

```
-----  
-- Company:  
-- Engineer: Olasubomi Borishade  
--  
-- Create Date: 01/30/2026 07:46:57 PM  
-- Design Name: Computational Unit  
-- Module Name: Comp_Unit_TestBench - Behavioral  
-- Project Name: EENG 5560 Homework 1  
-- Description: A computational unit (CU) for 4 bit operands with 4 bit outputs that  
can compute the following operations:  
-- AND, NOR, Addition (ADD), Subtraction (SUB), Multiplication (MULT), Greater than  
(GT), Less than (LT) & Equal to (EQ)  
-- Revision:  
-- Revision 0.01 - File Created  
-- Additional Comments:  
--  
-----
```

```
library IEEE;  
use IEEE.STD_LOGIC_1164.ALL;  
use work.OPERATIONS_ARRAY_CUSTOMPACK.ALL;  
use IEEE.NUMERIC_STD.ALL;
```

```
-- Uncomment the following library declaration if instantiating  
-- any Xilinx leaf cells in this code.  
--library UNISIM;  
--use UNISIM.VComponents.all;
```

```
entity Computation_Unit is
```

```
Generic (d_w: integer := 4);
```

```
Port ( A : in STD_LOGIC_VECTOR (d_w - 1 downto 0); -- Declaring A as a input vector  
      B : in STD_LOGIC_VECTOR (d_w - 1 downto 0); -- Declaring B as a input vector  
      Opsel : in CU_Operation; -- Declaring the operation selection signal as an  
input vector  
      Y : out STD_LOGIC_VECTOR (d_w - 1 downto 0) -- Declaring Y as a output vector  
);
```

```
end Computation_Unit;
```

```
architecture Behavioral of Computation_Unit is
```

```
begin
```

```
ComputationUnit_proc: process (A, B, Opsel)
```

```
    variable Full_Addition : STD_LOGIC_VECTOR (d_w - 1 downto 0); -- Defining  
full-bit width intermediate addition signal
```

```
    variable Full_Subtraction : STD_LOGIC_VECTOR (d_w - 1 downto 0); -- Defining
```

```

full-bit width intermediate subtraction signal
    variable Full_Multiplication : STD_LOGIC_VECTOR (2 * d_w - 1 downto 0); --
Defining full-bit width intermediate multiplication signal

begin
    case Opsel is
        when lAND =>
            Y <= A AND B; -- Computes the logical AND of inputs A and B if the operation
selection signal is logical AND

        when lNOR =>
            Y <= A NOR B; -- Computes the logical NOR of inputs A and B if the operation
selection signal is logical NOR

        when aADD =>
            Full_Addition := STD_LOGIC_VECTOR (Unsigned(A) + Unsigned(B)); -- Computes
the full-bit width arithmetic addition of inputs A and B if the operation selection
signal is 010
            Y <= Full_Addition (d_w - 1 downto 0); -- Truncates the sum to our specified
bit width

        when aSUB =>
            Full_Subtraction := STD_LOGIC_VECTOR (Unsigned(A) - Unsigned(B)); --
Computes the full-bit width arithmetic subtraction of inputs A and B if the operation
selection signal is 011
            Y <= Full_Subtraction (d_w - 1 downto 0); -- Truncates the difference to our
specified bit width

        when aMULT =>
            Full_Multiplication := STD_LOGIC_VECTOR (Unsigned(A) * Unsigned (B)); --
Computes the full-bit width arithmetic multiplication of inputs A and B if the
operation selection signal is 100
            Y <= Full_Multiplication (d_w - 1 downto 0); -- Truncates the difference to
our specified bit width

        when cGTH =>
            if A > B then -- Checks if input A is greater than input B
                Y <= (others => '1'); -- If true, output is 1111
            else
                Y <= (others => '0'); -- If false, output is 0000
            end if;

        when cLTH =>
            if A < B then -- Checks if input A is less than input B
                Y <= (others => '1'); -- If true, output is 1111
            else

```

```
        Y <= (others => '0'); -- If false, output is 0000
    end if;

when cEQT =>
    if A = B then -- Checks if input A is equal to input B
        Y <= (others => '1'); -- If true, output is 1111
    else
        Y <= (others => '0'); -- If false, output is 0000
    end if;

when others =>
    Y <= (others => '0');

end case;
end process;

end Behavioral;
```