Lab 3: Ripple-Carry and Carry-Lookahead Adders

EEL 4712 – Spring 2019

Objective:

The objective of this lab is to create a generic ripple-carry adder, a generic carry-lookahead adder (CLA), and a hierarchical CLA that supports widths that are a power of 2. You will learn how to use the VHDL generate statement, in addition to how to use configurations to select different architecture possibilities.

Required tools and parts:

Quartus2 software package, ModelSim-Altera Starter Edition, Altera DE10-lite board, Oscilloscope.

Pre-requisite:

You must be "up-to-speed" with Quartus, ModelSim, and the board before coming to lab.

Pre-lab requirements:

Ripple-carry Adder

1. Design a full adder entity in VHDL (fa.vhd). A full adder adds two 1-bit inputs with a carry in, and produces a 1-bit sum and a carry out. Next, design a generic ripple-carry adder using a structural architecture consisting of a chain of full adders (as was discussed in lecture). The ripple-carry adder architecture must be able to support any width. The ripple-carry architecture must be placed in the RIPPLE_CARRY architecture of the adder.vhd file, which is provided on the lab website. **Do not change any part of the adder entity, otherwise the testbench used for grading will not work. The testbench used for grading is provided on the website, so you can verify if your code works.**

Hierarchical CLA

2. Design a 2-bit carry-lookahead adder (CLA) (cla2.vhd) using a behavioral architecture. The entity should have two 2-bit inputs (X and Y), a carry in (Cin), a 2-bit sum (S), a carry out (Cout), a block propagate (BP), and a block generate (BG). See lecture notes and 5.4 in your text book for a description of BP and BG.

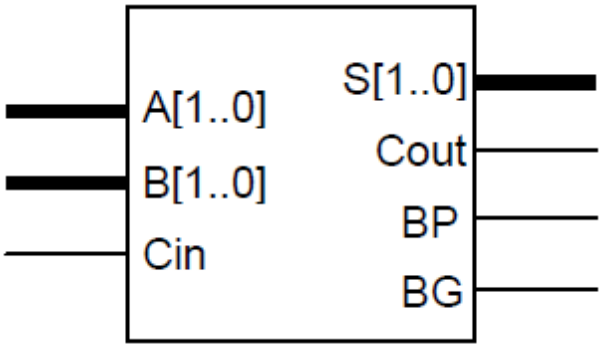

Figure 1. CLA2

3. Design a carry generator (cgen2.vhd) entity that calculates carries (Ci+1 and Ci+2), block generate (BG), and block propagate (BP) for 2 instances of the 2-bit CLA (see next step for clarification). The lectures notes and book will describe the exact purpose of this block, but it will be used to hierarchically define CLAs. Use a behavioral architecture. Note that the block propagate and generate are a function of the P and G of each 2-bit CLA, but it is up to you to figure out the exact logic.

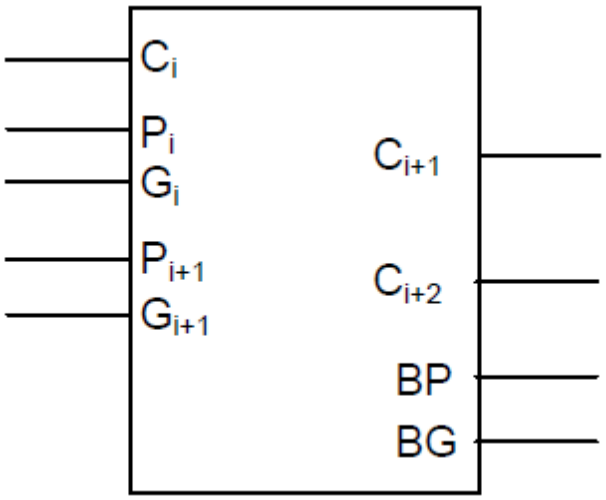

Figure 2. CGEN2

Note that Ci is the carry in to the adder entity that uses CGEN2. Pi and Gi are the propagate and generate outputs from adders using CGEN2 (see next step). In the case that these adders are multiple bits, they correspond to block propagate and generate outputs.

4. Create a 4-bit hierarchical CLA (cla4.vhd) entity with a structural architecture that connects 2 of the 2-bit CLAs with one of the carry generators. It is up to you to determine the exact connections. See lecture notes and section 5.4.

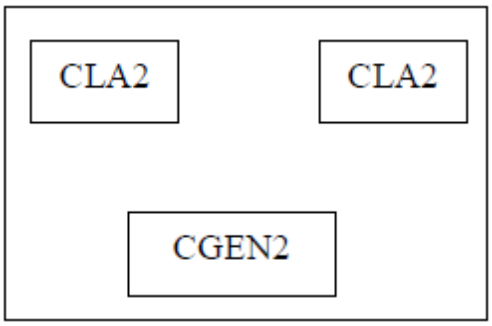

Figure 3. CLA4

5. Fill in the HIERARCHICAL architecture in adder.vhd with an 8-bit hierarchical CLA using a structural description that connects two of the 4-bit CLAs with one carry generator. This will be very similar to the previous step.

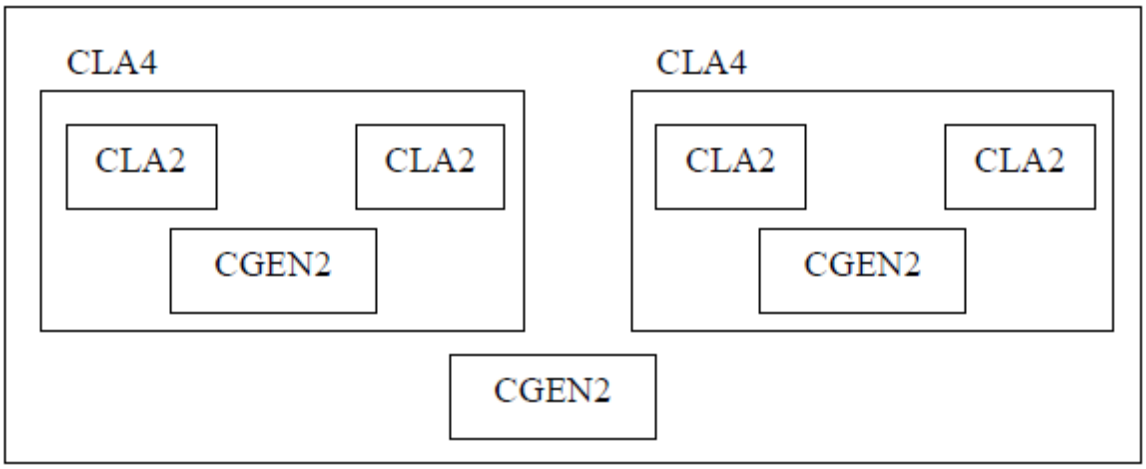

Figure 4. "HIERARCHICAL" architecture of adder entity

6. (Extra credit) Create another adder architecture that extends the hierarchical CLA to the width specified by the generic WIDTH. Hint: the architecture will need to instantiate instances of itself. You only need to support widths that are a power of 2. I don't expect many people to get this working, so email your TA so they know to look for it.

Generic, one-level CLA

7. Fill in the CARRY_LOOKAHEAD architecture in adder.vhd with a generic behavioral description of a carry-lookahead adder (i.e., it must support any possible width). Skeleton code has been provided to get you started. See 5.4 in your text book for clues.
8. Use the provided test bench to test all three architectures. Notice that the test bench assigns a particular architecture to each adder instance using a configuration statement. The same test bench will be used for grading. **When you simulate in ModelSim, make sure you select the configuration tb_config and not the entity adder_tb.**

Turn in on e-learning: All vhd files (fa.vhd, cla2.vhd, cgen2.vhd, cla4.vhd, adder.vhd). The TAs will grade your adder implementations by using a testbench for the adder entity. Therefore, it is critical you do not change the entity declaration or the names of the architectures.

In-lab procedure (prepare steps 1 and 2 ahead of time):

1. Include top_level.vhd with the rest of your VHDL. Note that there is a configuration statement that specifies which architecture the U_ADDER instance uses. The provided code uses the RIPPLE_CARRY architecture. Also, include the 7-segment decoder from the previous lab.
2. Using Quartus, assign pins to each of the top_level.vhd inputs/outputs such that the signals are connected to the appropriate locations on the board. Read top_level.vhd to appropriate determine pin assignments.
3. Download your design to the board, and test it for different inputs and outputs. Demonstrate for the TA at least one example for each possible select.
4. Using the oscilloscope, you are to measure the propagation delays in the following way:
   - Design a test fixture that will allow you to perform the measurement. Hint: Changes in the desired signals are most easily seen using the test equipment. For example, if two fixed numbers are used to drive the "A" and "B" inputs of the adder, a change on Cin can be seen as a change on the Cout pin. In your lab report, describe exactly how you performed the measurements.
   - Use a setting on the oscilloscope which allows an accurate measurement of the propagation delay from Cin to Cout.

- Record the propagation delay from Cin to Cout.
- Make a drawing of the signals (or take a picture using a camera, save a screenshot, etc.) seen on the oscilloscope.
- Performing a timing simulation in ModelSim and record the corresponding propagation delay. Use the provided timing testbench (timing_tb.vhd) and new top-level file (adder_top.vhd), while still using the /UUT label. All you need to do is change the configuration in adder_top.vhd to the architecture that you want to use. Synthesize adder_top in Quartus, add the .vho file to your ModelSim project, select the timing_tb testbench for simulation, add the .sdo file and apply it to region /UUT, and everything should work.

5. Change the configuration statement in top_level.vhd to use the CARRY_LOOKAHEAD architecture, download it to the board, and repeat step 4.
6. Change the configuration statement in top_level.vhd to use the HIERARCHICAL architecture, download it to the board, and repeat step 4.
7. Show your TA the different timings of each architecture. Note that due to FPGA architectures, the propagation delays may be counterintuitive. The results will be explained in future lectures.
8. **Be prepared to answer simple questions or to make simple extensions that your TA may request. There is no need to memorize the different packages. If you have done the pre-lab exercises, these questions should not be difficult.**

Lab report: (In-lab part only)

If you had any problems with portions of the lab that could not be resolved during lab, please discuss them along with possible justifications and solutions. If you had no problems, this report is not necessary.

Turn the lab report in on e-learning, if explanation is needed for partial credit. Make sure to turn it in to the “lab” section and not the “pre-lab” section.