VELLORE INSTITUTE OF TECHNOLOGY, CHENNAI

BECE407P – ASIC DESIGN

LAB-1

DESIGN AND SIMULATION OF SEQUENTIAL AND COMBINATIONAL MODULES USING CADENCE NCLaunch

Name of the Student: Sudharsan S

Roll Number: 21BLC1079

Date of the Lab: 18-07-2024

AIM:

The main goals of this experiment are to:

i) Develop Verilog HDL code for essential circuits, including an up-counter, a 1-bit half adder, a full adder using half adders, an SR flip-flop, and a D flip-flop using an SR flip-flop.

ii) Introduce and gain proficiency with the Cadence NC launch tool for running simulations.

EDA TOOLS USED:

For this experiment, Cadence NC launch is the primary EDA tool, focusing on the functional verification of Verilog modules. The key tools include:

Ncvlog , ncelab , ncsim , ncsimwave and ncverilog .

DETAILED DESCRIPTION OF THE DESIGNS:

4-BIT UPCOUNTER:

This sequential circuit design functions as a binary counter with a 4-bit output.

Inputs: It features a clock signal and an active-high reset signal. The clock manages the counting sequence, while the reset ensures the counter initializes correctly.

Functionality: The 4-bit output counts from 0 to 15, incrementing by one with each clock pulse. Once the count reaches 15, it wraps around to 0. If the reset signal is activated, the counter immediately resets to 0.

1-BIT HALF ADDER:

A half adder is used to add two single-bit binary numbers, producing a sum and a carry output.

Inputs/Outputs: It has two inputs, labelled 'a' and 'b', and two outputs, 'sum' and 'carry'.

Components: It is typically constructed using an XOR gate for generating the sum and an AND gate for the carry output.

Truth Table: The half adder's truth table shows how the outputs relate to the binary inputs.

Input		Output	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

<u>1-BIT FULL ADDER:</u>

This circuit extends the half adder by adding support for an additional carry input, allowing it to handle multi-bit binary addition.

Inputs/Outputs: It has three inputs: 'a', 'b', and 'carry-in', and two outputs: 'sum' and 'carry-out'.

Components: Usually built using two half adders and an OR gate to manage carry propagation through multiple bits.

Truth Table: The full adder's truth table illustrates the output combinations for various input scenarios, including the carry input and output.

Input			**Output**	
A	**B**	**C_{in}**	**Sum**	**Carry**
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

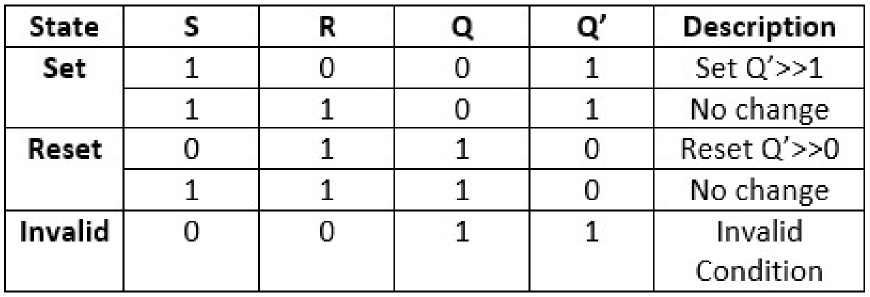
<u>S-R FLIPFLOP:</u>

The SR flip-flop is a basic memory element used to store a single bit of data.

Inputs/Outputs: It has three inputs: Set, Reset, and Clock, with two outputs: Q and $\overline{Q}$.

Functionality: This flip-flop changes its output state based on the Set and Reset signals, triggered by the clock pulse. It is widely used in digital circuits such as counters and registers.

Truth Table: The SR flip-flop's truth table details how the state of the output's changes with different combinations of the inputs.



State	S	R	Q	Q'	Description
Set	1	0	0	1	Set Q'>>1
	1	1	0	1	No change
Reset	0	1	1	0	Reset Q'>>0
	1	1	1	0	No change
Invalid	0	0	1	1	Invalid Condition

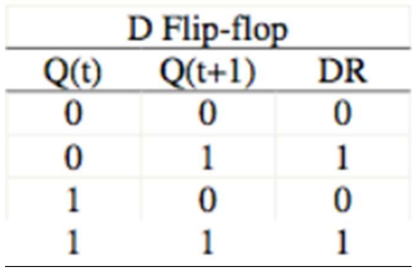
<u>D-FLIPFLOP:</u>

The D flip-flop is a type of flip-flop that stores a single bit of data, using a single data input.

Inputs/Outputs: It has two inputs: Data (D) and Clock, and two outputs: Q and $\bar{Q}$.

Functionality: It captures the value of the Data input on the rising edge of the Clock signal and maintains this value until the next clock edge.

Excitation Table: The excitation table shows how the Data input influences the output state, depending on the clock edge.

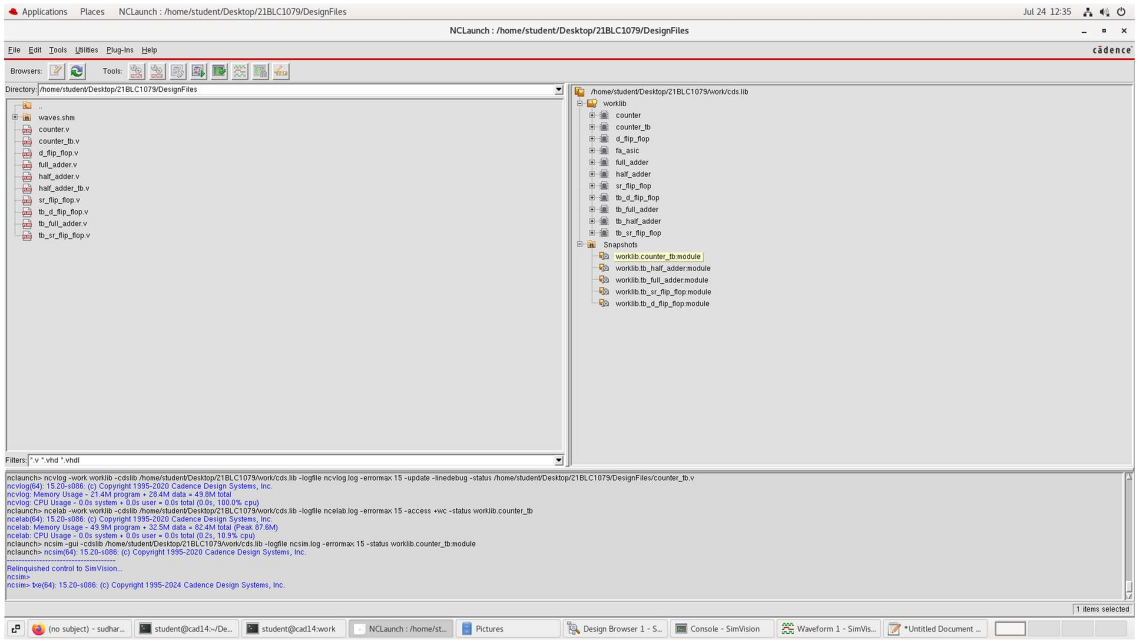


D Flip-flop

Q(t)	Q(t+1)	DR
0	0	0
0	1	1
1	0	0
1	1	1

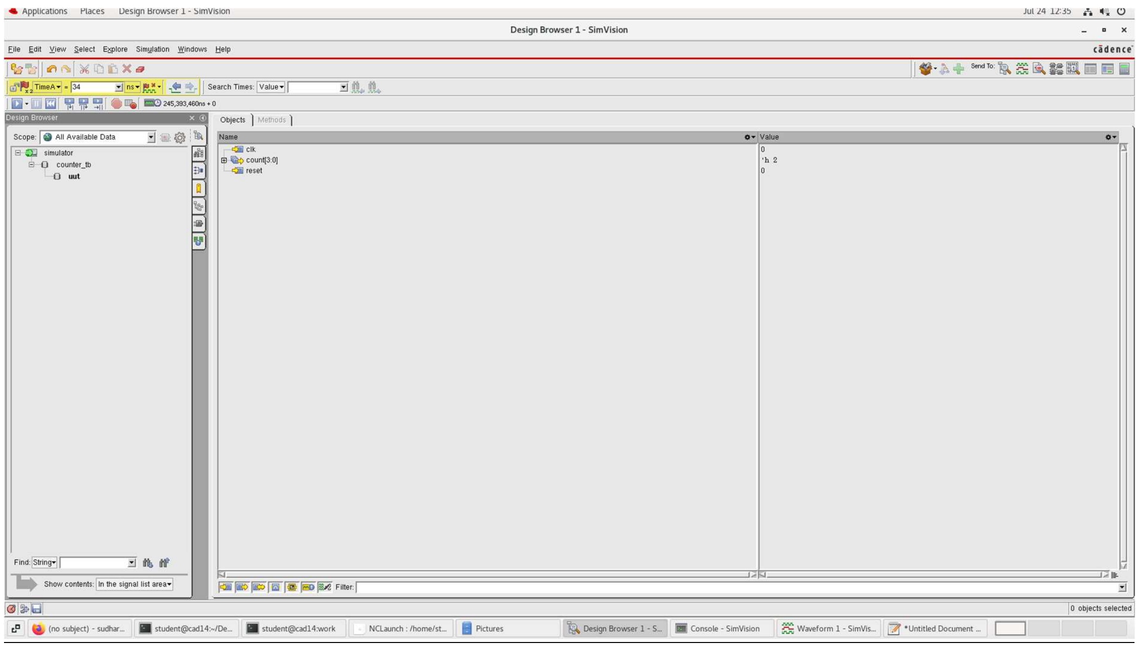
<u>PROCEDURE</u>:

- Create two directories named DesignFiles and work.
- In the DesignFiles folder, create a Verilog file with a. v extension using the gedit command to write the RTL codes.
- Create two files: one containing the Verilog code for the circuit and another for the testbench.
- Navigate to the work folder and use the csh command to launch the shell.
- Source the environment file using the command source /home/install/cshrc.
- Start NClaunch with the command nclaunch -new &.
- In NClaunch, select the multistep option and set your design directory to the DesignFiles folder.
- Set the work folder in the cds.lib file.
- Select the Verilog files from the DesignFiles folder.



```
nclaunch> ncvlog -work worklib -cdslib /home/student/Desktop/21BLC1079/work/cds.lib -logfile ncvlog.log -errormax 15 -update -linedebug -status /home/student/Desktop/21BLC1079/DesignFiles/counter_tb.v
ncvlog(64): 15.20-s086: (c) Copyright 1995-2020 Cadence Design Systems, Inc.
ncvlog: Memory Usage - 21.4M program + 28.4M data = 49.8M total
ncvlog: CPU Usage - 0.0s system + 0.0s user = 0.0s total (0.0s, 100.0% cpu)
nclaunch> ncelab -work worklib -cdslib /home/student/Desktop/21BLC1079/work/cds.lib -logfile ncelab.log -errormax 15 -access +wc -status worklib.counter_tb
ncelab(64): 15.20-s086: (c) Copyright 1995-2020 Cadence Design Systems, Inc.
ncelab: Memory Usage - 49.9M program + 32.5M data = 82.4M total (Peak 87.6M)
ncelab: CPU Usage - 0.0s system + 0.0s user = 0.0s total (0.2s, 10.9% cpu)
nclaunch> ncsim -gui -cdslib /home/student/Desktop/21BLC1079/work/cds.lib -logfile ncsim.log -errormax 15 -status worklib.counter_tb:module
nclaunch> ncsim(64): 15.20-s086: (c) Copyright 1995-2020 Cadence Design Systems, Inc.
-----------------------------------
Relinquished control to SimVision...
ncsim>
ncsim> txe(64): 15.20-s086: (c) Copyright 1995-2024 Cadence Design Systems, Inc.
```

- Use the vlob tool to check the code for errors, selecting both the circuit code and the testbench file.
- Compile the .v files and resolve any compilation errors.
- Open the worklib directory, select the testbench, and click on Launch Elaborator.
- Under Snapshots, select the testbench file and launch the simulator with the current simulation settings.
- Select the inputs and outputs to be displayed and open the waveform window.

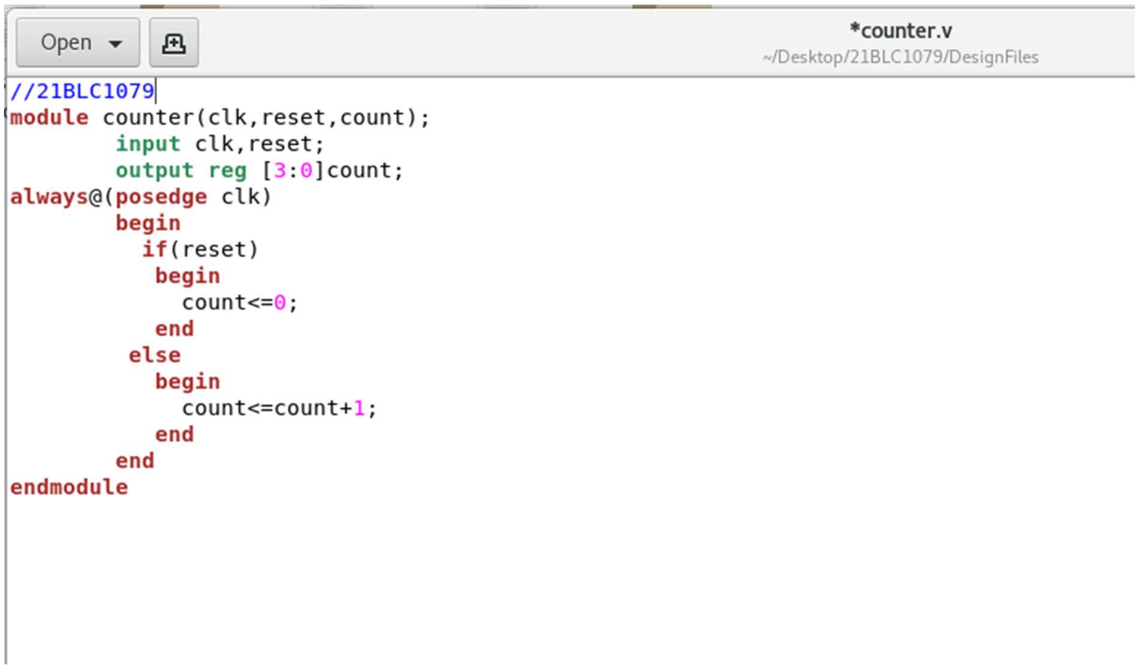


Name	Value
clk	0
count[3:0]	'h 2
reset	0

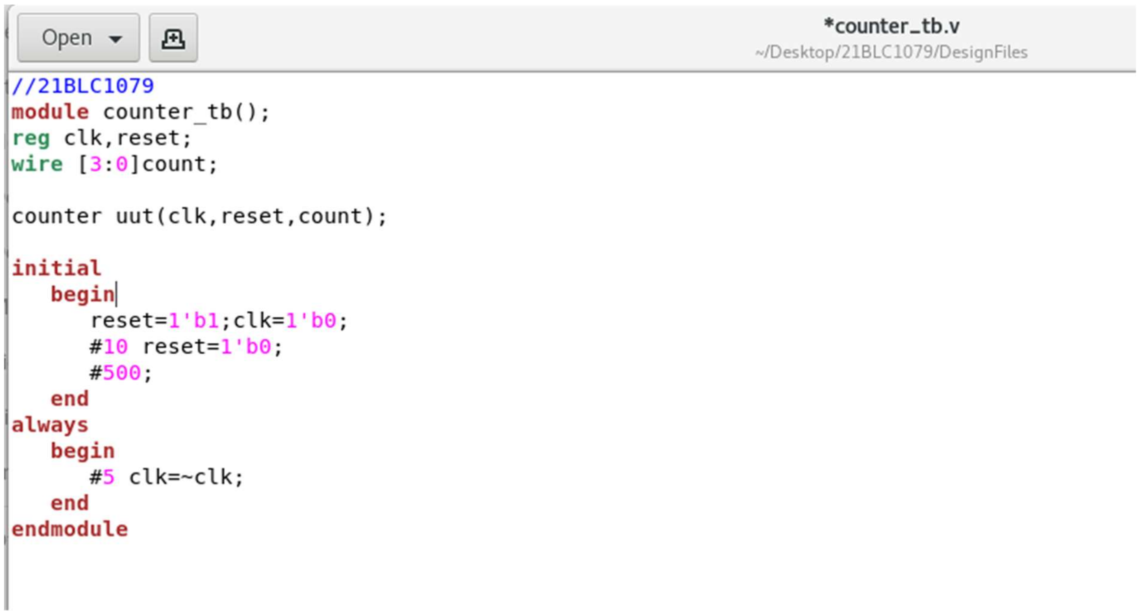
- Run the simulation and verify the waveform against the truth table.
- Save your work and simulation results for future reference.

RTL CODES:

4-bit Up Counter and its Test Bench:

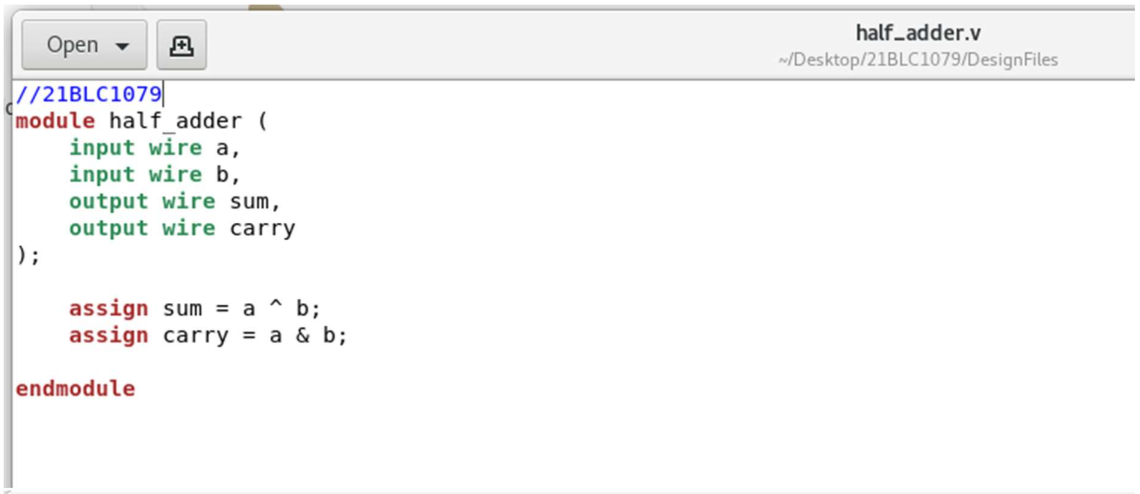

Open
*counter.v
~/Desktop/21BLC1079/DesignFiles

```
//21BLC1079
module counter(clk,reset,count);
        input clk,reset;
        output reg [3:0]count;
always@(posedge clk)
        begin
          if(reset)
           begin
             count<=0;
           end
         else
           begin
             count<=count+1;
           end
        end
endmodule
```


Open
*counter_tb.v
~/Desktop/21BLC1079/DesignFiles

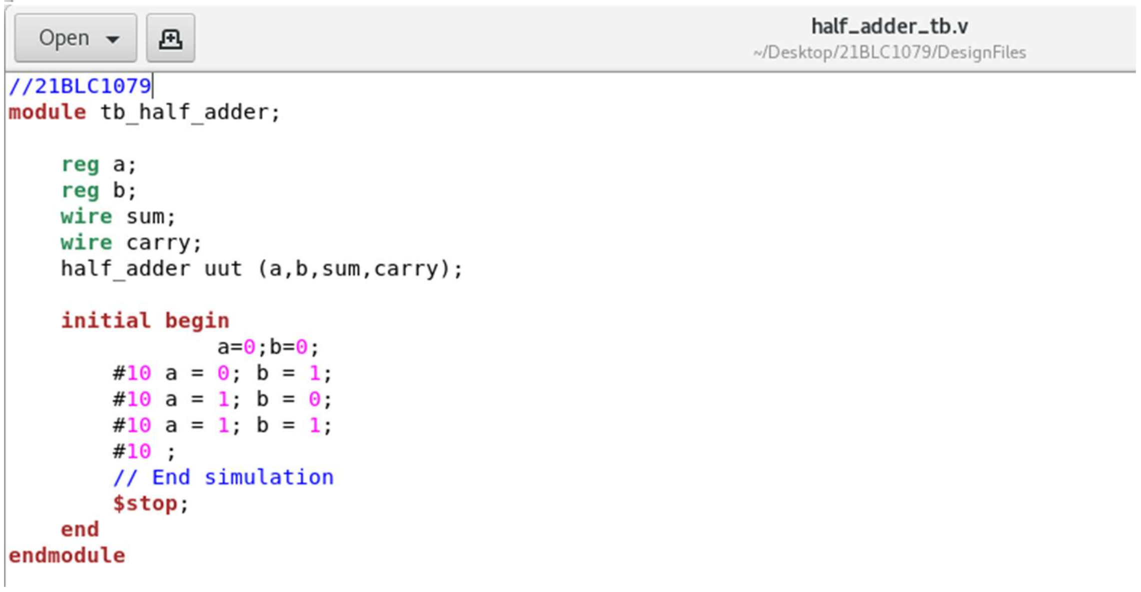
```
//21BLC1079
module counter_tb();
reg clk,reset;
wire [3:0]count;

counter uut(clk,reset,count);

initial
   begin
      reset=1'b1;clk=1'b0;
      #10 reset=1'b0;
      #500;
   end
always
   begin
      #5 clk=~clk;
   end
endmodule
```

1-Bit Half Adder and its Test Bench:



Open | half_adder.v ~/Desktop/21BLC1079/DesignFiles

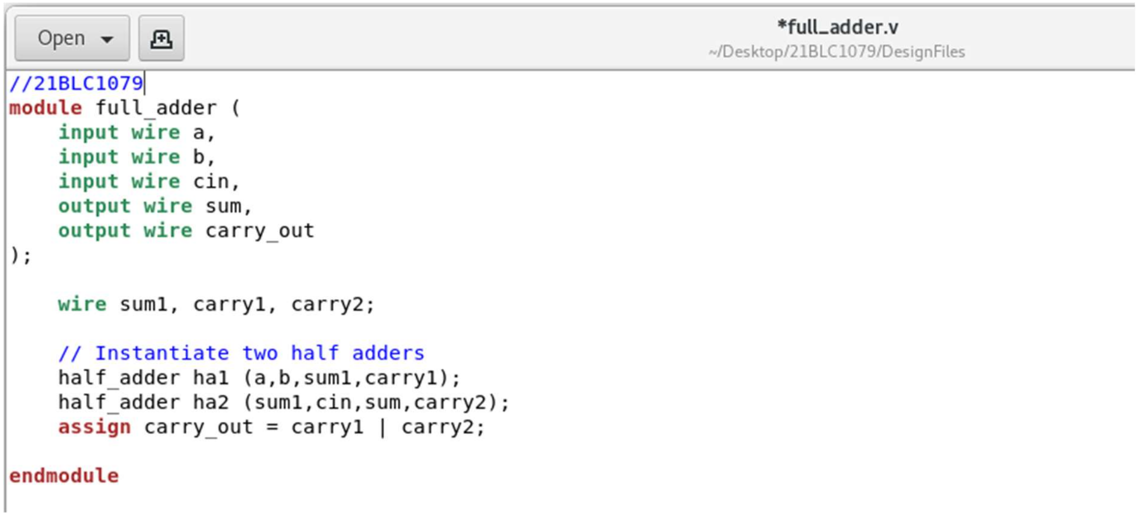
```
//21BLC1079
module half_adder (
    input wire a,
    input wire b,
    output wire sum,
    output wire carry
);

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

endmodule
```



Open | half_adder_tb.v ~/Desktop/21BLC1079/DesignFiles

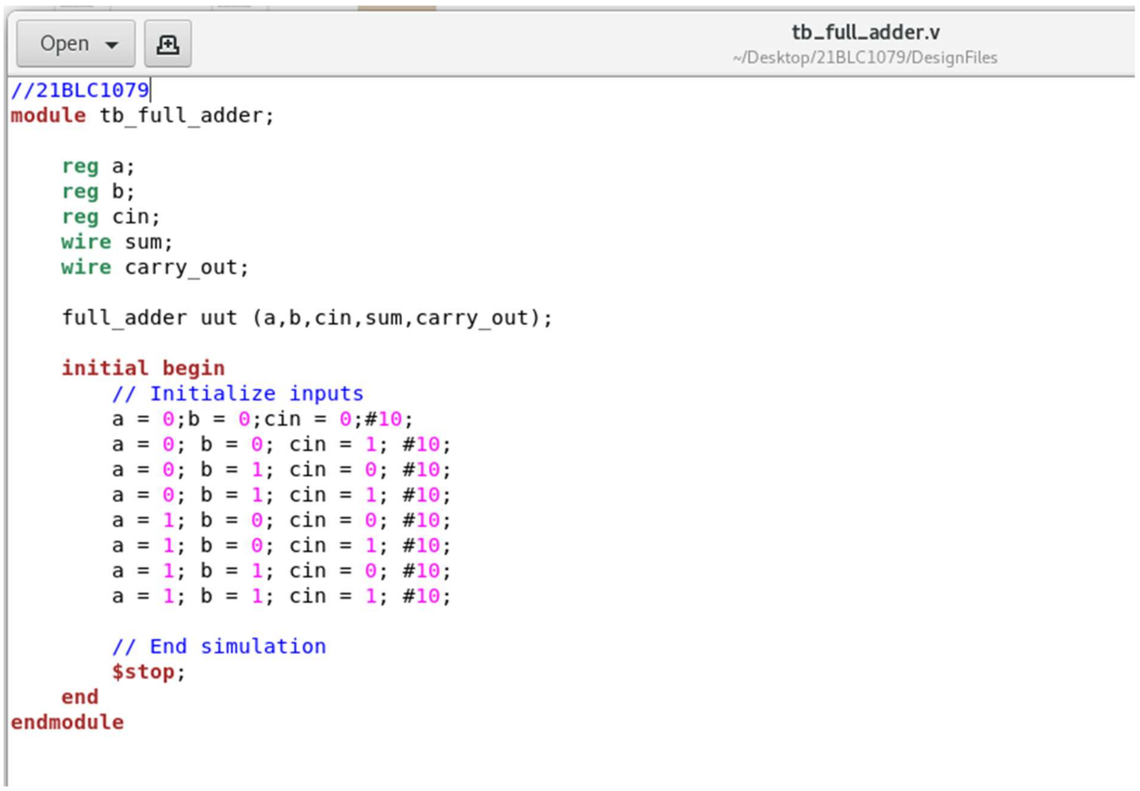
```
//21BLC1079
module tb_half_adder;

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

    initial begin
                a=0;b=0;
        #10 a = 0; b = 1;
        #10 a = 1; b = 0;
        #10 a = 1; b = 1;
        #10 ;
        // End simulation
        $stop;
    end
endmodule
```

1-Bit Full Adder and its Test Bench:



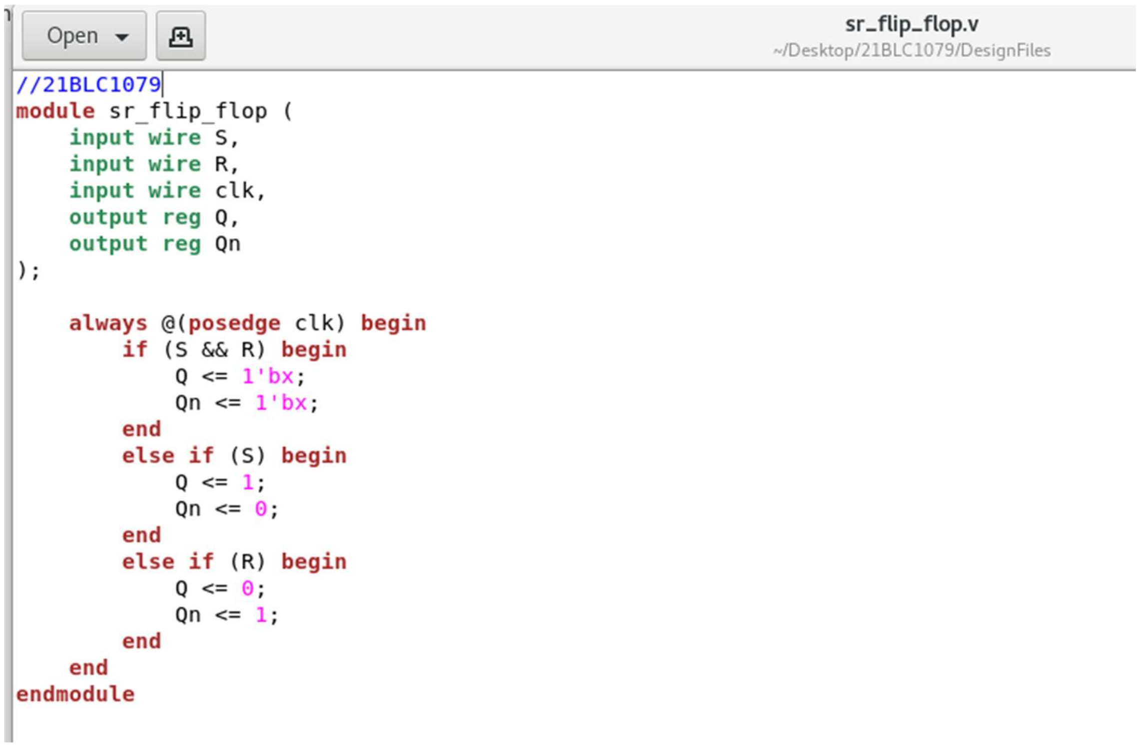
Open | *full_adder.v ~/Desktop/21BLC1079/DesignFiles

```
//21BLC1079
module full_adder (
    input wire a,
    input wire b,
    input wire cin,
    output wire sum,
    output wire carry_out
);

    wire sum1, carry1, carry2;

    // Instantiate two half adders
    half_adder ha1 (a,b,sum1,carry1);
    half_adder ha2 (sum1,cin,sum,carry2);
    assign carry_out = carry1 | carry2;

endmodule
```

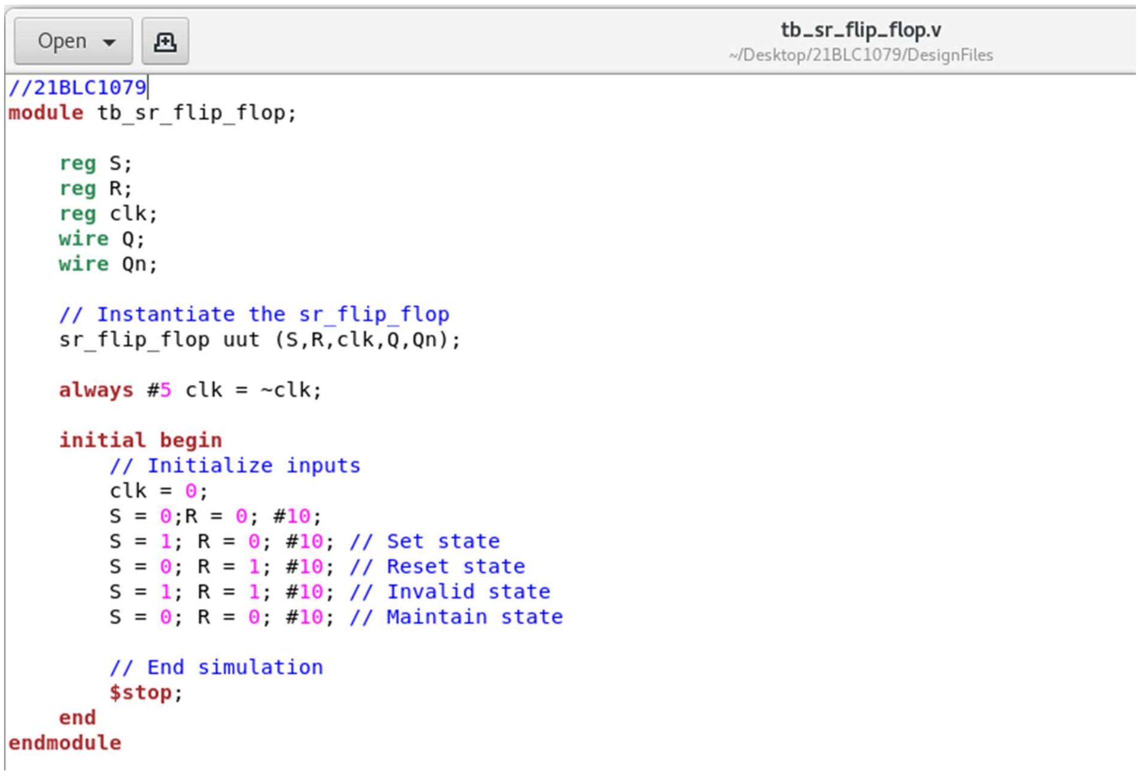


```verilog
//21BLC1079
module tb_full_adder;

    reg a;
    reg b;
    reg cin;
    wire sum;
    wire carry_out;

    full_adder uut (a,b,cin,sum,carry_out);

    initial begin
        // Initialize inputs
        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;

        // End simulation
        $stop;
    end
endmodule
```

S-R Flipflop and its test bench:



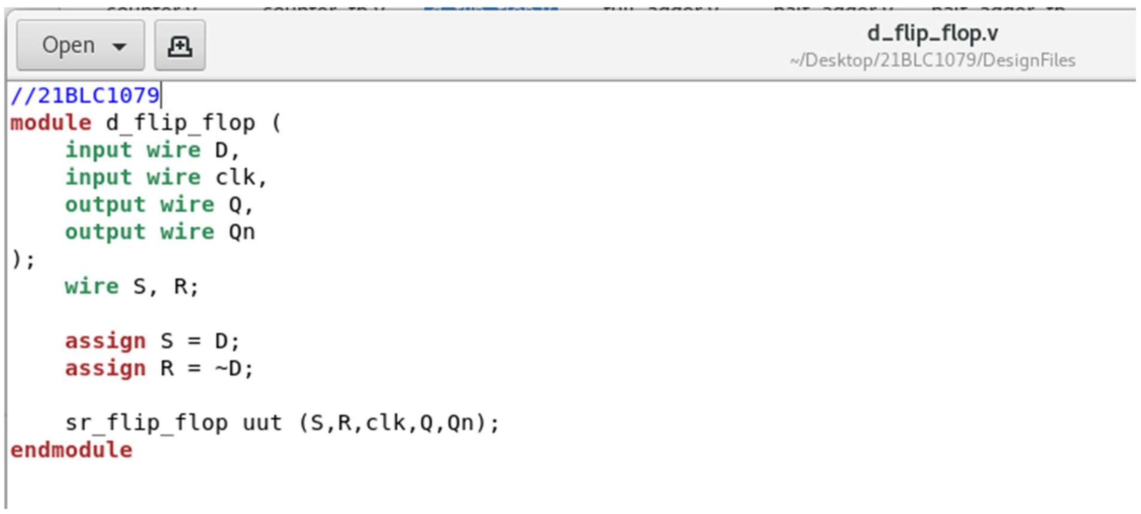
```verilog
//21BLC1079
module sr_flip_flop (
    input wire S,
    input wire R,
    input wire clk,
    output reg Q,
    output reg Qn
);

    always @(posedge clk) begin
        if (S && R) begin
            Q <= 1'bx;
            Qn <= 1'bx;
        end
        else if (S) begin
            Q <= 1;
            Qn <= 0;
        end
        else if (R) begin
            Q <= 0;
            Qn <= 1;
        end
    end
endmodule
```


Open

tb_sr_flip_flop.v
~/Desktop/21BLC1079/DesignFiles

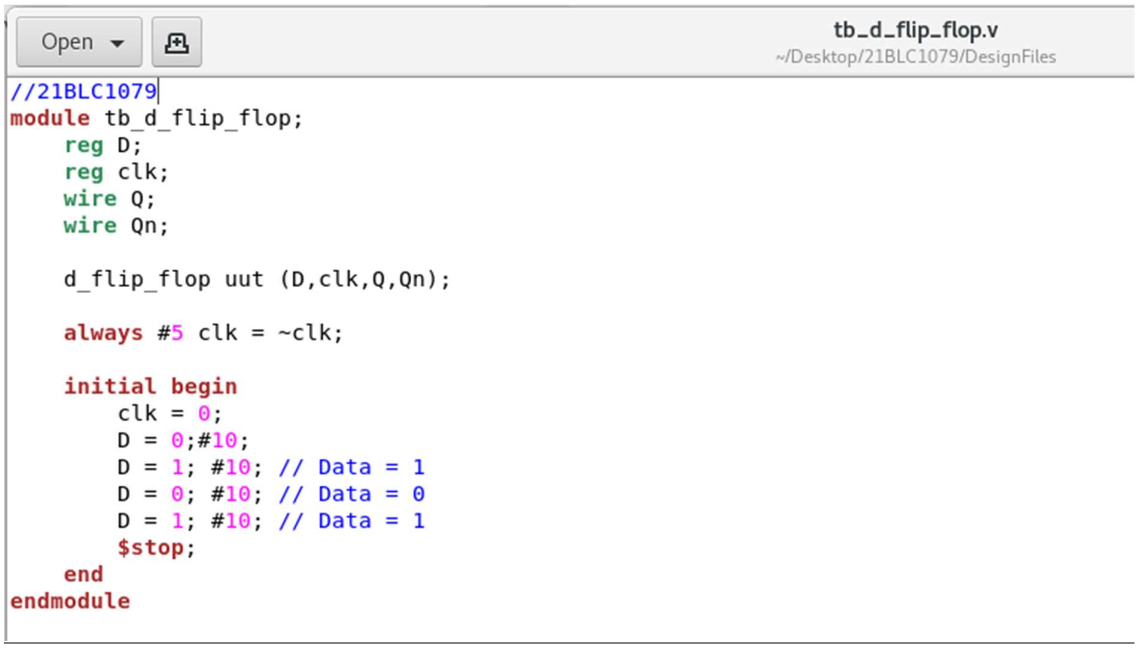
```
//21BLC1079
module tb_sr_flip_flop;

    reg S;
    reg R;
    reg clk;
    wire Q;
    wire Qn;

    // Instantiate the sr_flip_flop
    sr_flip_flop uut (S,R,clk,Q,Qn);

    always #5 clk = ~clk;

    initial begin
        // Initialize inputs
        clk = 0;
        S = 0;R = 0; #10;
        S = 1; R = 0; #10; // Set state
        S = 0; R = 1; #10; // Reset state
        S = 1; R = 1; #10; // Invalid state
        S = 0; R = 0; #10; // Maintain state

        // End simulation
        $stop;
    end
endmodule
```

D-flip flop and its test bench:


Open

d_flip_flop.v
~/Desktop/21BLC1079/DesignFiles

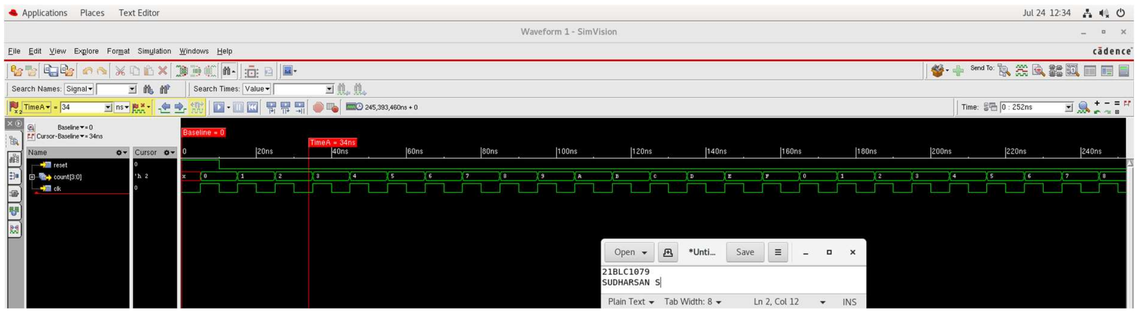
```
//21BLC1079
module d_flip_flop (
    input wire D,
    input wire clk,
    output wire Q,
    output wire Qn
);
    wire S, R;

    assign S = D;
    assign R = ~D;

    sr_flip_flop uut (S,R,clk,Q,Qn);
endmodule
```



```
//21BLC1079
module tb_d_flip_flop;
    reg D;
    reg clk;
    wire Q;
    wire Qn;

    d_flip_flop uut (D,clk,Q,Qn);

    always #5 clk = ~clk;

    initial begin
        clk = 0;
        D = 0;#10;
        D = 1; #10; // Data = 1
        D = 0; #10; // Data = 0
        D = 1; #10; // Data = 1
        $stop;
    end
endmodule
```

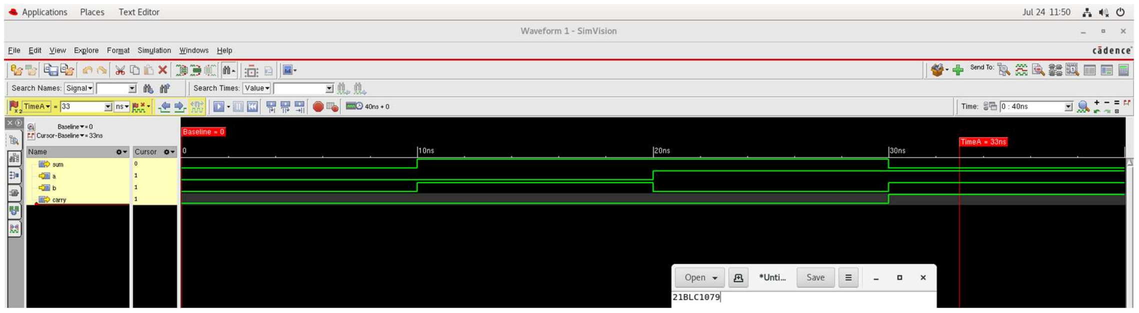
OBSERVATIONS:

4-bit up counter



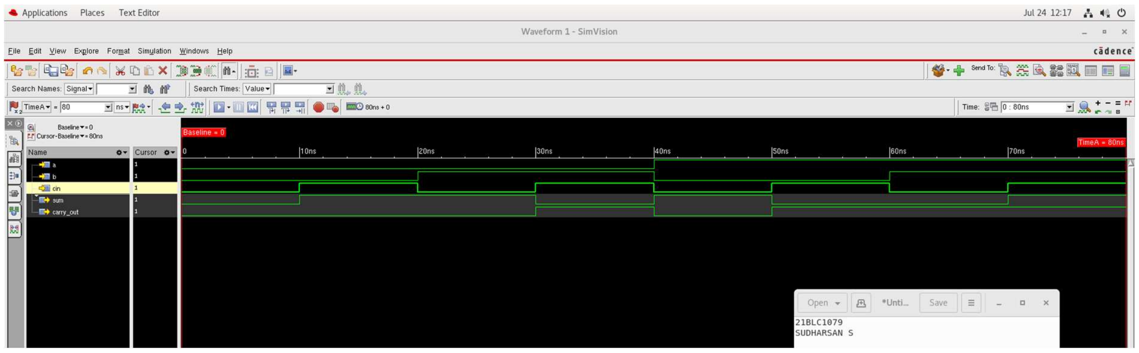
```
21BLC1079
SUDHARSAN S
```

1-bit Half adder



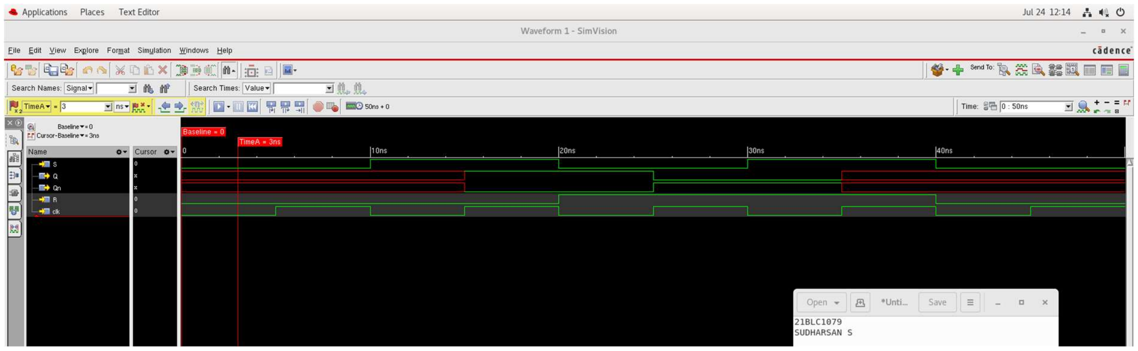
```
21BLC1079
```

1-bit Full adder using half adder



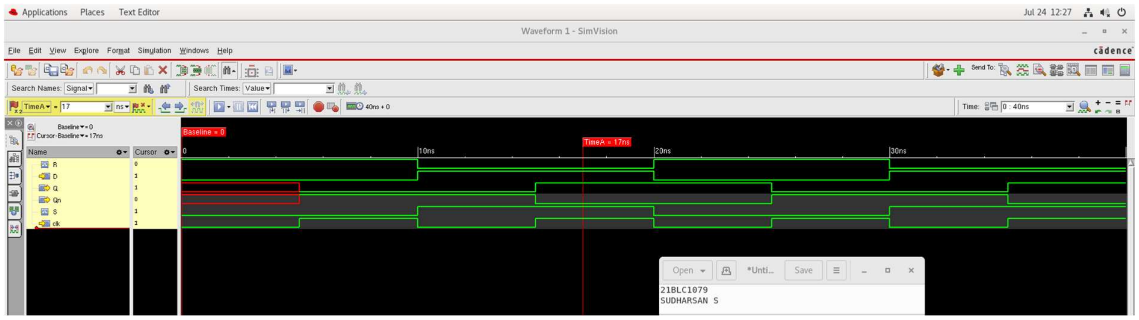
```
21BLC1079
SUDHARSAN S
```

S-R Flipflop



```
21BLC1079
SUDHARSAN S
```

D-Flipflop using S-R Flipflop



```
21BLC1079
SUDHARSAN S
```

<u>INFERENCE:</u>

This experiment provided valuable insights into the functioning of basic sequential and combinational circuits. It involved writing Verilog HDL code for these circuits and using Cadence NC launch along with related EDA tools. Through this process, we gained practical experience and were able to verify the circuits' functionality by examining the resulting waveforms.