2023 全国大学生

嵌入式芯片与系统设计竞赛

——FPGA 创新设计竞赛

作品名称：VisionZoom Pro

作者姓名：廖月伟，胡嘉翔，邓心阳

VisionZoom Pro

姓名：廖月伟, 胡嘉翔, 邓心阳

第一部分　设计概述

1.1 设计目的

视频与流媒体作为最直观的信息传输形式，一直是信息技术发展的前沿，随着 VR 等技术的发展，视频类文件如何高效高清的传递成为了亟待解决的问题。本小组通过全面研究 [1, 2]，我们开发高清视频-流媒体传输器 VisionZoom Pro，并应用于任意比例缩放与视野平移的任务。同时，为了使用户获得更佳的体验，我们采用了中值滤波技术与边缘检测技术来强化视频的舒适性与稳定性，并充分使用了 RISC-V 核与 UART 传输进行用户的反馈。基于易灵思提供的平台，我们的 VisionZoom Pro 及配套的 GUI 界面，提供高性能优体验的技术实现。

1.2 应用领域

我们的产品 VisionZoom Pro 在各领域上的应用广泛多样。首先，在传统的会议中 VisionZoom Pro 提供优质的视频资源做到低延迟，零模糊，即插即用，确保不同网络环境下的参会者都能获得良好的视频体验。其次，在医疗影像处理方面，VisionZoom Pro 提供了新方法，将 X 射线、CT 扫描和 fMRI 等医学图像以视频形式助力医生分析。此外，在军事方面，通过视频的实时处理和缩放，提升了监视和侦察的效率。最后，VisionZoom Pro 在虚拟现实（VR）和元宇宙领域同样具有巨大潜力，通过实时缩放技术，用户能够调整虚拟显示体验中的图像大小，提高内容的清晰度和逼真感，无论是在 PC 端还是头戴式设备上，都能为用户提供更佳的视觉体验。

1.3 主要技术特点

本小组在 VisionZoom Pro 的开发过程中，运用了各式各样的技术，其中最主要的创新点在信息传递与信息处理模块上。在信息的传递上，我们采用 HDMI 信号作为编码形式在 FPGA 中传入传出，这是最大量的最广泛被用户使用的接口方式。通过钛金系列 DDR3 IP 的使用，达到 AXI4 全速模式，大幅度提高数据传输速率，高达 1000Mbps;通过完全参数化的设计，可与符合 JESD79-3 标准的 DDR3 SDRAM 兼容，具有高可移植性。而算法控制信号的传输则是基于 RISC-V 核与 UART 协议进行的，并将当前设备状态及时在屏幕上反馈给用户，营造良好的交互体验。而在算法部分，我们的主要技术点分为包含最近邻插值与双线性插值两种算法的像素处理模块，以及以中值滤波与边缘检测算法为主体的图像处理模块。像素处理模块中的两种算法能够通过 UART 传输的数据进行合理的切换，充分利用 FPGA 的并行性，实现高效的缩放处理。同时，中值滤波与边缘检测算法能够降低环境噪音在数据传输过程中的损失同时也有效的缓解了像素处理模块造成的边缘锯齿与边缘模糊的问题。

1.4 关键性能指标

图像质量评价：图像作为系统的输入和输出两个关键部分，图像质量是对于系统的评判最重要的指标。为了对图像质量进行全面的评定，我们将整体的评价标准分为主观和客观两部分。客观评价标准主要聚焦于一些可感的参数与图像整体呈现效果，我们选取了峰值信号比（PSNR）作为参数进行量化评价 [1,2]；同时，通过是否产生锯齿，物体边缘是否模糊，以及是否出现闪屏。而主观的参数，我们主要进行面向用户的实验，评价标准基于相同画面的观看舒适度，眼睛是否感觉疲劳，色彩是否有衰减，以及是否存在边缘模糊，综合给出感受分，最终与客观评分一起构成我们对图像质量的评价。经测量，我们的双线性插值与最近邻算法 PSNR 分别达到了 22.3 与 22.9。

处理速度与延迟：对于视频流媒体的评价，另一个极为重要指标的是画面的流畅度。经过 VisionZoom Pro 系统的延迟是肉眼无法察觉的，而测量出的帧率均为理想情况下的 60Hz。

资源利用率：充分利用 FPGA 硬件加速的性能和资源，避免无效占用资源与过度资源空置。

1.5 主要创新点

(1) 在经典算法最近邻插值和双线性插值算法的基础上增加两个数字滤波器，减小噪声等干扰，提高系统鲁棒性。

(2) RAM FIFO 使用：为计算插值算法所需要的系数和存储插值点周围的四个像素值，采取方法为使用两个行缓冲，但是大量数据缓存会导致延时增加；采用乒乓轮换机制则又增加了硬件需求。因此我们采用以 RAM 为元素的 FIFO 缓冲区存储数据。

(3) 任意倍数缩放，既可以将任意大小扩大至全屏，也可将全屏缩小至任意分辨率，均通过为用户提供的 GUI 界面完成。

(4) 用串口指令实现对图像某一部分进行处理，实现对图像局部放大可适应多场景需求

(5) 双线性和最近邻在同一个模块内通过多路调制器进行选择，用户可通过 GUI 界面选择使用的算法。

第二部分　系统组成及功能说明

2.1 整体介绍

我们的视频流媒体缩放器的系统整体框图如下所示：

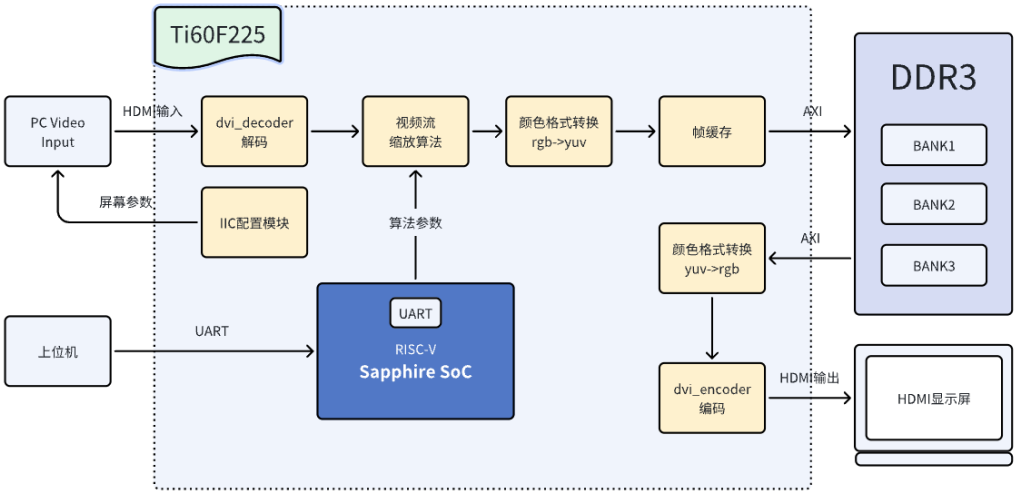

Figure 1. 系统整体框图

按照设计目的，我们的视频流媒体缩放系统需要做到低延迟，高画质和任意比例缩放效果三个基本需求。对于一个完整帧传输过程，首先是通过 IIC 配置模块，由 PC 端作为输入源向 FPGA 发送的初始图像分辨率信息为 1920*1080，60Hz。在建立通信后，通过 HDMI 输入输出将像素信息分为三个 10 位的颜色通道，这样的输入输出要求则要求我们在输入与输出之间增加十位颜色通道与三个八位 RGB 通道的编解码转换模块。通过译码得到的八位 RGB 数据输入包含预处理与插值的算法模块补全图像完成缩放功能的实现，并最终通过 HDMI 将本帧图像显示在显示器上。本系统的数据暂存部分需要缓存三帧缩放后的图像数据，而我们采用的 DDR3 模块选择了 16 位输入，因此在输入缓存模块之前需要将三通道共 24 位的颜色数据转换成 16 位的 yCbCr 格式并进行 yuv444 到 yuv422 的转换用以减小在保留良好视频像素点质量的情况下减小存储需要的空间。

在实现了缩放的基础工程的基础上，我们使用易灵思 RISC-V 软核 Sapphire SoC 来接取来自电脑上位机传来的用户控制信息。我们将 IP Sapphire SoC 软核的主频配置为 50MHz，并使能其 UART 以及对应的中断功能，而其他部件保持默认。更具体的使用过程中，由上位机向 Ti60F225 发送一串数据，RISC-V 软核接收到数据后解码出相应的视频参数，并将得到的参数写入 DDR 的相应位置（0x200）。每当帧同步信号传来，算法部分将读取存入 DDR 的视频参数，相应的调整算法的输入输出。将 UART 解码得到的参数，转换为具体算法与模式的控制信号，指导系统的视频流媒体输出。

2.2 各模块介绍

2.2.1 hdmi_rx/ hdmi_tx module

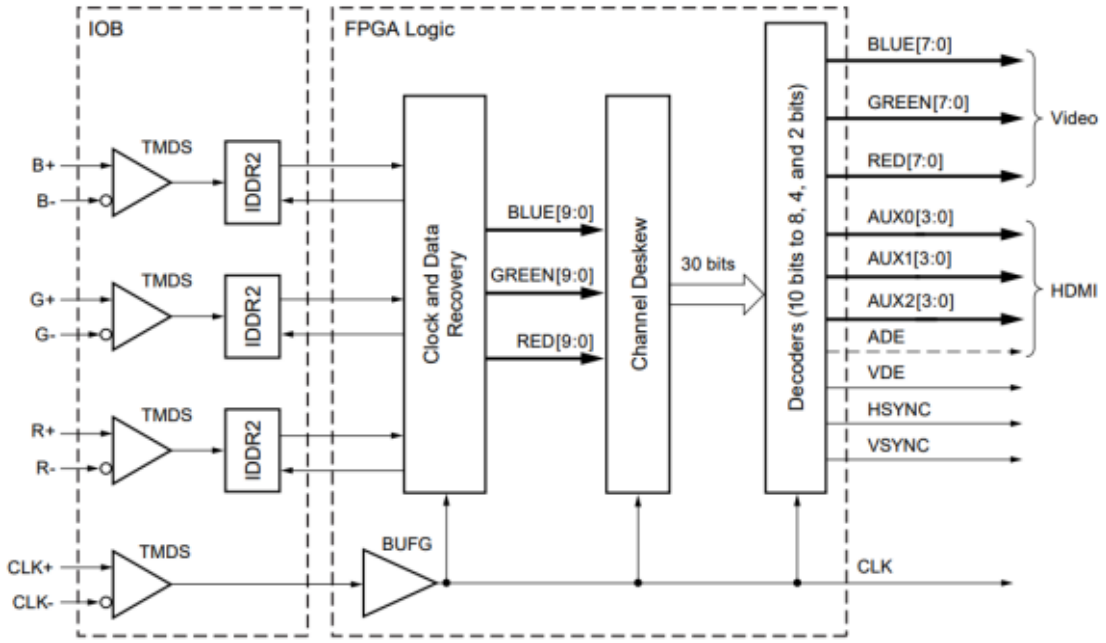

Figure 2. DVI 解码器示意图

信号名称	工程中含义	信号名称	工程中含义
hs	行同步	de	使能信号输入下一级模块
vs	场同步	data	像素点数据

上电瞬间，电脑通过 HDMI 的 DDC 通道以 IIC 协议读取 Ti60F225 ROM 中存储的 EDID 数据架构，从而获得 Ti60F225 的接收能力和接收特性：如屏幕分辨率、帧率、颜色设置等。电脑以对应分辨率的 hmdi 时序将视频流发送给 Ti60F225。hdmi_rx 模块接收到视频流数据后，DVI 解码器在一系列的串行数据中确定数据边界的位置。当在视频数据的数据周期和数据岛周期处理 TMDS 字时，则包含 5 次或更少的转换，当在控制周期处理 TMDS 字时，则包含 7 次或更多的转换。这些字符跳转次数的不同将会被用于同步边界。当边界被确定后，将串并转换器转换后的并行数据进行 8B/10B 的解码，得到像素数据和相关控制信号。最后将产生 hs、vs、de、data 四个信号传递到算法模块。

2.2.2 algorithm module

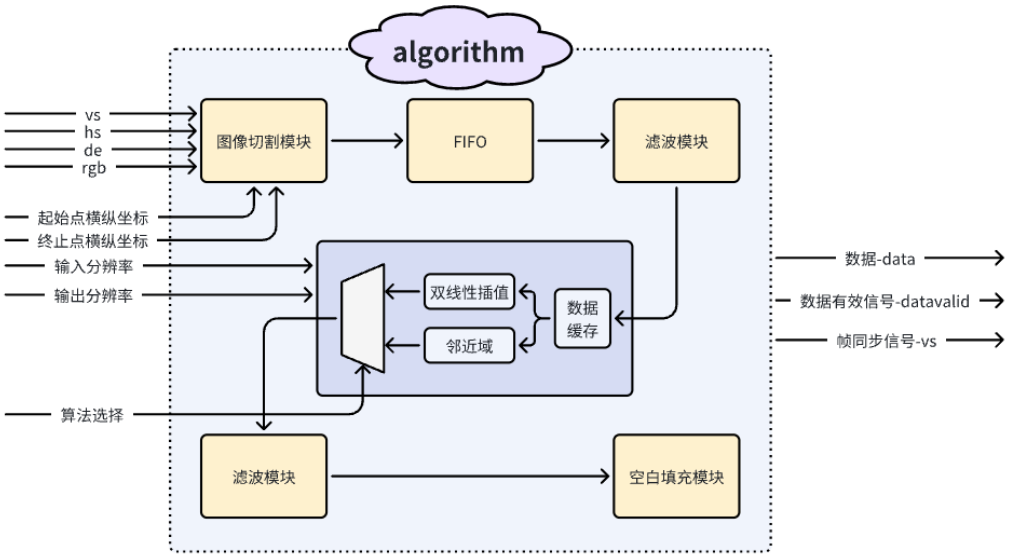

Figure 3.算法部分模块框图

帧图像缩放算法部分，首先通过串口数据得到放大、缩小的模式选择。若是放大模式，则从输入源视频图像中截取用户指定的部分（通过起始点和终止点横纵坐标确定切割的范围），通过原始帧切割模块执行实现；若是缩小模式，则将输入源视频图像缩小到用户指定的输出分辨率大小。考虑到算法中需要读取两行数据并写一行数据，我们使用异步读写 FIFO 采用不同时钟频率完成。通过用户选择，算法执行模块具体选择最近邻或双线性插值实现具体功能并在算法实现前后图像数据输入输出端均采用滤波模块减少噪声信号干扰 [3]。在实现缩小算法时，缩小后的输入图像将被展示在 HDMI 显示屏左上角，因此采用空白补全模块将屏幕空余部分显示为黑色防止闪屏现象的出现。通过使用加入滤波模块和不加入滤波模块的工程使用彩条 COLOR__ B A R进行仿真对比，缺少滤波模块时，查看仿真结果中像素点数据值，发现在每一行的数据中都会存在部分真实数据数据点数据不一致的情况如非理想出现的 0 值；上板验证是否加入滤波模块是否具有显著实际效果，得出结论若缺少滤波模块会在输出视频流图像中出现数十条黑色虚线，且相邻虚线中还有很多不规则排列的黑点或绿色噪点。

2.2.3 frame_buffer module

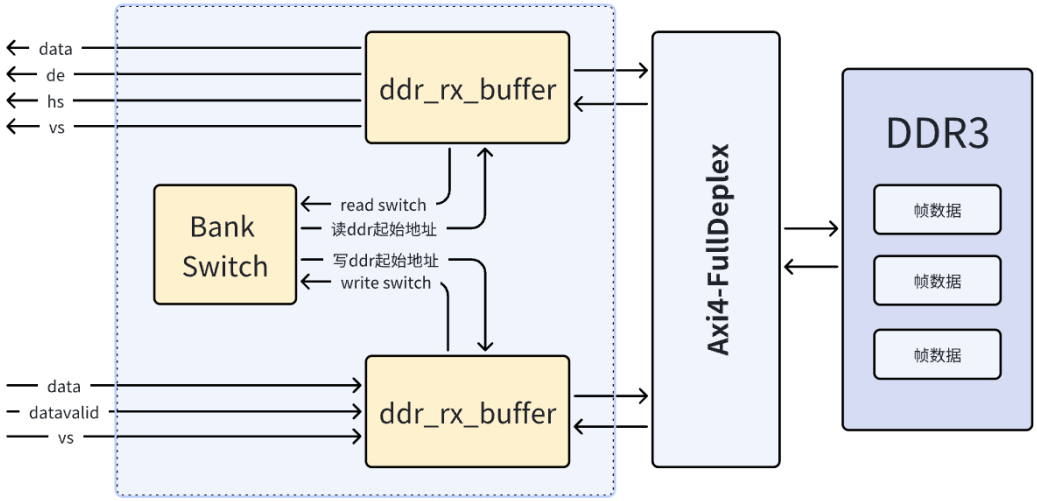

Figure 4. 帧缓存模块框图

DDR3 写时序中， 主机发送地址和控制信息到写地址通道中，然后主机发送每一个写数据到写数据通道中。当主机发送最后一个数据时，"LAST"信号拉盖。设备接受完所有数据后会有写相应指令信号发送回之际表明写事务完成。DDR3 读时序中，当地址出现在数据总线后，传输的数据将出现在读数据通道上，设备一直保持设备数据无效知道读取数据有效。处理器可能需要访问 DDR3 存储器来读取或写入数据，而这种访问需要通过一种高效的总线协议进行——AXI（Advanced eXtensible Interface）。AXI 是一种高性能、高带宽的总线协议，用于在数字系统中连接和通信各种硬件组件 [4]。我们调用易灵思提供的钛金系列 DDR3 IP 核，可以通过软核实现 AXI4 全速模式并实现自动校准（在工程中我们通过配置可以从 LED 是否闪烁看出是否自校验成功，便于查验）。

2.2.4 上位机发送模块——客户友好型的架构设计

为了便于用户使用我们的产品，我们做了精美易使用的人机交互界面，用户选择正确的串口后即可打开串口发送功能并通过下拉菜单栏选择合适的传输波特率与放大或缩小模式；通过点击对应算法前面的空心圆点，如下图所示：

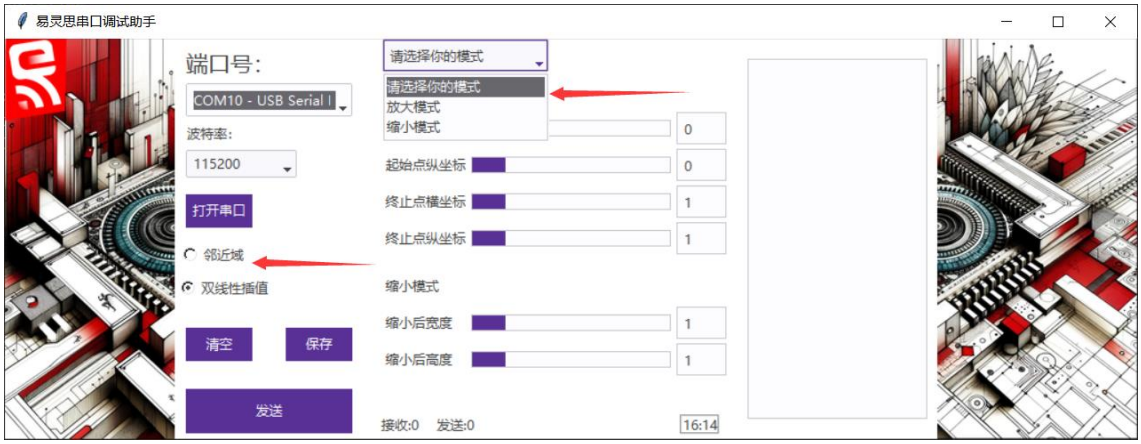

Figure 5. 通过单选或下拉菜单栏选择

选择放大或缩小模式之一后，另一个功能相关参数选择会变为深灰色无法输入或拖动滑条；针对已选择的模式则可以任意拖动滑条选择像素点坐标或缩小后的分辨率值，如 Fig. 所示，显示的视频流将会从 1920*1080 分辨率情况切割出左上角四分之一图像放大至 1920*1080；Fig. 功能为将 1920*1080 缩小到 800*600，且显示屏空余部分用黑色像素点填充。

Figure 6. 切割并放大左上角 1/4 区域

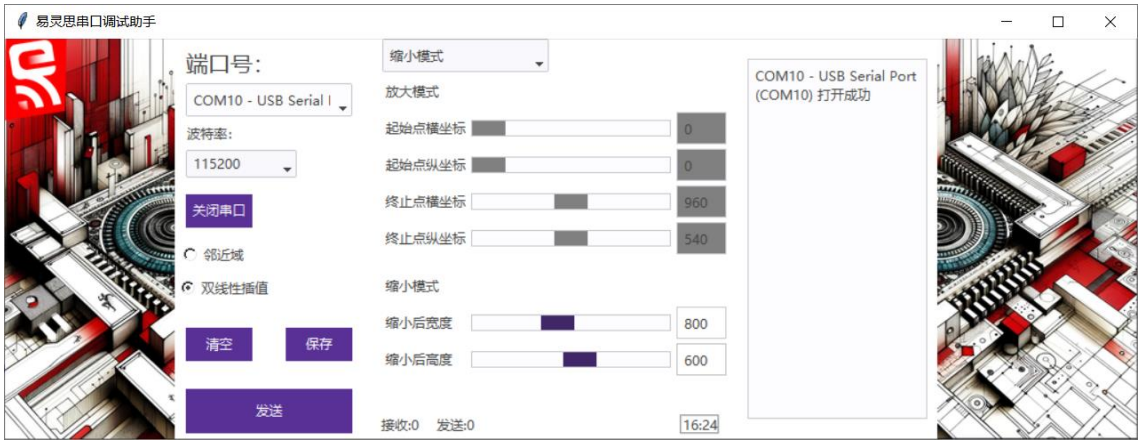

Figure 7. 将 1920*1080 缩小到 800*600

第三部分　完成情况及性能参数

3.1 仿真波形

放大：这里设置为切割出视频流每帧的中间 1/2 部分，并把切出的图像放大为原来两倍

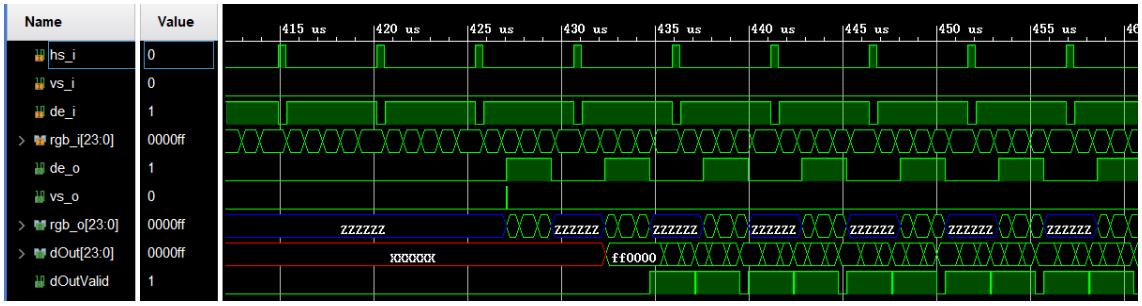

Figure 8. 双线性插值算法仿真结果图

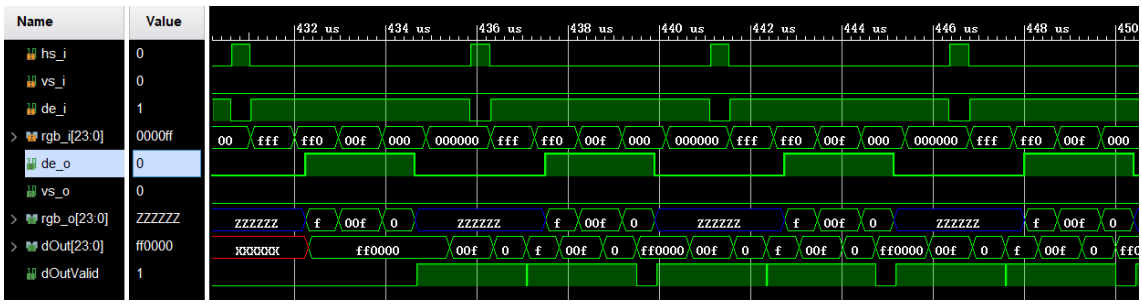

Figure 9. 最近邻算法仿真结果图

信号名称	工程中含义	信号名称	工程中含义
de_o	输出使能信号	hs_i	输入行同步
vs_o	输出场同步	vs_i	输入场同步
rgb_o	输出像素点数据	de_i	输入使能信号
		rgb_i	输入像素点数据

在算法模块前例化一个彩条视频流模块（480*272）以模仿电脑传与算法的视频流。首先，视频流经过图像切割模块，将 pc 传来的原始视频流的每帧数据切出用户想要放大的部分。从仿真结果看出图像切割模块（de_o、vs_o、rgb_o）正确地从原图像（hs_i、vs_i、de_i、rgb_i）中切出用户需要放大的部分。

缩放算法模块中包含 3 个双端口 RAM 来存取寄存的视频像素数据。从仿真图像看出，当算法模块正确寄存了两行图像数据后，该模块将计算应该输出的图像数据。因这里简单设置算法将输入图像扩大为两倍，那么从起始读取两行数据后，算法每存取一行图像数据，则输出两行像素。（这里算法使用了 2 倍时钟）

展示任意比例放大仿真结果：159*146 -> 480*272

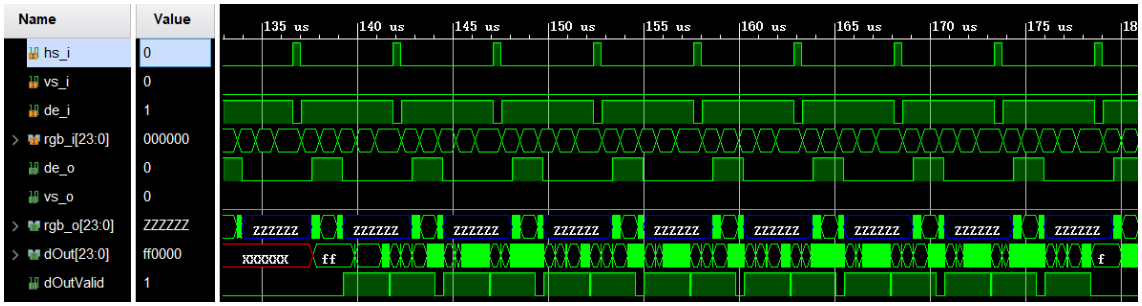

Figure 10. 双线性插值算法

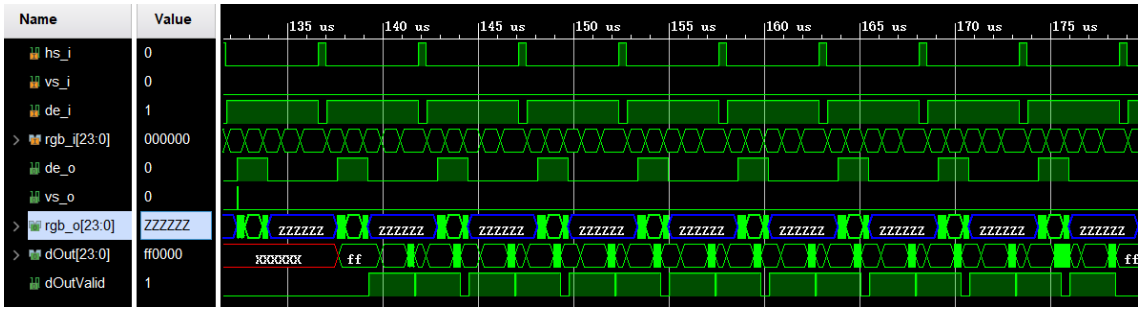

Figure 11. 最近邻算法

展示任意比例缩小： 480*272 -> 253*123

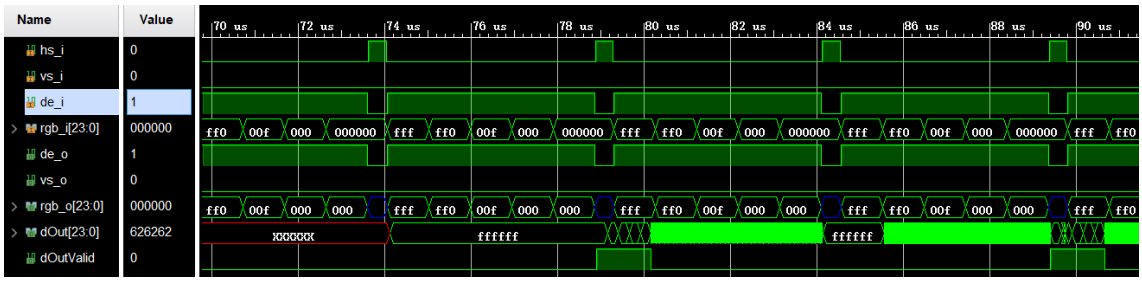

Figure 12. 双线性插值算法

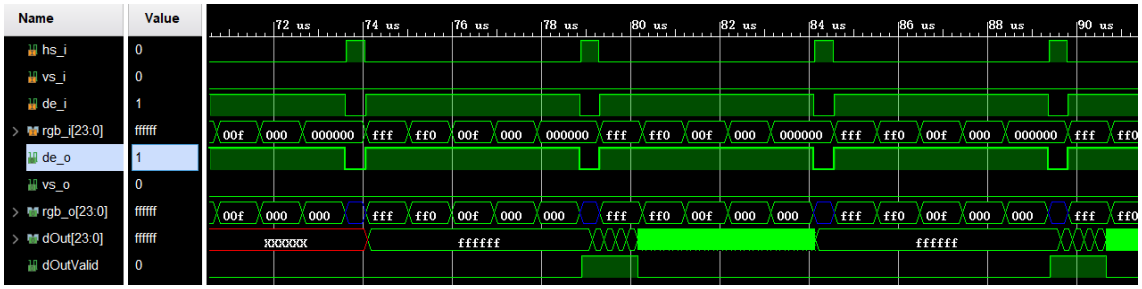

Figure 13. 最近邻算法

仿真结果看出算法模块支持 2 倍以下任意比例放大，以及任意比例缩小。

3.2 图像仿真效果

我们的算法支持大范围分辨率从图像缩放，考虑到文档报告篇幅问题，在这里使用小分辨率的图像缩放结果作为结果验证。

300*200 图像放大至 800*600：

Figure 14. 300*200 待缩放图像

Figure 15. 800*600 放大图像

将分辨率为 480*272 图像缩小为 300*150：

Figure 16. 480*272 待缩放图像

Figure 17. 300*150 缩小后图像

从使用图片放大和缩小得出结果，算法效果理想，图像颜色通道正确、边缘清晰。

3.3 实际波形展示

放大中心区域至 1920*1080 分辨率信号抓起部分结果如 Figure 18 所示：

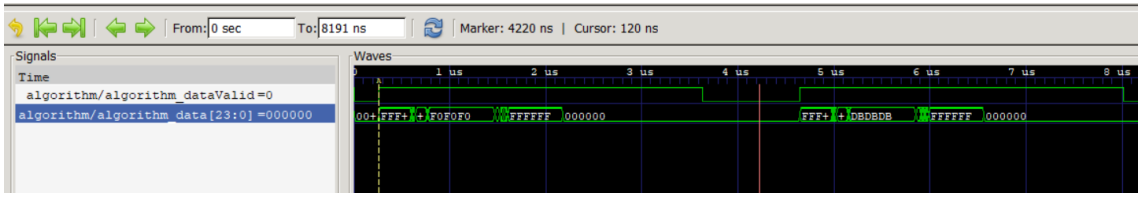

Figure 18. 实际拉取算法输出的 data 以及 data_valid 波形

3.4 平台搭建

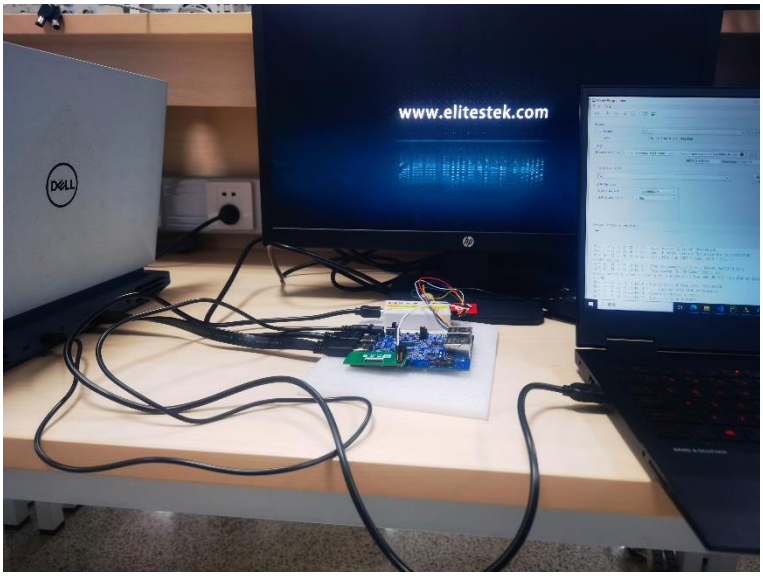

Figure 19. 实际展示平台搭建

我们本次产品以 Efinity EDA 工具为开发环境，在 FPGA 芯片 Ti60F225 上进行验证与演示。整体平台如 Figure 19 所示，包括了 FPGA 开发板、PC 输入源、显示器、JTAG 下载器、HDMI 传输线以及电源线等。FPGA 通过工程算法将输入源图像经过滤波、插值等处理后输出至显示器。缩小模式，我们选择从 1920*1080 缩小至 1800*950 和 960*950；放大模式，我们首先选择屏幕正中央区域即起始点坐标（280，1440）、终止点坐标（270，810）的一块 960*540 区域放大至显示器全屏 1920*1080。为体现我们可以任意比例缩放，我们随机选取一块起始点为远点的一块 1272*0737 矩形区域放大至全屏。如 Figure. 20, Figure. 21, Figure. 22, Figure 23.所示，效果理想，图像边缘清晰无锯齿不模糊，几乎无延迟不卡顿，颜色通道显示正常。

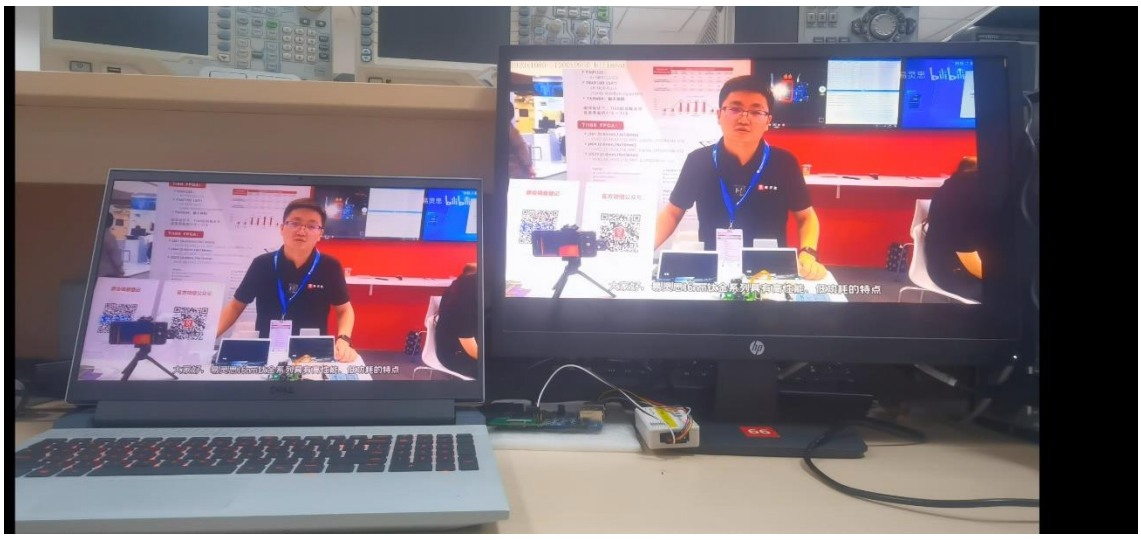

Figure 20. 1920*1080 -> 1800*950

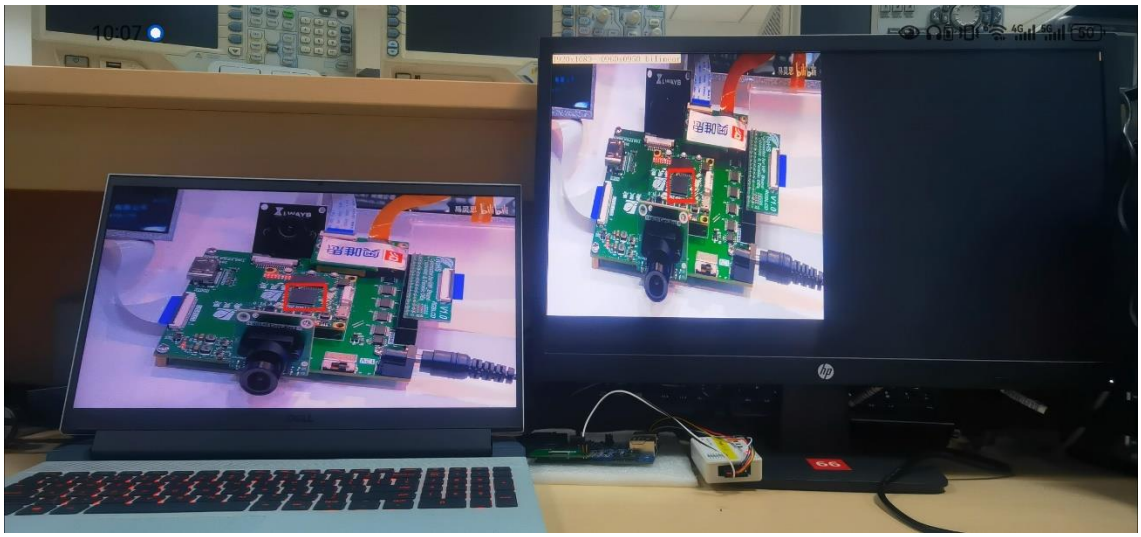

Figure 21. 1920*1080->960*950

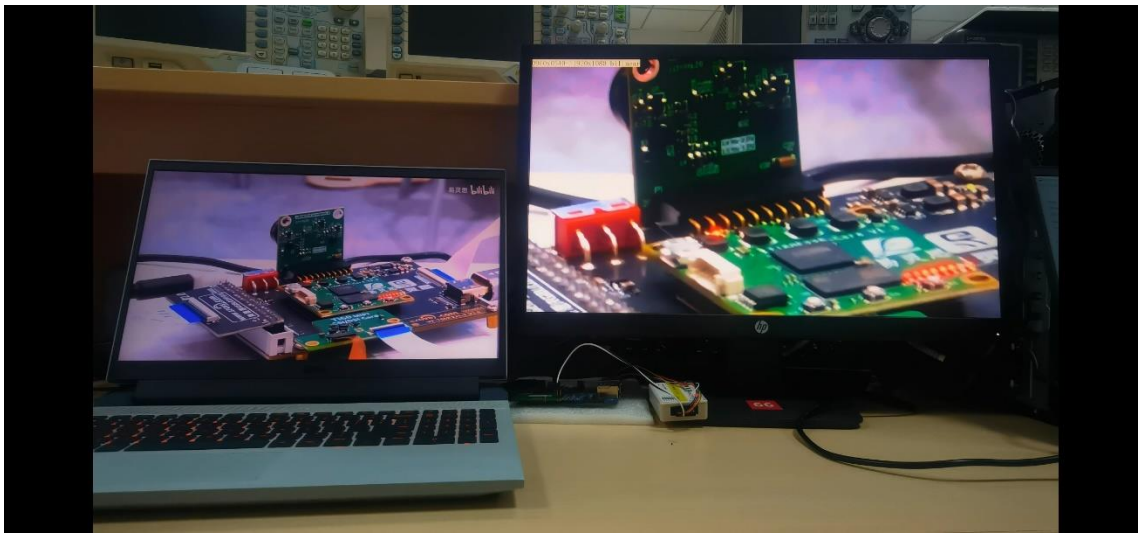

Figure 22. 图像中央区域 960*540->1920*1080

Figure 23. 1272*0737->1920*1080

3.5 资源利用

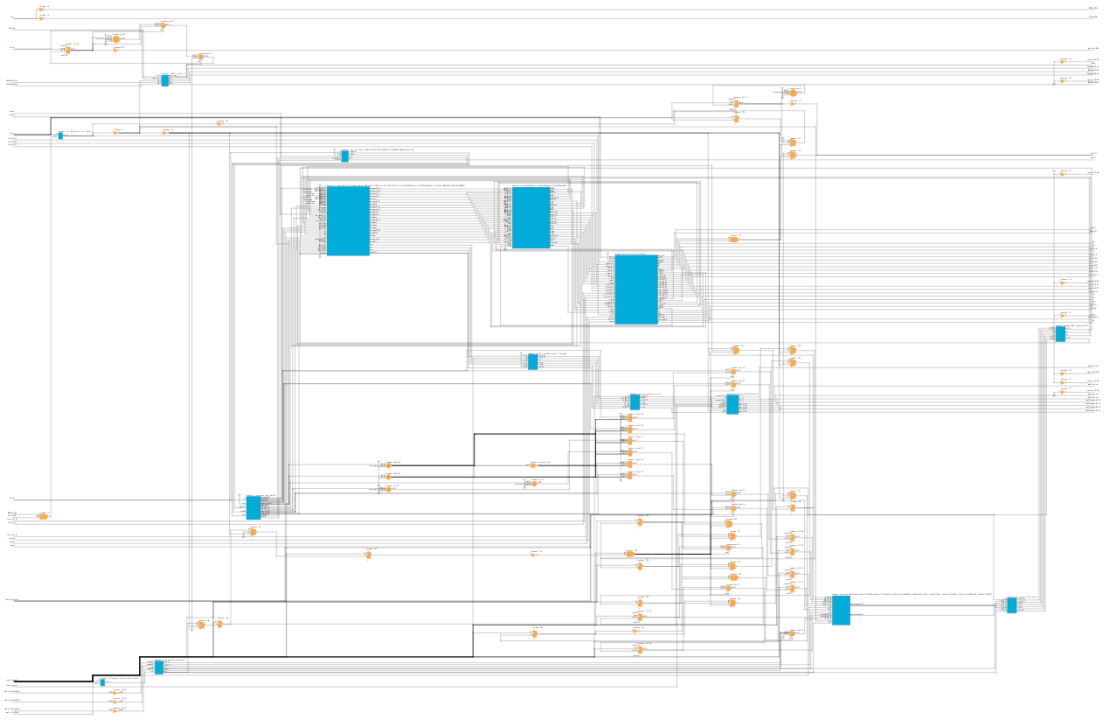

Figure 24. 系统整体 RTL 图

```
### ### Clock Load Distribution Report (begin) ### ###

                Clock     Flip-Flops     Shift-Regs   Memory Ports    Multipliers
                -----     ----------     ----------   ------------    -----------
              osc_clk            119              0              2              0
              sys_clk           1451            346             53              0
    hdmi_rx_slow_clk           1629             15             18              0
    hdmi_tx_fast_clk             57              0              0              0
             tdqss_clk             88              0              3              0
              core_clk            713              0             12              0
               twd_clk            383            144              8              0
               tac_clk            252              2              7              0
 hdmi_rx_slow_clk_2x            861             15             29              0
              clk_10m            372              0              0              0
```

Figure 25. 不同时钟占用资源报告

```
### ### Resource Summary (begin) ### ### ###
INPUT  PORTS     :   105
OUTPUT PORTS     :   208

EFX_ADD          :   2326
EFX_LUT4         :   9740
    1-2  Inputs  :   2543
    3    Inputs  :   3197
    4    Inputs  :   4000
EFX_DSP48        :   23
EFX_DSP24        :   9
EFX_FF           :   5925
EFX_SRL8         :   522
EFX_RAM10        :   68
EFX_DPRAM10      :   15
### ### Resource Summary (end) ### ### ###
```

Figure 26. RAM、寄存器资源的使用报告

```
---------- Resource Summary (begin) ----------
Inputs: 94 / 1703 (5.52%)
Outputs: 330 / 2267 (14.56%)
Global Clocks (GBUF): 20 / 32 (62.50%)
Regional Clocks (RBUF): 1 / 32 (3.12%)
    RBUF: Core: 0 / 16 (0.00%)
    RBUF: Periphery: 1 / 8 (12.50%)
    RBUF: Multi-Region: 0 / 8 (0.00%)
XLRs: 15508 / 60800 (25.51%)
    XLRs needed for Logic: 7438 / 60800 (12.23%)
    XLRs needed for Logic + FF: 2304 / 60800 (3.79%)
    XLRs needed for Adder: 1623 / 60800 (2.67%)
    XLRs needed for Adder + FF: 703 / 60800 (1.16%)
    XLRs needed for FF: 2918 / 60800 (4.80%)
    XLRs needed for SRL8: 522 / 14720 (3.55%)
    XLRs needed for SRL8+FF: 0 / 14720 (0.00%)
    XLRs needed for Routing: 0 / 60800 (0.00%)
Memory Blocks: 83 / 256 (32.42%)
DSP Blocks: 29 / 160 (18.12%)
---------- Resource Summary (end) ----------
```

Figure 27. 布局时 Memory 资源和 IO 管脚资源的使用情况

```
---------- 1. Periphery Usage Summary (begin) ----------
clkmux: 4 / 4 (100.0%)
control: 0 / 1 (0.0%)
gpio: 9 / 23 (39.13%)
hsio: 54.0 / 70 (77.14%)
    gpio: 90
    lvds or mipi dphy: 9
hsio_bg: 0 / 8 (0.0%)
hvio_poc: 0 / 4 (0.0%)
jtag: 1 / 4 (25.0%)
osc: 1 / 1 (100.0%)
pll: 4 / 4 (100.0%)
seu: 0 / 1 (0.0%)
---------- Periphery Usage Summary (end) ----------
```

Figure 28. 总体的资源整理

从 Figure 27 中可以看出，在实时缩放效果完全实现的情况下 Memory Blocks 和 DSP Blocks 占用量均较小，分别为 32.42%和 18.12%，证明了算法的有效性。从工程总体报告中可以看出，时钟频率配置被我们最大程度利用达到 100%。这些文档报告为我们团队进行工程的完善提供了重要的数据支持，帮助我们对工程不断进行规划和优化，避免出现过多闲置资源或资源瓶颈问题。

3.6 主客观性能指标评估

客观评估指标：

定量分析：峰值信噪比（ PSNR）

峰值信噪比是根据图像和算法处理后的输出图像在同一个坐标下的像素值的差异来评估缩放图像的质量，通常以分贝（dB）为单位表示。它是两个图像之间的最大可能像素值和均方误差（MSE）之间的比率的对数表达。这个值越大，表示图像失真越小。表达式为

$$\mathrm{PSNR} = 10\log_{10}\left(\frac{(2^n-1)^2}{\mathrm{MSE}}\right)$$

式中 MSE 代表均方误差，n 为像素值的比特数。可通过下式计算得到：

$$\mathrm{MSE} = \frac{1}{M \times N}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}[f(m,n) - g(m,n)]^2$$

通过分别对双线性插值与最近邻算法得到的不同结果与源图像进行对比比较不同算法的结果差异。由于 PSNR 通常用于比较两个相同尺寸的图像而缩放前后的图像尺寸与源图像不同，因此需要对源图像进行一定比例降采样再通过不同

算法放大至与源图像相同分辨率。计算脚本置于附录中，只需指定输入源图片路径与缩放后分辨率值即可得到 PSNR 与 MSE 的值。

以 Figure 14, Figure 15 演示的放大为例，计算得到：

缩放比例	最近邻插值	双线性插值
1/2	27.8	29.6
1/4	22.9	22.3

通过表格对比得知，一般认定若 PSNR 的值高于 36 则人眼无法区分图像差异，高于 24 则可以认定图像差异可以忽略不计 [1, 2]。由此得出结论，两个算法均反应出了理想性能且双线性插值在测试中表现优于最近邻插值算法。

定性分析

	最近邻插值	双线性插值	最近邻插值（经过预处理）	双线性插值（经过预处理）
是否产生锯齿	产生锯齿	否	否	否
物体边缘是否模糊	略微模糊	略有模糊	否	否
是否出现闪屏	否	否	否	否

主观评估指标：

我们在实验室中随机邀请实验者观察相同输入内容下的演示结果，基于观看舒适度，眼睛是否感觉疲劳，色彩是否有衰减，以及是否存在边缘模糊，综合给出感受分。

最近邻插值（预处理）

	观看舒适度	眼睛是否疲劳	色彩是否衰减	边缘模糊
1 分人数（最劣指标）	1	0	0	1
2 分人数	0	0	0	0

3 分人数	6	0	2	0
4 分人数	9	0	0	14
5 分人数（最优指标）	2	18	16	3
总计	18	18	18	18

双线性插值（预处理）

	观看舒适度	眼睛是否疲劳	色彩是否衰减	边缘模糊
1 分人数（最劣指标）	0	1	0	0
2 分人数	0	0	0	0
3 分人数	2	0	0	0
4 分人数	3	0	0	0
5 分人数（最优指标）	14	17	18	18
总计	18	18	18	18

第四部分　总结

4.1 可扩展之处

我们本次参赛使用的 Ti60F225 开发板支持 1.5Gbps MIPI CSI 和 DSI，使用于需要低功耗、小尺寸、多 I/O 的高度集成化移动和边缘设备，非常适用于人工智能互联网领域。因此，超分辨率、老电影修复是我们准备扩展的部分，通过本低功耗的硬件加速经典超分辨率算法如 SRCNN（超分辨率卷进神经网络）并利用高速接口传输图像数据。修复老电影时同样需要视频处理技术，如滤波去噪等，我们目前使用的数字滤波器等模块具有很高的复用性可以节约资源并节约开发周期；老电影往往存在帧速率较低的问题，可以使用帧插值技术来增加帧率提高视频的流畅性。常见算法又线性插值，在相邻帧之间插入新的一帧，新插入帧的每一个像素取值都来自前后帧对应像素点取值的线性插值，与我们采用的单帧图像中采用的二维线性插值法具有很高的相似性 [5]。因此，加入

AI 算法实现视频超分辨率是我们在完成任意比例视频流媒体缩放后的扩展步骤，正在逐步实现中。

4.2 心得体会

4.2.1 易灵思 IP 使用

在本次的比赛过程中，我们的项目中有效的使用了易灵思公司提供的 IP 核进行设计，有效的提高了我们的项目设计效率。以项目中用于存储双线性插值算法输入输出数据的 FIFO 为例，在进行生成并例化调用易灵思提供的 IP 核前，我们使用了自己编写的 FIFO 模块，但是在 Efinity 中编译的过程中，由于使用过大位宽在综合时消耗大量的逻辑资源甚至超过了 LUT 资源的十倍。为了减少资源的消耗，我们通过 IP 核的使用有效的降低了资源的消耗并极大提高了综合速度，节约了工程开发时间。

4.2.2 传输参数设置

.BURST_LEN 是每次进行 AXI 总线传输时的突发数据长度。由于突发长度参数增加时可以在相同传输周期内传输更多的数据从而提高总体的传输效率同时减少总线的占用时间，但会导致单次传输的延迟增加以及增加总线负责，增大总线竞争的可能，并且考虑到每次帧缓存模块总是接受 8 个 16 位的 yCbCr 颜色数据再发送，得出（1920*1080/8/BUSRT_LEN）的计算结果必须是一个整数的结论。在使用常见长度 128 后发现图像抖动严重可能是数据发送存在错位等问题，因此尝试了长度为 32、64 等符合上述条件并满足 2^N 的数据，经过实际测试得出当 BURST_LEN 取值 32 时效果最佳，视频帧几乎无抖动。因此我们得出结论，根据实际情况调整常见数据传输参数有时可能会得到意料之外的优质结果。

4.2.3 和谐的团队合作

通过本次比赛，本小组的三位组员在团结的小组氛围中通过共同学习，交叉指导攻克技术难题，一起通过视频播放出现的闪屏、模糊等问题与 Debugger 提供的波形分析代码架构与时序上的不足。和谐的团队氛围为项目的完成提供了坚实保障，浓郁的学习氛围与大家的全身心投入为项目的迭代增进动力。在分工的过程中，我们合理的进行了任务的划分，我们优先进行模块间连线的设计再分配任务，共同完成了设计。

4.2.4 首次使用国产 FPGA、国产 EDA 的感受

数电实验等课程安排中教学方均使用 Vivado EDA 工具并且使用 Xilinx 旗下的 FPGA 做教学演示与开发，因此学生并没有机会接触到国产 FPGA。通过本次竞赛中我们对于国产 FPGA 公司易灵思的选择，使用了易灵思自主研发的 Ti60F225 开发板、EDA 工具 Efinity 完成工程与产品的设计，体验了良好的开发体验。Efinity EDA 工具也对现有工具的一些不太便于使用的地方做出了相应调整，如在 RTL 框图部分加入"filter"功能方便开发者查找对应信号和模块、直接在对话框中打印 warnings 及 errors 信息，无需开发者点击层层文件夹去寻找日志文件，对于开发者保持平稳心态和节约开发时间起到重要作用；可以多“USER”抽头点添加并发调试，便于开发者通过仿真寻找错误所在。首次使用国产的 FPGA 和 EDA 工具做开发让我们体会到了国内相关产业飞速发展的势头。

第五部分 参考文献

[1]王轶楷. 实时视频图像缩放系统的 FPGA 硬件实现[D].中北大学,2023.DOI:10.27470/d.cnki.ghbgc.2023.000987.

[2] 吴以凯. 基于 FPGA 的视频缩放的设计与实现[D].江苏大学,2018.

[3] Bongsoon Kang, Jun Sunwoo, Byung-Hwan Chun and J. Gerard, "Improved performance of video decoder using down scaler and adaptive comb filter," *ISCE '97. Proceedings of 1997 IEEE International Symposium on Consumer Electronics (Cat. No.97TH8348)*, Singapore, 1997, pp. 47-50, doi: 10.1109/ISCE.1997.658348.

[4] M. Gupta and A. K. Nagawat, "Design and implementation of high performance advanced extensible interface(AXI) based DDR3 memory controller," *2016 International Conference on Communication and Signal Processing (ICCSP)*, Melmaruvathur, India, 2016, pp. 1175-1179, doi: 10.1109/ICCSP.2016.7754337.

[5] H. Okuhata, M. Ise, R. Y. Omaki and I. Shirakawa, "Implementation of super-resolution scaler for Full HD and 4K video," *2013 IEEE Third International Conference on Consumer Electronics ¿ Berlin (ICCE-Berlin)*, Berlin, Germany, 2013, pp. 1-5, doi: 10.1109/ICCE-Berlin.2013.6697987.

[6]苗莉,王昱煜,于小燕.基于 FPGA 的双线性插值视频缩放算法实现[J].信息与电脑(理论版),2021,33(08):84-86.

第六部分 附录

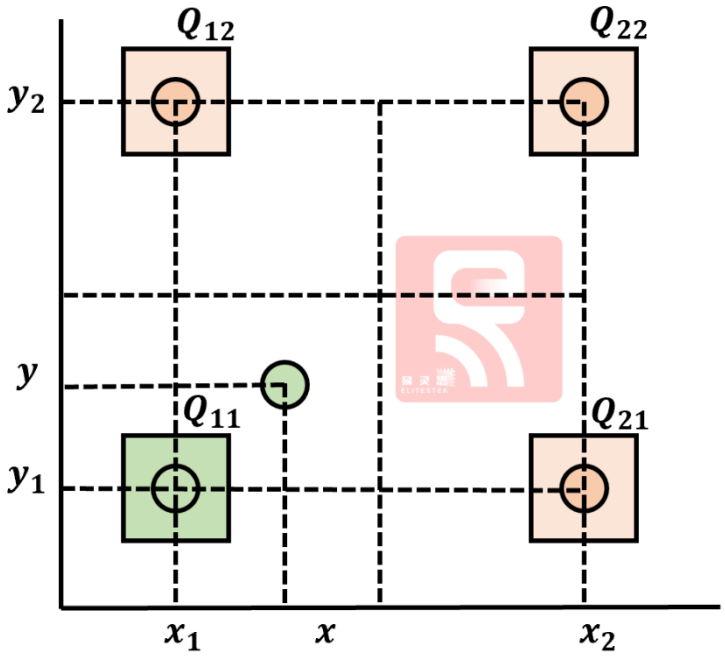

Figure 29. 最近邻算法示意图

最近邻（最近邻插值）算法的基本原理是选择距离期望位置最近的像素进行填补即相当于像素的复制粘贴，通过源图像像素点坐标、源图像与目标图像长宽比得出目标像素点的坐标。以屏幕左上角为远点（0，0），则有如下计算公式：

$$src_x = dst_x \times \left(\frac{src_{width}}{dst_{width}}\right)$$

$$src_y = dst_y \times \left(\frac{src_{height}}{dst_{height}}\right)$$

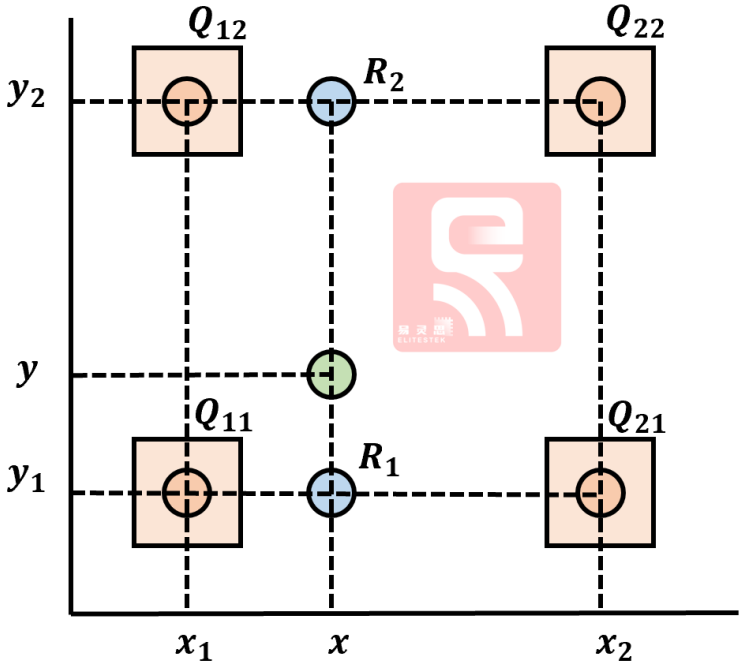

Figure 30. 双线性插值算法示意图

双线性插值的理论推导如下所示：

$$f(R_1) = \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21})$$

$$f(R_1) = \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

$$f(P) = \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2)$$

$$f(P) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} [x_2 - x \quad x - x_1] \begin{bmatrix} f(Q_{11}) & f(Q_{12}) \\ f(Q_{21}) & f(Q_{22}) \end{bmatrix} \begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix}$$

双线性插值实际应用时，相对于最近邻算法需要坐标轴远点的考虑。例如将3*3 的图像放大到 9*9 时，以左上角为坐标原点则源图像与目标图像的中心点

坐标分别为（2，2）、（5，5）；通过上述公式计算可以得知通过插值后计算取得源图像像素点坐标则为（1.67，1.67）。因此对于源图像，插值后的像素点取样偏离源像素点且整体偏左上并非理想的均匀分布整个图像。因此将原点选择放置在图像中心。坐标变换计算公式变为如下：

$$src_x = (dst_x + 0.5) \times (src_{width}/dst_{width}) - 0.5$$

$$src_y = (dst_y + 0.5) \times (src_{height}/src_{width}) - 0.5$$

相关重点代码：

双线性插值中系数的具体计算代码：

```
//Compute bilinear interpolation coefficinets. Done here because these pre-registerd values are used twice.
//Adding coeffHalf to get the nearest value.
wire [COEFF_WIDTH-1:0]  preCoeff00 = (((coeffOne - xBlend) * (coeffOne - yBlend) + (coeffHalf - 1)) >> FRACTION_BITS) & {{COEFF_WIDTH{1'b0}}, {COEFF_WIDTH{1'b1}}};
wire [COEFF_WIDTH-1:0]  preCoeff01 = ((xBlend * (coeffOne - yBlend) + (coeffHalf - 1)) >> FRACTION_BITS) & {{COEFF_WIDTH{1'b0}}, {COEFF_WIDTH{1'b1}}};
wire [COEFF_WIDTH-1:0]  preCoeff10 = (((coeffOne - xBlend) * yBlend + (coeffHalf - 1)) >> FRACTION_BITS) & {{COEFF_WIDTH{1'b0}}, {COEFF_WIDTH{1'b1}}};
coeff11 <= coeffOne - preCoeff00 - preCoeff01 - preCoeff10;
//Guarantee that all coefficients sum to coeffOne. Saves a multiply too. Reverted to previous method due to timing issues.
```

Figure 31. 双线性插值中系数的具体计算

判断是否为最近邻算法，若不是（即为双线性插值算法），则使用上述代码块实现；若是，则使用对应公式计算：

```
begin
    xBlend <= {1'b0, xScaleAmount[SCALE_FRAC_BITS-1:SCALE_FRAC_BITS-FRACTION_BITS]};    //Changed to registered to improve timing

    if(nearestNeighbor == 1'b0)
    begin
        //Normal bilinear interpolation
        coeff00 <= preCoeff00;
        coeff01 <= preCoeff01;
        coeff10 <= preCoeff10;
        coeff11 <= ((xBlend * yBlend + (coeffHalf - 1)) >> FRACTION_BITS) & {{COEFF_WIDTH{1'b0}}, {COEFF_WIDTH{1'b1}}};
    end
    else
    begin
        //Nearest neighbor interploation, set one coefficient to 1.0, the rest to zero based on the fractions
        coeff00 <= xBlend < coeffHalf && yBlend < coeffHalf ? coeffOne : {COEFF_WIDTH{1'b0}};
        coeff01 <= xBlend >= coeffHalf && yBlend < coeffHalf ? coeffOne : {COEFF_WIDTH{1'b0}};
        coeff10 <= xBlend < coeffHalf && yBlend >= coeffHalf ? coeffOne : {COEFF_WIDTH{1'b0}};
        coeff11 <= xBlend >= coeffHalf && yBlend >= coeffHalf ? coeffOne : {COEFF_WIDTH{1'b0}};
    end
end
```

Figure 32. 算法选择

计算 PSNR python 脚本代码如下：

```
import numpy as np
from PIL import Image
from skimage.metrics import mean_squared_error
from skimage.transform import resize

def calculate_psnr(mse, bit_depth=8):
    """
    Calculate the PSNR based on MSE and bit depth.

    Parameters:
```

```
    - mse: Mean Squared Error between the two images.
    - bit_depth: Bit depth of the images.

    Returns:
    - PSNR value.
    """
    max_pixel_value = 2 ** bit_depth - 1
    psnr = 20 * np.log10(max_pixel_value / np.sqrt(mse))
    return psnr


def load_and_resize_image(image_path, new_dimensions):
    """
    Load an image and resize it to new dimensions.

    Parameters:
    - image_path: Path to the image file.
    - new_dimensions: New dimensions as a tuple (width, height).

    Returns:
    - Resized image as a numpy array.
    """
    with Image.open(image_path) as img:
        # Convert image to grayscale if it's not already (to simplify PSNR calculation)
        if img.mode != 'L':
            img = img.convert('L')
        img_resized = img.resize(new_dimensions)
        return np.array(img_resized)


image_path = 'logo.png'  # Path to the original image
new_dimensions = (1920, 1080)  # New image dimensions (width, height)

# Load and resize the image
resized_image = load_and_resize_image(image_path, new_dimensions)

original_image = np.random.rand(new_dimensions[1], new_dimensions[0])

# Calculate MSE
mse = mean_squared_error(original_image, resized_image)

# Calculate PSNR
psnr = calculate_psnr(mse)
```

```
print(f"PSNR: {psnr} dB")
```