NYCU-ECE DCS-2025

HW03

Design: Sparse Matrix Calculator

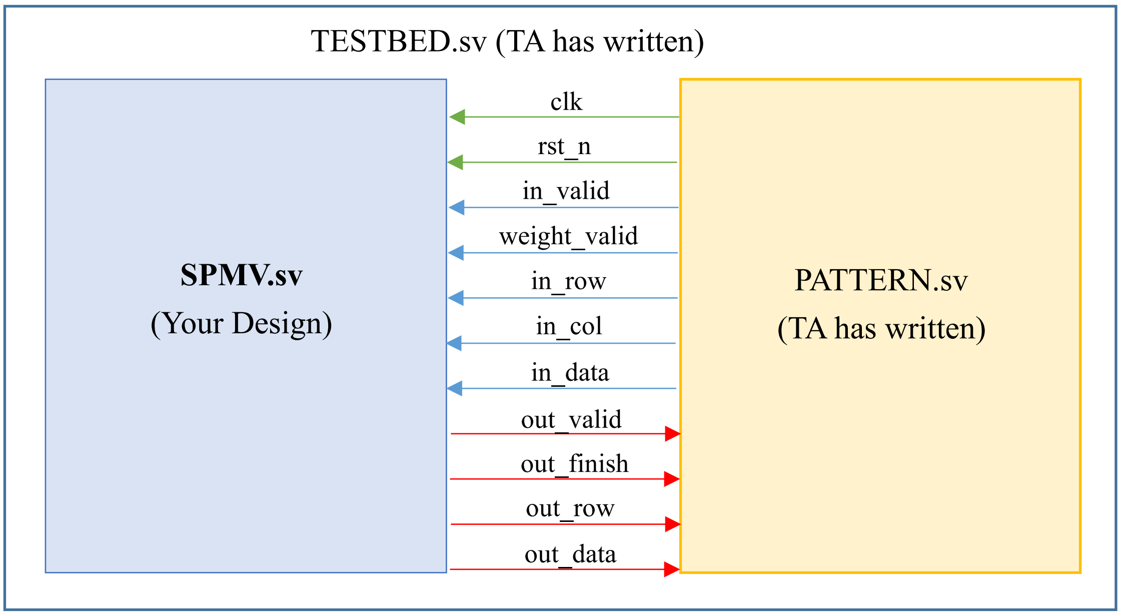
Learning Objectives

1. Separate sequential and combinational block.
2. Design a Finite State Machine (FSM) to control the data flow.
3. Handle and organize data efficiently.
4. Be familiar with shifter and counter.
5. Design and optimize a specialized circuit for sparse matrix operation.

Data Preparation

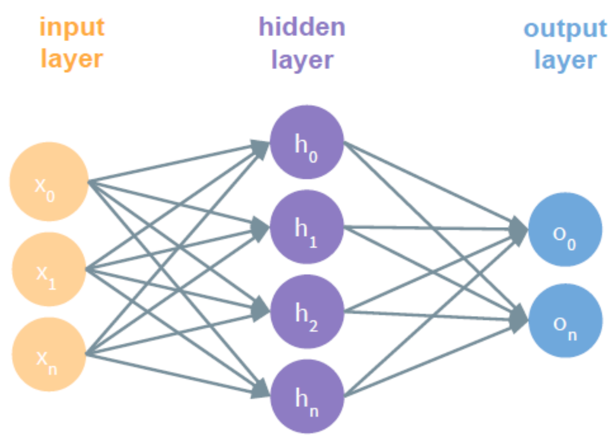
1. Extract the files needed from TA's directory:
   % tar -xvf ~dcsTA01/HW03.tar
2. The extracted files contain:
   a. 00_TESTBED/
   b. 01_RTL/
   c. 02_SYN/
   d. 03_GATE/

Block Diagram



Background

Sparse matrix multiplication is a critical operation in many fields. The term "sparse" refers to a matrix or vector in which a significant portion of the elements are zero, allowing us to skip operations on those elements. In deep learning, sparse matrix-vector operations are commonly used in feed-forward neural networks where weights and activations are pruned. Pruning redundant model weights can reduce model size and even accelerate execution speed. Below is a simple example of a feed-forward neural network.



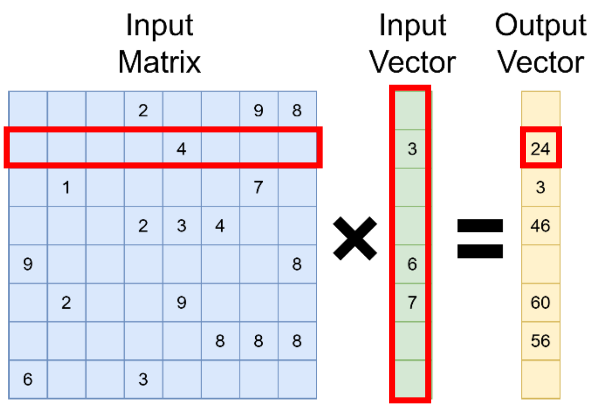
$$h_0 = w_{00}x_0 + w_{01}x_1 + w_{02}x_2$$
$$h_1 = w_{10}x_0 + w_{11}x_1 + w_{12}x_2$$
$$h_2 = w_{20}x_0 + w_{21}x_1 + w_{22}x_2$$
$$h_3 = w_{30}x_0 + w_{31}x_1 + w_{32}x_2$$

$$\begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix} = \begin{bmatrix} w_{00} & w_{01} & w_{02} \\ w_{10} & w_{11} & w_{12} \\ w_{20} & w_{21} & w_{22} \\ w_{30} & w_{31} & w_{32} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\vec{h} = W\vec{x}$$

Design Description

In this homework, you need to design a circuit to compute **sparse matrix-vector multiplication**. The matrix size is fixed at **32×32**, and both the input and output vectors have a fixed size of **32×1**. Below is a simple example where an 8×8 matrix is multiplied by an 8×1 vector. Empty squares indicate elements that are zero.



$$\begin{bmatrix} & & & 2 & & & 9 & 8 \\ & & & & 4 & & & \\ & 1 & & & & & 7 & \\ & & & 2 & 3 & 4 & & \\ 9 & & & & & & & 8 \\ & 2 & & & 9 & & & \\ & & & & & 8 & 8 & 8 \\ 6 & & & 3 & & & & \end{bmatrix} \times \begin{bmatrix} \\ 3 \\ \\ \\ 6 \\ 7 \\ \\ \\ \end{bmatrix} = \begin{bmatrix} \\ 24 \\ 3 \\ 46 \\ \\ 60 \\ 56 \\ \\ \end{bmatrix}$$

The matrix-vector multiplication can be expressed by the following equaltion:

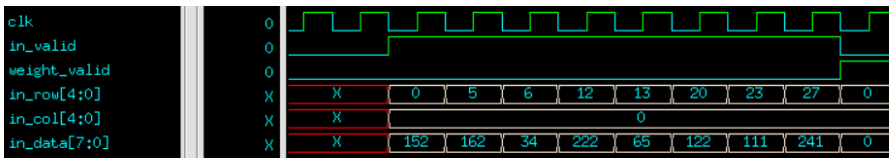
$$O_i = \sum_{j=0}^{N} M_{ij} V_j$$

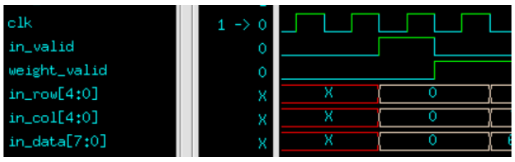
where O_i is the i-th element in the output vector, V_j is the j-th element in the input vector, M_{ij} is the element in i-th row and j-th column of the input matrix, and N is the matrix size.

INPUT:

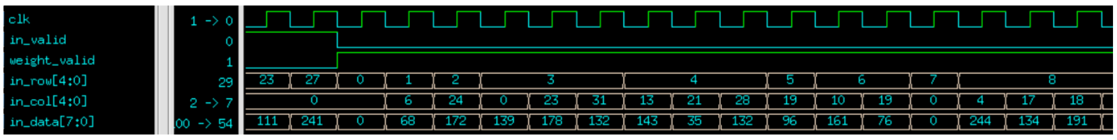
The pattern will sequentially provide the nonzero elements of the input vector, followed by the nonzero elements of the input matrix. All nonzero elements are given along with their explicit row and column indices, while zero elements are omitted.

The following two waveforms illustrate how the input vector is provided. The **"in_valid"** signal is asserted when the pattern is supplying the input vector. The **"in_row"** signal represents the row index of nonzero elements, which are provided in ascending order. The **"in_data"** signal contains the corresponding nonzero values at the indices specified by **"in_row"**. Since the input vector is one-dimensional, the **"in_col"** signal is not used and is set to zero during this stage. The second waveform demonstrates a case where all elements in the input vector are zero. In this case, the **"in_valid"** signal is asserted for one cycle, and all other input signals remain zero.



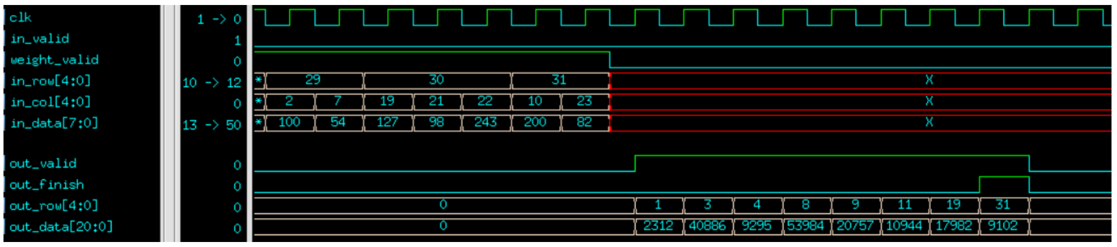


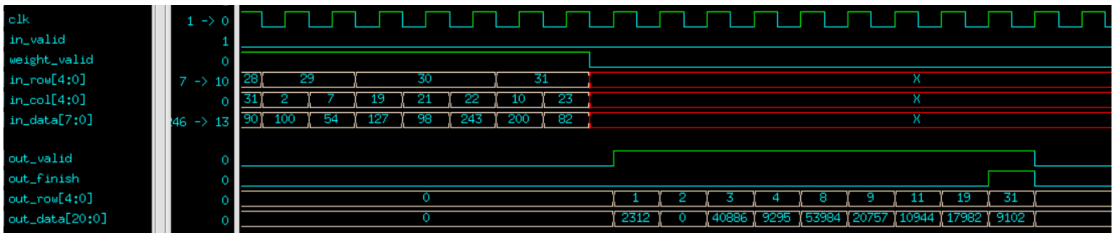
After the input vector is provided, the input matrix is immediately supplied. The following waveform illustrates how the input matrix is given. When the pattern provides the input matrix, the **"weight_valid"** signal is asserted, while the **"in_valid"** signal is deasserted. The **"in_row"** and **"in_col"** signals indicate the row and column indices of the nonzero elements in the input matrix. The indices are provided in raster-scan order (left-to-right, then top-to-bottom). Note that if an entire row consists of zeros, the pattern will still provide a zero data value for one cycle (e.g., the zeroth and seventh rows in the example waveform).



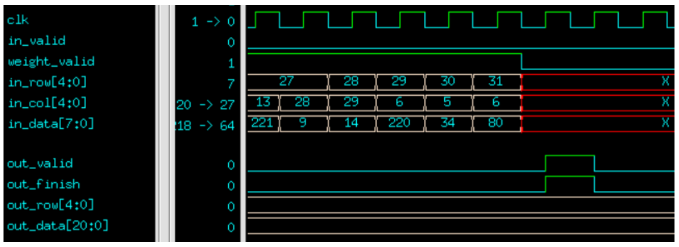
OUTPUT:

After all the elements in the input matrix have been provided, the **“weight_valid”** signal is deasserted, and your circuit can begin computing the matrix-vector multiplication. When you are ready to output the result, you should assert the **“out_valid”** signal. Note that the **“out_valid”** signal **must NOT** overlap with the **“in_valid”** or **“weight_valid”** signals. You should output all nonzero values along with their corresponding indices in the output vector. When your circuit outputs the last nonzero value, you must assert the **“out_finish”** signal for exactly one cycle. Afterward, both the **“out_valid”** and **“out_finish”** signals should be deasserted. The pattern will verify whether the computed results are correct and whether any nonzero values were omitted. Additionally, you are allowed to output zero elements in the output vector; the first and second waveforms below represent the same output vector, and both are considered valid. If the result is correct, the pattern will provide the next input within **2 to 5 cycles**.





If all elements in the output vector are zero, you should still assert **“out_valid”** and output a redundant zero value. The **“out_finish”** signal must also be asserted for exactly one cycle. The following waveform illustrates a case where all elements in the output vector are zero.



Inputs

Signal name	Bit width	Description
clk	1	Clock signal.
rst_n	1	Asynchronous active-low reset signal
in_valid	1	Pulled high when the input vector is valid
weight_valid	1	Pulled high when the input matrix is valid
in_row	5	The row index of the nonzero elements in the input vector/matrix
in_col	5	The column index of the nonzero elements in the input vector/matrix
in_data	8	The data of the input vector/matrix (unsigned integer)

Outputs

Signal name	Bit width	Description
out_valid	1	Pulled high when output is valid
out_finish	1	Pulled high when the last output is given
out_row	5	The row index of the nonzero elements in the output vector
out_data	21	The data of the output vector (unsigned integer)

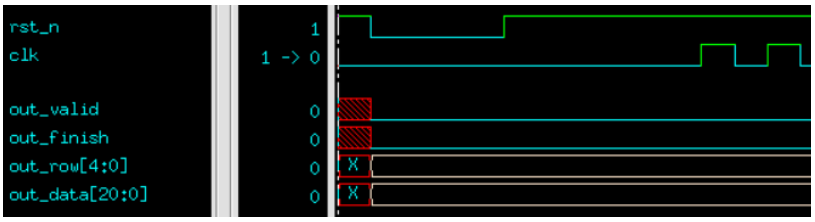
Specifications

1. Top module name: **SPMV** (File name : **SPMV.sv**)
2. After asynchronous active-low reset, all output signals must be reset to zero.
3. **out_valid, in_valid, and weight_valid must NOT overlap.**
4. The output must be generated within **5000 cycles** after all inputs have been provided.
5. out_finish must be asserted for exactly one cycle at the end of the output.
6. The next input will be provided 2~5 cycles after out_valid is pulled down.
7. The 02_SYN result must not have any errors and must not contain any latches.
8. Note that the **maximum synthesis time** must not exceed **1 hour** (normally, this design should not take more than 1 hour).
9. The slack in the timing report must be non-negative and the result must be "Met".

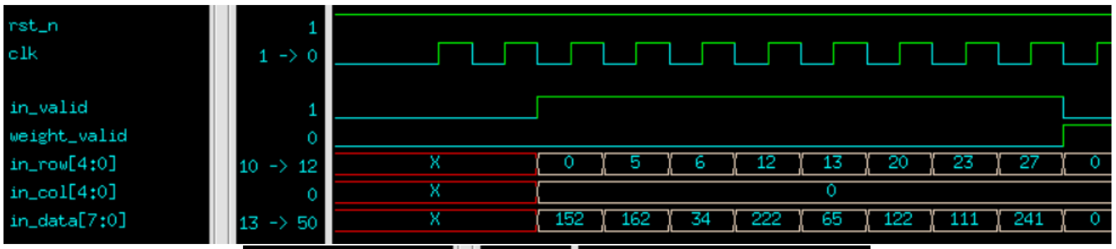
10. Gate level simulation must not have any timing violation.
11. Clock period is 8.0 ns.
12. Input delay = 0.5 * clock period, output delay = 0.5 * clock period
13. The design must actually implement the required functionality. Do not design specifically for the test patterns, such as determining it's the n-th pattern and directly setting the output. Such designs will be judged as fail during the demo.
14. 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.
15. It is recommended to separate combinational blocks and sequential blocks.
16. **For loops are allowed in this homework**. If you choose to use them, ensure they are carefully implemented in your design.

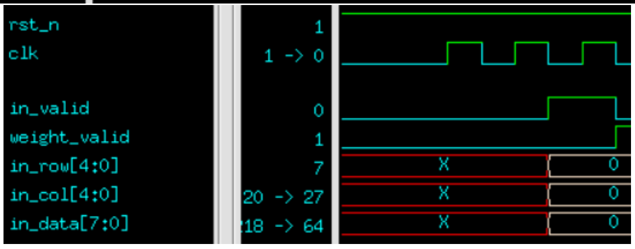
Example waveform

Reset all output after asynchronous and active-low reset signal.

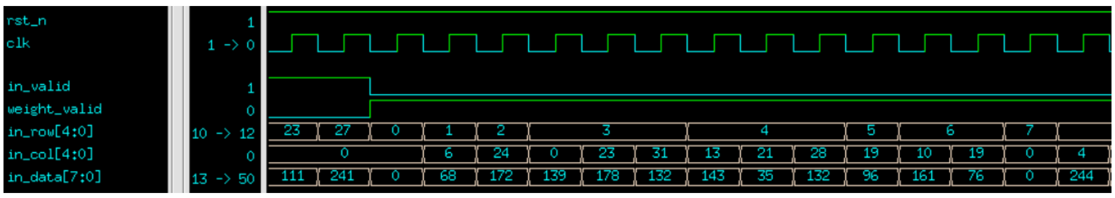


Input vector example.

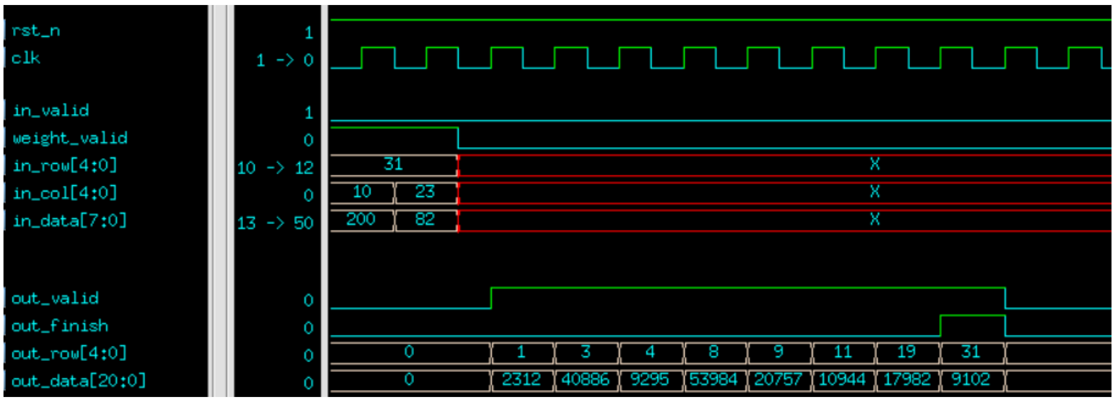


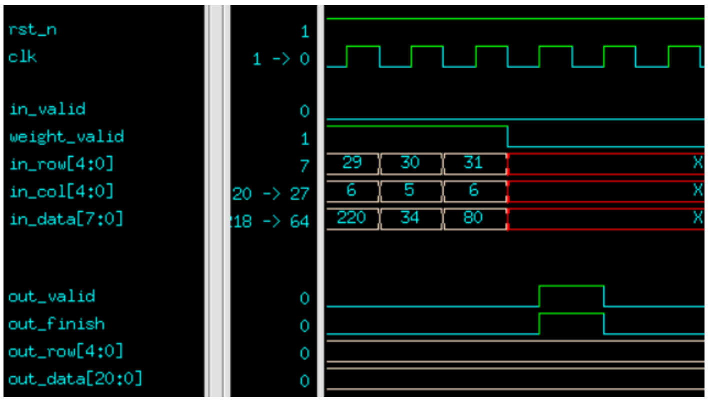


Input matrix example.

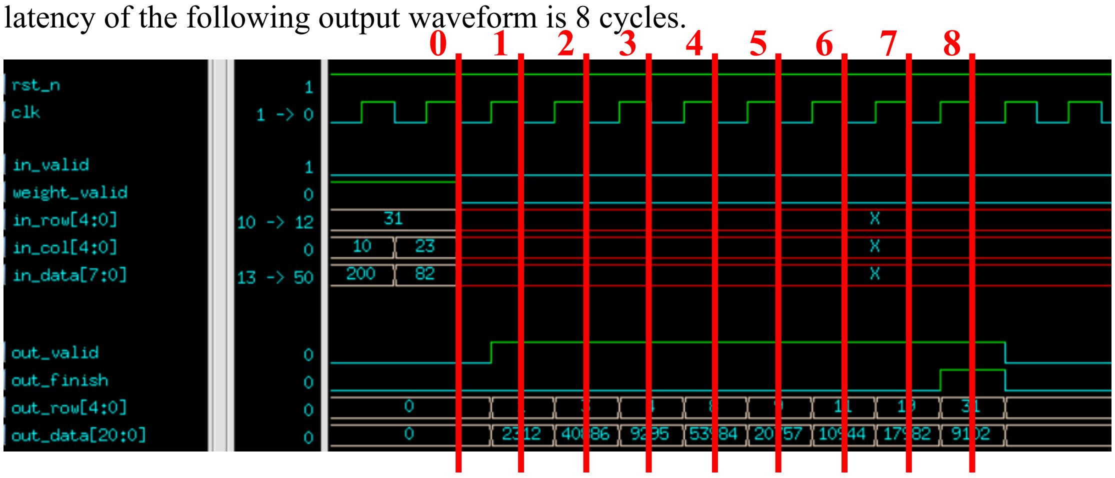


Output example.





The latency is calculated from the end of input to the end of output. For example, the latency of the following output waveform is 8 cycles.



Upload file

1. Files to be submitted: SystemVerilog Code and Report, a total of two files. Please upload them to new E3.
2. Code filename: SPMV_dcsxxx.sv, where xxx is your server account.
3. Report filename: report_dcsxxx.pdf, where xxx is your server account.
4. 1de deadline: 4/10 23:59 / 2de deadline: 4/17 23:59.
5. Incorrect filenames will result in a 5-point deduction.

Grading policy

1. Pass the RTL, Synthesis, and GATE level simulation: 60%
2. Performance: 30%

   Ranking formula: total latency * area
3. Report: 10%

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 two 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.