Recitation 5

Behavioral Verilog

# Introduction

In this laboratory assignment, you will be learning more about **behavioral** Verilog and re-create your ALU in behavioral Verilog.

# Collaboration Policy

You will be working in groups of 2-3. Groups are allowed to collaborate.

# Equipment

* Computer with Quartus Prime software

# Tasks

To receive credit for this lab, you must complete:

* Task 1: Watch YouTube video: <https://www.youtube.com/watch?v=pOlxQnMw3Ss>
* Task 2: Recreate your ALU in **behavioral** Verilog. You can choose 2 or 3 operations among add, subtract, and, or, sll, and sra to implement. Use testbenches to test your code.

You must complete all parts of this recitation to receive credit. Ensure that a TA marks the completion of the tasks in Sakai.

# Grading

* Completing Recitation Tasks: 1 point (pass/fail)

Structural vs Behavioral Verilog

Structural Verilog is writing “code” that physically describes how components such as gates, decoders, multiplexers, and other digital logic devices are connected/wired together. It is similar to taking a breadboard circuit and writing up how each component is put together by labeling every IC and connecting wire. Since you are describing how a set of components is connected, the order does not matter.

Behavioral Verilog describes the function of a circuit. This is usually done with always and/or assign statements. With behavioral Verilog, the compiler and synthesizer will determine and optimize the circuit layout of your code.

### Module Interface

Your module should use the following interface:

module alu(data\_operandA, data\_operandB, ctrl\_ALUopcode, ctrl\_shiftamt, data\_result, isNotEqual, isLessThan, overflow);

##### input [31:0] data\_operandA, data\_operandB;

##### input [4:0] ctrl\_ALUopcode, ctrl\_shiftamt;

##### output [31:0] data\_result;

##### output isNotEqual, isLessThan, overflow;

##### endmodule

Each operation should be associated with the following ALU opcodes:

| **Operation** | **ALU Opcode** | **Description** |
| --- | --- | --- |
| ADD | 00000 | Performs data\_operandA + data\_operandB |
| SUBTRACT | 00001 | Performs data\_operandA - data\_operandB |
| AND | 00010 | Performs (bitwise) data\_operandA & data\_operandB |
| OR | 00011 | Performs (bitwise) data\_operandA | data\_operandB |
| SLL | 00100 | Logical left-shift on data\_operandA |
| SRA | 00101 | Arithmetic right-shift on data\_operandA |

#### Control Signals (In)

##### ctrl\_shiftamt

* + Shift amount for SLL and SRA operations
  + Only needs to be used in SLL and SRA operations

#### Information Signals (Out)

##### isNotEqual

* + Asserts true **iff** data\_operandA and data\_operandB are not equal
  + Only needs to be correct after a SUBTRACT operation

##### isLessThan

* + Asserts true **iff** data\_operandA is **strictly** less than data\_operandB
  + Only needs to be correct after a SUBTRACT operation

##### overflow

* + Asserts true **iff** there is an overflow in ADD or SUBTRACT
  + Only needs to be correct after an ADD or SUBTRACT operation

### Additional Resources

<http://www.csit-sun.pub.ro/courses/Masterat/Materiale_Suplimentare/Xilinx%20Synthesis%20Technology/toolbox.xilinx.com/docsan/xilinx4/data/docs/xst/verilog3.html>

### Sample Testbench

Below is a sample testbench. Please try to write your own test cases as needed. For each one of the operations that you choose to implement, your testbench must have at least 3 test cases to verify the correctness of your implementation.

