CDA 4205L Lab #5: Computer Arithmetic Design

University of South Florida

Lab Date: Feb. 12

Report due: One week after the lab section, see Canvas

***This is a group lab**. One submission per group is sufficient.*

Welcome to CDA 4205L Lab #5! The goal of this lab is to help you understand the trade-offs between different implementations of adders and understand how hardware description languages like Verilog can be used to design complex circuits.

Prelab

In Labs 3 and 4, you implemented an assembler, which took instructions, defined by an ISA, and converted them to a machine code representation that works for a given microarchitecture. In future labs, we will look at the microarchitecture level. In this lab, however, we will turn our attention to Register-Transfer Level (RTL). Recall that RTL defines how certain components, or functional units, within a microarchitecture are implemented. For example, if an ISA defines an addition instruction, the microarchitecture must support addition in some way. There are many ways to implement addition, and in this lab, we will look at two of them, while exploring Hardware Description Languages (HDLs).

Verilog and VHDL are two examples of HDLs. These languages are similar to software, but instead of writing instructions for a processor, you are defining how a circuit behaves (RTL), or how gates within a circuit are connected (one level of abstraction below RTL). A synthesis tool can take a high-level description of a circuit (RTL) and generate a netlist, or a set of gates and wires, that implements the design you described in RTL. The synthesis tool will do this however it determines to be the most efficient, within some constraints. If you want greater control over exactly how the circuit is defined, you need to write a lower-level description and define the structure of the circuit or the gates themselves.

In this lab, you will explore two possible implementations of adders. Two such adders are Ripple-Carry Adders and Carry Look-Ahead Adders. These both have pros and cons. You will need to implement both, according to the schematics shown in the provided empty modules, and use the synthesis tool to report the power, performance, and area for each.

Lab

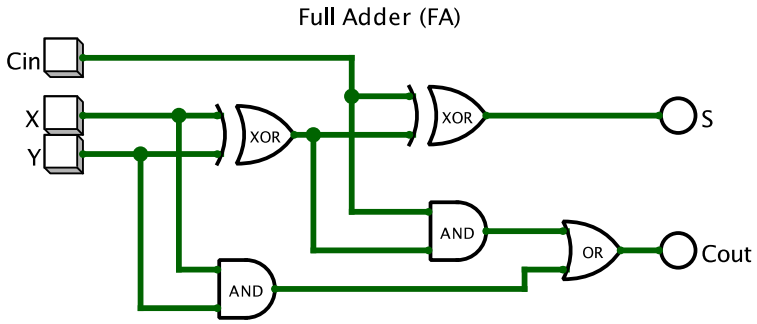
Setup

1. Use the Vivado software pre-installed on the lab computers.
2. Download ‘Lab5_arithmetic_design.zip’ file from Canvas (Labs > Lab5) and open it in Vivado.
3. An example Vivado tutorial: https://digilent.com/reference/programmable-logic/guides/getting-started-with-vivado

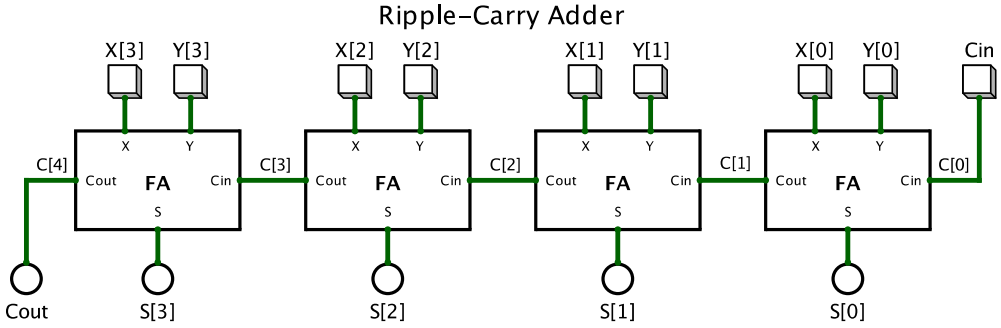
Part 1: Ripple-Carry Adder

4. In “fulladder.v” design a 1 bit full adder module as in the schematic below.

Note: you are NOT allowed to use Verilog’s built-in addition operator (+) for your adder modules.



5. In “ripple_carry_adder.v” design a 4 bit ripple-carry adder module as in the schematic below. Instantiate your “fulladder” module from step 4 to implement your ripple-carry adder.



6. Inside “Simulation Sources” in the “Sources” pane, ensure that testbench.v is assigned as top (bolded). If not, right click it and select “Set as top”.

7. In “testbench.v” make sure the line beginning with “ripple_carry_adder” is uncommented and the line beginning with “carry_look_ahead_adder” is commented out.

8. Make sure to uncomment the “ripple_carry_adder” instance and comment “carry_look_ahead_adder” instance in the testbench. Click “Run Simulation” to simulate your “ripple_carry_adder.v” design. If the simulation value of “score” is 10 at ~1,000ns, then your adder is functioning as expected.

T1: Take a screenshot of your ripple-carry adder simulation waveform.

9. Set “top.v” as Top (should be in bold). Make sure to uncomment the “ripple_carry_adder” instance and comment “carry_look_ahead_adder” instance in “top.v”. Click “Run Synthesis” for “ripple_carry_adder.v” and then “Run Implementation” when the synthesis is complete.

10. Open the implementation report and identify:
    - number of FPGA resources used (LUTs, FFs, BRAMs, DSPs)

      IMPLEMENTATION > Open Implemented Design > *tab*: **Design Runs** > *row*: "Impl_1"

    - dynamic power consumption

      IMPLEMENTATION > Open Implemented Design > *tab*: **Power**

      Summary (#.## W, Margin: N/A): **On-Chip Power** > Dynamic

    - latency of your design

      IMPLEMENTATION > Open Implemented Design > *tab*: **Timing**

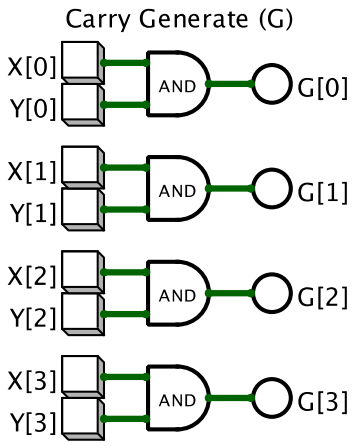
      Unconstrained Paths > NONE to NONE > Setup(#) > *row*: path w/ worst **Total Delay**

> T2: Collect and summarize in a table each of your results metrics: resources utilizations (number of LUTs, FFs, BRAMs, and DSPs), dynamic power consumption, and latency.

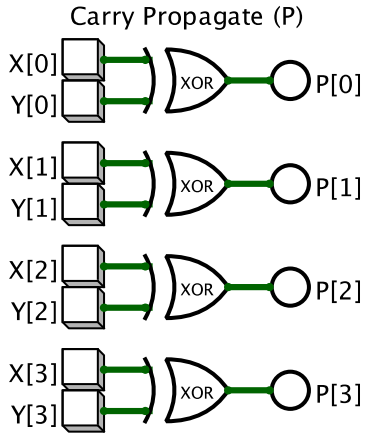
Part 2: Carry Look-Ahead Adder

11. In "carry_generate.v" design a 4 bits carry generate (G) module as in the schematic below.

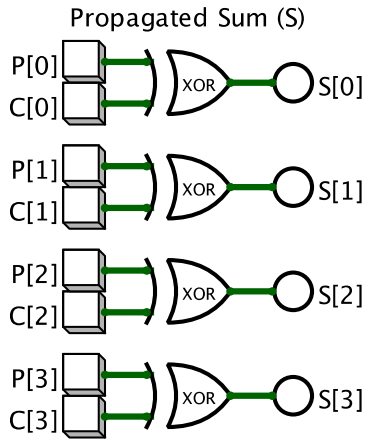
Note: you are NOT allowed to use Verilog's built-in addition operator (+) for your adder modules.



12. In “carry_propagate.v” design a 4 bits carry propagate (P) module as in the schematic below.

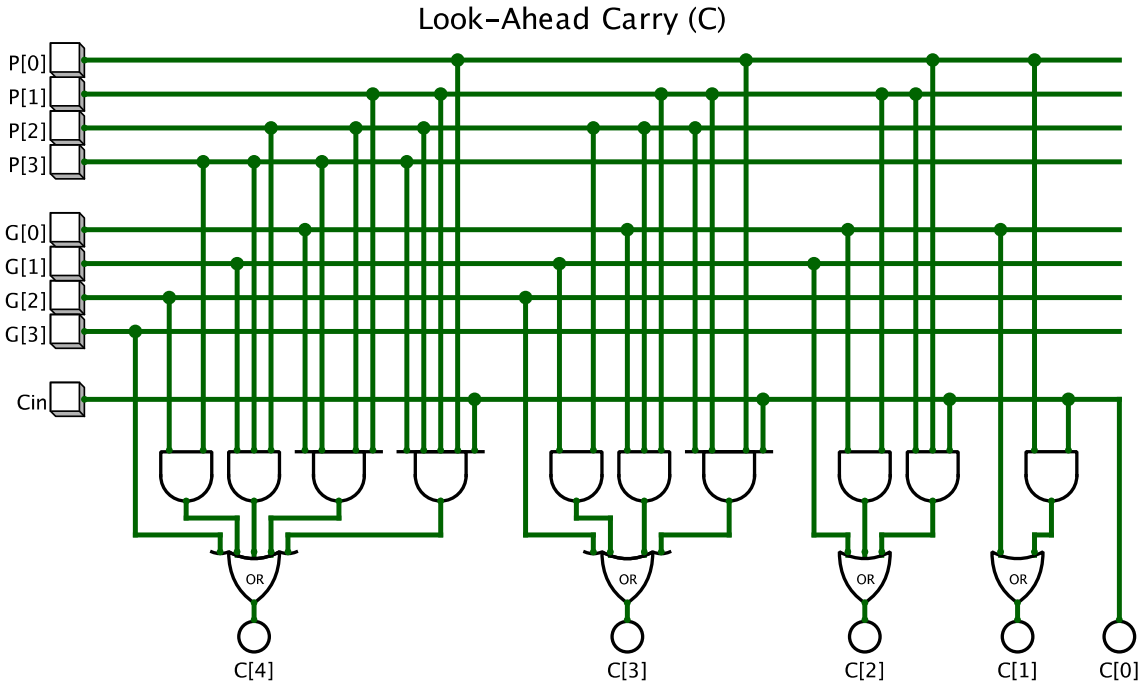


13. In “propagated_sum.v” design a 4 bits propagated sum (S) module as in the schematic below.

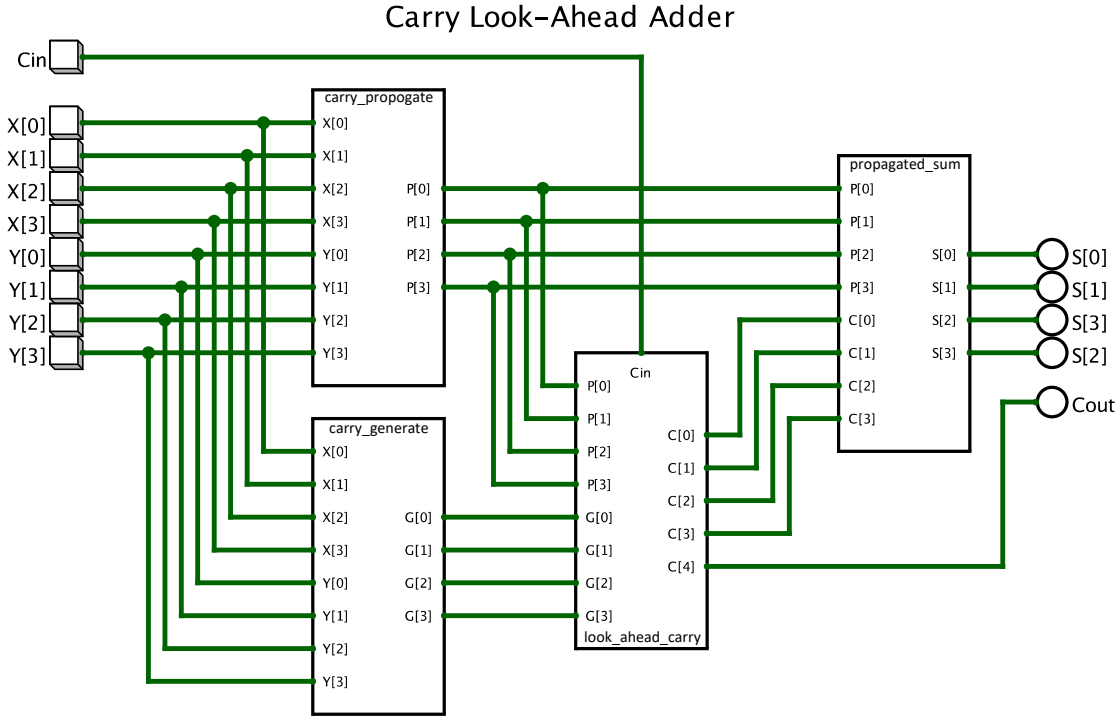


14. In “look_ahead_carry.v” design a 4 bits look-ahead carry (C) module as in the schematic below.

Note: you are NOT allowed to use Verilog’s built-in addition operator (+) for your adder modules.



15. In “carry_look_ahead_adder.v” design a 4 bits carry look-ahead adder module as in the schematic below. Instantiate your “carry_generate”, “carry_propagate”, “propagated_sum”, and “look_ahead_carry” modules using steps 12-15 (respectively) to implement your carry look-ahead adder.



16. Uncomment the “carry_look_ahead_adder” instance and comment “ripple_carry_adder” instance in “testbench.v”. Click “Run Simulation” **with “testbench.v” (set as top)** to simulate your “carry_look_ahead_adder.v” design. Verify their behavior functions as expected.

    T3: Take a screenshot of your carry look-ahead adder simulation waveform.

17. Run “Synthesis” and the “Implementation” for “carry_look_ahead_adder.v”. Set “top.v” as Top (should be in bold). Make sure to uncomment the “carry_look_ahead_adder” instance and comment “ripple_carry_adder” instance in “top.v”. Then, run “Synthesis” and “Implementation”.

18. Open implementation report and identify the number of FPGA resources uses (LUTs, FFs, BRAMs, DSPs), the dynamic power consumption, and the latency of your design as in step 11.

    T4: Collect and summarize in a table each of your results metrics: resources utilizations (number of LUTs, FFs, BRAMs, and DSPs), dynamic power consumption, and latency.

Questions

19. Answer the following questions:

    T5: Using your results from T2 and T4, compute the percent overhead of your Carry Look-Ahead Adder compared to your Ripple-Carry Adder for each metric: resources utilizations (number of LUTs, FFs, BRAMs, DSPs), dynamic power consumption, and latency. (T4-T2/T2)%

    T6: Based on the percent overheads from T5, indicate and explain which adder (Ripple-Carry vs Carry Look-Ahead) is better based on each metric: resources utilizations (number of LUTs, FFs, BRAMs, DSPs), dynamic power consump., and latency. E.g., if Overhead>0, T2 is better.

    T7: Based on your tradeoffs analysis from T6, identify and explain which adder would you recommend for the following devices:
    (a) servers (e.g., Google’s server)
    (b) modern high-performance PC (e.g., gaming computer)
    (c) modern common PC (e.g., your laptop or home desktop)
    (d) modern smartphone (e.g., Apple iPhone 14 Pro or Samsung Galaxy S23 Ultra)
    (e) modern small embedded system (e.g., handheld digital thermometer, single-use insulin pump)

In-class work - Complete **T1 and T2**, show it to the TA for review and submit it on Canvas before the lab session ends. Submit your in-class work as a Word document (.docx) format.

Final Report - Submit your report pdf with answers to all questions and screenshots. Also, submit the final compressed Vivado/Verilog files. Submit one .zip file, per group.