数电Lab8报告

秦禹潮 2200012730

设计目的

目标：实现512*512的矩阵乘法模块。即

$$\mathbf{Z} = \mathbf{X} \times \mathbf{Y} \tag{1}$$

已知X，Y计算Z的结果。

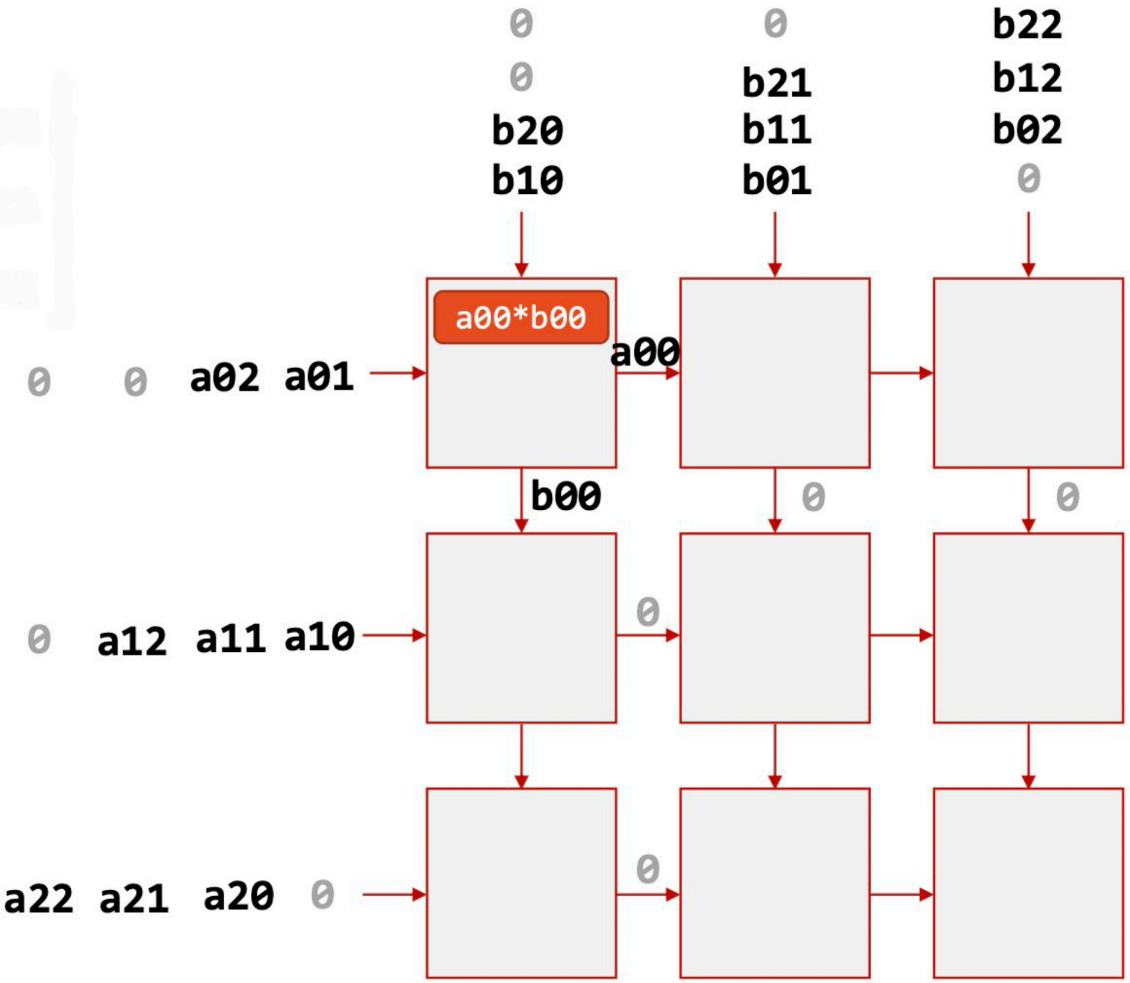
限制：对于输入，数据位宽仅仅为64bit；对于输出，数据位宽同样仅仅为64bit。

数据设置：本设计中结果计算全部采用int24计算，追求绝对精度。

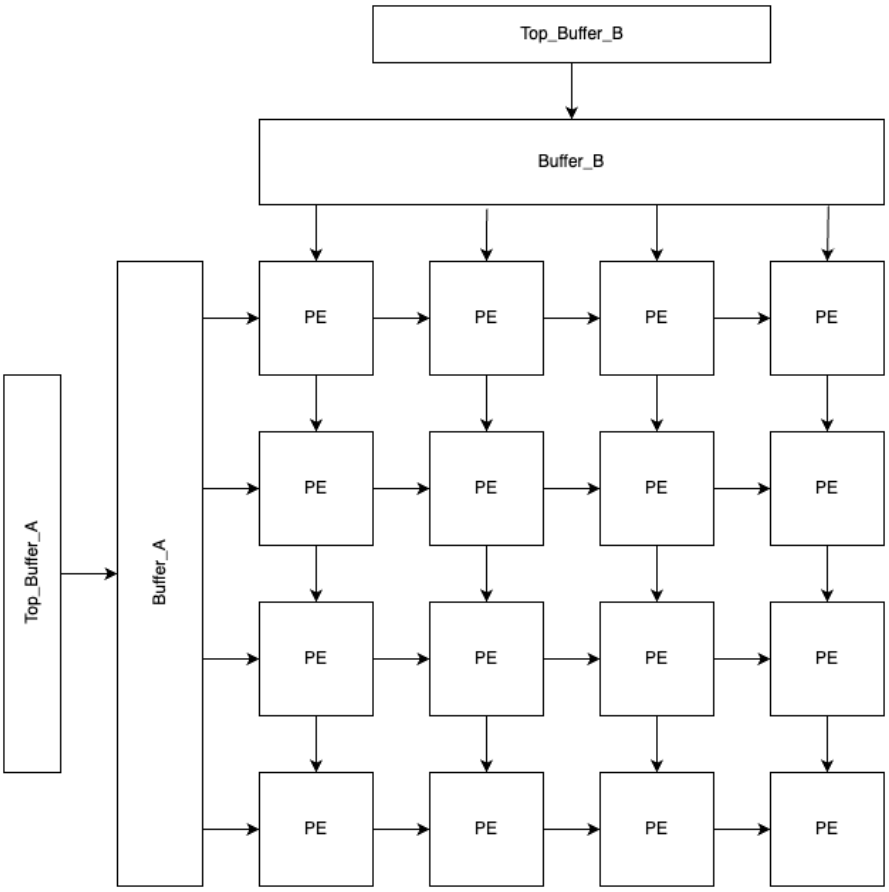
设计思路

模块架构

本次大作业采用128×128的脉动阵列PE计算模块，将512×512的矩阵块分成16份，通过分块矩阵相乘的方法得到相应的结果。其中脉动阵列采用平行四边形的输入方法，将128×512的长条数据单元和另一矩阵有相同的分块的计算单元进行相乘得到相应的计算结果。



脉动阵列

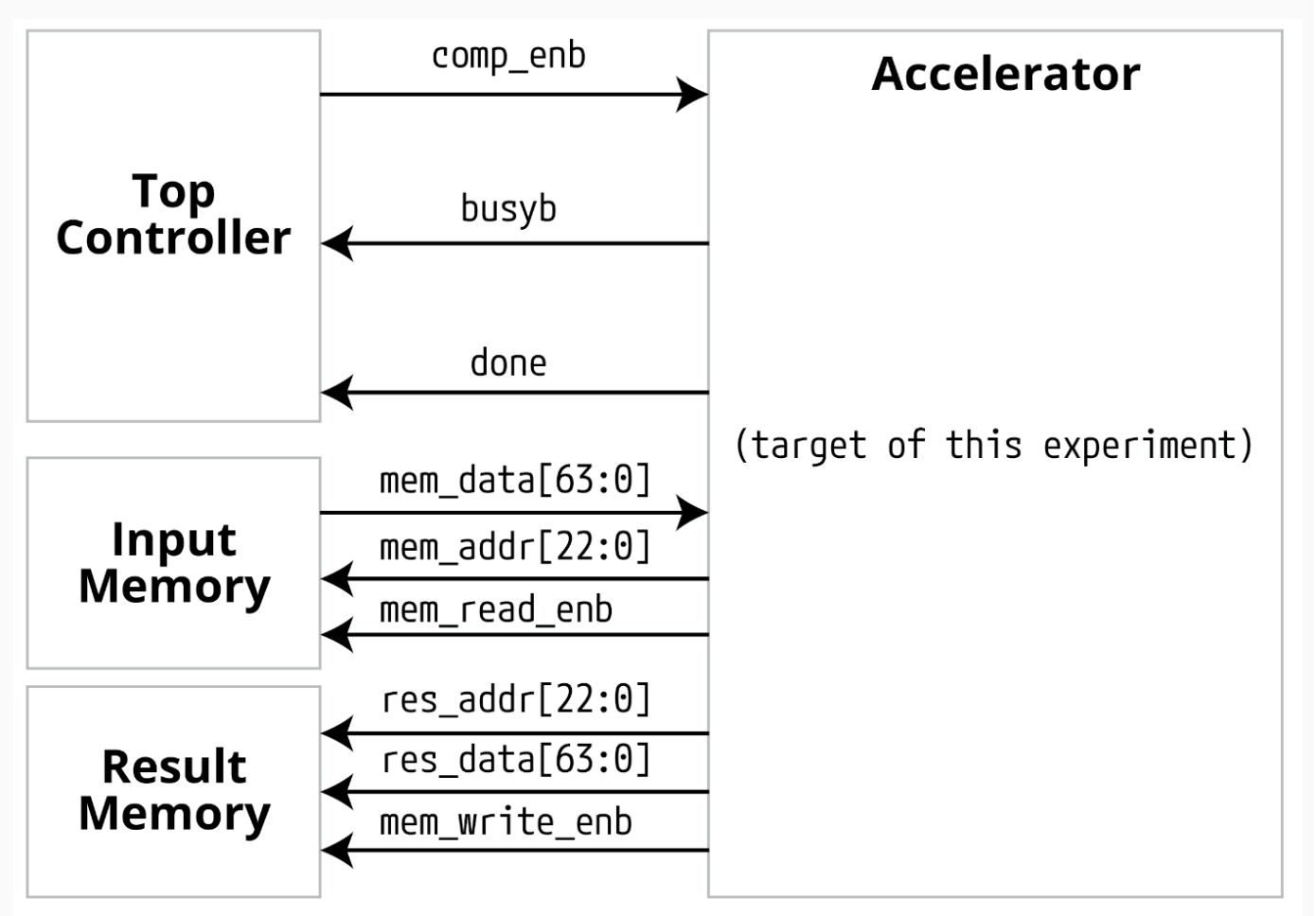


上面为一个简答的阵列设计架构。

- Top_Buffer_A&Top_Buffer_B：Accelerator和input_mem交互的部分，共有8个数据位宽，用于存储每次从外部一次性读取的8个数据。
- Buffer_A&Buffer_B：将Top_Buffer中的数据加载到Buffer中，来适应脉动阵列大小的宽度，每次Buffer中数据准备完毕了就送入脉动阵列中进行计算。
- PE：核心计算单元，将输入数据通过乘加的方式，计算相应的结果。

内外部交互

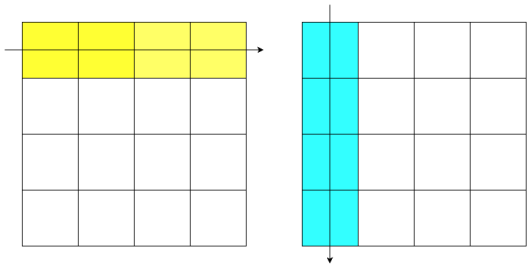
本次设计中并没有考虑用到sram，而是直接在寄存器内部实现数据存储交互，虽然耗费一定的时间，因为数据需要反复读入，但在一定程度上会减小运算的面积消耗。



在Accelerator模块中就已经包含了Systolic Array的阵列，在数据进入阵列后完成计算即可输出相应的结果。

工作原理

分块矩阵乘法



通过矩阵分割的方法，将矩阵分块，通过复用数据的方式，实现相应的计算。在分块矩阵运算中，对于结果中的Z_{ij}块矩阵，有计算公式：

$$Z_{ij} = \sum_{k=1}^{4} X_{ik} \times Y_{kj} \tag{2}$$

对于每个长条形矩阵块来说，其具有头地址：

$$Address_{head} = i \times 2^{13} \tag{3}$$

这里我们已经假定了一个地址内存储8个数据。

Accelerator

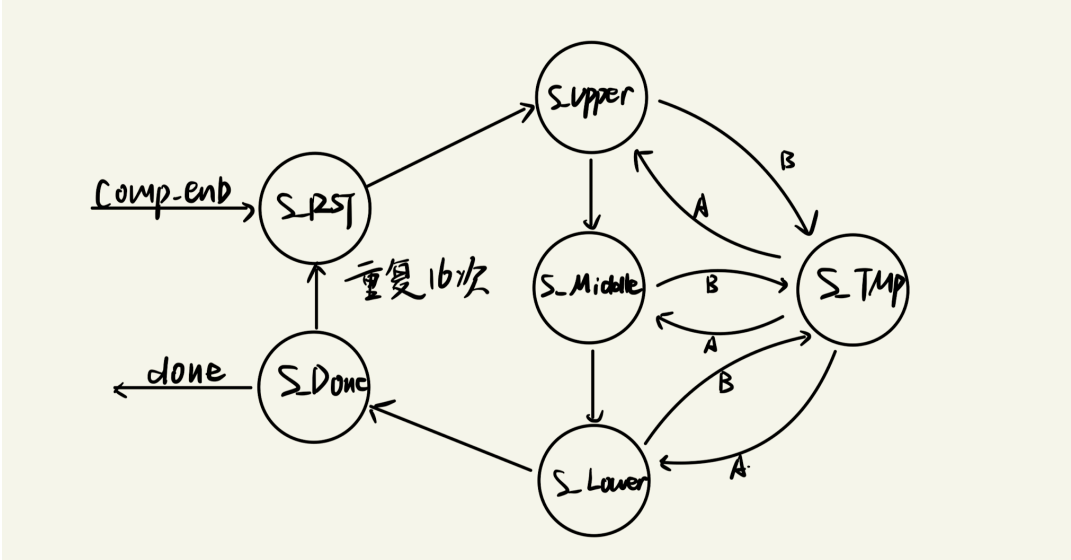
```
module accelerator #(
    parameter integer INPUT_DATA_WIDTH  = 64,    //输入的数据位宽
    parameter integer INPUT_ADDR_WIDTH  = 23,    //输入的地址位宽
    parameter integer RESULT_DATA_WIDTH = 64,    //输出的数据位宽
    parameter integer RESULT_ADDR_WIDTH = 23,    //输出的地址位宽
    parameter integer MAC_SIZE = 128,             //决定矩阵的大小
    parameter integer BIG_MAC_SIZE = 512
) (
    input clk,                                   //时钟信号
    input comp_enb,
    output [INPUT_ADDR_WIDTH-1:0] mem_addr,      //内存地址
    input [INPUT_DATA_WIDTH-1:0] mem_data,       //内存数据
    output mem_read_enb,                         //内存读使能
    output mem_write_enb,                        //内存写使能
    output [RESULT_ADDR_WIDTH-1:0] res_addr,     //结果地址
    output [RESULT_DATA_WIDTH-1:0] res_data,     //结果数据
    output busyb,                                //忙信号
    output done                                  //完成信号
);
```

接口展示如上图。

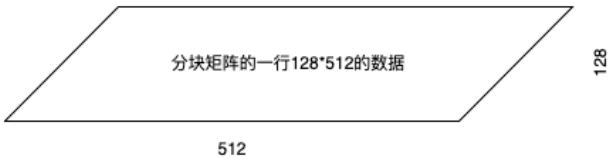
In_Contorller

```
module In_controller#(
    parameter integer DATA_WIDTH = 64,           //mem的数据位宽
    parameter integer ADDR_WIDTH = 23,           //mem的地址位宽
    parameter integer BUFFER_DATA_WIDTH = 8,     //内部buffer的数据位宽
    parameter integer TOP_BUFFER_RANGE = 8,      //top buffer的大小
    parameter integer MAC_SIZE = 32,
    parameter integer B_ADDR_HEAD = 15,
    parameter integer BIG_MAC_SIZE  = 512
)(
    input clk, comp_enb,                         //时钟和使能信号
    input [DATA_WIDTH-1:0] data_in,              //外部输入数据
    output reg done_in_enb,                      // 低有效
    output wire [ADDR_WIDTH-1:0] mem_address,    // mem的地址，和inputmem进行交互的内容
    output reg mem_read_enb,                     // mem的读使能
    output reg copy_enb                          // 复制使能
);
output reg [BUFFER_DATA_WIDTH-1:0] buffer_A [0:MAC_SIZE-1];                 // A矩阵的buffer
output reg [BUFFER_DATA_WIDTH-1:0] buffer_B [0:MAC_SIZE-1];                 // B矩阵的buffer
```

接口展示如上图。



In_Controller部分主要包含上图中的Top_Buffer和Buffer部分，输入的是从Input-mem里的数据，输出的分别为A、B矩阵的一行元素。在进入脉动阵列的过程中，矩阵被拉成了平行四边形的形状。



在其余地方通过0补齐为一个长方形的输入单元。对于两个计算的矩阵，我们分别称其为**A**、**B**

- **S_RST**：初始化输入模块，将中间的各类控制信号复位，并开始接收信息。我们规定先接收一个A的信息再接收一个B的信息。
- **S_UPPER**：处理平行四边形中的上三角矩阵，通过控制补0的方式实现数据的填充。
- **S_Middle**：处理中间长条形数据，无需补0，得到了足够的数据直接输入。
- **S_Lower**：处理末尾的下三角矩阵，仍然是通过控制补0的操作实现。

在上述状态中，对于数据的接收都只包含矩阵A的接收，对于矩阵B，通过设计单独的状态S_TMP实现。

- **S_TMP**：接收B矩阵的数据并存储。
- **S_DONE**：包含两个状态。
  - S_DONE_1：表征现在是第几个矩阵输入，并完成状态的改变，通过引入col_B和row_A两个变量实现。同时实现一个等待的状态，让输入脉动阵列的数据能够及时全部通过脉动阵列流出，得到计算结果。
  - S_DONE_2：改变A、B矩阵数据地址头，让再一次矩阵的数据读取得以进行。

在完成所有计算之后，矩阵会输出DONE信号，并停滞在S_DONE，直到有新的comp_enb信号输入则进行下一个矩阵计算。

Out_Controller

```
module Out_controller #(
    parameter integer DATA_IN_WIDTH = 24,
    parameter integer ADDR_IN_WIDTH = 18,
    parameter integer DATA_OUT_WIDTH = 64,
    parameter integer ADDR_OUT_WIDTH = 23,
    parameter integer MAC_SIZE = 128
```

```
)(
    input clk, comp_enb,
    input [DATA_IN_WIDTH-1:0] pe_result [0:MAC_SIZE-1][0:MAC_SIZE-1],
    input done_out [0:MAC_SIZE-1][0:MAC_SIZE-1],
    input done_finish,
    output reg mem_write_enb,
    output reg busyb,
    output reg done,
    output reg [ADDR_OUT_WIDTH-1:0] res_out_addr,
    output reg [DATA_OUT_WIDTH-1:0] res_data
);
```

接口展示如上。

在输出中，因为输出位宽为64bit，而我们中间的计算结果为int24，因此每个输出地址中存2个数据，计算结果之前补上8个0数据位，表示没有意义的数据位（❗不是将其变成int32的意思，在python转换结果中只取24位）。输出的过程即先将计算结果存储到一个128*128的寄存器阵列中，再将其按行逐个输出。

PE

```
module PE #(
    parameter integer DATA_WIDTH = 24,
    parameter integer BUFFER_DATA_WIDTH = 8,
    parameter integer MAC_SIZE = 512
)(
    input clk,
    input signed [BUFFER_DATA_WIDTH-1:0] up,
    input signed [BUFFER_DATA_WIDTH-1:0] left,
    input enable_b,
    input comp_enb,
    input reset_b,
    output reg signed [BUFFER_DATA_WIDTH-1:0] right,
    output reg signed [BUFFER_DATA_WIDTH-1:0] down,
    output reg signed [DATA_WIDTH-1:0] out_data
);
```

在PE模块中，我们没有使用专门设计的乘法器，而是使用Verilog中自带的`signed`类型数据进行有符号的运算。

实验结果

仿真效果

上面为Accelerator输出接口波形，可以看到输入一直在进行中，输出共有16次集体输出，由于输出相比于输入耗时较少，因此，针对输出设计诸如FIFO来充分利用输出位宽的设计并不是必要。从仿真波形图上看出，在1ns/1ns的时间规定下，计算时间为552032ns，即276016个时间周期。我们取时间周期为1ns，此时计算即过即为276016ns，约为0.276ms，比python计算的0.2s快了约720倍。

综合效果

由于综合时间花费长，较为慢，我们采用综合一个8*8的Systolic Array来得到计算结果

Total Area	Power（w）
3.847×10^6	8.85

由于采用int24的计算结果，不存在误差，因此相对误差和绝对误差均为0。

得分计算

根据燕老师博客上的计算公式可以得到计算结果：

$$Score = exp(SSE/C0) \times 功率 power(unit:mW) \times 面积 area(unit:um^2) \times (时间(us)^2) \quad (4)$$

根据计算公式得到的Score大小为：2.5935×10^{15}