CS203: Lab 3

Model 16 bit ripple carry and 16 bit CLA adder in Verilog

Objective: Understand delays and array-based design in Verilog
Language: Verilog
Points: 20
Release date: 23 September 2020
Deadline: 6 October 2020

Detailed problem statement:

1. Delays of gates (it could be behavior or structural)

   ```
   Delay of 3 input xor gate : 4 units
   Delay of 2 input xor gate: 3 units
   Delay of 2 input and/or gate: 1 units
   Delay of 3 or 4 input and/or gate: 2 units
   Delay of 5 to 8 input and/or gate: 3 units
   Delay of not/invertor : No delay
   ```

   Use these delays in all the modules, wherever necessary. It may be noted that delay statements should be incorporated in basic gates or data-flow statements involving logic.

2. Module Full Adder: Design full_adder module include delay models.

   Module definition:

   ```
   module full_adder(sum, cout, a, b, cin);
   ```

   You may use data-flow or structural model for full_adder module.

3. Module add16_rc: Model 16 bit ripple carry adder using structural design. Instantiate full_adder instances in this module.

   Module definition:

   ```
   module add16_rc(sum16, cout, a16, b16, cin)

   Sum16 is 16 bit wide output, cout single bit output.
   a16, b16 are 16 bit input and cin is single bit input.
   ```

4. Module add4_cla: Model 4 bit carry lookahead adder.

   ```
   module add4_cla(sum4, cout, a4, b4, cin)

   Sum4, a4 and b4 are 4 bit wide. cin and cout is single bit wide
   ```

```
//add4_cla may have two additional outputs gp and gg (group
propagate and group generate)
```

Add4_cla should be modeled in structural way using the following two components

```
module full_adder_pg(sum, cout, p, g, a, b, cin);
//sum, cout, p and g are single bit output
a, b and cin are single bit input
//you may avoid creating this module, if p and g are generated
as output of "full_adder" module. In that case "full_adder" can
be reused here

module  carry_generator(couts, p, g, cin)
// couts is 4 bit carry output, p and g are 4 bit input and cin
is single bit input
```

5. module 16 bit carry look ahead adder: add16_cla

   Use add4_cla and carry_generator module to model 16 bit carry look ahead adder.

   ```
   module add16_cla(sum16, cout, a16, b16, cin)

   Sum16, a16, and b16 are 16 bit wide. cin and cout is single bit
   wide
   ```

   Read through page 112 and 113 of the textbook for design clarification.

6. Create testbench.v for 16 bit carry look ahead adder and 16 bit ripple carry adder. Give different possible stimulus values, print them on screen using $display. Dump the vcd file using $dumpvars and observe the output in waveform viewer. Observe the latency of 16 bit adder.

Logistics:

- Module definitions should be the same as specified in the problem statement.
- full_adder, add16_rc, add4_cla, add16_cla, carry_generator, all modules should be in a single file named **`lab3_add.v`**
- **`testbench.v`** will be a separate file containing testbench.
- Testbench need to test add16_rc and adder16_cla. You need to include both functional as well timing tests. Timing test ensures that output is available at expected delays or not.
- Zip both the files into "Lab3_<entry_number>.zip" and submit on moodle. Use only zip for compression, do not use rar, tgz, bgz or any other compression.
    - It may be a good idea to include your VCD file in submission for reference.

- For your benefit, it may be helpful to test full_adder, adder4_cla modules before you jump onto final modules. However, testbench of “full_adder” and “adder4_cla” should not be submitted.

Evaluation:

- Lab will be evaluated automatically using scripts, so it is important to follow the above instructions and naming conventions.
- Use only the latest version of iverilog. Using any other verilog compiler can lead to compatibility issues.
- Course policy of late submission and plagiarism, as given on course webpage would apply.

The theory regarding the assignment will be covered in the lecture on 25th September.