```
USE work.dlx_types.ALL;
USE work.bv_arithmetic.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 logic of alu is
begin
	process(operand1, operand2, operation) is
		variable overflow : boolean := false;
		variable op_result : dlx_word;
		variable divide_by_zero: boolean;
	begin
		error <= "0000";
		case(operation) is
			--unsigned add operation
			when "0000" =>
				bv_addu(operand1, operand2, op_result, overflow);
				if overflow then
					error <= "0001" after prop_delay; --overflow error code
				end if;
				result <= op_result after prop_delay;
```

```
--unsigned subtract operation
when "0001" =>
        bv_subu(operand1, operand2, op_result, overflow);
        if overflow then
                error <= "0001" after prop_delay;  --overflow error code
        end if;
        result <= op_result after prop_delay;
--twos compliment add operation
when "0010" =>
        bv_add(operand1, operand2, op_result, overflow);
        if overflow then
                error <= "0001" after prop_delay; --overflow error code
        end if;
        result <= op_result after prop_delay;
--twos compliment subtract
when "0011" =>
        bv_sub(operand1, operand2, op_result, overflow);
        if overflow then
                error <= "0001" after prop_delay; --overflow error code
        end if;
        result <= op_result after prop_delay;
--twos compliment multiply
when "0100" =>
        bv_mult(operand1, operand2, op_result, overflow);
        if overflow then
                error <= "0001" after prop_delay; --overflow error code
```

```
                end if;
                result <= op_result after prop_delay;
            --twos compliment divide
            when "0101" =>
                bv_div(operand1, operand2, op_result, divide_by_zero,
overflow);
                if divide_by_zero then
                    error <= "0010" after prop_delay; --divide by zero error
code
                end if;
                result <= op_result after prop_delay;
            --bitwise AND
            when "0111" =>
                result <= operand1 AND operand2 after prop_delay;
                error <= "0000" after prop_delay; --no error code
            --bitwise OR
            when "1001" =>
                result <= operand1 OR operand2 after prop_delay;
                error <= "0000" after prop_delay; --no error code
            --bitwise NOT of operand1 (ignore operand2)
            when "1011" =>
                result <= NOT operand1 after prop_delay;
                error <= "0000" after prop_delay; --no error code
            --pass operand1 through to output
            when "1100" =>
                result <= operand1 after prop_delay;
                error <= "0000" after prop_delay; --no error code
```

```
                --pass operand2 through to output
                when "1101" =>
                        result <= operand2 after prop_delay;
                        error <= "0000" after prop_delay; --no error code
                --output all zeros
                when "1110" =>
                        result <= (others => '0') after prop_delay;
                        error <= "0000" after prop_delay; --no error code
                --output all ones
                when "1111" =>
                        result <= (others => '1') after prop_delay;
                        error <= "0000" after prop_delay; --no error code
                --other cases
                when others =>
                        result <= (others => '0') after prop_delay;
                        error <= "0000" after prop_delay; --no error code
        end case;
    end process;
end architecture logic;
```