TRƯỜNG ĐẠI HỌC BÁCH KHOA TP.HCM
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

ĐỒ ÁN THIẾT KẾ
KỸ THUẬT MÁY TÍNH

Thiết kế SoC RISC-V tích hợp EdgeAI cho ứng dụng IoT

Học kỳ 251

GVHD: PGS. TS. Trần Ngọc Thịnh
ThS. Huỳnh Phúc Nghị

STT	Họ và tên	MSSV	Ghi chú
1	Lâm Nữ Uyển Nhi	2212429	
2	Vũ Đức Lâm	2211824	

TP. Hồ Chí Minh, Tháng 12/2025

Mục lục

Danh sách hình vẽ

Danh sách bảng biểu

Danh mục Ký hiệu và Chữ viết tắt

Ký hiệu	**Ý nghĩa**
H, W	Chiều cao và chiều rộng của đặc trưng đầu vào (Input Feature Map)
C	Số lượng kênh đầu vào (Input Channels)
N_f	Số lượng bộ lọc / Số kênh đầu ra (Number of Filters / Output Channels)
H_{out}, W_{out}	Chiều cao và chiều rộng của đặc trưng đầu ra (Output Feature Map)
R, S	Chiều cao và chiều rộng của bộ lọc (Kernel Height, Kernel Width)
P	Kích thước vùng đệm (Padding)
Str (hoặc U)	Bước trượt (Stride)
T_h	Chiều cao của mảnh dữ liệu đầu vào trong một Pass (Tile Height)
T_c	Số kênh đầu vào được xử lý song song trong một Pass (Tile Input Channels)
T_m	Số bộ lọc được tính toán song song trong một Pass (Tile Output Channels)
T_{ho}	Chiều cao hợp lệ của mảnh dữ liệu đầu ra trong một Pass

Ký hiệu	Ý nghĩa
b	Số chu kỳ đồng hồ để truyền một giá trị dữ liệu (Cycles per Data Transfer)
T_{comp}	Thời gian tính toán (Computation time)
T_{load}	Thời gian nạp dữ liệu (Load time)
T_{store}	Thời gian ghi dữ liệu (Store time)
T_{pass}	Thời gian hoàn thành một Pass
I	Tensor dữ liệu đầu vào
O	Tensor dữ liệu đầu ra
W	Tensor trọng số (Weights)
B	Vector hệ số chệch (Bias)
μ, σ	Giá trị trung bình (Mean) và Phương sai (Variance) trong Batch Norm
γ, β	Tham số tỉ lệ (Scale) và dịch chuyển (Shift) trong Batch Norm

Viết tắt	Ý nghĩa
AI	Trí tuệ nhân tạo (Artificial Intelligence)
CNN	Mạng nơ-ron tích chập (Convolutional Neural Network)
DNN	Mạng nơ-ron sâu (Deep Neural Network)
FPGA	Mảng cổng lập trình được dạng trường (Field-Programmable Gate Array)
SoC	Hệ thống trên chip (System-on-Chip)
RTL	Mức chuyển giao thanh ghi (Register Transfer Level)
IFM	Đặc trưng đầu vào (Input Feature Map)
OFM	Đặc trưng đầu ra (Output Feature Map)
PE	Phần tử xử lý (Processing Element)
PU	Đơn vị xử lý (Processing Unit - Chứa nhiều PE)
PA	Mảng xử lý (Process Array - Chứa nhiều PU)
MAC	Phép tính Nhân-Cộng tích lũy (Multiply-Accumulate)

Ký hiệu	**Ý nghĩa**
DMA	Truy cập bộ nhớ trực tiếp (Direct Memory Access)
AXI-Lite	Giao diện mở rộng nâng cao rút gọn (Advanced eXtensible Interface Lite)
AXI-Stream	Giao diện luồng dữ liệu mở rộng nâng cao (Advanced eXtensible Interface Stream)
BRAM	Block RAM (Bộ nhớ nội trên FPGA)
DMA	Truy cập bộ nhớ trực tiếp (Direct Memory Access)
AXI	Giao diện mở rộng nâng cao (Advanced eXtensible Interface)
DSP	Digital Signal Processing (Khối xử lý tín hiệu số trên FPGA)
LUT	Bảng tra (Look-Up Table)
FF	Flip-Flop
OSPI	Giao diện ngoại vi nối tiếp 8 kênh (Octal Serial Peripheral Interface)
SPI	Giao diện ngoại vi nối tiếp (Serial Peripheral Interface)
UART	Bộ truyền nhận dữ liệu nối tiếp bất đồng bộ (Universal Asynchronous Receiver-Transmitter)
I2C	Giao thức giao tiếp giữa các vi mạch (Inter-Integrated Circuit)
DVP	Cổng dữ liệu hình ảnh kỹ thuật số (Digital Video Port)
GPIO	Cổng vào/ra đa dụng (General Purpose Input/Output)

Chương 1

Giới thiệu đề tài

Chương này trình bày tổng quan về bối cảnh nghiên cứu, xác định mục tiêu cụ thể, phạm vi thực hiện.

1.1 Tổng quan về đề tài

Tên đề tài: Thiết kế SoC RISC-V tích hợp EdgeAI cho ứng dụng IoT.

Đề tài tập trung vào việc thiết kế và phát triển một hệ thống trên chip (SoC) dựa trên vi xử lý RISC-V tích hợp phần tăng tốc EdgeAI (Accelerator), nhằm xử lý các tác vụ trí tuệ nhân tạo ngay tại biên. Hệ thống sẽ được triển khai thử nghiệm trên nền tảng FPGA, với kiến trúc được tối ưu hóa nhằm hướng tới khả năng chuyển đổi sang thiết kế ASIC (Application-Specific Integrated Circuit) trong tương lai.

Bên cạnh việc thiết kế phần cứng, đề tài cũng bao gồm quá trình thử nghiệm hiệu suất hệ thống với một tập dữ liệu cố định và triển khai một số ứng dụng IoT thực tế làm "case study" (ví dụ: nhận diện hình ảnh từ camera) nhằm đánh giá tính khả thi và hiệu quả hoạt động của hệ thống trong môi trường thực tế.

Hệ thống hoàn chỉnh sẽ bao gồm các thành phần chính:

- Lõi vi xử lý RISC-V (CPU Core).
- Bộ tăng tốc mạng nơ-ron tích chập (CNN Accelerator).
- Hệ thống Bus giao tiếp nội bộ (AXI-Lite, AXI-Stream).
- Bộ truy cập bộ nhớ trực tiếp (DMA).
- Các giao tiếp I/O với ngoại vi (OSPI, SPI, UART, I2C, DVP, GPIO, TIMER,...).

1.2 Mục tiêu và Nhiệm vụ nghiên cứu

Mục tiêu chính của đề tài là nghiên cứu, thiết kế và hiện thực một hệ thống trên chip (SoC) hoàn chỉnh tích hợp lõi vi xử lý RISC-V và bộ tăng tốc phần cứng (Hardware Accelerator) cho các tác vụ trí tuệ nhân tạo tại biên (EdgeAI). Cụ thể, đề tài hướng tới các mục tiêu sau:

Về kiến trúc hệ thống: Xây dựng kiến trúc SoC tối ưu năng lượng, sử dụng chuẩn giao tiếp AXI để kết nối giữa vi xử lý trung tâm và khối tăng tốc tính toán.

Về xử lý AI: Thiết kế khối Accelerator chuyên dụng hỗ trợ các phép toán trọng yếu của mạng nơ-ron tích chập (CNN) như AlexNet, VGG16, MobileNetv1, nhằm giảm tải cho CPU và tăng tốc độ xử lý thực tế.

Về ứng dụng thực tế: Tích hợp đầy đủ các giao tiếp ngoại vi (Camera/HDMI DVP, UART, SPI, OSPI, I2C, GPIO, TIMER) để xây dựng một ứng dụng IoT trọn vẹn (ví dụ: nhận diện vật thể hoặc phân loại ảnh) chạy trực tiếp trên nền tảng FPGA và hướng tới ASIC.

Về quy trình thiết kế: Làm chủ quy trình thiết kế từ mức RTL (Verilog) đến mô phỏng (Simulation), tổng hợp (Synthesis) và kiểm tra trên phần cứng thực (FPGA Prototyping).

1.3 Phạm vi đề tài

Để đảm bảo tính khả thi trong khuôn khổ thời gian của đồ án, nhóm thực hiện xác định phạm vi nghiên cứu như sau:

1.3.1 Phạm vi và Giới hạn đề tài

- **Vi xử lý:** Sử dụng lõi PicoRV32 (RISC-V 32-bit - RV32I) mã nguồn mở, tập trung vào việc tích hợp và xây dựng hệ thống bus (System Interconnect) thay vì thiết kế lại kiến trúc nhân CPU.
- **Mô hình AI:** Tập trung hỗ trợ các mạng CNN cơ bản (như LeNet-5, MobileNet dạng rút gọn) đã được lượng tử hóa (Quantization) xuống 8-bit integer để phù hợp với tài nguyên phần cứng, không đi sâu vào việc huấn luyện (training) các mô hình lớn.
- **Nền tảng phần cứng:** Hệ thống được thiết kế bằng ngôn ngữ Verilog và kiểm chứng trên Kit FPGA (AMD Virtex™ 7 FPGA VC707, Arty A7-100T Artix-7 FPGA). Chưa bao gồm các bước thiết kế vật lý (Physical Design) để ra chip ASIC thực tế (Layout, GDSII).

1.3.2 Đối tượng và Công cụ nghiên cứu

- Ngôn ngữ thiết kế: Verilog, C/C++.
- Công cụ mô phỏng và tổng hợp: Vivado Design Suite.
- Framework AI hỗ trợ: PyTorch/TensorFlow (để trích xuất trọng số mô hình).

1.4 Phân chia công việc

Đồ án được thực hiện bởi hai thành viên với sự phân chia công việc cụ thể dựa trên kiến trúc hệ thống như sau:

Bảng 1.1: Bảng phân chia công việc của các thành viên

STT	**Thành viên**	**Nội dung thực hiện**
1	**Lâm Nữ Uyển Nhi** (Chịu trách nhiệm về Accelerator)	• Nghiên cứu lý thuyết về mạng CNN và các kỹ thuật tối ưu phần cứng. • Thiết kế kiến trúc khối CNN Accelerator (PE Array, Buffer Controller). • Hiện thực các khối tính toán: Convolution, Pooling, ReLU. • Viết Testbench kiểm tra chức năng khối Accelerator.
2	**Vũ Đức Lâm** (Chịu trách nhiệm về SoC & System)	• Nghiên cứu kiến trúc RISC-V và chuẩn bus AMBA AXI. • Thiết kế hệ thống SoC: Tích hợp CPU, Interconnect, Memory Controller. • Thiết kế các giao tiếp ngoại vi: UART, SPI, OSPI, I2C, GPIO, TIMER, DVP (Camera/HDMI). • Phát triển Firmware/Driver để điều khiển hệ thống. • Tổng hợp hệ thống lên FPGA và đo đạc hiệu năng.

1.5 Cấu trúc báo cáo

Báo cáo được trình bày trong 7 chương với nội dung cụ thể như sau:

Chương 1 - Giới thiệu đề tài: Trình bày tổng quan, mục tiêu, phạm vi và kế hoạch thực hiện đồ án.

Chương 2 - Cơ sở lý thuyết: Cung cấp kiến thức nền tảng về kiến trúc tập lệnh RISC-V, mạng nơ-ron tích chập (CNN), các chuẩn giao tiếp (AXI, UART, SPI,...) và công nghệ FPGA.

Chương 3 - Phân tích và Kiến trúc hệ thống: Phân tích yêu cầu bài toán, từ đó đề xuất kiến trúc tổng thể của SoC và sơ đồ khối chi tiết.

Chương 4 - Thiết kế Bộ tăng tốc AI (AI Accelerator): Trình bày chi tiết thiết kế phần cứng của khối xử lý CNN, bao gồm kiến trúc mảng tính toán và quản lý dữ liệu.

Chương 5 - Hiện thực SoC và Tích hợp hệ thống: Mô tả quá trình tích hợp các module vào hệ thống bus, thiết kế bộ nhớ và các ngoại vi, cũng như quy trình tổng hợp trên FPGA.

Chương 6 - Đánh giá kết quả: Trình bày phương pháp kiểm thử, kết quả tài nguyên sử dụng (Resource Utilization), công suất tiêu thụ và so sánh hiệu năng thực tế.

Chương 7 - Kết luận và Hướng phát triển: Tóm tắt các kết quả đạt được và đề xuất các hướng cải tiến trong tương lai.

Chương 2

Cơ sở lý thuyết

Chương này cung cấp các kiến thức nền tảng về kiến trúc tập lệnh RISC-V, mô hình mạng nơ-ron tích chập (CNN), các chuẩn giao tiếp dữ liệu (AXI, UART, SPI, OSPI, I2C, Camera/HDMI DVP) và công nghệ FPGA được sử dụng trong đề tài.

2.1 Kiến trúc tập lệnh RISC-V

2.1.1 Tổng quan về kiến trúc RISC-V

RISC-V là một kiến trúc tập lệnh (ISA - Instruction Set Architecture) mã nguồn mở, ra đời vào năm 2010 tại Đại học California, Berkeley. Khác với các kiến trúc thương mại phổ biến như x86 (Intel) hay ARM, RISC-V được thiết kế dựa trên nguyên lý máy tính tập lệnh rút gọn (RISC) thuần túy, loại bỏ các gánh nặng tương thích ngược của các kiến trúc cũ để tối ưu hóa hiệu năng và năng lượng.

Đặc điểm cốt lõi của RISC-V là tính mô-đun hóa và khả năng mở rộng. Kiến trúc này không định nghĩa một tập lệnh khổng lồ duy nhất, mà chia thành:

Tập lệnh cơ sở (Base ISA): Là phần cứng tối thiểu bắt buộc phải

có để một vi xử lý được gọi là RISC-V. Đối với các ứng dụng nhúng 32-bit, chuẩn này là **RV32I** (Base Integer). Nó cung cấp đầy đủ các lệnh để thực thi tính toán nguyên, truy cập bộ nhớ và điều khiển luồng chương trình.

Các phần mở rộng (Extensions): Là các mô-đun tùy chọn để tăng cường sức mạnh xử lý. Ví dụ: M (Integer Multiplication/Division), A (Atomic instructions), F (Single-precision Floating-point), C (Compressed instructions - nén lệnh 16-bit để tiết kiệm bộ nhớ).

Sự kết hợp này tạo nên chuỗi định danh cho vi xử lý, ví dụ **RV32IMAC** biểu thị vi xử lý 32-bit có hỗ trợ nhân chia, thao tác nguyên tử và lệnh nén.

2.1.2 Mô hình lập trình và Tập thanh ghi

Theo đặc tả của RV32I, trạng thái kiến trúc của một luồng xử lý (Hart - Hardware Thread) bao gồm hai thành phần chính: bộ đếm chương trình (PC) và tập thanh ghi mục đích chung (GPR).

2.1.2.1 Bộ đếm chương trình (Program Counter - PC)

PC là một thanh ghi 32-bit lưu trữ địa chỉ của lệnh đang được thực thi. Trong RISC-V, PC không phải là một thanh ghi mục đích chung (không thể đánh địa chỉ trực tiếp như GPR). Giá trị của PC chỉ có thể thay đổi thông qua các lệnh rẽ nhánh, nhảy hoặc lệnh hệ thống. Khi khởi động (Reset), PC sẽ được nạp một địa chỉ cố định (Reset Vector) để bắt đầu chu trình nạp lệnh.

2.1.2.2 Tập thanh ghi mục đích chung (General Purpose Registers)

RV32I cung cấp 32 thanh ghi, được đánh số từ `x0` đến `x31`, mỗi thanh ghi rộng 32-bit (XLEN=32). Để đảm bảo chương trình phần mềm hoạt động chính xác với phần cứng, đặc biệt khi sử dụng bộ công cụ biên dịch **RISC-V GNU Toolchain (GCC)**, các thanh ghi này phải tuân thủ chuẩn Giao diện Nhị phân Ứng dụng (ABI - Application Binary Interface). Trình biên dịch GCC sử dụng các tên quy ước (như `sp`, `ra`, `a0`...) thay vì tên phần cứng (`x2`, `x1`, `x10`...) để quản lý việc gọi hàm và truyền tham số. Chi tiết chức năng được trình bày trong Bảng 2.1.

Bảng 2.1: Tập thanh ghi mục đích chung của RISC-V (RV32I)

Tên thanh ghi	**Tên ABI**	**Mô tả chức năng**	**Lưu bởi**
`x0`	`zero`	Luôn bằng 0 (Hardwired zero)	N/A
`x1`	`ra`	Địa chỉ trả về (Return Address)	Caller
`x2`	`sp`	Con trỏ ngăn xếp (Stack Pointer)	Callee
`x3`	`gp`	Con trỏ toàn cục (Global Pointer)	N/A
`x4`	`tp`	Con trỏ luồng (Thread Pointer)	N/A
`x5`	`t0`	Thanh ghi tạm thời / Địa chỉ trả về thay thế	Caller
`x6 - x7`	`t1 - t2`	Thanh ghi tạm thời (Temporaries)	Caller
`x8`	`s0 / fp`	Thanh ghi lưu trữ / Con trỏ khung (Frame Pointer)	Callee
`x9`	`s1`	Thanh ghi lưu trữ (Saved register)	Callee
`x10 - x11`	`a0 - a1`	Đối số hàm / Giá trị trả về	Caller
`x12 - x17`	`a2 - a7`	Đối số hàm (Function Arguments)	Caller
`x18 - x27`	`s2 - s11`	Thanh ghi lưu trữ (Saved registers)	Callee
`x28 - x31`	`t3 - t6`	Thanh ghi tạm thời (Temporaries)	Caller

Trong đó:

Caller-saved: Giá trị không được bảo toàn qua lời gọi hàm (hàm con có thể ghi đè).

Callee-saved: Giá trị phải được bảo toàn (nếu hàm con muốn dùng, phải lưu ra stack trước và khôi phục lại trước khi return).

2.1.3 Đặc tả tập lệnh cơ sở RV32I

Tập lệnh RV32I bao gồm 47 lệnh cơ bản. Một điểm đặc biệt trong thiết kế của RISC-V là việc cố định độ dài lệnh ở 32-bit và căn chỉnh bộ nhớ theo từ (word-aligned), giúp đơn giản hóa mạch giải mã lệnh và dự đoán rẽ nhánh.

2.1.3.1 Định dạng lệnh (Instruction Formats)

RISC-V sử dụng 6 định dạng lệnh cơ bản (R, I, S, B, U, J). Điểm tối ưu trong thiết kế định dạng lệnh của RISC-V là vị trí của các trường thanh ghi nguồn (`rs1`, `rs2`) và thanh ghi đích (`rd`) luôn được giữ cố định tại các bit giống nhau trong mọi định dạng lệnh (xem Hình 2.1).
Điều này cho phép bộ giải mã (Decoder) có thể bắt đầu đọc dữ liệu từ tập thanh ghi (Register File) ngay lập tức mà không cần phải chờ xác định xong loại lệnh (Opcode), giúp giảm độ trễ trong đường ống xử lý.

Bit	**31...25**	**24...20**	**19...15**	**14...12**	**11...7**	**6...0**
R-type	funct7	rs2	rs1	funct3	rd	opcode
I-type	imm[11:0]		rs1	funct3	rd	opcode
S-type	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
B-type	imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode
U-type	imm[31:12]				rd	opcode
J-type	imm[20|10:1|11|19]				rd	opcode

Hình 2.1: Cấu trúc bit của các định dạng lệnh RV32I

Dưới đây là giải thích chi tiết ý nghĩa của từng loại định dạng lệnh:

R-type (Register): Dùng cho các lệnh thao tác trực tiếp giữa thanh ghi và thanh ghi (ví dụ: `add x1, x2, x3`).

I-type (Immediate): Dùng cho các lệnh thao tác với hằng số ngắn (Immediate) và các lệnh nạp dữ liệu (Load) từ bộ nhớ.

S-type (Store): Dùng chuyên biệt cho các lệnh lưu dữ liệu từ thanh ghi vào bộ nhớ.

B-type (Branch): Dùng cho các lệnh rẽ nhánh có điều kiện (ví dụ: so sánh bằng, so sánh lớn hơn).

U-type (Upper Immediate): Dùng để thao tác với các hằng số lớn (20-bit cao), thường dùng để nạp địa chỉ nền.

J-type (Jump): Dùng cho các lệnh nhảy vô điều kiện (dùng trong gọi hàm hoặc vòng lặp).

Một kỹ thuật quan trọng khác là việc mã hóa giá trị tức thời (Immediate Encoding). Trong các lệnh dạng S và B, các bit giá trị tức thời bị phân mảnh và xáo trộn. Tuy nhiên, việc xáo trộn này được thiết kế có chủ đích để các bit này luôn tương ứng với cùng một vị trí bit đầu ra của bộ tạo giá trị tức thời (Immediate Generator), giúp giảm số lượng tầng logic (Fan-out) trong phần cứng.

2.1.3.2 Phân nhóm chức năng chi tiết

1. Lệnh tính toán số nguyên (Integer Computational Instructions):

Nhóm lệnh này thực hiện các phép toán số học và logic. Chúng không gây ra ngoại lệ số học và không thay đổi bất kỳ cờ trạng thái nào (RISC-V không sử dụng thanh ghi cờ như ARM/x86).

Tính toán với hằng số (I-Type): `ADDI`, `ANDI`, `ORI`, `XORI`, `SLTI` (Set Less Than Immediate). Lệnh `LUI` (Load Upper Immediate) dùng để nạp 20-bit cao vào thanh ghi.

Tính toán giữa các thanh ghi (R-Type): `ADD`, `SUB`, `AND`, `OR`, `XOR`. Lệnh `SLT/SLTU` so sánh hai thanh ghi và ghi giá trị 1 vào đích nếu nhỏ hơn, ngược lại ghi 0.

Dịch bit: `SLL/SLLI` (Dịch trái logic), `SRL/SRLI` (Dịch phải logic - chèn 0), `SRA/SRAI` (Dịch phải số học - giữ nguyên dấu).

2. Lệnh truy cập bộ nhớ (Load and Store Instructions):

RISC-V sử dụng kiến trúc Load-Store thuần túy. Việc tính toán địa chỉ bộ nhớ luôn thông qua công thức: $Address = rs1 + sign\_extend(imm)$.

Load: `LW` (32-bit), `LH` (16-bit), `LB` (8-bit). Các biến thể `LHU` và `LBU` dùng để nạp dữ liệu không dấu, trong đó phần bit cao của thanh ghi đích sẽ được điền 0 (Zero-extension) thay vì mở rộng dấu (Sign-extension).

Store: `SW`, `SH`, `SB`. Lệnh store chỉ lấy các bit thấp tương ứng trong thanh ghi nguồn để ghi vào bộ nhớ.

3. Lệnh điều khiển luồng (Control Transfer Instructions):

RISC-V khác biệt so với các kiến trúc cũ ở chỗ lệnh rẽ nhánh thực hiện so sánh trực tiếp hai thanh ghi.

Rẽ nhánh có điều kiện (Branch): `BEQ` (Bằng), `BNE` (Không bằng), `BLT/BGE` (So sánh có dấu), `BLTU/BGEU` (So sánh không dấu). Việc tách biệt so sánh có dấu và không dấu giúp lập trình viên kiểm soát chính xác các cấu trúc điều khiển.

Nhảy vô điều kiện (Jump):

`JAL` (Jump and Link): Nhảy đến địa chỉ tương đối so với PC, đồng thời lưu địa chỉ lệnh kế tiếp (PC+4) vào thanh ghi `rd` (thường là `ra`).

`JALR` (Jump and Link Register): Nhảy đến địa chỉ tuyệt đối được tính từ thanh ghi cơ sở + offset. Lệnh này hỗ trợ việc gọi hàm qua con trỏ hoặc quay về từ hàm (Return).

4. Lệnh môi trường hệ thống (System Environment):

Hai lệnh quan trọng nhất là `ECALL` (Environment Call) dùng để tạo yêu cầu

phục vụ từ hệ điều hành (System Call) và `EBREAK` (Environment Break) dùng để chuyển quyền kiểm soát cho trình gỡ lỗi (Debugger). Ngoài ra, các lệnh `CSRRW, CSRRS, CSRRC` dùng để đọc/ghi các thanh ghi trạng thái điều khiển (CSR) nhằm quản lý ngắt và cấu hình hệ thống.

2.1.4 Vi xử lý PicoRV32

Trong đồ án này, nhóm thực hiện lựa chọn lõi vi xử lý **PicoRV32** để làm bộ xử lý trung tâm cho hệ thống SoC.

PicoRV32 là một hiện thực phần cứng (CPU Core) của kiến trúc RISC-V, hỗ trợ đầy đủ tập lệnh cơ sở **RV32I**. Đặc điểm nổi bật của PicoRV32 là sự tối ưu hóa về mặt diện tích và tài nguyên trên FPGA, thay vì tập trung vào hiệu năng đường ống (Pipeline) phức tạp. Nó hoạt động dựa trên máy trạng thái đa chu kỳ, cho phép đạt tần số hoạt động cao và dễ dàng tích hợp vào các thiết kế SoC nhỏ gọn phục vụ ứng dụng IoT. Ngoài ra, PicoRV32 cung cấp giao diện đồng xử lý (PCPI), cho phép mở rộng khả năng tính toán thông qua các bộ tăng tốc phần cứng bên ngoài.

2.2 Tổng quan về Mạng nơ-ron tích chập (CNN)

Phần này trình bày các cơ sở lý thuyết về trí tuệ nhân tạo, trọng tâm là các mạng nơ-ron tích chập (CNN). Đồng thời, các phân tích về xu hướng tính toán biên (Edge AI) và đặc tả toán học của các phép tính cốt lõi cũng được thảo luận chi tiết nhằm làm rõ động lực thiết kế phần cứng của đồ án.

2.2.1 Trí tuệ nhân tạo và Học sâu

Trí tuệ nhân tạo (Artificial Intelligence - AI) là lĩnh vực khoa học kỹ thuật với mục tiêu kiến tạo các hệ thống máy móc thông minh, sở hữu khả năng

thực hiện các tác vụ vốn đòi hỏi trí tuệ con người. Là một tập con quan trọng của AI, Học máy (Machine Learning - ML) cho phép máy tính tự học hỏi từ dữ liệu và cải thiện hiệu suất mà không cần lập trình cụ thể cho từng tác vụ. Thay vì dựa vào các quy tắc thủ công tĩnh, các thuật toán ML sử dụng quá trình huấn luyện để xây dựng mô hình giải quyết vấn đề. Trong đó, Học sâu (Deep Learning - DL) là bước tiến vượt bậc của ML, tập trung phát triển các Mạng nơ-ron sâu (Deep Neural Networks - DNNs). Các mạng hiện đại có thể sở hữu từ 5 đến hàng nghìn lớp, vượt xa quy mô của các mạng nơ-ron truyền thống. Sức mạnh vượt trội của DNN nằm ở khả năng phân cấp đặc trưng (Feature Hierarchy). Khi dữ liệu đi qua các lớp của mạng, thông tin được trích xuất theo mức độ trừu tượng tăng dần: từ các đặc trưng cấp thấp như cạnh, đường thẳng ở lớp đầu, đến hình dạng phức tạp ở lớp giữa, và cuối cùng là nhận diện vật thể hoàn chỉnh ở lớp cuối. Cấu trúc này đặc biệt hiệu quả trong các bài toán Thị giác máy tính (Computer Vision) như phân loại ảnh hay xe tự hành.

2.2.2 Xu hướng chuyển dịch tính toán xuống biên (Edge AI)

Vòng đời của một mô hình AI bao gồm hai giai đoạn chính là Huấn luyện (Training) và Suy luận (Inference). Trong khi quá trình huấn luyện đòi hỏi tài nguyên khổng lồ thường thực hiện trên Cloud, quá trình suy luận đang có xu hướng dịch chuyển mạnh mẽ xuống các thiết bị biên (Edge devices/IoT).

Việc đưa tác vụ Inference xuống biên, hay còn gọi là Edge AI, giải quyết được ba thách thức cốt lõi của mô hình tập trung. Thứ nhất là độ trễ (latency), yếu tố sống còn đối với các ứng dụng thời gian thực như xe tự lái, nơi độ trễ đường truyền Cloud có thể gây rủi ro an toàn. Thứ hai là tối ưu hóa băng thông mạng khi không cần truyền tải dữ liệu thô (như video giám sát) lên máy chủ. Cuối cùng là đảm bảo quyền riêng tư và bảo mật

dữ liệu người dùng.

Tuy nhiên, các nền tảng nhúng thường bị giới hạn nghiêm ngặt về ngân sách năng lượng, tài nguyên tính toán và dung lượng bộ nhớ. Do đó, việc thiết kế các kiến trúc phần cứng chuyên dụng (AI Accelerator) để xử lý hiệu quả các thuật toán DNN dưới các ràng buộc này là yêu cầu cấp thiết.

2.2.3 Cơ sở toán học của Mạng Nơ-ron Tích chập (CNN)

Mạng nơ-ron tích chập (CNN) là kiến trúc phổ biến nhất trong Deep Learning để xử lý dữ liệu hình ảnh. Để đảm bảo tính linh hoạt cho phần cứng, đồ án tập trung phân tích đặc tả toán học của hai loại phép tính cốt lõi thường gặp: Standard Convolution và Depthwise Separable Convolution.

2.2.3.1 Standard Convolution (Tích chập tiêu chuẩn)

Standard Convolution thực hiện trượt bộ lọc trên không gian đầu vào và tích lũy giá trị qua toàn bộ chiều sâu kênh (Channels). Giá trị đầu ra O tại kênh m, vị trí (h,w) được xác định bởi công thức:

$$O[m][h][w] = B[m] + \sum_{c=0}^{C-1} \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} I[c][h \cdot U + r - P][w \cdot U + s - P] \times W[m][c][r][s] \tag{2.1}$$

Trong đó, U là bước trượt (Stride), P là lượng đệm (Padding), W là trọng số và I là đầu vào. Việc xử lý biên (Padding) đóng vai trò quan trọng để duy trì kích thước không gian, yêu cầu phần cứng phải có logic tự động chèn giá trị 0 (Zero-padding) khi chỉ số truy cập nằm ngoài phạm vi hình ảnh thực tế.

2.2.3.2 Depthwise Separable Convolution

Để tối ưu hóa cho các thiết bị biên có tài nguyên hạn chế, các kiến trúc hiện đại như MobileNet sử dụng kỹ thuật Depthwise Separable Convolution. Kỹ thuật này tách tích chập chuẩn thành hai bước riêng biệt nhằm giảm đáng kể khối lượng tính toán:

1. Depthwise Convolution (DW): Áp dụng bộ lọc riêng biệt cho từng kênh đầu vào mà không tích lũy qua các kênh. Do tính độc lập giữa các kênh, các đơn vị tính toán có thể hoạt động song song hoàn toàn.

$$O_{dw}[c][h][w] = \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} I[c][h \cdot U + r - P][w \cdot U + s - P] \times W_{dw}[c][r][s] \quad (2.2)$$

2. Pointwise Convolution (PW): Là tích chập chuẩn với kích thước kernel 1×1, thực hiện nhiệm vụ trộn thông tin giữa các kênh (channel mixing).

$$O_{pw}[m][h][w] = \sum_{c=0}^{C-1} I[c][h][w] \times W_{pw}[m][c] \quad (2.3)$$

Từ phân tích trên, kiến trúc phần cứng đề xuất cần có khả năng cấu hình linh hoạt (reconfigurable) để hỗ trợ cả chế độ tích lũy theo không gian (cho Standard/Pointwise) và chế độ tính toán độc lập theo kênh (cho Depthwise).

2.2.4 Kỹ thuật Gập Batch Normalization (BN Folding)

Trong giai đoạn suy luận, để giảm thiểu độ phức tạp tính toán, đồ án áp dụng kỹ thuật BN Folding. Lớp Batch Normalization thường đi kèm sau Convolution có các tham số $(\mu, \sigma, \gamma, \beta)$ là hằng số cố định khi suy luận. Ta có thể gộp các phép tính này vào trực tiếp trọng số (W) và bias (B) của lớp Convolution phía trước:

$$W' = W_{orig} \cdot \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}}; \quad B' = \left(B_{orig} - \mu\right) \cdot \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{2.4}$$

Kỹ thuật này giúp loại bỏ hoàn toàn khối tính toán Batch Normalization trên phần cứng, giúp tiết kiệm tài nguyên và giảm độ trễ xử lý mà không làm ảnh hưởng đến độ chính xác của mô hình.

2.2.5 Mô hình và Dữ liệu kiểm thử

Để đánh giá hiệu năng hệ thống, đồ án sử dụng các mô hình đã huấn luyện sẵn (Pretrained Models) được chuẩn hóa qua định dạng ONNX. Các bộ dữ liệu kiểm thử bao gồm MNIST (nhận diện chữ số), CIFAR-10 (phân loại vật thể cơ bản) và ImageNet. Trong đó, ImageNet với 1000 lớp vật thể là chuẩn mực quan trọng để đánh giá độ chính xác Top-1 và Top-5 của các mạng nơ-ron sâu hiện đại.

2.3 Các chuẩn giao tiếp hệ thống

2.3.1 Chuẩn giao tiếp AMBA AXI4

AMBA (Advanced Microcontroller Bus Architecture) là tiêu chuẩn kết nối trên chip (On-Chip Interconnect) phổ biến nhất hiện nay, được phát triển bởi ARM. Trong đó, giao thức AXI (Advanced eXtensible Interface) là chuẩn giao tiếp hiệu năng cao, được thiết kế cho các hệ thống SoC yêu cầu băng thông lớn và độ trễ thấp.

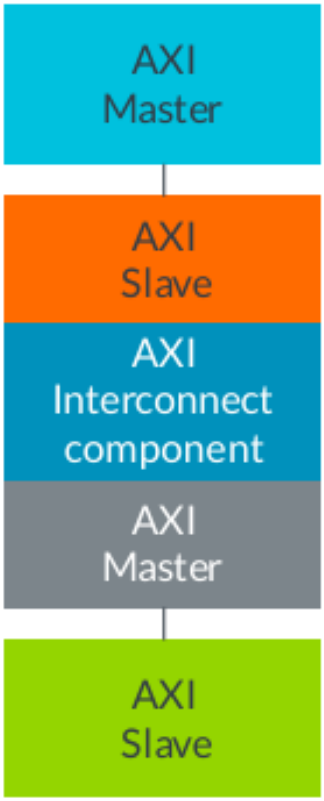
Phiên bản AXI4 (được giới thiệu trong AMBA 4.0) hỗ trợ các tính năng vượt trội so với các thế hệ trước:

Tách biệt hoàn toàn pha địa chỉ/điều khiển và pha dữ liệu.

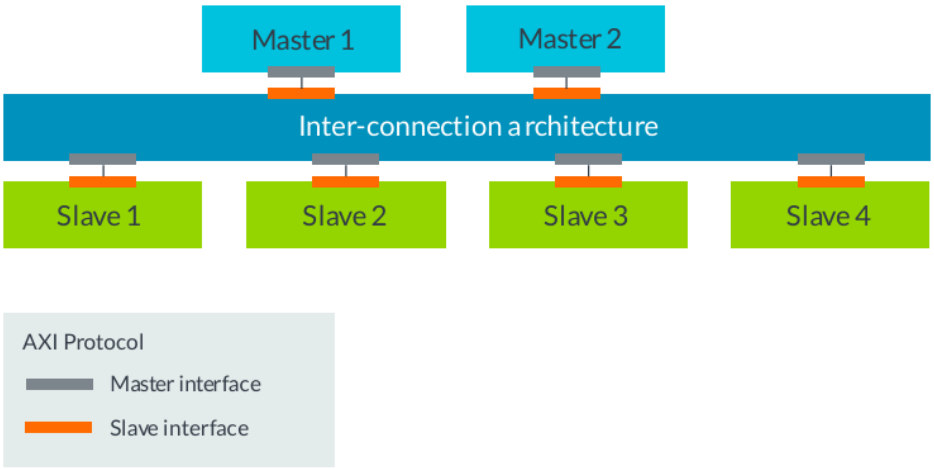
Hỗ trợ giao dịch dữ liệu không thẳng hàng (Unaligned data transfers).

Cho phép phát hành nhiều địa chỉ chờ (Outstanding addresses) trước khi dữ liệu hoàn tất.

Hỗ trợ hoàn thành giao dịch không theo thứ tự (Out-of-order completion) thông qua ID.



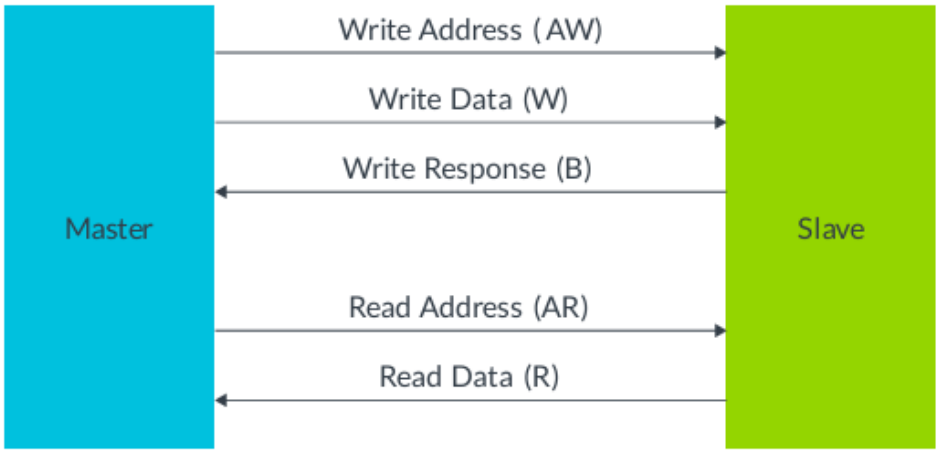
Hình 2.2: a. Tổng quan giao thức AXI4



Hình 2.3: b. Tổng quan giao thức AXI4

2.3.1.1 Kiến trúc 5 kênh độc lập (Channel Architecture)

AXI chia nhỏ một giao dịch truyền thông thành 5 kênh riêng biệt hoạt động song song. Kiến trúc này cho phép đường truyền dữ liệu hai chiều (Full-duplex), nghĩa là Master có thể ghi dữ liệu vào Slave trong khi đang đọc dữ liệu từ Slave khác.



Hình 2.4: Mô hình 5 kênh giao tiếp của AXI4

Năm kênh tín hiệu bao gồm:

1. **Write Address Channel (AW):** Master gửi địa chỉ bắt đầu và thông tin điều khiển (loại burst, độ dài) cho giao dịch ghi. Các tín

hiệu bắt đầu bằng `AW...` (ví dụ: `AWADDR`, `AWVALID`).

2. **Write Data Channel (W):** Master truyền dữ liệu thực tế tới Slave. Kênh này hỗ trợ tín hiệu `WSTRB` (Strobe) để đánh dấu các byte hợp lệ trong một word (hỗ trợ ghi từng byte). Các tín hiệu bắt đầu bằng `W...`.

3. **Write Response Channel (B):** Slave gửi phản hồi trạng thái (OKAY, ERROR) cho Master sau khi toàn bộ dữ liệu đã được ghi thành công. Tín hiệu bắt đầu bằng `B...`.

4. **Read Address Channel (AR):** Master gửi địa chỉ bắt đầu cho giao dịch đọc. Tín hiệu bắt đầu bằng `AR...`.

5. **Read Data Channel (R):** Slave trả về dữ liệu yêu cầu cùng với trạng thái đọc. Tín hiệu bắt đầu bằng `R...`.

2.3.1.2 Cơ chế bắt tay (Handshake Mechanism)

Toàn bộ 5 kênh AXI đều sử dụng chung một cơ chế bắt tay hai chiều `VALID/READY` để điều khiển luồng dữ liệu:

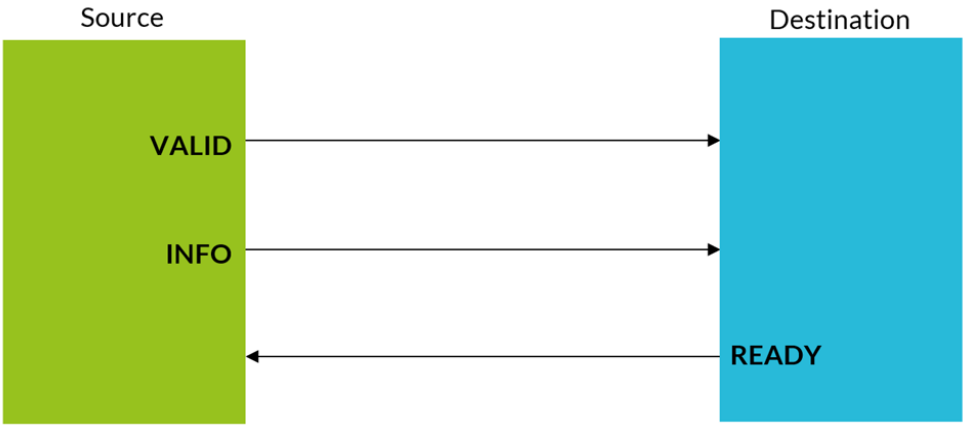
VALID (từ Bên gửi): Báo hiệu rằng dữ liệu hoặc địa chỉ trên đường truyền đã hợp lệ và ổn định.

READY (từ Bên nhận): Báo hiệu rằng bên nhận đã sẵn sàng chấp nhận dữ liệu mới.

Hình 2.5: Cơ chế bắt tay VALID/READY trong AXI

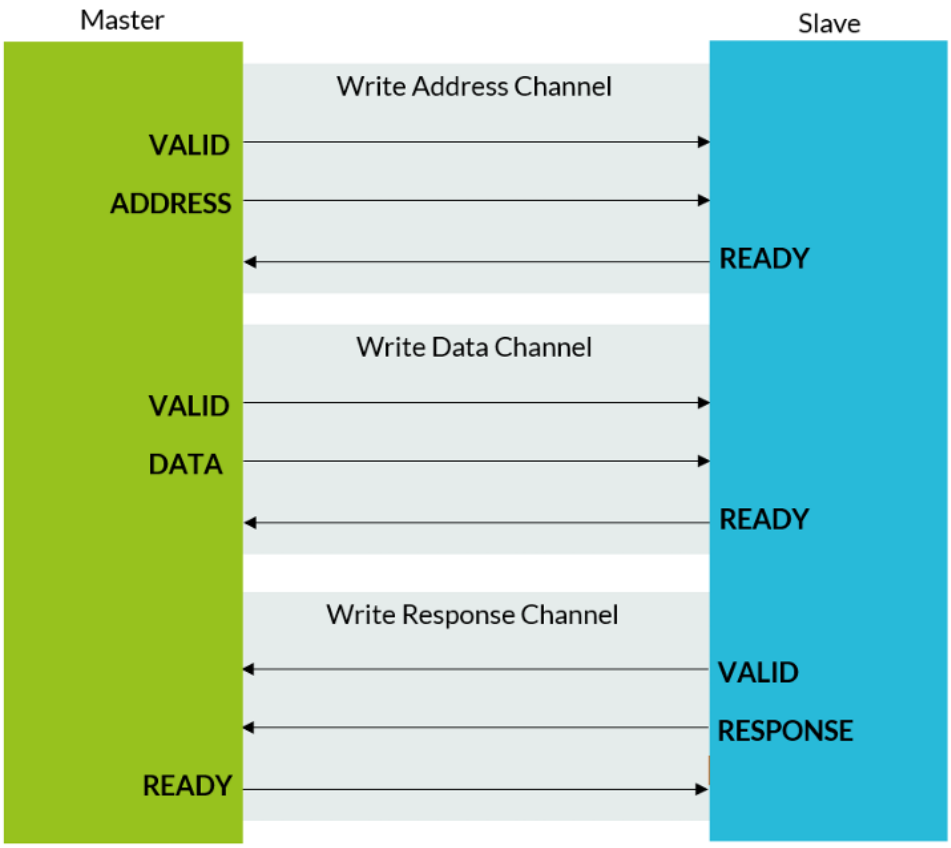
Giao dịch chỉ thực sự diễn ra tại cạnh dương của xung nhịp khi và chỉ khi cả `VALID` và `READY` đều ở mức cao (High). Cơ chế này cho phép bên nhận có thể "kìm" (back-pressure) bên gửi nếu bộ đệm bị đầy, hoặc bên gửi có thể đợi chuẩn bị dữ liệu xong mới phát tín hiệu.
Dựa trên cơ chế bắt tay này, chuẩn AXI định nghĩa hai cấp độ truyền tải dữ liệu cần phân biệt rõ:

Transfer (hoặc Beat): Là một lần trao đổi dữ liệu đơn lẻ thành công (một lần bắt tay VALID/READY = 1). Trong một chuỗi dữ liệu (Burst), mỗi nhịp truyền một gói tin (ví dụ 32-bit) được gọi là một Transfer.



Hình 2.6: Minh họa một Transfer trong AXI

Transaction (Giao dịch): Là một hoạt động đọc hoặc ghi hoàn chỉnh. Một Transaction bao gồm toàn bộ quá trình: gửi địa chỉ (Address Phase), truyền một hoặc nhiều dữ liệu (Data Phase - gồm nhiều Transfers) và nhận phản hồi (Response Phase).



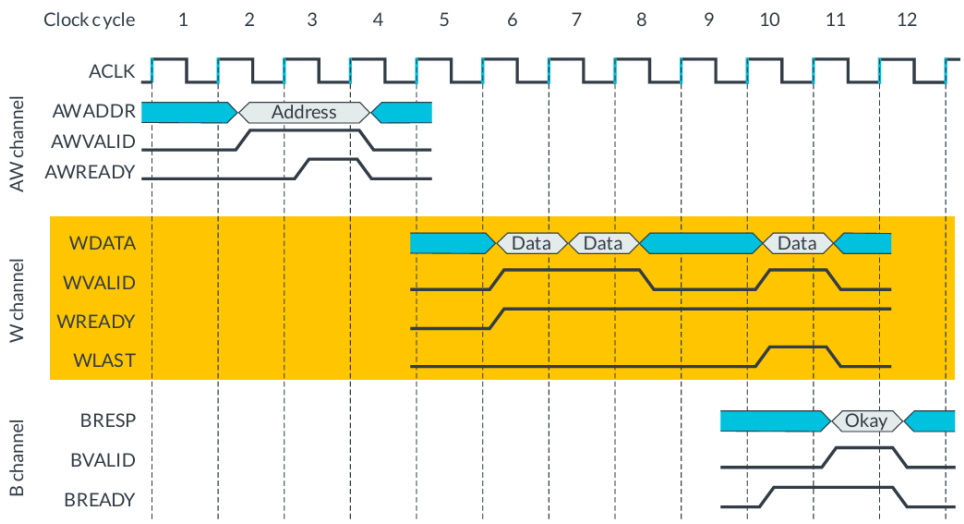
Hình 2.7: Minh họa một Transaction trong AXI

2.3.1.3 Quy trình thực hiện giao dịch chi tiết (Transaction Steps)

Để đảm bảo toàn vẹn dữ liệu, chuẩn AXI quy định chặt chẽ về hướng đi của tín hiệu và trình tự bắt tay giữa Master và Slave. Dưới đây là mô tả chi tiết các tín hiệu tham gia vào hai loại giao dịch cơ bản.

1. Giao dịch Ghi (Write Transaction)

Quá trình ghi dữ liệu diễn ra qua 3 pha, sử dụng các kênh AW, W và B.



Hình 2.8: Giản đồ tín hiệu chi tiết của giao dịch Ghi

Pha địa chỉ (Write Address Channel):

Master → **Slave:** Master đặt địa chỉ lên bus `AWADDR` và các thông tin điều khiển (Burst type, length) lên `AWLEN`, `AWSIZE`... sau đó xác lập tín hiệu `AWVALID = 1`.

Slave → **Master:** Khi Slave sẵn sàng nhận địa chỉ, nó bật `AWREADY = 1`. Giao dịch địa chỉ hoàn tất.

Pha dữ liệu (Write Data Channel):

Master → **Slave:** Master đưa dữ liệu lên bus `WDATA`. Nếu đây là gói cuối cùng trong Burst, Master bật tín hiệu `WLAST = 1`. Đồng thời, Master xác lập `WVALID = 1`.

Slave → **Master:** Slave bật `WREADY = 1` để báo hiệu đã nhận gói dữ liệu đó. Quá trình lặp lại cho đến hết Burst.

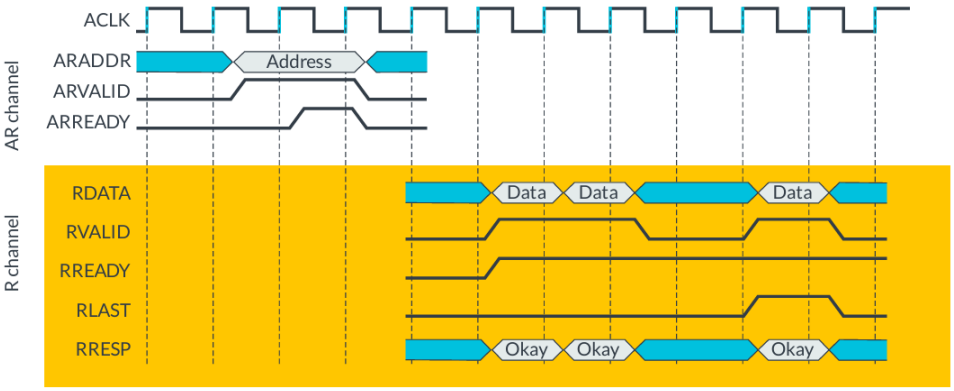
Pha phản hồi (Write Response Channel):

Slave → **Master:** Sau khi nhận đủ dữ liệu và hoàn tất việc ghi vào bộ nhớ, Slave gửi trạng thái (ví dụ: OKAY) qua bus `BRESP` và xác lập `BVALID = 1`.

Master → Slave: Master xác nhận đã nhận được phản hồi bằng cách bật `BREADY = 1`. Kết thúc giao dịch.

2. Giao dịch Đọc (Read Transaction)

Quá trình đọc dữ liệu diễn ra qua 2 pha, sử dụng kênh AR và R.



Hình 2.9: Giản đồ tín hiệu chi tiết của giao dịch Đọc

Pha địa chỉ (Read Address Channel):

Master → Slave: Master đặt địa chỉ cần đọc lên bus `ARADDR` cùng các tham số điều khiển, sau đó bật `ARVALID = 1`.

Slave → Master: Slave chấp nhận địa chỉ bằng cách bật `ARREADY = 1`.

Pha dữ liệu (Read Data Channel):

Slave → Master: Slave truy xuất dữ liệu và đưa lên bus `RDATA`. Nếu thành công, Slave gửi kèm trạng thái OKAY trên bus `RRESP`. Tại gói dữ liệu cuối cùng, Slave bật `RLAST = 1`. Tín hiệu `RVALID = 1` được xác lập khi dữ liệu trên bus là hợp lệ.

Master → Slave: Master nhận dữ liệu bằng cách bật `RREADY = 1`.

2.3.1.4 Cấu trúc giao dịch Burst (Burst Transaction)

AXI là giao thức dựa trên Burst, nghĩa là chỉ cần gửi một địa chỉ khởi đầu, Master có thể truyền liên tiếp một chuỗi dữ liệu (tức là thực hiện một Transaction gồm nhiều Transfers). Các tham số chính điều khiển Burst bao gồm:

Burst Length (AxLEN): Số lượng gói dữ liệu (beat/transfer) trong một burst. AXI4 hỗ trợ lên đến 256 beat cho kiểu INCR.

Burst Size (AxSIZE): Số byte trong mỗi beat (ví dụ: 4 bytes cho hệ thống 32-bit).

Burst Type (AxBURST): Xác định cách tính địa chỉ cho các beat tiếp theo:

FIXED: Địa chỉ giữ nguyên (dùng cho FIFO).

INCR (Incrementing): Địa chỉ tăng dần (dùng cho RAM). Đây là kiểu phổ biến nhất.

WRAP: Địa chỉ tăng đến giới hạn biên rồi quay vòng (dùng cho Cache Line fill).

2.3.1.5 Các biến thể giao thức trong thiết kế

Trong phiên bản AXI4, chuẩn AMBA định nghĩa thêm các biến thể rút gọn để phù hợp với từng mục đích sử dụng cụ thể:

1. Giao thức AXI4-Lite (AXI-Lite)

AXI4-Lite là một phiên bản rút gọn của AXI4, được thiết kế cho các giao tiếp điều khiển đơn giản, không yêu cầu truyền dữ liệu tốc độ cao (Burst transfer). Đặc điểm chính của AXI4-Lite bao gồm:

Mỗi giao dịch chỉ truyền một gói dữ liệu đơn lẻ (Burst length = 1).

Dữ liệu thường có độ rộng 32-bit hoặc 64-bit cố định.

Đơn giản hóa logic điều khiển, giảm diện tích phần cứng.

Nhờ sự đơn giản này, AXI4-Lite thường được sử dụng làm giao diện cấu hình cho các thanh ghi điều khiển (Control Registers) bên trong các khối IP (Intellectual Property).

2. Giao thức AXI4-Stream (AXI-Stream)

AXI4-Stream được thiết kế chuyên biệt cho việc truyền tải các luồng dữ liệu liên tục tốc độ cao (Streaming data) mà không cần sử dụng địa chỉ. Khác với AXI4-Lite hay AXI4-Full (Memory Mapped), AXI4-Stream chỉ tập trung vào việc đẩy dữ liệu từ nguồn (Master) đến đích (Slave) nhanh nhất có thể.

Không có kênh địa chỉ (Address Channel), giảm đáng kể số lượng dây tín hiệu.

Hỗ trợ truyền dữ liệu liên tục không giới hạn độ dài Burst.

Thích hợp cho dữ liệu video, âm thanh hoặc dữ liệu mạng nơ-ron (Feature maps).

2.3.1.6 Áp dụng trong hệ thống đề tài

Trong khuôn khổ đồ án thiết kế SoC RISC-V tích hợp EdgeAI này, nhóm thực hiện áp dụng kết hợp cả hai chuẩn giao tiếp trên để tối ưu hóa hiệu năng và tài nguyên:

Sử dụng AXI4-Lite: Đóng vai trò là kênh điều khiển (Control Plane). Vi xử lý PicoRV32 (Master) sẽ sử dụng AXI4-Lite để ghi vào các thanh ghi cấu hình của khối ngoại vi, khối Accelerator và DMA, thiết lập các thông số như kích thước ảnh, địa chỉ bộ nhớ và tín hiệu bắt đầu (Start).

Sử dụng AXI4-Stream: Đóng vai trò là kênh dữ liệu (Data Plane). Dữ liệu hình ảnh từ Camera và các ma trận trọng số (Weights) sẽ

được truyền trực tiếp từ DMA vào khối Accelerator thông qua AXI4-Stream. Việc loại bỏ overhead của kênh địa chỉ giúp tối đa hóa băng thông xử lý cho mạng CNN.

2.3.2 Giao thức truyền thông UART

UART (Universal Asynchronous Receiver-Transmitter) là một vi mạch phần cứng dùng để truyền tải dữ liệu nối tiếp giữa hai thiết bị. Khác với các giao thức đồng bộ như SPI hay I2C, UART hoạt động theo cơ chế bất đồng bộ (Asynchronous), nghĩa là không cần tín hiệu xung nhịp (Clock) chung để đồng bộ hóa việc truyền nhận giữa bên gửi và bên nhận. Trong các thiết kế SoC, UART thường được tích hợp như một khối ngoại vi (Peripheral) để phục vụ việc gỡ lỗi (Debug), in log hệ thống hoặc giao tiếp với máy tính.

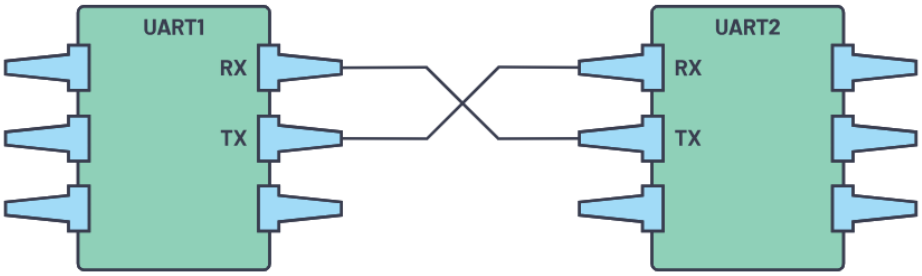
2.3.2.1 Nguyên lý hoạt động

Giao thức UART truyền dữ liệu trên hai dây tín hiệu riêng biệt:

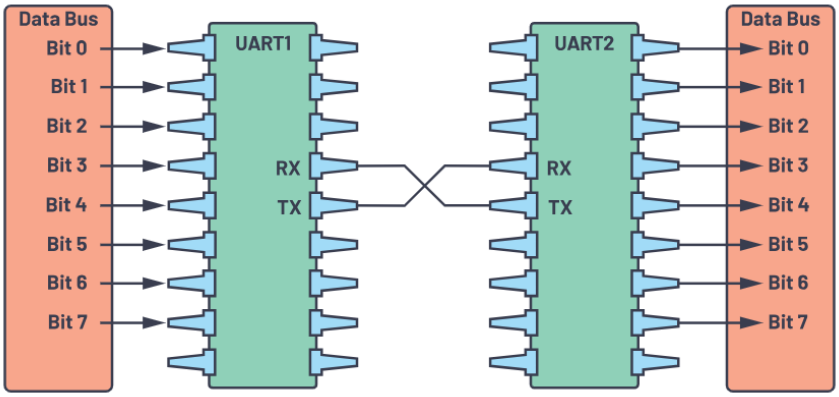
TX (Transmit): Chân truyền dữ liệu đi.

RX (Receive): Chân nhận dữ liệu về.

Để giao tiếp thành công, chân TX của thiết bị này phải được nối với chân RX của thiết bị kia và ngược lại. Quá trình truyền tin diễn ra bằng cách chuyển đổi dữ liệu song song (Parallel data) từ bus hệ thống thành luồng dữ liệu nối tiếp (Serial bit stream) tại phía phát, và khôi phục lại thành song song tại phía thu.



Hình 2.10: Minh họa chân kết nối truyền nhận dữ liệu UART



Hình 2.11: Chuyển đổi dữ liệu song song thành nối tiếp và ngược lại trong UART

2.3.2.2 Cấu trúc khung dữ liệu (Data Frame)

Do không có xung nhịp đồng bộ, UART sử dụng các bit điều khiển đặc biệt để đánh dấu điểm bắt đầu và kết thúc của một gói tin. Một khung dữ liệu chuẩn bao gồm các thành phần sau:

1. **Trạng thái nghỉ (Idle State):** Khi không có dữ liệu truyền, đường truyền luôn được giữ ở mức điện áp cao (Logic 1).

2. **Start Bit:** Để bắt đầu một phiên truyền, thiết bị phát sẽ kéo đường truyền từ mức cao xuống mức thấp (Logic 0) trong một chu kỳ bit. Bên thu phát hiện cạnh xuống này để bắt đầu quá trình đồng bộ.

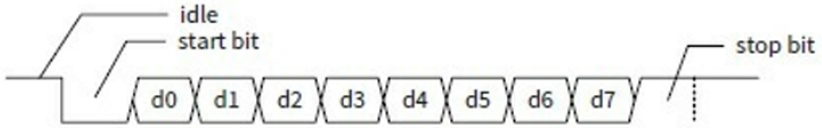
3. **Data Bits:** Chứa thông tin thực tế cần truyền, thường có độ dài từ 5 đến 9 bit (phổ biến nhất là 8 bit). Theo quy ước, bit có trọng số

nhỏ nhất (LSB) được truyền đi trước.

4. **Parity Bit (Tùy chọn):** Dùng để kiểm tra lỗi đơn giản. Bit này có thể được cấu hình là chẵn (Even), lẻ (Odd) hoặc không sử dụng (None). Nếu sử dụng, tổng số bit '1' trong gói dữ liệu (bao gồm cả parity) phải thỏa mãn quy tắc chẵn/lẻ đã thiết lập.

5. **Stop Bit:** Đánh dấu kết thúc gói tin bằng cách kéo đường truyền về mức cao (Logic 1). Độ dài có thể là 1, 1.5, hoặc 2 bit thời gian. Stop bit đảm bảo đường truyền quay về trạng thái nghỉ để sẵn sàng cho Start bit tiếp theo.

Hình 2.12: Khung dữ liệu UART



Hình 2.13: Ví dụ khung dữ liệu UART với 8bit dữ liệu, không parity và 1 stop bit

2.3.2.3 Tốc độ Baud (Baud Rate)

Vì thiếu xung nhịp đồng bộ, hai thiết bị UART phải thống nhất trước một tốc độ truyền nhận, gọi là Baud Rate (đơn vị: bit/giây - bps).

Bên phát sẽ đẩy từng bit dữ liệu ra đường truyền với chu kỳ $T = 1/BaudRate$.

Bên thu sẽ lấy mẫu tín hiệu (sample) tại điểm giữa của mỗi chu kỳ bit dự kiến để đọc dữ liệu.

Theo khuyến cáo kỹ thuật, độ sai lệch tốc độ Baud giữa hai thiết bị không được vượt quá 10% để đảm bảo dữ liệu được đọc chính xác. Các tốc độ phổ biến thường dùng là 9600, 19200, 115200 bps.

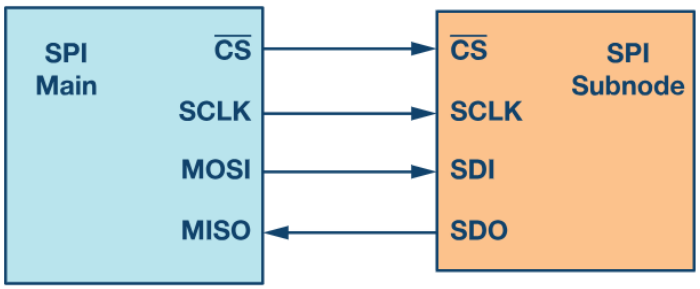
2.3.3 Giao thức truyền thông SPI

SPI (Serial Peripheral Interface) là chuẩn giao tiếp nối tiếp đồng bộ tốc độ cao, hoạt động ở chế độ song công toàn phần (Full-duplex). Chuẩn này được Motorola giới thiệu vào giữa những năm 1980 và hiện nay đã trở thành tiêu chuẩn công nghiệp để kết nối vi xử lý với các thiết bị ngoại vi như cảm biến, bộ nhớ Flash (SPI Flash), màn hình LCD, hoặc bộ chuyển đổi ADC/DAC.

Khác với UART (bất đồng bộ) hay I2C (bán song công, tốc độ thấp), SPI sử dụng đường xung nhịp riêng biệt và kiến trúc Master-Slave chặt chẽ, cho phép đạt băng thông truyền tải rất cao (có thể lên tới hàng chục MHz).

2.3.3.1 Cấu hình tín hiệu vật lý

Một bus SPI tiêu chuẩn (4-wire mode) bao gồm 4 đường tín hiệu logic kết nối giữa Master và Slave.



Hình 2.14: Sơ đồ kết nối tín hiệu chuẩn 4 dây của SPI

Chức năng các chân tín hiệu bao gồm:

SCLK (Serial Clock): Tín hiệu xung nhịp do Master tạo ra. Toàn bộ quá trình truyền nhận dữ liệu được đồng bộ theo cạnh lên hoặc cạnh xuống của xung này. Slave không được phép tạo xung Clock.

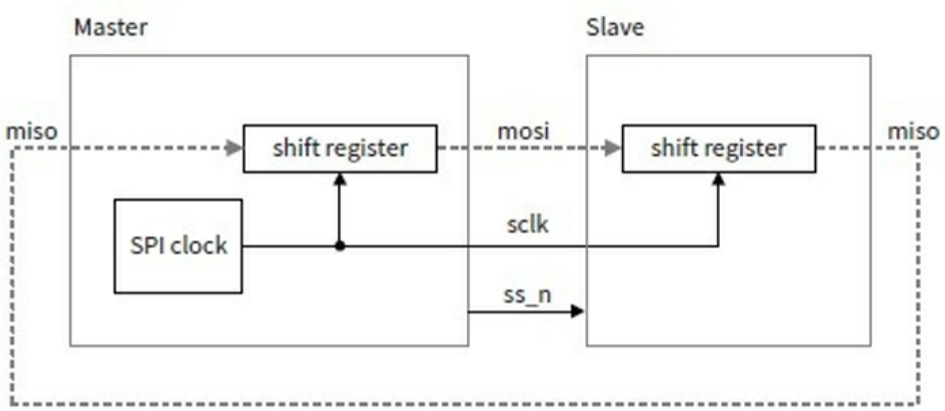
MOSI (Master Out Slave In): Đường truyền dữ liệu từ Master đến Slave.

MISO (Master In Slave Out): Đường truyền dữ liệu từ Slave về Master. Nếu chỉ có Master gửi dữ liệu (ví dụ điều khiển LCD), chân này có thể bỏ qua.

CS/SS (Chip Select / Slave Select): Tín hiệu chọn thiết bị, thường hoạt động ở mức thấp (Active Low). Master kéo chân này xuống 0V để bắt đầu giao dịch với một Slave cụ thể.

2.3.3.2 Cơ chế hoạt động: Thanh ghi dịch (Shift Register)

Cốt lõi của giao thức SPI là cấu trúc thanh ghi dịch vòng tròn (Circular Shift Register).



Hình 2.15: Cơ chế trao đổi dữ liệu dùng thanh ghi dịch trong SPI

Quá trình truyền nhận diễn ra như sau:

1. Master và Slave mỗi bên đều có một thanh ghi dịch (thường là 8-bit hoặc 16-bit).

2. Tại mỗi chu kỳ xung nhịp SCLK:

   1 bit dữ liệu từ Master được đẩy ra đường MOSI và dịch vào thanh ghi của Slave.

Đồng thời, 1 bit dữ liệu từ Slave được đẩy ra đường MISO và dịch vào thanh ghi của Master.

3. Sau N chu kỳ xung nhịp (với N là độ rộng dữ liệu), giá trị trong thanh ghi của Master và Slave được trao đổi hoàn toàn cho nhau.

2.3.3.3 Các chế độ hoạt động (Clock Polarity & Phase)

SPI định nghĩa 4 chế độ hoạt động (Modes) dựa trên trạng thái của xung Clock, được quy định bởi hai tham số:

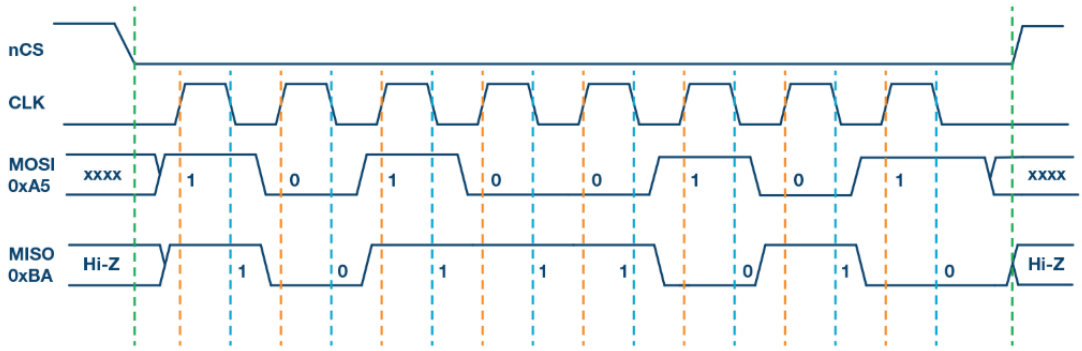
CPOL (Clock Polarity): Trạng thái nghỉ của đường SCLK (0 hoặc 1).

CPHA (Clock Phase): Cạnh lên hoặc xuống của xung dùng để lấy mẫu (Sample) và dùng để thay đổi dữ liệu (Shift).

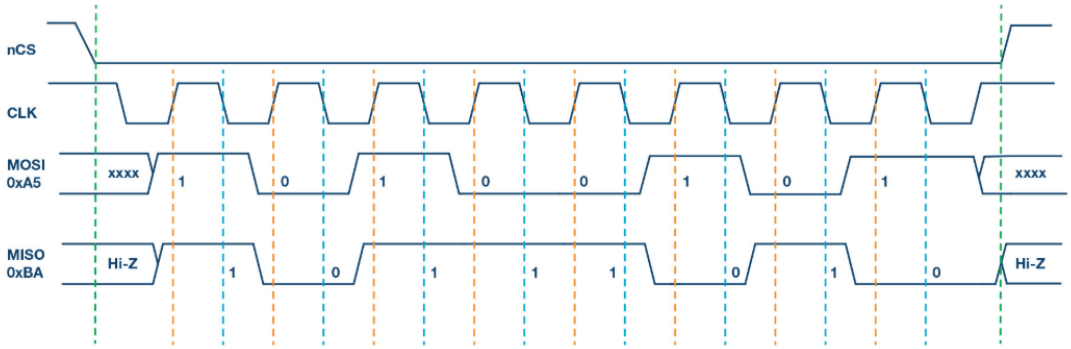
SPI Mode	CPOL	CPHA	Clock Polarity in Idle State	Clock Phase Used to Sample and/or Shift the Data
0	0	0	Logic low	Data sampled on rising edge and shifted out on the falling edge
1	0	1	Logic low	Data sampled on the falling edge and shifted out on the rising edge
2	1	0	Logic high	Data sampled on the falling edge and shifted out on the rising edge
3	1	1	Logic high	Data sampled on the rising edge and shifted out on the falling edge

Hình 2.16: 4 chế độ hoạt động của SPI(CPOL/CPHA)

Lưu ý: Mode 0 và Mode 3 là hai cấu hình phổ biến nhất. Master và Slave phải được cấu hình cùng một Mode để giao tiếp thành công.



Hình 2.17: SPI MODE 0 (CPOL=0, CPHA=0), trạng thái SCLK ban đầu ở mức low, dữ liệu được lấy mẫu tại cạnh lên của SCLK và dịch ở cạnh xuống



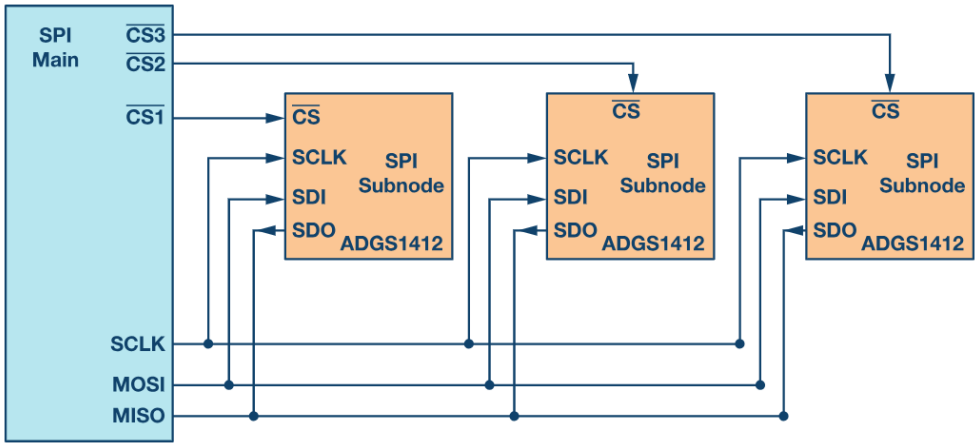
Hình 2.18: SPI MODE 3 (CPOL=1, CPHA=1), trạng thái SCLK ban đầu ở mức high, dữ liệu được lấy mẫu tại cạnh lên của SCLK và dịch ở cạnh xuống

2.3.3.4 Các mô hình kết nối đa thiết bị

SPI cho phép một Master giao tiếp với nhiều Slave thông qua hai cấu hình chính:

1. Cấu hình Slave độc lập (Independent Slaves):

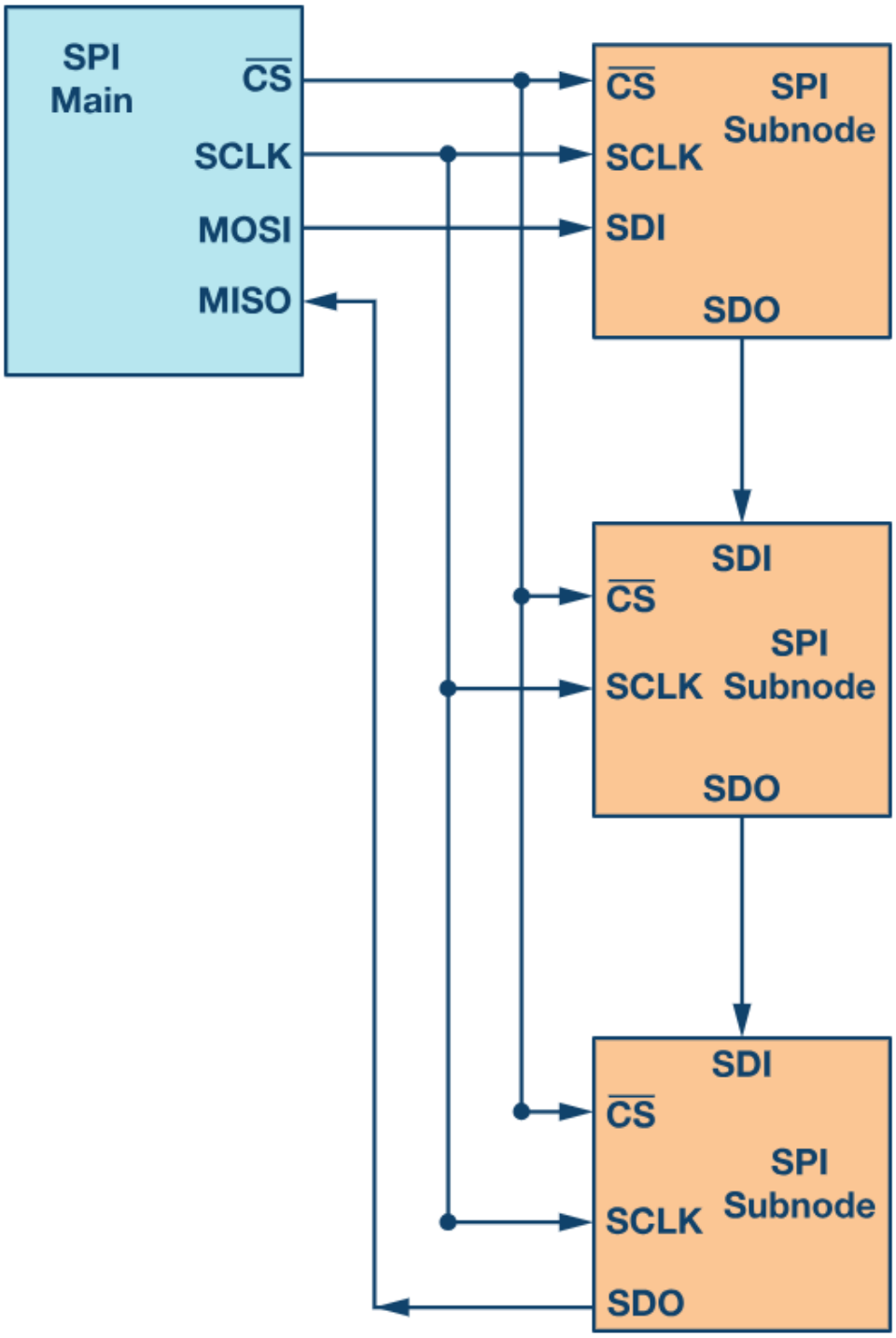
Master sử dụng các chân CS riêng biệt ($CS_1, CS_2, ...$) cho từng Slave. Đây là cấu hình phổ biến giúp tối ưu băng thông.



Hình 2.19: Cấu hình Slave độc lập trong SPI

2. Cấu hình Chuỗi (Daisy Chain):

Các Slave được nối tiếp nhau (MISO của Slave này nối vào MOSI của Slave kia). Dữ liệu đi qua chuỗi các thiết bị, giúp tiết kiệm chân điều khiển của Master nhưng làm giảm tốc độ truyền tổng thể.



Hình 2.20: Cấu hình Chuỗi (Daisy Chain) trong SPI

SPI có tốc độ truyền cao nhất so với UART và I2C, phần cứng đơn giản, hỗ trợ Full-duplex. Nhưng tốn nhiều dây tín hiệu, khoảng cách truyền ngắn, không có cơ chế xác nhận lỗi (ACK) như I2C.

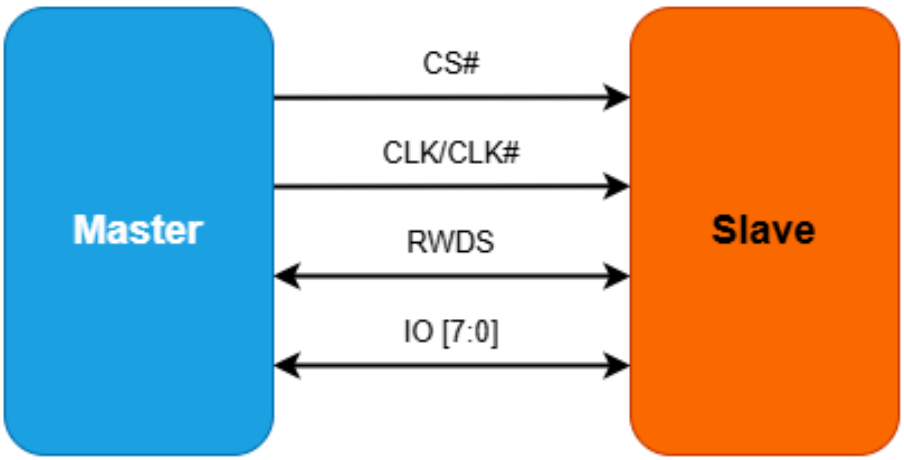
2.3.4 Giao thức truyền thông OSPI (Octal SPI)

Mặc dù giao thức SPI truyền thống có ưu điểm về sự đơn giản, nhưng giới hạn về độ rộng băng thông (chỉ truyền 1 bit mỗi chu kỳ) trở thành nút thắt cổ chai đối với các ứng dụng hiện đại yêu cầu truy xuất dữ liệu lớn như EdgeAI. Để giải quyết vấn đề này, các biến thể mở rộng độ rộng bus dữ liệu đã lần lượt ra đời: từ Dual-SPI (2 đường dữ liệu), Quad-SPI (4 đường dữ liệu - QSPI) và bước tiến mới nhất là **OSPI (Octal SPI)**.

OSPI (còn được gọi là xSPI theo chuẩn JEDEC JESD251) mở rộng giao tiếp lên **8 đường dữ liệu song song**, đồng thời tích hợp công nghệ **DDR (Double Data Rate)**. Đây là giải pháp tối ưu được lựa chọn trong đề tài để kết nối SoC RISC-V với các bộ nhớ ngoài tốc độ cao (như Octal Flash hoặc HyperRAM), đảm bảo khả năng nạp trọng số mạng nơ-ron (Weights) và dữ liệu hình ảnh với độ trễ thấp nhất.

2.3.4.1 Cấu hình tín hiệu vật lý

Để hỗ trợ truyền tải 8 bit song song, giao diện OSPI yêu cầu số lượng chân tín hiệu nhiều hơn so với chuẩn SPI 4 dây truyền thống. Các tín hiệu chính bao gồm:



Hình 2.21: Sơ đồ chân tín hiệu của giao diện OSPI/HyperBus

CLK (Serial Clock): Tín hiệu xung nhịp đồng bộ do Master cấp.

CS/SS (Chip Select): Tín hiệu chọn chip (Active Low).

IO0 - IO7 (Data Lines): 8 đường dữ liệu hai chiều (Bi-directional). Trong một chu kỳ xung nhịp, 8 bit có thể được truyền đi đồng thời (1 Byte).

DQS / DS (Data Strobe): Đây là tín hiệu đặc biệt chỉ xuất hiện trên các chuẩn tốc độ cao (như OSPI và bộ nhớ DDR DRAM).

DQS là tín hiệu hai chiều, được tạo ra bởi thiết bị đang *phát* dữ liệu (Source Synchronous).

Nó đóng vai trò như một xung nhịp tham chiếu đi kèm với dữ liệu, giúp bên thu xác định chính xác thời điểm lấy mẫu dữ liệu hợp lệ mà không bị ảnh hưởng bởi độ trễ đường truyền ở tần số cao.

RWDS (Read Write Data Strobe): Trong giao diện HyperRAM (một biến thể tương tự OSPI), chân này vừa đóng vai trò là DQS, vừa dùng để chỉ thị mặt nạ dữ liệu (Data Mask) khi ghi.

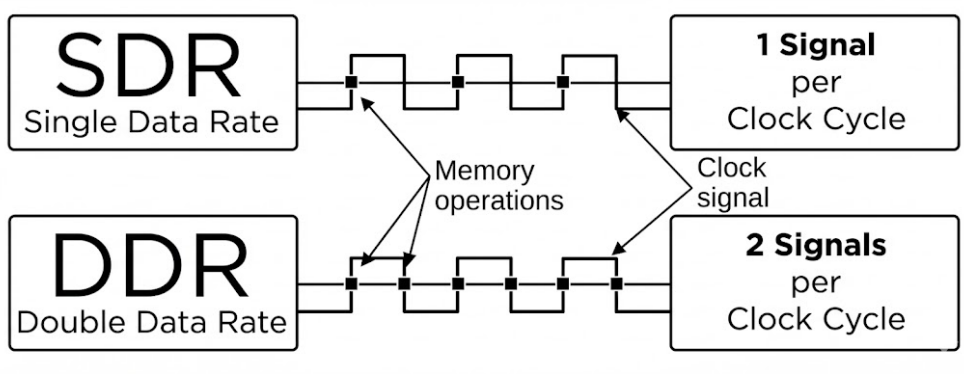
2.3.4.2 Cơ chế truyền tải DDR (Double Data Rate)

Điểm đột phá về hiệu năng của OSPI so với các thế hệ trước (SPI/QSPI) nằm ở khả năng hỗ trợ chế độ **DDR** (còn gọi là DTR - Double Transfer Rate).

1. So sánh SDR và DDR:

SDR (Single Data Rate): Dữ liệu chỉ được truyền ở một **cạnh** của xung nhịp (thường là **cạnh lên**). Đây là cách hoạt động của SPI và QSPI truyền thống.

DDR (Double Data Rate): Dữ liệu được truyền ở **cả cạnh lên và cạnh xuống** của xung nhịp.



Hình 2.22: Giản đồ thời gian truyền tải SDR: Dữ liệu thay đổi ở cạnh lênh, DDR: Dữ liệu thay đổi ở cả hai cạnh của xung nhịp

2. Hiệu năng tính toán: Với giao diện 8 đường dữ liệu (IO0-IO7) hoạt động ở chế độ DDR:

Tại **cạnh lên (Rising Edge)**: Truyền 8 bit.

Tại **cạnh xuống (Falling Edge)**: Truyền 8 bit.

Tổng cộng: 16 bit (2 Bytes) được truyền trong một chu kỳ xung nhịp (Clock Cycle).

Ví dụ: Với xung nhịp 200MHz, băng thông lý thuyết của OSPI DDR đạt: $200\ \text{MHz} \times 2\ \text{Bytes} = 400\ \text{MB/s}$. Tốc độ này nhanh gấp 8 lần so với QSPI thông thường (chạy SDR) và tiệm cận với các giao tiếp DRAM, đủ đáp ứng nhu cầu xử lý thời gian thực.

2.3.4.3 Cấu trúc giao dịch Octal-DDR

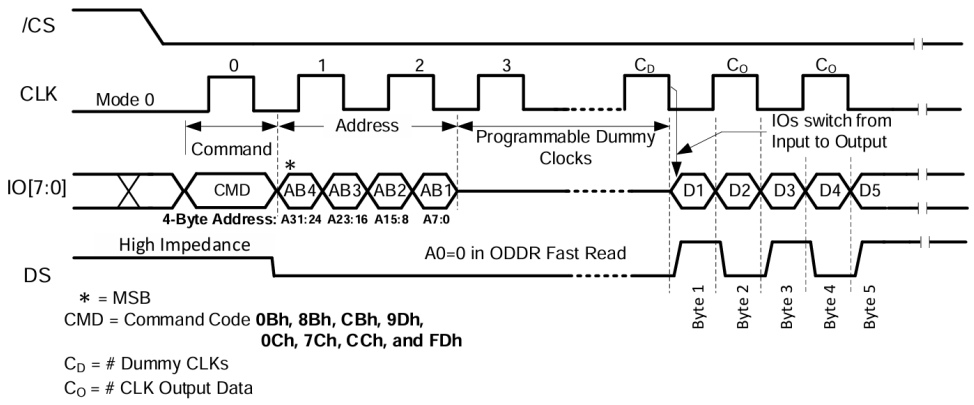
Một giao dịch OSPI điển hình (ví dụ đọc bộ nhớ Flash W35T51NW) diễn ra theo các pha, trong đó toàn bộ Command, Address và Data đều có thể được truyền trên 8 dây (Mode **8-8-8**, nghĩa là sử dụng toàn bộ 8 đường dữ liệu cho cả ba giai đoạn Command Phase, Address Phase và Data Phase):

1. **Command Phase:** Master gửi mã lệnh (8-bit hoặc 16-bit) trên 8 dây IO.

2. **Address Phase:** Master gửi địa chỉ truy cập (32-bit hoặc 64-bit) trên 8 dây IO theo chế độ DDR.

3. **Dummy Cycles:** Các chu kỳ chờ để bộ nhớ chuẩn bị dữ liệu. Số lượng chu kỳ này có thể cấu hình được để phù hợp với tần số hoạt động.

4. **Data Phase:** Dữ liệu được truyền đi (Write) hoặc nhận về (Read) trên cả 8 dây IO tại cả hai **cạnh** xung nhịp, đồng bộ với tín hiệu DQS.



Hình 2.23: Giản đồ thời gian giao dịch OSPI DDR: Command, Address và Data truyền trên 8 dây IO

2.3.4.4 Ưu điểm trong ứng dụng SoC IoT

Tốc độ cao: Băng thông lớn giúp giảm thời gian nạp Bootloader và nạp trọng số mạng nơ-ron (Weights) vào Accelerator.

Số lượng chân ít: So với các giao tiếp bộ nhớ song song truyền thống (Parallel Flash/SRAM) cần 30-40 chân, OSPI chỉ cần khoảng 12 chân, giúp tiết kiệm diện tích SoC.

2.3.5 Giao thức truyền thông I2C (Inter-Integrated Circuit)

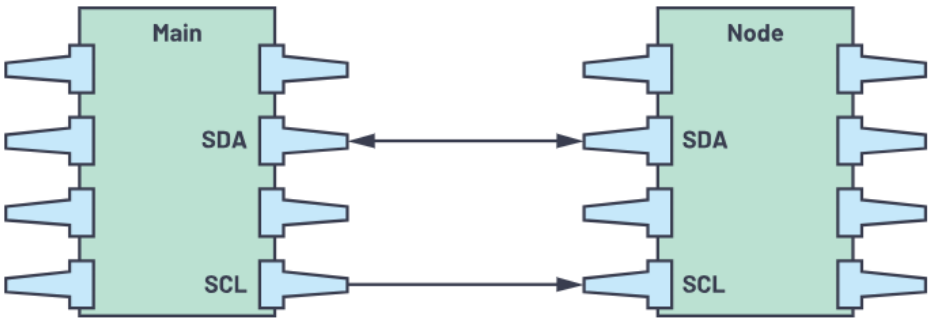
I2C (Inter-Integrated Circuit), thường được viết là I^2C, là một giao thức truyền thông nối tiếp đồng bộ, hoạt động ở chế độ bán song công (Half-duplex). Chuẩn này được Philips Semiconductors (nay là NXP Semiconductors) phát triển vào đầu những năm 1980 với mục đích đơn giản hóa việc kết nối giữa vi xử lý trung tâm và các linh kiện ngoại vi tốc độ thấp trên cùng một bo mạch (như EEPROM, cảm biến nhiệt độ, RTC).
Khác với SPI (cần 4 dây) hay UART (cần 2 dây riêng biệt cho TX/RX), I2C chỉ sử dụng duy nhất **2 đường dây tín hiệu** để kết nối nhiều thiết bị (Multi-master, Multi-slave), giúp tiết kiệm đáng kể số lượng chân IO và diện tích đi dây trên PCB.

2.3.5.1 Cấu hình vật lý và Nguyên lý Open-Drain

Mạng lưới I2C bao gồm hai đường tín hiệu hai chiều (Bidirectional):

SDA (Serial Data): Đường truyền dữ liệu.

SCL (Serial Clock): Đường xung nhịp đồng bộ (thường do Master tạo ra).



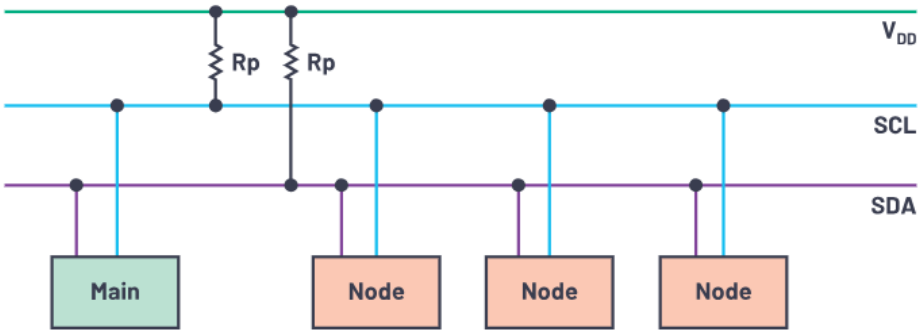
Hình 2.24: Sơ đồ kết nối vật lý I2C

Đặc điểm phần cứng quan trọng nhất của I2C là cấu trúc ngõ ra **(Open-Drain)**

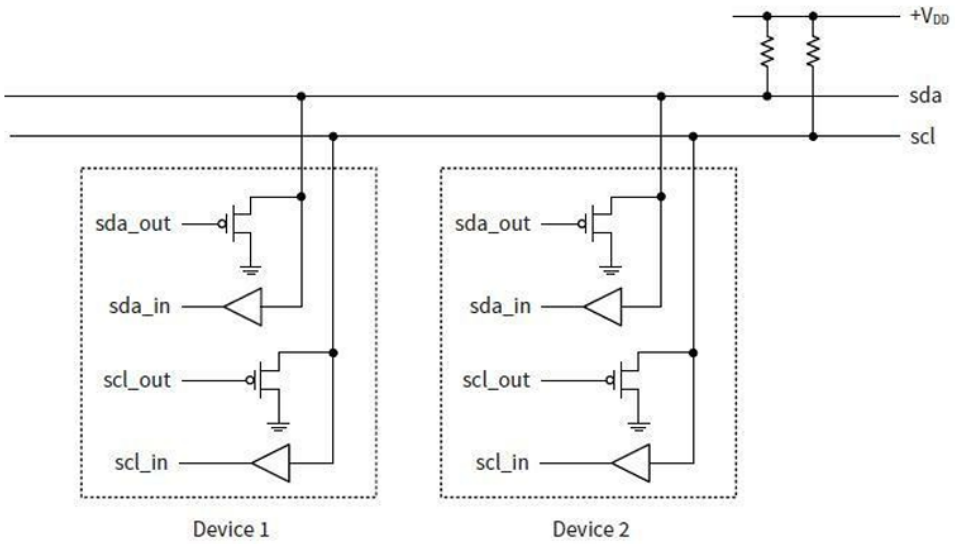
Các thiết bị Master và Slave chỉ có thể kéo đường dây xuống mức thấp (Logic 0). Chúng không thể chủ động đẩy đường dây lên mức cao (Logic 1).

Để tạo ra mức Logic 1, cần phải có các **điện trở kéo lên (Pull-up resistors)** nối từ đường SDA/SCL lên nguồn VCC.

Cơ chế này cho phép thực hiện kỹ thuật "Wired-AND", giúp tránh hiện tượng ngắn mạch khi hai thiết bị cùng lái bus và hỗ trợ tính năng *Clock Stretching* (kéo dãn xung nhịp).



Hình 2.25: a. Điện trở kéo lên (Pull-up Resistors)



Hình 2.26: b. Điện trở kéo lên (Pull-up Resistors)

2.3.5.2 Giao thức truyền dữ liệu

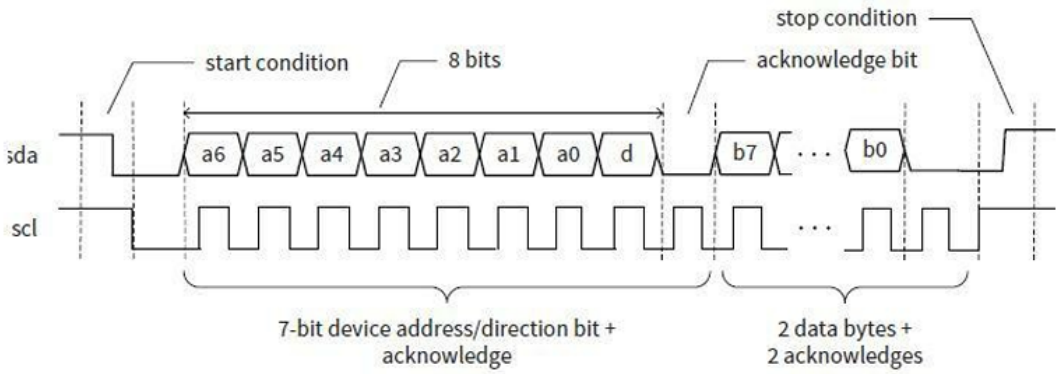
Quá trình truyền tin trên I2C tuân thủ chặt chẽ các quy tắc về định dạng khung (Frame format) và trạng thái bit.

Hình 2.27: Cấu trúc khung truyền dữ liệu I2C

1. Điều kiện Bắt đầu (Start) và Kết thúc (Stop): Thông thường, dữ liệu chỉ được phép thay đổi khi SCL ở mức Thấp. Tuy nhiên, có hai ngoại lệ đặc biệt dùng để báo hiệu trạng thái bus:

Start Condition (S): SDA chuyển từ Cao xuống Thấp trong khi SCL đang ở mức Cao. Báo hiệu bắt đầu một giao dịch.

Stop Condition (P): SDA chuyển từ Thấp lên Cao trong khi SCL đang ở mức Cao. Báo hiệu kết thúc giao dịch.



Hình 2.28: Giản đồ thời gian của điều kiện Start và Stop trong I2C

2. Định dạng địa chỉ và Bit R/W: Mỗi thiết bị Slave trên bus I2C được định danh bởi một địa chỉ duy nhất (thường là 7-bit, hỗ trợ tối đa 128 địa chỉ).

Sau tín hiệu Start, Master gửi 1 byte đầu tiên bao gồm: **7 bit địa chỉ** của Slave + **1 bit R/W** (Read/Write).

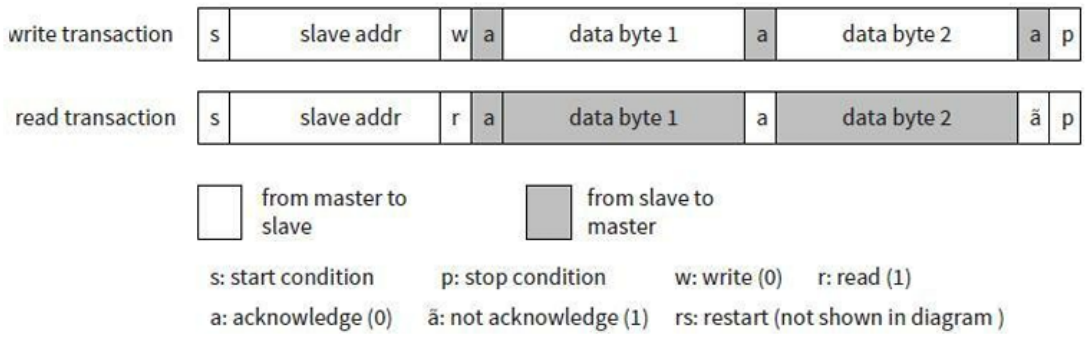
Nếu Bit R/W = 0: Master muốn Ghi dữ liệu vào Slave.

Nếu Bit R/W = 1: Master muốn Đọc dữ liệu từ Slave.

3. Cơ chế xác nhận (ACK/NACK): Sau mỗi byte dữ liệu (8 bit) được truyền đi, bên nhận (Receiver) bắt buộc phải phản hồi bằng một bit xác nhận (Acknowledge bit) trong chu kỳ xung nhịp thứ 9.

ACK (Logic 0): Bên nhận kéo đường SDA xuống thấp, báo hiệu đã nhận byte thành công và sẵn sàng nhận tiếp.

NACK (Logic 1): Bên nhận để đường SDA ở mức cao (do điện trở pull-up). Báo hiệu lỗi, hoặc Slave đang bận, hoặc kết thúc quá trình đọc.



Hình 2.29: Cấu trúc một khung truyền dữ liệu I2C cơ bản

2.3.5.3 Các tốc độ hoạt động

Giao thức I2C hỗ trợ nhiều cấp độ tốc độ khác nhau tùy thuộc vào ứng dụng:

Standard Mode (Sm): Tốc độ lên đến 100 kbps (phổ biến nhất).

Fast Mode (Fm): Tốc độ lên đến 400 kbps.

Fast Mode Plus (Fm+): Tốc độ lên đến 1 Mbps.

High Speed Mode (Hs): Tốc độ lên đến 3.4 Mbps.

Trong phạm vi đồ án này, các ngoại vi cảm biến và cấu hình Camera thường sử dụng Standard Mode.

2.3.5.4 Đánh giá ưu nhược điểm

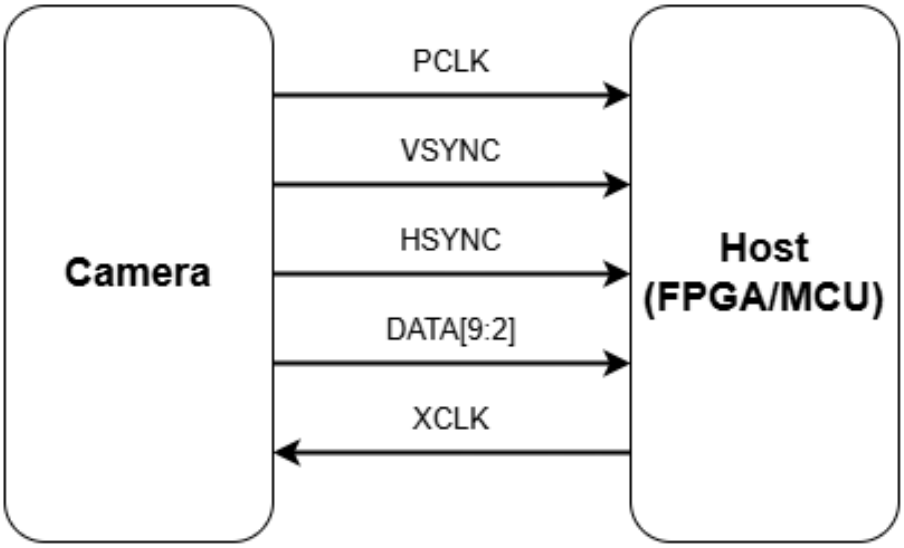
I2C giúp tiết kiệm chân phần cứng (chỉ cần 2 dây cho hàng trăm thiết bị). Có cơ chế xác nhận lỗi (ACK) giúp truyền tin tin cậy. Hỗ trợ đa chủ (Multi-master).
Nhưng về tốc độ chậm hơn nhiều so với SPI và OSPI. Cấu trúc cực máng hở tiêu thụ năng lượng qua điện trở kéo lên. Logic điều khiển phức tạp hơn SPI do phải phát hiện Start/Stop/ACK.

2.3.6 Giao diện Camera song song (DVP Interface)

Trong các hệ thống nhúng và FPGA tầm trung, Digital Video Port (DVP) là chuẩn giao tiếp song song phổ biến nhất để kết nối với cảm biến hình ảnh, điển hình như module OV5640 được sử dụng trong đồ án này. Khác với chuẩn MIPI CSI-2 yêu cầu phần cứng vật lý (PHY) tốc độ cao và logic giải mã phức tạp, DVP hoạt động dựa trên cơ chế truyền dữ liệu song song đồng bộ nguồn (Source Synchronous), giúp đơn giản hóa đáng kể việc thiết kế bộ điều khiển trên FPGA.

2.3.6.1 Đặc tả tín hiệu và Cơ chế vật lý

Về mặt vật lý, giao diện DVP thiết lập một kênh truyền dữ liệu một chiều từ Camera sang FPGA. Tín hiệu xung nhịp điểm ảnh **PCLK** (Pixel Clock) đóng vai trò là nhịp đập trung tâm của hệ thống; toàn bộ dữ liệu trên bus song song chỉ được xác định là hợp lệ tại cạnh tích cực (thường là cạnh lên) của xung PCLK. Để đồng bộ hóa luồng dữ liệu này thành các bức ảnh hoàn chỉnh, DVP sử dụng hai tín hiệu điều khiển quan trọng là **VSYNC** (Vertical Sync) và **HREF** (Horizontal Reference).



Hình 2.30: Sơ đồ kết nối tín hiệu vật lý giữa Camera DVP và FPGA

Tín hiệu VSYNC xác định thời điểm bắt đầu của một khung hình mới. Khi VSYNC được kích hoạt (thường là một xung mức cao ngắn), bộ thu phía FPGA sẽ nhận biết để đặt lại các con trỏ bộ nhớ về vị trí gốc $(0, 0)$. Trong khi đó, tín hiệu HREF chịu trách nhiệm định khung cho từng dòng quét ngang. Dữ liệu trên bus **DATA[9:2]** chỉ được coi là điểm ảnh hợp lệ khi HREF ở mức cao. Ngược lại, khi HREF ở mức thấp, hệ thống hiểu rằng Camera đang trong thời gian nghỉ (Blanking time) để chuẩn bị cho dòng quét tiếp theo. Ngoài ra, FPGA cần cấp một xung nhịp hệ thống **XCLK** (thường là 24MHz) để cảm biến có thể hoạt động và tạo ra PCLK nội bộ.

2.3.6.2 Định dạng dữ liệu RGB565 trên bus 8-bit

Một thách thức kỹ thuật khi sử dụng cảm biến OV5640 ở chế độ DVP là sự chênh lệch về độ rộng dữ liệu. Mặc dù điểm ảnh màu RGB565 yêu cầu 16 bit để biểu diễn (5 bit Red, 6 bit Green, 5 bit Blue), nhưng để tiết kiệm tài nguyên chân I/O trên module Camera, bus dữ liệu thường chỉ được cấu hình sử dụng 8 dây (D[9:2]).

Do đó, một điểm ảnh 16-bit buộc phải được truyền tải trong hai chu kỳ

xung nhịp PCLK liên tiếp. Chu kỳ đầu tiên truyền byte cao (bao gồm phần màu Đỏ và 3 bit cao của màu Lục), và chu kỳ thứ hai truyền byte thấp (bao gồm 3 bit thấp của màu Lục và phần màu Lam). Cơ chế này đòi hỏi bộ điều khiển trên FPGA phải được thiết kế logic ghép kênh (Byte Packing) để tái tạo lại giá trị pixel chính xác trước khi lưu vào bộ nhớ.

Bảng 2.2: Cấu trúc truyền tải Pixel RGB565 qua giao diện 8-bit

Thứ tự	**Dữ liệu trên Bus D[9:2]**	**Thành phần màu tương ứng**
Chu kỳ 1	Byte Cao (D_{High})	R[4:0] (5 bit) + G[5:3] (3 bit)
Chu kỳ 2	Byte Thấp (D_{Low})	G[2:0] (3 bit) + B[4:0] (5 bit)

2.3.6.3 Quy trình thu thập khung ảnh (Frame Capture Sequence)

Để đảm bảo tính toàn vẹn của dữ liệu hình ảnh, khối điều khiển DVP trên FPGA (DVP Capture Core) hoạt động theo một máy trạng thái hữu hạn (FSM) chặt chẽ. Quá trình thu thập một khung ảnh diễn ra tuần tự qua bốn giai đoạn chính.

Hình 2.31: Giản đồ thời gian và trạng thái thu thập khung ảnh

Đầu tiên, hệ thống luôn nằm ở trạng thái chờ đồng bộ khung. Bộ điều khiển sẽ giám sát liên tục tín hiệu **VSYNC**; chỉ khi phát hiện cạnh lên của tín hiệu này, quá trình ghi dữ liệu mới được phép bắt đầu. Sau khi đã đồng bộ được khung hình, hệ thống chuyển sang trạng thái chờ dòng bằng cách theo dõi tín hiệu **HREF**.

Khi HREF chuyển lên mức cao, giai đoạn thu thập dữ liệu tích cực (Active Data Capture) bắt đầu. Tại đây, FPGA thực hiện đọc dữ liệu tại mỗi cạnh lên của PCLK. Do đặc thù truyền tải 2 pha như đã đề cập, bộ điều khiển sẽ sử dụng một thanh ghi đệm tạm thời để lưu byte cao ở chu kỳ đầu, sau đó ghép với byte thấp ở chu kỳ sau để tạo thành một pixel 16-bit hoàn chỉnh ($Pixel = \{Byte_{High}, Byte_{Low}\}$). Giá trị này sau đó được đẩy vào FIFO để chuyển sang miền xung nhịp xử lý.

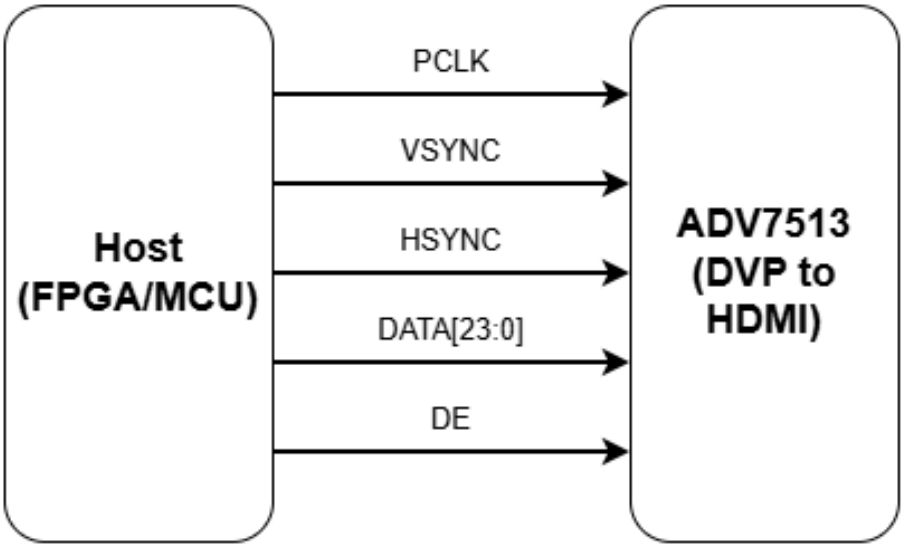
Quá trình này lặp lại liên tục cho đến khi HREF xuống mức thấp, báo hiệu kết thúc một dòng. Cuối cùng, khi VSYNC xuất hiện trở lại, hệ thống kết thúc khung hình hiện tại và quay trở lại trạng thái chờ ban đầu. Việc tuân thủ nghiêm ngặt trình tự này giúp loại bỏ hiện tượng trượt byte (Byte misalignment) và đảm bảo hình ảnh thu được không bị nhiễu hoặc sai lệch màu sắc.

2.3.7 Giao diện hiển thị HDMI (High-Definition Multimedia Interface)

Để hiển thị hình ảnh xử lý từ FPGA lên màn hình độ phân giải cao, đề tài sử dụng chuẩn giao tiếp HDMI. Tuy nhiên, do các chân I/O tiêu chuẩn của FPGA không trực tiếp hỗ trợ mức điện áp và giao thức vật lý của HDMI, hệ thống sử dụng một chip chuyển đổi chuyên dụng (HDMI Transmitter) là **Analog Devices ADV7513**. Đây là IC phát HDMI 1.4a hiệu năng cao, tiêu thụ năng lượng thấp, hỗ trợ độ phân giải lên đến 1080p ở 60Hz và tương thích ngược với các chuẩn HDMI trước đó.

2.3.7.1 Kiến trúc phần cứng hiển thị

Hệ thống xuất hình ảnh bao gồm hai tầng xử lý chính: tầng logic trên FPGA và tầng vật lý trên chip ADV7513.



Hình 2.32: Sơ đồ kết nối tín hiệu giữa FPGA và ADV7513

1. Giao diện đầu vào (FPGA → ADV7513):

FPGA đóng vai trò là nguồn video (Video Source), tạo ra các tín hiệu video số dạng song song. Các tín hiệu này bao gồm:

Video Data Bus (D[23:0]): Bus dữ liệu màu song song. ADV7513 hỗ trợ nhiều định dạng đầu vào như RGB 4:4:4, YCbCr 4:2:2. Trong đồ án này, ta sử dụng định dạng **RGB 24-bit** (8 bit cho mỗi kênh màu).

Sync Signals: Tín hiệu đồng bộ ngang (**HSYNC**) và đồng bộ dọc (**VSYNC**), tương tự như chuẩn VGA truyền thống.

DE (Data Enable): Tín hiệu cho phép dữ liệu. Mức cao (High) báo hiệu rằng dữ liệu trên bus D[23:0] là điểm ảnh tích cực (Active Pixel) và được hiển thị. Mức thấp tương ứng với khoảng thời gian xóa (Blanking period).

CLK (Pixel Clock): Xung nhịp điểm ảnh do FPGA cấp để đồng bộ hóa dữ liệu gửi sang ADV7513.

2. Giao diện đầu ra (ADV7513 → HDMI Connector):
Chip ADV7513 thực hiện mã hóa dữ liệu song song từ FPGA thành tín hiệu nối tiếp tốc độ cao để truyền qua cáp HDMI.

2.3.7.2 Công nghệ truyền dẫn TMDS

Ở phía đầu ra vật lý, HDMI sử dụng công nghệ **TMDS (Transition Minimized Differential Signaling)** - Truyền tín hiệu vi sai cực tiểu hóa chuyển mạch. Công nghệ này giúp truyền tải dữ liệu băng thông lớn với khả năng kháng nhiễu cao. Cáp HDMI tiêu chuẩn bao gồm 4 cặp dây vi sai:

TMDS Clock Channel: Một cặp dây truyền xung nhịp tham chiếu. Tần số của kênh này thường bằng 1/10 tốc độ bit dữ liệu (đối với HDMI 1.4).

TMDS Data Channels (0, 1, 2): Ba cặp dây truyền dữ liệu màu (Red, Green, Blue) và thông tin đồng bộ.

Chip ADV7513 sử dụng thuật toán mã hóa **8b/10b**, chuyển đổi mỗi 8-bit dữ liệu màu thành 10-bit ký tự TMDS nhằm cân bằng dòng DC và giảm thiểu nhiễu điện từ (EMI) trên đường truyền.

2.3.7.3 Cấu hình hoạt động qua I2C

Chip ADV7513 không thể tự động hoạt động ngay khi cấp nguồn mà cần được cấu hình thông qua giao thức **I2C**. FPGA đóng vai trò là I2C Master sẽ ghi vào các thanh ghi của ADV7513 để thiết lập các thông số quan trọng:

Power Management: Kích hoạt các khối chức năng bên trong chip (mặc định chip ở trạng thái ngủ để tiết kiệm điện).

Input Video Format: Khai báo cho chip biết FPGA đang gửi dữ liệu dạng RGB hay YCbCr, căn lề trái hay phải.

Color Space Conversion (CSC): ADV7513 có bộ xử lý phần cứng để chuyển đổi không gian màu (ví dụ từ RGB sang YCbCr cho TV) nếu cần thiết.

Việc thiết kế bộ điều khiển I2C (như đã trình bày ở phần 2.x) là điều kiện tiên quyết để khởi động hệ thống hiển thị HDMI.

2.4 Công nghệ FPGA và Quy trình thiết kế

2.4.1 Tổng quan về công nghệ FPGA

FPGA (Field Programmable Gate Array) là giải pháp vi mạch bán dẫn cho phép tái cấu hình logic sau khi sản xuất, mang lại sự linh hoạt vượt trội so với các thiết kế ASIC cố định. Cấu trúc của FPGA dựa trên một ma trận các khối logic khả trình (Configurable Logic Blocks - CLB) được kết nối với nhau thông qua hệ thống dây dẫn nội bộ linh hoạt (Programmable Interconnects).

Trong lĩnh vực thiết kế SoC và trí tuệ nhân tạo, FPGA mang lại những ưu thế đặc biệt. Khả năng tái cấu hình cho phép các kỹ sư cập nhật thuật toán phần cứng tức thời mà không cần thay đổi bo mạch vật lý. Quan trọng hơn, kiến trúc song song của FPGA rất phù hợp để hiện thực hóa các mảng tính toán Systolic Array trong mạng nơ-ron tích chập (CNN). Điều này giúp giảm thiểu đáng kể rủi ro thiết kế và rút ngắn thời gian đưa sản phẩm ra thị trường (Time-to-market) so với quy trình sản xuất chip ASIC truyền thống.

2.4.2 Kiến trúc phần cứng Xilinx 7-Series

Đề tài được triển khai trên nền tảng kiến trúc **Xilinx 7-Series**. Cấu trúc phần cứng cơ bản của dòng chip này được hình thành từ hai thành phần tài nguyên cốt lõi:

2.4.2.1 Configurable Logic Block (CLB)

CLB đóng vai trò xương sống của FPGA, chịu trách nhiệm thực hiện các hàm logic tuần tự và tổ hợp. Mỗi CLB chứa các đơn vị nhỏ hơn gọi là Slices, bao gồm các bảng tra 6 đầu vào (**LUT6**) có thể cấu hình để thực hiện bất kỳ hàm logic nào, cùng với các phần tử nhớ **Flip-Flop** (FF) để lưu trạng thái và đồng bộ tín hiệu. Ngoài ra, các chuỗi nhớ số học (Carry Chain) tốc độ cao cũng được tích hợp để tối ưu hóa cho các bộ cộng/trừ.

2.4.2.2 Bộ nhớ nội BRAM (Block RAM)

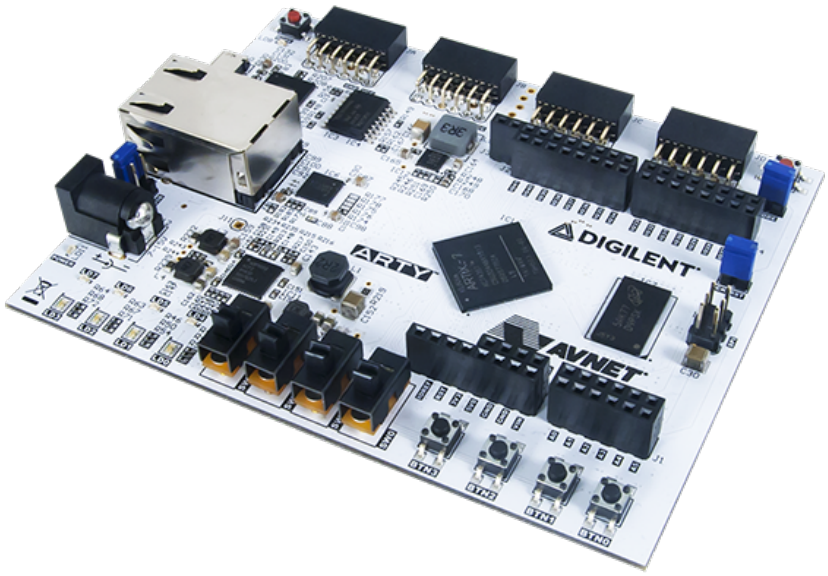
BRAM là các khối bộ nhớ tĩnh (SRAM) dung lượng 36Kb được nhúng rải rác trong FPGA. Chúng đóng vai trò là bộ đệm (Buffer) lưu trữ.

2.4.3 Nền tảng phần cứng thực nghiệm

Quá trình hiện thực hệ thống SoC được tiến hành qua hai giai đoạn thử nghiệm trên hai nền tảng phần cứng khác nhau nhằm đánh giá tính khả thi và tối ưu hóa tài nguyên.

2.4.3.1 Giai đoạn 1: Thử nghiệm trên Digilent Arty A7 (Artix-7)

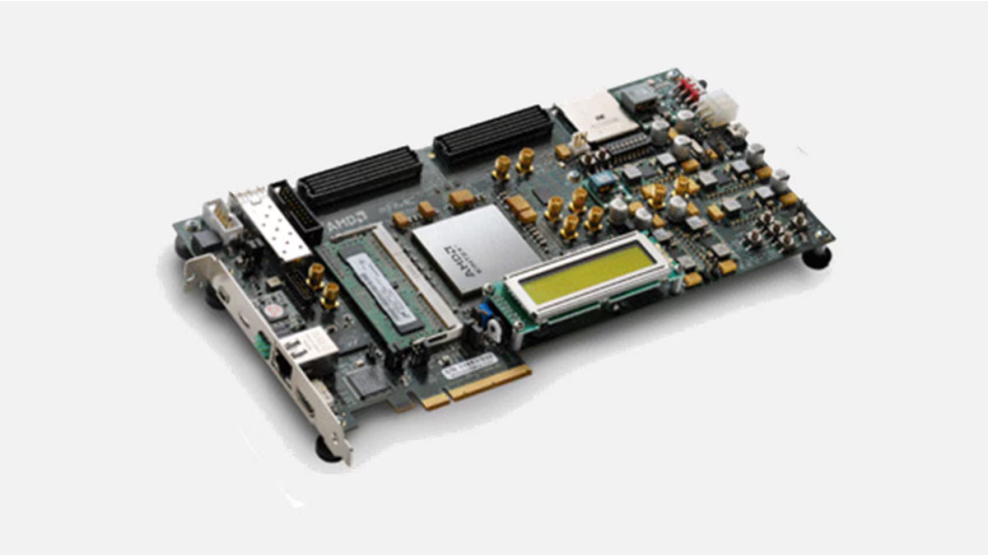
Ở giai đoạn đầu, nhóm nghiên cứu lựa chọn bo mạch **Arty A7-100T** (sử dụng chip XC7A100T) làm nền tảng mục tiêu. Tuy nhiên, trong quá trình thiết kế SoC, giới hạn về tài nguyên phần cứng của chip Artix-7 đã trở thành nút thắt cổ chai và giới hạn việc mở rộng. Cụ thể, số dung lượng bộ nhớ BRAM yêu cầu đã vượt quá khả năng cung cấp của chip, dẫn đến việc không thể tổng hợp (Synthesis) thành công thiết kế tối ưu hoặc phải cắt giảm quá nhiều tính năng quan trọng.



Hình 2.33: FPGA Arty A7-100T

2.4.3.2 Giai đoạn 2: Triển khai trên Xilinx VC707 (Virtex-7)

Để giải quyết bài toán thiếu hụt tài nguyên và tập trung vào việc kiểm chứng kiến trúc hệ thống (Proof of Concept), đề tài đã chuyển sang sử dụng bo mạch **Xilinx VC707 Evaluation Kit** (sử dụng chip Virtex-7 XC7VX485T). Đây là dòng FPGA hiệu năng cao với tài nguyên logic và bộ nhớ vượt trội. Việc chuyển đổi sang VC707 cho phép nhóm hiện thực trọn vẹn kiến trúc SoC, tích hợp vi xử lý PicoRV32 và các ngoại vi tốc độ cao mà không bị giới hạn bởi phần cứng.



Hình 2.34: FPGA Xilinx VC707

Bảng 2.3: So sánh tài nguyên giữa Arty A7 (Thử nghiệm ban đầu) và VC707 (Triển khai chính thức)

Tài nguyên	**Arty A7 (XC7A100T)**	**VC707 (XC7VX485T)**	**Tỷ lệ tăng**
Logic Cells	101,440	485,760	≈ 4.8x
Block RAM	4.8 Mb	37 Mb	≈ 7.7x
DSP Slices	240	2,800	≈ 11.6x
Transceivers	N/A	GTX (12.5 Gbps)	-

Số liệu từ Bảng 2.3 cho thấy sự vượt trội về tài nguyên Logic Cells và Block RAM của VC707, đảm bảo không gian rộng lớn cho việc mở rộng quy mô mảng tính toán Systolic Array.

2.4.4 Quy trình thiết kế trên Vivado

Toàn bộ quy trình hiện thực hệ thống SoC được thực hiện trên môi trường **Xilinx Vivado Design Suite**, tuân thủ luồng thiết kế dựa trên mã nguồn (HDL-based Design Flow) để đảm bảo khả năng kiểm soát chi tiết và tối ưu hóa tài nguyên phần cứng cũng như hướng tới ASIC trong tương lai.

Quy trình bắt đầu bằng giai đoạn **Thiết kế (Design Entry)**, trong đó

toàn bộ hệ thống được mô tả bằng ngôn ngữ **Verilog HDL**. Thay vì sử dụng công cụ thiết kế dạng sơ đồ khối (IP Integrator), các thành phần lõi như PicoRV32, hệ thống Bus AXI4, khối Accelerator, khối DMA và các khối ngoại vi như UART, SPI, OSPI, I2C, DVP,... được kết nối trực tiếp thông qua kỹ thuật khởi tạo module (Module Instantiation) bên trong một tập tin thiết kế đỉnh (Top-level Module). Sau khi hoàn tất mã nguồn, hệ thống trải qua bước **Mô phỏng (Simulation)** hành vi bằng Testbench để kiểm chứng tính đúng đắn của logic trước khi đi vào **Tổng hợp (Synthesis)** để chuyển đổi mã RTL thành danh sách lưới cổng (Netlist). Giai đoạn quan trọng tiếp theo là **Hiện thực (Implementation)**, bao gồm việc sắp xếp linh kiện (Place) và đi dây (Route) trên chip thực tế, quyết định tần số hoạt động tối đa (Fmax) của hệ thống. Cuối cùng, công cụ sẽ thực hiện **Tạo Bitstream** (tệp nhị phân .bit) để nạp cấu hình xuống bo mạch FPGA VC707, hoàn tất quy trình thiết kế phần cứng.

Chương 3

Công trình nghiên cứu liên quan kiến trúc bộ gia tốc CNN

3.1 Kiến trúc tham chiếu: Hệ thống Eyeriss

Để giải quyết bài toán tối ưu hóa năng lượng cho các mạng nơ-ron tích chập (CNN), kiến trúc Eyeriss (Chen et al., 2017) tập trung vào việc cực tiểu hóa chi phí di chuyển dữ liệu thông qua thiết kế phần cứng chuyên biệt và phân cấp bộ nhớ hiệu quả. Hệ thống bao gồm chip tăng tốc kết nối với DRAM ngoại vi qua giao diện bất đồng bộ, cho phép tách biệt miền xung nhịp tính toán và giao tiếp. Trung tâm của kiến trúc là mảng 12×14 phần tử xử lý (PE) hoạt động độc lập, mỗi PE sở hữu bộ nhớ đệm cục bộ (Scratchpads - Spads) để lưu trữ trọng số, dữ liệu đầu vào và các tổng riêng. Thiết kế này tạo nên mô hình phân cấp bộ nhớ bốn mức, từ DRAM, Global Buffer, Inter-PE đến Spads, giúp khai thác tối đa tính cục bộ của dữ liệu.Đóng vai trò trung chuyển là bộ đệm toàn cục (Global Buffer) dung lượng 108KB, giúp giảm thiểu các truy cập bộ nhớ ngoài tốn

kém. Hệ thống sử dụng mạng kết nối trên chip (NoC) tùy biến gồm Mạng đầu vào (GIN) hỗ trợ multicast và Mạng đầu ra (GON). Điểm đặc biệt của Eyeriss là luồng dữ liệu "Row Stationary" (RS), cho phép tái sử dụng dữ liệu hiệu quả ngay tại các bộ nhớ cục bộ trong từng PE, từ đó tối ưu hóa công suất tiêu thụ tổng thể.

3.2 Kiến trúc tham chiếu: Bộ tăng tốc Pixel-Level Fully Pipelined

Nhằm khắc phục nhược điểm về độ trễ và thông lượng của các kiến trúc truyền thống xử lý theo lớp (layer-by-layer), Li và cộng sự (2025) đã đề xuất kiến trúc đường ống toàn phần ở cấp độ pixel (Pixel-Level Fully Pipelined). Thay vì yêu cầu bộ nhớ đệm lớn để lưu trữ bản đồ đặc trưng giữa các lớp, kiến trúc này triển khai toàn bộ mạng nơ-ron thành chuỗi nối tiếp, cho phép luồng pixel được xử lý liên tục từ đầu vào đến đầu ra. Quy trình xử lý dựa trên chiến lược "pixel-by-pixel", trong đó mỗi lớp tính toán tích hợp ba thành phần chính: bộ chọn kênh (MUX), bộ đệm chỉnh lưu (Rectified FIFO) và đơn vị tính toán (CU). Cụ thể, MUX và Rectified FIFO chịu trách nhiệm trích xuất, đồng bộ và chuẩn hóa dữ liệu cửa sổ trượt từ luồng đầu vào để đảm bảo tính liên tục cho đường ống.Tại đơn vị tính toán (CU), hệ thống thực hiện các phép nhân chập song song và áp dụng kỹ thuật ghép kênh theo thời gian (Time-Division Multiplexing). Kỹ thuật này dựa trên các tham số "Initial Sparsity" (IS) và "Pooling Sparsity" (PS), cho phép tái sử dụng tài nguyên DSP cho nhiều tác vụ trong cùng một chu kỳ xung nhịp. Về mặt lưu trữ, toàn bộ trọng số và bias dạng 8-bit fixed-point được lưu trực tiếp trên các khối BRAM nội bộ đặt cạnh đơn vị xử lý. Thiết kế này loại bỏ hoàn toàn việc truy cập DRAM trong quá trình suy luận, giúp giảm độ trễ xuống dưới mức mili-giây và tối đa hóa hiệu quả năng lượng.

3.3 Kiến trúc tham chiếu: Tăng tốc CNN trên FPGA dựa trên OpenCL

Zhang và Li (2017) đã đề xuất một giải pháp tăng tốc CNN sử dụng ngôn ngữ OpenCL nhằm cân bằng giữa hiệu năng phần cứng và tính linh hoạt trong lập trình. Hệ thống vận hành theo mô hình tính toán dị thể, bao gồm một CPU chủ (Host) điều khiển luồng chương trình và FPGA (Device) thực thi các tác vụ tính toán chuyên sâu. Để giải quyết nút thắt về băng thông bộ nhớ, nhóm tác giả xây dựng "Mô hình phân tích cân bằng" (Balance Analysis Model) giúp định lượng mối tương quan giữa năng lực tính toán và băng thông, từ đó xác định cấu hình tài nguyên tối ưu để thông lượng không bị giới hạn bởi tốc độ truy xuất Global Memory.Hiệu năng hệ thống được nâng cao nhờ thiết kế kernel OpenCL tối ưu. Cụ thể, các kernel được thiết kế dạng đường ống sâu (deep pipelining) để thực thi song song các chỉ lệnh tích chập. Đồng thời, hệ thống quản lý bộ nhớ phân cấp bằng cách tận dụng tối đa Local Memory/BRAM để lưu đệm các bản đồ đặc trưng và trọng số, giảm thiểu truy cập bộ nhớ ngoài (Off-chip Memory). Các kỹ thuật tối ưu hóa vòng lặp như trải phẳng (loop unrolling) và chia nhỏ dữ liệu (loop tiling) cũng được áp dụng triệt để. Kết quả thực nghiệm trên Altera Arria 10 cho thấy kiến trúc đạt hiệu suất 866 GOPS và hiệu quả năng lượng vượt trội so với các thiết kế RTL truyền thống.

3.4 Kiến trúc tham chiếu: Hệ thống xử lý dị thể trên nền tảng RISC-V cho IoT

Hướng đến các ứng dụng IoT với ràng buộc khắt khe về tài nguyên, Liu và cộng sự (2020) đề xuất kiến trúc xử lý dị thể kết hợp giữa lõi CPU RISC-V nhúng và khối tăng tốc phần cứng CNN chuyên biệt. Mô hình đồng thiết kế phần cứng/phần mềm này phân chia trách nhiệm rõ ràng: CPU RISC-V

đóng vai trò bộ xử lý đa dụng, quản lý luồng chương trình và các tác vụ tiền/hậu xử lý ở tần số thấp (20 MHz) để tiết kiệm năng lượng nền.Trong khi đó, khối CNN Accelerator đảm nhận các phép toán chuyên sâu ở tần số cao hơn (100 MHz) nhằm đảm bảo thông lượng. Giao tiếp giữa hai thành phần được thực hiện qua cơ chế "lệnh vĩ mô" (macro instructions), cho phép CPU cấu hình và kích hoạt Accelerator xử lý trọn vẹn các lớp mạng phức tạp mà không cần can thiệp liên tục. Kiến trúc này minh chứng cho tính hiệu quả khi kết hợp sự linh hoạt của tập lệnh mở RISC-V với hiệu năng xử lý song song của các bộ tăng tốc miền cụ thể (domain-specific accelerators).

3.5 Kiến trúc tham chiếu: Bộ tăng tốc luồng cấu hình lại (RSA) cho IoT

Du và cộng sự (2017) giới thiệu kiến trúc "Reconfigurable Streaming Architecture" (RSA) dành cho các thiết bị IoT, với đặc điểm cốt lõi là khả năng xử lý dữ liệu theo luồng liên tục (streaming). RSA loại bỏ hoàn toàn nhu cầu lưu trữ các bản đồ đặc trưng trung gian vào DRAM, giúp giảm đáng kể độ trễ và năng lượng tiêu thụ. Thay vì lưu trữ toàn bộ khung hình, hệ thống sử dụng các bộ đệm dòng (Line Buffers) dựa trên FIFO để lưu tạm thời các dòng pixel đầu vào cần thiết cho cửa sổ trượt, giúp tối ưu hóa tài nguyên bộ nhớ on-chip.Các phép tính tích chập, pooling và kích hoạt được thực hiện bởi các Đơn vị tính toán cấu hình lại, kết nối qua mạng lưới chuyển mạch tùy biến. Thiết kế này cho phép định tuyến luồng dữ liệu động để hỗ trợ đa dạng kích thước kernel (như 3×3, 1×1) mà không cần thay đổi phần cứng vật lý. Đồng thời, băng thông bộ nhớ ngoài được dành riêng cho việc nạp trọng số, hoặc trọng số được lưu trực tiếp trên SRAM nội bộ nhằm tối đa hóa hiệu quả năng lượng cho toàn hệ thống.

Chương 4

Phân tích và Kiến trúc hệ thống

Dựa trên cơ sở lý thuyết và công trình nghiên cứu bộ gia tốc đã trình bày, chương này đi sâu vào phân tích các yêu cầu kỹ thuật, từ đó đề xuất kiến trúc tổng thể của hệ thống SoC (System-on-Chip). Đồng thời, chương này cũng xác định đặc tả chức năng của từng khối thành phần và quy hoạch không gian địa chỉ bộ nhớ (Memory Map) cho toàn hệ thống.

4.1 Phân tích yêu cầu thiết kế

4.1.1 Yêu cầu chức năng

Để đảm bảo mục tiêu xây dựng một hệ thống SoC hoàn chỉnh có khả năng xử lý trí tuệ nhân tạo tại biên, thiết kế cần đáp ứng bốn nhóm yêu cầu chức năng cốt lõi liên quan đến thu thập dữ liệu, tính toán chuyên dụng, giao tiếp hệ thống và hiệu năng vận hành.

Thứ nhất, đối với phân hệ xử lý hình ảnh, hệ thống được yêu cầu phải có khả năng thu thập dữ liệu video liên tục từ Camera thông qua giao diện song song **DVP** (Digital Video Port). Luồng dữ liệu này cần được đồng bộ

hóa và chuyển đổi định dạng màu sắc để hiển thị trực tiếp lên màn hình qua chuẩn **HDMI** với độ phân giải tối thiểu là VGA (640x480) hoặc HD (1280x720). Yêu cầu quan trọng đặt ra là quá trình hiển thị phải diễn ra song song với quá trình xử lý, đảm bảo người dùng có thể quan sát hình ảnh thời gian thực với tốc độ khung hình ổn định từ 30 đến 60 fps.

Thứ hai, về năng lực tính toán, hệ thống phải tích hợp một bộ gia tốc phần cứng **CNN Accelerator** đóng vai trò là một thiết bị ngoại vi chuyên dụng (Memory-mapped Peripheral). Khối này chịu trách nhiệm thực thi các phép toán nhân chập (Convolution) và các hàm kích hoạt phi tuyến của mạng nơ-ron sâu. Accelerator cần có cơ chế truy cập trực tiếp vào bộ nhớ chứa dữ liệu ảnh đầu vào mà không làm gián đoạn luồng video đang hiển thị, đồng thời trả về kết quả phân lớp để vi xử lý tổng hợp.

Thứ ba, để đảm bảo tính tương thích và khả năng mở rộng như một vi điều khiển thương mại, SoC cần hỗ trợ đầy đủ các giao thức giao tiếp tiêu chuẩn công nghiệp. Cụ thể, giao thức **UART** được sử dụng cho giao diện dòng lệnh (CLI) và gỡ lỗi hệ thống; giao thức **I2C** đóng vai trò kênh điều khiển cấu hình cho các chip ngoại vi như Camera và HDMI PHY; và giao thức **SPI/OSPI** được tích hợp để giao tiếp với bộ nhớ Flash hoặc bộ nhớ RAM mở rộng (tốc độ từ 25MHz đến 100MHz), phục vụ cho việc lưu trữ trọng số mạng và chương trình cơ sở (Firmware).

Thứ tư, về chiến lược quản lý xung nhịp và hiệu năng, hệ thống được yêu cầu thiết kế theo kiến trúc đa miền tần số (Multi-Clock Domains) nhằm tối ưu hóa tài nguyên cho từng phân hệ cụ thể. Miền xung nhịp trung tâm (System Clock) điều khiển vi xử lý RISC-V và bộ gia tốc CNN được đặt mục tiêu hoạt động ở tần số **200 MHz**, đảm bảo thông lượng tính toán cao nhất cho các tác vụ AI. Đối với phân hệ Video Streaming, kiến trúc xung nhịp được phân chia thành ba tầng xử lý riêng biệt: mức **150 MHz** dành cho các khối xử lý dữ liệu video băng thông rộng và giao tiếp bộ nhớ; mức **50 MHz** và **25 MHz** phục vụ cho các giao diện hiển thị và đồng bộ

hóa tín hiệu Pixel Clock theo chuẩn VESA. Việc giao tiếp giữa miền 200 MHz của SoC và các miền tần số video thấp hơn phải được thực hiện thông qua các bộ đệm FIFO bất đồng bộ và cơ chế đồng bộ hóa Clock Domain Crossing(CDC) để triệt tiêu hiện tượng Metastability.

4.1.2 Yêu cầu phi chức năng

Bên cạnh các chức năng vận hành cơ bản, hệ thống phải tuân thủ các ràng buộc kỹ thuật nghiêm ngặt về hiệu năng thời gian thực, tần số hoạt động và quản lý tài nguyên trên nền tảng FPGA đích.

Thứ nhất, về hiệu năng xử lý, hệ thống phải đảm bảo tốc độ khung hình hiển thị ổn định ở mức **60 FPS** (khung hình/giây) tại độ phân giải mục tiêu. Độ trễ suy luận (Inference Latency) của mô hình AI phải được tối thiểu hóa để kết quả nhận dạng (như nhãn, khung bao) xuất hiện đồng bộ với vật thể đang chuyển động trên màn hình, triệt tiêu hiện tượng trễ pha (Lag) giữa hình ảnh thực tế và kết quả xử lý.

Thứ hai, về tần số hoạt động, thiết kế phải thỏa mãn các chỉ tiêu khắt khe của kiến trúc đa miền xung nhịp. Cụ thể, sau quá trình tổng hợp và hiện thực (Implementation), miền xung nhịp trung tâm (System Clock) cho vi xử lý và bộ gia tốc phải đạt tần số hoạt động ổn định **200 MHz** để tối đa hóa thông lượng tính toán. Các miền xung nhịp phụ trợ cho video (150 MHz, 50 MHz, 25 MHz) phải đảm bảo sự chính xác về định thời (Timing constraints) để duy trì sự ổn định của tín hiệu hiển thị và giao tiếp bộ nhớ.

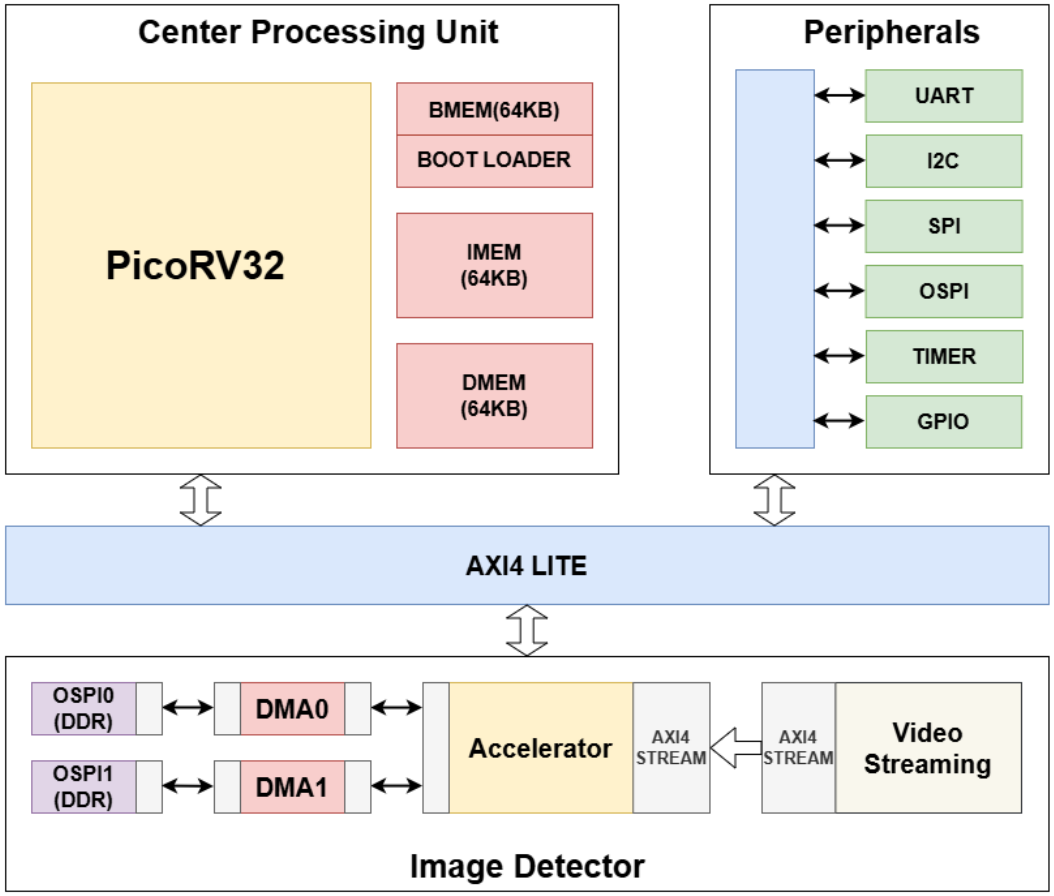
Thứ ba, về mặt tài nguyên, thiết kế được tối ưu hóa cho nền tảng bo mạch **Xilinx VC707** (sử dụng chip Virtex-7 XC7VX485T). Mặc dù đây là dòng FPGA hiệu năng cao với tài nguyên logic dồi dào, thách thức lớn nhất nằm ở việc quản lý hiệu quả băng thông bộ nhớ. Hệ thống phải điều phối chặt chẽ quyền truy cập giữa ba tác nhân tiêu thụ băng thông lớn: Vi xử lý (nạp lệnh/dữ liệu), Bộ gia tốc (đọc/ghi ma trận đặc trưng) và Bộ điều khiển hiển thị (quét bộ đệm khung hình). Việc tối ưu hóa này nhằm đảm

bảo luồng video 60 FPS luôn mượt mà ngay cả khi bộ gia tốc hoạt động ở mức tải cao nhất.

4.2 Kiến trúc tổng thể SoC

4.2.1 Tổng quan kiến trúc SoC

Để hiện thực hóa các yêu cầu phân tích nêu trên, đề tài đề xuất kiến trúc hệ thống **SoC không đồng nhất (Heterogeneous SoC)**, kết hợp giữa tính linh hoạt trong điều khiển của vi xử lý mềm (Soft-core Processor) và sức mạnh tính toán song song của phần cứng chuyên dụng.



Hình 4.1: Sơ đồ mô-đun kiến trúc tổng thể của hệ thống SoC RISC-V EdgeAI

Hệ thống được tổ chức thành ba phân hệ chính hoạt động phối hợp chặt

chẽ. Đầu tiên là **Center Processing Unit** với trung tâm là lõi vi xử lý PicoRV32. Phân hệ này đóng vai trò bộ não của hệ thống, chịu trách nhiệm khởi tạo, cấu hình các ngoại vi và quản lý giao tiếp người dùng.

Tiếp theo là **Image Detector**, bao gồm khối Accelerator được thiết kế tùy biến để thực thi các phép toán nhân chập (Convolution) nặng nề nhất trong mạng nơ-ron và khối Video Streaming để quản lý luồng video vào từ Camera và hiện thị ra HDMI.

Cuối cùng là **Peripherals**, tập hợp các mô-đun giao tiếp và lưu trữ thiết yếu để đảm bảo tính hoàn chỉnh của một hệ thống máy tính nhúng. Phân hệ này tích hợp các bộ điều khiển giao diện chuẩn công nghiệp như **UART** cho mục đích gỡ lỗi và **I2C** để cấu hình tham số phần cứng. Đối với giao tiếp lưu trữ, hệ thống áp dụng kiến trúc phân tầng. Trước hết, bộ điều khiển **SPI** được sử dụng để kết nối với các thiết bị lưu trữ thứ cấp phổ biến như thẻ nhớ SD Card, phục vụ việc lưu trữ dữ liệu ảnh mẫu, chương trình điều khiển(Firmware) hoặc logs hệ thống. Tuy nhiên, để đáp ứng nhu cầu truy xuất băng thông lớn cho trọng số mạng nơ-ron, thiết kế tích hợp thêm mô-đun giao tiếp bộ nhớ tốc độ cao **OSPI**. Mô-đun này hỗ trợ các chế độ truyền dẫn tiên tiến (Octal-SPI hỗ trợ **DDR**), giúp tăng tốc độ truy suất bộ nhớ bên ngoài hiệu quả, giải quyết bài toán giới hạn phải xài tài nguyên bộ nhớ nội bộ (BRAM) trên FPGA.

Để đáp ứng yêu cầu khắt khe về định thời trong các ứng dụng thời gian thực, mô-đun **Timer** được thiết kế với độ chính xác cao dựa trên xung nhịp hệ thống. Chức năng của mô-đun là tạo ra các khoảng trễ (Delay) chính xác cho các giao thức giao tiếp hoặc dùng đề làm Software Timer.

Cuối cùng, mô-đun **GPIO** cung cấp giao diện điều khiển linh hoạt ở cấp độ bit. Mặc dù có cấu trúc đơn giản, GPIO đóng vai trò không thể thiếu trong việc tương tác trực tiếp với người dùng thông qua hệ thống đèn LED báo trạng thái và nút nhấn điều khiển. Ngoài ra, các chân GPIO còn được quy hoạch để điều khiển các tín hiệu phần cứng quan trọng như tín hiệu

Reset cứng cho Camera hay tín hiệu kích hoạt cho màn hình, đảm bảo quy trình khởi động và vận hành của các phân hệ diễn ra theo đúng trình tự thiết kế.

4.2.2 Tổ chức hệ thống Bus phân tầng

Thách thức lớn nhất trong thiết kế này là giải quyết sự tranh chấp băng thông bộ nhớ giữa vi xử lý, bộ gia tốc AI và luồng video thời gian thực. Để khắc phục vấn đề này, kiến trúc Bus được thiết kế theo mô hình **Bus phân tầng (Hierarchical Bus Architecture)**.

Tầng thứ nhất là **Bus Ngoại vi (Peripheral Bus)**, sử dụng giao thức **AXI4-Lite** (biểu diễn bằng các đường kết nối màu xanh dương trong sơ đồ) để kết nối CPU với các ngoại vi bao gồm UART, I2C Master, SPI/OSPI Controller, Timer và GPIO. Các giao dịch trên tuyến bus này chủ yếu là các lệnh đọc/ghi thanh ghi cấu hình, do đó không yêu cầu băng thông lớn và vi xử lý đóng vai trò là Master (AXI4-Lite có hỗ trợ Multi Master và Multi Slave).

Tầng thứ hai là **Bus Dữ liệu Tốc độ cao (High-Performance Bus)**, được hiện thực chủ yếu dựa trên giao thức **AXI4-Stream** (biểu diễn bằng các đường kết nối màu xám trong sơ đồ). Đây là giao thức truyền dẫn dòng dữ liệu một chiều không cần địa chỉ, cho phép loại bỏ các chu kỳ trễ (Latency) phát sinh do quá trình bắt tay địa chỉ, từ đó tối đa hóa băng thông cho hệ thống. Tuyến bus này kết nối trực tiếp các thành phần tiêu thụ dữ liệu lớn thông qua cơ chế truy cập bộ nhớ trực tiếp (DMA): Bộ điều khiển Camera (Video DMA Write), Bộ điều khiển truy suất bộ nhớ (DRAM DMA) và Bộ gia tốc AI.

4.3 Đặc tả các khối chức năng chính

4.3.1 Vi xử lý trung tâm (Central Processing Unit)

Đóng vai trò là bộ não điều phối toàn bộ hoạt động của hệ thống SoC, phân hệ này được xây dựng xung quanh lõi vi xử lý mềm **PicoRV32** - một hiện thực tối ưu về tài nguyên của kiến trúc tập lệnh RISC-V chuẩn RV32I. Lõi vi xử lý này chịu trách nhiệm thực thi các tác vụ điều khiển logic chính, từ việc khởi tạo hệ thống đến quản lý các ngoại vi.
Để hỗ trợ hoạt động của CPU, kiến trúc bộ nhớ cục bộ được tổ chức thành ba vùng riêng biệt nhằm tối ưu hóa hiệu năng truy xuất:

Bộ nhớ Khởi động (BMEM - Bootloader Memory): Đây là thành phần quan trọng chứa các tập lệnh khởi động cơ bản (Boot ROM). Ngay khi hệ thống được cấp nguồn hoặc reset, CPU sẽ trỏ thanh ghi bộ đếm chương trình (PC) vào vùng nhớ này đầu tiên. Nhiệm vụ của Bootloader là thiết lập các thông số phần cứng ban đầu và nạp chương trình chính từ bộ nhớ ngoài (Flash SPI) vào IMEM trước khi trao quyền điều khiển lại cho ứng dụng.

Bộ nhớ Lệnh (IMEM - Instruction Memory): Là vùng nhớ chứa mã chương trình chính (Main Application) mà CPU sẽ thực thi sau quá trình khởi động. Vùng nhớ này thường được ánh xạ vào Block RAM để đảm bảo tốc độ truy xuất lệnh nhanh nhất (một chu kỳ máy).

Bộ nhớ Dữ liệu (DMEM - Data Memory): Dùng để lưu trữ các biến toàn cục, ngăn xếp (Stack) và dữ liệu tạm thời trong quá trình tính toán của chương trình. Việc tách biệt DMEM và IMEM (kiến trúc Harvard sửa đổi) giúp tránh xung đột khi CPU thực hiện nạp lệnh và truy xuất dữ liệu đồng thời.

4.3.2 Nhận diện Hình ảnh (Image Detector)

Đây là phân hệ cốt lõi tạo nên tính năng thông minh của hệ thống, chịu trách nhiệm thực hiện song song hai tác vụ: duy trì luồng hình ảnh thời gian thực và thực thi các thuật toán trí tuệ nhân tạo. Cấu trúc của phân hệ này là sự tích hợp chặt chẽ giữa chuỗi xử lý video (Video Streaming Pipeline) và khối tính toán chuyên dụng.

4.3.2.1 Hệ thống Video Streaming

Khối này quản lý dòng chảy dữ liệu hình ảnh liên tục để phục vụ nhu cầu quan sát. Tại ngõ vào, giao diện thu thập dữ liệu tiếp nhận tín hiệu từ Camera qua chuẩn song song DVP, thực hiện đồng bộ và đóng gói dữ liệu vào bộ đệm khung hình (Frame Buffer). Tại ngõ ra, bộ điều khiển hiển thị đọc dữ liệu từ bộ đệm này và chuyển đổi thành tín hiệu chuẩn HDMI, đảm bảo xuất hình ảnh mượt mà lên màn hình với độ trễ tối thiểu.

4.3.2.2 Khối Gia tốc (Accelerator)

Đóng vai trò là trung tâm xử lý AI, khối gia tốc được thiết kế dựa trên kiến trúc Mảng tâm thu (Systolic Array) kích thước $N \times N$, chuyên trách xử lý các phép toán nhân chập (Convolution) nặng nề của mạng nơ-ron.
Để đảm bảo khả năng cung cấp dữ liệu liên tục cho mảng tính toán mà không làm nghẽn bus hệ thống, khối gia tốc được tích hợp cơ chế truy xuất bộ nhớ băng thông rộng thông qua hai kênh DMA chuyên biệt:

DMA 0 (Data/Weight Reader): Kênh này chịu trách nhiệm đọc dữ liệu đặc trưng đầu vào (Input Feature Maps) và các bộ trọng số (Weights) từ bộ nhớ hệ thống (Frame Buffer hoặc Weight Memory) để nạp vào bộ đệm nội của Accelerator.

DMA 1 (Result Writer): Kênh này chịu trách nhiệm thu thập kết quả tính toán (Output Feature Maps) từ mảng Systolic và ghi ngược

trở lại bộ nhớ chính, sẵn sàng cho các lớp xử lý tiếp theo hoặc để vi xử lý trung tâm đọc kết quả phân lớp.

4.3.3 Các Ngoại vi (Peripherals)

Phân hệ Ngoại vi tích hợp các khối chức năng chuẩn hóa, cung cấp các giao thức giao tiếp phổ biến để đảm bảo khả năng tương thích và mở rộng cho hệ thống nhúng:

UART: Cung cấp giao thức truyền thông nối tiếp không đồng bộ (Asynchronous Serial Communication), phục vụ việc trao đổi dữ liệu dòng và hỗ trợ giao diện gỡ lỗi hệ thống.

I2C: Cung cấp giao thức giao tiếp nối tiếp hai dây (Two-wire Interface), đóng vai trò Master điều khiển và cấu hình các thiết bị ngoại vi tham gia vào bus hệ thống.

SPI/OSPI: Cung cấp giao thức truyền thông nối tiếp đồng bộ tốc độ cao (Serial Peripheral Interface), hỗ trợ mở rộng kết nối với các bộ nhớ ngoài hoặc các thiết bị ngoại vi yêu cầu băng thông truyền tải lớn.

Timer & GPIO: Cung cấp tài nguyên định thời gian thực cho hệ thống và các giao diện điều khiển tín hiệu số vào/ra đa mục đích (General Purpose Input/Output).

4.4 Tổ chức bộ nhớ và Bản đồ địa chỉ (Memory Map)

4.4.1 Khái niệm và vai trò của Memory Map

Bản đồ bộ nhớ (Memory Map) là một cấu trúc dữ liệu mô hình hóa cách thức hệ thống phân bổ các địa chỉ số (thường dưới dạng hệ thập lục

phân - Hexadecimal) vào các tài nguyên phần cứng vật lý trong hệ thống SoC. Trong kiến trúc xử lý, vi xử lý PicoRV32 không tương tác trực tiếp với các thiết bị ngoại vi bằng tên gọi, mà thông qua một không gian địa chỉ phẳng duy nhất.

Việc quy hoạch bản đồ bộ nhớ là bước thiết kế tiên quyết vì những lý do sau:

Thống nhất giao tiếp (Memory-mapped I/O): Cho phép CPU coi các thanh ghi điều khiển của ngoại vi (như UART, I2C) tương tự như các ô nhớ thông thường. Điều này giúp đơn giản hóa tập lệnh của vi xử lý vì chỉ cần các lệnh nạp/lưu dữ liệu (*Load*/*Store*) để điều khiển toàn bộ phần cứng.

Định tuyến dữ liệu (Address Decoding): Cung cấp thông tin cho bộ giải mã địa chỉ (Address Decoder) trong khối **AXI Interconnect**. Dựa trên địa chỉ mà CPU phát ra, hệ thống sẽ biết chính xác cần kích hoạt tín hiệu chọn thiết bị (*ChipSelect*) nào để dẫn luồng dữ liệu đến đúng đích.

Tránh xung đột tài nguyên: Đảm bảo mỗi thành phần phần cứng được cấp phát một vùng không gian riêng biệt, không chồng lấn, từ đó triệt tiêu các lỗi xung đột địa chỉ khi hệ thống vận hành.

Cơ sở cho phát triển phần mềm (Firmware): Bản đồ bộ nhớ cung cấp các địa chỉ cơ sở (*BaseAddress*) giúp người lập trình xây dựng các trình điều khiển thiết bị (Drivers) và cấu hình trình biên dịch (Linker Script) để nạp mã nguồn vào đúng vị trí trong bộ nhớ.

Dựa trên kiến trúc SoC đề xuất, không gian địa chỉ được chia thành hai phân vùng lớn: Vùng nhớ hệ thống (System Memory) và Vùng địa chỉ ngoại vi (Peripherals).

4.4.2 Bản đồ vùng nhớ hệ thống

Vùng nhớ hệ thống bao gồm các khối BRAM chứa mã thực thi và dữ liệu hoạt động của vi xử lý. Chi tiết phân bổ được trình bày trong Bảng 4.1.

Bảng 4.1: Bản đồ địa chỉ vùng nhớ hệ thống (System Memory Map)

Thành phần	**Dải địa chỉ (Hex)**	**Mô tả Chức năng**
DMEM	`0x0000_0000` `0x0001_0000`	**Data Memory (64KB)**. Vùng nhớ dữ liệu, Stack, Heap
BMEM	`0x0100_0000` `0x0101_0000`	**Boot Memory (64KB)**. Chứa mã khởi động (Bootloader).
IMEM	`0x0110_0000` `0x0111_0000`	**Instruction Memory (64KB)**. Vùng nhớ chứa mã lệnh chương trình chính (Firmware).

4.4.3 Bản đồ vùng ngoại vi

Vùng ngoại vi bắt đầu từ địa chỉ cơ sở `0x8000_0000`. Mỗi ngoại vi được cấp phát một không gian 4KB (Offset `0x1000`) để chứa các thanh ghi cấu hình. Chi tiết được trình bày trong Bảng 4.2.

Bảng 4.2: Bản đồ địa chỉ vùng ngoại vi (Peripheral Memory Map)

Thành phần	**Dải địa chỉ (Hex)**	**Mô tả Chức năng**
GPIO	`0x8000_0000` `0x8000_0FFF`	Điều khiển các tín hiệu vào/ra cơ bản (LEDs, Buttons).
UART	`0x8000_1000` `0x8000_1FFF`	Bộ điều khiển giao tiếp nối tiếp (Console/Debug).
I2C	`0x8000_2000` `0x8000_2FFF`	Giao tiếp cấu hình Camera và chip HDMI PHY.
SPI	`0x8000_3000` `0x8000_3FFF`	Giao tiếp thẻ nhớ SD Card hoặc Flash phụ trợ.
OSPI	`0x8000_4000` `0x8000_4FFF`	Giao tiếp bộ nhớ tốc độ cao (Octal-SPI/DDR).
Timer	`0x8000_5000` `0x8000_5FFF`	Bộ định thời gian thực và đo đạc hiệu năng.

Cơ chế giải mã địa chỉ được thực hiện bởi bộ **AXI Interconnect**, đảm bảo tín hiệu chọn thiết bị tớ (Slave Select) được gửi chính xác đến từng khối chức năng dựa trên địa chỉ mà CPU phát ra trên bus hệ thống.

Chương 5

Thiết kế Bộ tăng tốc AI (AI Accelerator)

Chương này trình bày chi tiết quy trình thiết kế lõi IP AI Accelerator, bắt đầu từ phân tích cơ sở toán học, đề xuất chiến lược tối ưu dòng dữ liệu (Dataflow) đến hiện thực hóa kiến trúc vi mô (Micro-architecture).

5.1 Cơ sở Toán học và Thách thức Thiết kế

Để xây dựng một kiến trúc phần cứng thống nhất (Unified Architecture) có khả năng xử lý linh hoạt các mô hình mạng nơ-ron đa dạng—từ các mạng kinh điển (như VGG16) đến các mạng tối ưu cho thiết bị biên (như MobileNet)—chúng tôi tập trung phân tích đặc tả toán học của hai phép tính cốt lõi: **Standard Convolution** và **Depthwise Separable Convolution**.

Mục tiêu là tìm ra điểm chung trong cấu trúc tính toán và cơ chế xử lý biên (Padding) để tối ưu hóa phần cứng.

5.1.1 Standard Convolution (Tích chập tiêu chuẩn)

Đây là phép tính nền tảng trong CNN truyền thống. Đặc trưng của nó là sự liên kết dày đặc: mỗi điểm ảnh đầu ra là kết quả của việc tổng hợp thông tin từ toàn bộ không gian không gian đầu vào và toàn bộ chiều sâu của kênh (Channels).

5.1.1.1 Mô hình toán học

Xét lớp tích chập với đầu vào I ($C \times H_{in} \times W_{in}$) và bộ trọng số W ($M \times C \times R \times S$). Giá trị đầu ra O tại kênh m, vị trí (h, w) được tính như sau:

$$O[m][h][w] = B[m] + \sum_{c=0}^{C-1} \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} I[c][h \cdot U + r - P][w \cdot U + s - P] \times W[m][c][r][s] \tag{5.1}$$

Trong đó: U là bước trượt (Stride), P là đệm (Padding).

5.1.1.2 Thuật toán xử lý

Để hiện thực hóa trên phần cứng, phép tính được mô hình hóa thành 6 vòng lặp lồng nhau. Việc xử lý Padding được tích hợp trực tiếp vào logic điều khiển: nếu chỉ số truy cập nằm ngoài biên ảnh, giá trị trả về là 0 (Zero-padding).

Algorithm 1: Standard Convolution (Standard Conv2D)

Input: $I[C][H_{in}][W_{in}]$, $W[M][C][R][S]$, Padding P, Stride U

Output: $O[M][H_{out}][W_{out}]$

for $m = 0$ ***to*** $M - 1$ **do**
  for $c = 0$ ***to*** $C - 1$ **do**
    for $h = 0$ ***to*** $H_{out} - 1$ **do**
      for $w = 0$ ***to*** $W_{out} - 1$ **do**
        for $r = 0$ ***to*** $R - 1$ **do**
          for $s = 0$ ***to*** $S - 1$ **do**
            $h_{in} = h \cdot U + r - P$
            $w_{in} = w \cdot U + s - P$
            if $h_{in} \geq 0 \land h_{in} < H_{in} \land w_{in} \geq 0 \land w_{in} < W_{in}$ **then**
              $val = I[c][h_{in}][w_{in}]$
            else
              $val = 0$ `/* Zero Padding */`
            end
            $O[m][h][w] \leftarrow O[m][h][w] + val \times W[m][c][r][s]$
          end
        end
      end
    end
  end
end

5.1.2 Depthwise Separable Convolution

Nhằm giảm tải khối lượng tính toán cho thiết bị biên, kỹ thuật này tách phép chập chuẩn thành hai bước độc lập:

5.1.2.1 Depthwise Convolution (DW)

Phép tính này áp dụng bộ lọc riêng biệt cho từng kênh đầu vào, không có sự cộng gộp giữa các kênh.

$$O_{dw}[c][h][w] = \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} I[c][h \cdot U + r - P][w \cdot U + s - P] \times W_{dw}[c][r][s] \quad (5.2)$$

Algorithm 2: Depthwise Convolution (với Padding)

for $c = 0$ ***to*** $C - 1$ `/* Parallel Channels */`
do
  for $h = 0$ ***to*** $H_{out} - 1$ **do**
    for $w = 0$ ***to*** $W_{out} - 1$ **do**
      for $r = 0$ ***to*** $R - 1$ **do**
        for $s = 0$ ***to*** $S - 1$ **do**
          $h_{in} = h \cdot U + r - P$
          $w_{in} = w \cdot U + s - P$
          if $h_{in} \in [0, H_{in}) \wedge w_{in} \in [0, W_{in})$ **then**
            $O_{dw}[c][h][w] += I[c][h_{in}][w_{in}] \times W_{dw}[c][r][s]$
          end
        end
      end
    end
  end
end

5.1.2.2 Pointwise Convolution (PW)

Thực chất là phép chập chuẩn với kernel 1×1. Nó chịu trách nhiệm trộn thông tin giữa các kênh sau khi lớp Depthwise đã xử lý không gian.

$$O_{pw}[m][h][w] = \sum_{c=0}^{C-1} I[c][h][w] \times W_{pw}[m][c] \tag{5.3}$$

5.2 Chiến lược Phân mảnh và Quản lý Dòng dữ liệu

Do tài nguyên bộ nhớ on-chip (BRAM) trên FPGA là hữu hạn, không thể nạp toàn bộ Feature Map của các mạng lớn vào cùng lúc. Chúng tôi áp dụng chiến lược **Phân mảnh dữ liệu (Tiling)** kết hợp với cơ chế quản lý bộ nhớ **Ping-Pong** để xử lý vấn đề này. Bên cạnh đó, vì kích thước ifmap ở các layer rất đa dạng nên việc cố định kích thước input tile sẽ gây lãng phí tài nguyên tính toán. Vì vậy chúng tôi đề xuất một phương pháp phân mảnh linh hoạt theo chiều dọc của ảnh đầu vào, giúp tận dụng tối đa tài nguyên phần cứng.

5.2.1 Chiến lược Phân mảnh không gian (Space Partitioning)

Chúng tôi định nghĩa một "Tile" (Mảnh dữ liệu) là đơn vị dữ liệu cơ sở được nạp và xử lý trong một lần. Không gian tính toán được chia nhỏ theo 3 chiều:

1. **Chiều dọc (H):** Chia ảnh đầu vào thành $N_h = \lceil H/T_h \rceil$ phần.
2. **Chiều sâu kênh (C):** Chia số kênh đầu vào thành $N_c = \lceil C/T_c \rceil$ nhóm.
3. **Số bộ lọc (M):** Chia số bộ lọc đầu ra thành $N_m = \lceil M/T_m \rceil$ nhóm.

Một chu trình xử lý trọn vẹn một cặp (Input Tile, Weight Tile) để cập nhật giá trị cho Output Tile được gọi là một **Pass**.

5.2.2 Mô hình hóa và Tham số thiết kế

Các ký hiệu và tham số thiết kế cho bài toán phân mảnh được tóm tắt trong Bảng 5.1.

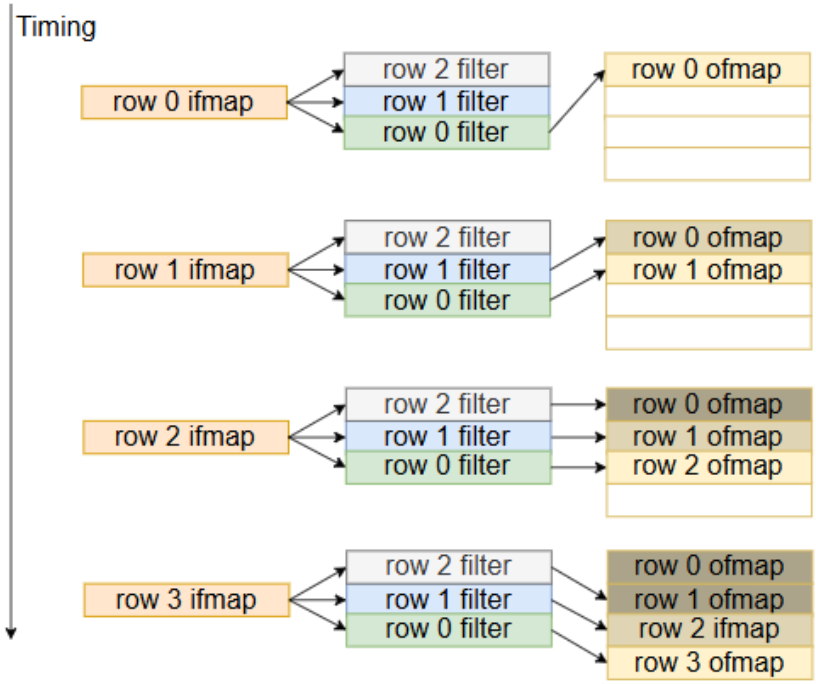
Bảng 5.1: Bảng tham số thiết kế và ánh xạ ký hiệu

Nhóm tham số	**Ký hiệu**	**Mô tả**
Filter	S, R	Độ rộng (w_f) và Độ dài (h_f) của bộ lọc
	N_f	Tổng số bộ lọc (Filters)
Feature Map	W, H, C	Kích thước Input Feature Map (Rộng, Dài, Số kênh)
	W_{out}, H_{out}, N_f	Kích thước Output Feature Map
Tiling (Pass)	T_h	Chiều cao IFM nạp trong 1 pass (h)
	T_c	Số kênh IFM tính toán song song (k)
	T_m	Số bộ lọc tính toán song song (m)
Output Tile	T_{ho}	Chiều cao OFM hợp lệ tạo ra trong 1 pass (h_o)
Khác	P, Str	Padding và Stride

5.2.3 Bài toán Dữ liệu biên và Cơ chế Ping-Pong

Thách thức lớn nhất của việc chia nhỏ ảnh theo chiều dọc là xử lý biên giữa các Tile. Khi bộ lọc trượt đến hàng cuối cùng của Tile hiện tại (H_k), nó cần dữ liệu của các hàng đầu tiên thuộc Tile tiếp theo (H_{k+1}) để hoàn thành phép tính.

5.2.3.1 Phân tích Dữ liệu dôi ra (Residual Data)



Hình 5.1: Minh họa sự hình thành dữ liệu dôi ra. Tại hàng 2 và 3, bộ lọc thiếu dữ liệu từ hàng 4, 5 (thuộc tile sau) nên kết quả chưa hoàn thiện.

Như hình minh họa, các kết quả tính toán tại biên dưới (nơi thiếu dữ liệu lân cận) được gọi là **Dữ liệu dôi ra (Residual Data)**. Thay vì loại bỏ hoặc tính lại từ đầu, hệ thống lưu giữ các giá trị bán hoàn chỉnh này và cộng dồn với kết quả từ Pass tiếp theo.

Số lượng hàng đầu ra hợp lệ (T_{ho}) trong mỗi Pass tuân theo quy tắc:

$$T_{ho} = \begin{cases} T_h - R + 1 & \text{với Tile đầu tiên (chưa có residual)} \\ T_h & \text{với các Tile sau (nhờ cộng gộp residual)} \end{cases} \tag{5.4}$$

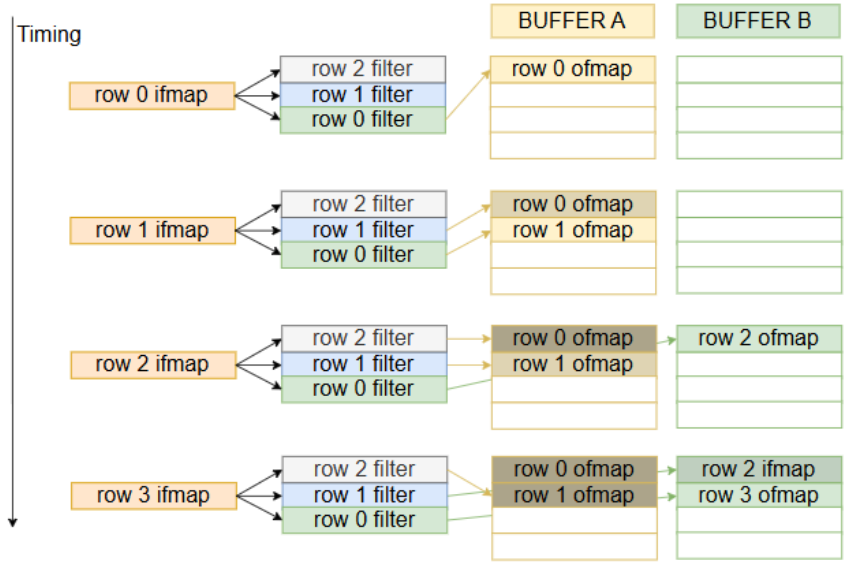
5.2.3.2 Logic Hoạt động Ping-Pong

Hệ thống sử dụng hai bộ đệm đầu ra ($Buffer_A, Buffer_B$) luân phiên vai trò để xử lý vấn đề này:

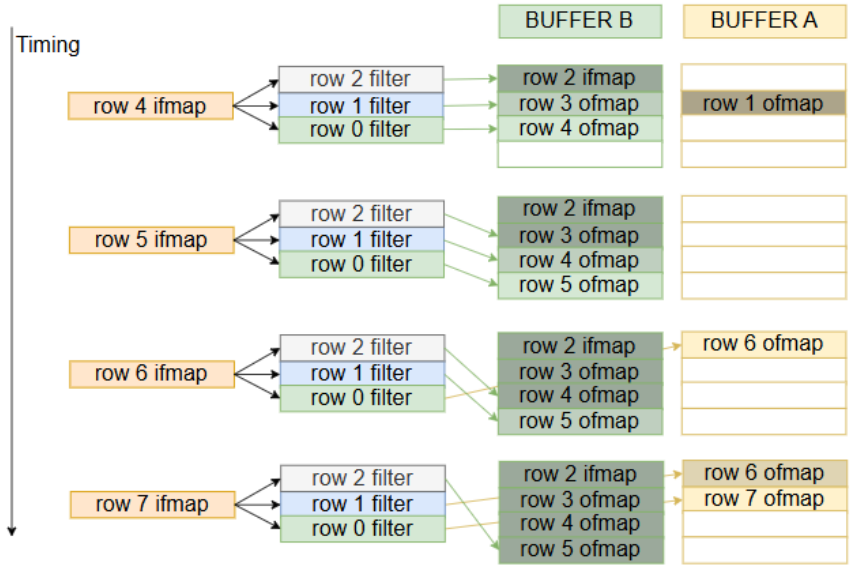
1. **Pass k:** Ghi kết quả hợp lệ vào Buffer hiện tại. Các hàng dôi ra được

ghi vào Buffer kế tiếp.

2. **Pass $k+1$:** Buffer kế tiếp (chứa dữ liệu dôi ra cũ) trở thành Buffer hiện tại. Phép tính mới cộng dồn vào đó, hoàn thiện các hàng dôi ra thành hợp lệ.



(a) Giai đoạn 1: Tích lũy Valid vào A, lưu Residual vào B.



(b) Giai đoạn 2: B hoàn thiện kết quả từ Residual cũ, A lưu Residual mới.

Hình 5.2: Cơ chế Ping-Pong Buffer luân phiên để quản lý vùng dữ liệu biên liên tục.

5.2.4 Thuật toán Điều phối Pass (Pass Scheduling)

Trình tự thực thi các Pass (Scheduling) đóng vai trò quyết định đến hiệu năng và tính đúng đắn của dòng dữ liệu. Chúng tôi đề xuất hai thuật toán riêng biệt cho Standard Conv và Depthwise Conv.

5.2.4.1 Trường hợp Standard Convolution

Do đặc tính cộng gộp kênh, thuật toán cần ưu tiên vòng lặp tích lũy (Reduction Loop) theo chiều C trước khi chuyển sang xử lý không gian H.

Algorithm 3: Lịch trình Pass cho Standard Convolution

Input: N_m (Groups), N_h (Height Blocks)

Output: DRAM (Valid OFM)

Initialize pointers: $Buf_{curr} \leftarrow A$, $Buf_{next} \leftarrow B$

for $m = 0$ ***to*** $N_m - 1$ **do**

  1. Load Weights (Weight Stationary)

  for $h = 0$ ***to*** $N_h - 1$ **do**

    for $c = 0$ ***to*** $N_c - 1$ **do**

      Pass (m, h, c)**:** Tính toán và tích lũy Partial Sum vào Buffer

    end

    2. Xử lý biên & Ghi Output:

    - Kiểm tra Buffer, tách phần Valid và Residual.

    - Ghi phần Valid xuống DRAM.

    - Hoán đổi Ping-Pong Buffer.

  end

end

5.2.4.2 Trường hợp Depthwise Convolution

Do tính độc lập giữa các kênh, thuật toán loại bỏ vòng lặp tích lũy, giúp đơn giản hóa luồng dữ liệu.

Algorithm 4: Lịch trình Pass cho Depthwise Convolution

Input: N_m (Groups), N_h (Height Blocks)

Output: DRAM (Valid OFM)

Initialize pointers: $Buf_{curr} \leftarrow A$, $Buf_{next} \leftarrow B$

for $m = 0$ ***to*** $N_m - 1$ **do**

  1. Load Weights

  for $h = 0$ ***to*** $N_h - 1$ **do**

    Pass (m, h)**:** Tính toán Depthwise (1-to-1)

    if $h == 0$ **then**

      // Tile đầu: Lưu Residual vào Buf_next

      - Ghi Valid ($T_h - R + 1$ hàng) xuống DRAM

    else

      // Tile sau: Hoàn thiện Residual cũ trong Buf_curr

      - Ghi toàn bộ Valid (T_h hàng) xuống DRAM

    end

    3. Chuẩn bị tiếp theo:

    - Clear Buf_{curr}, Swap pointers: $Buf_{curr} \leftrightarrow Buf_{next}$

  end

end

Pass 0 --- row 0-10, channel 0-10, filter 0	Pass 1 --- row 0-10, channel 11-20, filter 0	Pass 2 --- row 11-20, channel 0-10, filter 0	Pass 3 --- row 11-20, channel 11-20, filter 0
Pass 4 --- row 0-10, channel 0-10, filter 1	Pass 5 --- row 0-10, channel 11-20, filter 1	Pass 6 --- row 11-20, channel 0-10, filter 1	Pass 7 --- row 11-20, channel 11-20, filter 1

(a) Standard Convolution ($H = 21, M = 2$)

Pass 0 --- row 0-10, channel 0-10, filter 0-10	Pass 1 --- row 11-20, channel 0-10, filter 0-10	Pass 2 --- row 0-10, channel 11-20, filter 11-20	Pass 3 --- row 11-20, channel 11-20, filter 11-20

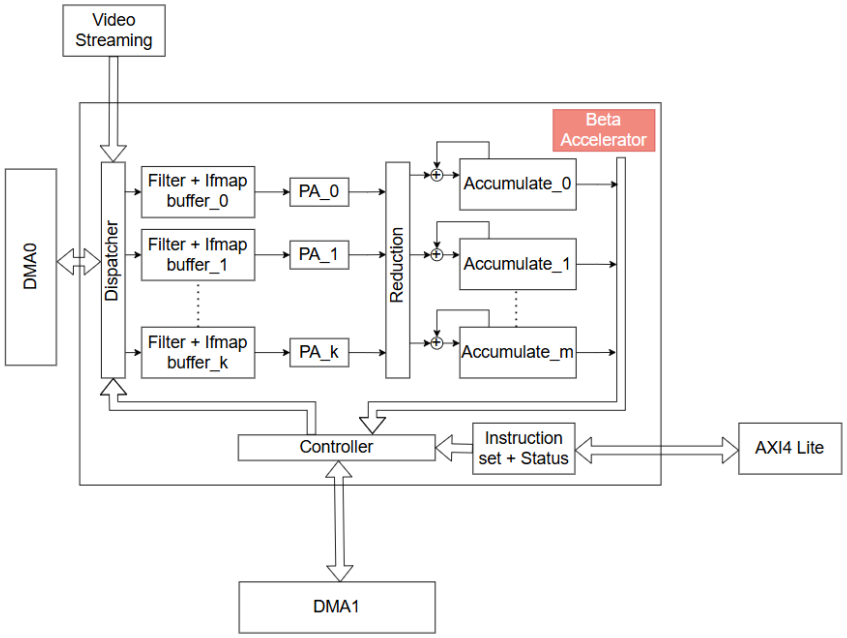
(b) Depthwise Convolution ($H = 21, M = 21$)

Hình 5.3: So sánh chiến lược phân chia Pass: Standard Conv cần tích lũy theo chiều sâu (hình a), trong khi Depthwise Conv xử lý song song độc lập (hình b).

5.3 Thiết kế Kiến trúc Vi mô (Micro-architecture)

Dựa trên các phân tích dòng dữ liệu, chúng tôi đề xuất kiến trúc phần cứng **Beta Accelerator**. Điểm nhấn của kiến trúc là việc tách biệt hoàn toàn đường dẫn dữ liệu (Data Path) và trọng số (Weight Path) để tối đa hóa băng thông.

5.3.1 Sơ đồ khối tổng quát



Hình 5.4: Kiến trúc Beta Accelerator với Bus dữ liệu và Trọng số tách biệt.

Hệ thống bao gồm các thành phần chính:

- **Controller:** Điều phối hoạt động toàn hệ thống. Quản lý hai giao tiếp bộ nhớ độc lập: *Weight Memory Interface* và *Activation Memory Interface*.
- **Dispatcher:** Phân phối dữ liệu từ Bus vào các bộ đệm cục bộ.
- **Ping-Pong Buffers:** Hệ thống bộ nhớ đệm kép cho cả IFM và

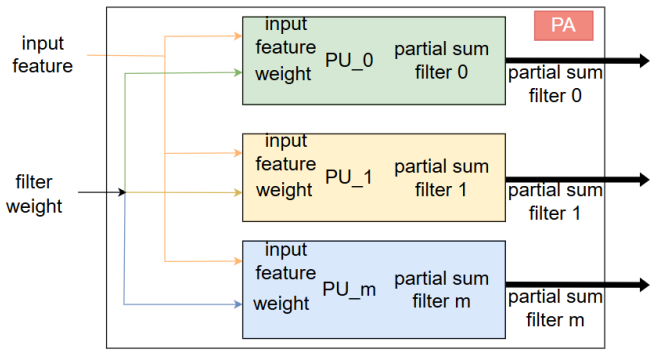
Weight, cho phép nạp dữ liệu Pass $k+1$ song song với việc tính toán Pass k.

- **Process Array (PA):** Mảng tính toán song song, thực hiện phép nhân chập.
- **Reduction Unit & Accumulator:** Thực hiện cộng dồn kết quả từ các kênh (đối với Standard Conv) và quản lý việc ghi kết quả xuống DRAM.

5.3.2 Tổ chức Phân cấp Đơn vị Tính toán

Kiến trúc tính toán được thiết kế theo mô hình phân cấp 3 tầng: PA $\rightarrow$ PU $\rightarrow$ PE.

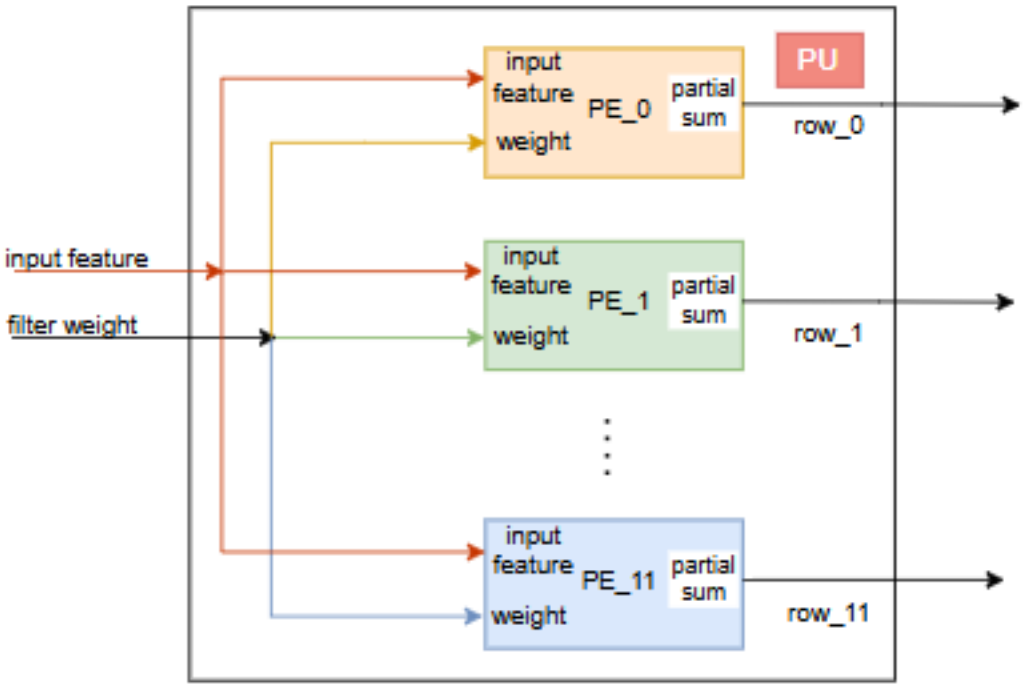
5.3.2.1 Mảng xử lý (Process Array - PA)



Hình 5.5: Mỗi PA xử lý 1 kênh Input và tạo ra kết quả cho T_m kênh Output.

Khối PA tận dụng tính song song mức bộ lọc (Filter Parallelism). Dữ liệu đầu vào (IFM) được Broadcast tới tất cả các đơn vị bên trong, trong khi trọng số được phân phối riêng biệt.

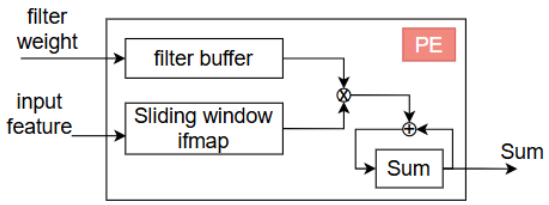
5.3.2.2 Đơn vị xử lý (Process Unit - PU)



Hình 5.6: Khối PU chứa 11 PE trong trường hợp chạy model AlexNet.

Mỗi PU chịu trách nhiệm cho một bộ lọc. Khối PU chứa số PE song song bằng với độ cao của filter weight, từ đó giúp xử lý được bộ lọc có kích thước lớn nhất. Như trong model AlexNet kích thước filter lớn nhất là 11×11 nên số PE trong PU là 11. Mỗi PE xử lý một hàng của kernel.

5.3.2.3 Phần tử xử lý (Process Element - PE)



Hình 5.7: PE thực hiện phép MAC với cơ chế Weight Stationary.

PE là đơn vị nhỏ nhất thực hiện phép nhân cộng (MAC). Nó sử dụng thanh ghi trượt (Sliding Window Register) để di chuyển dữ liệu IFM qua bộ lọc cố định.

5.3.3 Đánh giá thời gian thực thi (Performance Estimation)

Thời gian thực thi của hệ thống phụ thuộc vào loại lớp tích chập (Standard hay Depthwise) do sự khác biệt trong chiến lược luồng dữ liệu.

5.3.3.1 Thời gian xử lý một Pass cơ sở (T_{pass})

Dựa trên kiến trúc Pipeline của các Process Element (PE), thời gian để hoàn thành tính toán cho một tile có chiều cao T_h và độ rộng OFM W_{out} được xác định bởi:

$$T_{pass} = \Big[(W_{out} - 1) \times (S + U - 1) + S\Big] \times T_h \tag{5.5}$$

Trong đó:

- S: Kích thước bộ lọc (Filter width).
- U: Bước trượt (Stride).
- W_{out}: Chiều rộng của OFM.
- $(S + U - 1)$: Số chu kỳ trung bình để tính một điểm ảnh tiếp theo nhờ tối ưu hóa Pipeline (khi $U = 1$, thời gian này là S).

5.3.3.2 Tổng thời gian thực thi (T_{total})

Trường hợp 1: Standard Convolution

Với tích chập tiêu chuẩn, mỗi điểm ảnh đầu ra là tổng hợp của tất cả C kênh đầu vào. Hệ thống phải thực hiện vòng lặp tích lũy qua các khối kênh T_c.

$$T_{total\_std} = \underbrace{\left\lceil \frac{N_f}{T_m} \right\rceil}_{\text{Output Blocks}} \times \underbrace{\left\lceil \frac{C}{T_c} \right\rceil}_{\text{Input Blocks}} \times \underbrace{\left\lceil \frac{H}{T_h} \right\rceil}_{\text{Height Blocks}} \times T_{pass} \quad (5.6)$$

Trường hợp 2: Depthwise Convolution

Với tích chập chiều sâu, các kênh hoạt động độc lập ($N_f = C$). Hệ thống không cần thực hiện vòng lặp tích lũy kênh đầu vào ($\lceil C/T_c \rceil$ bị loại bỏ). Các nhóm kênh được xử lý song song dựa trên khả năng của phần cứng (T_m).

$$T_{total\_dw} = \underbrace{\left\lceil \frac{N_f}{T_m} \right\rceil}_{\text{Channel Groups}} \times \underbrace{\left\lceil \frac{H}{T_h} \right\rceil}_{\text{Height Blocks}} \times T_{pass} \quad (5.7)$$

Nhận xét: So với Standard Convolution, Depthwise Convolution giảm được hệ số $\lceil C/T_c \rceil$ lần số lượng tính toán, giúp tăng tốc độ xử lý đáng kể đối với các mạng nhẹ (Lightweight CNNs) như MobileNet.

5.3.4 Mô hình hóa độ trễ toàn hệ thống

Để xác định cấu hình phần cứng tối ưu cho từng lớp mạng, chúng tôi xây dựng mô hình ước lượng thời gian thực thi. Mô hình này thực hiện quét qua không gian các tham số chia khối (Tiling parameters) gồm (T_c, T_m, T_h) để tìm ra bộ tham số giúp cực tiểu hóa tổng số chu kỳ hoạt động (Total Cycles).

5.3.4.1 Cơ chế hoạt động

Trước khi bắt đầu tính toán Pass đầu tiên, hệ thống cần nạp đầy đủ dữ liệu (IFM, Weights) vào buffer. Sau giai đoạn khởi tạo này, quy trình hoạt động theo nguyên lý "gối đầu":

- Trong khi lõi tính toán đang xử lý Pass i, bộ điều khiển DMA đồng thời nạp dữ liệu cho Pass $i+1$ vào nửa còn lại của Buffer.

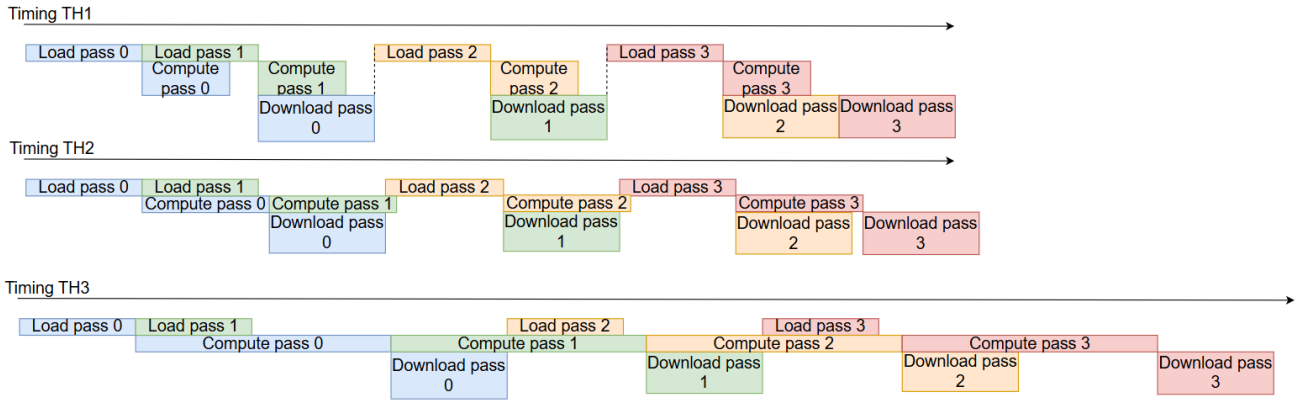
- Đồng thời, kết quả của Pass $i-1$ (nếu đã hoàn tất) được ghi trả về bộ nhớ ngoài.

Do hệ thống sử dụng bus dữ liệu dùng chung (Shared Data Bus) cho cả luồng nạp (Load) và ghi (Store), băng thông bộ nhớ phải được chia sẻ thời gian. Bộ điều khiển sẽ ưu tiên nạp Pass tiếp theo, sau đó mới đến ghi Pass trước đó (hoặc xen kẽ tùy theo chính sách trọng tài).

5.3.4.2 Các kịch bản hiệu năng (Performance Scenarios)

Gọi T_{load} là thời gian nạp 1 Input Pass, T_{store} là thời gian ghi 1 Output Pass, và T_{comp} là thời gian tính toán 1 Pass (cũng chính là T_{pass} đã tính ở mục 4.3.3). Ta định nghĩa tham số b là **số chu kỳ đồng hồ cần thiết để truyền 1 giá trị dữ liệu** (Cycles per Data Transfer).

Mô hình thời gian hoàn thành 1 layer được phân tích dựa trên sự chênh lệch giữa năng lực tính toán và băng thông bộ nhớ, được minh họa trong Hình 5.8.



Hình 5.8: Biểu đồ thời gian thực thi trong 3 trường hợp: (Trên cùng) Memory Bound 1, (Giữa) Memory Bound 2, (Dưới cùng) Compute Bound.

Trường hợp 1: Memory Bound 1 (Nghẽn băng thông nghiêm trọng)

Xảy ra khi thời gian nạp dữ liệu lớn hơn thời gian tính toán ($T_{load} \geq T_{comp}$). Lõi tính toán phải chờ dữ liệu nạp xong mới có thể chạy. Tổng thời gian

hoàn thành layer được quyết định chủ yếu bởi tổng lượng dữ liệu cần truyền tải (Input + Output).

- **Đối với Standard Convolution:** Do phải nạp lại Input Feature Map cho mỗi nhóm Filter khác nhau (nếu không đủ bộ nhớ on-chip), tổng thời gian là:

$$T_{total} \approx \left[\left(H \times W \times C \times \left\lceil \frac{N_f}{T_m} \right\rceil \right) + \left(H_{out} \times W_{out} \times N_f \right) \right] \times b \quad (5.8)$$

- **Đối với Depthwise Convolution:** Mỗi kênh Input chỉ tương tác với 1 kênh Filter tương ứng ($N_f = C$), nên Input Feature Map chỉ cần nạp 1 lần duy nhất:

$$T_{total} \approx \left[(H \times W \times C) + (H_{out} \times W_{out} \times C) \right] \times b \quad (5.9)$$

Lưu ý: Công thức này chỉ áp dụng khi số lượng pass cần để download 1 pass = 1 (tức là trong trường hợp Depthwise Convolution với hoặc $C \leq T_k$). Với trường hợp còn lại, công thức sẽ phức tạp hơn, tạm thời không thảo luận tới.

Trường hợp 2: Memory Bound 2 (Nghẽn băng thông trung bình) Xảy ra khi thời gian tính toán nhanh hơn tổng thời gian nạp và ghi, nhưng chậm hơn thời gian nạp ($T_{load} < T_{comp} < T_{load} + T_{store}$). Lúc này, thời gian thực thi bao gồm thời gian nạp, ghi và một phần chênh lệch thời gian tính toán.

$$T_{total} \approx T_{total\_IO} + (T_{comp} - T_{store} + (T_{comp} - T_{load}) \quad (5.10)$$

Trong đó T_{total_IO} được tính theo công thức tại Trường hợp 1 tùy thuộc loại Convolution. Lưu ý: Công thức này chỉ áp dụng khi số lượng pass cần để download 1 pass = 1 (tức là trong trường hợp Depthwise Convolution với hoặc $C \leq T_k$). Với trường hợp còn lại, công thức sẽ phức tạp hơn, tạm

thời không thảo luận tới.

Trường hợp 3: Compute Bound (Nghẽn tính toán)

Xảy ra khi thời gian tính toán lớn hơn tổng thời gian nạp và ghi ($T_{comp} > T_{load} + T_{store}$). Lúc này, toàn bộ thời gian truyền tải dữ liệu (trừ pass đầu và cuối) được che giấu hoàn toàn bên dưới thời gian tính toán.

Công thức tổng quát:

$$T_{total} = T_{load} + \sum_{all\_passes} T_{comp} + T_{store} \tag{5.11}$$

5.3.4.3 Tổng thời gian toàn mạng (Model Latency)

Thời gian thực thi của toàn bộ mô hình (Model) bao gồm N lớp tích chập là tổng thời gian của từng lớp, do sự phụ thuộc dữ liệu tuần tự giữa các lớp (Layer $i+1$ cần OFM của Layer i làm IFM):

$$T_{model} = \sum_{i=1}^{N} T_{total}^{(i)} \tag{5.12}$$

Mục tiêu của bài toán tối ưu hóa thiết kế là tìm bộ tham số cấu hình (T_h, T_m, T_c) cho từng layer sao cho $T_{total}^{(i)}$ là nhỏ nhất, cân bằng giữa tài nguyên tính toán và băng thông bộ nhớ.

5.3.5 Tự động sinh mã cấu hình (Auto-Generation)

Để vận hành hệ thống, chúng tôi xây dựng công cụ phần mềm nhằm tìm kiếm bộ tham số phân mảnh tối ưu $\mathbf{S}_i = \{T_h, T_c, T_m\}$ cho từng lớp.

Kết quả tối ưu được đóng gói thành chuỗi lệnh (Descriptor) để nạp xuống Controller sẽ có dạng như sau (chỉ có tác dụng tượng trưng ý tưởng, chưa chắc là mã thực thi sẽ được triển khai thật sự):

Bảng 5.2: Cấu trúc Descriptor điều khiển phần cứng

Offset	**Trường thông tin**	**Mô tả**
0x00 - 0x04	Layer Kernel Info	Thông tin kích thước gốc
0x08	**Tiling Config**	**Tham số tối ưu** (T_h, T_c, T_m)
0x0C - 0x14	Base Addresses	Địa chỉ vùng nhớ IFM, WGT, OFM
0x18	Control Flags	Cờ báo hiệu loại layer, hàm kích hoạt...

Chương 6

Hiện thực SoC và Tích hợp hệ thống

Chương này trình bày chi tiết quá trình hiện thực hệ thống SoC. Nội dung bao gồm việc tích hợp lõi vi xử lý PicoRV32, thiết kế các khối ngoại vi (Camera, UART, SPI, OSPI, I2C), xây dựng hệ thống Bus AXI kết nối và phát triển lớp phần mềm điều khiển (Firmware) để vận hành toàn bộ hệ thống.

6.1 Môi trường và Công cụ hiện thực

6.2 Tích hợp Lõi RISC-V và Hệ thống Bus

6.3 Thiết kế và Tích hợp các khối Ngoại vi

6.4 Phát triển Firmware và Trình điều khiển (Driver)

6.5 Quy trình Tổng hợp và Triển khai trên FPGA

Chương 7

Ước lượng hiệu năng

Chương này trình bày các kết quả thực nghiệm thu được từ mô hình ước lượng hiệu năng của kiến trúc phần cứng được đề xuất trong đồ án. Nội dung đánh giá tập trung vào việc phân tích hiệu quả của thuật toán tối ưu tham số (Codegen) và khả năng xử lý của phần cứng đối với ba lớp mô hình đại diện gồm AlexNet, VGG-16 và MobileNetV1. Bên cạnh đó, nhóm thực hiện cũng tiến hành so sánh kết quả với kiến trúc Eyeriss để làm rõ các ưu điểm và hạn chế của giải pháp thiết kế.

7.1 Môi trường và Phương pháp thực nghiệm

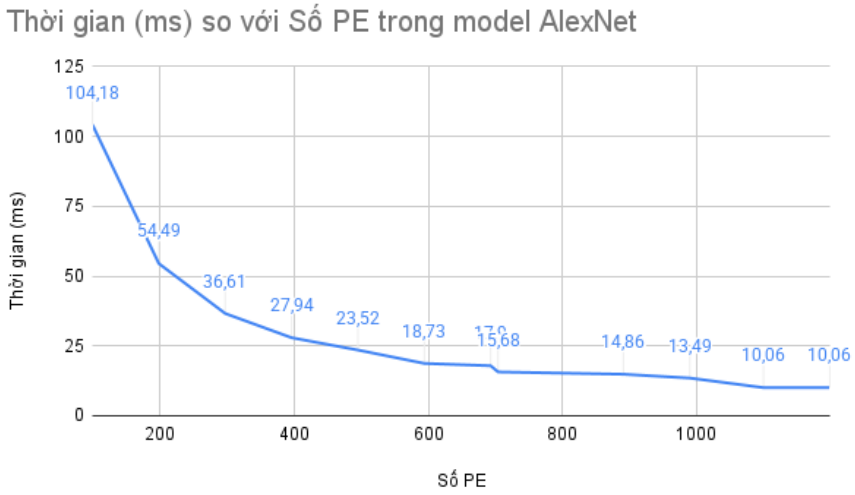
Nhằm đánh giá để tiên liệu trước về hiệu năng của đồ án này, kiến trúc được đánh giá thông qua một mô hình ước lượng (Analytical Estimator) xây dựng bằng ngôn ngữ C++. Mô hình này hiện thực hóa các công thức toán học đã được thiết lập tại Chương 4 nhằm dự báo độ trễ và tài nguyên tiêu thụ.

Các tham số cấu hình cho quá trình mô phỏng được thiết lập dựa trên các ràng buộc phần cứng dự kiến. Cụ thể, hệ thống hoạt động ở tần số 200 MHz với số lượng đơn vị xử lý (PEs) tiêu chuẩn là 385. Tốc độ của off-chip memory là 1 byte 1 cycle tương ứng tần số 200 MHz. Các giá trị

đã được quantized ở dạng int8.Phạm vi đánh giá chỉ tập trung đo đạc thời gian thực thi của các lớp tích chập (Convolutional Layers), vốn là thành phần chiếm tỷ trọng tính toán lớn nhất trong mạng CNN. Chiến lược xử lý được lựa chọn là Batch Size = 1 nhằm tối ưu hóa độ trễ cho tác vụ xử lý từng ảnh đơn lẻ. Về cấu hình bộ nhớ, đồ án giả lập kiến trúc bộ nhớ tách biệt (Separate Off-chip Memory), trong đó Trọng số (Weights) và Dữ liệu (Activations) được truy xuất trên các kênh độc lập để tối ưu băng thông.

7.2 Đánh giá khả năng xử lý trên AlexNet

AlexNet là một mạng nơ-ron tích chập điển hình, được đặc trưng bởi việc sử dụng các bộ lọc kích thước lớn ở các lớp đầu tiên (11×11, 5×5). Để đánh giá hiệu năng, đồ án thực hiện chạy quá trình sinh mã (codegen) nhằm phân tích mối tương quan giữa số lượng phần tử xử lý (PE) và thời gian hoàn thành mô hình. Kết quả phân tích được thể hiện qua biểu đồ dưới đây.



Hình 7.1: Biểu đồ tương quan giữa số lượng PE và độ trễ xử lý trên AlexNet

Quan sát biểu đồ tại Hình 7.1, có thể thấy điểm tối ưu nhất đạt được tại cấu hình $PE = 704$ với độ trễ (latency) là 14.1 ms, tương đương tốc độ

khung hình 70.92 fps. Nguyên nhân là khi tăng số lượng PE, số lượng các kênh bản đồ đặc trưng đầu ra (ofmap) được tính toán song song sẽ tăng lên, đồng nghĩa với việc số lần phải tải lại toàn bộ bản đồ đặc trưng đầu vào (ifmap) giảm xuống. Với các giá trị T_m tăng dần, thời gian xử lý giảm xuống rất nhanh. Tuy nhiên, khi chênh lệch giữa thời gian tải ifmap và thời gian tải ofmap không còn đáng kể, tốc độ giảm của thời gian xử lý sẽ bắt đầu bão hòa và chậm dần. Kết quả mô phỏng chi tiết với giới hạn tài nguyên 704 PEs được trình bày tại Bảng 7.1.

Bảng 7.1: Chi tiết hiệu năng từng lớp của AlexNet (Cập nhật theo Log - Total: 14.10 ms)

Layer	**Filter Size**	**PE Used**	**Optimized Config**			**Latency (ms)**	**Bottleneck**
			$\mathbf{T_k}$	$\mathbf{T_m}$	$\mathbf{T_h}$		
Conv1	11×11	528	1	48	11	5.17	Compute
Conv2	5×5	640	1	64	5	3.53	Compute
Conv3	3×3	576	1	64	3	1.62	Memory
Conv4	3×3	576	1	64	3	2.27	Memory
Conv5	3×3	576	1	64	3	1.51	Memory
Total	**-**	**Max 640**	**-**	**-**	**-**	**14.10**	**Mixed**

Dựa trên kết quả thực nghiệm được trình bày tại Bảng 7.1, nhóm em có những phân tích cụ thể về hiệu năng của mô hình AlexNet với tổng thời gian thực thi là 14.10 ms như sau:

Trước hết, về đặc thù tính toán, hệ thống cho thấy một sự phân hóa rõ rệt giữa các tầng dựa trên kích thước bộ lọc. Tại hai lớp đầu tiên là Conv1 và Conv2, do sử dụng các bộ lọc có kích thước lớn (11×11 và 5×5), mật độ tính toán trên mỗi đơn vị dữ liệu là rất cao. Điều này cho phép hệ thống huy động tối đa tài nguyên với số lượng PE sử dụng lần lượt là 528 và 640. Trong giai đoạn này, hệ thống rơi vào trạng thái *Compute Bound*, nghĩa là năng lực xử lý của mảng PE là yếu tố chính quyết định độ trễ. Việc tối ưu hóa các tham số cấu hình như T_m và T_h đã giúp tận dụng tốt sức mạnh

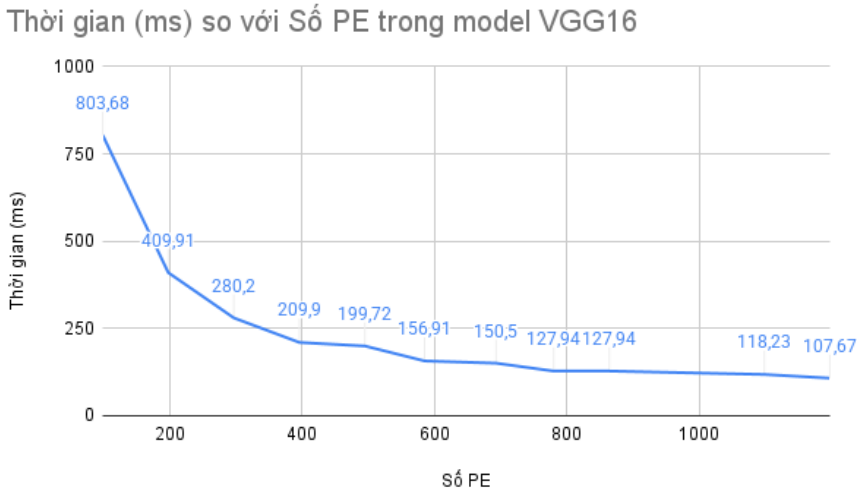
phần cứng để xử lý khối lượng công việc khổng lồ, chiếm tỷ trọng lớn trong tổng thời gian thực thi.

Tuy nhiên, đối với các lớp từ Conv3 đến Conv5, mặc dù các tham số song song hóa vẫn được duy trì ở mức cao ($T_m = 64$ và sử dụng 576 PEs), nhưng điểm nghẽn hệ thống đã chuyển dịch hoàn toàn sang trạng thái *Memory Bound*. Do kích thước bộ lọc giảm xuống đáng kể (3×3), khối lượng tính toán trên mỗi lần truy xuất dữ liệu không còn đủ lớn để che giấu độ trễ của bộ nhớ. Hệ quả là mảng PE thường xuyên phải chờ dữ liệu, khiến thời gian xử lý tại các lớp này phụ thuộc vào băng thông hơn là tốc độ tính toán thuần túy. Đặc biệt, lớp Conv4 ghi nhận mức độ trễ 2.27 ms, cao hơn so với Conv3 và Conv5, cho thấy đây là điểm nhạy cảm về mặt lưu lượng dữ liệu.

Tổng kết lại, mô hình đang vận hành dưới một cơ chế điểm nghẽn hỗn hợp (*Mixed Bottleneck*). Để cải thiện tổng độ trễ 14.10 ms này, việc chỉ tăng thêm số lượng PE sẽ không mang lại hiệu quả tối ưu cho các lớp sau. Thay vào đó, cần có các giải pháp về cải thiện băng thông hoặc tối ưu hóa sơ đồ luồng dữ liệu (*Dataflow*) để giải quyết tình trạng nghẽn bộ nhớ tại các tầng có bộ lọc kích thước nhỏ.

7.3 Đánh giá khả năng xử lý trên VGG-16

Để kiểm chứng khả năng chịu tải của hệ thống đối với các mạng nơ-ron tích chập có độ sâu lớn, đồ án thực hiện mô phỏng trên mô hình VGG-16. Quá trình sinh mã và phân tích sự tương quan giữa số lượng PE và thời gian hoàn thành mô hình cho kết quả như biểu đồ sau.



Hình 7.2: Biểu đồ tương quan giữa số lượng PE và độ trễ xử lý trên VGG-16

Dựa vào biểu đồ Hình 7.2, điểm tối ưu nhất được xác định tại $PE = 390$ với độ trễ đạt 216.99 ms, tương đương 4.61 fps. Tương tự như AlexNet, việc tăng số lượng PE giúp tăng số kênh ofmap được tính toán song song và giảm số lần tải lại ifmap. Tuy nhiên, khi độ trễ giữa các lần truy cập bộ nhớ giảm xuống đến mức bão hòa, đường cong hiệu năng cũng dần đi ngang. Kết quả mô phỏng chi tiết trên tập cấu hình phần cứng tối ưu với giới hạn 390 PEs được trình bày tại Bảng 7.2.

Bảng 7.2: Chi tiết hiệu năng từng lớp của VGG-16 (Cập nhật theo Log - Total: 216.99 ms)

Layer	Filter / Channels	Map Size	PE Used	Optimized Config			Latency (ms)	Bottleneck
				T_k	T_m	T_h		
Conv1_1	$3 \times 3/64$	224×224	198	1	33	224	10.13	Memory
Conv1_2	$3 \times 3/64$	224×224	384	1	64	3	32.12	Memory
Conv2_1	$3 \times 3/128$	112×112	390	1	65	112	12.16	Memory
Conv2_2	$3 \times 3/128$	112×112	384	1	64	3	24.09	Memory
Conv3_1	$3 \times 3/256$	56×56	384	1	64	3	12.04	Memory
Conv3_2	$3 \times 3/256$	56×56	384	1	64	3	24.09	Compute
Conv3_3	$3 \times 3/256$	56×56	384	1	64	3	24.09	Compute
Conv4_1	$3 \times 3/512$	28×28	384	1	64	3	12.04	Compute
Conv4_2	$3 \times 3/512$	28×28	384	1	64	3	24.09	Compute
Conv4_3	$3 \times 3/512$	28×28	384	1	64	3	24.09	Compute
Conv5_1	$3 \times 3/512$	14×14	384	1	64	3	6.02	Compute
Conv5_2	$3 \times 3/512$	14×14	384	1	64	3	6.02	Compute
Conv5_3	$3 \times 3/512$	14×14	384	1	64	3	6.02	Compute
Total	–	–	**Max 390**	–	–	–	**216.99**	**Mixed**

Kết quả mô phỏng thực tế xác nhận tính hiệu quả cao trong chiến lược phân bổ tài nguyên phần cứng cho mô hình VGG-16 với tổng thời gian thực thi đạt 216.99 ms. Việc thiết lập giới hạn phần cứng tại mức MAX_M = 65 và PE Limit = 390 cho thấy sự tương thích đặc biệt với kiến trúc của VGG-16, nơi các tham số về kênh (channels) thường là bội số của 64 hoặc 32. Hệ thống duy trì mức sử dụng tài nguyên rất cao, đạt tối đa 390 PEs tại lớp Conv2_1 và ổn định ở mức 384 PEs cho hầu hết các lớp còn lại, minh chứng cho khả năng song song hóa tối ưu trên chiều kênh đầu ra (T_m) và chiều cao khối dữ liệu (T_h).

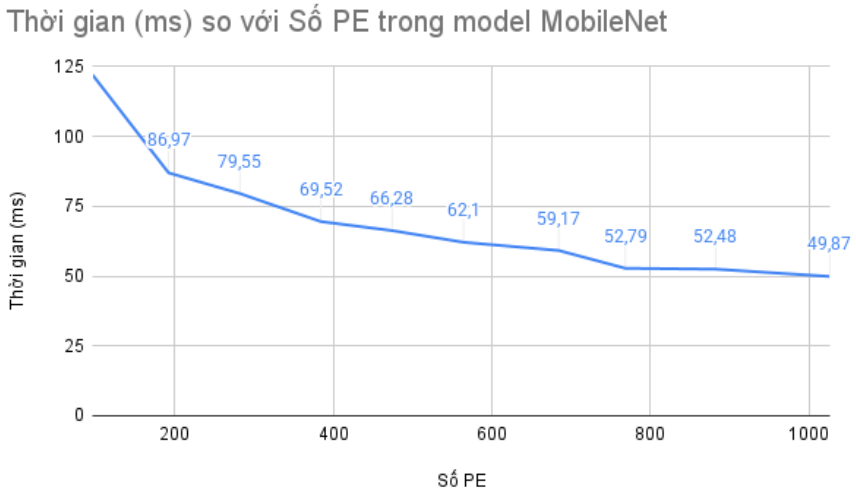
Sự phân hóa về điểm nghẽn hệ thống (Bottleneck) trong bảng dữ liệu phản ánh rõ rệt tác động của việc tăng cường năng lực tính toán. Tại các tầng đầu tiên của mạng (từ Conv1_1 đến Conv3_1), hệ thống hoàn toàn bị giới hạn bởi băng thông bộ nhớ (Memory Bound). Điều này xảy ra do kích thước bản đồ đặc trưng (Map Size) còn quá lớn (224×224 và 112×112), khiến khối lượng dữ liệu cần nạp vào và xuất ra vượt quá tốc độ xử lý của các đơn vị PE, dù số lượng PE sử dụng đã đạt ngưỡng tối đa. Trong giai

đoạn này, sự chênh lệch chu kỳ (Difference cycle) giữa bộ nhớ và tính toán là số dương rất lớn, khẳng định rằng các PE phải chờ đợi dữ liệu từ bộ nhớ đệm.

Ngược lại, bắt đầu từ lớp Conv3_2 trở đi, khi kích thước bản đồ đặc trưng giảm xuống mức 56×56 và nhỏ hơn, hệ thống chuyển sang trạng thái giới hạn tính toán (Compute Bound). Lúc này, độ sâu của các bộ lọc tăng lên khiến khối lượng phép tính nhân tích lũy (MAC) trở nên đồ sộ, đủ để che lấp độ trễ truy xuất bộ nhớ. Với tổng thời gian xử lý rút ngắn xuống còn 216.99 ms so với các cấu hình thấp hơn, hệ thống đã đạt đến điểm tối ưu về hiệu suất, cho phép đáp ứng tốt các ứng dụng nhận diện hình ảnh thời gian thực, đồng thời chỉ ra rằng các nỗ lực tối ưu hóa tiếp theo nên tập trung vào việc cải thiện băng thông bộ nhớ cho các lớp tích chập ban đầu.

7.4 Đánh giá khả năng xử lý trên MobileNet v1

Khác với VGG-16, MobileNet v1 sử dụng kiến trúc tích chập tách biệt theo chiều sâu (Depthwise Separable Convolution) nhằm giảm khối lượng tính toán. Đồ án tiến hành chạy codegen để phân tích sự tương quan giữa số lượng PE và thời gian hoàn thành mô hình, kết quả được thể hiện trong biểu đồ sau.



Hình 7.3: Biểu đồ tương quan giữa số lượng PE và độ trễ xử lý trên MobileNet v1

Dựa trên biểu đồ Hình 7.3, điểm tối ưu nhất đạt được tại $PE = 384$ với độ trễ 76.64 ms, tương đương 13 fps. Tốc độ giảm của thời gian hoàn thành khi tăng số PE chậm hơn rất nhiều so với biểu đồ của AlexNet. Lý do chính là hệ thống không vận dụng được cơ chế tích luỹ theo chiều sâu, vốn là điểm mạnh của mô hình tại các lớp Depthwise Convolution. Điều này dẫn tới thời gian xử lý không giảm đáng kể ở các lớp này, mà chủ yếu chỉ được cải thiện ở các lớp Pointwise Convolution và Standard Convolution. Kết quả mô phỏng trên tập cấu hình phần cứng tối ưu với giới hạn 385 PEs được trình bày tại Bảng 7.3.

Bảng 7.3: Hiệu năng chi tiết từng lớp của MobileNet v1 (Total Latency: 76.64 ms)

Layer Type	**Layer Idx**	**PE Used**	**Optimized Config**			**Latency (ms)**	**Bottleneck**
			$\mathbf{T_k}$	$\mathbf{T_m}$	$\mathbf{T_h}$		
Standard Conv	0	96	1	32	3	2.77	Memory
Depthwise	1	3	1	1	3	6.02	Compute
Pointwise	2	192	1	64	1	6.02	Memory
Depthwise	3	3	1	1	3	7.99	Compute
Pointwise	4	384	1	128	1	3.01	Memory
Depthwise	5	3	1	1	3	6.02	Compute
Pointwise	6	384	1	128	1	4.01	Memory
Depthwise	7	3	1	1	3	3.98	Compute
Pointwise	8	384	1	128	1	2.01	Memory
Depthwise	9	3	1	1	3	3.01	Compute
Pointwise	10	384	1	128	1	3.01	Memory
Depthwise	11	3	1	1	3	1.97	Compute
Pointwise	12	384	1	128	1	1.51	Memory
Depthwise	13	3	1	1	3	1.51	Compute
Pointwise	14, 16, 18, 20, 22	384	1	128	1	2.51 (avg)	Memory
Depthwise	15, 17, 19, 21	3	1	1	3	1.51 (avg)	Compute
Depthwise	23	3	1	1	3	0.97	Compute
Pointwise	24	384	1	128	1	1.25	Memory
Depthwise	25	3	1	1	3	0.75	Compute
Pointwise	26	384	1	128	1	2.26	Memory
Total	**All Layers**	**Max 384**	**-**	**-**	**-**	**76.64**	**Mixed**

**Ghi chú: Dựa trên log thực nghiệm, các lớp Depthwise bị giới hạn bởi tính toán (Compute Bound) do cấu hình $T_m = 1$ dẫn đến số PE sử dụng rất thấp (3 PE), trong khi các lớp Pointwise bị giới hạn bởi băng thông bộ nhớ (Memory Bound) khi sử dụng tối đa 384 PE.*

Dựa trên bảng số liệu thực nghiệm, kết quả cho thấy hệ thống có sự chuyển đổi trạng thái liên tục giữa *Compute Bound* và *Memory Bound* (hiện tượng Toggle Bottleneck), phản ánh chính xác đặc thù kiến trúc phân mảnh của mô hình MobileNet v1. Trong cơ chế này, các lớp Pointwise (1×1 Conv)

hầu như luôn rơi vào trạng thái nghẽn bộ nhớ (*Memory Bound*). Do đặc tính tái sử dụng dữ liệu thấp nhưng khả năng song song hóa cực cao, hệ thống đã tận dụng triệt để 384 PEs với cấu hình tối ưu $T_m = 128$ nhằm tiêu thụ dữ liệu nhanh hơn tốc độ cung cấp của băng thông, khiến hiệu năng bị giới hạn bởi tốc độ truy xuất từ bộ nhớ ngoài.

Ngược lại, các lớp Depthwise (3×3 DW) tuy được log hệ thống đánh dấu là nghẽn tính toán (*Compute Bound*), nhưng thực chất đây là hệ quả của hiện tượng giới hạn do kém hiệu quả trong việc sử dụng tài nguyên (*under-utilization*). Cấu trúc đặc thù của phép toán Depthwise không có sự cộng gộp giữa các kênh ($C_{in} = C_{out}$), khiến cấu hình phần cứng hiện tại buộc phải giảm tham số tối ưu xuống mức thấp nhất là $T_m = 1$. Hệ quả là trong tổng số 384 PEs sẵn có, chỉ có vỏn vẹn 3 PEs được kích hoạt để xử lý, trong khi 381 PEs còn lại rơi vào trạng thái nhàn rỗi, trực tiếp kéo dài thời gian tính toán của các lớp này một cách không cần thiết.

Đáng chú ý, mặc dù khối lượng tính toán lý thuyết của các lớp Depthwise là rất nhỏ so với Pointwise, chúng lại chiếm tới hơn 50% tổng thời gian thực thi của toàn bộ mô hình (khoảng 38.2 ms trên tổng số 76.64 ms). Sự mất cân đối này chỉ ra một điểm yếu trong thiết kế phần cứng hiện tại khi chưa tối ưu cho các kiến trúc mạng nơ-ron hiện đại. Để đạt được độ trễ thấp hơn và tối ưu hóa hiệu suất trên mỗi đơn vị tài nguyên, kiến trúc phần cứng cần được cải tiến để hỗ trợ cơ chế song song hóa kênh (Channel Parallelism) đặc thù cho các lớp Depthwise, thay vì chỉ dựa trên các khối nhân ma trận truyền thống.

7.5 So sánh với các Nghiên cứu liên quan

Để đánh giá khách quan hiệu quả của kiến trúc đề xuất, chúng tôi thực hiện so sánh đối chứng với kết quả đo đạc thực tế trên silicon của chip gia tốc Eyeriss [Chen et al., ISSCC 2016]. Các thông số tham chiếu của Eyeriss được lấy từ cấu hình tối ưu với Batch Size $N = 4$ cho AlexNet và $N = 3$ cho

VGG16.Nhằm đảm bảo tính tương đồng trong điều kiện thử nghiệm, kiến trúc đề xuất được cấu hình với 165 PE (xấp xỉ số lượng PE của Eyeriss), đồng thời sử dụng mô hình AlexNet với Grouped Convolution tại các lớp conv2, conv4 và conv5 đúng theo nguyên bản. Bên cạnh đó, các tham số hệ thống cũng được đồng bộ hóa với thiết kế Eyeriss: băng thông bộ nhớ ngoài (off-chip memory) giới hạn ở mức 480MB/s và độ chính xác tính toán là 16-bit fixed-point. Kết quả so sánh chi tiết được trình bày tại Bảng 7.4.

Bảng 7.4: So sánh hiệu năng xử lý Convolution trên AlexNet và VGG16

Thông số	**Đề xuất (Ours)**	**Eyeriss [Chen et al.]**
Số lượng PE	165	168
Kiến trúc bộ nhớ	Separate Off-chip Memory	Shared DRAM
Chiến lược xử lý	**Batch Size = 1** (Real-time)	**Batch Size = 3-4** (Throughput)
AlexNet (Latency)	45.87 ms (21.8 fps)	**28.57 ms*** (35.0 fps)
VGG16 (Latency)	**510.48 ms** (1.96 fps)	1428.57 ms** (0.7 fps)

**AlexNet Eyeriss: Tính trung bình trên Batch=4 ($N = 4$).*

***VGG16 Eyeriss: Tính trung bình trên Batch=3 ($N = 3$).*

Kết quả so sánh cho thấy hai xu hướng đối lập tương ứng với độ phức tạp của mạng nơ-ron:

- **Đối với VGG16 (Mạng sâu và nặng):** Kiến trúc đề xuất đạt hiệu năng vượt trội với độ trễ thấp hơn khoảng **2.78 lần** so với Eyeriss (514.31 ms so với 1428.57 ms). Kết quả này đạt được nhờ cấu hình phần cứng tối ưu ($T_m = 28$) giúp tận dụng tối đa 100% tài nguyên PE ở hầu hết các lớp. Đồng thời, trạng thái *Compute Bound* ổn định chứng tỏ hệ thống bộ nhớ tách biệt đã loại bỏ được nút thắt cổ chai về dữ liệu mà các kiến trúc dùng chung bộ nhớ (Shared DRAM) thường gặp phải.
- **Đối với AlexNet (Mạng nông, có Grouped Convolution):** Eyeriss giữ lợi thế về thông lượng (35 fps) nhờ cơ chế xử lý theo lô ($N = 4$) và kiến trúc luồng dữ liệu (Dataflow) đặc thù giúp xử lý hiệu

quả việc phân chia nhóm kênh. Tuy nhiên, giải pháp đề xuất ($N = 1$) vẫn đạt độ trễ xấp xỉ 45ms. Đây là kết quả khả quan, cung cấp khả năng phản hồi thời gian thực (Real-time) tốt hơn cho các ứng dụng đơn lẻ, loại bỏ được độ trễ tích lũy (batching latency) mà cơ chế xử lý theo lô của Eyeriss gặp phải.

Chương 8

Kết luận và Hướng phát triển

Chương này tổng kết các kết quả đạt được trong Giai đoạn 1 và đề ra kế hoạch chi tiết cho việc hiện thực và kiểm thử trong Giai đoạn 2.

8.1 Đánh giá mức độ hoàn thành Giai đoạn 1

8.2 Kế hoạch thực hiện Giai đoạn 2

8.3 Tiến độ dự kiến