ECE 566A Modern System-on-Chip Design, Spring 2017
Lab 2: Pipelined Unsigned Integer Multiplier
Due: Mar 6, 2:30 pm

1. Overview

In Lab2, you need to read source code of several different multipliers and then you need to design your own pipelined multiplier using Verilog HDL based on these open source codes. After that, you will go through the ASIC design flow you did in Lab1 on all these multipliers. So, in this lab, you need to know how to use ASIC design flow tools including Synopsys VCS, Design Compiler, and Cadence Encounter. In addition, you will compare the performance of different multipliers in your report.

2. Introduction

2.1 Fixed-latency iterative unsigned integer multiplier[1]

```
1  def imul_fixed_algo( a, b ):
2
3    result = Bits( 64, 0 )
4    for i in range(32):
5      if b[0] == 1:
6      result += a
7      a = a << 1
8      b = b >> 1
9
10   return result
```

Figure 2.1: Fixed-Latency Iterative Multiplication Algorithm – Assumes a and b are 32-bit Bits objects.

The fixed-latency iterative multiplier will always take approximately 32 cycles. Figure 2.1 illustrates the fixed-latency iterative multiplication algorithm using “pseudocode” which is really executable Python code. Each iteration we check the least significant bit of the *b* operand; if this bit is zero then we shift the *b* operand to right and the *a* operand to the left, but if this bit is one then we add a to the result before shifting. Each iteration is essentially calculating a partial product for the multiplication. Note that we say “approximately 32 cycles” because there may be a few extra cycles of overhead in handling the *operands, result, rdy (*signal to indicate if the result is ready, check Verilog source code), although a more optimized design can remove this overhead.

You will find the Verilog version code of Fixed-latency iterative unsigned integer multiplier in the folder *fixedlatency* on our classroom Github of Lab2_src. Detailed explanation of the folder content is in section 3.

2.2 One-Cycle unsigned integer multiplier

While the iterative multipliers will likely require minimal area, they also require many cycles to calculate each result. As a point of comparison, we will also consider a simple single-cycle integer multiplier. We use registered inputs for both the operands and the valid bit. If the response interface is not ready, then we stall the multiplier by disabling the register enable signals and combinational propagating the response ready signal to the request ready signal. For the actual *multiplier* we simply use the * operator; we will rely on the ASIC toolflow to choose the most appropriate multiplication hardware.

You can find the Verilog version code of One-Cycle unsigned integer multiplier in the folder *onecycle* on our classroom Github of Lab2_src. Detailed explanation of the folder content is in section 3.

2.3 Pipelined unsigned integer multiplier

The area and cycles of pipelined multiplier are between the one of Fixed-Latency and One-Cycle multiplier. In the Pipelined multiplier design, the calculation of fixed-latency multiplier is divided into several steps with equal cycles. The output of last step is connected to the input of next step and all steps work at the same time. So the delay of one partial calculation is approximately 32/steps cycles.

You will need to design your own Verilog code of pipelined unsigned integer multiplier for this lab. And you can find the template for this design in the folder *pipelined* on our classroom Github of Lab2_src. Detailed explanation of the folder content is in section 3.

3. How to get started

You are expected to accept the lab assignment in the link by click the button “Accept this assignment”.

https://classroom.github.com/assignment-invitations/ab3e4e519cd8f20c57ef85fe3291ac75

If you are the first time to use github, you should generate your own public key and add it to your github account, or you will have permission denied when you git clone the repository form our github classroom. When you add the ssh public key, you can follow the link below:

https://help.github.com/articles/connecting-to-github-with-ssh/

And then create a folder in your own linux server account and git clone your lab assignment repository. There are the command lines you may refer below:

% mkdir name_folder (i.e., mkdir lab2)
% cd name_folder
% git clone git@github.com:wustl-ese566/lab2-your_user_name.git (i.e., git clone git@github.com:wustl-ese566/lab2-YunfeiGu.git)
% cd lab2-your_user_name/ (i.e., cd lab2-YunfeiGu/)

This repository lab2-your_user_name/ contains three folders:

- fixedlatency: a folder contains the source code of fixed-latency multiplier;
- onecycle: a folder contains the source code of one-cycle multiplier;
- pipelined: a folder contains the *template code* of pipelined multiplier;

And the README.md file:

- README.md: a file that records the instruction how to use this open source core.

For each multiplier, the inputs and outputs are:

- rst: Reset;

- clk: Clock input;
- req_msg_a, req_msg_b(a, b in Figure 2.1): Operands, 32 bits;
- rsq_val: Request. Used by the host to indicate the start of the calculating process.
- rsq_rdy: Request ready. Used by the multiplier to indicate that it can accept new input. If *rsq_rdy* is de-asserted, it means the multiplier will ignore the *req_msg* and *req_val*
- resp_msg: Result, 64 bits
- resp_val: Response request. Used by the multiplier to indicate the validation of the result.
- resp_rdy: Ready for response. Used by the host to indicate that it can accept new result. If *resp_rdy* is de-asserted, it means the host will ignore the *resp_msg* and *resp_val*;

The direction of each signal is defined in the provided Verilog code.

(Note: signals of rsq_val, rsq_rdy, resp_val, resp_rdy are mainly used in the pipelined multiplier design, since the head and the tail of pipeline work separately. In order to be compatible, we implement all these signals in all three designs).

Please be sure to source the class setup script using the following command before compiling your source code: module add ese461.

4. Tasks

4.1 Pipelined unsigned integer multiplier design

You should implement a pipelined unsigned integer multiplier with the same interface used in the fixed-latency design (Code template is in the folder *pipelined* on github). You can assume that the number of stages will be a power of two between 1 and 32 inclusive.

The *resp_val* will indicate the validity of the result. When the *req_rdy* is asserted but the *req_val* not, the status need to be transmitted through the pipeline to the *resp_val* so that the host know the validity of the result.

Each partial step of the pipeline will perform two shifts, an addition, and then mux the result of the addition conditionally based on the least-significant bit of the *req_msg_b* input.

(Note: If you are interest, you can put FIFOs at the input and output of the multipiler. So, if the *resp_rdy* is de-asserted, the multiplier still could queue the calculation result in the FIFO until the FIFO is full[1]. The same is the FIFO at the input end. We will not use *resp_rdy* in our default design and assume it is always asserted.)

4.2 Compare the performance of different multiplier

In this section, you need to go through all the steps you did in lab1:

- Use Synopsys VCS to compile the Verilog source code of different multipliers;
- Use Design Compiler to do the synthesis of different multipliers;
- Use Cadence Encounter to do the place and route of different multipliers.

Then in your report, compare the power, area, timing and other facts that you think are important of these different multipliers and briefly explain the result. It is better to visualize your experimental results (tables and/or plots).

5. Submission

Please submit your lab assignment on Github. You are expected to submit your report, *pipelined/multiplier.v* file, and lab2_dc.tcl file for each *multiplier*. If you modify the test benches or create new ones, submit them too.

To submit your job, execute the following command:
(Note: the first two commands just need to be done once for the entire semester.)

```
% git config --global user.name "your_user_name"
% git config --global user.email "your_email_for_github"
% cd directory_of_your_lab_assignment/lab2-your_user_name/
% git add pipelined/multiplier.v
% git add fixedlatency/lab2_dc.tcl
% git add onecycle/lab2_dc.tcl
% git add pipelined/lab2_dc.tcl
% git add Lab2-report.pdf
% git add fixedlatency/multiplier_tb.v          (optional)
% git add onecycle/multiplier_tb.v              (optional)
% git add pipelined/multiplier_tb.v             (optional)
% git commit –m "your commits"
% git push -u origin master
```

You also should submit anything else you think may help us understand your code and result.

Please do not submit files like compiling result (*simv*) or simulation data (*.vpd*).

6. Acknowledgement

[1] http://www.csl.cornell.edu/courses/ece5745/handouts/ece5745-lab1-imul.pdf