ECEN248-LabReport

Lab Number: 7

Lab Title: Introduction to Sequential Logic

Student’s Name: Paola Avila

Student’s UIN: 731007033

Date:3/26/2025

Objectives:

This lab aimed to explore the principles and practical uses of sequential logic circuits. The lab started with an in-depth examination of storage elements, focusing on latches and flip-flops, which play a vital role in maintaining state in digital systems. We modeled and simulated these components using Verilog within the Vivado design suite, thereby gaining insights into their behavior and functional characteristics. The lab built upon this foundation by developing synchronous sequential circuits, which focused on integrating flip-flops with combinational logic. By simulating and analyzing synchronous logic systems, we demonstrated how clock signals orchestrate state transitions in digital circuits. In addition, the lab added simulated delays to the combinational logic components. Incorporating these delays allowed us to examine their impact on clock signal timing and synchronization, ultimately deepening our understanding of timing analysis and the practical challenges of real-world digital circuit design.

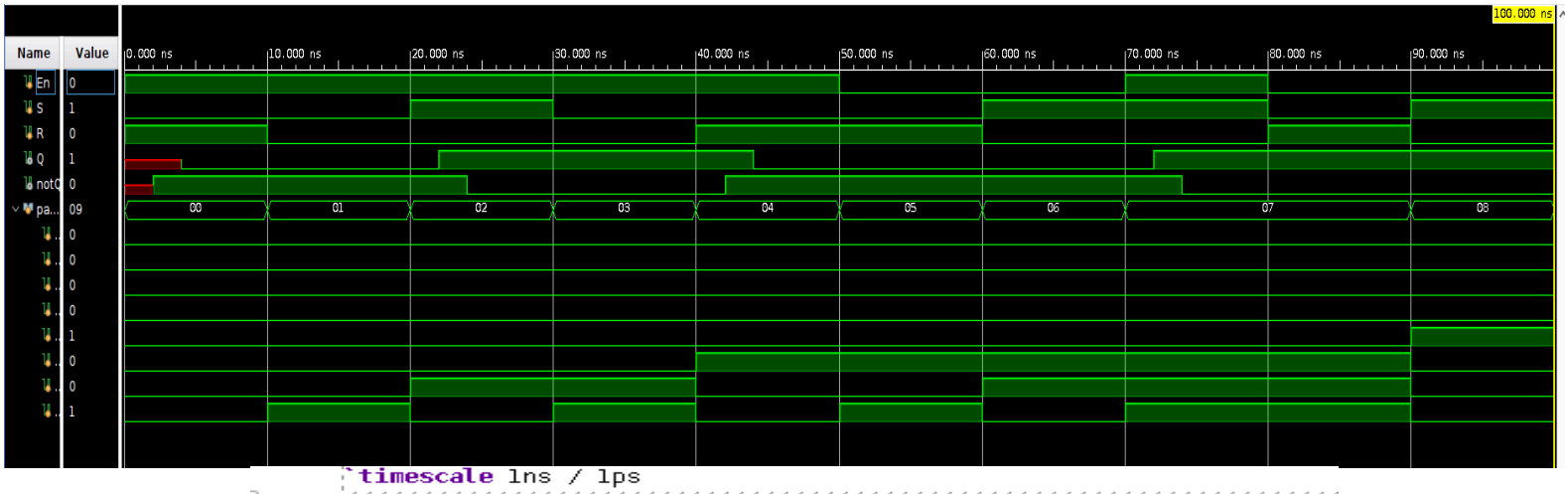
Design :

To begin the experiment, I started by launching Vivado on a Linux platform and creating a new project called ‘lab7.’ Next, I developed a file named sr_latch.v, where I carefully entered the SR latch code and made sure all necessary components were accurately defined..Upon completion, I created a testbench file for the SR latch and then ran the simulation..In the scopes file, I added the nandSEN and nandREN wires to the waveform window to observe their signal behaviors.After capturing a screenshot of the waveform, I adjusted the delay from #2 to #4 and took another screenshot for comparison.. Next, I created a file called d_latch.v, which included 2ns delays in all NAND gates and the inverter, building upon the previous implementation.Following the creation of the testbench file, I ran the simulation and compared the resulting waveform to the specifications listed in Table 2 of the lab manual.. After that, I created a dedicated file for the D flip-flop.I built upon a template, adding the necessary code to guarantee its seamless operation.I created a testbench file to accompany it and incorporated it into the simulation sources.I repeated the simulation process for the SR latch by dragging the relevant wires into the waveform simulation, capturing another screenshot for documentation purposes.I employed behavioral modeling techniques to create a synthesizable Verilog model, describing memory components and generating additional files, including d_latch_behavioral.v and d_flip_flop_behavioral.v, each accompanied by its own testbench file..Following simulation of these files, I captured screenshots of the waveforms and console outputs for documentation.. In the lab’s second part, my focus shifted to designing a 2-bit ripple-carry adder, which I dubbed adder_2bit.v.To facilitate testing, I created a corresponding testbench file.After finalizing the required code, I modified the testbench to incorporate around 16 test cases, setting the maximum value for evaluation at 15 .Having completed the 2-bit ripple-carry adder, I then developed the synchronous adder. Then simulated the synchronous adder and defined the clock period from 40

to 18 and observed the waveform. Also increase the defined Clock period and until the test all the test bench passes.

Results :

Part 1 : (SR _Latch (#2))

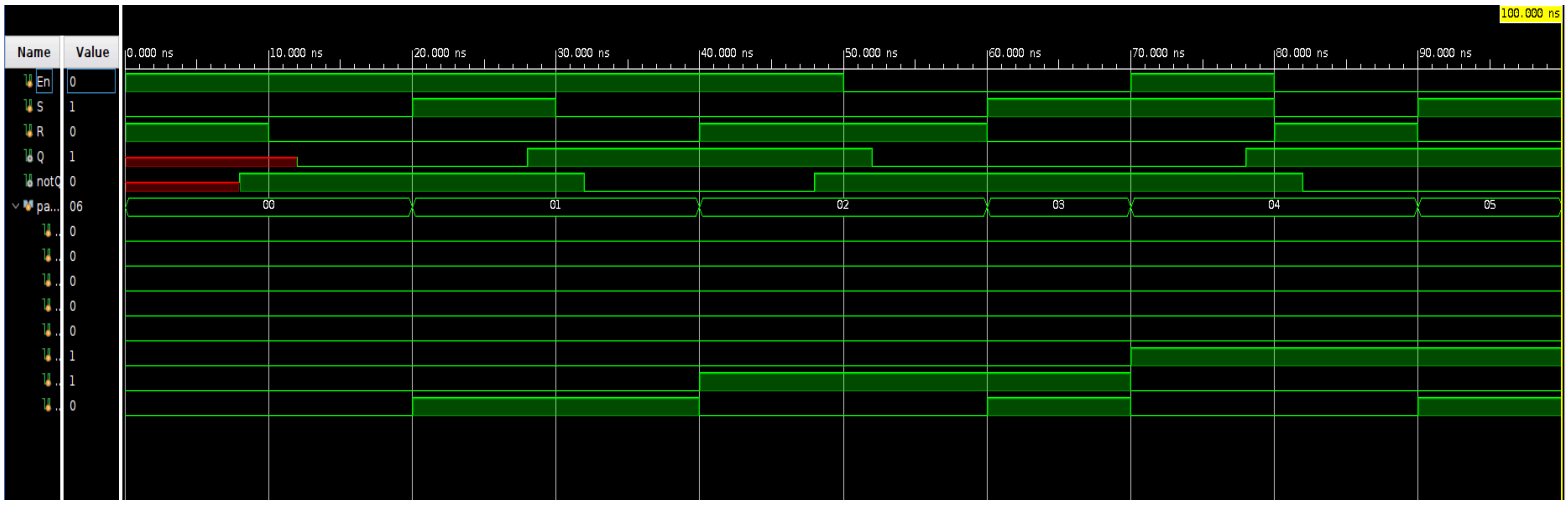



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/21/2025 03:56:37 PM
// Design Name:
// Module Name: sr_latch_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module sr_latch (Q, notQ, En, S, R);
// Port list
output wire Q, notQ; // Outputs of the latch

input wire En, S, R; // Enable, Set, and Reset inputs
// Intermediate nets

wire nandSEN, nandREN; // Intermediate signals from NAND gates
// NAND gates to create the SR latch functionality

 nand #2 nand0(Q, nandSEN, notQ); // 2ns delay
 nand #2 nand1(notQ, nandREN, Q); //also change to #4
    // assign intermediate wires

    assign nandSEN = ~(S & En);
    assign nandREN = ~(R & En);
endmodule
```

```
# ∫
add_wave: Time (s): cpu = 00:00:03 ; elapsed = 0(
# run 1000ns
              SR-latch Reset Test passed
            SR-latch Hold 0 Test passed
               SR-latch Set Test passed
            SR-latch Hold 1 Test passed
        SR-latch Reset from Set Test passed
         SR-latch Enable Hold Test 1 passed
         SR-latch Enable Hold Test 2 passed
         SR-latch Enable Hold Test 3 passed
         SR-latch Enable Hold Test 4 passed
```

Part 1 : (SR _Latch (#4)) :
Change all the delays in your code from“#2” to “#4”. Explain the results of the simulation. Run the test bench again and take a screenshot with the console message

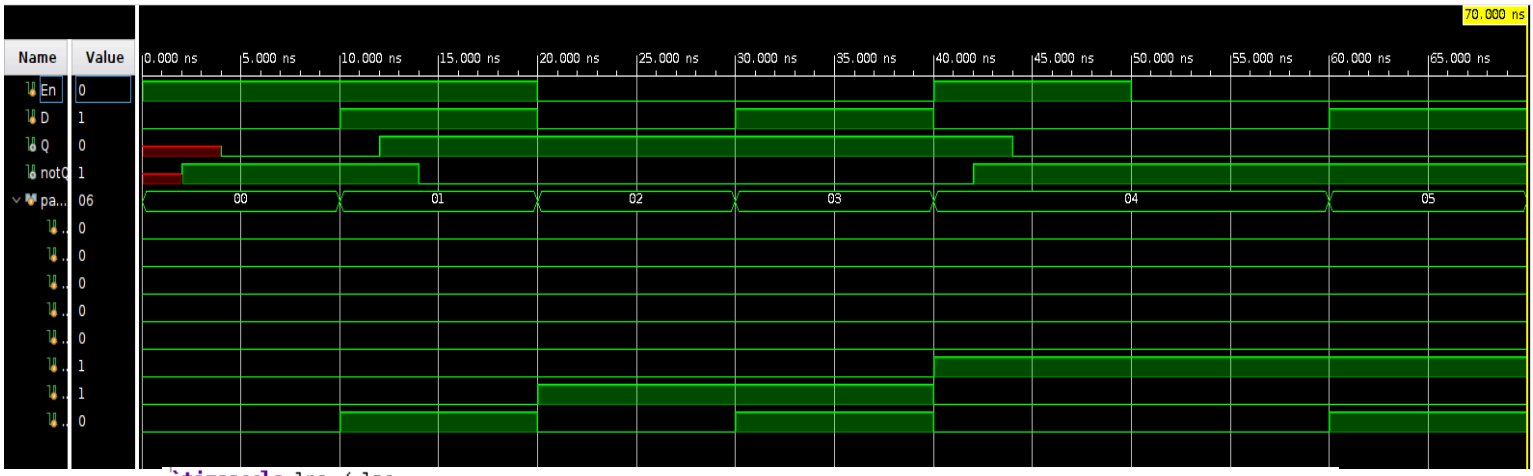
The SRlatch failed the reset and set tests mainly because it mishandled states, causing undefined behavior and output instability when both S and R are high. The propagation delay set at #4 causes input changes to take longer to influence the outputs, thereby disrupting the timing of the test sequences. Improper management of the enable signal En can cause the latch to malfunction, resulting in incorrect responses to the SSS and RRR inputs and subsequent test failures




```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/21/2025 03:56:37 PM
// Design Name:
// Module Name: sr_latch_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module sr_latch (Q, notQ, En, S, R);
 // Port list
 output wire Q, notQ; // Outputs of the latch
 input wire En, S, R; // Enable, Set, and Reset inputs
 // Intermediate nets
 wire nandSEN, nandREN; // Intermediate signals from NAND gates
 // NAND gates to create the SR latch functionality
 // Generates intermediate signal for Set input
 nand #4 nand0 (nandSEN, S, En);
 nand #4 nand1 (nandREN, R, En);// Generates intermediate signal for Reset input
 nand #4 nand2 (Q, nandSEN, notQ); // Generates Q output based on nandSEN and
 nand #4 nand3 (notQ, nandREN, Q); // Generates notQ output based on nandREN and Q
 endmodule
```

```
add_wave: Time (s): cpu = 00:00:00.33 ; elapsed = 00:00:0
# run 1000ns
                SR-latch Reset Test failed: X should be 1
               SR-latch Hold 0 Test passed
                  SR-latch Set Test failed: 3 should be 2
               SR-latch Hold 1 Test passed
     SR-latch Reset from Set Test failed: 3 should be 1
      SR-latch Enable Hold Test 1 passed
      SR-latch Enable Hold Test 2 passed
      SR-latch Enable Hold Test 3 passed
      SR-latch Enable Hold Test 4 passed
Some tests failed
$stop called at time : 100 ns : File "/home/ugrads/n/navi
```

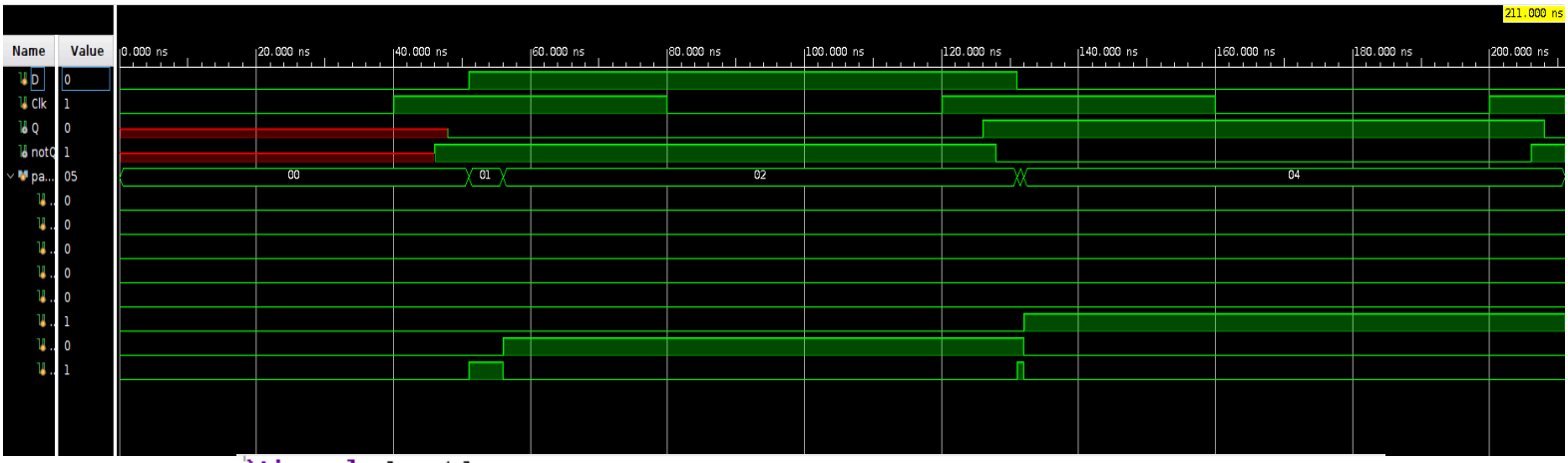
Experiment Part 1 : (D_latch)




```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/21/2025 04:13:55 PM
// Design Name:
// Module Name: d_latch_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module d_latch(Q, notQ, En, D);
    input wire D, En; // only 2 inputs for the D-latch
    output wire Q, notQ; // 2 outputs, Q, and ~Q

    wire nandEnD, nandEnDnot; // Internal wires

    nand #2 nand0(Q, nandEnD, notQ);       // NAND gate creating Q output using D and control signals
    nand #2 nand1(notQ, nandEnDnot, Q);    // NAND gate creating notQ output, cross-coupled with Q
    assign nandEnD = ~(En & D);            // Generates control signal when Enable and D are high
    assign nandEnDnot = ~(En & ~D);        // Generates control signal when Enable is high and D is low
endmodule
```

```
#       Send_msg_to Add_wave-1 WARNING  No top level s
#   }
# }
# run 1000ns
            D-latch Enable Test 1 passed
            D-latch Enable Test 2 passed
              D-latch Hold Test 1 passed
              D-latch Hold Test 2 passed
              D-latch Hold Test 3 passed
              D-latch Hold Test 4 passed
All tests passed
$stop called at time : 70 ns : File "/home/ugrads/p/r
```

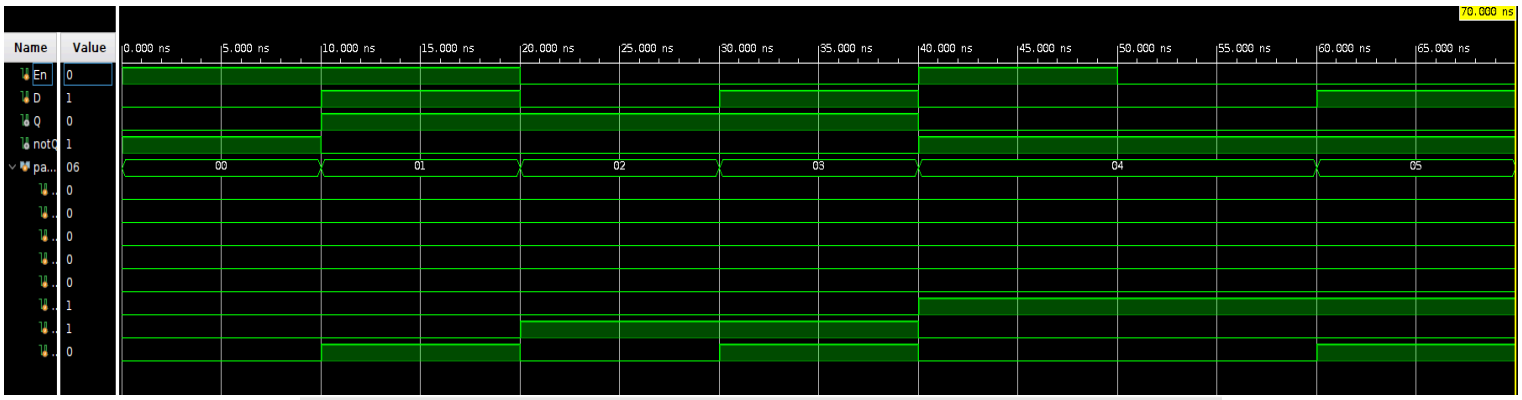
Part 1 : (D_Flip_Flop)




```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/21/2025 04:23:58 PM
// Design Name:
// Module Name: d_flip_flop_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module d_flip_flop(Q, notQ, Clk, D); // D flip-flop with outputs Q, notQ and inputs Clk D

 output wire Q, notQ; // Outputs
 input wire Clk, D; // Inputs
 wire notClk; // Inverted clock
 wire notNotClk; // Double inverted clock
 wire Qm; // Master output
 wire notQm; // Inverted master output
 not #2 not0(notClk, Clk); // Invert Clk
 not #2 not1(notNotClk, notClk); // Invert notClk
 d_latch #2 master(Qm, notQm, notClk, D); // Master latch
 d_latch #2 slave(Q, notQ, notNotClk, Qm); // Slave latch
 endmodule
```

```
# }
# run 1000ns
         D flip-flop Store 0 Test passed
          D flip-flop Hold 0 Test passed
              D flip-flop Store 1 passed
          D flip-flop Hold 1 Test passed
  flip-flop Store 0 Test Again... passed
All tests passed
```

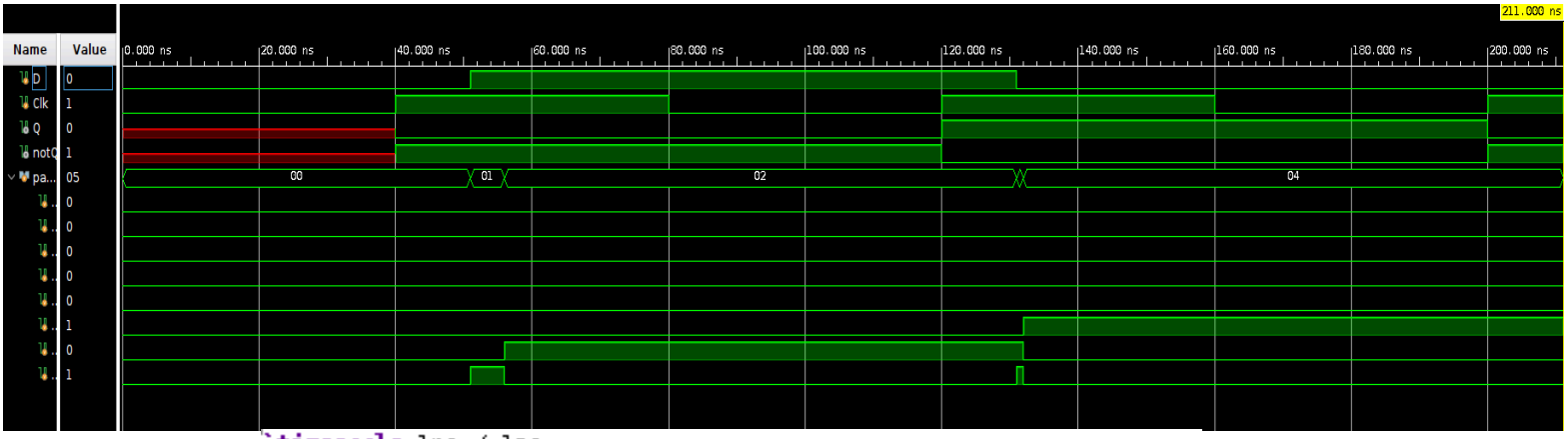
Part 1 :(D_latch_behavioral)




```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/21/2025 04:32:30 PM
// Design Name:
// Module Name: d_latch_behavioral_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module d_latch_behavioral (
 output reg Q, // Driven with a behavioral statement
 output wire notQ, // Driven with a dataflow statement
 input wire D, En // Input wires
 );
 // Describe behavior of D latch
 always @ (En or D) // Trigger on changes to En or D
 begin
 if (En) // If En is high
 Q=D; //Set QtoD
 // else Q remains unchanged (implicit)
 end
 assign notQ = ~Q; // notQ is the inverse of Q
 endmodule
```

```
add_wave: Time (s): cpu = 00:00:00.2 ; elapsed = 00:00:0
# run 1000ns
            D-latch Enable Test 1 passed
            D-latch Enable Test 2 passed
              D-latch Hold Test 1 passed
              D-latch Hold Test 2 passed
              D-latch Hold Test 3 passed
              D-latch Hold Test 4 passed
All tests passed
$stop called at time : 70 ns : File "/home/ugrads/p/pavi
```

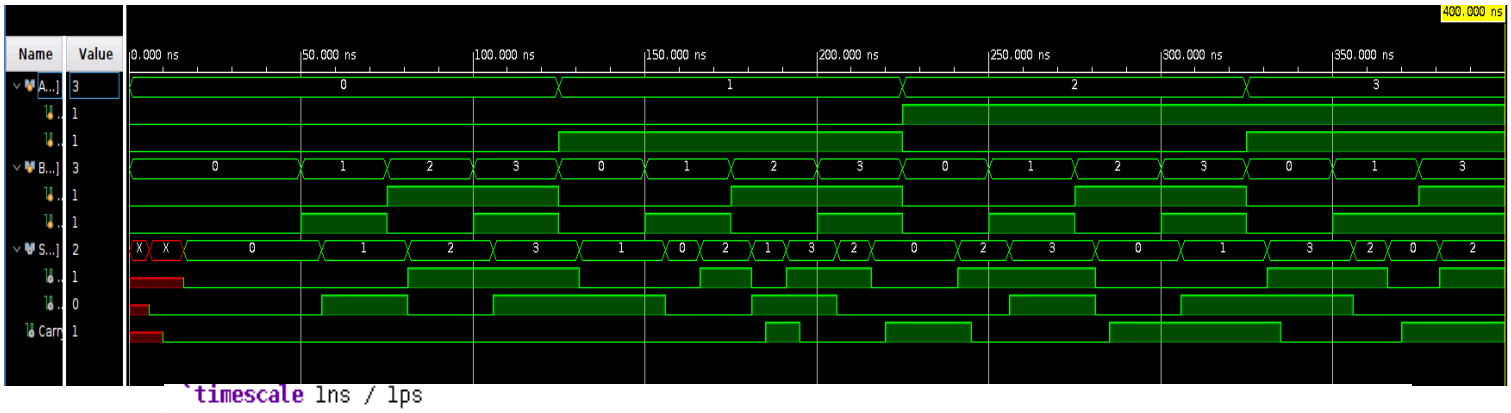
Part 1: (D_flip_flop_behavioral)




```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/21/2025 04:39:24 PM
// Design Name:
// Module Name: d_flip_flop_behavioral_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module d_flip_flop_behavioral (
 output reg Q, // Output Q (driven by a behavioral statement)
 output wire notQ, // Output notQ (driven by a dataflow statement)
 input wire D, // Input D
 input wire Clk // Clock signal
 );
 // Describe behavior of D flip-flop
 always @ (posedge Clk) // Trigger on positive edge of Clk
 begin
 Q<=D; //Non-blocking assignment to update Q
 end
 assign notQ = ~Q; // notQ is the inverse of Q
 endmodule
```

```
        # }
        # run 1000ns
                D flip-flop Store 0 Test passed
                 D flip-flop Hold 0 Test passed
                     D flip-flop Store 1 passed
                 D flip-flop Hold 1 Test passed
          flip-flop Store 0 Test Again... passed
        All tests passed
        $stop called at time : 211 ns : File "/home/ugrads/n/nayilal
```

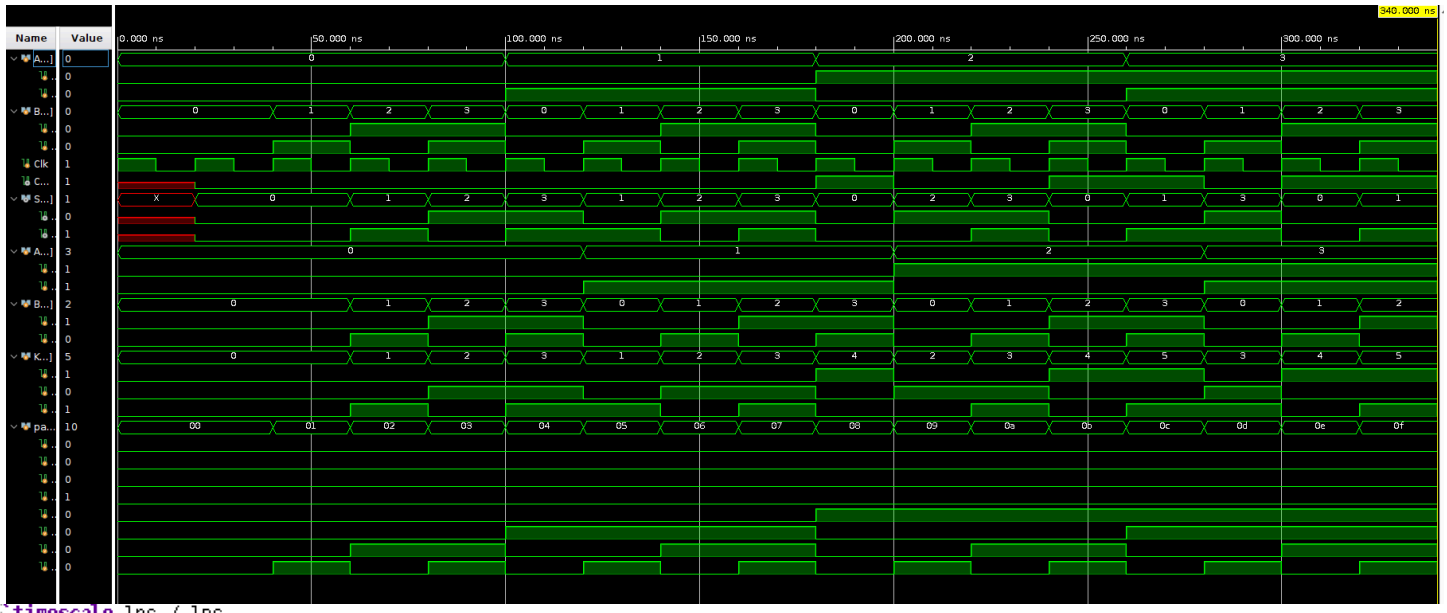
Experiment Part 2 : (Adder _2bit)




```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/21/2025 04:46:14 PM
// Design Name:
// Module Name: adder_2bit_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module adder_2bit(Carry, Sum, A, B); // 2-bit adder module with inputs A, B and outpust sum and carry

 input wire [1:0] A, B; // 2-bit input wires A and B
 output wire [1:0] Sum; // 2-bit output for Sum
 output wire Carry; // Output Carry
 wire Cinter; // Intermediate carry wire
 // Instantiate first full adder
 full_adder add1(Sum[0], Cinter, A[0], B[0], 1'b0); // Add least significant bits
 // Instantiate second full adder
 full_adder add2(Sum[1], Carry, A[1], B[1], Cinter); // Add most significant bits
 endmodule
```

```
# run 1000ns
Hey! The UUT passed this test vector...
Test vector passed!!!
Test vector passed!!!
Test vector passed!!!
Test vector passed!!!
Test vector passed!!!
Test vector passed!!!
Test vector passed!!!
Test vector passed!!!
Test vector passed!!!
Test vector passed!!!
Test vector passed!!!
Test vector passed!!!
Test vector passed!!!
Test vector passed!!!
$stop called at time : 400 ns : File "/home/ugrads/p
```

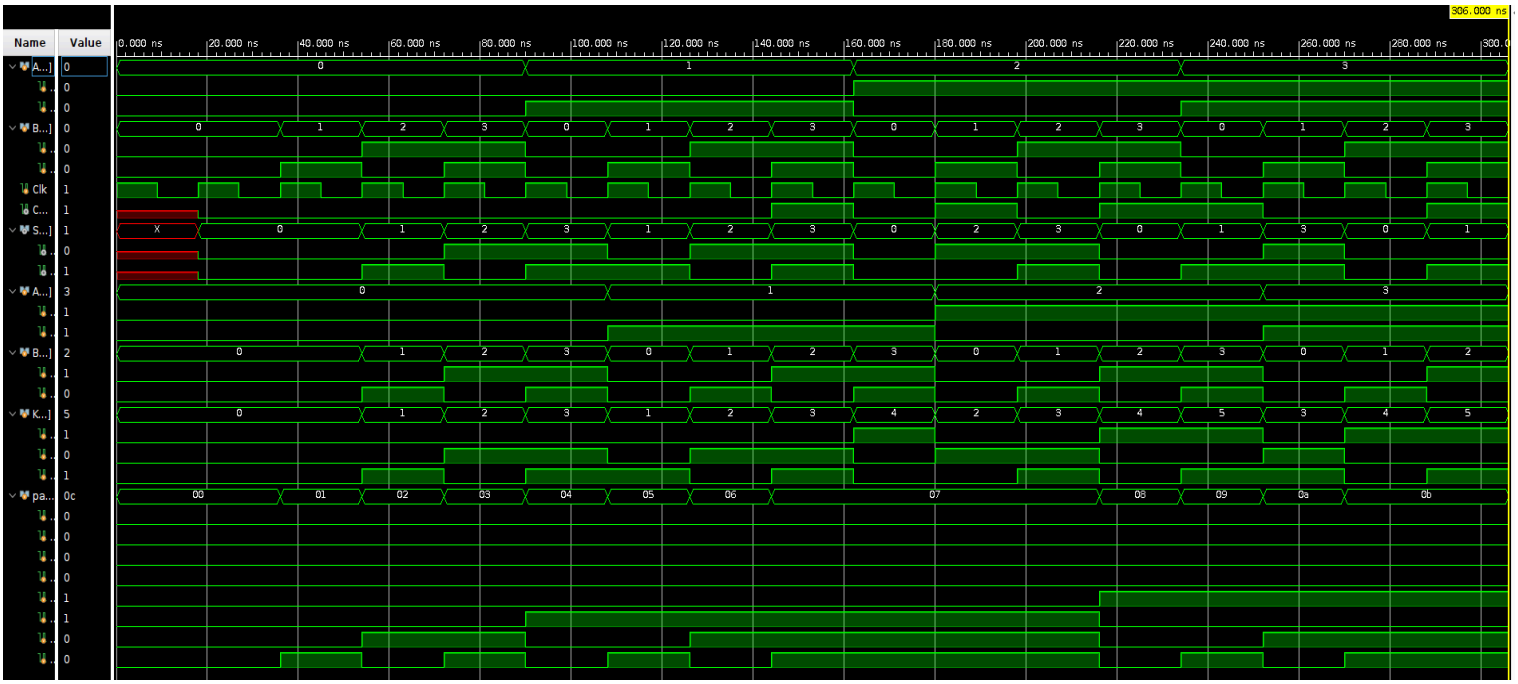
Experiment Part 2 : (Adder_synchronous) (Clock Period 20)




```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/21/2025 05:06:24 PM
// Design Name:
// Module Name: adder_synchronous_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module adder_synchronous(Carry_reg, Sum_reg, Clk, A, B);
 /////Output ports are registers, since they need to hold state */
    output reg Carry_reg;
    output reg [1:0] Sum_reg;
   /////Inputs are still wires */
    input wire Clk;
    input wire [1:0] A, B;
    reg [1:0] A_reg, B_reg; // connect to 2-bit registers
    wire Carry; // need this to connect to the registers later
    wire [1:0] Sum;

    /* 2-bit adder is instantiated */
    adder_2bit a_2_b(Carry, Sum, A_reg, B_reg);

    /* this block describes two 2-bit registers */

    /* inputs */
    always@ (posedge Clk) // trigger on pos edge
        begin
            A_reg <= A; // non-block assignments, concurrent
           B_reg <= B;
        end
    /* outputs */
    always@ (posedge Clk)
        begin
            Carry_reg <= Carry; // wires can drive registers
            Sum_reg <= Sum; // same statement, different bit-width
        end
endmodule
```

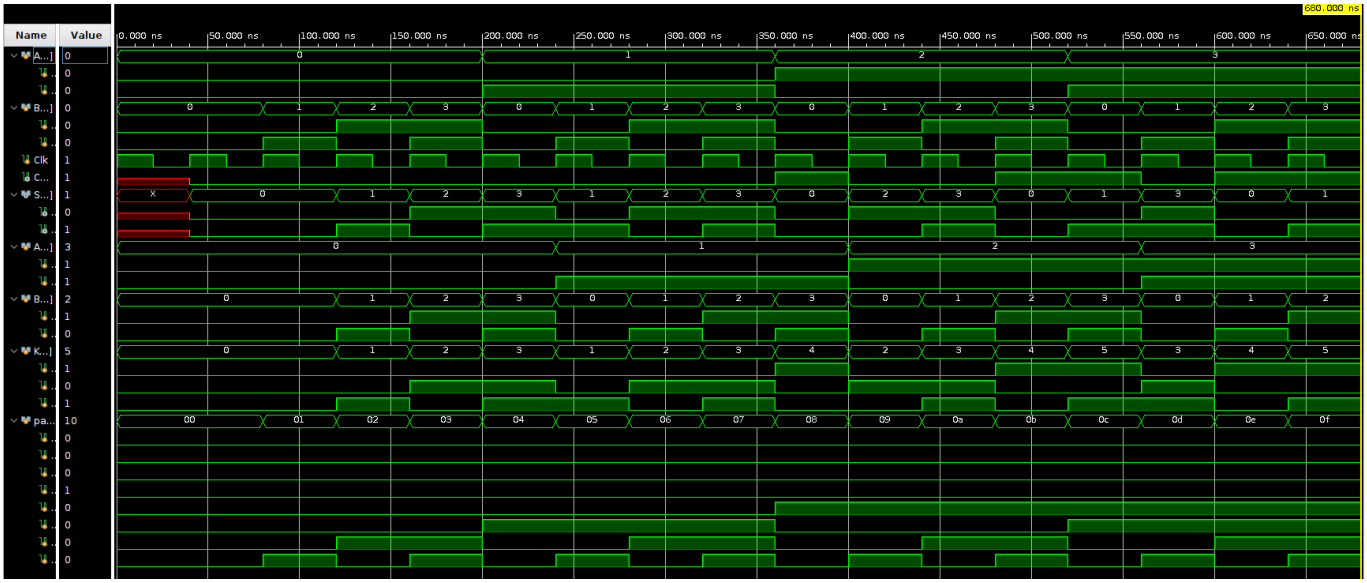
```
add_wave: Time (s): cpu = 00:00:01 ; elapsed = 00:00:06 . M
# run 1000ns
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
All tests passed
$stop called at time : 340 ns : File "/home/ugrads/n/navil
```

Experiment Part 2 : (Adder_synchronous) (Clock Period 18)




```
# }
# run 1000ns
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test failed: 07 should be 03
            Synchronous Adder Test failed: 00 should be 04
            Synchronous Adder Test failed: 06 should be 02
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test passed
            Synchronous Adder Test failed: 00 should be 04
            Synchronous Adder Test passed
Some tests failed
```

Experiment Part 2 : (Adder_synchronous) (Clock Period 30)




```
add_wave: Time (s): cpu = 00:00:00.88 ; elapsed = 00:00:18 . Memor
# run 1000ns
                Synchronous Adder Test passed
                Synchronous Adder Test passed
                Synchronous Adder Test passed
                Synchronous Adder Test passed
                Synchronous Adder Test passed
                Synchronous Adder Test passed
                Synchronous Adder Test passed
                Synchronous Adder Test passed
                Synchronous Adder Test passed
                Synchronous Adder Test passed
                Synchronous Adder Test passed
                Synchronous Adder Test passed
                Synchronous Adder Test passed
                Synchronous Adder Test passed
                Synchronous Adder Test passed
                Synchronous Adder Test passed
                Synchronous Adder Test passed
All tests passed
```

Post lab deliverable

1.Modifying the delay from 2 ns to 4 ns caused failures in several test cases, particularly in the reset and set tests, which impacts test bench validity. The 10 ns wait period is insufficient, as multiple NAND gates introduce a cumulative delay that prevents outputs from stabilizing before being read.

2.The increased delay in output synchronization causes the Q and not Q outputs to fail in accurately capturing the clock signal at its rising edge. This misalignment can lead to incorrect output states and affects the reliability of the sequential logic.

3.Behavioral Verilog simulations do not account for the propagation delays observed in structural simulations, implying that latches and flip-flops are synchronized. This discrepancy can lead to misleading interpretations of the timing behavior in the design.

4.Assessment of Worst-Case Propagation Delay: The design exhibits a worst-case propagation delay of 20 ns, indicating potential suboptimal circuit architecture.

5.Clock Rate and Circuit Efficiency Considerations:The maximum frequency matches the CLOCK_PERIOD set in the testbench. Increasing the adder's width reduces the clock rate. To speed it up, raise the clock rate of the adder module. In testing, CLOCK_PERIOD values of 20 and 40 worked.

Important Student Feedback

One of the most important aspects of this lab that I appreciated was the comprehensive coverage of Verilog, which provided a wealth of learning opportunities, and the diverse range of intricate waveforms that were showcased. Overall, I found this lab to be quite interesting, with a clear and concise set of instructions, and I really enjoyed this valuable learning experience