CASLAB

Lab1: Systolic Array Implementation
AAML 2022

TA: **Hung Yan Tai**
Email: **nobertai.c@nycu.edu.tw**

1 Introduction

Machine Learning is booming in this generation due to the progress of hardware, such as CPU, GPU, and Domain Specific Accelerator. From now on, the DSA is more important as people pursuing the performance especially in Computer Vision. For the current start-of-art ML algorithm, it has many operations(OPs) that is highly parallel and doesn't need many condition check or the condition is simple. Hence, we should start to understand how can we achieve the high performance in this lab.

In this Lab, we will implement a linear algebra hardware in simulation. The basic operation is the Matrix Multiply, we will need to use a 4x4 PE array to implement one of two different stationary that depends on you. (Hint: The weight stationary is more complicated than output stationary.)

- Output Stationary
- Weight Stationary

1.1 Getting Started

This lab will require a beginner's level of verilog.

```
> git clone git@github.com:nycu-caslab/AAML2022-Lab1.git
```

2 Background

For the TPU posted by the Google, it uses 256 by 256 PE array to create a highly parallel computing for 8bits floating. It outperforms current GPU and CPU. It mainly accelerate the Matrix Multiply, so we will need to implement the matrix multiply algorithm based on the systolic dataflow.

Let review the Matrix Multiply learning from senior high school. For three matrix A(M,K), B(K,N) and C(M,N),that is, $C = A * B$, Here is the pseudo of MM algorithm that you already know.

```
for(int k = 0; k < K; k++) {
    for(int m = 0; m < M; m++) {
        for(int n = 0; n < N; n++) {
            C[m, n] += A[m, k] * B[k, n];
        }
    }
}
```

Code 1: Matrix Multiplication in Weight Stationary

This particular dataflow is called a *Weight-stationary* dataflow, since the weight element remain constant until the next row to be computed.

In your lab report, answer the following question:

1. Write down the output stationary pseudo code with 3-level for loop which is output stationary.
2. Write down your implementation of the systolic array.
3. What the difference between the weight stationary and output stationary? (Discuss from the perspective of the Performance, Power, Area)

3 Your Assignment

The goals of this lab are to familiarize you with the concepts of dataflows in systolic array architectures. This will get you hand-on experience with dataflow routing and processing elements implementations. In this lab, you only need to construct the TPU module.

3.1 Requirement

You need to perform Matrix multiplication with one of stationary method with correct functional simulation in 4x4 Processing elements(PEs). That is this design can perform (4*K)*(K*4) 8-bit integer matrix multiplication.

1. Your design should be written in the Verilog language. There is no limitation in how you program your design.
2. Your PEs shouldn't more than 4x4, where a 2D systolic array architecture is recommended.
3. An 8-bits input data, 32-bits accumulated data design. Please be careful with the bit-width problem.
4. $(1024 + 256 * 2)$KiBytes in total of global buffer size.

4 Interface

We have three SRAM for you to access the A & B matrix and finally write back matrix C to the third SRAM. Table 2, Table 3. There are also five signals, which is used to communication with the Pattern according to the Table 1

I/O	Signal name	Bit Width	Description
Input	clk	1	The clock signal
Input	rst_n	1	The reset signal, which is active low.
Input	in_valid	1	The input is valid when in_valid is high and will only high for one cycle.
Input	K	8	The dimension K of the matrix (M,K), (K,N)
Input	M	8	The dimension M of the matrix (M,K), (K,N)
Input	N	8	The dimension N of the matrix (M,K), (K,N)
Output	busy	1	High when the design is busy. Pattern will check your answer when busy is low after every in_valid.

Tabel 1: The control signals

I/O	Signal name	Bit Width	Description
Input	wr_en	1	The write enable signal.
Input	index	20	The address of the sram to be read or write.
Input	data_in	32	The data input to write to the SRAM
Output	data_out	32	The data output from the SRAM

Tabel 2: The SRAM interface of A and B SRAM

I/O	Signal name	Bit Width	Description
Input	wr_en	1	The write enable signal.
Input	index	20	The address of the sram to be read or write.
Input	data_in	128	The data input to write to the SRAM
Output	data_out	128	The data output from the SRAM

Tabel 3: The SRAM interface of C SRAM

5 Specification

5.1 Module Specification

1. Top module: TPU (filename: TPU.V)
2. Input Pins: clk, rst_n, in_valid, K, M, N
3. Output Pins: busy, [ABC]_wr_en, [ABC]_index, [ABC]_data_in, [ABC]_data_out

5.2 Rules

1. Your TPU design should be under the top module which provided by TA.
2. Pattern module includes three global buffers prepared for your TPU. Two of the global buffers have its own read write port, 256x256x32bit=256 KiBytes size and the final one for you to write back the data has 256x256x128bit=1024 KiBytes. For the detail of the mapping of matrix into global buffer. Please refer to the Appendix 9. There are two types of mapping. Type A for matrix A, and Type B for matrix B and C.
3. At the start of the simulation, testbench will load the global buffer A & B, which assume that CPU or DMA has already prepared the data for TPU in global buffer. When signal in_valid==1, the size of the two matrices will be available for TPU (m, n, k) for only one cycle.
4. Testbench will compare your output global buffer with golden, when you finish the calculation, that is (busy==0). Then you need to wait for the next in_valid for the next test case.
5. You should implement your own data loader, process elements(PEs), and controller which schedule the data in global buffer A & B to be calculated in the systolic array.
6. You need to **set busy to high immediately** after in_valid fall from high to low.
7. Cannot modify the name of input or output port.
8. Use **asynchronous reset active low** architecture
9. All your output register **should set to zero after reset.**
10. The execution latency is limited in **1500000** cycles.
11. Don' t write Chinese comments or other language comments in the file you turned in.
12. Don't use any wire/reg/submodule/parameter name called *error*, *congratulation*, *latch* or *fail* otherwise you will fail the lab. Note: * means any char in front of or behind the word. e.g: error_note is forbidden.

5.3 Handing Rules

1. Please upload the following files on e3 platform before 23:59 on Oct 14
   - `TPU.v`.
     - Name the TPU as `TPU_StudentID.v`. Such as `TPU_20220909.v`
   - `Lab report`.
     - Name with `report_StudentID.docx`

5.4 Block Diagram

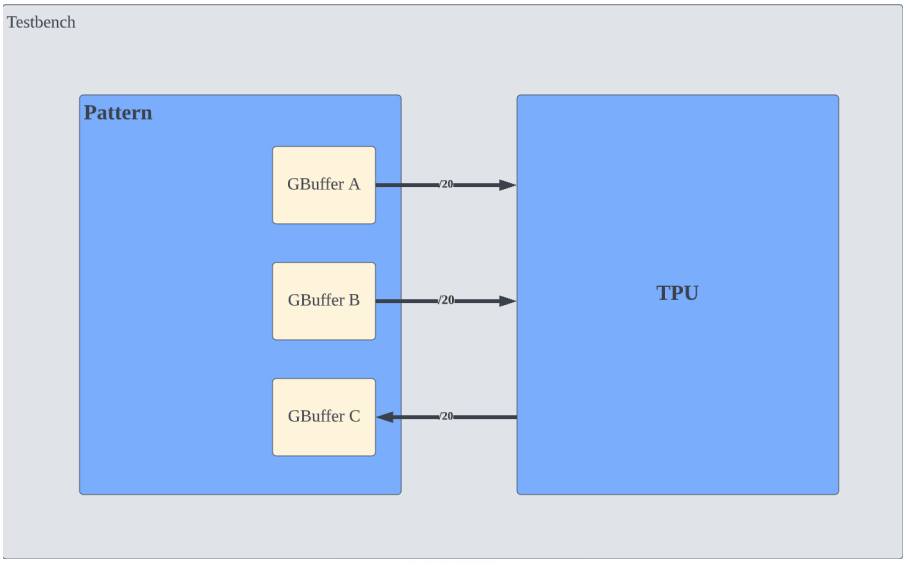

Figure 1: Top Architecture

6 Prerequisite

- Python3 with numpy library installed
- iverilog, ncverilog, vivado
- Makefile

6.1 Makefile

- make verif1
  - 100 test cases of A(2*2) * B(2*2)
- make verif2
  - 100 test cases of A(4*4)*B(4*4)
- make verif3
  - 100 test cases of A(4*K)*B(K*4), where K is dynamic
- make real
  - 100 test cases of A(M*K)*B(K*N), where $M, K, N \in [4, 256)$

7 Grading Scores (105%)

- Functional Verification (70%)
  - Designs of dataflow in TPU
  - Data reuse method
  - Pass at least verification 1 to 4, and TA will test with more test cases
- Report (10%)
  - Members' Student ID
  - TPU architecture graph
  - Explain your dataflow in TPU
  - Please describe your design as much as you can
  - Remember to answer the required question.
- Performance (20%)
  - Test with (MxK)*(KxN) case.
  - Only functional pass can have performance score.
- Extra (5%)
  - Other features
  - Good coding style
  - Plagiarizing(copy & paste) others code is prohibited, please don't try to do that, warning from TAs -100%

8 Example Waveform

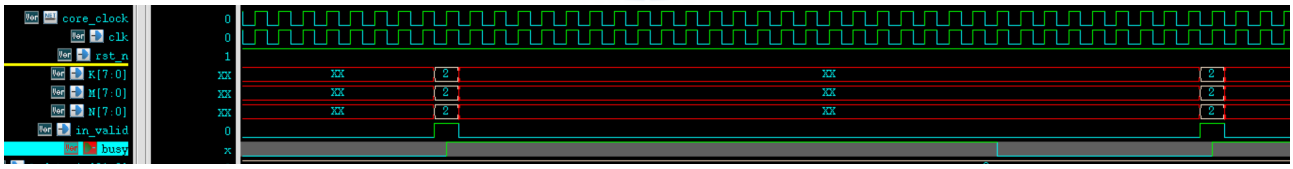

Figure 2: The example of input waveform

9 Appendix: Global Buffer Mapping

9.1 Memory Mapping - Type A (with transpose)

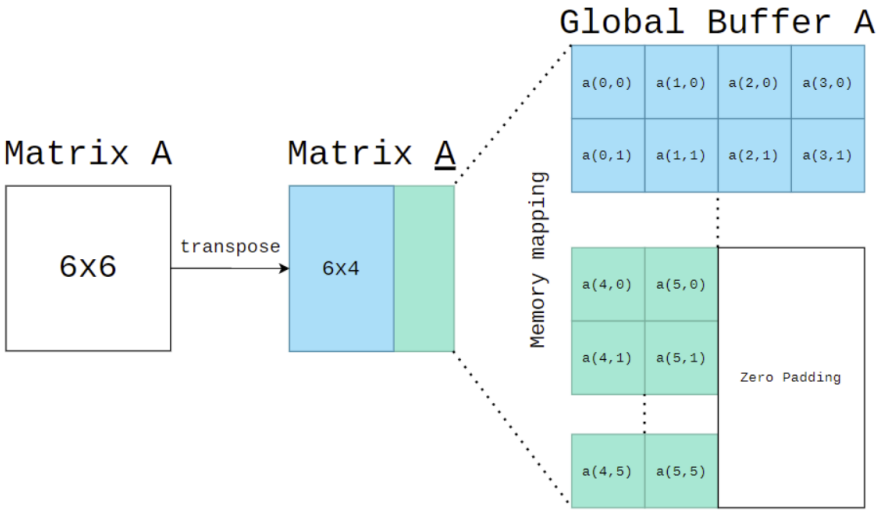

Figure 3: Matrix mapping type A

9.2 Memory Mapping - Type B (without transpose)

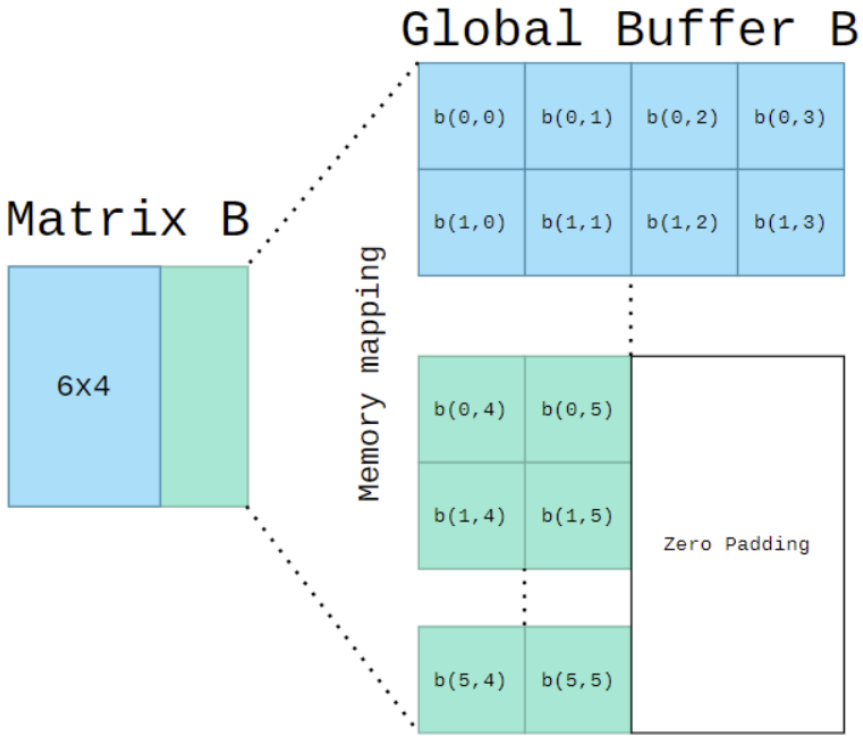

Figure 4: Matrix mapping type B