ECEN 248 - Lab Report

Lab Number: 7

Lab Title: Introduction to Sequential Logic

Section Number: 513

Student's Name: Abhishek Bhattacharyya

Student's UIN: 731002289

Date: 3/29/2023

TA: Hesam Mazaheri

Objectives:

The main objective of this lab is to use Verilog for digital circuit simulation of sequential logic circuits. While the previous lab focused on structural and data flow modeling for describing digital circuits in Verilog HDL, this lab will implement sequential circuits and memory cells based on what has been covered previously. The first part of this lab is implementing memory cells such as latches and flip-flops, describing these components at the gate level using Verilog, and then simulating them within Vivado. The second part is implementing behavioral versions of these digital circuits which can be implemented on an FPGA board. The final part of this lab is implementing a synchronous adder circuit based on work from Lab 5.

This lab also introduces simulation delays into combinational logic and observing the effects it has on clock timing. Vivado is used to simulate the period of a clock in a digital circuit and the effects of reducing the clock delay in the circuit are explored to the breaking point (until the circuit stops passing tests).

Design:

Lab 5 and Lab 6 served to help familiarize students with setting up an environment in Vivado and learning how to code in structural, dataflow, and behavioral Verilog. Using this knowledge, a new project was started in Vivado for Lab 7 where the code for each module was completed, along with running each testbench and programming the testbench file for the 2-bit adder.

To start, first open the terminal and run the two commands provided in Lab 5 to start Vivado. Then, create a new project in Vivado named 'lab7' to house all the modules. Now, Vivado is ready to start coding each of the modules for this lab. First, code the sr_latch.v module using the sample code in the lab manual and test it on the test bench. Once this passes, collect all data required for the lab, complete additional testing steps, and then move on to the next module (d-latch.v). These coding steps were followed until all code files (sr_latch, d_latch, d_flip_flip, d_latch_behavioral, d_flip_flop_behavioral, adder_2bit, adder_2bit_tb, and adder_synchronus) were completed. Provided below are all completed code files for this lab.

```
`timescale 1ns / 1ps     // delay unit is 1ns
`default_nettype none

// Code to describe an SR Latch module using structural verilog
module sr_latch(Q, notQ, En, S, R);
    // input/output port list
    output wire Q, notQ;
    input wire En, S, R;

    // intermediate nets
    wire nandSEN = !(S & En);
    wire nandREN = !(R & En);

    //delay of gate nand0 is 2ns
    nand #2 nand0 (Q, nandSEN, notQ);
    nand #2 nand1 (notQ, nandREN, Q);

endmodule
```

Figure 1: Code w/ comments for SR Latch

```
`timescale 1ns/ 1ps //the delay unit is 1ns
`default_nettype none

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

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

//wire nandSEn, nandREn;

wire nandDEn, nandnotDEn,notD;

not #2 notd(notD,D);

nand #2 nand0(nandDEn, D, En);

nand #2 nand1(nandnotDEn,notD, En);

nand #2 nand2(Q,nandDEn,notQ);

nand #2 nand3(notQ,nandnotDEn,Q);


//the rest will be filled in

endmodule
```

Figure 2: Code w/ comments for D Latch

```
`timescale 1ns/ 1ps //the delay unit is 1ns
`default_nettype none

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

//declare all ports

output wire Q, notQ; //outputs of slave latch

input wire Clk, D;

/*intermediate nets*/

wire notClk, NotClk, notNotClk;

wire Qm;//output of master latch

//notQm is used in instantiation
//but left unconnected
wire notQm;

//instatiate the NOT gates, adds 2ns delay

not not0(notClk,Clk);
not not1(notNotClk,notClk);

d_latch master_en(Qm,notQm,notClk,D);

d_latch servant_en(Q,notQ,notNotClk,Qm);


// instatiate d latches

endmodule
```

Figure 3: Code w/ comments for D Flip Flop

```
`timescale 1ns/ 1ps //the delay unit is 1ns
`default_nettype none

module d_latch_behavioral(

output reg Q, //driven with behavioural statements
output wire notQ,//driven with dataflow statemtn
input wire D, En //wires can drive regs

);

/*describe behaviour of D-latch*/
always@(En or D)

if(En) //we will look at what this says: if En!= 1'b0
Q=D;
//else Q=Q;//this is implied so uneccessary

assign notQ = ~Q; //regs can drive wires

endmodule
```

Figure 4: Code w/ comments for D Latch Behavioral

```
`timescale 1ns/ 1ps //the delay unit is 1ns
`default_nettype none

module d_flip_flop_behavioral(

output reg Q,
output wire notQ,
input wire D,
input wire Clk //this is the clock signal
);
/*describe behaviour of D flip-flop*/

//posedge means positive or rising edge

always@(posedge Clk) //trigger when clk goes high
Q<=D;//non blocking assignement
assign notQ=~Q;
endmodule
```

Figure 5: Code w/ comments for D Flip Flop Behavioral

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

//carry is a 1 bit output
output wire [1:0]Sum,Carry;
input wire [1:0]A,B;

wire C0;
//wire c1;

//Sum is a 2-bit output

//A, B are are 2 bit inputs

//we then need the adder_2bit tb and do the proper implementation

full_adder fulleradder0(Sum[0],C0,A[0],B[0],1'b0);

full_adder fulladder1(Sum[1],Carry,A[1],B[1],C0);

endmodule
```

Figure 6.1: Code w/ comments for Adder 2-bit

```
1   `timescale 1ns / 1ps
2
3   module add_2bit_tb;//a test bench does not have any ports of its own!
4
5       /* Input nets */
6       reg [1:0] A; //these are regs because they are modified in
7       reg [1:0] B; //a behavioral block
8
9       /* Output nets */
10      wire [1:0] Sum; //these are wires because they will be driven
11      wire Carry;//by the inantiated module
12
13      /* Instantiate the Unit Under Test (UUT) */
14      adder_2bit uut ( //this is a different way
15          .A(A),       //to instantiate a module.
16          .B(B),       //the nice thing about this style
17          .Sum(Sum),   //is that the order does not matter!
18          .Carry(Carry)//notice the ports are in a different order!
19      );
20
21
22      /*-this is a behavioral block which is executed only once!  *
23       *-the statements within this behavioral block are executed *
24       *-sequentially because we are using blocking statements    *
25       *-an '=' sign within a behavioral construct is considered a*
26       * blocking statement. We will talk more about this later...*/
27      initial
28        begin
29
30          /* Initialize inputs*/
31          A = 0;
32          B = 0;
33
34          #25; //just delay 25 ns
35          {A,B} = 4'b0000; //stimulate the inputs
36          #25; //wait a bit for the result to propagate
37          //here is where we could put a check to see if the results
38          //are as expected!
39          if({Carry, Sum} != 3'b000)//you could put your own message here
40              $display("Ah crap... something went wrong here...");
41              else
42                  $display("Hey! The UUT passed this test vector...");
43          //let's do it again with a different input...
44          {A,B} = 4'b0001; //stimulate the inputs
45          #25; //wait a bit for the result to propagate
46          //check output
47          if({Carry, Sum} != 3'b001)
48              $display("Your own message here...");
49              else
50                  $display("Test vector passed!!!");
51          //okay this is fun... you try it now...
52
53          //go through all possible input combinations (2^4 = 16)
54          //cut and paste makes this task a lot easier
55
56          {A,B} = 4'b0000; //stimulate the inputs
57          #25; //wait a bit for the result to propagate
58          //here is where we could put a check to see if the results
59          //are as expected!
60          if({Carry, Sum} != 3'b000)//you could put your own message here
61              $display("Ah crap... something went wrong here...");
62              else
63                  $display("Hey! The UUT passed this test vector...");
64
65
66          {A,B} = 4'b0001; //stimulate the inputs
67          #25; //wait a bit for the result to propagate
68          //here is where we could put a check to see if the results
69          //are as expected!
70          if({Carry, Sum} != 3'b001)//you could put your own message here
71              $display("Ah crap... something went wrong here...");
72              else
73                  $display("Hey! The UUT passed this test vector...");
74
75
76          {A,B} = 4'b0010; //stimulate the inputs
77          #25; //wait a bit for the result to propagate
78          //here is where we could put a check to see if the results
79          //are as expected!
80          if({Carry, Sum} != 3'b010)//you could put your own message here
81              $display("Ah crap... something went wrong here...");
82              else
83                  $display("Hey! The UUT passed this test vector...");
84
85          {A,B} = 4'b0011; //stimulate the inputs
86          #25; //wait a bit for the result to propagate
87          //here is where we could put a check to see if the results
88          //are as expected!
89          if({Carry, Sum} != 3'b011)//you could put your own message here
90              $display("Ah crap... something went wrong here...");
91              else
92                  $display("Hey! The UUT passed this test vector...");
93
94          {A,B} = 4'b0100; //stimulate the inputs
95          #25; //wait a bit for the result to propagate
96          //here is where we could put a check to see if the results
97          //are as expected!
98          if({Carry, Sum} != 3'b001)//you could put your own message here
99              $display("Ah crap... something went wrong here...");
100             else
101                 $display("Hey! The UUT passed this test vector...");
102
103         {A,B} = 4'b0101; //stimulate the inputs
104         #25; //wait a bit for the result to propagate
105         //here is where we could put a check to see if the results
106         //are as expected!
107         if({Carry, Sum} != 3'b010)//you could put your own message here
108             $display("Ah crap... something went wrong here...");
109             else
110                 $display("Hey! The UUT passed this test vector...");
111
112         {A,B} = 4'b0110; //stimulate the inputs
113         #25; //wait a bit for the result to propagate
114         //here is where we could put a check to see if the results
115         //are as expected!
116         if({Carry, Sum} != 3'b011)//you could put your own message here
117             $display("Ah crap... something went wrong here...");
118             else
119                 $display("Hey! The UUT passed this test vector...");
120
121         {A,B} = 4'b0111; //stimulate the inputs
122         #25; //wait a bit for the result to propagate
123         //here is where we could put a check to see if the results
124         //are as expected!
125         if({Carry, Sum} != 3'b100)//you could put your own message here
126             $display("Ah crap... something went wrong here...");
127             else
128                 $display("Hey! The UUT passed this test vector...");
129
130         {A,B} = 4'b1000; //stimulate the inputs
131         #25; //wait a bit for the result to propagate
132         //here is where we could put a check to see if the results
133         //are as expected!
134         if({Carry, Sum} != 3'b010)//you could put your own message here
135             $display("Ah crap... something went wrong here...");
136             else
137                 $display("Hey! The UUT passed this test vector...");
138
139         {A,B} = 4'b1001; //stimulate the inputs
140         #25; //wait a bit for the result to propagate
141         //here is where we could put a check to see if the results
142         //are as expected!
143         if({Carry, Sum} != 3'b011)//you could put your own message here
144             $display("Ah crap... something went wrong here...");
145             else
146                 $display("Hey! The UUT passed this test vector...");
147
148         {A,B} = 4'b1010; //stimulate the inputs
149         #25; //wait a bit for the result to propagate
150         //here is where we could put a check to see if the results
151         //are as expected!
152         if({Carry, Sum} != 3'b100)//you could put your own message here
153             $display("Ah crap... something went wrong here...");
154             else
155                 $display("Hey! The UUT passed this test vector...");
156
157         {A,B} = 4'b1011; //stimulate the inputs
158         #25; //wait a bit for the result to propagate
159         //here is where we could put a check to see if the results
160         //are as expected!
161         if({Carry, Sum} != 3'b101)//you could put your own message here
162             $display("Ah crap... something went wrong here...");
163             else
164                 $display("Hey! The UUT passed this test vector...");
165
166         {A,B} = 4'b1100; //stimulate the inputs
167         #25; //wait a bit for the result to propagate
168         //here is where we could put a check to see if the results
169         //are as expected!
170         if({Carry, Sum} != 3'b011)//you could put your own message here
171             $display("Ah crap... something went wrong here...");
172             else
173                 $display("Hey! The UUT passed this test vector...");
174
175         {A,B} = 4'b1101; //stimulate the inputs
176         #25; //wait a bit for the result to propagate
177         //here is where we could put a check to see if the results
178         //are as expected!
179         if({Carry, Sum} != 3'b100)//you could put your own message here
180             $display("Ah crap... something went wrong here...");
181             else
182                 $display("Hey! The UUT passed this test vector...");
183
184         {A,B} = 4'b1110; //stimulate the inputs
185         #25; //wait a bit for the result to propagate
186         //here is where we could put a check to see if the results
187         //are as expected!
188         if({Carry, Sum} != 3'b101)//you could put your own message here
189             $display("Ah crap... something went wrong here...");
190             else
191                 $display("Hey! The UUT passed this test vector...");
192
193         {A,B} = 4'b1111; //stimulate the inputs
194         #25; //wait a bit for the result to propagate
195         //here is where we could put a check to see if the results
196         //are as expected!
197         if({Carry, Sum} != 3'b110)//you could put your own message here
198             $display("Ah crap... something went wrong here...");
199             else
200                 $display("Hey! The UUT passed this test vector...");
201
202         //when we are done, let's stop the simulation
203         $stop;
204     end
205
206 endmodule
```

Figure 6.2: Code w/ comments for Adder 2-bit Testbench

```
`timescale 1ns/ 1ps //the delay unit is 1ns
`default_nettype none

/*here we will create the very first synchronous adder*/

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

/*output ports are regs*/

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

/*inputs are still wires*/

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

/*here are our intermediate nets*/

reg[1:0] A_reg, B_reg; //we will use these two-bit registers


wire Carry; //this needs to connect to registers described below

wire [1:0] Sum;

/*here we instantiate our 2bit adder*/

/*this behavioural block describes 2 bit registers*/

adder_2bit adder0(Carry,Sum,A,B);

always@(posedge Clk) //this triggers on the positive edge of the clock

begin

A_reg<=A;//nonblocking assignments are used because we want the two statements to happen concurrently

B_reg<=B;

end

/*here is described the registers for the result*/
always@(posedge Clk)
begin

Carry_reg<=Carry; //this is to basically drive the registers
Sum_reg<=Sum;

end

endmodule

```

Figure 7: Code w/ comments for Adder Synchronous

Results:

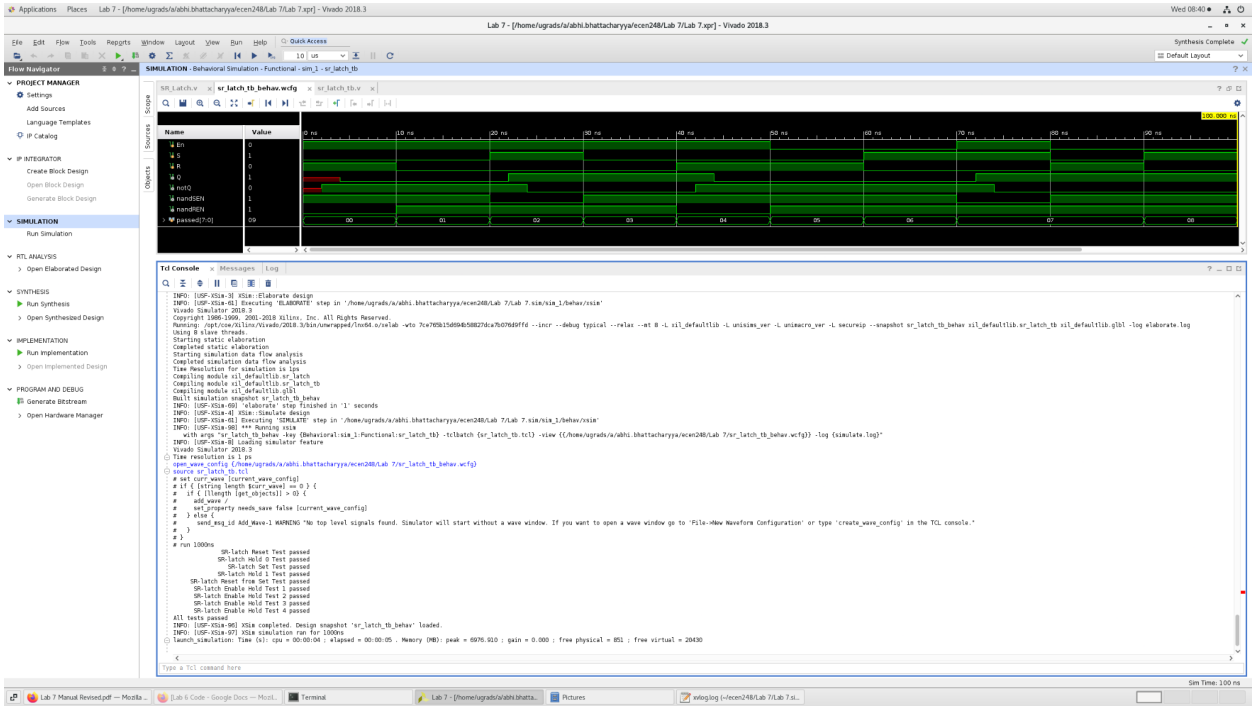


Figure 8.1: SR_Latch with #2 delay, waveform and console output

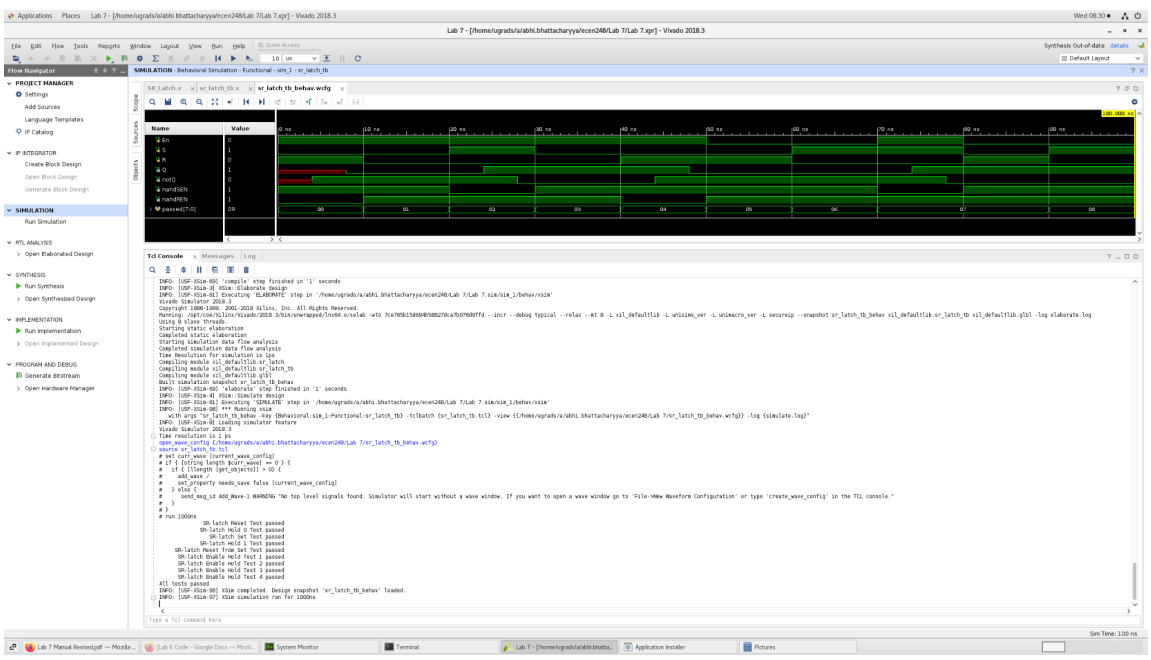


Figure 8.2: SR_Latch with #4 delay, waveform and console output

Comparing the above SR Latch results to the given Table 1:

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	

Looking at the En, S, and R waveforms, and considering how Q and notQ waveforms change, the behavior of the gate is consistent with the table. When the En pin is low (0), Q and notQ are held; when the En pin is high (1), Q and notQ respond to S and R as expected. All tests pass.

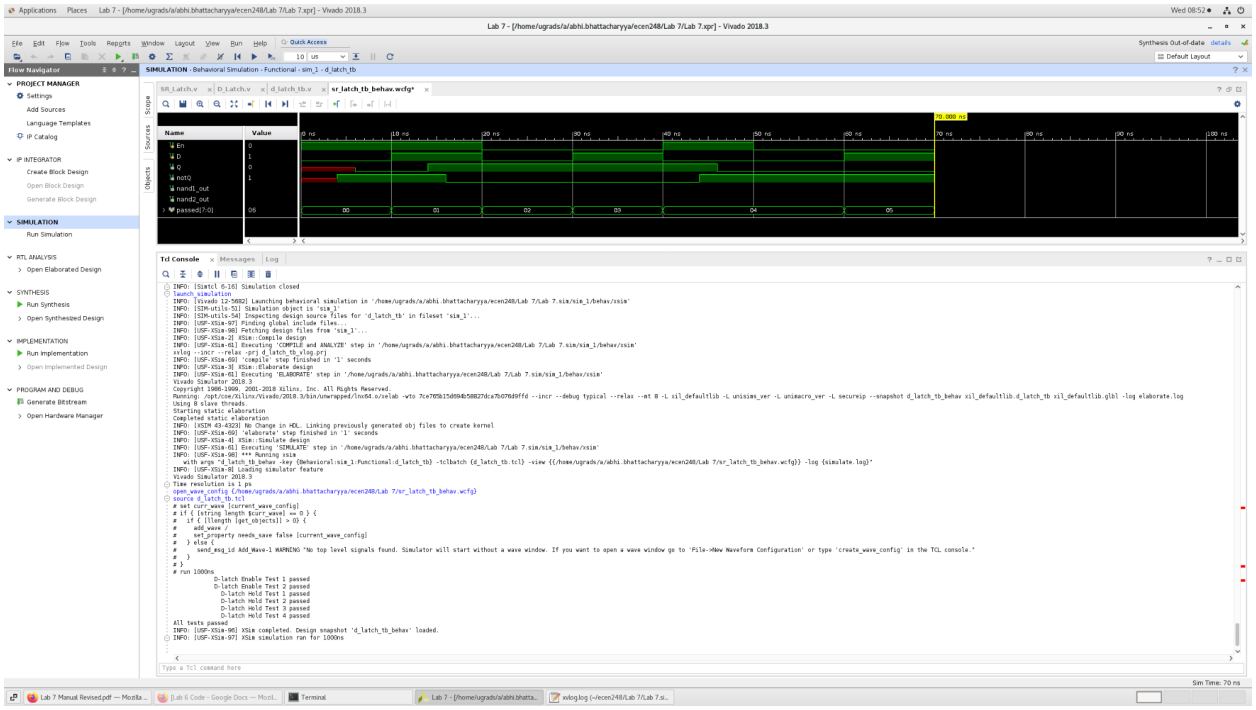


Figure 9: D Latch waveform and console output

Comparing the above D Latch results to the given Table 2:

Table 2: D-latch Function Table

En	D	Q	$\overline{Q}$
0	X	hold	
1	0	0	1
1	1	1	0

Looking at the En and D waveforms, and considering how Q and notQ waveforms change, the behavior of the gate is consistent with the table. When the En pin is low (0), Q and notQ are held; when the En pin is high (1), Q and notQ respond to D as expected (when D goes high and En is high, Q goes high and notQ goes low, with a gate propagation delay exhibited. When D goes low and En is high, Q goes low and notQ goes high). All tests pass.

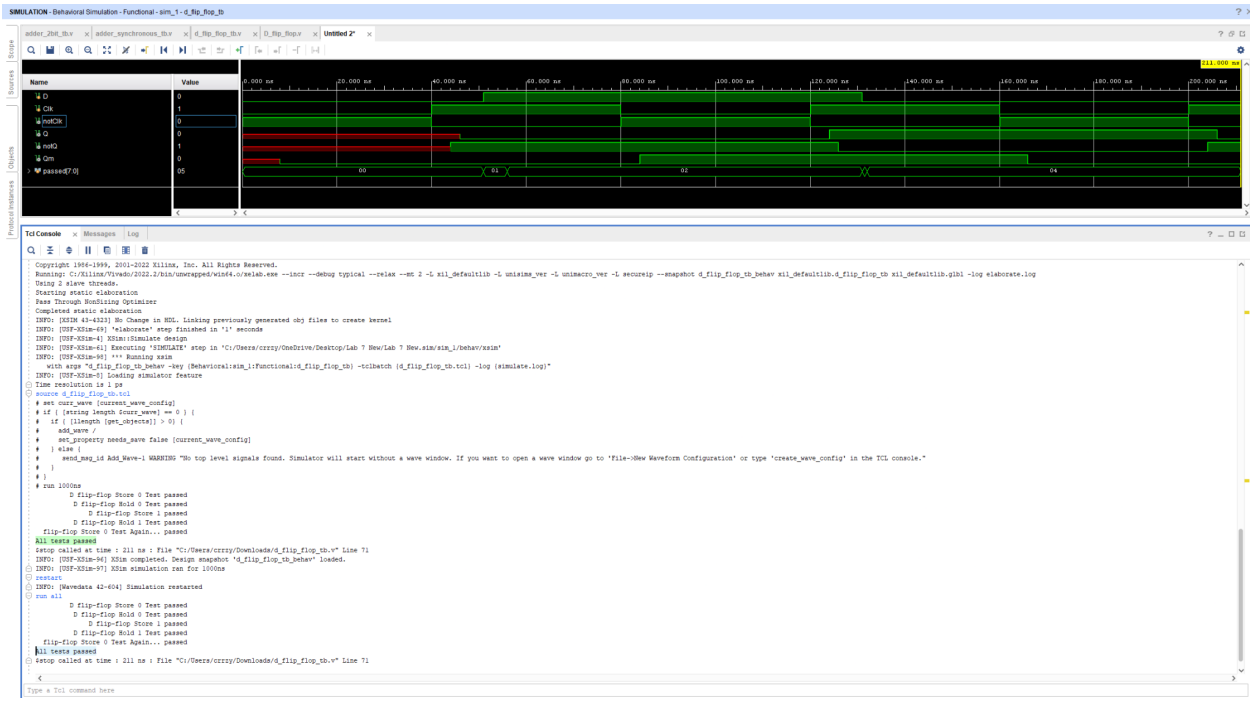


Figure 10: D Flip-Flop waveform and console output

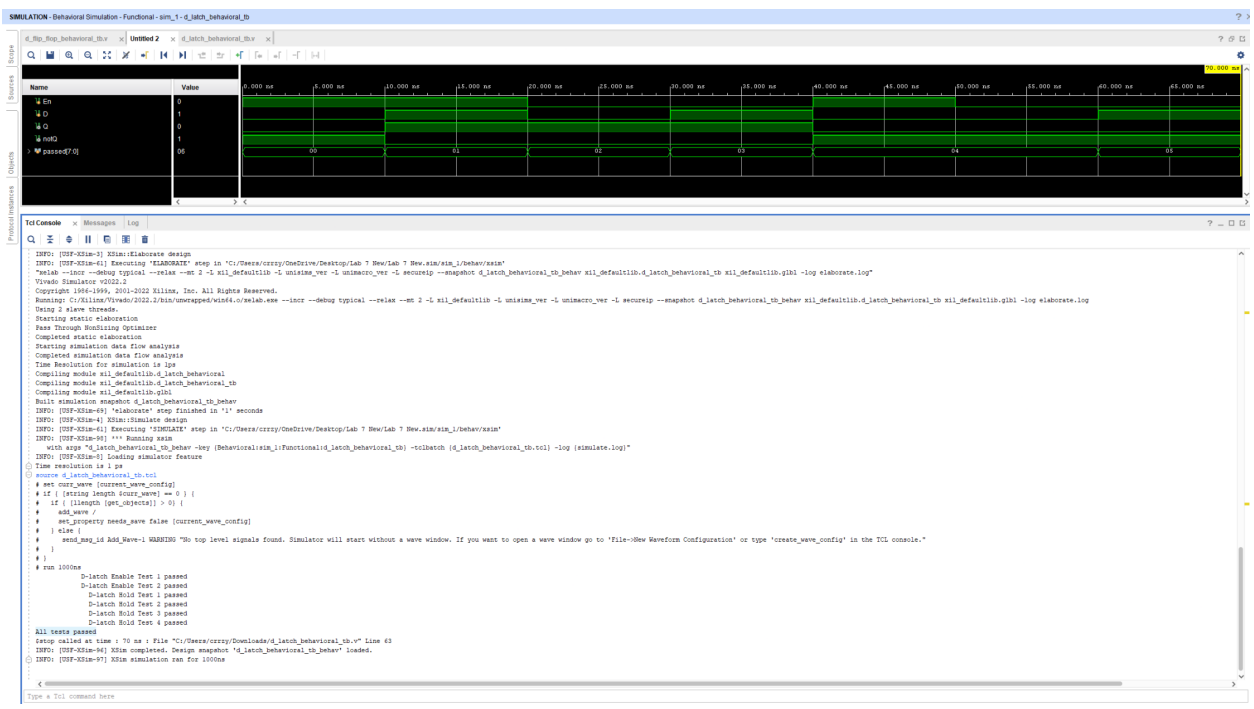


Figure 11: D Latch Behavioral waveform and console output

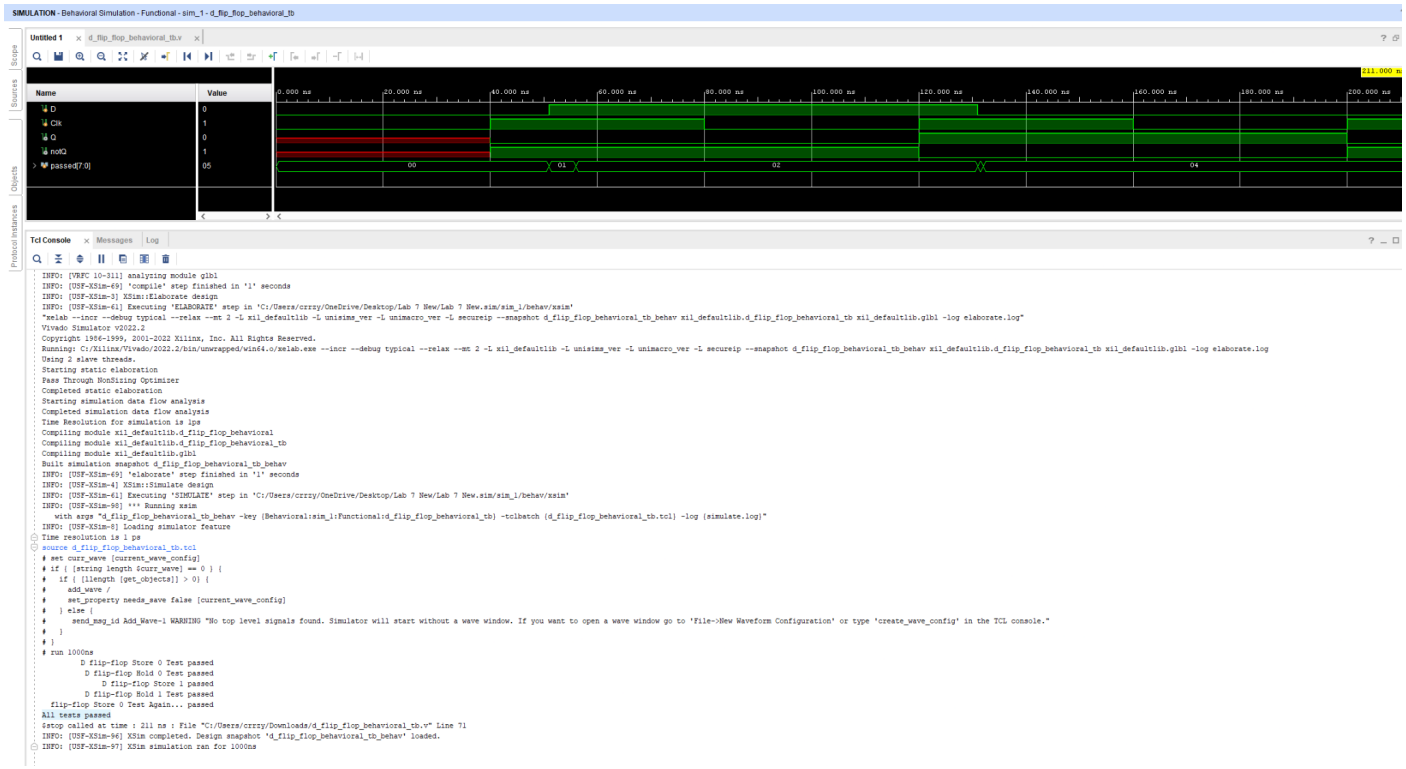


Figure 12: D Flip Flop Behavioral waveform and console output

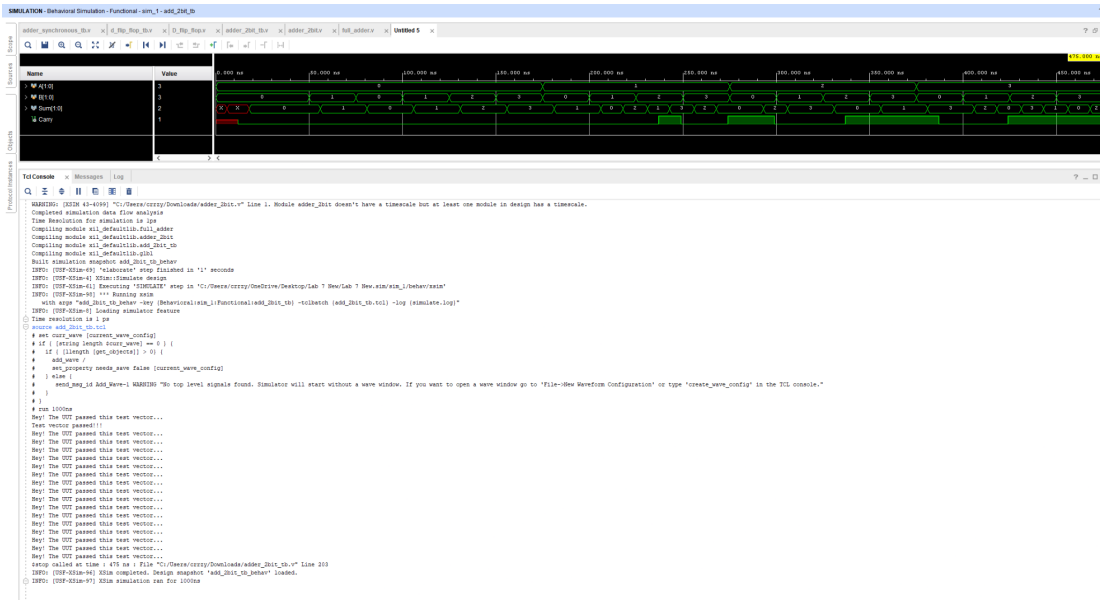


Figure 13.1: Adder 2 bit waveform and console output

Using the simulation waveform to determine the worst-case propagation delay through the ripple carry adder, we can take a look at the carry signal and the time markers on the simulation:

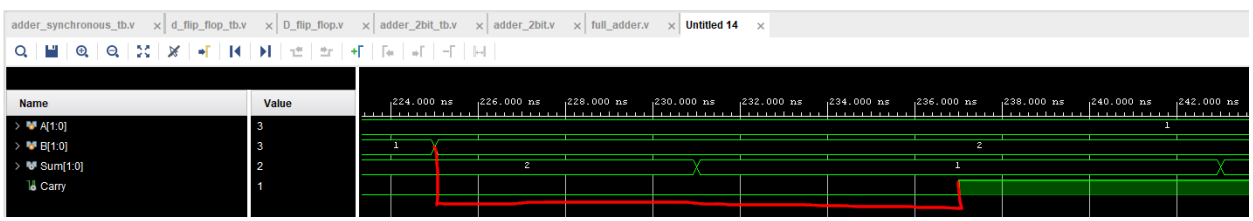


Figure 13.2: Zoomed in Adder 2 bit waveform to analyze worst-case delay

Analyzing the gap between the state change (change in value of B) and the state change in Carry, we can see that the worst case propagation delay is ~12ns for this design. This makes sense, as each full adder has a gate delay of #6 (6ns) and there are 2 full-adders in the ripple adder.

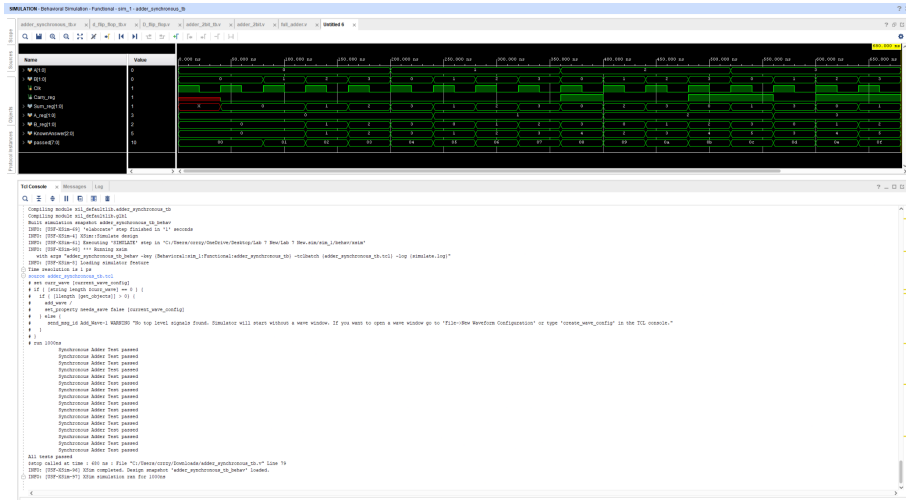


Figure 14.1: Adder synchronous waveform and console output

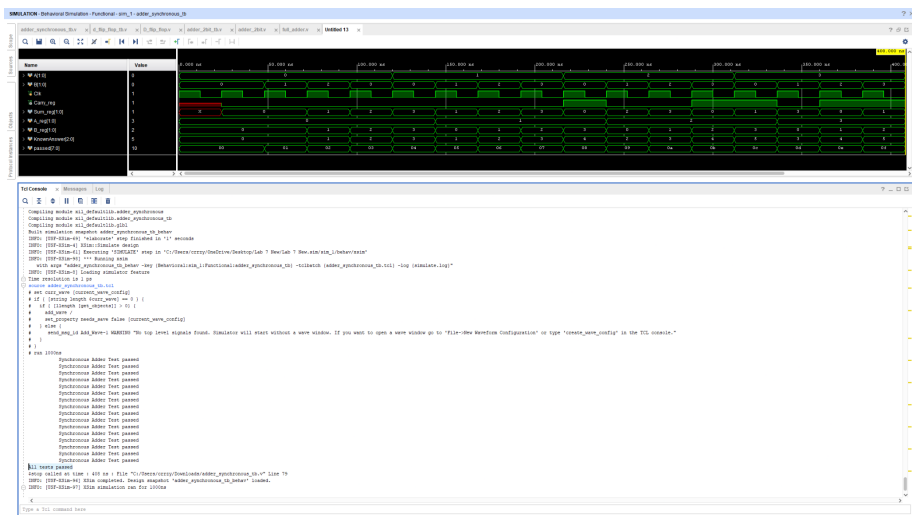


Figure 14.2: Adder synchronous waveform and console output (clock period 24)

We can see above that the fastest possible clock period that resulted in no failed tests for the synchronous adder was a period of 24. This makes sense since the ripple adder's delay is 12ns. Since the code defines the clock as the clock period / 2:

```
/*generate clock signal*/
always
    #(`CLOCK_PERIOD/2) Clk <= ~Clk;
```

Since the worst case delay for the ripple adder is 12ns, it will not run reliably with a delay of less than 12ns, or in this case, a clock period of less than 24 (ns). This is why tests fail with a clock period of less than 24, the ripple adder does not finish in time and outputs garbage data.

Conclusion:

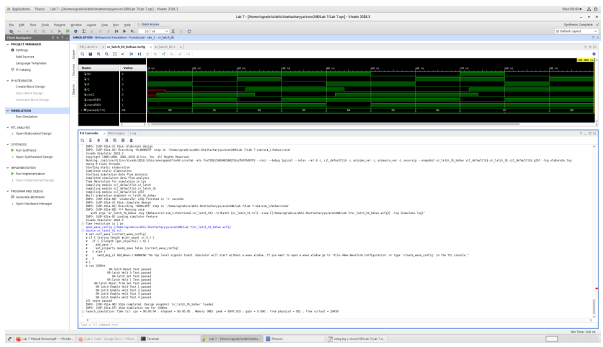
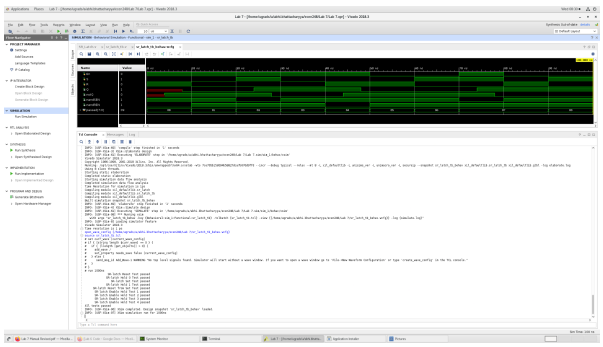
In conclusion, this lab successfully introduced the knowledge of running and setting up Vivado to create digital simulations of sequential and synchronous logic circuits. Using structural modeling, an SR latch, D latch, and D flip flop were coded. Then, a behavioral version of the D latch and D flip flop were created. Finally, a 2-bit adder was imported from Lab 5, which was used to implement a synchronous adder.

Drawing conclusions from the data collected from the synchronous adder, it can be seen that the gate delay of the individual components of a synchronous circuit influences the maximum clock speed (minimum clock period) the circuit can run at. The overall conclusion from the testing done in this lab is that by minimizing gate delay and optimizing circuit design, the clock speed of a circuit can be maximized to achieve improved performance.

Post-lab Deliverables:

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

Looking at the results below, it can be clearly seen that the SR_Latch with the #2 gate delay has a much quicker response time at transitions than the SR_Latch with the #4 gate delay. This is especially prominent in the Q and notQ signals. In a perfect world, these signals would ALWAYS be opposite with no changeover delay - however, with gate delays and in the real world, these signals will overlap a bit at transitions, and this can be clearly seen in the simulation below. In the #2 gate delay simulation, the overlap is much smaller than in the #4 gate delay simulation. This tighter timing/tolerance in the #2 gate delay design means it will result in a much higher quality (and potentially more expensive to fabricate) part, which can be run at a higher clock speed in devices where it is implemented. This effect on clock speed is elaborated on in Post-Lab Questions #4 and #5.

 SR_Latch #2 Delay (2ns)	 SR_Latch #4 Delay (4ns)

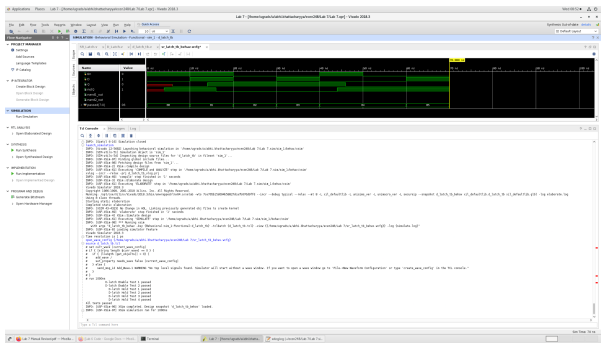
2. For #3 in part 1, check the waveform with the internal signals. Are the latches behaving as expected? Why or why not?

As can be seen in the simulation waveform, the latches are behaving as expected. Looking at the Clk and notClk signals, notClk is a perfect inverse to Clk, so notClk is performing in the gate perfectly as expected. Qm also updates with the notClk signal while Q (slave output) updates with the notNotClk signal (so basically the Clk signal), so the latch is behaving exactly as expected.

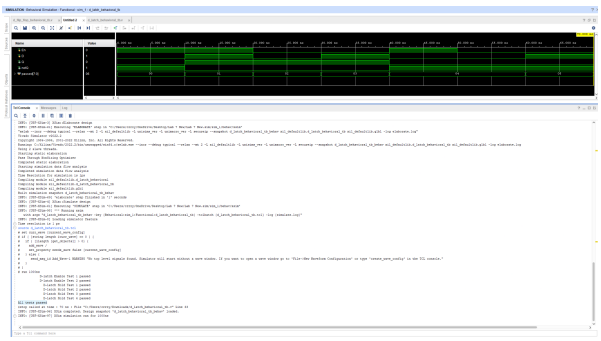
3. For #4 in part 1, compare the waveforms you captured from the behavioral Verilog to those from the structural Verilog. Are they different? If so, how?

Looking at the results below, it can be seen there are some differences, mainly at the transitions between high and low. While the structural Verilog results for both the D-latch and D-flip flop have some transition delay where the signals for multiple channels overlap, the behavioral Verilog outputs perfect waveforms - that is, there is no “overlap” or transition delay, the signals transition perfectly and immediately.

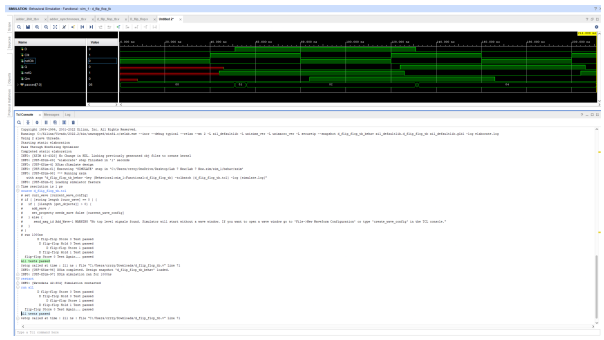
Other than that, in the D Latch structural vs. behavioral comparison, it can be seen that the “Don’t Care” values (in red at the start of the waveform) are different between the structural and behavioral Verilog codes.



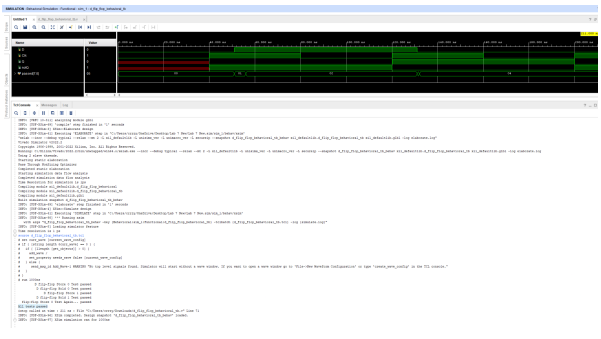
D Latch Structural



D Latch Behavioral



D Flip Flop Structural

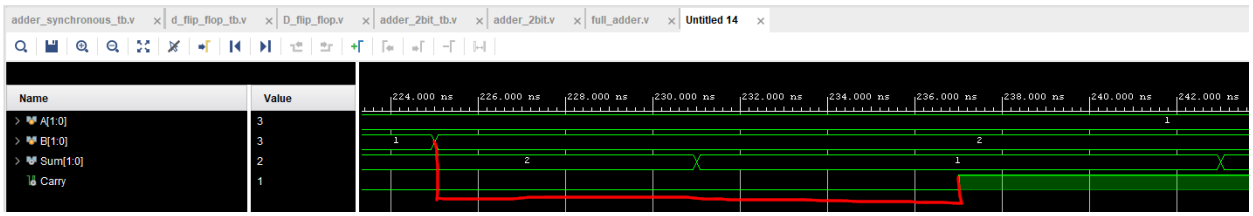


D Flip Flop Behavioral

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

This was covered in the results section, and is repeated here.

Using the simulation waveform to determine the worst-case propagation delay through the ripple carry adder, we can take a look at the carry signal and the time markers on the simulation:



Analyzing the gap between the state change (change in value of B) and the state change in Carry, we can see that the worst case propagation delay is ~12ns for this design. This makes sense, as each full adder has a gate delay of #6 (6ns) and there are 2 full adders in the ripple adder.

5. Based on the clock period you measured for your synchronous adder, what would be the theoretical maximum clock rate (frequency)? 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 maximum clock period measured for the synchronous adder with all tests passing was 18ns. This translates to a theoretical maximum clock rate of f = 1/18ns = 55.556 mHz.

Increasing the width of the adder would necessitate a longer clock period as the added amount of logic leads to longer propagation delays within the circuit, necessitating a longer time for the logic to process. So, the increasing width of the adder would require a lower clock frequency.

We can improve the clock rate of the design by reducing the propagation delay through the circuit. This comes through architectural improvements, down to improving the gate delay through individual gates and the memory cells (latches and flip flops).

Important Student Feedback:

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

I most liked finishing each piece of code and seeing it all come together, it was very satisfying.

I least liked having to write the testbench code. I think that this should have been provided as it lets us focus on writing Verilog code and lets us be sure it tests correctly.

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

Some of the post-lab deliverables did not include the stuff asked for previously in the lab manual. It would be a lot easier to have everything needed for points described in the post-lab deliverables rather than having to search the whole lab manual multiple times.

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

I think we just need more assistance to ensure the lab is completed during the assigned lab time and doesn't have to bleed over into time outside of the lab, as it is very difficult to troubleshoot errors without the help of a TA present. I was also having issues with running Vivado on my home computer which took a while to resolve as a result of having to work outside of the lab.