ECEN 248: Introduction to Digital Design
Department of Electrical and Computer Engineering
Texas A&M University
Lab Report

Laboratory Exercise #9

An Introduction to High-Speed Addition

ECEN-248-509

Joaquin Salas

731000141

11-15-2022

TA: Sri Hari Pada Chandanam Kodi

Objectives:

In the last lab assignment, we saw how the propagation delay through a combinational logic directly impacts our synchronous circuits' speed. We also witnessed the impractical delays associated with ripple-carry addition for larger bit widths. We can surmise the ripple-carry adder is not appropriate for high-speed arithmetic units, like those found in mobile devices or gaming units. As digital circuit designers, it is important that we understand other techniques that exist for binary addition. In this lab assignment, we will introduce a well-known fast-adder circuit for two-operand addition, namely carry-lookahead addition. We will design the necessary components using both dataflow and structural Verilog and simulate them in Xilinx Vivado. This hands-on experience in trying to work with fast-adder circuits will hopefully reinforce the concepts we discuss in the lecture.

Lab Procedures:

The following lab procedures will guide us through the simulation of the circuits discussed in the background section of the lab manual. The first part of this experiment had us create a 4-bit carry-lookahead adder and show the wavelengths. After this was done, we had to test the circuit to make sure it successfully passed the testbench for the 4-bit carry-lookahead adder. After this was done we had to add a 2ns gate delay to the circuit and modify the code to fit the testbench for the 16-bit carry lookahead adder.

Experiment Part 1:

1. Test the 4-bit carry-lookahead adder, we can ensure that the sub-modules required to implement the 16-bit carry-lookahead adder are working properly.

Code: *4-bit carry lookahead*

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

module carry_lookahead_4bit(Cout, S, X, Y, Cin);
    output wire Cout; //C4 for a 4-bit adder
    output wire [3:0] S; // final 4-bit sum vector
    input wire [3:0] X, Y; //the 4-bit addends
    input wire Cin; //input carry
    wire [3:0] G, P;
    wire [4:1] carry;
    generate_propagate_unit unit0(G, P, X, Y);
    carry_lookahead_unit unit1(carry, G, P, Cin);
    summation_unit unit2(S, P, {carry[3:1], Cin});
    assign Cout = carry[4];

endmodule
```

2. With the 4bit carry-lookahead adder working properly, we can examine the propagation delay.

Code: *generate propagate unit*

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

module generate_propagate_unit(G, P, X, Y);
    output wire [3:0] G, P;
    input wire [3:0] X, Y;

    assign #2 G[0] = X[0] & Y[0];
    assign #2 P[0] = X[0] ^ Y[0];
    assign #2 G[1] = X[1] & Y[1];
    assign #2 P[1] = X[1] ^ Y[1];
    assign #2 G[2] = X[2] & Y[2];
    assign #2 P[2] = X[2] ^ Y[2];
    assign #2 G[3] = X[3] & Y[3];
    assign #2 P[3] = X[3] ^ Y[3];

endmodule
```

Code: *4-bit carry lookahead unit*

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

module carry_lookahead_unit(C, G, P, C0);
    output wire [4:1] C;
    input wire [3:0] G, P;
    input wire C0;

    assign #4 C[1] = G[0] | (P[0] & C0);
    assign #4 C[2] = G[1] | (P[1] & C[1]);
    assign #4 C[3] = G[2] | (P[2] & C[2]);
    assign #4 C[4] = G[3] | (P[3] & C[3]);
endmodule
```

Experiment Part 2:

For this part of the experiment, we will create a two-level carry-lookahead unit to add 16-bit numbers together.

1. Design a two-level carry-lookahead adder from the submodules of the 4-bit carry-lookahead adder.

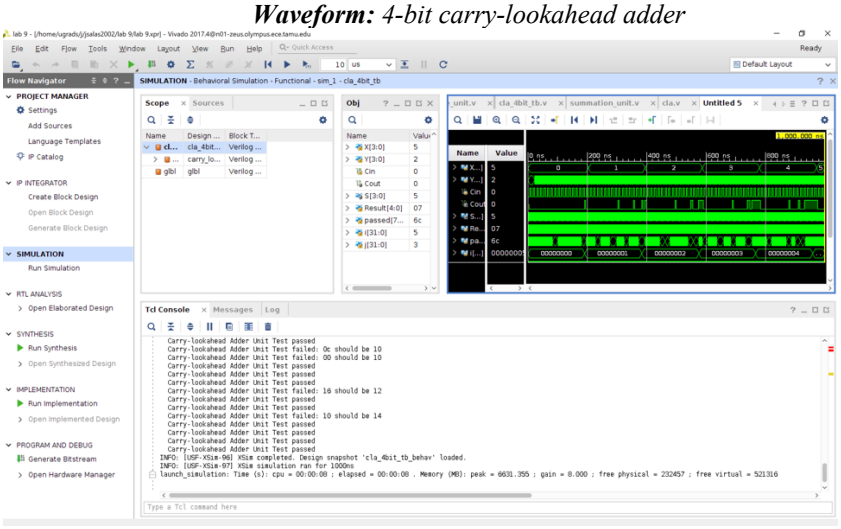
Code: *16-bit carry lookahead*

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

module carry_lookahead_16bit(Cout, S, X, Y, Cin);

    /*ports are wires as we will use structural*/
    output wire Cout; //C_16 for a 16-bit adder
    output wire [15:0] S; //final 16-bit sum vector
    input wire [15:0] X, Y; //the 16-bit addends
    input wire Cin; //input carry!
    /*intermediate nets*/
    wire [16:0] C; //17-bit carry vector
    wire [15:0] P, G; //generate and propagate vectors
    wire [3:0] P_star, G_star; //block gens and props
    /*hook up input and output carry*/
    assign C[0] = Cin;
    /*sub-modules*/
    //instantiate the generate/propagate unit here
    generate_propagate_unit GPU(G, P, X, Y);
    block_carry_lookahead_unit BCLAU0(
        /*since we are being explicit with out ports,  *
        *the order does not matter!                    */
```

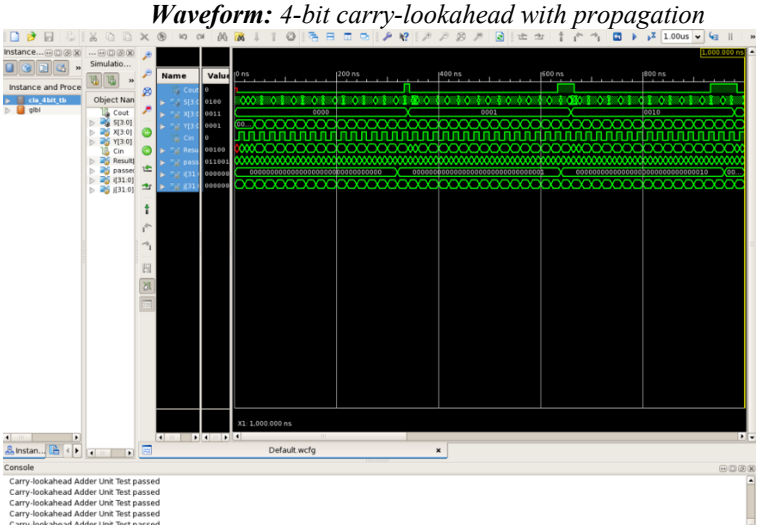
```
        .G_star (G_star[0]),
        .P_star (P_star[0]),
        .C (C[3:1]),
        .G (G[3:0]),
        .P (P[3:0]),
        .C0 (C[0]),
    );
    //instantiate BCLAUs 1-3 here (DONE)
    block_carry_lookahead_unit BCLAU1(
        .G_star (G_star[1]),
        .P_star (P_star[1]),
        .C (C[7:5]),
        .G (G[7:4]),
        .P (P[7:4]),
        .C0 (C[4]),
    );
    block_carry_lookahead_unit BCLAU2(
        .G_star (G_star[2]),
        .P_star (P_star[2]),
        .C (C[11:9]),
        .G (G[11:8]),
        .P (P[11:8]),
        .C0 (C[8]),
    );
    block_carry_lookahead_unit BCLAU3(
        .G_star (G_star[3]),
        .P_star (P_star[3]),
        .C (C[15:13]),
        .G (G[15:12]),
        .P (P[15:12]),
        .C0 (C[12]),
    );
    /*we will use the same form for this one too*/
    carry_lookahead_unit CLAU(
        /* we need to use some fancy             *
        *concatenation syntax to create the carry*
        * vector connected to this module        */
        .C ({C[16], C[12], C[8], C[4]}),
        .G (G_star),
        .P (P_star),
        .C0 (C[0]),
        //two lines are missing (NOT ANYMORE)
    );
    //instantiate the summation unit here
    summation_unit SUM0(S, P, C[15:0]);
    assign Cout = C[16];

endmodule
```

Results:

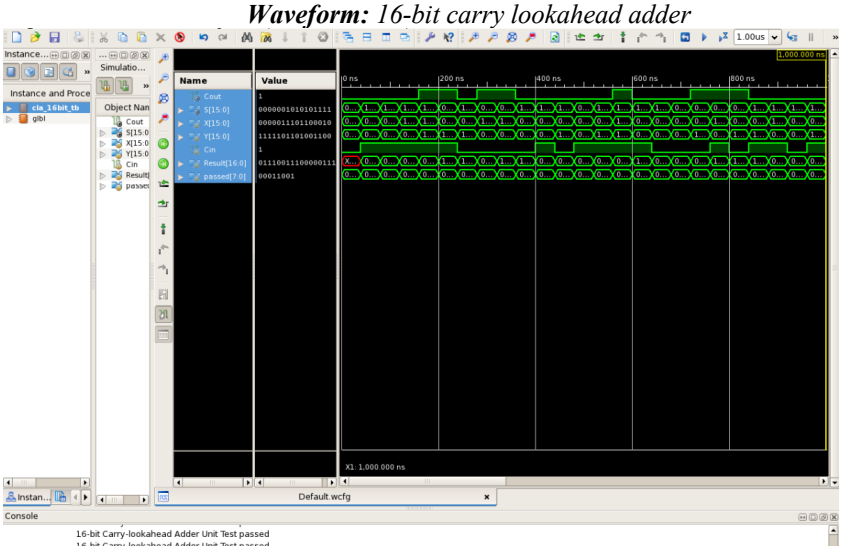
Waveform: *4-bit carry-lookahead adder*



Waveform: *4-bit carry-lookahead with propagation*



Waveform: *16-bit carry lookahead adder*



This lab resulted in me completing all the experiment parts correctly and with the waveforms provided. I was able to learn the basics of high-speed addition in Xilinx Verilog using both structural and dataflow.

Conclusion:

This lab let us gain a better understanding of high-speed arithmetic units. By using both dataflow and structural Verilog Code, we first began by creating a 4-bit carry-lookahead adder and after that a two-level 16-bit carry-lookahead adder. We used this code to observe the effects of the adders when they included gate delays and even when the gate delays weren't included.

Post-lab Deliverables:

1. Include the source code with comments for all modules in lab. You do not have to include test bench code. Code without comments will not be accepted!

The code is shown above and properly labeled.

2. Include the simulation screenshots requested in the above experiments in addition to the corresponding test bench console output.

The simulation screenshots requested are shown above and properly labeled.

3. Answer all questions throughout the lab manual.
• Experiment Part 1, 2.b
• Experiment Part 2, 3.a
• Experiment Part 2, 3.b

4. How does the gate-count of the 16-bit carry-lookahead adder compare to that of a ripple-carry adder of the same size? Give gate counts for both.

The 16-bit carry lookahead adder has 82 gates in the circuit. The 16-bit ripple carry adder has 80 gates. Since the 16-bit ripple carry adder has less gates, the carry-lookahead adder has a higher speed.

5. How does the propagation delay of the 4-bit carry-lookahead adder compare to that of a ripple-carry adder of the same size? Give delay values for both.

The 4-bit carry-lookahead adder has a propagation delay of 3 while the 4-bit ripple carry adder has a propagation delay that is larger at 9.

6. Similarly, how does the propagation delay of the 16-bit carry-lookahead adder compare to that of a ripple-carry adder of the same size? Give delay values for both.

The 16-bit carry lookahead adder has a delay of 8 while the 16-bit ripple carry adder has a delay of 33. Since the carry-lookahead adder has less delay, it makes this adder more effective.

7. Compare the delay growth of a ripple-carry adder versus a carry-lookahead adder. (ie. how does the delay increase as we increase the size of each adder?)

The bigger the size of each adder, the bigger the delay will be. The relationship here is proportional.

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 enjoyed learning the difference between the three types of ways that code is written in Verilog, structural, dataflow, and behavioral, the pre-lab did take me a fair amount of time but I believe it was much worth the work. I did not like how the Zybo-10 board did not work with our computer at the end of the lab.

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?

All parts of the lab manual were very clear, except for the last part on experiment 3 where there was a crucial step that was left out to get the Zybo board to work correctly.

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

I have no suggestions for improving the overall lab assignment.