

Vellore Institute of Technology, Chennai

BECE407P - ASIC Design

Lab-1

Design and Simulation of sequential & combinational modules using Cadence® NCLaunch

Name of the Student: _____

Roll Number: _____

Date of the Lab. Class: 18/07/24

1. Aim:

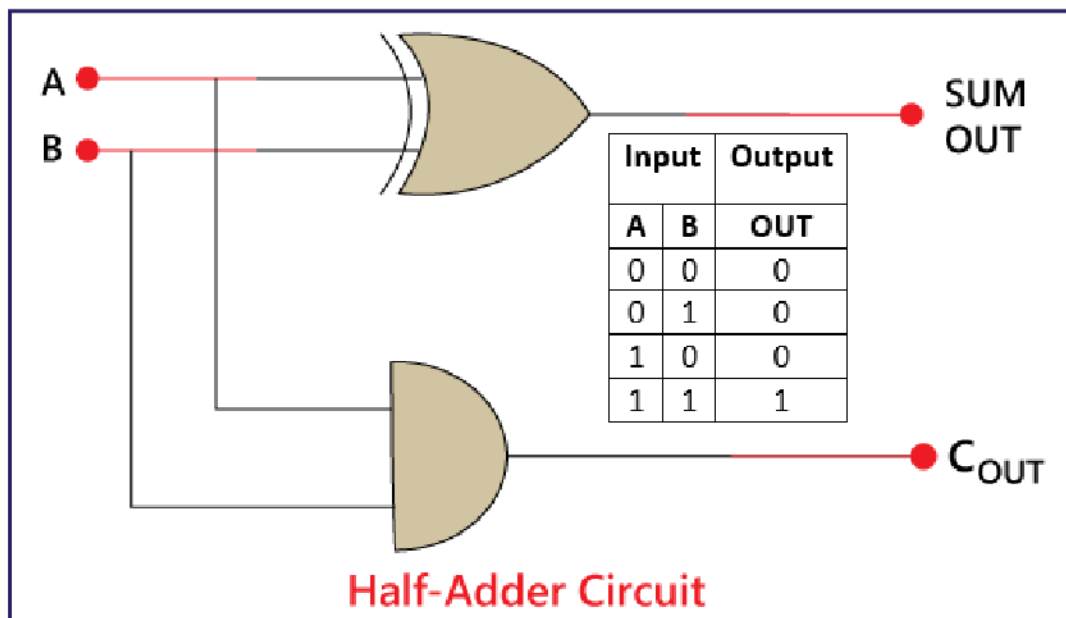
To design and verify the circuits(4-bit Up-Counter,1-bit Half Adder,1-bit Full Adder using 1-bit Half Adders, S-R Flip Flop, DFlip Flop using S-R Flip Flop) using Verilog in Cadence® NCLaunch.

2. EDA Tools Used:

Cadence® NCLaunch.

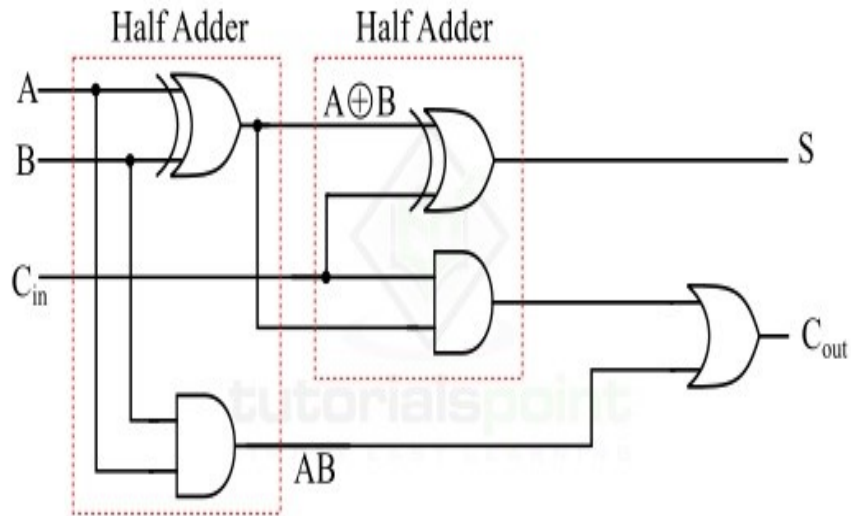
3. Detailed description of the Designs:

Half Adder :



A half adder is a simple digital circuit that performs the addition of two single-bit binary numbers, producing a sum and a carry as outputs. The half adder has two inputs, typically labeled A and B. It uses an XOR gate to generate the sum (S), which is true when either A or B is true, but not both. It also contains an AND gate to generate the carry (C), which is true only when both inputs are true.

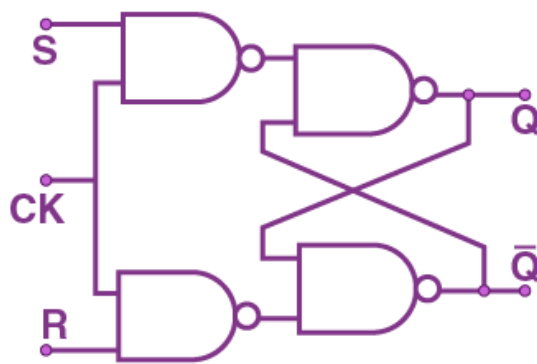
Full Adder:



Inputs			Outputs	
A	B	C – IN	Sum	C – Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A full adder is a digital circuit that adds three single bit to produce a sum and a carry output. It can be constructed using two half adders and an OR gate. The first half adder takes the two significant bits A and B, as inputs and produces a partial sum and a carry. The second half adder then takes this partial sum and the carry-in bit (C_{in}) as its inputs to produce the final sum and another carry. The two carry outputs from the half adders are then combined using an OR gate to produce the final carry-out (C_{out}).

SR Flip Flop:

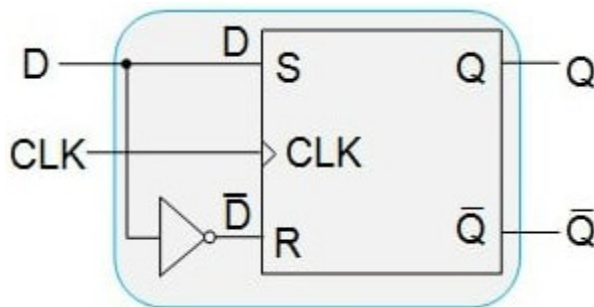


Truth Table

S	R	Q_N	Q_{N+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	-
1	1	1	-

An SR (Set-Reset) flip-flop has two inputs, S (Set) and R (Reset), and two outputs, Q and Qbar (the inverse of Q). When the Set input (S) is high and the Reset input is low, the flip-flop sets the output Q to 1. When the Reset input (R) is high (1) and the Set input (S) is low (0), it resets the Q output to 0. If both inputs are low (0), the flip-flop maintains its current state, preserving the stored bit. However, if both inputs are high (1), it results in an invalid state.

D Flip Flop:



Inputs		Outputs		Comments
D	CLK	Q	\bar{Q}	
1	↑	1	0	SET
0	↑	0	1	RESET

When designing a D flip-flop using an SR flip-flop, connect the D input directly to the S input of the SR flip-flop. The D input also goes through an inverter before being connected to the R input. When the clock (CLK) input is high, the D input value is latched and transferred to the output Q. If D is high (1), S becomes high and R becomes low (0) thereby making Q equal to 1; if D is low (0), S will be low while R will be high thus resetting Q back to 0. While the clock input is low, these outputs remain unchanged meaning that this type of flip flop retains its previous state.

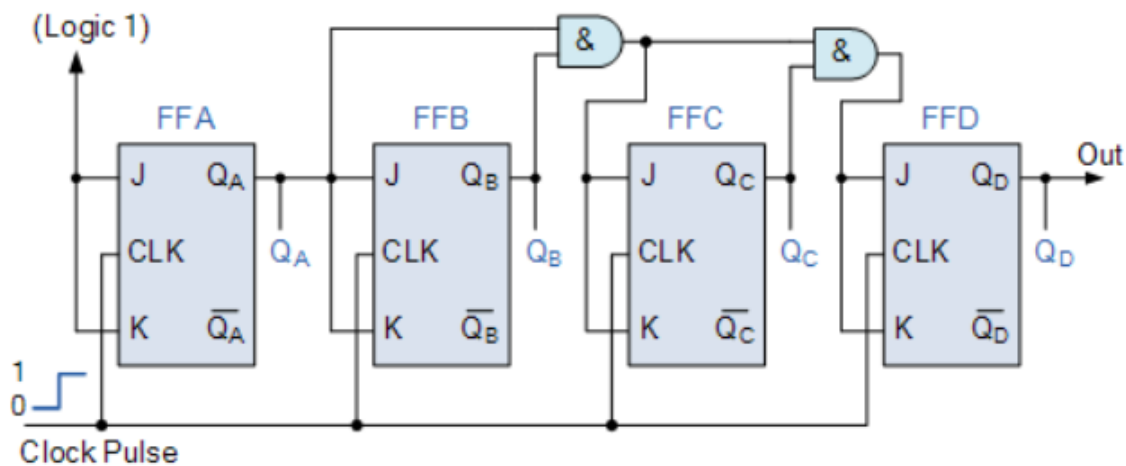
4bit Up counter:

It can be seen above, that the external clock pulses (pulses to be counted) are fed directly to each of the J-K flip-flops in the counter chain and that both the J and K inputs are all tied together in toggle mode, but only in the first flip-flop, flip-flop FFA (LSB) are they connected HIGH, logic "1" allowing the flip-flop to toggle on every clock pulse. Then the synchronous counter follows a predetermined sequence of states in response to the common clock signal, advancing one state for each pulse.

The J and K inputs of flip-flop FFB are connected directly to the output QA of flip-flop FFA, but

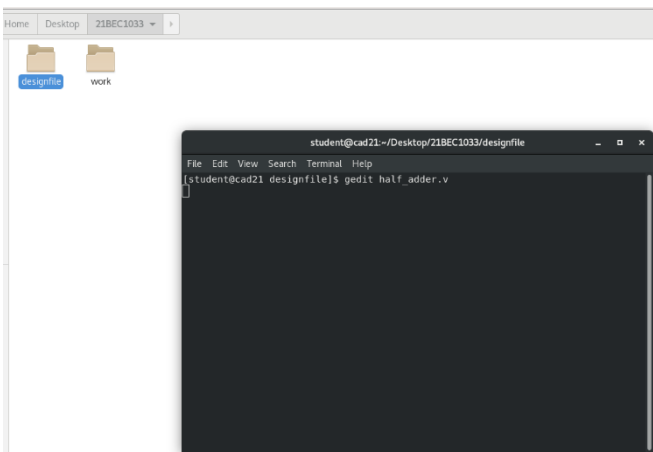
the J and K inputs of flip-flops FFC and FFD are driven from separate AND gates which are also supplied with signals from the input and output of the previous stage. These additional AND gates generate the required logic for the JK inputs of the next stage.

If we enable each JK flip-flop to toggle based on whether or not all preceding flip-flop outputs (Q) are "HIGH" we can obtain the same counting sequence as with the asynchronous circuit but without the ripple effect, since each flip-flop in this circuit will be clocked at exactly the same time. Then as there is no inherent propagation delay in synchronous counters, because all the counter stages are triggered in parallel at the same time, the maximum operating frequency of this type of frequency counter is much higher than that for a similar asynchronous counter circuit.



4. Procedure:

- i) Turn on the PC and select Red Hat 8.7 version.
- ii) Create a new folder (21BEC1033) in the desktop and inside this folder create two folder with the name designfiles and work.
- iii) Open the folder designfiles in terminal and type "gedit name.v" and press enter. Now type the Verilog code for the given logic and save it and close it.



```

half_adder.v
~/Desktop/21BEC1033/designfile

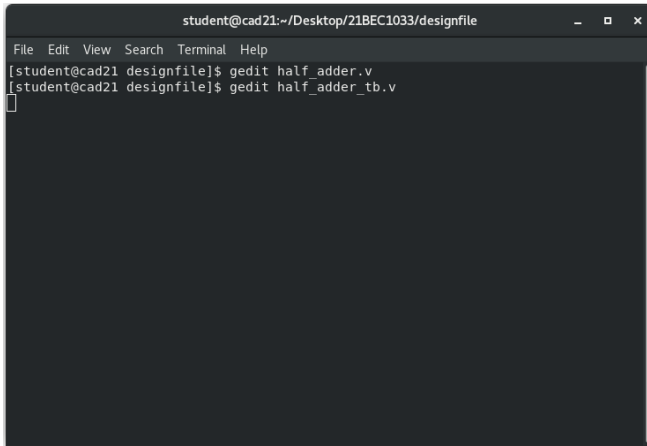
module half_adder (
    input a,b,
    output sum,carry
);

    assign sum = a ^ b;
    assign carry = a & b;

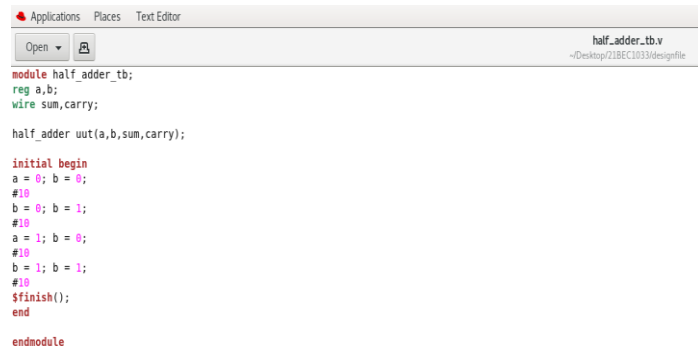
endmodule

```

- iv) In the same designfiles folder terminal now type “gedit name_tb.v” and press enter. Now type the testbench and save it.



```
student@cad21:~/Desktop/21BEC1033/designfile
File Edit View Search Terminal Help
[student@cad21 designfile]$ gedit half_adder.v
[student@cad21 designfile]$ gedit half_adder_tb.v
```



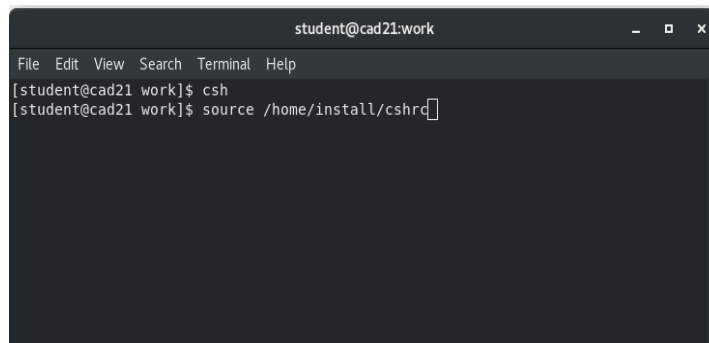
```
half_adder_tb.v
~/Desktop/21BEC1033/designfile

module half_adder_tb;
  reg a,b;
  wire sum,carry;

  half_adder uut(a,b,sum,carry);

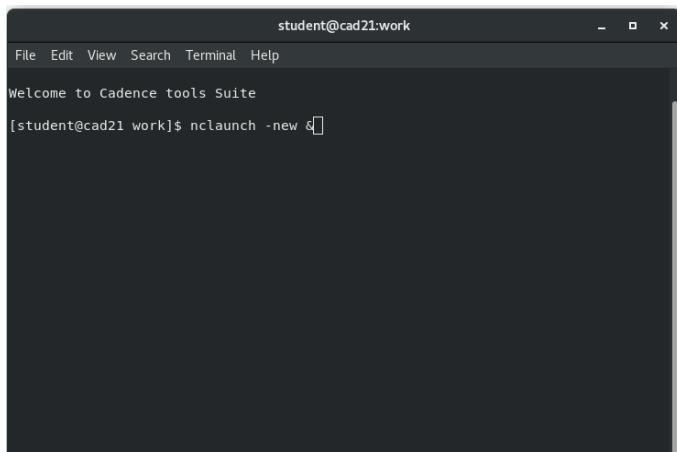
  initial begin
    a = 0; b = 0;
    #10
    b = 0; b = 1;
    #10
    a = 1; b = 0;
    #10
    b = 1; b = 1;
    #10
    $finish();
  end
endmodule
```

- v) Now open the work folder in terminal and invoke cs shell by typing the command “csh” . Then type “source /home/install/cshrc” and press enter to enter into cadence suite.

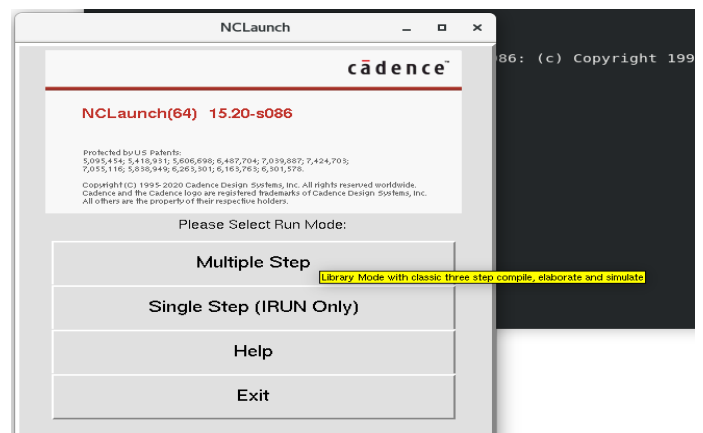


```
student@cad21:work
File Edit View Search Terminal Help
[student@cad21 work]$ csh
[student@cad21 work]$ source /home/install/cshrc
```

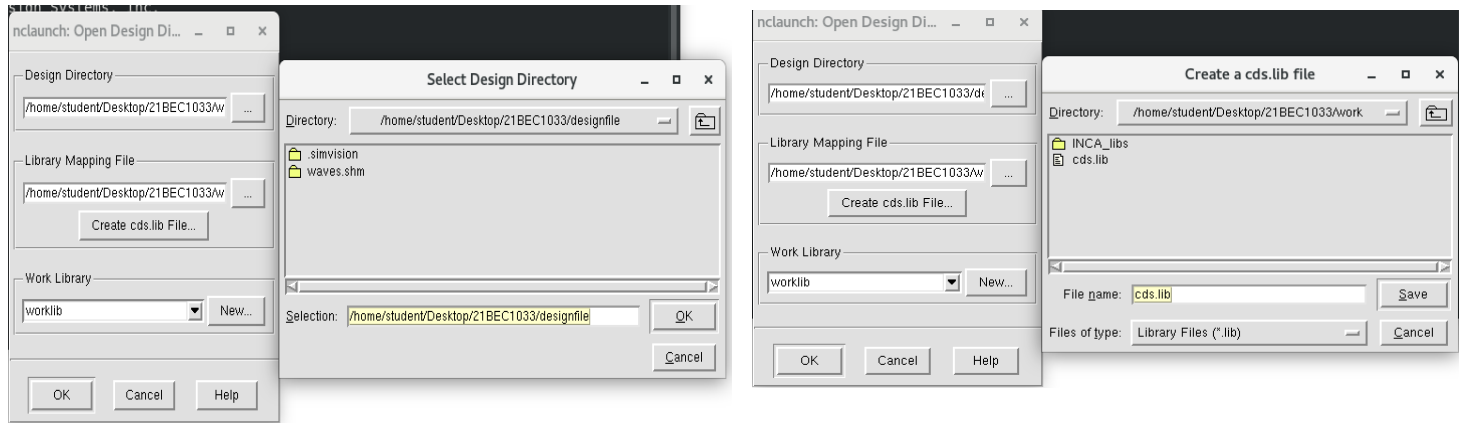
- vi) Now type “nclaunch -new &” and press enter to open the cadence nclaunch software. Select multiple steps option in the software.



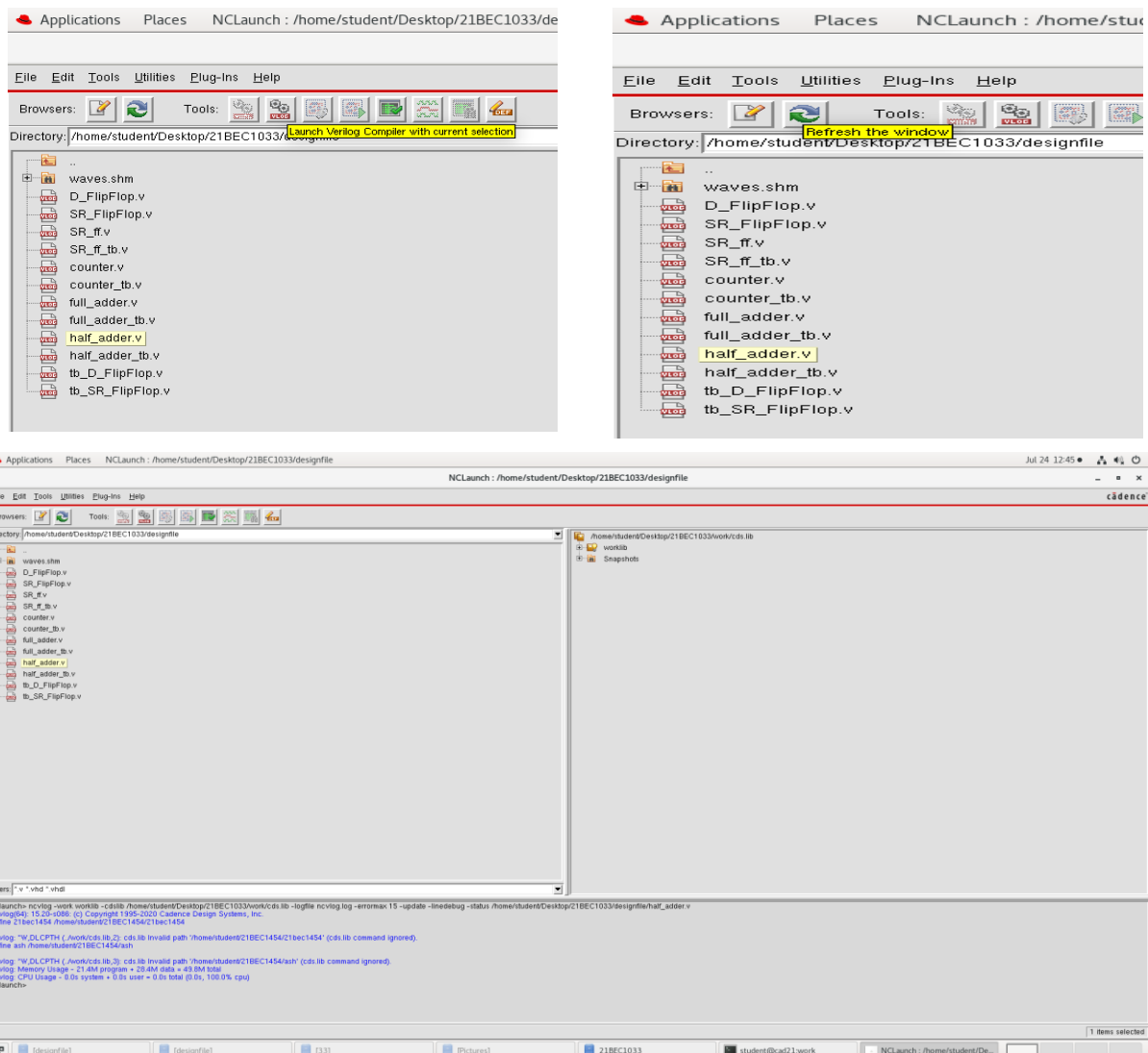
```
student@cad21:work
File Edit View Search Terminal Help
Welcome to Cadence tools Suite
[student@cad21 work]$ nclaunch -new &
```



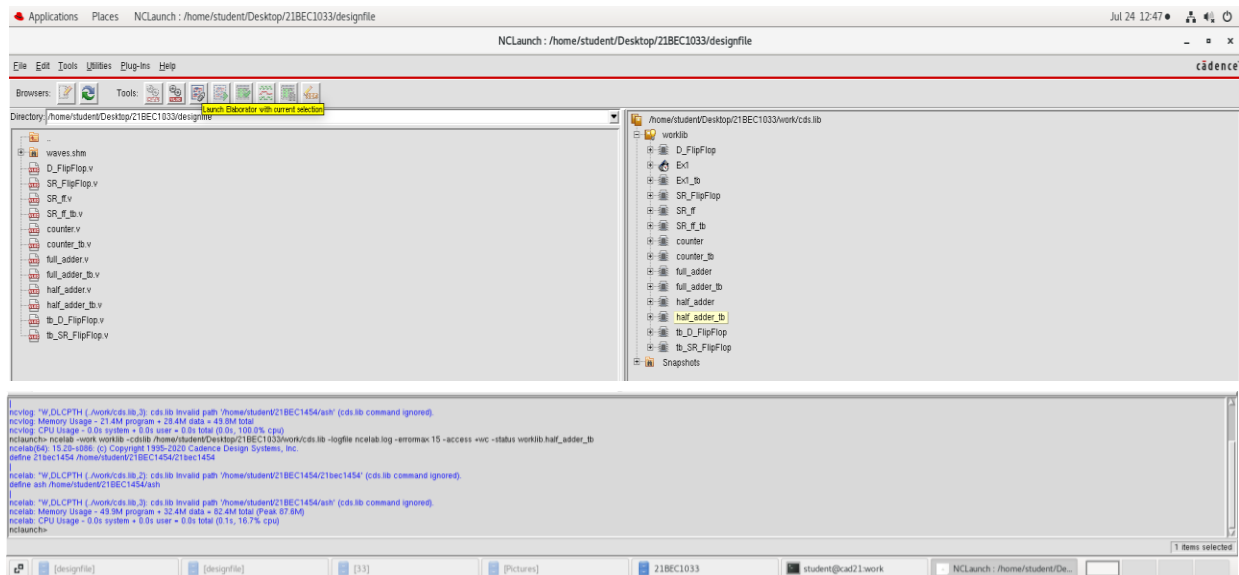
- vii) Choose the designfiles folder path in the design directory and select create cds.lib File option and choose the work folder path to save the cds.lib file. Click OK.



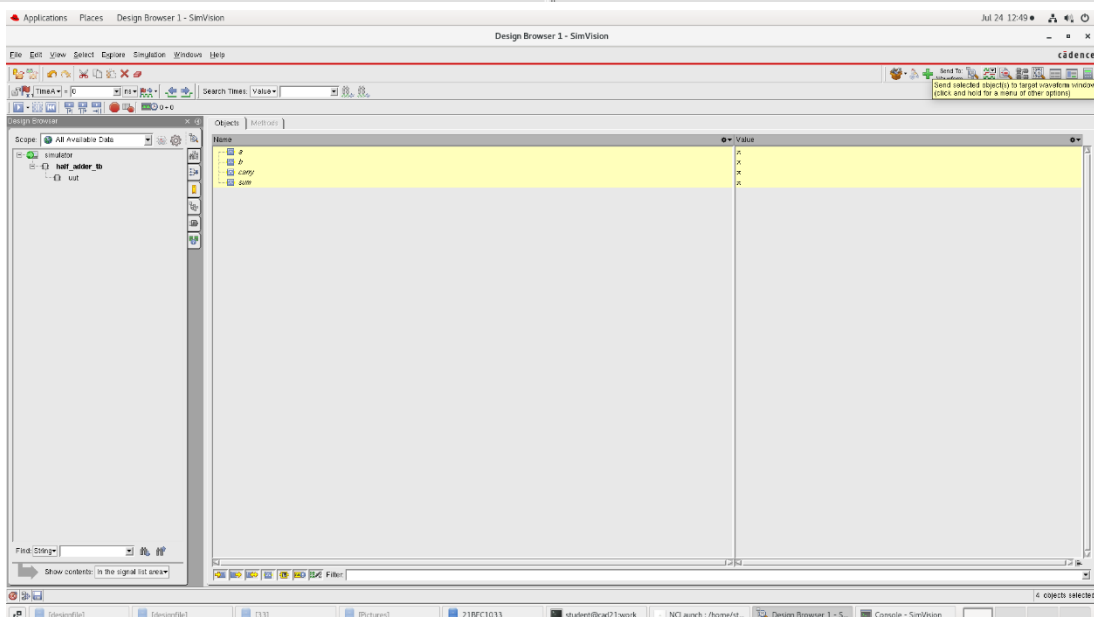
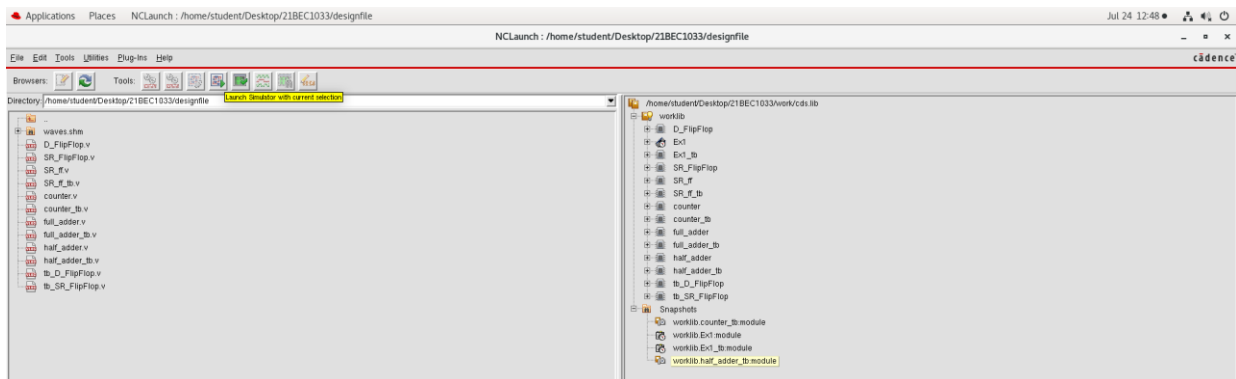
- viii) Select the “name.v” and compile it . If errors occurs then rectify the error in “name.v” and refresh the files in nclaunch and then compile the file again. Like wise compile the testbench file as well.



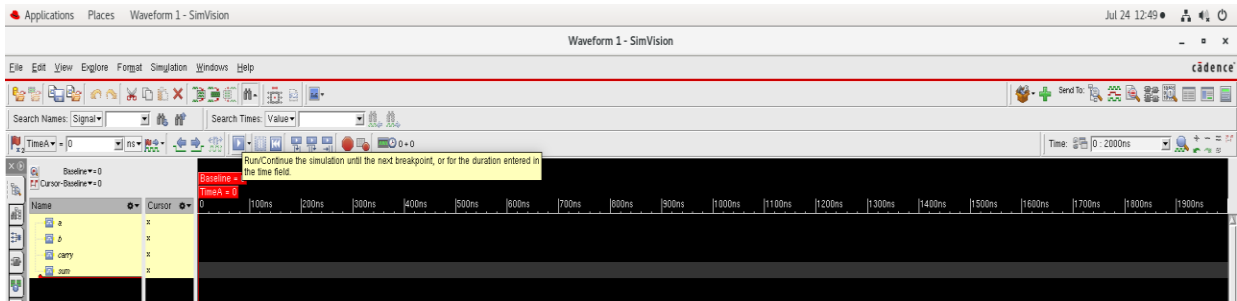
- ix) Select the testbenck file in worklib folder and launch the elaborator.



- x) Now select the testbench module in snapshots and launch the simulator. Click testbench in the simulator and select all the input and output ports of dut and send it to waveform window.



- xi) Now run simulation and stop it when required to obtain the waveforms.



- xii) Repeat the above steps for other circuit implementations.

5. RTL Codes:

Half Adder:

```
module half_adder ( input a,b,output sum,carry);
    assign sum = a ^ b;
    assign carry = a & b;
endmodule
```

Half Adder Testbench:

```
module half_adder_tb;
    reg a,b;
    wire sum,carry;
    half_adder uut(a,b,sum,carry);
    initial begin
        a = 0; b = 0;
        #10
        b = 0; b = 1;
        #10
        a = 1; b = 0;
        #10
        b = 1; b = 1;
        #10
        $finish();
    end
endmodule
```

Full Adder:

```
module full_adder( input a,b,cin,output sum,carry);
    wire c,c1,s;
    half_adder ha0(a,b,s,c);
```



```

half_adder ha1(cin,s,sum,c1);
assign carry = c | c1 ;
endmodule

```

Full Adder Testbench:

```

module full_adder_tb;
reg a,b,cin;
wire sum,carry;
full_adder uut(a,b,cin,sum,carry);
initial begin
a = 0; b = 0; cin = 0;
#10
a = 0; b = 0; cin = 1;
#10
a = 0; b = 1; cin = 0;
#10
a = 0; b = 1; cin = 1;
#10
a = 1; b = 0; cin = 0;
#10
a = 1; b = 0; cin = 1;
#10
a = 1; b = 1; cin = 0;
#10
a = 1; b = 1; cin = 1;
#10
$finish();
end
endmodule

```

SR FlipFlop:

```

module sr_flip_flop (
input wire s,
input wire r,
input wire clk,
output reg q,
output reg q_bar
);
always @(posedge clk) begin
if (s && ~r)
q <= 1;
else if (~s && r)
q <= 0;

```

```

else if (s && r)
q <= 1'bx;
q_bar <= ~q;
end
endmodule

```

SR FlipFlop Testbench:

```

module tb_sr_flip_flop;
reg s;
reg r;
reg clk;
wire q;
wire q_bar;
sr_flip_flop uut (
.s(s),
.r(r),
.clk(clk),
.q(q),
.q_bar(q_bar)
);
initial begin
clk = 0;
s = 0;
r = 0;
#10 s = 1; r = 0;
#10 s = 0; r = 1;
#10 s = 1; r = 1;
#10 s = 0; r = 0;
#10 $finish;
end
always #5 clk = ~clk;
endmodule

```

D FlipFlop:

```

module d_flip_flop (
input wire d,
input wire clk,
output wire q,
output wire q_bar
);
wire s, r;
assign s = d;
assign r = ~d;
sr_flip_flop sr_ff (
.s(s),

```

```

.r(r),
.clk(clk),
.q(q),
.q_bar(q_bar)
);
endmodule

```

D FlipFlop Testbench:

```

module tb_d_flip_flop;
  reg d;
  reg clk;
  wire q;
  wire q_bar;
  d_flip_flop uut (
    .d(d),
    .clk(clk),
    .q(q),
    .q_bar(q_bar)
  );
  initial begin
    clk = 0;
    d = 0;
    #10 d = 1;
    #10 d = 0;
    #10 d = 1;
    #10 d = 0;
    #10 $finish;
  end
  always #5 clk = ~clk;
endmodule

```

4 bit UP Counter:

```

module counter(q, clk, reset);
  input clk, reset;
  output reg [3:0] q;

  always @(posedge clk)
  begin
    if (reset)
    begin
      q <= 0;
    end
  end
endmodule

```

```

else
begin
    q <= q + 1;
end
end
endmodule

```

4 bit UP Counter Testbench:

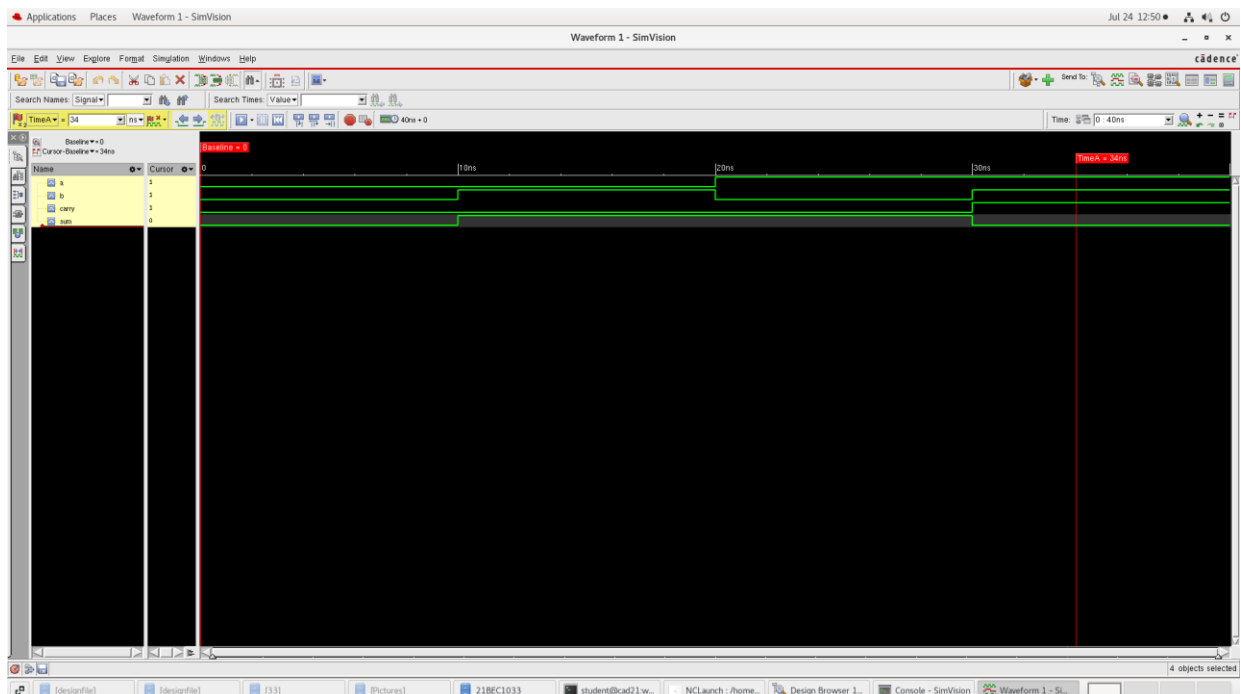
```

module counter_tb();
reg clk,reset; wire [3:0]q;
counter uut(q,clk,reset);
initial
begin
reset=1'b1; clk=1'b0;
#10; reset=1'b0;
#500;
end
always begin
end
#5 clk=~clk;
endmodule

```

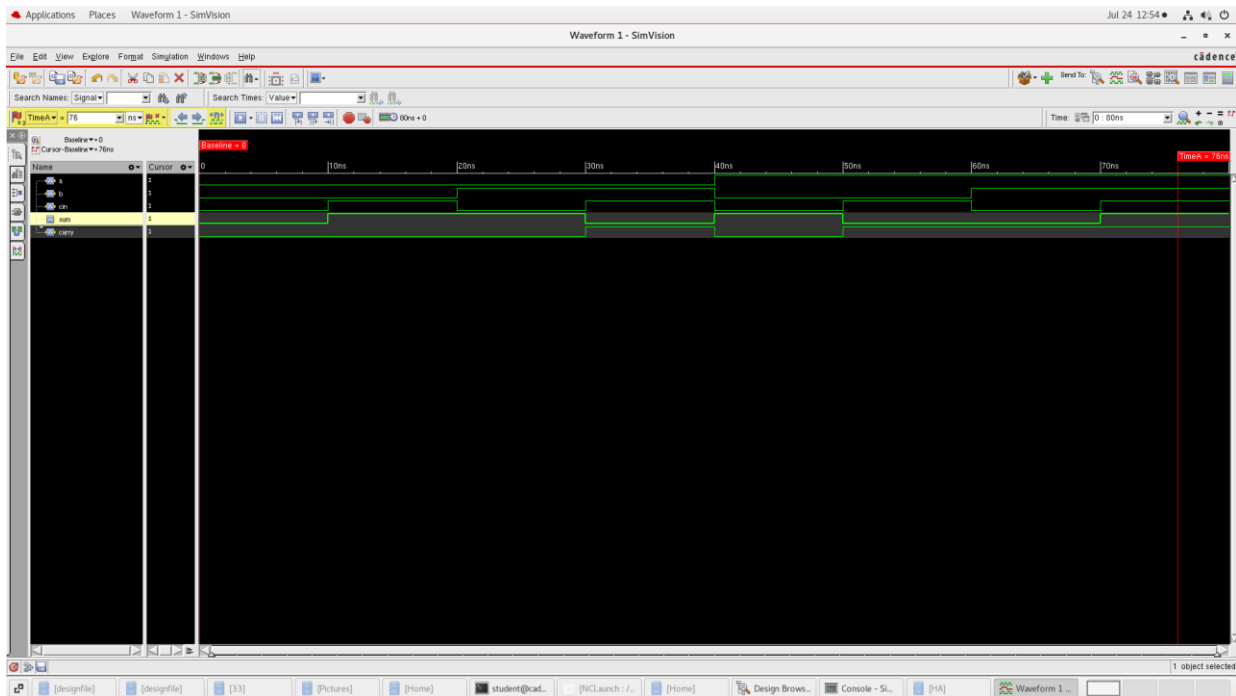
6. Observations:

Half Adder :



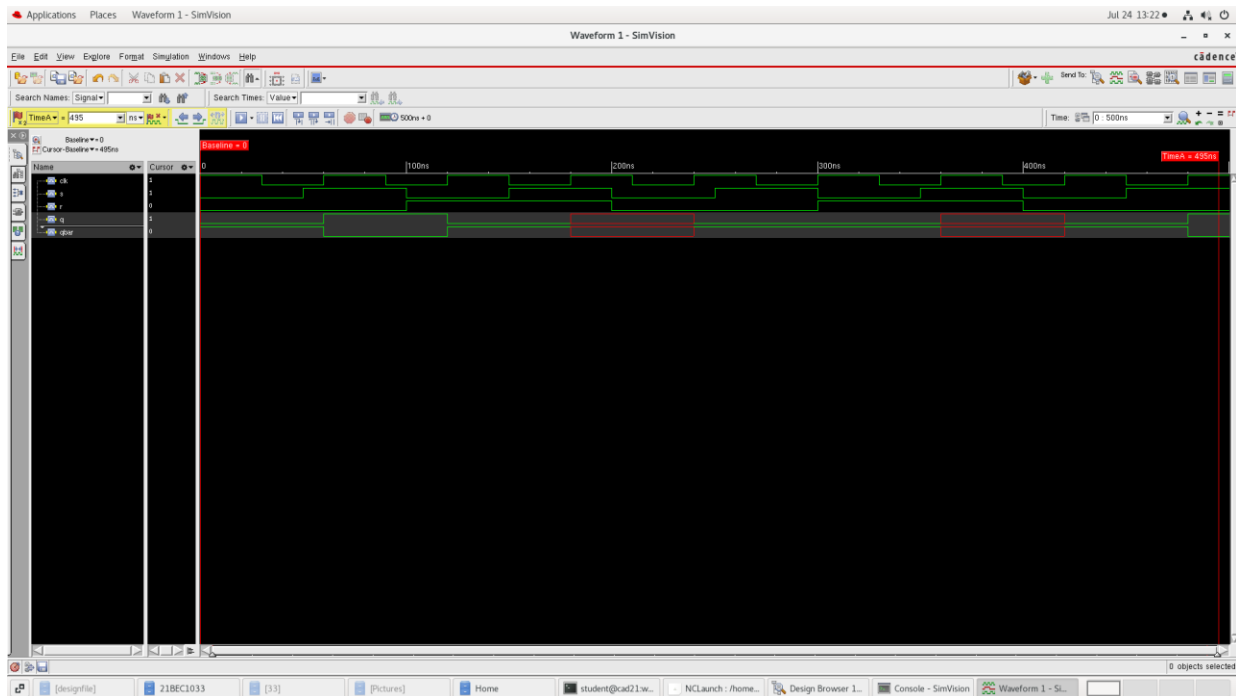
When one of the input is high and the other is low the sum is found to high and carry is found to be low. If both the inputs are high the sum is found to be low and carry is found to be high . If both the inputs are low the sum and carry are found to be low.

Full Adder:



When all inputs are 0 (A=0, B=0, C-IN=0): Sum = 0 , C-OUT = 0
When A=0, B=0, C-IN=1: Sum = 1 (since $0 + 0 + 1 = 1$), C-OUT = 0
When A=0, B=1, C-IN=0: Sum = 1 (since $0 + 1 + 0 = 1$), C-OUT = 0
When A=0, B=1, C-IN=1: Sum = 0 (since $0 + 1 + 1 = 10$), C-OUT = 1
When A=1, B=0, C-IN=0: Sum = 1 (since $1 + 0 + 0 = 1$), C-OUT = 0
When A=1, B=0, C-IN=1: Sum = 0 (since $1 + 0 + 1 = 10$), C-OUT = 1
When A=1, B=1, C-IN=0: Sum = 0 (since $1 + 1 = 10$), C-OUT = 1
When A=1, B=1, C-IN=1: Sum = 1 (since $1 + 1 + 1 = 11$), C-OUT = 1

SR Flip Flop:



When $S=0$ and $R=0$:

- The flip-flop maintains its current state (Q_n). There is no change in the output.

When $S=0$ and $R=1$:

- The flip-flop is reset, and the output Q becomes 0.

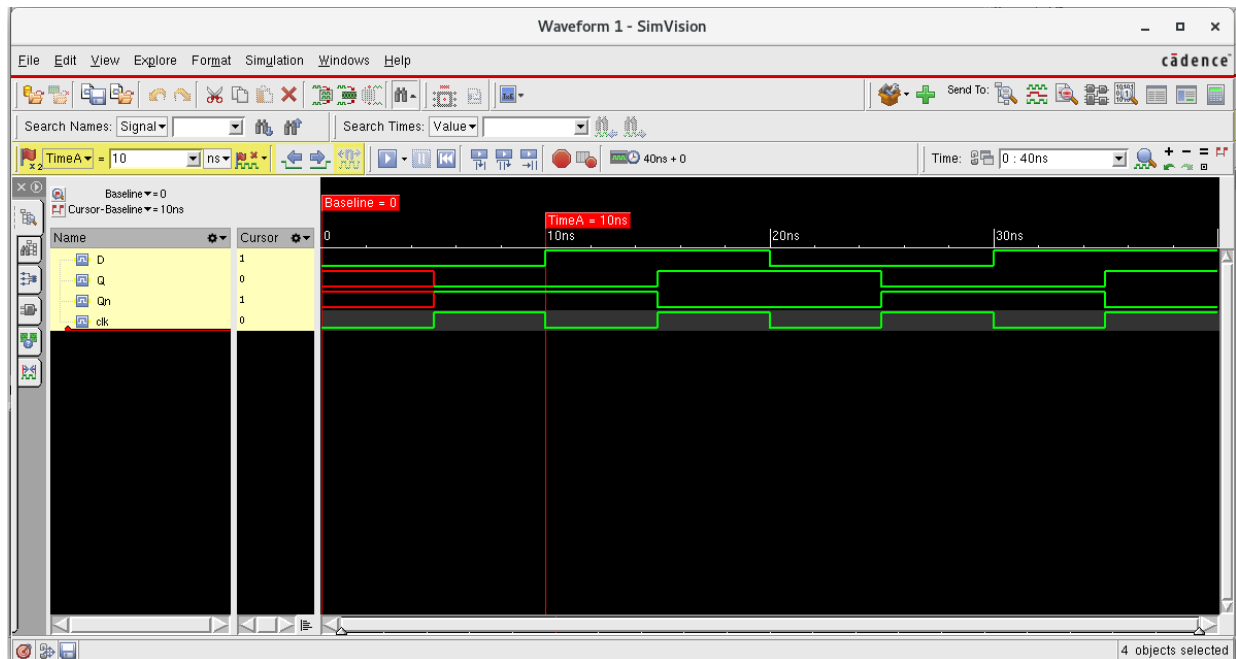
When $S=1$ and $R=0$:

- The flip-flop is set, and the output Q_n becomes 1.

When $S=1$ and $R=1$:

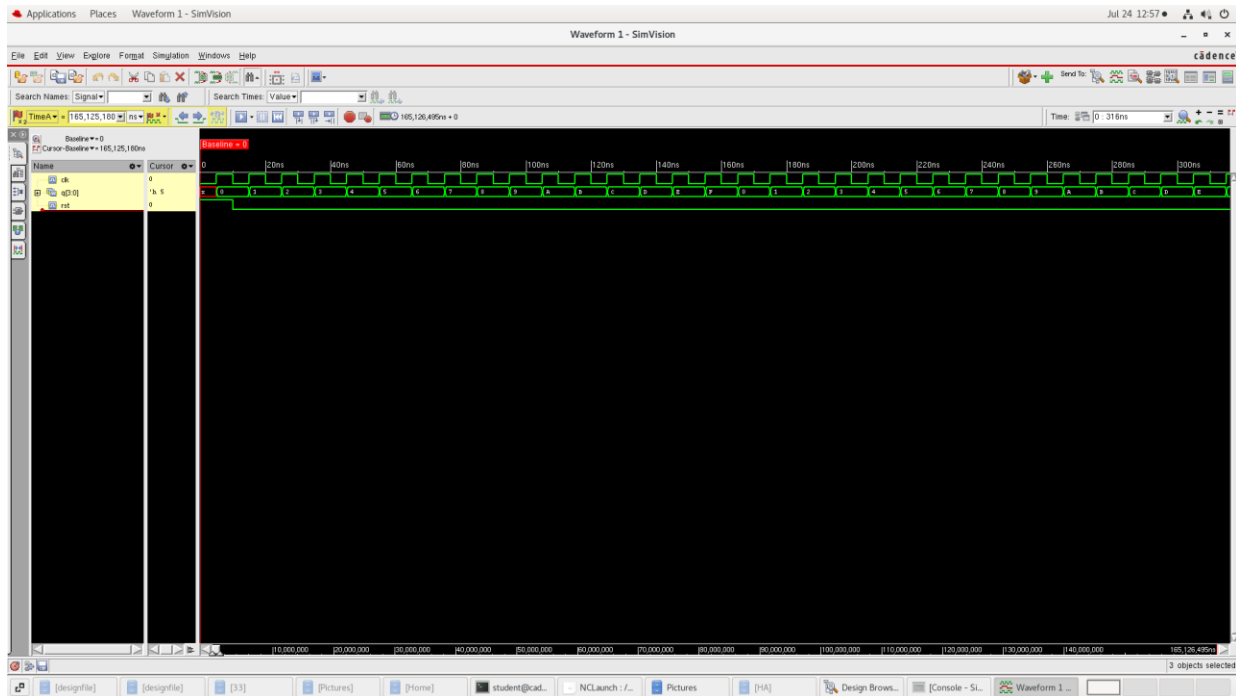
- This condition is undefined (represented by x).

D Flip Flop:



The value of D will be the output Q when a positive edge of the clock occurs and complement of Q will be \bar{Q} .

4bit UP Counter:



Whenever a posedge is encountered and the reset is low the value stored in q is incremented by 1 . Since it is 4 bit counter it can count the values from 0 to F(15) . Once q reaches its maximum count value it again starts to count from zero. When the reset is high the value stored in q becomes 0.

7. Inference:

From the waveforms obtained for the different circuits, it is observed that it is matching with the theoretical concepts. Hence any sequential and combinational circuits can be designed and verified using Cadence NCLaunch software.