基于 ZYNQ-7020 的老人摔倒检测系统

马嘉庆；杨冯；乔钰理

第一部分　设计概述

1.1 设计目的

本设计的初衷是应对当前老龄化社会对安全监护的迫切需求，特别是针对老年人摔倒事故的预防与快速响应。摔倒是老年人最常见的意外伤害原因，严重时会导致骨折、头部受伤，甚至可能引发生命危险。传统的监控方法往往依赖人工观察，无法实现实时性，且在人力成本和响应速度上存在诸多限制。因此，研发一款高效、低功耗的摔倒检测系统，能够在第一时间发现并报告摔倒事件，为老年人提供自动化的监护成为实际需求。

基于此，本项目提出了一种基于 AMD Zynq-7020 FPGA 的嵌入式摔倒检测系统，集成了 YOLOv7 目标检测算法，以实现高精度、低延迟的摔倒识别。Zynq-7020 的异构架构使我们可以将计算密集型任务交给 FPGA 加速处理，同时利用处理系统（PS）完成控制逻辑，从而提高系统的处理速度与响应能力。本系统旨在为老年人监护提供有效的技术支撑，减少意外摔倒带来的风险，并为家庭和社会提供高效的监护解决方案。

1.2 应用领域

本摔倒检测系统的应用领域主要聚焦于老年人监护、医疗辅助和居家安全等场景，针对当前摔倒监测中的效率低、成本高、响应慢等痛点提供了解决方案。随着人口老龄化问题的加剧，越来越多的老年人选择居家养老或独居，他们一旦摔倒往往难以及时获得帮助，甚至可能因长时间无人发现而产生严重后果。传统的监护依赖人工观察，不仅人力成本高，且难以做到全天候监控，尤其在夜间和无人值守时容易疏漏。借助本系统的实时监测功能，可以精准识别摔倒行为，第一时间发出报警，确保家人或护理人员及时响应，为老年人安全提供保障。

在医疗环境中，本系统也可以应用于康复中心和医院病房，监控老年患者和行动不便者的安全状况，减轻医护人员负担，提升护理质量。此外，在智能家居

领域，本系统能为独居老人和有需求的家庭成员提供实时安全保障，成为现代家居生活中的一项安全增值功能。系统的高精度、低延迟和低功耗特性，使其能在不同场景下实现 24 小时实时、可靠的监护功能，填补了传统安防设备在摔倒监测领域的空白。如图 1 所示，老人摔倒如救助不及时，可能会错失最佳救治时间。

图 1 老人摔倒案例

1.3 主要技术特点

本项目基于 AMD Zynq-7020 FPGA 平台设计了一套高效的实时摔倒检测系统，充分发挥硬件和软件协同工作的优势，实现了低延迟、高精度和低功耗的摔倒检测效果。硬件端使用了 Zynq-7020 的异构架构，将 YOLOv7 算法中的卷积计算等计算密集型任务交由 FPGA 的可编程逻辑部分（PL）处理。通过这种硬件加速方式，大幅度提高了系统的运算速度和实时处理能力，达到了毫秒级别的检测延迟效果，确保了摔倒检测的及时性。

在软件端，本系统对 YOLOv7 算法进行了深度优化，包括模型剪枝和量化，以适应嵌入式平台的资源限制。这些优化手段有效降低了计算复杂度，使得 YOLOv7 可以在 FPGA 有限的硬件资源上高效运行。此外，通过处理系统（PS）对系统的逻辑管理和控制，软件端负责将硬件加速计算结果进行分类和判断，快

速识别摔倒行为，确保检测的准确性和稳定性。

通过硬件与软件的协同设计，本项目实现了高精度、低延迟的摔倒检测效果，并且在功耗方面表现出色，使系统适合全天候监控的嵌入式应用场景。整体设计达到了资源优化和性能提升的双重目标，为实时摔倒检测提供了可靠的技术支撑。

关键性能指标

本项目基于 ZYNQ-7020 平台，采用软硬件协同设计策略，构建了一套高效的实时摔倒检测系统。在硬件部分（PL），使用 HLS 方法创建了核心的摔倒检测计算模块，并通过 IP 核进行加速；在软件部分（PS），设计了上位机程序来管理加速器调用和系统控制流程。系统通过视频输入实时分析，识别老人摔倒等异常行为。关键性能指标如下：

（1）系统的总体运行时间减少了约 60%，单帧图像的处理延迟显著降低，实现了快速响应的实时检测效果。

（2）ARM 处理器在运行过程中显著降低了内存使用，释放了约 62%的存储资源，使系统资源利用更加高效。

（3）系统的运算能力达到约 9.48 BFLOPs，检测准确率超过 95%，保障了摔倒检测的可靠性和精确性。

（4）在 ZYNQ-7020 平台上运行时，检测帧率达到 3.57 帧/秒，能够满足实时性要求。

1.4 主要创新点

本项目在实时摔倒检测系统的设计中，针对硬件加速、算法优化、系统架构和功耗控制等方面进行了创新，取得了良好的效果。以下是本项目的四个主要创新点：

（1）异构架构中的软硬件协同优化，提高推理效率

本系统利用 ZYNQ-7020 的异构架构，将卷积神经网络（CNN）中的卷积、池化等计算密集型算子通过高层次综合（HLS）在可编程逻辑部分（PL）实现硬件加速，同时 ARM Cortex-A9 处理器在处理系统（PS）中负责系统控制和结果处理。通过这种软硬件协同设计，实现了数据流在硬件与软件间的高效传递，减少了数据传输延迟，在一定程度上提升了推理速度，使得系统可以在嵌入式环境

中实现接近实时的检测。

（2）模型剪枝与量化，实现嵌入式平台的高效模型部署

针对 YOLOv7 模型在嵌入式平台上的资源限制问题，本项目采用了模型剪枝和量化技术。通过通道剪枝，减少了不必要的参数和网络规模，降低了 FPGA 的资源需求，剪枝后权重矩阵依旧保持规则结构，便于 FPGA 的并行加速；通过 8 位定点量化，将浮点运算转化为低比特整型操作，减少了计算复杂度。这些压缩处理使得模型在有限的 FPGA 资源下可以高效运行，同时在保持较高检测精度的情况下降低了延迟和存储需求。

（3）动态功耗管理与中断机制，实现较低功耗的监控

为了满足嵌入式平台对低功耗的要求，本项目采用了动态功耗管理策略，利用动态电压与频率调节（DVFS）技术，根据系统负载动态调整 FPGA 的电压和频率。此外，在 ARM Cortex-A9 处理器端引入基于中断的事件驱动机制，在系统空闲时减少资源消耗。这些措施有效降低了系统的整体功耗，使其适合于持续监控场景，实测表明功耗相较于固定频率设计有所降低。

第二部分　系统组成及功能说明

2.1 整体介绍

在硬件端（PL 部分），使用 Vivado 和 ISE 14.7 设计并构建了系统逻辑。通过将 OV5640 摄像头连接到 ZYNQ-7020 的 USB HOST 接口，实现了视频数据的实时输入。在 PS 部分的配置下，系统采用加速策略执行 7 层神经网络的前向计算，并设计了基于原生 Verilog 的计算模型。通过移位寄存器结构进行加速，同时结合通道分组实现并行化计算，显著提升了处理速度和计算效率。

在软件端（PS 部分），首先对收集到的人流数据集进行数据预处理，包括数据归一化和无效值的清除等步骤。接着导入相关数据处理库，如 NumPy、matplotlib、pyplot 和 pandas。通过 NumPy 的数组生成函数将单张图像数据转换为适合后续处理的格式。在数据预处理后，将标定后的特征数据与分类标签关联，整理并汇总成适用于机器学习训练的数据集。将一段时间内的序列数据排列成数据矩阵，作为 DeepSORT 跟踪算法和 CNN 卷积神经网络的输入数据，以提高检测的准确性。

在显示与通信模块中，系统基于 ZYNQ-7020 的 RJ45 网口与管理端计算机实现高速网络通信，能够实时传输和记录检测到的信息，便于工作人员进行人流监测和分析，形成完整的智能化监控系统。通过此架构，系统可以实现对老人摔倒行为的实时检测，满足智能监控需求。根据总体系统框图，给出各模块的具体设计说明。如图 2 所示，为系统框架图。

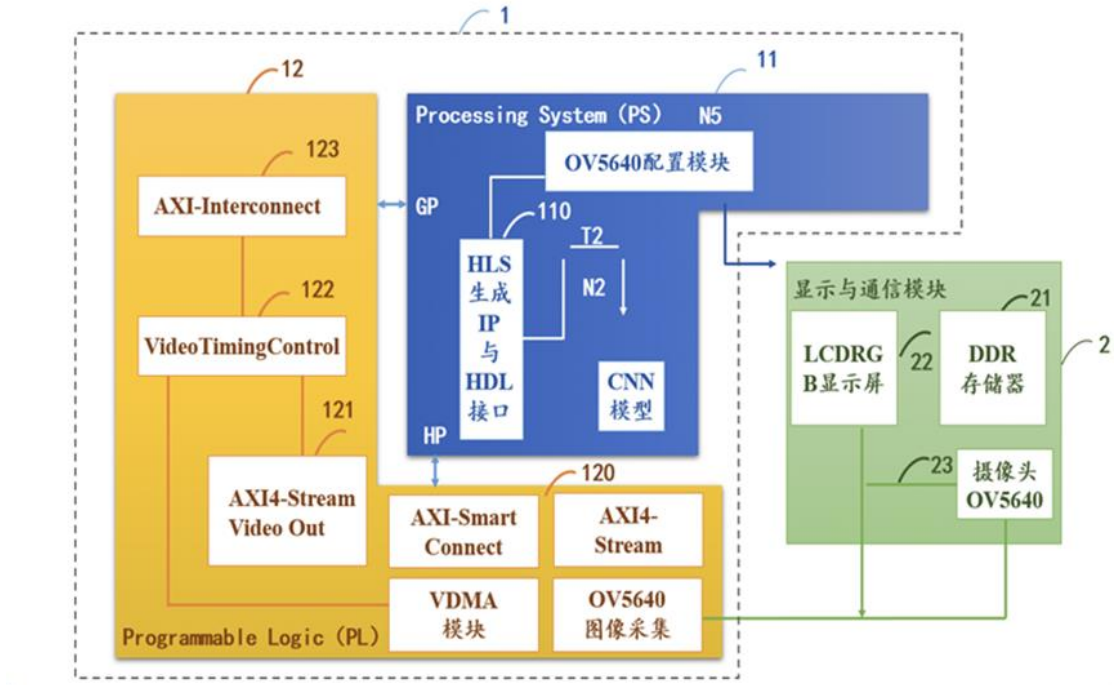

图 2 老人摔倒智能识别系统结构图

2.2 各模块介绍

(1) 基于 OV5640 的 ZYNQ-7020 硬件连接

ZYNQ-7020 是一款基于 Xilinx ZYNQ XC7Z020 SoC 的 FPGA 开发板，集成了可编程逻辑（Programmable Logic, PL）和处理系统（Processing System, PS），在嵌入式开发中具有很强的灵活性和处理能力。ZYNQ-7020 的开发工具包括 Vivado IDE、SDK 和 Vivado HLS，Vivado IDE 用于硬件开发，能够配置处理器、存储器、总线和互连等资源；SDK 则为 ZYNQ-7020 提供了软件开发平台，支持对 Xilinx IP 的驱动开发，使用 C 和 C++进行程序编写。

ZYNQ-7020 的处理系统（PS）部分包含双核 ARM Cortex-A9 处理器，工作频率为 650MHz。在执行神经网络推理任务时，ARM 内核与专用卷积加速器协同工作。通过在 PL 部分嵌入卷积引擎和可编程软核，可减轻 ARM 处理器的负担，并允许卷积加速器在精细控制下运行，从而提高处理效率。

在可编程逻辑（PL）部分，ZYNQ-7020 包含 13300 个逻辑片，每个逻辑片中包括四个六输入查找表（LUT）和八个触发器。板载 630KB 的块 RAM 为数据处理提供了必要的存储支持，220 个 DSP 片用于实现高速的数值运算，满足卷积

计算的需求。此外，四个时钟管理模块中集成了锁相环（PLL）和混合模式时钟管理器（MMCM），支持复杂的时钟设计要求。ZYNQ-7020 开发板还配备丰富的外设接口，包括以太网、HDMI 输入/输出、音频输入/输出、Arduino 和树莓派接口、多个 Pmod 接口、用户 LED、按钮和开关。兼容的 GPIO 接口和扩展端口支持灵活的外围设备连接，可以通过 HDMI 或 DSI 接口外接显示器，并通过网口进行 SSH 远程调试。

摄像头方面，OV5640 摄像头模块基于 OmniVision 的 1/4 英寸 QSXGA CMOS 传感器，支持自动对焦功能。该摄像头通过 24 针 FPC 软排线连接到主板，通过 I2C 接口与 FPGA 进行数据传输，以实现实时图像传输和处理。具体的连接方式如图 3 所示。

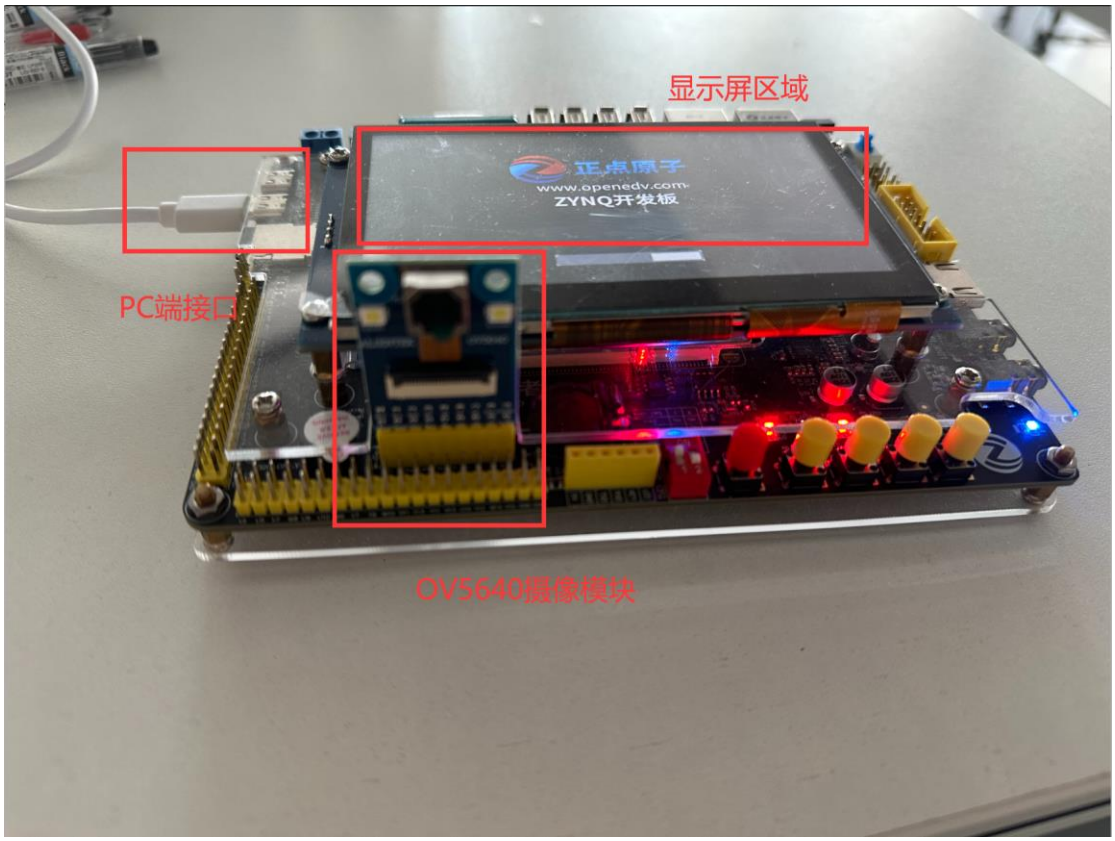

图 3 ZYNQ-7020 连接示意图

(2) 老人摔倒检测 YOLOv7 神经网络模块

本系统采用 OV5640 摄像头获取实时视频数据，结合 ZYNQ-7020 开发板部

署的 YOLOv7 模型，实现对老人摔倒行为的实时检测。YOLOv7 是一种改进型的目标检测算法，基于 anchor-based 机制，通过识别边界框并评估目标置信度来定位图像中的人物目标，非常适合在资源受限的嵌入式系统上实现高效检测。

在网络架构方面，YOLOv7 通过使用优化的 CSPDarknet 作为主干网络，增强了模型的特征提取能力。主干网络中的跨阶段分割网络（Cross Stage Partial Network, CSPNet）结构通过特征融合和轻量化设计，既减少了计算量，又保持了较高的检测精度。YOLOv7 结合了高效的特征金字塔网络（FPN），实现了多尺度特征的融合，使系统能够在不同的分辨率下精确识别多种姿态的摔倒行为。该设计使得 YOLOv7 在嵌入式环境中实现了较好的实时性和准确性平衡。

系统处理流程为：首先，将摄像头采集到的图像送入 YOLOv7 的主干网络，提取特征图。随后，通过并行的检测分支，系统预测出边界框、目标置信度和类别标签，以实现对摔倒行为的识别和定位。YOLOv7 的设计在主干层和检测层引入了特征金字塔网络，确保不同尺度的特征图可以在检测过程中充分利用，从而有效提升系统在复杂环境下的检测准确性。

通过这种架构，YOLOv7 在 ZYNQ-7020 平台上能够以较低的延迟实现实时摔倒检测。整体系统在较低功耗的条件下维持了较高的检测精度和实时性，非常适合用于老年人监护场景。

具体检测图实物图如图 4 所示。

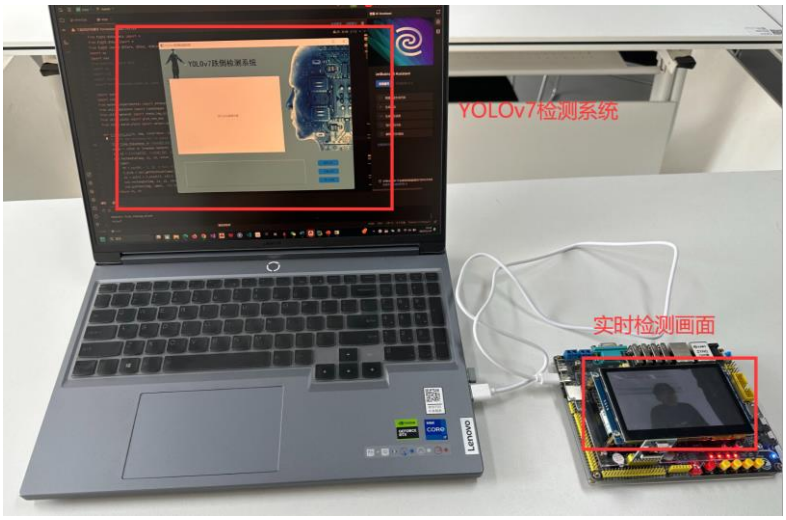

图 4 YOLOv7 检测实物图

(3) 基于 PyQt5 的老人摔倒检测 UI 交互界面

该系统的 UI 交互界面采用 PyQt5 设计，专为老人摔倒检测应用量身定制，实现了清晰、直观的实时检测反馈和用户交互功能。通过连接部署了 YOLOv7 模型的摔倒检测系统，界面可以实时显示视频流，并在检测到疑似摔倒事件时，用红框标注检测到的目标，及时提醒用户异常情况。检测结果会以时间戳和事件信息的形式在界面下方滚动显示，便于用户查看检测历史记录。

该系统具备较强的实时性，基于 ZYNQ-7020 FPGA 加速 YOLOv7 模型的推理，结合界面反馈机制，使得摔倒检测可以在较低延迟下完成，非常适合老人监控应用。界面直观，检测到的疑似摔倒目标会通过红框标出，并在下方实时显示事件记录，便于用户及时查看并采取措施。此外，系统自动记录每次检测的时间和事件描述，方便用户随时查看历史数据，形成完整的监控日志，这对护理人员或管理者进行日常监控与分析尤为有用。UI 界面设计简洁，包含“选择文件”“开始识别”“停止识别”等按钮，操作便捷，适合各年龄层的用户。系统采用了优化的 YOLOv7 模型，具备高检测精度和低误报率，极大提升了老人摔倒检测的可靠性。

UI 界面如图 5 所示。

图 5 基于 PyQt5 的 UI 交互界面

第三部分　完成情况及性能参数

3.1　FPGA 硬件平台准备与开发环境搭建

我们使用 OV5640 高像素 CMOS 摄像头模块模拟家用监控摄像头。ZYNQ-7020 FPGA 开发板通过 CSI 接口采集视频流，将实时视频数据输入到板上部署的 YOLOv7 神经网络模块中进行摔倒检测处理。当检测到摔倒事件时，ZYNQ-7020 通过串口将检测结果传输至 PC 端，以便进行进一步的存储和处理。

3.1.1　OV5640 摄像头

图像采集使用 OV5640 摄像头连接至 ZYNQ-7020 开发板的 CSI 接口，作为视频输入入口，使用 I2C 总线进行配置控制。OV5640 的实物图与结构图如图 6、图 7 所示。

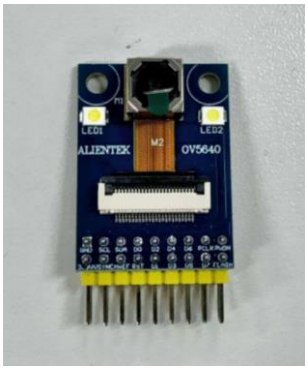

图 6 OV5640 摄像头实物图

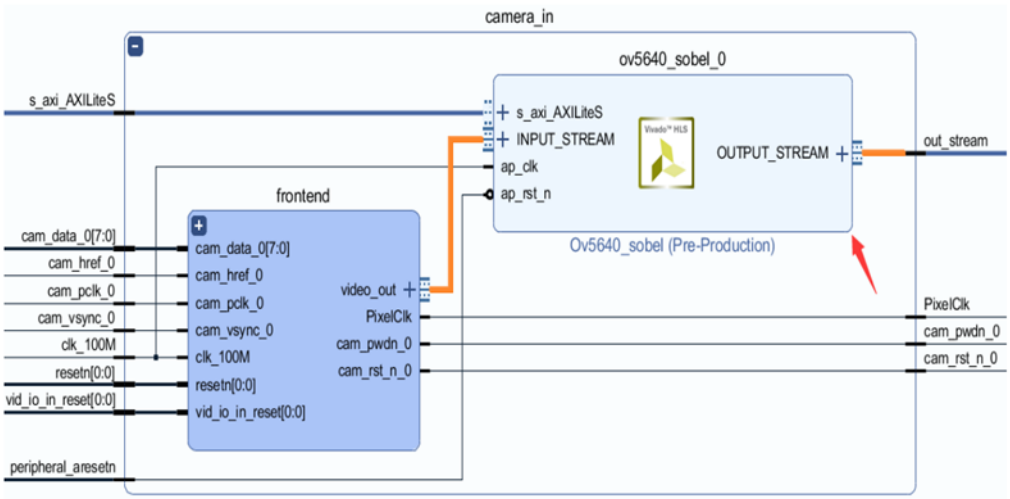

图 7 OV5640 结构图

OV5640 摄像头采集居家老人跌倒信息后，通过接口将数据传输到 ZYNQ-7020 的 ARM 处理系统部分进行处理和分析。针对复杂的测试环境，在 Overlay 中例化了一个 I2C 接口来对 OV5640 进行配置，通过改动 ov5640_config.py 文件的内容修改配置。

首先实例化 IIC：

```
iic = AxiIIC(ov5640_ol.ip_dict['cam_iic'])
# OV5640 器件地址
address = 0x3c
ov5640= OV5640(address, iic)
```

初始化 OV5640 并调用相关的方法配置 OV5640，如图 8 所示。

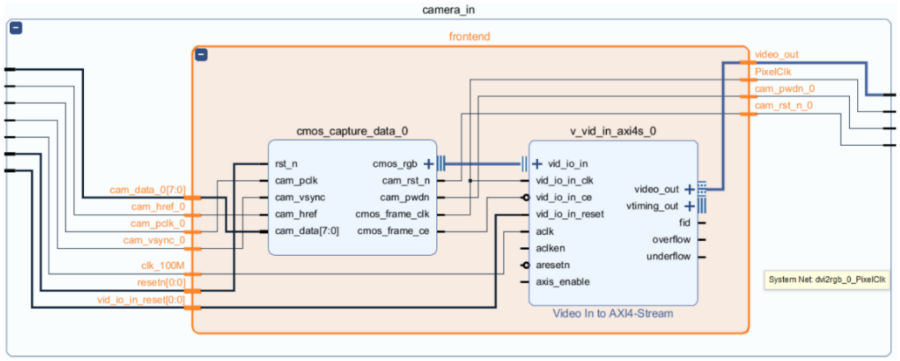

图 8 camera in 层次架构

3.1.2 基于正点原子 ZYNQ-7020 开发板的功能设计

正点原子 ZYNQ-7020 开发板是一款基于 Xilinx ZYNQ-7020 芯片的高性能 SoC（System on Chip）开发板，结合了 Python 编程和 FPGA 硬件设计。利用 ZYNQ-7020，开发者可以充分发挥 Xilinx ZYNQ All Programmable SoC (APSoC) 的功能，同时通过嵌入式系统实现灵活的硬件控制。

其中，ZYNQ 系列 SoC 的设计包括两个主要部分：PS（Processing System）和 PL（Programmable Logic）。PS 部分包含两个 ARM Cortex-A9 核，可以运行 Linux 操作系统，并在操作系统上进一步运行 python 程序，如 YOLOv7 神经网络模块。借助 Linux 系统的灵活性，可利用 python 主程序来控制系统或传输数据。PL 部分就是 FPGA 的逻辑资源，开发者在 PL 中添加 IP 或者将自己用 C

或者 HDL 语言写好的模块封装成 IP，这些 IP 都被连接到 PS 端，一般都是通过 AXI 总线。通常，这些 IP 核通过 AXI 总线连接到 PS 端，允许 PS 和 PL 之间进行数据交换和控制。

在本项目的开发流程中，我们使用 vivado 进行集成设计环境。每次开始一个新的涉及 PL 端的开发的时候，先在 vivado 里面建一个工程，添加需要的各种 IP，然后以 ZYNQ 为核心连接，将设计的模块与 ZYNQ 的 PS 部分相连，通过 AXI 总线实现 PS 和 PL 之间的通信。完成设计并编译后，Vivado 会生成一个 bit 文件和一个 tcl 文件。bit 文件就是硬件设计，tcl 文件描述了接口关系。将这些文件复制到 ZYNQ-7020 的文件系统目录下，即可进行调用。

第一步，完成 ZYNQ-7020 板卡的打开：

首先，下载适用于 ZYNQ-7020 的嵌入式 Linux 镜像，并将其烧录到 SD 卡中。使用开源工具 Win32DiskImager 将镜像文件写入 SD 卡，如下图 9 所示。

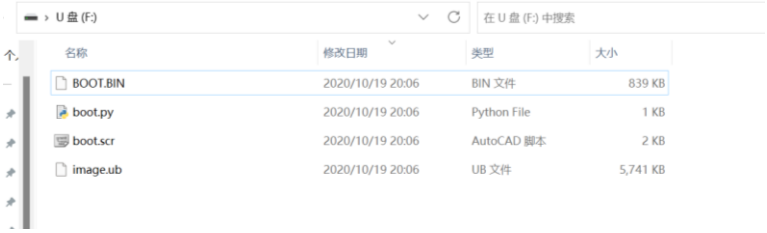

图 9 镜像文件烧录图

其次，将 SD 卡插入 ZYNQ-7020 开发板，以便在其上启动 Linux 操作系统。如图 10 所示。

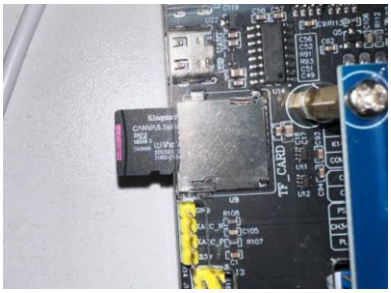

图 10 SD 卡接入图

之后用电源线插入 SDK 凹槽上方的 TypeC 接口，注意按下电源开关，开发板正常供电启动。先是红灯亮，表示上电成功，大概一分钟后，蓝灯和黄灯闪烁，最后蓝灯灭，黄灯亮，表示板子工作正常，如图 11 所示。

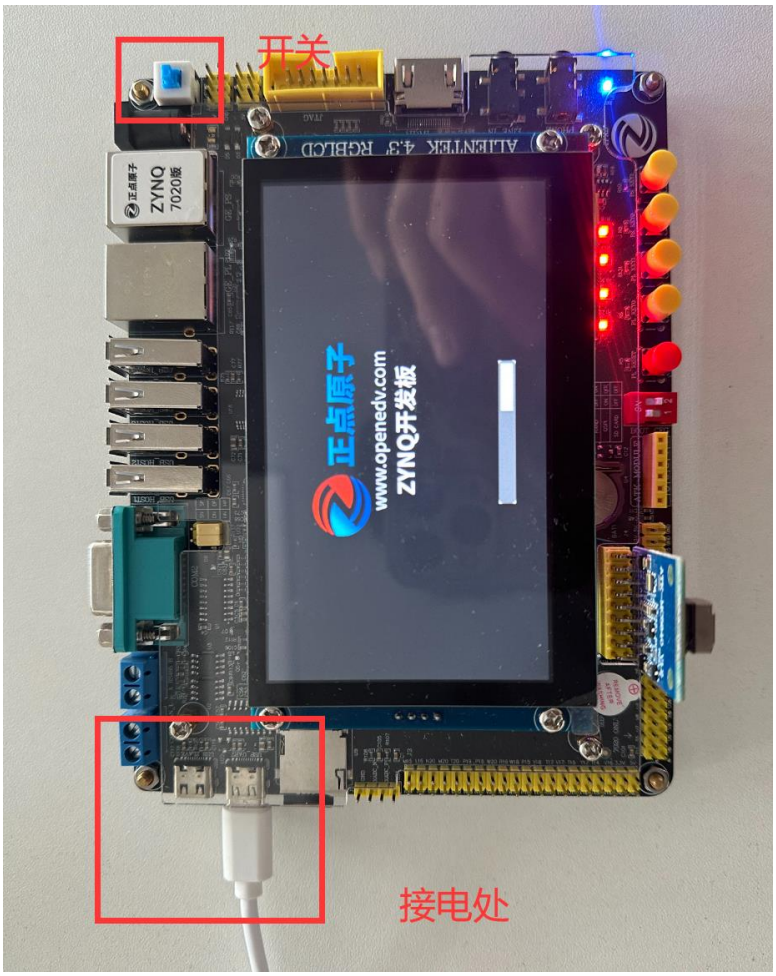

图 11 开发板供电图

最后，通过 samba 传输文件。

在开发过程中，需要在 PC 机与板卡之间传输一些较大的文件，可以利用 samba 协议将 ZYNQ 的文件系统当作一个网络硬盘直接读取。不过由于 ZYNQ-7020 开发板并不像 PYNQ 开发板那样自带 Samba 服务，需要手动安装 Samba 服务、配置共享文件夹、使用默认用户 xilinx 设置 Samba 用户。

在 Windows 中只需要打开资源管理器，输入\pynq\xilinx 即可成功连接。在 Mac/Linux 中同样可以打开文件管理器，输入 smb: //pynq/xilinx 进行挂载。注意，用户名和密码均为 xilinx。

第二步：Vivado 中 ZYNQ-7020 主体框架撰写与代码烧录：

启动 vivado 2019.2，在 Quick Start 中创建项目，找到 ZYNQ-7020 并创建工作区。

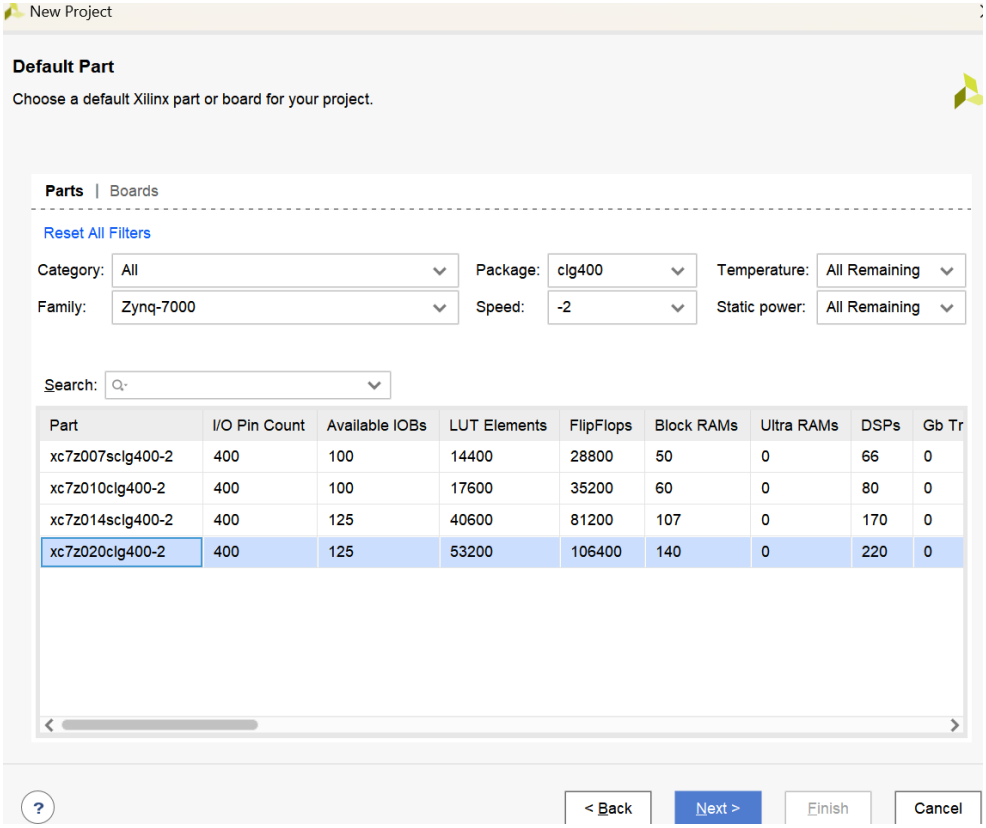

图 12　ZYNQ-7020 项目

在工程设置中，选择 “Create Block Design” 选项，进入 Block Design 环境。Vivado 会打开一个 Block Design 界面，搜索并添加包含 ARM 处理器的 “ZYNQ7 Processing System” IP 核。双击 PS 模块，Vivado 会自动配置 PS 端的默认参数，关闭无用通道，保留一个串口通讯口与 I2C 通讯口，因需要与 PL 部分连接，PS 端的 AXI 总线接口可以导出，用于与自定义 IP 或其他 AXI 外设连接，配置完毕如图 13 所示。

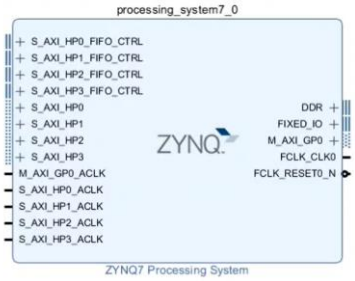

图 13　“ZYNQ-7020 Processing System” IP 核

搜索并添加 conv 核，调整“Block Automation”与“Connection Automation”，

优化布局，最终结果如图 14 所示。在完成连接后，选择 “Validate Design” 以检查设计中的错误或未连接的接口。Vivado 会给出提示，确保设计的完整性。确认并生成 HDL Wrapper，将 Block Design 转换为顶层的 HDL 文件，生成 Bitstream 后将硬件文件导出供后续综合和实现流程使用。

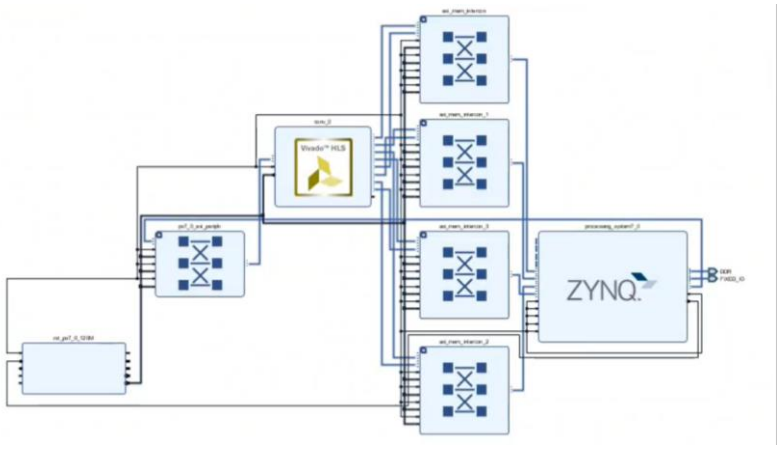

图 14 接口连接

之后进行 SDK 开发：

①创建平台项目

file ->export -> export hardware。然后需要勾选 include bistream，点击 OK。

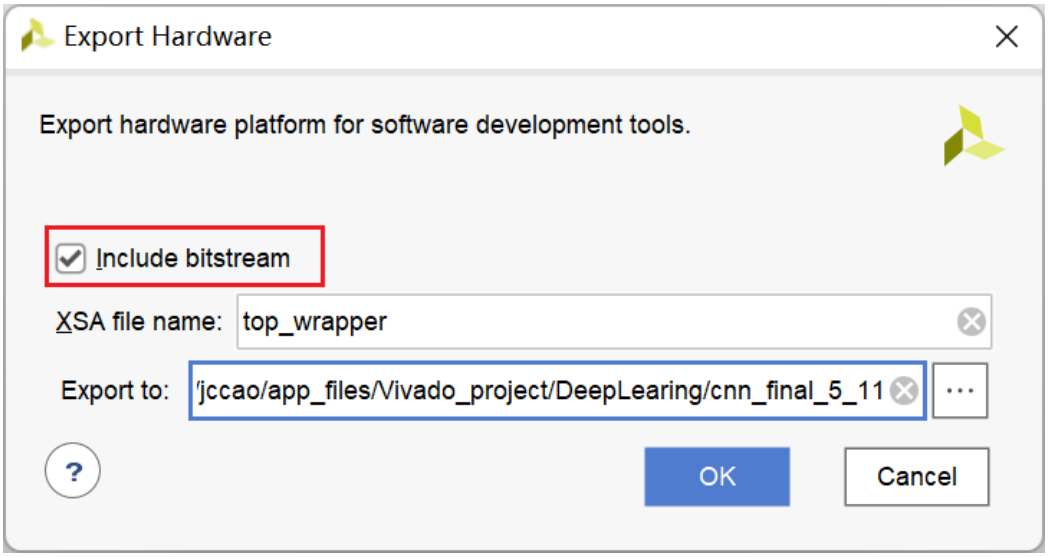

图 15 创建平台项目步骤一

生成的.xsa 文件位于工程目录下。

这里使用的 Vivado 版本为 2019.2，从此版本开始，导出的硬件描述文件

为.xsa 文件，给 vitis 平台使用，Vitis 是 Xilinx SDK 的继承开发工具。

选择工作目录后，点击 Lanch，点击 Create Platform Project 创建工程，如图 16 所示：

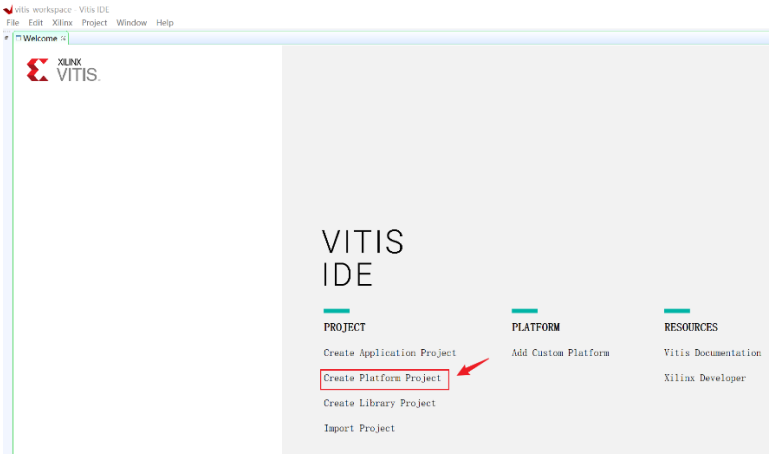

图 16 创建平台项目步骤二

输入工程名字，长度需在 3-40 个字符之间，指定刚刚生成的 .xsa 文件，点击 finish 即可。

②创建应用项目，进行编译工程，选中 APP 工程，右键“Build Project”按键，进行工程编译。

工程编译结束后，成功生成 elf 文件。至此，硬件和软件设计均已完成，如图 17 所示。

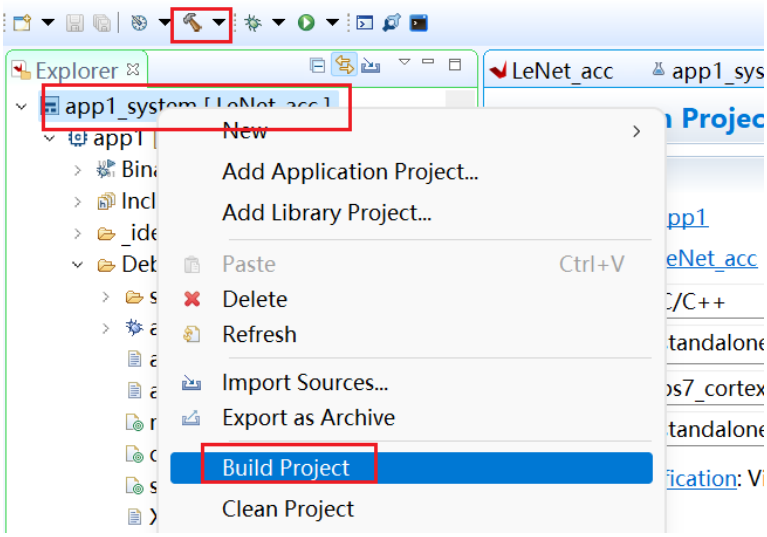

图 17 生成编译工程

第三步：下载验证并烧录程序

首先我们将 microusb 数据线与 ZYNQ-7020 开发板上的接口连接，数据线另外一端与电脑连接。

在菜单栏中依次点击“Window->Show view->Terminal 文件夹->Terminal”，最后点击“Open”，接口成功添加 Terminal 窗口。点击图标， 进行串口设置界面，如图 18 所示：

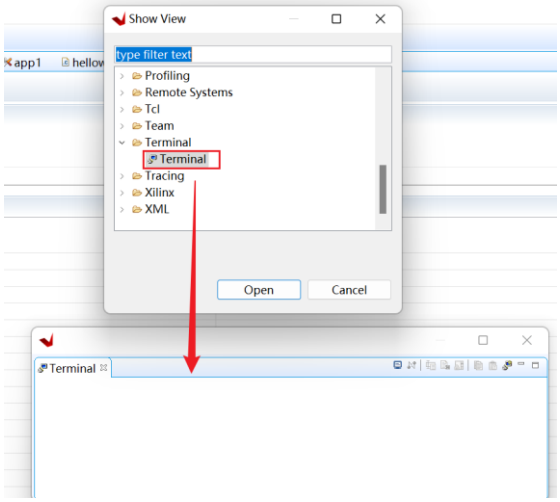

图 18 串口设置

其次，选择串口“Serial Terminal”，设置的参数需要与硬件设计过程中配置的 axi_uartlite_0 保持一致，即波特率为 “115200” ，数据位为 8 位，停止位为 1 位。 点击“OK” 后，如图 19 所示，证明串口连接成功。

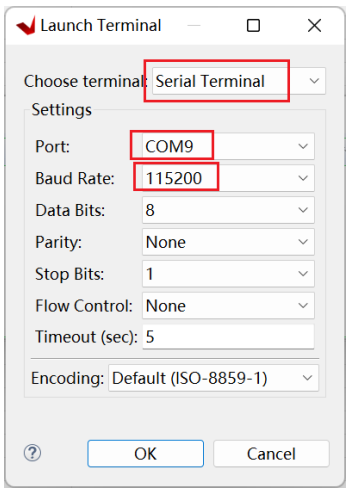

图 19 串口连接

最后，在 Run Configuration 页面点击“Single Application Debug(GDB) -> Debugger_app1-GDB”，点击菜单栏“Target Setup”。其中，“HardwarePlatform”为硬件平台， Bitstream File 为加载的 bit 流文件。 勾选“Reset entire system”（系统复位）和“ProgramFPGA”（下载 FPGA）然后点击“Run”开始下载程序，如图 20 所示。

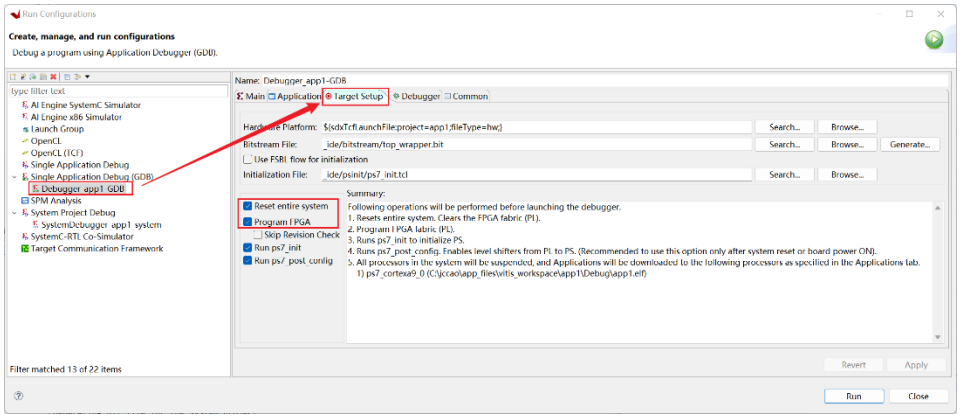

图 20 下载程序图

最后，将文件按照以上方法烧录进板卡。

第四步：效果验证

文件烧录进板卡之后，打开 pycharm，UI 界面正常运行，如图 21 所示。

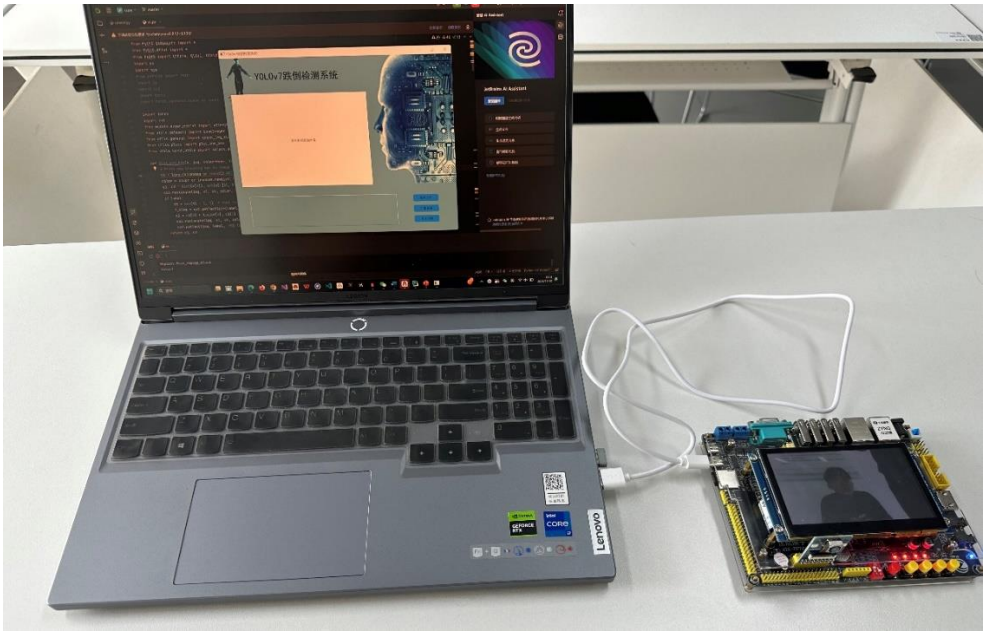

图 21 老人摔倒智能检测系统识别图

3.1.3 基于 YOLOv7 网络的 IP 核加速器

在 Vivado2019.2 中使用 Block Design 硬件设计平台进行架构设计。先添加 Zynq IP 核并配置对应参数，如时钟频率、复位信号、DDR、外围 IP 引脚、PS-PL 交互等，再加入由 HLS 导入的 YOLOv7 IP 核，将其与 Zynq IP 核进行连接。连接使用了 Xilinx 官方提供的 AXI SmartConnect IP 核，ZYNQ 系统通过该 IP 核进行 PL 端通信和数据交互，将 S_AXI_HP 端口与 m_axi_DATA_BUS 端口互通，即建立与 YOLOv7 IP 核的连接。最终硬件搭建完成。

具体过程如下面系列图所示。

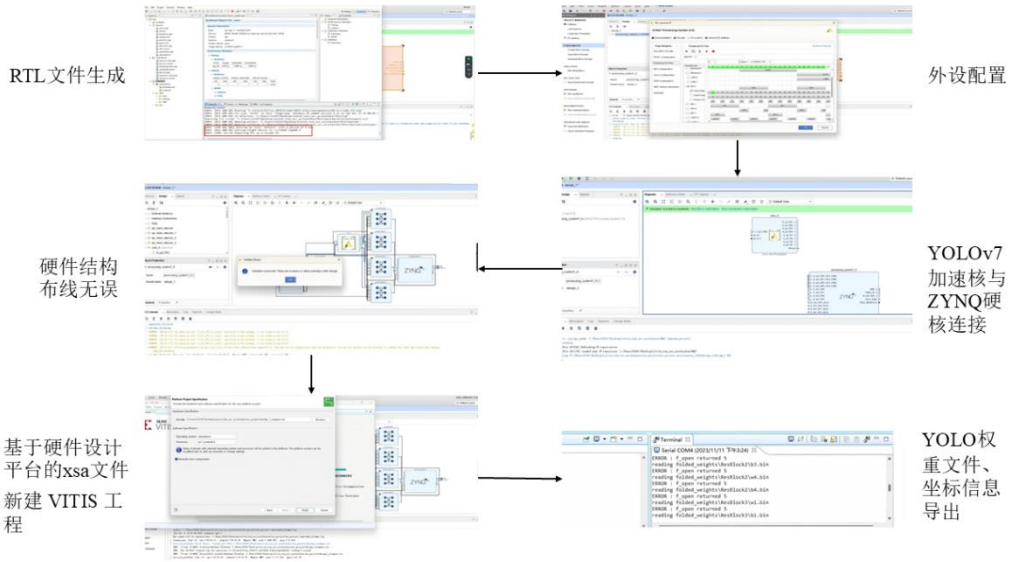

图 22 IP 核加速器连接

然后使用 Vivado Vitis 工具将上述设计应用到实际场景中。在应用程序中，应该特别注意在脚本中调整堆栈大小和堆大小的需要。Ld 文件在工程中价值相对较高。同时，为了防止由于程序栈不足导致训练失败或中断，需要将 BSP 中 standalone 中的 use_lfn 参数值调整为 1。

通过实验，未经加速优化的模型平均准确率维持在 95.8%，中间推理运行时间维持在 6-8 分钟；运行时间显著减少，保持在 300～800ms。从实验结果中不难发现，与未优化模型相比，优化模型的推理精度几乎没有变化，但运行时间普遍提高了近 100 倍；即优化系统将显著提高模型推理速度，损失精度几乎可以

忽略不计，这为应用神经网络模型提供了显著的优势和便利。

表 1 未经加速优化模型与优化模型参数对比

操作系统	平均推理速度	模型推理精度
CPU Cortex-A9	7.23 mins/Picture	78.9%
FPGA-Based Accelerator	487.95 mins/Picture	78.1%

3.2 YOLOv7 神经网络模块

3.2.1 理论基础

YOLOv7（You Only Look Once version 7）是一个基于深度学习的目标检测算法，能够在实时检测中实现高效、精确的目标定位。在进行老年人跌倒智能识别时，我们选用了 YOLOv7 模型。该模型是一种端到端的卷积神经网络，它能够直接接收整张图像作为输入，并在输出中给出每个目标的边界框位置、物体类别的预测概率以及该预测的置信度。

第一步：将图像统一裁剪大小，作为神经网络的输入（yolo 模型将图像大小裁剪到 448×448）。

第二步：通过一个端到端的卷积神经网络，得到一些边界框的坐标、框中物体所属类别的概率和置信度；

第三步：进行非极大值抑制（NMS），筛选框（Boxes）。

针对老年人跌倒检测的复杂环境，我们基于自定义的跌倒数据集进行了训练。数据集包含了多种跌倒和非跌倒行为的图像，通过摄像头采集居家环境中的视频数据，并根据拍摄角度和距离选择了 4000 张样本图像。使用 LabelImg 工具对这些图像进行标注，将模型导出为 ONNX 格式，用于模型训练和测试。

在 Yolov7 模型训练上，首先，准备并标注包含目标对象的图像数据集，按比例划分为训练集和验证集。然后，配置模型环境，包括安装依赖项、配置数据文件。接着，通过训练脚本启动模型训练，过程中输出模型的各项指标如精度和损失。训练完成后，使用最佳模型权重文件对验证集进行评估，以获取平均精度（mAP）、准确率和召回率等性能指标。最后，将训练好的模型部署在实际环境中，通过检测脚本进行推理，实现老人跌倒检测功能。

为了提高跌倒检测的精度和鲁棒性，我们结合了 ZYNQ-7020 开发板的硬件加速能力，将 YOLOv7 模型的一部分计算任务卸载到 FPGA 中，进行加速处理，从而提升了整体检测的实时性和准确性。

在系统实现中，我们还使用了 Optical Flow 中 Lucas-Kanade 光流算法来实现检测视频帧间的人体运动方向和速度变化，从而更高精度地识别跌倒行为。Lucas-Kanade 光流算法一种相对简单的光流算法，适合在资源有限的 FPGA 上实现。它通过逐像素的局部窗口来估计运动方向和速度，计算速度快。Lucas-Kanade 光流算法在人体区域内计算相邻帧之间的运动矢量，以获得人体运动方向和速度信息。

Lucas-Kanade 光流算法依赖以下几个**关键数学公式**来估计运动矢量。

1. 图像亮度一致性假设

假设在连续帧中，一个像素的亮度保持一致，即图像亮度一致性假设。对于第一个帧中人体区域内某个像素点的亮度$I(x,y,t)$，在下一帧中其亮度保持不变：

$$I(x,y,t) = I(x+\Delta x, y+\Delta y, t+\Delta t)$$

其中：

- (x,y)是当前帧中的像素位置。
- Δx和Δy是该像素在下一帧中的位移。
- t和$t+\Delta t$是相邻的两个时间帧。

2. 光流约束方程

对上式进行泰勒展开并忽略高阶项后，得到：

$$I(x+\Delta x, y+\Delta y, t+\Delta t) \approx I(x,y,t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t$$

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0$$

根据亮度一致性假设，左右两侧相等，因此可以得到：

定义$u = \Delta x/\Delta t$和$v = \Delta y/\Delta t$，分别表示光流在 x 和 y 方向上的速度（位移/时间）。于是得到：

$$I_x u + I_y v = -I_t$$

该公式就是光流约束方程，表示在给定图像梯度下的运动约束。

3. 多点窗口内运动一致性假设

为了增强光流计算的稳定性，Lucas-Kanade 算法在局部窗口（例如 3×3 或 5×5 的窗口）内假设运动一致性，即窗口内所有像素的光流向量相同。在人体区域内的小窗口中，对每个像素点均有光流约束方程：

$$I_{x,i}u + I_{y,i}v = -I_{t,i}$$

其中i表示窗口中的每一个像素点。

4. 光流矢量的计算结果

求解得到的u和v分别为当前窗口中心点在相邻帧间x和y方向的运动速度。

基于 YOLOv7 与 Lucas-Kanade 光流算法的结合，对老人跌倒检测任务提供了更高的精度。YOLOv7 可实现精确的目标检测与跌倒识别估计，而 Lucas-Kanade 光流算法通过计算连续帧之间的运动矢量，跟踪并捕捉人体的运动方向和速度变化，能够更加有效识别跌倒等突发行为。实验表明，在高人流密度的场景下，YOLOv7 与 Lucas-Kanade 光流算法结合后，不仅增强了目标的时序信息，提升了检测的稳定性，而且在老人跌倒检测任务中相比单独使用 YOLOv7 具有显著的精度提升，尤其是在运动模糊和目标信息模糊的情况下。

表 2 老人跌倒检测随目标运动模糊程度的准确率变化

运动模糊程度/%	准确率/%
30%	96.4
50%	93.1
70%	80.2
90%	77.4

3.2.2 Yolov7 模型剪枝与量化

Yolov7 剪枝主要有两种方法：非结构化剪枝和通道剪枝。非结构化剪枝通过在权重矩阵中剪掉不重要的权重来减少计算需求。非结构化剪枝的效果较好，但不利于在 FPGA 上进行加速，因为不规则的数据结构不利于硬件的并行化。而通道剪枝在硬件实现中更为高效，因为它剪掉的是整个卷积核的通道，因此剪枝后权重矩阵依旧保持规则结构，便于 FPGA 的并行加速。故我们选择采用通道剪

枝方法。首先定义剪枝标准，并选择剪枝比例，之后按照定义的剪枝标准，计算每一层中每个通道的重要性。由此确定要剪除的通道。通道剪枝会导致模型的结构发生变化，精度往往会有所下降。为了恢复精度，我们对剪枝后的模型进行微调。通道剪枝效果如表 3 所示。

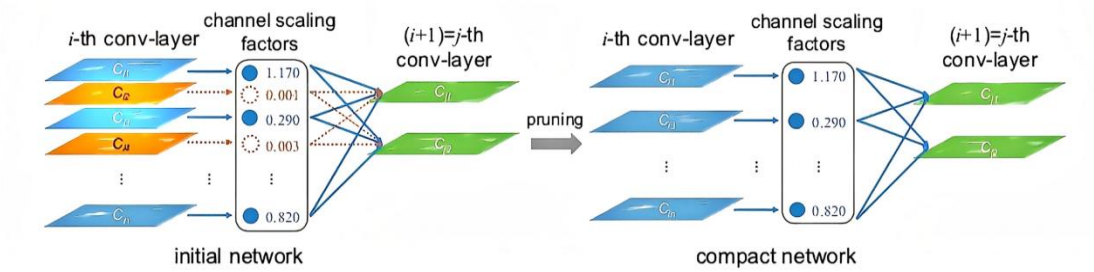

图 23 通道剪枝原理图

表 3 通道剪枝前后效果对比图

指标	剪枝前	剪枝后	减少幅度
参数量（M）	50.2M	15.3M	约 69%
计算量（GFLOPs）	125 GFLOPs	38 GFLOPs	约 69.6%

8 位定点量化是一种模型压缩和加速技术，通过将模型的权重和激活从 32 位浮点数转换为 8 位整数表示，以减少模型的存储需求和计算成本。这种量化方式可以显著提升推理速度和减少模型大小，同时对精度的影响较小。在 8 位量化中，我们将模型的浮点权重和激活值映射到 8 位整数范围，然后在推理时使用整型操作代替浮点操作。8 位定点量化与通道剪枝结合使用，能够进一步提高 YOLOv7 的轻量化效果。图 24 为 8 位量化曲线示意图。

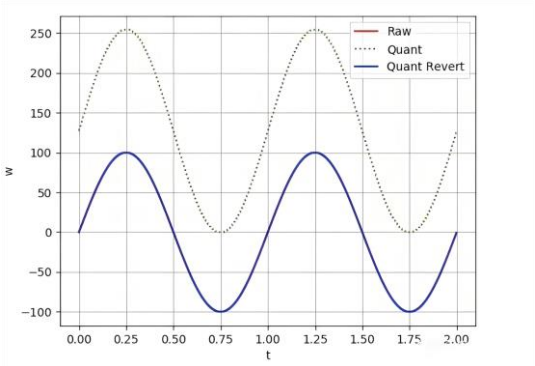

图 24 8 位量化曲线示意图

3.2.3 数据集准备与训练

针对复杂的测试环境，本文根据人体摔倒数据集 fall-dataset，在各类环境，不同场合，

各类的摔倒姿势和不同的拍摄角度采集相关图像 1428 张，通过 labimg 对选取的图片进行标注，制作 VOC 格式数据集，作为公共复杂环境人体摔倒检测数据集。

图 25 数据集情况

在训练前，分别建立文件夹存放.xml 格式的标签和采集的图片，划分摔倒数据集 fall-dataset 的 81%为训练集进行训练，9%为验证集进行验证，10%为测试集进行测试，完成对数据集的预处理。

对 YOLOv7 进行训练，我们修改相应的配置文件，在 model_data 文件夹下新建自己的标签类别 txt 文件，在其中写入我们训练的类别标签，我们只有一个类别就是 fall。之后，我们下载预训练权重文件，将下载好的文件放在 model_data 下，下载好后，我们就可以通过调用 train.py 文件来进行模型训练，我们设置的 epoch 为 300 轮：

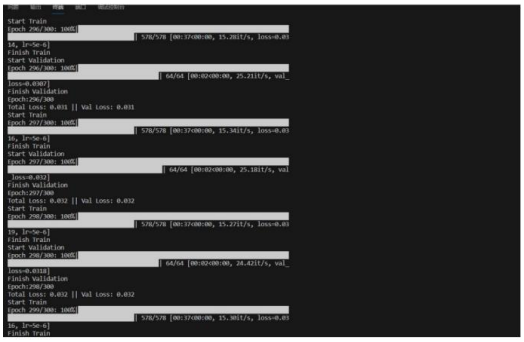

图 26 训练情况

训练完成后，我们得到了训练中的权重文件和训练效果最好的权重文件。

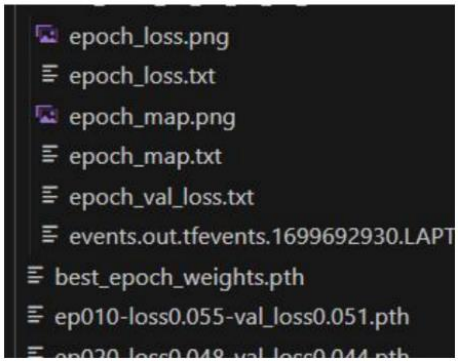

图 27 训练完成情况

还有模型相关性能参数平均精确度 Map 和损失函数值 Loss 的收敛过程，判断模型精度。本次 300 轮训练的模型中得到的训练结果如下：

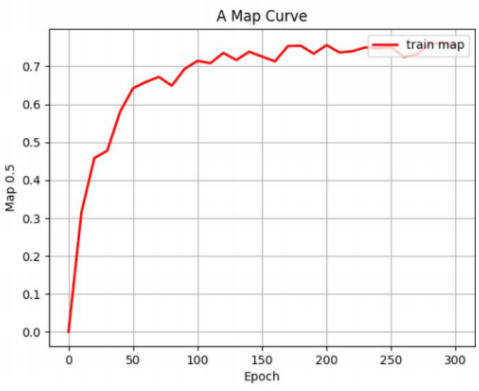

图 28 平均精确度 Map

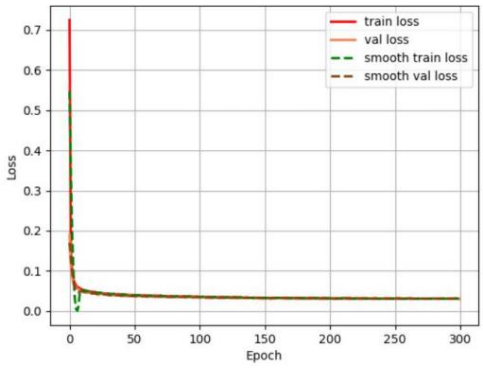

图 29 损失函数值 Loss

3.3 识别方法与效果

我们的识别方法如下：首先，系统在监控画面中选中人物，分析其姿态和状

态，特别是检测其是否发生摔倒。当人物出现摔倒姿势时，系统将此行为记录为摔倒事件。在 UI 交互界面上，系统实时显示摔倒事件的总数，并记录每个摔倒人物的时间及相关信息，便于用户进行后续管理和分析。如图 30 所示。

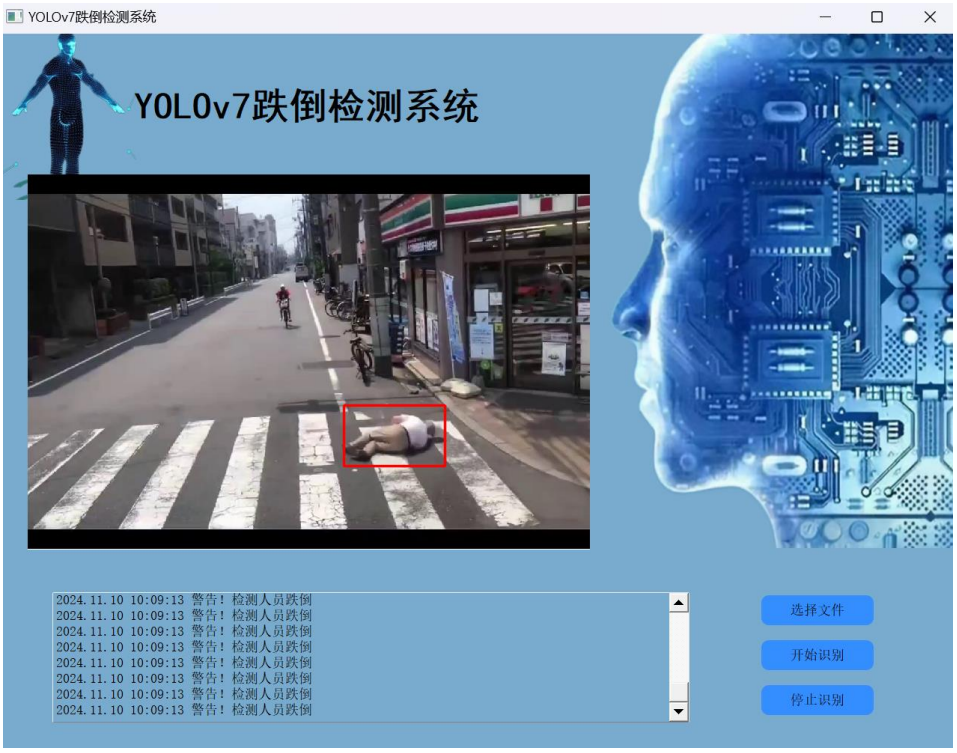

图 30　UI 界面图

对于一些特殊情况，一般能处理地较好处理。

1. 多场景应用

本检测系统具备在多种复杂场景下应用的能力。很多传统跌倒检测系统只能在特定环境（如光线充足且背景简单的室内）下运行，一旦背景复杂、光线变化或出现动态场景，这些系统往往失去准确性。而 YOLOv7 系统在户外、室内、商场扶梯、公共场所等多种环境下均能保持高精度的检测效果。无论是在空旷的公园、繁忙的交通枢纽，还是在家居环境中，系统都能够迅速识别人员跌倒情况，为用户提供可靠的实时安全预警。特别是在公共场所或开放空间，系统能够适应环境中的各类干扰因素，展现出良好的鲁棒性。这种多场景适应性使 YOLOv7 系统更具实用价值，满足了监护老人、病人或公共安全等多样化的需求，显著超越了市面上大多数单一场景应用的检测系统。

图 31 场景一精准识别

图 32 场景二精准识别

图 33 场景三精准识别

2. 精准识别摔倒，其余动作不识别

本系统在动作识别上展现出一定的精确度，这使得它能够有效区分跌倒与其他类似动作（如坐下、弯腰、蹲下等）。市面上许多跌倒检测系统由于算法设计和模型训练的局限，往往在遇到非跌倒动作时出现误报，比如将坐在地板上的人错误识别为跌倒，导致大量误报。本系统在设计时特别注重细化姿态识别，通过深度学习和大量数据训练模型，使其能够识别各种不同的行为姿势并智能区分。系统能够分析人体姿态、地面接触方式等多种因素，确保只对真正的跌倒事件发出警报。这种精细的辨别能力大幅减少了误报，增加了系统在实际应用中的可靠性，让家属和监护人员不再因频繁误报而疲于应对，从而提高了系统的使用体验。

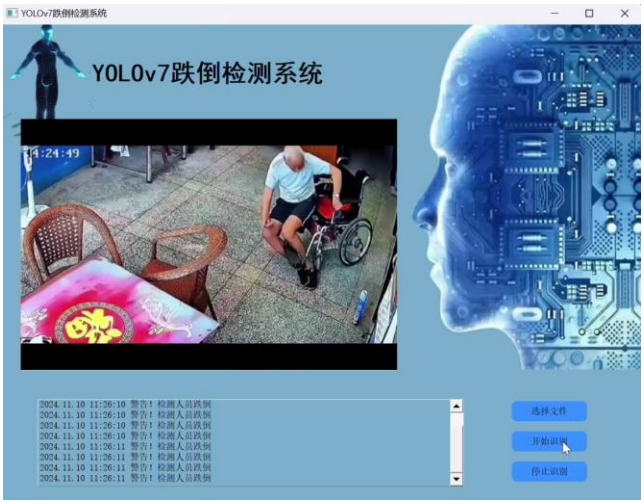

图 34 坐着不识别

图 35 倚着不识别

3. 多人干扰，精准识别摔倒者

在多人、多物体干扰的复杂环境中，本系统依然能够精准识别跌倒事件。尤其是在人员密集或有多种干扰因素的场景中，许多现有检测系统难以准确检测跌倒情况，往往会因环境复杂而误判或漏报。本系统通过多目标过滤技术，能够有效排除其他人员、宠物甚至动态背景的干扰，仅聚焦于真正的跌倒对象，确保检测的精准性。例如，在人流密集的街道或家庭场景中，系统可以忽略宠物或其他行人的干扰，专注于识别目标人员的异常状态。这种差异化识别能力保证了在复杂背景下的检测准确性，减少了误报，提高了安全监控的有效性和可信度。

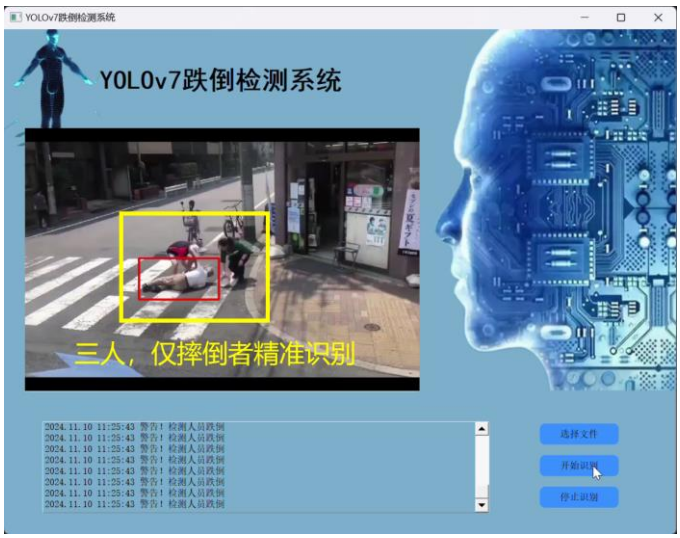

图 36 多人干扰，精准识别情况 1

图 37 多人、物干扰，精准识别情况 2

第四部分　总结

4.1 可扩展之处

（1）模型量化：利用 Vitis AI 对 ONNX 模型进行 INT8 量化，以降低计算需求，从而进一步提升推理速度。这种量化处理可以有效减少模型计算复杂度，适合资源受限的 FPGA 环境。

（2）优化数据传输：在 FPGA 内部使用 DMA（Direct Memory Access）传输数据。这种方法能显著降低系统延迟，提高数据传输效率，适用于需要快速响应的实时监控系统。

（3）精简 PetaLinux 系统：通过禁用不必要的服务和进程，减少系统资源消耗，提升整体系统的实时性和稳定性，从而使系统更适合在实时监控场景下的嵌入式应用。

4.2 心得体会

在本项目的开发过程中，通过构建基于 ZYNQ-7020 FPGA 的老人摔倒检测系统，我深刻体会到了软硬件协同优化在高效嵌入式系统设计中的关键性。该项目不仅让我对 FPGA 硬件架构有了更深入的理解，还使我更了解目标检测算法的实际应用效果。通过在 ZYNQ-7020 FPGA 上部署 YOLOv7 模型，成功地将卷积计算等高强度任务交给可编程逻辑部分（PL）进行硬件加速，大大提升了检测速度和实时性，让我切实感受到 FPGA 在满足高实时性要求应用中的独特优势。此外，通过软硬件协同设计，我们能够将计算密集型任务交给 FPGA 的 PL 部分处理，而数据管理和系统控制则由处理系统（PS）负责，从而实现了高效的软硬件结合，极大地提升了系统性能并降低了响应延迟，为实时监控应用奠定了坚实基础。在实现过程中，YOLOv7 模型的剪枝和量化成为优化的重要一环，通过减少模型复杂度和降低计算精度，我们成功地在资源受限的 FPGA 平台上保持了较高的检测精度。这一过程使我更加理解算法优化在嵌入式 AI 应用中的重要性。在系统功耗管理方面，采用动态功耗管理和中断机制，在保障系统长时间稳定运行的同时有效降低了能耗，这种动态调整的策略加深了我对嵌入式设备低功耗设计的理解。此外，在项目开发中，我也遇到了一些技术难题，例如摄像头的实时

数据传输、FPGA 端与上位机的通信以及如何平衡模型精度和实时性能。通过查阅资料和不断试错，我逐渐掌握了 DMA 数据传输、界面交互设计和 Linux 系统配置等多项技术，增强了解决复杂嵌入式系统问题的信心和能力。总体而言，本项目让我收获良多，软硬件的深度融合设计、低功耗优化、模型量化和实时性能提升等方面的探索不仅提升了我的专业技能，也让我更加体会到嵌入式 AI 系统开发的复杂性和成就感。这次经历为未来参与智能监控、物联网等领域的开发奠定了良好的基础，并激励我继续探索技术的创新应用。

第五部分　参考文献

[1] Zhang, Z., Liu, Y., & Jin, Y. (2017). "A Survey on FPGA-Based Deep Learning Accelerators: Challenges and Opportunities." IEEE Transactions on Neural Networks and Learning Systems, 30(7), 1955–1972.

[2] Jiang, W., Chen, X., & Chen, X. (2020). "Energy-efficient YOLOv3 Implementation on FPGA for Real-Time Object Detection." IEEE Transactions on Circuits and Systems for Video Technology, 30(12), 4861–4873.

[3] Sze, V., Chen, Y.-H., Yang, T.-J., & Emer, J. S. (2017). "Efficient Processing of Deep Neural Networks: A Tutorial and Survey." Proceedings of the IEEE, 105(12), 2295–2329.

[4] Li, S., Liu, X., & Wu, X. (2018). "Object Detection on Embedded Systems and FPGA Accelerators." Journal of Signal Processing Systems, 90, 217–229.

[5] Redmon, J., & Farhadi, A. (2018). "YOLOv3: An Incremental Improvement." arXiv preprint arXiv:1804.02767. Available at: https://arxiv.org/abs/1804.02767.

[6] Simonyan, K., & Zisserman, A. (2014). "Very Deep Convolutional Networks for Large-Scale Image Recognition." arXiv preprint arXiv:1409.1556. Available at: https://arxiv.org/abs/1409.1556.

[7] Gao, Z., Yu, G., & Liu, C. (2019). "Real-Time Object Detection on Embedded FPGA Platforms Using Lightweight YOLOv3 Model." IEEE Access, 7, 68261–68271.

[8] Lane, N. D., Bhattacharya, S., & Georgiev, P. (2015). "DeepEar: Robust Smartphone Audio Sensing in Unconstrained Acoustic Environments Using Deep

Learning." ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 1(2), 1-24.

[9] Liu, D., Zhang, T., & Yang, Y. (2020). "Efficient and Compact YOLO Model Deployment on FPGA for Real-Time Object Detection." IEEE Transactions on Circuits and Systems for Video Technology, 30(12), 4951–4963.

[10] Han, S., Pool, J., Tran, J., & Dally, W. J. (2015). "Learning both Weights and Connections for Efficient Neural Networks." Advances in Neural Information Processing Systems (NeurIPS), 28, 1135–1143.

[11] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). "ImageNet Classification with Deep Convolutional Neural Networks." Advances in Neural Information Processing Systems (NeurIPS), 25, 1097–1105.

[12] Yu, H., Xie, Y., & Shao, H. (2021). "An Improved YOLO Algorithm for Embedded AI Systems." IEEE Transactions on Instrumentation and Measurement, 70, 1–9.

第六部分　附录

Yolov7 重要代码

def det_yolov7(info1): 从实时流中获取帧，调用 predict 方法对每一帧进行推理。

```
global model, stride, names, pt, jit, onnx, engine
if str(info1) != '0':
    if info1[-3:] in ['jpg','png','jpeg','tif','bmp']:
        image = cv2.imread(info1)  # 读取识别对象
        try:
            results = detector.predict(info1)
            for i in results:
                box = i[1]
                p1, p2 = (int(box[0]), int(box[1])), (int(box[2]), int(box[3]))
                color = [255,0,0]
```

```
                    if i[0] == 'fall':
                        color = [0,0,255]
                        ui.printf(str(time.strftime('%Y.%m.%d  %H:%M:%S  ',
time.localtime(time.time()))) + '警告！检测人员跌倒')
                        cv2.rectangle(image,  p1,  p2,  color,  thickness=3,
lineType=cv2.LINE_AA)
            except:
                pass
            ui.showimg(image)
        if info1[-3:] in ['mp4','avi']:
            capture = cv2.VideoCapture(info1)
            while True:
                _, image = capture.read()
                if image is None:
                    break
                cv2.imwrite('./save.png', image)
                try:
                    results = detector.predict('./save.png')
                    for i in results:
                        box = i[1]
                        p1,  p2  =  (int(box[0]),  int(box[1])),  (int(box[2]),
int(box[3]))
                        if i[0] == 'fall':
                            color = [0, 0, 255]

ui.printf(str(time.strftime('%Y.%m.%d %H:%M:%S ', time.localtime(time.time()))) + '
警告！检测人员跌倒')
                        cv2.rectangle(image,  p1,  p2,  color,  thickness=3,
```

lineType=cv2.LINE_AA)

except:

pass

ui.showimg(image)

QApplication.processEvents()

else:

capture = cv2.VideoCapture(0)

while True:

_, image = capture.read()

if image is None:

break

cv2.imwrite('./save.png', image)

try:

results = detector.predict('./save.png')

for i in results:

box = i[1]

p1, p2 = (int(box[0]), int(box[1])), (int(box[2]), int(box[3]))

if i[0] == 'fall':

color = [0, 0, 255]

ui.printf(str(time.strftime('%Y.%m.%d %H:%M:%S ', time.localtime(time.time()))) + '警告！检测人员跌倒')

cv2.rectangle(image, p1, p2, color, thickness=3, lineType=cv2.LINE_AA)

except:

pass

ui.showimg(image)

QApplication.processEvents()

def plot_one_box(x, img, color=None, label=None, line_thickness=3): 在图像

上绘制检测框，并标注出检测的类别。

```
# Plots one bounding box on image img
tl = line_thickness or round(0.002 * (img.shape[0] + img.shape[1]) / 2) + 1  # line/font thickness
color = color or [random.randint(0, 255) for _ in range(3)]
c1, c2 = (int(x[0]), int(x[1])), (int(x[2]), int(x[3]))
cv2.rectangle(img, c1, c2, color, thickness=tl, lineType=cv2.LINE_AA)
if label:
    tf = max(tl - 1, 1)  # font thickness
    t_size = cv2.getTextSize(label, 0, fontScale=tl / 3, thickness=tf)[0]
    c2 = c1[0] + t_size[0], c1[1] - t_size[1] - 3
    cv2.rectangle(img, c1, c2, color, -1, cv2.LINE_AA)  # filled
    cv2.putText(img, label, (c1[0], c1[1] - 2), 0, tl / 3, [225, 255, 255], thickness=tf, lineType=cv2.LINE_AA)
return c1, c2
```

def predict(self, source): predict 方法处理视频帧，通过非极大值抑制 (NMS) 输出检测结果。

```
# Load images
dataset = LoadImages(source, img_size=self.img_size, stride=int(self.model.stride.max()))
# Get class names
names = self.model.module.names if hasattr(self.model, 'module') else self.model.names


for path, img, im0s, _ in dataset:
    img = torch.from_numpy(img).to(self.device)
    img = img.float() / 255.0  # scale images to [0, 1]
    if img.ndimension() == 3:
```

```
            img = img.unsqueeze(0)
        # Inference
        pred = self.model(img, augment=self.augment)[0]
        # Apply NMS
        pred = non_max_suppression(pred, self.conf_thres, self.iou_thres,
classes=self.classes, agnostic=self.agnostic_nms)
        results = []
        for i, det in enumerate(pred):  # detections per image
            if len(det):
                # Rescale boxes
                det[:, :4] = scale_coords(img.shape[2:], det[:, :4],
im0s.shape).round()
                for *xyxy, conf, cls in reversed(det):
                    # Add bbox to image
                    label = f'{names[int(cls)]}'
                    plot_one_box(xyxy, im0s, label=label, color=[255, 0, 0],
line_thickness=3)
                    results.append([label, xyxy, conf.item()])
        # Display image
        return results
```