海云捷迅
AWCLOUD

全国大学生集成电路创新创业大赛

基于FPGA加速的工业缺陷检测系统

A组 CICC1577
冰糖葫芦儿

成员：纪佳骏 张峰 宋德浩

目录

/01

团队介绍与研发情况

·团队介绍

·项目进展

团队介绍

01

纪佳骏

专业：电子信息工程

负责：Linux工程优化

02

张峰

专业：通信工程

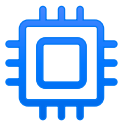
负责：FPGA工程优化

03

宋德浩

专业：电子信息工程

负责：模型优化



项目进展

240ms

3月

Demo整合

112ms

4月

SDK优化、FPGA工程优化、频率提升、模型量化训练

64ms

5月

重写DVP、VGA IP，时序约束、逻辑固化、剪枝量化联合训练、数据重排PL加速可行性验证

6、7月

剪枝算法优化、引入PL侧绘制预测框IP、PL侧Resize加速IP、DVP_VGA连续传输、增量编译、工程时序优化等

/02

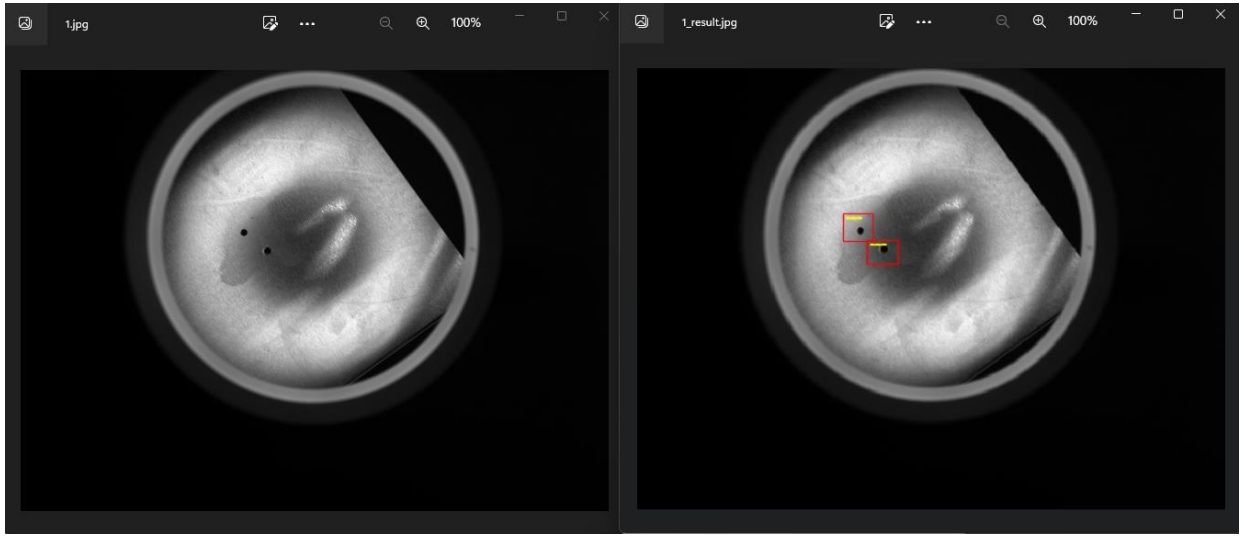
效果演示与性能指标

效果演示—单次推理（DEMO测试）

推理速率

```
------------------graph begin------
input_organize_time:10.588405ms
fpga_time:37.580349ms
output_organize_time:3.950532ms
------------------graph end------
------------------graph begin------
input_organize_time:9.738510ms
fpga_time:37.172298ms
output_organize_time:3.657476ms
------------------graph end------
predictor run:64.015ms
success detection, image size: 640, 480, detect object: zhen_kong, score: 0.996985, location: x
=236, y=157, width=23, height=25
success detection, image size: 640, 480, detect object: zhen_kong, score: 0.784001, location: x
=267, y=186, width=35, height=30
----------------- Config info ------------------
runtime_device: armv8
precision: int8/float3
num_threads: 2
---------------- Data info ----------------------
batch_size: 1
---------------- Model info ---------------------
Model_name: new1ch_3.nb
---------------- Perf info ----------------------
Total number of predicted data: 1 and total time spent(s): 77
preproce_time(ms): 12.7588, inference_time(ms): 64.1086, postprocess_time(ms): 1.05696
fpga release
root@awcloud:/opt/paddle_frame#
```

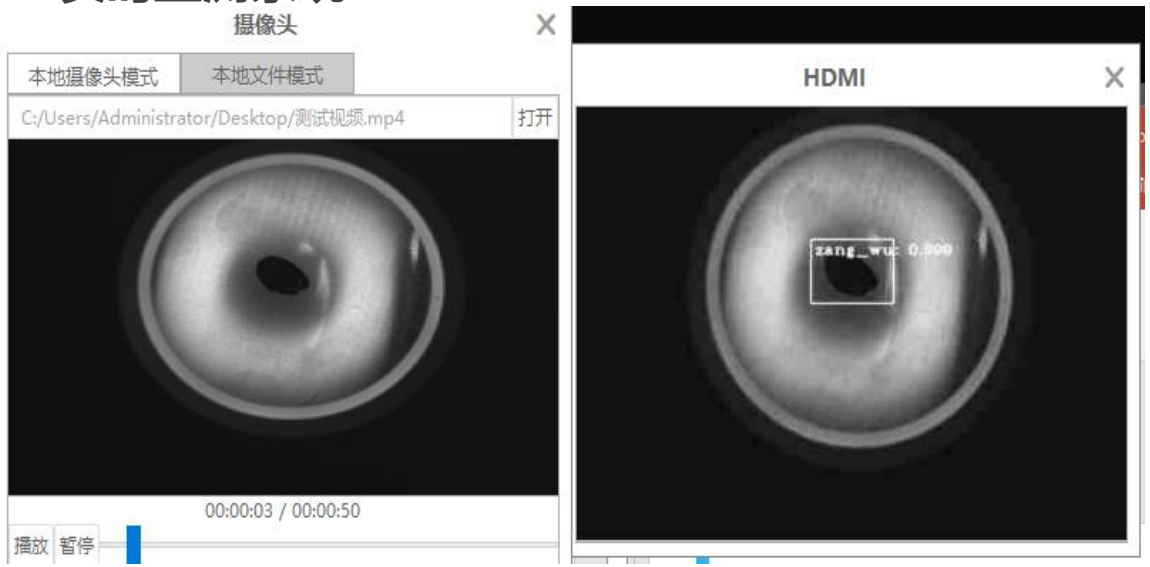
预测结果



该程序使用SD卡中图片进行推理，可以看到推理时间为64.1ms。并且我们对预处理时间进行了优化，达到了12.75ms。推理结果写入SD卡，如右图所示，可以看到成功检测出了缺陷（这里我们使用了较难监测出的“**针孔**”缺陷，以体现我们模型的**泛化性**）。

效果演示

实时监测系统



- 重构DVP_VGA IP实现连续传输，使视频的推流无需PS控制即可自动完成。
- 重构PL_Plot模块，完成PL侧绘制预测框，PS侧通过Avalon_lw桥控制状态寄存器，实现预测框的实时显示。

进程演示

```
[2023-8-16 10:34:5.497]
success detection, image size: 300, 300, detect object: , score: 1, location: x=279, y=134, width=95, heig
ht=70
------------------------------------------------------------------------------------------
------------------graph begin------
input_organize_time:10.023458ms
fpga_time:38.502155ms
output_organize_time:3.674703ms
------------------graph end------
[2023-8-16 10:34:5.566]
success detection, image size: 300, 300, detect object: , score: 1, location: x=276, y=135, width=95, heig
ht=69
------------------------------------------------------------------------------------------
------------------graph begin------
input_organize_time:10.228271ms
fpga_time:38.563095ms
output_organize_time:3.669041ms
------------------graph end------
[2023-8-16 10:34:5.637]
success detection, image size: 300, 300, detect object: , score: 1, location: x=276, y=135, width=94, heig
ht=69
------------------------------------------------------------------------------------------
```

使用官方提供的LOG参考范式，完成推理结果的输出与时间统计。可以看到推理结果的刷新时间为69ms。

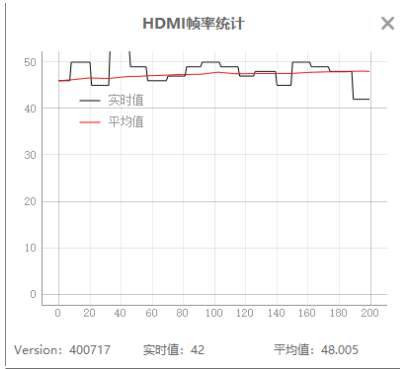
性能指标

a.Demo单次推理时间:

```
------------------graph begin------
input_organize_time:10.588405ms
fpga_time:37.580349ms
output_organize_time:3.950532ms
------------------graph end------
------------------graph begin------
input_organize_time:9.738510ms
fpga_time:37.172298ms
output_organize_time:3.657476ms
------------------graph end------
predictor run:64.015ms
success detection, image size: 640, 480, detect object: zhen_kong, score: 0.996985, location: x
=236, y=157, width=23, height=25
success detection, image size: 640, 480, detect object: zhen_kong, score: 0.784001, location: x
=267, y=186, width=35, height=30
----------------- Config info ------------------
runtime_device: armv8
precision: int8/float3
num_threads: 2
---------------- Data info ---------------------
batch_size: 1
---------------- Model info --------------------
Model_name: new1ch_3.nb
---------------- Perf info ---------------------
Total number of predicted data: 1 and total time spent(s): 77
preproce_time(ms): 12.7588, inference_time(ms): 64.1086, postprocess_time(ms): 1.05696
fpga release
root@awcloud:/opt/paddle_frame#
```

Inference_time:64.108ms

b1.输出图像刷新速率:　　平均值：48帧



HDMI_refresh_time:20.808ms

b2.推理结果刷新速率:

```
[2023-8-16 10:34:5.497]
success detection, image size: 300, 300, detect object: , score: 1, location: x=279, y=134, width=95, heig
ht=70
----------------------------------------------------------------------------------------
------------------graph begin------
input_organize_time:10.023458ms
fpga_time:38.502155ms
output_organize_time:3.674703ms
------------------graph end------
[2023-8-16 10:34:5.566]
success detection, image size: 300, 300, detect object: , score: 1, location: x=276, y=135, width=95, heig
ht=69
----------------------------------------------------------------------------------------
------------------graph begin------
input_organize_time:10.228271ms
fpga_time:38.563095ms
output_organize_time:3.669041ms
------------------graph end------
[2023-8-16 10:34:5.637]
success detection, image size: 300, 300, detect object: , score: 1, location: x=276, y=135, width=94, heig
ht=69
----------------------------------------------------------------------------------------
```

Inference_refresh_time:69.384ms

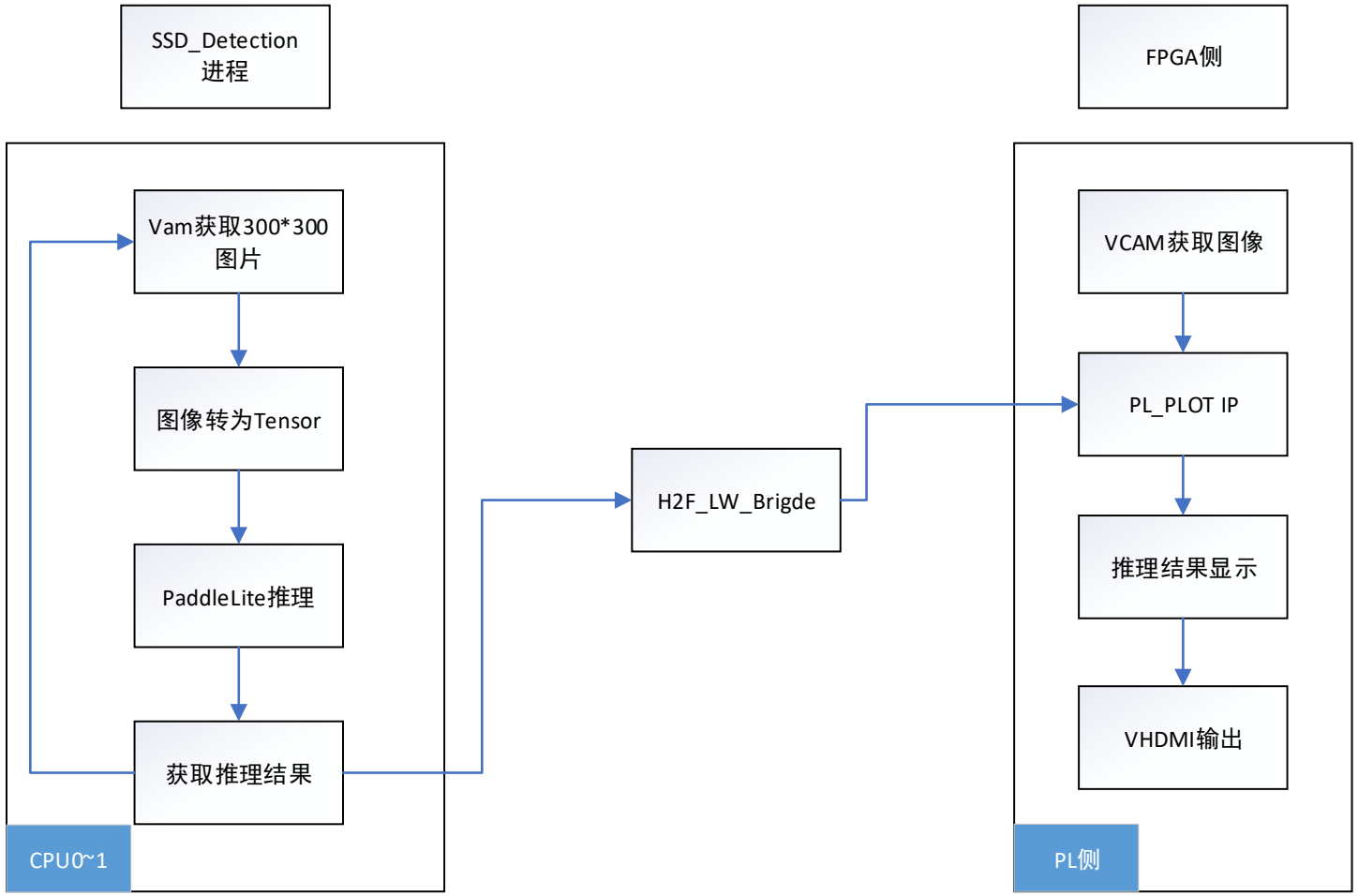
创新点：我们的单次推理时间与推理刷新时间相差仅5ms！！

/03

关键技术分析

- 软件程序设计
- FPGA工程优化
- 模型优化
- SDK与PaddLite优化

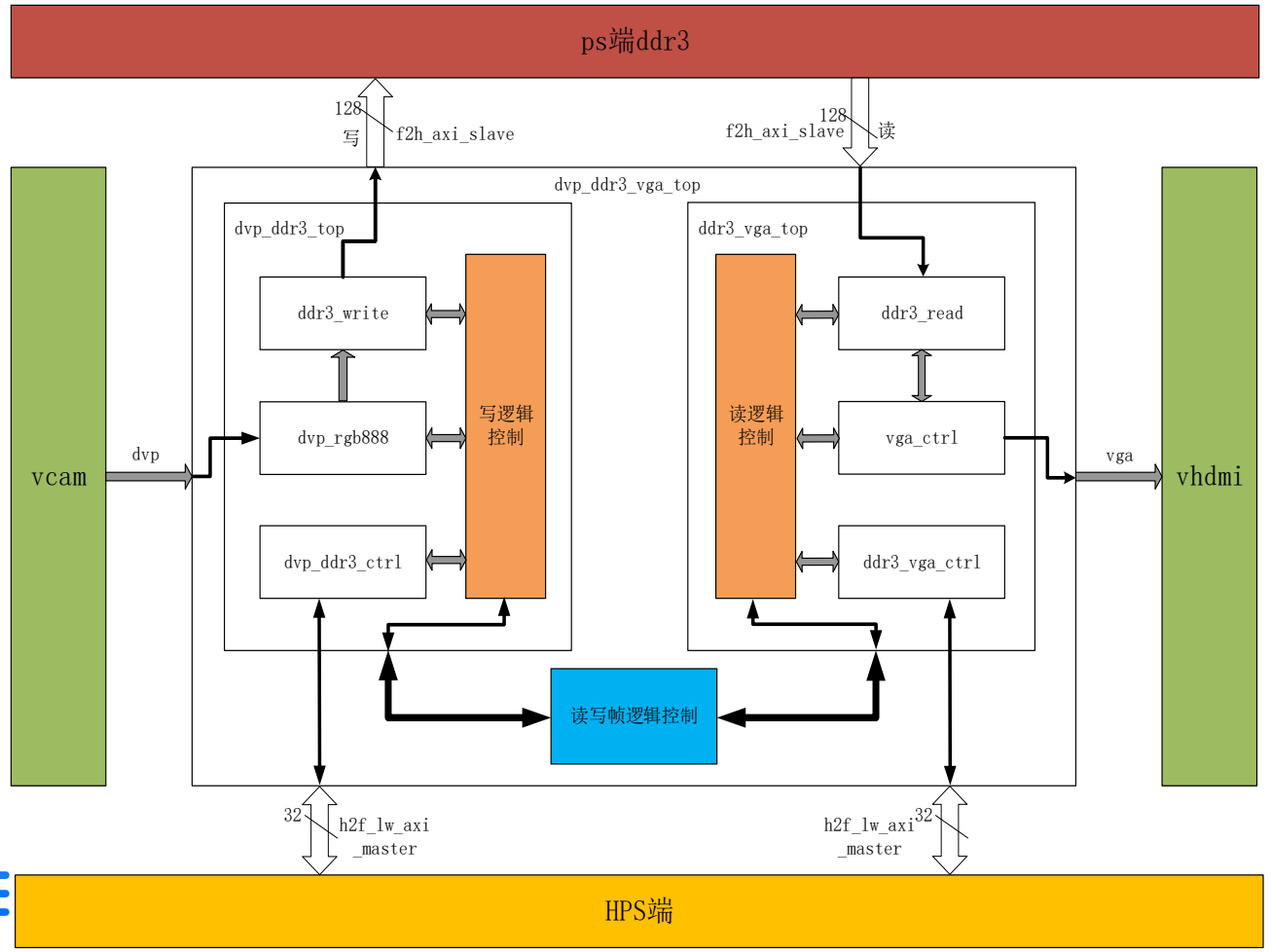
软件程序设计



- 为实现实时推理，我们在分区赛双进程的基础上进一步优化。
- 提出了PL侧绘制预测框的思想，从而可将全部CPU资源用于推理，进一步提升推理刷新速率

FPGA工程优化-重写dvp_ddr3以及ddr3_vga

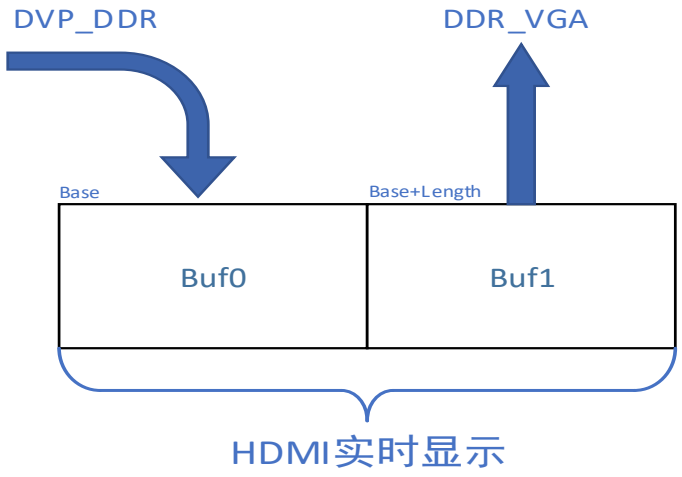
dvp_ddr3_vga模块框图

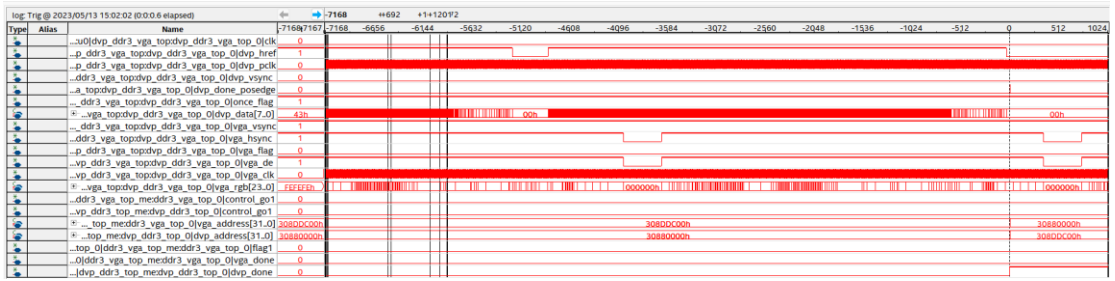


- 重写官方ip，将其整合为dvp_ddr_vga一个ip专门实现**HDMI视频流显示**。
- 对于该模块，其输入为vcam传来的**dvp时序**，输出为传输给vhdmi的**vga时序**。
- HPS端通过h2f_lw桥读取dvp_ddr_vga模块的**状态寄存器**确认工作状态。
- 可自定义添加图片预处理、Resize、PL端预测框绘制等模块

FPGA工程优化-双buffer连续写

双buffer连续读写

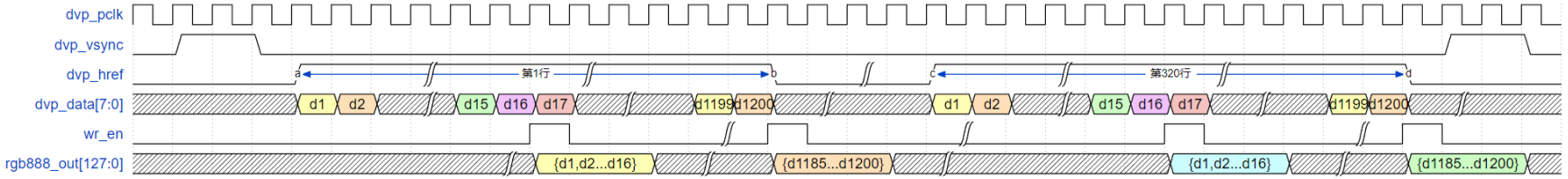




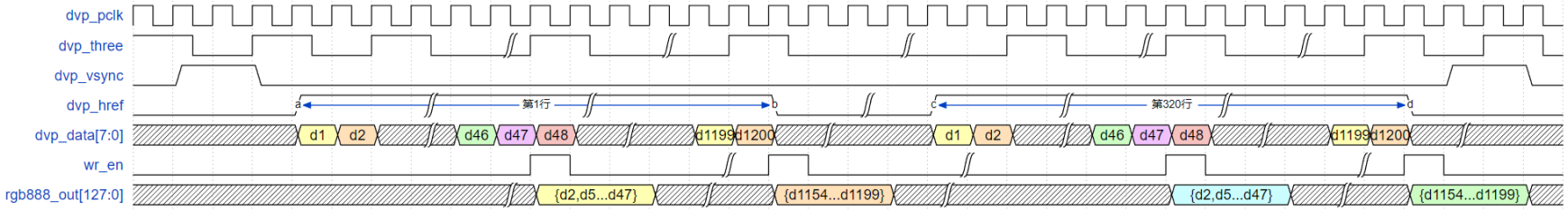
- 官方的dvp_ddr3模块HPS请求一帧经过**66ms**才能得到数据。
- 我们自己设计的dvp_ddr3_vga模块，其无需hps请求控制即可完成对HPS端DDR两个buffer的连续读写，从而无需请求等待的**66ms**。
- Vcam的写入与Vhdmi的读取共用双Buf，通过DVP_DDR以及DDR_VGA模块互斥读取DDR进行实时视频推流，无需HPS复制。
- 我们**重写了amm驱动**，使其能申请更大的连续内存空间。

FPGA工程优化-单通道传输优化

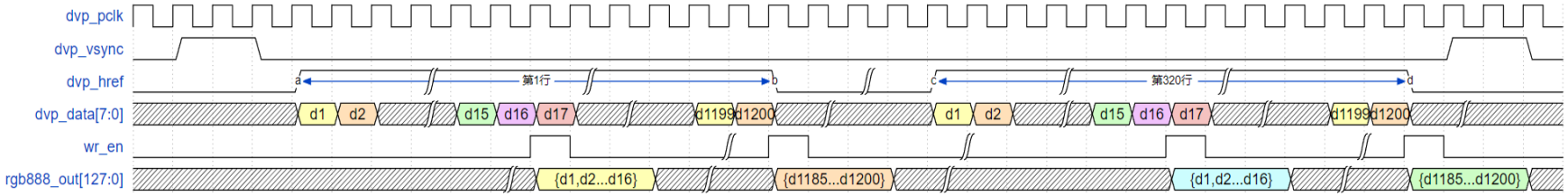
三通道dvp_rgb888时序



单通道dvp_rgb888时序



三通道vga_ctrl时序

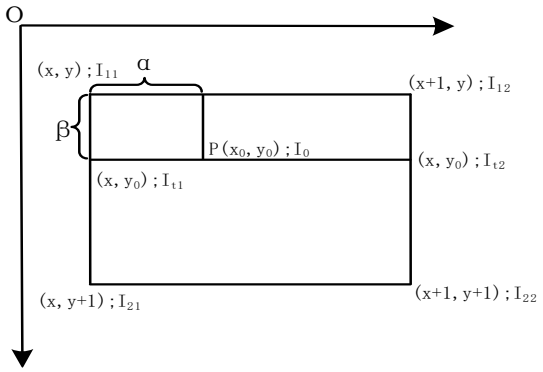


- 由于测试数据集图片都是灰度图像，三个通道是完全一致的，因此可以训练一个单通道模型、可有效**降低模型大小**并**提升推理速度**。

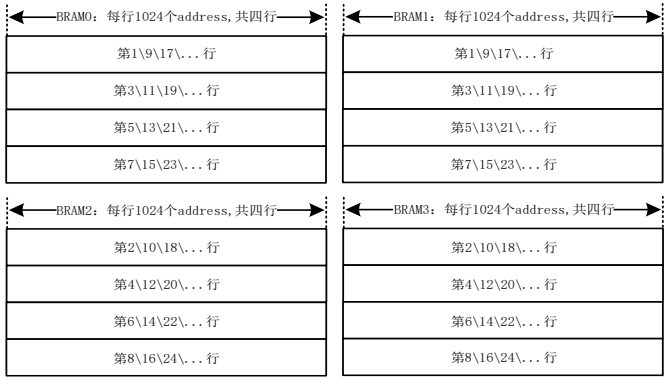
- 为了配合HPS端进行单通道模型推理，FPGA端写入HPS的ddr时只写入单通道**400*320**大小图片，与原三通道**400*320*3**大小图片相比传输数据量降低**两倍**。

FPGA工程优化-重构Resize

双线性插值示意图

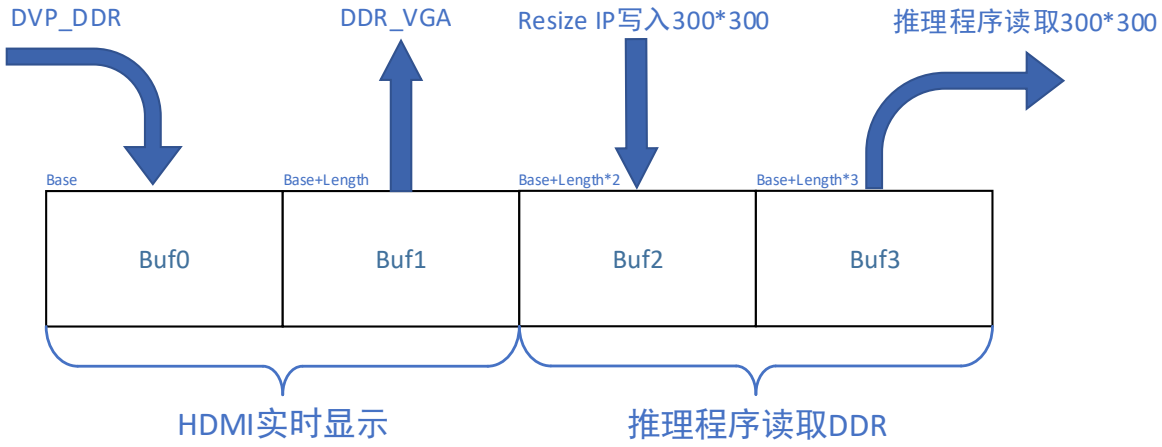


FPGA实现

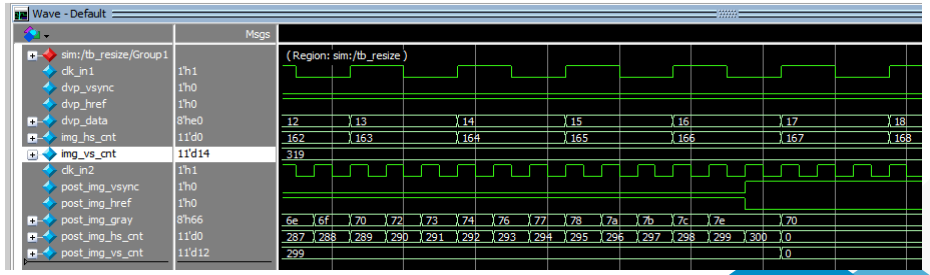


- 我们在**PL侧**重构了**双线性插值**算法，进行图像的Resize加速。
- 在FPGA中我们使用四个**双口BRAM**存储行，分别输出四个像素进行处理。
- 由于在推理刷新过程中，将图片Resize为300*300大小需要14ms。因此减少**resize时间**是提高整体推理刷新速率的重要手段。

使用resize模块后的六buffer结构

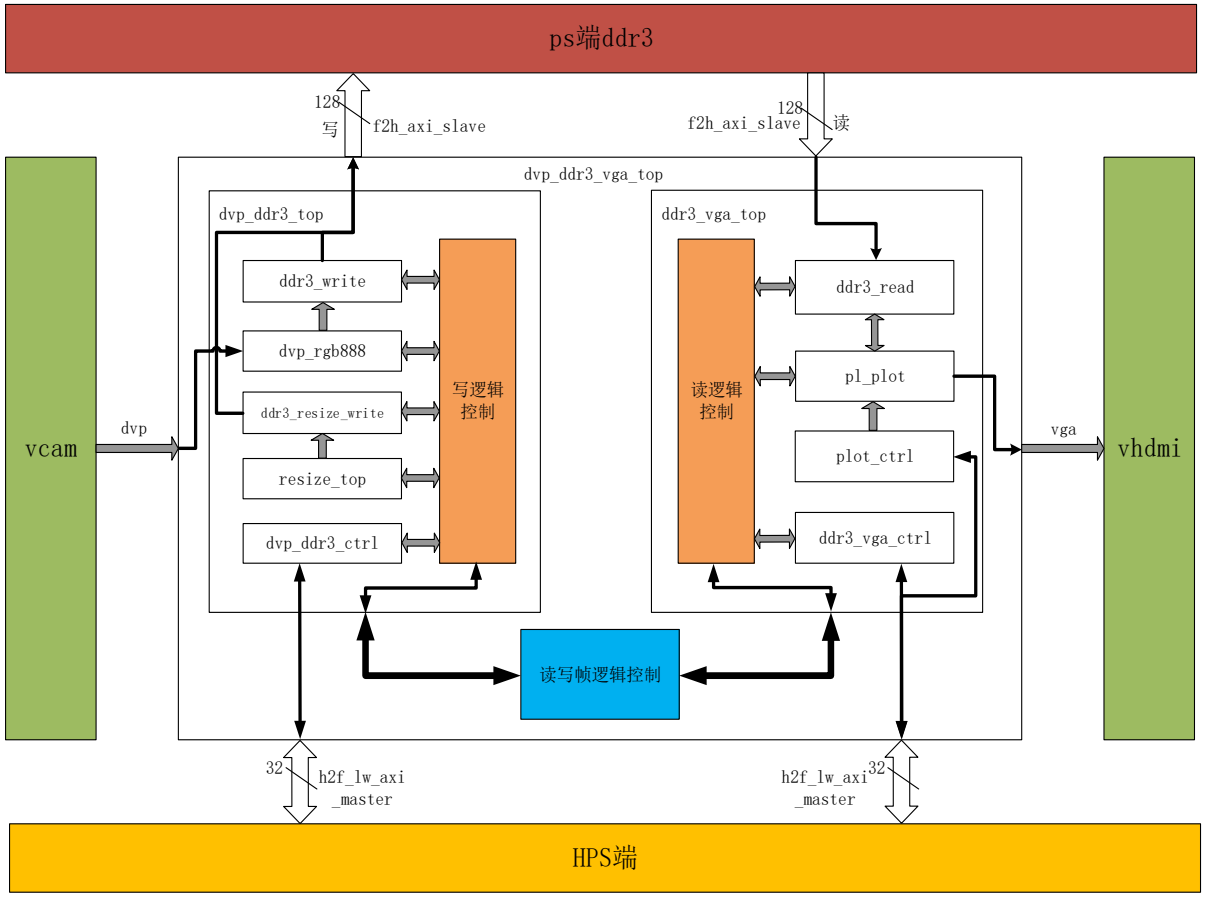


- 使用Resize的buffer结构如左图所示，其中Buf0~Buf1存储400*320的单通道图像，进行视频推流工作。Buf2~Buf3为Resize后的300*300单通道图像，使用FPGA加速后总的图片预处理时间从**29ms**减少到了**2ms。**

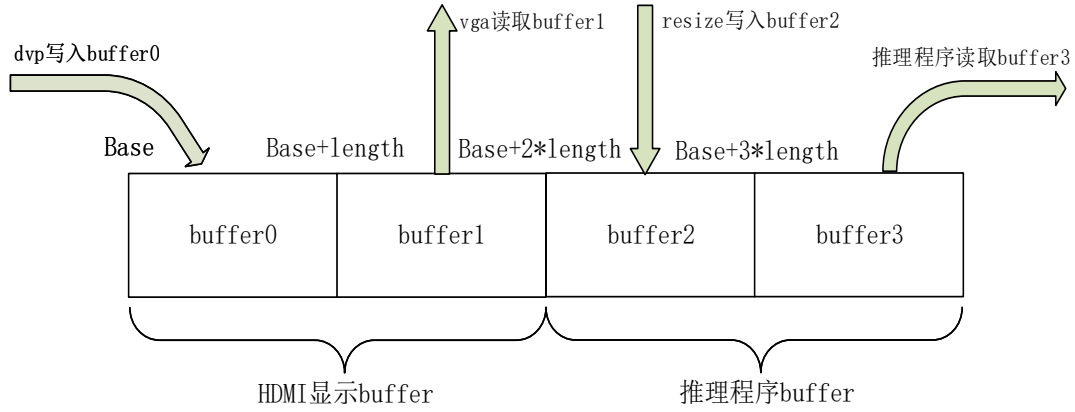


FPGA工程优化-PL端叠加推理结果

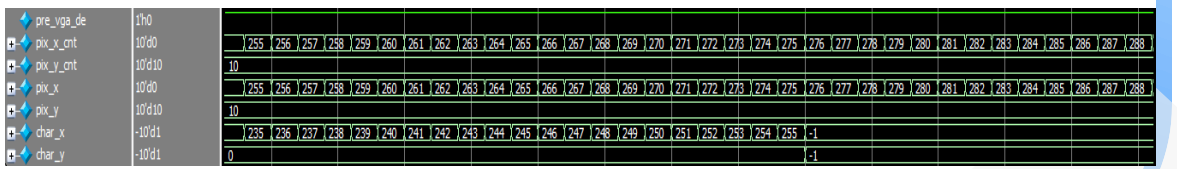
最终的dvp_ddr3_vga模块框图



使用resize以及plot模块后的四buffer结构



- 我们在**PL侧**实现了推理结果的绘制，因此可以将HDMI刷新程序完全由PL端实现，不需要PS端将推理结果叠加到帧图像上，实现HDMI刷新与推理刷新的分离。
- 推理结果的显示由推理程序通过lw总线写入到PL_Plot的状态寄存器，完成预测框的实时显示功能。

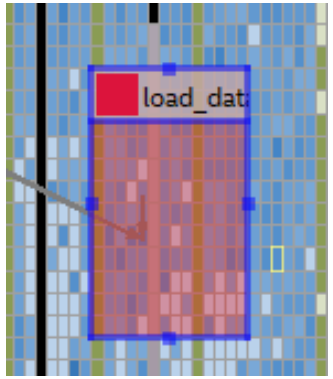


FPGA工程优化-CNN频率提升

- 移除组件——移除JTAG电路以及无用电路
- 时序约束——使用175Mhz的时序约束文件进行约束
- 时序分析——找到出现不符合时序要求的部分
- 逻辑固化——使用Chip Planner对关键路径逻辑固化
- 时序一致——使f2h_sdram总线与CNN时钟一致
- 增量编译——优化PL侧模块过多引起的时序不收敛

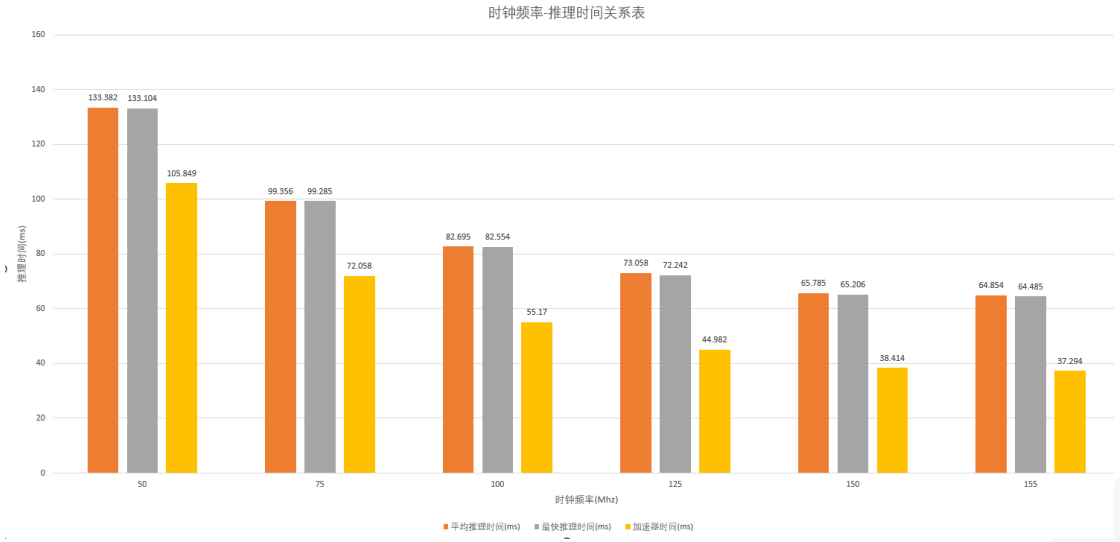
```
create_generated_clock -name {u0|pll_1_cnn|altera_pll_i|general[0].gpll~FRACTIONAL_PLL|vcoph[0]}
-source [get_pins {u0|pll_1_cnn|altera_pll_i|general[0].gpll~FRACTIONAL_PLL|refclkin}] -duty_cycle
50.000 -multiply_by 7 -master_clock {FPGA_CLK1_50} [get_pins {
u0|pll_1_cnn|altera_pll_i|general[0].gpll~FRACTIONAL_PLL|vcoph[0] }]
create_generated_clock -name {u0|pll_1_cnn|altera_pll_i|general[0].gpll~PLL_OUTPUT_COUNTER|divclk}
-source [get_pins {u0|pll_1_cnn|altera_pll_i|general[0].gpll~PLL_OUTPUT_COUNTER|vco0ph[0]}]
-duty_cycle 50.000 -multiply_by 1 -divide_by 2 -master_clock
{u0|pll_1_cnn|altera_pll_i|general[0].gpll~FRACTIONAL_PLL|vcoph[0]} [get_pins {
u0|pll_1_cnn|altera_pll_i|general[0].gpll~PLL_OUTPUT_COUNTER|divclk }]
create_generated_clock -name
{u0|pll_0_sdram|altera_pll_i|general[0].gpll~PLL_OUTPUT_COUNTER|divclk} -source [get_pins
{u0|pll_0_sdram|altera_pll_i|general[0].gpll~PLL_OUTPUT_COUNTER|vco0ph[0]}] -duty_cycle 50.000
-multiply_by 1 -divide_by 3 -master_clock
{u0|pll_1_cnn|altera_pll_i|general[0].gpll~FRACTIONAL_PLL|vcoph[0]} [get_pins {
u0|pll_0_sdram|altera_pll_i|general[0].gpll~PLL_OUTPUT_COUNTER|divclk }]
```

Partition Name	Netlist Type	Color
Top	Source File	
cnn_top:cnn_top_0	Post-Fit	
dvp_ddr3_top_me:dvp_ddr3_top_0	Post-Fit	
receive_top:vhdmi_0	Post-Synthesis	
vcam_top:vcam_0	Post-Synthesis	



在82%剪枝率，其他模块最优化条件下

频率(MHZ)	平均推理时间(MS)	最快推理时间(MS)	加速器运行时间(MS)
50	133.382	133.104	105.849
75	99.356	99.285	72.058
100	82.695	82.554	55.170
125	73.058	72.242	44.982
150	65.785	65.206	38.414
155	64.854	64.485	37.294

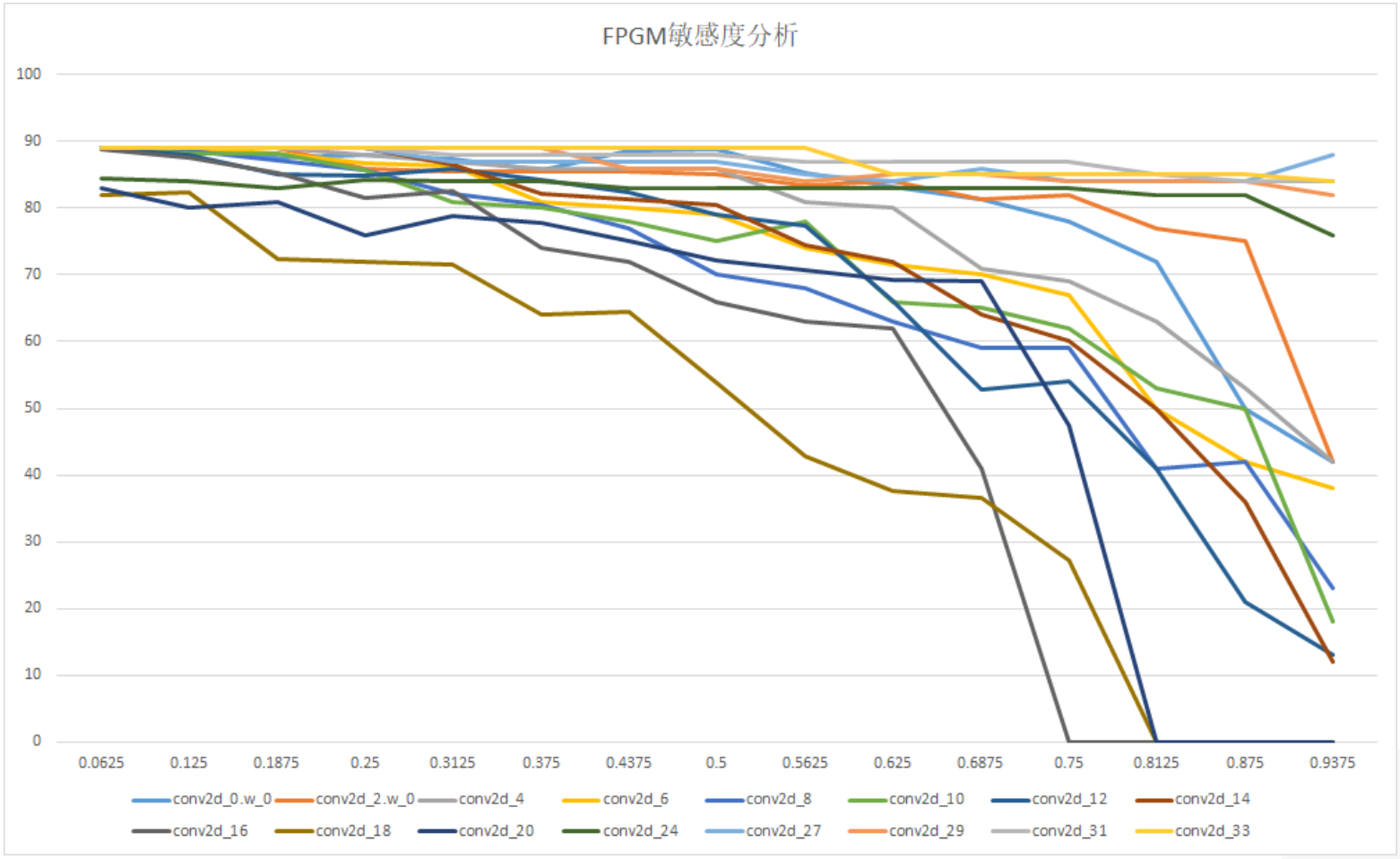


模型优化-敏感度分析剪枝

敏感度分析剪枝

在剪枝率步进的选择上，由于加速器是**8通道**的，因此我们选择8为步进进行敏感度分析（使得通道数为8的整数倍）。

注意：在通道数较少时，对于剪枝率选择应满足剩余通道数是**8的整数倍**，避免多余通道使加速器多运算一次。



基于L1和FPGM法则的综合剪枝策略

在参考相关综合剪枝论文的基础上，提出在计算**卷积层通道重要性**时选择**FPGM和L1重要性评估方法加权**得到重要性分数。

在训练集与测试集划分比例为9：1情况下比较剪枝策略优化前后的mAP指标，发现mAP由原有的**86%**提高到**91%**，证明混合不同剪枝策略能够一定程度上在相同压缩率下达到更高的检测精度。

```
        reduce_dims = [i for i in range(len(value.shape)) if i != pruned_axis]
        l1norm = np.mean(np.abs(value), axis=tuple(reduce_dims))
        if groups > 1:
            l1norm = l1norm.reshape([groups, -1])
            l1norm = np.mean(l1norm, axis=1)

        # sorted_idx = l1norm.argsort()

        dist_sum_list = []
        for out_i in range(value.shape[0]):
            dist_sum = self.get_distance_sum(value, out_i)
            dist_sum_list.append(dist_sum)
        scores = np.array(dist_sum_list)

        if groups > 1:
            scores = scores.reshape([groups, -1])
            scores = np.mean(scores, axis=1)

        sorted_idx = (scores/2+l1norm/2).argsort()
```

模型优化-输出层缩放与单通道模型训练

输出层缩放

修改mobilenet_v1.py的源码对6个**输出卷积层**的输出通道数进行缩放，从而达到输出层剪枝的目的。

```
output_prune_ratios=np.array([0.5,0.5,0.5,0.5,0.5,0.5],dtype=float)
```

```
        elif(i==4):
            tmp = self.add_sublayer(
            "conv5_" + str(i + 1),
                sublayer=DepthwiseSeparable(
                    in_channels=int(512*prune_ratios[10]),
                    # in_channels=128,#128
                    out_channels1=int(512*prune_ratios[10]),
                    out_channels2=int(512*output_prune_ratios[0]),
                    num_groups=int(512*prune_ratios[10]),
                    stride=1,
                    scale=scale,
                    conv_lr=conv_learning_rate,
                    conv_decay=conv_decay,
                    norm_decay=norm_decay,
                    norm_type=norm_type,
                    name="conv5_" + str(i + 1)))
            self.dwsl.append(tmp)
            self._update_out_channels(int(512*output_prune_ratios[0]), len(self.dwsl), feature_maps)
    dws56 = self.add_sublayer(
        "conv5_6",
        sublayer=DepthwiseSeparable(
            in_channels=int(512* output_prune_ratios[0]* scale),
            out_channels1=int(512*output_prune_ratios[0]),
            out_channels2=1024*prune_ratios[11],
            num_groups=int(512*output_prune_ratios[0]),
```

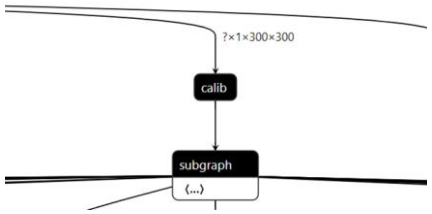
单通道模型

1.对第一个输入卷积层的通道数进行修改
2.对图像读取的算子进行修改
（**Decode**、Resize、NormalizeImage）

输入卷积层修改

```
self.conv1 = ConvBNLayer(
    in_channels=1,
```

```
im_copy = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
im= np.zeros((im.shape[0], im.shape[1], 1), np.uint8)
im[:,:,0]=im_copy
```

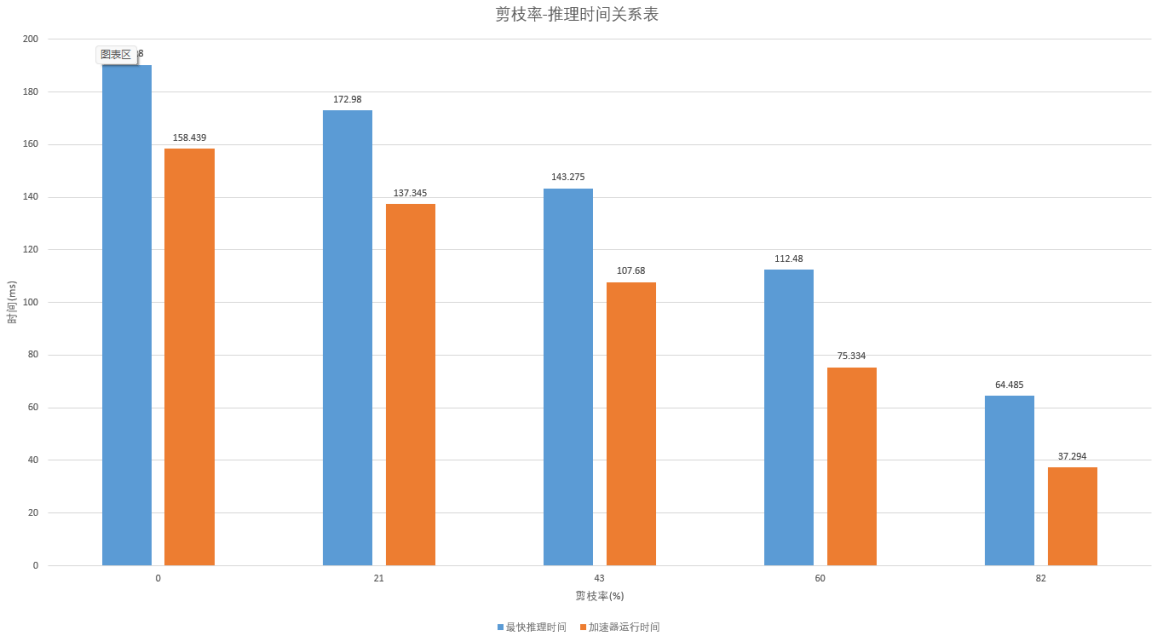


Decode:
像素值单通道填充
产生输入数组

模型优化-模型剪枝率提升效果图

我们最终实现了**82%**的剪枝率，提升效果如下所示

剪枝率(%)	输出层缩放 全量化	GFLOPs	最快推理时间(MS)	加速器运行时间(MS)
0	是	5.09	190.328	158.439
21	是	3.99	172.98	137.345
43	是	2.88	143.275	107.680
60	是	1.97	112.48	75.334
82	是	0.93	64.102	37.294



SDK优化-数据重排优化

1.开启O3优化

```
set(CMAKE_CXX_FLAGS "-march=armv7-a -mfloat-abi=hard -mfpu=neon ${CMAKE_CXX_FLAGS} -O3")
set(CMAKE_C_FLAGS "-march=armv7-a -mfloat-abi=hard -mfpu=neon ${CMAKE_C_FLAGS} -O3" )
```

2.去除无用memset()操作

```
for (int cc = 0; cc < area; cc++)
{
  memset(dout + cc * INPUT_EXTEND_SCALE + c, 0, INPUT_EXTEND_SCALE - c);
}
```

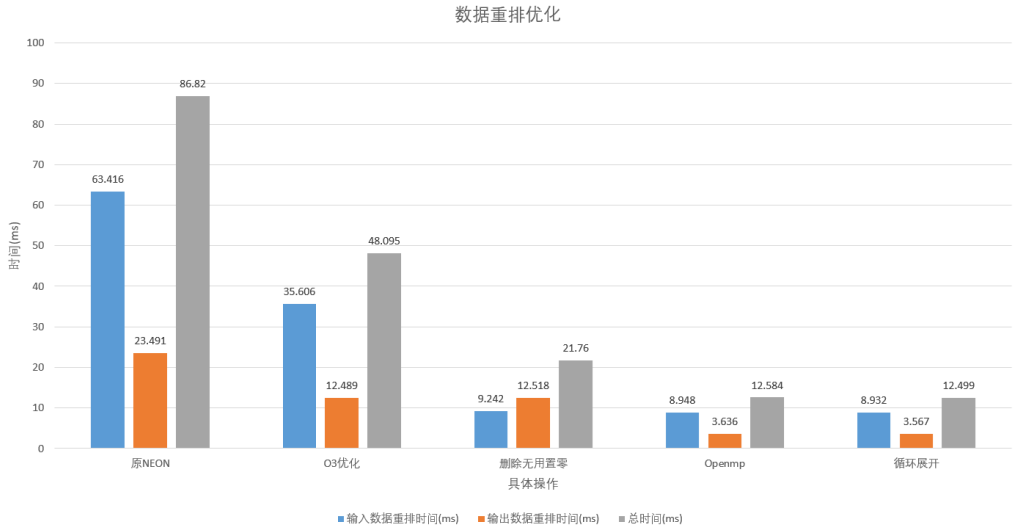
3.使用OPENMP优化

```
#pragma omp parallel for
for (int i = 0; i < up_round(c, INPUT_EXTEND_SCALE); i++)
{
  tran_8((uint8_t *)din + i * area * INPUT_EXTEND_SCALE, (uint8_t *)dout + i * area * INPUT_EXTEND_SCALE, INPUT_EXTEND_SCALE, area);
}
```

4.循环展开

```
#pragma unroll(32)
for (int i = 0; i < up_round(c, INPUT_EXTEND_SCALE); i++)
{
  tran_8((uint8_t *)din + i * area * INPUT_EXTEND_SCALE, (uint8_t *)dout + i * area * INPUT_
}
```

具体操作	输入数据重排时间(MS)	输出数据重排时间(MS)	总时间(MS)
原NEON	63.416	23.491	86.82
O3优化	35.606	12.489	48.095
删除无用置零	9.242	12.518	21.76
OPENMP	8.948	3.636	12.584
循环展开	8.932	3.567	12.499



PaddLite优化-修复PaddLite能耗模式BUG

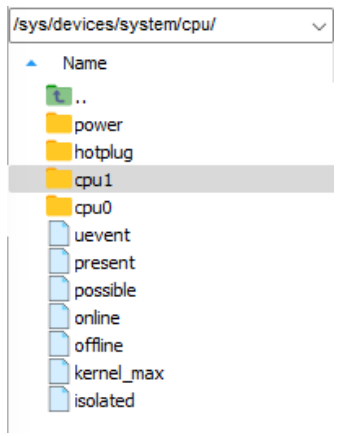
在模型加载时会出现如下报错：查询CPU0状态失败，切换到NO_BIND模式

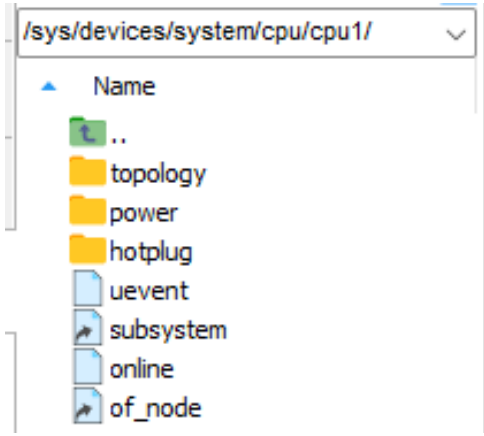
```
[I  5/24 12:50:20.521 ...nna/Paddle-Lite/lite/core/device_info.cc:509 check_cpu_online] Failed to query the online statu
e of CPU id:0
[I  5/24 12:50:20.521 ...nna/Paddle-Lite/lite/core/device_info.cc:514 check_cpu_online] CPU id:0 is offine
[W  5/24 12:50:20.522 ...nna/Paddle-Lite/lite/core/device_info.cc:1352 SetRunMode] Some cores are offline, switch to NO
BIND MODE
```

经过调试，定位PaddleLite绑定核心源码位于device_info.cc文件下

```cpp
bool bind_threads(const std::vector<int> cpu_ids) {
#ifdef ARM_WITH_OMP
  int thread_num = cpu_ids.size();
  omp_set_num_threads(thread_num);
  std::vector<int> ssarets(thread_num, 0);
#pragma omp parallel for
  for (int i = 0; i < thread_num; i++) {
    ssarets[i] = set_sched_affinity(cpu_ids);
  }
  for (int i = 0; i < thread_num; i++) {
    if (ssarets[i] != 0) {
      LOG(ERROR) << "Set cpu affinity failed, core id: " << cpu_ids[i];
      return false;
    }
  }
#else   // ARM_WITH_OMP
  std::vector<int> first_cpu_id;
  first_cpu_id.push_back(cpu_ids[0]);
  int ssaret = set_sched_affinity(first_cpu_id);
  if (ssaret != 0) {
    LOG(ERROR) << "Set cpu affinity failed, core id: " << cpu_ids[0];
    return false;
  }
#endif  // ARM_WITH_OMP
  return true;
}
```

经过分析，PaddleLite通过查询/sys/devices/system/cpu/cpu0目录下的online文件，来确认核心是否可用并绑定。但由于HPS中CPU0为主CPU，默认在线，文件中并未含有online文件。以至于PaddleLite误以为CPU0不可用，将进程绑定在CPU1上，从而限制系统性能





通过修改源码，使其绑定双核心并利用双线程运行，充分利用硬件资源，修改前后推理刷新时间提高了**10ms**。

```
------------------graph begin------
input_organize_time:10.116718ms
fpga_time:37.284771ms
output_organize_time:7.526860ms
------------------graph end------
FPS run:87.175ms
```

```
------------------graph begin------
input_organize_time:10.141111ms
fpga_time:37.325283ms
output_organize_time:3.697364ms
------------------graph end------
FPS run:77.784ms
```

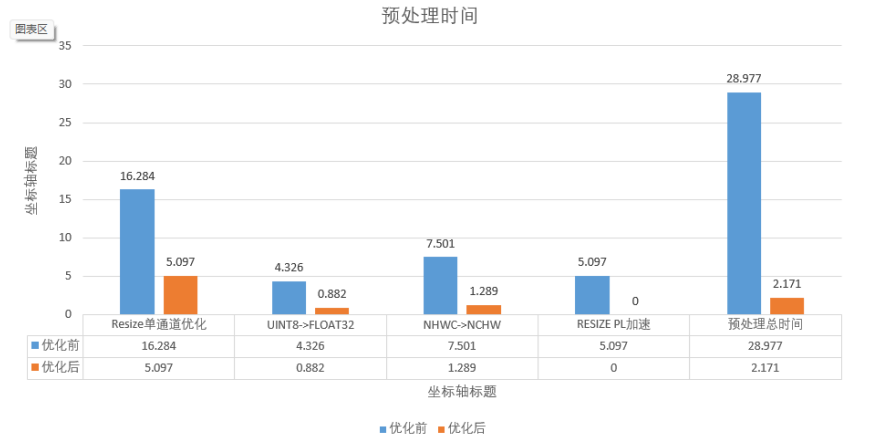
PaddLite优化-预处理优化

主办方提供数据集为**灰度图**，RGB888格式，且PaddleLite官方Demo仅支持三通道输入，将框架输入改为**单通道**有如下好处：

1. dvp_ddr IP输入为单通道，其相邻像素间地址连续，HPS从DDR读取原始图像数据将节省**读取时间**。
2. 图片需要Resize为300*300送入模型推理，单通道图像可减少**Resize**时间。
3. 图像数据需转换为(-1,1)范围内的Tensor格式，即uint8->float32,单通道将减少**转换时间**。
4. NHWC->NCHW通道转换，PaddleLite的输入特征图为**NCHW**格式，需进行格式转换以及归一化操作，单通道可以缩短**通道转换**以及**归一化**时间。

我们通过修改单通道预处理过程，并且FPGA加速Resize，将数据预处理时间从29ms缩短至2ms，如下图所示。

预处理过程	修改前时间(MS)	修改后时间(MS)
RESIZE单通道优化	16.284	5.097
UINT8->FLOAT32	4.326	0.882
NHWC->NCHW	7.501	1.289
RESIZE PL加速	5.097	0
预处理总时间	28.977	2.171



/04

总结与补充

总结

PL侧相关优化

- 重写DVP_DDR,DDR_VGA IP，修改其为单通道输入输出，减少数据传输时间。
- 重写架构，视频推理完全由PL控制，不受HPS性能影响。
- 使用PL对推理过程中的Resize进行加速，完成双线性插值算法的FPGA实现。
- 使用PL设计了PL_Plot IP，利用FPGA完成预测框的绘制，从而可将全部CPU资源用于推理，提升推理刷新速度。
- 使用增量编译、逻辑固化、时序约束、去除无用组件如JTAG等。

模型训练优化

- 剪枝与量化结合训练，加快模型推理速度
- 基于敏感度的剪枝分析，达到81.6%的剪枝率
- 卷积层缩放，进一步减小模型大小
- 卷积连续，算子融合，能够利用CNN加速器加速卷积运算
- 数据增强：增强轮廓特征、水平变换，提高模型识别准确度
- 单通道模型修改，为预处理时间优化铺垫

PS侧相关优化

- 修改PaddleLite架构，使其完成单通道推理，将预处理时间缩短至2ms。
- 修改PaddleLite底层源码，使其能够利用双核心推理，提升推理刷新速度。
- 利用NEON指令集完成数据重排加速，从83ms缩短至13ms。
- 修改Cmake文件，开启O3优化，使用OpenMP优化进程。
- 利用Linux的进程调度功能，使用CPU核心绑定，进程优先级调整，利用脚本文件提升系统推理性能。

补充

PL端数据重排

verilog实现

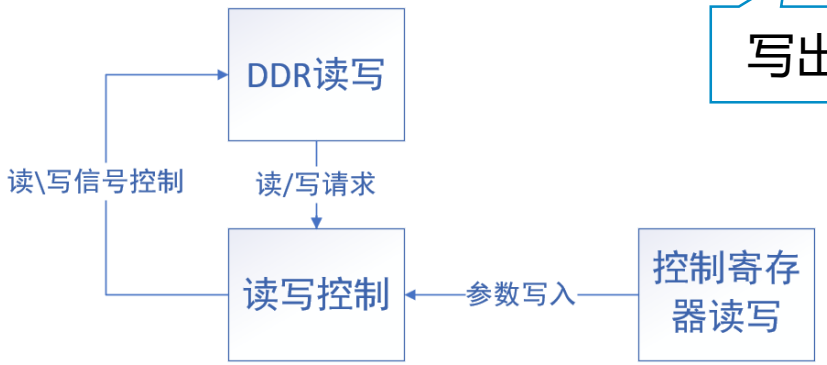
我们的基本思路是在读写控制中设法构建一个**8*8的数据矩阵**，对于该矩阵，数据**按行读入**，之后按**列写出**，以实现重排。

1	2	3	4	5	6	7	8
M+1	M+2	M+3	M+4	M+5	M+6	M+7	M+8
2M+1	2M+2	2M+3	2M+4	2M+5	2M+6	2M+7	2M+8
3M+1	3M+2	3M+3	3M+4	3M+5	3M+6	3M+7	3M+8
4M+1	4M+2	4M+3	4M+4	4M+5	4M+6	4M+7	4M+8
5M+1	5M+2	5M+3	5M+4	5M+5	5M+6	5M+7	5M+8
6M+1	6M+2	6M+3	6M+4	6M+5	6M+6	6M+7	6M+8
7M+1	7M+2	7M+3	7M+4	7M+5	7M+6	7M+7	7M+8

读入

1	2	3	4	5	6	7	8
M+1	M+2	M+3	M+4	M+5	M+6	M+7	M+8
2M+1	2M+2	2M+3	2M+4	2M+5	2M+6	2M+7	2M+8
3M+1	3M+2	3M+3	3M+4	3M+5	3M+6	3M+7	3M+8
4M+1	4M+2	4M+3	4M+4	4M+5	4M+6	4M+7	4M+8
5M+1	5M+2	5M+3	5M+4	5M+5	5M+6	5M+7	5M+8
6M+1	6M+2	6M+3	6M+4	6M+5	6M+6	6M+7	6M+8
7M+1	7M+2	7M+3	7M+4	7M+5	7M+6	7M+7	7M+8

写出



OpenCL实现

OpenCL的实现主要起验证作用，由于经过分析，我们认为PL端数据重排并不能比ARM侧使用NEON指令集加速，因此最终并未选用PL数据重排

```
double time1 = getCurrentTimestamp();
status = clEnqueueWriteBuffer(queue, dev_src, CL_TRUE, 0, sizeof(cl_int)*src_size, src, 0, NULL, NULL);

status = clEnqueueWriteBuffer(queue, dev_in_c, CL_TRUE, 0, sizeof(cl_int), in_c, 0, NULL, NULL);
status = clEnqueueWriteBuffer(queue, dev_in_h, CL_TRUE, 0, sizeof(cl_int), in_h, 0, NULL, NULL);
status = clEnqueueWriteBuffer(queue, dev_in_w, CL_TRUE, 0, sizeof(cl_int), in_w, 0, NULL, NULL);
```

```
!============================================================================
! The report below may be inaccurate. A more comprehensive
! resource usage report can be found at mnist_simple/reports/report.html
!============================================================================

+--------------------------------------------------------------------+
; Estimated Resource Usage Summary                                   ;
+----------------------------------------+---------------------------+
; Resource                               + Usage                     ;
+----------------------------------------+---------------------------+
; Logic utilization                      ;   13%                     ;
; ALUTs                                  ;    8%                     ;
; Dedicated logic registers              ;    6%                     ;
; Memory blocks                          ;   15%                     ;
; DSP blocks                             ;    5%                     ;
+----------------------------------------+---------------------------+
```

```
total_time is:1.289770 ms
root@socfpga:~/zhuanzhi#
```

感谢观看

汇报结束
请评委专家提问！

A组 CICC1577
冰糖葫芦儿