Kria KV260 上的 YOLO V7 + RISC-V 整合架構設計

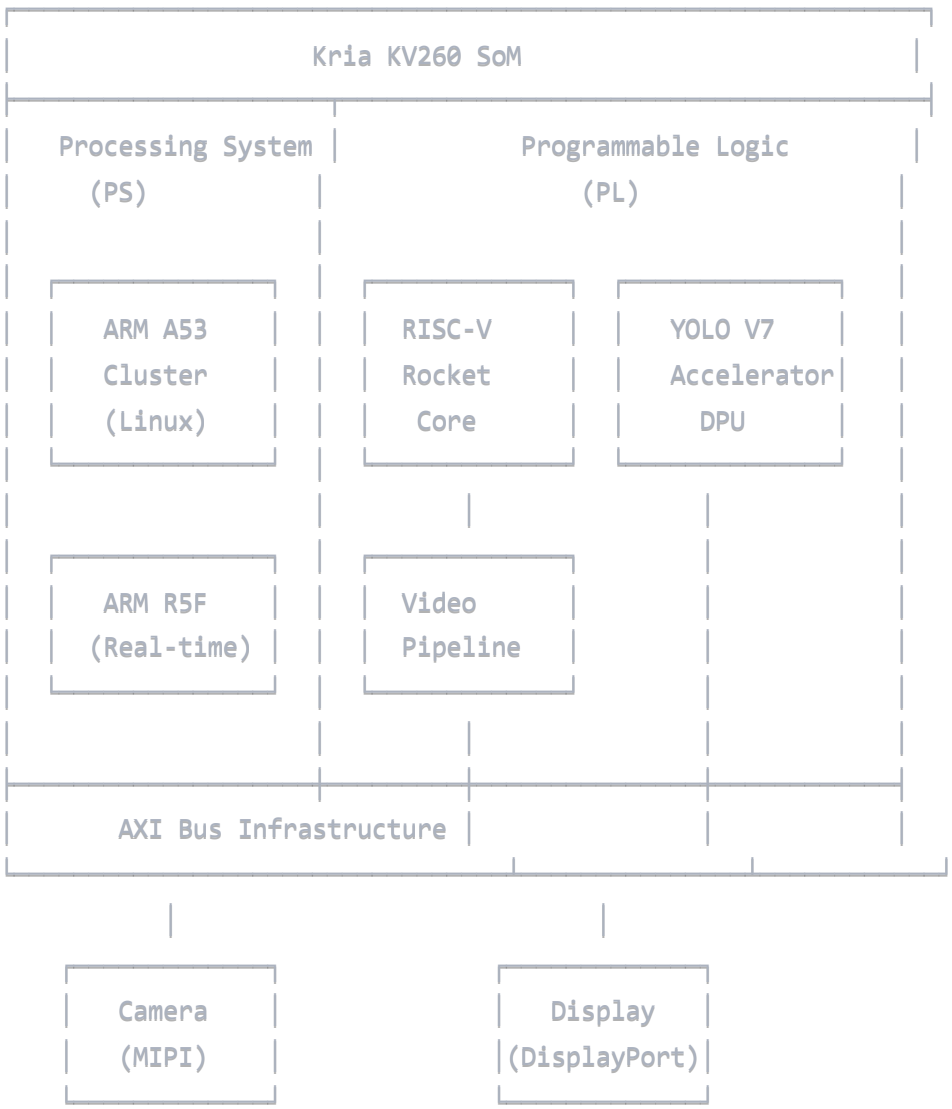
1. 系統概述

本設計將在 Xilinx Kria KV260 Vision AI Starter Kit 上實現一個整合了 YOLO V7 物件檢測和 RISC-V 處理器的異構運算系統。

1.1 硬體平台特性

- **主處理器**: Zynq UltraScale+ MPSoC (Quad-core ARM Cortex-A53 + Dual-core ARM Cortex-R5F)
- **FPGA 邏輯**: Kintex UltraScale FPGA 邏輯
- **AI 加速**: 內建 DPU (Deep Learning Processing Unit)
- **記憶體**: 4GB LPDDR4, 256Mb QSPI Flash, microSD
- **介面**: 4x USB 3.0, Gigabit Ethernet, MIPI CSI-2, DisplayPort

2. 整體系統架構



3. 詳細模組設計

3.1 RISC-V 核心整合

3.1.1 從 ECP5 移植到 Kria KV260

```verilog
// 主要修改點
module rocket_wrapper_kv260(
    // AXI4 介面 (連接 PS)
    input         axi_aclk,
    input         axi_aresetn,

    // AXI4-Lite Slave (配置介面)
    input  [31:0] s_axi_awaddr,
    input         s_axi_awvalid,
    output        s_axi_awready,
    // ... 其他 AXI 信號

    // AXI4 Master (記憶體存取)
    output [31:0] m_axi_araddr,
    output        m_axi_arvalid,
    input         m_axi_arready,
    // ... 其他 AXI 信號

    // 中斷信號
    output        risc_v_interrupt,

    // 自定義介面
    input  [7:0]  gpio_in,
    output [7:0]  gpio_out
);
```

3.1.2 記憶體映射

```
0x0000_0000 - 0x3FFF_FFFF : DDR4 (透過 AXI 存取)
0x4000_0000 - 0x4FFF_FFFF : PL 內部 BRAM
0x5000_0000 - 0x5FFF_FFFF : YOLO 加速器控制暫存器
0x6000_0000 - 0x6FFF_FFFF : MMIO (UART, GPIO 等)
0x7000_0000 - 0x7FFF_FFFF : 保留
```

3.2 YOLO V7 硬體加速器

3.2.1 DPU 整合設計

python

```python
# DPU 配置 (使用 Vitis AI)
dpu_config = {
    "arch": "DPUCZDX8G",
    "freqency": 300,  # MHz
    "ram_usage_low": True,
    "channel_augmentation": True,
    "dwcv": True
}
```

3.2.2 加速器介面

verilog

```verilog
module yolo_accelerator(
    input         clk,
    input         rst_n,

    // AXI4-Stream 視訊輸入
    input  [23:0] s_axis_video_tdata,
    input         s_axis_video_tvalid,
    output        s_axis_video_tready,
    input         s_axis_video_tlast,
    input         s_axis_video_tuser,

    // AXI4-Stream 結果輸出
    output [31:0] m_axis_result_tdata,
    output        m_axis_result_tvalid,
    input         m_axis_result_tready,
    output        m_axis_result_tlast,

    // 控制介面 (AXI4-Lite)
    input  [11:0] s_axi_lite_awaddr,
    input         s_axi_lite_awvalid,
    output        s_axi_lite_awready,
    input  [31:0] s_axi_lite_wdata,
    input         s_axi_lite_wvalid,
    output        s_axi_lite_wready,

    // 中斷
    output        interrupt_done
);
```

3.3 視訊處理管線

3.3.1 視訊輸入處理

verilog

```verilog
module video_input_pipeline(
    input         clk,
    input         rst_n,

    // MIPI CSI-2 介面
    input  [1:0]  mipi_data_p,
    input  [1:0]  mipi_data_n,
    input         mipi_clk_p,
    input         mipi_clk_n,

    // AXI4-Stream 輸出
    output [23:0] m_axis_video_tdata,
    output        m_axis_video_tvalid,
    input         m_axis_video_tready,
    output        m_axis_video_tlast,
    output        m_axis_video_tuser,

    // 配置介面
    input  [31:0] img_width,
    input  [31:0] img_height,
    input         enable
);
```

4. 軟體架構設計

4.1 作業系統分層

4.1.1 ARM A53 (Linux) - 主控制器

- **作業系統**: Ubuntu 22.04 或 PetaLinux
- **主要功能**:
  - 系統管理和使用者介面
  - 網路通訊
  - 檔案系統管理
  - RISC-V 程式載入

4.1.2 ARM R5F (Real-time) - 即時控制

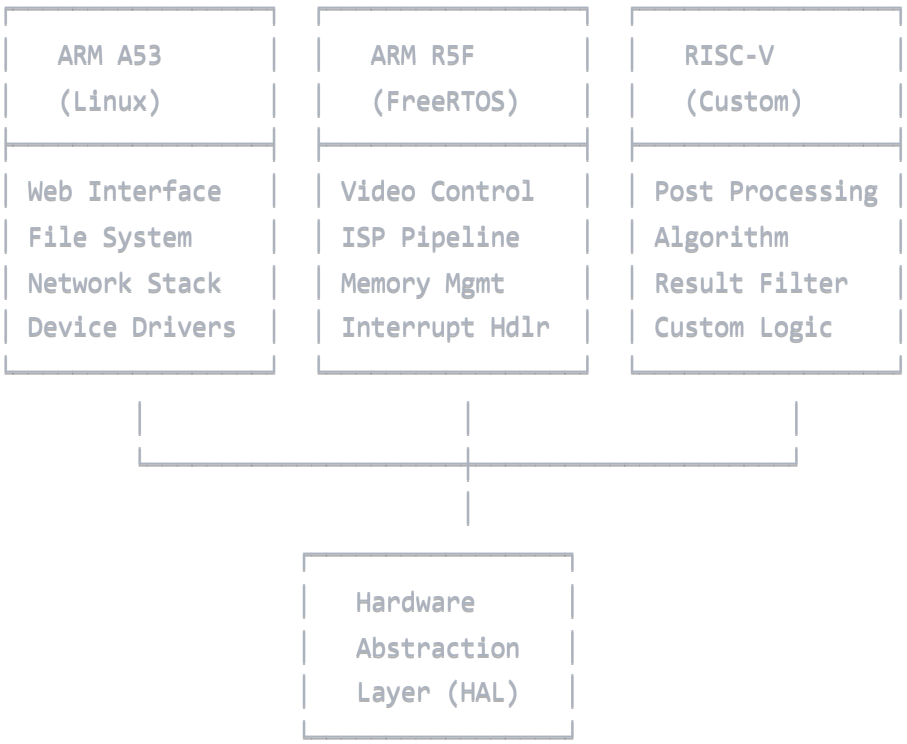
- **作業系統**: FreeRTOS 或 bare-metal
- **主要功能**:
  - 視訊 pipeline 控制
  - 即時影像預處理

- 中斷處理

4.1.3 RISC-V (Custom) - 專用運算

- **作業系統**: 輕量級 RTOS 或 bare-metal
- **主要功能**:
  - YOLO 後處理
  - 檢測結果濾波
  - 自定義演算法

4.2 軟體架構圖



5. 開發流程

5.1 硬體開發

5.1.1 工具鏈設定

bash

```bash
# Vivado 工程設置
source /opt/Xilinx/Vivado/2023.1/settings64.sh
source /opt/Xilinx/Vitis/2023.1/settings64.sh

# 建立專案
vivado -mode batch -source create_project.tcl
```

5.1.2 RISC-V 核心移植

1. **修改 Rocket Chip 配置**:

```scala
// 針對 Kria KV260 的配置
class KV260RocketConfig extends Config(
  new WithNBigCores(1) ++
  new WithRV32 ++
  new WithoutFPU ++
  new WithNMemoryChannels(1) ++
  new WithNBanks(1) ++
  new BaseConfig
)
```

2. **AXI 介面整合**:
   - 將 TileLink 轉換為 AXI4
   - 實現 AXI4-Lite 控制介面
   - 設計中斷控制器

5.1.3 YOLO 加速器整合

```tcl
# DPU IP 核心整合
create_ip -name dpu -vendor xilinx.com -library ip -module_name dpu_0
set_property -dict [list \
  CONFIG.DPU_NUM {1} \
  CONFIG.DPU_ARCH {4096} \
  CONFIG.DPU_RAM_USAGE {low} \
] [get_ips dpu_0]
```

5.2 軟體開發

5.2.1 交叉編譯工具鏈

```bash
# RISC-V 工具鏈
export RISCV=/opt/riscv
export PATH=$RISCV/bin:$PATH

# ARM 工具鏈
export CROSS_COMPILE=aarch64-linux-gnu-
```

5.2.2 核心間通訊

c

```c
// ARM A53 端
typedef struct {
    uint32_t command;
    uint32_t addr;
    uint32_t data;
    uint32_t status;
} ipc_message_t;

int send_to_riscv(ipc_message_t *msg) {
    // 透過 AXI GPIO 發送
    writel(msg->command, RISCV_CMD_REG);
    writel(msg->addr, RISCV_ADDR_REG);
    writel(msg->data, RISCV_DATA_REG);

    // 觸發中斷
    writel(1, RISCV_IRQ_REG);

    return 0;
}
```

c

```c
// RISC-V 端
void riscv_ipc_handler(void) {
    uint32_t cmd = readl(CMD_REG);
    uint32_t addr = readl(ADDR_REG);
    uint32_t data = readl(DATA_REG);

    switch(cmd) {
        case CMD_PROCESS_FRAME:
            process_yolo_results(addr, data);
            break;
        case CMD_UPDATE_PARAMS:
            update_algorithm_params(addr, data);
            break;
    }

    // 清除中斷
    writel(0, IRQ_CLR_REG);
}
```

6. YOLO V7 模型最佳化

6.1 模型量化

python

```python
# 使用 Vitis AI 進行量化
from vai_q_pytorch import QuantModel

# 載入預訓練模型
model = torch.load('yolov7.pth')

# 建立量化模型
quant_model = QuantModel(model, dummy_input)

# 校準
quant_model.calibrate(calib_loader)

# 量化
quant_model.export()
```

6.2 模型編譯

bash

```bash
# 編譯為 DPU 指令
vai_c_xir \
  --xir_path yolov7_quantized.xir \
  --arch /opt/vitis_ai/arch/DPUCZDX8G/KV260/arch.json \
  --output_dir ./compiled_model \
  --net_name yolov7_kv260
```

7. 效能最佳化策略

7.1 硬體最佳化

- **管線化設計**: 重疊視訊輸入、處理和輸出
- **記憶體頻寬最佳化**: 使用 burst 傳輸
- **時脈域最佳化**: 不同模組使用適當的時脈頻率

7.2 軟體最佳化

- **零拷貝**: 使用 DMA 直接記憶體存取
- **預取策略**: 預先載入下一幀資料
- **負載平衡**: 在不同處理器間分配工作負載

8. 驗證與測試

8.1 單元測試

- RISC-V 核心功能測試
- YOLO 加速器精度測試
- 視訊 pipeline 延遲測試

8.2 整合測試

- 端對端物件檢測測試
- 即時效能測試
- 功耗測試

8.3 測試腳本範例

bash

```bash
#!/bin/bash
# 整合測試腳本

echo "開始 YOLO V7 + RISC-V 整合測試..."

# 載入 FPGA bitstream
fpgautil -b design.bit.bin

# 載入 RISC-V 程式
./load_riscv_firmware.sh

# 啟動 YOLO 推理
python3 test_yolo_inference.py --input test_video.mp4

# 檢查結果
python3 validate_results.py

echo "測試完成"
```

9. 預期效能目標

- **影片處理**: 1080p@30fps
- **檢測延遲**: < 100ms
- **功耗**: < 15W (整體系統)
- **檢測精度**: mAP > 0.5 (與原始 YOLO V7 相比 > 95%)

10. 部署與維護

10.1 系統部署

- 建立 SD 卡映像檔

- 設計 OTA 更新機制
- 提供 Web 管理介面

10.2 監控與維護

- 系統健康監控
- 效能資料收集
- 遠端診斷功能

這個架構設計提供了一個完整的解決方案，將 YOLO V7、RISC-V 和 Kria KV260 的優勢結合起來，實現高效能的邊緣 AI 視覺處理系統。