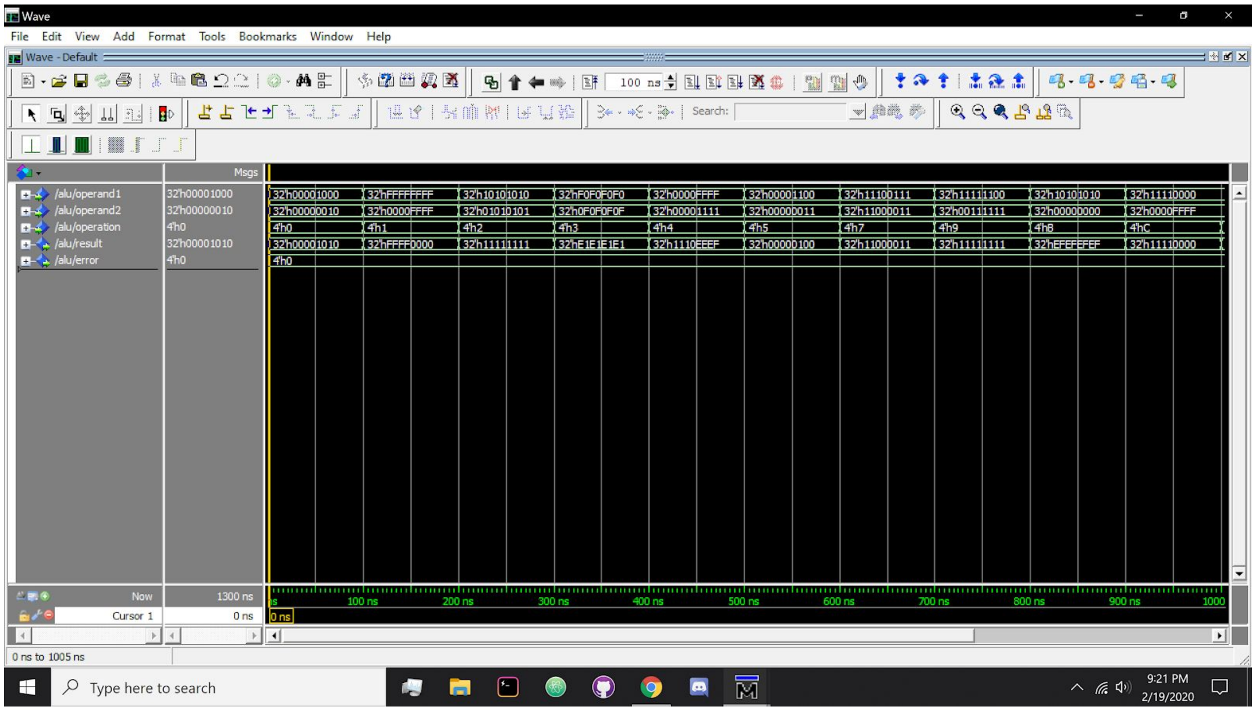
Run Log - No Errors

```
# vsim -gui
# Start time: 21:12:02 on Feb 19,2020
# Loading std.standard
# Loading work.bv_arithmetic(body)
# Loading work.dlx_types
# Loading work.alu(behaviour)
add wave sim:/alu/*
force -freeze sim:/alu/operand1 32'h00001000 0
force -freeze sim:/alu/operand2 32'h00000010 0
run
force -freeze sim:/alu/operand1 32'hFFFFFFFF 0
force -freeze sim:/alu/operand2 32'h0000FFFF 0
force -freeze sim:/alu/operation 4'h1 0
run
force -freeze sim:/alu/operand1 32'h10101010 0
force -freeze sim:/alu/operand2 32'h01010101 0
force -freeze sim:/alu/operation 4'h2 0
run
force -freeze sim:/alu/operand1 32'hF0F0F0F0 0
force -freeze sim:/alu/operand2 32'h0F0F0F0F 0
force -freeze sim:/alu/operation 4'h3 0
run
force -freeze sim:/alu/operand1 32'h0000FFFF 0
force -freeze sim:/alu/operand2 32'h00001111 0
force -freeze sim:/alu/operation 4'h4 0
run
force -freeze sim:/alu/operand1 32'h00001100 0
force -freeze sim:/alu/operand2 32'h00000011 0
force -freeze sim:/alu/operation 4'h5 0
run
force -freeze sim:/alu/operand1 32'h11100111 0
force -freeze sim:/alu/operand2 32'h11000011 0
force -freeze sim:/alu/operation 4'h7 0
run
force -freeze sim:/alu/operand1 32'h11111100 0
force -freeze sim:/alu/operand2 32'h00111111 0
force -freeze sim:/alu/operation 4'h9 0
run
force -freeze sim:/alu/operand1 32'h10101010 0
force -freeze sim:/alu/operand2 32'h00000000 0
force -freeze sim:/alu/operation 4'hB 0
```

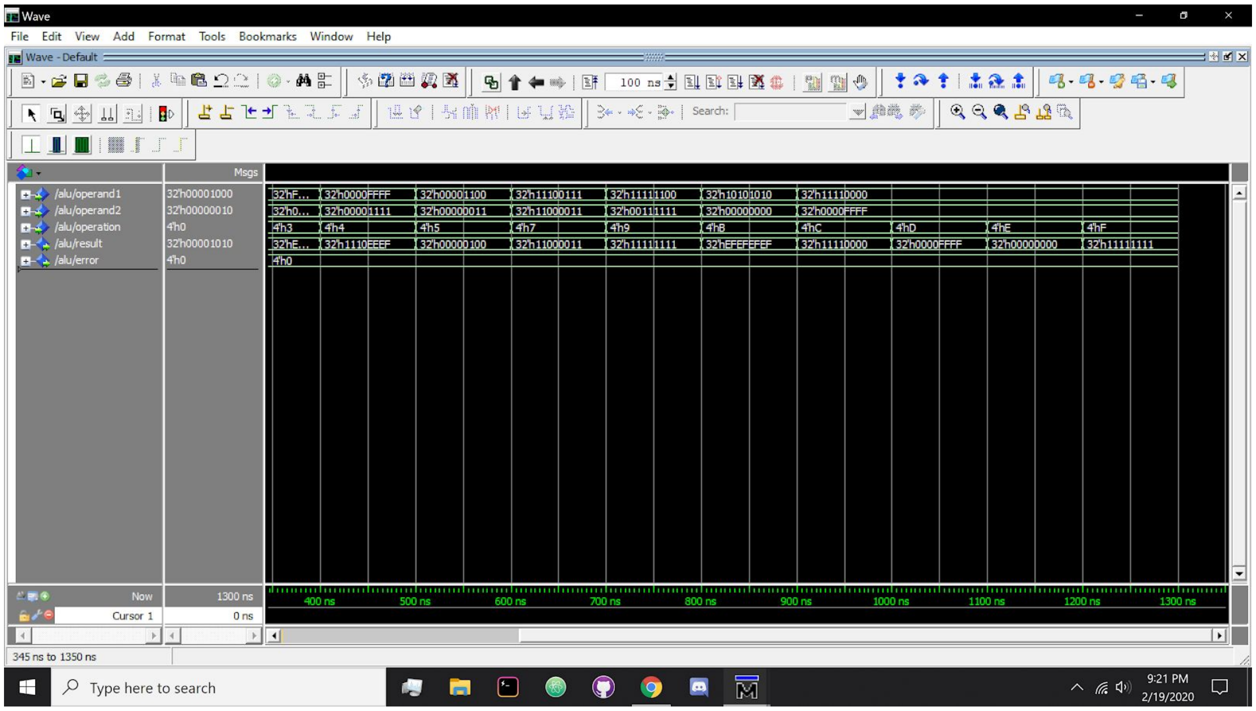
```
run
force -freeze sim:/alu/operand1 32'h11110000 0
force -freeze sim:/alu/operand2 32'h0000FFFF 0
force -freeze sim:/alu/operation 4'hC 0
run
force -freeze sim:/alu/operation 4'hD 0
run
force -freeze sim:/alu/operation 4'hE 0
run
force -freeze sim:/alu/operation 4'hF 0
run
```

## Screenshots - No Errors



```
Wave - Default
                 Msgs
/alu/operand1    32'h00001000   32'h00001000  32'hFFFFFFFF  32'h10101010  32'hF0F0F0F0  32'h0000FFFF  32'h00001100  32'h11100111  32'h11111100  32'h10101010  32'h11110000
/alu/operand2    32'h00000010   32'h00000010  32'h0000FFFF  32'h01010101  32'h0F0F0F0F  32'h00001111  32'h00000011  32'h11000011  32'h00111111  32'h00000000  32'h0000FFFF
/alu/operation   4'h0           4'h0          4'h1          4'h2          4'h3          4'h4          4'h5          4'h7          4'h9          4'hB          4'hC
/alu/result      32'h00001010   32'h00001010  32'hFFFF0000  32'h11111111  32'hE1E1E1E1  32'h1110EEEF  32'h00000100  32'h11000011  32'h11111111  32'hEFEFEFEF  32'h11110000
/alu/error       4'h0           4'h0

Now       1300 ns      100 ns  200 ns  300 ns  400 ns  500 ns  600 ns  700 ns  800 ns  900 ns  1000
Cursor 1  0 ns         0 ns
0 ns to 1005 ns
```



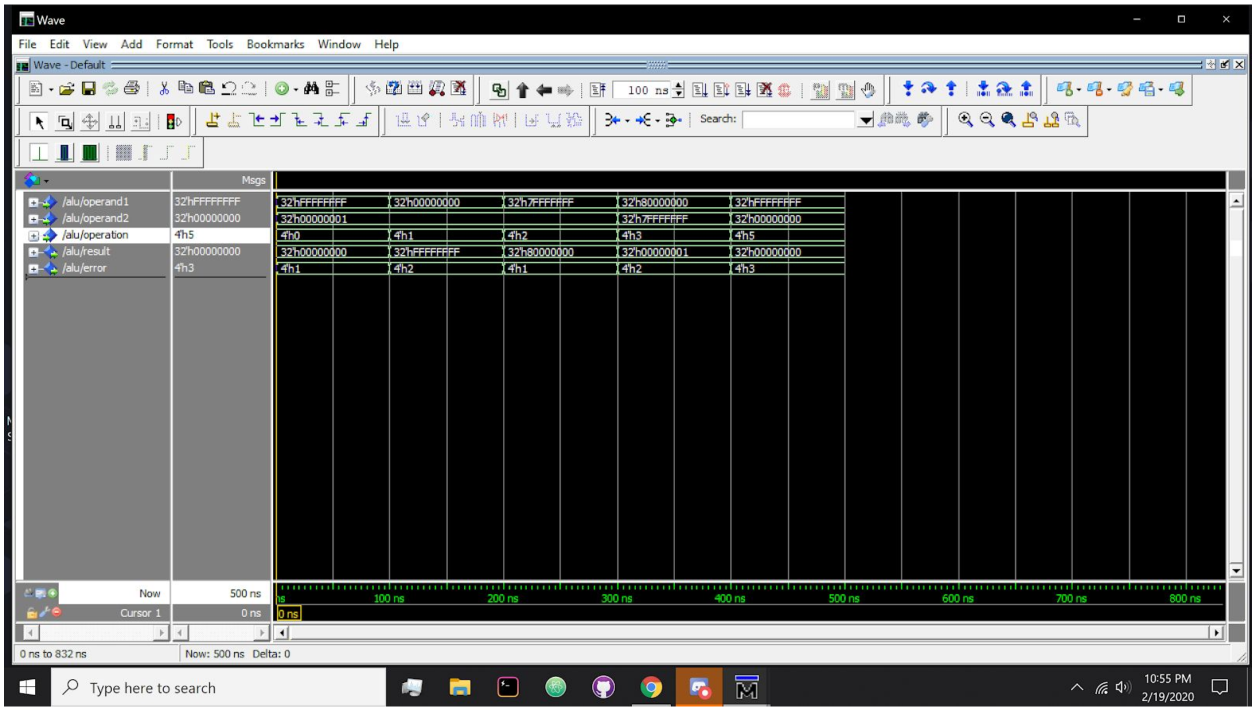
```
Wave - Default
                 Msgs
/alu/operand1    32'h00001000   32'hF...  32'h0000FFFF  32'h00001100  32'h11100111  32'h11111100  32'h10101010  32'h11110000
/alu/operand2    32'h00000010   32'h0...  32'h00001111  32'h00000011  32'h11000011  32'h00111111  32'h00000000  32'h0000FFFF
/alu/operation   4'h0           4'h3      4'h4          4'h5          4'h7          4'h9          4'hB          4'hC          4'hD          4'hE          4'hF
/alu/result      32'h00001010   32'hE...  32'h1110EEEF  32'h00000100  32'h11000011  32'h11111111  32'hEFEFEFEF  32'h11110000  32'h0000FFFF  32'h00000000  32'h11111111
/alu/error       4'h0           4'h0

Now       1300 ns      400 ns  500 ns  600 ns  700 ns  800 ns  900 ns  1000 ns  1100 ns  1200 ns  1300 ns
Cursor 1  0 ns
345 ns to 1350 ns
```

Run Log - Errors

```
# vsim -gui
# Start time: 22:49:08 on Feb 19,2020
# Loading std.standard
# Loading work.bv_arithmetic(body)
# Loading work.dlx_types
# Loading work.alu(behaviour)
add wave sim:/alu/*
force -freeze sim:/alu/operand1 32'hFFFFFFFF 0
force -freeze sim:/alu/operand2 32'h00000001 0
run
force -freeze sim:/alu/operand1 32'h00000000 0
force -freeze sim:/alu/operation 4'h1 0
run
force -freeze sim:/alu/operand1 32'h7FFFFFFF 0
force -freeze sim:/alu/operation 4'h2 0
run
force -freeze sim:/alu/operand1 32'h80000000 0
force -freeze sim:/alu/operand2 32'h7FFFFFFF 0
force -freeze sim:/alu/operation 4'h3 0
run
force -freeze sim:/alu/operand1 32'hFFFFFFFF 0
force -freeze sim:/alu/operand2 32'h00000000 0
force -freeze sim:/alu/operation 4'h5 0
run
```

## Screenshot - Errors



```
Wave - Default
                    Msgs
/alu/operand1   32'hFFFFFFFF   32'hFFFFFFFF  32'h00000000  32'h7FFFFFFF  32'h80000000  32'hFFFFFFFF
/alu/operand2   32'h00000000   32'h00000001                              32'h7FFFFFFF  32'h00000000
/alu/operation  4'h5           4'h0          4'h1          4'h2          4'h3          4'h5
/alu/result     32'h00000000   32'h00000000  32'hFFFFFFFF  32'h80000000  32'h00000001  32'h00000000
/alu/error      4'h3           4'h1          4'h2          4'h1          4'h2          4'h3

Now       500 ns
Cursor 1  0 ns

0 ns to 832 ns    Now: 500 ns  Delta: 0
```

VHDL Code

```
--Author:         Tanner Bonds
--Due Date:     2/21/2020
--Assignment Two

library work;
use work.bv_arithmetic.all;
use work.dlx_types.all;

entity alu is
        generic(prop_delay: Time := 15 ns);
        port(operand1, operand2: in dlx_word;
        operation: in alu_operation_code;
        result: out dlx_word;
        error: out error_code);
end entity alu;

architecture behaviour of alu is
        begin process (operand1, operand2, operation)
                variable overflow: boolean;
                variable zero: boolean;
                variable logic1: boolean := false;
                variable logic2: boolean := false;
                variable inputResult: dlx_word := x"00000000";
                begin case operation is

                        --Unsigned Add
                        when "0000" => error <= "0000";
                        bv_addu(operand1, operand2, inputResult, overflow);
                                if overflow then
                                        error <= "0001";
                                end if;
                                result <= inputResult;

                        --Unsigned Subtract
                        when "0001" => error <= "0000";
                        bv_subu(operand1, operand2, inputResult, overflow);
                                if overflow then
                                        error <= "0010";
                                end if;
                                result <= inputResult;
```

```
--Add Two's Complement
when "0010" => error <= "0000";
bv_add(operand1, operand2, inputResult, overflow);
        --Check For Overflow Error
        if overflow then
                --Determine Underflow or Overflow
                if (operand1(31) = '1') and (operand2(31) = '1') then
                        if inputResult(31) = '0' then
                                error <= "0010";
                        end if;
                elsif (operand1(31) = '0') and (operand2(31) = '0') then
                        if inputResult(31) = '1' then
                                error <= "0001";
                        end if;
                end if;
        end if;
        result <= inputResult;

-- Substract Two's Complement
when "0011" => error <= "0000";
bv_sub(operand1, operand2, inputResult, overflow);
        --Check For Overflow Error
        if overflow then
                --Determine Underflow or Overflow
                if (operand1(31) = '1') and (operand2(31) = '0') then
                        if inputResult(31) = '0' then
                                error <= "0010";
                        end if;
                elsif (operand1(31) = '0') and (operand2(31) = '1') then
                        if inputResult(31) = '1' then
                                error <= "0001";
                        end if;
                end if;
        end if;
        result <= inputResult;

-- Multiply Two's Complement
when "0100" => error <= "0000";
bv_mult(operand1, operand2, inputResult, overflow);
        --Check For Overflow Error
        if overflow then
                --Determine Underflow or Overflow
                if (operand1(31) = '1') and (operand2(31) = '0') then
```

```
                        error <= "0010";
                elsif (operand1(31) = '0') and (operand2(31) = '1') then
                        error <= "0010";
                end if;
        end if;
        result <= inputResult;

-- Divide Two's Complement
when "0101" => error <= "0000";
bv_div(operand1, operand2, inputResult, zero, overflow);
        --Check for Overflow
        if overflow then
                error <= "0010";
        elsif zero then
                error <= "0011";
        end if;
        result <= inputResult;

--Bitwise AND
when "0111" => error <= "0000";
for i in 31 downto 0 loop
        inputResult(i) := operand1(i) and operand2(i);
end loop;
result <= inputResult;

--Bitwise OR
when "1001" => error <= "0000";
for i in 31 downto 0 loop
        inputResult(i) := operand1(i) or operand2(i);
end loop;
result <= inputResult;

--Bitwise NOT
when "1011" => error <= "0000";
for i in 31 downto 0 loop
        inputResult(i) := NOT operand1(i);
end loop;
result <= inputResult;

--Operand1 To Output
when "1100" => error <= "0000";
--Operand2 To Output
when "1101" => error <= "0000";
```

```
                              result <= operand2;

                              --Output All Zeros
                              when "1110" => error <= "0000";
                              result <= x"00000000";

                              --Output All Ones
                              when "1111" => error <= "0000";
                              result <= x"11111111";

                              --All Other Results
                              when others=> result <= x"00000000";

                        end case;
                  end process;
      end architecture;
```