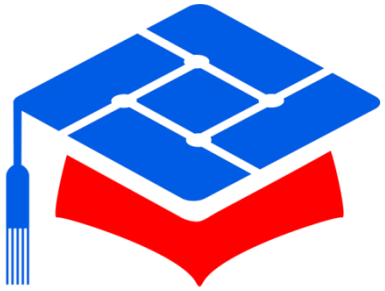


第六届

全国大学生集成电路创新创业大赛

报告类型*: 技术文档

参赛杯赛*: 景嘉微杯

作品名称*: 一种应用于图形显示的 Upsampling IP

队伍编号*: CICC1215

团队名称*: 虹ヶ咲学园芯片设计同好会

技术文档

——第六届全国大学生集成电路创新创业大赛景嘉微杯分赛区决赛提交文档

队名：虹ヶ咲学园芯片设计同好会

成员：黄金源 邓立唯 林明锋

2022 年 7 月 29 日

前言

本技术文档仅作为虹ヶ咲学园芯片设计同好会（成员：黄金源、邓立唯、林明锋）参加第六届全国大学生集成电路创新创业大赛景嘉微杯赛分赛区决赛提交文档供评委评分使用。

虹ヶ咲学园芯片设计同好会

2022 年 7 月 29 日

目录

第一章 算法设计说明 **1**

1.1 概要 1

1.1.1 背景介绍 1

1.1.2 赛题分析 1

1.2 算法分析 2

1.2.1 下采样模型概括 2

1.2.2 双三次插值算法介绍 2

1.2.3 高斯-拉普拉斯算子介绍 3

1.2.4 局部二值模式介绍 4

1.2.5 RAISR 算法 5

1.3 算法优化 8

1.3.1 概述 8

1.3.2 实现步骤 9

1.4 效果展示 10

1.5 总结 10

第二章 实现函数说明 **13**

2.1 基本介绍 13

2.1.1 整体流程 13

2.1.2 数据结构 13

2.2 实现函数细节 16

2.2.1 超分辨率实现 16

2.2.2 锐化 18

2.2.3 其余组件 18

第三章 寄存器说明 23

第四章 RTL模块设计说明 25

4.1 概要 25
4.2 上采样模块 26
4.3 纹理分类模块 26
4.3.1 运算位宽与量化 26
4.3.2 高斯卷积核 27
4.3.3 拉普拉斯卷积核 30
4.3.4 纹理分类器 31
4.3.5 行缓冲模块 35
4.4 自适应锐化模块 38
4.4.1 运算位宽与量化 38
4.4.2 5×5 卷积单元 39
4.4.3 滤波器参数存储单元 40
4.4.4 关于设计补充 40
4.5 色域转换模块 40
4.5.1 RGBtoYUV Unit 41
4.5.2 YUVtoRGB Unit 42
4.5.3 注意 43
4.6 总结 43

第五章 仿真验证环境及说明 44

5.1 验证工具 44
5.2 验证方法及策略 44
5.3 验证范围 45
5.3.1 各子运算单元 45
5.3.2 Bicubic 上采样模块 45
5.3.3 纹理分类模块 45
5.3.4 自适应锐化模块 46

5.3.5 色域转换模块 46
5.3.6 注意事项 46
5.4 验证环境 47
5.4.1 验证平台 47
5.4.2 验证计划 47
5.4.3 待验证设计 47
5.4.4 完成验证设计 48
5.5 覆盖率 48
5.5.1 各子运算单元 48
5.5.2 Bicubic 上采样模块 48
5.5.3 纹理分类模块 48
5.5.4 自适应锐化模块 48
5.5.5 色域转换模块 48
5.6 验证分析 49
5.6.1 各子运算单元 49
5.6.2 Bicubic 上采样模块 49
5.6.3 纹理分类模块 49
5.6.4 自适应锐化模块 49
5.6.5 色域转换模块 49

第六章 性能评估说明 **50**
6.1 概述 50
6.2 Bicubic 上采样模块性能评估 50
6.2.1 性能参数 50
6.2.2 资源使用量 51
6.3 纹理分类模块性能评估 52
6.3.1 性能参数 52
6.3.2 资源使用量 53
6.4 自适应锐化模块性能评估 53
6.4.1 性能参数 53
6.4.2 资源使用量 54

第七章 FPGA 验证报告 **55**

7.1 概述 55

7.2 验证方案 55

7.2.1 验证平台介绍 55

7.2.2 验证框架介绍 55

7.2.3 验证流程介绍 55

7.2.4 上位机介绍 56

7.3 Demo 展示 57

参考文献 **59**

附录 A **62**

附录 B **110**

附录 C **112**

附录 D **114**

附录 E **116**

第一章　算法设计说明

1.1　概要

1.1.1　背景介绍

单幅图像超分辨率重建是指将一副低分辨率图像通过特定算法处理获得高分辨率图像的的一种技术。

图像超分辨率重建一直以来都是图像处理领域中一个重要的研究方向之一，在医学、遥感、图像识别、网络媒体传输、动画制作与合成等领域都有着重要的应用。目前已有许多优秀的算法应用于现实的产品上，但是有效保持图像纹理细节且使图像边缘区域不失真，同时兼顾处理速度一直是图像超分辨率重建技术的一个难题。

在 GPU 处理领域，为了减轻像素引擎的负载，通常使用渲染低分辨率后经由算法放大至期望图像大小的方法已经成为一个被广泛使用的性能优化手段。实现一种对帧存颜色缓冲区图像的超采样处理 IP，用于硬件性能不足时将低分辨率图像放大至高分辨率，从而在尽可能贴合高分辨率渲染效果的同时提升帧率、降低功耗。

1.1.2　赛题分析

在本赛题中，由于举办方提供的图像样例仅包含像素信息，缺失了渲染相关的运动信息、位置信息、光线信息等，故可将本赛题视作 Single Image Super Resolution(SISR) 任务。由于比赛方限制使用基于神经网络设计的算法，我们队伍将基于传统算法结合机器学习的方法进行图像超分辨率核心算法设计。

1.2　算法分析

1.2.1　下采样模型概括

在 SISR 任务中，实际上是完成了一个对低分辨率输入图像 LR 进行高分辨率图像 HR 的预测过程。图像的线性退化模型可以表示为式1.1：

$$z = D_s H x + n \tag{1.1}$$

其中，z 为输入大小为 $M * N$ 的 LR 图像；x 为大小是 $Ms * Ns$ 的待预测 HR 图像；H 是线性模糊核；D_s 为下采样算子，系数 s 是下采样的倍数； n 为额外的噪声。图像超分辨率的任务就是通过 z 恢复出未知的 HR 图像 x。由于在退化模型之中，下采样操作会带来信息损失，同时随着下采样倍数 s 增大，LR 所包含的 HR 信息越少。在这过程中，HR 与 LR 的对应是多对一的关系，完成超分辨率任务则变成了求解一个欠定逆问题，即求得结果可能产生多个输出。因此需要引入先验信息，通过约束获得唯一解。

在本次赛题中，由于给出的测试集为 GPU 渲染图片的 4 倍线性下采样图，公式 1.1 中的线性模糊核 H 则可以消去，同时假设没有引入来自其他的噪声干扰，则公式中噪声 n 也可消去。最终得出得出式 1.2

$$z = D_s x \tag{1.2}$$

1.2.2　双三次插值算法介绍

在数值分析这个数学分支中，双三次插值(Bicubic)[10][11]是二维空间中最常用的插值算法，是三次插值的一个拓展。而在图像处理中，双三次插值算法更是首选。双三次插值需要参考周围 16 个像素(4×4)进行上采样，所得到的上采样图像会比最近邻插值和双线性插值更平滑，同时插值造成的伪影也更少。

作为一个整体，双三次插值算法需要计算插值像素中心与原始像素中心之间的距离。使用与距离存在映射关系的双三次插值函数计算相关系数，然后使用该系数与原始像素点相乘，最终与 16 个原始像素点的乘积之和就是新的插值像素点结果。可以简单地将新插值点看作与原始像素点对应的曼哈顿距离加权，可以用式 1.3 和式 1.4 表

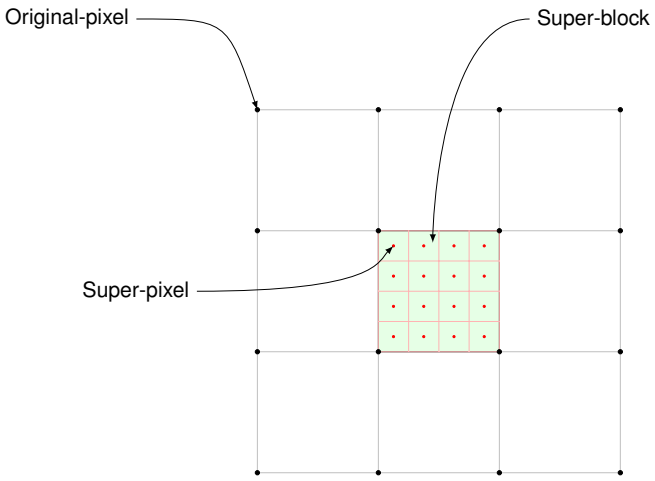


图 1.1: 双三次插值算法简介

示:

$$SP_{\vec{s}} = \sum_{\vec{o} \in RSB} f_B(|\vec{s} - \vec{o}|) \cdot OP_{\vec{o}} \tag{1.3}$$

$$f_B(|x|) = \begin{cases} (\alpha+2)|x|^3 - (\alpha+3)|x|^2 + 1 & , if\ |x| \leq 1 \\ \alpha|x|^3 - 5\alpha|x|^2 + 8\alpha|x| - 4\alpha & , if\ 1 < |x| < 2 \\ 0 & , otherwise \end{cases} \tag{1.4}$$

其中，$SP_{\vec{s}}$ 为插值后像素，$\vec{s}$ 是对应的索引向量。RSB 为当前插值像素所在的插值块对应的原始像素的索引集合。 $OP_{\vec{o}}$ 为原始像素，$\vec{o}$ 是对应的索引向量。f_B 则是双三次插值的核心函数。

由于双三次插值算法在我们算法设计中为预处理部分，在实现上我们着重关注在硬件设计上，并针对性的进行了大量优化，具体请参考 **附录 A** 介绍，此处不展开介绍。

1.2.3　高斯-拉普拉斯算子介绍

概述

拉普拉斯算子是图像二阶空间导数的二维各向同性测度。图像的拉普拉斯算子强调快速强度变化的区域，因此经常用于边缘检测。拉普拉斯算子通常应用于近似高斯平滑滤波器进行平滑后的图像，以降低其对噪声的敏感性，因此这两种变体将在这里

一起描述。该算子通常以单个灰度图像作为输入，生成另一个灰度图像作为输出。在本算法设计中，在该算子后引入一个阈值控制输出，从而实现图像二值化效果。

原理

记图片像素的强度值为$I(x,y)$，其所对应的拉普拉斯算子$L(x,y)$如式 1.5 所示：

$$L(x,y)=\frac{\partial^2 y}{\partial x^2}+\frac{\partial^2 y}{\partial y^2} \tag{1.5}$$

这个过程可以采用卷积核进行计算。由于图片是采用离散的像素值集合进行表示的，因此我们可以寻找一个近似于拉普拉斯算子的二阶导数离散卷积核。比如如下所示的两个卷积核：

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

表 1.1: 近似于拉普拉斯算子的二阶导数离散卷积核

以上的两个卷积核均可以近似为拉普拉斯算子。因为这两个卷积核都是对图片二阶导数的近似估计，它们对于图片中的噪声均很敏感。因此，为了解决这一问题，我们在进行拉普拉斯操作之前先对图像进行高斯平滑滤波处理，二维的高斯平滑卷积核可以用式 1.6 进行表示。

$$G_\sigma(x,y)=\frac{1}{2\pi\sigma^2}exp(-\frac{x^2+y^2}{2\sigma^2}) \tag{1.6}$$

先利用高斯平滑滤波进行处理，可以降低图片中的高频噪声，方便后续的拉普拉斯操作。

1.2.4　局部二值模式介绍

概述

局部二值模式 (LBP, Local Binary Pattern)[8][9][13] 是一种用来描述图像局部纹理特征的算子；它具有旋转不变形和灰度不变性等显著优点，适合用来提取图像的局部纹理特征。

原理

原始的 LBP 算子[13]定义在 3×3 的窗口内，以窗口中心像素为阈值，将相邻的 8 个像素的灰度值与其进行比较，若周围像素值大于中心像素值，则该像素点的位置被标记为 1，否则为 0。如此，3×3 领域内的 8 个点经过比较可产生 8 位二进制数 (通常转换为十进制数即 LBP 码，共 256 种)，即得到该窗口中心像素点的 LBP 值，并用这个值来反映该区域的纹理信息。可用公式 1.7 来进行描述该过程。

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(g_p - g_c)2^p, \quad s(x) = \begin{cases} 1 & ,x \geq 0 \\ 0 & ,x \leq 0 \end{cases} \tag{1.7}$$

实际上，基于原始 LBP 算子的纹理分类方法过于粗糙，对于图像区域的纹理分类有一定的局限性。后来出现了不少基于 LBP 改进的算法，通过引入新的参数，来获得更好的纹理分类能力。如圆形 LBP 算子[8]具有旋转不变性、经过统一编码 (Uniform Pattern) 后对算子种类进行降维， CLBP 算子[9]引入图像的局部梯度参数以增强纹理分类能力。

1.2.5　RAISR 算法

概述

Google 在 2016 年提出了 **RAISR**[1] 算法，一种基于样本学习的超分辨率算法。RAISR 主要的特点是快速、准确，在运行速度上比基于深度学习的方法有大幅提升，同时保持了具有竞争力的图像重建质量。其核心思想是通过简单的插值方式把 LR 图像转化成 Pre-HR 图像，然后根据 Pre-HR 图像局部的梯度信息来对图像块进行分类，对于不同类别的图像块对应采用不同的与训练的滤波器进行卷积操作对图像纹理进行

增强。在足够的训练数据下，通过学习一组滤波器建立成对的 LR patch与 HR 像素的映射关系。

RAISR 将 patch 分为了 216 类并结合 R^2 个像素类型 (R为上采样因子)，这其中包括了 3类像素强度、24类纹理角度、3类图像块相关性。因此，当上采样因子 R 为4时，便需要 3456 个大小为 11 x 11 的滤波器。通过哈希映射的方式将 patch 分类到不同类别中，而不需要使用较为复杂的聚类方式如 K-means[2]、或者是高斯混合模型[3]，从而降低分类查找时间。在 RAISR 中，哈希表键值是通过计算 patch 的局部梯度统计来获得。接下来我们会分析 RAISR 的一些具体实现。

哈希表键计算方法

通过特征分析[6]来计算用作哈希表键的图像块的局部梯度特征。第 k 个的最近领域通常是一个 $\sqrt{n} * \sqrt{n}$ 的 patch，其中，像素包含于 $k_1, ..., k_n$。每个图像块的局部梯度特征放置在一个 $n \times 2$ 的矩阵 1.8 中：

$$G_k = \begin{bmatrix} g_{x_{k_1}} & g_{y_{k_1}} \\ \vdots & \vdots \\ g_{x_{k_n}} & g_{y_{k_n}} \end{bmatrix} \tag{1.8}$$

在特征分析[6]中，局部梯度特征可以通过奇异值分解 (SVD) 进行计算。求解出来的右向量表示为该图像块的梯度方向，两个奇异值用于表示梯度的强度和相关性。通过一个可分离的归一化高斯核构建了一个对角矩阵 W_k，用于简化特征值分解。根据 $G_k^T W_k G_k$ 矩阵的特征分解得到较大的特征值 λ_1^k 、较小的特征值 λ_2^k 和两个特征向量 ϕ_1^k 和 ϕ_2^k，用来表示梯度的强度 λ_k、角度 θ_k 和相关系数 μ_k，其中

$$\lambda_k = \lambda_1^k \tag{1.9}$$

$$\theta_k = \arctan(\frac{\phi_{1,y}^k}{\phi_{1,x}^k}) \tag{1.10}$$

$$\mu_k = \frac{\sqrt{\lambda_1^k} - \sqrt{\lambda_2^k}}{\sqrt{\lambda_1^k} + \sqrt{\lambda_2^k}} \tag{1.11}$$

这三个哈希表键量化后可以用 λ，θ， μ 来表示，其中

$$\lambda = \lceil \frac{\lambda_k}{Q_s} \rceil \tag{1.12}$$

$$\theta = \lceil \frac{\theta_k}{Q_\theta} \rceil \tag{1.13}$$

$$\mu = \lceil \frac{\mu_k}{Q_\mu} \rceil \tag{1.14}$$

在以上式 1.12 、式 1.13 和式 1.14 中，Q_s、Q_θ、Q_μ 分别代表了强度、角度和相关系数的量化因子。通过这三种参数，构建了 216 个哈希键映射分类。

滤波器学习

在 RAISR 的学习阶段，需要通过建立一个 LR 图像块与 HR 像素点映射的训练数据集，用于学习一个 $d \times d$ 的滤波器 h。滤波器可以通用求解最小二乘法最小值来计算：

$$h = \min_h \sum_{i=1}^{L} \|A_i h - b_i\|_2^2 \tag{1.15}$$

其中，h 是一个 $d^2 \times 1$ 向量表示学习的滤波器，A_i 是由 Pre-SR 图像 y_i 中提取图像块 $d \times d$ 后组成的大小为 $MN \times d^2$ 矩阵， b_i 是由 HR 图像 x_i 中提取的像素点所构成，对应 y_i 图像块的中心坐标。在论文[1]中，作者为了降低求解滤波器的运算量和数据存储，将式1.15拓展为：

$$h = \min_h \|Qh - V\|_2^2 \tag{1.16}$$

其中 $Q = A^T A$，$V = A^T b$。经过这样的转换可以降低数据存储空间以及求解时候的运算，可以表示为:

$$Q = A^T A = \sum_{i=1}^{L} A_i^T A_i \tag{1.17}$$

$$V = A^T b = \sum_{i=1}^{L} A_i^T b_i \tag{1.18}$$

其中，Q 是一个 $d^2 \times d^2$ 的小矩阵和 V 是一个 $d^2 \times 1$ 的列向量。到此滤波器的学习介绍就此结束。

CT变换

由于在学习滤波器的时候会引入锐化操作，在使用该滤波器应用于简易插值后的图像，可能会引起结构变形。为了保留重要的结构，作者引入了 CT变换[7]，作用于简易插值后的图像与滤波后的图像。具体变换如图 1.2 所示： 通过一个 3×3 的图像块的周围像素与中心像素构建出一个布尔类型进行比较，然后以汉明距离来计算每个像素的变化位数。由于结构的变化是取决于汉明距离的，可以根据变化的位数来确定权重

a	d	g
b	e	h
c	f	i

(a)

$e > a$	$e > d$	$e > g$
$e > b$	–	$e > h$
$e > c$	$e > f$	$e > i$

(b)

100	80	90
105	95	85
110	120	170

(c)

0	1	1
0	–	1
0	0	0

(d)

8 bit string							
a	b	c	d	f	g	h	i
0	0	0	1	0	1	1	0

(e)

图 1.2: CT变换

用于加权插值图像与滤波图像，从而得到输出图像。至此，RAISR 的核心设计思路已经清晰。

1.3　算法优化

通过上节我们可以看到，RAISR 作者对于滤波器的学习以及应用进行了不少的优化，但是对于我们将该算法应用到硬件实现上，有着不小的挑战。第一个是在于，每个像素类型需要学习 216 个滤波器，而对于进行 4 倍上采样的整幅图像可分为 16 种类型，总共就需要 3456 个滤波器进行片上存储。在 FPGA 上存储如此大量的数据是不划算的。第二点是，为了能获得像素更多的信息，RAISR 采用了 11×11 大小的滤波器进行卷积，这对于硬件资源消耗量极大，在考虑量化同时保留运算过程中数据合适位宽并满足运算时延，每个像素进行锐化操作至少需要 180 个 DSP 进行乘累加运算(不考虑硬件复用以减少资源消耗)，总延时至少 12 个时钟周期输出延迟，该结果已经考虑了使用树形结构对数据进行运算。第三点，哈希表键的运算不仅需要进行除法、开方及反正切运算，还需要进行奇异值分解，这对于 FPGA 实时性设计要求难度提升了一个等级。

1.3.1　概述

我们提出了一个创新的 **LBP-RAISR** 算法，以 **RAISR** 为主体框架，整体上分为了三个阶段。第一步，通过双三次插值算法[4][5]，将 LR 图像映射到 HR 图像上，得到最终需要的目的分辨率图像 Pre-SR；第二步，对图像以每个像素为中心进行 5×5 分块，遍历 Pre-HR 图像的每一个图像块，进行高斯-拉普拉斯(LoG)纹理检测后使用 CLBP 算法[8][9]结合角度信息求解进行纹理分类。显而易见，这里我们的设计与 **RAISR** 有较大的区别，这是出于对硬件设计的考量，具体的设计细节会在下一节和 RTL设计文档中详细解释。第三步，利用上一步的得到每个图像块的类型找到对应的滤波器，作用于

图像块进行卷积，得到更接近于原始 HR 的像素，最后得到更高质量的 SR 图像。整体框架如图1.3所示。

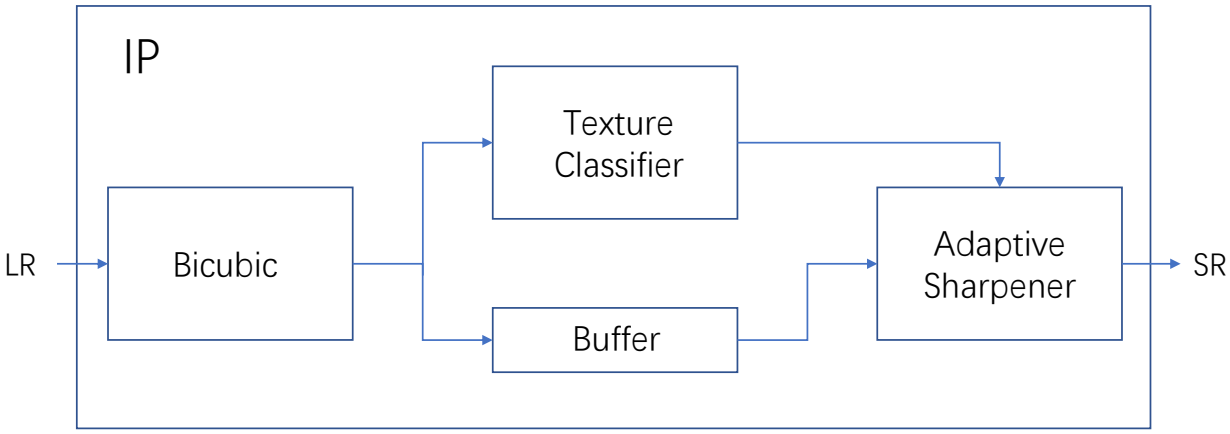


图 1.3: 算法架构

1.3.2　实现步骤

双三次插值

作为第一步，我们需要将 LR 图像先上采样到需要的分辨率，可以使用传统的插值方法如最近邻插值、双线性插值、双三次插值、Lanczos 插值等算法。由于考虑到上采样图片的质量效果会应影响到最终的高分辨率图片质量与运算过程中的尽可能降低复杂度，双三次插值成为了我们算法预处理的最优选择。

纹理分类

我们对经过 4 倍上采样后的 Pre-SR 图片先进行高斯-拉普拉斯滤波(LoG)[12]，然后给定一个阈值区间对进行滤波后的像素值进行二值化操作，采用 CLBP 算法对二值化后的图像纹理进行按块分类。这里需要注意的是，我们采用 5×5 的图像块进行纹理分类，由于已经对图像块进行了纹理检测这一步操作，所以 CLBP 算法步骤我们并没有严格执行，保留 C 中心像素与 u 统一编码这两个参数。考虑到该算法需要对圆形区域检测需要进行三角函数进行运算，我们将其近似等效为最外边缘的二值像素，从而减

轻运算量。但是 CLBP 算法经过统一编码后具有旋转不变形，与滤波器需要学习局部纹理方向角度特征相矛盾，故此我们增加了一步操作，当进行对像素统一编码时，保留了角度信息，从而增加选择滤波器参数条件。此时，纹理分类结束。

卷积输出

在上一步中我们已经通过纹理分类得到了图像块的纹理类型，根据这个类型我们可以选择预学习的滤波器，对相应的 Pre-SR 图像块进行卷积操作，从而得到了 SR 像素，最终输出 SR 图像。

1.4　效果展示

我们将提出的算法与 RAISR 进行了对比, 其中 Set5 是单帧图像超分辨率任务中最常用的数据集，GameSet 是景嘉微在本次杯赛中提供的测试效果图片。

DataSet	**Scale**	**Bilinear**	**Bicubic**	**RAISR**	**LBP-RAISR**	**LBP-RAISR (Quantized)**
Set5(PSNR)	x4	25.79	26.84	27.05	**27.61**	**27.61**
Set5(SSIM)	x4	0.765	0.790	0.803	0.812	**0.817**
Set5(LPIPS)	x4	0.335	0.310	0.269	**0.198**	0.200
GameSet(PSNR)	x4	30.83	31.51	31.59	**32.02**	32.00
GameSet(SSIM)	x4	0.847	0.862	0.861	0.870	**0.871**
GameSet(LPIPS)	x4	0.288	0.265	0.263	**0.209**	0.211

表 1.2: 各算法实现评分对比

1.5　总结

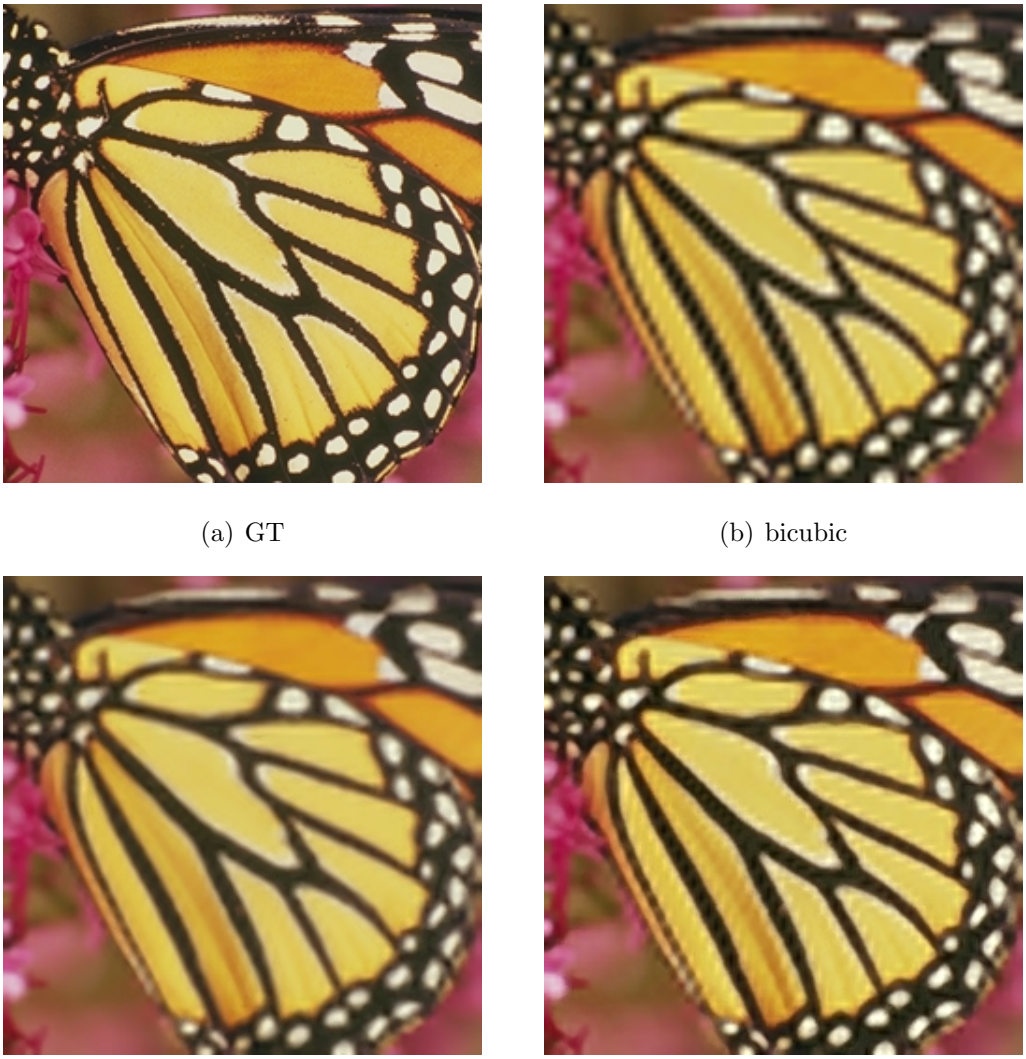
基于 RASIR[1] 的整体思路，我们提出了改进的 LBP-RAISR 算法。LBP-RAISR 背后的核心思想与 RAISR 相同，学习从低分辨率图像到高分辨率图像的映射关系，用于增强经过简单插值后的图像质量。锐化过程是通过使用一组滤波器，对简单插值放大的 Pre-SR 图像进行操作。这些滤波器旨在最小化输入图像和真实图像之间的欧氏距

(a) GT　(b) bicubic

(c) raisr　(d) lbp-raisr

图 1.4: 各算法实现效果对比-例1

离。更具体地说，LBP-RAISR 使用了一种更简便快捷的纹理分类方法，从存储空间上和运算量上做了大量的优化，同时也保留了图像质量的效果。根据大量实验得知，可以较好的提升 RAISR 的性能以及运算时间。经过量化后适配硬件设计并没有损失太大的图像质量。由于我们采用的是基于中心的局部二值模式 (CLBP) 变体与高斯-拉普拉斯滤波 (LoG) 结合的算法进行纹理分类，与 RAISR 中的哈希映射方法相比较，进一步减少了运算量，同时选择滤波器的方法是固定的，这对于硬件上实现是高效便捷的。从更广泛的角度来看，我们希望 LBP-RAISR 中预学习的滤波器映射与训练集的关联影响尽可能小，在这种情况下，学习的过滤器可以用于任意尺寸输入图像进行 $\times 4$ 的超分辨率图像输出。由于运算量的大大降低，让高分辨率图像进行实时的超分辨率任



(a) GT　(b) bicubic

(c) raisr　(d) lbp-raisr

图 1.5: 各算法实现效果对比-例2

务也是可能的。在软件实现过程中，多线程优化下处理一张 4K 图像仅需要 3.6 秒左右(包含图像读写时间)。

第二章　实现函数说明

2.1　基本介绍

2.1.1　整体流程

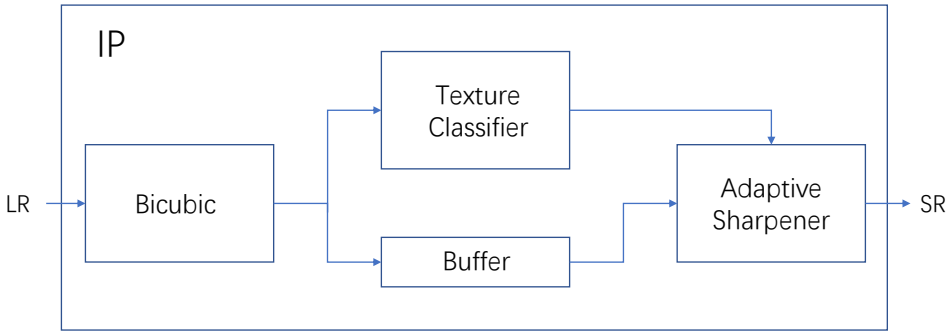


图 2.1: 程序整体流程

2.1.2　数据结构

图片格式

1. **BitmapHead:**包含bmp文件信息头各个信息相对于文件头的偏移量

```
typedef struct {
  uint32_t biSize;
  uint32_t biWidth;
  uint32_t biHeight;
```

```
  uint16_t biPlanes;
  uint16_t biBitCount;
  uint32_t biCompression;
  uint32_t biSizeImage;
  uint32_t biXPelsPerMeter;
  uint32_t biYPelsPerMeter;
  uint32_t biClrUsed;
  uint32_t biClrImportant;
}BitmapHead;
```

2. **Bitmap:**包含bmp图像的文件信息：文件名、文件信息头结构体指针和图像数据指针

```
typedef struct {
  char* fileName;
  BitmapHead *head;
  ImageMat *image;
}Bitmap;
```

3. **ImageMat：** 包含图像的宽度、高度和数据所在的内存首地址

```
typedef struct {
  int width;
  int height;
  uint8_t *pData;
}ImageMat;
```

量化

1. **qint16_t:**带量化位数的16位变量类型，real表示变量真实值，Qn表示该变量需要量化的位数

```
typedef struct {
  int16_t real;
```

```
  uint8_t Qn;
}qint16_t;
```

2. **qint32_t：**带量化位数的32位变量类型，real表示变量真实值，Qn表示该变量需要量化的位数

```
typedef struct {
  int32_t real;
  uint8_t Qn;
}qint32_t;
```

3. **qImageMat：**带量化位数的图像数据存储格式

```
typedef struct {
  int width;
  int height;
  qint16_t* pData;
}qImageMat;
```

4. **QuantizeError：**量化运算成功或失败标志

```
typedef enum {
  OK = 0,
  QN_ALARGE,
  QN_BLARGE
}QuantizeError;
```

5. **qCompareRes：**量化比较结果

```
typedef enum {
  Q_EQ = 0,
  Q_GR,       // >
  Q_LT        // <
}qCompareRes;
```

2.2　实现函数细节

2.2.1　超分辨率实现

1. **双三次插值**

```
void bicubic_interp_fixed(Bitmap* bitmap, Bitmap**
    pOutput);
```

参数:

bitmap: 要进行双三次插值的图像

pBitmap: Bitmap类型的指针变量，用于存储fileName的文件信息

返回值:

无返回值

作用:

对输入图像进行双三次插值，提高图像分辨率

双三次插值内部实现具体可参考 **附录 A**。

2. **上采样——单通道实现**

```
void upsampling(ImageMat* src, ImageMat** dst, qint16_t**
    weight, uint32_t num_instance);
```

参数:

src: 要进行质量增强的图像

dst: 质量增强后的图像

weight: 带有量化位数的权重

num_instance: 需要运行的线程数 **返回值:**

无返回值

作用:

对输入图像进行纹理分类并进行锐化操作，提高图像质量

说明:

在本次设计中，需要先根据patch_edge进行图像分类，再将图像的patch 和 patch_edge 一一对应，执行相应计算。本次设计中，需要处理 4k 分辨率图像，采用顺序执行的方式非常耗费时间，因此该函数采用了并行计算的方式，先将待处理图像分为 num_instance 个部分，每个部分创建一个对应的线程进行并行计算，等待所有线程返回后再执行下一步计算，大大提高了运行速率。

3. **上采样——RGB通道实现**

```
void upsampling_rgb(ImageMat* src, ImageMat** dst, qint16_t
    ** weight, uint32_t num_instance, ImageMat* src_r,
    ImageMat* src_g, ImageMat* src_b, ImageMat** dst_r,
    ImageMat** dst_g, ImageMat** dst_b);
```

参数:

src: 要进行纹理分类的灰度图像

dst: 质量增强后的图像

weight: 带有量化位数的权重

num_instance: 需要运行的线程数

src_r: 需要进行质量提升的R通道图像

src_g: 需要进行质量提升的G通道图像

src_b: 需要进行质量提升的B通道图像

dst_r: 质量增强后的R通道图像

dst_g: 质量增强后的G通道图像

dst_b: 质量增强后的B通道图像

返回值:

无返回值

作用:

对输入图像进行纹理分类并进行锐化操作，提高图像质量

说明:

该函数与上一函数不同，对三个通道同时继续了锐化操作，进而获得更好的整体图像质量。

2.2.2　锐化

2.2.3　其余组件

内存管理

```
void freeqWeightSpace(qint16_t ** weight, int32_t type);
```

参数:

weight：qint_16 类型的指针

type: int32_t 类型的变量

返回值:

无返回值

作用:

用于释放申请的内容空间

图片读取与格式转换

图片处理接口细节可参考 **附录 A**。

1. **ReadBitmap**——读取bmp格式图片

```
BitmapReadError ReadBitmap(const char* fileName, Bitmap**
    pBitmap);
```

参数:

fileName: 要读取的bmp格式图片路径

pBitmap: Bitmap类型的指针变量，用于存储fileName的文件信息

返回值:

BitmapReadError 类型的枚举变量，用于表示读取文件是否成功

作用:

用于读取bmp格式图片

2. **WriteBitmap**——写bmp格式图片

```
BitmapWriteError WriteBitmap(Bitmap* bitmap, const char*
    fileName);
```

参数:

fileName: 要写入的bmp格式图像路径

bitmap: Bitmap类型的指针变量，用于存储fileName的文件信息

返回值:

BitmapWriteError 类型的枚举变量，用于表示写入文件是否成功

作用:

用于写bmp格式图像

3. **ImageMatRGBtoYUV**——将RGB编码图片转换为YUV编码图片

```
void ImageMatRGBtoYUV(ImageMat* mat);
```

参数:

mat: ImageMat类型的结构体指针，用于存储图像的长度、宽度和数据信息

返回值:

无返回值

作用:

将RGB编码图片转换为YUV编码图像

4. **ImageMatYUVtoRGB**——将YUV编码图片转换为RGB编码图片

```
void ImageMatYUVtoRGB(ImageMat* mat);
```

参数:

mat: ImageMat类型的结构体指针，用于存储图像的长度、宽度和数据信息

返回值:

无返回值

作用：

将YUV编码图像转换为RGB编码图像

5. **qImageMatRGBtoYUV**——将量化的RGB编码图片转换为YUV编码图片

```
void qImageMatRGBtoYUV(qImageMat* qmat_r, qImageMat* qmat_g,
    qImageMat* qmat_b, qImageMat* qmat_y, qImageMat* qmat_u,
    qImageMat* qmat_v);
```

参数：

qmat_r: qImageMat类型的结构体指针，用于存储量化图像R通道的长度、宽度和数据信息

qmat_g: qImageMat类型的结构体指针，用于存储量化图像G通道的长度、宽度和数据信息

qmat_b: qImageMat类型的结构体指针，用于存储量化图像B通道的长度、宽度和数据信息

qmat_y: qImageMat类型的结构体指针，用于存储量化图像Y通道的长度、宽度和数据信息

qmat_u: qImageMat类型的结构体指针，用于存储量化图像U通道的长度、宽度和数据信息

qmat_v: qImageMat类型的结构体指针，用于存储量化图像V通道的长度、宽度和数据信息

返回值:

无返回值

作用：

将量化的RGB编码图片转换为量化的YUV编码图像

6. **qImageMatYUVtoRGB**——将量化的YUV编码图片转换为RGB编码图片

```
void qImageMatYUVtoRGB(qImageMat* qmat_y, qImageMat* qmat_u,
    qImageMat* qmat_v, qImageMat* qmat_r, qImageMat* qmat_g,
    qImageMat* qmat_b);
```

参数：

qmat_y: qImageMat类型的结构体指针，用于存储量化图像Y通道的长度、宽度和数据信息

qmat_u: qImageMat类型的结构体指针，用于存储量化图像U通道的长度、宽度和数据信息

qmat_v: qImageMat类型的结构体指针，用于存储量化图像V通道的长度、宽度和数据信息

qmat_r: qImageMat类型的结构体指针，用于存储量化图像R通道的长度、宽度和数据信息

qmat_g: qImageMat类型的结构体指针，用于存储量化图像G通道的长度、宽度和数据信息

qmat_b: qImageMat类型的结构体指针，用于存储量化图像B通道的长度、宽度和数据信息

返回值:

无返回值

作用：

将量化的YUV编码图片转换为量化的RGB编码图像

7. **ImageMatRGBtoYUV**——提取YUV编码格式中Y通道数据

```
ImageMat* splitYChannel(ImageMat* mat);
```

参数:

mat: ImageMat类型的结构体指针，用于存储图像的长度、宽度和数据信息

返回值：

ImageMat 类型指针，存储图片的长度、宽度和YUV编码格式中Y通道数据

作用：

提取YUV编码格式中Y通道数据

8. **mergeYChannel2Image**——将YUV编码格式中Y通道数据合并到图像中

```
void mergeYChannel2Image(ImageMat* img, ImageMat* im_y);
```

参数:

mat: 要合并的目标图像，ImageMat类型的结构体指针，用于存储图像的长度、宽度和数据信息

im_y：ImageMat 类型指针，存储要合并图像的长度、宽度和Y通道数据信息

返回值：

无返回值

作用：

将YUV编码格式中Y通道数据合并到图像中

9. **DestoryImageMat**——释放图像内存空间

```
void DestoryImageMat(ImageMat* mat);
```

参数:

mat: ImageMat 类型的结构体指针，用于存储图像的长度、宽度和数据信息

返回值：

无返回值

作用：

内部调用 free 释放申请的图像内存

第三章　寄存器说明

本 IP 有计划使用寄存器文件进行参数权重配置，

已留有 Dummy 接口用于与 SoC 的 AXI-4 Full 接口互联。

该文档会在下一大版本中进行更新：)

第四章　RTL模块设计说明

4.1　概要

上采样 IP 设计采用了模块化设计，主要分为四大部分：

1. Bicubic 上采样模块
2. 纹理分类模块
3. 自适应锐化模块
4. 色域转换模块

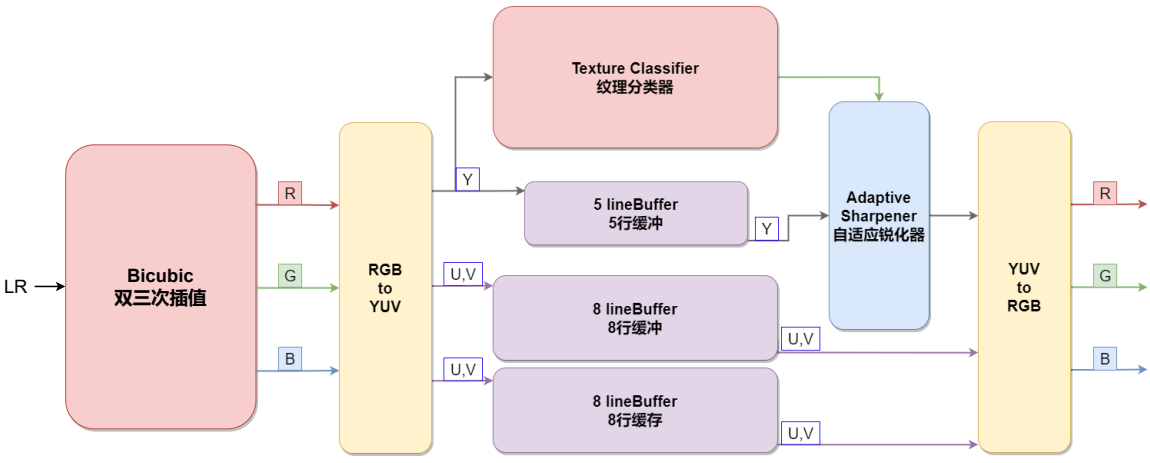


图 4.1: IP 概览

4.2　上采样模块

具体说明详见 **附录 A**。

4.3　纹理分类模块

纹理分类 IP 可提供对于单通道图像每个 5×5 或 3×3 图像块的纹理特征进行实时分类。包含了一个归一化高斯卷积核、一个标准拉普拉斯算子、一个 LBP 分类器、三个行缓冲模块、一个滤波器参数存储单元和一个控制单元。为了确保卷积操作后图像仍保持原始尺寸，行缓冲模块包含了图像填充处理操作与数据映射模块，由控制单元进行管理。

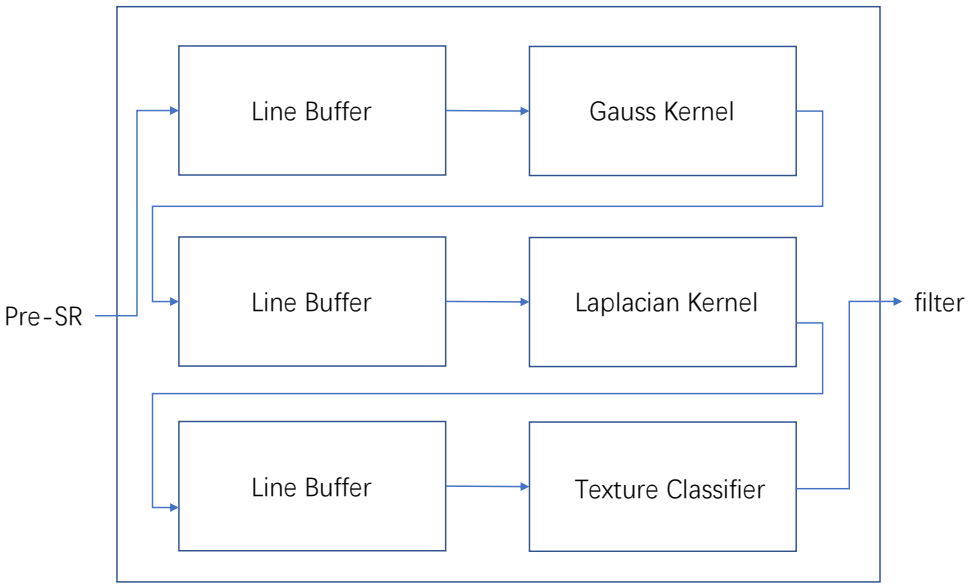


图 4.2: 纹理分类模块结构图

4.3.1　运算位宽与量化

为了节省片上资源的消耗，最大化提升 DSP 利用率，在本模块中，不同部分对于 DSP48E2 均有不同程度上的优化。其中，针对高斯滤波卷积，将核内权重进行 9 位无符号量化，在保证高斯卷积核性能的同时充分利用了 DSP48E2 单元的乘法器。这种量

化方法仅支持 8 位图像输入。另外，为了保证后续拉普拉斯滤波数据误差尽可能降低，我们将高斯滤波后的数据量化为 20 位，尽可能保留数据位宽，避免引入过大误差影响后续操作。在完成拉普拉斯算子卷积后生成的是 1 位图像输出至 LBP 分类器。

4.3.2　高斯卷积核

高斯卷积核包含了一个高斯系数寄存器单元、一组并行乘法器单元、一组并行的加法器单元和一组并行舍入单元。 每个像素进行高斯滤波卷积是以像素本身为中心，

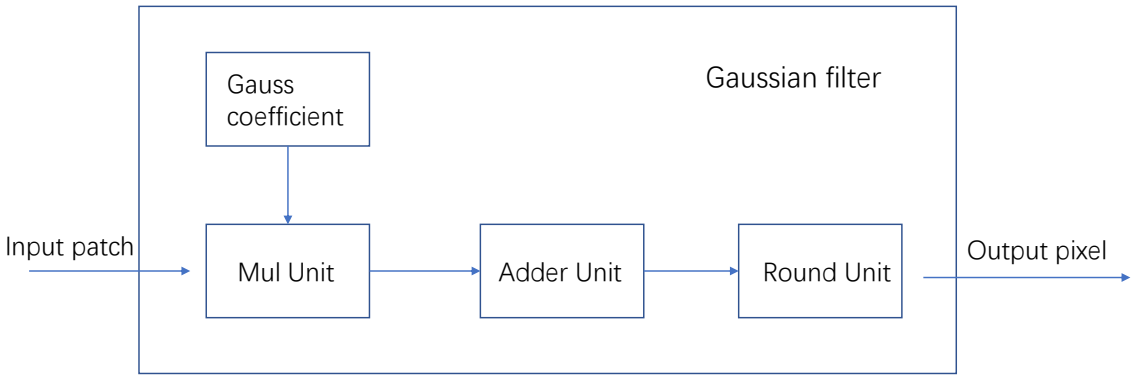


图 4.3: 高斯卷积核结构图

边缘 5×5 的像素块作为数据输入。每个像素块需要与对应的高斯系数进行相乘然后累和。该高斯卷积核 IP 每个时钟周期处理 4 个像素。

高斯系数寄存器单元

高斯滤波卷积核系数是由 σ 所决定，在运行过程中，σ 不会发生改变，所以该系数为常数。同时，5×5 卷积核系数与所在卷积核的位置有关，并意义对应，因此25 个系数只需要存储 6 个系数即可，其余均可通过对称性获取。

乘法器单元

并行乘法单元包含了 7 个 DSP48E2 用于实现 5×5 大小的卷积乘法操作。在两个时钟周期内可输出结果(一周期内也可实现结果输出，但为了提升 IP 最大时钟频率，输出端插入一级寄存器增大时序违例余量)

5	4	3	4	5
4	2	1	2	4
3	1	0	1	3
4	2	1	2	4
5	4	3	4	5

图 4.4: 高斯卷积核数据矩阵数据分布

由于高斯系数具有对称性，当计算图像块对应的位置的时候，存在几对像素所相乘同一个高斯权重系数。例如上图 (0,0)、(0,4)、(4,0)、(4,4) 对应像素均是乘以同一个系数。因此，我们可以利用这一特性，并结合 DSP48E2 的大位宽乘法器，同时进行两个 8 位数据乘以一个 9 位权重，并保留完整位宽输出。具体设计分析可以查看 **附录 A** 中 Biubic RTL设计详细解释中的乘加单元(MA Unit)介绍。

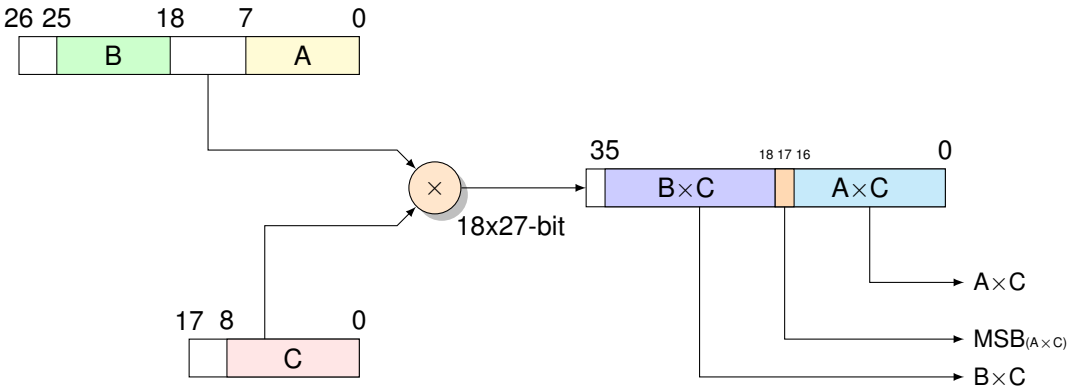


图 4.5: DSP乘法器单元结构图

累和单元

并行累和单元包含了 12 个 DSP48E2 用于实现 25 个数的累和操作。在六个时钟周期后可输出结果。

$$result = \sum_{i=0}^{24} A_i \tag{4.1}$$

图 4.6 为一个 DSP48E2 内部架构图。

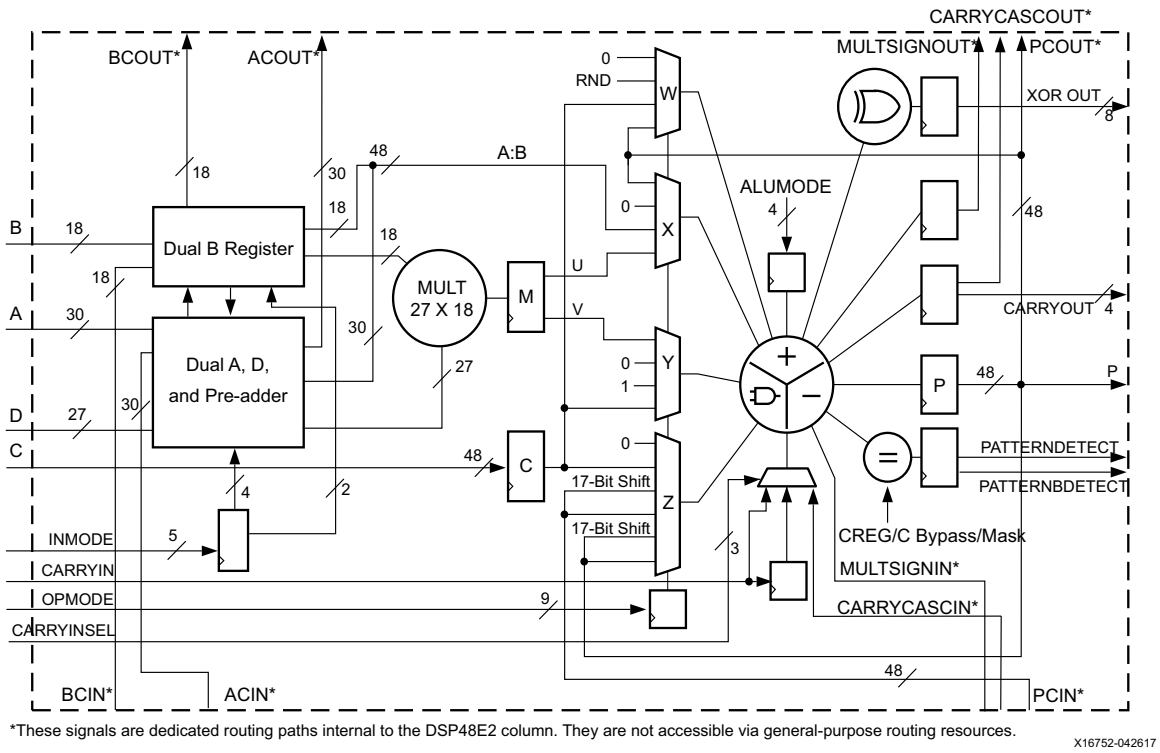


图 4.6: DSP内部结构图

我们可以看到，在 DSP48E2 内部，有两个可以实现加法的环节，第一个 A 与 B 进行乘法之前，有一个相对较小位宽的加法器，另外一个是在 A 与 B 乘法之后一个较大位宽的加法器。我们可以利用这两个加法器实现在一个 DSP48E2 内完成三个数累和，两个时钟周期后输出。

于是，我们可以基于这一个 3 输入 1 输出加法器搭建一个具有三级的 25 输入 1 输出的加法器单元，结果将会在六个时钟周期后输出。

其中需要注意的两个点，第一，我们充分利用了 DSP48E2 内的寄存器资源，以便提升该 IP 最大能达到的时钟频率；第二，我们尽可能的利用上了 DSP48E2 的位宽，保证数据运算时不会因为引入误差而影响后续运算。

舍入单元

并行舍入单元是通过判断后截断位数的低一位是否为 1 进行简单的四舍五入运算，可以直接通过一个 DSP48E2 完成该操作。由于输入数据与权重数据均是经过量化的，定点位数进行四舍五入是简单的操作。舍入运算均会在两个时钟周期后输出结果。

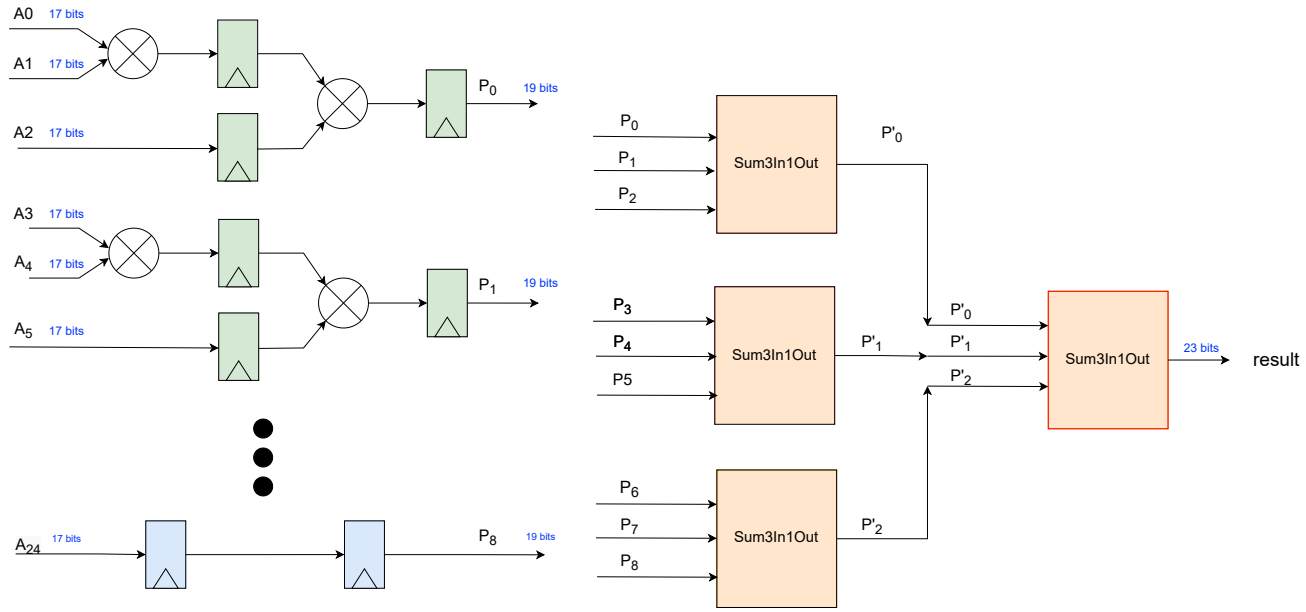


图 4.7: 累和模块

4.3.3　拉普拉斯卷积核

拉普拉斯卷积核包含了一组并行的累和单元、一组并行比较单元。

在拉普拉斯卷积核中，我们使用 4 个 DSP48E2 进行 9 数累和运算。每个像素进行拉普拉斯滤波是以像素本身为中心，边缘 3×3 的像素块作为数据输入。每个像素亏需要与对应的拉普拉斯系数进行相乘然后累和。该拉普拉斯卷积核 IP 每个时钟周期处理 4 个像素。

1	1	1
1	-8	1
1	1	1

图 4.8: 拉普拉斯卷积核数据

由图 4.8 可以看到，拉普拉斯算子中心为 -8，边缘为 1 的权重分布。所以我们直接利用累和完成 9 数累和。其中中心像素我们直接对像素值的低位补 3 个 0 操作，同时将它所传入的 DSP48E2 的加法器设置为减法操作即可。该拉普拉斯卷积核可在四个时钟周期后输出结果。

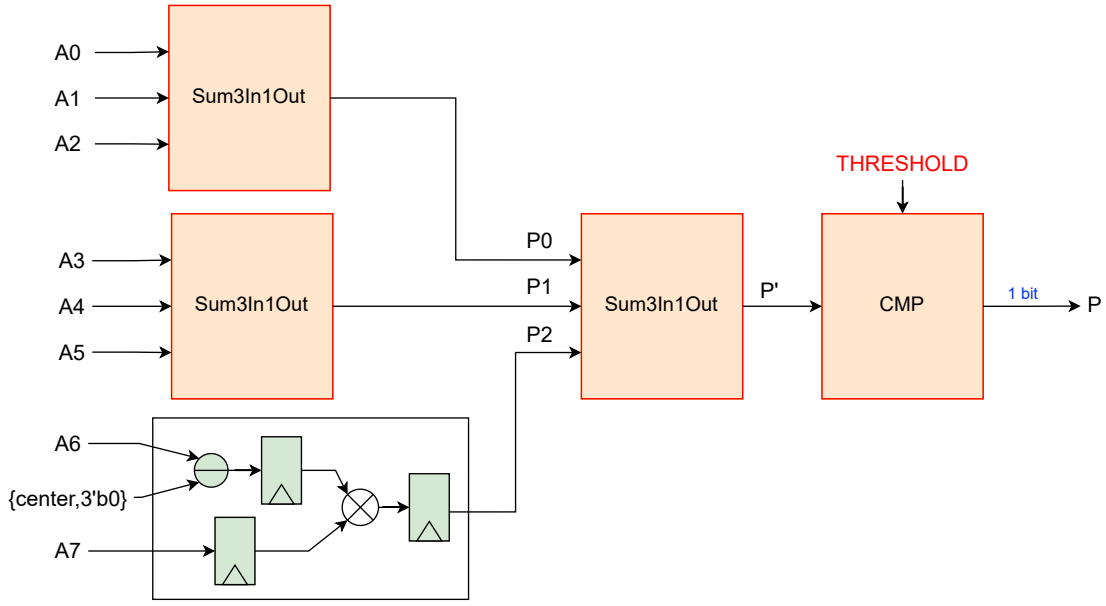


图 4.9: 拉普拉斯卷积

需要注意的是，在高斯卷积与拉普拉斯卷积中均使用了 **3输入1输出** 累和模块，但此时拉普拉斯使用时需要将输入数据转换为有符号数再进行运算。当数据位宽拓展时，需要注意高符号位选择。

4.3.4　纹理分类器

纹理分类器主要包含了四个部分：一个区域划分模块、一个统一编码模块、一个角度编码模块以及一个地址编码模块。 纹理分类单元每个时钟周期处理一个 5×5 经纹理检测后的二值图像块。经过一个时钟周期后输出纹理分类对应滤波器地址。

区域划分模块

区域划分模块主要将后续进行统一编码与角度编码的像素区域进行提取。

其中，对于边缘信息输出的 16 位数据按图像块顺时针提取，输出数据的第 0 位为像素块对应的(0,0)，第 1 位为像素块的(0,1)以此类推。

统一编码模块

统一编码模块用于对边缘区域的像素进行统一编码。我们可以将 16 位的边缘像素

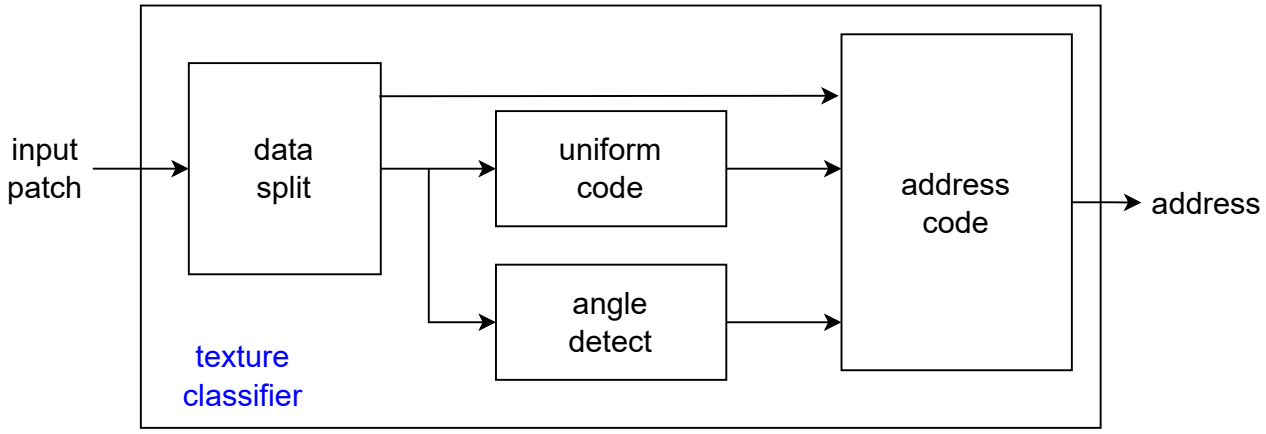


图 4.10: 纹理分类器

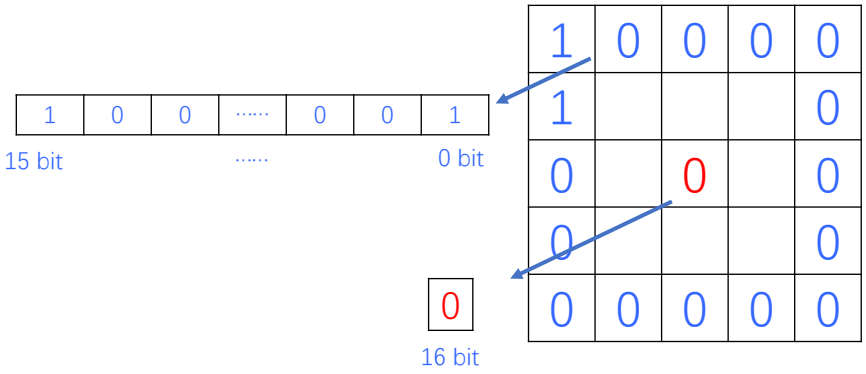


图 4.11: 区域划分模块

数据看作一个环形队列。然后对其按照顺时针的顺序进行边沿检测。可用公式表示为式 4.2:

$$F = P_0 \oplus P_{15} + \sum_{i=0}^{14} P_i \oplus P_{i+1} \tag{4.2}$$

这里我们举一个简单的例子。黑色 0-1 数字代表了像素的数据，蓝色数字代表了该像素数据原本所在边缘数据的位置。每一位数据与它对应的下一位数据进行异或操作。异或结果使用了红色数字进行表示。其中 1 代表检测到边沿跳变，0 代表没有边沿跳变。

边缘像素数据异或运算后的结果通过计数 1 的个数，则为我们最终需要的跳变次

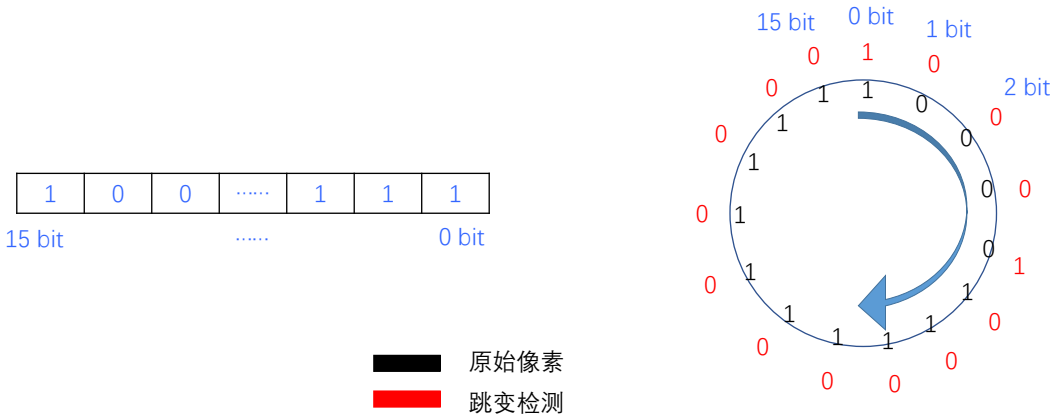


图 4.12: 边缘跳变检测

数 F，编码为 0-16 中的偶数。

在进行边沿检测的时候，同时会统计边缘像素数据中1的个数 N。

当跳变次数 F 为 0 时，统一编码结果输出为 0；当跳变次数 F 为 2 时，统一编码结果输出为 N，其中 N 为整数 1-15；当跳变次数 $F \geq 4$ 时，统一编码结果输出为 16。

edge 1s	u	p
0/16	0	0
1∼15	2	1∼15
/	>=4	16

表 4.1: 跳变次数与编码

对于16个1比特统计个数，我们将其拆分为 4 个 **4输入1输出** 的查找表与 1 个 **4输入1输出** 的小位宽加法器。由于这里进行的数据运算皆是小位宽，我们设计的加法器直接使用查找表进行实现。

角度编码模块

角度编码用于对边缘区域的像素进行角度编码。我们可以将 16 位的边缘像素数据看作一个环形队列。然后对其按顺时针的顺序进行上升沿检测。

这里与统一编码模块的检测边缘思路一致，但在该模块中我们检测的是边缘数据下降沿跳变的位置。然后对于这 16 位的独热码进行提取 1 的位置，其结果为我们所需

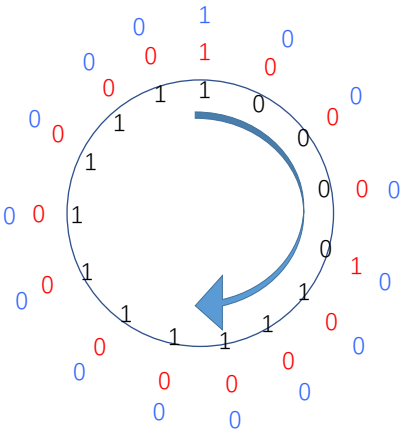


图 4.13: 角度编码

要的角度编码，结果为 0-15。

此处需要注意的是，由于我们角度编码产生的结果有效是建立在统一编码结果为 2 的基础，如果统一编码为其他结果，角度编码输出则为 0。同时，当数据经过统一编码后输出为 2 时，该数据进行角度编码的结果输出必然是独热码。

地址编码模块

地址编码模块通过接收来自统一编码模块、角度编码模块与二值像素块中心像素的信息，进行地址编码，用于查找该像素块纹理特征类别的滤波器地址。

由于统一编码的结果可以分为三种情况，第一种情况为特殊的全 0 或全 1，第二种情况为统一编码为 16，第三种情况为统一编码结果为 1-15。在第一种和第二种情况中，我们不考虑角度编码信息，默认置 0。结合中心像素信息，会产生四个对应的地址。在第三种情况中，我们直接将中心像素信息、统一编码信息与角度信息拼接一起，输出为 9 位的地址。

$$\textbf{case1:}\quad t = \begin{cases} 2'b10 & ,center = 1 \\ 2'b00 & ,center = 0 \end{cases}$$

$$\textbf{case2:}\quad t = \begin{cases} 2'b11 & ,center = 1 \\ 2'b01 & ,center = 0 \end{cases}$$

$$where, addr = \{7'b0, t\}$$

$$\textbf{case3:}\quad addr = \{center, u, angle\}$$

注意，在统一编码的三种情况下，第一、二种情况占用了第三种情况的部分地址，但由于第三种情况不会出现编码为 0 的结果，所以在访问地址上并不冲突。

4.3.5　行缓冲模块

在纹理分类模块中，一共使用了三个行缓冲单元，其中包含两个五行缓冲单元以及一个三行缓冲单元。

每个行缓冲模块用于对输入数据进行缓冲，从而实现为后续计算同时输出三行数据；同时在缓冲时可以由控制信号进行像素填充操作。

缓冲模块的接口可以配置为 AXI4-Stream 接口，方便模块调用。注意，在纹理分类模块内部中，为了减少信号线扇出，仅保留数据总线与数据有效信号。

三行缓冲单元

三行缓冲单元由三个 FIFO 级联形成，读写信号由控制单元进行控制；输出数据连接至输出映射控制器，由控制单元控制，用于实现填充像素功能。

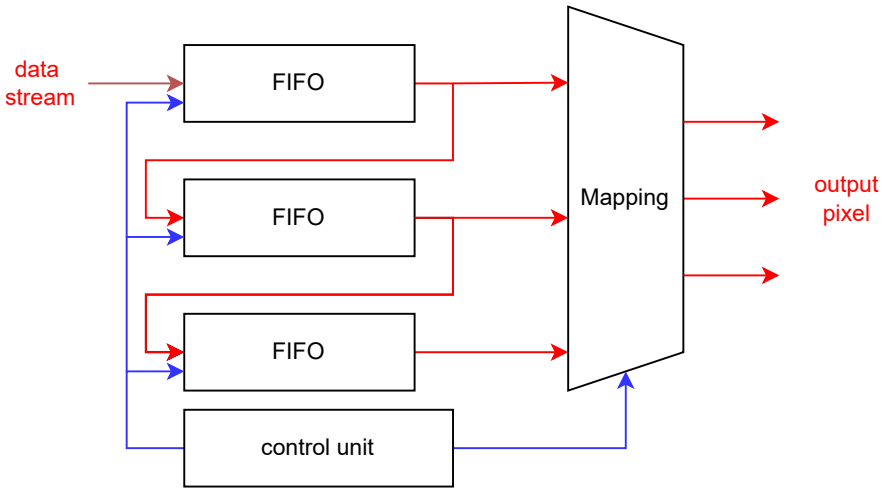


图 4.14: 三行数据缓冲

缓冲与运行　当图像的第一行数据流入 FIFO 时，第一行数据保留在第一行 FIFO 进行缓冲不输出；当第二行数据开始流入 FIFO 时，会流入第一行 FIFO，原本处于第一行 FIFO 的数据开始流向第二行 FIFO，此时仍处于缓冲模式；当第三行数据到来，会流入第一行 FIFO，原本在第一行 FIFO 的数据会流入第二行 FIFO，原本在第二行 FIFO 的数据会流入第三行 FIFO。同时，当第三行数据到来，由于算法上需要进行像素填充操作，所以可以开始进行缓冲数据流出用于后续运算；当最后一行流入的数据完全进入第三行 FIFO 时，该帧图像缓冲结束。

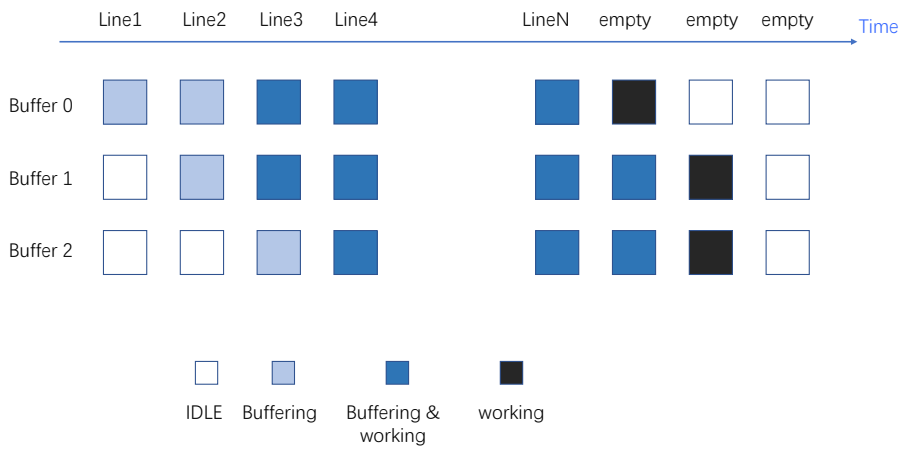


图 4.15: 三行缓冲运行

需要注意的是，刚开始运算像素行的数据仅由第一行 FIFO 与第二行 FIFO 提供，具体表现为第一、二行 FIFO 输出数据有效，第三行 FIFO 输出数据为无效；当最后运算像素行的数据仅由第二行 FIFO 和第三行 FIFO 提供，具体表现为第二、三行 FIFO 输出数据有效，第一行 FIFO 输出数据无效。

输出映射控制　在输出控制映射模块中，主要完成像素行输入填充的映射以及列填充。具体映射规则由控制单元控制。

当只需要进行上边缘填充的时候，只需要将进入的两行像素中的上边缘像素行映射到另外一个输出端口中即可。下边缘填充同理。

列填充的映射方式与行映射一致。需要建立一级寄存器存储该行像素的上一个时

钟周期的末尾像素，由控制单元控制输出像素选择。

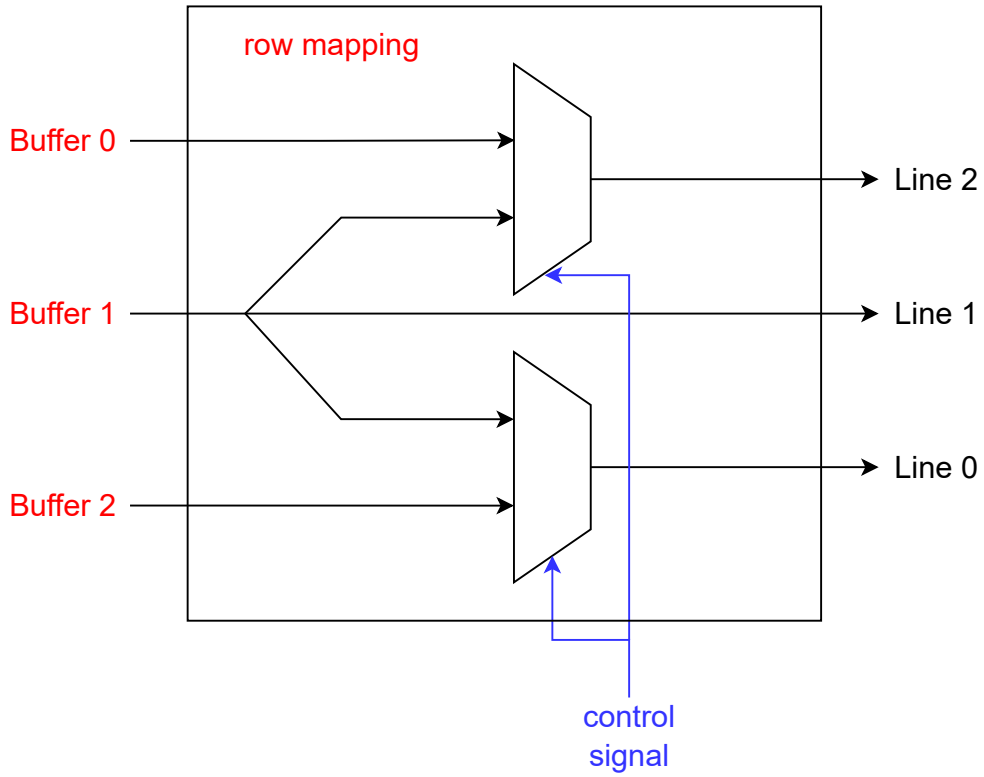


图 4.16: 行映射

控制单元　主要实现了对 FIFO 进行读写控制，以及输出映射的模式选择。其核心为一个计数器单元，计数输入流有效数据个数。同时检测当前行输入是否为图像帧结尾，如果是，则控制 FIFO 内数据自动排出。可实现单帧图像自动输出。

需要注意的是，上述设计介绍均基于每时钟周期处理一个像素。实际上设计的模块运行在每时钟周期处理四个像素。 FIFO 每次缓冲从 1 个像素 N 比特数据变为个像素 $4N$ 比特输入，输出为四路像素阵列至对应的四个运算单元。在 FIFO 与输出映射单元数据通路间会插入一级寄存器，用于暂存后续需要使用到当前周期的像素。具体表现为将该寄存器数据经过多路选择模块进入输出映射单元。

五行缓冲单元

五行缓冲单元由五个 FIFO 级联形成，读写信号由控制单元进行控制；输出数据连接至输出映射控制器，由控制单元控制，用于实现填充像素功能。

具体设计细节可参考三行缓冲单元设计，五行缓冲单元为其拓展设计，设计思路与其一致。

4.4　自适应锐化模块

自适应锐化模块可进行对于不同 5×5 的图像块根据其纹理特征进行实时锐化。该 IP 包含了一个 5×5 卷积单元、一个滤波器参数存储单元以及一个行缓冲模块。为了确保卷积操作后仍使图像保持原始尺寸，行缓冲模块包含了图像填充处理操作与数据映射模块，由控制单元进行管理。滤波器存储单元用于存储不同纹理类型的锐化参数，为图像锐化卷积提供权重数据。

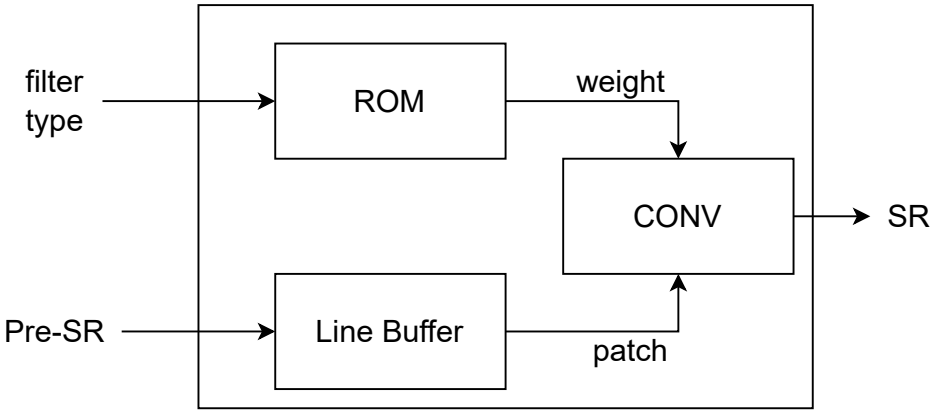


图 4.17: 自适应锐化模块

该自适应锐化模块每个时钟周期可处理 4 个像素。

4.4.1　运算位宽与量化

为了保证运算结果精度以及存储空间尽可能降低，最大化提升 DSP 和片上 RAM 利用率，在本模块中，对于 DSP48E2 和片上 RAM 的使用有进行特殊优化。其中，我

们将滤波器参数进行 12 位有符号量化，在保证滤波器性能的同时充分降低了存储空间占用。另外，在 DSP 运算过程中，我们保留所有运算位宽，到结果输出时方进行截断操作。

4.4.2　5×5 卷积单元

5×5 卷积单元包含了一组并行乘法器单元、一组并行累和单元、一组并行舍入单元以及一组输出限幅单元。 每个像素进行滤波卷积是以像素本身中心，大小为 5×5

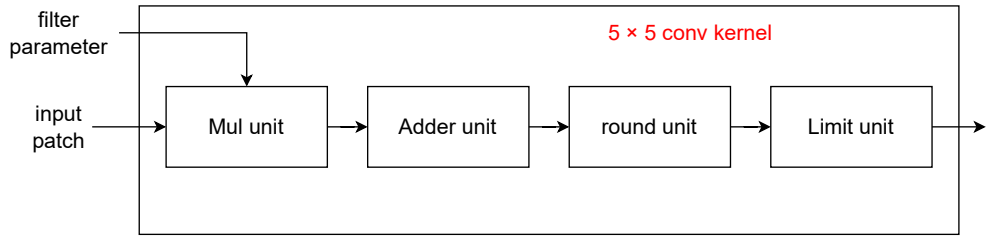


图 4.18: 卷积单元

的像素块作为数据输入，对应的 5×5 滤波器参数作为权重输入。每个像素块需要与对应的预学习的滤波器权重进行相乘然后累和。该卷积核 IP 每个时钟单元处理 4 个像素。

乘法器单元

并行乘法单元包含了 25 个 DSP48E2 用于实现 5×5 大小的卷积乘法操作。在两个时钟周期后可输出结果(一个时钟周期后也可以实现结果输出，但为了提升 IP 最大时钟频率，输出端插入一级寄存器增大时序冗余)

需要注意的是，此处的自适应滤波器乘法单元与上面所介绍的 **高斯滤波器乘法单元** 有所区别，由于尽可能保留位宽，不能对 DSP48E2 进行进一步优化。

舍入单元

此处舍入单元与 **高斯滤波器舍入单元** 设计一致。

输出限幅单元

由于自适应锐化单元输出为 SR 图像像素，需要将像素数据进行限制在 8bit 内(0 - 255)。此限幅单元实现的功能与舍入单元进行耦合，通过 DSP48E2 内部功能模块，能同时实现输出限幅与数据四舍五入功能。像素结果可在两个时钟周期后正确输出。

4.4.3　滤波器参数存储单元

滤波器参数存储单元用于存储 484 个 5×5 滤波器权重参数，每个参数被量化为 12 位有符号二进制数进行存储。为了满足卷积操作实时性要求，尽可能一次性将 25 个权重参数获取，我们将每个滤波器的参数拼接为一个 325 比特位宽二进制数以补码形式进行存储。

在硬件实现上，我们可以选择 1 个 RAM18BE2 与 8 个 RAMB36E2 拼接而成，或者选择 5 个 URAM288 拼接而成。为了更好的利用 Xilinx UltraScale+ 架构，我们选择了后者进行实现，首先是因为其提供了更大位宽的选择，其次是因为我们可以利用其真双口读写的性质，有效减少多个运算单元访问存储权重参数问题，进而降低一半的权重存储成本。

需要注意的是，由于 URAM288 数据输出位宽为 72 比特，所以需要拼接 5 个 URAM288 才能满足 325 比特数据同时输出。对于第 5 个 URAM288 是浪费了大半的资源。可以通过将第五个 URAM288 替换成 RAMB18E2 进行优化，但为了方便设计以及存储架构的兼容性，暂时保留目前设计方案。

4.4.4　关于设计补充

由于该模块设计为每个时钟周期完成 4 像素运算操作，故需要使用 4 个 5×5 **卷积单元** 与 2 个 **滤波器参数存储单元**。每 2 个 **卷积运算单元** 共用 1 个滤波器参数存储单元。

4.5　色域转换模块

色域转换模块主要分为两部分： RGBtoYUV Unit 和 YUVtoRGB Unit。

4.5.1 RGBtoYUV Unit

在 RGB 色域至 YUV 色域转换单元中，遵循 ITU-R 标准版本，即公式 4.3。

$$\begin{aligned} Y &= 0.299R &&+0.587G + 0.114B \\ U &= -0.169R &&-0.331G + 0.500B &&+128 \\ V &= 0.500R &&-0.418G - 0.081B &&+128 \end{aligned} \tag{4.3}$$

为了保证色域转换时保持像素尽可能正确，减少引入因色域转换带来的误差，采用了对系数进行 10bit 含符号位的量化。由于后续单元仅对 Y 通道的像素进行操作，故我们尽可能的保留 U、V 通道的位宽，且不进行区间偏移运算。整理后的公式如 4.4 所示。

$$\begin{aligned} Y &= 0.298828125R &&+0.5869140625G + 0.1142578125B \\ U &= -0.1689453125R &&-0.3310546875G + 0.5000000000B \\ V &= 0.5000000000R &&-0.4189453125G - 0.0810546875B \end{aligned} \tag{4.4}$$

此单元所需要的运算单元为自适应锐化模块中的 **乘法器单元** 与 **累和单元** 复用。其中，对于 Y 通道的运算采用了三个 二输入乘法器 进行系数运算，一个 三输入累加器 进行求和操作；对于 U、V 通道的运算采用了两个 二输入乘法器 进行系数运算，一个 三输入累加器 进行求和操作。

运算结束后，Y 通道像素保留 8bit 位宽，方便后续纹理分类器中 **高斯滤波器的乘法运算单元** 优化操作； U、V 通道则保留 16bit 位宽 (需要截去低四位)，尽可能减少误差的引入。

在该单元进入时，需要对三通道像素做同步操作，使用深度为 16 的 FIFO 完成操作。避免因通道间像素不同步引起的运算错误，造成图像质量衰退，提升该 IP 的兼容性。该单元结束后不进行通道同步，由该 IP 内后续的缓冲单元接收。

需要注意的是，0.5 系数乘法运算在 **累加器** 中以舍去最低位完成。

4.5.2 YUVtoRGB Unit

在 YUV 色域至 RGB 色域转换单元中，遵循公式 4.5。

$$\begin{aligned} R &= Y & &+ 1.403(V-128) \\ G &= Y & -0.343(U-128) &- 0.714(V-128) \\ B &= Y & +1.770(U-128) & \end{aligned} \tag{4.5}$$

为了保证色域转换时保持像素尽可能正确，减少引入因色域转换带来的误差，采用了对系数进行 10bit 含符号位的量化。同时在经过 RGB 至 YUV 色域转换公式调整，整理后的公式如 4.6 所示。

$$\begin{aligned} R &= Y << 20 & &+ 1.4033203125(V << 4) \\ G &= Y << 20 & -0.3427734375(U << 4) &- 0.7138671875(V << 4) \\ B &= Y << 20 & +1.7695312500(U << 4) & \end{aligned} \tag{4.6}$$

此单元所需要的运算单元为自适应锐化模块中的 **乘法器单元** 与 **累和单元** 复用。其中，对于 B 通道的运算采用了两个 带进位的三输入乘法累和器 进行系数运算和累和操作；对于 R、B 通道的运算采用了一个 三输入的乘法累和器 继续系数运算和累和操作。

需要注意的是，公式 4.6 需要进行的移位操作，在进入 乘法运算单元 前以末位补零实现。

运算结束后，R、G、B 通道需要保留 8bit 位宽并需要进行输出限幅操作，此处复用了自适应模块中的 **输出限幅单元**。限幅输入单元的数据为运算结果实际小数位数后一位。

在该单元进入时，需要对三通道像素做同步操作，使用深度为 16 的 FIFO 完成操作。避免因通道间像素不同步引起的运算错误，造成图像质量衰退，提升该 IP 的兼容性。由于该色域转换单元为整个 IP 的输出末尾，故此需要对三通道像素进行同步后输出，同样采用了深度为 16 的 FIFO 完成操作。

4.5.3　注意

以上两个模块均符合 AXI4-Stream 协议接口。如有其他需要，可直接移植。

4.6　总结

具体硬件RTL设计已经详细介绍完毕，各模块参数性能请查看性能说明文档。

第五章　仿真验证环境及说明

5.1　验证工具

本次 IP 验证平台将基于 System Verilog 进行编写。

使用由队伍成员邓立唯开源的 *Bitmap Processing Library & AXI-Stream Video Image VIP* 进行测试样例图片读取写回，简化测试样例生成步骤及测试结果输出对比。

To verificate a video or a image processing IP, you may need to read a real image into your design, send its data by an interface. Then, get the output from the interface, and convert it to a new image, save or compare it. ——*Bitmap Processing Library & AXI-Stream Video Image VIP*

由于队伍成员每个人有不同的验证工具使用习惯，在本次项目验证中将会使用到：

- Intel ModelSim
- Synopsys VCS & Verdi
- Verilator & GtkWave

5.2　验证方法及策略

采用 **直接验证** 与 **随机验证** 结合。

直接验证: 通过对比测试图片在 C Model 进行超分辨率算法运算结果与 RTL 代码在

仿真中输出结果对比。

随机验证: 对部分子模块(如高斯滤波器、纹理分类器等)所需的运算数据通过产生随机种子产生随机数据，对比参考模型运算输出结果与待测模块输出结果。

5.3　验证范围

5.3.1　各子运算单元

- 运算结果正确
- 运算结果有效输出
- 结果输出是否超出数据范围
- 满足时序要求

5.3.2　Bicubic 上采样模块

- 满足 AXI4-Stream 协议要求，完成视频流数据收发
- 完成 Biubic 上采样算法结果输出
- 运算结果与参考模型匹配
- 结果输出是否超出数据范围
- 支持视频帧停顿
- 满足时序要求

5.3.3　纹理分类模块

- 满足 AXI4-Stream 协议要求，完成视频流数据接收
- 完成纹理分类算法结果输出
- 输出寻址结果与参考模型匹配

- 结果输出是否超出地址范围
- 支持视频帧停顿
- 满足时序要求

5.3.4　自适应锐化模块

- 满足 AXI4-Stream 协议要求，完成视频流数据收发
- 完成自适应锐化算法结果输出
- 输出像素结果与参考模型匹配
- 结果输出是否超出数据范围
- 支持视频帧停顿
- 满足时序要求

5.3.5　色域转换模块

- 满足 AXI4-Stream 协议要求，完成视频流数据收发
- 完成色域转换运算结果输出
- 输出像素结果与参考公式一致
- 结果输出是否超出数据范围
- 满足时序要求

5.3.6　注意事项

- 不需进行跨时钟域检查
- 需进行覆盖率验证

5.4　验证环境

5.4.1　验证平台

该验证平台包含了测试样例生成器、驱动器、待测单元、其他外设VIP(如DDR)、监视器、记分板、参考模型以及一个全局配置。

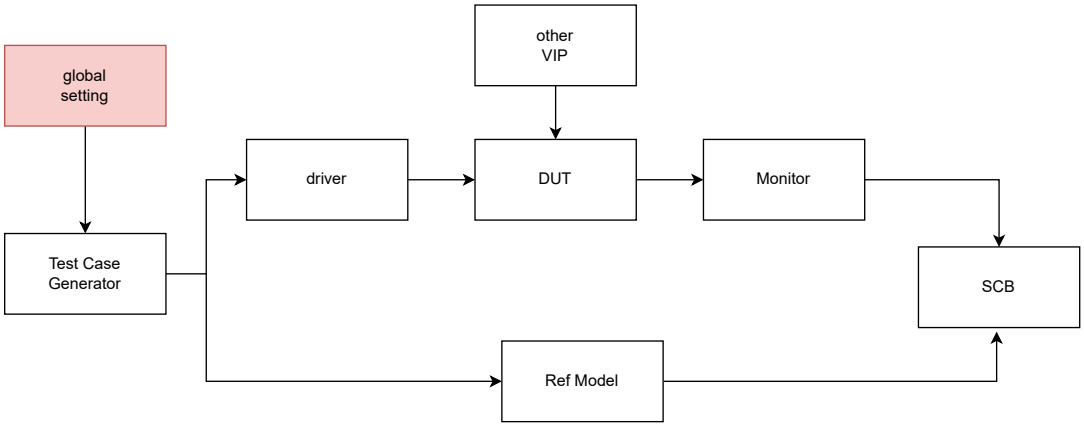


图 5.1: 验证平台

5.4.2　验证计划

- 使用 60 组真实图片进行验证；
- 使用 100 组随机生成图片进行验证；
- 使用极端数值进行验证；
- 使用随机数值进行验证。

5.4.3　待验证设计

无待验证设计

5.4.4　完成验证设计

- 各子运算单元
- Bicubic 上采样模块
- 纹理分类模块
- 自适应锐化模块
- 色域转换模块

5.5　覆盖率

5.5.1　各子运算单元

该单元无收集验证覆盖率。

5.5.2　Bicubic 上采样模块

详细验证覆盖率结果可参考 **附录 A**。

5.5.3　纹理分类模块

详细验证覆盖率结果可查看 **附录 B**。

5.5.4　自适应锐化模块

详细验证覆盖率结果可查看 **附录 C**。

5.5.5　色域转换模块

RGB2YUV Unit

详细验证覆盖率结果可查看 **附录 D**。

YUV2RGB Unit

详细验证覆盖率结果可查看 **附录 E**。

5.6　验证分析

5.6.1　各子运算单元

符合设计预期。

5.6.2　Bicubic 上采样模块

符合设计预期。

5.6.3　纹理分类模块

验证结果设计需要更完备，测试条件未覆盖完全。

5.6.4　自适应锐化模块

验证结果设计需要更完备，测试条件未覆盖完全。

5.6.5　色域转换模块

符合设计预期。

第六章　性能评估说明

6.1　概述

性能评估主要分为三大部分，针对于每个设计 IP 进行单独评估分析，其中包括 Bicubic 上采样模块性能评估、纹理分类模块性能评估以及自适应锐化模块性能评估。

6.2　Bicubic 上采样模块性能评估

本章性能评估数据参考 **附录 A**。

6.2.1　性能参数

使用 Vivado 与 Synopsys Synplify Premier 进行时序分析。以输入 960×540 的图像数据为基准。

最大频率

最大延迟

仅用于评估该IP的路径时延，不考虑受系统延迟或其他限制。假设图像输入宽为 W 位像素。

吞吐量

用于评估不同图像帧大小进入。

FPGA Device Family	Analysis Tool	Fmax (MHz)
Xilinx Virtex UltraScale+	Synopsys Synplify Premier 2020.03	628.6
Xilinx Kintex UltraScale+	Synopsys Synplify Premier 2020.03	417.2
2*Xilinx Zynq UltraScale+	Synopsys Synplify Premier 2020.03	394.1
	Vivado 2021.1	428.4
Xilinx Kintex 7	Synopsys Synplify Premier 2020.03	361.7
Xinlinx Artix 7	Synopsys Synplify Premier 2020.03	192.2
Intel Stratix 10	Synopsys Synplify Premier 2020.03	260.7
Intel Max 10	Synopsys Synplify Premier 2020.03	220.5
Intel Arria V	Synopsys Synplify Premier 2020.03	142.5

表 6.1: 最大频率

描述	时钟周期
Bicubic流水运算从输入到输出	13
第一个像素进入到第一个像素输出	$10 \cdot W + 28$
最后一个像素进入到最后一个像素输出	$13 \cdot W + 31$

表 6.2: 最大延迟

6.2.2　资源使用量

Xilinx Zynq UltraScale+ 器件的资源使用量的结果是在 Vivado 综合器下，使用 DSP48E2 和 XPM 宏进行评估的。

其他器件的评估结果是使用 Verilog 自动推断完成的，可能由于每个器件的 DSP 模块的数据宽度不同而导致不同。实时视频 Bicubic 上采样 IP 已为 Xilinx UltraScale+ 系列器件的 DSP48E2 模块特别优化。为了最大限度地利用资源，建议使用这些器件进行合成。

输入分辨率	输出分辨率	吞吐量(FPS/MHz)	FPS@150MHz
320 x 240	1280 x 960	3.23	484.7
480 x 270	1920 x 1080	1.92	287.6
640 x 360	2560 x 1440	1.08	162.1
960 x 540	3840 x 2160	0.48	72.1

表 6.3: 吞吐量

2*Device	configuration		Resource Utilization			
	Input Resolution	fCLK(MHz)	LUTS	FFTs	DSPs	BRAMs
XCZU15EG	960 × 540	300	431	694	49ˆ1	2.5ˆ3
XC7K325T	960 × 540	150	1979	2713	30ˆ2	2ˆ3

表 6.4: 资源使用量

6.3　纹理分类模块性能评估

6.3.1　性能参数

使用 Vivado 进行时序分析。

最大频率

本模块与自适应锐化模块搭配使用，不单独进行最大频率估计。

最大延迟

仅用于评估该IP的路径时延，不考虑受系统延迟或其他限制。假设图像输入宽为 W 位像素。

吞吐量

与 Bicubic 上采样模块吞吐量评估结果一致。

描述	时钟周期
高斯滤波运算从输入到输出	8
拉普拉斯滤波运算从输入到输出	4
纹理检测器运算从输入到输出	2
第一个像素进入到第一个像素输出	$8 \cdot W + 17$
最后一个像素进入到最后一个像素输出	$8 \cdot W + 17$

表 6.5: 最大延迟

6.3.2　资源使用量

本模块与自适应锐化模块搭配使用，不单独进行资源使用量估计。

6.4　自适应锐化模块性能评估

6.4.1　性能参数

使用 Vivado 进行时序分析。

最大频率

此模块最大频率限制是由于内部设计使用了 URAM288，其最大速率由工艺决定。

FPGA Device Family	Analysis Tool	Fmax (MHz)
Xilinx Virtex UltraScale+	Vivado 2021.2	355.1
Xilinx Kintex UltraScale+	Vivado 2021.2	351.3
Xilinx Zynq UltraScale+	Vivado 2021.2	344.4
Xilinx Versal AI Core Series	Vivado 2021.2	348.8

表 6.6: 最大频率

最大延迟

仅用于评估该IP的路径时延，不考虑受系统延迟或其他限制。假设图像输入宽为 W 位像素。

描述	时钟周期
锐化卷积运算从输入到输出	8
第一个像素进入到第一个像素输出	$3 \cdot W + 12$
最后一个像素进入到最后一个像素输出	$3 \cdot W + 12$

表 6.7: 最大延迟

吞吐量

与 Bicubic 上采样模块吞吐量评估结果一致。

6.4.2　资源使用量

Xilinx Zynq UltraScale+ 器件的资源使用量的结果是在 Vivado 综合器下，使用 DSP48E2 和 XPM 宏进行评估的。其他器件的评估结果是使用 Verilog 自动推断完成的，可能由于每个器件的 DSP 模块的数据宽度不同而导致不同。实时视频自适应锐化 IP 已为 Xilinx UltraScale+ 系列器件的 DSP48E2 模块特别优化。为了最大限度地利用资源，建议使用这些器件进行合成。

er 2*Device	Configuration Parameter		resource utilization				
	Input resolution	fclk(MHz)	LUTs	FFs	DSPs	BRAM	URAM
XCZU15EG	960 × 540	300	4608	479	348	37.5	21

表 6.8: 资源使用量

第七章　FPGA 验证报告

7.1　概述

在本项目中，需要进行 FPGA 板上验证的模块包括：Bicubic 上采样模块、纹理分类模块、自适应锐化模块。

目前所有模块已完成 FPGA 板上验证。

7.2　验证方案

7.2.1　验证平台介绍

本项目 FPGA 片上验证基于米联客 MZU15A-15EG (Xilinx Zynq UltraScale+ MPSoC)完成。

7.2.2　验证框架介绍

验证框架参考 **附录 A** 概述或图 7.1 所示。

7.2.3　验证流程介绍

验证流程参考 **附录 A** 设计工作流程。

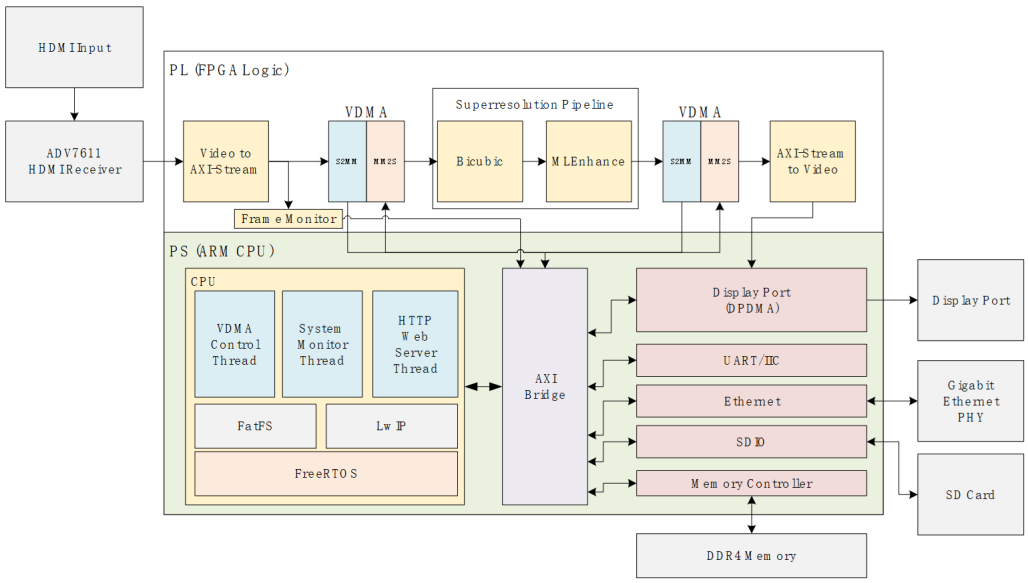


图 7.1: FPGA 验证系统框架

7.2.4　上位机介绍

基于 Bootstrap 前端框架，使用 JavaScript 和 CSS 开发了网页端上位机，本地主机或服务器通过千兆以太网与 FPGA 开发板 PS 侧连接，可实现实时图像、视频数据、控制指令交互功能。页面如 7.2 、 7.3 所展示。

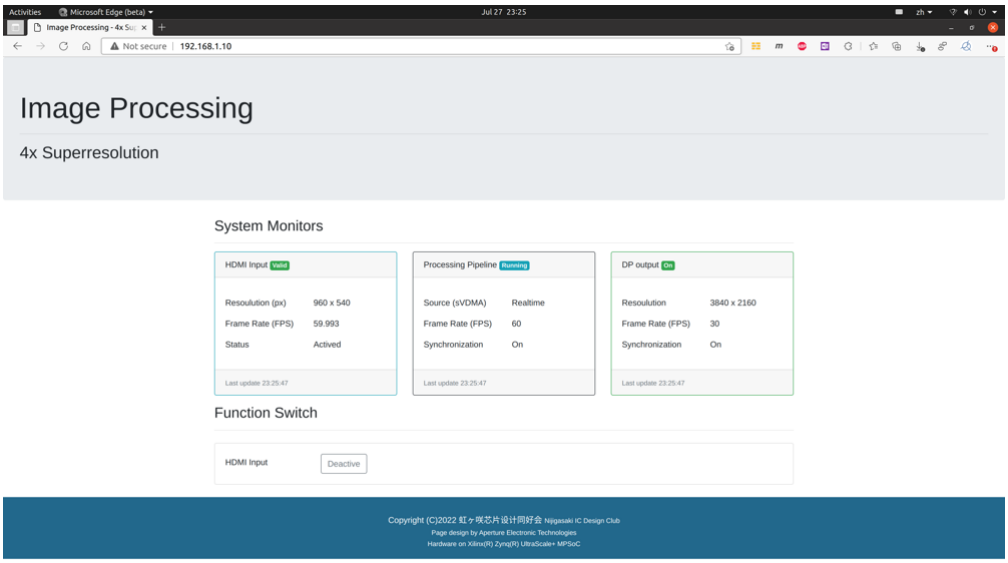


图 7.2: 上位机界面展示1

输入模式选择

- HDMI 实时输入
- 网页上传图片

其他功能

- 实时状态监视
- 网页控制(一键)处理流程
- 自动将处理好的图片并打包成 ZIP 并保存

7.3　Demo 展示

Demo0(自动从外部播放器打开)

Demo1(自动从外部播放器打开)

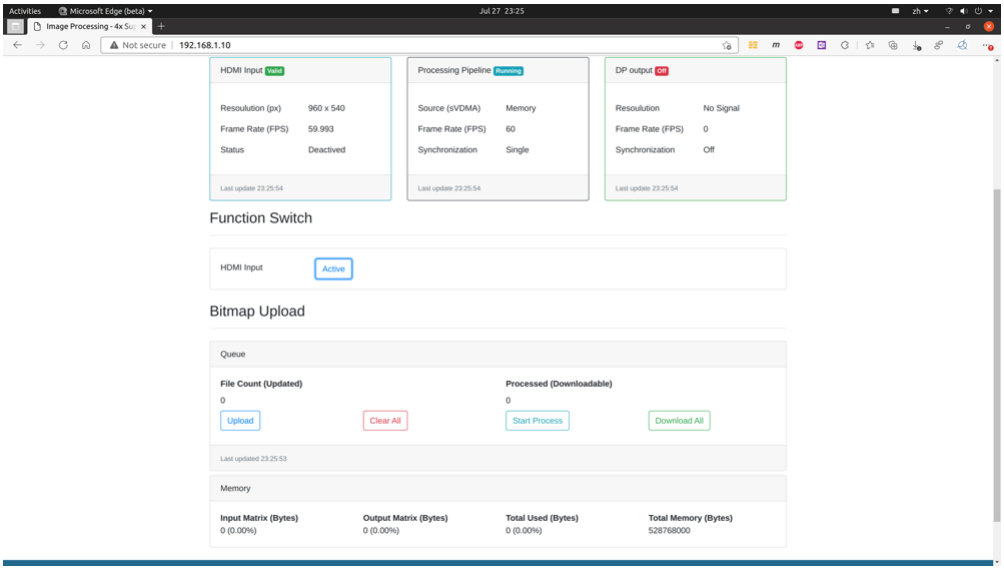


图 7.3: 上位机界面展示2

参考文献

参考文献

[1] Romano, Yaniv, John Isidoro, and Peyman Milanfar. "RAISR: rapid and accurate image super resolution." IEEE Transactions on Computational Imaging 3.1 (2016): 110-125.

[2] Jeong, S. C.; Song, B. C. Training-based superresolution algorithm using k-means clustering and detail enhancement. In: Proceedings of the 18th European Signal Processing Conference, 1791–1795, 2010.

[3] Yu, G. S.; Sapiro, G.; Mallat, S. Solving inverse problems with piecewise linear estimators: From Gaussian mixture models to structured sparsity. IEEE Transactions on Image Processing Vol. 21, No. 5, 2481–2499, 2012

[4] R.G. Keys, "Cubic convolution interpolation for digital image processing," IEEE Transaction on Acoustics, Speech and Signal processing, vol. 29, no. 6, Dec. 1981, pp. 1153-1160.

[5] H.S. Hou and H.C. Andrews, "Cubic splines for image interpolation and digital filtering," IEEE Transaction Signal processing, vol. 26, no. 6, Dec. 1978, pp. 1153-1160.

[6] X. Feng and P. Milanfar, "Multiscale principal components analysis for image local orientation estimation," Proceedings of the 36th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, CA, November 2002. 5

[7] Zabih, R.; Woodfill, J. Non-parametric local transforms for computing visual correspondence. In: Computer Vision — ECCV '94. Lecture Notes in Computer Science, Vol. 801. Eklundh, J. O. Ed. Springer Berlin Heidelberg, 151–158, 1994.

[8] T. Ojala, M. Pietikainen and T. Maenpaa, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 24, no. 7, pp. 971-987, July 2002, doi: 10.1109/TPAMI.2002.1017623.

[9] Z. Guo, L. Zhang and D. Zhang, "A Completed Modeling of Local Binary Pattern Operator for Texture Classification," in IEEE Transactions on Image Processing, vol. 19, no. 6, pp. 1657-1663, June 2010, doi: 10.1109/TIP.2010.2044957.

[10] R. Keys, "Cubic convolution interpolation for digital image processing," IEEE Trans. on Acoustics, Speech and Signal Proc., vol. 29, no. 6, pp. 1153–1160, 1981. 1

[11] H. Hou and H. Andrews, "Cubic splines for image interpolation and digital filtering," IEEE Trans. on Acoustics, Speech and Signal Proc., vol. 26, no. 6, pp. 508–517, 1978. 1

[12] Marr D, Hildreth E. Theroy of edge detection. Proc. of the Royal Society of London. Series B. Biological Sciences, 1980,207(1167): 187-217.

[13] Ojala, T., Pietikäinen, M. and Harwood, D. (1996), A Comparative Study of Texture Measures with Classification Based on Feature Distributions. Pattern Recognition 19(3):51-59.

附录 A

APV21B - Real-time Video 16X Bicubic Super-resolution IP, AXI4-Stream Video Interface Compatible, 4K 60FPS

Deng LiWei
Nijigasaki IC Design Club

July 29, 2022

INTRODUCTION

The APV21B Real-time Video 16X Bicubic Super-resolution core is a soft IP core. It provides fully real-time 16X Bicubic interpolation video super-resolution, and its high performance design allows it to support video output resolutions in excess of 4K 60FPS.

The APV21B is compatibled with the AXI4-Stream Video protocol as described in the **Video IP: AXI Feature Adoption** section of the *Vivado AXI Reference Guide* (Xilinx Inc. UG1037) and **AXI4-Stream Signaling Interface** section of the *AXI4-Stream Video IP and System Design Guide* (Xilinx Inc. UG934).

FEATURES

- AXI4-Stream Video Interface Input/Output
- Supported single 8-bit channel input
- Supported 4 pixel per clock (4 x 8-bit) output to reduce the interface frequency.
- Real-time Bicubic computation core
- Parametric configurable input resolution
- Output supported to 4K 60FPS (@150 MHz)
- Optimized design for DSP48E2 unit of Xilinx UltraScale+ architecture
- Complete synthesizable HDL design
- Macro switches to using different implementations (DSP Macro/Verilog Inferring)

FACTS TABLE

Core Specifics

Supported Device[1]	Xilinx, Intel and Lattice FPGAs Digital ASICs
Supported User Interface	AXI4-Stream
Resources	

Provided with Core

Design Files[2]	System Verilog
Example Design	Provided
Test Bench	Provided
Constraints File	Not Provided
Simulation Model	Not Provided
Microarchitecture Design[3]	Open Source

Tested Design Flows

Design Entry	Vivado Design Suite
Simulation	Mentor Modelsim
Synthesis	Synopsys Synplify Premier and Vivado Synthesis

[1] To maximize the performance and minimize the resource usage, a Xilinx UltraScale+ architecture device is recommended.
[2] Fully synthesizable subset of System Verilog syntax.
[3] Detailed description of the microarchitecture design can be found in this document

Contents

1 Overview **5**
1.1 Feature Summary . . . 5
1.2 Applications . . . 6
1.3 Unsupported Features . . . 6

2 Product Specification **8**
2.1 Performance . . . 8
2.2 Resource Utilization . . . 9
2.3 Port Descriptions . . . 10
2.4 Timing Diagrams . . . 10

3 Algorithm Description **11**
3.1 Terminology . . . 11
3.2 Bicubic Algorithm Process . . . 12
3.3 Bicubic Interpolation Function . . . 12
3.4 Super-pixel Symmetry in Super-block . . . 13
3.5 Pixel Alignment and Edge Padding of the Super-pixels . . . 14

4 Microarchitecture Design **16**
4.1 Operational Bit-width and Quantization . . . 16
4.2 Real-time Bicubic Pipeline . . . 16
4.2.1 Super-block Buffer . . . 17
4.2.2 Symmetric Multiplexer . . . 18
4.2.3 Bicubic Coefficient LUT . . . 18
4.2.4 Multiplier Adder Unit (MA Unit) . . . 18
4.2.5 Add Unit . . . 22
4.2.6 Round Unit and Limit Unit . . . 24
4.3 Line Buffers . . . 24
4.4 Control Unit . . . 27
4.4.1 Column Padding . . . 27
4.4.2 Symmetry Control Signal Generating . . . 28
4.4.3 Super-pixel Realign Signal Generating . . . 28
4.4.4 AXI4-Stream Video Source Control Signal Generating . . . 28
4.5 Pixel Realign Unit . . . 29

5 Design Verification **30**
5.1 Verification Platform . . . 30
5.2 Software Platform . . . 31
5.3 Verification Methodology . . . 31
5.4 Verification Plan . . . 31
5.5 AXI-Stream Video Image VIP . . . 32
5.6 Test Result . . . 32

6 Design with the Core **34**
6.1 Operation . . . 34
6.2 Clock . . . 34
6.3 Clock Enable . . . 34
6.4 Reset . . . 34

7 Design Flow Steps **35**
7.1 File Directory Structure . . . 35
7.2 Source Files . . . 35
7.3 Parameters . . . 37
7.4 Macro Definitions . . . 37
7.5 Constraining the Core . . . 38

8 Example Design **40**
8.1 Overview . . . 40
8.2 IP Cores . . . 42
8.3 Peripherals of the CPU . . . 42
8.4 Buses of the CPU . . . 43
8.5 Workflow of the Design . . . 43

9 Licensing and Open Soruce Information **45**

1 Overview

Currently, the resolution of the display devices is getting higher and higher. Outputting high resolution (HR) graphics has become one of the general ability of various image processing systems. However, process HR graphics directly makes processing or rendering systems face greater challenges in terms of resources, bandwidth and storage.

Resources consumption of systems can be reduced if graphics can be processed or rendered at low resolution (LR). Then, a scale subsystem can scaled up the graphics to HR using super-resolution algorithms. Nvidia's *Deep Learning Super Sampling* (DLSS), AMD's *FideltiyFX Super Resolution* (FSR) and Intel's *Xe Super Sampling* (XeSS) are all well-established methods for obtaining HR graphics under LR rendering using super-resolution techniques.

The real-time Bicubic super-resolution IP core is designed to scale up the LR video streams. It is based on the effective and moderately complex Bicubic algorithm, and supports AXI4-Stream video stream input/output.

Figure 1 illustrates the Real-time Bicubic Super-resolution IP Block Diagram.

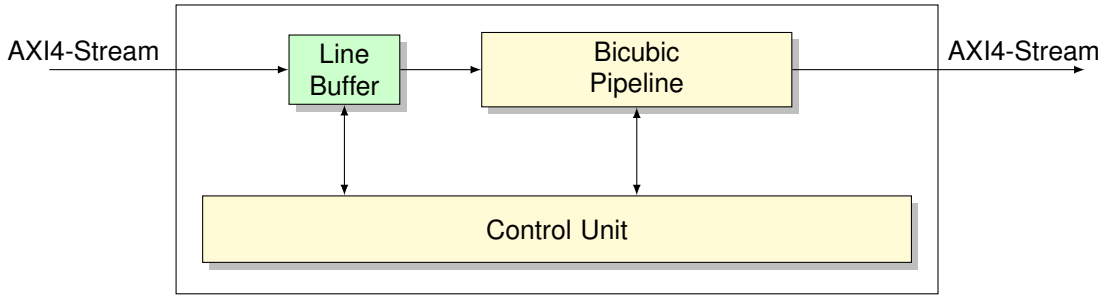


Figure 1: Real-time Bicubic Super-resolution IP Block Diagram

1.1 Feature Summary

AXI4-Stream Video Interface Compliant

The real-time Bicubic super-resolution IP is fully compliant with the AXI4-Stream Video interface. Both input and output inteface has *Start of Frame* and *End of Line* signal.

Detailed protocol of AXI4-Stream Video Interface is provided in **AXI4-Stream Signaling Interface** section of the *AXI4-Stream Video IP and System Design Guide* (Xilinx Inc. UG934).

Please note that the output interface has not TREADY signal which means backpressure blocking on output interface is unsupported. If the design need this function, please consider using an AXI4-Stream FIFO.

AXI4-Stream Data Width of Input Side

The AXI4-Stream interface of input side of the Bicubic IP is 8-bit data width. This 8-bit data can be connected to a channel of 8 bits per channel (BPC) input video stream.

To process multi-channel video, create multiple instances of the IP, and make them work in parallel. It is recommended that video streams in YUV format be processed for best results.

10 BPC or 12 BPC video streams are not supported in this IP.

AXI4-Stream Data Width of Output Side

The AXI4-Stream interface of output side of the Bicubic IP is 32-bit data width. This 32-bit data has 4x 8-bit pixels parallel output.

When transmitting high resolution video streams, using a higher pixel-clock ratio (or PPC) can reduce clock frequency pressure, and being more complatible with parallel processing units, to improving system performance.

Line Buffer

The Bicubic algorithm uses the pixels around the super-resolution pixels (hereinafter called the 'super-pixels') to perform the computation, which requires the IP core to buffer a certain number of pixels before the computing unit (CU) can start the computation. To maximize compatibility with real-time video streaming data, this IP uses full-line buffering, which using block RAM to buffer the entire line of input frame.

The super-resolution calculation also requires the image to be padded around. Controller of this IP can do the padding automatically with line buffers and pixel block buffers without user intervention.

Note that due to the special line buffering and padding mechanism, please ensure that the input video stream supproted backpressure and blocking (this can supported by using an AXI4-Stream FIFO). Also, please make sure that the resolution and start of frame (SOF) position of input video stream are correct. In this IP, the SOF signal of the input video stream is only used to pass-through to the output video stream and cannot reset the line buffer counter. If an error frame transmission occurs, please use the synchronous reset pin (SCLR) of this IP to reset before starting the correct frame transmission, otherwise, you may encounter an unexpected situation.

Bicubic Scaling Up Pipeline

The Real-time Video Bicubic Super-resolution IP uses a fully pipelined Bicubic interpolation unit. It can scale up the resolution of the input video stream by a fixed factor of 16X (4X each for width and height). The latency of the scaling up process of any video stream is fixed.

Multiple Resolution Support

The Real-time Video Bicubic Super-resolution IP has ability to process various resolutions of videos by modifying parameters. This modification supported in the hardware synthesis phase only, and it is not possible to configure the IP dynamically to support multiple resolutions. Depending on the required resolution, the resource usage of the Real-time Video Bicubic Super-resolution IP may vary.

1.2 Applications

The Real-time Video Bicubic Super-resolution IP provides the ability to scale up the LR video streams to HR in real time.

1.3 Unsupported Features

The following features are not supported by the Real-time Video Bicubic Super-resolution IP.

- Error frame detection and automatic recovery. Error frame includes resolution mismatched and SOF position misaligned.

- Output stream backpressure and blocking.
- Dynamic resolution.

2 Product Specification

2.1 Performance

The Real-time Video Bicubic Super-resolution IP is benchmarking by timing analysis tools in Vivado and Synopsys Synplify Premier. The parameters used in the benchmark test were configured as follows

- Input Video Width: 960
- Input Video Height: 540

Table 1 shows the results of the maximum frequencies benchmark.

FPGA Device Family	**Analysis Tool**	**Fmax (MHz)**
Xilinx Virtex UltraScale+	Synopsys Synplify Premier 2020.03[1]	628.6
Xilinx Kintex UltraScale+	Synopsys Synplify Premier 2020.03[1]	417.2
Xilinx Zynq UltraScale+	Vivado 2021.1[1][2]	428.4
	Synopsys Synplify Premier 2020.03[1]	394.1
Xilinx Kintex 7	Synopsys Synplify Premier 2020.03[3]	361.7
Xilinx Artix 7	Synopsys Synplify Premier 2020.03[3]	192.2
Intel Stratix 10	Synopsys Synplify Premier 2020.03[3]	260.7
Intel MAX 10	Synopsys Synplify Premier 2020.03[3]	220.5
Intel Arria V	Synopsys Synplify Premier 2020.03[3]	142.5

Table 1: Maximum Frequencies

Latency

Table 2 shows the Real-time Video Bicubic Super-resolution IP latency cycles measured on real-time video path. It does not include system dependent latency or throttling. Suppose the width of the input video frame is W.

Description	**Clocks**
Bicubic pipeline input to output	13
First pixel input to First pixel (group) output	$10 \cdot W + 28$
Last pixel input to last pixel (group) output	$13 \cdot W + 31$

Table 2: Latency

[1] Using DSP48E2 Macro.
[2] Using XPM Macro.
[3] Using pure Verilog inferring synthesis.

Throughput

Table 3 shows the Real-time Video Bicubic Super-resolution IP throughput measured for different input frame size.

Input Resolution	Output Resolution	Throughput (FPS/MHz)	FPS @150MHz
320 x 240	1280 x 960	3.23	484.7
480 x 270	1920 x 1080	1.92	287.6
640 x 360	2560 x 1440	1.08	162.1
960 x 540	3840 x 2160	0.48	72.1

Table 3: Throughput

2.2 Resource Utilization

Table 4 shows the Real-time Video Bicubic Super-resolution IP resource utilization on specified FPGA devices with specified settings for reference.

Resource utilization results of Xilinx Zynq UltraScale+ devices is evaluated under with Vivado synthesizer and using DSP48E2 and XPM Macros. The evaluation result of other devices are complete using Verilog automatically inferring, may different caused by the difference of data width of DSP module of each device. The Real-time Video Bicubic Super-resolution IP is specially optimized for DSP48E2 block of Xilinx UltraScale+ series devices. In order to maximum the resource utilization, it is recommended to use these devices for synthesis.

Device	Configuration Parameters		Resource Utilization			
	Input Resolution	fCLK (MHz)	LUTs	FFs	DSPs	BRAMs
XCZU15EG	960 x 540	300	431	694	49[1]	2.5[3]
XC7K325T	960 x 540	150	1979	2713	30[2]	2[3]

Table 4: Resource Utilization

[1] Xilinx UltraScale+ architecture DSP48E2 unit
[2] Xilinx 7 series FPGA DSP48E1 unit
[3] Xilinx FPGA Block RAM 36K unit

2.3 Port Descriptions

This section describes the details for each interface. The Real-time Video Bicubic Super-resolution signals are described in Table 5

Signal Name	Interface	Signal Type	Description
Clock, Reset and Clock Enable Signals			
clk	Clock	I	AXI4-Stream interface clock
aresetn	Reset	I	Active-Low asynchronous reset. When asserted Low, resets entire Real-time Video Bicubic Super-resolution core.
dsp_reset	Reset	I	Active-High synchronous reset. When asserted High, resets all DSP blocks (or computation pipelines). Must be synchronous to aclk and asserted for a minimum of sixteen clock cycles.
bram_reset	Reset	I	Active-High synchronous reset. When asserted High, resets all BRAM blocks (line buffers). Must be synchronous to aclk and asserted for a minimum of 16 clock cycles.
sclr	Reset	I	Active-High synchronous reset. When asserted High, resets all control units (line and column counters). Must be synchronous to aclk and asserted for a minimum of 16 clock cycles.
clken	Clock Enable	I	Active-High clock enable. When asserted High, the entire IP has clock to work, otherwise the clock is disabled.
AXI4-Stream Video Input Interface Signals			
s_axis_video_in*	S_AXIS_VIDEO_IN	-	AXI4-Stream Video Interface (Sink)
AXI4-Stream Video Output Interface Signals			
m_axis_video_out*	M_AXIS_VIDEO_OUT	-	AXI4-Stream Video Interface (Source)

Table 5: I/O Signal Description

2.4 Timing Diagrams

For the timing diagrams of the AXI4-Stream Video Interface, you can refer to the **AXI4-Stream Signaling Interface** section of the *AXI4-Stream Video IP and System Design Guide* (Xilinx Inc. UG934).

3 Algorithm Description

In mathematics, bicubic interpolation is an extension of cubic interpolation for interpolating data points on a two-dimensional regular grid. The interpolated surface is smoother than corresponding surfaces obtained by bilinear interpolation or nearest-neighbor interpolation.

In image processing, bicubic interpolation is often chosen over bilinear or nearest-neighbor interpolation in image resampling. In contrast to bilinear interpolation, which only takes 4 pixels (2x2) into account, bicubic interpolation considers 16 pixels (4x4). Images resampled with bicubic interpolation are smoother and have fewer interpolation artifacts.

3.1 Terminology

Original-image The source (or input) image of the super-resolution algorithm, which has low resolution (W_i, H_i).

Original-pixels Pixels in the *Original-image*. The index of these pixels are from $(0, 0)$ to (W_i, H_i)

Super-image The result (or output) of the super-resolution algorithm. Its resolution (W_o, H_o) is a specific multiple of the *Original-image* ($k \cdot (W_i, H_i)$).

Super-pixels Pixels in the *Super-image*. The index of these pixels are from $(0, 0)$ to (W_o, H_o).

Pixel reference In the image interpolation algorithm, a new *Super-pixel* is generated from the *Original-pixels* (These pixels are generally around the target *Super-pixel*). The *Original-pixels* associated with a new *Super-pixel* are called the *Reference of this Super-pixel*.

Super-block The *Super-block* refers to the area between every four 2 x 2 *Original-pixels*. At a super-resolution of 4 (i.e. $(W_o, H_o) = 4 \cdot (W_i, H_i)$), this area is filled with 16 *Super-pixels*, which are referenced to the 16 *Original-pixels* surrounding the area. In other words, the *Original-pixels* referenced by the *Super-pixels* within a *Super-block* are identical, except that the distance between each *Super-pixel* and the *Original-pixel* is different.

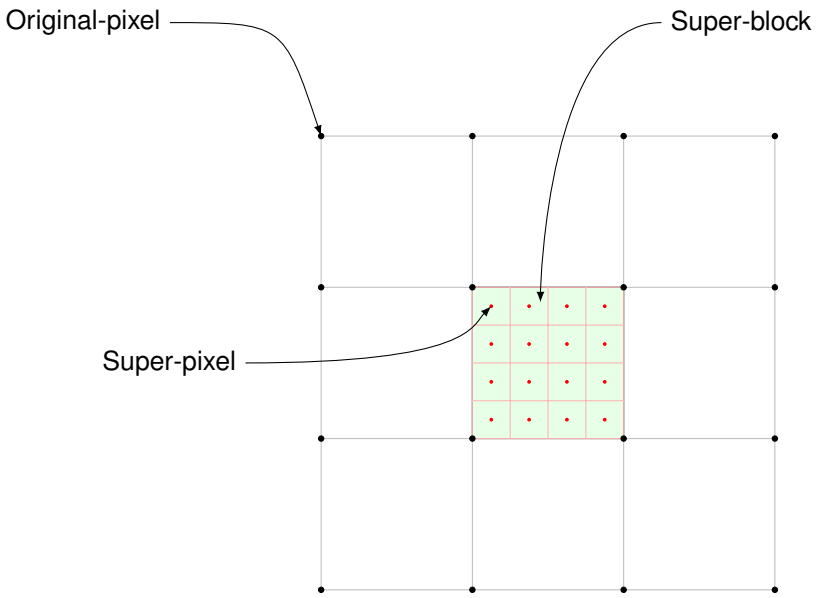


Figure 2: Schematic of reference pixels of the Super-block, Original-pixels, Super-pixels

3.2 Bicubic Algorithm Process

As a whole, the Bicubic interpolation algorithm computes the distance between the center of the *Super-pixel* and the center of the *Original-pixel*. Then, calculates the coefficient of this pair by a Bicubic interpolation function with the distance. Use this coefficient to multipiles with the value of the *Original-pixel*. Finally, the total of the product of 16 *Original-pixels* in the *Super-block* is the value of new *Super-pixel*.

Here is the formula of the Bicubic algorithm.

$$\mathrm{SP}_{\vec{s}} = \sum_{\vec{o} \in \mathrm{RSB}} f_B\left(|\vec{s} - \vec{o}|\right) \cdot \mathrm{OP}_{\vec{o}}$$

where $\mathrm{SP}_{\vec{s}}$ is the value of the *Super-pixel*, index is $\vec{s}$. RSB is the set of all *Original-pixels* index of the *Super-block* where the current *Super-pixel* is located. $OP_{\vec{o}}$ is the *Original-pixel* value on the index $\vec{o}$. The function f_B is the Bicubic interpolation, which returns a distance-dependent coefficient.

3.3 Bicubic Interpolation Function

The Bicubic interpolation function is a function dependent on distance between *Super-pixel* and reference *Original-pixel*. Denoting the distance by $|x|$, the Bicubic interpolation function takes the form

$$f_B(|x|) = \begin{cases} (\alpha+2)\cdot|x|^3 - (\alpha+3)\cdot|x|^2 + 1 & \text{, if } |x| \le 1 \\ \alpha\cdot|x|^3 - 5\cdot\alpha\cdot|x|^2 + 8\cdot\alpha\cdot|x| - 4\cdot\alpha & \text{, if } 1 < |x| < 2 \\ 0 & \text{, otherwise} \end{cases}$$

where $\alpha = -0.5$ for a general image quality.

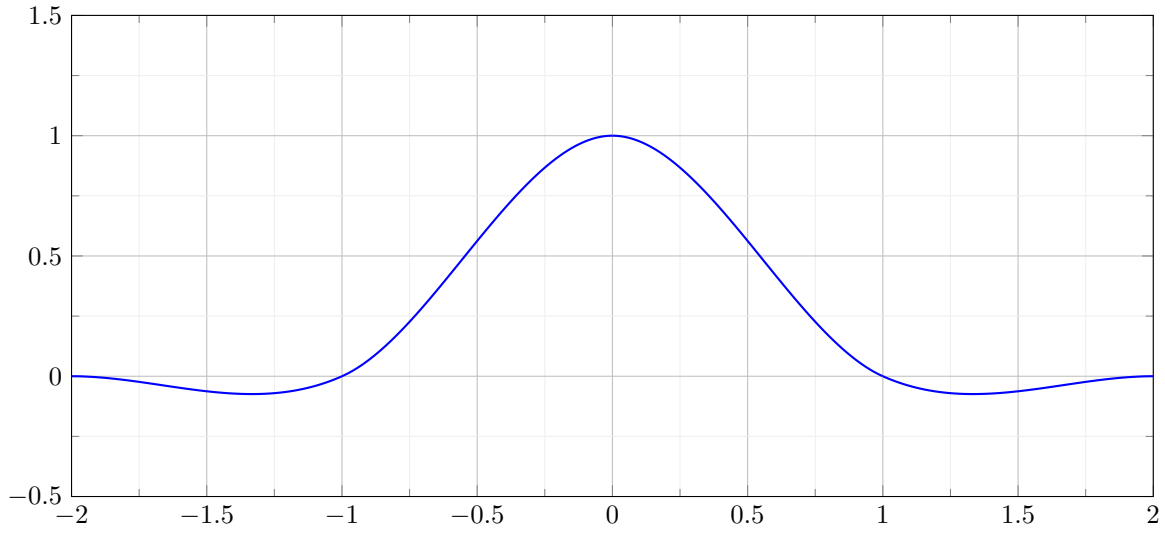


Figure 3: Plot of the Bicubic Interpolation Function

3.4 Super-pixel Symmetry in Super-block

The Bicubic interpolation function has one and only one return value at the same distance input, which means that any two *Super-pixels* and *Original-pixels* at same distance have the same coefficient. In a *Super-block*, there exist several groups of *Super-pixels* and *Original-pixels* pair, which have the same distance between each of them, because of their symmetric positions. For example, a symmetric group of the 4x super-resolution *Super-block* is shown in Figure 4. The four blue arrows shows the distance of corresponding *Super-pixel*-*Original-pixel* pair are same.

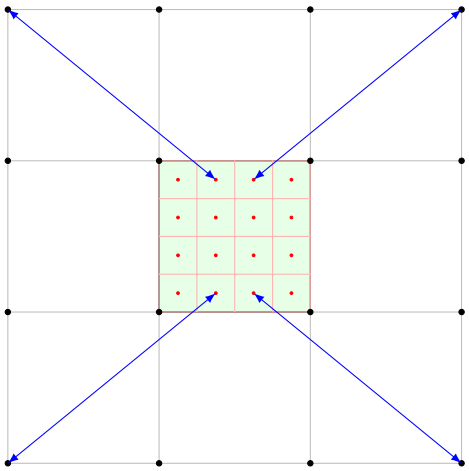


Figure 4: Example of a Symmetric Group of a 4x Super-resolution Super-block

Divide the 4x super-resolution *Super-block* into 4 parts, and then number the *Original-pixels* and *Super-pixels* separately as $O(0,0)$ to $O(3,3)$ and $S(0,0)$ to $S(3,3)$ like shown in the Firuge 5. Denote the distance from any *Super-pixel* $S(x,y)$ to any *Original-pixel* $O(m,n)$ as $d\left[(x,y) \rightarrow (m,n)\right]$.

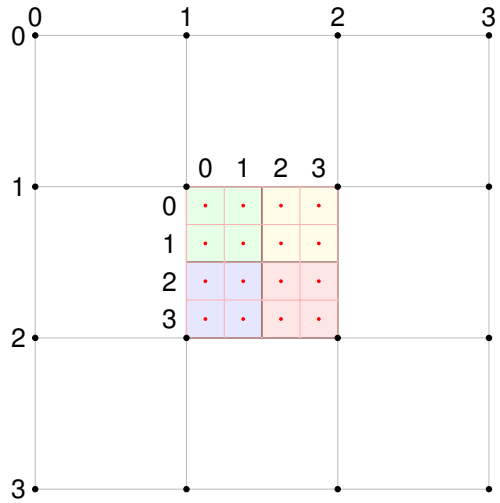


Figure 5: Symmetric Groups in the Super-block

We can find that for a single *Super-pixel*, there are 16 distance values (i.e. 16 coefficients) to 16 *Original-pixels*. For example, the *Super-pixel* $S(0,0)$ has $d\,[(0,0) \to (0,0)]$, $d\,[(0,0) \to (1,0)]$, ..., $d\,[(0,0) \to (3,3)]$. However, when we consider the symmetry of the *Super-pixel* in the *Super-block*, we can find that

$$d\,[(0,0) \to (0,0)] = d\,[(3,0) \to (3,0)] = d\,[(0,3) \to (0,3)] = d\,[(3,3) \to (3,3)]$$

In this way, we only need to consider the coefficients of the *Super-pixels* in the green part of Figure 5. We can find the coefficients of the remaining part of the *Super-pixels* by symmetry. The yellow and green parts are mirrored vertically along the center; the blue and green parts are mirrored horizontally along the center; and the red part is mirrored both vertically and horizontally. Using this property, the number of calculations of the coefficients is reduced to a quarter.

3.5 Pixel Alignment and Edge Padding of the Super-pixels

Super-pixels are not exist on the input LR image. Any *Super-pixels* indexed by $S(x,y)$ need mapped to the absolute coordinate on the input LR image using this way as follows:

$$\vec{S}_{map}(x_m, y_m) = \frac{\vec{S}(x,y) + (0.5, 0.5)}{k}$$

where k is the scale coefficient.

When $k = 4$, the top-left *Super-pixel* is mapped to the coordinate $(-0.375, -0.375)$. This means that the center of the top-left *Super-pixel* $S(0,0)$ is on the left and top of the *Original-pixel* $O(0,0)$.

In Figure 6, we can see that the center of $S(0,0)$ is located to the left and top of the center of O(0,0) and belongs to the *Super-block* in the blue box. 16 pixels from O(-2,-2) to O(1,1) are referenced to this *Super-block*.

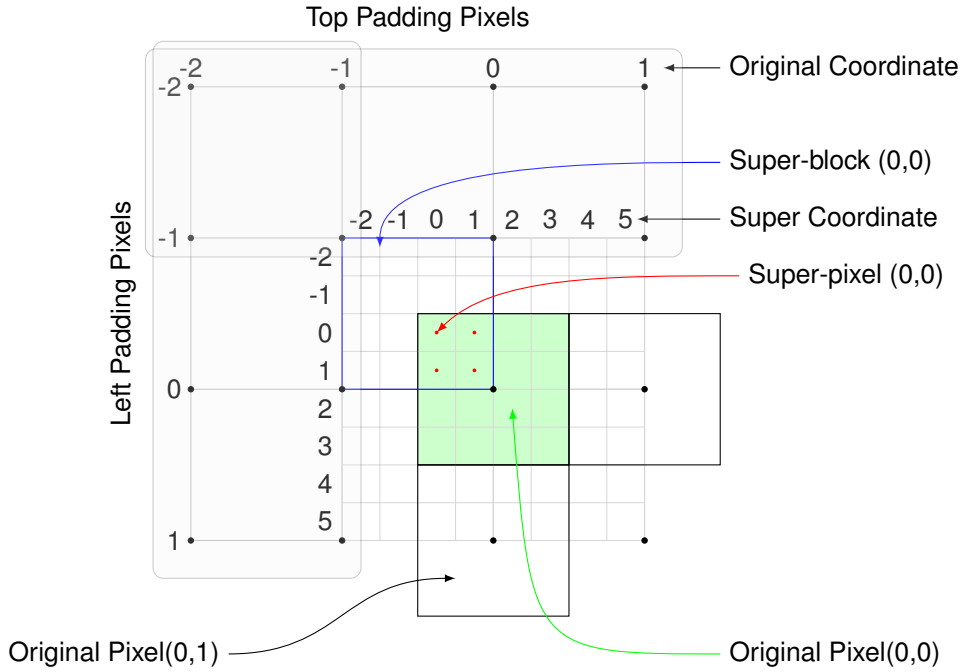


Figure 6: Super-pixel Alignment and Padding at the Top-Left of the Image

Note that $S(0,0)$ is not at postion $(0,0)$ but at position $(2,2)$ in the *Super-block*. If $S(0,0)$ is incorrectly computed using the coefficients of the $(0,0)$ position in *Super-block*, it will cause pixel alignment problems and affect the output result.

Since the *Original-image* does not have pixel values in negative coordinates and the coordinates out of the image size for reference, we must padding 2 rows and 2 columns of the edge of the image to provide them. One way is to padding the pixel using the nearest pixel, i.e., copy 2 rows(or 2 columns) of pixels on the edge of the *Original-image*. In this way, when we using the value of $O(-1,0)$ or $O(1,-2)$, we are actually using the value of $O(0,0)$ and $O(1,0)$.

After pixel alignment and padding, the entire super-resolution image has (W_i+1) x (H_i+1) *Super-blocks*, where only 2 rows (or columns) of *Super-pixels* are used in the boundary *Super-blocks*. And only a quarter *Super-pixels* of the *Super-pixels* located on the corners are used.

4 Microarchitecture Design

The Real-time Video Bicubic Super-resolution IP provides a complete Bicubic processing implementation. Including a real-time Bicubic interpolation pipeline, a line buffer module and a control unit for controlling the above 2 modules.

Because of the requirements of padding and pixel alignment processing, this IP also contains a pixel realignment module.

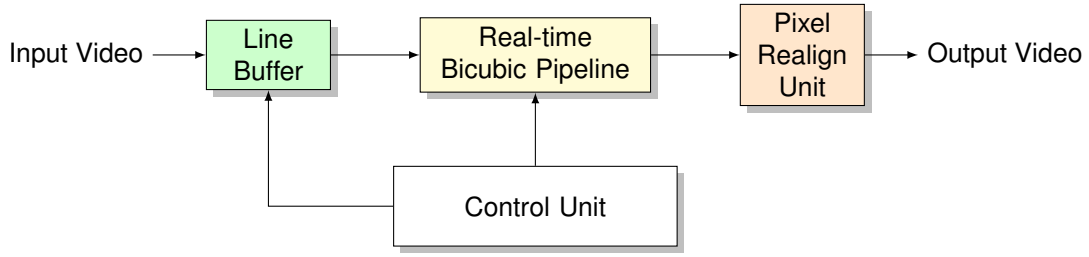


Figure 7: Real-time Video Bicubic Super-resolution IP Processing Units

4.1 Operational Bit-width and Quantization

In order to save resource consumption, maximize DSP resource utilization, and to optimize for the DSP48E2 unit, the Real-time Video Bicubic Super-resolution IP performs 9-bit signed quantization of the Bicubic coefficients during the operation, which ensures image quality while fully utilizing the multiplier of the DSP48E2 unit. This quantization method supports 8-bit image channel inputs.

4.2 Real-time Bicubic Pipeline

The real-time Bicubic pipeline contains a *Super-block* buffer, a Bicubic coefficient lookup table (LUT), a set of parallel multiplier-adder units, a set of parallel adders, a set of parallel rounding units and a limit unit. The structure of the pipeline is shown in Figure 8.

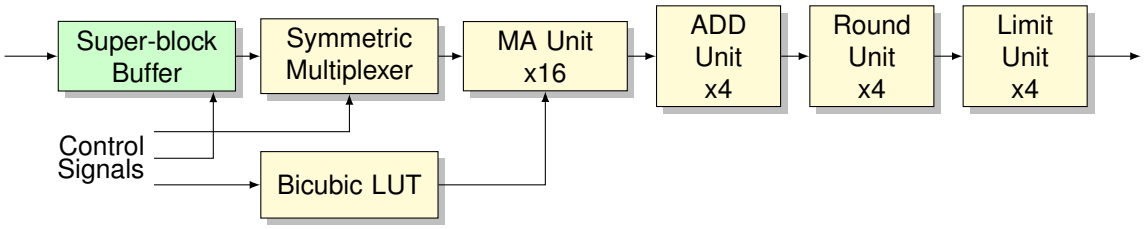


Figure 8: Real-time Bicubic Pipeline

The value of each *Super-pixel* is the sum of the product of 16 referenced *Original-pixel* and the corresponding Bicubic coefficient. A single *Super-pixel* requires at least 16 multiplication and addition operations for it.

The Real-time Video Bicubic Super-resolution IP performs 4 *Super-pixels* per clock cycle.

4.2.1 Super-block Buffer

The *Super-block* Buffer sotres the current *Reference-pixels* used by the *Super-block* for computation. It has 4 pixel inputs which shifts in 1-column x 4-row of *Original-pixels* each clock cycle. It also has an 4-column x 4-row pixel buffer matrix. Each row of which is a shift register that allows the *Super-block* to slide horizontally on the *Original-image*.

The *Super-block* Buffer can output 16 buffered *Original-pixels* at a clock cycle for computation. To meet the needs of pixel alignment and edge padding, the data of each output column is selected using a multiplexer. There are five modes of the multiplexer, as shown in Table 6, available for the 4 padding cases at the left and right edges of the *Original-image*, and the normal case at the middle of the *Original-image*.

Mode	**Column Outputs**				**Description**
	0	**1**	**2**	**3**	
0000	Col0	Col1	Col2	Col3	Normal
0001	Col1	Col1	Col2	Col3	Padding left 1 column
0010	Col2	Col2	Col2	Col3	Padding left 2 columns
0100	Col0	Col1	Col2	Col2	Padding right 1 column
1000	Col0	Col1	Col1	Col1	Padding right 2 columns

Table 6: Super-block Buffer Output Multiplexer Modes

Figure 9 shows the structure of the *Super-block* Buffer.

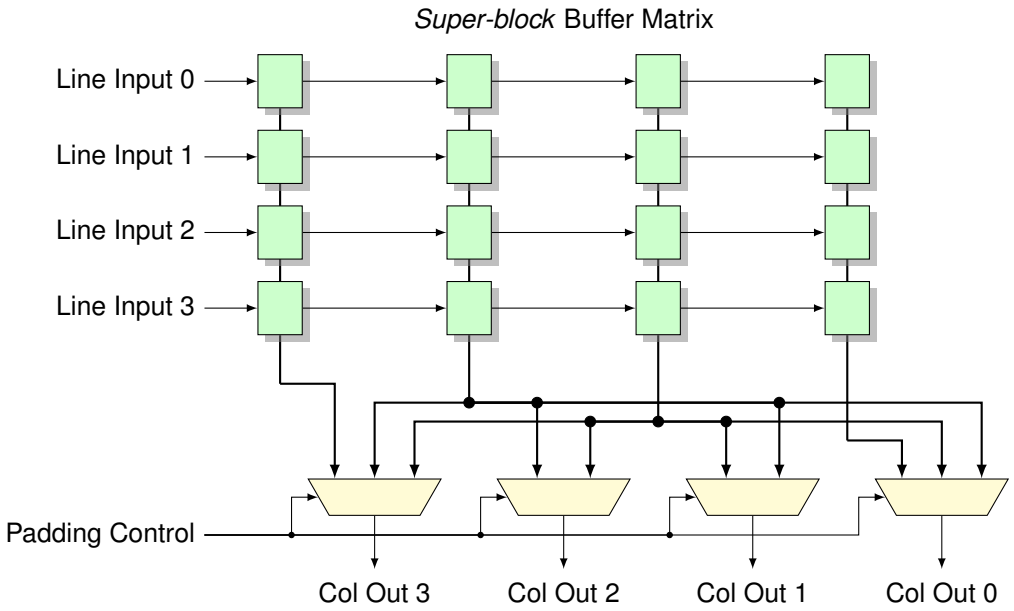


Figure 9: Diagram of the Super-block Buffer

4.2.2 Symmetric Multiplexer

The Symmetric Multiplexer is used to flip the pixel matrix of the Super-block Buffer output vertically. We are using symmetry to reduce the size of the Bicubic LUT. When the Super-pixel to be computed is located in row 2 or 3 of a Super-block, the coefficient of its corresponding Original-pixel is vertically mirrored with the Super-pixel located in row 1 or 0. For example, in Figure 10, $d[(1,2) \rightarrow (0,3)] = d[(1,1) \rightarrow (0,0)]$(Red arrows) and $d[(2,3) \rightarrow (2,1)] = d[(2,0) \rightarrow (2,2)]$(Blue arrows).

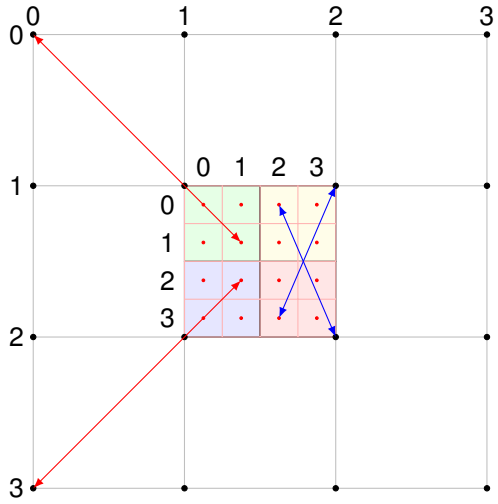


Figure 10: Example of Vertical Symmetric in a Super-block

Thus, by directly mirroring the Original-pixels in the entire *Super-block* horizontally along the center, it is possible to compute the values of the *Super-pixel* located in rows 1 or 2 without changing the order of the coefficients.

4.2.3 Bicubic Coefficient LUT

The Bicubic Coefficient LUT is used to lookup the Bicubic coefficients used in the product with the *Original-pixels*. This IP exploits the symmetry of *Super-blocks* in horizontal and vertical directions. In the horizontal direction, *Super-pixels* located in columns 2 or 3 (in each *Super-block*) can using the coefficients of columns 1 or 0. In the vertical direction, *Super-pixels* located in rows 2 or 3 can using the coefficients of rows 1 or 0. This allows the Bicubic Coefficient LUT to store a total of only 2-row x 2-*Super-pixel* x 16 coefficients. The Bicubic Coefficient LUT outputs 1-row x 2-*Super-pixel* x 16 coefficients each clock cycle. The outputs of rows located in 0 and 3 are same, and the outputs of rows located 1 and 2 is also the same. The symmetric conversion is done by the Symmetric Multiplexer.

Coefficient output count in single clock cycle in the Bicubic Coefficient LUT can meet the requirement of 4 Super-pixels output in a single clock cycle of this IP.

4.2.4 Multiplier Adder Unit (MA Unit)

The Real-time Bicubic Pipeline contains 16 MA Units, each of them multiplies 4 sets of *Original-pixels*-Coefficients and adds them two by two in one cycle.

The mathematical structure of the MA unit is shown in Figure 11.

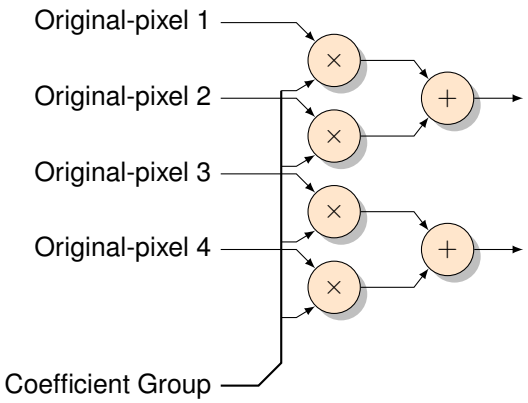


Figure 11: Mathematical Structure of MA Unit

Using the horizontal symmerty, when computing 4 *Suepr-pixels* located in the same row of a *Super-block* in parallel, a single Bicubic coefficient can be multiplied with 2 different *Original-pixels* for different *Super-pixels*. For example, since the coefficient $C_0 = f_B(d[(0,0) \rightarrow (0,0)]) = f_B(d[(3,0) \rightarrow (3,0)])$, multiplying the *Original-pixel* located at (0,0) and (3,0) in the *Super-block* reference pixel matrix with C_0 at same time, the partial values used for *Super-pixel* (0,0) and (3,0) can be obtained, respectively.

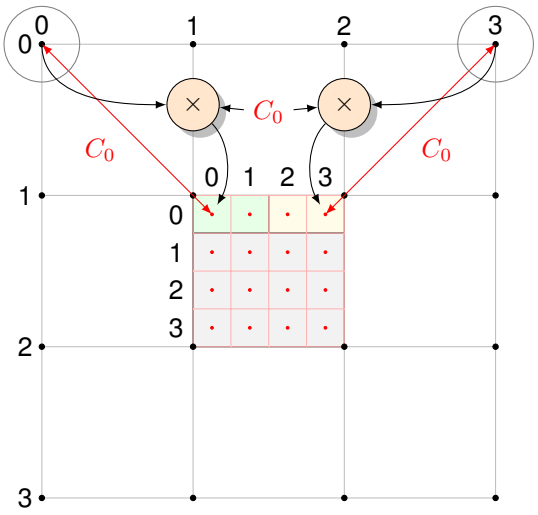


Figure 12: Example of Multiplication Operation Using Horizontal Symmetry

Split-multiplier Structure (SMS) To save multiplier resources and fully utilize the 18x27-bit multiplier of the DSP48E2 slice in the Xilinx UltraScale+ FPGA architecture, this IP innovatively proposes a single DSP48E2 split-multiplier structure (SMS), which splits a signal 18x27-bit multiplier into two 8x9-bit multipliers. This design not only halves the resource consumption of the multiplier, but also reduces the length of the wiring and improves the IP performance.

The SMS structure uses the mathematical relationship of binary multiplication to split the multiplier. The basic principle is derived from the following equation

$$M = C \cdot (B \cdot 2^n + A) = C \cdot A + C \cdot B \cdot 2^n$$

where C is the common coefficient to be multiplied and A and B are the two multipliers.

When we want to compute $M_1 = A \cdot C$ and $M_2 = B \cdot C$, we can first shift B left by n bits, i.e., $B \cdot 2^n$, then add it to A, and multiply it with C. Get the result $M = C \cdot (B \cdot 2^n + A)$, which is the result of adding M_1 and M_2 shifting left by n bits.

In binary multiplication, if the multipliers A and B are both x-bit binary numbers and C is a y-bit binary number, then neither $A \cdot C$ nor $B \cdot C$ will result in more than $x + y$ bits.

Therefore, when $n > x + y$, M_1 in $M = C \cdot (B \cdot 2^n + A)$ will not round to the high bit where M_2 is, so that M_1 and M_2 can be separated directly. Of course, when computing with signed numbers, if the result of M_1 is negative, it will cause it to round to the high bit by 1 bit, which can be compensated by the sign bit of M_1.

In this IP, the 9-bit signed Bicubic coefficients and 8-bit unsigned pixel data are used to multiply. Due to the symmetry of the Bicubic coefficients, the form of the split multiplication $M = C \cdot (A + B)$ is exactly satisfied, so that the splitting method can be used to operate two multiplication operations in a single 18x27-bit multiplier in a single clock cycle.

In this IP, the SMS structure multiply unit contains an 18x27-bit multiplier that inputs 2 *Original-pixels* A and B and a Bicubic coefficient input C. The 18-bit input of the multiplier is directly connected to the C input. And the 27-bit input of the multiplier is connected to the merged lines consists of A and B which is left-shifted by 18 bits, i.e., $A + B \cdot 2^{18}$.

Thus, in the output result of the multiplier, the bits [17:0] are the result M_1 and the bits [35:18] are the result M_2. Since the Bicubic coefficients are signed, the result of M_2 also needs to be compensated based on the most significant bit (sign bit) of M_1. The computation is shown in the Figure 13.

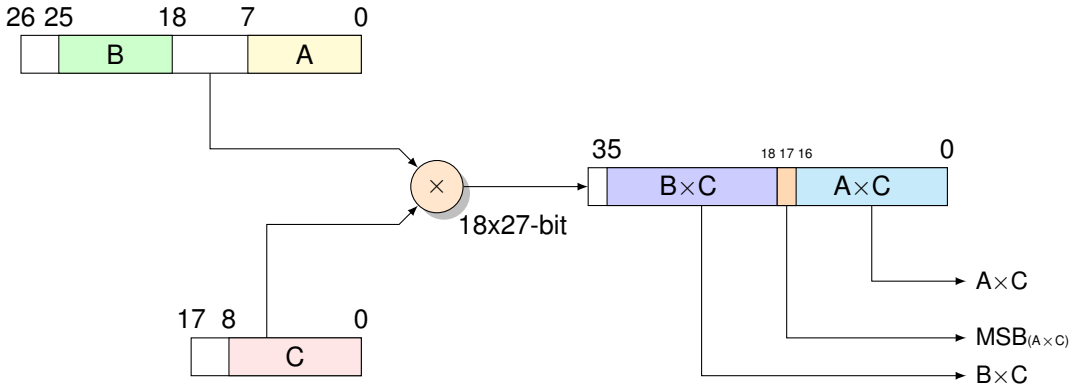


Figure 13: Bicubic Computation Process of SMS Structure

By connecting two such SMS units in series, an MA unit is obtained. The two SMS cells are used as two stages. The 1st stage can perform 2 multiplications (2 *Original-pixels* x 1 Coefficent), and output 2 results in parallel. The 2nd stage, while completing the 2 multiplications, can add the result of the 1st stage and 2nd stage together using the 48-bit adder in the DSP48E2 slice (or DSP slice). And finally when a MA unit completes the operation, it can get 2 summation result in parallel. The structure and data flow of the MA unit is shown in Figure 14.

The MA unit uses the equations are

$$M = C_1 \cdot \left(A_1 + B_1 \cdot 2^{18}\right) + C_2 \cdot \left(A_2 + B_2 \cdot 2^{18}\right)$$

$$M_1 = M_{[17:0]} = C_1 \cdot A_1 + C_2 \cdot A_2$$

$$M_2 = M_{[35:18]} - M_{[17]} = C_1 \cdot B_1 + C_2 \cdot B_2 - M_{[17]}$$

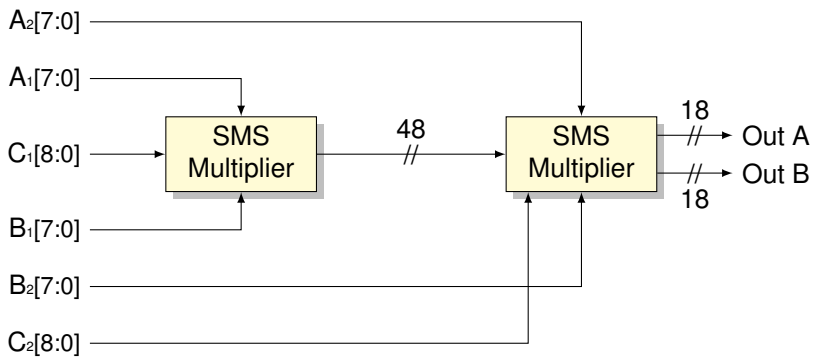


Figure 14: Structure and Data Flow of MA Unit

Because of the feature of the SMS structure for signed multiplication, all the *Original-pixels* input as B need to be compensated with the most significant bit of the result of $C \cdot A$ from same SMS unit.

Low Segment and High Segment We call the result of the A-way computation without compensation *Low Segment*, and the result of the B-way computation with compensation *High Segment*.

Bicubic Coefficient Assignment Taking advantage of the symmetry of the Bicubic coefficients, this IP assigns the coefficients and *Original-pixels* to the 16 MA units reasonably, and the assignment results are shown in Table 7. Each word string of the form $X \rightarrow (x, y)$ refers to a Bicubic coefficient. The X means X-direction index of the *Super-pixel* in the *Super-block* and (x, y) is the corresponding *Original-pixel* in the *Super-block*.

With this assignment, each *Super-pixel* has 4 data in *Low Segment* and 4 data in *High Segment* of the 4 *Super-pixels* to be computed in a single clock cycle. In this way, we can input the 8 results of each *Super-pixel* into an 8-input 4-carry adder and get the result value of the *Super-pixel*.

Multiplication result of *Super-pixel* contains 4 *High Segment* results that need to be compensated, which can be compensated by the most significant bit of the corresponding *Low Segment* feed to the 4 carry inputs.

Table 7: Multiplier Adder Unit Coefficient Assignments

DSP Unit	**DSP Stage 1**			
	Coefficient	**Symmetry**	**Coefficient**	**Symmetry**
0	$0 \rightarrow (0,0)$	$3 \rightarrow (3,0)$	$0 \rightarrow (0,1)$	$3 \rightarrow (3,1)$
1	$1 \rightarrow (0,0)$	$2 \rightarrow (3,0)$	$1 \rightarrow (0,1)$	$2 \rightarrow (3,1)$
2	$2 \rightarrow (0,0)$	$1 \rightarrow (3,0)$	$2 \rightarrow (0,1)$	$1 \rightarrow (3,1)$
3	$3 \rightarrow (0,0)$	$0 \rightarrow (3,0)$	$3 \rightarrow (0,1)$	$0 \rightarrow (3,1)$
4	$0 \rightarrow (1,0)$	$3 \rightarrow (2,0)$	$0 \rightarrow (1,1)$	$3 \rightarrow (2,1)$
5	$1 \rightarrow (1,0)$	$2 \rightarrow (2,0)$	$1 \rightarrow (1,1)$	$2 \rightarrow (2,1)$
6	$2 \rightarrow (1,0)$	$1 \rightarrow (2,0)$	$2 \rightarrow (1,1)$	$1 \rightarrow (2,1)$

Continued on next page

Table 7: Multiplier Adder Unit Coefficient Assignments (Continued)

7	$3 \rightarrow (1,0)$	$0 \rightarrow (2,0)$	$3 \rightarrow (1,1)$	$0 \rightarrow (2,1)$
8	$0 \rightarrow (0,2)$	$3 \rightarrow (3,2)$	$0 \rightarrow (0,3)$	$3 \rightarrow (3,3)$
9	$1 \rightarrow (0,2)$	$2 \rightarrow (3,2)$	$1 \rightarrow (0,3)$	$2 \rightarrow (3,3)$
10	$2 \rightarrow (0,2)$	$1 \rightarrow (3,2)$	$2 \rightarrow (0,3)$	$1 \rightarrow (3,3)$
11	$3 \rightarrow (0,2)$	$0 \rightarrow (3,2)$	$3 \rightarrow (0,3)$	$0 \rightarrow (3,3)$
12	$0 \rightarrow (1,2)$	$3 \rightarrow (2,2)$	$0 \rightarrow (1,3)$	$3 \rightarrow (2,3)$
13	$1 \rightarrow (1,2)$	$2 \rightarrow (2,2)$	$1 \rightarrow (1,3)$	$2 \rightarrow (2,3)$
14	$2 \rightarrow (1,2)$	$1 \rightarrow (2,2)$	$2 \rightarrow (1,3)$	$1 \rightarrow (2,3)$
15	$3 \rightarrow (1,2)$	$0 \rightarrow (2,2)$	$3 \rightarrow (1,3)$	$0 \rightarrow (2,3)$

4.2.5 Add Unit

The Real-time Bicubic Pipeline contains 4 8-input 4-carry adders used to sum up all multiplication results of *Super-pixel*.

The arithmetic equation of the 8-input 4-carry adder is

$$Y = A + B + C + D + E + F + G + H + C_0 + C_1 + C_2 + C_3$$

where A to H is 18-bit multiplication result input, and C_0 to C_3 is the 1-bit carry input, used to compensate for the multiplication results from *High Segment*.

In order to perform so many addition operations in fewer cycles using a minimum number of DSP units, this IP is optimized for the DSP48E2 slice. An 8-input 4-carry adder consists of 2 3-input adders with carry input and 2 DSP cascaded adders with carry input. Their respective structure are shown in the Figure 15 and Figure 16.

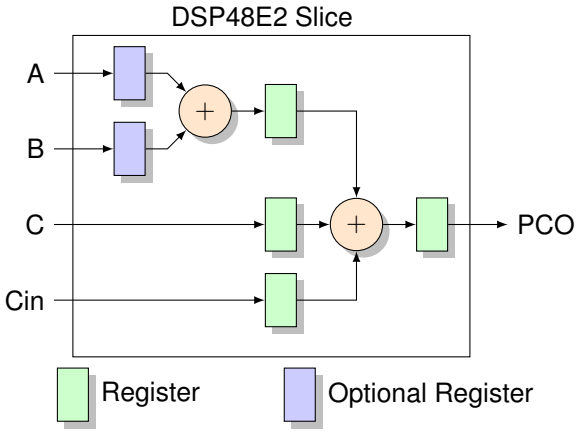


Figure 15: 3-input Adder Implemented by DSP48E2 Slice

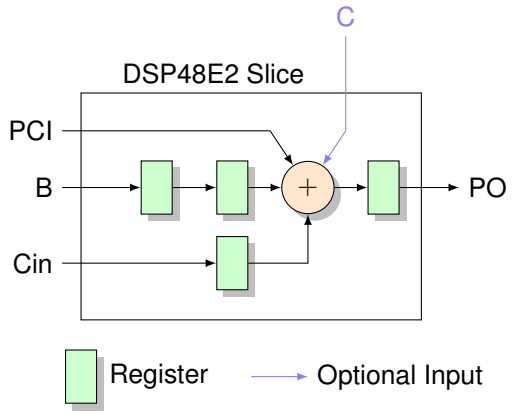


Figure 16: DSP Cascade Adder Implemented by DSP48E2 Slice

The DSP48E2 architecture has a DSP Slice Cascade Function that enables the use of cascaded outputs (PCO) and inputs (PCI) to minimize the data path length between two DSP48E2 Slices. The 8-input 4-carry adders in this IP use this design when optimizing for the DSP48E2 Slice, cascading multiple DSP48E2 slices through cascade pins to achieve an ultra-low latency adder unit. The specific structure of which is shown in the Figure 17.

The latency of the 8-input 4-carry adders in this IP is only 4 clock cycles.

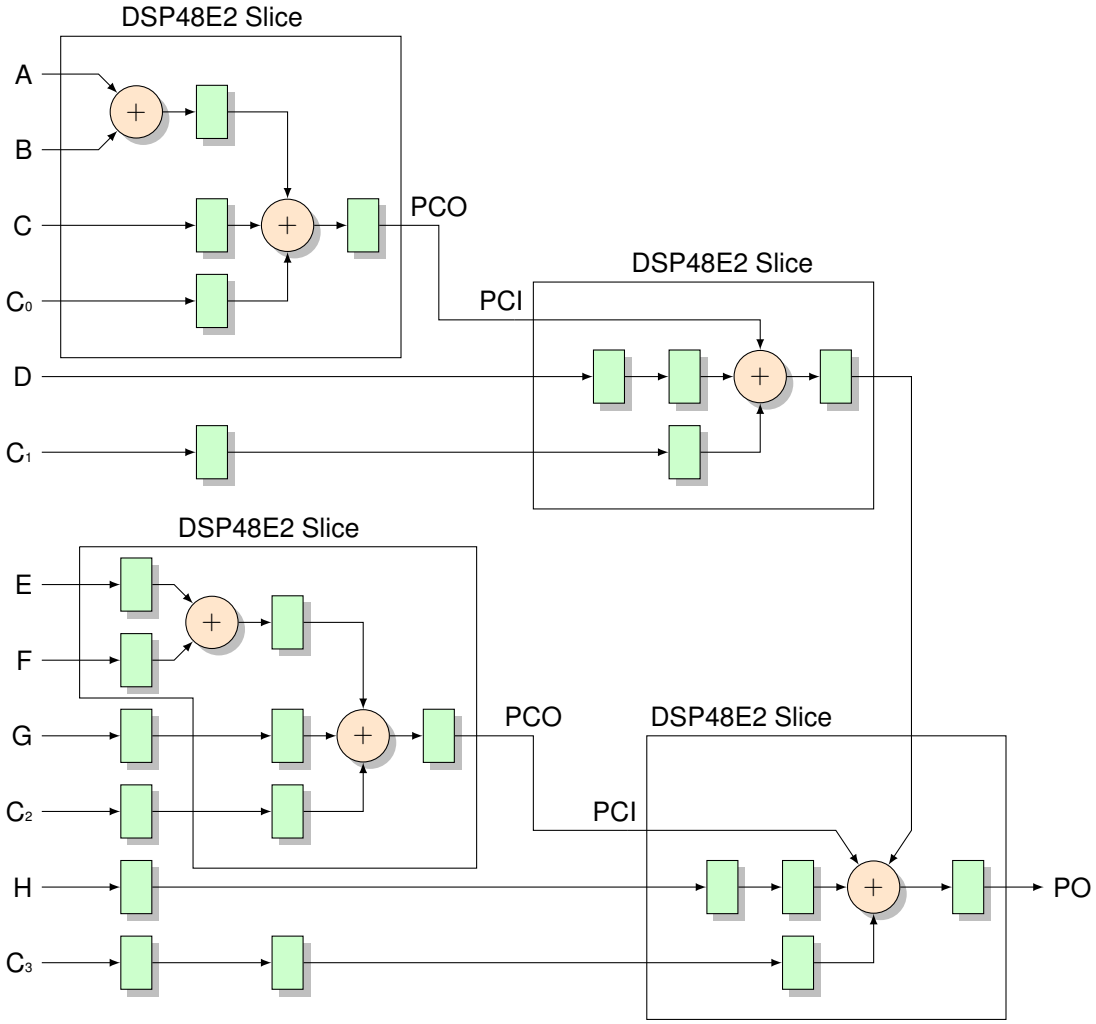


Figure 17: Structure of 8-Input 4-Carry Adder

4.2.6 Round Unit and Limit Unit

Since the fixed-point quantization is used to the Bicubic coefficients, the result of *Super-pixels* must be right-shifted and rounded before it output, and the output is limited to the standard 8-bit pixel values (0 to 255).

The Real-time Bicubic Pipeline uses a SIMD 4x12-bit adder for the rounding operation (optimized for the DSP48E2). All rounding and limiting operations are done in 2 clock cycles.

4.3 Line Buffers

There are 5 line buffers in Real-time Video Bicubic Super-resolution IP. All of them can be dynamically configured to either *working* or *buffering* state. At the same time, 4 of them will be working lines, and the remaining one will be buffering line. So that the input video stream data will not affect the super-pixel calculation. After the working lines have been calculated, the controller will automatically switch

to make the latest buffered line participate in the calculation, and configure the oldest line as buffering line to continuously receive new video pixels.

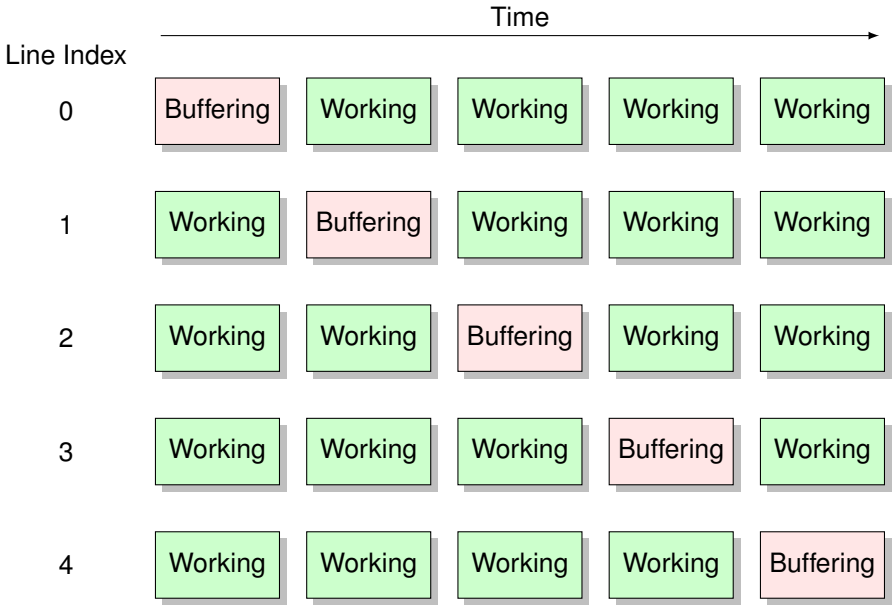


Figure 18: Working Lines and Buffering Line in Line Buffer

The line buffer contains an AXI4-Stream Video Sink interface that can be connected directly to the external video input stream. This interface has a blocking signal (TREADY) that blocks the input stream while edge padding is performed on the input video. The line buffer also has internal line counter for controlling the working lines and buffering line; and line pixel counter, for controlling the pixel out.

The line buffer outputs 1 column x 4 rows pixels from working lines in a clock cycle. Since a *Super-block* contains 4 rows of *Super-pixels*, and the *Original-pixels* corresponding to each of them are the same, the pixel counter in line buffer will perform a repeated 4 output.

The 1 column x 4 rows pixel data output from the line buffer will be directly connected to the *Super-block* buffer of pipeline. Due to the padding of the left and right edges of each line, a handshaking protocol is available between the line buffer and the pipeline controller in order to block the data input during padding.

The diagram of the line buffer is shown in the Figure 19;

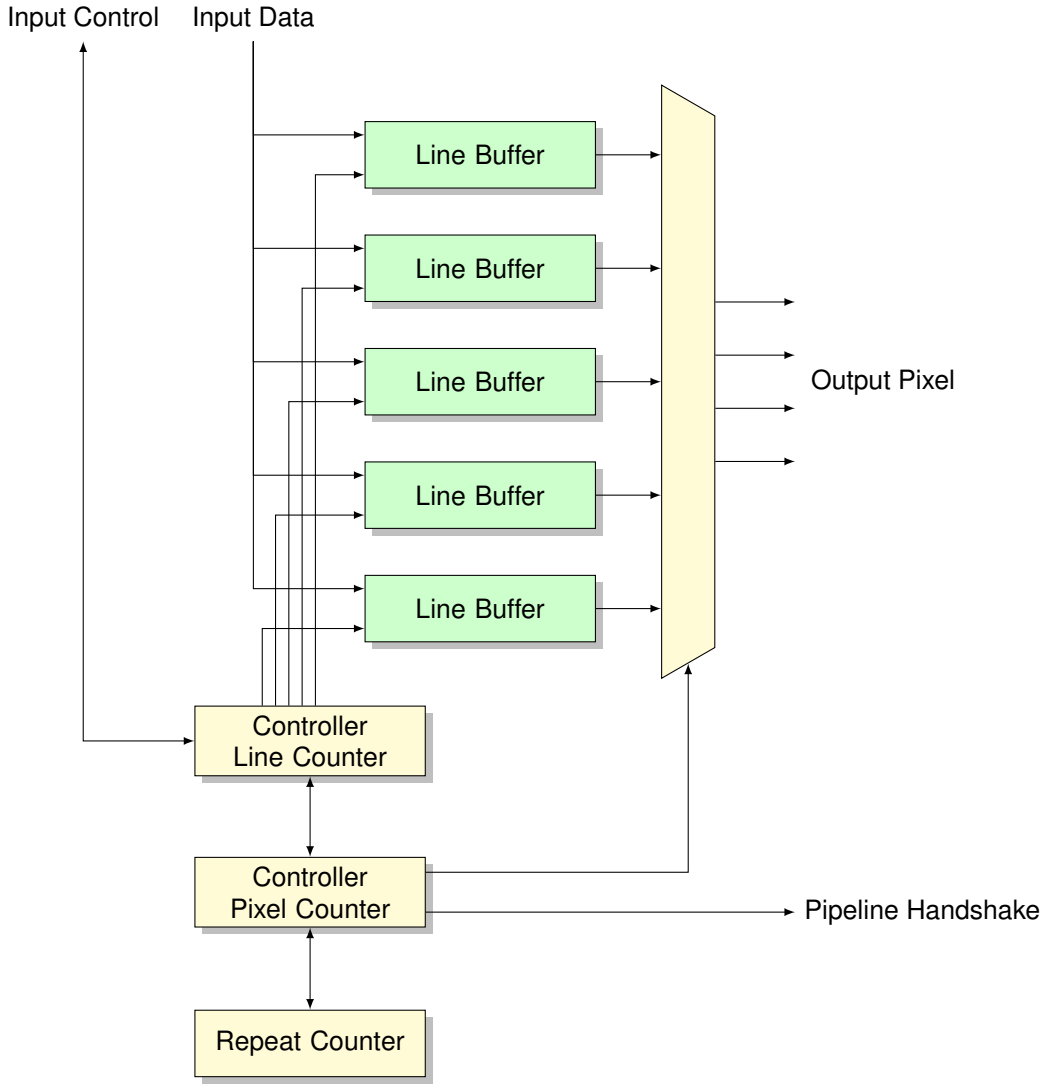


Figure 19: Diagram of Line Buffer

Padding Control in Line Buffer The line buffer blocks the input from the AXI4-Stream Video Sink interface while padding on the top and bottom eages of the frame. Each input frame has $H+1$ *Super-blocks* in the Y-direction, and we takes the time of 1 *Original-image* line input to process 1 line of *Super-blocks*, so each frame require at least 1 line of blocking time.

The line buffer will block while processing the $H-1$-line *Super-block* (when the last line of the previous frame is already stored in the line buffer), blocking the first line of the new frame until the $H-1$-line *Super-block* completes its operation. When the H-line *Super-block* starts its operation, new frames are allowed to be received again. If the input video stream is continuous, by the time the H-line and $H+1$-line *Super-block* operations are completed, the first 2 lines is already stored in the line buffer, available for the 0-line *Super-block* of new frame. In order to keep the data of the previous and the next frame from affecting each other during padding and to achieve copy-padding, the multiplexer in the line buffer can copy the data from valid working line.

The Figure 20 shows how line buffer blocking and padding. Note the **F-1** means the line or *Super-block* is from previous frame, and **F** means it is from the new frame.

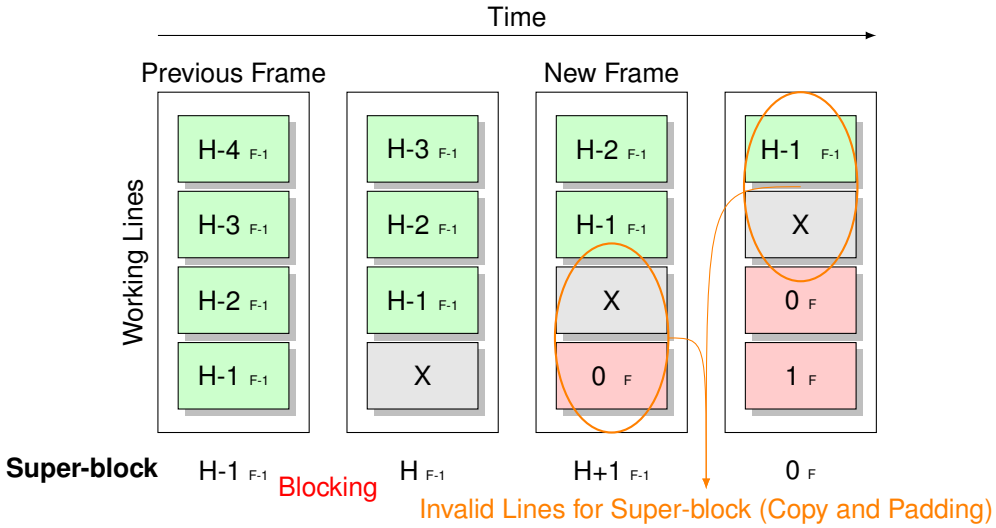


Figure 20: Line Buffer Line Padding (Top and Bottom) Between Frames

4.4 Control Unit

The control unit in the Real-time Video Bicubic Super-resolution IP is mainly used to control the following functions

1. Padding the left and right edges(Column Padding) of *Original-image*.
2. *Super-block* counting and generate line symmetry signal.
3. Generate the *Super-pixel* realign control signal.
4. Generate the control signals of AXI4-Stream Video Source interface.
5. Delay the control signals to match the latency of pipeline.

4.4.1 Column Padding

The control unit uses the same principle as the line buffer for column padding. Since the *Original-image* has $W-1$ *Super-blocks* in the X-direction, at least 1 *Super-block* blocking time is needed for padding when processing each *Super-block* line.

The control unit has an X-direction *Super-block* counter, which is able to generate a blocking signal to the line buffer during the cycle of the operation on the $W-1$-column *Super-block* of the previous line. *Reference-pixels* of new *Super-block* line will be allowed during the cycle of the W-column *Super-block* is operating.

The Figure 21 shows how control unit blocking and padding. Note the **L-1** means the *Super-block* column is from previous *Super-block* line, and **L** means it is from the new *Super-block* line.

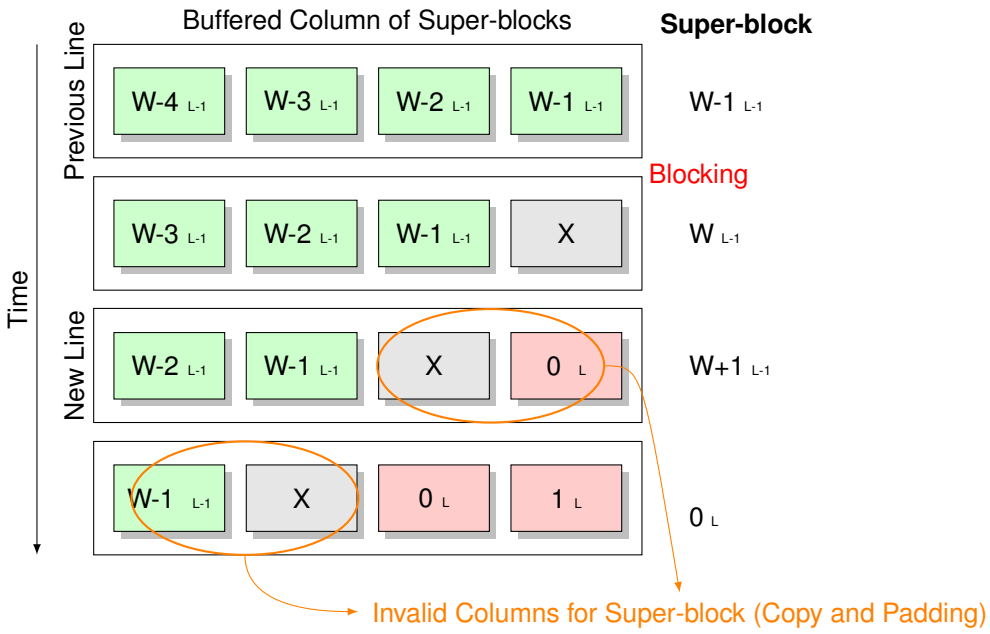


Figure 21: Control Unit Column Padding (Left and Right) Between Super-block Lines

Unlike the line buffer, the control unit completes the padding operation by controlling the multiplexer of the *Super-block* buffer in the pipeline.

4.4.2 Symmetry Control Signal Generating

There is a Y-direction *Super-pixel* counter in the control unit, which can get whether the current *Super-pixel* is in row 0, 1 or 2, 3 in the *Super-block* based on the current number of line, and generate the control signal to the symmetric multiplexer in the pipeline.

4.4.3 Super-pixel Realign Signal Generating

Due to padding and pixel alignment, only 2 *Super-pixels* are available for output when the first *Super-block* finishes its calculation, and it needs to wait until the second *Super-block* finishes its calculation to have enough data to output 4 pixels at one clock cycle. The control unit contains an X-direction *Super-pixel* counter, which can determine whether the current *Super-block* is the first column of a line, and if so, control the pixel realign module to buffer two valid *Super-pixels* in the first *Super-block*. And then starts the outputting when the next *Super-block* is processed. When processing the last *Super-block* of a line, since it also has only 2 valid *Super-pixels*, combining the last 2 *Super-pixels* with the 2 *Super-pixels* buffered in the realign module can yield the last 4 *Super-pixels* of a line of the output frame.

4.4.4 AXI4-Stream Video Source Control Signal Generating

The control unit uses the *Super-pixel* counters to determine the video frame signals (SOF and EOL), to meet the protocol of AXI4-Stream Video Interface.

4.5 Pixel Realign Unit

The pixel realign unit contains a *Super-pixel* buffer, which can buffer 2 *Super-pixels*. This unit can split the input 4 *Super-pixels* to 2 parts. Combining a part and the buffered *Super-pixels* for output, and buffering the other part into the *Super-pixel* buffer.

5 Design Verification

5.1 Verification Platform

The APV21B Real-time Video 16X Bicubic Super-resolution IP is a dedicated IP for image processing, and a bottom-up layered verification and UVM-like verification platform is used to verify the correct functionality of the IP.

To improve the algorithm verification coverage and accuracy, a mixed random + extreme verification pattern is used for the verification of computation units. The *AXI-Stream Video Image VIP* is used for bus transaction driving and monitoring for the overall verification of the IP, and a mixed verification pattern of preset bitmaps and random bitmaps is used.

The verification of the underlying arithmetic unit is performed by the SystemVerilog non-synthesizable subset for reference arithmetic and comparison. The overall verification of the IP is done by the external C model reference program and the pixel comparison program.

The block diagram of the verification platform is shown in Figure 22 and Figure 23.

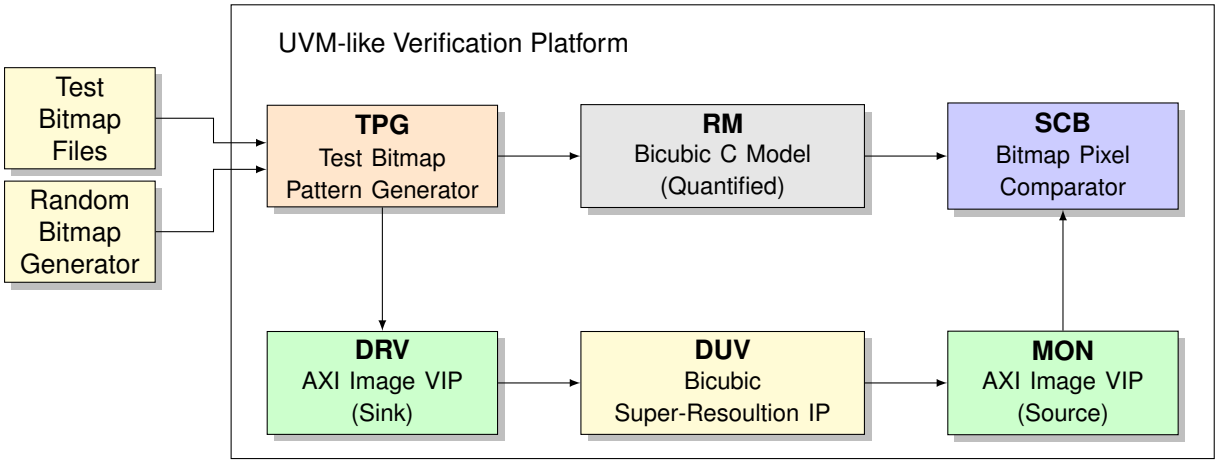


Figure 22: Diagram of UVM-like Verification Platform (Overall IP)

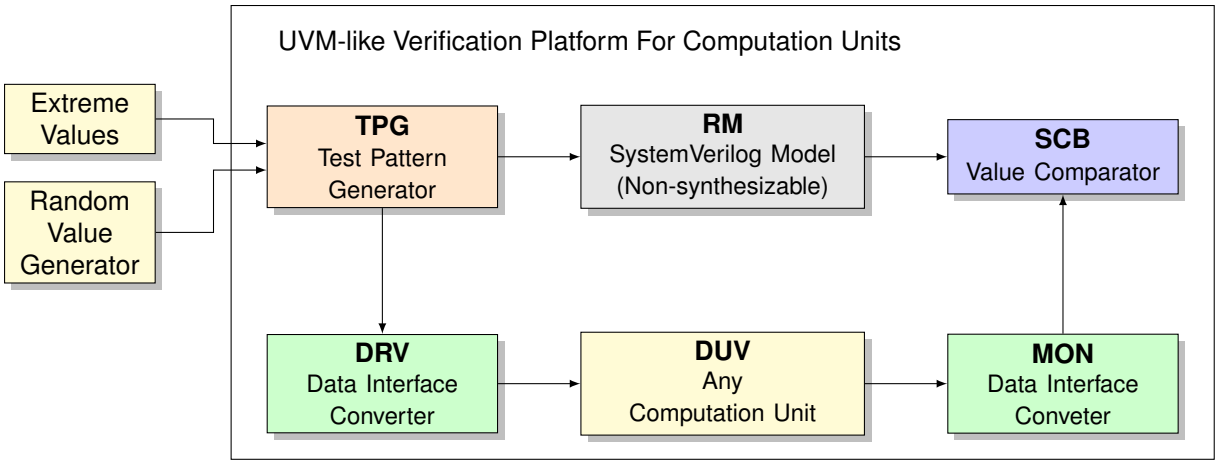


Figure 23: Diagram of UVM-like Verification Platform (Computation Units)

5.2 Software Platform

The detail information of the software platform used for IP verification is shown in Table 8.

Table 8: Software Verification Platform

Software Name	Mentor Modelsim
Version	2020.04
Referenced Libraries	Xilinx Secure IP, Xilinx XPM, Xilinx Unisims Verilog

5.3 Verification Methodology

The detail information of the verification methodology of this IP is shown in Table 9.

Table 9: Verification Methodology

Verification Language	System Verilog
Reference Model Language	System Verilog, C
Verification Platform	System Verilog Implemented, UVM-like
Test Patterns	Random Values, Extreme Values, Real-world Bitmaps

5.4 Verification Plan

The detail information of the verification plan of this IP is shown in Table 10.

Table 10: Verification Plan

Design Under Test	Test Pattern	Count of Patterns
Overall IP	Real-world Bitmap	60
	Random Bitmap	100
Multiplier Adder Unit	Extreme Values	1296
	Random Values	50000
8-input Add Unit	Extreme Values	104976
	Random Values	200000
Round Unit	Extreme Values	36
	Random Values	1000
Limit Unit	Extreme Values	3
	Random Values	1000
Overall Pipeline	Extreme Values	147456
	Random Values	200000

5.5 AXI-Stream Video Image VIP

The AXI-Stream Video Image VIP using the bitmap processing library which can read and write windows bitmap files (.BMP) into a bit array (virtural memory) by System Verilog for IP Verfication. The ”axi_stream_video_image_in_vip” IP can read a bitmap file into the memory, and send it by a AXI-Stream Video Interface (Defined in Xilinx User Guide UG934). And the ”axi_stream_video_image_out_vip” IP can monitor a AXI-Stream interface, obtain a frame which transmitting on the interface and save it to a bitmap file.

Please refer this GitHub Link (https://github.com/Aperture-Electronic/SystemVerilog-Bitmap-Library-AXI-Image-VIP) for detailed information of this important VIP in the verification platform.

The verifaction IP and the bitmap processing library were developed by the Nijigasaki IC Design Club.

5.6 Test Result

The detailed result report of the verification is shown in Table 11.

Table 11: Test Result

Design Under Test	**Test Pattern**	**Tested**	**Mismatched**	**Pass/Fail**
Overall IP	Real-world Bitmap	60	0	Pass
	Random Bitmap	100	0	Pass
Multiplier Adder Unit	Extreme Values	1296	0	Pass
	Random Values	50000	0	Pass
8-input Add Unit	Extreme Values	104976	0	Pass
	Random Values	200000	0	Pass
Round Unit	Extreme Values	36	0	Pass
	Random Values	1000	0	Pass
Limit Unit	Extreme Values	3	0	Pass
	Random Values	1000	0	Pass
Overall Pipeline	Extreme Values	147456	0	Pass
	Random Values	200000	0	Pass

6 Design with the Core

6.1 Operation

The Real-time Video Bicubic Super-resolution IP is a standalone co-processing pipeline and can work completely independently from any processor. If a processor is required to work with it for video frame synchronization, video frame access, etc., Video Direct Memory Access (VDMA) can be used to work with this IP.

6.2 Clock

The Real-time Video Bicubic Super-resolution IP operates on the clock input from the pin **clk**.

6.3 Clock Enable

The Real-time Video Bicubic Super-resolution IP has a clock enable pin **clken**, which is a active-High clock enable signal synchronous to **clk**. When the clock enable de-asserted, the IP will not continue to work, but all internal work in progress will be retained.

6.4 Reset

The Real-time Video Bicubic Super-resolution IP is fully reset when **aresetn** is asserted. This is an asynchronous active-Low reset.

There are also some synchronous reset used to reset some hardcore components (DSPs and BRAMs), they are **dsp_reset** for DSPs and **bram_reset** for BRAMs. These reset are active-High reset, and synchronous to **aclk**.

There is a synchronous reset siganl **sclr** in the Real-time Video Bicubic Super-resolution IP. This reset signal can reset entire IP to default state, an it is synchronous to **clk**. When the IP received a error frame (mismatched SOF signal, or mismatched resolution), this synchronous reset can be used to clear the error counters in the IP and resume the work.

7 Design Flow Steps

This section describes customizing and generating the IP core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core.

7.1 File Directory Structure

The directory structure of the source files for Real-time Video Bicubic Super-resolution IP is shown in Table 12.

Directory	Sub Directory	Description
doc		Documentations
ip		Packaged IP for Vivado Design Suite
modelsim		Modelsim Simulation Project
sim	pkg	Verification IPs and Packages
	ref	Reference Models
	scb	Scoreboard for UVM-like Testbenches
	tb	Testbenches
src		Design Sources
vivado		Vivado Synthesis Project

Table 12: Directory Structure

7.2 Source Files

The design source files in the **src** directory are shown in Table 13.

Table 13: Source Files

File	Type	Description
Top Level Design File		
bicubic.sv	System Verilog Source File	Top level design file of the IP
Design Unit Design File		
bicubic_line_buffer.sv	System Verilog Source File	Line buffer module
bicubic_pipeline_controller.sv	System Verilog Source File	Pipeline controller module
bicubic_pixel_out_realign.sv	System Verilog Source File	Pixel re-align module

Continued on next page

Table 13: Source Files (Continued)

bicubic_pipeline.sv	System Verilog Source File	Top level design file of the computation pipeline
Computation Pipeline Design File		
bicubic_coeff_rom.sv	System Verilog Source File	Bicubic coefficient LUT
bicubic_symmertial_mux.sv	System Verilog Source File	Symmertial multiplexer
bicubic_refpx_block_buffer.sv	System Verilog Source File	Super-block buffer
simd4x_round.sv	System Verilog Source File	4x Parallel rounding unit
pixel_limit_output.sv	System Verilog Source File	Pixel limit unit
DSP Computation Unit Design File		
dsp_simd2x_int9xuint8.sv	System Verilog Source File	2x parallel INT9xUINT8 multiplier uint (SMS architecture)
dsp_simd2x_int9xuint8_cascade_add.sv	System Verilog Source File	2x parallel SMS multiplier unit with cascade adder
dsp_simd4x_int12_add.sv	System Verilog Source File	4x parallel INT12 adder (for rounding)
dsp_add8_cin.sv	System Verilog Source File	8-input 4-carry INT18 adder
dsp_add_cascade_cin.sv	System Verilog Source File	Adder with cascade input and carry input
dsp_add3_cin.sv	System Verilog Source File	3-input adder with carry input
DSP Unit Parallel Instantiation File		
bicubic_nx_dsp_add.sv	System Verilog Source File	Nx parallel 8-input 4-carry INT18 adder instantiation
bicubic_nx_pixel_limit.sv	System Verilog Source File	Nx parallel pixel limit unit instantiation
bicubic_nx_simd_mul.sv	System Verilog Source File	Nx parallel SMS multiplier unit instantiation
bicubic_nx_simd_round.sv	System Verilog Source File	Nx parallel rounding unit instantiation
Global Marco Settings File		
bicubic_global_settings.sv	System Verilog Source File	Global macro settings
Hardware Resource Wrapper File		

Continued on next page

Table 13: Source Files (Continued)

sdpram_wrapper.sv	System Verilog Source File	Wrapper for simple dual-port RAM instantiation
Basic Unit Design File		
sfr.sv	System Verilog Source File	Customizable parameterized shift register
sfr_ce.sv	System Verilog Source File	Customizable parameterized shift register with clock enable
sfr_ce_sclr.sv	System Verilog Source File	Customizable parameterized shift register with clock enable and synchronous reset

7.3 Parameters

The Real-time Video Bicubic Super-resolution IP has customizable parameters that can be modified in the IP core top-level file. The parameters are shown in Table 14.

Parameter	**Format**	**Default Value**	**Range**	**Description**
INPUT_VIDEO_WIDTH	Integer	960	16 to 3840	The width of the input video frame
INPUT_VIDEO_HEIGHT	Integer	540	16 to 2160	The height of the input video frame

Table 14: Customizable Parameters

7.4 Macro Definitions

The Real-time Video Bicubic Super-resolution IP has customizable macro definitions in the global settings file (**bicubic_global_settings.sv**). These macros control the synthesis method used during synthesis to adapt to the different devices.

In this IP, all macro definitions are switch types. If you want to use a macro switch, please remove its comment in the file, otherwise, please comment the corresponding macro.

The macros are shown in Table 15.

Macro	Description
USE_DSP48E2_PRIMITIVE[1]	**On**: Using the DSP48E2 primitive in Xilinx UltraScale+ devices, which can force the synthesizer to use our optimized DSP48E2 wiring and architecture, maximizing performance and minimizing resource usage. **Off**: Disable the use of DSP48E2 primitive and use Verilog inference to complete the implementation of all computational statements, maximizing compatibility.
USE_LUT_FOR_ROUNDING	**On**: Use LUT instead of DSP unit for rounding operation to reduce DSP usage, minimizing resource usage. **Off**: Using DSP units for rounding operations, maximizing performance.
USE_XPM_MEMORY[2]	**On**: Use Xilinx Parameterized Macros (XPM) to implement memory elements (Line buffers), maximizing performance and minimizing resource usage. **Off**: Implement memory components using Verilog inference, maximizing compatibility.

Table 15: Macro Switches

Note For more information about Xilinx Parameterized Macros (XPM), please refer the following Xilinx documentations.

1. UltraScale Architecture Libraries Guide, UG974
2. Vivado Design Suite 7 Series FPGA and Zynq-7000 SoC Libraries Guide, UG953

7.5 Constraining the Core

Constraints are delivered when the IP is generated.

Required Constraints

This section is not applicable for this IP core.

Device, Package, and Speed Grade Selections

This section is not applicable for this IP core.

Clock Frequencies

This section is not applicable for this IP core.

[1] Only valid on Xilinx UltraScale+ devices.
[2] Only valid on Xilinx devices.

Clock Management

This section is not applicable for this IP core.

Clock Placement

This section is not applicable for this IP core.

Banking

This section is not applicable for this IP core.

Transceiver Placement

This section is not applicable for this IP core.

I/O Standard and Placement

This section is not applicable for this IP core.

8 Example Design

This chapter contains information about the example design of the Real-time Video Bicubic Super-resolution IP.

8.1 Overview

This example design for scaling live video from 540p to 4K resolution. The design is done based on the Xilinx Zynq UltraScale+ platform.

This design is capable of inputting 540p video in real time through the HDMI interface, scaling it up in real time, and finally outputting a 4K 30 fps video stream from the DP interface on the PS side.

The top-level module block diagram of this design is shown in Figure 24.

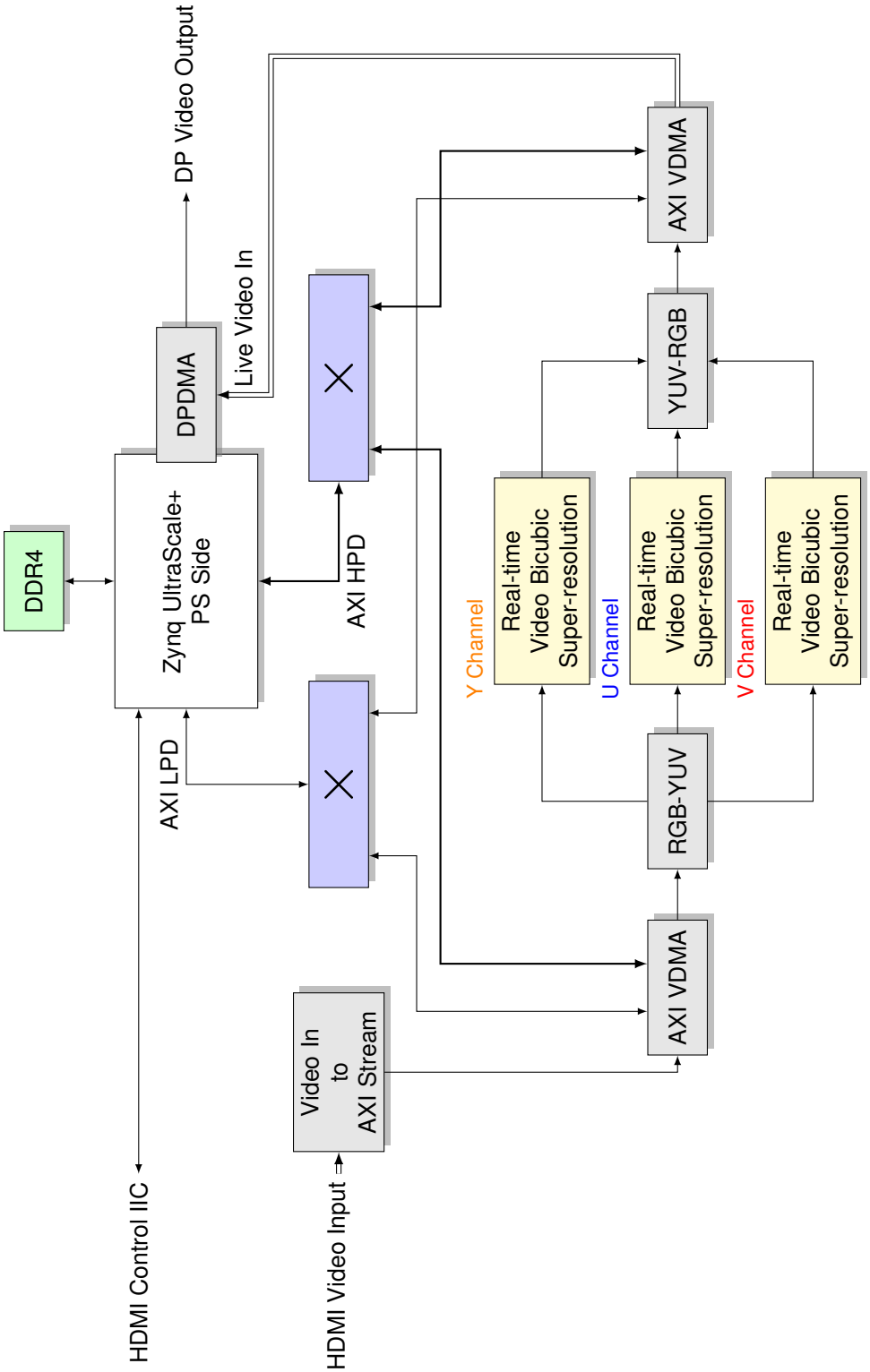


Figure 24: Top Level of the Example Design

8.2 IP Cores

Many third-party IP cores are used in the example design, and the description and functions of these IP cores are as follows.

AXI VDMA The Xilinx AXI Video Direct Memory Access provides high-bandwidth direct memory access between memory and AXI4-Stream video type target peripherals. The AXI VDMA supports a mechanism to synchronize writing and reading of frames in the frame buffer through Genlock synchronization.

RGB-YUV The Xilinx RGB to YCrCb Color-Space Converter is a simplified 3x3 matrix multiplier converting three input color samples to three output samples in a single clock cycle. The optimized structure uses only four XtremeDSP™ slices by taking advantage of the dependencies between coefficients in the conversion matrix of most RGB to YCrCb 4:4:4 or RGB to YUV 4:4:4 standards.

YUV-RGB The Xilinx YCrCb to RGB Color-Space Converter is a simplified 3x3 matrix multiplier converting three input color samples to three output samples in a single clock cycle. The optimized structure uses only four XtremeDSP™ slices by taking advantage of the dependencies between coefficients in the conversion matrix of most YCrCb 4:4:4 or YUV 4:4:4 to RGB standards.

Video In to AXI4-Stream The Xilinx Video In to AXI4-Stream is designed to interface from a video source (clocked parallel video data with synchronization signals - active video with either syncs, blanks or both) to the AXI4-Stream Video Protocol Interface.

8.3 Peripherals of the CPU

The example design uses several peripherals on the PS side of the Zynq UltraScale+, and the functions of these peripherals are described below.

DisplayPort Controller

The DisplayPort controller implements a flexible display and audio pipeline architecture.

The DisplayPort controller can source data from memory (non-live input) or the (live input) programmable logic (PL). The DisplayPort processes data, and sends it out through the DisplayPort source-only controller block to external display devices or to the PL (live output). The DisplayPort pipeline consists of the DisplayPort direct memory access (DMA) for fetching data from memory, a centralized buffer manager, a display rendering block, an audio mixer block, and the DisplayPort source controller, along with the PS-GTR block. The DisplayPort pipeline supports an ultra-high definition (UHD) aggregate video bandwidth of 30 Hz.

The DisplayPort DMA controller (DPDMA) supports up to six input channels as non-live input. Video/graphics, and audio streams can be sourced from the PL as live streams. The video processing stage involves mixing video and graphics streams, color space conversion, and chroma sub-sampling. The audio processing stage involves mixing two audio streams and volume control. The output of the audio/video processing pipeline can be output to the DisplayPort source controller or optionally be routed to the PL as live output.

I2C Controllers

The I2C controllers can function as a master or a slave in a multi-master design. They can operate over a clock frequency range up to 400 kb/s.

8.4 Buses of the CPU

The example design uses the bus connections (AXI LPD and AXI FPD) between the Zynq UltraScale+ PS and PL, and their functions are described below.

AXI LPD (AXI HPM0 LPD)

The low-latency interface port (AXI HPM0 LPD) from the high-performance interface port (LPD) to the PL includes the following features.

- Configurable to 32, 64, or 128-bit data widths on the PL side.
- AXI4 access in the PL, but is limited to a burst length of 16.
- Responds to lowest 512 MB memory in LPD's 32-bit address space.
- Enables direct access to the PL (for example for block RAM, DDR) for the safety use cases.

AXI HPD

Six high-performance interfaces provide the PL bus masters access to all PS slaves. However, these are designed to provide high-bandwidth datapaths to the DDR memory

The PL-PS interfaces are designed to provide a high-throughput datapath between the PL masters and PS memories, including the DDR and OCM memories. The main features of these interfaces are list as follows.

- Support for AXI4. The conversion to AXI3 takes place in the PS.
- 32, 64, or 128-bit data-wide master interfaces that are independently programmed for read and write per port.
- Efficient dynamic upsizing for all full-width AXI INCR commands.
- Asynchronous clock frequency domain crossing for all AXI interfaces between the PL and PS. Two PL clocks per interface, one for read and one for write.

8.5 Workflow of the Design

The Zynq UltraScale+ CPU in the example design is mainly used to configure the parameters of each peripheral. At system startup, the CPU runs a program to initialize each peripheral. The flow is as follows.

1. Initialize each bus interface.
2. Initialize the I2C peripheral and configure the registers of the external HDMI-to-video signal converter chip via I2C.
3. Initialize the DP controller and DPDMA and configure it to Live Video mode to receive and display live video from the video processing pipeline.

4. Configure DP link monitoring interrupt to enable DP cable hot-plugging.

5. Configure input and output VDMA and enable multi-frame buffering and vertical synchronization.

When the CPU completes the system configuration, the system enters normal operation.

During normal system operation, the HDMI signal is converted to a standard video signal through the video converter chip and input to the FPGA. Video In to AXI-Stream IP converts the input video signal to an AXI4-Stream video stream and input to Input VDMA.

The Input VDMA saves the input video data to the DDR on the PS side via the AXI FPD bus, and sends frames to the video processing pipeline with frame synchronization.

The video processing pipeline consists of RGB-YUV conversion IP, YUV-RGB conversion IP, and three Real-time Video Bicubic Super-resolution IPs. Each Real-time Video Bicubic Super-resolution IP processes one of the Y/U/V channel independently but in parallel and synchronized.

After the Real-time Video Bicubic Super-resolution IP super-resolution processing and YUV-RGB conversion, the video stream is input to Output VDMA.

Output VDMA can save the processed video to the DDR on the PS side via AXI FPD bus, and at the same time send the video to the Live Video interface of DP controller under frame synchronization. DPDMA and DP controller sends the high-resolution video to DP port for display.

9 Licensing and Open Soruce Information

The Real-time Video Bicubic Super-resolution IP is a fully open source IP core, with all source code and detailed design materials publicly available.

The Real-time Video Bicubic Super-resolution IP is licensed under the GNU Lesser General Public License (LGPL).

REVISION HISTORY

The Table 16 shows the revision history for this document.

Date	Version	Description of Revisions
5/10/2022	1.0	Initial Nijigasaki IC Design Club release.
7/20/2022	2.0	Merge ”Design Verification” section into the documemt.

Table 16: Revision History

NOTICE OF DISCLAIMER

The information disclosed to you hereunder (the “Materials”) is provided solely for the selection and use of Nijigasaki IC Design Club products. Tothe maximum extent permitted by applicable law: (1) Materials are made available “AS IS” and with all faults, Nijigasaki IC Design Club hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOTLIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULARPURPOSE; and (2) Nijigasaki IC Design Club shall not be liable (whether in contract or tort, including negligence, or under any other theory ofliability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (includingyour use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including lossof data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if suchdamage or loss was reasonably foreseeable or Nijigasaki IC Design Club had been advised of the possibility of the same. Nijigasaki IC Design Club assumes noobligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to productspecifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. IP cores may be subject to warranty and support terms contained in a license issued toyou by Nijigasaki IC Design Club. Nijigasaki IC Design Club products are not designed or intended to be fail-safe or for use in any plication requiring fail-safeperformance; you assume sole risk and liability for use of Nijigasaki IC Design Club products in Critical.

附录 B

Dashboard

dashboard | hierarchy | modlist | groups | tests | asserts

Date: Wed Jul 27 14:36:50 2022
User: huang
Version: O-2018.09-SP2
Command line: urg -full64 -dir ./TextureClassifierTop.vdb/
Number of tests: 1

Total Coverage Summary

SCORE	LINE	COND	TOGGLE	FSM
67.50	51.76	76.48	74.24	

Hierarchical coverage data for top-level instances

SCORE	LINE	COND	TOGGLE	FSM	NAME
67.54	51.58	76.48	74.56		TextureClassifierTop
100.00	100.00				__simulation_def
57.50	100.00		15.00		glbl

Total Module Definition Coverage Summary

SCORE	LINE	COND	TOGGLE	FSM
74.01	60.23	82.01	79.78	

0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%

附录 C

Dashboard

dashboard | hierarchy | modlist | groups | tests | asserts

Date: Wed Jul 27 14:48:15 2022
User: huang
Version: O-2018.09-SP2
Command line: urg -full64 -dir ./AdaptedSharpener.vdb/
Number of tests: 1

Total Coverage Summary

SCORE	LINE	COND	TOGGLE	FSM
51.68	46.70	55.98	52.35	

Hierarchical coverage data for top-level instances

SCORE	LINE	COND	TOGGLE	FSM	NAME
51.62	46.54	55.98	52.36		AdaptedSharpener
100.00	100.00				__simulation_def
57.50	100.00		15.00		glbl

Total Module Definition Coverage Summary

SCORE	LINE	COND	TOGGLE	FSM
63.33	50.85	66.27	72.88	

0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%

附录 D

Dashboard

dashboard | hierarchy | modlist | groups | tests | asserts

Date: Tue Jul 26 22:07:48 2022
User: huang
Version: O-2018.09-SP2
Command line: urg -full64 -dir ./YUVConvertUnit.vdb/
Number of tests: 1

Total Coverage Summary

SCORE	LINE	COND	TOGGLE
61.25	96.00		26.50

Hierarchical coverage data for top-level instances

SCORE	LINE	COND	TOGGLE	NAME
60.41	94.29		26.53	YUVConvertUnit
57.50	100.00		15.00	glbl

Total Module Definition Coverage Summary

SCORE	LINE	COND	TOGGLE
65.79	96.00		35.57

0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%

附录 E

Dashboard

dashboard | hierarchy | modlist | groups | tests | asserts

Date: Tue Jul 26 22:53:14 2022
User: huang
Version: O-2018.09-SP2
Command line: urg -full64 -dir ./RGBConvertUnit.vdb/
Number of tests: 1

Total Coverage Summary

SCORE	LINE
100.00	100.00

Hierarchical coverage data for top-level instances

SCORE	LINE	NAME
100.00	100.00	RGBConvertUnit
100.00	100.00	glbl

Total Module Definition Coverage Summary

SCORE	LINE	0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%
100.00	100.00	