Simple Computation

Ben Heard

March 2, 2021

Abstract

In this project you will be designing and implementing a system that accepts four 4-bit numbers, performs the computation $(A + B) - (C + D)$, and provides the result as output.

1 Introduction

In this project you will be designing and implementing (in Verilog) a system that accept four 4-bit numbers on its input, computes the value for $(A + B) - (C + D)$, and provides the result as output.

1.1 Top Level Interfaces

At the top level the following signals shall be connected.

- **rst_n**: INPUT An active low synchronous reset
- **clock**: INPUT A freerunning clock
- **d_in**: INPUT A 4-bit unsigned input
- **op**: INPUT A 2-bit addend indication
- **capture**: INPUT A single bit capture indication
- **result**: OUTPUT A 5-bit unsigned result as output
- **valid**: OUTPUT A single valid indication

1.2 Output Interface

The output interface is the simpler of the two. It is the same **valid** interface that we've used many times in in-class designs. Once your system has calculated the result it will assert the **valid** signal along with the result indicating that the result may be captured by another system.

1.3 Input Interface

The input interface is a little more complex. In much the same way that we have done in class, the input will be accepted at a rising clock edge when capture is high. However, there is the additional selection input, **op**, that provides indication as to which operand is being provided. Specifically, whether A, B, C, or D is being provided on the in put during that **capture**.

To note, this means that A, B, C, and D will all come in on the same **d_in** input, just at different times. Which is currently being provided is indicated by **op**. Another note is that this allows the values for A, B, C, and D to be capture in any order that they are provided. It will be a combination of your controller and datapath to ensure that the correct values are used as the correct operand.

This input interface looks similar to ones that we have been using in class; with the added complexity of the **op** signal. When **capture** is asserted your subsystem is to capture the value on **d_in** as the value indicated by **op**. Once all four values are captured, then your system is to compute the required function and provide the result.

1.4 Requirements

- Naming Requirements

    1. The Verilog module shall be called **proj001**. This is just the name of the module, the file(s) may have any name you choose.
    2. The project shall have an active low synchronous rest input called **rst_n** that, when asserted, will reset the entire system to it's initial state.
    3. The project shall hae a free-running clock input called **clock**.
    4. The project shall accept a 4-bit input called **d_in** that will be an unsigned quantity.
    5. The project shall accept a 2-bit input called **op** that indicates the operand that is being provided.

6. The project shall accept a single bit input called **capture** that provides indication that the value on **d_in** should be captured.
7. The project shall provide a 5-bit output called **result** that represents the result of the computed function.
8. The project shall provide a single bit output called **valid** that provides indication as to when the **result** may be captured.

- Data Requirements

1. The output, **result** shall be an unsigned quantity.

- Timing Requirements

1. An active low synchronous reset will be provided to your system at the beginning of operation to initialize the system to a known state. This will be asserted to the **rst_n** port of your system.
2. To facilitate grading, the **result** and **valid** must be asserted within 10 clock cycles of the last operand being provided to your system. Any time within the 10 clock cycles will be sufficient.
3. The **valid** signal shall assert for a single clock cycle.

- Implementation Requirements

1. The implementation shall be any combination of structural or dataflow, with the exception of the adders and subtractors. The base component of the adder/subtractor may be dataflow but the wider adder/subtractors shall be structural combinations of the base components.
2. The top level module shall instance a datapath and a controller, structurally, connecting them to the inputs, outputs and each other as necessary.
3. The core D Flop Flop shall be the one provided by the instructor.

- Git Submission Requirements

1. The completed project shall be pushed to you class Git repository at the NCSU GitHub. The SHA for the submission shall be provided in the Moodle project assignment.
2. The directory containing your submission shall be the projects project_001 folder created for you in your repository.
3. There are no naming requirements for the files that are submitted. Only that the files be text files with a .v extension.

2 Notes

2.1 Input Details

- While the inputs may be provided in any order, each operand will only be provided once. In other words, you will only be provided a single value for each of A, B, C, and D between expected computation results.
- No new values for A, B, C, or D will be provided until valid is seen asserted on your output.
- The testbenches will ensure that the result of $(A + B) - (C + D)$ will always be positive or zero; there is no need to check for whether a result goes negative.