Lab 1: ALU Designs

Submission Due Dates:
Source Code: 2020/09/29 18:30
Report: 2020/10/04 23:59

Objective

Getting familiar with basic Verilog modeling styles.

Action Items

1 Full Adder (20%)

A. lab1_1.v

Write a Verilog module (lab1_1.v) that models a **1-bit full-adder** with and, or, not, xor as basic components. **The value of "b" has to be negated when "sub" is equal to 1, otherwise not.** Test your design with lab1_1_t.v (see the description in the following).

a. IO list:
   - ✓ Inputs: a, b, c, sub
   - ✓ Output: d, e

b. **sub**: the control signal to determine whether b have to be negated **({e, d} = a + b + c when sub == 0; {e, d} = a + ~b + c when sub == 1).**

c. You have to use the following template for your design:

`timescale 1ns/100ps

```
module lab1_1 (a, b, c, sub, d, e);
    input a, b, c;
    input sub;
    output d, e;
    // add your design here
endmodule
```

B. lab1_1_t.v

Complete the testbench lab1_1_t.v to verify your design. Check the template code and the following TODO hints carefully.

a. TODO 1: lease instantiate lab1_1 with correct interconnection where a, b, c and sub are inputs; e, d are outputs.

b. TODO 2: Execute the task function "test" at posedge clock edges to verify whether the circuit is correct with input value of counter {a, b, c, sub}.

c. TODO 3: Increase the counter to try all the 4-bit combinations.

d. TODO 4: Execute the task function "printerror" if the behavior of module "lab1_1" is incorrect. Please verify the circuit with the counter of {a, b, c, sub} and finish the

code of the “if” condition.

e. TODO 5: Set the value of the signal "pass" to 0 and display the value of a, b, c, d, e and sub when error happens. Fill in the correct order of variable in the display function.

2 4-bit Adder (40%)

A. lab1_2.v

Write a Verilog module that models a **4-bit 2’s-complement adders** and test your module using the testbench file lab1_2_t.v. The 4-bit adders must be implemented by using four 1-bit full-adder (i.e. lab1_1) with necessary interconnects. Note that if input port “sub” is equal to 1, the output will be a – b. Otherwise, output will be a + b. Test your design with lab1_2_t.v (see the description in the following).

a. IO list:
   - ✓ Inputs: a, b, sub
   - ✓ Output: d

b. **sub**: The circuit should perform the subtraction when sub is 1 **(d = a – b when sub == 1), otherwise perform the addition (d = a + b when sub == 0).**

c. You have to use the following template for your design:

`timescale 1ns/100ps

```
module lab1_2 (a, b, sub, d);
  input[3:0] a, b;
  input sub;
  output[3:0] d;
  // add your design here
endmodule
```

B. lab1_2_t.v

Complete the testbench lab1_2_t.v to verify your design. Check the template code and the following TODO hints carefully.

a. TODO 1: Please instantiate lab1_2 with correct interconnection where a, b and d are inputs; d is output.

b. TODO 2: Complete the **repeat** block with correct number of times.

c. TODO 3: Execute the task function "test" at positive clock edges to verify whether the circuit is correct with the input value of {a, b , sub}.

d. TODO 4: Increase the counter by 1 at negative clock edges to try all possible 9-bit combinations.

e. TODO 5: Execute the task function "printerror" if the behavior of module "lab1_2" is incorrect. Please verify the circuit with the value of {a, b, sub}. You may refer to the

template code from lab1_1_t.v.

f. TODO 6: Set the value of signal "pass" to 0 and display the value of a, b, d and sub when error happens. Please output the error message as follows:

```
Error:  a = XXXX, b = XXXX, d = XXXX, sub = X
```

where X are the corresponding data bits.

3 4-bits ALU (40%)

A. lab1_3.v

Write a Verilog module that models a **4-bit ALU** and test your module using the testbench file lab1_3_t.v. The 4-bit ALU must reuse the previous 4-bit adders (i.e., lab1_2) with necessary interconnects. The table below is the spec of ALU. You can finish this module with either structural modeling, data flow modeling or behavioral modeling.

a. IO list:
   - ✓ Inputs: a, b, aluctr
   - ✓ Output: d

b. **aluctr**:

If aluctr == 2'b00:
    d = a + b
else if aluctr == 2'b01:
    d = a - b
else if aluctr == 2'b10:
    d = a & b
else
    d = a ^ b

a. You have to use the following template for your design:

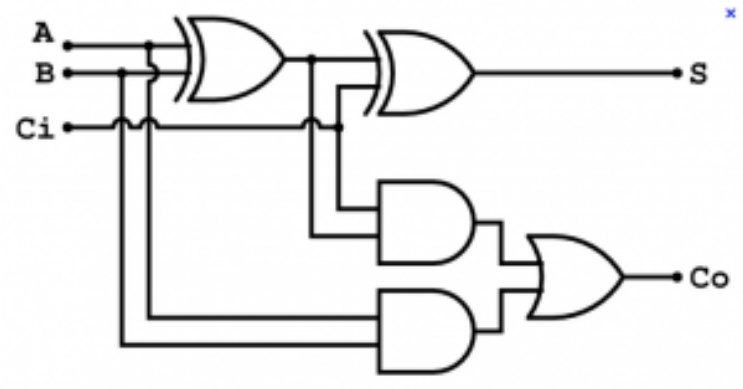
```
`timescale 1ns/100ps

module lab1_3 (a, b, aluctr, d);
  input [3:0] a,b;
  input [1:0] aluctr;
  output reg [3:0] d /*Notice that d can be either reg or
wire. It depends on how you design your module. */
  // add your design here
endmodule
```

Attention

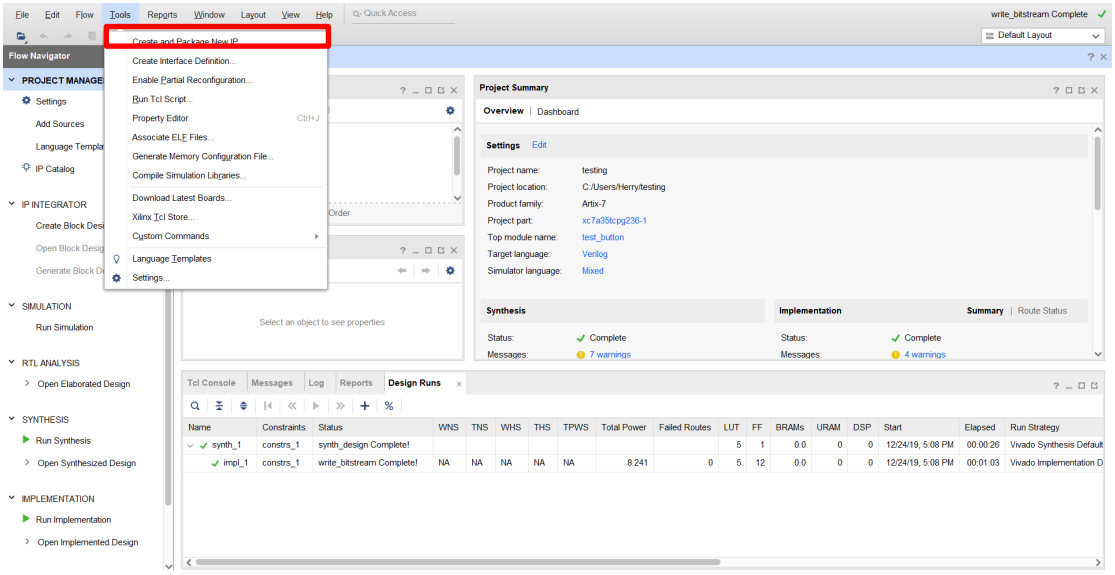
- ✓ DO NOT copy-and-paste code segments from the PDF materials. Occasionally, it will also paste invisible non-ASCII characters and lead to hard-to-debug syntax errors.
- ✓ You should hand in five source files, including **lab1_1.v, lab1_1_t.v, lab1_2.v, lab1_2_t.v and lab1_3.v**. Upload each source file directly! DO NOT hand in a compressed ZIP file!
- ✓ You should also hand in your report as **lab1_report_StudentID.pdf** (i.e., lab1_report_108456789.pdf).
- ✓ You should be able to answer questions of this lab from TA during the demo.
- ✓ You may also add a **$monitor** in your testbench to show all the information of your inputs and outputs during the simulation.

- ✓ Hint: here is one possible gate-level implementation of a full-Adder for your reference:

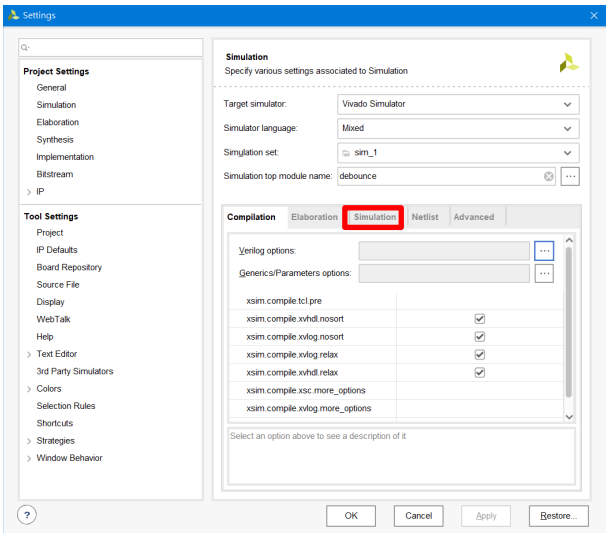


- ✓ When you are simulating lab1_3 and lab1_2, you have to change your runtime to 10000ns (or a large enough period) in “Simulation Settings” before you run the simulation. See the guide below.

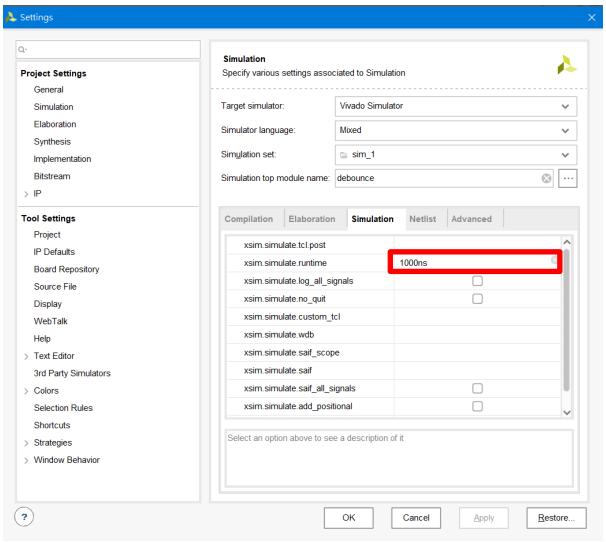
1. Click Tools in the top of Vivado and select Settings.



2. Select Simulation.



3. Change the runtime to 10000ns or larger value to finish the simulation.



4. The result should look like this.