紫光同创杯

基于PGL22G的MobileNet-YOLO水果识别系统

答辩人：刘宇昂
队伍编号：CICC1084
团队名称：点灯稳过

赛题分析

- **分区赛赛题**
  - 水果识别：
    - 每个队伍的系统进行8次的识别测试，其中有5次是单种水果的识别，2次是2种水果混合的识别，1次是3种水果混合的识别。
    - 涉及水果模型：红苹果、青苹果、单只香蕉、紫红葡萄、火龙果、黄色胖梨子、乳白色雪梨、绿色青梨、黄芒果、青芒果、猕猴桃、红橙子。（随机的两种水果为：红山竹、栗色无花果）
  - 输出要求：
    - 当摄像头出现水果时自动输出识别信息。
    - 系统每次识别测试需要输出的信息为：XX个XX颜色的XX，其中XX依次是个数，颜色与水果种类。
  - 速度要求：
    - 从相机拍到水果开始计时，结果输出后结束计时。
    - 以3秒为一个单位，每增加3s时间分值减1分，满分5分。

赛题分析

- **分区赛赛题**
  - 水果识别：
    - 识别出红色、橙色、黄色、绿色、紫色、白色、栗色这些颜色；同时识别出苹果、香蕉、葡萄、火龙果、梨、芒果、猕猴桃、句子、山竹、无花果这些水果种类
    - 识别出对应种类的水果数量
  - 输出要求：
    - 由于选取算法限制，团队决定设置三种输出模式，分别输出当前图片的水果种类、数量和颜色，通过外部按钮控制输出

方案选择

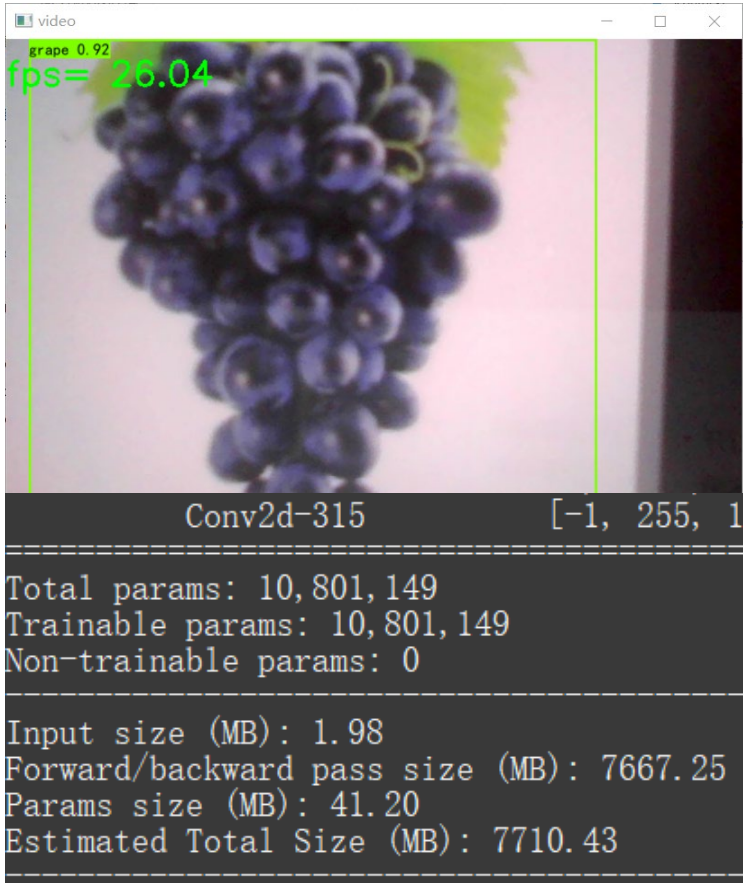
- 水果种类识别（包含随机水果种类识别）
  1. 传统ML算法（KNN&LBP）
  2. 基于CNN的YOLO神经网络（YOLO-MobileNet）
  3. 基于SNN的YOLO神经网络（Spiking-Yolo）
- 水果数量识别
  1. 遍历所有大小超过阈值的色块
  2. 对神经网络输出的数组计数
- 水果颜色识别
  1. 传统CV算法：直方图特征提取
  2. 传统ML算法：KNN

算法	主要优点	主要缺点
传统CV算法	实现简单，易于部署	抗干扰能力差
卷积神经网络	泛化能力强，可适应更多环境	片上资源消耗大，实现难度大，需要额外采集数据集
ViT、SNN等	准确度高、前景广阔	片上资源消耗大，实现难度极高，需要额外采集数据集

方案选择

改进的YOLO-MobileNet算法

- 将Backbone部分从ResNet-53替换为MobileNetv2
- 将YOLOv4的YoloBody部分移植到YOLOv3，并将主体中所有传统3x3卷积核的CBL块改为使用1x1卷积核的inverse CBL块
- 将每个特征图输出的五层卷积都改为四层
- 将上采样阶段多余卷积层删除，只留下基本的上采样层
- 借鉴YOLOv4的masaic数据增强、KNN预规划选择框等技巧，并使用基于低阶矩的自适应估计的Adam算法进行训练



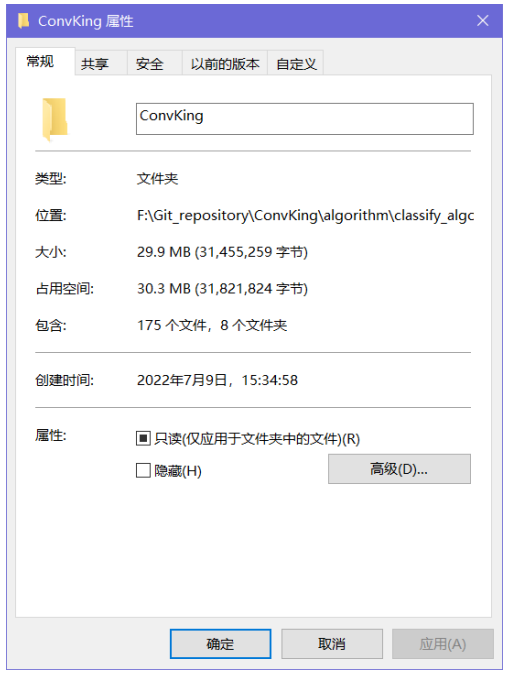
在PC测试平台（Nvidia RTX 2060）上达到约27帧的识别速率

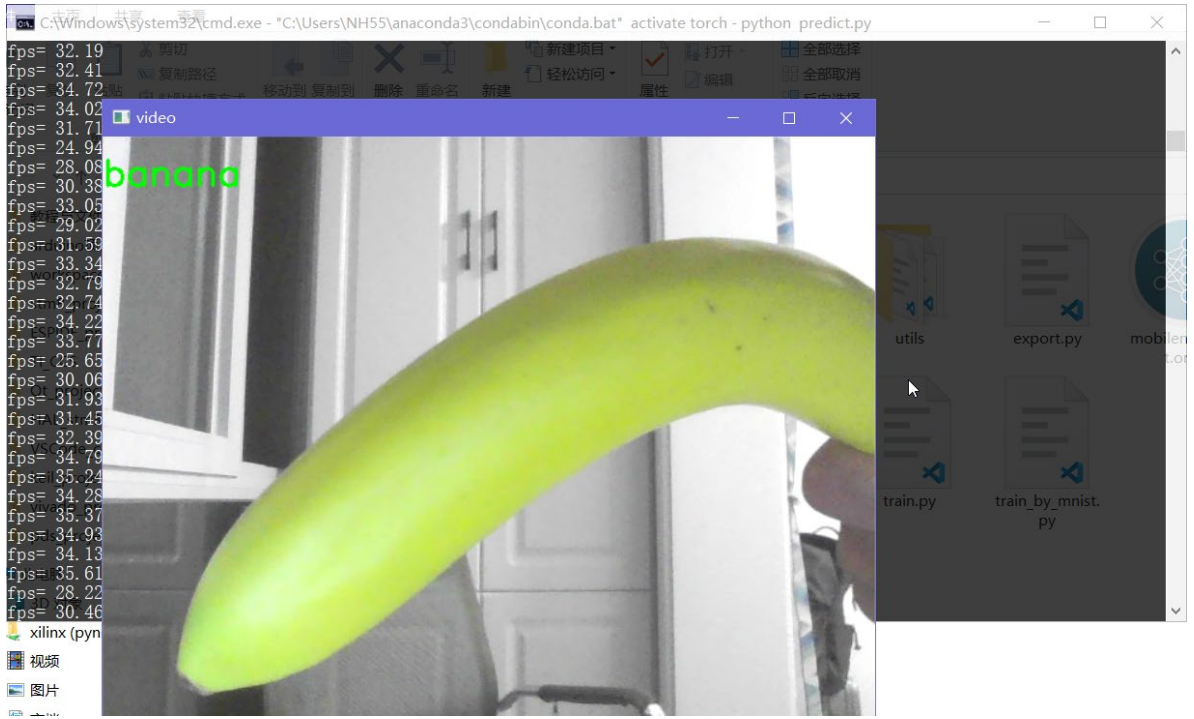
模型所需的网络总大小减少到7710.43MB

当前实现

基于MobileNet的水果分类算法

- 基于MobileNetv2在PC平台上训练了水果分类算法
- 只需要较少数据集即可实现单个水果的种类识别
  - 复赛要求识别的水果种类仅需约30MB数据集即可得到较好训练效果





在PC测试平台（Nvidia RTX 2060）上达到约33帧的识别速率

当前实现

基于KNN的水果颜色识别算法

- 预先确定水果颜色的KNN权重-图片去噪-对图像进行运算-输出最可能的颜色
- 通过计算量较小的传统机器学习算法处理颜色
- 算法参数由上位机预训练结果提供

```
64  def responseOfNeighbors(neighbors):
65      """最近邻回归"""
66      all_possible_neighbors = {}
67
68      for x in range(len(neighbors)):
69          response = neighbors[x][-1]
70          if response in all_possible_neighbors:
71              all_possible_neighbors[response] += 1
72          else:
73              all_possible_neighbors[response] = 1
74
75      sortedVotes = sorted(all_possible_neighbors.items(),
76                           key=operator.itemgetter(1), reverse=True)
77
78      return sortedVotes[0][0]
79
```

当前实现

直方图统计实现

- 直方图统计算法基于片上双口RAM
- 将RAM地址作为每个通道的所有bins，每个地址所存储的数据表示该bin对应像素数目
- 单周期内同时完成计数和比较大小输出

```
histogram_ram histogram_ram_inst (
                    .a_clk(sys_clk_i),
                    .b_clk(sys_clk_i),
                    .a_rst(!sys_rst_n_i),
                    .b_rst(1'b0),

                    .a_addr(ram_rd_addr),
                    .a_wr_data(),
                    .a_rd_data(ram_rd_data),
                    .a_wr_en(0),

                    .b_addr(ram_wr_addr_link),
                    .b_wr_data(ram_wr_data_link),
                    .b_rd_data(),
                    .b_wr_en(ram_wr_en_link)
                );
```

```
//compare
reg [judge_bitwidth - 1: 0] judge_temp;
always @( * ) begin
    if (!rst_n) begin
        judge_temp = 0;
        judge_mark = 0;
    end
    else if (compare_en) begin
        if (cnt_inside_q == 0) begin
            judge_temp[judge_bitwidth - 1: 0] = judge_rd[judge_bitwidth - 1: 0];
            judge_mark = cnt_inside;
        end
        else begin
            if (judge_temp[judge_bitwidth - 1: color_bitwidth] > judge_rd[judge_bitwidth - 1: color_bitwidth]) begin
                judge_temp[judge_bitwidth - 1: 0] = judge_rd[judge_bitwidth - 1: 0];
                judge_mark = cnt_inside_q;
            end
            else begin
                judge_temp = judge_temp;
                judge_mark = judge_mark;
            end
        end
    end
    else begin
        judge_temp = 0;
        judge_mark = 0;
    end
```

当前实现

K近邻均值算法实现

- KNN算法基于片上LUT资源，构建多个并行乘法器执行距离计算任务
- 第一个周期计算两点间距离，第二个周期完成比较任务，第三个输出分类
- 三个任务可以流水线执行

```
//compare
reg [judge_bitwidth - 1: 0] judge_temp;
always @( * ) begin
    if (!rst_n) begin
        judge_temp = 0;
        judge_mark = 0;
    end
    else if (compare_en) begin
        if (cnt_inside_q == 0) begin
            judge_temp[judge_bitwidth - 1: 0] = judge_rd[judge_bitwidth - 1: 0];
            judge_mark = cnt_inside;
        end
        else begin
            if (judge_temp[judge_bitwidth - 1: color_bitwidth] > judge_rd[judge_bitwidth - 1: color_bitwidth]) begin
                judge_temp[judge_bitwidth - 1: 0] = judge_rd[judge_bitwidth - 1: 0];
                judge_mark = cnt_inside_q;
            end
            else begin
                judge_temp = judge_temp;
                judge_mark = judge_mark;
            end
        end
    end
    else begin
        judge_temp = 0;
        judge_mark = 0;
    end
end
```

当前实现

卷积层-非线性ReLU层

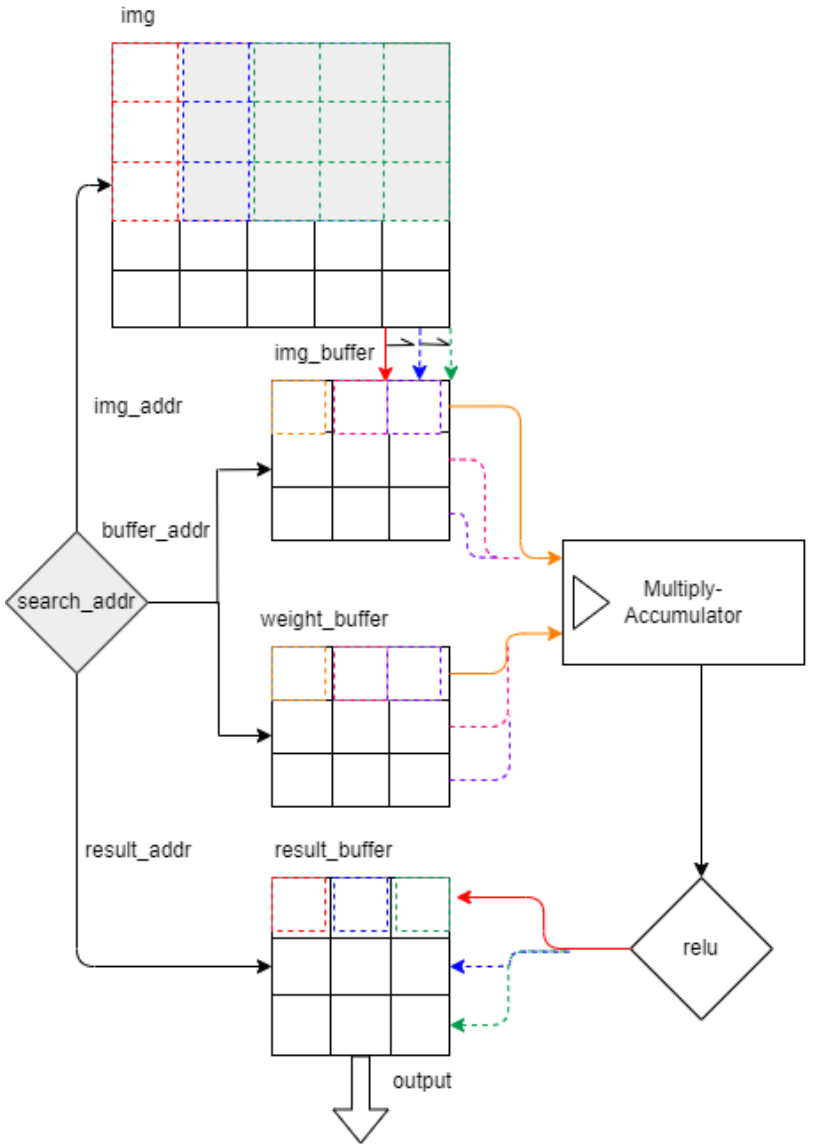
- 将MobileNet的结构简化成多个Conv-BN-ReLU层算子的结合，尽量保证卷积层的大小相等
- RTL代码中卷积层和ReLU层直接耦合
- 流水线采用多级流水的卷积层时序逻辑+一级ReLU层组合逻辑构成

search_addr模块：控制图像和计算、缓存区滑窗移动

img_buffer模块：暂存数据，图像动态加载

weight_buffer模块：权重加载缓存

- 流水线结构：依次计算滑窗位置-遍历buffer-计算



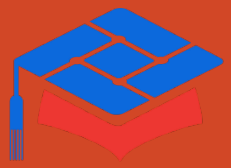
当前实现

最大池化层

- 最大池化层实现思路与卷积层类似
- 将卷积计算改为取最大值
- 使用排序移位寄存器，将遍历到的局部数据全部存入，按列排序，每排完一列就进行移动从而实现流水线

```
reg [3:0] x;
reg [3:0] y;
reg [datao_width*datao_height*bitwidth-1:0] rdata_o;
assign data_o = rdata_o;
always @(posedge clk_en)begin
    if(!reset_n)begin
        rdata_o<=0;
    end
    else if(turn_fin)begin
        for(x=0;x<datao_height;x=x+1)begin
            for(y=0;y<datao_width;y=y+1)begin
                rdata_o[(x*datao_width+y)*bitwidth +:bitwidth] <= result_array[x][y];
            end
        end
    end
    else if(all_fin)begin
        rdata_o<=rdata_o;
    end
    else begin
        rdata_o<=0;
    end
end
```

当前实现

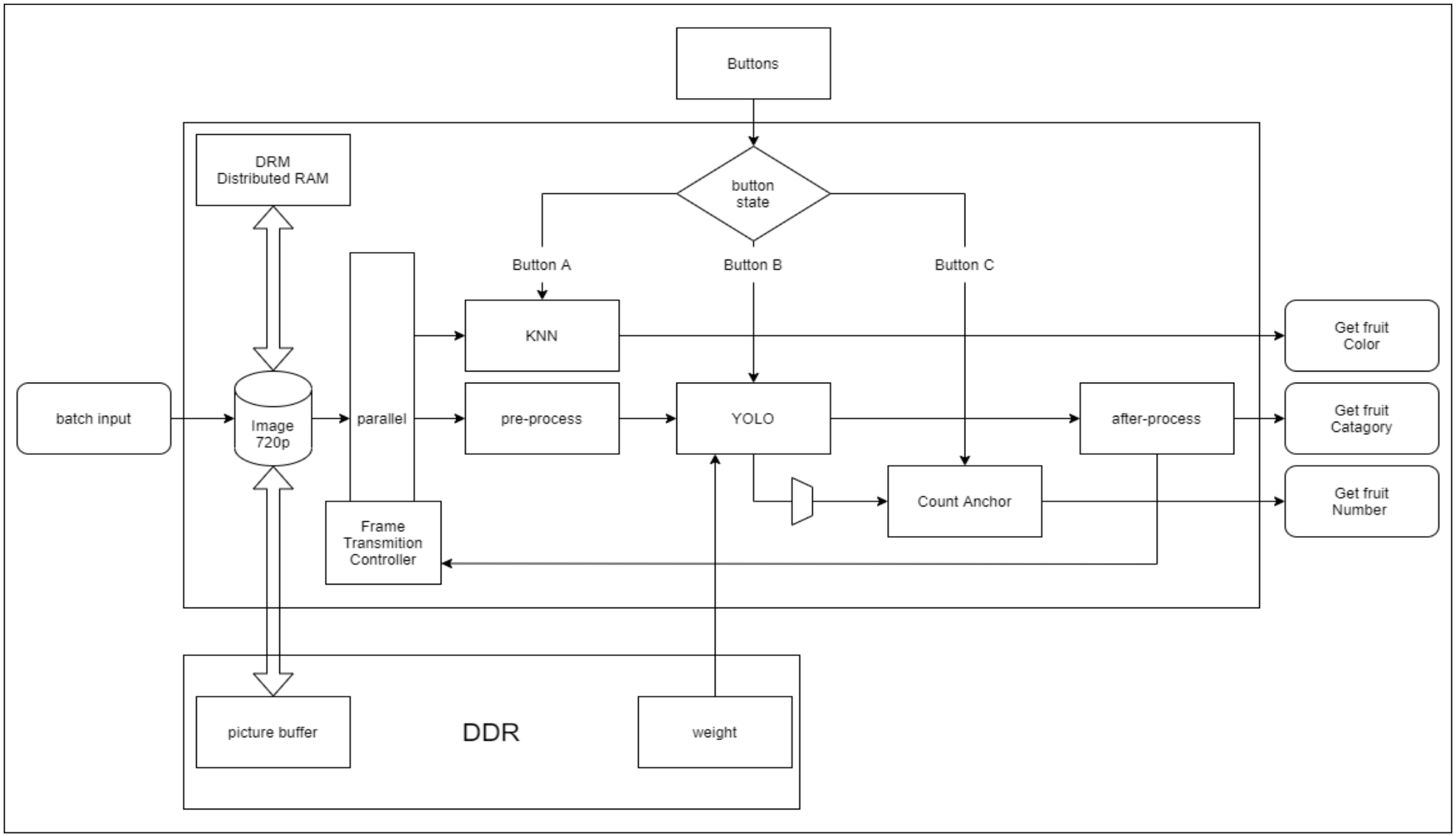


上采样层

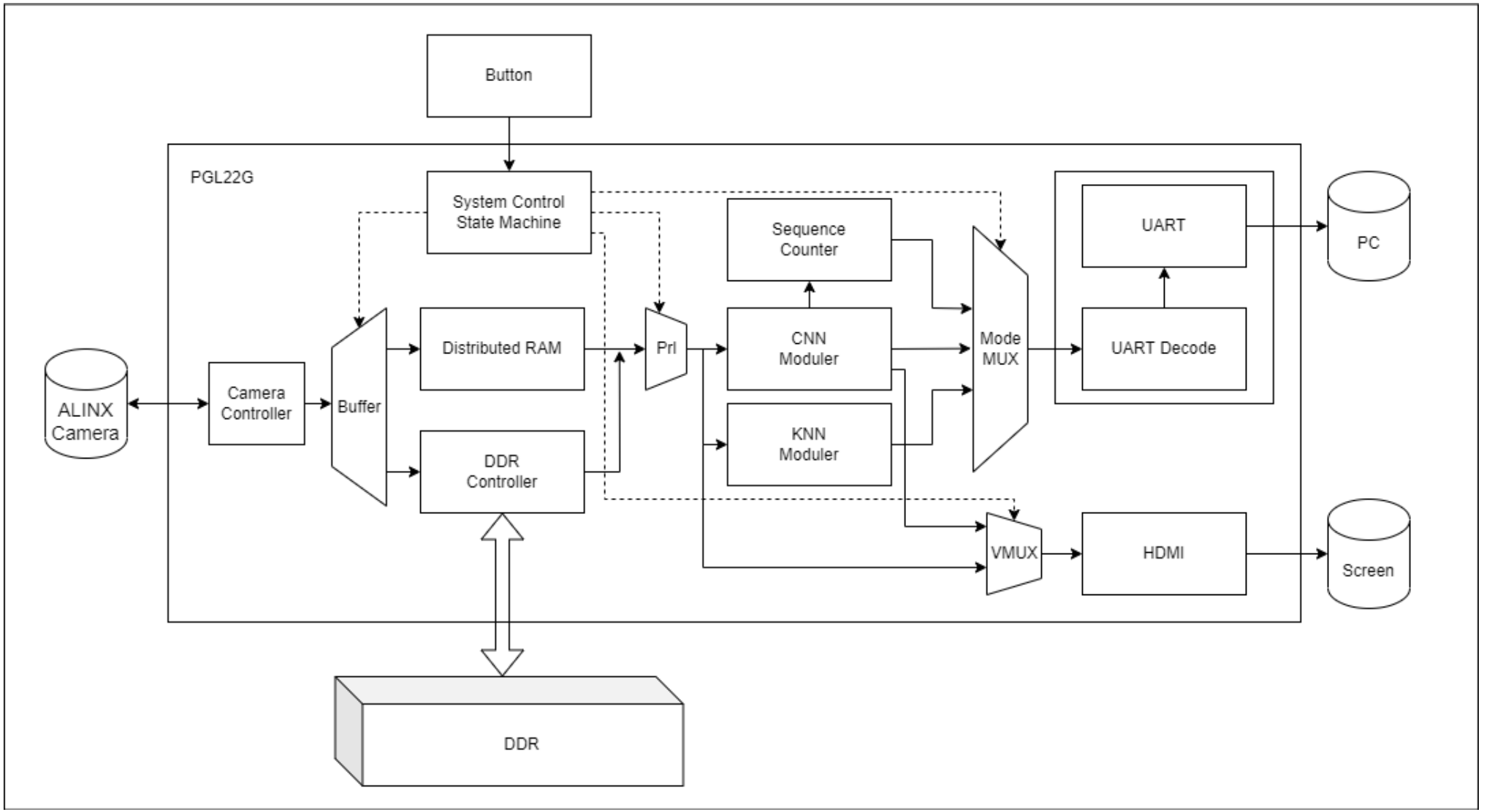
- 全并行计算，以空间换时间
- 例化多个计算模块，在单周期内完成缓存区内所有数据的上采样计算

```
generate
    genvar i, j, m, n;
    for (i = 0;i < data_i_height;i = i + 1) begin
        for (j = 0;j < data_i_width;j = j + 1) begin
            for (m = i * scale_factor;m < i * scale_factor + scale_factor;m = m + 1) begin
                for (n = j * scale_factor;n < j * scale_factor + scale_factor;n = n + 1) begin
                    assign data_o [(m * data_o_width + n) * bitwidth +: bitwidth] = data_i [(i * data_i_width + j) * bitwidth
                end
            end
        end
    end
endgenerate
```

算法结构


Buttons
DRM
Distributed RAM
button
state
Button A
Button B
Button C
KNN
batch input
Image
720p
parallel
pre-process
YOLO
after-process
Get fruit
Color
Get fruit
Catagory
Get fruit
Number
Count Anchor
Frame
Transmition
Controller
picture buffer
DDR
weight

系统架构


Button
PGL22G
System Control State Machine
Sequence Counter
UART
PC
Distributed RAM
Prl
CNN Moduler
Mode MUX
UART Decode
ALINX Camera
Camera Controller
Buffer
KNN Moduler
DDR Controller
VMUX
HDMI
Screen
DDR

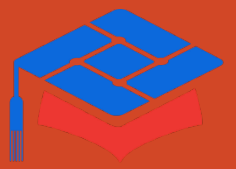
子模块设计

- 硬件加速模块
  - CBL结构
    - Relu层
    - 卷积层
  - 线性层
  - 上采样层
  - 最大池化层
- 颜色识别模块
  - 直方图统计
  - KNN
- 前处理与后处理模块

Accelerator	2022/6/26 22:08	文件夹
Bridge	2022/4/20 19:30	文件夹
Camera	2022/4/20 12:05	文件夹
Peripheral	2022/6/27 10:33	文件夹
Uart	2022/5/10 18:55	文件夹
Weight		
mult_add.v		
top.v		

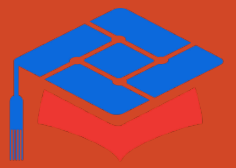
CBL_Layer	2022/6/26 14:49	文件夹
Conv	2022/7/8 9:55	文件夹
Conv-v1	2022/4/12 19:48	文件夹
Conv-v2	2022/5/30 10:23	文件夹
Linear	2022/4/12 16:11	文件夹
MaxPool	2022/7/8 9:58	文件夹
ReLU	2022/7/8 9:58	文件夹
UpSample	2022/7/8 9:58	文件夹
mobilenet_invres.v	2022/6/26 22:43	V 文件
mobilenet_top.v	2022/6/26 22:43	V 文件
rgb2gray.v	2022/5/11 17:09	V 文件
yolo_aft.v	2022/5/9 14:03	V 文件
yolo_CBL.v	2022/6/26 21:49	V 文件
yolo_mobilenet.v	2022/6/26 22:42	V 文件
yolo_pre.v	2022/5/10 21:11	V 文件
yolo_top.v	2022/6/26 22:42	V 文件

主要创新点



- 可部署在FPGA上的mobilenet图像分类神经网络算法CBL结构
- 基于mobilenet对当前主流的yolo算法进行改进
- 具有内部流水的卷积层、非线性层、最大池化层算子
- 能够硬件执行的KNN算法直方图统计

应用前景



- 物品识别是智能机器的基本功能之一，该功能无论在军事还是民用中都有着广泛的应用场景
- 当前的目标识别算法通常运行在GPU或其他通用加速器上；基于FPGA的目标识别算法仍有待探索
- 本项目期望基于FPGA平台实现一套高可执行性的YOLO目标识别（目标检测）算法，可大大降低计算能耗、提高计算速度，在摩尔定律减缓的情况下帮助高性能的专用算法落地部署

谢谢观看