CSE100 Lab 2 – Arithmetic Logic Unit

Contents

1 Submission

- Due: Thursday 10/10/2024 end of lab section.
- Submit your Prelab answers to Gradescope.
- Submit your code to the Gradescope autograder. (You have unlimited submission attempts).
- Demonstrate your implementation to a TA.

2 Goals

You will implement an Arithmetic Logic Unit, capable of doing addition, subtraction, square root, and base 2 logarithms. This assignment will give you practice with MUXes.

Once you finish your implementation, you will program it to a Basys 3 FPGA Board.

3 Background

An Arithmetic Logic Unit is often used in processors to perform arithmetic. By performing all arithmetic in one hardware block, the data path logic is simplified.

ALUs make strong use of multiplexers to send the correct output for a desired operation. This lab will provide good practice for using MUXes.

4 Prelab

1. Find first set

A common hardware operation is called "find first set"; (https://en.wikipedia.org/wiki/Find_first_set). This operation should return the index of the most-significant 1 in a binary number. For example, `FFS(0b1)==0`, `FFS(0b100)==2`, `FFS(0b100101)==5`. Note that this operation also performs $\lfloor \log_2 \rfloor$.

Draw out a circuit diagram of how to implement "find first set" using MUXes, up to a word length of 8 bits. Assume `FFS(0b0)==0`. You may use an ellipsis (...) to denote repeated cells.

2. Square-root Lookup Table

Large, custom lookup tables are common in hardware design. Some operations are very slow and complicated to compute, so it is often beneficial to precompute them and store the results in a large read-only memory (ROM).

You are provided a script, `"rtl/generate_sqrt_lut.py"`, which automatically generates a `"sqrt.memh"` file which has a list of precomputed square-roots. You can initialize the ROM by loading the `".memh"` file into an array using `$readmemh` from IEEE 1800-2023, Section 21.4; https://ieeexplore.ieee.org/document/10458102. Then, you should read from the precomputed Verilog array with continuous assignment, (`assign` statement).

The Artix-7 has dedicated memory cells that Verilog arrays can be mapped onto. However, the Artix-7 memories do not support being read asynchronously, (with continuous assignment). Therefore, Vivado will map your Verilog array onto LUTs, not its dedicated memories. In the next lab, you will learn how to use the Artix-7 memory cells.

Answer the following questions. (Be concise).

a. What dimensions should your SQRT Lookup Table have? (word length $\times$ number of words).
b. What line of code is needed to initialize your BRAM with the precomputed SQRT values?
c. Explain how the sqrt ROM would be trivial to implement if the Artix-7 had LUT8 cells.
d. The Artix-7's largest LUT is a LUT6: https://docs.amd.com/r/en-US/ug953-vivado-7series-libraries/LUT6. Show how to build a
   - LUT7 from two LUT6s and a MUX2
   - LUT8 from two LUT7s and a MUX2

5 Lab

You need to complete and submit the following files:

- `"rtl/flog2.sv"`
- `"rtl/sqrt.sv"`
- `"rtl/alu.sv"`
- `"synth/basys3/Basys3_Master.xdc"`
- `"synth/basys3/basys3.sv"`

Note that you may only use continuous assignment. Always blocks are disallowed.

1. Create a `flog2` module inside the file `"rtl/flog2.sv"`, and implement your design you drew in the Prelab. Assume `FFS(0b0)==0`.
2. Create a `sqrt` module inside the file `"rtl/sqrt.sv"`, and implement the design using a ROM, as specifed in the Prelab.
3. Finish the `alu` module in `"rtl/alu.sv"` according to this operation encoding:

Encoding	Operation
`operation==4'b0001`	ADD
`operation==4'b0010`	SUB
`operation==4'b0100`	FLOG2
`operation==4'b1000`	SQRT
else	`y = 0`

The `alu` module should instantiate `flog2` and `sqrt` and create an adder and subtractor using the `+` and `-` operators. Then, a series of MUX2s should be used to select between the 4 different operators.

4. Complete the Basys3 configuration: according to these specifications:

- The `basys3` module inside the file `"synth/basys3/basys3.sv"` should instantiate and drive an `alu`.
- The Basys3 constraints file `"synth/basys3/Basys3_Master.xdc"` was downloaded from Digilent's GitHub: https://github.com/Digilent/Basys3/blob/master/Resources/XDC/Basys3_Master.xdc. Complete it to match your `basys3` module.
- The bottom 8 switches should control `alu.a_i`.
- The top 8 switches should control `alu.b_i`.
- The 4 directional buttons should control `alu.operation_i`.
- The bottom 8 LEDs should be controlled by `alu.y_o`.

Refer to Lab 1 and the Basys3 Reference Manual if you have questions: https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual.

6. Simulate your design using commands specified in the `"README.md"`. Submit to the autograder until you get 100%.
7. Synthesize your design and program it to the Basys3 using commands specified in the `"README.md"`. Get checked off by a TA.