EC 413 - LAB 5

ALU Module Description
In terms of the ALU module, we have both arithmetic and logic units. The following ALU modules are parameterized. With the naming convention OPERATION_Nbit

Opcode	**Operation**
000	MOV
001	NOT
010	ADD
011	NOR
100	SUB
101	NAND
110	AND

Parameterized NOT, NOR, NAND, AND are designed using the built in gates in Vivado within a loop and using genvar.

Move utilizes wires and not gates. In1[i] is passed through a not gate and a wire stores the output. That wire output then is passed through a second not gate and the output is stored into Out[i].

ADD and SUB utilize the FA_module within the genvar loop. However, to distinguish between ADD and SUB, when instantiating them into our top module, C_in for ADD is set to 0 and C_in for SUB is set to 1.

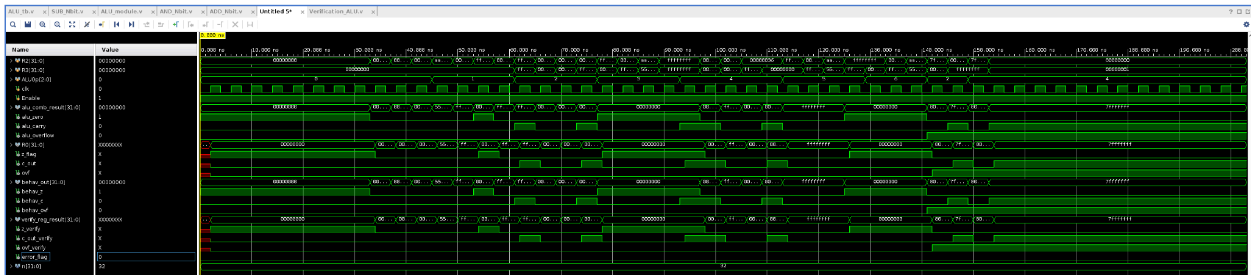
In terms of the ALU implementation, we perform every operation and store the results in a wire. Then, we have 3 to 8 MUX that selects the operation we would like to do and assigns the previously calculated result to a temporary variable (which is later assigned to the actual result register). Our ALU also has the following flag generations:

- **Zero Flag:** Set when Result == 0
- **Carry Out Flag:** Captures carry/borrow from addition/subtraction
- **Overflow Flag:** Detects signed overflow in ADD/SUB.

We also have a top module (which was not instantiated within the testbench) that connects the register and the ALU. It accepts 5 inputs (Operand 1, Operand 2, Opcode, Enable, Clk) and has 4 outputs (the same as those in the testbench). In the ALU testbench, the register and ALU are connected through wires.

The register module consists of a parameterized register (Register_Nbit.v) which stores ALU results. It only writes results when enable is set to one on the rising edge of the clk. The module uses the generate function to instantiate the specified WIDTH of D flip flops.

Generated Waveforms



Name	Value
R2[31:0]	00000000
R3[31:0]	00000000
ALUOp[2:0]	0
clk	0
Enable	1
alu_comb_result[31:0]	00000000
alu_zero	1
alu_carry	0
alu_overflow	0
R0[31:0]	XXXXXXXX
z_flag	X
c_out	X
ovf	X
behav_out[31:0]	00000000
behav_z	1
behav_c	0
behav_ovf	0
verify_reg_result[31:0]	XXXXXXXX
z_verify	X
c_out_verify	X
ovf_verify	X
error_flag	0
n[31:0]	32

You can see our generated waveform above. The opcode values range from 0 to 6 (as shown in the table above). We have examples showing functionality for each of our opcodes. Our testbench consists of a variety of test cases that show that our zero, carry, and overflow flags are raised whenever appropriate. It also shows the results of our verification module — the results match with the waveforms above, further corroborating that our ALU module performs operations correctly.