AACS Project Report

023039910034 徐浏凯

实验要求

本次课程项目要求设计时序逻辑神经网络加速器，基于实验 1 中所设计的乘累加单元，实现特定功能的神经网络推理。要求所设计神经网络加速器能够在一定周期内输出结算结果，并进行 DC 综合，神经网络计算周期数不定，系统计算周期为 10ns。本项目不要求系统吞吐、功耗、能效、延时等指标。

电路设计

神经网络算法分析

要求实现神经网络整体结构如下 Fig. 1 所示。整体网络中包含两个全连接层，两个 scale 层，和一个 ReLU 激活层。其中第一个全连接层执行两个尺寸分别为[1,100]和[100,10]的矩阵乘法，第二个全连接层执行两个尺寸分别为[1,10]和[10,10]的矩阵乘法。该神经网络能够实现对手写数字的识别和分类。

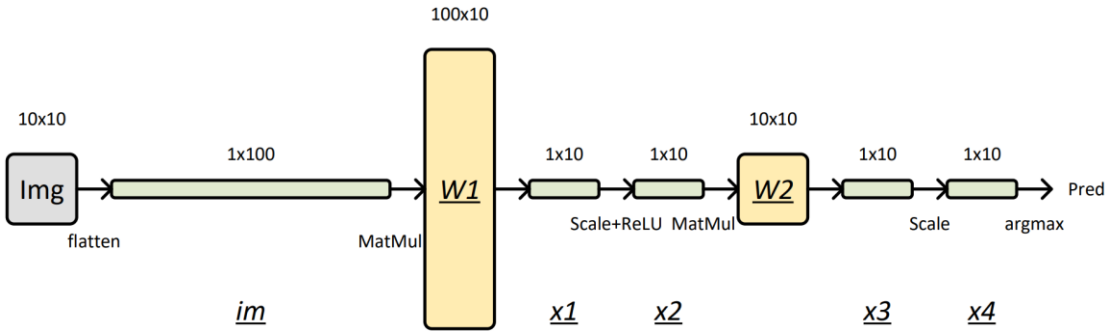

Figure 1 **神经网络整体结构图**

神经网络输入数据为 10*10 的手写数字图片，以矩阵的形式输入，其中只包含 0/1 值，有笔记的位置为 1，而无笔记的位置为 0。Fig. 2 分别展示了 python 代码中两张输入图像 6 和 7 的示例。

```
im6 = [
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
0, 0, 1, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
]
```

```
im7 = [
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 1, 1, 1, 1, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
]
```

Figure 2 **输入图像 6 和 7 示例**

神经网络详细计算流程如下。首先，输入数据在进行计算前将先展开为 1 维形式，得到[1,100]的矩阵，再输入到第一个全连接层中，得到尺寸为[1,10]的输出矩阵。该输出矩阵将经过 scale 移位和 ReLU 激活层，得到尺寸为[1,10]的输出矩阵。该矩阵将输入到第二个全连

接层，得到尺寸为[1,10]的输出矩阵，最后取得输出矩阵中最大值所对应的 index 即为神经网络的最终分类结果。综上所述，该神经网络的整体计算流程可以由 Fig.3 所示的公式进行表示。

$$\text{pred} = \text{argmax}\left(\left(\text{ReLU}\left(\left(im \times w1\right) \gg 2\right) \times w2\right) \gg 4\right)$$

Figure 3 **神经网络整体计算表示公式**

其中，神经网络中的 scale 层以数据移位的形式实现，ReLU 函数可由如 Fig. 4 所示的公式进行表示。在神经网络加速器中，所有的网络层的输入输出数据均以 8 比特有符号数的形式进行表示。

$$\text{ReLU}\left(x\right) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Figure 4 ReLU **函数表示公式**

另一方面，由于第一层全连接层的输入数据值仅为 0/1，因此对于第一个全连接层，利用实验 1 中的可重构乘累加单元，能够在不损失神经网络精度的前提下实现两张输入图像同时推理。

神经网络加速器电路设计方法

（一）神经网络加速器端口设置

加速器整体包含多个输入输出信号，整体端口设置如下图 Fig. 5 所示。其中 clk 为时钟信号，rst_n 为重置信号，start 为推理开始信号，split 为并行计算信号。Weight_we, weight_addr, weight_addr 为加速器与权值数据缓存单元的交互信号，weight_we 为权值数据读取信号，weight_addr 为权值数据读取地址，weight_data 为读取的权值数据。Input_we，input_addr，input_addr 为加速器与输入数据缓存的交互信号，input_we 为输入数据读取信号，input_addr 为输入数据读取地址，weight_data 为读取的输入数据。最后的 inference_result 信号代表神经网络推理结果，valid 表示信号表示此时推理结果有效。

```
module accelerator(
    input clk,
    input rst_n,
    input start,
    input split,

    // weight buffer read
    output  reg weight_we,
    output  reg [6:0]   weight_addr,
    input   [79:0]  weight_data,

    // input buffer read
    output  reg input_we,
    output  reg [6:0]   input_addr,
    input   [15:0]  input_data,

    // output
    output  reg valid,
    output  reg [3:0]   inference_result
);
```

Figure 5 **神经网络加速器端口设置**

（二）数据流控制

由于整体神经网络包含多个层，因此通过状态机对整体神经网络执行流程进行控制。设置 state 变量控制整体计算流程，其整体变化流程如下图 Fig. 6 所示。整体流程上，神经网络逐层执行，因此状态机也设置为顺序跳转，在一张图片执行完成后，根据预先配置的并行执行寄存器，直接跳出计算回到 IDLE 状态或继续执行第二张图片的后续计算，若继续计算则在完成第二张图片的计算后跳回至 IDLE 状态。

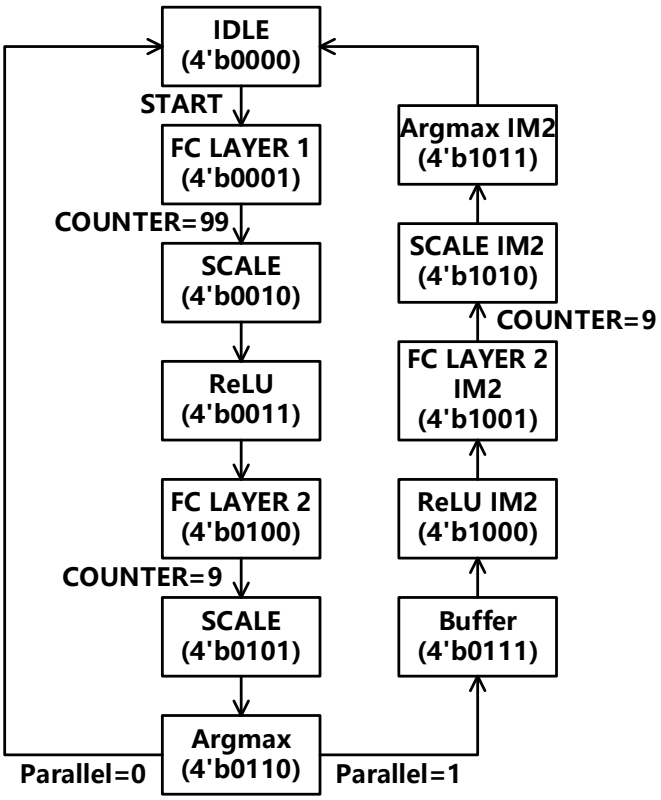

Figure 6 状态机 STATE 变量跳转图

实际 RTL 代码根据状态机 STATE 变量变化图进行撰写，由于代码段较长因此不在这里贴出完整代码。所设计加速器在接收到 start 信号后直接开始计算第一个神经网络层，并在后续过程中通过 case 语句进行判断和状态跳转。在完成神经网络整体计算流程后加速器将重新回到 4'b0000 的 IDLE 状态。

（三）加速器矩阵乘单元设计

整体加速器 PE 阵列设计如下图 Fig. 7 所示。这里采用多输入并行计算的形式，其原因在于这种形式不需要额外的寄存器用于流水打拍，整体结构上更为紧凑；同时由于整体神经网络尺寸较小，且具有两个全连接层具有相同的输出数据个数（即列数）的特点，因此在进行神经网络计算时，广播输入数据并将网络权值同时输入到各个 PE 单元即可使整个 PE 阵列以满载的形式进行工作。

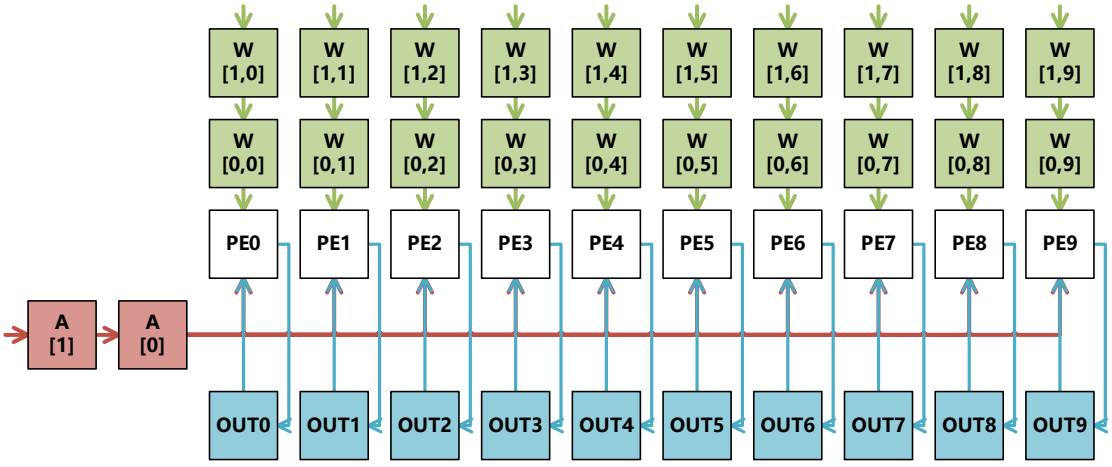

Figure 7 PE 阵列设计图

设计加速器单元如下图 Fig. 8。输入数据和权值数据会经过寄存器缓存之后进行后续的乘累加计算，避免 SRAM 单元的读取失败出现 PE 单元 X 态的输入数据，导致乘累加单元产生 X 态输出和问题输出传递的问题。

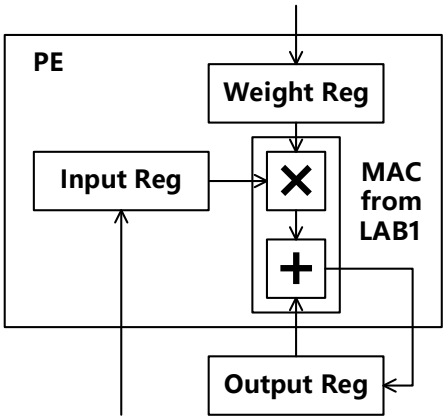

Figure 8 PE 单元结构

另一方面，缓存在 PE 单元中的数据会在新的神经网络层执行时进行刷新，防止预存的数据导致后续网络层的计算出现错误。其中 weight 部分的代码如下图 Fig. 9 所示。其中 b 为 PE 获得的 8 比特权值输入数据；working 为工作信号，代表加速器正处于工作状态；flush 为输入的数据冲刷信号，在完成当前全连接层的计算后重置 PE 单元中的数据为 0，防止后续计算错误。

```
// weight data b
always@(posedge clk or negedge rst_n) begin
    if(!rst_n == 1'b1) begin
        weight <= 8'b0;
    end
    else begin
        if(flush) begin
            weight <= 8'b0;
        end
        else if(working) begin
            weight <= b;
        end
        else begin
            weight <= weight;
        end
    end
end
```

Figure 9 PE 单元 weight 部分代码

全连接层计算过程中通过 counter 进行计数，控制需要读取的输入数据和权值数据的地址和计算周期数，防止整体计算结果出错。

（四）加速器移位/ReLU 计算设计

所设计的神经网络加速器的移位计算通过选取输出计算结果的部分比特来完成，而 ReLU 激活操作通过对输出结果的最高位判断来完成。下图 Fig. 10 展示了第一个全连接层输出结果的移位 scale 和 ReLU 激活函数的具体实现方式。

```
4'b0010:    begin
                // scale
                mid_result[i_mid] <= pe_line_out[i_mid*48+9:i_mid*48+2];
            end
4'b0011:    begin
                // relu
                if (mid_result[i_mid][7] == 1) begin
                    mid_result[i_mid] <= 8'b0;
                end
                else begin
                    mid_result[i_mid] <= mid_result[i_mid];
                end
            end
```

Figure 10 所设计加速器的移位 scale 和 ReLU 实现方式

（四）结果输出

在输出神经网络推理结果时，将同时将 valid 信号拉高表示输出结果有效。下图 Fig. 11 展示了 valid 信号的具体输出逻辑。从图中可以看到，valid 信号仅在 state 跳转到相应完成结果输出的状态后拉高一个周期（不会反复拉高的原因是 state 信号也只会在该状态停留一个周期，若停留时间较长则 valid 信号会被反复拉高）。

```
// valid
always@(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        valid <= 1'b0;
    end
    else begin
        if(start) begin
            valid <= 1'b0;
        end
        else if(valid == 1'b1) begin
            valid <= 1'b0;
        end
        else if(state == 4'b0110 | state == 4'b1011) begin
            valid <= 1'b1;
        end
        else begin
            valid <= valid;
        end
    end
end
```

Figure 11 所设计加速器的 valid 信号控制逻辑

（五）多精度计算设计

由于第一层全连接层具有较长的计算延时，占到整体神经网络计算延时的 85%以上；同时由于该层的输入 input 数据具有 0/1 的分布特征，因此可以采用多精度的 PE 阵列计算该层，实现两张输入图片的同时推理，通过这种方式能够有效提高神经网络加速器吞吐量和计算能效。

为支持多精度计算，设置了一个并行计算位，在整体神经网络开始计算前记录外部输入的计算精度要求，是否同时执行两张图片推理。如下图 Fig. 12 所示，parallel 寄存器单元记录了外部输入的多精度要求，并始终保持该信息直到下一次输入的 start 信号同时对该位进行修改和调整。

```
// parallel
// parallel = 1 for two images processing parallelly
always@(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        parallel <= 1'b0;
    end
    else begin
        if(start) begin
            parallel <= split;
        end
        else begin
            parallel <= parallel;
        end
    end
end
```

Figure 12 神经网络加速器 parallel 寄存器设置代码

如 Fig. 6 所示，在 parallel 为 1 时，即要求两张图片同时进行第一层全连接层的计算时，所设计的神经网络加速器将同时计算两张图片推理的第一层全连接层的计算，但后续的神经网络层将先针对第一张输入图片进行计算，在完成第一张图片的计算后，再对第二张图片进行推理。因此，而后续的全连接层在计算过程中并不采用两张图片同时输入的方式，因此设置 PE 阵列并行计算信号如下图 Fig. 13 所示。

```
assign parallel_en = parallel & (state[3:1] == 3'b000);
```

Figure 13 PE 阵列并行计算控制方式

另一方面，在两张图片同时输入的情形下，需要先对第一个全连接层针对两张图片的输出进行缓存。下图 Fig. 14 展示了部分结果的缓存方式，输出乘累加结果中的较高位部分的结果将被保存到 mid_result_buffer 寄存器当中。在后续的针对第二张图片的计算过程中，缓

存在寄存器 mid_result_buffer 将在状态机控制的 buffer state 中被载入到 mid_result 寄存器中，继续进行下一步的计算，如图 Fig. 15 所示。中间缓存部分的神经网络加速器电路结构如 Fig. 16 所示。

```
// mid_result_buffer
generate
    genvar i_2mid;
    for(i_2mid=0;i_2mid<10;i_2mid=i_2mid+1) begin: mid_result_buffer_gen
        always@(posedge clk or negedge rst_n) begin
            if(!rst_n) begin
                mid_result_buffer[i_2mid] <= 8'b0;
            end
            else begin
                if(state==4'b0010) begin
                    // scale
                    mid_result_buffer[i_2mid] <= pe_line_out[i_2mid*48+33:i_2mid*48+26];
                end
                else begin
                    mid_result_buffer[i_2mid] <= mid_result_buffer[i_2mid];
                end
            end
        end
    end
endgenerate
```

Figure 14 并行图片输入时的部分结果缓存

```
4'b0111:    begin
                // buffer
                mid_result[i_mid] <= mid_result_buffer[i_mid];
            end
4'b1000:    begin
                // relu
                if (mid_result[i_mid][7] == 1) begin
                    mid_result[i_mid] <= 8'b0;
                end
                else begin
                    mid_result[i_mid] <= mid_result[i_mid];
                end
            end
4'b1010:    begin
                // scale
                mid_result[i_mid] <= pe_line_out[i_mid*48+11:i_mid*48+4];
            end
```

Figure 15 缓存结果的重新载入和后续计算

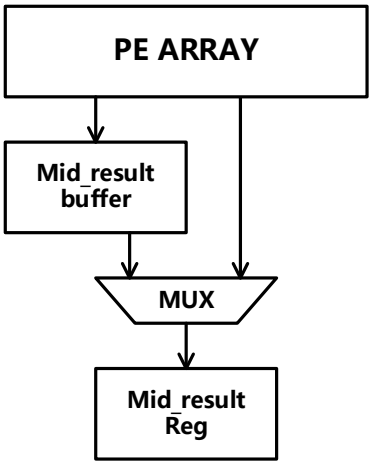

Figure 16 并行计算模式下，部分结果缓存的加速器电路结构

另外，为实现多精度计算，要求输入数据和权值数据能够有对应的值。这里首先讨论输入数据，这些数据在 buffer 中以 8 比特的形式进行存储，而需要同时读入两张输入图像时，就需要同时读入 16 比特数据，因此 input buffer 采用 16 比特位宽的存储形式，在非并行计算的情形下仅有低 8 比特保存输入数据，而在并行计算模式下低 8 比特和高 8 比特分别保存两张图片对应位置的输入数据。此时，需要取输入数据高位中的部分比特和低位中的部分比特作为输入数据，如下图 Fig. 17 所示。

```
assign in_x = (state == 4'b0001) ? {(parallel == 1'b1) ? {input_data[11:8], input_data[3:0]} : {input_data[7:0]}} : {((state == 4'b0100 | state == 4'b1001) ? {mid_result[counter]} : {8'b0})};
assign in_w = (state == 4'b0001 | state == 4'b0100 | state == 4'b1001) ? weight_data : 80'b0;
```

Figure 17 PE 阵列输入数据控制形式

同时，由于第二个全连接层需要执行两次，即读取两次权值，因此需要进行跳转回到第

二层网络权值开始的存储地址，而不能直接按顺序继续读取（出现报错，读取值为 X），或在 buffer 中存储两次相同的权值。权值读取地址代码如下图 Fig. 18 所示，在图中可以看到，在进行计算的过程中，若需要并行计算两张图片，则在第二张输入图片的第二个全连接层计算前，权值地址将被重置为 7'B1100100 的固定值。

```
// weight_addr
always@(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        weight_addr <= 7'b0;
    end
    else begin
        if(start) begin
            weight_addr <= 7'b0;
        end
        else if(state==4'b0001 | state == 4'b0011 | state == 4'b0100 | state == 4'b1000 | state == 4'b1001) begin
            weight_addr <= weight_addr + 1;
        end
        else if(state == 4'b0111) begin
            weight_addr <= 7'b1100100;
        end
        else begin
            weight_addr <= weight_addr;
        end
    end
end
```

Figure 18 **权值读取地址设置代码**

（六）其他说明

所设计加速器要求权值数据和输入数据均从地址为 0 的位置开始连续存储。权值缓存单元的宽度为 80 比特，深度为 128；输入数据缓存单元的宽度为 16 比特（支持同时存储两张图片，具体存储方式如（五）中所描述），深度为 128。

仿真结果

仿真部分以助教给出的第一版权值和输入数据作为输入，采用 im6 和 im7 输入图像进行功能测试。

（一）python 仿真

首先采用 python 仿真观察中间数据的数据结果，两张图像 im6 和 im7 的输出结果分别如下图 Fig. 19 所示。中间数据将用于最后检验输出结果正确性。

```
[-22, 22, -31, 79, 44, -50, 11, 17, -17, 14]
[0, 5, 0, 19, 11, 0, 2, 4, 0, 3]
[-26, -650, -53, -109, -5, 17, 141, -519, -174, -314]
[-2, -41, -4, -7, -1, 1, 8, -33, -11, -20]
The number in the image is predicted to be 6
```

```
[43, 47, 63, -30, 57, 22, 50, 49, -22, 22]
[10, 11, 15, 0, 14, 5, 12, 12, 0, 5]
[-692, -613, -173, -185, -451, -465, -855, 37, -332, -273]
[-44, -39, -11, -12, -29, -30, -54, 2, -21, -18]
The number in the image is predicted to be 7
```

Figure 19 python **仿真输出结果**

（二）缓存单元数据文件生成

调整缓存文件生成代码，将两张图片保存在同一个缓存单元中，图片对应的点在同一行中进行保存，代码如下图 Fig. 20 所示。另一方面，权值缓存文件不需要进行修改，依然以原始形式进行保存，每次读出权值矩阵的一行，80 个比特。输入数据和权值的 bin 文件部分如图 Fig. 21 所示。

```
def two_image_to_bin(im1,im2):
    binfile = ''
    for i in range(100):
        binfile += (int_to_bin(im1[i], 8) + '_' + int_to_bin(im2[i], 8)) + '\n'
    return binfile
```

Figure 20 **缓存单元数据文件生成代码**

```
00000000_00000000
00000000_00000000
00000000_00000000
00000000_00000000
00000000_00000001
00000000_00000001
00000001_00000001
00000000_00000001
```

```
11111111_00000000_11111111_00000001_11111111_11111111_00000000_00000000_00000001_00000001
00000000_00000001_11111111_00000000_11111111_00000001_00000000_11111111_00000000_11111111
00000000_00000001_11111100_00000010_11111110_00000000_00000001_00000001_00000010_00000000
11111111_00000001_11111111_00000001_11111111_11111101_00000001_11111110_00000010_00000001
00000010_11111111_11111011_00000010_11111111_11111101_00000001_11111111_00000010_11111110
11111111_00000010_11111011_00000000_11111110_11111011_00000001_00000001_00000010_00000010
00000000_00000000_11111110_00000001_11111111_11111101_00000001_00000000_00000011_11111111
00000000_00000001_00000000_00000010_11111111_11111101_00000011_11111111_00000001_00000001
```

Figure 21 **输入数据 bin 文件示意图**

（三）电路仿真结果——非并行模式

该模式下的电路仿真结果如下图 Fig. 22 所示。从图中可以看到，所设计的神经网络加速器能够正确输出推理结果，并在输出的同时拉高 valid 信号一个周期。同时，在完成神经网络推理计算后，加速器将跳转回到 IDLE 状态。

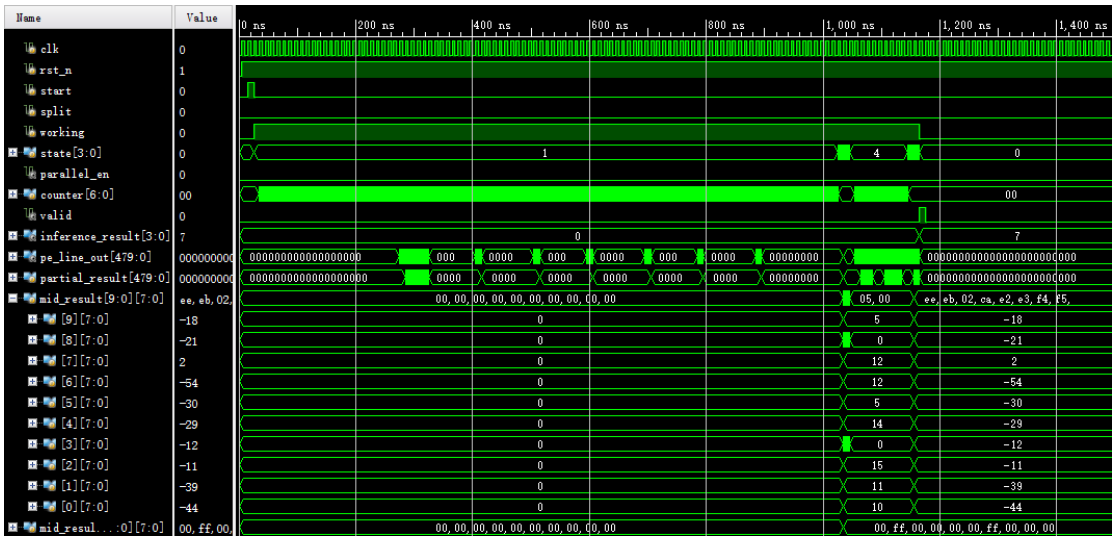

Figure 22 非并行模式下的电路仿真波形图

Fig. 23 和 Fig. 24 展示了计算过程的中间结果，两者仿真时间对应。Fig. 23 展示了第一个全连接层后的 scale、ReLU、和第二个全连接层后的 scale 结果，从图中可以看到数据结果和 Fig. 19 中 python 仿真结果能够对应。Fig. 24 展示了第一个全连接层的乘累加结果的变化值和后续的数据变化，可以看到数据结果同样可以和 Fig. 19 中的 python 仿真结果对应。

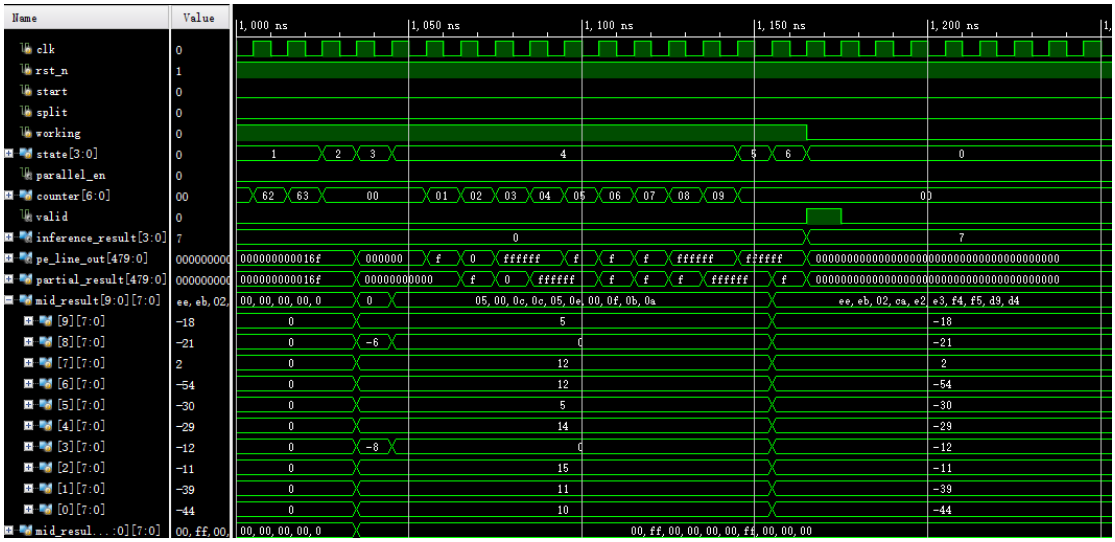

Figure 23 非并行模式下的电路仿真中间数据波形图

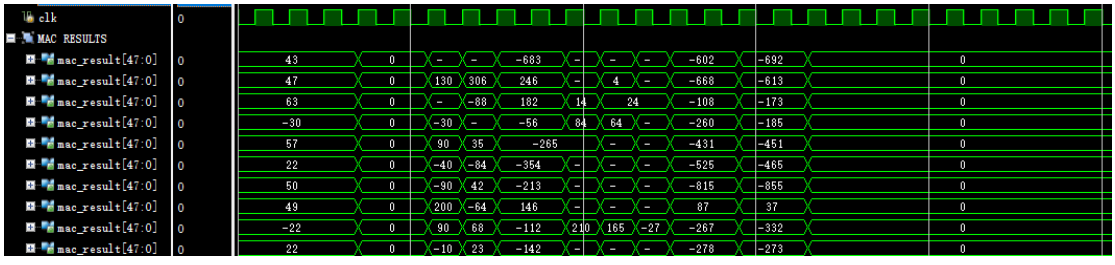

Figure 24 非并行模式下的乘累加结果变化波形图

（四）电路仿真结果——并行模式

该模式下的电路仿真结果如下图 Fig. 25 所示。从图中可以看到，所设计的神经网络加速器能够正确输出推理结果，并在输出的同时拉高 valid 信号一个周期，两个输出结果下均分别拉高了一个周期。同时，在完成神经网络推理计算后，加速器将跳转回到 IDLE 状态。

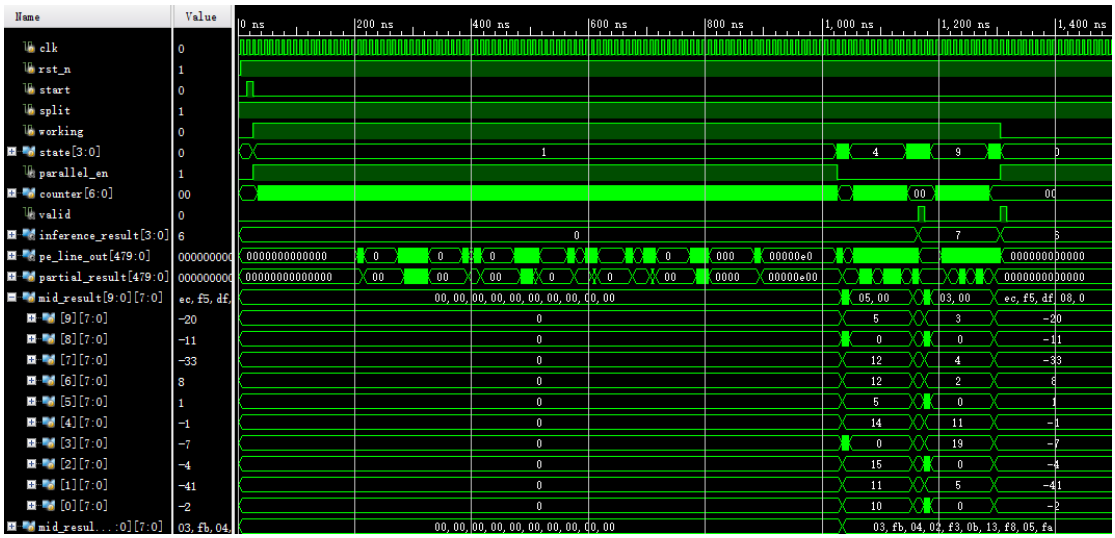

Figure 25 **并行模式下的电路仿真波形图**

Fig. 26 和 Fig. 27 展示了计算过程的中间结果，两者仿真时间对应。Fig. 26 展示了两张输入图片在第一个全连接层后的 scale、ReLU、和第二个全连接层后的 scale 结果，从图中可以看到数据结果和 Fig. 19 中 python 仿真结果能够对应。Fig. 27 展示了第一个全连接层的乘累加结果的变化值和后续的数据变化，可以看到数据结果同样可以和 Fig. 19 中的 python 仿真结果对应。但这里第一层输出的 MAC 值结果为两个 24 比特数据的拼接后结果，因此无法直接和 python 中的结果对应。

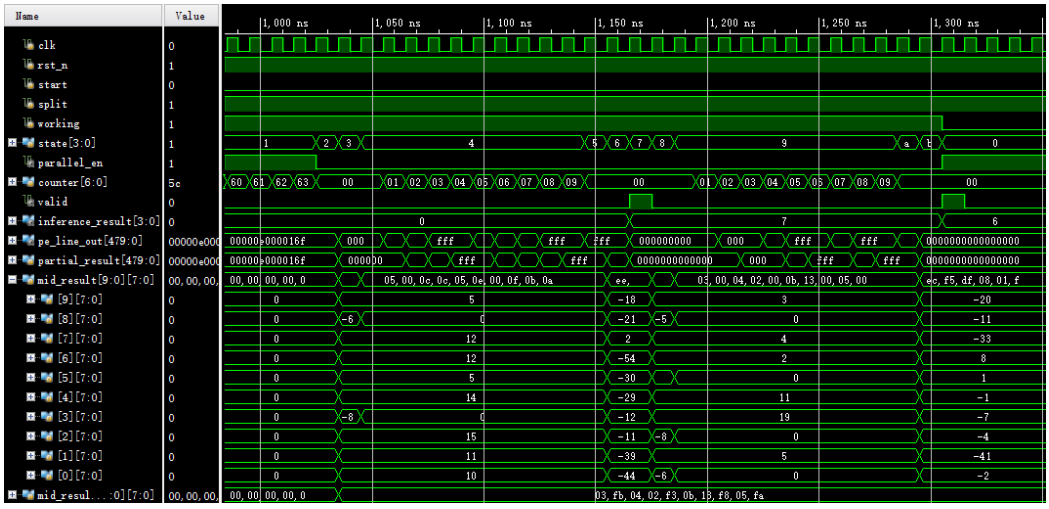

Figure 26 **并行模式下的电路仿真中间数据波形图**

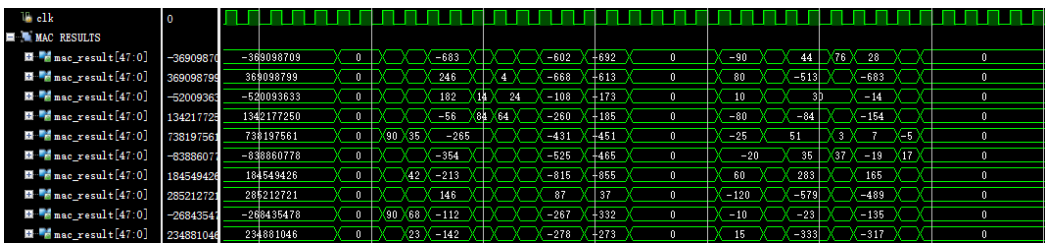

Figure 27 **并行模式下的乘累加结果变化波形图**

（四）计算延时

从 Fig. 22 和 Fig. 25 中可以看出，非并行模式下完成单张输入图像的推理和并行模式下完成两张输入图像的推理分别需要 1150ns 和 1310ns。相比之下，并行计算模式能够降低43%的神经网络推理延时。

DC 综合结果

对所编写的电路通过 DC 工具进行综合，功耗和面积结果如下表所示。详细 DC 输出文件见附件。

Table 1 Power and area of the proposed accelerator

	Power (uW)	Area (μm^2)
Proposed accelerator	1466.2	26050.457405

由表可知，所设计的 CORDIC 计算单元功耗为 1466.2uW，面积为 26050.457405um^2。其中，功耗部分，寄存器功耗为 718.9084uW（占比 49.03%），组合逻辑功耗为 747.2482uW (占比 50.97%)。面积部分，组合逻辑面积为 15014.065231 um^2，缓冲单元面积为 798.012170 um^2，非组合逻辑部分面积为 4924.294230 um^2，互连线部分面积为 6112.097944 um^2。

在时序上，输出 slack 为 4.98ns，有足够的裕量进行计算。

所设计的神经网络加速器能够并行识别两张输入图像，平均每张图像的面积为 13025um^2，功耗为 733.1uW。非并行模式下和并行模式下，识别每张图片的延时平均为 1150ns 和 655ns。

Discussion

所编写电路还有进一步优化的空间。

乘累加电路中，可通过电路复用的形式来进一步降低面积开销，如将部分和的输入数据输入到累加单元阵列中来降低乘法结果和部分和的加法器单元位宽。另外，累加单元的第一行可通过与门的形式和与非门完成乘法，同时由于第一行没有进位数据产生，可以将所有固定 1 数据的输入移动到第二行以进位信号的形式输入，通过这种方式能够有效降低电路复杂度。

另一方面，本次课程项目中选择了将所有中间数据在加速器内部进行缓存的方式，若在并行计算两张图片时，选择将其中一张图片的输出数据写到 buffer 中，则能够降低一部分的寄存器开销，但同时也会增大整体计算延时。

Appendix

Testbench top_tb.v

```
module top_tb();

    reg clk;
    reg rst_n;
    reg start;
    reg split;
```

```
    wire [3:0] inference_result;

    wire weight_we;
    wire [6:0] weight_addr;
    wire [79:0] weight_data;

    wire input_we;
    wire [6:0] input_addr;
    wire [15:0] input_data;
    wire valid;

    // clk
    initial begin
        clk = 1'b0;
        forever begin
            #5 clk = ~clk;
        end
    end

    //rst_n
    initial begin
        rst_n = 1'b0;
        #5 rst_n = 1'b1;
    end

    // start
    initial begin
        start = 1'b0;
        #16 start = 1'b1;
        #10 start = 1'b0;
    end

    // split
    initial begin
        split = 1'b1;
    end

    accelerator i_acc (
        .clk(clk),
        .rst_n(rst_n),
        .start(start),
        .split(split),

        // weight buffer read
```

```
        .weight_we(weight_we),
        .weight_addr(weight_addr),
        .weight_data(weight_data),

        // input buffer read
        .input_we(input_we),
        .input_addr(input_addr),
        .input_data(input_data),

        .valid(valid),
        .inference_result(inference_result)
    );

    sram #(
        .ADDR_WIDTH(7),
        .DATA_WIDTH(80)
    ) sram_weight (
        .clk(clk),
        .addr(weight_addr),
        .din({80'b0}),
        .we(weight_we),
        .dout(weight_data)
    );

    sram #(
        .ADDR_WIDTH(7),
        .DATA_WIDTH(16)
    ) sram_input (
        .clk(clk),
        .addr(input_addr),
        .din({16'b0}),
        .we(input_we),
        .dout(input_data)
    );

    // weight loading initially
    initial begin
        $readmemb("D:\\code\\courses\\proj_multiprecision\\project_ref\\
w1.bin", sram_weight.mem_r, 0,99);
        $readmemb("D:\\code\\courses\\proj_multiprecision\\project_ref\\
w2.bin", sram_weight.mem_r, 100,109);
        $readmemb("D:\\code\\courses\\proj_multiprecision\\project_ref\\
w2.bin", sram_weight.mem_r, 110,119);
    end
```

```
    // input loading initially
    initial begin
        $readmemb("D:\\code\\courses\\proj_multiprecision\\project_ref\\
twoim.bin", sram_input.mem_r);
    end

endmodule
```

accelerator.area.rpts

```
****************************************
Report : area
Design : accelerator
Version: O-2018.06-SP1
Date   : Tue Dec 26 15:08:06 2023
****************************************

Information: Updating design information... (UID-85)
Library(s) Used:

    saed32rvt_ss0p95v125c                                    (File:
/home/xuliukai/class/courses/proj_multiprecision/lib/saed32rvt_ss0p95v125c.db)

Number of ports:                         1174
Number of nets:                          7942
Number of cells:                         6473
Number of combinational cells:           5779
Number of sequential cells:               693
Number of macros/black boxes:               0
Number of buf/inv:                        574
Number of references:                      33

Combinational area:              15014.065231
Buf/Inv area:                      798.012170
Noncombinational area:            4924.294230
Macro/Black Box area:                0.000000
Net Interconnect area:           6112.097944

Total cell area:                 19938.359461
Total area:                      26050.457405

Hierarchical area distribution
```

```
------------------------------

                                   Global cell area                Local cell area
                                   ------------------   ------------------------
-----
Hierarchical  cell                 Absolute      Percent   Combi-          Noncombi-
Black-
                                   Total         Total     national        national
boxes     Design
-------------------------------  ----------  -------  ----------  ---------  -
-----  ---------------
accelerator                        19938.3595    100.0    2764.5785   3785.7291
0.0000    accelerator
plp                                13388.0519     67.1   12249.4867   1138.5651
0.0000    pe_line_parallel
-------------------------------  ----------  -------  ----------  ---------  -
-----  ---------------
Total                                                    15014.0652   4924.2942
0.0000

1
```

accelerator.power.rpts

```
****************************************
Report : power
         -analysis_effort low
Design : accelerator
Version: O-2018.06-SP1
Date   : Tue Dec 26 15:08:06 2023
****************************************


Library(s) Used:

    saed32rvt_ss0p95v125c                                                    (File:
/home/xuliukai/class/courses/proj_multiprecision/lib/saed32rvt_ss0p95v125c.db)


Operating Conditions: ss0p95v125c   Library: saed32rvt_ss0p95v125c
Wire Load Model Mode: enclosed

Design            Wire Load Model                Library
------------------------------------------------
```

```
accelerator                35000                saed32rvt_ss0p95v125c
pe_line_parallel           16000                saed32rvt_ss0p95v125c


Global Operating Voltage = 0.95
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1pW


  Cell Internal Power  = 458.2787 uW   (93%)
  Net Switching Power  =  36.3123 uW   (7%)
                         ---------
Total Dynamic Power    = 494.5910 uW  (100%)

Cell Leakage Power     = 971.5659 uW


                 Internal         Switching         Leakage           Total
Power Group      Power            Power             Power             Power
( % ) Attrs
--------------------------------------------------------------------------------
------------------
io_pad           0.0000           0.0000            0.0000            0.0000
(  0.00%)
memory           0.0000           0.0000            0.0000            0.0000
(  0.00%)
black_box        0.0000           0.0000            0.0000            0.0000
(  0.00%)
clock_network    0.0000           0.0000            0.0000            0.0000
(  0.00%)
register       387.7987           5.0130        3.2610e+08          718.9084
(  49.03%)
sequential       0.0000           0.0000            0.0000            0.0000
(  0.00%)
combinational   70.4799          31.2994        6.4547e+08          747.2482
(  50.97%)
--------------------------------------------------------------------------------
------------------
Total          458.2786 uW       36.3123 uW    9.7157e+08 pW     1.4662e+03 uW
```

accelerator.timing.rpts

```
****************************************
Report : timing
         -path full
         -delay max
         -max_paths 1
Design : accelerator
Version: O-2018.06-SP1
Date   : Tue Dec 26 15:08:06 2023
****************************************

Operating Conditions: ss0p95v125c   Library: saed32rvt_ss0p95v125c
Wire Load Model Mode: enclosed

  Startpoint: state_reg[1]
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: mid_result_buffer_reg[3][6]
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  accelerator        35000                 saed32rvt_ss0p95v125c
  pe_line_parallel   16000                 saed32rvt_ss0p95v125c

  Point                                                   Incr       Path
  --------------------------------------------------------------------------
  clock clk (rise edge)                                   0.00       0.00
  clock network delay (ideal)                             0.00       0.00
  state_reg[1]/CLK (DFFARX1_RVT)                          0.00       0.00 r
  state_reg[1]/QN (DFFARX1_RVT)                           0.16       0.16 r
  U1439/Y (AND2X1_RVT)                                    0.07       0.22 r
  U1440/Y (AND3X1_RVT)                                    0.09       0.31 r
  plp/split (pe_line_parallel)                            0.00       0.31 r
  plp/U46/Y (INVX2_RVT)                                   0.08       0.40 f
  plp/U873/Y (NBUFFX8_RVT)                                0.12       0.52 f
  plp/U23/Y (INVX0_RVT)                                   0.13       0.64 r
  plp/U424/Y (NAND3X0_RVT)                                0.07       0.71 f
  plp/U425/Y (OAI221X1_RVT)                               0.13       0.84 r
  plp/U426/Y (NAND3X0_RVT)                                0.06       0.90 f
```

plp/U427/Y (OA221X1_RVT)	0.10	1.01 f
plp/U3117/S (FADDX1_RVT)	0.18	1.19 r
plp/U3103/S (FADDX1_RVT)	0.17	1.35 f
plp/U3109/Y (NAND2X0_RVT)	0.06	1.42 r
plp/intadd_60/U5/S (FADDX1_RVT)	0.16	1.58 f
plp/intadd_61/U4/S (FADDX1_RVT)	0.15	1.73 r
plp/intadd_2/U8/S (FADDX1_RVT)	0.16	1.89 f
plp/intadd_72/U2/S (FADDX1_RVT)	0.18	2.07 r
plp/U3136/Y (AO222X1_RVT)	0.16	2.22 r
plp/U3153/Y (AO222X1_RVT)	0.16	2.38 r
plp/U3168/Y (AO222X1_RVT)	0.15	2.53 r
plp/U3181/Y (AO222X1_RVT)	0.15	2.68 r
plp/U3191/Y (AO222X1_RVT)	0.15	2.83 r
plp/U3200/Y (AO222X1_RVT)	0.15	2.98 r
plp/U3209/Y (AO222X1_RVT)	0.15	3.13 r
plp/U3216/Y (AO222X1_RVT)	0.15	3.28 r
plp/U3217/Y (OR2X1_RVT)	0.06	3.34 r
plp/U3229/Y (AO21X1_RVT)	0.08	3.42 r
plp/U3233/Y (AO21X1_RVT)	0.08	3.50 r
plp/U3242/Y (AO21X1_RVT)	0.08	3.58 r
plp/U3255/Y (AO21X1_RVT)	0.08	3.66 r
plp/U3259/Y (AO21X1_RVT)	0.08	3.74 r
plp/U3271/Y (AO21X1_RVT)	0.08	3.82 r
plp/U3278/Y (AO21X1_RVT)	0.08	3.90 r
plp/U3283/Y (AO21X1_RVT)	0.08	3.98 r
plp/U3291/Y (AO21X1_RVT)	0.08	4.06 r
plp/U3296/Y (AO21X1_RVT)	0.08	4.14 r
plp/U3303/Y (AO21X1_RVT)	0.08	4.22 r
plp/U3310/Y (AO21X1_RVT)	0.08	4.30 r
plp/U3317/Y (AO21X1_RVT)	0.08	4.38 r
plp/U3323/Y (NAND2X0_RVT)	0.04	4.42 f
plp/U3325/Y (NAND2X0_RVT)	0.05	4.48 r
plp/U3334/Y (OAI21X1_RVT)	0.11	4.59 f
plp/U3342/Y (OA221X1_RVT)	0.07	4.66 f
plp/U3343/SO (HADDX1_RVT)	0.10	4.76 r
plp/U3344/Y (AND2X1_RVT)	0.06	4.82 r
plp/U3349/Y (AO21X1_RVT)	0.05	4.87 r
plp/out[176] (pe_line_parallel)	0.00	4.87 r
U2490/Y (AO22X1_RVT)	0.08	4.95 r
mid_result_buffer_reg[3][6]/D (DFFARX1_RVT)	0.00	4.95 r
data arrival time		4.95
clock clk (rise edge)	10.00	10.00
clock network delay (ideal)	0.00	10.00

mid_result_buffer_reg[3][6]/CLK (DFFARX1_RVT)	0.00	10.00 r
library setup time	-0.07	9.93
data required time		9.93
---	---	---
data required time		9.93
data arrival time		-4.95
---	---	---
slack (MET)		4.98