Report for Matrix Operations

Introduction

This project involves the implementation of a VHDL program that performs matrix operations on a 2x2 matrix derived from a std_logic_vector input. The operations include finding the greatest and least numbers in the matrix, rotating the matrix 90 degrees clockwise, and computing the sum of the original and rotated matrices. The project consists of three files: the main entity and architecture, a custom package for matrix types, and a testbench.

File 1: MatrixOps.vhd

Entity Declaration

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.std_logic_arith.all;

-- Import the custom package
use work.MatrixTypes.all;

entity MatrixOps is
    Port (
        input_vector   : in std_logic_vector(31 downto 0);
        max_val        : out integer;
        min_val        : out integer;
        rotated_matrix : out integer_matrix;
        sum_matrix     : out integer_matrix
    );
end MatrixOps;
```

The entity **MatrixOps** defines the input and output ports:

- **input_vector**: A 32-bit **std_logic_vector** representing the input matrix.
- **max_val** and **min_val**: Integers representing the maximum and minimum values in the matrix.
- **rotated_matrix**: A 2x2 integer matrix representing the 90-degree clockwise rotated matrix.
- **sum_matrix**: A 2x2 integer matrix representing the sum of the original and rotated matrices.

Architecture Implementation

```
20  architecture Behavioral of MatrixOps is
21      -- Define the original 2x2 matrix signal
22      signal matrix : integer_matrix;
23      signal rotated : integer_matrix;
24      signal sum_result : integer_matrix;
25  begin
26      -- Extract the 8-bit segments of input_vector and convert them to integers for matrix elements
27      matrix(0,0) <= conv_integer((signed(input_vector(31 downto 24))));
28      matrix(0,1) <= conv_integer((signed(input_vector(23 downto 16))));
29      matrix(1,0) <= conv_integer((signed(input_vector(15 downto 8))));
30      matrix(1,1) <= conv_integer((signed(input_vector(7 downto 0))));
```

The architecture **Behavioral** of **MatrixOps** defines signals for the original matrix, the rotated matrix, and the sum matrix. The input vector is divided into 8-bit segments, which are converted to integers and assigned to the matrix elements.

Finding Maximum and Minimum Values

```
32      -- Determine the maximum and minimum values in the matrix
33      max_val <= matrix(0,0) when (matrix(0,0) >= matrix(0,1) and matrix(0,0) >= matrix(1,0) and matrix(0,0) >= matrix(1,1)) else
34                 matrix(0,1) when (matrix(0,1) >= matrix(1,0) and matrix(0,1) >= matrix(1,1)) else
35                 matrix(1,0) when (matrix(1,0) >= matrix(1,1)) else
36                 matrix(1,1);
37
38      min_val <= matrix(0,0) when (matrix(0,0) <= matrix(0,1) and matrix(0,0) <= matrix(1,0) and matrix(0,0) <= matrix(1,1)) else
39                 matrix(0,1) when (matrix(0,1) <= matrix(1,0) and matrix(0,1) <= matrix(1,1)) else
40                 matrix(1,0) when (matrix(1,0) <= matrix(1,1)) else
41                 matrix(1,1);
```

The maximum and minimum values in the matrix are determined using a series of **when else** statements to compare the matrix elements.

Rotating the Matrix

```
43      -- Rotate the matrix 90 degrees clockwise and assign to rotated signal
44      rotated(0,0) <= matrix(1,0);
45      rotated(0,1) <= matrix(0,0);
46      rotated(1,0) <= matrix(1,1);
47      rotated(1,1) <= matrix(0,1);
```

The matrix is rotated 90 degrees clockwise by reassigning the elements to their new positions in the **rotated** signal. The rotated matrix is then assigned to the **rotated_matrix** output.

Computing the Sum of Matrices

```
52      -- Compute the sum of the original matrix and the rotated matrix, assigning to sum_result signal
53      sum_result(0,0) <= matrix(0,0) + rotated(0,0);
54      sum_result(0,1) <= matrix(0,1) + rotated(0,1);
55      sum_result(1,0) <= matrix(1,0) + rotated(1,0);
56      sum_result(1,1) <= matrix(1,1) + rotated(1,1);
57
58      -- Assign sum matrix to output
59      sum_matrix <= sum_result;
```

The sum of the original matrix and the rotated matrix is computed elementwise and assigned to the **sum_result** signal. The sum matrix is then assigned to the **sum_matrix** output.

File 2: MatrixTypes.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

package MatrixTypes is
    -- Define a custom 2D array type for integer matrices
    type integer_matrix is array (0 to 1, 0 to 1) of integer;
end MatrixTypes;
```

This file defines a custom package **MatrixTypes** that includes a type definition for a 2x2 integer matrix. This package is used in the main entity and testbench to handle matrix operations.

File 3: MatrixOps_tb.vhd

Testbench Entity

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Import the custom package to use integer_matrix
use work.MatrixTypes.all;

entity MatrixOps_tb is
end MatrixOps_tb;
```

The testbench entity **MatrixOps_tb** is defined to test the **MatrixOps** entity.

Testbench Architecture

```
architecture Behavioral of MatrixOps_tb is
    -- Component declaration for the Unit Under Test (UUT)
    component MatrixOps
        Port (
            input_vector   : in std_logic_vector(31 downto 0);
            max_val        : out integer;
            min_val        : out integer;
            rotated_matrix : out integer_matrix;
            sum_matrix     : out integer_matrix
        );
    end component;

    -- Signals for input and output ports
    signal input_vector   : std_logic_vector(31 downto 0) := (others => '0');
    signal max_val        : integer;
    signal min_val        : integer;
    signal rotated_matrix : integer_matrix;
    signal sum_matrix     : integer_matrix;
```

The testbench architecture **Behavioral** declares the **MatrixOps** component and signals for the input and output ports.

Stimulus Process

```
begin
    -- Instantiate the Unit Under Test (UUT)
    uut: MatrixOps
        Port map (
            input_vector   => input_vector,
            max_val        => max_val,
            min_val        => min_val,
            rotated_matrix => rotated_matrix,
            sum_matrix     => sum_matrix
        );

    -- Stimulus process to apply test cases
    stimulus: process
    begin
        -- Test Case 1: input_vector with values {4, 2, -1, 3} (represented in 8-bit signed integers)
        input_vector <= x"04" & x"02" & x"FF" & x"03"; -- 0x04, 0x02, 0xFF (-1), 0x03 in matrix
        wait for 10 ns;

        -- Assert statements for Test Case 1
        assert (max_val = 4) report "Test Case 1 Failed: max_val is incorrect" severity error;
        assert (min_val = -1) report "Test Case 1 Failed: min_val is incorrect" severity error;
        assert (rotated_matrix(0,0) = -1 and rotated_matrix(0,1) = 4 and rotated_matrix(1,0) = 3 and rotated_matrix(1,1) = 2)
            report "Test Case 1 Failed: rotated_matrix is incorrect" severity error;
        assert (sum_matrix(0,0) = 3 and sum_matrix(0,1) = 6 and sum_matrix(1,0) = 2 and sum_matrix(1,1) = 5)
            report "Test Case 1 Failed: sum_matrix is incorrect" severity error;
```

The stimulus process applies test cases to the **MatrixOps** entity and uses assert statements to verify the correctness of the outputs.

Additional Test Cases

```
        -- Assert statements for Test Case 2
        assert (max_val = 7) report "Test Case 2 Failed: max_val is incorrect" severity error;
        assert (min_val = -8) report "Test Case 2 Failed: min_val is incorrect" severity error;
        assert (rotated_matrix(0,0) = 5 and rotated_matrix(0,1) = -8 and rotated_matrix(1,0) = -6 and rotated_matrix(1,1) = 7)
            report "Test Case 2 Failed: rotated_matrix is incorrect" severity error;
        assert (sum_matrix(0,0) = -3 and sum_matrix(0,1) = -1 and sum_matrix(1,0) = -1 and sum_matrix(1,1) = 1)
            report "Test Case 2 Failed: sum_matrix is incorrect" severity error;

        -- Test Case 3: input_vector with values {0, -1, 1, 0}
        input_vector <= x"00" & x"FF" & x"01" & x"00"; -- 0x00, 0xFF (-1), 0x01, 0x00
        wait for 10 ns;

        -- Assert statements for Test Case 3
        assert (max_val = 1) report "Test Case 3 Failed: max_val is incorrect" severity error;
        assert (min_val = -1) report "Test Case 3 Failed: min_val is incorrect" severity error;
        assert (rotated_matrix(0,0) = 1 and rotated_matrix(0,1) = 0 and rotated_matrix(1,0) = 0 and rotated_matrix(1,1) = -1)
            report "Test Case 3 Failed: rotated_matrix is incorrect" severity error;
        assert (sum_matrix(0,0) = 1 and sum_matrix(0,1) = -1 and sum_matrix(1,0) = 1 and sum_matrix(1,1) = -1)
            report "Test Case 3 Failed: sum_matrix is incorrect" severity error;
```

Additional test cases are applied to ensure the correctness of the matrix operations. The testbench uses assert statements to verify the expected outputs for each test case.

Conclusion

This project successfully implements matrix operations in VHDL, including finding the maximum and minimum values, rotating the matrix, and computing the sum of matrices. The testbench verifies the correctness of the implementation through various test cases. The detailed report provides an overview of the project structure, code explanations, and the testing process.