基于ZYNQ的图像拼接全局系统

参赛队员：曾阳 彭鹏 李傲雪

2023 年 11 月

基于 ZYNQ 的图像拼接全局系统

作品简介：

图像全局拼接系统是利用 FPGA 加速，采集图像上传对图像进行拼接，可以将数张有重叠部分的图像（可能是不同时间、不同视角或者不同传感器获得的）拼成一幅无缝的全景图或高分辨率图像,改善了目前对图像拼接技术不成熟,存在精度低和速度慢的问题。针对这些问题,我们将采用 FPGA 作为视频采集卡，充分利用 FPGA 高度并行的特性，在视频采集部分做到加速整体系统的功能，采集到图像后通过千兆网传出视频或者图像信息，实现整体的硬件层面加速，结合后续特征检测与匹配、图像匹配图验证、用 APAP 生成匹配点、焦距和三维旋转估计、缩放和旋转、网格优化、通过纹理映射合成结果等处理实现整体系统功能实现。为了更直观地观察结果，我们将拼接好的图像结果实时输出在 HDMI 显示屏上，最终实现了较好的图像拼接效果，识别率和识别效果均满足我们的预期。

作品创新点：

1）我们采用 FPGA 和 NPU CPU 配合处理,弥补了 PS 核性能不足的缺点。NPU（神经网络处理器/网络处理器）是一种专门应用于网络应用数据包的处理器，采用“数据驱动并行计算”的架构，擅长处理视频、图像类的海量多媒体数据。以 NPU CPU 辅助 FPGA，更好地对图像进行处理，进行算法解析，可以提高速度，减小功耗

2）研究基于 FPGA 平台的图像拼接检测系统能够实现良好的人机交互功能;又利用 FPGA 在并行算法加速、可动态重配置的特点,实现硬件加速、增加系统的灵活性，提高图像拼接检测的速度。

3）摒弃使用传统采集方式，创新利用 FPGA 并行处理数据的特点，处理速度更快，整体稳定性更高，充分利用 FPGA 的资源与其并行性，与千兆网通讯实现数据传输过程中的提速的特点加快拼接算法的执行速度，提高了拼接的效率，实现高效的图像拼接。

4）可以实现高精度拼接，采用了更高级的特征提取方法，提取和匹配图像特征，从而提高拼接的精度；实时拼接：实现实时图像拼接，即对实时视频流进行拼接处理。通过优化算法和并行计算技术，实现快速的实时拼接，可以适用于实时监控、虚拟现实等应用场景，可开发空间大。

5）采用基于全局相似先验的自然图像拼接，充分避免了传统图像拼接

因视觉中心位置导致的图像翘曲问题，采用适当比例的缩放和旋转，更加自然的实现图像拼接效果。算法具有以下优点：精确对准，减少形状失真，自然性并且不受视野限制。

基于 ZYNQ 的图像拼接全局系统

曾阳；彭鹏；李傲雪

目录

第一部分　设计概述

1.1 设计目的

图像拼接技术是将一组相互间存在重叠部分的图像序列进行空间配准,经采样融合后形成一幅宽视角场景的,高清晰的新图像的技术。由于一些前沿产业的视频图像流畅度，分辨率的飞速提高，使得传统算法的工作量不断攀升，处理时间不断增长，已无法满足现有市场要求。而在基于FPGA的图像拼接技术方面，可行的方案较少。因此，我们所做的项目就是利用FPGA设计开发一个基于ZYNQ的图像拼接全局检测系统,在识别的基础上做到图像拼接全局,形成一幅宽视角场景的,高清晰的新图像，并且做到实时显示图像

1.2 应用领域

图像拼接技术在计算机视觉,遥感图像分析,医学,军事,气象,地质勘测,物体的3D 建模,相机的超分辨率等领域都有广泛的应用。主要表现为:医学领域，医学图像拼接技术可以将不同角度和类型的医学图拼接在一起，帮助医生更准确地诊断疾病；遥感领域，将不同时间、不同传感器拍摄的遥感图像拼接在一起，生成大范围的遥感图像；虚拟现实（VR）和增强现实（AR），将现实世界和虚拟世界的图像拼接在一起，为用户提供沉浸式的体验；还体现在在安防监控电影制作，地图导航，科学研究等方面

1.3 主要技术特点

本作品简单易用且功能强大。在硬件方面，我们设计了双路 Ov5640 图像视频信息摄像头采集,通过设计多个 IP 核实现图像显示,也设计了千兆网交互模块,以便上位机方便直接采集数据进行处理。在软件设计部分,我们设计了程序固化,视频流处理，以太网交互，串口发送，以及系列的驱动配置

我们将采用 FPGA 实现算法加速,FPGA 具有灵活性,可编程和可重构性,可以依照神经网络的算法来设计硬件结构，利用 FPGA 在与 CPU、NPU 平台实现相互配合提高运行速率，利用 FPGA 并行化这一特性加速系统整体处理速度，我们采用千兆网通讯实现数据传输过程中的提速，也由于 FPGA 自身低延迟

的优势，也加快了图像拼接全局检测的速度，FPGA 相比于通用处理器，FPGA 在执行特定任务时通常具有较低的功耗，可以通过关闭未使用的部分来最小化功耗，并通过定制电路设计来提高效率。

1.4 关键性能指标

我们的图像拼接全局检测系统包括双路采集模块、网口通讯模块，资源利用率如图 1-1 所示，查找表资源如图 1-2 所示

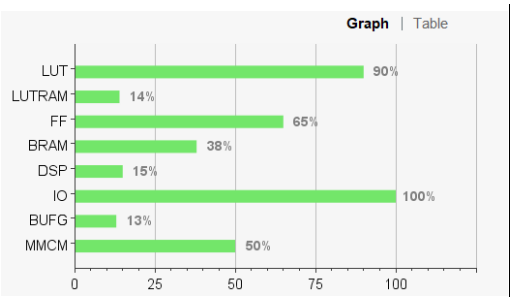

图 1-1 资源利用率

Resource	Estimation	Available	Utilization %
LUT	15897	17600	90.32
LUTRAM	856	6000	14.27
FF	22890	35200	65.03
BRAM	22.50	60	37.50
DSP	12	80	15.00
IO	100	100	100.00
BUFG	4	32	12.50
MMCM	1	2	50.00

图 1-2 查找表资源

经过对系统的不断完善，目前我们的系统能够实现图像拼接，并且可以实时显示，响应时间短，分辨率高，实现对于图像的拼接全局系统设计。

在图像拼接全局设计中，由于 FPGA 的并行能力以及一些附加的模块等，使得其运算速度快，图像拼接加载速度快，系统处理能力强，吞吐量大，错误率小，图像拼接不会产生太大的误差，拼接好的图像比较清晰，屏幕的刷新率和分辨率高，可以完整地显示整个图像。

1.5 主要创新点

1） 我们采用 FPGA 和 NPU CPU 配合处理，弥补了 PS 核性能不足的缺点。NPU（神经网络处理器/网络处理器）是一种专门应用于网络应用数据包的处理器，采用“数据驱动并行计算”的架构，擅长处理视频、图像类的海量多媒体数据。以 NPU CPU 辅助 FPGA，更好地对图像进行处理，进行算法解析，可以提高速度，减小功耗

2）研究基于 FPGA 平台的图像拼接检测系统能够实现良好的人机交互功能;又利用 FPGA 在并行算法加速、可动态重配置的特点，实现硬件加速、增加系统的灵活性，提高图像拼接检测的速度。

3）摒弃使用传统采集方式，创新利用 FPGA 并行处理数据的特点，处理速度更快，整体稳定性更高，充分利用 FPGA 的资源与其并行性，与千兆网通讯实现数据传输过程中的提速的特点加快拼接算法的执行速度，提高了拼接的效率，实现高效的图像拼接。

4）可以实现高精度拼接，采用了更高级的特征提取方法，提取和匹配图像特征，从而提高拼接的精度；实时拼接：实现实时图像拼接，即对实时视频流进行拼接处理。通过优化算法和并行计算技术，实现快速的实时拼接，可以适用于实时监控、虚拟现实等应用场景，可开发空间大。

5）采用基于全局相似先验的自然图像拼接，充分避免了传统图像拼接因视觉中心位置导致的图像翘曲问题，采用适当比例的缩放和旋转，更加自然的实现图像拼接效果。算法具有以下优点：精确对准，减少形状失真，自然性并且不受视野限制。

第二部分　系统组成及功能说明

2.1 整体介绍

给出芯片或系统整体框图，各子模块标注清楚，并进行整体的文字说明，需要表达出各模块之间的关系。

2.1.1 板子资源介绍

图 2-1　zynq 7010 开发板

我们选用了 xlinx zynq 7010 开发板，如图 2-1 所示。此开发板芯片可分成处理器系统部分 Processor System（PS）和可编程逻辑部分 Programmable Logic（PL），PS 部分和 PL 部分都搭载了丰富的外部接口 和设备，方便我们的使用和功能验证。其中 PS 系统集成了两个 ARM cortex a9 处理器，AMBA 互连，内部存储器，外部存储器接口和外设。这些外设主要包括 USB 总线接口，以太网接口，SD/SDIO 接口与 I2C 总线接口， CAN 总线接口，UART 接口，GPIO 等。硬件资源十分丰富，满足作品需要的同时，最大限度降低了整体功耗。

2.1.2 FPGA 硬件软件设计

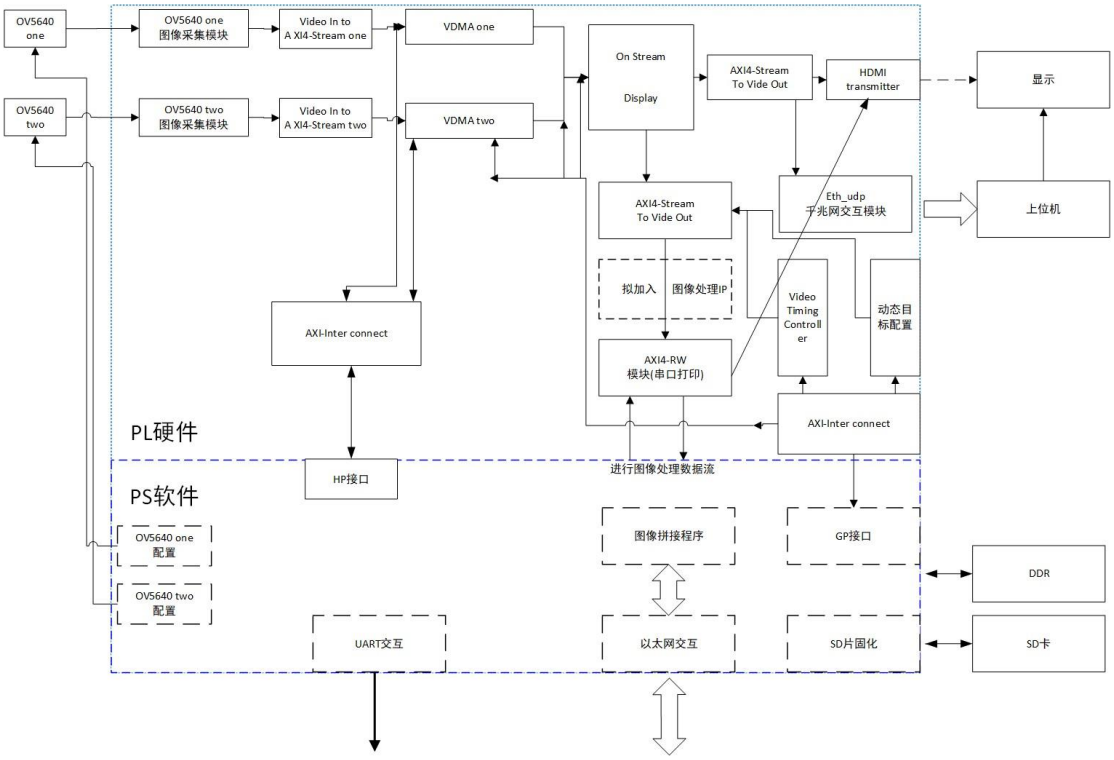

图 2-2 FPGA 硬件软件设计

如图 2-2 所示是我们识别系统的整体框架。其中 OV5640 图像采集是我们自定义的 IP 核，负责将 OV5640 的数据转成视频流数据；视频流数据经过 Video in to AXI4-Stream IP 转换成 AXI4-Stream IP 格式数据流，然后通过 VDMA 的 写通道转成 AXI4 Memory Map 格式，并最终写入 DDR 内存中。VDMA 通过 AXI InterConnect IP 核与 AXI_HP 端口进行连接，从而高效访问 DDR3。同时我们在 PS 软件中设计了程序固化，视频流处理，以太网交互，串口发送，以及系列的驱动配置，以便更好地进行通信。

VDMA 输出的 AXI4-Stream 格式的图像数据连接至 OSD IP 核，经 OSD IP 核拼接和叠加字符后，连接至 AXI4-Stream to Video Out IP 核。AXI4- Stream to Video Out IP 核在 VTC IP 核的控制下，把 AXI4-Stream 格式的数据转换成视频输出的数据格式，一部分将输出的视频数据流连接至 DVI Transmitter IP 核的输入端，DVI Transmitter IP 核是用于驱动 HDMI 接口的 IP 核，HDMI 进行显示；另一部分将输出的数据通过串口打印，驱动 HDMI，HDMI 显示图像视频数据。

此外，我们在 PL 设计中也设计了千兆网交互模块，数据通过 PL 千兆网口通讯模块，传输到上位机，由上位机驱使显示图像。千兆网交互在数据和上位机间起着一个桥梁的作用，当数据准备好，千兆网将数据快速地传输到上位机中，从而进行后续的显示，以便上位机方便直接采集数据进行处理，提高速度与效率。

2.2 各模块介绍

根据总体系统框图，给出各模块的具体设计说明。

2.2.1 OV5640 摄像头

OV5640 摄像头：如图 2-3 所示，OV5640 摄像头模组采用美国 OmniVision(豪威)CMOS 芯片图像传感器 OV5640，支持自动对焦 的功能。OV5640 芯片支持 DVP 和 MIPI 接口, OV5640 摄像头模组通过 DVP 接口和 FPGA 连接实现图像的传输。

FPGA 与 CMOS 模块通过 DVP 接口进行连接，利用 IIC 总线进行通讯。每个连接到 IIC 总线的设备都具有唯一地址，该总线主要用于主控制器与从控制器之间的通信，只需串行时钟线（SCL）和串行数据线（SDA）两根信号，所以一次性可挂载多个设备。IIC 协议读写时序状态机如图 2-4，整个状态机分为八个状态，分别为空闲态、写从设备 ID、写从设备寄存器地址、写数据、读地址、读数据和写结束。利用多段式的有限状态机，在写从设备寄存器地址和写数据完成后，可完成对 CMOS 模块内部寄存器的配置，接收到来自 DVP 引脚的图像数据。

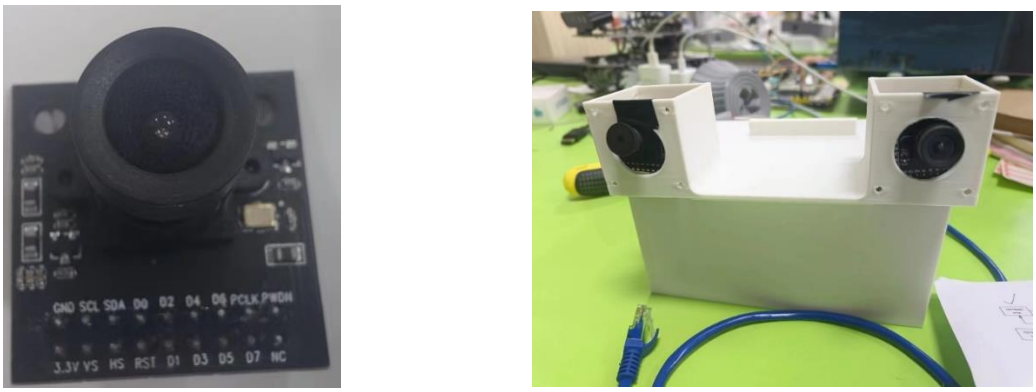

图 2-3 OV5640 摄像头模块

2.2.2 PL 千兆网口通讯模块

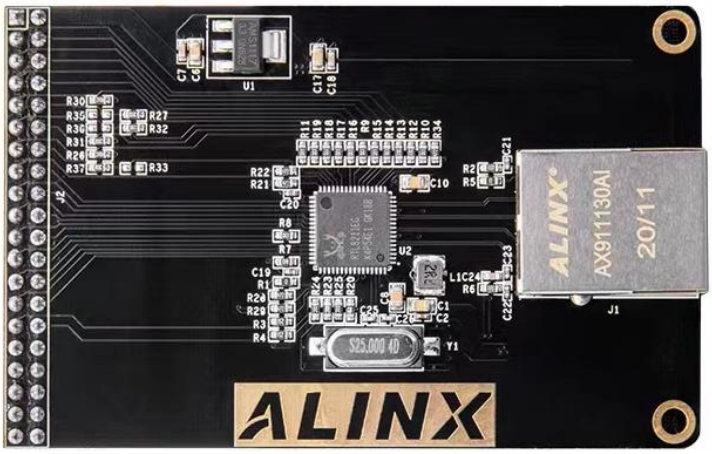

图 2-4 PL 千兆网口通讯模块

PL 千兆网口通讯模块：如图 2-4 所示，PLC (Programmable Logic Controller, 可编程逻辑控制器)是一种智能控制设备。千兆网口通讯，顾名思义，指的是 PLC 设备通过以太网千兆网口进行数据通讯。千兆网口通讯具有速度快、稳定可靠等优势，成为了现代工业自动化控制系统中的常见选择。AXI4- Stream to Video Out IP 核在 VTC IP 核的控制下，把 AXI4-Stream 格式的数据转换成视频输出的数据格式，通过 PL 千兆网口通讯模块，传输到上位机，由上位机驱使显示图像。千兆网交互在数据和上位机间起着一个桥梁的作用，当数据准备好，千兆网将数据快速地传输到上位机中，从而进行后续的显示。

2.2.3 OV5640 双路采集模块

OV5640 双路采集是我们自定义的 IP 核，负责将 OV5640 的数据转成视频流数据；视频流数据经过 Video in to AXI4-Stream IP 转换成 AXI4-Stream IP 格式数据流，然后通过 VDMA 的写通道转成 AXI4 Memory Map 格式，并最终写入 DDR 内存中。VDMA 通过AXI InterConnect IP 核与AXI_HP 端口进行连接，从而高效访问DDR3

其中VDMA 是 xilinx 的视频处理中一个很关键的 IP，VDMA针对视频处理做了特殊的设计。VDMA 有 AXI4 Memory Map 接口，用于对存储器进行读写视频数据；AXI4-Lite 接口用于读取 VDMA 状态以及配置 VDMA 的参数；

AXI4-Stream 接口，用于视频的输入和输出。

AXI Interconnect IP 将一个或多个 AXI 存储器映射的主器件连接到一个或多个存储器映射的从器件，此模块真正实现了总线通信。

AXI4_Stream to Video Out 核心设计用于从实现视频协议的 AXI4-Stream 接口连接到视频源。AXI4-Stream to Video Out IP 将AXI4-Stream视频协议从使用此协议的 Xilinx 视频处理 IP 转换为带有行场同步和特定时序信号的视频输出接口，从而与使用该视频时序的普通视频系统接口相互连接。

Video Timing Controller IP 是一个通用视频时序信号发生器和检测器。所有的视频系统都需要视频时序信号的管理，这些信号用于同步传输进程。

OV5640模块原理图如图2.5

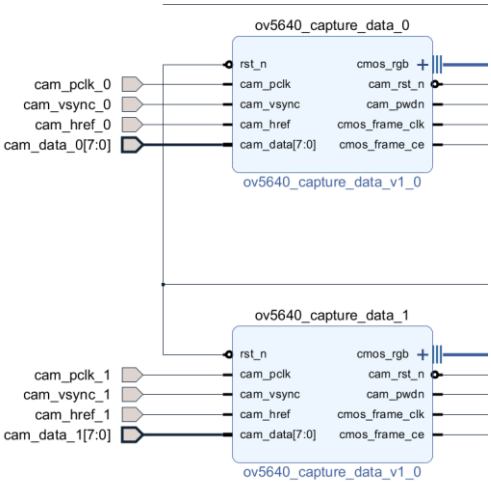

图2.5 OV5640双目采集

2.2.4 网口通讯模块

网口通信是指通过以太网接口进行数据传输和通信的方式。以太网接口也称为网口，是计算机或设备与局域网或互联网相连的接口。通过网口通信，可以实现计算机之间的数据交换和通信，包括传输文件、发送消息、远程控制等功能。在网口通信中，数据会被分成数据包进行传输，每个数据包包含目标地址、源地址、数据内容和错误检测等信息。网口通信使用的协议通常是TCP/IP协议栈，其中TCP（传输控制协议）负责可靠的数据传输，而IP（互联网协议）则负责数据的路由和寻址。网口通信的优点包括数据传输速度快、传输距离远、连接数量多等。它广泛应用于局域网、广域网、互联网等各种网络环境中。在PS软件中我们设计了以太网交互，图像拼接程序可以通过以太网交互，以达到可以远程控制该

程序，同时，由于以太网的设计，便于开发，与其他功能相适应，提升的空间很大

同时，在PS软件中还设计了UART交互，与以太网交相辅相成。UART交互适用于简单、实时性要求较高的点对点通信场景，而以太网交互适用于网络化、远程连接和大规模连接的应用场景

UART交互的优势：

1. 简单和直接：UART通信是基于串口的，不需要任何网络设备或协议栈的支持，只需要连接两个设备的串口线即可进行通信。

2. 实时性较高：UART通信通常是点对点的单向或双向通信，数据传输速度较快，延迟较低，适用于对实时性要求较高的应用场景。

3. 成本较低：UART通信只需要简单的串口线和相关硬件支持，成本相对较低。

以太网交互的优势：

1. 网络化和远程连接：以太网通信是基于网络的，可以连接多个设备并通过路由器进行远程连接，实现设备之间的远程通信和控制。

2. 大规模连接和扩展性：以太网可以连接大量的设备，支持多设备同时通信，具有较高的扩展性和连接数量的灵活性。

3. 通信速度快：以太网通信采用TCP/IP协议栈，具有较高的传输速度，适用于大规模数据传输和对带宽要求较高的应用场景。

2.2.5 HDMI 显示模块

图2.6 HDMI屏幕

HDMI屏幕显示拼接到的图像结果并根据 Zynq ax7z010 开发板输出的预设指令给出相应反馈。屏幕为7寸，分辨率为1024*600px，刷新率和分辨率高。

VDMA 输出的 AXI4-Stream 格式的图像数据连接至 OSD IP 核，经 OSD IP 核拼接和叠加字符后，连接至 AXI4-Stream to Video Out IP 核。AXI4- Stream to Video Out IP 核在 VTC IP 核的控制下，把 AXI4-Stream 格式的数据转换成视频输出的数据格式，一部分将输出的视频数据流连接至 DVI Transmitter IP 核的输入端，DVI Transmitter IP 核是用于驱动 HDMI 接口的 IP 核，HDMI进行显示；另一部分将输出的数据通过串口打印，驱动HDMI，HDMI 显示图像视频数据。

HDMI显示原理图如图2.7 HDMI显示原理图

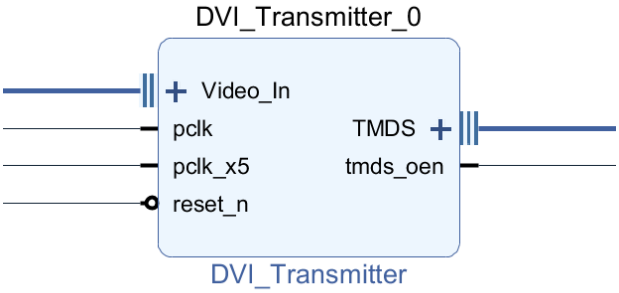

图2.7 HDMI显示原理图

2.3 基于全局相似性先验的自然图像拼接算法原理

2.3.1 算法定义及优点

基于全局相似性先验的自然图像拼接算法采用局部扭曲模型，用网格引导每个图像的变形。目标函数用于指定扭曲的所需特征。除了良好的对齐和最小的局部失真之外，我们还在目标函数中添加了全局先验相似性。该先验约束每个图像的扭曲，使其类似于整体的相似变换。选择相似性变换对结果的自然性至关重要。我们提出了为每个图像选择合适的比例和旋转的方法。所有图像的扭曲被一起解决，以最小化全局失真。综合评估表明，所提出的方法始终优于多种最先进的方法，包括 AutoStitch，APAP，SPHP 和 ANNAP。

图像拼接是将多个图像组合成具有更宽视场的更大图像的过程。早期的方法专注于提高无缝拼接的对齐精度，例如找到全局参数扭曲以使图像对齐。全球扭曲很强大，但往往不够灵活。我们的方法具有以下优点：精确对准，减少形状失真，自然性并且不受视野限制。

2.3.2 算法原理

找到每个图片的特征及其匹配，确定图像之间的邻接，对相邻图像应用 APAP，并使用对齐结果来生成匹配点。通过网格变形来缝合图像，为了使拼接尽可能自

然，我们添加了一个全局相似项，要求每个变形图像经历相似变换。为了确定每个图像的相似性变换，估计每个图像的焦距和 3D 旋转，然后选择最佳比例和旋转，最后进行网格优化，通过纹理映射合成结果

a）寻找每个图片的特征及其匹配点，确定图像之间的邻接，对相邻图像应用 APAP，并使用对齐结果来生成匹配点，如图 2.8 寻找匹配点

图 2.8 寻找匹配点

b）通过网格变形来缝合图像，使用网格来引导图像变形，设 Vi 和 Ei 表示图像 Ti 的网格中的顶点和边的集合。V 表示所有顶点的集合。我们的拼接算法试图找到一组变形的顶点位置 V¯，使得能量函数 Ψ（V）最小化。我们的方法尽可能地保留每个图像的原始视角。 同时，在全局范围内，它试图通过为图像寻找合适的比例和旋转来保持良好的结构。 两者都有助于缝合的自然性。 因此，我们的能量函数由三个项组成：

对齐项 Ψa，通过保持匹配点与它们的对应关系对齐来确保变形后的对准质量

$$\Psi_a(\mathbf{V}) = \sum_{i=1}^{N} \sum_{(i,j)\in\mathbf{J}} \sum_{p_k^{ij}\in\mathbf{M}^{ij}} \|\tilde{v}(p_k^{ij}) - \tilde{v}(\Phi(p_k^{ij}))\|^2,$$

局部相似项 Ψl，用于正则化并且将对齐约束从重叠区域传播到非重叠区域，确保每个四边形经历相似变换，以便形状不会过度扭曲

$$\Psi_l(\mathbf{V}) = \sum_{i=1}^{N} \sum_{(j,k)\in\mathbf{E}_i} \|(\tilde{v}_k^i - \tilde{v}_j^i) - \mathbf{S}_{jk}^i(v_k^i - v_j^i)\|^2,$$

全局相似项 Ψg，要求每个变形图像尽可能地经历相似变换

$$\Psi_g(\mathbf{V}) = \sum_{i=1}^{N} \sum_{e_j^i\in\mathbf{E}_i} w(e_j^i)^2 \left[(c(e_j^i) - s_i\cos\theta_i)^2 + (s(e_j^i) - s_i\sin\theta_i)^2\right]$$

网格的最佳变形由以下因素确定

$$\tilde{\mathbf{V}} = \arg\min_{\tilde{\mathbf{V}}} \Psi_a(\mathbf{V}) + \lambda_l \Psi_l(\mathbf{V}) + \Psi_g(\mathbf{V}).$$

网格变形引导图像变形如图 2.9，图 2.10

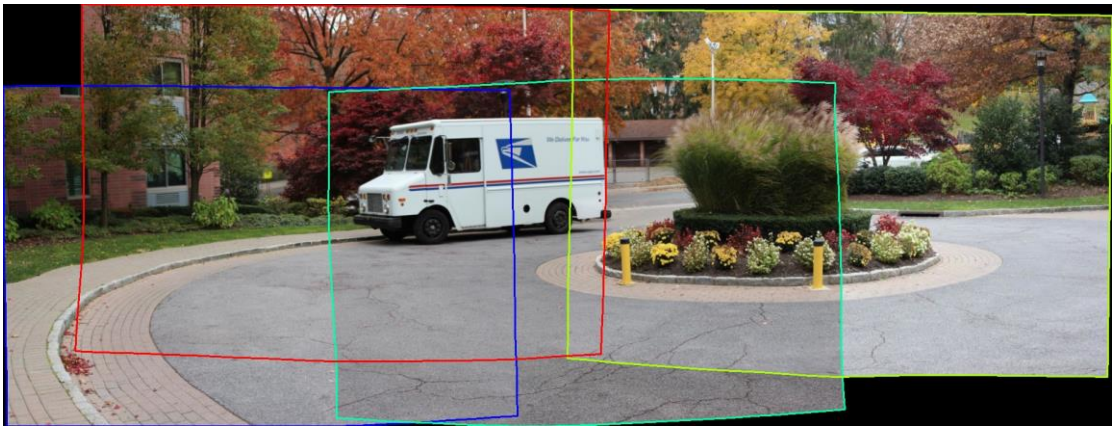

图 2.9

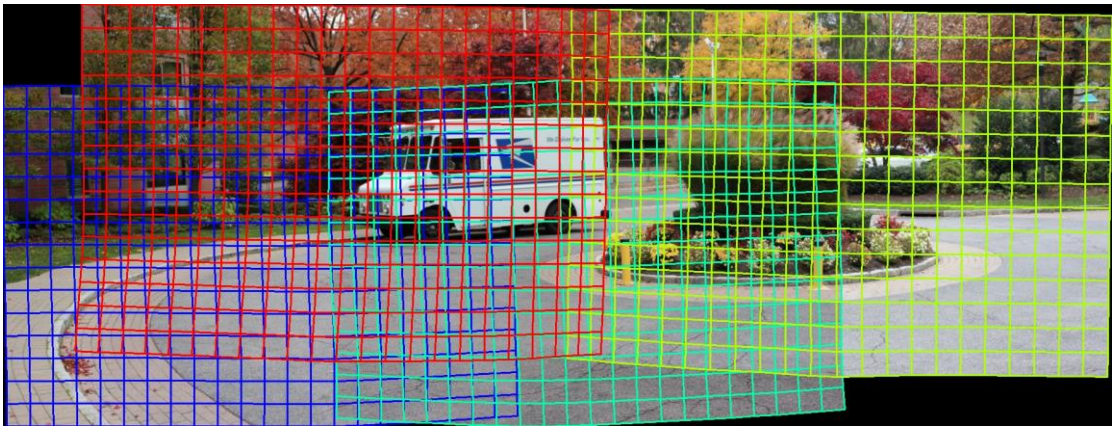

图 2.10

c）缩放和旋转选择

从两幅图像之间的单应性，我们可以估计两幅图像的焦距。 在执行 APAP 之后，我们对网格的每个四边形都有一个单应性。 因此，每个四边形给出了对图像焦距的估计。 我们将这些估计的中值作为焦距的初始化并形成 Ii 的初始内在矩阵 Ki。一旦我们得到 Ki，我们通过最小化以下投影误差获得 Ii 和 Ij 之间的 3D 旋转 Rij 的初始估计：

$$\mathbf{R}_{ij} = \arg\min_{\mathbf{R}} \sum_{p_k^{ij} \in \mathbf{M}^{ij}} \|\mathbf{K}_j \mathbf{R} \mathbf{K}_i^{-1} p_k^{ij} - \Phi(p_k^{ij})\|^2.$$

旋转选择：旋转选择的目标是为每个图像 Ii 分配旋转角 θ_k^{ij} 。相对旋转范围。给定一对相邻图像 Ii 和 Ij，其每对匹配点唯一地确定相对旋转。假设第 k 对匹配

点给出了相对旋转角θ_k^{ij}。我们将 Ii 和 Ij 之间的相对旋转范围定义为

$$\Theta^{ij} = [\theta^{ij}_{min}, \theta^{ij}_{max}],$$

$$\theta^{ij}_{min} = \min_k \theta^{ij}_k \text{ and } \theta^{ij}_{max} = \max_k \theta^{ij}_k.$$

旋转如图 2.11

图 2.11 旋转

d）最小线路失真旋转

在线对齐方面找到两个相邻图像之间的最佳相对旋转，通过 APAP 给出的对齐，我们可以找到两个相邻图像 Ii 和 Ij 之间的线的对应关系。 每对相应的线唯一地确定相对旋转。我们将表示为由 MLDR 确定的 Ii 和 Ij 之间的相对旋转角。

给定由 MLDR 估计的所有相对旋转角，我们可以找到一组旋转角{}以尽可能地满足 MLDR 成对旋转关系。 我们将θ_k^{ij}表示为单位 2D 矢量(u_i, v_i)并表示以下能量函数：

$$\mathbf{E}_{MLDR} = \sum_{(i,j)\in\mathbf{J}} \left\| R(\phi^{ij}) \begin{bmatrix} u_i \\ v_i \end{bmatrix} - \begin{bmatrix} u_j \\ v_j \end{bmatrix} \right\|^2$$

最小线路失真旋转如图 2.12

图 2.12 最小线路失真旋转

e）旋转方法

$\mathbf{E}_{MLDR} + \lambda_z \mathbf{E}_{ZERO}$, where

$$\mathbf{E}_{ZERO} = \sum_{i \in \Omega} \left\| \begin{bmatrix} u_i \\ v_i \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\|^2$$

2D 旋转

$$\sum_{(i,j) \in \Omega} \left\| R(\phi^{ij}) \begin{bmatrix} u_i \\ v_i \end{bmatrix} - \begin{bmatrix} u_j \\ v_j \end{bmatrix} \right\|^2 + \lambda_r \sum_{(i,j) \in \Omega} \left\| R(\alpha^{ij}) \begin{bmatrix} u_i \\ v_i \end{bmatrix} - \begin{bmatrix} u_j \\ v_j \end{bmatrix} \right\|^2$$

3D 旋转

第三部分　完成情况及性能参数

具体完成的情况，可图文结合，具体的性能参数等量化指标。系统由展示台、Zynq 7010 开发板、OV5640 摄像头、HDMI 显示屏组成如图 3.1、3.2、3.3 所示；OV5640 双目采集图像如图 3.4、3.5；HDMI 显示拼接好的图像如图 3.6

图 3.1 图 3.2 图 3.3 测试平台多方位展示

图 3.4 图 3.5 OV5640 双目采集图像

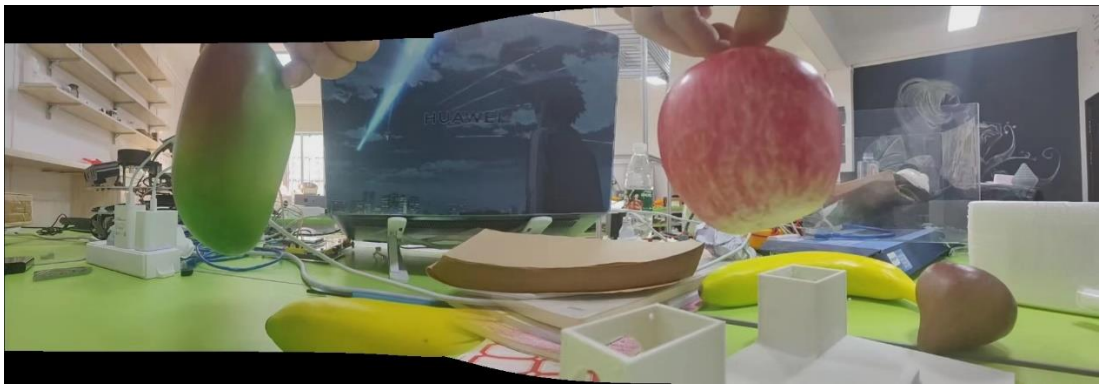

图 3.6 HDMI 显示拼接好的图像

目前能够借助上位机能够达到图像拼接全局检测识别功能，能够实现基于

Zynq 7010 的图像拼接全局检测系统设计,优化图像处理技术,图像拼接全局检测技术,更美观,快速且清晰地显示图像。

已实现的功能: ①OV5640 双路采集，HDMI 显示 ，上位机利用千兆网模块驱动显；②程序固化 SD 卡，DDR 缓存数据；③PS 端以太网数据交互，UART 交互。其中设置的 IP 核有：①OV5640 图像采集 IP ②HDMI 数据流转换 IP ③PL 兆网交互 IP ④AXI4_RW IP 等，随着设计规模增大，复杂度提高，使用 IP 核可以提高开发效率，减少设计和调试时间，加速开发进程，降低开发成本，利用 IP 核设计电子系统，引用方便，修改基本元件的功能容易，在开发中设计了不少 IP 核，便于开发。

在图像拼接全局检测中，由于 FPGA 的并行能力以及一些附加的模块等，使得其运算速度快，图像拼接加载速度快，系统处理能力强，吞吐率大，错误率小，图像拼接不会产生太大的误差，拼接好的图像比较清晰，可以完整地显示整个图片。

第四部分　总结

4.1 可扩展之处

我们将此应用到 ZYNQ 7010 上，图像拼接技术的效果好，能够在此基础上进行大多数的图像应用，也可以加入更好的深度学习框架达到更好的效果，将初步处理后的图像结果，放入 FPGA 设备中即能扩展完成其他的识别任务，甚至可以运用到一些更加高端的产业，由于 FPGA 特有的并行处理能力，可以实现更稳定，更快，更精确的实时显示，当然也可以配合一些功能实现，例如可以配合完成机器视觉算法，又可实现对机器人的电控，节约资源。当然，对于图像拼接技术，还可以加强更进，实现更多的功能:

(1)摄像头采集图像可与手机用 WIFI 相连，通过手机的 APP 也可以控制拍摄。

(2)显示屏也可以实现无线传输，更加便利。

(3)可以图像进行加密，或者可以设定指令加密具体某个信息，随信息变换位置而随着加密

4.2 心得体会

通过这一次竞赛，我们最开始对 FPGA 的一无所知，在过程中步步了解，不断学习新知识，然后我们利用 FPGA 的特性开发了这个图像拼接，并且效果符合我们预期。

在学习过程中，我们学习了采用基于全局相似先验的自然图像拼接的算法，它充分避免了传统图像拼接因视觉中心位置导致的图像翘曲问题，采用适当比例的缩放和旋转，更加自然的实现图像拼接效果。首先找到每个图片的匹配点，确定图像之间的邻接；通过网格变形来缝合图像，添加全局相似项，要求每个变形图像经历相似变换；同时进行图像的相似性变换，估计每个图像的焦距和 3D 旋转，然后选择最佳比例和旋转，最后进行网格优化，通过纹理映射合成结果，完成图像拼接全局设计。

通过对图像拼接技术的搭建，了解了数字图像处理的各种算法及如何运用 FPGA 进行图像处理的办法，也学习了如何充分发挥 FPGA 设备的并行性这一特点，并且在保持精度的同时如何高效利用 FPGA 的资源。在此过程中我们也遇到了很多问题，但是最后大部分都被我们一一解决，利用数字图像处理技术对采集的图像进行处理，从而对图像进行拼接，在这方面，FPGA 应用于各个领域具备很强的潜在的社会经济效益。

总体来说，对于此次研究的图像拼接检测技术有了更深的了解，对于编程的语言和算法有深刻感悟，每次有阶段性进展，就离成功更进一步不。付出就会有收获，失败是成功之母，不论困难有多大，只要在我们小组共同的团结努力下，都能迎风前行。

第五部分　参考文献

[1]赵创.基于 FPGA 的实时图像拼接系统的设计与实现[D].天津大学,2023.DOI:10.27356/d.cnki.gtjdu.2020.004116.

[2]徐扬,王晓曼,朱佶等.基于 FPGA 的图像拼接技术研究与实现[J].长春理工大学学报(自然科学版),2018,41(06):94-98+103.

[3]李勇,王磊,钱罕林.基于 FPGA 的全景视频图像拼接的设计与实现[J].电子设计工程,2018,26(02):80-83.DOI:10.14022/j.cnki.dzsjgc.2018.02.018.

[4]陈全兵. 基于 FPGA 的高清实时全景视频拼接的研究与设计[D].电子科技大学,2016.

[5]武晓斌. 基于FPGA的图像拼接系统的设计[D].西安电子科技大学,2016.

[6]田玉晓. 图像拼接技术的 FPGA 实现[D].广西科技大学,2021.DOI:10.27759/d.cnki.ggxgx.2017.000037.

[7]MATTHEW BROWN* AND DAVID G. LOWE;Automatic Panoramic Image Stitching using Invariant Features;Department of Computer Science, University of British Columbia, Vancouver, Canada;2007 年

[8]Yu-Sheng Chen(B) and Yung-Yu Chuang;Natural Image Stitching with the Global Similarity Prior;Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan;2016 年

第六部分 附录

6.1 视频播放并转串

```
`timescale 1ns / 1ps
module serializer_10_to_1(
    input               reset,
    input               paralell_clk,
    input               serial_clk_5x,
    input   [9:0]   paralell_data,

    output          serial_data_out
    );
//wire define
wire        cascade1;
wire        cascade2;
OSERDESE2 #(
```

```
        .DATA_RATE_OQ     ("DDR"),
        .DATA_RATE_TQ     ("SDR"),
        .DATA_WIDTH       (10),
        .SERDES_MODE      ("MASTER"),
        .TBYTE_CTL        ("FALSE"),
        .TBYTE_SRC        ("FALSE"),
        .TRISTATE_WIDTH (1)
)
OSERDESE2_Master (
        .CLK              (serial_clk_5x),
        .CLKDIV           (paralell_clk),
        .RST              (reset),
        .OCE              (1'b1),

        .OQ               (serial_data_out),

        .D1               (paralell_data[0]),
        .D2               (paralell_data[1]),
        .D3               (paralell_data[2]),
        .D4               (paralell_data[3]),
        .D5               (paralell_data[4]),
        .D6               (paralell_data[5]),
        .D7               (paralell_data[6]),
        .D8               (paralell_data[7]),
        .SHIFTIN1         (cascade1),
        .SHIFTIN2         (cascade2),
        .SHIFTOUT1        (),
        .SHIFTOUT2        (),
        .OFB              (),
        .T1               (1'b0),
        .T2               (1'b0),
        .T3               (1'b0),
```

```
        .T4             (1'b0),
        .TBYTEIN        (1'b0),
        .TCE            (1'b0),
        .TBYTEOUT       (),
        .TFB            (),
        .TQ             ()
);
OSERDESE2 #(
        .DATA_RATE_OQ   ("DDR"),
        .DATA_RATE_TQ   ("SDR"),
        .DATA_WIDTH     (10),
        .SERDES_MODE    ("SLAVE"),
        .TBYTE_CTL      ("FALSE"),
        .TBYTE_SRC      ("FALSE"),
        .TRISTATE_WIDTH (1)
)
OSERDESE2_Slave (
        .CLK            (serial_clk_5x),
        .CLKDIV         (paralell_clk),
        .RST            (reset),
        .OCE            (1'b1),
        .OQ             (),
        .D1             (1'b0),
        .D2             (1'b0),
        .D3             (paralell_data[8]),
        .D4             (paralell_data[9]),
        .D5             (1'b0),
        .D6             (1'b0),
        .D7             (1'b0),
        .D8             (1'b0),
        .SHIFTIN1       (),
        .SHIFTIN2       (),
```

```
        .SHIFTOUT1    (cascade1),
        .SHIFTOUT2    (cascade2),
        .OFB          (),
        .T1           (1'b0),
        .T2           (1'b0),
        .T3           (1'b0),
        .T4           (1'b0),
        .TBYTEIN      (1'b0),
        .TCE          (1'b0),
        .TBYTEOUT     (),
        .TFB          (),
        .TQ           ()
);
Endmodule
```

6.2 摄像头采集

```
module    ov5640_capture_data(
        input                   rst_n           ,   //复位信号
       //摄像头接口
        input                   cam_pclk        ,   //cam  数据像素时钟
        input                   cam_vsync       ,   //cam  场同步信号
        input                   cam_href        ,   //cam  行同步信号
        input          [7:0]    cam_data        ,   //cam  数据
        output                  cam_rst_n       ,   //cmos  复位信号，低电平有效
      output                    cam_pwdn        ,   //电源休眠模式选择
       //RGB888 接口
        output                  cmos_frame_clk,     //时钟信号
        output                  cmos_frame_ce,      //时钟使能信号
       output                   cmos_frame_vsync,   //帧有效信号
        output                  cmos_frame_href ,   //行有效信号
        output                  cmos_frame_de ,     //数据有效
```

```
    output          [23:0]    cmos_frame_data      //有效数据
);
//parameter define
//寄存器全部配置完成后，先等待 10 帧数据
//待寄存器配置生效后再开始采集图像
localparam   WAIT_FRAME = 4'd10   ;                  //寄存器数据稳定等待的帧
个数
//reg define
reg              rst_n_d0 =1;
reg              rst_n_syn =1;
reg              cam_vsync_d0 ;
reg              cam_vsync_d1 ;
reg              cam_href_d0 ;
reg              cam_href_d1 ;
reg    [3:0]     cmos_ps_cnt ;                       //等待帧数稳定计数器
reg              wait_done ;
reg              byte_flag ;
reg    [7:0]     cam_data_d0 ;
reg    [15:0]    cmos_data_16b ;                     //用于 8 位转 16 位的临时寄存器
reg              byte_flag_d0 ;
//wire define
wire   pos_vsync ;
//采输入场同步信号的上升沿
assign   pos_vsync   = (~cam_vsync_d1) & cam_vsync_d0 ;
//不对摄像头硬件复位,固定高电平
assign   cam_rst_n = 1'b1;
//电源休眠模式选择 0：正常模式 1：电源休眠模式
assign   cam_pwdn   = 1'b0;
assign   cmos_frame_clk     = cam_pclk;
//由于摄像头输入接口的两个时钟周期对应于 RGB888 输出接口的一个有效时钟
周期
//所以要给出数据有效标志即 RGB888 输出接口的时钟使能信号
```

```
assign  cmos_frame_ce     = wait_done  ?  (byte_flag_d0 & cmos_frame_href) || (!cmos_frame_href) : 1'b0;
assign  cmos_frame_vsync = wait_done  ?  cam_vsync_d1  :  1'b0; //输出帧有效信号
assign  cmos_frame_href   = wait_done  ?  cam_href_d1    :  1'b0; //输出行有效信号
assign  cmos_frame_de     = cmos_frame_href ;
assign  cmos_frame_data   = wait_done  ?
    {       cmos_data_16b[15:11],3'd0       ,       cmos_data_16b[10:5],2'd0       , cmos_data_16b[4:0],3'd0 }
    :   24'd0; //输出数据

//复位信号的异步复位、同步释放处理
always @(posedge cam_pclk or negedge rst_n) begin
    if(!rst_n) begin
        rst_n_d0 <= 1'b0;
        rst_n_d0 <= 1'b0;
    end
    else begin
        rst_n_d0   <= 1'b1;
        rst_n_syn <= rst_n_d0;
    end
end
//对行、场同步信号的延迟打拍
always @(posedge cam_pclk or negedge rst_n_syn) begin
    if(!rst_n_syn) begin
        cam_vsync_d0 <= 1'b0;
        cam_vsync_d1 <= 1'b0;

        cam_href_d0 <= 1'b0;
        cam_href_d1 <= 1'b0;
    end
```

```
        else begin
            cam_vsync_d0 <= cam_vsync;
            cam_vsync_d1 <= cam_vsync_d0;

            cam_href_d0 <= cam_href;
            cam_href_d1 <= cam_href_d0;
        end
    end
    //寄存器全部配置完成后，先等待 10 帧数据
    //待寄存器配置生效后再开始采集图像
    always @(posedge cam_pclk or negedge rst_n_syn) begin
        if(!rst_n_syn)
            cmos_ps_cnt <= 4'd0;
        else if(pos_vsync && (cmos_ps_cnt < WAIT_FRAME))
            cmos_ps_cnt <= cmos_ps_cnt + 4'd1;    //对帧数进行计数
    end
    //等待完成后  给出  等待完成信号
    always @(posedge cam_pclk or negedge rst_n_syn) begin
        if(!rst_n_syn)
            wait_done <= 1'b0;
        else if((cmos_ps_cnt == WAIT_FRAME) && pos_vsync)
            wait_done <= 1'b1;
    end
    //8 位数据转 16 位 RGB565 数据
    always @(posedge cam_pclk or negedge rst_n_syn) begin
        if(!rst_n_syn) begin
            cmos_data_16b <= 16'd0;
            cam_data_d0 <= 8'd0;
            byte_flag <= 1'b0;
        end
        else if( cam_href ) begin    //cam  行同步信号
            byte_flag     <= ~byte_flag;
```

```
            cam_data_d0 <= cam_data;
            if(byte_flag)
                cmos_data_16b <= {cam_data_d0,cam_data};
        end
        else begin
            byte_flag <= 1'b0;
            cam_data_d0 <= 8'b0;
        end
end
always @(posedge cam_pclk or negedge rst_n_syn) begin
        if(!rst_n_syn)
            byte_flag_d0 <= 1'b0;
        else
            byte_flag_d0 <= byte_flag;
end
endmodule
```

6.3 千兆网模块

```
module gmii_to_rgmii(
        input                   idelay_clk    ,
        output                  gmii_rx_clk ,
        output                  gmii_rx_dv    ,
        output          [7:0]   gmii_rxd      ,
        output                  gmii_tx_clk ,
        input                   gmii_tx_en    ,
        input           [7:0]   gmii_txd      ,
        input                   rgmii_rxc     ,
        input                   rgmii_rx_ctl,
        input           [3:0]   rgmii_rxd     ,
        output                  rgmii_txc     ,
        output                  rgmii_tx_ctl,
```

```
        output          [3:0]    rgmii_txd
        );
//parameter define
parameter IDELAY_VALUE = 0;
assign gmii_tx_clk = gmii_rx_clk;
rgmii_rx
        #(
          .IDELAY_VALUE    (IDELAY_VALUE)
          )
        u_rgmii_rx(
        .idelay_clk       (idelay_clk),
        .gmii_rx_clk      (gmii_rx_clk),
        .rgmii_rxc        (rgmii_rxc      ),
        .rgmii_rx_ctl     (rgmii_rx_ctl),
        .rgmii_rxd        (rgmii_rxd      ),

        .gmii_rx_dv       (gmii_rx_dv ),
        .gmii_rxd         (gmii_rxd      )
        );
rgmii_tx u_rgmii_tx(
        .gmii_tx_clk      (gmii_tx_clk ),
        .gmii_tx_en       (gmii_tx_en    ),
        .gmii_txd         (gmii_txd       ),

        .rgmii_txc        (rgmii_txc      ),
        .rgmii_tx_ctl     (rgmii_tx_ctl),
        .rgmii_txd        (rgmii_txd      )
        );
Endmodule
……
```