# ECE 3401 Digital Systems Design – Spring 2026

## Programming Assignment 1:
## Digital Design of General Matrix-Matrix Multiplication (GEMM)

*Due February 27, 2026 @ 11:59 PM on HuskyCT*

This assignment's goal is to design the multiplication of two matrices A and B using the inner dot product matrix multiplication (GEMM), and output the calculations to a third matrix, C. The goals are:

- Become familiarized with modeling combinatorial and sequential logic in VHDL
- Implement a state machine that drives the control signals for the datapath
- Implement Datapath for the Multiply-Accumulate (MAC) Unit, and the control signals
- Use testbench to verify the correctness of a design

You are given a file `array.vhd`, which contains the matrix of size 3x3 type for use in the source file. You do not need to modify this file. `pa1.vhd` contains skeleton code that you need to complete for the controller and datapath components of the design. You are also provided with one test bench `smtest.vhd`.

## Matrix times Matrix Multiplication https://en.wikipedia.org/wiki/Matrix_multiplication

Assume NxN matrix sizes with N rows and N columns. The GEMM **inner-product matrix multiplication** of two input matrices A and B to compute the output matrix C follows the pseudocode:

```
for i = 1 to N do.                          ▷ iterate through row of A
    for j = 1 to N do                       ▷ iterate through columns of B
      for k = 1 to N do                     ▷ Iterate nonzeros in A row and B col
        C[i, j] = C[i, j] + A[i, k] * B[k, j]
```

GEMM iterates over each non-zero element $A[i,k]$ in the $i$th row of matrix $A$; for each such element, it iterates over all column indices $k$ of matrix $B$ and multiplies $A[i,k]$ with the corresponding scalar element $B[k,j]$. Each product contributes a partial sum to the inner product for $C[i,j]$. By accumulating these contributions across all $k$, the final value of each output element $C[i,j]$ is obtained. Repeating this process for all column indices $j$ computes the entire row $C[i,:]$ of the output matrix.

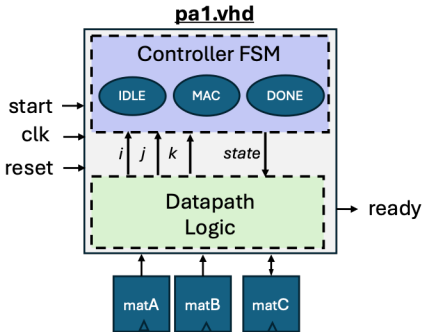
## GEMM Hardware Design

You will implement the inner product GEMM matrix multiplication in VHDL by using a state machine and datapath logic. The dot product is performed one element per cycle using a multiply-accumulate (MAC) unit.

The processes to implement the controller and datapath components need to be implemented in pa1.vhd where each input and output, internal signals, as well as state types are defined. The design is also mapped using the external input/output signals to the test bench, smtest.vhd. The design uses the array_pkg package as given to you in arrays.vhd, which contains the matrix type for the design. Note that DIM is a constant parameter set to 3 and each row and column of the 3x3 matrix indexes from 0 to 2. In any given testcase, the input matrices are initialize and the start input is set for 1 cycle to indicate to the design that it needs to compute the GEMM inner products. When the design completes the GEMM operations, it must set the *ready* output signal for 1 cycle to indicate to the testbench that the output matrix has the computed result.

You are not allowed to modify smtest.vhd or the arrays.vhd files. The input matrices received from the testbench are accessible to the design. The DIM parameter is also accessible to the design. The internal

signals are given, and you are not allowed to add or remove any of these internal signals. However, you will modify the pa1.vhd to implement the specified processes and concurrent assignments.




The figure shows the input/output interface for pa1.vhd. The datapath logic can read the input matrices and write the output matrix as required for the GEMM functionality. Furthermore, you are given a fixed number of state types that you must use in the design. The asynchronous reset logic is already given to you in pa1.vhd. It initializes the relevant state of the design components and must initialize the state to IDLE. The design must be synchronous to the rising edge of the clock signal.

The controller has three states, IDLE, MAC, and DONE. In IDLE state, the datapath i, j, k signals are reset to 0, and the start signal determines whether the design remains in IDLE or transition to the MAC state. The MAC state must increment the datapath i, j, k signals to transition the design to compute one dot product per cycle. At the end of the last MAC operation, the state machine must move to DONE state where the ready signal is asserted for only 1 cycle, and the datapath i, j, k signals are reset to 0. The state machine must then transition to either IDLE or MAC state based on the start signal.

The MAC unit datapath may use state, i, j, and/or k signals for the logic to read the two input matrices matA and matB and compute the dot product using behavioral arithmetic units (+ for addition and * for multiplication). The output matrix matC is not assumed to be initialized to zero, therefore the first partial product must only perform multiplication and skip accumulation. However, remaining partial products may use the multiply, add, and accumulate.

You must implement the controller and datapath logic using concurrent statements and/or processes to complete the implementation of the GEMM design in pa1.vhd.

## Design Verification

We have provided a fully functional smtest.vhd that initializes different GEMM inputs. Note that the initialization of the input is given in smtest.vhd for each testcase. After initialization and reset, each testcase asserts the start signal for one cycle and then waits for the ready signal from the design to report the matC output of the design. This process is done sequentially for each testcase.

## Deliverables

Please submit the following:

1. Your modified pa1.vhd VHDL file.
2. Submit a single PDF with your state machine diagram with explanation of inputs, outputs, and transitions. Also include the separate Datapath logic diagrams. Provide a short description of each design component to get credit.
3. Submit a single PDF with a screenshot of the following: Your output waveform that fully captures one iteration of all testcases. Note that the testbench will repeat if the simulation continues past the completion of the testcases. You must use the provided tb.gtkw file to generate this waveform.

In case your design is not fully functional, you will need to schedule a code review with the TA.