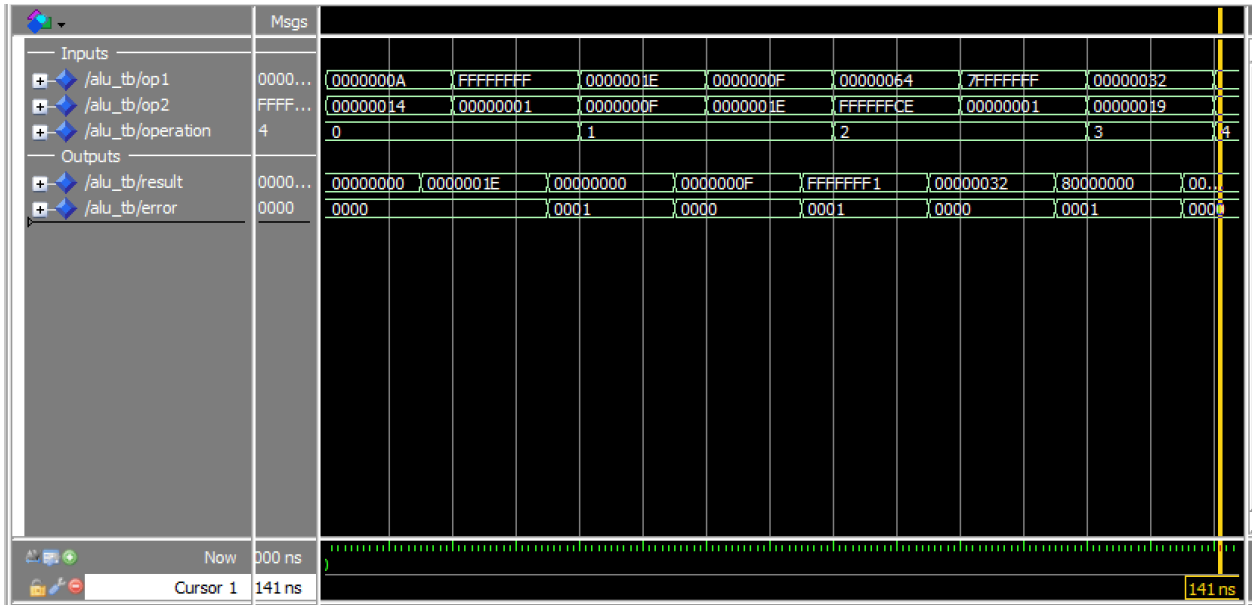
Will Gasser | wbg0023
COMP 4300
Lab 2

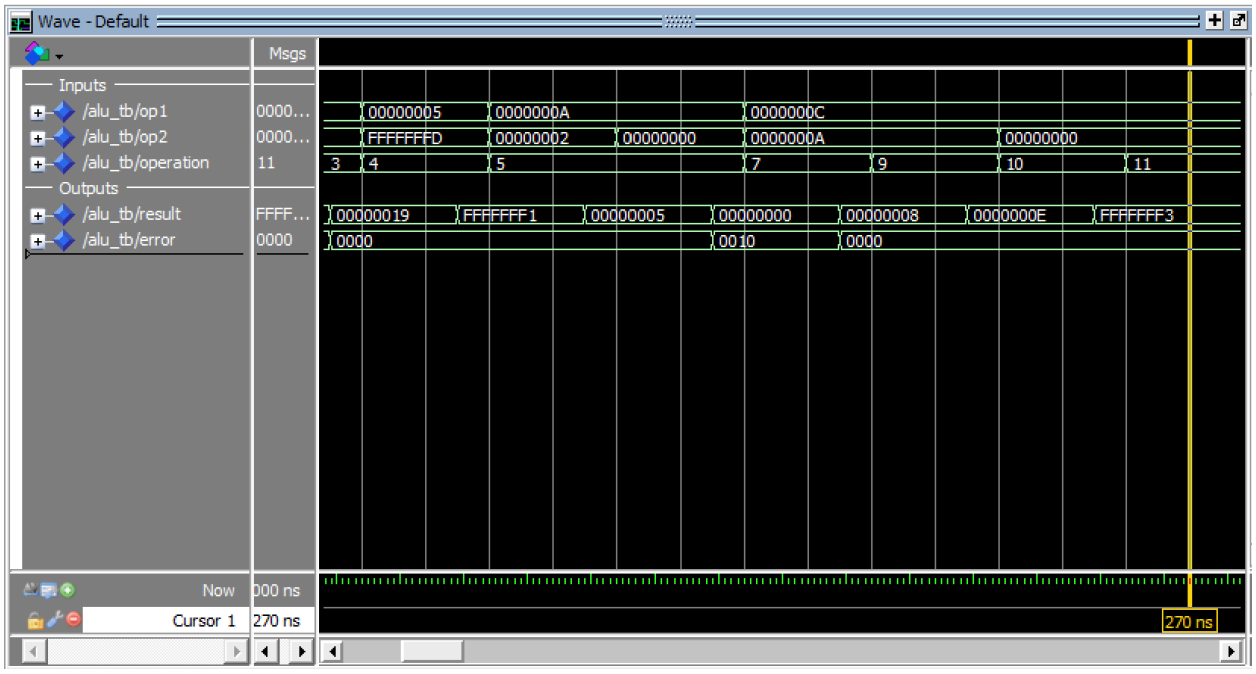
Testing

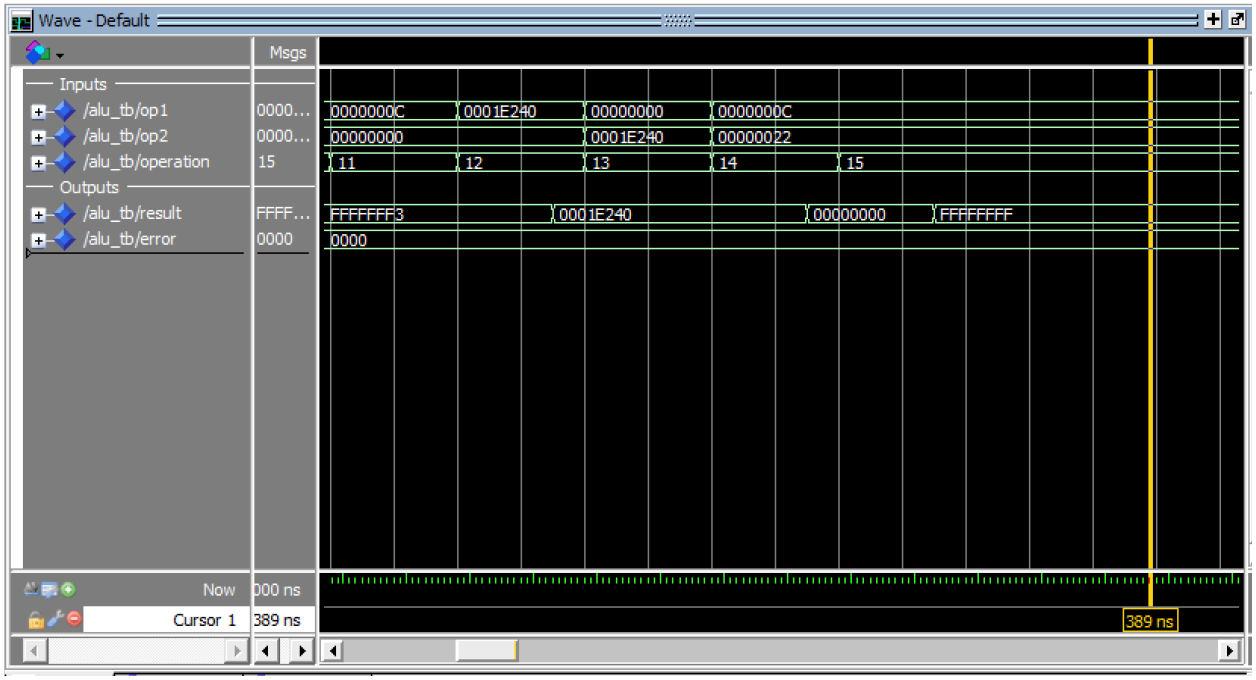
The following images show the simulation results over 500ns with each test case taking 20ns in a VHDL sim. The waveform displays input operands (op1, op2), the operation code, and the resulting output along with any error flags. The 15ns propagation delay is visible between input changes and corresponding output updates across all operations.

Test Descriptions

1. **Unsigned Addition (Normal)**: Tests 10 + 20 = 30. No overflow expected.
2. **Unsigned Addition (Overflow)**: Tests MAX_UINT + 1, causing overflow. Result should be 0 with error code 0001.
3. **Unsigned Subtraction (Normal)**: Tests 30 - 15 = 15. No underflow expected.
4. **Unsigned Subtraction (Underflow)**: Tests 15 - 30, causing underflow. Result should wrap around with error code 0001.
5. **Two's Complement Addition (Normal)**: Tests 100 + (-50) = 50. No overflow expected.
6. **Two's Complement Addition (Overflow)**: Tests INT_MAX + 1, causing positive overflow. Result should be MIN_INT with error code 0001.
7. **Two's Complement Subtraction (Normal)**: Tests 50 - 25 = 25. No overflow expected.
8. **Two's Complement Multiplication**: Tests 5 × (-3) = -15. No overflow expected.
9. **Two's Complement Division (Normal)**: Tests 10 ÷ 2 = 5. No error expected.
10. **Two's Complement Division (Divide by Zero)**: Tests division by zero. Should generate error code 0010.
11. **Bitwise AND**: Tests 0xC AND 0xA = 0x8. AND operation on bit patterns 1100 and 1010.
12. **Bitwise OR**: Tests 0xC OR 0xA = 0xE. OR operation on bit patterns 1100 and 1010.
13. **Logical NOT**: Tests NOT 0xC = 0xFFFFFFF3. Inverts all bits of operand1.
14. **Bitwise NOT**: Tests NOT 0xC = 0xFFFFFFF3. Should be identical to logical NOT for bit vectors.
15. **Pass Through Operations**: Tests operand passing and constant generation:
    - Pass operand1 (0x1E240)
    - Pass operand2 (0x1E240)
    - Output all zeros
    - Output all ones


Msgs
Inputs
/alu_tb/op1
/alu_tb/op2
/alu_tb/operation
Outputs
/alu_tb/result
/alu_tb/error
0000...
FFFF...
4
0000
0000000A
FFFFFFFF
0000001E
0000000F
00000064
7FFFFFFF
00000032
00000014
00000001
0000000F
0000001E
FFFFFFCE
00000001
00000019
0
1
2
3
4
00000000
0000001E
00000000
0000000F
FFFFFFF1
00000032
80000000
0000
0001
0000
0001
0000
0001
Now
000 ns
Cursor 1
141 ns
141 ns


Wave - Default
Msgs
Inputs
/alu_tb/op1
/alu_tb/op2
/alu_tb/operation
Outputs
/alu_tb/result
/alu_tb/error
0000...
FFFF...
11
0000
00000005
0000000A
0000000C
FFFFFFFD
00000002
00000000
0000000A
00000000
3
4
5
7
9
10
11
00000019
FFFFFFF1
00000005
00000000
00000008
0000000E
FFFFFFF3
0000
0010
0000
Now
000 ns
Cursor 1
270 ns
270 ns



ALU.vhd Code:

```
library ieee;
use ieee.std_logic_1164.all;
use work.dlx_types.all;
use work.bv_arithmetic.all;

entity alu is
        port(
        operand1, operand2: in dlx_word;
        operation: in alu_operation_code;
        result: out dlx_word;
        error: out error_code
        );
end entity alu;
```

```
architecture behavioral of alu is
begin
	alu_process: process(operand1, operand2, operation)
	variable temp_result: dlx_word;
	variable temp_error: error_code;
	variable overflow_flag: boolean;
	variable div_by_zero_flag: boolean;
	variable zero: dlx_word := (others => '0');
	variable ones: dlx_word := (others => '1');
	begin
	temp_error := "0000";

	case operation is
	when "0000" =>
		bv_addu(operand1, operand2, temp_result, overflow_flag);
		if overflow_flag then
		temp_error := "0001";
		end if;

	when "0001" =>
		bv_subu(operand1, operand2, temp_result, overflow_flag);
		if overflow_flag then
		temp_error := "0001";
		end if;

	when "0010" =>
		bv_add(operand1, operand2, temp_result, overflow_flag);
		if overflow_flag then
		temp_error := "0001";
		end if;

	when "0011" =>
		bv_sub(operand1, operand2, temp_result, overflow_flag);
		if overflow_flag then
		temp_error := "0001";
		end if;

	when "0100" =>
```

```
        bv_mult(operand1, operand2, temp_result, overflow_flag);
        if overflow_flag then
        temp_error := "0001";
        end if;

when "0101" =>
        if operand2 = zero then
        temp_error := "0010";
        temp_result := zero;
        else
        bv_div(operand1, operand2, temp_result, div_by_zero_flag, overflow_flag);
        if div_by_zero_flag then
        temp_error := "0010";
        elsif overflow_flag then
        temp_error := "0001";
        end if;
        end if;

when "0111" =>
        temp_result := operand1 and operand2;

when "1001" =>
        temp_result := operand1 or operand2;

when "1010" =>
        temp_result := not operand1;

when "1011" =>
        temp_result := not operand1;

when "1100" =>
        temp_result := operand1;

when "1101" =>
        temp_result := operand2;

when "1110" =>
        temp_result := zero;
```

```
        when "1111" =>
                temp_result := ones;

        when others =>
                temp_result := zero;
                temp_error := "0000";
        end case;

        result <= temp_result after 15 ns;
        error <= temp_error after 15 ns;
        end process alu_process;
end architecture behavioral;
```

testfile.tcl Code:

```
vcom -93 work/dlx_types.vhd
vcom -93 work/bva.vhd
vcom -93 work/bva-b.vhd
vcom -93 ALU.vhd


vsim -t ns work.alu

# waves
add wave -divider "Inputs"
add wave -radix hexadecimal /alu/operand1
add wave -radix hexadecimal /alu/operand2
add wave -radix unsigned /alu/operation
add wave -divider "Outputs"
add wave -radix hexadecimal /alu/result
```

```
add wave -radix binary /alu/error

# Test unsigned add (0000): normal case
echo "Test: Unsigned Addition - Normal Case"
force -freeze /alu/operand1 32'h0000000A 0
force -freeze /alu/operand2 32'h00000014 0
force -freeze /alu/operation 4'h0 0
run 20 ns

# Test unsigned add (0000): overflow case
echo "Test: Unsigned Addition - Overflow Case"
force -freeze /alu/operand1 32'hFFFFFFFF 0
force -freeze /alu/operand2 32'h00000001 0
force -freeze /alu/operation 4'h0 0
run 20 ns

# Test unsigned subtract (0001): normal case
echo "Test: Unsigned Subtraction - Normal Case"
force -freeze /alu/operand1 32'h0000001E 0
force -freeze /alu/operand2 32'h0000000F 0
force -freeze /alu/operation 4'h1 0
run 20 ns

# Test unsigned subtract (0001): underflow case
echo "Test: Unsigned Subtraction - Underflow Case"
force -freeze /alu/operand1 32'h0000000F 0
force -freeze /alu/operand2 32'h0000001E 0
force -freeze /alu/operation 4'h1 0
run 20 ns

# Test two's complement add (0010): normal case
echo "Test: Two's Complement Addition - Normal Case"
force -freeze /alu/operand1 32'h00000064 0
force -freeze /alu/operand2 32'hFFFFFFCE 0
force -freeze /alu/operation 4'h2 0
run 20 ns

# Test two's complement add (0010): overflow case
echo "Test: Two's Complement Addition - Overflow Case"
```

```
force -freeze /alu/operand1 32'h7FFFFFFF 0
force -freeze /alu/operand2 32'h00000001 0
force -freeze /alu/operation 4'h2 0
run 20 ns

# Test two's complement subtract (0011): normal case
echo "Test: Two's Complement Subtraction - Normal Case"
force -freeze /alu/operand1 32'h00000032 0
force -freeze /alu/operand2 32'h00000019 0
force -freeze /alu/operation 4'h3 0
run 20 ns

# Test two's complement multiply (0100): normal case
echo "Test: Two's Complement Multiply - Normal Case"
force -freeze /alu/operand1 32'h00000005 0
force -freeze /alu/operand2 32'hFFFFFFFD 0
force -freeze /alu/operation 4'h4 0
run 20 ns

# Test two's complement divide (0101): normal case
echo "Test: Two's Complement Divide - Normal Case"
force -freeze /alu/operand1 32'h0000000A 0
force -freeze /alu/operand2 32'h00000002 0
force -freeze /alu/operation 4'h5 0
run 20 ns

# Test two's complement divide (0101): divide by zero
echo "Test: Two's Complement Divide - Divide by Zero"
force -freeze /alu/operand1 32'h0000000A 0
force -freeze /alu/operand2 32'h00000000 0
force -freeze /alu/operation 4'h5 0
run 20 ns

# Test bitwise AND (0111)
echo "Test: Bitwise AND"
force -freeze /alu/operand1 32'h0000000C 0
force -freeze /alu/operand2 32'h0000000A 0
force -freeze /alu/operation 4'h7 0
run 20 ns
```

```
# Test bitwise OR (1001)
echo "Test: Bitwise OR"
force -freeze /alu/operand1 32'h0000000C 0
force -freeze /alu/operand2 32'h0000000A 0
force -freeze /alu/operation 4'h9 0
run 20 ns

# Test logical NOT (1010)
echo "Test: Logical NOT"
force -freeze /alu/operand1 32'h0000000C 0
force -freeze /alu/operand2 32'h00000000 0
force -freeze /alu/operation 4'hA 0
run 20 ns

# Test bitwise NOT (1011)
echo "Test: Bitwise NOT"
force -freeze /alu/operand1 32'h0000000C 0
force -freeze /alu/operand2 32'h00000000 0
force -freeze /alu/operation 4'hB 0
run 20 ns

# Test pass operand1 (1100)
echo "Test: Pass Operand1"
force -freeze /alu/operand1 32'h0001E240 0
force -freeze /alu/operand2 32'h00000000 0
force -freeze /alu/operation 4'hC 0
run 20 ns

# Test pass operand2 (1101)
echo "Test: Pass Operand2"
force -freeze /alu/operand1 32'h00000000 0
force -freeze /alu/operand2 32'h0001E240 0
force -freeze /alu/operation 4'hD 0
run 20 ns

# Test output all zeros (1110)
echo "Test: Output All Zeros"
force -freeze /alu/operand1 32'h0000000C 0
```

```
force -freeze /alu/operand2 32'h00000022 0
force -freeze /alu/operation 4'hE 0
run 20 ns

# Test output all ones (1111)
echo "Test: Output All Ones"
force -freeze /alu/operand1 32'h0000000C 0
force -freeze /alu/operand2 32'h00000022 0
force -freeze /alu/operation 4'hF 0
run 20 ns

echo "ALU test completed"
```