

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 mục Ký hiệu và Chữ viết tắt	x
<b>1 Giới thiệu đề tài</b>	<b>1</b>
1.1 Tổng quan về đề tài . . . . .	1
1.2 Mục tiêu và Nhiệm vụ nghiên cứu . . . . .	2
1.3 Phạm vi đề tài . . . . .	3
1.3.1 Phạm vi và Giới hạn đề tài . . . . .	3
1.3.2 Đối tượng và Công cụ nghiên cứu . . . . .	3
1.4 Phân chia công việc . . . . .	4
1.5 Cấu trúc báo cáo . . . . .	6
<b>2 Cơ sở lý thuyết</b>	<b>7</b>
2.1 Kiến trúc tập lệnh RISC-V . . . . .	7
2.1.1 Tổng quan về kiến trúc RISC-V . . . . .	7
2.1.2 Mô hình lập trình và Tập thanh ghi . . . . .	8
2.1.2.1 Bộ đếm chương trình (Program Counter - PC) . . . . .	8
2.1.2.2 Tập thanh ghi mục đích chung (General Purpose Registers) . . . . .	9
2.1.3 Đặc tả tập lệnh cơ sở RV32I . . . . .	10
2.1.3.1 Định dạng lệnh (Instruction Formats) .	10
2.1.3.2 Phân nhóm chức năng chi tiết . . . . .	11
2.1.4 Vị xử lý PicoRV32 . . . . .	13

2.2	Tổng quan về Mạng nơ-ron tích chập (CNN) . . . . .	13
2.3	Các chuẩn giao tiếp hệ thống . . . . .	13
2.3.1	Chuẩn giao tiếp AMBA AXI4 . . . . .	13
2.3.1.1	Kiến trúc 5 kênh độc lập (Channel Ar-chitecture) . . . . .	15
2.3.1.2	Cơ chế bắt tay (Handshake Mechanism)	16
2.3.1.3	Quy trình thực hiện giao dịch chi tiết (Transaction Steps) . . . . .	18
2.3.1.4	Cấu trúc giao dịch Burst (Burst Trans-action) . . . . .	21
2.3.1.5	Các biến thể giao thức trong thiết kế . .	21
2.3.1.6	Áp dụng trong hệ thống đè tài . . . . .	22
2.3.2	Giao thức truyền thông UART . . . . .	23
2.3.2.1	Nguyên lý hoạt động . . . . .	23
2.3.2.2	Cấu trúc khung dữ liệu (Data Frame) .	24
2.3.2.3	Tốc độ Baud (Baud Rate) . . . . .	25
2.3.3	Giao thức truyền thông SPI . . . . .	26
2.3.3.1	Cấu hình tín hiệu vật lý . . . . .	26
2.3.3.2	Cơ chế hoạt động: Thanh ghi dịch (Shift Register) . . . . .	27
2.3.3.3	Các chế độ hoạt động (Clock Polarity & Phase) . . . . .	28
2.3.3.4	Các mô hình kết nối đa thiết bị . . . . .	29
2.3.4	Giao thức giao tiếp OSPI (Octal SPI) . . . . .	32
2.3.4.1	Cấu hình tín hiệu vật lý . . . . .	32
2.3.4.2	Cơ chế truyền tải DDR (Double Data Rate)	33
2.3.4.3	Cấu trúc giao dịch Octal-DDR . . . . .	34
2.3.4.4	Ưu điểm trong ứng dụng SoC IoT . . . . .	35
2.4	Công nghệ FPGA và Quy trình thiết kế . . . . .	35

<b>3 Phân tích và</b>	
<b>Kiến trúc hệ thống</b>	<b>36</b>
3.1 Phân tích yêu cầu thiết kế . . . . .	36
3.2 Kiến trúc tổng thể SoC . . . . .	36
3.3 Đặc tả các khối chức năng chính . . . . .	36
3.4 Tổ chức bộ nhớ và Bản đồ địa chỉ (Memory Map) . . . . .	36
<b>4 Thiết kế Bộ tăng tốc AI (AI Accelerator)</b>	<b>37</b>
4.1 Cơ sở Toán học và Thách thức Thiết kế . . . . .	37
4.1.1 Standard Convolution (Tích chập tiêu chuẩn) . .	38
4.1.1.1 Mô hình toán học . . . . .	38
4.1.1.2 Thuật toán xử lý . . . . .	38
4.1.2 Depthwise Separable Convolution . . . . .	39
4.1.2.1 Depthwise Convolution (DW) . . . . .	40
4.1.2.2 Pointwise Convolution (PW) . . . . .	40
4.1.3 Tối ưu hóa: Kỹ thuật Gộp Batch Normalization (BN Folding) . . . . .	41
4.2 Chiến lược Phân mảnh và Quản lý Dòng dữ liệu . . . . .	41
4.2.1 Chiến lược Phân mảnh không gian (Space Partitioning) . . . . .	42
4.2.2 Mô hình hóa và Tham số thiết kế . . . . .	42
4.2.3 Bài toán Dữ liệu biên và Cơ chế Ping-Pong . . . .	43
4.2.3.1 Phân tích Dữ liệu dội ra (Residual Data) .	44
4.2.3.2 Logic Hoạt động Ping-Pong . . . . .	44
4.2.4 Thuật toán Điều phối Pass (Pass Scheduling) . .	46
4.2.4.1 Trường hợp Standard Convolution . . . . .	46
4.2.4.2 Trường hợp Depthwise Convolution . . . . .	47
4.3 Thiết kế Kiến trúc Vi mô (Micro-architecture) . . . . .	49
4.3.1 Sơ đồ khái quát . . . . .	49

4.3.2	Tổ chức Phân cấp Đơn vị Tính toán . . . . .	50
4.3.2.1	Mảng xử lý (Process Array - PA) . . . . .	50
4.3.2.2	Đơn vị xử lý (Process Unit - PU) . . . . .	51
4.3.2.3	Phần tử xử lý (Process Element - PE) . .	51
4.3.3	Đánh giá thời gian thực thi (Performance Estimation) . . . . .	52
4.3.3.1	Thời gian xử lý một Pass cơ sở ( $T_{pass}$ ) .	52
4.3.3.2	Tổng thời gian thực thi ( $T_{total}$ ) . . . . .	52
4.3.4	Mô hình hóa độ trễ toàn hệ thống . . . . .	53
4.3.4.1	Cơ chế hoạt động . . . . .	53
4.3.4.2	Các kịch bản hiệu năng (Performance Scenarios) . . . . .	54
4.3.4.3	Tổng thời gian toàn mạng (Model Latency)	56
4.3.5	Tự động sinh mã cấu hình (Auto-Generation) . .	57
<b>5</b>	<b>Hiện thực SoC và Tích hợp hệ thống</b>	<b>58</b>
5.1	Môi trường và Công cụ hiện thực . . . . .	59
5.2	Tích hợp Lõi RISC-V và Hệ thống Bus . . . . .	59
5.3	Thiết kế và Tích hợp các khối Ngoại vi . . . . .	59
5.4	Phát triển Firmware và Trình điều khiển (Driver) . . . .	59
5.5	Quy trình Tổng hợp và Triển khai trên FPGA . . . .	59
<b>6</b>	<b>Đánh giá và Thảo luận kết quả</b>	<b>60</b>
6.1	Môi trường và Phương pháp thực nghiệm . . . . .	60
6.2	Đánh giá khả năng xử lý trên AlexNet . . . . .	61
6.3	Đánh giá khả năng xử lý trên VGG-16 . . . . .	62
6.4	Đánh giá khả năng xử lý trên MobileNet v1 . . . .	64
6.5	So sánh với các Nghiên cứu liên quan . . . . .	65
<b>7</b>	<b>Kết luận và Hướng phát triển</b>	<b>67</b>

7.1	Dánh giá mức độ hoàn thành Giai đoạn 1 . . . . .	67
7.2	Kế hoạch thực hiện Giai đoạn 2 . . . . . . . . .	67
7.3	Tiến độ dự kiến . . . . . . . . . . . . . . . . .	67

# Danh sách hình vẽ

Figure 2.1 Cấu trúc bit của các định dạng lệnh RV32I . . . . .	10
Figure 2.2 a. Tổng quan giao thức AXI4 . . . . .	14
Figure 2.3 b. Tổng quan giao thức AXI4 . . . . .	15
Figure 2.4 Mô hình 5 kênh giao tiếp của AXI4 . . . . .	15
Figure 2.5 Cơ chế bắt tay VALID/READY trong AXI . . . . .	16
Figure 2.6 Minh họa một Transfer trong AXI . . . . .	17
Figure 2.7 Minh họa một Transaction trong AXI . . . . .	18
Figure 2.8 Giản đồ tín hiệu chi tiết của giao dịch Ghi . . . . .	19
Figure 2.9 Giản đồ tín hiệu chi tiết của giao dịch Đọc . . . . .	20
Figure 2.10 Minh họa chân kết nối truyền nhận dữ liệu UART . . .	24
Figure 2.11 Chuyển đổi dữ liệu song song thành nối tiếp và ngược lại trong UART . . . . .	24
Figure 2.12 Khung dữ liệu UART . . . . .	25
Figure 2.13 Ví dụ khung dữ liệu UART với 8bit dữ liệu, không parity và 1 stop bit . . . . .	25
Figure 2.14 Sơ đồ kết nối tín hiệu chuẩn 4 dây của SPI . . . . .	26
Figure 2.15 Cơ chế trao đổi dữ liệu dùng thanh ghi dịch trong SPI .	27

Figure 2.16 4 chế độ hoạt động của SPI(CPOL/CPHA) . . . . .	28
Figure 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 . . . . .	29
Figure 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 . . . . .	29
Figure 2.19 Cấu hình Slave độc lập trong SPI . . . . .	30
Figure 2.20 Cấu hình Chuỗi (Daisy Chain) trong SPI . . . . .	31
Figure 2.21 Sơ đồ chân tín hiệu của giao diện OSPI/HyperBus . . .	32
Figure 2.22 Giản đồ thời gian truyền tải SDR: Dữ liệu thay đổi ở cạnh lệnh, DDR: Dữ liệu thay đổi ở cả hai cạnh của xung nhịp . . . . .	34
Figure 2.23 Giản đồ thời gian giao dịch OSPI DDR: Command, Ad- dress và Data truyền trên 8 dây IO . . . . .	35
Figure 4.1 Minh họa sự hình thành dữ liệu dôi ra. Tại hàng 2 và 3, bộ lọc thiêu dữ liệu từ hàng 4, 5 (thuộc tile sau) nên kết quả chưa hoàn thiện. . . . .	44
Figure 4.2 Cơ chế Ping-Pong Buffer luân phiên để quản lý vùng dữ liệu biên liên tục. . . . .	45
Figure 4.3 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). . . . .	48
Figure 4.4 Kiến trúc Beta Accelerator với Bus dữ liệu và Trọng số tách biệt. . . . .	49



# Danh sách bảng biểu

Table 1.1	Bảng phân chia công việc của các thành viên . . . . .	5
Table 2.1	Tập thanh ghi mục đích chung của RISC-V (RV32I)	9
Table 4.1	Bảng tham số thiết kế và ánh xạ ký hiệu . . . . .	43
Table 4.2	Cấu trúc Descriptor điều khiển phần cứng . . . . .	57
Table 6.1	Chi tiết hiệu năng từng lớp của AlexNet (Mô phỏng với 168 PEs) . . . . .	61
Table 6.2	Chi tiết hiệu năng từng lớp của VGG-16 (Mô phỏng với 168 PEs) . . . . .	63
Table 6.3	Hiệu năng chi tiết từng lớp của MobileNet v1 (Total Latency: 109.88 ms) . . . . .	64
Table 6.4	So sánh hiệu năng xử lý Convolution trên AlexNet và VGG16 . . . . .	66

# 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 chêch (Bias)
$\mu, \sigma$	Giá trị trung bình (Mean) và Phương sai (Variance) trong Batch Norm
$\gamma, \beta$	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**

Dồ 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	<b>Lâm Nữ Uyển Nhi</b> (Chịu trách nhiệm về Accelerator)	<ul style="list-style-type: none"> <li>Nghiên cứu lý thuyết về mạng CNN và các kỹ thuật tối ưu phần cứng.</li> <li>Thiết kế kiến trúc khối CNN Accelerator (PE Array, Buffer Controller).</li> <li>Hiện thực các khối tính toán: Convolution, Pooling, ReLU.</li> <li>Viết Testbench kiểm tra chức năng khối Accelerator.</li> </ul>
2	<b>Vũ Đức Lâm</b> (Chịu trách nhiệm về SoC & System)	<ul style="list-style-type: none"> <li>Nghiên cứu kiến trúc RISC-V và chuẩn bus AMBA AXI.</li> <li>Thiết kế hệ thống SoC: Tích hợp CPU, Interconnect, Memory Controller.</li> <li>Thiết kế các giao tiếp ngoại vi: UART, SPI, OSPI, I2C, GPIO, TIMER, DVP (Camera/HDMI).</li> <li>Phát triển Firmware/Driver để điều khiển hệ thống.</li> <li>Tổng hợp hệ thống lên FPGA và đo đặc hiệu năng.</li> </ul>

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

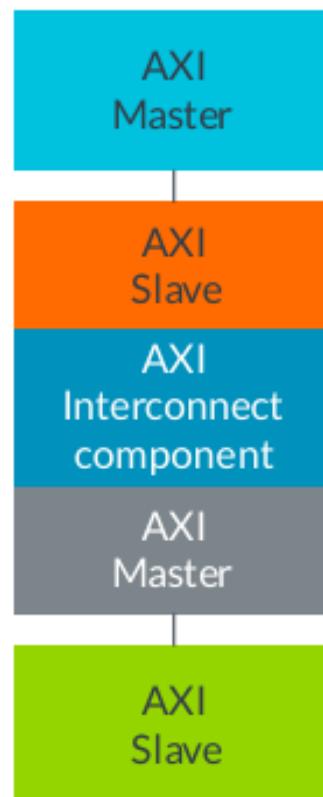
## 2.3 Các chuẩn giao tiếp hệ thống

### 2.3.1 Chuẩn giao tiếp AMBA AXI4

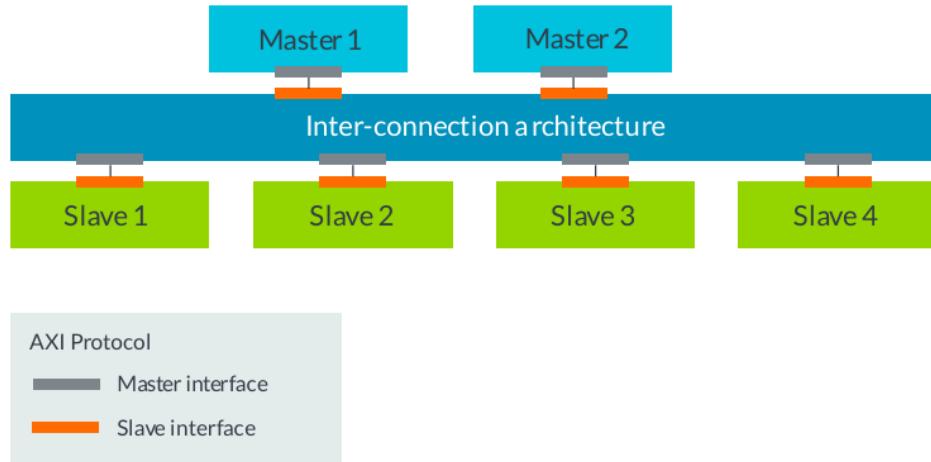
AMBA (Advanced Microcontroller Bus Architecture) là tiêu chuẩn kết nối trên chip (On-Chip Interconnect) phổ biến nhất hiện nay, được phát triển bởi ARM. Trong đó, giao thức AXI (Advanced eXtensible Interface) là chuẩn giao tiếp hiệu năng cao, được thiết kế cho các hệ thống SoC yêu cầu băng thông lớn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



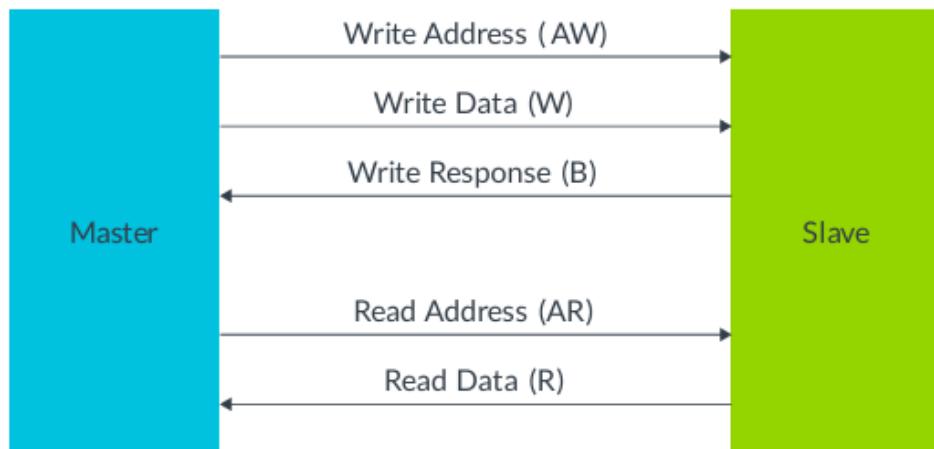
**Hình 2.2: a. Tổng quan giao thức AXI4**



**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.

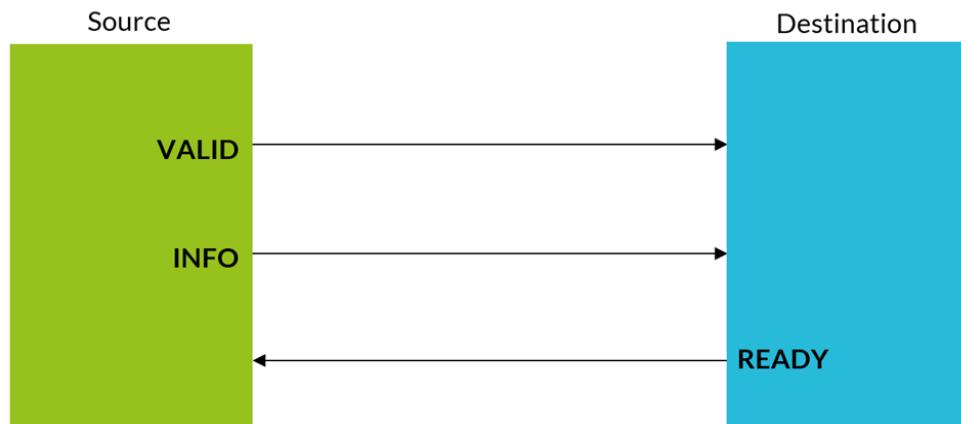


**Hình 2.5:** Cơ chế bắt tay VALID/READY trong AXI

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.

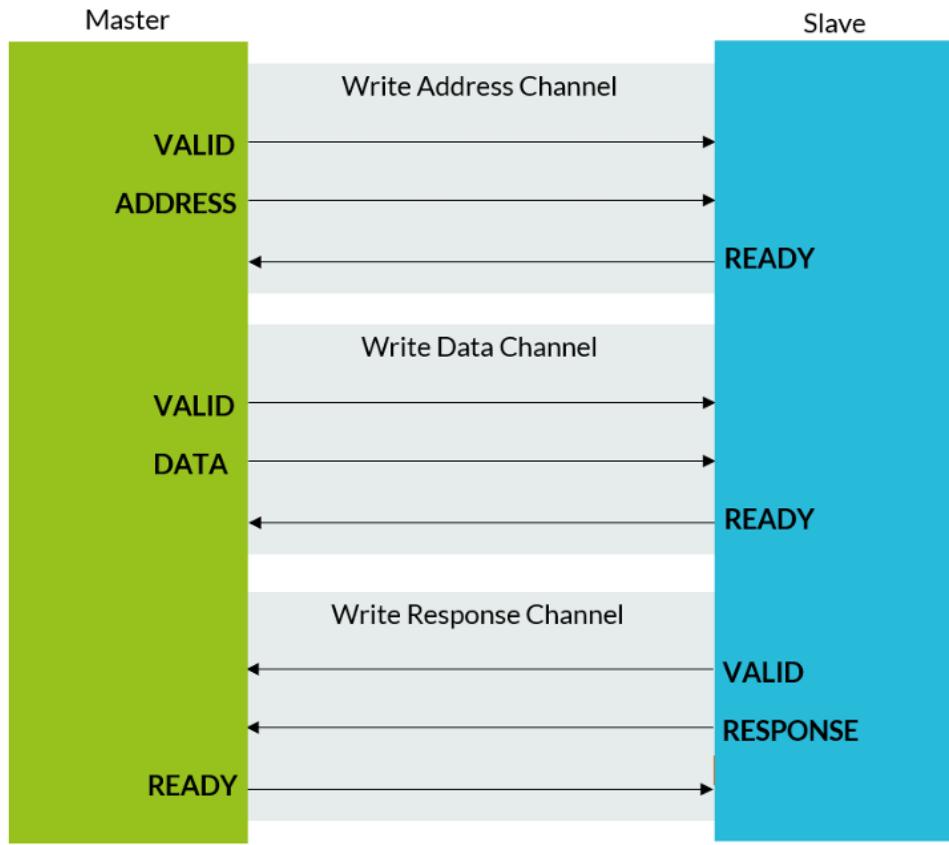
Dựa trên cơ chế bắt tay này, chuẩn AXI định nghĩa hai cấp độ truyền tải dữ liệu cần phân biệt rõ:

- **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).



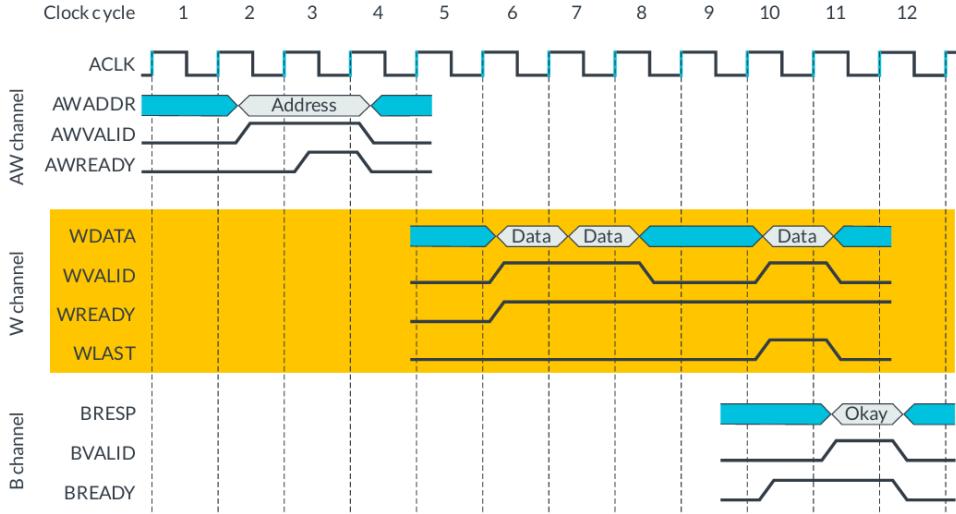
**Hình 2.7:** Minh họa một Transaction trong AXI

### 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.

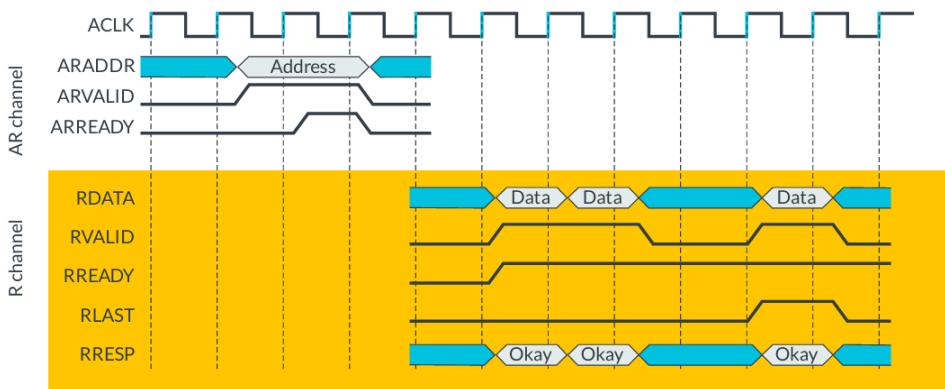
- **Pha phản hồi (Write Response Channel):**

- **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 đề 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). Vì 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

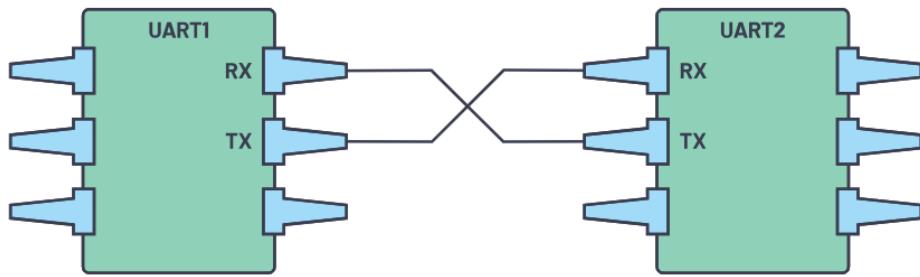
UART (Universal Asynchronous Receiver-Transmitter) là một vi mạch phần cứng dùng để truyền tải dữ liệu nối tiếp giữa hai thiết bị. Khác với các giao thức đồng bộ như SPI hay I2C, UART hoạt động theo cơ chế bất đồng bộ (Asynchronous), nghĩa là không cần tín hiệu xung nhịp (Clock) chung để đồng bộ hóa 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

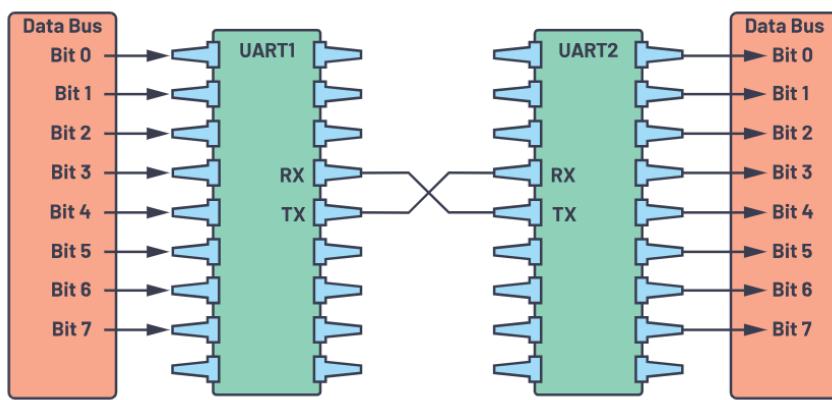
Giao thức UART truyền dữ liệu trên hai dây tín hiệu riêng biệt:

- **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:

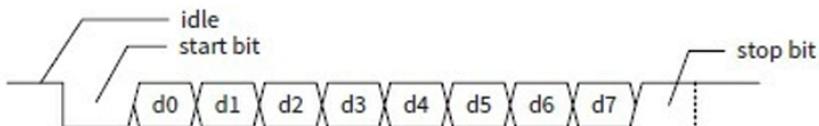
- Trạng thái nghỉ (Idle State):** Khi không có dữ liệu truyền, đường truyền luôn được giữ ở mức điện áp cao (Logic 1).
- Start Bit:** Để bắt đầu một phiên truyền, thiết bị phát sẽ kéo đường truyền từ mức cao xuống mức thấp (Logic 0) trong một chu kỳ bit. Bên thu phát hiện cạnh xuống này để bắt đầu quá trình đồng bộ.
- 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.

Start Bit ( 1 bit )	Data Frame ( 5 to 9 Data Bits )	Parity Bits ( 0 to 1 bit )	Stop Bits ( 1 to 2 bits )
------------------------	------------------------------------	-------------------------------	------------------------------

**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.

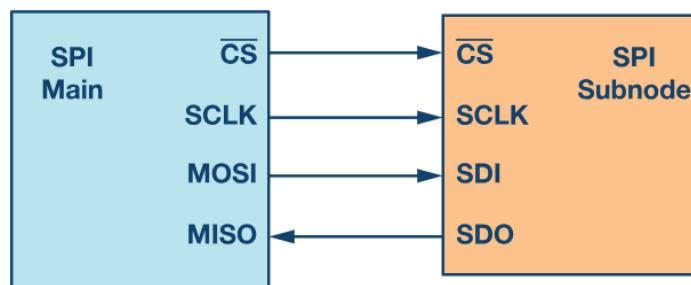
### 2.3.3 Giao thức truyền thông SPI

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

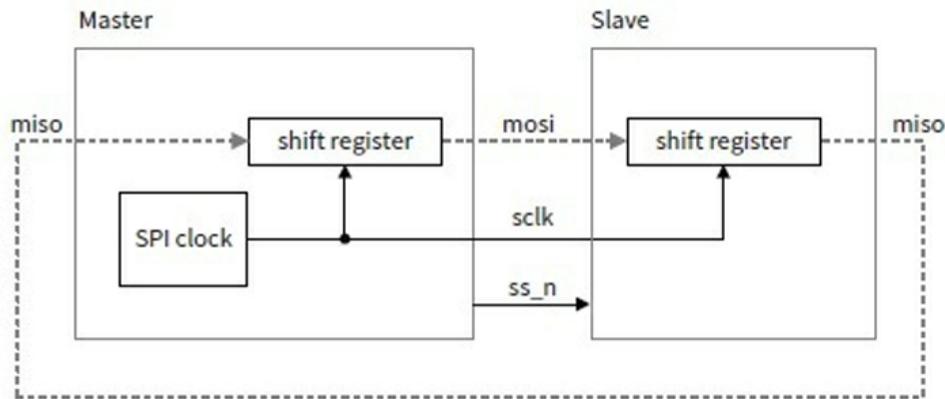
Chức năng các chân tín hiệu bao gồm:

- **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)

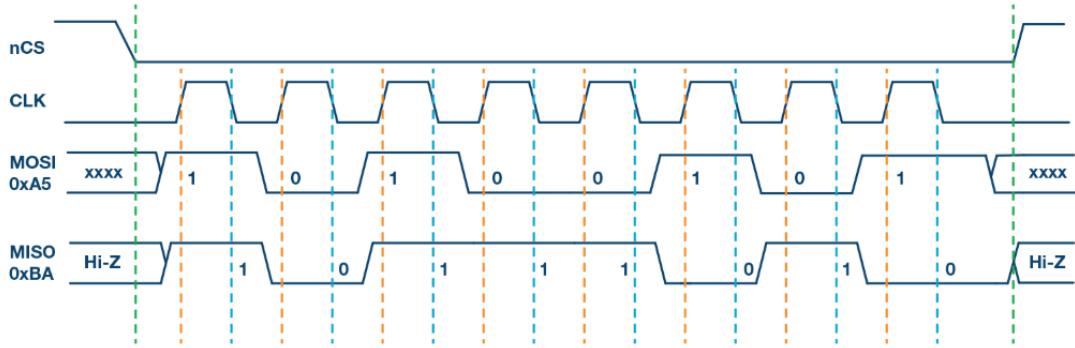
SPI định nghĩa 4 chế độ hoạt động (Modes) dựa trên trạng thái của xung Clock, được quy định bởi hai tham số:

- 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).

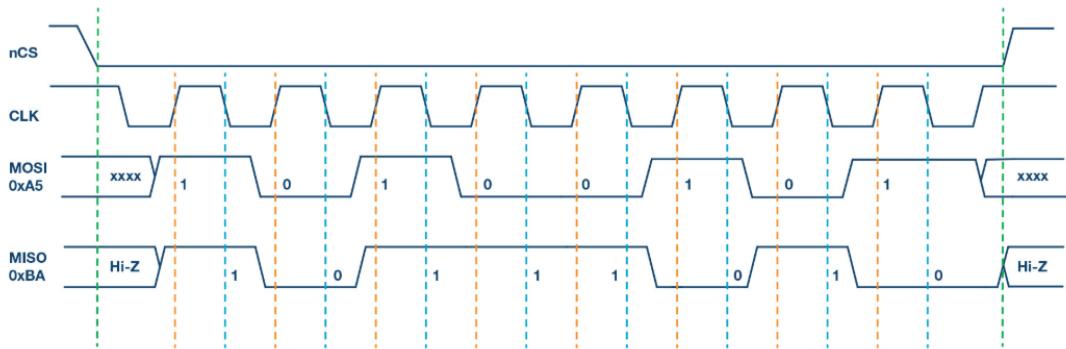
SPI Mode	CPOL	CPHA	Clock Polarity in Idle State	Clock Phase Used to Sample and/or Shift the Data
0	0	0	Logic low	Data sampled on rising edge and shifted out on the falling edge
1	0	1	Logic low	Data sampled on the falling edge and shifted out on the rising edge
2	1	0	Logic high	Data sampled on the falling 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 ( $\text{CPOL}=0$ ,  $\text{CPHA}=0$ ), trạng thái SCLK ban đầu ở mức low, dữ liệu được lấy mẫu tại cạnh lên của SCLK và dịch ở cạnh xuống



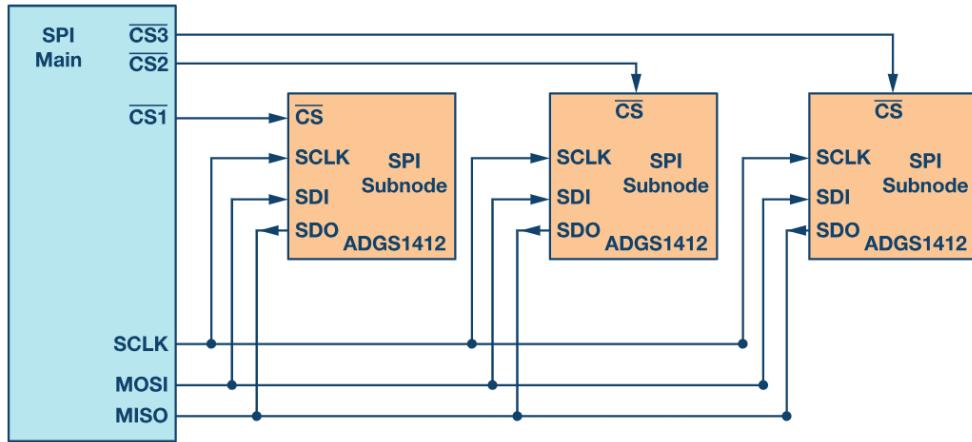
**Hình 2.18:** SPI MODE 3 ( $\text{CPOL}=1$ ,  $\text{CPHA}=1$ ), trạng thái SCLK ban đầu ở mức high, dữ liệu được lấy mẫu tại cạnh lên của SCLK và dịch ở cạnh xuống

#### 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):

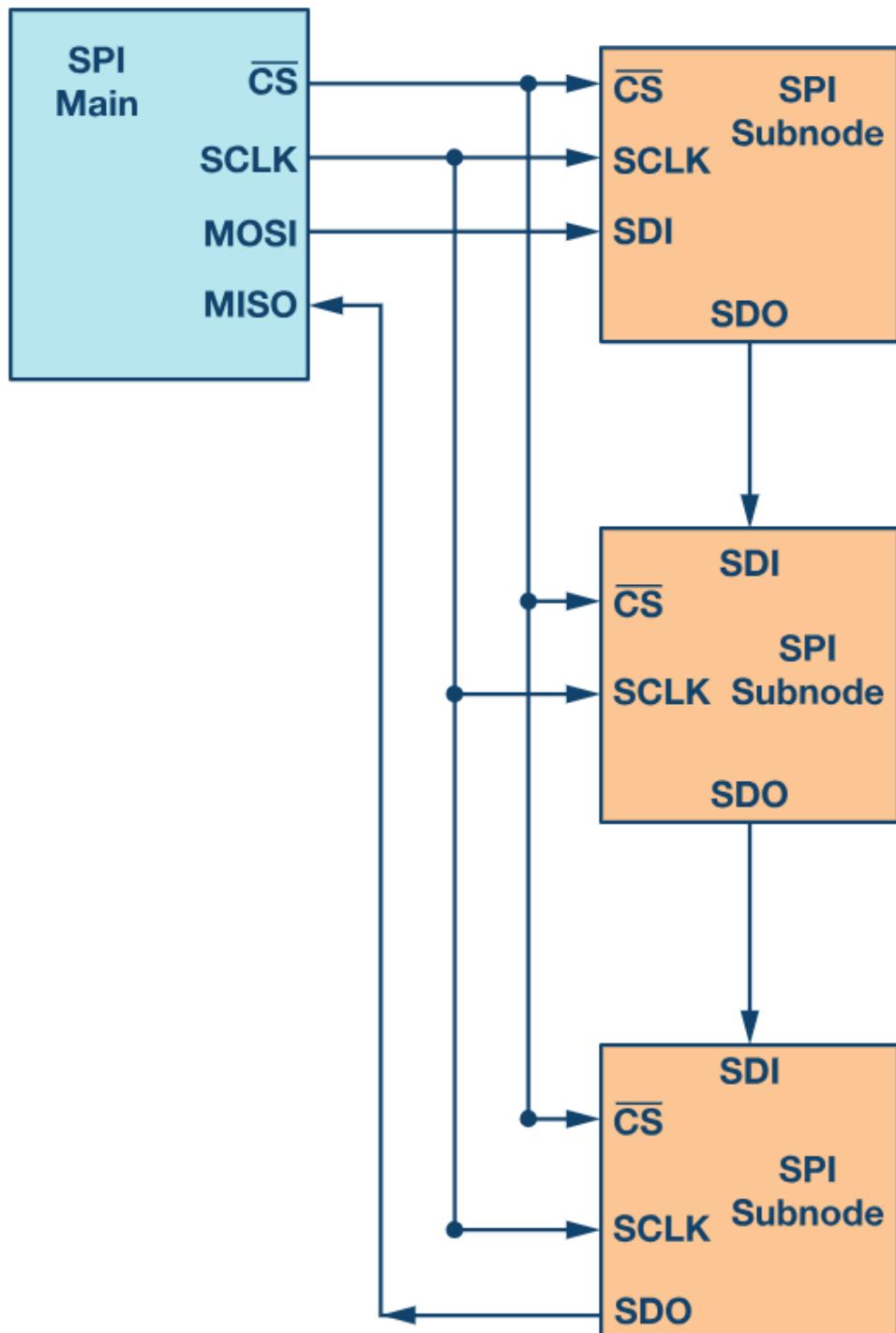
Master sử dụng các chân CS riêng biệt ( $CS_1, CS_2, \dots$ ) cho từng Slave. Đây là cấu hình phổ biến giúp tối ưu băng thông.



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.

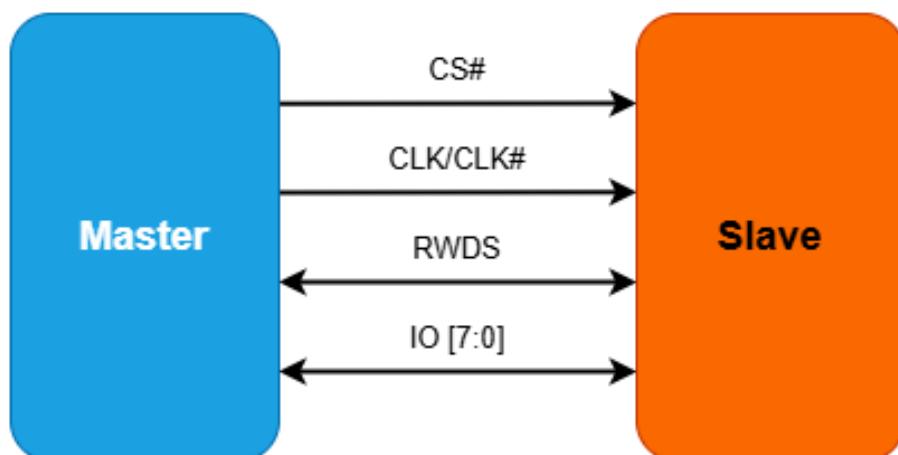
### 2.3.4 Giao thức giao tiếp OSPI (Octal SPI)

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

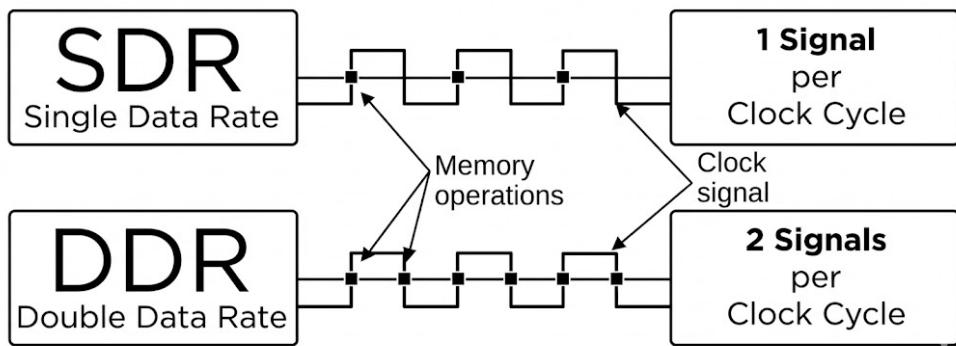
- **CLK (Serial Clock):** Tín hiệu xung nhịp đồng bộ do Master cấp.
- **CS/SS (Chip Select):** Tín hiệu chọn chip (Active Low).
- **IO0 - IO7 (Data Lines):** 8 đường dữ liệu hai chiều (Bi-directional).  
Trong một chu kỳ xung nhịp, 8 bit có thể được truyền đi đồng thời (1 Byte).
- **DQS / DS (Data Strobe):** 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 lênh (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).

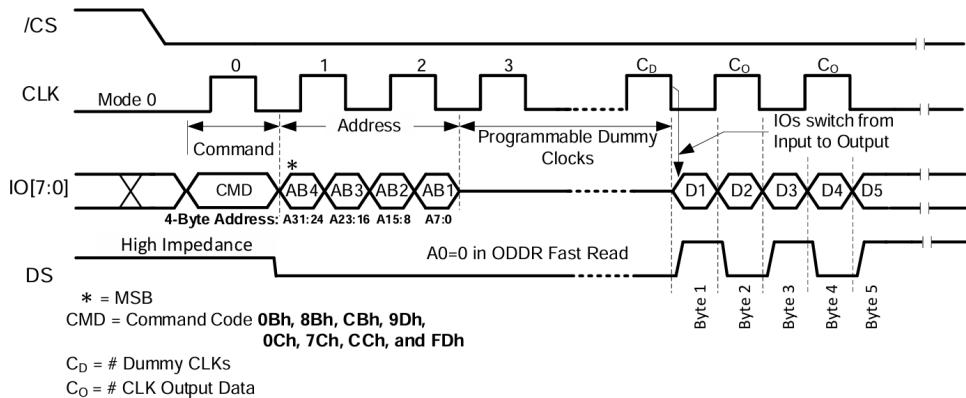
Ví dụ: Với xung nhịp 200MHz, bằng thông lý thuyết của OSPI DDR đạt:  $200 \text{ MHz} \times 2 \text{ Bytes} = 400 \text{ MB/s}$ . Tốc độ này nhanh gấp 8 lần so với QSPI thông thường (chạy SDR) và tiệm cận với các giao tiếp DRAM, đủ đáp ứng nhu cầu xử lý thời gian thực.

#### 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 FPGA.

## 2.4 Công nghệ FPGA và Quy trình thiết kế

# Chương 3

## Phân tích và Kiến trúc hệ thống

*Dựa trên cơ sở lý thuyết đã trình bày, chương này đi sâu vào phân tích các yêu cầu kỹ thuật, từ đó để 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.*

### 3.1 Phân tích yêu cầu thiết kế

### 3.2 Kiến trúc tổng thể SoC

### 3.3 Đặc tả các khối chức năng chính

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

# Chương 4

# 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).*

## 4.1 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.

### 4.1.1 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).

#### 4.1.1.1 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:

$$O[m][h][w] = B[m] + \sum_{c=0}^{C-1} \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} I[c][h \cdot U + r - P][w \cdot U + s - P] \times W[m][c][r][s] \quad (4.1)$$

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

#### 4.1.1.2 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**  $h_{in} \geq 0 \wedge h_{in} < H_{in} \wedge w_{in} \geq 0 \wedge w_{in} < W_{in}$  **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**

---

#### 4.1.2 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:

#### 4.1.2.1 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.

$$O_{dw}[c][h][w] = \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} I[c][h \cdot U + r - P][w \cdot U + s - P] \times W_{dw}[c][r][s] \quad (4.2)$$

---

#### 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

```

---

#### 4.1.2.2 Pointwise Convolution (PW)

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

$$O_{pw}[m][h][w] = \sum_{c=0}^{C-1} I[c][h][w] \times W_{pw}[m][c] \quad (4.3)$$

### 4.1.3 Tối ưu hóa: Kỹ thuật Gập Batch Normalization (BN Folding)

Để giảm độ phức tạp phần cứng, chúng tôi loại bỏ hoàn toàn khối tính toán Batch Normalization (BN) riêng biệt bằng kỹ thuật **BN Folding**.

Nguyên lý là gộp các tham số của lớp BN ( $\mu, \sigma, \gamma, \beta$ ) vào trực tiếp trọng số ( $W$ ) và bias ( $B$ ) của lớp Convolution liền trước nó. Quá trình này được thực hiện offline bởi phần mềm (driver) trước khi nạp xuống phần cứng.

Công thức biến đổi trọng số mới ( $W', B'$ ):

$$W'[m][c][r][s] = W_{orig}[m][c][r][s] \cdot \frac{\gamma_m}{\sqrt{\sigma_m^2 + \epsilon}} \quad (4.4)$$

$$B'[m] = (B_{orig}[m] - \mu_m) \cdot \frac{\gamma_m}{\sqrt{\sigma_m^2 + \epsilon}} + \beta_m \quad (4.5)$$

Nhờ đó, Accelerator chỉ cần thực hiện phép tính Convolution thuần túy mà vẫn đảm bảo độ chính xác toán học.

## 4.2 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.

#### 4.2.1 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**.

#### 4.2.2 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 4.1.

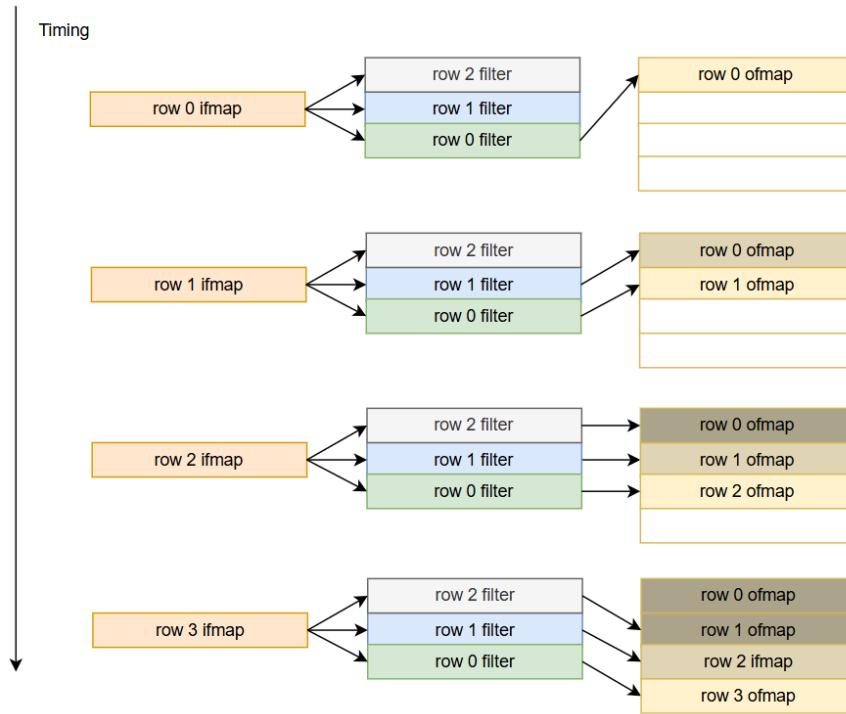
**Bảng 4.1:** Bảng tham số thiết kế và ánh xạ ký hiệu

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

#### 4.2.3 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.

#### 4.2.3.1 Phân tích Dữ liệu dôi ra (Residual Data)



**Hình 4.1:** 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:

$$T_{ho} = \begin{cases} T_h - R + 1 & \text{với Tile đầu tiên (chưa có residual)} \\ T_h & \text{với các Tile sau (nhờ cộng gộp residual)} \end{cases} \quad (4.6)$$

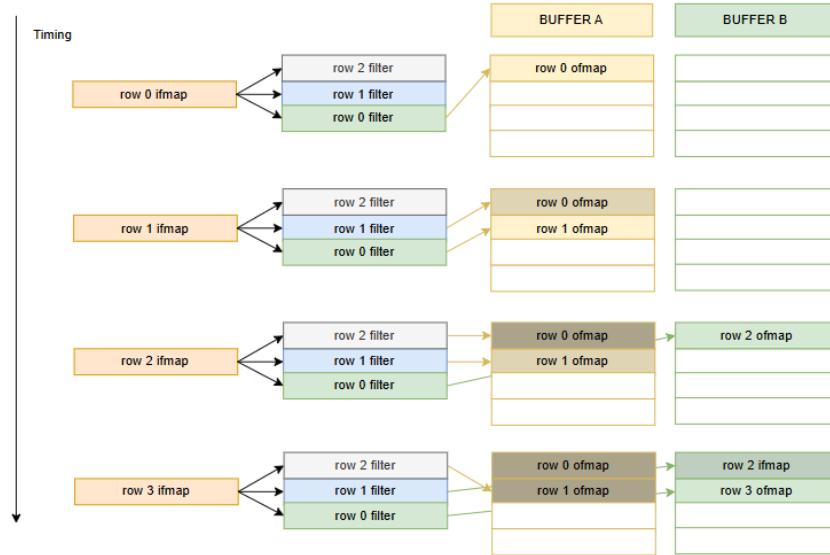
#### 4.2.3.2 Logic Hoạt động Ping-Pong

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

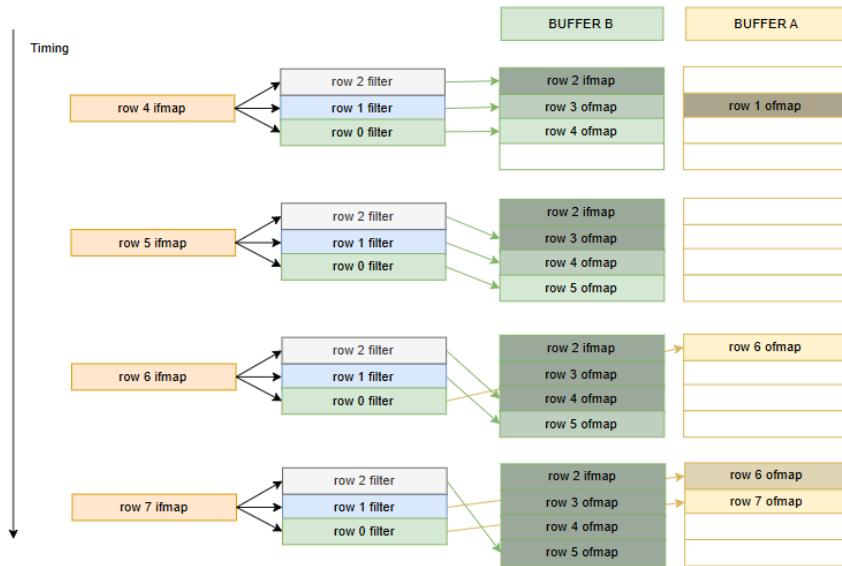
1. **Pass  $k$ :** Ghi kết quả hợp lệ vào Buffer hiện tại. Các hàng dôi ra được

ghi vào Buffer kế tiếp.

2. **Pass  $k+1$ :** Buffer kế tiếp (chứa dữ liệu dội ra cũ) trở thành Buffer hiện tại. Phép tính mới cộng dồn vào đó, hoàn thiện các hàng dội 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 4.2:** Cơ chế Ping-Pong Buffer luân phiên để quản lý vùng dữ liệu biên liên tục.

#### 4.2.4 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.

##### 4.2.4.1 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:**  $N_m, N_h, N_c$

**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**

---

Chi tiết kỹ thuật về việc quản lý bộ nhớ trong quá trình tích lũy và xử lý biên được mô tả kỹ hơn ở hai thuật toán phụ trợ sau:

---

**Algorithm 4:** Lịch trình Pass cho Standard Conv (Phần 1: Tích lũy)**Input:**  $N_m, N_h, N_c$ **for**  $m = 0$  **to**  $N_m - 1$  **do**

1. Load Weights...

**for**  $h = 0$  **to**  $N_h - 1$  **do**        **for**  $c = 0$  **to**  $N_c - 1$  **do**            Pass  $(m, h, c)$ : ... (Code phần tích lũy) ...        **end**        (*Chuyển sang xử lý biên tại Alg 5*)    **end****end**

---

**Algorithm 5:** Lịch trình Pass cho Standard Conv (Phần 2: Xử lý biên)... Tiếp tục từ vòng lặp  $h$ **foreach** Tile  $h$  đã hoàn tất tích lũy **do**

2. Xử lý biên &amp; Ghi Output:

- Ghi Valid xuống DRAM, giữ Residual lại.
- Hoán đổi Ping-Pong Buffer.

**end**

---

#### 4.2.4.2 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 6:** 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
---	--	---	--

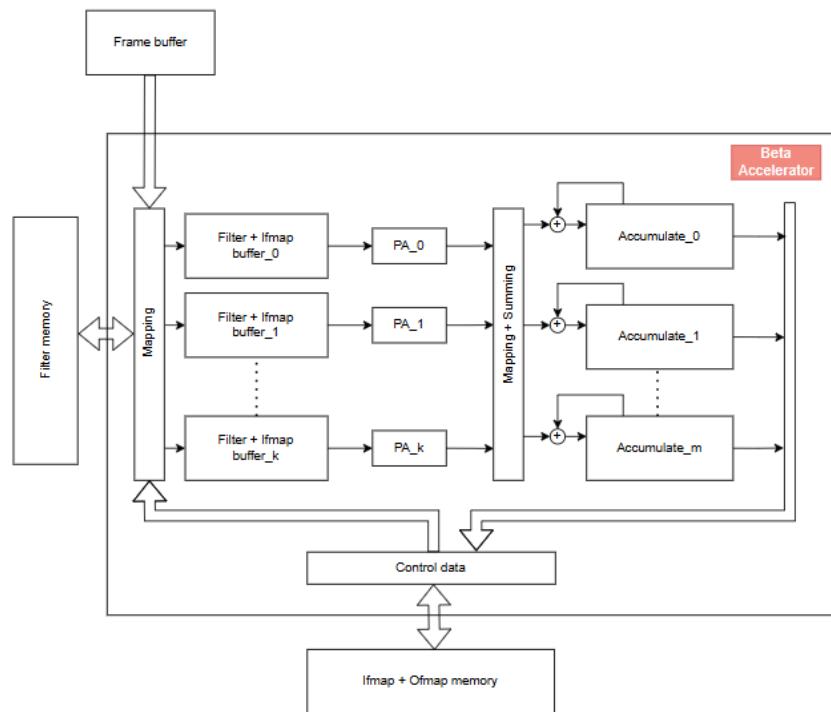
(b) Depthwise Convolution ( $H = 21, M = 21$ )

**Hình 4.3:** 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).

## 4.3 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.

### 4.3.1 Sơ đồ khái quát



**Hình 4.4:** Kiến trúc Beta Accelerator với Bus dữ liệu và Trọng số tách biệt.

Hệ thống bao gồm các thành phần chính:

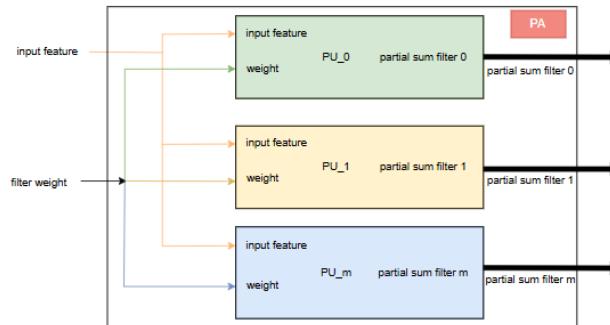
- **Controller:** Điều phối hoạt động toàn hệ thống. Quản lý hai giao tiếp bộ nhớ độc lập: *Weight Memory Interface* và *Activation Memory Interface*.
- **Dispatcher:** Phân phối dữ liệu từ Bus vào các bộ đệm cục bộ.

- **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.

### 4.3.2 Tổ chức Phân cấp Đơn vị Tính toán

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

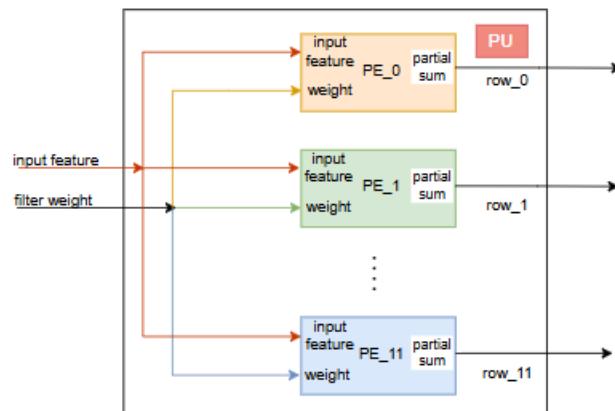
#### 4.3.2.1 Mảng xử lý (Process Array - PA)



**Hình 4.5:** Mô hình 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.

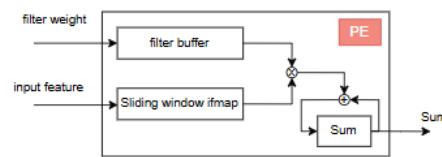
#### 4.3.2.2 Đơn vị xử lý (Process Unit - PU)



Hình 4.6: 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  $11 \times 11$  nên số PE trong PU là 11. Mỗi PE xử lý một hàng của kernel.

#### 4.3.2.3 Phần tử xử lý (Process Element - PE)



Hình 4.7: 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.

### 4.3.3 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.

#### 4.3.3.1 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:

$$T_{pass} = \left[ (W_{out} - 1) \times (S + U - 1) + S \right] \times T_h \quad (4.7)$$

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  $S$ ).

#### 4.3.3.2 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  $T_c$ .

$$T_{total\_std} = \underbrace{\left[ \frac{N_f}{T_m} \right]}_{\text{Output Blocks}} \times \underbrace{\left[ \frac{C}{T_c} \right]}_{\text{Input Blocks}} \times \underbrace{\left[ \frac{H}{T_h} \right]}_{\text{Height Blocks}} \times T_{pass} \quad (4.8)$$

## 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 ( $T_m$ ).

$$T_{total\_dw} = \underbrace{\left[ \frac{N_f}{T_m} \right]}_{\text{Channel Groups}} \times \underbrace{\left[ \frac{H}{T_h} \right]}_{\text{Height Blocks}} \times T_{pass} \quad (4.9)$$

**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.

### 4.3.4 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).

#### 4.3.4.1 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":

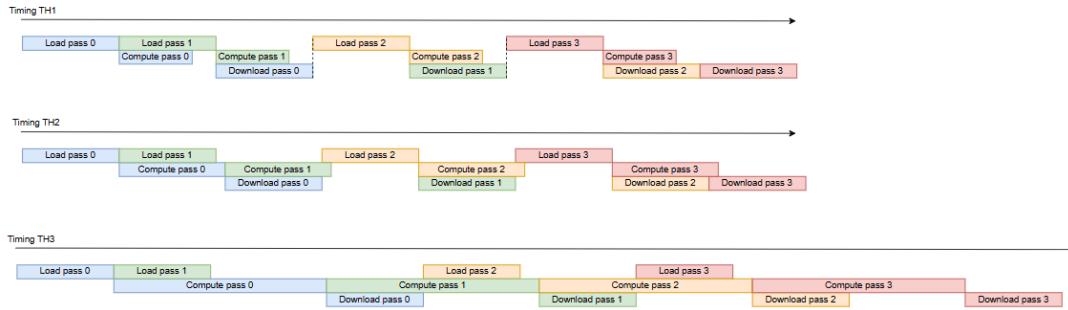
- Trong khi lõi tính toán đang xử lý Pass  $i$ , bộ điều khiển DMA đồng thời nạp dữ liệu cho Pass  $i + 1$  vào nửa còn lại của Buffer.
- Đồng thời, kết quả của Pass  $i - 1$  (nếu đã hoàn tất) được ghi trả về bộ nhớ ngoài.

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).

#### 4.3.4.2 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 ( $T_{pass}$  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 4.8.



**Hình 4.8:** 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 ( $T_{load} \geq T_{comp}$ ). Lỗi 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:

$$T_{total} \approx \left[ \left( H \times W \times C \times \left\lceil \frac{N_f}{T_m} \right\rceil \right) + (H_{out} \times W_{out} \times N_f) \right] \times b \quad (4.10)$$

- **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:

$$T_{total} \approx [(H \times W \times C) + (H_{out} \times W_{out} \times C)] \times b \quad (4.11)$$

### 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.

$$T_{total} \approx T_{total\_IO} + (T_{comp} - T_{load}) \quad (4.12)$$

Trong đó  $T_{total\_IO}$  được tính theo công thức tại Trường hợp 1 tùy thuộc loại Convolution.

### 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:

$$T_{total} = T_{load\_first\_pass} + \sum_{all\_passes} T_{comp} + T_{store\_residual} \quad (4.13)$$

- **Đối với Standard Convolution:**

$$T_{total} \approx (T_h W T_c b) + \left( \left\lceil \frac{H}{T_h} \right\rceil \left\lceil \frac{C}{T_c} \right\rceil \left\lceil \frac{N_f}{T_m} \right\rceil \times T_{comp} \right) + T_{res\_std} \quad (4.14)$$

Với  $T_{res\_std}$  là thời gian ghi phần dư cuối cùng phụ thuộc số filter dư ( $N_f\%T_m$ ):

$$T_{res\_std} = \left( \left\lceil \frac{H\%T_h}{Str} \right\rceil + 1 \right) \times W_{out} \times (N_f\%T_m) \times b \quad (4.15)$$

- **Đối với Depthwise Convolution:** Do không có vòng lặp tích lũy kênh đầu vào ( $C/T_c$ ), tổng số pass giảm đi đáng kể:

$$T_{total} \approx (T_h W T_c b) + \left( \left\lceil \frac{H}{T_h} \right\rceil \left\lceil \frac{C}{T_m} \right\rceil \times T_{comp} \right) + T_{res\_dw} \quad (4.16)$$

Với  $T_{res\_dw}$  là thời gian ghi phần dư cuối cùng phụ thuộc số kênh dư ( $C\%T_m$ ):

$$T_{res\_dw} = \left( \left\lceil \frac{H\%T_h}{Str} \right\rceil + 1 \right) \times W_{out} \times (C\%T_m) \times b \quad (4.17)$$

#### 4.3.4.3 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):

$$T_{model} = \sum_{i=1}^N T_{total}^{(i)} \quad (4.18)$$

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.

### 4.3.5 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:

Bảng 4.2: Cấu trúc Descriptor điều khiển phần cứng

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

## Chương 5

# Hiện thực SoC và Tích hợp hệ thống

*Chương này trình bày chi tiết quá trình hiện thực hệ thống SoC trên nền tảng FPGA. Nội dung bao gồm việc tích hợp lõi vi xử lý RISC-V, thiết kế các khối ngoại vi (Camera, DMA, UART), xây dựng hệ thống Bus kết nối và phát triển lớp phần mềm điều khiển (Firmware) để vận hành toàn bộ hệ thống.*

- 5.1 Môi trường và Công cụ hiện thực**
- 5.2 Tích hợp Lõi RISC-V và Hệ thống Bus**
- 5.3 Thiết kế và Tích hợp các khối Ngoại vi**
- 5.4 Phát triển Firmware và Trình điều khiển (Driver)**
- 5.5 Quy trình Tổng hợp và Triển khai trên FPGA**

# Chương 6

# Đánh giá và Thảo luận kết quả

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.

## 6.1 Môi trường và Phương pháp thực nghiệm

Do các hạn chế về thời gian tổng hợp phần cứng (Synthesis) và tài nguyên FPGA thực tế, trong phạm vi đồ án này, hiệu năng của 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 165. Phạm vi đánh giá chỉ tập trung đo đặc 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.

## 6.2 Đánh giá khả năng xử lý trên AlexNet

AlexNet là mạng nơ-ron tích chập điển hình, đặc trưng bởi việc sử dụng các bộ lọc kích thước lớn ở các lớp đầu ( $11 \times 11$ ,  $5 \times 5$ ). Kết quả mô phỏng trên cấu hình phần cứng tối ưu ( $T_k = 1, T_m = 15$ ) với giới hạn tài nguyên 168 PEs được trình bày chi tiết tại Bảng 6.1.

**Bảng 6.1:** Chi tiết hiệu năng từng lớp của AlexNet (Mô phỏng với 168 PEs)

Layer	Filter Size	PE Used	Optimized Config			Latency (ms)	Bottleneck
			$T_k$	$T_m$	$T_h$		
Conv1	$11 \times 11$	165	1	15	11	18.06	Compute
Conv2	$5 \times 5$	150	1	15	5	15.86	Compute
Conv3	$3 \times 3$	135	1	15	3	5.95	Memory
Conv4	$3 \times 3$	135	1	15	3	8.76	Memory
Conv5	$3 \times 3$	135	1	15	3	6.06	Memory
<b>Total</b>	-	<b>Max 165</b>	-	-	-	<b>54.69</b>	-

Qua bảng số liệu, có thể nhận thấy sự chuyển dịch rõ rệt của trạng thái điểm nghẽn (bottleneck) dựa trên kích thước của bộ lọc. Tại lớp đầu tiên (Conv1), hiệu suất sử dụng tài nguyên đạt mức rất cao với 165/168 PEs

được kích hoạt. Do kích thước bộ lọc lớn ( $11 \times 11$  và  $5 \times 5$ ) tại các lớp Conv1 và Conv2 làm tăng đáng kể mật độ tính toán (Arithmetic Intensity), tốc độ xử lý của PE trở thành yếu tố giới hạn (Compute Bound) thay vì băng thông bộ nhớ.

Tuy nhiên, khi chuyển sang các lớp sử dụng bộ lọc chuẩn  $3 \times 3$  (từ Conv3 đến Conv5), hiệu suất sử dụng PE giảm xuống còn 135 PEs. Nguyên nhân của hiện tượng này là do giới hạn cấu hình phần cứng đối với tham số song song hóa kênh đầu ra (Output Channel Parallelism), được cố định ở mức tối đa  $T_m^{max} = 15$ . Thực nghiệm cho thấy, với ràng buộc này, việc huy động tối đa tài nguyên (168 PEs) không mang lại sự cải thiện về tốc độ xử lý do kiến trúc đã đạt ngưỡng bão hòa về khả năng song song trên chiều  $T_m$ . Do đó, thuật toán tối ưu đã chủ động lựa chọn cấu hình sử dụng ít PE hơn (135 PEs) nhằm đảm bảo hiệu quả tài nguyên, tránh việc kích hoạt các PE dư thừa mà không đóng góp vào việc giảm độ trễ thực thi. Đồng thời, do khối lượng tính toán giảm, hệ thống chuyển sang trạng thái bị giới hạn bởi bộ nhớ (Memory Bound), khẳng định chiến lược tiết kiệm tài nguyên tính toán trong trường hợp này là hợp lý.

### 6.3 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 sâu, đồ án thực hiện mô phỏng trên VGG-16. Các thông số hiệu năng chi tiết được tổng hợp trong Bảng 6.2.

**Bảng 6.2:** Chi tiết hiệu năng từng lớp của VGG-16 (Mô phỏng với 168 PEs)

Layer	Filter / Channels	Map Size	PE Used	Optimized Config			Latency (ms)	Bottleneck
				T <sub>k</sub>	T <sub>m</sub>	T <sub>h</sub>		
Conv1_1	$3 \times 3/64$	$224 \times 224$	132	1	22	3	18.32	Memory
Conv1_2	$3 \times 3/64$	$224 \times 224$	168	1	28	4	72.27	Compute
Conv2_1	$3 \times 3/128$	$112 \times 112$	168	1	28	4	30.12	Compute
Conv2_2	$3 \times 3/128$	$112 \times 112$	168	1	28	4	60.22	Compute
Conv3_1	$3 \times 3/256$	$56 \times 56$	168	1	28	4	30.11	Compute
Conv3_2	$3 \times 3/256$	$56 \times 56$	168	1	28	4	60.21	Compute
Conv3_3	$3 \times 3/256$	$56 \times 56$	168	1	28	4	60.21	Compute
Conv4_1	$3 \times 3/512$	$28 \times 28$	168	1	28	4	28.60	Compute
Conv4_2	$3 \times 3/512$	$28 \times 28$	168	1	28	4	57.20	Compute
Conv4_3	$3 \times 3/512$	$28 \times 28$	168	1	28	4	57.20	Compute
Conv5_1	$3 \times 3/512$	$14 \times 14$	168	1	28	7	14.30	Compute
Conv5_2	$3 \times 3/512$	$14 \times 14$	168	1	28	7	14.30	Compute
Conv5_3	$3 \times 3/512$	$14 \times 14$	168	1	28	7	14.30	Compute
<b>Total</b>	-	-	<b>Max 168</b>	-	-	-	<b>517.37</b>	-

Kết quả mô phỏng cho thấy sự tối ưu đáng kể trong việc sử dụng tài nguyên. Ngoại trừ lớp đầu tiên (Conv1\_1), tất cả các lớp còn lại đều tận dụng tối đa 100% tài nguyên phần cứng (168/168 PEs). Điều này chứng minh rằng cấu hình  $T_m = 28$  là cực kỳ phù hợp với kiến trúc VGG-16, cho phép song song hóa tối đa trên chiều Output Channel.

Chính vì các đơn vị xử lý luôn hoạt động hết công suất (Full Load), tốc độ tính toán trở thành yếu tố giới hạn chính (Compute Bound) thay vì băng thông bộ nhớ. Đây là trạng thái lý tưởng cho các thiết kế bộ tốc (Accelerator), khẳng định kiến trúc bộ nhớ phân cấp (Memory Hierarchy) đã cung cấp đủ dữ liệu để duy trì hoạt động liên tục cho các nhân tính toán. Với tổng thời gian xử lý khoảng 517 ms, hệ thống đáp ứng tốt các tác vụ phân loại ảnh offline hoặc các ứng dụng không yêu cầu thời gian thực quá khắt khe.

## 6.4 Đánh giá khả năng xử lý trên MobileNet v1

Khác với VGG-16, MobileNet v1 sử dụng kiến trúc Depthwise Separable Convolution nhằm giảm khối lượng tính toán. Kết quả mô phỏng trên tập cấu hình phần cứng tối ưu ( $T_k = 1, T_m = 54$ ) với giới hạn 168 PEs được trình bày tại Bảng 6.3.

**Bảng 6.3:** Hiệu năng chi tiết từng lớp của MobileNet v1 (Total Latency: 109.88 ms)

Layer Type	Layer Idx	PE Used	Optimized Config			Latency (ms)	Bottleneck
			$T_k$	$T_m$	$T_h$		
Standard Conv	0	96	1	32	3	2.76	Memory
Depthwise	1	3	1	1	3	6.02	Compute*
Pointwise	2	96	1	32	1	8.03	Memory
Depthwise	3	3	1	1	3	7.99	Compute*
Pointwise	4	129	1	43	1	5.02	Memory
Depthwise	5	3	1	1	3	6.02	Compute*
Pointwise	6	129	1	43	1	8.03	Memory
Depthwise	7	3	1	1	3	3.98	Compute*
Pointwise	8	156	1	52	1	3.51	Memory
Depthwise	9	3	1	1	3	3.01	Compute*
Pointwise	10–20 (Even)	156	1	52	1	~ 5.5 avg	Memory
Depthwise	11–21 (Odd)	3	1	1	3	~ 1.5 avg	Compute*
Depthwise	23	3	1	1	3	0.97	Compute*
Pointwise	24	162	1	54	1	2.63	Memory
Depthwise	25	3	1	1	3	0.75	Compute*
Pointwise	26	162	1	54	1	5.02	Memory
<b>Total</b>	All	<b>Max 162</b>	-	-	-	<b>109.88</b>	Mixed

\*Ghi chú: Các lớp Depthwise hiển thị Compute Bound do hiệu suất sử dụng PE thấp

(Under-utilization), không phải do khối lượng tính toán quá lớn.

Phân tích kết quả cho thấy hệ thống có sự chuyển đổi trạng thái liên tục

giữa *Compute Bound* và *Memory Bound* (Toggle Bottleneck), phản ánh đúng đặc thù kiến trúc của MobileNet. Cụ thể, các lớp Pointwise ( $1 \times 1$  Conv) luôn rơi vào trạng thái Memory Bound do đặc tính tái sử dụng dữ liệu thấp nhưng khả năng song song hóa cao. Hệ thống đã tận dụng tới 162/168 PEs ( $T_m = 54$ ) để tiêu thụ dữ liệu nhanh hơn tốc độ cung cấp của bộ nhớ.

Ngược lại, các lớp Depthwise ( $3 \times 3$  DW) tuy được đánh dấu là Compute Bound, nhưng thực chất đây là hiện tượng giới hạn do kém hiệu quả trong sử dụng tài nguyên (under-utilization). Kiến trúc dữ liệu của Depthwise không có sự cộng gộp giữa các kênh (Channel in = Channel out), khiến cấu hình phần cứng hiện tại chỉ kích hoạt được 3 PEs ( $T_m = 1$ ). Việc 165 PEs còn lại bị nhàn rỗi đã kéo dài thời gian tính toán, biến nó thành điểm nghẽn của hệ thống. Để khắc phục vấn đề này và giảm độ trễ tổng thể (hiện tại khoảng 40% thời gian dành cho Depthwise), kiến trúc phần cứng cần được cải tiến để hỗ trợ cơ chế song song kênh (Channel Parallelism) tốt hơn cho các lớp Depthwise trong tương lai.

## 6.5 So sánh với các Nghiên cứu liên quan

Để đánh giá khách quan hiệu quả của kiến trúc đề xuất, nhóm thực hiện so sánh kết quả mô phỏng với chip gia tốc **Eyeriss** [Chen et al., ISSCC 2016]. Số liệu của Eyeriss được trích xuất từ báo cáo thực tế trên silicon với cấu hình tối ưu: Batch Size  $N = 4$  cho AlexNet và  $N = 3$  cho VGG16. Cần lưu ý rằng, để đảm bảo tính tương đồng trong so sánh, mô hình AlexNet được sử dụng trong thực nghiệm này là phiên bản có sử dụng **Grouped Convolution** (phân nhóm kênh) ở các lớp conv2, conv4 và conv5, tuân theo đúng cấu trúc mạng gốc mà Eyeriss đã tối ưu hóa.

**Bảng 6.4:** So sánh hiệu năng xử lý Convolution trên AlexNet và VGG16

Thông số	Đề xuất (Ours)	Eyeriss [Chen et al.]
Số lượng PE	168	168
Kiến trúc bộ nhớ	Separate Off-chip Memory	Shared DRAM
Chiến lược xử lý	<b>Batch Size = 1</b> (Real-time)	<b>Batch Size = 3-4</b> (Throughput)
<b>AlexNet (Latency)</b>	39.86 ms (25.1 fps)	<b>28.57 ms*</b> (35.0 fps)
<b>VGG16 (Latency)</b>	<b>517.37 ms</b> (1.93 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 **2.76 lần** so với Eyeriss (517.37 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 40ms. 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 7

# 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.*

### 7.1 Đánh giá mức độ hoàn thành Giai đoạn 1

### 7.2 Kế hoạch thực hiện Giai đoạn 2

### 7.3 Tiến độ dự kiến