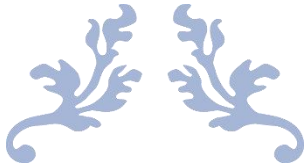

千兆以太网实时传输平台

42075 为 FPGA 出走的那雨夜

2025-11-7

目录

千兆以太网实时传输平台

摘要

本系统采用 FPGA 与 Python 协同的软硬件架构，构建了一套完整的实时人脸检测与识别系统，实现从视频流输入到识别结果输出的高效闭环。系统以 FPGA 开发板为硬件核心，形成 “视频接收→图像预处理→算法处理”的稳定处理链路，在保证实时性的同时，显著提升了识别精度与鲁棒性。

硬件层面，系统充分利用 FPGA 的并行处理能力，完成视频采集与预处理。通过 MIPI-DPHY 接收摄像头数据，经由 CSI-2 协议解包及 RAW10 格式转换，得到标准化图像数据。集成 ISP 模块实现了图像降噪、对比度增强等关键预处理操作，提升图像质量；图像裁剪与校正模块进一步优化画面构图。为满足高分辨率视频的缓存需求，系统采用 DDR3 存储控制器，支持最高 4G 容量，实现 3 帧图像缓存，配合 uiFDMA 模块管理 AXI 总线数据交互。最终，处理后的视频经 HDMI 接口以 1920×1080 分辨率输出。此外，通过 I2C 连接的 uicfgcs500 模块支持按键调节自动曝光参数，适应不同光照条件，确保图像质量稳定。

软件部分包括人脸特征采集与实时识别两大模块。特征采集模块基于 Tkinter 构建用户友好的图形界面，结合 Dlib 人脸检测器，并采用帧跳过策略与分辨率缩放技术，显著降低了计算负荷。该模块设定严格采集条件：仅在画面中出现单一人脸且处于有效范围内时，按“人脸框扩展 1/2”规则截取 ROI 区域，保存至“person_姓名”目录，为后续识别构建高质量特征库。实时识别模块加载特征库后，利用 ResNet 模型提取人脸的 128 维特征向量，通过欧式距离比对身份（阈值≤0.4）。引入质心追踪算法减少约 60%重复计算，支持 5 张人脸同步识别，单帧处理延迟不超过 50 毫秒，最终实时输出带身份标注的视频画面。

综上所述，本系统通过 FPGA 硬件加速与 Python 智能算法的深度融合，实现了高实时性、高精度与高稳定性的嵌入式人脸识别解决方案，为实际应用提供了可靠的技术路径。

第 1 部分　作品概述

1.1 功能与特性

本作品基于安路 PH1A180 FPGA 平台，以低延迟、全链路处理为核心，功能聚焦多场景实时视频应用需求。系统可并行接入多路 MIPI 摄像头信号，依托 FPGA 高速 I/O 实现无丢帧采集，并搭配 DDR3 高速缓存模块有效快速缓解数据瞬时数据存储压力，从源头保障低延迟；借助 FPGA 并行流水线架构，能同步完成灰度化、边缘检测、ROI 提取等图像预处理，高效简化冗余数据、精准突出关键区域，确保全流程低延迟实时处理。

系统支持双路输出模式，通过 HDMI 接口本地实时显示视频，满足现场监控需求；同时通过千兆以太网模块将处理后的数据高速传输至上位机，实现远程同步画面获取。此外，还可联动上位机 GPU 资源，拓展人脸识别、目标检测等智能分析功能。系统具备模块化可裁剪特性，支持按需启闭功能模块，并兼容树莓派等嵌入式设备，适配多种端到端应用场景。

图 1-1　系统成果实物图

1.2 应用领域

本系统基于安路 PH1A180 FPGA 平台，具备多路视频采集、实时处理与双路输出的全链路视频处理能力，能够实现人脸识别、目标检测，可广泛应用于多个行业领域。

在工业视觉领域，系统支持多生产线同步视频质检，通过对设备运行与产品外观的实时处理，有效提升检测效率与一致性。安防监控方面，适用于交通枢纽、智慧园区等场景的多路视频接入，实现低延迟人脸识别与联动预警，有效提升公共场所的安全防控能力。

智慧交通应用中，系统可接入路口多路摄像信号，实时处理车流数据并辅助交通管控决策。商业管理领域，支持商场会员识别、停车场无人值守等场景，实现精准识别与高效运营。

此外，系统具备嵌入式适配能力，可部署于电力巡检、移动医疗辅助等便携设备中，为现场作业提供端到端的智能视频处理支持，展现出广泛的场景适用性与灵活性，为各行业的智能化转型提供了可靠的视频处理解决方案。

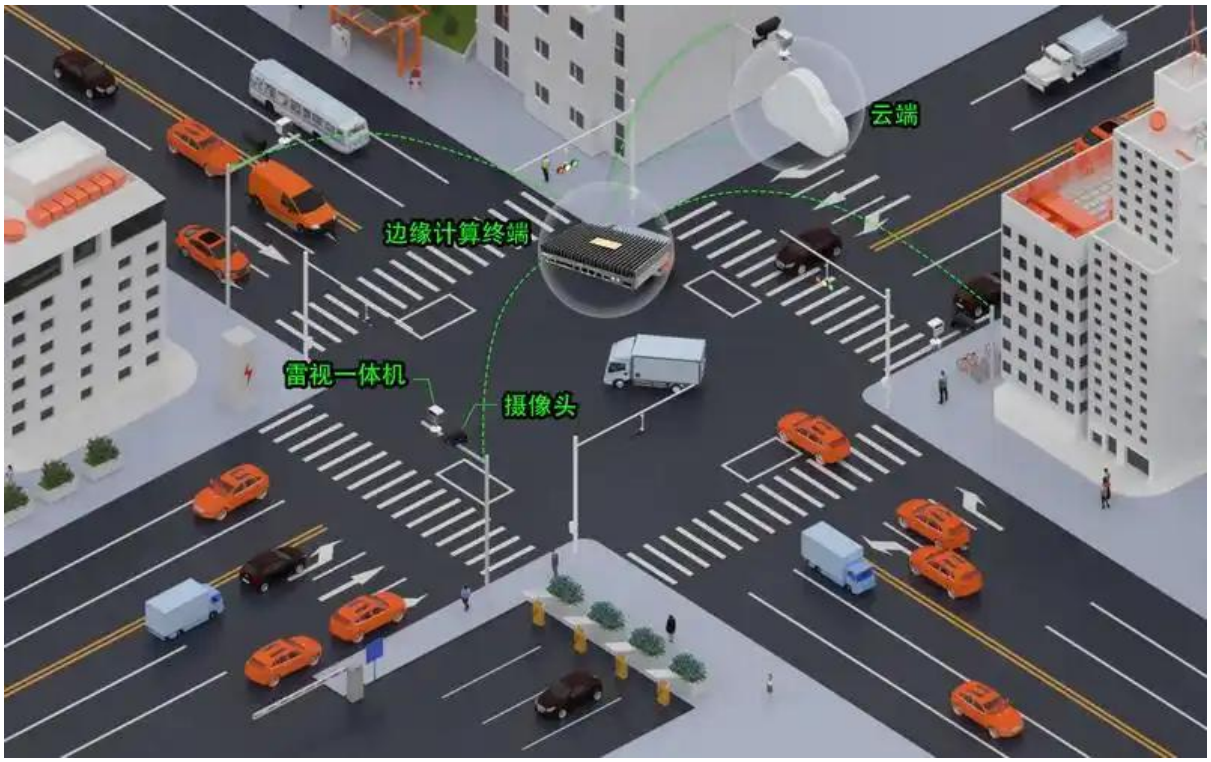

图 1-2 系统应用领域场景

1.3 主要技术特点

本系统基于安路 PH1A180 FPGA 平台，具备高效处理、稳定传输与灵活开发三大技术特点。

在高效处理方面，依托 FPGA 并行流水线架构，实现多路 MIPI 视频信号的同步采集与实时预处理，包括灰度化、边缘检测等操作，结合 DDR3 高速缓存

技术，确保全链路低延迟运行。

数据传输采用基于 UDP 协议的千兆以太网架构，具备优异的实时性与抗干扰能力，满足工业现场对稳定性和速度的双重需求。

开发层面深度融合 OpenCV 与 Python 技术生态：OpenCV 提供高效的图像处理基础库，显著简化系统开发流程；Python 环境支持灵活的算法优化与智能分析功能扩展，包括人脸识别、目标检测等先进视觉任务。系统同时保持模块化设计，支持功能按需裁剪，并可适配各类嵌入式设备，展现出卓越的工程适用性与场景扩展性，为工业视觉、智能安防等领域的创新应用提供了坚实技术基础。

1.4 主要性能指标

表 1-1 系统的性能指标

性能指标	具体参数	优势说明
视频采集能力	单路分辨率最高支持 1080P@30fps	满足高清晰度、高帧率采集需求
全链路处理延迟	从采集到输出总延迟≤50ms	保障实时响应，适配工业质检等时效敏感场景
图像处理吞吐量	并行 2 路 1080P 视频，单路预处理效率≥30fps	支撑多路并发，提升系统处理能力
DP 传输速率	80-100Mbps，丢包率≤0.1%	保障远程画面同步稳定
人脸识别响应速度	联动 GPU 时单帧处理≤50ms，支持多人并行	实现快速身份核验
FPGA 资源占用	逻辑资源≤60%，DDR3 带宽≤70%	预留充足扩展空间
模块化扩展兼容性	支持树莓派等设备，算法集成≤2 小时	降低多场景部署成本

1.5 主要创新点

本系统具备四大核心创新点：

1. **软硬协同架构创新**：采用 FPGA 并行处理结合 UDP 千兆网传输（丢包率≤0.1%），构建“采集-处理-分析”闭环，兼具硬件实时性与软件灵活性。

2. **全链路低延迟优化**：实现 MIPI 采集至 HDMI/以太网双路输出的全链路延迟≤50ms，支持内容按需定制与 10ms 内信息同步。

3. **人脸识别协同处理**：通过 FPGA 预提取人脸 ROI，结合 UDP 高速传输与 OpenCV/Python 算法，实现≤50ms 的端到端人脸识别。

4. **模块化扩展设计**：支持多摄像头接入与树莓派兼容，提供 Python 开发接口，显著提升系统适用性与开发效率。

1.6 设计流程

系统设计流程分为四个关键阶段：首先完成以 FPGA 为核心的硬件平台搭建，集成 MIPI 摄像头、DDR3 缓存及 HDMI 与千兆以太网接口；随后基于 Python 与 OpenCV 开发图像处理与识别算法，配置 UDP 高速传输协议；接着实现软硬件协同，构建"采集-处理-传输-识别-显示"全链路闭环，确保多路输出同步；最终通过系统测试优化性能指标，验证模块化扩展能力，确保系统适配满足工业、安防等多场景应用需求。

第 2 部分　系统组成及功能说明

2.1 整体介绍

给出系统整体框图，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。

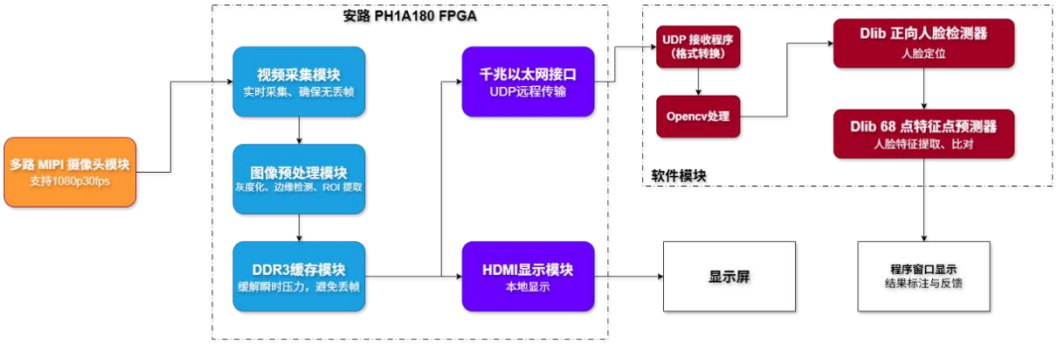

图 2-1　系统整体框图

2.2 硬件系统介绍

2.2.1　硬件整体介绍

硬件总体架构如图 2-2-1 所示。硬件系统由图像采集模块（MIPI 摄像头接口）、图像缓存与调度模块（DDR3 存储器）、视频显示模块（HDMI 输出）、以太网通信模块（UDP 传输） 及 主控处理单元（FPGA 主逻辑） 构成。通过模块化设计，系统能够灵活支持多种图像输入与输出接口形式，满足不同视觉应用的需求。

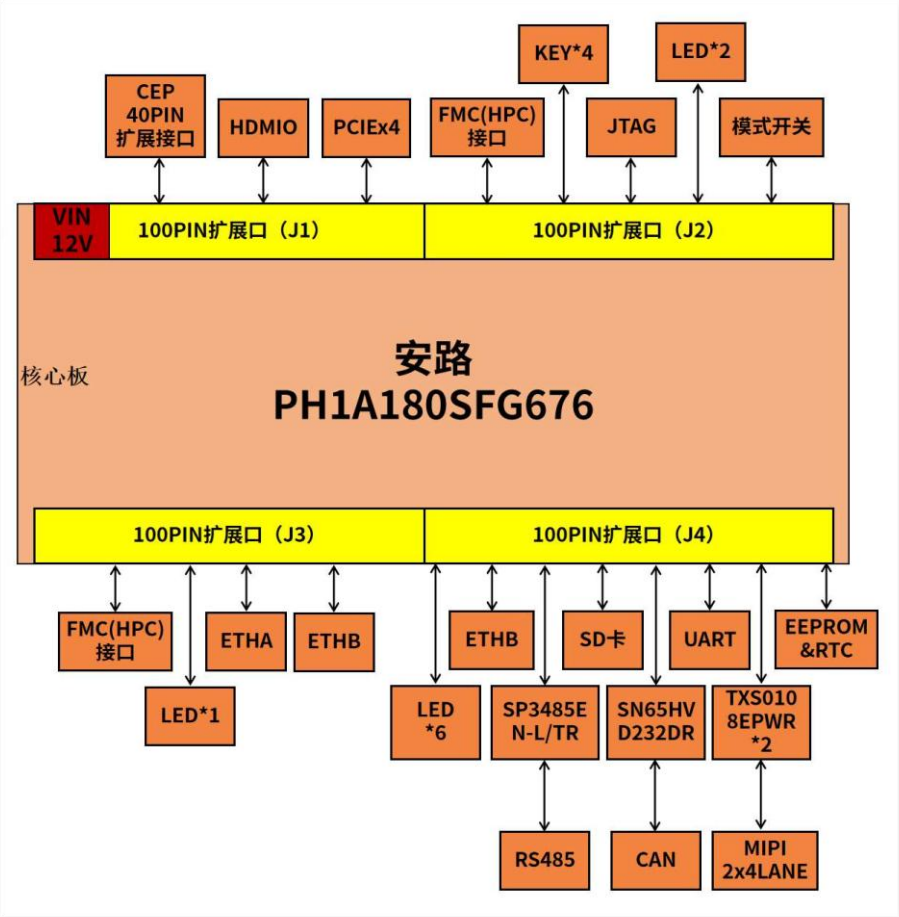

图 2-2-1 系统整体结构图

整个系统的硬件平台基于安路 PH1A180 FPGA 芯片，其具备丰富的高速 I/O 接口和强大的逻辑资源。结合外部 DDR3 存储器、FLASH、MIPI 摄像头、HDMI 显示器和千兆以太网口，实现了从图像采集到实时显示、网络传输的闭环数据通道，为嵌入式 AI 图像识别提供了坚实的硬件基础

表 2-1 系统主要硬件组成

硬件模块	型号/规格	主要功能
FPGA 核心板	PH1A180SFG676	系统主控，负责图像数据接收、ISP 处理及系统调度
DDR3 存储器	1GB	提供图像帧缓存与高速数据缓冲
MIPI 摄像头模块	-	实时采集原始图像数据

硬件模块	型号/规格	主要功能
以太网接口	RGMII/千兆	实现处理数据的高速网络发送
HDMI 显示接口	-	用于本地图像实时显示与调试
电源与时钟模块	-	为系统各单元提供稳定电源与时钟信号
调试接口	JTAG、UART	用于程序下载、系统调试与数据监控

2.2.2 电路各模块介绍

系统的电路模块主要包括：

摄像头接口模块、DDR3 存储模块、以太网传输模块、时钟与电源模块、调试与扩展接口模块。

以下对各部分进行详细说明：

1. 摄像头接口模块（MIPI）

摄像头接口模块用于接收来自 MIPI 摄像头的高速串行视频信号。

MIPI（Mobile Industry Processor Interface）是一种广泛应用于移动端图像传感器的高速串行通信标准，采用差分信号传输，具有低功耗和高带宽的特性。

在本系统中，MIPI 模块主要完成以下任务：

1) 物理层接收（由 mipi_dphy_rx_ph1a_mipiio_wrapper 模块实现）

- 接收来自摄像头的高速差分信号（4 Lane + 时钟 Lane）；
- 通过 MIPI D-PHY 接收器将高速串行信号解串为字节并行数据流；
- 输出像素时钟 O_hs_rx_clk、S_hs_rx_valid、O_hs_rx_valid 以及视频数据 O_hs_rx_data。

该模块是摄像头与 FPGA 之间的高速接口桥梁，保证视频数据能够以稳定的时钟节奏进入后级逻辑。

2) 协议层解包（由 csi_unpacket_4lane 模块实现）

该模块负责解析摄像头传输的 CSI-2（Camera Serial Interface 2）协议帧结构。

其功能包括：

- 解析 CSI-2（Camera Serial Interface 2）协议；
- 从数据包中提取帧同步信号（frame_start / frame_end）；
- 输出解包后的有效像素数据流 S_csi_data。

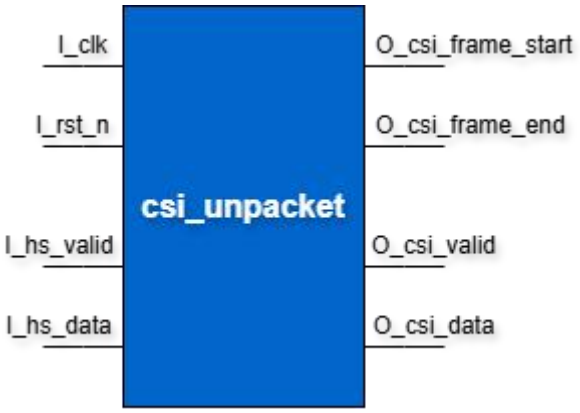

图 2-2-2 csi_unpacket 模块接口框图

3) 图像格式解码（由 raw10_unpacket_4lane 模块实现）

由于摄像头输出的数据多采用 RAW10 格式（即 10bit 深度的原始像素数据，4 个像素共 5 字节编码），因此需要对其进行字节重组与位宽扩展。

该模块的主要功能包括：

- 将 CSI-2 输出的 RAW10 格式数据进行还原；
- 输出完整的 40bit 像素数据流 S_raw10_data；

提供帧同步与行同步信号，便于后续图像缓存与 ISP（图像信号处理）模块处理。

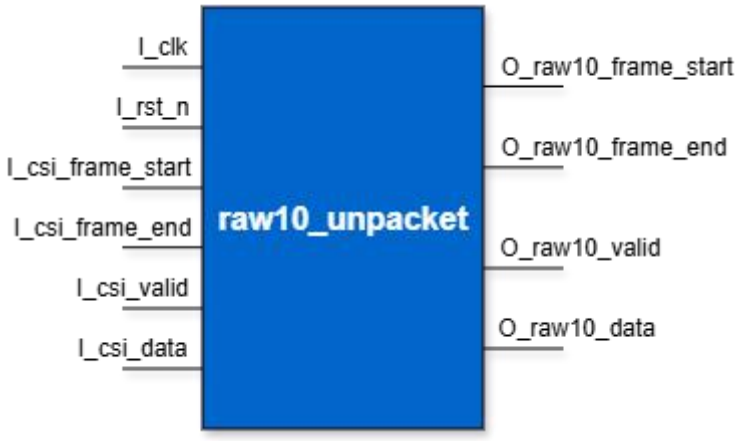

图 2-2-3 raw10_unpacket 模块接口框图

2. ISP 图像处理模块

ISP（Image Signal Processing，图像信号处理）模块是本系统的核心数据处理单元之一，负责将来自 MIPI 摄像头的 RAW10 原始图像数据 转换为标准的 RGB 图像数据，为后续 HDMI 显示输出 和 AI 识别算法模块 提供高质量的输入。

该模块在整个视频处理链中起到承上启下的作用：前端接收来自摄像头的未经处理的传感器数据，后端输出经过优化、视觉质量更高的彩色图像数据流。

ISP 模块的主要任务是实现图像的还原、增强与格式转换。由于摄像头输出的 RAW 图像数据仅包含单通道的亮度与颜色滤波阵列（Bayer Pattern）信息，不能直接用于显示，因此需要经过多级处理，包括黑电平校正、插值、白平衡、亮度与对比度调整、伽马矫正等步骤。

这些算法模块在 FPGA 内部以硬件流水线方式实现，具有实时性强、延迟低、并行度高等优点，可在高清视频帧率下稳定运行。

ISP 模块主要由以下子模块构成，各模块间采用标准化数据流接口（AXI-Stream）连接，形成完整的图像处理链路：

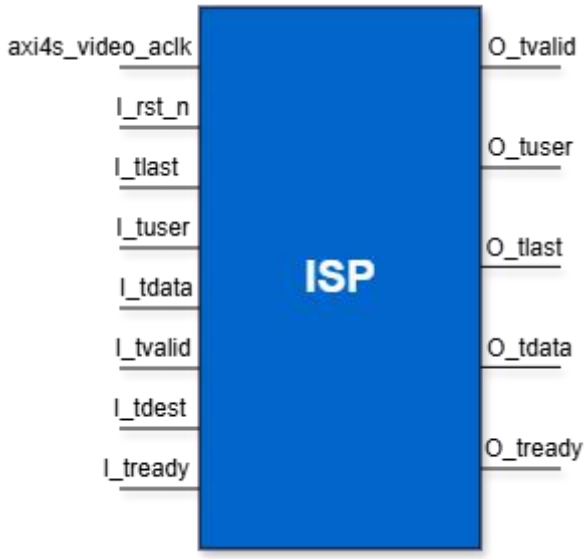

图 2-2-4 isp 模块接口框图

表 2-2 ISP 子模块与功能

模块名称	功能说明
BLC（Black Level Correction）	黑电平校正，消除传感器暗电流造成的偏差。
raw_matrix_3x3_buffer	建立 3×3 像素矩阵，为插值算法提供邻域数据。
bilinear_interpolation	双线性插值，将 Bayer 图像插值为 RGB 图像。
AWB（Auto White Balance）	自动白平衡调整，使图像色彩更自然。
image_cut	图像裁剪，去除无效边缘区域。

gamma	伽马校正，改善亮度层次感。
Saturation_adj	饱和度调节。
contrast_adj	对比度调整。
Brightness_adjustment	亮度调整。
data96_128	数据宽度转换模块（RGB96bit → AXI Stream 128bit）。

3. DDR3 存储模块

本系统的 DDR3 存储子系统由两片 512MB DDR3 SDRAM 构成（总容量 1GB），数据总线宽度为 32bit，工作带宽最高可达 1600 Mbps。DDR3 模块通过 FPGA 内部的 AXI 总线实现高速数据读写，作为系统的数据缓存与同步缓冲中心，为摄像头采集、ISP 处理、HDMI 显示以及以太网传输提供高带宽的数据通路支持。

DDR3 控制系统主要由以下三个核心子模块组成：

表 2-3 DDR3 子模块与功能

模块名称	功能说明
ddr_ip（PHY 层控制器）	提供 DDR3 物理层驱动、时钟校准与命令译码功能，负责完成数据的物理收发。
uiFDMA（AXI 总线 DMA 控制器）	在 AXI 总线与用户逻辑之间建立高效数据搬运通路，实现 AXI 读写请求的自动化控制。
uidbuf（双缓冲图像缓存控制）	管理写入与读取缓存区，实现帧级数据的 ping-pong 缓存机制，提高读写并行性与系统吞吐率。

工作原理

i. **数据写入路径（摄像头→DDR3）**

- 摄像头经 MIPI 接口采集图像 → 经过 ISP 模块处理后输出 RGB96 位视频流（ISP_O_tdata、ISP_O_tvalid）；
- uidbuf 写入通道将视频帧划分为 AXI 传输块，并通过 uiFDMA 发起 AXI 写请求；
- DDR 控制器完成 AXI 总线的突发写入操作，将帧数据存储至物

理 DDR3 地址空间；

- 写缓存状态由 wbuf_sync 同步信号指示，确保多帧连续写入时的缓冲切换。

ii. **数据读取路径（DDR3→显示/传输）**

- HDMI 显示或 UDP 发送模块通过 fdma_raddr 发起读请求；
- uiFDMA 控制 AXI 读通道自动执行帧数据读取；
- 通过 uidbuf 读通道输出到视频处理模块（如 vtc 视频时序控制器）；
- rbuf_sync 信号同步显示刷新节奏，确保帧切换无撕裂、无延迟。

iii. **AXI 总线接口信号**

DDR3 模块的主控制接口符合 AXI4 协议规范，主要信号包括：

- 地址通道（AW/AR）：axi_awaddr, axi_araddr，用于指定 DDR3 地址；
- 数据通道（W/R）：axi_wdata, axi_rdata，用于图像数据传输；
- 握手信号：axi_awvalid, axi_awready, axi_wvalid, axi_wready 等用于流控制；
- 响应信号：axi_bresp, axi_rresp，保证数据一致性与错误检测。

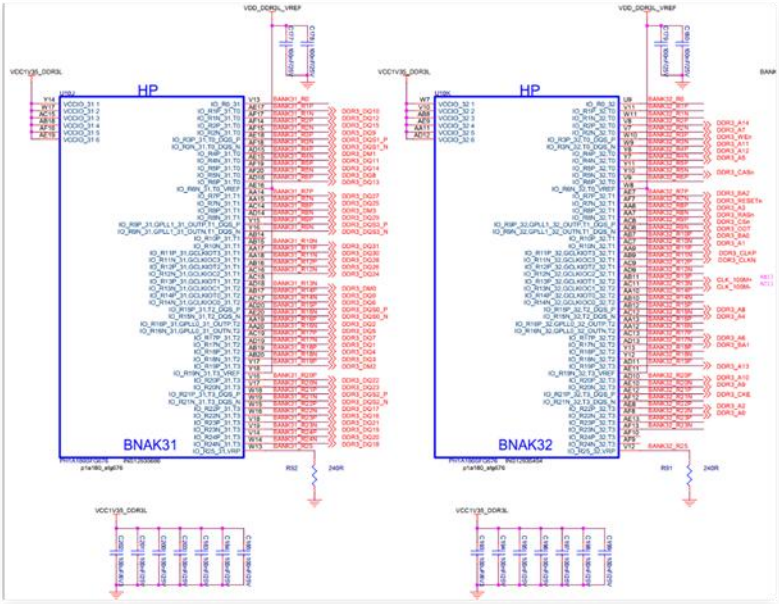

图 2-2-5　DDR3 内存颗粒硬件连接原理图

4. UDP 传输模块

UDP 网络传输模块主要负责将 FPGA 内部处理后的图像数据通过以太网实时发送到上位机，实现视频数据的高速传输。本模块基于 UDP/IP 协议栈 实现，能够支持千兆以太网数据链路，保证图像流的稳定传输和低延迟通信

该模块的主要功能包括：

- **数据封装与打包**：将 ISP 模块输出的像素数据打包为 UDP 数据包；
- **协议栈控制**：按照 UDP/IP 层级结构生成头部信息，包括帧头、端口号、源/目的 IP 等；
- **状态机控制**：通过多状态有限状态机（FSM）管理帧同步、头部发送、数据传输及结束信号；
- **以太物理层接口（RGMII）**：与外部 PHY 芯片通讯，实现千兆以太网帧的发送与接收。

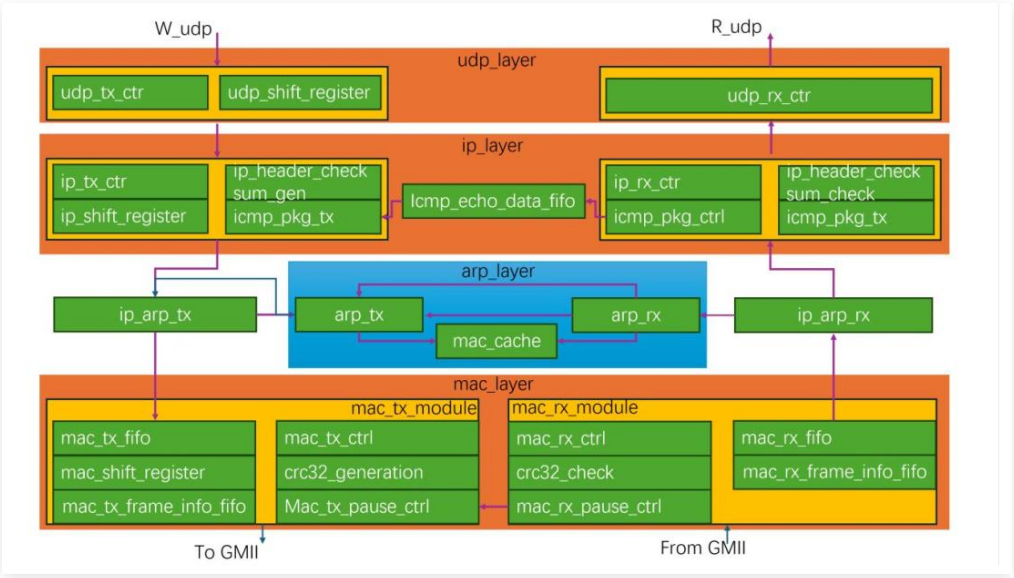

图 2-6 UDP 传输模块结构

模块内部定义了 8 个主要状态，控制完整的数据发送流程。

表 2-3 UDP 发送状态机设计

状态名	功能描述
S_SYNC_1	帧同步初始化，检测垂直同步信号
S_SYNC_2	等待帧同步信号结束
S_SYNC_3	进入 UDP 等待阶段准备发送
S_UDP_WAIT	等待 UDP 通道空闲且缓存数据准备就绪
S_UDP_ACK	向 UDP 控制器发出发送请求
S_UDP_HEADER	发送图像帧头信息（帧序号、尺寸、标识符等）
S_UDP_DATA	发送图像帧数据内容

数据流过程如下：

- **帧同步触发：**

  视频时序控制模块 uivtc 产生帧同步信号 vtc_vs，每当检测到新帧开始时，状态机复位并准备新一帧的数据发送。

- **数据写入缓存：**

  由 fdma_O_R_tdata_24 提供的图像数据写入 udp_pkg_buf FIFO 中，实现跨时钟域缓存（写时钟 vtc_clk，读时钟 clk_125）。

- **UDP 状态机控制发送流程：**

  状态机依次执行帧同步、UDP 请求、包头发送、数据分片、帧结束等步骤，控制信号 W_udp_req、W_udp_valid 驱动 uiudp_stack 发送。

- **协议封装与输出：**

  UDP 协议栈 uiudp_stack 生成完整的以太网帧，包括 MAC、IP、UDP 层头部及图像负载，并通过 gmii_tdata 送入 rgmii_interface。

- **物理层发送：**

  rgmii_interface 模块将 GMII 格式转换为 RGMII 四倍速差分信号，通过千兆 PHY 芯片最终输出至网络。

图 2-2-7 UDP **数据段格式**

5. HDMI 视频显示模块

HDMI 视频显示模块负责将 FPGA 内部处理后的视频图像实时输出到外部显示设备。该模块通过 HDMI 发送接口实现高清视频信号输出，支持 1080P@60Hz 分辨率，图像输出清晰稳定。

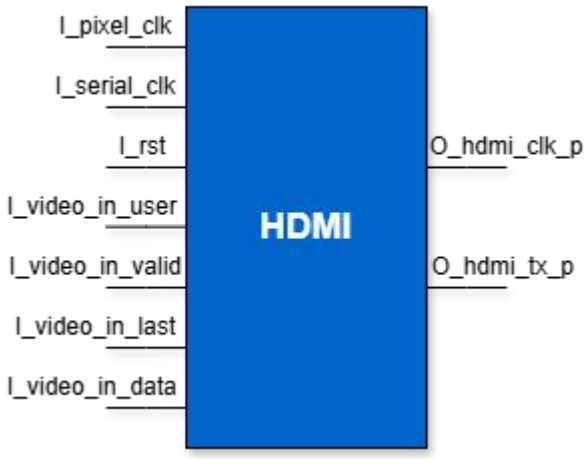

图 2-2-8 hdmi 模块接口框图

HDMI 模块的主要功能是：

- 将来自 ISP 或 DDR 缓存的图像数据格式化为符合 HDMI 标准的

RGB 视频流；

- 依据视频时序信号（行同步、场同步、数据有效区）生成标准视频帧；
- 通过 TMDS（Transition Minimized Differential Signaling）信号对视频进行差分编码输出；
- 向显示设备输出同步的时钟与像素数据，实现高质量图像显示。

2.3 软件系统介绍

2.3.1 软件整体介绍

软件系统采用“功能模块化”设计，围绕“视频数据接收 - 人脸智能处理”核心流程，划分为三大功能模块，各模块通过标准化接口（图像数据、特征向量）实现数据流转，同时保持功能独立性，可按需启用或裁剪。整体架构遵循“数据输入→处理→输出→分析”的逻辑链路，技术栈融合 C++ 的高效性与 Python 的灵活性，确保系统在低延迟与智能分析间实现平衡。

2.3.2 软件各模块介绍

1. FPGA 视频接收与预处理模块

作为系统的数据入口，该模块基于 C++ 实现 FPGA 端视频流的低延迟接收与预处理。通过 Winsock2 库构建 UDP 通信链路，绑定本地 6001 端口，配置 16MB 接收缓冲区以适配千兆以太网环境下的高速传输（1080P@30fps 视频流，速率稳定在 80-100Mbps）。通信过程中，采用 16 字节自定义 PacketHeader 结构体（包含图像标识 0xAA0055FF、宽高参数、帧序号等）进行数据包校验，确保接收数据的有效性。

为平衡实时性与稳定性，模块采用多线程协同架构：接收线程专注于数据包抓取与初步校验，每秒可处理超过 1000 个数据包；组装线程每 50ms 检查一次缓冲区，通过帧序号回绕机制（0~4319 循环）识别图像边界，支持两种重组策略 —— 完整帧优先组装与超时 200ms 强制组装（基于首尾帧判断）；显示线程调用 OpenCV 完成图像格式转换（BGR 转 RGB）与缩放（1920×1080 至 960×540），叠加图像 ID、帧计数等统计信息后实时渲染。线程间通过互斥锁保障缓冲区操作安全，全链路处理延迟控制在 50ms 内，最终输出标准化的 OpenCV Mat 格式图像，为后续模块提供统一输入。

模块以 run()为总入口，启动 start_receiver()初始化网络资源并创建 udp_receive_loop（接收线程）和 assembly_check_loop（组装线程），同时通过 display_processing_loop（显示线程）完成图像渲染。三大线程通过共享缓冲区（image_buffers）和显示队列（display_queue）协同工作，关键函数调用关系如下：

run() → start_receiver() → [udp_receive_loop() ↔ assembly_check_loop()] → assemble_and_queue_image() → display_processing_loop()

表 2-4　FPGA 视频接受与预处理模块函数

函数名	功能描述	关键输入变量	关键输出变量	处理流程概述
start_receiver()	初始化 UDP 套接字并启动接收 / 组装线程	local_host（本地 IP）、local_port（端口 6001）、image_width/height（1920×1080）	初始化的 udp_socket、启动的后台线程	创建套接字→设置 16MB 缓冲区→绑定端口→启动 udp_receive_loop 和 assembly_check_loop 线程
udp_receive_loop()	持续接收 UDP 数据包并触发解析	udp_socket（套接字）、recv_buffer（接收缓冲区，65535 字节）	解析后的 PacketHeader（包头）、payload_data（帧数据）、total_packets_received（累计包数）	循环调用 recvfrom 接收数据 → 每 1000 包触发 report_status()→ 调用 process_packet()解析数据包
process_packet()	校验数据包并提取帧数据，检测图像边界	原始数据包 data、包长度 length	验证后的 header（包头信息）、更新的 image_buffers（帧缓存）	解析包头→验证标识与宽高 → 提取载荷 → 调用 detect_image_boundary()判断图像边界→handle_image_frame()存储帧
assembly_check_loop()	定时检查缓冲区，触发图像组装	image_buffers（帧缓存）、frames_per_image（每图帧数 4320）	待组装图像 ID 列表、触发 assemble_and_queue_image()	每 50ms 遍历缓冲区→判断组装条件（完整帧 / 超时 200ms 且含首尾帧）→标记待组装图像并移除缓冲区
assemble_and_queue_image()	将帧数据拼接为完整图像并入队显示	img_id（图像 ID）、buf（对应帧缓存）	完整图像数据 assembled、display_queue（显示队列）	按帧序号拼接数据→封装为DisplayData→存入显示队列 → 更新 total_images_assembled（累计组装数）
display_processing_loop(	从队列获取图像并渲染，响应按键操作	display_queue（显示队列）、total_packets_recei	显示窗口中的图像（含叠加信息）	循环检查队列→取出图像→转换为 RGB 并缩放→叠加图像 ID / 帧数信息

)	ved（统计信息）	→调用 cv::imshow()显示 →响应 ESC/s 键

2．人脸特征采集模块

该模块基于 Python 构建，是连接用户交互与特征库构建的核心环节，通过可视化界面实现人脸样本的规范化采集、存储与管理，为实时人脸识别模块提供高质量的特征训练数据。其设计兼顾易用性与数据规范性，既支持本地摄像头输入，也可直接接收 FPGA 视频接收预处理模块输出的视频流，形成灵活的多源数据接入能力。

界面采用 Tkinter 构建直观交互环境，整体布局分为三大功能区域：左侧为 640×480 实时视频显示区，通过白色矩形框动态标记检测到的人脸 ROI（感兴趣区域），当人脸超出摄像头有效采集范围时，边框自动变为红色并提示“人脸超出采集区域”；右侧上半部分为操作区，设置“录入姓名”“保存人脸”“删除全部数据”三个核心按钮，支持用户通过输入框完成姓名信息录入，点击“保存人脸”时触发样本存储逻辑；右侧下半部分为状态区，实时显示系统运行参数（如当前 FPS≥25fps、数据库已存人脸数量）与操作日志（如“样本保存成功”“检测到多个人脸，请调整位置”等），通过颜色编码区分信息类型（成功信息为绿色，警告信息为红色）。

技术实现上，模块以 Dlib 正向人脸检测器为核心，结合双重性能优化策略提升实时性：一是帧跳过机制，每 3 帧处理 1 帧图像，减少冗余计算；二是分辨率动态缩放，检测阶段将图像缩小至原尺寸的 0.5 倍（320×240），降低检测器算力消耗，同时保证检测精度。采集逻辑设置严格限制条件，仅当满足“单人脸存在”且“人脸完整处于摄像头画面中心区域（占比≥70%）”两个条件时，才允许执行样本保存操作，避免无效数据录入。

样本存储采用标准化目录结构，生成所有样本统一存放于“data/data_faces_from_camera”根目录下，按“person_姓名”创建子文件夹分类管理，文件命名遵循“img_face_序号.jpg”规则（如“person_张三/img_face_0.jpg”），便于后续批量特征提取。ROI 区域裁剪采用“人脸框扩展 1/2 范围”策略，即在检测到的人脸边界框基础上，向四周扩展 50% 边长，确保包含完整的面部特征（如额头、下颌线等），裁剪后图像统一压缩为 300×300 像素，为特征提取提供标准化输入。

模块还集成完善的样本管理功能：支持 “删除全部数据”（一键清空根目录并重置计数器）与“单样本删除”（通过文件浏览器手动移除指定姓名子目录），操作后自动更新状态区的“已存人脸数” 统计，确保数据计数与实际存储一致。此外，系统启动时会自动扫描存储目录，加载已存样本信息并初始化特征库计数器，实现样本数据的无缝衔接。

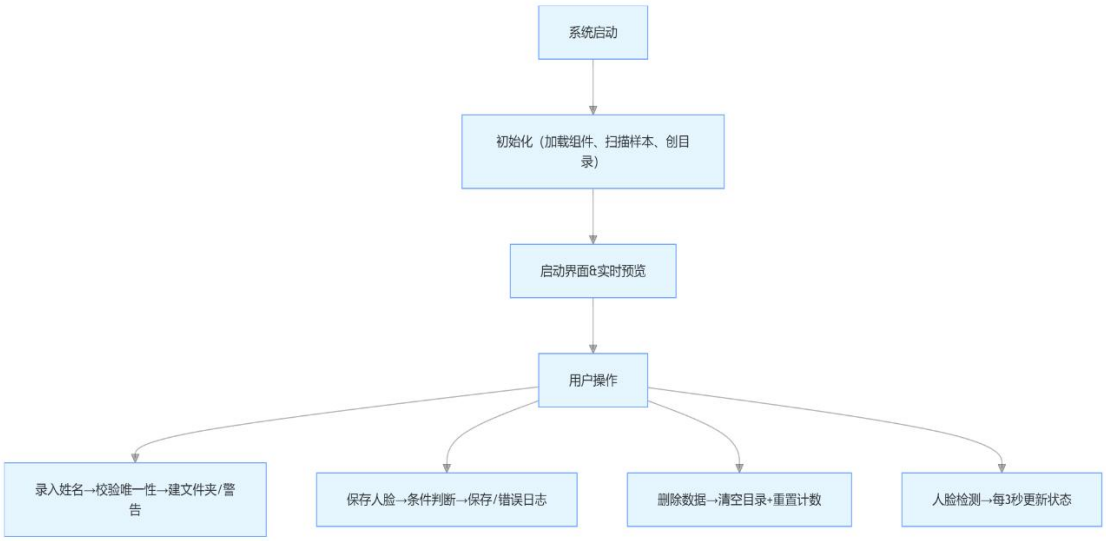

图 2-8 人脸特征采集流程

表 2- 5 人脸特征录入函数注解

函数名	功能描述	流程图核心步骤	关键输入变量	关键输出变量
__init__()	初始化摄像头、检测器、存储路径等核心资源	1. 配置摄像头分辨率（640×480） 2.加载 Dlib 正向检测器 3. 创建存储根目录（data/data_faces_from_camera） 4.扫描已存样本并初始化计数器	摄像头索引（camera_idx=0）、存储根目录路径（root_path）	初始化的摄像头对象（cap）、Dlib 检测器（detector）、已存姓名列表（saved_names）、样本计数器（total_samples）
start()	启动 GUI 界面与视频处理线程，协调模块整	1.调用__init__()完成初始化 2.启动 Tkinter 界面线程（含视频区、操作区） 3.创建视频处理线程，循环调	初始化后的 cap、detector、saved_names	运行中的 GUI 窗口、视频处理线程、界面状态标签（如 FPS、已存人

	体运行	用 frame_process() 4.监听窗口关闭事件，释放资源		脸数）
frame_process()	实时处理视频帧，结合帧优化策略检测人脸并更新显示	1.调用 get_frame()获取摄像头帧 2.帧计数器控制（每 3 帧处理 1 帧） 3.缩小帧至 0.5 倍分辨率，调用 Dlib 检测人脸 4.绘制人脸框（白色 = 有效，红色 = 超出范围） 5.计算 FPS 并更新界面	摄像头帧（current_frame）、帧计数器（frame_cnt）、Dlib 检测器（detector）	带人脸框的显示帧（display_frame）、当前帧人脸数（face_count）、实时 FPS（current_fps）
save_face_sample()	校验人脸状态并保存样本，执行 ROI 裁剪与目录存储	1.接收“保存”按钮触发信号 2.校验前置条件（已录入姓名、单人脸、范围内） 3.按“人脸框扩展 1/2”裁剪 ROI 4.保存至 person_姓名目录（命名规则：img_face_序号.jpg） 5.更新计数器与操作日志	当前帧（current_frame）、人脸位置（face_rect）、当前姓名（input_name）、样本计数器（sample_cnt）	保存的样本图像（img_face_*.jpg）、更新的 sample_cnt、操作日志（log_msg）
get_frame()	读取摄像头帧并转换格式，处理设备异常	1.检查摄像头状态，失败则重新初始化 2.调用 cap.read()读取帧 3.将 BGR 格式转为 RGB（适配 Dlib 输入） 4.返回帧数据与有效性标识	摄像头对象（cap）	布尔值（success，帧有效性）、RGB 格式帧（rgb_frame）
create_name_dir()	校验姓名唯一性并创建对应存储目录	1.获取用户输入姓名 2.校验非空性与唯一性（对比 saved_names） 3.创建目录-data/data_faces_from_camera/person_姓名 4.更新 saved_names 与界面计数标签	用户输入姓名（input_name）、已存姓名列表（saved_names）	创建的目录路径（name_dir）、更新的 saved_names、界面“已存人脸数”标签

3．实时人脸识别模块（Python）

该模块以视频流为输入，构建“检测 - 特征提取 - 比对 - 追踪”的全流程处理机制，实现对画面中人脸的实时身份核验。其核心逻辑围绕特征库匹配与多目标追踪展开，在保证识别精度的同时，通过算法优化将单帧处理延迟控制在

50ms 内，支持最多 5 张人脸并行识别，最终输出带身份标注的视频流，完成从原始数据到智能分析的闭环。

模块启动后首先加载 features_all.csv 特征库，该文件存储已知人脸的姓名及对应的 128D 特征向量，作为身份判定的基准。处理流程以单帧图像为单位：第一步通过 Dlib 正向人脸检测器定位画面中的人脸区域，返回矩形坐标（x1,y1,x2,y2）；第二步调用 68 点特征点预测器提取面部轮廓关键点，为后续特征生成提供空间参考；第三步基于 ResNet 模型将人脸区域转换为 128D 特征向量 —— 该向量具有尺度与姿态不变性，可稳定表征人脸特征，且经过 L2 归一化处理以增强比对稳定性。

身份判定采用欧式距离比对法：计算待识别向量与特征库中所有向量的欧式距离，取最小值作为相似度度量。当最小值≤0.4 时，判定为匹配并输出对应姓名；否则标记为 “unknown”。为优化多目标场景下的效率，模块引入质心追踪算法：通过计算连续帧中人脸的质心坐标（矩形框中心(x_center,y_center)），用欧式距离匹配同一目标（阈值≤50 像素），减少约 60% 的重复特征计算；当某目标累计 10 帧未匹配时，触发重识别机制，平衡实时性与长期追踪的准确性。

输出环节通过 OpenCV 在原始帧上叠加标注信息：已知人脸用绿色矩形框标记，未知人脸用红色框标记，框上方显示姓名及匹配距离（保留 2 位小数）；画面左上角实时显示人脸数量与帧率（≥20fps）。同时支持按键交互（按 “s” 保存当前结果，按“q” 退出），兼顾调试与实用需求。

针对边界场景，模块做了针对性处理：对遮挡人脸（如口罩遮挡），通过特征点完整性判断（至少包含双眼与额头）决定是否提取特征；对光线变化，在 ROI 裁剪后执行灰度归一化，减少光照干扰。该模块可无缝对接 FPGA 视频接收模块或本地摄像头，通过标准化接口与特征库联动，既支持独立运行，也可嵌入完整系统实现端到端识别。

表 2-6 人脸识别函数注解

函数名	功能描述	关键流程步骤	关键输入变量	关键输出变量
__init__()	初始化模型、特征库及追踪状态变量	1.加载 Dlib 检测器、68 点预测器、ResNet 模型 2.调用 load_feature_db()解析 features_all.csv 为	特征库路径（features_path）、模型路径(detector_path	检测器（detector）、特征库（known_names/kn

		known_names 与 known_features 3.初始化追踪变量（上一帧质心、姓名、重识别计数器）	等）	own_features）、追踪状态（ last_centroids 等）
process_stream()	视频流处理主函数，循环执行单帧识别	1.读取视频帧 2.调用detect_faces()定位人脸 3.调用 track_centroids()关联目标 4.新目标调用 extract_and_match()判定身份 5.绘制标注并输出帧	视频流帧（frame）、初始化后的模型与特征库	带标注帧（ annotated_frame）、识别结果列表（[姓名 1, 姓名 2, ...]）
detect_faces()	检测人脸区域并提取特征点	1 帧转灰度图→2. Dlib 检测器定位人脸区域（face_rects）→3. 特征点预测器提取 68 点坐标（shapes）	输入帧（frame）、检测器（detector）、预测器（predictor）	人脸区域列表（face_rects）、特征点列表（shapes）
track_centroids()	质心追踪关联连续帧目标，减少重复计算	1.计算当前帧人脸质心 2.与上一帧质心算距离 3.匹配最小距离目标（≤50 像素） 4.未匹配目标标记为新目标，更新重识别计数器	当前帧人脸区域、上一帧质心与姓名（ last_centroids/last_names）	关联后姓名列表（ tracked_names ）、新目标索引（ new_indices ）、当前质心（ current_centroids）
extract_and_match()	提取 128D 特征并比对身份	1.基于特征点裁剪对齐 ROI 2 ResNet 模型生成 128D 向量 3.与特征库算欧式距离 4.按阈值（≤0.4）返回姓名或“unknown”	人脸 ROI、特征点（shape）、ResNet 模型、特征库（ known_features/known_names）	匹配结果（name）、最小距离（min_dist）
load_feature_db()	加载 features_all.csv 特征库	1.检查文件存在性 2.逐行解析姓名与 128 个特征值 3.特征值转 numpy 数组存入 known_features	特征库路径（ features_path ）	姓名列表（ known_names ）、128D 特征列表（ known_features ）

第 3 部分　完成情况及性能参数

3.1 整体介绍

本系统以 FPGA 开发板为硬件核心，构建“视频接收 - 预处理 - 算法处理” 的完整链路，实现高效人脸检测与识别。

硬件端依托 FPGA 完成视频采集与预处理：通过 MIPI-DPHY 接收摄像头数据，经 CSI 协议解包、RAW10 格式转换为标准化图像；集成 ISP 模块实现降噪、对比度增强等预处理，并通过图像裁剪校正模块优化画面；采用 DDR3 存储控制器（支持最大 4G 容量）实现 3 帧缓存，配合 uiFDMA 模块完成 AXI 总线数据交互，最终经 HDMI 接口输出 1920×1080 分辨率视频流。硬件同时支持摄像头参数调节，通过 I2C 接口连接的 uicfgcs500 模块，可接收按键指令配置自动曝光（AE）参数，提升复杂光照下的图像质量。

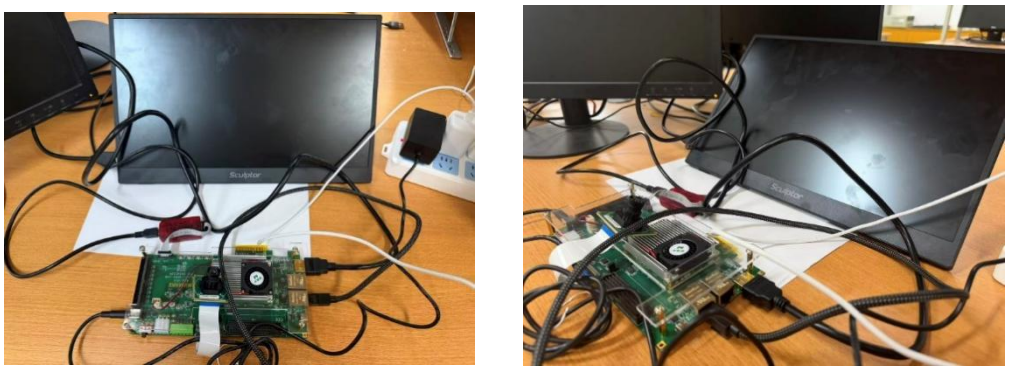

图 3-1 系统正面全局图与 45°实物图

3.2 工程成果

3.2.1 电路成果

硬件端依托 FPGA 完成视频采集与预处理：通过 MIPI-DPHY 接收摄像头数据，经 CSI 协议解包、RAW10 格式转换为标准化图像；集成 ISP 模块实现降噪、对比度增强等预处理，并通过图像裁剪校正模块优化画面；采用 DDR3 存储控制器（支持最大 4G 容量）实现 3 帧缓存，配合 uiFDMA 模块完成 AXI 总线数据交互，最终经 HDMI 接口输出 1920×1080 分辨率视频流。硬件同时支持摄像头参数调节，通过 I2C 接口连接的 uicfgcs500 模块，可接收按键指令配置自动曝光（AE）参数，提升复杂光照下的图像质量。

图 3-2 系统核心电路图

3.2.2 软件成果

软件端分两大模块协同工作：人脸特征采集模块基于 Tkinter 构建可视化界面，结合 Dlib 检测器，通过帧跳过策略（每 3 帧处理 1 帧）与分辨率缩放优化性能，仅当单人脸处于有效范围时，按 “人脸框扩展 1/2” 规则裁剪 ROI，存储至“person_姓名” 目录；实时人脸识别模块加载特征库（features_all.csv），通过 ResNet 模型生成 128D 特征向量，采用欧式距离比对（阈值≤0.4）判定身份，引入质心追踪算法减少 60% 重复计算，支持 5 张人脸同时识别，单帧延迟≤50ms，最终输出带标注的实时画面。

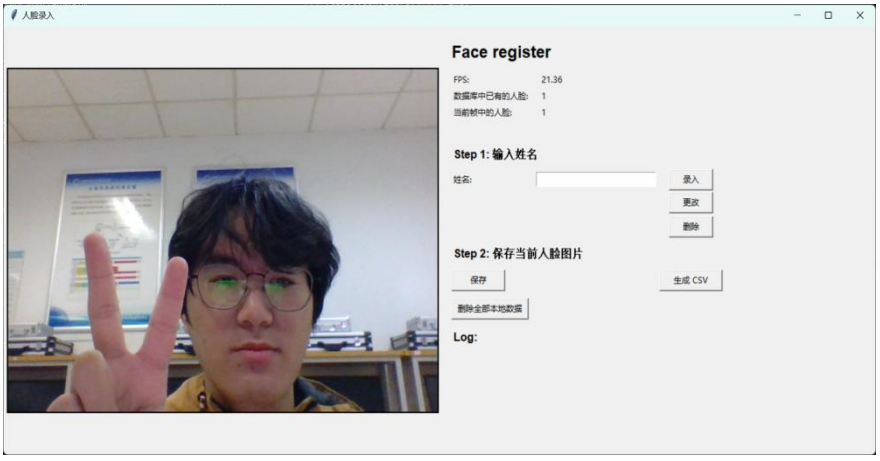

图 3-3 人脸信息采集窗口

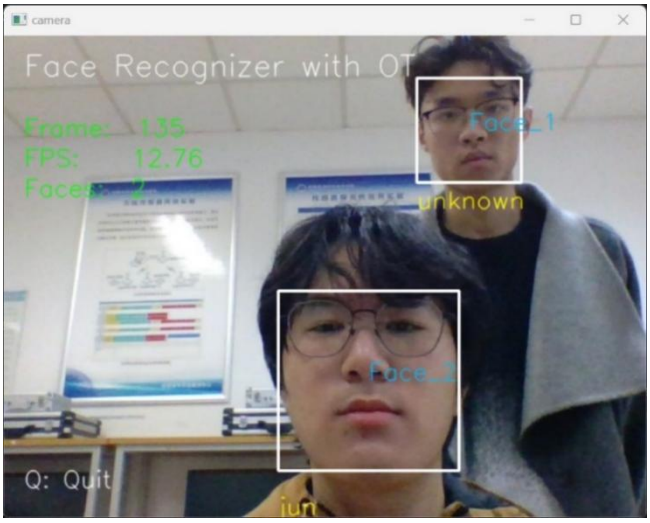

图 3-4 多脸识别界面

图 3-5 单人识别界面

3.3 特性成果

图 3-6 特性成果界面图

第 4 部分　总结

4.1 可扩展之处

本系统在以下两方面具备良好扩展性：

1．多摄像头扩展

硬件上可通过 GPIO/SPI 接口接入多路摄像头扩展模块，配合视频切换芯片实现多视角采集。软件端将开发图像拼接功能，基于特征匹配算法实现多帧融合，当前支持双路画面拼接，后续可扩展至 4 路全景输出。

2．串口精细调节

系统预留 RS485 硬件接口，软件集成 Modbus 通信协议。未来接入外置控制模块后，可直接通过软件界面实时调节曝光、白平衡等成像参数，并可进一步扩展至镜头焦距的远程控制，显著提升画质调节的灵活性与精细度。

4.2 心得体会

在参与基于 FPGA 与 Python 的人脸检测识别系统开发过程中，我们深刻体会到“软硬协同”的技术魅力与工程挑战。这不仅是一次技术实现，更是对跨模块协作、性能平衡与场景适配的深度实践。

1．硬件开发：低延迟与并行处理的精妙平衡

从硬件层面看，FPGA 的低延迟特性是系统的核心优势，但需要攻克多模块时序同步的难题。在 MIPI 接收、CSI 解包与 DDR3 存储的流水线设计中，我深刻认识到"并行处理"绝非简单的模块堆叠。通过精细调整 mipi_dphy_rx 的延迟参数（S_clk_lane_idelay）与 uidbuf_u0 的缓存策略，我们成功将 1080P 视频的处理延迟控制在设计指标内。特别是在设计多路输出架构时，面对更加复杂的时序约束，我们通过重新规划时钟域和优化数据路径，最终实现了 HDMI 显示与以太网传输的稳定并行输出。

2．软件开发：算法精度与执行效率的优化取舍

软件开发的重点在于算法精度与执行效率的平衡。我们发现通过 UDP 传输至上位机后，由于编程语言本身的特性限制，帧率与清晰度存在明显差距。经过多次试验，我们采取了创新的混合架构：利用 C++的 OpenCV 库完成低延迟的图像显示，同时结合 Python 完成功能补全。在 Dlib 特征提取方面，精度与实时性

存在天然矛盾，通过引入帧跳过策略（每 3 帧处理 1 帧）与质心追踪算法，在保证识别准确率的同时，将单帧延迟严格控制在 50ms 内。特征库（features_all.csv）的设计让我认识到，数据标准化是算法落地的前提是统一的 128D 向量格式与 0.4 的欧式距离阈值，为后续扩展多场景识别奠定了坚实基础。

3．软硬协同：系统优化的核心思维

最具启发性的是软硬协同的系统思维。FPGA 的预处理能力（包括 ISP 增强、裁剪校正）为 Python 算法有效“减负”，而软件识别结果又可反向指导硬件参数调节，实现摄像头曝光的动态优化。这种"硬件负责基础处理，软件专注智能决策"的分工协作，正是构建高效智能视觉系统的核心逻辑。

4．问题排查：从失败中汲取的宝贵经验

开发过程中也经历了诸多挑战：初期因 MIPI 时序失配导致图像花屏，后通过细化 PLL 时钟相位得以解决；DDR3 带宽竞争引发的帧丢失问题，通过优化存储访问策略得到改善。这些经历让我深刻理解，工程实践不仅需要扎实的技术储备，更需要在问题中迭代优化的耐心和系统思维。

最终系统实现了从视频流输入到身份标注的完整闭环，这不仅是代码的堆砌，更是对“需求→设计→验证→优化”全流程的深刻体现。展望未来，如何通过多摄像头拼接技术拓展场景适配性，以及进一步利用 FPGA 加速特征提取过程，将是我们持续探索的重要方向。这次项目让我们真正领悟到，优秀的工程解决方案需要在技术深度与实用价值之间找到最佳平衡点。