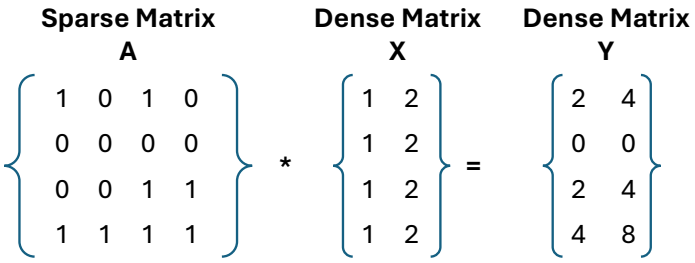
# ECE 3401 Digital Systems Design – Spring 2025

## Programming Assignment 4:
## Sparse Matrix Matrix Multiplication (SpMM)
## Datapath and Controller Design

*Due April 28, 2025 (Monday) @ 11:59 PM on HuskyCT*

Sparse Matrix Matrix Multiplication (SpMM) involves multiplying a sparse matrix (A) with a dense matrix (X) to produce a dense matrix (Y) that computes Y = AX. SpMM is a fundamental operator in sparse linear algebra and used for many scientific and engineering applications, specifically graph analytics and graph neural networks.

The sparse matrix A follows the Compressed Sparse Row (CSR) format used in programming assignment 3. This CSR data structure is highly useful as it provides, through adjacent reads, the starting and ending indices for each sparse matrix row. The Sparse Matrix Matrix Multiplication (SpMM) multiplies the sparse matrix A (of size ROW×ROW considering a square matrix) with a dense matrix X (of size ROW×DIMS) to compute the output dense matrix Y of size (ROW×DIMS). Each row of the A matrix computes the sparse dot products of its nonzeros with the corresponding row vectors of the dense matrix X to calculate the corresponding row vector output for the dense matrix Y. In the example below, the 4×4 sparse matrix performs row-wise sparse dot products to compute the output vector Y.




The SpMM performs dot product between a sparse row from the first matrix A and a dense column from the second matrix X to update the corresponding row (of A) and column (of B) element of the output matrix Y. For each row of matrix A, the SpMM performs matrix multiplications by iterating over all dimensions (DIMS) of the dense matrix X. For row 0 of sparse matrix A, the dim 0 column of dense matrix X is computed first. The nonzero at column 0 of A performs a dot product with the nonzero of the dim 0 column and the corresponding row 0 of X. This partial product is accumulated with the dot product between the second row 0 nonzero at column 2 of A and the corresponding dim 0 column and row 2 nonzero of X. The accumulated value of 2 is stored in dim 0 column and row 0 of dense output matrix Y. Since the dense matrices have DIMS=2, the second iteration commences and for row 0 of sparse matrix A, the dim 1 column of the dense matrix X is computed next to update dim 1 column and row 0 of output Y. The nonzero at column 0 of A performs a dot product with the nonzero of the dim 1 column and the corresponding row 0 of X. This partial product is accumulated with the dot product between the second row 0 nonzero at column 2 of A and the corresponding dim 1 column and row 2 nonzero of X. The accumulated value of 4 is stored in dim 1 column and row 0 of dense output matrix Y. Since no nonzeros

exist for row 1 in sparse matrix A, no dot product accumulations are needed. The row 2 of sparse matrix also performs two dot products and accumulations first for dim 0 column and then dim 1 column of the dense matrix A and update the dim 0 column and then the dim 1 column at row 2 of output matrix Y. Finally, the last row 3 of the sparse matrix performs four dot products and accumulations first for dim 0 column and then dim 1 column of the dense matrix X to compute the dim 0 and then dim 1 column and row 3 of dense output matrix Y.

## SpMM Algorithm

This assignment implements an SpMM algorithm in VHDL by using a state machine to sequence the sparse matrix-matrix operations. The pseudocode for the SpMM algorithm is given below.

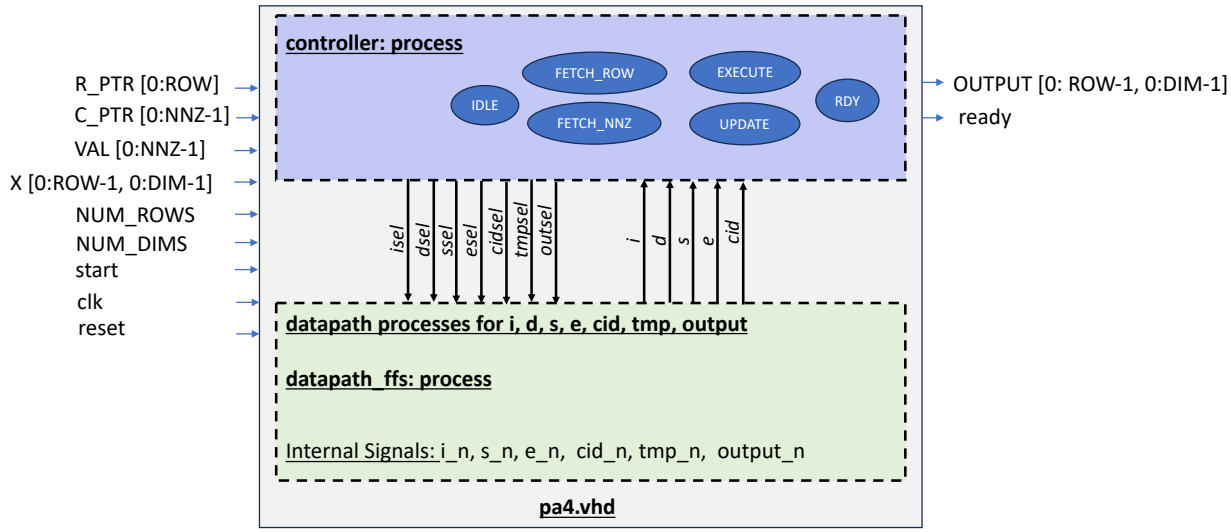
```
ROWS ← # of rows in the sparse matrix, e.g., 4 for a 4×4 matrix
DIMS ← # of dimensions in the dense matrices, e.g., 2 for a 2x4 matrix
NNZ ← Number of nonzeros in the sparse matrix, e.g., 8 in our example
Row Pointer [0: ROWS] ← Sparse matrix row pointer array
Column Pointer [0: NNZ-1] ← Sparse matrix column pointer array
Nonzero Values [0: NNZ-1] ← Sparse matrix values array
X [0: ROWS-1, 0: DIMS-1] ← Dense input matrix
OUT [0: ROWS-1, 0: DIMS-1] ← Dense output matrix
for i = 0 to ROWS-1 do
   for d = 0 to DIMS-1 do
      tmp = 0;
      s = Row Pointer [i];
      e = Row Pointer [i+1];
      for j = s to e-1 do
            cid = Column Pointer [j];
            tmp += Nonzero Values [j] * X [cid][d];
      OUT [i][d] = tmp;
```

The first for loop iterates over all rows in the sparse matrix. The second for loop iterates over all the dimensions of the dense input matrix. This way the inner product SpMM is computed for each row of sparse matrix by iterating over all dimensions (columns) of the dense matrix. The *tmp* variable is used to compute the dot product accumulations and is reset to ZERO before starting the computations for each row of A and a dim (column) of X. The *s* and *e* variables are used to compute the number of nonzeros for the row being processed in the sparse matrix. The third for loop iterates over all nonzeros of the row using *s* as the starting index and *e-1* as the last index for the Column Pointer and the Nonzero Values arrays. For each nonzero, j index is used to compute the row index (cid) for the dense input matrix and the corresponding dim (column indexed by d) is read from the dense input matrix X. The dot product is then performed with the corresponding nonzero value in the sparse matrix A. The dot products for all nonzeros of the row of A and the dim (column) of X are accumulated in the *tmp* variable. When the third loop terminates, the tmp value is written to the output matrix (OUT) at the row index corresponding to the row of the sparse matrix and the dim index corresponding to the column of X being processed. The second loop terminates when all dims (columns) of X have completed their computations for a given row of A. The first loop terminates when all rows of the sparse matrix A have completed their computations, and all output matrix elements are updated. The output matrix, OUT contains the SpMM output.

# SpMM Datapath and Controller Design

The processes to implement the controller and datapath components are defined in pa4.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 various array and matrix types for the design. Note that DIM, ROW and NNZ are constant parameters set to 4, 8 and 10 respectively. These determine the maximum number of dimensions (columns) allowed for the test bench to use for the dense input matrix, as well as rows and nonzeros allowed for the test bench to use for the sparse input matrix for testing the SpMM design. In any given testcase, the input sparse matrix may use up to ROW number of rows and NNZ number of nonzeros, and DIM number of columns for the dense input matrix. You are not allowed to modify smtest.vhd or the arrays.vhd files. However, you will modify the pa4.vhd to implement your SpMM design.

The figure below shows the input/output interface for the pa4.vhd that is given to you. All processes can read the input signal, and write the output signal as required for the SpMM functionality. Furthermore, you are given a fixed number of state types that you must use to design the SpMM. The synchronous reset logic is already given to you in pa4.vhd for the controller and datapath_ffs processes. It initializes the relevant state of the controller and datapath design components, and sets the ready output to 0 and initial state to IDLE. The input arrays, matrices, NUM_ROWS, and NUM_DIMS received from the test bench are accessible to the design. The NUM_ROWS input identifies the number of rows of the sparse input matrix being processed for a given testcase SpMM, whereas the NUM_DIMS input identifies the number of dimensions (or columns) of the dense input matrix being processed for a given testcase. The internal signals are given, and you are not allowed to add or remove any of these internal signals. The functionality of i, d, s, e, cid, tmp, and output datapath components is intended to correspond to the SpMM pseudocode algorithm described earlier. You must implement the controller and datapth logic in the processes provided to you to complete the implementation of the SpMM design in pa4.vhd.



## Design Verification

We have provided a fully functional smtest.vhd that initializes four different SpMM inputs. Note that the initialization of the input matrices and arrays as well as the expected output (cmpOut) is given in smtest.vhd for each testcase. For each testcase, the NUM_ROWS identifies how many rows are processed for the sparse input matrix, and the NUM_DIMS identifies how many dimensions (or columns) are processed for the dense input matrix. After initialization and reset, each testcase asserts the start signal for one cycle and then waits for the ready signal from the design to check and compare the OUTPUT of the design against the expected compOut output. This process is done sequentially for each testcase.

## Deliverables

Please submit the following:

1. Your modified pa4.vhd VHDL file.
2. Submit a single PDF with:
    a. Your pa4,vhd code and a state machine diagram with explanation of each state inputs, outputs, and transitions. Also provide a separate datapath logic diagram for each component.
    b. A screenshot of the following: Your output waveform that fully captures one iteration of all four testcases. Note that the testbench will repeat if the simulation continues past the completion of the four testcases. The clk, reset, start, ready, state, i, d, s, e, cid, tmp and the OUTPUT signals should be clearly visible.
3. In case your design is not fully functional, you will need to schedule a code review with the TA.