MIPS 1 ALU

By Nik, Ben, Brycen

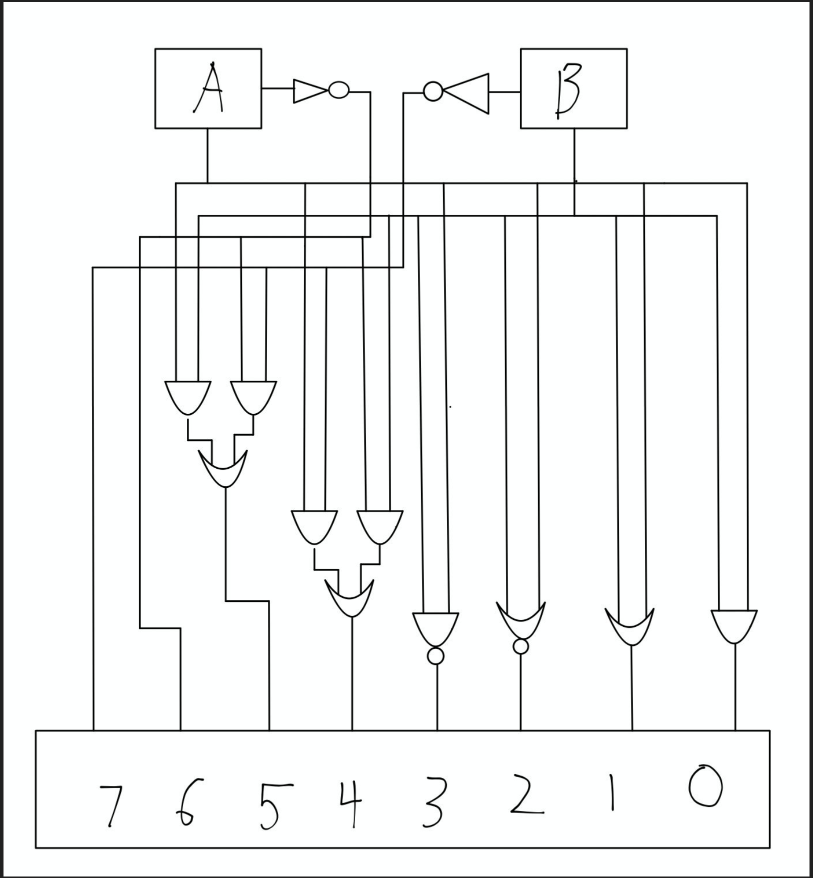
Instructions

The instructions we chose were AND, OR, XOR, NAND, NOR, XNOR, ~A, ~B.

We chose these because we thought it would be fun and interesting to put some less common logic gates used in ALUs into the system.

Hardware

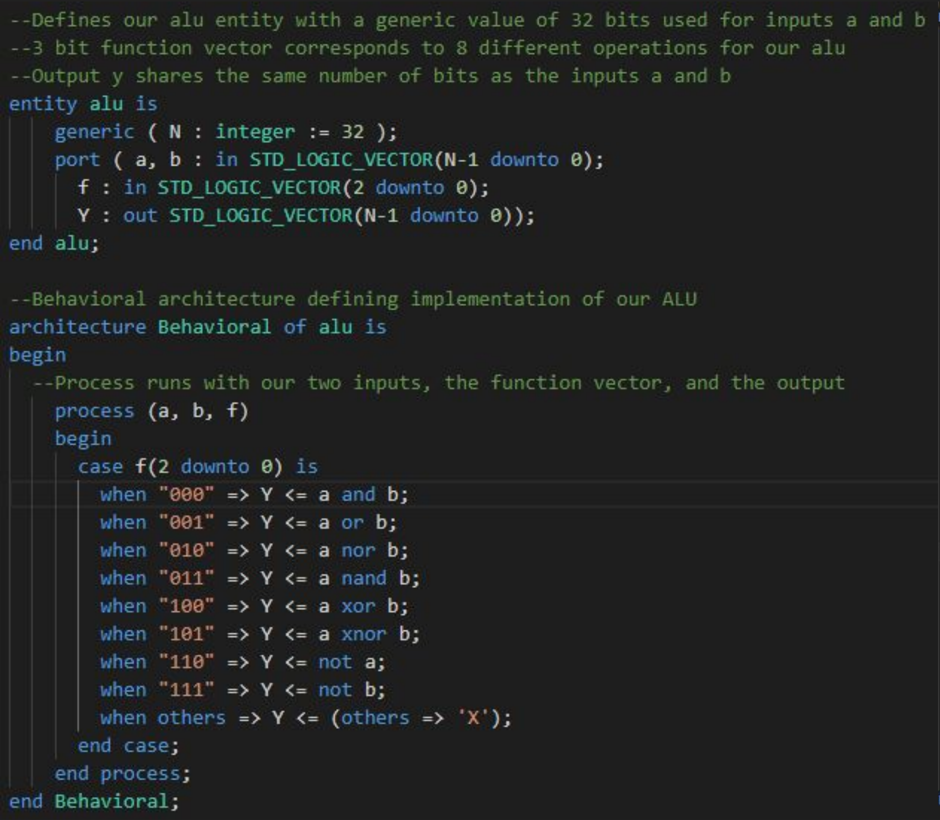
Op Code	Logic Function
0	A AND B
1	A OR B
2	A NOR B
3	A NAND B
4	A XOR B
5	A XNOR B
6	NOT A
7	NOT B



Efficient???

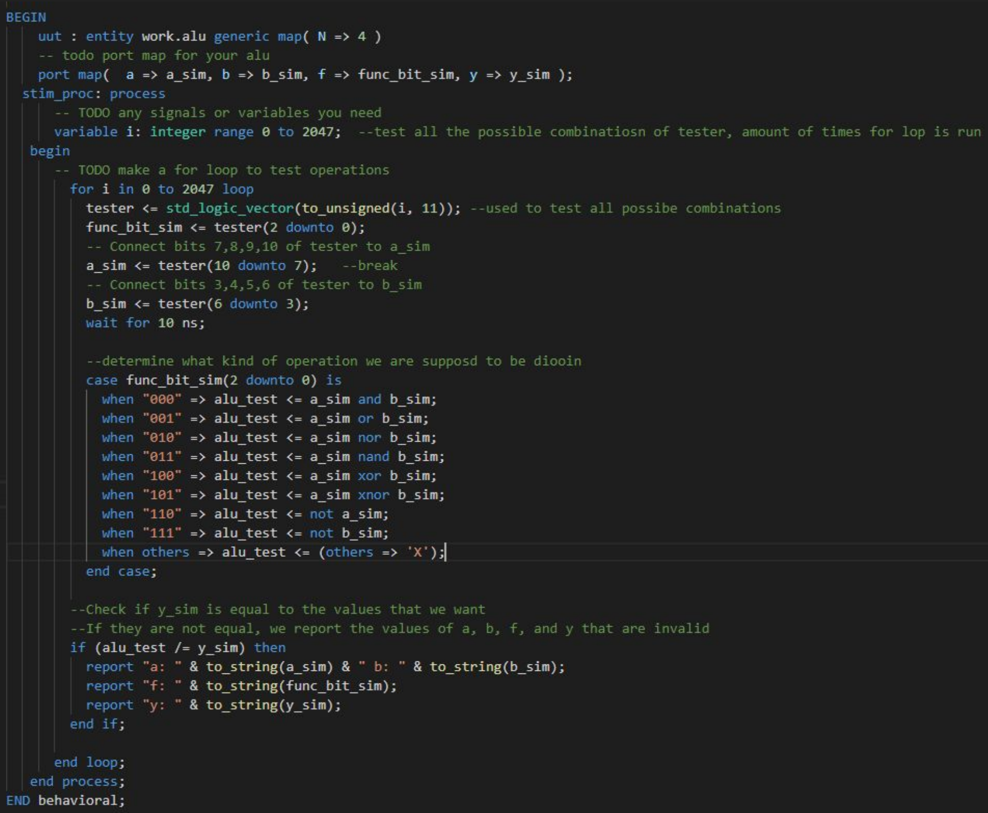
- 4 Slices of LUT
- 2 Slices total
- LUT as logic: 4
- 39 Bonded IOB’s used
- 1 slice per 4 instructions

Da Code



```vhdl
--Defines our alu entity with a generic value of 32 bits used for inputs a and b
--3 bit function vector corresponds to 8 different operations for our alu
--Output y shares the same number of bits as the inputs a and b
entity alu is
    generic ( N : integer := 32 );
    port ( a, b : in STD_LOGIC_VECTOR(N-1 downto 0);
      f : in STD_LOGIC_VECTOR(2 downto 0);
      Y : out STD_LOGIC_VECTOR(N-1 downto 0));
end alu;

--Behavioral architecture defining implementation of our ALU
architecture Behavioral of alu is
begin
  --Process runs with our two inputs, the function vector, and the output
    process (a, b, f)
    begin
      case f(2 downto 0) is
        when "000" => Y <= a and b;
        when "001" => Y <= a or b;
        when "010" => Y <= a nor b;
        when "011" => Y <= a nand b;
        when "100" => Y <= a xor b;
        when "101" => Y <= a xnor b;
        when "110" => Y <= not a;
        when "111" => Y <= not b;
        when others => Y <= (others => 'X');
      end case;
    end process;
end Behavioral;
```



```vhdl
BEGIN
    uut : entity work.alu generic map( N => 4 )
    -- todo port map for your alu
    port map(  a => a_sim, b => b_sim, f => func_bit_sim, y => y_sim );
  stim_proc: process
      -- TODO any signals or variables you need
      variable i: integer range 0 to 2047;  --test all the possible combinatiosn of tester, amount of times for lop is run
   begin
      -- TODO make a for loop to test operations
        for i in 0 to 2047 loop
          tester <= std_logic_vector(to_unsigned(i, 11)); --used to test all possibe combinations
          func_bit_sim <= tester(2 downto 0);
          -- Connect bits 7,8,9,10 of tester to a_sim
          a_sim <= tester(10 downto 7);   --break
          -- Connect bits 3,4,5,6 of tester to b_sim
          b_sim <= tester(6 downto 3);
          wait for 10 ns;

          --determine what kind of operation we are supposd to be diooin
          case func_bit_sim(2 downto 0) is
            when "000" => alu_test <= a_sim and b_sim;
            when "001" => alu_test <= a_sim or b_sim;
            when "010" => alu_test <= a_sim nor b_sim;
            when "011" => alu_test <= a_sim nand b_sim;
            when "100" => alu_test <= a_sim xor b_sim;
            when "101" => alu_test <= a_sim xnor b_sim;
            when "110" => alu_test <= not a_sim;
            when "111" => alu_test <= not b_sim;
            when others => alu_test <= (others => 'X');
          end case;

        --Check if y_sim is equal to the values that we want
        --If they are not equal, we report the values of a, b, f, and y that are invalid
        if (alu_test /= y_sim) then
          report "a: " & to_string(a_sim) & " b: " & to_string(b_sim);
          report "f: " & to_string(func_bit_sim);
          report "y: " & to_string(y_sim);
        end if;

      end loop;
   end process;
END behavioral;
```