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；

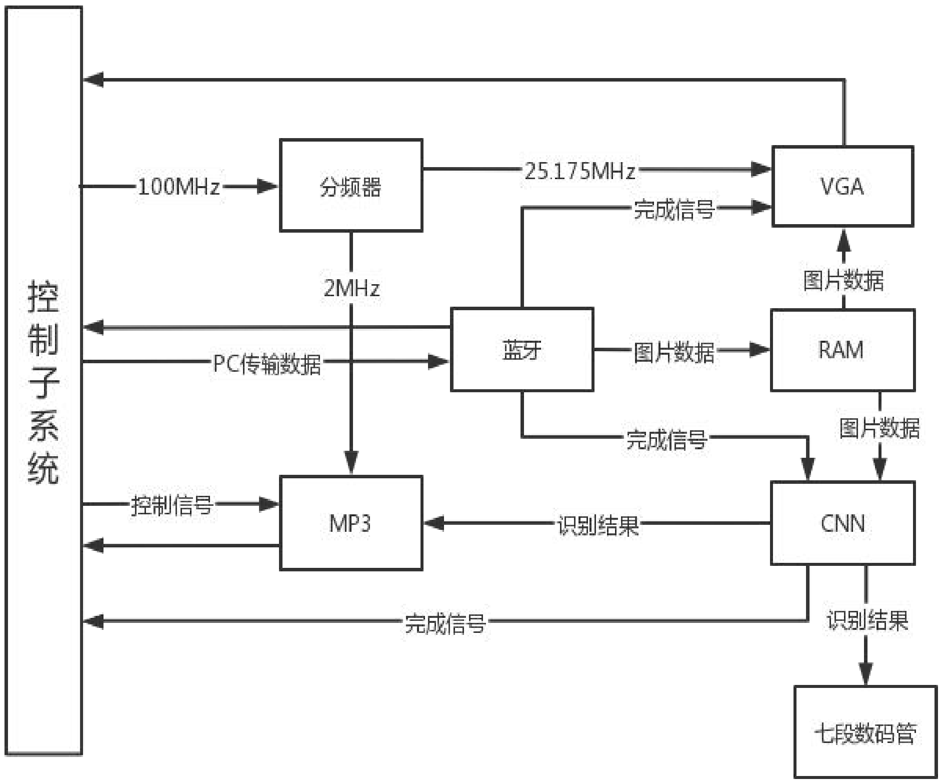
   (4) 将灰度图像素数据输入进简单卷积神经网络，网络输出的结果即为手写数字识别结果；

   (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。

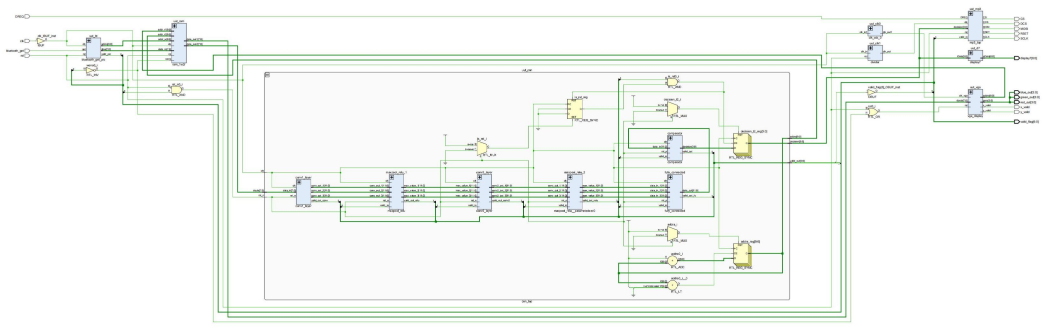
5. VGA 模块：
   读取 RAM 模块存储的灰度图像素数据，并将其显示在 VGA 显示器上。其内有 VGA 控制子模块。

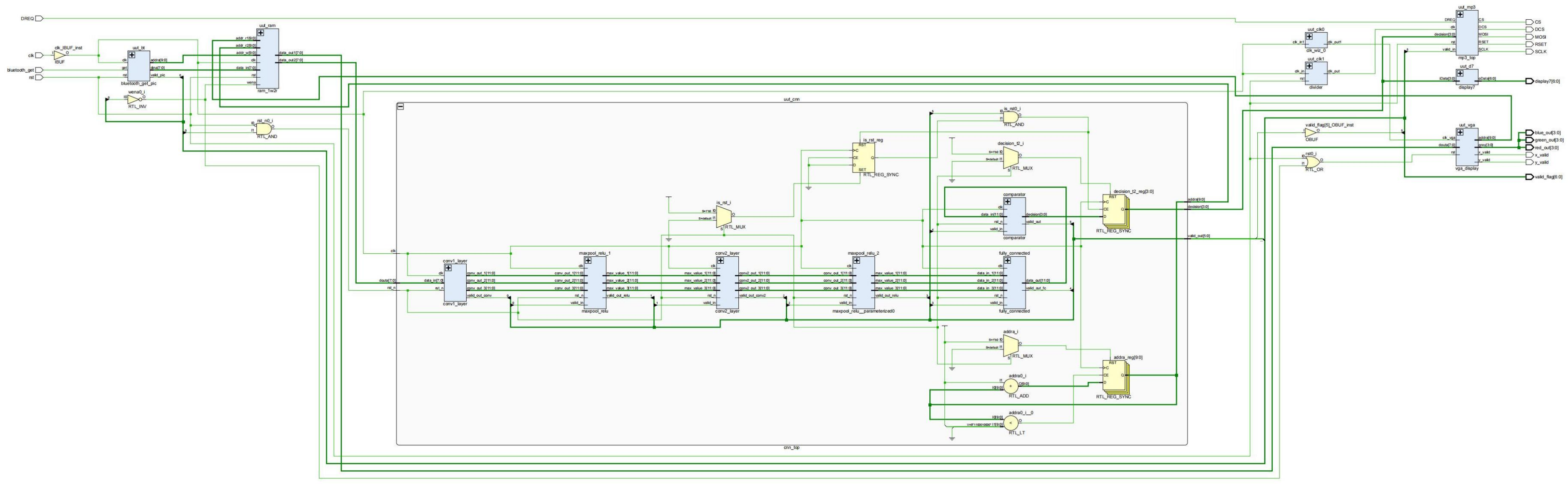
6. CNN 模块：
   读取 RAM 模块存储的灰度图像素数据，并输入进网络进行处理，最后输出识别结果。其内有多层卷积层子模块。

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。

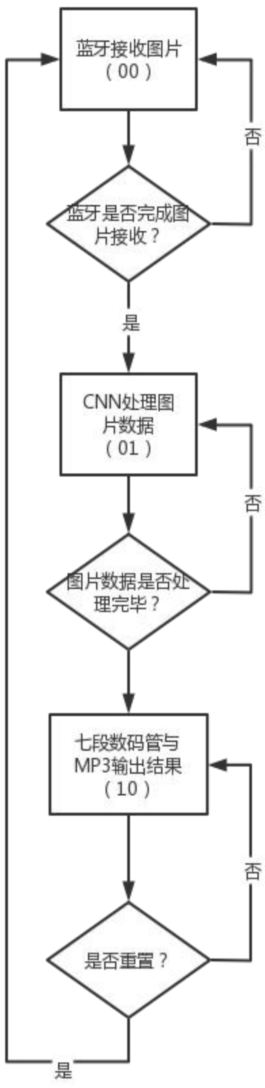
RTL 设计原理图如下：




DREQ
clk
bluetooth_get
rst
uut_bt
bluetooth_get_pic
uut_ram
ram_1w2r
wena0_i
RTL_INV
rst_n0_i
RTL_AND
uut_cnn
cnn_top
conv1_layer
maxpool_relu_1
conv2_layer
maxpool_relu_2
maxpool_relu__parameterized0
fully_connected
comparator
is_rst_i
RTL_MUX
is_rst_reg
RTL_REG_SYNC
is_rst0_i
decision_t2_i
decision_t2_reg[3:0]
addra_i
addra_reg[9:0]
addra0_i
RTL_ADD
addra0_i__0
RTL_LT
uut_clk0
clk_wiz_0
uut_clk1
divider
uut_mp3
mp3_top
uut_d7
display7
valid_flag[5]_OBUF_inst
OBUF
rst0_i
RTL_OR
uut_vga
vga_display
CS
DCS
MOSI
RSET
SCLK
display7[6:0]
blue_out[3:0]
green_out[3:0]
red_out[3:0]
x_valid
y_valid
valid_flag[6:0]

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：

```
`timescale 1ns / 1ps

```

```
module mnist_top(
    input clk,
    input rst,
    output [6:0] valid_flag,
    output [6:0] display7,
    // BLUETOOTH
    input bluetooth_get,
    // VGA
    output [3:0] red, green, blue,
    output Hs_valid,
    output Vs_valid,
    // MP3
    input DREQ,
    output xRSET,
    output xCS,
    output xDCS,
    output MOSI,
    output SCLK
    );

    wire clk_vga ;//vga 25.175mhz
    wire clk_mp3_std;//mp3 12.288mhz
    wire clk_mp3;//mp3 2mhz
    clk_wiz_0 uut_clk0(.clk_in1(clk), .clk_out1(clk_vga), .clk_out2(clk_mp3_std));
    divider #(.mod(200)) uut_clk1(.clk_in(clk),.rst(rst),.clk_out(clk_mp3));

    wire [9:0] addr_w, addr_r1, addr_r2;
    wire [7:0] data_in, data_out1, data_out2;

    ram_1w2r uut_ram (
        .clk(clk),
        .rst(rst),
        .wena(~valid_flag[0]),
        .addr_w(addr_w),
        .data_in(data_in),
        .addr_r1(addr_r1),
        .data_out1(data_out1),
        .addr_r2(addr_r2),
        .data_out2(data_out2)
        );

    bluetooth_get_pic uut_bt (
        .clk(clk),
        .rst(rst),
        .get(bluetooth_get),
        .dina(data_in),
        .addra(addr_w),
        .valid_pic(valid_flag[0])
        );

    vga_display uut_vga (
        .clk_vga(clk_vga),//25.175MHz
        .rst(rst | ~valid_flag[0]),
        .douta(data_out1),
        .addra(addr_r1),
        .Hs_valid(Hs_valid),
        .Vs_valid(Vs_valid),
        .grey(red)
        );

    assign green = red;
    assign blue = red;

    wire [3:0] decision;
    cnn_top uut_cnn (
        .clk(clk),
        .rst_n(~rst & valid_flag[0]),
        .douta(data_out2),
        .addra(addr_r2),
        .valid_out(valid_flag[6:1]),
        .decision(decision)
        );

    display7 uut_d7 (.iData(decision), .oData(display7));

    mp3_top uut_mp3 (
        .clk(clk_mp3),
        .rst(rst),
        .valid_in(valid_flag[6]),
        .decision(decision),
        .DREQ(DREQ),
        .xRSET(xRSET),
        .xCS(xCS),
        .xDCS(xDCS),
        .MOSI(MOSI),
        .SCLK(SCLK)
        );

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：

```verilog
`timescale 1ns / 1ps

module divider #(parameter mod = 32'd20) (
    input clk_in,
    input rst,
    output reg clk_out
    );

    localparam T = mod / 2;

    reg [31:0] cnt = 0;

    initial begin
        clk_out <= 0;
        end

    always @(posedge clk_in) begin
        if(rst == 1) begin
            cnt <= 0;
            clk_out <= 0;
            end
        else begin
            if(cnt == T) begin
                cnt <= 1;
                clk_out <= ~clk_out;
                end
            else
                cnt <= cnt + 1;
            end
        end

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 28×28×8 的手写数字灰度图片像素数据。

建模 Verilog 代码：

```verilog
`timescale 1ns / 1ps

module ram_1w2r(
    input clk,
    input rst,
    input wena,
    input [9:0] addr_w,
    input [7:0] data_in,
    input [9:0] addr_r1,
    output reg [7:0] data_out1,
    input [9:0] addr_r2,
    output reg [7:0] data_out2
    );

    reg [7:0] pic[0:783];

    always @(posedge clk or negedge rst) begin
        if (rst) begin
            data_out1 <= 0;
            data_out2 <= 0;
            end
        else begin
            if (wena)
                pic[addr_w] <= data_in;
            else begin
                data_out1 <= pic[addr_r1];
                data_out2 <= pic[addr_r2];
                end
            end
        end

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：

```verilog
`timescale 1ns / 1ps

module bluetooth_get_pic(
    input clk,
    input rst,
    input get,
    output reg [7:0] dina,
    output reg [9:0] addra,
    output reg valid_pic
);

    parameter bps = 10417;// 1/9600
    reg [14:0] count_1;
    reg [3:0] count_2;
    reg buffer_0, buffer_1, buffer_2;
    wire buffer_en;
    reg add_en;

    always @(posedge clk) begin
        if (rst) begin
            buffer_0 <= 1;
            buffer_1 <= 1;
            buffer_2 <= 1;
            end
        else begin
            buffer_0 <= get;
            buffer_1 <= buffer_0;
            buffer_2 <= buffer_1;
            end
        end

    assign buffer_en = buffer_2 & ~buffer_1;

    always @(posedge clk) begin
        if (rst)
            count_1 <= 0;
        else if (add_en) begin
            if(count_1 == bps-1)
                count_1 <= 0;
            else
                count_1 <= count_1+1;
            end
        end

    always @(posedge clk) begin
        if (rst)
            count_2 <= 0;
        else if (add_en && count_1 == bps-1) begin
            if(count_2 == 8)
                count_2 <= 0;
            else
                count_2 <= count_2 + 1;
            end
        end

    always @(posedge clk) begin
        if (rst)
            add_en <= 0;
        else if (buffer_en)
            add_en <= 1;
        else if (add_en && (count_2 == 8) && (count_1 == bps-1))
            add_en <= 0;
        end

    always @(posedge clk) begin
        if (rst) begin
            dina <= 0;
            addra <= 0;
            valid_pic <= 0;
            end
        else if(add_en && (count_1 == bps / 2 - 1) && (count_2 != 0)) begin
            dina[count_2 - 1] <= get;
            if (count_2 == 8) begin
                addra = addra + 1;
                if (addra == 10'd784) begin
                    valid_pic <= 1;
                    end
                end
            end
        end

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：

```
`timescale 1ns / 1ps

```

```verilog
module vga_display(
    input clk_vga,//25.175MHz
    input rst,
    input [7:0] douta,
    output reg [9:0] addra,
    output Hs_valid,
    output Vs_valid,
    output reg [3:0] grey
);
    parameter Hs_before=11'd144;
    parameter Vs_before=11'd35;
    parameter Hs_size_pic=11'd280;
    parameter Vs_size_pic=11'd280;

    wire [11:0] Hs_poi;
    wire [11:0] Vs_poi;
    wire is_display;

    vga_control control(clk_vga,rst,Hs_poi,Vs_poi,is_display,Hs_valid,Vs_valid);

    always @(*) begin
        grey <= 0;
        if (is_display && Hs_poi-Hs_before <= Hs_size_pic && Vs_poi-Vs_before <= Vs_size_pic) begin
            addra <= (Vs_poi - Vs_before)/10*28 + (Hs_poi - Hs_before)/10;
            grey <= douta * 10;
            end
        end

endmodule
```

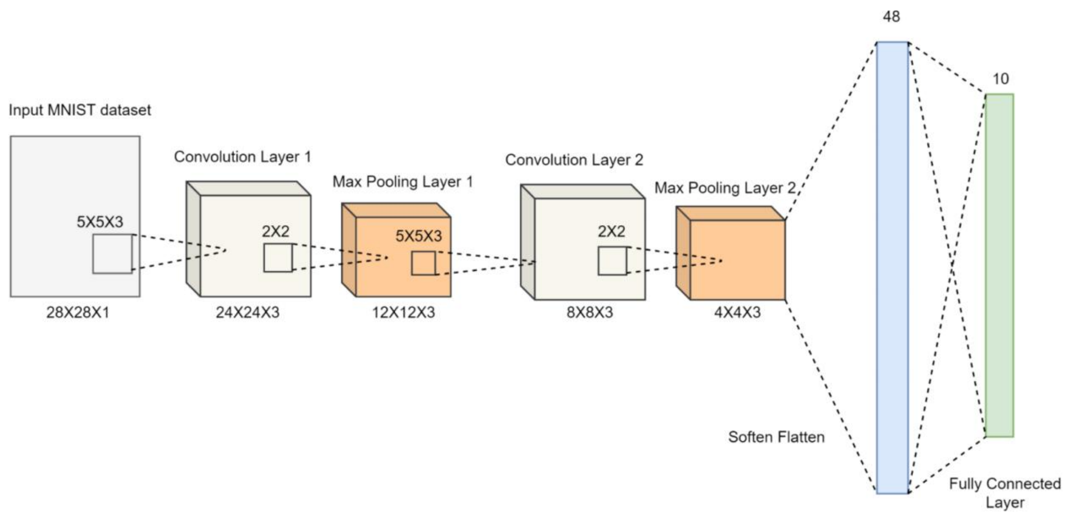
```verilog
`timescale 1ns / 1ps

module vga_control(
    input vga_clk,
    input rst,
    output reg[11:0] Hs_poi,
    output reg[11:0] Vs_poi,
    output is_display,
    output Hs_valid,
    output Vs_valid
);
```

```

    parameter x_sync = 11'd96;
    parameter x_before = 11'd144;
    parameter x_beside_after = 11'd784;
    parameter x_all = 11'd800;

    parameter y_sync = 11'd2;
    parameter y_before = 11'd35;
    parameter y_beside_after = 11'd515;
    parameter y_all = 11'd525;

    assign is_display = ((Hs_poi >= x_before) && (Hs_poi < x_beside_after)
                        && (Vs_poi >= y_before) && (Vs_poi < y_beside_after)) ? 1 : 0;

    assign Hs_valid = (Hs_poi < x_sync) ? 0 : 1;
    assign Vs_valid = (Vs_poi < y_sync) ? 0 : 1;

    always @(posedge vga_clk) begin
        if (rst) begin
            Hs_poi <= 0;
            Vs_poi <= 0;
        end
        else begin
            if (Hs_poi == x_all - 1) begin
                Hs_poi <= 0;
                if (Vs_poi == y_all - 1)
                    Vs_poi <= 0;
                else
                    Vs_poi <= Vs_poi + 1;
                end
            else
                Hs_poi <= Hs_poi + 1;
            end
        end

endmodule
```

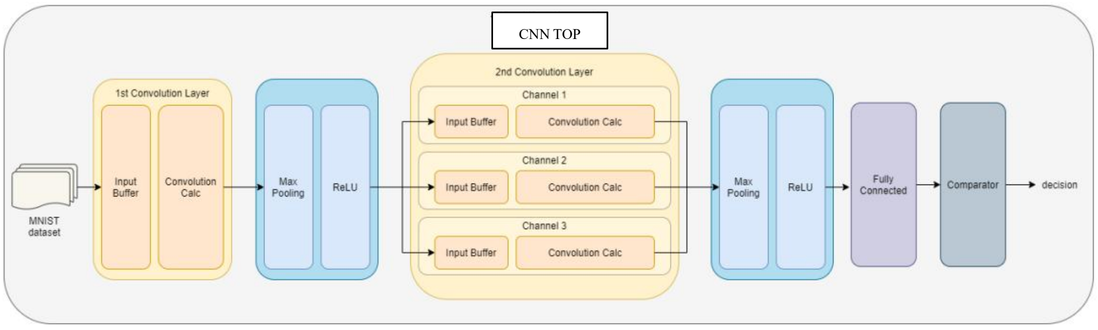
6. CNN 图片处理模块

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。

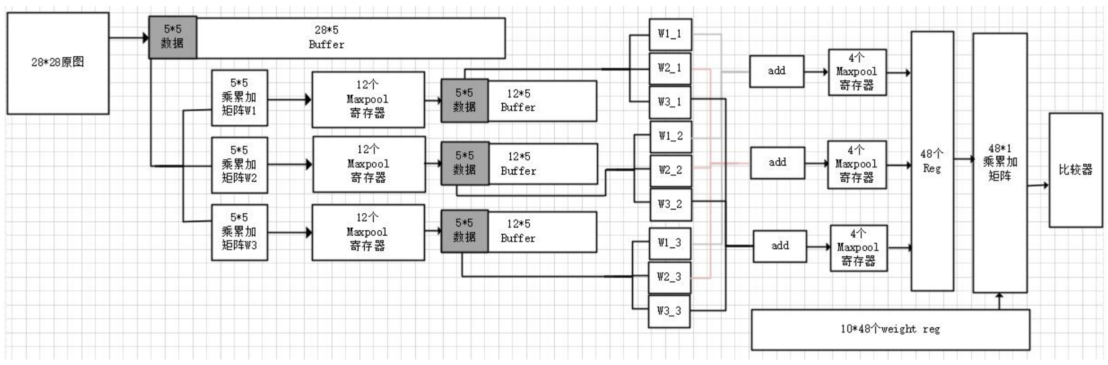
使用的神经网络的结构如下



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：

```verilog
`timescale 1ns / 1ps

module cnn_top(
    input clk,
    input rst_n,
    input [7:0] douta,
    output reg [9:0] addra,
    output [5:0] valid_out,
    output [3:0] decision
    );

    wire signed [11:0] conv_out_1, conv_out_2, conv_out_3;
    wire signed [11:0] conv2_out_1, conv2_out_2, conv2_out_3;
    wire signed [11:0] max_value_1, max_value_2, max_value_3;
    wire signed [11:0] max2_value_1, max2_value_2, max2_value_3;
    wire signed [11:0] fc_out_data;

    // Module Instantiation
    conv1_layer conv1_layer(
      .clk(clk),
      .rst_n(rst_n),
      .data_in(douta),
      .conv_out_1(conv_out_1),
      .conv_out_2(conv_out_2),
      .conv_out_3(conv_out_3),
      .valid_out_conv(valid_out[0])
    );

    maxpool_relu #(.CONV_BIT(12), .HALF_WIDTH(12), .HALF_HEIGHT(12), .
HALF_WIDTH_BIT(4))
    maxpool_relu_1(
      .clk(clk),
      .rst_n(rst_n),
      .valid_in(valid_out[0]),
      .conv_out_1(conv_out_1),
      .conv_out_2(conv_out_2),
      .conv_out_3(conv_out_3),
      .max_value_1(max_value_1),
      .max_value_2(max_value_2),
      .max_value_3(max_value_3),
      .valid_out_relu(valid_out[1])
    );

    conv2_layer conv2_layer(
      .clk(clk),
      .rst_n(rst_n),
      .valid_in(valid_out[1]),
      .max_value_1(max_value_1),
      .max_value_2(max_value_2),
      .max_value_3(max_value_3),
      .conv2_out_1(conv2_out_1),
      .conv2_out_2(conv2_out_2),
      .conv2_out_3(conv2_out_3),
      .valid_out_conv2(valid_out[2])
    );

    maxpool_relu #(.CONV_BIT(12), .HALF_WIDTH(4), .HALF_HEIGHT(4), .HALF_WIDTH_BIT(3))
    maxpool_relu_2(
      .clk(clk),
      .rst_n(rst_n),
      .valid_in(valid_out[2]),
      .conv_out_1(conv2_out_1),
      .conv_out_2(conv2_out_2),
      .conv_out_3(conv2_out_3),
      .max_value_1(max2_value_1),
      .max_value_2(max2_value_2),
      .max_value_3(max2_value_3),
      .valid_out_relu(valid_out[3])
    );

    fully_connected fully_connected(
      .clk(clk),
      .rst_n(rst_n),
      .valid_in(valid_out[3]),
      .data_in_1(max2_value_1),
      .data_in_2(max2_value_2),
      .data_in_3(max2_value_3),
      .data_out(fc_out_data),
      .valid_out_fc(valid_out[4])
    );

    wire [3:0] decision_t1;

    comparator comparator(
      .clk(clk),
      .rst_n(rst_n),
      .valid_in(valid_out[4]),
      .data_in(fc_out_data),
      .decision(decision_t1),
      .valid_out(valid_out[5])
    );

```

```verilog
//    ila_1 outer (
//            .clk(clk),
//            .probe0(valid_out),
//            .probe1(decision_t1),
//            .probe2(douta),
//            .probe3(conv_out_1),
//            .probe4(max_value_1),
//            .probe5(conv2_out_1),
//            .probe6(max2_value_1),
//            .probe7(fc_out_data)
//            );

    reg [3:0] decision_t2;
    reg is_rst;

    always @(posedge clk) begin
        if (~rst_n) begin
            is_rst <= 1;
            decision_t2 <= 0;
            end
        else if(valid_out[5] && is_rst) begin
            decision_t2 <= decision_t1;
            is_rst <= 0;
            end
        end

    assign decision = decision_t2;

    always @(posedge clk) begin
        if (~rst_n)
            addra <= 0;
        else if(addra < 10'd784)
            addra <= addra + 1'b1;
        end

endmodule
```

```verilog
`timescale 1ns / 1ps

//  Design     : 1st Convolution Layer for CNN MNIST dataset

 module conv1_layer (
    input clk,
    input rst_n,
    input [7:0] data_in,
   output [11:0] conv_out_1, conv_out_2, conv_out_3,
   output valid_out_conv
   );

wire [7:0] data_out_0, data_out_1, data_out_2, data_out_3, data_out_4,
 data_out_5, data_out_6, data_out_7, data_out_8, data_out_9,
 data_out_10, data_out_11, data_out_12, data_out_13, data_out_14,
 data_out_15, data_out_16, data_out_17, data_out_18, data_out_19,
 data_out_20, data_out_21, data_out_22, data_out_23, data_out_24;

wire valid_out_buf;

conv1_buf #(.WIDTH(28), .HEIGHT(28), .DATA_BITS(8)) conv1_buf(
  .clk(clk),
  .rst_n(rst_n),
  .data_in(data_in),
  .data_out_0(data_out_0),
  .data_out_1(data_out_1),
  .data_out_2(data_out_2),
  .data_out_3(data_out_3),
  .data_out_4(data_out_4),
  .data_out_5(data_out_5),
  .data_out_6(data_out_6),
  .data_out_7(data_out_7),
  .data_out_8(data_out_8),
  .data_out_9(data_out_9),
  .data_out_10(data_out_10),
  .data_out_11(data_out_11),
  .data_out_12(data_out_12),
  .data_out_13(data_out_13),
  .data_out_14(data_out_14),
  .data_out_15(data_out_15),
  .data_out_16(data_out_16),
  .data_out_17(data_out_17),
  .data_out_18(data_out_18),
  .data_out_19(data_out_19),
  .data_out_20(data_out_20),
  .data_out_21(data_out_21),
  .data_out_22(data_out_22),
  .data_out_23(data_out_23),
  .data_out_24(data_out_24),
  .valid_out_buf(valid_out_buf)
);
```

```

 conv1_calc conv1_calc(
   .clk(clk),
   .valid_out_buf(valid_out_buf),
   .data_out_0(data_out_0),
   .data_out_1(data_out_1),
   .data_out_2(data_out_2),
   .data_out_3(data_out_3),
   .data_out_4(data_out_4),
   .data_out_5(data_out_5),
   .data_out_6(data_out_6),
   .data_out_7(data_out_7),
   .data_out_8(data_out_8),
   .data_out_9(data_out_9),
   .data_out_10(data_out_10),
   .data_out_11(data_out_11),
   .data_out_12(data_out_12),
   .data_out_13(data_out_13),
   .data_out_14(data_out_14),
   .data_out_15(data_out_15),
   .data_out_16(data_out_16),
   .data_out_17(data_out_17),
   .data_out_18(data_out_18),
   .data_out_19(data_out_19),
   .data_out_20(data_out_20),
   .data_out_21(data_out_21),
   .data_out_22(data_out_22),
   .data_out_23(data_out_23),
   .data_out_24(data_out_24),
   .conv_out_1(conv_out_1),
   .conv_out_2(conv_out_2),
   .conv_out_3(conv_out_3),
   .valid_out_calc(valid_out_conv)
 );

 endmodule
```

```
`timescale 1ns / 1ps

// Description: 1st Convolution Layer for CNN MNIST dataset Input Buffer

module conv1_buf #(parameter WIDTH = 28, HEIGHT = 28, DATA_BITS = 8)(
   input clk,
    input rst_n,
    input [DATA_BITS - 1:0] data_in,
    output reg [DATA_BITS - 1:0] data_out_0, data_out_1, data_out_2, data_out_3, data_out_4,
    data_out_5, data_out_6, data_out_7, data_out_8, data_out_9,
    data_out_10, data_out_11, data_out_12, data_out_13, data_out_14,
    data_out_15, data_out_16, data_out_17, data_out_18, data_out_19,
    data_out_20, data_out_21, data_out_22, data_out_23, data_out_24,
    output reg valid_out_buf
    );

  localparam FILTER_SIZE = 5;

  reg [DATA_BITS - 1:0] buffer [0:WIDTH * FILTER_SIZE - 1];
  reg [DATA_BITS - 1:0] buf_idx;
  reg [4:0] w_idx, h_idx;
  reg [2:0] buf_flag;  // 0 ~ 4
  reg state;

  always @(posedge clk) begin
    if (~rst_n) begin
      buf_idx <= -1;
      w_idx <= 0;
      h_idx <= 0;
      buf_flag <= 0;
      state <= 0;
      valid_out_buf <= 0;
      data_out_0 <= 12'bx;
      data_out_1 <= 12'bx;
      data_out_2 <= 12'bx;
      data_out_3 <= 12'bx;
      data_out_4 <= 12'bx;
      data_out_5 <= 12'bx;
      data_out_6 <= 12'bx;
      data_out_7 <= 12'bx;
      data_out_8 <= 12'bx;
      data_out_9 <= 12'bx;
      data_out_10 <= 12'bx;
      data_out_11 <= 12'bx;
      data_out_12 <= 12'bx;
      data_out_13 <= 12'bx;
      data_out_14 <= 12'bx;
      data_out_15 <= 12'bx;
      data_out_16 <= 12'bx;
      data_out_17 <= 12'bx;
      data_out_18 <= 12'bx;
      data_out_19 <= 12'bx;
      data_out_20 <= 12'bx;
      data_out_21 <= 12'bx;
      data_out_22 <= 12'bx;
      data_out_23 <= 12'bx;
      data_out_24 <= 12'bx;
      end
    else begin
      buf_idx <= buf_idx + 1;

      if (buf_idx == WIDTH * FILTER_SIZE - 1) // buffer size = 140 = 28(w) * 5(h)
        buf_idx <= 0;

      buffer[buf_idx] <= data_in;  // data input

      // Wait until first 140 input data filled in buffer
      if (!state) begin
        if (buf_idx == WIDTH * FILTER_SIZE - 1)
          state <= 1'b1;
        end
      else begin // valid state
        w_idx <= w_idx + 1'b1; // move right

        if (w_idx == WIDTH - FILTER_SIZE + 1)
          valid_out_buf <= 1'b0; // unvalid area
        else if (w_idx == WIDTH - 1) begin
          buf_flag <= buf_flag + 1'b1;
          if (buf_flag == FILTER_SIZE - 1)
            buf_flag <= 0;
          w_idx <= 0;
          if (h_idx == HEIGHT - FILTER_SIZE) begin  // done 1 input read -> 28 * 28
            h_idx <= 0;
            state <= 1'b0;
            end
          h_idx <= h_idx + 1'b1;
          end
        else if(w_idx == 0)
          valid_out_buf <= 1'b1; // start valid area

      // Buffer Selection -> 5 * 5
```

```verilog
        if (buf_flag == 3'd0) begin
          data_out_0 <= buffer[w_idx];
          data_out_1 <= buffer[w_idx + 1];
          data_out_2 <= buffer[w_idx + 2];
          data_out_3 <= buffer[w_idx + 3];
          data_out_4 <= buffer[w_idx + 4];

          data_out_5 <= buffer[w_idx + WIDTH];
          data_out_6 <= buffer[w_idx + 1 + WIDTH];
          data_out_7 <= buffer[w_idx + 2 + WIDTH];
          data_out_8 <= buffer[w_idx + 3 + WIDTH];
          data_out_9 <= buffer[w_idx + 4 + WIDTH];

          data_out_10 <= buffer[w_idx + WIDTH * 2];
          data_out_11 <= buffer[w_idx + 1 + WIDTH * 2];
          data_out_12 <= buffer[w_idx + 2 + WIDTH * 2];
          data_out_13 <= buffer[w_idx + 3 + WIDTH * 2];
          data_out_14 <= buffer[w_idx + 4 + WIDTH * 2];

          data_out_15 <= buffer[w_idx + WIDTH * 3];
          data_out_16 <= buffer[w_idx + 1 + WIDTH * 3];
          data_out_17 <= buffer[w_idx + 2 + WIDTH * 3];
          data_out_18 <= buffer[w_idx + 3 + WIDTH * 3];
          data_out_19 <= buffer[w_idx + 4 + WIDTH * 3];

          data_out_20 <= buffer[w_idx + WIDTH * 4];
          data_out_21 <= buffer[w_idx + 1 + WIDTH * 4];
          data_out_22 <= buffer[w_idx + 2 + WIDTH * 4];
          data_out_23 <= buffer[w_idx + 3 + WIDTH * 4];
          data_out_24 <= buffer[w_idx + 4 + WIDTH * 4];
          end
        else if (buf_flag == 3'd1) begin
          data_out_0 <= buffer[w_idx + WIDTH];
          data_out_1 <= buffer[w_idx + 1 + WIDTH];
          data_out_2 <= buffer[w_idx + 2 + WIDTH];
          data_out_3 <= buffer[w_idx + 3 + WIDTH];
          data_out_4 <= buffer[w_idx + 4 + WIDTH];

          data_out_5 <= buffer[w_idx + WIDTH * 2];
          data_out_6 <= buffer[w_idx + 1 + WIDTH * 2];
          data_out_7 <= buffer[w_idx + 2 + WIDTH * 2];
          data_out_8 <= buffer[w_idx + 3 + WIDTH * 2];
          data_out_9 <= buffer[w_idx + 4 + WIDTH * 2];

          data_out_10 <= buffer[w_idx + WIDTH * 3];
          data_out_11 <= buffer[w_idx + 1 + WIDTH * 3];
          data_out_12 <= buffer[w_idx + 2 + WIDTH * 3];
          data_out_13 <= buffer[w_idx + 3 + WIDTH * 3];
          data_out_14 <= buffer[w_idx + 4 + WIDTH * 3];

          data_out_15 <= buffer[w_idx + WIDTH * 4];
          data_out_16 <= buffer[w_idx + 1 + WIDTH * 4];
          data_out_17 <= buffer[w_idx + 2 + WIDTH * 4];
          data_out_18 <= buffer[w_idx + 3 + WIDTH * 4];
          data_out_19 <= buffer[w_idx + 4 + WIDTH * 4];

          data_out_20 <= buffer[w_idx];
          data_out_21 <= buffer[w_idx + 1];
          data_out_22 <= buffer[w_idx + 2];
          data_out_23 <= buffer[w_idx + 3];
          data_out_24 <= buffer[w_idx + 4];
          end
        else if (buf_flag == 3'd2) begin
          data_out_0 <= buffer[w_idx + WIDTH * 2];
          data_out_1 <= buffer[w_idx + 1 + WIDTH * 2];
          data_out_2 <= buffer[w_idx + 2 + WIDTH * 2];
          data_out_3 <= buffer[w_idx + 3 + WIDTH * 2];
          data_out_4 <= buffer[w_idx + 4 + WIDTH * 2];

          data_out_5 <= buffer[w_idx + WIDTH * 3];
          data_out_6 <= buffer[w_idx + 1 + WIDTH * 3];
          data_out_7 <= buffer[w_idx + 2 + WIDTH * 3];
          data_out_8 <= buffer[w_idx + 3 + WIDTH * 3];
          data_out_9 <= buffer[w_idx + 4 + WIDTH * 3];

          data_out_10 <= buffer[w_idx + WIDTH * 4];
          data_out_11 <= buffer[w_idx + 1 + WIDTH * 4];
          data_out_12 <= buffer[w_idx + 2 + WIDTH * 4];
          data_out_13 <= buffer[w_idx + 3 + WIDTH * 4];
          data_out_14 <= buffer[w_idx + 4 + WIDTH * 4];

          data_out_15 <= buffer[w_idx];
          data_out_16 <= buffer[w_idx + 1];
          data_out_17 <= buffer[w_idx + 2];
          data_out_18 <= buffer[w_idx + 3];
          data_out_19 <= buffer[w_idx + 4];

          data_out_20 <= buffer[w_idx + WIDTH];
          data_out_21 <= buffer[w_idx + 1 + WIDTH];
          data_out_22 <= buffer[w_idx + 2 + WIDTH];
          data_out_23 <= buffer[w_idx + 3 + WIDTH];
          data_out_24 <= buffer[w_idx + 4 + WIDTH];
          end
        else if (buf_flag == 3'd3) begin
          data_out_0 <= buffer[w_idx + WIDTH * 3];
          data_out_1 <= buffer[w_idx + 1 + WIDTH * 3];
          data_out_2 <= buffer[w_idx + 2 + WIDTH * 3];
          data_out_3 <= buffer[w_idx + 3 + WIDTH * 3];
          data_out_4 <= buffer[w_idx + 4 + WIDTH * 3];

          data_out_5 <= buffer[w_idx + WIDTH * 4];
          data_out_6 <= buffer[w_idx + 1 + WIDTH * 4];
          data_out_7 <= buffer[w_idx + 2 + WIDTH * 4];
          data_out_8 <= buffer[w_idx + 3 + WIDTH * 4];
          data_out_9 <= buffer[w_idx + 4 + WIDTH * 4];

          data_out_10 <= buffer[w_idx];
          data_out_11 <= buffer[w_idx + 1];
          data_out_12 <= buffer[w_idx + 2];
          data_out_13 <= buffer[w_idx + 3];
          data_out_14 <= buffer[w_idx + 4];

          data_out_15 <= buffer[w_idx + WIDTH];
          data_out_16 <= buffer[w_idx + 1 + WIDTH];
          data_out_17 <= buffer[w_idx + 2 + WIDTH];
          data_out_18 <= buffer[w_idx + 3 + WIDTH];
          data_out_19 <= buffer[w_idx + 4 + WIDTH];

          data_out_20 <= buffer[w_idx + WIDTH * 2];
          data_out_21 <= buffer[w_idx + 1 + WIDTH * 2];
          data_out_22 <= buffer[w_idx + 2 + WIDTH * 2];
          data_out_23 <= buffer[w_idx + 3 + WIDTH * 2];
          data_out_24 <= buffer[w_idx + 4 + WIDTH * 2];
          end
        else if (buf_flag == 3'd4) begin
          data_out_0 <= buffer[w_idx + WIDTH * 4];
          data_out_1 <= buffer[w_idx + 1 + WIDTH * 4];
          data_out_2 <= buffer[w_idx + 2 + WIDTH * 4];
          data_out_3 <= buffer[w_idx + 3 + WIDTH * 4];
          data_out_4 <= buffer[w_idx + 4 + WIDTH * 4];

          data_out_5 <= buffer[w_idx];
```

```
            data_out_6 <= buffer[w_idx + 1];
            data_out_7 <= buffer[w_idx + 2];
            data_out_8 <= buffer[w_idx + 3];
            data_out_9 <= buffer[w_idx + 4];

            data_out_10 <= buffer[w_idx + WIDTH];
            data_out_11 <= buffer[w_idx + 1 + WIDTH];
            data_out_12 <= buffer[w_idx + 2 + WIDTH];
            data_out_13 <= buffer[w_idx + 3 + WIDTH];
            data_out_14 <= buffer[w_idx + 4 + WIDTH];

            data_out_15 <= buffer[w_idx + WIDTH * 2];
            data_out_16 <= buffer[w_idx + 1 + WIDTH * 2];
            data_out_17 <= buffer[w_idx + 2 + WIDTH * 2];
            data_out_18 <= buffer[w_idx + 3 + WIDTH * 2];
            data_out_19 <= buffer[w_idx + 4 + WIDTH * 2];

            data_out_20 <= buffer[w_idx + WIDTH * 3];
            data_out_21 <= buffer[w_idx + 1 + WIDTH * 3];
            data_out_22 <= buffer[w_idx + 2 + WIDTH * 3];
            data_out_23 <= buffer[w_idx + 3 + WIDTH * 3];
            data_out_24 <= buffer[w_idx + 4 + WIDTH * 3];
            end

          end

        end

      end

    endmodule
```

```
`timescale 1ns / 1ps

 //  Design     : 1st Convolution Layer for CNN MNIST dataset
 //                 Convolution Sum Calculation

 module conv1_calc #(parameter WIDTH = 28, HEIGHT = 28, DATA_BITS = 8)
(
    input clk,
   input valid_out_buf,
   input [DATA_BITS - 1:0] data_out_0, data_out_1, data_out_2, data_out_3, data_out_4,
   data_out_5, data_out_6, data_out_7, data_out_8, data_out_9,
  data_out_10, data_out_11, data_out_12, data_out_13, data_out_14,
  data_out_15, data_out_16, data_out_17, data_out_18, data_out_19,
  data_out_20, data_out_21, data_out_22, data_out_23, data_out_24,
  output signed [11:0] conv_out_1, conv_out_2, conv_out_3,
  output valid_out_calc
);

localparam FILTER_SIZE = 5;
localparam CHANNEL_LEN = 3;

wire signed [DATA_BITS - 1:0] weight_1 [0:FILTER_SIZE * FILTER_SIZE - 1];
wire signed [DATA_BITS - 1:0] weight_2 [0:FILTER_SIZE * FILTER_SIZE - 1];
wire signed [DATA_BITS - 1:0] weight_3 [0:FILTER_SIZE * FILTER_SIZE - 1];
wire signed [DATA_BITS - 1:0] bias [0:CHANNEL_LEN - 1];

assign weight_1[0] = 8'h06;
assign weight_1[1] = 8'h00;
assign weight_1[2] = 8'h1b;
assign weight_1[3] = 8'h29;
assign weight_1[4] = 8'h35;
assign weight_1[5] = 8'he7;
assign weight_1[6] = 8'h13;
assign weight_1[7] = 8'hfd;
assign weight_1[8] = 8'h0b;
assign weight_1[9] = 8'h3e;
assign weight_1[10] = 8'hf1;
assign weight_1[11] = 8'hf3;
assign weight_1[12] = 8'h0e;
assign weight_1[13] = 8'hf5;
assign weight_1[14] = 8'hf6;
assign weight_1[15] = 8'hd2;
assign weight_1[16] = 8'hd8;
assign weight_1[17] = 8'hdb;
assign weight_1[18] = 8'hd6;
assign weight_1[19] = 8'hd9;
assign weight_1[20] = 8'he7;
assign weight_1[21] = 8'he3;
assign weight_1[22] = 8'hd6;
assign weight_1[23] = 8'he2;
assign weight_1[24] = 8'hdc;

```

```
    assign weight_2[0] = 8'h08;
    assign weight_2[1] = 8'h00;
    assign weight_2[2] = 8'h06;
    assign weight_2[3] = 8'hf5;
    assign weight_2[4] = 8'hfd;
    assign weight_2[5] = 8'h1a;
    assign weight_2[6] = 8'hf0;
    assign weight_2[7] = 8'h19;
    assign weight_2[8] = 8'h0a;
    assign weight_2[9] = 8'h24;
    assign weight_2[10] = 8'h0f;
    assign weight_2[11] = 8'h12;
    assign weight_2[12] = 8'h21;
    assign weight_2[13] = 8'h46;
    assign weight_2[14] = 8'h41;
    assign weight_2[15] = 8'h0d;
    assign weight_2[16] = 8'h3a;
    assign weight_2[17] = 8'h6c;
    assign weight_2[18] = 8'h76;
    assign weight_2[19] = 8'h2e;
    assign weight_2[20] = 8'h31;
    assign weight_2[21] = 8'h1d;
    assign weight_2[22] = 8'h58;
    assign weight_2[23] = 8'h35;
    assign weight_2[24] = 8'h4e;

    assign weight_3[0] = 8'h21;
    assign weight_3[1] = 8'h1a;
    assign weight_3[2] = 8'h33;
    assign weight_3[3] = 8'h29;
    assign weight_3[4] = 8'h47;
    assign weight_3[5] = 8'h1b;
    assign weight_3[6] = 8'h1b;
    assign weight_3[7] = 8'h20;
    assign weight_3[8] = 8'h0c;
    assign weight_3[9] = 8'hf8;
    assign weight_3[10] = 8'hd8;
    assign weight_3[11] = 8'h02;
    assign weight_3[12] = 8'he0;
    assign weight_3[13] = 8'hdd;
    assign weight_3[14] = 8'hd5;
    assign weight_3[15] = 8'hdc;
    assign weight_3[16] = 8'hc3;
    assign weight_3[17] = 8'hce;
```

```
  assign weight_3[18] = 8'hbc;
  assign weight_3[19] = 8'he5;
  assign weight_3[20] = 8'hd5;
  assign weight_3[21] = 8'hd9;
  assign weight_3[22] = 8'hda;
  assign weight_3[23] = 8'h03;
  assign weight_3[24] = 8'hf9;

  assign bias[0] = 8'h07;
  assign bias[1] = 8'h0f;
  assign bias[2] = 8'h2f;

//integer is_fill = 0;

//always @(*) begin
//if(is_fill == 0) begin
//     weight_1[0] <= 8'h06;
//     ...
//     weight_1[24] <= 8'hdc;

//     weight_2[0] <= 8'h08;
//     ...
//     weight_2[24] <= 8'h4e;

//     weight_3[0] <= 8'h21;
//     ...
//     weight_3[24] <= 8'hf9;

//     bias[0] <= 8'h07;
//     bias[1] <= 8'h0f;
//     bias[2] <= 8'h2f;

//     is_fill = 1;

//     end
//     end


  wire signed [19:0] calc_out_1, calc_out_2, calc_out_3;
  wire signed [DATA_BITS:0] exp_data [0:FILTER_SIZE * FILTER_SIZE - 1];
  wire signed [11:0] exp_bias [0:CHANNEL_LEN - 1];

// initial begin
//    $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/
conv1_weight_1.txt", weight_1);
//   $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/conv1_weight_2.txt", weight_2);
//   $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/conv1_weight_3.txt", weight_3);
//   $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/conv1_bias.txt", bias);
// end



  // Unsigned -> Signed
  assign exp_data[0] = {1'd0, data_out_0};
  assign exp_data[1] = {1'd0, data_out_1};
  assign exp_data[2] = {1'd0, data_out_2};
  assign exp_data[3] = {1'd0, data_out_3};
  assign exp_data[4] = {1'd0, data_out_4};
  assign exp_data[5] = {1'd0, data_out_5};
  assign exp_data[6] = {1'd0, data_out_6};
  assign exp_data[7] = {1'd0, data_out_7};
  assign exp_data[8] = {1'd0, data_out_8};
  assign exp_data[9] = {1'd0, data_out_9};
  assign exp_data[10] = {1'd0, data_out_10};
  assign exp_data[11] = {1'd0, data_out_11};
  assign exp_data[12] = {1'd0, data_out_12};
  assign exp_data[13] = {1'd0, data_out_13};
  assign exp_data[14] = {1'd0, data_out_14};
  assign exp_data[15] = {1'd0, data_out_15};
  assign exp_data[16] = {1'd0, data_out_16};
  assign exp_data[17] = {1'd0, data_out_17};
  assign exp_data[18] = {1'd0, data_out_18};
  assign exp_data[19] = {1'd0, data_out_19};
  assign exp_data[20] = {1'd0, data_out_20};
  assign exp_data[21] = {1'd0, data_out_21};
  assign exp_data[22] = {1'd0, data_out_22};
  assign exp_data[23] = {1'd0, data_out_23};
  assign exp_data[24] = {1'd0, data_out_24};

  //  Re-calibration of extracted weight data according to MSB
  assign exp_bias[0] = (bias[0][7] == 1) ? {4'b1111, bias[0]} : {4'd0, bias[0]};
  assign exp_bias[1] = (bias[1][7] == 1) ? {4'b1111, bias[1]} : {4'd0, bias[1]};
  assign exp_bias[2] = (bias[2][7] == 1) ? {4'b1111, bias[2]} : {4'd0,
bias[2]};

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] +
    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] +
    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] +
    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] +
    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];

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] +
    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] +
    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] +
    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] +
    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];

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] +
    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] +
    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] +
    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] +
    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];
```

```verilog

 assign conv_out_1 = calc_out_1[19:8] + exp_bias[0];
 assign conv_out_2 = calc_out_2[19:8] + exp_bias[1];
 assign conv_out_3 = calc_out_3[19:8] + exp_bias[2];

 assign valid_out_calc = valid_out_buf;

//  ila_0 debugg (
//          .clk(clk),
//          .probe0(valid_out_buf),
//          .probe1(valid_out_calc),
//          .probe2(data_out_0),
//          .probe3(conv_out_1),
//          .probe4(weight_1[0]),
//          .probe5(weight_2[0]),
//          .probe6(weight_3[0]),
//          .probe7(bias[0])
//          );

endmodule
```

```verilog
`timescale 1ns / 1ps

 //  Design     : (1) MaxPooling for CNN
     //(2) Activation Function for CNN - ReLU Function

module maxpool_relu #(parameter CONV_BIT = 12, HALF_WIDTH = 12, HALF_HEIGHT = 12, HALF_WIDTH_BIT = 4) (
 input clk,
 input rst_n, // asynchronous reset, active low
 input valid_in,
 input signed [CONV_BIT - 1 : 0] conv_out_1, conv_out_2, conv_out_3,
 output reg [CONV_BIT - 1 : 0] max_value_1, max_value_2, max_value_3,
 output reg valid_out_relu
);

reg signed [CONV_BIT - 1:0] buffer1 [0:HALF_WIDTH - 1];
reg signed [CONV_BIT - 1:0] buffer2 [0:HALF_WIDTH - 1];
reg signed [CONV_BIT - 1:0] buffer3 [0:HALF_WIDTH - 1];

reg [HALF_WIDTH_BIT - 1:0] pcount;
reg state;
reg flag;

```

```
always @(posedge clk) begin
 if(~rst_n) begin
  valid_out_relu <= 0;
  pcount <= 0;
  state <= 0;
  flag <= 0;
     end
 else begin

 if(valid_in == 1'b1) begin
  flag <= ~flag;

  if(flag == 1) begin
   pcount <= pcount + 1;
   if(pcount == HALF_WIDTH - 1) begin
    state <= ~state;
    pcount <= 0;
       end
      end

  if(state == 0) begin // first line
   valid_out_relu <= 0;
   if(flag == 0) begin // first input
    buffer1[pcount] <= conv_out_1;
    buffer2[pcount] <= conv_out_2;
    buffer3[pcount] <= conv_out_3;
       end
   else begin // second input -> comparison
    if(buffer1[pcount] < conv_out_1)
     buffer1[pcount] <= conv_out_1;
    if(buffer2[pcount] < conv_out_2)
        buffer2[pcount] <= conv_out_2;
    if(buffer3[pcount] < conv_out_3)
        buffer3[pcount] <= conv_out_3;
       end
      end
  else begin // second line
   if(flag == 0) begin // third input -> comparison
    valid_out_relu <= 0;
    if(buffer1[pcount] < conv_out_1)
      buffer1[pcount] <= conv_out_1;
    if(buffer2[pcount] < conv_out_2)
      buffer2[pcount] <= conv_out_2;
       if(buffer3[pcount] < conv_out_3)
      buffer3[pcount] <= conv_out_3;
       end
   else begin // fourth input -> comparison + relu
    valid_out_relu <= 1;
    if(buffer1[pcount] < conv_out_1) begin
     if(conv_out_1 > 0)
      max_value_1 <= conv_out_1;
     else
      max_value_1 <= 0;
        end
    else begin
     if(buffer1[pcount] > 0)
      max_value_1 <= buffer1[pcount];
     else
      max_value_1 <= 0;
        end

    if(buffer2[pcount] < conv_out_2) begin
     if(conv_out_2 > 0)
      max_value_2 <= conv_out_2;
     else
      max_value_2 <= 0;
        end
    else begin
     if(buffer2[pcount] > 0)
      max_value_2 <= buffer2[pcount];
     else
      max_value_2 <= 0;
     end

    if(buffer3[pcount] < conv_out_3) begin
     if(conv_out_3 > 0)
      max_value_3 <= conv_out_3;
     else
      max_value_3 <= 0;
     end
    else begin
     if(buffer3[pcount] > 0)
      max_value_3 <= buffer3[pcount];
     else
      max_value_3 <= 0;
     end

       end
```

```
      end
     end
//    end
 else
  valid_out_relu <= 0;
    end
end
endmodule
```

```
`timescale 1ns / 1ps

//  Design     : 2nd Convolution Layer for CNN MNIST dataset

 module conv2_layer (
   input clk,
   input rst_n,
   input valid_in,
   input [11:0] max_value_1, max_value_2, max_value_3,
   output [11:0] conv2_out_1, conv2_out_2, conv2_out_3,
   output valid_out_conv2
 );

 localparam CHANNEL_LEN = 3;
 ///////////////////////////////////////////
  /*wire [11:0] out_data1_0, out_data1_1, out_data1_2, out_data1_3, out_data1_4,
   out_data1_5, out_data1_6, out_data1_7, out_data1_8, out_data1_9,
   out_data1_10, out_data1_11, out_data1_12, out_data1_13, out_data1_14,
   out_data1_15, out_data1_16, out_data1_17, out_data1_18, out_data1_19,
   out_data1_20, out_data1_21, out_data1_22, out_data1_23, out_data1_24,

   out_data2_0, out_data2_1, out_data2_2, out_data2_3, out_data2_4,
   out_data2_5, out_data2_6, out_data2_7, out_data2_8, out_data2_9,
   out_data2_10, out_data2_11, out_data2_12, out_data2_13, out_data2_14,
   out_data2_15, out_data2_16, out_data2_17, out_data2_18, out_data2_19,
   out_data2_20, out_data2_21, out_data2_22, out_data2_23, out_data2_24,

   out_data3_0, out_data3_1, out_data3_2, out_data3_3, out_data3_4,
     out_data3_5, out_data3_6, out_data3_7, out_data3_8, out_data3_9,
     out_data3_10, out_data3_11, out_data3_12, out_data3_13, out_data3_14,
     out_data3_15, out_data3_16, out_data3_17, out_data3_18, out_data3_19,
     out_data3_20, out_data3_21, out_data3_22, out_data3_23, out_data3_24;*/
        /////////////////////////////////
   // Channel 1
   wire [11:0] data_out1_0, data_out1_1, data_out1_2, data_out1_3, data_out1_4,
    data_out1_5, data_out1_6, data_out1_7, data_out1_8, data_out1_9,
    data_out1_10, data_out1_11, data_out1_12, data_out1_13, data_out1_14,
    data_out1_15, data_out1_16, data_out1_17, data_out1_18, data_out1_19,
    data_out1_20, data_out1_21, data_out1_22, data_out1_23, data_out1_24;
   wire valid_out1_buf;

   // Channel 2
   wire [11:0] data_out2_0, data_out2_1, data_out2_2, data_out2_3, data_out2_4,
    data_out2_5, data_out2_6, data_out2_7, data_out2_8, data_out2_9,
    data_out2_10, data_out2_11, data_out2_12, data_out2_13, data_out2_14,
    data_out2_15, data_out2_16, data_out2_17, data_out2_18, data_out2_19,
    data_out2_20, data_out2_21, data_out2_22, data_out2_23, data_out2_24;
   wire valid_out2_buf;

   // Channel 3
   wire [11:0] data_out3_0, data_out3_1, data_out3_2, data_out3_3, data_out3_4,
    data_out3_5, data_out3_6, data_out3_7, data_out3_8, data_out3_9,
    data_out3_10, data_out3_11, data_out3_12, data_out3_13, data_out3_14,
    data_out3_15, data_out3_16, data_out3_17, data_out3_18, data_out3_19,
    data_out3_20, data_out3_21, data_out3_22, data_out3_23, data_out3_24;
   wire valid_out3_buf;

```

```verilog
    wire signed [13:0] conv_out_1, conv_out_2, conv_out_3;
    wire valid_out_buf, valid_out_calc_1, valid_out_calc_2, valid_out_calc_3;
    assign valid_out_buf = valid_out1_buf & valid_out2_buf & valid_out3_buf;
    assign valid_out_conv2 = valid_out_calc_1 & valid_out_calc_2 & valid_out_calc_3;

    wire signed [7:0] bias [0:CHANNEL_LEN - 1];
    wire signed [11:0] exp_bias [0:CHANNEL_LEN - 1];

   conv2_buf #(.WIDTH(12), .HEIGHT(12), .DATA_BITS(12)) conv2_buf_1(
      .clk(clk),
      .rst_n(rst_n),
      .valid_in(valid_in),
      .data_in(max_value_1),
      .data_out_0(data_out1_0),
      .data_out_1(data_out1_1),
      .data_out_2(data_out1_2),
      .data_out_3(data_out1_3),
      .data_out_4(data_out1_4),
      .data_out_5(data_out1_5),
      .data_out_6(data_out1_6),
      .data_out_7(data_out1_7),
      .data_out_8(data_out1_8),
      .data_out_9(data_out1_9),
      .data_out_10(data_out1_10),
      .data_out_11(data_out1_11),
      .data_out_12(data_out1_12),
      .data_out_13(data_out1_13),
      .data_out_14(data_out1_14),
      .data_out_15(data_out1_15),
      .data_out_16(data_out1_16),
      .data_out_17(data_out1_17),
      .data_out_18(data_out1_18),
      .data_out_19(data_out1_19),
      .data_out_20(data_out1_20),
      .data_out_21(data_out1_21),
      .data_out_22(data_out1_22),
      .data_out_23(data_out1_23),
      .data_out_24(data_out1_24),
      .valid_out_buf(valid_out1_buf)
    );

```

```verilog
conv2_buf #(.WIDTH(12), .HEIGHT(12), .DATA_BITS  (12)) conv2_buf_2(
  .clk(clk),
  .rst_n(rst_n),
  .valid_in(valid_in),
  .data_in(max_value_2),
  .data_out_0(data_out2_0),
  .data_out_1(data_out2_1),
  .data_out_2(data_out2_2),
  .data_out_3(data_out2_3),
  .data_out_4(data_out2_4),
  .data_out_5(data_out2_5),
  .data_out_6(data_out2_6),
  .data_out_7(data_out2_7),
  .data_out_8(data_out2_8),
  .data_out_9(data_out2_9),
  .data_out_10(data_out2_10),
  .data_out_11(data_out2_11),
  .data_out_12(data_out2_12),
  .data_out_13(data_out2_13),
  .data_out_14(data_out2_14),
  .data_out_15(data_out2_15),
  .data_out_16(data_out2_16),
  .data_out_17(data_out2_17),
  .data_out_18(data_out2_18),
  .data_out_19(data_out2_19),
  .data_out_20(data_out2_20),
  .data_out_21(data_out2_21),
  .data_out_22(data_out2_22),
  .data_out_23(data_out2_23),
  .data_out_24(data_out2_24),
  .valid_out_buf(valid_out2_buf)
);

conv2_buf #(.WIDTH(12), .HEIGHT(12), .DATA_BITS(12)) conv2_buf_3(
  .clk(clk),
  .rst_n(rst_n),
  .valid_in(valid_in),
  .data_in(max_value_3),
  .data_out_0(data_out3_0),
  .data_out_1(data_out3_1),
  .data_out_2(data_out3_2),
  .data_out_3(data_out3_3),
  .data_out_4(data_out3_4),
  .data_out_5(data_out3_5),
    .data_out_6(data_out3_6),
    .data_out_7(data_out3_7),
    .data_out_8(data_out3_8),
    .data_out_9(data_out3_9),
    .data_out_10(data_out3_10),
    .data_out_11(data_out3_11),
    .data_out_12(data_out3_12),
    .data_out_13(data_out3_13),
    .data_out_14(data_out3_14),
    .data_out_15(data_out3_15),
    .data_out_16(data_out3_16),
    .data_out_17(data_out3_17),
    .data_out_18(data_out3_18),
    .data_out_19(data_out3_19),
    .data_out_20(data_out3_20),
    .data_out_21(data_out3_21),
    .data_out_22(data_out3_22),
    .data_out_23(data_out3_23),
    .data_out_24(data_out3_24),
    .valid_out_buf(valid_out3_buf)
  );

conv2_calc_1 conv2_calc_1(
    .clk(clk),
    .rst_n(rst_n),
    .valid_out_buf(valid_out_buf),
    .data_out1_0(data_out1_0),
    .data_out1_1(data_out1_1),
    .data_out1_2(data_out1_2),
    .data_out1_3(data_out1_3),
    .data_out1_4(data_out1_4),
    .data_out1_5(data_out1_5),
    .data_out1_6(data_out1_6),
    .data_out1_7(data_out1_7),
    .data_out1_8(data_out1_8),
    .data_out1_9(data_out1_9),
    .data_out1_10(data_out1_10),
    .data_out1_11(data_out1_11),
    .data_out1_12(data_out1_12),
    .data_out1_13(data_out1_13),
    .data_out1_14(data_out1_14),
    .data_out1_15(data_out1_15),
    .data_out1_16(data_out1_16),
    .data_out1_17(data_out1_17),
.data_out1_18(data_out1_18),
.data_out1_19(data_out1_19),
.data_out1_20(data_out1_20),
.data_out1_21(data_out1_21),
.data_out1_22(data_out1_22),
.data_out1_23(data_out1_23),
.data_out1_24(data_out1_24),
.data_out2_0(data_out2_0),
.data_out2_1(data_out2_1),
.data_out2_2(data_out2_2),
.data_out2_3(data_out2_3),
.data_out2_4(data_out2_4),
.data_out2_5(data_out2_5),
.data_out2_6(data_out2_6),
.data_out2_7(data_out2_7),
.data_out2_8(data_out2_8),
.data_out2_9(data_out2_9),
.data_out2_10(data_out2_10),
.data_out2_11(data_out2_11),
.data_out2_12(data_out2_12),
.data_out2_13(data_out2_13),
.data_out2_14(data_out2_14),
.data_out2_15(data_out2_15),
.data_out2_16(data_out2_16),
.data_out2_17(data_out2_17),
.data_out2_18(data_out2_18),
.data_out2_19(data_out2_19),
.data_out2_20(data_out2_20),
.data_out2_21(data_out2_21),
.data_out2_22(data_out2_22),
.data_out2_23(data_out2_23),
.data_out2_24(data_out2_24),
.data_out3_0(data_out3_0),
.data_out3_1(data_out3_1),
.data_out3_2(data_out3_2),
.data_out3_3(data_out3_3),
.data_out3_4(data_out3_4),
.data_out3_5(data_out3_5),
.data_out3_6(data_out3_6),
.data_out3_7(data_out3_7),
.data_out3_8(data_out3_8),
.data_out3_9(data_out3_9),
.data_out3_10(data_out3_10),
.data_out3_11(data_out3_11),
```

```verilog
    .data_out3_12(data_out3_12),
    .data_out3_13(data_out3_13),
    .data_out3_14(data_out3_14),
    .data_out3_15(data_out3_15),
    .data_out3_16(data_out3_16),
    .data_out3_17(data_out3_17),
    .data_out3_18(data_out3_18),
    .data_out3_19(data_out3_19),
    .data_out3_20(data_out3_20),
    .data_out3_21(data_out3_21),
    .data_out3_22(data_out3_22),
    .data_out3_23(data_out3_23),
    .data_out3_24(data_out3_24),
    .conv_out_calc(conv_out_1),
    .valid_out_calc(valid_out_calc_1)
);

conv2_calc_2 conv2_calc_2(
    .clk(clk),
    .rst_n(rst_n),
    .valid_out_buf(valid_out_buf),
    .data_out1_0(data_out1_0),
    .data_out1_1(data_out1_1),
    .data_out1_2(data_out1_2),
    .data_out1_3(data_out1_3),
    .data_out1_4(data_out1_4),
    .data_out1_5(data_out1_5),
    .data_out1_6(data_out1_6),
    .data_out1_7(data_out1_7),
    .data_out1_8(data_out1_8),
    .data_out1_9(data_out1_9),
    .data_out1_10(data_out1_10),
    .data_out1_11(data_out1_11),
    .data_out1_12(data_out1_12),
    .data_out1_13(data_out1_13),
    .data_out1_14(data_out1_14),
    .data_out1_15(data_out1_15),
    .data_out1_16(data_out1_16),
    .data_out1_17(data_out1_17),
    .data_out1_18(data_out1_18),
    .data_out1_19(data_out1_19),
    .data_out1_20(data_out1_20),
    .data_out1_21(data_out1_21),
    .data_out1_22(data_out1_22),
    .data_out1_23(data_out1_23),
    .data_out1_24(data_out1_24),
    .data_out2_0(data_out2_0),
    .data_out2_1(data_out2_1),
    .data_out2_2(data_out2_2),
    .data_out2_3(data_out2_3),
    .data_out2_4(data_out2_4),
    .data_out2_5(data_out2_5),
    .data_out2_6(data_out2_6),
    .data_out2_7(data_out2_7),
    .data_out2_8(data_out2_8),
    .data_out2_9(data_out2_9),
    .data_out2_10(data_out2_10),
    .data_out2_11(data_out2_11),
    .data_out2_12(data_out2_12),
    .data_out2_13(data_out2_13),
    .data_out2_14(data_out2_14),
    .data_out2_15(data_out2_15),
    .data_out2_16(data_out2_16),
    .data_out2_17(data_out2_17),
    .data_out2_18(data_out2_18),
    .data_out2_19(data_out2_19),
    .data_out2_20(data_out2_20),
    .data_out2_21(data_out2_21),
    .data_out2_22(data_out2_22),
    .data_out2_23(data_out2_23),
    .data_out2_24(data_out2_24),
    .data_out3_0(data_out3_0),
    .data_out3_1(data_out3_1),
    .data_out3_2(data_out3_2),
    .data_out3_3(data_out3_3),
    .data_out3_4(data_out3_4),
    .data_out3_5(data_out3_5),
    .data_out3_6(data_out3_6),
    .data_out3_7(data_out3_7),
    .data_out3_8(data_out3_8),
    .data_out3_9(data_out3_9),
    .data_out3_10(data_out3_10),
    .data_out3_11(data_out3_11),
    .data_out3_12(data_out3_12),
    .data_out3_13(data_out3_13),
    .data_out3_14(data_out3_14),
    .data_out3_15(data_out3_15),
    .data_out3_16(data_out3_16),
    .data_out3_17(data_out3_17),
    .data_out3_18(data_out3_18),
    .data_out3_19(data_out3_19),
    .data_out3_20(data_out3_20),
    .data_out3_21(data_out3_21),
    .data_out3_22(data_out3_22),
    .data_out3_23(data_out3_23),
    .data_out3_24(data_out3_24),
    .conv_out_calc(conv_out_2),
    .valid_out_calc(valid_out_calc_2)
);

conv2_calc_3 conv2_calc_3(
    .clk(clk),
    .rst_n(rst_n),
    .valid_out_buf(valid_out_buf),
    .data_out1_0(data_out1_0),
    .data_out1_1(data_out1_1),
    .data_out1_2(data_out1_2),
    .data_out1_3(data_out1_3),
    .data_out1_4(data_out1_4),
    .data_out1_5(data_out1_5),
    .data_out1_6(data_out1_6),
    .data_out1_7(data_out1_7),
    .data_out1_8(data_out1_8),
    .data_out1_9(data_out1_9),
    .data_out1_10(data_out1_10),
    .data_out1_11(data_out1_11),
    .data_out1_12(data_out1_12),
    .data_out1_13(data_out1_13),
    .data_out1_14(data_out1_14),
    .data_out1_15(data_out1_15),
    .data_out1_16(data_out1_16),
    .data_out1_17(data_out1_17),
    .data_out1_18(data_out1_18),
    .data_out1_19(data_out1_19),
    .data_out1_20(data_out1_20),
    .data_out1_21(data_out1_21),
    .data_out1_22(data_out1_22),
    .data_out1_23(data_out1_23),
    .data_out1_24(data_out1_24),
    .data_out2_0(data_out2_0),
    .data_out2_1(data_out2_1),
    .data_out2_2(data_out2_2),
    .data_out2_3(data_out2_3),
    .data_out2_4(data_out2_4),
    .data_out2_5(data_out2_5),
    .data_out2_6(data_out2_6),
    .data_out2_7(data_out2_7),
    .data_out2_8(data_out2_8),
    .data_out2_9(data_out2_9),
    .data_out2_10(data_out2_10),
    .data_out2_11(data_out2_11),
    .data_out2_12(data_out2_12),
    .data_out2_13(data_out2_13),
    .data_out2_14(data_out2_14),
    .data_out2_15(data_out2_15),
    .data_out2_16(data_out2_16),
    .data_out2_17(data_out2_17),
    .data_out2_18(data_out2_18),
    .data_out2_19(data_out2_19),
    .data_out2_20(data_out2_20),
    .data_out2_21(data_out2_21),
    .data_out2_22(data_out2_22),
    .data_out2_23(data_out2_23),
    .data_out2_24(data_out2_24),
    .data_out3_0(data_out3_0),
    .data_out3_1(data_out3_1),
    .data_out3_2(data_out3_2),
    .data_out3_3(data_out3_3),
    .data_out3_4(data_out3_4),
    .data_out3_5(data_out3_5),
    .data_out3_6(data_out3_6),
    .data_out3_7(data_out3_7),
    .data_out3_8(data_out3_8),
    .data_out3_9(data_out3_9),
    .data_out3_10(data_out3_10),
    .data_out3_11(data_out3_11),
    .data_out3_12(data_out3_12),
    .data_out3_13(data_out3_13),
    .data_out3_14(data_out3_14),
    .data_out3_15(data_out3_15),
    .data_out3_16(data_out3_16),
    .data_out3_17(data_out3_17),
    .data_out3_18(data_out3_18),
    .data_out3_19(data_out3_19),
    .data_out3_20(data_out3_20),
    .data_out3_21(data_out3_21),
    .data_out3_22(data_out3_22),
    .data_out3_23(data_out3_23),
    .data_out3_24(data_out3_24),
    .conv_out_calc(conv_out_3),
    .valid_out_calc(valid_out_calc_3)
);

    assign bias[0] = 8'h0f;
    assign bias[1] = 8'hd5;
    assign bias[2] = 8'h13;

// integer is_fill = 0;

//always @(*) begin
//if(is_fill == 0) begin
////   $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/conv2_bias.txt", bias);
//   bias[0] <= 8'h0f;
//   bias[1] <= 8'hd5;
//   bias[2] <= 8'h13;
//   is_fill <= 1;
//   end
// end

   assign exp_bias[0] = (bias[0][7] == 1) ? {4'b1111, bias[0]} : {4'b0000, bias[0]};
   assign exp_bias[1] = (bias[1][7] == 1) ? {4'b1111, bias[1]} : {4'b0000, bias[1]};
   assign exp_bias[2] = (bias[2][7] == 1) ? {4'b1111, bias[2]} : {4'b0000, bias[2]};

   assign conv2_out_1 = conv_out_1[13:1] + exp_bias[0];
   assign conv2_out_2 = conv_out_2[13:1] + exp_bias[1];
   assign conv2_out_3 = conv_out_3[13:1] + exp_bias[2];

   endmodule
```

```
 `timescale 1ns / 1ps

 //  Design     : 2nd Convolution Layer for CNN MNIST dataset Input Buffer

   module conv2_buf #(parameter WIDTH = 12, HEIGHT = 12, DATA_BITS = 12)(
    input clk,
    input rst_n,
    input valid_in,
    input [DATA_BITS - 1:0] data_in,
    output reg [DATA_BITS - 1:0] data_out_0, data_out_1, data_out_2, data_out_3, data_out_4,
    data_out_5, data_out_6, data_out_7, data_out_8, data_out_9,
    data_out_10, data_out_11, data_out_12, data_out_13, data_out_14,
    data_out_15, data_out_16, data_out_17, data_out_18, data_out_19,
    data_out_20, data_out_21, data_out_22, data_out_23, data_out_24,
    output reg valid_out_buf
  );

  localparam FILTER_SIZE = 5;

  reg [DATA_BITS - 1:0] buffer [0:WIDTH * FILTER_SIZE - 1];
  reg [DATA_BITS - 1:0] buf_idx;
  reg [4:0] w_idx, h_idx;
  reg [2:0] buf_flag;  // 0 ~ 4
  reg state;

  always @(posedge clk) begin
    if(~rst_n) begin
      buf_idx <= 0;
      w_idx <= 0;
      h_idx <= 0;
      buf_flag <= 0;
      state <= 0;
      valid_out_buf <= 0;
      data_out_0 <= 12'bx;
      data_out_1 <= 12'bx;
      data_out_2 <= 12'bx;
      data_out_3 <= 12'bx;
      data_out_4 <= 12'bx;
      data_out_5 <= 12'bx;
      data_out_6 <= 12'bx;
      data_out_7 <= 12'bx;
      data_out_8 <= 12'bx;
      data_out_9 <= 12'bx;
      data_out_10 <= 12'bx;
      data_out_11 <= 12'bx;
      data_out_12 <= 12'bx;
      data_out_13 <= 12'bx;
      data_out_14 <= 12'bx;
      data_out_15 <= 12'bx;
      data_out_16 <= 12'bx;
      data_out_17 <= 12'bx;
      data_out_18 <= 12'bx;
      data_out_19 <= 12'bx;
      data_out_20 <= 12'bx;
      data_out_21 <= 12'bx;
      data_out_22 <= 12'bx;
      data_out_23 <= 12'bx;
      data_out_24 <= 12'bx;
    end else begin
    if(valid_in) begin
      buf_idx <= buf_idx + 1'b1;
      if(buf_idx == WIDTH * FILTER_SIZE - 1) begin // buffer size = 140 = 28(w) * 5(h)
        buf_idx <= 0;
      end

      buffer[buf_idx] <= data_in;  // data input

      // Wait until first 140 input data filled in buffer
      if(!state) begin
        if(buf_idx == WIDTH * FILTER_SIZE - 1) begin
          state <= 1;
        end
      end else begin // valid state
        w_idx <= w_idx + 1'b1; // move right

       if(w_idx == WIDTH - FILTER_SIZE + 1) begin
         valid_out_buf <= 1'b0;  // unvalid area
       end else if(w_idx == WIDTH - 1) begin
         buf_flag <= buf_flag + 1;
         if(buf_flag == FILTER_SIZE - 1) begin
           buf_flag <= 0;
         end

         w_idx <= 0;

         if(h_idx == HEIGHT - FILTER_SIZE) begin // done 1 input read -> 28 * 28
           h_idx <= 0;
           state <= 0;
         end
           h_idx <= h_idx + 1;

       end else if(w_idx == 0) begin
         valid_out_buf <= 1'b1;  // start valid area
       end

       // Buffer Selection -> 5 * 5
      if(buf_flag == 3'd0) begin
        data_out_0 <= buffer[w_idx];
        data_out_1 <= buffer[w_idx + 1];
        data_out_2 <= buffer[w_idx + 2];
        data_out_3 <= buffer[w_idx + 3];
        data_out_4 <= buffer[w_idx + 4];

        data_out_5 <= buffer[w_idx + WIDTH];
        data_out_6 <= buffer[w_idx + 1 + WIDTH];
        data_out_7 <= buffer[w_idx + 2 + WIDTH];
        data_out_8 <= buffer[w_idx + 3 + WIDTH];
        data_out_9 <= buffer[w_idx + 4 + WIDTH];

        data_out_10 <= buffer[w_idx + WIDTH * 2];
        data_out_11 <= buffer[w_idx + 1 + WIDTH * 2];
        data_out_12 <= buffer[w_idx + 2 + WIDTH * 2];
        data_out_13 <= buffer[w_idx + 3 + WIDTH * 2];
        data_out_14 <= buffer[w_idx + 4 + WIDTH * 2];

        data_out_15 <= buffer[w_idx + WIDTH * 3];
        data_out_16 <= buffer[w_idx + 1 + WIDTH * 3];
        data_out_17 <= buffer[w_idx + 2 + WIDTH * 3];
        data_out_18 <= buffer[w_idx + 3 + WIDTH * 3];
        data_out_19 <= buffer[w_idx + 4 + WIDTH * 3];

        data_out_20 <= buffer[w_idx + WIDTH * 4];
        data_out_21 <= buffer[w_idx + 1 + WIDTH * 4];
        data_out_22 <= buffer[w_idx + 2 + WIDTH * 4];
        data_out_23 <= buffer[w_idx + 3 + WIDTH * 4];
        data_out_24 <= buffer[w_idx + 4 + WIDTH * 4];
      end else if(buf_flag == 3'd1) begin
        data_out_0 <= buffer[w_idx + WIDTH];
        data_out_1 <= buffer[w_idx + 1 + WIDTH];
        data_out_2 <= buffer[w_idx + 2 + WIDTH];
        data_out_3 <= buffer[w_idx + 3 + WIDTH];
        data_out_4 <= buffer[w_idx + 4 + WIDTH];

        data_out_5 <= buffer[w_idx + WIDTH * 2];
        data_out_6 <= buffer[w_idx + 1 + WIDTH * 2];
        data_out_7 <= buffer[w_idx + 2 + WIDTH * 2];
        data_out_8 <= buffer[w_idx + 3 + WIDTH * 2];
        data_out_9 <= buffer[w_idx + 4 + WIDTH * 2];

        data_out_10 <= buffer[w_idx + WIDTH * 3];
        data_out_11 <= buffer[w_idx + 1 + WIDTH * 3];
        data_out_12 <= buffer[w_idx + 2 + WIDTH * 3];
        data_out_13 <= buffer[w_idx + 3 + WIDTH * 3];
        data_out_14 <= buffer[w_idx + 4 + WIDTH * 3];

        data_out_15 <= buffer[w_idx + WIDTH * 4];
        data_out_16 <= buffer[w_idx + 1 + WIDTH * 4];
        data_out_17 <= buffer[w_idx + 2 + WIDTH * 4];
        data_out_18 <= buffer[w_idx + 3 + WIDTH * 4];
        data_out_19 <= buffer[w_idx + 4 + WIDTH * 4];

        data_out_20 <= buffer[w_idx];
        data_out_21 <= buffer[w_idx + 1];
        data_out_22 <= buffer[w_idx + 2];
        data_out_23 <= buffer[w_idx + 3];
        data_out_24 <= buffer[w_idx + 4];
      end else if(buf_flag == 3'd2) begin
        data_out_0 <= buffer[w_idx + WIDTH * 2];
        data_out_1 <= buffer[w_idx + 1 + WIDTH * 2];
        data_out_2 <= buffer[w_idx + 2 + WIDTH * 2];
        data_out_3 <= buffer[w_idx + 3 + WIDTH * 2];
        data_out_4 <= buffer[w_idx + 4 + WIDTH * 2];

        data_out_5 <= buffer[w_idx + WIDTH * 3];
        data_out_6 <= buffer[w_idx + 1 + WIDTH * 3];
        data_out_7 <= buffer[w_idx + 2 + WIDTH * 3];
        data_out_8 <= buffer[w_idx + 3 + WIDTH * 3];
        data_out_9 <= buffer[w_idx + 4 + WIDTH * 3];

        data_out_10 <= buffer[w_idx + WIDTH * 4];
        data_out_11 <= buffer[w_idx + 1 + WIDTH * 4];
        data_out_12 <= buffer[w_idx + 2 + WIDTH * 4];
        data_out_13 <= buffer[w_idx + 3 + WIDTH * 4];
        data_out_14 <= buffer[w_idx + 4 + WIDTH * 4];

        data_out_15 <= buffer[w_idx];
        data_out_16 <= buffer[w_idx + 1];
        data_out_17 <= buffer[w_idx + 2];
        data_out_18 <= buffer[w_idx + 3];
        data_out_19 <= buffer[w_idx + 4];

        data_out_20 <= buffer[w_idx + WIDTH];
        data_out_21 <= buffer[w_idx + 1 + WIDTH];
        data_out_22 <= buffer[w_idx + 2 + WIDTH];
        data_out_23 <= buffer[w_idx + 3 + WIDTH];
        data_out_24 <= buffer[w_idx + 4 + WIDTH];
      end else if(buf_flag == 3'd3) begin
        data_out_0 <= buffer[w_idx + WIDTH * 3];
        data_out_1 <= buffer[w_idx + 1 + WIDTH * 3];
        data_out_2 <= buffer[w_idx + 2 + WIDTH * 3];
        data_out_3 <= buffer[w_idx + 3 + WIDTH * 3];
        data_out_4 <= buffer[w_idx + 4 + WIDTH * 3];

        data_out_5 <= buffer[w_idx + WIDTH * 4];
        data_out_6 <= buffer[w_idx + 1 + WIDTH * 4];
        data_out_7 <= buffer[w_idx + 2 + WIDTH * 4];
        data_out_8 <= buffer[w_idx + 3 + WIDTH * 4];
        data_out_9 <= buffer[w_idx + 4 + WIDTH * 4];

        data_out_10 <= buffer[w_idx];
        data_out_11 <= buffer[w_idx + 1];
        data_out_12 <= buffer[w_idx + 2];
        data_out_13 <= buffer[w_idx + 3];
        data_out_14 <= buffer[w_idx + 4];

        data_out_15 <= buffer[w_idx + WIDTH];
        data_out_16 <= buffer[w_idx + 1 + WIDTH];
        data_out_17 <= buffer[w_idx + 2 + WIDTH];
        data_out_18 <= buffer[w_idx + 3 + WIDTH];
        data_out_19 <= buffer[w_idx + 4 + WIDTH];

        data_out_20 <= buffer[w_idx + WIDTH * 2];
        data_out_21 <= buffer[w_idx + 1 + WIDTH * 2];
        data_out_22 <= buffer[w_idx + 2 + WIDTH * 2];
        data_out_23 <= buffer[w_idx + 3 + WIDTH * 2];
        data_out_24 <= buffer[w_idx + 4 + WIDTH * 2];
      end else if(buf_flag == 3'd4) begin
        data_out_0 <= buffer[w_idx + WIDTH * 4];
        data_out_1 <= buffer[w_idx + 1 + WIDTH * 4];
        data_out_2 <= buffer[w_idx + 2 + WIDTH * 4];
        data_out_3 <= buffer[w_idx + 3 + WIDTH * 4];
        data_out_4 <= buffer[w_idx + 4 + WIDTH * 4];
```

```

       data_out_5 <= buffer[w_idx];
       data_out_6 <= buffer[w_idx + 1];
       data_out_7 <= buffer[w_idx + 2];
       data_out_8 <= buffer[w_idx + 3];
       data_out_9 <= buffer[w_idx + 4];

       data_out_10 <= buffer[w_idx + WIDTH];
       data_out_11 <= buffer[w_idx + 1 + WIDTH];
       data_out_12 <= buffer[w_idx + 2 + WIDTH];
       data_out_13 <= buffer[w_idx + 3 + WIDTH];
       data_out_14 <= buffer[w_idx + 4 + WIDTH];

       data_out_15 <= buffer[w_idx + WIDTH * 2];
       data_out_16 <= buffer[w_idx + 1 + WIDTH * 2];
       data_out_17 <= buffer[w_idx + 2 + WIDTH * 2];
       data_out_18 <= buffer[w_idx + 3 + WIDTH * 2];
       data_out_19 <= buffer[w_idx + 4 + WIDTH * 2];

       data_out_20 <= buffer[w_idx + WIDTH * 3];
       data_out_21 <= buffer[w_idx + 1 + WIDTH * 3];
       data_out_22 <= buffer[w_idx + 2 + WIDTH * 3];
       data_out_23 <= buffer[w_idx + 3 + WIDTH * 3];
       data_out_24 <= buffer[w_idx + 4 + WIDTH * 3];
     end
     end
   end
 end
 end
endmodule
```

```
`timescale 1ns / 1ps

//  Design      : 2nd Convolution Layer for CNN MNIST dataset
//                Convolution Sum Calculation - 1st Channel

 module conv2_calc_1(
 input clk,
    input rst_n,
    input valid_out_buf,
 input signed [11:0] data_out1_0, data_out1_1, data_out1_2, data_out1_3, data_out1_4,
    data_out1_5, data_out1_6, data_out1_7, data_out1_8, data_out1_9,
    data_out1_10, data_out1_11, data_out1_12, data_out1_13, data_out1_
14,
        data_out1_15, data_out1_16, data_out1_17, data_out1_18, data_out1_19,
        data_out1_20, data_out1_21, data_out1_22, data_out1_23, data_out1_24,

        data_out2_0, data_out2_1, data_out2_2, data_out2_3, data_out2_4,
        data_out2_5, data_out2_6, data_out2_7, data_out2_8, data_out2_9,
        data_out2_10, data_out2_11, data_out2_12, data_out2_13, data_out2_14,
        data_out2_15, data_out2_16, data_out2_17, data_out2_18, data_out2_19,
        data_out2_20, data_out2_21, data_out2_22, data_out2_23, data_out2_24,

        data_out3_0, data_out3_1, data_out3_2, data_out3_3, data_out3_4,
        data_out3_5, data_out3_6, data_out3_7, data_out3_8, data_out3_9,
        data_out3_10, data_out3_11, data_out3_12, data_out3_13, data_out3_14,
        data_out3_15, data_out3_16, data_out3_17, data_out3_18, data_out3_19,
        data_out3_20, data_out3_21, data_out3_22, data_out3_23, data_out3_24,

   output [13:0] conv_out_calc,
      output reg valid_out_calc
       );

       wire signed [19:0] calc_out, calc_out_1, calc_out_2, calc_out_3;

       wire signed [7:0] weight_1 [0:24];
       wire signed [7:0] weight_2 [0:24];
       wire signed [7:0] weight_3 [0:24];

  assign weight_1[0] = 8'h01; //01;
  assign weight_1[1] = 8'h03; //03
  assign weight_1[2] = 8'h01; //01;
  assign weight_1[3] = 8'h0c;
  assign weight_1[4] = 8'hfd;
  assign weight_1[5] = 8'hf9;
  assign weight_1[6] = 8'hf9;
  assign weight_1[7] = 8'h0a;
  assign weight_1[8] = 8'hf2;
  assign weight_1[9] = 8'heb;
```

```verilog
assign weight_1[10] = 8'hea;
assign weight_1[11] = 8'he9;
assign weight_1[12] = 8'hee;
assign weight_1[13] = 8'he1;
assign weight_1[14] = 8'hf8;
assign weight_1[15] = 8'he7;
assign weight_1[16] = 8'hd6;
assign weight_1[17] = 8'hd7;
assign weight_1[18] = 8'hfa;
assign weight_1[19] = 8'hf8;
assign weight_1[20] = 8'he2;
assign weight_1[21] = 8'hd8;
assign weight_1[22] = 8'he4;
assign weight_1[23] = 8'hfd;
assign weight_1[24] = 8'h18;

assign weight_2[0] = 8'hfe;
assign weight_2[1] = 8'hea;
assign weight_2[2] = 8'h05;
assign weight_2[3] = 8'hfa;
assign weight_2[4] = 8'he4;
assign weight_2[5] = 8'h00; //00
assign weight_2[6] = 8'hed;
assign weight_2[7] = 8'h1f;
assign weight_2[8] = 8'h47;
assign weight_2[9] = 8'h0c;
assign weight_2[10] = 8'hc8;
assign weight_2[11] = 8'h1e;
assign weight_2[12] = 8'h4a;
assign weight_2[13] = 8'h21;
assign weight_2[14] = 8'hd6;
assign weight_2[15] = 8'h0f;
assign weight_2[16] = 8'h32;
assign weight_2[17] = 8'h11;
assign weight_2[18] = 8'hd6;
assign weight_2[19] = 8'hf5;
assign weight_2[20] = 8'h2a;
assign weight_2[21] = 8'h16;
assign weight_2[22] = 8'he6;
assign weight_2[23] = 8'hf8;
assign weight_2[24] = 8'hf4;

assign weight_3[0] = 8'hf0;
assign weight_3[1] = 8'heb;
assign weight_3[2] = 8'h0c;
assign weight_3[3] = 8'heb;
assign weight_3[4] = 8'hf2;
assign weight_3[5] = 8'hfc;
assign weight_3[6] = 8'h08;
assign weight_3[7] = 8'h14;
assign weight_3[8] = 8'he9;
assign weight_3[9] = 8'hf5;
assign weight_3[10] = 8'h10;
assign weight_3[11] = 8'h06;
assign weight_3[12] = 8'hdc;
assign weight_3[13] = 8'he3;
assign weight_3[14] = 8'he5;
assign weight_3[15] = 8'h03; //03;
assign weight_3[16] = 8'he3;
assign weight_3[17] = 8'hf3;
assign weight_3[18] = 8'hfa;
assign weight_3[19] = 8'hf3;
assign weight_3[20] = 8'he7;
assign weight_3[21] = 8'hef;
assign weight_3[22] = 8'hf6;
assign weight_3[23] = 8'h05;
assign weight_3[24] = 8'hf5;


// integer is_fill = 0;

//always @(*) begin
//if(is_fill == 0) begin
////    $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/conv2_weight_11.txt", weight_1);
////    $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/conv2_weight_12.txt", weight_2);
////    $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/conv2_weight_13.txt", weight_3);
//    weight_1[0] <= 8'h01;

//weight_1[1] <= 8'h03;
//..
//weight_1[24] <= 8'h18;

//weight_2[0] <= 8'hfe;
//..
//weight_2[24] <= 8'hf4;
```

```

//weight_3[0] <= 8'hf0;
//..
//weight_3[24] <= 8'hf5;

//is_fill <= 1;
//        end
//        end

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] +
     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] +
     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] +
     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] +
     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];

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] +
     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] +
     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] +
     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] +
     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];

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] +
     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] +
     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] +
     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] +
     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];

assign calc_out = calc_out_1 + calc_out_2 + calc_out_3;


//reg signed [19:0] calc_out, calc_out_1, calc_out_2, calc_out_3;

assign conv_out_calc = calc_out[19:6]; // 14bit

always @ (posedge clk) begin
 if(~rst_n) begin
  valid_out_calc <= 0;
    //conv_out_calc <= 0;
 end
 else begin
  // Toggling Valid Output Signal
  if(valid_out_buf == 1) begin
   if(valid_out_calc == 1)
    valid_out_calc <= 0;
   else
    valid_out_calc <= 1;
  end
 end
end

//integer state;
//always @ (posedge clk) begin
// if(~rst_n) begin
//     valid_out_calc <= 0;
//     state <= 0;
//     //conv_out_calc <= 0;
//     end
// else begin
//  // Toggling Valid Output Signal
//     if(valid_out_buf == 1) begin
//         if(valid_out_calc == 1) begin
//            valid_out_calc <= 0;
//            state <= 0;
//            end
//        else if (state == 0) begin
//            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] +
//                               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] +
//                               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] +
//                               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] +
//                               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];
//            state <= 1;
//            end
//        else if (state == 1) begin
//               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] +
//                               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] +
//                               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] +
//                               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] +
//                               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];

//               state <= 2;
//               end
//           else if (state == 2) begin
//               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] +
```

```verilog
//                              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] +
//                              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] +
//                              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] +
//                              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];
//                calc_out <= calc_out_1 + calc_out_2 + calc_out_3;
//                valid_out_calc <= 1;
//                end
//  end
// end
//end

endmodule
```

```verilog
m`timescale 1ns / 1ps

 // Design     : Fully Connected Layer for CNN

 module fully_connected (//#(parameter INPUT_NUM = 48, OUTPUT_NUM = 10, DATA_BITS = 8) (
   input clk,
   input rst_n,
   input valid_in,
   input signed [11:0] data_in_1, data_in_2, data_in_3,
   output [11:0] data_out,
   output reg valid_out_fc
//   output reg delay
 );

parameter INPUT_NUM = 48, OUTPUT_NUM = 10, DATA_BITS = 8;

 localparam INPUT_WIDTH = 16;
 localparam INPUT_NUM_DATA_BITS = 5;

 reg state;
 reg [INPUT_WIDTH - 1:0] buf_idx;
 reg [3:0] out_idx;
```

```
    reg signed [13:0] buffer [0:INPUT_NUM - 1];

    wire signed [DATA_BITS - 1:0] weight [0:INPUT_NUM * OUTPUT_NUM - 1];
    wire signed [DATA_BITS - 1:0] bias [0:OUTPUT_NUM - 1];

    wire signed [19:0] calc_out;
    wire signed [13:0] data1, data2, data3;

   assign weight[0] = 8'hfe;
   //......
   assign weight[479] = 8'h09;

   assign bias[0] = 8'hf5;
   assign bias[1] = 8'h1d;
   assign bias[2] = 8'h09;
   assign bias[3] = 8'he0;
   assign bias[4] = 8'h00;
   assign bias[5] = 8'hfb;
   assign bias[6] = 8'h07;
   assign bias[7] = 8'h05;
   assign bias[8] = 8'hf9;
   assign bias[9] = 8'hf4;


   //integer is_fill = 0;

   //always @(*) begin
   //if(is_fill == 0) begin
   ////   $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/fc_weight.txt", weight);
   ////   $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/fc_bias.txt", bias);

   //weight[0] <= 8'hfe;
   //...
   //weight[479] <= 8'h09;

   //bias[0] <= 8'hf5;
   //...
   //bias[9] <= 8'hf4;

   //is_fill <= 1;

   // end
```

```
  // end

  assign data1 = (data_in_1[11] == 1) ? {2'b11, data_in_1} : {2'b00, data_in_1};
  assign data2 = (data_in_2[11] == 1) ? {2'b11, data_in_2} : {2'b00, data_in_2};
  assign data3 = (data_in_3[11] == 1) ? {2'b11, data_in_3} : {2'b00, data_in_3};

  always @(posedge clk) begin
    if(~rst_n) begin
      valid_out_fc <= 0;
      buf_idx <= 0;
      out_idx <= 0;
      state <= 0;
    end

    if(valid_out_fc == 1) begin
      valid_out_fc <= 0;
    end

    if(valid_in == 1) begin
      // Wait until 48 input data filled in buffer
      if(!state) begin
        buffer[buf_idx] <= data1;
        buffer[INPUT_WIDTH + buf_idx] <= data2;
        buffer[INPUT_WIDTH * 2 + buf_idx] <= data3;
        buf_idx <= buf_idx + 1'b1;
        if(buf_idx == INPUT_WIDTH - 1) begin
          buf_idx <= 0;
          state <= 1;
          valid_out_fc <= 1;
        end
      end else begin // valid state
        out_idx <= out_idx + 1'b1;
        if(out_idx == OUTPUT_NUM - 1) begin
          out_idx <= 0;
        end
        valid_out_fc <= 1;
      end
    end
  end

  assign calc_out = weight[out_idx * INPUT_NUM] * buffer[0] + weight[ou
```

```
t_idx * INPUT_NUM + 1] * buffer[1] +
        weight[out_idx * INPUT_NUM + 2] * buffer[2] + weight[out_idx * INPUT_NUM + 3] * buffer[3] +
        weight[out_idx * INPUT_NUM + 4] * buffer[4] + weight[out_idx * INPUT_NUM + 5] * buffer[5] +
        weight[out_idx * INPUT_NUM + 6] * buffer[6] + weight[out_idx * INPUT_NUM + 7] * buffer[7] +
        weight[out_idx * INPUT_NUM + 8] * buffer[8] + weight[out_idx * INPUT_NUM + 9] * buffer[9] +
        weight[out_idx * INPUT_NUM + 10] * buffer[10] + weight[out_idx * INPUT_NUM + 11] * buffer[11] +
        weight[out_idx * INPUT_NUM + 12] * buffer[12] + weight[out_idx * INPUT_NUM + 13] * buffer[13] +
        weight[out_idx * INPUT_NUM + 14] * buffer[14] + weight[out_idx * INPUT_NUM + 15] * buffer[15] +
        weight[out_idx * INPUT_NUM + 16] * buffer[16] + weight[out_idx * INPUT_NUM + 17] * buffer[17] +
        weight[out_idx * INPUT_NUM + 18] * buffer[18] + weight[out_idx * INPUT_NUM + 19] * buffer[19] +
        weight[out_idx * INPUT_NUM + 20] * buffer[20] + weight[out_idx * INPUT_NUM + 21] * buffer[21] +
        weight[out_idx * INPUT_NUM + 22] * buffer[22] + weight[out_idx * INPUT_NUM + 23] * buffer[23] +
        weight[out_idx * INPUT_NUM + 24] * buffer[24] + weight[out_idx * INPUT_NUM + 25] * buffer[25] +
        weight[out_idx * INPUT_NUM + 26] * buffer[26] + weight[out_idx * INPUT_NUM + 27] * buffer[27] +
        weight[out_idx * INPUT_NUM + 28] * buffer[28] + weight[out_idx * INPUT_NUM + 29] * buffer[29] +
        weight[out_idx * INPUT_NUM + 30] * buffer[30] + weight[out_idx * INPUT_NUM + 31] * buffer[31] +
        weight[out_idx * INPUT_NUM + 32] * buffer[32] + weight[out_idx * INPUT_NUM + 33] * buffer[33] +
        weight[out_idx * INPUT_NUM + 34] * buffer[34] + weight[out_idx * INPUT_NUM + 35] * buffer[35] +
        weight[out_idx * INPUT_NUM + 36] * buffer[36] + weight[out_idx * INPUT_NUM + 37] * buffer[37] +
        weight[out_idx * INPUT_NUM + 38] * buffer[38] + weight[out_idx * INPUT_NUM + 39] * buffer[39] +
        weight[out_idx * INPUT_NUM + 40] * buffer[40] + weight[out_idx * INPUT_NUM + 41] * buffer[41] +
        weight[out_idx * INPUT_NUM + 42] * buffer[42] + weight[out_idx * INPUT_NUM + 43] * buffer[43] +
        weight[out_idx * INPUT_NUM + 44] * buffer[44] + weight[out_idx *
```

```
INPUT_NUM + 45] * buffer[45] +
        weight[out_idx * INPUT_NUM + 46] * buffer[46] + weight[out_idx *
INPUT_NUM + 47] * buffer[47] +
        bias[out_idx];

assign data_out = calc_out[18:7];

endmodule

//    integer step;
//    reg [19:0] result1[0:3];
//    reg [19:0] result2;

//    always @(posedge clk) begin
//        if (~rst_n) begin
//            valid_out_fc <= 0;
//            buf_idx <= 0;
//            out_idx <= 0;
//            state <= 0;
//            step <= 0;
//            delay <= 0;
//            end

//        if (valid_out_fc == 1) begin
//            valid_out_fc <= 0;
//            step <= 0;
//            end

//        if (valid_in == 1) begin
//            // Wait until 48 input data filled in buffer
//            if (!state) begin
//                buffer[buf_idx] <= data1;
//                buffer[INPUT_WIDTH + buf_idx] <= data2;
//                buffer[INPUT_WIDTH * 2 + buf_idx] <= data3;
//                buf_idx <= buf_idx + 1'b1;
//                if(buf_idx == INPUT_WIDTH - 1) begin
//                    buf_idx <= 0;
//                    state <= 1;
//                    step <= 0;
//                    delay <= 1;
////                    valid_out_fc <= 1;
//                    end
//                end
//            else begin // valid state

//                result1[step] <= weight[out_idx * INPUT_NUM + step * 12] * buffer[0 + step * 12] + weight[out_idx * INPUT_NUM + 1 + step * 12] * buffer[1 + step * 12]
//                    + weight[out_idx * INPUT_NUM + 2 + step * 12] * buffer[2 + step * 12] + weight[out_idx * INPUT_NUM + 3 + step * 12] * buffer[3 + step * 12]
//                    + weight[out_idx * INPUT_NUM + 4 + step * 12] * buffer[4 + step * 12] + weight[out_idx * INPUT_NUM + 5 + step * 12] * buffer[5 + step * 12]
//                    + weight[out_idx * INPUT_NUM + 6 + step * 12] * buffer[6 + step * 12] + weight[out_idx * INPUT_NUM + 7 + step * 12] * buffer[7 + step * 12]
//                    + weight[out_idx * INPUT_NUM + 8 + step * 12] * buffer[8 + step * 12] + weight[out_idx * INPUT_NUM + 9 + step * 12] * buffer[9 + step * 12]
//                    + weight[out_idx * INPUT_NUM + 10 + step * 12] * buffer[10 + step * 12] + weight[out_idx * INPUT_NUM + 11 + step * 12] * buffer[11 + step * 12];

//                if (step == 3) begin
//                    result2 <= result1[0] + result1[1] + result1[2] + result1[3] + bias[out_idx];
//                    step <= 0;
//                    delay <= 0;
//                    out_idx <= out_idx + 1'b1;
//                    if (out_idx == OUTPUT_NUM - 1) begin
//                        out_idx <= 0;
//                        end
//                    valid_out_fc <= 1;
//                    end
//                else
//                    step <= step + 1;

//                end
//            end
//        end

//  assign data_out = result2[18:7];

//  endmodule
```

```verilog
`timescale 1ns / 1ps

// Description:  Final Comparator for decision

module comparator (
    input clk,
    input rst_n,
    input valid_in,
    input [11:0] data_in,
    output reg [3:0] decision,
    output reg valid_out
);

    reg signed [11:0] buffer [0:9];
    reg signed [11:0] max;
    reg signed [11:0] cmp1_0, cmp1_1, cmp1_2, cmp1_3, cmp1_4,
                    cmp2_0, cmp2_1, cmp2_2, cmp3_0, cmp3_1;
    reg [3:0] buf_idx;
    reg [11:0] delay_cnt;
    reg state;

    always @(posedge clk) begin
        if (~rst_n) begin
            valid_out <= 0;
            buf_idx <= 0;
            delay_cnt <= 0;
            state <= 0;
            decision <= 4'd0;
            cmp1_0 <= 12'd0;
            cmp1_1 <= 12'd0;
            cmp1_2 <= 12'd0;
            cmp1_3 <= 12'd0;
            cmp1_4 <= 12'd0;
            cmp2_0 <= 12'd0;
            cmp2_1 <= 12'd0;
            cmp2_2 <= 12'd0;
            cmp3_0 <= 12'd0;
            cmp3_1 <= 12'd0;
            max <= 12'd0;
            buffer[0] <= 12'd0;
            buffer[1] <= 12'd0;
            buffer[2] <= 12'd0;
            buffer[3] <= 12'd0;
            buffer[4] <= 12'd0;
            buffer[5] <= 12'd0;
            buffer[6] <= 12'd0;
            buffer[7] <= 12'd0;
            buffer[8] <= 12'd0;
            buffer[9] <= 12'd0;
            end
        else begin
            if (valid_in == 1) begin // read data input
                buffer[buf_idx] <= data_in;
                buf_idx <= buf_idx + 1'b1;
                if (buf_idx == 9)
                    state <= 1;
                end
            else begin
                if(state == 1) begin
                    delay_cnt <= delay_cnt + 1'b1;
                    if(delay_cnt == 12'd5)
                        valid_out <= 1;
                    else
                        valid_out <= 0;

                    // Decision Process
                    cmp1_0 <= (buffer[0] >= buffer[1]) ? buffer[0] : buffer[1];
                    cmp1_1 <= (buffer[2] >= buffer[3]) ? buffer[2] : buffer[3];
                    cmp1_2 <= (buffer[4] >= buffer[5]) ? buffer[4] : buffer[5];
                    cmp1_3 <= (buffer[6] >= buffer[7]) ? buffer[6] : buffer[7];
                    cmp1_4 <= (buffer[8] >= buffer[9]) ? buffer[8] : buffer[9];

                    cmp2_0 <= (cmp1_0 >= cmp1_1) ? cmp1_0 : cmp1_1;
                    cmp2_1 <= (cmp1_2 >= cmp1_3) ? cmp1_2 : cmp1_3;
                    cmp2_2 <= cmp1_4;

                    cmp3_0 <= (cmp2_0 >= cmp2_1) ? cmp2_0 : cmp2_1;
                    cmp3_1 <= cmp2_2;

                    //if(delay_cnt == 12'd3)
                    max <= (cmp3_0 >= cmp3_1) ? cmp3_0 : cmp3_1;

                    //if(delay_cnt == 12'd4) begin
                    if(max == buffer[0])
                        decision <= 4'd0;
                    else if(max == buffer[1])
                        decision <= 4'd1;
                    else if(max == buffer[2])
                        decision <= 4'd2;
                    else if(max == buffer[3])
                        decision <= 4'd3;
                    else if(max == buffer[4])
                        decision <= 4'd4;
                    else if(max == buffer[5])
                        decision <= 4'd5;
                    else if(max == buffer[6])
                        decision <= 4'd6;
                    else if(max == buffer[7])
                        decision <= 4'd7;
                    else if(max == buffer[8])
                        decision <= 4'd8;
                    else if(max == buffer[9])
                        decision <= 4'd9;
      //end
                    end
                end
            end
        end

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：

```
`timescale 1ns / 1ps

module display7(
    input [3:0] iData,
    output reg [6:0] oData
    );

     always@(iData)
        case(iData)
            4'b0000: oData = 7'b1000000;
            4'b0001: oData = 7'b1111001;
            4'b0010: oData = 7'b0100100;
            4'b0011: oData = 7'b0110000;
            4'b0100: oData = 7'b0011001;
            4'b0101: oData = 7'b0010010;
            4'b0110: oData = 7'b0000010;
            4'b0111: oData = 7'b1111000;
            4'b1000: oData = 7'b0000000;
            4'b1001: oData = 7'b0010000;
            default: oData = 7'b1111111;
        endcase

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：

```
`timescale 1ns / 1ps

module mp3_top(
    input clk,
    input rst,
    input valid_in,
    input [3:0] decision,
    input DREQ,
    output xRSET,
    output xCS,
    output xDCS,
    output MOSI,
    output SCLK
    );

    reg is_play;

    always @(posedge clk or posedge rst) begin
        if(rst)
            is_play <= 0;
        else if(valid_in)
            is_play <= 1;
        end

    mp3_driver #(.MUSIC_SIZE(25112)) uut_mp3 (
        .mp3_clk(clk),
```

```verilog
        .rst(rst | ~is_play),
        .decision(decision),
        .DREQ(DREQ),
        .xRSET(xRSET),
        .xCS(xCS),
        .xDCS(xDCS),
        .MOSI(MOSI),
        .SCLK(SCLK)
        );

endmodule
```

```verilog
`timescale 1ns / 1ps

module mp3_driver #(parameter MUSIC_SIZE = 25112) (
    input mp3_clk,
    input rst,
    input [3:0] decision,
    input DREQ,
    output reg xRSET = 0,
    output reg xCS = 1,
    output reg xDCS = 1,
    output reg MOSI,
    output reg SCLK = 0
    );

reg [1:0] music_over;

reg [14:0] mp3_addr;
wire [31:0] mp3_data_all[0:10];

localparam  RESET = 0,
            CMD_CONTROL = 1,
            CMD_SEND = 2,
            DATA_CONTROL = 3,
            DATA_SEND = 4;

reg [2:0] state = RESET;

reg [31:0] mp3_data, mp3_data0;

localparam  cmd_mode = 32'h02000804,
            cmd_vol = 32'h020B1010,
            cmd_bass = 32'h02020055,
            cmd_clock = 32'h02039800;

reg [127:0] cmd = {cmd_mode, cmd_vol, cmd_bass, cmd_clock};
reg [2:0] cmd_id;
reg [5:0] cnt;

//reg music_over=0;

localparam CMD_NUM = 4;

always @(posedge mp3_clk or posedge rst) begin
    if (rst == 1) begin
        state<=RESET;
        music_over <= 2'b00;
        end
    else if (music_over == 2'b00 || music_over == 2'b01) begin
        if (state == RESET) begin
            cmd <= {cmd_mode, cmd_vol, cmd_bass, cmd_clock};
            mp3_addr <= 0;
            cmd_id <= 0;
            cnt <= 0;
            xCS <= 1;
            xDCS <= 1;
            xRSET <= 0;
            SCLK <= 0;

            if (rst == 0)
                state <= CMD_CONTROL;
            else
                state <= RESET;
            end

        else if (state == CMD_CONTROL) begin
            xRSET <= 1;
            if (cmd_id < CMD_NUM && DREQ) begin
                cmd_id <= cmd_id + 1;
                xCS <= 0;
                MOSI <= cmd[127];
                cmd <= {cmd[126:0], cmd[127]};
                cnt <= 1;
                state <= CMD_SEND;
                end
            else begin
                cmd_id <= 0;
                state <= DATA_CONTROL;
                end
            end

        else if (state == CMD_SEND) begin
            if (DREQ) begin
                if(SCLK) begin
                    if(cnt < 32) begin
                        cnt <= cnt + 1;
                        MOSI <= cmd[127];
                        cmd <= {cmd[126:0], cmd[127]};
                        end
                    else begin
                        xCS <= 1;
                        cnt <= 0;
                        state <= CMD_CONTROL;
                        end
                    end
                SCLK <= ~SCLK;
                end
            end

        else if(state == DATA_CONTROL) begin
            if(mp3_addr >= MUSIC_SIZE) begin
                state <= RESET;
                music_over <= music_over + 1'b1;
                //DCS<=1;
                end
            else if (DREQ) begin
                //music_over<=0;
                xDCS <= 0;
                SCLK <= 0;

                if (music_over == 2'b00)
                    mp3_data0 <= mp3_data_all[10];
                else if (music_over == 2'b01)
                    mp3_data0 <= mp3_data_all[decision];

                MOSI <= mp3_data0[31];
                mp3_data <= {mp3_data0[30:0], mp3_data0[31]};
                cnt <= 1;
                state <= DATA_SEND;
                end
            end

        else if(state == DATA_SEND) begin
            if (DREQ) begin
                if (SCLK) begin
                    if (cnt < 32) begin
                        MOSI <= mp3_data[31];
                        cnt <= cnt + 1;
                        mp3_data <= {mp3_data[30:0], mp3_data[31]};
                        end
                    else begin
                        xDCS <= 1;
                        cnt <= 0;
                        mp3_addr <= mp3_addr + 1;
                        state <= DATA_CONTROL;
                        end
                    end
                SCLK <= ~SCLK;
                end
            end

        else
            state <= RESET;
        end
    end

    blk_mem_gen_0 mic_tips (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr), .douta(mp3_data_all[10]));
    blk_mem_gen_1 num_0 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[0]));
    blk_mem_gen_2 num_1 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[1]));
    blk_mem_gen_3 num_2 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[2]));
    blk_mem_gen_4 num_3 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[3]));
    blk_mem_gen_5 num_4 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[4]));
    blk_mem_gen_6 num_5 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[5]));
    blk_mem_gen_7 num_6 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[6]));
    blk_mem_gen_8 num_7 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[7]));
    blk_mem_gen_9 num_8 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[1
2:0]), .douta(mp3_data_all[8]));
    blk_mem_gen_10 num_9 (.clka(mp3_clk), .ena(~rst), .addra(mp3_addr[12:0]), .douta(mp3_data_all[9]));

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

```
`timescale 1ns / 1ps

module cnn_tb();

reg clk, rst_n;
reg [7:0] pixels [0:783];
reg [9:0] img_idx;
reg [7:0] data_in;

wire signed [11:0] conv_out_1, conv_out_2, conv_out_3;
wire signed [11:0] conv2_out_1, conv2_out_2, conv2_out_3;
wire signed [11:0] max_value_1, max_value_2, max_value_3;
wire signed [11:0] max2_value_1, max2_value_2, max2_value_3;
wire signed [11:0] fc_out_data;
wire [3:0] decision;

wire valid_out_1, valid_out_2, valid_out_3, valid_out_4, valid_out_5, valid_out_6;

// Module Instantiation
conv1_layer conv1_layer(
  .clk(clk),
  .rst_n(rst_n),
  .data_in(data_in),
  .conv_out_1(conv_out_1),
  .conv_out_2(conv_out_2),
  .conv_out_3(conv_out_3),
  .valid_out_conv(valid_out_1)
);

maxpool_relu #(.CONV_BIT(12), .HALF_WIDTH(12), .HALF_HEIGHT(12), .HALF_WIDTH_BIT(4))
maxpool_relu_1(
  .clk(clk),
```

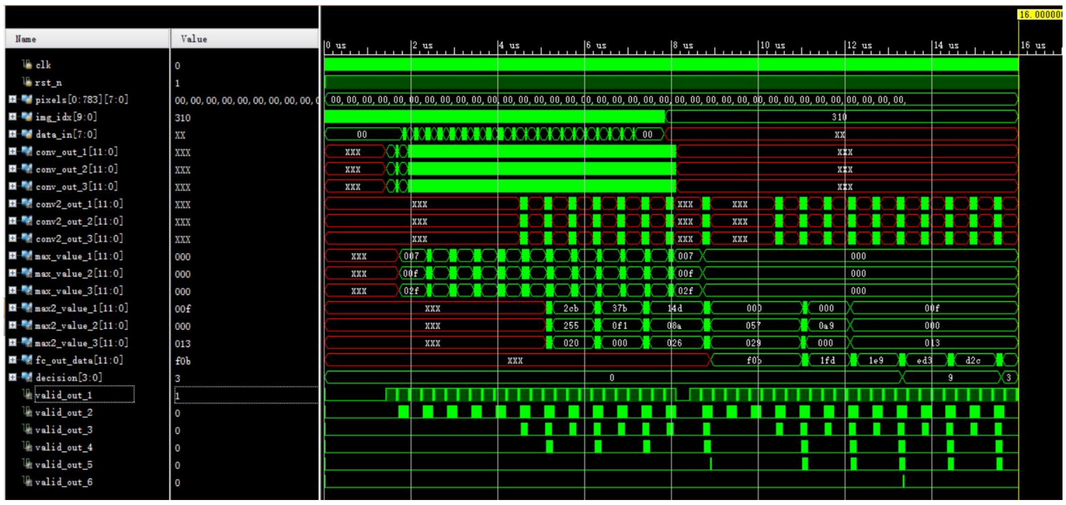
```verilog
    .rst_n(rst_n),
    .valid_in(valid_out_1),
    .conv_out_1(conv_out_1),
    .conv_out_2(conv_out_2),
    .conv_out_3(conv_out_3),
    .max_value_1(max_value_1),
    .max_value_2(max_value_2),
    .max_value_3(max_value_3),
    .valid_out_relu(valid_out_2)
  );

  conv2_layer conv2_layer(
    .clk(clk),
    .rst_n(rst_n),
    .valid_in(valid_out_2),
    .max_value_1(max_value_1),
    .max_value_2(max_value_2),
    .max_value_3(max_value_3),
    .conv2_out_1(conv2_out_1),
    .conv2_out_2(conv2_out_2),
    .conv2_out_3(conv2_out_3),
    .valid_out_conv2(valid_out_3)
  );

  maxpool_relu #(.CONV_BIT(12), .HALF_WIDTH(4), .HALF_HEIGHT(4), .HALF_WIDTH_BIT(
3))
  maxpool_relu_2(
    .clk(clk),
    .rst_n(rst_n),
    .valid_in(valid_out_3),
    .conv_out_1(conv2_out_1),
    .conv_out_2(conv2_out_2),
    .conv_out_3(conv2_out_3),
    .max_value_1(max2_value_1),
    .max_value_2(max2_value_2),
    .max_value_3(max2_value_3),
    .valid_out_relu(valid_out_4)
  );

  fully_connected #(.INPUT_NUM(48), .OUTPUT_NUM(10), .DATA_BITS(8))
  fully_connected(
    .clk(clk),
    .rst_n(rst_n),
    .valid_in(valid_out_4),
    .data_in_1(max2_value_1),
    .data_in_2(max2_value_2),
    .data_in_3(max2_value_3),
    .data_out(fc_out_data),
    .valid_out_fc(valid_out_5)
  );

  comparator comparator(
    .clk(clk),
    .rst_n(rst_n),
    .valid_in(valid_out_5),
    .data_in(fc_out_data),
    .decision(decision),
    .valid_out(valid_out_6)
  );

  // Clock generation
  always #5 clk = ~clk;

  // Read image text file
  initial begin
    $readmemh("C:/Users/DELL/Desktop/cnn/cnn.srcs/sources_1/new/data/9_0.txt", pixels);
    clk <= 1'b0;
    rst_n <= 1'b1;
    #3
    rst_n <= 1'b0;
    #3
    rst_n <= 1'b1;
  end

  always @(posedge clk) begin
    if(~rst_n) begin
      img_idx <= 0;
    end else begin
      data_in <= pixels[img_idx];
      if(img_idx < 10'd784) begin
        img_idx <= img_idx + 1'b1;
      end
    end
  end

  endmodule
```

六、实验结果

1. 待输入手写数字灰度图像素数据：

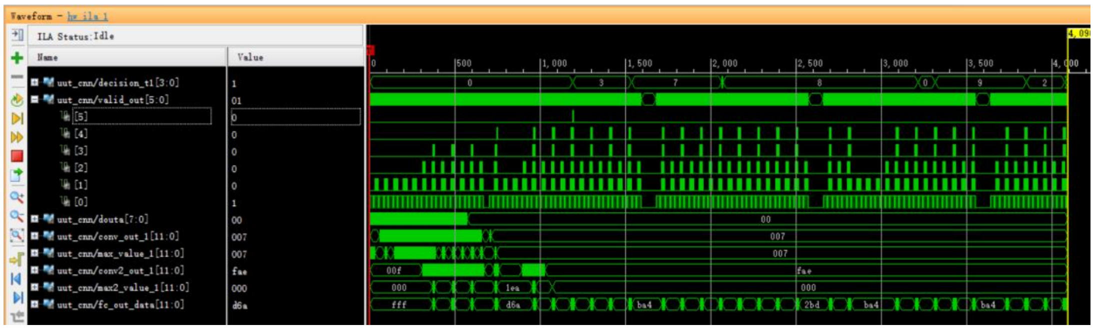
```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 3e bf ff fd 89 0f 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 77 ec fc cf ec fc a0 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 07 8e ea e3 82 00 2c ca e3 11 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 a0 fc e3 32 00 00 00 5a fc 16 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 6a fd fc 4f 00 00 00 00 46 fc 4b 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 74 ff fd 45 00 00 00 00 51 fd be 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 2a e8 fc 66 09 00 2f 2f e4 fc 40 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 37 fc fc c4 b8 fd dd fc e3 10 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 05 43 a0 c2 a8 d3 fc fc a0 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 09 02 56 fc dd 23 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff fd b8 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 16 fd f4 62 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 92 fd ae 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 4c f8 fd 5c 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 90 fc f3 32 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 31 e9 fd 35 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 09 c5 fc e7 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 82 fc fc 6a 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 a1 fc b0 04 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 a1 fc 38 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

2. CNN 模块仿真波形图



Name	Value
clk	0
rst_n	1
pixels[0:783][7:0]	00,00,00,00,00,00,00,00,00,0
img_idx[9:0]	310
data_in[7:0]	XX
conv_out_1[11:0]	XXX
conv_out_2[11:0]	XXX
conv_out_3[11:0]	XXX
conv2_out_1[11:0]	XXX
conv2_out_2[11:0]	XXX
conv2_out_3[11:0]	XXX
max_value_1[11:0]	000
max_value_2[11:0]	000
max_value_3[11:0]	000
max2_value_1[11:0]	00f
max2_value_2[11:0]	000
max2_value_3[11:0]	013
fc_out_data[11:0]	f0b
decision[3:0]	3
valid_out_1	1
valid_out_2	0
valid_out_3	0
valid_out_4	0
valid_out_5	0
valid_out_6	0

3. CNN 模块 ILA 下板调试结果



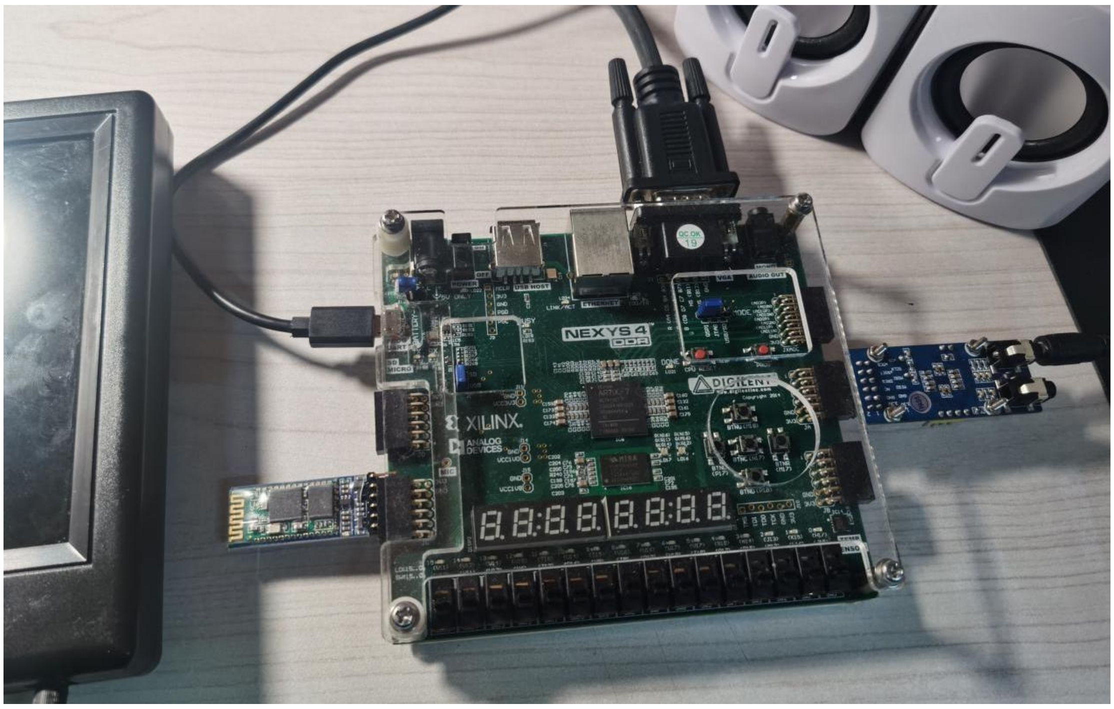
Waveform - hw_ila_1

ILA Status:Idle

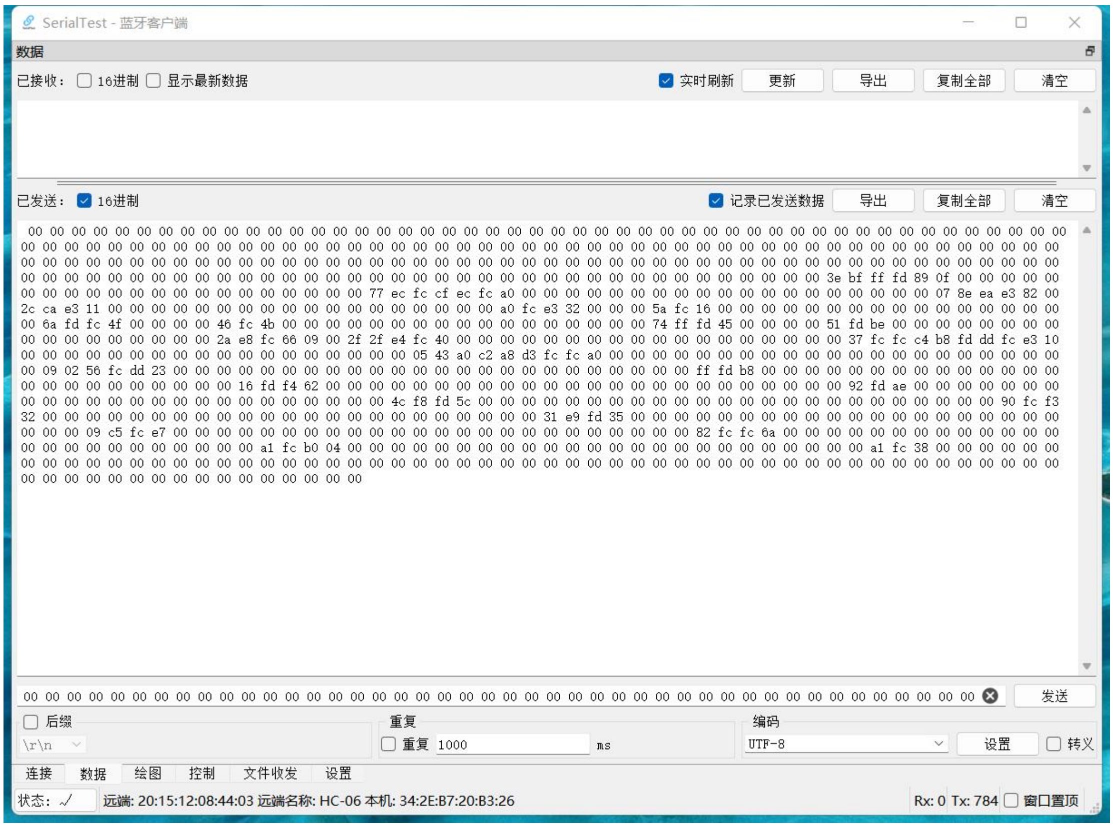
Name	Value
uut_cnn/decision_t1[3:0]	1
uut_cnn/valid_out[5:0]	01
[5]	0
[4]	0
[3]	0
[2]	0
[1]	0
[0]	1
uut_cnn/douta[7:0]	00
uut_cnn/conv_out_1[11:0]	007
uut_cnn/max_value_1[11:0]	007
uut_cnn/conv2_out_1[11:0]	fae
uut_cnn/max2_value_1[11:0]	000
uut_cnn/fc_out_data[11:0]	d6a

4. 下板实验

(1) 设备连接状态



(2) 蓝牙串口工具发送

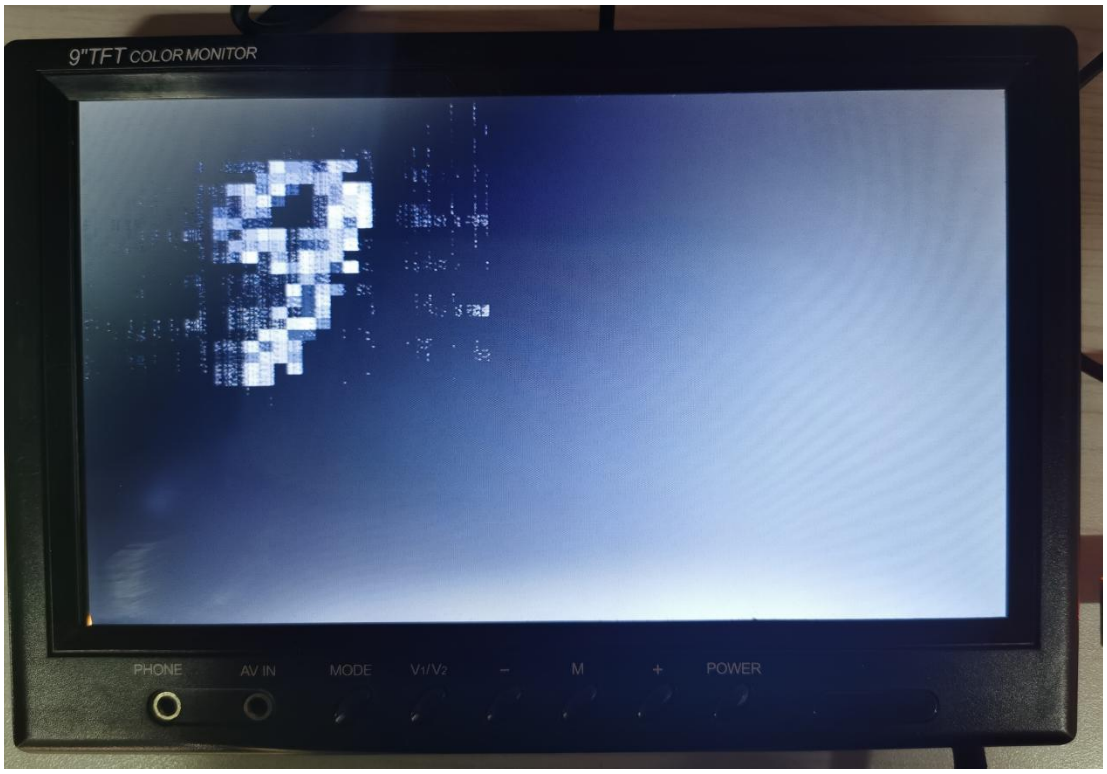


```
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 3e bf ff fd 89 0f 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 77 ec fc cf ec fc a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 07 8e ea e3 82 00
2c ca e3 11 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 a0 fc e3 32 00 00 00 5a fc 16 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 6a fd fc 4f 00 00 00 00 46 fc 4b 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 74 ff fd 45 00 00 00 00 51 fd be 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 2a e8 fc 66 09 00 2f 2f e4 fc 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 37 fc fc c4 b8 fd dd fc e3 10
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 05 43 a0 c2 a8 d3 fc fc a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 09 02 56 fc dd 23 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff fd b8 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 16 fd f4 62 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 92 fd ae 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 4c f8 fd 5c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 90 fc f3
32 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 31 e9 fd 35 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 09 c5 fc e7 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 82 fc fc 6a 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 a1 fc b0 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 a1 fc 38 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

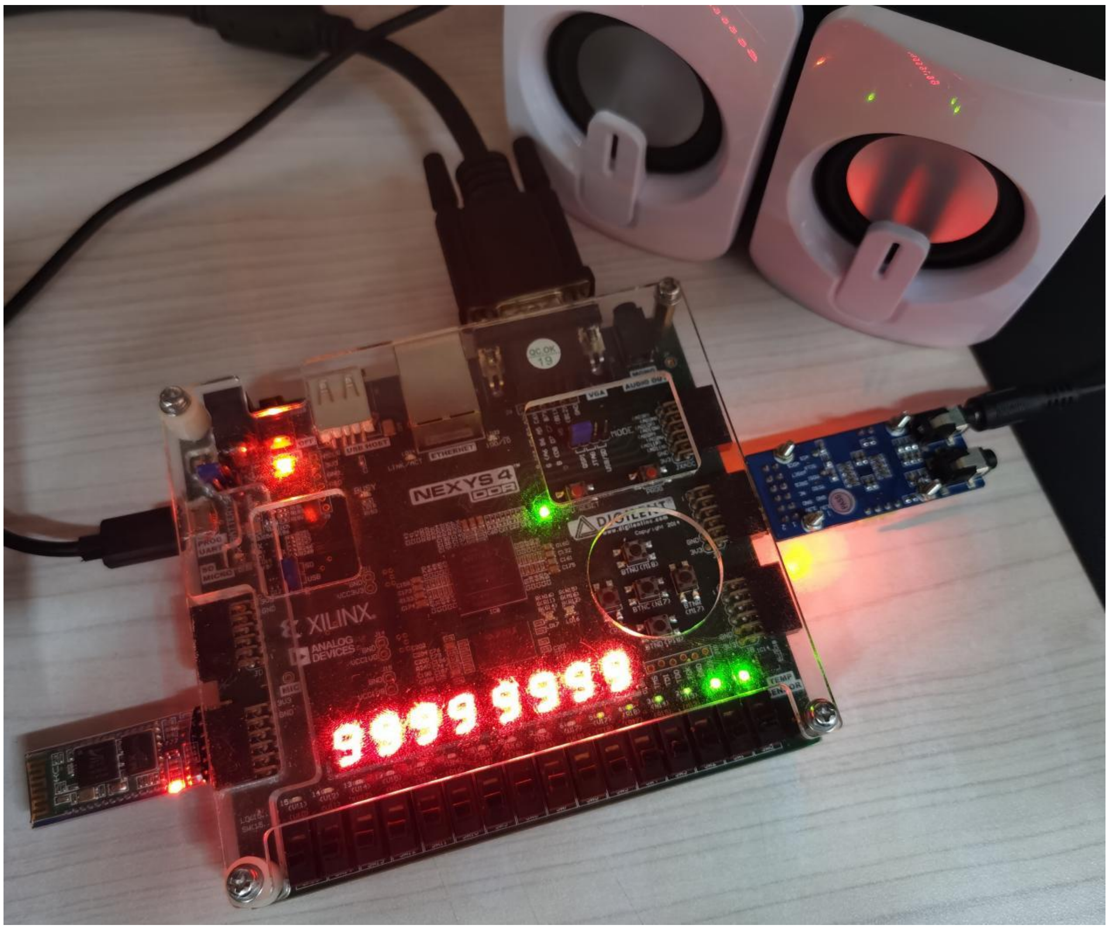
```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

状态: ✓ 远端: 20:15:12:08:44:03 远端名称: HC-06 本机: 34:2E:B7:20:B3:26 Rx: 0 Tx: 784 窗口置顶

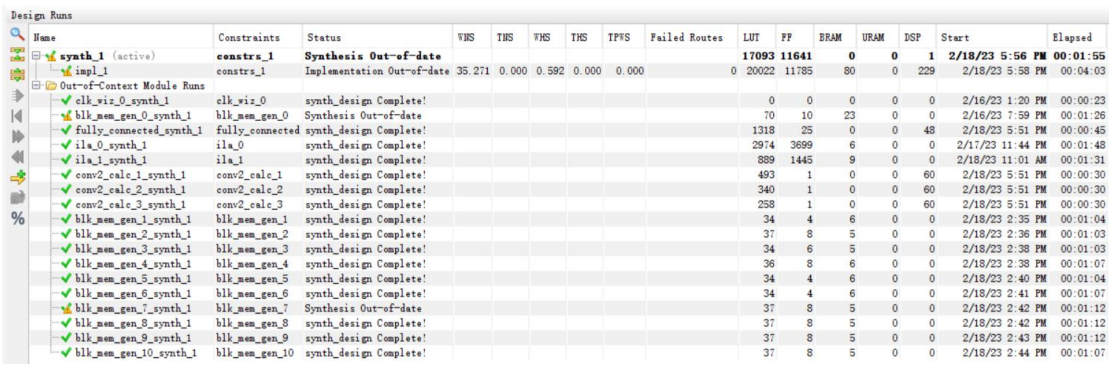
(3) 蓝牙接收完毕，VGA 显示



(4) 识别结果显示与语音播报



(5) 资源消耗



Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Failed Routes	LUT	FF	BRAM	URAM	DSP	Start	Elapsed
synth_1 (active)	**constrs_1**	**Synthesis Out-of-date**							**17093**	**11641**	**0**	**0**	**1**	**2/18/23 5:56 PM**	**00:01:55**
impl_1	constrs_1	Implementation Out-of-date	35.271	0.000	0.592	0.000	0.000	0	20022	11785	80	0	229	2/18/23 5:58 PM	00:04:03
Out-of-Context Module Runs															
clk_wiz_0_synth_1	clk_wiz_0	synth_design Complete!							0	0	0	0	0	2/16/23 1:20 PM	00:00:23
blk_mem_gen_0_synth_1	blk_mem_gen_0	Synthesis Out-of-date							70	10	23	0	0	2/16/23 7:59 PM	00:01:26
fully_connected_synth_1	fully_connected	synth_design Complete!							1318	25	0	0	48	2/18/23 5:51 PM	00:00:45
ila_0_synth_1	ila_0	synth_design Complete!							2974	3699	6	0	0	2/17/23 11:44 PM	00:01:48
ila_1_synth_1	ila_1	synth_design Complete!							889	1445	9	0	0	2/18/23 11:01 AM	00:01:31
conv2_calc_1_synth_1	conv2_calc_1	synth_design Complete!							493	1	0	0	60	2/18/23 5:51 PM	00:00:30
conv2_calc_2_synth_1	conv2_calc_2	synth_design Complete!							340	1	0	0	60	2/18/23 5:51 PM	00:00:30
conv2_calc_3_synth_1	conv2_calc_3	synth_design Complete!							258	1	0	0	60	2/18/23 5:51 PM	00:00:30
blk_mem_gen_1_synth_1	blk_mem_gen_1	synth_design Complete!							34	4	6	0	0	2/18/23 2:35 PM	00:01:04
blk_mem_gen_2_synth_1	blk_mem_gen_2	synth_design Complete!							37	8	5	0	0	2/18/23 2:36 PM	00:01:03
blk_mem_gen_3_synth_1	blk_mem_gen_3	synth_design Complete!							34	6	5	0	0	2/18/23 2:38 PM	00:01:03
blk_mem_gen_4_synth_1	blk_mem_gen_4	synth_design Complete!							36	8	6	0	0	2/18/23 2:38 PM	00:01:07
blk_mem_gen_5_synth_1	blk_mem_gen_5	synth_design Complete!							34	4	6	0	0	2/18/23 2:40 PM	00:01:04
blk_mem_gen_6_synth_1	blk_mem_gen_6	synth_design Complete!							34	4	6	0	0	2/18/23 2:41 PM	00:01:07
blk_mem_gen_7_synth_1	blk_mem_gen_7	Synthesis Out-of-date							37	8	5	0	0	2/18/23 2:42 PM	00:01:12
blk_mem_gen_8_synth_1	blk_mem_gen_8	synth_design Complete!							37	8	5	0	0	2/18/23 2:42 PM	00:01:12
blk_mem_gen_9_synth_1	blk_mem_gen_9	synth_design Complete!							37	8	5	0	0	2/18/23 2:43 PM	00:01:12
blk_mem_gen_10_synth_1	blk_mem_gen_10	synth_design Complete!							37	8	5	0	0	2/18/23 2:44 PM	00:01:07

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 屏幕这样几乎是必须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首列，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。