

同济大学计算机系

数字逻辑课程综合实验报告



学 号 2053620

姓 名 张润泽

专 业 计算机科学与技术

授课老师 郭玉臣

一、实验内容

1. 项目名称

基于 FPGA 的手写数字识别系统。

2. 内容简介

本实验结合蓝牙、MP3 和 VGA 显示器三个外设进行信息交互，并通过部署在开发板的简单卷积神经网络，实现了基于 FPGA 的手写数字识别系统。

3. 操作流程

(1) Vivado 生成比特流文件，程序下板；

(2) PC 端连接蓝牙，并通过蓝牙串口工具将手写数字灰度图的像素数据发送给 FPGA；

(3) FPGA 将接收到的灰度图像素数据进行储存，并将其在 VGA 显示器上显示；

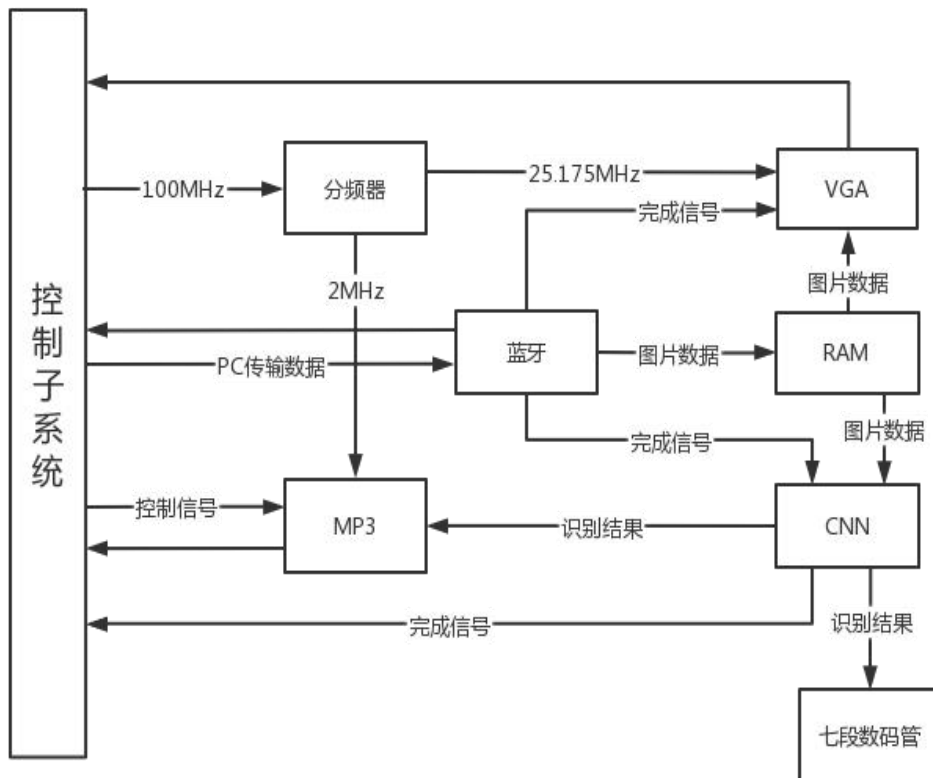
(4) 将灰度图像素数据输入进简单卷积神经网络，网络输出的结果即为手写数字识别结果；

(5) 将识别结果在七段数码管上显示，并通过 MP3 进行语音播报结果；

(6) 拨动置位开关（SW0），回到流程（2）重新开始。

二、基于 FPGA 的手写数字识别系统总框图

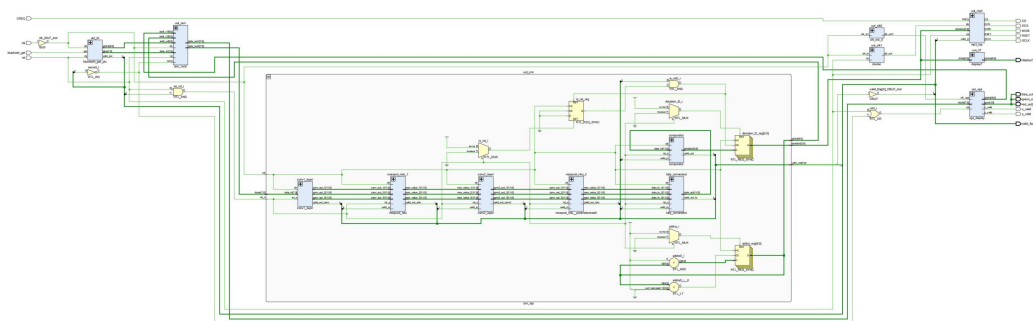
总框图：

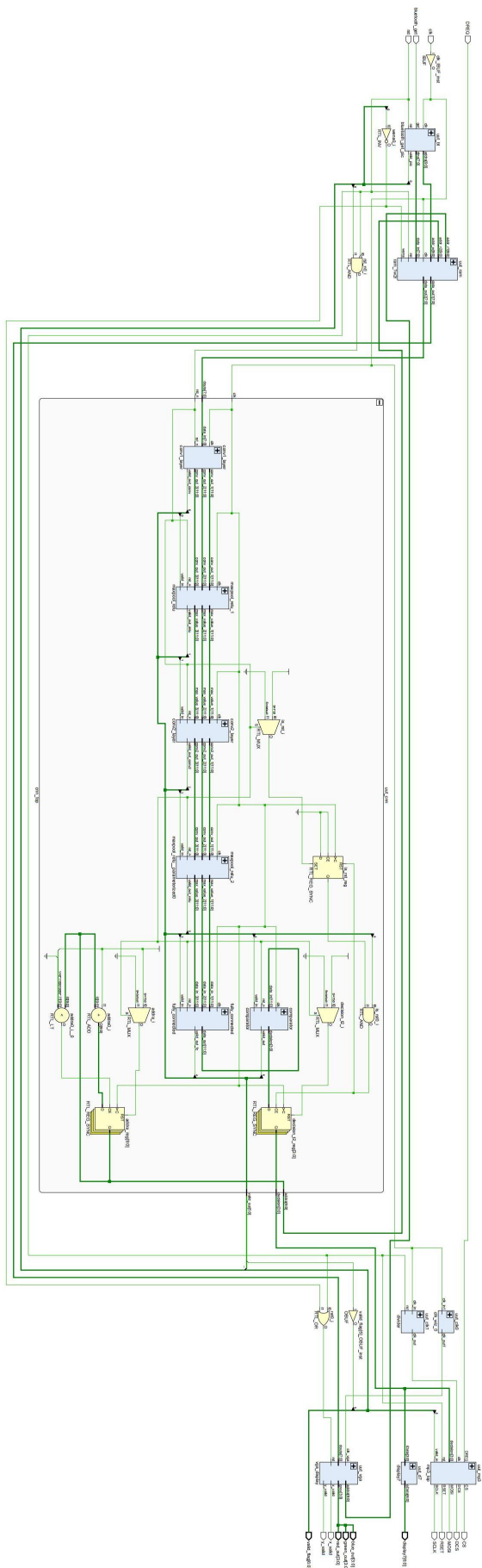


整个系统共用一个顶层的控制系统，下设多级系统，包含众多子模块。

1. 顶层模块：
对分频器模块、RAM 模块、蓝牙模块、VGA 模块、CNN 模块、七段数码管模块以及 MP3 模块进行实例化调用。
2. 分频器模块：
为 VGA 和 MP3 模块提供分频时钟。
3. RAM 模块：
存储灰度图像素数据，含有一读双写共三个接口。
4. 蓝牙模块：
将蓝牙传递过来的灰度图像素数据存储进 RAM 模块。
5. VGA 模块：
读取 RAM 模块存储的灰度图像素数据，并将其显示在 VGA 显示器上。
其内有 VGA 控制子模块。
6. CNN 模块：
读取 RAM 模块存储的灰度图像素数据，并输入进网络进行处理，最后输出识别结果。其内有多层卷积层子模块。
7. 七段数码管模块：
将识别结果在七段数码管上进行显示。
8. MP3 模块：
识别结果通过 MP3 进行语音播报。其内有 MP3 驱动子模块。

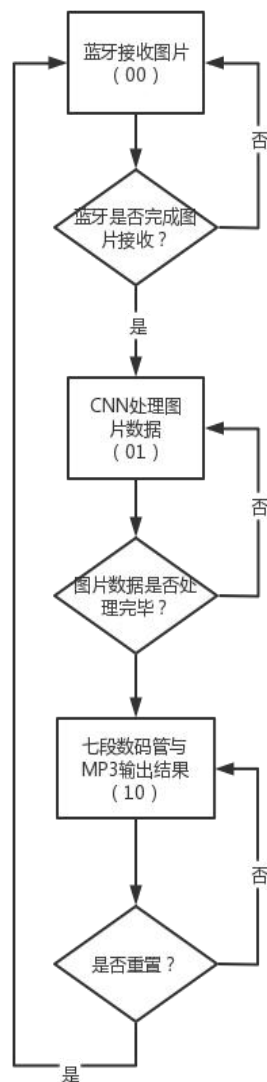
RTL 设计原理图如下：





三、系统控制器设计

顶层 ASM 流程图：



状态转移真值表、次态激励函数表达式和控制命令逻辑表达式较为容易，此处略。

四、子系统模块建模

本系统共划分了7个子系统模块受顶层模块调用，共包括：时钟分频模块、存储数据RAM模块、蓝牙获取图片模块、VGA显示模块、CNN图片处理模块、七段数码管显示模块和MP3播放模块，下面将分别介绍之。

1. 顶层模块

功能：实例化调用各子模块。

建模 Verilog 代码：

```
1. `timescale 1ns / 1ps
2.
```

```

3.  module mnist_top(
4.      input clk,
5.      input rst,
6.      output [6:0] valid_flag,
7.      output [6:0] display7,
8.      // BLUETOOTH
9.      input bluetooth_get,
10.     // VGA
11.     output [3:0] red, green, blue,
12.     output Hs_valid,
13.     output Vs_valid,
14.     // MP3
15.     input DREQ,
16.     output xRSET,
17.     output xCS,
18.     output xDCS,
19.     output MOSI,
20.     output SCLK
21. );
22.
23.     wire clk_vga ;//vga 25.175mhz
24.     wire clk_mp3_std;//mp3 12.288mhz
25.     wire clk_mp3;//mp3 2mhz
26.     clk_wiz_0 uut_clk0(.clk_in1(clk), .clk_out1(clk_vga), .clk_out2(clk_
        k_mp3_std));
27.     divider #(.(mod(200))) uut_clk1(.clk_in(clk),.rst(rst),.clk_out(clk_
        mp3));
28.
29.     wire [9:0] addr_w, addr_r1, addr_r2;
30.     wire [7:0] data_in, data_out1, data_out2;
31.
32.     ram_1w2r uut_ram (
33.         .clk(clk),
34.         .rst(rst),
35.         .wena(~valid_flag[0]),
36.         .addr_w(addr_w),
37.         .data_in(data_in),
38.         .addr_r1(addr_r1),
39.         .data_out1(data_out1),
40.         .addr_r2(addr_r2),
41.         .data_out2(data_out2)
42.     );
43.
44.     bluetooth_get_pic uut_bt (

```

```

45.         .clk(clk),
46.         .rst(rst),
47.         .get(bluetooth_get),
48.         .dina(data_in),
49.         .addra(addr_w),
50.         .valid_pic(valid_flag[0])
51.     );
52.
53.     vga_display uut_vga (
54.         .clk_vga(clk_vga), //25.175MHz
55.         .rst(rst | ~valid_flag[0]),
56.         .douta(data_out1),
57.         .addra(addr_r1),
58.         .Hs_valid(Hs_valid),
59.         .Vs_valid(Vs_valid),
60.         .grey(red)
61.     );
62.
63.     assign green = red;
64.     assign blue = red;
65.
66.     wire [3:0] decision;
67.     cnn_top uut_cnn (
68.         .clk(clk),
69.         .rst_n(~rst & valid_flag[0]),
70.         .douta(data_out2),
71.         .addra(addr_r2),
72.         .valid_out(valid_flag[6:1]),
73.         .decision(decision)
74.     );
75.
76.     display7 uut_d7 (.iData(decision), .oData(display7));
77.
78.     mp3_top uut_mp3 (
79.         .clk(clk_mp3),
80.         .rst(rst),
81.         .valid_in(valid_flag[6]),
82.         .decision(decision),
83.         .DREQ(DREQ),
84.         .xRSET(xRSET),
85.         .xCS(xCS),
86.         .xDCS(xDCS),
87.         .MOSI(MOSI),
88.         .SCLK(SCLK)

```

```

89.         );
90.
91.     endmodule

```

2. 时钟分频模块

功能：使用 IP 核 Clocking Wizard 将系统 100MHz 标准时钟 clk 分频为 25.175MHz 的 VGA 所需时钟；使用数字逻辑小作业中实现的 divider 分频器，获取 2MHz 的 MP3 所需时钟（由于分频器 IP 核不能得到很小频率的时钟）。

建模 Verilog 代码：

```

1.  `timescale 1ns / 1ps
2.
3.  module divider #(parameter mod = 32'd20) (
4.      input clk_in,
5.      input rst,
6.      output reg clk_out
7.  );
8.
9.      localparam T = mod / 2;
10.
11.     reg [31:0] cnt = 0;
12.
13.     initial begin
14.         clk_out <= 0;
15.     end
16.
17.     always @(posedge clk_in) begin
18.         if(rst == 1) begin
19.             cnt <= 0;
20.             clk_out <= 0;
21.         end
22.         else begin
23.             if(cnt == T) begin
24.                 cnt <= 1;
25.                 clk_out <= ~clk_out;
26.             end
27.             else
28.                 cnt <= cnt + 1;
29.             end
30.         end
31.
32.     endmodule

```

3. 存储数据 RAM 模块

功能：改进数字逻辑小作业中实现的 RAM 模块，实现一写双读的 RAM 模

块，地址宽度 10 位，数据宽度 8 位，深度为 784，用于储存 $28 \times 28 \times 8$ 的手写数字灰度图片像素数据。

建模 Verilog 代码：

```
1.  `timescale 1ns / 1ps
2.
3.  module ram_1w2r(
4.      input clk,
5.      input rst,
6.      input wena,
7.      input [9:0] addr_w,
8.      input [7:0] data_in,
9.      input [9:0] addr_r1,
10.     output reg [7:0] data_out1,
11.     input [9:0] addr_r2,
12.     output reg [7:0] data_out2
13. );
14.
15.     reg [7:0] pic[0:783];
16.
17.     always @(posedge clk or negedge rst) begin
18.         if (rst) begin
19.             data_out1 <= 0;
20.             data_out2 <= 0;
21.         end
22.         else begin
23.             if (wena)
24.                 pic[addr_w] <= data_in;
25.             else begin
26.                 data_out1 <= pic[addr_r1];
27.                 data_out2 <= pic[addr_r2];
28.             end
29.         end
30.     end
31.
32. endmodule
```

4. 蓝牙获取图片模块

功能：实现了 8 位数据位 1 位停止位的 UART 串口协议，在 9600 波特率下与蓝牙进行通信，并将获取的手写数字灰度图片像素数据存储进 RAM 模块，获取完成后向外发出信号，进行下一阶段 VGA 显示以及 CNN 处理图片数据。

建模 Verilog 代码：

```
1.  `timescale 1ns / 1ps
2.
3.  module bluetooth_get_pic(
```

```

4.      input clk,
5.      input rst,
6.      input get,
7.      output reg [7:0] dina,
8.      output reg [9:0] addra,
9.      output reg valid_pic
10. );
11.
12.      parameter bps = 10417; // 1/9600
13.      reg [14:0] count_1;
14.      reg [3:0] count_2;
15.      reg buffer_0, buffer_1, buffer_2;
16.      wire buffer_en;
17.      reg add_en;
18.
19.      always @(posedge clk) begin
20.          if (rst) begin
21.              buffer_0 <= 1;
22.              buffer_1 <= 1;
23.              buffer_2 <= 1;
24.          end
25.          else begin
26.              buffer_0 <= get;
27.              buffer_1 <= buffer_0;
28.              buffer_2 <= buffer_1;
29.          end
30.      end
31.
32.      assign buffer_en = buffer_2 & ~buffer_1;
33.
34.      always @(posedge clk) begin
35.          if (rst)
36.              count_1 <= 0;
37.          else if (add_en) begin
38.              if(count_1 == bps-1)
39.                  count_1 <= 0;
40.              else
41.                  count_1 <= count_1+1;
42.          end
43.      end
44.
45.      always @(posedge clk) begin
46.          if (rst)
47.              count_2 <= 0;

```

```

48.         else if (add_en && count_1 == bps-1) begin
49.             if(count_2 == 8)
50.                 count_2 <= 0;
51.             else
52.                 count_2 <= count_2 + 1;
53.             end
54.         end
55.
56.         always @(posedge clk) begin
57.             if (rst)
58.                 add_en <= 0;
59.             else if (buffer_en)
60.                 add_en <= 1;
61.             else if (add_en && (count_2 == 8) && (count_1 == bps-1))
62.                 add_en <= 0;
63.             end
64.
65.         always @(posedge clk) begin
66.             if (rst) begin
67.                 dina <= 0;
68.                 addra <= 0;
69.                 valid_pic <= 0;
70.             end
71.             else if(add_en && (count_1 == bps / 2 - 1) && (count_2 != 0))
begin
72.                 dina[count_2 - 1] <= get;
73.                 if (count_2 == 8) begin
74.                     addra = addra + 1;
75.                     if (addra == 10'd784) begin
76.                         valid_pic <= 1;
77.                     end
78.                 end
79.             end
80.         end
81.
82.     endmodule

```

5. VGA 显示模块

功能：读取 RAM 模块的存储的数据，并控制 VGA 的行和场同步信号以及颜色信号，以将图片显示在 VGA 显示器上，此模块在接收到蓝牙模块的信号后开始运行。

建模 Verilog 代码：

```

1.     `timescale 1ns / 1ps
2.

```

```

3.  module vga_display(
4.      input clk_vga,//25.175MHz
5.      input rst,
6.      input [7:0] douta,
7.      output reg [9:0] addra,
8.      output Hs_valid,
9.      output Vs_valid,
10.     output reg [3:0] grey
11. );
12.     parameter Hs_before=11'd144;
13.     parameter Vs_before=11'd35;
14.     parameter Hs_size_pic=11'd280;
15.     parameter Vs_size_pic=11'd280;
16.
17.     wire [11:0] Hs_poi;
18.     wire [11:0] Vs_poi;
19.     wire is_display;
20.
21.     vga_control control(clk_vga,rst,Hs_poi,Vs_poi,is_display,Hs_valid,
        Vs_valid);
22.
23.     always @(*) begin
24.         grey <= 0;
25.         if (is_display && Hs_poi-Hs_before <= Hs_size_pic && Vs_poi-Vs
            _before <= Vs_size_pic) begin
26.             addra <= (Vs_poi - Vs_before)/10*28 + (Hs_poi - Hs_before)
                /10;
27.             grey <= douta * 10;
28.         end
29.     end
30.
31. endmodule

```

```

1.  `timescale 1ns / 1ps
2.
3.  module vga_control(
4.      input vga_clk,
5.      input rst,
6.      output reg[11:0] Hs_poi,
7.      output reg[11:0] Vs_poi,
8.      output is_display,
9.      output Hs_valid,
10.     output Vs_valid
11. );

```

```

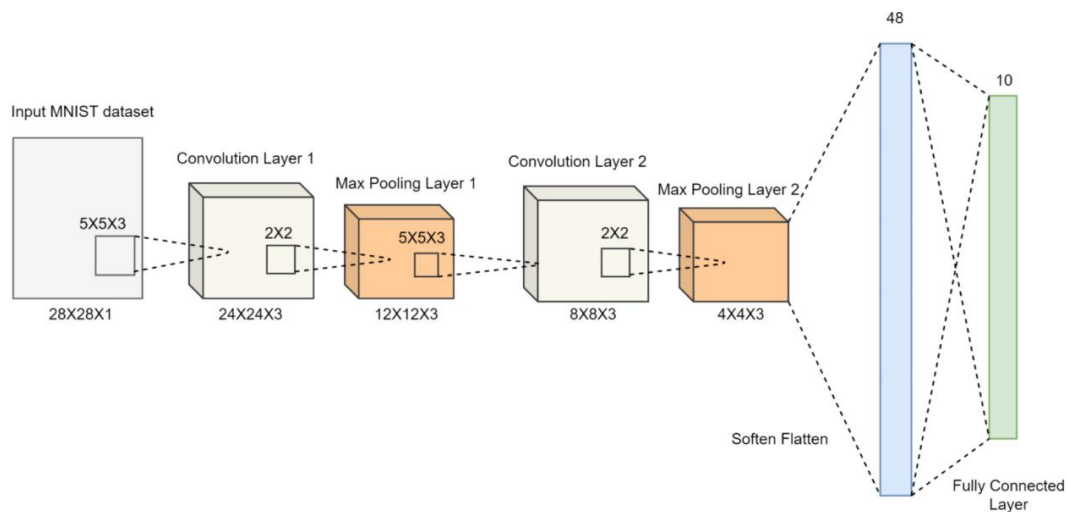
12.
13.     parameter x_sync = 11'd96;
14.     parameter x_before = 11'd144;
15.     parameter x_beside_after = 11'd784;
16.     parameter x_all = 11'd800;
17.
18.     parameter y_sync = 11'd2;
19.     parameter y_before = 11'd35;
20.     parameter y_beside_after = 11'd515;
21.     parameter y_all = 11'd525;
22.
23.     assign is_display = ((Hs_poi >= x_before) && (Hs_poi < x_beside_af
        ter)
24.                               && (Vs_poi >= y_before) && (Vs_poi < y_beside_
        after)) ? 1 : 0;
25.
26.     assign Hs_valid = (Hs_poi < x_sync) ? 0 : 1;
27.     assign Vs_valid = (Vs_poi < y_sync) ? 0 : 1;
28.
29.     always @(posedge vga_clk) begin
30.         if (rst) begin
31.             Hs_poi <= 0;
32.             Vs_poi <= 0;
33.         end
34.         else begin
35.             if (Hs_poi == x_all - 1) begin
36.                 Hs_poi <= 0;
37.                 if (Vs_poi == y_all - 1)
38.                     Vs_poi <= 0;
39.                 else
40.                     Vs_poi <= Vs_poi + 1;
41.             end
42.             else
43.                 Hs_poi <= Hs_poi + 1;
44.             end
45.         end
46.
47.     endmodule

```

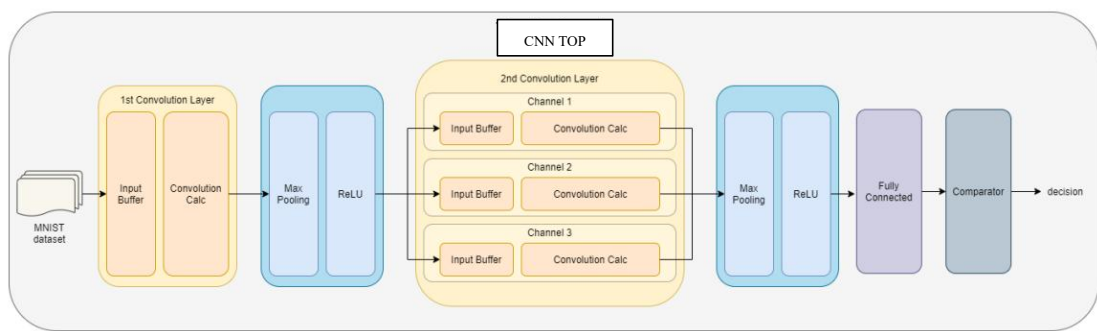
6. CNN 图片处理模块

功能：读取 RAM 模块的存储的手写数字灰度图像素数据，输入进简单卷积神经网络中进行处理，处理完毕后输出识别结果；该模块由两个卷积层、两个最大值池化层、一个全连接层和一个十位比较器总共六个子模块构成。

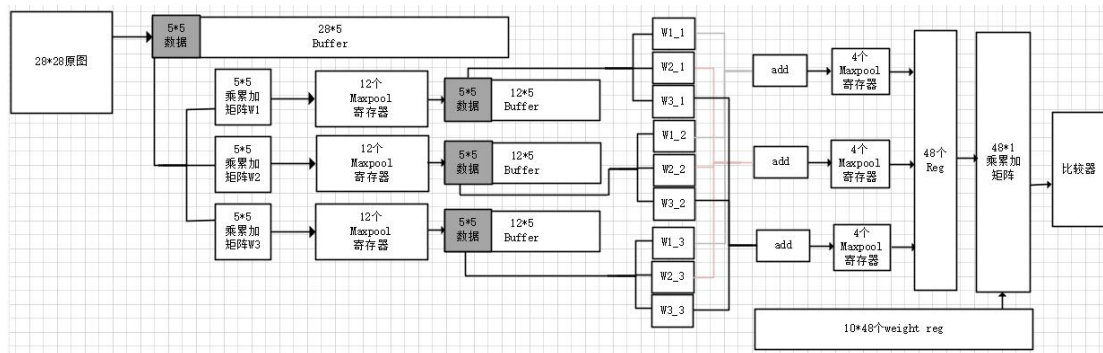
使用的神经网络的结构如下



该子模块整体架构如下



图片数据处理分解后具体架构如下



其中 Buffer 即为一次性缓存的最大数据量，各子模块均在收到前一模块处理完毕的信号后才使能运行，输入的数据经由上述流程一步步处理，最后得到识别结果。

建模 Verilog 代码：

```
1. `timescale 1ns / 1ps
2.
3. module cnn_top(
4.     input clk,
5.     input rst_n,
```

```

6.      input [7:0] douta,
7.      output reg [9:0] addra,
8.      output [5:0] valid_out,
9.      output [3:0] decision
10.    );
11.
12.    wire signed [11:0] conv_out_1, conv_out_2, conv_out_3;
13.    wire signed [11:0] conv2_out_1, conv2_out_2, conv2_out_3;
14.    wire signed [11:0] max_value_1, max_value_2, max_value_3;
15.    wire signed [11:0] max2_value_1, max2_value_2, max2_value_3;
16.    wire signed [11:0] fc_out_data;
17.
18.    // Module Instantiation
19.    conv1_layer conv1_layer(
20.        .clk(clk),
21.        .rst_n(rst_n),
22.        .data_in(douta),
23.        .conv_out_1(conv_out_1),
24.        .conv_out_2(conv_out_2),
25.        .conv_out_3(conv_out_3),
26.        .valid_out_conv(valid_out[0])
27.    );
28.
29.    maxpool_relu #(.CONV_BIT(12), .HALF_WIDTH(12), .HALF_HEIGHT(12), .
    HALF_WIDTH_BIT(4))
30.    maxpool_relu_1(
31.        .clk(clk),
32.        .rst_n(rst_n),
33.        .valid_in(valid_out[0]),
34.        .conv_out_1(conv_out_1),
35.        .conv_out_2(conv_out_2),
36.        .conv_out_3(conv_out_3),
37.        .max_value_1(max_value_1),
38.        .max_value_2(max_value_2),
39.        .max_value_3(max_value_3),
40.        .valid_out_relu(valid_out[1])
41.    );
42.
43.    conv2_layer conv2_layer(
44.        .clk(clk),
45.        .rst_n(rst_n),
46.        .valid_in(valid_out[1]),
47.        .max_value_1(max_value_1),
48.        .max_value_2(max_value_2),

```

```

49.         .max_value_3(max_value_3),
50.         .conv2_out_1(conv2_out_1),
51.         .conv2_out_2(conv2_out_2),
52.         .conv2_out_3(conv2_out_3),
53.         .valid_out_conv2(valid_out[2])
54.     );
55.
56.     maxpool_relu #(.CONV_BIT(12), .HALF_WIDTH(4), .HALF_HEIGHT(4), .HALF_WIDTH_BIT(3))
57.     maxpool_relu_2(
58.         .clk(clk),
59.         .rst_n(rst_n),
60.         .valid_in(valid_out[2]),
61.         .conv_out_1(conv2_out_1),
62.         .conv_out_2(conv2_out_2),
63.         .conv_out_3(conv2_out_3),
64.         .max_value_1(max2_value_1),
65.         .max_value_2(max2_value_2),
66.         .max_value_3(max2_value_3),
67.         .valid_out_relu(valid_out[3])
68.     );
69.
70.     fully_connected fully_connected(
71.         .clk(clk),
72.         .rst_n(rst_n),
73.         .valid_in(valid_out[3]),
74.         .data_in_1(max2_value_1),
75.         .data_in_2(max2_value_2),
76.         .data_in_3(max2_value_3),
77.         .data_out(fc_out_data),
78.         .valid_out_fc(valid_out[4])
79.     );
80.
81.     wire [3:0] decision_t1;
82.
83.     comparator comparator(
84.         .clk(clk),
85.         .rst_n(rst_n),
86.         .valid_in(valid_out[4]),
87.         .data_in(fc_out_data),
88.         .decision(decision_t1),
89.         .valid_out(valid_out[5])
90.     );
91.

```



```

92. //    ila_1 outer (
93. //        .clk(clk),
94. //        .probe0(valid_out),
95. //        .probe1(decision_t1),
96. //        .probe2(douta),
97. //        .probe3(conv_out_1),
98. //        .probe4(max_value_1),
99. //        .probe5(conv2_out_1),
100. //        .probe6(max2_value_1),
101. //        .probe7(fc_out_data)
102. //    );
103.
104.    reg [3:0] decision_t2;
105.    reg is_rst;
106.
107.    always @(posedge clk) begin
108.        if (~rst_n) begin
109.            is_rst <= 1;
110.            decision_t2 <= 0;
111.        end
112.        else if(valid_out[5] && is_rst) begin
113.            decision_t2 <= decision_t1;
114.            is_rst <= 0;
115.        end
116.    end
117.
118.    assign decision = decision_t2;
119.
120.    always @(posedge clk) begin
121.        if (~rst_n)
122.            addra <= 0;
123.        else if(addra < 10'd784)
124.            addra <= addra + 1'b1;
125.        end
126.
127. endmodule

```

```

1.    `timescale 1ns / 1ps
2.
3.    // Design    : 1st Convolution Layer for CNN MNIST dataset
4.
5.    module conv1_layer (
6.        input clk,
7.        input rst_n,
8.        input [7:0] data_in,

```

```
9.         output [11:0] conv_out_1, conv_out_2, conv_out_3,
10.         output valid_out_conv
11.     );
12.
13.     wire [7:0] data_out_0, data_out_1, data_out_2, data_out_3, data_out_4
14.         ,
15.         data_out_5, data_out_6, data_out_7, data_out_8, data_out_9,
16.         data_out_10, data_out_11, data_out_12, data_out_13, data_out_14,
17.         data_out_15, data_out_16, data_out_17, data_out_18, data_out_19,
18.         data_out_20, data_out_21, data_out_22, data_out_23, data_out_24;
19.     wire valid_out_buf;
20.
21.     conv1_buf #(.WIDTH(28), .HEIGHT(28), .DATA_BITS(8)) conv1_buf(
22.         .clk(clk),
23.         .rst_n(rst_n),
24.         .data_in(data_in),
25.         .data_out_0(data_out_0),
26.         .data_out_1(data_out_1),
27.         .data_out_2(data_out_2),
28.         .data_out_3(data_out_3),
29.         .data_out_4(data_out_4),
30.         .data_out_5(data_out_5),
31.         .data_out_6(data_out_6),
32.         .data_out_7(data_out_7),
33.         .data_out_8(data_out_8),
34.         .data_out_9(data_out_9),
35.         .data_out_10(data_out_10),
36.         .data_out_11(data_out_11),
37.         .data_out_12(data_out_12),
38.         .data_out_13(data_out_13),
39.         .data_out_14(data_out_14),
40.         .data_out_15(data_out_15),
41.         .data_out_16(data_out_16),
42.         .data_out_17(data_out_17),
43.         .data_out_18(data_out_18),
44.         .data_out_19(data_out_19),
45.         .data_out_20(data_out_20),
46.         .data_out_21(data_out_21),
47.         .data_out_22(data_out_22),
48.         .data_out_23(data_out_23),
49.         .data_out_24(data_out_24),
50.         .valid_out_buf(valid_out_buf)
51.     );
```

```

52.
53.     conv1_calc conv1_calc(
54.         .clk(clk),
55.         .valid_out_buf(valid_out_buf),
56.         .data_out_0(data_out_0),
57.         .data_out_1(data_out_1),
58.         .data_out_2(data_out_2),
59.         .data_out_3(data_out_3),
60.         .data_out_4(data_out_4),
61.         .data_out_5(data_out_5),
62.         .data_out_6(data_out_6),
63.         .data_out_7(data_out_7),
64.         .data_out_8(data_out_8),
65.         .data_out_9(data_out_9),
66.         .data_out_10(data_out_10),
67.         .data_out_11(data_out_11),
68.         .data_out_12(data_out_12),
69.         .data_out_13(data_out_13),
70.         .data_out_14(data_out_14),
71.         .data_out_15(data_out_15),
72.         .data_out_16(data_out_16),
73.         .data_out_17(data_out_17),
74.         .data_out_18(data_out_18),
75.         .data_out_19(data_out_19),
76.         .data_out_20(data_out_20),
77.         .data_out_21(data_out_21),
78.         .data_out_22(data_out_22),
79.         .data_out_23(data_out_23),
80.         .data_out_24(data_out_24),
81.         .conv_out_1(conv_out_1),
82.         .conv_out_2(conv_out_2),
83.         .conv_out_3(conv_out_3),
84.         .valid_out_calc(valid_out_conv)
85.     );
86.
87.     endmodule

```

```

1.     `timescale 1ns / 1ps
2.
3.     // Description: 1st Convolution Layer for CNN MNIST dataset Input Buffer
4.
5.     module conv1_buf #(parameter WIDTH = 28, HEIGHT = 28, DATA_BITS = 8)(
6.         input clk,

```

```

7.     input rst_n,
8.     input [DATA_BITS - 1:0] data_in,
9.     output reg [DATA_BITS - 1:0] data_out_0, data_out_1, data_out_2, da
        ta_out_3, data_out_4,
10.    data_out_5, data_out_6, data_out_7, data_out_8, data_out_9,
11.    data_out_10, data_out_11, data_out_12, data_out_13, data_out_14,
12.    data_out_15, data_out_16, data_out_17, data_out_18, data_out_19,
13.    data_out_20, data_out_21, data_out_22, data_out_23, data_out_24,
14.    output reg valid_out_buf
15.    );
16.
17.    localparam FILTER_SIZE = 5;
18.
19.    reg [DATA_BITS - 1:0] buffer [0:WIDTH * FILTER_SIZE - 1];
20.    reg [DATA_BITS - 1:0] buf_idx;
21.    reg [4:0] w_idx, h_idx;
22.    reg [2:0] buf_flag; // 0 ~ 4
23.    reg state;
24.
25.    always @(posedge clk) begin
26.        if (~rst_n) begin
27.            buf_idx <= -1;
28.            w_idx <= 0;
29.            h_idx <= 0;
30.            buf_flag <= 0;
31.            state <= 0;
32.            valid_out_buf <= 0;
33.            data_out_0 <= 12'bx;
34.            data_out_1 <= 12'bx;
35.            data_out_2 <= 12'bx;
36.            data_out_3 <= 12'bx;
37.            data_out_4 <= 12'bx;
38.            data_out_5 <= 12'bx;
39.            data_out_6 <= 12'bx;
40.            data_out_7 <= 12'bx;
41.            data_out_8 <= 12'bx;
42.            data_out_9 <= 12'bx;
43.            data_out_10 <= 12'bx;
44.            data_out_11 <= 12'bx;
45.            data_out_12 <= 12'bx;
46.            data_out_13 <= 12'bx;
47.            data_out_14 <= 12'bx;
48.            data_out_15 <= 12'bx;
49.            data_out_16 <= 12'bx;

```

```

50.      data_out_17 <= 12'bx;
51.      data_out_18 <= 12'bx;
52.      data_out_19 <= 12'bx;
53.      data_out_20 <= 12'bx;
54.      data_out_21 <= 12'bx;
55.      data_out_22 <= 12'bx;
56.      data_out_23 <= 12'bx;
57.      data_out_24 <= 12'bx;
58.      end
59.  else begin
60.      buf_idx <= buf_idx + 1;
61.
62.      if (buf_idx == WIDTH * FILTER_SIZE - 1) // buffer size = 140 = 28
(w) * 5(h)
63.          buf_idx <= 0;
64.
65.      buffer[buf_idx] <= data_in; // data input
66.
67.      // Wait until first 140 input data filled in buffer
68.      if (!state) begin
69.          if (buf_idx == WIDTH * FILTER_SIZE - 1)
70.              state <= 1'b1;
71.          end
72.      else begin // valid state
73.          w_idx <= w_idx + 1'b1; // move right
74.
75.          if (w_idx == WIDTH - FILTER_SIZE + 1)
76.              valid_out_buf <= 1'b0; // unvalid area
77.          else if (w_idx == WIDTH - 1) begin
78.              buf_flag <= buf_flag + 1'b1;
79.              if (buf_flag == FILTER_SIZE - 1)
80.                  buf_flag <= 0;
81.              w_idx <= 0;
82.              if (h_idx == HEIGHT - FILTER_SIZE) begin // done 1 input rea
d -> 28 * 28
83.                  h_idx <= 0;
84.                  state <= 1'b0;
85.                  end
86.                  h_idx <= h_idx + 1'b1;
87.                  end
88.          else if(w_idx == 0)
89.              valid_out_buf <= 1'b1; // start valid area
90.
91.          // Buffer Selection -> 5 * 5

```

```
92.         if (buf_flag == 3'd0) begin
93.             data_out_0 <= buffer[w_idx];
94.             data_out_1 <= buffer[w_idx + 1];
95.             data_out_2 <= buffer[w_idx + 2];
96.             data_out_3 <= buffer[w_idx + 3];
97.             data_out_4 <= buffer[w_idx + 4];
98.
99.             data_out_5 <= buffer[w_idx + WIDTH];
100.            data_out_6 <= buffer[w_idx + 1 + WIDTH];
101.            data_out_7 <= buffer[w_idx + 2 + WIDTH];
102.            data_out_8 <= buffer[w_idx + 3 + WIDTH];
103.            data_out_9 <= buffer[w_idx + 4 + WIDTH];
104.
105.            data_out_10 <= buffer[w_idx + WIDTH * 2];
106.            data_out_11 <= buffer[w_idx + 1 + WIDTH * 2];
107.            data_out_12 <= buffer[w_idx + 2 + WIDTH * 2];
108.            data_out_13 <= buffer[w_idx + 3 + WIDTH * 2];
109.            data_out_14 <= buffer[w_idx + 4 + WIDTH * 2];
110.
111.            data_out_15 <= buffer[w_idx + WIDTH * 3];
112.            data_out_16 <= buffer[w_idx + 1 + WIDTH * 3];
113.            data_out_17 <= buffer[w_idx + 2 + WIDTH * 3];
114.            data_out_18 <= buffer[w_idx + 3 + WIDTH * 3];
115.            data_out_19 <= buffer[w_idx + 4 + WIDTH * 3];
116.
117.            data_out_20 <= buffer[w_idx + WIDTH * 4];
118.            data_out_21 <= buffer[w_idx + 1 + WIDTH * 4];
119.            data_out_22 <= buffer[w_idx + 2 + WIDTH * 4];
120.            data_out_23 <= buffer[w_idx + 3 + WIDTH * 4];
121.            data_out_24 <= buffer[w_idx + 4 + WIDTH * 4];
122.        end
123.    else if (buf_flag == 3'd1) begin
124.        data_out_0 <= buffer[w_idx + WIDTH];
125.        data_out_1 <= buffer[w_idx + 1 + WIDTH];
126.        data_out_2 <= buffer[w_idx + 2 + WIDTH];
127.        data_out_3 <= buffer[w_idx + 3 + WIDTH];
128.        data_out_4 <= buffer[w_idx + 4 + WIDTH];
129.
130.        data_out_5 <= buffer[w_idx + WIDTH * 2];
131.        data_out_6 <= buffer[w_idx + 1 + WIDTH * 2];
132.        data_out_7 <= buffer[w_idx + 2 + WIDTH * 2];
133.        data_out_8 <= buffer[w_idx + 3 + WIDTH * 2];
134.        data_out_9 <= buffer[w_idx + 4 + WIDTH * 2];
135.
```

```
136.      data_out_10 <= buffer[w_idx + WIDTH * 3];
137.      data_out_11 <= buffer[w_idx + 1 + WIDTH * 3];
138.      data_out_12 <= buffer[w_idx + 2 + WIDTH * 3];
139.      data_out_13 <= buffer[w_idx + 3 + WIDTH * 3];
140.      data_out_14 <= buffer[w_idx + 4 + WIDTH * 3];
141.
142.      data_out_15 <= buffer[w_idx + WIDTH * 4];
143.      data_out_16 <= buffer[w_idx + 1 + WIDTH * 4];
144.      data_out_17 <= buffer[w_idx + 2 + WIDTH * 4];
145.      data_out_18 <= buffer[w_idx + 3 + WIDTH * 4];
146.      data_out_19 <= buffer[w_idx + 4 + WIDTH * 4];
147.
148.      data_out_20 <= buffer[w_idx];
149.      data_out_21 <= buffer[w_idx + 1];
150.      data_out_22 <= buffer[w_idx + 2];
151.      data_out_23 <= buffer[w_idx + 3];
152.      data_out_24 <= buffer[w_idx + 4];
153.      end
154.      else if (buf_flag == 3'd2) begin
155.          data_out_0 <= buffer[w_idx + WIDTH * 2];
156.          data_out_1 <= buffer[w_idx + 1 + WIDTH * 2];
157.          data_out_2 <= buffer[w_idx + 2 + WIDTH * 2];
158.          data_out_3 <= buffer[w_idx + 3 + WIDTH * 2];
159.          data_out_4 <= buffer[w_idx + 4 + WIDTH * 2];
160.
161.          data_out_5 <= buffer[w_idx + WIDTH * 3];
162.          data_out_6 <= buffer[w_idx + 1 + WIDTH * 3];
163.          data_out_7 <= buffer[w_idx + 2 + WIDTH * 3];
164.          data_out_8 <= buffer[w_idx + 3 + WIDTH * 3];
165.          data_out_9 <= buffer[w_idx + 4 + WIDTH * 3];
166.
167.          data_out_10 <= buffer[w_idx + WIDTH * 4];
168.          data_out_11 <= buffer[w_idx + 1 + WIDTH * 4];
169.          data_out_12 <= buffer[w_idx + 2 + WIDTH * 4];
170.          data_out_13 <= buffer[w_idx + 3 + WIDTH * 4];
171.          data_out_14 <= buffer[w_idx + 4 + WIDTH * 4];
172.
173.          data_out_15 <= buffer[w_idx];
174.          data_out_16 <= buffer[w_idx + 1];
175.          data_out_17 <= buffer[w_idx + 2];
176.          data_out_18 <= buffer[w_idx + 3];
177.          data_out_19 <= buffer[w_idx + 4];
178.
179.          data_out_20 <= buffer[w_idx + WIDTH];
```

```
180.      data_out_21 <= buffer[w_idx + 1 + WIDTH];
181.      data_out_22 <= buffer[w_idx + 2 + WIDTH];
182.      data_out_23 <= buffer[w_idx + 3 + WIDTH];
183.      data_out_24 <= buffer[w_idx + 4 + WIDTH];
184.      end
185.      else if (buf_flag == 3'd3) begin
186.          data_out_0 <= buffer[w_idx + WIDTH * 3];
187.          data_out_1 <= buffer[w_idx + 1 + WIDTH * 3];
188.          data_out_2 <= buffer[w_idx + 2 + WIDTH * 3];
189.          data_out_3 <= buffer[w_idx + 3 + WIDTH * 3];
190.          data_out_4 <= buffer[w_idx + 4 + WIDTH * 3];
191.
192.          data_out_5 <= buffer[w_idx + WIDTH * 4];
193.          data_out_6 <= buffer[w_idx + 1 + WIDTH * 4];
194.          data_out_7 <= buffer[w_idx + 2 + WIDTH * 4];
195.          data_out_8 <= buffer[w_idx + 3 + WIDTH * 4];
196.          data_out_9 <= buffer[w_idx + 4 + WIDTH * 4];
197.
198.          data_out_10 <= buffer[w_idx];
199.          data_out_11 <= buffer[w_idx + 1];
200.          data_out_12 <= buffer[w_idx + 2];
201.          data_out_13 <= buffer[w_idx + 3];
202.          data_out_14 <= buffer[w_idx + 4];
203.
204.          data_out_15 <= buffer[w_idx + WIDTH];
205.          data_out_16 <= buffer[w_idx + 1 + WIDTH];
206.          data_out_17 <= buffer[w_idx + 2 + WIDTH];
207.          data_out_18 <= buffer[w_idx + 3 + WIDTH];
208.          data_out_19 <= buffer[w_idx + 4 + WIDTH];
209.
210.          data_out_20 <= buffer[w_idx + WIDTH * 2];
211.          data_out_21 <= buffer[w_idx + 1 + WIDTH * 2];
212.          data_out_22 <= buffer[w_idx + 2 + WIDTH * 2];
213.          data_out_23 <= buffer[w_idx + 3 + WIDTH * 2];
214.          data_out_24 <= buffer[w_idx + 4 + WIDTH * 2];
215.      end
216.      else if (buf_flag == 3'd4) begin
217.          data_out_0 <= buffer[w_idx + WIDTH * 4];
218.          data_out_1 <= buffer[w_idx + 1 + WIDTH * 4];
219.          data_out_2 <= buffer[w_idx + 2 + WIDTH * 4];
220.          data_out_3 <= buffer[w_idx + 3 + WIDTH * 4];
221.          data_out_4 <= buffer[w_idx + 4 + WIDTH * 4];
222.
223.          data_out_5 <= buffer[w_idx];
```



```

224.      data_out_6 <= buffer[w_idx + 1];
225.      data_out_7 <= buffer[w_idx + 2];
226.      data_out_8 <= buffer[w_idx + 3];
227.      data_out_9 <= buffer[w_idx + 4];
228.
229.      data_out_10 <= buffer[w_idx + WIDTH];
230.      data_out_11 <= buffer[w_idx + 1 + WIDTH];
231.      data_out_12 <= buffer[w_idx + 2 + WIDTH];
232.      data_out_13 <= buffer[w_idx + 3 + WIDTH];
233.      data_out_14 <= buffer[w_idx + 4 + WIDTH];
234.
235.      data_out_15 <= buffer[w_idx + WIDTH * 2];
236.      data_out_16 <= buffer[w_idx + 1 + WIDTH * 2];
237.      data_out_17 <= buffer[w_idx + 2 + WIDTH * 2];
238.      data_out_18 <= buffer[w_idx + 3 + WIDTH * 2];
239.      data_out_19 <= buffer[w_idx + 4 + WIDTH * 2];
240.
241.      data_out_20 <= buffer[w_idx + WIDTH * 3];
242.      data_out_21 <= buffer[w_idx + 1 + WIDTH * 3];
243.      data_out_22 <= buffer[w_idx + 2 + WIDTH * 3];
244.      data_out_23 <= buffer[w_idx + 3 + WIDTH * 3];
245.      data_out_24 <= buffer[w_idx + 4 + WIDTH * 3];
246.      end
247.
248.      end
249.
250.      end
251.
252.      end
253.
254. endmodule

```

```

1.  `timescale 1ns / 1ps
2.
3.  // Design      : 1st Convolution Layer for CNN MNIST dataset
4.  //              Convolution Sum Calculation
5.
6.  module conv1_calc #(parameter WIDTH = 28, HEIGHT = 28, DATA_BITS = 8)
7.  (
8.      input clk,
9.      input valid_out_buf,
10.     input [DATA_BITS - 1:0] data_out_0, data_out_1, data_out_2, data_out_3, data_out_4,
11.     data_out_5, data_out_6, data_out_7, data_out_8, data_out_9,

```

```

11.     data_out_10, data_out_11, data_out_12, data_out_13, data_out_14,
12.     data_out_15, data_out_16, data_out_17, data_out_18, data_out_19,
13.     data_out_20, data_out_21, data_out_22, data_out_23, data_out_24,
14.     output signed [11:0] conv_out_1, conv_out_2, conv_out_3,
15.     output valid_out_calc
16. );
17.
18.     localparam FILTER_SIZE = 5;
19.     localparam CHANNEL_LEN = 3;
20.
21.     wire signed [DATA_BITS - 1:0] weight_1 [0:FILTER_SIZE * FILTER_SIZE -
1];
22.     wire signed [DATA_BITS - 1:0] weight_2 [0:FILTER_SIZE * FILTER_SIZE -
1];
23.     wire signed [DATA_BITS - 1:0] weight_3 [0:FILTER_SIZE * FILTER_SIZE -
1];
24.     wire signed [DATA_BITS - 1:0] bias [0:CHANNEL_LEN - 1];
25.
26.     assign weight_1[0] = 8'h06;
27.     assign weight_1[1] = 8'h00;
28.     assign weight_1[2] = 8'h1b;
29.     assign weight_1[3] = 8'h29;
30.     assign weight_1[4] = 8'h35;
31.     assign weight_1[5] = 8'he7;
32.     assign weight_1[6] = 8'h13;
33.     assign weight_1[7] = 8'hfd;
34.     assign weight_1[8] = 8'h0b;
35.     assign weight_1[9] = 8'h3e;
36.     assign weight_1[10] = 8'hf1;
37.     assign weight_1[11] = 8'hf3;
38.     assign weight_1[12] = 8'h0e;
39.     assign weight_1[13] = 8'hf5;
40.     assign weight_1[14] = 8'hf6;
41.     assign weight_1[15] = 8'hd2;
42.     assign weight_1[16] = 8'hd8;
43.     assign weight_1[17] = 8'hdb;
44.     assign weight_1[18] = 8'hd6;
45.     assign weight_1[19] = 8'hd9;
46.     assign weight_1[20] = 8'he7;
47.     assign weight_1[21] = 8'he3;
48.     assign weight_1[22] = 8'hd6;
49.     assign weight_1[23] = 8'he2;
50.     assign weight_1[24] = 8'hdc;
51.

```

```
52.  assign weight_2[0] = 8'h08;
53.  assign weight_2[1] = 8'h00;
54.  assign weight_2[2] = 8'h06;
55.  assign weight_2[3] = 8'hf5;
56.  assign weight_2[4] = 8'hfd;
57.  assign weight_2[5] = 8'h1a;
58.  assign weight_2[6] = 8'hf0;
59.  assign weight_2[7] = 8'h19;
60.  assign weight_2[8] = 8'h0a;
61.  assign weight_2[9] = 8'h24;
62.  assign weight_2[10] = 8'h0f;
63.  assign weight_2[11] = 8'h12;
64.  assign weight_2[12] = 8'h21;
65.  assign weight_2[13] = 8'h46;
66.  assign weight_2[14] = 8'h41;
67.  assign weight_2[15] = 8'h0d;
68.  assign weight_2[16] = 8'h3a;
69.  assign weight_2[17] = 8'h6c;
70.  assign weight_2[18] = 8'h76;
71.  assign weight_2[19] = 8'h2e;
72.  assign weight_2[20] = 8'h31;
73.  assign weight_2[21] = 8'h1d;
74.  assign weight_2[22] = 8'h58;
75.  assign weight_2[23] = 8'h35;
76.  assign weight_2[24] = 8'h4e;
77.
78.  assign weight_3[0] = 8'h21;
79.  assign weight_3[1] = 8'h1a;
80.  assign weight_3[2] = 8'h33;
81.  assign weight_3[3] = 8'h29;
82.  assign weight_3[4] = 8'h47;
83.  assign weight_3[5] = 8'h1b;
84.  assign weight_3[6] = 8'h1b;
85.  assign weight_3[7] = 8'h20;
86.  assign weight_3[8] = 8'h0c;
87.  assign weight_3[9] = 8'hf8;
88.  assign weight_3[10] = 8'hd8;
89.  assign weight_3[11] = 8'h02;
90.  assign weight_3[12] = 8'he0;
91.  assign weight_3[13] = 8'hdd;
92.  assign weight_3[14] = 8'hd5;
93.  assign weight_3[15] = 8'hdc;
94.  assign weight_3[16] = 8'hc3;
95.  assign weight_3[17] = 8'hce;
```

```

96.    assign weight_3[18] = 8'hbc;
97.    assign weight_3[19] = 8'he5;
98.    assign weight_3[20] = 8'hd5;
99.    assign weight_3[21] = 8'hd9;
100.   assign weight_3[22] = 8'hda;
101.   assign weight_3[23] = 8'h03;
102.   assign weight_3[24] = 8'hf9;
103.
104.   assign bias[0] = 8'h07;
105.   assign bias[1] = 8'h0f;
106.   assign bias[2] = 8'h2f;
107.
108.   //integer is_fill = 0;
109.
110.   //always @(*) begin
111.   //if(is_fill == 0) begin
112.   //    weight_1[0] <= 8'h06;
113.   //    ...
114.   //    weight_1[24] <= 8'hdc;
115.
116.   //    weight_2[0] <= 8'h08;
117.   //    ...
118.   //    weight_2[24] <= 8'h4e;
119.
120.   //    weight_3[0] <= 8'h21;
121.   //    ...
122.   //    weight_3[24] <= 8'hf9;
123.
124.   //    bias[0] <= 8'h07;
125.   //    bias[1] <= 8'h0f;
126.   //    bias[2] <= 8'h2f;
127.
128.   //    is_fill = 1;
129.
130.   //    end
131.   //    end
132.
133.
134.   wire signed [19:0] calc_out_1, calc_out_2, calc_out_3;
135.   wire signed [DATA_BITS:0] exp_data [0:FILTER_SIZE * FILTER_SIZE - 1];
136.   wire signed [11:0] exp_bias [0:CHANNEL_LEN - 1];
137.
138.   // initial begin
139.   //    $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/

```

```

conv1_weight_1.txt", weight_1);
140. // $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/
conv1_weight_2.txt", weight_2);
141. // $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/
conv1_weight_3.txt", weight_3);
142. // $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/
conv1_bias.txt", bias);
143. // end
144.
145.
146.
147. // Unsigned -> Signed
148. assign exp_data[0] = {1'd0, data_out_0};
149. assign exp_data[1] = {1'd0, data_out_1};
150. assign exp_data[2] = {1'd0, data_out_2};
151. assign exp_data[3] = {1'd0, data_out_3};
152. assign exp_data[4] = {1'd0, data_out_4};
153. assign exp_data[5] = {1'd0, data_out_5};
154. assign exp_data[6] = {1'd0, data_out_6};
155. assign exp_data[7] = {1'd0, data_out_7};
156. assign exp_data[8] = {1'd0, data_out_8};
157. assign exp_data[9] = {1'd0, data_out_9};
158. assign exp_data[10] = {1'd0, data_out_10};
159. assign exp_data[11] = {1'd0, data_out_11};
160. assign exp_data[12] = {1'd0, data_out_12};
161. assign exp_data[13] = {1'd0, data_out_13};
162. assign exp_data[14] = {1'd0, data_out_14};
163. assign exp_data[15] = {1'd0, data_out_15};
164. assign exp_data[16] = {1'd0, data_out_16};
165. assign exp_data[17] = {1'd0, data_out_17};
166. assign exp_data[18] = {1'd0, data_out_18};
167. assign exp_data[19] = {1'd0, data_out_19};
168. assign exp_data[20] = {1'd0, data_out_20};
169. assign exp_data[21] = {1'd0, data_out_21};
170. assign exp_data[22] = {1'd0, data_out_22};
171. assign exp_data[23] = {1'd0, data_out_23};
172. assign exp_data[24] = {1'd0, data_out_24};
173.
174. // Re-calibration of extracted weight data according to MSB
175. assign exp_bias[0] = (bias[0][7] == 1) ? {4'b1111, bias[0]} : {4'd0,
bias[0]};
176. assign exp_bias[1] = (bias[1][7] == 1) ? {4'b1111, bias[1]} : {4'd0,
bias[1]};
177. assign exp_bias[2] = (bias[2][7] == 1) ? {4'b1111, bias[2]} : {4'd0,

```

```

    bias[2]};

178.
179.  assign calc_out_1 = exp_data[0]*weight_1[0] + exp_data[1]*weight_1[1]
    + exp_data[2]*weight_1[2] + exp_data[3]*weight_1[3] + exp_data[4]*weight_1[4] +
180.      exp_data[5]*weight_1[5] + exp_data[6]*weight_1[6] + exp_data[7]*weight_1[7] + exp_data[8]*weight_1[8] + exp_data[9]*weight_1[9] +
181.      exp_data[10]*weight_1[10] + exp_data[11]*weight_1[11] + exp_data[12]*weight_1[12] + exp_data[13]*weight_1[13] + exp_data[14]*weight_1[14]
    +
182.      exp_data[15]*weight_1[15] + exp_data[16]*weight_1[16] + exp_data[17]*weight_1[17] + exp_data[18]*weight_1[18] + exp_data[19]*weight_1[19]
    +
183.      exp_data[20]*weight_1[20] + exp_data[21]*weight_1[21] + exp_data[22]*weight_1[22] + exp_data[23]*weight_1[23] + exp_data[24]*weight_1[24];
184.
185.  assign calc_out_2 = exp_data[0]*weight_2[0] + exp_data[1]*weight_2[1]
    + exp_data[2]*weight_2[2] + exp_data[3]*weight_2[3] + exp_data[4]*weight_2[4] +
186.      exp_data[5]*weight_2[5] + exp_data[6]*weight_2[6] + exp_data[7]*weight_2[7] + exp_data[8]*weight_2[8] + exp_data[9]*weight_2[9] +
187.      exp_data[10]*weight_2[10] + exp_data[11]*weight_2[11] + exp_data[12]*weight_2[12] + exp_data[13]*weight_2[13] + exp_data[14]*weight_2[14]
    +
188.      exp_data[15]*weight_2[15] + exp_data[16]*weight_2[16] + exp_data[17]*weight_2[17] + exp_data[18]*weight_2[18] + exp_data[19]*weight_2[19]
    +
189.      exp_data[20]*weight_2[20] + exp_data[21]*weight_2[21] + exp_data[22]*weight_2[22] + exp_data[23]*weight_2[23] + exp_data[24]*weight_2[24];
190.
191.  assign calc_out_3 = exp_data[0]*weight_3[0] + exp_data[1]*weight_3[1]
    + exp_data[2]*weight_3[2] + exp_data[3]*weight_3[3] + exp_data[4]*weight_3[4] +
192.      exp_data[5]*weight_3[5] + exp_data[6]*weight_3[6] + exp_data[7]*weight_3[7] + exp_data[8]*weight_3[8] + exp_data[9]*weight_3[9] +
193.      exp_data[10]*weight_3[10] + exp_data[11]*weight_3[11] + exp_data[12]*weight_3[12] + exp_data[13]*weight_3[13] + exp_data[14]*weight_3[14]
    +
194.      exp_data[15]*weight_3[15] + exp_data[16]*weight_3[16] + exp_data[17]*weight_3[17] + exp_data[18]*weight_3[18] + exp_data[19]*weight_3[19]
    +
195.      exp_data[20]*weight_3[20] + exp_data[21]*weight_3[21] + exp_data[22]*weight_3[22] + exp_data[23]*weight_3[23] + exp_data[24]*weight_3[24]
    ;

```

```

196.
197. assign conv_out_1 = calc_out_1[19:8] + exp_bias[0];
198. assign conv_out_2 = calc_out_2[19:8] + exp_bias[1];
199. assign conv_out_3 = calc_out_3[19:8] + exp_bias[2];
200.
201. assign valid_out_calc = valid_out_buf;
202.
203. // ila_0 debugg (
204. //     .clk(clk),
205. //     .probe0(valid_out_buf),
206. //     .probe1(valid_out_calc),
207. //     .probe2(data_out_0),
208. //     .probe3(conv_out_1),
209. //     .probe4(weight_1[0]),
210. //     .probe5(weight_2[0]),
211. //     .probe6(weight_3[0]),
212. //     .probe7(bias[0])
213. // );
214.
215. endmodule

```

```

1.  `timescale 1ns / 1ps
2.
3.  // Design      : (1) MaxPooling for CNN
4.  // (2) Activation Function for CNN - ReLU Function
5.
6.  module maxpool_relu #(parameter CONV_BIT = 12, HALF_WIDTH = 12, HALF_HEIGHT = 12, HALF_WIDTH_BIT = 4) (
7.      input clk,
8.      input rst_n, // asynchronous reset, active low
9.      input valid_in,
10.     input signed [CONV_BIT - 1 : 0] conv_out_1, conv_out_2, conv_out_3,
11.     output reg [CONV_BIT - 1 : 0] max_value_1, max_value_2, max_value_3,
12.     output reg valid_out_relu
13. );
14.
15. reg signed [CONV_BIT - 1:0] buffer1 [0:HALF_WIDTH - 1];
16. reg signed [CONV_BIT - 1:0] buffer2 [0:HALF_WIDTH - 1];
17. reg signed [CONV_BIT - 1:0] buffer3 [0:HALF_WIDTH - 1];
18.
19. reg [HALF_WIDTH_BIT - 1:0] pcount;
20. reg state;
21. reg flag;
22.

```

```

23. always @(posedge clk) begin
24.     if(~rst_n) begin
25.         valid_out_relu <= 0;
26.         pcount <= 0;
27.         state <= 0;
28.         flag <= 0;
29.     end
30. else begin
31.
32.     if(valid_in == 1'b1) begin
33.         flag <= ~flag;
34.
35.         if(flag == 1) begin
36.             pcount <= pcount + 1;
37.             if(pcount == HALF_WIDTH - 1) begin
38.                 state <= ~state;
39.                 pcount <= 0;
40.             end
41.         end
42.
43.         if(state == 0) begin // first line
44.             valid_out_relu <= 0;
45.             if(flag == 0) begin // first input
46.                 buffer1[pcount] <= conv_out_1;
47.                 buffer2[pcount] <= conv_out_2;
48.                 buffer3[pcount] <= conv_out_3;
49.             end
50.         else begin // second input -> comparison
51.             if(buffer1[pcount] < conv_out_1)
52.                 buffer1[pcount] <= conv_out_1;
53.             if(buffer2[pcount] < conv_out_2)
54.                 buffer2[pcount] <= conv_out_2;
55.             if(buffer3[pcount] < conv_out_3)
56.                 buffer3[pcount] <= conv_out_3;
57.         end
58.     end
59. else begin // second line
60.     if(flag == 0) begin // third input -> comparison
61.         valid_out_relu <= 0;
62.         if(buffer1[pcount] < conv_out_1)
63.             buffer1[pcount] <= conv_out_1;
64.         if(buffer2[pcount] < conv_out_2)
65.             buffer2[pcount] <= conv_out_2;
66.         if(buffer3[pcount] < conv_out_3)

```



```
67.     buffer3[pcount] <= conv_out_3;
68.     end
69. else begin // fourth input -> comparison + relu
70.     valid_out_relu <= 1;
71.     if(buffer1[pcount] < conv_out_1) begin
72.         if(conv_out_1 > 0)
73.             max_value_1 <= conv_out_1;
74.         else
75.             max_value_1 <= 0;
76.         end
77.     else begin
78.         if(buffer1[pcount] > 0)
79.             max_value_1 <= buffer1[pcount];
80.         else
81.             max_value_1 <= 0;
82.         end
83.
84.     if(buffer2[pcount] < conv_out_2) begin
85.         if(conv_out_2 > 0)
86.             max_value_2 <= conv_out_2;
87.         else
88.             max_value_2 <= 0;
89.         end
90.     else begin
91.         if(buffer2[pcount] > 0)
92.             max_value_2 <= buffer2[pcount];
93.         else
94.             max_value_2 <= 0;
95.         end
96.
97.     if(buffer3[pcount] < conv_out_3) begin
98.         if(conv_out_3 > 0)
99.             max_value_3 <= conv_out_3;
100.        else
101.            max_value_3 <= 0;
102.        end
103.    else begin
104.        if(buffer3[pcount] > 0)
105.            max_value_3 <= buffer3[pcount];
106.        else
107.            max_value_3 <= 0;
108.        end
109.
110.    end
```

```

111.     end
112.     end
113. //     end
114. else
115.     valid_out_relu <= 0;
116.     end
117. end
118. endmodule

```

```

1.  `timescale 1ns / 1ps
2.
3.  // Design      : 2nd Convolution Layer for CNN MNIST dataset
4.
5.  module conv2_layer (
6.      input clk,
7.      input rst_n,
8.      input valid_in,
9.      input [11:0] max_value_1, max_value_2, max_value_3,
10.     output [11:0] conv2_out_1, conv2_out_2, conv2_out_3,
11.     output valid_out_conv2
12. );
13.
14.     localparam CHANNEL_LEN = 3;
15.     //////////////////////////////////
16.     /*wire [11:0] out_data1_0, out_data1_1, out_data1_2, out_data1_3, out_data1_4,
17.         out_data1_5, out_data1_6, out_data1_7, out_data1_8, out_data1_9,
18.         out_data1_10, out_data1_11, out_data1_12, out_data1_13, out_data1_14,
19.         out_data1_15, out_data1_16, out_data1_17, out_data1_18, out_data1_19,
20.         out_data1_20, out_data1_21, out_data1_22, out_data1_23, out_data1_24,
21.
22.         out_data2_0, out_data2_1, out_data2_2, out_data2_3, out_data2_4,
23.         out_data2_5, out_data2_6, out_data2_7, out_data2_8, out_data2_9,
24.         out_data2_10, out_data2_11, out_data2_12, out_data2_13, out_data2_14,
25.         out_data2_15, out_data2_16, out_data2_17, out_data2_18, out_data2_19,
26.         out_data2_20, out_data2_21, out_data2_22, out_data2_23, out_data2_24,
27.
28.         out_data3_0, out_data3_1, out_data3_2, out_data3_3, out_data3_4,

```

```

29.     out_data3_5, out_data3_6, out_data3_7, out_data3_8, out_data3_9,
30.     out_data3_10, out_data3_11, out_data3_12, out_data3_13, out_data3_1
    4,
31.     out_data3_15, out_data3_16, out_data3_17, out_data3_18, out_data3_1
    9,
32.     out_data3_20, out_data3_21, out_data3_22, out_data3_23, out_data3_2
    4;*/
33.     //////////////////////////////////////
34.     // Channel 1
35.     wire [11:0] data_out1_0, data_out1_1, data_out1_2, data_out1_3, data_
    out1_4,
36.     data_out1_5, data_out1_6, data_out1_7, data_out1_8, data_out1_9,
37.     data_out1_10, data_out1_11, data_out1_12, data_out1_13, data_out1_14
    ,
38.     data_out1_15, data_out1_16, data_out1_17, data_out1_18, data_out1_19
    ,
39.     data_out1_20, data_out1_21, data_out1_22, data_out1_23, data_out1_24
    ;
40.     wire valid_out1_buf;
41.
42.     // Channel 2
43.     wire [11:0] data_out2_0, data_out2_1, data_out2_2, data_out2_3, data_
    out2_4,
44.     data_out2_5, data_out2_6, data_out2_7, data_out2_8, data_out2_9,
45.     data_out2_10, data_out2_11, data_out2_12, data_out2_13, data_out2_14
    ,
46.     data_out2_15, data_out2_16, data_out2_17, data_out2_18, data_out2_19
    ,
47.     data_out2_20, data_out2_21, data_out2_22, data_out2_23, data_out2_24
    ;
48.     wire valid_out2_buf;
49.
50.     // Channel 3
51.     wire [11:0] data_out3_0, data_out3_1, data_out3_2, data_out3_3, data_
    out3_4,
52.     data_out3_5, data_out3_6, data_out3_7, data_out3_8, data_out3_9,
53.     data_out3_10, data_out3_11, data_out3_12, data_out3_13, data_out3_14
    ,
54.     data_out3_15, data_out3_16, data_out3_17, data_out3_18, data_out3_19
    ,
55.     data_out3_20, data_out3_21, data_out3_22, data_out3_23, data_out3_24
    ;
56.     wire valid_out3_buf;
57.

```

```

58.    wire signed [13:0] conv_out_1, conv_out_2, conv_out_3;
59.    wire valid_out_buf, valid_out_calc_1, valid_out_calc_2, valid_out_cal
    c_3;
60.    assign valid_out_buf = valid_out1_buf & valid_out2_buf & valid_out3_b
    uf;
61.    assign valid_out_conv2 = valid_out_calc_1 & valid_out_calc_2 & valid_
    out_calc_3;
62.
63.    wire signed [7:0] bias [0:CHANNEL_LEN - 1];
64.    wire signed [11:0] exp_bias [0:CHANNEL_LEN - 1];
65.
66.    conv2_buf #(.WIDTH(12), .HEIGHT(12), .DATA_BITS(12)) conv2_buf_1(
67.        .clk(clk),
68.        .rst_n(rst_n),
69.        .valid_in(valid_in),
70.        .data_in(max_value_1),
71.        .data_out_0(data_out1_0),
72.        .data_out_1(data_out1_1),
73.        .data_out_2(data_out1_2),
74.        .data_out_3(data_out1_3),
75.        .data_out_4(data_out1_4),
76.        .data_out_5(data_out1_5),
77.        .data_out_6(data_out1_6),
78.        .data_out_7(data_out1_7),
79.        .data_out_8(data_out1_8),
80.        .data_out_9(data_out1_9),
81.        .data_out_10(data_out1_10),
82.        .data_out_11(data_out1_11),
83.        .data_out_12(data_out1_12),
84.        .data_out_13(data_out1_13),
85.        .data_out_14(data_out1_14),
86.        .data_out_15(data_out1_15),
87.        .data_out_16(data_out1_16),
88.        .data_out_17(data_out1_17),
89.        .data_out_18(data_out1_18),
90.        .data_out_19(data_out1_19),
91.        .data_out_20(data_out1_20),
92.        .data_out_21(data_out1_21),
93.        .data_out_22(data_out1_22),
94.        .data_out_23(data_out1_23),
95.        .data_out_24(data_out1_24),
96.        .valid_out_buf(valid_out1_buf)
97.    );
98.

```

```
99.     conv2_buf #(.WIDTH(12), .HEIGHT(12), .DATA_BITS (12)) conv2_buf_2(
100.         .clk(clk),
101.         .rst_n(rst_n),
102.         .valid_in(valid_in),
103.         .data_in(max_value_2),
104.         .data_out_0(data_out2_0),
105.         .data_out_1(data_out2_1),
106.         .data_out_2(data_out2_2),
107.         .data_out_3(data_out2_3),
108.         .data_out_4(data_out2_4),
109.         .data_out_5(data_out2_5),
110.         .data_out_6(data_out2_6),
111.         .data_out_7(data_out2_7),
112.         .data_out_8(data_out2_8),
113.         .data_out_9(data_out2_9),
114.         .data_out_10(data_out2_10),
115.         .data_out_11(data_out2_11),
116.         .data_out_12(data_out2_12),
117.         .data_out_13(data_out2_13),
118.         .data_out_14(data_out2_14),
119.         .data_out_15(data_out2_15),
120.         .data_out_16(data_out2_16),
121.         .data_out_17(data_out2_17),
122.         .data_out_18(data_out2_18),
123.         .data_out_19(data_out2_19),
124.         .data_out_20(data_out2_20),
125.         .data_out_21(data_out2_21),
126.         .data_out_22(data_out2_22),
127.         .data_out_23(data_out2_23),
128.         .data_out_24(data_out2_24),
129.         .valid_out_buf(valid_out2_buf)
130.     );
131.
132.     conv2_buf #(.WIDTH(12), .HEIGHT(12), .DATA_BITS(12)) conv2_buf_3(
133.         .clk(clk),
134.         .rst_n(rst_n),
135.         .valid_in(valid_in),
136.         .data_in(max_value_3),
137.         .data_out_0(data_out3_0),
138.         .data_out_1(data_out3_1),
139.         .data_out_2(data_out3_2),
140.         .data_out_3(data_out3_3),
141.         .data_out_4(data_out3_4),
142.         .data_out_5(data_out3_5),
```

```
143.     .data_out_6(data_out3_6),
144.     .data_out_7(data_out3_7),
145.     .data_out_8(data_out3_8),
146.     .data_out_9(data_out3_9),
147.     .data_out_10(data_out3_10),
148.     .data_out_11(data_out3_11),
149.     .data_out_12(data_out3_12),
150.     .data_out_13(data_out3_13),
151.     .data_out_14(data_out3_14),
152.     .data_out_15(data_out3_15),
153.     .data_out_16(data_out3_16),
154.     .data_out_17(data_out3_17),
155.     .data_out_18(data_out3_18),
156.     .data_out_19(data_out3_19),
157.     .data_out_20(data_out3_20),
158.     .data_out_21(data_out3_21),
159.     .data_out_22(data_out3_22),
160.     .data_out_23(data_out3_23),
161.     .data_out_24(data_out3_24),
162.     .valid_out_buf(valid_out3_buf)
163. );
164.
165. conv2_calc_1 conv2_calc_1(
166.     .clk(clk),
167.     .rst_n(rst_n),
168.     .valid_out_buf(valid_out_buf),
169.     .data_out1_0(data_out1_0),
170.     .data_out1_1(data_out1_1),
171.     .data_out1_2(data_out1_2),
172.     .data_out1_3(data_out1_3),
173.     .data_out1_4(data_out1_4),
174.     .data_out1_5(data_out1_5),
175.     .data_out1_6(data_out1_6),
176.     .data_out1_7(data_out1_7),
177.     .data_out1_8(data_out1_8),
178.     .data_out1_9(data_out1_9),
179.     .data_out1_10(data_out1_10),
180.     .data_out1_11(data_out1_11),
181.     .data_out1_12(data_out1_12),
182.     .data_out1_13(data_out1_13),
183.     .data_out1_14(data_out1_14),
184.     .data_out1_15(data_out1_15),
185.     .data_out1_16(data_out1_16),
186.     .data_out1_17(data_out1_17),
```

187.	.data_out1_18(data_out1_18),
188.	.data_out1_19(data_out1_19),
189.	.data_out1_20(data_out1_20),
190.	.data_out1_21(data_out1_21),
191.	.data_out1_22(data_out1_22),
192.	.data_out1_23(data_out1_23),
193.	.data_out1_24(data_out1_24),
194.	.data_out2_0(data_out2_0),
195.	.data_out2_1(data_out2_1),
196.	.data_out2_2(data_out2_2),
197.	.data_out2_3(data_out2_3),
198.	.data_out2_4(data_out2_4),
199.	.data_out2_5(data_out2_5),
200.	.data_out2_6(data_out2_6),
201.	.data_out2_7(data_out2_7),
202.	.data_out2_8(data_out2_8),
203.	.data_out2_9(data_out2_9),
204.	.data_out2_10(data_out2_10),
205.	.data_out2_11(data_out2_11),
206.	.data_out2_12(data_out2_12),
207.	.data_out2_13(data_out2_13),
208.	.data_out2_14(data_out2_14),
209.	.data_out2_15(data_out2_15),
210.	.data_out2_16(data_out2_16),
211.	.data_out2_17(data_out2_17),
212.	.data_out2_18(data_out2_18),
213.	.data_out2_19(data_out2_19),
214.	.data_out2_20(data_out2_20),
215.	.data_out2_21(data_out2_21),
216.	.data_out2_22(data_out2_22),
217.	.data_out2_23(data_out2_23),
218.	.data_out2_24(data_out2_24),
219.	.data_out3_0(data_out3_0),
220.	.data_out3_1(data_out3_1),
221.	.data_out3_2(data_out3_2),
222.	.data_out3_3(data_out3_3),
223.	.data_out3_4(data_out3_4),
224.	.data_out3_5(data_out3_5),
225.	.data_out3_6(data_out3_6),
226.	.data_out3_7(data_out3_7),
227.	.data_out3_8(data_out3_8),
228.	.data_out3_9(data_out3_9),
229.	.data_out3_10(data_out3_10),
230.	.data_out3_11(data_out3_11),

```
231.     .data_out3_12(data_out3_12),
232.     .data_out3_13(data_out3_13),
233.     .data_out3_14(data_out3_14),
234.     .data_out3_15(data_out3_15),
235.     .data_out3_16(data_out3_16),
236.     .data_out3_17(data_out3_17),
237.     .data_out3_18(data_out3_18),
238.     .data_out3_19(data_out3_19),
239.     .data_out3_20(data_out3_20),
240.     .data_out3_21(data_out3_21),
241.     .data_out3_22(data_out3_22),
242.     .data_out3_23(data_out3_23),
243.     .data_out3_24(data_out3_24),
244.     .conv_out_calc(conv_out_1),
245.     .valid_out_calc(valid_out_calc_1)
246. );
247.
248. conv2_calc_2 conv2_calc_2(
249.     .clk(clk),
250.     .rst_n(rst_n),
251.     .valid_out_buf(valid_out_buf),
252.     .data_out1_0(data_out1_0),
253.     .data_out1_1(data_out1_1),
254.     .data_out1_2(data_out1_2),
255.     .data_out1_3(data_out1_3),
256.     .data_out1_4(data_out1_4),
257.     .data_out1_5(data_out1_5),
258.     .data_out1_6(data_out1_6),
259.     .data_out1_7(data_out1_7),
260.     .data_out1_8(data_out1_8),
261.     .data_out1_9(data_out1_9),
262.     .data_out1_10(data_out1_10),
263.     .data_out1_11(data_out1_11),
264.     .data_out1_12(data_out1_12),
265.     .data_out1_13(data_out1_13),
266.     .data_out1_14(data_out1_14),
267.     .data_out1_15(data_out1_15),
268.     .data_out1_16(data_out1_16),
269.     .data_out1_17(data_out1_17),
270.     .data_out1_18(data_out1_18),
271.     .data_out1_19(data_out1_19),
272.     .data_out1_20(data_out1_20),
273.     .data_out1_21(data_out1_21),
274.     .data_out1_22(data_out1_22),
```


275.	.data_out1_23(data_out1_23),
276.	.data_out1_24(data_out1_24),
277.	.data_out2_0(data_out2_0),
278.	.data_out2_1(data_out2_1),
279.	.data_out2_2(data_out2_2),
280.	.data_out2_3(data_out2_3),
281.	.data_out2_4(data_out2_4),
282.	.data_out2_5(data_out2_5),
283.	.data_out2_6(data_out2_6),
284.	.data_out2_7(data_out2_7),
285.	.data_out2_8(data_out2_8),
286.	.data_out2_9(data_out2_9),
287.	.data_out2_10(data_out2_10),
288.	.data_out2_11(data_out2_11),
289.	.data_out2_12(data_out2_12),
290.	.data_out2_13(data_out2_13),
291.	.data_out2_14(data_out2_14),
292.	.data_out2_15(data_out2_15),
293.	.data_out2_16(data_out2_16),
294.	.data_out2_17(data_out2_17),
295.	.data_out2_18(data_out2_18),
296.	.data_out2_19(data_out2_19),
297.	.data_out2_20(data_out2_20),
298.	.data_out2_21(data_out2_21),
299.	.data_out2_22(data_out2_22),
300.	.data_out2_23(data_out2_23),
301.	.data_out2_24(data_out2_24),
302.	.data_out3_0(data_out3_0),
303.	.data_out3_1(data_out3_1),
304.	.data_out3_2(data_out3_2),
305.	.data_out3_3(data_out3_3),
306.	.data_out3_4(data_out3_4),
307.	.data_out3_5(data_out3_5),
308.	.data_out3_6(data_out3_6),
309.	.data_out3_7(data_out3_7),
310.	.data_out3_8(data_out3_8),
311.	.data_out3_9(data_out3_9),
312.	.data_out3_10(data_out3_10),
313.	.data_out3_11(data_out3_11),
314.	.data_out3_12(data_out3_12),
315.	.data_out3_13(data_out3_13),
316.	.data_out3_14(data_out3_14),
317.	.data_out3_15(data_out3_15),
318.	.data_out3_16(data_out3_16),

```
319.     .data_out3_17(data_out3_17),
320.     .data_out3_18(data_out3_18),
321.     .data_out3_19(data_out3_19),
322.     .data_out3_20(data_out3_20),
323.     .data_out3_21(data_out3_21),
324.     .data_out3_22(data_out3_22),
325.     .data_out3_23(data_out3_23),
326.     .data_out3_24(data_out3_24),
327.     .conv_out_calc(conv_out_2),
328.     .valid_out_calc(valid_out_calc_2)
329. );
330.
331. conv2_calc_3 conv2_calc_3(
332.     .clk(clk),
333.     .rst_n(rst_n),
334.     .valid_out_buf(valid_out_buf),
335.     .data_out1_0(data_out1_0),
336.     .data_out1_1(data_out1_1),
337.     .data_out1_2(data_out1_2),
338.     .data_out1_3(data_out1_3),
339.     .data_out1_4(data_out1_4),
340.     .data_out1_5(data_out1_5),
341.     .data_out1_6(data_out1_6),
342.     .data_out1_7(data_out1_7),
343.     .data_out1_8(data_out1_8),
344.     .data_out1_9(data_out1_9),
345.     .data_out1_10(data_out1_10),
346.     .data_out1_11(data_out1_11),
347.     .data_out1_12(data_out1_12),
348.     .data_out1_13(data_out1_13),
349.     .data_out1_14(data_out1_14),
350.     .data_out1_15(data_out1_15),
351.     .data_out1_16(data_out1_16),
352.     .data_out1_17(data_out1_17),
353.     .data_out1_18(data_out1_18),
354.     .data_out1_19(data_out1_19),
355.     .data_out1_20(data_out1_20),
356.     .data_out1_21(data_out1_21),
357.     .data_out1_22(data_out1_22),
358.     .data_out1_23(data_out1_23),
359.     .data_out1_24(data_out1_24),
360.     .data_out2_0(data_out2_0),
361.     .data_out2_1(data_out2_1),
362.     .data_out2_2(data_out2_2),
```

```
363. .data_out2_3(data_out2_3),
364. .data_out2_4(data_out2_4),
365. .data_out2_5(data_out2_5),
366. .data_out2_6(data_out2_6),
367. .data_out2_7(data_out2_7),
368. .data_out2_8(data_out2_8),
369. .data_out2_9(data_out2_9),
370. .data_out2_10(data_out2_10),
371. .data_out2_11(data_out2_11),
372. .data_out2_12(data_out2_12),
373. .data_out2_13(data_out2_13),
374. .data_out2_14(data_out2_14),
375. .data_out2_15(data_out2_15),
376. .data_out2_16(data_out2_16),
377. .data_out2_17(data_out2_17),
378. .data_out2_18(data_out2_18),
379. .data_out2_19(data_out2_19),
380. .data_out2_20(data_out2_20),
381. .data_out2_21(data_out2_21),
382. .data_out2_22(data_out2_22),
383. .data_out2_23(data_out2_23),
384. .data_out2_24(data_out2_24),
385. .data_out3_0(data_out3_0),
386. .data_out3_1(data_out3_1),
387. .data_out3_2(data_out3_2),
388. .data_out3_3(data_out3_3),
389. .data_out3_4(data_out3_4),
390. .data_out3_5(data_out3_5),
391. .data_out3_6(data_out3_6),
392. .data_out3_7(data_out3_7),
393. .data_out3_8(data_out3_8),
394. .data_out3_9(data_out3_9),
395. .data_out3_10(data_out3_10),
396. .data_out3_11(data_out3_11),
397. .data_out3_12(data_out3_12),
398. .data_out3_13(data_out3_13),
399. .data_out3_14(data_out3_14),
400. .data_out3_15(data_out3_15),
401. .data_out3_16(data_out3_16),
402. .data_out3_17(data_out3_17),
403. .data_out3_18(data_out3_18),
404. .data_out3_19(data_out3_19),
405. .data_out3_20(data_out3_20),
406. .data_out3_21(data_out3_21),
```

```

407.     .data_out3_22(data_out3_22),
408.     .data_out3_23(data_out3_23),
409.     .data_out3_24(data_out3_24),
410.     .conv_out_calc(conv_out_3),
411.     .valid_out_calc(valid_out_calc_3)
412. );
413.
414.     assign bias[0] = 8'h0f;
415.     assign bias[1] = 8'hd5;
416.     assign bias[2] = 8'h13;
417.
418. // integer is_fill = 0;
419.
420. //always @(*) begin
421. //if(is_fill == 0) begin
422. ////    $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/
        a/conv2_bias.txt", bias);
423. //    bias[0] <= 8'h0f;
424. //    bias[1] <= 8'hd5;
425. //    bias[2] <= 8'h13;
426. //    is_fill <= 1;
427. //    end
428. // end
429.
430.     assign exp_bias[0] = (bias[0][7] == 1) ? {4'b1111, bias[0]} : {4'b0000, bias[0]};
431.     assign exp_bias[1] = (bias[1][7] == 1) ? {4'b1111, bias[1]} : {4'b0000, bias[1]};
432.     assign exp_bias[2] = (bias[2][7] == 1) ? {4'b1111, bias[2]} : {4'b0000, bias[2]};
433.
434.     assign conv2_out_1 = conv_out_1[13:1] + exp_bias[0];
435.     assign conv2_out_2 = conv_out_2[13:1] + exp_bias[1];
436.     assign conv2_out_3 = conv_out_3[13:1] + exp_bias[2];
437.
438. endmodule

```

```

1.     `timescale 1ns / 1ps
2.
3. // Design      : 2nd Convolution Layer for CNN MNIST dataset Input Buffer
        fer
4.
5.     module conv2_buf #(parameter WIDTH = 12, HEIGHT = 12, DATA_BITS = 12)
        (

```

```

6.     input clk,
7.     input rst_n,
8.     input valid_in,
9.     input [DATA_BITS - 1:0] data_in,
10.    output reg [DATA_BITS - 1:0] data_out_0, data_out_1, data_out_2, data_out_3, data_out_4,
11.    data_out_5, data_out_6, data_out_7, data_out_8, data_out_9,
12.    data_out_10, data_out_11, data_out_12, data_out_13, data_out_14,
13.    data_out_15, data_out_16, data_out_17, data_out_18, data_out_19,
14.    data_out_20, data_out_21, data_out_22, data_out_23, data_out_24,
15.    output reg valid_out_buf
16. );
17.
18.    localparam FILTER_SIZE = 5;
19.
20.    reg [DATA_BITS - 1:0] buffer [0:WIDTH * FILTER_SIZE - 1];
21.    reg [DATA_BITS - 1:0] buf_idx;
22.    reg [4:0] w_idx, h_idx;
23.    reg [2:0] buf_flag; // 0 ~ 4
24.    reg state;
25.
26.    always @(posedge clk) begin
27.        if(~rst_n) begin
28.            buf_idx <= 0;
29.            w_idx <= 0;
30.            h_idx <= 0;
31.            buf_flag <= 0;
32.            state <= 0;
33.            valid_out_buf <= 0;
34.            data_out_0 <= 12'bx;
35.            data_out_1 <= 12'bx;
36.            data_out_2 <= 12'bx;
37.            data_out_3 <= 12'bx;
38.            data_out_4 <= 12'bx;
39.            data_out_5 <= 12'bx;
40.            data_out_6 <= 12'bx;
41.            data_out_7 <= 12'bx;
42.            data_out_8 <= 12'bx;
43.            data_out_9 <= 12'bx;
44.            data_out_10 <= 12'bx;
45.            data_out_11 <= 12'bx;
46.            data_out_12 <= 12'bx;
47.            data_out_13 <= 12'bx;
48.            data_out_14 <= 12'bx;

```

```

49.      data_out_15 <= 12'bx;
50.      data_out_16 <= 12'bx;
51.      data_out_17 <= 12'bx;
52.      data_out_18 <= 12'bx;
53.      data_out_19 <= 12'bx;
54.      data_out_20 <= 12'bx;
55.      data_out_21 <= 12'bx;
56.      data_out_22 <= 12'bx;
57.      data_out_23 <= 12'bx;
58.      data_out_24 <= 12'bx;
59.  end else begin
60.      if(valid_in) begin
61.          buf_idx <= buf_idx + 1'b1;
62.          if(buf_idx == WIDTH * FILTER_SIZE - 1) begin // buffer size = 140
              = 28(w) * 5(h)
63.              buf_idx <= 0;
64.          end
65.
66.          buffer[buf_idx] <= data_in; // data input
67.
68.          // Wait until first 140 input data filled in buffer
69.          if(!state) begin
70.              if(buf_idx == WIDTH * FILTER_SIZE - 1) begin
71.                  state <= 1;
72.              end
73.          end else begin // valid state
74.              w_idx <= w_idx + 1'b1; // move right
75.
76.              if(w_idx == WIDTH - FILTER_SIZE + 1) begin
77.                  valid_out_buf <= 1'b0; // unvalid area
78.              end else if(w_idx == WIDTH - 1) begin
79.                  buf_flag <= buf_flag + 1;
80.                  if(buf_flag == FILTER_SIZE - 1) begin
81.                      buf_flag <= 0;
82.                  end
83.
84.                  w_idx <= 0;
85.
86.                  if(h_idx == HEIGHT - FILTER_SIZE) begin // done 1 input read -
                      > 28 * 28
87.                      h_idx <= 0;
88.                      state <= 0;
89.                  end
90.                  h_idx <= h_idx + 1;

```

```

91.
92.     end else if(w_idx == 0) begin
93.         valid_out_buf <= 1'b1; // start valid area
94.     end
95.
96.     // Buffer Selection -> 5 * 5
97.     if(buf_flag == 3'd0) begin
98.         data_out_0 <= buffer[w_idx];
99.         data_out_1 <= buffer[w_idx + 1];
100.        data_out_2 <= buffer[w_idx + 2];
101.        data_out_3 <= buffer[w_idx + 3];
102.        data_out_4 <= buffer[w_idx + 4];
103.
104.        data_out_5 <= buffer[w_idx + WIDTH];
105.        data_out_6 <= buffer[w_idx + 1 + WIDTH];
106.        data_out_7 <= buffer[w_idx + 2 + WIDTH];
107.        data_out_8 <= buffer[w_idx + 3 + WIDTH];
108.        data_out_9 <= buffer[w_idx + 4 + WIDTH];
109.
110.        data_out_10 <= buffer[w_idx + WIDTH * 2];
111.        data_out_11 <= buffer[w_idx + 1 + WIDTH * 2];
112.        data_out_12 <= buffer[w_idx + 2 + WIDTH * 2];
113.        data_out_13 <= buffer[w_idx + 3 + WIDTH * 2];
114.        data_out_14 <= buffer[w_idx + 4 + WIDTH * 2];
115.
116.        data_out_15 <= buffer[w_idx + WIDTH * 3];
117.        data_out_16 <= buffer[w_idx + 1 + WIDTH * 3];
118.        data_out_17 <= buffer[w_idx + 2 + WIDTH * 3];
119.        data_out_18 <= buffer[w_idx + 3 + WIDTH * 3];
120.        data_out_19 <= buffer[w_idx + 4 + WIDTH * 3];
121.
122.        data_out_20 <= buffer[w_idx + WIDTH * 4];
123.        data_out_21 <= buffer[w_idx + 1 + WIDTH * 4];
124.        data_out_22 <= buffer[w_idx + 2 + WIDTH * 4];
125.        data_out_23 <= buffer[w_idx + 3 + WIDTH * 4];
126.        data_out_24 <= buffer[w_idx + 4 + WIDTH * 4];
127.     end else if(buf_flag == 3'd1) begin
128.         data_out_0 <= buffer[w_idx + WIDTH];
129.         data_out_1 <= buffer[w_idx + 1 + WIDTH];
130.         data_out_2 <= buffer[w_idx + 2 + WIDTH];
131.         data_out_3 <= buffer[w_idx + 3 + WIDTH];
132.         data_out_4 <= buffer[w_idx + 4 + WIDTH];
133.
134.         data_out_5 <= buffer[w_idx + WIDTH * 2];

```

```
135.      data_out_6 <= buffer[w_idx + 1 + WIDTH * 2];
136.      data_out_7 <= buffer[w_idx + 2 + WIDTH * 2];
137.      data_out_8 <= buffer[w_idx + 3 + WIDTH * 2];
138.      data_out_9 <= buffer[w_idx + 4 + WIDTH * 2];
139.
140.      data_out_10 <= buffer[w_idx + WIDTH * 3];
141.      data_out_11 <= buffer[w_idx + 1 + WIDTH * 3];
142.      data_out_12 <= buffer[w_idx + 2 + WIDTH * 3];
143.      data_out_13 <= buffer[w_idx + 3 + WIDTH * 3];
144.      data_out_14 <= buffer[w_idx + 4 + WIDTH * 3];
145.
146.      data_out_15 <= buffer[w_idx + WIDTH * 4];
147.      data_out_16 <= buffer[w_idx + 1 + WIDTH * 4];
148.      data_out_17 <= buffer[w_idx + 2 + WIDTH * 4];
149.      data_out_18 <= buffer[w_idx + 3 + WIDTH * 4];
150.      data_out_19 <= buffer[w_idx + 4 + WIDTH * 4];
151.
152.      data_out_20 <= buffer[w_idx];
153.      data_out_21 <= buffer[w_idx + 1];
154.      data_out_22 <= buffer[w_idx + 2];
155.      data_out_23 <= buffer[w_idx + 3];
156.      data_out_24 <= buffer[w_idx + 4];
157.  end else if(buf_flag == 3'd2) begin
158.      data_out_0 <= buffer[w_idx + WIDTH * 2];
159.      data_out_1 <= buffer[w_idx + 1 + WIDTH * 2];
160.      data_out_2 <= buffer[w_idx + 2 + WIDTH * 2];
161.      data_out_3 <= buffer[w_idx + 3 + WIDTH * 2];
162.      data_out_4 <= buffer[w_idx + 4 + WIDTH * 2];
163.
164.      data_out_5 <= buffer[w_idx + WIDTH * 3];
165.      data_out_6 <= buffer[w_idx + 1 + WIDTH * 3];
166.      data_out_7 <= buffer[w_idx + 2 + WIDTH * 3];
167.      data_out_8 <= buffer[w_idx + 3 + WIDTH * 3];
168.      data_out_9 <= buffer[w_idx + 4 + WIDTH * 3];
169.
170.      data_out_10 <= buffer[w_idx + WIDTH * 4];
171.      data_out_11 <= buffer[w_idx + 1 + WIDTH * 4];
172.      data_out_12 <= buffer[w_idx + 2 + WIDTH * 4];
173.      data_out_13 <= buffer[w_idx + 3 + WIDTH * 4];
174.      data_out_14 <= buffer[w_idx + 4 + WIDTH * 4];
175.
176.      data_out_15 <= buffer[w_idx];
177.      data_out_16 <= buffer[w_idx + 1];
178.      data_out_17 <= buffer[w_idx + 2];
```



```
179.      data_out_18 <= buffer[w_idx + 3];
180.      data_out_19 <= buffer[w_idx + 4];
181.
182.      data_out_20 <= buffer[w_idx + WIDTH];
183.      data_out_21 <= buffer[w_idx + 1 + WIDTH];
184.      data_out_22 <= buffer[w_idx + 2 + WIDTH];
185.      data_out_23 <= buffer[w_idx + 3 + WIDTH];
186.      data_out_24 <= buffer[w_idx + 4 + WIDTH];
187.  end else if(buf_flag == 3'd3) begin
188.      data_out_0 <= buffer[w_idx + WIDTH * 3];
189.      data_out_1 <= buffer[w_idx + 1 + WIDTH * 3];
190.      data_out_2 <= buffer[w_idx + 2 + WIDTH * 3];
191.      data_out_3 <= buffer[w_idx + 3 + WIDTH * 3];
192.      data_out_4 <= buffer[w_idx + 4 + WIDTH * 3];
193.
194.      data_out_5 <= buffer[w_idx + WIDTH * 4];
195.      data_out_6 <= buffer[w_idx + 1 + WIDTH * 4];
196.      data_out_7 <= buffer[w_idx + 2 + WIDTH * 4];
197.      data_out_8 <= buffer[w_idx + 3 + WIDTH * 4];
198.      data_out_9 <= buffer[w_idx + 4 + WIDTH * 4];
199.
200.      data_out_10 <= buffer[w_idx];
201.      data_out_11 <= buffer[w_idx + 1];
202.      data_out_12 <= buffer[w_idx + 2];
203.      data_out_13 <= buffer[w_idx + 3];
204.      data_out_14 <= buffer[w_idx + 4];
205.
206.      data_out_15 <= buffer[w_idx + WIDTH];
207.      data_out_16 <= buffer[w_idx + 1 + WIDTH];
208.      data_out_17 <= buffer[w_idx + 2 + WIDTH];
209.      data_out_18 <= buffer[w_idx + 3 + WIDTH];
210.      data_out_19 <= buffer[w_idx + 4 + WIDTH];
211.
212.      data_out_20 <= buffer[w_idx + WIDTH * 2];
213.      data_out_21 <= buffer[w_idx + 1 + WIDTH * 2];
214.      data_out_22 <= buffer[w_idx + 2 + WIDTH * 2];
215.      data_out_23 <= buffer[w_idx + 3 + WIDTH * 2];
216.      data_out_24 <= buffer[w_idx + 4 + WIDTH * 2];
217.  end else if(buf_flag == 3'd4) begin
218.      data_out_0 <= buffer[w_idx + WIDTH * 4];
219.      data_out_1 <= buffer[w_idx + 1 + WIDTH * 4];
220.      data_out_2 <= buffer[w_idx + 2 + WIDTH * 4];
221.      data_out_3 <= buffer[w_idx + 3 + WIDTH * 4];
222.      data_out_4 <= buffer[w_idx + 4 + WIDTH * 4];
```

```

223.
224.     data_out_5 <= buffer[w_idx];
225.     data_out_6 <= buffer[w_idx + 1];
226.     data_out_7 <= buffer[w_idx + 2];
227.     data_out_8 <= buffer[w_idx + 3];
228.     data_out_9 <= buffer[w_idx + 4];
229.
230.     data_out_10 <= buffer[w_idx + WIDTH];
231.     data_out_11 <= buffer[w_idx + 1 + WIDTH];
232.     data_out_12 <= buffer[w_idx + 2 + WIDTH];
233.     data_out_13 <= buffer[w_idx + 3 + WIDTH];
234.     data_out_14 <= buffer[w_idx + 4 + WIDTH];
235.
236.     data_out_15 <= buffer[w_idx + WIDTH * 2];
237.     data_out_16 <= buffer[w_idx + 1 + WIDTH * 2];
238.     data_out_17 <= buffer[w_idx + 2 + WIDTH * 2];
239.     data_out_18 <= buffer[w_idx + 3 + WIDTH * 2];
240.     data_out_19 <= buffer[w_idx + 4 + WIDTH * 2];
241.
242.     data_out_20 <= buffer[w_idx + WIDTH * 3];
243.     data_out_21 <= buffer[w_idx + 1 + WIDTH * 3];
244.     data_out_22 <= buffer[w_idx + 2 + WIDTH * 3];
245.     data_out_23 <= buffer[w_idx + 3 + WIDTH * 3];
246.     data_out_24 <= buffer[w_idx + 4 + WIDTH * 3];
247.     end
248.     end
249.     end
250. end
251. end
252. endmodule

```

```

1.  `timescale 1ns / 1ps
2.
3.  // Design      : 2nd Convolution Layer for CNN MNIST dataset
4.  //              Convolution Sum Calculation - 1st Channel
5.
6.  module conv2_calc_1(
7.      input clk,
8.      input rst_n,
9.      input valid_out_buf,
10.     input signed [11:0] data_out1_0, data_out1_1, data_out1_2, data_out1_
11.         3, data_out1_4,
12.         data_out1_5, data_out1_6, data_out1_7, data_out1_8, data_out1_9,
13.         data_out1_10, data_out1_11, data_out1_12, data_out1_13, data_out1_

```

```

14,
13.    data_out1_15, data_out1_16, data_out1_17, data_out1_18, data_out1_
19,
14.    data_out1_20, data_out1_21, data_out1_22, data_out1_23, data_out1_
24,
15.
16.    data_out2_0, data_out2_1, data_out2_2, data_out2_3, data_out2_4,
17.    data_out2_5, data_out2_6, data_out2_7, data_out2_8, data_out2_9,
18.    data_out2_10, data_out2_11, data_out2_12, data_out2_13, data_out2_
14,
19.    data_out2_15, data_out2_16, data_out2_17, data_out2_18, data_out2_
19,
20.    data_out2_20, data_out2_21, data_out2_22, data_out2_23, data_out2_
24,
21.
22.    data_out3_0, data_out3_1, data_out3_2, data_out3_3, data_out3_4,
23.    data_out3_5, data_out3_6, data_out3_7, data_out3_8, data_out3_9,
24.    data_out3_10, data_out3_11, data_out3_12, data_out3_13, data_out3_
14,
25.    data_out3_15, data_out3_16, data_out3_17, data_out3_18, data_out3_
19,
26.    data_out3_20, data_out3_21, data_out3_22, data_out3_23, data_out3_
24,
27.
28.    output [13:0] conv_out_calc,
29.    output reg valid_out_calc
30.    );
31.
32.    wire signed [19:0] calc_out, calc_out_1, calc_out_2, calc_out_3;
33.
34.    wire signed [7:0] weight_1 [0:24];
35.    wire signed [7:0] weight_2 [0:24];
36.    wire signed [7:0] weight_3 [0:24];
37.
38.    assign weight_1[0] = 8'h01; //01;
39.    assign weight_1[1] = 8'h03; //03
40.    assign weight_1[2] = 8'h01; //01;
41.    assign weight_1[3] = 8'h0c;
42.    assign weight_1[4] = 8'hfd;
43.    assign weight_1[5] = 8'hf9;
44.    assign weight_1[6] = 8'hf9;
45.    assign weight_1[7] = 8'h0a;
46.    assign weight_1[8] = 8'hf2;
47.    assign weight_1[9] = 8'heb;

```

```
48. assign weight_1[10] = 8'hea;
49. assign weight_1[11] = 8'he9;
50. assign weight_1[12] = 8'hee;
51. assign weight_1[13] = 8'he1;
52. assign weight_1[14] = 8'hf8;
53. assign weight_1[15] = 8'he7;
54. assign weight_1[16] = 8'hd6;
55. assign weight_1[17] = 8'hd7;
56. assign weight_1[18] = 8'hfa;
57. assign weight_1[19] = 8'hf8;
58. assign weight_1[20] = 8'he2;
59. assign weight_1[21] = 8'hd8;
60. assign weight_1[22] = 8'he4;
61. assign weight_1[23] = 8'hfd;
62. assign weight_1[24] = 8'h18;
63.
64. assign weight_2[0] = 8'hfe;
65. assign weight_2[1] = 8'hea;
66. assign weight_2[2] = 8'h05;
67. assign weight_2[3] = 8'hfa;
68. assign weight_2[4] = 8'he4;
69. assign weight_2[5] = 8'h00; //00
70. assign weight_2[6] = 8'hed;
71. assign weight_2[7] = 8'h1f;
72. assign weight_2[8] = 8'h47;
73. assign weight_2[9] = 8'h0c;
74. assign weight_2[10] = 8'hc8;
75. assign weight_2[11] = 8'h1e;
76. assign weight_2[12] = 8'h4a;
77. assign weight_2[13] = 8'h21;
78. assign weight_2[14] = 8'hd6;
79. assign weight_2[15] = 8'h0f;
80. assign weight_2[16] = 8'h32;
81. assign weight_2[17] = 8'h11;
82. assign weight_2[18] = 8'hd6;
83. assign weight_2[19] = 8'hf5;
84. assign weight_2[20] = 8'h2a;
85. assign weight_2[21] = 8'h16;
86. assign weight_2[22] = 8'he6;
87. assign weight_2[23] = 8'hf8;
88. assign weight_2[24] = 8'hf4;
89.
90. assign weight_3[0] = 8'hf0;
91. assign weight_3[1] = 8'heb;
```

```
92. assign weight_3[2] = 8'h0c;
93. assign weight_3[3] = 8'h0b;
94. assign weight_3[4] = 8'hf2;
95. assign weight_3[5] = 8'hfc;
96. assign weight_3[6] = 8'h08;
97. assign weight_3[7] = 8'h14;
98. assign weight_3[8] = 8'he9;
99. assign weight_3[9] = 8'hf5;
100. assign weight_3[10] = 8'h10;
101. assign weight_3[11] = 8'h06;
102. assign weight_3[12] = 8'hdc;
103. assign weight_3[13] = 8'he3;
104. assign weight_3[14] = 8'he5;
105. assign weight_3[15] = 8'h03; //03;
106. assign weight_3[16] = 8'he3;
107. assign weight_3[17] = 8'hf3;
108. assign weight_3[18] = 8'hfa;
109. assign weight_3[19] = 8'hf3;
110. assign weight_3[20] = 8'he7;
111. assign weight_3[21] = 8'hef;
112. assign weight_3[22] = 8'hf6;
113. assign weight_3[23] = 8'h05;
114. assign weight_3[24] = 8'hf5;
115.
116.
117. // integer is_fill = 0;
118.
119. //always @(*) begin
120. //if(is_fill == 0) begin
121. ////    $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/conv2_weight_11.txt", weight_1);
122. ////    $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/conv2_weight_12.txt", weight_2);
123. ////    $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/conv2_weight_13.txt", weight_3);
124. //    weight_1[0] <= 8'h01;
125.
126. //weight_1[1] <= 8'h03;
127. //..
128. //weight_1[24] <= 8'h18;
129.
130. //weight_2[0] <= 8'hfe;
131. //..
132. //weight_2[24] <= 8'hf4;
```

```

133.
134. //weight_3[0] <= 8'hf0;
135. //..
136. //weight_3[24] <= 8'hf5;
137.
138. //is_fill <= 1;
139. //      end
140. //      end
141.
142. assign calc_out_1 = data_out1_0*weight_1[0] + data_out1_1*weight_1[1]
    + data_out1_2*weight_1[2] + data_out1_3*weight_1[3] + data_out1_4*weight
    _1[4] +
143.      data_out1_5*weight_1[5] + data_out1_6*weight_1[6] + data_out1_7*w
    eight_1[7] + data_out1_8*weight_1[8] + data_out1_9*weight_1[9] +
144.      data_out1_10*weight_1[10] + data_out1_11*weight_1[11] + data_out1
    _12*weight_1[12] + data_out1_13*weight_1[13] + data_out1_14*weight_1[14]
    +
145.      data_out1_15*weight_1[15] + data_out1_16*weight_1[16] + data_out1
    _17*weight_1[17] + data_out1_18*weight_1[18] + data_out1_19*weight_1[19]
    +
146.      data_out1_20*weight_1[20] + data_out1_21*weight_1[21] + data_out1
    _22*weight_1[22] + data_out1_23*weight_1[23] + data_out1_24*weight_1[24]
    ;
147.
148. assign calc_out_2 = data_out2_0*weight_2[0] + data_out2_1*weight_2[1]
    + data_out2_2*weight_2[2] + data_out2_3*weight_2[3] + data_out2_4*weight
    _2[4] +
149.      data_out2_5*weight_2[5] + data_out2_6*weight_2[6] + data_out2_7*w
    eight_2[7] + data_out2_8*weight_2[8] + data_out2_9*weight_2[9] +
150.      data_out2_10*weight_2[10] + data_out2_11*weight_2[11] + data_out2
    _12*weight_2[12] + data_out2_13*weight_2[13] + data_out2_14*weight_2[14]
    +
151.      data_out2_15*weight_2[15] + data_out2_16*weight_2[16] + data_out2
    _17*weight_2[17] + data_out2_18*weight_2[18] + data_out2_19*weight_2[19]
    +
152.      data_out2_20*weight_2[20] + data_out2_21*weight_2[21] + data_out2
    _22*weight_2[22] + data_out2_23*weight_2[23] + data_out2_24*weight_2[24]
    ;
153.
154. assign calc_out_3 = data_out3_0*weight_3[0] + data_out3_1*weight_3[1]
    + data_out3_2*weight_3[2] + data_out3_3*weight_3[3] + data_out3_4*weight
    _3[4] +
155.      data_out3_5*weight_3[5] + data_out3_6*weight_3[6] + data_out3_7*w
    eight_3[7] + data_out3_8*weight_3[8] + data_out3_9*weight_3[9] +

```

```

156.      data_out3_10*weight_3[10] + data_out3_11*weight_3[11] + data_out3_
      _12*weight_3[12] + data_out3_13*weight_3[13] + data_out3_14*weight_3[14]
      +
157.      data_out3_15*weight_3[15] + data_out3_16*weight_3[16] + data_out3_
      _17*weight_3[17] + data_out3_18*weight_3[18] + data_out3_19*weight_3[19]
      +
158.      data_out3_20*weight_3[20] + data_out3_21*weight_3[21] + data_out3_
      _22*weight_3[22] + data_out3_23*weight_3[23] + data_out3_24*weight_3[24]
      ;
159.
160. assign calc_out = calc_out_1 + calc_out_2 + calc_out_3;
161.
162.
163. //reg signed [19:0] calc_out, calc_out_1, calc_out_2, calc_out_3;
164.
165. assign conv_out_calc = calc_out[19:6]; // 14bit
166.
167. always @ (posedge clk) begin
168.     if(~rst_n) begin
169.         valid_out_calc <= 0;
170.         //conv_out_calc <= 0;
171.     end
172.     else begin
173.         // Toggling Valid Output Signal
174.         if(valid_out_buf == 1) begin
175.             if(valid_out_calc == 1)
176.                 valid_out_calc <= 0;
177.             else
178.                 valid_out_calc <= 1;
179.         end
180.     end
181. end
182.
183. //integer state;
184. //always @ (posedge clk) begin
185. // if(~rst_n) begin
186. //     valid_out_calc <= 0;
187. //     state <= 0;
188. //     //conv_out_calc <= 0;
189. //     end
190. // else begin
191. //     // Toggling Valid Output Signal
192. //     if(valid_out_buf == 1) begin
193. //         if(valid_out_calc == 1) begin

```

```

194. //          valid_out_calc <= 0;
195. //          state <= 0;
196. //          end
197. //          else if (state == 0) begin
198. //              calc_out_1 <= data_out1_0*weight_1[0] + data_out1_1*wei
                ht_1[1] + data_out1_2*weight_1[2] + data_out1_3*weight_1[3] + data_out1_
                4*weight_1[4] +
199. //              data_out1_5*weight_1[5] + data_out1_6
                *weight_1[6] + data_out1_7*weight_1[7] + data_out1_8*weight_1[8] + data_
                out1_9*weight_1[9] +
200. //              data_out1_10*weight_1[10] + data_out1
                _11*weight_1[11] + data_out1_12*weight_1[12] + data_out1_13*weight_1[13]
                + data_out1_14*weight_1[14] +
201. //              data_out1_15*weight_1[15] + data_out1
                _16*weight_1[16] + data_out1_17*weight_1[17] + data_out1_18*weight_1[18]
                + data_out1_19*weight_1[19] +
202. //              data_out1_20*weight_1[20] + data_out1
                _21*weight_1[21] + data_out1_22*weight_1[22] + data_out1_23*weight_1[23]
                + data_out1_24*weight_1[24];
203. //          state <= 1;
204. //          end
205. //          else if (state == 1) begin
206. //              calc_out_2 <= data_out2_0*weight_2[0] + data_out2_1*w
                eight_2[1] + data_out2_2*weight_2[2] + data_out2_3*weight_2[3] + data_ou
                t2_4*weight_2[4] +
207. //              data_out2_5*weight_2[5] + data_out2_6
                *weight_2[6] + data_out2_7*weight_2[7] + data_out2_8*weight_2[8] + data_
                out2_9*weight_2[9] +
208. //              data_out2_10*weight_2[10] + data_out2
                _11*weight_2[11] + data_out2_12*weight_2[12] + data_out2_13*weight_2[13]
                + data_out2_14*weight_2[14] +
209. //              data_out2_15*weight_2[15] + data_out2
                _16*weight_2[16] + data_out2_17*weight_2[17] + data_out2_18*weight_2[18]
                + data_out2_19*weight_2[19] +
210. //              data_out2_20*weight_2[20] + data_out2
                _21*weight_2[21] + data_out2_22*weight_2[22] + data_out2_23*weight_2[23]
                + data_out2_24*weight_2[24];
211.
212. //          state <= 2;
213. //          end
214. //          else if (state == 2) begin
215. //              calc_out_3 <= data_out3_0*weight_3[0] + data_out3_1*w
                eight_3[1] + data_out3_2*weight_3[2] + data_out3_3*weight_3[3] + data_ou
                t3_4*weight_3[4] +

```



```

216. //                                data_out3_5*weight_3[5] + data_out3_6
    *weight_3[6] + data_out3_7*weight_3[7] + data_out3_8*weight_3[8] + data_
    out3_9*weight_3[9] +
217. //                                data_out3_10*weight_3[10] + data_out3
    _11*weight_3[11] + data_out3_12*weight_3[12] + data_out3_13*weight_3[13]
    + data_out3_14*weight_3[14] +
218. //                                data_out3_15*weight_3[15] + data_out3
    _16*weight_3[16] + data_out3_17*weight_3[17] + data_out3_18*weight_3[18]
    + data_out3_19*weight_3[19] +
219. //                                data_out3_20*weight_3[20] + data_out3
    _21*weight_3[21] + data_out3_22*weight_3[22] + data_out3_23*weight_3[23]
    + data_out3_24*weight_3[24];
220. //                                calc_out <= calc_out_1 + calc_out_2 + calc_out_3;
221. //                                valid_out_calc <= 1;
222. //                                end
223. // end
224. // end
225. //end
226.
227. endmodule

```

```

1.    `timescale 1ns / 1ps
2.
3.    // Design      : Fully Connected Layer for CNN
4.
5.    module fully_connected (/*#(parameter INPUT_NUM = 48, OUTPUT_NUM = 10
    , DATA_BITS = 8) (
6.        input clk,
7.        input rst_n,
8.        input valid_in,
9.        input signed [11:0] data_in_1, data_in_2, data_in_3,
10.       output [11:0] data_out,
11.       output reg valid_out_fc
12.    //    output reg delay
13.    );
14.
15.    parameter INPUT_NUM = 48, OUTPUT_NUM = 10, DATA_BITS = 8;
16.
17.    localparam INPUT_WIDTH = 16;
18.    localparam INPUT_NUM_DATA_BITS = 5;
19.
20.    reg state;
21.    reg [INPUT_WIDTH - 1:0] buf_idx;
22.    reg [3:0] out_idx;

```

```

23.   reg signed [13:0] buffer [0:INPUT_NUM - 1];
24.
25.   wire signed [DATA_BITS - 1:0] weight [0:INPUT_NUM * OUTPUT_NUM - 1];
26.   wire signed [DATA_BITS - 1:0] bias [0:OUTPUT_NUM - 1];
27.
28.   wire signed [19:0] calc_out;
29.   wire signed [13:0] data1, data2, data3;
30.
31.   assign weight[0] = 8'hfe;
32.   //.....
33.   assign weight[479] = 8'h09;
34.
35.   assign bias[0] = 8'hf5;
36.   assign bias[1] = 8'h1d;
37.   assign bias[2] = 8'h09;
38.   assign bias[3] = 8'he0;
39.   assign bias[4] = 8'h00;
40.   assign bias[5] = 8'hfb;
41.   assign bias[6] = 8'h07;
42.   assign bias[7] = 8'h05;
43.   assign bias[8] = 8'hf9;
44.   assign bias[9] = 8'hf4;
45.
46.
47.   //integer is_fill = 0;
48.
49.   //always @(*) begin
50.   //if(is_fill == 0) begin
51.   ///    $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/dat
a/fc_weight.txt", weight);
52.   ///    $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/dat
a/fc_bias.txt", bias);
53.
54.   //weight[0] <= 8'hfe;
55.   //...
56.   //weight[479] <= 8'h09;
57.
58.   //bias[0] <= 8'hf5;
59.   //...
60.   //bias[9] <= 8'hf4;
61.
62.   //is_fill <= 1;
63.
64.   // end

```

```

65. // end
66.
67. assign data1 = (data_in_1[11] == 1) ? {2'b11, data_in_1} : {2'b00, data_in_1};
68. assign data2 = (data_in_2[11] == 1) ? {2'b11, data_in_2} : {2'b00, data_in_2};
69. assign data3 = (data_in_3[11] == 1) ? {2'b11, data_in_3} : {2'b00, data_in_3};
70.
71. always @(posedge clk) begin
72.     if(~rst_n) begin
73.         valid_out_fc <= 0;
74.         buf_idx <= 0;
75.         out_idx <= 0;
76.         state <= 0;
77.     end
78.
79.     if(valid_out_fc == 1) begin
80.         valid_out_fc <= 0;
81.     end
82.
83.     if(valid_in == 1) begin
84.         // Wait until 48 input data filled in buffer
85.         if(!state) begin
86.             buffer[buf_idx] <= data1;
87.             buffer[INPUT_WIDTH + buf_idx] <= data2;
88.             buffer[INPUT_WIDTH * 2 + buf_idx] <= data3;
89.             buf_idx <= buf_idx + 1'b1;
90.             if(buf_idx == INPUT_WIDTH - 1) begin
91.                 buf_idx <= 0;
92.                 state <= 1;
93.                 valid_out_fc <= 1;
94.             end
95.         end else begin // valid state
96.             out_idx <= out_idx + 1'b1;
97.             if(out_idx == OUTPUT_NUM - 1) begin
98.                 out_idx <= 0;
99.             end
100.            valid_out_fc <= 1;
101.        end
102.    end
103. end
104.
105. assign calc_out = weight[out_idx * INPUT_NUM] * buffer[0] + weight[out_idx * INPUT_NUM + 1] * buffer[1] + weight[out_idx * INPUT_NUM + 2] * buffer[2];

```

```

t_idx * INPUT_NUM + 1] * buffer[1] +
106.      weight[out_idx * INPUT_NUM + 2] * buffer[2] + weight[out_idx * I
INPUT_NUM + 3] * buffer[3] +
107.      weight[out_idx * INPUT_NUM + 4] * buffer[4] + weight[out_idx * I
INPUT_NUM + 5] * buffer[5] +
108.      weight[out_idx * INPUT_NUM + 6] * buffer[6] + weight[out_idx * I
INPUT_NUM + 7] * buffer[7] +
109.      weight[out_idx * INPUT_NUM + 8] * buffer[8] + weight[out_idx * I
INPUT_NUM + 9] * buffer[9] +
110.      weight[out_idx * INPUT_NUM + 10] * buffer[10] + weight[out_idx *
INPUT_NUM + 11] * buffer[11] +
111.      weight[out_idx * INPUT_NUM + 12] * buffer[12] + weight[out_idx *
INPUT_NUM + 13] * buffer[13] +
112.      weight[out_idx * INPUT_NUM + 14] * buffer[14] + weight[out_idx *
INPUT_NUM + 15] * buffer[15] +
113.      weight[out_idx * INPUT_NUM + 16] * buffer[16] + weight[out_idx *
INPUT_NUM + 17] * buffer[17] +
114.      weight[out_idx * INPUT_NUM + 18] * buffer[18] + weight[out_idx *
INPUT_NUM + 19] * buffer[19] +
115.      weight[out_idx * INPUT_NUM + 20] * buffer[20] + weight[out_idx *
INPUT_NUM + 21] * buffer[21] +
116.      weight[out_idx * INPUT_NUM + 22] * buffer[22] + weight[out_idx *
INPUT_NUM + 23] * buffer[23] +
117.      weight[out_idx * INPUT_NUM + 24] * buffer[24] + weight[out_idx *
INPUT_NUM + 25] * buffer[25] +
118.      weight[out_idx * INPUT_NUM + 26] * buffer[26] + weight[out_idx *
INPUT_NUM + 27] * buffer[27] +
119.      weight[out_idx * INPUT_NUM + 28] * buffer[28] + weight[out_idx *
INPUT_NUM + 29] * buffer[29] +
120.      weight[out_idx * INPUT_NUM + 30] * buffer[30] + weight[out_idx *
INPUT_NUM + 31] * buffer[31] +
121.      weight[out_idx * INPUT_NUM + 32] * buffer[32] + weight[out_idx *
INPUT_NUM + 33] * buffer[33] +
122.      weight[out_idx * INPUT_NUM + 34] * buffer[34] + weight[out_idx *
INPUT_NUM + 35] * buffer[35] +
123.      weight[out_idx * INPUT_NUM + 36] * buffer[36] + weight[out_idx *
INPUT_NUM + 37] * buffer[37] +
124.      weight[out_idx * INPUT_NUM + 38] * buffer[38] + weight[out_idx *
INPUT_NUM + 39] * buffer[39] +
125.      weight[out_idx * INPUT_NUM + 40] * buffer[40] + weight[out_idx *
INPUT_NUM + 41] * buffer[41] +
126.      weight[out_idx * INPUT_NUM + 42] * buffer[42] + weight[out_idx *
INPUT_NUM + 43] * buffer[43] +
127.      weight[out_idx * INPUT_NUM + 44] * buffer[44] + weight[out_idx *

```

```

        INPUT_NUM + 45] * buffer[45] +
128.        weight[out_idx * INPUT_NUM + 46] * buffer[46] + weight[out_idx *
        INPUT_NUM + 47] * buffer[47] +
129.        bias[out_idx];
130.
131.    assign data_out = calc_out[18:7];
132.
133. endmodule
134.
135. //    integer step;
136. //    reg [19:0] result1[0:3];
137. //    reg [19:0] result2;
138.
139. //    always @(posedge clk) begin
140. //        if (~rst_n) begin
141. //            valid_out_fc <= 0;
142. //            buf_idx <= 0;
143. //            out_idx <= 0;
144. //            state <= 0;
145. //            step <= 0;
146. //            delay <= 0;
147. //            end
148.
149. //            if (valid_out_fc == 1) begin
150. //                valid_out_fc <= 0;
151. //                step <= 0;
152. //                end
153.
154. //            if (valid_in == 1) begin
155. //                // Wait until 48 input data filled in buffer
156. //                if (!state) begin
157. //                    buffer[buf_idx] <= data1;
158. //                    buffer[INPUT_WIDTH + buf_idx] <= data2;
159. //                    buffer[INPUT_WIDTH * 2 + buf_idx] <= data3;
160. //                    buf_idx <= buf_idx + 1'b1;
161. //                    if(buf_idx == INPUT_WIDTH - 1) begin
162. //                        buf_idx <= 0;
163. //                        state <= 1;
164. //                        step <= 0;
165. //                        delay <= 1;
166. //                        valid_out_fc <= 1;
167. //                        end
168. //                    end
169. //                else begin // valid state

```

```

170.
171. //          result1[step] <= weight[out_idx * INPUT_NUM + step *
    12] * buffer[0 + step * 12] + weight[out_idx * INPUT_NUM + 1 + step * 1
    2] * buffer[1 + step * 12]
172. //          + weight[out_idx * INPUT_NUM + 2 + step * 12] *
    buffer[2 + step * 12] + weight[out_idx * INPUT_NUM + 3 + step * 12] * bu
    ffer[3 + step * 12]
173. //          + weight[out_idx * INPUT_NUM + 4 + step * 12] *
    buffer[4 + step * 12] + weight[out_idx * INPUT_NUM + 5 + step * 12] * bu
    ffer[5 + step * 12]
174. //          + weight[out_idx * INPUT_NUM + 6 + step * 12] *
    buffer[6 + step * 12] + weight[out_idx * INPUT_NUM + 7 + step * 12] * bu
    ffer[7 + step * 12]
175. //          + weight[out_idx * INPUT_NUM + 8 + step * 12] *
    buffer[8 + step * 12] + weight[out_idx * INPUT_NUM + 9 + step * 12] * bu
    ffer[9 + step * 12]
176. //          + weight[out_idx * INPUT_NUM + 10 + step * 12] *
    buffer[10 + step * 12] + weight[out_idx * INPUT_NUM + 11 + step * 12] *
    buffer[11 + step * 12];
177.
178. //          if (step == 3) begin
179. //          result2 <= result1[0] + result1[1] + result1[2]
    + result1[3] + bias[out_idx];
180. //          step <= 0;
181. //          delay <= 0;
182. //          out_idx <= out_idx + 1'b1;
183. //          if (out_idx == OUTPUT_NUM - 1) begin
184. //          out_idx <= 0;
185. //          end
186. //          valid_out_fc <= 1;
187. //          end
188. //          else
189. //          step <= step + 1;
190.
191. //          end
192. //          end
193. //          end
194.
195. // assign data_out = result2[18:7];
196.
197. // endmodule

```

```

1.  `timescale 1ns / 1ps
2.
3.  // Description:  Final Comparator for decision
4.
5.  module comparator (
6.      input clk,
7.      input rst_n,
8.      input valid_in,
9.      input [11:0] data_in,
10.     output reg [3:0] decision,
11.     output reg valid_out
12. );
13.
14.     reg signed [11:0] buffer [0:9];
15.     reg signed [11:0] max;
16.     reg signed [11:0] cmp1_0, cmp1_1, cmp1_2, cmp1_3, cmp1_4,
17.                    cmp2_0, cmp2_1, cmp2_2, cmp3_0, cmp3_1;
18.     reg [3:0] buf_idx;
19.     reg [11:0] delay_cnt;
20.     reg state;
21.
22.     always @(posedge clk) begin
23.         if (~rst_n) begin
24.             valid_out <= 0;
25.             buf_idx <= 0;
26.             delay_cnt <= 0;
27.             state <= 0;
28.             decision <= 4'd0;
29.             cmp1_0 <= 12'd0;
30.             cmp1_1 <= 12'd0;
31.             cmp1_2 <= 12'd0;
32.             cmp1_3 <= 12'd0;
33.             cmp1_4 <= 12'd0;
34.             cmp2_0 <= 12'd0;
35.             cmp2_1 <= 12'd0;
36.             cmp2_2 <= 12'd0;
37.             cmp3_0 <= 12'd0;
38.             cmp3_1 <= 12'd0;
39.             max <= 12'd0;
40.             buffer[0] <= 12'd0;
41.             buffer[1] <= 12'd0;
42.             buffer[2] <= 12'd0;
43.             buffer[3] <= 12'd0;
44.             buffer[4] <= 12'd0;

```

```

45.         buffer[5] <= 12'd0;
46.         buffer[6] <= 12'd0;
47.         buffer[7] <= 12'd0;
48.         buffer[8] <= 12'd0;
49.         buffer[9] <= 12'd0;
50.     end
51.     else begin
52.         if (valid_in == 1) begin // read data input
53.             buffer[buf_idx] <= data_in;
54.             buf_idx <= buf_idx + 1'b1;
55.             if (buf_idx == 9)
56.                 state <= 1;
57.             end
58.         else begin
59.             if(state == 1) begin
60.                 delay_cnt <= delay_cnt + 1'b1;
61.                 if(delay_cnt == 12'd5)
62.                     valid_out <= 1;
63.                 else
64.                     valid_out <= 0;
65.
66.                 // Decision Process
67.                 cmp1_0 <= (buffer[0] >= buffer[1]) ? buffer[0] : b
uffer[1];
68.                 cmp1_1 <= (buffer[2] >= buffer[3]) ? buffer[2] : b
uffer[3];
69.                 cmp1_2 <= (buffer[4] >= buffer[5]) ? buffer[4] : b
uffer[5];
70.                 cmp1_3 <= (buffer[6] >= buffer[7]) ? buffer[6] : b
uffer[7];
71.                 cmp1_4 <= (buffer[8] >= buffer[9]) ? buffer[8] : b
uffer[9];
72.
73.                 cmp2_0 <= (cmp1_0 >= cmp1_1) ? cmp1_0 : cmp1_1;
74.                 cmp2_1 <= (cmp1_2 >= cmp1_3) ? cmp1_2 : cmp1_3;
75.                 cmp2_2 <= cmp1_4;
76.
77.                 cmp3_0 <= (cmp2_0 >= cmp2_1) ? cmp2_0 : cmp2_1;
78.                 cmp3_1 <= cmp2_2;
79.
80.                 //if(delay_cnt == 12'd3)
81.                 max <= (cmp3_0 >= cmp3_1) ? cmp3_0 : cmp3_1;
82.
83.                 //if(delay_cnt == 12'd4) begin

```



```

84.         if(max == buffer[0])
85.             decision <= 4'd0;
86.         else if(max == buffer[1])
87.             decision <= 4'd1;
88.         else if(max == buffer[2])
89.             decision <= 4'd2;
90.         else if(max == buffer[3])
91.             decision <= 4'd3;
92.         else if(max == buffer[4])
93.             decision <= 4'd4;
94.         else if(max == buffer[5])
95.             decision <= 4'd5;
96.         else if(max == buffer[6])
97.             decision <= 4'd6;
98.         else if(max == buffer[7])
99.             decision <= 4'd7;
100.        else if(max == buffer[8])
101.            decision <= 4'd8;
102.        else if(max == buffer[9])
103.            decision <= 4'd9;
104.        //end
105.        end
106.    end
107. end
108. end
109.
110. endmodule

```

7. 七段数码管显示模块

功能：使用数字逻辑小作业中实现的 display7 模块，将 CNN 的处理结果在七段数码管上进行显示。

建模 Verilog 代码：

```

1.  `timescale 1ns / 1ps
2.
3.  module display7(
4.      input [3:0] iData,
5.      output reg [6:0] oData
6.  );
7.
8.      always@(iData)
9.          case(iData)
10.             4'b0000: oData = 7'b1000000;
11.             4'b0001: oData = 7'b1111001;
12.             4'b0010: oData = 7'b0100100;

```

```

13.         4'b0011: oData = 7'b0110000;
14.         4'b0100: oData = 7'b0011001;
15.         4'b0101: oData = 7'b0010010;
16.         4'b0110: oData = 7'b0000010;
17.         4'b0111: oData = 7'b1111000;
18.         4'b1000: oData = 7'b0000000;
19.         4'b1001: oData = 7'b0010000;
20.         default: oData = 7'b1111111;
21.     endcase
22.
23. endmodule

```

8. MP3 播放模块

功能：将 CNN 的处理结果以音频提示的方式进行展示，当收到处理完毕的信号时，首先播放“识别结果为”的 MP3 音频，而后根据识别结果播放“X，X，X”的 MP3 音频，其中 X 为识别结果对应的数字音频；其中使用到的 MP3 音频，使用 ROM IP 核预先加载 coe 文件储存并访问。

建模 Verilog 代码：

```

1.  `timescale 1ns / 1ps
2.
3.  module mp3_top(
4.      input clk,
5.      input rst,
6.      input valid_in,
7.      input [3:0] decision,
8.      input DREQ,
9.      output xRSET,
10.     output xCS,
11.     output xDCS,
12.     output MOSI,
13.     output SCLK
14. );
15.
16.     reg is_play;
17.
18.     always @(posedge clk or posedge rst) begin
19.         if(rst)
20.             is_play <= 0;
21.         else if(valid_in)
22.             is_play <= 1;
23.         end
24.
25.     mp3_driver #(.MUSIC_SIZE(25112)) uut_mp3 (
26.         .mp3_clk(clk),

```

```

27.         .rst(rst | ~is_play),
28.         .decision(decision),
29.         .DREQ(DREQ),
30.         .xRSET(xRSET),
31.         .xCS(xCS),
32.         .xDCS(xDCS),
33.         .MOSI(MOSI),
34.         .SCLK(SCLK)
35.     );
36.
37. endmodule

```

```

1.  `timescale 1ns / 1ps
2.
3.  module mp3_driver #(parameter MUSIC_SIZE = 25112) (
4.      input mp3_clk,
5.      input rst,
6.      input [3:0] decision,
7.      input DREQ,
8.      output reg xRSET = 0,
9.      output reg xCS = 1,
10.     output reg xDCS = 1,
11.     output reg MOSI,
12.     output reg SCLK = 0
13. );
14.
15. reg [1:0] music_over;
16.
17. reg [14:0] mp3_addr;
18. wire [31:0] mp3_data_all[0:10];
19.
20. localparam RESET = 0,
21.             CMD_CONTROL = 1,
22.             CMD_SEND = 2,
23.             DATA_CONTROL = 3,
24.             DATA_SEND = 4;
25.
26. reg [2:0] state = RESET;
27.
28. reg [31:0] mp3_data, mp3_data0;
29.
30. localparam cmd_mode = 32'h02000804,
31.             cmd_vol = 32'h020B1010,
32.             cmd_bass = 32'h02020055,

```

```

33.             cmd_clock = 32'h02039800;
34.
35.     reg [127:0] cmd = {cmd_mode, cmd_vol, cmd_bass, cmd_clock};
36.     reg [2:0] cmd_id;
37.     reg [5:0] cnt;
38.
39.     //reg music_over=0;
40.
41.     localparam CMD_NUM = 4;
42.
43.     always @(posedge mp3_clk or posedge rst) begin
44.         if (rst == 1) begin
45.             state<=RESET;
46.             music_over <= 2'b00;
47.         end
48.         else if (music_over == 2'b00 || music_over == 2'b01) begin
49.             if (state == RESET) begin
50.                 cmd <= {cmd_mode, cmd_vol, cmd_bass, cmd_clock};
51.                 mp3_addr <= 0;
52.                 cmd_id <= 0;
53.                 cnt <= 0;
54.                 xCS <= 1;
55.                 xDCS <= 1;
56.                 xRSET <= 0;
57.                 SCLK <= 0;
58.
59.                 if (rst == 0)
60.                     state <= CMD_CONTROL;
61.                 else
62.                     state <= RESET;
63.             end
64.
65.             else if (state == CMD_CONTROL) begin
66.                 xRSET <= 1;
67.                 if (cmd_id < CMD_NUM && DREQ) begin
68.                     cmd_id <= cmd_id + 1;
69.                     xCS <= 0;
70.                     MOSI <= cmd[127];
71.                     cmd <= {cmd[126:0], cmd[127]};
72.                     cnt <= 1;
73.                     state <= CMD_SEND;
74.                 end
75.             else begin
76.                 cmd_id <= 0;

```

```

77.         state <= DATA_CONTROL;
78.     end
79. end
80.
81.     else if (state == CMD_SEND) begin
82.         if (DREQ) begin
83.             if(SCLK) begin
84.                 if(cnt < 32) begin
85.                     cnt <= cnt + 1;
86.                     MOSI <= cmd[127];
87.                     cmd <= {cmd[126:0], cmd[127]};
88.                 end
89.             else begin
90.                 xCS <= 1;
91.                 cnt <= 0;
92.                 state <= CMD_CONTROL;
93.             end
94.         end
95.         SCLK <= ~SCLK;
96.     end
97. end
98.
99.     else if(state == DATA_CONTROL) begin
100.        if(mp3_addr >= MUSIC_SIZE) begin
101.            state <= RESET;
102.            music_over <= music_over + 1'b1;
103.            //DCS<=1;
104.        end
105.    else if (DREQ) begin
106.        //music_over<=0;
107.        xDCS <= 0;
108.        SCLK <= 0;
109.
110.        if (music_over == 2'b00)
111.            mp3_data0 <= mp3_data_all[10];
112.        else if (music_over == 2'b01)
113.            mp3_data0 <= mp3_data_all[decision];
114.
115.        MOSI <= mp3_data0[31];
116.        mp3_data <= {mp3_data0[30:0], mp3_data0[31]};
117.        cnt <= 1;
118.        state <= DATA_SEND;
119.    end
120. end

```

```

121.
122.         else if(state == DATA_SEND) begin
123.             if (DREQ) begin
124.                 if (SCLK) begin
125.                     if (cnt < 32) begin
126.                         MOSI <= mp3_data[31];
127.                         cnt <= cnt + 1;
128.                         mp3_data <= {mp3_data[30:0], mp3_data[31]};
129.                     end
130.                 else begin
131.                     xDCS <= 1;
132.                     cnt <= 0;
133.                     mp3_addr <= mp3_addr + 1;
134.                     state <= DATA_CONTROL;
135.                 end
136.             end
137.             SCLK <= ~SCLK;
138.         end
139.     end
140.
141. else
142.     state <= RESET;
143. end
144. end
145.
146. blk_mem_gen_0 mic_tips (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr), .douta(mp3_data_all[10]));
147. blk_mem_gen_1 num_0 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[0]));
148. blk_mem_gen_2 num_1 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[1]));
149. blk_mem_gen_3 num_2 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[2]));
150. blk_mem_gen_4 num_3 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[3]));
151. blk_mem_gen_5 num_4 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[4]));
152. blk_mem_gen_6 num_5 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[5]));
153. blk_mem_gen_7 num_6 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[6]));
154. blk_mem_gen_8 num_7 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[7]));
155. blk_mem_gen_9 num_8 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[8]));

```

```

        2:0]), .douta(mp3_data_all[8]));
156.     blk_mem_gen_10 num_9 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[
        12:0]), .douta(mp3_data_all[9]));
157.
158. endmodule

```

五、测试模块建模

本实验主要对 CNN 模块进行了测试，其 testbench 模块如下

```

1.  `timescale 1ns / 1ps
2.
3.  module cnn_tb();
4.
5.      reg clk, rst_n;
6.      reg [7:0] pixels [0:783];
7.      reg [9:0] img_idx;
8.      reg [7:0] data_in;
9.
10.     wire signed [11:0] conv_out_1, conv_out_2, conv_out_3;
11.     wire signed [11:0] conv2_out_1, conv2_out_2, conv2_out_3;
12.     wire signed [11:0] max_value_1, max_value_2, max_value_3;
13.     wire signed [11:0] max2_value_1, max2_value_2, max2_value_3;
14.     wire signed [11:0] fc_out_data;
15.     wire [3:0] decision;
16.
17.     wire valid_out_1, valid_out_2, valid_out_3, valid_out_4, valid_out_5, valid_out
        _6;
18.
19.     // Module Instantiation
20.     conv1_layer conv1_layer(
21.         .clk(clk),
22.         .rst_n(rst_n),
23.         .data_in(data_in),
24.         .conv_out_1(conv_out_1),
25.         .conv_out_2(conv_out_2),
26.         .conv_out_3(conv_out_3),
27.         .valid_out_conv(valid_out_1)
28.     );
29.
30.     maxpool_relu #(.CONV_BIT(12), .HALF_WIDTH(12), .HALF_HEIGHT(12), .HALF_WIDTH_BI
        T(4))
31.     maxpool_relu_1(
32.         .clk(clk),

```

```

33.     .rst_n(rst_n),
34.     .valid_in(valid_out_1),
35.     .conv_out_1(conv_out_1),
36.     .conv_out_2(conv_out_2),
37.     .conv_out_3(conv_out_3),
38.     .max_value_1(max_value_1),
39.     .max_value_2(max_value_2),
40.     .max_value_3(max_value_3),
41.     .valid_out_relu(valid_out_2)
42. );
43.
44. conv2_layer conv2_layer(
45.     .clk(clk),
46.     .rst_n(rst_n),
47.     .valid_in(valid_out_2),
48.     .max_value_1(max_value_1),
49.     .max_value_2(max_value_2),
50.     .max_value_3(max_value_3),
51.     .conv2_out_1(conv2_out_1),
52.     .conv2_out_2(conv2_out_2),
53.     .conv2_out_3(conv2_out_3),
54.     .valid_out_conv2(valid_out_3)
55. );
56.
57. maxpool_relu #(.CONV_BIT(12), .HALF_WIDTH(4), .HALF_HEIGHT(4), .HALF_WIDTH_BIT(
    3))
58. maxpool_relu_2(
59.     .clk(clk),
60.     .rst_n(rst_n),
61.     .valid_in(valid_out_3),
62.     .conv_out_1(conv2_out_1),
63.     .conv_out_2(conv2_out_2),
64.     .conv_out_3(conv2_out_3),
65.     .max_value_1(max2_value_1),
66.     .max_value_2(max2_value_2),
67.     .max_value_3(max2_value_3),
68.     .valid_out_relu(valid_out_4)
69. );
70.
71. fully_connected #(.INPUT_NUM(48), .OUTPUT_NUM(10), .DATA_BITS(8))
72. fully_connected(
73.     .clk(clk),
74.     .rst_n(rst_n),
75.     .valid_in(valid_out_4),

```



```

76.     .data_in_1(max2_value_1),
77.     .data_in_2(max2_value_2),
78.     .data_in_3(max2_value_3),
79.     .data_out(fc_out_data),
80.     .valid_out_fc(valid_out_5)
81. );
82.
83. comparator comparator(
84.     .clk(clk),
85.     .rst_n(rst_n),
86.     .valid_in(valid_out_5),
87.     .data_in(fc_out_data),
88.     .decision(decision),
89.     .valid_out(valid_out_6)
90. );
91.
92. // Clock generation
93. always #5 clk = ~clk;
94.
95. // Read image text file
96. initial begin
97.     $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/9_0.txt", pi
        xels);
98.     clk <= 1'b0;
99.     rst_n <= 1'b1;
100.    #3
101.    rst_n <= 1'b0;
102.    #3
103.    rst_n <= 1'b1;
104. end
105.
106. always @(posedge clk) begin
107.     if(~rst_n) begin
108.         img_idx <= 0;
109.     end else begin
110.         data_in <= pixels[img_idx];
111.         if(img_idx < 10'd784) begin
112.             img_idx <= img_idx + 1'b1;
113.         end
114.     end
115. end
116.
117. endmodule

```

六、实验结果

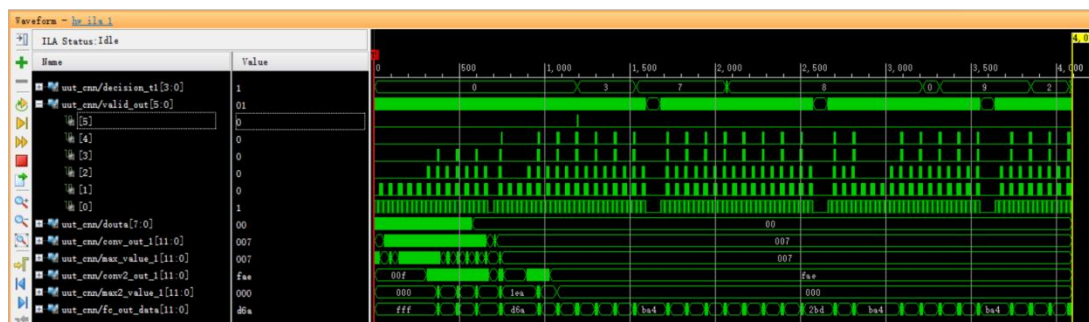
1. 待输入手写数字灰度图像素数据:

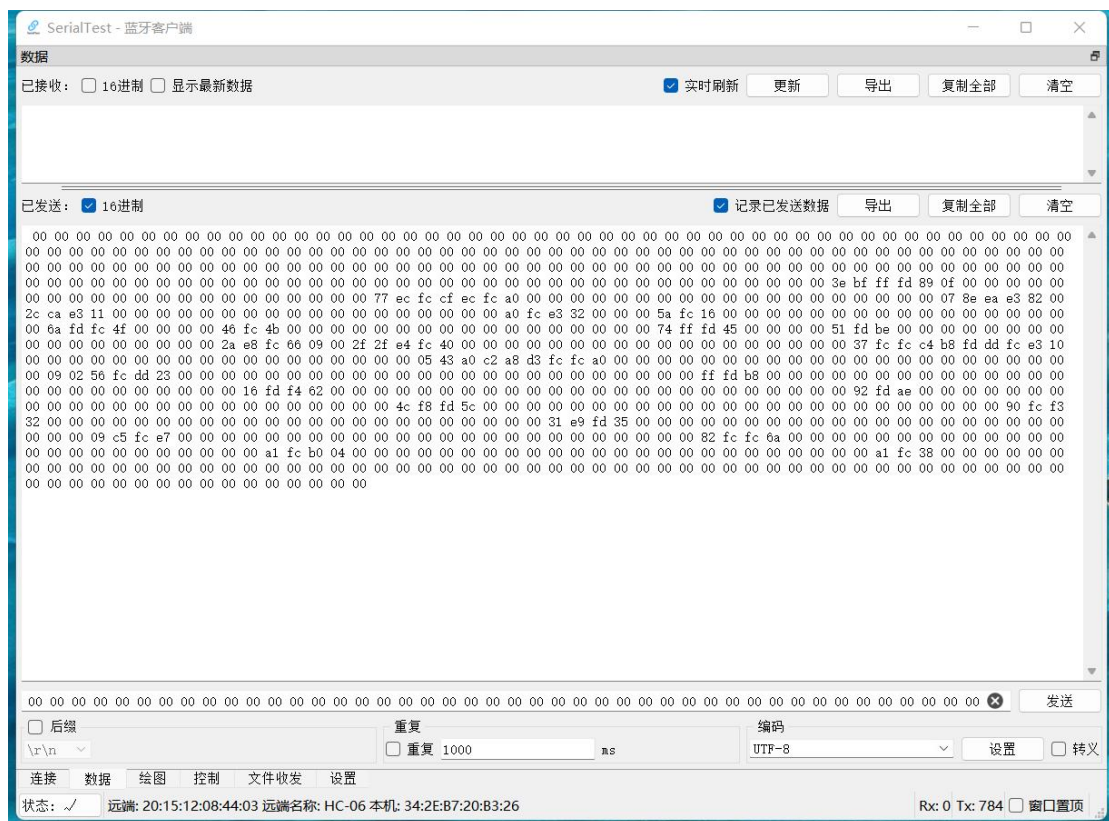
[illegible]

2. CNN 模块仿真波形图

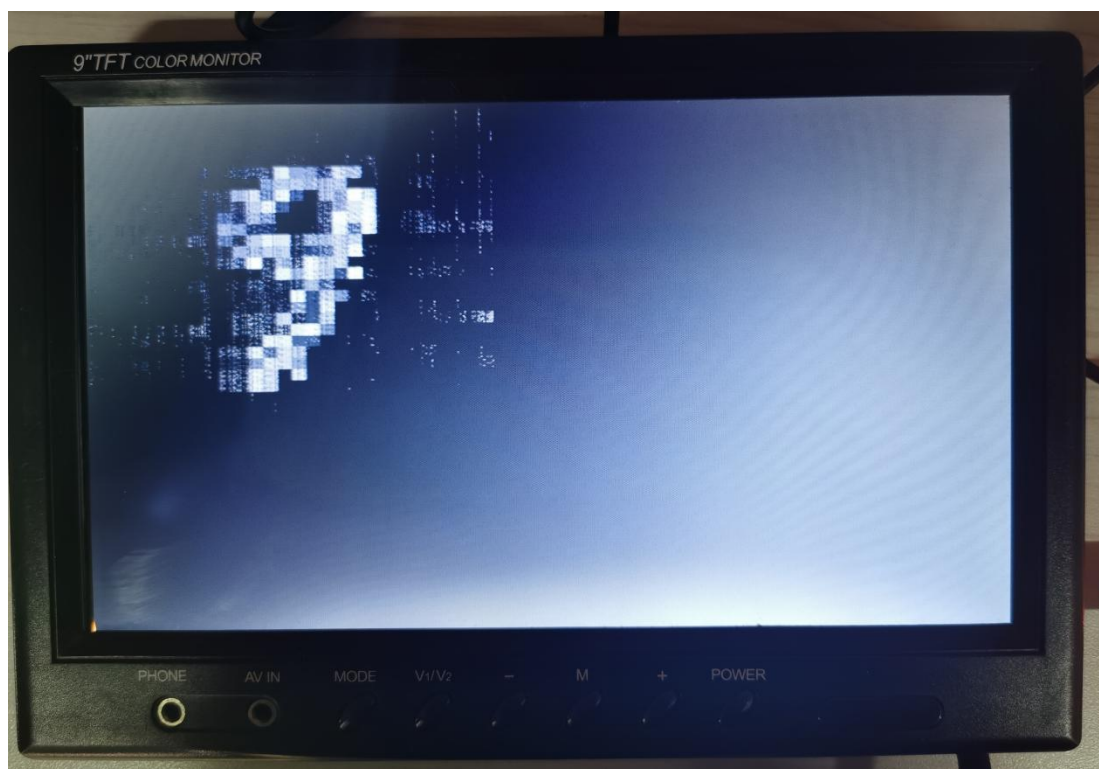


3. CNN 模块 ILA 下板调试结果

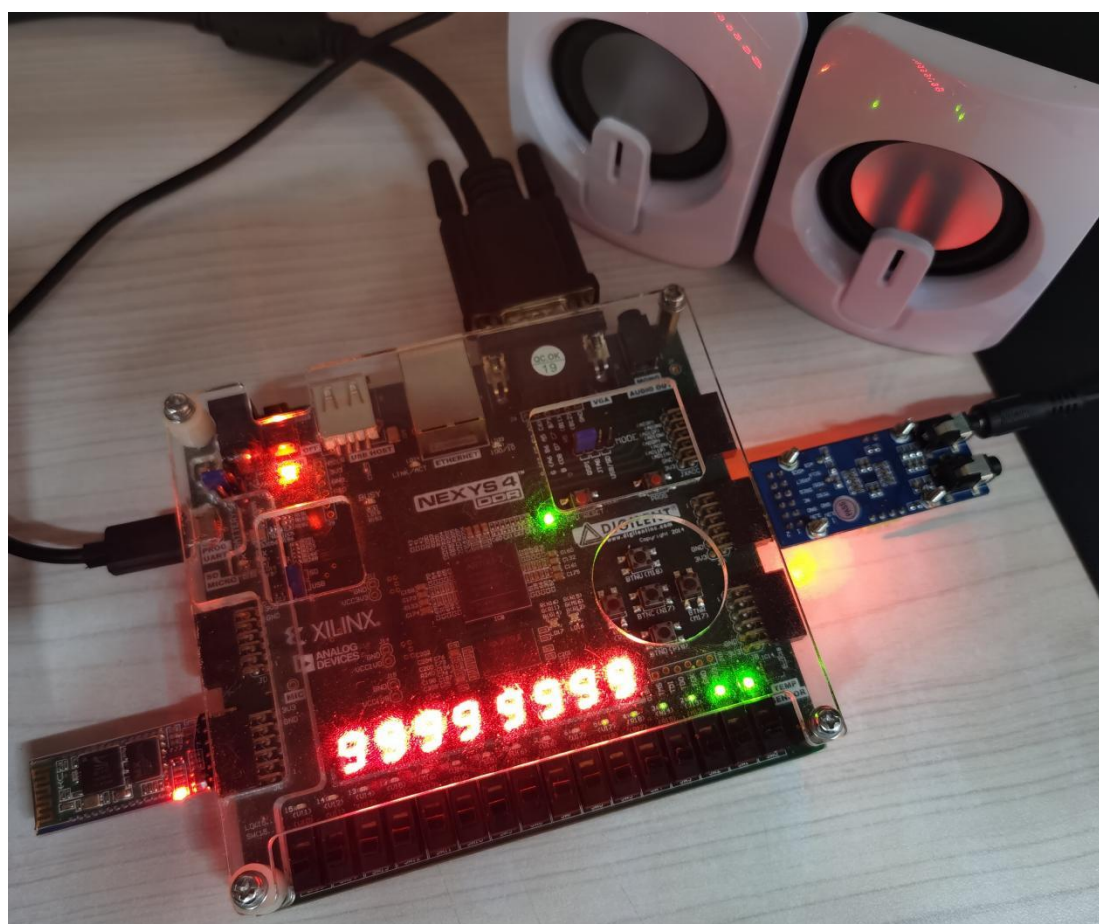




(3) 蓝牙接收完毕，VGA 显示



(4) 识别结果显示与语音播报



(5) 资源消耗

Design Runs													
Name	Constraints	Status	VHS	TNS	VHS	TNS	TPFS	Failed Routes	LUT	FF	BRAM	URAM	DSP
synth_1 (active)	constrs_1	Synthesis Out-of-date							17093	11641	0	0	1
impl_1	constrs_1	Implementation Out-of-date	35.271	0.000	0.592	0.000	0.000	0	20022	11785	80	0	229
Out-of-Context Module Runs													
clk_vio_0	clk_vio_0	synth_design Complete!							0	0	0	0	0
blk_mem_gen_0	blk_mem_gen_0	Synthesis Out-of-date							70	10	23	0	0
fully_connected	fully_connected	synth_design Complete!							1318	25	0	0	48
ila_0	ila_0	synth_design Complete!							2974	3699	6	0	0
ila_1	ila_1	synth_design Complete!							889	1445	9	0	0
conv2_calc_1	conv2_calc_1	synth_design Complete!							493	1	0	0	60
conv2_calc_2	conv2_calc_2	synth_design Complete!							340	1	0	0	60
conv2_calc_3	conv2_calc_3	synth_design Complete!							258	1	0	0	60
blk_mem_gen_1	blk_mem_gen_1	synth_design Complete!							34	4	6	0	0
blk_mem_gen_2	blk_mem_gen_2	synth_design Complete!							37	8	5	0	0
blk_mem_gen_3	blk_mem_gen_3	synth_design Complete!							34	6	5	0	0
blk_mem_gen_4	blk_mem_gen_4	synth_design Complete!							36	8	6	0	0
blk_mem_gen_5	blk_mem_gen_5	synth_design Complete!							34	4	6	0	0
blk_mem_gen_6	blk_mem_gen_6	synth_design Complete!							34	4	6	0	0
blk_mem_gen_7	blk_mem_gen_7	Synthesis Out-of-date							37	8	5	0	0
blk_mem_gen_8	blk_mem_gen_8	synth_design Complete!							37	8	5	0	0
blk_mem_gen_9	blk_mem_gen_9	synth_design Complete!							37	8	5	0	0
blk_mem_gen_10	blk_mem_gen_10	synth_design Complete!							37	8	5	0	0

七、结论

本次实验中我研究了 VGA 显示屏、MP3 以及蓝牙 UART 串口的时序及控制方法，并研究实现了流水线设计的简单卷积神经网络，即实现了一个基于 FPGA 的手写数字识别系统。

受开发板资源限制，CNN 的部署实现需要小心翼翼，在这个过程中我学会了如何设计精巧的结构控制 BRAM 使用数量，以及如何使计算的串、并行交替进行，设计高效的流水线式结构。

本程序虽在模拟仿真时效果较好，但下板后运行结果不稳定，经过很多天的努力也未彻底解决，经过查询相关资料及与老师讨论，得出可能原因如下：第一，程序的时序控制仍有问题，在实际下板时可能存在潜在的竞争冲突等问题；第二，开发板资源使用过多，开发板一旦存在小部分电路老化，Lut、FF 以及 DSP 等单元受损功能异常，可能就会导致计算结果异常。

此外，开发类似神经网络等耗资源较多、时序较复杂的模型时，行业上一般将其部署在 ZNYQ 系列的开发板上，用以硬件加速计算，在本次实验中，我也深深体会到这样做的原因，单纯的 FPGA 部署神经网络确有其困难之处，且需根据神经网络结构的不同以及开发板资源的大小特别设计时序，并没有统一的标准流程，若有机会本人将在 ZNYQ 系列的开发板进行试验，深刻体会二者之间的异同。

八、心得体会及建议

1. 本课程收获

通过本学期数字逻辑课程的学习，我可以说收获满满。首先最大的收获就是步入了硬件的大门，通过数字逻辑的学习，我们从底层门电路开始研究，逐步建立起一个综合实验这样的比较庞大的电路系统，在硬件的学习上可以说有了一定的基础。

其次这门课程让我对计算机底层电路有了更深的理解。虽然接触程序设计已久，但一直都是在计算机的软件层面进行设计，对于底层电路的实现方式及原理并不知道。通过课程学习，我了解了一些底层电路的工作方式，对于计算机的工作原理有了更深入的认识。

最后，这次综合实验也让我收获颇丰。在这学期的理论学习与平时作业中，我们虽然学到了很多知识，但是并没有把他们联系起来，也没有用到一些较复杂的功能。但在这次综合实验中，我们不得不将本学期所学全部联系起来，并额外学习很多知识，才能够真正完成这一个实验。因此，正是这次综合实验让我真正吸收了本学期所学，并使我的自学能力与数字系统设计能力有了不少提升。

2. 对本课程建议

(1) 建议采购一批新的外设。在这次综合实验过程中，我意识到目前学校存在的外设存在型号较老、数量不足、种类不足等问题。因此我建议：淘汰一部分型号过老的部件；对于 VGA 屏幕这样几乎是必需品却数量不足的部件，建议再采购一批补齐数量；对于摄像头、陀螺仪、键盘这样比较困难而缺少相关资料的部件，建议额外补充一些可靠的相关资料。

(2) 课堂中对于外设开发部分的讲解过少。在本次实验中，我发现有些部件过于困难，有些部件资料太少，并且网络上的资料鱼龙混杂，很容易落入陷阱以致开发停滞不前，虽然实验目的中有让我们自行学习的要求，但是我认为不应该在这种部件上耽误过多时间。

(3) 课堂中增加对 Vivado 一些功能的讲解，如 IP 核、DEBUG 工具等，在本次实验过程中我发现这些工具是很有用的，可以大大减轻开发难度。

3. 结合数字芯片设计国内外现状深入谈自己的认识与体会

在现阶段，我国芯片设计和国外发达国家相比仍有不小差距，虽然近年来国家已经认识到了这一差距并投入了大量人力物力，但是距离“中国芯”自给自足仍有不小的距离。我国作为制造业大国，在各种应用类电子产品的制造上均位于世界前列，但这些电子产品的核心芯片我们却一直受制于人。近年来华为等企业相继被美国制裁，使我们更加深刻的认识到了这一切。

其实我国对于芯片的研究也已经有了一定的成果，业界知名的“星光中国芯工程”启动十多年来，通过自主创新结束了中国无“芯”的历史、成功开发安防监控 SVAC 芯片并形成国标、国内首推人工智能神经网络处理器芯片等，为我国芯片产业发展做出了重要贡献。但是我们也要认识到，虽然在科研层面上我们已经有了不错的芯片，但是对于芯片的小型化与量产上，我们仍有一段不短的路要走。

此外，在 EDA 设计软件方面，当前主流仍是外国软件，我们使用的 Vivado 也是外国公司开发的，其复杂程度令人震惊，由于其全英文界面并且缺少中文帮助资源，对于初学者而言，使用这种软件也是一大困难。

最后，对于我们今后学习与发展的方向，也应当有新的认识。我国当下硬件人才短缺，国家的许多政策都和大力培养芯片开发工程师有关，可以说，硬件人才的培养是当下大势所趋。我们作为计算机专业的学生，本身就需要“软硬件两手抓”，面对“中国芯”缺乏的现状，我们更不能放松对硬件课的学习，而应当更加重视对硬件的设计与发展。