University of Connecticut
ECE 3401: Digital Systems Design
Spring 2024

Programming Assignment 3: Parallel Factorial Design

Due April 19, 2024 (Friday) @ 11:59 PM on HuskyCT

1 Introduction

In this assignment we will revisit the factorial function discussed in Lecture 16. To compute the factorial of a non-negative integer n, $n!$, you simply perform the following multiplication:

$$n! = \begin{cases} 1 & n = 0 \\ \prod_{i=1}^{n} i & n > 0 \end{cases}$$

You will design hardware that accelerates computing the factorial by performing multiplications in parallel. There are three main parts in this design:

1. The state machine design, which is responsible for keeping track of the state of the computation and modifying control signals.
2. The synchronous datapath design, which is responsible for computing multiplications.
3. The top level design, which incorporates the design modules and serves as I/O to the test benches.

The state machine module is contained in `controller.vhd`, the datapath is contained in `datapath.vhd`, and the top level module is `pa3.vhd`. There are two test benches: `fact_tb1.vhd` and `fact_tb2.vhd`. More details will be provided later on each module. Goals of this assignment are to implement the design using VHDL:

- Implement the datapath design.
- Implement a Moore state machine to provide control to the datapath.
- Evaluate design correctness and performance using test benches.

2 Design Overview

2.1 Summary

In our design, you will implement an accelerated version of the factorial function. **Your goal is to design a controller + datapath to compute the factorial of an arbitrary integer $n \geq 0$ in as few cycles as possible**. Details are in the following sections.

2.2 Structure of PA Modules

The overall design is split into three components i) The PA3 top module, ii) The state machine controller module, and iii) The synchronous datapath controller module:

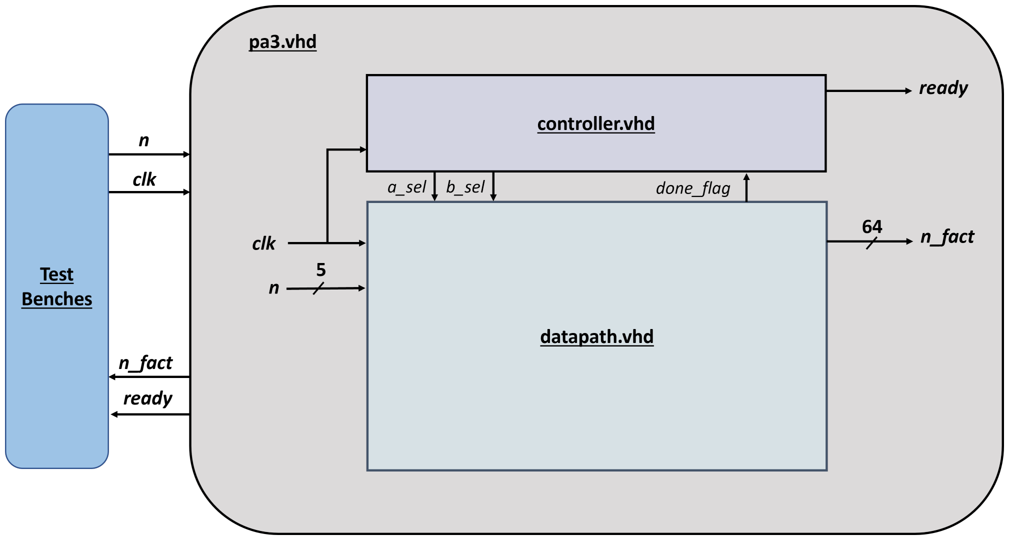

Figure 2.1: Overall structure of the assignment

In the top module `pa3.vhd`, the design performs the port maps to tie the I/O between the `controller.vhd` and `datapath.vhd` module, as well as the test bench interaction. In `controller.vhd`, a state machine keeps track of current phase in the calculation, setting the `ready` signal once the factorial calculation is finished. The controller also generates outputs `a_sel` and `b_sel`, feeding them to the datapath module. In `datapath.vhd`, the outputs generated from the controller are used as inputs to the datapath, and the output, `n_fact` is computed. The output `done_flag` is also generated and fed back to the controller. The clock signal is an input to both the datapath and controller modules, synchronizing them. For all arithmetic ALU operators (such as subtractors, multipliers, comparitors, etc.) you should just use the VHDL operators directly, as with PA1 and PA2. Remember that operations that modify the output width must get resized accordingly if stored in a signal of varying width.

2.3 PA3 Top Module

In the top module `pa3.vhd`, a 5-bit unsigned value `n` serves as the input to the factorial calculation. The `clk` signal is also an input to the top module. `n` is an input to just the datapath, while `clk` gets used by both the controller and the datapath. For all registers in the designs, they are rising-edge triggered.

The two outputs, `n_fact` and `ready`, are generated by the datapath and controller respectively. `n_fact` is a 64-bit unsigned value which contains the mathematical value of $n!$ at the end of computation, and `ready` is a single bit that indicates when the computation is finished.

The port maps for the controller and datapath are already provided, so **the pa3.vhd module should not be modified**.

2.4 Datapath

In `datapath.vhd`, the logic for producing the output must be implemented using the signals from `controller.vhd`. The 5-bit signal `n` and `clk` are inputs from the `pa3.vhd` module. The 2-bit MUX select signals `a_sel` and `b_sel` select which inputs are fed into the `a` and `b` register, respectively. The `a` register is used to maintain the factorial computation, while `b` is used as the decrementor to generate multiplicands for the datapath. You are responsible for the overall design of the datapath. You are given the following guidelines to create your design:

- The datapath may contain 2-port multipliers, 2-port subtractors, registers, 2:1 MUXs, and 4:1 MUXs.

- The special register `a` is 64-bits in length, and its input MUX controlled by `a_sel` from the controller must be capable of choosing between (i) resetting to 1, (ii) multiplying with a partial product, and (iii) maintaining its current value.
- The special register `b` is 5-bits in length, and its input MUX controlled by `b_sel` from the controller must be capable of choosing between (i) loading in the current value of `n`, (ii) decrimenting, and (iii) maintaining its current value.
- A multiplier output must be registered before being used as an input to another multiplier (see Figure 2.2).
- You may assume that all input values of n are small enough such that overflow of any registers will not be an issue.
- You must ensure that the register generating the factorial is mapped to the output `n_fact`. You must also ensure that the logic generating `done_flag` is correctly implemented so that the controller knows when to transition to the final state.

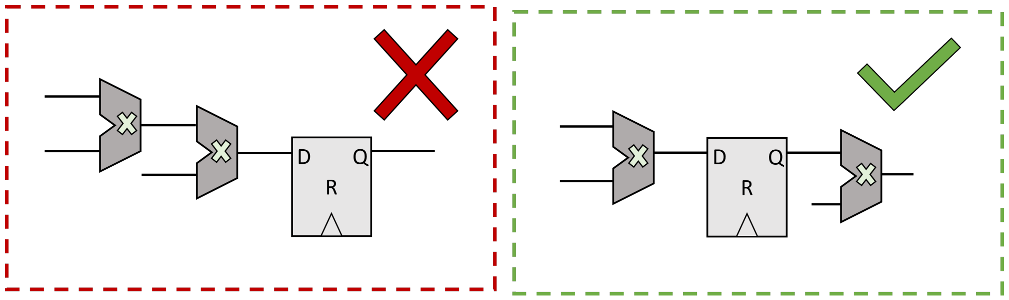

Figure 2.2: Example of legal and illegal dataflow.

2.5 Controller

The controller module, `controller.vhd`, is responsible for (i) tracking the progress of the calculation through a state machine and (ii) generating the appropriate output signals `a_sel` and `b_sel` for `datapath.vhd` and (iii) asserting the `ready` output when appropriate. You are responsible for designing the state machine and producing the control logic used in the datapath. Guidelines are given below:

- The state machine must have four states. A suggestion of states are given below:
  1. START: A default starting state to initialize necessary signals.
  2. MUL1: A state that performs one step the multiplication computation.
  3. MUL2: A state that performs another step in the multiplication computation, while also checking the `done_flag` to decide which state to transition to (either MUL1 if there are more computation cycles needed, or DONE if done).
  4. DONE: A state for when the FSM completes computation which loops back to START.

  You may wish to rename states to fit your design.
- The `a_sel` to `b_sel` signals serve as MUX select wires for the datapath. They are each 2-bit in length, and can be used to control one 4:1 MUX in the datapath each. These outputs must be a function of the `state` signal (Moore outputs).
- The `ready` output, which is '1' only when the datapath has finished computing `n_fact`, must also be a function of the `state` signal (a Moore output).

- As there is no start / reset signal, your final state should immediately transition back to your start state. The `state` signal must be initialized to the last state, so that the the simulation begins in your starting state.
- The `fact_tb1.vhd` will (i) set a particular value of n to the `n` input and (ii) wait for the ready signal to transition to a '1' (indicating that `n_fact` should have the correct value). It then repeats these steps with a different `n` to test a few n values. You should keep this is mind for the state machine design, as after `ready` is asserted for one pulse, the FSM should immediately begin its setup computation with the next `n`.
- Keep in mind that mathematically, $0! = 1$. Your logic must account for this special case.

3 Test Bench

There are two total test benches: `fact_tb1.vhd` and `fact_tb2.vhd`. The first test bunch runs through a suite of tests why waiting for your `ready` signal to get set, then verifies that your output `n_fact` matches the expected value $n!$ for the current input `n`. If it does not match, a message is printed to the TCL console that an incorrect output is produced, but the test bench will not halt. It is your responsibility to ensure your design produces the correct values. No modification to the first test bench is required.

For the second test bench, `fact_tb2.vhd`, you are to design a test bench that allows to to determine the performance of your multiplier. Come up with a mathematical equation as a function of n that describes how many cycles the design takes to compute $n!$ (from the cycle that the `n` value gets updated to the cycle that `ready` goes high and `n_fact` is available).

4 Deliverables

Please submit a single PDF containing the following:

1. Your code of the following:
   - `controller.vhd`
   - `datapath.vhd`
   - `fact_tb2.vhd`
2. A 1 - 2 paragraph description of your strategy for your design. Explain why your design works well for accelerating the factorial computation, and tradeoffs you considered.
3. An analysis of your design's performance. Give the equation $f(n)$ that models your design's performance (in clock cycles), and explain why it is correct. Describe what tests you decided to implement in `fact_tb2.vhd` to measure the performance and validate your equation. Also if applicable, list optimizations you made to improve the performance.
4. Your diagrams of (i) the datapath you designed and (ii) the controller FSM you designed.
5. The screenshots listed below. For the waveform screenshots, please **clearly** show `cur_cycle`, `n`, `n_fact`, `clk`, `ready`, `a`*, `b`*, `a_sel`*, `b_sel`*, and any other registers created in the datapath*.

   *Some of these may need to be added to the waveform from the controller / datapath modules.

   - The output waveform from `prefix_tb1.vhd` from 0ns until the test bench completes. This is the cycle that the test bench completes the last test of `n` = 5, and loops back to testing `n` = 3.
   - The output waveform of your `prefix_tb2.vhd` from 0ns until the test bench completes.