

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH**



ĐỒ ÁN TỐT NGHIỆP

**NHẬN DẠNG BIỂN SỐ XE MÁY VÀ ÔTÔ TRÊN NỀN TẢNG FPGA
THEO THỜI GIAN THỰC DỰA TRÊN TRÍ TUỆ NHÂN TẠO**

**NGÀNH: KỸ THUẬT MÁY TÍNH
HỘI ĐỒNG: 06 KTMT**

Người hướng dẫn:

PGS. TS. Phạm Quốc Cường - Trường ĐHBK

Người Phản biện:

ThS. Huỳnh Phúc Nghị - Trường ĐHBK

Sinh viên thực hiện:

Trần Tuấn Anh - 2110759

Nguyễn Hữu Chiến - 2110855

Lê Ngọc Thảo - 2114758

THÀNH PHỐ HỒ CHÍ MINH - THÁNG 5 NĂM 2025

KHOA: KH & KT MÁY TÍNH
BỘ MÔN: KỸ THUẬT MÁY TÍNH

HỌ VÀ TÊN: Nguyễn Hữu Chiến
HỌ VÀ TÊN: Trần Tuấn Anh
HỌ VÀ TÊN: Lê Ngọc Thảo
NGÀNH: Kỹ thuật Máy tính

NHIỆM VỤ LUẬN VĂN/ ĐỒ ÁN TỐT NGHIỆP
Chú ý: Sinh viên phải dán tờ này vào trang nhất của bản thuyết trình

MSSV: 2110855
MSSV: 2110759
MSSV: 2114758
LỚP: MT21KTM

1. Đầu đề luận văn/ đồ án tốt nghiệp:

Nhận dạng biển số xe máy và ôtô trên nền tảng FPGA theo thời gian thực dựa trên trí tuệ nhân tạo
(AI-based Real-time vehicle license plates recognition with FPGA platforms)

2. Nhiệm vụ (yêu cầu về nội dung và số liệu ban đầu):

- Nghiên cứu mô hình nhận dạng biển số xe cơ giới kết hợp giữa FPGA và bộ xử lý truyền thông
- Nghiên cứu phương pháp nhận dạng biển số khả hiện thực trên FPGA
- Thiết kế và hiện thực hệ thống đạt được tốc độ xử lý thời gian thực (24 hình/giây) với độ chính xác biển số nhận dạng không nhầm hơn 95% trong điều kiện ánh sáng tốt
- Tìm hiểu ứng dụng và tập dữ liệu đánh giá hệ thống

3. Ngày giao nhiệm vụ: 06/01/2025

4. Ngày hoàn thành nhiệm vụ: 22/5/2025

5. Họ tên giảng viên hướng dẫn:

1) Phạm Quốc Cường

Phản hướng dẫn:

CHỦ NHIỆM BỘ MÔN
(Ký và ghi rõ họ tên)

Phạm Quốc Cường

Ngày 06 tháng 01 năm 2025
GIẢNG VIÊN HƯỚNG DẪN CHÍNH
(Ký và ghi rõ họ tên)

Phạm Quốc Cường

PHẦN DÀNH CHO KHOA, BỘ MÔN:

Người duyệt (chấm sơ bộ):

Đơn vị:

Ngày bảo vệ:

Điểm tổng kết:

Nơi lưu trữ LVTN/DATN:

Ngày 15 tháng 5 năm 2025

PHIẾU ĐÁNH GIÁ LUẬN VĂN/ ĐỒ ÁN TỐT NGHIỆP

(Dành cho người hướng dẫn)

1. HỌ VÀ TÊN: Nguyễn Hữu Chiến
Trần Tuấn Anh
Lê Ngọc Thảo
MSSV: 2110855
MSSV: 2110759
MSSV: 2114758
NGÀNH: Kỹ thuật Máy tính
LỚP: MT21KTM

2. Đề tài:

Nhận dạng biển số xe máy và ôtô trên nền tảng FPGA theo thời gian thực dựa trên trí tuệ nhân tạo
(AI-based Real-time vehicle license plates recognition with FPGA platforms)

3. Họ tên người hướng dẫn: Phạm Quốc Cường

4. Tổng quát về bản thuyết minh:

Số trang:

Số trang:

Số tài liệu tham khảo

Hiện vật (sản phẩm)

5. Những ưu điểm chính của LV/ ĐATN:

- Sinh viên hoàn thành tốt nhiệm vụ đặt ra
 - Hệ thống hoạt động ổn định, chính xác, đáp ứng được tiêu chí đề ra
 - Sinh viên làm việc nghiêm túc, báo cáo đạt yêu cầu

6. Những thiếu sót chính của JV/DATN:

- Phần tăng tốc còn đơn giản (chi giai đoạn detection) nên dẫn đến hiệu suất chưa thật sự ấn tượng
 - Phần ứng dụng và tập dữ liệu test còn đơn giản

7. Đề nghị: Được bảo vệ Bổ sung thêm để bảo vệ Không được bảo vệ

8. Các câu hỏi SV phải trả lời trước Hội đồng:

a. Đâu là rào cản cho việc tăng tốc giai đoạn recognition trên FPGA?

9. Đánh giá chung (bằng chữ: Xuất sắc, Giới, Khá, TB):

Điểm: 9.0/10

Ký tên (ghi rõ họ tên)

Emm

Phạm Quốc Cường

Ngày tháng năm

PHIẾU ĐÁNH GIÁ LUẬN VĂN/ ĐỒ ÁN TỐT NGHIỆP

(Dành cho người hướng dẫn/phản biện)

1. Họ và tên SV: Trần Tuấn Anh MSSV: 2110759
Nguyễn Hữu Chiến MSSV: 2110855
Lê Ngọc Thảo MSSV: 2114758
Ngành (chuyên ngành): Kỹ thuật máy tính
2. Đề tài: NHẬN DẠNG BIÊN SÓ XE MÁY VÀ ÔTÔ TRÊN NỀN TẢNG FPGA THEO THỜI GIAN THỰC DỰA TRÊN TRÍ TUỆ NHÂN TẠO
3. Họ tên người hướng dẫn/phản biện: Huỳnh Phúc Nghị

4. Tổng quát về bản thuyết minh:

Số trang: 95	Số chương: 7
Số bảng số liệu: 6	Số hình vẽ: 24
Số tài liệu tham khảo: 30	Phần mềm tính toán:
Hiện vật (sản phẩm)	

5. Những ưu điểm chính của LV/ ĐATN:

- Thiết kế, hiện thực và đánh giá một hệ thống tăng tốc phần cứng cho bài toán nhận diện biển số xe theo thời gian thực trên nền tảng SoC.
- Xây dựng thành công một kiến trúc pipeline sâu trên PL, tối ưu hóa cho tính toán INT8. Kiến trúc bao gồm các khối Line Buffer, Processing Element (PE) với các chiến lược unroll Incha, và các FIFO trung gian để đảm bảo luồng dữ liệu hiệu quả. Đồng thời lưu trữ toàn bộ trọng số và feature map trung gian trên bộ nhớ BRAM của FPGA

6. Những thiếu sót chính của LV/ĐATN:

- Nhóm chi tăng tốc được khối backbone CNN của mô hình YOLO custom trên PL, ngoài ra phần tính toán khác trong mạng và hậu xử lý được thực hiện trên PS. Ngoài ra nhóm chưa phân tích cơ sở nghiên cứu của mô hình không sử dụng anchor.
- Kết quả đánh giá chi tiết về thời gian truyền dữ liệu giữa PS và PL, thời gian giao tiếp DMA và BRAM. Mà chi có kết quả chạy PS và PL, không có đánh giá thời gian chi tiết.
- Việc chi đánh giá backbone CNN để so sánh với CPU và GPU không phù hợp. Nhất là không có nghiên cứu liên quan để so sánh.

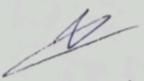
7. Đề nghị: Được bảo vệ Bổ sung thêm để bảo vệ Không được bảo vệ

8. Các câu hỏi SV phải trả lời trước Hội đồng:

- a. Hãy phân tích tương quan kích thước của BRAM và kích thước mô hình CNN để tính toán được khả năng tối đa có thể triển khai của mô hình CNN trên board hiện tại.

9. Đánh giá chung (bằng chữ: Xuất sắc, Giỏi, Khá, TB): Giỏi Điểm : 8.5 /10

Ký tên (ghi rõ họ tên)


 Huynh Phuc Ngan

LỜI CẢM ƠN

Chúng em xin chân thành gửi lời cảm ơn đến PGS.TS Phạm Quốc Cường và ThS Huỳnh Phúc Nghị vì sự hướng dẫn tận tình, sự cố vấn quý báu và sự hỗ trợ nhiệt tình trong suốt quá trình thực hiện đồ án ngành Kỹ thuật Máy tính. Kiến thức chuyên môn sâu rộng cùng sự động viên của các quý thầy đã đóng vai trò then chốt trong việc định hướng và nâng cao chất lượng công việc của chúng em.

LỜI CAM ĐOAN

Chúng em xin tuyên bố rằng đồ án tốt nghiệp là công trình do chính chúng em thực hiện, trừ những phần đã được trích dẫn hoặc ghi nhận nguồn rõ ràng. Nội dung của đồ án là kết quả của quá trình nghiên cứu và nỗ lực cá nhân của chúng em, và chúng em đã tuân thủ đầy đủ các quy định đạo đức và tiêu chuẩn học thuật.

Chúng em nhận thức rõ hậu quả của hành vi gian lận học thuật và hiểu rằng bất kỳ vi phạm nào đối với chuẩn mực đạo đức trong đồ án này đều có thể dẫn đến các hình thức kỷ luật theo quy định của Trường Đại học Bách Khoa – Đại học Quốc gia TP. Hồ Chí Minh.

TÓM TẮT

Luận văn này trình bày việc thiết kế và triển khai một hệ thống nhận dạng biển số xe tự động (ALPR) thời gian thực sử dụng mô hình học sâu YOLO (You Only Look Once), được tăng tốc trên nền tảng Xilinx Kria KV260 dựa trên Zynq UltraScale+ MPSoC. Hệ thống nhằm giải quyết các thách thức khi triển khai các mô hình học sâu yêu cầu tính toán cao trên các hệ thống nhúng có tài nguyên hạn chế, bằng cách khai thác khả năng xử lý song song của FPGA. Kiến trúc được đề xuất có mục tiêu chính là đạt được sự cân bằng giữa độ trễ thấp và độ chính xác cao trong việc phát hiện và nhận dạng biển số xe Việt Nam, bao gồm cả biển số một hàng và hai hàng, từ nhiều loại phương tiện khác nhau trong điều kiện đủ sáng, không bị che khuất và góc nghiêng tối đa 25 độ. Nghiên cứu này khảo sát các phiên bản khác nhau của mô hình YOLO và các kỹ thuật lượng tử hóa để xác định cấu hình tối ưu nhằm đạt tốc độ xử lý mục tiêu là 24 khung hình/giây (FPS). Hiệu năng của hệ thống được đánh giá dựa trên tốc độ, độ chính xác và mức độ sử dụng tài nguyên. Kết quả cho thấy tính khả thi của việc triển khai hệ thống ALPR thời gian thực được tăng tốc bằng FPGA cho nhiều ứng dụng khác nhau như quản lý bãi đỗ xe thông minh, giám sát giao thông, thu phí tự động và hỗ trợ thực thi pháp luật. Công trình này đóng góp cho lĩnh vực thị giác máy tính nhúng bằng cách cung cấp một mô hình triển khai các mô hình học sâu phức tạp trong các tình huống có giới hạn tài nguyên.

*Bài luận văn này là dành cho Cha Mẹ, Thầy Cô và tất cả những người
đồng hành cạnh bên trong suốt quãng đời sinh viên đáng nhớ.*

MỤC LỤC

Lời cảm ơn	v
Lời Cam Đoan	vi
Tóm tắt	vii
Danh sách hình vẽ	xiii
Danh sách bảng	xv
1 GIỚI THIỆU	1
1.1 Giới thiệu	1
1.2 Mục tiêu nghiên cứu	2
1.3 Phạm vi nghiên cứu	3
1.4 Đối tượng nghiên cứu	3
1.5 Bố cục luận văn	4
2 CƠ SỞ LÝ THUYẾT VÀ CÔNG TRÌNH LIÊN QUAN	7
2.1 Cơ Sở Lý Thuyết	7
2.1.1 FPGA và SoC.	7
2.1.2 Board Kria KV260.	10
2.1.3 PYNQ Framework	13
2.1.4 Giao thức AXI4 và IP AXI-DMA	15
2.1.5 Học Sâu và Mạng Nơ-ron Sâu	17
2.1.6 Mạng Nơ-ron Tích chập (CNN)	18
2.1.7 Mô hình You Only Look Once (YOLO)	20

2.2	Công Trình Liên Quan	22
2.2.1	Các Phương pháp Nhận dạng Biển số Xe Tự động (ALPR)	22
2.2.2	Phát hiện Biển số Xe	24
2.2.3	Nhận dạng Biển số Xe	25
3	KIẾN TRÚC PHẦN MỀM ĐỀ XUẤT	29
3.1	Quy trình nhận diện biển số đề xuất	29
3.2	Kiến trúc đề xuất cho Phát hiện Biển số Xe	33
3.3	Kiến trúc đề xuất cho Nhận Diện Kí Tự	38
3.4	Kiến trúc xử lí Convolution	41
4	KIẾN TRÚC PHẦN CỨNG ĐỀ XUẤT	47
4.1	Kiến trúc tổng quan	47
4.2	Kiến trúc chi tiết	49
4.2.1	Line Buffer (Bộ đếm dòng)	49
4.2.2	Processing Element (PE - Khối xử lý)	50
4.2.3	Khối FIFO (FIRST-IN, FIRST-OUT BUFFER)	51
4.2.4	Hệ thống bộ nhớ BRAM	53
4.2.5	Tối ưu hóa phép nhân trên khối DSP32E2	54
5	HIỆN THỰC HỆ THỐNG	57
5.1	Hiện thực lượng tử hóa	57
5.1.1	Quy trình lượng tử hóa INT8 PTQ	58
5.1.2	Trích xuất tham số INT8 cho phần cứng	59
5.1.3	Đánh giá độ chính xác sau lượng tử hóa INT8	59
5.2	Tích hợp hệ thống trong Vivado	60
5.3	Giao tiếp PS-PL qua AXI DMA với PYNQ	63
5.3.1	Giới thiệu AXI DMA và AXI Stream IP core	63
5.3.2	Khả năng hỗ trợ của PYNQ cho giao thức AXI DMA	65
5.3.3	Xác định và Khởi tạo các Kênh DMA	65
5.3.4	Chuẩn bị và Truyền dữ liệu	66

5.4	Register Bank và Điều khiển IP	69
5.5	Quản lý Bộ nhớ BRAM	69
5.6	Giao diện Khối IP và Controllers	70
6	KẾT QUẢ THỰC NGHIỆM	71
6.1	Dánh giá độ chính xác phần mềm	71
6.1.1	Tập dữ liệu đánh giá	71
6.1.2	Phương pháp đánh giá.	73
6.1.3	Kết quả đánh giá phần mềm	76
6.2	Môi trường thực nghiệm phần cứng	77
6.3	Kết quả Tổng hợp và Hiện thực trên PL	77
6.4	Dánh giá Hiệu năng Hệ thống.	79
6.4.1	Phương pháp đánh giá.	79
6.4.2	Đánh giá Tốc độ xử lý End-to-End.	80
6.5	So sánh với các nền tảng khác.	82
6.6	So sánh với các công trình liên quan.	83
7	KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN	87
7.1	Kết luận	87
7.2	Hạn chế	89
7.3	Hướng phát triển tương lai	90
	Tài liệu tham khảo	93



DANH SÁCH HÌNH VẼ

2.1	Kiến trúc FPGA Cơ bản [1].	9
2.2	Kiến trúc MPSoC Zynq [2].	10
2.3	The Kria KV260 Vision AI Starter Kit [3].	11
2.4	PYNQ Framework [4].	14
2.5	Các kênh AXI [5].	16
2.6	Mạng nơ-ron sâu [6].	18
2.7	Kiến trúc CNN áp dụng cho nhận dạng chữ số [7].	20
2.8	Phát hiện bằng YOLO[8].	22
2.9	Các giai đoạn chính trong hệ thống nhận dạng biển số xe đa giai đoạn [9].	23
2.10	Phân loại các kỹ thuật phát hiện biển số xe liên quan [9]. .	24
2.11	Quy trình nhận dạng biển số xe với các kỹ thuật liên quan [9].	26
3.1	Quy trình ALPR tổng thể	32
3.2	Kiến trúc tổng thể của mô hình phát hiện biển số xe.	34
4.1	Kiến trúc tổng quan và luồng dữ liệu của bộ tăng tốc YOLO trên PL.	48
4.2	Sơ đồ khối kiến trúc Line Buffer [10]	49
4.3	Sơ đồ khối kiến trúc PE Incha	51
4.4	Kiến trúc tổng quan khối DSP48E2.	54
4.5	Thực hiện hai phép nhân 8-bit ($a*c$, $b*c$) trên một khối DSP.	56
5.1	Sơ đồ khối tích hợp hệ thống tăng tốc YOLO trong Vivado.	61
5.2	Sơ đồ khối IP Model trong Vivado.	62

5.3	AXIS module.	64
5.4	AXI DMA và AXI Interconnect trong giao tiếp giữa PS và PL.	64
5.5	Giao tiếp giữa PYNQ và AXI DMA IP.	65
6.1	Tài nguyên phần cứng PL sử dụng bởi bộ tăng tốc YOLO trên Kria KV260.	78

DANH SÁCH BẢNG

2.1	Chi tiết Sản phẩm The Kria KV260 Vision AI Starter Kit	13
3.1	Chi tiết kiến trúc mạng CNN nhận diện ký tự	40
5.1	So sánh độ chính xác trước và sau lượng tử hóa INT8 PTQ.	60
6.1	Kết quả đánh giá độ chính xác phần mềm trên 2 tập dữ liệu.	76
6.2	So sánh thời gian xử lý và tốc độ trên Kria KV260.	81
6.3	So sánh hiệu năng giữa phần cứng FPGA và nền tảng khác.	83
6.4	So sánh hiệu năng và thông số với các công trình liên quan.	83



1

GIỚI THIỆU

Chương này giới thiệu tổng quan về luận văn. Nội dung bao gồm bối cảnh thực hiện đề tài, mục tiêu nghiên cứu, phạm vi nghiên cứu, đối tượng nghiên cứu và bối cảnh của luận văn.

1.1. GIỚI THIỆU

Trong một thế giới ngày càng kết nối, khả năng tự động và hiệu quả trong việc nhận dạng phương tiện thông qua biển số xe đã trở nên vô cùng quan trọng. Hệ thống nhận dạng biển số xe tự động (ALPR) không còn là một khái niệm viễn tưởng, mà đã trở thành một thành phần thiết yếu trong hạ tầng hiện đại, thúc đẩy sự phát triển trong các lĩnh vực như giao thông thông minh, an ninh và hỗ trợ thực thi pháp luật. Từ các giải pháp đỗ xe thông minh, tối ưu hóa lưu lượng giao thông cho đến thu phí tự động, ứng dụng của ALPR rất đa dạng và không ngừng mở rộng.

Tuy nhiên, để đạt được khả năng nhận dạng biển số chính xác và thời gian thực, đặc biệt trong môi trường tài nguyên hạn chế, cần vượt qua nhiều rào cản kỹ thuật đáng kể. Mặc dù các mô hình học sâu, đặc biệt

1

là họ mô hình YOLO (You Only Look Once), đã thể hiện khả năng vượt trội trong nhận dạng đối tượng, nhưng việc triển khai chúng trên các hệ thống nhúng vẫn là một thách thức lớn. Cường độ tính toán cao của các mô hình này thường xung đột với giới hạn về tốc độ xử lý và tài nguyên phần cứng sẵn có của các nền tảng nhúng.

Chính thách thức này là động lực cốt lõi thúc đẩy nghiên cứu: tìm cách thu hẹp khoảng cách giữa tiềm năng của hệ thống ALPR tiên tiến và các giới hạn thực tế khi triển khai. Luận văn này đề xuất một hướng tiếp cận mới nhằm hiện thực hóa một hệ thống ALPR hiệu năng cao nhưng vẫn đảm bảo hiệu quả, thông qua việc khai thác sức mạnh của mảng logic lập trình được (FPGA) trong kiến trúc hệ thống trên chip (SoC). Bằng cách chuyển các tác vụ tính toán nặng sang phần cứng FPGA, mục tiêu là đạt hiệu suất thời gian thực đồng thời duy trì độ chính xác cao. Cụ thể, nghiên cứu tập trung vào nền tảng Zynq UltraScale+ MPSoC, sử dụng bộ kit Kria KV260 Vision AI Starter và khung phần mềm PYNQ để xây dựng một giải pháp tối ưu.

1.2. MỤC TIÊU NGHIÊN CỨU

Luận văn hướng đến bốn mục tiêu chính:

- Thứ nhất, nghiên cứu về nền tảng lý thuyết và các yếu tố thực tiễn trong việc triển khai mô hình học sâu trên cá hệ thống máy tính tiên tiến (sử dụng python) và trên FPGA.
- Thứ hai, thiết kế, phát triển và kiểm thử một hệ thống ALPR hoàn chỉnh.
- Thứ ba, nghiên cứu và hiện thực một phần hệ thống sử dụng FPGA để tăng tốc.
- Thứ tư, nghiên cứu và áp dụng mô hình học máy CNN vào xử lý hình ảnh sử dụng FPGA.

- Cuối cùng, phân tích và tối ưu hiệu năng hệ thống, tận dụng lợi thế vốn có của FPGA như khả năng xử lý song song và tùy biến phần cứng, để đạt tốc độ xử lý mục tiêu 24 khung hình/giây, đồng thời so sánh hiệu quả với các phương pháp khác.

1.3. PHẠM VI NGHIÊN CỨU

Phạm vi nghiên cứu bao gồm việc thiết kế hệ thống nhận dạng biển số được triển khai một phần trên nền tảng Zynq UltraScale+ MPSoC, tối ưu hóa kiến trúc CNN cho phần logic lập trình (PL) trên FPGA Kria KV260 (tập trung vào cấu trúc layer, kích thước kernel, định lượng tham số để phù hợp với tài nguyên DSP, LUT và BRAM), xây dựng giao diện AXI DMA PS-PL để truyền dữ liệu ảnh và kết quả phát hiện giữa hai khối, huấn luyện mô hình trên môi trường GPU và lượng tử hóa trọng số để hiện thực trên FPGA, cũng như đánh giá hệ thống về độ chính xác phát hiện, độ trễ xử lý trên PL và mức sử dụng tài nguyên FPGA.

1.4. ĐỐI TƯỢNG NGHIÊN CỨU

Dựa trên phạm vi và mục tiêu nghiên cứu, luận văn tập trung vào các đối tượng chính sau:

- Xây dựng một hệ thống ALPR bằng python để nhận diện biển số xe, đánh giá hệ thống để có cái nhìn tổng quát về việc hiện thực trên các hệ thống máy tính.
- Nghiên cứu các mô hình học máy, trí tuệ nhân tạo như các mô hình CNN cơ bản, SSD, YOLO. Xây dựng một mô hình CNN và hiện thực nó trên FPGA để tận dụng các lợi thế của FPGA trong việc tăng tốc xử lý.
- Kỹ thuật lượng tử hóa mô hình và tối ưu hóa kiến trúc phần cứng.
- Truyền dữ liệu giữa hệ thống xử lý (PS) và logic lập trình được (PL).

1

1.5. BỘ CỤC LUẬN VĂN

Cấu trúc luận văn bao gồm các chương sau:

- **Chương 1 - Giới thiệu:** Trình bày động lực triển khai và tăng tốc hệ thống ALPR, đồng thời nêu rõ mục tiêu, phạm vi và đóng góp của đề tài. Cũng bao gồm mô tả ngắn gọn về nội dung các chương tiếp theo.
- **Chương 2 - Cơ sở lý thuyết và công trình liên quan:** Trình bày nền tảng lý thuyết cần thiết cho nghiên cứu, bao gồm FPGA, SoC, học sâu và các công nghệ cụ thể được sử dụng trong đề tài. Phân tích các công trình nghiên cứu hiện có liên quan đến ALPR, YOLO trên FPGA và kỹ thuật tối ưu hóa mô hình, từ đó xác định các chiến lược và đổi mới có thể áp dụng cho đề tài.
- **Chương 3 - Kiến trúc phần mềm đề xuất:** Trình bày chi tiết kiến các quy trình nhận diện biển số xe. Nêu ra kiến trúc cho 2 giai đoạn phát hiện và nhận diện kí tự. Đề xuất kiến trúc tăng tốc xử lí Convolution.
- **Chương 4 - Kiến trúc phần cứng đề xuất:** Mô tả chi tiết kiến trúc phần cứng của bộ tăng tốc YOLO INT8 được thiết kế cho vùng PL của Kria KV260. Bắt đầu bằng sơ đồ khái quát và luồng dữ liệu chính trong PL. Sau đó, đi sâu vào thiết kế của từng khối chức năng cốt lõi như Line Buffer, Processing Element, quản lý bộ nhớ BRAM (cho trọng số và feature maps), các khối FIFO trung gian, và giao diện AXI (Stream và Lite/DMA-controlled) để kết nối các thành phần.
- **Chương 5 - Hiện thực Hệ thống:** Trình bày quá trình hiện thực hóa kiến trúc phần cứng đã thiết kế. Mô tả việc tích hợp các khối IP (Zynq US+ PS, AXI DMA, AXI SmartConnect, IP YOLO tùy chỉnh) trong môi trường Vivado. Tập trung vào cơ chế giao tiếp PS-PL thông qua AXI DMA, bao gồm cách cấu hình DMA và tương tác phần mềm

sử dụng framework PYNQ để cấp phát bộ đệm, truyền dữ liệu (ảnh, trọng số, kết quả), và điều khiển/giám sát khối IP tăng tốc.

- **Chương 6 - Kết quả thực nghiệm và Đánh giá:** Trình bày môi trường thực nghiệm (phần cứng Kria KV260, phần mềm PYNQ, dataset), các chỉ số đánh giá (độ chính xác mAP, tốc độ FPS, độ trễ ms, tài nguyên sử dụng LUTs/FFs/BRAMs/DSPs, công suất tiêu thụ W). Phân tích và so sánh kết quả đo đặc thực tế của hệ thống tăng tốc phần cứng (kịch bản PL+PS) với baseline phần mềm (chạy hoàn toàn trên PS), đánh giá mức độ tăng tốc, hiệu quả sử dụng tài nguyên và năng lượng.
- **Chương 7 - Kết luận và Hướng phát triển:** Tóm tắt lại các kết quả chính đã đạt được trong khóa luận, đổi chiều với các mục tiêu đề ra ban đầu và khẳng định những đóng góp của đề tài. Nêu ra những hạn chế còn tồn tại của hệ thống hiện tại và đề xuất các hướng nghiên cứu, phát triển tiềm năng trong tương lai để cải thiện hiệu năng, độ chính xác hoặc mở rộng tính năng.



2

CƠ SỞ LÝ THUYẾT VÀ CÔNG TRÌNH LIÊN QUAN

Chương này trình bày các công nghệ và khái niệm cốt lõi làm nền tảng cho đề tài. Sau đó sẽ trình bày nghiên cứu về các công trình liên quan tới đề tài.

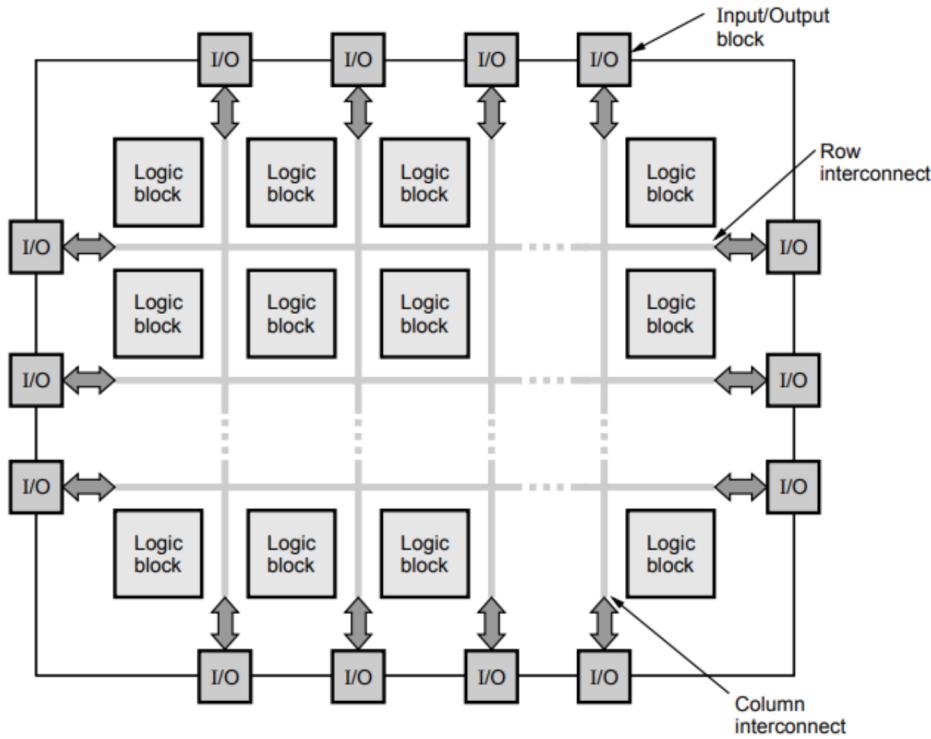
2.1. Cơ Sở Lý Thuyết

2.1.1. FPGA và SoC

Trong những năm gần đây, sự tăng trưởng nhanh chóng của nhu cầu tính toán, đặc biệt là trong các lĩnh vực như học sâu, trí tuệ nhân tạo và xử lý thời gian thực, đã thúc đẩy sự phát triển của công nghệ phần cứng. Mặc dù bộ vi xử lý và bộ vi điều khiển được sử dụng rộng rãi trong điện toán đa dụng, chúng thường gặp khó khăn trong việc đáp ứng các yêu cầu chuyên biệt của các ứng dụng tiên tiến này. Các tác vụ như chạy thuật toán phức tạp hoặc xử lý khối lượng dữ liệu lớn đòi hỏi sức mạnh xử lý mà các bộ xử lý truyền thống không thể cung cấp một cách hiệu quả. Để khắc phục hạn

2

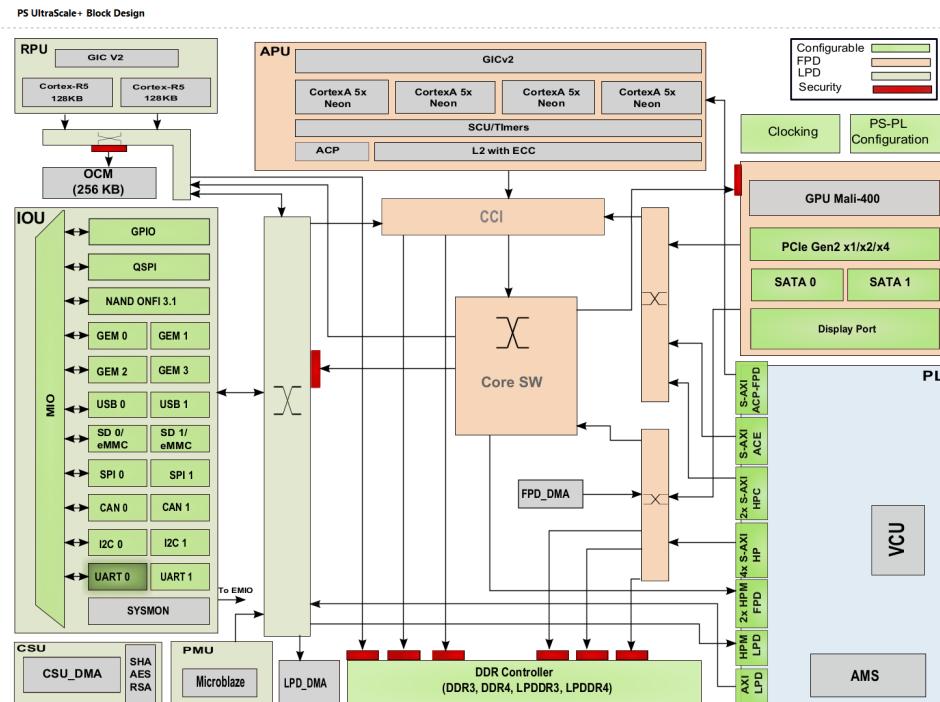
chế này, Mạch Logic Khả Lập Trình (FPGA - Field-Programmable Gate Array) đã nổi lên như một giải pháp linh hoạt. Không giống như các bộ xử lý thông thường, FPGA cung cấp một kiến trúc phần cứng có thể cấu hình lại, cho phép tùy chỉnh cho các tác vụ cụ thể. Điều này cho phép tính toán hiệu năng cao, tiêu thụ điện năng thấp, đặc biệt hữu ích trong các ứng dụng yêu cầu xử lý thời gian thực và tính toán song song, chẳng hạn như xử lý video, xử lý tín hiệu và học máy. Khả năng lập trình lại FPGA mang lại sự linh hoạt đáng kể, làm cho chúng phù hợp với nhiều ứng dụng, từ hệ thống nhúng đến điện toán hiệu năng cao. FPGA bao gồm Khối Logic Cấu hình được (CLB - Configurable Logic Block), các kết nối có thể lập trình và các tài nguyên bổ sung như các khối Xử lý Tín hiệu Số (DSP - Digital Signal Processing) và Bộ nhớ Khối RAM (BRAM - Block RAM). Các thành phần này được kết nối với nhau để tạo ra các cấu hình phần cứng tùy chỉnh phù hợp với nhu cầu cụ thể của một ứng dụng. Với sự hỗ trợ của các ngôn ngữ mô tả phần cứng (HDL - hardware description languages) như Verilog và VHDL, các nhà phát triển có thể thiết kế và triển khai các mạch logic tùy chỉnh, tối ưu hóa phần cứng về tốc độ, công suất và hiệu quả. Hơn nữa, các FPGA hiện đại hỗ trợ tái cấu hình động, cho phép sửa đổi các phần của thiết kế trong thời gian chạy mà không ảnh hưởng đến hoạt động tổng thể.



Hình 2.1: Kiến trúc FPGA Cơ bản [1].

Khi các hệ thống phần cứng ngày càng phức tạp, việc tích hợp các thành phần khác nhau, chẳng hạn như bộ nhớ, bộ vi xử lý và logic lập trình được, trong một hệ thống duy nhất trở nên cần thiết. Công nghệ Hệ thống trên Chip (SoC - System-on-Chip) giải quyết nhu cầu này bằng cách kết hợp các tài nguyên phần cứng này vào một chip duy nhất. Ví dụ, Hệ thống trên Chip Đa xử lý (MPSoC - Multi-Processor System on Chip) của Xilinx hợp nhất logic lập trình được với một bộ xử lý cố định trên một chip duy nhất, được kết nối bằng Giao diện Mở rộng Tiên tiến (AXI - Advanced Extensible Interfaces) để tạo điều kiện giao tiếp. Kiến trúc MPSoC được chia thành hai khu vực chính: Logic Lập trình được (PL - Programmable Logic) và Hệ thống Xử lý (PS - Processor System). Khu vực PL là một FPGA có khả năng xử lý các tác vụ song song, trong khi khu vực

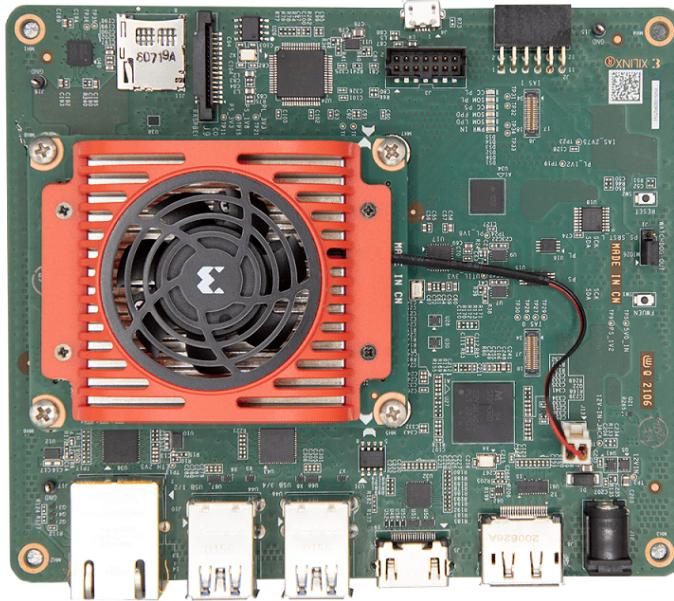
PS, thường dựa trên bộ xử lý ARM Cortex-A9, được tối ưu hóa để thực thi các tác vụ tuần tự, đa dụng. Quy trình làm việc đồng thiết kế này, tích hợp phát triển phần cứng và phần mềm, đã trở thành một phương pháp được áp dụng rộng rãi để thiết kế hệ thống hiệu quả.



Hình 2.2: Kiến trúc MPSoc Zynq [2].

2.1.2. BOARD KRIA KV260

Board Kria KV260 Vision AI Starter Kit [11] (Hình 2.3) là bo mạch phát triển tiên tiến được thiết kế đặc biệt để tăng tốc các ứng dụng thị giác dựa trên AI. Là một giải pháp linh hoạt và mạnh mẽ, nền tảng KV260 hợp lý hóa quy trình phát triển, cho phép các chuyên gia triển khai các hệ thống thị giác hiệu năng cao cho nhiều ứng dụng, bao gồm thành phố thông minh, tự động hóa công nghiệp, thị giác máy và giám sát an ninh.



Hình 2.3: The Kria KV260 Vision AI Starter Kit [3].

Các Tính năng Chính của KV260

Bộ kit bo mạch KV260 được trang bị một tập hợp các tính năng toàn diện khiến nó trở nên lý tưởng cho điện toán biên và tăng tốc thị giác:

- **K26 SOM (Hệ thống trên Mô-đun - System-on-Module):** Trái tim của nền tảng là Kria K26 SOM, được trang bị XCK26 Zynq UltraScale+ MPSoC, được thiết kế để tăng tốc các khối lượng công việc AI đòi hỏi nhiều tính toán.
- **Các Tùy chọn Kết nối:**
 - 8 giao diện hỗ trợ kết nối camera.
 - Các giao diện cảm biến MIPI CSI-2.
 - Bộ xử lý Tín hiệu Hình ảnh Tích hợp (ISP - Integrated Image Signal Processor).
 - Các đầu ra HDMI và DisplayPort cho video độ nét cao.

- Ethernet 1Gb cho kết nối mạng.
- Các giao diện USB 3.0/2.0 cho các thiết bị ngoại vi.
- Khe cắm thẻ nhớ MicroSD cho lưu trữ bổ sung và tùy chọn khởi động.

- **Bộ nhớ Hệ thống và Lưu trữ:**

- DDR4 64-bit 4GB để xử lý dữ liệu hiệu quả.
- Bộ nhớ flash eMMC 16GB cho hệ điều hành và ứng dụng.
- Mô-đun Nền tảng Đáng tin cậy (Trusted Platform Module) để tăng cường bảo mật.
- Bộ nhớ flash QSPI 512Mb cho firmware.

Ưu điểm của K26 SOM

K26 SOM nổi bật nhờ sự cân bằng giữa sức mạnh xử lý và hiệu quả năng lượng. Các lợi ích chính bao gồm:

- Tăng tốc AI Nâng cao: Được thiết kế đặc biệt để tăng tốc các ứng dụng AI tập trung vào thị giác.
- Tính linh hoạt: Cung cấp nhiều tùy chọn kết nối, cho phép tích hợp với các hệ thống và môi trường đa dạng.
- Sử dụng Tài nguyên Hiệu quả: Được xây dựng trên công nghệ FinFET 16nm tiên tiến, SoC XCK26 mang lại hiệu năng tối ưu trong khi giảm thiểu mức tiêu thụ điện năng.

Luận văn của chúng tôi tập trung vào lõi tăng tốc được phát triển trên phần logic lập trình được của XCK26. Bảng dưới đây cung cấp phân tích chi tiết về các tài nguyên phần cứng của K26 SOM, thể hiện khả năng của nó trong việc giải quyết nhu cầu của các ứng dụng thị giác AI hiện đại.

Thông số kỹ thuật	Mô tả
Cells logic hệ thống	256K
Khối Block RAM	144
Khối UltraRAM	64
DSP Slices	1.2K

Bảng 2.1: Chi tiết Sản phẩm The Kria KV260 Vision AI Starter Kit

2

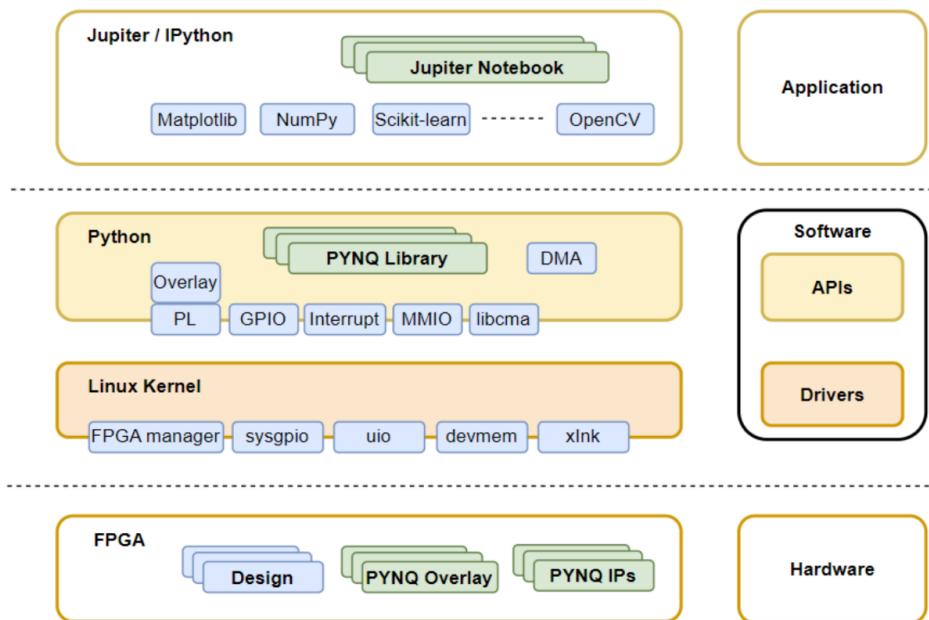
2.1.3. PYNQ FRAMEWORK

PYNQ Framework (Python Productivity for Zynq) là một nền tảng mã nguồn mở được phát triển bởi AMD-Xilinx nhằm đơn giản hóa việc thiết kế và triển khai các ứng dụng trên các hệ thống dựa trên Zynq. Bằng cách cho phép các nhà phát triển lập trình logic FPGA bằng Python, PYNQ thu hẹp khoảng cách giữa thiết kế phần mềm và phần cứng, giúp công nghệ FPGA trở nên dễ tiếp cận ngay cả với các nhà phát triển không có kiến thức sâu rộng về các ngôn ngữ mô tả phần cứng như Verilog hoặc VHDL.

Các Tính năng Chính của PYNQ

- Điều khiển dựa trên Python: PYNQ cho phép các nhà phát triển điều khiển các thiết bị ngoại vi phần cứng, quản lý luồng dữ liệu và thậm chí thực thi các chức năng FPGA trực tiếp từ các tập lệnh Python. Tính năng này giúp giảm đáng kể đường cong học tập cho việc lập trình FPGA.
- Các Overlay dựng sẵn: bao gồm các tệp bitstream FPGA được dựng sẵn, được gọi là overlay, thực hiện các chức năng phần cứng cụ thể. Các nhà phát triển có thể sử dụng các overlay này cho các ứng dụng như xử lý hình ảnh, học máy hoặc hệ thống truyền thông mà không cần phải thiết kế phần cứng từ đầu.
- Tích hợp Jupyter Notebook: PYNQ tích hợp với Jupyter Notebooks, cung cấp một môi trường phát triển tương tác nơi người dùng có thể viết và thực thi mã Python, trực quan hóa dữ liệu và tài liệu hóa quy trình làm việc một cách liền mạch.

- Thư viện Cấp cao: PYNQ cung cấp một bộ thư viện Python phong phú để tương tác với phần cứng, chẳng hạn như GPIO, I2C và các lõi IP tùy chỉnh. Các thư viện này trừu tượng hóa các chi tiết cấp thấp, cho phép các nhà phát triển tập trung vào logic ứng dụng.



Hình 2.4: PYNQ Framework [4].

Tương thích Phần cứng

PYNQ được thiết kế đặc biệt cho các thiết bị Zynq-7000 và Zynq UltraScale+ MPSoC. Các bo mạch như Kria KV260 được sử dụng rộng rãi trong cộng đồng cho mục đích giáo dục, tạo mẫu và sản xuất. tận dụng sự kết hợp giữa hệ thống xử lý (PS) và logic lập trình được (PL) vốn có trong các thiết bị Zynq, cho phép quy trình làm việc đồng thời thiết kế phần mềm-phần cứng liền mạch [4].

2.1.4. GIAO THỨC AXI4 VÀ IP AXI-DMA

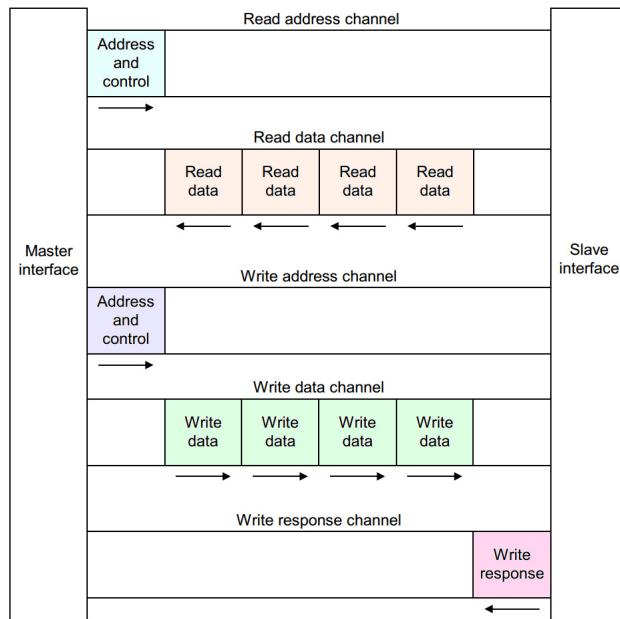
Giao thức AXI4 (Advanced eXtensible Interface 4) là một phần của đặc tả ARM AMBA (Advanced Microcontroller Bus Architecture), là một giao thức hiệu năng cao cho giao tiếp trong các thiết kế Hệ thống trên Chip (SoC), chẳng hạn như nền tảng Kria KV260. Nó cho phép truyền dữ liệu hiệu quả giữa các thành phần khác nhau, bao gồm bộ xử lý, bộ nhớ và thiết bị ngoại vi, điều này rất quan trọng để tối ưu hóa hiệu năng của các hệ thống nhúng trong các tác vụ như nhận dạng biển số xe.

AXI4 hỗ trợ các giao dịch cụm (burst transactions), cho phép nhiều mục dữ liệu được truyền trong một giao dịch duy nhất, điều này có lợi khi xử lý các tập dữ liệu lớn hoặc các luồng dữ liệu liên tục, chẳng hạn như trong xử lý hình ảnh. Khả năng truyền theo cụm này giảm thiểu độ trễ bằng cách giảm chi phí khởi tạo nhiều giao dịch riêng lẻ. Nó cũng sử dụng các kênh đọc và ghi riêng biệt, cho phép xử lý song song và tối đa hóa thông lượng, điều này rất cần thiết khi thực hiện các tác vụ đòi hỏi nhiều tính toán như học sâu hoặc nhận dạng hình ảnh thời gian thực.

Giao thức dựa trên kiến trúc chủ-tớ (master-slave), trong đó chủ (master) khởi tạo yêu cầu và tớ (slave) phản hồi. Thiết kế này cho phép phối hợp hiệu quả giữa các thành phần khác nhau trong hệ thống. Đối với các hệ thống dựa trên FPGA như Kria KV260, các giao diện AXI4 có thể được triển khai trong phần logic lập trình được (PL), cho phép linh hoạt và khả năng cấu hình lại để đáp ứng nhu cầu cụ thể của ứng dụng.

AXI4 cũng hỗ trợ đường ống (pipelining) địa chỉ và dữ liệu, giúp giảm thời gian chờ đợi dữ liệu có sẵn, do đó cải thiện hiệu năng hệ thống tổng thể. Điều này đặc biệt hữu ích trong các hệ thống thời gian thực nơi việc giảm thiểu độ trễ là rất quan trọng. Ngoài ra, khả năng tương thích của giao thức với các loại dữ liệu khác nhau và khả năng xử lý băng thông cao là rất cần thiết khi triển khai các thuật toán phức tạp, chẳng hạn như mạng nơ-ron tích chập.

2



Hình 2.5: Các kênh AXI [5].

Direct Memory Access (DMA) là một tính năng trong hệ thống máy tính cho phép một số phần cứng truy cập trực tiếp vào bộ nhớ chính mà không cần sự can thiệp của bộ xử lý trung tâm (CPU). Điều này giúp tăng hiệu suất bằng cách giảm tải cho CPU trong việc truyền dữ liệu giữa bộ nhớ và các thiết bị ngoại vi.

AXI DMA (AXI Direct Memory Access) là một IP core do Xilinx cung cấp, cho phép truyền dữ liệu tốc độ cao giữa bộ nhớ hệ thống (giao diện AXI4) và các thiết bị ngoại vi sử dụng giao thức AXI4-Stream. AXI DMA hỗ trợ cả hai chế độ truyền dữ liệu: Direct Mode và Scatter-Gather Mode, giúp linh hoạt trong việc quản lý và truyền dữ liệu.

Các mô-đun sử dụng giao thức AXI4-Stream có thể được kết nối theo chuỗi, trong đó mỗi khối thực hiện một chức năng xử lý cụ thể và truyền dữ liệu đến khối tiếp theo. Cấu trúc này cho phép xây dựng các pipeline xử lý dữ liệu hiệu quả, ví dụ như trong các ứng dụng xử lý tín hiệu số, xử

lý hình ảnh hoặc truyền thông tốc độ cao.

2.1.5. HỌC SÂU VÀ MẠNG NƠ-RON SÂU

Học Sâu (Deep Learning) là một tập hợp con của học máy liên quan đến việc sử dụng các mạng nơ-ron có nhiều lớp, được thiết kế để mô phỏng cách con người học và xử lý thông tin. Các mô hình này đặc biệt hiệu quả trong các tác vụ như nhận dạng hình ảnh, xử lý giọng nói và hiểu ngôn ngữ tự nhiên, nhờ khả năng tự động trích xuất các đặc trưng từ dữ liệu thô mà không cần kỹ thuật đặc trưng thủ công. Cốt lõi của học sâu là Mạng Nơ-ron Sâu (DNN - Deep Neural Network), bao gồm nhiều lớp nơ-ron (hoặc nút) được kết nối với nhau. Các lớp này bao gồm lớp đầu vào, nhiều lớp ẩn và lớp đầu ra.

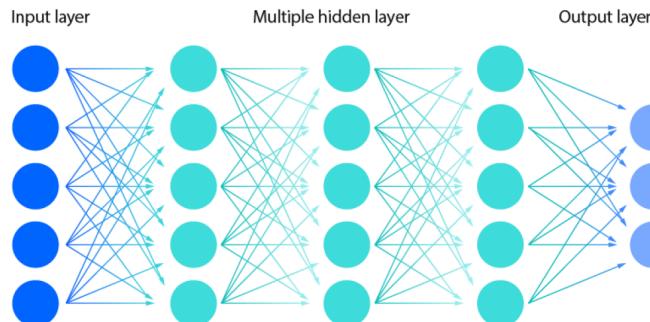
2

Nguyên tắc chính của DNN là khả năng học các biểu diễn phân cấp của dữ liệu. Nói một cách đơn giản, khi dữ liệu đi qua các lớp, mạng học được các đặc trưng trừu tượng hơn của dữ liệu đầu vào, làm cho nó rất hiệu quả cho các tác vụ phức tạp như nhận dạng biển số xe trong dự án của chúng tôi. Ví dụ, trong các tác vụ dựa trên hình ảnh, các lớp đầu tiên có thể học các đặc trưng cơ bản như cạnh hoặc kết cấu, trong khi các lớp sâu hơn học các mẫu phức tạp hơn như hình dạng hoặc đối tượng.

Việc huấn luyện các mạng này được thực hiện bằng cách sử dụng lan truyền ngược (backpropagation), một quá trình trong đó mạng điều chỉnh trọng số của nó dựa trên lỗi của đầu ra so với kết quả mong đợi. Quá trình học này đòi hỏi nhiều tính toán và thường yêu cầu tăng tốc phần cứng như GPU hoặc phần cứng chuyên dụng như FPGA, làm cho nó trở thành một nhiệm vụ phù hợp cho nền tảng Kria KV260 của chúng tôi.

Trong dự án FPGA của chúng tôi, các kỹ thuật Học Sâu như Mạng Nơ-ron Tích chập (Convolutional Neural Network) có thể được áp dụng để phát hiện biển số xe. Các mô hình này có thể được huấn luyện để nhận dạng và phân loại hình ảnh biển số xe, sử dụng các mô hình được huấn luyện trước và sau đó triển khai chúng trên FPGA để tận dụng khả năng

Deep neural network



Hình 2.6: Mạng nơ-ron sâu [6].

xử lý song song của phần cứng. Giao thức AXI4, như đã đề cập trước đó, có thể được sử dụng để kết nối hiệu quả các bộ tăng tốc học sâu trong phần logic lập trình được của Kria KV260 với hệ thống xử lý.

Bằng cách triển khai các mô hình học sâu này trực tiếp trên phần cứng, chúng tôi có thể đạt được những cải tiến hiệu năng đáng kể so với việc chạy chúng trên CPU hoặc GPU truyền thống. Điều này cho phép xử lý thời gian thực trong các hệ thống nhúng, chẳng hạn như những hệ thống được yêu cầu trong các hệ thống nhận dạng biển số xe tự động.

2.1.6. MẠNG NƠ-RON TÍCH CHẬP (CNN)

Mạng Nơ-ron Tích chập (CNN - Convolutional Neural Network) là một loại mô hình học sâu chuyên biệt được thiết kế để xử lý dữ liệu có cấu trúc dạng lưới, chẳng hạn như hình ảnh hoặc chuỗi thời gian. Không giống như các mạng nơ-ron kết nối đầy đủ truyền thống, CNN tận dụng kiến trúc của chúng để nhận dạng hiệu quả các mẫu, cấu trúc và phân cấp trong dữ liệu, làm cho chúng đặc biệt hiệu quả cho các tác vụ như phân loại hình ảnh, phát hiện đối tượng và trích xuất đặc trưng.

Các Thành phần Chính của một CNN

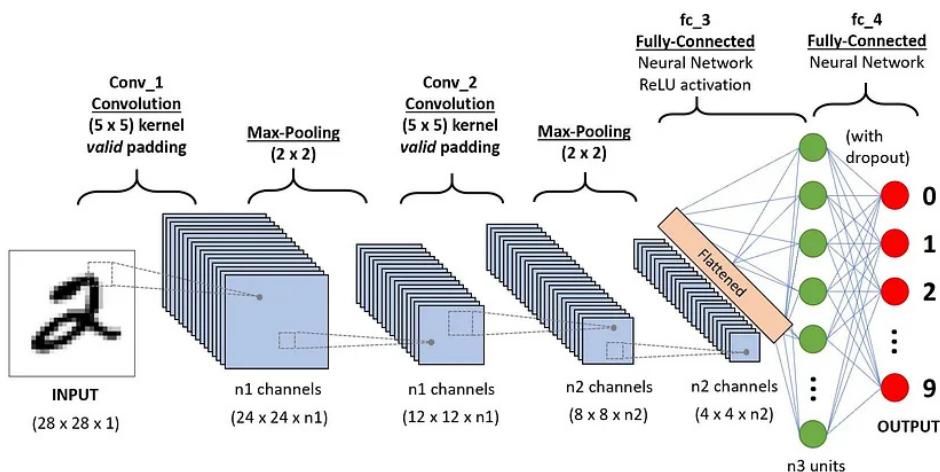
1. Các Lớp Tích chập (Convolutional Layers): Lớp tích chập áp dụng một tập hợp các bộ lọc (filter) vào dữ liệu đầu vào để trích xuất các đặc trưng quan trọng. Các bộ lọc trượt trên dữ liệu đầu vào, thực hiện các phép nhân tích vô hướng, giúp xác định các mẫu như cạnh, kết cấu hoặc hình dạng.
2. Các Lớp Gộp (Pooling Layers): Các lớp này giảm kích thước không gian của dữ liệu trong khi vẫn giữ lại các đặc trưng quan trọng nhất, làm cho mạng hiệu quả hơn về mặt tính toán. Các phương pháp gộp phổ biến bao gồm gộp cực đại (max pooling) và gộp trung bình (average pooling).
3. Các Lớp Kết nối Đầy đủ (Fully Connected Layers): Ở cuối mạng, các lớp kết nối đầy đủ tổng hợp các đặc trưng đã được trích xuất để tạo ra đầu ra phân loại hoặc dự đoán cuối cùng.
4. Các Hàm Kích hoạt (Activation Functions): Các hàm phi tuyến, chẳng hạn như ReLU (Rectified Linear Unit), được áp dụng sau các phép toán tích chập để giới thiệu tính phi tuyến, cho phép mạng mô hình hóa các mối quan hệ phức tạp.
5. Dropout và Điều chỉnh chuẩn hóa (Regularization): Các kỹ thuật như dropout được sử dụng để ngăn chặn việc quá khớp (overfitting) bằng cách ngẫu nhiên vô hiệu hóa một số nơ-ron trong quá trình huấn luyện.

CNN xử lý dữ liệu thông qua một chuỗi các lớp, mỗi lớp tập trung vào các cấp độ phân cấp khác nhau của các đặc trưng. Ví dụ, các lớp đầu có thể phát hiện các đặc trưng đơn giản như các cạnh, trong khi các lớp sâu hơn kết hợp chúng để xác định các mẫu phức tạp như đối tượng hoặc cảnh.

Đối với các tác vụ như nhận dạng biển số xe, CNN đóng vai trò quan trọng trong việc trích xuất các đặc trưng liên quan từ dữ liệu hình ảnh,

chẳng hạn như đường viền của các ký tự hoặc bộ cục của biển số. Bằng cách huấn luyện một mô hình CNN sử dụng tập dữ liệu hình ảnh biển số xe được chú thích, mạng học cách xác định và định vị các mẫu cụ thể, hỗ trợ phát hiện chính xác và hiệu quả.

2



Hình 2.7: Kiến trúc CNN áp dụng cho nhận dạng chữ số [7].

Nền tảng này sẽ được tích hợp với khả năng tăng tốc phần cứng của nền tảng Kria KV260, tận dụng kiến trúc FPGA của nó để tối ưu hóa các hoạt động đòi hỏi nhiều tính toán của CNN.

2.1.7. MÔ HÌNH YOU ONLY LOOK ONCE (YOLO)

Mô hình You Only Look Once (YOLO) là một khung phát hiện đối tượng thời gian thực được công nhận rộng rãi về tốc độ và độ chính xác. Không giống như các mô hình phát hiện đối tượng truyền thống dựa vào nhiều lượt quét qua hình ảnh để phân loại và định vị, YOLO xử lý toàn bộ hình ảnh trong một lượt duy nhất, giúp tăng hiệu quả đáng kể.

Các Khái niệm Chính của YOLO

1. Khung Phát hiện Thông nhât: YOLO coi việc phát hiện đối tượng như một bài toán hồi quy duy nhất, trực tiếp dự đoán các hộp giới

hạn (bounding boxes) và xác suất lớp từ một hình ảnh đầu vào. Điều này loại bỏ sự cần thiết của các quy trình riêng biệt cho việc đề xuất vùng và phân loại.

- 2
2. Dự đoán dựa trên Lưới: YOLO chia hình ảnh đầu vào thành một lưới $S \times S$. Mỗi ô lưới dự đoán các hộp giới hạn, điểm tin cậy và xác suất lớp cho các đối tượng chồng lên ô đó.
 3. Hộp Neo (Anchor Boxes): Để xử lý các đối tượng có kích thước và tỷ lệ khung hình khác nhau, YOLO sử dụng các hộp neo được xác định trước, đại diện cho các hình dạng hộp giới hạn phổ biến.
 4. Hàm Mất mát (Loss Function): Mô hình tối ưu hóa một hàm mất mát tùy chỉnh cân bằng lỗi định vị, lỗi phân loại và dự đoán điểm tin cậy.

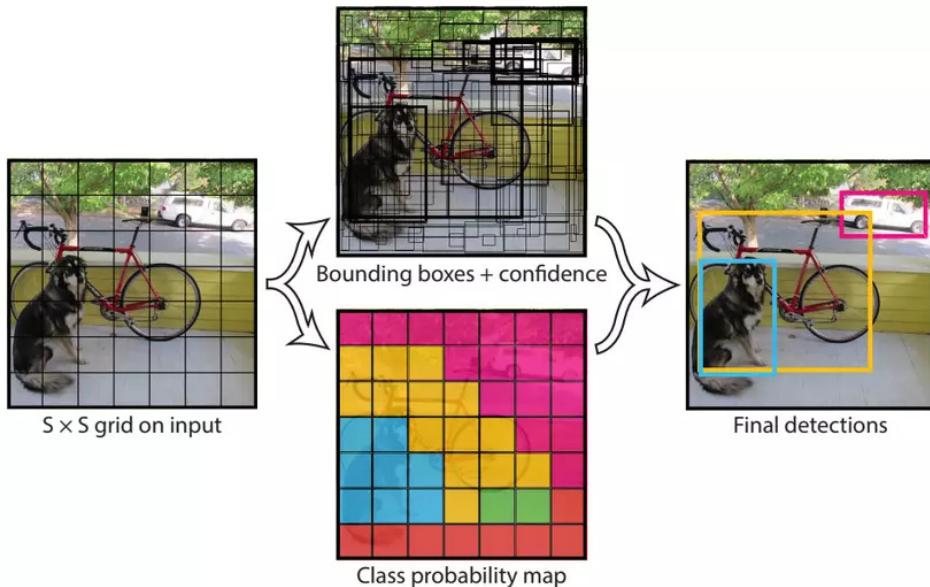
Ưu điểm của YOLO

- Hiệu năng Thời gian thực: YOLO rất hiệu quả, phù hợp cho các ứng dụng thời gian thực.
- Nhận thức Ngũ cảnh Toàn cục: Bằng cách phân tích toàn bộ hình ảnh trong một lần, YOLO giảm thiểu các dương tính giả từ các đối tượng nền.
- Tốc độ Cao: Các mô hình như YOLOv4 và YOLOv5 có thể xử lý hình ảnh độ phân giải cao với tốc độ vượt quá 60 khung hình mỗi giây (FPS) trên các GPU hiện đại.

Trong bối cảnh phát hiện biển số xe trên nền tảng Kria KV260, YOLO cung cấp một giải pháp hiệu quả để phát hiện và định vị biển số trong các luồng video thời gian thực. Kiến trúc nhẹ của nó, kết hợp với khả năng tăng tốc dựa trên FPGA, đảm bảo độ trễ và mức tiêu thụ điện năng tối thiểu, lý tưởng cho các ứng dụng biên. Bằng cách huấn luyện một mô hình YOLO trên tập dữ liệu hình ảnh biển số xe được chú thích, nó có thể

tổng quát hóa để phát hiện biến số trong các điều kiện khác nhau, chẳng hạn như ánh sáng hoặc góc nhìn khác nhau.

2



Hình 2.8: Phát hiện bằng YOLO[8].

2.2. CÔNG TRÌNH LIÊN QUAN

2.2.1. CÁC PHƯƠNG PHÁP NHẬN DẠNG BIỂN SỐ XE TỰ ĐỘNG (ALPR)

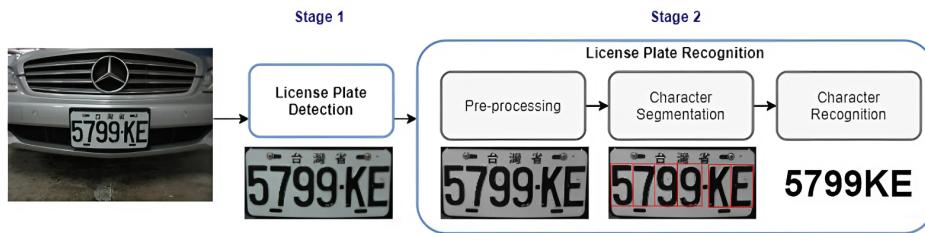
Hệ thống Nhận dạng Biển số Xe Tự động (ALPR) có thể được phân loại rộng rãi thành các phương pháp **đa giai đoạn** và **đơn giai đoạn**. Mỗi phương pháp có ưu và nhược điểm riêng, tùy thuộc vào yêu cầu về độ chính xác, tốc độ và độ phức tạp triển khai.

Hệ thống Nhận dạng Biển số Xe Đa giai đoạn:

Đây là phương pháp truyền thống và phổ biến nhất, chia nhiệm vụ thành nhiều giai đoạn độc lập, thường là:

- **Phát hiện Biển số Xe:** Xác định vị trí biển số trong ảnh đầu vào.
- **Phân đoạn Ký tự:** Tách các ký tự riêng lẻ trên biển số.

- **Nhận dạng Ký tự:** Nhận dạng các ký tự đã được phân đoạn.



Hình 2.9: Các giai đoạn chính trong hệ thống nhận dạng biển số xe đa giai đoạn [9].

Trong giai đoạn phát hiện, các kỹ thuật thị giác máy tính truyền thống thường dựa vào đặc trưng như hình dạng, màu sắc, kết cấu. Tuy nhiên, việc tích hợp các phương pháp học sâu, như mô hình phát hiện đối tượng **You Only Look Once (YOLO)**, đã cải thiện đáng kể độ chính xác và độ bền vững. YOLO đặc biệt phù hợp do hiệu suất thời gian thực và khả năng phát hiện đối tượng trong một lượt duy nhất. Giai đoạn phân đoạn ký tự có thể không luôn cần thiết. Giai đoạn cuối cùng là nhận dạng ký tự, thường sử dụng mạng nơ-ron hoặc kỹ thuật khớp mẫu. Hệ thống đa giai đoạn nhạy cảm với lỗi ở các giai đoạn đầu.

Nghiên cứu này áp dụng **hệ thống ALPR đa giai đoạn** và sử dụng YOLO cho cả nhiệm vụ phát hiện biển số và nhận dạng ký tự, nhằm cải thiện hiệu quả và độ chính xác trong khi vẫn duy trì tính mô-đun.

Hệ thống Nhận dạng Biển số Xe Đơn giai đoạn:

Ngược lại, hệ thống đơn giai đoạn hợp nhất tất cả các bước vào một mô hình học sâu duy nhất, được huấn luyện end-to-end để thực hiện phát hiện, định vị và nhận dạng trong một lượt. Phương pháp này có thể đạt được thời gian suy luận nhanh hơn và giảm độ phức tạp của mô hình bằng cách chia sẻ tham số giữa các tác vụ (ví dụ: sử dụng CNN được tùy chỉnh, tối ưu hóa đồng thời phát hiện và nhận dạng). Tuy nhiên, chúng có thể thiếu tính linh hoạt và mô-đun so với phương pháp đa giai đoạn.

Do tính chất mô-đun và linh hoạt cần thiết, dự án này tập trung vào

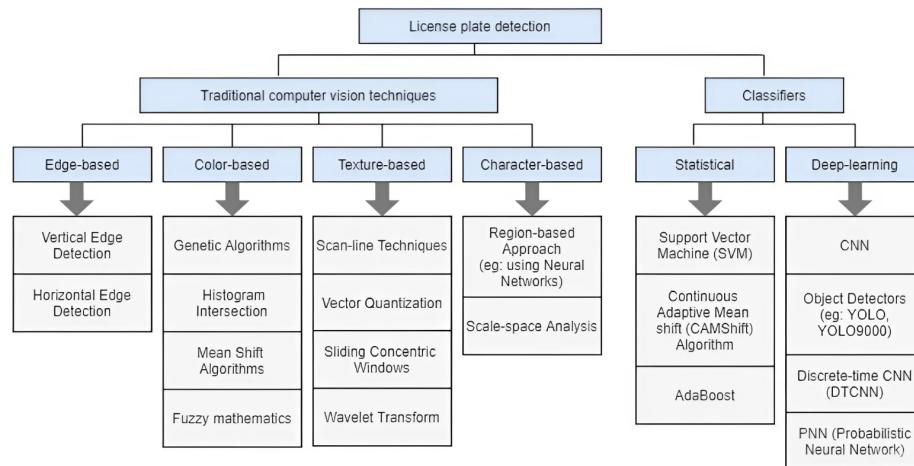
phương pháp đa giai đoạn, cho phép tối ưu hóa mục tiêu ở từng giai đoạn và sử dụng YOLO để tăng cường cả quá trình phát hiện và nhận dạng.

2

2.2.2. PHÁT HIỆN BIỂN SỐ XE

Phát hiện biển số xe là một bước quan trọng. Biển số thường được định nghĩa là "một tấm kim loại hoặc nhựa gắn vào xe để nhận dạng duy nhất". Đối với máy móc, nó có thể được mô tả là "một vùng hình chữ nhật trên xe có mật độ cạnh ngang và dọc cao" [9].

Các phương pháp phát hiện bao gồm kỹ thuật thị giác máy tính truyền thống và phương pháp dựa trên học sâu.



Hình 2.10: Phân loại các kỹ thuật phát hiện biển số xe liên quan [9].

Các Phương pháp Truyền thống:

Các phương pháp này dựa vào đặc trưng thủ công như cạnh, màu sắc, kết cấu. Chúng hiệu quả về mặt tính toán nhưng thường nhạy cảm với thay đổi môi trường (ánh sáng, nhiễu).

- **Phương pháp Dựa trên Cạnh:** Tận dụng tính chất hình chữ nhật và tỷ lệ khung hình của biển số (ví dụ: bộ lọc Sobel). Dễ bị nhiễu và

dương tính giả. Các cải tiến có thể đạt độ chính xác cao [12].

- **Phương pháp Dựa trên Màu sắc:** Khai thác sự tương phản màu sắc (ví dụ: mô hình HLS). Gặp khó khăn với thay đổi ánh sáng và trùng lặp màu.
- **Phương pháp Dựa trên Kết cấu:** Phân tích đặc điểm kết cấu độc đáo (ví dụ: bộ lọc Gabor, biến đổi wavelet). Mạnh mẽ trước biến dạng nhưng tốn kém tính toán và nhạy cảm với nền phức tạp.

2

Các Phương pháp Học Sâu:

Các phương pháp học sâu đã cho thấy hiệu quả vượt trội.

- Selmi và cộng sự [13] đã giới thiệu phương pháp định vị dựa trên CNN, đạt kết quả tốt trên tập dữ liệu Caltech.
- Zou và cộng sự [14] kết hợp CNN nông và sâu để huấn luyện end-to-end, giảm chi phí tính toán và tăng độ chính xác.
- YOLO được chú ý nhờ khả năng phát hiện đối tượng thời gian thực. Laroca và cộng sự [15] đã sử dụng YOLOv2 cho phát hiện xe và biển số với hiệu suất mạnh mẽ.
- Để giải quyết các hạn chế của YOLO trong các tình huống phức tạp, Gee-Sern và cộng sự [16] đã sửa đổi YOLO bằng cách điều chỉnh kích thước lưới và tham số hộp giới hạn. Xie và cộng sự [17] đề xuất MD-YOLO, bổ sung dự đoán góc quay, cải thiện khả năng phát hiện trong điều kiện ánh sáng yếu hoặc bị che khuất.

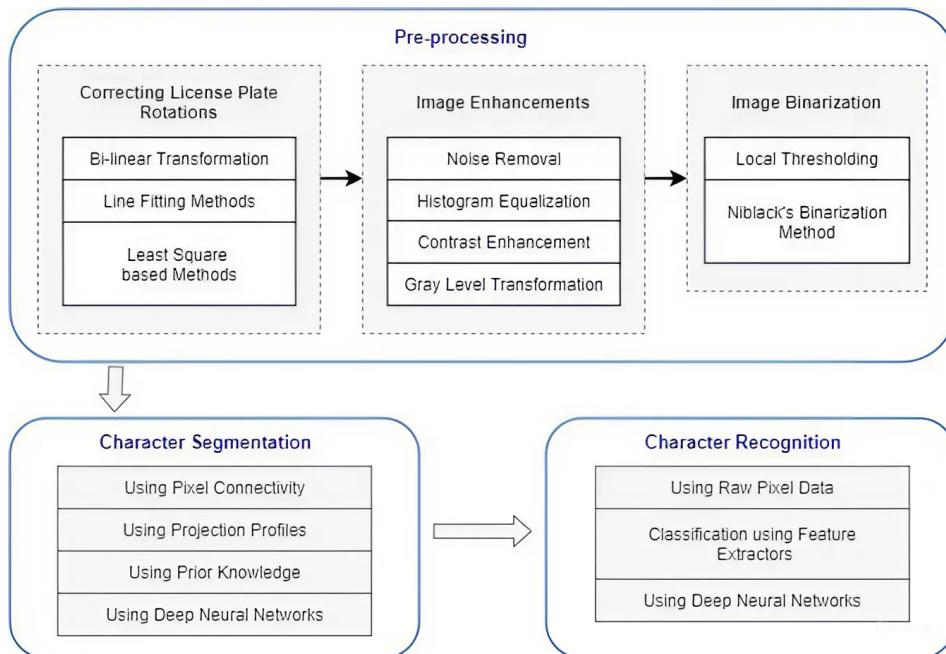
Trong luận văn này, chúng tôi sử dụng YOLO do hiệu quả và độ bền vững của nó trong việc phát hiện biển số xe trong các môi trường đa dạng.

2.2.3. NHẬN DẠNG BIỂN SỐ XE

Nhận dạng biển số xe là giai đoạn thứ hai, nhằm mục đích trích xuất và diễn giải các ký tự từ biển số đã được định vị. Đây là một dạng Nhận dạng

Kỹ tự Quang học (OCR) chuyên biệt, đối mặt với các thách thức do môi trường ngoài trời (ánh sáng thay đổi, thời tiết, hư hỏng, xoay biển số).

2



Hình 2.11: Quy trình nhận dạng biển số xe với các kỹ thuật liên quan [9].

Các Kỹ thuật Tiền xử lý:

Tiền xử lý rất quan trọng để giảm thiểu ảnh hưởng của môi trường và chuẩn bị hình ảnh biển số. Các kỹ thuật phổ biến bao gồm:

- Chỉnh sửa Độ xoay:** Sử dụng các phép biến đổi song tuyến tính, phương pháp bình phương tối thiểu, hoặc khớp đường thẳng để căn chỉnh biển số bị xoay [18] [19] [20].
- Nâng cao Chất lượng Ảnh:** Loại bỏ nhiễu, cân bằng biểu đồ độ sáng (histogram equalization), tăng cường độ tương phản, và biến đổi mức xám [21].
- Nhi phân hóa:** Chuyển đổi ảnh thành dạng nhị phân (đen trắng).

Các kỹ thuật nhị phân hóa thích ứng (ví dụ: ngưỡng cục bộ, phương pháp Niblack) thường hiệu quả hơn ngưỡng cố định [22].

Tuy nhiên, tiền xử lý không thể giải quyết hoàn hảo mọi vấn đề.

2

Phân đoạn Ký tự:

Việc phân đoạn chính xác các ký tự là rất cần thiết. Các kỹ thuật phổ biến:

1. **Kết nối Điểm ảnh (Pixel Connectivity):** Xác định các vùng điểm ảnh được kết nối dựa trên kích thước và tỷ lệ khung hình. Mạnh mẽ trước xoay nhưng gặp khó khăn với ký tự bị gãy hoặc dính liền [23].
2. **Biểu đồ Chiếu (Projection Profiles):** Sử dụng phép chiếu dọc và ngang để xác định ranh giới ký tự. Hoạt động tốt trên ảnh sạch nhưng nhạy cảm với nhiễu và xoay [21].
3. **Kiến thức Tiên nghiệm (Prior Knowledge):** Sử dụng giả định về kích thước hoặc vị trí ký tự. Dễ thực hiện nhưng thiếu tính tổng quát [24].
4. **Học Sâu:** Sử dụng mạng nơ-ron (ví dụ: CNN) để thực hiện phân đoạn, thường là một phần của quy trình end-to-end. Mạnh mẽ và linh hoạt hơn nhưng đòi hỏi tài nguyên tính toán và dữ liệu lớn [15] [25].

Nhận dạng Ký tự:

Sau khi phân đoạn (hoặc đôi khi không cần phân đoạn), các ký tự được phân loại bằng các phương pháp:

1. **Khớp Mẫu (Template Matching):** So sánh các đoạn với mẫu được xác định trước. Đơn giản nhưng gặp khó khăn với sự thay đổi font chữ, xoay hoặc biến dạng [26] [27].

2. **Trích xuất Đặc trưng và Học Máy:** Trích xuất đặc trưng và phân loại bằng các mô hình như SVM hoặc HMM. Hiệu quả tính toán và mạnh mẽ trước biến dạng nhỏ, nhưng độ chính xác phụ thuộc vào chất lượng trích xuất đặc trưng [22] [28].
3. **Học Sâu:** Mạng nơ-ron (đặc biệt là CNN) tích hợp trích xuất đặc trưng và phân loại. Các mô hình phát hiện đối tượng như YOLO ngày càng được sử dụng, cho phép phát hiện và phân loại đồng thời tất cả các ký tự. Mặc dù tốn kém tính toán, chúng đạt độ chính xác và độ bền vững cao hơn [25] [29].

Các phương pháp học sâu, bao gồm CNN và mô hình dựa trên YOLO, đã chứng minh độ chính xác và khả năng thích ứng vượt trội. YOLO nổi bật với khả năng phát hiện và phân loại ký tự trong một lượt duy nhất, giảm độ phức tạp.

Xem xét những phân tích trên, công trình này áp dụng phương pháp dựa trên YOLO để nhận dạng biển số xe. YOLO kết hợp phát hiện và phân loại ký tự vào một thông nhât, giải quyết hiệu quả các vấn đề như xoay, hư hỏng và điều kiện ánh sáng thay đổi. Cơ chế phát hiện dựa trên lưới của nó cũng đơn giản hóa việc sắp xếp các ký tự. Phương pháp đề xuất nhằm đạt được độ chính xác và độ bền vững cao trong điều kiện thực tế.

3

KIẾN TRÚC PHẦN MỀM ĐỀ XUẤT

Dựa trên những tiến bộ trong lĩnh vực phát hiện đối tượng sử dụng AI và khả năng tính toán của các FPGA hiện đại, chương này đề xuất một kiến trúc mới cho hệ thống Nhận dạng Biển số Xe (ALPR) thời gian thực. Thiết kế này tận dụng các tài nguyên xử lý của nền tảng Xilinx Kria KV260, phân chia nhiệm vụ một cách chiến lược giữa Hệ thống Xử lý (PS) và Logic Lập trình được (PL).

3.1. QUY TRÌNH NHẬN DIỆN BIỂN SỐ ĐỀ XUẤT

Phần này giới thiệu quy trình được đề xuất cho hệ thống Nhận dạng Biển số Xe Tự động (ALPR) thời gian thực, được thiết kế để giải quyết những thách thức khi triển khai các mô hình học sâu đòi hỏi tính toán cao trên các hệ thống nhúng có tài nguyên hạn chế. Hệ thống tận dụng một phương pháp dựa trên AI mạnh mẽ và hiệu quả, lấy cảm hứng từ bài báo "Character Time-series Matching For Robust License Plate Recognition" [30],

3

để đạt được độ chính xác cao và hiệu năng thời gian thực. Quy trình được thiết kế đặc biệt để xử lý các luồng video từ camera, phù hợp với nhiều ứng dụng khác nhau, bao gồm quản lý giao thông thông minh, thu phí tự động và hỗ trợ thực thi pháp luật. Nhiệm vụ đòi hỏi tính toán cao là phát hiện biển số xe được chuyển sang Logic Lập trình được (PL) của FPGA để tăng tốc, trong khi các nhiệm vụ còn lại được xử lý bởi Hệ thống Xử lý (PS).

Quy trình ALPR được đề xuất, minh họa trong Hình 3.1, bao gồm các giai đoạn chính sau:

Các Giai đoạn của Quy trình:

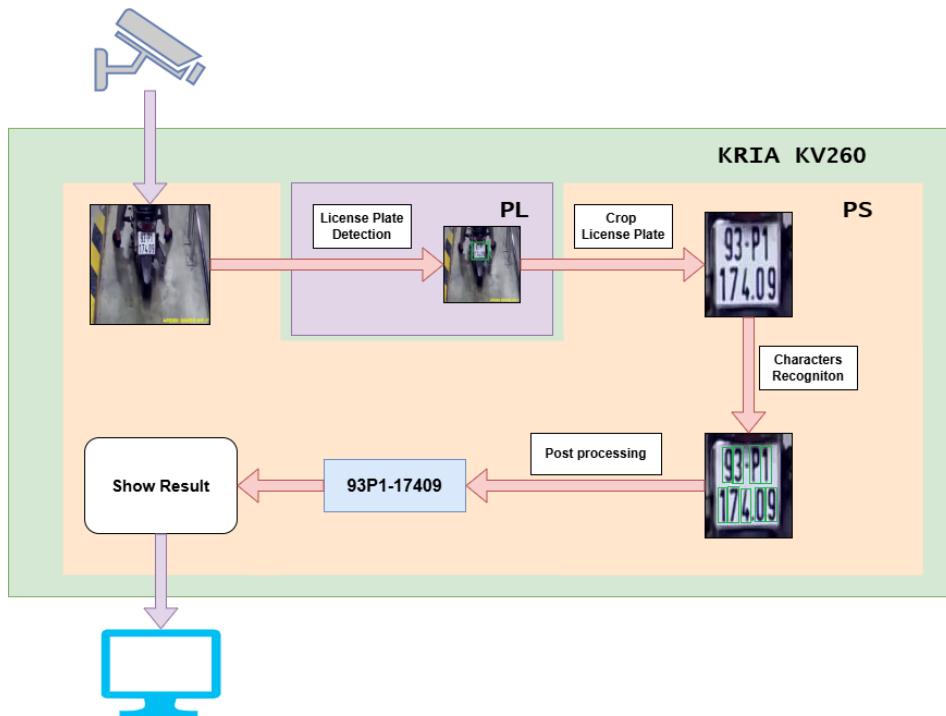
- Thu thập Ảnh (PS)** Hệ thống bắt đầu bằng việc thu thập hình ảnh từ camera được kết nối với nền tảng Zynq. Giao diện camera được quản lý bởi PS, tận dụng khả năng của bộ xử lý ARM để xử lý giao tiếp ngoại vi. Sau khi nhận được hình ảnh PS sẽ biến đổi và chuyển hình ảnh về dạng mã hóa nhị phân để đưa xuống phía PL.
- Phát hiện Biển số Xe (PL):** Giai đoạn này, được tăng tốc trên PL, sử dụng một mô hình custom dựa theo YOLO để xác định các biển số xe tiềm năng trong ảnh đầu vào. Phương pháp này được chọn do hiệu năng vượt trội trong việc cân bằng tốc độ và độ chính xác so với các kỹ thuật thị giác máy tính truyền thống. Khả năng vốn có của mô hình dựa theo YOLO trong việc xử lý toàn bộ hình ảnh trong một lượt duy nhất làm cho nó đặc biệt phù hợp với các ứng dụng thời gian thực như ALPR.
 - Đầu vào:** Dữ liệu hình ảnh ở dạng nhị phân được nhận từ PS.
 - Xử lý:** Sử dụng thiết kế đã được tăng tốc để tính toán ra output tensor của layer conv cuối cùng.
 - Đầu ra:** Output tensor của layer cuối cùng, được đưa lên PS áp dụng Post-Processing để tìm ra bounding box của biển số.

- **Triển khai:** Bước này sẽ được thực thi trên Logic Lập trình được (PL) để tận dụng khả năng xử lý song song của FPGA.
3. **Cắt ảnh Biển số (PS)** Dựa trên output tensor nhận được từ PL, PS sẽ áp dụng NMS để tìm ra bounding box của biển số, PS cắt các vùng tương ứng từ ảnh gốc dựa trên bounding box đó. Mỗi ảnh được cắt đại diện cho một biển số xe tiềm năng để xử lý tiếp. Bước này cô lập các vùng quan tâm, giảm tải tính toán cho các giai đoạn tiếp theo.
4. **Nhận dạng Ký tự (PS):** Giai đoạn này tập trung vào việc nhận dạng các ký tự riêng lẻ trong ảnh biển số xe. Dựa trên những tiến bộ trong nhận dạng ký tự dựa trên học sâu, một mô hình YOLO đã sửa đổi được sử dụng. Mô hình này đã được huấn luyện đặc biệt để vừa phát hiện vừa phân loại ký tự, cung cấp một phương pháp được tinh giản so với các phương pháp OCR truyền thống thường bao gồm các bước phân đoạn và phân loại riêng biệt.
- **Đầu vào:** Ảnh biển số xe đã cắt.
 - **Xử lý:** Mô hình YOLO tính toán kết quả.
 - **Đầu ra:** Các ký tự được nhận dạng cùng với các bounding-box và điểm tin cậy của chúng.
5. **Hậu xử lý (PS):** Các ký tự được nhận dạng được nối lại để tạo thành chuỗi biển số hoàn chỉnh.
- **Đầu vào:** Các ký tự được nhận dạng cùng với điểm tin cậy.
 - **Xử lý:** Hình thành chuỗi dựa trên thứ tự các ký tự được nhận dạng và xác thực tùy chọn dựa trên quy tắc.
 - **Đầu ra:** Chuỗi biển số xe được nhận dạng.
6. **Hiển thị Kết quả (PS):** Để cung cấp phản hồi trực quan, ảnh gốc được bổ sung các bounding-box xung quanh các biển số xe được phát hiện, và các chuỗi biển số được nhận dạng tương ứng được

hiển thị gần đó. Điều này cho phép theo dõi và xác minh dễ dàng đầu ra của hệ thống.

3

- **Đầu vào:** Ảnh gốc, các bounding-box của biển số xe, và các chuỗi biển số được nhận dạng.
- **Xử lý:** Chồng các bounding-box và chuỗi được nhận dạng lên ảnh gốc.
- **Output:** Ảnh hiển thị các biển số xe được phát hiện và chuỗi được nhận dạng của chúng.



Hình 3.1: Quy trình ALPR tổng thể

Lý giải về phương pháp dựa trên AI và thiết kế quy trình:

- **Hiệu năng Thời gian thực:** Việc chuyển nhiệm vụ phát hiện biển số xe dựa trên mô hình custom theo YOLO đòi hỏi tính toán cao sang

PL của FPGA cho phép xử lý thời gian thực, đáp ứng yêu cầu quan trọng của các ứng dụng ALPR.

- **Độ chính xác Cao:** Tận dụng các mô hình học sâu, đặc biệt là YOLO đã sửa đổi để nhận dạng ký tự, đảm bảo độ chính xác cao so với các kỹ thuật thị giác máy tính truyền thống.
- **Khả năng Thích ứng:** Quy trình có thể được điều chỉnh cho các định dạng và tiêu chuẩn biển số xe khác nhau bằng cách huấn luyện lại mô hình nhận dạng ký tự trên các tập dữ liệu liên quan.
- **Sử dụng Tài nguyên Hiệu quả:** Kiến trúc được đề xuất sử dụng hiệu quả các tài nguyên của nền tảng Zynq bằng cách phân phối các tác vụ giữa PS và PL dựa trên yêu cầu và khả năng tính toán của chúng.

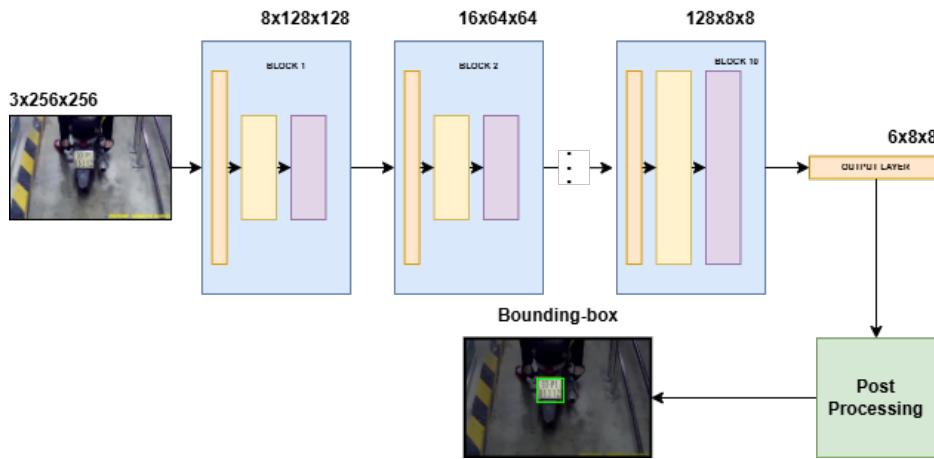
3

Quy trình này ưu tiên sự cân bằng giữa tốc độ, độ chính xác và hiệu quả. Các phần tiếp theo sẽ thảo luận về kiến trúc của bộ tăng tốc lõi YOLO trên PL và trình bày chi tiết hơn về việc triển khai và đánh giá hệ thống.

3.2. KIẾN TRÚC ĐỀ XUẤT CHO PHÁT HIỆN BIỂN SỐ XE

Để giải quyết bài toán phát hiện biển số xe trong ảnh đầu vào, nhóm đề xuất một mô hình CNN dựa trên kiến trúc YOLO (You Only Look Once) nhưng không sử dụng anchor box ('YoloNoAnchor'). Mô hình này được thiết kế để có kích thước nhẹ, phù hợp cho việc triển khai trên phần cứng. Mô hình nhận đầu vào là một ảnh RGB (kích thước $3 \times 256 \times 256$) và trả về một bản đồ đặc trưng (feature map) chứa thông tin dự đoán về vị trí và loại đối tượng (biển số xe) trên một lưới ô (grid). Hình 3.2 minh họa kiến trúc tổng thể của mô hình.

3



Hình 3.2: Kiến trúc tổng thể của mô hình phát hiện biển số xe.

Kiến trúc của mô hình bao gồm một chuỗi các khối convolutional có stride là 1 và 2 để trích xuất đặc trưng và giảm kích thước không gian, theo sau là các lớp convolutional cuối cùng để đưa ra dự đoán.

Giai đoạn Trích xuất Đặc trưng (Feature Extraction Stages):

Phần này bao gồm một chuỗi các khối convolutional, mỗi khối thường gồm một lớp Convolution (Conv), sau đó là Batch Normalization (BN) để ổn định quá trình huấn luyện và hàm kích hoạt phi tuyến tính ReLU (nn.ReLU(inplace=True)). Thay vì sử dụng các lớp Max Pooling riêng biệt, việc giảm kích thước không gian được thực hiện trực tiếp bằng cách sử dụng **stride=2** trong một số lớp convolutional.

- **Khối 1 (conv1, bn1, relu1):** Nhận ảnh đầu vào 3 kênh (3x256x256).
 - Sử dụng conv(3→16, k=3, s=2, p=1) + bn + relu.
 - Tác dụng: Trích xuất đặc trưng cơ bản và giảm kích thước không gian xuống một nửa (16x128x128).
- **Khối 2 (conv2, bn2, relu2):**

- Sử dụng conv(16→32, k=3, s=2, p=1) + bn + relu.
- Tác dụng: Tiếp tục trích xuất đặc trưng và giảm kích thước không gian xuống H/4, W/4 (32x64x64).

• **Khối 3 (conv3, bn3, relu3):**

- Sử dụng conv(32→64, k=3, s=1, p=1) + bn + relu.
- Tác dụng: Tăng độ sâu đặc trưng lên 64 kênh, giữ nguyên kích thước không gian (64x64x64).

3

• **Khối 4 (conv4, bn4, relu4):** Sử dụng lớp conv 1x1 với stride=2.

- Sử dụng conv(64→32, k=1, s=2, p=0) + bn + relu.
- Tác dụng: Giảm số kênh xuống 32 (hoạt động như lớp bottleneck) và đồng thời giảm kích thước không gian xuống H/8, W/8 (32x32x32). Đây là cách giảm chiều hiệu quả.

• **Khối 5 (conv5, bn5, relu5):**

- Sử dụng conv(32→64, k=3, s=1, p=1) + bn + relu.
- Tác dụng: Tăng lại độ sâu đặc trưng lên 64 kênh, giữ nguyên kích thước không gian (64x32x32).

• **Khối 6 (conv6, bn6, relu6):** Lớp bottleneck 1x1.

- Sử dụng conv(64→32, k=1, s=1, p=0) + bn + relu.
- Tác dụng: Giảm số kênh xuống 32, giữ nguyên kích thước không gian (32x32x32).

• **Khối 7 (conv7, bn7, relu7):**

- Sử dụng conv(32→64, k=3, s=2, p=1) + bn + relu.
- Tác dụng: Trích xuất đặc trưng và giảm kích thước không gian xuống H/16, W/16 (64x16x16).

3

- **Khối 8 (conv8, bn8, relu8):** Sử dụng lớp conv 1x1 với stride=2.
 - Sử dụng conv(64→128, k=1, s=2, p=0) + bn + relu.
 - Tác dụng: Tăng số kênh lên 128 và giảm kích thước không gian xuống H/32, W/32 (128x8x8).
- **Khối 9 (conv9, bn9, relu9):** Lớp bottleneck 1x1.
 - Sử dụng conv(128→64, k=1, s=1, p=0) + bn + relu.
 - Tác dụng: Giảm số kênh xuống 64, giữ nguyên kích thước không gian (64x8x8).
- **Khối 10 (conv10, bn10, relu10):**
 - Sử dụng conv(64→128, k=1, s=1, p=0) + bn + relu.
 - Tác dụng: Tăng lại số kênh lên 128, giữ nguyên kích thước không gian (128x8x8). Đây là feature map cuối cùng trước lớp output.

Việc sử dụng ‘stride=2’ trong các lớp convolution (cả 3x3 và 1x1) thay thế hiệu quả cho các lớp Max Pooling, giúp kiến trúc trở nên đồng nhất hơn và có thể thuận lợi hơn cho việc tối ưu hóa phần cứng pipeline. Các lớp bottleneck 1x1 giúp giảm độ phức tạp tính toán trước khi áp dụng các lớp 3x3 sâu hơn hoặc để điều chỉnh số lượng kênh.

Lớp Output (Output Layer):

Sau giai đoạn trích xuất đặc trưng, một lớp convolutional cuối cùng được sử dụng để tạo ra bản đồ dự đoán.

- **Lớp out_conv:** Đây là một lớp conv 1x1 (conv(128, 5 + num_classes, k=1, s=1, p=0)) không có batch normalization hay hàm kích hoạt theo sau. Lớp này biến đổi bản đồ đặc trưng 128 kênh thành bản đồ dự đoán cuối cùng.
- **Ý nghĩa Output:** Với num_classes = 1 (chỉ phát hiện biển số xe), lớp output sẽ có $5 + 1 = 6$ kênh. Tại mỗi vị trí (i, j) trên lưới ô của bản đồ output 8x8, 6 giá trị này đại diện cho:

- 4 giá trị: Thông tin về bounding-box (bounding box), ví dụ: tọa độ tâm (x, y), chiều rộng (w), chiều cao (h).
- 1 giá trị: Điểm tin cậy về sự tồn tại của đối tượng (objectness score).
- 1 giá trị: Xác suất lớp của đối tượng (ở đây là xác suất là biển số xe).

3

Hậu xử lý (Post-processing):

Output thô từ lớp out_conv là một tensor (bản đồ đặc trưng) chứa các dự đoán cho mỗi ô trong lưới 8×8 . Để thu được kết quả phát hiện cuối cùng dưới dạng các bounding-box (bounding boxes) trên ảnh gốc, các bước hậu xử lý sau được thực hiện (như minh họa trong khối "Post Processing" ở Hình 3.2):

- **Giải mã Output:** Chuyển đổi các giá trị output thô (tọa độ tương đối, chiều rộng/cao tương đối so với ô lưới) thành tọa độ bounding box thực tế trên ảnh gốc. Đồng thời tính toán điểm tin cậy cuối cùng cho mỗi box.
- **Áp dụng Ngưỡng Tin cậy (Confidence Thresholding):** Loại bỏ các bounding box có điểm tin cậy cuối cùng thấp hơn một ngưỡng xác định trước. Chỉ giữ lại các dự đoán mà mô hình đủ tự tin.
- **Non-Maximum Suppression (NMS):** Do nhiều ô lưới có thể cùng phát hiện một biển số xe, dẫn đến nhiều bounding box chồng chéo, NMS được sử dụng để loại bỏ các dự đoán dư thừa. Thuật toán này chọn bounding box có điểm tin cậy cao nhất, sau đó loại bỏ tất cả các box khác có độ chồng lấp (Intersection over Union - IoU) với box được chọn vượt quá một ngưỡng nhất định. Quá trình này lặp lại cho đến khi không còn box nào bị loại bỏ.

Kết quả cuối cùng của quá trình hậu xử lý là một danh sách các bounding box đã được lọc và tinh chỉnh, chỉ rõ vị trí của các biển số xe được phát

hiện trong ảnh đầu vào, sẵn sàng để hiển thị hoặc sử dụng cho các bước tiếp theo.

3.3. KIẾN TRÚC ĐỀ XUẤT CHO NHẬN DIỆN KÝ TỰ

3

Sau khi vị trí biển số được xác định bởi bộ tăng tốc phần cứng trên PL, ảnh chứa biển số sẽ được cắt ra và gửi lên Hệ thống xử lý (PS) để thực hiện bước nhận diện ký tự chi tiết. Để thực hiện nhiệm vụ này hiệu quả trên bộ xử lý ARM của nền tảng Kria KV260, một kiến trúc mạng nơ-ron tích chập (CNN) được thiết kế và triển khai.

Kiến trúc này được xây dựng dựa trên nền tảng YOLOv5, một kiến trúc đã chứng minh được hiệu quả trong nhiều bài toán thị giác máy tính. Tuy nhiên, để tối ưu cho nhiệm vụ cụ thể là nhận diện ký tự trên ảnh biển số đã được chuẩn hóa về kích thước, một số điều chỉnh đã được thực hiện so với các phiên bản YOLOv5 tiêu chuẩn dùng cho phát hiện đối tượng đa lớp:

1. Tối ưu hóa cho Đặc thù Nhận diện Ký tự:

- Kích thước đầu vào:** Mô hình nhận diện ký tự được thiết kế để xử lý ảnh đầu vào có kích thước cố định là 128×128 pixels. Kích thước này được chọn sau khi ảnh biển số gốc đã được cắt ra từ kết quả phát hiện của PL và thay đổi kích thước. Việc chuẩn hóa kích thước đầu vào là bước cần thiết để đảm bảo tính nhất quán cho các lớp tích chập và pooling trong mạng.
- Tập trung vào Feature Map phù hợp:** Do các ký tự thường chiếm một tỷ lệ không gian tương đối lớn trong ảnh biển số đã cắt, kiến trúc mạng tập trung vào việc trích xuất đặc trưng ở các mức độ phân giải phù hợp cho việc phân loại ký tự. Các đầu ra (detection heads) vốn được thiết kế trong YOLOv5 gốc để phát hiện các đối tượng rất nhỏ đã được lược bỏ, giúp giảm độ phức tạp tính toán không cần thiết.

- **Điều chỉnh số lượng bộ lọc:** Số lượng bộ lọc (channels) trong các lớp tích chập được điều chỉnh để cân bằng giữa khả năng biểu diễn đặc trưng và hiệu quả tính toán. Mặc dù vẫn giữ cấu trúc backbone và neck tương tự YOLOv5, việc giảm nhẹ số lượng bộ lọc ở một số tầng giúp mô hình trở nên "nhẹ" hơn, phù hợp hơn với tài nguyên của PS.
- **Sử dụng Attention (C3TR):** Để tăng cường khả năng phân biệt giữa các ký tự có hình dạng tương đồng, module C3TR tích hợp cơ chế self-attention đa đầu được sử dụng ở giai đoạn cuối của phần trích xuất đặc trưng (neck). Cơ chế này cho phép mô hình tập trung vào các vùng điểm ảnh quan trọng nhất trên feature map trước khi đưa ra dự đoán phân loại.

3

2. Cơ chế nhận diện Đơn khung hình (Single-frame Recognition): Kiến trúc nhận diện ký tự được đề xuất hoạt động trên cơ sở xử lý từng ảnh biến số một cách độc lập. Khi một ảnh biến số được đưa vào, mô hình sẽ phân tích và đưa ra kết quả nhận diện các ký tự có trong ảnh đó. Kiến trúc này không sử dụng thông tin từ các khung hình trước đó hoặc các kỹ thuật hậu xử lý phức tạp dựa trên chuỗi thời gian.

Việc lựa chọn cơ chế đơn khung hình này nhằm mục đích:

- **Đảm bảo tốc độ thời gian thực:** Giảm thiểu các bước xử lý phụ thuộc vào nhiều khung hình giúp giảm độ trễ tổng thể, đáp ứng yêu cầu xử lý nhanh của ứng dụng ALPR.
- **Đơn giản hóa luồng xử lý trên PS:** Giúp việc triển khai thuật toán trên bộ xử lý ARM trở nên gọn gàng và dễ quản lý hơn, tập trung tài nguyên cho việc chạy mô hình CNN.

Mặc dù việc không tận dụng thông tin thời gian có thể ảnh hưởng đến khả năng xử lý các trường hợp ảnh bị mờ hoặc nhiễu nặng trong một khung hình đơn lẻ, nhưng kiến trúc này được kỳ vọng mang lại hiệu quả tốt trong điều kiện hoạt động thông thường và đáp ứng yêu cầu về tốc độ.

3. Sơ đồ kiến trúc chi tiết: Kiến trúc cụ thể của mạng CNN nhận diện ký tự, bao gồm trình tự các lớp, loại module và kích thước feature map qua từng tầng, được mô tả chi tiết trong Bảng 3.1. Kiến trúc này bao gồm các khối cơ bản như Convolution (Conv), C3 (CSP Bottleneck với 3 lớp Conv), SPPF (Spatial Pyramid Pooling Fast), Upsample, Concatenate và lớp Detect cuối cùng để phân loại ký tự.

3

Bảng 3.1: Chi tiết kiến trúc mạng CNN nhận diện ký tự

Module index	Connected from	Module	Input size	Output size
0	-	Conv	$3 \times 128 \times 128$	$16 \times 64 \times 64$
1	0	Conv	$16 \times 64 \times 64$	$32 \times 32 \times 32$
2	1	C3	$32 \times 32 \times 32$	$32 \times 32 \times 32$
3	2	Conv	$32 \times 32 \times 32$	$64 \times 16 \times 16$
4	3	C3	$64 \times 16 \times 16$	$64 \times 16 \times 16$
5	4	Conv	$64 \times 16 \times 16$	$128 \times 8 \times 8$
6	5	C3	$128 \times 8 \times 8$	$128 \times 8 \times 8$
7	6	Conv	$128 \times 8 \times 8$	$256 \times 4 \times 4$
8	7	SPP	$256 \times 4 \times 4$	$256 \times 4 \times 4$
9	8	C3TR	$256 \times 4 \times 4$	$256 \times 4 \times 4$
10	9	Conv	$256 \times 4 \times 4$	$128 \times 4 \times 4$
11	10	Upsample	$128 \times 4 \times 4$	$128 \times 8 \times 8$
12	[6,11]	Concatenate	$[128 \times 8 \times 8]$ $[128 \times 8 \times 8]$	$256 \times 8 \times 8$
13	12	C3	$256 \times 8 \times 8$	$128 \times 8 \times 8$
14	13	Conv	$128 \times 8 \times 8$	$64 \times 8 \times 8$
15	14	Upsample	$64 \times 8 \times 8$	$64 \times 16 \times 16$
16	[4,15]	Concatenate	$[64 \times 16 \times 16]$ $[64 \times 16 \times 16]$	$128 \times 16 \times 16$
17	16	C3	$128 \times 16 \times 16$	$64 \times 16 \times 16$
18	17	Detect	$64 \times 16 \times 16$	$82 \times 16 \times 16$

3.4. KIẾN TRÚC XỬ LÍ CONVOLUTION

Phép tích chập (convolution) là nền tảng tính toán chính trong các mạng CNN như YOLO. Lớp convolution 3D nhận đầu vào là tensor đặc trưng $C_{in} \times H_{in} \times W_{in}$ và tạo ra output $C_{out} \times H_{out} \times W_{out}$ bằng cách áp dụng C_{out} bộ lọc (kernels) $C_{in} \times K_h \times K_w$.

Nếu thực hiện tuần tự, quá trình này đòi hỏi nhiều vòng lặp lồng nhau, như mô tả trong Pseudocode 3.1.

3

```

1 // Input: X[Cin][Hin][Win], Kernel
2   W[Cout][Cin][Kh][Kw], Bias B[Cout]
3 // Output: Y[Cout][Hout][Wout]
4 // Kh, Kw: Kernel height, width
5 // Sh, Sw: Stride height, width
6
7 FOR cout from 0 to Cout-1
8   FOR r from 0 to Hout-1
9     FOR c from 0 to Wout-1
10
11    Y[cout][r][c] = B[cout];
12
13    FOR cin from 0 to Cin-1
14      FOR kr from 0 to Kh-1
15        FOR kc from 0 to Kw-1
16
17          in_r = r * Sh + kr;
18          in_c = c * Sw + kc;
19
20          Y[cout][r][c] = Y[cout][r][c] +
21            X[cin][in_r][in_c] *
22            W[cout][cin][kr][kc];
23
24        END FOR // kc
25      END FOR // kr
26    END FOR // cin

```

```

24
25     END FOR // c
26     END FOR // r
27 END FOR // cout
28
29 RETURN Y;

```

3

Listing 3.1: Convolution 3D Tiêu chuẩn (Tuần tự)

Cách tiếp cận tuần tự này quá chậm cho các ứng dụng thời gian thực. Để tăng tốc, chúng ta khai thác tài nguyên song song của FPGA thông qua kỹ thuật "Loop Unrolling". Hai hướng unrolling chính được xem xét:

UNROLLING THEO KÊNH ĐẦU VÀO (INPUT CHANNEL PARALLELISM - INCHA)
 Phương pháp này song song hóa việc tính toán trên các kênh đầu vào (cin) và các phần tử kernel (kr, kc). Mục tiêu là tính tổng tích lũy cho một điểm ảnh output ($Y[cout][r][c]$) trong ít chu kỳ nhất có thể.

Pseudocode 3.2 minh họa ý tưởng này. Phép tính parallel_sum_acc đại diện cho việc thực hiện đồng thời $C_{in} \times K_h \times K_w$ phép nhân và cộng chung lại (thường dùng adder tree).

```

1 // Input: X, W, B
2 // Output: Y
3
4 FOR cout from 0 to Cout-1
5     FOR r from 0 to Hout-1
6         FOR c from 0 to Wout-1
7
8             accumulation = 0;
9             FOR cin from 0 to Cin-1 (in parallel)
10                FOR kr from 0 to Kh-1 (in parallel)
11                    FOR kc from 0 to Kw-1 (in parallel)
12                        in_r = r * Sh + kr;
13                        in_c = c * Sw + kc;
14                        partial_product = X[cin][in_r][in_c] *

```

```

15         accumulation = accumulation +
16             partial_product;
17     END FOR
18
19     END FOR
20
21
22     Y[cout][r][c] = accumulation + B[cout];
23
24 END FOR // cout
25
26 RETURN Y;

```

3

Listing 3.2: Convolution Song song theo Kênh Đầu vào (Incha)

- **Ưu điểm:** Tính nhanh giá trị cho một kênh output tại một vị trí.
- **Nhược điểm:** Cần C_{out} chu kỳ để xong một pixel output. Có thể gây nghẽn pipeline nếu C_{out} (hiện tại) $> C_{in}$ (lớp sau). Tài nguyên phụ thuộc vào $C_{in} \times K_h \times K_w$.

UNROLLING THEO KÊNH ĐẦU RA (OUTPUT CHANNEL PARALLELISM - OUTCHA)

Phương pháp này song song hóa việc tính toán cho nhiều (hoặc tất cả) các kênh đầu ra ($cout$) cùng lúc. Tại mỗi bước lặp của kênh đầu vào (cin) và kernel (kr, kc), giá trị input X được nhân với các trọng số tương ứng của tất cả C_{out} kênh đầu ra.

Pseudocode 3.3 mô tả điều này. Vòng lặp $cout$ được thực hiện song song (biểu thị qua $Y[:]$ và $W[:]$).

```

1 // Input: X, W, B
2 // Output: Y
3
4 FOR r from 0 to Hout-1
5   FOR c from 0 to Wout-1

```

```

6
7     FOR cout from 0 to Cout-1 (in parallel)
8         Y[cout][r][c] = B[cout];
9     END FOR
10
11    FOR cin from 0 to Cin-1
12        FOR kr from 0 to Kh-1
13            FOR kc from 0 to Kw-1
14
15                in_r = r * Sh + kr;
16                in_c = c * Sw + kc;
17                input_val = X[cin][in_r][in_c];
18
19                FOR cout from 0 to Cout-1 (in parallel)
20                    kernel_val = W[cout][cin][kr][kc];
21                    Y[cout][r][c] = Y[cout][r][c] + input_val *
22                        kernel_val;
23                END FOR
24
25            END FOR // kc
26        END FOR // kr
27        END FOR // cin
28
29    END FOR // c
30
31 RETURN Y;

```

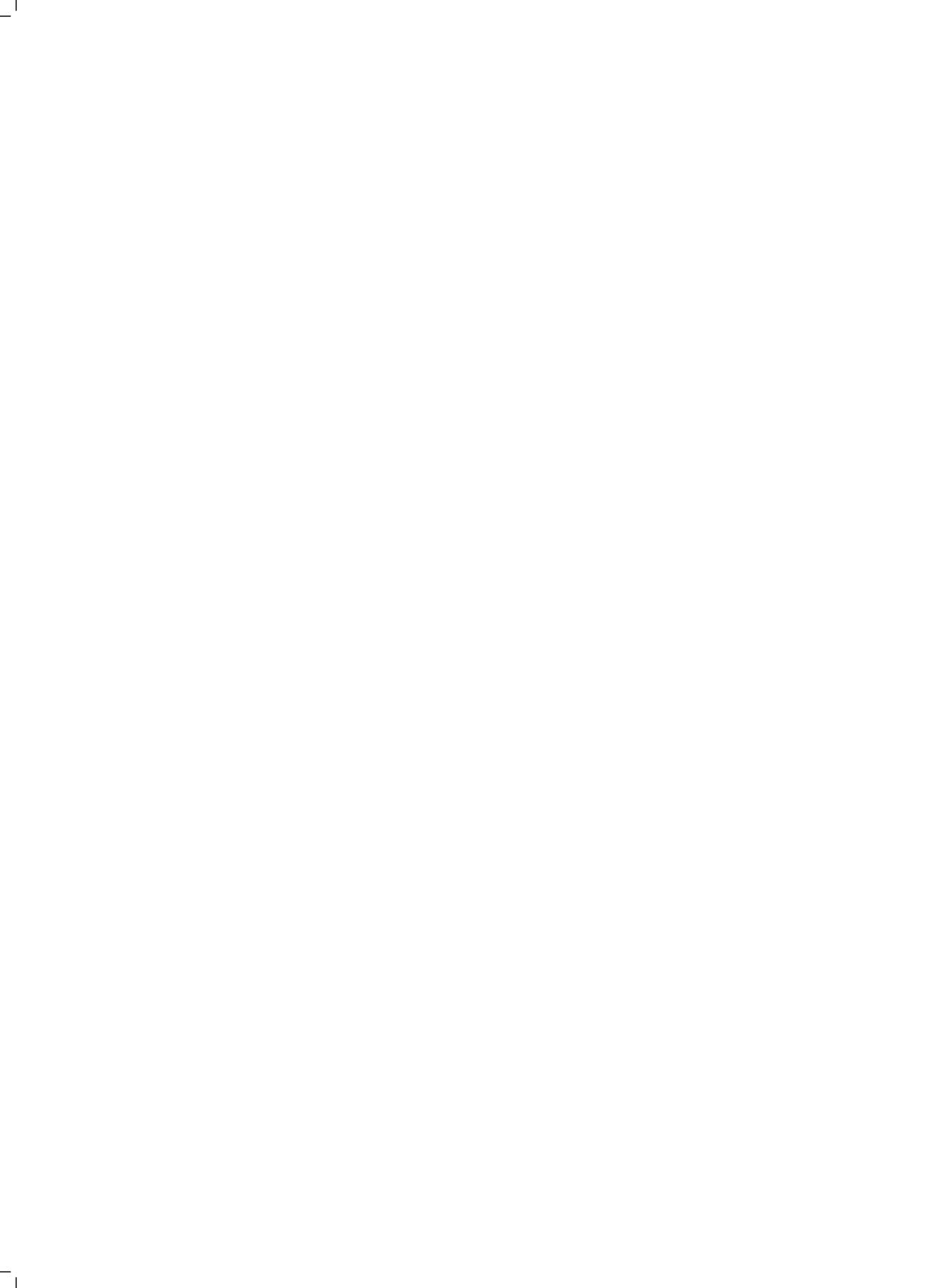
Listing 3.3: Convolution Song song theo Kênh Đầu ra (Outcha)

- **Ưu điểm:** Tạo ra tất cả C_{out} kênh của một pixel output sau khoảng $C_{in} \times K_h \times K_w$ chu kỳ. Hiệu quả hơn Incha khi $C_{in} > C_{out}$.
- **Nhược điểm:** Tài nguyên phụ thuộc vào $C_{out} \times K_h \times K_w$. Việc broadcast dữ liệu input có thể phức tạp.

LỰA CHỌN HƯỚNG UNROLLING CHO DỰ ÁN

Việc lựa chọn chiến lược song song hóa (unrolling) cho các lớp convolution là một quyết định thiết kế quan trọng, ảnh hưởng trực tiếp đến hiệu năng, tài nguyên sử dụng và độ phức tạp của bộ tăng tốc phần cứng. Mặc dù cả hai hướng unrolling chính, Incha (song song hóa theo kênh đầu vào) và Outcha (song song hóa theo kênh đầu ra), đều có những ưu điểm riêng, nhưng dựa trên đặc điểm của các mạng YOLO hiện đại và các ràng buộc phần cứng của nền tảng FPGA mục tiêu (Kria KV260), kiến trúc **Incha-Single** được lựa chọn làm phương pháp song song hóa chính cho dự án này. Lý do chính cho quyết định này là vì kiến trúc Incha-Single, yêu cầu một số lượng bộ nhân song song tỉ lệ thuận với $C_{in} \times K_h \times K_w$. Trong khi đó, kiến trúc Outcha-Single yêu cầu $C_{out} \times K_h \times K_w$ bộ nhân. Các mạng YOLO, đặc biệt ở các lớp sâu hơn, thường có số lượng kênh đầu vào (C_{in}) và đầu ra (C_{out}) lớn. Tuy nhiên, việc hiện thực C_{out} "slice" xử lý song song hoàn chỉnh như trong Outcha có thể dẫn đến yêu cầu về số lượng bộ nhân (và do đó là DSP slices) cực kỳ lớn, vượt quá khả năng của Kria KV260, đặc biệt khi C_{out} lớn. Kiến trúc Incha, mặc dù vẫn cần nhiều bộ nhân khi C_{in} lớn, thường linh hoạt hơn trong việc chia sẻ tài nguyên tính toán qua thời gian (xử lý tuần tự qua các kênh C_{out}). Việc chỉ sử dụng mức độ song song hóa 'single' giúp kiểm soát chặt chẽ hơn việc sử dụng DSP.

Do đó, mặc dù một kiến trúc lai có thể tối ưu hơn trong một số trường hợp lý thuyết, nhưng để đảm bảo tính khả thi về tài nguyên, đơn giản hóa thiết kế và định tuyến trên Kria KV260, đồng thời vẫn duy trì được pipeline hiệu quả, hướng unrolling **Incha-Single** được ưu tiên lựa chọn làm kiến trúc PE cơ sở cho tất cả các lớp convolution trong bộ tăng tốc YOLO của dự án này.

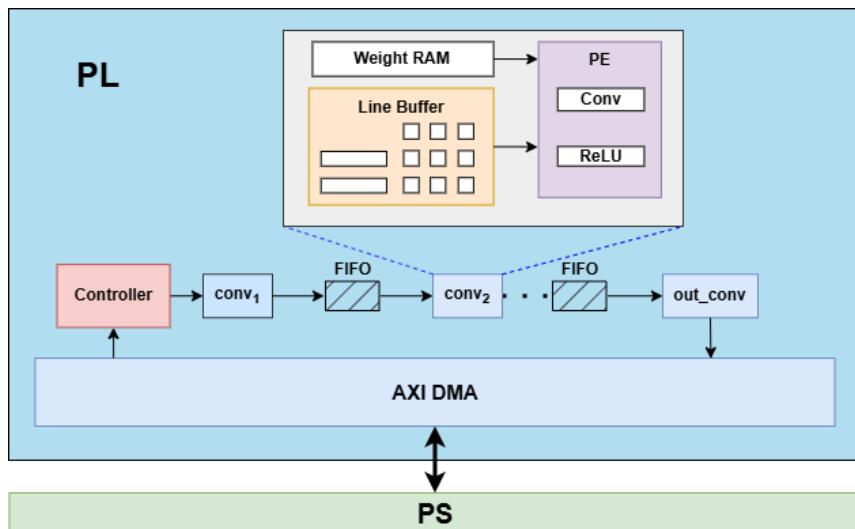


4

KIẾN TRÚC PHẦN CỨNG ĐỀ XUẤT

4.1. KIẾN TRÚC TỔNG QUAN

Bộ tăng tốc được thiết kế dưới dạng một pipeline phần cứng chuyên dụng, tối ưu hóa cho việc xử lý luồng dữ liệu ảnh và thực hiện các phép tính của mạng YOLO INT8. Hình 4.1 mô tả sơ đồ khái quát và luồng dữ liệu chính giữa PS và PL cũng như bên trong PL.



Hình 4.1: Kiến trúc tổng quan và luồng dữ liệu của bộ tăng tốc YOLO trên PL.

Luồng dữ liệu tổng thể:

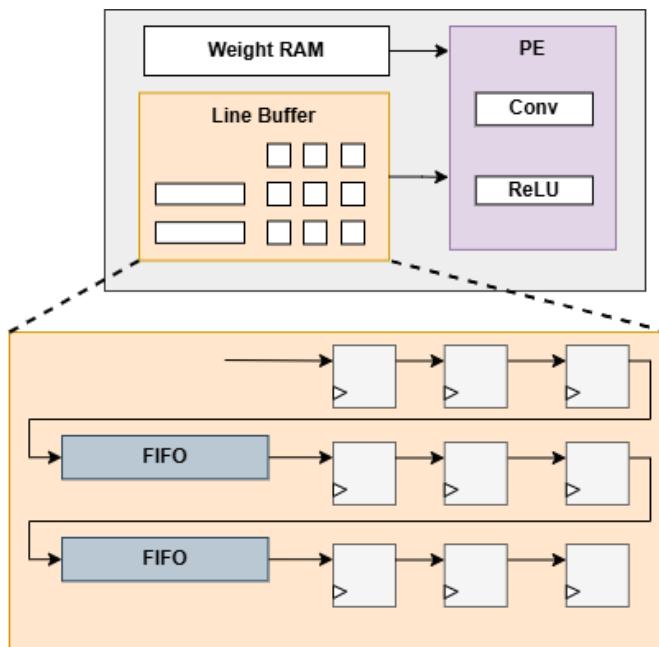
- PS → PL (Input):** Dữ liệu ảnh đầu vào (đã qua tiền xử lý và lượng tử hóa INT8 bởi PS nếu cần) được PS ghi vào bộ nhớ DDR. PS sau đó cấu hình và khởi chạy khối AXI DMA để truyền dữ liệu ảnh từ DDR vào **Input FIFO** trên PL.
- Xử lý trong PL:** Dữ liệu ảnh từ AXI-DMA đi vào Controller và được đưa qua một chuỗi các khối xử lý convolution ('conv_1' đến 'out_conv') được kết nối tuần tự thông qua các **FIFO trung gian**. Mỗi khối convolution thực hiện các phép tính của một tầng mạng YOLO. Các tham số (trọng số, bias, tham số lượng tử hóa) và các bản đồ đặc trưng trung gian (feature maps) được lưu trữ trong **BRAM** và Register trên PL.
- PL → PS (Output):** Sau khi PL hoàn thành xử lý (báo hiệu cho PS), PS cấu hình AXI DMA để đọc tensor kết quả từ lớp out_conv về bộ nhớ DDR của PS.
- Hậu xử lý (PS):** PS nhận tensor kết quả, thực hiện giải mã bounding box và áp dụng Non-Maximum Suppression (NMS) để lọc ra các dự

đoán cuối cùng về vị trí biển số xe.

Kiến trúc pipeline sâu và chiến lược sử dụng BRAM trên chip nhằm mục đích tối đa hóa thông lượng và giảm thiểu độ trễ truy cập bộ nhớ ngoài.

4.2. KIẾN TRÚC CHI TIẾT

4.2.1. LINE BUFFER (BỘ ĐỆM DÒNG)



4

Hình 4.2: Sơ đồ khối kiến trúc Line Buffer [10]

- **Chức năng:** Chuyển đổi luồng dữ liệu pixel INT8 (đến tuần tự theo thứ tự quét raster) thành một cửa sổ dữ liệu không gian 2D cần thiết cho phép tích chập. Nó nhận pixel từ FIFO đầu vào của lớp.
- **Cấu trúc hoạt động:** (Tham khảo Hình 4.2) Sử dụng kết hợp các hàng bộ đếm (line buffers, thường dùng BRAM/LUTRAM để lưu trữ $K_h - 1$ hàng pixel trước đó) và các thanh ghi dịch (shift registers) để lưu trữ các pixel trong cửa sổ $K_h \times K_w$ hiện tại. Khi một pixel mới

đến, dữ liệu được dịch chuyển và cửa sổ $K_h \times K_w$ gồm các pixel (mỗi pixel có C_{in} kênh) được cập nhật và cung cấp cho PE.

4

- **Xử lý biên (Padding):** Logic bên trong Line Buffer và logic điều khiển truy cập có thể đảm nhiệm việc đệm các giá trị zero (hoặc giá trị phù hợp khác) khi cửa sổ trượt xử lý các pixel ở rìa của feature map đầu vào, tương ứng với tham số padding của lớp convolution.

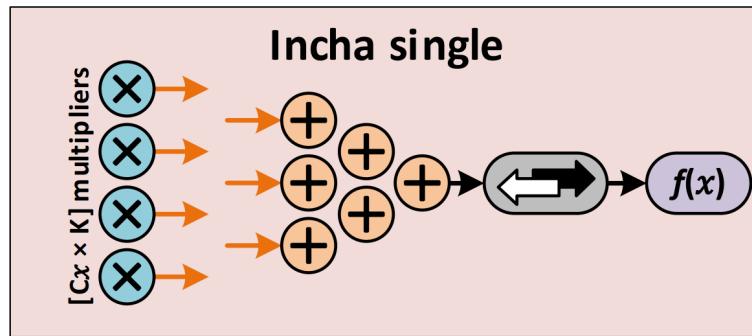
4.2.2. PROCESSING ELEMENT (PE - KHỐI XỬ LÝ)

Dựa trên các phương pháp xử lý song song đã thảo luận (Mục 3.4), khối tính toán cốt lõi cho mỗi lớp convolution, thường được gọi là Processing Element (PE), cần được thiết kế. PE chịu trách nhiệm nhận các cửa sổ dữ liệu đầu vào (input feature map window) từ Line Buffer, thực hiện phép nhân tích lũy (MACC) với các trọng số kernel tương ứng, và áp dụng hàm kích hoạt.

Trong khuôn khổ của dự án này, do đặc thù của mô hình YOLO tùy chỉnh và mục tiêu cân bằng giữa tài nguyên và hiệu năng trên Kria KV260, chúng tôi tập trung vào việc hiện thực các PE dựa trên chiến lược unrolling. Kiến trúc PE Incha chính dựa trên unrolling được xem xét:

PE THEO KIẾN TRÚC INCHA

Kiến trúc này trực tiếp hiện thực hóa phương pháp song song hóa theo kênh đầu vào (Incha - Mục 3.4). Mỗi PE Incha được thiết kế để tính toán giá trị output cho **một kênh đầu ra (c_{out}) tại một thời điểm**.



Hình 4.3: Sơ đồ khối kiến trúc PE Incha

4

Như minh họa trong Hình 4.3, cấu trúc chính của PE Incha bao gồm:

- **Mảng bộ nhân (Multiplier Array):** Một mảng gồm $C_{in} \times K_h \times K_w$ bộ nhân 8-bit (INT8 x INT8). Mỗi bộ nhân nhận một giá trị activation từ cửa sổ input và một giá trị trọng số kernel tương ứng (đọc từ BRAM chứa trọng số).
- **Cây cộng (Adder Tree):** Các kết quả 16-bit từ mảng bộ nhân được cộng dồn lại bằng một cây cộng lớn để tạo ra một giá trị tích lũy duy nhất.
- **Cộng Bias và Requantization:** Kết quả tích lũy được nhân với hệ số MACC M dạng fixed-point. Sau đó, giá trị bias được cộng vào kết quả tích lũy.
- **Khối Kích hoạt (Activation Unit):** Giá trị sau tính toán được đưa qua khối xử lý hàm kích hoạt.

PE Incha cần C_{out} chu kỳ để tính toán đầy đủ các kênh output cho một vị trí pixel (r, c) . Kiến trúc này phù hợp cho các lớp có C_{in} không quá lớn và khi C_{out} không lớn hơn đáng kể so với C_{in} của lớp tiếp theo.

4.2.3. KHỐI FIFO (FIRST-IN, FIRST-OUT BUFFER)

Các khối FIFO (First-In, First-Out) đóng vai trò then chốt trong việc đệm và đồng bộ hóa luồng dữ liệu giữa các tầng xử lý pipeline trong bộ tăng

tốc YOLO. Chúng hoạt động như những bộ đếm đòn hồi, cho phép các tầng kết nối với nhau có thể hoạt động với tốc độ xử lý hoặc thời gian sẵn sàng khác nhau một chút mà không làm đình trệ toàn bộ hệ thống. Trong thiết kế này, một module FIFO được sử dụng cho các kết nối điểm-điểm giữa các tầng.

4

- **Chức năng:** Lưu trữ tạm thời dữ liệu đầu ra (ví dụ: pixel feature map INT8) từ một tầng xử lý nguồn (producer) và cung cấp tuần tự cho tầng xử lý đích (consumer) theo đúng thứ tự dữ liệu được ghi vào.

- **Cấu trúc và Hiện thực:**

- *Bộ nhớ Lưu trữ (Storage):* Dữ liệu được lưu trong một khối Block RAM. Kích thước của bộ nhớ này được xác định bởi tham số DEPTH và độ rộng dữ liệu là DATA_WIDTH. Việc sử dụng BRAM giúp lưu trữ hiệu quả lượng lớn dữ liệu trên chip.
- *Con trỏ Đọc/Ghi (Read/Write Pointers):* Hai bộ đếm riêng biệt (wr_cnt và rd_cnt) được sử dụng để theo dõi vị trí ghi và đọc tiếp theo trong bộ nhớ BRAM. Mỗi bộ đếm có độ rộng là ADDR_WIDTH + 1 bits, với $\text{ADDR_WIDTH} = \lceil \log_2(\text{DEPTH}) \rceil$. Bit cao nhất (MSB) của mỗi bộ đếm đóng vai trò quan trọng trong việc xác định trạng thái đầy/rỗng khi các con trỏ.
- *Logic Điều khiển và Trạng thái:* Logic tổ hợp được sử dụng để tạo ra các tín hiệu trạng thái quan trọng dựa trên giá trị của các con trỏ đọc/ghi:

- **Hoạt động:**

- *Ghi (Write):* Khi tín hiệu wr_en được khẳng định và FIFO không đầy (full=0), dữ liệu trên wr_data sẽ được ghi vào vị trí ram [wr_addr] (với wr_addr là các bit thấp của wr_cnt) tại sườn lên clock tiếp theo, đồng thời bộ đếm wr_cnt được tăng lên.

- *Đọc (Read)*: Khi tín hiệu rd_en được khảng định và FIFO không rỗng (`empty=0`), dữ liệu tại vị trí ram [`rd_addr`] (với `rd_addr` là các bit thấp của `rd_cnt`) được xuất ra trên `rd_data`. Tại sườn lên clock tiếp theo, bộ đếm `rd_cnt` được tăng lên.

- **Tác dụng trong Pipeline:**

- *Decoupling*: Cho phép tầng hoạt động với các độ trễ không đồng đều. Producer có thể ghi dữ liệu vào FIFO khi sẵn sàng, và consumer có thể đọc khi cần, miễn là FIFO không đầy hoặc rỗng.
- *Đồng bộ hóa và Đảm bảo Thứ tự*: Duy trì thứ tự dữ liệu nghiêm ngặt theo nguyên tắc "vào trước, ra trước".
- *Điều khiển luồng (Flow Control)*: Các tín hiệu `empty`, `full`, `almost_full` cung cấp cơ chế phản hồi cần thiết để các tầng producer và consumer có thể tạm dừng (stall) hoạt động ghi/đọc của mình, ngăn chặn tràn hoặc đọc thiếu dữ liệu.

4

4.2.4. HỆ THỐNG BỘ NHỚ BRAM

Như đã nhấn mạnh, kiến trúc này dựa chủ yếu vào BRAM trên chip.

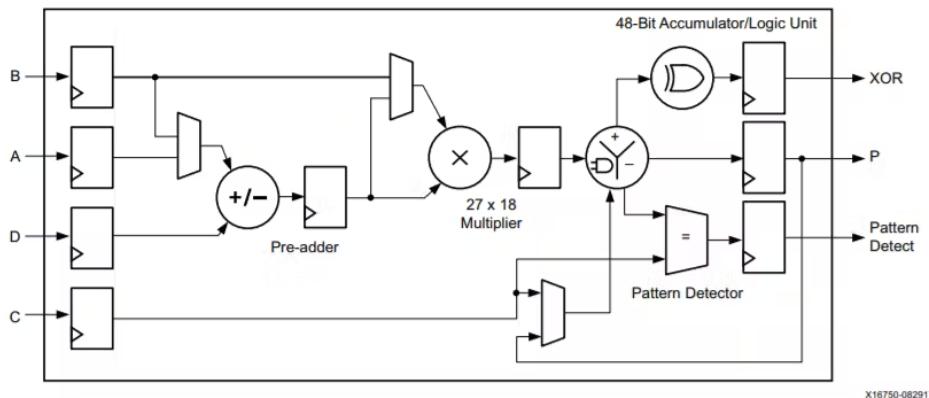
- **Lưu trữ Trọng số/Bias/Tham số Quant**: Mỗi PE/lớp convolution có các khối BRAM riêng để lưu trữ trọng số kernel, bias, và các hệ số MACC cần thiết. Việc này đảm bảo truy cập nhanh và song song.
- **Lưu trữ Feature Map Trung gian**: Đây là điểm đặc biệt và thách thức. *Tất cả* các feature map đầu ra INT8 của mỗi lớp (ngoại trừ lớp cuối) được ghi vào các BRAM đóng vai trò như bộ đệm trung gian (thường được tổ chức để hỗ trợ FIFO hoặc truy cập theo địa chỉ bởi Line Buffer của lớp sau). Cần có cơ chế quản lý địa chỉ hiệu quả để đọc/ghi đúng các vùng dữ liệu.
- **Thách thức**: Tổng dung lượng BRAM yêu cầu cho *tất cả* trọng số, bias, tham số và feature maps trung gian phải nhỏ hơn dung lượng

BRAM có sẵn trên Kria KV260. Cần phân tích kỹ lưỡng yêu cầu bộ nhớ của mô hình YOLO tùy chỉnh.

4.2.5. TỐI ƯU HÓA PHÉP NHÂN TRÊN KHỐI DSP32E2

Để tối đa hóa hiệu năng tính toán và tận dụng hiệu quả tài nguyên phần cứng chuyên dụng trên FPGA, việc tối ưu hóa các phép toán cơ bản như phép nhân là rất quan trọng. Các FPGA hiện đại của Xilinx, bao gồm cả dòng Zynq UltraScale+ trên Kria KV260, được trang bị các khối xử lý tín hiệu số (Digital Signal Processing - DSP) mạnh mẽ, thường là các khối DSP48E2.

Khối DSP48E2, cung cấp nhiều chức năng toán học phức tạp, trong đó nổi bật là khả năng thực hiện phép nhân 27×18 bit và tích lũy kết quả 48-bit. Đặc biệt, khói này có một bộ cộng trước (Pre-adder) 27-bit ở đầu vào cổng A (30-bit) và cổng D (27-bit), cho phép thực hiện phép tính $(A + D) \times B$ trong một chu kỳ DSP. Hình 4.4 minh họa kiến trúc tổng quan này.



Hình 4.4: Kiến trúc tổng quan khói DSP48E2.

Xilinx đã công bố một kỹ thuật trong White Paper WP486 để tận dụng kiến trúc này nhằm thực hiện **hai phép nhân 8-bit có chung một thừa số ('a*c' và 'b*c') trong một chu kỳ xử lý. Kỹ**

thuật này đặc biệt hữu ích cho các kiến trúc PE unroll theo kênh đầu ra (Outcha), nơi một giá trị activation đầu vào cần được nhân với hai trọng số kernel khác nhau.

Ý tưởng chính của kỹ thuật này là ánh xạ hai phép nhân 8-bit vào một phép nhân 27×18 bit lớn hơn. Giả sử chúng ta cần tính $AC = a \times c$ và $BC = b \times c$, với a, b, c là các số nguyên 8-bit có dấu (INT8). Kỹ thuật thực hiện như sau (minh họa trong Hình 4.5):

1. Tạo toán hạng 27-bit (cho cổng A+D):

4

- Dịch trái giá trị a 18 bit: $a_{shifted} = a \ll 18$.
- Mở rộng dấu (sign-extend) giá trị b thành 27 bit: $b_{extended}$.
- Cộng hai giá trị trên: $Operand_{AD} = a_{shifted} + b_{extended}$. Giá trị này sẽ được đưa vào bộ Pre-adder của DSP thông qua cổng A và D.

2. Tạo toán hạng 18-bit (cho cổng B):

- Mở rộng dấu giá trị c thành 18 bit: $Operand_B = c_{extended}$.

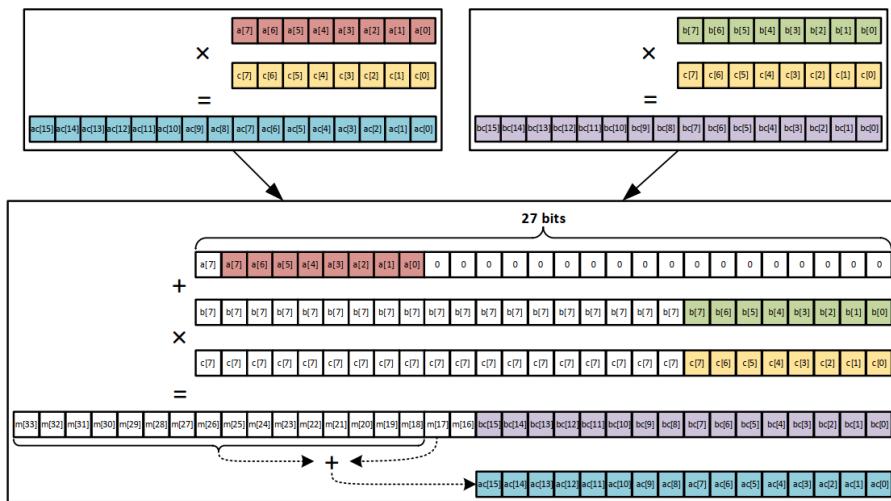
3. Thực hiện phép nhân DSP:

Khối DSP thực hiện phép nhân $P = Operand_{AD} \times Operand_B = (a \ll 18 + b_{ext}) \times c_{ext}$.

4. Tách kết quả:

Do tính chất phân phối của phép nhân và cách các toán hạng được cấu trúc, kết quả P sẽ chứa cả hai tích AC và BC :

- AC (16-bit) có thể được trích xuất từ các bit cao của P (ví dụ: $P[33 : 18]$ với điều chỉnh dựa trên bit dấu, như trong Hình 4.2).
- BC (16-bit) có thể được trích xuất từ các bit thấp của P (ví dụ: $P[15 : 0]$).



Hình 4.5: Thực hiện hai phép nhân 8-bit ($a*c$, $b*c$) trên một khối DSP.

Việc áp dụng kỹ thuật này cho phép tăng gấp đôi số lượng phép nhân 8-bit có thể thực hiện trên mỗi khối DSP, giúp tiết kiệm đáng kể tài nguyên DSP quý giá trên FPGA hoặc tăng gấp đôi thông lượng tính toán với cùng số lượng DSP.

Trong các thiết kế PE, kỹ thuật nhân kép này sẽ được xem xét áp dụng khi cần thực hiện các phép nhân song song có chung một toán hạng đầu vào (activation) với nhiều trọng số khác nhau.

5

HIỆN THỰC HỆ THỐNG

Chương này đi sâu vào chi tiết hiện thực của hệ thống tăng tốc nhận diện biển số xe trên nền tảng Kria KV260. Dựa trên kiến trúc phần cứng đã trình bày, chương này sẽ mô tả cách tích hợp các thành phần trong môi trường Vivado, cơ chế giao tiếp và luồng dữ liệu giữa PS và PL, việc quản lý bộ nhớ và điều khiển khối IP tăng tốc thông qua framework PYNQ.

5.1. HIỆN THỰC LƯỢNG TỬ HÓA

Việc triển khai các mô hình học sâu phức tạp như YOLO để phát hiện đối tượng theo thời gian thực trên các nền tảng FPGA có tài nguyên hạn chế, như Kria KV260, đòi hỏi sự tối ưu hóa cẩn thận về mặt tính toán và bộ nhớ. Lượng tử hóa mô hình, tức là giảm độ chính xác biểu diễn số của trọng số và/hoặc activations, là một kỹ thuật then chốt để đạt được mục tiêu này. Các nghiên cứu gần đây đã khám phá nhiều chiến lược lượng tử hóa khác nhau cho YOLO trên FPGA, mỗi chiến lược có ưu và nhược điểm riêng.

Xét đến mục tiêu của khóa luận là nhận diện biển số xe với độ chính

xác cao trong thời gian thực trên Kria KV260, phương pháp **lượng tử hóa 8-bit (INT8)** được lựa chọn. INT8 cung cấp sự cân bằng tốt nhất giữa việc bảo toàn độ chính xác (quan trọng cho việc nhận diện chính xác biến số) và việc giảm thiểu tài nguyên/tăng tốc độ xử lý so với FP32. Mặc dù đòi hỏi nhiều tài nguyên hơn 1-bit hay 4-bit, INT8 vẫn khả thi trên các nền tảng SoC hiện đại như Zynq UltraScale+ và được hỗ trợ tốt bởi các công cụ phát triển. Cụ thể, kỹ thuật **Lượng tử hóa Tĩnh sau Huấn luyện (Post-Training Static Quantization - PTQ)** được sử dụng để chuyển đổi mô hình YOLO tùy chỉnh (anchor-free) đã huấn luyện sang INT8.

5.1.1. QUY TRÌNH LƯỢNG TỬ HÓA INT8 PTQ

5

Quy trình lượng tử hóa mô hình YOLO tùy chỉnh từ FP32 sang INT8 bằng PTQ với thư viện PyTorch được thực hiện theo các bước sau:

1. **Nạp mô hình gốc (FP32):** Tải mô hình ‘YoloNoAnchor’ đã huấn luyện.
2. **Hợp nhất Modules (Fuse Modules):** Kết hợp các lớp Conv-BatchNorm-ReLU thành các đơn vị tối ưu hơn cho lượng tử hóa.
3. **Định nghĩa Cấu hình Lượng tử hóa (QConfig):** Thu thập dải giá trị, áp dụng sơ đồ đối xứng trên mỗi tensor (per_tensor_symmetric), với kiểu dữ liệu torch.qint8 cho trọng số và torch.quint8 cho activations.
4. **Chuẩn bị (Prepare):** Chèn các observers vào mô hình.
5. **Calibration:** Chạy mô hình với dữ liệu calibration để observers thu thập thống kê dải giá trị của activations.
6. **Chuyển đổi (Convert):** Tính toán tham số S, Z , thay thế các lớp bằng phiên bản lượng tử hóa và lượng tử hóa trọng số sang INT8.
7. **Lưu mô hình INT8:** Lưu state dictionary của mô hình đã lượng tử hóa.

5.1.2. TRÍCH XUẤT THAM SỐ INT8 CHO PHẦN CỨNG

Để hiện thực bộ tăng tốc trên PL, các tham số của mô hình INT8 cần được trích xuất và định dạng phù hợp. Script `check_weight.py` thực hiện nhiệm vụ này bằng cách:

1. Nạp mô hình INT8 từ file `full_calib.pth`.
2. Lặp qua các lớp convolution (từ `conv1` đến `out_conv`).
3. **Trích xuất Trọng số (Kernel Weights - INT8):** Lấy giá trị nguyên 8-bit có dấu (`qint8`) của trọng số bằng `.int_repr()`.
4. **Trích xuất và Xử lý Sơ bộ Bias:** Lấy giá trị bias và chia cho scaling factor của output lớp đó (S_{output}). Giá trị này cần được lượng tử hóa thêm với scale phù hợp ($S_{bias} = S_{input} \times S_{weight}$) và được chuyển đổi sang dạng fixed-point trước khi sử dụng trong phần cứng.
5. **Tính toán Hệ số MACC (M):** Tính toán hệ số requantization $M = (S_{input} \times S_{weight}) / S_{output}$ cho mỗi lớp, sử dụng scaling factor của input (từ lớp trước hoặc input gốc), weight và output của lớp hiện tại. Giá trị M này sẽ được chuyển đổi sang dạng fixed-point trong thiết kế phần cứng.
6. **Trích xuất Layer Scale (Lớp Output):** Lưu lại scaling factor S_{output} của lớp convolution cuối cùng `out_conv` để có thể de-quantize kết quả.

Script cũng thực hiện kiểm tra dải giá trị min/max của các tham số trích xuất được (kernel, bias đã xử lý sơ bộ, hệ số MACC, layer scale) để hỗ trợ việc xác định độ rộng bit cần thiết khi thiết kế các thanh ghi và bộ nhớ trong phần cứng.

5.1.3. ĐÁNH GIÁ ĐỘ CHÍNH XÁC SAU LƯỢNG TỬ HÓA INT8

Hiệu quả của quá trình lượng tử hóa INT8 được đánh giá bằng cách so sánh độ chính xác của mô hình trước (FP32), sau (INT8 PTQ) và trên

Board (INT8) khi lượng tử hóa. Việc đánh giá được thực hiện trên tập dữ liệu kiểm tra sử dụng chỉ số Mean IoU.

Kết quả được tổng hợp trong Bảng 5.1.

5

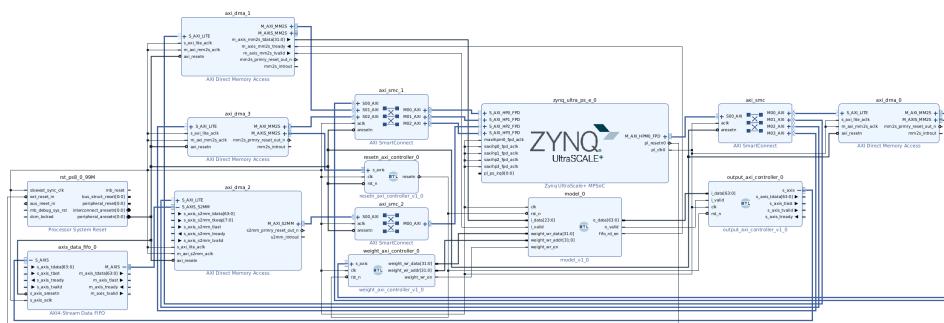
Bảng 5.1: So sánh độ chính xác trước và sau lượng tử hóa INT8 PTQ.

Phiên bản mô hình	Mean IoU (%)
YOLO Custom FP32 (Gốc)	78.86%
YOLO Custom INT8 (PTQ)	73.12%
YOLO Custom INT8 on Board	72.37%
Độ sụt giảm	6.49 %

Kết quả trong Bảng 5.1 cho thấy độ sụt giảm độ chính xác sau khi lượng tử hóa INT8 trên Board là 6.49%, một mức chấp nhận được. Điều này xác nhận INT8 là lựa chọn phù hợp cho mô hình YOLO tùy chỉnh trong ứng dụng nhận diện biển số xe, cân bằng tốt giữa độ chính xác và yêu cầu triển khai trên phần cứng FPGA. Các tham số INT8 sẽ được sử dụng để xây dựng bộ tăng tốc phần cứng chi tiết trong các phần tiếp theo.

5.2. TÍCH HỢP HỆ THỐNG TRONG VIVADO

Quá trình hiện thực bắt đầu bằng việc tích hợp các khôi IP cần thiết và khôi IP YOLO tùy chỉnh vào một thiết kế khôi (Block Design) trong môi trường Vivado. Hình 5.1 minh họa sơ đồ kết nối mức cao của hệ thống.



Hình 5.1: Sơ đồ khái niệm về cách tích hợp hệ thống tăng tốc YOLO trong Vivado.

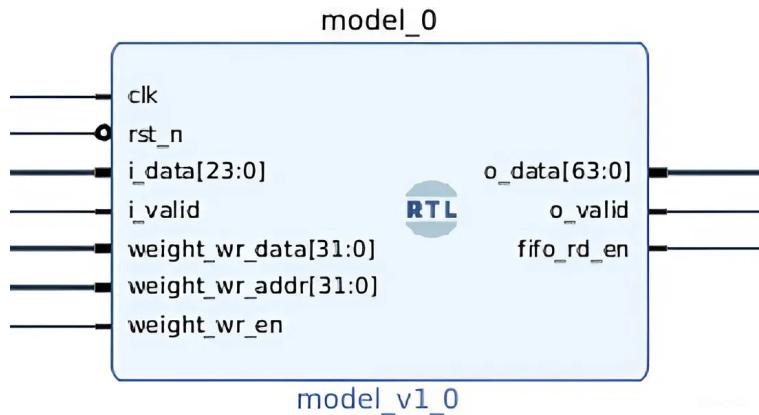
Các thành phần chính và kết nối quan trọng bao gồm:

- **Zynq UltraScale+ MPSoC (zynq_ultra_ps_e_0):** Khối PS được cấu hình để giao tiếp với PL qua các cổng AXI tốc độ cao (HP) và cổng đa dụng (GP). Các cổng HP (M_AXI_HPMx_FPD) được sử dụng cho các giao dịch DMA với bộ nhớ DDR. Cổng GP (M_AXI_GPx) dùng để truy cập các thanh ghi điều khiển AXI-Lite.
- **AXI Direct Memory Access (DMA):** Bốn khối IP AXI DMA (axi_dma_0 đến axi_dma_3) được sử dụng để quản lý các luồng dữ liệu và điều khiển khác nhau giữa PS và PL:
 - axi_dma_0: Truyền trọng số (weights) từ PS vào PL.
 - axi_dma_1: Truyền dữ liệu ảnh đầu vào từ PS vào PL.
 - axi_dma_2: Truyền tensor kết quả từ PL về PS.
 - axi_dma_3: Truyền tín hiệu điều khiển từ PS vào PL.

Mỗi DMA có cổng AXI4 Memory Map kết nối tới PS (qua SmartConnect) và cổng AXI4-Stream kết nối tới các thành phần tương ứng trong PL.

- **AXI SmartConnect (axi_smc_1, axi_smc_2):** Đóng vai trò trung gian kết nối các AXI Master (PS, DMA) với các AXI Slave (các khối điều khiển tùy chỉnh, DDR). Chúng xử lý việc định tuyến, chuyển đổi độ rộng bus và giao thức khi cần thiết.

- Khối IP YOLO tùy chỉnh (model_v1_0):** Đây là bộ xử lý trung tâm, chứa pipeline các lớp convolution INT8. Nó nhận dữ liệu ảnh qua FIFO đầu vào và trọng số qua bộ điều khiển trọng số, sau đó xuất kết quả ra bộ điều khiển output.



Hình 5.2: Sơ đồ khối IP Model trong Vivado.

- Các khối giao diện và điều khiển tùy chỉnh:**
 - axis_data_fifo_0: FIFO AXI-Stream đệm dữ liệu ảnh từ axi_dma_1 trước khi vào model_v1_0.
 - output_axi_controller_0: Nhận dữ liệu tensor đầu ra từ model_v1_0 và cung cấp giao diện AXI-Stream cho axi_dma_2 đọc về PS.
 - weight_axi_controller_v1_0: Nhận dữ liệu trọng số từ axi_dma_0 (qua axi_smc_2) và điều khiển việc ghi trọng số vào các BRAM bên trong hoặc liên kết với model_v1_0 qua các cổng weight_wr_*.
 - resetn_axi_controller_0: Nhận tín hiệu điều khiển từ axi_dma_3 (qua axi_smc_1) và tạo tín hiệu reset (rst_n) cho các khối logic cần thiết trong PL.
- Processor System Reset (rst_ps8_0_99M):** Cung cấp tín hiệu reset hệ thống.

- **Clocking Wizard IP:** Tạo và phân phối các tín hiệu clock cần thiết cho hệ thống PL từ nguồn clock đầu vào.

Sau khi kết nối các khối IP, thiết kế được tổng hợp (synthesis), hiện thực (implementation), và sinh ra file bitstream (.bit) để cấu hình cho PL của Kria KV260.

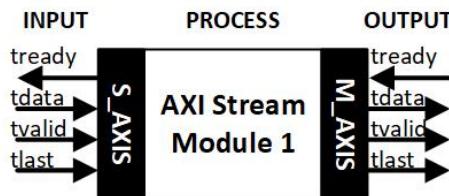
5.3. GIAO TIẾP PS-PL QUA AXI DMA VỚI PYNQ

5.3.1. GIỚI THIỆU AXI DMA VÀ AXI STREAM IP CORE.

Việc truyền dữ liệu ảnh đầu vào (kích thước lớn) và tensor kết quả giữa PS và PL được thực hiện hiệu quả bằng AXI DMA. AXI DMA (Direct Memory access) IP core là một IP core do Xilinx cung cấp với một tốc độ truy xuất cao, đồng thời cũng có những giao thức kết nối giống với AXI4 hay AXI4-Stream.

Trong luận văn này, chúng em sẽ sử dụng DMA với protocol là AXI Stream protocol. Một module AXI stream (AXIS) có 2 phần, một là giao tiếp ports cho master (M_AXIS) và một cho slave (S_AXIS), mỗi giao tiếp này có thể được điều khiển bằng 4 tín hiệu cơ bản bao gồm **tready**, **tdata**, **tvalid** và **tlast**.

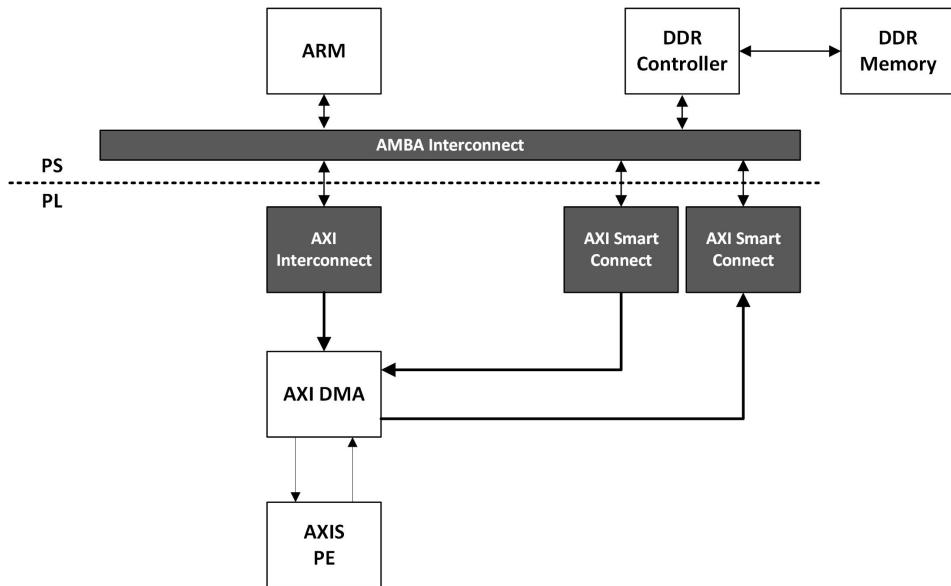
- **tready:** dùng để thông báo cho AXIS module đã sẵn sàng để nhận dữ liệu
- **tdata:** chứa dữ liệu dùng để truyền (hỗ trợ 32, 64, 128, 256, 512, và 1024 bits độ rộng dữ liệu)
- **tvalid:** chỉ ra rằng dữ liệu trong tdata cần được phải xử lý
- **tlast:** dùng để thông báo sau khi đã truyền xong một gói dữ liệu (packet)



Hình 5.3: AXIS module.

Để kết nối với AXIS module tới PS, chúng ta sẽ sử dụng IP của Xilinx tên là AXI DMA. Module này cho phép truyền dữ liệu từ memory-mapped data (from DDR memory) tới các kênh stream data và ngược lại, cơ chế giao tiếp có thể được mô tả bằng hình sau:

5

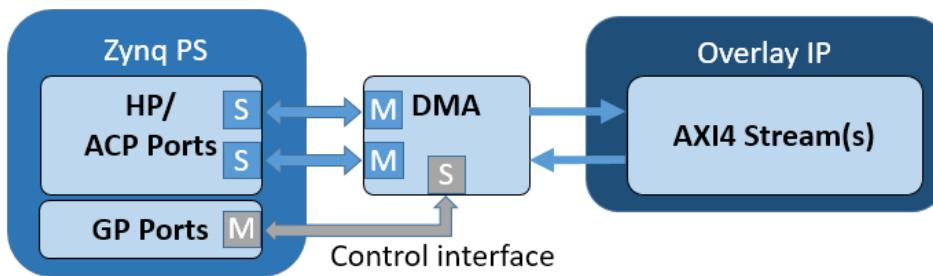


Hình 5.4: AXI DMA và AXI Interconnect trong giao tiếp giữa PS và PL.

Nhóm sẽ sử dụng tối đa 4 cổng HP* của IP Zynq để thiết lập 4 kênh AXI Stream dành cho 4 chức năng bao gồm tín hiệu điều khiển reset, kênh nạp weight cho hệ thống, kênh truyền dữ liệu input và cuối cùng là kênh truyền dữ liệu output.

5.3.2. KHẢ NĂNG HỖ TRỢ CỦA PYNQ CHO GIAO THỨC AXI DMA

PYNQ hỗ trợ DMA IP với một class có tên là PYNQ DMA class, hiện tại class này chỉ hỗ trợ simple mode, đây cũng là mode là chúng em sử dụng trong phạm vi luận văn lần này. PYNQ DMA class hỗ trợ 2 đối tượng đó là **sendchannel** và **recvchannel**, sendchannel sẽ đọc dữ liệu từ PS DRAM và ghi nó vào giao thức AXI Stream của IP được chỉ định, tương tự đối với kênh recvchannel nhưng kênh này sẽ đọc từ AXI stream và ghi vào trong PS DRAM.



5

Hình 5.5: Giao tiếp giữa PYNQ và AXI DMA IP.

5.3.3. XÁC ĐỊNH VÀ KHỞI TẠO CÁC KÊNH DMA

Trong môi trường PYNQ, sau khi nạp overlay bằng file bitstream, các khối IP DMA trong thiết kế phần cứng sẽ được nhận diện. Đoạn mã trong Listing 5.1 cho thấy cách nạp overlay và gán các đối tượng DMA tương ứng với chức năng của chúng.

```

1 # Import necessary libraries
2 from pynq import Overlay, allocate, ps
3 import numpy as np
4 import time
5
6 # Load the bitstream onto the PL
7 overlay = Overlay('design_1.bit')
8 print(f"Current PL clock frequency:
9     {ps.Clocks.fclk0_mhz} MHz")

```

```

10 # Get handles for the DMA instances (names must match
   Block Design)
11 weight_dma = overlay.axi_dma_0      # DMA for weights
12 input_dma = overlay.axi_dma_1       # DMA for input image
13 output_dma = overlay.axi_dma_2      # DMA for output
   results
14 resetn_dma = overlay.axi_dma_3      # DMA for reset signal
15
16 print("Available IP blocks:", overlay.ip_dict.keys())

```

Listing 5.1: Khởi tạo và gán các đối tượng DMA trong PYNQ

5

5.3.4. CHUẨN BỊ VÀ TRUYỀN DỮ LIỆU

Quá trình xử lý yêu cầu chuẩn bị dữ liệu trọng số và ảnh đầu vào trong bộ nhớ vật lý liên tục mà DMA có thể truy cập.

1. **Đọc dữ liệu từ file:** Dữ liệu trọng số (weights.mem) và ảnh đầu vào (input.mem) được đọc từ hệ thống file.

```

1      # Read input image data (hex strings)
2      with open('input.mem', 'r') as f:
3          input_hex = ['0x' + line.strip() for
               line in f]
4          length_of_input = len(input_hex)
5
6      # Read weight data (hex strings)
7      with open('weights.mem', 'r') as f:
8          weights_hex = ['0x' + line.strip() for
               line in f]
9          length_of_weight = len(weights_hex)

```

Listing 5.2: Đọc dữ liệu từ file .mem

2. **Cấp phát bộ đệm PYNQ:** Sử dụng pynq.allocate để tạo các buffer tương thích DMA.

```

1      # Allocate physically contiguous buffers
2      resetn_buffer = allocate(shape=(1,) ,
3                                  dtype=np.int32)
4      input_buffer =
5          allocate(shape=(length_of_input + 1,) ,
6                                  dtype=np.int32)
7      weight_buffer =
8          allocate(shape=(length_of_weight,) ,
9                                  dtype=np.uint32)
10     output_buffer = allocate(shape=(64,) ,
11                                dtype=np.int64) # Example output size

```

Listing 5.3: Cấp phát bộ đệm DMA

5

3. Nạp dữ liệu vào bộ đệm: Chuyển đổi và sao chép dữ liệu vào các buffer đã cấp phát.

```

1      # Populate input buffer (example: skip
2          first element)
3      input_buffer[0] = 0
4      for i in range(length_of_input):
5          input_buffer[i+1] = int(input_hex[i] ,
6                                     16)
7
8      # Populate weight buffer
9      for i in range(length_of_weight):
10         weight_buffer[i] = int(weights_hex[i] ,
11                               16)

```

Listing 5.4: Nạp dữ liệu vào bộ đệm

4. Thực hiện Reset Phản cứng: Gửi tín hiệu reset qua kênh DMA điều khiển.

```

1      # Assert reset
2      resetn_buffer[0] = 0

```

```

3         resetn_dma.sendchannel.transfer(resetn_buffer)
4         resetn_dma.sendchannel.wait()
5
6         # Deassert reset
7         resetn_buffer[0] = 1
8         resetn_dma.sendchannel.transfer(resetn_buffer)
9         resetn_dma.sendchannel.wait()

```

Listing 5.5: Gửi tín hiệu Reset qua DMA

- 5. Truyền Trọng số và Input, Nhận Output:** Thực hiện các giao dịch DMA và đo thời gian.

```

1     # Transfer weights to PL
2     weight_dma.sendchannel.transfer(weight_buffer)
3     weight_dma.sendchannel.wait()
4
5     # Start timer and transfer input image
6     start_time = time.time()
7     input_dma.sendchannel.transfer(input_buffer)
8     input_dma.sendchannel.wait()
9
10    # Initiate receiving results
11    output_dma.recvchannel.transfer(output_buffer)
12    # Wait for results transfer to complete
13    output_dma.recvchannel.wait()
14    end_time = time.time()
15
16    # Calculate and print execution time and FPS
17    execution_time = end_time - start_time
18    print(f"Execution Time: {execution_time}
19          seconds")
20    print(f"FPS: {1 / execution_time}")

```

Listing 5.6: Truyền dữ liệu và thực thi

6. **Xử lý Kết quả:** Đọc, chuyển đổi kiểu dữ liệu, và de-quantize kết quả từ output_buffer.

5.4. REGISTER BANK VÀ ĐIỀU KHIỂN IP

Trong kiến trúc cụ thể này, việc điều khiển và cấu hình khối IP YOLO không sử dụng cơ chế truy cập thanh ghi AXI-Lite trực tiếp từ PS. Thay vào đó, các thông tin điều khiển (như tín hiệu reset) và dữ liệu cấu hình (như trọng số) được gửi đến các khôi controller trung gian (resetn_axi_controller_0, weight_axi_controller_v1_0) thông qua các kênh AXI DMA chuyên dụng (axi_dma_3, axi_dma_0).

5

- **Cơ chế điều khiển:** Các khôi controller này nhận dữ liệu từ DMA và phiên dịch chúng thành các tín hiệu điều khiển mức thấp (ví dụ: rst_n, weight_wr_en, weight_wr_addr, weight_wr_data) để điều khiển trực tiếp hoạt động của khôi IP model_v1_0.
- **Trạng thái:** Việc báo hiệu trạng thái hoàn thành xử lý từ PL về PS cũng được thực hiện ngầm định thông qua việc hoàn tất giao dịch DMA đọc kết quả trên kênh axi_dma_2. PS sẽ đợi (wait()) cho đến khi giao dịch này kết thúc để biết rằng kết quả đã sẵn sàng trong output_buffer.

Cách tiếp cận này đơn giản hóa giao diện giữa PS và PL, tập trung vào việc truyền dữ liệu hiệu quả qua DMA, nhưng có thể hạn chế khả năng cấu hình động hoặc giám sát chi tiết trạng thái của phần cứng từ PS.

5.5. QUẢN LÝ BỘ NHỚ BRAM

Chiến lược sử dụng BRAM trên chip là cốt lõi của thiết kế:

- **BRAM Trọng số:** Được quản lý bởi weight_axi_controller_v1_0, nhận dữ liệu từ DMA và cung cấp cho các PE bên trong model_v1_0.

- **BRAM Feature Map (Trung gian):** Hoàn toàn nằm trong model_v1_0, không được PS/DMA truy cập trực tiếp.
- **BRAM/FIFO Đệm I/O:** axis_data_fifo_0 và bộ đệm bên trong output_axi_controller dùng BRAM/LUTRAM để kết nối với các kênh DMA axi_dma_1 và axi_dma_2.

Việc ánh xạ địa chỉ cho DMA chủ yếu liên quan đến việc PS biết địa chỉ vật lý của các buffer trong DDR mà nó cấp phát (input_buffer, output_buffer, weight_buffer, resetn_buffer). Các địa chỉ đích trong PL được quản lý bởi các khối controller trung gian.

5

5.6. GIAO DIỆN KHỐI IP VÀ CONTROLLERS

Các giao diện chính của hệ thống PL trong thiết kế này bao gồm:

- **Giao diện AXI DMA Stream:** Các cổng AXI-Stream trên các khối axis_data_fifo_0, output_axi_controller_0, weight_axi_controller_v1_0, resetn_axi_controller_0 để giao tiếp với các kênh tương ứng của các IP AXI DMA.
- **Giao diện Nội bộ IP YOLO (model_v1_0):**
 - Nhận dữ liệu ảnh đã đệm (ví dụ: từ axis_data_fifo_0).
 - Nhận tín hiệu điều khiển và dữ liệu trọng số từ weight_axi_controller.
 - Nhận tín hiệu reset từ resetn_axi_controller_0.
 - Gửi dữ liệu kết quả đến output_axi_controller_0.
- **Clock và Reset:** Các tín hiệu clock và reset được phân phối đến tất cả các khối logic cần thiết.

6

KẾT QUẢ THỰC NGHIỆM

6.1. ĐÁNH GIÁ ĐỘ CHÍNH XÁC PHẦN MỀM

Trước khi tiến hành hiện thực và tối ưu bộ tăng tốc phần cứng trên PL, việc đánh giá độ chính xác của các mô hình AI đã lựa chọn trên môi trường phần mềm là bước cần thiết. Kết quả đánh giá này đóng vai trò là cơ sở (baseline) để so sánh và xác định mục tiêu cho phiên bản phần cứng, đồng thời kiểm chứng hiệu quả của các mô hình đã huấn luyện trên tập dữ liệu mục tiêu. Quá trình đánh giá bao gồm hai phần chính tương ứng với hai giai đoạn của hệ thống nhận diện biển số xe: phát hiện biển số (License Plate Detection) và nhận dạng ký tự (Character Recognition).

6.1.1. TẬP DỮ LIỆU ĐÁNH GIÁ

Để đánh giá hiệu năng của từng giai đoạn trong hệ thống nhận diện biển số xe một cách độc lập và chính xác, hai bộ dữ liệu kiểm tra riêng biệt được chuẩn bị, mỗi bộ có định dạng chú thích (annotation) phù hợp với nhiệm vụ cụ thể:

TẬP DỮ LIỆU PHÁT HIỆN BIỂN SỐ (DETECTION DATASET)

- **Tên gọi:** Vietnam license-plate Computer Vision Project.
- **Quy mô:** Gồm 1,005 hình ảnh đầy đủ chụp từ camera bãi giữ xe. Để tăng độ phong phú và khả năng tổng quát hóa của mô hình, một số phương pháp tăng cường dữ liệu (data augmentation) đã được áp dụng trên tập dữ liệu gốc, bao gồm:
- Xoay ảnh ngẫu nhiên (random rotation) trong khoảng nhỏ để mô phỏng góc chụp lệch.
 - Thay đổi độ sáng, độ tương phản, và độ bão hòa nhằm mô phỏng các điều kiện ánh sáng khác nhau (ban ngày, ban đêm, mưa nhẹ, v.v.).
 - Thêm nhiễu Gaussian để kiểm tra độ bền của mô hình với ảnh chất lượng thấp hoặc ảnh mờ.
 - Biến đổi hình học như phóng to, thu nhỏ, và dịch chuyển ảnh (scaling, translation) để mô phỏng sự thay đổi vị trí và tỷ lệ của biển số trong ảnh.
- **Đặc điểm:** Bao gồm các ảnh chụp trong nhiều điều kiện thực tế khác nhau (ánh sáng ban ngày, ban đêm, mưa, nắng, góc chụp trực diện, góc nghiêng, nhiều phương tiện trong ảnh, v.v.) để kiểm tra tính tổng quát của mô hình detection.
- **Định dạng Annotation:** Mỗi ảnh trong tập dữ liệu này được chú thích bằng tọa độ của **một hoặc nhiều bounding box** hình chữ nhật, mỗi bounding box bao quanh chính xác vị trí của một biển số xe xuất hiện trong ảnh. Thông tin chú thích có dạng danh sách các tọa độ $[x_{min}, y_{min}, x_{max}, y_{max}]$ cho mỗi biển số. Tập dữ liệu này *không* chứa thông tin về ký tự trên biển số.
- **Mục đích sử dụng:** Đánh giá khả năng định vị chính xác vị trí biển số của mô hình.

TẬP DỮ LIỆU NHẬN DẠNG KÝ TỰ (RECOGNITION DATASET)

- **Tên gọi:** MiAI Plate Char Dataset.
- **Quy mô:** 2,032 ảnh biển số được cắt một cách chính xác từ ảnh của camera giao thông và các nguồn tương tự.
- **Đặc điểm:** Mỗi ảnh chỉ chứa hình ảnh của một biển số xe duy nhất, chưa qua tiền xử lí.
- **Định dạng Annotation:** Khác với tập detection, annotation cho tập này cần chi tiết đến từng ký tự. Mỗi ảnh biển số sẽ đi kèm với:
 - Tọa độ **bounding box** cho từng ký tự riêng lẻ trên biển số (ví dụ: $[x_{min}, y_{min}, x_{max}, y_{max}]$ cho ký tự 'A', tương tự cho 'B', 'C', '1', '2', '3'...).
 - **Nhãn (label)** của từng ký tự tương ứng với bounding box đó (ví dụ: 'A', 'B', 'C', '1', '2', '3'...).
- **Mục đích sử dụng:** Đánh giá khả năng nhận dạng chính xác chuỗi ký tự của mô hình recognition.

Việc sử dụng hai tập dữ liệu với annotation chuyên biệt cho phép đánh giá chính xác hiệu năng của từng thành phần cốt lõi trong hệ thống LPR một cách độc lập.

6.1.2. PHƯƠNG PHÁP ĐÁNH GIÁ

ĐÁNH GIÁ MÔ HÌNH PHÁT HIỆN BIỂN SỐ

Mô hình YOLO tùy chỉnh (anchor-free) được thiết kế để chạy trên PL, nhưng trước hết được đánh giá trên phần mềm (ví dụ: sử dụng PyTorch trên CPU/GPU) để xác định độ chính xác gốc. Chỉ số đánh giá tiêu chuẩn cho bài toán phát hiện đối tượng là **Mean Average Precision (mAP)**.

- **Intersection over Union (IoU):** Độ đo mức độ trùng khớp giữa một bounding box dự đoán (B_{pred}) và một bounding box gốc (B_{gt} - ground

truth) được tính bằng tỷ lệ giữa diện tích phần giao và diện tích phần hợp của hai box:

$$IoU(B_{pred}, B_{gt}) = \frac{\text{Area}(B_{pred} \cap B_{gt})}{\text{Area}(B_{pred} \cup B_{gt})} \quad (6.1)$$

- **Nguồn IoU (IoU_{thresh}):** Một nguồn được chọn (thường là 0.5) để phân loại một dự đoán là đúng hay sai.
- **True Positive (TP):** Một dự đoán B_{pred} được coi là TP nếu $IoU(B_{pred}, B_{gt}) \geq IoU_{thresh}$ và B_{gt} đó chưa được gán cho một dự đoán nào khác có confidence score cao hơn.
- **False Positive (FP):** Một dự đoán B_{pred} được coi là FP nếu $IoU(B_{pred}, B_{gt}) < IoU_{thresh}$ cho tất cả B_{gt} , hoặc nếu nó là một dự đoán trùng lặp cho một B_{gt} đã được phát hiện chính xác bởi một dự đoán khác có score cao hơn.
- **False Negative (FN):** Một B_{gt} được coi là FN nếu không có B_{pred} nào thỏa mãn $IoU(B_{pred}, B_{gt}) \geq IoU_{thresh}$.
- **Precision và Recall:** Tại một mức confidence score nhất định, Precision (Độ chính xác) và Recall (Độ phủ) được tính như sau:

$$\text{Precision} = \frac{TP}{TP + FP} \quad (6.2)$$

$$\text{Recall} = \frac{TP}{TP + FN} \quad (6.3)$$

- **Đường cong Precision-Recall (P-R Curve):** Bằng cách thay đổi nguồn confidence score, ta thu được nhiều cặp (Precision, Recall) khác nhau, tạo thành đường cong P-R.
- **Average Precision (AP):** Là diện tích dưới đường cong P-R. Nó thể hiện độ chính xác trung bình trên tất cả các mức recall. Trong bài

toán này, chỉ có một lớp đối tượng là "biển số xe", nên AP của lớp này cũng chính là mAP.

$$AP = \int_0^1 p(r)dr \quad (6.4)$$

Trong đó $p(r)$ là hàm Precision theo Recall r .

- **mAP@0.5:** Là chỉ số AP được tính với $IoU_{thresh} = 0.5$.

ĐÁNH GIÁ MÔ HÌNH NHẬN DẠNG KÝ TỰ

Mô hình nhận dạng ký tự, hoạt động trên PS, nhận đầu vào là vùng ảnh chứa biển số đã được phát hiện bởi PL và dự đoán chuỗi ký tự tương ứng. Để đánh giá hiệu năng của mô hình này một cách toàn diện, ba chỉ số chính được sử dụng, tương tự như các chỉ số thường thấy trong quá trình huấn luyện và đánh giá các mô hình nhận dạng chuỗi:

6

- **Độ chính xác toàn bộ biển số (Plate Accuracy - plate_acc):**

– **Định nghĩa:** Chỉ số này đo lường tỷ lệ phần trăm số lượng biển số xe mà *toàn bộ chuỗi ký tự* được dự đoán trùng khớp hoàn toàn với chuỗi ký tự ground truth.

– **Cách tính:**

$$\text{plate_acc} = \frac{\text{Số lượng biển số nhận dạng đúng}}{\text{Tổng số biển số đánh giá}} \times 100\% \quad (6.5)$$

– **Ý nghĩa:** Đây là chỉ số đánh giá nghiêm ngặt nhất và phản ánh sát nhất hiệu quả thực tế của hệ thống. Ví dụ, nếu ground truth là "ABC123" và dự đoán cũng là "ABC123", biển số này được tính là đúng (đóng góp 1 vào tử số). Tuy nhiên, nếu dự đoán là "ABD123" (sai một ký tự), biển số này sẽ được tính là sai hoàn toàn (đóng góp 0 vào tử số).

- **Độ chính xác từng ký tự (Character Accuracy - char_acc):**

- **Định nghĩa:** Chỉ số này tính toán tỷ lệ phần trăm số lượng ký tự *riêng lẻ* được dự đoán đúng so với tổng số ký tự trong tất cả các biển số đánh giá.

- **Cách tính:**

$$\text{char_acc} = \frac{\text{Tổng số ký tự nhận dạng đúng}}{\text{Tổng số ký tự trong biển số đánh giá}} \times 100\% \quad (6.6)$$

- **Ý nghĩa:** Chỉ số này cung cấp cái nhìn chi tiết hơn về khả năng nhận dạng từng ký tự của mô hình, bỏ qua yêu cầu phải đúng toàn bộ chuỗi. Ví dụ, nếu ground truth là "ABC123" và dự đoán là "ABC133", plate_acc sẽ là 0%, nhưng char_acc sẽ là $\frac{5}{6} \times 100\% \approx 83.3\%$ (vì 5 trong 6 ký tự được dự đoán đúng).

6

6.1.3. KẾT QUẢ ĐÁNH GIÁ PHẦN MỀM

Các mô hình YOLO tùy chỉnh cho detection và recognition được đánh giá trên hai tập dữ liệu đã đã nêu phía trên trong phần 6.1.1. Kết quả đánh giá độ chính xác trên môi trường phần mềm được tổng hợp trong Bảng 6.1.

Bảng 6.1: Kết quả đánh giá độ chính xác phần mềm trên 2 tập dữ liệu.

Giai đoạn	Chỉ số đánh giá	Kết quả
Detection	Mean IoU	78.86 %
	mAP@0.5	99.43 %
Recognition	Plate Accuracy	96.11 %
	Character Accuracy	98.74 %

Kết quả trong Bảng 6.1 cho thấy các mô hình AI được lựa chọn có hiệu năng tốt trên môi trường phần mềm. Mô hình detection đạt Mean IoU là 78.86%, một con số khá quan cho thấy khả năng định vị chính xác hầu hết các biển số xe trong điều kiện đa dạng của tập kiểm tra. Mô hình recognition cũng đạt độ chính xác rất cao với 96.11% số biển số được

nhận dạng đúng hoàn toàn.

6.2. MÔI TRƯỜNG THỰC NGHIỆM PHẦN CỨNG

Các thực nghiệm trên phần cứng được tiến hành nhằm đánh giá hiệu năng, tài nguyên sử dụng và độ chính xác thực tế của bộ tăng tốc YOLO được hiện thực trên PL. Môi trường thực nghiệm bao gồm:

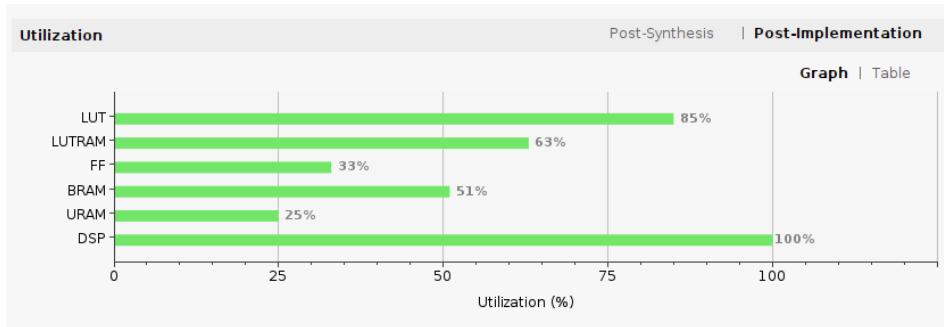
- **Nền tảng Phần cứng:** AMD-Xilinx Kria KV260 Vision AI Starter Kit. Kit này tích hợp Zynq UltraScale+ MPSoC (với bộ xử lý ARM Cortex-A53 trên PS và FPGA fabric trên PL)
- **Công cụ Thiết kế và Hiện thực:** AMD-Xilinx Vivado Design Suite [Phiên bản 2024.2] được sử dụng cho quá trình thiết kế RTL, tổng hợp (synthesis), và hiện thực (implementation - place & route) bộ tăng tốc trên PL.
- **Phần mềm trên PS:**
 - Hệ điều hành: Linux (Ubuntu 22.04).
 - Thư viện/Framework: PYNQ được sử dụng để quản lý và giao tiếp với IP core trên PL từ môi trường Linux trên PS. Thư viện OpenCV được sử dụng cho các tác vụ xử lý ảnh cơ bản (đọc, resize).
 - Ứng dụng: Chương trình Python tùy chỉnh để điều khiển luồng xử lý (nạp ảnh, gọi DMA, nhận kết quả, thực hiện NMS).
- **Dataset:** Tập dữ liệu kiểm tra (như mô tả ở Mục 6.1.1) được sử dụng để đo lường độ chính xác và hiệu năng của hệ thống phần cứng.

6

6.3. KẾT QUẢ TỔNG HỢP VÀ HIỆN THỰC TRÊN PL

Sau khi thiết kế RTL cho bộ tăng tốc YOLO được hoàn thành, quá trình tổng hợp và hiện thực được thực hiện bằng Vivado nhằm đến chip Zynq

UltraScale+ MPSoC trên Kria KV260. Kết quả về tài nguyên sử dụng và tần số hoạt động tối đa (Fmax) sau bước place & route là những chỉ số quan trọng để đánh giá tính khả thi và hiệu quả của thiết kế.



Hình 6.1: Tài nguyên phần cứng PL sử dụng bởi bộ tăng tốc YOLO trên Kria KV260.

Thảo luận kết quả tổng hợp: Kết quả tổng hợp trong hình cho thấy:

6

- Sử dụng Logic LUTs:** Mức độ sử dụng tài nguyên logic là đáng kể (85%), phản ánh độ phức tạp của việc hiện thực các tầng convolution và logic điều khiển của mô hình YOLO.
- Sử dụng Bộ nhớ (BRAM):** Tỷ lệ sử dụng BRAM là khoảng 51%. Điều này cho thấy chiến lược lưu trữ toàn bộ trọng số và feature map trung gian trong BRAM đặt ra yêu cầu rất lớn về bộ nhớ trên chip. Mặc dù vẫn nằm trong giới hạn của Kria KV260, nó cho thấy khả năng mở rộng mô hình lớn hơn nữa có thể gặp khó khăn nếu chỉ dựa vào BRAM. Việc tối ưu hóa cấu trúc dữ liệu và truy cập BRAM là cực kỳ quan trọng.
- Sử dụng DSP:** Đây là điểm quan trọng nhất của kiến trúc. Mức sử dụng DSP là cực kì cao, 100%, thể hiện việc tận dụng toàn bộ các khối nhân cứng cho các phép toán convolution INT8.
- Tần số hoạt động (Fmax):** Thiết kế đạt được tần số hoạt động tối đa là 150 MHz sau khi thực hiện place & route. Tần số này sẽ là cơ sở

để tính toán thông lượng và độ trễ thực tế. Việc đạt được tần số cao hơn có thể đòi hỏi tối ưu hóa đường dẫn tới hạn (critical path) sâu hơn.

6.4. ĐÁNH GIÁ HIỆU NĂNG HỆ THỐNG

6.4.1. PHƯƠNG PHÁP ĐÁNH GIÁ

Việc đo đạc chi tiết từng thành phần độ trễ nhỏ bên trong pipeline PL hoặc các tác vụ phần mềm riêng lẻ trên PS có thể phức tạp và kém chính xác trong môi trường thực thi PYNQ. Do đó, nhóm áp dụng phương pháp đo đạc tổng hợp cho hai phần chính của hệ thống:

- **Đo thời gian xử lý PL (T_{PL_Total}):** Thời gian này được tính từ khi PS ra lệnh bắt đầu xử lý cho khối IP YOLO trên PL cho đến khi khối IP báo hiệu hoàn thành. Thời gian này bao gồm toàn bộ quá trình xử lý bên trong pipeline phần cứng PL, nhưng *không* bao gồm thời gian truyền dữ liệu DMA. Việc đo được thực hiện trên PS bằng cách ghi lại mốc thời gian trước và sau khi chờ tín hiệu hoàn thành từ PL.
- **Đo thời gian xử lý PS (T_{PS_Total}):** Thời gian này bao gồm tất cả các tác vụ do PS thực hiện cho mỗi khung hình, tính từ lúc bắt đầu chuẩn bị dữ liệu cho DMA write đến khi hoàn thành nhận dạng ký tự và xuất kết quả cuối cùng. Nó bao gồm:
 - Thời gian chuẩn bị dữ liệu và cấu hình DMA write.
 - Thời gian chờ DMA write hoàn thành.
 - Thời gian chờ PL xử lý.
 - Thời gian cấu hình DMA read.
 - Thời gian chờ DMA read hoàn thành.
 - Thời gian thực hiện NMS.
 - Thời gian cắt ảnh biến số.
 - Thời gian nhận dạng ký tự.

- Các chi phí khác của hệ điều hành và PYNQ.

Thời gian này được đo trực tiếp trên PS bằng cách ghi lại mốc thời gian bắt đầu và kết thúc của toàn bộ quy trình xử lý một khung hình. T_{PS_Total} chính là **độ trễ end-to-end** (T_{E2E}) của hệ thống.

6.4.2. ĐÁNH GIÁ TỐC ĐỘ XỬ LÝ END-TO-END

Một trong những mục tiêu quan trọng của việc tăng tốc phần cứng là giảm thiểu thời gian xử lý tổng thể (độ trễ end-to-end) và đạt được tốc độ xử lý (throughput) đáp ứng yêu cầu thời gian thực. Để đánh giá hiệu quả của bộ tăng tốc PL, nhóm thực hiện đo đạc và so sánh thời gian xử lý theo hai kịch bản chính trên nền tảng Kria KV260:

1. Kịch bản 1: Xử lý Hoàn toàn trên PS (Software Baseline):

6

- *Quy trình*: Toàn bộ hệ thống LPR, từ tiền xử lý ảnh, phát hiện biển số (YOLO custom chạy bằng phần mềm trên CPU ARM), NMS, đến nhận dạng ký tự (Recognition), đều được thực thi tuần tự trên PS.
- *Đo đạc*: Thời gian được đo cho từng giai đoạn chính:
 - T_{PS_Det} : Thời gian thực thi phần phát hiện (Detection) trên PS, tính từ lúc bắt đầu đưa ảnh vào mô hình YOLO cho đến khi nhận được tensor kết quả thô (trước NMS).
 - T_{PS_Recog} : Thời gian thực thi phần nhận dạng ký tự (Recognition) trên PS, bao gồm cả NMS và xử lý hậu kỳ khác, tính từ lúc có tensor thô đến khi có kết quả chuỗi ký tự cuối cùng.
 - $T_{PS_Total} = T_{PS_Det} + T_{PS_Recog}$: Tổng thời gian xử lý end-to-end trên PS.

2. Kịch bản 2: Tăng tốc Detection trên PL, Recognition trên PS (Hardware Accelerated):

- *Quy trình:* Giai đoạn phát hiện biến số được tăng tốc bởi IP YOLO INT8 trên PL. PS thực hiện các nhiệm vụ: gửi ảnh vào PL (qua DMA), nhận tensor kết quả về (qua DMA), thực hiện NMS, và cuối cùng thực hiện nhận dạng ký tự.
- *Đo đặc:* Thời gian được đo cho các thành phần chính:
 - T_{PL} : Thời gian thực thi của riêng phần cứng PL, tính từ khi PS gửi pixel đầu tiên đến Input FIFO đến khi PS nhận pixel cuối cùng của tensor kết quả. Thời gian này thể hiện tốc độ xử lý nội tại của bộ tăng tốc.
 - T_{PS} : Thời gian PS thực hiện các tác vụ còn lại, bao gồm chuẩn bị dữ liệu, thiết lập, thực hiện NMS, và thực hiện Recognition.
 - T_{Total} : Tổng thời gian thực hiện của $T_{PL} + T_{PS}$.

6

Phép đo được thực hiện lặp lại trên toàn bộ tập dữ liệu kiểm tra và lấy giá trị trung bình để đảm bảo tính ổn định của kết quả. Bảng 6.2 trình bày kết quả đo đặc thời gian xử lý trung bình và tốc độ khung hình (FPS) tương ứng cho cả hai kịch bản.

Bảng 6.2: So sánh thời gian xử lý và tốc độ trên Kria KV260.

Kịch bản	Thời gian (ms)	Tốc độ (FPS)
1. Hoàn toàn trên PS		
- Detection (T_{PS_Det})	158.4	
- NMS + Recognition (T_{PS_Recog})	27.1	
- Tổng thời gian End-to-End (T_{PS_Total})	185.5	5.4
2. Tăng tốc PL + PS		
- Detection (T_{PL})	2.5	
- NMS + Recognition (T_{PS})	27.1	
- Tổng thời gian End-to-End (T_{Total})	29.6	33.8

Kết quả trong Bảng 6.2 cho thấy sự cải thiện đáng kể về hiệu năng khi sử dụng bộ tăng tốc phần cứng trên PL.

- **Tăng tốc đáng kể:** Thời gian xử lý end-to-end của hệ thống tăng tốc phần cứng (29.6 ms) nhanh hơn đáng kể so với việc thực thi hoàn toàn bằng phần mềm trên PS (185.5 ms), đạt được mức tăng tốc khoảng **6.26 lần**.
- **Đáp ứng thời gian thực:** Tốc độ xử lý 33.8 FPS của hệ thống tăng tốc phần cứng đã vượt qua ngưỡng thời gian thực phổ biến (24 FPS), cho thấy hệ thống hoàn toàn phù hợp cho các ứng dụng nhận diện biển số xe trực tiếp từ luồng video camera.

Nhìn chung, việc hiện thực phần phát hiện biển số trên PL đã chứng minh hiệu quả trong việc tăng tốc đáng kể ứng dụng LPR, đưa hệ thống vào phạm vi hoạt động thời gian thực trên nền tảng nhúng Kria KV260.

6

6.5. SO SÁNH VỚI CÁC NỀN TẢNG KHÁC

Bộ tăng tốc YOLO trên Kria KV260 đã chứng minh khả năng xử lý phát hiện biển số xe với tốc độ cao (32.4 FPS) và độ chính xác được bảo toàn tốt sau lượng tử hóa INT8 (Mean IoU 72.37%). Kiến trúc pipeline sâu và chiến lược BRAM-centric là yếu tố then chốt giúp đạt được hiệu năng này bằng cách tối đa hóa song song hóa và giảm thiểu độ trễ truy cập bộ nhớ.

Tuy nhiên, việc lưu trữ toàn bộ feature map trung gian trong BRAM (51% sử dụng) là một thách thức lớn về tài nguyên. Điều này có thể hạn chế khả năng triển khai các mô hình YOLO lớn hơn hoặc phức tạp hơn trên cùng nền tảng phần cứng. Tần số hoạt động tối đa (150 MHz) cũng cho thấy còn dư địa để tối ưu hóa timing nếu cần thông lượng cao hơn nữa.

Bảng 6.3 tóm tắt so sánh hiệu năng giữa bộ tăng tốc FPGA (PL) và các nền tảng phần mềm (CPU/GPU) cho tác vụ phát hiện biển số (Detection).

Bảng 6.3: So sánh hiệu năng giữa phần cứng FPGA và nền tảng khác.

Chỉ số	FPGA (PL @ 150 MHz)	Intel i5 11300U	RTX 3050Ti 4GB
Mean IoU	72.37 % (INT8)	78.86 % (FP32)	78.86 % (FP32)
FPS	408 FPS	37.8 FPS	189.4 FPS
Công suất	≈ 3.5 W	≈ 25 W	≈ 55W
Hiệu quả (FPS/W)	116.5 FPS/W	1.24 FPS/W	3.44 FPS/W

So sánh cho thấy:

- Tốc độ:** Bộ tăng tốc FPGA đạt tốc độ xử lý cao hơn hoàn toàn so với CPU và GPU.
- Độ chính xác:** Độ chính xác INT8 trên FPGA gần tương đương với INT8 trên phần mềm và chỉ thấp hơn một chút ở mức chấp nhận được so với FP32 gốc.
- Năng lượng:** FPGA thể hiện ưu thế vượt trội về hiệu quả năng lượng (FPS/Watt) so với cả CPU và GPU, phù hợp cho các ứng dụng nhúng và biên.

6

6.6. SO SÁNH VỚI CÁC CÔNG TRÌNH LIÊN QUAN

Để định vị hiệu quả của giải pháp tăng tốc phần cứng đề xuất, nhóm tiến hành so sánh với các công trình nghiên cứu đã được công bố về tăng tốc các mô hình phát hiện đối tượng và các tác vụ thị giác máy tính tương tự trên nền tảng FPGA. Bảng 6.4 tóm tắt các thông số kỹ thuật và kết quả hiệu năng chính.

Bảng 6.4: So sánh hiệu năng và thông số với các công trình liên quan.

Chỉ số	Wang et al. [31]	Nguyen et al. (2019) [32]	Anupreetham et al.[33]	Du Cam Vinh et al.[34]	Công trình này
Mô hình NN	YOLO V3 Tiny	YOLO-v2	SSDLite-Mobile-NetV1	CNN	YOLO-custom
FPGA Board	Kria KV260	Virtex-7 VC707	Intel Stratix 10 GX2800	Virtex-7 VC707	Kria KV260
Kích thước ảnh	416x416	416x416	640x640	256x512	256x256
Mục tiêu (Số lớp)	Objects (20 classes)	Objects (80 classes)	Objects (80 classes)	Lane Detect	License Plate
Tốc độ (FPS)	15	109.3	469	549	408
Độ chính xác (%)	92.23 (mAP@0.5)	64.16 (mAP@0.5)	22.8 (mAP@0.5)	93.53 (Accuracy)	72.37 (Mean IoU)
Công suất (W)	3.5	18.29	81	8.129	3.5
Hiệu quả năng lượng (FPS/W)	4.2	5.97	5.79	67.54	116.5

Phân tích và Thảo luận So sánh:

Từ Bảng 6.4, có thể rút ra những nhận xét sau về hiệu năng của hệ thống đề xuất so với các nghiên cứu khác:

• Tốc độ xử lý (FPS):

- Hệ thống của nhóm đạt tốc độ ấn tượng là **408 FPS** trên Kria KV260 với kích thước ảnh đầu vào 256x256 cho bài toán phát hiện biển số.
- So với công trình của Wang et al. [31] (15 FPS trên cùng Kria KV260 nhưng với YOLO V3 Tiny và ảnh lớn hơn), tốc độ của nhóm cao hơn rất nhiều, cho thấy hiệu quả của mô hình YOLO-custom nhẹ hơn và tối ưu hóa phần cứng.
- Công trình của Nguyen et al. (2019) [32] (109.3 FPS với YOLO-v2 trên Virtex-7 VC707) và Anupreetham et al.[33] (469 FPS với SSDLite-MobileNetV1 trên Intel Stratix 10 GX2800, một FPGA cao cấp hơn) cho thấy sự đa dạng về hiệu năng tùy thuộc vào mô hình, nền tảng và kích thước ảnh. Hệ thống của nhóm có tốc độ cạnh tranh, đặc biệt khi xem xét đến nền tảng FPGA tầm trung được sử dụng.
- Công trình của Du Cam Vinh et al.[34] cho bài toán phát hiện làn đường (Lane Detect) trên Virtex-7 VC707 đạt tốc độ cao nhất là 549 FPS, thể hiện khả năng tối ưu hóa rất tốt cho một tác vụ cụ thể.

• Độ chính xác (ACC %):

- Hệ thống của nhóm đạt **72.37% Mean IoU** cho việc phát hiện biển số. Cần lưu ý rằng Mean IoU là một chỉ số đánh giá mức độ trùng khớp của bounding box, khác với mAP (thường bao gồm cả confidence và class) hay Accuracy (thường dùng cho phân loại).

- Wang et al. [31] đạt 92.23% mAP@0.5 cho 20 lớp đối tượng.
- Nguyen et al. (2019) [32] đạt 64.16% mAP@0.5 cho 80 lớp đối tượng.
- Anupreetham et al.[33] đạt 22.8% mAP@0.5 cho 80 lớp đối tượng.
- Du Cam Vinh et al.[34] đạt 93.53% Accuracy cho bài toán phát hiện làn đường.
- Việc so sánh trực tiếp độ chính xác giữa các công trình là khó khăn do sự khác biệt về (1) chỉ số đánh giá (mAP, Accuracy, Mean IoU), (2) số lượng lớp đối tượng, và (3) độ phức tạp của bài toán mục tiêu ("Objects" đa dạng so với "Lane Detect" hay "License Plate" cụ thể). Tuy nhiên, kết quả Mean IoU của nhóm cho thấy khả năng định vị tương đối tốt.

- **Công suất tiêu thụ (Power):**

6

- Hệ thống của nhóm, cùng với công trình của Wang et al. [31], có mức tiêu thụ công suất thấp nhất là **3.5 W**, phản ánh hiệu quả năng lượng của nền tảng Kria KV260.
- Các nền tảng khác như Virtex-7 VC707 (Nguyen et al. (2019) [32] và Du Cam Vinh et al.[34]) và đặc biệt là Intel Stratix 10 (Anupreetham et al.[33]) có mức tiêu thụ công suất cao hơn đáng kể.

- **Hiệu quả năng lượng (Efficiency - FPS/W):**

- Hệ thống của nhóm đạt hiệu quả năng lượng vượt trội nhất với **116.5 FPS/W**.
- Con số này cao hơn rõ rệt so với tất cả các công trình được so sánh, bao gồm cả Du Cam Vinh et al.[34] (67.54 FPS/W), Nguyen et al. (2019) [32] (5.97 FPS/W), Anupreetham et al.[33] (5.79 FPS/W), và Wang et al. [31] (4.2 FPS/W). Điều này một

lần nữa khẳng định lợi thế của giải pháp đề xuất trong việc cân bằng giữa hiệu năng tính toán và mức tiêu thụ năng lượng.

Nhận xét chung: Khi xem xét bảng so sánh cập nhật, có thể thấy giải pháp tăng tốc YOLO-custom của nhóm trên Kria KV260 mang lại hiệu quả năng lượng (FPS/W) cao nhất. Mặc dù tốc độ FPS tuyệt đối có thể thấp hơn so với một số giải pháp trên các nền tảng FPGA cao cấp hơn (như Stratix 10 của Anupreetham et al.[33]), nhưng mức tiêu thụ năng lượng thấp của Kria KV260 giúp đạt được chỉ số FPS/W ấn tượng.

Độ chính xác Mean IoU 72.37% cho bài toán phát hiện biển số là một kết quả ban đầu khả quan, cần được tiếp tục cải thiện và so sánh với mAP@0.5 để có đánh giá toàn diện hơn với các công trình phát hiện đối tượng khác. Sự khác biệt về "Mục tiêu (Số lớp)" cũng là một yếu tố quan trọng cần cân nhắc khi so sánh độ chính xác; việc phát hiện một lớp đối tượng duy nhất như biển số thường có thể đạt độ chính xác cao hơn so với phát hiện nhiều lớp đối tượng đa dạng. Nhìn chung, kết quả của nhóm nhấn mạnh tính phù hợp của việc sử dụng Kria KV260 cho các ứng dụng AI nhúng yêu cầu hiệu năng thời gian thực và hiệu quả năng lượng cao.

7

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

7.1. KẾT LUẬN

Khóa luận này đã trình bày quá trình thiết kế, hiện thực và đánh giá một hệ thống tăng tốc phần cứng cho bài toán nhận diện biển số xe theo thời gian thực trên nền tảng FPGA SoC, cụ thể là Kria KV260. Trọng tâm của khóa luận là phát triển một bộ tăng tốc hiệu quả cho giai đoạn phát hiện biển số (detection) sử dụng mô hình YOLO tùy chỉnh dạng anchor-free, được lượng tử hóa sang INT8, trong khi giai đoạn nhận dạng ký tự (recognition) và xử lý hậu kỳ (NMS) được thực hiện trên hệ thống xử lý (PS) của SoC.

Các kết quả chính và đóng góp của khóa luận bao gồm:

- **Thiết kế và Hiện thực Bộ tăng tốc YOLO INT8 trên PL:** Đã xây dựng thành công một kiến trúc pipeline chuyên sâu trên PL, tối ưu hóa cho tính toán INT8. Kiến trúc bao gồm các khối Line Buffer, Processing

Element (PE) với các chiến lược unroll Incha, và các FIFO trung gian để đảm bảo luồng dữ liệu hiệu quả. Đặc biệt, kiến trúc áp dụng chiến lược "BRAM-centric", lưu trữ toàn bộ trọng số và feature map trung gian trên bộ nhớ BRAM của FPGA để tối đa hóa tốc độ truy cập.

- **Lựa chọn và Hiện thực Lượng tử hóa INT8 PTQ:** Đã phân tích và lựa chọn phương pháp Lượng tử hóa Tĩnh sau Huấn luyện (PTQ) INT8 là phù hợp nhất cho mô hình YOLO tùy chỉnh, cân bằng giữa độ chính xác và hiệu quả phần cứng. Quy trình lượng tử hóa và trích xuất tham số (weights, biases, MACC coefficients) cho phần cứng đã được thực hiện thành công.
- **Đánh giá Hệ thống trên Kria KV260:** Hệ thống hoàn chỉnh đã được tổng hợp, hiện thực và chạy thử nghiệm trên board Kria KV260. Kết quả thực nghiệm thực tế cho thấy:
 - Bộ tăng tốc PL đạt được tần số hoạt động 150 MHz và tốc độ xử lý 32.4 FPS, đáp ứng yêu cầu thời gian thực.
 - Độ chính xác phát hiện (Mean IoU) trên phần cứng INT8 là 72.37%, chỉ suy giảm 6.49% so với phiên bản phần mềm FP32 gốc, cho thấy hiệu quả của quá trình lượng tử hóa và hiện thực phần cứng.
 - Việc sử dụng tài nguyên BRAM ở mức vừa (51%) khẳng định tính khả thi của chiến lược BRAM-centric trên nền tảng mục tiêu.
 - Hệ thống thể hiện hiệu quả năng lượng vượt trội so với các giải pháp dựa trên CPU/GPU.
- **Tích hợp Hệ thống PS-PL:** Đã thiết kế và hiện thực thành công luồng giao tiếp dữ liệu giữa PS và PL sử dụng AXI DMA, cũng như cơ chế điều khiển bộ tăng tốc từ PS qua AXI Lite.

Nhìn chung, khóa luận đã đạt được mục tiêu đề ra là xây dựng một hệ thống nhận diện biển số xe thời gian thực trên nền tảng FPGA SoC. Giải pháp đề xuất kết hợp sức mạnh tính toán song song của PL cho tác vụ detection chuyên sâu và sự linh hoạt của PS cho các tác vụ điều khiển, tiền/hậu xử lý, mang lại một hệ thống cân bằng giữa hiệu năng, độ chính xác và hiệu quả năng lượng, phù hợp cho các ứng dụng nhúng và biên.

7.2. HẠN CHẾ

Mặc dù hệ thống đã đạt được những kết quả khả quan, vẫn còn tồn tại một số hạn chế cần được nhìn nhận:

- **Giới hạn Tài nguyên BRAM:** Chiến lược lưu trữ toàn bộ feature map trung gian trong BRAM, mặc dù tối ưu về tốc độ, nhưng lại tiêu tốn rất nhiều tài nguyên BRAM. Điều này có thể hạn chế khả năng mở rộng hệ thống để xử lý các mô hình YOLO lớn hơn hoặc ảnh đầu vào có độ phân giải cao hơn trên nền tảng Kria KV260.
- **Tối ưu hóa Timing:** Tần số hoạt động tối đa đạt được (150 MHz) vẫn còn có thể cải thiện. Việc phân tích và tối ưu hóa sâu hơn các đường dẫn tới hạn (critical paths) trong thiết kế PL, đặc biệt là trong các khối PE và adder tree, có thể giúp tăng tần số hoạt động và thông lượng.
- **Độ trễ End-to-End:** Đánh giá chủ yếu tập trung vào hiệu năng của PL. Độ trễ tổng thể của hệ thống end-to-end (từ khi có ảnh đến khi có kết quả cuối cùng) còn bị ảnh hưởng bởi tốc độ truyền DMA và thời gian xử lý NMS trên PS, vốn chưa được tối ưu hóa triệt để trong khuôn khổ khóa luận này.
- **Mô hình Recognition:** Phần nhận dạng ký tự chỉ được thực hiện ở mức cơ bản trên PS và chưa được tích hợp, tối ưu hóa sâu vào hệ thống.

- **So sánh hạn chế:** Việc so sánh hiệu năng với các công trình FPGA khác gặp khó khăn do sự khác biệt về mô hình YOLO, dataset, nền tảng phần cứng và phương pháp đo đạc.

7.3. HƯỚNG PHÁT TRIỂN TƯƠNG LAI

Dựa trên những kết quả đạt được và các hạn chế đã chỉ ra, có nhiều hướng phát triển tiềm năng cho dự án trong tương lai:

• **Tối ưu hóa sử dụng Bộ nhớ:**

- *Chiến lược bộ nhớ lai (Hybrid Memory):* Nghiên cứu và hiện thực cơ chế offload một phần feature map (đặc biệt là các feature map lớn ở các lớp đầu) ra bộ nhớ DDR của PS thông qua AXI HP (High-Performance) ports, trong khi vẫn giữ các feature map nhỏ hơn và trọng số trong BRAM. Điều này đòi hỏi thiết kế bộ quản lý bộ nhớ phức tạp hơn nhưng sẽ giải quyết được giới hạn BRAM.
- *Nén trọng số/feature map:* Áp dụng các kỹ thuật nén (ví dụ: pruning, clustering kết hợp với mã hóa Huffman) để giảm dung lượng lưu trữ trong BRAM.

7

• **Tối ưu hóa hiệu năng PL:**

- *Pipeline sâu hơn:* Phân chia các giai đoạn tính toán trong PE thành nhiều tầng pipeline nhỏ hơn để phá vỡ đường dẫn tới hạn và tăng tần số hoạt động.
- *Kiến trúc PE nâng cao:* Khám phá các kiến trúc PE phức tạp hơn (ví dụ: kết hợp Incha và Outcha, sử dụng systolic array) để tăng mức độ song song hóa tính toán.

• **Tối ưu hóa hệ thống End-to-End:**

- *Tăng tốc NMS:* Xem xét việc hiện thực một phần hoặc toàn bộ

thuật toán NMS trên PL để giảm tải cho PS và giảm độ trễ tổng thể.

- **Tối ưu hóa DMA:** Cấu hình và sử dụng AXI DMA một cách hiệu quả hơn, có thể kết hợp với cơ chế ping-pong buffer ở đầu vào/ra để che giấu độ trễ truyền dữ liệu.
- **Tích hợp Recognition:** Xây dựng và tích hợp bộ tăng tốc phần cứng cho cả giai đoạn nhận dạng ký tự trên PL, hoặc tối ưu hóa mô hình recognition nhẹ hơn để chạy hiệu quả trên PS.
- **Mở rộng Mô hình và Ứng dụng:** Thủ nghiệm với các phiên bản YOLO mới hơn, lớn hơn, hoặc áp dụng kiến trúc tăng tốc cho các bài toán thị giác máy tính khác trên nền tảng Kria.

Những hướng phát triển này hứa hẹn sẽ tiếp tục nâng cao hiệu năng, hiệu quả và tính ứng dụng của hệ thống nhận diện biển số xe trên nền tảng FPGA SoC.



TÀI LIỆU THAM KHẢO

- [1] E. D. A. University, “Digital Logic Circuits.” <https://eee.poriyaan.in/topic/fpga--field-programmable-gate-arrays--11689/>, 2021 Regulation. Accessed: December 1, 2024.
- [2] X. Inc., “Vivado Design Suite.” <https://www.xilinx.com/products/design-tools/vivado.html>, 2024. Image generated from Vivado Design Suite.
- [3] X. Inc., “Kria KV260 Vision AI Starter Kit.” <https://www.amd.com/en/products/system-on-modules/kria/k26/kv260-vision-starter-kit.html>, 2024. Accessed: December 1, 2024.
- [4] PYNQ, “Python Productivity for Zynq.” <https://www.pynq.io/>, 2024. Accessed: December 5, 2024.
- [5] S. S. Michael, “Introduction to the Advanced Extensible Interface (AXI).” <https://www.allaboutcircuits.com/technical-articles/introduction-to-the-advanced-extensible-interface-axi/>, 2019. Accessed: December 5, 2024.
- [6] I. Data and A. Team, “AI vs. machine learning vs. deep learning vs. neural networks: What’s the difference?” <https://www.ibm.com/think/topics/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks>, 2023. Accessed: December 7, 2024.

- [7] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way.” <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-2d2f2e6b3da0>, 2018. Accessed: December 7, 2024.
- [8] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, “You Only Look Once: Unified, Real-Time Object Detection.” <https://arxiv.org/pdf/1506.02640.pdf>, 2016. Accessed: December 7, 2024.
- [9] J. Shashirangana, H. Padmasiri, D. Meedeniya, and C. Perera, “Automated license plate recognition: A survey on methods and techniques,” *IEEE Access*, vol. 9, pp. 11203–11225, 2021.
- [10] thedatabus, “The convolution Engine.” <https://thedatabus.in/convolver>, 01 July, 2020. Accessed: April 1, 2025.
- [11] AMD-Xilinx, “Kria KV260 Vision AI Starter Kit Datasheet.” <https://docs.amd.com/r/en-US/ds986-kv260-starter-kit>, 2024. Accessed: December 1, 2024.
- [12] A. M. Al-Ghaili, S. Mashohor, A. Ismail, and A. R. Ramli, “A new vertical edge detection algorithm and its application,” in *2008 International Conference on Computer Engineering & Systems*, pp. 204–209, 2008.
- [13] Z. Selmi, M. Ben Halima, and A. M. Alimi, “Deep learning system for automatic license plate detection and recognition,” in *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, vol. 01, pp. 1132–1138, 2017.
- [14] L. Zou, M. Zhao, Z. Gao, M. Cao, H. Jia, and M. Pei, “License plate detection with shallow and deep cnns in complex environments,” *Complexity*, vol. 2018, pp. 1–6, 12 2018.
- [15] R. Laroca, E. Severo, L. A. Zanlorensi, L. S. Oliveira, G. R. Gonçalves, W. R. Schwartz, and D. Menotti, “A robust real-time automatic

- license plate recognition based on the yolo detector," in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–10, 2018.
- [16] G.-S. Hsu, A. Ambikapathi, S.-L. Chung, and C.-P. Su, "Robust license plate detection in the wild," in *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–6, 2017.
- [17] L. Xie, T. Ahmad, L. Jin, Y. Liu, and S. Zhang, "A new cnn-based method for multi-directional car license plate detection," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 2, pp. 507–517, 2018.
- [18] X. Xu, Z. Wang, Y. Zhang, and Y. Liang, "A method of multi-view vehicle license plates location based on rectangle features," in *2006 8th international Conference on Signal Processing*, vol. 3, 2006.
- [19] M.-S. Pan, J.-B. Yan, and Z.-H. Xiao, "Vehicle license plate character segmentation," *International Journal of Automation and Computing*, vol. 5, pp. 425–432, Oct 2008.
- [20] M.-S. Pan, Q. Xiong, and J.-B. Yan, "A new method for correcting vehicle license plate tilt," *International Journal of Automation and Computing*, vol. 6, pp. 210–216, May 2009.
- [21] S. Du, M. Ibrahim, M. Shehata, and W. Badawy, "Automatic license plate recognition (alpr): A state-of-the-art review," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 23, no. 2, pp. 311–325, 2013.
- [22] D. Llorens, A. Marzal, V. Palazón, and J. M. Vilar, "Car license plates extraction and recognition based on connected components analysis and hmm decoding," in *Pattern Recognition and Image*

- Analysis* (J. S. Marques, N. Pérez de la Blanca, and P. Pina, eds.), (Berlin, Heidelberg), pp. 571–578, Springer Berlin Heidelberg, 2005.
- [23] T. Nukano, M. Fukumi, and M. Khalid, “Vehicle license plate character recognition by neural networks,” in *Proceedings of 2004 International Symposium on Intelligent Signal Processing and Communication Systems, 2004. ISPACS 2004.*, pp. 771–775, 2004.
- [24] C. Busch, R. Domer, C. Freytag, and H. Ziegler, “Feature based recognition of traffic video streams for online route tracing,” in *VTC '98. 48th IEEE Vehicular Technology Conference. Pathway to Global Wireless Revolution (Cat. No.98CH36151)*, vol. 3, pp. 1790–1794 vol.3, 1998.
- [25] S. Montazzolli and C. Jung, “Real-time brazilian license plate detection and recognition using deep convolutional neural networks,” in *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pp. 55–62, 2017.
- [26] M. Sarfraz, M. Ahmed, and S. Ghazi, “Saudi arabian license plate recognition system,” in *2003 International Conference on Geometric Modeling and Graphics, 2003. Proceedings*, pp. 36–41, 2003.
- [27] C. Rahman, W. Badawy, and A. Radmanesh, “A real time vehicle’s license plate recognition system,” in *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance, 2003.*, pp. 163–166, 2003.
- [28] K. Kim, K. Kim, J. Kim, and H. Kim, “Learning-based approach for license plate recognition,” in *Neural Networks for Signal Processing X. Proceedings of the 2000 IEEE Signal Processing Society Workshop (Cat. No.00TH8501)*, vol. 2, pp. 614–623 vol.2, 2000.

- [29] R. Laroca, L. Zanlorensi, G. Gonçalves, E. Todt, W. Schwartz, and D. Menotti, “An efficient and layout-independent automatic license plate recognition system based on the yolo detector,” 02 2021.
- [30] H. C. Quang, T. D. Thanh, and C. T. Van, “Character time-series matching for robust license plate recognition,” in *2022 International Conference on Multimedia Analysis and Pattern Recognition (MAPR)*, p. 1–6, IEEE, Oct. 2022.
- [31] Y. Wang, Y. Liao, J. Yang, H. Wang, Y. Zhao, C. Zhang, B. Xiao, F. Xu, Y. Gao, M. Xu, and J. Zheng, “An fpga-based online reconfigurable cnn edge computing device for object detection,” *Microelectronics Journal*, vol. 137, p. 105805, 2023.
- [32] D. T. Nguyen, “A high-throughput and power-efficient fpga implementation of yolo cnn for object detection,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, 08 2019.
- [33] A. Anupreetham, M. Ibrahim, M. Hall, A. Boutros, A. Kuzhively, A. Mohanty, E. Nurvitadhi, V. Betz, Y. Cao, and J.-S. Seo, “High throughput fpga-based object detection via algorithm-hardware co-design,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 17, Jan. 2024.
- [34] D. K. Lam, C. V. Du, and H. L. Pham, “Quantlanenet: A 640-fps and 34-gops/w fpga-based cnn accelerator for lane detection,” *Sensors*, vol. 23, no. 15, 2023.