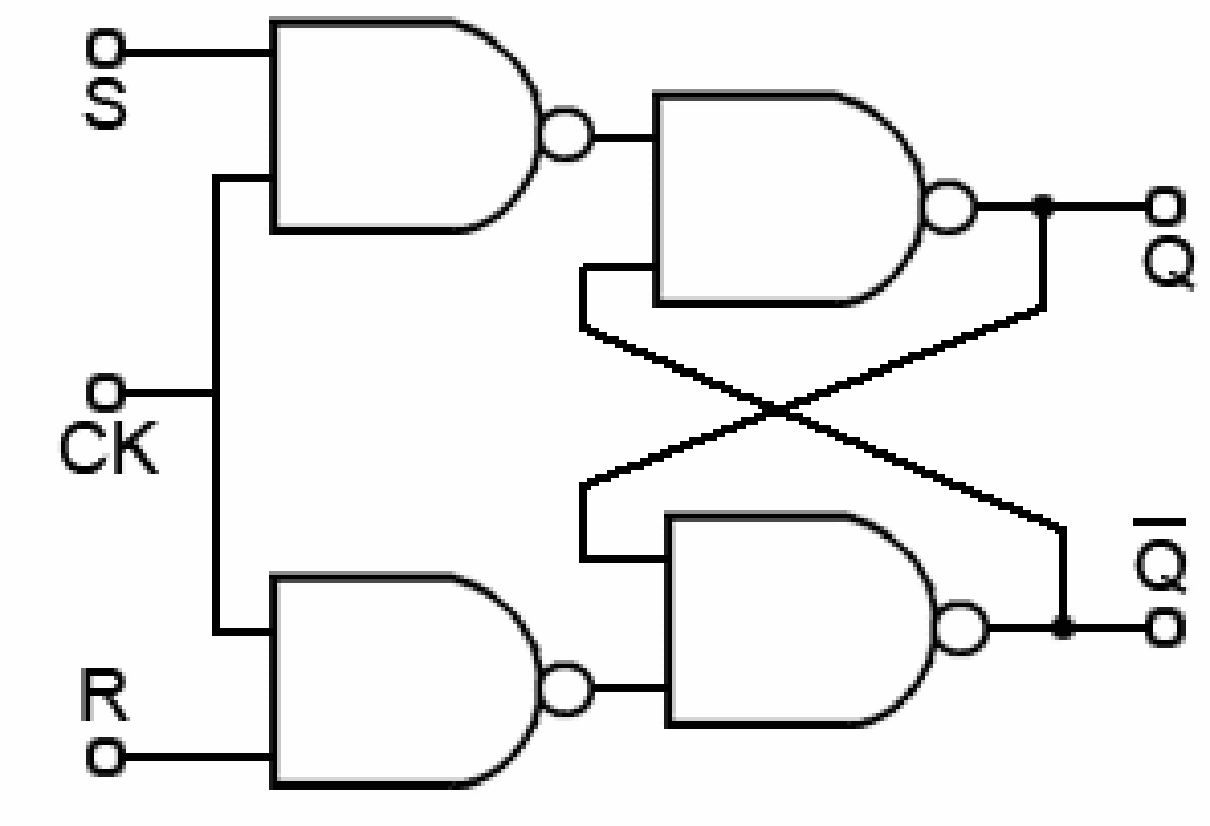
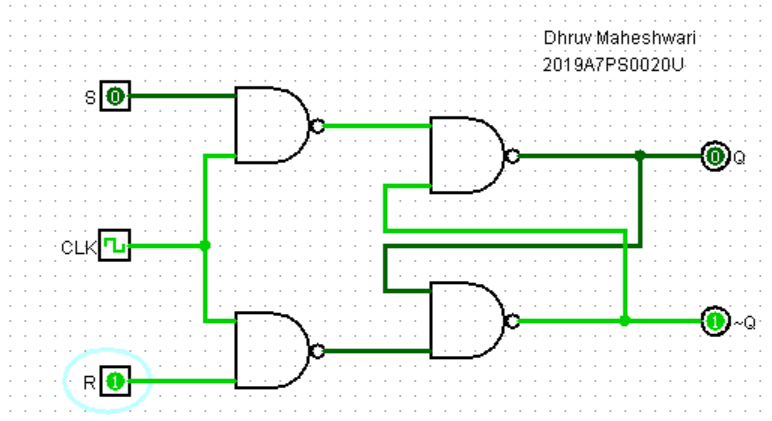
**Hardware runs**

**Run 1: Clocked SR Latch using NAND gates**

 **Diagram**

****

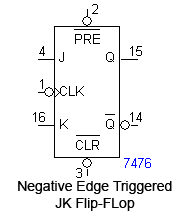
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S** | **R** | **Clock** | **Q(t)** | **Q(t+1)** | **Operation** |
| 0 | 0 | 0 | x | Q(t) | Memory |
| 0 | 1 | 0 | x | Q(t) | Memory |
| 1 | 0 | 0 | x | Q(t) | Memory |
| 1 | 1 | 0 | x | Q(t) | Memory |
| 0 | 0 | 1 | 0 | 0 | Memory |
| 0 | 1 | 1 | x | 0 | Reset |
| 1 | 0 | 1 | x | 1 | Set |
| 1 | 1 | 1 | 1 | 1 | Invalid |

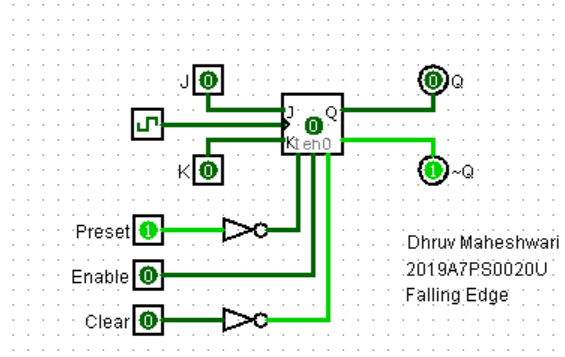
**Q:** What is the disadvantage with an SR latch?

When both inputs S=1 and R=1, both Q and ~Q are same giving invalid output.

**Run 2: JK Flip Flop**

**Diagram**





**Truth Table**

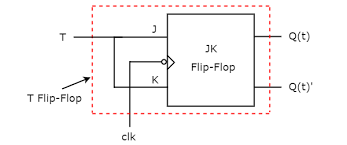
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **J** | **K** | **Pre** | **Clr** | **Clock** | **Q(t)** | **Q(t+1)** | **Operation** |
| x | x | 0 | 0 | x | X | x | Non functional |
| x | x | 0 | 1 | x | 1 | 1 | latched |
| x | x | 1 | 0 | x | 0 | 0 | latched |
| 0 | 0 | 1 | 1 | 1 | x | Q(t) | Memory |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | Reset |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | Set |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | Toggle |

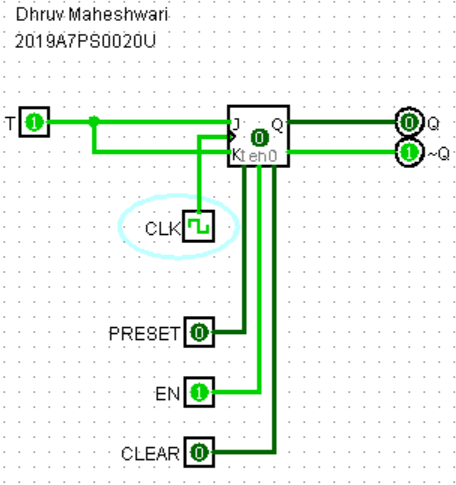
**Q:** Are the preset and clear inputs synchronous or asynchronous.

**A:** Synchronous

**Run 3: T Flip Flop (Using JK Flip Flop)**

**Diagram**



****

**Truth Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **T** | **Clock** | **Q(t)** | **Q(t+1)** |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 |

**Software runs**

**Run 4: SR Latch and Flip-flop**

**1. Write the Verilog code for clocked SR latch as shown in run-1 of this experiment using four NAND gates (Gate level modeling). Write the testbench also for all possible scenarios and also check for undefined case in**

**waveforms when both S = R = ‘1’. NOTE:- assign some propagation delay for NAND gates.**

**A: Verilog Code and testbench-** [**https://www.edaplayground.com/x/9c7B**](https://www.edaplayground.com/x/9c7B)

**Verilog:**

module sr\_latch(q, qbar, s, r, clk);

input s,r,clk;

output q, qbar;

wire nand1;

wire nand2;

nand #3 g1(nand1,clk,s);

nand #3 g2(nand2,clk,r);

nand #3 g3(q,nand1,qbar);

nand #3 g4(qbar,nand2,q);

endmodule

**Testbench:**

module testbench\_SR\_latch;

reg s,r,clk;

wire q,qb;

initial

begin

$dumpfile("dump.vcd");

$dumpvars(1,testbench\_SR\_latch);

clk = 1'b1;

repeat (4) #200 clk=~clk;

end

initial

begin

s=1'b0;

repeat (16) #50 s=~s;

end

initial

begin

r=1'b1;

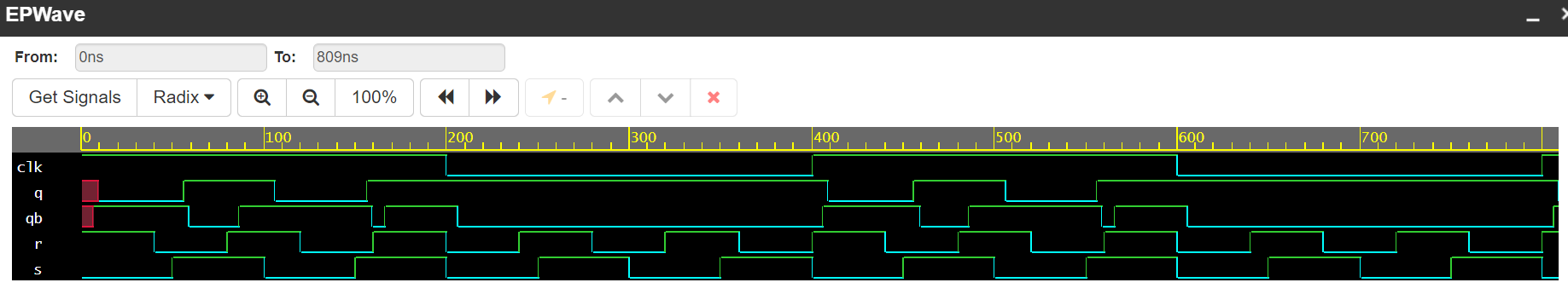
repeat (20) #40 r=~r;

end

sr\_latch U1(q, qb, s, r, clk);

endmodule

**Q:** Paste the screenshot of waveform window where you get the waveforms for the above code. (online students also paste the URL)

**A: **

**2.** Write Verilog code and testbench for clocked JK flip-flop and compare their response in waveforms. (please use exhaustive testbench).

**A: Verilog Code and testbench-** [**https://www.edaplayground.com/x/7Q6h**](https://www.edaplayground.com/x/7Q6h)

**Verilog:**

module jk\_latch(q, qbar, j, k, clk);

input j,k,clk;

output q, qbar;

wire nand1;

wire nand2;

nand #3 g1(nand1,clk,j);

nand #3 g2(nand2,clk,k);

nand #3 g3(q,nand1,qbar);

nand #3 g4(qbar,nand2,q);

endmodule

**Testbench:**

module testbench\_JK\_latch;

reg j,k,clk;

wire q,qb;

initial

begin

$dumpfile("dump.vcd");

$dumpvars(1,testbench\_JK\_latch);

clk = 1'b1;

repeat (4) #200 clk=~clk;

end

initial

begin

j=1'b0;

repeat (16) #50 j=~j;

end

initial

begin

k=1'b1;

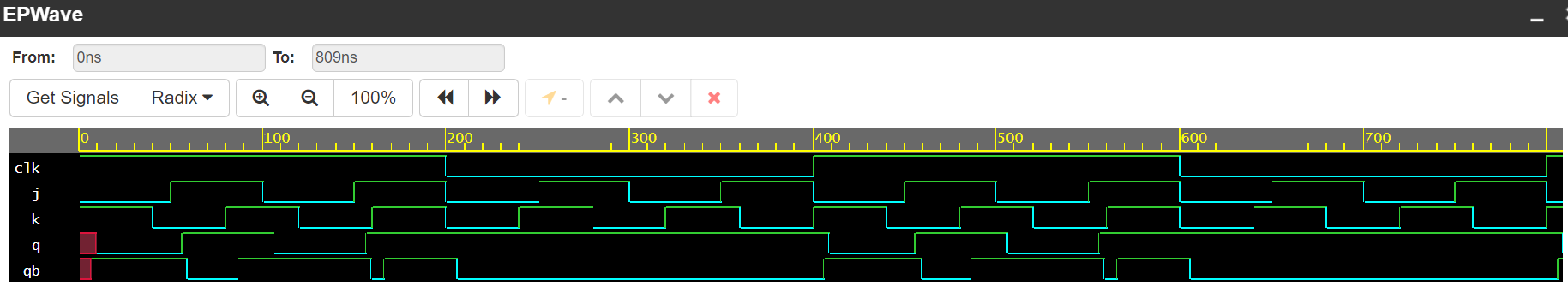
repeat (20) #40 k=~k;

end

jk\_latch U1(q, qb, j, k, clk);

endmodule

**Q:** Paste the screenshot of waveform window where you get the waveforms for the above code. (online students also paste the URL)

**A: **

**Run 5: D-Flipflop**

1. Write Verilog code and testbench for positive edge triggered D-Flip-flop with asynchronous set and reset.

**A: Verilog Code and testbench-** [**https://www.edaplayground.com/x/AKUw**](https://www.edaplayground.com/x/AKUw)

**Verilog:**

module d\_flipflop (input d, set, rst, clk, output reg q, qb);

always @ (posedge clk, negedge set, negedge rst)

begin

if (rst == 1'b0)

q <= 0; qb <=~q;

if (set == 1'b0)

q <= 1; qb <=0;

if (rst ==1'b1 & set ==1'b1)

q <= d; qb <=~q;

end

endmodule

**Testbench:**

module testbench\_D\_latch;

reg d,set,rst,clk;

wire q,qb;

initial

begin

$dumpfile("dump.vcd");

$dumpvars(1,testbench\_D\_latch);

clk = 1'b1;

repeat (4) #200 clk=~clk;

end

initial

begin

set=1'b0;

repeat (16) #50 set=~set;

end

initial

begin

rst=1'b1;

repeat (20) #40 rst=~rst;

end

initial

begin

d=1'b1;

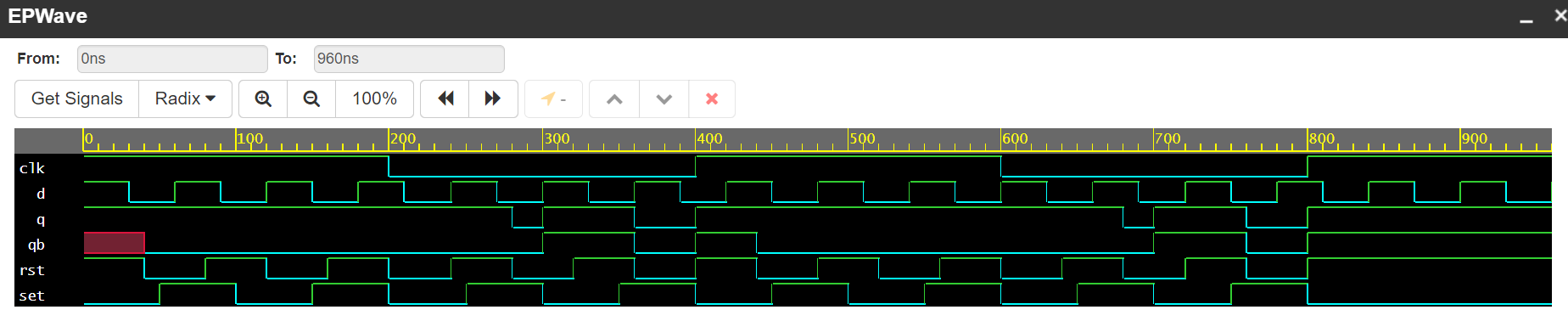
repeat (32) #30 d=~d;

end

d\_flipflop U1(d,set,rst,clk,q,qb);

endmodule

**Q:** Paste the screenshot of waveform window where you get the waveforms for the above code. (online students also paste the URL)

**A: **

**Assignment** All assignments are to be submitted strictly before start of next lab session through online only. Late assignments will not be entertained and will be awarded ‘0’ marks.

1. **Verilog code and testbench for T Flip-Flop for positive edge triggered.**

**Ans: Link1:**

**3. Identify the logic from the Verilog code below. (hint: Create testbench to identify).**

module circuit\_1 (input A,B, output C);

assign C = A ? B : C;

// ? : is the conditional operator (e.g. w=x ? y : z ; if x=true, then w=y if x =false then w=z)

endmodule

**Ans: Link2:**

**4. Identify the logic from the Verilog code below.** (hint: Create testbench to identify).

module circuit\_1 (input A,B, output C);

assign C = A ? B : C;

// ? : is the conditional operator (e.g. w=x ? y : z ; if x=true, then w=y if x =false then w=z)

endmodule

**Ans: Link3:**

**Self-Practice and self-evaluation**

1. Verilog code and testbench for D Flip-Flop for negative edge triggered.

2. Identify the logic for the code below by writing the testbench

module circuit\_2 (input D\_in, en, rst, output q);

assign q = !(rst ==1’b0) ? 0 : en ? D\_in : q;

endmodule

3. Identify the logic for the code below by writing the testbench

module circuit\_2 (output reg q, input d, en);

always @ (en, d)

if (en ==1’b1) q &lt;= d;

endmodule

4. Identify the logic for the code below by writing the testbench

module circuit\_3 (q, q\_bar, d, set, rst, clk);

input d, set, rst, clk;

output reg q;

output q\_bar;

assign q\_bar = !q;

always @ (posedge clk) // code enters here only at rising edge or positive edge of clock

// this also makes set and reset signals synchronous to clock edge only

if (rst == 1’b0) q &lt;= 0; // operator ‘&lt;=’ is the non-blocking assignment operator

else if (set == 1’b0) q&lt;=1;

else q &lt;= d;

endmodule