```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity alu is
port (
     clk   : in  std_logic;
     op    : in  std_logic_vector(3  downto 0);
     a     : in  std_logic_vector(31 downto 0);
     b     : in  std_logic_vector(31 downto 0);
     y     : out std_logic_vector(63 downto 0)
);
end entity;

architecture behavioral of alu is

   component bmul32
      port( a : in  std_logic_vector(31 downto 0);    -- multiplier
            b : in  std_logic_vector(31 downto 0);    -- multiplicand
            p : out std_logic_vector(63 downto 0) -- product
      );
   end component bmul32;

   component CLAdder
      port(Xin: in std_logic_vector(15 downto 0);
       Yin: in std_logic_vector(15 downto 0);
       Cin: in std_logic;
       Sum: out std_logic_vector (15 downto 0);
       Cout: out std_logic
       );
   end component CLAdder;

   component divider
      PORT
      (
         denom    : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
         numer    : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
         quotient    : OUT STD_LOGIC_VECTOR (31 DOWNTO 0);
         remain      : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
      );
      end component divider;



   type op_type is (op_and, op_add, op_sub, op_nop, op_mul, op_div,op_or, op_shr, op_shl,
op_ror, op_rol, op_neg, op_not);

   signal enum_op : op_type;
   signal c_in: std_logic :='0';
   signal c_outH: std_logic;
   signal c_outL: std_logic;
   signal a_minus_b : std_logic_vector(32 downto 0);
   signal a_plus_b  : std_logic_vector(32 downto 0);
   signal a_mul_b  : std_logic_vector(63 downto 0);
   signal a_div_b : std_logic_vector(63 downto 0);
   signal a_shr_b : std_logic_vector(31 downto 0);
   signal a_shl_b : std_logic_vector(31 downto 0);
   signal a_ror_b : std_logic_vector(31 downto 0);
   signal a_rol_b : std_logic_vector(31 downto 0);
   signal reg       : std_logic_vector(32 downto 0);
   signal reg64       : std_logic_vector(63 downto 0);
   signal quotient : std_logic_vector(31 downto 0);
```

```
    signal remainder : std_logic_vector(31 downto 0);

begin

    CLAhigh: CLAdder port map (a(15 downto 0),
                               b (15 downto 0),
                               c_in,
                               a_plus_b(15 downto 0),
                               c_outL
                               );

    CLAlow: CLAdder port map (a(31 downto 16),
                               b (31 downto 16),
                               c_outL,
                               a_plus_b(31 downto 16),
                               c_outH
                               );
    a_plus_b(32)<=c_outH;
    booth: bmul32 port map (a,
                            b,
                            a_mul_b
                            );

    div32: divider port map (b,
                           a,
                           quotient,
                           remainder
                            );

    a_minus_b <= std_logic_vector(signed(a(a'high) & a) - signed(b(b'high) & b));

    a_div_b   <= remainder & quotient;
    a_shr_b   <= to_stdlogicvector(to_bitvector(a) srl to_integer( unsigned(b)));
    a_shl_b   <= to_stdlogicvector(to_bitvector(a) sll to_integer( unsigned(b)));
    a_ror_b   <= to_stdlogicvector(to_bitvector(a) ror to_integer( unsigned(b)));
    a_rol_b   <= to_stdlogicvector(to_bitvector(a) rol to_integer( unsigned(b)));
    process(op,a,b,clk)
    begin
       case op is
       when "0000" => enum_op <= op_and;
       when "0001" => enum_op <= op_add;
       when "0010" => enum_op <= op_sub;
       when "0011" => enum_op <= op_mul;
       when "0100" => enum_op <= op_div;
       when "0101" => enum_op <= op_or;
       when "0110" => enum_op <= op_shr;
       when "0111" => enum_op <= op_shl;
       when "1000" => enum_op <= op_ror;
       when "1001" => enum_op <= op_rol;
       when "1010" => enum_op <= op_neg;
       when "1011" => enum_op <= op_not;
       when others => enum_op <= op_nop;
       end case;
    end process;

    process(clk,a,b,op,a_plus_b,a_minus_b,a_mul_b,a_div_b,a_shl_b,a_shr_b,a_rol_b,a_ror_b)
    begin
       if clk='1' then
          case enum_op is
          when op_add       => reg64 <= (63 downto a_plus_b'length => '0')& a_plus_b;
          when op_sub       => reg64 <= (63 downto a_minus_b'length => '0') &  a_minus_b;
          when op_and       => reg64 <= (63 downto 32 => '0') & (a and b);
```

```
            when op_mul        => reg64 <= (63 downto a_mul_b'length => '0') & a_mul_b;
            when op_div        => reg64 <= (63 downto a_div_b'length => '0') & a_div_b;
            when op_or         => reg64 <= (63 downto 32 => '0') & (a or b);
            when op_shr        => reg64 <= (63 downto a_shr_b'length => '0') & a_shr_b;
            when op_shl        => reg64 <= (63 downto a_shl_b'length => '0') & a_shl_b;
            when op_ror        => reg64 <= (63 downto a_ror_b'length => '0') & a_ror_b;
            when op_rol        => reg64 <= (63 downto a_rol_b'length => '0') & a_rol_b;
            when op_neg        => reg64 <= (63 downto 32 => '0') & std_logic_vector(unsigned(not
(a)) + 1);
            when op_not        => reg64 <= (63 downto 32 => '0') & (not(To_X01(a)));

            when op_nop        =>
               reg(32) <= '0';
            end case;
         end if;
      end process;
      process(a_plus_b,a_minus_b,a_mul_b,a_div_b,a_shr_b,reg64)
         begin
            y <= reg64;
      end process;
   end architecture;
```