

Lab 1 : Carry Lookahead Adder

CME433-01

Addi Amaya

Caa746

11255790

October 8th, 2021

Contents

[Ripple-Carry Adder 2](#_Toc84533640)

[Testbench structure 3](#_Toc84533641)

[Ripple Carry Adder Verilog File 5](#_Toc84533642)

[Testbench Verilog File 7](#_Toc84533643)

[Quartus Output for Ripple Carry Adder 8](#_Toc84533644)

[Testbench Output for the Adder 10](#_Toc84533645)

[Carry Lookahead Adder 10](#_Toc84533646)

[Testbench Structure 10](#_Toc84533647)

[Carry Lookahead Verilog File 12](#_Toc84533648)

[Quartus Output for Carry Lookahead Adder 15](#_Toc84533649)

[Testbench Output for Carry Lookahead Adder 16](#_Toc84533650)

[Different Bitsize test for Carry Lookahead Adder 17](#_Toc84533651)

[4-bit 18](#_Toc84533652)

[16-bit 19](#_Toc84533653)

[32-bit 20](#_Toc84533654)

[Conclusion 20](#_Toc84533655)

# Ripple-Carry Adder

In this section I am going to outline the software structure I used to test the DUT and the outcome of quartus. All files are in the “RippleCarryAdder” folder within “caa746\_lab2” folder

## Testbench structure

The top-level module is called tbench\_top.sv. This file controls the bit size of the adder, it is where the main clk is toggles, its where all other files are instantiated. Text

Description automatically generated with medium confidence

The interface takes a parameter that changes the bitsize of “a” and “b”, the two inputs for the dut.

A picture containing table

Description automatically generated

The dut\_top is how the dut will integrate with the interface mentioned above

Graphical user interface, text, application, email

Description automatically generated

The testbench is also instantiated in the top but they will be broken down in the next section

## Ripple Carry Adder Verilog File

Since the testbench hierarchy is not expected a latched version of the dut. I needed to make two different version of the dut. One with registers and one without. The reason I had to do this is because I ran into a problem where if I got the Verilog file working on the testbench it would not produce a timing analyzing section. And when I got the timing analyzer working on quartus it broke my testbench. Although one version is latched and one isn’t, the logic and how the values are handled are the exact same.

The rippleCarryadder works as the following: First I made a 1 bit full adder and in a generate wrapper I looped the instance of that full adder depending on the size of the modules parameter (which is controlled by tbench\_top) as mentioned before

A picture containing text

Description automatically generated

Graphical user interface, text

Description automatically generated with medium confidence

Now for the version of the Verilog file that will be included in the testbench which uses the exact same full adder

Text

Description automatically generated

Please note that the only difference between this and the other version is the registered I/O

## Testbench Verilog File

The main method of testing is using random variables with a class randomizer which can be found in lines 8 to 12. The file then go on to check if the output of the dut is equal to adding the two variables (a and b) together and increments a success or failed counter

Text

Description automatically generated

## Quartus Output for Ripple Carry Adder

After running the compatible version of the adder through Quartus and looking under the “Slow 1200mV 0C Model” the FMAX was 107.52Mhz.

Graphical user interface, text, application, Word

Description automatically generated

Graphical user interface, application

Description automatically generated

## Testbench Output for the Adder

After running >1000 cases it was determined that the adder did work. Text

Description automatically generated

# Carry Lookahead Adder

This section will outline how the carry lookahead adder was done and the outcome

## Testbench Structure

The structure for this adder is setup the same as the carry ripple adder. Tbench\_top.sv is the exact same. The only change to interface and dut\_top is that they now incorporate generate (G) and propagate (P) and a I/O

Table

Description automatically generated with low confidence

Text

Description automatically generated

## Carry Lookahead Verilog File

The same problem happened for the carry lookahead as the one I encountered with the ripple adder. Where I could get a working timing analyzer or a working dut. Two different versions were made, one latched and one isn’t. That being said, the exact same logic was used and it will be seen below. Both use the exact same full adder.

A picture containing diagram

Description automatically generated

Below is the version that would allow the timing analyzer in quartus.

Text

Description automatically generated

Text

Description automatically generated

The version that was used for the testbench is as follows

Text

Description automatically generated

As you can see the logic itself is the exact same

## Quartus Output for Carry Lookahead Adder

After running the file through quartus and it compiled, in the “Slow 1200mV 0C Model” it says that the Fmax was 1016.26Mhz. Which admittedly seem high, but it will be shown later that the testbench proves its functionality.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

## Testbench Output for Carry Lookahead Adder

The testbench provided the same number of success cases as the ripple carry adder which makes me very confident that my design does work.

Text

Description automatically generated

## Different Bitsize test for Carry Lookahead Adder

As mentioned before, the size of the carry lookahead adder is completely dependent on parameter variable set up on tbench\_top. That parameter is on line 11. The following photos will demonstrate the testbench in different bit size

### 4-bit

Text

Description automatically generated

### 16-bit

Text

Description automatically generated

### 32-bit

Text

Description automatically generated

# Conclusion

Since I made the testbench prior to the Modelsim. I could not directly import the files to Modelsim. It gave me a working saying that “(vsim-7099) Unable to checkout a verification license for the randomize() feature.” So my attempt for a modelsim waveform will be unsuccessful. If you are familiar with the problem some feedback for how I could fix this in the future would be greatly appreciated. For this lab, I hope that my bash testbench is enough to convince you that my approach does in fact work.

This lab has made it clear that the carry lookahead adder is faster than the ripple carry adder. That being said, it is also clear from the flow summaries that were produced by both that the carry lookahead adder requires more space to accomplish. Both of these statements are consistent with what was taught in class. Another thing I learned was the usefulness of setting up a testbench “environment” as taught in CME435 (I am aware this structure is not taught in CME433) because of its ability to easily integrate different designs into a complex workbench