Lab 7:Introduction to Sequential Logic

ECEN 248 - 505

TA: Younggyun Cho

Date: October 20, 2020

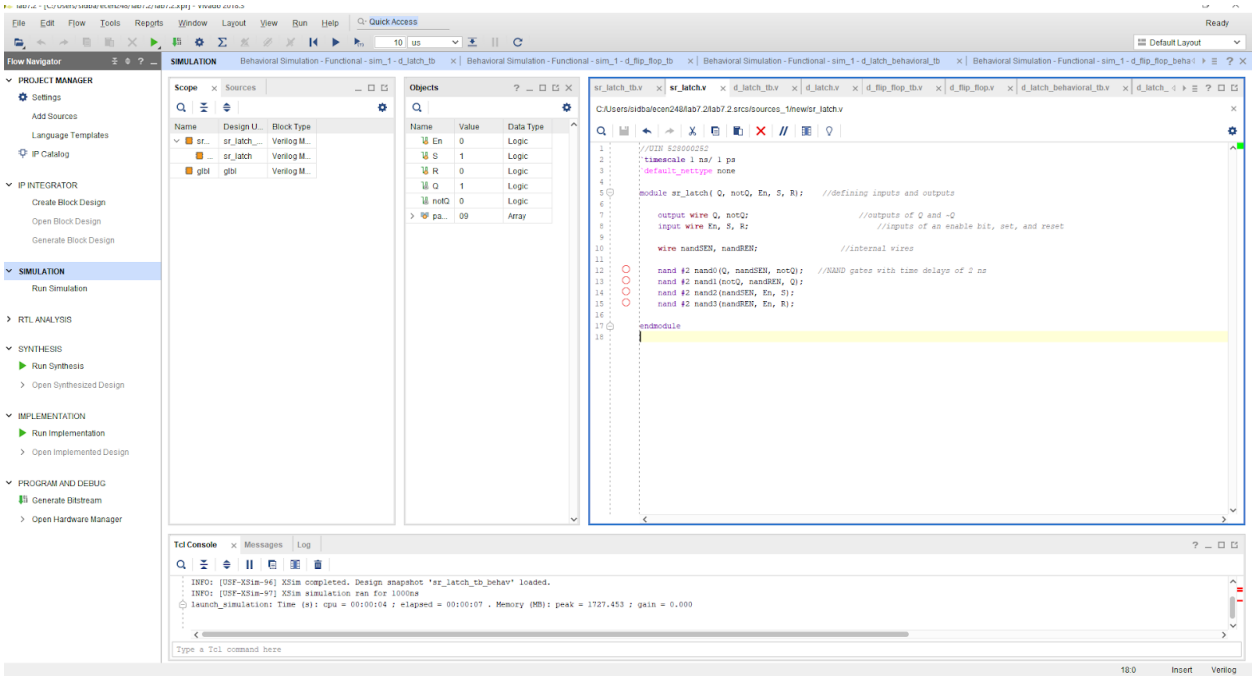
Objectives

The goal of this lab was to introduce the sequential logic circuit. This covers the storage elements like latches and flip flops. I utilized Verilog to describe then at gate level and to simulate it. This lab also goes over synchronous sequential logic circuits. Combining flip-flops with combinational logic helped to simulate the operation of synchronous logic. This lab also introduces the student to simulation delays into combinational logic which has effects it has on clock timing.

Design

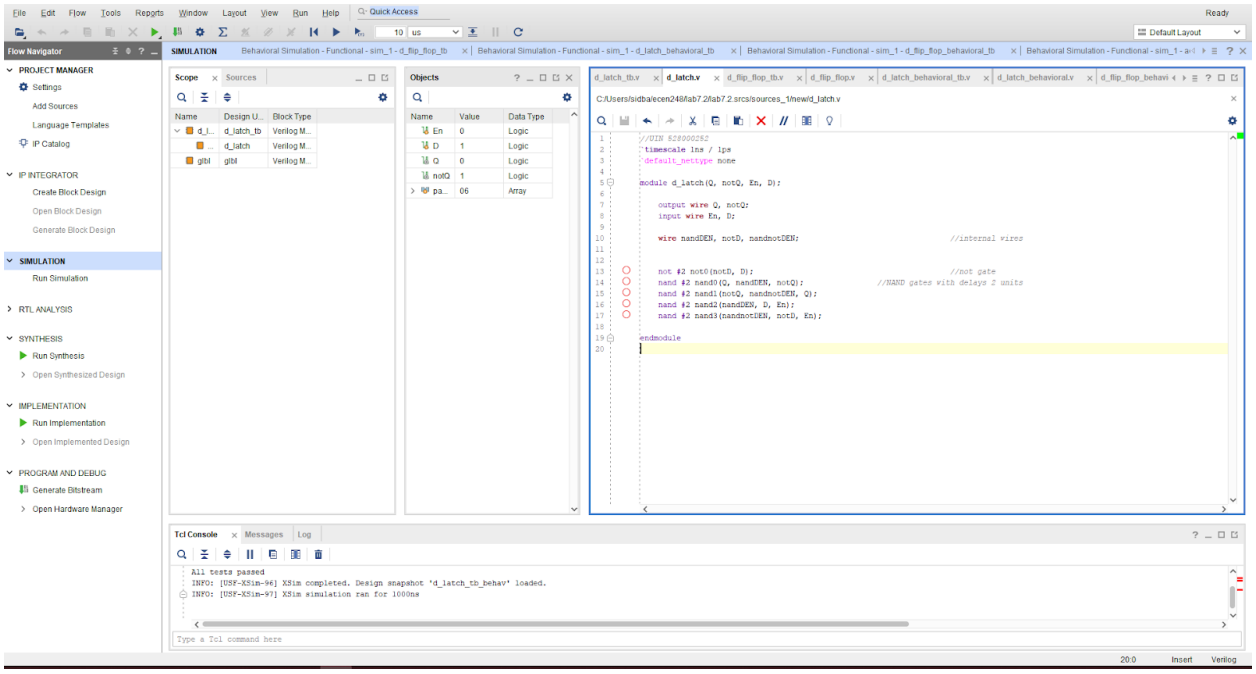
sr latch

```
//UIN 528000252
`timescale 1 ns/ 1 ps
`default_nettype none

module sr_latch( Q, notQ, En, S, R);      //defining inputs and outputs

    output wire Q, notQ;                  //outputs of Q and ~Q
    input wire En, S, R;                  //inputs of an enable bit, set, and reset

    wire nandSEN, nandREN;                //internal wires

    nand #2 nand0(Q, nandSEN, notQ);   //NAND gates with time delays of 2 ns
    nand #2 nand1(notQ, nandREN, Q);
    nand #2 nand2(nandSEN, En, S);
    nand #2 nand3(nandREN, En, R);
```



D_latch

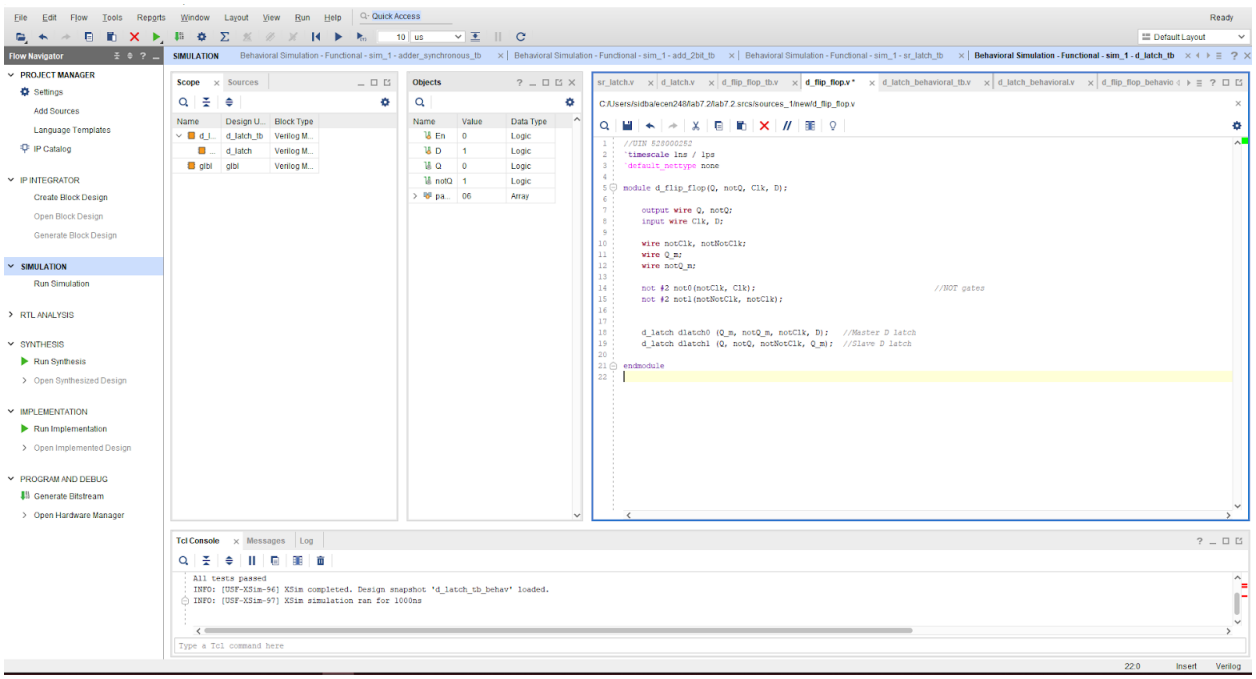
```
//UIN 528000252
`timescale 1ns / 1ps
`default_nettype none

module d_latch(Q, notQ, En, D);

        output wire Q, notQ;
        input wire En, D;

        wire nandDEN, notD, nandnotDEN;                 //internal wires


        not #2 not0(notD, D);                           //not gate
        nand #2 nand0(Q, nandDEN, notQ);                //NAND gates with delays 2 units
        nand #2 nand1(notQ, nandnotDEN, Q);
        nand #2 nand2(nandDEN, D, En);
        nand #2 nand3(nandnotDEN, notD, En);

endmodule
```



D_flip_flop

```
//UIN 528000252
`timescale 1ns / 1ps
`default_nettype none

module d_flip_flop(Q, notQ, Clk, D);
```

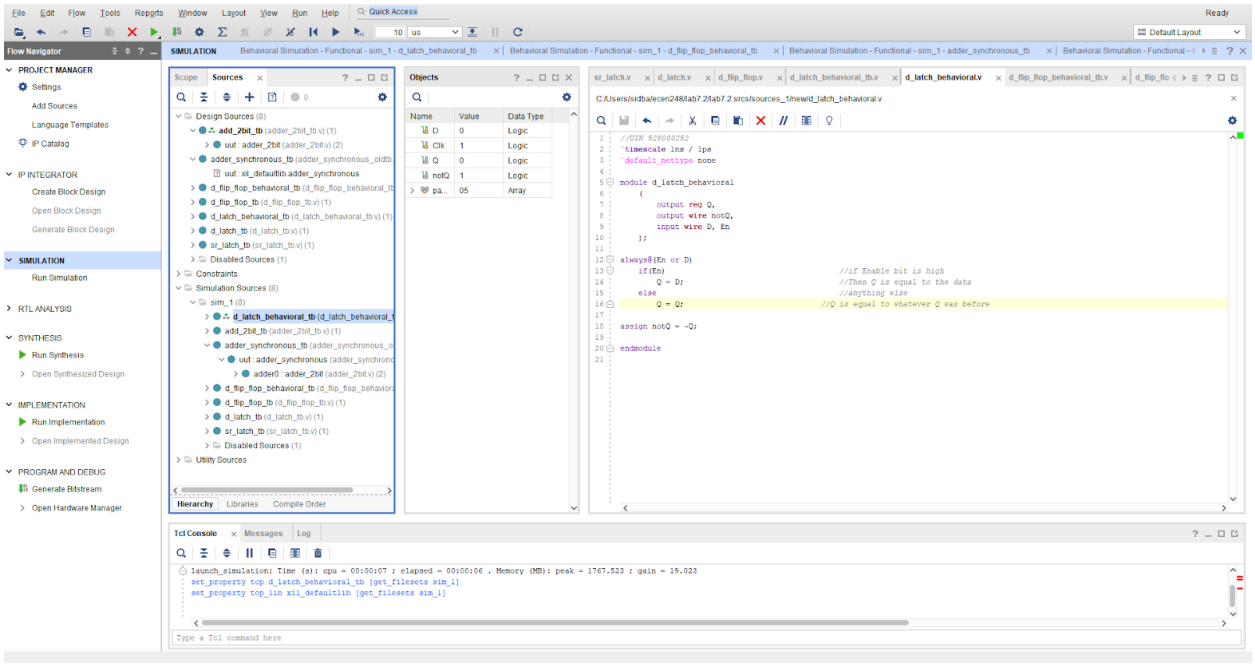
```
        output wire Q, notQ;
        input wire Clk, D;

        wire notClk, notNotClk;
        wire Q_m;
        wire notQ_m;

        not #2 not0(notClk, Clk);                       //NOT gates
        not #2 not1(notNotClk, notClk);


        d_latch dlatch0 (Q_m, notQ_m, notClk, D);   //Master D latch
        d_latch dlatch1 (Q, notQ, notNotClk, Q_m);  //Slave D latch

endmodule
```



D_latch_behavioral

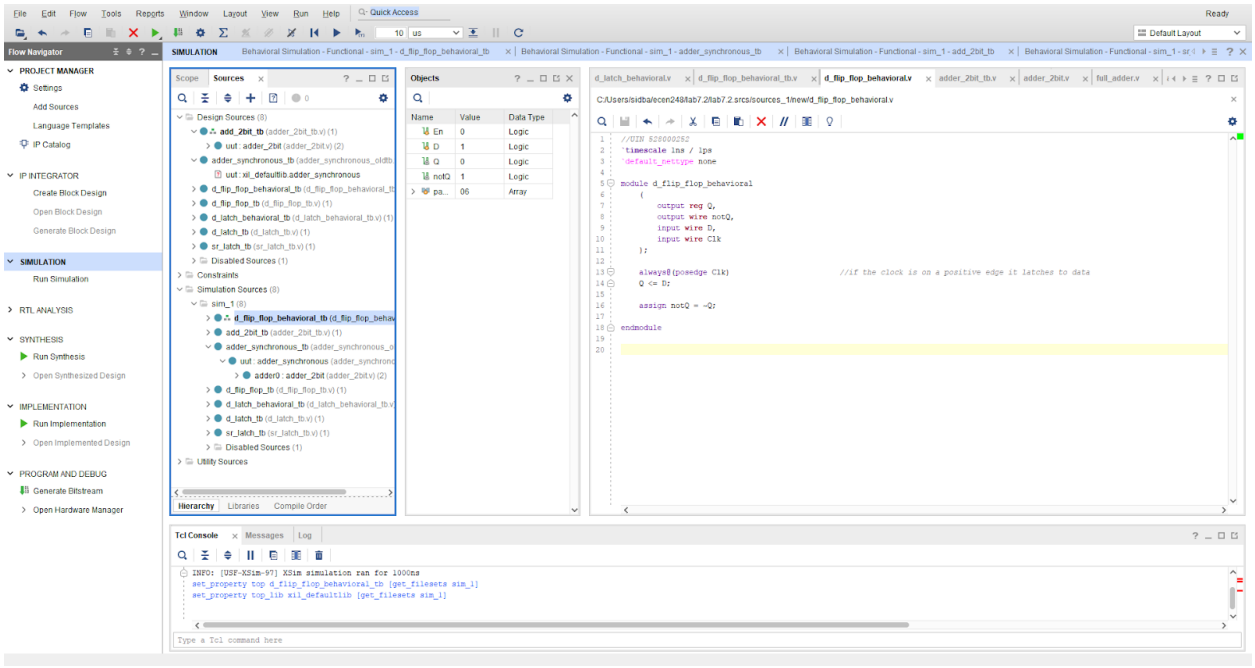
```
//UIN 528000252
`timescale 1ns / 1ps
`default_nettype none

module d_latch_behavioral
        (
                output reg Q,
                output wire notQ,
                input wire D, En
        );
```

```
always@(En or D)
        if(En)                          //if Enable bit is high
                Q = D;                          //Then Q is equal to the data
        else                            //anything else
                Q = Q;                          //Q is equal to whatever Q was before

assign notQ = ~Q;
endmodule
```



D_flip_flop_behavioral

```
//UIN 528000252
`timescale 1ns / 1ps
`default_nettype none

module d_flip_flop_behavioral
        (
                output reg Q,
                output wire notQ,
                input wire D,
                input wire Clk
        );

        always@(posedge Clk)            //if the clock is on a positive edge it latches to data
        Q <= D;

        assign notQ = ~Q;

endmodule
```



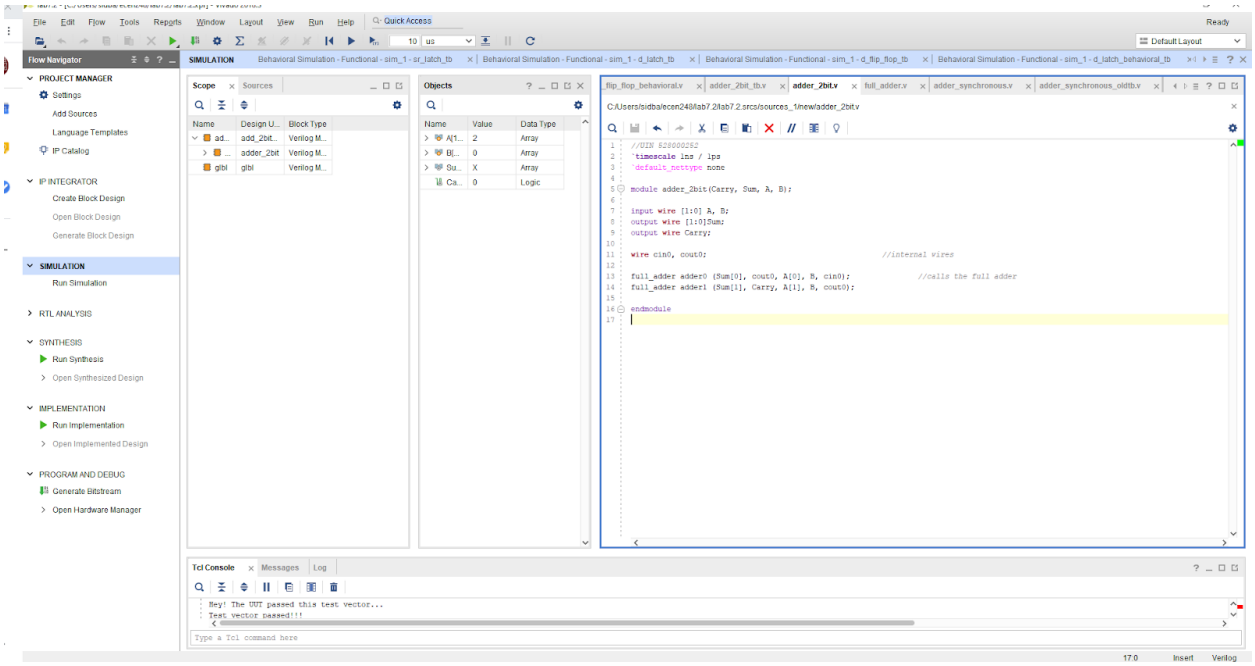
Full adder

```
`timescale 1ns / 1ps
`default_nettype none
module full_adder(S, Cout, A, B, Cin);
// 1 bit wire
    output wire S, Cout;
    input wire A, B, Cin;

    //declare internal nets
    wire andBCin, andACin, andAB;

    //use date flow to describe gatelevel activity
    assign S = A ^ B ^Cin;
    assign andAB= A & B;
    assign andBCin = B & Cin;
    assign andACin = A & Cin;
    assign Cout = andAB | andBCin | andACin;

endmodule
```

adder_2bit

```
//UIN 528000252
`timescale 1ns / 1ps
`default_nettype none

module adder_2bit(Carry, Sum, A, B);
```

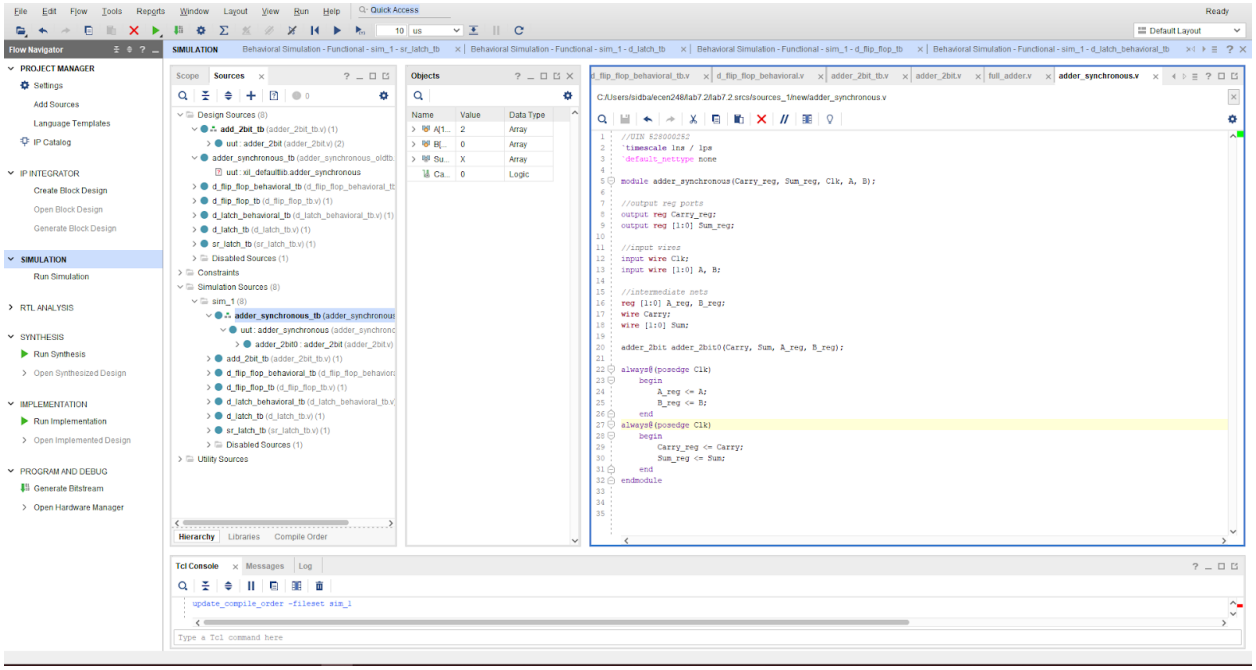
```
input wire [1:0] A, B;
output wire [1:0]Sum;
output wire Carry;

wire cin0, cout0;                          //internal wires

full_adder adder0 (Sum[0], cout0, A[0], B, cin0);          //calls the full adder
full_adder adder1 (Sum[1], Carry, A[1], B, cout0);

endmodule
```



adder_synchronous

```
//UIN 528000252
`timescale 1ns / 1ps
`default_nettype none

module adder_synchronous(Carry_reg, Sum_reg, Clk, A, B);

//output reg ports
output reg Carry_reg;
output reg [1:0] Sum_reg;

//input wires
input wire Clk;
input wire [1:0] A, B;

//intermediate nets
reg [1:0] A_reg, B_reg;
```

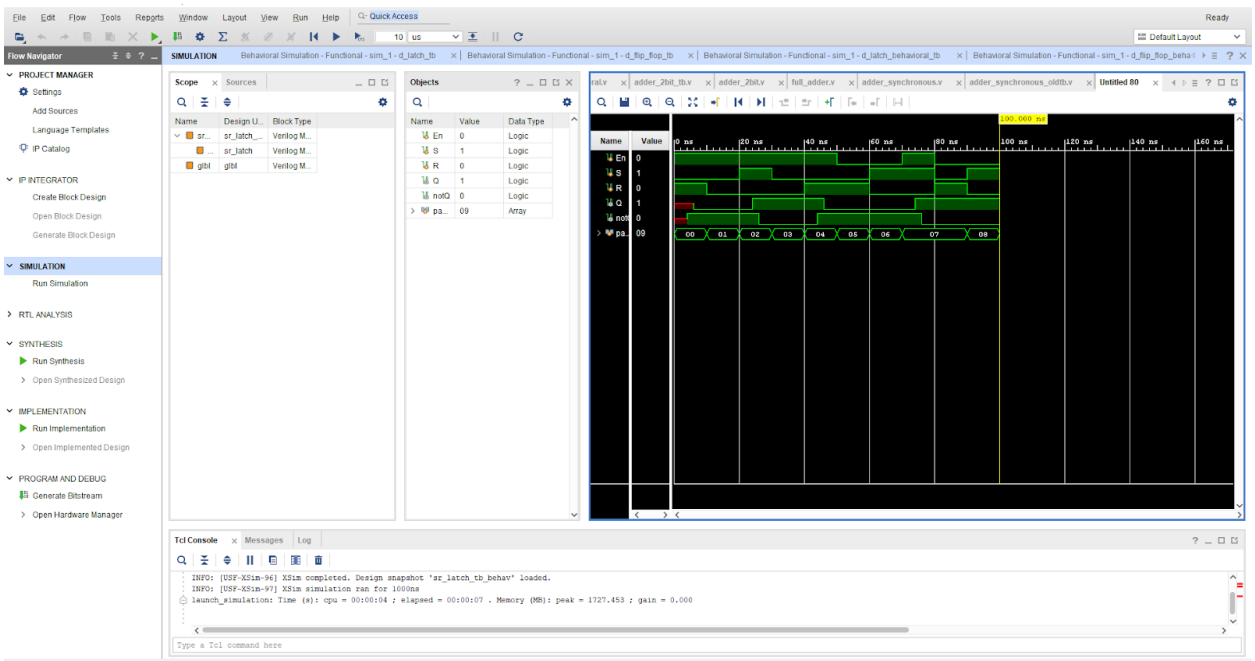
```
wire Carry;
wire [1:0] Sum;

adder_2bit adder_2bit0(Carry, Sum, A_reg, B_reg);

always@(posedge Clk)
        begin
                A_reg <= A;
                B_reg <= B;
        end
always@(posedge Clk)
        begin
                Carry_reg <= Carry;
                Sum_reg <= Sum;
        end
endmodule
```



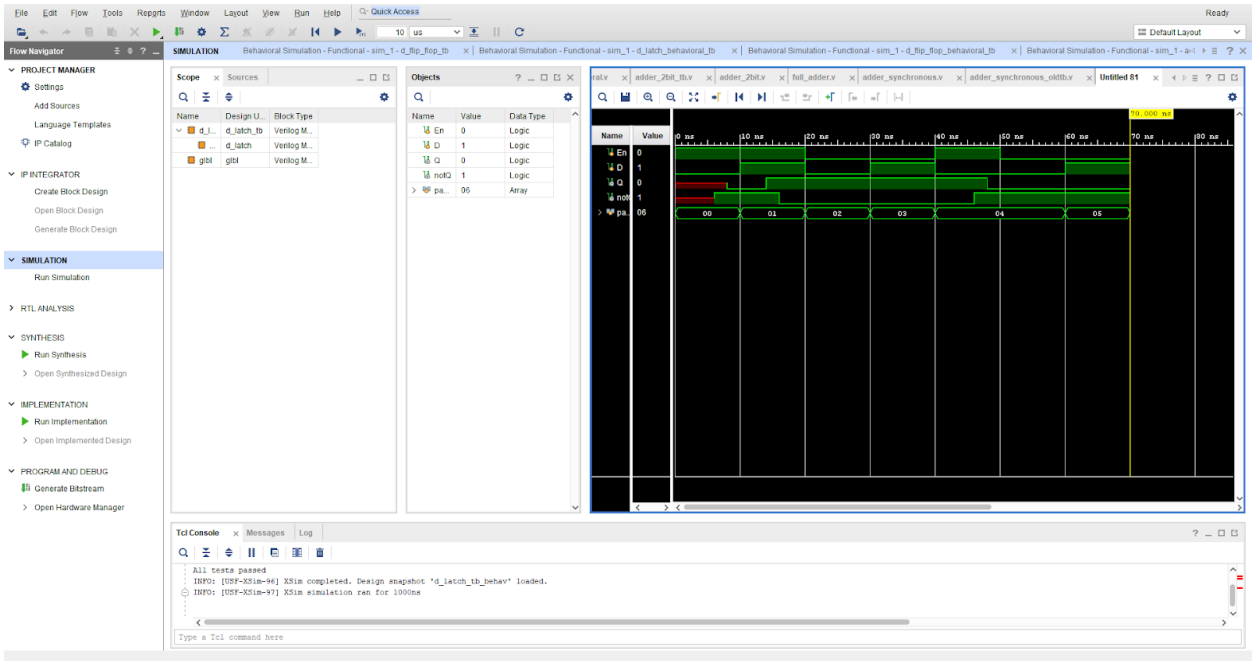
Result

_____Running each module successfully and creating the waveforms is what's counted for results. Below are the waveforms for all the modules.

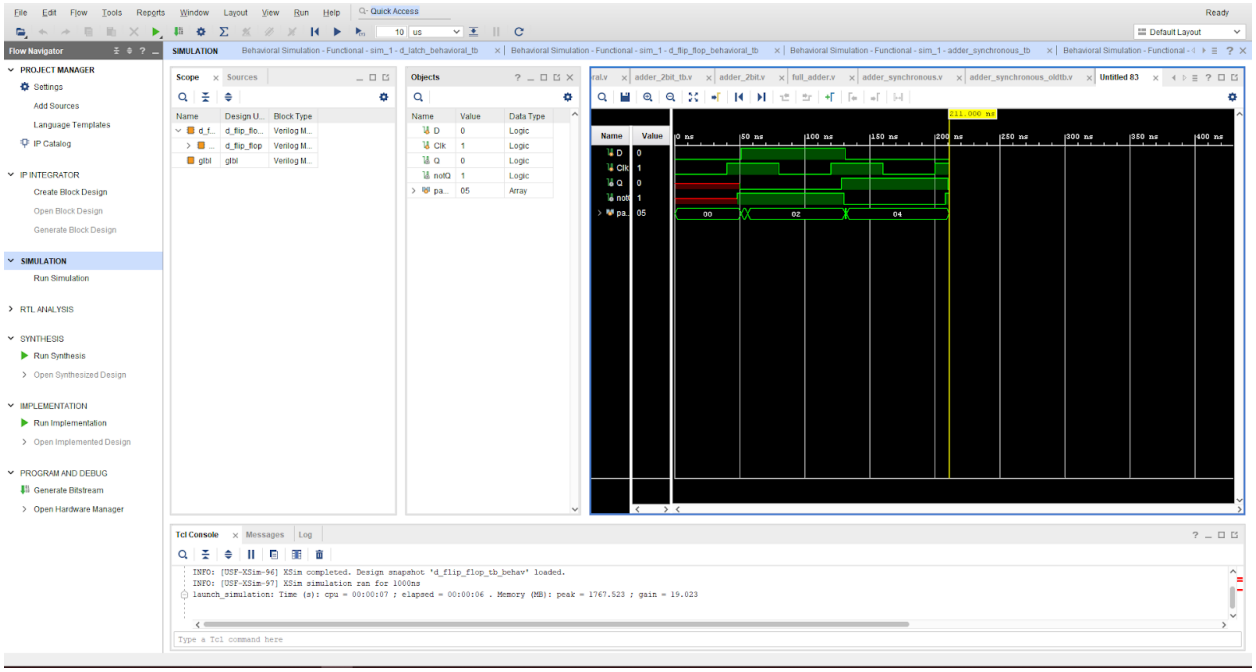
sr latch



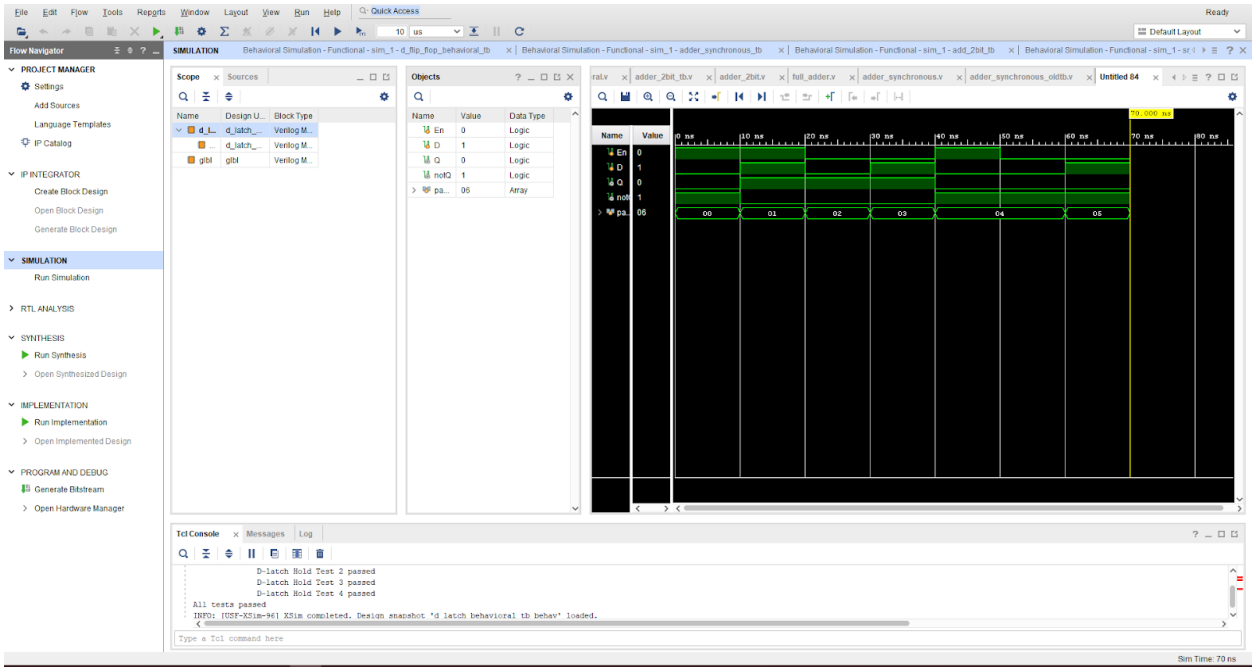
D_latch



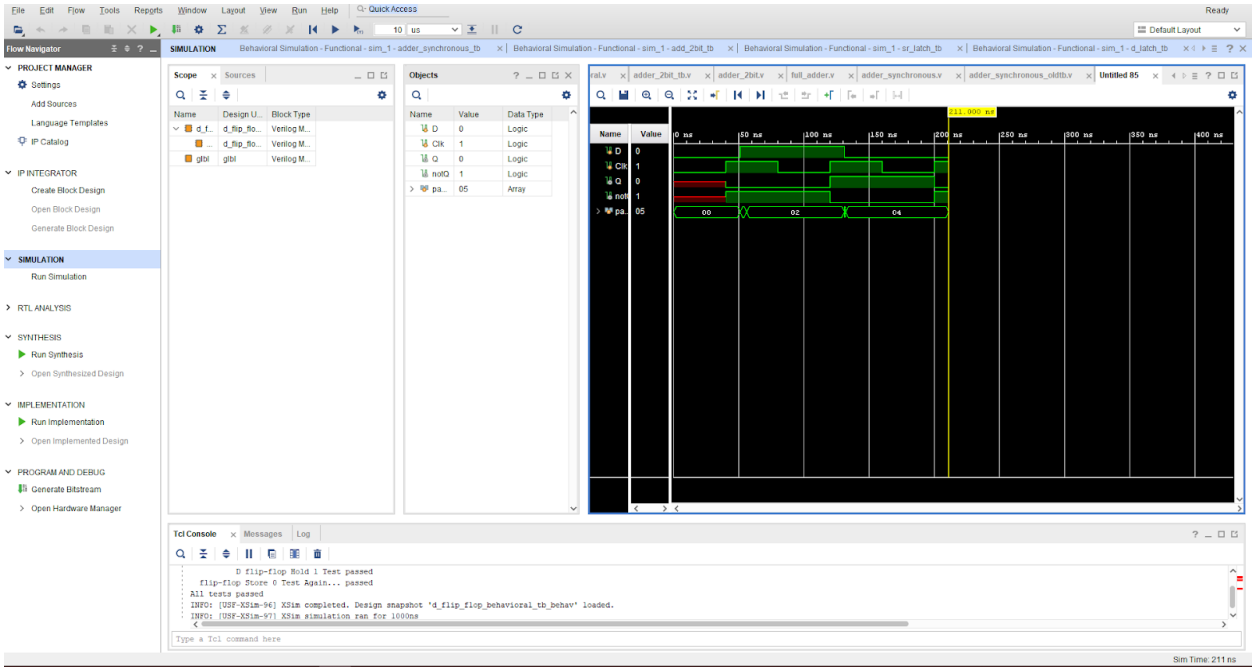
D_flip_flop



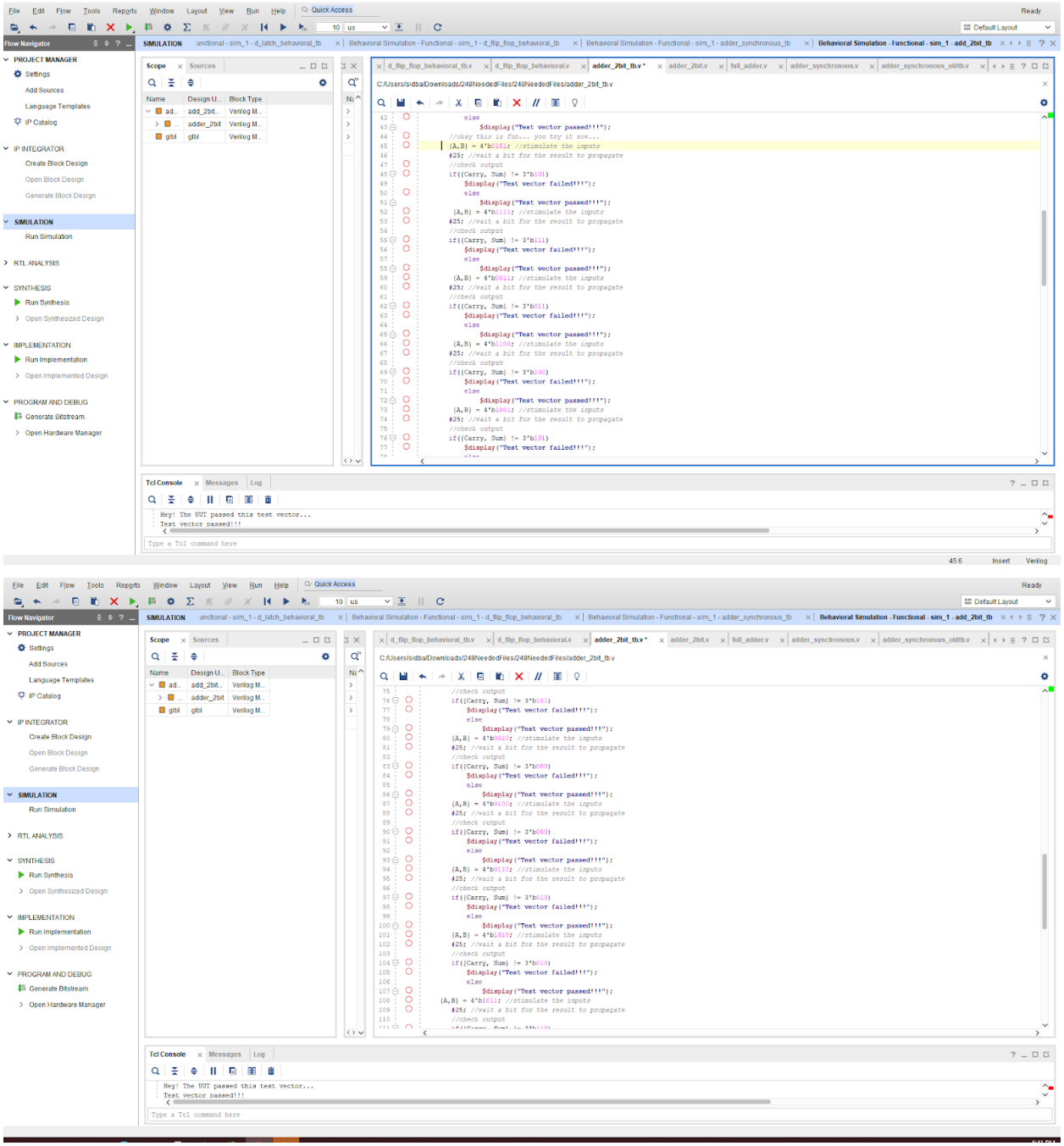
D_latch_behavioral

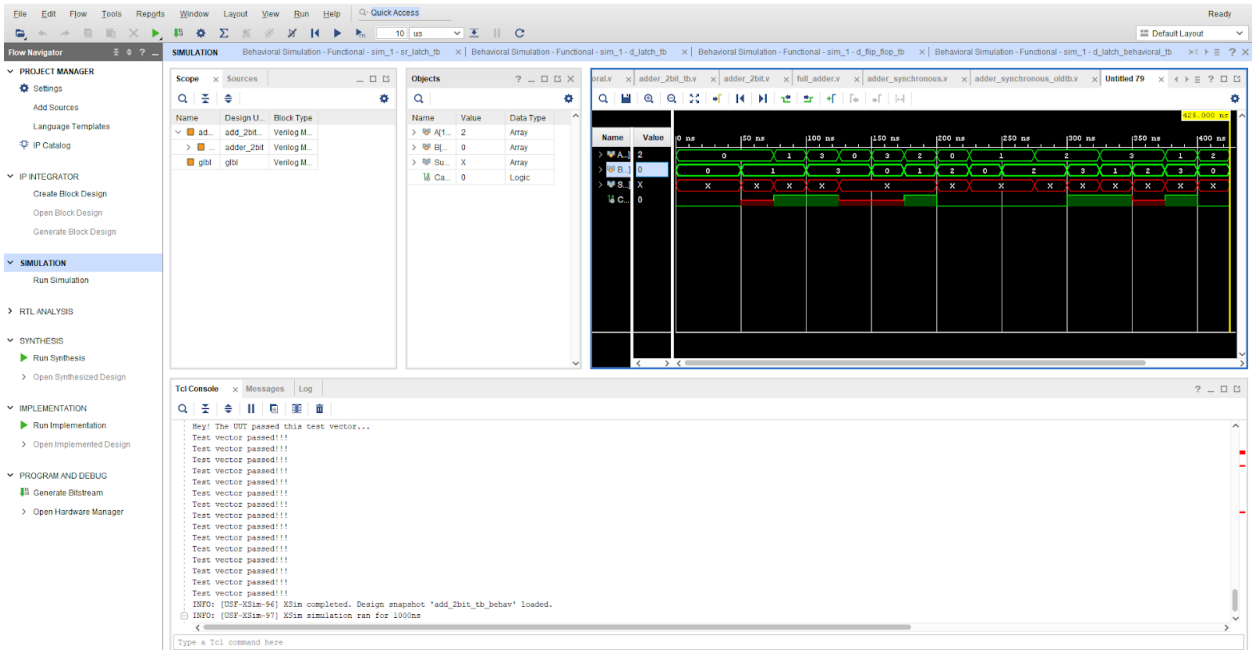


D_flip_flop_behavioral

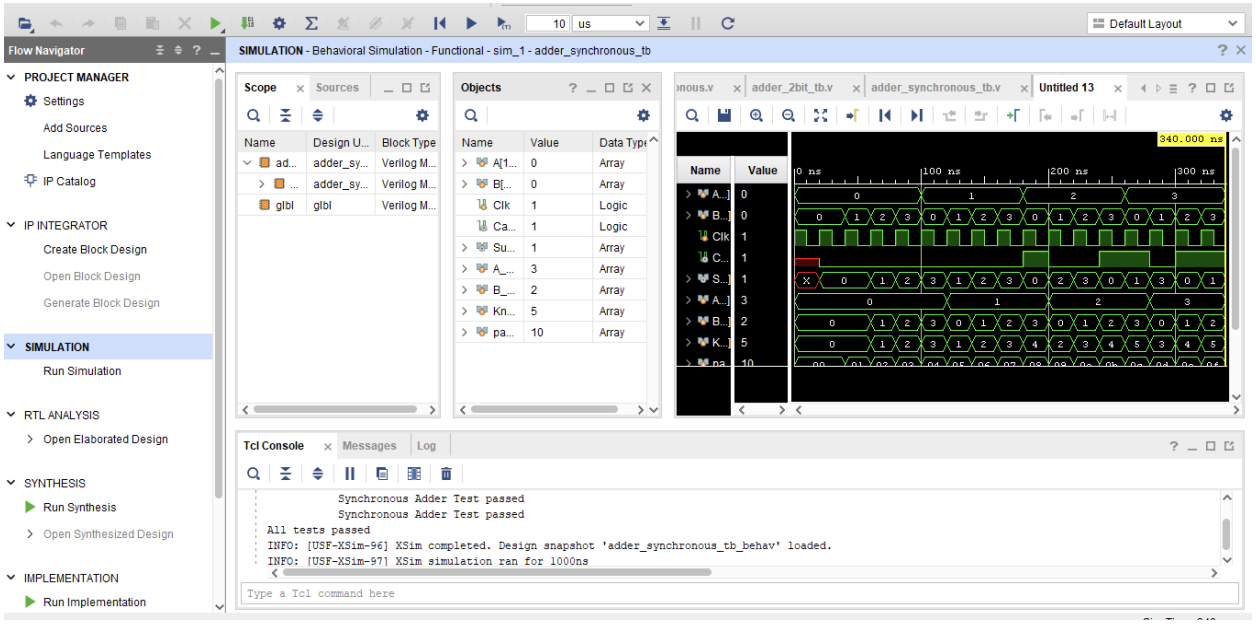


Adder_2bit

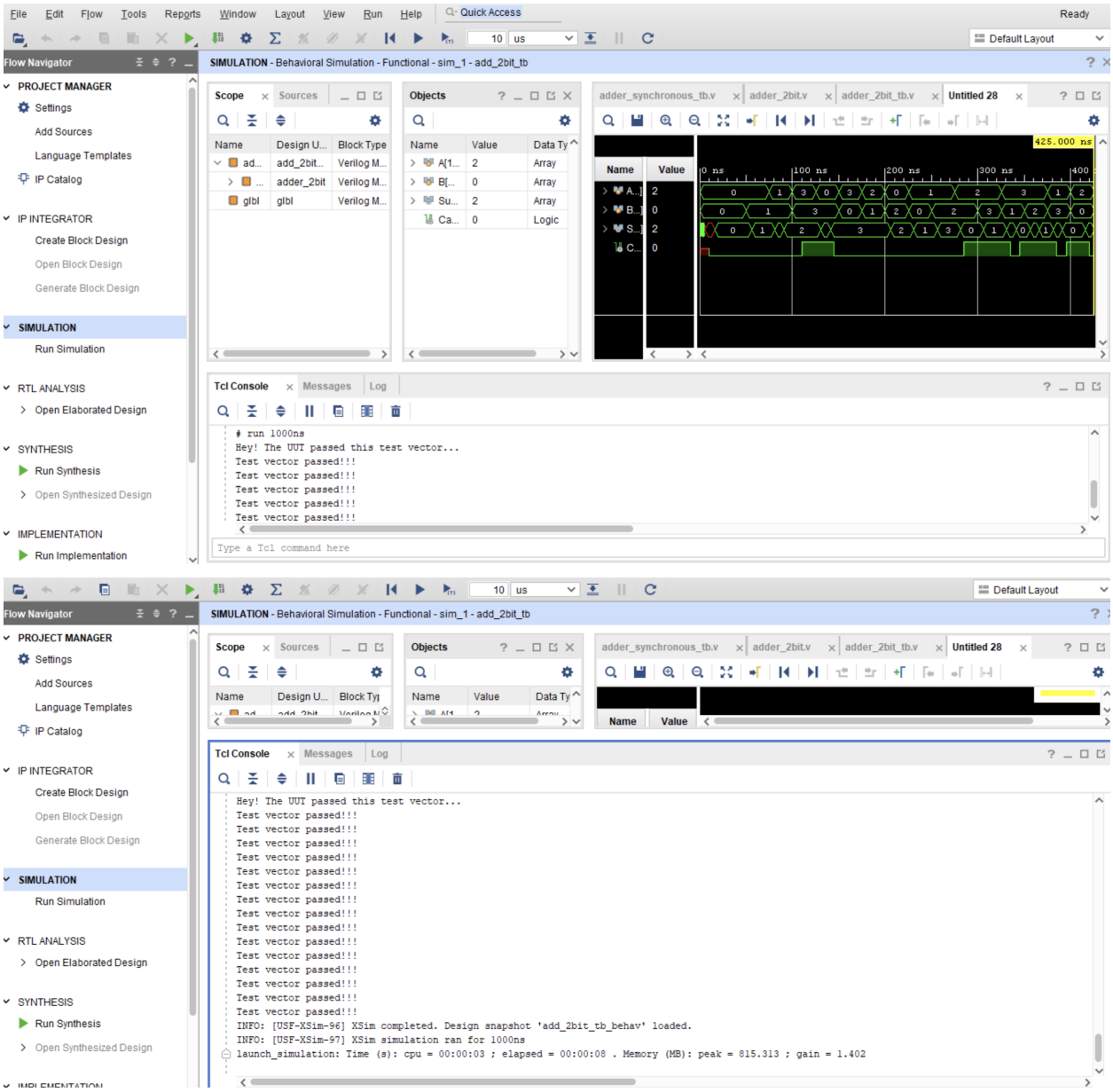


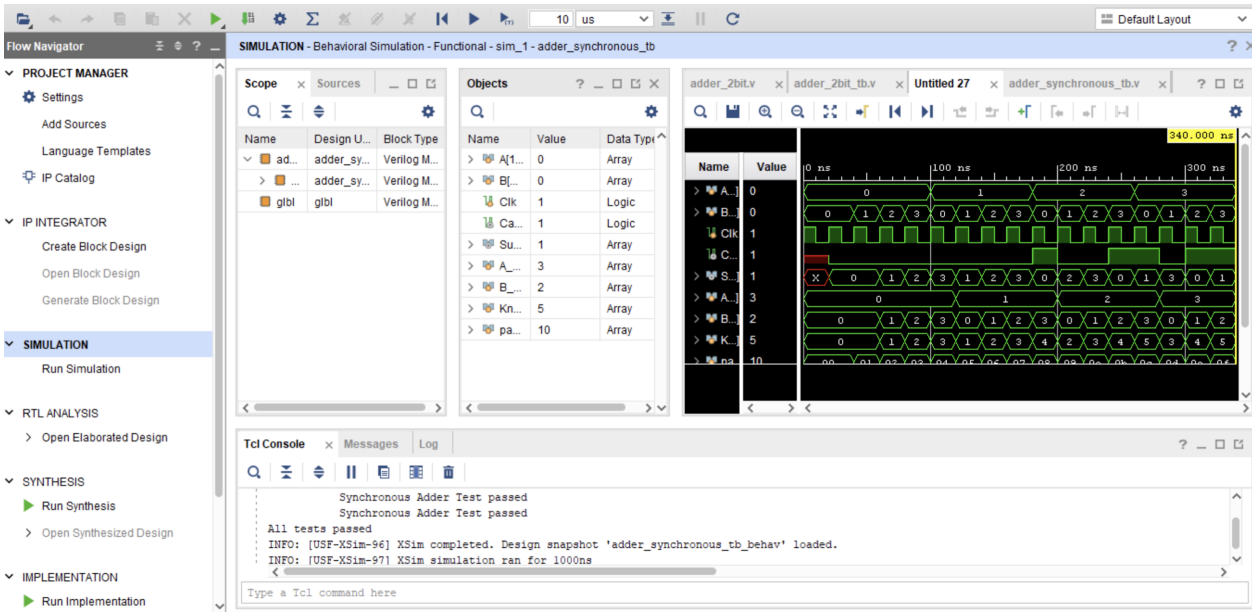


adder_synchronous



//part 3


File Edit Flow Tools Reports Window Layout View Run Help
Quick Access
Ready
Default Layout
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
Open Elaborated Design
SYNTHESIS
Run Synthesis
Open Synthesized Design
IMPLEMENTATION
Run Implementation
SIMULATION - Behavioral Simulation - Functional - sim_1 - add_2bit_tb
Scope
Sources
Objects
adder_synchronous_tb.v
adder_2bit.v
adder_2bit_tb.v
Untitled 28
425.000 ns
Tcl Console
Messages
Log
run 1000ns
Hey! The UUT passed this test vector...
Test vector passed!!!
Type a Tcl command here
INFO: [USF-XSim-96] XSim completed. Design snapshot 'add_2bit_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:03 ; elapsed = 00:00:08 . Memory (MB): peak = 815.313 ; gain = 1.402



Conclusion

In lab 7 I was introduced to the concept of sequential logic circuit. This lab contained modules with delays like SR-latch, a D-latch, a D flip-flop, a behavioral D-latch, a behavioral D flip-flop, and a 2-bit synchronous adder. We were also required to manipulate the test bench code to test all inputs. We also had to provide screenshots of our tests, as shown above, and manipulate the modules with different clock rates to see its behavior. This was performed on Verilog through the gate level, and combining flip-flops with combinational logic to simulate synchronous logic.

Post - Lab Questions

1. For experiment 1(e) in part 1, explain the results of the simulation.

For this part of the lab I changed the 2 unit delays to 4 unit delays. Initially some tests failed during the transition of the inputs. This happens because modifying the clock rate to a slower rate, the output changes at a slower pace leading the circuit to skip some of the inputs. However, going through multiple inputs, the sr_latch gaved many outputs and all test cases passed.

Table 1: SR-latch Function Table

En	S	R	Q	$\overline{Q}$
0	X	X	hold	
1	0	0	hold	
1	0	1	0	1
1	1	0	1	0
1	1	1	undefined	

2. For experiment 3 in part 1, check the waveform with internal signals. Are the latches behaving as expected? Why or why not?

While restarting the simulation, the latches behaved as they should since the D-latches already contain delays. We can confirm this since all the test cases passed and the waveform is an accurate representation of the code.

3. For experiment 1.4 in part 1, compare the waveforms you captured from the behavioral Verilog to those from the structural Verilog. Are they different? If so, how?

Comparing structural D-latch and the behavioral D-latch the student can say how the behavioral computes the output at a faster rate than structural, the same goes for structural D flip-flop and behavioral D flip-flop. For structural, you are writing code that is synthesizable and so the waveform is more accurate, whereas behavioral has more abstraction and so waveforms are less detailed.

4. For experiment 1(e) in part 2, what is the worst-case propagation delay through the adder?

The final value for the `define CLOCK_PERIOD in comparison to the propagation delay of the ripp;es adder will determine the worst case. The worst case delay is 10 ns, by definginthe CLOCK_PERIOD to 19ns or less, the simulations will fail since the inputs change before the transition event takes place.

5. Based on the clock period you measured for your synchronous adder, what would be the theoretical maximum clock rate? What would be the effect of increasing the width of the adder on the clock rate? How might you improve the clock rate of the design?

The theoretical maximum clock rate is the same at the CLOCK_PERIOD which is defined in the test bench cases.Increasing the width of the adder on the clock rate, slows down the circuit, as it has to go to all the adders. This could be improved by the clock rate of the design by making the clock rate faster for each adder.

Feedback

1. What did you like most about the lab assignment and why? What did you like least about it and why?

I liked adding all the test cases for the adder_2bit because it made me mindful of accurately testing my code. I did not like the complexity of the lab manual.

2. Were there any sections of the lab manual that were unclear? If so, what was unclear? Do you have any suggestions for improving clarity?

The adders part of the lab was a bit unclear, and some more detail would have been more helpful.

3. What suggestions do you have to improve the overall lab assignment?

I don't have any suggestions on changing anything in the lab assignment, other than making the instructions a little more clear.