Lab3: Designing Arithmetic and Logic Unit(ALU)

Aaron Mai, Dustin Nguyen
CECS 440
March 11, 2021

College of Computer Engineering and Science
California State University, Long Beach

1. Introduction

In this lab we wrote and designed an ALU component that performs certain opcode operations depending on the control bits. The ALU takes in 2 32 bit inputs to which are then passed through the case statements. The ALU then outputs a 32 bit output as well as a 1 bit zero, carryout, and overflow flag. Each of the flags represent an aspect of the output. The zero flag is raised when the output is a 0. The carryout flag is raised when the msb of both numbers is the same and the end output has a different msb. This indicates that there has been a carry used. The overflow flag is similar in that it checks for the change in msb but this time it checks the sign bit to see if it has changed in the end result. After we have designed the ALU we proceed to test our design through a testbench. We first wrote out our tables for the values expected by the waveforms. We then coded 12 test cases to input to our ALU, each going under different operations. We checked our ALUout, Zero flag, overflow flag, and carryout flag to verify the functionality of our design. After confirming that our calculated values are the same as the ones on the waveform, we were able to confirm the operation of our ALU design.

2. ALU Unit Block Diagram and Control Line Operations

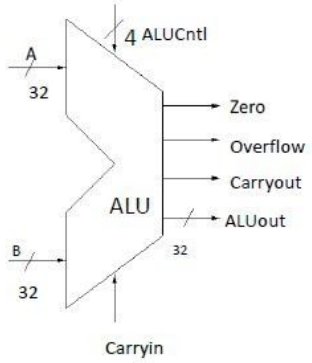


Figure 1: ALU Unit to be developed in this Lab project

Table 1: ALU control Lines and corresponding ALU Operations

ALUCntl	Function
0000	And
0001	Or
0011	Xor
0010	Add
0110	Subtract
1100	Nor

3. ALU VDHL code

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity alu_2 is
    port(
    -- inputs
    a: in std_logic_vector(31 downto 0);
    b: in std_logic_vector(31 downto 0);
    cin: in std_logic;
    sel: in std_logic_vector(3 downto 0);
    -- outputs
    AluOut: out std_logic_vector(31 downto 0);
    Z: out std_logic;
    overflow:  out std_logic;
    cout: out std_logic
    );
end alu_2;

architecture behavioral of alu_2 is

-- making a comparator signal thats 33 bits, use bit 33 to check cout
signal res: std_logic_vector(32 downto 0);
-- making registers for a + b input
signal areg, breg: std_logic_vector(32 downto 0);

begin
    -- concatenation operator
    areg <= '0' & a;
    breg <= '0' & b;

    process(areg,breg,cin,sel)

    begin
    -- lab 3 opcodes
    case(sel) is
        when "0000" => res  -- and
        <= areg and breg;
        when "0001" => res -- or
        <= areg or breg;
        when "0011" => res -- xor
        <= areg xor breg;
        when "0010" => res -- add
        <= areg + breg + cin;
        when "0110" => res -- sub
        <= areg - breg - cin;
        when "1100" => res -- nor
        <= areg nor breg;
        when others => NULL;
    end case;
```

```
    end process;
    -- ALUout gets temp res signal 32 bits
    AluOut <= res(31 downto 0);
    -- cout gets the 33rd bit of res
    cout <= res(32);
    -- zero flag triggers when our signal is = to zero
    Z <= '1' when
    (res = 0) else '0';
    -- coded overflow to occur whenever the same inputs get the opposite output
    overflow <= '1' when
    ((a(31) = '1' and b(31) = '1' and res(31) =  '0') or (a(31) = '0' and b(31) = '0' and
res(31) = '1')) else '0';

    end behavioral;
```

4. TestBenches

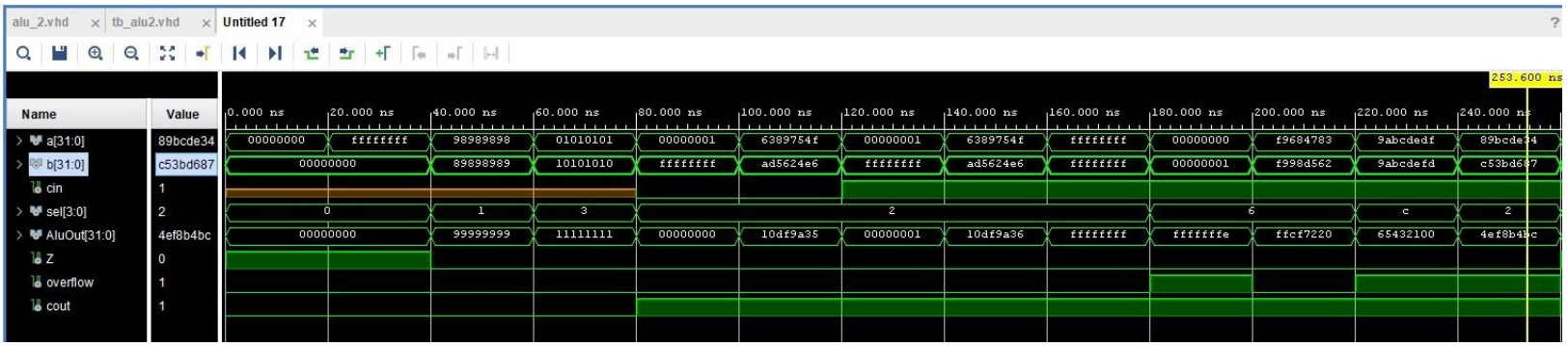
Testbench code for ALU

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY tb_alu2 IS
END tb_alu2;

ARCHITECTURE behavior OF tb_alu2 IS

 -- Component Declaration for the Unit Under Test (UUT)

 COMPONENT alu_2
 PORT(
 a : IN std_logic_vector(31 downto 0);
 b : IN std_logic_vector(31 downto 0);
 cin: IN std_logic;
 sel : IN std_logic_vector(3 downto 0);

 Z: OUT std_logic;
 overflow:  out std_logic;
 cout: out std_logic;
 AluOut: OUT std_logic_vector(31 downto 0)
 );
 END COMPONENT;


 -- Inputs
 signal a : std_logic_vector(31 downto 0) := (others => '0');
 signal b : std_logic_vector(31 downto 0) := (others => '0');
 signal cin: std_logic;
 signal sel : std_logic_vector(3 downto 0) := (others => '0');


 -- Outputs
 signal AluOut : std_logic_vector(31 downto 0);
 signal Z: std_logic;
 signal overflow: std_logic;
 signal cout: std_logic;


BEGIN

 -- Instantiate the Unit Under Test (UUT)
 uut: alu_2 PORT MAP (
 a => a,
 b => b,
 cin => cin,
 sel => sel,
```

```
Z => Z,
overflow => overflow,
cout => cout,
AluOut => AluOut
);

-- Stimulus process
stim_proc: process
begin
-- hold reset state for 100 ns.
wait for 20 ns;
-- insert stimulus here
a <= x"FFFFFFFF";
b <= x"00000000";
sel <= "0000"; -- and
wait for 20 ns;
a <= x"98989898";
b <= x"89898989";
sel <= "0001"; --or
wait for 20 ns;
a <= x"01010101";
b <= x"10101010";
sel <= "0011"; -- xor
wait for 20 ns;
a <= x"00000001";
b <= x"FFFFFFFF";
cin <= '0';
sel <= "0010"; -- add
wait for 20 ns;
a <= x"6389754F";
b <= x"AD5624E6";
cin <= '0';
sel <= "0010"; -- add
wait for 20 ns;
a <= x"00000001";
b <= x"FFFFFFFF";
cin <= '1';
sel <= "0010"; -- add
wait for 20 ns;
a <= x"6389754F";
b <= x"AD5624E6";
cin <= '1';
sel <= "0010"; -- add
wait for 20 ns;
a <= x"FFFFFFFF";
b <= x"FFFFFFFF";
cin <= '1';
sel <= "0010"; -- add
wait for 20 ns;
a <= x"00000000";
b <= x"00000001";
sel <= "0110"; -- sub
```

```
 wait for 20 ns;
 a <= x"F9684783";
 b <= x"F998D562";
 sel <= "0110"; -- sub
 wait for 20 ns;
 a <= x"9ABCDEDF";
 b <= x"9ABCDEFD";
 sel <= "1100"; -- nor
 wait for 20 ns;
 a <= x"89BCDE34";
 b <= x"C53BD687";
 cin <= '1';
 sel <= "0010"; -- add
 end process;
end;
```

5. Waveforms



6. Table 2 + Calculations

#	A_{hex}	B_{hex}	Carryin	$ALUCntl_{hex}$	$ALUout_{hex}$	Zero	Over flow	CarryOut
1	FFFFFFFF	00000000	-	0000	00000000	**1**	**0**	**0**
2	98989898	89898989	-	0001	99999999	**0**	**0**	**0**
3	01010101	10101010	-	0011	11111111	**0**	**0**	**0**
4	00000001	FFFFFFFF	0	0010	00000000	**0**	**0**	**1**
5	6389754F	AD5624E6	0	0010	10df9a35	**0**	**0**	**1**
6	00000001	FFFFFFFF	**1**	0010	00000001	**0**	**0**	**1**
7	6389754F	AD5624E6	**1**	0010	10df9a36	**0**	**0**	**1**
8	FFFFFFFF	FFFFFFFF	**1**	0010	ffffffff	**0**	**0**	**1**
9	00000000	00000001	-	0110	fffffffe	**0**	**1**	**1**
10	F9684783	F998D562	-	0110	ffcf7220	**0**	**0**	**1**
11	9ABCDEDF	9ABCDEFD	-	1100	65432100	**0**	**1**	**1**
12	89BCDE34	C53BD687	**1**	0010	4ef8b4bc	**0**	**1**	**1**

1. FFFFFFFF
   00000000 AND
   ————————
   00000000 And only gives a value of 1 if both inputs are 1, otherwise it will be a 0

2. 98989898
   89898989 OR
   ————————
   99999999 9 is 1001 and 8 is 1000 in binary. Since 8 has 1s in the same bit as 9, when we or we essential convert all the hex numbers to 9

3. 01010101
   10101010 Xor
   ————————
   11111111 Xor gives use a value of 1 if both inputs have different values therefore we get an answer with only 1s.

4. 00000001
   FFFFFFFF Add

   00000000 Since we go past 32 bits by adding we trigger the carry out but since the answer is 0 we have no overflow

5. 6389754F
   AD5624E6 ADD

   10df9a35 Since we are working with unsigned numbers and our values go over our 32 bits we raise a carry out flag but not a overflow flag

6. 00000001
   FFFFFFFF ADD

   00000001 And only gives a value of 1 if both inputs are 1, otherwise it will be a 0.

7. 6389754F
   AD5624E6 ADD

   10df9a36 Since we are adding bits and flagging our cout flag high, our result will be 10df9a36

8. FFFFFFFF
   FFFFFFFF ADD
   ---
   ffffffff And only gives a value of 1 if both inputs are 1, otherwise it will be a 0.

9. 00000000
   00000001 SUB
   ---
   fffffffe Since we are subtracting we are doing twos complement and therefore have a signed bit. This then lets us both raise the overflow and carryout flag.

10. F9684783
    F998D562 SUB
    ---
    ffcf7220 When subtracting we should get a value of ffcf7221 but this accounted with our carry out flagging high so we get ffcf7220

11. 9ABCDEDF
    9ABCDEFD NOR

    65432100 Since we are inversing NOR we only get a value of 1 if both numbers are 0.

12. 89BCDE34
    C53BD687 ADD

    4ef8b4bc We have a carry in therefore adding shifts the value of each bit up by 1.

Conclusion

In conclusion we were able to create, build, and verify a 32 bit ALU that performs certain opcode operations depending on the control bits. Within the lab we solidified our knowledge on creating signals, mapping ports, and vdhl's usage of bitwise operators. Specifically this lab helped strengthened my knowledge of creating a test bench with multiple stimuli as well as strengthen my knowledge of how an ALU operates.