Lab3实验报告

思考题

计算异常原因

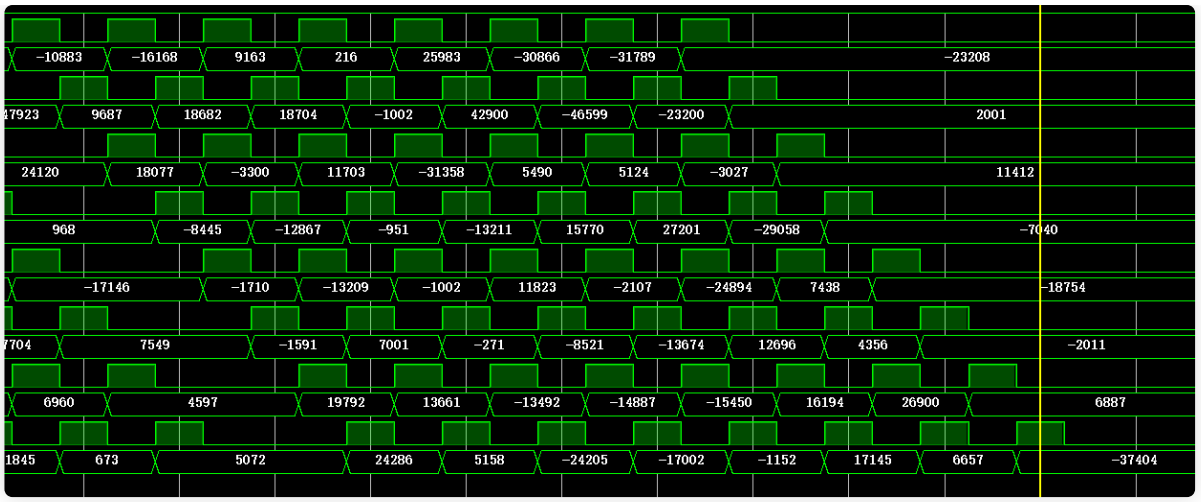
异常发生在第二个矩阵计算之中，观察可知，第一个矩阵中涉及到了负数，同时 `MAC.v` 的计算如下

```Verilog
wire signed [15:0] sf_data;
assign sf_data = $signed({8'b0, f_data});
// 本级计算
reg signed [31:0] data_reg;
always @(posedge clk or posedge rst) begin
    if (rst) begin
        data_reg <= 32'b0;
    end
    else if (valid) begin
        if (last) begin
            data_reg <= 32'b0;
        end
        else begin
            data_reg <= data_reg + $signed(w_data)*$signed(sf_data);
        end
    end
    else begin
        data_reg <= data_reg;
    end
end
```

8 位的操作数乘法运算，可能出现 16 位的结果，而零拓展只适用于无符号乘法，而第二个矩阵中出现的第一个矩阵中出现了负数，这就导致了运算错误的出现。只需要将其修改：

```Verilog
assign sf_data = $signed({{8{f_data[7]}}, f_data});
```

即将f_data的扩展方式由0扩展改为符号扩展，即可得到正确的答案，对应的波形图如下



对MAC.v进行修改

没有必要修改，因为根据题义描述，这个模块将被用于全连接层，feature 是 uint8，weight 是 int8。全连接层的计算方式为 F·W 。从上面的分析可以看出，这个模块未修改前对于 W 中的负数分量就可以正确处理，只是没有办法处理 F 中的负数分量，但是由输入格式可知，F 的分量是 `uint8`，其范围是 `0 ~ 255` 所以不会出现负数情况，所以是没有必要修改的。

MAC设计文档

设计功能

`Multiply_8×8` 模块接受一个 `8 x n` 的分量为 `unit8` 的Feature矩阵和一个 `n x 8` 的份量为 `int8` 的Weight矩阵，经过一定周期的计算输出 `FW` 的计算结果。

`MAC` 模块将输入的uint8的值和int8的值相乘，并于保存在寄存器的值进行相加求和，存回到寄存器中。同时 `MAC` 回传递输入数据和计算结果。

接口说明

`Multiply_8×8` 模块：

名称	位宽	符号	方向	含义
clk	1	unsigned	input	时钟信号
rst	1	unsigned	input	复位信号
fvalid0	1	unsigned	input	F 矩阵的第 0 行输入是否有效
fdata0	8	unsigned	input	F 矩阵的第 0 行输入数据
fvalid1	1	unsigned	input	F 矩阵的第 1 行输入是否有效
fdata1	8	unsigned	input	F 矩阵的第 1 行输入数据
fvalid2	1	unsigned	input	F 矩阵的第 2 行输入是否有效
fdata2	8	unsigned	input	F 矩阵的第 2 行输入数据
fvalid3	1	unsigned	input	F 矩阵的第 3 行输入是否有效
fdata3	8	unsigned	input	F 矩阵的第 3 行输入数据
fvalid4	1	unsigned	input	F 矩阵的第 4 行输入是否有效
fdata4	8	unsigned	input	F 矩阵的第 4 行输入数据
fvalid5	1	unsigned	input	F 矩阵的第 5 行输入是否有效
fdata5	8	unsigned	input	F 矩阵的第 5 行输入数据
fvalid6	1	unsigned	input	F 矩阵的第 6 行输入是否有效
fdata6	8	unsigned	input	F 矩阵的第 6 行输入数据
fvalid7	1	unsigned	input	F 矩阵的第 7 行输入是否有效
fdata7	8	unsigned	input	F 矩阵的第 7 行输入数据
wvalid0	1	unsigned	input	W 矩阵的第 0 列输入是否有效
wdata0	8	signed	input	W 矩阵的第 0 列输入数据
wvalid1	1	unsigned	input	W 矩阵的第 1 列输入是否有效
wdata1	8	signed	input	W 矩阵的第 1 列输入数据
wvalid2	1	unsigned	input	W 矩阵的第 2 列输入是否有效
wdata2	8	signed	input	W 矩阵的第 2 列输入数据
wvalid3	1	unsigned	input	W 矩阵的第 3 列输入是否有效
wdata3	8	signed	input	W 矩阵的第 3 列输入数据
wvalid4	1	unsigned	input	W 矩阵的第 4 列输入是否有效

名称	位宽	符号	方向	含义
wdata4	8	signed	input	W 矩阵的第 4 列输入数据
wvalid5	1	unsigned	input	W 矩阵的第 5 列输入是否有效
wdata5	8	signed	input	W 矩阵的第 5 列输入数据
wvalid6	1	unsigned	input	W 矩阵的第 6 列输入是否有效
wdata6	8	signed	input	W 矩阵的第 6 列输入数据
wvalid7	1	unsigned	input	W 矩阵的第 7 列输入是否有效
wdata7	8	signed	input	W 矩阵的第 7 列输入数据
num_valid_ori	1	unsigned	input	W 矩阵的列数（同时也是 F 的行数）是否有效
num_ori	32	unsigned	input	W 矩阵的列数（同时也是 F 的行数）
valid_o0	1	unsigned	output	结果矩阵的第 0 行是否有效
data_o0	32	signed	output	结果矩阵的第 0 行输出数据
valid_o1	1	unsigned	output	结果矩阵的第 1 行是否有效
data_o1	32	signed	output	结果矩阵的第 1 行输出数据
valid_o2	1	unsigned	output	结果矩阵的第 2 行是否有效
data_o2	32	signed	output	结果矩阵的第 2 行输出数据
valid_o3	1	unsigned	output	结果矩阵的第 3 行是否有效
data_o3	32	signed	output	结果矩阵的第 3 行输出数据
valid_o4	1	unsigned	output	结果矩阵的第 4 行是否有效
data_o4	32	signed	output	结果矩阵的第 4 行输出数据
valid_o5	1	unsigned	output	结果矩阵的第 5 行是否有效
data_o5	32	signed	output	结果矩阵的第 5 行输出数据
valid_o6	1	unsigned	output	结果矩阵的第 6 行是否有效
data_o6	32	signed	output	结果矩阵的第 6 行输出数据
valid_o7	1	unsigned	output	结果矩阵的第 7 行是否有效
data_o7	32	signed	output	结果矩阵的第 7 行输出数据

`MAC` 模块：

名称	位宽	符号	方向	含义
clk	1	unsigned	input	时钟信号
rst	1	unsigned	input	复位信号
num_valid	1	unsigned	input	乘累加长度 `num` 是否有效
num	32	unsigned	input	乘累加长度
num_r_valid	1	unsigned	output	传播到下一 MAC 的乘累加长度是否有效
num_r	32	unsigned	output	传播到下一 MAC 的乘累加长度
w_valid	1	unsigned	input	W 矩阵的一个分量是否有效
w	8	signed	input	W 矩阵的一个分量
w_r_valid	1	unsigned	output	传播到下一 MAC 的 W 矩阵的一个分量是否有效
w_r	8	signed	output	传播到下一 MAC 的 W 矩阵的一个分量长度
f_valid	1	unsigned	input	F 矩阵的一个分量是否有效
f	8	unsigned	input	F 矩阵的一个分量
f_r_valid	1	unsigned	output	传播到下一 MAC 的 F 矩阵的一个分量是否有效
f_r	8	unsigned	output	传播到下一 MAC 的 F 矩阵的一个分量长度
valid_l	1	unsigned	input	下一个 MAC 的运算结果是否有效
data_l	32	signed	input	下一个 MAC 的运算结果
valid_o	1	unsigned	output	该 MAC 的运算结果是否有效
data_o	32	signed	output	该 MAC 的运算结果

设计思想及流程描述

矩阵乘法的计算流程主要分为计算和传递两个部分。

从时间顺序来看，MAC的作用流程如下：

1. 计算累加结果：进行相乘和累加运算。
2. 运算元素传递：在第一步计算完成后，运算数被传递出去，同时接收新的操作数；这两步可以并行进行。
3. 提交运算结果：计算完成后，立即提交结果。

4. 传递运算结果：由于深处模块接收到操作数的时间较晚，MAC模块的结果传递时间必然晚于自身计算完成的时间。这一步与第三步是串行关系。

矩阵乘法可以视为一系列有规律且可复用的乘加运算。为了解决这个问题，构建模块需要按照规律逐步进行计算。

对于MAC模块，它对应结果矩阵中的一个分量。计算该分量通常需要进行n次乘加操作，通常在n个周期内完成。需要注意的是，每个分量不仅被一个MAC使用，MAC还负责传播这些分量。此外，MAC计算出的结果还需要通过MAC网络传递出去。

内部关键信号与变量

在MAC中，固定长度的累加操作通过一个计数器`num_cnt`实现。具体如下：

```Verilog
// 本级控制信号
reg [31:0] num_cnt;                         // 已经计算的轮数
wire valid = w_valid & f_valid;             // 输入数据是否有效
wire last = (num_cnt == num_r - 1'b1);      // 是否是最后一次计算

// num_cnt 用于计数
always @(posedge clk or posedge rst) begin
    // 复位
    if (rst) begin
        num_cnt <= 32'b0;
    end
    // 当 valid 信号有效时，num_cnt 会变化
    else if (valid) begin
        // 达到 num_r 后重新开始
        if (last) begin
            num_cnt <= 32'b0;
        end
        // num_cnt 递增
        else begin
            num_cnt <= num_cnt + 1'b1;
        end
    end
    // valid 信号无效，则保持原值
    else begin
        num_cnt <= num_cnt;
    end
end
```

MAC不仅需要传输自身的运算结果，还需传输下方MAC的结果，通过MUX实现：

```verilog
// 数据输出，纵向向上传播
always @(posedge clk or posedge rst) begin
    if (rst) begin
        data_o <= 32'b0;
    end
    // 最后一次的计算在这里进行
    else if (valid & last) begin
        data_o <= data_reg + $signed(w_data) * $signed(sf_data);
    end
    // 数据被更新成外来值
    else if (valid_l) begin
        data_o <= data_l;
    end
    // 保持原值
    else begin
        data_o <= data_o;
    end
end

// valid_o 是输出项，当 data_o 有效时，它就是有效的
always @(posedge clk or posedge rst) begin
    if (rst) begin
        valid_o <= 32'b0;
    end
    else begin
        valid_o <= (valid & last) | valid_l;
    end
end
```

可以看出，输出数据有两个来源：本身计算的数据`data_reg`和下方MAC的计算数据`data_l`。`data_l`的优先级更高，因为当下方数据有效时，当前MAC的计算已经完成。