ECE4700J Computer Architecture

Summer 2022

Lab #3 A Pipelined Integer Square Root Module

Due: 11:59pm Jun. 10th, 2022 (Beijing Time)

Logistics

- This lab is an individual exercise.
- All the design code should be in SystemVerilog.
- This lab must be checked off by a TA before end of lab on Friday, Jun. 10th, 2022 (Beijing Time). It is highly recommended you complete the lab before the following week's lab release.
- All code and reports (if available) MUST be submitted to the assignment of Canvas.
- Internet usage is allowed and encouraged.
- No late submission is allowed for this lab.

Overview

In this lab, you will study how to design some complex hardware module design.

- Study how user-defined types are used in SystemVerilog.
- Study how pipelining is achieved by Verilog.
- Be able to implement more complex FSMs.

Assignments

I. Designing the ISR Module (Mandatory)

a) Part A: Pipelined Multiplier

We have provided you with a skeleton of pipelined multiplier (incomplete), found in pipe_mult.sv and mult_stage.sv. The multiplier does multiplication in stages, somewhat like you would have learned to carry out multiplication in elementary school. It multiplies the first 8 bits of the multiplier with the whole multiplicand in one clock cycle, then the next 8 bits of the multiplier against a shifted multiplicand, and so on to get 8 partial products. Summing those gives us the desired multiplication. This means that each multiplication will take 8 clock cycles. Each partial product is created by a separate multiplier stage, which can be found in the mult_stage.sv file.

Your first assignment will be to finish the 8-stage pipelined multiplier design and

pass the post-synthesis timing simulation. We have provided you a testbench in mult_test.sv, carefully read it and ensure you understand it before using.

Your second assignment is to modify the pipelined multiplier we've provided to work as both a 4 stage multiplier and as a 2 stage multiplier. You can either do this by copying the files and having separate 2, 4 and 8 stage multipliers or by figuring out a combination of preprocessor macros or parameters that set pipeline depth. Once you have the other two multipliers, you will need to pass the post-synthesis timing simulation. (Hint: The default clock period 500ns in our testbench may not be enough for your 2-stage or 4-stage pipelined multiplier. You may need to keep increasing your clock cycle length to pass the tests.)

b) Part B: Integer Square Root

Now, you will need to create a module that uses the multiplier that we supplied, the 8 stage multiplier. You will be writing a module to compute the integer square root of a 64-bit number. It will generate a 32-bit number that is the largest integer that is not larger than the square root of the number provided. For example, the integer square root of 24 is 4.

The module declaration is as follows:

```
module ISR(
    input                   reset,
    input           [63:0]  value,
    input                   clock,
    output logic    [31:0]  result,
    output logic            done
);
```

It should operate as follows:

- If ***reset*** is asserted during a rising clock edge (synchronous reset), the ***value*** signal is to be stored.
- If ***reset*** is asserted part way through a computation, the result of that computation is discarded and a new ***value*** is latched into the module.
- When the module has finished computing the answer, the output is placed on the ***result*** line and ***done*** line is raised on the same cycle.
- It must not take more than 600 clock cycles to compute a result (from the last clock that ***reset*** is asserted to the first clock that ***done*** is asserted.)

We do not suggest that you pipeline this module. You will likely need to perform something like a binary search to find the result a simple algorithm is as follows:

Algorithm 1 Integer Square Root

```
1: procedure ISR(value)
2:     for i ← 31 to 0 do
3:         proposed_solution[i]←1
4:         if proposed_solution^2 > value then
5:             proposed_solution[i]←0
6:         end if
7:     end for
8: end procedure
```

Note that loops do not have a direct hardware equivalent. What hardware design technique lets us implement a procedure like this?

In addition to writing this module, you will need to write a testbench (test_ISR.sv) for it. This testbench should probably test specific corner cases, random testing and the short loops. Your testbench should print either **@@@Passed** or **@@@Failed**.

Once you have the module written and tested, synthesize it and do post-synthesis timing simulation. This will probably take several minutes at least.

II. Submission

After you are confident in your solution, make sure you have the following files in your directory:

1. **ISR.sv**
2. **test_ISR.sv**
3. **mult_stage.sv**
4. **pipe_mult.sv**

References:

1. Umich EECS470 WN 2021 Project2

Acknowledgement:

- Jon Beaumont (University of Michigan)

Deliverables:

- Live Demo of Mandatory Assignment:
  - Show TA with the waveform of your post-synthesis timing simulation of your part B.
- Individual Deliverables:
  - Submit design files of the Mandatory Assignment: ISR.sv, test_ISR.sv, mult_stage.sv, pipe_mult.sv (contrasted into a zip/tar file) via canvas assignment.

Grading Policy

Live Demo – 70%
Canvas Files Submission and Correctness – 30%