Kria KV260 YOLO V7 + RISC-V 程式碼架構分析

1. 程式碼分類概覽

1.1 按開發層級分類

1.2 按功能模組分類

功能類別	模組名稱	檔案	主要功能
🏗️ **專案建構**	Project Builder	`create_project.tcl`	Vivado 專案自動化建構
🚀 **AI 加速**	E-ELAN Accelerator	`e-elan.v`	YOLO V7 核心硬體加速
🚀 **AI 加速**	Upsample Accelerator	`e-elan.v`	特徵圖上採樣硬體加速
🚀 **AI 加速**	Concat Optimizer	`e-elan.v`	記憶體拼接最佳化
🚀 **AI 加速**	RepConv Converter	`e-elan.v`	推理模式轉換器
⚡ **處理器核心**	RISC-V Wrapper	`rocket_axi_wrapper.v`	RISC-V 核心 AXI 包裝
⚡ **處理器核心**	YOLO Controller	`rocket_axi_wrapper.v`	YOLO 加速器控制
💻 **主控軟體**	ARM A53 Controller	`main_controller.c`	Linux 主控制程式
💻 **後處理**	RISC-V Processor	`main_controller.c`	專用後處理程式

2. 詳細程式碼架構

2.1 硬體描述層 (HDL) 架構

🎯 專案建構腳本 (`create_project.tcl`)

```tcl
專案設置與 IP 整合
├── 創建 KV260 專案
├── 配置 Zynq UltraScale+ MPSoC
│   ├── PS 端配置 (ARM 處理器)
│   ├── PL 端配置 (FPGA 邏輯)
│   └── 介面啟用 (UART、USB、SD、Ethernet)
├── 整合 IP 核心
│   ├── DPU (Deep Learning Processing Unit)
│   ├── RISC-V Rocket Core
│   ├── Video Pipeline (MIPI CSI-2, V_PROC_SS)
│   ├── AXI Interconnect
│   └── 時脈管理 (Clock Wizard)
├── 系統連接
│   ├── 時脈域連接
│   ├── AXI 匯流排連接
│   └── 中斷信號連接
└── 建構輸出
    ├── HDL Wrapper 生成
    ├── 約束檔案整合
    └── Bitstream 生成
```

🚀 YOLO V7 專用硬體加速器 (`e-elan.v`)

A. E-ELAN 加速器

```verilog
elan_accelerator 模組架構
├── 📥 AXI4-Stream 輸入介面
├── 🔄 分組卷積單元 (4 組並行)
│   ├── group_conv_3x3 #0
│   ├── group_conv_3x3 #1
│   ├── group_conv_3x3 #2
│   └── group_conv_3x3 #3
├── 🔀 特徵重組 (Shuffle) 單元
│   ├── 分組大小配置 (2/4/8)
│   ├── 基數 (Cardinality) 控制
│   └── 動態重組邏輯
├── 🎯 合併 (Merge) 單元
│   ├── 基數合併邏輯
│   ├── 輸出緩衝管理
│   └── 流量控制
└── 📤 AXI4-Stream 輸出介面
```

B. Upsample 加速器

```
verilog

upsample_accelerator 模組架構
├── 📥 像素輸入介面
├── 📊 狀態機控制
│   ├── IDLE → WAIT_INPUT
│   ├── WAIT_INPUT → COMPUTE_WEIGHTS
│   ├── COMPUTE_WEIGHTS → INTERPOLATE
│   ├── INTERPOLATE → OUTPUT_PIXEL
│   └── OUTPUT_PIXEL → (循環)
├── 🧮 插值計算單元
│   ├── 雙線性插值 (4 點加權)
│   ├── 最近鄰插值
│   ├── 權重係數計算
│   └── 定點數運算
├── 💾 線緩衝區管理
│   ├── 像素緩衝區 (4 個像素)
│   ├── 線緩衝區 (最大寬度)
│   └── 地址管理
└── 📤 上採樣輸出介面
```

C. Concat 記憶體最佳化器

```
verilog

concat_memory_optimizer 模組架構
├── 📥 多路輸入介面 (可配置數量)
├── 🔄 同步控制邏輯
│   ├── 全輸入就緒檢測
│   ├── SOF/EOL 同步
│   └── 流量控制協調
├── 🚚 DMA 零拷貝引擎
│   ├── 多通道 DMA 控制
│   ├── 記憶體地址管理
│   ├── 傳輸長度計算
│   └── 完成狀態監控
├── 🔗 直接拼接邏輯
│   ├── 即時數據拼接
│   ├── 位元組對齊處理
│   └── 輸出格式化
└── 📤 統一輸出介面
```

D. RepConv 推理轉換器

verilog

```
repconv_inference_converter 模組架構
├── 📥 特徵圖輸入
├── 💾 多權重存儲系統
│   ├── 3x3 卷積權重
│   ├── 1x1 卷積權重
│   ├── 恆等映射權重
│   └── 合併權重緩存
├── 🔧 權重合併引擎
│   ├── 訓練模式權重載入
│   ├── 推理模式權重合併
│   ├── 自動轉換邏輯
│   └── 完成狀態管理
├── ⚙️ 標準卷積單元
│   ├── 3x3 卷積計算
│   ├── 多通道處理
│   └── 流水線最佳化
└── 📤 推理結果輸出
```

⚡ RISC-V 核心包裝器 (`rocket_axi_wrapper.v`)

A. Rocket Core AXI 包裝器

verilog

```
rocket_axi_wrapper 模組架構
├── 🎛️ AXI4-Lite Slave 控制介面
│   ├── 寫入通道 (AW/W/B)
│   ├── 讀取通道 (AR/R)
│   └── GPIO 控制
├── 🚀 Rocket Core 實例
│   ├── 時脈與重置管理
│   ├── TileLink 介面
│   ├── MMIO 連接
│   └── 中斷處理
├── 🔄 TileLink 到 AXI4 轉換器
│   ├── 協定轉換邏輯
│   ├── 地址映射
│   ├── 數據寬度轉換
│   └── 流量控制
├── 🖥️ AXI4 Master 記憶體介面
│   ├── 讀寫通道完整實現
│   ├── Burst 傳輸支援
│   ├── Cache 一致性
│   └── QoS 控制
└── ⚡ 中斷與 GPIO 介面
```

B. YOLO 加速器控制器

verilog

```
yolo_accelerator_ctrl 模組架構
├── 🎛️ AXI4-Lite 暫存器介面
│   ├── REG_CONTROL (0x000) - 控制暫存器
│   ├── REG_STATUS (0x004) - 狀態暫存器
│   ├── REG_INPUT_ADDR (0x008) - 輸入地址
│   ├── REG_OUTPUT_ADDR (0x00C) - 輸出地址
│   ├── REG_INPUT_WIDTH (0x010) - 輸入寬度
│   ├── REG_INPUT_HEIGHT (0x014) - 輸入高度
│   └── REG_INTERRUPT (0x018) - 中斷控制
├── 🔄 DPU 控制邏輯
│   ├── 啟動信號生成
│   ├── 狀態監控
│   ├── 參數配置
│   └── 完成檢測
├── ⚡ 中斷管理
│   ├── DPU 完成中斷
│   ├── 中斷啟用控制
│   ├── 中斷清除邏輯
│   └── 中斷狀態輸出
└── 📊 狀態暫存器更新
```

2.2 軟體應用層架構

💻 ARM A53 主控程式 (`main_controller.c`)

c

```
主控程式架構
├── 🔧 硬體初始化模組
│   ├── /dev/mem 記憶體映射
│   ├── 硬體暫存器映射
│   ├── 共享記憶體設置
│   └── 錯誤處理機制
├── 🤝 RISC-V 通訊介面
│   ├── 命令發送函數
│   ├── 狀態檢查邏輯
│   ├── 參數傳遞機制
│   └── 中斷觸發控制
├── 🎯 DPU 控制介面
│   ├── 推理啟動函數
│   ├── 參數配置邏輯
│   ├── 完成等待機制
│   └── 超時保護處理
├── 🎬 影片處理管線
│   ├── 幀資料預處理
│   ├── DPU 推理呼叫
│   ├── RISC-V 後處理
│   ├── 效能統計計算
│   └── 結果整合輸出
├── 🚦 多執行緒架構
│   ├── 視訊處理執行緒
│   ├── 統計監控執行緒
│   ├── 執行緒同步機制
│   └── 資源管理邏輯
└── 📊 系統監控模組
    ├── 效能統計收集
    ├── FPS 計算
    ├── 延遲監控
    └── 週期性報告
```

⚙️ RISC-V 後處理程式 (`main_controller.c`)

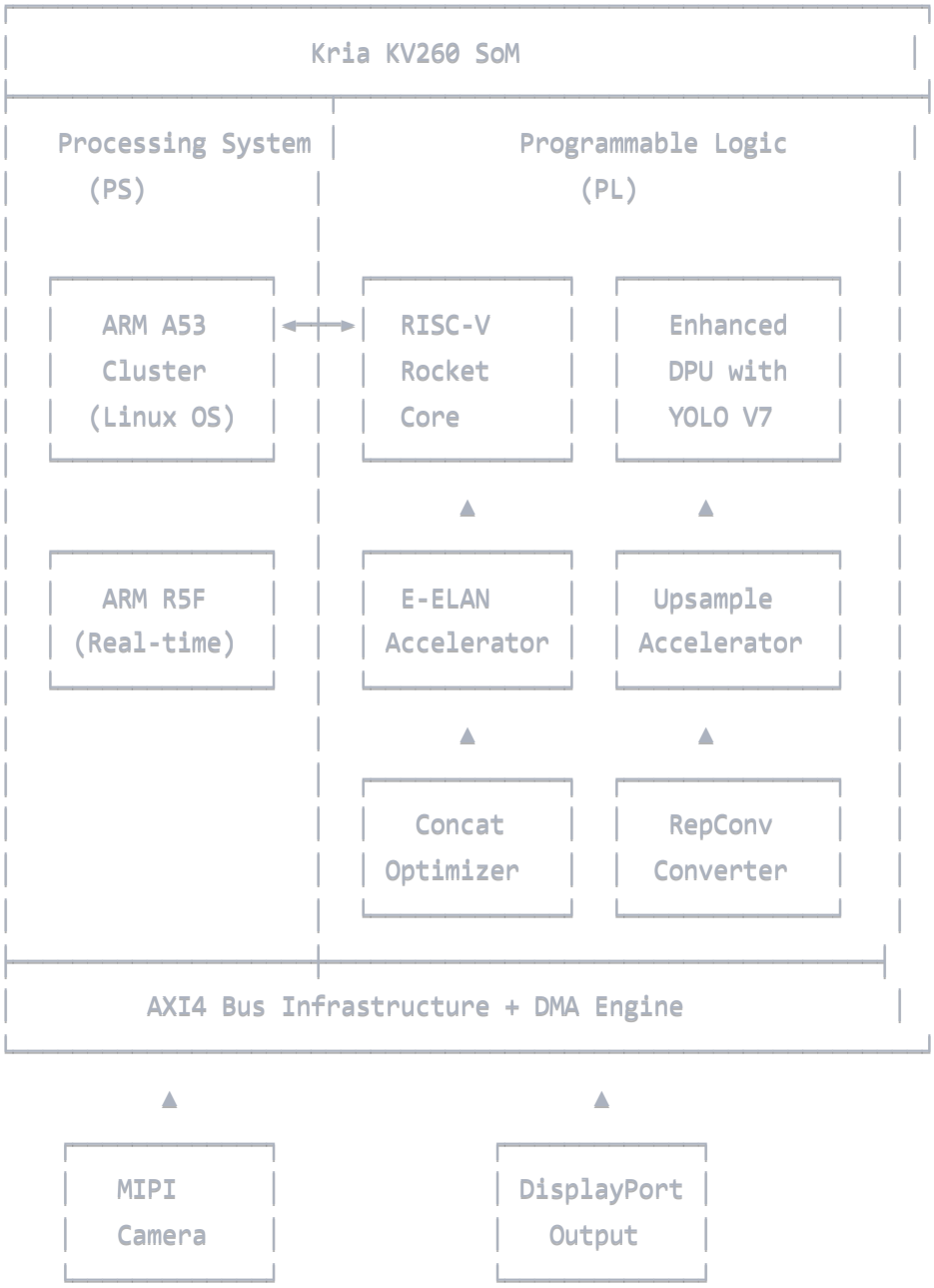
c

```
RISC-V 後處理程式架構
├── 🎛️ 硬體暫存器介面
│   ├── 命令暫存器讀取
│   ├── 狀態暫存器更新
│   ├── 數據地址管理
│   └── 中斷控制邏輯
├── 🎯 YOLO 結果解析
│   ├── 原始輸出解析 (25200 檢測)
│   ├── 信心度過濾 (>0.25)
│   ├── 類別機率計算
│   ├── 邊界框座標轉換
│   └── 檢測結構體填充
├── 🖌️ NMS 非最大值抑制
│   ├── IoU 計算函數
│   ├── 重疊檢測濾除
│   ├── 信心度排序
│   ├── 保留檢測標記
│   └── 最終結果統計
├── 💾 共享記憶體管理
│   ├── DPU 輸出讀取
│   ├── 處理結果寫入
│   ├── 記憶體地址計算
│   └── 數據格式轉換
├── ⚡ 中斷處理機制
│   ├── 外部中斷處理
│   ├── 命令分派邏輯
│   ├── 處理完成回報
│   └── 狀態更新機制
└── 🔄 主迴圈控制
    ├── 中斷等待 (WFI)
    ├── 全域中斷啟用
    ├── 外部中斷啟用
    └── 系統初始化
```

3. 系統整合架構

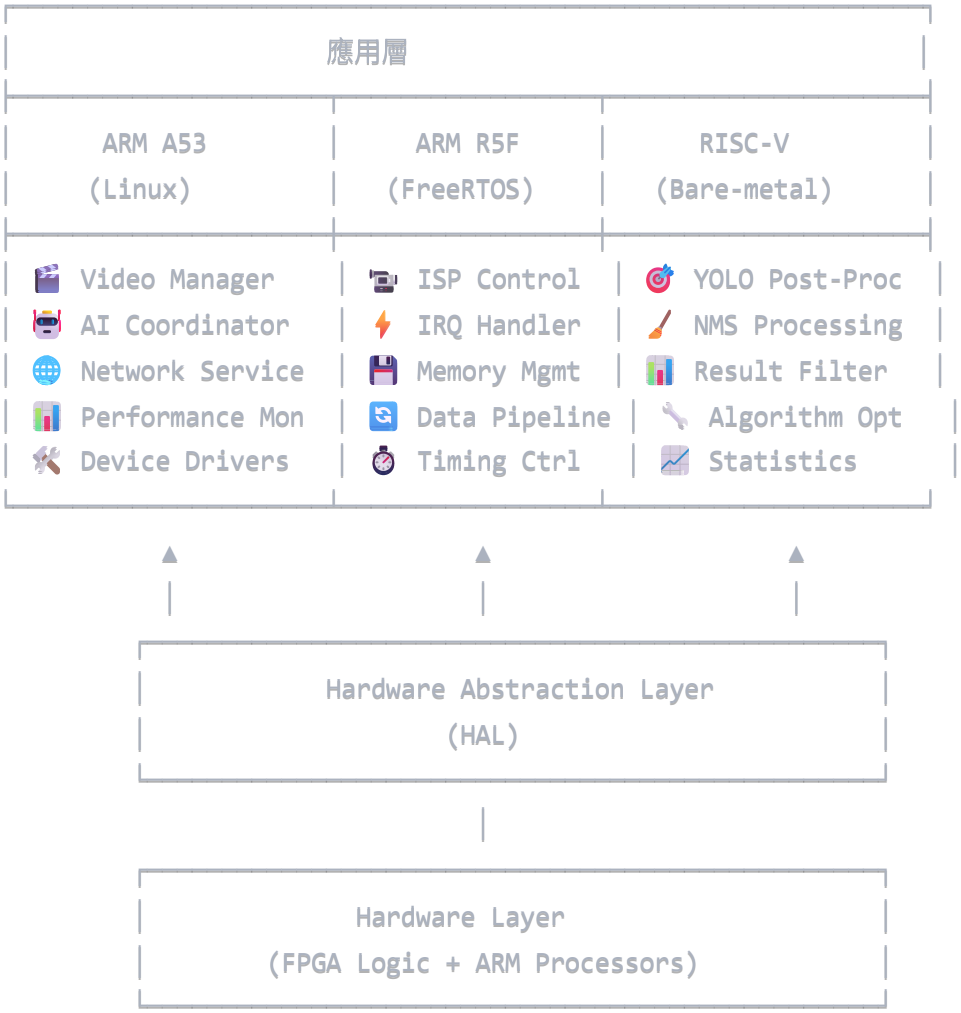
3.1 硬體層級整合

硬體系統架構

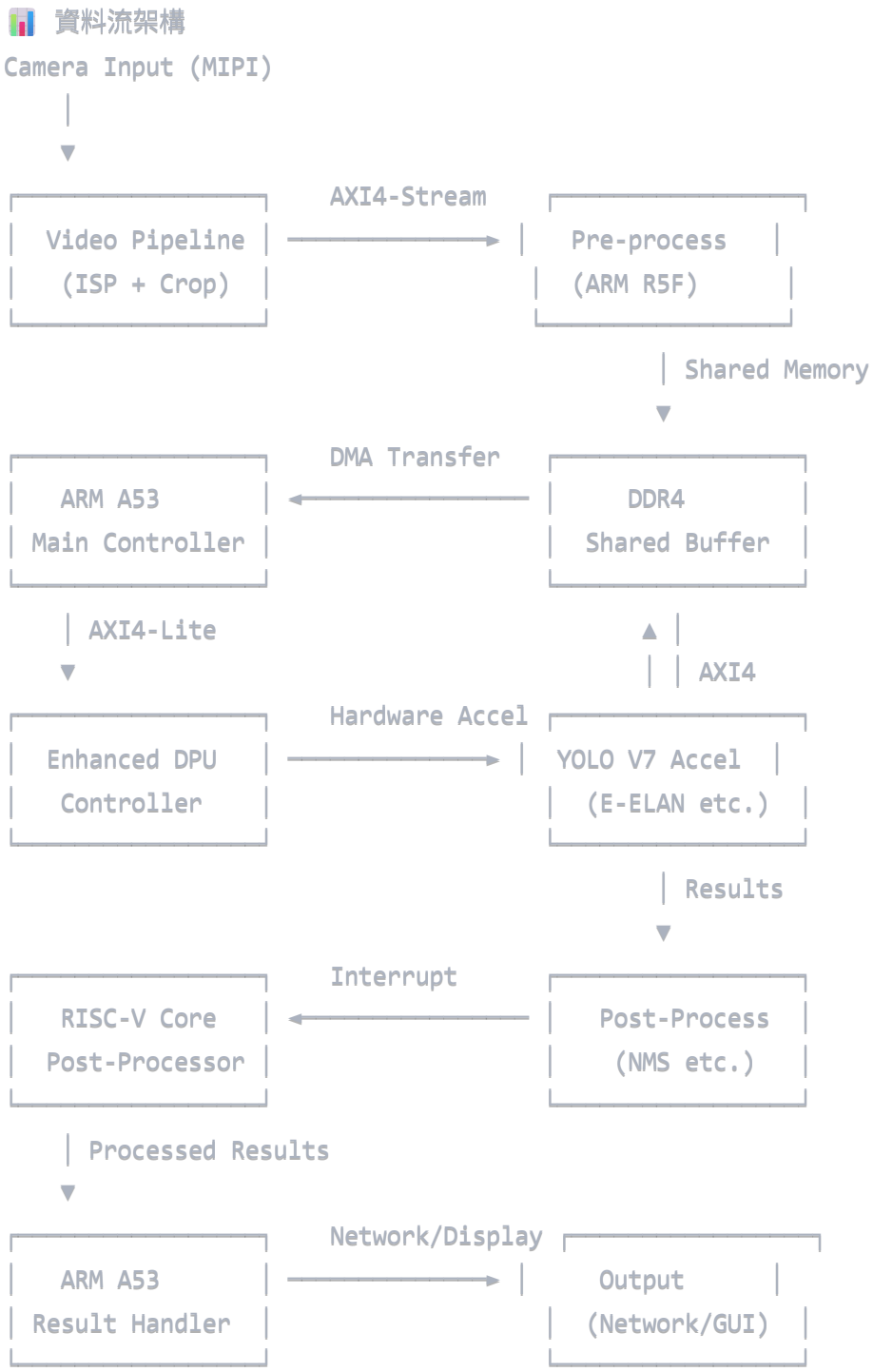


3.2 軟體層級整合

💻 軟體系統架構



3.3 資料流架構



4. 關鍵設計特色

4.1 🚀 硬體加速創新

- **E-ELAN 專用加速器**：首創針對 YOLO V7 核心算法的硬體實現
- **管線化處理**：多級流水線確保高吞吐量
- **記憶體最佳化**：零拷貝 DMA 減少記憶體頻寬瓶頸
- **動態重配置**：支援不同模型尺寸的動態切換

4.2 ⚡ 異構運算協調

- **三核心協作**：ARM A53、ARM R5F、RISC-V 各司其職

- **中斷驅動**：低延遲事件響應機制
- **共享記憶體**：高效數據交換協定
- **負載平衡**：智慧工作負載分配

4.3 🎯 系統最佳化

- **端對端管線**：從影像輸入到結果輸出的完整最佳化
- **即時監控**：完整的效能監控和統計系統
- **模組化設計**：高度模組化便於維護和擴展
- **錯誤處理**：完善的錯誤處理和恢復機制

4.4 🔧 開發友善性

- **自動化建構**：一鍵式 Vivado 專案建構
- **標準介面**：符合 AXI4 標準的統一介面
- **可擴展架構**：支援未來的功能擴展
- **豐富調試**：完整的調試和效能分析工具

這個架構設計實現了 YOLO V7 在 Kria KV260 上的高效能部署，充分發揮了異構運算的優勢，為邊緣 AI 視覺處理提供了完整的解決方案。