University of Connecticut
ECE 3401: Digital Systems Design
Spring 2024

Programming Assignment 1:
Design of Parallel ALUs with Feedback

Due February 16, 2024 (Friday) @ 11:59 PM on HuskyCT

---

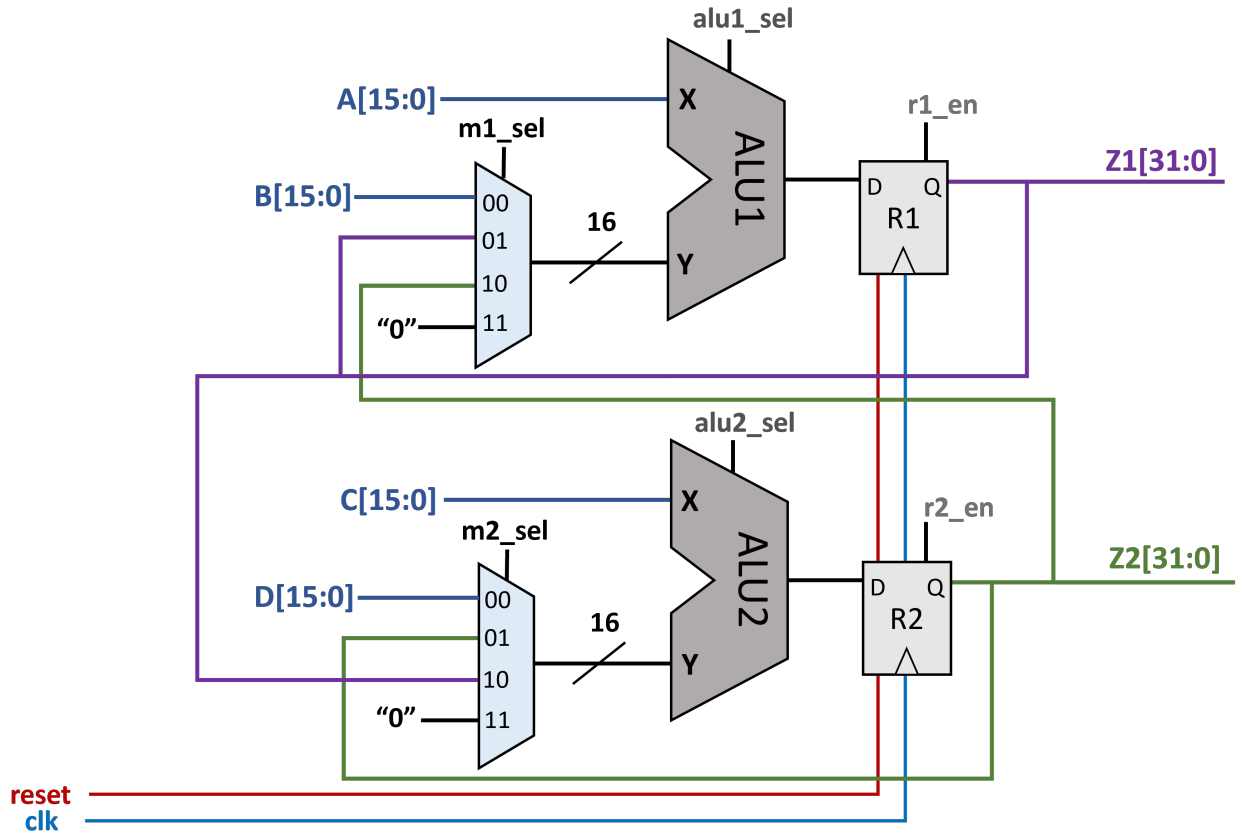
1 Introduction

This programming assignment will require you to design a system with two ALUs that can perform arithmetic operations on 16-bit operands in a parallel design. Operands either come from external 16-bit inputs or feedback from one of two registers. This assignment has the following functionality:

- The following ALU operations:
  - 16-bit addition
  - 16-bit subtraction
  - 16-bit multiplication
  - 16-bit shift-right-logical
- Logic to control arithmetic operations, based on a select input.
- Logic to control ALU and 32-bit register inputs using multiplexers.
- Verification of the design using a test bench.

2 Description

The figure below shows the architecture of the system:



Overall Design of PA1

The ALU operation of the inputs and values stored in the assigned register bank are performed continuously. `A[15:0]`, `B[15:0]`, `C[15:0]`, and `D[15:0]` serve as external input signals to the system, and are 16-bits wide. The arithmetic logical units ALU1 and ALU2 have operations controlled by the `alu1_sel` and `alu2_sel` signals, which are described in table 1 below:

Table 1: ALU Operation Selection

alu1_sel / alu2_sel	OPERATION
00	addition (X + Y)
01	subtraction (Y - X)
10	multiplication (X * Y)
11	shift-right-logical (Y » 1)

The `r1_en` and `r2_en` signals control the 32-bit registers R1 and R2. When '1,' the register

should propagate the value from D → Q on the **rising edge** of `clk`. Otherwise, the register should instead hold the old value of Q.

The register reset signal `reset` *asynchronously* clears the values of R1 and R2. When brought to a '1,' the values held by R1 and R2 are set to 0 regardless of the `clk` signal. When the signal is '0,' the registers operate normally. The reset signal should remain '0,' except when you need to clear the outputs of the registers. Initially, the register output values should be '0' by asserting the reset signal at the beginning of simulation.

There are also two 4:1 MUX units with control signal `m1_sel` and `m2_sel`. `m1_sel` selects input `Y` to ALU1, while `m2_sel` selects input `Y` to ALU2. For both MUXes, the select signals `m1_sel` and `m2_sel` operate as follows:

- `sel` = "00" : selects `C[15:0]` for ALU1 or `D[15:0]` for ALU2 as the input.
- `sel` = "01" : selects self ALU feedback as the input.
- `sel` = "10" : selects alternate ALU feedback as the input.
- `sel` = "11" : selects a zero vector as the input.

Each ALU is expected to perform an unsigned integer add, subtract, multiply, and shift-right-logical operation. We do not require structural VHDL for these operations. You should use behavioral operators and the appropriate IEEE libraries for your design. One design option is to use constrained unsigned to represent the input operand (i.e. `A: in unsigned (15 downto 0);` for the `A` operand) and use `IEEE.numeric_bit.all` library.

The register output values, `Z1` and `Z2`, are 32-bit buses. As mentioned above, the values of the multiplexer select signals determine which outputs to use for the ALU1 and ALU2 alternate inputs. The lower 16-bit output of each register bank is connected to the inputs of the multiplexers (i.e. any ALU input signals coming from register outputs need to be truncated appropriately to match the 16-bit widths of each ALU input). Keep in mind we do not consider overflow in the design of our digital system, so these conditions may be ignored.

You will use separate VHDL modules for i) the register bank (`dff.vhd`) ii) the ALU (`alu.vhd`) and iii) the overall PA1 module (`pa1.vhd`). You are given top-level modules with entity instantiations, and you are expected to write the architecture for each module. In the pa1 module specifically (`pa1.vhd`), you are given the architecture declaration, signal definitions, and entity instantiations, and you are required to fill in the logic for each signal and the port maps for each of these entity instantiations. Note, entity instantiation is an alternative way to instantiate modules. You can read more about entity instantiation and how it works compared to component instantiation here. It is recommended that you use explicit port connection definitions in your port map definitions, which was used exclusively in the examples from the hyperlink provided above. You will be graded on the design of these modules and their functionality.

3 Test Bench

A good design develops a test bench. Complete the following simulations using test benches `testbench1.vhd` and `testbench2.vhd`. The first test bench is provided and described in section 3.1. The second test bench in section 3.2 gives you a high level function that you need to implement. You should use `testbench1.vhd` as a reference to help you when writing your own code in `testbench2.vhd`.

3.1 Calculating Parallel Summation with a Test Bench

In the provided example test bench (`testbench1.vhd`) we use the digital design to find the value of two different summations in parallel. The summation formula used is as follows:

$$f(n) = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \tag{1}$$

When you are finished writing each module (`dff.vhd`, `alu.vhd`, and `pa1.vhd`), simulate the design using `testbench1.vhd`. To select the current simulation, right click "testbench1" and select "set as top" (it should be bold). In the test bench, the ALU1 computes $f(3)$ and ALU2 computes $f(5)$. After testing and confirming the result, feel free to change the values of the defined constants (lines 30, 31, and 32 in `testbench1.vhd`) to different values. The description of the testbench is below:

- Reset the system so R1 and R2 are zero. Set the inputs `A` $\leftarrow 3$, `B` $\leftarrow 1$, `C` $\leftarrow 5$, and `D` $\leftarrow 1$. Set up the MUXes and ALUs so that R1 $\leftarrow$ `A` + `B` and R2 $\leftarrow$ `C` + `D`. Wait one clock cycle (R1 $= 3 + 1$ and R2 $= 5 + 1$).
- Set up R1 $\leftarrow$ `A` $\cdot$ `Z1` and R2 $\leftarrow$ `C` $\cdot$ `Z2`. Wait one clock cycle (R1 $= 3 \cdot 4$ and R2 $= 5 \cdot 6$).
- Set up R1 $\leftarrow$ `Z1` $>>$ 1 and R2 $\leftarrow$ Z2 $>>$ 1. Wait one clock cycle (R1 $= \frac{(3)(4)}{2}$, R2 $= \frac{(5)(6)}{2}$). Done!

Keep in mind that the R1 output is mapped to `Z1` and R2 is mapped to `Z2`, so these are the signals you should observe in your simulation. For in-depth details on how this is implemented using the designed system, read through `testbench1.vhd`. Comments are provided for you to follow the test bench flow as a tutorial.

3.2 Parallel Formula Calculation

For a certain calculation, you need to calculate the following equation:

$$g(x, y, z) = (2x + y)z^2 - x \tag{2}$$

You now want to use your digital design to efficiently compute g using the skeleton test bench `testbench2.vhd`. Assuming the input mapping of `A` $\leftarrow a$, `B` $\leftarrow b$, `C` $\leftarrow c$, and `D` $\leftarrow d$, you are to compute the output of $g(a, b, c)$ and $g(c, d, a)$ in the minimum number of cycles

by taking advantage of the parallel hardware. Note the following restriction: **you may not change the input mappings**. In other words, your design should work correctly no matter if a, b, c or d is numerically changed.

Using equation 2 and your completed digital design, compute the outputs of both $g(a, b, c)$ and $g(c, d, a)$ using the pre-defined values of $a = 5$, $b = 6$, $c = 7$, and $d = 8$ using the skeleton test bench `testbench2.vhd` to write your code. Although you are only responsible for showing the predefined simulation, you are free to try using different input values of a, b, c or d by modifying the constant assignments starting at line 33. Remember that you cannot modify `A`, `B`, `C`, or `D` after the assignments, but all other control signals (MUX select, ALU select, register enable, and reset) may be changed at any point, unrestricted (Hint: sketch out the data flow required for the calculation first, and write out what values are accumulated at each clock cycle. Then, translate it to test bench control signals to get the desired behavior). The desired outputs can appear on either Z1 or Z2, but both $g(a, b, c)$ and $g(c, d, a)$ outputs must be shown at some point together. It will help to first read through the `testbench1.vhd` code and comments to understand the PA1 test bench flow.

To earn the maximum number of points for the testbench you must (i) ensure it is functionally correct and (ii) finish both calculations in 5 cycles total. You will still get partial credit if more cycles are needed and the correct outputs are computed.

4 Deliverables

Please submit the following report **saved as a single PDF**:

1. Your code for each module and your second testbench. You can copy and paste the code into a Word document; make sure to clearly label each code block.
   - `alu.vhd`
   - `dff.vhd`
   - `pa1.vhd`
   - `testbench2.vhd`
2. Submit screenshots of the following:
   - Your output waveform of `testbench1.vhd` using. Show the waveform from 0ns to 100ns. The `clk`, `reset`, control, input, and output signals should all be clearly visible. The bit vectors `A`, `B`, `C`, `D`, `Z1`, and `Z2` should also be unsigned decimal radix (check the adding signals to waveform guide if you need help formatting or changing the radix).
   - Your output waveform of `testbench2.vhd` computing $g(a, b, c)$ and $g(c, d, a)$ using the default values of $a = 5$, $b = 6$, $c = 7$, and $d = 8$. Show the waveform from 0ns to 150ns. If it takes longer that 150ns to compute your output, show enough of the waveform to clearly show the result (multiple screenshots are fine). Format all the waveform signals the same way as the previous test bench.