**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN – ĐIỆN TỬ

**BỘ MÔN ĐIỆN TỬ**

---------------o0o---------------

****

**ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC**

**ĐÈN GIAO THÔNG CÓ PHÁT HIỆN KẸT XE TẠI GIAO LỘ BẰNG MÔ HÌNH MÁY HỌC TRÊN FPGA**

**GVHD: ThS. Trần Hoàng Quân**

**SVTH: Nguyễn Thanh Toàn**

**MSSV: 2014777**

**TP. HỒ CHÍ MINH, THÁNG 06 NĂM 2025**

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

TRƯỜNG ĐẠI HỌC BÁCH KHOA Độc lập – Tự do – Hạnh phúc.

-----✩----- -----✩-----

Số: \_\_\_\_\_\_ /BKĐT

Khoa: **Điện – Điện tử**

Bộ Môn: **Điện Tử**

N**HIỆM VỤ LUẬN VĂN TỐT NGHIỆP**

1. HỌ VÀ TÊN : NGUYỄN THANH TOÀN MSSV: 2014777

1. NGÀNH: **ĐIỆN TỬ - VIỄN THÔNG** LỚP : DD20DV1
2. Đề tài: Đèn giao thông có phát hiện kẹt xe tại giao lộ bằng mô hình máy học trên FPGA
3. Nhiệm vụ (Yêu cầu về nội dung và số liệu ban đầu):

- Thiết kế mô hình mobile net có khả năng phân biệt kẹt xe và không kẹt xe từ bức ảnh đầu vào.

- Nghiên cứu thực hiện mô hình mobile net trên FPGA, cụ thế kit DE0-nano.

.............................................................................................................................................

.............................................................................................................................................

.............................................................................................................................................

.............................................................................................................................................

1. Ngày giao nhiệm vụ luận văn: 10/02/2025.
2. Ngày hoàn thành nhiệm vụ: 26/05/2025.
3. Họ và tên người hướng dẫn: Phần hướng dẫn

**ThS. Trần Hoàng Quân** .....................................

................................................................. .....................................

Nội dung và yêu cầu LVTN đã được thông qua Bộ Môn.

*Tp.HCM, ngày…... tháng….. năm 20*

**CHỦ NHIỆM BỘ MÔN NGƯỜI HƯỚNG DẪN CHÍNH**

**PHẦN DÀNH CHO KHOA, BỘ MÔN:**

Người duyệt (chấm sơ bộ):.......................

Đơn vị:......................................................

Ngày bảo vệ : ...........................................

Điểm tổng kết: .........................................

Nơi lưu trữ luận văn: ...............................

***LỜI CẢM ƠN***

***Lời đầu tiên, em xin trân trọng cảm ơn giảng viên thầy Trần Hoàng Quân - người đã trực tiếp hướng dẫn em trong quá trình hoàn thành đồ án. Em cũng xin được gửi lời cảm ơn đến quý thầy, cô giáo trường Đại học Bách Khoa TP. Hồ Chí Minh, đặc biệt là các thầy, cô khoa Kỹ thuật Điện tử, những người đã truyền lửa và giảng dạy kiến thức cho em suốt thời gian qua. Em đã cố gắng vận dụng những kiến thức đã học được và tìm tòi thêm nhiều thông tin để hoàn thành đồ án. Tuy nhiên, do kiến thức còn hạn chế và không có nhiều kinh nghiệm trên thực tiễn nên khó tránh khỏi những thiếu sót trong bài làm. Rất kính mong quý thầy, cô cho em thêm những góp ý để kết quả của em được hoàn thiện hơn. Cuối cùng, em xin cảm ơn gia đình, bạn bè đã luôn chia sẻ, ủng hộ, động viên và giúp đỡ trong suốt quá trình học tập của bản thân. Em xin trân trọng cảm ơn!***

*Tp. Hồ Chí Minh, ngày 25 tháng 03 năm 2025*

**Sinh viên**

**TÓM TẮT LUẬN VĂN**

Trong bối cảnh phát triển nhanh chóng của các công nghệ xử lý tín hiệu và trí tuệ nhân tạo, việc ứng dụng các mô hình học sâu vào các hệ thống nhúng đang trở thành xu hướng thiết yếu trong các ứng dụng thực tiễn. Luận văn này trình bày quá trình nghiên cứu và triển khai mô hình MobileNet trên nền tảng FPGA với bộ kit DE0-Nano, hướng đến giải pháp phát hiện kẹt xe tại các vị trí giao lộ – một vấn đề đang được quan tâm trong quản lý giao thông đô thị.

Phần nghiên cứu tập trung vào việc thực hiện mô hình MobileNet phù hợp với kiến trúc FPGA, qua đó đảm bảo khả năng thực thi nhanh chóng và tiêu thụ điện năng thấp. Quá trình chuyển đổi mô hình từ môi trường phần mềm sang phần cứng được thực hiện thông qua các kỹ thuật giảm độ chính xác (quantization) và tối ưu hóa cấu trúc mạch số, cố gắng thực hiện mô hình tính toán trên nguồn tài nguyên hạn chế.

Kết quả khẳng định tiềm năng ứng dụng của FPGA trong việc triển khai các mô hình học sâu cho các bài toán nhận dạng trong thời gian thực, đồng thời mở ra hướng phát triển mới cho các giải pháp thông minh trong lĩnh vực quản lý giao thông. Luận văn không chỉ góp phần đẩy mạnh ứng dụng công nghệ hiện đại vào thực tiễn mà còn là cơ sở nghiên cứu cho các hệ thống nhúng hiệu năng cao trong tương lai.

Luận văn này trình bày về tổng quan lý thuyết về máy học (các mô hình dự đoán, phân loại hình ảnh, đặc biệt là mô hình MobileNet phù hợp cho những thiết bị phần cứng có nguồn tài nguyên hạn chế). Chi tiết về xây dựng bộ dữ liệu, xây dựng mô hình, chạy thử nghiệm mô hình trên máy tính. Chi tiết về thiết kế phần cứng hệ thống trên FPGA để thực hiện được mô hình trên FPGA.

**MỤC LỤC**

[**1. GIỚI THIỆU 1**](#_Toc199185082)

[**1.1 Tổng quan 1**](#_Toc199185083)

[**1.2 Tình hình nghiên cứu trong và ngoài nước 1**](#_Toc199185084)

[**1.3 Nhiệm vụ luận văn 2**](#_Toc199185085)

[**2. LÝ THUYẾT VỀ MÔ HÌNH PHÂN LOẠI ẢNH. 4**](#_Toc199185086)

[**2.1 Tìm hiểu về mô hình CNN 4**](#_Toc199185087)

[**2.2 Những hạn chế của mô hình CNN 6**](#_Toc199185088)

[**2.3 Giới thiệu mô hình Mobile Net 7**](#_Toc199185089)

[**3. CHUẨN BỊ DỮ LIỆU 10**](#_Toc199185090)

[**3.1 Bộ dữ liệu tranning và testing 10**](#_Toc199185091)

[**3.2 Thêm dữ liệu cho testing phù hợp với giao thông Việt Nam 11**](#_Toc199185092)

[**4. THIẾT KẾ VÀ XÂY DỰNG MÔ HÌNH TRÊN MÁY TÍNH. 15**](#_Toc199185093)

[**5. KẾT QUẢ THỰC HIỆN MÔ HÌNH TRÊN MÁY TÍNH. 22**](#_Toc199185094)

[**6. THỰC HIỆN PHẦN CỨNG 24**](#_Toc199185095)

[**6.1 Kiến trúc tổng quan 24**](#_Toc199185096)

[**6.2 Khối Camera control 24**](#_Toc199185097)

[**6.2.1 Camera control read 25**](#_Toc199185098)

[**6.2.2 Control write to FIFO 26**](#_Toc199185099)

[**6.2.3 control store data to sdram 27**](#_Toc199185100)

[**6.3 Khối main mobile net 30**](#_Toc199185101)

[**6.3.2 Ram nguồn (source ram) và Ram đích (destination ram) 33**](#_Toc199185102)

[**6.3.4 Khối thuật toán Average 42**](#_Toc199185103)

[**6.3.5 Write back destiantion ram to SDRAM 46**](#_Toc199185104)

[**6.4 Module master control 48**](#_Toc199185105)

[**7. VERIFICATION 54**](#_Toc199185106)

[**7.1 Kiểm tra khối camera read và lưu dữ liệu vào FIFO 54**](#_Toc199185107)

[**7.2 Testbench module fetching memory 58**](#_Toc199185108)

[**7.3 Testbench module write back 59**](#_Toc199185109)

[**7.5 Testbench module master control. 62**](#_Toc199185110)

[**8. SYNTHESIS 67**](#_Toc199185111)

[**8.1 Tìm hiều về kít DE0-Nano 67**](#_Toc199185112)

[**8.2 Synthesis project 68**](#_Toc199185113)

[**9. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 74**](#_Toc199185114)

[**9.1 Kết luận 74**](#_Toc199185115)

[**9.2 Hướng phát triển 74**](#_Toc199185116)

[**10. TÀI LIỆU THAM KHẢO 75**](#_Toc199185117)

[**11. PHỤ LỤC 76**](#_Toc199185118)

**DANH SÁCH HÌNH MINH HỌA**

[Hình 1: Minh họa lớp tích chập của mô hình CNN 4](#_Toc199683376)

[Hình 2: Minh họa lớp max pooling thường được sử nhầm giảm kích thước trong mô hình CNN 5](#_Toc199683377)

[Hình 3: Minh họa lớp liên kết đầy đủ (Fully connected) trong mô hình CNN 6](#_Toc199683378)

[Hình 4: Minh họa kiến trúc, tính toán trọng số của mô hình CNN truyền thống 8](#_Toc199683379)

[Hình 5: Minh họa kiến trúc Depthwise Separable Convolution 9](#_Toc199683380)

[Hình 6: Minh họa tổ chức dữ liệu của bộ dataset Mendely 10](#_Toc199683381)

[Hình 7: Minh họa bộ lable của dataset 11](#_Toc199683382)

[Hình 8: Minh họa ứng dụng camera giao thông Việt Nam 12](#_Toc199683383)

[Hình 9: Hình ảnh "thô" chưa qua xữ lý cho bộ data dùng để kiểm thử mô hình trên giao thông Việt Nam 12](#_Toc199683384)

[Hình 10: Lưu đồ xữ lý hình ảnh đầu vào. 13](#_Toc199683385)

[Hình 11: Hình ảnh minh họa dữ liệu hình ảnh kiểm thử sau khi đã xữ lý. 13](#_Toc199683386)

[Hình 12: Minh họa lớp tích chập 2D. 16](#_Toc199683387)

[Hình 13: Minh họa hàm ReLU 16](#_Toc199683388)

[Hình 14: Minh họa lớp sử dụng kiến trúc depthwise 17](#_Toc199683389)

[Hình 15: Hình ảnh minh họa lớp global average pooling 2D.` 17](#_Toc199683390)

[Hình 16: Lưu đồ xây dựng và kiểm thử mô hình trên máy tính 18](#_Toc199683391)

[Hình 17: Kiến trúc mô hình sau khi xây dựng trên mô hình máy tính. 19](#_Toc199683392)

[Hình 18: Kiến trúc thực tế của mô hình. 20](#_Toc199683393)

[Hình 19: Kết quả huẩn luyện mô hình trên máy tính. 22](#_Toc199683394)

[Hình 20: Kiểm thử mô hình trên ảnh của giao thông Việt Nam 1 23](#_Toc199683395)

[Hình 21: Kiểm thử mô hình với ảnh giao thông Việt Nam 2 23](#_Toc199683396)

[Hình 22: Sơ đồ khối camera control 25](#_Toc199683397)

[Hình 23: Sơ đồ mô hình Mobile Net 31](#_Toc199683398)

[Hình 24: Kiến trúc module conv 35](#_Toc199683399)

[Hình 25: Minh họa thêm module chức năng padding cho module convolution 38](#_Toc199683400)

[Hình 26: Minh họa khối control convolution giao tiếp với khối convolution 40](#_Toc199683401)

[Hình 27: Kiến trúc khối thực hiện chức năng tính average 45](#_Toc199683402)

[Hình 28: Kiến trúc phần tính average bao gồm luôn phần fullyconnected và drive led 46](#_Toc199683403)

[Hình 29: Kiến trúc khối chức năng write back 47](#_Toc199683404)

[Hình 30: Minh họa giao tiếp module master control và các module control tiến trình chức năng thành phần 49](#_Toc199683405)

[Hình 31: Ánh xạ địa chỉ feature 1 52](#_Toc199683406)

[Hình 32: Ánh xạ địa chỉ feature 2 53](#_Toc199683407)

[Hình 33: Mô phỏng waveform của camera Ov7670 54](#_Toc199683408)

[Hình 34: Chạy testbench cho khối camera read 55](#_Toc199683409)

[Hình 35: Testbench cho module control write to FIFO case 1 56](#_Toc199683410)

[Hình 36: Testbench cho module control write to FIFO case2 57](#_Toc199683411)

[Hình 37: Testbench cho module control write to FIFO case 3 57](#_Toc199683412)

[Hình 38: Waveform của bộ fetching control 58](#_Toc199683413)

[Hình 39: Testbench cho module fetching memory. 59](#_Toc199683414)

[Hình 40: Waveform của bộ control write back 60](#_Toc199683415)

[Hình 41: Testbench cho module Write back 60](#_Toc199683416)

[Hình 42: Kết quả so sánh phần mềm dùng floating point và phần cứng dùng fixing point 61](#_Toc199683417)

[Hình 43: Hình ảnh thực tế Kit DE0-nano 67](#_Toc199683418)

[Hình 44: Kết quả synthesis module RAM 69](#_Toc199683419)

[Hình 45: Kết quả synthesis khối convolution 70](#_Toc199683420)

[Hình 46: Kết quả synthesis khối aveage register 70](#_Toc199683421)

[Hình 47: Kết quả systhesis module weight generate 71](#_Toc199683422)

[Hình 48: Kết quả synthesis module master control 72](#_Toc199683423)

[Hình 49: Kết quả synthesis và tài nguyên sử dụng của module top 73](#_Toc199683424)

[Hình 50: Netlist viewer của module top được tạo ra bởi quartus 73](#_Toc199683425)

**DANH SÁCH BẢNG SỐ LIỆU**

[Bảng 1: Bảng chuyển trạng thái khối cameraRead 26](#_Toc196441964)

[Bảng 2: Bảng chuyển trạng thái khối control write to FIFO 27](#_Toc196441965)

[Bảng 3: Bảng ngõ ra máy trạng thái khối control write to FIFO 27](#_Toc196441966)

[Bảng 4: Bảng mô tả danh sách ngõ vào khối control read cam to SDRAM 27](#_Toc196441967)

[Bảng 5: Bảng mô tả danh sách ngõ ra khối control read cam to SDRAM 28](#_Toc196441968)

[Bảng 6: Bảng chuyển trạng thái khối control read cam to sdram 28](#_Toc196441969)

[Bảng 7: Bảng ngõ ra máy trạng thái khối control read camera to SDRAM 29](#_Toc196441970)

[Bảng 8: Danh sách input port của module memory fetching 31](#_Toc196441971)

[Bảng 9: Danh sách output port của module fetching memory 32](#_Toc196441972)

[Bảng 10: Bảng chuyển trạng thái module memory fetching 32](#_Toc196441973)

[Bảng 11: Danh sách input output port của khối điều khiển RAM. 34](#_Toc196441974)

[Bảng 12: Danh sách input/output module convCore 36](#_Toc196441975)

[Bảng 13: Danh sách input output module padding control 38](#_Toc196441976)

[Bảng 14: Danh sách input output port module convolution control 40](#_Toc196441977)

[Bảng 15: Bảng chuyển trạng thái khối convolution control 41](#_Toc196441978)

[Bảng 16: Bảng output của máy trạng thái module convolution control 41](#_Toc196441979)

[Bảng 17: Dánh sách input output của module average 42](#_Toc196441980)

[Bảng 18: Danh sách input output của module averageControl 43](#_Toc196441981)

[Bảng 19: Bảng chuyển trạng thái module average control 43](#_Toc196441982)

[Bảng 20: Danh sách input, output của module write back control 47](#_Toc196441983)

[Bảng 21: Bảng chuyển trạng thái module write back control 48](#_Toc196441984)

[Bảng 22: Danh sách input output của module master control 49](#_Toc196441985)

[Bảng 23: Bảng chuyển trạng thái master control 50](#_Toc196441986)

[Bảng 24: Thông số tài nguyên của Altera Cyclone IV EP4CE22F17C6N 67](#_Toc196441987)

[Bảng 25: Thông số kỹ thuật SDRAM trên kit DE0-Nano 68](#_Toc196441988)

# GIỚI THIỆU

## Tổng quan

Trong những năm gần đây, với sự bùng nổ của công nghệ trí tuệ nhân tạo và học sâu (deep learning), việc ứng dụng các mô hình mạng nơ-ron tích chập (CNN) vào các hệ thống nhúng đã mở ra nhiều hướng nghiên cứu mới, đặc biệt là trong lĩnh vực xử lý ảnh và nhận dạng đối tượng. MobileNet, với thiết kế nhẹ và tối ưu cho các thiết bị có tài nguyên hạn chế, là một trong những mô hình CNN phổ biến, cho phép triển khai trên các nền tảng nhúng như FPGA.

FPGA (Field Programmable Gate Array) mang lại những ưu điểm vượt trội như khả năng xử lý song song, tốc độ thực thi cao và tiêu thụ năng lượng thấp, điều này rất phù hợp cho các ứng dụng thời gian thực như nhận diện tình trạng giao thông và phát hiện kẹt xe tại các giao lộ. Tuy nhiên, việc chuyển đổi một mô hình được huấn luyện trên môi trường phần mềm sang một nền tảng phần cứng như FPGA đòi hỏi phải tiến hành các bước tối ưu hóa như giảm độ chính xác (quantization), tối ưu hóa cấu trúc mạch số và đảm bảo tính ổn định của hệ thống trong điều kiện hoạt động thực tế.

Trong bối cảnh đô thị hiện nay, vấn đề kẹt xe không chỉ gây ảnh hưởng đến hiệu quả giao thông mà còn tác động đến môi trường và chất lượng cuộc sống. Việc phát triển một hệ thống nhận diện giao thông dựa trên FPGA với mô hình MobileNet không chỉ giúp cải thiện hiệu quả giám sát và quản lý giao thông mà còn mở ra hướng nghiên cứu ứng dụng trí tuệ nhân tạo vào các bài toán nhúng thời gian thực.

## Tình hình nghiên cứu trong và ngoài nước

Hiện tại, có dự án đã thành công triển khai mô hình MobileNet trên FPGA. Trong đó, nổi bật là việc nhận diện chữ viết tay trên FPGA sử dụng camera OV7670, nhúng MobileNet vào kit DE0-Nano để thực hiện tính toán và dùng màn hình TFT để hiển thị kết quả. Hệ thống hoạt động theo nguyên tắc xây dựng khối xử lý tính toán bên trong FPGA, thực hiện tính toán trực tiếp trên bộ nhớ RAM (do mô hình nhỏ với dữ liệu đầu vào 28x28x1). Dự án này đã thành công trong việc nhận diện chữ viết bằng mô hình MobileNet trên FPGA.

Nguyên tắc cơ bản của dự án bao gồm: Thực hiện mô hình bằng phần mềm máy tính, huấn luyện mô hình trên máy tính để thu được bộ trọng số, sau đó sử dụng bộ trọng số này để triển khai MobileNet trên FPGA giúp hệ thống hoạt động. Vì vậy, hệ thống trên FPGA sẽ không có khả năng học (training) mà chỉ sử dụng trọng số sẵn có từ mô hình trên máy tính để tính toán và đưa ra kết quả. Điều này phù hợp vì hệ thống nhúng có tài nguyên hạn chế, việc tích hợp khả năng học (training) sẽ làm tăng độ phức tạp và tiêu tốn nhiều tài nguyên phần cứng. Đa phần ứng dụng không yêu cầu hệ thống học thêm, mà chỉ cần dựa trên dữ liệu đầu vào để tính toán và đưa ra kết quả từ bộ trọng số đã được huấn luyện (trained) trên máy tính – nơi có bộ nhớ dồi dào và dễ lập trình nhờ điều khiển bằng phần mềm.

## Nhiệm vụ luận văn

Nội dung 1: Tìm hiểu lý thuyết về mô hình học máy trên máy tính, đặc biệt là mô hình MobileNet (Mô hình tối ưu sử dụng trên hệ thống có tài nguyên hạn chế).

Đã có rất nhiều bài nghiên cứu về mô hình máy học để nhận diện hoặc phân loại hình ảnh. Ví dụ như mô hình CNN, tuy nhiên mô hình CNN thuần túy đòi hỏi khá lớn về tài nguyên tính toán và tài nguyên bộ nhớ cần sử dụng. Vì vậy, ta sẽ tập trung chủ yếu vào mô hình MobileNet.

Nội dung 2: Chuẩn bị dataset, xây dựng mô hình trên máy tính, và tiến hành training mô hình để được bộ trọng số.

Bởi vì nhiệm vụ của mô hình là phân loại tình trạng giao thông trong một bức ảnh có bị tắc nghẽn hay không. Vì vậy, nguồn dữ liệu ta có thể cân nhắc: Nguồn dữ liệu có sẵn trên internet, open camera giao thông được chiếu trực tiếp trên các nền tảng web hoặc ứng dụng. Xây dựng mô hình MobileNet trên máy tính, ta sẽ sử dụng thư viện TensorFlow bằng ngôn ngữ Python nhằm nhanh chóng và dễ dàng xây dựng mô hình hay training mô hình.

Nội dung 3: Nghiên cứu phần cứng FPGA được sử dụng – Kit DE0-Nano.

Tài liệu tham khảo chính dùng để hiểu và sử dụng Kit DE0-Nano sẽ là DE0-Nano User Manual. Tập trung chủ yếu vào việc tìm hiểu các ngoại vi nằm trên Kit DE0-Nano, tài nguyên trên Kit (SDRAM, Led, Button, ...), tài nguyên trên FPGA (LUT, Memory Block, DSP Block, ...).

Nội dung 4: Nghiên cứu thực thi mô hình MobileNet trên Kit đã lựa chọn.

Dựa vào thuật toán của mô hình MobileNet trên máy tính (computer vision), cố gắng thực hiện lại thuật toán đó trên FPGA thông qua sự phối hợp hoạt động của các khối chức năng. Sẽ được trình bày cụ thể trong phần 3: Thiết kế và thực hiện phần cứng.

# LÝ THUYẾT VỀ MÔ HÌNH PHÂN LOẠI ẢNH.

## Tìm hiểu về mô hình CNN

Mạng nơ-ron tích chập (CNN) là một trong những kiến trúc học sâu được sử dụng phổ biến trong xử lý ảnh và nhận dạng đối tượng. Những đặc điểm cơ bản của CNN bao gồm:

Tầng tích chập (Convolutional Layers): Đây là tầng cốt lõi của CNN, nơi mà các phép tích chập được thực hiện giữa bộ lọc (filter) hay các parameter (kernel) và một vùng nhỏ của ảnh đầu vào có kích thước thường tương ứng với kích thước của filter (kernel). Mỗi bộ lọc có khả năng trích xuất các đặc trưng (features) như cạnh, góc cạnh, kết cấu từ ảnh,... Qua đó, các đặc trưng này được học một cách tự động qua quá trình huấn luyện. Trong quá trình huấn luyện, trọng số của bộ lọc (filter) hay kernel sẽ được điều chỉnh nhằm thu được kết quả tốt nhất. Sau khi thực hiện xong trên một vùng ảnh, bộ lọc (filter) hay kernel này được dịch và di chuyển để thực hiện tích chập tiếp theo trên một vùng ảnh khác cho đến khi đã thực hiện trên toàn bộ bức ảnh hay dữ liệu đầu vào.A close-up of a grid

AI-generated content may be incorrect.

Hình 1: Minh họa lớp tích chập của mô hình CNN

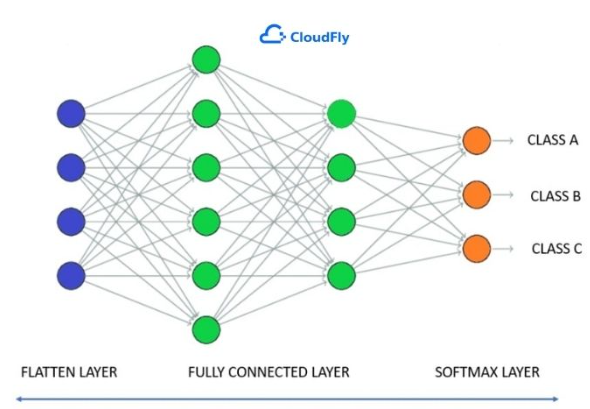
Tầng pooling (Pooling Layers): Lớp pooling có tác dụng giảm bớt số lượng tham số khi hình ảnh quá lớn. Không gian pooling còn được gọi là lấy mẫu con hoặc lấy mẫu xuống làm giảm kích thước của mỗi feature map nhưng vẫn giữ lại thông tin quan trọng. Một số loại pooling phổ biến bao gồm: Max Pooling, average pooling, sum pooling.[[1]](#footnote-1)

A diagram of a graph

AI-generated content may be incorrect.

Hình 2: Minh họa lớp max pooling thường được sử nhầm giảm kích thước trong mô hình CNN

Tầng kết nối đầy đủ (Fully Connected Layers): Sau các lớp tích chập và pooling, các đặc trưng được chuyển thành dạng vector và đưa vào tầng kết nối đầy đủ. Tại đây, mạng sẽ thực hiện các phép biến đổi tuyến tính và phi tuyến để đưa ra kết quả cuối cùng (phân loại, nhận dạng đối tượng, …).



Hình 3: Minh họa lớp liên kết đầy đủ (Fully connected) trong mô hình CNN

[[2]](#footnote-2)

Tính chất trích xuất đặc trưng tự động: CNN được thiết kế để tự động học các đặc trưng từ dữ liệu đầu vào thông qua quá trình huấn luyện, giảm thiểu sự phụ thuộc vào các đặc trưng thủ công và cải thiện khả năng tổng quát hóa của mô hình.

## Những hạn chế của mô hình CNN

Mặc dù các mô hình CNN truyền thống đã đạt được những hiệu quả rất tốt trong các bài toán nhận dạng ảnh, tuy nhiên mô hình CNN truyền thống lại gặp một số hạn chế khi triển khai trên các thiết bị có tài nguyên hạn chế như:

Số lượng tham số lớn: Tuy so với mô hình neuron network truyền thống (neural network) thì mô hình CNN sử dụng các filter đã giảm được rất nhiều các tham số, tuy nhiên khi ảnh đầu vào là 3 chiều, các đặc trưng được trích xuất nhiều hơn thì các mô hình CNN truyền thống này vẫn có số lượng tham số khổng lồ, dẫn đến yêu cầu bộ nhớ và khã năng xữ lý tính toán khá cao. Vì vậy mô hình CNN truyền thống có yêu cầu lớn về tài nguyên bộ nhớ (Lưu trữ trọng số của các filter, trọng số của lớp fully connected, chứa dữ liệu hình ảnh, chứa kết quả quả các lớp – các đặc trưng (các feature map) được trích xuất qua các fitler)

Tốc độ xử lý chậm: Vì có nhiều trọng số vì vậy mô hình CNN truyền thống cũng có số lượng phép tính lớn, việc triển khai trên các thiết bị nhúng (embedded systems) hoặc FPGA gặp khó khăn do hạn chế về tài nguyên tính toán và độ phức tạp của hệ thống khi thiết kế.

Tiêu thụ năng lượng cao: Với yêu cầu tài nguyên bộ nhớ và tài nguyên tính toán lớn, thực hiện nhiều phép toán cũng đồng nghĩa với các mô hình CNN truyền thống sẽ thường tiêu thụ nhiều năng lượng, không phù hợp với các ứng dụng thời gian thực cần tiết kiệm năng lượng.

* 1. **Giới thiệu mô hình Mobile Net**

Các khái niệm cơ bản về CNN đã cung cấp nền tảng vững chắc cho việc phát triển các kiến trúc tiên tiến như MobileNet. Các yếu tố quan trọng trong việc thiết kế MobileNet có thể được xem là sự tinh chỉnh và tối ưu hóa dựa trên các nguyên tắc cơ bản của mô hình CNN:

Từ việc trích xuất đặc trưng thông qua các tầng tích chập truyền thống, MobileNet tối ưu bằng cách áp dụng depthwise separable convolutions nhằm giảm tài nguyên tính toán tiêu tốn và số lượng feature map đầu ra.

Các tầng pooling và fully connected vẫn giữ vai trò quan trọng nhưng được thiết kế lại để phù hợp với khả năng xử lý song song và giới hạn về tài nguyên của các thiết bị nhúng. MobileNet không chỉ giảm thiểu số lượng tham số mà còn tối ưu hóa kiến trúc để đảm bảo rằng hệ thống có thể được thực hiện trên FPGA với độ phức tạp của hệ thống có thể chấp nhận được. Đáp ứng được yêu cầu về tài nguyên tính toàn và tài nguyên bộ nhớ của FPGA.

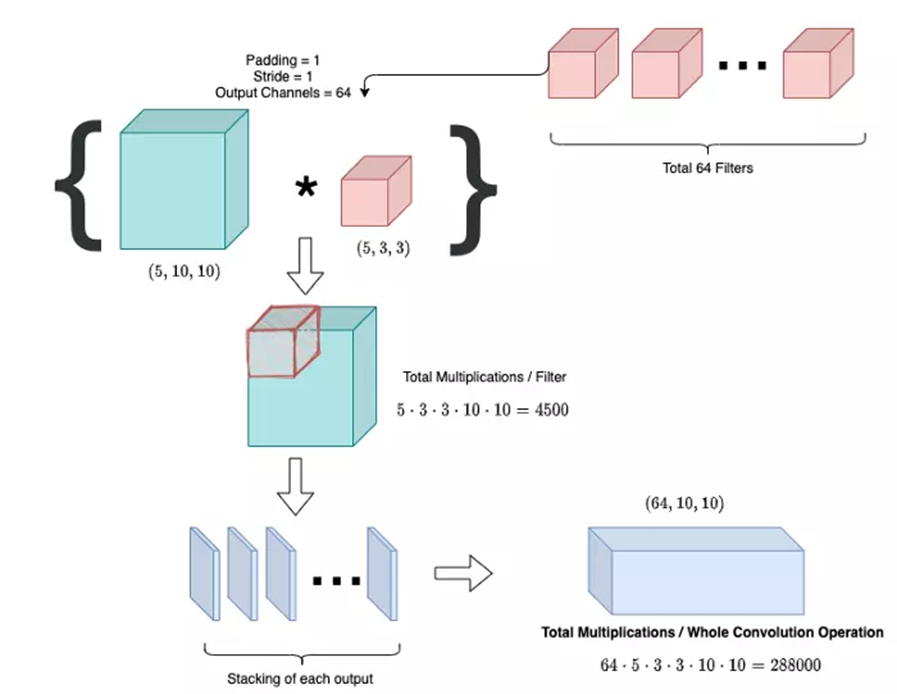
Kiến trúc Depthwise Separable Convolution: MobileNet thay thế các phép tích chập chuẩn bằng phép tích chập phân tách theo chiều sâu (depthwise separable convolution). Quá trình này được chia thành hai bước bao gồm:

Depthwise Convolution: Thực hiện phép tích chập riêng lẻ cho mỗi kênh của ảnh đầu vào, giúp trích xuất đặc trưng một cách riêng biệt.

Pointwise Convolution: Sau đó, kết hợp các kênh thông qua một phép tích chập 1x1 để tạo ra đầu ra mong muốn.

Việc tách nhỏ phép tính này giúp giảm số lượng phép toán và tham số xuống đáng kể mà vẫn duy trì được chất lượng trích xuất đặc trưng.

Để hiểu rõ hơn về kiến trúc Depthwise Separable Convolution của mô hình mobile net, Ta xem xét kiến trúc mô hình ví dụ sau:



Hình 4: Minh họa kiến trúc, tính toán trọng số của mô hình CNN truyền thống

Đây là ví dụ về kiến trúc của lớp tích chập trong mô hình CNN truyền thống. Input có kích thước là 10x10x5 và output có kích thước là 10x10x64. Như tính toán đã được trình bày trong ảnh. Ta cần tổng cộng 64 filter kích thước 3x3x5. Như vậy tổng số trọng số là: 64x3x3x5 = 2880 tham số. Tổng số phép tính cần thực hiện là: 64x3x3x5x10x10 = 288000 phép tính. Chưa nói đến các phép cộng liên quan.

ĐâyA diagram of a diagram of a complex structure

AI-generated content may be incorrect. là ví dụ về kiến trúc Depthwise Seprable Convolution được sử dụng trong mô hình mobile net, tương tự như ví dụ trong convolution truyền thống được sử dụng trong mô hình CNN ta có input có kích thước là 10x10x5 và output mô hình là 64x10x10. Tổng số tham số của kiến trúc là: 5x3x3 + 64x5 = 365 (tham số). Tổng số phép tính: 5x3x3x10x10 + 64x5x10x10 = 36500 (phép nhân)

Hình 5: Minh họa kiến trúc Depthwise Separable Convolution

# CHUẨN BỊ DỮ LIỆU

* 1. **Bộ dữ liệu tranning và testing**

Bởi vì nhiệm vụ của đề tài là phân loại tình hình giao thông trong một bức ảnh có hiện tượng tắc nghẽn hay không ? Ta cần có nguồn dữ liệu đáp ứng yêu cầu sau: Đầu vào định dạng hình ảnh (format ảnh phổ biến như .jpg, .png,... để dễ cho việc xử lý), Ngoài ra cần có nhãn đánh dấu ảnh đầu vào là “Kẹt xe” hay “Không có kẹt xe”, hoặc bộ dữ liệu được phân loại sẵn thành 2 thư mục riêng biệt.

Ở đây, ta sử dụng bộ dataset của Mendeley data[[3]](#footnote-3). “Mục đích chính của tập dữ liệu này là cho phép phát hiện tình trạng tắc nghẽn giao thông từ camera giám sát bằng cách sử dụng các bộ phát hiện vật thể một giai đoạn. Tập dữ liệu chứa các cảnh giao thông tắc nghẽn và không tắc nghẽn với các nhãn tương ứng của chúng. Tập dữ liệu này được thu thập từ các cảnh quay video của các camera giám sát khác nhau. Để chuẩn bị cho tập dữ liệu, các khung hình được trích xuất từ ​​các nguồn video và được thay đổi kích thước thành 500 x 500 với định dạng hình ảnh .jpg. Định dạng của nhãn phân loại file text định dạng .txt có cùng tên với hình ảnh. Tập dữ liệu chủ yếu được chuẩn bị cho các mô hình YOLO, nhưng có thể được chuyển đổi sang dùng cho dạng mô hình khác.”[[4]](#footnote-4)

A group of papers with cars in the road

AI-generated content may be incorrect.

Hình 6: Minh họa tổ chức dữ liệu của bộ dataset Mendely

Như vậy, bộ dữ liệu được tổ chức dạng riêng lẽ thành những tấm ảnh, định dạng .jpg, Mỗi ảnh sẽ đi kèm với một tập tin .txt có cùng tên với ảnh chưa thông tin về lable của ảnh.

A close-up of a number

AI-generated content may be incorrect.

Hình 7: Minh họa bộ lable của dataset

Mặc dù tập dữ liệu được thiết kế dành cho mô hình Yolo. Tuy nhiên, trong phần lable của một ảnh ta có thông số đầu tiên 1 hoặc 0, biểu thị cho việc có xuất hiện trình trạng kẹt xe trong bức ảnh hay không. Như vậy ta có thể dùng bộ dataset này để tranning mô hình.

* 1. **Thêm dữ liệu cho testing phù hợp với giao thông Việt Nam**

Trong trường hợp muốn có độ chính xác cao nhất, phù hợp nhất cho tính chất giao thông ở Việt Nam thì ta nên xây dựng một bộ dataset cho giao thông trong nước. Tuy nhiên, do hạn chế về thời gian thực hiện đồ án nên ta chỉ có thể thu thập một số hình ảnh giao thông ở các tuyến đường của Việt Nam để kiểm thử mô hình có hoạt động ổn định và chính xác với tính chất của giao thông Việt Nam.

Ta có sử dụng ứng dụng xem trực tiếp giao thông tại Việt Nam để lấy dữ liệu hình ảnh. Ứng dụng “Camera Giao Thông Việt Nam”[[5]](#footnote-5) nhầm thu thập hình ảnh giao thông thực tế tại các tuyến đường ở Việt Nam.

A screenshot of a chat

AI-generated content may be incorrect.

Hình 8: Minh họa ứng dụng camera giao thông Việt Nam

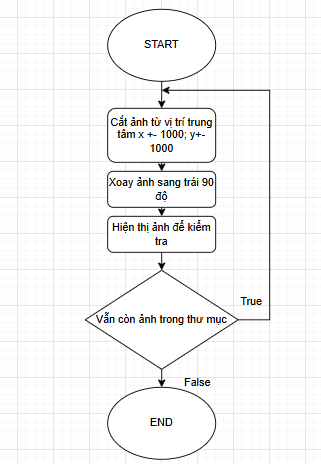
Chúng ta sẽ sử dụng hình ảnh từ camera giao thông để tạo hình ảnh để kiểm thử. Tuy nhiên, hình ảnh lấy được từ ứng dụng sẽ có tỉ lệ và kích thước không phù hợp với mô hình, hình ảnh bị lật sang phải 90 độ, có các thanh màu trắng của bộ cục ứng dụng.

A screenshot of a computer

AI-generated content may be incorrect.

Hình 9: Hình ảnh "thô" chưa qua xữ lý cho bộ data dùng để kiểm thử mô hình trên giao thông Việt Nam

Vì vậy, ta cần xử lý hình ảnh này để đưa về hình ảnh phù hợp hơn cho mô hình. Ta cần thực hiện các phép biến đổi nhầm đạt được các tính chất hình ảnh phù hợp với mô hình bao gồm: Lật ảnh sang trái 90 độ, cắt và scale hình ảnh thành kích thước 500px x 500px hoặc ít nhất biến ảnh thành tỉ lệ 1:1 và dùng hàm resize của openCV để chuyển ảnh về kích thước 500x500px, Để thực hiên được các biển đổi trên ta sử dụng thư viện openCV và script python



Hình 10: Lưu đồ xữ lý hình ảnh đầu vào.

Script python chi tiết tại phần phụ lục. Sau khi thực hiện giải thuật xữ lí hình ảnh đầu vào ta được bộ data kiểm thử như sau có kích thước ảnh 500x500px.

A screenshot of a computer

AI-generated content may be incorrect.

Hình 11: Hình ảnh minh họa dữ liệu hình ảnh kiểm thử sau khi đã xữ lý.

Như ta thấy ngoài việc chuyển ảnh về tỉ lệ, gốc độ phù hợp với mô hình thì script python còn tạo ra file .txt nhầm chưa lable của hình ảnh. Điều này tuy không trực tiếp sử dụng trong đồ án này tuy nhiên có thể là tiền đề (một công cụ hỗ trợ) để thành lập bộ data cho riêng giao thông Việt Nam nhầm nâng cao độ chính xác của mô hình khi dữ liệu đầu vào là hình ảnh giao thông Việt Nam

# THIẾT KẾ VÀ XÂY DỰNG MÔ HÌNH TRÊN MÁY TÍNH.

Kiến trúc TensorFlow hoạt động được chia thành 3 phần:

– Tiền xử lý dữ liệu

– Dựng model

– Train và ước tính model[[6]](#footnote-6)

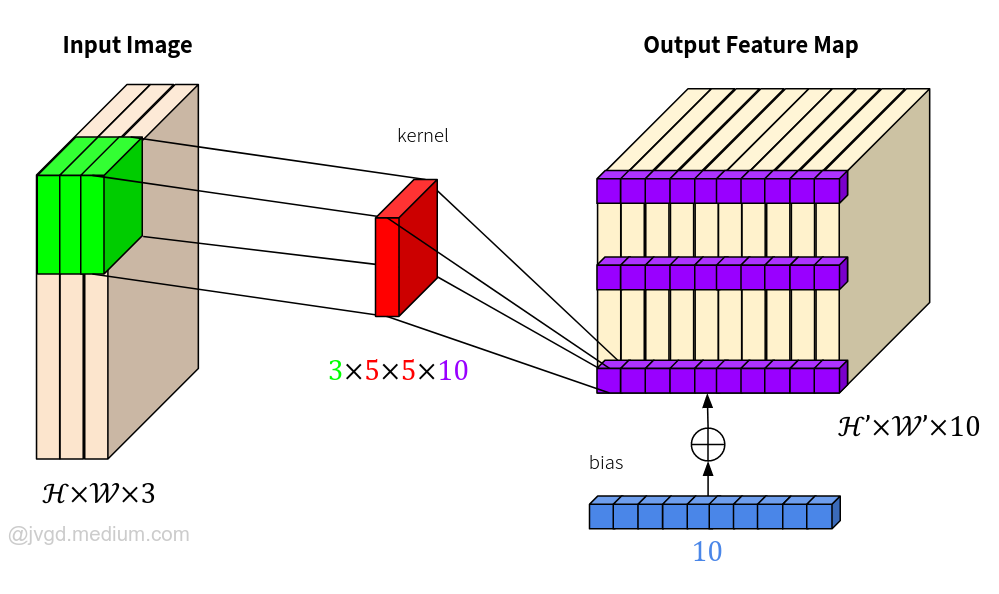
Trong đó tiền xử lý dữ liệu: Thư viện TensorFlow sẽ cung cấp các hàm, công cụ để biến đổi dữ liệu, làm sạch dữ liệu (scale, resize,..)

Dựng model: Thư viên TensorFlow cung cấp các câu lệnh để ta xây dựng mà ta có thể tùy chỉnh được các node, số lượng node có trong một lớp, số lượng lớp có trong mô hình. Thư viện TensorFlow cung cấp nhiều kiến trúc của các lớp khác nhau như: fully connected (Lớp liên kết đầy đủ), lớp Flatten (Lớp trải phẳng) – thường dùng khi chuyển đổi từ dạng 2D (một bức ảnh) sang 1D (dạng vector), Lớp Conv2D (Lớp tính tích chập truyền thống) – thường dùng trong mô hình CNN, Đặc biệt là lớp DepthwiseConv2D (Depthwise Seprable Convolution) – được dùng trong mô hình mobile net để tiết kiệm tài nguyên tính toán và tài nguyên bộ nhớ và cũng là lớp mà ta sẽ sử dụng chủ yếu trong mô hình mà ta sẽ thiết kế.

Các hàm cơ bản để thực hiện mô hình như sau:

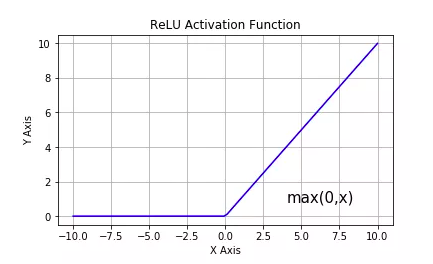
inputs = Input(shape=(64, 64,3)) : Tạo ra lớp input cho mô hình có kích thước 64x64x3 – bức ảnh có chiều dài 64px, chiều rộng là 64px, chiều sâu – kênh màu là 3 kênh (RGB)

x = Conv2D(16, kernel\_size=3, strides=1, padding='same', use\_bias=False)(inputs): Tạo ra lớp tích chập 2D nhầm trích xuất các đặc trưng của hình ảnh đầu vào. 16 là số filter được dùng đồng nghĩa với việc trích xuất 16 đặc trưng từ bức ảnh. Kernal\_size là kích thước của bộ lọc 3x3 tương ứng với mỗi bộ lọc có 9 tham số, strides độ dịch của bộ lọc sau mỗi lần tính (Dịch 1 pixel sau mỗi lần tích chập), padding = ‘same’ giữ nguyên kích thước của đặc trưng đầu ra so với ảnh đầu vào, use\_bias là hệ số bias thường dùng trong lớp fully connected, trong conv2d thường không sử dụng hệ số bias.

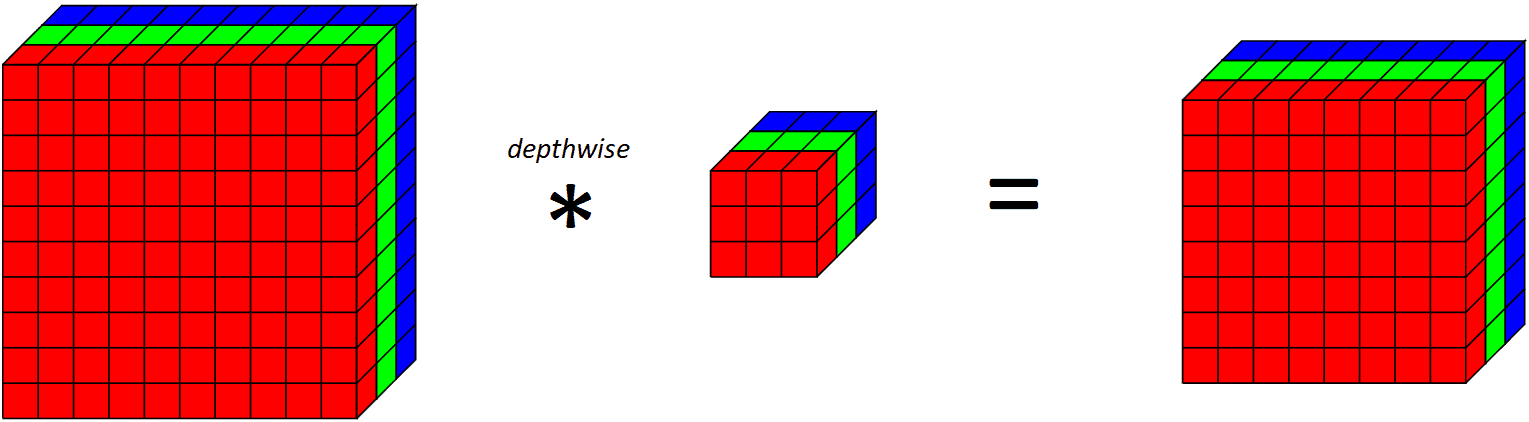


Hình 12: Minh họa lớp tích chập 2D.[[7]](#footnote-7)

x = ReLU()(x) Thực hiện hàm activation. Nhầm tạo ra hàm phi tuyến cho mô hình.

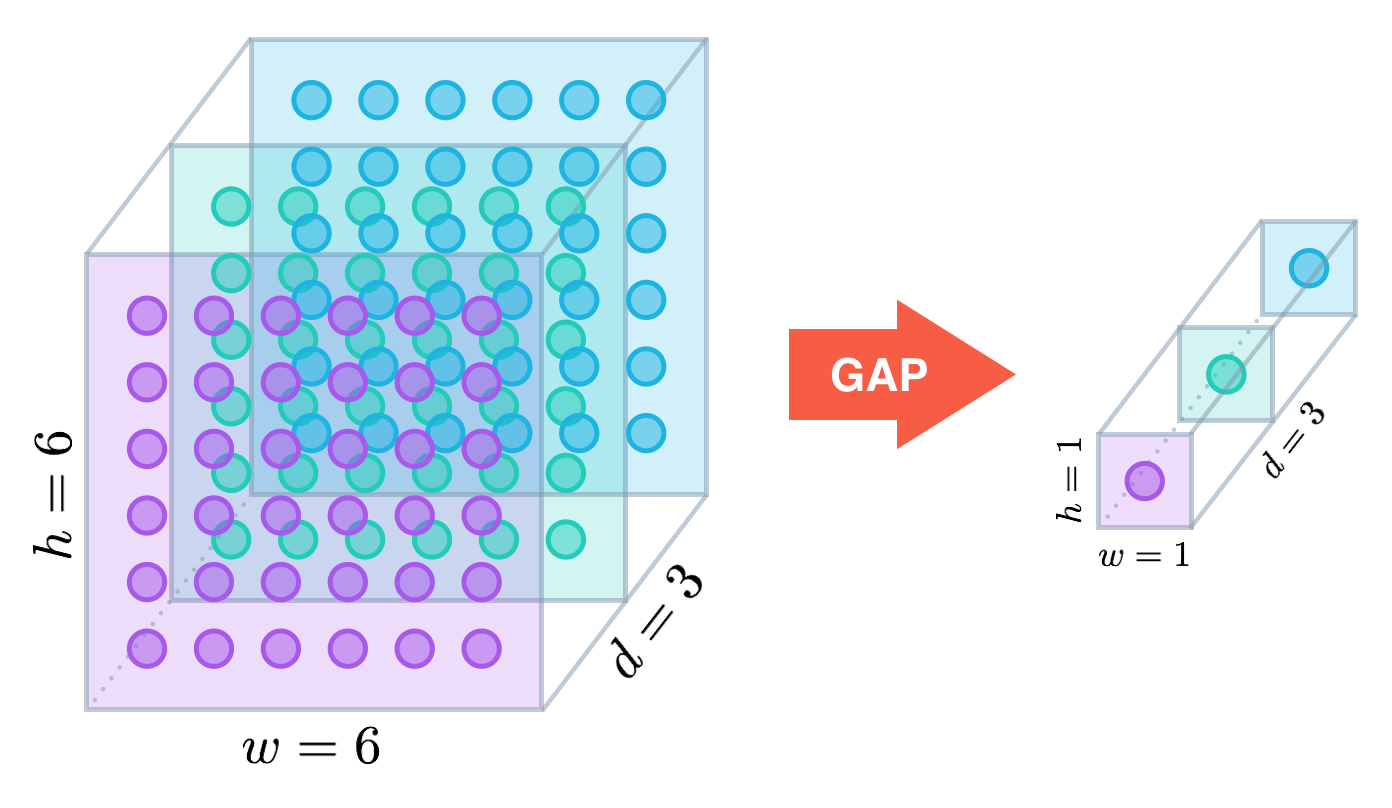


Hình 13: Minh họa hàm ReLU

x = DepthwiseConv2D(kernel\_size=3, strides=1, padding='same', use\_bias=False)(x): Tạo ra lớp tích chập với kiến trúc Depthwise. Tương tự như lớp conv2D: kernal\_size kích thước của bộ lọc, strides độ dịch chuyển của bộ lọc sau mỗi lần tích chập, padding = ‘same’ giữu nguyên kích thước của đầu ra so với đầu vào, trong lớp sử dụng kiến trúc Depthwise thường cũng không sử dụng hệ số bias, hệ số bias thường sẽ thường sử dụng trong lớp fully connected. 

Hình 14: Minh họa lớp sử dụng kiến trúc depthwise[[8]](#footnote-8)

x = GlobalAveragePooling2D()(x) Hàm của thư viện TensorFlow dùng để tạo ra lớp tính trung bình, nhầm giảm kích thước của dữ liệu giúp tiết kiệm tài nguyên tính toán và tài nguyên bộ nhớ.



Hình 15: Hình ảnh minh họa lớp global average pooling 2D.`

model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=25, batch\_size=64) Hàm của thư viện TensorFlow được dử dụng để trainning mô hình từ tập X\_train – chưa dữ liệu ảnh và tập Y\_train – chứa lable.

A diagram of a flowchart

AI-generated content may be incorrect.

Hình 16: Lưu đồ xây dựng và kiểm thử mô hình trên máy tính

Chi tiết script python để tạo lập mô hình:

|  |
| --- |
| # Define a simple lightweight model  inputs = Input(shape=(64, 64,3))  # Initial Conv Layer  x = Conv2D(16, kernel\_size=3, strides=1, padding='same', use\_bias=False)(inputs)  x = ReLU()(x)  # Depthwise Convolution Block 1  x = DepthwiseConv2D(kernel\_size=3, strides=1, padding='same', use\_bias=False)(x)  x = ReLU()(x)  # Depthwise Convolution Block 2  x = DepthwiseConv2D(kernel\_size=3, strides=1, padding='same', use\_bias=False)(x)  x = ReLU()(x)  # Depthwise Convolution Block 3  x = DepthwiseConv2D(kernel\_size=3, strides=1, padding='same', use\_bias=False)(x)  x = ReLU()(x)  # Global Average Pooling and Output  x = GlobalAveragePooling2D()(x)  outputs = Dense(1, activation='sigmoid')(x) # Binary output: Traffic Jam (1) or Not (0)  # Build model  model = Model(inputs, outputs) |

Kiến trúc của mô hình sau khi xây dựng như sau:

A screenshot of a computer program

AI-generated content may be incorrect.

Hình 17: Kiến trúc mô hình sau khi xây dựng trên mô hình máy tính.

Như vậy mô hình có kiến trúc 11 lớp: trong đó bao gồm 1 lớp input, 1 lớp output, 4 lớp activation ReLU.

Lớp 1: Input layer – Lớp này chịu trách nhiệm cho phần input của mô hình có kích thước 64x64x3 cũng chính là kích thước ảnh đầu vào của mô hình.

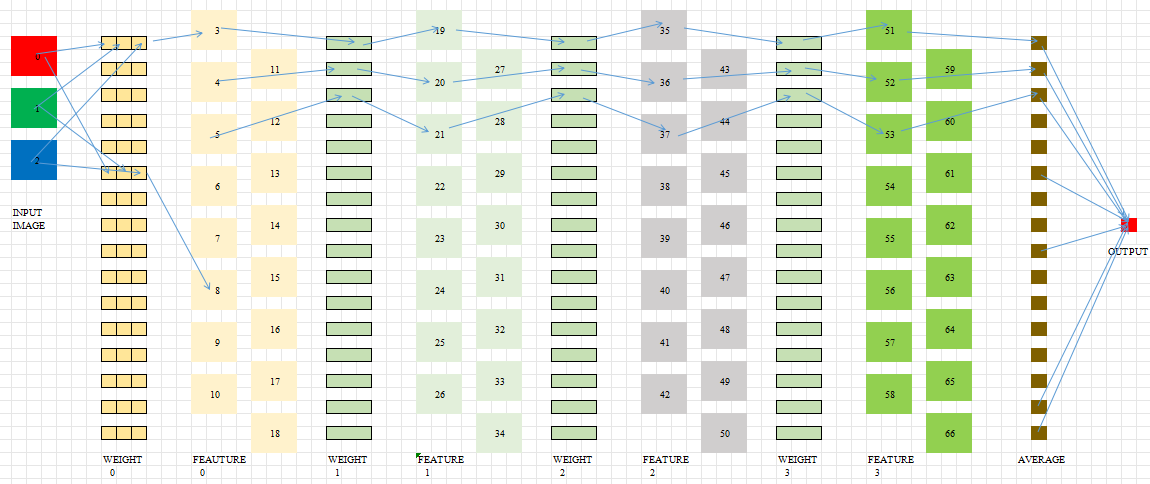
Lớp 2: Conv2D – Lớp này chịu trách nhiệm trích xuất các đặc trưng của hình ảnh đầu vào, vì ảnh đầu vào có chiều sâu là 3 kênh ảnh vì vậy lớp Conv2D sẽ phù hợp hơn tại vị trí này.

Lớp 3: ReLU – Lớp này đóng vai trò là lớp activation function, nhầm tạo ra tính phi tuyến cho mô hình. Sỡ di việc chọn activation funciton là ReLU vì đây là hàm rất đơn giản Max(y,0) nên sẽ dễ thiết kế và thực hiện hơn khi thực thi trên FPGA.

Lớp 4: Deepthwise\_conv2d – Lớp này đóng vai trò hidden layer, lớp theo kiến trúc Depthwise Seprable Convolution nhầm tiết kiệm tài nguyên tính toán và tài nguyên bộ nhớ.

Các lớp còn lại tương tự như các tương tự như các lớp phía trên.

Ở đây kích thước các đặc trưng của ảnh (các hidden layer) được giữa nguyên với mục đích nhầm đơn giản hóa mô hình. Vì khi các lớp tương tự nhau, các phép toán tương tự nhau sẽ tạo ra lợi thế khi thiết kế mô hình trên FPGA giúp đơn giản hóa thuật toán, đơn giản hóa các khối chức năng bằng cách cố gắng tương đồng hóa các lớp ẩn với nhau, với phương pháp này ta có thể thực hiện các lớp ẩn tương tự nhau giúp kiến trúc của các khối điều khiển, các khối chức năng được đơn giản hơn, không quá phức tạp góp phần khả thi hơn khi thiết kế trên FPGA.



Hình 18: Kiến trúc thực tế của mô hình.

Trong đó lớp đầu tiên (INPUT IMAGE) chính là 3 kênh màu của hình ảnh đầu vào.

WEIGHT0, WEIGHT1, WEIGHT2, WEIGHT3: Là các trọng số mô hình qua từng lớp. Ví dụ tích chấp của lớp Input Image và Weight0 ta sẽ được lớp FEATURE0 (Trích xuất đặc trưng của ảnh đầu vào).

FEATURE0, FEATURE1, FEATURE2, FEATURE3: Là các lớp chứa các đặc trưng của ảnh đầu vào. Ví dụ lớp Feature0 là kết quả của trích xuất đặc trưng của lớp input image với trọng số là lớp Weight0.

AVERAGE: Là lớp chứa giá trị trung bình của một Feauture. Đối mô hình CNN truyền thống ở lớp này thường là lớp Flatten (lớp trải phẳng) Tuy nhiên nếu trải phẳng sẽ sinh ra rất nhiều tham số cho phần fully connected. Vì vậy ở đây chúng ta dùng lớp trung bình nhầm tiết kiệm tài nguyên phần cứng.

Các quá trình (loại tính toán) bao gồm:

Từ INPUT IMAGE đến FEATURE0: Thực hiện thuật toán convolution 2D truyền thống.

Từ FEATURE0 đến FEATURE1, từ FEATURE1 đến FEATURE2, từ FEATURE2 đến FEATURE3: Thực hiện thuật toán depthwise convolution 2D nhầm tiết kiệm tài nguyên.

Từ FEAUTRE3 đến AVERAGE: thực hiện thuật toán global average polling 2D.

Từ AVERAGE đến OUTPUT: Thực hiện thuật toán fully connected (dense).

# KẾT QUẢ THỰC HIỆN MÔ HÌNH TRÊN MÁY TÍNH.

Kết quả của quá trình huấn luyện mô hình trên máy tính: Loss sẽ giảm dần qua những lần học, Độ chính xác của mô hình sẽ tăng lên sau mỗi lần học và đạt bão hòa vào khoảng 0.9 (90%) trên tập test.

A graph of loss and accuracy

AI-generated content may be incorrect.

Hình 19: Kết quả huẩn luyện mô hình trên máy tính.

Kết quả kiểm tra trên tập dữ liệu test của giao thông Việt Nam:

A screenshot of a video

AI-generated content may be incorrect.

Hình 20: Kiểm thử mô hình trên ảnh của giao thông Việt Nam 1

A group of cars and motorcycles in a busy street

AI-generated content may be incorrect.

Hình 21: Kiểm thử mô hình với ảnh giao thông Việt Nam 2

Ta có thể nhận thấy rằng độ chính xác của mô hình đã bị giảm, do nguồn dữ liệu của tập trainning không thật sự phù hợp với giao thông Việt Nam (tỉ lệ số lượng xe máy, xe ô tô, gốc độ của camera, ánh sáng, hình dạng cung đường, giao lộ). Tuy nhiên độ chính xác là 80% là vẫn chấp nhận được .

1. **THỰC HIỆN PHẦN CỨNG**
   1. **Kiến trúc tổng quan**

Hệ thống triển khai MobileNet trên FPGA được tổ chức thành ba khối xử lý dữ liệu chính và quan trọng nhất, nhầm thực hiện tuần tự các công việc từ thu nhận ảnh đầu vào xữ lý ảnh đầu vào qua các thực toán, đến xuất kết quả.

Phần 1: Camera control

Chức năng chính: Điều khiển đọc tín hiệu VSYNC, HREF, PCLK từ camera OV7670, lưu từng pixel vào FIFO. Xữ lý chia dữ liệu thô từ camera thành 3 kênh màu, điều khiển, điều phối thực hiện ghi dữ liệu từ FIFO vào SDRAM.

Phần 2: Main MobileNet

Chức năng chính: Đây là khối xử lý cốt lõi trong hệ thống mô hình MobileNet thực thi trên FPGA trong vấn đề thực thi tính toán. Chức năng chính của nó là thực hiện các hoạt động xử lý dữ liệu trung tâm của kiến ​​trúc MobileNet sau khi hình ảnh đầu vào được lấy từ camera và được lưu trữ trong SDRAM.

2.1 Fetching memory: Lấy dữ liệu từ SDRAM (các feature – đặc trưng) lưu vào source RAM, nhầm tăng tốc quá trình tính toán cho hệ thống.

2.2 Convolution algorithm: Lấy dữ liệu từ source RAM và weight generation thực hiện tính tích chập và lưu vào destination RAM.

2.3 Average algorithm: Lấy dữ liệu từ source RAM, thực hiện tính trung bình, lưu vào thanh ghi chứa giá trị trung bình.

2.4 Fully connected algotithm: Lấy dữ liệu từ thanh ghi chứa dữ liệu trung bình. Thực hiện phép toán tích chập với trọng số. Lưu giá trị vào drive và điều khiển LED báo kết quả.

2.5 Write back memory: Sau khi các khối tính toán xong và lưu vào RAM đích, khối này sẽ thực hiện chức năng lưu dữ liệu từ destination RAM về lại SDRAM nhầm giữ không gian cho việc tính toán feauture tiếp theo.

Phần 3: MASTER Control:

Chức năng chính: Đây là khối có vai tró quan trọng cốt lõi trong việc điều khiển, điều hòa, điều tiếp hoạt động của các nguồn tài nguyên như SDRAM, camera, source RAM, destination RAM,...

* 1. **Khối Camera control**

Phần này có chức năng nhận tính hiệu điều khiển từ bộ master control. Nhận dữ liệu thô từ camera lưu vào FIFO. Xữ lý dữ liệu thô từ FIFO thành dữ liệu 3 kênh RGB điều tiết, lưu vào bộ SDRAM.

A diagram of a camera control system

AI-generated content may be incorrect.

Hình 22: Sơ đồ khối camera control

Phần này gồm 3 module chính bao gồm: Camera control read, control write to FIFO, control store data to SDRAM và một bộ nhớ fifo memory.

* + 1. **Camera control read**

Module này có chứa năng nhận tín hiệu từ camera điều khiển ghi dữ liệu thô vào FIFO.

**Input port:**

PCLK, VSYNC, HREF: đồng bộ xung pixel, khung và dòng.

D[7:0]: dữ liệu màu/điểm ảnh.

**Output port:**

pixel\_data[15:0]: dữ liệu pixel 16 bit hợp lệ khi pixel\_valid = 1.

pixel\_valid: cao trong chu kỳ PCLK nếu dữ liệu của pixel là hợp lệ.

X\_index[8:0]: số chỉ vị trí của pixel theo trục x.

Y\_index[8:0]: số chỉ vị trí của pixel theo trục y.

|  |  |  |
| --- | --- | --- |
| State | Mô tả | Điều khiển chuyển trạng thái |
| Idle | Chờ tín hiệu VSYNC lên mức cao khi bắt đầu một khung. Reset y\_index về lại 0. | VSYNC = 1, trạng thái kế tiếp Row read |
| ReadFirstByte | Ghi byte đầu tiên | Chuyển sang ReadLastByte |
| ReadLastByte | Ghi byte cuối của một pixel. Cộng x\_index tiếp theo lên 1. | Nếu HREF = 1 chuyển lại về ReadFirstByte, Nếu HREF chuyển về 0 chuyển sang endRow |
| Endrow | Chờ HREF lên lại 1 cho hàng pixel tiếp theo, y\_index tăng lên 1. | Nếu HREF = 1 chuyển đến ReadFirstByte, Nếu VSYNC = 1, chuyển về trạng thái Idle. |

**FSM:**

Bảng 1: Bảng chuyển trạng thái khối cameraRead

* + 1. **Control write to FIFO**

Chức năng của khối này là nhận tín hiệu đọc frame từ bộ control store to sdram để bắt đầu nhận dữ liệu từ bộ camera read và ghi dữ liệu pixel từ cameraRead vào FIFO.

Module này điều khiển việc ghi dữ liệu từ camera vào FIFO dựa trên các tín hiệu i\_xIndex, i\_yIndex, và i\_get. Nó sử dụng một máy trạng thái (FSM) với 3 trạng thái: idle, process, finish.

Bảng chuyển trạng thái:

|  |  |  |  |
| --- | --- | --- | --- |
| Trạng thái hiện tại | Điều kiện chuyển trạng thái | Trạng thái kế tiếp | Mô tả |
| Idle | X\_index = 0; y\_index =0; i\_get = 1 | Process | Trạng thái nghĩ chờ tín hiệu bắt đầu và chờ đồng bộ frame |
| Idle | Không thỏa điều kiện trên | Idle | Chờ |
| Process | x\_index = 63, y\_index =63 | Finish | Kết thúc việc ghi vào fifo vì đã đủ 64x64 pixel. |
| Process | Không thỏa điều kiện trên | Process | Tiếp tục xử lý |
| Finish | Bấc kỳ | Idle | Bậc cờ hoàn thành. |

Bảng 2: Bảng chuyển trạng thái khối control write to FIFO

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Trạng thái | masterE | O\_complete | O\_process | O\_eWriteFIFO |
| Idle | 064 | 1 | 0 | 0 |
| Process | 1 | 0 | 1 | 1(nếu x < 64, y < ) |
| Finish | 0 | 1 | 0 | 0 |

Bảng output tương ứng với trạng thái:

Bảng 3: Bảng ngõ ra máy trạng thái khối control write to FIFO

* + 1. **control store data to sdram**

Module này điều khiển việc đọc dữ liệu từ FIFO (được ghi từ camera) và ghi vào SDRAM. Nó xử lý dữ liệu RGB, chia thành các thành phần Red, Green, Blue, và ghi vào các vùng nhớ khác nhau trên SDRAM.

Input port list:

Bảng 4: Bảng mô tả danh sách ngõ vào khối control read cam to SDRAM

|  |  |  |
| --- | --- | --- |
| Signal | Width | Mô tả |
| I\_clk | 1 | Clock signal input |
| I\_reset | 1 | Reset signal active low |
| I\_start | 1 | Start signal to start module |
| I\_remainOnFifo | 10 | Remain word data on FIFO |
| I\_process | 1 | Signal báo bộ điều khiển ghi vào FIFO vẫn còn hoạt động |
| I\_complete | 1 | Signal báo bộ điều khiển ghi vào FIFO đã hoàn thành, sẽ không còn word nào được ghi thêm vào FIFO. |
| I\_dataFIFO | 16 | Data đọc từ FIFO (RGB565) |

Output port list:

Bảng 5: Bảng mô tả danh sách ngõ ra khối control read cam to SDRAM

|  |  |  |
| --- | --- | --- |
| Signal | Width | Mô tả |
| o\_get | 1 | Enable và tín hiệu start cho bộ control write fifo bắt đầu hoạt động. |
| o\_EnReadFifo | 1 | Tín hiệu xin phép đọc FIFO |
| o\_RdClkFifo | 1 | Tín hiệu clock read FIFO |
| o\_dataSdram | 16 | Data đến SDRAM |
| o\_addressToSdram | 19 | SDRAM address |
| o\_wrSdram | 1 | Tín hiệu xin phép ghi vào SDRAM |
| o\_finish | 1 | Cờ báo đã lưu được một frame vào SDRAM |

Bảng chuyển trạng thái:

Bảng 6: Bảng chuyển trạng thái khối control read cam to sdram

|  |  |  |  |
| --- | --- | --- | --- |
| Trạng thái hiện tại | Điều kiện | Trạng thái tiếp theo | Mô tả |
| Idle | i\_start & i\_reset | Start | Trạng thái chờ |
| Idle | Không thỏa điều kiện trên | Idle |  |
| Start | I\_process | Suppend | Bậc cờ khởi động cho khối control write to FIFO. |
| Start | !I\_process | Start | Tiếp tục chờ đến khi khối control write to FIFO khởi động |
| Suppend | I\_remainOnFifo > 16  Hoặc i\_complete | readFifo0 | Sẳn sàng chuyển data từ fifo sang SDRAM |
| Suppend | Không thỏa điều kiện trên | Suppend | Tiếp tục chờ FIFO đủ dữ liệu |
| readFifo0 | Luôn luôn chuyển | readFifo | Tạo cạnh lên cho clock đọc dữ liệu fifo |
| readFifo1 | Luôn luôn chuyển | setRed | Kéo clock read fifo về lại 0 |
| setRed | Luôn luôn chuyển | Wait0 | Đưa cổng output data thành dữ liệu lênh đỏ, chuẩn bị address, bậc cờ cho phép ghi vào SDRAM |
| Wait0 | I\_sdramRead | setGreen | Chờ SDRAM hoàn thành việc ghi. |
| Wait0 | Không thỏa điều kiện trên | Wait0 | Tiếp tục chờ SDRAM hoàn thành ghi. |
| setGreen | Luôn luôn chuyển | Wait1 | Chuẩn bị ghi kênh Green |
| Wait1 | I\_sdramReady | setBlue | Chờ hoàn thành ghi |
| Wait1 | Không thỏa điều kiện trên | Wait1 |  |
| setBlue | Luôn luôn chuyển | Wait2 | Chuẩn bị ghi kênh Blue |
| Wait2 | I\_sdramReady | Update | Chờ hoàn thành ghi |
| Wait2 | Không thỏa điều kiện trên | Wait2 |  |
| Update | !i\_complete | i\_remain > 0 | Suspend | Cập nhật địa chỉ local, Kiểm tra nếu chưa hoàn thành frame. |
| Update | I\_complete & i\_remain = 0 | Finish | Cập nhật địa chỉ local, kiểm tra nếu đã hoàn thành frame |
| Finish | Luôn luôn chuyển | Idle | Bậc cờ hoàn thành |

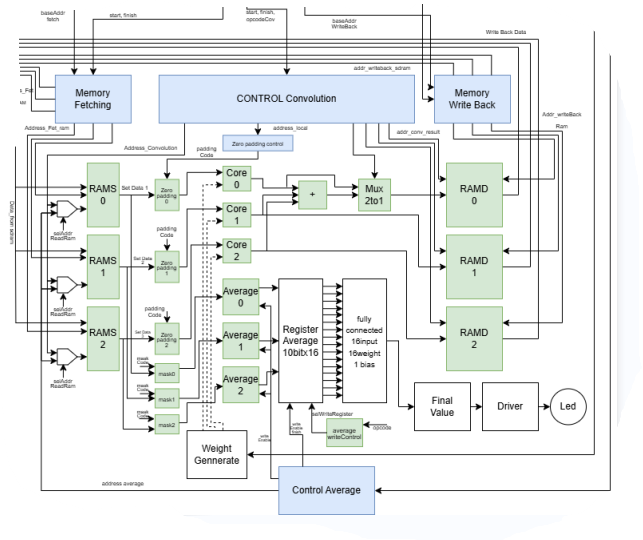
Bảng output của máy trạng thái:

Bảng 7: Bảng ngõ ra máy trạng thái khối control read camera to SDRAM

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Trạng thái | O\_get | O\_En  Read  FIFO | O\_Rd  Clk  Fifo | O\_wr  Sdram | O\_finish | Mô tả hành vi |
| Idle | 0 | 0 | 0 | 0 | 0 | Chờ tín hiêu i\_start |
| Start | 1 | 0 | 0 | 0 | 0 | Bậc cờ o\_get để khởi động khối control write to FIFO |
| Suppend | 0 | 1 | 0 | 0 | 0 | Bậc cờ xin phép đọc fifo, chờ FIFO đủ data |
| readFifo0 | 0 | 1 | 1 | 0 | 0 | Tạo rasing read clock fifo |
| readFifo1 | 0 | 1 | 0 | 0 | 0 | Reset clock read fifo trở lại 0, tắt tín hiệu xin phép đọc fifo. |
| setRed | 0 | 0 | 0 | 1 | 0 | Bậc tín hiệu xin phép ghi vào SDRAM, chuẩn bị data kênh đỏ, address kênh đỏ.  Addr = addrLocal + 0 |
| Wait0 | 0 | 0 | 0 | 0 | 0 | Chờ sdram ghi xong |
| SetGreen | 0 | 0 | 0 | 1 | 0 | Chuẩn bị ghi kênh xanh lá.  Addr = addrLocal + 4096 |
| Wait1 | 0 | 0 | 0 | 0 | 0 | Chờ sdram ghi xong |
| setBlue | 0 | 0 | 0 | 1 | 0 | Chuẩn bị ghi kênh xanh dương  Addr = addrLocal + 8192 |
| Wait2 | 0 | 0 | 0 | 0 | 0 | Chờ SDRAM ghi xong |
| Update | 0 | 0 | 0 | 0 | 0 | Tăng addrLocal lên 1, kiểm tra fifo còn data không. |
| Finish | 0 | 0 | 0 | 0 | 1 | Bậc cờ finish, reset addressLocal về lại 0 |

* 1. **Khối main mobile net**

Khối Main MobileNet là trung tâm xử lý chính trong hệ thống, thực hiện toàn bộ các bước tính toán của mô hình MobileNet từ khi nhận dữ liệu ảnh cho đến khi đưa ra kết quả. Bao gồm các giai đoạn: đọc dữ liệu (fetch), tính toán tích chập (convolution), tính trung bình cộng (average pooling), lớp kết nối đầy đủ (fully connected) và ghi lại dữ liệu (write back). Mỗi bước đều được thiết kế tối ưu để tận dụng tài nguyên phần cứng của FPGA, đồng thời đảm bảo khả năng xử lý song song và tuần tự phù hợp với kiến trúc MobileNet ban đầu (3 core xữ lý song song). Việc tổ chức độc lập các khối chức năng và dùng chung các khối RAM nguồn (source ram) và RAM đích (destination ram) nhầm tiếp kiệm tài nguyên bộ nhớ cho FPGA.



Hình 23: Sơ đồ mô hình Mobile Net

* + 1. **Memory fetching**

Module này có chức năng lấy dữ liệu từ SDRAM và lưu vào bộ nhớ RAM nguồn (source ram) nhầm chuẩn bị cho việc tính toán.

Input port của module fetching memory.

Bảng 8: Danh sách input port của module memory fetching

|  |  |  |
| --- | --- | --- |
| Tên tín hiệu | Độ rộng | Mô tả |
| I\_baseAddr0 | 19 | Địa chỉ cơ sở cho SDRAM vùng 0 |
| I\_baseAddr1 | 19 | Địa chỉ cơ sở cho SDRAM vùng 1 |
| I\_baseAddr2 | 19 | Địa chỉ cơ sở cho SDRAM vùng 2 |
| I\_clk | 1 | Clock |
| I\_reset | 1 | Reset signal |
| I\_sdramReady | 1 | Tín hiệu báo SDRAM sẵn sàng |
| I\_start | 1 | Tín hiệu khởi động module fetching |

Ouput port của module fetching memory.

Bảng 9: Danh sách output port của module fetching memory

|  |  |  |
| --- | --- | --- |
| Tín hiệu | Độ rộng | Mô tả |
| O\_rdSdram | 1 | Tín hiệu xin phép đọc SDRAM |
| O\_addrToSdram | 19 | Address đến SDRAM |
| O\_finish | 1 | Tín hiệu báo hoàn thành fetching |
| O\_wrRam0 | 1 | Tín hiệu xin phép ghi vào RAM0 |
| O\_wrRam1 | 1 | Tín hiệu xin phép ghi vào RAM1 |
| O\_wrRam2 | 1 | Tín hiệu xin phép ghi vào RAM2 |
| O\_addrToRam | 12 | Địa chỉ cho việc ghi vào RAM. Dùng chung cho 3 vùng ram. |

Bảng chuyển trạng thái của khối memory fetching

Bảng 10: Bảng chuyển trạng thái module memory fetching

|  |  |  |  |
| --- | --- | --- | --- |
| Trạng thái  hiện tại | Điều kiện chuyển | Trạng thái  kế tiếp | Mô tả |
| Idle | I\_start | setSdram0 | Chờ tín hiệu khởi động bộ fet |
| Idle | !i\_start | Idle |  |
| setSdram0 | Luôn luôn | waitSdram0 | Chuẩn bị address đọc dữ liệu từ SDRAM |
| waitSdram0 | I\_sdramReady | setRam0 | Chờ dữ sdram hoàn thành đọc dữ liệu |
| waitSdram0 | !i\_sdramReady | waitSdram0 |  |
| setRam0 | Luôn luôn | setSdram1 | Ghi dữ liệu vừa đọc được từ sdram vào block ram 0 |
| setSdram1 | Luôn luôn | waitSdram1 | Chuẩn bị address đọc dữ liệu từ sdram cho block ram 1 |
| waitSdram1 | i\_sdsramReady | setRam1 | Chờ sdram hoàn thành đọc |
| WaitSdram1 | !i\_sdramReady | waitSdram1 |  |
| setRam1 | Luôn luôn | setSdram2 | Ghi dữ liệu vừa đọc được từ sdram vào block ram 1 |
| setSdram2 | Luôn luôn | waitSdram2 | Chuẩn bị địa chỉ đọc dữ liệu sdram cho block ram 2 |
| waitSdram2 | i\_sdramReady | setRam2 | Chờ Sdram hoàn thành đọc |
| waitSdram2 | !i\_sdramReady | waitSdram2 |  |
| setRam2 | Luôn luôn | Update | Ghi dữ liệu vừa đọc được từ sdram vào block ram 2 |
| Update | addrLocal = 4095 | Finish | Cập nhật addrLocal và kiểm tra nếu đây là dữ liệu cuối cùng của feature |
| Update | addrLocal!= 4095 | setSdram0 |  |
| Finish | Luôn luôn | Idle | Bậc cờ báo fetching hoàn thành |

Output máy trạng thái module memory fetching

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Trạng thái | O\_rdSdram | O\_wr  Ram0 | O\_wr  Ram1 | O\_wr  Ram2 | O\_addrTo  Sdram | O\_finish |
| Idle | 0 | 0 | 0 | 0 | 0 | 0 |
| setSdram0 | 1 | 0 | 0 | 0 | I\_baseAddr0  +addrLocal | 0 |
| waitSdram0 | 0 | 0 | 0 | 0 | I\_baseAddr0  +addrLocal | 0 |
| setRam0 | 0 | 1 | 0 | 0 | I\_baseAddr0  +addrLocal | 0 |
| setSdram1 | 1 | 0 | 0 | 0 | I\_baseAddr1  +addrLocal | 0 |
| waitSdram1 | 0 | 0 | 0 | 0 | I\_baseAddr1  +addrLocal | 0 |
| setRam1 | 0 | 0 | 1 | 0 | I\_baseAddr1  +addrLocal | 0 |
| setSdram2 | 1 | 0 | 0 | 0 | I\_baseAddr2  +addrLocal | 0 |
| waitSdram2 | 0 | 0 | 0 | 0 | I\_baseAddr2  +addrLocal | 0 |
| setRam2 | 0 | 0 | 0 | 1 | I\_baseAddr2  +addrLocal | 0 |
| Update | 0 | 0 | 0 | 0 | I\_baseAddr0  +addrLocal | 0 |
| Finish | 0 | 0 | 0 | 0 | 0 | 1 |

* + 1. **Ram nguồn (source ram) và Ram đích (destination ram)**

Module Ram này có cấu trúc gồm một block Ram (convRam) và một bộ điều khiển đọc ghi tuần tự (ramControl)

Module ramControl quản lý việc đọc tuần tự từ RAM (ramConv), tổng hợp dữ liệu thành bộ đệm 90-bit và cung cấp giao diện đọc nhanh.

Module convRam là loại RAM một cổng (4096x10-bit) với hoạt động đọc/ghi đồng bộ.

Như vậy ta có thể thấy rằng module RAM mặc dù cần xuất 9 dữ liệu qua 9 cổng output (10 bit) để thực hiện tích chập với kernal. Tuy nhiên thiết kế vấn sử dụng module RAM một cổng ngõ ra 10 bit đồng bộ. Điều này sẽ có hạn chế là thay vì RAM có 9 port data output sẽ đọc được 9 mẫu dữ liệu trong một chu kỳ, Còn khi thực hiện đọc tuần tự như trên sẽ cần ít nhất 9 chu kỳ để có thể đọc được 9 dữ liệu ở 9 địa chỉ khác nhau. Tuy nhiên ta vẫn phải thực hiện điều này do các nguyên nhân sau đây: Tối Ưu Tài Nguyên Phần Cứng, Đọc song song 9x10-bit yêu cầu 9 bộ đọc độc lập, 9 đường địa chỉ, và 9 đường dữ liệu. Chiếm dụng nhiều LUTs, FFs, và BRAMs trên FPGA, đặc biệt với kích thước RAM 4096x10-bit. Đọc tuần tự chỉ cần 1 bộ đọc và 1 đường địa chỉ/dữ liệu, tái sử dụng qua các chu kỳ tiết kiệm tài nguyên phần cứng của FPGA. Quan trọng hơn là khi dùng ram 1 port dữ liệu ra sẽ được quartus synthesis sử dụng embedded ram trên FPGA. Nếu ta sử dụng 9 port song song quartus buộc phải dùng đến memory element phân tán (distribute memory elements) Điều này sẽ tiêu tốn rất nhiều tài nguyên của FPGA dẫn đến không thể systhesis được do thiếu tài nguyên.

Input/output port module ramControl

Bảng 11: Danh sách input output port của khối điều khiển RAM.

|  |  |  |  |
| --- | --- | --- | --- |
| Port name | In/out | Width | Discription |
| I\_addrOut | Input | 108 | 9x12bit địa chỉ đọc dữ liệu 9 port |
| I\_addrIn | Input | 12 | Địa chỉ ghi vào RAM |
| I\_dataIn | Input | 10 | Dữ liệu ghi vào RAM |
| I\_wrEnable | Input | 1 | Tín hiệu xin phép ghi vào bộ RAM |
| I\_quickGet | Input | 1 | Cờ kích hoạt chế độ đọc nhanh (Chỉ đọc 1 dữ liệu, mà không cần phải chờ 9 chu kỳ để đọc 9 bộ dữ liệu) |
| I\_addrOutQuick | Input | 12 | Địa chỉ cho việc đọc nhanh |
| I\_clk | Input | 1 | Xung clock |
| I\_reset | Input | 1 | Tín hiệu reset (tích cực thấp) |
| I\_start | Input | 1 | Tín hiệu bắt đầu đọc tuần tự |
| O\_dataOut | Output | 90 | Dữ liệu đọc được từ mode đọc tuần tự |
| O\_quickData | Output | 10 | Dữ liệu đọc được từ mode đọc nhanh 1 dữ liệu |
| O\_valid | Output | 1 | Cờ báo hiệu hợp lệ cho mode đọc tuần tự |
| O\_ready | Output | 1 | Cờ báo bộ RAM đang ở chế độ rãnh, sẵn sàng nhận lệnh |

* + 1. Module tính toán convolution

A diagram of a computer

AI-generated content may be incorrect.

Hình 24: Kiến trúc module conv

Module coreConv là một khối xử lý tích chập 9 mẫu: Nhận vào hai bus 90 bit (i\_data và i\_weight), mỗi bus chứa 9 phần tử 10 bit liên tiếp, sau đó giải nén thành mảng data[8:0] và weight[8:0]. 9 cặp data[i] và weight[i] được đưa song song vào chín khối multi để tính tích, sinh ra mảng resultMulti[9:0]. Cuối cùng, toàn bộ tám tích số này được cộng lại trong một khối plus9Para đa ngõ (9-input adder) để cho ra kết quả 10 bit duy nhất trên đầu ra o\_data.

Trên nền tảng đó, module conv đóng vai trò top-level, tái sử dụng ba module coreConv hoạt động song song: Mỗi instantiation có bus dữ liệu (i\_busData0/1/2) và bus trọng số (i\_busWeight0/1/2) riêng biệt dài 90 bit, cho phép thực hiện đồng thời ba phép tích chập khác nhau. Kết quả 10 bit của từng kênh được xuất ra qua o\_data0, o\_data1 và o\_data2. Tham số i\_opcode 1 bit dùng để điều khiển chế độ hoạt động (0: thực hiện mode tích chập 3 feauture thành 1 kết quả, 1: thực hiện mode tích chập 1 feature được 1 kết quả) cho cả ba khối con, giúp linh hoạt trong các ứng dụng yêu cầu các phép tính tích chập có 2 mode điều chỉnh.

Danh sách ngõ ra, ngõ vào module convCore:

Bảng 12: Danh sách input/output module convCore

|  |  |  |  |
| --- | --- | --- | --- |
| Port name | In/out | Width | Discription |
| I\_data | Input | 90 | Dữ liệu đầu vào 9x10bit |
| I\_weight | Input | 90 | Weight đầu vào 9x10bit |
| O\_data | Output | 10 | Kết quả phép tính chập (Tổng 9 phép nhân data và trọng số) |

Dánh sách ngõ ra, vào khối conv:

|  |  |  |  |
| --- | --- | --- | --- |
| Port name | In/out | Width | Discription |
| I\_busData0 | Input | 90 | Dữ liệu vào luồng 0, 9x10bit |
| I\_busData1 | Input | 90 | Dữ liệu vào luồng 1, 9x10bit |
| I\_busData1 | Input | 90 | Dữ liệu vào luồng 2, 9x10bit |
| I\_busWeight0 | Input | 90 | Weight vào luồng 0, 9x10bit |
| I\_busWeight1 | Input | 90 | Weight vào luồng 1, 9x10bit |
| I\_busWeight2 | Input | 90 | Weight vào luồng 2, 9x10bit |
| I\_opcode | Input | 90 | Mode tích chập 3 feature hoặc 1 feature |
| O\_data0 | Output | 10 | Kết quả tích chập luồng 0 |
| O\_data1 | Output | 10 | Kết quả tích chập luồng 1 |
| O\_data2 | Output | 10 | Kết quả tích chập luồng 2 |

Về việc dữ lý số fixing point khi thực hiện trên phần cứng. Bởi vì, việc thực hiện phép nhân trên số fixing point sẽ có độ phức tạp thấp hơn trên số floating point truyền thống trên máy tính. Ta buộc phải chấp nhận lượng tử hóa weight và kết quả đầu ra nhầm đơn giản hóa việc thiết kế phép nhân. Fixing point của mô hình sẽ được thiết kế như sau: Weight sẽ sử dụng định dạng số Fixing point là Q0.9 nghĩa là 1 bit dấu, 0 bit cho phần nguyên và 9 bit cho phần thập phân (Vì trọng số của mô hình sẽ nằm trong khoảng (-1;1) Như vậy ta có độ phân giải (resolution) của weight là 0.002. Feature map hoặc pixel đầu vào sẽ được sử dụng định dạng fixing point là Q4.6 (lưu ý rằng pixel đầu vào mỗi kênh có giá trị trong đoạn [0:255] vì vậy cần được scale về đoạn [0:1]) Như vậy đối với feature map và pixel input sẽ có độ phân giải là 0.0156. Vì phép nhân giữa số có dấu Q0.9 và Q3.6 ta sẽ được số Q3.15 có dấu và feature cần giữ đúng độ rộng bit là 10 vì vậy ta cần scale kết quả Q3.15 thành định dạng Q3.6.

assign o\_data = dataTemp[18:9];

Kết quả sẽ được lấy từ bit 18 đến bit thứ 9 thay vì lấy toàn bộ 19 bit kết quả phép nhân.

Ngoài ra khi tính toán đôi khi ra không lấy hoàn toàn 9 bộ dữ liệu từ RAM mà đôi khi phải chèn 0 vào một số mẫu. Lý do khi thực hiện tích chập ở các cạnh của feature ta cần thêm 0 vào những vị trí không có mẫu (ví dụ khi thực hiện tích chập tại vị trí có tọa độ x=12, y=0 – địa chỉ mẫu ngay hàng đầu tiên của feature, khi này ta cần thêm 3 mẫu 0 vào hàng trên của cửa sỗ.)

A diagram of a computer code

AI-generated content may be incorrect.

Hình 25: Minh họa thêm module chức năng padding cho module convolution

Như vậy ta có thêm khối Zero padding và khối padding control. Khối Zero padding có chức năng thêm các mẫu 0 vào bộ data với sự điều khiển từ khối padding control. Khối padding có chức năng nhận tín hiệu vị trí tham chiếu của vị trí đang được tính tích chập để đưa ra tính hiệu điều khiển padding.

Khối padding control port list:

Bảng 13: Danh sách input output module padding control

|  |  |  |  |
| --- | --- | --- | --- |
| Port name | In/out | Width | Discription |
| I\_localAddr | Input | 12 | Địa chỉ pixel trong ma trận (kích thước 64x64) |
| O\_sel[0:8] | Output | 1 | Tín hiệu chọn để xác định vị trí padding |

Logic: Phân tích i\_localAddr thành 2 phần: hHigh/hLow: Kiểm tra 6 bit cao (địa chỉ hàng). hHigh = 1 khi tất cả 6 bit cao là 1 → Hàng cuối cùng. hLow = 1 khi tất cả 6 bit cao là 0 → Hàng đầu tiên. lHigh/lLow: Kiểm tra 6 bit thấp (địa chỉ cột). lHigh = 1 khi tất cả 6 bit thấp là 1 → Cột cuối cùng. lLow = 1 khi tất cả 6 bit thấp là 0 → Cột đầu tiên.

Xác định vị trí padding: Các tín hiệu o\_sel0–o\_sel8 tương ứng với 9 vị trí trong kernel 3x3.

Khối zero padding: Đơn giản hơn rất nhiều chỉ cần thực hiện phép toán AND tương tứng data và tín hiệu select. Vd: o\_data[9:0] = i\_sel0 & i\_data[9:0] tương tự với các data còn lại.

Tiếp theo là khối rất quan trọng đóng vai trò điều khiển hoạt động của bộ convolution – Module controlConv

Module convolutionControl điều khiển quá trình thực hiện tích chập: từ việc phát lệnh khởi động, đọc dữ liệu từ source RAM, chờ source RAM trả dữ liệu hợp lệ, ghi kết quả ngược lại vào destination RAM rồi tự động cập nhật con trỏ địa chỉ cho đến khi hoàn thành duyệt hết vùng dữ liệu (đến địa chỉ 4095). Toàn bộ quá trình này được cài đặt dưới dạng FSM, sử dụng các tín hiệu điều khiển i\_start, i\_reset, i\_validRam và i\_opcode để đồng bộ và phân luồng truy cập RAM, đồng thời sinh ra các tín hiệu điều khiển đọc/ghi và báo hiệu hoàn thành.

A diagram of a computer system

AI-generated content may be incorrect.

Hình 26: Minh họa khối control convolution giao tiếp với khối convolution

Danh sách ngõ ra, ngõ vào khối convolutionControl:

Bảng 14: Danh sách input output port module convolution control

|  |  |  |  |
| --- | --- | --- | --- |
| Port name | In/out | Width | Discription |
| I\_clk | Input | 1 | Xung clock |
| I\_reset | Input | 1 | Reset signal (active low) |
| I\_start | Input | 1 | Tín hiệu khởi động quá trình tích chập |
| I\_opcode | Input | 1 | Tín hiệu điều khiển mode tích chập (3 feauture hoặc 1 feature) |
| I\_validRam | Input | 1 | Tín hiệu xác nhận dữ liệu đọc từ source ram đã sẳn sàng và hợp lệ |
| O\_addrRead0 | Output | 108 | Địa chỉ đọc dữ liệu cho luồng 0 (9x12bit) |
| O\_addrRead1 | Output | 108 | Địa chỉ đọc dữ liệu cho luồng 1 (9x12bit) |
| O\_addrRead2 | Output | 108 | Địa chỉ đọc dữ liệu cho luồng 2 (9x12bit) |
| O\_startRam | Output | 1 | Tín hiệu khởi động source RAM bắt đầu đọc |
| O\_selRamD0 | Output | 1 | Tín hiệu chọn opcode cho convolution |
| O\_addrWrite0 | Output | 12 | Địa chỉ ghi kết quả trên destination ram luồng 0 |
| O\_addrWrite1 | Output | 12 | Địa chỉ ghi kết quả trên destination ram luồng 1 |
| O\_addrWrite2 | Output | 12 | Địa chỉ ghi kết quả trên destination ram luồng 2 |
| O\_wrEnable | Output | 1 | Tín hiệu write enable cho destination ram |
| O\_finish | Output | 1 | Cờ báo hiệu feature đã được tích chập xong, kết quả đã được lưu vào destination ram |
| O\_localAddr | Output | 12 | Đại chỉ pixel hiện tại đang được xử lý |

Bảng chuyển trạng thái:

Bảng 15: Bảng chuyển trạng thái khối convolution control

|  |  |  |
| --- | --- | --- |
| Trạng thái hiện tại | Điều kiện chuyển trạng thái | Trạng thái kế tiếp |
| Idle | I\_start & i\_reset | readRam |
| Idle | Không thỏa điều kiện trên | Idle |
| readRam | Luôn luôn chuyển | waitDone |
| waitDone | I\_validRam | setWrite |
| waitDone | !i\_validRam | waitDone |
| setWrite | Luôn luôn chuyển | Update |
| Update | addrLocal = 4095 | Finish |
| Update | Không thỏa điều kiện trên | readRam |
| Finish | Luôn luôn chuyển | Idle |

Bảng output của mã trạng thái:

Bảng 16: Bảng output của máy trạng thái module convolution control

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Trạng thái | O\_startRam | O\_wrEnable | O\_finish | Mô tả hành vi |
| Idle | 0 | 0 | 0 | Chờ tín hiệu i\_start. Reset địa chỉ và tín hiệu điều khiển. |
| readRam | 1 | 0 | 0 | Kích hoạt đọc RAM. Tính toán địa chỉ đọc (9 vị trí xung quanh addrLocal\_q). |
| waitDone | 0 | 0 | 0 | Chờ tín hiệu i\_validRam để xác nhận dữ liệu đã sẵn sàng. |
| setWrite | 0 | 1 | 0 | Kích hoạt ghi kết quả vào RAM tại địa chỉ addrLocal\_q. |
| Update | 0 | 0 | 0 | Tăng addrLocal\_q. Nếu đạt 4095, chuyển sang finish; ngược lại, tiếp tục đọc. |
| Finish | 0 | 0 | 1 | Báo hiệu hoàn thành và reset trạng thái. |

Module tính toán 9 địa chỉ đọc xung quanh vị trí hiện tại (addrLocal\_q) để lấy dữ liệu cho kernel 3x3:

addrRead0 = addrLocal\_q - 65 (góc trên trái)

addrRead1 = addrLocal\_q - 64 (trên giữa)

addrRead2 = addrLocal\_q - 63 (góc trên phải)

addrRead3 = addrLocal\_q - 1 (giữa trái)

addrRead4 = addrLocal\_q (trung tâm)

addrRead5 = addrLocal\_q + 1 (giữa phải)

addrRead6 = addrLocal\_q + 63 (góc dưới trái)

addrRead7 = addrLocal\_q + 64 (dưới giữa)

addrRead8 = addrLocal\_q + 65 (góc dưới phải)

* + 1. **Khối thuật toán Average**

Average trong kiến trúc MobileNet đảm nhiệm chức năng pooling trung bình (“average pooling”) để gom thông tin không gian từ feature-map thành một vector giá trị đặc trưng cho mỗi kênh, giảm độ phân giải không gian mà vẫn giữ lại đặc trưng chung. Phần Average gồm một số module chính như sau: AverageControl: một FSM điều khiển tuần tự đọc tuần tự các pixel trong feature. Average: mạch chính thực hiện phép cộng tích lũy các giá trị đã đọc, rồi chia (hoặc dịch bit) theo số phần tử để tính trung bình. AverageWriteControl: module sinh tín hiệu select ghi và tín hiệu wrEnable để xuất kết quả trung bình vào bộ register chứa giá trị trung bình.

Module average: Có chức năng tính trung bình bằng cách cộng dồn dữ liệu, mỗi lần cộng đồn là 9 dữ liệu đầu vào.

Bảng 17: Dánh sách input output của module average

|  |  |  |  |
| --- | --- | --- | --- |
| Port name | In/out | Width | Discription |
| I\_data | Input | 90 | Dữ liệu đầu vào 9x10bit |
| I\_clk | Input | 1 | Xung clock |
| I\_reset | Input | 1 | Reset signal |
| I\_writeAdd | Input | 1 | Tín hiệu cộng dồn |
| O\_data | Output | 10 | Giá trị trung bình đầu ra |

Simple logic: Bấc kì khi nào có clock thì module sẽ kiểm tra nếu i\_writeAdd bằng 1 thì thanh ghi tính tổng sẽ cộng dồn i\_data vào. Và khi hoàn thành, để tính trung bình 64 phần tử ta cần chia giá trị tổng cho tổng số lượng phần tử ở đây là 64x64 phần tử hay 4096 đúng bằng 2 lũy thừa 12. Vì vậy ta chỉ cần dịch sang phải 12 bit là được tương ứng với chia 4096.

Như ta có thể thấy rằng module average không thể tính trung bình một lượt 64x64 phần tử vì Ram không thể xuất cùng lúc 64x64 phần tử. Đồng thời nếu làm vậy sẽ gây áp lức tính toán lớn hơn cho tài nguyên trên FPGA. Do vậy average đã được thiết kế để tính cộng dồn tích lũy mỗi lần 9 phần tử đến khi hoàn thành. Vì vậy ta cần một module để điều khiển việc load dữ liệu từ RAM, thực hiện cộng dồn, ghi dữ liệu vào register.

Bảng 18: Danh sách input output của module averageControl

|  |  |  |  |
| --- | --- | --- | --- |
| Port name | In/out | Width | Discription |
| I\_clk | In | 1 | Xung clock |
| I\_reset | In | 1 | Reset signal (active low) |
| I\_start | In | 1 | Tín hiểu khởi động quá trình tính trung bình của một feature |
| I\_validRam | In | 1 | Tín hiệu báo hiệu RAM đọc hoàn tấc và dữ liệu được sẳn sàng |
| O\_addrRead0 - 2 | Output | 108 | Địa chỉ đọc dữ liệu RAM 9x12bit, 3 core |
| O\_writeEnable | Output | 1 | Tín hiệu xin phép ghi kết quả tính trung bình vào register |
| O\_reserAverage | Output | 1 | Tín hiệu để reset giá trị cộng tích lũy về lại 0 |
| O\_mask | Output | 1 | Che dữ liệu khi thực hiện chu kỳ cuối cùng, feature không còn đủ 9 data để tính |
| O\_startRam | Output | 1 | Tín hiệu khởi động đọc RAM |
| O\_finish | Output | 1 | Cờ báo hoàn thành tính trung bình của một feature |

Như vậy ta cần thiết kế một máy trạng thái để thực hiện module average control:

Bảng 19: Bảng chuyển trạng thái module average control

|  |  |  |  |
| --- | --- | --- | --- |
| Trạng thái hiện tại | Điều kiện chuyển | Trạng thái kế tiếp | Mô tả hành vi |
| Idle | I\_start & i\_reset | Reset | Chờ tín hiệu khởi động |
| Reset | Luôn luôn chuyển | Buffer | Bậc tín hiệu reset để xóa giá trị tổng tích lũy cũ |
| Buffer | Luôn luôn chuyển | setAddr | Xóa tín hiệu reset |
| setAddr | Luôn luôn chuyển | waitDone | Thiết lập địa chỉ đọc dữ liệu cho bộ RAM, bậc cờ khởi động đọc |
| waitDone | I\_validRam | writeAdd | Chờ RAM hoàn thành việc đọc dữ liệu |
| writeAdd | Luôn luôn chuyển | Update | Bậc cờ cho phép cộng tích lũy |
| Update | addrLocal =4095 | Finish | Cập nhập địa chỉ addrLocal += 9, kiểm tra nếu đây là pixel cuối của feature. |
| Update | Không thỏa điều kiện trên | setAddr |  |
| Finish | Luôn luôn chuyển | Idle | Bậc cờ hoàn thành và bậc cờ ghi vào register. |

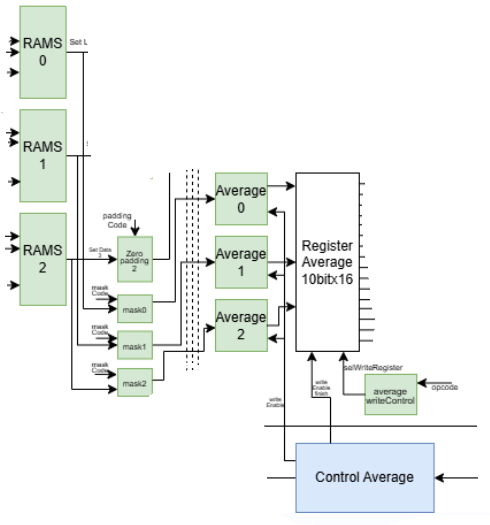
Vì dữ liệu sau khi tích trung bình được lưu vào một bộ register. Không phải là RAM đích (destinaiton ram) như module convolution. Vì vậy ta cần một module nữa để điều khiển việc ghi vào register file. Khối này có chức năng tương tự như khối write back của khối convolution.

|  |  |  |  |
| --- | --- | --- | --- |
| Port name | In/out | Width | Discription |
| I\_opcode | Input | 6 | Mã lệnh xác định feature đang được tính trung bình |
| O\_selWrite | Output | 16 | Một mặt nạ 16 bit dùng làm tín hiệu write enable chỏ register file |

Module registerAverage được thiết kế để quản lý một bộ gồm 16 thanh ghi, mỗi thanh ghi có độ rộng 10 bit, dùng để lưu trữ dữ liệu và xử lý sau này. Module hoạt động đồng bộ với tín hiệu xung nhịp và có tín hiệu reset tích cực mức thấp, giúp khởi tạo tất cả các thanh ghi về giá trị 0. Việc ghi dữ liệu được điều khiển thông qua tín hiệu cho phép ghi (i\_enableWrite) và đầu vào chọn 16 bit (i\_selWrite), xác định thanh ghi cụ thể nào sẽ được cập nhật. Khi ghi được cho phép, module sẽ ghi ba dữ liệu đầu vào 10 bit (i\_data0, i\_data1, hoặc i\_data2) vào thanh ghi được chọn, dựa trên điều kiện khớp trong câu lệnh case. Module đã có phần xử lý việc nhập và lưu trữ dữ liệu.

Bởi vì average sẽ tính trung bình tích lũy theo từng 9 mẫu data trong khi một feature có 4096 data. Vì vậy ở lần cộng tích lũy cuối cùng ta chỉ còn 1 data thứ 4095 duy nhất. Vì vậy ta cần một mast để xóa đi những dữ liệu trên mà RAM cho ra khi cố gắng đọc 9 phần tử.

Như vậy kiến trúc average sẽ như sau:

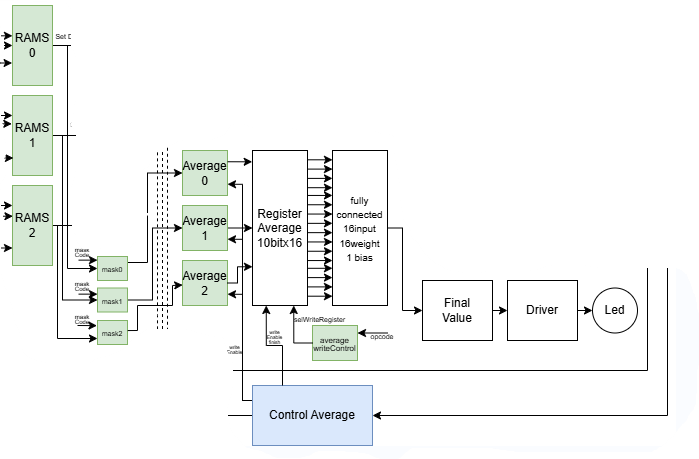


Hình 27: Kiến trúc khối thực hiện chức năng tính average

Tuy nhiên việc hoàn thành thực hiện tính toán trung bình vẫn chưa hoàn thành mô hình mobile net. Bước cuối của việc này là tình tích chập 16 giá trị trung bình này với 16 weight và 1 bias để được 1 giá trị output của mô hình. Và ta sẽ dùng giá trị output này để drive led nhầm hiển thị kết quả.

Module fullyConnected này triển khai một lớp mạng neural fully connected đơn giản, nhận vào 16 giá trị đầu vào (mỗi giá trị rộng 10 bit), nhân từng giá trị với 16 trọng số nội bộ, cộng chúng theo từng cặp, và cuối cùng cộng thêm giá trị bias để tính ra kết quả đầu ra. Các đầu vào được đóng gói vào một bus rộng 160 bit. Trong quá trình hoạt động, khi i\_enable ở mức cao, module sẽ thực hiện phép tính: mỗi đầu vào được nhân với trọng số tương ứng (ví dụ: in0 \* w0, in1 \* w1, v.v...), sau đó các kết quả được cộng theo cặp (như in0\*w0 + in1\*w1, rồi in2\*w2 + in3\*w3, v.v...). Cấu trúc nhóm này giúp giảm độ phức tạp của phép tính trong phần cứng. Kết quả cuối cùng, bao gồm cả bias, được lưu vào o\_result sau mỗi chu kỳ xung nhịp.

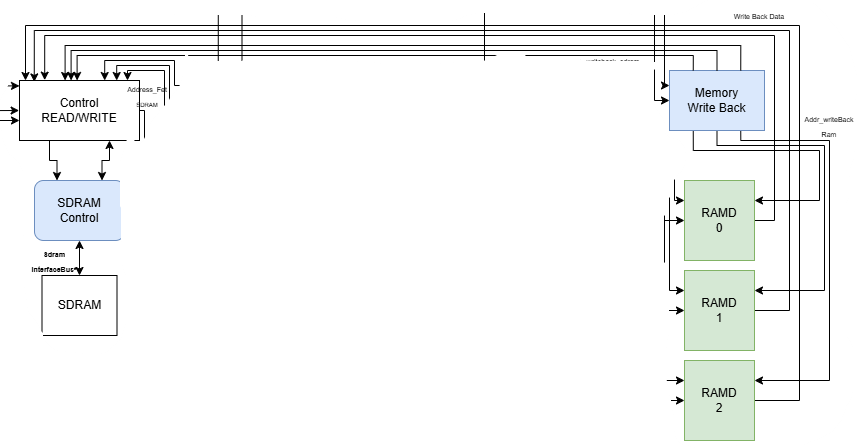
Như vậy kiến trúc của phần average và fully connected sẽ có dạng như sau:



Hình 28: Kiến trúc phần tính average bao gồm luôn phần fullyconnected và drive led

* + 1. **Write back destiantion ram to SDRAM**

Mô-đun Verilog này được thiết kế để kiểm soát quá trình ghi dữ liệu lại từ RAM đích (destination ram) đến SDRAM ngoài. Nó đảm bảo rằng dữ liệu đã xử lý được lưu trữ trong destiantiol RAM bên trong có thể được chuyển trở lại bộ nhớ SDRAM một cách hiệu quả và chính xác. Mô-đun này có khả năng quản lý việc tạo địa chỉ, xác thực dữ liệu và tín hiệu “hand shake” cần thiết để truyền dữ liệu an toàn. Nó hoạt động theo trình tự được kiểm soát, cho phép ghi dữ liệu vào SDRAM từng word một hoặc theo từng đợt. Ta buộc phải thực hiện việc write back mà không lưu lươn kết quả ở destination ram là vì ta cần giải phóng không gian cho việc thực hiện convolution cho các feature tiếp theo. Và SDRAM trên DE0-Nano có dung lượng rất lớn lên đến 32MB nên ta không cần phải lo lắng về việc ghi lại kết quả vào SDRAM.



Hình 29: Kiến trúc khối chức năng write back

Ta có thể thấy trong khối chức năng write back. Phần quan trọng nhất là khối Memory write back control.

Bảng 20: Danh sách input, output của module write back control

|  |  |  |  |
| --- | --- | --- | --- |
| Port name | In/out | Width | Discription |
| I\_clk | In | 1 | Xung clock |
| I\_reset | In | 1 | Tín hiệu reset (active low, asynchornus) |
| I\_sdramReady | In | 1 | Tín hiệu báo sdram đang rãnh |
| I\_baseAddr0 | In | 19 | Base address point to first SDRAM region |
| I\_baseAddr1 | In | 19 | Base address point to second SDRAM region |
| I\_baseAddr2 | In | 19 | Base address point to 3th SDRAM region |
| I\_start | In | 1 | Tín hiệu bắt đầu quá trình write back |
| O\_addrToRam0 | Out | 12 | Address to get data from RAM0 |
| O\_addrToRam1 | Out | 12 | Address to get data from RAM1 |
| O\_addrToRam2 | Out | 12 | Address to get data from RAM2 |
| O\_quickRam | Out | 1 | Tín hiệu chọn mode quick read of RAM |
| O\_addrToSdram | Out | 19 | Address to SDRAM |
| O\_wrSdram | Out | 1 | Tín hiệu xin phép ghi vào SDRAM |
| O\_selData | Out | 2 | Chọn data để ghi vào SDRAM |
| O\_finish | Out | 1 | Cờ báo hoàn thành quá trình write back |

Bảng 21: Bảng chuyển trạng thái module write back control

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Trạng thái  hiện tại | Điều kiện | Trạng thái  tiếp theo | O\_fnish | O\_  Quick  Ram | O\_  selData | O\_ wr  SDram |
| Idle | I\_reset & i\_start | setRam | 0 | 0 | 0 | 0 |
| setRam | Luôn luôn | setSdram0 | 0 | 1 | 0 | 0 |
| setSdram0 | Luôn luôn | waitDone0 | 0 | 0 | 0 | 1 |
| waitDone0 | I\_sdramReady | setSdram1 | 0 | 0 | 0 | 0 |
| setSdram1 | Luôn luôn | waitDone1 | 0 | 0 | 1 | 1 |
| waitDone1 | I\_sdramReady | setSdram2 | 0 | 0 | 1 | 0 |
| setSdram2 | Luôn luôn | waitDone2 | 0 | 0 | 2 | 1 |
| waitDone2 | I\_sdramReady | Update | 0 | 0 | 2 | 0 |
| Update | addrLocal = 4095 | Finish | 0 | 0 | 0 | 0 |
| Update | Không thỏa trường hợp trên | setRam | 0 | 0 | 0 | 0 |

* 1. **Module master control**

Module Master Control đóng vai trò là bộ điều phối trung tâm cho một hệ thống nhiều module (Bao gồm những khối chính về fetching memory, store data camera to SDRAM, convolution, average compute, write back). Nó điều khiển kích hoạt tuần tự các module con chẳng hạn như ghi nhận dữ liệu từ camera, thực hiện phép tích chập (conv), tính trung bình (average, averageControl), và ghi kết quả về bộ nhớ (writeBackControl)—nhằm đảm bảo sự đồng bộ và luồng dữ liệu đúng đắn và tránh cung đột nhau khi sử dụng chung một loại tài nguyên ví dụ như SDRAM, source RAM, destination Ram,....

Thông qua việc giải mã opcode và quản lý các địa chỉ cơ sở (thông qua baseAddrFetDecode và baseAddrWriteBackDecode), module này định tuyến dữ liệu giữa RAM, SDRAM và các khối xử lý, đồng thời xử lý các ràng buộc về thời gian. Bộ điều khiển sử dụng máy trạng thái (FSM) để kích hoạt các thao tác như lấy ảnh từ camera, lọc tích chập, tính trung bình, và lưu trữ kết quả; đồng thời giám sát các tín hiệu handshake (ví dụ: i\_sdramReady, i\_validRam, o\_finish) để tránh xung đột và đảm bảo tính hoàn tấc của quá trình xữ lý.

Giao diện điều khiển của khối master control lên các module tiến trình thành phần dựa trên kỹ thuật handshake rút gọn gồm tín hiệu bắt đầu (start) – hay tín hiệu khởi động được cấp từ master control và tín hiệu hoàn thành (finish) – hay tín hiệu kết thúc tiến trình từ module chức năng thành phần.

A diagram of a wiring diagram

AI-generated content may be incorrect.

Hình 30: Minh họa giao tiếp module master control và các module control tiến trình chức năng thành phần

Bảng 22: Danh sách input output của module master control

|  |  |  |  |
| --- | --- | --- | --- |
| Port name | In/out | Width | Discription |
| I\_clk | In | 1 | Xung clock |
| I\_reset | In | 1 | Reset signal active low |
| I\_finish\_cam | In | 1 | Tín hiệu báo hoàn thành lưu frame từ camera |
| I\_finish\_fet | In | 1 | Tín hiệu báo hoàn thành fet |
| I\_finish\_conv | In | 1 | Tín hiệu báo hoàn thành conv |
| I\_finish\_writeBack | In | 1 | Tín hiệu báo hoàn thành write back |
| I\_finish\_ave | In | 1 | Tín hiệu báo hoàn thành tính trung bình |
| O\_startCam | Out | 1 | Tín hiệu khởi động khối đọc camera lưu vào SDRAM |
| O\_startFet | Out | 1 | Tín hiệu khởi động khối đọc Fet |
| O\_startConvolution | Out | 1 | Tín hiệu khởi động khối convolution |
| O\_startAve | Out | 1 | Tín hiệu khỏi động khối tính trung bình AVE |
| O\_startWriteBack | Out | 1 | Tín hiệu khởi động khối write back |
| O\_wrActiveCam | Out | 1 | Tín hiệu ưu tiên luồng dữ liệu ghi vào SDRAM |
| O\_opConv | Out | 1 | Tín hiệu điều khiển loại convolution 3 feature hoặc 1 feature |
| O\_opcode | Out | 6 | Opcode tín hiệu xác định feature mục tiêu. |

Bảng 23: Bảng chuyển trạng thái master control

|  |  |  |  |
| --- | --- | --- | --- |
| Trạng thái  hiện tại | Điều kiện | Trạng thái  kế tiếp | Mô tả |
| Idle | Luôn luôn | startCam | Trạng thái nghĩ |
| startCam | Luôn luôn | waitdoneCam | Bậc cờ khởi động readCam |
| waitDoneCam | I\_finish\_cam | Fetch1 | Chờ cam xong |
| Fetch1 | Luôn luôn | waitDoneFet1 | Bậc cờ khởi động fetch |
| waitdoneFet1 | I\_fnish\_fet | Convolution0 | Chờ fet xong |
| Convolution0 | Luôn luôn | waitDoneConv0 | Bậc cờ khởi động convolution 3 feature |
| waitDoneConv0 | I\_finish\_conv | writeBack1 | Chờ convolution 3 feature xong |
| Writeback1 | Luôn luôn | waitDoneWriteback1 | Bậc cờ writeback |
| waitDoneWriteBack1 | I\_finish\_writeBack | Update1 | Chờ write back xong |
| Update 1 | Opcode = 15 | Fet2 | Cập nhật opcode kiểm ra chuyển sang tiến trình khác |
| Update 1 | Opcode < 15 | Convolution0 | Trở lại thực hiện feature tiếp theo |
| Fet2 | Luôn luôn | waitdoneFet2 | Khởi động tiến trình fet |
| waitDoneFet2 | I\_finish\_fet | Convolution1 | Chờ hoàn thành fet |
| Convolution1 | Luôn luôn | waitDoneConvolution1 | Khởi động khối convolution 1 feature |
| waitdoneConvolution1 | I\_finish\_conv | writeBack2 | Chờ tiến trình convolution 1 feature hoàn thành |
| writeBack2 | Luôn luôn | waitDoneWriteback2 | Khởi động khối write back |
| waitDoneWriteBack2 | I\_finish\_writeback | Update2 | Chờ tiến trình writeback hoàn thành |
| Update2 | Opcode = 31 | Fet3 | Cập nhật opcode kiểm tra chuyển đến tiến trình khác |
| Update2 | Opcode < 31 | Fet2 | Cập nhật opcode chuyển về và thực hiện feature tiếp theo |
| Fet3 | Luôn luôn | waitDoneFet3 | Khởi động tiến trình fet |
| waitDoneFet3 | I\_finish\_fet | Average | Chờ fet xong |
| Average | Luôn luôn | waitDoneAve | Khởi động tiến trình tính trung bình |
| waitDoneAve | I\_finish\_ave | Update3 | Chờ hoàn thành tính average |
| Update3 | Opcode = 37 | Idle | Hoàn thành tính toán một frame |
| Update | Opcode < 37 | Fet3 | Thực hiện tiếp feature tiếp theo |

Ánh xạ địa chỉ của feature được tổ chức như sau: Trong đó 3 feature đầu tiên index 0, 1, 2 là 3 kênh ảnh đầu vào (RGB).

A table with numbers and a number on it

AI-generated content may be incorrect.

Hình 31: Ánh xạ địa chỉ feature 1

A table with numbers and a number in it

AI-generated content may be incorrect.

Hình 32: Ánh xạ địa chỉ feature 2

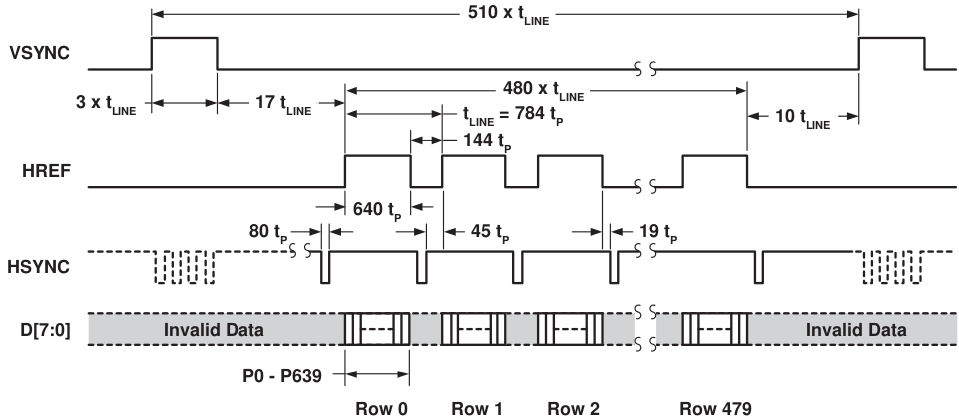
Ánh xạ địa chỉ có ý nghĩa khi nói đến feature có index là 60 thì ta sẽ biết feature này sẽ được lưu trong bộ nhớ SDRAM có địa chỉ bắt đầu từ địa chỉ 245760 (địa chỉ hệ 10) và kết thúc tại địa chỉ 249855 (địa chỉ hệ 10). Khi nói đến pixel thứ 10 của pixel map có index 60 thì có nghĩa là địa chỉ của pixel ấy là 245760 + 10 = 245770 (Địa chỉ hệ 10). Như vậy, ta sẽ dựa vào bảng ánh xạ địa chỉ này để truy cập đến pixel của một feature bấc kỳ.

Trong phần khối master control, có một khối phụ là khối weight generate. Mặc dù là khối phụ nhưng đây là khối rất quan trọng tạo ra các trọng số cho khối convolution hoạt động. Module này nhận vào giá trị opcode của khối master control (6bit) Tra ROM và xuất weight qua 3 ngõ ra 90 bit.

1. **VERIFICATION**

**7.1 Kiểm tra khối camera read và lưu dữ liệu vào FIFO**

Để kiểm thử khối đọc dữ liệu từ camera (OV7670) ta cần mô phỏng lại dạng sóng của các ngõ ra của khối camera OV7670.



Dạng sóng ngõ ra của camera OV7670 được cung cấp theo datasheet của camera. Camera Ov7670 có cơ chế đồng bộ bao gồm VSYNC bấc lên 1 báo hiệu 1 frame mới bắt đầu. Pixel chỉ valid khi HREF ở mức cao (đồng bộ hàng)

A screenshot of a computer

AI-generated content may be incorrect.

Hình 33: Mô phỏng waveform của camera Ov7670

Khối camera read sẽ đưa ra giá trị pixel 16 bit, địa chỉ X\_index, Y\_index. Ta thực hiện testbench theo luận lý. Giá trị pixel bằng X\_index của pixel – 1. Valid pixel bằng tín hiệu valid. Chỉ ghi và xét dữ liệu khi tín hiệu Valid = 1

A screenshot of a computer

AI-generated content may be incorrect.

Luận lý chốt pixel của sẽ dựa vào tín hiệu o\_valid và tín hiệu o\_pclk. Ta chỉ chốt lấy pixel tại vị trí o\_pclk = 1 và o\_valid = 1.

A screenshot of a computer program

AI-generated content may be incorrect.

Hình 34: Chạy testbench cho khối camera read

Nhận xét: module hoạt động chính xác giá trị pixel valid và có giá trị bằng X\_index-1.

Testbench khối control write to FIFO: Dữ liệu chỉ có thể ghi vào FIFO khi pixel nằm trong kích thước 64x64. Khi i\_get = 1 và x\_index = 0, y\_index =0 thì module sẽ tiến vào trạng thái process. Chỉ khi nào i\_index và i\_index thuộc trong kích thước 64x64 thì wrFIFO mới được phép bậc 1, ta sẽ dùng những ràng buộc này thực hiện kiểm tra hoạt động của khối.

A screenshot of a computer

AI-generated content may be incorrect.

Hình 35: Testbench cho module control write to FIFO case 1

Nhận xét: Module hoạt động đúng trong case khởi động. Chỉ tiến vào process khi i\_get bậc 1 và x\_index = 0, y\_index = 0. Và writeENfifo bậc 1 khi pixel thuộc 64x64.

A screenshot of a computer program

AI-generated content may be incorrect.

Hình 36: Testbench cho module control write to FIFO case2

Nhận xét: Module hoạt động đúng, Pixel x khi out khỏi 64x64. WriteEn đã được tắt đúng như kỳ vọng.

A screenshot of a computer

AI-generated content may be incorrect.

Hình 37: Testbench cho module control write to FIFO case 3

Nhận xét: Module hoạt động đúng. Khi y đạt 63 hàng cuối trong 64x64 và x bằng 64 thì module chuyển sang trạng thái complete, ngắt writeEn.

## 7.2 Testbench module fetching memory

Trong testbench module fetching memory. Ta sẽ thực hiện theo luận lý mô phỏng latency của SDRAM read operation. Cho đầu vào base address lần lược là 0, 10000, 20000. Ta bắt tín hiệu rdSdram, wrRam để xuất hành vi hệ thống. Mô phỏng tín hiệu i\_start để khởi động tiến trình, o\_finish báo hiệu hoàn thành tiến trình.

Giao diện tín hiệu để đọc dữ liệu từ sdram khá đơn giản. Chỉ cần tín hiệu rdSdram bậc 1 thì SDRAM sẽ bắt đầu thực thi lệnh đọc trong chu kỳ clock tiếp theo.

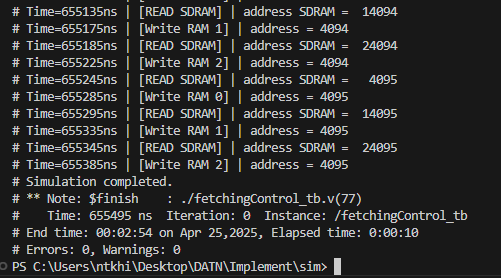
A screen shot of a video game

AI-generated content may be incorrect.

Hình 38: Waveform của bộ fetching control

Sau khi SDRAM thực hiện lệnh đọc xong và dữ liệu đã sẵn sàng tại đầu ra thì bộ SDRAM control sẽ bậc cờ sdramReady lên 1 và cờ này chỉ tắc khi SDRAM thực hiện một câu lệnh mới.

Việc ghi dữ liệu vào RAM cũng rất đơn giản. Ta chỉ cần bậc cờ o\_wrRAM0, o\_wrRAM1 hoặc o\_wrRAM2 tương ứng để thực hiện ghi vào chu kỳ clock tiếp theo. Vì ta có 3 source RAM vì vậy một chù kỳ FET sẽ có 3 chu kỳ đọc dữ liệu từ SDRAM và lưu dữ liệu vào RAM tương ứng theo thứ tự gồm Source\_ram0, Source\_ram1 và Source\_ram2 sau khi đã thực hiện hoàn thành 1 chu kỳ đọc ghi cho cả 3 source ram, bộ fetching control sẽ tiếp tục đến chu kỳ fet thứ 2 và tăng địa chỉ lên 1. Nhầm đọc ghi cho pixel tiếp theo của feature map. Và sau khi hoàn thành pixel cuối cùng (pixel thứ 4095) của feature map thì cờ o\_finish sẽ được bậc lên nhầm báo hiệu cho bộ control master biết quá trình fetching đã hoàn thành và có thể thực hiện công đoạn tiếp theo.



Hình 39: Testbench cho module fetching memory.

Nhận xét: Module hoạt động chính xác theo thứ tự thực hiện kỳ vọng. Đọc sdram cho cho Ram đầu tiên, ghi vào RAM 0, Đọc SDRAM cho Ram thứ 2, ghi vào Ram2, đọc SDRAM cho ram 3, ghi vào RAM3. Sao khi hoàn thành pixel cuối (base address 4095) thì kết thúc tiến trình.

## 7.3 Testbench module write back

Thực hiện luận lý testbench giống như testbench của khối fetching memory. Giao diện ghi của bộ SDRAM control rất đơn giản. Bậc cờ wrSdram lên 1 và SDRAM sẽ thực thi lệnh ghi trong cho kỳ clock tiếp theo, giao diện đọc của Desination Ram ta thực hiện bậc cờ quickMode lên 1 thực hiện lệnh đọc nhanh của Destination Ram, ba bộ destination ram sẽ được thực hiện lệnh RAM cũng một thời điểm. Trong một chu kỳ write back bộ control write back sẽ thực hiện đọc 3 luồng dữ liệu sẽ được chọn bởi tín hiệu o\_selData gồm 2 bit chọn luồng 0, luồng 1 hoặc luồng 1. Dữ liệu sẽ được ghi vào SDRAM lần lượt theo thứ tự ghi luồng 0 vào sdram trước, sau khi tín hiệu sdramReady bậc 1 thì tiếp tục ghi tiếp luồng 1 vào sdram, cuối cùng là luồng 2. Sau khi hoàn thành một chu kỳ write back, bộ sdram control sẽ thực hiện tiếp chu kỳ tiếp theo địa chỉ sẽ được cộng lên 1, sau khi hoàn thành write back cho toàn bộ pixel của feature map (4095 pixel) thì cờ o\_finish sẽ được bậc lên 1 báo hiệu hoàn thành tiến trình write back.

A screen shot of a computer

AI-generated content may be incorrect.

Hình 40: Waveform của bộ control write back

A screenshot of a computer

AI-generated content may be incorrect.

Hình 41: Testbench cho module Write back

Nhận xét: Module hoạt động đúng thứ tự kỳ vọng là đọc dữ liệu từ 3 ram trong cùng một chu kỳ với mode quick read. Thực hiện ghi vào sdram cho ram 0 trước, tiếp theo là ram 1 và cuối cùng là Ram2. Pixel cuối cùng là 4095, sau khi hoàn thành pixel này module sẽ bậc cờ finish và hoàn thành tiến trình về lại trạng thái idle.

**7.4 Testbench module conv**

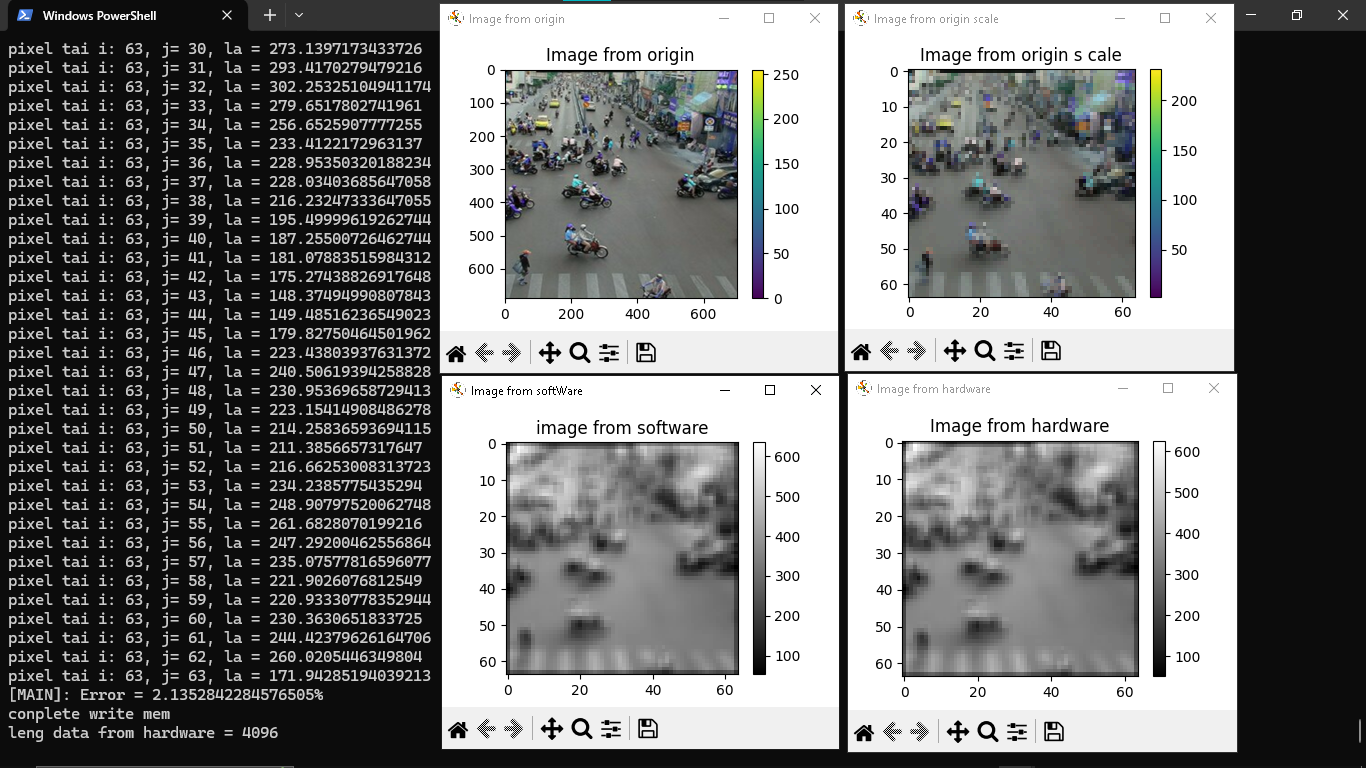
Để thực hiện testbench cho khối conv (đầy cũng là khối chức năng quan trọng nhất của mô hình) Ta thực hiện xây dựng mô hình tương tự trên máy tính tuy nhiên sử dụng số chấm động (floating point) và so sánh sai số giữ mô hình trên máy tính sử dụng floating point và mô hình hardware sử dụng số chấm tĩnh (fixing piont) từ đó đánh giá độ chính xác của mô hình.

Bước 1: Chuẩn bị bộ trọng số của mô hình được xây dựng trên máy tính và scale lại thành số 10 bit định dạng Q0.9 như đã thiết kế trong bộ conv lưu bộ weight vào một file hex.

Bước 2: Chuẩn bị bộ pixel từ ảnh mẫu, pixel của ảnh mẫu có giá trị từ 0 đến 255 sẽ được scale về 0 đến 1. Đối với đầu vào trên máy tính ta giữ nguyên định dạng số thực floating point của pixel, mặc khác feature map được đưa đến bộ conv sẽ được scale và lượng tử về định dạng fixing point Q3.6 và lưu feature vào file hex. Chúng ta có hai loại phép convolution là 3 feature và 1 feature. Vì phép convolution 3 feature có nhiều phép nhân hơn và phức tạp hơn nên có độ sai số lớn hơn so với phép convolution 1 feature vì vậy ta sẽ thử nghiệm đo sai số với phép convolution 3 feature.

Bước 3: Chạy testbench hardware verilog xuất feature đầu ra qua file .hex

Bước 4: Chạy chương trình python load feature đầu ra từ testbench verilog và so sánh với kết quả từ mô hình mô phỏng trên máy tính sử dụng floating point và tính toàn sai số.



Hình 42: Kết quả so sánh phần mềm dùng floating point và phần cứng dùng fixing point

Nhận xét: Chỉ bằng mắt thường ta sẽ khó nhận ra sự khác biệt giữa mô hình dùng floating point trên máy tính và mô hình dùng fixing point trên phần cứng (testbench verilog). Tổng sai số của giữ hai mô hình là 2.135% (Chấp nhận được)

## 7.5 Testbench module master control.

Khối master control là module rất quan trọng trong hệ thống, Vì module này có chức năng cung cấp các tín hiệu khởi động, các tín hiệu điều khiển giúp phân luồng dữ liệu tránh xung đột giữa các module chức năng. Đồng thời nhận tín hiệu complete từ các module chức năng để hoàn thành tiến trình hiệu tại và chuyển đến tiến trình tiếp theo.

Logic của testbench module master control sẽ là mô phỏng lại hoạt động của các khối chức năng bằng cách khi nhận được tín hiệu start từ module master control thì module mô phỏng sẽ delay lại một số chu kỳ trước khi bậc cờ complete để trả về cho module master control. Trong lúc đó ta sẽ quan sát các tín hiệu điều khiển của khối master control để chắc chắn rằng khối master control sẽ xuất tín hiệu điều khiển chính xác để phân luồng dữ liệu và tránh xung đột.

Kết quả chạy testbench khối master control như sau:

|  |
| --- |
| # vsim work.masterControl\_tb -c -do "run -all; exit"  # Start time: 18:38:12 on Apr 27,2025  # Loading work.masterControl\_tb  # Loading work.masterControl  # run -all  # | with opcode = 0 | write priority CAM = 0 | at time = 35 | [Cam start signal on]  # | with opcode = 0 | at time = 3035 | [Cam complete]  # | with opcode = 0 | at time = 3055 | [Fetching start]  # | with opcode = 0 | at time = 3305 | [Fetching complete]  # | with opcode = 0 | opcode convolution= 0 | priority read RAM for convolution = 1 | at time = 3325 | [convolution start]  # | with opcode = 0 | at time = 3625 | [convolution complete]  # | with opcode = 0 | at time = 3645 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 0 | at time = 3915 | [WriteBack complete]  # | with opcode = 1 | opcode convolution= 0 | priority read RAM for convolution = 1 | at time = 3945 | [convolution start]  # | with opcode = 1 | at time = 4245 | [convolution complete]  # | with opcode = 1 | at time = 4265 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 1 | at time = 4535 | [WriteBack complete]  # | with opcode = 2 | opcode convolution= 0 | priority read RAM for convolution = 1 | at time = 4565 | [convolution start]  # | with opcode = 2 | at time = 4865 | [convolution complete]  # | with opcode = 2 | at time = 4885 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 2 | at time = 5155 | [WriteBack complete]  # | with opcode = 3 | opcode convolution= 0 | priority read RAM for convolution = 1 | at time = 5185 | [convolution start]  # | with opcode = 3 | at time = 5485 | [convolution complete]  # | with opcode = 3 | at time = 5505 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 3 | at time = 5775 | [WriteBack complete]  # | with opcode = 4 | opcode convolution= 0 | priority read RAM for convolution = 1 | at time = 5805 | [convolution start]  # | with opcode = 4 | at time = 6105 | [convolution complete]  # | with opcode = 4 | at time = 6125 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 4 | at time = 6395 | [WriteBack complete]  # | with opcode = 5 | opcode convolution= 0 | priority read RAM for convolution = 1 | at time = 6425 | [convolution start]  # | with opcode = 5 | at time = 6725 | [convolution complete]  # | with opcode = 5 | at time = 6745 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 5 | at time = 7015 | [WriteBack complete]  # | with opcode = 6 | opcode convolution= 0 | priority read RAM for convolution = 1 | at time = 7045 | [convolution start]  # | with opcode = 6 | at time = 7345 | [convolution complete]  # | with opcode = 6 | at time = 7365 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 6 | at time = 7635 | [WriteBack complete]  # | with opcode = 7 | opcode convolution= 0 | priority read RAM for convolution = 1 | at time = 7665 | [convolution start]  # | with opcode = 7 | at time = 7965 | [convolution complete]  # | with opcode = 7 | at time = 7985 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 7 | at time = 8255 | [WriteBack complete]  # | with opcode = 8 | opcode convolution= 0 | priority read RAM for convolution = 1 | at time = 8285 | [convolution start]  # | with opcode = 8 | at time = 8585 | [convolution complete]  # | with opcode = 8 | at time = 8605 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 8 | at time = 8875 | [WriteBack complete]  # | with opcode = 9 | opcode convolution= 0 | priority read RAM for convolution = 1 | at time = 8905 | [convolution start]  # | with opcode = 9 | at time = 9205 | [convolution complete]  # | with opcode = 9 | at time = 9225 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 9 | at time = 9495 | [WriteBack complete]  # | with opcode = 10 | opcode convolution= 0 | priority read RAM for convolution = 1 | at time = 9525 | [convolution start]  # | with opcode = 10 | at time = 9825 | [convolution complete]  # | with opcode = 10 | at time = 9845 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 10 | at time = 10115 | [WriteBack complete]  # | with opcode = 11 | opcode convolution= 0 | priority read RAM for convolution = 1 | at time = 10145 | [convolution start]  # | with opcode = 11 | at time = 10445 | [convolution complete]  # | with opcode = 11 | at time = 10465 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 11 | at time = 10735 | [WriteBack complete]  # | with opcode = 12 | opcode convolution= 0 | priority read RAM for convolution = 1 | at time = 10765 | [convolution start]  # | with opcode = 12 | at time = 11065 | [convolution complete]  # | with opcode = 12 | at time = 11085 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 12 | at time = 11355 | [WriteBack complete]  # | with opcode = 13 | opcode convolution= 0 | priority read RAM for convolution = 1 | at time = 11385 | [convolution start]  # | with opcode = 13 | at time = 11685 | [convolution complete]  # | with opcode = 13 | at time = 11705 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 13 | at time = 11975 | [WriteBack complete]  # | with opcode = 14 | opcode convolution= 0 | priority read RAM for convolution = 1 | at time = 12005 | [convolution start]  # | with opcode = 14 | at time = 12305 | [convolution complete]  # | with opcode = 14 | at time = 12325 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 14 | at time = 12595 | [WriteBack complete]  # | with opcode = 15 | opcode convolution= 0 | priority read RAM for convolution = 1 | at time = 12625 | [convolution start]  # | with opcode = 15 | at time = 12925 | [convolution complete]  # | with opcode = 15 | at time = 12945 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 15 | at time = 13215 | [WriteBack complete]  # | with opcode = 16 | at time = 13245 | [Fetching start]  # | with opcode = 16 | at time = 13495 | [Fetching complete]  # | with opcode = 16 | opcode convolution= 1 | priority read RAM for convolution = 1 | at time = 13515 | [convolution start]  # | with opcode = 16 | at time = 13815 | [convolution complete]  # | with opcode = 16 | at time = 13835 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 16 | at time = 14105 | [WriteBack complete]  # | with opcode = 17 | at time = 14135 | [Fetching start]  # | with opcode = 17 | at time = 14385 | [Fetching complete]  # | with opcode = 17 | opcode convolution= 1 | priority read RAM for convolution = 1 | at time = 14405 | [convolution start]  # | with opcode = 17 | at time = 14705 | [convolution complete]  # | with opcode = 17 | at time = 14725 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 17 | at time = 14995 | [WriteBack complete]  # | with opcode = 18 | at time = 15025 | [Fetching start]  # | with opcode = 18 | at time = 15275 | [Fetching complete]  # | with opcode = 18 | opcode convolution= 1 | priority read RAM for convolution = 1 | at time = 15295 | [convolution start]  # | with opcode = 18 | at time = 15595 | [convolution complete]  # | with opcode = 18 | at time = 15615 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 18 | at time = 15885 | [WriteBack complete]  # | with opcode = 19 | at time = 15915 | [Fetching start]  # | with opcode = 19 | at time = 16165 | [Fetching complete]  # | with opcode = 19 | opcode convolution= 1 | priority read RAM for convolution = 1 | at time = 16185 | [convolution start]  # | with opcode = 19 | at time = 16485 | [convolution complete]  # | with opcode = 19 | at time = 16505 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 19 | at time = 16775 | [WriteBack complete]  # | with opcode = 20 | at time = 16805 | [Fetching start]  # | with opcode = 20 | at time = 17055 | [Fetching complete]  # | with opcode = 20 | opcode convolution= 1 | priority read RAM for convolution = 1 | at time = 17075 | [convolution start]  # | with opcode = 20 | at time = 17375 | [convolution complete]  # | with opcode = 20 | at time = 17395 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 20 | at time = 17665 | [WriteBack complete]  # | with opcode = 21 | at time = 17695 | [Fetching start]  # | with opcode = 21 | at time = 17945 | [Fetching complete]  # | with opcode = 21 | opcode convolution= 1 | priority read RAM for convolution = 1 | at time = 17965 | [convolution start]  # | with opcode = 21 | at time = 18265 | [convolution complete]  # | with opcode = 21 | at time = 18285 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 21 | at time = 18555 | [WriteBack complete]  # | with opcode = 22 | at time = 18585 | [Fetching start]  # | with opcode = 22 | at time = 18835 | [Fetching complete]  # | with opcode = 22 | opcode convolution= 1 | priority read RAM for convolution = 1 | at time = 18855 | [convolution start]  # | with opcode = 22 | at time = 19155 | [convolution complete]  # | with opcode = 22 | at time = 19175 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 22 | at time = 19445 | [WriteBack complete]  # | with opcode = 23 | at time = 19475 | [Fetching start]  # | with opcode = 23 | at time = 19725 | [Fetching complete]  # | with opcode = 23 | opcode convolution= 1 | priority read RAM for convolution = 1 | at time = 19745 | [convolution start]  # | with opcode = 23 | at time = 20045 | [convolution complete]  # | with opcode = 23 | at time = 20065 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 23 | at time = 20335 | [WriteBack complete]  # | with opcode = 24 | at time = 20365 | [Fetching start]  # | with opcode = 24 | at time = 20615 | [Fetching complete]  # | with opcode = 24 | opcode convolution= 1 | priority read RAM for convolution = 1 | at time = 20635 | [convolution start]  # | with opcode = 24 | at time = 20935 | [convolution complete]  # | with opcode = 24 | at time = 20955 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 24 | at time = 21225 | [WriteBack complete]  # | with opcode = 25 | at time = 21255 | [Fetching start]  # | with opcode = 25 | at time = 21505 | [Fetching complete]  # | with opcode = 25 | opcode convolution= 1 | priority read RAM for convolution = 1 | at time = 21525 | [convolution start]  # | with opcode = 25 | at time = 21825 | [convolution complete]  # | with opcode = 25 | at time = 21845 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 25 | at time = 22115 | [WriteBack complete]  # | with opcode = 26 | at time = 22145 | [Fetching start]  # | with opcode = 26 | at time = 22395 | [Fetching complete]  # | with opcode = 26 | opcode convolution= 1 | priority read RAM for convolution = 1 | at time = 22415 | [convolution start]  # | with opcode = 26 | at time = 22715 | [convolution complete]  # | with opcode = 26 | at time = 22735 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 26 | at time = 23005 | [WriteBack complete]  # | with opcode = 27 | at time = 23035 | [Fetching start]  # | with opcode = 27 | at time = 23285 | [Fetching complete]  # | with opcode = 27 | opcode convolution= 1 | priority read RAM for convolution = 1 | at time = 23305 | [convolution start]  # | with opcode = 27 | at time = 23605 | [convolution complete]  # | with opcode = 27 | at time = 23625 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 27 | at time = 23895 | [WriteBack complete]  # | with opcode = 28 | at time = 23925 | [Fetching start]  # | with opcode = 28 | at time = 24175 | [Fetching complete]  # | with opcode = 28 | opcode convolution= 1 | priority read RAM for convolution = 1 | at time = 24195 | [convolution start]  # | with opcode = 28 | at time = 24495 | [convolution complete]  # | with opcode = 28 | at time = 24515 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 28 | at time = 24785 | [WriteBack complete]  # | with opcode = 29 | at time = 24815 | [Fetching start]  # | with opcode = 29 | at time = 25065 | [Fetching complete]  # | with opcode = 29 | opcode convolution= 1 | priority read RAM for convolution = 1 | at time = 25085 | [convolution start]  # | with opcode = 29 | at time = 25385 | [convolution complete]  # | with opcode = 29 | at time = 25405 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 29 | at time = 25675 | [WriteBack complete]  # | with opcode = 30 | at time = 25705 | [Fetching start]  # | with opcode = 30 | at time = 25955 | [Fetching complete]  # | with opcode = 30 | opcode convolution= 1 | priority read RAM for convolution = 1 | at time = 25975 | [convolution start]  # | with opcode = 30 | at time = 26275 | [convolution complete]  # | with opcode = 30 | at time = 26295 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 30 | at time = 26565 | [WriteBack complete]  # | with opcode = 31 | at time = 26595 | [Fetching start]  # | with opcode = 31 | at time = 26845 | [Fetching complete]  # | with opcode = 31 | opcode convolution= 1 | priority read RAM for convolution = 1 | at time = 26865 | [convolution start]  # | with opcode = 31 | at time = 27165 | [convolution complete]  # | with opcode = 31 | at time = 27185 | write priority CAM = 0 | [WriteBack start]  # | with opcode = 31 | at time = 27455 | [WriteBack complete]  # | with opcode = 32 | at time = 27485 | [Fetching start]  # | with opcode = 32 | at time = 27735 | [Fetching complete]  # | with opcode = 32 | opcode convolution= 0 | priority read RAM for convolution = 0 | at time = 27755 | [Average compute start]  # | with opcode = 32 | at time = 27935 | [Average compute complete]  # | with opcode = 33 | at time = 27965 | [Fetching start]  # | with opcode = 33 | at time = 28215 | [Fetching complete]  # | with opcode = 33 | opcode convolution= 0 | priority read RAM for convolution = 0 | at time = 28235 | [Average compute start]  # | with opcode = 33 | at time = 28415 | [Average compute complete]  # | with opcode = 34 | at time = 28445 | [Fetching start]  # | with opcode = 34 | at time = 28695 | [Fetching complete]  # | with opcode = 34 | opcode convolution= 0 | priority read RAM for convolution = 0 | at time = 28715 | [Average compute start]  # | with opcode = 34 | at time = 28895 | [Average compute complete]  # | with opcode = 35 | at time = 28925 | [Fetching start]  # | with opcode = 35 | at time = 29175 | [Fetching complete]  # | with opcode = 35 | opcode convolution= 0 | priority read RAM for convolution = 0 | at time = 29195 | [Average compute start]  # | with opcode = 35 | at time = 29375 | [Average compute complete]  # | with opcode = 36 | at time = 29405 | [Fetching start]  # | with opcode = 36 | at time = 29655 | [Fetching complete]  # | with opcode = 36 | opcode convolution= 0 | priority read RAM for convolution = 0 | at time = 29675 | [Average compute start]  # | with opcode = 36 | at time = 29855 | [Average compute complete]  # | with opcode = 37 | at time = 29885 | [Fetching start]  # | with opcode = 37 | at time = 30135 | [Fetching complete]  # | with opcode = 37 | opcode convolution= 0 | priority read RAM for convolution = 0 | at time = 30155 | [Average compute start]  # | with opcode = 37 | at time = 30335 | [Average compute complete]  # | with opcode = 0 | write priority CAM = 0 | at time = 30375 | [Cam start signal on]  # All 38 operations completed 1 frame successfully!  # \*\* Note: $finish : ./masterControl\_tb.v(102)  # Time: 31345 ns Iteration: 0 Instance: /masterControl\_tb  # End time: 18:38:12 on Apr 27,2025, Elapsed time: 0:00:00  # Errors: 0, Warnings: 0 |

Như ta có thể thấy testbench sẽ xuất các lệnh ví dụ start cam, start Fetching. Kèm theo thời gian lệnh xuất hiện. Các lệnh xuất hiện đúng theo thư tự mà ta kỳ vọng. Không có khối chức năng nào sử dụng chung một loại tài nguyên được active tại cùng một thời điểm. Chống xung đột tranh tài nguyên giữa các khối chức năng. Các tín hiệu điều khiển cho các khối dùng chung tài nguyên như: o\_wrActiveCam – tín hiệu điều khiển ưu tiên dùng SDRAM cho Camera read được bậc lên khi khối Cam read được active và tắc đi khi khối write back được active. o\_rdActiveConvlotuion – tín hiệu điều khiển ưu tiên dùng RAM cho convolution block được bậc lên khi khối convolution được active và tắt đi khi khối average active. Như vậy module master control xuất tín hiệu control chính xác để điều khiển luồng cho dữ liệu.

Sau khi hoàn thành opcode cuối cùng. Như ta có thể thấy module tiếp tục với việc start cam read cho việc xữ lý frame tiếp theo mà không ngừng lại.

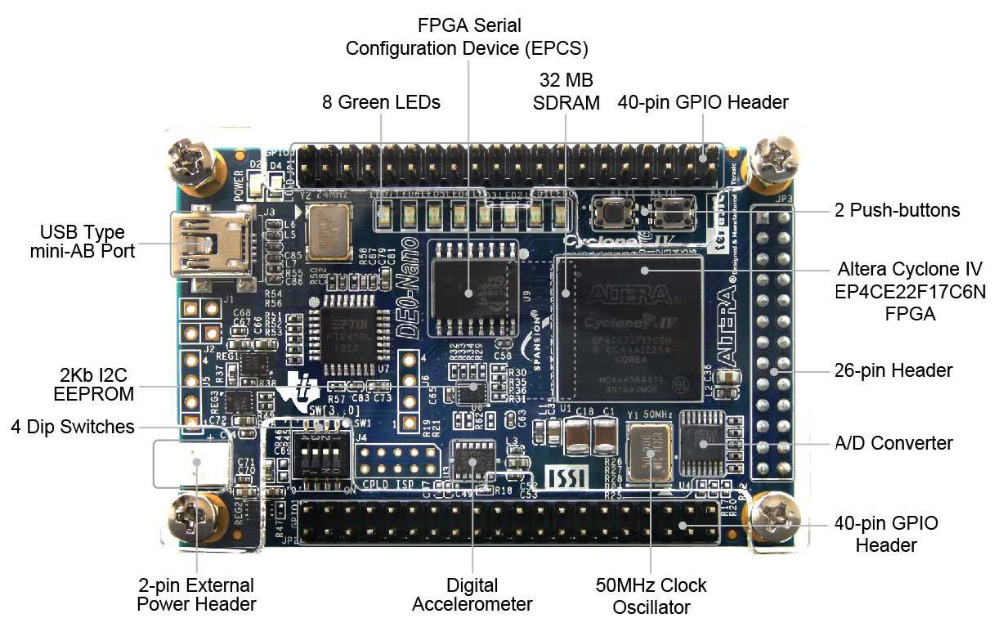
1. **SYNTHESIS** 
   1. **Tìm hiều về kít DE0-Nano**

FPGA Chip được sử dụng trên kit DE0-Nano: Altera Cyclone IV EP4CE22F17C6N

Các thông số quan trọng nhất liên quan trực tiếp đến project bao gồm:

Bảng 24: Thông số tài nguyên của Altera Cyclone IV EP4CE22F17C6N

|  |  |
| --- | --- |
| Resource | Available |
| Logic Elements (LEs) | 22,320 |
| Logic Register | 22.320 |
| Memory Bits (Block RAM) | 594,432 bit |
| Embedded Multipliers (18x18) | 66 |
| PLL (Phase-Locked Loops) | 4 |
| Maximum User I/O pin | 150 |



Hình 43: Hình ảnh thực tế Kit DE0-nano

Như ta có thể thấy Kit có kích thước tương đối nhỏ so với một số loại KIT FPGA khác nên sẽ phù hợp với các ứng dụng mang tính di động, ưu tiên sự gọn nhẹ.

Project của chúng ta còn một thông số tài nguyên rất quan trọng không nằm trên CHIP FPGA. Đó chính là SDRAM – bộ nhớ lưu trữ toàn bộ feature của mô hình mobile net. Trên kit DE0-nano được kèm sẵn chip SDRAM 32Mb onboard có thông số như sau:

Bảng 25: Thông số kỹ thuật SDRAM trên kit DE0-Nano

|  |  |
| --- | --- |
| Feature | Specification |
| Chip Part Number | ISSI IS42S16320F |
| Memory Type | Synchronous DRAM (SDRAM) |
| Memory Size | 32Mbyte (256Mbit = 2Mx16x8) |
| Data Bus Width | 16 bit |
| Address Bus Width | 13 bit |
| Row Address | A0-A12 (13bit) |
| Column Address | A0-A8 (9bit) |
| Band Address | BA0-BA1: 4 banks |
| Max Clock Frequency | 133Mhz |
| CAS Latency | 2 or 3 |
| Interface | Standard parallel SDRAM interface |

* 1. **Synthesis project**

Bởi về project vẫn chưa hoàn thiện đầy đủ. Vì vậy chỉ có thể synthesis từng module đã thiết kế và thực hiện testbenh thành công để đánh giá mức độ khả thi khi thực hiện mô hình mobile net trên FPGA (kit DE0-nano)

A screenshot of a computer

AI-generated content may be incorrect.

Hình 44: Kết quả synthesis module RAM

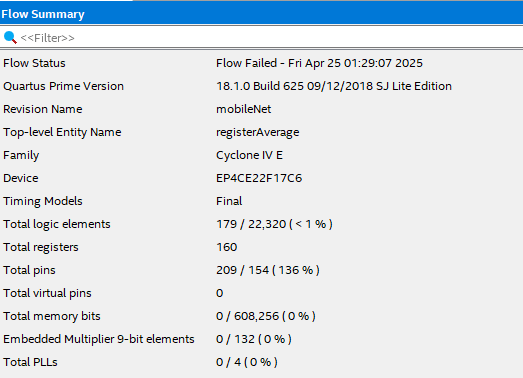
Khối RAM là khối sử dụng nhiều tài nguyên bộ nhớ nhất nên ta cần quan tâm đến tổng memory bits tài nguyên đã sử dụng. Ta nhận thấy rằng 1 block RAM sẽ sử dụng khoảng 7%. Theo diagram đã thiết kế hệ thông cần dùng 6 RAM tương tự nên tài nguyên bộ nhớ tiêu thụ cho tổng bộ RAM là: 6\*7%= 42% Hoàn toàn khả thi về tài nguyên bộ nhớ.

A screenshot of a computer

AI-generated content may be incorrect.

Hình 45: Kết quả synthesis khối convolution

Khối convolution được synthesis khả thi và tiêu thụ tài nguyên tính toán Embedded Multipler tương ứng 41%



Hình 46: Kết quả synthesis khối aveage register

Module này không tiêu tốn thêm memory bit. Chỉ tiêu tốn thanh ghi với số lượng ít. Nên hoàn toàn khả thi.

A screenshot of a computer

AI-generated content may be incorrect.

Hình 47: Kết quả systhesis module weight generate

Module weight generate cũng hoàn toàn khã thi tuy có số lượng weight tương đối nhưng module không tiếp tốn thên memory bit, Chỉ dùng logic elements khoảng 5%. Hoàn toàn khả thi trên FPGA.

A screenshot of a computer

AI-generated content may be incorrect.

Hình 48: Kết quả synthesis module master control

Ta có thể nhận thấy rằng module master control có độ phức tạp cao nhất, nhiều state nhất. Tuy nhiên module cũng không tiêu tốn quá nhiều tài nguyên FPGA. Vì vậy việc thực hiện mô hình mobile net trên FPGA (DE0-Nano) là hoàn toàn khả thi. Tuy nhiên vì độ phức tạp của hệ thống nên đổi hỏi khã năng thiết kế tương đối vững.

A screenshot of a computer

AI-generated content may be incorrect.

Hình 49: Kết quả synthesis và tài nguyên sử dụng của module top

Kết quả synthesis module top (module có level cao nhất và chứa toàn bộ các module con của hệ thống), Việc synthesis thành công chứng mình rằng hệ thống hoàn toàn khả thi trong triển khai phần cứng. Các loại tài nguyên sử dụng đều trong mức cung cấp của kit.

A diagram of a computer scheme

AI-generated content may be incorrect.

Hình 50: Netlist viewer của module top được tạo ra bởi quartus

Các module con đều được tạo ra đầy đủ, không có khối module con nào bị bỏ qua hay không thể synthesis. Input và output port đầy đủ và được láy bởi đúng module chức năng.

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## Kết luận

Tuy đồ án chưa thể hoàn thành hoàn chỉnh toàn bộ mô hình mobile net trên FPGA, tuy nhiên với việc hoàn thành tấc cả những khối chức năng, khối điều khiển và có mô phỏng kiểm tra hoạt động của các module. Đã chứng minh được tính khả thi của việc thiết kế mô hình mobile net trên FPGA.

## Hướng phát triển

Đồ án này còn rất nhiều thiếu sót và còn rất nhiều hướng để phát triển hơn nữa nếu có cơ hội. Như tiếp tục hoàn thành hoàn thiện hệ thống.

Tối ứu hóa hệ thống về tốc độ: Hiện tại có một số khối điều khiển đang được viết các state chưa được tối ưu, có thể điều chỉnh lại để giảm số lượng state, giúp giảm số chu kỳ cung clock cần thiết để hoàn thành một tiến trình.

Giao tiếp với SDRAM của đồ án đang chỉ sử dụng mode single word. Mode này đọc và ghi dữ liệu trên SDRAM rất chậm, phải chịu latency không pipeline. (cần nhiều chy kỳ xung clock để hoàn thành lệnh đọc hoặc ghi mà không thời gian đó không thể thực hiện lệnh khác). Trong standard parallel SDRAM interface có mode đọc, ghi brust cho phép thực hiện lệnh đọc và ghi liên tiếp các word cạnh nhau một cách liên tục giúp giảm chu kỳ xung clock phải chờ SDRAM hoàn thành lệnh đọc hoặc ghi.

# TÀI LIỆU THAM KHẢO

1. Machine Learning Mastery, Using Depthwise Separable Convolutions in Tensorflow, https://machinelearningmastery.com/using-depthwise-separable-convolutions-in-tensorflow/
2. Ti ,DE0-NanoUserNanual(Terasis/Altera), [https://www.ti.com/lit/ug/tidu7 37/tidu737.pdf](https://www.ti.com/lit/ug/tidu7%2037/tidu737.pdf)
3. ISSI Corp, IS42R86400F/16320F, IS45R86400F/16320F IS42S86400F/16320F, IS45S86400F/16320F, <https://www.issi.com/WW/pdf/42-45R-S_86400F-16320F.pdf>
4. Medium, Pytorch Conv2d Weights Explained, <https://medium.com/data-science/pytorch-conv2d-weights-explained-ff7f68f652eb>
5. TopDev, TensorFlow là gì? Tìm hiểu về TensoFlow từ A đến Z, <https://topdev.vn/blog/tensorflow-la-gi/>

# PHỤ LỤC

Mã nguồn RTL design verilog.

Master control:

|  |
| --- |
| module masterControl(  input i\_clk,  input i\_reset,  input i\_finish\_cam,  input i\_finish\_fet,  input i\_finish\_conv,  input i\_finish\_writeBack,  input i\_finish\_ave,  output reg o\_startCam,  output reg o\_startFet,  output reg o\_startConvolution,  output reg o\_startAve,  output reg o\_startWriteBack,  output reg o\_wrActiveCam,  output reg o\_opConv,  output [5:0] o\_opcode  );  localparam [4:0] idle = 5'd0,  startCam = 5'd1,  waitDoneCam = 5'd2,  fet1 = 5'd3,  waitDoneFet1 = 5'd4,  convolution0 = 5'd5,  waitDoneConv0 = 5'd6,  writeBack1 = 5'd7,  waitDoneWriteBack1 = 5'd8,  update1 = 5'd9,  fet2 = 5'd10,  waitDoneFet2 = 5'd11,  convolution1 = 5'd12,  waitDoneConv1 = 5'd13,  writeBack2 = 5'd14,  waitDoneWriteBack2 = 5'd15,  update2 = 5'd16,  fet3 = 5'd17,  waitDoneFet3 = 5'd18,  average = 5'd19,  waitDoneAve = 5'd20,  update3 = 5'd21;  reg [4:0] state\_q, state\_d;  reg [5:0] opcode\_q, opcode\_d;  always @(posedge i\_clk or negedge i\_reset) begin  if(!i\_reset) begin  state\_q <= idle;  opcode\_q <= 6'd0;  end else begin  state\_q <= state\_d;  opcode\_q <= opcode\_d;  end  end  always @(\*) begin  case (state\_q)  idle: begin  state\_d = startCam;  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 0;  opcode\_d = 6'd0;  end  startCam: begin  state\_d = waitDoneCam;  o\_startCam = 1;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 0;  opcode\_d = opcode\_q;  end  waitDoneCam: begin  if(i\_finish\_cam) begin  state\_d = fet1;  end else begin  state\_d = waitDoneCam;  end  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 0;  opcode\_d = opcode\_q;  end  fet1: begin  state\_d = waitDoneFet1;  o\_startCam = 0;  o\_startFet = 1;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 0;  opcode\_d = opcode\_q;  end  waitDoneFet1: begin  if(i\_finish\_fet) begin  state\_d = convolution0;  end else begin  state\_d = waitDoneFet1;  end  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 0;  opcode\_d = opcode\_q;  end  convolution0: begin  state\_d = waitDoneConv0;  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 1;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 0;  opcode\_d = opcode\_q;  end  waitDoneConv0: begin  if(i\_finish\_conv) begin  state\_d = writeBack1;  end else begin  state\_d = waitDoneConv0;  end  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 0;  opcode\_d = opcode\_q;  end  writeBack1: begin  state\_d = waitDoneWriteBack1;  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 1;  o\_wrActiveCam = 0;  o\_opConv = 0;  opcode\_d = opcode\_q;  end  waitDoneWriteBack1: begin  if(i\_finish\_writeBack) begin  state\_d = update1;  end else begin  state\_d = waitDoneWriteBack1;  end  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 0;  opcode\_d = opcode\_q;  end  update1: begin  if(opcode\_q == 6'd15) begin  state\_d = fet2;  end else begin  state\_d = convolution0;  end  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 0;  opcode\_d = opcode\_q + 6'd1;  end  fet2: begin  state\_d = waitDoneFet2;  o\_startCam = 0;  o\_startFet = 1;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 1;  opcode\_d = opcode\_q;  end  waitDoneFet2: begin  if(i\_finish\_fet) begin  state\_d = convolution1;  end else begin  state\_d = waitDoneFet2;  end  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 1;  opcode\_d = opcode\_q;  end  convolution1: begin  state\_d = waitDoneConv1;  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 1;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 1;  opcode\_d = opcode\_q;  end  waitDoneConv1: begin  if(i\_finish\_conv) begin  state\_d = writeBack2;  end else begin  state\_d = waitDoneConv1;  end  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 1;  opcode\_d = opcode\_q;  end  writeBack2: begin  state\_d = waitDoneWriteBack2;  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 1;  o\_wrActiveCam = 0;  o\_opConv = 1;  opcode\_d = opcode\_q;  end  waitDoneWriteBack2: begin  if(i\_finish\_writeBack) begin  state\_d = update2;  end else begin  state\_d = waitDoneWriteBack2;  end  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 1;  opcode\_d = opcode\_q;  end  update2: begin  if(opcode\_q == 6'd31) begin  state\_d = fet3;  end else begin  state\_d = fet2;  end  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 1;  opcode\_d = opcode\_q + 6'd1;  end  fet3: begin  state\_d = waitDoneFet3;  o\_startCam = 0;  o\_startFet = 1;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 0;  opcode\_d = opcode\_q;  end  waitDoneFet3: begin  if(i\_finish\_fet) begin  state\_d = average;  end else begin  state\_d = waitDoneFet3;  end  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 0;  opcode\_d = opcode\_q;  end  average: begin  state\_d = waitDoneAve;  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 1;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 0;  opcode\_d = opcode\_q;  end  waitDoneAve: begin  if(i\_finish\_ave) begin  state\_d = update3;  end else begin  state\_d = waitDoneAve;  end  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 0;  opcode\_d = opcode\_q;  end  update3: begin  if(opcode\_q == 6'd37) begin  state\_d = idle;  end else begin  state\_d = fet3;  end  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 0;  opcode\_d = opcode\_q + 6'd1;  end  default: begin  state\_d = idle;  o\_startCam = 0;  o\_startFet = 0;  o\_startConvolution = 0;  o\_startAve = 0;  o\_startWriteBack = 0;  o\_wrActiveCam = 0;  o\_opConv = 0;  opcode\_d = 6'd0;  end  endcase  end  assign o\_opcode = opcode\_q;  endmodule |

Convolution Block:

|  |
| --- |
| module conv(  input [89:0] i\_busData0,  input [89:0] i\_busData1,  input [89:0] i\_busData2,  input [89:0] i\_busWeight0,  input [89:0] i\_busWeight1,  input [89:0] i\_busWeight2,  input i\_opcode,  output [9:0] o\_data0,  output [9:0] o\_data1,  output [9:0] o\_data2  );  //internal wire connect result of coreConv block  wire [9:0] result [2:0];  //architechture  coreConv coreBlock0(.i\_data(i\_busData0), .i\_weight(i\_busWeight0), .o\_data(result[0]));  coreConv coreBlock1(.i\_data(i\_busData1), .i\_weight(i\_busWeight1), .o\_data(result[1]));  coreConv coreBlock2(.i\_data(i\_busData2), .i\_weight(i\_busWeight2), .o\_data(result[2]));  //plus to handel CONV operation  wire [9:0] resultConv;  plus3para plus3paraBlock(.i\_data0(result[0]), .i\_data1(result[1]), .i\_data2(result[2]), .o\_data(resultConv));  //assign output  mux2to1 mux2to1Block(.i\_dataA(resultConv), .i\_dataB(result[0]), .i\_sel(i\_opcode), .o\_data(o\_data0));  assign o\_data1 = result[1];  assign o\_data2 = result[2];  endmodule  module plus3para(  input [9:0] i\_data0,  input [9:0] i\_data1,  input [9:0] i\_data2,  output [9:0] o\_data  );  //for high performance  //state 0  wire [9:0] data01;  assign data01 = i\_data1 + i\_data0;  //final state  assign o\_data = data01 + i\_data2;  endmodule |

StoreCamControl:

|  |
| --- |
| module storeCamControl(      input               i\_clk,      input               i\_reset,      input               i\_start,      input [9:0]         i\_remainOnFifo,      input               i\_process,      input               i\_sdramReady,      input               i\_complete, //complete signal of controlWriteFifo      input [15:0]        i\_dataFifo,      output reg          o\_get,      output reg          o\_EnReadFifo,      output reg          o\_RdClkFifo,      output     [15:0]   o\_dataSdram,      output reg [18:0]   o\_addressToSdram,      output reg          o\_wrSdram,      output reg          o\_finish  );  localparam [3:0]    idle = 4'd0,                      start = 4'd1,                      suspend = 4'd2,                      readFifo0 = 4'd3,                      readFifo1 = 4'd4,                      setRed = 4'd5,                      wait0 = 4'd6,                      setGreen = 4'd7,                      wait1 = 4'd8,                      setBlue = 4'd9,                      wait2 = 4'd10,                      update = 4'd11,                      finish = 4'd12;  reg [3:0] state\_q, state\_d;  reg [18:0] addrLocal\_q, addrLocal\_d;  reg [7:0] dataToSdram8;  always @(posedge i\_clk or negedge i\_reset) begin    //sequential circuit update state, addr local      if(!i\_reset) begin          state\_q <= idle;          addrLocal\_q <= 19'd0;      end else begin          state\_q <= state\_d;          addrLocal\_q <= addrLocal\_d;      end  end  always @(\*) begin      case (state\_q)          idle: begin              if(i\_start & i\_reset) begin                  state\_d = start;              end else begin                  state\_d = idle;              end              o\_get = 0;              o\_EnReadFifo = 0;              o\_RdClkFifo = 0;              dataToSdram8 = 8'd0;              o\_addressToSdram = 19'd0;              o\_wrSdram = 0;              o\_finish = 0;              addrLocal\_d = 19'd0;          end          start: begin              if(i\_process) begin                  state\_d = suspend;              end else begin                  state\_d = start;              end              o\_get = 1;              o\_EnReadFifo = 0;              o\_RdClkFifo = 0;              dataToSdram8 = 8'd0;              o\_addressToSdram = 19'd0;              o\_wrSdram = 0;              o\_finish = 0;              addrLocal\_d = 19'd0;          end          suspend: begin              if((i\_remainOnFifo > 10'd16) | (i\_complete)) begin                  state\_d = readFifo0;              end else begin                  state\_d = suspend;              end              o\_get = 0;              o\_EnReadFifo = 1;              o\_RdClkFifo = 0;              dataToSdram8 = 8'd0;              o\_addressToSdram = 19'd0;              o\_wrSdram = 0;              o\_finish = 0;              addrLocal\_d = addrLocal\_q;          end          readFifo0: begin              state\_d = readFifo1;              o\_get = 0;              o\_EnReadFifo = 1;              o\_RdClkFifo = 1;              dataToSdram8 = 8'd0;              o\_addressToSdram = 19'd0;              o\_wrSdram = 0;              o\_finish = 0;              addrLocal\_d = addrLocal\_q;          end          readFifo1: begin              state\_d = setRed;              o\_get = 0;              o\_EnReadFifo = 1;              o\_RdClkFifo = 0;              dataToSdram8 = 8'd0;              o\_addressToSdram = 19'd0;              o\_wrSdram = 0;              o\_finish = 0;              addrLocal\_d = addrLocal\_q;          end          setRed: begin              state\_d = wait0;              o\_get = 0;              o\_EnReadFifo = 0;              o\_RdClkFifo = 0;              dataToSdram8 = {2'd0,i\_dataFifo[15:11], 1'd0};              o\_addressToSdram = addrLocal\_q;              o\_wrSdram = 1;              o\_finish = 0;              addrLocal\_d = addrLocal\_q;          end          wait0: begin              if(i\_sdramReady) begin                  state\_d = setGreen;              end else begin                  state\_d = wait0;              end              o\_get = 0;              o\_EnReadFifo = 0;              o\_RdClkFifo = 0;              dataToSdram8 = {2'd0,i\_dataFifo[15:11], 1'd0};              o\_addressToSdram = addrLocal\_q;              o\_wrSdram = 0;              o\_finish = 0;              addrLocal\_d = addrLocal\_q;          end          setGreen: begin              state\_d = wait1;              o\_get = 0;              o\_EnReadFifo = 0;              o\_RdClkFifo = 0;              dataToSdram8 = {2'd0,i\_dataFifo[10:5]};              o\_addressToSdram = addrLocal\_q + 19'd4096;    //sdram region 2 store feature Green              o\_wrSdram = 1;              o\_finish = 0;              addrLocal\_d = addrLocal\_q;          end          wait1: begin              if(i\_sdramReady) begin                  state\_d = setBlue;              end else begin                  state\_d = wait1;              end              o\_get = 0;              o\_EnReadFifo = 0;              o\_RdClkFifo = 0;              dataToSdram8 = {2'd0,i\_dataFifo[10:5]};              o\_addressToSdram = addrLocal\_q + 19'd4096;              o\_wrSdram = 0;              o\_finish = 0;              addrLocal\_d = addrLocal\_q;          end          setBlue: begin              state\_d = wait2;              o\_get = 0;              o\_EnReadFifo = 0;              o\_RdClkFifo = 0;              dataToSdram8 = {2'd0, i\_dataFifo[4:0], 1'd0};              o\_addressToSdram = addrLocal\_q + 19'd8192;    //sdram region 2 store feature Green              o\_wrSdram = 1;              o\_finish = 0;              addrLocal\_d = addrLocal\_q;          end          wait2: begin              if(i\_sdramReady) begin                  state\_d = update;              end else begin                  state\_d = wait2;              end              o\_get = 0;              o\_EnReadFifo = 0;              o\_RdClkFifo = 0;              dataToSdram8 = {2'd0, i\_dataFifo[4:0], 1'd0};              o\_addressToSdram = addrLocal\_q + 19'd8192;              o\_wrSdram = 0;              o\_finish = 0;              addrLocal\_d = addrLocal\_q;          end          update: begin              if((~i\_complete) | (i\_remainOnFifo > 10'd0)) begin                  state\_d = suspend;              end else begin                  state\_d = finish;              end              o\_get = 0;              o\_EnReadFifo = 0;              o\_RdClkFifo = 0;              dataToSdram8 = 8'd0;              o\_addressToSdram = 19'd0;              o\_wrSdram = 0;              o\_finish = 0;              addrLocal\_d = addrLocal\_q + 19'd1; //update local address          end          finish: begin              state\_d = idle;              o\_get = 0;              o\_EnReadFifo = 0;              o\_RdClkFifo = 0;              dataToSdram8 = 8'd0;              o\_addressToSdram = 19'd0;              o\_wrSdram = 0;              o\_finish = 1;              addrLocal\_d = 19'd0;          end          default: begin              state\_d = idle;              o\_get = 0;              o\_EnReadFifo = 0;              o\_RdClkFifo = 0;              dataToSdram8 = 8'd0;              o\_addressToSdram = 19'd0;              o\_wrSdram = 0;              o\_finish = 0;              addrLocal\_d = 19'd0;          end      endcase  end  assign o\_dataSdram = {8'd0, dataToSdram8}; //assign to 16 bit for SDRAM (Extend unsigned)  endmodule |

Mã nguồn python check mô hình:

|  |
| --- |
| import numpy as np  import cv2  def conv(data, weight):      data = np.array(data)      weight = np.array(weight)      result = 0      for i in range(9\*3):          result = result + data[i]\*weight[i]      return result  dataOutputToFile = []  def convImg(data):      data = np.array(data)      for i in range(64):          for j in range(64):              dataSet = []              dataSet.append(data[i-1][j-1][0] if (i > 0 and j > 0) else 0)              dataSet.append(data[i-1][j][0] if (i > 0) else 0)              dataSet.append(data[i-1][j+1][0] if (j < 63) else 0)              dataSet.append(data[i][j-1][0] if (j > 0) else 0)              dataSet.append(data[i][j][0])              dataSet.append(data[i][j+1][0] if (j < 63) else 0)              dataSet.append(data[i+1][j-1][0] if (i < 63 and j > 0) else 0)              dataSet.append(data[i+1][j][0] if (i < 63) else 0)              dataSet.append(data[i+1][j+1][0] if (i < 63 and j < 63) else 0)              dataSet.append(data[i-1][j-1][1] if (i > 0 and j > 0) else 0)              dataSet.append(data[i-1][j][1] if (i > 0) else 0)              dataSet.append(data[i-1][j+1][1] if (j < 63) else 0)              dataSet.append(data[i][j-1][1] if (j > 0) else 0)              dataSet.append(data[i][j][1])              dataSet.append(data[i][j+1][1] if (j < 63) else 0)              dataSet.append(data[i+1][j-1][1] if (i < 63 and j > 0) else 0)              dataSet.append(data[i+1][j][1] if (i < 63) else 0)              dataSet.append(data[i+1][j+1][1] if (i < 63 and j < 63) else 0)              dataSet.append(data[i-1][j-1][2] if (i > 0 and j > 0) else 0)              dataSet.append(data[i-1][j][2] if (i > 0) else 0)              dataSet.append(data[i-1][j+1][2] if (j < 63) else 0)              dataSet.append(data[i][j-1][2] if (j > 0) else 0)              dataSet.append(data[i][j][2])              dataSet.append(data[i][j+1][2] if (j < 63) else 0)              dataSet.append(data[i+1][j-1][2] if (i < 63 and j > 0) else 0)              dataSet.append(data[i+1][j][2] if (i < 63) else 0)              dataSet.append(data[i+1][j+1][2] if (i < 63 and j < 63) else 0)              dataSet = np.array(dataSet)              dataSet = dataSet / 255              weigh = []              weigh.append(0.65405077)              weigh.append(0.69987404)              weigh.append(0.4878358)              weigh.append(0.6011477)              weigh.append(0.60555035)              weigh.append(0.55344516)              weigh.append(0.74110985)              weigh.append(0.3748532)              weigh.append(0.37070867)              weigh.append(0.57734627)              weigh.append(0.36462057)              weigh.append(0.48534322)              weigh.append(0.4103786)              weigh.append(0.20865759)              weigh.append(0.5101151)              weigh.append(0.70207393)              weigh.append(0.261061)              weigh.append(0.4263476)              weigh.append(0.4891929)              weigh.append(0.26326942)              weigh.append(0.5449106)              weigh.append(0.4609151)              weigh.append(0.21110745)              weigh.append(0.46697062)              weigh.append(0.6671197)              weigh.append(0.3644622)              weigh.append(0.31627488)              temp = conv(dataSet, weigh)              print('pixel tai i: ' + str(i) + ', j= ' + str(j) + ', la = ' + str(temp\*64))  def int2Hex(value, bits):      if value < 0:          value = (1 << bits) + value  # Compute two's complement      binary = format(value, f'0{bits}b')      hex\_digits = format(int(binary, 2), '03X')  # 'FD'      return hex\_digits  img = cv2.imread('test50.jpg')  # Resize to 64x64  resized\_img = cv2.resize(img, (64, 64), interpolation=cv2.INTER\_AREA)  cv2.imshow("After resize", resized\_img)  cv2.waitKey(0)  cv2.destroyAllWindows()  # Optional: convert to list  pixel\_matrix = resized\_img.tolist()  print("Width: "+ str(len(pixel\_matrix)) + "Hight: " + str(len(pixel\_matrix[0])))  print(pixel\_matrix)  convImg(pixel\_matrix)  with open("picture.hex", "w") as f:      for row in pixel\_matrix:          for pixel in row:              r, g, b = pixel              r = int2Hex(int(r/255\*64), 10)              f.write(r + '\n')              g = int2Hex(int(g/255\*64), 10)              f.write(g + '\n')              b = int2Hex(int(b/255\*64), 10)              f.write(b + '\n')  print('conplete write mem') |

Phần mã nguồn còn lại được lưu trữ tại: https://github.com/nttoan-khiem/MobileNetOnFPGA

Sơ đồ hệ thống đầy đủ:

A diagram of a computer

AI-generated content may be incorrect.

1. VBLO, [Deep Learning] Tìm hiểu về mạng tích chập (CNN), (10/12/2020), truy cập tại https://viblo.asia/p/deep-learning-tim-hieu-ve-mang-tich-chap-cnn-maGK73bOKj2 [↑](#footnote-ref-1)
2. Cloufy, Thuật Toán CNN Là Gì? Tìm Hiểu Tất Tần Tật Về CNN, (18/9/2024), truy cập tại https://cloudfly.vn/docs/tin-cong-nghe/thuat-toan-cnn-la-gi-tim-hieu-tat-tan-tat-ve-cnn [↑](#footnote-ref-2)
3. Mendeley Data, Traffic congestion Dataset, (03/9/2020), truy cập tại https://data.mendeley.com/datasets/wtp4ssmwsd/1 [↑](#footnote-ref-3)
4. [↑](#footnote-ref-4)
5. Google play, Camera Giao Thông Việt Nam, (12/03/2024), truy cập tại https://play.google.com/store/apps/details?id=com.freeapp.camtraffic.traffic\_cam&hl=vi&pli=1 [↑](#footnote-ref-5)
6. Top-Dev, TensorFlow là gì? Tìm hiểu về TensoFlow từ A đến Z, (23/04/2025), truy cập tại https://topdev.vn/blog/tensorflow-la-gi/ [↑](#footnote-ref-6)
7. Medium, Pytorch Conv2d Weights Explained, (26/11/2021), truy cập tại https://medium.com/data-science/pytorch-conv2d-weights-explained-ff7f68f652eb [↑](#footnote-ref-7)
8. Machine Learning Mastery, Using Depthwise Separable Convolutions in Tensorflow, (10/08/2022), truy cập tại https://machinelearningmastery.com/using-depthwise-separable-convolutions-in-tensorflow/ [↑](#footnote-ref-8)