第六届
全国大学生集成电路创新创业大赛

参赛杯赛:	景嘉微杯
作品名称:	一种基于分层聚类算法的图片超分辨率IP
队伍编号:	CICC2710
团队名称:	狸猫换太子

2022年8月18日

摘要

图像上采样是指根据同一场景的一张或多张低分辨率图像，使用图像处理和机器学习等方法，构建高分辨率图像的技术。在医学图像领域，图像分辨率受限医学设备的分辨能力；在遥感领域，期望通过图像超分辨重建提升遥感精度；在GPU图形处理领域，图像上采样技术能够减轻像素引擎的负载，提升芯片图形性能。这些特征使得图像上采样技术成为近年来的研究热点。

本报告介绍了基于分层聚类算法的上采样IP。针对图像上采样这一赛题目标，该IP使用基于Bayes估计的机器学习算法，能够将输入的一张960*540的rgb图片，上采样为3840x2160的图片像素矩阵。基于该IP，我们在 Zynq UltraScale+ MPSoC ZCU106 Evaluation Kit（后称ZCU106）上，实现了通过存储设备获取原始图片数据存储到DDR，IP从DDR读取数据进行上采样，将上采样后的图片回传上位机并通过开发板的HDMI接口输出显示这一完整的流程。在赛题额外要求的软件并行处理实现领域，我们可以灵活设置线程数，并使得各线程的负载均衡。除赛题要求外，我们还利用ZCU106的ARM核，开发了完全在板上完成赛题方任务的功能。同时开发了gstreamer插件，能够对视频进行上采样处理。

在算法选型上，我们采用了NBSRF（Naive Bayes Super-Resolution Forest），该算法基于机器学习对双三次上采样的结果进行修正。我们对NBSRF进行了C语言实现，编写了量化的、符合硬件处理方式的C model，并按照赛题要求实现了并行处理的功能。在景嘉微官方提供的测试集上，经过量化的算法效果如表A所示：

Table A 赛题要求总结——图像质量

指标	基准值	我们的IP
psnr（越高越好）	28.360	32.153（+3.793）
ssim （越高越好）	0.800	0.876（+0.076）
lpips （越低越好）	0.284	0.200（-0.084）

在硬件实现上，我们采用Verilog编写了符合AXI Stream协议接口的IP。IP设计上采用流式处理，不设状态寄存器。IP的综合基于Vivado 2019.2，采用的硬件平台为ZCU106。经过优化，IP单独综合的最高频率可达到454.7MHz，等效硬件极限4K帧率为175.4fps。相较于初版IP，最终版IP的时钟频率提高了200.4%；处理延时由11个行缓存降低至8个；LUT用量由689.5K降至43.6K，降低了93.7%；REG用量由345.4K降至63.1K，降低了81.7%；DSP用量由2173降至733个，降低66.3%，如表B所示。

在系统平台上，我们基于petalinux 2019.2编译部署了片上Linux，片上系统可以通过ETH、USB、SD card多种介质与上位机交互；采用bperez77/xilix_axidma开源库驱动DMA，实现内存与IP的数据交互；采用Xilinx官方提供的VCU TRD设计驱动片上HDMI sink；结合GStreamer开源库，实现上采样数据推流到HDMI sink进行展示。最后，为了简化用户交互，我们还基于QT开发了一套上采样GUI界面。

Table B 赛题要求总结

指标	我们的IP	指标	我们的IP
IP时钟频率	434.7 MHz	LUT	43.6K
极限帧率	167.7 fps	REG	63.1K
延时	8 个行缓存	DSP	733

和其他图像上采样IP相比，本报告介绍的IP在上采样图像的质量与逻辑资源两方面做了很好的平衡。与传统的插值算法IP相比，本IP以一定的硬件资源为代价获得了明显的图像效果的提升；而与神经网络算法的IP相比，本IP的逻辑资源用量显著降低。在已发表的同类型算法的IP领域，本IP在图像质量几乎相同的情况下，资源用量显著减少。以IP为基础我们搭建了完整的系统框架，功能更加完善，也方便项目的迁移与进一步开发。

目录

1. 算法设计

1.1. 算法调研

在图像插值领域，如果从插值思想的角度划分，图像插值算法可大致分为两类：一类是基于局部数据的插值算法，另一类是基于样例学习的插值算法。基于局部数据的插值算法基于本地数据，利用固定的数学方法进行插值。传统的最邻近插值、双线性插值、双三次插值等方法对于图像的非边缘区域已经取得了很好的效果，但是无法有效处理图像边缘的插值问题。后续的各种优化都基于此展开，基于小波系数的方法，基于边缘信息的方法等方法都在此问题上给出了一定的改进。但是随着机器学习技术的发展，传统插值算法插值方式固定的缺点逐渐暴露，在和机器学习算法的对比中处于绝对的下风。

在机器学习插值算法领域，最开始发展起来的是内部学习的算法。该算法根据图像中跨区域的自相似性来回归高分辨率图像。但是对于相似性的判定引入额外的计算需求，后续的改进使得计算量有一定的减少，但是仍然没有测地解决这一问题。进一步发展的机器学习算法将图像上采样问题归结为低分辨率图像到高分辨率图像的映射。虽然映射是非线性的，但是能够在局部通过线性近似非线性函数，以此来取得更好的效果。

如今的图像上采样算法大都需要神经网络，不符合竞赛要求。我们考虑使用基于映射到机器学习算法，并对广受关注的五种机器学习算法性能进行了比较。

Table 1.1 机器学习算法上采样效果比较

算法	PSNR	SSIM	LPIPS
Forest	28.1	0.871	0.22
GR	31.9	0.870	0.25
ANR	32.1	0.874	0.23
A+	32.2	0.876	0.20
NBSRF	32.2	0.877	0.21

综合考虑三项指标，A+算法与NBSRF算法表现最为突出，但在NBSRF算法论文中对二者的运行时间进行了比较（Fig.1.1），可以看出NBSRF算法用时显著少于A+算法，在硬件实现上NBSRF算法对资源的需求量更少，更容易达到更高的速率，因此我们在使用机器学习算法进行优化时使用该算法为基础。

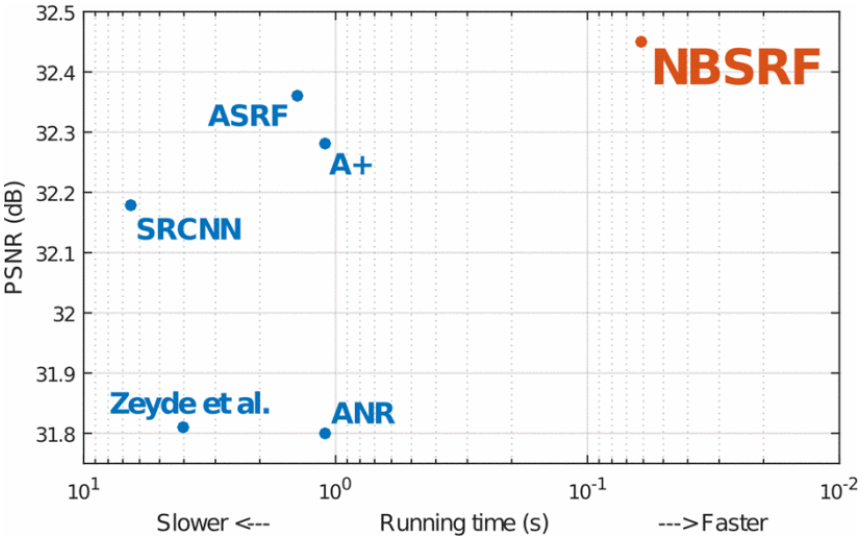

Fig. 1.1 算法用时对比图

首先我们实现了双三次插值，针对插值算法无法有效处理边缘的问题，我们选取了上文所述的贝叶斯算法进行修正。

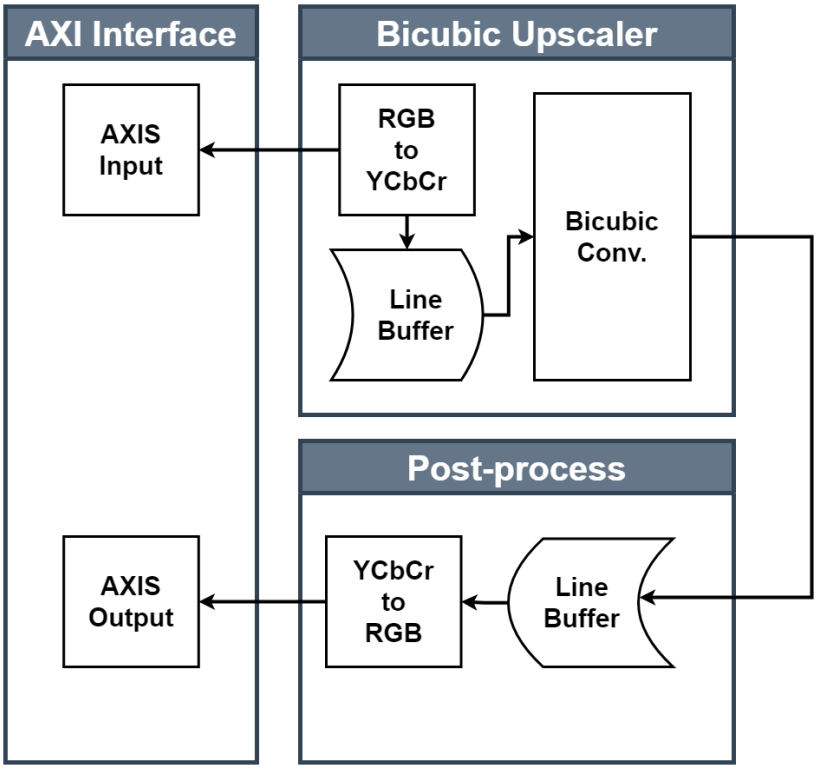

Fig. 1.2 双三次IP系统

贝叶斯算法的核心是在双三次插值的系统上增加贝叶斯回归校正通路，校正双三次插值的图像以获得高分辨率的图像。

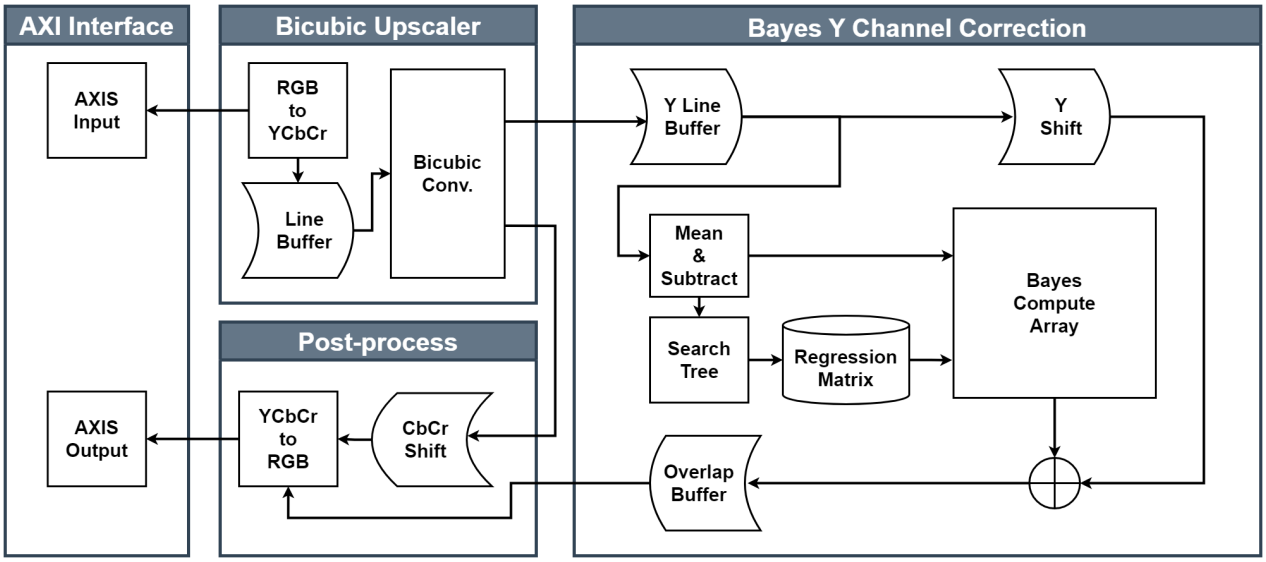

Fig. 1.3 贝叶斯 IP 系统

1. 我们的实现的算法上采样效果稳定良好，明显优于传统改进算法，传统改进算法即使使用更多的资源也无法达到如此稳定良好的上采样效果；
2. 我们的算法泛用性好，对各种方式得到的低分辨率图像上采样效果均稳定良好，而传统改进插值算法的优化效果不稳定，比如针对均值降采样方式优化的插值算法在上采样双三次降采样的图片时效果无法保障。

1.2. 算法模型思路

贝叶斯算法首先对输入低分辨率图像进行双三次上采样，然后对上采样的图像进行回归校正，最终输出高分辨率图像。
在回归校正阶段，首先对上采样图像的 Y 通道提取特征，然后基于分层聚类树搜索索引回归矩阵，最终进行回归校正。

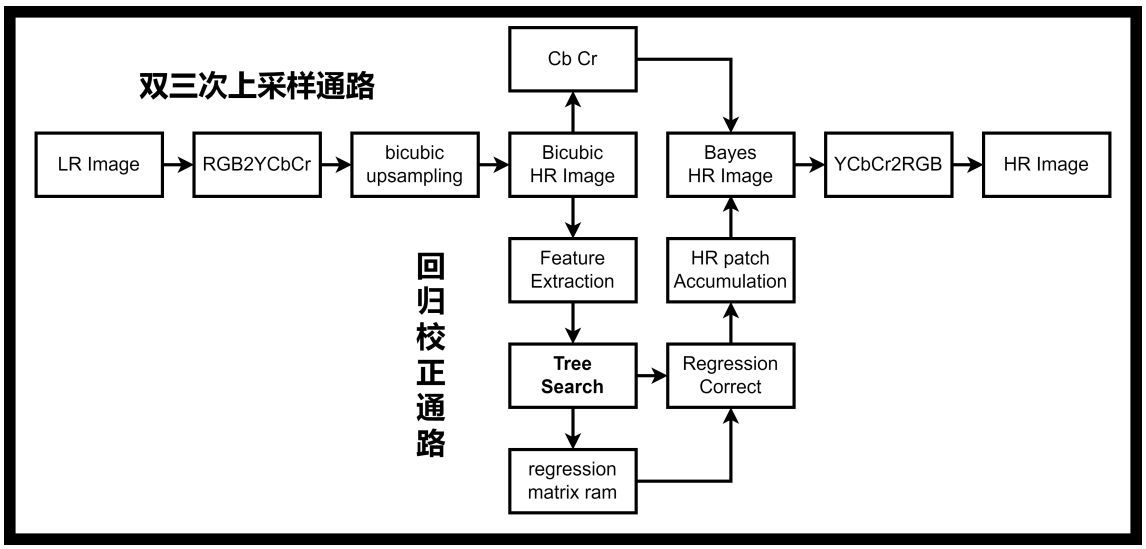

Fig. 1.4 贝叶斯算法流程

1.2.1. 算法概述

算法基于Jordi Salvador等人的文章Naive Bayes Super-Resolution Forest，总体包括训练和生成高分辨率图像两个部分。为便于算法的硬件实现，对算法进行了调整与简化，降低了数据的存储空间，同时C模块与Verilog IP模块基于训练好的参数，只实现生成高分辨率图像部分。

在训练部分，需要将图像分割成合适大小的patch，对低分辨率的patch对应的高分辨率patch进行初步的估计，根据估计的高分辨率patch和真正的高分辨率patch，找到他们之间的变换关系，即找到合适的系数矩阵。算法通过将低分辨率空间分层聚类，然后学习到高分辨率的局部线性映射来提供估计的patch和高分辨率patch之间的直接映射关系。映射关系是一个矫正层，用于对高分辨率图像修正。在生成高分辨率图像部分，同样将低分辨率图像分成不同的patch，通过对patch分层聚类（寻找与patch对应的向量的线性度最高的向量，得到所属的类），来使用相对应的系数矩阵矫正估计的高分辨率patch，就可以构造出新的高分辨率图像。

1.2.2. 模型训练

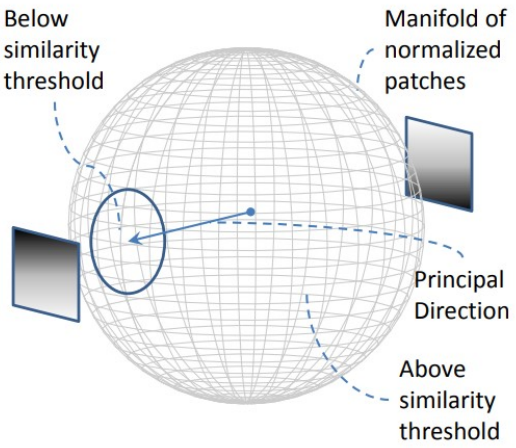

Fig. 1.5 超球面分布示意图

模型主要考虑patch中的差异值对修正结果造成的影响。为突出差异值，将patch中的像素值减去patch所有像素的均值，得到阶段性的结果。把结果线性排列为一个向量，对向量L2正则化处理，所有的向量就分布在一个超球面上（Fig. 1.2.2）。相似的patch的向量的线性度高，对应的变换关系也是高度吻合的。根据实际使用的系数矩阵的数目，设置聚类中心，为线性度设置一个阈值，将一定范围内的向量聚为一类，就可以使用同一个系数矩阵进行修正。

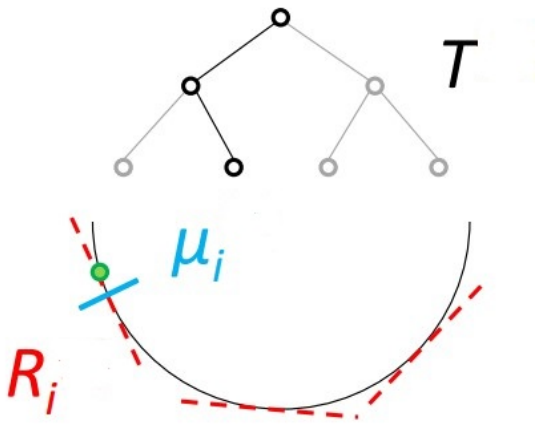

Fig. 1.6 二叉树聚类示意图

为提升聚类的准确性，得到更好的结果，使用二叉树结构进行分层聚类。首先将大量的低分辨率patch随机分为两类，将每一类的中心向量作为聚类中心，计算patch向量与两个中心向量的线性度，将所有patch重新分配到线性度更高的两个类中，完成第一层的二叉树聚类工作。然后在每个类中分别进行聚类工作，得到下一层的聚类结果。不断重复上述过程，直到得到的类的数量满足要求。过程中的中心向量（μ）保存作为生成高分辨率图像时的分类依据。根据最终的类的中心的低分辨率patch和其对应的高分辨率patch，计算系数矩阵R，作为最终的训练结果。

主要参数如下：

```
scale = 2;          % 上采样因子为2
model = 'bicubic';  % 初步估计的方式，双三次插值
ntrees = 1;         % 二叉树的个数为1
nclusters = 128;    % 叶节点数目128个（128个类）
psize = 3;          % patch大小3*3
overlap = 1;        % 相邻patch的距离
nsamples = 1500;    % 每个子节点的样本数量为1500
```

1.2.3. 生成高分辨率图像

使用C语言实现算法的生成高分辨率图像部分，主要步骤如下：

1. 由于算法只针对Y通道，首先对图像进行信号通道的转换，将RGB空间转换为YCbCr空间，并将每个通道的值分离。
2. 使用经典的双三次插值方法，得到初步的高分辨率图像。双三次插值的采样因子为2，即得到两倍放大的图像。
3. 针对人眼对Y通道更为敏感的特点，对Y通道数据单独进行基于聚类算法的回归矫正。
   3.1. 以3（psize）为边长、1（overlap）为步长，将Y通道数据划分为多个patch，并将patch的像素数据线性排列，转化为向量形式。
   3.2. 将每个向量的数据减去自身数据的均值，得到差异值向量。
   3.3. 将差异值向量输入预先构建的二叉树进行搜索。二叉树的每条边对应一个向量，将该向量与差异值向量相乘，绝对值大的边为选择的边。重复搜索过程直至到达叶节点，得到对应的叶节点的编号。
   3.4. 根据编号选择预先设计的回归矩阵参数，将差异值向量与回归矩阵相乘，得到矫正向量。
   3.5. 将矫正向量变换为Y通道矫正数据，与初步的高分辨率图像的Y通道数据相加，得到最终的结果。
4. 重复第二步与第三步，得到四倍放大的图像。
5. 将YCbCr空间转回RGB空间，得到最终的结果。

1.3. 算法实现与优化

我们按照算法流程搭建了硬件框架，并对硬件针对性地进行了大量的优化。

- 逻辑资源
  - 算法硬件友好化：消除非移位除法，消除归一化操作

    - 算法优化：合理设置图像块尺寸
    - 算法量化：量化参数
    - 硬件优化：SIMD乘法
- 存储
    - 算法优化：减小图像块尺寸
    - 硬件优化：分层存储参数
- 吞吐率
    - 硬件优化：4 倍并行架构
- 行延时
    - 算法优化：合理选取图像块尺寸
    - 硬件优化：数据复用
- 时钟频率
    - 流水线

1.3.1. 逻辑资源优化

首先我们对IP的逻辑资源进行了优化。

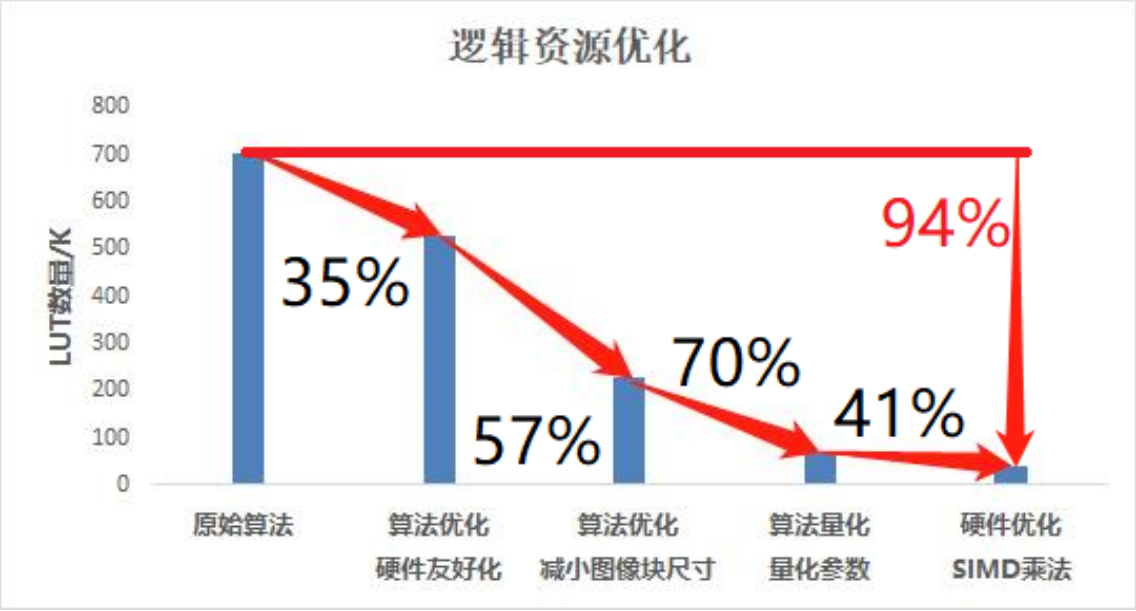

Fig. 1.7 逻辑资源优化

1.3.1.1. 算法硬件友好化

> 针对算法中硬件不友好的操作，我们合理设计消除了非移位除法和归一化操作，逻辑资源降低了35%。

消除L2正则化

在模型训练阶段需要确定不同向量的分布，而L2正则化能够消除向量的模这一无关因素的影响，是必须的。而在推断阶段，虽然L2正则化能够保持推断阶段搜索数据与训练阶段搜索数据完全一致，但是仔细考虑二叉树搜索的过程：每次搜索都是用当前的向量分别乘两个已知的分支向量，通过比较二者的大小来确定结

果。此时如果当前向量不进行L2正则化，两个结果都是扩大了相同的倍数，但是不影响相对大小关系，进而不影响最终的搜索结果。L2正则化的消除能够直接进行搜索，既减少了资源的开销，也加速了运行效率。

消除对9的除法

在算法计算patch均值阶段，每个patch有9个元素，均值的计算也就意味着需要将9个元素的和除9。同时由于算法本身overlap的设计，绝大部分像素值被矫正了9次，因此最终矫正值需要除9求得9次矫正的均值作为最终的结果。考虑到本次竞赛不允许使用出发IP，对除9的处理是困难的。同样考虑算法的流程，patch元素与均值做差之后乘R矩阵进行矫正得到一个patch的矫正结果，最终9个矫正结果相加再除9得到最终的结果。如果我们把原始的R矩阵的值除81，就取代了均值计算与overlap叠加计算中除9的步骤，由于R矩阵是固定参数的矩阵，其除以81的操作可以在IP外部提前进行而不会对IP本身产生影响。通过这种变换方式，IP本身就不再需要除法，算法对硬件的友好程度有了明显的提高。

1.3.1.2. 算法优化——减小图像块尺寸

> 针对原始算法图像块过大导致开销过大的问题，我们进行量化分析、合理地设置图像块尺寸；逻辑资源降低57%。

在进行回归校正时，需要切割图像块对图像块进行校正，图像块的尺寸影响了计算的算力需求和算法的效果。测量设置不同图像块尺寸时的上采样图像质量，在图像块尺寸等于3时，计算的算力需求以及算法的上采样效果都表现良好。

当图像块尺寸大于等于3时，算法均根据不同块的特征决定图像块是角落块还是边缘块或者是一般块，因此算法效果均表现良好。当图像块尺寸等于2时，算法难以根据图像特征区分图像块，因此算法效果骤降。

Table 1.2 图像块尺寸与图像质量的关系

图像块像素尺寸	PSNR
2	31.68
3	32.15
4	32.17
5	32.18
6	32.22

1.3.1.3. 算法量化——量化参数

> 针对原始算法精度过高算力要求过大的问题，我们量化算法对参数进行了量化，逻辑资源降低了70%。

二叉树搜索量化　　在二叉树搜索的过程中，每次搜索均使用一个向量乘两个分支向量，取绝对值大的结果所在的分支作为进一步搜索的分支。在这个过程中，与精确的绝对值相比，二者之间的相对大小关系是更重要的内容。而在乘加过程中，起主导作用的是大数据与大数据相乘的结果，因此在搜索过程中可以忽略掉低位数据

对结果的微小影响，仅采用高位数据运算。根据对不同精度量化数据的测量结果，最终采用7bits×7bits作为树搜索过程中乘法的位宽。

回归矩阵乘法量化　　算法原生的回归矩阵数据为浮点数，在硬件中与回归矩阵相乘的向量位宽是16bits，而在最终输出的RGB结果中只保留了8bits的数据。数据位宽的不同导致中间过程中的低位数据难以对最终的结果产生影响，因此也可以采用只保留高位数据计算结果的思路。在浮点数的量化上，与之相乘的是一组YCbCr数据与均值的差，YCbCr等效为八位，与均值作差后绝大部分数据在7位以内，因此回归矩阵采用7位就能够保留最终结果对整数位的影响。 对于实际的16bits的YCbCr数据，经过对不同精度的结果的对比，最终采用9bits数据，能够在不明显改变结果的情况下降低数据的位宽。

Table 1.3 回归矩阵参数精度与图像质量的关系

R bits	Y bits	PSNR 增益
6	9	0.19
7	8	0.39
7	9	0.45
7	10	0.46
7	11	0.46
8	9	0.45
8	10	0.46
9	9	0.46
9	10	0.48

1.3.1.4. 硬件优化——SIMD乘法

> 针对bayes核中算力要求高的问题，我们设计了基于SIMD的乘法，逻辑资源降低了41%。

输入的差异值向量 1*9 与回归矩阵 9*9 矩阵乘，相当于同一个差异值向量同时与9个不同的回归向量相乘。因此我们可以考虑使用DSP实现SIMD 乘法提高硬件效率。

我们最终使用 {9bit input} * {{7bit weight_1},{10bit},{7bit weight_2} 的模式进行计算，DSP的输出同时包含两个输出数据。

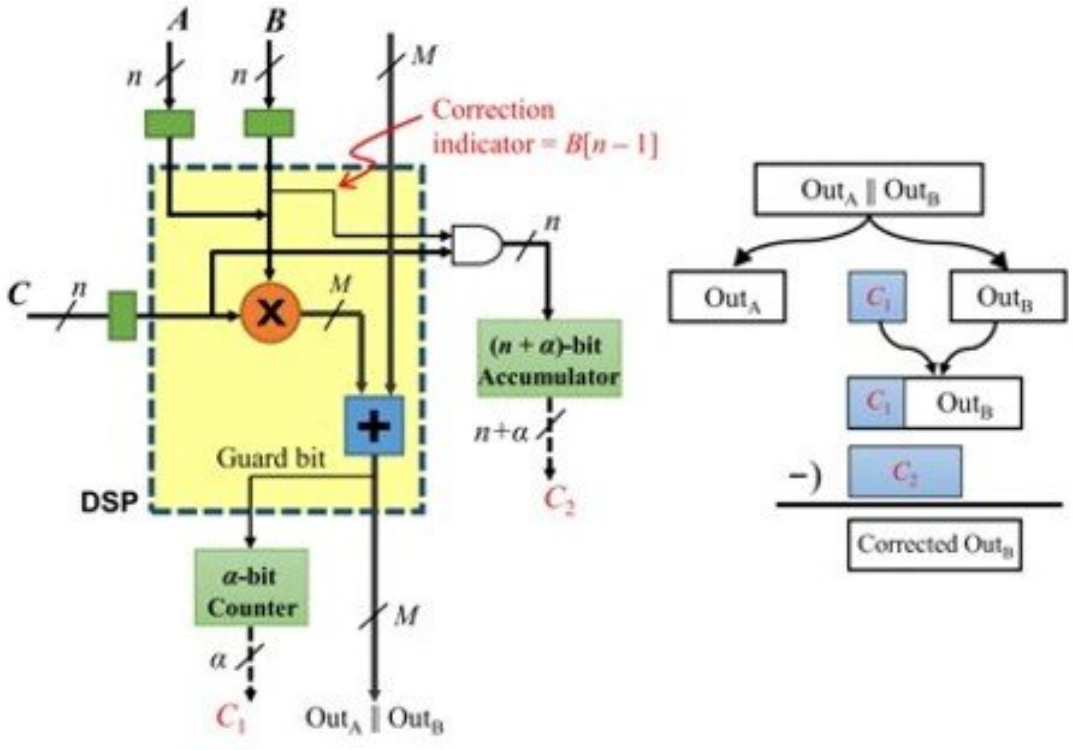

Fig. 1.8 基于DSP Block的SIMD乘法

1.3.2. 存储优化

然后我们对IP的存储进行了优化。

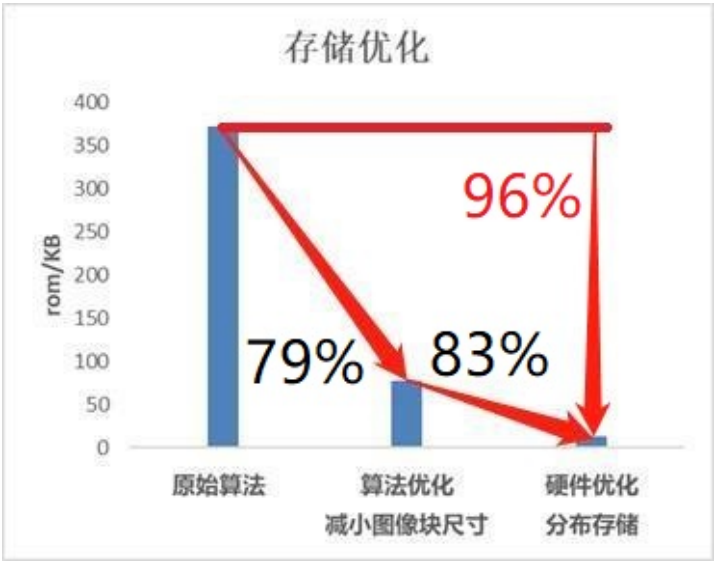

Fig. 1.9 存储优化

1.3.2.1. 算法优化——减小图像块尺寸

> 针对原始算法图像块过大导致存储开销大的问题，我们使用上文所述的方式合理选取图像块尺寸，存储降低了79%。

1.3.2.2. 硬件优化——分布存储提升访存效率

> 针对回归树搜索阶段存储需求大的特点，我们设计了分布存储分层存储参数，存储降低了83%；

如果将所有节点的评分向量放在一起，当某一差异值向量访问其中一层节点时，由于输出带宽有限，其他差异值向量不能访问该层节点，也不能访问其他层的节点。

由于某一层节点在一个时刻只会被一个差异值向量访问，因此考虑使用分布存储将每一层的评分向量存储在独立的存储器中。使用分布存储，同一时刻，七层的节点同时被访问，访存效率达到最大值。

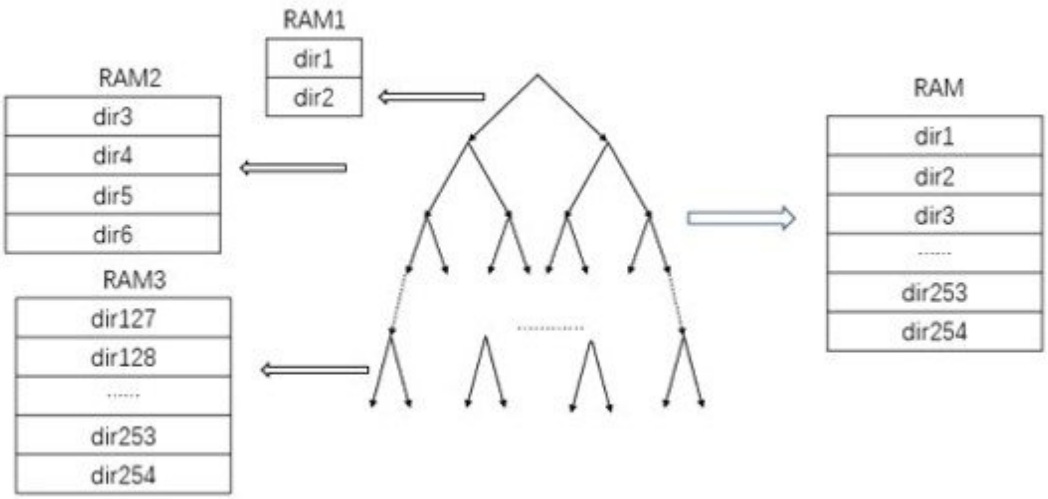

Fig. 1.10 分层存储参数矩阵

1.3.3. 其他优化

1.3.3.1. 吞吐率优化——并行架构

> 受益于合理的架构设计，我们很方便地对bayes核四倍并行，最终吞吐率和帧率提升了 4 倍

1.3.3.2. 行延时优化

算法优化——降低图像块尺寸

> 针对原始算法图像块过大导致预存过多行缓存的问题，我们使用上文所述的方式合理选取图像块尺寸，行延时从11降低至9；

硬件优化——数据复用

> 通过合理地进行了数据复用，行延时从9降低至8；

1.3.3.3. 时钟频率优化——流水线

> 我们使用流水线技术，将时钟频率提升了200.4%

1.4. IP架构

1.4.1. IP 架构

1.4.1.1. 概述

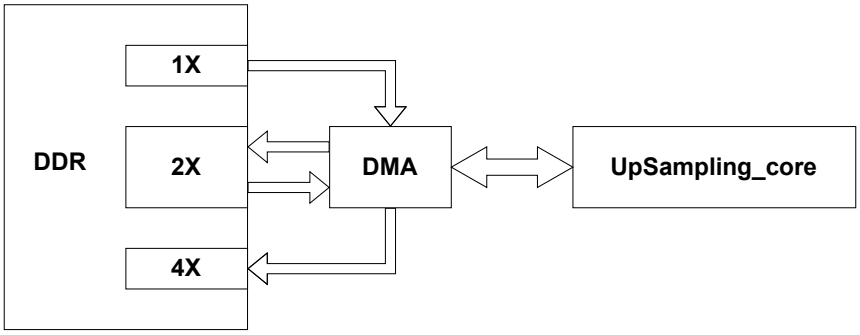

Fig. 1.11 IP 微架构

IP 由DDR，DMA，UpSampling_core组成。UpSampling_core 是 `2X` 上采样核，可以将低分辨率图像上采样至原来的2倍大小。 DDR 输入的`1X`低分辨率图像，两倍放大的`2X`中间过程图像，四倍放大的`4X`高分辨率图像。DMA 是 UpSampling_core 与 DDR 之间的数据传输模块。

Zynq开始调度IP进行超分辨率后，IP 开始工作，IP 对低分辨率图片进行两次放大得到`4X`高分辨率图片。

1.4.1.2. 数据流

在一张图片的处理流程中：

1. DMA 从 DDR `1X` 区域搬运低分辨率图片至 `Up_Sampling_core` 进行两倍放大处理；与此同时，DDR 从 `Up_Sampling_core` 中将放大后的中间过程图片数据搬运至 DDR `2X` 区域。
2. DMA 从 DDR `2X` 区域搬运中间图像左上部分 960 * 540 大小的图像数据搬运至 `Up_Sampling_core` 进行两倍放大处理; 与此同时，DDR 从 `Up_Sampling_core` 中将放大后的 `4X` 高分辨率图像搬运至 DDR `4X` 区域对应部分
3. 重复操作2，依次处理中间图像右上、左下、右下部分数据。
4. 处理结束后，DDR `4X` 区域存储`4X`上采样完成后的高分辨率图像。

1.4.2. UpSampling_core

1.4.2.1. 概述

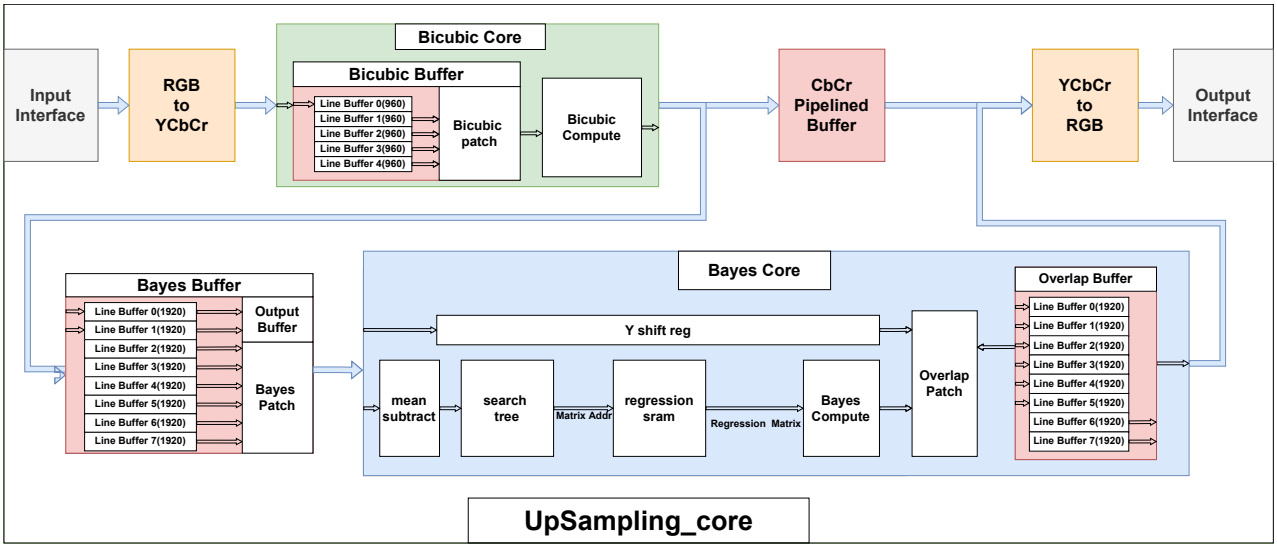

Fig. 1.12 UpSampling_core 架构图

UpSampling_core 的功能是实现两倍上采样。 UpSampling_core 由输入接口 `Input Interface` , 输入数据通道转换 `RGB to YCbCr` ,双三次上采样核心 `Bicubic Core` , Y 通道数据缓存 `Bayes Buffer` ,贝叶斯校正核心 `Bayes Core` , CbCr 通道数据缓存 `buffer CbCr Pipelined Buffer` , Y通道校正数据缓存 `Overlap Buffer` , 输出数据通道转换 `YCbCr to RGB` , 输出接口 `Output Interface` 组成。

1.4.2.2. 数据通路

1. `Input Interface` 接收低分辨率图像数据，然后完成32bit to 24bit 数据位宽转换;
2. `RGB to YCbCr` 接收输入数据 rgb 通道像素，转换成 ycbcr 通道像素;
3. `Bicubic Core` 进行双三次上采样，将产生的 cbcr 通道数据输出至 `CbCr Pipelined Buffer` ，将产生的 y 通道数据输出值 `Bayes Buffer` ;
4. `Bayes Buffer` 产生Y通道贝叶斯像素块 `Bayes Patch` 和Y通道缓存像素块输出到 `Bayes Core` 进行Y通道校正;
5. `Bayes Core` 将 y 通道数据校正后存储到 `Overlap Buffer` ， `Overlap Buffer` 输出校正后的数据;
6. `YCbCr to RGB` 接收 `CbCr Pipelined Buffer` 和 `Overlap Buffer` 的数据，将输出 YCbCr 通道数据转换成 rgb 通道数据并将输出 rgb 通道数据输出至 `Output Interface` ;
7. `Output Interface` 接收输出高分辨率图像数据，然后完成 96bit to 128bit 数据位宽转换。

1.4.2.3. 模块功能

- `Input Interface` : 进行位宽转换并协调输入与UpSampling_core的关系;
- `RGB to YCbCr` : 将输入数据 RGB 通道像素转换成 YCbCr 通道像素;
- `Bicubic Core` : 计算双三次上采样;
- `Bayes Core` : 基于分层聚类算法对 Y 通道数据进行修正;
- `CbCr Pipelined Buffer` : 存储输出 CbCr 通道数据;
- `Overlap Buffer` : 存储输出 Y 通道数据;
- `YCbCr to RGB` : 将输出数据 YCbCr 通道像素转换成 RGB 通道像素。

1.4.3. Bicubic Core

1.4.3.1. 概述

`Bicubic Core` 输入低分辨率图像进行双三次上采样。

`Bicubic Core` 由 `Bicubic buffer` 、 `Bicubic Patch` 以及`Bicubic Compute` 组成。

1.4.3.2. 数据通路

1. `Bicubic buffer` 接收输入低分辨率图像 YCbCr 通道数据流，在缓存一定数据后，输出到 `Bicubic Patch` ；
2. `Bicubic Patch` 产生像素块输出到双三次计算模块 `Bicubic Compute` ；
3. `Bicubic Compute` 计算双三次上采样结果并输出。

1.4.3.3. 模块功能

- `Bicubic buffer`： 由 5 组三通道buffer组成，其中四组buffer输出数据至 `Bicubic Patch` ，一组buffer用于接收输入数据。
- `Bicubic Patch`: 产生 4 * 4 大小的像素块，使用移位寄存的方式进行数据复用。
- `Bicubic Compute`： 产生双三次上采样的结果，通过固定双三次上采样参数的方式进行计算。

1.4.4. Bayes Core

1.4.4.1. 概述

Bayes Core 使用分层聚类的方法对 Y 通道数据进行修正。
Bayes Core 由 `Mean Subtract` 、 `Search Tree` 、 `Regreesion Sram` 、 `Bayes Compute` 、 `Y Shift` 、 `Overlap Patch` 、 `Overlap Buffer` 组成。

1.4.4.2. 数据通路

1. `Bayes Core` 接收 `Bayes Buffer` 产生的Y通道贝叶斯像素块 `Bayes Patch` 和Y通道缓存像素块，将Y通道贝叶斯像素块 `Bayes Patch` 导入 `Mean Subtract` 模块开始进行修正，将Y通道缓存像素块导入 `Y Shift` 模块缓存；
2. 准备 Y 通道校正数据
    1. 差异值向量校正
        1. `Mean Subtract` 计算贝叶斯像素块减去其平均值的结果，得到差异值向量并输出到 `Search Tree` ；
        2. `Search Tree` 将差异值向量与评分向量向量乘得到评分，使用得到的评分进行回归树树搜索，最终得到回归矩阵的地址索引；
        3. 使用回归矩阵的地址索引访问回归矩阵存储器 `Regression Sram` 得到回归矩阵输出值 `Bayes Compute` ；

    4. `Bayes Compute` 将差异值向量与回归矩阵矩阵乘得到校正后的差异值向量输出到 `Overalp Patch`；
  2. 原始数据缓存
    1. `Y Shift` 模块缓存 Y 通道缓存像素块输出到 `Overalp Patch`；
3. `Overlap Patch` 接收校正后的差异值向量与缓存的 Y 通道数据加和得到校正后的Y通道数据并输出至 `Overlap Buffer` ；
4. `Overlap Buffer` 接收校正 Y 通道数据，接收一定数据量后输出数据到输出数据通道转换模块 `YCbCr to RGB`

1.4.4.3. 模块功能

- `Mean Subtract`：接收 3 * 3 大小的像素块，用像素块原始数据减去9个像素的平均值得到差异值向量 3 * 3 像素块。
- `Search tree`：基于分层聚类算法计算评分，使用得到的评分进行回归树树搜索，最终得到回归矩阵的地址索引。
- `Regression sram`：使用树搜索得到的地址访问回归矩阵存储器得到回归矩阵。
- `Bayes Compute`：将回归矩阵和差异值向量做矩阵运算得到校正后的差异值向量。
- `Overlap Patch`：将校正后的差异值向量和原始数据相加校正后的数据。
- `Overlap Buffer` 缓存校正后的 Y 通道数据。

2. 实现函数说明

2.1. Structs

bmp图片包含图像的各项信息，使用结构体存储bmp图片便于信息的分类与使用。在bmp文件格式中，像素信息顺序存储，而在处理数据时，按照行列索引处理像素信息更为直观方便，因此定义矩阵结构体存储图像像素。

```
typedef struct _Bitmap
{
    BITMAPFILEHEADER bmfh; BITMAPINFOHEADER bmih;
    int width; int height; int imageSize;
    int widthStep;         BYTE* imageData;
}   Bitmap;
```

- 图像结构体：
  存储bmp的文件头、信息头、图像数据，关键信息。虽然信息头中包含宽度、高度等信息，直接在结构体中定义能够方便使用。

```
typedef struct int_matrix_size
{
  int **buf; size_t row; size_t col;
} int_buffer;
```

- 矩阵结构体：
  包含存储像素信息的二维数组，与二维数组的行数和列数。

2.2. Functions

```
int ReadBitmap(char* path, Bitmap* bmp)
```

- 将指定路径的bmp图像读入到Bitmap结构体中

```
int SaveBitmap(char* path, Bitmap* bmp)
```

- 保存bmp图像到指定路径中

```
void bmp2rgbmatrix(Bitmap *bmp, int_buffer* b,
                   int_buffer* g, int_buffer* r)
```

- 将读取的线性排列的一维图像数据转换为二维矩阵数据，同时将r、g、b通道的数据分离

```
Bitmap* rgbmatrix2bmp(int_buffer* b,int_buffer* g,int_buffer* r)
```

- 将分离的r、g、b通道的矩阵数据转换为bmp文件的数据排列方式

```
void RGB2Ycbcr(int_buffer* b, int_buffer* g, int_buffer* r,
               int_buffer* y, int_buffer* cb, int_buffer* cr)
```

- 将RGB图像转为YCbCr图像，RGB为8位数据，YCbCr为12位数据

```
void Ycbcr2RGB(int_buffer* y, int_buffer* cb, int_buffer* cr,
               int_buffer* b, int_buffer* g, int_buffer* r)
```

- 将YCbCr通道数据转为RGB通道数据，YCbCr为12位数据，RGB为8位数据

```
int_buffer bicubic(int_buffer *ptr_ori)
```

- 双三次插值函数，对单通道矩阵形式数据使用双三次插值二倍放大，缺失的数据以镜像的方式补全

```
int_buffer nbsrf(int_buffer* y, int psize, int overlap)
```

- 对输入的矩阵类型数据进行基于分层聚类的回归矫正
- 对输入的矩阵类型数据进行基于分层聚类的回归矫正

```
int_buffer image2patches(int_buffer* y, int psize, int overlap)
```

- 将矩阵分解为多个patch向量

```
int_buffer patchesdif(int_buffer* patches)
```

- 计算每个patch向量与自身均值的差，得到差异值向量

```
int cluster(int array[], int node, int nclusters)
```

- 对1×36的向量进行二叉树搜索

```
int* regress(int vector[], int cluster)
```

- 对1×36的向量进行回归矫正

```
int_buffer patches2image(int_buffer* patches, int row, int col,
                         int psize, int overlap)
```

- 根据图像和patch的对应关系，将差异值patch转为和图像大小相同的矩阵

```
void bufdiv(int_buffer *buf_2k, int_buffer buf_1k[])
```

- 将1920×1080的矩阵等分成四个960×540大小的矩阵

```
void bufjoin(int_buffer buf_2k[], int_buffer *buf_4k)
```

- 将4个1920×1080的矩阵合并成3840×2160的矩阵

2.3. Struct Documentation

```
typedef struct _Bitmap
{
    BITMAPFILEHEADER bmfh;
    BITMAPINFOHEADER bmih;
    int width;
    int height;
    int imageSize;
    int widthStep;
    BYTE* imageData;
}   Bitmap;
```

- 图像结构体：
  存储bmp的文件头、信息头、图像数据，关键信息。虽然信息头中包含宽度、高度等信息，直接在结构体中定义能够方便使用。
- Parameters
  `BITMAPFILEHEADER bmfh` 位图文件头结构体
  `BITMAPINFOHEADER bmih` 位图信息头结构体
  `int width` 位图的宽度
  `int height` 位图的高度
  `int imageSize` 图像的数据大小
  `BYTE* imageData` 图像像素数据
  `int widthStep` 排列的图像行大小，应对行像素不为4的倍数的情况

```
typedef struct int_matrix_size
{
  int **buf;
  size_t row;
  size_t col;
} int_buffer;
```

- 矩阵结构体：
  包含存储像素信息的二维数组，与二维数组的行数和列数。

- Parameters
  `int **buf` 排列的矩阵数据
  `size_t row` 矩阵的行数
  `size_t col` 矩阵的列数

2.4. Function Documentation

```
int ReadBitmap(char* path,
               Bitmap* bmp)
```

- 将指定路径的bmp图像读入到Bitmap结构体中
- Parameters
  [in]： `char* path` 图像路径
  [out]： `Bitmap* bmp` 存储图像的结构体
- Returns
  0：读取失败
  1：读取成功

```
int SaveBitmap(char* path,
               Bitmap* bmp)
```

- 保存bmp图像到指定路径中
- Parameters
  [out]： `char* path` 保存文件的路径
  [in]： `Bitmap* bmp` 需要保存的数据
- Returns
  0：保存失败
  1：保存成功

```
void bmp2rgbmatrix(Bitmap *bmp,
                   int_buffer* b,
                   int_buffer* g,
                   int_buffer* r)
```

- 将读取的线性排列的一维图像数据转换为二维矩阵数据，同时将r、g、b通道的数据分离
- Parameters
  [in]： `Bitmap *bmp` 读取到的图像结构体
  [out]： `int_buffer* b` B通道图像数据
  [out]： `int_buffer* g` G通道图像数据
  [out]： `int_buffer* r` R通道图像数据

```
Bitmap* rgbmatrix2bmp(int_buffer* b,
                      int_buffer* g,
                      int_buffer* r)
```

- 将分离的r、g、b通道的矩阵数据转换为bmp文件的数据排列方式
- Parameters
  [in]： `int_buffer* b` B通道数据
  [in]： `int_buffer* g` G通道数据
  [in]： `int_buffer* r` R通道数据
- Returns
  返回保存图像数据的Bitmap结构体的指针

```
void RGB2Ycbcr(int_buffer* b,
               int_buffer* g,
               int_buffer* r,
               int_buffer* y,
               int_buffer* cb,
               int_buffer* cr)
```

- 将RGB图像转为YCbCr图像，RGB为8位数据，YCbCr为12位数据
- Parameters
  [in]： `int_buffer* b` B通道数据，8位整数
  [in]： `int_buffer* g` G通道数据，8位整数
  [in]： `int_buffer* r` R通道数据，8位整数
  [out]： `int_buffer* y` Y通道数据，12位整数
  [out]： `int_buffer* cb` Cb通道数据，12位整数
  [out]： `int_buffer* cr` Cr通道数据，12位整数

```
void Ycbcr2RGB(int_buffer* y,
               int_buffer* cb,
               int_buffer* cr,
               int_buffer* b,
               int_buffer* g,
               int_buffer* r)
```

- 将YCbCr通道数据转为RGB通道数据，YCbCr为12位数据，RGB为8位数据
- Parameters
  [in]： `int_buffer* y` Y通道数据，12位整数
  [in]： `int_buffer* cb` Cb通道数据，12位整数
  [in]： `int_buffer* cr` Cr通道数据，12位整数
  [out]： `int_buffer* b` B通道数据，8位整数
  [out]： `int_buffer* g` G通道数据，8位整数
  [out]： `int_buffer* r` R通道数据，8位整数

```
int_buffer bicubic(int_buffer *ptr_ori)
```

- 双三次插值函数，对单通道矩阵形式数据使用双三次插值二倍放大，缺失的数据以镜像的方式补全
- Parameters
  [in]： `int_buffer *ptr_ori` 某一通道的矩阵数据
- Returns
  - 返回类型
    int_buffer矩阵结构体
  - 返回值
    返回二倍双三次插值后的数据

```
int_buffer nbsrf(int_buffer* y,
                 int psize,
                 int overlap)
```

- 对输入的矩阵类型数据进行基于分层聚类的回归矫正
- Parameters
  [in]： `int_buffer* y` Y通道数据
  [in]： `int psize` patch的边长，算法中设为6
  [in]： `int overlap` 分割patch的步长，算法中设为2
- Returns
  - 返回类型
    int_buffer矩阵结构体
  - 返回值
    矫正后的通道值

```
int_buffer image2patches(int_buffer* y,
                         int psize,
                         int overlap)
```

- 将矩阵分解为多个patch向量
- Parameters
  [in]： `int_buffer* y` Y通道矩阵数据
  [in]： `int psize` patch的边长，算法中设为6
  [in]： `int overlap` 分割patch的步长，算法中设为2
- Returns
  - 返回类型
    int_buffer矩阵结构体
  - 返回值
    每个Y通道数据分割成的patch构成矩阵的一行，所有patch顺序排列构成矩阵

```
int_buffer patchesdif(int_buffer* patches)
```

- 计算每个patch向量与自身均值的差，得到差异值向量
- Parameters
  [in]： `int_buffer* patches` 所有patch向量构成的矩阵结构体
- Returns
  - 返回类型
    int_buffer矩阵结构体
  - 返回值
    输入的矩阵的元素值减去所在patch的均值后的矩阵

```
int cluster(int array[],
            int node,
            int nclusters)
```

- 对大小为36的数组进行二叉树搜索
- Parameters
  [in]： `int array[]` 大小为36的数组
  [in]： `int node` 开始搜索的节点编号，算法中从0开始
  [in]： `int nclusters` 树的叶节点数目，算法中设为128
- Returns
  - 返回类型
    int
  - 返回值
    返回[0,127]之间的整数，表示聚类的类别

```
int* regress(int vector[],
             int cluster)
```

- 对1*36的向量进行回归矫正
- Parameters
  [in]： `int vector[]` 大小为36的数组
  [in]： `int cluster` 矫正矩阵的编号，为[0,127]之间的整数
- Returns
  - 返回类型
    int*
  - 返回值
    元素个数为36的矫正值数组的指针

```
int_buffer patches2image(int_buffer* patches,
                         int row,
                         int col,
                         int psize,
                         int overlap)
```

- 根据图像和patch的对应关系，将差异值patch转为和图像大小相同的矩阵
- Parameters
  [in]： `int_buffer* patches` 差异值向量构成的矩阵结构体
  [in]： `int row` 生成的图像的行数
  [in]： `int col` 生成的图像的列数
  [in]： `int psize` patch的边长，算法中设为6
  [in]： `int overlap` 分割patch的步长，算法中设为2
- Returns
  - 返回类型
    int_buffer 矩阵结构体
  - 返回值
    矫正后的图像数据

```
void bufdiv(int_buffer *buf_2k,
            int_buffer buf_1k[])
```

- 将1920×1080的矩阵等分成四个960×540大小的矩阵
- Parameters
  [in]： `buf_2k` 2k大小的矩阵结构体指针
  [in]： `buf_1k[]` 1k大小的矩阵结构体数组

```
void bufjoin(int_buffer buf_2k[],
             int_buffer *buf_4k)
```

- 将4个1920×1080的矩阵合并成3840×2160的矩阵
- Parameters
  [in]： `buf_2k[]` 2k大小的矩阵结构体数组
  [in]： `buf_4k` 4k大小的矩阵结构体指针

3. 寄存器说明

3.1. 外部输入控制寄存器

本IP没有外部输入的控制寄存器。

如**算法设计说明文件**所述，本IP为流式运行：由于图片大小给定、放大模式确定、输入输出定向，IP上电reset后接收数据即开始运行，因此不需要额外设计寄存器规定其工作状态状态。

3.2. 内部控制寄存器

内部控制寄存器主要为状态机的状态控制寄存器，包括Image_Control模块的INTR状态机控制寄存器和BUFFER状态机控制寄存器，buffer_in模块的RD状态机控制寄存器以及buffer_out模块的RD状态机控制寄存器。

其中，状态机控制寄存器的具体分析见 rtl 模块说明。

4. rtl 模块说明

4.1. rtl模块层次概述

本IP的rtl层次如下所示：

```
Upsampling_Bayes.v
├─ S_AXIS_2_pixel_low.v
├─ Upsampling_Bayes_M00_AXIS.v
├─ Upsampling_Bayes_S00_AXIS.v
├─ pixel_high_2_M_AXIS.v
└─ Super_Res.v
     └─ Image_Control.v
            ├─ bicubic_core.v
            │    ├─ conv.v
            │    └─ in_buffer.v
            │          └─ line_in_buffer.v
            ├─ bayes_core.v
            │    ├─ mean_subtract.v
            │    ├─ overlap_buffer.v
            │    │    └─ line_overlap_buffer.v
            │    ├─ overlap_patch.v
            │    ├─ regression_matrix.v
            │    │    └─ regression_matrix_unit.v
            │    │          └─ matrix_multiply_uint.v
            │    │                └─ dual_mul.v
            │    ├─ regression_tree.v
            │    │    ├─ regression_tree_stage1.v
            │    │    ├─ regression_tree_stage2_7.v
            │    │    └─ vector_multiply.v
            │    └─ y_shift.v
            ├─ bayes_buffer.v
            │    └─ line_bayes_buffer.v
            ├─ pipeline_buffer.v
            ├─ rgb2ycbcr.v
            └─ ycbcr2rgb.v
```

4.2. 顶层数据接口

4.2.1. 低分辨率像素输入接口

> IP的低分辨率像素输入接口由一个AXIS-slave总线接口（ `Upsampling_Bayes_S00_AXIS.v` ）与一个数据转接器（ `S_AXIS_2_piexl_low.v` ）将总线数据变为像素数据。

4.2.1.1. 模块功能

接口对外采用32位宽的AXI-Stream协议，对接上游DMA的M_AXIS_MM2S接口，接收来自内存的低分辨率像素序列**输入**，整理为 `{r[RGB_WIDTH],g[RGB_WIDTH],b[RGB_WIDTH]}` 标准排列的8位深通道像素，对内**输出**。

4.2.1.2. 组合模块接口

Table 4.1 低分辨率像素输入接口

参数	值	功能
.C_S_AXIS_TDATA_WIDTH	32	AXIS总线数据宽度
.PIXEL_WIDTH	24	RGB像素宽度

接口	方向	位宽	功能
.S_AXIS_ACLK	input	1	AXIS总线时钟
.S_AXIS_ARESETN	input	1	AXIS总线重置信号，低有效
.S_AXIS_TREADY	output	1	告知DMA可接收
.S_AXIS_TDATA	input	C_S_AXIS_TDATA_WIDTH	AXIS数据总线
.S_AXIS_TSTRB	input	C_S_AXIS_TDATA_WIDTH/8	未使用
.S_AXIS_TLAST	input	1	DMA端告知传送结束
.S_AXIS_TVALID	input	1	为高时，.S_AXIS_TDATA有效
.pixel_low	output	PIXEL_WIDTH	向IP内部返回RGB像素数据
.buf_rden	input	1	为高时，下游超分模块请求读
.trans_eff	output	1	为高时，.pixel_low有效

4.2.2. 高分辨率像素输出接口

> IP的高分辨率像素输出接口由一个数据转接器（ `piexl_high_2_M_AXIS.v` ）将像素数据变为总线数据与一个AXIS-master总线接口（ `Upsampling_Bayes_M00_AXIS.v` ）。

4.2.2.1. 模块功能

接口将超分IP产生的 `{r[RGB_WIDTH],g[RGB_WIDTH],b[RGB_WIDTH]}` 像素序列，打包整理为128位宽的AXI-Stream协议，对接下游DMA的M_AXIS_S2MM接口，将整理后的高分辨率像素序列**输出**至内存；

4.2.2.2. 组合模块接口

Table 4.2 高分辨率像素输出接口

参数	值	功能
.C_M00_AXIS_START_COUNT	16	AXIS总线初始化等待周期
.C_M_AXIS_TDATA_WIDTH	128	AXIS总线数据宽度
.PIXEL_WIDTH	24	RGB像素宽度

接口	方向	位宽	功能
.M_AXIS_ACLK	input	1	AXIS总线时钟
.M_AXIS_ARESETN	input	1	AXIS总线重置信号，低有效
.M_AXIS_TREADY	input	1	DMA可接收
.M_AXIS_TDATA	output	C_M_AXIS_TDATA_WIDTH	AXIS数据总线
.M_AXIS_TSTRB	output	C_M_AXIS_TDATA_WIDTH / 8	未使用
.M_AXIS_TLAST	output	1	告知DMA端两行传送完毕
.M_AXIS_TVALID	output	1	为高时，.M_AXIS_TDATA数据有效
.pixel_high	input	PIXEL_WIDTH*4	IP超分的一组4个RGB像素数据
.buf_wren	input	1	为高时，.pixel_high有效
.stuck	output	1	为高时，DMA或buffer堵塞

4.3. 双三次核

4.3.1. 双三次核

`bicubic_core` 模块是双三次核顶层控制模块，该模块由控制器和例化的底层模块组成

4.3.1.1. 模块功能

双三次核模块的功能建立数据通路以及协调外部接口模块与双三次核之间和 `buffer_in` 模块与 `bayes_buffer` 之间 的读写冲突。

4.3.1.2. 模块接口

Table 4.3 bicubic_core 模块接口

参数	值	功能

参数	值	功能
PIXEL_WIDTH	24	像素宽度
RGB_WIDTH	8	RGB单通道宽度
YCbCr_WIDTH	12	YCbCr单通道宽度
IN_LINE_WIDTH	960	输入图像宽度
OUT_LINE_WIDTH	1920	输出图像宽度
YCbCr_PARA_WIDTH	20	YCbCr参数量化位宽

接口	方向	位宽	功能
clk	input	1	用户时钟
rst_n	input	1	用户复位信号
i_pixel_data	input	[PIXEL_WIDTH-1:0]	输入低分辨率像素数据
o_pixel_data	output	[4 * PIXEL_WIDTH-1:0]	输出高分辨率像素数据
i_pixel_data_valid	input	1	输入数据有效
i_stuck	input	1	外部堵塞信号, 告知双三次核停止输出数据
o_pixel_data_valid	output	1	输出数据有效
o_ready	output	1	输出准备信号，告知外部接口输入数据

4.3.1.3. 工作模式

4.3.1.3.1. 概述

Bicubic_core 模块控制器的主要功能是建立数据通路，协调输入输出接口与双三次核的读写冲突，以及协调buffer_in与bayes_buffer之间的读写冲突。

- 对于双三次核与输出接口之间的关系，只要输出数据有效以及输出接口准备好接收双三次核的数据即可输出;
- 对于双三次核与输入接口的关系，使用 INTR 状态机控制是否读入以及读入的数据量;
- 对于 buffer_in 与 bayes_buffer 的关系，使用 BUFFER 状态机控制 bayes_buffer 是否能够接收 buffer_in 的数据。

4.3.1.3.2. 数据通路

- rgb2ycbcr: 输入数据rgb通道转换为ycbcr通道
  - data input
    - pixel_in[RGB_WIDTH*3]={r[RGB_WIDTH],g[RGB_WIDTH],b[RGB_WIDTH]}
  - data output
    - y[YCbCr_WIDTH]
    - cb[YCbCr_WIDTH]
    - cr[YCbCr_WIDTH]
- buffer_in：输入数据缓冲
  - data input
    - y[YCbCr_WIDTH]
    - cb[YCbCr_WIDTH]
    - cr[YCbCr_WIDTH]
  - data output
    - y[YCbCr_WIDTH*4*4]
    - cb[YCbCr_WIDTH*4*4]
    - cr[YCbCr_WIDTH*4*4]
- conv：双三次计算
  - data input
    - y[YCbCr_WIDTH*4*4]
    - cb[YCbCr_WIDTH*4*4]
    - cr[YCbCr_WIDTH*4*4]
  - data output
    - y[YCbCr_WIDTH*2*2]={y11,y10,y01,y00}
    - cb[YCbCr_WIDTH*2*2]={cb11,cb10,cb01,cb00}
    - cr[YCbCr_WIDTH*2*2]={cr11,cr10,cr01,cr00}

4.3.1.3.3. 中断状态机

Table 4.4 bicubic_core 模块接口

状态	值	描述
INTR_IDLE	3'b000	初始化状态
INTR_OUT_1	3'b001	中断状态1
INTR_OUT_3_0	3'b010	中断状态2
INTR_OUT_3_1	3'b100	中断状态3

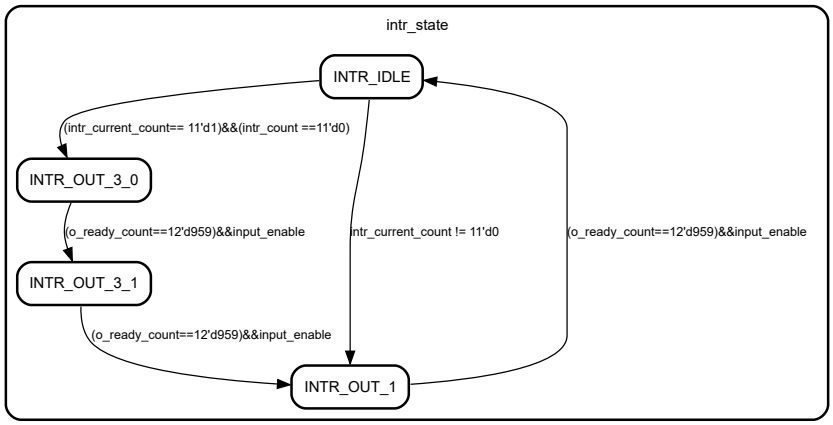

Fig. 4.1 Image_Control INTR状态机

INTR 状态机初始化状态为 `INTR_IDLE` ，有三个允许输入状态分别为 `INTR_OUT_1`， `INTR_OUT_3_0`， `INTR_OUT_3_1` 状态。

- 系统复位后，状态机处于 `RD_IDLE` 状态；
- 当状态机接收一帧图像的第一次中断时，状态机从 `INTR_IDLE` 跳转到 `INTR_OUT_3_0` 状态，此时输入第一行数据；
  - 第一行数据输入结束后，状态机从 `INTR_OUT_3_0` 跳转到 `INTR_OUT_3_1` 状态，此时输入第二行数据；
  - 第二行数据输入结束后，状态机从 `INTR_OUT_3_1` 跳转到 `INTR_OUT_1` 状态，此时输入第三行数据；
  - 第三次数据输入结束后，状态机跳转到 `INTR_IDLE` 状态等待下一次中断。
- 当接收的中断信号不是一帧图像的第一次中断时，状态机跳转到 `INTR_OUT_1` 状态，此时输入一行数据；
  - 一行数据输入结束后，状态机跳转到`INTR_IDLE`状态等待下一次中断。

4.3.2. rgb2ycbcr

`rgb2ycbcr` 模块是输入数据转换模块，将输入的 rgb 像素转换成 `YCbCr` 像素。

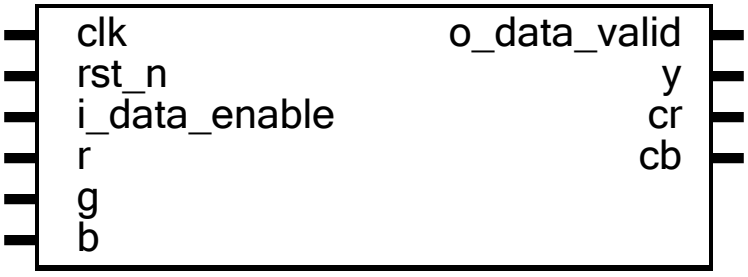

Fig. 4.2 rgb2cbcr 模块

4.3.2.1. 模块功能

rgb2ycbcr 模块将 rgb 通道像素转换为 ycbcr 通道像素。该模块采用流式处理。

4.3.2.2. 模块接口

Table 4.5 ycbcr 模块接口

接口	方向	位宽	功能
clk	input	1	用户时钟
rst_n	input	1	用户复位信号
i_data_enable	input	1	输入信号使能
o_data_valid	output	1	输出数据有效
r	input	[RGB_WIDTH-1 :0]	输入像素 r 通道
g	input	[RGB_WIDTH-1 :0]	输入像素 g 通道
b	input	[RGB_WIDTH-1 :0]	输入像素 b 通道
y	output	[YCbCr_WIDTH-1:0]	输出像素 y 通道
cr	output	[YCbCr_WIDTH-1:0]	输出像素 cb 通道
cb	output	[YCbCr_WIDTH-1:0]	输出像素 cr 通道

4.3.3. buffer_in

> `buffer_in` 是输入数据模块与运算模块的缓冲模块，由例化的行缓存与控制器组成。 `buffer_in` 总共例化5个三通道行缓存，其中一个用于输入，四个用于输出。

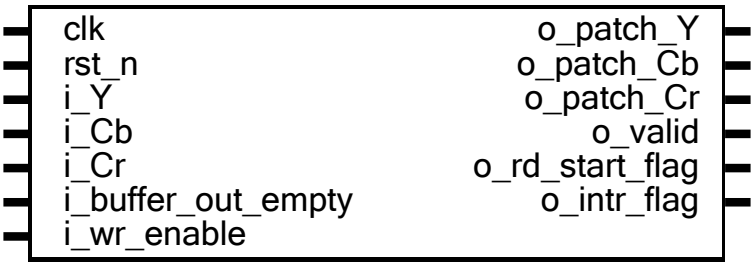

Fig. 4.3 buffer_in 模块

4.3.3.1. 模块功能

`buffer_in` 将 `rgb2ycbcr` 模块产生的数据存入行缓存，打包用于双三次运算的 4 * 4 大小的像素块输出到 `conv` 模块。

4.3.3.2. 模块接口

Table 4.6 buffer_in 模块接口

参数	值	功能
YCbCr_WIDTH	12	YCbCr像素宽度
IN_LINE_WIDTH	960	输入图像宽度

接口	方向	位宽	功能
clk	input	1	用户时钟
rst_n	input	1	用户复位信号，低电平使能
i_Y	input	[YCbCr_WIDTH-1:0]	输入像素Y通道
i_Cb	input	[YCbCr_WIDTH-1:0]	输入像素Cb通道
i_Cr	input	[YCbCr_WIDTH-1:0]	输入像素Cr通道
o_patch_Y	output	[16*YCbCr_WIDTH-1:0]	输出像素块Y通道
o_patch_Cb	output	[16*YCbCr_WIDTH-1:0]	输出像素块Cb通道
o_patch_Cr	output	[16*YCbCr_WIDTH-1:0]	输出像素块Cr通道
i_buffer_out_empty	input	1	输入 buffer_out 状态信号， 告知 buffer_in 可输出数据
i_wr_enable	input	1	输入写使能信号
o_valid	output	1	输出数据有效信号
o_rd_start_flag	output	1	输出读开始状态信号， 告知 buffer_out 输出开始
o_intr_flag	output	1	输出读请求状态信号， 告知 Image_Control 发送下一行数据

4.3.3.3. 工作模式

4.3.3.3.1. 工作模式概述

`buffer_in` 的功能包括输入功能和输出功能。

- 对于输入功能，只要输入数据有效，`buffer_in` 就接收数据；
- 对于输出功能，`buffer_in` 通过读状态机控制输出时序。

其中，`Image_Control` 模块控制 `buffer_in` 的输入和输出。

4.3.3.3.2. 读状态机

Table 4.7 buffer_in 读状态机状态表

状态	值	描述
RD_IDLE	1'b0	初始状态
RD_BUFFER	1'b1	读状态

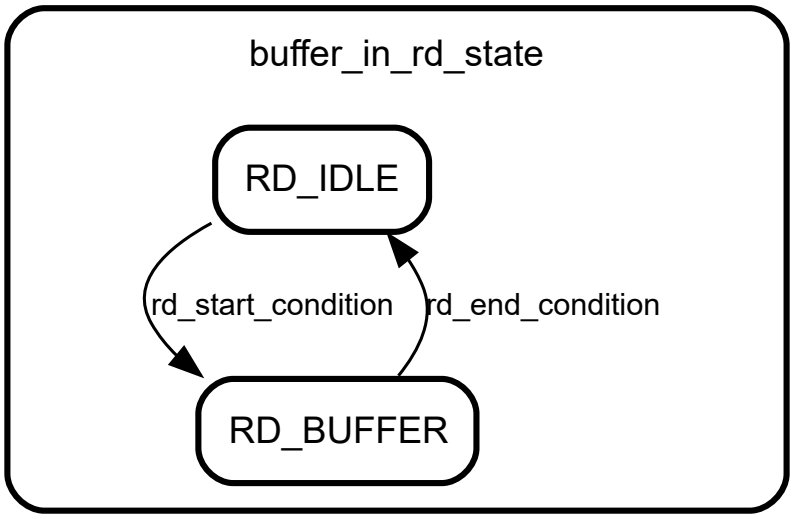

Fig. 4.4 buffer_in 读状态机

读状态机初始化状态为 `RD_IDLE` ，读状态为 `RD_BUFFER` 。

- 系统复位后，状态机处于 `RD_IDLE` 状态；
- `rd_start_condition = 1` 时，状态机从 RD_IDLE 跳转到 RD_BUFFER 状态；
- `rd_end_condition = 1` 时，状态机从 RD_BUFFER 跳转到 RD_IDLE 状态。

其中 `rd_start_condition = 1` 的满足条件为 buffer_in 一个通道的像素总数大于 4 * IN_LINE_WIDTH ； `rd_end_condition = 1` 的满足条件为 buffer_in 当前行读出的像素数等于 IN_LINE_WIDTH。

4.3.4. conv

> `conv` 模块是双三次上采样运算单元，使用固定的参数计算由输入低分辨率率像素 `patch` 决定的高分辨率像素 `patch` 。

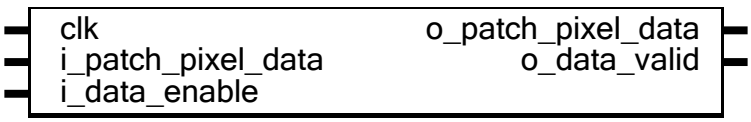

Fig. 4.5 conv 模块

4.3.4.1. 模块功能

`conv` 模块接收低分辨率图像 4 *4 大小的 `patch` ，产生该`patch` 对应的高分辨率图像 2* 2 大小的`patch`。该模块流式处理。

4.3.4.2. 模块接口

Table 4.8 conv 模块接口

参数	值	功能
YCbCr_WIDTH	12	YCbCr 像素单通道宽度

接口	方向	位宽	功能
clk	input	1	用户时钟
i_patch_pixel_data	input	[YCbCr_WIDTH * 16 -1:0]	输入低分辨率图像块
o_patch_pixel_data	output	[YCbCr_WIDTH *4 -1 :0]	输出高分辨率图像块
i_data_enable	input	1	输入数据使能
o_data_valid	output	1	输出数据有效

4.3.5. buffer_out

> `buffer_out` 模块是输出数据与接口之间的缓冲模块，由例化的行缓存 和控制器组成。
> `buffer_in` 总共例化8个三通道行缓存，其中两个用于输入，两个用于输出，其余的用于缓冲，此处的冗余设计是为了后续进一步扩展。

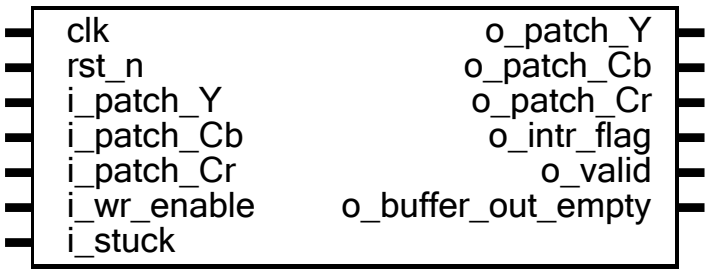

Fig. 4.6 buffer_out 模块

4.3.5.1. 模块功能

`buffer_out` 将 `conv` 模块产生的数据块存入行缓存，将输出高分辨率 `YCbCr` 像素打包输出到 `ycbcr2rgb` 模块。

4.3.5.2. 模块接口

Table 4.9 buffer_out 模块接口

参数	值	功能
YCbCr_WIDTH	12	YCbCr像素宽度
OUT_LINE_WIDTH	1920	输出图像宽度

接口	方向	位宽	功能
clk	input	1	用户时钟
rst_n	input	1	用户复位信号，低电平使能
out_patch_Y	input	[4*YCbCr_WIDTH-1:0]	输入像素块Y通道
out_patch_Cb	input	[4*YCbCr_WIDTH-1:0]	输入像素块Cb通道
out_patch_Cr	input	[4*YCbCr_WIDTH-1:0]	输入像素块Cr通道
out_Y	output	[4 * YCbCr_WIDTH-1:0]	输出像素块Y通道
out_Cb	output	[4 * YCbCr_WIDTH-1:0]	输出像素块Cb通道
out_Cr	output	[4 * YCbCr_WIDTH-1:0]	输出像素块Cr通道
i_wr_enable	input	1	写使能信号
stuck	input	1	输入 堵塞状态信号
o_intr_flag2	output	1	输出中断信号, 告知 Image_Control 向 buffer_in 发送下一帧图像
o_valid	output	1	输出数据有效
o_buffer_out_empty	output	1	空信号, 告知 buffer_in 向buffer_out 发送下一行数据

4.3.5.3. 工作模式

工作模式概述

`buffer_out` 的功能包括输入功能和输出功能。

- 对于输入功能，只要输入数据有效，`buffer_out` 就接收数据；
- 对于输出功能，`buffer_out` 通过读状态机控制输出时序。

其中，`Image_Control` 模块控制 `buffer_out` 的输入和输出。

读状态机

Table 4.10 buffer_out 读状态机

状态	值	描述
RD_IDLE	1'b0	初始状态
RD_BUFFER	1'b1	读状态

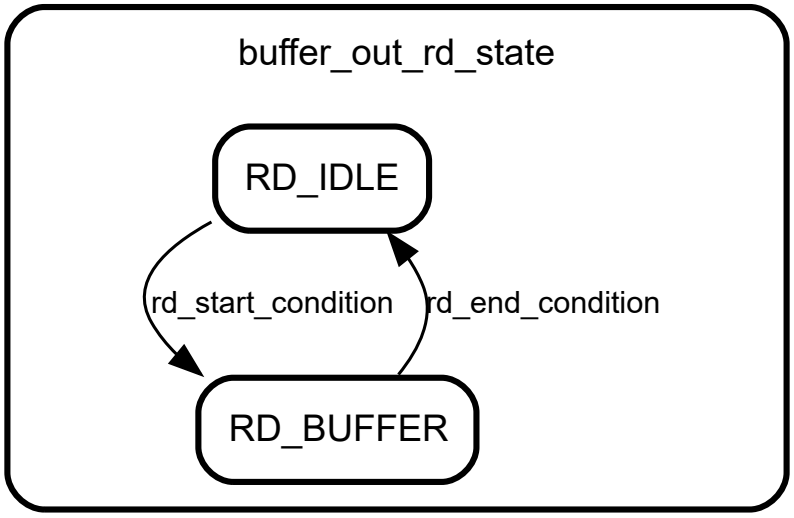

Fig. 4.7 buffer_out 读状态机

读状态机初始化状态为 `RD_IDLE` ，读状态为 `RD_BUFFER` 。

- 系统复位后，状态机处于 `RD_IDLE` 状态；
- `rd_start_condition = 1` 时，状态机从 RD_IDLE 跳转到 RD_BUFFER 状态；
- `rd_end_condition = 1` 时，状态机从 RD_BUFFER 跳转到 RD_IDLE 状态。

其中 `rd_start_condition = 1` 的满足条件为 buffer_out 一个通道的像素总数大于 6 *IN_LINE_WIDTH ； `rd_end_condition` 满足条件为 buffer_out 当前两行读出的像素数等于 2 * OUT_LINE_WIDTH。

4.4. Bayes_Core

4.4.1. mean_subtract

`mean_subtract` 计算输入贝叶斯像素块减去其平均值的结果，得到差异值向量并输出。

4.4.1.1. 模块功能

`mean_subtract` 接收 6 * 6 大小的像素块，用像素块原始数据减去36个像素的平均值得到差异值向量 6 * 6 像素块。

4.4.1.2. 模块接口

Table 4.11 mean_subtract 模块接口

Port name	Direction	Type	Description
clk	input		输入时钟
rst_n	input		复位信号
i_bayes_patch	input	[`YCbCr_WIDTH * 36 -1:0]	输入贝叶斯像素块
i_data_enable	input		输入数据有效
o_subtract_patch	output	[`Sub_WIDTH * 36 -1 :0]	输出差异值向量
o_data_valid	output		输出数据有效信号

4.4.2. regression_tree

> `regression_tree` 将差异值向量与评分向量向量乘得到评分，使用得到的评分进行回归树树搜索，最终得到回归矩阵的地址索引。

4.4.2.1. 模块功能

基于分层聚类算法计算评分，使用得到的评分进行回归树树搜索，最终得到回归矩阵的地址索引。

在回归树中的每一个节点具有两个评分向量：左分支评分向量、右分支评分向量，使用差异值与两个评分向量向量乘求绝对值得到两个评分，当左分支的评分大于等于右分支的评分时，访问该节点的左子节点，直到到达最底层。

4.4.2.2. 使用分布存储优化访存效率

如果将所有节点的评分向量放在一起，当某一差异值向量访问其中一层节点时，由于输出带宽有限，其他差异值向量不能访问该层节点，也不能访问其他层的节点。

由于某一层节点在一个时刻只会被一个差异值向量访问，因此考虑使用分布存储将每一层的评分向量存储在独立的存储器中。使用分布存储，同一时刻，七层的节点同时被访问，访存效率达到最大值。

4.4.2.3. 模块接口

Table 4.12 regression_tree 模块接口

Port name	Direction	Type	Description

Port name	Direction	Type	Description
clk	input		输入时钟
rst_n	input		复位信号
i_subtract_patch	input	[`Sub_Patch_WIDTH -1:0]	输入差异值向量
i_data_enable	input		输入数据有效
o_subtract_patch	output	[`Sub_Patch_WIDTH -1:0]	输出差异值向量缓存
o_addr	output	[`Regression_Addr_WIDTH-1:0]	输出回归矩阵地址索引
o_data_valid	output		输出数据有效

4.4.3. regresssion_matrix

`regresssion_matrix` 将差异值向量与回归矩阵矩阵乘得到校正后的差异值向量输出

4.4.3.1. 模块功能

regresssion_matrix 对输入的差异值向量和索引到的回归矩阵进行矩阵相乘，得到校正后的差异值向量。

4.4.3.2. 使用 DSP 实现 SIMD multiply 提高硬件效率

输入的差异值向量 1*36 与回归矩阵 36*36 矩阵乘，相当于同一个差异值向量同时与36个不同的回归向量相乘。因此我们可以考虑使用`DSP`实现`SIMD`乘法提高硬件效率。

我们最终使用 {9bit input} * {{7bit weight_1},{10bit},{7bit weight_2} 的模式进行计算，DSP的输出同时包含两个输出数据。

4.4.3.3. 全局考虑数据通路考虑避免非移位除法

由于存在`overlap`，计算一个最终的差异值向量校正值时，需要计算9组差异值向量数据的平均值。
和`mean_subtract`模块中求平均值时类似的，在固定回归矩阵参数前再提前除以9，使硬件不需要计算除以9这一步。

最终固定的回归矩阵参数是原生算法中得到的回归矩阵除以81。

4.4.3.4. 模块接口

Table 4.13 regression_matrix 模块接口

Port name	Direction	Type	Description
clk	input		输入时钟

Port name	Direction	Type	Description
rst_n	input		复位信号
i_subtract_patch	input	[`Sub_Patch_WIDTH -1:0]	输入差异值向量
i_addr	input	[`Regression_Addr_WIDTH -1:0]	输入地址
i_data_enable	input		输入数据有效
o_matrix_patch	output	[`YCbCr_Patch_WIDTH -1:0]	输出校正差异值向量
o_data_valid	output		输出数据有效

4.5. Image Control

4.5.1. ycbcr2rgb

> ycbcr2rgb 模块是输出数据转换模块，将 ycbcr 通道像素转换成 rgb 通道像素

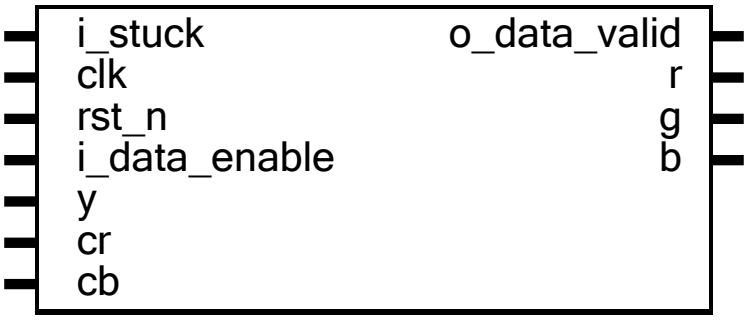

Fig. 4.8 ycbcr2rgb 模块

4.5.1.1. 模块功能

rgb2ycbcr 模块将 ycbcr 通道像素转换为 rgb 通道像素。该模块采用流式处理。

4.5.1.2. 模块接口

Table 4.14 rgb2ycbcr 模块接口

接口	方向	位宽	功能
clk	input	1	用户时钟
rst_n	input	1	用户复位信号
i_stuck	input	1	外部接口堵塞状态信号

接口	方向	位宽	功能
i_data_enable	input	1	输入数据使能
o_data_valid	output	1	输出数据有效
y	input	[YCbCr_WIDTH-1:0]	输入像素 y 通道
cb	input	[YCbCr_WIDTH-1:0]	输入像素 cb 通道
cr	input	[YCbCr_WIDTH-1:0]	输入像素 cr 通道
r	output	[RGB_WIDTH-1 :0]	输出像素 r 通道
g	output	[RGB_WIDTH-1 :0]	输出像素 g 通道
b	output	[RGB_WIDTH-1 :0]	输出像素 b 通道

5. 仿真验证环境说明

5.1. 仿真

> 系统控制逻辑较为清晰，直接使用 modelsim 进行仿真，其中使用到的 XLINX IP 导入到 modelsim 中一并进行仿真。

5.1.1. 系统数据通路

> 在testbench中将控制信号以及状态信号固定为工作状态时的值，使系统流式处理数据。验证系统的数据通路时，无须人为地在使输入数据随机变化。由于输入的图像数据具有一定随机性，只需对足够多的图片进行仿真，对比仿真结果和软件生成图像，只要结果一致就可认为系统在随机性的测试下能够满足要求。

5.1.1.1. 系统自启动

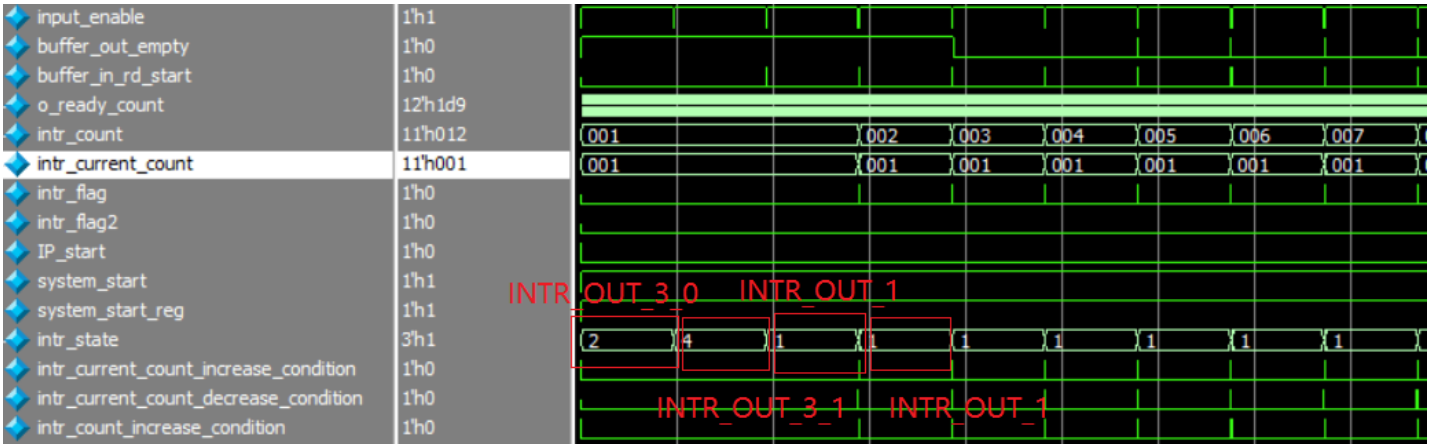

Fig. 5.1 流式处理INTR状态机

如图所示，在系统复位后，INTR 状态机从`INTR_IDLE`跳转到`INTR_OUT_3_0`，再跳转到`INTR_OUT_3_1`，再跳转到`INTR_OUT_1`并且读入三行数据。 读入三行数据后， INTR 状态机在`INTR_IDLE`与`INTR_OUT_1`之间切换并且读入一行数据，Bicubic Core 开始流式处理数据。

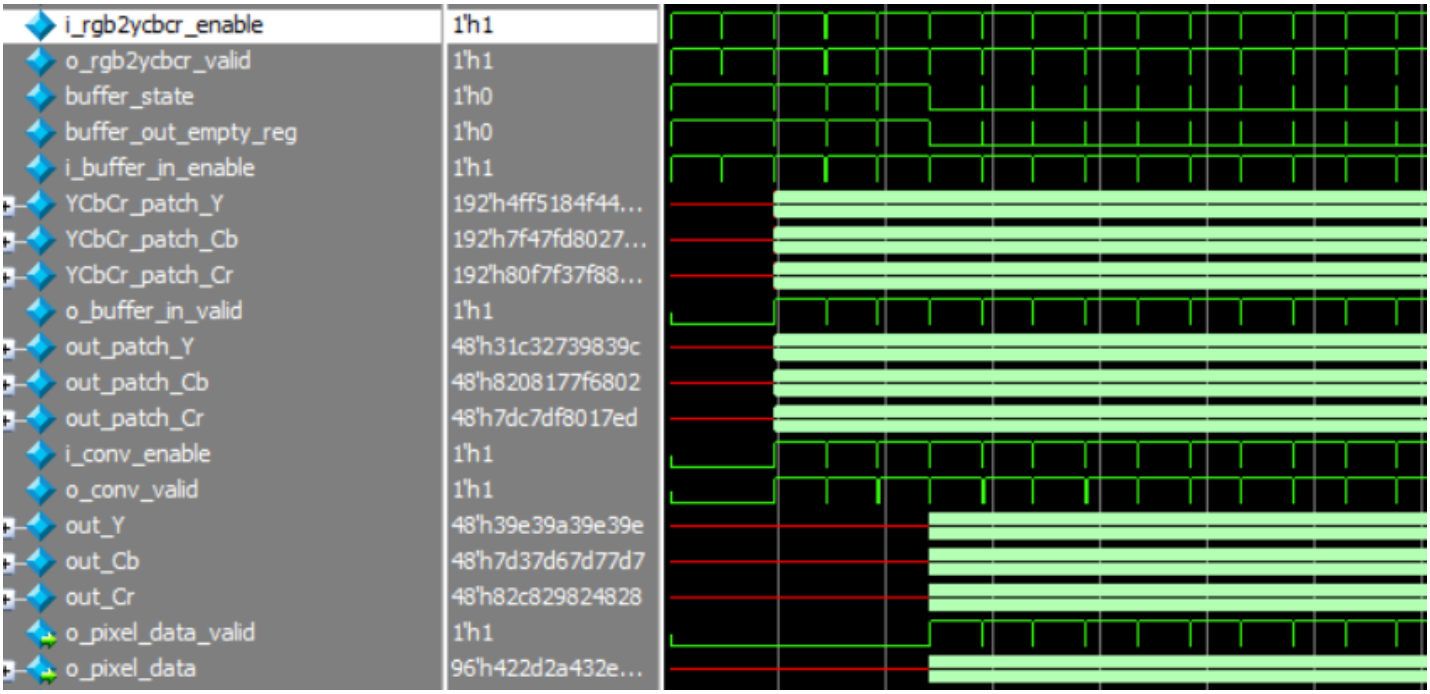

Fig. 5.2 流式处理数据通路

如图所示，数据通路符合 rtl 模块说明中所述的数据通路。

5.1.1.2. 图片切换

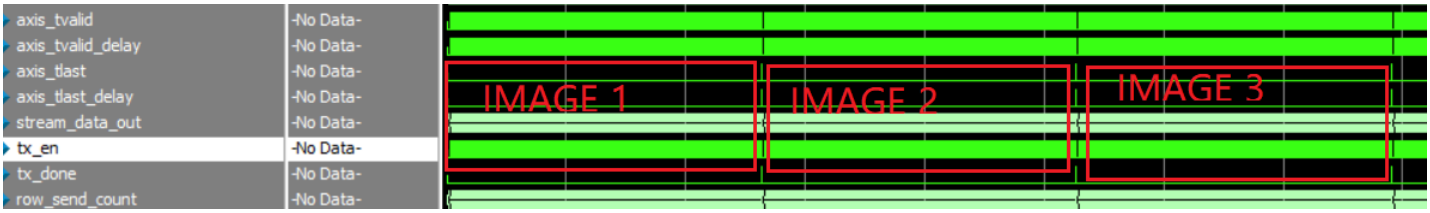

Fig. 5.3 流式处理图片切换

如图所示，图片切换功能正常。

5.1.1.3. 结果对比

对比仿真产生的超分辨率图像数据与软件产生的图像数据，结果完全一致。说明系统的数据通路正确。

5.1.2. 系统时序

> 输入输出接口时序使用 valid-ready 握手协议，在testbench令状态信号随机变化，使系统应对各种随机状态。

5.1.2.1. 接口时序

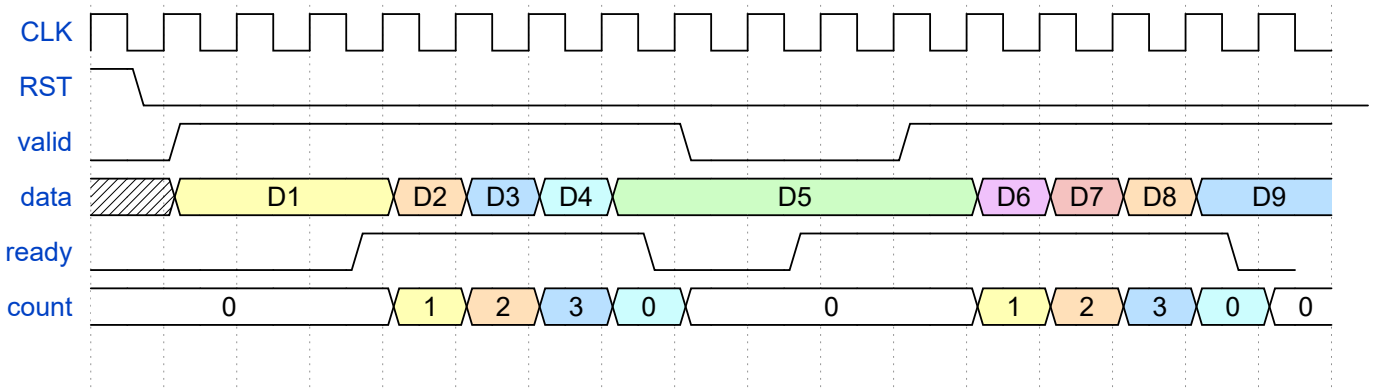

Fig. 5.4 输入接口时序

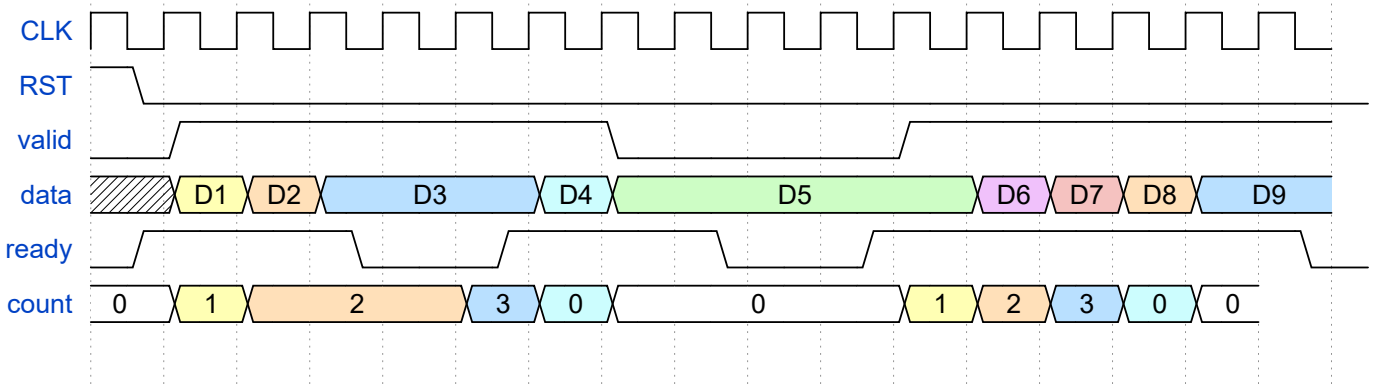

Fig. 5.5 输出接口时序

5.1.2.2. 仿真结果

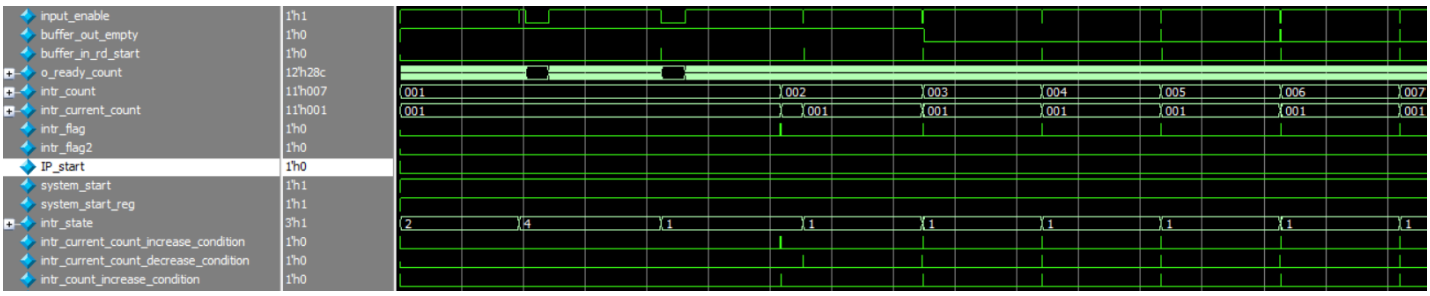

Fig. 5.6 随机状态处理仿真波形

如图所示，系统正常工作。 对比仿真产生的超分辨率图像数据与软件产生的图像数据，结果完全一致。

5.2. 硬件系统搭建

硬件系统总体架构设计如图5.7所示，ARM核上应用 petalinux 系统，图中控制流由粗虚线箭头标出，数据流（对于上采样IP是rgb像素流）由粗实线箭头标出：

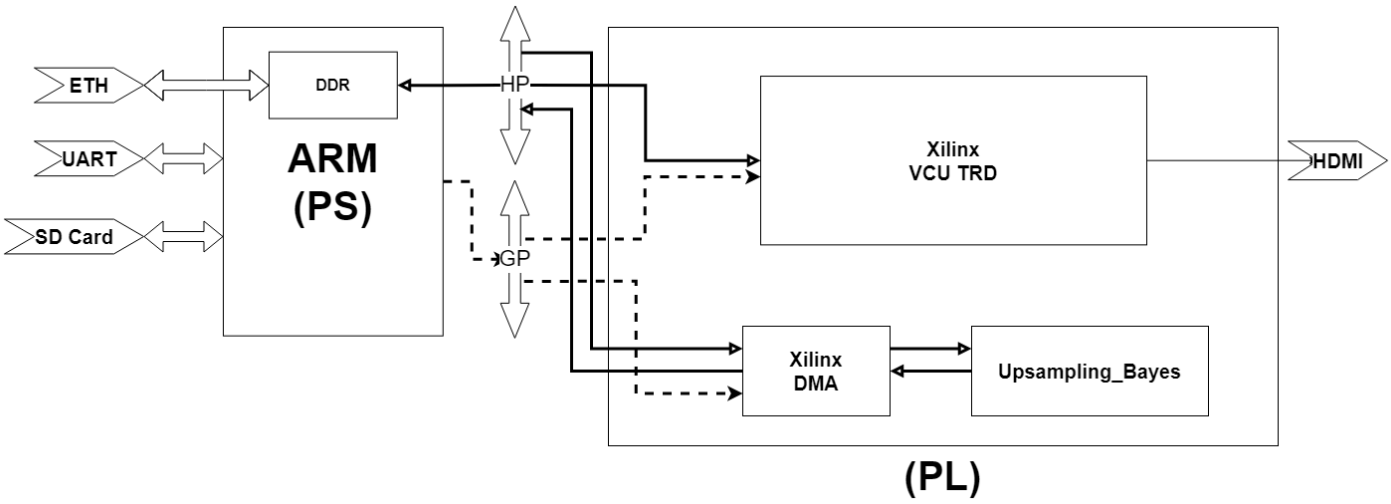

Fig. 5.7 硬件平台设计

在PS（processing system）端，ARM核通过SD卡（SD Card）boot启动，通过串口（UART）在上位机实现片上Linux的操作，通过以太网（ETH）实现上位机通信，上述接口全部使用Linux接管；对内部通过GP AXI（General Purpose Advanced eXtensible Interface）对PL（Progarmmable Logic）端进行控制传输，即寄存器配置，通过HP AXI（High Performance Advanced eXtensible Interface）进行数据流的传输。

PL端的数据流有两股，一股从DDR中将需要显示的图像搬送至HDMI显示子系统（2.1节）；另一股是由DMA作为桥梁，将DDR中的需要被放大的低分辨率像素搬运至超分IP，并将超分后的高分辨率像素搬运回DDR（2.2节），我们重点关注后者。

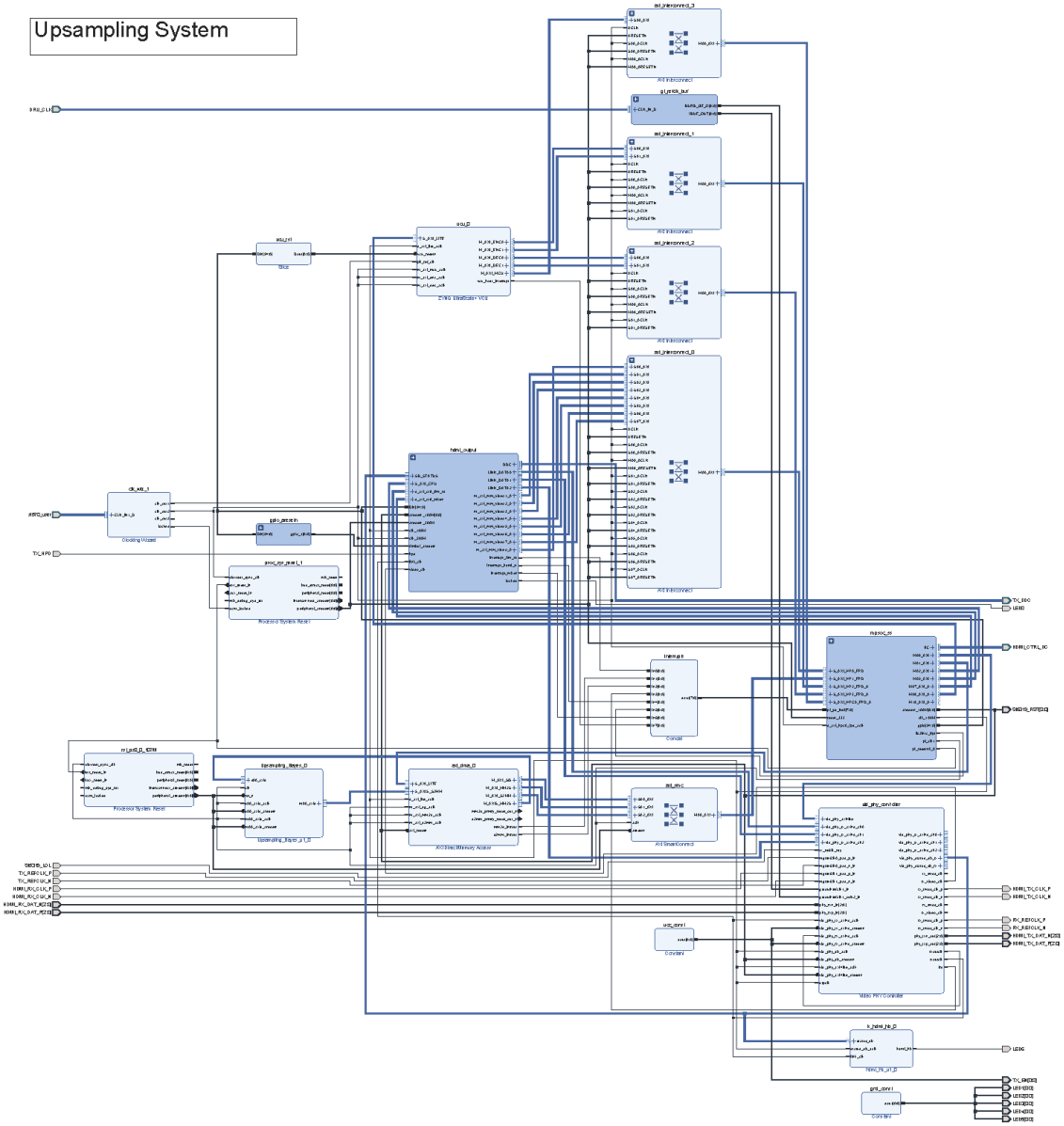

Fig. 5.8 系统架构

5.2.1. HDMI 显示通路

5.2.1.1. 架构

HDMI通路基于 Xilinx Zynq UltraScale+ MPSoC VCU TRD 2019.2 （hdmitx） 搭建而成，其block design如下图所示，该开源设计的具体细节不再赘述。

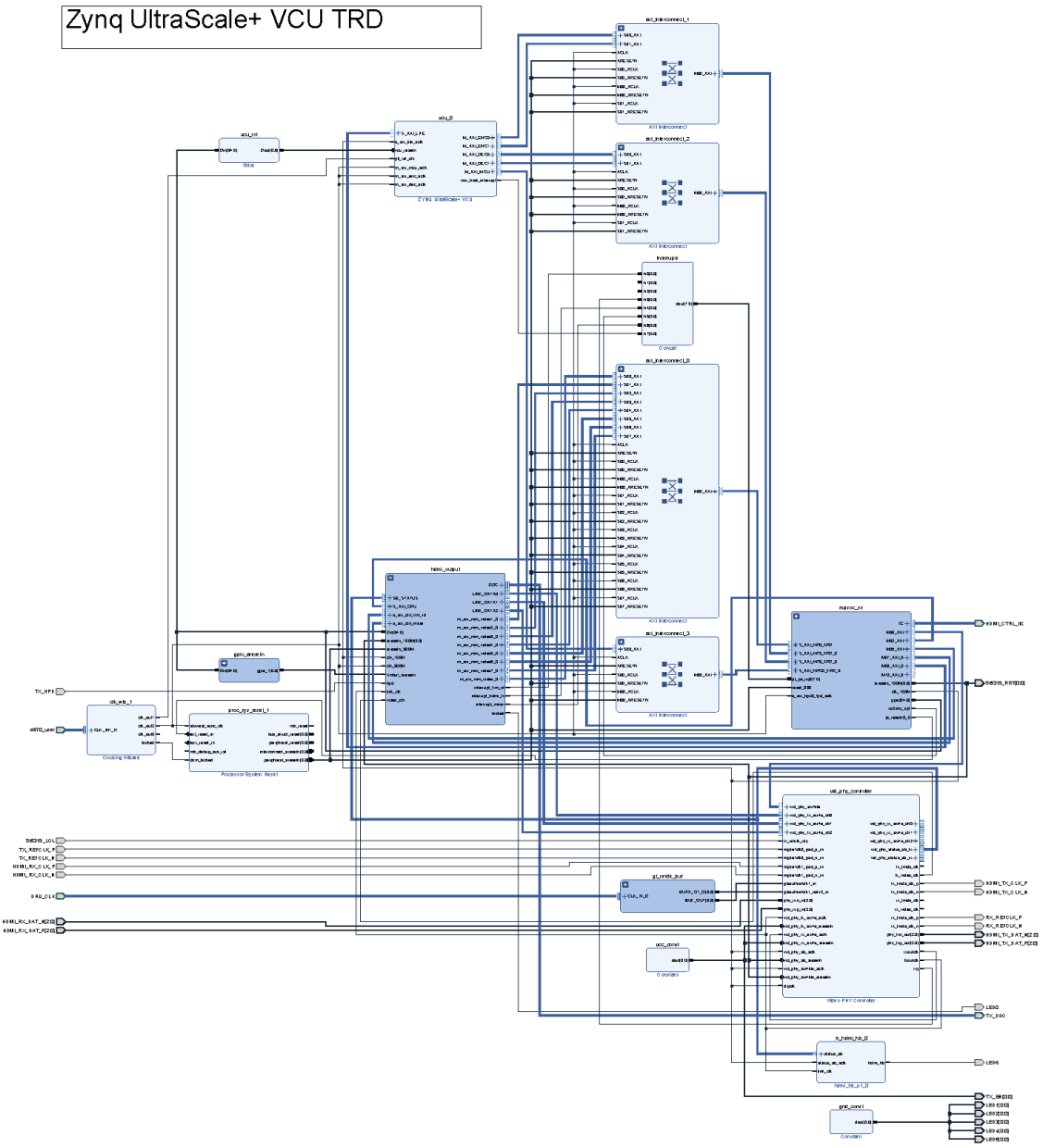

Fig. 5.9 VCU 架构（局部）

5.2.2. IP 通路

5.2.2.1. 架构

上采样IP：前面提到本IP未设置状态寄存器，因此没有配置与Zyqn 间的 AXI Lite 总线接口，而是 IP-DMA 间通过 MM2S 与 S2MM 两条 AXIS 总线构成回路，进行流式的收发。

DMA：DMA配置为收发双通道，启用 Scatter Gather Engine，buffer 宽度寄存器设置为 26bit；Zynq 通过一条 AXI Lite 驱动 DMA，通过MM2S 与 S2MM 两条 AXIS 总线进行数据传输，并接收通道中断。

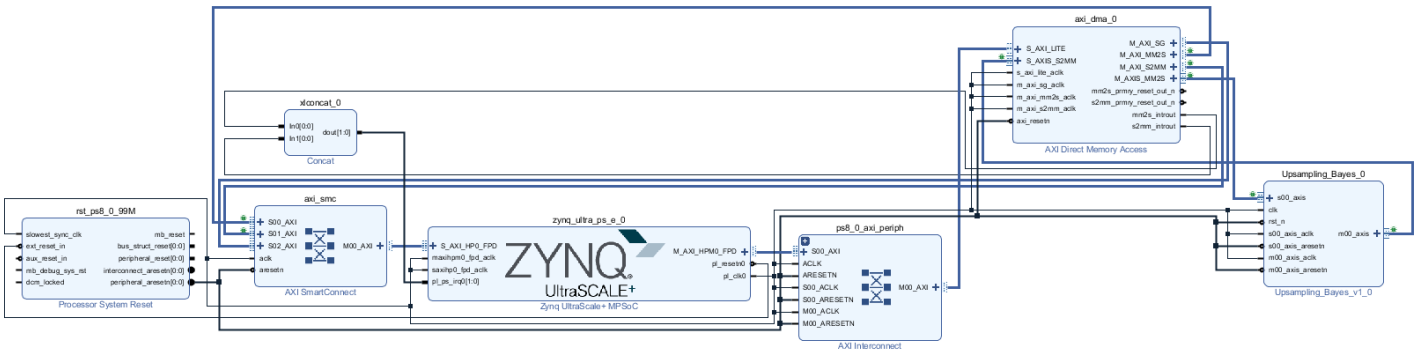

Fig. 5.10 IP-DMA环路（局部）

5.2.2.2. 频率与吞吐率

设计上，IP一次工作周期执行图片两倍放大的功能，因此，DMA-IP的AXIS总线宽32bit，IP-DMA的总线宽为128bit，以匹配输入输出带宽。

经过测试，系统中IP综合最高频率为322.19MHz，由于IP每时钟周期将一个像素上采样为四个像素，则极限吞吐率为 $322.19\text{MHz} \times 8\text{bit} = 322.19\text{MBps}$，进而等效4K帧率为：

$$322.19 \div (960 \times 540 \times 5) = 124.30\,\text{fps}$$

5.2.3. UART

在不接入 DisplayPort 的情况下片上系统通过 UART 在上位机显示系统终端并进行命令传输。

5.2.4. ETH

由于操作系统接管了网口，在接入有线网络并通过DHCP分配IP后，可以利用`scp`、`tftp`等命令进行以太网传输。

5.3. 软件流程

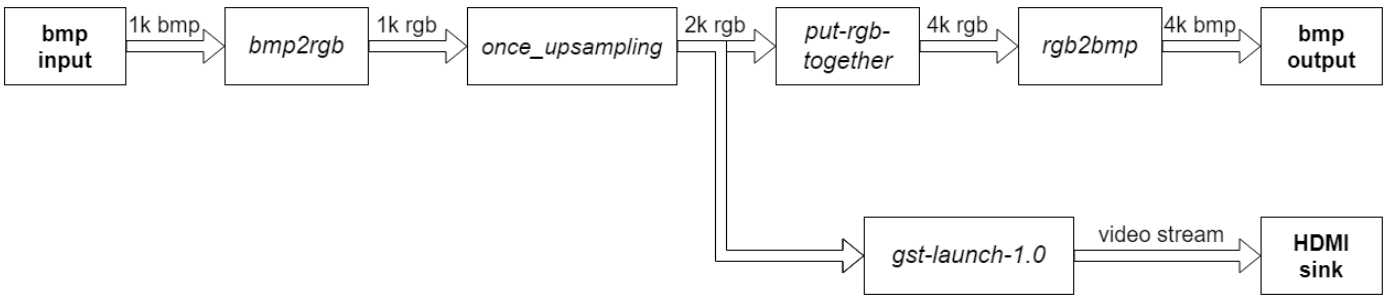

Fig. 5.12 软件流程

上采样的软件系统流程如上所示，空心箭头代表数据流向及其数据格式；斜体代表片上程序；粗体代表输入源与输出沉——根据赛题要求，输出4k bmp图片的同时，需要对于所有图片的4k rgb进行展示（对于1080p的屏幕，实际展示为四部分2k rgb）。

下面按重要性依次介绍片上程序：

5.3.1. once_upsampling

本系统的 DMA 驱动基于 GitHub 开源库 bperez77/xilinx_axidma 进行开发。该开源库提供了面向 Xilinx AXI DMA 的一种零拷贝的 Linux 驱动和一套用户空间界面库。利用 Zynq PS 其中一个 DMA 端口，构造了 PS 和 PL 之前的通讯桥梁。该库的分发基于 MIT 证书。

基于该库，我们开发了应用程序 once_upsampling，一次性将所有1k rgb读入，然后调用 DMA 驱动函数`axidma_twoway_transfer`将之送入 IP，得到2k rgb后将之分割为4张1k rgb，再调用4次`axidma_twoway_transfer`，得到目标4k图像四角的2k rgb。下面是其他的一些技术细节。

5.3.1.1. DMA 驱动配置

5.3.1.1.1. 设备树

配置用户（system-user.dtsi）设备树如下，将dma设备挂载到总线，注册收发通道并赋予通道一个唯一 ID 供驱动调用：

```
&amba_pl {
  axidma_chrdev: axidma_chrdev@0 {
    compatible = "xlnx,axidma-chrdev";
    dmas = <&axi_dma_0 0 &axi_dma_0 1>;
    dma-names = "tx_channel", "rx_channel";
  };
};

&axi_dma_0{
  dma-channel@a0000000 {
    xlnx,device-id = <0x0>;
  };
  dma-channel@a0000030 {
    xlnx,device-id = <0x1>;
  };
};
```

5.3.1.1.2. 驱动与库编译

开源库代码支持树外交叉编译，而不必编译到内核中，指定内核源码树的路径并指定交叉编译器：

```
# Cross Compilation Options
CROSS_COMPILE = aarch64-linux-gnu-
ARCH = arm64
# Build Options
KBUILD_DIR = <path/to/linux-xlnx-4.19-xilinx-v2019.2+git999>
```

编译得到模块 `axidma.ko` 与动态链接库 `libaxidma.so`，加载模块并链接动态库后便可以使用 DMA 的用户空间函数，例如上述提到的双向传输函数`axidma_twoway_transfer`，封装与抽象极大的简化了开发流程。

5.3.1.2. 程序性能

如前所述，IP频率为214MHz，极限帧率为82.66fps。经过测试，调用`axidma_twoway_transfer`函数进行一次两倍上采样用时2.53ms，则驱动的极限帧率为：

$$1\text{s} \div (2.53\text{ms} \times 5) = 79.05\,\text{fps}$$

而由于实际操作的是文件而非纯粹的数据流，对于22张图片，读取与上采样一共耗时1.31s，写回耗时6.32s，不难发现瓶颈主要在IO操作。若仅考虑前者时间，则软件系统的帧率为 $22 \div 1.31 = 16.79\text{fps}$。

5.3.2. gst-launch-1.0

GStreamer 是一个基于管道的多媒体框架，基于GObject，以C语言写成。 凭借GStreamer，程序员可以很容易地创建各种多媒体功能组件，包括简单的音频回放，音频和视频播放, 录音，流媒体和音频编辑。 GStreamer 遵循 GNU 通用公共许可协议。

由于显示并非赛题主要求，为了简便、稳定起见这里不编写C程序而采用Gstreamer 提供的命令行工具 gst-launch-1.0，构造传输视频内容的pipeline如下：

```
gst-launch-1.0 multifilesrc location="./output/rgb/disp-%04d.rgb" \
    ! rawvideoparse width=1920 height=1080 framerate=1/1 format=GST_VIDEO_FORMAT_BGR \
    ! videoconvert ! autovideosink
```

pipeline 的 source 来自于由1k rgb两次超分得到的4张2k rgb，也就是一张4k图片分成四部分，恰好在1080p的屏幕上得以展示；经过原始视频解码，规定流的宽、高、帧率、编码形式，形成一段1080p，1fps的幻灯片视频；最终输出到默认的视频沉，在系统中也就是HDMI sink。

5.3.3. 其他程序

其他程序均为自主编写的符合gnu99的c程序，其不执行任何对于像素的算术运算，仅对于像素流进行重排或者文件转换。

5.3.3.1. bmp2rgb

bmp文件分为如下三个部分

Table 5.1 bmp文件结构

块名称	结构体定义	大小（Byte）
bmp文件头	BITMAPFILEHEADER	14
bmp信息头	BITMAPINFOHEADER	40
图像数据	BYTE*	由图像长宽尺寸决定

因为IP处理大小为960*540的bmp图像的rgb色彩数据，不需要bmp的文件头和信息头数据，因此需要捕捉bmp的图像数据，生成仅含图像色彩数据的rgb文件。生成方式为自bmp文件起点跳过54字节至图像数据，将所有图像数据保存在新的rgb文件中。

5.3.3.2. rgb2bmp

IP对960*540的rgb数据四倍上采样，生成3840*2160的rgb数据，rgb数据只包含图像的色彩信息，我们需要bmp的文件头和信息头生成完整的bmp图像，文件头和信息头的各参数如下

Table 5.2 bmp文件头BITMAPFILEHEADER

字段名	大小（Byte）	值
bfType	2	0x4D42
bfSize	4	54+3840*2160*3
bfReserved1	2	0
bfReserved2	2	0

字段名	大小（Byte）	值
bfOffBits	4	54

Table 5.3 bmp信息头BITMAPINFOHEADER

字段名	大小（Byte）	值
biSize	4	40
biWidth	4	3840
biHeight	4	2160
biPlanes	2	1
biBitCount	2	24
biCompression	4	0
biSizeImage	4	3840*2160*3
biXPelsPerMeter	4	0
biYPelsPerMeter	4	0
biClrUsed	4	0
biClrImportant	4	0

5.3.3.3. put-rgb-together

IP最终的输出为4份1920*1080的rgb流数据，对应bmp图像数据的左上、右上、左下、右下部分。将4份rgb流数据拼接得到最终的4krgb流数据。

5.4. 脚本及演示

为了进一步简化、可视化 demo 流程，我们使用QT开发 GUI 对串口进行了包装。

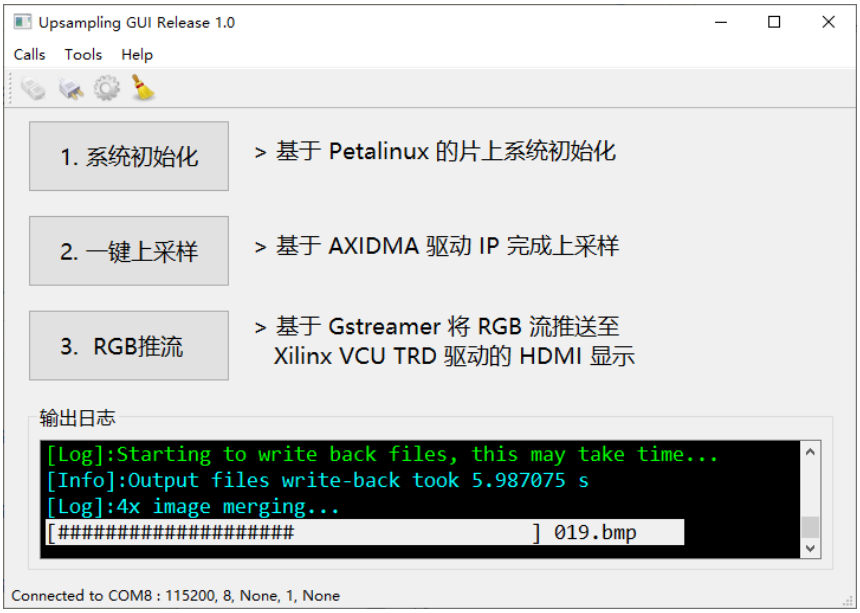

Fig. 5.13 Upsampling GUI Release 1.0

首先将反复调用的多条指令以及上述的软件流程封装进脚本，通过单语句调用执行所有任务，进一步用GUI的按钮将脚本执行进行封装。

1. 系统初始化（`. /media/cad/init.sh`）：从 boot 启动后自动挂载SD卡上的第二分区并执行其他的一些初始化任务。
2. 一键上采样（`./once_upsampling.sh -f`）：执行图2.1上方水平流程，遍历input文件夹，将所有的bmp转换成rgb交由once_upsampling程序处理，并最后统一写回、拼合、输出4k图片。
3. RGB推流（`./rgb2video.sh`）：执行图2.1下方水平流程，构建pipeline，将第二步留下的中间文件推流至HDMI sink。

6. 性能评估

6.1. 超分辨率指标表现

6.1.1. 算法评估

对官方提供的测试集进行上采样处理，对上采样结果进行算法评估，计算其SSIM及PSNR，得到的均值结果如下表。

Table 6.1 psnr,ssim对比

	基准分数	matlab双线性	matlab双三次	我们的IP
psnr	28.360	30.460	31.519	32.153
ssim	0.800	0.837	0.862	0.876

实现的算法效果明显优于基准上采样结果。

6.1.2. 模型评估

对官方提供的测试集进行上采样处理，对上采样结果进行模型评估，得到其lpips值，得到的均值结果如下表

Table 6.2 lpips对比

	基准分数	matlab双线性	matlab双三次	我们的IP
lpips	0.284	0.330	0.273	0.200

实现的算法效果明显优于基准上采样结果。

6.2. 超分辨率实时性

6.2.1. 行延迟

- Bicubic Core 需要缓存四行数据后开始流式处理，并将产生的超分辨率图像的 CbCr 通道数据缓存在 CbCr Pipelined Buffer， 将 Y 通道数据缓存在 Bayes Core 行缓存内。
- Bayes Core 需要缓存六行数据后开始流式处理，并将修正后的 Y 通道数据缓存在 Overlap Buffer 中。在 Overlap Buffer 在像素总数超过两行后流式输出。
- 当 Overlap Buffer 流式输出时，与 CbCr Pipelined Buffer 内对应的数据组合后输出到通道转换模块 YCbCr to RGB 模块并流式输出到输出接口。

综上所述，输出数据和输入数据的行延迟为 8 行。

6.2.2. 吞吐量/系统帧率

当系统正常工作时，系统的数据处理为流式处理的。

IP系统使用 2.2 ns 的时钟，时序收敛。时钟频率为 454.54MHz，换算为帧率，输出图像帧率达到 167.7 帧。

与其他机器学习上采样的IP进行对比，得到如下表格。

Table 6.3 吞吐率对比

	频率 (MHz)	**分辨率**	**阵列**	**吞吐率(Mpixels/s)**
Yang et al[1]	136	1920×1080	60	124
Lee et al[2]	431	3840×2160	30	431
Kim et al [3]	150	3840×2160	60	600
我们的IP	330	3840×2160	120	1738.8

实现的硬件吞吐率明显高于其他机器学习上采样的IP硬件。

6.2.3. 硬件效率

将本文实现的硬件效率与其他机器学习的上采样IP的硬件效率进行对比，得到如下表格。

Table 6.4 逻辑资源对比

资源	**Yang's at el[4]**	**我们的IP**
REG	561139	87500
LUT	477450	62554
DSP	3052	911

在硬件效率上，实现的硬件算法硬件效率远高于其他机器学习上采样IP。

7. FPGA 验证报告

FPGA验证软件：Vivado 2019.2
FPGA验证器件：Zynq UltraScale+ ZCU106 Evaluation Platform (xczu7ev-ffvc1156-2-e)

7.1. IP 报告

7.1.1. 时钟频率

使用 **2.2ns(454.54MHz)** 时钟对IP进行综合，建立时间为0.042s，保持时间为0.006s，建立时间和保持时间均收敛。

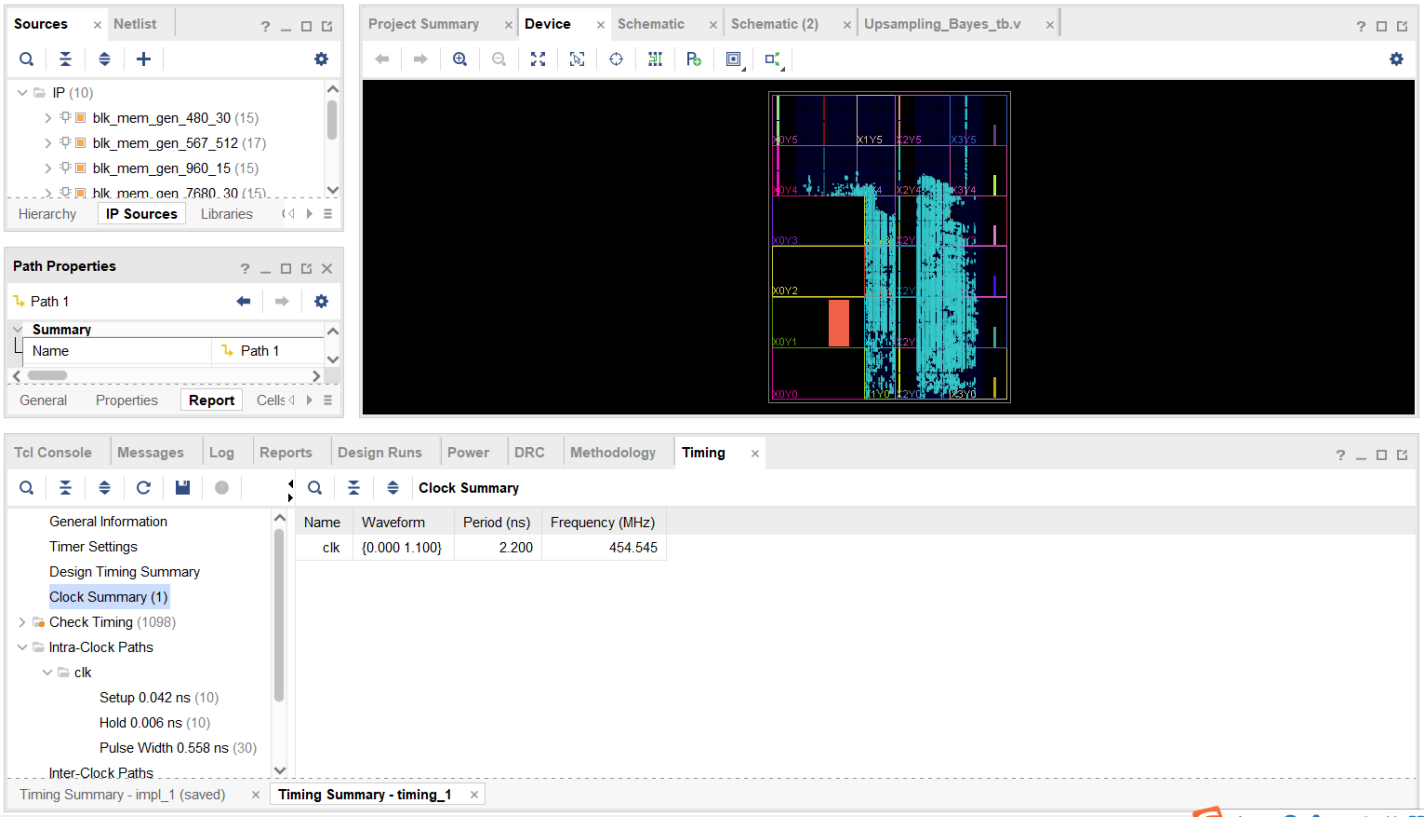

Fig. 7.1 IP 时钟频率

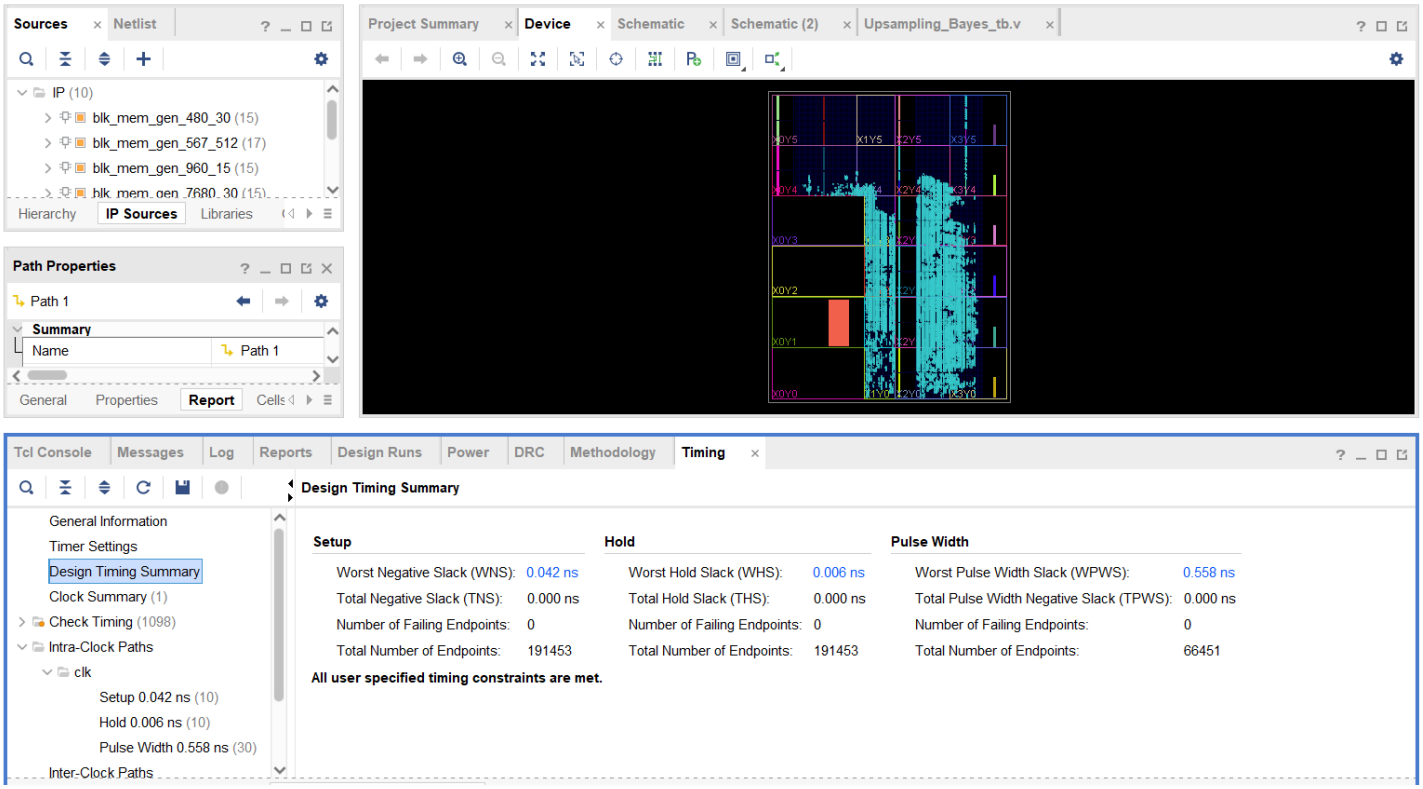

Fig. 7.2 IP 建立时间、保持时间

7.1.2. 资源

系统消耗的资源如图所示。使用 LUT 43.4K，使用REG 62.7K，使用 DSP 733个。

Name	CLB LUTs (230400)	CLB Registers (460800)	CARRY8 (28800)	F7 Muxes (115200)	CLB (28800)	LUT as Logic (230400)	LUT as Memory (101760)	Block RAM Tile (312)	DSPs (1728)
N Upsampling_Bayes	43453	62721	3891	4	9821	40608	2845	100.5	733
pixel_high_2_M_AXIS_0 (pi	173	198	0	0	62	173	0	0	0
S_AXIS_2_pixel_low_0 (S_	37	51	0	0	7	37	0	0	0
Super_Res_0 (Super_Res)	43178	62306	3888	4	9800	40357	2821	100.5	733
Upsampling_Bayes_M00_A	22	156	3	0	44	22	0	0	0
Upsampling_Bayes_S00_A	35	10	0	0	7	11	24	0	0

Fig. 7.3 FPGA 验证结果——资源消耗

7.2. 系统报告

7.2.1. 时序报告

参考文档《5.仿真验证环境说明》，系统中IP-DMA环路均采用与HDMI系统独立的时钟，即由ZYNQ MPSOC发出的`pl_clk_1`时钟域。经过测试，系统时序收敛的极限频率为322.19MHz，如图2.1.1所示。这是由于ZYNQ总线频率不能超过333.33MHz，经过PLL实际得到的极限频率就是322.19MHz，IP的运行频率已经不是系统瓶颈。

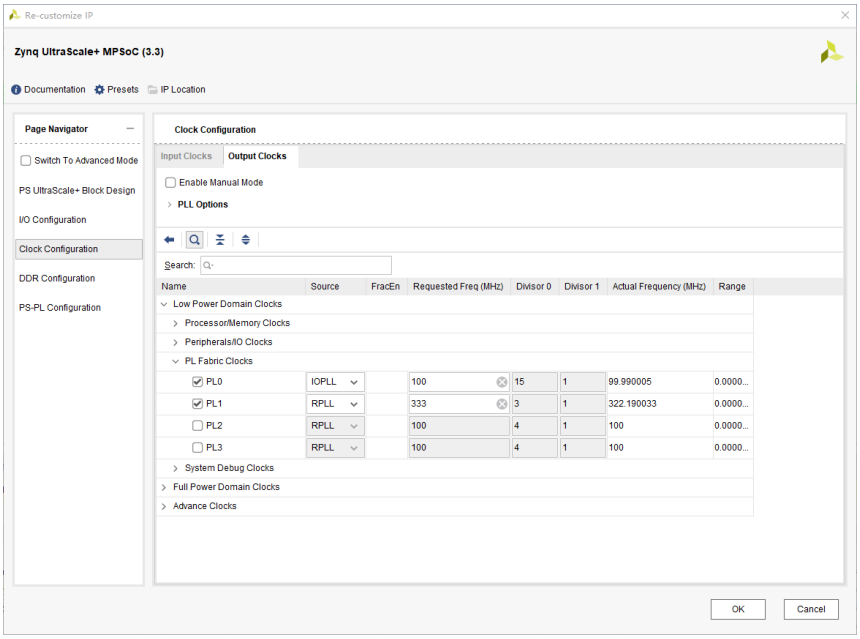

Fig. 7.4 系统时序报告汇总

如图2.1.2所示，建立时间裕量为0.073ns。

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.073 ns	Worst Hold Slack (WHS):	0.010 ns	Worst Pulse Width Slack (WPWS):	0.051 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	478188	Total Number of Endpoints:	477594	Total Number of Endpoints:	161479

All user specified timing constraints are met.

Fig. 7.5 系统时序报告汇总

7.2.2. 资源报告

系统资源用量如图2.2.1所示：

Name	CLB LUTs (230400)	CLB Registers (460800)	CARRY8 (28800)	F7 Muxes (115200)	F8 Muxes (57600)	CLB (28800)	LUT as Logic (230400)	LUT as Memory (101760)	Block RAM Tile (312)	URAM (96)	DSPs (1728)
bd_wrapper	104933	155669	5134	1797	685	21520	95231	9702	186.5	44	790
bd_i (bd)	104933	155669	5134	1797	685	21520	95231	9702	186.5	44	790
axi_dma_0 (bd_axi_dma_	2892	5993	35	0	0	913	2712	180	3.5	0	0
axi_interconnect_0 (bd_a	1234	623	0	49	0	495	1234	0	0	0	0
axi_interconnect_1 (bd_a	2024	3600	0	0	0	933	2020	4	0	0	0
axi_interconnect_2 (bd_a	2030	3600	0	0	0	831	2026	4	0	0	0
axi_interconnect_3 (bd_a	824	1584	2	32	0	413	752	72	0	0	0
axi_smc (bd_axi_smc_0)	3744	6273	0	0	0	1008	2663	1081	0	0	0
clk_wiz_1 (bd_clk_wiz_1_	0	0	0	0	0	0	0	0	0	0	0
gnd_const (bd_gnd_cons	0	0	0	0	0	0	0	0	0	0	0
gpio_aresetn (gpio_ares	0	0	0	0	0	0	0	0	0	0	0
gt_refclk_buf (gt_refclk_b	0	0	0	0	0	0	0	0	0	0	0
hdmi_output (hdmi_outpu	33690	50251	1034	1617	660	7152	29382	4308	70.5	0	57
interrupts (bd_interrupts_	0	0	0	0	0	0	0	0	0	0	0
mpsoc_ss (mpsoc_ss_ir	12529	15500	12	8	4	2647	11323	1206	0	0	0
proc_sys_reset_1 (bd_pr	14	36	0	0	0	11	13	1	0	0	0
rst_ps8_0_107M (bd_rst_	14	34	0	0	0	12	13	1	0	0	0
tx_hdmi_hb_0 (bd_tx_hdr	22	25	3	0	0	11	22	0	0	0	0
Upsampling_Bayes_0 (b	42509	60583	3891	4	0	10219	39664	2845	112.5	0	733
vcc_const (bd_vcc_const_	0	0	0	0	0	0	0	0	0	0	0
vcu_0 (bd_vcu_0_0)	352	1352	0	0	0	250	352	0	0	44	0
vcu_rst (bd_vcu_rst_0)	0	0	0	0	0	0	0	0	0	0	0
vid_phy_controller (bd_vi	3090	6215	157	87	21	955	3090	0	0	0	0

Fig. 7.6 系统时序报告汇总

可以看到 IP 在整个系统内作为算力单元，LUT 用量占 block design 40.5%，Reg 用量占 block design 38.9%，但DSP 用量占用 block design 92.7%，同时其参数矩阵（ROM）与行缓存（RAM）共同占用了 block design 60.3% 的 BRAM 用量，这证明我们侧重于对 IP 的 DSP 与 BRAM 用量优化具有显著意义。