NYCU-ECE DCS-2025

Final Project

Design: Transformer-based AI Accelerator

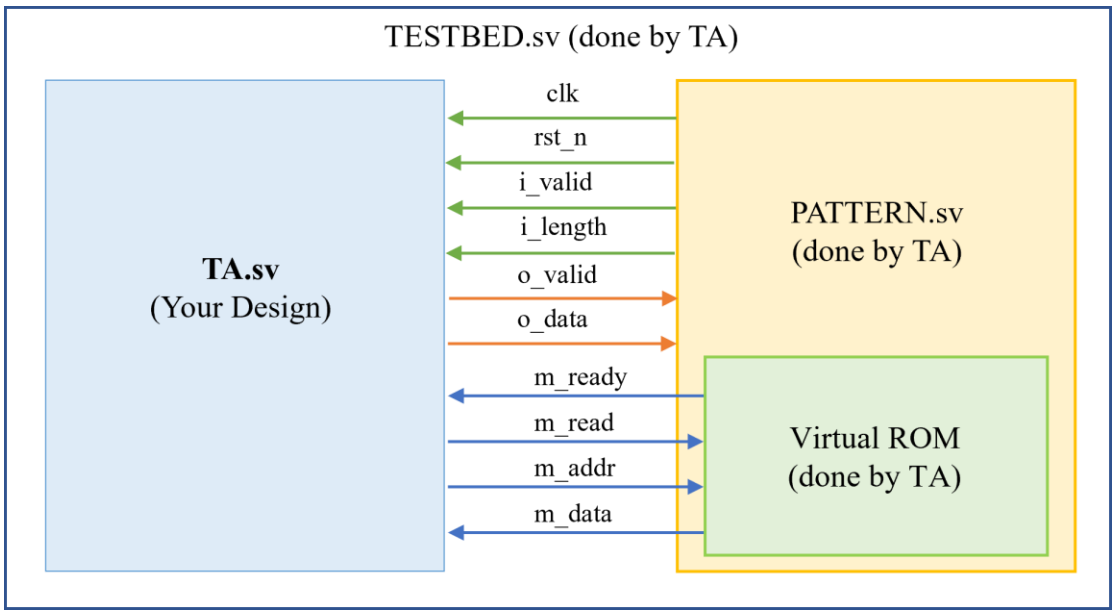
Learning Objectives

1. Master weight-, input-, and output-stationary data flows to reduce external memory access.
2. Implement a finite-state machine (FSM) to coordinate and regulate data movement.
3. Design and fine-tune a dedicated circuit for high-performance matrix multiplication.

Data Preparation

1. Extract the files needed from TA's directory:
   % tar -xvf ~dcsTA01/FP.tar
2. The extracted files contain:
   a. 00_TESTBED/
   b. 01_RTL/
   c. 02_SYN/
   d. 03_GATE/

Block Diagram

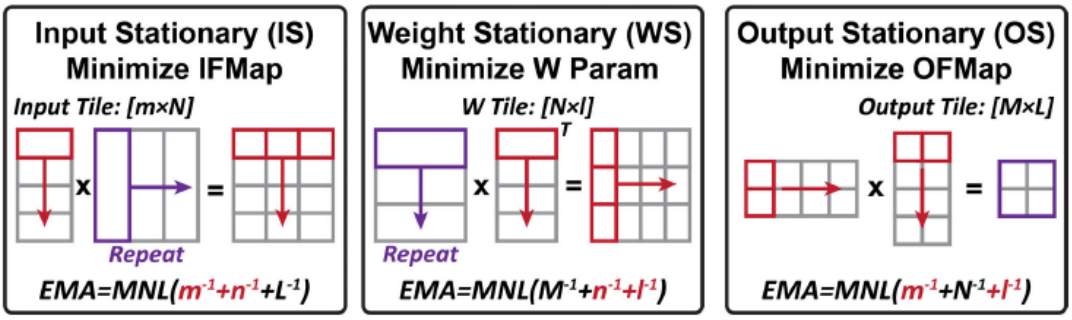


Background

The Transformer-based model has emerged as a powerful and versatile model, demonstrating robust performance across a wide range of applications, including generative AI, computer vision, and speech processing. It has become the *de facto* backbone in many modern AI systems. Unlike other types of neural networks, the Transformer's attention mechanism is significantly more complex.

In this project, we aim to design and implement an accelerator specifically tailored to enhance the efficiency of the attention operation. Moreover, to better reflect real-world design challenges, we must account for the heavy burden of data movement. Therefore, our evaluation will focus on minimizing external memory access by employing various data reuse strategies.

The following figure is the demonstration of the various schemes of data reuse strategy. (from https://ieeexplore.ieee.org/document/10530252)



Overall Architecture

The neural network architecture implemented in this project is a simplified Transformer-like structure, focusing on matrix operations and a novel normalization function. The computation flow and each component's function are described in detail below, with reference to Figure 1.

1. The formula of the following figure could be written as,
   i_token∈R^(L×D), and WQ,WK,WV∈R^(D×D)
   i_token'=CAT(i_token)
   q=i_token'×WQ,k=i_token'×WK,v=i_token'×WV
   scores=RAT(q×k^T)
   o_token=scores×v
   o_data=SLT(o_token)

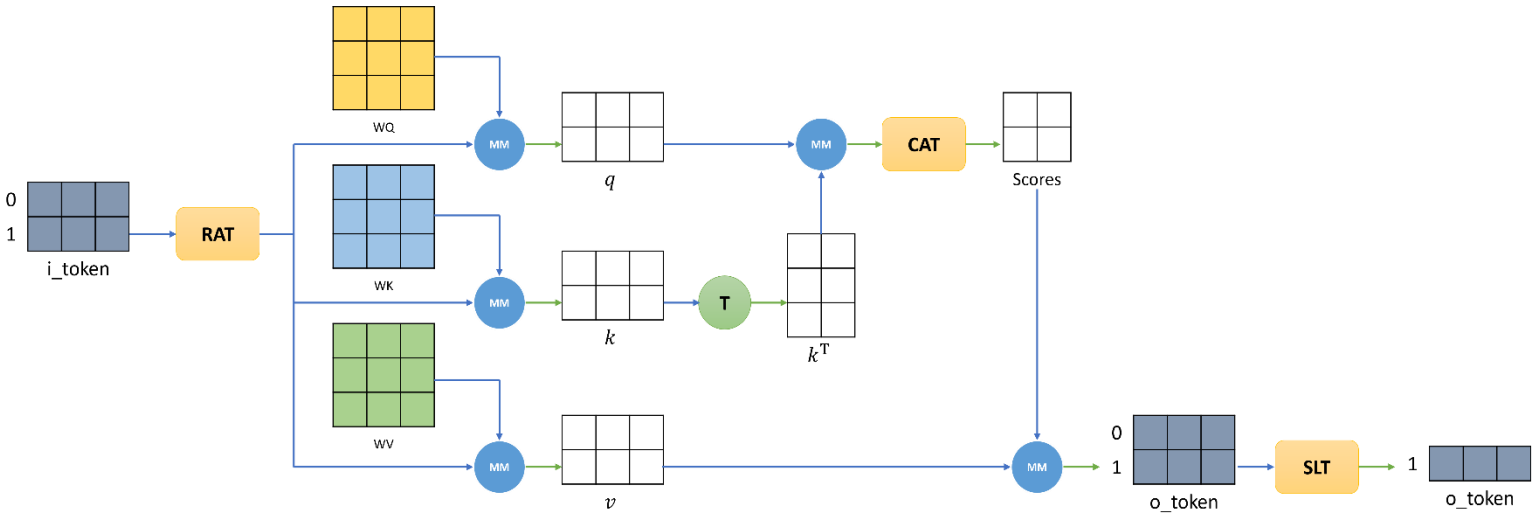


Figure 1

2. The architecture is composed of three main types of operations:
   A. Matrix Multiplication (Blue circles):
      Performs dot-product operations between two matrices. This operation forms the foundation of the architecture and has already been implemented in Homework 3 and Homework 4.
   B. Matrix Transposition (Green circles):
      Takes the matrix K, which is the result of multiplying the input matrix with the weight (key) matrix, and converts it into its transposed form.
   C. Activation Function — "Tom & Jerry" (Orange squares):
      A custom-designed activation module that applies either threshold-based filtering or token selection, depending on the chosen mode. It sparsifies the similarity matrix and emphasizes important features for further computation.

3. "Tom & Jerry" activation function:
   Note: The computation order is important—improper handling may lead to rounding issues.
   A. **RAT** (Row-wise Average Threshold) is an activation function that processes each row of a matrix individually. It first computes the average of the row (by summing all elements, then dividing by the count), which serves as the threshold. Values greater than or equal to this threshold are kept; others are set to zero.
   B. **CAT** (Column-wise Average Threshold) is similar to RAT, but it operates on each column of a matrix individually.
   C. **SLT** (Select the Last Token) is a function that selects the last token vector among all token vectors.

4. The pseudo code of the activation function, “Tom & Jerry” is shown as:

```python
import numpy as np

def RAT(i_mtx: np.ndarray) -> np.ndarray:
    """
    Parameters
    ----------
    i_mtx : np.ndarray        # shape (L, D)

    Returns
    -------
    np.ndarray                # shape (L, D), same dtype/device
    """
    row_means = i_mtx.mean(axis=1, keepdims=True)          # (L, 1)
    return np.where(i_mtx < row_means, 0, i_mtx)

def CAT(i_mtx: np.ndarray) -> np.ndarray:
    """
    Parameters
    ----------
    i_mtx : np.ndarray        # shape (L, D)

    Returns
    -------
    np.ndarray                # shape (L, D), same dtype/device
    """
    col_means = i_mtx.mean(axis=0, keepdims=True)          # (1, D)
    return np.where(i_mtx < col_means, 0, i_mtx)

def SLT(i_mtx: np.ndarray) -> np.ndarray:
    """
    Parameters
    ----------
    i_mtx : np.ndarray        # shape (L, D)

    Returns
    -------
    np.ndarray: The last token with shape (D).
    """
    return i_mtx[-1]
```

Memory Configuration

Unlike previous labs or homework assignments where the input data was directly provided by the test pattern, in this final project, all input elements are preloaded into a virtual read-only memory (V-ROM), which is prepared and provided by TA. The pattern will no longer feed the input data explicitly; instead, the accelerator must fetch the required data from the V-ROM during execution.

There are two data storage formats used to arrange matrix elements in the V-ROM: row-wise ordering and column-wise ordering. These determine the linear

storage sequence of matrix elements in memory. As shown in Figure 2(a), row-wise ordering means that elements are stored in memory one row at a time, from left to right and top to bottom. In contrast, column-wise ordering stores elements one column at a time, from top to bottom and left to right. The specific ordering used depends on the type of matrix being stored and its intended access pattern during computation.

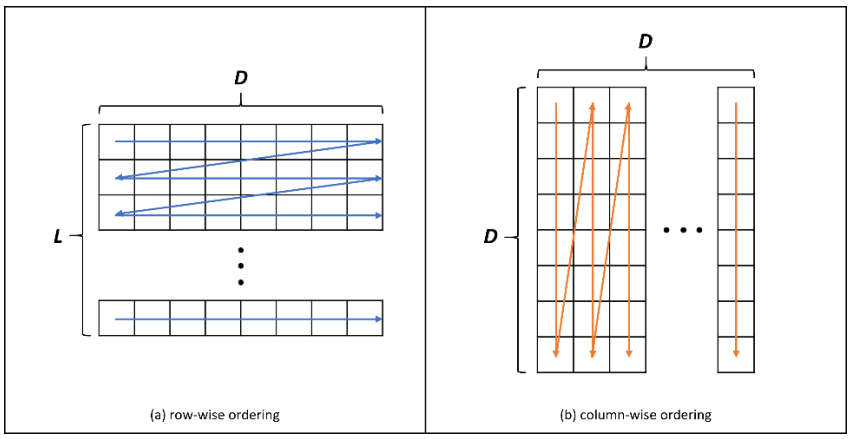


Figure 2

Each memory word in the V-ROM is 32 bits wide, containing a packed set of eight 4-bit unsigned data values. These 4-bit values can represent either input matrix elements or weights, depending on their location in memory.

The front portion of the V-ROM is reserved for storing the token input matrix, which contains the token vectors fed into the transformer-like architecture. Immediately following this region, the V-ROM holds the weights for the query (q), key (k), and value (v) matrices used in the attention mechanism. These weights are also stored using the 32-bit word format.

address	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7
0	Row 0 for i_token Matrix							
⋮								
15	Row 15 for i_token Matrix							
16	Column 0 for Q Matrix							
⋮								
23	Column 7 for Q Matrix							
24	Column 0 for K Matrix							
⋮								
31	Column 7 for K Matrix							
32	Column 0 for V Matrix							
⋮								
39	Column 7 for V Matrix							

Figure 3, example when i_length = ‘d2

Design Description

There are four possible input matrix shapes: $4 \times 8, 8 \times 8, 16 \times 8, \& 32 \times 8$, respectively. Initially, the pattern generates a one-cycle i_valid signal along with a 2-bit i_length signal, which indicates the sequence length. Specifically, when i_length is 0, 1, 2, or 3, it corresponds to sequence lengths of 4, 8, 16, and 32, respectively.

After 2 to 5 clock cycles, the pattern asserts a one-cycle m_ready signal to indicate that the V-ROM has been initialized and the accelerator can begin fetching data arbitrarily. Each time data needs to be fetched, the accelerator must assert both the m_read and 6-bit m_addr signals to access a specific memory region. The V-ROM returns a 32-bit m_data value, which is a packed set containing eight 4-bit input or weight data elements.

Following the computation process, the accelerator asserts the o_valid signal for 8 consecutive cycles, each accompanied by the corresponding 32-bit o_data output. It is important to note that o_data represents the sequential output for the final token vector, which has a shape of 1×8.

Inputs

Signal name	Bit width	Description
clk	1	Clock signal.
rst_n	1	Asynchronous active-low reset signal
i_valid	1	Pulled high when the input length is valid
i_length	1	Indicates that which shape of the input matrix
m_ready	1	Pulled high when V-ROM is ready to be fetched
m_data	32	V-ROM returns data based on the address output by your accelerator

Outputs

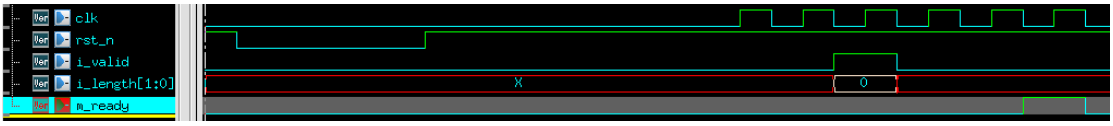
Signal name	Bit width	Description
m_read	1	Raise to high when accelerator needs to fetch memory
m_addr	6	Set to the correct address to fetch the specific memory region
o_valid	1	Pulled high when output is valid
o_data	32	The data of the output vector (unsigned integer)

Specifications

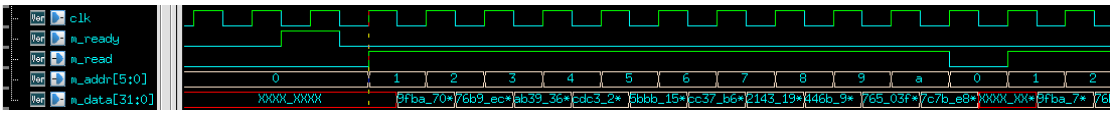
1. Top module name:**TA** (File name : **TA.sv**)
2. After asynchronous active-low reset, all output signals must be reset to zero.
3. **o_valid, i_valid, and m_ready must NOT overlap.**
4. The output must be generated within **10000 cycles** after all inputs have been provided.
5. The next pattern will be provided 2~5 cycles after o_valid is pulled down.
6. The 02_SYN result must not have any errors and must not contain any latches.
7. Note that the **maximum synthesis time** must not exceed **2 hours** (normally, this design should not take more than 2 hour).
8. The slack in the timing report must be non-negative and the result must be "Met".
9. Gate level simulation must not have any timing violation.
10. The clock period can be adjusted as needed, but must not exceed 20.0 ns.
11. Input delay = 0.5 * clock period, output delay = 0.5 * clock period
12. The design must actually implement the required functionality. Do not design specifically for the test patterns, such as determining it's the n-th pattern and directly setting the output. Such designs will be judged as fail during the demo.
13. Do not use *error*, *latch*, *congratulation* or *fail* as names for logic / wire / reg / submodule / parameter , otherwise the demo result will be fail.
    Note: * represents any symbol before or after the word. Ex: error_test is prohibited.
14. It is recommended to separate combinational blocks and sequential blocks.
15. **For loops are allowed in this homework**. If you choose to use them, ensure they are carefully implemented in your design.

Example Waveforms

Input:



Communicate with V-ROM:



Upload Files

1. Files to be submitted: SystemVerilog Code and Report, a total of two files. Please upload them to new E3.
2. Code filename: TA_dcsxxx.sv, where xxx is your server account.
3. Report filename: report_dcsxxx.pdf, where xxx is your server account.
4. 1de deadline: 6/5 23:59 / 2de deadline: 6/12 23:59.
5. Incorrect filenames will result in a 5-point deduction.

Grading Policy

1. Pass the RTL, Synthesis, and GATE level simulation: 70 %
2. Performance: 15 %

Ranking formula: $(3 \times Latency_{mem} + Latency_{exe}) \times Area^{1.5}$

$Latency_{mem} = (\#\ of\ clock\ cycles\ when\ m\_read)\ \ is\ high \times clock\ period$

$Latency_{exe} = (\#\ of\ clock\ cycles\ between\ i\_valid\ \ = 1\ to\ o\_valid\ \ to\ 1) \times clock\ period$

3. Report: 15 %

Note

Template folders and reference commands:

1. 01_RTL/ (RTL simulation) → **./01_run**
2. 02_SYN/ (synthesis) → **./01_run_dc**
3. 03_GATE/ (gate-level simulation) → **./01_run**

Please write your report concisely and to the point, not exceeding four A4 pages, and include the following content:

1. Describe your design method, including but not limited to how to accelerate (reduce critical path) or reduce area.
2. Based on the above, draw your architecture diagram (Block diagram).
3. Your thoughts, you can write about either the assignments or course content.
4. Difficulties encountered and how you solved them.
5. Have fun !!!