基于 Zynq 硬件加速 RISC-V 软核控制的水下机器人

刘宇昂 20202261064；王傲涛 20202261024；郭思睿 20202261030

第一部分　设计概述

1.1 设计目的

随着水下机器人（ROV/AUV）技术的发展，ROV 越来越多代替人类进行水下作业任务。传统水下设备基于声呐，成本高、数据难以解读，导致 ROV 成本激增；而 CMOS 传感器感知方案成本较低、直观明了，适用于光照充足浅水环境下的 ROV/AUV。为降低成本，ROV 舱内体积较小，主控电路需要尽可能集成化，SoC 常常被用作低成本 ROV 的控制核心。本项目旨在基于 XC7Z010 芯片实现近海浅水作业型遥控水下机器人图像传输、姿态控制、水下视觉/控制算法加速三位一体的解决方案。

1.2 应用领域

本项目可应用于小型（40kg 以下）、中型（40kg~100kg）乃至大型（大于 100kg）水下作业机器人，能够执行 ROV/AUV 实时控制和水下视觉算法。机器人只需具有基于电力载波的上位机通讯方式即可使用本系统替换原本的舱载主控 SoC 或 MCU。

本项目部署在一台树莓派-电力载波-PC 结构的教学用水下机器人上，预期根据实际项目进度适配一台 45kg 级别的中型水下作业机器人。

1.3 主要技术特点

不同于深水 ROV，浅水 ROV 主要为商用，因此需要尽可能降低技术、物料成本；浅水 ROV 多运行在光照充足、水质相对清澈的海域或河道，无需超短基线定位、惯性导航、多普勒测速仪、前探声呐等设备；使用低压供电即可满足用户水产捕捞、旅行娱乐需求而不需要像深水 ROV 那样使用油箱供电。在早期研究[1]中，已经实现了基于 ZYNQ 设备的深水 ROV，并引入了部署在嵌入式 Linux 平台的智能控制算法。本项目针对逐渐兴起的低成本商用浅水 ROV 设计的 ZYNQ-7010 解决方案能够，利用全可编程 SoC 单片取代中传统 ROV 中常见的高性能 ARM+高实时性 MCU 架构，大大缩减控制系统占用面积，留出更多舱内空间供传感器、锂电池/镍氢电池、电池管理系统使用。

1.4 关键性能指标

表 1.4.1　教学机器人参数

最大工作深度	100m

主体尺寸	338*324*210mm
重量	15kg
推进器最大推力	-5~+7.5kgf
供电电源	24V 电力载波脉动直流
传感器配置	水深、水温、九轴姿态

表 1.4.2　主控板参数

长度	150mm
宽度	70mm
核心板接口	SODIMM-DDR3
板层与板材	4 层板-FR4
接口	电力载波、推进器 PWM、USB、DVP
外设	九轴姿态传感器、GPIO、以太网、电力载波模块、USB、DVP

表 1.4.3　核心板参数

主控芯片	XC7Z010-CLG400
内存	DDR3 512MB
扩展存储	SPI Flash、eMMC、SD 卡
接口	千兆以太网、USB2.0

表 1.4.4　脉动阵列加速器参数

LUT-Logic 资源占用	4*1259
LUT-FF 资源占用	4*1254
DSP 资源占用（预期）	36/0（测试中未使用 DSP 硬核）
Block RAM 资源占用	0
适配算子	Conv(3x3)、Conv(1x1)、ReLU(CBR)
时钟频率（预期）	100MHz
加速比 Sp	(n*(n-1)+2)/((2*n-1)+2)

表 1.4.5　实时控制 SoC 参数

总 LUT 资源占用（预期）	8000~9000
总 DSP 资源占用（预期）	9
总 Block RAM 资源占用（预期）	30~40
CPU-LUT 资源占用（预期）	2000~4000（参考量 3232）
CPU-DSP 资源占用（预期）	4
CPU-Block RAM 资源占用（预期）	16
CPU 流水线	3 级流水：取指-译码-执行

CPU 超标量	标量处理器，双发射顺序执行
内核外设	内核定时器、外部中断控制器
TCM 容量	4KB(ITCM)+8KB(DTCM)
CPU 主频（预期）	50MHz
AXI 外设主频	50MHz
AHB 外设主频	24MHz
PWM 控制器	16 路 SPWM
GPIO	32 位 GPIO，支持外部中断
I2C 控制器	支持标准 I2C 和 SCCB 协议扩展
DVP 控制器	仅支持 OV 传感器的 DVP 总线协议

1.5 主要创新点

（1） 首次实现浅水 ROV/AUV 集成 ZYNQ 单片控制系统

（2） 针对水下实时控制自研 RV32I 指令集嵌入式处理器核心，以软核形式集成在 ZYNQ 芯片的 PL 端

第二部分 系统组成及功能说明

2.1 整体介绍

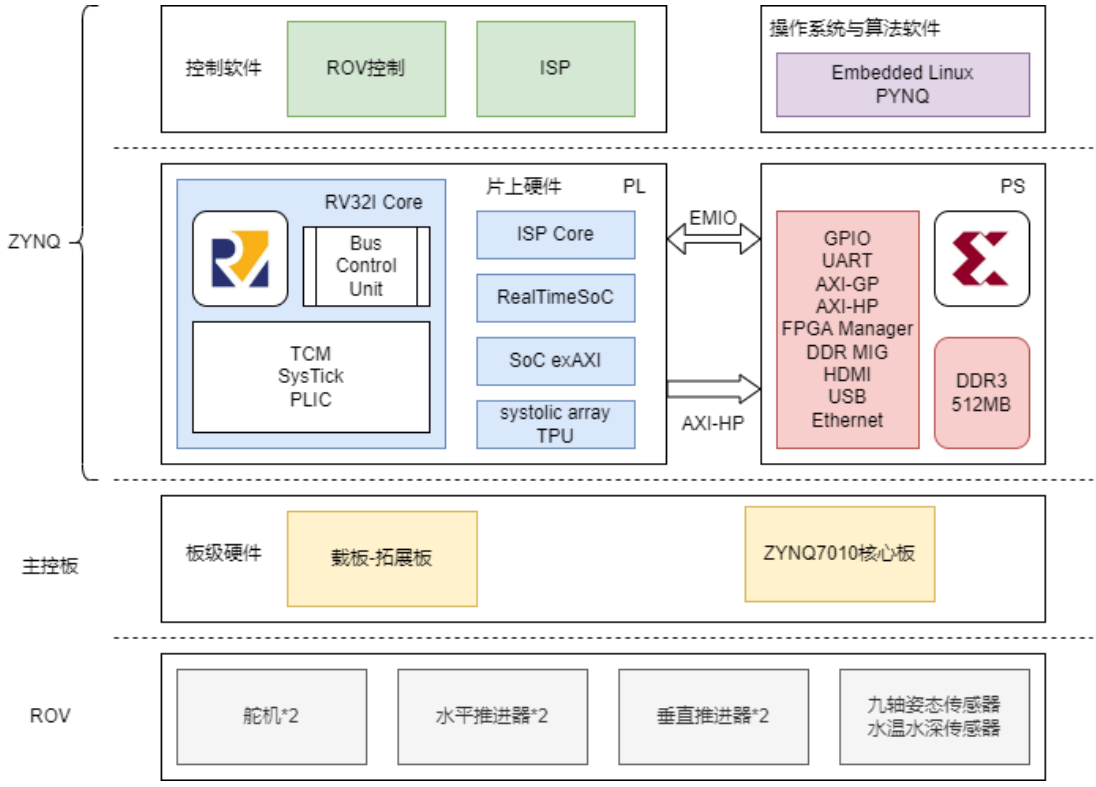

图 2.1.1 主体结构框图

本项目分为四层：板级硬件、片上硬件、控制软件、操作系统与算法软件。

2.1.1 板级硬件

根据水下机器人的规格型号进行针对性设计，本项目基于一台教学用水下机器人设计了板级硬件，集成九轴姿态传感器模组和水温、水深传感器接口，并从 ZYNQ 端引出了 IO 用于推进器和机械臂舵机控制。

2.1.2 片上硬件

利用 ZYNQ7010 的 PL 实现 RV32I 指令集的自研微架构嵌入式处理器，支持取指-译码-执行三级流水线，双发射顺序执行，支持外部中断和内核定时器，配备了 ITCM 和 DTCM 来存储基础控制程序。内核支持硬件 Booth 乘法器和软件除法运算，不支持浮点数，即完成 RISC-V 的 I 指令子集的所有功能，面向性能-面积比作优化。

PL 部分还配套设计 TCM、UART 接口控制器、I2C 接口控制器、DVP 接口控制器、VDMA、GPIO、PWM 控制器、PID 算法硬件处理器形成一整套实时控制 SoC。SoC 通过 AXI 接口读写 PS 端 DDR 数据，将视频数据实时传输给 PS 端，同时使用 EMIO 串口接收 PS 端从上位机转发的控制指令，上位机以此控制整个 ROV 设备。

片上硬件部分还集成了图像处理功能，包括两套加速器：ISP 硬件电路提供直方图均衡、边缘检测等水下图像优化功能；矩阵运算电路提供了基于脉动阵列的矩阵乘法硬件加速功能，为 PS 端提供当前卷积神经网络最常用的 3x3（depth-wise）和 1x1（point-wise）卷积运算加速。

2.1.3 控制软件

部署在 PL 端的实时控制 SoC 上，负责传感器数据接收、状态数据上传、上位机指令解析、PID 加速器调度、PWM 控制工作。在上电之后，控制软件还负责摄像头启动配置、DMA 传输配置、ISP 加速器调度。控制软件基于裸机驱动构建，采用简单的四轴 ROV 姿态控制算法。

2.1.4 操作系统与算法软件

嵌入式 Linux 操作系统部署在 PS 端，负责高性能的 USB 摄像头采集、水下图像处理、网络视频推流任务。上位机和 ROV 通过 UDP 协议通讯，UDP 数据报由嵌入式 Linux 转换成串口数据，转发到 PL 部分的 SoC；同时读取从 PL 端 VDMA 转发到 DDR 的视频数据，使用 mjpg-streamer 软件推流到上位机

嵌入式 Linux 系统中还编写了脉动阵列加速器驱动，可以调用 PL 端的卷积加速器 IP 实现 CNN 常用的 3x3 卷积操作。基于 YOLOv7 的水下图像识别算法进行 BN 融合、INT8 量化后使用 SD 卡存储网络结构和参数，在使用 PYNQ 框架的情况下可以调用 tensorflow-lite 框架执行水下图像识别算法

2.1.5 软件结构

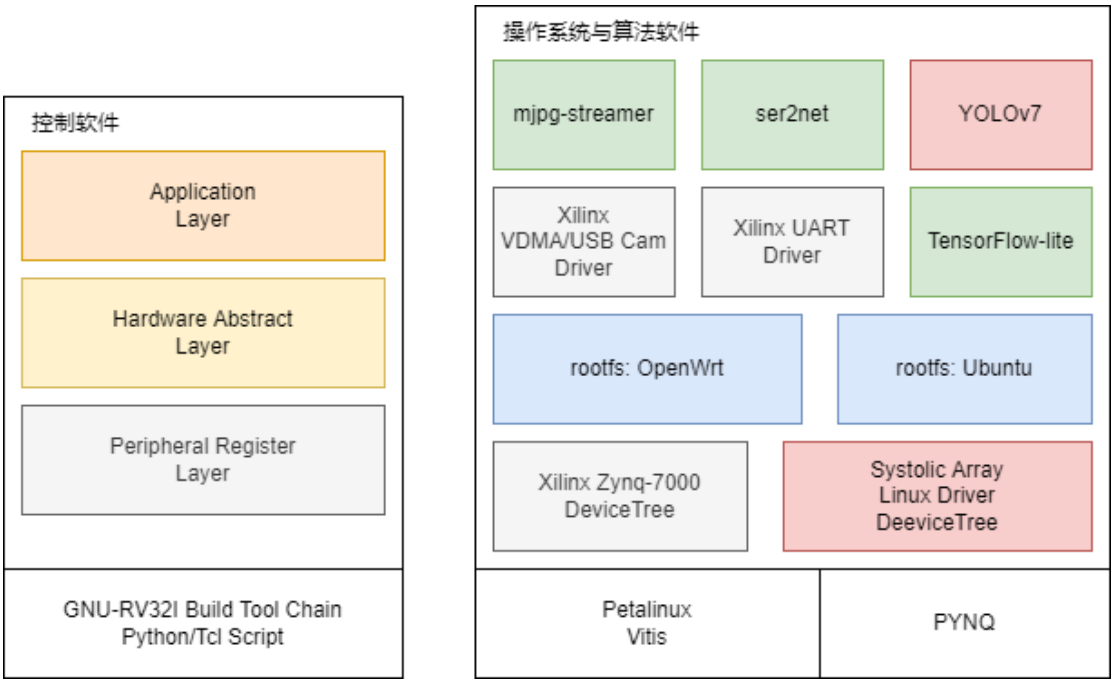

图 2.1.2 软件层次图

项目分为嵌入式端和高性能端两部分。

嵌入式端基于裸机实现了实时控制功能，分为底层外设寄存器、硬件抽象层（HAL）驱动、应用层三级。底层软件基于 Xilinx Vitis 套件生成，并针对为自行加入的 IP 编写寄存器驱动；硬件抽象层采用统一的代码规范，基于 C 面向对象编写了 ROV 所需外设类和算法 ADT，已经在多台水下机器人中得到验证，可以快速部署到 RISC-V 平台；应用层针对每台机器人所需控制功能和数据协议帧结构编写。整套软件专为浅海 ROV 设计，可以实现快速移植部署。项目使用 gnu-rv32i 工具链，基于 C 语言完成开发，使用 python 脚本生成 coe 格式的可执行文件，将程序固化入比特流进而烧录到 FPGA 中。高性能端使用 Xilinx Petalinux 套件移植嵌入式 Linux 或 PYNQ 框架完成高性能的网络推流、数据转发、神经网络算法任务。网络推流软件基于 mjpg-streamer，数据转发基于 ser2net，二者都可以通过预编译软件包来实现快速移植；为 PL 到 PS 的 DMA 图像传输编写了专用驱动程序，从而能让 mjpg-streamer 读取来自 PL 端 VDMA 的视频数据。神经网络算法则需要使 TensorFlow-lite 框架。本项目将 PYNQ 移植到设备中，使得 TensorFlow-lite 框架可以和 PL 端的脉动阵列加速器对接并进行卷积乘法加速。

2.1.6 算法结构

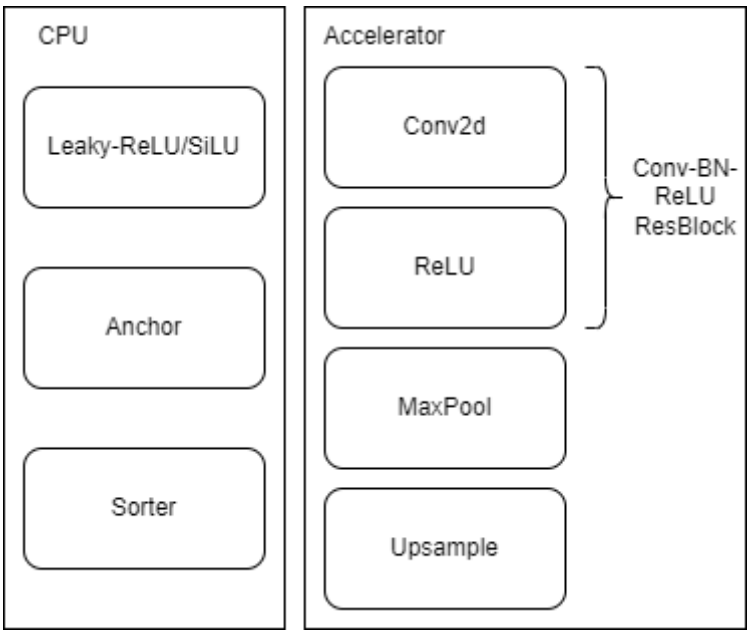

图 2.1.3　算法结构框图

项目中涉及的算法包括 YOLOv7、直方图统计、Sobel 边缘检测。YOLOv7 是典型的卷积神经网络算法，将其主体结构分成两部分，使用 PS 端的 CPU 计算难以硬件加速的 SiLU、Anchor、Sorter 结构；使用 PL 端的卷积加速器和硬件加速模块计算涉及大量矩阵运算的二维卷积、ReLU、最大池化、上采样算子。异构计算能够大大提高算法执行的能效比，并且尽可能利用 ZYNQ7010 的 PL 部分资源。直方图统计和 Sobel 边缘检测都是经典的机器视觉算法，项目中将两套算法作为 ISP（Image Signal Process，图像信号处理）的主要功能，从外界 CMOS 传感器传入的图像数据受实时控制 CPU 的控制被送入 ISP 模块，实时输出处理后的图像到 PS 端。

2.2 各模块介绍

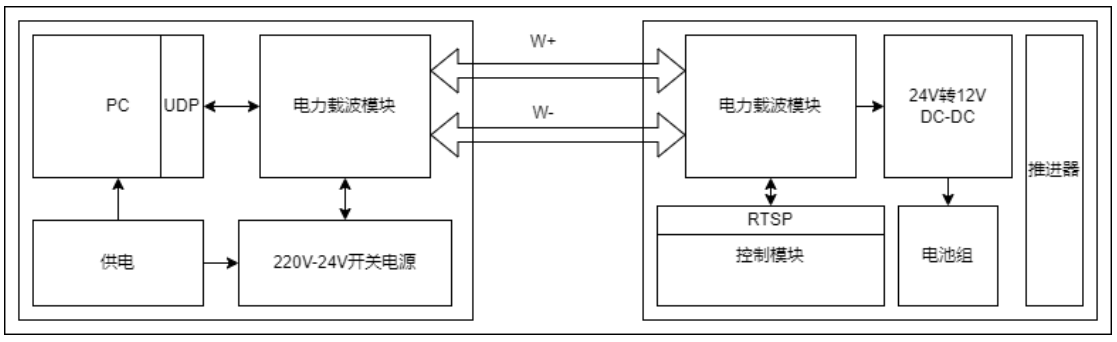

图 2.2.1　ROV 框架图

ROV 使用开放框架结构，成本更低，能有效减少水阻，提供良好正浮力支持，配备两台水平推进器和两台垂直推进器，一个舱内相机云台，一个舱外机械臂水密接插件。舱内分为上下两部分，上半部分固定项目主体电控板，下半部分安装连线转接板、回流排、动力电池组。

本项目使用电力载波通讯。相比传统的 RS232/RS485 通讯，电力载波通讯速率更高、使用线缆更少，可以用两根线同时传输电力、控制信号，通过电力载波调制解调器，还可以在电力线上叠加以太网信号。基于以太网-电力载波模块可以实现上位机和 ROV 之间的 UDP 数据传输，并能够基于 rtsp/http 协议将 ROV 采集的视频数据推流到上位机。

目前很多深水 ROV/AUV 采用光纤通讯的方式，可以在千米级深度完成视频传输和实时通信[2]。相比于光纤通讯，电力载波通讯稳定性略低，但光纤通讯还需要使用复杂的光端机，换用电力载波后成本显著减少，且能避免光纤不易弯折的问题，能够使用柔性零浮力线缆。

主控电路板使用 Altium Designer 绘制：

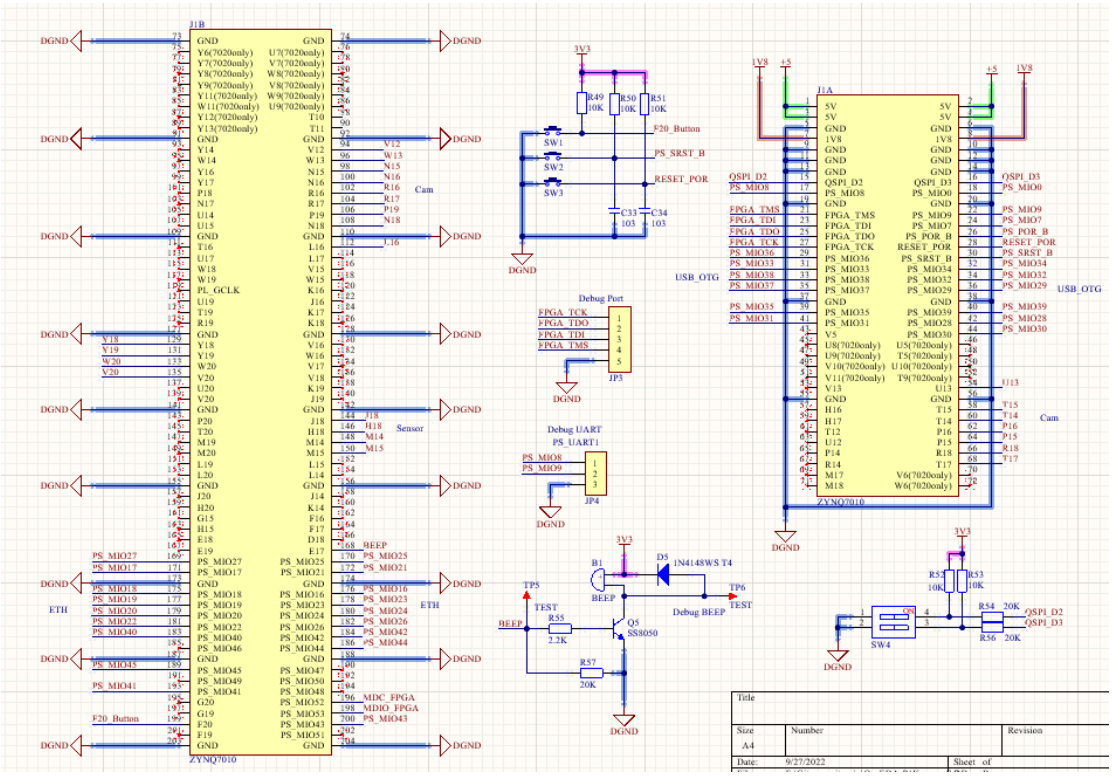

图 2.2.2　核心板部分原理图

ZYNQ-7010 核心板使用 ICQMCU 的 XC7Z010 核心板，SODIMM-DDR3 封装。上图为核心板封装与按键、蜂鸣器、启动拨码开关。

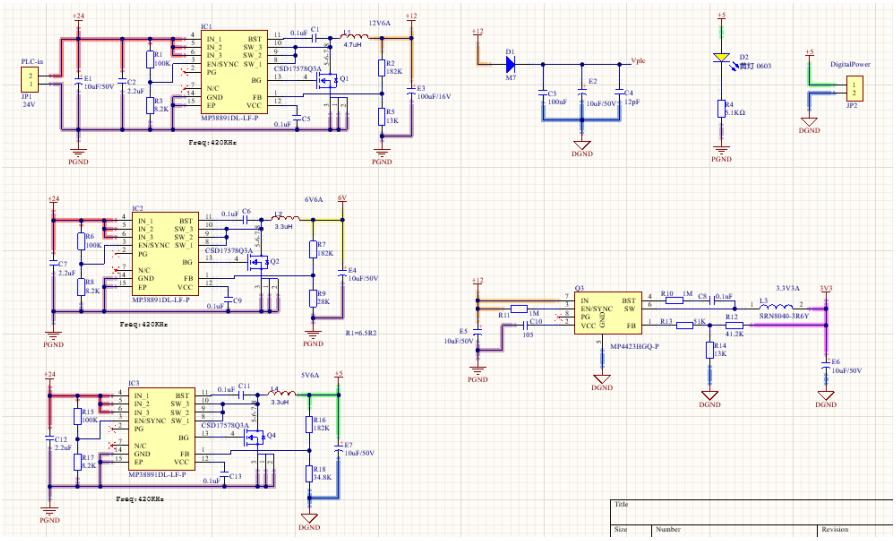

图 2.2.3　整版供电原理图

上图为整版供电，24V 输入后分别转出 12V 和 6V 供给推进器和云台舵机、机械臂，另有 5V 提供给核心板，3.3V 提供给传感器等其他板上外设。

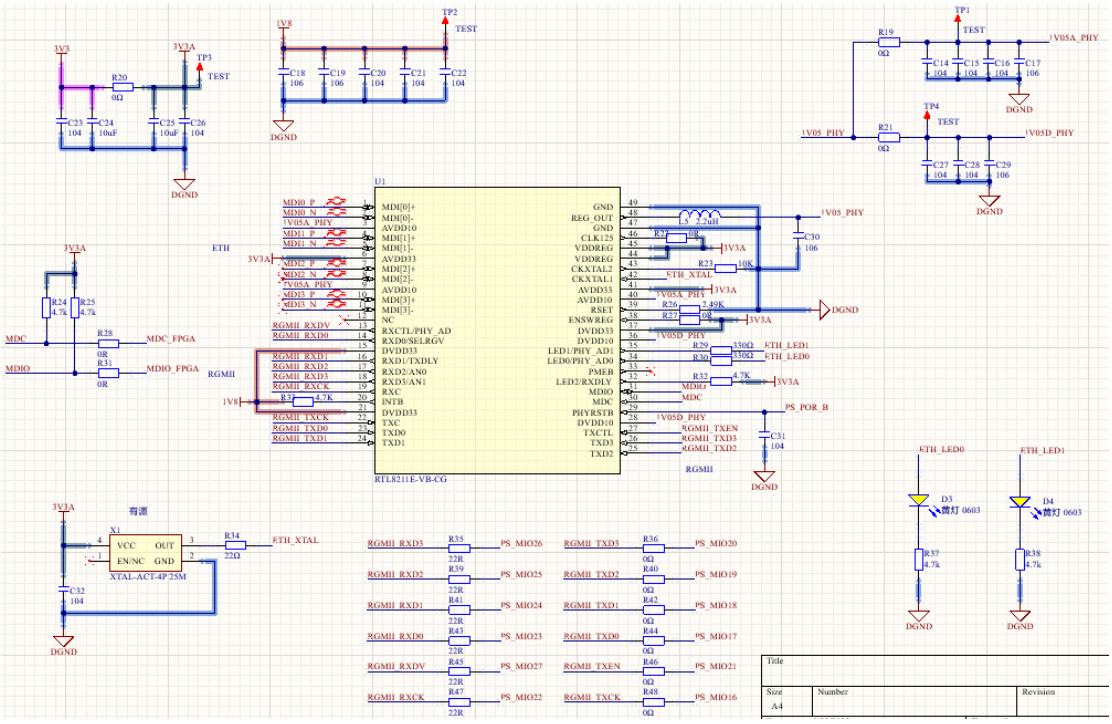

图 2.2.4　网卡部分原理图

1000M/100M 自适应以太网卡，使用 RGMII 与 ZYNQ 通信。

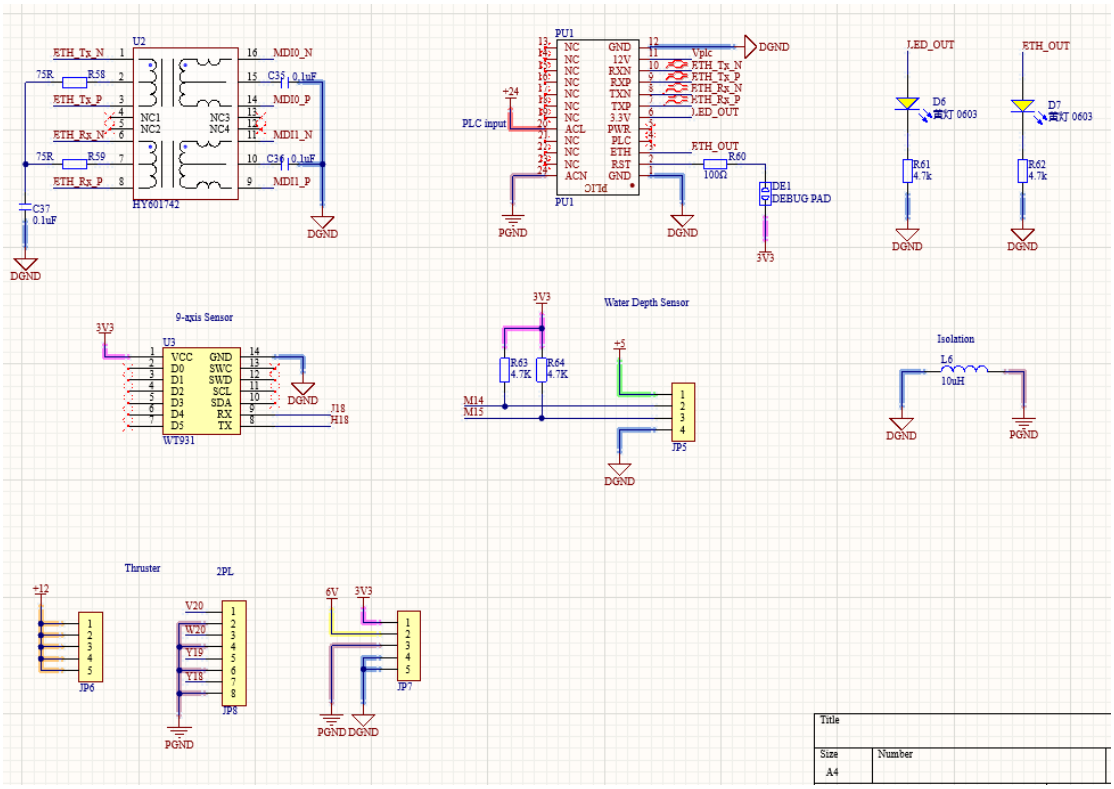

图 2.2.5　信号线接口与网口部分原理图

分别从 PL 端 bank0、bank1 引出传感器和信号线接口。上方原理图还包括网口 Bob-Smith 电路和电力载波调制解调器。

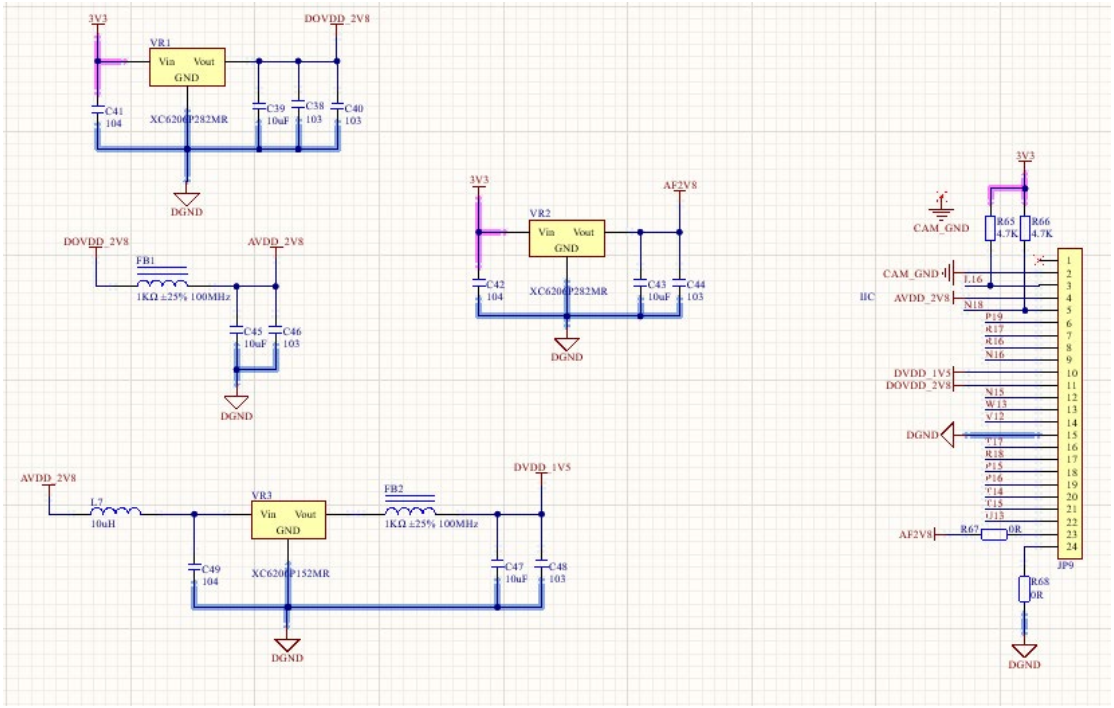

图 2.2.6　DVP 摄像头部分原理图

DVP 摄像头接口，兼容 OV5640 模块的 FPC 排线连接。

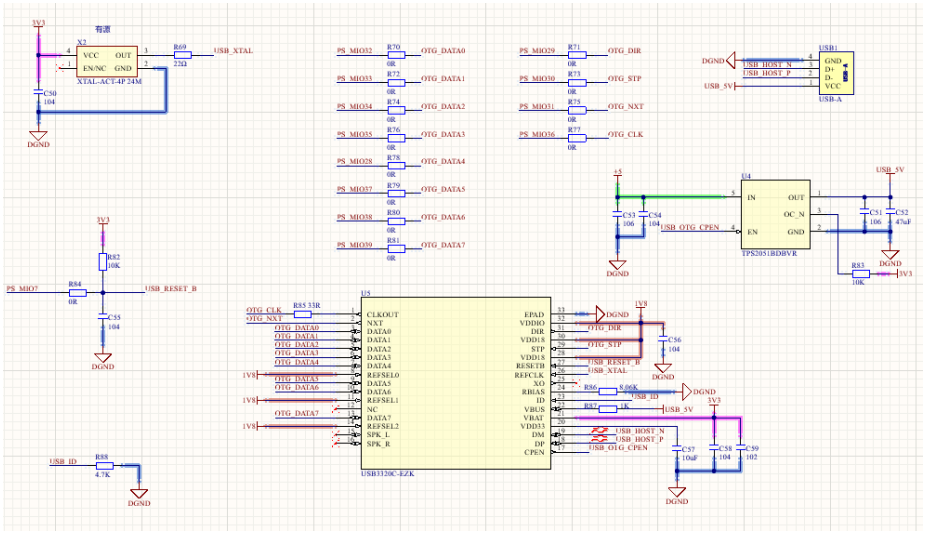

图 2.2.7　USB 部分原理图

USB2.0-PHY 芯片，使用 ULPI 协议与 ZYNQ 通信。

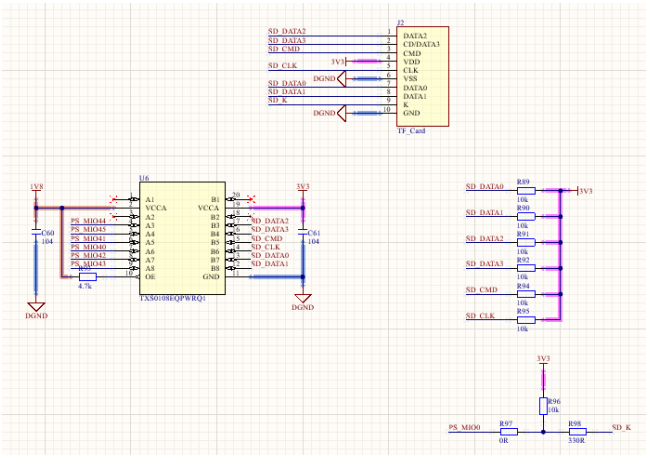

图 2.2.8　电平转换部分原理图

使用一个电平转换芯片将 ZYNQ 的 1.8V bank2 电平转换成 SD 卡可以使用的 3.3V 电平。

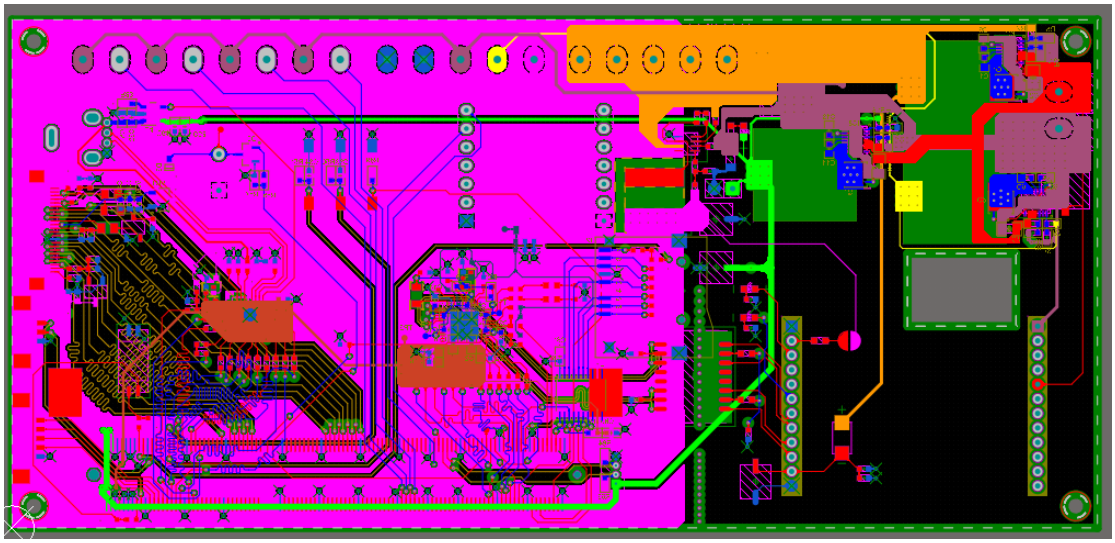

图 2.2.9　整版 PCB 设计图

整版 PCB 设计如上图所示。

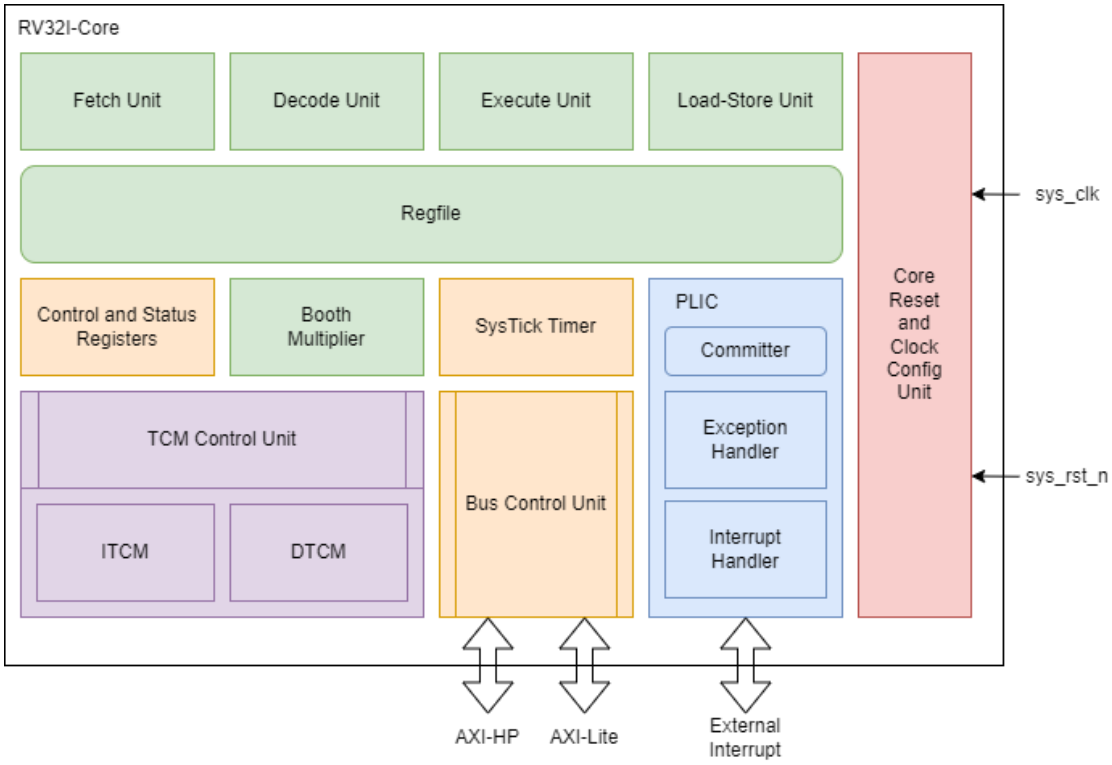

图 2.2.10　CPU 架构图

自研 RISC-V 内核为 3 级流水线双发射架构。取指单元 Fetch Unit 会从 ITCM 取指，每个周期取连续的两条指令送入译码单元 Decode Unit，这两条指令会被分别标记为指令 0 和指令 1。译码单元分成两个通路，指令 0、1 分别译码得到指令类型后送入执行单元 Execute Unit 对应的 ALU 或由写回单元 Load-Store Unit 完成访存工作。执行单元有两套相同的 ALU 电路，本内核为顺序执行，因此指令 0 一定在指令 1 之前执行，这样如果两个指令是相同类型的，那么会被分配到两个 ALU 电路中加速运算，如果两个指令类型不同，那么会按照 0 先运算，1 后运算的顺序送入同一个 ALU 电路。写回单元会根据指令 0 和 1 标号来依次写回数据到寄存器、TCM 抑或是通过总线控制单元 BusControlUnit 写回到外设。

内核实现为标准的 RV32I 指令集，并扩展了 M 指令子集中的乘法指令，但没有实现除法器，因此不支持除法指令。乘法器基于 Booth 快速乘法实现。I 指令子集中规定了处理器的中断和内核定时器，项目中也进行实现，支持 32 位的高精度内核定时器和 16 位用户自定义中断，并根据指令集要求实现了内核异常处理。中断控制器 PLIC 不支持嵌套中断或衔尾中断，只支持最基本的中断/异常处理，所有外部中断由 Interrupt Handler 模块接收，所有异常由 Exception Handler 模块接收，二者中 Exception Handler 优先级更高。PLIC 通过 Committer 模块将中断发送到内核，屏蔽中断也由 Committer 模块完成。

由于本 SoC 部署在 ZYNQ 的 PL 端，因此选择 Block RAM 作为 TCM 的基础硬件实现，分别例化 4KB 和 8KB 的 Block RAM 作为 ITCM 和 DTCM。内核中集成了总线控制单元 Bus Control Unit 用于 AXI-HP、AXI-Lite 总线的控制。内核只能作为主设备，访问 PS 端的 DDR 内存空间（被映射到一片固定的本地内存空间）和外设的寄存器空间（同样被映射到本地内存空间）。内核还集成了一套同步复位控制器和紧耦合时钟控制器，可以为内核外设提供适合 FPGA 的异步复位同步释放功能和更好的时钟时序。

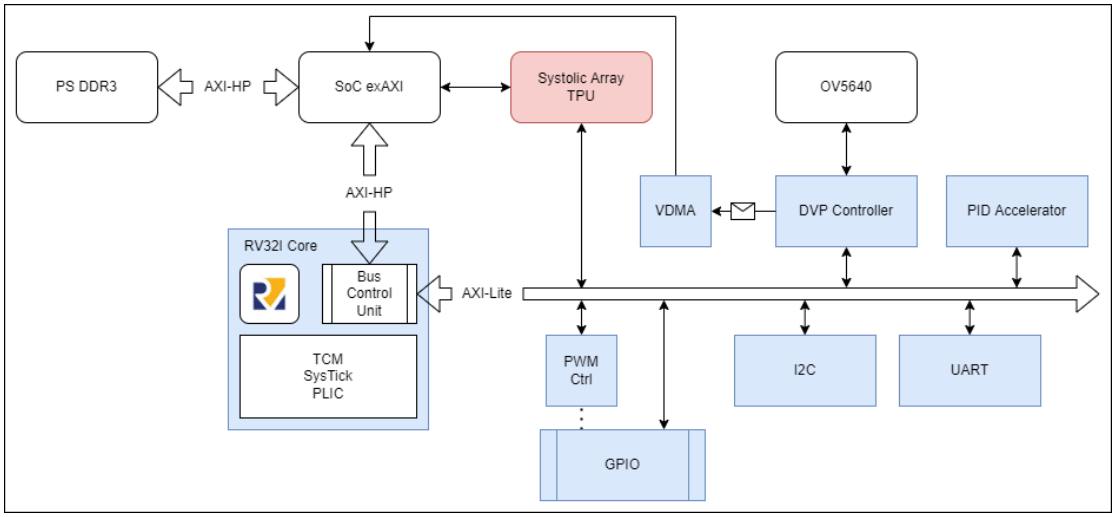

图 2.2.11　SoC 结构框图

基于 RV32I 指令集的自研微架构嵌入式处理器通过 AXI-Lite 总线控制外围设备，通过 AXI-HP 总线与 PS 通讯并能够存取 DDR3 中的数据。

AXI-Lite 总线上挂载有 PWM 控制模块、GPIO、IIC 通讯模块、UART 通讯模块、PID 加速器、DVP 控制模块、脉动阵列卷积加速模块。GPIO 外设支持外部中断和端口复用功能，可选择通过 AXI-Lite 总线互联映射寄存器到内核空间；也可以选择配置为复用模式，连接 PWM 控制器输出 PWM 信号。IIC 与 UART 则是 SoC 的两种协议接口外设，收取板载传感器数据并且通过 EMIO 接口与 PS 端进行数据交互。PID 加速器是一个简单的状态机，在使能状态下，读取 CPU 给定量，自动进行 PID 计算，向 PWM 控制器输出一组调节量，从而调节推进器的工作状态完成定向（Orbit）或定深（Hover）功能。DVP 控制器是摄像头图像数据的调度模块，接收来自 OV5640 CMOS 的图像数据传输至 VDMA 模块，VDMA 对帧数据缓存（同步 FIFO），以数据流的形式输出到 exAXI 总线互联。通过配置数据复用寄存器（DAR），还可以将 VDMA 数据传输到 exAXI 或旁路 ISP 模块，ISP 模块未在框图中绘出，在下文中详细介绍。

exAXI 总线互联是一个有数据分配功能的 AXI-HP-Lite 总线桥，综合调度来自 MCU、VDMA、加速模块的数据。MCU 作为主设备时，exAXI 允许

CPU 访问 PS 端的 DDR，CPU 通过内部 BCU 来完成访存，此时 TCM 作为 CPU 一级缓存，DDR 充当内存；VDMA 作主设备时，exAXI 调度数据通过 AXI-HP（Stream 模式）送入 PS 端 DDR3；脉动阵列作为主设备时，能够通过 exAXI 读写 DDR3 内指定段的数据。除此之外，exAXI 总线互联还允许脉动阵列加速模块的数据发送至 RV32I 核，此时内核作为主设备主动完成数据读写。

脉动阵列加速模块分别连接到 exAXI 总线互联和 AXI-Lite，分别用于数据传输和控制交互。RV32I 核的指令数据通过 AXI-Lite 发送给脉动卷积加速模块。PS 端指令数据通过 AXI-HP 发送到 exAXI，再由 exAXI 传输给加速模块。这是为了高性能运算和实时控制功能隔离。

CNN 中的大量 3x3 卷积运算[3]需要使用到大量重复的乘加算法，这些运算都可以归纳成矩阵计算。通用处理器，即使搭载了向量计算组件，也很难实时计算数百层卷积，因为 CPU 设备难以进行数据流复用，因此传统上我们使用 GPGPU 而不是 CPU 来进行卷积神经网络的运算加速。在 ROV 舱内，受空间限制很难安装大体积的 GPU 加速卡，因此团队借鉴了 Google TPUv1[4]中提出的脉动阵列（Systolic Array）矩阵计算单元作为神经网络加速的核心，部署在 ZYNQ-7010 的 PL 端。

团队根据 TPU 中的核心计算模块复现了一个简单版本的卷积加速器，每个单元能够实现任意大小的矩阵运算，可以根据需求改写例化参数来实现模块复用。本项目中例化了 4 个 3x3 的 8 位宽数据脉动阵列核，由外围电路和软件控制输入数据，可以按需求完成 1x1、3x3、5x5 大小的 INT8 卷积计算。每个 3x3 卷积单元的结构如下图所示。

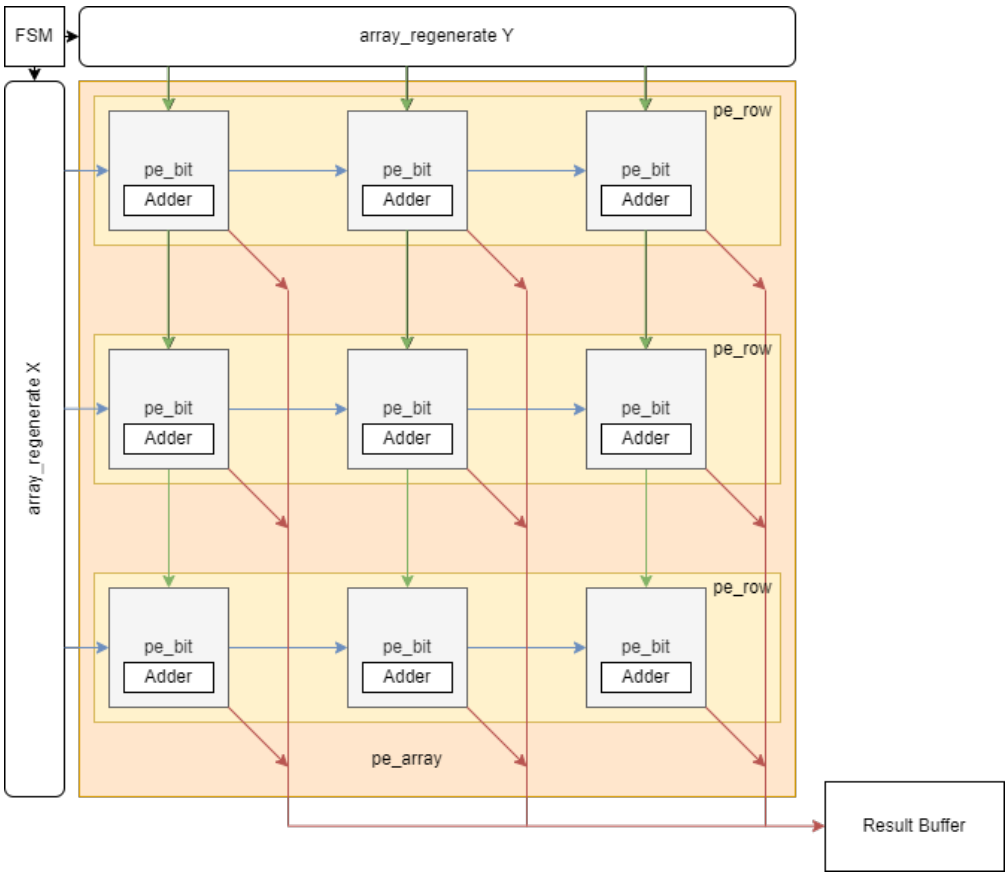

图 2.2.12　脉动阵列结构框图

核心元件为 pe_bit 模块，每个 PE 含有一个乘法器和一个加法器，可用一个乘加器来替代。PE 具有横纵两个方向的输入输出，每个周期都会从左侧和上侧读入数据，相乘后与本单元已有的数据相加得到部分和，并让数据正常从右侧和下侧输出。三个 PE 横向排一个 pe_row 模块，复用左侧传入数据，三个 pe_row 模块纵向排成一个 pe_array 模块，复用上侧传入数据。运算结果保存在对应 pe 内部的寄存器，由外部控制逻辑选择输出到缓存区。本加速器为加载-执行两级流水。针对低功耗、低资源占用的 FPGA 场景，没有使用单独的状态机来完成矩阵重排，直接使用硬连线逻辑完成数据加载和运算的流水衔接，一个 n 行 n 列的矩阵只需要 2n+1 个周期传入，在加载数据的同时，模块还会进行同步完成计算。

计算过程如下图所示：

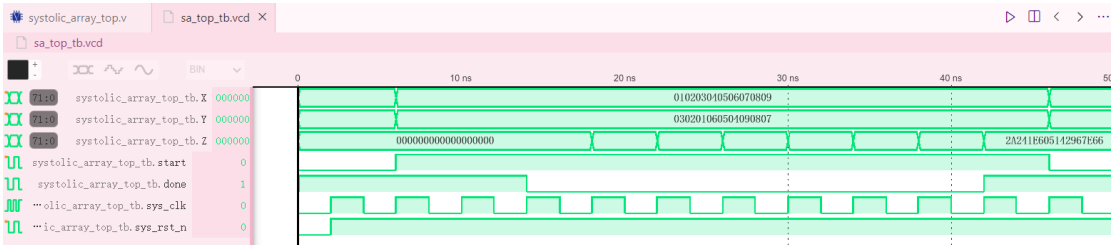

图 2.2.13　3x3 矩阵乘法计算示意图

这里仿真了 3x3 矩阵乘法，可以发现只需要使用 2+7 个时钟周期就可以

完成一次 3x3 的矩阵点乘，其中前两个周期用于加载数据和等待系统复位，属于固定开销，随着矩阵规模增大，运算效率还会提高。

加速器支持多通道卷积和 padding 功能，全部基于硬连线完成，需要在综合前修改例化参数完成定制。

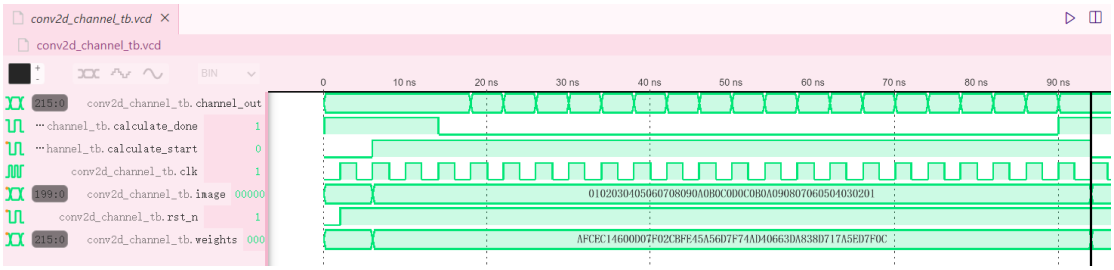

图 2.2.14　5x5 矩阵乘法计算示意图

上图仿真了 5x5 卷积计算，只需要使用 22 个时钟周期即可完成无 padding 单通道 5x5 卷积运算。

加速器还包含了 ReLU 计算模块和一个全展开矩阵加法模块，卷积处理后根据外围控制电路可直接进行 ReLU 和 Concat 运算，在完成 BN 融合的前提下，能够直接处理 depth-wize[5]的 Res-CBR 块。

项目预期构建基于 YOLOv7[6]的水下图像识别系统，因此例化的四个核中三个用于 E-ELAN 块，一个用于 REP 块，由 Linux 端进行调度。由于数据量较大，团队选择直接在 DDR 中开辟 Share-Buffer 供加速器使用，PS 通过 AXI 总线控制加速器，并在 DDR 固定区域中存入待运算数据；加速器取出数据后立刻计算并将结果保存到 DDR 中的固定地址，使用中断通知 PS 端去出数据。

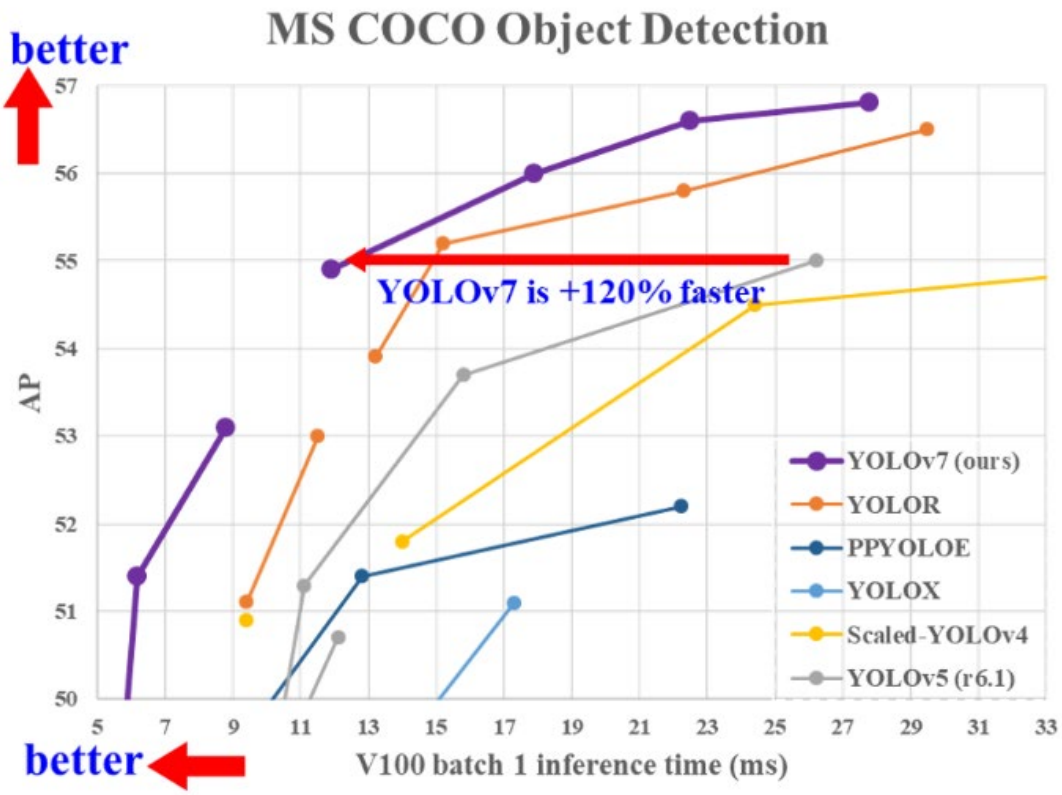

图 2.2.15　YOLOv7 算法结构图

YOLOv7 是 YOLOv4 原作者团队在 2022 年提出的新一代 YOLO 目标识别算法，达到了比以往 YOLO 算法更快、更小、更低开销、更精确的目标。由于 YOLOv7 体积相较 YOLOv5-n 显著缩小，计算开销更低，因此选用该算法作为水下目标检测的主要算法。

项目使用 DUO 数据集训练，利用 PyTorch 的 BN 融合与 INT8 量化 API 进行算法优化。基于 PYNQ 框架快速部署到 ZYNQ-7010 片上。

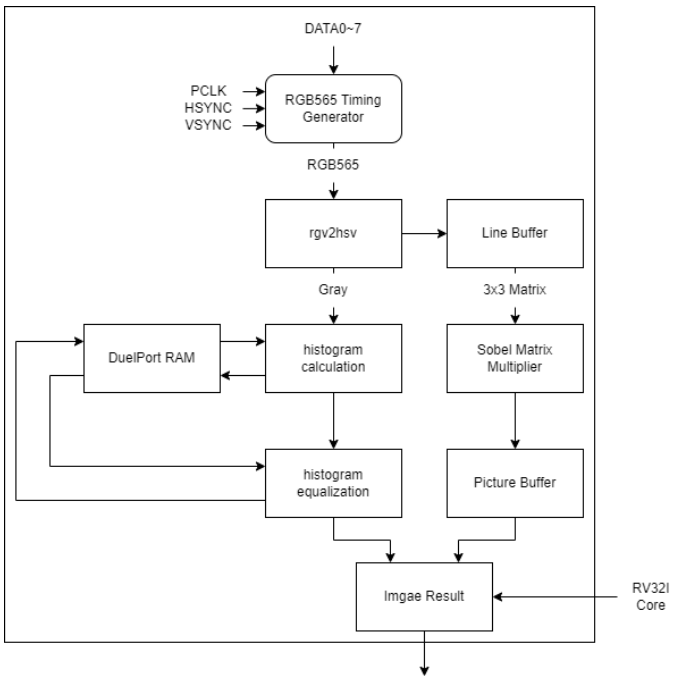

图 2.2.16　ISP IP 结构框图

PL 部分的 ISP IP 使用一个 RGB565 时序发生器模块从 VDMA 连续读入数据，以标准的 RGB565 格式输出到 rgb2hsv 模块，提取其中灰度值进行直方图均衡化。通过一个 Line Buffer 缓存 2 帧数据后连续将数据与 Sobel 算子进行卷积，得到边缘检测图像。直方图均衡图像能够优化水下的图像表现——锐化水下目标；边缘检测图像则能为 ROV 操作者提供更好的避障指示。两种数据根据实时控制 CPU 的指令输出到 PS 端 DDR 中的固定内存区域。

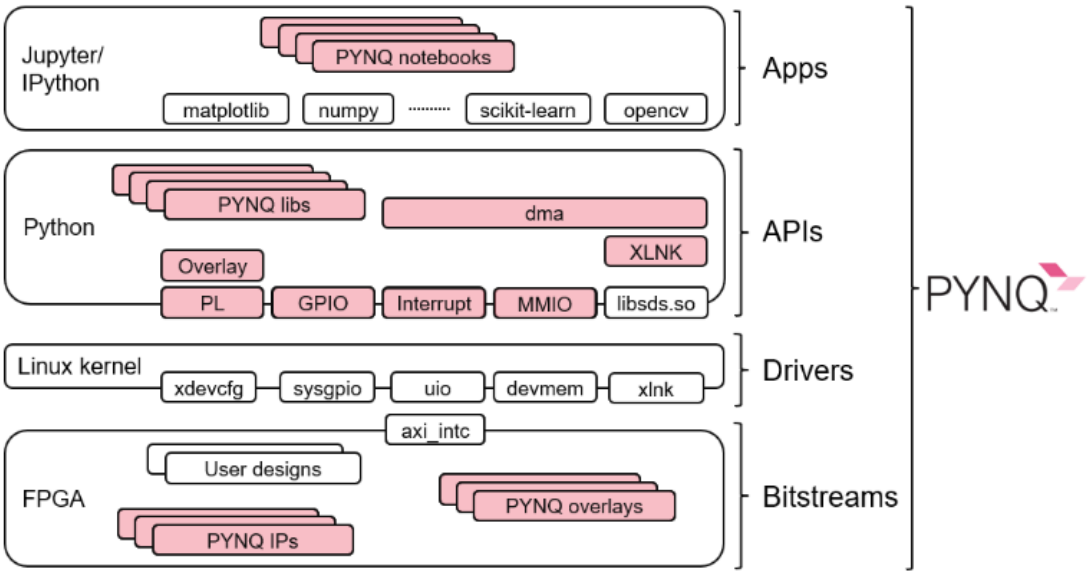

图 2.2.17　嵌入式 Linux 软件层次图

本项目的 PS 部分软件基于 PYNQ 框架进行开发，其嵌入式 Linux 软件

层次图如图所示，包括硬件层、驱动层和应用层。其中硬件层通过 PS 与 PL 部分的交互，生成比特流文件，调用 FPGA 上的逻辑资源；驱动层主要包括 Linux 内核以及 Linux 操作系统中的 Python，通过 API 库进行软硬件的连接；应用层则主要由载有 Jupyter Notebooks 设计环境的网络服务器、IPython 内核和程序包组成。通过 Petalinux 定制移植 PYNQ 框架，将底层硬件 FPGA 实现细节与上层应用层的使用脱耦，提供访问 FPGA 资源的 python 接口，使 FPGA 可并行计算、接口可方便扩展和可灵活配置，能够动态加载预编译好的 FPGA 算法应用。

项目将 PYNQ 框架移植到 7010 核心板，并编写脉动阵列加速器驱动，让设备通过 Tensorflow-Lite 调度加速器完成卷积计算。已经训练好并完成量化的神经网络算法直接通过 Tensorflow-Lite 框架进行计算。项目还移植了 Xilinx OpenCV 库，从而让图像能够直接推流到上位机。

第三部分　完成情况及性能参数

机器人主体框架如下图所示：

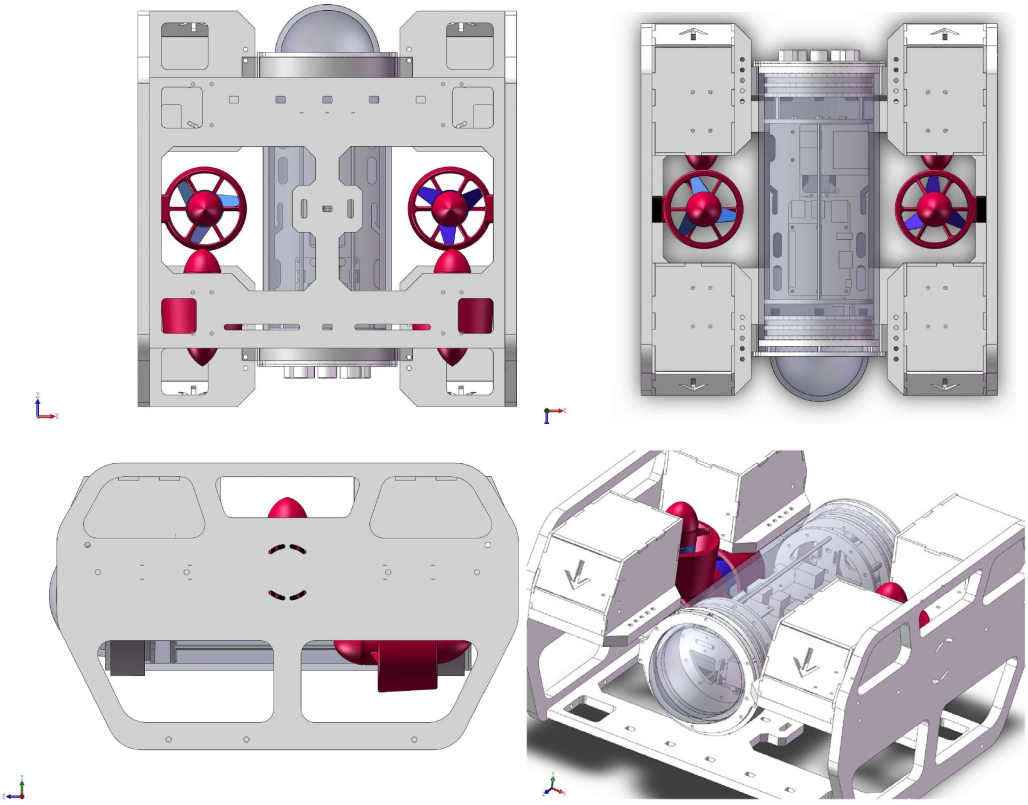

图 3.1　机器人主体框架

机器人实物图如下所示：

图 3.2　机器人实物图

目前完成了项目主体框架适配和板级硬件的设计制作，PCB 实物如下图所示 ：

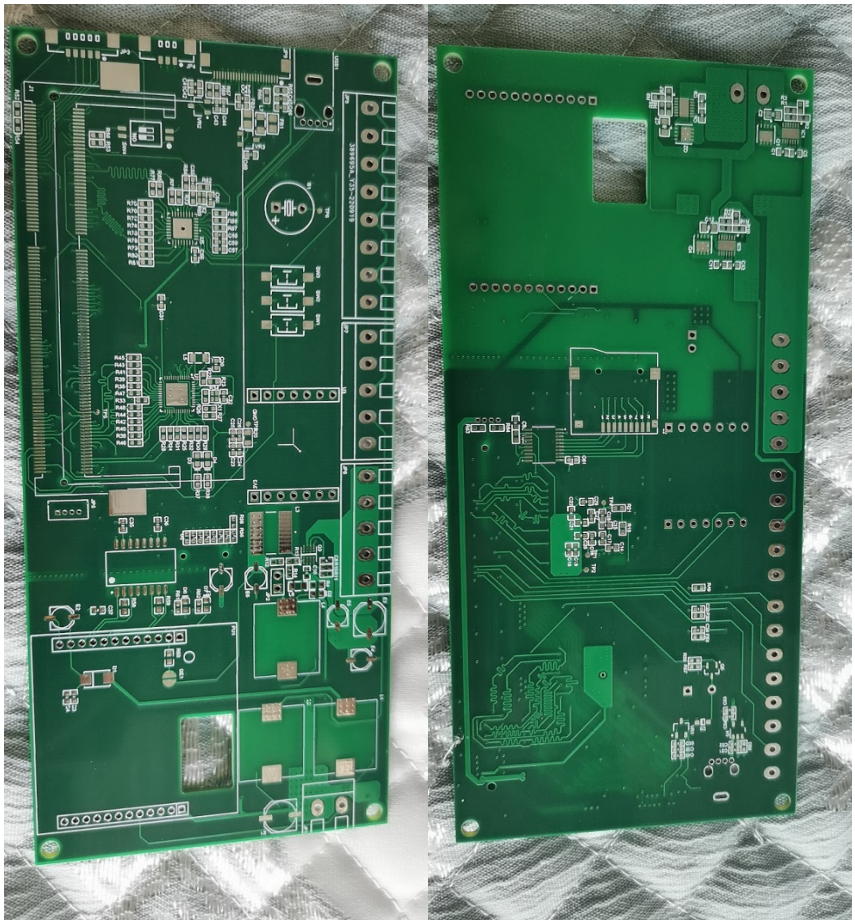

图 3.3　已完成 PCB 实物图

片上硬件部分，初版 SoC 使用开源 RISC-V 内核搭建，五级流水带分支预测的 RV32IM 指令集。目前已经完成了 GPIO、PWM、UART、I2C、DVP 控制器的编写，其余外设正在编写或验证中。

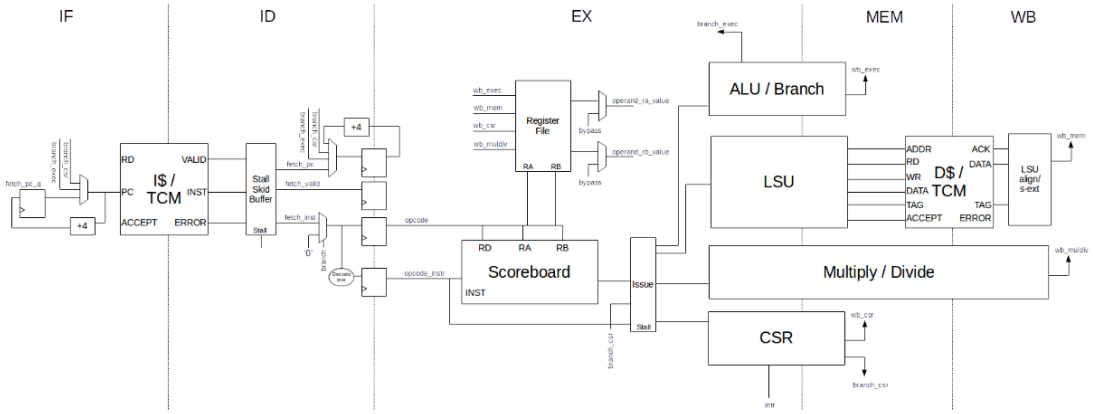

图 3.4　已完成板上硬件图

部分 RTL 文件如下所示：

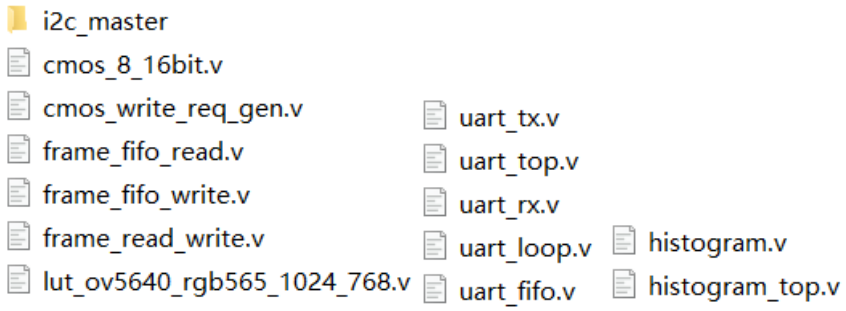

图 3.5　目前 RTL 文件图

脉动阵列加速器已经完成编写和测试工作，综合后资源使用量如下所示：

Name	Slice LUTs (53200)	Slice Registers (106400)
conv2d_channel	1259	1254
u_conv2d_feature (conv2d_feature)	1259	1254

图 3.6　脉动阵列加速器 LUT 使用量

第四部分　总结

4.1 可扩展之处

目前项目相对完善，如果有空余时间，会考虑将项目部署到一台 45kg 级别的中型水下作业机器人，并支持多自由度机械臂的实时控制。

4.2 心得体会

本项目复杂度较高，在努力完成的过程中，团队成员的综合工程能力都得到了锻炼。

第五部分　参考文献

[1] 贾献强. 基于 Zynq 平台的深海作业型 ROV 控制系统设计[D].哈尔滨工程大学,2017.

[2] 张泽鑫. 基于FPGA的水下ROV多业务高清视频光端机设计[D].杭州电子科技大学,2020.

[3] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)

[4] Jouppi, N. et al, In-datacenter performance of a tensor processing unit. In Proceedings of the 44th International Symposium on Computer Architecture (Toronto, Canada, June 24--28). ACM Press, New York, 2017, 1--12

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in IEEE CVPR, 2016

[6] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. arXiv preprint arXiv:2207.02696, 2022

第六部分　附录

脉动阵列加速器部分关键代码如下图所示：

```
22  multip_adder #(
23                  .BITWIDTH ( BITWIDTH ),
24                  .IS_BITWIDTH_DOUBLE_SCALE(IS_BITWIDTH_DOUBLE_SCALE)
25                  )
26              u_multip_adder(
27                  //ports
28                  .in_a ( in_a ),
29                  .in_b ( in_b ),
30                  .in_c ( product_acc ),
31                  .product ( product )
32              );
33
34  //delay
35  always @(posedge clk or negedge rst_n) begin
36      if (!rst_n) begin
37          product_acc <= {(BITWIDTH * (IS_BITWIDTH_DOUBLE_SCALE + 1)){1'b0}};
38          out_a_delay <= {BITWIDTH{1'b0}};
39          out_b_delay <= {BITWIDTH{1'b0}};
40      end
41      else begin
42          if (en) begin
43              product_acc <= product;
44              out_a_delay <= in_a;
45              out_b_delay <= in_b;
46          end
47          else begin
48              product_acc <= product_acc;
49              out_a_delay <= out_a_delay;
50              out_b_delay <= out_b_delay;
51          end
52      end
53  end
```

图 6.1　pe_bit.v

```
//trans array into matrix
//regenerate as 2-dimension array
//wait for sending to SA
genvar r, s;
generate
    for (s = 0;s < X_ROW;s = s + 1) begin
        assign in_row[(X_ROW * BITWIDTH - 1) - (s * BITWIDTH) -: BITWIDTH] = data_buffer_row[0][s];
    end

    for (r = 0;r < XCOL_YROW + X_ROW - 1;r = r + 1) begin
        for (s = 0;s < X_ROW;s = s + 1) begin
            always @(posedge sys_clk or negedge sys_rst_n) begin
                if (!sys_rst_n) begin
                    data_buffer_row[r][s] <= {BITWIDTH{1'b0}};
                end
                else if (calculate_flag) begin
                    if (r < XCOL_YROW + X_ROW - 2) begin
                        data_buffer_row[r][s] <= data_buffer_row[r + 1][s];
                    end
                    else begin
                        data_buffer_row[r][s] <= {BITWIDTH{1'b0}};
                    end
                end
                else begin
                    if ((s <= r) && (s >= r - (XCOL_YROW - 1))) begin
                        data_buffer_row[r][s] <= X[((BITWIDTH * XCOL_YROW * X_ROW - 1) - ((s * BITWIDTH * X
                    end
                    else begin
                        data_buffer_row[r][s] <= {BITWIDTH{1'b0}};
                    end
                end
            end
        end
    end
endgenerate
```

图 6.2　array_regenerate.v

```
genvar i;
generate
    for (i = 0;i < X_ROW;i = i + 1) begin
        if (i == 0) begin //first column receive matrix X input
            pe_row #(
                        .BITWIDTH ( BITWIDTH ),
                        .IS_BITWIDTH_DOUBLE_SCALE(IS_BITWIDTH_DOUBLE_SCALE),
                        .Y_COL ( Y_COL ))
                    u_pe_row(
                        //ports
                        .clk ( clk ),
                        .rst_n ( rst_n ),
                        .en ( en ),
                        .in_row ( in_row[(X_ROW * BITWIDTH - 1) -: BITWIDTH] ),
                        .in_col ( in_col ),
                        .out_col ( PulsedArray_COL[i] ),
                        .row_result ( result[((X_ROW * Y_COL * BITWIDTH * (IS_BITWIDTH_DOUBLE_SCALE + 1)) - 1) -: (Y_COL * BITWIDTH * (IS
                    );
        end
        else begin //interconnection
            pe_row #(
                        .BITWIDTH ( BITWIDTH ),
                        .IS_BITWIDTH_DOUBLE_SCALE(IS_BITWIDTH_DOUBLE_SCALE),
                        .Y_COL ( Y_COL ))
                    u_pe_row(
                        //ports
                        .clk ( clk ),
                        .rst_n ( rst_n ),
                        .en ( en ),
                        .in_row ( in_row[((X_ROW * BITWIDTH - 1) - (i * BITWIDTH)) -: BITWIDTH] ),
                        .in_col ( PulsedArray_COL[i - 1] ),
                        .out_col ( PulsedArray_COL[i] ),
                        .row_result ( result[(((X_ROW * Y_COL * BITWIDTH * (IS_BITWIDTH_DOUBLE_SCALE + 1)) - 1) - (i * Y_COL * BITWIDTH *
                    );
        end
    end
endgenerate
```

图 6.3　pe_array.v

```
107 //FSM
108 always @(posedge sys_clk or negedge sys_rst_n) begin
109     if (!sys_rst_n) begin
110         current_state <= IDLE;
111     end
112     else begin
113         current_state <= next_state;
114     end
115 end
116 
117 always @( * ) begin
118     case (current_state)
119         IDLE: begin
120             if (start_pos_sig) begin //start calculate when recv posedge of start
121                 next_state = BUSY;
122             end
123             else begin
124                 next_state = IDLE;
125             end
126         end
127         BUSY: begin
128             if (cal_cnt > XCOL_YROW + X_ROW - 1) begin //one calculation done when counter reach the end
129                 next_state = IDLE;
130             end
131             else begin
132                 next_state = BUSY;
133             end
134         end
135         default: begin
136             next_state = IDLE;
137         end
138     endcase
139 end
```

图 6.4　systolic_array.v

```
3   module relu #(
4               parameter BITWIDTH = 8,
5               parameter THRESSHOLD = {BITWIDTH{1'b0}},
6               parameter IS_RELU = 1,
7               parameter MAX_VAL = 6
8           )(
9               input wire signed [BITWIDTH - 1 : 0] data_i, //signed data
10              output reg signed [BITWIDTH - 1 : 0] result_o //signed data
11          );
12  if (IS_RELU) begin
13      always @( * ) begin
14          if (data_i > $signed(THRESSHOLD)) begin
15              result_o = data_i;
16          end
17          else begin
18              result_o = THRESSHOLD;
19          end
20      end
21  end
22  else begin
23      always @( * ) begin
24          if (data_i > $signed(THRESSHOLD)) begin
25              if (data_i < MAX_VAL) begin
26                  result_o = data_i;
27              end
28              else begin
29                  result_o = MAX_VAL;
30              end
31          end
32          else begin
33              result_o = THRESSHOLD;
34          end
35      end
36  end
37  endmodule
```

图 6.5　relu.v

```
23  //re-arrange image to array, add padding rows&cols
24  genvar m, n;
25  generate
26      if (PADDING) begin
27          for (m = 0;m < IMAGE_WIDTH + 2 * PADDING;m = m + 1) begin
28              for (n = 0;n < IMAGE_HEIGHT + 2 * PADDING;n = n + 1) begin
29                  if (m < PADDING) begin //first row
30                      assign image_padding[m][n] = 8'b0;
31                  end
32                  else if (m >= IMAGE_WIDTH + 2 * PADDING - PADDING) begin //last row
33                      assign image_padding[m][n] = 8'b0;
34                  end
35                  else if (n < PADDING) begin //first column
36                      assign image_padding[m][n] = 8'b0;
37                  end
38                  else if (n >= IMAGE_HEIGHT + 2 * PADDING - PADDING) begin //last column
39                      assign image_padding[m][n] = 8'b0;
40                  end
41                  else begin
42                      assign image_padding[m][n] = image[(IMAGE_WIDTH * IMAGE_HEIGHT * BITWIDTH - 1) - ((m - PADDING) * IM
43                  end
44              end
45          end
46      end
47      else begin
48          for (m = 0;m < IMAGE_WIDTH;m = m + 1) begin
49              for (n = 0;n < IMAGE_HEIGHT;n = n + 1) begin
50                  assign image_padding[m][n] = image[(IMAGE_WIDTH * IMAGE_HEIGHT * BITWIDTH - 1) - (m * IMAGE_HEIGHT * BIT
51              end
52          end
53      end
54  endgenerate
```

图 6.6　img2col.v

```
conv2d_img2col #(
                    .BITWIDTH ( BITWIDTH ),
                    .IMAGE_WIDTH ( IMAGE_WIDTH ),
                    .IMAGE_HEIGHT ( IMAGE_HEIGHT ),
                    .WEIGHT_WIDTH ( FEATURE_MAP_WIDTH ),
                    .WEIGHT_HEIGHT ( FEATURE_MAP_HEIGHT ),
                    .FEATURE_MAP_NUM ( FEATURE_MAP_NUM ),
                    .KERNEL_NUM ( KERNEL_NUM ),
                    .PADDING ( PADDING ),
                    .STRIDE ( STRIDE ))
                u_conv2d_img2col(
                    .image ( image ),
                    .weights ( weights ),
                    .feature_maps_v ( feature_maps_v ),
                    .weights_v ( weights_v )
                );

conv2d_feature #(
                    .BITWIDTH ( BITWIDTH ),
                    .IS_BITWIDTH_DOUBLE_SCALE ( IS_BITWIDTH_DOUBLE_SCALE ),
                    .FEATURE_MAP_NUM ( FEATURE_MAP_NUM ),
                    .FEATURE_MAP_WIDTH ( FEATURE_MAP_WIDTH ),
                    .FEATURE_MAP_HEIGHT ( FEATURE_MAP_HEIGHT ),
                    .KERNEL_NUM ( KERNEL_NUM ))
                u_conv2d_feature(
                    .clk ( clk ),
                    .rst_n ( rst_n ),
                    .calculate_start ( calculate_start ),
                    .calculate_done ( calculate_done ),
                    .feature_maps ( feature_maps_v ),
                    .weights ( weights_v ),
                    .channel_out ( channel_out_v )
                );
```

图 6.7　conv2d_channel.v

部分 SoC 外设关键代码如下所示：

```
//hsync negedge: frame start
//init RAM = 0
reg ram_clr_en;
always @(posedge sys_clk_i or negedge sys_rst_n_i) begin
    if (!sys_rst_n_i) begin
        ram_clr_en <= 1'b0;
    end
    else begin
        if (vsync_negedge) begin
            ram_clr_en <= 1'b1;
        end
        else if (ram_clr_addr == HSV_CNT - 1) begin
            ram_clr_en <= 1'b0;
        end
        else begin
            ram_clr_en <= ram_clr_en;
        end
    end
end

reg [HSV_LEVEL - 1: 0] ram_clr_addr;
always @(posedge sys_clk_i or negedge sys_rst_n_i) begin
    if (!sys_rst_n_i) begin
        ram_clr_addr <= 0;
    end
    else if (ram_clr_en) begin
        ram_clr_addr <= (ram_clr_addr >= HSV_CNT - 1) ? 0 : ram_clr_addr + 1;
    end
    else begin
        ram_clr_addr <= 0;
    end
end
```

图 6.8　histogram.v

```
// state machine
reg [4:0] c_state; // synopsys enum_state

always @(posedge clk or negedge nReset)
if(!nReset)
begin
    core_cmd <= #1 `I2C_CMD_NOP;
    core_txd <= #1 1'b0;
    shift    <= #1 1'b0;
    ld       <= #1 1'b0;
    cmd_ack  <= #1 1'b0;
    c_state  <= #1 ST_IDLE;
    ack_out  <= #1 1'b0;
end
else if (rst | i2c_al)
begin
        core_cmd <= #1 `I2C_CMD_NOP;
        core_txd <= #1 1'b0;
        shift    <= #1 1'b0;
        ld       <= #1 1'b0;
        cmd_ack  <= #1 1'b0;
        c_state  <= #1 ST_IDLE;
        ack_out  <= #1 1'b0;
end
else
begin
      // initially reset all signals
      core_txd <= #1 sr[7];
      shift    <= #1 1'b0;
      ld       <= #1 1'b0;
      cmd_ack  <= #1 1'b0;
```

图 6.9　i2c_master_byte_ctrl.v

```
29  always@(posedge pclk or posedge rst)
30  begin
31      if(rst)
32          de_o <= 1'b0;
33      else if(de_i && x_cnt[0])
34          de_o <= 1'b1;
35      else
36          de_o <= 1'b0;
37  end
38
39  always@(posedge pclk or posedge rst)
40  begin
41      if(rst)
42          hblank <= 1'b0;
43      else
44          hblank <= de_i;
45  end
46
47  always@(posedge pclk or posedge rst)
48  begin
49      if(rst)
50          pdata_o <= 16'd0;
51      else if(de_i && x_cnt[0])
52          pdata_o <= {pdata_i_d0,pdata_i};
53      else
54          pdata_o <= 16'd0;
55  end
```

图 6.10　cmos_8_16bit.v

```
29  //reveive latency: 2 clk-edge
30  always @(posedge sys_clk_i or negedge sys_rst_n_i) begin
31      if (!sys_rst_n_i) begin
32          uart_rxd_d0 <= 1'b0;
33          uart_rxd_d1 <= 1'b0;
34      end
35      else begin
36          uart_rxd_d0 <= uart_rxd_i;
37          uart_rxd_d1 <= uart_rxd_d0;
38      end
39  end
40
41  //when get start bit, start receive data
42  always @(posedge sys_clk_i or negedge sys_rst_n_i) begin
43      if (!sys_rst_n_i) begin
44          rx_flag <= 1'b0;
45      end
46      else begin
47          if (start_flag)
48              rx_flag <= 1'b1; //start to reveive
49          else if ((rx_cnt == 4'd9) && (clk_cnt == BPS_CNT / 2))
50              rx_flag <= 1'b0; //receive done
51          else
52              rx_flag <= rx_flag;
53      end
54  end
55
```

图 6.11　uart_rx.v