```
----------------------------------------------------------------------------------
-- Engineer: Olasubomi Borishade
--
-- Create Date: 02/04/2026 07:50:36 PM
-- Design Name: ARCA Logic Array
-- Module Name: ARCA_Logic_Array - Structural
-- Project Name: EENG 5560 Homework 2

-- Description: Heterogeneous 2 x 4 array of arithmetic, comparison and rotational
logic blocks

-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.OPERATIONS_ARRAY_CUSTOM_PACK.ALL;

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

entity ARCA_Logic_Array is
    Generic (d_w: integer := 3; -- parameterizing the data width
                rows: natural := 2; -- parameterizing the rows of the logic block
array
                cols: natural := 4 -- parameterizing the columns of the logic block
array
                );

    Port ( A, B : in VectorArray_1d (0 to cols - 1)(d_w - 1 downto 0); -- input
vector array for inputs A and B of each logic block
           OpselA : in ALBArray_2d (0 to rows - 1)(0 to 1); -- input vector array
for the arithmetic operation select signals of each logic block
           OpselC : in CLBArray_1d (0 to rows - 1); -- input vector array for the
comparison operation select signals of each logic block
           OpselR : in RLBArray_1d (0 to rows - 1); -- input vector array for the
rotational operation select signals of each logic block
           ADataFlow1: in VectorArray_1d (0 to 1)(0 downto 0); -- input selector
array for each A input of the second row of arithmetic logic blocks
```

```
            ADataFlow2: in VectorArray_1d (0 to 1)(0 downto 0); -- input selector
array for each B input of the second row of arithmetic logic blocks
            CDataFlow1: in STD_LOGIC_VECTOR (1 downto 0); -- input selector for each
A input of the second row of comparison logic block
            CDataFlow2: in STD_LOGIC_VECTOR (1 downto 0); -- input selector for each
B input of the second row of comparison logic block
            RDataFlow1: in STD_LOGIC_VECTOR (1 downto 0); -- input selector for each
A input of the second row of rotational logic block
            RDataFlow2: in STD_LOGIC_VECTOR (1 downto 0); -- input selector for each
B input of the second row of rotational logic block
            Y : out VectorArray_1d (0 to cols - 1)(d_w - 1 downto 0) -- output
vector array for the bottom row of logic blocks
            );

end ARCA_Logic_Array;

architecture Structural of ARCA_Logic_Array is
Signal As, Bs: VectorArray_1d (0 to cols - 1)(d_w - 1 downto 0); -- intermediate
input signal for the bottom row of logic blocks
signal Ys: VectorArray_2d(0 to rows - 1)(0 to cols - 1)(d_w - 1 downto 0); -- output
signal array of all logic blocks

begin
gen_rows: for r in 0 to rows - 1 generate
gen_cols: for c in 0 to cols - 1 generate

-- First Row Array Generation
--------------------------------------------------------------------------------

    First_Row_ALB1_Unit: if c = 0 and r = 0 generate
        ALB00_matrix: entity work.Arithmetic_Logic_Block(Behavioral)
        generic map(d_w => d_w)
        port map(A => A(c), B => B(c), OpSelA => OpSelA(0)(0), Y => Ys(r)(c));


    end generate First_Row_ALB1_Unit;


    First_Row_ALB2_Unit: if c = cols - 1 and r = 0 generate
        ALB03_matrix: entity work.Arithmetic_Logic_Block(Behavioral)
        generic map(d_w => d_w)
        port map(A => A(c), B => B(c), OpSelA => OpSelA(0)(1), Y => Ys(r)(c));

    end generate First_Row_ALB2_Unit;

-- Arithmetic Logic Blocks on the outer columns
```

```
-------------------------------------------------------------------------------


    First_Row_RLB_Unit: if c = 1 and r = 0 generate
        RLB01_matrix: entity work.Rotational_Logic_Block(Behavioral)
        generic map(d_w => d_w)
        port map(A => A(c), B => B(c), OpSelR => OpSelR(0), Y => Ys(r)(c));

    end generate First_Row_RLB_Unit;

-- Rotational Logic Blocks on the third column
-------------------------------------------------------------------------------

    First_Row_CLB_Unit: if c = 2 and r = 0 generate
        CLB02_matrix: entity work.Comparison_Logic_Block(Behavioral)
        generic map(d_w => d_w)
        port map(A => A(c), B => B(c), OpSelC => OpSelC(0), Y => Ys(r)(c));

    end generate First_Row_CLB_Unit;

-- Comparison Logic Blocks on the second column
-------------------------------------------------------------------------------







-- 2nd Row Array Generation
-------------------------------------------------------------------------------

Second_Row_ALB1_Unit: if c = 0 and r = rows - 1 generate
        ALB1_matrix: entity work.Arithmetic_Logic_Block(Behavioral)
        generic map(d_w => d_w)
        port map(A => As(c), B => Bs(c), OpSelA => OpSelA(1)(0), Y => Ys(r)(c));


    end generate  Second_Row_ALB1_Unit;


Second_Row_ALB2_Unit: if c = cols - 1 and r = rows - 1 generate
        ALB2_matrix: entity work.Arithmetic_Logic_Block(Behavioral)
        generic map(d_w => d_w)
        port map(A => As(c), B => Bs(c), OpSelA => OpSelA(1)(1), Y => Ys(r)(c));

    end generate Second_Row_ALB2_Unit;
```

```
-- Arithmetic Logic Blocks on the outer columns
-------------------------------------------------------------------------------


Second_Row_RLB_Unit: if c = 1 and r = rows - 1 generate
        RLB_matrix: entity work.Rotational_Logic_Block(Behavioral)
        generic map(d_w => d_w)
        port map(A => As(c), B => Bs(c), OpSelR => OpSelR(1), Y => Ys(r)(c));

    end generate Second_Row_RLB_Unit;

-- Rotational Logic Blocks on the third column
-------------------------------------------------------------------------------

Second_Row_CLB_Unit: if c = 2 and r = rows - 1 generate
        CLB_matrix: entity work.Comparison_Logic_Block(Behavioral)
        generic map(d_w => d_w)
        port map(A => As(c), B => Bs(c), OpSelC => OpSelC(1), Y => Ys(r)(c));

    end generate Second_Row_CLB_Unit;

-- Comparison Logic Blocks on the second column
-------------------------------------------------------------------------------




Adataflow1_proc: process(ADataFlow1) -- process is sensitive to each vector of the
ADataFlow1 array and selects the arithmetic block's A inputs
            begin
                case ADataFlow1(0) is -- if any vector in column 0 of array
ADataFlow1 reads:
                when "0" => -- a value of "0"
                As(0) <= Ys(0)(0); -- assign the same column of signal As the output
value of CU (0,0)

                when "1" => -- a value of "1"
                As(0) <= Ys(0)(1); -- assign the same column of signal As the output
value of CU (0,1)

                when others =>
                As(0) <= "000";
                end case;
```

```
                case ADataFlow1(1) is -- if any vector in column 1 of ADataFlow1
reads:
                when "0" => -- a value of "0"
                As(cols - 1) <= Ys(0)(2); -- assign the same column of signal As the
output value of ALB (0,2)

                when "1" => -- a value of "1"
                As(cols - 1) <= Ys(0)(3); -- assign the same column of signal As the
output value of ALB (0,3)

                when others =>
                As(cols - 1) <= "000";
                end case;
            end process;

Adataflow2_proc: process(ADataFlow2) -- process is sensitive to each vector of the
ADataFlow2 array and selects the arithmetic block's B inputs
            begin
                case ADataFlow2(0) is -- if any vector in a column 0 of array
ADataFlow2 reads:
                when "0" => -- a value of "0"
                Bs(0) <= Ys(0)(0); -- assign the same column of signal Bs the output
value of ALB (0,0)

                when "1" => -- a value of "1"
                Bs(0) <= Ys(0)(1); -- assign the same column of signal Bs the output
value of ALB (0,1)

                when others =>
                Bs(0) <= "000";
                end case;

                case ADataFlow2(1) is -- if any vector in column 1 of array
ADataFlow2 reads:
                when "0" => -- a value of "0"
                Bs(cols - 1) <= Ys(0)(2); -- assign the same column of signal Bs the
output value of ALB (0,2)

                when "1" => -- a value of "1"
                Bs(cols - 1) <= Ys(0)(3); -- assign the same column of signal Bs the
output value of ALB (0,3)

                when others =>
                Bs(cols -1 ) <= "000";
            end case;
            end process;
```

```
Cdataflow1_proc: process(CDataFlow1) -- process is sensitive to CDataFlow1 and
selects the comparison blocks A input
            begin
                case CDataFlow1 is -- if any vector in column 1 of array CDataFlow1
reads:
                when "00" => -- a value of "00"
                As(1) <= Ys(0)(0); -- assign the same column of signal As the output
value of CLB (0,0)

                when "01" => -- a value of "01"
                As(1) <= Ys(0)(1); -- assign the same column of signal As the output
value of CLB (0,1)

                when "10" => -- a value of "10"
                As(1) <= Ys(0)(2); -- assign the same column of signal As the output
value of CLB (0,2)

                when "11" => -- a value of "11"
                As(1) <= Ys(0)(3); -- assign the same column of signal As the output
value of CLB (0,3)

                when others =>
                As(1) <= "000";
            end case;
        end process;

Cdataflow2_proc: process(CDataFlow2) -- process is sensitive to CDataFlow2 and
selects the comparison block's B input
            begin
                case CDataFlow2 is -- if any vector in column 1 of array CDataFlow2
reads:
                when "00" => -- a value of "00"
                Bs(1) <= Ys(0)(0); -- assign the same column of signal Bs the output
value of CLB (0,0)

                when "01" => -- a value of "01"
                Bs(1) <= Ys(0)(1); -- assign the same column of signal Bs the output
value of CLB (0,1)

                when "10" => -- a value of "10"
                Bs(1) <= Ys(0)(2); -- assign the same column of signal Bs the output
value of CLB (0,2)

                when "11" => -- a value of "11"
                Bs(1) <= Ys(0)(3); -- assign the same column of signal Bs the output
```

```
value of CLB (0,3)

                when others =>
                Bs(1) <= "000";
            end case;
        end process;

Rdataflow1_proc: process(RDataFlow1) -- process is sensitive to RDataFlow1 and
selects the rotational block's A input
            begin
                case RDataFlow1 is -- if RDataFlow1 reads:
                when "00" => -- a value of "00"
                As(2) <= Ys(0)(0); -- assign the same column of signal As the output
value of CLB (0,0)

                when "01" => -- a value of "01"
                As(2) <= Ys(0)(1); -- assign the same column of signal As the output
value of CLB (0,1)

                when "10" => -- a value of "10"
                As(2) <= Ys(0)(2); -- assign the same column of signal As the output
value of CLB (0,2)

                when "11" => -- a value of "11"
                As(2) <= Ys(0)(3); -- assign the same column of signal As the output
value of CLB (0,3)

                when others =>
                As(2) <= "000";
            end case;
        end process;

Rdataflow2_proc: process(RDataFlow2) -- process is sensitive to RDataFlow2 and
selects the rotational block's B input
            begin
                case RDataFlow2 is -- if RDataFlow2 reads:
                when "00" => -- a value of "00"
                Bs(2) <= Ys(0)(0); -- assign the same column of signal Bs the output
value of CLB (0,0)

                when "01" => -- a value of "01"
                Bs(2) <= Ys(0)(1); -- assign the same column of signal Bs the output
value of CLB (0,1)

                when "10" => -- a value of "10"
                Bs(2) <= Ys(0)(2); -- assign the same column of signal Bs the output
```

```
value of CLB (0,2)

                when "11" => -- a value of "11"
                Bs(2) <= Ys(0)(3); -- assign the same column of signal Bs the output
value of CLB (0,3)

                when others =>
                Bs(2) <= "000";
            end case;
        end process;




end generate gen_cols;
end generate gen_rows;

Y <= Ys(rows - 1)(0 to cols - 1);

end Structural;
```