JOINT INSTITUTE
交大密西根学院

ECE4700J Computer Architecture

Summer 2022

Lab #5 CAM and FIFO Design and Chisel Integer Square Root Module

Due: 11:59pm Jul. 1st, 2022 (Beijing Time)

Logistics

- This lab is an individual exercise.
- All the design code should be in SystemVerilog / Chisel.
- If you implement the optional part, it must be checked off by TA before end of lab on Friday, July 1st, 2022 (Beijing Time) during the Friday OH time.
- 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 CAM and FIFO modules using Verilog, and how to design the ISR module with Chisel.

- Study how to design CAMs (Content Addressable Memory).
- Study how to design FIFO buffers.
- (Optional) Study how to install and run Chisel.
- (Optional) Study how to design and test a module in Chisel.
- (Optional) Study the differences between Verilog and Chisel.
- (Optional) Be able to synthesize Chisel to Verilog.

Assignments

I. Designing a simplified CAM (Mandatory)

In this lab you will be designing a simplified, 8-entry Content Addressable Memory (CAM). A CAM has the property that, when queried with a value, it will search through all of its contents in parallel, returning the index at which that data exists (if it does). Our simplified CAM has the following module header:

```
module CAM #(parameter SIZE=8) (

    input clock, reset,
    input enable,

    input COMMAND command,

    input [31:0] data,

    input [$clog2(SIZE)-1:0] write_idx,

    output logic [$clog2(SIZE)-1:0] read_idx,
    output logic hit
    );
```

CAM should have the following functionality:

- The design will maintain a set of “SIZE” (default of 8) 32-bit memory elements, which should only be updated on the positive edge of “clock”.
- When “reset” is high on the rising clock edge, all entries in memory should be invalidated.
- If “enable” is high and “command” is set to “WRITE” (see sys_defs.svh), CAM should store the value of “data” to memory address “write_idx” and validate it. If the index is out of bounds (larger than “SIZE-1”), CAM should not modify its memory.
- If “enable” is high and “command” is “READ”, CAM should write the lowest location of valid memory whose value equals ”data” to “read_idx”. If the memory does not contain “data”, “hit” should be set low. Otherwise, it should be high.

In this assignment, you just need to set the SIZE as 8. It is up to you how to implement this functionality. We recommend using a “for” loop inside of an “always” block. Remember that in a procedural block, all assignments happen simultaneously. If multiple assignments are made to the same variable, only the “later” one (the one listed furthest down in the block, or the furthest iteration of a “for” loop) takes effect.

Your design should pass the testbench after synthesis (post-synthesis timing simulation) as well; however, don't worry about the clock period. Just make a design that works. Remember to set the "sys_defs.svh" as "Global Include".

II. Thinking about Designing FIFO (Mandatory)

In this assignment, we want you to think about how to use Verilog to design some data structures. FIFO (First-in First-out) queue buffer is a very common type of hardware data structure. Imagine you have a FIFO interface like this:

```
module FIFO #(parameter SIZE=8) (

    input clock, reset,

    //Front end
    input in_ready,
    input [31:0] in_data,
    output full,

    //Back end
    input out_ready,
    output [31:0] out_data,
    output empty
    );
```

Assume your FIFO buffer has a fixed size 8. Whenever the front data is ready (in_ready=1) and your FIFO is not full, the input data (in_data) will be enqueued into your FIFO queue's tail at the next clock posedge. Whenever the output receiver is ready (out_ready) and your FIFO buffer is not empty, your FIFO need to output the queue-head data (out_data) and dequeue this node at the next clock posedge.

Besides, since the input giver should be aware of whether the data it gives can be accepted by the FIFO (== the buffer is not full), otherwise it should stall the data it gives. Also the output receiver should be aware of whether this cycle it can receive some data. So "full" and "empty" should all be outputs of the FIFO.

Please think about how you can use Verilog to design a module like this. Create a file called **fifo.txt**, and answer the following questions in brief sentences in that file:

1.A good way to implement this is to use two logic type registers as "head" and "tail" pointers. What's the bit width of these two variables?

2. How can you implement this module by using “head” “tail” pointers? What can be their initial values? How can you use these two registers to know whether the FIFO is empty or not empty, full or not full?

3. Must we use sequential logic to do the update of these two pointers? Why?

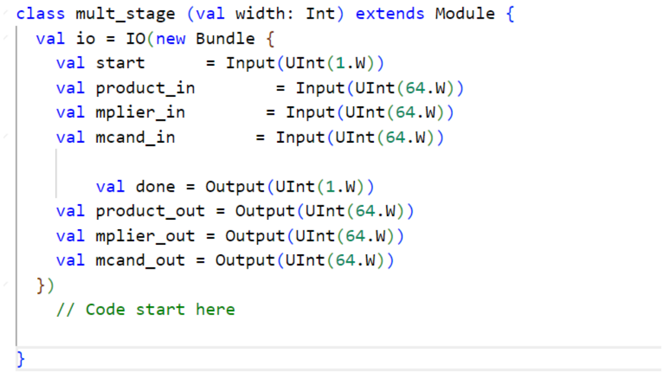
III. Designing the ISR Module in Chisel (Optional)

In this optional assignment, you will be designing an auto-test batch script for your pipeline. You will be given an up to 20% bonus to the total score of Lab 5 if you are able to correctly implement the optional assignment. This will be counted towards your final grade.

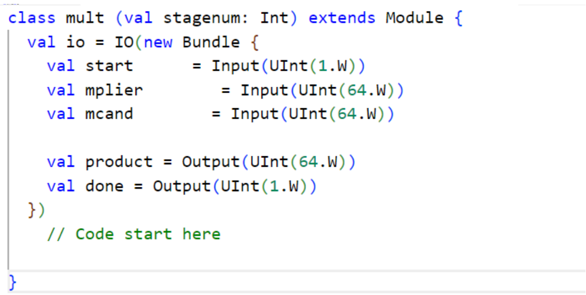
In this assignment, you are asked to reimplement the modules of Lab3 in Chisel. The installation and sample codes are provided by UC Berkeley (ucb-bar/chisel-tutorial: chisel tutorial exercises and answers (github.com)). *Digital Design with Chisel* is a very good book to learn Chisel (http://www.imm.dtu.dk/~masca/chisel-book.pdf). For any questions related to Chisel, feel free to post on Piazza with folder “chisel” selected.

a) Part A: Pipelined Multiplier

In this part, you are asked to design pipelined multipliers including two modules pipe_mult and mult_stage. The pipelined multiplier should have three stage number 8, 4, 2. You can either do this by copying the codes and having separate 2, 4 and 8 stage multiplier modules or by figuring out a combination of preprocessor macros or parameters that set pipeline depth. Below shows an example module declaration of pipe_mult and mult_stage.



```
class mult_stage (val width: Int) extends Module {
  val io = IO(new Bundle {
    val start       = Input(UInt(1.W))
    val product_in        = Input(UInt(64.W))
    val mplier_in        = Input(UInt(64.W))
    val mcand_in        = Input(UInt(64.W))

        val done = Output(UInt(1.W))
    val product_out = Output(UInt(64.W))
    val mplier_out = Output(UInt(64.W))
    val mcand_out = Output(UInt(64.W))
  })
    // Code start here

}
```

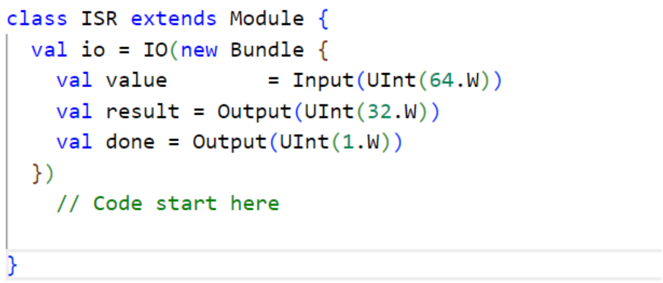


```
class mult (val stagenum: Int) extends Module {
  val io = IO(new Bundle {
    val start       = Input(UInt(1.W))
    val mplier        = Input(UInt(64.W))
    val mcand        = Input(UInt(64.W))

    val product = Output(UInt(64.W))
    val done = Output(UInt(1.W))
  })
    // Code start here

}
```

b) Part B: Integer Square Root

Now, you will need to create a module ISR 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. The algorithm is the same as Lab3, a binary search.

An example module declaration of ISR is as follows:



```
class ISR extends Module {
  val io = IO(new Bundle {
    val value         = Input(UInt(64.W))
    val result = Output(UInt(32.W))
    val done = Output(UInt(1.W))
  })
    // Code start here

}
```

After completing the source files, you need to test your ISR. Feel free to use any Chisel test methods like the peekpoke tester, ScalaTest, and printf as long as it's easy for you to identify a potential issue. However, synthesizing the Chisel module to Verilog first and testing with Verilog testbench will not grant you full score. Be aware of the case coverage of your test code.

If your design pass all testcases, synthesize it to Verilog and submit it together with all your Scala codes. You are not required to simulate or implement the synthesized Verilog module in Vivado but it's highly recommended to have a try.

IV. Submission

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

1. **part1/cam.sv**
2. **part2/fifo.txt**
3. **(Optional) optional/ISR.scala**
   - mult_stage
   - pipe_mult
   - ISR
   - ISR_test
4. **(Optional) optional/ISR.v (emitted by Chisel)**

If you implement the optional part, it must be checked off by TA before end of lab on Friday, Jul. 1st, 2022 (Beijing Time) during the Friday OH time.

References:

1. Umich EECS470 WN 2021 Lab6

Acknowledgement:

- Jon Beaumont (University of Michigan)

Deliverables:

- Submission of Mandatory Assignment:
  - Submit cam.sv and fifo.txt
- Live Demo of Optional Assignment:
  - If you implement the optional part, it must be checked off by TA before end of lab on Friday, Jul. 1st, 2022 (Beijing Time) during the Friday OH time.

Grading Policy

Canvas Files Submission and Correctness – 100%