L-Università ta' Malta

DEPARTMENT OF COMMUNICATIONS AND COMPUTER ENGINEERING

Faculty of Information and Communication Technology

CCE1013—Computer Logic 1

Practical 2

Trevor Spiteri
trevor.spiteri@um.edu.mt
http://staff.um.edu.mt/trevor.spiteri

1 November, 2017

Adders

Objective

The objective of this practical is to build, test, and compare a carry-ripple adder and a carry-lookahead adder.

Tasks

1. Create a new project, using the same procedure as Practical 1.
2. Copy the following files to the project directory: *full.vhd*
   full_tb.vhd
   ripple.vhd
   ripple_tb.vhd
   full_lookahead.vhd
   full_lookahead_tb.vhd
   lookahead.vhd
   lookahead_tb.vhd

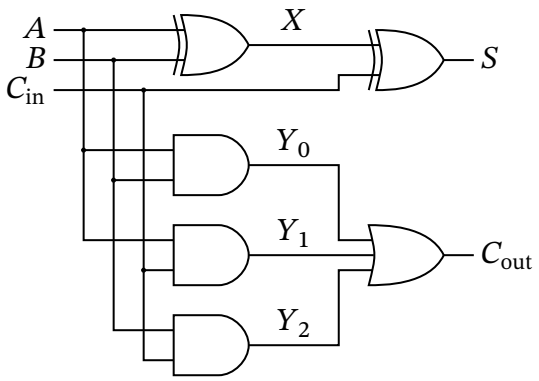

Figure 1: Single-bit full adder.

3. Add the eight files above to the project using the menu item *Project: Add Source...*. For the files without "*_tb*" in their filename, set the association to *All*. For the files with "*_tb*" in their filename, set the association to *Simulation*.

4. The entity *full* defines the single-bit full adder shown in Figure 1. The statements for X and S are provided in the file *full.vhd*, and both include a delay clause. Add statements for Y_0, Y_1, Y_2 and C_{out}, and include similar delay clauses.

5. The VHDL code in *full_tb.vhd* defines a test bench for the full adder. Simulate the full adder using the provided test bench and verify that the outputs are correct. Measure the propagation delays for the two outputs S and C_{out}. The emulated gate delay is 1 ns for each gate, so you can measure the delays in gate delays.

   Note: You should use behavioural simulations throughout this practical, like you did in Practical 1.

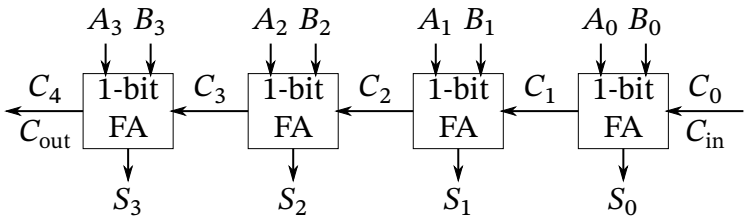

Figure 2: Four-bit carry-ripple adder.

6. The entity *ripple* defines the four-bit ripple-carry adder shown in Figure 2. Look at the VHDL code in *ripple.vhd*. In particular, observe how the four full adders are instantiated in the `gen_full` part.

   Note: This task does not need any action apart from observing the provided code.

7. The test bench *ripple_tb.vhd* will be used to stimulate inputs to test the carry-ripple adder. This time, you do not need to simulate all possible inputs exhaustively.

   For each addition you want to test, you can add code similar to this:

   ```
   A   <= "0000";
   B   <= "0000";
   Cin <= '0';
   wait for 100 ns;
   ```

   This is an addition with $A = 0000_2$, $B = 0000_2$, and $C_{in} = 0$, and is included in the test bench.

   Add more additions to test more cases. You can include as many additions as you like, but you should choose inputs carefully. For example you should have some additions with $C_{in} = 0$ and some with $C_{in} = 1$. You should also have some additions which result in $C_{out} = 0$ and some which result in $C_{out} = 1$. Verify that the sum and carry outputs are correct for all your additions.

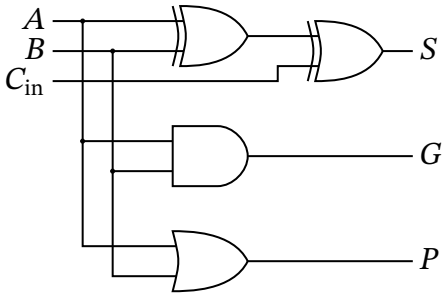

Figure 3: Single-bit full adder for a carry-lookahead adder.

8. In Task 5 you measured the propagation delays of the single-bit full adder. Now measure the propagation delay of C_{out} for the ripple-carry adder by following these steps:

   (a) Add an addition operation to the test bench with operands $A = 0000_2$, $B = 0000_2$, and $C_{\text{in}} = 0$. This should leave $C_0 = C_1 = C_2 = C_3 = C_4 = 0$.

   (b) Exactly after that addition, add an addition operation with operands $A = 0001_2$, $B = 1111_2$, and $C_{\text{in}} = 0$. This addition requires the carry to propagate through all internal carries.

   (c) Simulate the test bench and observe the internal propagation delay of the ripple. To do this, in the simulation window find the instances pane on the left, and select `ripple_tb: uut`. Then drag the object `C[4:0]` to the waveform pane. You may have to restart the simulation and run it again to see the added waveforms.

9. Figure 3 shows a single-bit full adder for a carry-lookahead adder. The file *full_lookahead.vhd* contains an incomplete implementation. Complete the implementation of this single-bit full adder by adding statements for G and P.

10. The file *full_lookahead_tb.vhd* contains a test bench for the lookahead version of the full adder. Measure the propagation delays for the outputs G and P.

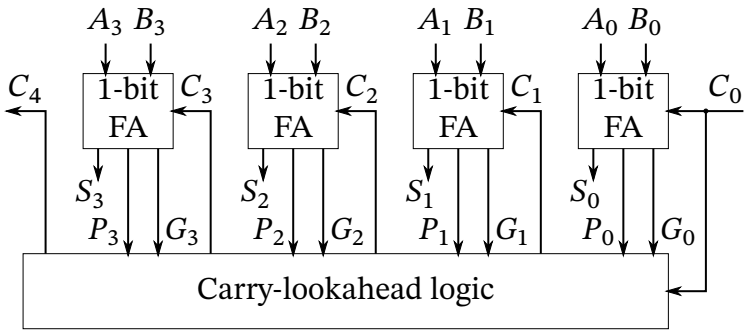

Figure 4: Four-bit carry-lookahead adder.

11. Figure 4 shows a four-bit carry-lookahead adder. The carry-lookahead logic block at the bottom of the figure contains the following logic:

$$C_1 \leftarrow G_0 + C_0P_0$$
$$C_2 \leftarrow G_1 + G_0P_1 + C_0P_0P_1$$
$$C_3 \leftarrow G_2 + G_1P_2 + G_0P_1P_2 + C_0P_0P_1P_2$$
$$C_4 \leftarrow G_3 + G_2P_3 + G_1P_2P_3 + G_0P_1P_2P_3 + C_0P_0P_1P_2P_3$$

    Complete the implementation of the carry-lookahead adder in the file *lookahead.vhd* by writing statements for C_2, C_3 and C_4. The code for C_1 is included as an example.

    Note: Each of the carries in the lookahead logic requires two levels of gate delays, one for the AND gates and one for the OR gates. So after each of the carries you need to include the clause: `after 2 * delay`.

12. Complete the implementation of the test bench for the carry-lookahead adder in the file *lookahead_tb.vhd* by copying the completed stimulus process from *ripple_tb.vhd*. Then verify correct adder operation by checking each output sum.

13. Measure the maximum carry propagation delay for the lookahead adder and compare your result with that of the carry-ripple adder in Task 8.

Report

Your report should describe your work concisely. It should include the VHDL snippets you wrote. You should also show the waveforms that verify that your design works, and include any measurements and comments you might have.