Programming Assignment 2: 
3x3 Matrix Multiplication State Machine

Due March 10, 2023 (Friday) @ 11:59 PM on HuskyCT

1 Introduction

Matrices are a valuable tool used to perform operations on grouped data. In general, a matrix can be thought of a collection of vectors, with a certain number of rows N and columns M, forming an NxM matrix. In this assignment, we will perform the multiplication of two matrices \(A\) and \(B\), both with 3 rows and 3 columns (i.e. two 3x3 square matrices), and output the calculations to a third 3x3 matrix, \(C\). The goals of this assignment are to:

- Become familiarized with VHDL array data types.
- Implement a state machine in VHDL using a process.
- Use test benches to verify the correctness of a design.

Section 2 goes through the basic matrix math. Section 3 discusses the overall assignment. Section 4 discusses the state machine in depth. Section 5 discusses the test benches. Finally, 6 summarizes the deliverable to submit.

You are given a file `matrix.vhd`, which contains the `matrix` type for use in the source files. You do not need to modify this file. `pa2.vhd` contain skeleton code that you need to complete. The entirety of the state machine is contained within `pa2.vhd`. You are also provided with one test bench `smttest.vhd` which is partially complete. More details are given in 5.

2 Background

The format of the two input matrices \(A\) and \(B\), and the output matrix \(C\) is given as:

\[
A = \begin{bmatrix}
    a_{11} & a_{12} & a_{13} \\
    a_{21} & a_{22} & a_{23} \\
    a_{31} & a_{32} & a_{33}
\end{bmatrix} \quad B = \begin{bmatrix}
    b_{11} & b_{12} & b_{13} \\
    b_{21} & b_{22} & b_{23} \\
    b_{31} & b_{32} & b_{33}
\end{bmatrix} \quad C = \begin{bmatrix}
    c_{11} & c_{12} & c_{13} \\
    c_{21} & c_{22} & c_{23} \\
    c_{31} & c_{32} & c_{33}
\end{bmatrix}
\]

2.1 Multiplication

Matrix addition is computed by calculating the dot products between rows of \(A\) and columns of \(B\). The output of each element in \(C\) is then given as:

\[
\begin{bmatrix}
    a_{11} & a_{12} & a_{13} \\
    a_{21} & a_{22} & a_{23} \\
    a_{31} & a_{32} & a_{33}
\end{bmatrix} \begin{bmatrix}
    b_{11} & b_{12} & b_{13} \\
    b_{21} & b_{22} & b_{23} \\
    b_{31} & b_{32} & b_{33}
\end{bmatrix} = \begin{bmatrix}
    c_{11} & c_{12} & c_{13} \\
    c_{21} & c_{22} & c_{23} \\
    c_{31} & c_{32} & c_{33}
\end{bmatrix}
\]

where

\[c_{ij} = a_{i1} \cdot b_{1j} + a_{i2} \cdot b_{2j} + a_{i3} \cdot b_{3j}\]
2.2 Example

Say we have the following matrices:

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Using equations 1 and 2, multiplication is calculated as:

$$\begin{bmatrix} 66 & 78 & 90 \\ 78 & 93 & 108 \\ 90 & 108 & 126 \end{bmatrix}$$

3 Design Overview

We will implement the 3x3 square matrix multiplication algorithm in VHDL, by using a state machine to sequence the operations. Pseudocode of the algorithm we are implementing and is provided below:

```
Algorithm 1 Matrix Multiplication of 3x3 Square Matrices A and B

procedure Multiply(A, B)
    C ← zero matrix of size 3x3
    for i = 1 to 3 do
        for j = 1 to 3 do
            for k = 1 to 3 do
                C[i, j] ← C[i, j] + A[i, k] · B[k, j]
            end for
        end for
    end for
    return C
end procedure
```

4 State Machine

The state machine implementation is a Mealy state machine, as outputs are dependent on both the current state of the SM, and the transition. The entirety of the state machine is defined in pa2.vhd, and each input / output signal to the module is mapped to the external signals to the test bench. The diagram below shows the incomplete state diagram:

Matrix state machine diagram

Note that you may not modify the states or their transition paths, but you are responsible for completing the transition and output logic of each stage. You may also add additional signals to the architecture as needed.
The four states are IDLE, MULT, ADD, and DONE. These states are used to model algorithm on. On each rising edge of the clk signal, the state machine process should be invoked. Synchronously, if the reset signal is set to 1, the state machine should transition to IDLE regardless of the current state, load A and B into the matrices $A_{int}$ and $B_{int}$, and reset the matrices $C_{int}$ and C. The reset = 1 functionality is implemented for you. The state machine operation is described below:

- **IDLE**: The SM waits for the start signal to be asserted. When start = 1, the state machine resets the iterators i, j, and k. Then the SM transitions to MULT. If start = 0, then the state machine remains in IDLE.
- **MULT**: The SM computes one of the terms from the partial product, depending on i, j, and k, and stores it into a temporary buffer. The state transitions to the ADD state.
- **ADD**: i, j and k must be used to determine the next action. The possibilities are:
  1. There are more partial products in the current term to be computed.
  2. There are more terms in the current row to be computed.
  3. Every term in current row is computed, but there are still remaining rows.
  4. The entire matrix is complete.

In each case, the partial product stored in the buffer should be summed with the corresponding entry in $C_{int}$ to accumulate the result. Additionally, i, j, and k must be modified depending on which step in the algorithm the SM is at. If the algorithm is complete, the state machine should transition to DONE. In all other cases the SM still has calculations to perform, so it should transition to MULT and modify i, j, and/or k appropriately so in the next cycle the correct values are being indexed.

- **DONE**: The state machine should store $C_{int}$ into the output matrix C and transition to IDLE in the next cycle.

The state machine should only output ready = 1 on the transition from DONE to IDLE (i.e. when C is updated with the complete output), and should be set to ready = 0 for all other transitions.

5 Test Bench

The skeleton of the test bench smtest.vhd is given. It defines the A and B matrices the same as in the example from section 2.

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$
Within smtest.vhd, a constant compC matrix is defined with the expected output of $AB$. The test bench first populates A and B with the above values, asserts reset (to load the matrices into the design), and then waits for the SM algorithm to terminate. You are required to fill in the test bench process to do the following:

Using for loops, iterate through the output C and compC to confirm the correctness of your design. Use assert to verify that all the corresponding elements are equal. If any elements do not match, you should report an error message with a severity level of “FAILURE” (this will cause the simulation to stop at the problematic entry). In the event of a failure, you will need to debug your state machine. If the loops without any failed assertions, you should report an enthusiastic success message to the console.

6 Deliverables

Please submit a single PDF containing the following:

1. Your code of the following:
   - pa2.vhd
   - smtest.vhd

2. The following screenshots:
   - The output waveform from smtest.vhd from 0ns to 600ns. Ensure that A, B, C, start, clk, reset, ready, state, i, j, and k are visible. You will need to add state, i, j, and k to the window from pa2, as it will not appear by default. Instructions are here. It is okay if the final matrix C / each state is not visible in this screenshot.
   - The output waveform from smtest.vhd from 0ns to 200ns. Show the same signals as in the previous one. For this screenshot, ensure that every state in this range is clearly visible.
   - The output waveform from smtest.vhd from 550ns to 700ns. Show the same signals as in the previous one. For this screenshot, ensure that the output matrix C[1:3, 1:3] is clearly visible.
   - A screenshot of the TCL console with your report message.