

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	xvi
<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>8</b>
2.1 Kiến trúc tập lệnh RISC-V . . . . .	8
2.1.1 Tổng quan về kiến trúc RISC-V . . . . .	8
2.1.2 Mô hình lập trình và Tập thanh ghi . . . . .	9
2.1.2.1 Bộ đếm chương trình (Program Counter - PC) . . . . .	9
2.1.2.2 Tập thanh ghi mục đích chung (General Purpose Registers) . . . . .	10
2.1.3 Đặc tả tập lệnh cơ sở RV32I . . . . .	11
2.1.3.1 Định dạng lệnh (Instruction Formats) .	11
2.1.3.2 Phân nhóm chức năng chi tiết . . . . .	12
2.1.4 Vị xử lý PicoRV32 . . . . .	14

2.2	Tổng quan về Mạng nơ-ron tích chập (CNN) . . . . .	14
2.2.1	Trí tuệ nhân tạo và Học sâu . . . . .	14
2.2.2	Xu hướng chuyển dịch tính toán xuồng biển (Edge AI) . . . . .	15
2.2.3	Cơ sở toán học của Mạng Nơ-ron Tích chập (CNN)	16
2.2.3.1	Standard Convolution (Tích chập tiêu chuẩn)	16
2.2.3.2	Depthwise Separable Convolution . . . . .	17
2.2.4	Kỹ thuật Gập Batch Normalization (BN Folding)	17
2.3	Các chuẩn giao tiếp hệ thống . . . . .	18
2.3.1	Chuẩn giao tiếp AMBA AXI4 . . . . .	18
2.3.1.1	Kiến trúc 5 kênh độc lập (Channel Architecture) . . . . .	20
2.3.1.2	Cơ chế bắt tay (Handshake Mechanism)	21
2.3.1.3	Quy trình thực hiện giao dịch chi tiết (Transaction Steps) . . . . .	23
2.3.1.4	Cấu trúc giao dịch Burst (Burst Transaction) . . . . .	25
2.3.1.5	Các biến thể giao thức trong thiết kế . .	25
2.3.1.6	Áp dụng trong hệ thống đè tài . . . . .	26
2.3.2	Giao thức truyền thông UART . . . . .	27
2.3.2.1	Nguyên lý hoạt động . . . . .	27
2.3.2.2	Cấu trúc khung dữ liệu (Data Frame) .	28
2.3.2.3	Tốc độ Baud (Baud Rate) . . . . .	29
2.3.3	Giao thức truyền thông SPI . . . . .	30
2.3.3.1	Cấu hình tín hiệu vật lý . . . . .	30
2.3.3.2	Cơ chế hoạt động: Thanh ghi dịch (Shift Register) . . . . .	31
2.3.3.3	Các chế độ hoạt động (Clock Polarity & Phase) . . . . .	32

2.3.3.4	Các mô hình kết nối đa thiết bị . . . . .	33
2.3.4	Giao thức truyền thông OSPI (Octal SPI) . . . . .	36
2.3.4.1	Cấu hình tín hiệu vật lý . . . . .	36
2.3.4.2	Cơ chế truyền tải DDR (Double Data Rate)	37
2.3.4.3	Cấu trúc giao dịch Octal-DDR . . . . .	38
2.3.4.4	Ưu điểm trong ứng dụng SoC IoT . . . . .	39
2.3.5	Giao thức truyền thông I2C (Inter-Integrated Circuit) . . . . .	40
2.3.5.1	Cấu hình vật lý và Nguyên lý Open-Drain	40
2.3.5.2	Giao thức truyền dữ liệu . . . . .	42
2.3.5.3	Các tốc độ hoạt động . . . . .	43
2.3.5.4	Đánh giá ưu nhược điểm . . . . .	44
2.3.6	Giao diện Camera song song (DVP Interface) . .	44
2.3.6.1	Đặc tả tín hiệu và Cơ chế vật lý . . . . .	44
2.3.6.2	Định dạng dữ liệu RGB565 trên bus 8-bit	45
2.3.6.3	Quy trình thu thập khung ảnh (Frame Capture Sequence) . . . . .	46
2.3.7	Giao diện hiển thị HDMI (High-Definition Multi-media Interface) . . . . .	47
2.3.7.1	Kiến trúc phần cứng hiển thị . . . . .	47
2.3.7.2	Công nghệ truyền dẫn TMDS . . . . .	49
2.3.7.3	Cấu hình hoạt động qua I2C . . . . .	49
2.4	Công nghệ FPGA và Quy trình thiết kế . . . . .	50
2.4.1	Tổng quan về công nghệ FPGA . . . . .	50
2.4.2	Kiến trúc phần cứng Xilinx 7-Series . . . . .	50
2.4.2.1	Configurable Logic Block (CLB) . . . . .	51
2.4.2.2	Bộ nhớ nội BRAM (Block RAM) . . . . .	51
2.4.3	Nền tảng phần cứng thực nghiệm . . . . .	51

2.4.3.1	Giai đoạn 1: Thủ nghiệm trên Digilent Arty A7 (Artix-7) . . . . .	51
2.4.3.2	Giai đoạn 2: Triển khai trên Xilinx VC707 (Virtex-7) . . . . .	52
2.4.4	Quy trình thiết kế trên Vivado . . . . .	53
<b>3</b>	<b>Công trình nghiên cứu liên quan kiến trúc bộ gia tốc CNN</b>	<b>55</b>
3.1	Kiến trúc tham chiếu: Hệ thống Eyeriss . . . . .	55
3.2	Kiến trúc tham chiếu: Bộ tăng tốc Pixel-Level Fully Pipelined	56
3.3	Kiến trúc tham chiếu: Tăng tốc CNN trên FPGA dựa trên OpenCL . . . . .	57
3.4	Kiến trúc tham chiếu: Hệ thống xử lý dị thể trên nền tảng RISC-V cho IoT . . . . .	57
3.5	Kiến trúc tham chiếu: Bộ tăng tốc luồng cấu hình lại (RSA) cho IoT . . . . .	58
<b>4</b>	<b>Phân tích và Kiến trúc hệ thống</b>	<b>60</b>
4.1	Phân tích yêu cầu thiết kế . . . . .	60
4.1.1	Yêu cầu chức năng . . . . .	60
4.1.2	Yêu cầu phi chức năng . . . . .	62
4.2	Kiến trúc tổng thể SoC . . . . .	63
4.2.1	Tổng quan kiến trúc SoC . . . . .	63
4.2.2	Tổ chức hệ thống Bus phân tầng . . . . .	65
4.3	Đặc tả các khối chức năng chính . . . . .	66
4.3.1	Vi xử lý trung tâm (Central Processing Unit) . .	66
4.3.2	Nhận diện Hình ảnh (Image Detector) . . . . .	67
4.3.2.1	Hệ thống Video Streaming . . . . .	67
4.3.2.2	Khối Gia tốc (Accelerator) . . . . .	67
4.3.3	Các Ngoại vi (Peripherals) . . . . .	68

4.4	Tổ chức bộ nhớ và Bản đồ địa chỉ (Memory Map) . . . . .	68
4.4.1	Khái niệm và vai trò của Memory Map . . . . .	68
4.4.2	Bản đồ vùng nhớ hệ thống . . . . .	70
4.4.3	Bản đồ vùng ngoại vi . . . . .	70
<b>5</b>	<b>Thiết kế Bộ tăng tốc AI (AI Accelerator)</b>	<b>72</b>
5.1	Cơ sở Toán học và Thách thức Thiết kế . . . . .	72
5.1.1	Standard Convolution (Tích chập tiêu chuẩn) . .	73
5.1.1.1	Mô hình toán học . . . . .	73
5.1.1.2	Thuật toán xử lý . . . . .	73
5.1.2	Depthwise Separable Convolution . . . . .	74
5.1.2.1	Depthwise Convolution (DW) . . . . .	75
5.1.2.2	Pointwise Convolution (PW) . . . . .	75
5.2	Chiến lược Phân mảnh và Quản lý Dòng dữ liệu . . . . .	76
5.2.1	Chiến lược Phân mảnh không gian (Space Partitioning) . . . . .	76
5.2.2	Mô hình hóa và Tham số thiết kế . . . . .	77
5.2.3	Bài toán Dữ liệu biên và Cơ chế Ping-Pong . . .	77
5.2.3.1	Phân tích Dữ liệu dội ra (Residual Data) . . . . .	78
5.2.3.2	Logic Hoạt động Ping-Pong . . . . .	78
5.2.4	Thuật toán Điều phối Pass (Pass Scheduling) . .	80
5.2.4.1	Trường hợp Standard Convolution . . . . .	80
5.2.4.2	Trường hợp Depthwise Convolution . . . . .	80
5.3	Thiết kế Kiến trúc Vi mô (Micro-architecture) . . . . .	82
5.3.1	Sơ đồ khối tổng quát . . . . .	82
5.3.2	Tổ chức Phân cấp Đơn vị Tính toán . . . . .	83
5.3.2.1	Mảng xử lý (Process Array - PA) . . . . .	83
5.3.2.2	Đơn vị xử lý (Process Unit - PU) . . . . .	84
5.3.2.3	Phần tử xử lý (Process Element - PE) .	84

5.3.3	Đánh giá thời gian thực thi (Performance Estimation) . . . . .	85
5.3.3.1	Thời gian xử lý một Pass cơ sở ( $T_{pass}$ ) . . . . .	85
5.3.3.2	Tổng thời gian thực thi ( $T_{total}$ ) . . . . .	85
5.3.4	Mô hình hóa độ trễ toàn hệ thống . . . . .	86
5.3.4.1	Cơ chế hoạt động và tối ưu hóa tham số . . . . .	86
5.3.4.2	Các kịch bản hiệu năng (Performance Scenarios) . . . . .	87
5.3.4.3	Tổng thời gian toàn mạng (Model Latency) . . . . .	89
5.3.5	Tự động sinh mã cấu hình (Auto-Generation) . . . . .	90
<b>6</b>	<b>Hiện thực SoC và Tích hợp hệ thống</b>	<b>91</b>
6.1	Môi trường và Công cụ hiện thực . . . . .	91
6.1.1	Môi trường thiết kế và kiểm thử phần cứng . . . . .	91
6.1.2	Môi trường phát triển phần mềm và quy trình nhúng mã . . . . .	92
6.2	Tích hợp Lõi RISC-V . . . . .	93
6.2.1	Khái quát về lõi vi xử lý PicoRV32 . . . . .	93
6.2.2	Phân tích cấu hình và tùy chọn tập lệnh . . . . .	94
6.2.3	Thiết lập ngữ cảnh thực thi và kết nối hệ thống . . . . .	95
6.2.3.1	Chi tiết thiết lập tham số phần cứng (Parameters) . . . . .	95
6.3	Thiết kế hệ thống Bus . . . . .	97
6.3.1	Mô hình điều phối đơn Master - đa Slave ( $1 \times N$ )	97
6.3.2	Mô hình trọng tài đa Master - đơn Slave ( $N \times 1$ )	99
6.3.3	Mô hình phức hợp đa Master - đa Slave ( $N \times M$ )	100
6.3.4	Kiến trúc tổng thể và cơ chế cấu hình linh hoạt . . . . .	102
6.3.5	Cơ chế điều phối giao dịch và giải mã địa chỉ . . . . .	102
6.3.6	Logic trọng tài Round-Robin và quản lý truy cập	103

6.3.7	Hệ thống an toàn và tầng giao diện Slave . . . . .	105
6.4	Thiết kế và Tích hợp các khối Ngoại vi . . . . .	106
6.4.1	UART . . . . .	106
6.4.1.1	Kiến trúc hệ thống và bộ tạo tốc độ Baud	106
6.4.1.2	Hiện thực máy trạng thái cho bộ nhận (RX) và bộ truyền (TX) . . . . .	108
6.4.1.3	Cơ chế đệm dữ liệu và quản lý dòng thông tin . . . . .	110
6.4.1.4	Tích hợp giao diện AXI4-Lite . . . . .	110
6.4.2	SPI . . . . .	111
6.4.2.1	Cấu trúc logic và bộ tạo xung nhịp SPI	111
6.4.2.2	Cơ chế điều khiển và Máy trạng thái FSMD	112
6.4.2.3	Tích hợp hệ thống qua giao diện AXI4-Lite	115
6.4.3	I2C . . . . .	116
6.4.3.1	Kiến trúc bộ điều khiển I2C Master . .	116
6.4.3.2	Hiện thực Máy trạng thái điều khiển (Con- trol FSM) . . . . .	117
6.4.3.3	Ánh xạ thanh ghi và tích hợp AXI4-Lite	120
6.4.4	OSPI-DDR . . . . .	121
6.4.4.1	Ý tưởng thiết kế . . . . .	121
6.4.4.2	Tích hợp hệ thống và giao diện AXI4-Lite	122
6.5	Video Streaming . . . . .	123
6.5.1	Kiến trúc tổng thể của Video Pipeline . . . . .	123
6.5.2	Khối thu nhận hình ảnh từ cảm biến OV5640 (DVP Interface) . . . . .	124
6.5.3	Cơ chế quản lý bộ đệm khung hình (Frame Buffer)	125
6.5.4	Điều khiển hiển thị và Giao diện HDMI (ADV7513)	125

7.1	Môi trường và Phương pháp thực nghiệm . . . . .	127
7.2	Dánh giá khả năng xử lý trên AlexNet . . . . .	128
7.3	Dánh giá khả năng xử lý trên VGG-16 . . . . .	130
7.4	Dánh giá khả năng xử lý trên MobileNet v1 . . . . .	132
7.5	So sánh với các Nghiên cứu liên quan . . . . .	134
<b>8</b>	<b>Kết quả thực nghiệm</b>	<b>136</b>
8.1	Môi trường thực nghiệm phần cứng . . . . .	136
8.2	Kết quả hiện thực các khối giao tiếp ngoại vi . . . . .	137
8.2.1	Giao tiếp UART . . . . .	137
8.2.2	Giao tiếp I2C và SPI . . . . .	139
8.2.3	Giao tiếp Octal-SPI (OSPI) hỗ trợ DDR . . . . .	140
8.3	Kết quả Video Streaming 60Hz . . . . .	142
8.4	Hiện thực chương trình Bootloader qua SPI Flash . . . . .	145
8.5	Dánh giá tài nguyên sử dụng trên FPGA . . . . .	145
<b>9</b>	<b>Kết luận và Hướng phát triển</b>	<b>148</b>
9.1	Dánh giá Giai đoạn 1 . . . . .	148
9.2	Kế hoạch thực hiện Giai đoạn 2 . . . . .	149
9.3	Tiến độ dự kiến . . . . .	150
<b>Tài liệu tham khảo</b>		<b>151</b>

# Danh sách hình vẽ

Figure 2.1 Cấu trúc bit của các định dạng lệnh RV32I . . . . .	11
Figure 2.2 a. Tổng quan giao thức AXI4 . . . . .	19
Figure 2.3 b. Tổng quan giao thức AXI4 . . . . .	19
Figure 2.4 Mô hình 5 kênh giao tiếp của AXI4 . . . . .	20
Figure 2.5 Cơ chế bắt tay VALID/READY trong AXI . . . . .	21
Figure 2.6 Minh họa một Transfer trong AXI . . . . .	22
Figure 2.7 Minh họa một Transaction trong AXI . . . . .	22
Figure 2.8 Giản đồ tín hiệu chi tiết của giao dịch Ghi . . . . .	23
Figure 2.9 Giản đồ tín hiệu chi tiết của giao dịch Đọc . . . . .	24
Figure 2.10 Minh họa chân kết nối truyền nhận dữ liệu UART . . .	28
Figure 2.11 Chuyển đổi dữ liệu song song thành nối tiếp và ngược lại trong UART . . . . .	28
Figure 2.12 Khung dữ liệu UART . . . . .	29
Figure 2.13 Ví dụ khung dữ liệu UART với 8bit dữ liệu, không parity và 1 stop bit . . . . .	29
Figure 2.14 Sơ đồ kết nối tín hiệu chuẩn 4 dây của SPI . . . . .	30
Figure 2.15 Cơ chế trao đổi dữ liệu dùng thanh ghi dịch trong SPI .	31

Figure 2.16 4 chế độ hoạt động của SPI(CPOL/CPHA) . . . . .	32
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 . . . . .	33
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 . . . . .	33
Figure 2.19 Cấu hình Slave độc lập trong SPI . . . . .	34
Figure 2.20 Cấu hình Chuỗi (Daisy Chain) trong SPI . . . . .	35
Figure 2.21 Sơ đồ chân tín hiệu của giao diện OSPI/HyperBus . . .	36
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 . . . . .	38
Figure 2.23 Giản đồ thời gian giao dịch OSPI DDR: Command, Ad- dress và Data truyền trên 8 dây IO . . . . .	39
Figure 2.24 Sơ đồ kết nối vật lý I2C . . . . .	40
Figure 2.25 a. Điện trở kéo lên (Pull-up Resistors) . . . . .	41
Figure 2.26 b. Điện trở kéo lên (Pull-up Resistors) . . . . .	41
Figure 2.27 Cấu trúc khung truyền dữ liệu I2C . . . . .	42
Figure 2.28 Giản đồ thời gian của điều kiện Start và Stop trong I2C	42
Figure 2.29 Cấu trúc một khung truyền dữ liệu I2C cơ bản . . . .	43
Figure 2.30 Sơ đồ kết nối tín hiệu vật lý giữa Camera DVP và FPGA	45
Figure 2.31 Giản đồ thời gian và trạng thái thu thập khung ảnh . .	46

Figure 2.32 Sơ đồ kết nối tín hiệu giữa FPGA và ADV7513 . . . . .	48
Figure 2.33 FPGA Arty A7-100T . . . . .	52
Figure 2.34 FPGA Xilinx VC707 . . . . .	53
Figure 4.1 Sơ đồ mô-đun kiến trúc tổng thể của hệ thống SoC RISC-V EdgeAI . . . . .	63
Figure 5.1 Minh họa sự hình thành dữ liệu dội ra. Tại hàng 2 và 3, bộ lọc thiếu dữ liệu từ hàng 4, 5 (thuộc tile sau) nên kết quả chưa hoàn thiện. . . . .	78
Figure 5.2 Cơ chế Ping-Pong Buffer luân phiên để quản lý vùng dữ liệu biên liên tục. . . . .	79
Figure 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). . . . .	81
Figure 5.4 Kiến trúc Beta Accelerator với Bus dữ liệu và Trọng số tách biệt. . . . .	82
Figure 5.5 Mỗi PA xử lý 1 kênh Input và tạo ra kết quả cho $T_m$ kênh Output. . . . .	83
Figure 5.6 Khối PU chứa 11 PE trong trường hợp chạy model AlexNet.	84
Figure 5.7 PE thực hiện phép MAC với cơ chế Weight Stationary.	84
Figure 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. . . . .	88
Figure 6.1 a. Sơ đồ khối mô hình kết nối AXI4-Lite $1 \times N$ . . . . .	98

Figure 6.2	b. Sơ đồ khối mô hình kết nối AXI4-Lite $1 \times N$	98
Figure 6.3	a. Sơ đồ khối cơ chế trọng tài trong mô hình $N \times 1$	99
Figure 6.4	b. Sơ đồ khối cơ chế trọng tài trong mô hình $N \times 1$	100
Figure 6.5	a. Kiến trúc kết hợp Cascaded Interconnect cho mô hình đa Master - đa Slave	101
Figure 6.6	b. Kiến trúc kết hợp Cascaded Interconnect cho mô hình đa Master - đa Slave	101
Figure 6.7	a. Sơ đồ khối cơ chế trọng tài Round-Robin	104
Figure 6.8	b. Sơ đồ khối cơ chế trọng tài Round-Robin	105
Figure 6.9	Sơ đồ khối kiến trúc module UART Unit	107
Figure 6.10	Sơ đồ máy trạng thái (FSM) của bộ nhận UART RX	109
Figure 6.11	Các thanh ghi kết nối giữa Bus AXI4-Lite và ngoại vi UART	110
Figure 6.12	SPI Mode	112
Figure 6.13	Thời gian thực hiện thao tác dịch dữ liệu trong SPI	113
Figure 6.14	SPI controller ASMD chart	114
Figure 6.15	Các thanh ghi kết nối giữa Bus AXI4-Lite và ngoại vi SPI	116
Figure 6.16	Chia các giai đoạn điều kiện trong chu kỳ xung nhịp I2C	117
Figure 6.17	Chia các giai đoạn dữ liệu trong chu kỳ xung nhịp I2C	117
Figure 6.18	Sơ đồ các trạng thái của bộ điều khiển I2C Master	118
Figure 6.19	Các thanh ghi kết nối giữa Bus AXI4-Lite và ngoại vi I2C	120
Figure 6.20	Các thanh ghi kết nối giữa Bus AXI4-Lite và ngoại vi OSPI	122

Figure 6.21 Sơ đồ khối Video Streaming tích hợp trong SoC . . . . .	124
Figure 7.1 Biểu đồ tương quan giữa số lượng PE và độ trễ xử lý trên AlexNet . . . . .	128
Figure 7.2 Biểu đồ tương quan giữa số lượng PE và độ trễ xử lý trên VGG-16 . . . . .	130
Figure 7.3 Biểu đồ tương quan giữa số lượng PE và độ trễ xử lý trên MobileNet v1 . . . . .	132
Figure 8.1 Kết nối UART từ Arty A7 với máy tính . . . . .	137
Figure 8.2 Chương trình kiểm thử giao tiếp UART . . . . .	138
Figure 8.3 Cài đặt thông số PuTTY để giao tiếp UART . . . . .	138
Figure 8.4 Kết quả giao tiếp UART thành công . . . . .	139
Figure 8.5 Kết nối SPI từ Arty A7 với bộ nhớ Flash . . . . .	140
Figure 8.6 Sơ đồ sóng được đo thực tế bằng Logic Analyzer giữa SPI từ Arty A7 với bộ nhớ Flash . . . . .	140
Figure 8.7 Kết nối chip nhớ HyperRAM (màu xanh lá) với Arty A7 qua giao tiếp OSPI . . . . .	141
Figure 8.8 Chương trình cho việc ghi và đọc dữ liệu qua giao tiếp OSPI và xuất ra Terminal UART . . . . .	142
Figure 8.9 Phần cứng Video Streaming trên VC707 hiển thị hình ảnh từ camera OV5640 ra màn hình qua cổng HDMI .	143
Figure 8.10 Kiểm tra Màn hình HDMI ổn định với tần số 60Hz .	143
Figure 8.11 Kiểm tra Patten Bar từ camera OV5640 hiển thị qua HDMI . . . . .	144

Figure 8.12 Chế độ Streaming Video thời gian thực từ camera OV5640 qua HDMI . . . . .	144
Figure 8.13 a. Mức độ sử dụng tài nguyên FPGA riêng phần Video Streaming trên VC707 . . . . .	145
Figure 8.14 b. Mức độ sử dụng tài nguyên FPGA riêng phần Video Streaming trên VC707 . . . . .	146
Figure 8.15 c. Mức độ sử dụng tài nguyên FPGA riêng phần SoC trên VC707 . . . . .	146
Figure 8.16 d. Mức độ sử dụng tài nguyên FPGA riêng phần SoC trên VC707 . . . . .	147

# 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)	10
Table 2.2	Cấu trúc truyền tải Pixel RGB565 qua giao diện 8-bit	46
Table 2.3	So sánh tài nguyên giữa Arty A7 (Thử nghiệm ban đầu) và VC707 (Triển khai chính thức) . . . . .	53
Table 4.1	Bản đồ địa chỉ vùng nhớ hệ thống (System Memory Map)	70
Table 4.2	Bản đồ địa chỉ vùng ngoại vi (Peripheral Memory Map)	71
Table 5.1	Bảng tham số thiết kế và ánh xạ ký hiệu . . . . .	77
Table 5.2	Cấu trúc Descriptor điều khiển phần cứng . . . . .	90
Table 6.1	Cấu hình tham số phần cứng cho lõi PicoRV32 . .	96
Table 7.1	Chi tiết hiệu năng từng lớp của AlexNet (Cập nhật)	129
Table 7.2	Chi tiết hiệu năng từng lớp của VGG-16 . . . . .	131
Table 7.3	Chi tiết hiệu năng từng lớp của MobileNet v1 . . .	133
Table 7.4	So sánh hiệu năng xử lý Convolution trên AlexNet và VGG16 . . . . .	134
Table 9.1	Bảng tiến độ thực hiện Giai đoạn 2 . . . . .	150

# 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 MobileNetv1) 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: Standard Convolution, Depthwise Convolution, Pointwise Convolution.</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 - Công trình nghiên cứu liên quan kiến trúc bộ nhớ CNN:** Tổng quan các nghiên cứu trước đây về thiết kế bộ nhớ CNN.

**Chương 4 - 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 5 - 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 6 - 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 7 - Ước lượng hiệu năng:** Phân tích và đánh giá hiệu năng của hệ thống thông qua mô hình ước lượng, bao gồm độ trễ và tài nguyên sử dụng.

**Chương 8 - Kết quả thực nghiệm:** Trình bày các kết quả đạt được sau quá trình thiết kế, tổng hợp và triển khai hệ thống SoC trên nền tảng phần cứng thực tế.

**Chương 9 - 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)

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 tinh, 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 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:

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

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  $1 \times 1$ , 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:

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

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.

## 2.3 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.

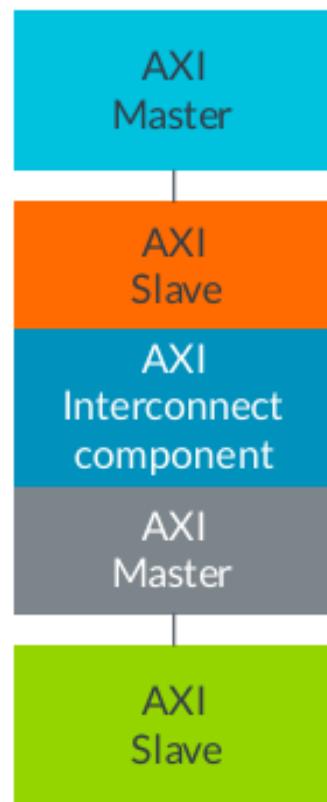
Phiên bản AXI4 (được giới thiệu trong AMBA 4.0) hỗ trợ các tính năng vượt trội so với các thế hệ trước:

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

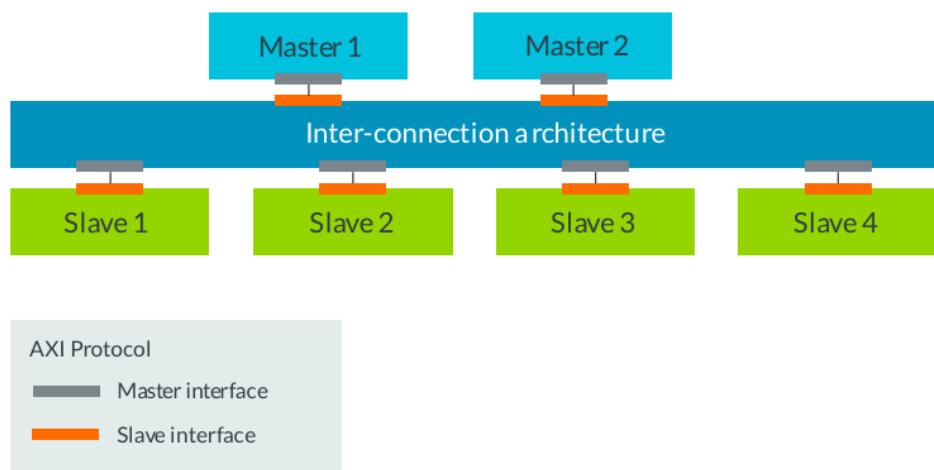
Hỗ trợ giao dịch dữ liệu không thẳng hàng (Unaligned data transfers).

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.



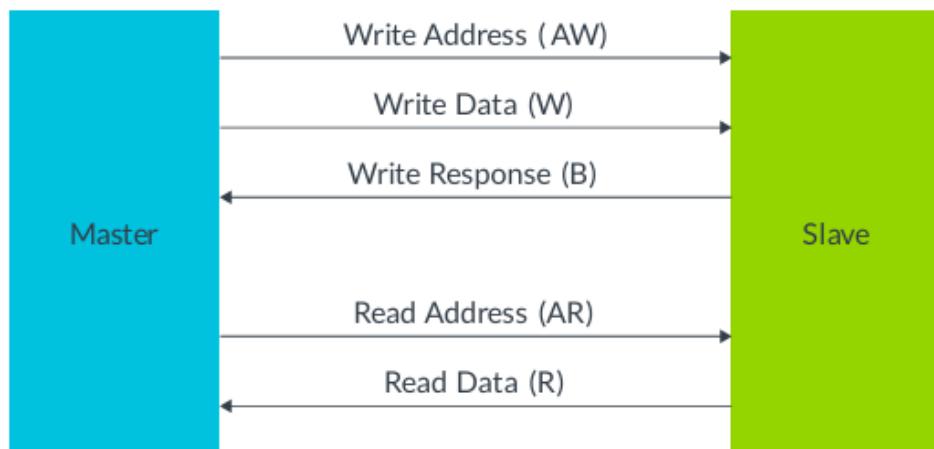
**Hình 2.2: a.** Tổng quan giao thức AXI4



**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.

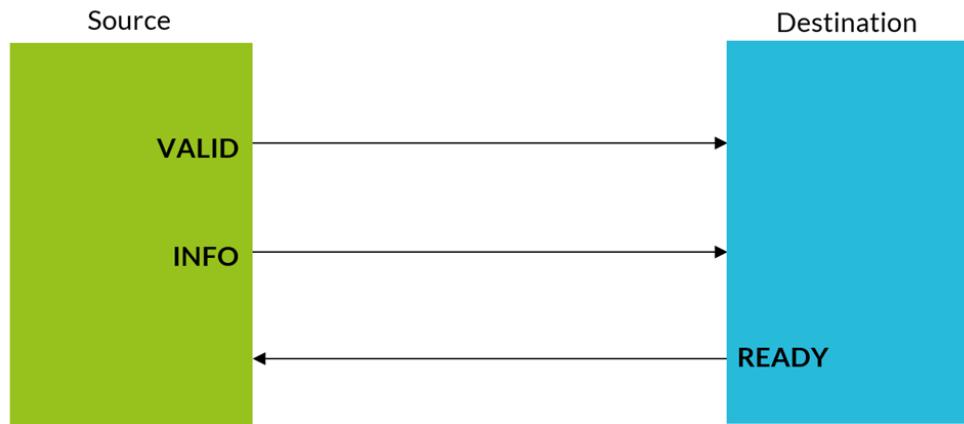


**Hình 2.5:** Cơ chế bắt tay VALID/READY trong AXI

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.

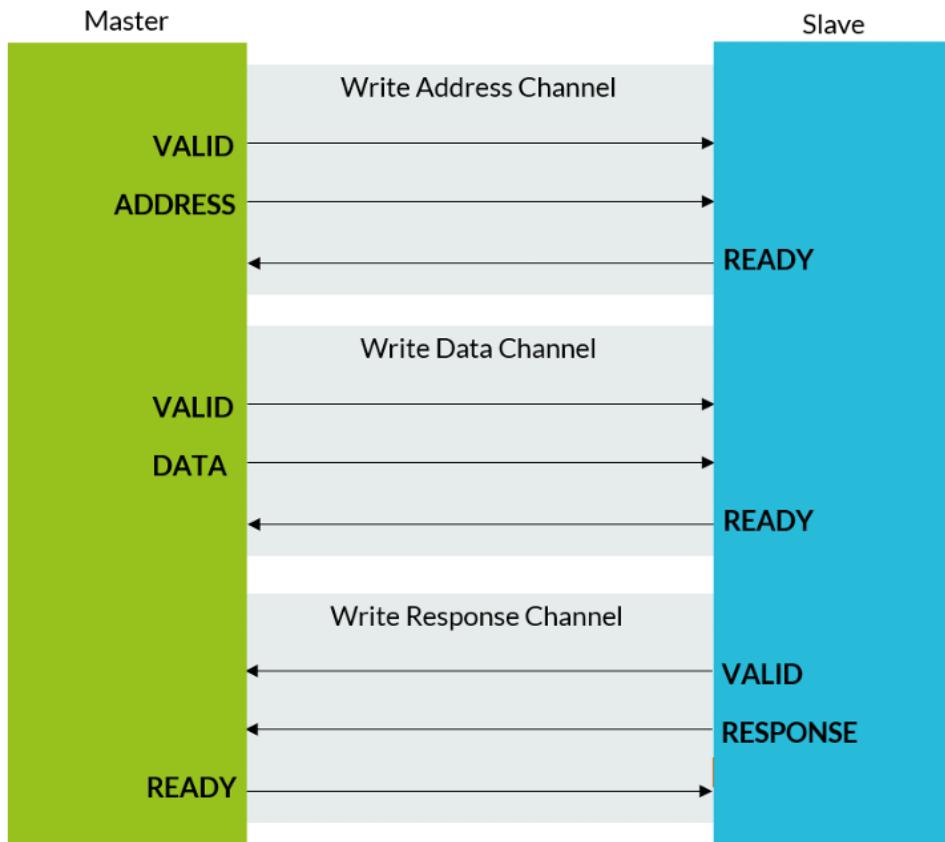
Dựa trên cơ chế bắt tay này, chuẩn AXI định nghĩa hai cấp độ truyền tải dữ liệu cần phân biệt rõ:

**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).



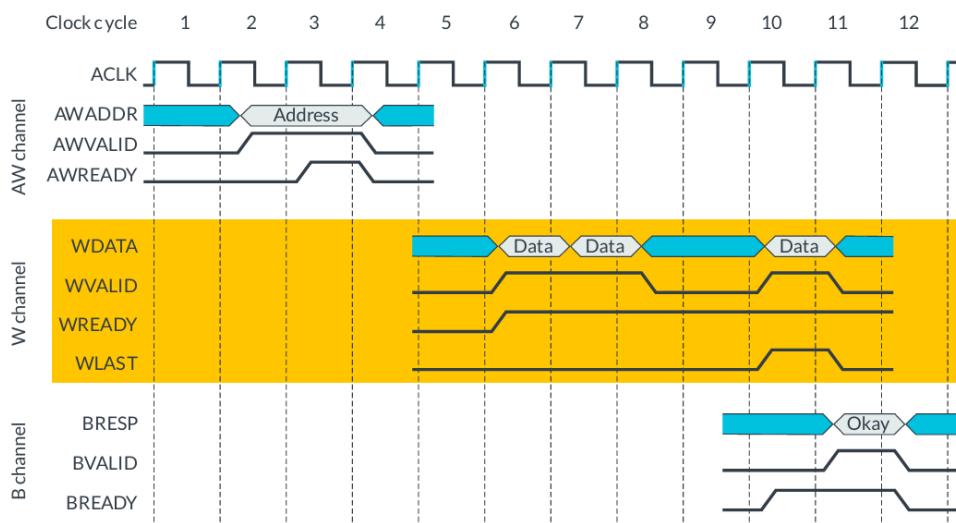
**Hình 2.7:** Minh họa một Transaction trong AXI

### 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.

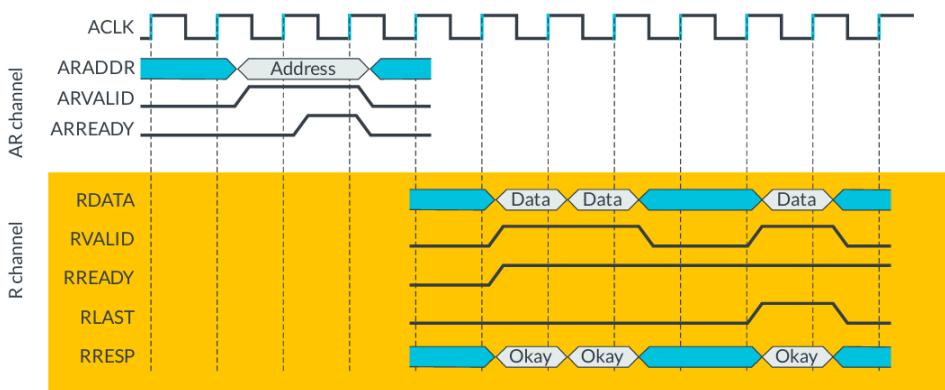
### Pha phản hồi (Write Response Channel):

**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.

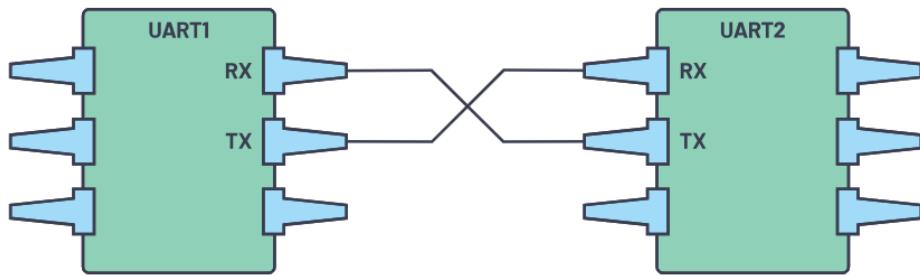
#### 2.3.2.1 Nguyên lý hoạt động

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

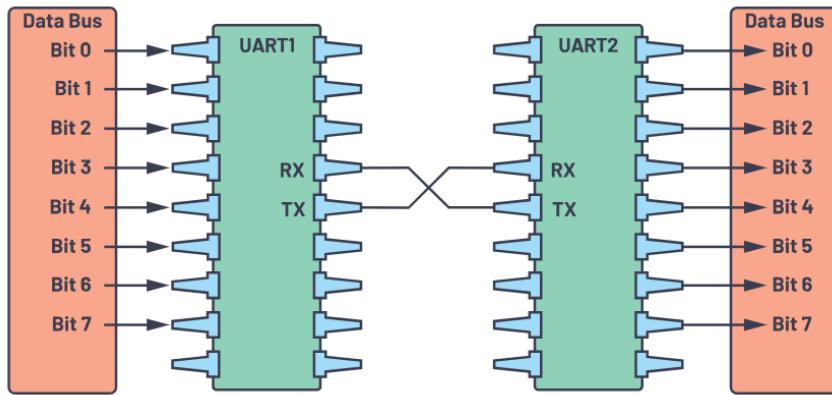
**TX (Transmit):** Chân truyền dữ liệu đi.

**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.

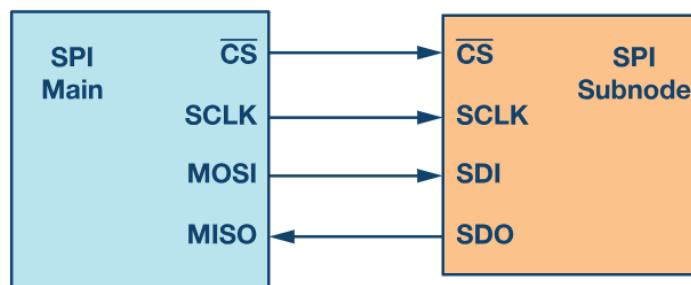
### 2.3.3 Giao thức truyền thông SPI

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.

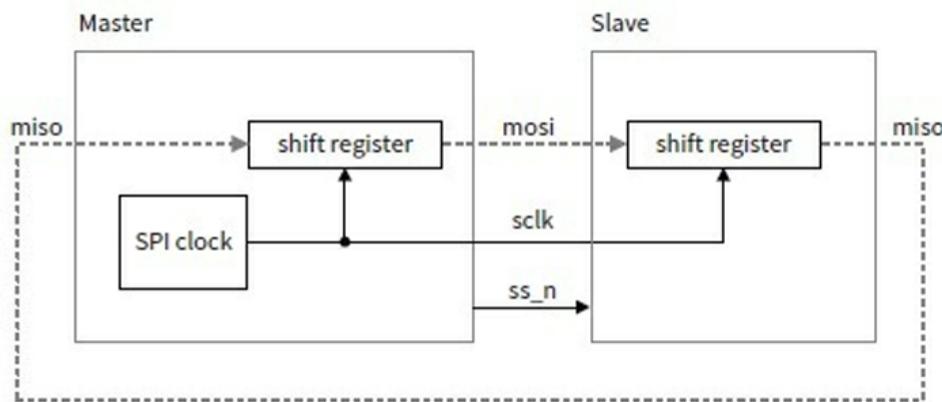
**MOSI (Master Out Slave In):** Đường truyền dữ liệu từ Master đến Slave.

**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:

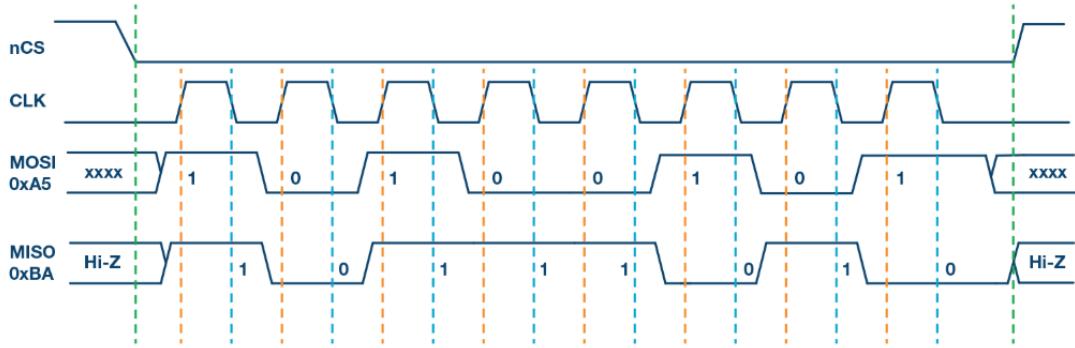
**CPOL (Clock Polarity):** Trạng thái nghỉ của đường SCLK (0 hoặc 1).

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

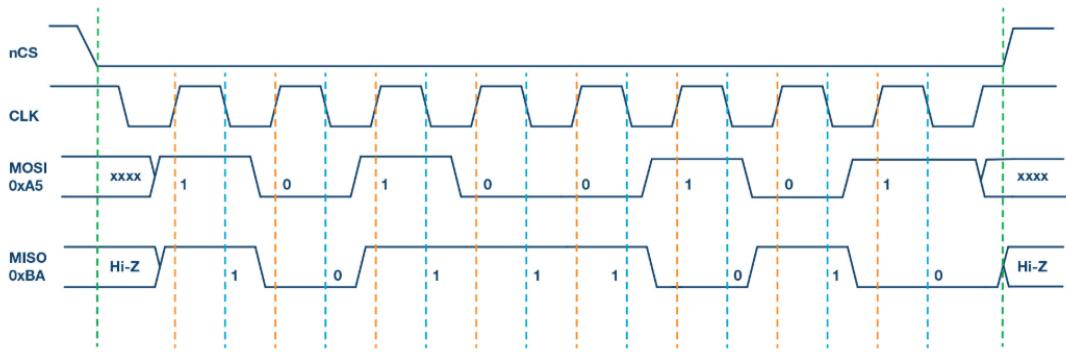
SPI Mode	CPOL	CPHA	Clock Polarity in Idle State	Clock Phase Used to Sample and/or Shift the Data
0	0	0	Logic low	Data sampled on rising edge and shifted out on the falling edge
1	0	1	Logic low	Data sampled on the falling edge and shifted out on the rising edge
2	1	0	Logic high	Data sampled on the falling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



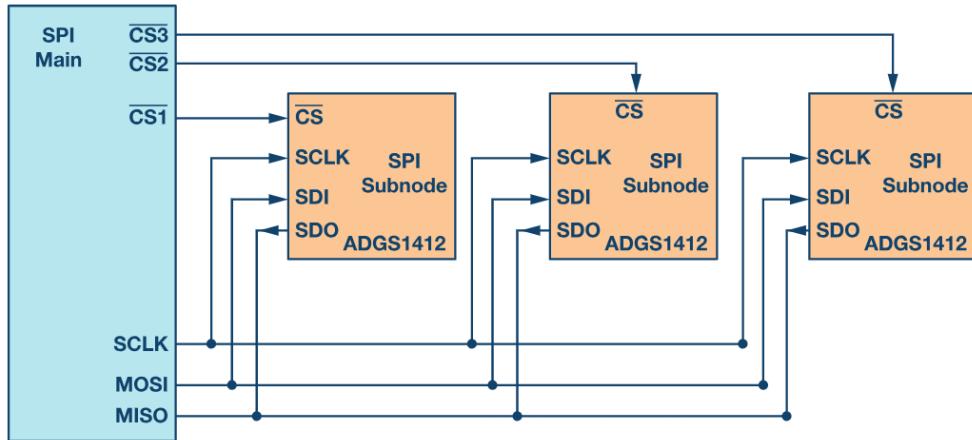
**Hình 2.18:** SPI MODE 3 (CPOL=1, CPHA=1), trạng thái SCLK ban đầu ở mức high, dữ liệu được lấy mẫu tại cạnh lên của SCLK và dịch ở cạnh xuống

#### 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):

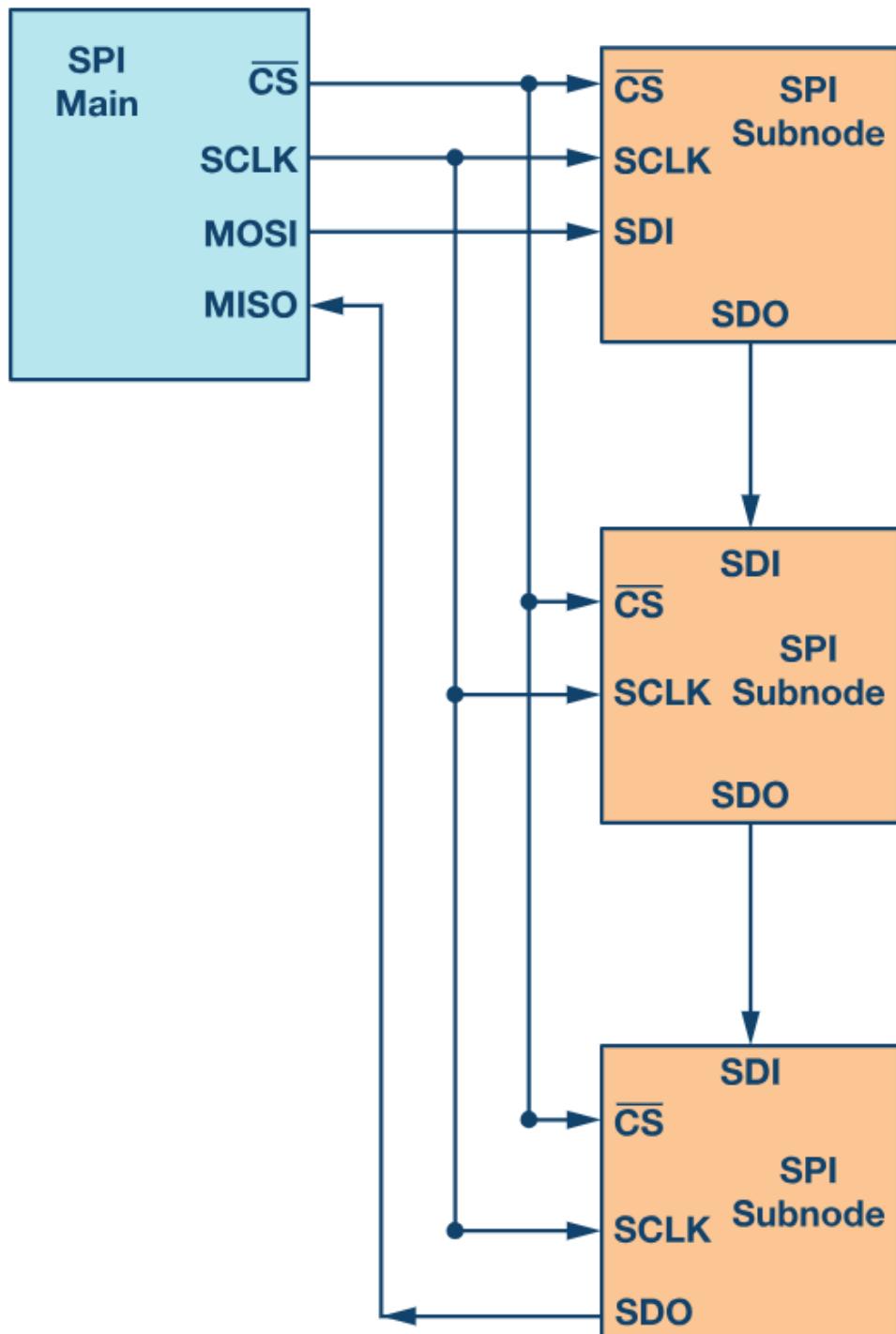
Master sử dụng các chân CS riêng biệt ( $CS_1, CS_2, \dots$ ) cho từng Slave. Đây là cấu hình phổ biến giúp tối ưu băng thông.



**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.

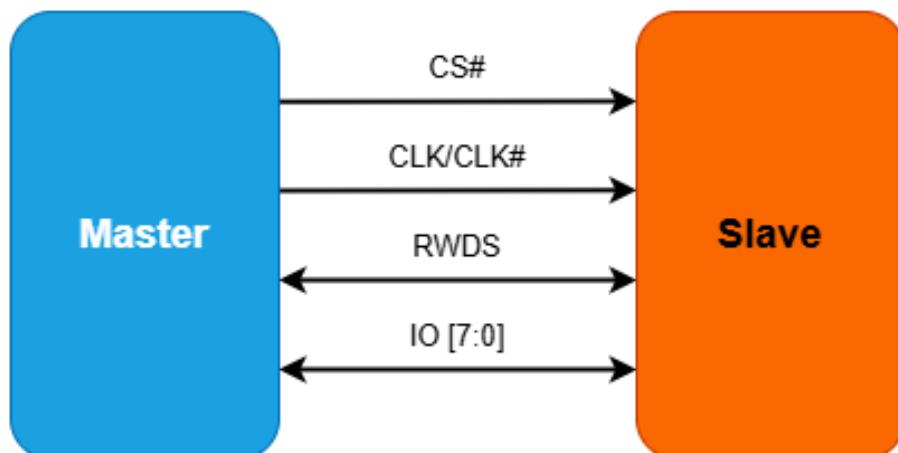
### 2.3.4 Giao thức truyền thông OSPI (Octal SPI)

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.

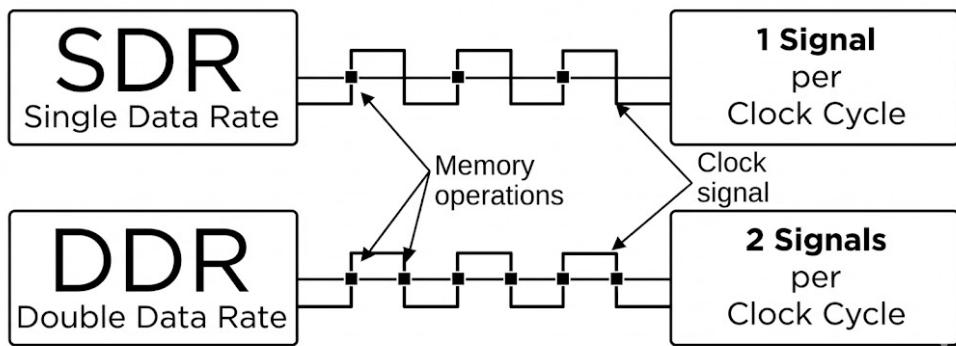
#### 2.3.4.2 Cơ chế truyền tải DDR (Double Data Rate)

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 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).

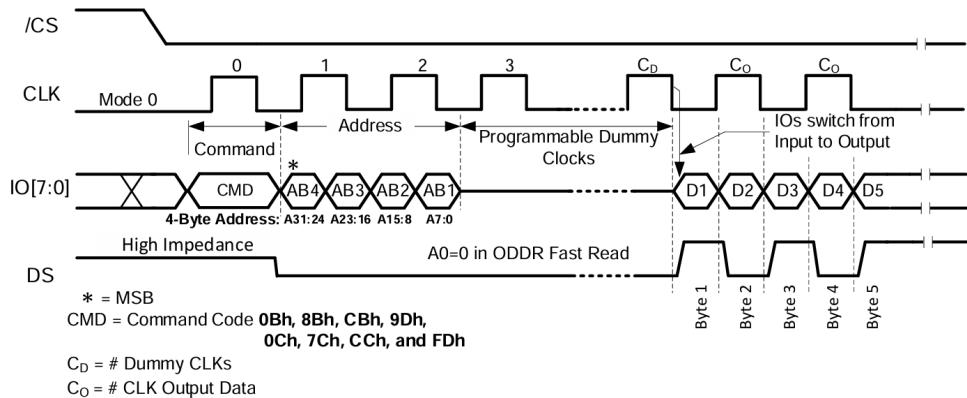
Ví dụ: Với xung nhịp 200MHz, bằng thông lý thuyết của OSPI DDR đạt:  $200 \text{ MHz} \times 2 \text{ Bytes} = 400 \text{ MB/s}$ . Tốc độ này nhanh gấp 8 lần so với QSPI thông thường (chạy SDR) và tiệm cận với các giao tiếp DRAM, đủ đáp ứng nhu cầu xử lý thời gian thực.

#### 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 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).

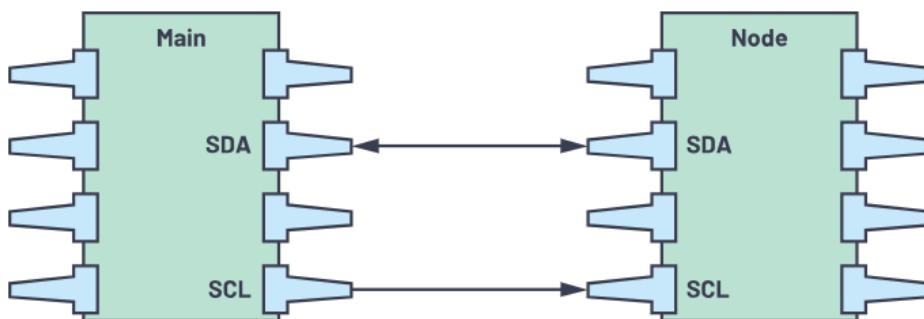
Khác với SPI (cần 4 dây) hay UART (cần 2 dây riêng biệt cho TX/RX), I2C chỉ sử dụng duy nhất **2 đường dây tín hiệu** để kết nối nhiều thiết bị (Multi-master, Multi-slave), giúp tiết kiệm đáng kể số lượng chân IO và diện tích đi dây trên PCB.

### 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).



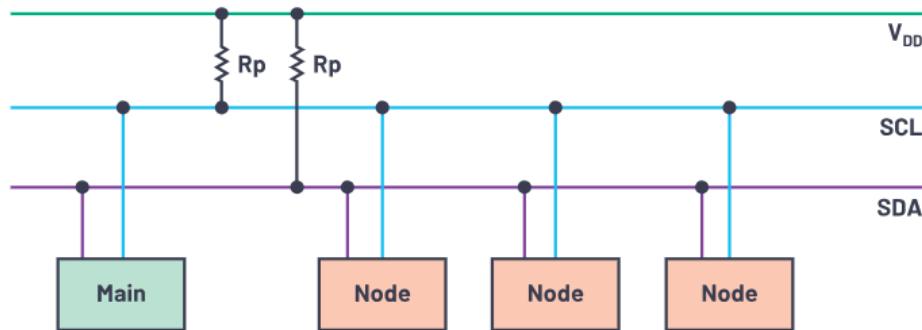
Hình 2.24: Sơ đồ kết nối vật lý I2C

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

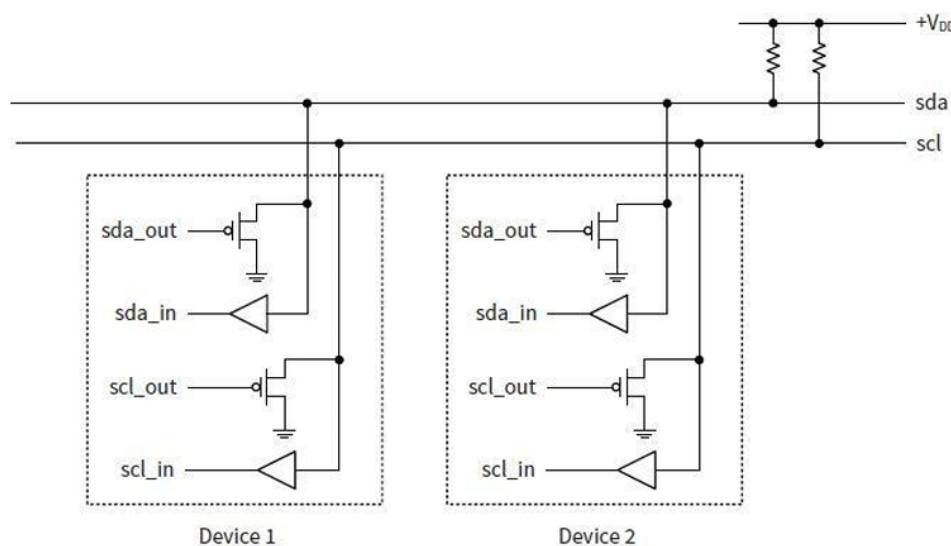
Các thiết bị Master và Slave chỉ có thể kéo đường dây xuống mức thấp (Logic 0). Chúng không thể chủ động đẩy đường dây lên mức cao (Logic 1).

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.

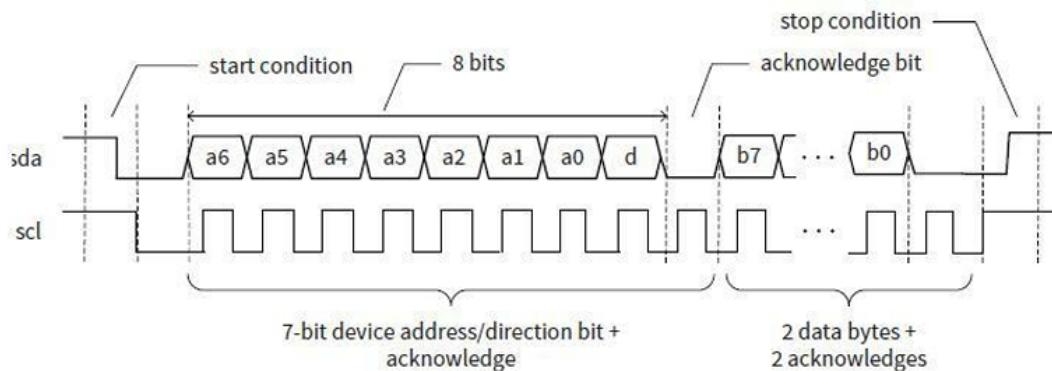


Hình 2.27: Cấu trúc khung truyền dữ liệu I2C

**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).

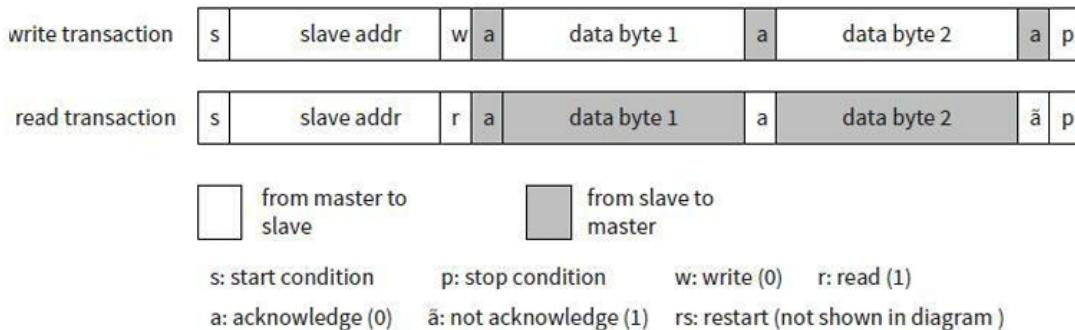
Nếu Bit R/W = 0: Master muốn Ghi dữ liệu vào Slave.

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).

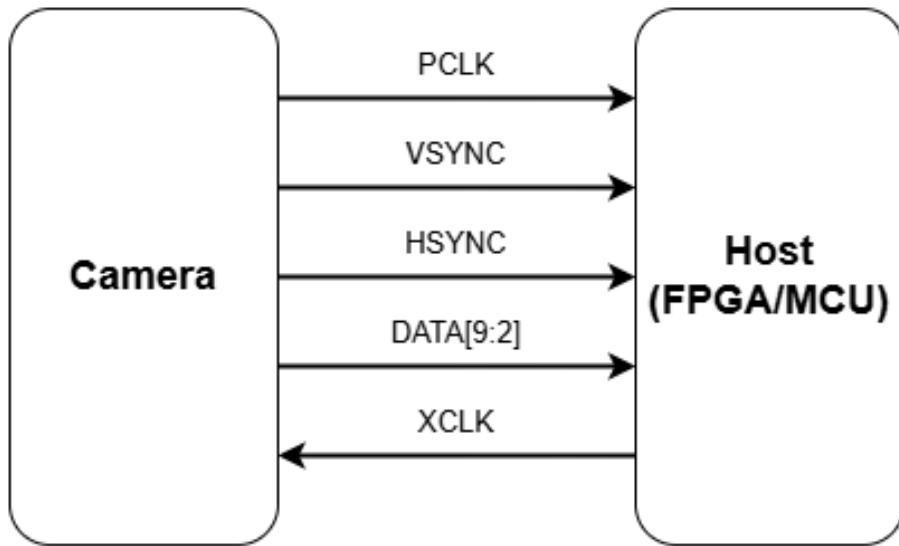
Nhưng về tốc độ chậm hơn nhiều so với SPI và OSPI. Cấu trúc cực máng hở tiêu thụ năng lượng qua điện trở kéo lên. Logic điều khiển phức tạp hơn SPI do phải phát hiện Start/Stop/ACK.

#### 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.

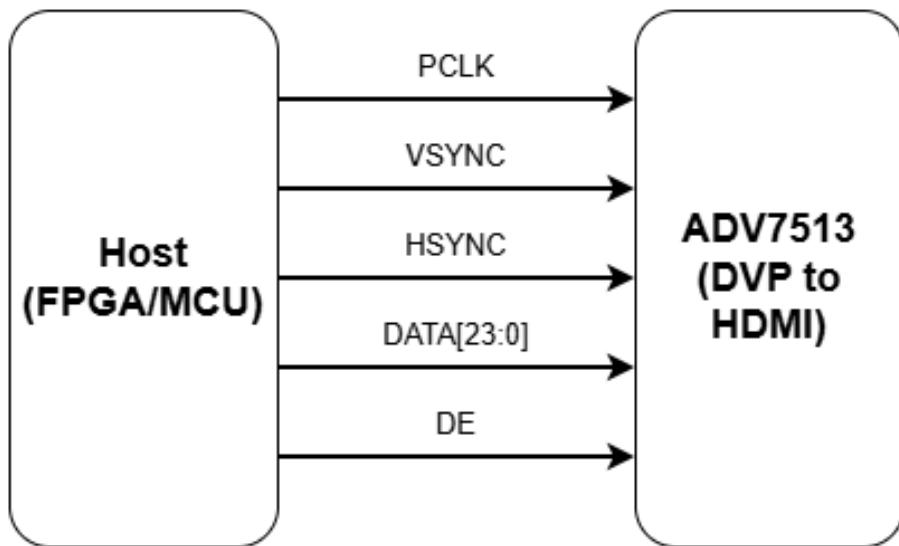
Quá trình này lặp lại liên tục cho đến khi HREF xuống mức thấp, báo hiệu kết thúc một dòng. Cuối cùng, khi VSYNC xuất hiện trở lại, hệ thống kết thúc khung hình hiện tại và quay trở lại trạng thái chờ ban đầu. Việc tuân thủ nghiêm ngặt trình tự này giúp loại bỏ hiện tượng trượt byte (Byte misalignment) 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 đọc (**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 bằng 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 trinh (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)

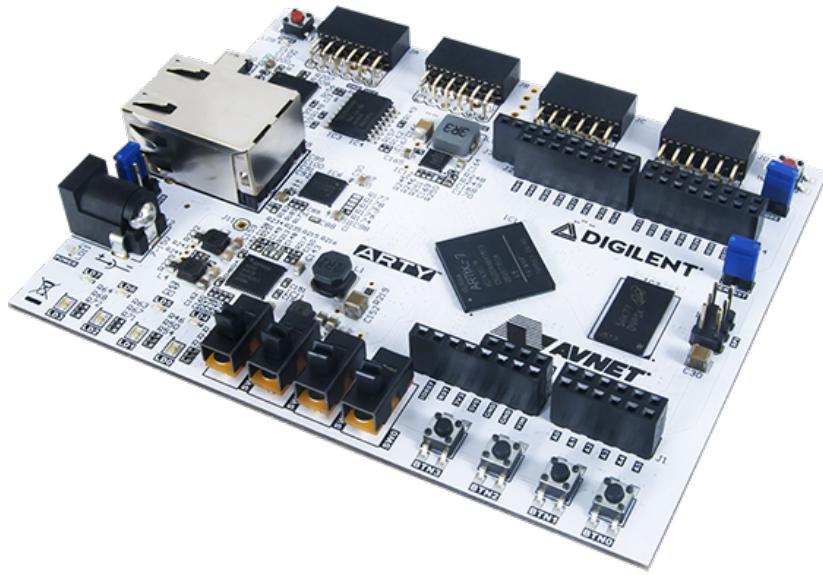
BRAM là các khối bộ nhớ tĩnh (SRAM) dung lượng 36Kb được nhúng rải rác trong FPGA. Chúng đóng vai trò là bộ đệm (Buffer) lưu trữ.

### 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 nhớ 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) **chen2017eyeriss** 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  $12 \times 14$  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) **li2025pixel** 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 chính 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 của số 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) **zhang2017opencl** 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) **liu2020riscv** 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, khói 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 macro" (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) [du2017rsa](#) 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  $3 \times 3$ ,  $1 \times 1$ ) 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

thông.

# 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 nhớ 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.

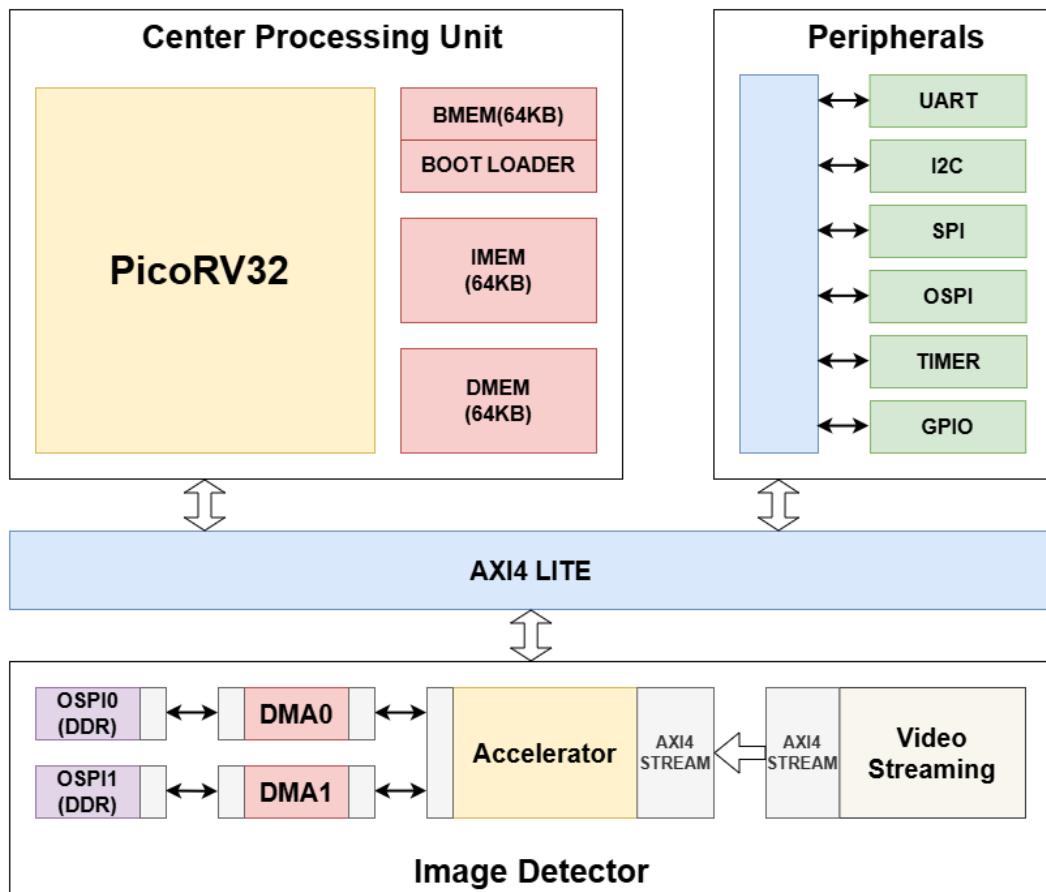
Thứ ba, về mặt tài nguyên, thiết kế được tối ưu hóa cho nền tảng bo mạch **Xilinx VC707** (sử dụng chip Virtex-7 XC7VX485T). Mặc dù đây là dòng FPGA hiệu năng cao với tài nguyên logic dồi dào, thách thức lớn nhất nằm ở việc quản lý hiệu quả băng thông bộ nhớ. Hệ thống phải điều phối chia sẻ quyền truy cập giữa ba tác nhân tiêu thụ băng thông lớn: Vi xử lý (nạp lệnh/dữ liệu), Bộ nhớ (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) nồng nàn 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 để 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 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 trả 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 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 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 đọc dữ liệu đặc trưng đầu vào (Input Feature Maps) và thu thập kết quả tính toán (Output Feature Maps) từ khối Image Detector 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	<b>Data Memory (64KB).</b> Vùng nhớ dữ liệu, Stack, Heap
	0x0001_0000	
BMEM	0x0100_0000	<b>Boot Memory (64KB).</b> Chứa mã khởi động (Bootloader).
	0x0101_0000	
IMEM	0x0110_0000	<b>Instruction Memory (64KB).</b> Vùng nhớ chứa mã lệnh chương trình chính (Firmware).
	0x0111_0000	

#### 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
<b>GPIO</b>	0x8000_0000 0x8000_0FFF	Điều khiển các tín hiệu vào/ra cơ bản (LEDs, Buttons).
<b>UART</b>	0x8000_1000 0x8000_1FFF	Bộ điều khiển giao tiếp nối tiếp (Console/Debug).
<b>I2C</b>	0x8000_2000 0x8000_2FFF	Giao tiếp cấu hình Camera và chip HDMI PHY.
<b>SPI</b>	0x8000_3000 0x8000_3FFF	Giao tiếp thẻ nhớ SD Card hoặc Flash phụ trợ.
<b>OSPI</b>	0x8000_4000 0x8000_4FFF	Giao tiếp bộ nhớ tốc độ cao (Octal-SPI/DDR).
<b>Timer</b>	0x8000_5000 0x8000_5FFF	Bộ định thời gian thực và đo đặc 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:

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

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

---

### 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  $1 \times 1$ . Nó chịu trách nhiệm trộn thông tin giữa các kênh sau khi lớp Depthwise đã xử lý không gian.

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

## 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.

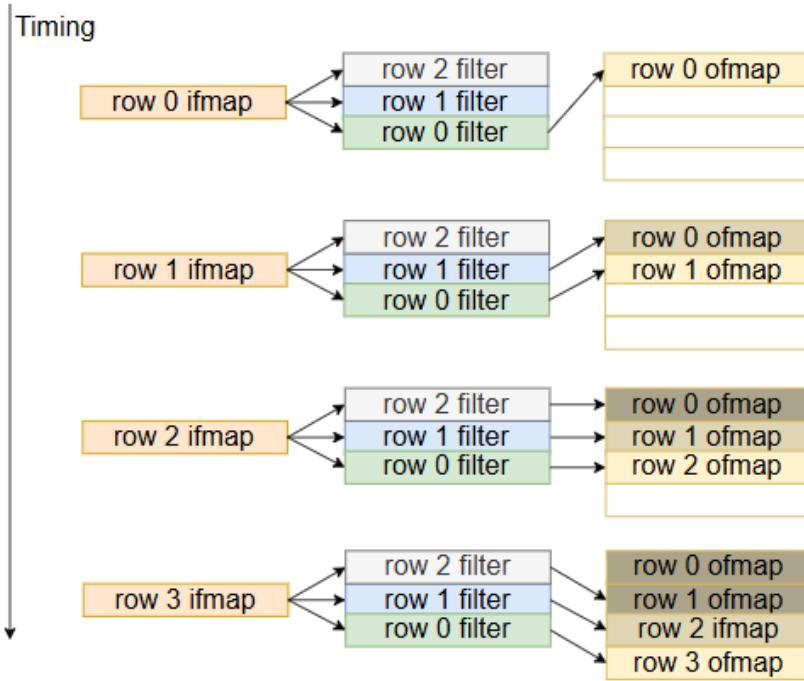
**Bảng 5.1:** Bảng tham số thiết kế và ánh xạ ký hiệu

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:

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

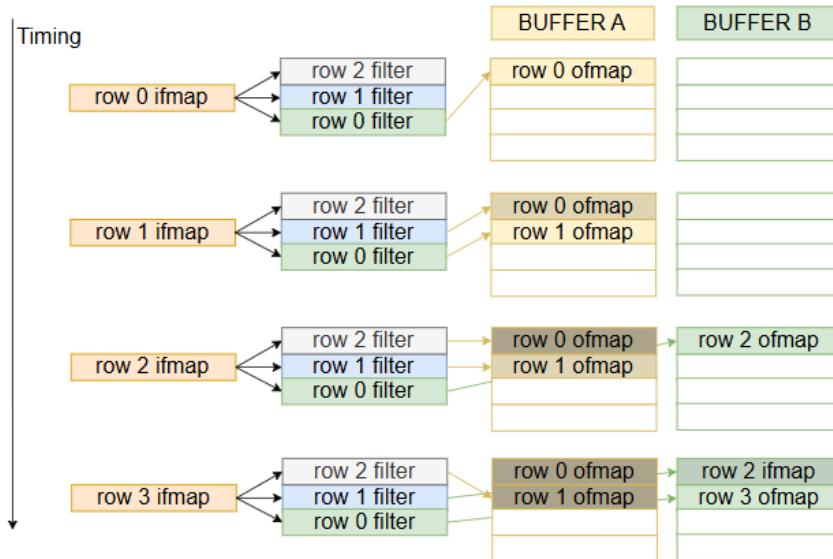
### 5.2.3.2 Logic Hoạt động Ping-Pong

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

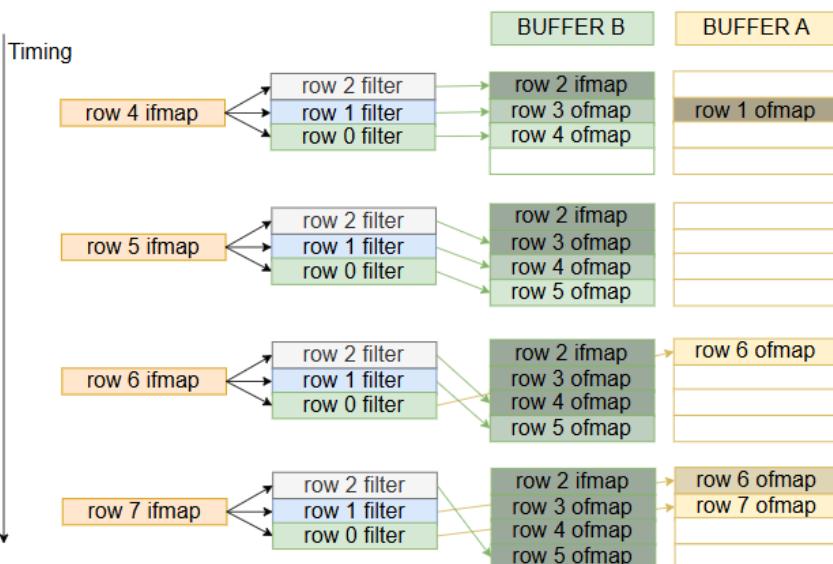
1. **Pass  $k$ :** Ghi kết quả hợp lệ vào  $Buffer$  hiện tại. Các hàng dôi ra được

ghi vào Buffer kế tiếp.

2. **Pass  $k+1$ :** Buffer kế tiếp (chứa dữ liệu 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 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	Pass 1	Pass 2	Pass 3
row 0-10, channel 0-10, filter 0	row 0-10, channel 11-20, filter 0	row 11-20, channel 0-10, filter 0	row 11-20, channel 11-20, filter 0
Pass 4	Pass 5	Pass 6	Pass 7
row 0-10, channel 0-10, filter 1	row 0-10, channel 11-20, filter 1	row 11-20, channel 0-10, filter 1	row 11-20, channel 11-20, filter 1

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

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

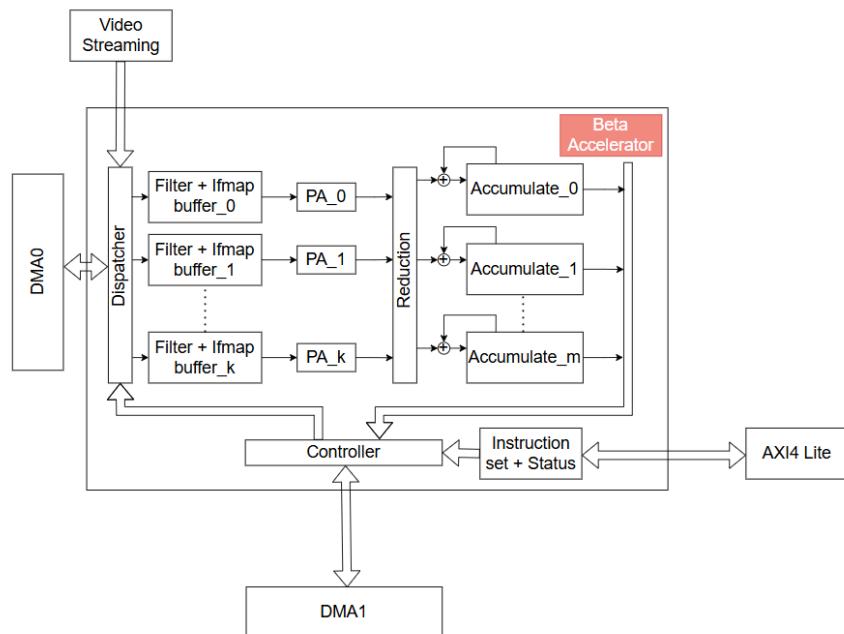
(b) Depthwise Convolution ( $H = 21, M = 21$ )

**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 khái 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

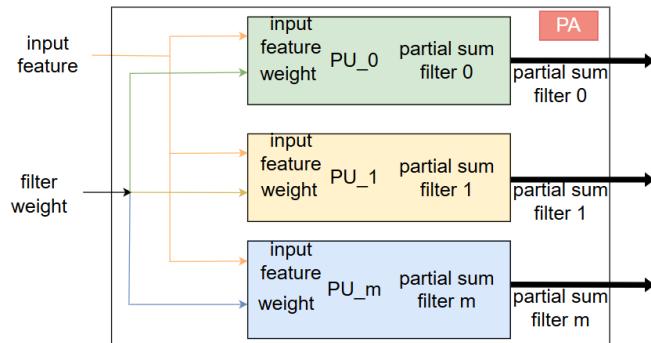
Weight, cho phép nạp dữ liệu Pass  $k + 1$  song song với việc tính toán Pass  $k$ .

- **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 → PU → PE.

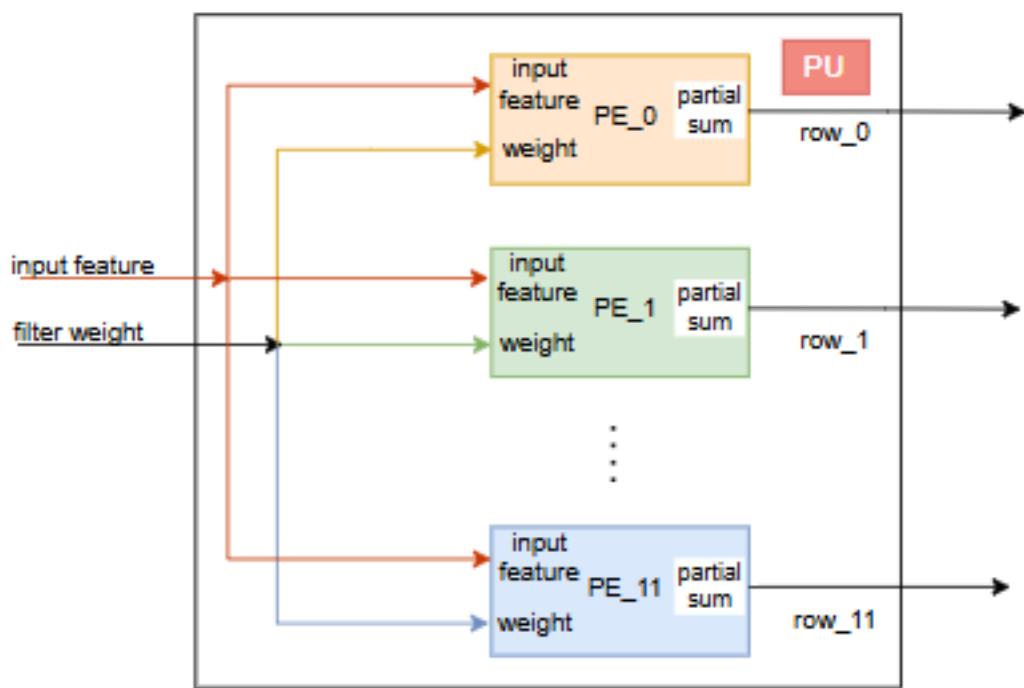
#### 5.3.2.1 Mảng xử lý (Process Array - PA)



**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.

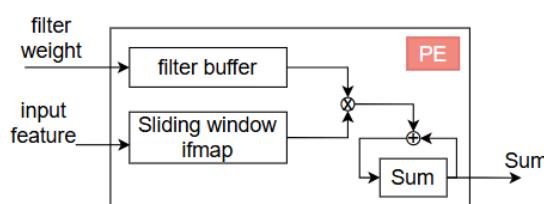
### 5.3.2.2 Đơn vị xử lý (Process Unit - PU)



**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  $11 \times 11$  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:

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

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  $T_c$ .

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

### 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 ( $T_m$ ).

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

**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 và tối ưu hóa tham số

Nhằm xác định cấu hình phần cứng tối ưu cho từng lớp mạng, một mô hình ước lượng thời gian thực thi đã được xây dựng để thực hiện việc khảo sát và đánh giá hiệu năng. Mô hình này tiến hành quét qua toàn bộ không gian của các tham số chia khối (Tiling parameters), bao gồm bộ ba ( $T_c, T_m, T_h$ ), với mục tiêu then chốt là tìm ra tổ hợp tham số giúp tối thiểu hóa tổng số chu kỳ hoạt động (Total Cycles) của hệ thống.

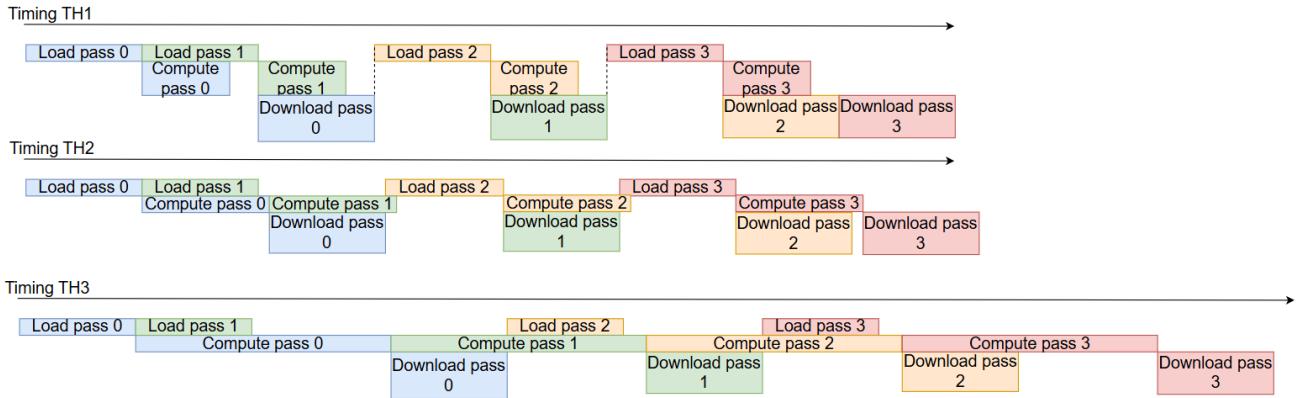
Về cơ chế vận hành, quy trình xử lý bắt đầu bằng giai đoạn khởi tạo để nạp đầy đủ dữ liệu đầu vào (IFM) và trọng số (Weights) vào các bộ đệm (buffer). Sau bước chuẩn bị này, hệ thống sẽ vận hành theo nguyên lý "gối đầu" (pipelining) để tối ưu hóa hiệu suất sử dụng tài nguyên. Cụ thể, trong khi lõi tính toán đang thực hiện xử lý dữ liệu của lượt chạy (Pass) thứ  $i$ , bộ điều khiển DMA sẽ đồng thời nạp dữ liệu cho Pass  $i + 1$  vào nửa còn lại của bộ đệm kép. Song song với đó, nếu kết quả của Pass  $i - 1$  đã hoàn tất, chúng sẽ được ghi trả về bộ nhớ ngoài một cách độc lập.

Tuy nhiên, do hệ thống sử dụng cấu trúc bus dữ liệu dùng chung (Shared Data Bus) cho cả hai luồng nạp (Load) và ghi (Store), nên băng thông bộ nhớ cần được phân bổ theo phương thức chia sẻ thời gian. Để đảm bảo lõi tính toán luôn có sẵn dữ liệu cho chu kỳ kế tiếp, bộ điều khiển sẽ ưu tiên thực hiện tác vụ nạp dữ liệu của Pass tiếp theo trước, sau đó mới tiến hành ghi kết quả của Pass trước đó, hoặc thực hiện xen kẽ dựa trên các chính sách điều phối của bộ trọng tài bus.

#### 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 ( $T_{load} \geq T_{comp}$ ). 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:

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

- **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:

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

### 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:

$$T_{total} = T_{load} + \sum_{all\_passes} T_{comp} + T_{store} \quad (5.11)$$

#### 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):

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

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 tương 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
<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 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 (UART, SPI, OSPI, I2C, Camera/HDMI DVP), 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

Để quá trình hiện thực hóa hệ thống SoC diễn ra đồng bộ và chính xác từ mức thiết kế phần cứng (RTL) đến lớp phần mềm điều khiển (Firmware), đề tài sử dụng một hệ thống các công cụ chuyên dụng, đảm bảo tính tương thích chặt chẽ giữa kiến trúc tập lệnh RISC-V và nền tảng FPGA.

### 6.1.1 Môi trường thiết kế và kiểm thử phần cứng

Quá trình thiết kế logic cho hệ thống SoC được thực hiện chủ yếu trong môi trường **Xilinx Vivado Design Suite**. Đây là nền tảng thiết kế tích hợp (IDE) cho phép quản lý các khối IP tùy biến, thực hiện các bước từ

phân tích logic, tổng hợp (Synthesis) cho đến việc tối ưu hóa và hiện thực hóa (Implementation). Toàn bộ các phân hệ chức năng, bao gồm lõi xử lý PicoRV32, hệ thống Bus, Video Streaming và các bộ điều khiển ngoại vi như UART, SPI, I2C, OSPI, TIMER, GPIO đều được mô tả bằng ngôn ngữ **Verilog HDL**. Việc sử dụng ngôn ngữ mô tả phần cứng ở mức thanh ghi (RTL) giúp nhóm thiết kế kiểm soát chính xác tài nguyên logic trên FPGA và hướng tới ASIC, đảm bảo hệ thống đáp ứng được các ràng buộc khắt khe về định thời (Timing constraints) để vận hành ổn định tại tần số 200 MHz.

Trong giai đoạn xác minh thiết kế, công cụ **Vivado Simulator** đóng vai trò then chốt trong việc kiểm tra tính đúng đắn về mặt chức năng (Functional Verification). Tất cả các khối IP tự thiết kế đều phải trải qua quá trình mô phỏng thông qua hệ thống Testbench trước khi tích hợp vào hệ thống tổng thể. Quy trình này cho phép phát hiện sớm các lỗi về giao thức bắt tay trên bus AXI, logic điều phối dữ liệu của bộ gia tốc CNN và khả năng xử lý dòng dữ liệu của hệ thống Video Streaming. Đối với các lỗi phát sinh trong điều kiện thực tế trên phần cứng, lõi **Integrated Logic Analyzer (ILA)** được nhúng trực tiếp vào chip để giám sát các tín hiệu nội bộ theo thời gian thực, giúp gỡ lỗi các giao tiếp vật lý phức tạp như Camera/HDMI DVP, AXI,....

### 6.1.2 Môi trường phát triển phần mềm và quy trình nhúng mã

Phần mềm điều khiển hệ thống được phát triển dựa trên ngôn ngữ C/C++ tiêu chuẩn, cho phép tận dụng hệ sinh thái mã nguồn mở phong phú của kiến trúc RISC-V. Để chuyển đổi mã nguồn thành các tệp thực thi tương thích với phần cứng đề xuất, đề tài sử dụng bộ công cụ **RISC-V GNU Toolchain**. Trình biên dịch `riscv32-unknown-elf-gcc` được cấu hình với tham số `-march=rv32im` và `-mabi=ilp32`, tạo ra mã máy tối ưu hóa

cho tập lệnh số nguyên cơ bản của lõi PicoRV32.

Điểm cốt lõi trong quy trình hiện thực hóa phần mềm là cơ chế nạp chương trình hai giai đoạn, giúp biến SoC từ một thiết kế phần cứng cố định thành một nền tảng lập trình nhúng đa năng. Trong giai đoạn tổng hợp trên Vivado, chỉ có khôi bộ nhớ khởi động **BMEM** được nhúng sẵn chương trình **Bootloader** dưới dạng tệp tin Instruction Hex (.hex). Ngay khi hệ thống khởi vận, chương trình Bootloader này sẽ tự động thiết lập các giao thức cần thiết để đọc dữ liệu từ bộ nhớ Flash ngoại vi và chuyển vào bộ nhớ lệnh **IMEM**.

Cơ chế này mang lại sự thuận tiện vượt trội cho người dùng trong quá trình phát triển ứng dụng. Thay vì phải tái tổng hợp (Re-synthesis) và nạp lại toàn bộ tệp tin cấu hình FPGA (.bit) mỗi khi thay đổi phần mềm — một quy trình vốn tiêu tốn nhiều thời gian — người lập trình chỉ cần biên dịch chương trình C/C++ mới và cập nhật vào bộ nhớ Flash. Sau khi hoàn tất quá trình nạp dữ liệu vào IMEM, Bootloader sẽ thực hiện lệnh nhảy tới địa chỉ khởi chạy ứng dụng. Giải pháp này giúp tách biệt hoàn toàn giữa lớp hạ tầng phần cứng và lớp ứng dụng người dùng, tạo điều kiện cho việc gỡ lỗi nhanh chóng và triển khai linh hoạt các bài toán khác nhau, từ quản trị hệ thống đơn thuần đến các thuật toán xử lý dữ liệu phức tạp.

## 6.2 Tích hợp Lõi RISC-V

### 6.2.1 Khái quát về lõi vi xử lý PicoRV32

Trái tim của hệ thống SoC được xây dựng dựa trên lõi **PicoRV32**, một hiện thực vi xử lý mã nguồn mở tuân thủ kiến trúc tập lệnh (ISA) RISC-V. Được thiết kế với mục tiêu tối ưu hóa tài nguyên logic, PicoRV32 đặc biệt phù hợp cho các hệ thống nhúng đòi hỏi kích thước nhỏ gọn nhưng vẫn duy trì được tần số xung nhịp cao. Trong đề tài này, nhóm thiết kế lựa chọn phiên bản **picorv32\_axi** để hiện thực hóa việc kết nối. Khác với phiên

bản giao tiếp bộ nhớ trực tiếp thông thường, biến thể này tích hợp sẵn giao diện bus chuẩn **AXI4** (Advanced eXtensible Interface), cho phép vi xử lý tương tác đồng bộ với hạ tầng Interconnect và các ngoại vi phức tạp trong hệ thống thông qua các giao thức bắt tay (handshake) tiêu chuẩn.

### 6.2.2 Phân tích cấu hình và tùy chọn tập lệnh

Để đạt được sự cân bằng giữa hiệu năng tính toán và mức tiêu thụ tài nguyên trên FPGA, lõi PicoRV32 được cấu hình thông qua một hệ thống các tham số (*parameters*) chuyên biệt. Về năng lực xử lý số học, mặc dù hệ thống dựa trên nền tảng cơ bản là tập lệnh số nguyên 32-bit (**RV32I**), để tài quyết định kích hoạt các khối tính toán phần cứng cho phép nhân và phép chia thông qua tham số `ENABLE_MUL` và `ENABLE_DIV`. Việc chuyển đổi sang tập lệnh **RV32IM** này đóng vai trò then chốt trong việc tăng tốc các thuật toán xử lý dữ liệu mà không cần phụ thuộc vào các thư viện phần mềm mô phỏng phép tính, vốn thường gây trễ lớn trong các ứng dụng thời gian thực. Ngược lại, tập lệnh nén (`COMPRESSED_ISA`) được thiết lập ở mức 0 để giữ cho logic giải mã lệnh đơn giản, ưu tiên độ ổn định về định thời tại tần số 200 MHz.

Về quản lý tài nguyên nội tại, hệ thống tận dụng tối đa khả năng của lõi thông qua việc kích hoạt đầy đủ 32 thanh ghi đa năng và cơ chế truy xuất hai cổng (`ENABLE_REGS_DUALPORT`), giúp tối ưu hóa luồng thực thi lệnh trong ALU. Bên cạnh đó, các bộ đếm hiệu năng (`ENABLE_COUNTERS`) cũng được tích hợp để phục vụ quá trình đo đạc thời gian thực thi của các đoạn mã điều khiển. Nhằm đảm bảo an toàn hệ thống, các cơ chế bẫy lỗi như bắt lỗi lệnh không hợp lệ (`CATCH_ILLINSN`) và lỗi căn chỉnh bộ nhớ (`CATCH_MISALIGN`) luôn ở trạng thái hoạt động, cho phép hệ thống tự động nhảy vào trạng thái bảo vệ (Trap) khi xảy ra sự cố phần mềm.

### 6.2.3 Thiết lập ngữ cảnh thực thi và kết nối hệ thống

Một khía cạnh quan trọng trong việc tích hợp lõi là quy hoạch địa chỉ khởi vận và không gian ngăn xếp để phù hợp với Memory Map toàn cục. Tham số PROGADDR\_RESET được thiết lập tại địa chỉ  $32'h0100_0000$ , trỏ trực tiếp vào vùng nhớ **BMEM** nơi chứa chương trình Bootloader. Điều này đảm bảo ngay sau khi tín hiệu Reset được giải phóng, CPU sẽ bắt đầu quy trình nạp chương trình ứng dụng từ Flash vào IMEM như đã thiết kế. Đồng thời, địa chỉ đỉnh ngăn xếp (STACKADDR) được xác định tại  $32'h0001_0000$  trong vùng nhớ dữ liệu DMEM, cung cấp không gian lưu trữ an toàn cho các biến cục bộ và ngữ cảnh hàm của ngôn ngữ C.

Trong quá trình hiện thực kết nối vật lý, lõi cpu0 được ánh xạ các tín hiệu giao diện AXI4 bao gồm các kênh địa chỉ ghi (awaddr), dữ liệu ghi (wdata), địa chỉ đọc (araddr) và dữ liệu đọc (rdata). Sự phối hợp giữa các tín hiệu valid và ready trên bus đảm bảo rằng mọi giao dịch truy xuất giữa vi xử lý và các ngoại vi như UART, I2C hay bộ giao tiếp AI đều diễn ra chính xác theo đúng chu kỳ xung nhịp hệ thống. Mặc dù cơ chế ngắn (IRQ) được hỗ trợ về mặt logic, trong thiết kế hiện tại, các chân ngắn được giữ ở mức 0 để tập trung vào cơ chế thăm dò (*Polling*) chủ động, giúp đơn giản hóa lớp trình điều khiển thiết bị trong giai đoạn phát triển ban đầu.

#### 6.2.3.1 Chi tiết thiết lập tham số phần cứng (Parameters)

Để lõi vi xử lý PicoRV32 vận hành tối ưu trong hệ thống SoC, các tham số cấu hình được thiết lập cụ thể nhằm cân bằng giữa tài nguyên logic và hiệu năng. Bảng 6.1 liệt kê các tham số chính được sử dụng trong quá trình hiện thực thực thể `picorv32_axi`.

**Bảng 6.1:** Cấu hình tham số phần cứng cho lõi PicoRV32

Tham số (Parameter)	Giá trị	Mô tả chức năng
ENABLE_MUL	1	Kích hoạt bộ nhân phần cứng (RV32M).
ENABLE_DIV	1	Kích hoạt bộ chia phần cứng (RV32M).
COMPRESSED_ISA	0	Không sử dụng tập lệnh nén (C).
BARREL_SHIFTER	0	Tối ưu diện tích, dịch bit theo chu kỳ.
ENABLE_REGS_16_31	1	Sử dụng đầy đủ 32 thanh ghi RV32I.
ENABLE_REGS_DUALPORT	1	Hỗ trợ truy xuất đồng thời hai thanh ghi.
ENABLE_COUNTERS	1	Kích hoạt bộ đếm hiệu năng hệ thống.
CATCH_MISALIGN	1	Bắt lỗi truy cập bộ nhớ không căn chỉnh.
CATCH_ILLINSN	1	Bẫy lỗi khi gặp tập lệnh không hợp lệ.
REGS_INIT_ZERO	1	Khởi tạo giá trị thanh ghi bằng 0 khi Reset.
PROGADDR_RESET	0x0100_0000	Địa chỉ nạp chương trình Boot-loader.
STACKADDR	0x0001_0000	Địa chỉ đỉnh ngăn xếp trong vùng DMEM.

Việc thiết lập các tham số trên biến lõi xử lý thành kiến trúc **RV32IM**. Điều này cho phép SoC thực hiện các phép toán số học phức tạp một cách nhanh chóng, đóng vai trò quan trọng trong việc điều phối các thuật toán xử lý dữ liệu. Để đảm bảo hệ thống đạt được tần số 200 MHz, tham số **BARREL\_SHIFTER** được đặt bằng 0 nhằm giảm mức độ phức tạp của logic tổ hợp, giúp thiết kế đạt được các ràng buộc về mặt định thời trên FPGA.

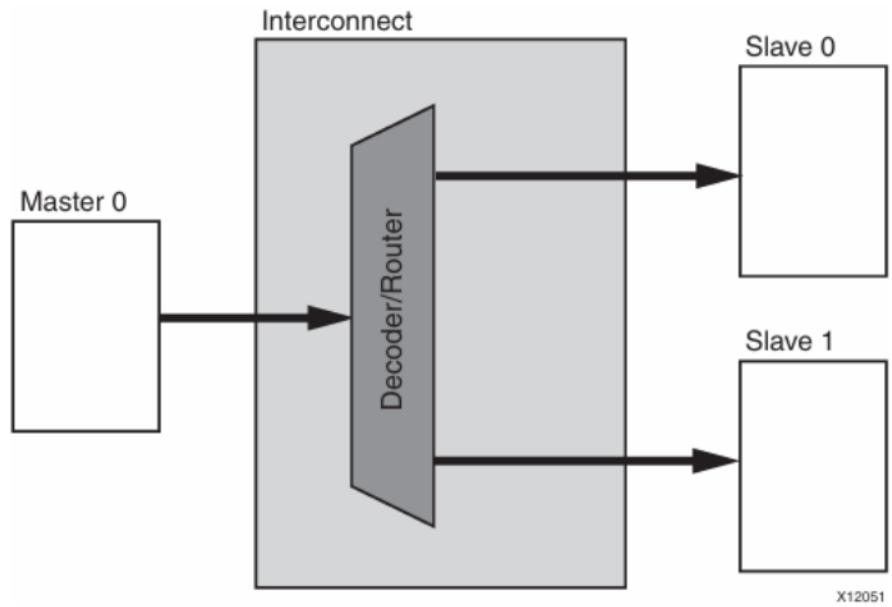
## 6.3 Thiết kế hệ thống Bus

Hệ thống Bus đóng vai trò là hạ tầng giao tiếp xương sống của SoC, thực hiện việc kết nối và điều phối luồng dữ liệu giữa lõi vi xử lý PicoRV32 cùng các Master khác với các phân hệ ngoại vi và bộ nhớ nội. Để đảm bảo tính chuẩn hóa, hiệu năng truyền tải và khả năng tùy biến cao, đề tài đã hiện thực hóa một bộ **AXI4-Lite Interconnect** hoàn chỉnh. Hệ thống này không chỉ tuân thủ chặt chẽ giao thức bắt tay (handshake) của chuẩn AXI4-Lite mà còn tích hợp các cơ chế quản lý giao dịch phức tạp để tối ưu hóa băng thông và đảm bảo an toàn vận hành thông qua ba mô hình kết nối đặc thù.

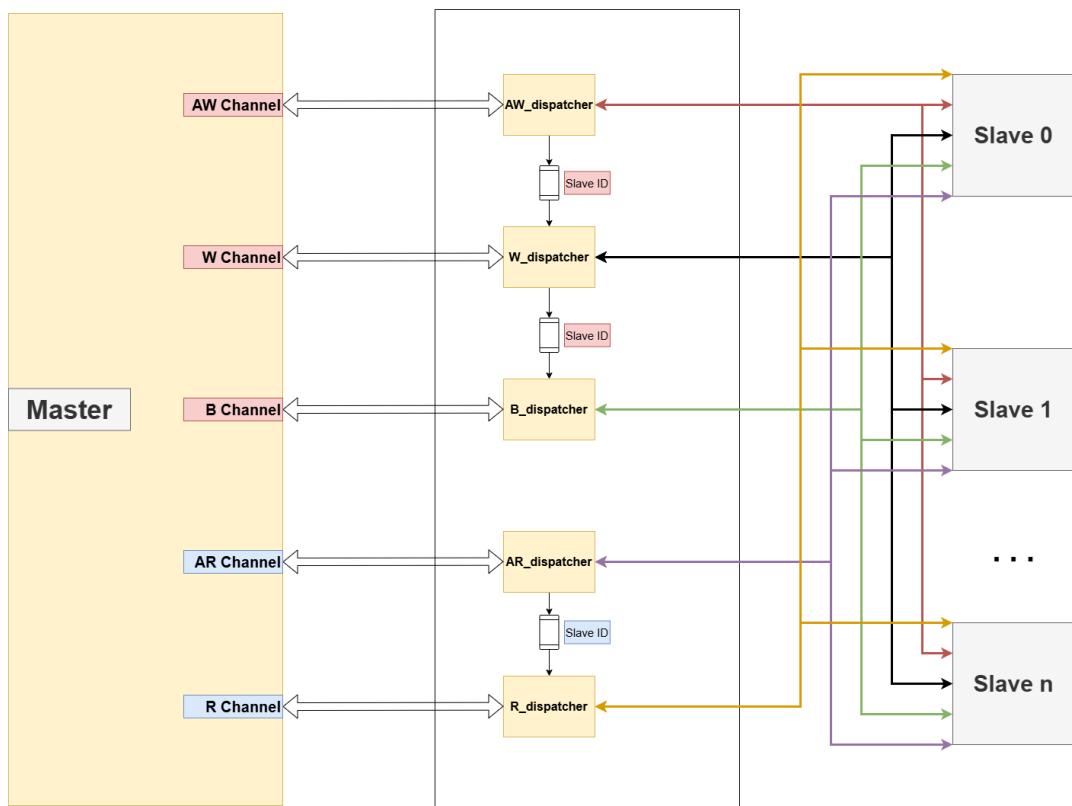
### 6.3.1 Mô hình điều phối đơn Master - đa Slave ( $1 \times N$ )

Mô hình kết nối  $1 \times N$  đóng vai trò là bộ định tuyến dữ liệu từ một Master duy nhất đến hệ thống nhiều Slaves ngoại vi. Trong cấu trúc này, chức năng cốt lõi là giải mã địa chỉ (Address Decoding) dựa trên tham số cấu hình **ADDR\_MAP\_SLAVES**. Khi Master khởi tạo một giao dịch, các khối điều phối **AW\_dispatcher** và **AR\_dispatcher** sẽ phân tích địa chỉ đích để xác định chính xác Slave cần tương tác. Cơ chế này cho phép vi xử lý trung tâm quản lý toàn bộ tài nguyên hệ thống một cách thông nhât trên không gian địa chỉ phẳng, đồng thời tối ưu hóa tài nguyên logic bằng cách loại bỏ các bộ trọng tài phức tạp không cần thiết khi chỉ có một thực thể điều

khiến.



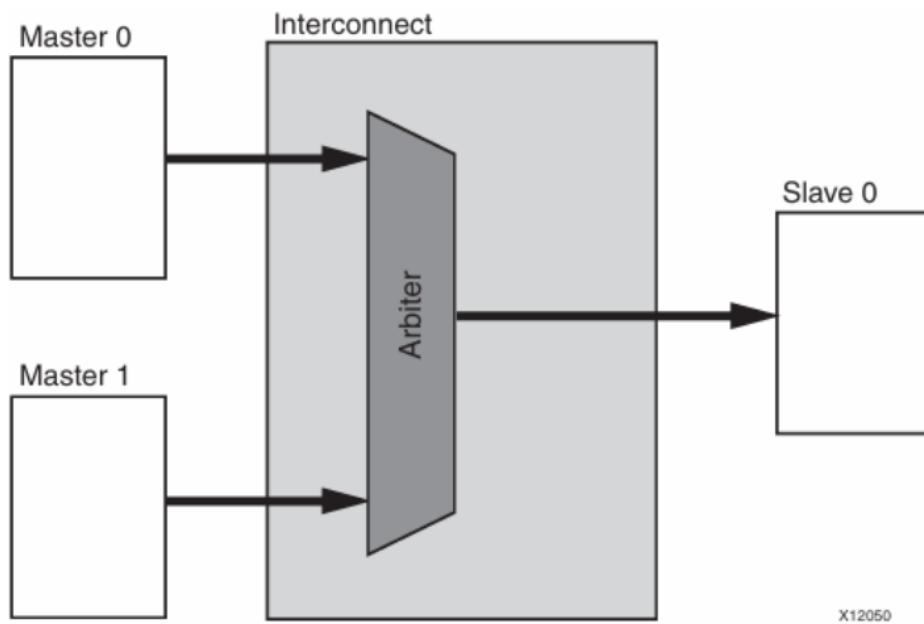
**Hình 6.1:** a. Sơ đồ khái niệm mô hình kết nối AXI4-Lite  $1 \times N$



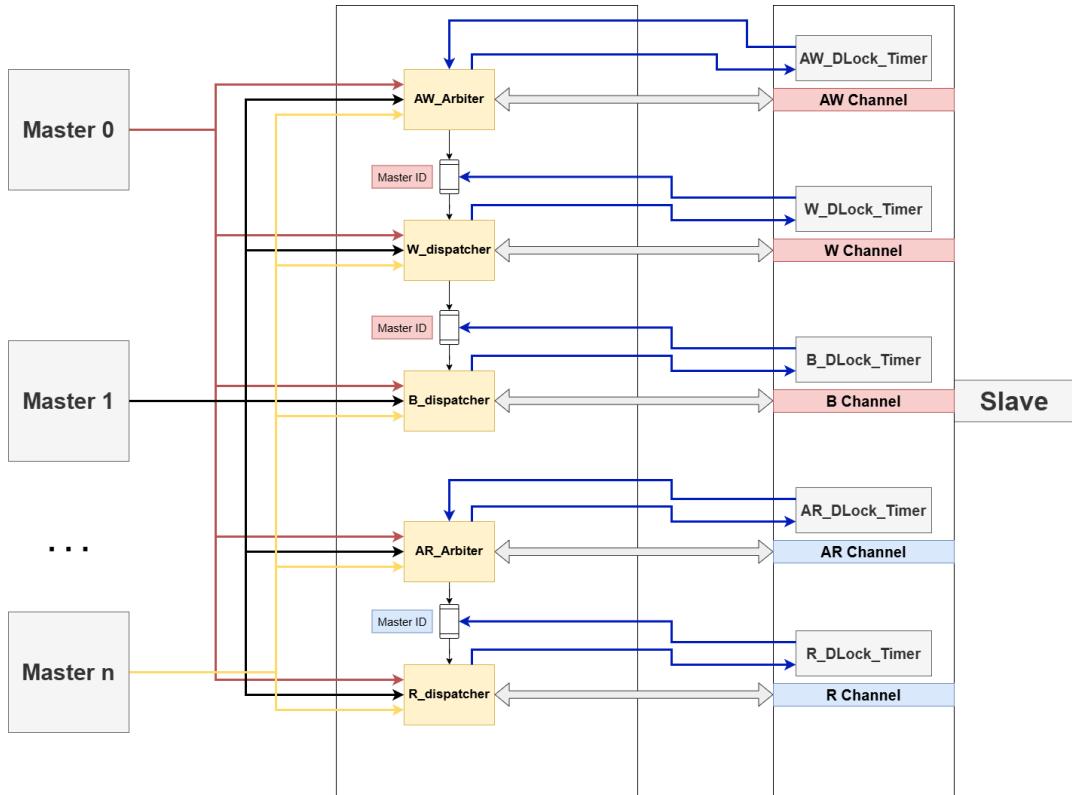
**Hình 6.2:** b. Sơ đồ khái niệm mô hình kết nối AXI4-Lite  $1 \times N$

### 6.3.2 Mô hình trọng tài đa Master - đơn Slave ( $N \times 1$ )

Ngược lại với mô hình trên, mô hình này giải quyết bài toán xung đột truy cập khi nhiều Master (như CPU và các bộ gia tốc DMA) cùng muốn tương tác với một tài nguyên dùng chung duy nhất, điển hình là bộ nhớ nội. Tại đây, trọng tâm của thiết kế nằm ở bộ trọng tài (Arbiter) hoạt động theo thuật toán xoay vòng Round-Robin. Logic này đảm bảo tính công bằng trong việc cấp quyền chiếm giữ bus (grant\_permission), triệt tiêu hiện tượng nghẽn mạch và đảm bảo rằng không có Master nào chiếm dụng tài nguyên quá lâu gây ảnh hưởng đến tính thời gian thực của hệ thống. Sự kết hợp giữa bộ đếm counter\_arbiter và các mạng lưới Multiplexer giúp luân chuyển dữ liệu từ Master được chọn đến Slave một cách mượt mà và chính xác.



Hình 6.3: a. Sơ đồ khái niệm về trọng tài tài nguyên trong mô hình  $N \times 1$

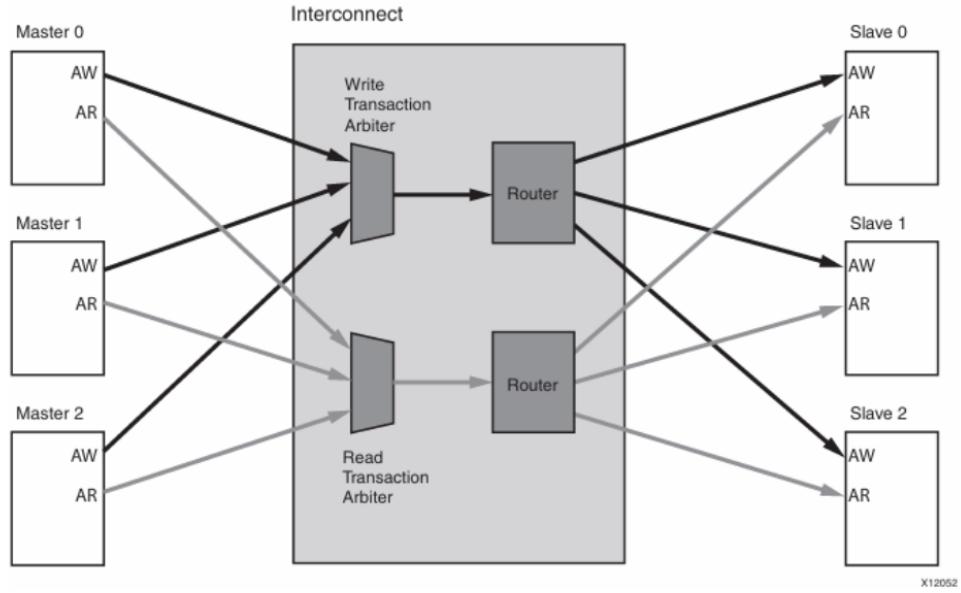


**Hình 6.4: b. Sơ đồ khối cơ chế trọng tài trong mô hình  $N \times 1$**

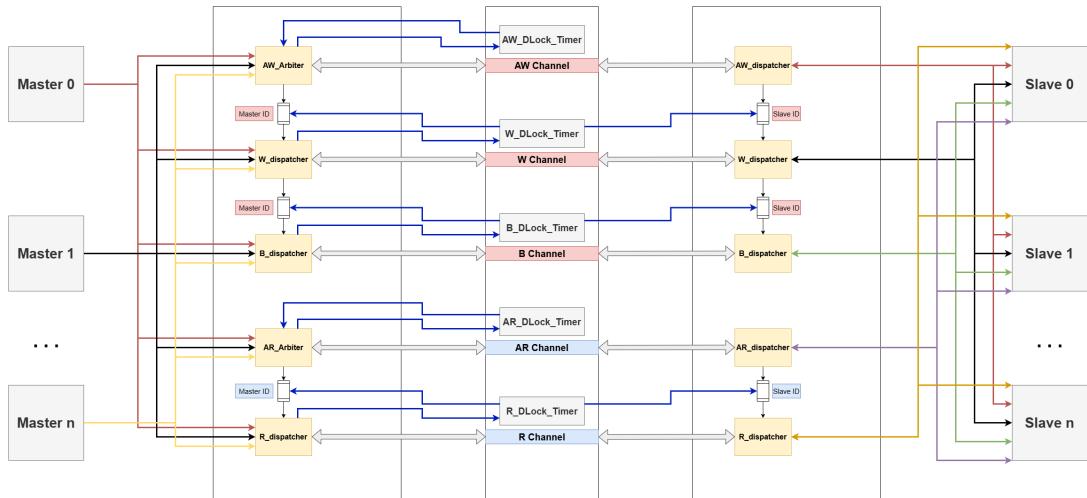
### 6.3.3 Mô hình phức hợp đa Master - đa Slave ( $N \times M$ )

Mô hình đa Master - đa Slave ( $N \times M$ ) đại diện cho cấu trúc kết nối tổng quát và linh hoạt nhất trong hệ thống SoC, được hiện thực hóa bằng phương pháp kết hợp phân tầng (cascading) hai mô hình cơ sở  $1 \times N$  và  $N \times 1$ . Khi hệ thống được cấu hình với số lượng Master lớn hơn một, một quy trình xử lý luồng dữ liệu hai giai đoạn sẽ được thiết lập.

Ở giai đoạn đầu tiên, tầng  $N \times 1$  đảm nhiệm vai trò trọng tài để điều phối và lựa chọn một yêu cầu giao dịch duy nhất từ các Master đang cùng truy cập bus. Sau khi Master được cấp quyền truy cập được xác định, luồng tín hiệu sẽ được chuyển tiếp qua tầng  $1 \times N$ . Tại đây, hệ thống tiếp tục thực hiện các thao tác giải mã địa chỉ và dẫn hướng dữ liệu đến đúng thiết bị Slave mục tiêu trong mạng lưới  $M$  ngoại vi.



**Hình 6.5:** a. Kiến trúc kết hợp Cascaded Interconnect cho mô hình đa Master - đa Slave



**Hình 6.6:** b. Kiến trúc kết hợp Cascaded Interconnect cho mô hình đa Master - đa Slave

Việc áp dụng kiến trúc phân tầng này mang lại khả năng mở rộng quy mô cực kỳ linh hoạt cho SoC. Người thiết kế có thể dễ dàng thay đổi số lượng cổng Master và Slave thông qua các tham số `NUM_MASTER`s và `NUM_SLAVE`s mà không cần can thiệp vào cấu trúc logic cốt lõi. Bên cạnh đó, việc tách biệt rõ rệt giữa logic quản lý xung đột truy cập và logic định tuyến dữ liệu giúp tối ưu hóa các đường trẽ tín hiệu (Critical Path). Đây là yếu tố then chốt giúp thiết kế bus đáp ứng được các ràng buộc khắt khe về định thời,

đảm bảo hệ thống vận hành ổn định tại tần số xung nhịp 200MHz.

### 6.3.4 Kiến trúc tổng thể và cơ chế cấu hình linh hoạt

Thành phần trung tâm của hạ tầng bus là module top **AXI4-Lite Interconnect**, được thiết kế với khả năng thích ứng cao theo cấu hình Master-Slave của hệ thống. Điểm đặc biệt trong hiện thực này là dựa trên tham số **NUM\_MASTERS** và **NUM\_SLAVES**.

Khi hệ thống chỉ vận hành với một Master duy nhất, Interconnect sẽ tự động cấu hình theo mô hình **1 × N**, giúp tối ưu hóa tài nguyên bằng cách chỉ sử dụng logic giải mã địa chỉ đơn giản. Trong trường hợp hệ thống có nhiều Master cùng truy cập (như khi tích hợp thêm các kênh DMA cho bộ gia tốc AI), Interconnect sẽ thiết lập một cấu trúc phân cấp phức hợp theo mô hình **N × M**. Cụ thể, hệ thống sẽ sử dụng tầng **N × 1** để thực hiện trọng tài giữa các Master, sau đó dẫn hướng luồng dữ liệu qua tầng **1 × N** để phân phối đến các thiết bị Slave mục tiêu.

Kiến trúc phân tầng (cascaded) này giúp tách biệt rõ ràng giữa logic điều phối quyền truy cập Master và logic giải mã địa chỉ Slave, từ đó giảm thiểu độ phức tạp của các đường trẽ logic tổ hợp, đảm bảo tính toàn vẹn của dữ liệu trên toàn hệ thống Bus.

### 6.3.5 Cơ chế điều phối giao dịch và giải mã địa chỉ

Trong tầng phân phối tín hiệu, hệ thống sử dụng các module *Dispatcher* chuyên trách cho từng kênh của giao thức AXI. Các module **AW\_dispatcher** và **AR\_dispatcher** đảm nhiệm việc thực thi bản đồ địa chỉ thông qua tham số **ADDR\_MAP\_SLAVES**. Bằng cách so sánh địa chỉ ghi (**awaddr**) hoặc địa chỉ đọc (**araddr**) phát ra từ CPU với các dải địa chỉ đã quy hoạch, khối logic sẽ kích hoạt tín hiệu chọn Slave (**slave\_selected**) tương ứng.

Điểm nổi bật trong thiết kế này là khả năng hiện thực hóa cơ chế **Pipeline giao dịch** (Transaction Pipelining) nhằm tối ưu hóa hiệu suất truyền tải.

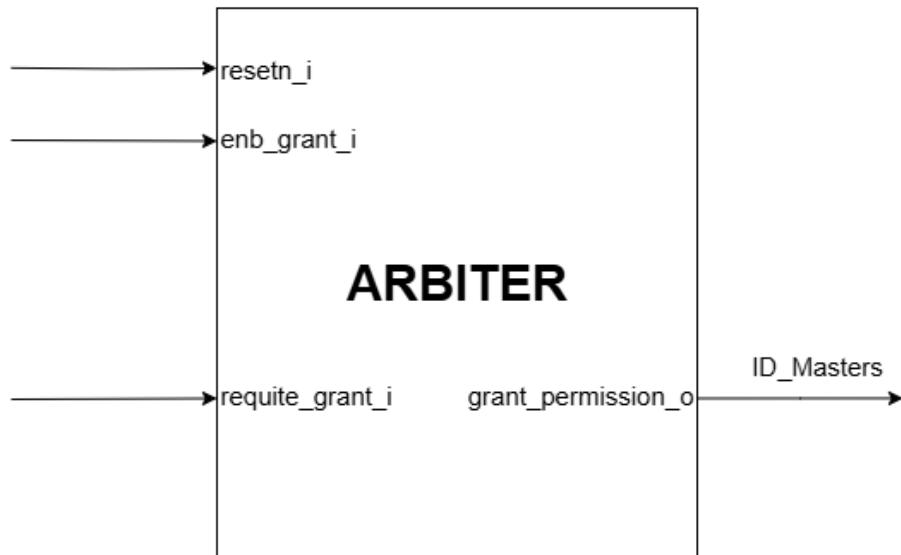
Giao thức AXI vốn có đặc tính tách biệt giữa pha địa chỉ, pha dữ liệu và pha phản hồi; hệ thống đã tận dụng đặc điểm này bằng cách tích hợp các hàng đợi **FIFO** làm bộ đệm trung gian giữa các kênh. Cụ thể, đối với giao dịch ghi, ngay khi pha địa chỉ tại kênh **AW** hoàn tất, định danh của Slave mục tiêu (**slave\_id**) sẽ được đẩy vào khố **fifo\_AW\_W** (nằm ở giữa AW và W). Tại thời điểm này, kênh **AW** được giải phóng hoàn toàn để sẵn sàng tiếp nhận và giải mã yêu cầu địa chỉ tiếp theo từ CPU, trong khi luồng dữ liệu hiện hành vẫn đang được xử lý song song tại kênh **W**.

Quá trình pipeline này tiếp tục được duy trì thông qua khố **fifo\_W\_B** (nằm ở giữa W và B) để chuyển tiếp thông tin định danh sang pha phản hồi (**B**), và tương tự với khố **fifo\_AR\_R** (nằm ở giữa AR và R) cho luồng giao dịch đọc. Sự kết hợp giữa Dispatcher và các tầng FIFO cho phép nhiều giao dịch có thể diễn ra dưới hình thức gối đầu (overlapping) trên các kênh khác nhau mà không gây xung đột. Cơ chế này không chỉ đảm bảo tính toàn vẹn của dữ liệu bằng cách truy vấn FIFO để dẫn hướng chính xác tín hiệu bắt tay về đúng Master, mà còn triệt tiêu các trạng thái chờ không cần thiết, giúp hệ thống đạt được băng thông tối đa và giảm thiểu độ trễ cho vi xử lý trung tâm.

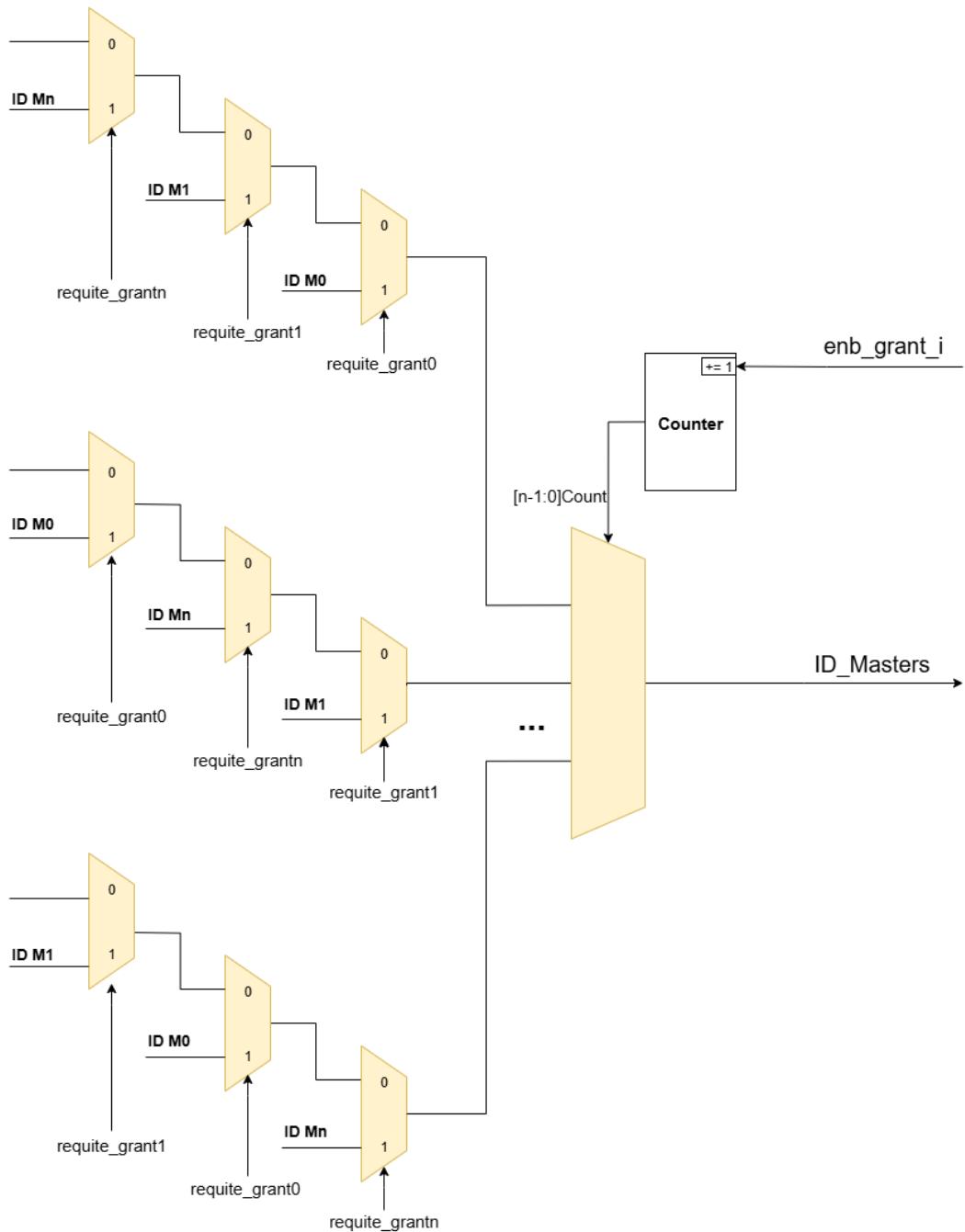
### 6.3.6 Logic trọng tài Round-Robin và quản lý truy cập

Để quản lý việc truy cập tài nguyên dùng chung từ nhiều Master, hệ thống hiện thực thuật toán trọng tài xoay vòng (**Round-Robin**) trong module **arbiter**. Khác với cơ chế ưu tiên cố định có thể gây ra hiện tượng "đói tài nguyên" cho các Master mức thấp, Round-Robin đảm bảo mọi Master đều có cơ hội chiếm giữ bus một cách công bằng. Logic trọng tài được xây dựng dựa trên sự phối hợp giữa module **counter\_arbiter** để luân chuyển quyền ưu tiên và module **ID\_Masters** để thực hiện ánh xạ tín hiệu. Khi một Master được cấp quyền thông qua tín hiệu **grant\_permission**, toàn bộ

kênh địa chỉ và dữ liệu sẽ được kết nối trực tiếp đến Slave thông qua một mạng lưới Multiplexer đóng gói trong module **axi\_interconnect\_n\_1**. Cơ chế này triệt tiêu các xung đột truy cập và đảm bảo rằng hiệu năng của bộ gia tốc AI không bị ảnh hưởng bởi các tác vụ quản trị của vi xử lý trung tâm.



**Hình 6.7: a. Sơ đồ khối cơ chế trọng tài Round-Robin**



Hình 6.8: b. Sơ đồ khối cơ chế trọng tài Round-Robin

### 6.3.7 Hệ thống an toàn và tầng giao diện Slave

Nhằm đối phó với các kịch bản ngoại vi bị treo hoặc không phản hồi tín hiệu bắt tay, thiết kế bus tích hợp khối **DLock\_timer** (Deadlock Timer). Cơ chế này giám sát thời gian thực hiện của mỗi giao dịch dựa trên tham số

`QUANTUM_TIME` (mặc định là 16 chu kỳ). Nếu một Slave giữ tín hiệu `valid` mà không trả về `ready` vượt quá khoảng thời gian này, bộ định thời sẽ kích hoạt `tick_timer` để tự động giải phóng bus, ngăn chặn tình trạng toàn bộ SoC bị treo vô thời hạn. Cuối cùng, tại điểm cuối của mỗi nhánh bus, module `axi_lite_slave_interface` thực hiện vai trò cầu nối, chuyển đổi các kênh tín hiệu phức tạp của AXI thành các chân điều khiển cơ bản như `addr_w`, `wen` và `data_w`. Việc sử dụng các thanh ghi đệm `register_DFF` tại tầng giao diện này giúp cô lập miền thời gian và tối ưu hóa đường truyền.

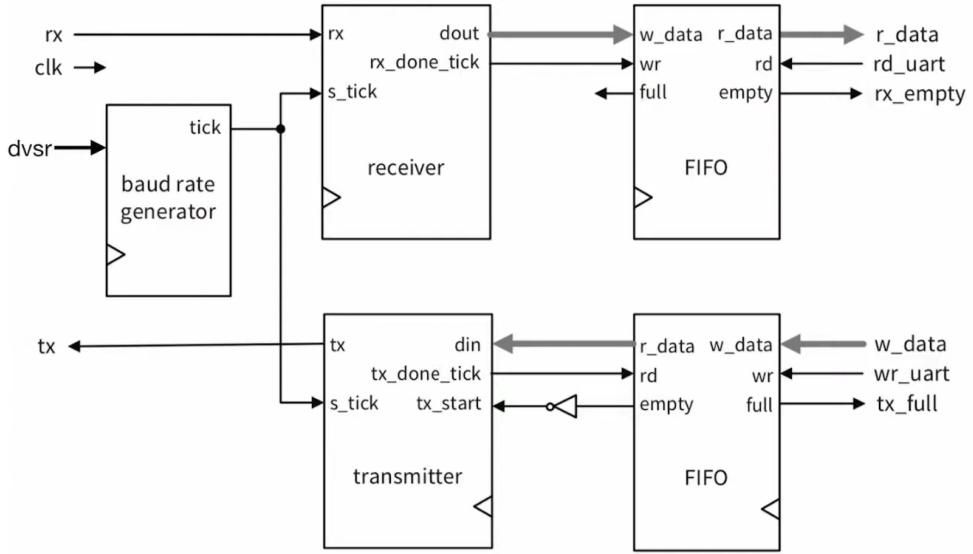
## 6.4 Thiết kế và Tích hợp các khối Ngoại vi

### 6.4.1 UART

Khối UART được thiết kế để đảm nhiệm vai trò kênh giao tiếp gỡ lỗi (Console/Debug) và truyền nhận dữ liệu tốc độ thấp. Toàn bộ khối IP được xây dựng theo kiến trúc phân lớp, từ các tầng vật lý xử lý tín hiệu nối tiếp đến tầng giao diện thanh ghi tương thích chuẩn AXI.

#### 6.4.1.1 Kiến trúc hệ thống và bộ tạo tốc độ Baud

Cấu trúc tổng quát của ngoại vi UART dựa trên module `uart_unit`, tích hợp các thành phần cốt lõi bao gồm bộ tạo tốc độ Baud, khối transmitter/receiver và các bộ đệm FIFO hàng đợi.



**Hình 6.9:** Sơ đồ khối kiến trúc module UART Unit

Nền tảng vận hành của khối UART là module **buad\_gen**, đóng vai trò như một bộ chia tần số linh hoạt. Với xung nhịp hệ thống cao (200 MHz), bộ tạo tốc độ Baud sử dụng một thanh ghi đếm và một thanh ghi giá trị chia (**dvsr**) để tạo ra các xung **tick** định thời. Các xung này được thiết lập tại tần số gấp 16 lần tốc độ Baud mục tiêu (Oversampling), cho phép khối nhận (*Receiver*) thực hiện lấy mẫu dữ liệu chính xác tại trung tâm của mỗi bit truyền. Cơ chế này giúp tăng cường khả năng chống nhiễu và bù đắp sai số xung nhịp giữa SoC và các thiết bị ngoại vi giao tiếp.

Giá trị nạp vào thanh ghi cấu hình **dvsr** được xác định dựa trên mối tương quan giữa tần số hệ thống và tốc độ truyền thông mong muốn thông qua công thức sau:

$$v = \frac{f_{clk}}{16 \times B} - 1 \quad (6.1)$$

Trong đó các tham số được định nghĩa cụ thể:

- $v$ : Giá trị số nguyên được nạp vào thanh ghi **dvsr** của bộ tạo Baud.
- $f_{clk}$ : Tần số xung nhịp hệ thống đầu vào (trong thiết kế này là 200

MHz).

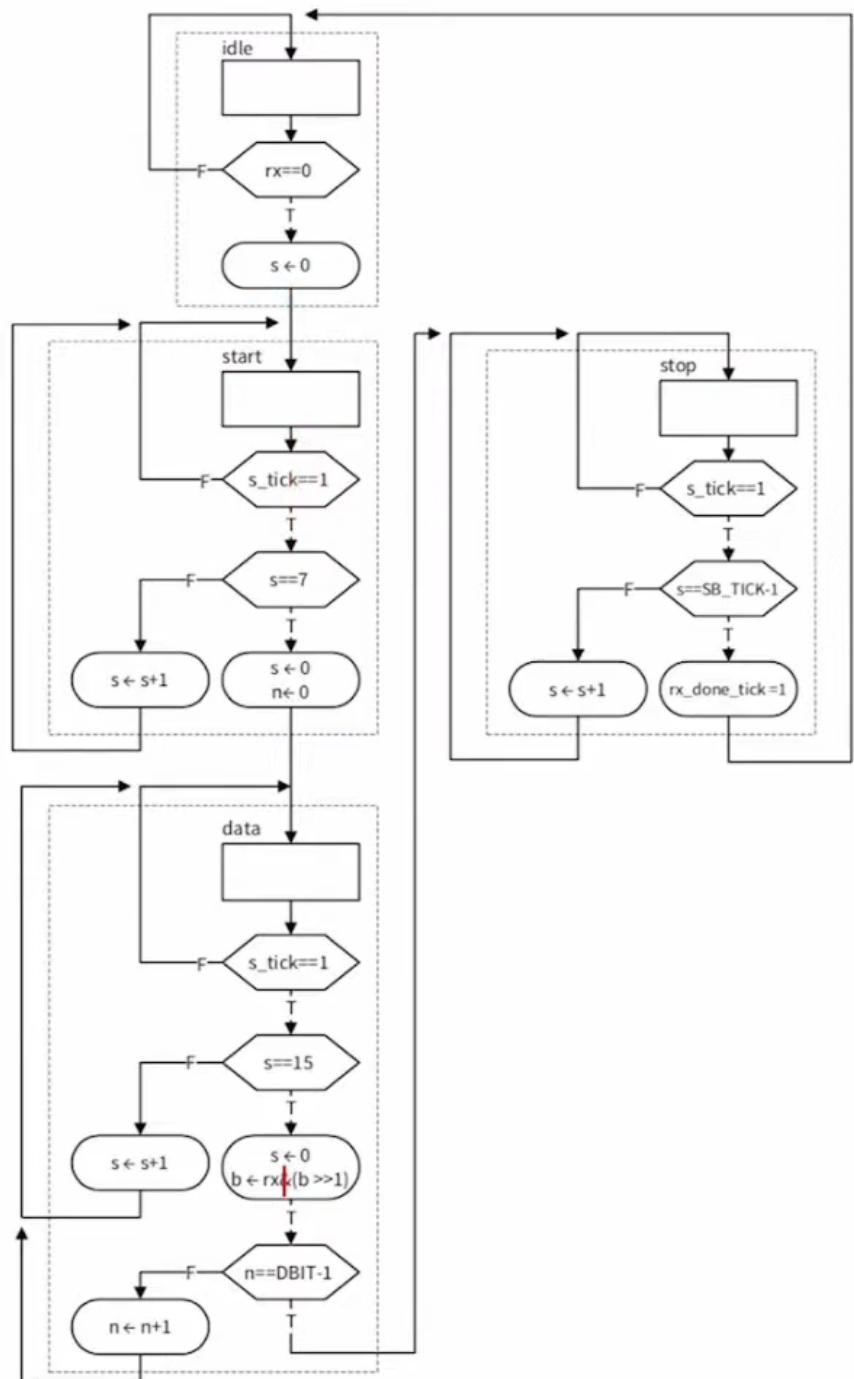
- $B$ : Tốc độ Baud mong muốn (ví dụ: 9600, 115200 bits/second).
- **16**: Hệ số lấy mẫu dư (Oversampling factor) cố định, đảm bảo bộ nhận UART lấy mẫu mỗi bit 16 lần để tối ưu hóa độ chính xác.

Ví dụ, với tần số hệ thống  $f_{clk} = 100$  MHz và tốc độ Baud yêu cầu  $B = 9600$ , giá trị  $v$  tính toán được xấp xỉ 650. Đối với cấu hình thực tế của đề tài tại tần số 200 MHz và tốc độ 115200 bps, giá trị này sẽ được phần mềm tính toán và nạp vào thanh ghi điều khiển tương ứng tại thời điểm khởi tạo hệ thống.

#### 6.4.1.2 Hiện thực máy trạng thái cho bộ nhận (RX) và bộ truyền (TX)

Logic truyền nhận vật lý được điều khiển bởi các máy trạng thái hữu hạn (FSM) trong module `uart_rx` và `uart_tx`.

Đối với bộ nhận, FSM hoạt động qua 4 trạng thái chính: *IDLE*, *START*, *DATA* và *STOP*. Khi phát hiện Start-bit, hệ thống sẽ đếm 7 xung tick để nhảy vào giữa bit và sau đó cứ mỗi 16 xung tick sẽ thực hiện lấy mẫu một bit dữ liệu. Quy trình này đảm bảo dữ liệu được thu thập tại thời điểm tín hiệu ổn định nhất.



**Hình 6.10:** Sơ đồ máy trạng thái (FSM) của bộ nhận UART RX

Tương tự, bộ truyền **uart\_tx** thực hiện quá trình chuyển đổi từ dữ liệu song song 8-bit sang khung truyền nối tiếp. Dữ liệu được đẩy ra chân tx kèm theo các bit khung (Start/Stop) theo đúng định thời được cấu hình, đảm bảo tính đồng bộ hoàn toàn với thiết bị đầu cuối.

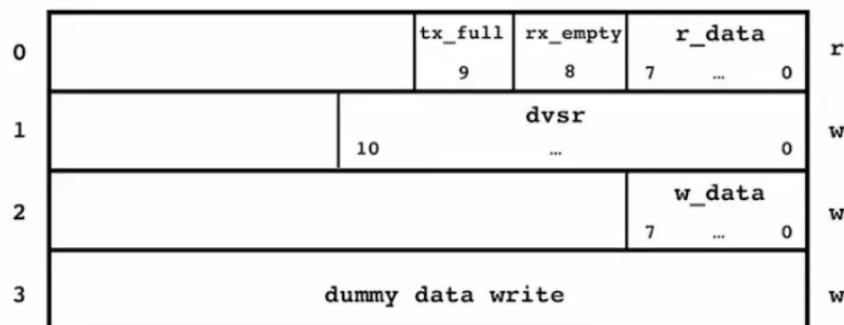
#### 6.4.1.3 Cơ chế đệm dữ liệu và quản lý dòng thông tin

Nhằm tối ưu hóa hiệu suất cho CPU, hệ thống tích hợp các khối **fifo\_unit** cho cả hai chiều truyền và nhận. Với độ sâu được thiết lập qua tham số **FIFO\_W**, các bộ đệm này cho phép CPU có thể nạp một chuỗi dữ liệu dài vào hàng đợi và tiếp tục thực thi các thuật toán AI mà không cần chờ đợi từng byte được truyền xong.

Sự hiện diện của FIFO giúp triệt tiêu hiện tượng mất dữ liệu (Data Overflow) khi vi xử lý đang bận xử lý các tác vụ ưu tiên cao, đồng thời tăng băng thông tổng thể cho hệ thống giao tiếp.

#### 6.4.1.4 Tích hợp giao diện AXI4-Lite

Việc nhúng khối UART vào hệ thống SoC được thực hiện thông qua module **uart\_axi\_lite**. Module áp dụng **axi\_lite\_slave\_interface** để ánh xạ các tài nguyên của UART thành các thanh ghi trong không gian địa chỉ của CPU.



Hình 6.11: Các thanh ghi kết nối giữa Bus AXI4-Lite và ngoại vi UART

Qua việc truy cập vào các địa chỉ Base Address đã quy hoạch trong Memory Map, phần mềm điều khiển có thể:

Đọc dữ liệu từ hàng đợi nhận hoặc ghi dữ liệu cần gửi vào hàng đợi truyền.

Kiểm tra các trạng thái hệ thống như **tx\_full** (đầy bộ đệm truyền) hoặc **rx\_empty** (hết dữ liệu nhận) để điều phối luồng chương trình.

Cấu hình lại tốc độ Baud linh hoạt ngay trong quá trình vận hành bằng cách ghi giá trị mới vào thanh ghi `dvsr`.

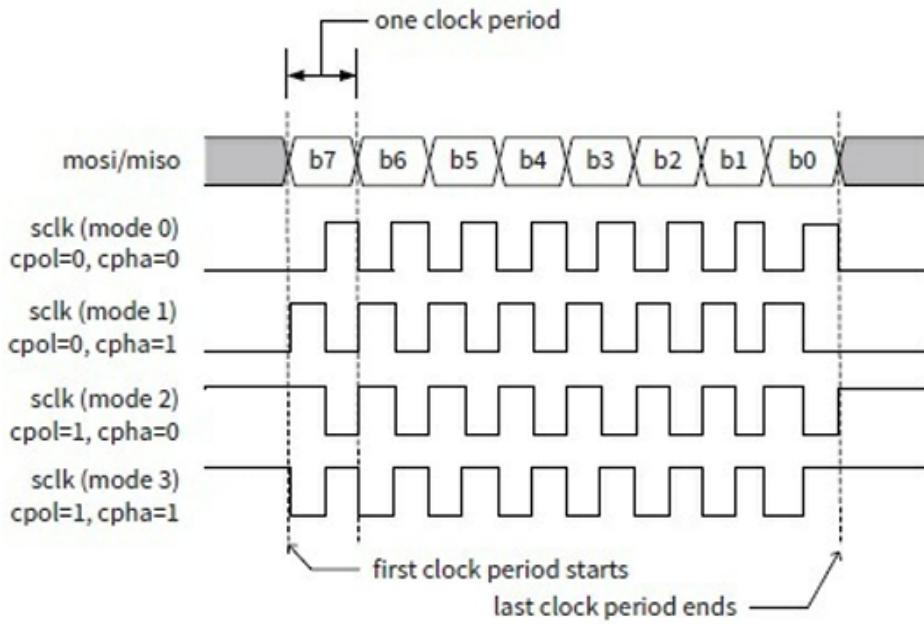
Kiến trúc này đảm bảo tính tách biệt hoàn toàn giữa logic bus và logic truyền nhận, giúp khôi ngoại vi hoạt động ổn định và dễ dàng tái sử dụng trong các thiết kế SoC khác nhau.

## 6.4.2 SPI

Bên cạnh UART, giao thức giao tiếp ngoại vi nối tiếp (SPI - Serial Peripheral Interface) đóng vai trò quan trọng trong việc kết nối SoC với các cảm biến tốc độ cao hoặc các bộ nhớ Flash phụ trợ. Khối IP SPI được thiết kế theo cấu trúc Master, cho phép lõi PicoRV32 chủ động điều phối luồng dữ liệu thông qua cơ chế truyền nhận đồng bộ.

### 6.4.2.1 Cấu trúc logic và bộ tạo xung nhịp SPI

Khác với UART sử dụng cơ chế lấy mẫu dữ, SPI là một giao thức đồng bộ nên yếu tố then chốt nằm ở việc tạo ra xung nhịp `sclk` ổn định. Module tích hợp một bộ chia tần số linh hoạt, cho phép phần mềm cấu hình tốc độ truyền thông dựa trên tần số hệ thống 200 MHz. Điểm đặc biệt trong hiện thực này là khả năng tùy biến các chế độ hoạt động (*SPI Modes*) thông qua việc điều chỉnh cực tính (`CPOL`) và pha của xung nhịp (`CPHA`), đảm bảo tính tương thích với đa dạng các chủng loại linh kiện ngoại vi trên thị trường.



**Hình 6.12:** SPI Mode

#### 6.4.2.2 Cơ chế điều khiển và Máy trạng thái FSMD

Hệ thống SPI Master được hiện thực hóa dựa trên kiến trúc **FSMD** (Finite State Machine with Datapath), nơi bộ điều khiển máy trạng thái (FSM) đóng vai trò chỉ đạo và khôi dữ liệu (Datapath) thực hiện các phép toán logic như dịch bit và đếm số bit đã truyền. Quy trình này được mô phỏng chi tiết dựa trên sơ đồ máy trạng thái và giản đồ định thời của giao thức SPI.

Hoạt động của FSM bao gồm các trạng thái chính sau:

**Trạng thái IDLE:** Đây là trạng thái mặc định của hệ thống. Tín hiệu chọn chip **ss** được giữ ở mức cao (*Logic 1*) để ngắt kết nối với các Slave ngoại vi. FSM liên tục giám sát tín hiệu khởi tạo từ Bus AXI; ngay khi có lệnh truyền, hệ thống sẽ nạp dữ liệu vào thanh ghi dịch và chuyển sang trạng thái kế tiếp.

**Trạng thái TRANSFER:** Tại đây, tín hiệu **ss** được kéo xuống thấp (*Logic 0*) để bắt đầu phiên làm việc. Khối Datapath bắt đầu vận hành

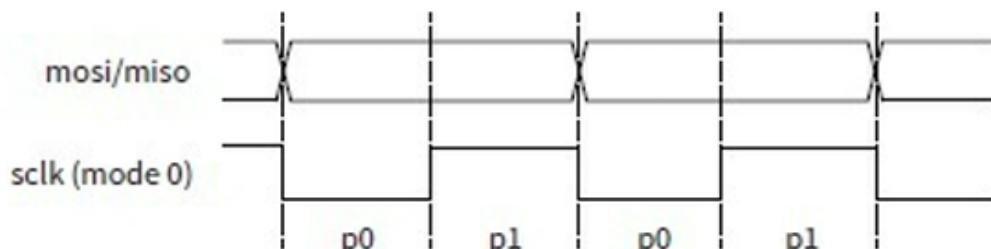
đồng bộ với xung nhịp `sclk`.

**Dịch bit dữ liệu (Shift-out):** Tại mỗi cạnh tích cực (hoặc cạnh dẫn tùy theo cấu hình CPOL/CPHA), bit có trọng số cao nhất (MSB) của thanh ghi dịch được đẩy ra chân `mosi`.

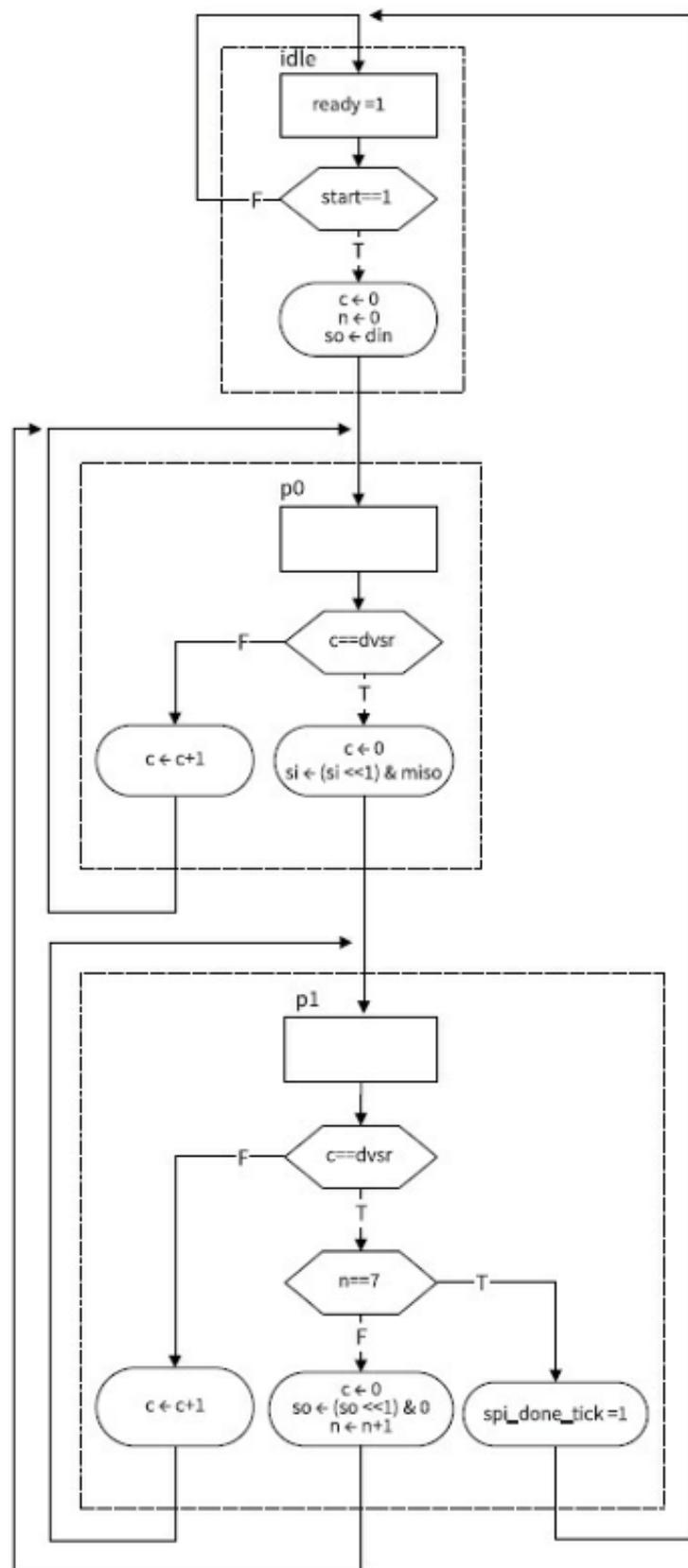
**Lấy mẫu dữ liệu (Sample-in):** Đồng thời, tín hiệu từ chân `miso` được lấy mẫu và đưa vào vị trí bit thấp nhất (LSB) của thanh ghi dịch.

**Bộ đếm bit:** Một bộ đếm nội bộ (*Bit Counter*) theo dõi số lượng xung nhịp. Sau khi hoàn tất đủ 8 chu kỳ (tương đương 1 byte dữ liệu), FSM sẽ quyết định kết thúc hoặc tiếp tục truyền byte tiếp theo.

**Trạng thái DONE/READY:** Sau khi trao đổi đủ dữ liệu, tín hiệu `ss` được đưa trở lại mức cao. Đồng thời, tín hiệu `ready` sẽ được kích hoạt ở mức logic 1 trong một chu kỳ xung nhịp hệ thống để báo hiệu cho vi xử lý PicoRV32 rằng dữ liệu trong thanh ghi đã sẵn sàng để đọc hoặc byte tiếp theo có thể được nạp vào.



**Hình 6.13:** Thời gian thực hiện thao tác dịch dữ liệu trong SPI



Hình 6.14: SPI controller ASMD chart

Sự phối hợp chặt chẽ giữa bộ đếm và thanh ghi dịch trong tầng Datapath giúp triệt tiêu các sai số về định thời. Việc sử dụng thanh ghi đếm tại chân miso trước khi nạp vào FSM đảm bảo rằng dữ liệu nhận được luôn ổn định, tránh các hiện tượng nhiễu tức thời tại cạnh xung nhịp, từ đó duy trì tính toàn vẹn của dữ liệu trong suốt quá trình giao tiếp tốc độ cao với các ngoại vi.

#### 6.4.2.3 Tích hợp hệ thống qua giao diện AXI4-Lite

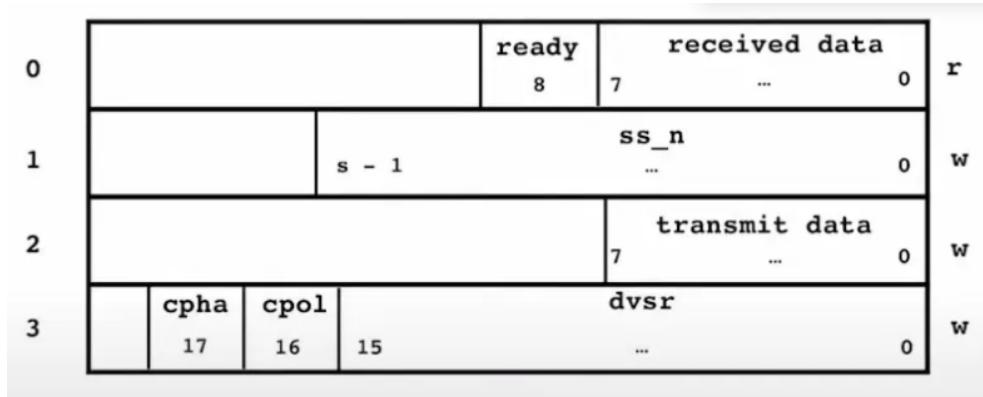
Khối SPI được nhúng vào không gian địa chỉ SoC thông qua module **Spi\_axi\_lite\_core**. Tương tự như khói UART, việc giao tiếp được thực hiện theo cơ chế nạp/lưu dữ liệu trực tiếp vào các thanh ghi chức năng (Memory-mapped I/O).

Giao diện **axi\_lite\_interface** đóng vai trò là tầng trung gian xử lý các giao thức bắt tay của bus. CPU quản lý ngoại vi SPI thông qua tập hợp các thanh ghi 32-bit sau:

**Thanh ghi dữ liệu (Data Register):** Chứa dữ liệu cần truyền đi hoặc dữ liệu vừa nhận về từ slave.

**Thanh ghi điều khiển (Control Register):** Cho phép cấu hình tốc độ xung nhịp sclk, chọn thiết bị tớ (Slave Select) và thiết lập các tham số truyền thông.

**Thanh ghi trạng thái (Status Register):** Cung cấp các cờ báo hiệu ready đang sẵn sàng.



**Hình 6.15:** Các thanh ghi kết nối giữa Bus AXI4-Lite và ngoại vi SPI

Việc sử dụng các thanh ghi đệm *D-Flip-Flop* tại tầng vật lý giúp cô lập hoàn toàn logic bus hệ thống với các tín hiệu I/O bên ngoài. Thiết kế này giúp triệt tiêu các vấn đề về nhiễu tín hiệu và đảm bảo rằng hệ thống Bus AXI vẫn duy trì được hiệu suất cao tại tần số 200 MHz trong khi khối SPI có thể hoạt động tại các dải tần số thấp hơn tùy theo yêu cầu của linh kiện ngoại vi.

### 6.4.3 I2C

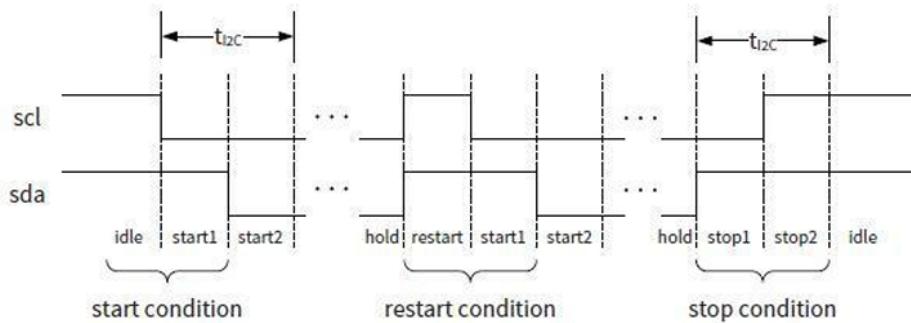
Trong hệ thống SoC đề xuất, giao thức I2C (Inter-Integrated Circuit) đóng vai trò là kênh điều khiển yếu để cấu hình các tham số vận hành cho cảm biến hình ảnh OV5640 và chip phát HDMI (ADV7513). Khối IP I2C Master được hiện thực hóa với khả năng điều khiển bus hai dây (**scl** và **sda**), hỗ trợ đa thiết bị tách thông qua cơ chế địa chỉ hóa và bắt tay chia sẻ.

#### 6.4.3.1 Kiến trúc bộ điều khiển I2C Master

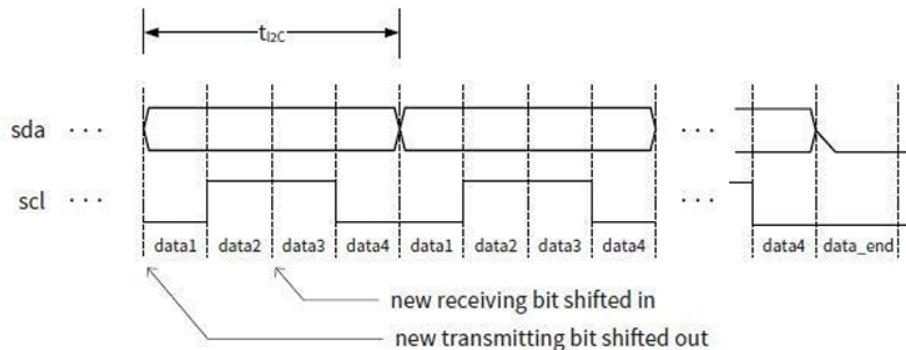
Thành phần hạt nhân của phân hệ là module **i2c\_master**, được thiết kế theo mô hình FSMD để quản lý các điều kiện phức tạp trên đường truyền. Khác với các giao thức nối tiếp khác, I2C sử dụng cấu trúc cực Open-drain, đòi hỏi bộ điều khiển phải quản lý linh hoạt trạng thái trớ kháng cao (*High-Impedance*) để cho phép các thiết bị tách phản hồi tín hiệu

Ack/Nack.

Để tạo ra xung nhịp **scl** từ nền tảng 200 MHz, module tích hợp một bộ chia tần số dựa trên thanh ghi **dvsr**. Tuy nhiên, do đặc thù của I2C yêu cầu duy trì các khoảng thời gian thiết lập (*Setup time*) và giữ (*Hold time*) khắt khe cho điều kiện Start và Stop, bộ tạo xung được chia thành 4 giai đoạn (*Quarters*) trong mỗi chu kỳ. Việc lấy mẫu tín hiệu **sda** luôn được thực hiện tại trung tâm của mức cao xung **scl** để đảm bảo tính ổn định tối đa của dữ liệu.



**Hình 6.16:** Chia các giai đoạn điều kiện trong chu kỳ xung nhịp I2C

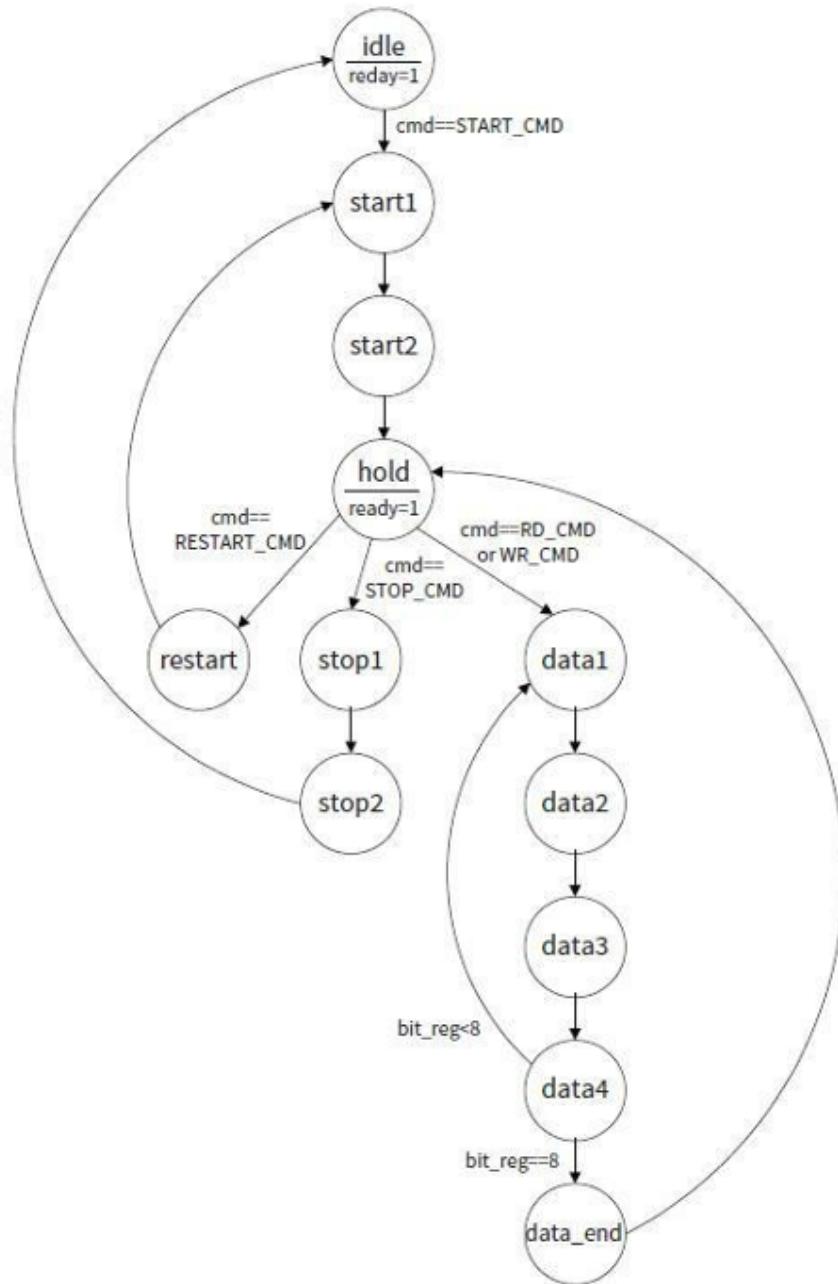


**Hình 6.17:** Chia các giai đoạn dữ liệu trong chu kỳ xung nhịp I2C

#### 6.4.3.2 Hiện thực Máy trạng thái điều khiển (Control FSM)

Máy trạng thái hữu hạn (FSM) là thành phần cốt lõi điều phối toàn bộ trình tự logic của khối I2C Master. FSM tiếp nhận tín hiệu đầu vào **cmd** để xác định loại tác vụ cần thực hiện, bao gồm: **RD\_CMD** (đọc dữ liệu), **WR\_CMD** (ghi dữ liệu), **START\_CMD** (tạo điều kiện bắt đầu), **RESTART\_CMD** (khởi động

lai giao dịch) và STOP\_CMD (tạo điều kiện kết thúc). Các trạng thái trong FSM được thiết kế để khớp nối chính xác với các pha thời gian của giao thức I2C, đảm bảo tính ổn định cho hai đường tín hiệu **scl** và **sda**.



**Hình 6.18:** Sơ đồ các trạng thái của bộ điều khiển I2C Master

Quy trình vận hành của FSM được hiện thực hóa qua các giai đoạn cụ thể như sau:

**Khởi tạo giao dịch (Start Phase):** Khi nhận lệnh START\_CMD, FSM sẽ chuyển lần lượt qua các trạng thái start1 và start2. Tại đây, bộ điều khiển sẽ tạo ra điều kiện Start bằng cách điều phối mức logic trên các chân I/O để thông báo phiên giao dịch mới cho các thiết bị trên bus.

**Xử lý dữ liệu và bit xác nhận (Data Phase):** Sau khi thiết lập điều kiện Start, các lệnh WR\_CMD hoặc RD\_CMD sẽ được thực thi. Cả hai tác vụ này đều đi qua chu trình gồm 4 trạng thái: data1, data2, data3 và data4.

Chu trình này được lặp lại 9 lần cho mỗi byte dữ liệu (bao gồm 8 bit dữ liệu chính và 1 bit xác nhận ACK/NACK).

Một bộ đếm nội bộ (bit\_reg) được sử dụng để theo dõi số lần lặp và đảm bảo trình tự dịch bit diễn ra chính xác.

Sau khi xử lý đủ 9 bit, FSM chuyển sang trạng thái data\_end để kéo thấp các đường scl và sda trong một khoảng thời gian tương ứng với 1/4 chu kỳ xung nhịp trước khi chuyển về trạng thái giữ.

**Duy trì trạng thái (Hold State) và Phản hồi Ready:** Sau khi hoàn tất một lệnh đơn lẻ, FSM sẽ duy trì ở trạng thái hold để chờ đợi chỉ thị tiếp theo từ vi xử lý. Ở trạng thái này, cả scl và sda đều được giữ ở mức thấp. Tín hiệu trạng thái ready sẽ được kích hoạt tại các trạng thái idle hoặc hold để báo hiệu cho lõi PicoRV32 rằng bộ điều khiển đã sẵn sàng nhận lệnh mới.

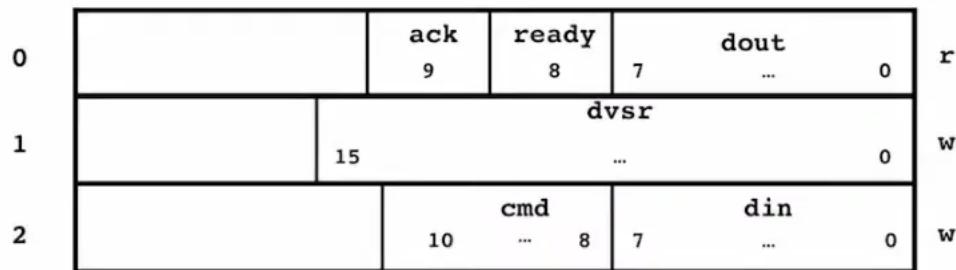
**Kết thúc hoặc Khởi động lại (Stop/Restart Phase):** Giao dịch được kết thúc bằng lệnh STOP\_CMD, buộc FSM đi qua các trạng thái stop1 và stop2 để tạo điều kiện Stop. Nếu nhận lệnh RESTART\_CMD, FSM sẽ trực tiếp tái tạo lại điều kiện Start để bắt đầu một phiên truy

cập mới mà không giải phóng bus, giúp tối ưu hóa thời gian truyền tải khi giao tiếp với đa thiết bị.

Cơ chế phân tách các pha xung nhịp thành các trạng thái FSM nhỏ lẻ giúp hệ thống kiểm soát chặt chẽ các thông số thời gian (*setup/hold time*), đảm bảo sự tương thích hoàn toàn với các thiết bị ngoại vi như cảm biến camera OV5640 tại tần số vận hành của SoC.

#### 6.4.3.3 Ánh xạ thanh ghi và tích hợp AXI4-Lite

Khối I2C được nhúng vào hạ tầng Bus hệ thống thông qua module **I2C\_axi\_lite\_core**. Module này đóng vai trò cầu nối, chuyển đổi các giao dịch bus AXI thành các xung lệnh nạp dữ liệu (**din**) và mã lệnh (**cmd**) cho lõi I2C Master.



**Hình 6.19:** Các thanh ghi kết nối giữa Bus AXI4-Lite và ngoại vi I2C

Hệ thống thanh ghi điều khiển được quy hoạch theo cơ chế MMIO, cho phép vi xử lý PicoRV32 quản lý ngoại vi thông qua các tác vụ phần mềm đơn giản:

**Thanh ghi Data/Command:** Kết hợp giữa dữ liệu 8-bit và các bit lệnh (Start, Stop, Write, Read). Việc ghi vào thanh ghi này sẽ kích hoạt FSM bắt đầu vận hành.

**Thanh ghi Divisor (dvsrc):** Lưu trữ giá trị bộ chia xung nhịp, cho phép thiết lập tốc độ bus ở chế độ Standard (100 kbps) hoặc Fast mode (400 kbps) linh hoạt tại thời điểm chạy.

**Thanh ghi Status:** Cung cấp thông tin thời gian thực về trạng thái ready (sẵn sàng tiếp nhận lệnh mới) và kết quả của bit ack gần nhất.

Sự phối hợp giữa kiến trúc phần cứng tối ưu và lớp giao diện AXI chuẩn hóa giúp khôi ngoại vi I2C vận hành tin cậy tại tần số cao, đồng thời đơn giản hóa việc viết trình điều khiển (*Driver*) cho cảm biến camera và hệ thống hiển thị trong các giai đoạn phát triển phần mềm ứng dụng.

#### 6.4.4 OSPI-DDR

Trong các hệ thống SoC hiện đại, nhu cầu về băng thông bộ nhớ mở rộng là cực kỳ cấp thiết, đặc biệt khi hệ thống cần xử lý các thuật toán mạng nơ-ron hoặc lưu trữ lượng lớn frame hình ảnh từ camera. Để giải quyết bài toán này, đề tài hiện thực hóa khôi ngoại vi **Octal-SPI (OSPI)** tích hợp công nghệ **DDR (Double Data Rate)**, cho phép tối ưu hóa tốc độ truyền tải dữ liệu giữa FPGA và các chip Flash/RAM ngoại vi.

##### 6.4.4.1 Ý tưởng thiết kế

Thành phần chính của thiết kế là module **ospi\_ddr**, sự kết hợp giữa SPI và I2C thì module này mở rộng xây dựng dựa trên cấu trúc máy trạng thái hữu hạn kết hợp với các khối I/O chuyên dụng của FPGA. Để thực hiện được việc dịch bit tại cả hai cạnh xung nhịp, tầng vật lý sử dụng các nguyên lý chuyển đổi dữ liệu tốc độ cao, đảm bảo pha của xung nhịp và dữ liệu luôn được căn chỉnh chính xác.

Hệ thống điều khiển bao gồm các khối chức năng chính: bộ chia tần số linh hoạt để tạo xung nhịp giao tiếp từ nền tảng 200 MHz, khối logic nạp/dịch dữ liệu 8-bit và bộ điều phối tín hiệu chọn chip (**c<sub>sn</sub>**). Máy trạng thái điều khiển được thiết kế để quản lý các pha giao thức phức tạp của OSPI, bao gồm pha gửi mã lệnh (Instruction), pha địa chỉ (Address) và pha dữ liệu (Data), trong đó mỗi pha có thể được cấu hình chạy ở chế độ SDR hoặc DDR tùy theo đặc tính của linh kiện ngoại vi kết nối.

#### 6.4.4.2 Tích hợp hệ thống và giao diện AXI4-Lite

Khối OSPI-DDR được nhúng vào không gian địa chỉ của vi xử lý thông qua module **OSPI\_axi\_lite\_core**. Module này đóng vai trò là một tầng cầu nối giữa giao thức bus AXI4-Lite tiêu chuẩn và logic điều khiển OSPI nội bộ. Việc tích hợp module **axi\_lite\_slave\_interface** giúp chuẩn hóa quy trình bắt tay dữ liệu, đảm bảo vi xử lý PicoRV32 có thể truy xuất bộ nhớ ngoài thông qua các lệnh nạp/lưu (Load/Store) đơn giản.

0		cmd_addr_upper								w
1		cmd_addr_lower								w
2		capture_shmoo								w
3		start	rd	wr	lentency	burst_len				w
4		start_rdy	empty	full	7	rdata	0			r

**Hình 6.20:** Các thanh ghi kết nối giữa Bus AXI4-Lite và ngoại vi OSPI

Cấu trúc thanh ghi điều khiển được quy hoạch theo cơ chế Memory-mapped I/O bao gồm:

**Thanh ghi Dữ liệu (Data Register):** Đây là thanh ghi trung tâm thực hiện nhiệm vụ truyền nhận dữ liệu.

**Thanh ghi Cấu hình Chế độ (Configuration Register):** Thanh ghi này cho phép người dùng tùy chỉnh các thông số của transaction.

**Thanh ghi Trạng thái (Status Register):** Cung cấp các cờ báo hiệu thời gian thực về tiến trình của bộ điều khiển vật lý.

Việc hiện thực hóa khối OSPI-DDR không chỉ giúp SoC mở rộng được không gian lưu trữ mã lệnh chương trình (XIP - Execute In Place) mà còn tạo ra một hạ tầng dữ liệu vững chắc cho các tác vụ xử lý AI đòi hỏi bộ nhớ đệm lớn, đảm bảo tính ổn định và hiệu năng cao cho toàn bộ hệ thống SoC.

## 6.5 Video Streaming

Video Streaming đóng vai trò then chốt trong việc thu nhận, lưu trữ tạm thời và hiển thị hình ảnh từ cảm biến camera đến thiết bị đầu cuối. Hệ thống được thiết kế theo cấu trúc Pipeline (đường ống) nhằm tối ưu hóa băng thông và đảm bảo luồng dữ liệu hình ảnh được truyền tải với độ trễ thấp nhất, phục vụ các tác vụ xử lý ảnh thời gian thực.

### 6.5.1 Kiến trúc tổng thể của Video Pipeline

Hệ thống xử lý video trong SoC được tổ chức thành bốn tầng chức năng chính, phối hợp chặt chẽ để đảm bảo luồng dữ liệu hình ảnh được xử lý ổn định và hiệu quả:

**Tầng thu nhận dữ liệu vật lý (*Capture Layer*):** Thu nhận tín hiệu từ giao diện DVP, thực hiện ghép byte và đồng bộ khung hình từ cảm biến camera.

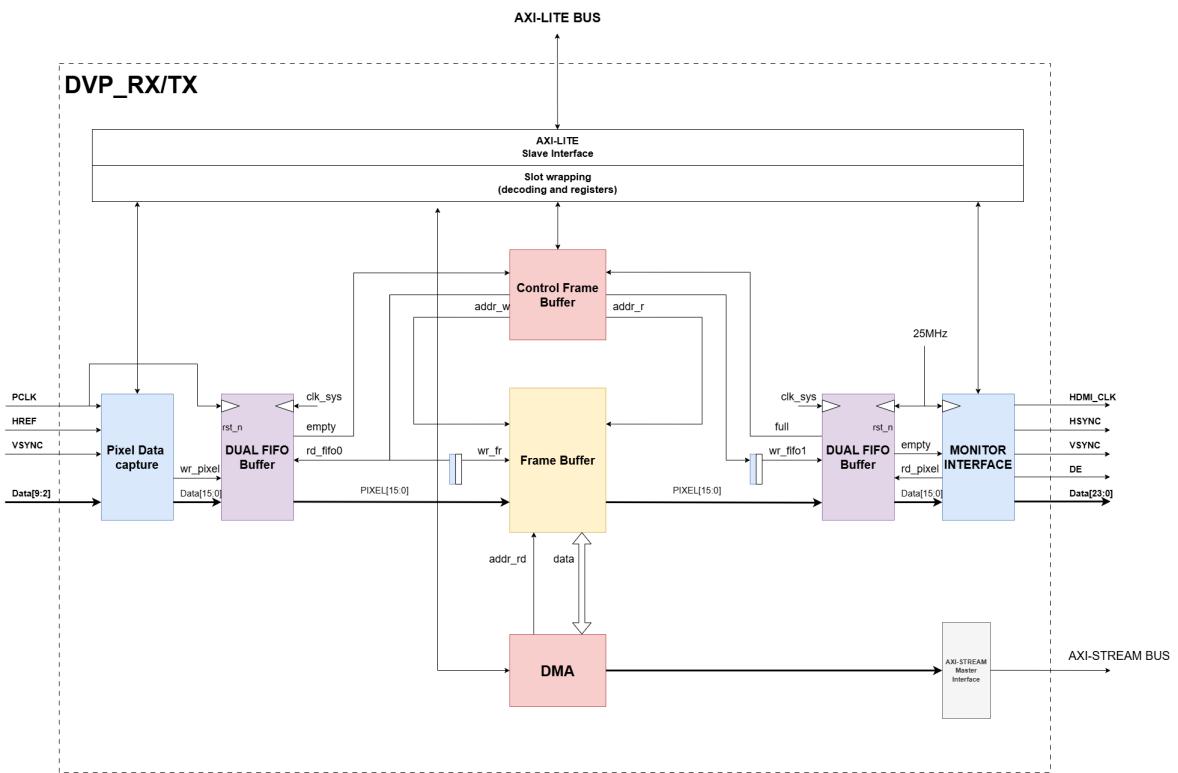
**Tầng quản lý bộ đệm (*Memory Buffer Layer*):** Đóng vai trò là trung tâm lưu trữ tạm thời, sử dụng bộ điều khiển để quản lý việc truy cập vào vùng nhớ BRAM thông qua các giao dịch Burst tốc độ cao, giúp triệt tiêu sự lệch pha giữa tốc độ camera và tốc độ hiển thị.

**Tầng truyền tải dữ liệu nội bộ (*DMA Data Transfer Layer*):** Đây là tầng chức năng quan trọng cho phép hệ thống tự động trích xuất dữ liệu từ *Frame Buffer* để truyền tải đến các tài nguyên khác trong SoC. Cơ chế này tận dụng khả năng truy cập trực tiếp bộ nhớ (*Direct Memory Access*) để đẩy dữ liệu hình ảnh vào bộ gia tốc AI hoặc các kênh xử lý tín hiệu số mà không làm tiêu tốn chu kỳ xử lý của lõi PicoRV32, từ đó tối ưu hóa hiệu năng tổng thể của hệ thống.

**Tầng điều khiển hiển thị (*Display Controller Layer*):** Đảm

bảo dữ liệu pixel được xuất ra màn hình thông qua giao diện HDMI một cách mượt mà và không bị nhiễu.

Sự phối hợp giữa bốn tầng chức năng này không chỉ đảm bảo dữ liệu từ cảm biến được đồng bộ hóa chính xác mà còn tạo ra một hạ tầng dữ liệu linh hoạt, sẵn sàng cung cấp thông tin hình ảnh cho các thuật toán xử lý thông minh trong các bước tiếp theo.



Hình 6.21: Sơ đồ khái niệm Video Streaming tích hợp trong SoC

### 6.5.2 Khối thu nhận hình ảnh từ cảm biến OV5640 (DVP Interface)

Để tài sử dụng cảm biến hình ảnh **OV5640** giao tiếp thông qua giao diện video kỹ thuật số song song **DVP** (Digital Video Port). Khối logic thu nhận thực hiện nhiệm vụ giải mã các tín hiệu định thời từ camera để trích xuất dữ liệu pixel hợp lệ.

Quy trình thu nhận dựa trên việc giám sát ba tín hiệu xung nhịp và đồng

bộ:

**VSYNC (Vertical Sync):** Đánh dấu thời điểm bắt đầu một khung hình mới.

**HREF (Horizontal Reference):** Xác định vùng dữ liệu pixel hợp lệ trên mỗi dòng quét.

**PCLK (Pixel Clock):** Xung nhịp đồng bộ để lấy mẫu dữ liệu trên bus 8-bit ( $D[9 : 2]$ ).

### 6.5.3 Cơ chế quản lý bộ đệm khung hình (Frame Buffer)

Do sự mất cân bằng về tốc độ giữa tần số lấy mẫu của camera và tần số quét của màn hình hiển thị, hệ thống sử dụng một bộ đệm khung hình (**Frame Buffer**) đóng vai trò vùng nhớ đệm trung gian. Khối này được hiện thực hóa bằng bộ nhớ *BRAM* với cấu trúc hai cổng (Dual-port) độc lập.

Cơ chế vận hành của Frame Buffer bao gồm:

**Luồng Ghi (Write Path):** Đồng bộ với xung nhịp PCLK, thực hiện việc ghi dữ liệu pixel từ khói Capture vào bộ nhớ dựa trên địa chỉ được tạo ra từ bộ đếm tọa độ dòng/cột.

**Luồng Đọc (Read Path):** Đồng bộ với xung nhịp hiển thị (*Pixel Clock*), truy xuất dữ liệu pixel dựa trên yêu cầu từ bộ tạo định thời hiển thị.

### 6.5.4 Điều khiển hiển thị và Giao diện HDMI (ADV7513)

Tầng cuối cùng của Pipeline là bộ điều khiển hiển thị, chịu trách nhiệm tạo ra các tín hiệu đồng bộ chuẩn VGA/HD để giao tiếp với chip phát **ADV7513**.

Bộ tạo định thời Video thực hiện tính toán các khoảng thời gian quét bao gồm: *Active Video*, *Front Porch*, *Sync Pulse* và *Back Porch* cho cả hai chiều ngang (Horizontal) và dọc (Vertical).

**HSYNC/VSYNC:** Điều khiển quá trình quét dòng và quét khung hình trên màn hình.

**Data Enable (DE):** Tín hiệu kích hoạt cho phép chip ADV7513 lấy mẫu dữ liệu pixel từ bus dữ liệu để chuyển đổi sang định dạng TMDS truyền qua cổng HDMI.

Việc cấu hình các tham số cho cảm biến OV5640 và chip ADV7513 được thực hiện thông qua khối ngoại vi **I2C** đã trình bày ở phần trước.

# Chương 7

## Ước lượng hiệu năng

Chương này trình bày các kết quả thu được từ mô hình ước lượng hiệu năng của kiến trúc phần cứng AI Accelerator 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

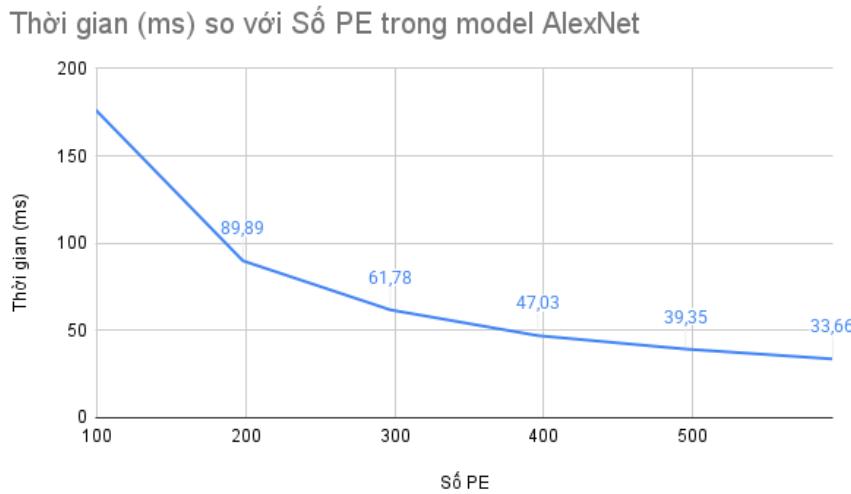
Nhằm đánh giá để tiên liệu trước về hiệu năng của đồ án này, kiến trúc được đánh giá thông qua một mô hình ước lượng (Analytical Estimator) xây dựng bằng ngôn ngữ C++. Mô hình này hiện thực hóa các công thức toán học đã được thiết lập tại Chương 4 nhằm dự báo độ trễ và tài nguyên tiêu thụ.

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. 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 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.

## 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 ( $11 \times 11$ ,  $5 \times 5$ ). 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  $PE = 396$  với độ trễ (latency) là 47.04 ms, tương đương tốc độ

khung hình 21.26 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 572 PEs được trình bày tại Bảng 7.1.

**Bảng 7.1:** Chi tiết hiệu năng từng lớp của AlexNet (Cập nhật)

Layer	Filter / Channels	Map Size	PE Used	Optimized Config			Latency (ms)	Bottleneck
				$T_k$	$T_m$	$T_h$		
Conv1	$11 \times 11/96$	$224 \times 224$	396	3	12	1	8.12	Mixed
Conv2	$5 \times 5/256$	$27 \times 27$	180	3	12	1	13.60	Mixed
Conv3	$3 \times 3/384$	$13 \times 13$	108	3	12	1	7.25	Memory
Conv4	$3 \times 3/384$	$13 \times 13$	108	3	12	1	10.71	Memory
Conv5	$3 \times 3/256$	$13 \times 13$	108	3	12	1	7.36	Memory
<b>Total</b>	—	—	<b>Max 396</b>	—	—	—	<b>47.04</b>	<b>Memory</b>

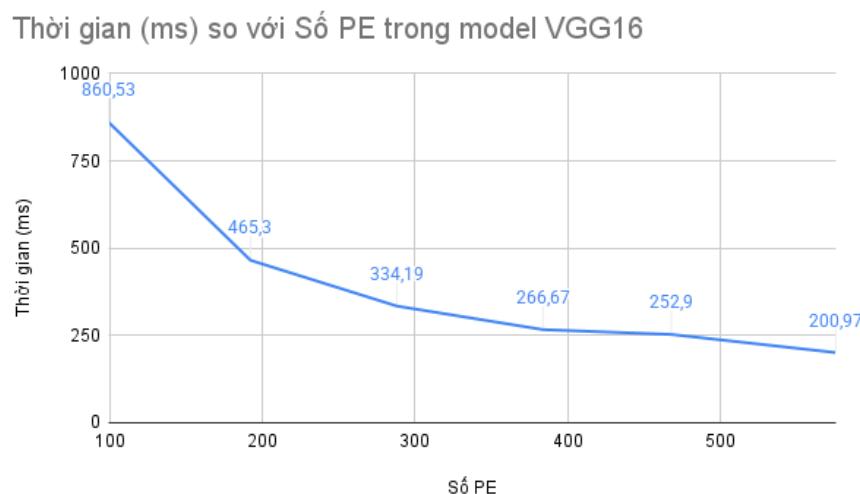
Dựa trên kết quả thực nghiệm tại Bảng 7.1, mô hình AlexNet đạt tổng thời gian thực thi là 47,04 ms. Tại hai lớp đầu tiên (*Conv1* và *Conv2*), hệ thống ghi nhận trạng thái *Mixed Bound*, cho thấy hiệu năng đang bị kìm hãm đồng thời bởi năng lực tính toán của mảng PE và tốc độ đáp ứng của băng thông bộ nhớ. Trong đó, *Conv2* có độ trễ cao nhất (13,60 ms) do phải xử lý khối lượng tính toán lớn cùng áp lực truyền tải dữ liệu cao.

Từ lớp *Conv3* đến *Conv5*, nút thắt cỗ chai chuyển dịch hoàn toàn sang *Memory Bound*. Mặc dù số lượng PE huy động thực tế giảm xuống, tốc độ truy xuất từ bộ nhớ ngoài vẫn không kịp cung cấp dữ liệu cho các đơn vị tính toán, dẫn đến việc lãng phí tài nguyên phần cứng. Tổng kết lại, thực nghiệm cho thấy hệ thống đang gặp thách thức về băng thông (*Memory Wall*), do đó các giải pháp tối ưu trong tương lai cần tập trung vào việc tăng cường khả năng tái sử dụng dữ liệu (*data reuse*) thay vì mở rộng quy

mô mảng PE.

### 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  $PE = 384$  với độ trễ đạt 266.67 ms, tương đương 3.75 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 dữ liệu 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

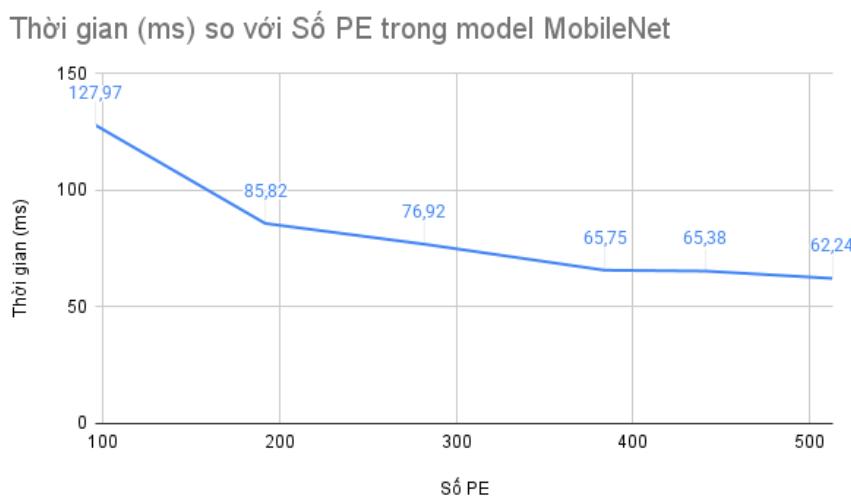
Layer	Filter / Channels	Map Size	PE Used	Optimized Config			Latency (ms)	Bottleneck
				$T_k$	$T_m$	$T_h$		
Conv1_1	$3 \times 3/64$	$224 \times 224$	384	2	64	1	17.07	Mixed
Conv1_2	$3 \times 3/64$	$224 \times 224$	384	2	64	1	39.90	Mixed
Conv2_1	$3 \times 3/128$	$112 \times 112$	384	2	64	1	19.95	Mixed
Conv2_2	$3 \times 3/128$	$112 \times 112$	384	2	64	1	31.99	Mixed
Conv3_1	$3 \times 3/256$	$56 \times 56$	384	2	64	1	16.00	Mixed
Conv3_2	$3 \times 3/256$	$56 \times 56$	384	2	64	1	28.04	Mixed
Conv3_3	$3 \times 3/256$	$56 \times 56$	384	2	64	1	28.04	Mixed
Conv4_1	$3 \times 3/512$	$28 \times 28$	384	2	64	1	14.02	Mixed
Conv4_2	$3 \times 3/512$	$28 \times 28$	384	2	64	1	26.06	Mixed
Conv4_3	$3 \times 3/512$	$28 \times 28$	384	2	64	1	26.06	Mixed
Conv5_1	$3 \times 3/512$	$14 \times 14$	384	2	64	1	6.52	Mixed
Conv5_2	$3 \times 3/512$	$14 \times 14$	384	2	64	1	6.52	Mixed
Conv5_3	$3 \times 3/512$	$14 \times 14$	384	2	64	1	6.52	Mixed
<b>Total</b>	—	—	<b>Max 384</b>	—	—	—	<b>266.66</b>	Mixed

Kết quả thực nghiệm tại Bảng 7.2 cho thấy mô hình VGG-16 đạt tổng thời gian thực thi là 266,66 ms với cấu hình phần cứng huy động ổn định 384 PE. Điểm đặc biệt toàn bộ các lớp từ *Conv1\_1* đến *Conv5\_3* đều duy trì trạng thái *Mixed Bound*. Điều này chứng tỏ cấu hình tối ưu hóa  $T_m = 64$  đã tạo ra sự cân bằng nhất định giữa năng lực tính toán và lưu lượng dữ liệu, khiến hệ thống bị giới hạn đồng thời bởi cả băng thông bộ nhớ và tốc độ xử lý của mảng PE trên mọi tầng kiến trúc.

Dộ trễ có xu hướng giảm dần tại các tầng sâu hơn (từ 39,90 ms ở *Conv1\_2* xuống còn 6,52 ms ở các lớp *Conv5*), hệ quả trực tiếp của việc thu nhỏ kích thước bản đồ đặc trưng (*Map Size*) dù số lượng kênh tăng mạnh. Việc duy trì số lượng PE hoạt động tối đa xuyên suốt các lớp cho thấy hiệu suất khai thác phần cứng của thiết kế đối với VGG-16 là rất cao. Tuy nhiên, để tối ưu hóa thêm tổng thời gian thực thi, cần có những điều chỉnh linh hoạt hơn về tham số  $T_k$  và  $T_m$  nhằm thoát khỏi trạng thái nghẽn hõn hợp tại các lớp có khối lượng tính toán lớn ở giai đoạn 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 65.75 ms, tương đương 15.2 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 cầu hình phần cứng tối ưu với giới hạn 385 PEs được trình bày tại Bảng 7.3.

**Bảng 7.3:** Chi tiết hiệu năng từng lớp của MobileNet v1

Layer	Filter / Channels	Map Size	PE Used	Optimized Config			Latency (ms)	Bottleneck
				T <sub>k</sub>	T <sub>m</sub>	T <sub>h</sub>		
Standard Conv (L0)	$3 \times 3/32$	$224 \times 224$	96	1	32	1	3.39	Mixed
Depthwise (L1)	$3 \times 3/32$	$112 \times 112$	3	1	1	1	4.02	Mixed
Pointwise (L2)	$1 \times 1/64$	$112 \times 112$	64	1	64	1	6.02	Memory
Depthwise (L3)	$3 \times 3/64$	$56 \times 56$	3	1	1	23	6.70	Mixed
Pointwise (L4)	$1 \times 1/128$	$56 \times 56$	128	1	128	1	3.01	Memory
Depthwise (L5)	$3 \times 3/128$	$56 \times 56$	3	1	1	1	4.02	Mixed
Pointwise (L6)	$1 \times 1/128$	$56 \times 56$	128	1	128	1	4.01	Memory
Depthwise (L7)	$3 \times 3/128$	$28 \times 28$	3	1	1	56	2.57	Mixed
Pointwise (L8)	$1 \times 1/256$	$28 \times 28$	128	1	128	1	2.01	Memory
Depthwise (L9)	$3 \times 3/256$	$28 \times 28$	3	1	1	1	2.01	Mixed
Pointwise (L10)	$1 \times 1/256$	$28 \times 28$	128	1	128	1	3.01	Memory
Depthwise (L11)	$3 \times 3/256$	$14 \times 14$	3	1	1	28	1.27	Mixed
Pointwise (L12)	$1 \times 1/512$	$14 \times 14$	128	1	128	1	1.51	Memory
Avg. DW (L13-25 odd)	$3 \times 3/512$	$14 \times 14$	3	1	1	1*	0.86 <sup>†</sup>	Mixed
Avg. PW (L14-26 even)	$1 \times 1/512$	$14 \times 14$	128	1	128	1	2.30 <sup>†</sup>	Memory
<b>Total</b>	—	—	<b>Max 128</b>	—	—	—	<b>65.75</b>	<b>Mixed</b>

\* Giá trị  $T_h$  thay đổi tùy lớp. <sup>†</sup> Giá trị trung bình dựa trên tổng chu kỳ thực tế của các lớp lặp lại (L13-L26).

Dựa trên kết quả thực nghiệm tại Bảng 7.3, mô hình MobileNet v1 đạt tổng thời gian thực thi là 65,75 ms với mảng PE huy động tối đa 128 đơn vị. Hệ thống ghi nhận sự phân hóa rõ rệt về nút thắt cổ chai giữa hai loại hình tích chập đặc trưng: các lớp *Pointwise* luôn ở trạng thái *Memory Bound*, trong khi các lớp *Depthwise* và *Standard Conv* duy trì trạng thái *Mixed Bound*.

Tại các lớp *Pointwise*, dù đã tận dụng tối đa cấu hình song song  $T_m = 128$ , hiệu năng vẫn bị giới hạn bởi tốc độ truy xuất dữ liệu từ bộ nhớ ngoài do khối lượng tham số và đặc trưng lớn. Ngược lại, các lớp *Depthwise* bộc lộ sự hạn chế trong việc khai thác tài nguyên khi chỉ huy động được 3 PE ( $T_m = 1$ ) do đặc thù tính toán đơn kênh, dẫn đến việc năng lực tính toán và băng thông rơi vào trạng thái nghẽn hõn hợp với hiệu suất sử dụng phần cứng thấp. Tổng kết lại, kết quả thực nghiệm chỉ ra rằng hiệu năng của MobileNet v1 trên kiến trúc hiện tại bị kìm hãm đồng thời bởi giới hạn băng thông bộ nhớ tại các lớp tích chập điểm và sự thiếu tương thích về

cấu trúc song song tại các lớp tích chập sâu.

## 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 đặc 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 (AlexNet) - 162 (VGG16)	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>	71.24 ms (14.03 fps)	<b>28.57 ms*</b> (35.0 fps)
<b>VGG16 (Latency)</b>	<b>555.8 ms</b> (1.80 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 ( $N = 3$ ).*

Dựa trên kết quả đối sánh chi tiết tại Bảng 7.4, kiến trúc đề xuất thể hiện những ưu thế chiến lược về khả năng xử lý thời gian thực và hiệu quả quản lý băng thông so với kiến trúc Eyeriss. Sự khác biệt rõ rệt nhất được ghi nhận tại mô hình VGG16, nơi giải pháp của nhóm nghiên cứu đạt hiệu năng vượt trội với độ trễ thấp hơn khoảng **2,59 lần** (chỉ 555,8 ms so với 1428,57 ms của Eyeriss). Kết quả đột phá này đạt được nhờ việc áp dụng

kiến trúc bộ nhớ tách biệt (*Separate Off-chip Memory*), giúp mở rộng băng thông truy xuất dữ liệu độc lập cho trọng số và bản đồ đặc trưng. Cách tiếp cận này giúp giải tỏa triệt để hiện tượng nghẽn mạch bộ nhớ (*Memory Bottleneck*) — một hạn chế cốt yếu của các kiến trúc sử dụng bộ nhớ chia sẻ (*Shared DRAM*) khi phải đối mặt với khối lượng dữ liệu không lồ của các mạng nơ-ron sâu như VGG16.

Đối với mô hình AlexNet, mặc dù Eyeriss đạt thông lượng cao hơn (35,0 fps) nhờ tận dụng cơ chế xử lý theo lô (*Batch Size = 4*), kiến trúc đề xuất vẫn khẳng định được giá trị thực tiễn với độ trễ phản hồi đơn lẻ đạt 71,24 ms tại  $N = 1$ . Trong khi Eyeriss ưu tiên tối ưu hóa hiệu suất tổng thể (*Throughput*) dựa trên việc tích lũy dữ liệu, giải pháp của nhóm hướng tới các ứng dụng thực tế yêu cầu phản hồi tức thời (*Real-time*). Việc loại bỏ độ trễ tích lũy (*Batching Latency*) giúp hệ thống đề xuất trở nên phù hợp hơn cho các bài toán biên (*Edge computing*), nơi tính thời điểm của thông tin quan trọng hơn lưu lượng xử lý tổng thể. Tổng kết lại, sự kết hợp giữa mảng PE linh hoạt và kiến trúc bộ nhớ tối ưu đã giúp hệ thống đề xuất đạt được sự cân bằng hiệu quả giữa năng lực tính toán và tốc độ phản hồi trên các cấu trúc mạng có độ phức tạp khác nhau.

# Chương 8

## Kết quả thực nghiệm

*Trong chương này, đề tài trình bày các kết quả đạt được sau quá trình thiết kế, tổng hợp và triển khai hệ thống SoC trên nền tảng phần cứng thực tế. Nội dung bao gồm đánh giá tài nguyên sử dụng, kết quả kiểm tra các khối ngoại vi, hiệu năng của Video Streaming và quy trình khởi động hệ thống thông qua Bootloader.*

### 8.1 Môi trường thực nghiệm phần cứng

Toàn bộ hệ thống SoC được triển khai và đánh giá trên kit phát triển **Xilinx Virtex-7 VC707 FPGA** và **Arty A7-100T Artix-7 FPGA**. Các FPGA này đều cung cấp đầy đủ các giao diện cần thiết cho việc thực nghiệm.

Các thông số thiết lập xung nhịp hệ thống bao gồm:

Xung nhịp hệ thống chính (`sys_clk`): 200 MHz đối với VC707 và 100 MHz đối với Arty A7.

Xung nhịp chính cho khối Video Streaming là 150 MHz trên VC707, còn Arty A7 không đủ bộ nhớ Bram nên sẽ không triển khai khối Video Streaming trên đây.

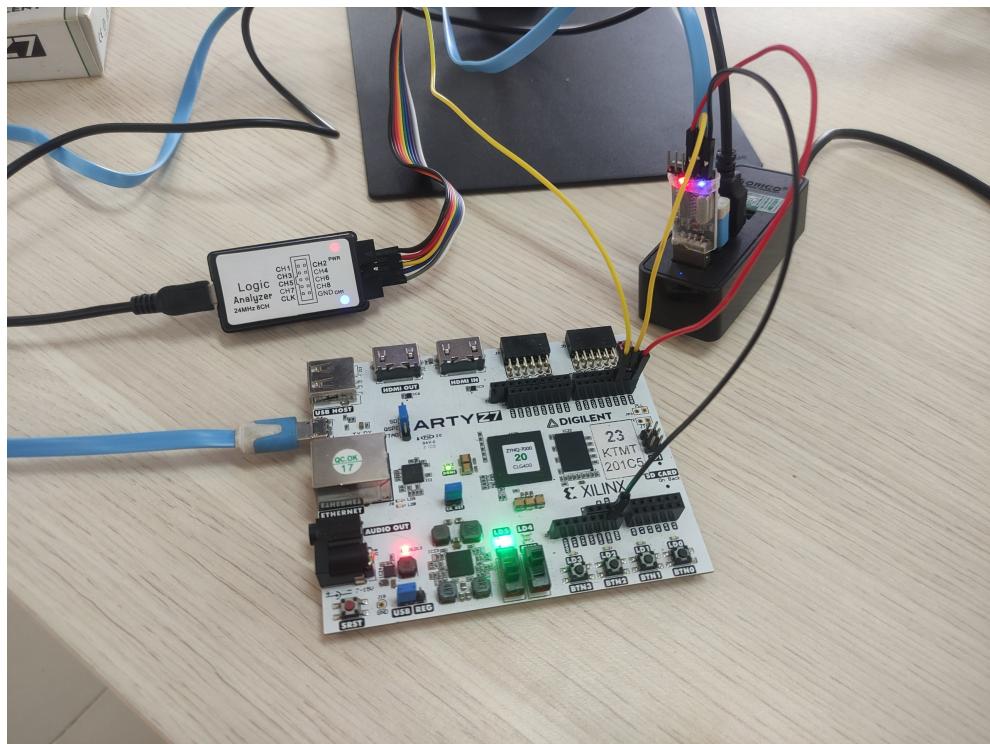
Xung nhịp hiển thị (`pixel_clk`): 25 MHz và 50 MHz.

## 8.2 Kết quả hiện thực các khối giao tiếp ngoại vi

Hệ thống đã hoàn thiện việc tích hợp và kiểm tra độ tin cậy của các chuẩn giao tiếp nối tiếp quan trọng, đảm bảo khả năng tương tác toàn diện với các thiết bị ngoại vi.

### 8.2.1 Giao tiếp UART

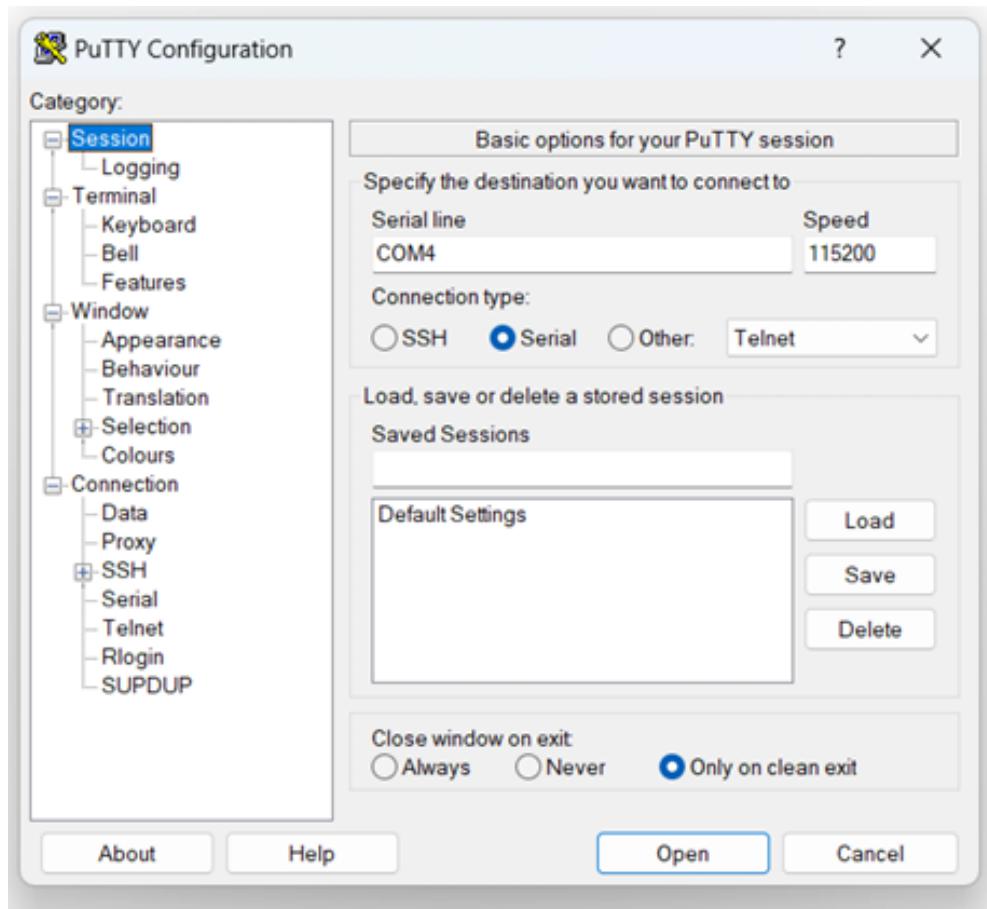
Khối UART đã được thực nghiệm thành công với các tốc độ baud chuẩn như: 9600, 115200, 230400, 460800 bps. Kết quả kiểm tra thông qua phần mềm Terminal (PuTTY) cho thấy dữ liệu được truyền nhận chính xác giữa SoC và máy tính cá nhân, phục vụ tốt cho việc xuất log gỡ lỗi và tương tác lệnh.



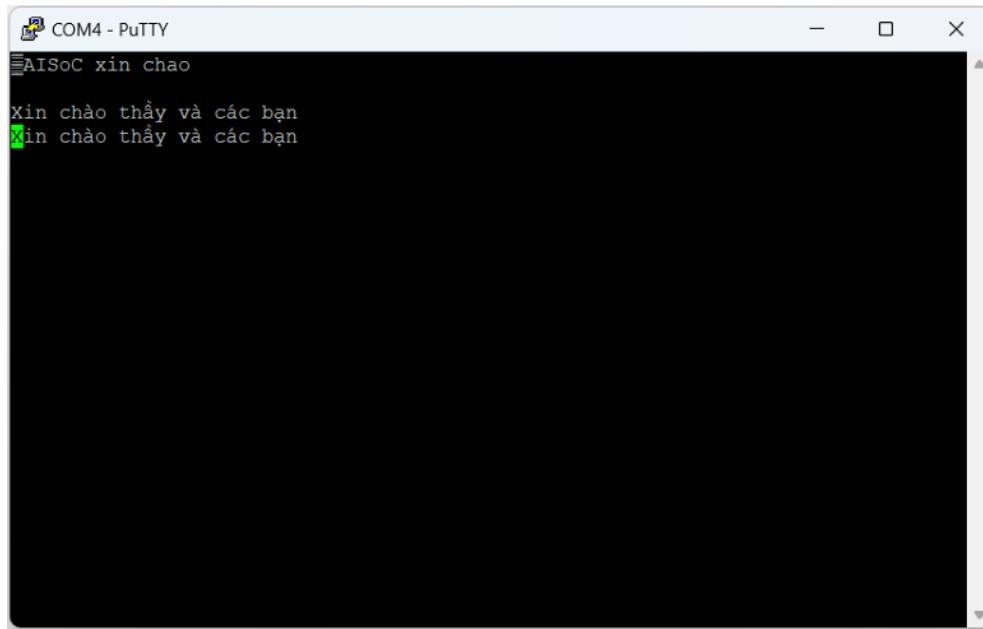
**Hình 8.1:** Kết nối UART từ Arty A7 với máy tính

```
main.cpp > ...
1 #include <iostream>
2 #include "uart.hpp"
3
4 int main() {
5     Uart uart; // Khởi tạo UART với địa chỉ mặc định
6
7     // Cấu hình UART với tần số clock 125 MHz và baud rate 115200
8     uart.init(125000000, 115200);
9
10    // Gửi "AISoC xin chào" tới terminal Laptop
11    uart.write('A'); uart.write('I'); uart.write('S'); uart.write('o'); uart.write('C'); uart.write(' ');
12    uart.write('x'); uart.write('i'); uart.write('n'); uart.write(' ');
13    uart.write('c'); uart.write('h'); uart.write('a'); uart.write('o'); uart.write('\n');
14
15    // Nhận dữ liệu từ Laptop và gửi lại terminal của Laptop
16    while (1) {
17        int16_t data = uart.read();
18        if (data != -1) {
19            uart.write(data); // Echo dữ liệu nhận được
20        }
21    }
22    return 0;
23
24
25 }
```

Hình 8.2: Chương trình kiểm thử giao tiếp UART



Hình 8.3: Cài đặt thông số PuTTY để giao tiếp UART

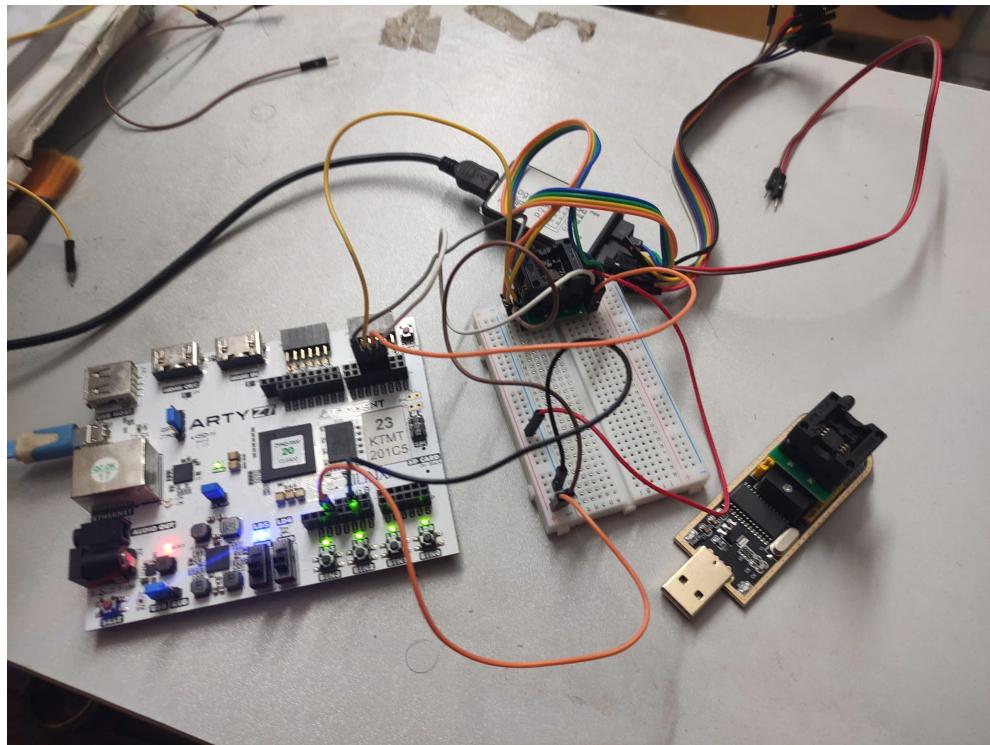


**Hình 8.4:** Kết quả giao tiếp UART thành công

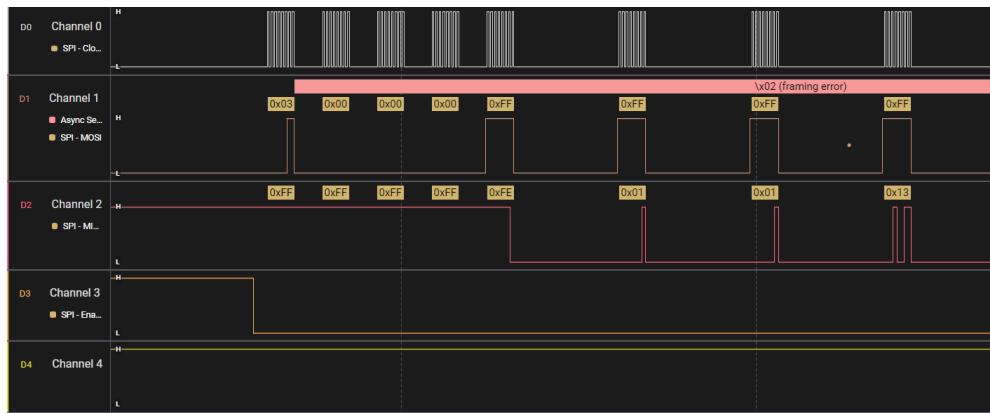
### 8.2.2 Giao tiếp I2C và SPI

**I2C:** Được sử dụng để cấu hình các thanh ghi nội của cảm biến hình ảnh OV5640 và chip phát HDMI ADV7513. Thực nghiệm cho thấy quá trình cấu hình diễn ra ổn định, các thiết bị phản hồi đúng địa chỉ và mã lệnh.

**SPI:** Đã hiện thực hóa bộ điều khiển SPI Master để giao tiếp với bộ nhớ Flash. Thực nghiệm với chip Flash W25Q32 cho kết quả đọc/ghi dữ liệu chính xác ở tốc độ lên đến 25 MHz.



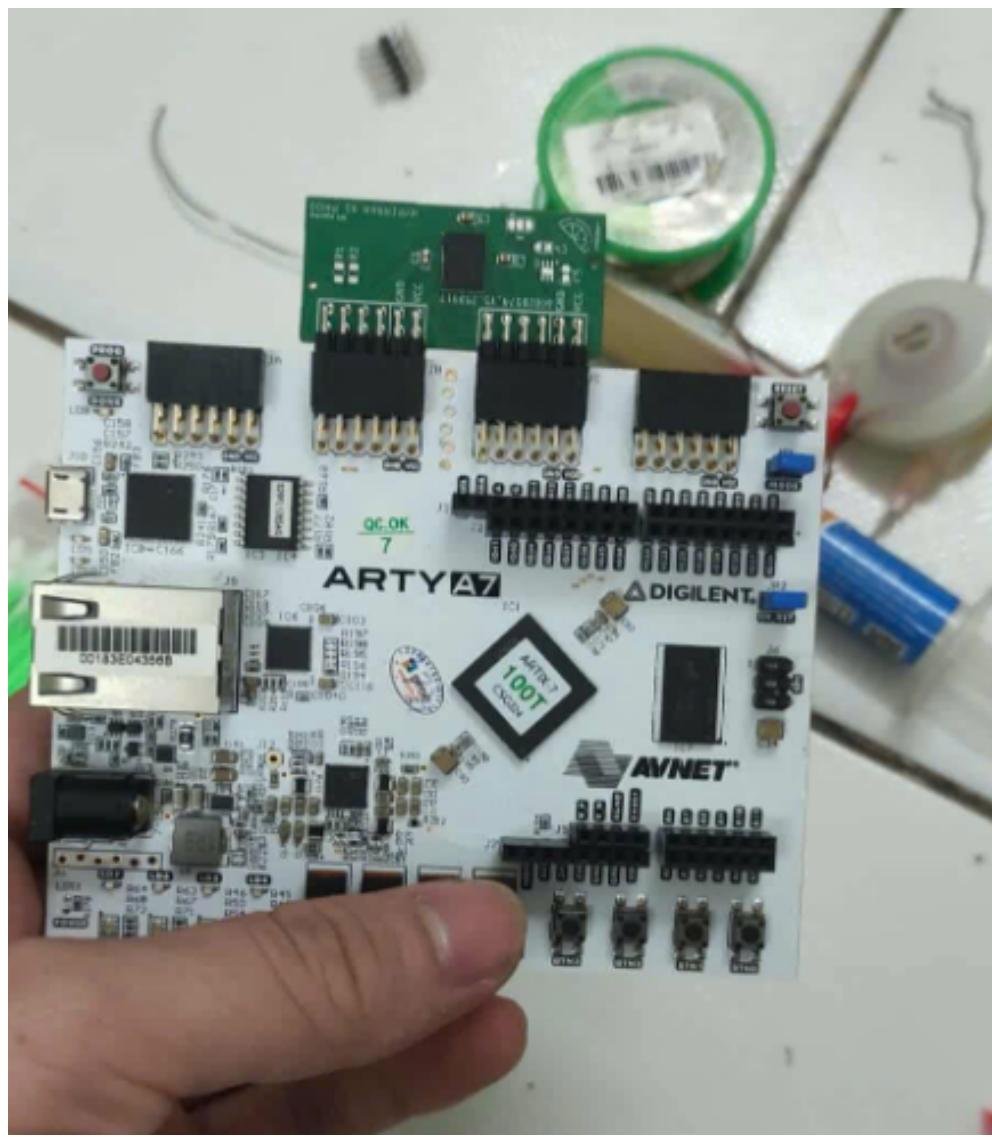
**Hình 8.5:** Kết nối SPI từ Arty A7 với bộ nhớ Flash



**Hình 8.6:** Sơ đồ sóng được đo thực tế bằng Logic Analyzer giữa SPI từ Arty A7 với bộ nhớ Flash

### 8.2.3 Giao tiếp Octal-SPI (OSPI) hỗ trợ DDR

Khối OSPI đã được thực nghiệm trực tiếp với chip nhớ **W958D8NBYA** HyperRAM với dung lượng **256 Mb**. Tốc độ đạt được là 50MHz trong chế độ DDR, kiểm tra bằng cách ghi dữ liệu vào chip và đọc lại rồi gửi qua Terminal UART để xác nhận tính chính xác của dữ liệu.



**Hình 8.7:** Kết nối chip nhớ HyperRAM (màu xanh lá) với Arty A7 qua giao tiếp OSPI

The screenshot shows a terminal window with OSPI code and a Hercules SETUP utility interface.

```

5 int main() {
27     ospi.set_config(capture_shmoo, recovery, latency, burst_len);
28
29     // Step 3: Write sample data to FIFO (8 bytes pattern)
30     const uint8_t write_data0[8] = {0xAA, 0x55, 0xAA, 0x55, 0xCC, 0x33, 0xCC, 0x33};
31     const uint8_t write_data1[8] = {0xBB, 0x55, 0xAA, 0x55, 0xCC, 0x33, 0xCC, 0x44};
32     const uint8_t write_data2[8] = {0xCC, 0x55, 0xAA, 0x55, 0xCC, 0x33, 0xCC, 0x55};
33     const uint8_t write_data3[8] = {0xDD, 0x55, 0xAA, 0x55, 0xCC, 0x33, 0xCC, 0x66};
34
35     // const uint8_t write_data1[8] = {0xAA, 0x55, 0xAA, 0x55, 0xCC, 0x33, 0xCC, 0x33}
36     // };
37     ospi.burst_write(write_data0, 8);
38     ospi.burst_write(write_data1, 8);
39     ospi.burst_write(write_data2, 8);
40     ospi.burst_write(write_data3, 8);
41
42
43 Hercules SETUP utility by HW-group.com
44 UDP Setup | Serial | TCP Client | TCP Server | UDP | Test Mode | About |
45 Received/Sent data
46 (AA){55}(AA){55}(CC){33}(CC){33}(BB){55}(AA){55}(CC){33}
47 (CC){44}(CC){55}(AA){55}(CC){33}(CC){55}(DD){55}(AA){55}
48 (CC){33}(CC){66}
49
50
51
52
PROBLEMS 96
TERMINAL
sora@Ra
sora@Ra
sora@Ra
sora@Ra
-bash:
sora@Ra
sora@Ra

```

The Hercules SETUP utility interface includes tabs for UDP Setup, Serial, TCP Client, TCP Server, UDP, Test Mode, and About. The Serial tab is selected, showing settings for Name (COM14), Baud (115200), Data size (8), Parity (none), Handshake (OFF), and Mode (Free). Modem lines status is shown as CD, RI, DSR, CTS, DTR, and RTS.

**Hình 8.8:** Chương trình cho việc ghi và đọc dữ liệu qua giao tiếp OSPI và xuất ra Terminal UART

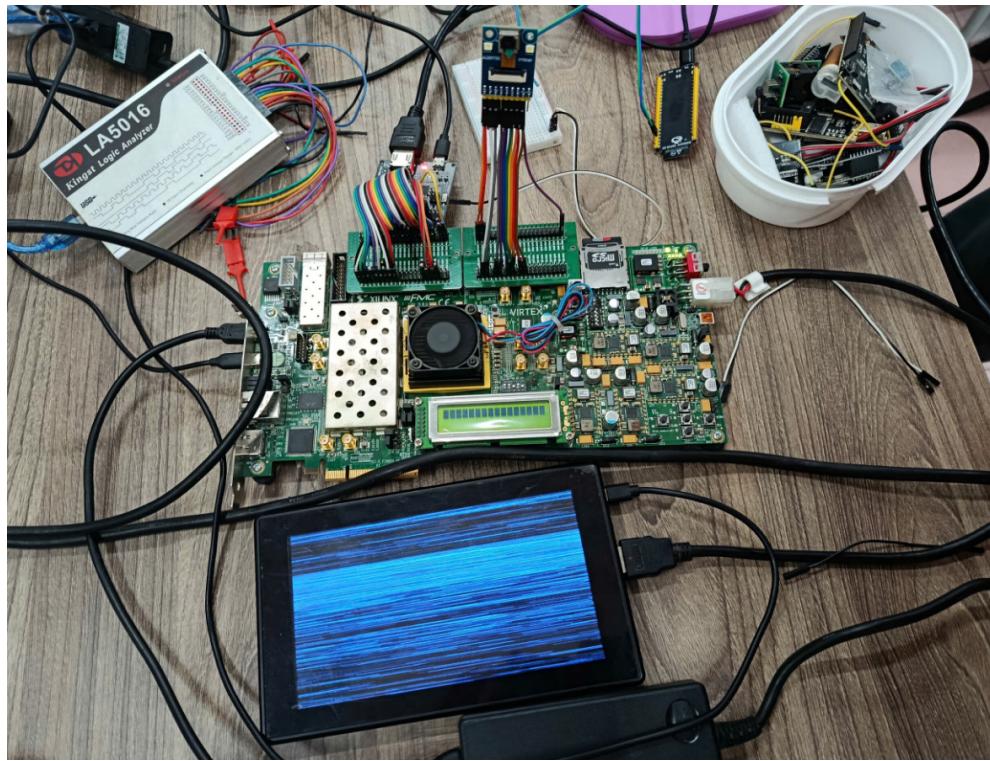
### 8.3 Kết quả Video Streaming 60Hz

Một trong những kết quả trọng tâm của đề tài là việc hiện thực hóa luồng dữ liệu Video thời gian thực từ camera ra màn hình.

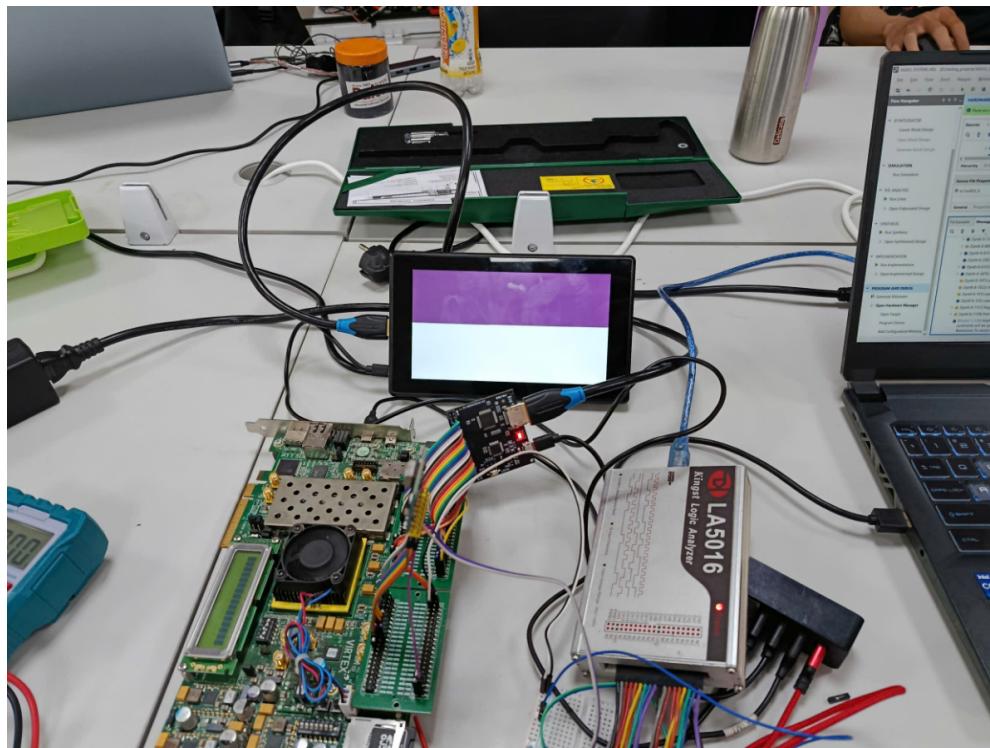
**Tốc độ khung hình:** Hệ thống đạt mức hiển thị ổn định **60 Hz**.

**Độ trễ:** Nhờ vào cơ chế quản lý bộ đệm khung hình (Frame Buffer) bằng BRAM, hiện tượng xé hình và trễ tích lũy đã được triệt tiêu đáng kể.

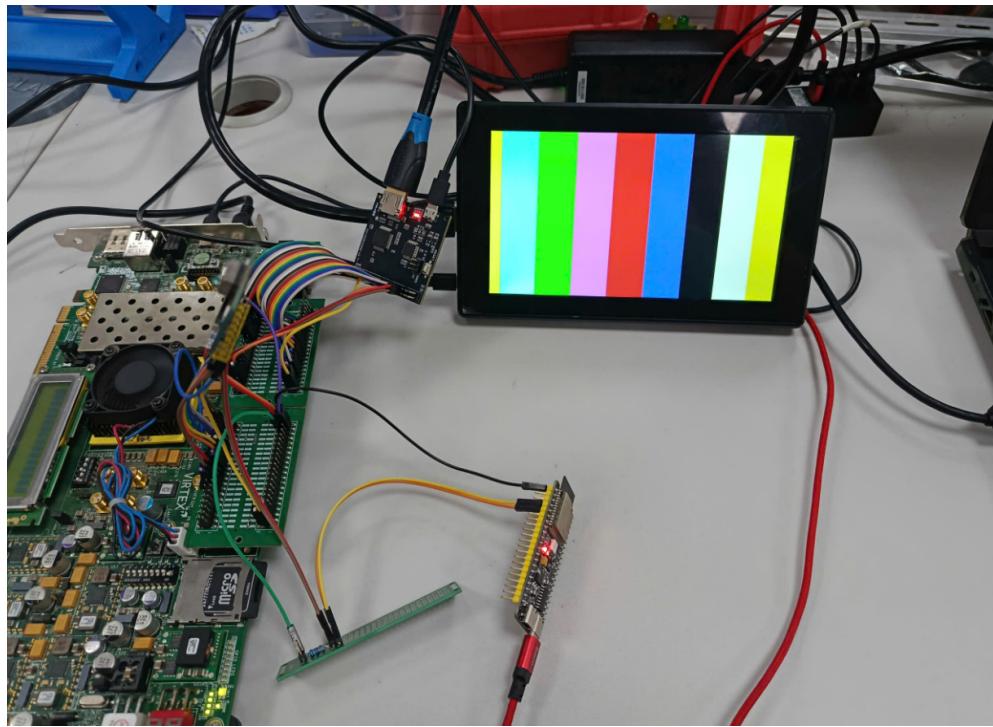
**Chất lượng hình ảnh:** Tín hiệu xuất ra qua cổng HDMI rõ nét, không có điểm ảnh lỗi, nhưng vẫn còn hai lỗi cần khắc phục, lỗi hình ảnh chạy sang bên phải và lỗi khung hình chưa được đồng bộ với màn hình.



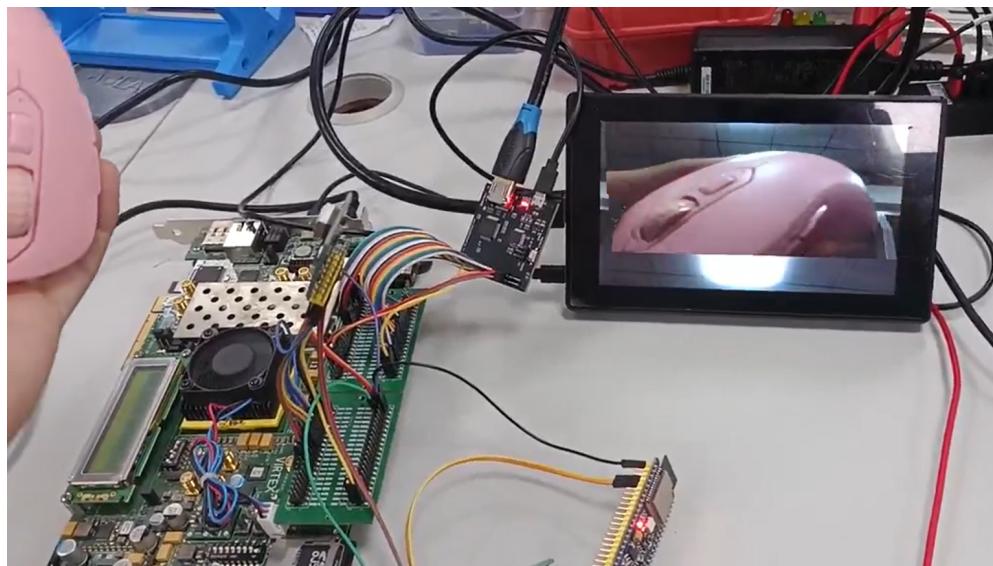
**Hình 8.9:** Phần cứng Video Streaming trên VC707 hiển thị hình ảnh từ camera OV5640 ra màn hình qua cổng HDMI



**Hình 8.10:** Kiểm tra Màn hình HDMI ổn định với tần số 60Hz



**Hình 8.11:** Kiểm tra Patten Bar từ camera OV5640 hiển thị qua HDMI



**Hình 8.12:** Chế độ Streaming Video thời gian thực từ camera OV5640 qua HDMI

## 8.4 Hiện thực chương trình Bootloader qua SPI Flash

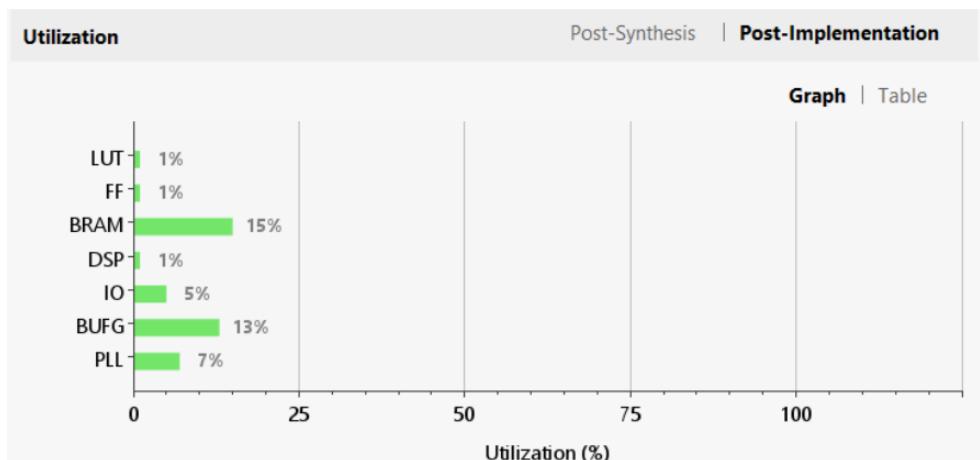
Để tăng tính độc lập cho SoC, chương trình **Bootloader** đã được thiết kế để nạp từ bộ nhớ SPI Flash ngoại vi.

**Quy trình:** Khi hệ thống khởi động (Power-on Reset), lõi PicoRV32 sẽ thực thi mã lệnh từ vùng nhớ ROM khởi tạo, thực hiện đọc dữ liệu thực thi từ SPI Flash và nạp vào bộ nhớ lệnh (IMEM).

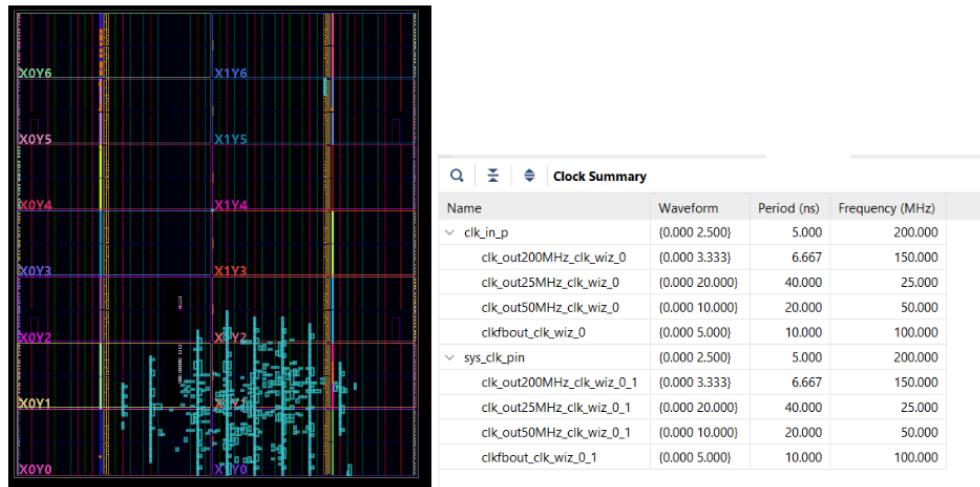
**Kết quả:** Hệ thống có khả năng tự khởi động và chạy các ứng dụng phần mềm mà không cần tổng hợp lại trên FPGA.

## 8.5 Đánh giá tài nguyên sử dụng trên FPGA

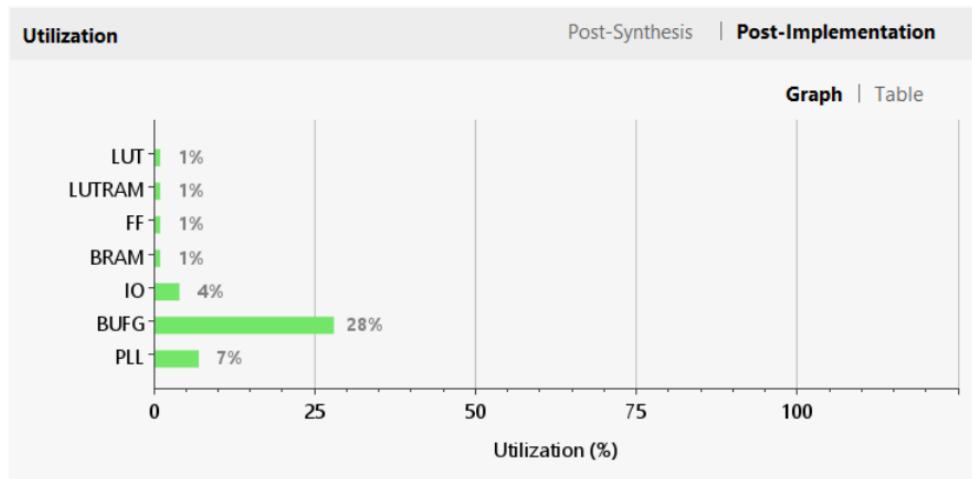
Dựa trên báo cáo tổng hợp từ công cụ Vivado cho thiết kế SoC (chưa có bộ tăng tốc) trên board VC707, tài nguyên hệ thống được sử dụng rất thấp, cho phép mở rộng thêm các bộ gia tốc AI phức tạp trong tương lai:



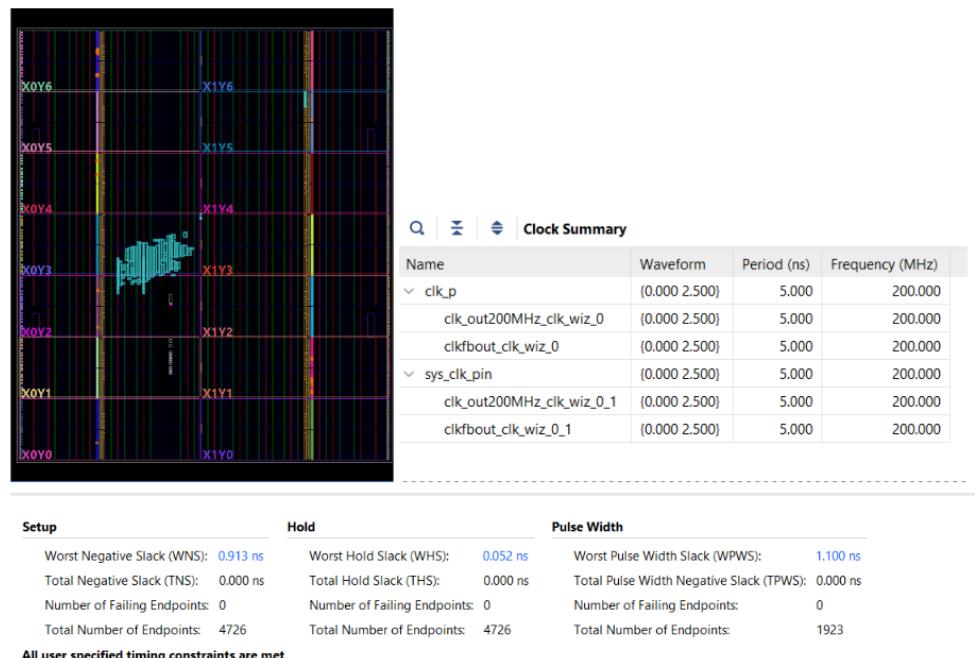
**Hình 8.13: a.** Mức độ sử dụng tài nguyên FPGA riêng phần Video Streaming trên VC707



**Hình 8.14: b.** Mức độ sử dụng tài nguyên FPGA riêng phần Video Streaming trên VC707



**Hình 8.15: c.** Mức độ sử dụng tài nguyên FPGA riêng phần SoC trên VC707



**Hình 8.16: d.** Mức độ sử dụng tài nguyên FPGA riêng phần SoC trên VC707

Việc chỉ sử dụng 15% BRAM trong khi vẫn duy trì được luồng Video 60Hz và đầy đủ các ngoại vi cho thấy hiệu quả cao của kiến trúc quản lý bộ đệm và hệ thống Bus AXI4-Lite đã thiết kế.

# Chương 9

# Kết luận và Hướng phát triển

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

## 9.1 Đánh giá Giai đoạn 1

Qua giai đoạn 1, đề tài đã cơ bản hoàn thiện phần cứng cốt lõi cho hệ thống SoC RISC-V. Thành phần trung tâm của hệ thống là lõi vi xử lý PicoRV32 đã vận hành ổn định trên nền tảng FPGA Virtex-7 VC707 tại tần số 200 MHz, được kết nối thông qua hệ thống Bus AXI4-Lite Interconnect linh hoạt hỗ trợ cấu hình đa Master - đa Slave. Bên cạnh đó, các khối ngoại vi giao tiếp thiết yếu đã được tích hợp và kiểm thử thành công, bao gồm UART, I2C Master phục vụ cấu hình cảm biến, và SPI Master. Đặc biệt, việc hiện thực hóa khối Octal-SPI (OSPI) hỗ trợ chế độ Double Data Rate (DDR) giao tiếp trực tiếp với chip nhớ W958D8NBYA ở tần số 50 MHz đã khẳng định khả năng đáp ứng băng thông lớn cho các tác vụ xử lý dữ liệu.

Về khả năng vận hành độc lập, đề tài đã xây dựng hoàn tất chương trình Bootloader nạp từ SPI Flash, cho phép hệ thống tự khởi động mã lệnh thực thi mà không cần sự tổng hợp lại trên FPGA. Đối với hệ thống Video Streaming, hệ thống đã bước đầu thu nhận được dữ liệu từ cảm biến camera OV5640 và hiển thị lên màn hình qua cổng HDMI với tốc độ khung hình 60 Hz. Tuy nhiên, qua thực nghiệm cho thấy luồng dữ liệu vẫn còn tồn tại hiện tượng lỗi đồng bộ (*synchronization issues*), gây ảnh hưởng đến độ ổn định của hình ảnh trong một số điều kiện vận hành. Song song đó, khối gia tốc AI đã được định hình kiến trúc và thiết kế ổn định về mặt cấu trúc logic, nhưng vẫn cần thực hiện các tinh chỉnh bổ sung để tối ưu hóa hiệu suất tính toán và làm rõ ràng hơn các luồng dữ liệu nội bộ. Với mức độ sử dụng tài nguyên FPGA hiện tại chỉ đạt 15% BRAM và 1% LUT/FF, hệ thống vẫn còn dư địa rất lớn để tích hợp các tính năng phức tạp hơn trong tương lai.

## 9.2 Kế hoạch thực hiện Giai đoạn 2

Dựa trên những kết quả đã đạt được và các hạn chế còn tồn tại, kế hoạch cho Giai đoạn 2 tập trung vào việc hoàn thiện hiệu năng và tích hợp trí tuệ nhân tạo vào SoC. Trọng tâm hàng đầu là hiện thực hóa khối điều khiển truy cập bộ nhớ trực tiếp (DMA) chuyên dụng để quản lý việc truyền tải dữ liệu tốc độ cao giữa bộ đệm khung hình và khối gia tốc, qua đó giảm thiểu tải xử lý cho CPU và tối ưu hóa băng thông Bus. Đồng thời, hệ thống Video Streaming sẽ được tinh chỉnh kỹ lưỡng về mặt đồng bộ hóa miền xung nhịp (*Clock Domain Crossing*) để khắc phục triệt để các lỗi hiển thị hiện tại, đảm bảo dòng video mượt mà phục vụ cho các thuật toán nhận diện.

Sau khi đường truyền dẫn dữ liệu hình ảnh ổn định, đề tài sẽ tiến hành hoàn thiện và nhúng khối gia tốc AI vào hệ thống SoC. Tập trung vào việc tối ưu hóa tài nguyên tính toán. Mục tiêu cuối cùng là thực hiện quá trình

tối ưu hóa thiết kế hướng tới ASIC, đồng thời cho ra đời một sản phẩm SoC tích hợp AI hoàn chỉnh và có khả năng demo thực tế các ứng dụng Edge AI trên nền tảng FPGA một cách thuyết phục.

### 9.3 Tiết độ dự kiến

Lộ trình thực hiện Giai đoạn 2 dự kiến kéo dài trong 12 tuần với các cột mốc quan trọng được quy hoạch như sau:

Bảng 9.1: Bảng tiến độ thực hiện Giai đoạn 2

Tuần	Nội dung công việc	Ghi chú
1 - 3	Thiết kế, kiểm thử khối DMA và khắc phục lỗi đồng bộ Video Stream.	Trọng tâm
1 - 6	Tinh chỉnh cấu trúc, tối ưu hóa và kiểm thử độc lập khối gia tốc AI.	Kỹ thuật
7 - 8	Tích hợp toàn hệ thống (SoC + AI Accel + DMA + Video).	Phức hợp
9 - 10	Phát triển hệ sinh thái phần mềm, driver điều khiển và ứng dụng mẫu.	Phần mềm
11 - 12	Tối ưu hóa định thời hệ thống, đóng gói sản phẩm và chuẩn bị báo cáo.	Hoàn tất

Việc bám sát lộ trình này sẽ đảm bảo hệ thống đạt được sự cân bằng tối ưu giữa hiệu năng phần cứng và tính linh hoạt của phần mềm, hướng tới một giải pháp SoC RISC-V tích hợp Edge AI mạnh mẽ và khả thi trong các ứng dụng IoT thực tiễn.

# Bibliography

- [1] Chen, Y.-H., Krishna, T., Emer, J. S., & Sze, V. (2017). Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits*, 52(1), 127–138.
- [2] Li, Z., Zhang, Z., Hu, J., Meng, Q., Shi, X., Luo, J., Wang, H., Huang, Q., & Chang, S. (2025). A High-Performance Pixel-Level Fully Pipelined Hardware Accelerator for Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 36(5), 7970–7981.
- [3] Zhang, J., & Li, J. (2017). Improving the Performance of OpenCL-based FPGA Accelerator for Convolutional Neural Network. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '17)*, 25–34.
- [4] Liu, Z., Jiang, J., Lei, G., Chen, K., Qin, B., & Zhao, X. (2020). A Heterogeneous Processor Design for CNN-Based AI Applications on IoT Devices. *Procedia Computer Science*, 174, 2–8.
- [5] Du, L., Du, Y., Li, Y., & Chang, M.-C. F. (2017). A Reconfigurable Streaming Deep Convolutional Neural Network Accelerator for Internet of Things. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(1), 198–208.