

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

ĐỒ ÁN TỐT NGHIỆP Nghiên cứu, thiết kế và triển khai thuật toán SIFT trên FPGA

NGUYỄN VĂN HÙNG

hung.nv161972@sis.hust.edu.vn

NGUYỄN HỮU TRÁNG

trang.nh164196@sis.hust.edu.vn

Ngành Công nghệ Thông tin

Giảng viên hướng dẫn: TS. Đỗ Công Thuần

Chữ ký của GVHD1

ThS. Nguyễn Đức Tiến

Chữ ký của GVHD2

Bộ môn:

Kỹ thuật máy tính

Viện:

Công nghệ Thông tin và Truyền thông

HÀ NỘI, 12/2020

PHIẾU GIAO NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

1. Thông tin về sinh viên

Họ và tên sinh viên: Nguyễn Văn Hùng

Điện thoại liên lạc: 0326.540.044

Lớp: CNTT1.01 K61

Họ và tên sinh viên: Nguyễn Hữu Tráng

Điện thoại liên lạc: 0352.408.617

Lớp: CNTT1.02 K61

Email: hung.nv161972@sis.hust.edu.vn

Hệ đào tạo: Đại học chính quy

Email: trang.nh164196@sis.hust.edu.vn

Hệ đào tạo: Đại học chính quy

Đồ án tốt nghiệp được thực hiện tại: Đại học Bách khoa Hà Nội

Thời gian làm ĐATN: Từ ngày 20/09/2020 đến 30/12/2020

2. Mục đích nội dung của ĐATN

Mục đích chính của đồ án là xây dựng và triển khai thuật toán trích chọn đặc trưng ảnh SIFT trên nền tảng FPGA.

3. Các nhiệm vụ cụ thể của ĐATN

- Sử dụng giao thức PCIe sử dụng để truyền ảnh/video giữa FPGA và PC.
- Tìm hiểu, sử dụng các công cụ DSP Builder, Simulink, MATLAB để thiết kế và mô phỏng kết quả IP Core thực hiện thuật toán SIFT.
- Xây dựng hệ thống triển khai IP Core thuật toán SIFT trên kit DE2i-150.
- Triển khai hệ thống trên FPGA và đánh giá kết quả thu được.

4. Lời cam đoan của sinh viên:

Tôi – Nguyễn Văn Hùng và tôi – Nguyễn Hữu Tráng – cam kết ĐATN là công trình nghiên cứu của bản thân chúng tôi dưới sự hướng dẫn của TS Đỗ Công Thuần và ThS Nguyễn Đức Tiến. Các kết quả nêu trong ĐATN là trung thực, không phải là sao chép toàn văn của bất kỳ công trình nào khác.

Hà Nội, ngày 30 tháng 12 năm 2020
Tác giả ĐATN

Nguyễn Văn Hùng

Hà Nội, ngày 30 tháng 12 năm 2020
Tác giả ĐATN

Nguyễn Hữu Tráng

5. Xác nhận của giáo viên hướng dẫn về mức độ hoàn thành của ĐATN và cho phép bảo vệ:

Hà Nội, ngày 30 tháng 12 năm 2020
Giảng viên hướng dẫn

TS Đỗ Công Thuần

Hà Nội, ngày 30 tháng 12 năm 2020
Giảng viên hướng dẫn

ThS Nguyễn Đức Tiến

PHÂN CHIA CÔNG VIỆC

Đồ án tốt nghiệp này là sản phẩm tìm tòi, nghiên cứu, phát triển và thử nghiệm của nhóm hai người chúng em: sinh viên Nguyễn Văn Hùng – 20161972 và sinh viên Nguyễn Hữu Tráng – 20164196. Trong quá trình thực hiện đồ án, chúng em có sự phân chia công việc để mỗi người có thể tập trung vào một phần chính trong đồ án nhưng cả hai vẫn hiểu những gì người còn lại đang làm để có thể phối hợp với nhau tốt hơn.

Về mặt cơ sở kiến thức của đồ án như thuật toán SIFT, chuẩn giao tiếp Avalon, cả hai đều thống nhất dùng thời gian đầu để cùng tìm hiểu và nắm vững, trao đổi với nhau, qua đó tránh trường hợp hiểu sai hay hai người hiểu khác nhau. Sau khi cả hai đã nắm vững cơ sở kiến thức, Nguyễn Văn Hùng đảm nhiệm việc nghiên cứu phát triển và mô phỏng thuật toán từng bước trên DSP Builder, Nguyễn Hữu Tráng đảm nhiệm tìm hiểu, xây dựng framework dùng để triển khai thiết kế của Nguyễn Văn Hùng trên kit DE2i-150 để thử nghiệm hoạt động của IP Core. Một tuần sẽ có ít nhất 2 buổi dành cho trao đổi nhóm, Nguyễn Văn Hùng sẽ giới thiệu công việc của mình, nêu ý tưởng thiết kế cho Nguyễn Hữu Tráng hiểu, đồng thời Nguyễn Hữu Tráng cũng trao đổi với Nguyễn Văn Hùng về những thay đổi trong giao diện kết nối IP Core do Nguyễn Văn Hùng thiết kế vào framework cũng như kết quả thử nghiệm các IP Core Nguyễn Văn Hùng đã mô phỏng thành công. Khi xảy ra lỗi, Nguyễn Hữu Tráng sẽ thông báo lỗi cho Nguyễn Văn Hùng và cả 2 phối hợp để tìm và sửa lỗi.

Nguyễn Văn Hùng	Nguyễn Hữu Tráng
Tìm hiểu kit FPGA DE2i-150 Chuẩn giao tiếp Avalon PCI Express Core NiOS Thuật toán SIFT	
Phát triển Mô phỏng và Kiểm tra thuật toán trên DSP Builder	Phát triển soft trên nửa mạch Atom Xây dựng giao tiếp Atom-FPGA Triển khai thuật toán trên DE2i-150
Thử nghiệm và So sánh kết quả Phát hiện sửa lỗi	

LỜI CẢM ƠN

Nhóm sinh viên thực hiện xin chân thành gửi lời cảm ơn tới:

Toàn thể các thầy cô trường Đại học Bách khoa Hà Nội đã truyền thụ cho chúng em những kiến thức, kinh nghiệm quý báu trong suốt thời gian chúng em học tập và rèn luyện tại trường.

Thầy Nguyễn Đức Tiến và thầy Đỗ Công Thuần đã nhiệt tình hướng dẫn chúng em hoàn thành đồ án tốt nghiệp.

Anh Nguyễn Hữu Dũng – kỹ sư tại Viện Hàng không Vũ trụ Viettel đã nhiệt tình giúp đỡ chúng em ngay từ những ngày đầu thực hiện đồ án tốt nghiệp.

Ngoài ra, chúng em xin được gửi lời cảm ơn đến gia đình, bạn bè đã luôn ở bên động viên, tạo điều kiện cho chúng em có thể hoàn thành đồ án tốt nghiệp này.

Do thời gian và kiến thức còn hạn chế nên đồ án này khó tránh khỏi có nhiều thiếu sót, chúng em rất mong nhận được sự đóng góp ý kiến của các thầy cô và các bạn.

TÓM TẮT NỘI DUNG ĐO ÁN

Mục tiêu chính của đồ án là thiết kế, triển khai IP Core thực hiện thuật toán trích chọn đặc trưng ảnh SIFT trên nền tảng FPGA. Nội dung chính của đồ án gồm:

CHƯƠNG 1: MỤC TIÊU ĐỀ TÀI VÀ CÔNG NGHỆ

Giới thiệu đề tài, công nghệ FPGA, Kit phát triển DE2i-150 và chuẩn giao tiếp Avalon được sử dụng để thiết kế và triển khai thuật toán cũng như hệ thống.

CHƯƠNG 2: THUẬT TOÁN TRÍCH SUẤT ĐẶC TRƯNG ẢNH SIFT

Trình bày về cơ sở kiến thức của thuật toán trích chọn đặc trưng ảnh SIFT.

CHƯƠNG 3: THIẾT KẾ VÀ MÔ PHỎNG TRÊN MATLAB

Trình bày thiết kế và kết quả mô phỏng các IP Core trong thuật toán SIFT trên DSP Builder, MATLAB.

CHƯƠNG 4: TRIỂN KHAI HỆ THỐNG TRÊN KIT DE2i-150

Trình bày kiến trúc của hệ thống triển khai trên kit DE2i-150.

CHƯƠNG 5: KẾT QUẢ THỬ NGHIỆM

Trình bày quy trình, kết quả thử nghiệm trên kit DE2i-150.

CHƯƠNG 6: KẾT LUẬN

Tổng kết lại kết quả và hướng phát triển của đồ án.

Nhóm sinh viên thực hiện
Ký và ghi rõ họ tên

MỤC LỤC

PHIẾU GIAO NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP.....	ii
PHÂN CHIA CÔNG VIỆC	iii
CHƯƠNG 1. MỤC TIÊU ĐỀ TÀI VÀ CÔNG NGHỆ.....	1
1.1 Tổng quan về đề tài	1
1.1.1 Thực trạng	1
1.1.2 Đề xuất mục tiêu	1
1.1.3 Đề xuất giải pháp khả thi	1
1.2 Giới thiệu về công nghệ FPGA	2
1.3 Kit phát triển DE2i-150.....	3
1.4 Chuẩn giao tiếp Avalon.....	8
1.4.1 Tổng quan về chuẩn giao tiếp Avalon	8
1.4.2 Sử dụng giao diện Avalon Streaming để truyền ảnh/video	8
CHƯƠNG 2. THUẬT TOÁN TRÍCH CHỌN ĐẶC TRUNG ẢNH SIFT....	12
2.1 Giới thiệu thuật toán SIFT	12
2.2 Các bước thực hiện thuật toán.....	13
2.2.1 Phát hiện những điểm cực trị trong không gian tỷ lệ.....	13
2.2.2 Xác định vị trí những điểm hấp dẫn.....	17
2.2.3 Xác định hướng và độ lớn biến đổi của những điểm hấp dẫn ..	17
2.2.4 Tạo bộ mô tả cho các điểm hấp dẫn trong ảnh	18
2.3 Kỹ thuật đối sánh (Matching)	19
CHƯƠNG 3. THIẾT KẾ VÀ MÔ PHỎNG TRÊN MATLAB	20
3.1 Giới thiệu các công cụ sử dụng để thiết kế và thử nghiệm	20
3.1.1 Công cụ Altera DSP Builder	20
3.1.2 Tổng quan một số IP Core trong Altera DSP Builder	22
3.2 Tổng quan thiết kế.....	26
3.3 Thiết kế và kết quả mô phỏng IP Core rgb2gray	28
3.3.1 Thiết kế	28
3.3.2 Mô phỏng trên matlab	29
3.4 Thiết kế và mô phỏng IP Core Gaussian Filter	29
3.4.1 Thiết kế	29
3.4.2 Thực hiện mô phỏng	31

3.5	Thiết kế và mô phỏng IP Core SIFT Detector	35
3.5.1	Thiết kế khối scale_space	36
3.5.2	Thiết kế khối detect_keypoint.....	36
3.6	Thiết kế và mô phỏng IP Core calculate_descriptor	41
3.7	Thiết kế IP Core Matching.....	50
CHƯƠNG 4. TRIỂN KHAI HỆ THỐNG TRÊN KIT DE2i-150.....		53
4.1	Kiến trúc tổng quan hệ thống	53
4.2	Trao đổi thông tin giữa nửa PC và nửa FPGA của kit DE2i-150	54
4.3	Các IP Core có sẵn trong Qsys sử dụng trong hệ thống	58
4.3.1	IP Core SDRAM Controller.....	58
4.3.2	IP Core Frame Reader.....	60
4.3.3	IP Frame Buffer	61
4.3.4	IP Clocked Video Output.....	65
4.4	Triển khai hệ thống trên nửa FPGA	66
4.5	Triển khai hệ thống trên nửa PC	68
CHƯƠNG 5. KẾT QUẢ THỬ NGHIỆM.....		72
5.1	Quy trình và kịch bản thử nghiệm.....	72
5.1.1	Quy trình thử nghiệm	72
5.1.2	Kịch bản thử nghiệm.....	72
5.2	Kết quả thử nghiệm IP Core rgb2gray	72
5.3	Kết quả thử nghiệm IP Core Gaussian Filter	73
5.4	Kết quả thử nghiệm IP Core SIFT Detector	75
5.5	Những khó khăn gặp phải khi thực hiện đồ án và giải pháp.....	78
CHƯƠNG 6. KẾT LUẬN.....		80
6.1	Kết luận	80
6.2	Hướng phát triển đồ án trong tương lai.....	80
TÀI LIỆU THAM KHẢO		81
PHỤ LỤC		82

DANH MỤC HÌNH VẼ

Hình 1.1 Kiến trúc cơ bản của FPGA	2
Hình 1.2 Hình ảnh thực tế mặt trên kit DE2i-150 (kèm chú thích)	4
Hình 1.3 Hình ảnh thực tế mặt dưới kit DE2i-150 (kèm chú thích)	4
Hình 1.4 Block Diagram kit DE2i-150	5
Hình 1.5 Tín hiệu truyền gói hình ảnh bằng giao diện Avalon – ST	9
Hình 1.6 Timing Diagram Showing R'G'B' Transferred in Parallel	10
Hình 1.7 Thiết kế giao diện vào ra cho IP Core SIFT	11
Hình 2.1 Ứng dụng so khớp 2 ảnh sử dụng thuật toán SIFT	12
Hình 2.2 Sự thay đổi của đối tượng khi thay đổi kích thước	13
Hình 2.3 Ví dụ Scale space	14
Hình 2.4 Mô tả quá trình tính không gian tỉ lệ (L) và hàm sai khác (D)	15
Hình 2.5 DoG và Laplacian of Gaussian	16
Hình 2.6 Quá trình tìm điểm cực trị trong các hàm sai khác DoG	16
Hình 2.7 Xây dựng descriptor cho Keypoint	18
Hình 2.8 Lấy mẫu hướng biến đổi của điểm ảnh	19
Hình 3.1 DSP Builder overview.....	20
Hình 3.2 Block DSP Builder	20
Hình 3.3 DSP Builder – Signal Compiler	21
Hình 3.4 Sử dụng DSP Builder để hỗ trợ thiết kế phần cứng	21
Hình 3.5 Sơ đồ khối tổng quan thiết kế theo chiều dữ liệu	26
Hình 3.6 Sơ đồ tổng quan thiết kế khối Octave0 theo chiều dữ liệu	27
Hình 3.7 Thiết kế khối rgb2gray	28
Hình 3.8 Ảnh sau khi đi qua khối rgb2gray	29
Hình 3.9 Sơ đồ mức top của khối Gaussian Filter	29
Hình 3.10 Mô phỏng line buffer 3x3	30
Hình 3.11 Thiết kế khối line_buff5x5	30
Hình 3.12 Thiết kế khối convolution	31
Hình 3.13 Tạo file script sink dữ liệu đầu vào Simulink	32
Hình 3.14 Không gian Workspace trên MATLAB	32
Hình 3.15 Tiến hành chạy mô phỏng thiết kế	33
Hình 3.16 Hiển thị kết quả ảnh sau khối Gaussian_filter	33
Hình 3.17 Ảnh nhận được khi qua các bộ lọc Gaussian filter	33
Hình 3.18 Theo dõi giá trị các tín hiệu qua cửa sổ Scope.....	34
Hình 3.19 Top level of a design SIFT Detector	35
Hình 3.20 Thiết kế scale space.....	36

Hình 3.21 Thiết kế khối detect_keypoint.....	36
Hình 3.22 Thiết kế khối dog_space.....	37
Hình 3.23 Kết quả mô phỏng ảnh trong không gian dog_space	37
Hình 3.24 Thiết kế khối detect_keypoint.....	38
Hình 3.25 Khối line_buff3x3 cho ảnh 640*480	38
Hình 3.26 Thiết kế khối find_keypoint.....	38
Hình 3.27 Thiết kế khối find_max	39
Hình 3.28 Ảnh hưởng khi giảm độ chính xác không gian dog_space đến kết quả tìm kiếm keypoint	39
Hình 3.29 Kết quả mô phỏng detect keypoint trên ảnh logo BKHN.....	40
Hình 3.30 Mô phỏng detect keypoint trên ảnh.....	41
Hình 3.31 Sơ đồ thiết kế khối calculate_descriptor	41
Hình 3.32 Thiết kế khối GradOrian	42
Hình 3.33 Thiết kế khối line_buff3x3.....	42
Hình 3.34 Thiết kế khối sub_GradOrian.....	43
Hình 3.35 Kết quả mô phỏng IP Core GradOrien với detect keypoint.....	43
Hình 3.36 Thiết kế khối hist_bin.....	44
Hình 3.37 Các chân input của khối hist_bin1	45
Hình 3.38 Thiết kế các phần so sánh của khối hist_bin1	45
Hình 3.39 Tính tổng độ lớn biến đổi của các điểm ảnh có cùng hướng biến đổi	45
Hình 3.40 Thiết kế khối Bus_Concatenation	46
Hình 3.41 Mô phỏng vị trí của các histogram được sử dụng xây dựng descriptor	46
Hình 3.42 Thiết kế khối counter	47
Hình 3.43 Thiết kế khối descriptor	47
Hình 3.44 Thiết kế khối BusSplitter	47
Hình 3.45 Thiết kế khối findmax	48
Hình 3.46 Thiết kế khối findmax	48
Hình 3.47 Thiết kế khối subsystem descriptor.....	49
Hình 3.48 Thiết kế khối final_descriptor	49
Hình 3.49 Kết quả so khớp dựa trên dữ liệu thu được từ mô phỏng của SIFT Detector và calculate_descriptor	50
Hình 3.50 Sơ đồ khối cho matching song song	51
Hình 3.51 Sơ đồ khối matching nối tiếp	52
Hình 4.1 Sơ đồ kiến trúc tổng quan hệ thống	53
Hình 4.2 PCI Express System Framework.....	55
Hình 4.3 PCIe framework trên Altera Qsys	55

Hình 4.4 Ví dụ về cấu hình Memory-to-Memory của SG – DMA	56
Hình 4.5 PCI Express Avalon-MM Bridge.....	57
Hình 4.6 Block diagram IP Core SDRAM Controller	58
Hình 4.7 Sơ đồ kết nối và sử dụng SDRAM Controller	59
Hình 4.8 Block diagram IP Core Frame Reader	60
Hình 4.9 Sơ đồ đọc dữ 3 kênh song song, mỗi kênh 8-bit bằng Frame Reader ..	60
Hình 4.10 Block diagram IP Core Frame Buffer	61
Hình 4.11 Sơ đồ chiêu chuyển dữ liệu của IP Frame Buffer	62
Hình 4.12 Block diagram IP Core Clocked Video Output	65
Hình 4.13 Sinh IP Core sử dụng trong Qsys từ thiết kế trên DSP Builder	66
Hình 4.14 User Logic IP Core trong Qsys	67
Hình 4.15 Kết nối IO của hệ thống sinh ra từ Qsys với ngoại vi bằng Verilog...	68
Hình 4.16 Ví dụ về một giản đồ sóng hiển thị bởi SignalTap II Logic Analyzer	68
Hình 4.17 PCI Express Software Stack on Yocto.....	69
Hình 5.1 Kết quả thử nghiệm IP Core rgb2gray	73
Hình 5.2 Một đoạn giản đồ sóng I/O của IP Core rgb2gray	73
Hình 5.3 Kết quả thử nghiệm IP Core Gaussian Filter	74
Hình 5.4 Một đoạn giản đồ sóng I/O của IP Core Gaussian Filter	75
Hình 5.5 Kết quả IP Core SIFT Detector với logo BKHN	76
Hình 5.6 Kết quả IP Core SIFT Detector với logo Coca Cola.....	77

DANH MỤC BẢNG BIỂU

Bảng 1.1 Thông tin chi tiết phần cứng nửa FPGA của kit DE2i-150	6
Bảng 1.2 Thông tin chi tiết phần cứng nửa PC của kit DE2i-150	7
Bảng 1.3 Các giao diện trong chuẩn giao tiếp Avalon.....	8
Bảng 1.4 Danh sách các tín hiệu của giao diện Avalon – ST sử dụng trong đồ án	9
Bảng 3.1 Một vài IP Core trong Altera DSP Builder sử dụng trong đồ án	22
Bảng 4.1 Các tham số cấu hình IP SDRAM Controller.....	59
Bảng 4.2 Tham số cấu hình cho Frame Reader	61
Bảng 4.3 Chi tiết tham số định dạng dữ liệu khung hình của Frame Buffer	63
Bảng 4.4 Chi tiết tham số bộ xử lý dữ liệu phi hình ảnh của Frame Buffer	63
Bảng 4.5 Chi tiết tham số behavior của Frame Buffer.....	64
Bảng 4.6 Chi tiết tham số cho Avalon – MM Master của Frame Buffer.....	64
Bảng 4.7 Chi tiết các tham số định dạng dữ liệu khung hình CVO.....	65
Bảng 4.8 Chi tiết hàm PCIE_Open	69
Bảng 4.9 Chi tiết hàm PCIE_Close.....	70
Bảng 4.10 Chi tiết hàm PCIE_Read32.....	70
Bảng 4.11 Chi tiết hàm PCIE_Write32.....	70
Bảng 4.12 Chi tiết hàm PCIE_DmaRead.....	71
Bảng 4.13 Chi tiết hàm PCIE_DmaWrite	71
Bảng 5.1 Kết quả triển khai thử nghiệm các IP Core trên kit DE2i-150	78
Bảng 5.2 Thông kê tài nguyên sử dụng và thời gian triển khai trên FPGA.....	78

DANH MỤC TỪ VIẾT TẮT VÀ THUẬT NGỮ

STT	Thuật ngữ	Ý nghĩa
1	FPGA	Field-programmable gate array, là một mạch tích hợp được thiết kế để người dùng hoặc các nhà thiết kế có khả năng cấu hình sau khi sản xuất (lập trình được)
2	IP Core	Intellectual property core, là một mạch xử lý được xây dựng nhằm thực hiện một thao tác xử lý hay một thuật toán nào đó trên FPGA/ASIS
3	Component	Thuật ngữ trong ngôn ngữ VHDL, mang ý nghĩa tương tự IP Core
4	HDL	Hardware Description Language, ngôn ngữ mô tả phần cứng, rất hữu ích và được sử dụng rất rộng rãi trong thiết kế IC. Một số ngôn ngữ HDL phổ biến như VHDL, Verilog, System C, ...
5	API	Application Programming Interface, một giao diện lập trình ứng dụng, cung cấp khả năng truy suất đến một tập các hàm hay dùng, tạo thuận tiện cho lập trình ứng dụng
6	SIFT	Scale Invariant Feature Transform, là một thuật toán trích chọn đặc trưng ảnh cục bộ bất biến với phép dịch, phép xoay và phép scale (zoom)
7	LoG	Laplacian of Gaussian, dùng tính xấp xỉ đạo hàm bậc hai của ảnh, ứng dụng trong lọc ảnh hay các thuật toán phức tạp như Blob, SIFT, ...
8	DoG	Difference of Gaussian, được dùng để tính xấp xỉ LoG, giảm lượng tính toán. DoG được tính bằng cách trừ hai ảnh kết quả thu được khi nhân chập một ảnh với hai mặt nạ gaussian khác nhau.
9	Scale	Thuật ngữ dùng trong thuật toán SIFT, một scale là kết quả của phép nhân chập ảnh cần xử lý (kích thước có thể khác nhau) với mặt nạ gaussian (kích thước và sigma khác nhau)
10	Octave	Thuật ngữ dùng trong thuật toán SIFT, một octave ứng với một kích thước ảnh trong không gian tỉ lệ, mỗi octave sẽ gồm nhiều scale

CHƯƠNG 1. MỤC TIÊU ĐỀ TÀI VÀ CÔNG NGHỆ

1.1 Tổng quan về đề tài

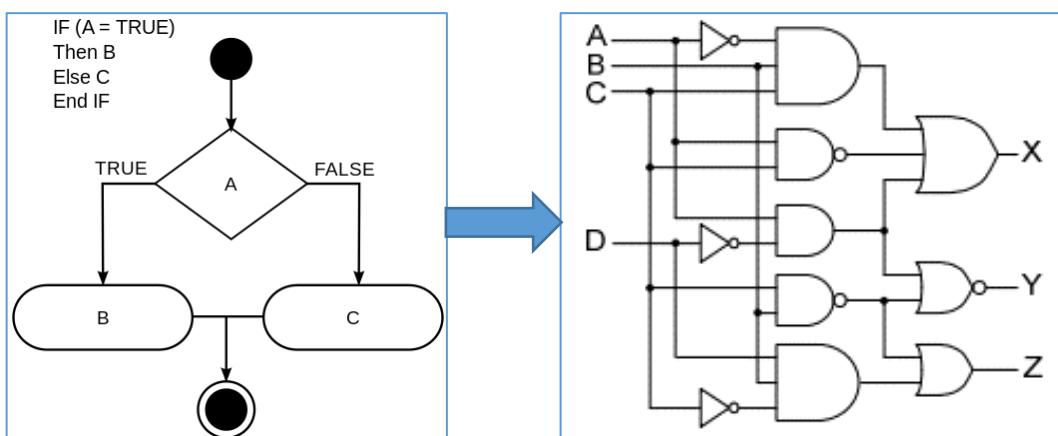
1.1.1 Thực trạng

Thị giác máy tính, Computer Vision, luôn là một lĩnh vực nghiên cứu rất rộng, đã có từ lâu và vẫn đang phát triển mạnh mẽ. Rất nhiều thuật giải khác nhau đã được ứng dụng trong thị giác máy tính, đem lại kết quả tốt, hiệu năng cao, từ các thuật toán đơn giản như histogram cho tới các thuật giải CNN...

Dù hết sức hiệu quả, các thuật giải xử lý ảnh cũng chỉ giải quyết được yếu tố tốc độ nhanh, chứ không xác định được chính xác xử lý trong bao nhiêu lâu. Chính vì vậy, **xử lý ảnh bằng phần mềm khó đáp ứng được yếu tố thời gian thực**, không phải là real-time system, đặc biệt là ở các thiết bị có tốc độ cao như là máy bay, tên lửa, ...

1.1.2 Đề xuất mục tiêu

Ưu thế của một thuật toán mềm là tính linh hoạt, dễ thay đổi, dễ nâng cấp. Tuy nhiên, nhằm đạt tốc độ xử lý nhanh nhất, đáp ứng cho các bài toán đặc thù cho tình huống tốc độ cao như lĩnh vực hàng không vũ trụ, nhóm đề tài đề xuất mục tiêu **cứng hoá thuật toán SIFT** để đạt được ưu thế tuyệt đối về tốc độ. Yếu tố linh hoạt trong trường hợp này có thể tạm gác lại và điều này hoàn toàn hợp lý bởi vì lĩnh vực hàng không luôn đặt cao yếu tố tin cậy, và tốc độ.



1.1.3 Đề xuất giải pháp khả thi

SIFT là một thuật toán xử lý ảnh khá cơ bản và phổ biến trong các ứng dụng. Cứng hoá thuật toán SIFT sẽ giúp đem lại lợi ích cho nhiều ứng dụng và có tính thực tế hơn. Nhóm đề tài quyết định chọn thuật toán này để thực hiện.

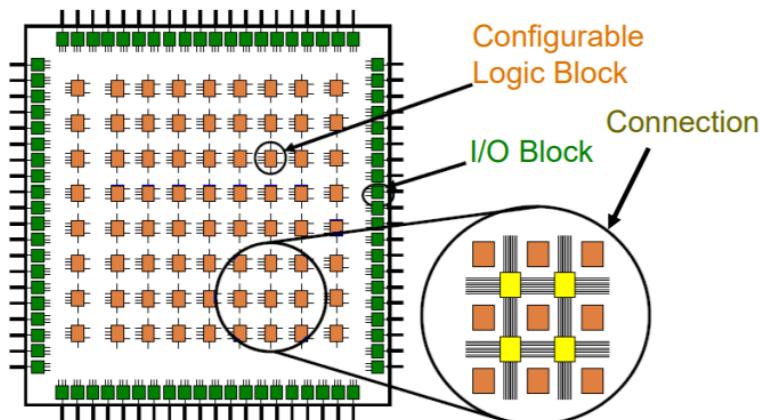
Để cứng hoá, chắc chắn mọi thuật toán cứng sẽ được triển khai trên các chip prototype như FPGA hoặc CPLD. Bởi vì cứng hoá hoàn toàn một thuật toán cứng đòi hỏi khá nhiều cell tính toán và cần nhiều phần tử nhớ nên FPGA phù hợp hơn so với các CPLD vốn mạnh về các tính toán tổ hợp.

Triển khai một thuật toán trên FPGA thường đòi hỏi bước trung gian là mô hình hoá được bài toán và mô phỏng. Hiện nay công cụ hỗ trợ tốt nhất là DSP Builder và MATLAB.

Vì vậy đề tài đưa ra giải pháp khả thi là: thực hiện cứng hoá thuật toán SIFT.

1.2 Giới thiệu về công nghệ FPGA

FPGA là từ viết tắt của ***Field-programmable gate array***, là một mạch tích hợp được thiết kế để người dùng hoặc các nhà thiết kế có khả năng cấu hình sau khi sản xuất (lập trình được). Kiến trúc cơ bản của FPGA gồm ba thành phần chính là các khối logic (Logic Block hay Logic Element) là các đơn vị xử lý, khối vào/ra (IO Cell) dùng giao tiếp với bên ngoài và khối liên kết (Interconnection) dùng để liên kết các khối logic với nhau. Cả ba thành phần đều có thể lập trình được. Để lập trình cho FPGA, người ta sử dụng các ngôn ngữ mô tả phần cứng (Hardware Description Language) phổ biến như VHDL, Verilog, SystemC, ...



Hình 1.1 Kiến trúc cơ bản của FPGA

Theo sự phát triển của công nghệ FPGA, ngày nay FPGA đều tích hợp các khối xử lý chuyên dụng như các khối xử lý DSP, bộ vi xử lý 32-bit, ... nhằm đáp ứng nhu cầu xây dựng các hệ thống xử lý hiệu năng cao của người dùng. Các khối logic được sắp xếp thành một mảng 2 chiều, và các dây liên kết được được tổ chức thành các kênh định tuyến giữa các hàng và các cột của các khối logic. Có 2 loại khối chuyển mạch trong cấu trúc của FPGA: các khối chuyển mạch nằm lân cận một khối logic theo phương ngang và phương dọc được dùng để kết nối cổng vào/ra của khối logic tới các kênh định tuyến; các hộp chuyển mạch nằm ở vị trí góc chéo của các khối logic dùng để kết nối các dây nối giữa kênh định tuyến nằm ngang và kênh định tuyến nằm dọc với nhau. FPGA có khả năng thực hiện bất kỳ một mạch số nào nhờ khả năng tái cấu hình của các khối logic và mạng liên kết. Số lượng các khối logic và tài nguyên định tuyến thay đổi tùy theo từng loại vi mạch và nhà xuất bán ra trên thị trường.

Trên các chip FPGA thường được tích hợp thêm lõi vi xử lý để điều khiển, cấu hình, ... Có hai giải pháp tích hợp là tích hợp lõi vi xử lý cứng (hard core) và tích hợp lõi vi xử lý mềm (soft core). Lõi vi xử lý cứng là một bộ vi xử lý thực hiện

một chức năng chuyên dụng trên chip FPGA, là thành phần có sẵn, hiệu năng cao và không thể thay đổi. Lỗi vi xử lý mềm cho phép cấu hình các khối logic trên FPGA để tạo nên một bộ vi xử lý mong muốn. Như vậy lỗi vi xử lý mềm không có sẵn trên chip FPGA khi người sử dụng mua về mà được tạo ra trong quá trình thiết kế. Lỗi vi xử lý mềm là một giải pháp linh hoạt, cho phép sử dụng các tài nguyên logic trên FPGA một cách hiệu quả. Tuy nhiên, vì lỗi vi xử lý mềm được tạo ra từ việc ghép nối nhiều khối logic trên FPGA lại với nhau nên thường có tần số hoạt động tương đối thấp.

Altera (Intel) là một trong những nhà sản xuất chip FPGA hàng đầu. FPGA của họ cung cấp nhiều loại SRAM nhúng, DSP, bộ thu phát, vào ra tốc độ cao (28 Gbps hoặc nhanh hơn), các khối logic và kết nối có thể tái cấu hình hiệu quả. Các khối logic trong FPGA của Altera được thiết kế dựa trên kiến trúc PAL/PLA. Altera tích hợp các lỗi vi xử lý ARM (hard core) vào FPGA và cung cấp lỗi Nios (soft core) để thực hiện cấu hình cũng như điều khiển chip FPGA. Altera cung cấp các kit phát triển FPGA với môi trường thiết kế hoàn chỉnh, chất lượng cao cho các kỹ sư. Altera được đánh giá cao với các phần mềm, công cụ phát triển có GUI mang cảm giác trực quan như Quartus, Qsys, ModelSim, ... Ngoài ra việc cung cấp các IP Core xây dựng sẵn giúp tăng tốc việc thiết kế, giúp thiết kế hệ thống trở nên đơn giản và thuận tiện hơn.

Đồ án tốt nghiệp này được nghiên cứu và triển khai trên kit DE2i-150, là một kit phát triển FPGA của Altera với chip FPGA Cyclone IV GX. Bộ phần mềm và các công cụ phát triển của Altera đã hỗ trợ sẵn các IP Core xử lý tín hiệu số (DSP), truyền tải, xử lý hình ảnh/video. Các IP Core này đều được thiết kế tương thích với chuẩn giao tiếp Avalon. Do trong đồ án tốt nghiệp này có sử dụng các IP Core nói trên nên việc thiết kế IP Core thực hiện thuật toán trích chọn đặc trưng ảnh SIFT trên hình ảnh/video sẽ sử dụng các giao diện tương ứng trong chuẩn giao tiếp Avalon. Chi tiết về kit DE2i-150, chuẩn giao tiếp Avalon và cơ sở kiến thức về thuật toán sẽ được trình bày ở các phần dưới.

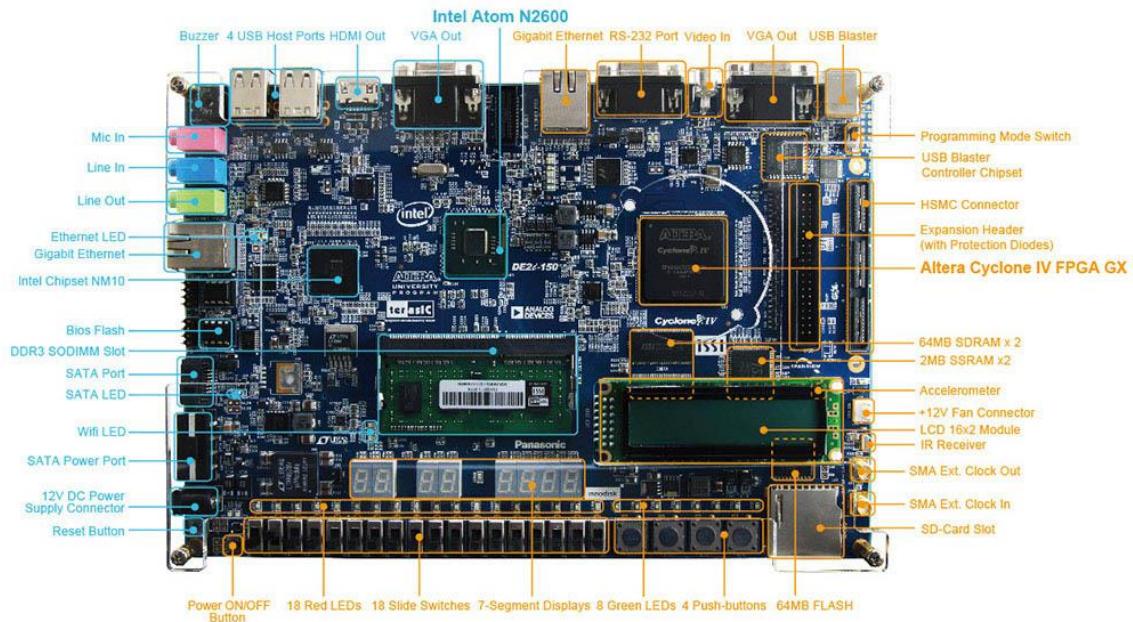
1.3 Kit phát triển DE2i-150

Là một nền tảng nhúng đột phá khi kết hợp bộ vi xử lý nhúng N2600 của Intel với sự linh hoạt của chip FPGA *Altera Cyclone IV GX*, kit phát triển DE2i-150 là một hệ thống máy tính với đầy đủ các tính năng, kết hợp với khả năng xử lý hiệu suất cao và tính cấu hình cao. Chip FPGA *Altera Cyclone IV GX* trên bo mạch DE2i-150 có thể tăng tốc độ đáp ứng của hệ thống nhưng vẫn giữ hiệu quả về mặt chi phí và tiêu thụ điện năng.

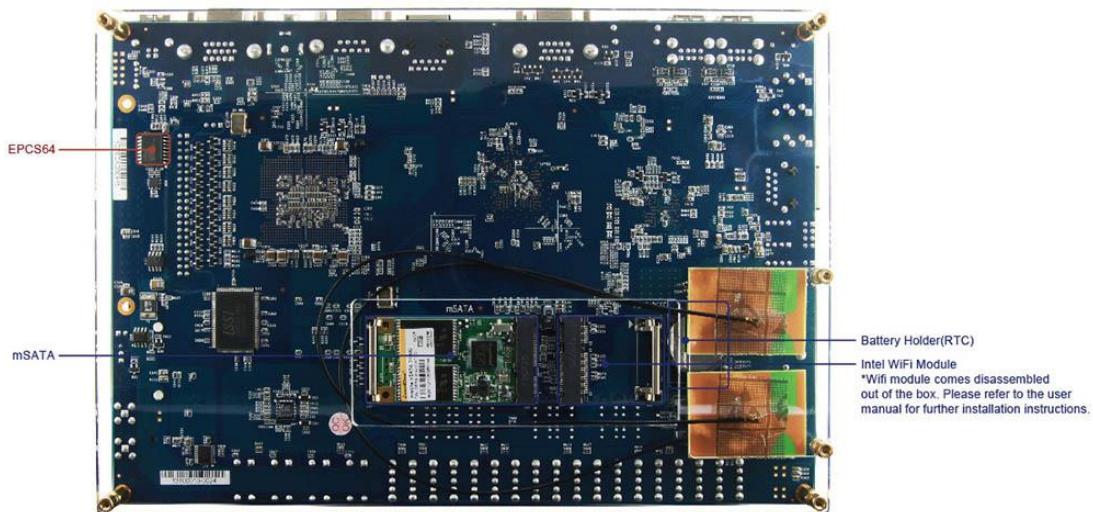
DE2i-150 được trang bị khoảng 150K phần tử logic với tính linh hoạt cao về mặt cấu hình lại mạch phần cứng thực tế và IP Core cũng như các thiết bị ngoại vi đa phương tiện trên bo mạch, các tùy chọn mở rộng sẵn sàng cho mọi tác vụ.

Bộ xử lý *Intel Atom N2600* và chip FPGA *Altera Cyclone IV GX* được liên kết với nhau qua hai làn PCI Express tốc độ cao để đảm bảo giao tiếp tốc độ cao giữa chúng. DE2i-150 cung cấp một môi trường đồng phát triển phần cứng và phần mềm mạnh mẽ và thân thiện.

Dưới đây là hình ảnh thực tế, sơ đồ cũng như cũng như thông tin chi tiết hơn về phần cứng của kit DE2i-150.

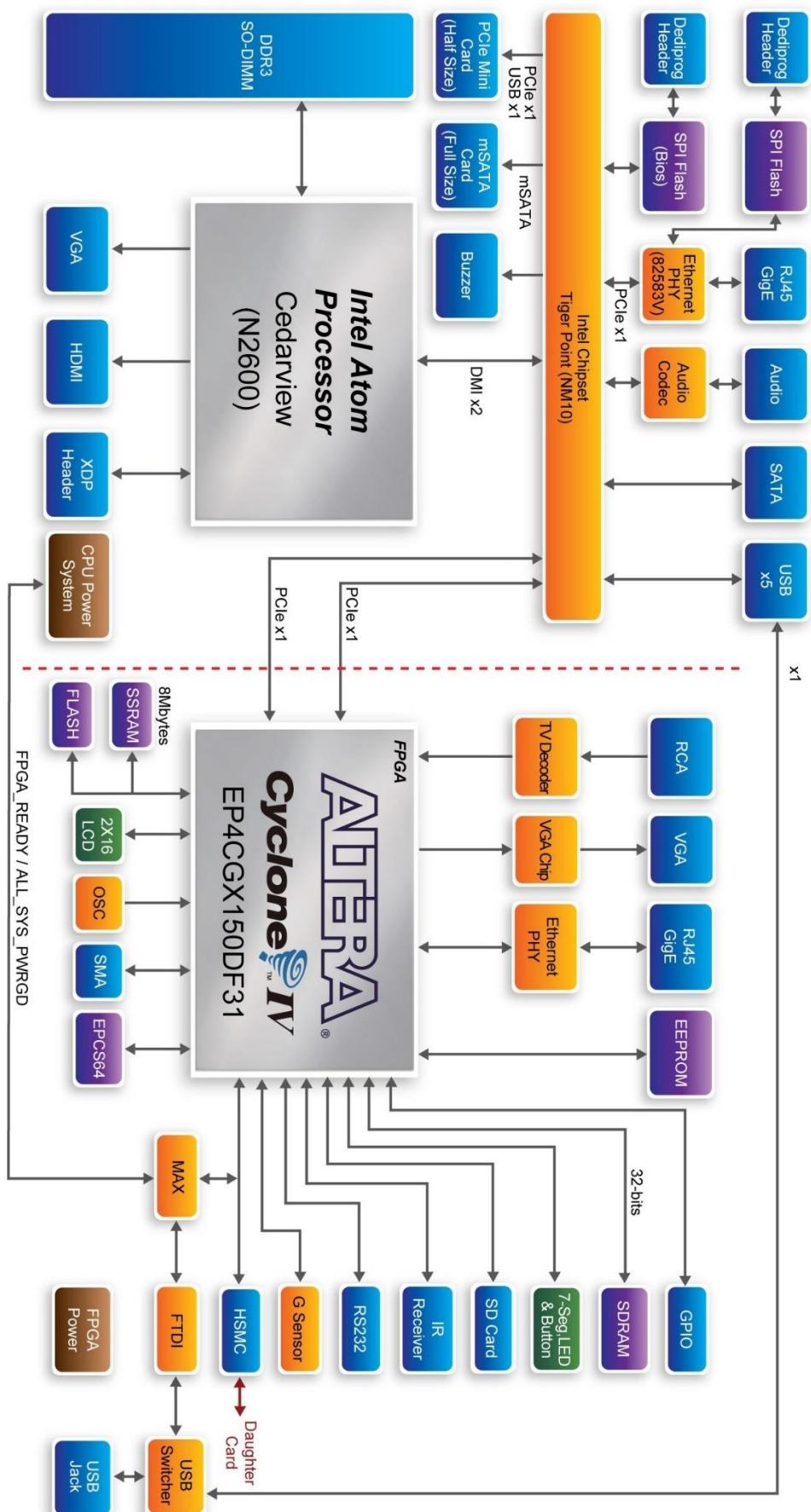


Hình 1.2 Hình ảnh thực tế mặt trên kit DE2i-150 (kèm chú thích)



Hình 1.3 Hình ảnh thực tế mặt dưới kit DE2i-150 (kèm chú thích)

Để thuận tiện về sau, ta sẽ chia kit DE2i-150 thành hai nửa: nửa PC gồm vi xử lý Intel Atom N2600 cùng các thành phần liên quan (chú thích màu xanh trên Hình 1.2) và nửa FPGA gồm chip FPGA Altera Cyclone IV GX cùng các thành phần liên quan (chú thích màu cam Hình 1.2). Sự phân chia này cũng thể hiện bởi đường nét đứt màu đỏ trong Hình 1.4. Nửa PC hoạt động như một máy tính bình thường. Nửa FPGA có thể được cấu hình để hoạt động riêng biệt với nửa PC, cũng có thể cấu hình thành card tăng tốc cho nửa PC.



Hình 1.4 Block Diagram kit DE2i-150

Bảng 1.1 Thông tin chi tiết phần cứng của kit DE2i-150

Featured Devices	Cyclone IV EP4CGX150DF31 device
	149,760 Logic Elements
	720 M9K memory blocks
	6,480 Kbits embedded memory
	8 Phase-Locked Loop
FPGA configuration	JTAG and AS mode configuration
	EPCS64 serial configuration device
	On-board USB Blaster circuitry
Memory devices	128MB (32Mx32bit) SDRAM
	4MB (1Mx32) SSRAM
	64MB (4Mx16) Flash with 16-bit mode
Switches and indicators	18 slide switches
	4 push – buttons
	18 red LEDs
	9 green LEDs
	8 7-segment displays
Connectors	Ethernet 10/100/1000 Mbps ports
	High Speed Mezzanine Card (HSMC)
	40-pin expansion port
	VGA-out connector
	VGA DAC (high speed triple DACs)
	DB9 serial connector for RS-232 port
Clock	Three 50MHz oscillator clock inputs
	2 SMA connectors (external clock in/out)
Display	16x2 LCD module
SD Card socket	Provides SPI and 4-bit SD mode for SD Card access
Other features	Infrared remote-control receiver module
	TV decoder (NTSC/PAL/SECAM)
	TV-in connector

Bảng 1.2 Thông tin chi tiết phần cứng nửa PC của kit DE2i-150

CPU: Intel® Atom™ Dual Core Processor N2600 (1M Cache, 1.6GHz)	Intel® Hyper-Threading Technology (4 execution threads) Intel SpeedStep® Technology Instruction Set: 64-bit Instruction Set Extensions: SSE2, SSE3, SSSE3 Integrated Graphics Graphics Base Frequency: 400MHz
Chipset: Intel® NM10 Express Chipset	DMI x2 to CPU Intel® High Definition Audio Serial ATA (SATA) 3 Gb/s Universal Serial Bus (USB) Hi-Speed USB 2.0 PCI Express Gen 1
Memory	DDR3 SO-DIMM SDRAM
Display	VGA HDMI 1.3a
Intel® Centrino® Wireless-N 135	802.11b/g/n Bluetooth 4.0 Wi-Fi Direct
Audio Codec	Realtek ALC272VA3-GR
BIOS	DIP package Bios Flash: GD25Q16 Programming Interface for Bios: Dedi-Prog Interface
Debug Interface	XDP header
Clock System	CK505: 9VRS4339B 32768 Hz RTC crystal 27MHz VGA clock source
Ethernet	Intel® 82583V GbE Controller 10/100/1000 Mb/s RJ45 3 status indicating LEDs
Others	Power header for hard-disk Current limit for USB Buzzer Mini PCIE header (Default for Intel® Centrino® Wireless-N 135 WiFi module) mSATA header RTC battery: CR2032

1.4 Chuẩn giao tiếp Avalon

1.4.1 Tổng quan về chuẩn giao tiếp Avalon

Chuẩn giao tiếp Avalon đơn giản hóa việc thiết kế hệ thống bằng cách cho phép ghép nối trực tiếp các thành phần trong FPGA của Intel. Chuẩn giao tiếp này định nghĩa các giao diện thích hợp truyền dữ liệu tốc độ cao, đọc ghi các thanh ghi và bộ nhớ cũng như điều khiển các thiết bị ngoại vi, ... Chi tiết về các giao diện này được mô tả trong [1].

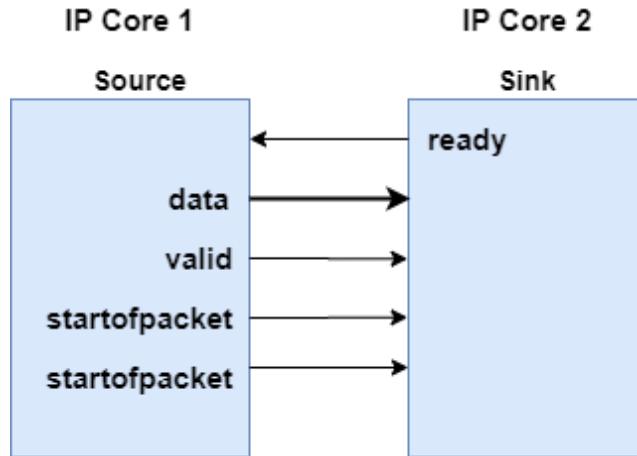
Bảng 1.3 Các giao diện trong chuẩn giao tiếp Avalon

Tên giao diện	Đặc điểm
Avalon Streaming Interface (Avalon-ST)	Hỗ trợ truyền dữ liệu với băng thông cao, độ trễ thấp, một chiều, không quan tâm đến địa chỉ, bao gồm các luồng, các gói và các tín hiệu số được ghép kênh với nhau. Thích hợp truyền dữ liệu ảnh, video, các tín hiệu số, ...
Avalon Memory Mapped Interface (Avalon-MM)	Sử dụng để đọc/ghi dữ liệu dựa trên địa chỉ trong mô hình kết nối master – slave điển hình như vi điều khiển (Microprocessors), bộ nhớ (Memories), thanh ghi, UARTs, DMAs, các bộ định thời (Timers), ...
Avalon Conduit Interface	Giao diện chứa các tín hiệu riêng lẻ hoặc các nhóm tín hiệu không tương thích với với các giao diện Avalon nào khác. Chúng có thể được kết nối với các giao diện Conduit khác trong cùng Platform Designer, ngoài ra cũng có thể xuất và kết nối chúng với các mô-đun khác hoặc các ngoại vi như switch, button, ... của FPGA.
Avalon Tri-State Conduit Interface (Avalon-TC)	Hỗ trợ kết nối với các thiết bị ngoại vi bên ngoài FPGA. Thích hợp sử dụng ở đầu vào hoặc đầu ra.
Avalon Interrupt Interface	Cho phép gửi tín hiệu ngắt đến các components khác.
Avalon Clock Interface	Điều khiển hoặc nhận tín hiệu clock.
Avalon Reset Interface	Cung cấp kết nối cho tín hiệu reset.

1.4.2 Sử dụng giao diện Avalon Streaming để truyền ảnh/video

Như đã đề cập ở phần trên, Avalon Streaming Interface (Avalon-ST) cho phép truyền dữ liệu với băng thông cao, độ trễ thấp, truyền theo một chiều, ...,

thích hợp sử dụng truyền tín hiệu hình ảnh, video, tín hiệu số. Trên thực tế, hầu hết các IP Core xử lý dữ liệu dạng hình ảnh/video được cung cấp bởi Altera (Intel) đều sử dụng giao diện Avalon – ST để truyền dẫn dữ liệu hình ảnh/video từ IP Core này sang IP Core khác. Giao diện Avalon – ST trong các IP Core này truyền dữ liệu một chiều từ Source (đầu ra của IP Core 1) sang Sink (đầu vào của IP Core 2) theo sơ đồ sau:



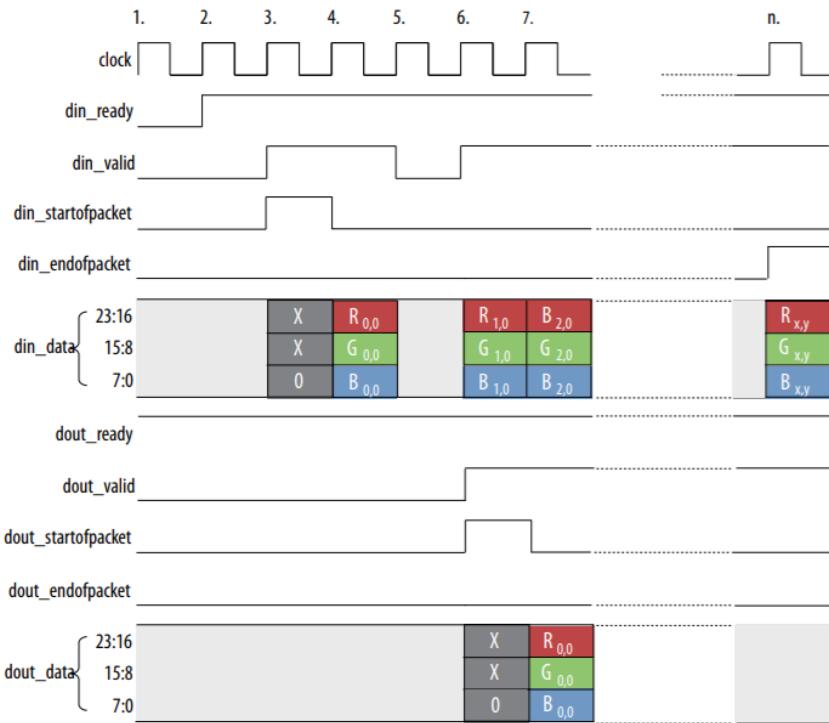
Hình 1.5 Tín hiệu truyền gói hình ảnh bằng giao diện Avalon – ST

Bảng 1.4 Danh sách các tín hiệu của giao diện Avalon – ST sử dụng trong đồ án

Tên tín hiệu	Độ rộng bus	Chiều truyền	Mô tả đặc điểm
data	1 – 8192	Source → Sink	Dữ liệu mang phần lớn thông tin được truyền từ source đến sink
ready	1	Sink → Source	Báo hiệu phía sink có sẵn sàng để nhận dữ liệu hay không.
valid	1	Source → Sink	Báo hiệu các tín hiệu đang truyền từ source đến sink là hợp lệ.
End of packet	1	Source → Sink	Đánh dấu kết thúc một gói tin.
Start of packet	1	Source → Sink	Đánh dấu bắt đầu một gói tin.

Trong đồ án tốt nghiệp này, dữ liệu cần xử lý là dữ liệu ảnh/video với định dạng màu RGB 24-bit, dùng mỗi 8-bit để biểu diễn giá trị một kênh màu, giá trị của cả ba kênh màu của từng pixel được truyền song song tại cùng một thời điểm, do đó độ rộng bus *data* là 24.

Ta xét một ví dụ về một IP Core có một cổng Avalon – ST tên là *din* (cổng vào) và một cổng Avalon – ST tên là *dout* (cổng ra). Dữ liệu sẽ đi vào IP Core từ cổng *din* và đi ra khỏi IP Core thông qua cổng *dout*.



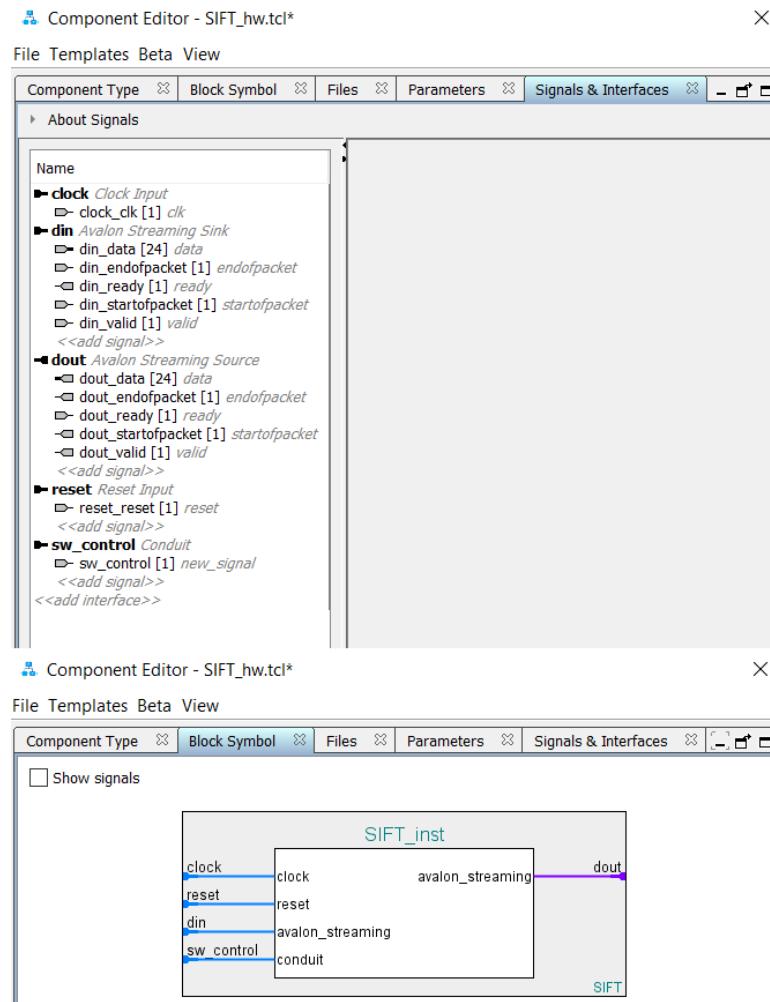
Hình 1.6 Timing Diagram Showing R'G'B' Transferred in Parallel

Mỗi cổng **din** và **dout** đều chứa 5 tín hiệu: *ready*, *valid*, *data*, *startofpacket*, *endofpacket*. Tín hiệu *din_ready* cho biết IP Core có sẵn sàng nhận dữ liệu hay không, nó là một đầu ra (output) của IP Core được gắn vào cổng **din**. Các tín hiệu còn lại của cổng **din** đều là đầu vào (input) của IP Core. Tín hiệu *din_valid* sẽ báo hiệu dữ liệu đang được truyền là hợp lệ, có nghĩa dữ liệu đang truyền trên tín hiệu *din_data* là dữ liệu pixel của ảnh. Tín hiệu *din_data* truyền dữ liệu các pixel, 24-bit mang thông tin các kênh màu RGB của từng pixel sẽ được truyền song song tại mỗi chu kỳ clock nếu cả *din_ready* và *din_valid* đều ở mức logic ‘1’. Tín hiệu *din_startofpacket* báo hiệu bắt đầu một packet mới và tín hiệu *din_endofpacket* báo hiệu kết thúc một packet. Các tín hiệu của cổng **dout** tương tự như cổng **din** nhưng ngữ nghĩa ngược lại.

Trong Hình 1.6, ta thấy ban đầu *din_ready* ở mức logic ‘0’, báo hiệu rằng IP Core không sẵn sàng để nhận dữ liệu từ cổng **din** và không có dữ liệu các pixel được truyền vào. Đến chu kỳ 2, *din_ready* được nâng lên mức logic ‘1’, IP Core sẵn sàng nhận dữ liệu sau một chu kỳ clock. Số chu kỳ trễ này được quyết định bởi tham số tên *ready latency*, có thể cài đặt khi thiết kế và tất cả các IP Core trong bộ *Video and Image Processing* được cung cấp bởi Altera đều đặt giá trị tham số này là 1. Tại chu kỳ 3, *din_valid* được nâng lên logic ‘1’ để báo hiệu dữ liệu hình ảnh/video được gửi qua *din_data* là hợp lệ. Đồng thời, *din_startofpacket* được nâng lên logic ‘1’ để báo hiệu giá trị được truyền là giá trị đầu tiên của một gói (pixel ở góc trái trên). Ở đây giá trị này bằng 0, có nghĩa là gói này là dữ liệu video. Chu kỳ 4, *din_valid* vẫn giữ ở mức logic ‘1’, *din_startofpacket* thì bị kéo xuống logic ‘0’, báo hiệu rằng đang gửi phần thân gói vào IP Core. Chu kỳ 5, *din_ready* vẫn ở logic ‘1’ báo hiệu IP Core vẫn sẵn sàng

nhận dữ liệu, nhưng có thể do không có dữ liệu để gửi nên *din_valid* bị kéo xuống logic ‘0’. Ở chu kỳ 6, việc truyền dữ liệu tiếp tục diễn ra bình thường và *din_valid* được nâng lên logic ‘1’. Đồng thời, IP Core bắt đầu truyền dữ liệu qua cổng ***dout*** (trong ví dụ này giả định IP Core có độ trễ bên trong là 3 chu kỳ clock, với các IP Core được cung cấp sẵn bởi Altera hay IP Core sẽ thiết kế trong đồ án này thì giá trị này sẽ lớn hơn nhiều và khác biệt tùy các bộ xử lý dữ liệu bên trong của IP Core). Đến chu kỳ n, *din_endofpacket* được đặt thành logic ‘1’ trong 1 chu kỳ báo hiệu đã truyền xong pixel cuối cùng của gói (pixel ở góc phải dưới).

Trong đồ án này, để thuận tiện cho việc ghép nối giữa các IP Core, IP Core thực hiện thuật toán SIFT sẽ được thiết kế tương tự ví dụ trên với các cổng vào ra như hình sau:



Hình 1.7 Thiết kế giao diện vào ra cho IP Core SIFT

Về cơ bản thiết kế này khá giống như ví dụ vừa phân tích phía trên với đầu vào ***din*** và đầu ra ***dout*** đều sử dụng giao diện Avalon – ST để truyền dữ liệu hình ảnh/video. Ngoài ra còn có thêm cổng ***clock*** sử dụng giao diện Avalon Clock, cổng ***reset*** sử dụng giao diện Avalon Reset để truyền các tín hiệu clock và reset cho IP Core. Bên cạnh đó, thiết kế còn có thêm cổng ***sw_control*** với giao diện Avalon Conduit để truyền tín hiệu ngoại vi từ các switch trên board DE2i-150 dùng điều khiển, cấu hình trong IP Core.

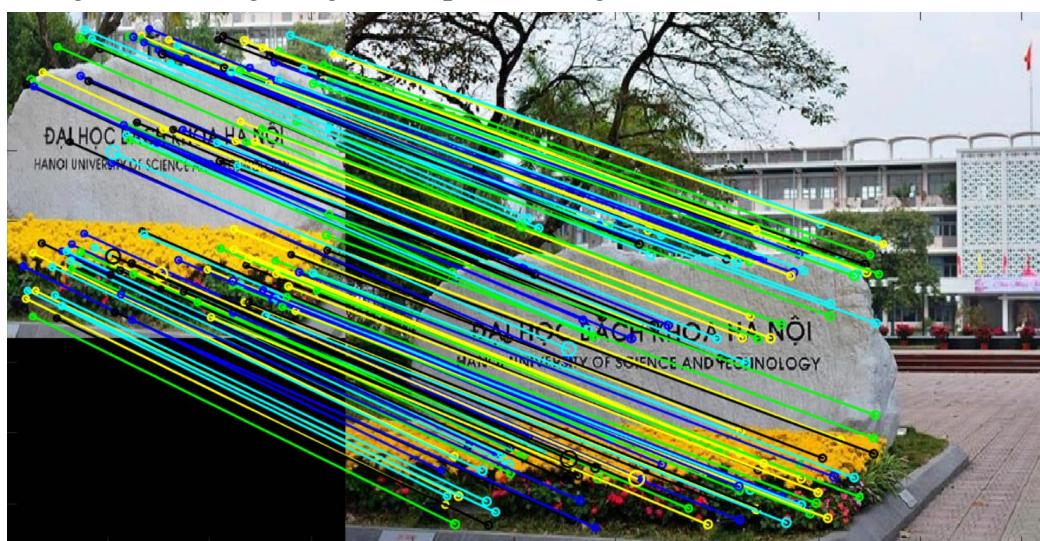
CHƯƠNG 2. THUẬT TOÁN TRÍCH CHỌN ĐẶC TRUNG ẢNH SIFT

2.1 Giới thiệu thuật toán SIFT

Trong lĩnh vực xử lý ảnh, những đặc trưng trong ảnh là một phần quan trọng của ảnh. Các đặc trưng về ảnh là những kết cấu đặc biệt như: các cạnh, các điểm của đối tượng trong ảnh. Trích chọn đặc trưng là quá trình xử lý hình ảnh ban đầu để thu được những dạng mà máy tính có thể nhận dạng dễ dàng do máy tính không có những khả năng như thị giác con người (hình ảnh máy tính hiểu đó chính là các ma trận điểm ảnh). Việc trích chọn đặc trưng trong ảnh có vai trò quan trọng trong việc phân tích các hình ảnh tương đồng, có thể phát triển thành những công nghệ ghép nối ảnh, xây dựng ảnh 3D....

Việc trích chọn các điểm đặc trưng của ảnh là công việc cơ bản và cũng là nền tảng trong lĩnh vực xử lý ảnh, bởi vậy đã có rất nhiều thuật toán ra đời để thực hiện công việc đó. Phương pháp tìm kiếm những điểm nổi bật chính xác và cơ bản nhất trong xử lý ảnh là tìm các điểm nội biên của ảnh, các điểm nằm trên góc cạnh của đối tượng trên ảnh. Một số thuật toán thông thường thì chỉ tìm được góc cạnh ở một tỷ lệ zoom nhất định của ảnh và chưa hiệu quả với phép zoom ảnh ví dụ như thuật toán tìm kếm góc Harris.

Để khắc phục vấn đề trên thuật toán SIFT (Scale Invariant Feature Transform) đã được đưa ra năm 1999 bởi tác giả David Lowe. SIFT là một kỹ thuật để phát hiện tìm kiếm các điểm đặc trưng (Keypoints) trong ảnh, với mỗi điểm hấp dẫn sẽ được cung cấp một bộ mô tả (descriptor) của vùng ảnh nhỏ xung quanh dùng để đối sánh và nhận dạng đối tượng trong ảnh. Ưu điểm của thuật toán là có tính bất biến với phép dịch (translation), phép xoay (rotation) và phép zoom-out (scaling). Ngoài ra thuật toán còn ít bị ảnh hưởng bởi các phép biến đổi ảnh như nhiễu, mờ, độ tương phản, thay đổi về điểm quan sát và có đủ tính phân biệt dùng cho các ứng dụng so khớp (matching).



Hình 2.1 Ứng dụng so khớp 2 ảnh sử dụng thuật toán SIFT

Hình 2.1 là kết quả khi ứng dụng thuật toán SIFT để so khớp 2 ảnh. Những điểm hấp dẫn đã match với nhau được khoanh tròn và nối với nhau. Từ Hình 2.1

cho thấy SIFT đã hoạt động rất tốt mặc dù cùng một đối tượng nhưng trong 2 ảnh đã có sự khác nhau về kích thước, có độ biến dạng và độ sáng khác nhau.

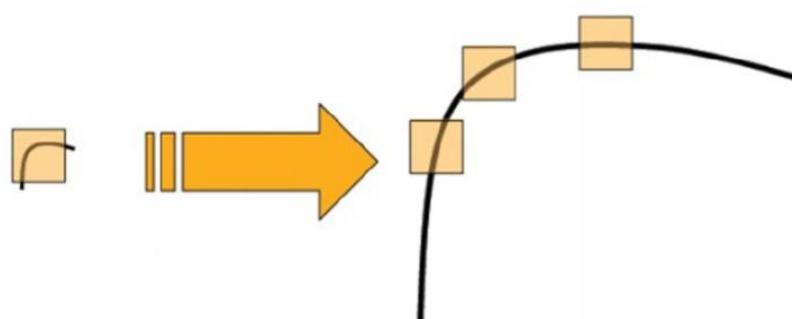
Thuật toán có nhiều ưu điểm trong công việc đối sánh và nhận dạng đối tượng trong ảnh, nhưng khi ứng dụng với một cơ sở dữ liệu ảnh lớn, việc đối sánh cần một chi phí tính toán rất lớn do có số lượng lớn các điểm đặc trưng ở một ảnh và các điểm đặc trưng trong một ảnh phải được so sánh với tất cả điểm đặc trưng trong ảnh còn lại.

2.2 Các bước thực hiện thuật toán

Thuật toán SIFT bao gồm các bước:

- Bước 1. Phát hiện những điểm cực trị trong không gian tỷ lệ (Scale-space Extrema Detection): Sử dụng hàm sai khác Gaussian (DoG – Difference of Gaussian) để tìm kiếm các điểm có khả năng làm điểm đặc trưng tiềm năng (những điểm có khả năng ít phụ thuộc vào sự thu phóng và xoay ảnh).
- Bước 2. Xác định vị trí những điểm hấp dẫn (Keypoint localization): Từ những điểm cực trị ở bước 1 sẽ được lọc và lấy ra tập các điểm đặc trưng tốt nhất (Keypoint).
- Bước 3. Xác định hướng và độ lớn biến đổi của những điểm hấp dẫn (Oriented Assignment): Mỗi đặc trưng sẽ được gán hướng gradient và độ lớn biến đổi trong ảnh.
- Bước 4. Tạo bộ mô tả cho các điểm hấp dẫn trong ảnh (Keypoint Descriptor): Lấy histogram hướng biến đổi của những vùng lân cận với mỗi điểm đặc trưng để biểu diễn thành một bộ mô tả.

2.2.1 Phát hiện những điểm cực trị trong không gian tỷ lệ



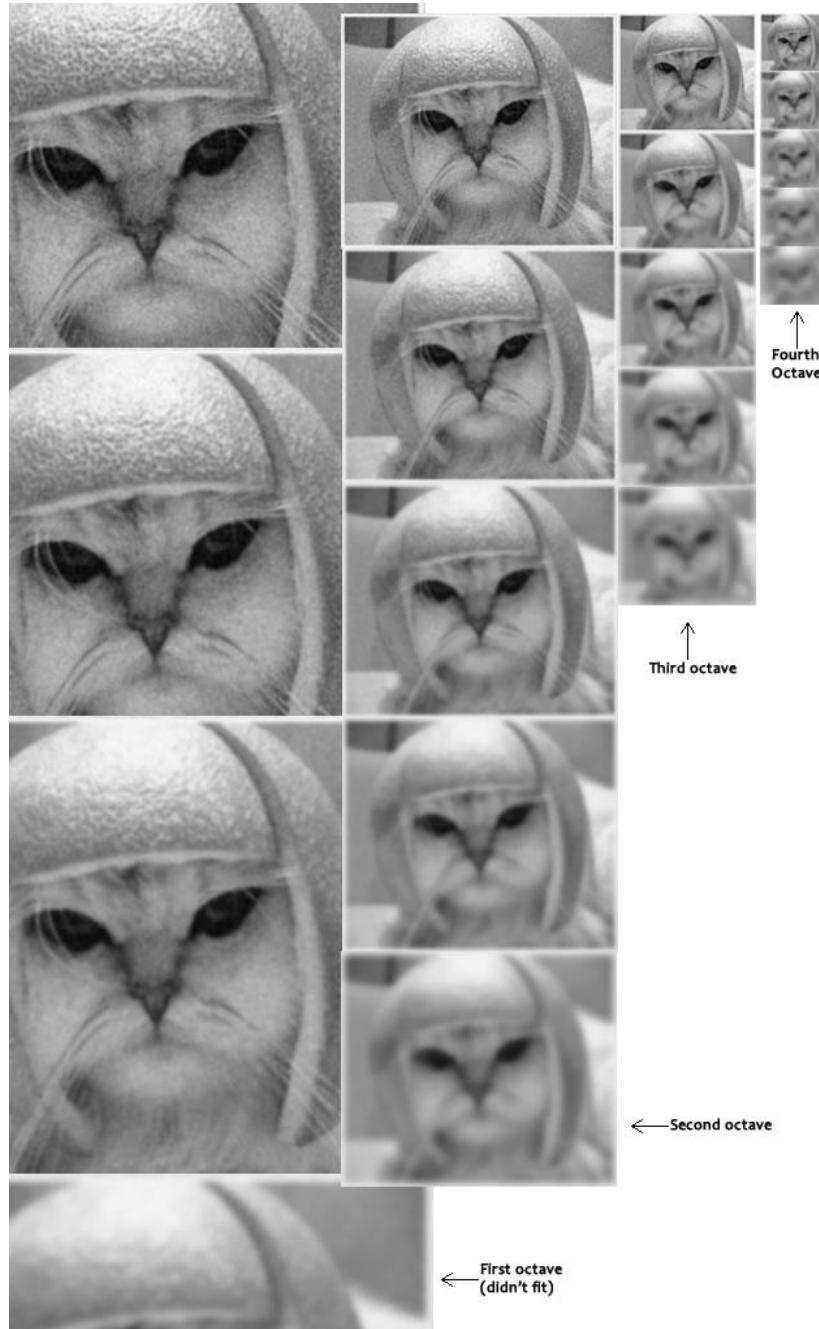
Hình 2.2 Sự thay đổi của đối tượng khi thay đổi kích thước

Từ minh họa Hình 2.2, ta thấy hình ảnh được cắt ra bên trái là một góc của đối tượng trong ảnh đã được phát hiện bằng thuật toán tìm kiếm góc Harris. Mặt khác nếu như xét góc này với một tỉ lệ zoom lớn hơn nhiều thì nó không còn là một góc theo thuật toán tìm kiếm góc Harris, nhưng trong thực tế nó hoàn toàn là một góc.

Để khắc phục vấn đề đó thuật toán SIFT đã tạo ra một không gian tỷ lệ Scale-space của hình ảnh là một hàm tỷ lệ $L(x,y,\sigma)$ được tính bằng cách nhân

chập mặt nạ Gaussian $G(x,y, \sigma)$ có tham số σ với từng tỷ lệ của ảnh đầu vào. Thực hiện tìm kiếm phát hiện những điểm hấp dẫn trên không gian Scale-space giúp thuật toán SIFT bất biến với phép zoom và các phép làm mờ ảnh.

Mỗi tỷ lệ của ảnh đầu vào sẽ tạo ra một Octave (số lượng Octave có thể tạo ra phụ thuộc vào kích thước ban đầu của ảnh đầu vào, xử lý trên ảnh có kích thước càng lớn thì có thể tạo nhiều octave). Trong thuật toán SIFT áp dụng kỹ thuật downsampled (kích thước của ảnh trên mỗi octave sẽ giảm một nửa so với octave trước nó, octave đầu tiên sẽ có kích thước bằng ảnh gốc) (Hình 2.3).



Hình 2.3 Ví dụ Scale space

Công thức tính hàm tỷ lệ:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

Trong đó:

- Toán hạng * : Phép nhân chập (convolution) 2 ma trận.
- $I(x, y)$: Ảnh đầu vào.
- $G(x, y, \sigma)$: Hàm Gaussian có tham số σ .

Công thức hàm Gaussian:

$$G_{(x,y,\sigma)} = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Công thức phép toán nhân chập:

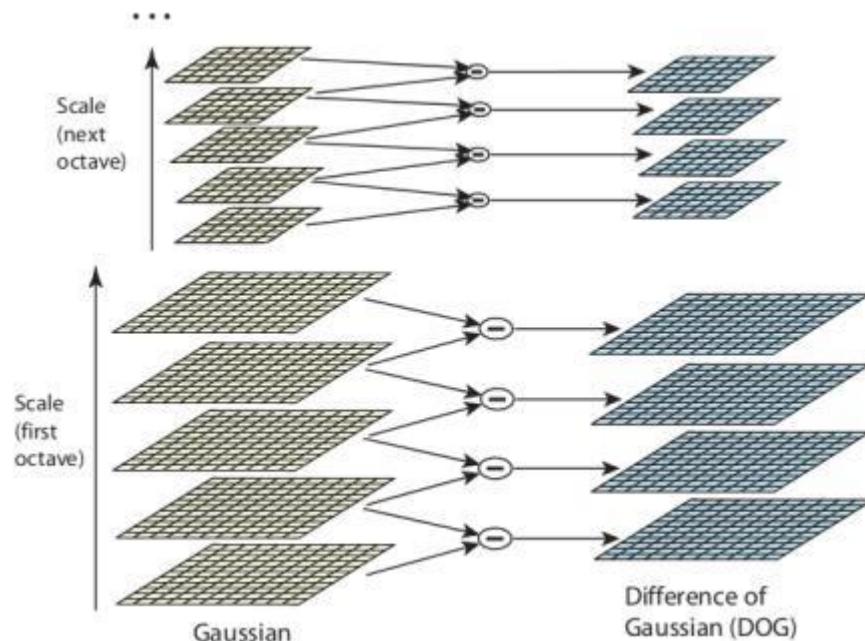
$$Y(m, n) = X(m, n) * H(k, l) = \sum_{k=-r}^{r} \sum_{l=-r}^{r} X(m-k, n-l)H(k, l)$$

Trong đó:

- $X(m, n)$: Ma trận ban đầu của ảnh kích thước $m \times n$.
- $H(k, l)$: Ma trận hạt nhân của phép nhân chập hay còn gọi mặt nạ.
- $Y(m, n)$: Ma trận đầu ra của phép nhân chập giữa X và H

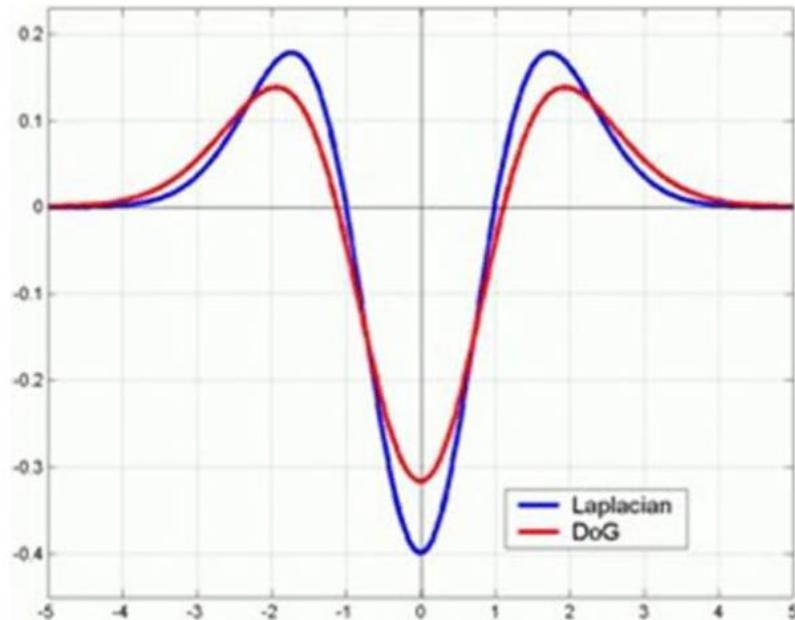
Muốn tìm những điểm hấp dẫn có tính bát biến cao trong ảnh, ta phải tìm các điểm cực trị cục bộ (cực đại/cực tiểu địa phương) bằng cách dùng hàm sai khác DoG (Difference of Gaussian), ký hiệu là $D(x, y, \sigma)$. Hàm DoG được tính toán từ sự sai khác giữa hai không gian tỉ lệ của ảnh với tham số σ lệch nhau bằng một hằng số k theo công thức:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y)$$



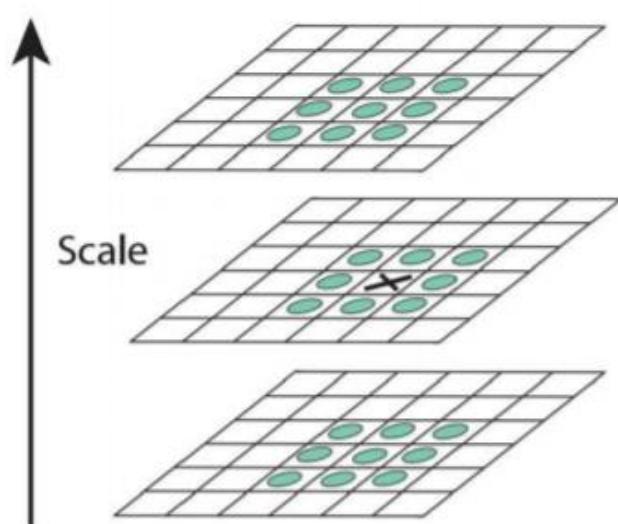
Hình 2.4 Mô tả quá trình tính không gian tỉ lệ (L) và hàm sai khác (D)

DoG được sử dụng để tạo ra sự xấp xỉ với đạo hàm bậc hai Laplace có kích thước chuẩn của hàm Gaussian (Laplacian of Gaussian) do tác giả Lindeberg đề xuất năm 1994. Ông đã chỉ ra rằng việc chuẩn hóa đạo hàm bậc hai với hệ số σ là cần thiết cho các giá trị cực đại và cực tiểu là những giá trị có tính ổn định nhất (tính bất biến) so với các hàm khác như: gradient, hessian hay Harris. Do DoG xấp xỉ như LoG (Laplacian of Gaussian) (Hình 2.5) nhưng chi phí tính toán thấp hơn rất nhiều so với LoG.



Hình 2.5 DoG và Laplacian of Gaussian

Sau khi áp dụng tính DoG ta thu được các lớp kết quả khác nhau (scale) từ ảnh gốc, bước tiếp theo là tìm các cực trị trong các lớp kết quả theo từng miền cục bộ. Mỗi điểm trên các lớp DoG sẽ được so sánh với 8 điểm lân cận trên cùng lớp và 9 điểm lân cận trên lớp trên và 9 điểm lân cận trên lớp dưới như Hình 2.6.



Hình 2.6 Quá trình tìm điểm cực trị trong các hàm sai khác DoG

2.2.2 Xác định vị trí những điểm hấp dẫn

Sau khi thực hiện tìm kiếm những điểm cực trị tiềm năng bằng hàm sai khác DoG ta sẽ thu được rất nhiều điểm có thể làm điểm hấp dẫn, trong số những điểm đó có một số điểm không cần thiết. Ở bước này, ta sẽ thực hiện loại bỏ những điểm không cần thiết có xu hướng nằm trên cạnh (đường biên của đối tượng), có độ tương phản kém (nhạy cảm với nhiễu) hoặc tính đặc trưng cục bộ ít hơn những điểm khác. Thực hiện bước này trong 3 công đoạn:

- a) Sử dụng phép nội suy lân cận để xác định vị trí các điểm hấp dẫn.

Phép nội suy lân cận sử dụng mở rộng Taylor (Taylor expansion) cho hàm Difference of Gaussian $D(x,y,\sigma)$:

$$D(x) = D + \left(\frac{\partial D}{\partial x} \right)^T x + \frac{1}{2} x^T \frac{\partial^2 D}{\partial x^2} x$$

Trong đó: D và đạo hàm của nó được tính tại một thời điểm tiềm năng và $X(x,y,\sigma)$ là khoảng cách từ điểm đó. Vị trí của điểm cực trị \hat{x} được xác định bằng cách lấy đạo hàm của hàm trên với đối số X và tiến dần đến 0.

$$\hat{x} = - \left(\frac{\partial^2 D}{\partial x^2} \right)^{-1} \frac{\partial D}{\partial x}$$

Nếu $\hat{x} > threshold$ cho trước theo một chiều nào đó thì nó có chỉ số cực trị không gần với các điểm tiềm năng khác, nó sẽ bị thay đổi và phép nội suy sẽ thay thế vai trò của nó bằng điểm khác gần nó.

- b) Loại trừ các điểm có tính tương phản kém.

Các điểm nhạy cảm với độ sáng và nhiễu thì sẽ bị loại bỏ. Tiếp tục sử dụng kết quả tính giá trị \hat{x} trong khai triển Taylor mở rộng ở trên, nếu những điểm nào có giá trị $\hat{x} < 0.03$ thì điểm đó sẽ bị loại, ngược lại thì nó được giữ lại theo vị trí mới $(y + \hat{x})$ với giá trị là vị trí cũ của nó cùng giá trị biến σ .

- c) Loại bỏ các điểm dư thừa theo biên.

Khi sử dụng hàm DoG sẽ cho tác động mạnh đến cạnh khi vị trí của cạnh là khó xác định vì vậy các điểm tiềm năng trên cạnh sẽ không bắt biến và bị nhiễu. Để tăng tính ổn định ta sẽ loại bỏ những điểm dễ bị ảnh hưởng thay đổi khi có nhiễu do nằm trên biên. Giải pháp là sử dụng giá trị của ma trận Hessian cấp 2:

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Tính $tr(H)$ và $det(H)$ nếu $\frac{tr(H)^2}{det(H)} < \frac{(C_{edge}+1)}{C_{edge}}$ (với C_{edge} mặc định = 10) thì keypoint đó được giữ lại, nếu không thì sẽ bị loại bỏ [2].

2.2.3 Xác định hướng và độ lớn biến đổi của những điểm hấp dẫn

Việc gán hướng cho mỗi điểm hấp dẫn (keypoint) dựa vào các thuộc tính ảnh cục bộ, descriptor của keypoint sẽ được biểu diễn tương đối theo hướng này và do đó đạt được tính bắt biến đối với các hiện tượng quay ảnh.

Tính độ lớn và hướng của các điểm hấp dẫn được xác định theo công thức:

$$m(x, y) = \sqrt{(L(x + 1, y) - L(x - 1, y))^2 + (L(x, y + 1) - L(x, y - 1))^2}$$

$$\theta(x, y) = \tan^{-1}((L(x, y + 1) - L(x, y - 1))/(L(x + 1, y) - L(x - 1, y)))$$

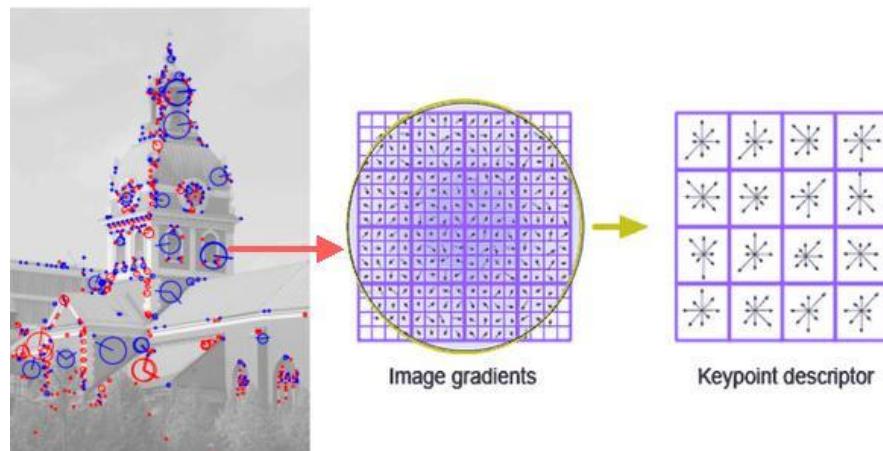
Trong đó:

- $m(x, y)$: Độ lớn của vector định hướng
- $\theta(x, y)$: Hướng của vector định hướng (biểu diễn qua góc θ)

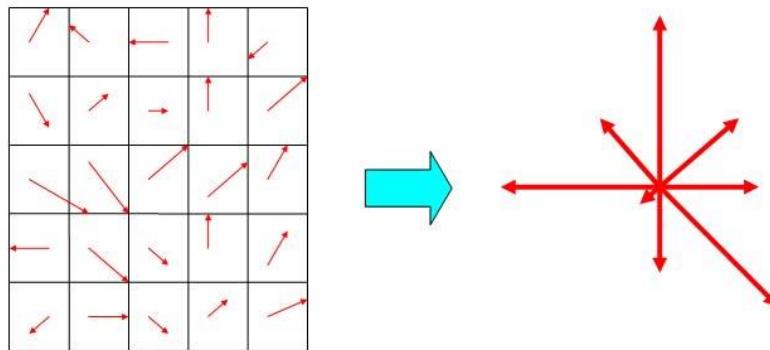
2.2.4 Tạo bộ mô tả cho các điểm hấp dẫn trong ảnh

Các phép xử lý trên đã thực hiện tìm kiếm phát hiện và gán tọa độ, kích thước và hướng cho mỗi điểm hấp dẫn (keypoint). Bước tiếp theo sẽ tính toán một bộ mô tả (descriptor) cho một vùng ảnh địa phương mà có chứa điểm hấp dẫn có tính đặc trưng cao (bất biến với các phép thay đổi khác nhau về độ sáng, zoom ảnh, xoay).

Một cách tiếp cận đơn giản là lấy histogram của những vùng ảnh cục bộ nhỏ lân cận điểm đặc trưng. Tính toán trên mặt nạ 16×16 chia thành 4×4 khu vực quanh điểm hấp dẫn. Mỗi khu vực 4×4 sẽ tính histogram của những điểm ảnh trong đó. Hướng biến đổi của điểm ảnh sẽ được lấy mẫu thành 8 bin như Hình 2.7. Từ đó histogram sẽ gồm 8 chiều mỗi chiều sẽ được tính bằng tổng độ lớn biến đổi của các điểm ảnh có cùng hướng biến đổi trong khu vực đó. Điểm hấp dẫn sẽ được biểu diễn dưới dạng vector $4 \times 4 \times 8 = 128$ chiều (số chiều = 8 hướng x (4×4) khu vực lân cận điểm hấp dẫn).



Hình 2.7 Xây dựng descriptor cho Keypoint



Hình 2.8 Lấy mẫu hướng biến đổi của điểm ảnh

2.3 Kỹ thuật đối sánh (Matching)

Trước tiên để đối sánh các ảnh với nhau thì cần trích xuất tập descriptor tương ứng với mỗi ảnh từ các bước ở trên. Sau đó việc đối sánh sẽ thực hiện trên các tập descriptor này. Công việc chính của đối sánh là thực hiện tìm tập con descriptor so khớp với nhau ở hai ảnh. Tập con descriptor khớp với nhau là vùng ảnh chứa đối tượng giống nhau (hoặc tương đồng).

Việc đối sánh thực hiện vét cạn tất cả các trường hợp trên từng keypoint, bởi vậy khi xử lý đối sánh trên một tập cơ sở dữ liệu lớn sẽ gặp nhiều khó khăn do chi phí thực hiện tính toán lớn.

Để so sánh 2 keypoint với nhau ta thực hiện tính khoảng cách giữa các descriptor của 2 keypoint. Nếu khoảng cách này nhỏ hơn một ngưỡng xác định trước thì được cho là match với nhau.

Giả sử \mathcal{L}^A và \mathcal{L}^B là 2 tập descriptor của các điểm đặc trưng tìm được tương ứng từ 2 ảnh A và B. Với mỗi descriptor a thuộc \mathcal{L}^A phải so sánh với tất cả descriptor b thuộc \mathcal{L}^B .

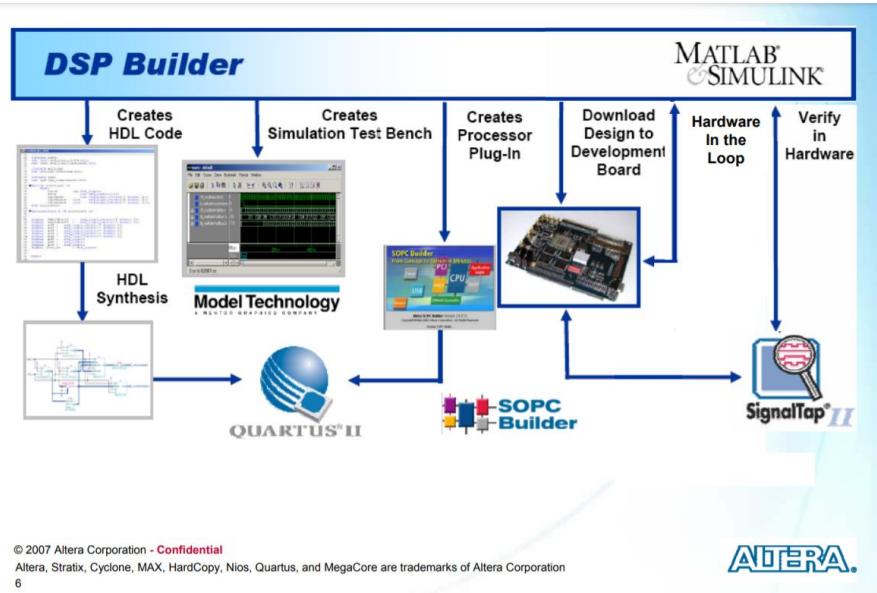
$$a = (a_1, a_2, a_3, \dots, a_{128}) \text{ và } b = (b_1, b_2, b_3, \dots, b_{128})$$

$$D(a, b) = \sum (a_i - b_i)^2$$

CHƯƠNG 3. THIẾT KẾ VÀ MÔ PHỎNG TRÊN MATLAB

3.1 Giới thiệu các công cụ sử dụng để thiết kế và thử nghiệm

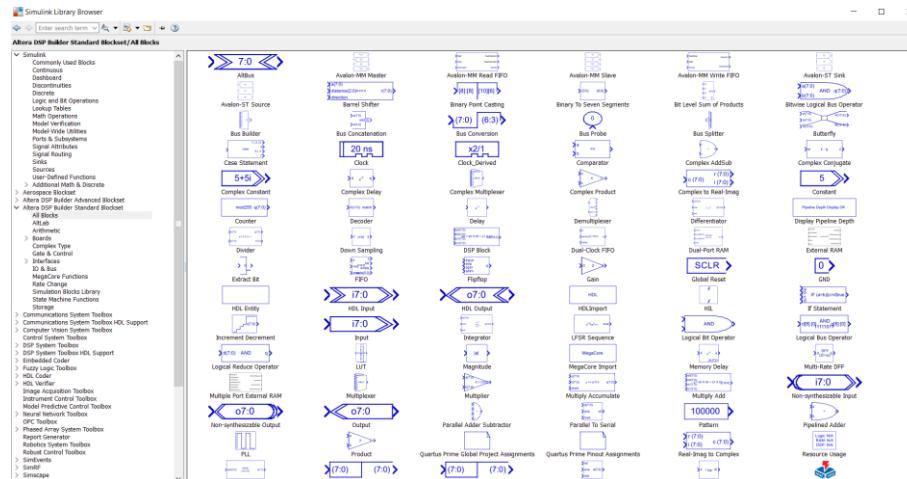
3.1.1 Công cụ Altera DSP Builder



Hình 3.1 DSP Builder overview

Công cụ Altera DSP Builder được tích hợp trên môi trường Simulink của MATLAB, hỗ trợ triển khai thiết kế các IP Core của FPGA theo mô hình sơ đồ khối (block-diagram). Trên MATLAB có thể tiến hành chạy mô phỏng kiểm thử, có thể tạo dữ liệu đầu vào và dữ liệu đầu ra để kiểm tra khả năng hoạt động của thiết kế. Ngoài ra còn có nhiều core sẵn có của Simulink hỗ trợ hiển thị kết quả mô phỏng như module Scope.

DSP builder cung cấp rất nhiều IP Core cơ bản cho FPGA như các bộ logic and, or, xor, not, các bộ mux, demux, delay, dịch bit, compare, ... đến các bộ phức tạp như các phép toán, Ram, Rom... như Hình 3.2.

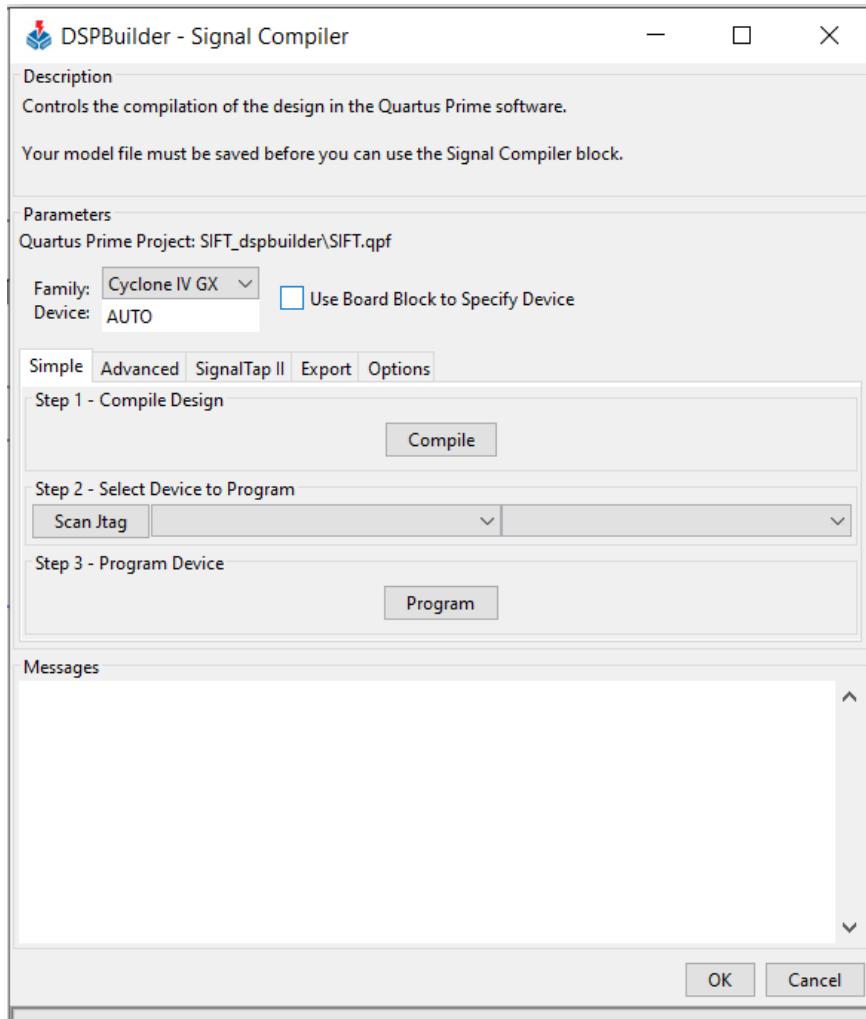


Hình 3.2 Block DSP Builder

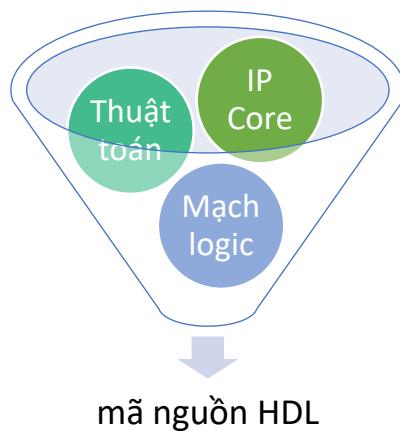
Nghiên cứu, thiết kế và triển khai thuật toán SIFT trên FPGA

Các IP Core cho phép cấu hình nhiều tham số để phù hợp với mục đích sử dụng. Có thể thêm các IP Core bằng cách xây dựng từ ngôn ngữ HDL vào để sử dụng trong thiết kế.

DSP Builder hỗ trợ generate ra mã nguồn VHDL hoặc SystemVerilog (Hình 3.3). Có thể sử dụng mã nguồn này để dịch và nạp lên Kit FPGA bằng công cụ Quartus.



Hình 3.3 DSP Builder – Signal Compiler

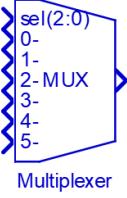
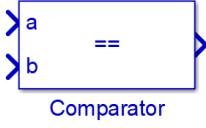
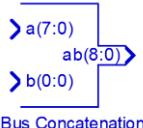
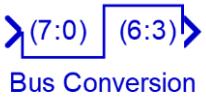
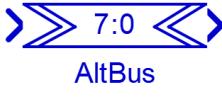
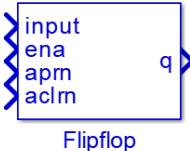


Hình 3.4 Sử dụng DSP Builder để hỗ trợ thiết kế phần cứng

3.1.2 Tổng quan một số IP Core trong Altera DSP Builder

Bảng 3.1 Một vài IP Core trong Altera DSP Builder sử dụng trong đồ án

Tên khối	Hình ảnh	Chức năng
Delay		Làm trễ tín hiệu đi qua nó. Tín hiệu ở đầu ra bằng tín hiệu ở đầu vào ở thời điểm (t-n) với t là thời điểm hiện tại và n là độ trễ cài đặt
Memory Delay		Chức năng giống khối Delay nhưng được sử dụng khi muốn tạo ra độ trễ lớn.
Counter		Mỗi chu kì clock bộ Counter sẽ tăng hoặc giảm đầu ra 1. Thông thường bộ Counter sẽ đếm từ 0 đến (n-1) (Với n do người dùng cài đặt). Quá trình này sẽ lặp đi lặp lại hoặc thay đổi theo các chân điều khiển người dùng thiết lập.
Parallel Adder Subtractor		Thực hiện các phép toán cộng và trừ. Tín hiệu ở đầu ra sẽ là kết quả thực hiện các phép toán của tín hiệu đầu vào với độ trễ là: $\text{ceil}(\log_2(\text{number of inputs}))$
Gain		Thực hiện phép nhân dữ liệu ở đầu vào với hằng số thiết lập trước. Tín hiệu ở đầu ra sẽ trễ so với đầu vào một lượng clock do người dùng cài đặt.
Multiplier		Thực hiện phép toán nhân. Kết quả ở đầu ra sẽ bằng tích của dữ liệu ở đầu vào và có số bit bằng tổng số bit của các tín hiệu ở đầu vào. Tín hiệu ở đầu ra sẽ trễ so với đầu vào một lượng clock do người dùng cài đặt.

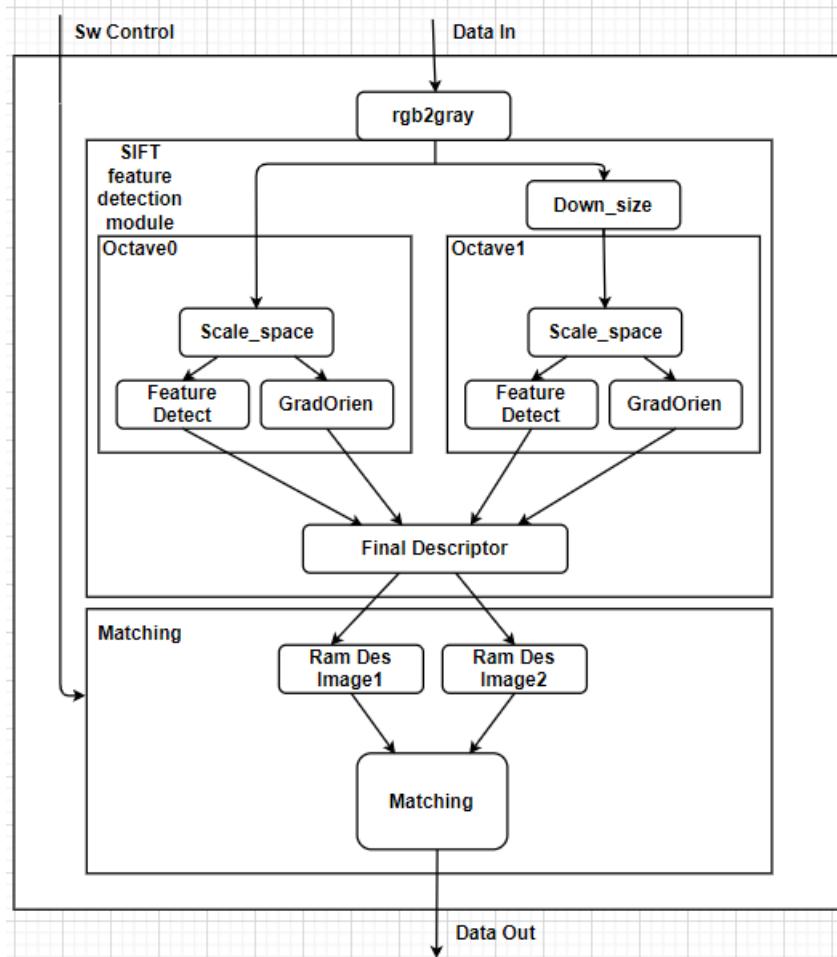
Multiplexer		Chức năng giống như một bộ chuyển mạch. Tín hiệu đầu ra sẽ bằng 1 trong những tín hiệu ở đầu vào được lựa chọn bằng chân sel. Tín hiệu ở đầu ra và đầu vào cùng ở một thời điểm.
Logical Bit Operator		Thực hiện các phép toán logic: AND, OR, XOR, NOT,.... Đầu vào là các tín hiệu có độ rộng 1 bit. Tín hiệu ở đầu ra và đầu vào cùng ở một thời điểm.
Logical Bus Operator		Thực hiện các phép toán: AND, OR, XOR, NOT, Shift Left, Shift Right, ... Tín hiệu đầu vào có thể cài đặt độ rộng bus theo nhu cầu. Tín hiệu đầu ra và tín hiệu đầu vào ở cùng một thời điểm.
Comparator		Thực hiện so sánh 2 dữ liệu ở đầu vào. Đầu ra là kết quả của phép so sánh được biểu diễn bằng 1 bit dữ liệu.
Bus Concatenation		Thực hiện ghép bit hai đầu vào tạo ra đầu ra mới có số bit bằng tổng số bit của 2 đầu vào.
Bus Conversion		Thực hiện trích chọn bit từ tín hiệu đầu vào theo tham số cài đặt. Tín hiệu đầu ra và đầu vào ở cùng một thời điểm.
AltBus		Thực hiện thay đổi kiểu dữ liệu tín hiệu đầu vào. Tín hiệu đầu ra và đầu vào ở cùng một thời điểm.
Flipflop		Chức năng là như một phần tử nhớ. Lưu lại tín hiệu ở input và dùng các chân điều khiển để chọn giá trị đầu ra q. Tín hiệu ở đầu ra sẽ trễ một clock so với tín hiệu đầu vào input.

Constant		Tạo ra hằng số để sử dụng trong thiết kế.
Pattern		Lặp đi lặp lại xuất từng bit dữ liệu từ trái qua phải ra đầu ra theo từng clock. Được sử dụng trong phần kiểm thử tạo dữ liệu đầu vào cho các IP Core nhỏ trên MATLAB.
Time Scope		Hiển thị giá trị tín hiệu ở đầu vào. Được sử dụng trong phần kiểm thử, kiểm tra hoạt động của IP Core khi thiết kế.
From - Goto		Tín hiệu dữ liệu được gửi Goto và có thể lấy ra từ nhãn tương ứng từ From.
To Workspace		Tín hiệu dữ liệu đi đến đầu vào của khối sẽ được đưa ra không gian Workspace của MATLAB để có thể kiểm tra hoạt động của thiết kế.
From Workspace		Có thể tạo dữ liệu từ Workspace của MATLAB để đưa vào IP Core để kiểm thử hoạt động của thiết kế.
Input		Định nghĩa đầu vào trên phần cứng của IP Core.
Output		Định nghĩa đầu ra trên phần cứng của IP Core.
In - Out		Cung cấp Input Port và Output Port cho các hệ thống con (Subsystem).

Dual-Port RAM		<p>Thực hiện lưu trữ và truy xuất dữ liệu đầu vào. Cho phép đọc ghi dữ liệu bằng cách đưa các địa chỉ vào chân rd_add (địa chỉ đọc dữ liệu) và chân wr_add (địa chỉ ghi dữ liệu) tương ứng.</p> <p>Tín hiệu đọc ra có độ trễ 2 clock kể từ khi đưa địa chỉ vào chân rd_add.</p>
Signal Compiler		<p>Thực hiện generate ra mã nguồn VHDL hoặc SystemVerilog. Tạo IP core ghép nối vào trong Qsys trên công cụ Quartus rồi nạp lên kit.</p>
Clock		<p>Sử dụng ở top level of a design để thiết lập clock chính cơ sở cho phần cứng.</p>
Avalon-ST Sink		<p>Interface hỗ trợ kết nối với các IP Core truyền nhận dữ liệu ảnh theo chuẩn Avalon Streaming.</p>
Avalon-ST Source		<p>Interface hỗ trợ kết nối với các IP Core truyền nhận dữ liệu ảnh theo chuẩn Avalon Streaming.</p>

3.2 Tổng quan thiết kế

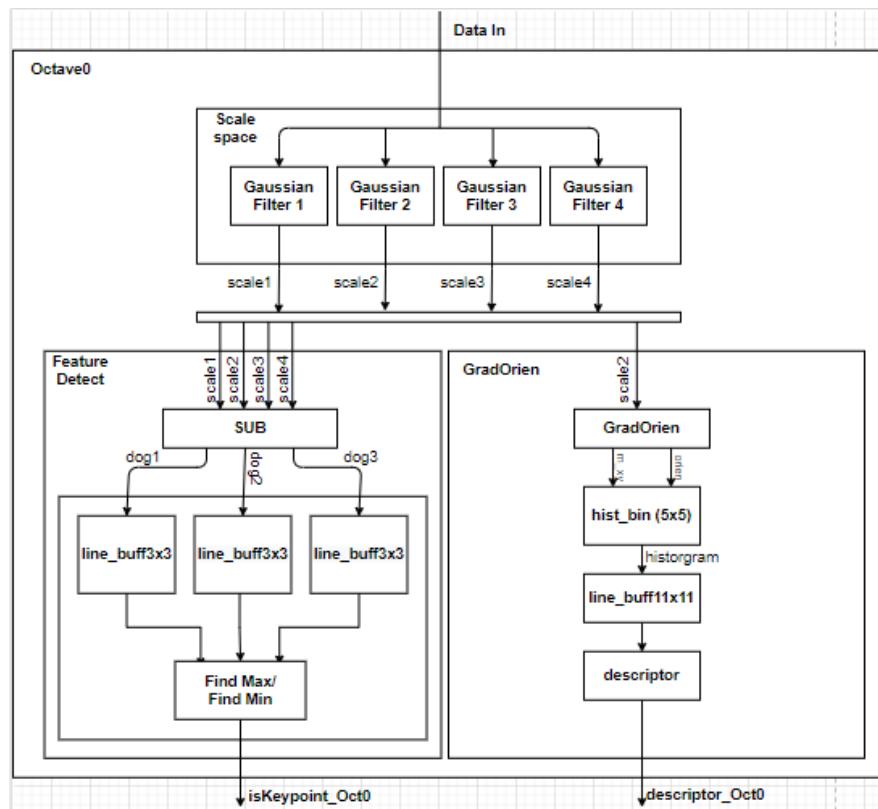
Do thuật toán SIFT có khối lượng công việc tính toán rất lớn và phức tạp, cùng với sự giới hạn về tài nguyên của phần cứng và kém linh hoạt trong các phép toán tác động lên nhiều điểm ảnh cùng một lúc. Bởi vậy trong đồ án, thuật toán SIFT đã được tinh chỉnh một số bước, giới hạn số Octave để tìm kiếm điểm hấp dẫn trên ảnh là 2, số lượng scale trên mỗi Octave là 4 và chỉ xử lý trên ảnh đầu vào có kích thước 640*480 pixel để đạt được hiệu suất tốt nhất cân bằng giữa phần cứng và phần mềm (ý tưởng thực hiện dựa theo [3]).



Hình 3.5 Sơ đồ khái quát tổng quan thiết kế theo chiều dữ liệu

Hình 3.5 mô tả tổng quan thiết kế, dữ liệu Data In là dữ liệu ảnh (kích thước ảnh cố định là 640*480 pixel) RGB 24-bit truyền theo luồng nối tiếp từng pixel (mỗi pixel có 3 kênh màu, mỗi kênh màu có kích thước 8-bit và được truyền song song) theo chuẩn Avalon Streaming (trình bày trong mục 1.4.2). Trước khi đi vào module **SIFT detection feature** dữ liệu ảnh sẽ đi qua khối **rgb2gray** chuyển lần lượt từng pixel ảnh RGB (24-bit) thành đa mức xám (8-bit). Ảnh đa mức xám sau khối **rgb2gray** sẽ được truyền nối tiếp vào khối **SIFT detection feature** chia làm 2 luồng truyền song song với nhau đi qua hai khối nhỏ **Octave0** và **Octave1** để thực hiện tính toán tìm kiếm phát hiện những điểm hấp dẫn (keypoint) có tính bất biến cao (không bị ảnh hưởng bởi các phép xoay, phép dịch, phép zoom, thay đổi độ sáng hay nhiễu trong ảnh) và tạo bộ mô tả

(descriptor) cho những điểm đó. **Octave0** sẽ thực hiện tính toán trên ảnh gốc đa mức xám có độ phân giải là 640*480 pixel, **Octave1** thực hiện tính toán trên ảnh đa mức xám có được bằng cách giảm kích thước ảnh ban đầu đi một nửa (kỹ thuật downsampling) sử dụng khối **Down_size** (thực hiện giảm kích thước ảnh từ 640*480 pixel về 320*240 pixel). Kết quả tính toán của 2 khối **Octave0** và **Octave1** là 2 tín hiệu **isKeypoint** (cho biết pixel trong ảnh tương ứng có phải điểm hấp dẫn không) và descriptor (bộ mô tả của các pixel trong ảnh) được tổng hợp lại, loại bỏ trùng lặp bằng khối **Final Descriptor** để tạo dữ liệu đầu vào cho khối **Matching**. Module **Matching** được xây dựng để thực hiện đối sánh giữa 2 ảnh. Cho từng ảnh đi qua module **SIFT detection feature** sử dụng SW control để điều khiển cho phép lưu tập descriptor của từng ảnh vào vùng Ram tương ứng. Sau đó đọc các descriptor đó ra rồi thực hiện matching với nhau.



Hình 3.6 Sơ đồ tổng quan thiết kế khối Octave0 theo chiều dữ liệu

Hình 3.6 mô tả tổng quan khói **Octave0** (Khối **Octave1** tương tự thiết kế với khói **Octave0**) đây là một khói rất quan trọng trong module **SIFT detection feature**. Khối **Octave0** nhận dữ liệu ảnh kích thước 640*480 pixel đa mức xám truyền nối tiếp từng điểm ảnh để xử lý tạo ra 4 scale bằng cách sử dụng 4 mặt nạ Gaussian có kích thước cố định 5x5 và tham số σ ở scale 1 là 1.1 và các tham số σ bằng 1.3, 1.6, 2.0 được sử dụng cho lần lượt 3 scale còn lại. Dữ liệu 4 scale được đưa vào khói **Feature Detect** để thực hiện tìm kiếm những điểm hấp dẫn trong Octave0. Ở bước này để giảm độ phức tạp của thuật toán ta thực hiện giảm độ chính xác của dữ liệu trong không gian DoG để loại bỏ những điểm nằm trên biên cạnh hoặc có độ tương phản thấp. Những cực đại hoặc cực tiểu địa phương tìm được trong không gian DoG sẽ được sử dụng làm điểm hấp dẫn. Thực hiện

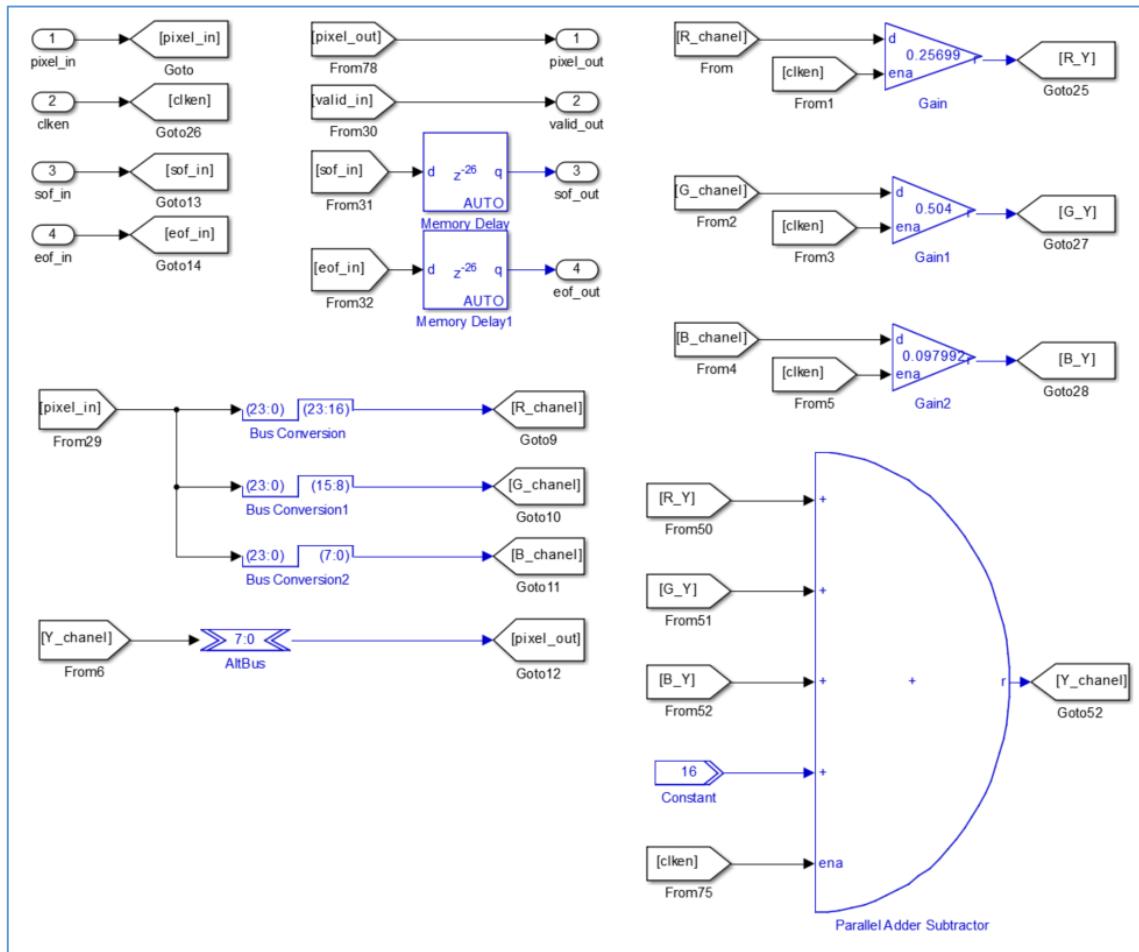
song song với khối **Feature Detect**, dữ liệu từ scale2 được đưa vào khối **GradOrien** để tính toán tạo descriptor cho từng điểm ảnh. Khối **GradOrien** thay vì tìm ra một descriptor là 1 vector 128 ($8 \times 4 \times 4$) chiều thì để giảm khối lượng tính toán chúng ta giảm xuống còn 72 chiều ($8 \times 3 \times 3$). Ý tưởng lý thuyết được trình bày trong [3].

3.3 Thiết kế và kết quả mô phỏng IP Core rgb2gray

3.3.1 Thiết kế

Để tạo ra không gian Scale-space ta sử dụng dữ liệu ảnh đầu vào là ảnh đa mức xám thay vì ảnh RGB. IP Core **rgb2gray** sẽ thực hiện chuyển ảnh từ RGB (1 pixel có độ rộng 24-bit) sang ảnh đa mức xám (1 pixel có độ rộng 8-bit) theo công thức:

$$gray = 0.257 * red + 0.504 * green + 0.098 * blue + 16$$



Hình 3.7 Thiết kế khói **rgb2gray**

3.3.2 Mô phỏng trên matlab

Kết quả mô phỏng (Hình 3.8)

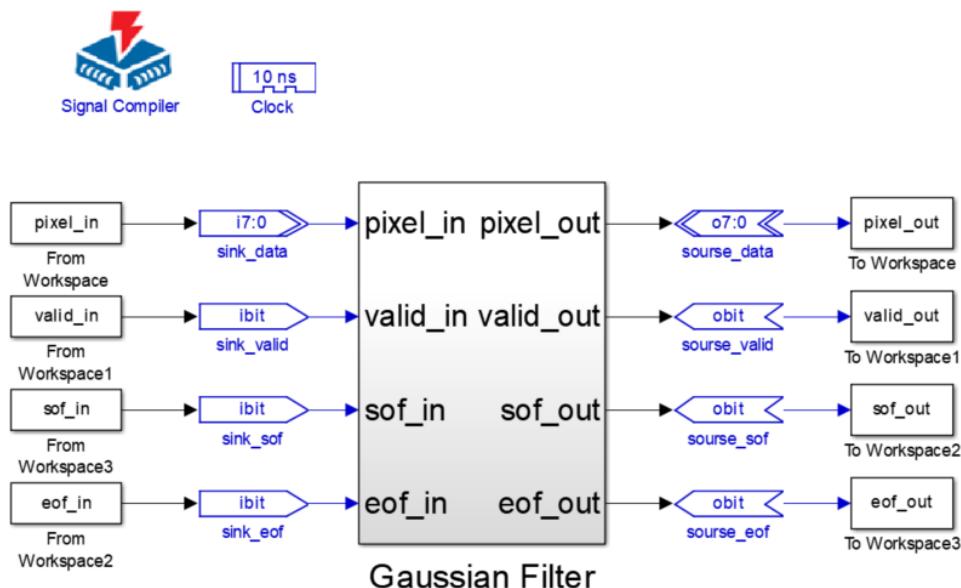


Hình 3.8 Ảnh sau khi đi qua khối *rgb2gray*

3.4 Thiết kế và mô phỏng IP Core Gaussian Filter

3.4.1 Thiết kế

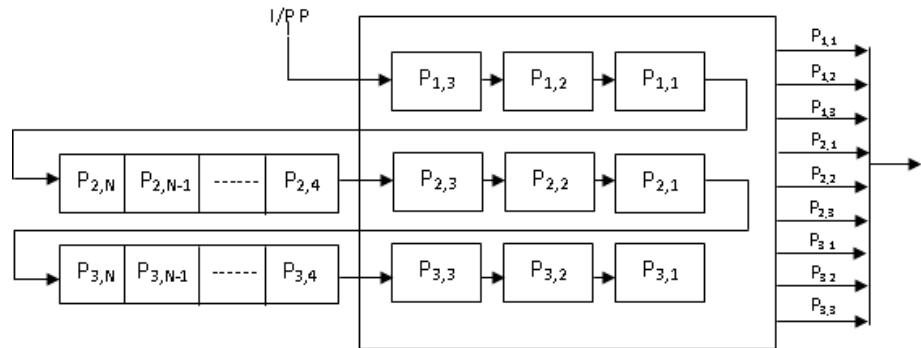
- IP Core với đầu vào là Avalon Video Streaming, nhận vào lần lượt từng điểm ảnh của mỗi frame theo thứ tự từ trái sang phải, từ trên xuống dưới.
- IP Core Gaussian Filter bản chất là phép toán nhân chập (convolution) ảnh đầu vào với mặt nạ Gaussian được sử dụng rất nhiều trong khối SIFT có vài trò quan trọng để tạo lên khôn gian Scale-space.



Hình 3.9 Sơ đồ mức top của khối Gaussian Filter

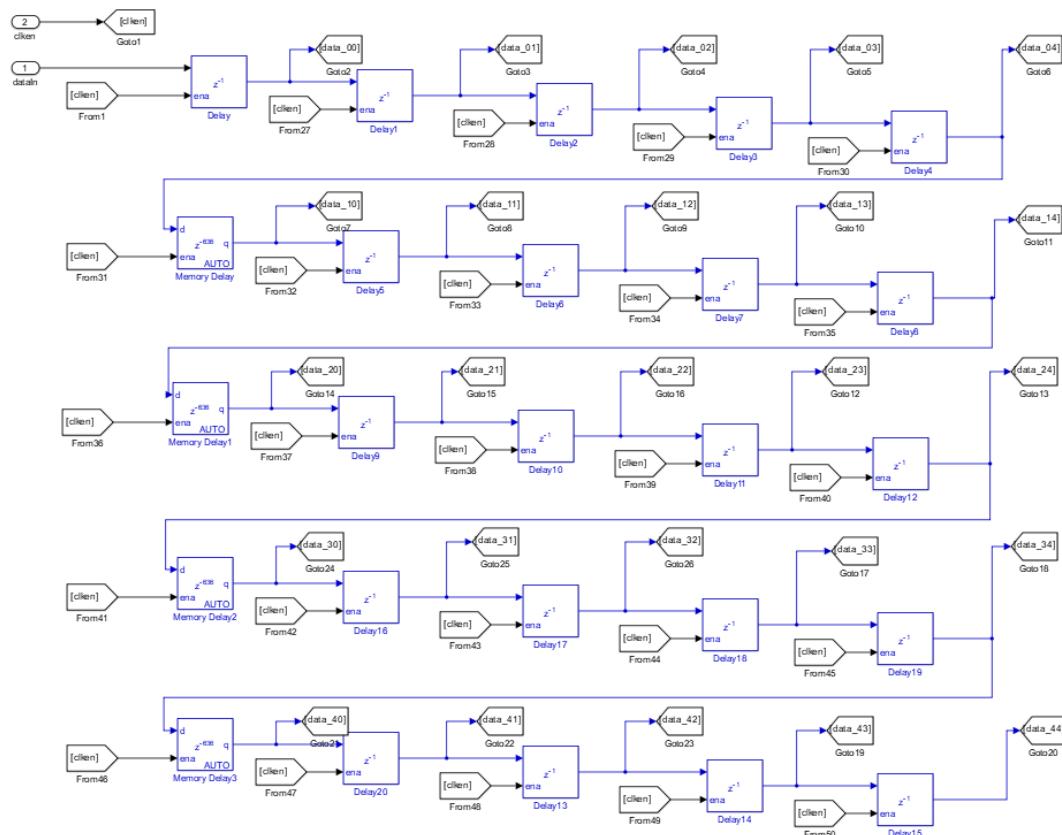
Khối gaussian scale thiết kế gồm 2 khối **line_buff5x5** và **convolution**.

Khối **line_buff5x5** có chức năng đệm lại dữ liệu ảnh (thực hiện lưu trữ thông tin để nhân chập).



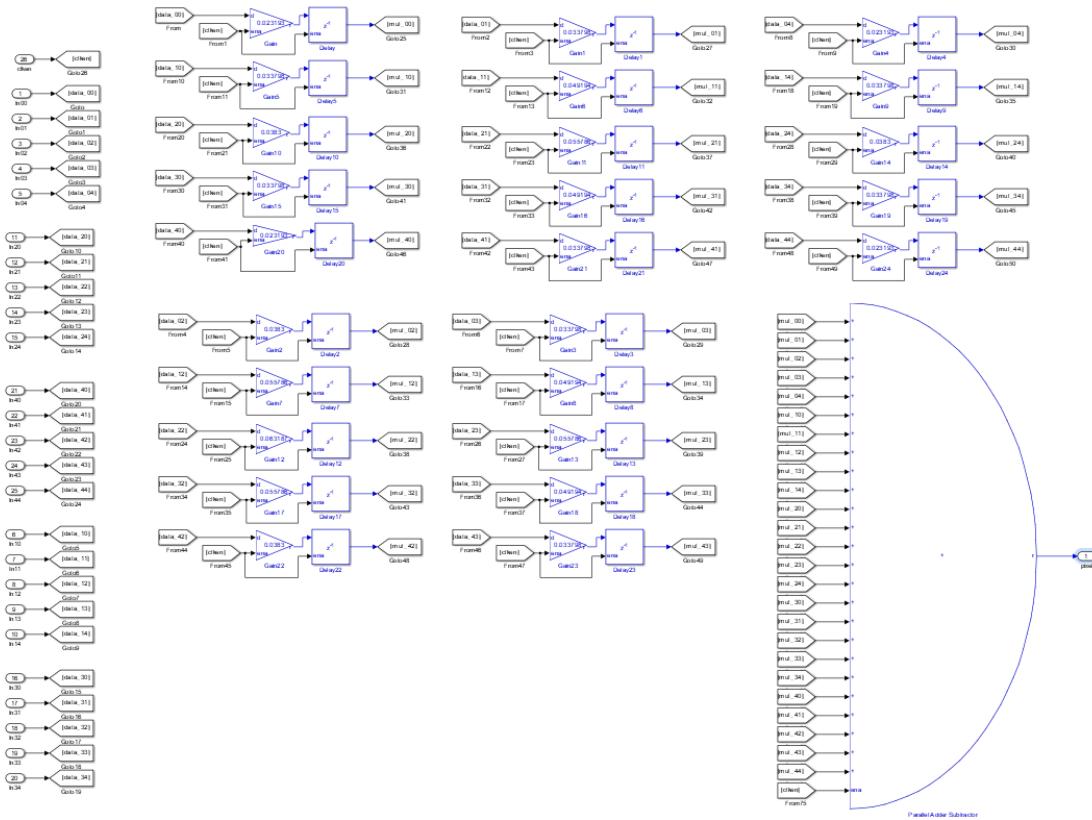
Hình 3.10 Mô phỏng line buffer 3x3

Trong thiết kế sử dụng các khối Delay/Memory Delay để thực hiện lưu trữ lại từng pixel, dùng tín hiệu valid trong chuẩn Avalon Streaming để cho phép các khối Delay/Memory Delay hoạt động. Giá trị của các pixel sẽ dịch chuyển từ trái qua phải và từ trên xuống dưới tại sườn lên (Rising Edge Trigger) của mỗi clock (thiết kế như Hình 3.10).



Hình 3.11 Thiết kế khối line buff5x5

Khối **convolution** tính toán nhân chập ảnh với mặt nạ gaussian 5x5 (thiết kế như Hình 3.12). Output của khối **convolution** được sử dụng để tính descriptor và đưa vào khối **detect_keypoint** để tìm kiếm các điểm hấp dẫn.

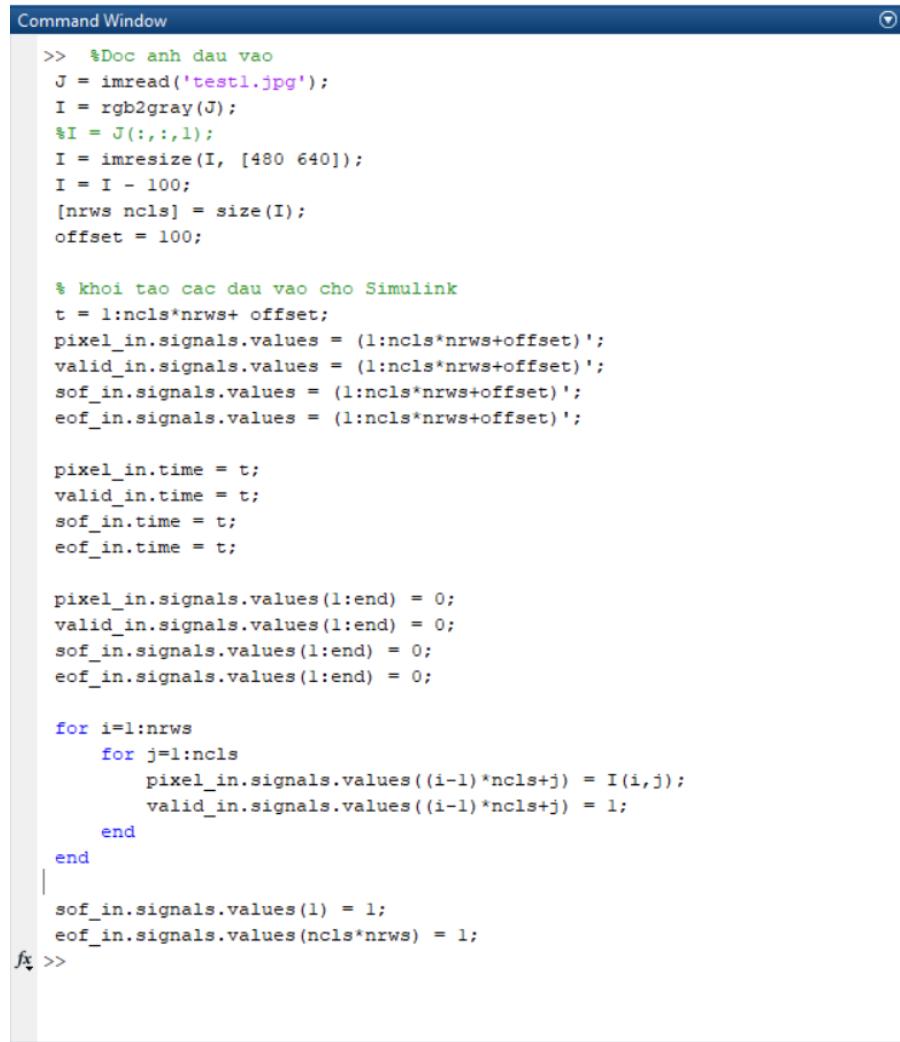


Hình 3.12 Thiết kế khối convolution

3.4.2 Thực hiện mô phỏng

- DSP Builder cho phép thực hiện chạy mô phỏng thiết kế trên MATLAB/Simulink, ta chỉ cần tạo dữ liệu đầu vào tương ứng với dữ liệu thực tế trong không gian Workspace của MATLAB, chạy mô phỏng dữ liệu đó, có thể lấy kết quả đầu ra để kiểm tra qua các công cụ của MATLAB hỗ trợ như Scope hay hiển thị hình ảnh, vẽ biểu đồ trực quan.
- Trong quá trình thiết kế xây dựng project ta nên thực hiện chạy mô phỏng từng module/ IP Core nhỏ rồi đến các IP Core lớn. Thực hiện chạy hoàn chỉnh từng IP Core trên MATLAB thì mới nạp chạy thiết kế IP Core lên kit. Việc kiểm thử từng module nhỏ sẽ giúp ta hạn chế sai sót và có thể nắm chắc hoạt động của từng module. Khi thiết kế càng lớn thì thời gian chạy mô phỏng sẽ lâu hơn và khó debug lỗi hơn.
- Trong thiết kế thuật toán SIFT có rất nhiều khối nhỏ lồng nhau, bởi vậy việc chạy mô phỏng là rất quan trọng, giúp thiết kế chính xác với thực tế trên kit. Sau đây là các bước kiểm thử một số khối trong thiết kế thuật toán.

Đầu tiên ta cần thực hiện tạo dữ liệu (môi trường mô phỏng) đầu vào cho thiết kế:



```

Command Window
>> %Doc anh dau vao
J = imread('test1.jpg');
I = rgb2gray(J);
%I = J(:,:,1);
I = imresize(I, [480 640]);
I = I - 100;
[nrws ncls] = size(I);
offset = 100;

% khai tao cac dau vao cho Simulink
t = 1:ncls*nrws+ offset;
pixel_in.signals.values = (1:ncls*nrws+ offset)';
valid_in.signals.values = (1:ncls*nrws+ offset)';
sof_in.signals.values = (1:ncls*nrws+ offset)';
eof_in.signals.values = (1:ncls*nrws+ offset)';

pixel_in.time = t;
valid_in.time = t;
sof_in.time = t;
eof_in.time = t;

pixel_in.signals.values(1:end) = 0;
valid_in.signals.values(1:end) = 0;
sof_in.signals.values(1:end) = 0;
eof_in.signals.values(1:end) = 0;

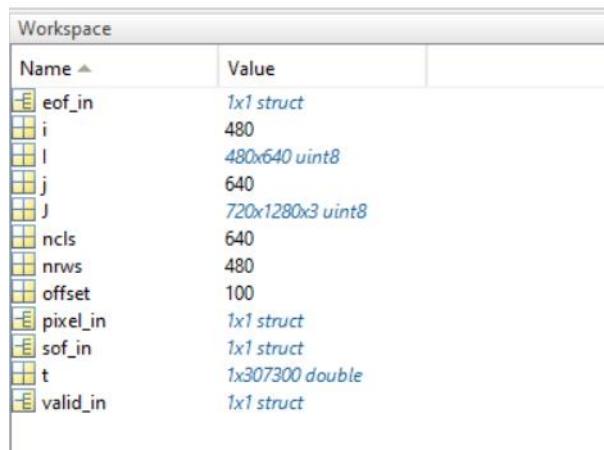
for i=1:nrws
    for j=1:ncls
        pixel_in.signals.values((i-1)*ncls+j) = I(i,j);
        valid_in.signals.values((i-1)*ncls+j) = 1;
    end
end
|
sof_in.signals.values(1) = 1;
eof_in.signals.values(ncls*nrws) = 1;
fx >>

```

Hình 3.13 Tạo file script sink dữ liệu đầu vào Simulink

Tạo một file script như Hình 3.13, chuẩn bị một ảnh ‘test1.jpg’ đặt cùng thư mục chứa project.

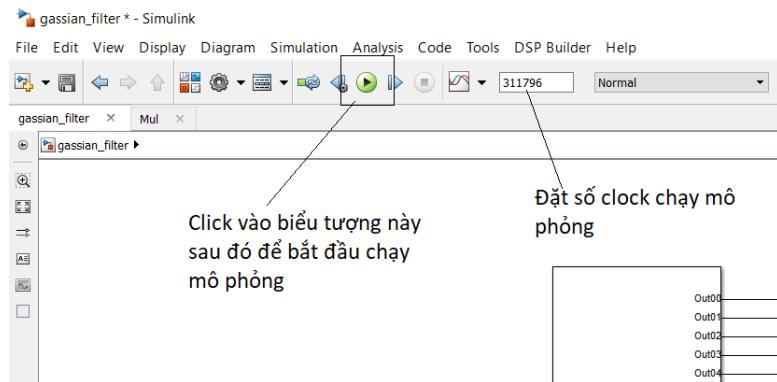
Chạy file script trên cửa sổ Command Window của MATLAB ta được dữ liệu đầu vào trên Workspace như Hình 3.14.



Name	Value
eof_in	1x1 struct
i	480
I	480x640 uint8
j	640
J	720x1280x3 uint8
ncls	640
nrws	480
offset	100
pixel_in	1x1 struct
sof_in	1x1 struct
t	1x307300 double
valid_in	1x1 struct

Hình 3.14 Không gian Workspace trên MATLAB

Trên Simulink của thiết kế đặt tham số số clock chạy mô phỏng và nhấn vào biểu tượng để tiến hành chạy mô phỏng thiết kế (Hình 3.15).



Hình 3.15 Tiến hành chạy mô phỏng thiết kế

Để kiểm tra kết quả mô phỏng ta có thể chạy đoạn script sau để hiện thị kết quả dữ liệu là ảnh ở đầu ra (Hình 3.16):

```
>> %Hiển thị ảnh ban đầu
figure, imshow(uint8(I));
%Hiển thị ảnh sau khi qua khơi gaussian_filter
image = pixel_out.data(1304: 1303 + 480*640);
img_out = round(reshape(image, 640, 480)');
figure, imshow(uint8(img_out));
>>
```

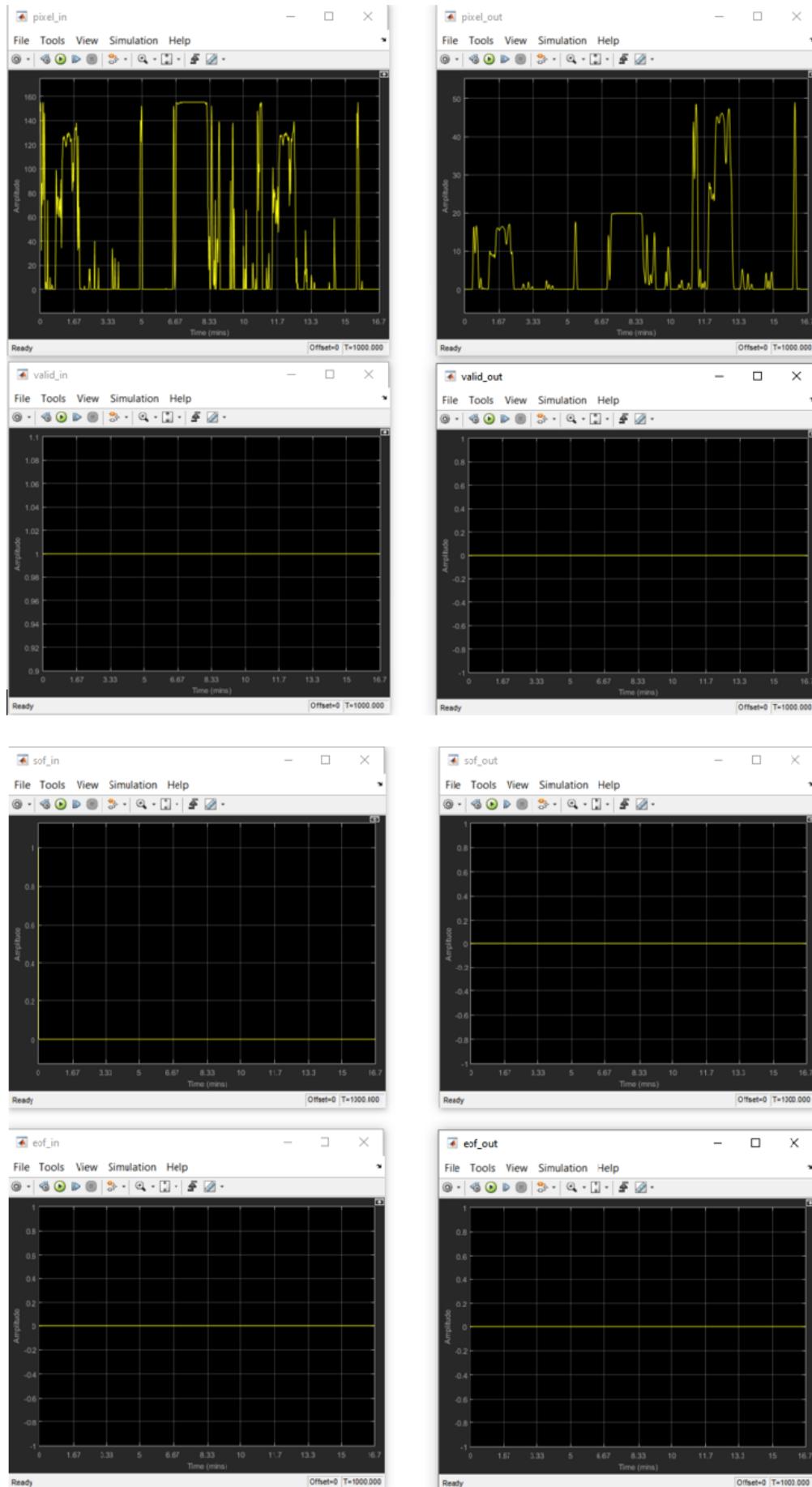
Hình 3.16 Hiển thị kết quả ảnh sau khói Gaussian filter

Kết quả mô phỏng khói **Gaussian_filter** (Hình 3.17)



Hình 3.17 Ảnh nhận được khi qua các bộ lọc Gaussian filter

Nghiên cứu, thiết kế và triển khai thuật toán SIFT trên FPGA

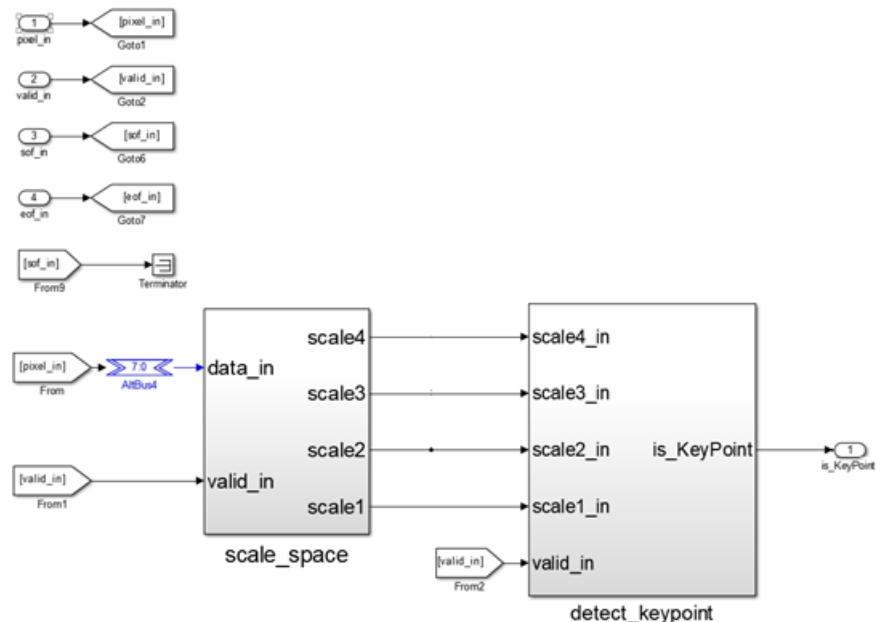


Hình 3.18 Theo dõi giá trị các tín hiệu qua cửa sổ Scope

Dánh giá kết quả mô phỏng:

- Dựa trên hình ảnh Hình 3.17 các ảnh đi qua khối Gaussian đã có hiệu ứng bị mờ hơn so với ảnh gốc và khi ta tăng dần tham số σ ảnh có hiệu ứng mờ dần. Kết quả sát với lý thuyết được trình bày ở phần trên.
- Dựa vào các giá trị của các tín hiệu hiển thị qua cửa sổ scope ta có thể quan sát kiểm soát dữ liệu khi đi qua IP Core, từ đó có thể debug thiết kế.

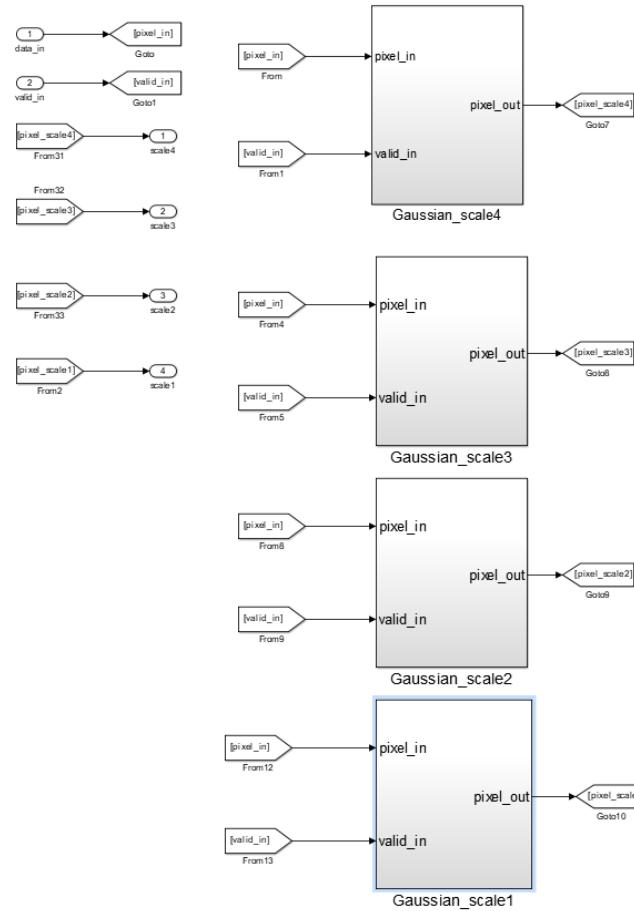
3.5 Thiết kế và mô phỏng IP Core SIFT Detector



Hình 3.19 Top level of a design SIFT Detector

Hình 3.19 mô tả thiết kế Top level of a design của khối SIFT Detector. Dữ liệu ảnh được đưa lần lượt từng pixel vào khối **scale_space** để tạo ra không gian ảnh gồm 4 scale (Hình 3.17). Đầu ra 4 scale sau khối **scale_space** sẽ được đi vào khối **detect_keypoint** để tìm kiếm phát hiện điểm hấp dẫn.

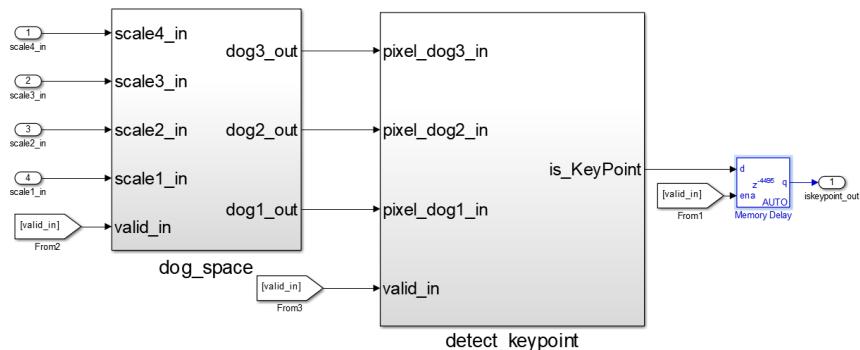
3.5.1 Thiết kế khối scale_space



Hình 3.20 Thiết kế scale space

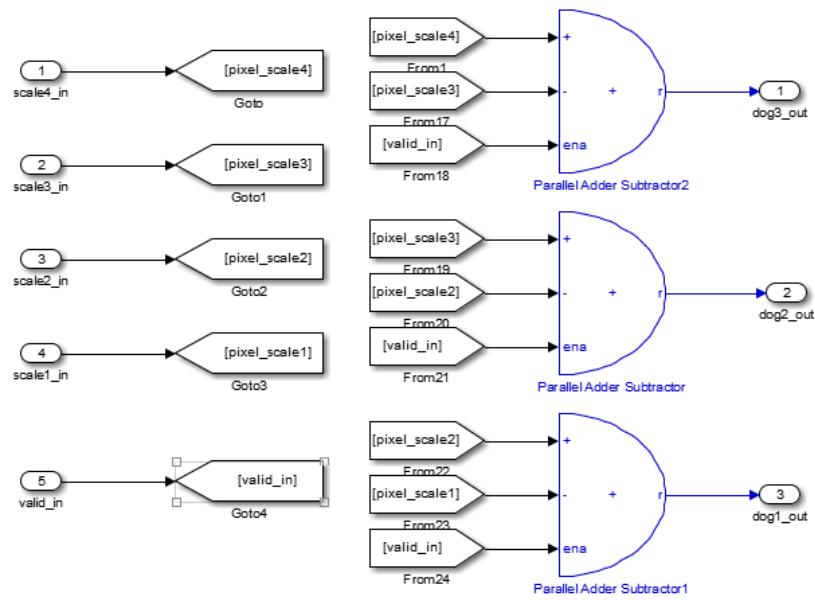
Khối **scale space** được thiết kế gồm 4 khối **gaussian_scale** (3.4) thực hiện nhân chập dữ liệu ảnh với mặt nạ gaussian 5×5 có các tham số σ tăng dần từ scale1 đến scale 4.

3.5.2 Thiết kế khối detect_keypoint

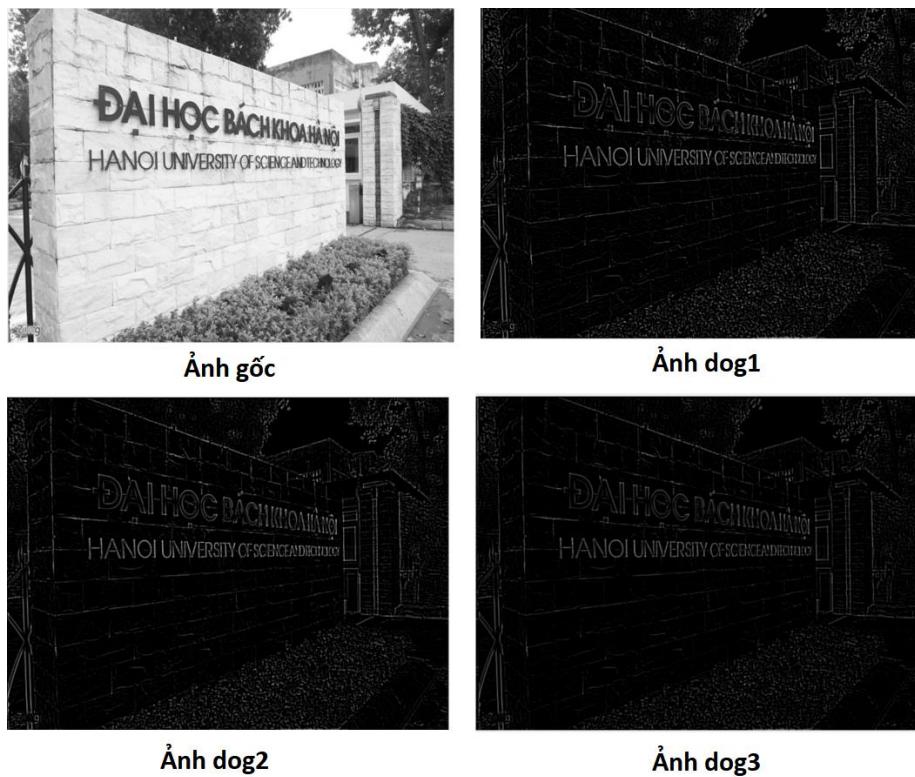


Hình 3.21 Thiết kế khối detect_keypoint

Khối **dog_space** thực hiện trừ lần lượt các scale trên scale dưới, 4 scale sẽ có 3 dog (difference of gaussian) (thiết kế như Hình 3.22).



Hình 3.22 Thiết kế khối dog_space

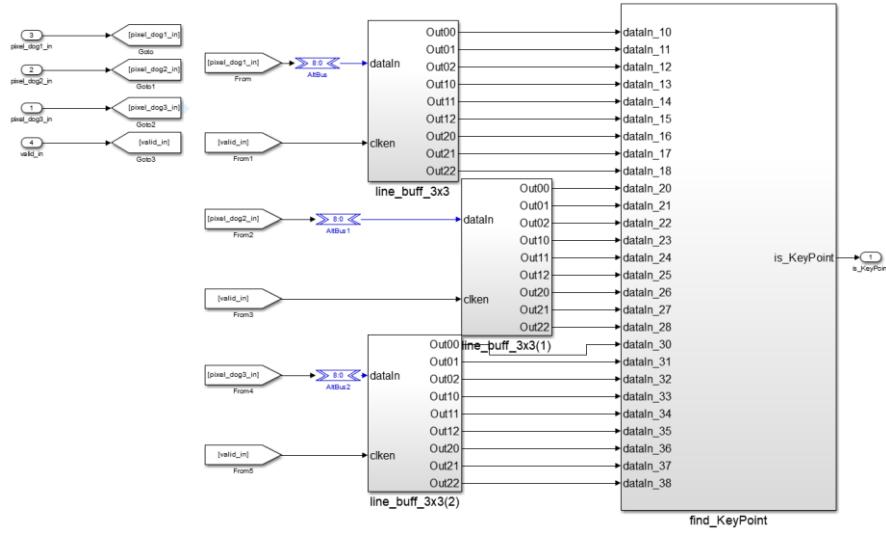


Hình 3.23 Kết quả mô phỏng ảnh trong không gian dog_space

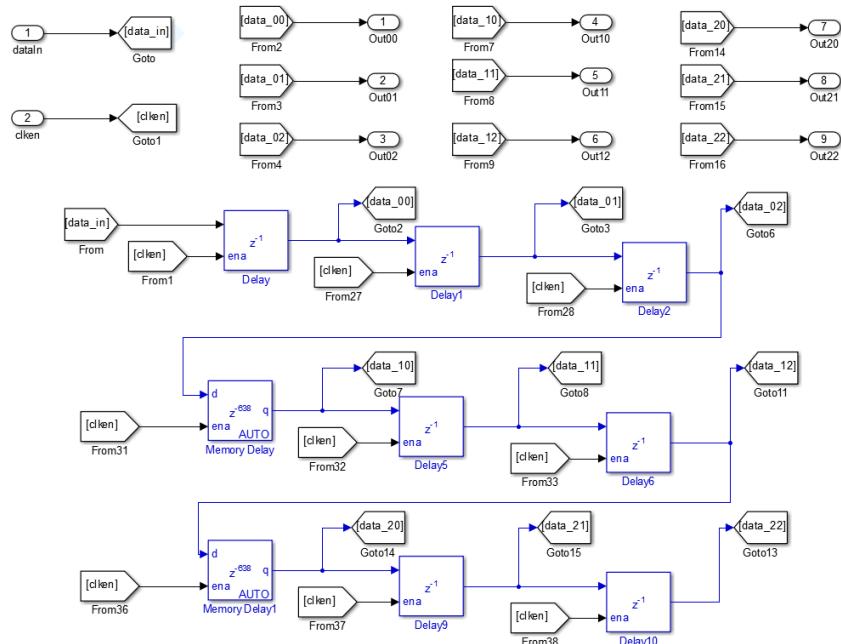
Hình 3.23 là kết quả mô phỏng tính DoG của ảnh thu được các ảnh dog1 đến dog3 từ 4 scale. Ảnh DoG chứa viền của các đối tượng trong ảnh.

Khối detect_keypoint: tìm cực trị địa phương trên 3 miền dog_space (2.2.1) có thiết kế như các hình sau:

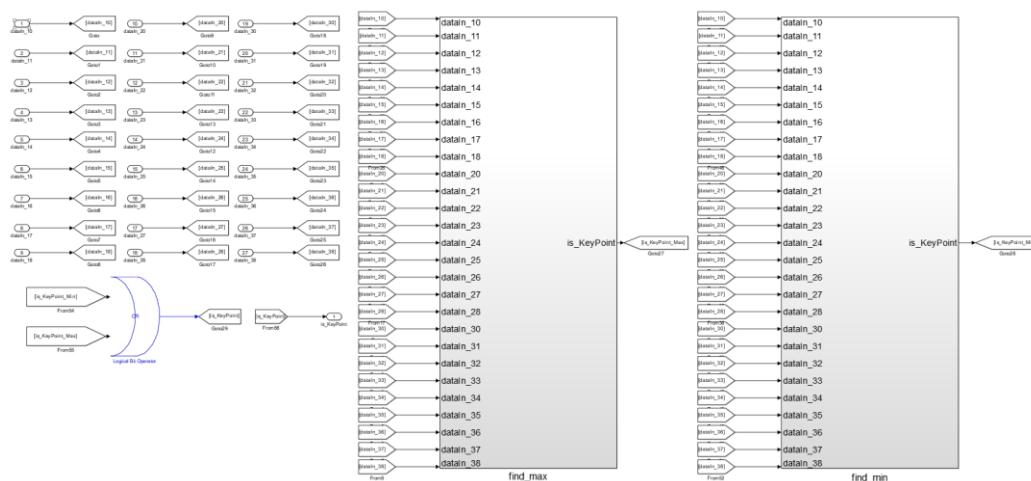
Nghiên cứu, thiết kế và triển khai thuật toán SIFT trên FPGA



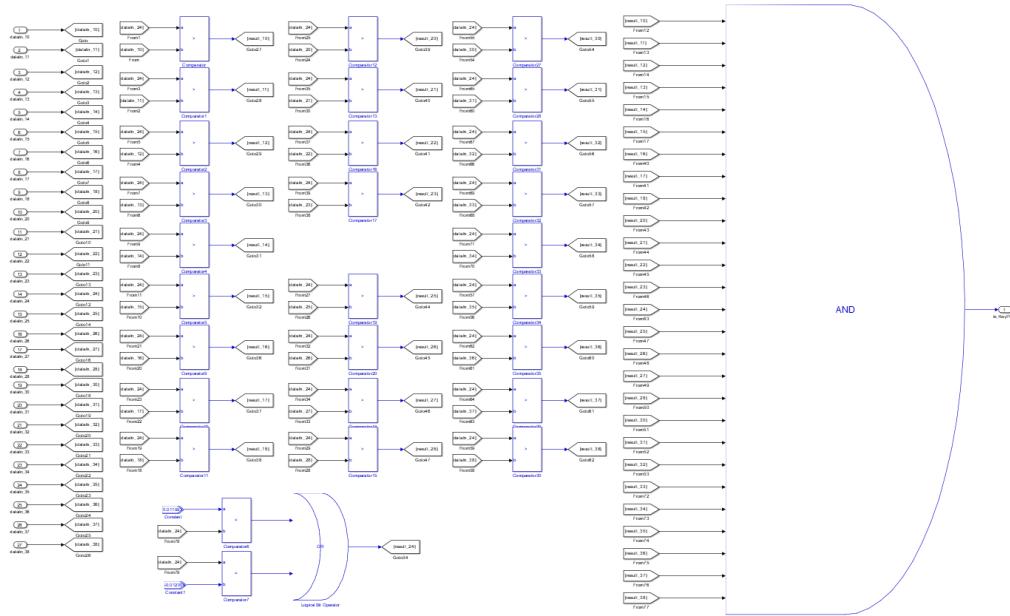
Hình 3.24 Thiết kế khói detect_keypoint



Hình 3.25 Khối line_buf3x3 cho ảnh 640*480



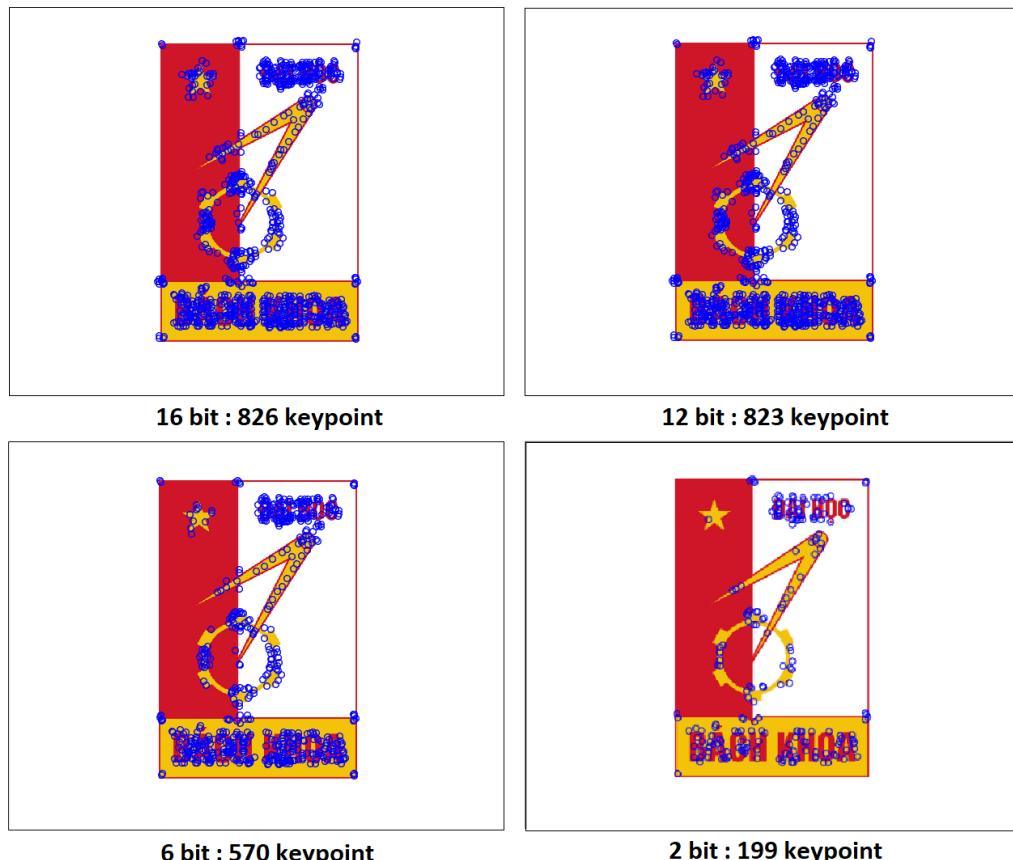
Hình 3.26 Thiết kế khói find_keypoint



Hình 3.27 Thiết kế khói *find_max*

Hình 3.27 thiết kế khói **find_max** tìm cực đại địa phương trên 3 miền dog (difference of gaussian). Mỗi điểm trong miền dog2 sẽ được so sánh với 8 điểm xung quanh nó và 9 điểm ở miền dog3 và 9 điểm ở miền dog1. Khối **find_min** có thiết kế tương tự.

Kết quả mô phỏng khói detect_keypoint trên MATLAB:



Hình 3.28 Ảnh hưởng khi giảm độ chính xác không gian dog_space đến kết quả tìm kiếm keypoint

Hình 3.28 là kết quả mô phỏng trên MATLAB khi chạy thiết kế **detect_keypoint** có thay đổi độ chính xác của không gian dog_space bằng cách giảm số lượng bit phần thập phân để biểu diễn các giá trị trong không gian. Ta thấy rằng số lượng điểm hấp dẫn tăng dần khi chúng ta tăng số lượng bit biểu diễn phần thập phân và khi giảm số lượng bit biểu diễn thì số lượng điểm hấp dẫn cũng giảm và khá sát với các đối tượng trong ảnh. Từ đó bằng thực nghiệm ta có thể thấy việc giảm độ chính xác trong không gian dog_space (cụ thể giảm số lượng bit biểu diễn phần thập phân của dữ liệu) có ảnh hưởng tích cực đến việc giảm các điểm nhiễu, điểm không có tiềm năng làm điểm hấp dẫn

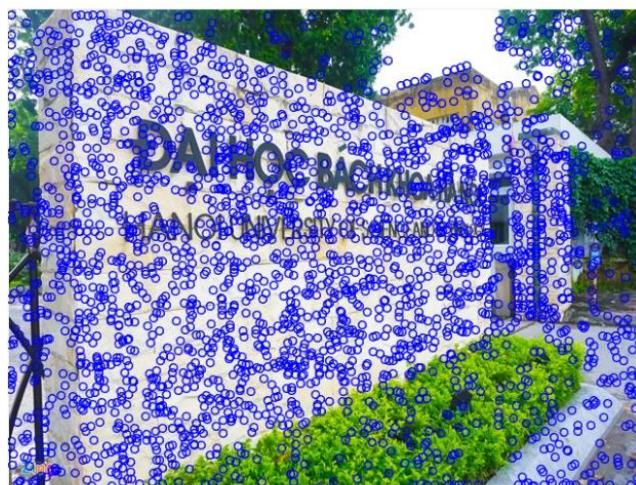


Hình 3.29 Kết quả mô phỏng detect keypoint trên ảnh logo BKHN

Hình 3.29 miêu tả các điểm hấp dẫn được tìm kiếm và phát hiện ra trên Octave0, Octave1 và khi sử dụng cả 2 Octave. Các điểm hấp dẫn tập chung chủ yếu ở các vị trí góc cạnh của đối tượng trong ảnh, những vị trí nằm trên nền của ảnh thì không xuất hiện điểm hấp dẫn. Khi tăng số lượng Octave thì số lượng điểm hấp dẫn trên ảnh sẽ nhiều hơn.



Ảnh gốc

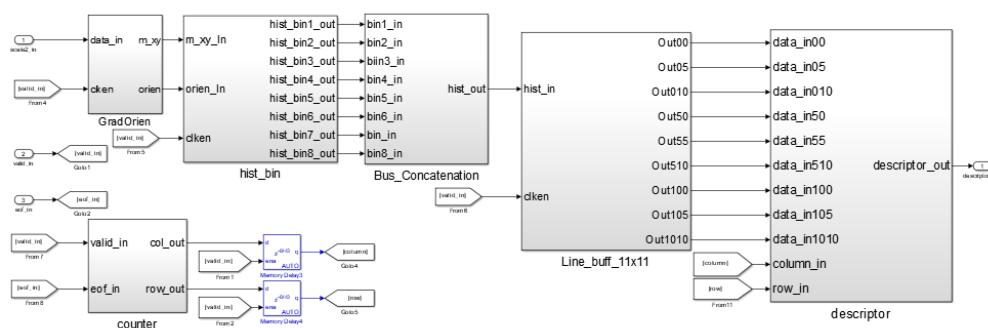


Ảnh keypoint được phát hiện: 2665 keypoint

Hình 3.30 Mô phỏng detect keypoint trên ảnh

Số lượng điểm hấp dẫn sẽ tỷ lệ thuận với số lượng đối tượng trong ảnh. Ảnh chứa càng nhiều đối tượng thì có càng nhiều điểm hấp dẫn (Hình 3.30)

3.6 Thiết kế và mô phỏng IP Core calculate_descriptor



Hình 3.31 Sơ đồ thiết kế khối calculate_descriptor

Khối **calculate_descriptor** thực hiện tính toán descriptor cho từng điểm ảnh gồm các khối:

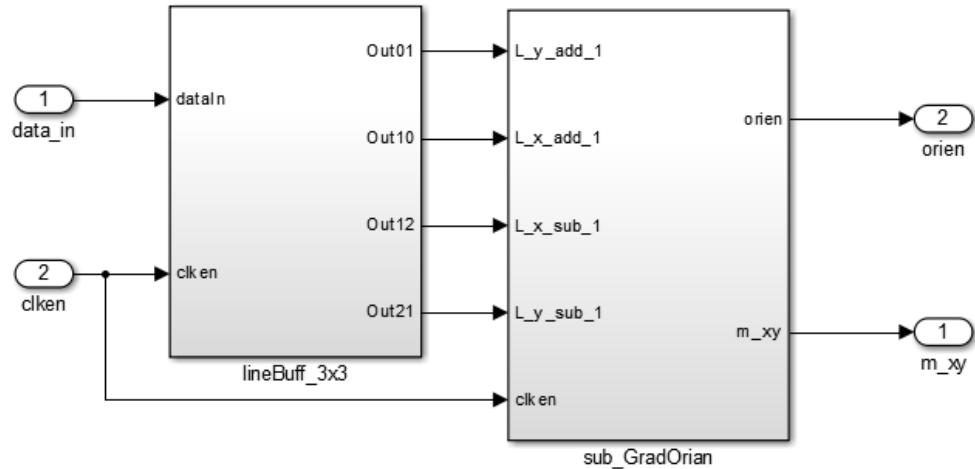
- Khối **GradOrien** tính toán độ lớn và hướng biến đổi của pixel trong ảnh theo công thức:

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

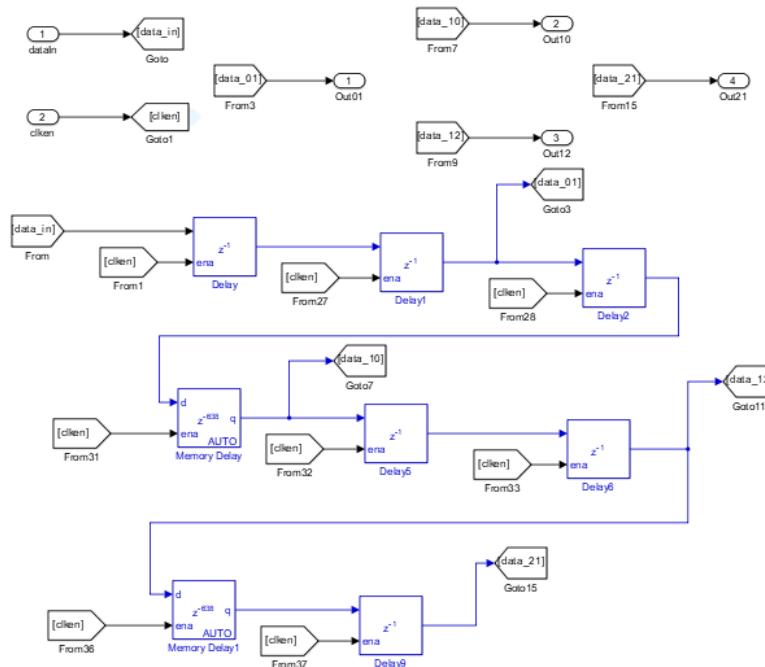
$$\theta(x, y) = \tan^{-1}((L(x, y+1) - L(x, y-1)) / (L(x+1, y) - L(x-1, y)))$$

Để giảm thiểu khối lượng tính toán và sử dụng tài nguyên phần cứng ít hơn ta thực hiện tính toán với công thức xấp xỉ sau:

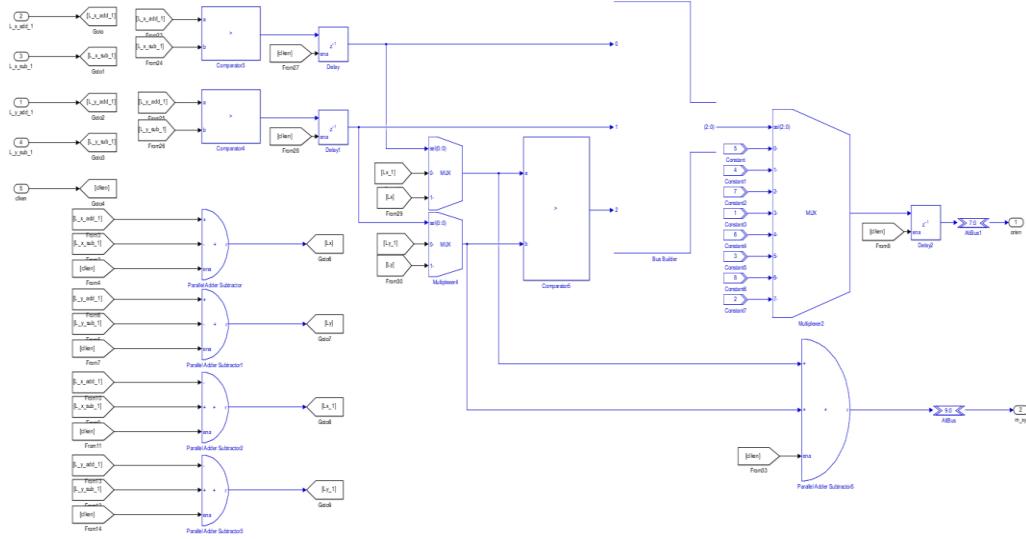
$m(x, y) \approx |L(x+1, y) - l(x-1, y)| + |L(x, y+1) - l(x, y-1)|$
và hướng biến đổi sẽ được quy đổi về 8 bin hướng.



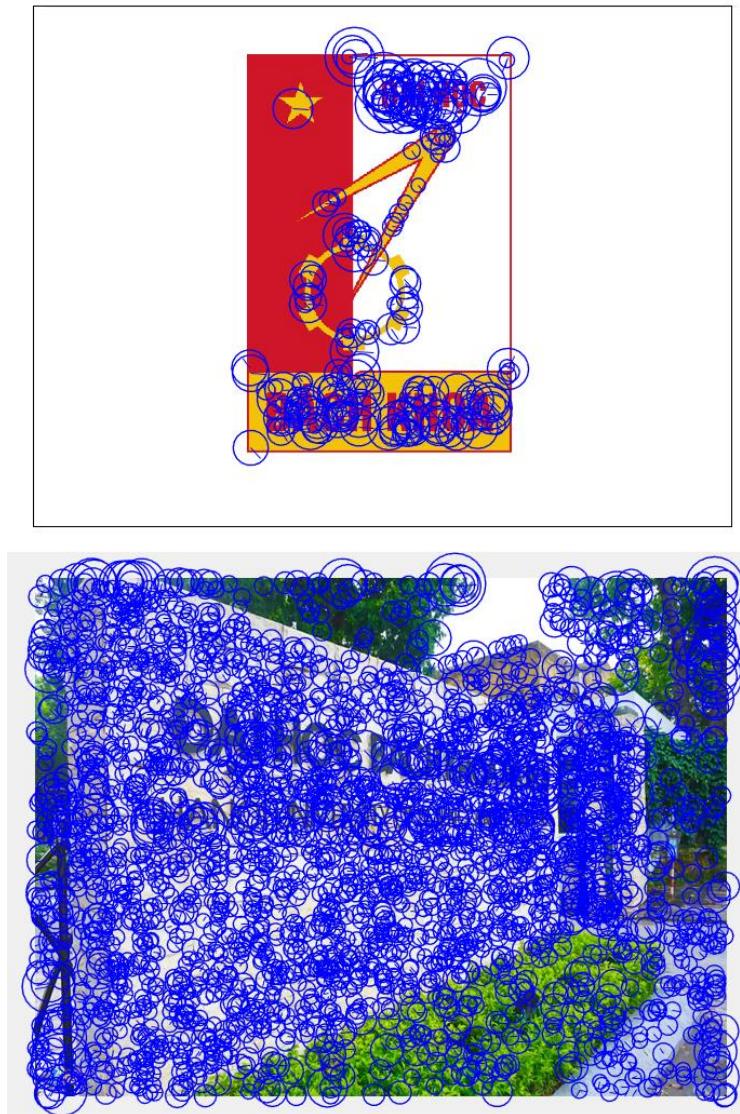
Hình 3.32 Thiết kế khối GradOrien



Hình 3.33 Thiết kế khối line_buff3x3



Hình 3.34 Thiết kế khói sub_GradOrien

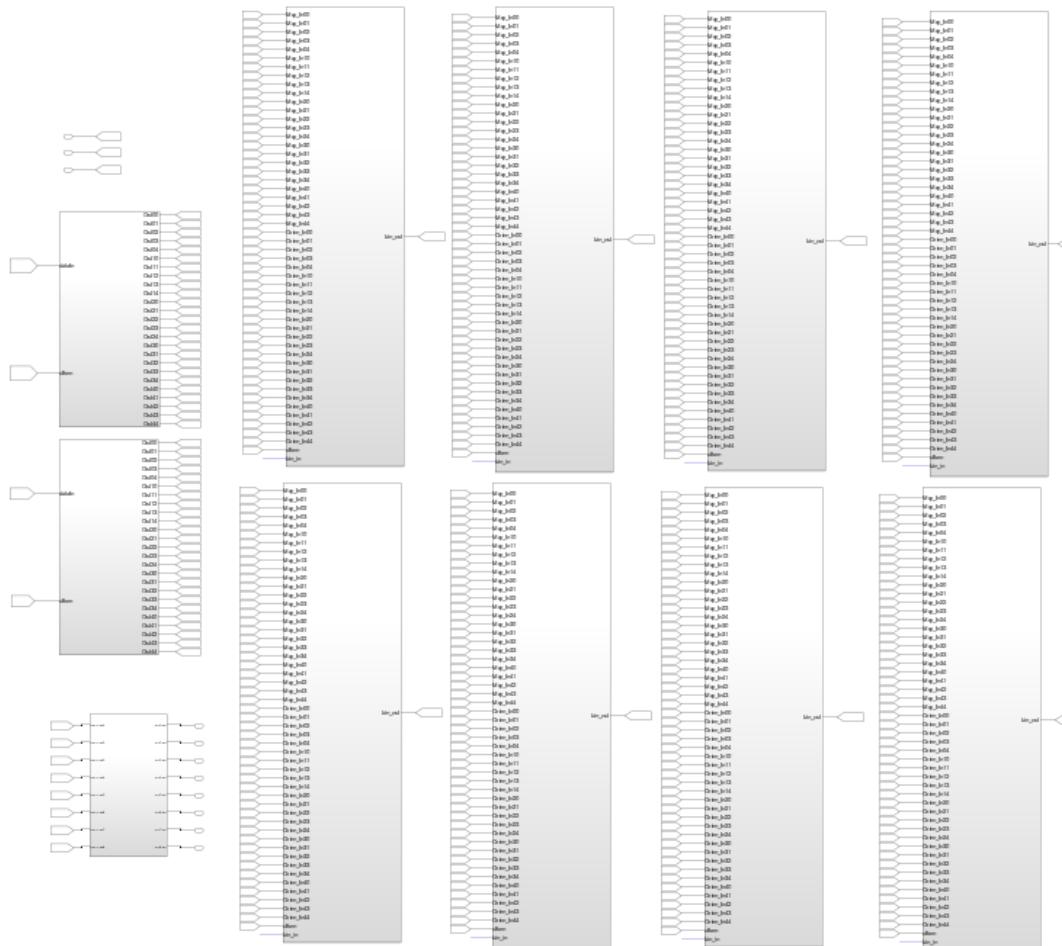


Hình 3.35 Kết quả mô phỏng IP Core GradOrien với detect keypoint

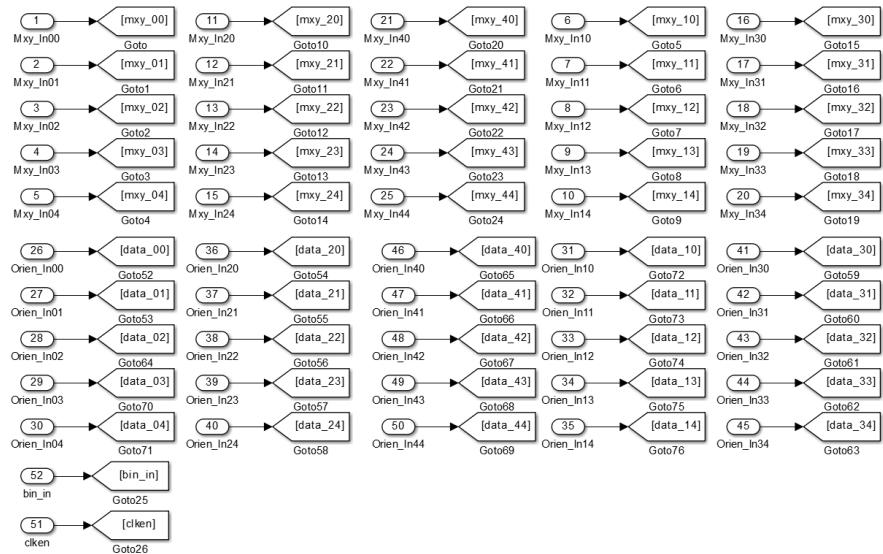
Hình 3.35 mô tả kết quả mô phỏng tìm kiếm phát hiện điểm hấp dẫn (keypoint) từ ảnh đầu vào và tính toán hướng và độ lớn biến đổi của điểm hấp dẫn đó trong ảnh.

- Khối **hist_bin** thực hiện tính histogram hướng và độ lớn biến đổi của điểm ảnh trong từng ma trận 5×5 điểm ảnh (điểm chính giữa là điểm đang xét) trả về 1 vector 8 chiều đại diện cho 8 bin hướng, giá trị của mỗi bin là tổng độ lớn của các điểm ảnh có cùng hướng biến đổi. Mỗi bin sẽ được biểu diễn bằng 16bit.

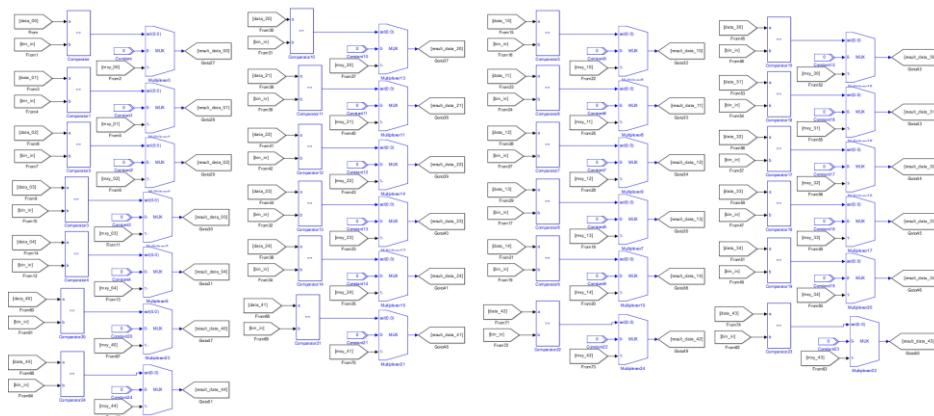
Khối **hist_bin** gồm 2 khối **line_buff5x5** để đếm lại 5 hàng dữ liệu hướng và độ lớn biến đổi của mỗi pixel trong ảnh, 8 khối **hist_bin1** đến **hist_bin8** để tính toán histogram của từng hướng biến đổi và 1 khối **final_hist_bin** tìm kiếm bin có histogram lớn nhất và giữ lại giá trị của những bin có histogram lớn hơn 50% giá trị histogram của bin lớn nhất.



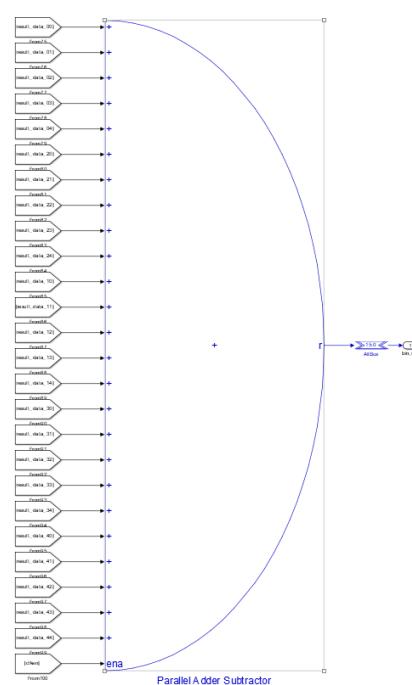
Hình 3.36 Thiết kế khối **hist_bin**



Hình 3.37 Các chân input của khối hist_bin1

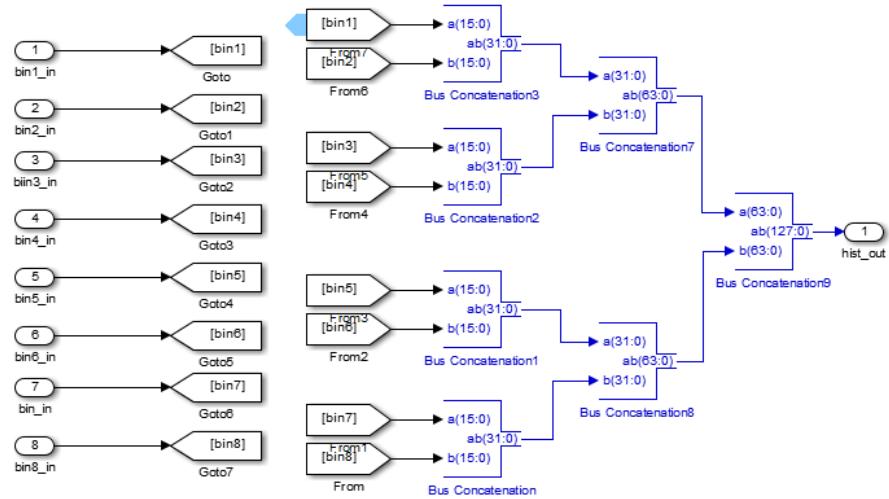


Hình 3.38 Thiết kế các phần so sánh của khối hist_bin1



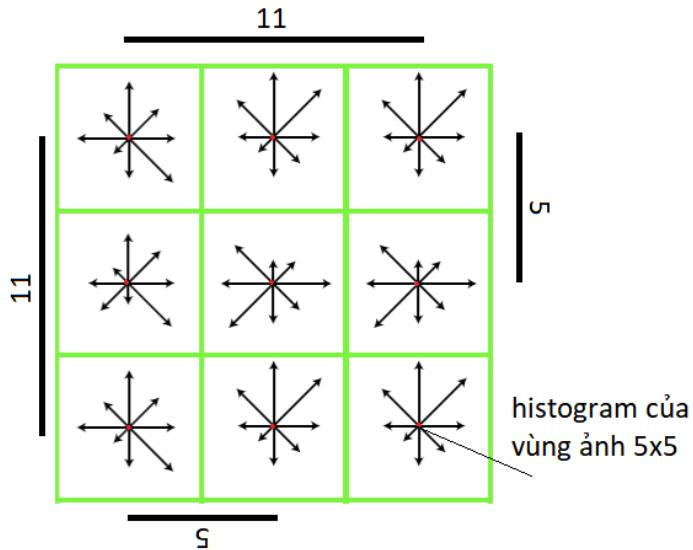
Hình 3.39 Tính tổng độ lớn biến đổi của các điểm ảnh có cùng hướng biến đổi

- Khối **Bus_Concatenation** thực hiện ghép histogram tính được lại thành 1 gói bus dữ liệu 128-bit để dễ dàng lưu trữ và truyền nhận giữa các IP Core thành phần khác.



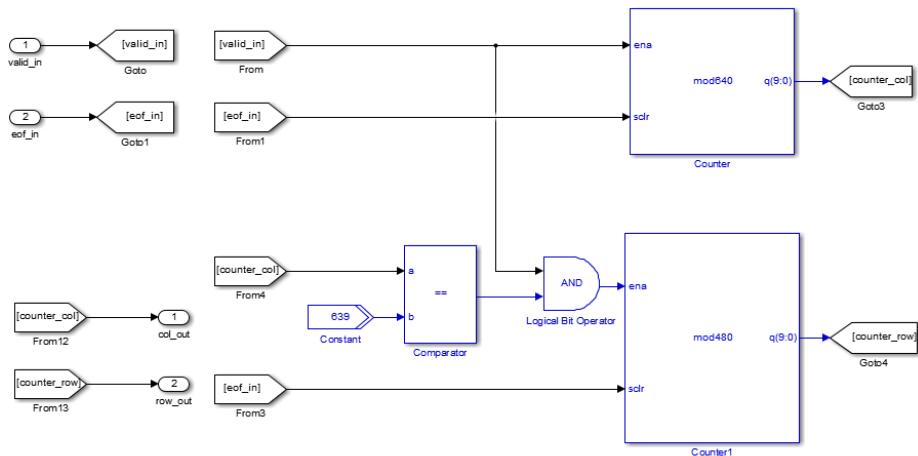
Hình 3.40 Thiết kế khối Bus_Concatenation

- Khối **line_buff11x11** dùng để đếm lại luồng giá trị histogram tính toán ở trên và trích ra 3x3 histogram của 9 vùng ảnh 5x5 xung quanh điểm ảnh chính giữa vùng ảnh đó để tạo descriptor cho điểm ảnh đó. Descriptor có $9 \times 8 = 72$ chiều.



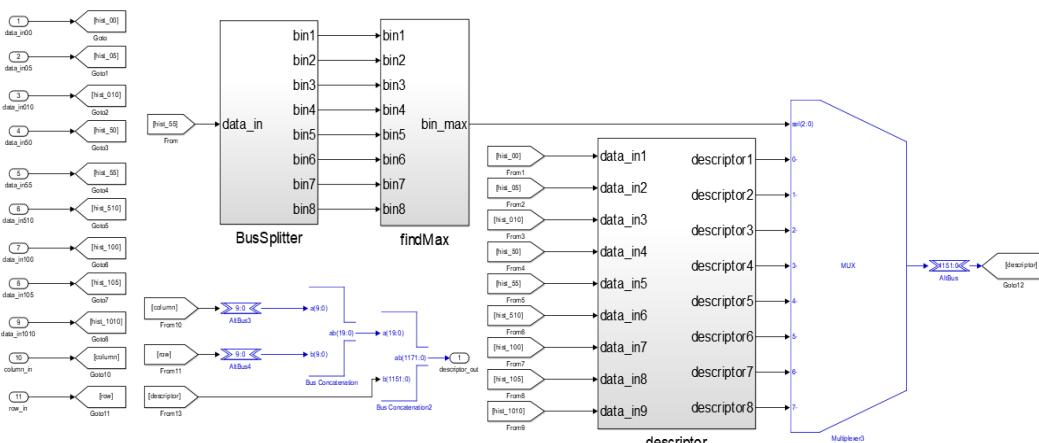
Hình 3.41 Mô phỏng vị trí của các histogram được sử dụng xây dựng descriptor

- Khối **counter** xác định vị trí của dữ liệu đang xử lý tương ứng với vị trí pixel trên ảnh.

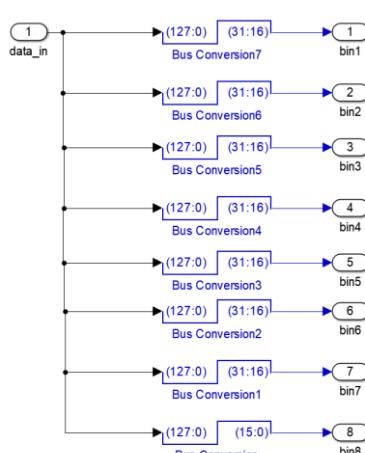


Hình 3.42 Thiết kế khối counter

- Khối **descriptor** gồm các khối nhỏ chính **BusSplitter**, **findMax** và **descriptor**. Khối **BusSplitter** và khối **findMax** sẽ tìm ra hướng biến đổi lớn nhất của vùng ảnh 5x5 chính giữa **line_buff5x5** để làm tín hiệu lựa chọn descriptor tương ứng được sắp xếp lại trong khối **descriptor** theo một chiều duy nhất bắt đầu từ hướng biến đổi lớn nhất của vùng ảnh 5x5 chính giữa sau đó đến các histogram của các vùng ảnh xung quanh.

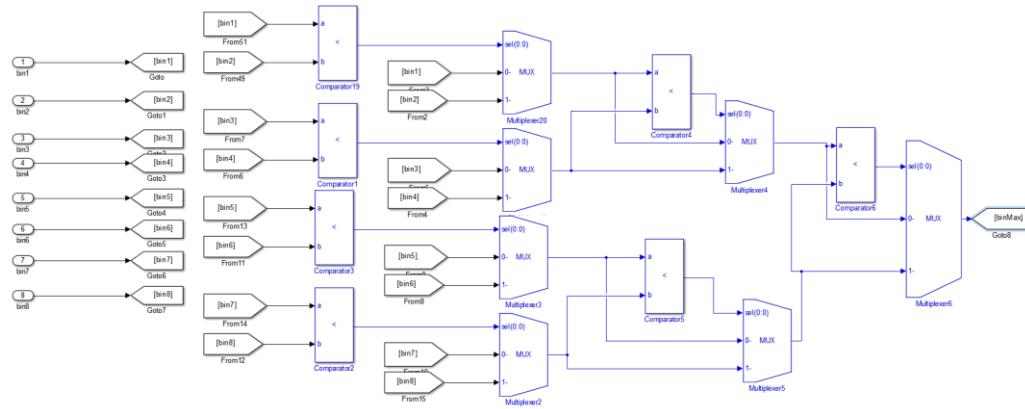


Hình 3.43 Thiết kế khối descriptor

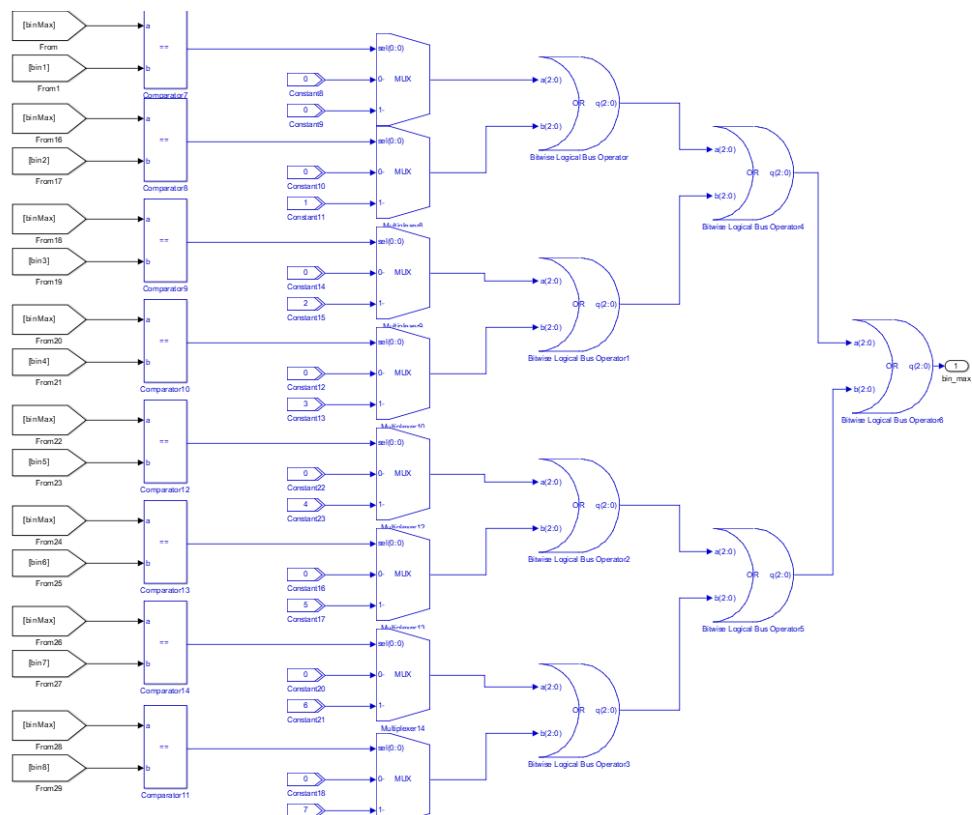


Hình 3.44 Thiết kế khối BusSplitter

Nghiên cứu, thiết kế và triển khai thuật toán SIFT trên FPGA

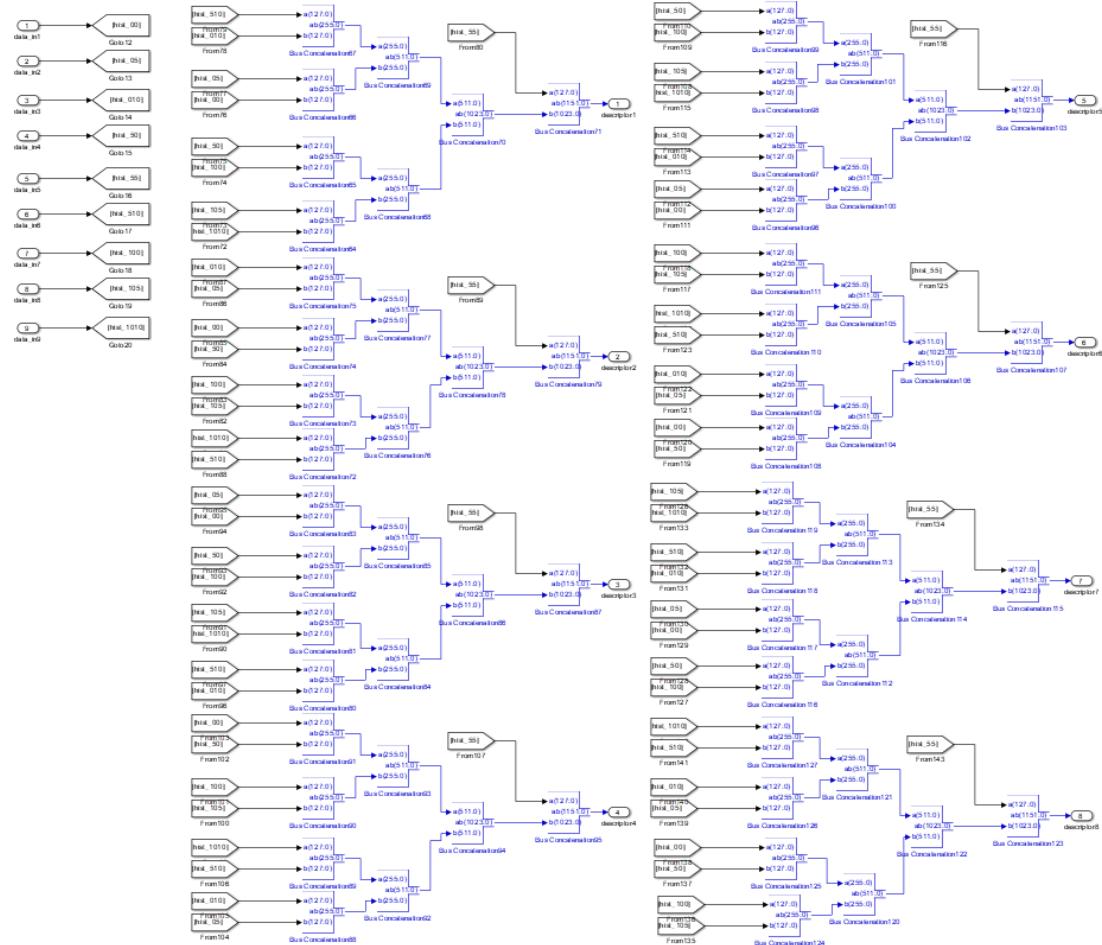


Hình 3.45 Thiết kế khói *findmax*



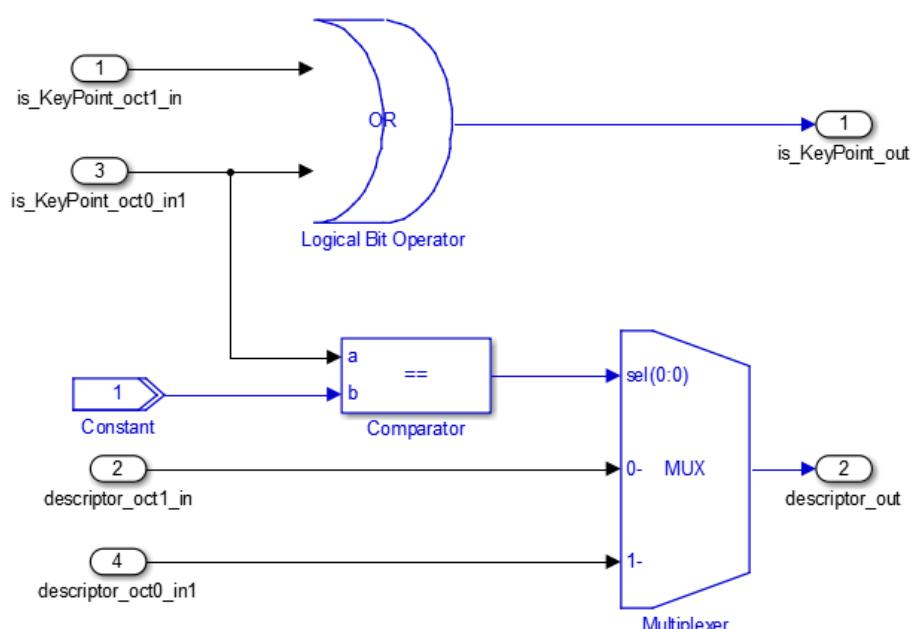
Hình 3.46 Thiết kế khói *findmax*

Nghiên cứu, thiết kế và triển khai thuật toán SIFT trên FPGA



Hình 3.47 Thiết kế khối subsystem descriptor

Khối final_descriptor lấy output của 2 khối **octave0** và **octave1** tổng hợp lại thành 2 tín hiệu **isKeyPoint** và **descriptor** duy nhất được đưa vào khối **matching**.



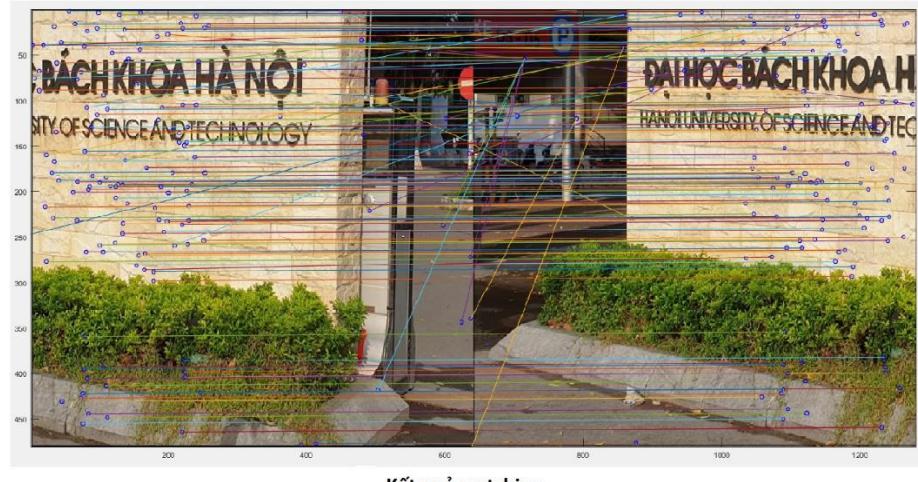
Hình 3.48 Thiết kế khối final_descriptor



Ảnh 1



Ảnh 2

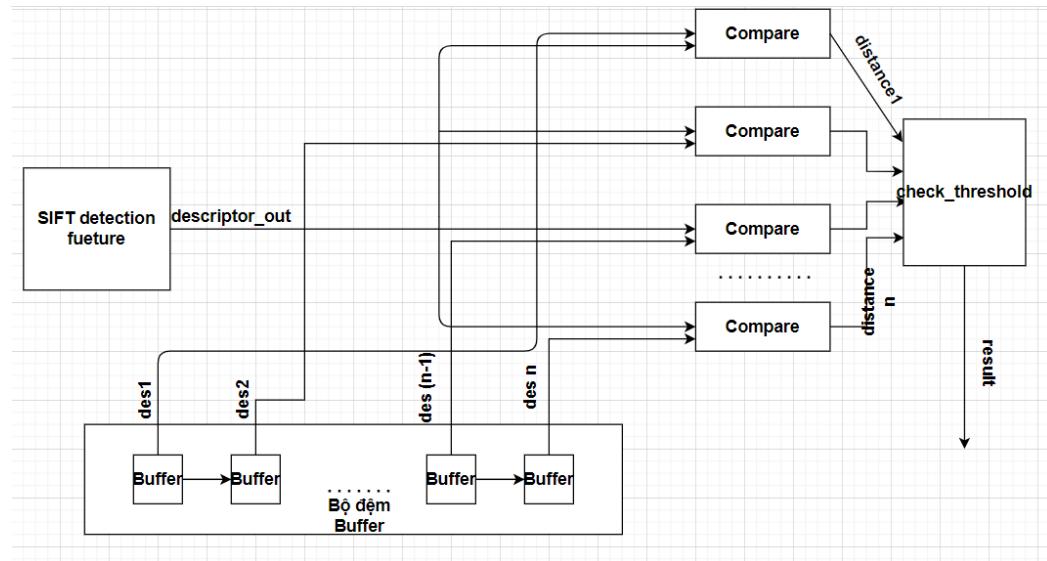


Kết quả matching

Hình 3.49 Kết quả so khớp dựa trên dữ liệu thu được từ mô phỏng của SIFT Detector và calculate_descriptor

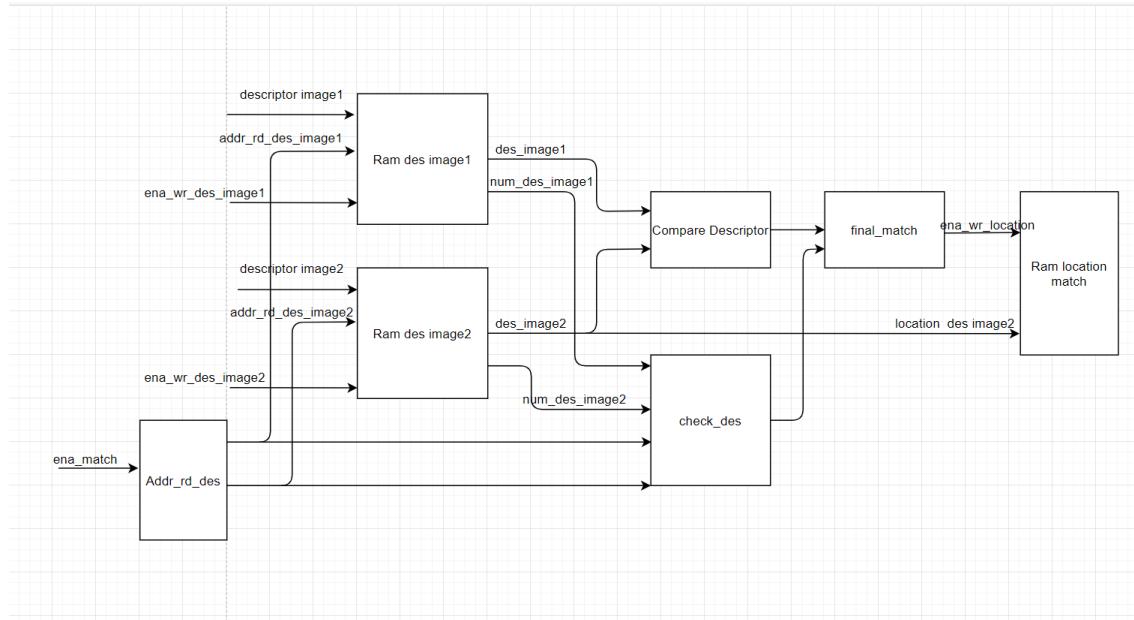
3.7 Thiết kế IP Core Matching

Ý tưởng thiết kế ban đầu của bộ matching là tận dụng ưu thế tính toán song song mạnh mẽ của phần cứng, tiến hành so khớp song song tất cả các điểm hấp dẫn của ảnh mẫu (như ảnh logo trong bài toán sử dụng thuật toán SIFT để phát hiện logo trên ảnh) với từng điểm hấp dẫn được tìm thấy trên luồng pixel được truyền theo giao diện Avalon-ST vào bộ xử lý thuật toán. Cách này đảm bảo được tính ưu việt về xử lý trên thời gian thực. Ảnh mẫu sẽ được xử lý trước, các bộ mô tả của điểm hấp dẫn sẽ được lưu trong một bộ đệm (buffer) và các ảnh này thường có số lượng điểm hấp dẫn ít (khoảng 200-500 điểm hấp dẫn). Tuy nhiên cách này không thể triển khai thành công trên kit thực tế vì khi dịch (compile) trên phần mềm Quartus để nạp lên Kit thì phần mềm báo lỗi vượt quá số lượng tài nguyên phần cứng có thể cung cấp. Nguyên nhân do khi tăng số lượng bộ so sánh 2 descriptor với nhau để thực hiện tính toán song song đã làm tăng quá nhiều tài nguyên phần cứng sử dụng. Trong đồ án chúng em sử dụng 500 bộ so sánh (tự thiết kế) nhưng tài nguyên trên Kit chỉ đủ cho khoảng 10 bộ so sánh. Do vậy phương án tận dụng thế mạnh của phần cứng xử lý song song không thể thực hiện.



Hình 3.50 Sơ đồ khái niệm cho matching song song

Do vậy, nhóm sinh viên thực hiện thực hiện chuyển hướng tiếp cận sang chuyển từ xử lý song song sang nối tiếp. Sử dụng 2 khối IP Core **Dual-Port RAM** do DSP Builder cung cấp cho phép người dùng có thể ghi vào và đọc ra dữ liệu (dữ liệu có kích thước do người dùng cài đặt) theo địa chỉ. Với cách tiếp cận đó chúng em sẽ gửi lần lượt từng ảnh bằng luồng pixel theo giao diện Avalon-ST qua bộ xử lý thuật toán. Các bộ mô tả của điểm hấp dẫn và vị trí của điểm hấp dẫn trên mỗi ảnh sẽ được lưu vào khối **Dual-Port RAM** tương ứng được chọn bởi tín hiệu từ Sw_control (Switch ngoại vi). Khi đã có đủ 2 tập các bộ mô tả điểm hấp dẫn của 2 ảnh sẽ bắt đầu quá trình đối sánh. Mỗi mô tả của một điểm hấp dẫn trong ảnh thứ nhất sẽ được so sánh với tất cả bộ mô tả điểm hấp dẫn của ảnh thứ hai (thực hiện vét cạn) vì vậy quá trình này sẽ lấy ra 1 bộ mô tả điểm hấp dẫn của ảnh thứ nhất và lấy lần lượt tất cả bộ mô tả điểm hấp dẫn của ảnh thứ 2 để so sánh. Khi đã so sánh hết với những điểm hấp dẫn của ảnh thứ 2 thì tiếp tục lấy bộ mô tả tiếp theo của ảnh thứ nhất ra so sánh. Trong quá trình đối sánh nếu phát hiện keypoint nào khớp với nhau thì sẽ lưu lại vị trí của điểm hấp dẫn trong ảnh vào một khối **Dual-Port RAM** thứ 3. Sau khi đối sánh song hai ảnh, thực hiện đọc vùng Ram lưu vị trí những điểm hấp dẫn khớp với nhau và thể hiện chúng trên luồng ảnh gửi ra màn hình bằng VGA để hiển thị kết quả kiểm tra.



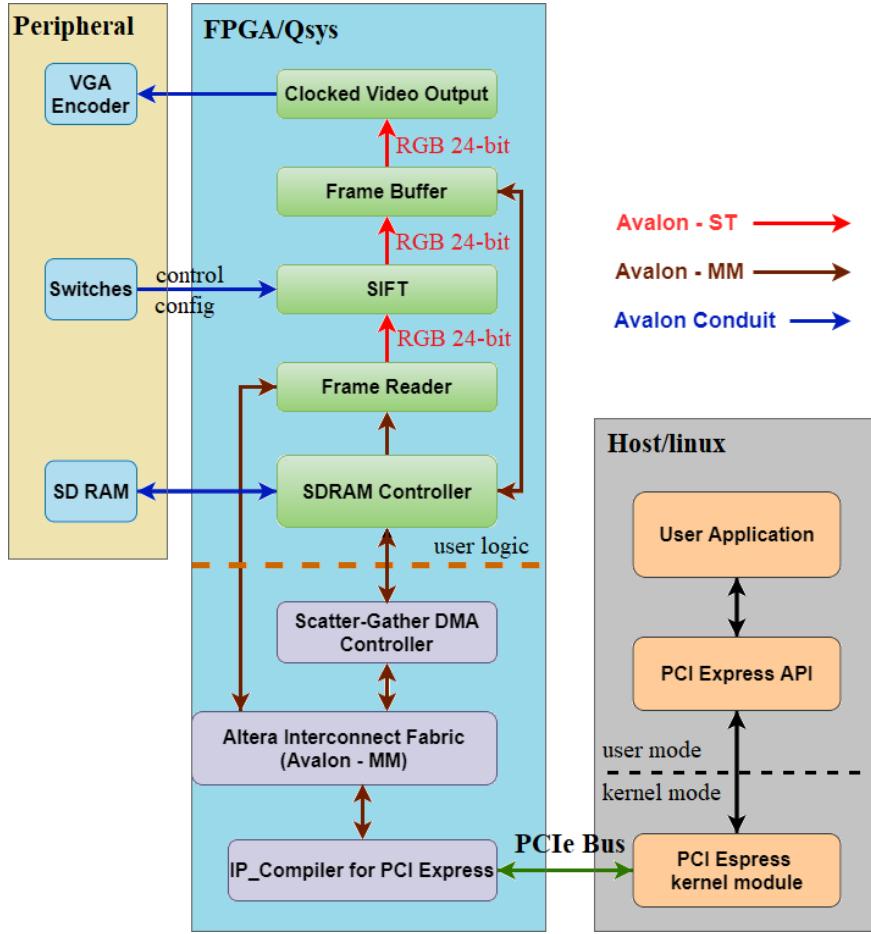
Hình 3.51 Sơ đồ khối matching nối tiếp

Định dạng của một bộ mô tả điểm hấp dẫn là một vector 72 chiều, giá trị mỗi chiều được biểu diễn bằng 16 bit. Bởi vậy khi thiết kế sẽ ghép các vector lại thành một tín hiệu duy nhất có độ rộng 1172-bit để dễ dàng trong việc truyền nhận giữa các khối và lưu trữ lại vào Ram. Nhưng trong quá trình thực hiện mô phỏng nhiều lần phát hiện giá trị dữ liệu khi gộp lại thành 1172-bit trên MATLAB thì bị làm tròn giá trị tại những bit thấp nên không thực hiện mô phỏng như những IP Core thiết kế phía trên. Do vậy, IP Core này cần được gỡ lỗi trực tiếp trên kit.

Chi tiết thiết kế khá phức tạp và nêu ở phụ lục A1 trang 84

CHƯƠNG 4. TRIỂN KHAI HỆ THỐNG TRÊN KIT DE2i-150

4.1 Kiến trúc tổng quan hệ thống



Hình 4.1 Sơ đồ kiến trúc tổng quan hệ thống

Hệ thống kế thừa và phát triển *PCI Express System Framework* (Hình 4.2), bao gồm 3 khối chính: Khối **Host/linux** – gồm chương trình ứng dụng, PCIe API, PCIe driver; khối **FPGA/Qsys** – gồm các IP Core, các bus kết nối, ... trên *Cyclone IV GX*; khối **Peripheral** – gồm các ngoại vi như SDRAM, Switches, VGA Chip được kết nối với *Cyclone IV GX*.

IP Core *Compiler for PCI Express* thực thi theo chuẩn giao thức PCIe, đóng vai trò giao tiếp với nửa PC, nhận và thực thi các yêu cầu gửi/nhận dữ liệu. Những dữ liệu này bao gồm các dữ liệu hình ảnh/video hay giá trị các thanh ghi cấu hình của các IP Core trong hệ thống được gửi từ phía Host qua bus PCIe; cũng có thể là các dữ liệu được trả về từ yêu cầu đọc dữ liệu phía Host. Vì vậy, những IP core cần giá trị cấu hình từ phía Host như *Frame Reader* hay làm nhiệm vụ đọc/ghi dữ liệu trên SDRAM của nửa FPGA như *SG – DMA* đều kết nối với IP Core *Compiler for PCI Express* thông qua *Altera Interconnect Fabric (Avalon – MM)*.

IP Core *SG – DMA (Scatter-Gather DMA Controller)* sẽ thực hiện việc trao đổi dữ liệu giữa RAM của nửa PC (được truyền qua PCIe tới) và SDRAM của nửa FPGA. Tuy nhiên IP Core *SG – DMA* không làm trực tiếp kết nối với RAM

của nửa PC cũng như SDRAM của nửa FPGA. *SG – DMA* sẽ thông qua IP Core *Compiler for PCI Express* để thực hiện việc trao đổi dữ liệu cần đọc/ghi với RAM của nửa PC, thông qua IP Core *SDRAM Controller* điều khiển việc đọc ghi SDRAM của nửa FPGA (chi tiết tại 4.3.1).

Các IP Core dùng để truyền dẫn và xử lý dữ liệu hình ảnh/video gọi chung là các user logic (các khối có màu xanh lá trong Hình 4.1). Trong đó, IP Core *SDRAM Controller*, *Frame Reader*, *Frame Buffer*, *Clocked Video Output* là các IP Core được hỗ trợ sẵn trong Altera Qsys. *Frame Reader* đọc các khung hình từ bộ nhớ ngoài (SDRAM) theo giao diện Avalon-MM thông qua IP Core *SDRAM Controller* và xuất chúng dưới dạng một luồng video theo giao diện Avalon – ST (chi tiết tại 4.3.2). *Frame Buffer* đệm dữ liệu vào bộ nhớ ngoài (SDRAM) thông qua *SDRAM Controller* và *SG – DMA*, dữ liệu này được chính *Frame Buffer* đọc ra và chuyển sang *Clocked Video Output* để thực hiện chuyển đổi dữ liệu hình ảnh/video dưới định dạng giao diện Avalon – ST sang các định dạng chuẩn như VGA để hiển thị. Ngoài ra, khối *SIFT* là IP Core thực hiện thuật toán trích chọn đặc trưng SIFT trên ảnh/video được thiết kế trong đồ án này.

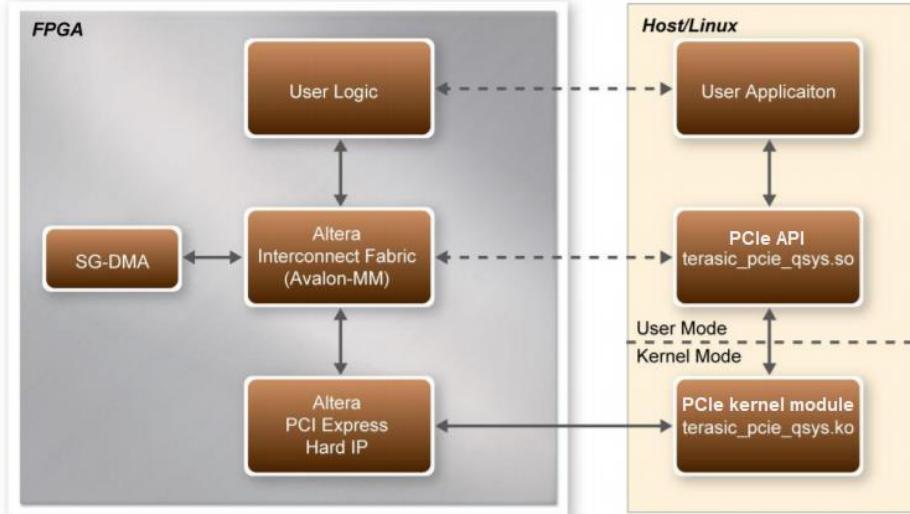
Dữ liệu được truyền từ ứng dụng phía Host sẽ được PCIe driver ở tầng nhân chuyển đổi theo chuẩn giao thức PCI Express và truyền qua bus PCIe tới IP Core *Compiler for PCI Express* ở phía FPGA. Nếu đây là dữ liệu cấu hình IP Core chúng sẽ thông qua thanh ghi địa chỉ cơ sở (*Base Address Register*) truyền tới và cấu hình các IP Core này. Nếu đây là dữ liệu cần ghi vào SDRAM như dữ liệu của từng khung hình, chúng sẽ được *SG – DMA* chuyển tới *SDRAM Controller* điều khiển quá trình ghi vào SDRAM. Các khung hình này sau đó sẽ được *Frame Reader* yêu cầu *SDRAM Controller* đọc từ SDRAM và gửi cho nó. Sau đó *Frame Reader* biến đổi dữ liệu này thành dữ liệu theo chuẩn giao diện Avalon – ST rồi chuyển tới IP Core *SIFT* để thực hiện thuật toán SIFT trên dữ liệu ảnh này. IP Core *SIFT* còn có một đầu vào là 18-bit tương ứng với trạng thái 18 slide switch trên kit DE2i-150 dùng để cấu hình cũng như điều khiển. Dữ liệu sau khi xử lý xong sẽ đưa vào IP Core *Frame Buffer* để đệm lại vào trong SDRAM rồi chuyển tới IP Core *Clocked Video Output* để chuyển đổi dữ liệu sang định dạng VGA và gửi tới VGA chip thực hiện mã hóa (VGA encoder) và xuất ra cổng VGA, hiển thị trên màn hình.

Khi ứng dụng phía bên Host muốn đọc dữ liệu đã xử lý được đệm trong SDRAM, nó sẽ gửi yêu cầu đến IP Core *Compiler for PCI Express* và yêu cầu này được truyền đạt lại cho IP Core *SG – DMA*. *SG – DMA* sẽ điều khiển IP Core *SDRAM Controller* đọc dữ liệu cần thiết và chuyển tới cho nó, sau đó bản thân nó sẽ trung chuyển dữ liệu này về cho IP Core *Compiler for PCI Express*. Cuối cùng dữ liệu này sẽ được chuyển qua bus PCI Express về phía Host và được lưu vào RAM (của nửa CPU) chờ đợi ứng dụng xử lý.

4.2 Trao đổi thông tin giữa nửa PC và nửa FPGA của kit DE2i-150

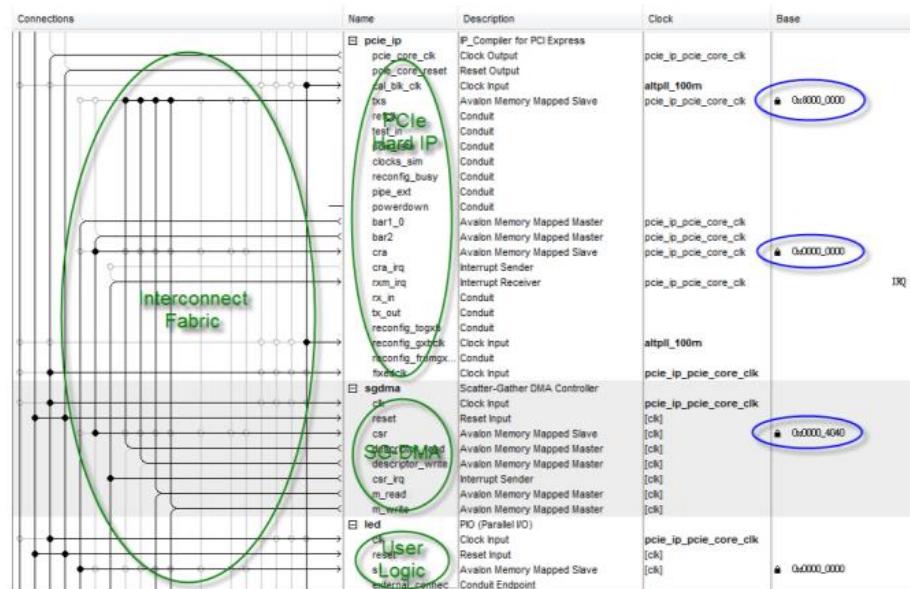
Như đã nêu ở mục 1.3, bộ xử lý *Intel Atom N2600* (ở nửa PC) và chip FPGA *Altera Cyclone IV GX* (ở nửa FPGA) được liên kết với nhau qua hai làn PCI

Express tốc độ cao để đảm bảo giao tiếp tốc độ cao giữa chúng. Việc giao tiếp giữa hai nửa PC và FPGA được thực hiện theo một framework có hai phần chính là: hệ thống PCIe trên FPGA được phát triển dựa trên *Altera Qsys System* và bộ phát triển phần mềm (SDK) PCIe trên CPU được cung cấp bởi Altera với các API thuận tiện cho việc giao tiếp với FPGA.



Hình 4.2 PCI Express System Framework

Ở nửa FPGA (phần framework trên FPGA), tất cả các các IP Core đều kết nối với nhau thông qua giao diện *Avalon Memory – Mapped* (Avalon – MM), tương ứng với khối *Altera Interconnect Fabric* trong Hình 4.2. Khối này không phải là IP Core mà là các nút kết nối trong hệ thống *Altera Qsys*.

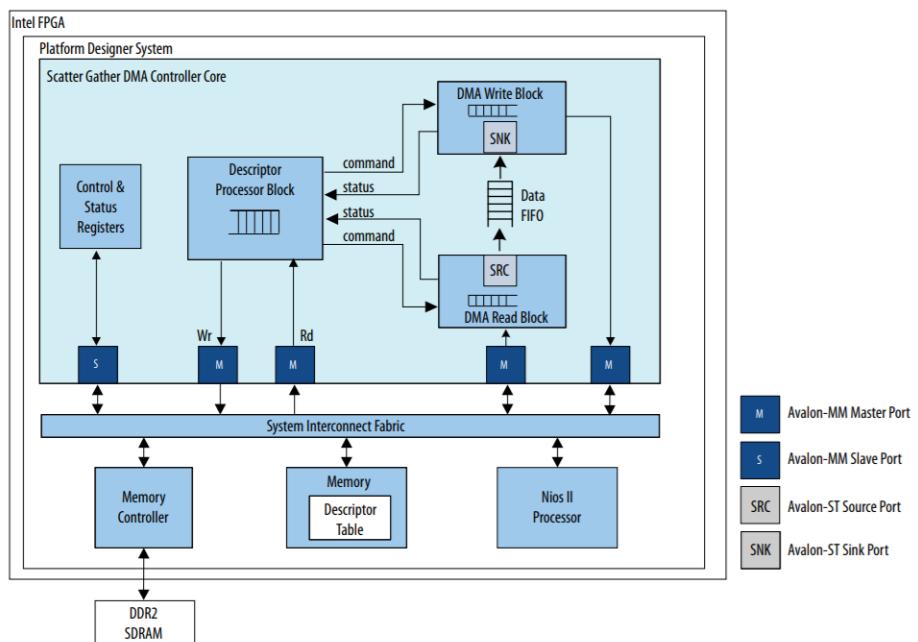


Hình 4.3 PCIe framework trên Altera Qsys

Khối SG – DMA (trong Qsys chính là *IP Scatter-Gather DMA Controller*) thực hiện việc truyền dữ liệu tốc độ cao giữa hai components (memory to memory, memory to stream, stream to memory). Trong đồ án này, ta **SG – DMA**

làm nhiệm vụ dữ liệu của từng frame từ Ram (của nửa CPU) đến SDRAM (của nửa FPGA) qua đường bus PCI Express Gen1 x1.

Cấu hình của **SG – DMA** trong trường hợp truyền memory to memory bao gồm ba khối: *Descriptor Processor*, *DMA Read*, *DMA Write* như Hình 4.4. *Descriptor Processor* mô tả rõ dữ liệu sẽ được truyền. SG – DMA sử dụng một giao diện chuyên dụng để đọc/ghi các bộ mô tả này và lưu trữ chúng dưới dạng danh sách liên kết trên các bộ nhớ on-chip hay off-chip với độ dài tùy ý. Chi tiết về cấu trúc DMA Descriptor được mô tả rõ trong mục *Scatter-Gather DMA Controller Core* của tài liệu [4].

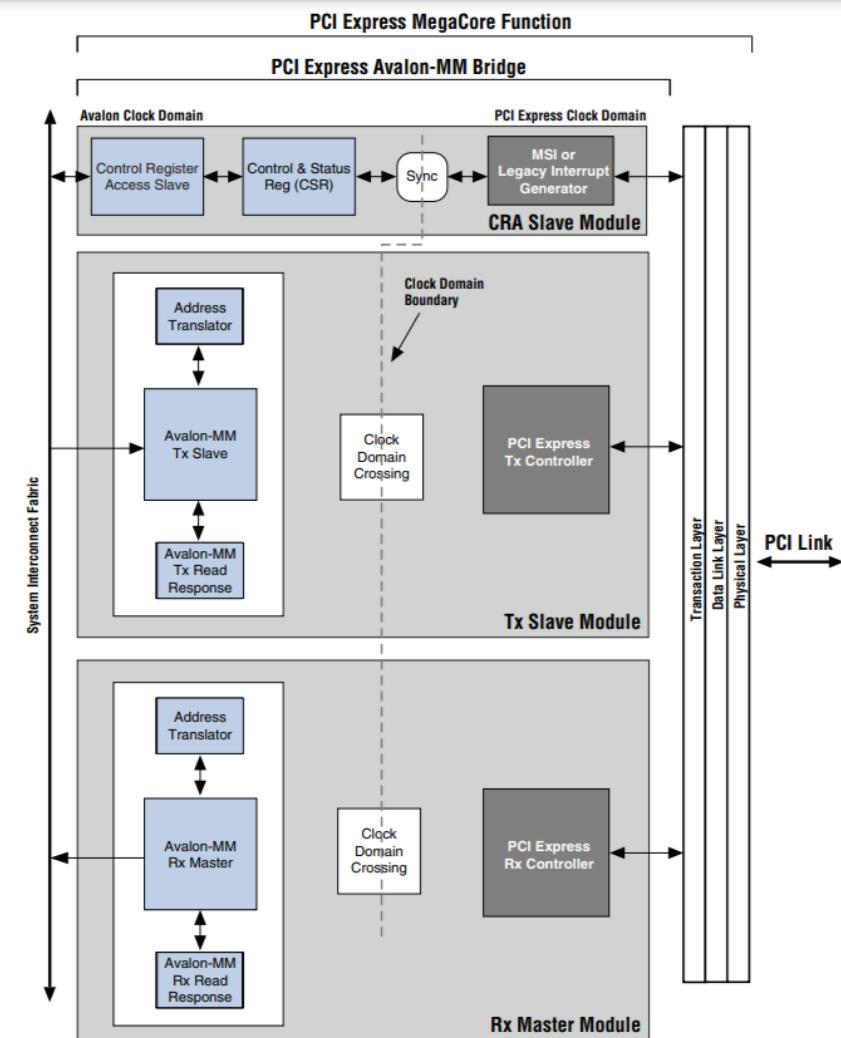


Hình 4.4 Ví dụ về cấu hình Memory-to-Memory của SG – DMA

Quá trình hoạt động bắt đầu với việc phần mềm phía **Host** sẽ xây dựng danh sách liên kết cho bộ mô tả nói trên và ghi địa chỉ của phần tử đầu tiên vào thanh ghi *next_descriptor_pointer* (thực hiện trong PCIe driver), đồng thời khởi tạo quá trình truyền dữ liệu. Khối *Descriptor Processor* đọc địa chỉ đầu tiên của danh sách mô tả trong thanh ghi *next_descriptor_pointer* và đẩy nó vào trong command FIFO của mình, cung cấp command cho cả *DMA Read* và *DMA Write*. Sau khi đọc xong bộ mô tả đầu tiên trong danh sách, khối này sẽ đọc bộ mô tả tiếp theo tương tự như trên và quá trình này lặp lại đến khi hết danh sách. *DMA Read* nhận dữ liệu địa chỉ nguồn từ FIFO command của nó và bắt đầu đọc dữ liệu đẩy vào Data FIFO thông qua SRC. *DMA Read* sẽ tạm dừng hoạt động nếu Data FIFO đầy cho đến khi Data FIFO có không gian trống để có thể tiếp nhận thêm dữ liệu. *DMA Write* đọc địa chỉ đích từ FIFO của nó và bắt đầu ghi ra cổng Avalon – MM Master cho đến khi chuyển hết số byte được chỉ định. *DMA Write* sẽ dừng hoạt động nếu Data FIFO trống.

Khối **Altera PCI Express Hard IP** (trong Qsys chính là **IP Compiler for PCI Express**) thực thi theo chuẩn giao thức PCIe để truyền dữ liệu qua lại với

phía CPU (thông qua PCIe driver). IP Core này có 2 thanh ghi địa chỉ cơ sở (*Base Address Register*) là ***bar1_0*** để điều khiển trực tiếp các khối ***User Logic***, ***bar2*** được cung cấp cho ***Host*** để cấu hình cho ***SG – DMA*** thông qua việc đọc/ghi giá trị của 2 thanh ghi này qua bus PCIe. Để cấu hình các tham số cho IP Core Altera PCI Express Hard IP xem thêm [5].



Hình 4.5 PCI Express Avalon-MM Bridge

IP Compiler for PCI Express sử dụng module ***PCI Express Avalon-MM bridge*** làm cầu nối giữa ***PCI link*** và ***System Interconnect Fabric*** (Avalon – MM). Thiết kế này giúp đơn giản hóa việc kết nối các IP Core khác vào hệ thống bus PCIe khi thiết kế hệ thống trong *Qsys Platform Designer*. Module này bao gồm 3 thành phần chính: ***TX Slave Module***, ***RX Slave Module***, ***CRA Slave Module***, hỗ trợ cơ chế chuyển đổi giữa 2 không gian địa chỉ ***Avalon-MM*** sang ***PCIe link*** và ngược lại. Việc chuyển đổi dựa trên ***Base Address Registers*** và ***Address Translation Table***.

TX Slave Module chuyển đổi các yêu cầu đọc/ghi bursting lên đến 4KB từ phía ***System Interconnect Fabric*** (Avalon – MM) sang các yêu cầu đọc/ghi theo gói trên hệ thống bus ***PCIe link***.

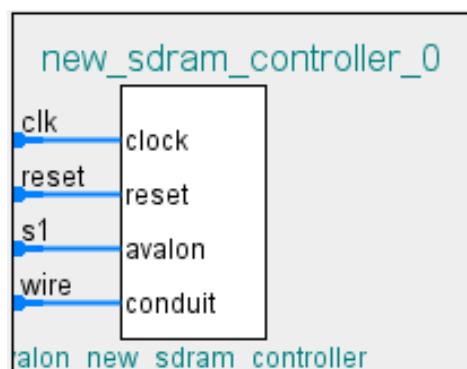
RX Master Module chuyển đổi các yêu cầu đọc/ghi theo gói nhận từ phía **PCIe link** sang các yêu cầu đọc/ghi bursting trên **System Interconnect Fabric**.

CRA Slave Module dùng để hỗ trợ việc đọc/ghi vào các thanh ghi điều khiển, trạng thái bên trong FPGA từ phía CPU, ví dụ như cấu hình *dynamic address translation* trên PCIe IP Core.

Ở nửa CPU (phần framework trên Host/linux), cần cài đặt driver cho PCI Express (**PCIe kernel module**). Altera (Intel) đã cung cấp sẵn cho driver này cùng với các hàm API tương ứng để lập trình ứng dụng đọc/ghi dữ liệu thông qua PCI Express. Chương trình ứng dụng được viết trên hệ điều hành linux, sẽ gọi các hàm PCIe API để ra lệnh đọc/ghi dữ liệu vào SDRAM phía FPGA. Các lệnh này sẽ được gửi tới PCIe driver để mã hóa theo chuẩn giao thức PCI Express và truyền sang phía FPGA, **IP Compiler for PCI Express** nhận lệnh, giải mã và tiến hành điều khiển các IP Core khác như **SG – DMA** để thực hiện đọc/ghi dữ liệu. Những dữ liệu này được trao đổi thông qua hai đường kết nối PCIe gen1.

4.3 Các IP Core có sẵn trong Qsys sử dụng trong hệ thống

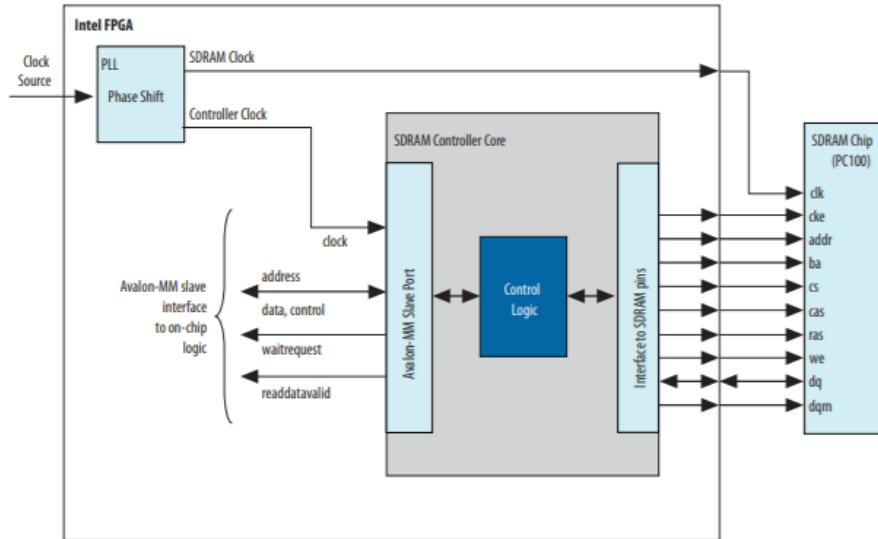
4.3.1 IP Core SDRAM Controller



Hình 4.6 Block diagram IP Core SDRAM Controller

IP Core *SDRAM Controller* cung cấp giao diện Avalon Memory – Mapped (Avalon – MM) kết nối với off – chip SDRAM, cho phép các nhà thiết kế có thể dễ dàng tạo ra các hệ thống tùy chỉnh trong FPGA của Intel có khả năng kết nối dễ dàng với SDRAM. IP Core này hỗ trợ kết nối các SDRAM theo tiêu chuẩn thông số kỹ thuật PC100.

IP Core *SDRAM Controller* kết nối với các IP Core khác trong hệ thống có nhu cầu đọc/ghi dữ liệu với SDRAM thông qua giao diện Avalon – MM. Cổng **s1** của IP Core là một Avalon – MM Slave, được thiết kế để có khả năng kết nối với Avalon – MM Master được cấu hình cho đọc hay ghi dữ liệu. IP Core cho phép truy cập SDRAM để đọc/ghi dữ liệu với độ rộng bus dữ liệu tùy chỉnh (8, 16, 32 hoặc 64 bit), kích thước bộ nhớ, và số lượng chip theo tham số. Cổng **wire** của IP Core sử dụng giao diện Avalon Conduit, được kết nối với SDRAM truyền dữ liệu được đọc/ghi giữa SDRAM và cổng **s1**.



Hình 4.7 Sơ đồ kết nối và sử dụng SDRAM Controller

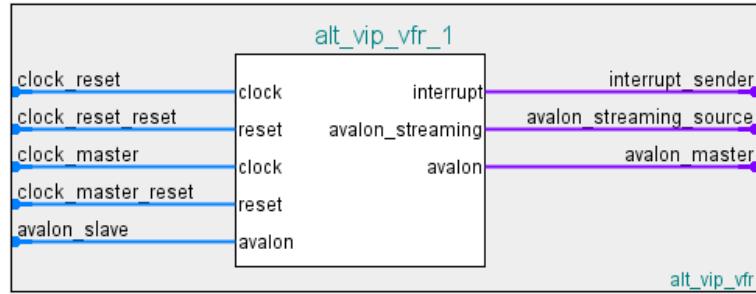
Cổng **s1** và **wire** ở Hình 4.6 tương ứng với Avalon – MM slave và Interface to SDRAM pins ở Hình 4.7. Khối PLL tạo các xung đồng hồ để đồng bộ giữa *SDRAM Controller* với chip SDRAM, các IP Core on-chip muốn sử dụng *SDRAM Controller* để đọc ghi thì cần có một giao diện Avalon – MM master kết nối vào cổng **s1** của *SDRAM Controller*.

Bảng 4.1 Các tham số cấu hình IP SDRAM Controller

Tên tham số	Giá trị (mặc định)	Mô tả
Data Width	8, 16, 32, 64 (32)	Độ rộng bus dữ liệu sẽ đọc/ghi.
Chip Selects	1, 2, 4, 8 (1)	Số lượng chip nhớ độc lập trong SDRAM.
Banks	2, 4 (4)	Giá trị banks của SDRAM, có trong data sheet của SDRAM sử dụng.
Row	11, 12, 13, 14 (12)	Số bit dùng cho địa chỉ hàng. Ví dụ 1 ram được tổ chức thành 4096 hàng và 512 cột thì Row = 12 ($4096 = 2^{12}$)
Column	≥ 8 $< \text{Row}$ (8)	Số bit dùng cho địa chỉ cột. Với ví dụ trên thì Column = 9 ($512 = 2^9$)

Ngoài các tham số cấu hình bộ nhớ cho SDRAM nêu ở Bảng 4.1, IP *SDRAM Controller* còn có các tham số cấu hình Timing cho SDRAM. Giá trị các tham số này sẽ đặt theo data sheet của SDRAM sử dụng do nhà sản xuất cung cấp. Chi tiết xem tại mục “32. *SDRAM Controller Core*” trong [4].

4.3.2 IP Core Frame Reader

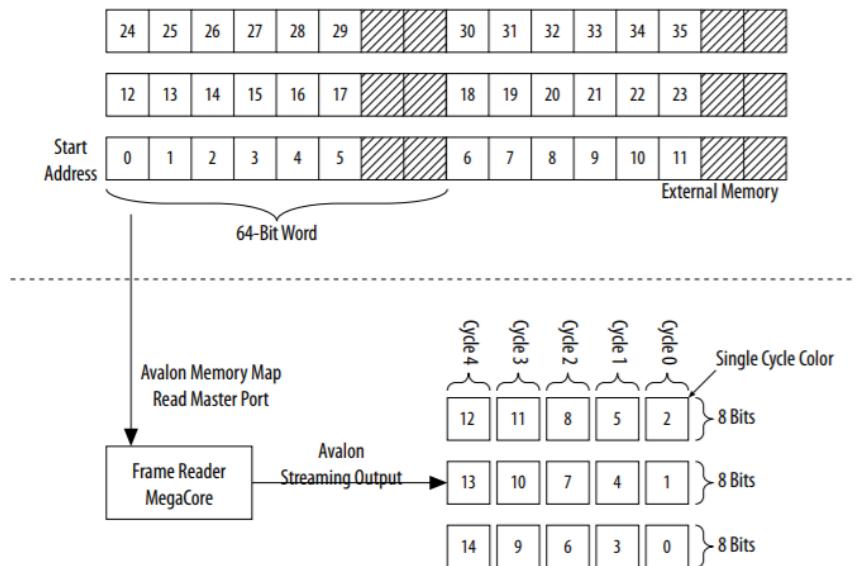


Hình 4.8 Block diagram IP Core Frame Reader

IP Core *Frame Reader* đọc các khung hình (frames video) từ bộ nhớ ngoài qua cổng **avalon_master** theo giao diện Avalon-MM và xuất chúng dưới dạng một luồng video qua cổng **avalon_streaming_source** theo giao diện Avalon – ST.

Cổng **avlon_slave** cung cấp các dữ liệu cấu hình cho IP Core, cho phép xác định tối đa hai vị trí bộ nhớ, mỗi vị trí chứa một khung hình. Cổng này được kết nối IP Core *Compiler for PCI Express* thông qua thanh ghi địa chỉ cơ sở (Base Address Register) **bar1_0**. Để điều khiển IP Core *Frame Reader*, ta sẽ ghi giá trị cho thanh ghi này thông qua bus PCIe.

Các khung hình (frames video) được lưu trữ trong bộ nhớ ngoài dưới dạng dữ liệu video thô (chỉ bao gồm giá trị các pixel), ngay trước khi IP Core *Frame Reader* đọc những khung hình từ bộ nhớ ngoài, nó tạo ra một gói (packer) điều khiển và tiêu đề cho gói dữ liệu video được đẩy ra giao diện Avalon – ST. Sau đó, dữ liệu video từ bộ nhớ ngoài sẽ được truyền trực tiếp qua IP Core. Nội dung của khối điều khiển dữ liệu được thiết lập thông qua cổng **avalon_slave**. Quá trình này được lặp lại với mọi khung hình được đọc từ bộ nhớ ngoài.



Hình 4.9 Sơ đồ đọc dữ 3 kênh song song, mỗi kênh 8-bit bằng Frame Reader

Hình 4.9 minh họa một bộ nhớ ngoài được tổ chức thành từng word (từ nhớ) 64-bit. Frame Reader sẽ đọc những dữ liệu này theo từng word 64-bit thông qua

Avalon – MM Master Port, thêm gói điều khiển, tiêu đề và xuất ra qua cổng Avalon – ST dưới định dạng 3 kênh màu song song, giá trị mỗi kênh màu được thể hiện qua 8-bit.

Bảng 4.2 Tham số cấu hình cho Frame Reader

Tham số	Giá trị	Mô tả
Bits per pixel per color plane	4 – 20 Default = 8	Lựa chọn số lượng bit của mỗi kênh màu cho mọi pixel.
Number of color planes in parallel	1 – 4 Default = 3	Lựa chọn số lượng kênh màu được truyền song song.
Number of color planes in sequence	1 – 3 Default = 1	Lựa chọn số lượng kênh màu được truyền nối tiếp.
Maximum image width	32 – 2600 Default = 640	Chiều rộng tối đa của khung hình tính bằng pixel.
Maximum image height	32 – 2600 Default = 480	Chiều cao tối đa của khung hình tính bằng pixel.
Master port width	16 – 256 Default = 256	Độ rộng bus dữ liệu truy cập vào bộ nhớ ngoài.
Read master FIFO depth	16 – 1024 Default = 64	Chọn kích thước FIFO cho read master.
Read master FIFO burst target	2 – 256 Default = 32	Chọn kích thước burst dữ liệu cho read master.
Use separate clocks for the Avalon-MM master interfaces	On / Off	Chọn On nếu muốn dùng clock riêng cho Avalon-MM master interfaces.

Ngoài tham số cấu hình trình bày ở Bảng 4.2, chi tiết về các tín hiệu, các thanh ghi điều khiển của IP Core *Frame Reader* được trình bày ở mục “*Frame Reader IP Core*” của tài liệu [6].

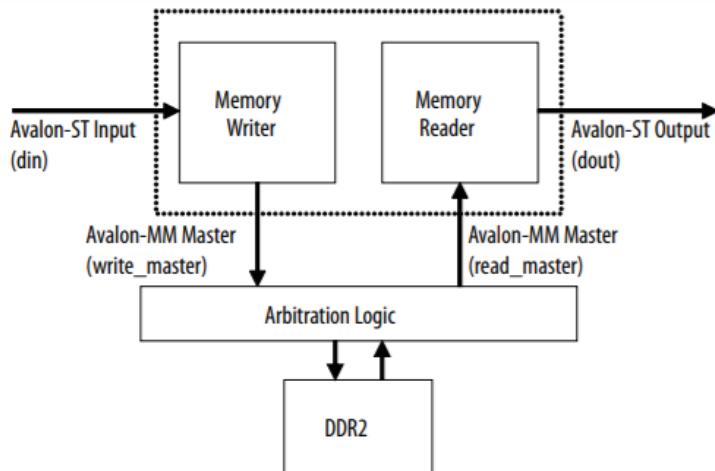
4.3.3 IP Frame Buffer



Hình 4.10 Block diagram IP Core Frame Buffer

IP Frame Buffer đệm dữ liệu của từng frame vào trong bộ nhớ ngoài (external RAM), gồm hai khối chính là khối Writer – lưu trữ những input pixel vào trong bộ nhớ ngoài và khối Reader – lấy các khung hình có trong bộ nhớ ngoài ra rồi định dạng chúng theo chuẩn Avalon – ST để xuất ra cổng **dout** (Hình 4.11).

Frame Buffer hỗ trợ truyền một pixel trong một lần truyền. Ngoài ra, IP Core còn hỗ trợ hai chế độ double-buffering và triple-buffering với một loạt các tùy chọn để xử lý trường hợp thả khung (frame dropping) hay lặp khung (frame repeating). Nếu không cho phép thả khung và lặp khung, IP Core sẽ cung cấp chế độ double-buffering với khả năng giải quyết các vấn đề về thông lượng trong đường dẫn dữ liệu. Nếu cho phép thả khung hoặc lặp khung, IP Core sẽ cung cấp chế độ triple-buffering với khả năng chuyển đổi tốc độ khung hình đơn giản hơn.



Hình 4.11 Sơ đồ chiều chuyển dữ liệu của IP Frame Buffer

Chế độ double-buffering, IP Core sử dụng hai buffer trong bộ nhớ ngoài để đệm khung hình. Khối Writer sử dụng buffer thứ nhất để lưu những pixel đầu vào, khối Reader sẽ khóa và đọc các pixel ở buffer thứ hai, xử lý và xuất ra đầu ra. Khi cả hai khối Writer và Reader cùng hoàn thành việc xử lý một khung, hai buffer sẽ được hoán đổi, sau đó có thể đọc lại dữ liệu của khung hình vừa xử lý và gửi đến đầu ra trong khi bộ đệm đang xử lý khung hình ngay sau nó.

Chế độ triple-buffering, IP Core sử dụng 3 buffer trong bộ nhớ ngoài để đệm khung hình. Khối Writer ghi dữ liệu vào buffer thứ nhất, khối Reader đọc dữ liệu từ buffer thứ hai tương tự như chế độ double-buffering. Khác ở chỗ thay việc hoán đổi hai buffer này khi cả hai khối Reader và Writer đều xử lý xong một khung hình thì sử dụng buffer thứ ba (buffer dự phòng) cho phép hoán đổi hai buffer một cách không đồng bộ. Buffer dự phòng có hai trạng thái là clean – chứa một khung hình mới chưa được gửi đi và dirty – chứa một khung hình đã được gửi đi bởi khối Reader.

Khi Writer hoàn thành việc ghi một khung hình vào bộ nhớ, nó sẽ hoán đổi với buffer nó vừa ghi dữ liệu với buffer dự phòng nếu buffer dự phòng ở trạng thái dirty. Khi đó, buffer chứa khung hình mới vừa được hoán đổi sẽ trở thành buffer dự phòng với trạng thái clean (do dữ liệu là khung hình mới chưa được khối Reader đọc). Khi Reader hoàn thành việc đọc và tạo ra một khung hình từ dữ liệu trong bộ nhớ, nó sẽ hoán đổi bộ đệm của nó với bộ đệm dự phòng nếu bộ đệm dự phòng ở trạng thái clean. Sau khi hoán đổi, bộ đệm vừa được sử dụng bởi Reader sẽ trở thành bộ đệm dự phòng với trạng thái dirty.

Trong trường hợp Writer hoàn thành ghi một khung hình mới nhưng buffer dự phòng ở trạng thái clean, nếu cho phép thả khung hình thì Writer sẽ thả khung hình vừa nhận và ghi đè bộ đệm bằng khung hình kế tiếp chứ không hoán đổi. Còn nếu không cho phép thả khung hình thì Writer sẽ dừng lại chờ cho đến khi Reader hoàn thành việc đọc khung hình ở buffer của nó và hoán đổi buffer này với buffer dự phòng đang ở trạng thái clean và buffer dự phòng mới sẽ ở trạng thái dirty.

Trong trường hợp bộ đệm dự phòng ở trạng thái dirty khi Reader hoàn thành khung hình đầu ra, nếu cho phép lặp khung, Reader sẽ lập tức lặp lại khung hình vừa được gửi đến đầu ra. Nếu không cho phép lặp khung, Reader sẽ dừng lại cho đến khi Writer hoàn thành việc ghi dữ liệu vào buffer của nó rồi trao đổi với buffer dự phòng, lúc này buffer dự phòng mới sẽ ở trạng thái clean.

Bảng 4.3 Chi tiết tham số định dạng dữ liệu khung hình của Frame Buffer

Tham số	Giá trị	Mô tả
Maximum image width	32 – 2600 Default = 640	Chiều rộng tối đa của khung hình tính bằng pixel.
Maximum image height	32 – 2600 Default = 480	Chiều cao tối đa của khung hình tính bằng pixel.
Bits per pixel per color plane	4 – 20 Default = 8	Lựa chọn số lượng bit của mỗi kênh màu cho mọi pixel.
Number of color planes in sequence	1 – 3 Default = 3	Lựa chọn số lượng kênh màu được truyền nối tiếp.
Number of color planes in parallel	1 – 3 Default = 1	Lựa chọn số lượng kênh màu được truyền song song.

Bảng 4.4 Chi tiết tham số bộ xử lý dữ liệu phi hình ảnh của Frame Buffer

Tham số	Giá trị	Mô tả
Number of packets buffered per frame	0 – 32 Default = 0	Chỉ định số lượng các gói video Avalon – ST phi hình ảnh, không có gói điều khiển có thể được lưu vào trong bộ đệm với mỗi khung hình. Các gói cũ hơn sẽ bị loại bỏ trước trong trường hợp tràn.
Maximum packet length	10 – 1024 Default = 10	Chọn độ dài gói tối đa dưới dạng một số ký hiệu. Giá trị tối thiểu là 10 vì đây là kích thước của gói điều khiển Avalon-ST (bao gồm tiêu đề). Các mẫu thừa bị loại bỏ nếu gói lớn hơn mức cho phép.

Bảng 4.5 Chi tiết tham số behavior của Frame Buffer

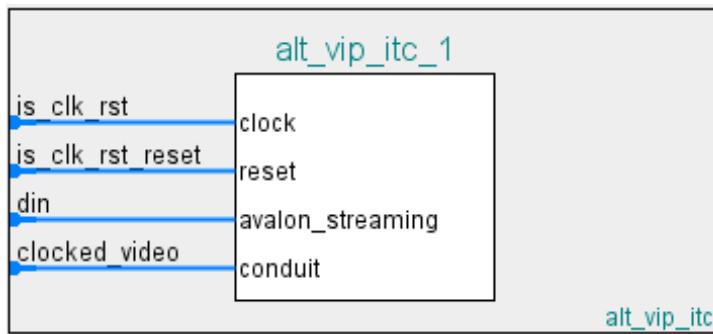
Tham số	Giá trị	Mô tả
Frame dropping	On / Off	Bật/néu cho phép thả khung.
Discard invalid frames/fields	On / Off	Bật để thả các gói dữ liệu hình ảnh có độ dài không tương thích với khai báo ở cuối gói.
Frame repetition	On / Off	Bật/néu cho phép lặp khung.
Run-time control for the writer thread	On / Off	Bật để điều khiển run-time cho giao diện ghi.
Run-time control for the reader thread	On / Off	Bật để điều khiển run-time cho giao diện đọc.
Support for locked frame rate conversion	On / Off	Bật để thêm giao diện đồng bộ hóa khung đầu vào và đầu ra (Avalon – MM Slave)
Support for interlaced streams	On / Off	Bật để hỗ trợ thả và lặp các trường nhất quán trong luồng video xen kẽ.

Bảng 4.6 Chi tiết tham số cho Avalon – MM Master của Frame Buffer

Tham số	Giá trị	Mô tả
Use separate clocks for the Avalon-MM master interfaces	On / Off	Bật để sử dụng clock riêng cho giao diện Avalon – MM master.
Avalon-MM master ports width	16–256, Default = 64	Chọn độ rộng bus dữ liệu truy cập vào bộ nhớ ngoài.
Write-only master interface FIFO depth	16–1024, Default = 64	Chọn độ sâu FIFO cho giao diện Avalon – MM chỉ ghi.
Write-only master interface burst target	2–256, Default = 32	Chọn mục tiêu burst dữ liệu cho giao diện Avalon – MM chỉ ghi
Read-only master interface FIFO depth	16–1024, Default = 64	Chọn độ sâu FIFO cho giao diện Avalon – MM chỉ đọc.
Read-only master interface burst target	2–256, Default = 32	Chọn mục tiêu burst dữ liệu cho giao diện Avalon – MM chỉ đọc
Base address of frame buffers	Một giá trị 32 bit. Default: 0x00000000	Đặt địa chỉ cơ sở của các bộ buffer trên bộ nhớ ngoài dùng để đệm dữ liệu.
Align read/write bursts with burst boundaries	On / Off	Bật để tránh đọc và ghi cùng một vị trí dẫn đến truy cập vượt giới hạn bộ nhớ.

Chi tiết về các tín hiệu và những thanh ghi điều khiển của IP Frame Buffer có thể tham khảo mục “Frame Buffer IP Cores” trong tài liệu [6].

4.3.4 IP Clocked Video Output



Hình 4.12 Block diagram IP Core Clocked Video Output

IP Clocked Video Output chuyển đổi video ở định dạng Avalon – ST sang các định dạng video tiêu chuẩn như BT656 hay VGA (gồm các tín hiệu R, G, B, HSync, VSync). Đầu vào của IP Core (din) là một giao diện Avalon – ST, video ở định dạng Avalon – ST sẽ đi vào IP Core từ cổng din, sau khi chuyển đổi hoàn thành, video định dạng tiêu chuẩn như BT656 hay VGA sẽ được xuất ra cổng clocked video theo giao diện Avalon Conduit. Video qua chuyển đổi này có thể đưa vào các bộ giải mã tương ứng như VGA để giải mã và hiển thị ra màn hình.

Bảng 4.7 Chi tiết các tham số định dạng dữ liệu khung hình CVO

Tham số	Giá trị	Mô tả
Image width / Active pixels	32 – 65536 Default = 1920	Chỉ định chiều rộng của khung hình bằng cách chọn số lượng active pixels.
Image height / Active lines	32 – 65536 Default = 1080	Chỉ định chiều cao của khung hình bằng cách chọn số lượng active lines.
Bits per pixel per color plane	4 – 20 Default = 8	Lựa chọn số lượng bit của mỗi kênh màu cho mọi pixel.
Number of color planes	1 – 4 Default = 3	Lựa chọn số lượng kênh màu sử dụng.
Color plane transmission format	Sequence hoặc Parallel	Lựa chọn truyền các kênh màu nối tiếp (Sequence) hoặc song song (Parallel).
Allow output of color planes in sequence	On / Off	Bật nếu muốn cho phép chuyển đổi run-time giữa các định dạng tuần tự (ví dụ NTSC) và các định dạng truyền mặt phẳng màu song song (ví dụ 1080p).
Interlaced video	On / Off	Bật nếu muốn sử dụng video được trộn với nhau.

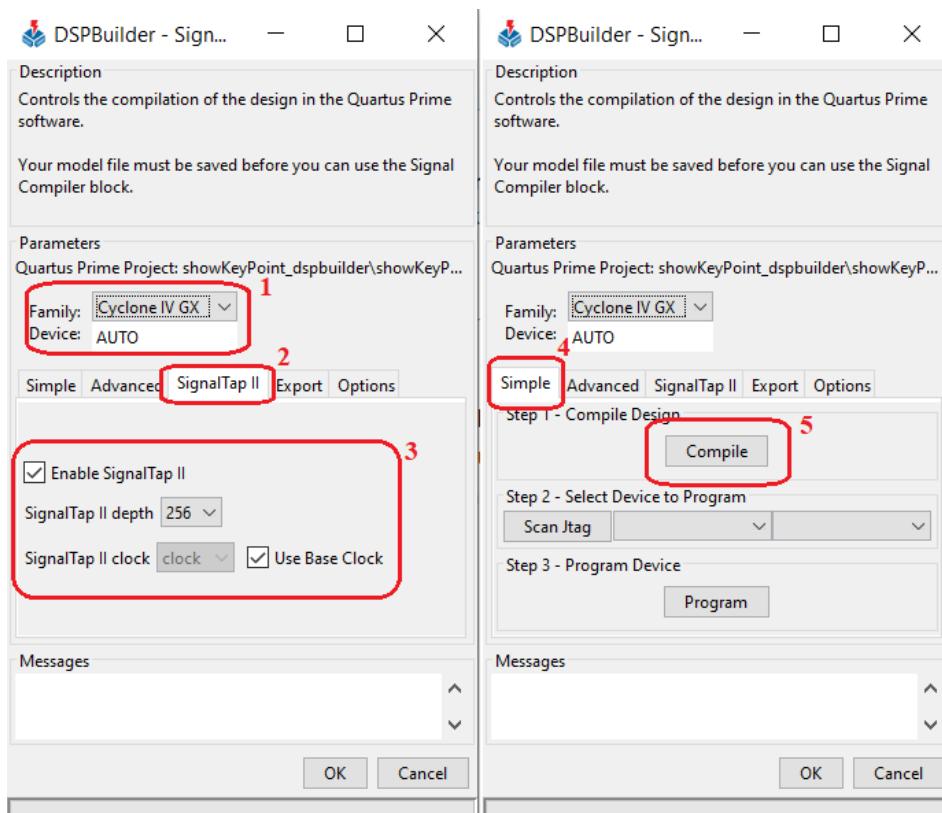
Chi tiết về các tham số còn lại và các thông tin khác có thể tham khảo mục “Clocked Video Interface IP Cores” trong tài liệu [6].

4.4 Triển khai hệ thống trên nửa FPGA

Triển khai hệ thống trên nửa FPGA của kit DE2i-150 là việc cấu hình chip FPGA Cyclone IV GX theo thiết kế hệ thống nêu ở mục 4.1. Việc triển khai IP Core được thiết kế và mô phỏng thành công trên MATLAB lên FPGA cần các phần mềm, công cụ hỗ trợ mà nhà sản xuất chip FPGA cung cấp. Với kit DE2i-150, ta sử dụng phần mềm Quartus và công cụ Qsys trên đó.

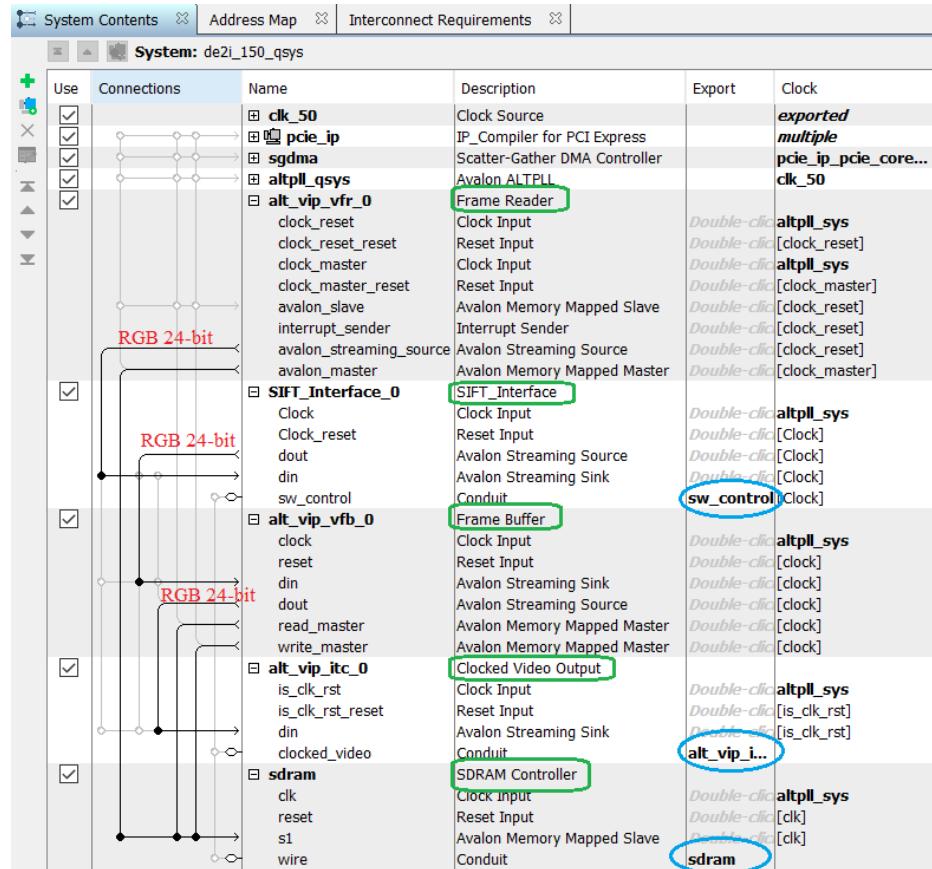
Để bắt đầu, ta cần một project Quartus được cấu hình tương thích với chip FPGA Cyclone IV GX. Đồ án này lựa chọn việc tái sử dụng project *PCIE_Display* mà Altera cung cấp trong CD ROM kèm theo kit DE2i-150, xem thêm mục **6-7 Demo: VGA Display** trong [7].

Với công cụ Qsys, ta có thể tạo mới một IP Core từ code HDL và kết nối nó với các IP Core tự tạo khác hoặc các IP Core được cung cấp sẵn. Như đã trình bày, các IP Core trong đồ án này được thiết kế bằng công cụ DSP Builder, vì vậy, ta cần sinh code HDL của thiết kế và dùng nó tạo IP Core trong Qsys. Khi sử dụng phiên bản DSP Builder và Quartus tương thích với nhau thì khi sinh code HDL của thiết kế, ta có thể trực tiếp tạo ra IP Core tương ứng trong Qsys bằng cách đặt tệp thiết kế vào trong một thư mục con của thư mục project rồi mới sinh code HDL.



Hình 4.13 Sinh IP Core sử dụng Qsys từ thiết kế trên DSP Builder

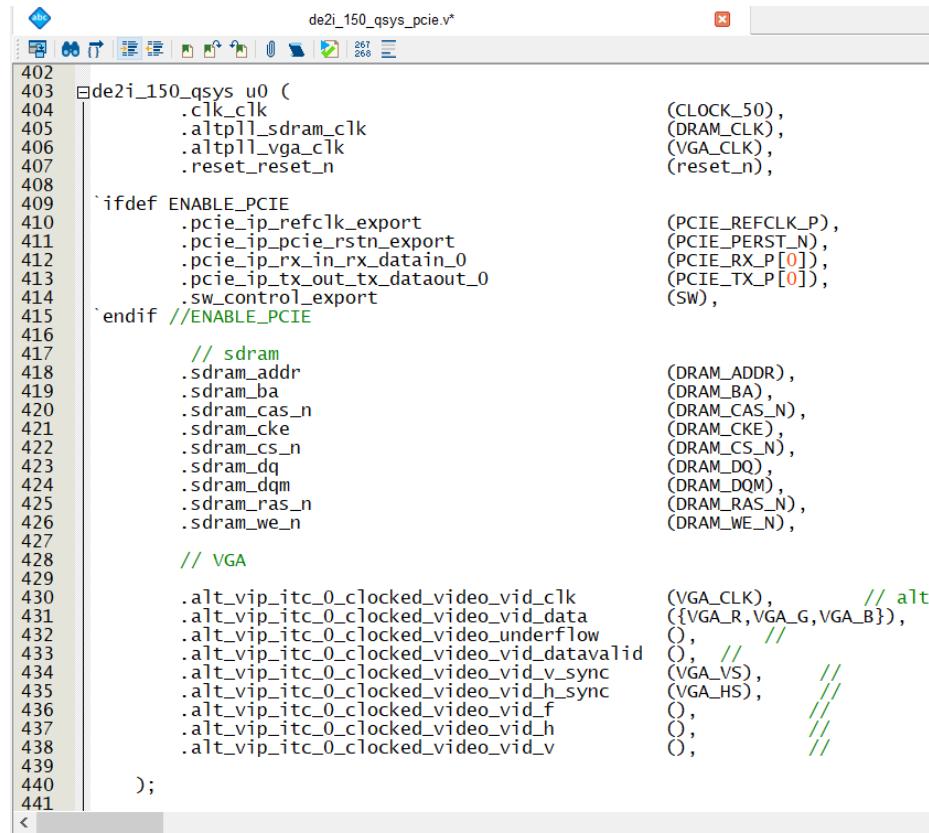
Tiếp theo, ta sẽ chỉnh sửa project này để triển khai hệ thống đã xây dựng ở trên. Trong công cụ Qsys, ta sẽ thêm IP Core vừa sinh ra từ thiết kế, sau đó điều chỉnh các kết nối giữa các IP Core, thiết lập các tham số, ... rồi lưu lại sửa đổi, tiến hành sinh code HDL của component Qsys mới này.



Hình 4.14 User Logic IP Core trong Qsys

Hình 4.14 cho thấy kết nối giữa các IP Core xử lý ảnh/video sử dụng giao diện Avalon – ST trong Qsys. Các vị trí tín hiệu đánh dấu bằng hình elip màu xanh dương là các tín hiệu kết nối với ngoại vi sử dụng giao diện Avalon Conduit, tên các IP Core được đánh dấu bởi các hình chữ nhật màu xanh lá. Chi tiết về kết nối toàn bộ trên Qsys, các tham số và sơ đồ khối có trong phụ lục trang 82.

Trở lại Quartus, ta chỉnh sửa lại các kết nối ngoại vi với component Qsys trong file top-level của project như gắn pin switch cho cổng sw_control, thêm các file HDL của các IP Core tự thiết kế vào trong project bằng code Verilog.



```

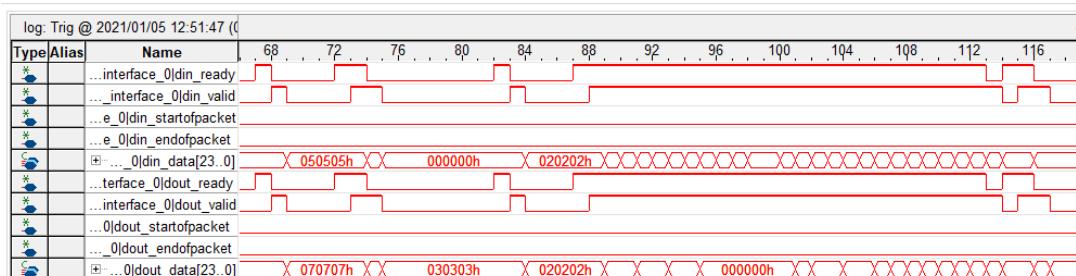
402
403     de2i_150_qsys_u0 (
404         .clk_clk                (CLOCK_50),
405         .altpll_sdram_clk      (DRAM_CLK),
406         .altpll_vga_clk        (VGA_CLK),
407         .reset_reset_n         (reset_n),
408
409     `ifdef ENABLE_PCIE
410         .pcie_ip_refclk_export (PCIE_REFCLK_P),
411         .pcie_ip_pcierstn_export (PCIE_PERT_N),
412         .pcie_ip_rx_in_rx_datain_0 (PCIE_RX_P[0]),
413         .pcie_ip_tx_out_tx_dataout_0 (PCIE_TX_P[0]),
414         .sw_control_export      (SW),
415     `endif //ENABLE_PCIE
416
417         // sdram
418         .sdram_addr             (DRAM_ADDR),
419         .sdram_ba               (DRAM_BA),
420         .sdram_cas_n            (DRAM_CAS_N),
421         .sdram_cke              (DRAM_CKE),
422         .sdram_cs_n             (DRAM_CS_N),
423         .sdram_dq               (DRAM_DQ),
424         .sdram_dqm              (DRAM_DQM),
425         .sdram_ras_n            (DRAM_RAS_N),
426         .sdram_we_n             (DRAM_WE_N),
427
428         // VGA
429
430         .alt_vip_itc_0_clocked_video_vid_clk (VGA_CLK),
431         .alt_vip_itc_0_clocked_video_vid_data ({VGA_R,VGA_G,VGA_B}),
432         .alt_vip_itc_0_clocked_video_underflow (),
433         .alt_vip_itc_0_clocked_video_vid_datavalid (),
434         .alt_vip_itc_0_clocked_video_vid_v_sync (VGA_VS),
435         .alt_vip_itc_0_clocked_video_vid_h_sync (VGA_HS),
436         .alt_vip_itc_0_clocked_video_vid_f (),
437         .alt_vip_itc_0_clocked_video_vid_h (),
438         .alt_vip_itc_0_clocked_video_vid_v ()
439
440     );
441

```

Hình 4.15 Kết nối IO của hệ thống sinh ra từ Qsys với ngoại vi bằng Verilog

Sau khi hoàn thành các công việc này, ta tiến hành dịch (compile) project. Trong trường hợp thiết kế không xảy ra lỗi, compile thành công sẽ sinh ra file *.sof* được dùng để cấu hình chip FPGA.

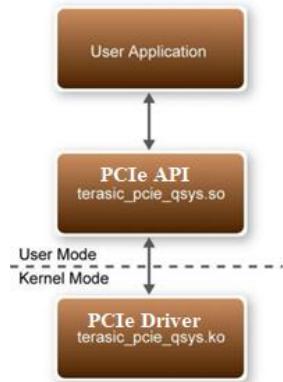
Khi cấu hình xong, thiết kế đang được chạy trên FPGA nhưng kết quả không như mong muốn, ta có thể sử dụng công cụ *SignalTap II Logic Analyzer* để bắt giá trị của các tín hiệu mong muốn trên hệ thống đang chạy, thông qua JTAG truyền trở về Qsys và hiển thị giản đồ sóng của các tín hiệu mong muốn. Tuy nhiên khi sử dụng công cụ này, cần cấu hình tham số tương ứng và phải dịch lại project.



Hình 4.16 Ví dụ về một giản đồ sóng hiển thị bởi SignalTap II Logic Analyzer

4.5 Triển khai hệ thống trên nửa PC

Triển khai hệ thống trên nửa PC của kit DE2i-150 là bao gồm việc cài đặt driver PCI Express, sử dụng PCIe API được cung cấp để lập trình một ứng dụng (sử dụng ngôn ngữ lập trình C) đơn giản thực hiện việc điều khiển truyền nhận dữ liệu dữ hai nửa PC và FPGA.



Hình 4.17 PCI Express Software Stack on Yocto

Nửa PC của kit DE2i-150 sử dụng trong đồ án được cài hệ điều hành Yocto Linux với phiên bản nhân linux (kernel version) là 3.0.32 và được Altera cung cấp driver PCI Express cùng API tương ứng. Vì vậy công việc chủ yếu của phần này là tìm hiểu và sử dụng các hàm API được cung cấp để lập trình ứng dụng cho phía Host. Ứng dụng này sẽ thông qua PCI Express driver để ghi giá trị cho các thanh ghi cấu hình phần cứng trong nửa FPGA, gửi dữ liệu ảnh nguồn cần xử lý trong RAM của nửa PC tới SDRAM của nửa FPGA, đồng thời cũng đọc trả về dữ liệu ảnh và các dữ liệu thu được qua quá trình xử lý được hệ thống phần cứng trên nửa FPGA ghi vào SDRAM. Dưới đây là chi tiết về các hàm API được sử dụng trong đồ án.

Bảng 4.8 Chi tiết hàm PCIE_Open

PCIE_Open	
Mục đích	Mở một thẻ PCIe dựa trên các tham số.
Khuôn mẫu hàm	PCIE_HANDLE PCIE_Open(WORD wVendorID, WORD wDeviceID, WORD wCardIndex);
Tham số	wVendorID Chỉ định vendor ID (ID nhà sản xuất), là 0 nếu không sử dụng giá trị này. wDeviceID Chỉ định device ID (ID thiết bị), là 0 nếu không sử dụng giá trị này. wCardIndex Chỉ định Card index tương ứng với phần cứng thiết bị PCIe. Với phiên bản linux, cả 3 tham số nhận giá trị 0.
Giá trị trả về	Trả về một handle liên kết với thẻ PCIe được mở. Nếu kết quả là một số dương tức mở thành công. Trả về giá trị 0 có nghĩa là không kết nối được với thẻ PCIe mong muốn. Giá trị này sẽ được dùng làm tham số cho các hàm khác sẽ trình bày phía sau.

Bảng 4.9 Chi tiết hàm PCIE_Close

PCIE_Close		
Mục đích	Đóng handle liên kết với thẻ PCIe.	
Khuôn mẫu hàm	void PCIE_Close(PCIE_HANDLE hPCIE);	
Tham số	hPCIE	Một PCIe handle trả về từ hàm PCIE_Open.
Giá trị trả về	Không có giá trị trả về.	

Bảng 4.10 Chi tiết hàm PCIE_Read32

PCIE_Read32		
Mục đích	Đọc 32-bit dữ liệu từ FPGA.	
Khuôn mẫu hàm	bool PCIE_Read32(PCIE_HANDLE hPCIE, PCIE_BAR PcieBar, PCIE_ADDRESS PcieAddress, DWORD * pdwData);	
	hPCIE	Handle trả về bởi hàm PCIE_Open.
	PcieBar	Chỉ định BAR truyền dữ liệu.
Tham số	PcieAddress	Địa chỉ vùng nhớ trên FPGA mà dữ liệu sẽ được đọc từ đó.
	pdwData	Một bộ đệm dùng truy suất 32-bit dữ liệu.
Giá trị trả về	TRUE nếu đọc thành công, ngược lại trả về FALSE.	

Bảng 4.11 Chi tiết hàm PCIE_Write32

PCIE_Write32		
Mục đích	Ghi 32-bit dữ liệu vào FPGA.	
Khuôn mẫu hàm	bool PCIE_Write32(PCIE_HANDLE hPCIE, PCIE_BAR PcieBar, PCIE_ADDRESS PcieAddress, DWORD dwData);	
	hPCIE	Handle trả về bởi hàm PCIE_Open.
	PcieBar	Chỉ định BAR truyền dữ liệu.
Tham số	PcieAddress	Địa chỉ vùng nhớ trên FPGA mà dữ liệu sẽ được ghi vào đó.
	pdwData	Chứa 32-bit dữ liệu sẽ được ghi vào FPGA.
Giá trị trả về	TRUE nếu ghi thành công, ngược lại trả về FALSE.	

Bảng 4.12 Chi tiết hàm PCIE_DmaRead

PCIE_DmaRead										
Mục đích	Đọc dữ liệu từ bộ nhớ – mapped của FPGA qua DMA.									
Khuôn mẫu hàm	<pre>bool PCIE_DmaRead(PCIE_HANDLE hPCIE, PCIE_LOCAL_ADDRESS LocalAddress, void *pBuffer, DWORD dwBufSize);</pre>									
Tham số	<table border="1"> <tr> <td>hPCIE</td><td>Handle trả về bởi hàm PCIE_Open.</td></tr> <tr> <td>LocalAddress</td><td>Địa chỉ vùng nhớ – mapped muốn đọc dữ liệu trên FPGA.</td></tr> <tr> <td>pBuffer</td><td>Một con trỏ trả tới một bộ đệm dùng để nhận dữ liệu đọc về từ FPGA và không được nhỏ hơn tham số dwBufSize.</td></tr> <tr> <td>dwBufSize</td><td>Số byte dữ liệu sẽ đọc được từ FPGA.</td></tr> </table>		hPCIE	Handle trả về bởi hàm PCIE_Open.	LocalAddress	Địa chỉ vùng nhớ – mapped muốn đọc dữ liệu trên FPGA.	pBuffer	Một con trỏ trả tới một bộ đệm dùng để nhận dữ liệu đọc về từ FPGA và không được nhỏ hơn tham số dwBufSize.	dwBufSize	Số byte dữ liệu sẽ đọc được từ FPGA.
hPCIE	Handle trả về bởi hàm PCIE_Open.									
LocalAddress	Địa chỉ vùng nhớ – mapped muốn đọc dữ liệu trên FPGA.									
pBuffer	Một con trỏ trả tới một bộ đệm dùng để nhận dữ liệu đọc về từ FPGA và không được nhỏ hơn tham số dwBufSize.									
dwBufSize	Số byte dữ liệu sẽ đọc được từ FPGA.									
Giá trị trả về	Trả về TRUE nếu đọc dữ liệu thành công, ngược lại trả về FALSE.									

Bảng 4.13 Chi tiết hàm PCIE_DmaWrite

PCIE_DmaWrite										
Mục đích	Ghi dữ liệu vào bộ nhớ – mapped của FPGA qua DMA.									
Khuôn mẫu hàm	<pre>bool PCIE_DmaWrite(PCIE_HANDLE hPCIE, PCIE_LOCAL_ADDRESS LocalAddress, void *pData, DWORD dwDataSize);</pre>									
Tham số	<table border="1"> <tr> <td>hPCIE</td><td>Handle trả về bởi hàm PCIE_Open.</td></tr> <tr> <td>LocalAddress</td><td>Địa chỉ vùng nhớ – mapped muốn ghi dữ liệu trên FPGA.</td></tr> <tr> <td>pData</td><td>Một con trỏ trả tới một bộ đệm chứa dữ liệu sẽ ghi vào từ FPGA.</td></tr> <tr> <td>dwDataSize</td><td>Số byte dữ liệu sẽ ghi vào FPGA.</td></tr> </table>		hPCIE	Handle trả về bởi hàm PCIE_Open.	LocalAddress	Địa chỉ vùng nhớ – mapped muốn ghi dữ liệu trên FPGA.	pData	Một con trỏ trả tới một bộ đệm chứa dữ liệu sẽ ghi vào từ FPGA.	dwDataSize	Số byte dữ liệu sẽ ghi vào FPGA.
hPCIE	Handle trả về bởi hàm PCIE_Open.									
LocalAddress	Địa chỉ vùng nhớ – mapped muốn ghi dữ liệu trên FPGA.									
pData	Một con trỏ trả tới một bộ đệm chứa dữ liệu sẽ ghi vào từ FPGA.									
dwDataSize	Số byte dữ liệu sẽ ghi vào FPGA.									
Giá trị trả về	Trả về TRUE nếu ghi dữ liệu thành công, ngược lại trả về FALSE.									

CHƯƠNG 5. KẾT QUẢ THỬ NGHIỆM

5.1 Quy trình và kịch bản thử nghiệm

5.1.1 Quy trình thử nghiệm

Quy trình thử nghiệm hệ thống trên kit DE2i-150 bao gồm những bước sau:

- Bước 1: Tạo một thư mục con trong thư mục Project Quatus và copy file thiết kế DSP vào đây, sau đó dịch thiết kế để sinh ra IP Core dùng trong Qsys.
- Bước 2: Mở Qsys của project, thêm IP Core vừa tạo, chỉnh sửa kết nối. Sau đó lưu thiết kế và sinh code HDL (Generate HDL).
- Bước 3: Đóng Qsys, trở lại Quartus, chỉnh sửa file top-level để gắn ngoại vi thêm nếu có (ví dụ như switch cho SIFT), thêm file *<tên thiết kế>.qip* được sinh ra trong thư mục tạo ở bước 1.
- Bước 4a: Dịch (compile) project vừa chỉnh sửa
- Bước 4b: Cấu hình tham số cho công cụ *SignalTap II Logic Analyzer* trong Quartus để hiển thị giản đồ sóng của các tín hiệu cần thiết có trong thiết kế vừa dịch. Bước này cần file SOF sinh ra ở bước 4a làm tham số chính. Sau khi thiết lập các tín hiệu cần vẽ giản đồ sóng và các tham số khác, tiến hành dịch lại (recompile) project.
- Bước 5: Dùng công cụ Programmer trong Quartus để nạp file cấu hình (file SOF) FPGA được sinh ra sau bước 4a (nếu không cần xem giản đồ sóng) hoặc ở bước 4b (nếu muốn xem giản đồ sóng) lên chip FPGA Cyclone IV GX.
- Bước 6: Khởi động lại hệ điều hành trên nửa PC để tránh xung đột đọc/ghi bộ nhớ.
- Bước 7: Cài PCI Express Driver cho Yocto (insmod).
- Bước 8: Chạy ứng dụng đã viết và dịch sẵn trên Yocto và quan sát kết quả trên màn hình hiển thị VGA, giản đồ sóng trên Quartus và dữ liệu đọc về thông qua PCIe.

5.1.2 Kịch bản thử nghiệm

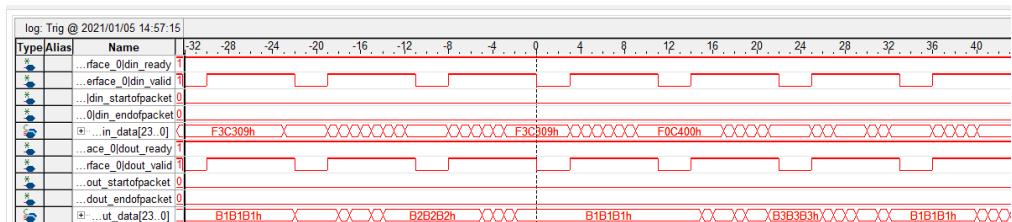
Dữ liệu là ảnh/video màu (định dạng RGB 24-bit) kích thước cố định 640x480 được ứng dụng truyền từ RAM của nửa PC đến SDRAM của nửa FPGA thông qua PCIe. Hệ thống phần cứng xử lý trên FPGA sẽ thử nghiệm lần lượt các bộ thuật toán xử lý ảnh thành phần của thuật toán SIFT (như chuyển từ ảnh màu sang ảnh xám, nhân chập, tìm và hiển thị điểm hấp dẫn, ...) song song với quá trình thiết kế và mô phỏng trên MATLAB để tiến hành chỉnh sửa và kiểm soát hoạt động của từng module. Dữ liệu xử lý qua các IP Core thuật toán trên đều được hiển thị trực tiếp trên màn hình thông qua VGA cũng như lưu lại trong SDRAM để đọc trở về phía PC phục vụ cho phân tích, đánh giá.

5.2 Kết quả thử nghiệm IP Core rgb2gray

IP Core rgb2gray là IP Core thuật toán thực hiện việc chuyển ảnh từ ảnh màu RGB sang ảnh đa mức xám để sử dụng trong thuật toán SIFT.



Hình 5.1 Kết quả thử nghiệm IP Core *rgb2gray*



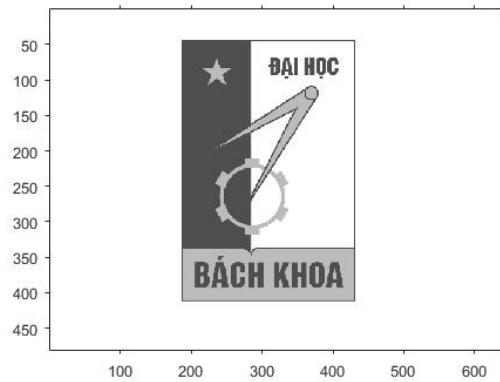
Hình 5.2 Một đoạn giản đồ sóng I/O của IP Core *rgb2gray*

Giản đồ sóng trên thể hiện một giá trị của các tín hiệu đầu vào (din – Avalon Streaming Sink) và đầu ra (dout – Avalon Streaming Source) của IP Core.

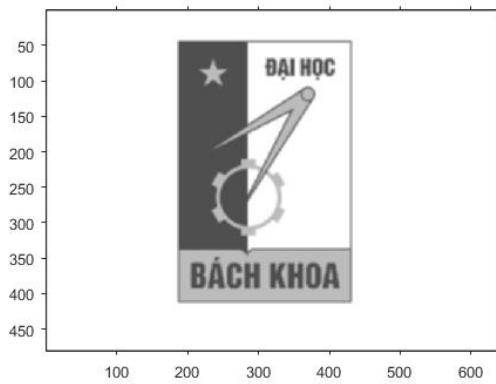
Với mục đích hiển thị kết quả thử nghiệm IP Core *rgb2gray* trên màn hình thông qua đầu ra VGA của nửa FPGA nên giá trị đa mức xám là 8-bit sẽ được lặp 3 lần, tức là đầu ra sẽ là ảnh màu 24-bit với giá trị các kênh màu đều là giá trị đa mức xám. Có thể thấy rõ sự lặp lại này ở trên tín hiệu dout_data[23..0] trên giản đồ sóng.

5.3 Kết quả thử nghiệm IP Core Gaussian Filter

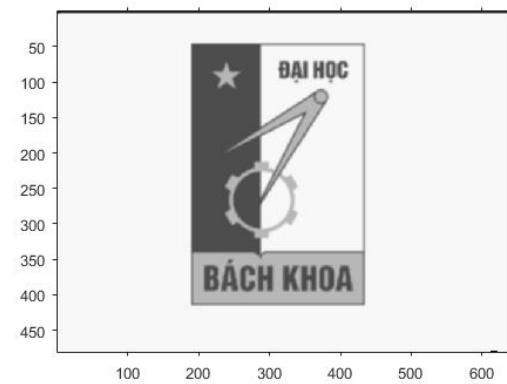
IP Core Gaussian Filter là IP Core thuật toán thực hiện phép nhân chập măt nạ Gaussian 5x5 với ảnh đa mức xám (ảnh đầu ra của IP Core *rgb2gray*).



A. Ảnh đa mức xám thu được từ IP Core rgb2gray



B. Kết quả nhân chập ảnh A với mặt nạ gaussian
kích thước 5x5, sigma = 2.0 trên MATLAB



C. Kết quả thu được khi cho ảnh A đi qua IP Core
Gaussian Filter, kích thước mặt nạ 5x5, sigma = 2.0

Hình 5.3 Kết quả thử nghiệm IP Core Gaussian Filter

Hình 5.3 thể hiện kết quả thử nghiệm IP Core Gaussian Filter có kích thước mặt nạ 5x5, $\sigma = 2.0$. Từ kết quả trên thu được, đối chiếu với kết quả thực hiện thuật toán trên MATLAB ta thấy ảnh được làm mờ đi khá tốt. Tuy nhiên có sự khác biệt giữa tương đối rõ ràng khi so sánh. Thứ nhất, ảnh kết quả thực hiện trên kit DE2i-150 có một vạch đen ở phía trên cùng ảnh trong khi ảnh thực hiện ở MATLAB và cả ảnh đa mức xám đầu vào cũng không có. Vạch đen này là 2 hàng của ảnh, giá trị thực của chúng không phải là 0 mà là tăng dần, tuy nhiên độ tăng không quá lớn nên không có cảm nhận quá trực quan. Sở dĩ có sự khác nhau này là do trong thiết kế bộ *line_buffer* sẽ không đủ dữ liệu để tính toán cho 2 dòng đầu mà mặc định là giá trị 0 (do bị reset). Tuy nhiên, xác suất để điểm hắp dẫn nằm ở vùng này rất thấp nên chấp nhận bị mất thông tin của vùng này để việc thiết kế đơn giản hơn.

Khác nhau thứ hai dễ nhìn ra là nền của ảnh thực hiện trên kit DE2i-150 tối màu hơn so với hai ảnh còn lại. Lý do dẫn đến sự khác nhau này là do việc làm tròn số. Làm tròn thứ nhất xảy ra ở việc sử dụng số *Fractional* để biểu diễn các số thực dấu phẩy động trong các hệ số trong mặt nạ gaussian. Việc làm tròn thứ hai xảy ra ở làm tròn kết quả sau phép cộng. Kết quả thực sự sau IP Core gauss là một số *Fractional* với 9-bit cho phần nguyên và 16-bit phần thập phân. Tương tự với IP Core *rgb2gray*, để hiển thị trên màn hình nên làm tròn kết quả thành số nguyên 8-bit và lặp 3 lần. Chính vì thế những giá trị này còn 248 thay vì là 255.

Nhưng điều này không ảnh hưởng đến tính chính xác của thuật toán SIFT do việc làm tròn này là xảy ra trên toàn ảnh, sẽ bị triệt tiêu trong phép tính DOG phía sau.



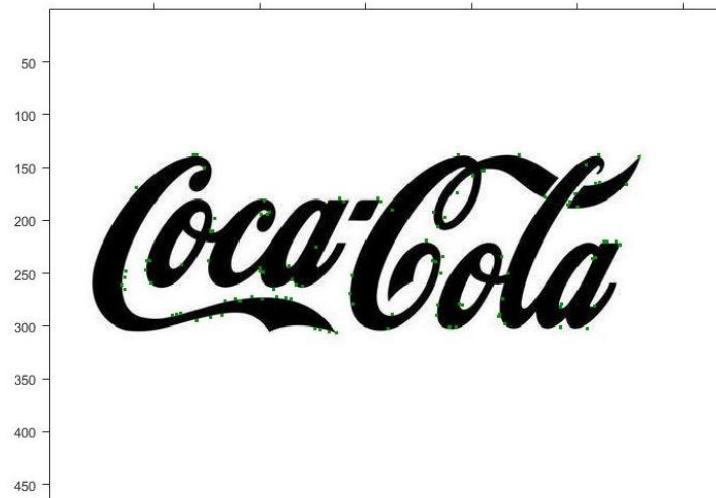
Hình 5.4 Một đoạn ngắn đồ sóng I/O của IP Core Gaussian Filter

5.4 Kết quả thử nghiệm IP Core SIFT Detector

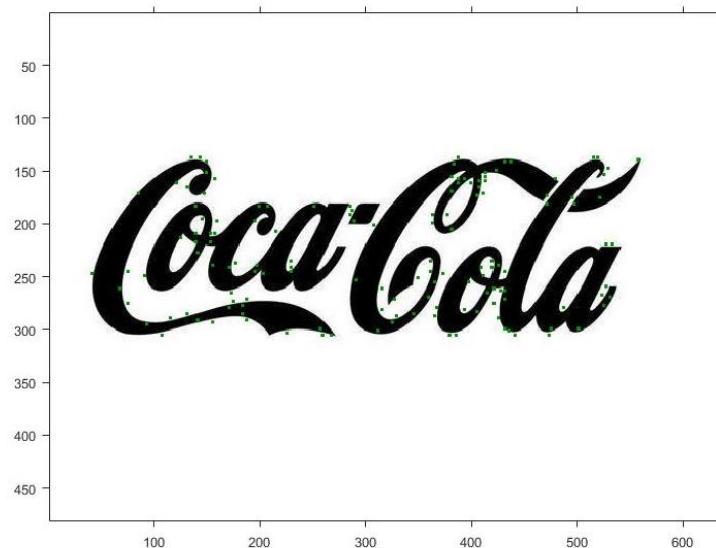
IP Core SIFT Detector là IP Core thuật toán thực hiện phát hiện và đánh dấu các điểm hấp dẫn tìm được. IP Core này có thể dùng switch để cấu hình chạy với 1 octave hay 2 octave, octave thứ nhất ứng với kích thước ảnh gốc của ảnh đầu vào (640x480), octave thứ hai ứng với kích thước bằng một nửa kích thước ảnh octave thứ nhất (320x240). Thiết kế được triển khai thử nghiệm này dùng 2-bit ở phần thập phân của quá trình tính DOG (Hình 3.28). Trong kết quả thử nghiệm, điểm hấp dẫn được đánh dấu trên ảnh bằng các hình vuông màu xanh lá, kích thước 3x3, tâm là điểm hấp dẫn.



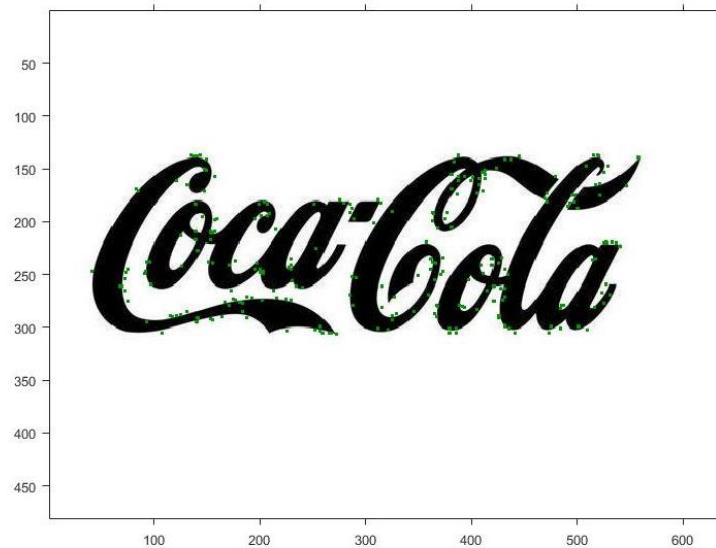
Hình 5.5 Kết quả IP Core SIFT Detector với logo BKHN



A. Kết quả khi chỉ dùng 1 octave ứng với ảnh kích thước 640x480



B. Kết quả khi chỉ dùng 1 octave ứng với ảnh kích thước 320x240



C. Kết quả khi dùng cả 2 octave trên

Hình 5.6 Kết quả IP Core SIFT Detector với logo Coca Cola

5.5 Những khó khăn gặp phải khi thực hiện đồ án và giải pháp

Do đặc thù về mặt kỹ thuật khi phát triển thuật toán trên nền tảng FPGA cũng như kiến thức và kinh nghiệm không đủ, chúng em gặp phải khá nhiều khó khăn trong quá trình thực hiện đồ án. Khó khăn lớn nhất chúng em gặp phải là vấn đề gỡ lỗi thiết kế trên FPGA. Điều này tiêu tốn rất nhiều thời gian khi mô phỏng thành công trên MATLAB nhưng khi triển khai trên kit lại không thu được kết quả mong muốn.

Về mặt thời gian khi triển khai, khi thiết kế của thuật toán càng hoàn thiện dần thì thời gian để triển khai thuật toán trên kit lại càng tăng mạnh, điều này cộng với việc khó gỡ lỗi khiến cho thời gian sửa chữa và triển khai thử nghiệm tốn nhiều thời gian.

Bảng 5.1 Kết quả triển khai thử nghiệm các IP Core trên kit DE2i-150

IP Core	Mục đích/tác dụng	Kết quả triển khai
rgb2gray	Chuyển frame/ảnh màu RBG thành ảnh đa mức xám	Thành công
Gaussian Filter	Thực hiện phép nhân chập frame/ảnh với mặt nạ gaussian	Thành công
SIFT Detector	Phát hiện và đánh dấu điểm hấp dẫn trên frame/ảnh (2 octave)	Thành công
SIFT	Phát hiện, đánh dấu trên frame/ảnh và tính toán bộ mô tả cho các điểm hấp dẫn (1 octave)	Thành công
SIFT Matching	Phát hiện, tính toán bộ mô tả của các điểm hấp dẫn trong các frame/ảnh và so khớp chúng với bộ mô tả có sẵn. (1 octave)	Chưa thành công

Bảng 5.2 Thống kê tài nguyên sử dụng và thời gian triển khai trên FPGA

		Total logic elements	Total registers	Total memory bits	Thời gian triển khai
rgb2gray	IP Core	1,174	1,016	13,978	~ 10 phút
	HTTN	15,653(10%)	11,255	596,956(9%)	
Gaussian Filter	IP Core	3,004	2,628	44,048	~15 phút
	HTTN	17,483(12%)	12,867	627,026(9%)	
SIFT Detector	IP Core	16,672	14,813	367,759	~30 phút
	HTTN	31,151(21%)	25,052	950,557(14%)	
SIFT	IP Core	15,379	10,576	870,203	~40 phút
	HTTN	29,858(19%)	20,815	1,453,181(22%)	
SIFT Matching	IP Core	25,711	16,813	3,735,055	~60 phút
	HTTN	40,190(27%)	27,206	4,318,033(65%)	

Ghi chú: HTTN là hệ thống thử nghiệm IP Core tương ứng

Bảng 5.2 thống kê tài nguyên sử dụng trên chip FPGA và thời gian để triển khai trên kit một lần. Thời gian triển khai này bao gồm thời gian thực hiện dịch thiết kế trên DSP Builder thành Qsys IP Core, thời gian chỉnh sửa và sinh mã HDL (generate HDL) cho thiết kế hệ thống trên Qsys, thời gian dịch project (compile project) sinh tệp cấu hình FPGA (SOF file). Nếu sử dụng công cụ **SignalTap II Logic Analyzer** để hiển thị giản đồ sóng thì cần dịch lại project (recompile), tức cần thêm 1 khoảng thời gian từ 5 đến 30 phút tùy thiết kế.

Giải pháp khắc phục đề ra là hoàn thiện thiết kế trên DSP Builder và mô phỏng trên MATLAB để loại bỏ hầu hết các lỗi dễ phát hiện trước khi thử nghiệm trên kit. Tuy nhiên, theo độ phức tạp của thiết kế tăng lên, việc mô phỏng cũng gặp khó khăn và tốn nhiều thời gian, càng về cuối, khi IP Core dẫn hoàn thiện, thời gian mô phỏng thậm chí còn dài hơn thời gian triển khai trên kit. Để giải quyết vấn đề này, thiết kế được chia thành các module để đơn giản khi mô phỏng cũng như thử nghiệm trên kit.

Một khó khăn khác là tài liệu về các IP Core trong DSP Builder không quá chi tiết, cho kết quả hoạt động khác nhau khi mô phỏng trên MATLAB và khi thực nghiệm trên kit. Ví dụ như các khối IP Core Delay được dùng rất nhiều để tạo các bộ line buffer khi không có tín hiệu reset, mô phỏng trên MATLAB sẽ cho giá trị 0 ở đầu ra khi chưa có dữ liệu qua nó, nhưng ở trên kit lại cho các giá trị rác. Hay IP Core Dual Port Ram dùng để lưu trữ các bộ mô tả của các điểm hấp dẫn trước khi so khớp cần delay một khoảng thời gian khi đọc/ghi, tài liệu cho số liệu hoạt động tốt trên mô phỏng nhưng khi thử nghiệm trên kit lại thất bại, lỗi này rất khó phát hiện và tốn nhiều thời gian để thử nghiệm thực tế mới có thể sửa được. Để giải quyết vấn đề này, mỗi khi thiết kế triển khai trên kit và mô phỏng bị sai lệch, sẽ tiến hành kiểm tra lại các IP Core có nhớ, các IP Core mới được dùng sẽ được tách riêng để tạo test đơn giản, kiểm tra hoạt động để giảm thời gian test trên kit.

Ngoài những khó khăn nêu trên, vấn đề về môi trường phát triển và triển khai (FPGA) ảnh hưởng đến tiến độ khá nhiều. Tài nguyên trên FPGA có hạn nên không thể thiết kế phần so khớp song song bằng cách nhân số lượng các bộ thực hiện so khớp lên, nhưng môi trường phần cứng (FPGA) lại thích hợp xử lý song song, điều này là khó khăn lớn nhất dẫn đến việc so khớp trực tiếp trên kit chưa thành công.

CHƯƠNG 6. KẾT LUẬN

6.1 Kết luận

SIFT là một thuật toán trích suất đặc trưng ảnh được ứng dụng nhiều trong thực tế. Trong đồ án tốt nghiệp này, chúng em đã xây dựng được IP Core thực hiện thuật toán SIFT theo giao diện vào ra Avalon Streaming Video – chuẩn giao diện dùng trong các IP Core xử lý ảnh/video được cung cấp sẵn bởi Altera (Intel) trên MATLAB và mô phỏng thành công. Tuy nhiên do kinh nghiệm với các dự án thực tế trên FPGA còn thiếu cũng như thời gian thực hiện đồ án có hạn nên mới chỉ thành công triển khai phần phát hiện điểm hấp dẫn và tính được vector mô tả các điểm hấp dẫn ấy trên kit FPGA, còn phần so khớp các điểm hấp dẫn của hai ảnh với nhau vẫn chưa thể hoàn thành. Kết quả thu được trên kit FPGA tương tự với kết quả mô phỏng trên MATLAB và khá tốt.

Qua quá trình thực hiện đồ án, chúng em học được kỹ năng làm việc nhóm, kỹ năng lập kế hoạch và phân công công việc. Cùng với đó là các kỹ năng hữu ích khác như kỹ năng tìm kiếm tài liệu, kỹ năng tổng hợp, trình bày thông tin, kỹ năng ché bản, viết báo cáo, ...

6.2 Hướng phát triển đồ án trong tương lai

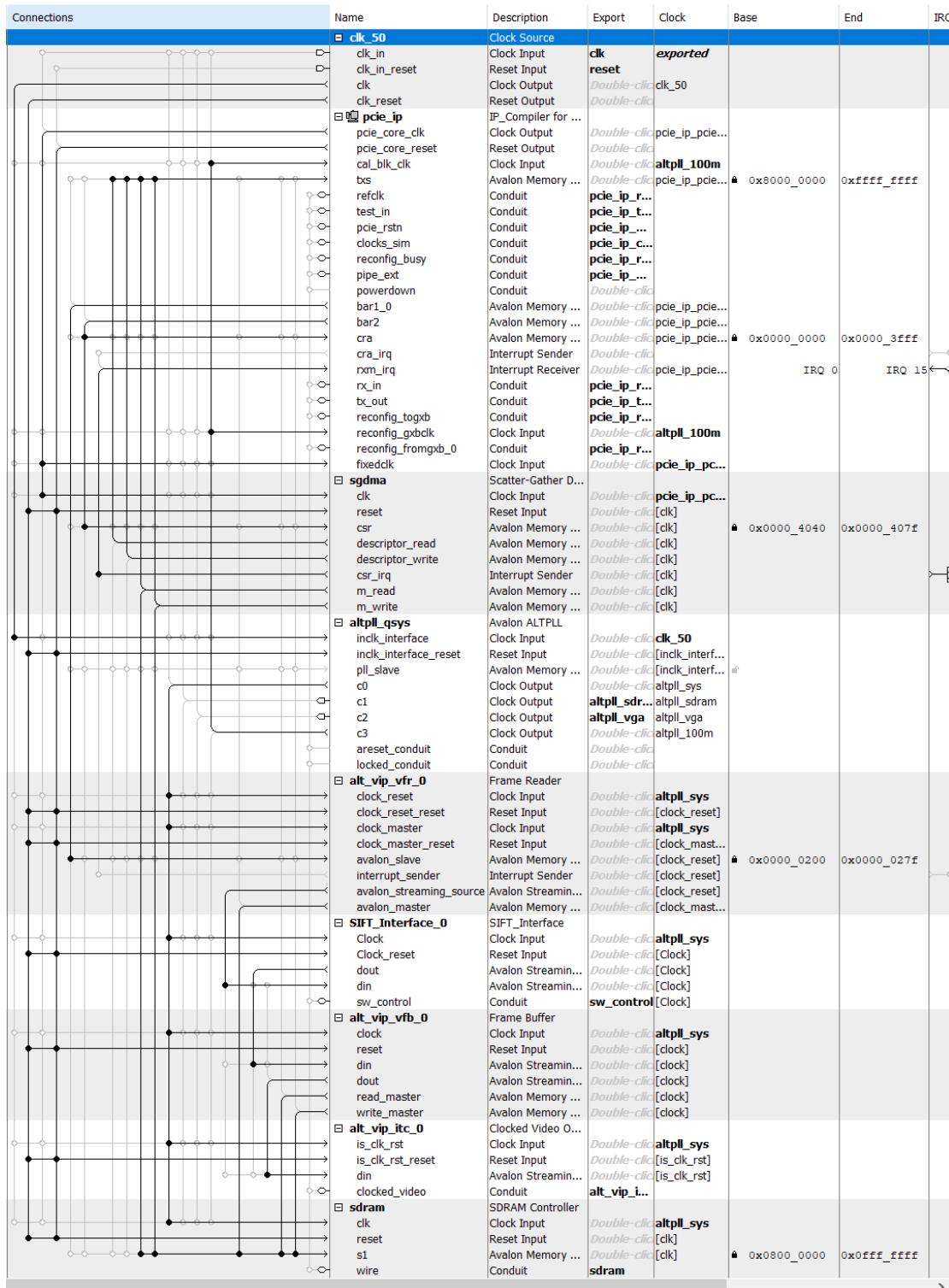
Trong quá trình thực hiện đồ án, chúng em nhận ra việc triển khai thuật toán SIFT trên FPGA với mục đích tăng tốc độ xử lý là khả thi. Với phần phát hiện điểm hấp dẫn và tính toán các vector đặc trưng của những điểm hấp dẫn ấy đã được triển khai thành công, kết quả có thể còn chưa tốt như phần mềm nhưng có thể cải thiện bằng cách gia tăng số lượng octave trong không gian tỉ lệ. Tuy nhiên, phần so khớp các điểm hấp dẫn của hai ảnh với nhau gặp khá nhiều khó khăn do kích thước các vector mô tả khá lớn và tài nguyên của chip FPGA là hữu hạn. Vì vậy, hướng phát triển của đồ án tương lai là biến FPGA thành một card tăng tốc thuật toán, trên đó, ta sẽ tối ưu phần phát hiện điểm hấp dẫn và vector mô tả của nó. Phần so khớp sẽ được thực hiện trên phần mềm trở nên đơn giản hơn. Và kết quả so khớp này có thể dùng để thực hiện nhiều ứng dụng như Around View Monitor, Video Stitching, ...

TÀI LIỆU THAM KHẢO

- [1] Intel, "Avalon Interface Specifications," 2020.
- [2] I. R. Otero and M. Delbracio, "Anatomy of the SIFT Medthod," 2014.
- [3] L. Yao, H. Feng, Y. Zhu, Z. Jiang, D. Zhao and W. Feng, "An architecture of optimised SIFT feature detection for an FPGA implementation of an image matcher," *Field-Programmable Technology*, 2009.
- [4] Intel, "Embedded Peripherals IP User Guide," 2020.
- [5] Altera, "IP Compiler for PCI Express User Guide," 2014.
- [6] Altera, "Video and Image Processing Suite User," 2016.
- [7] Terasic, "DE2i-150_FPGA_System_manual.pdf".

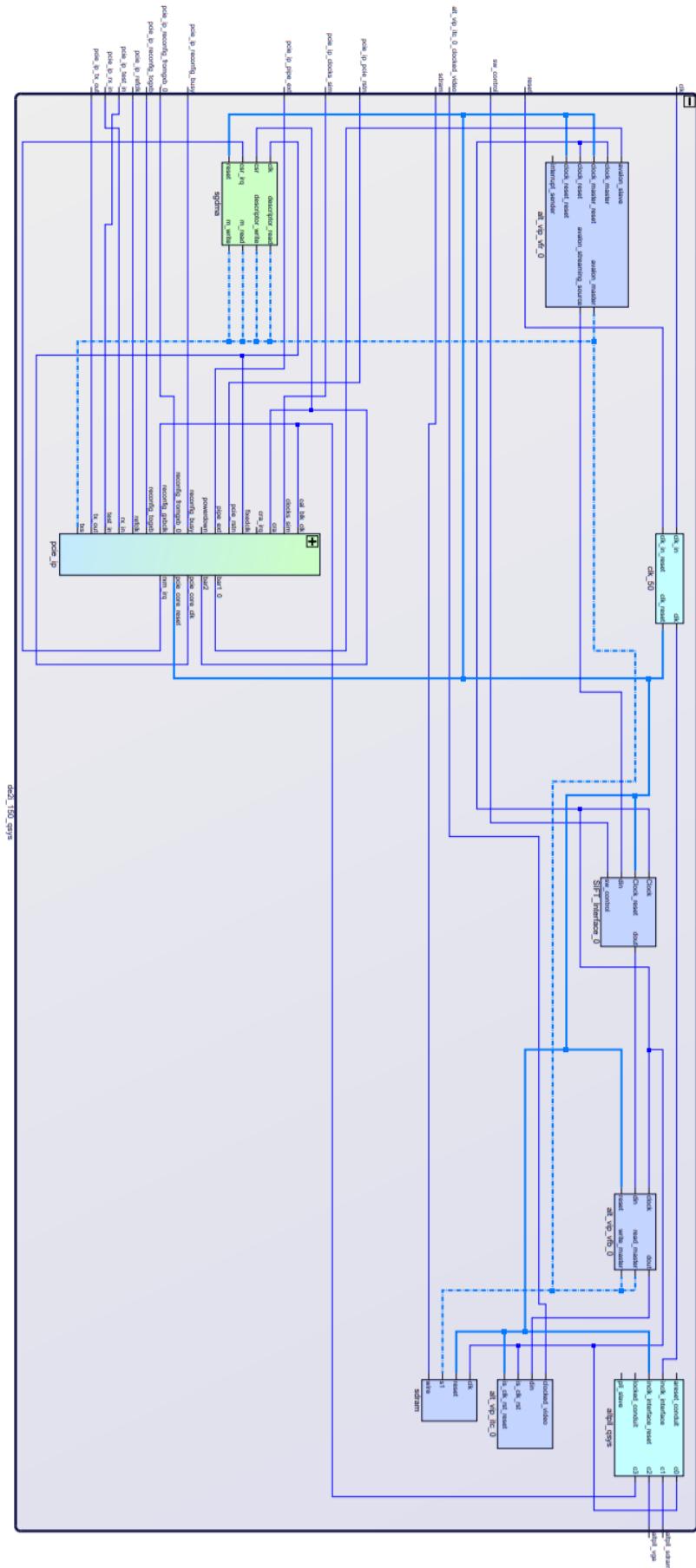
PHỤ LỤC

A1. Chi tiết thiết kế Qsys của hệ thống trên nửa FPGA



Hình 1 Hệ thống Qsys được triển khai thử nghiệm

Nghiên cứu, thiết kế và triển khai thuật toán SIFT trên FPGA



Hình 2 Schematic của hệ thống Qsys được triển khai thử nghiệm

Nghiên cứu, thiết kế và triển khai thuật toán SIFT trên FPGA

System Contents				Address Map	Interconnect Requirements
System: de2_150_qsys Path: clk_50					
alt_vip_vfr_0.avalon_slave	alt_vip_vfb_0.read_master	alt_vip_vfb_0.write_master	alt_vip_vfr_0.avalon_master	pcie_ip.bar1_0	0x0000 0200 - 0x0000 027f
altpll_qsys pll_slave					
pcie_ip.tbs					
pcie_ip.cra					
sdram.s1	0x0800 0000 - 0x0fff ffff	0x0800 0000 - 0x0fff ffff	0x0800 0000 - 0x0fff ffff		
sgdma.cs1					

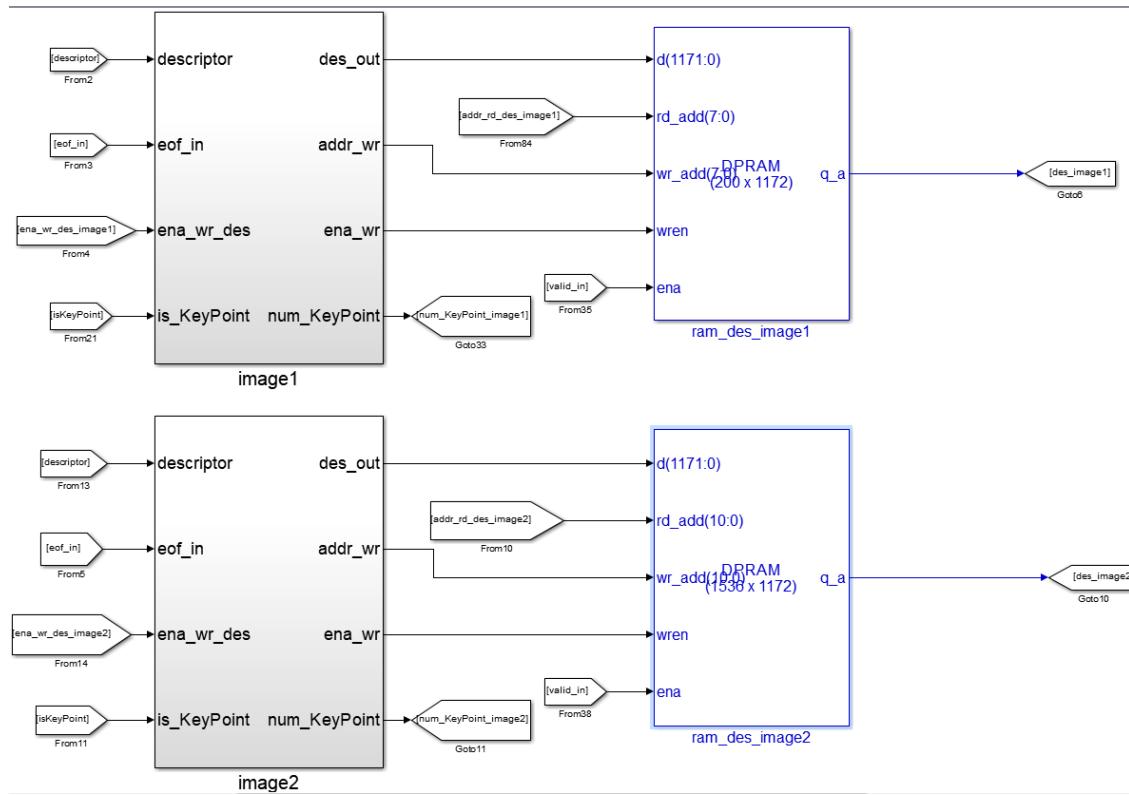
Hình 3 Bảng địa chỉ được cấu hình cho các IP Core trong Qsys

A1. Chi tiết thiết kế IP Core matching

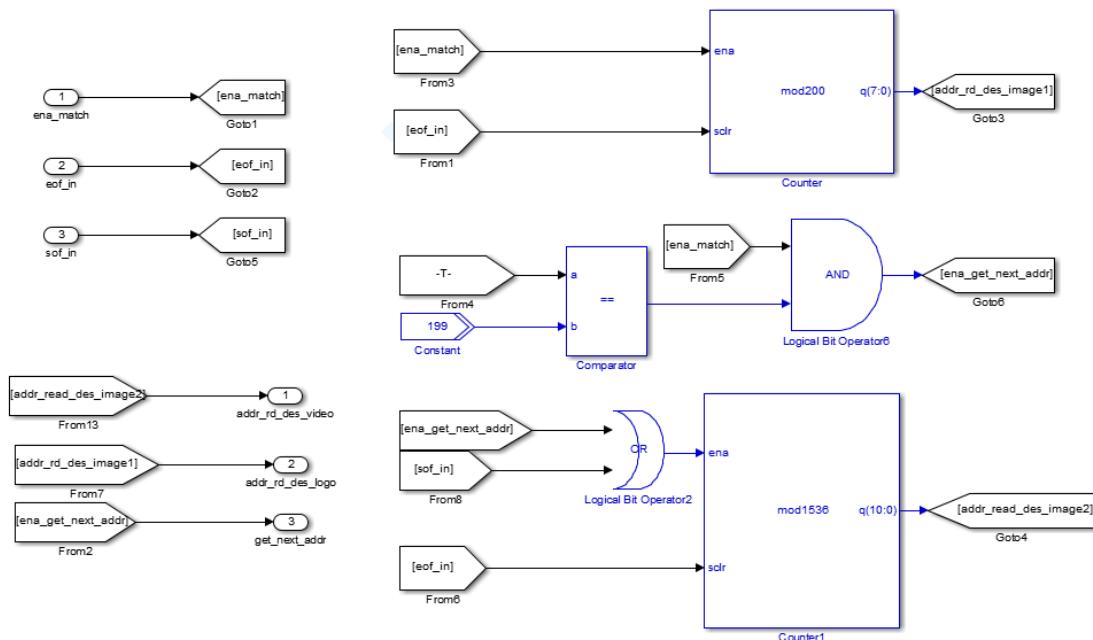


Hình 4 Đầu vào dữ liệu khởi matching

Nghiên cứu, thiết kế và triển khai thuật toán SIFT trên FPGA

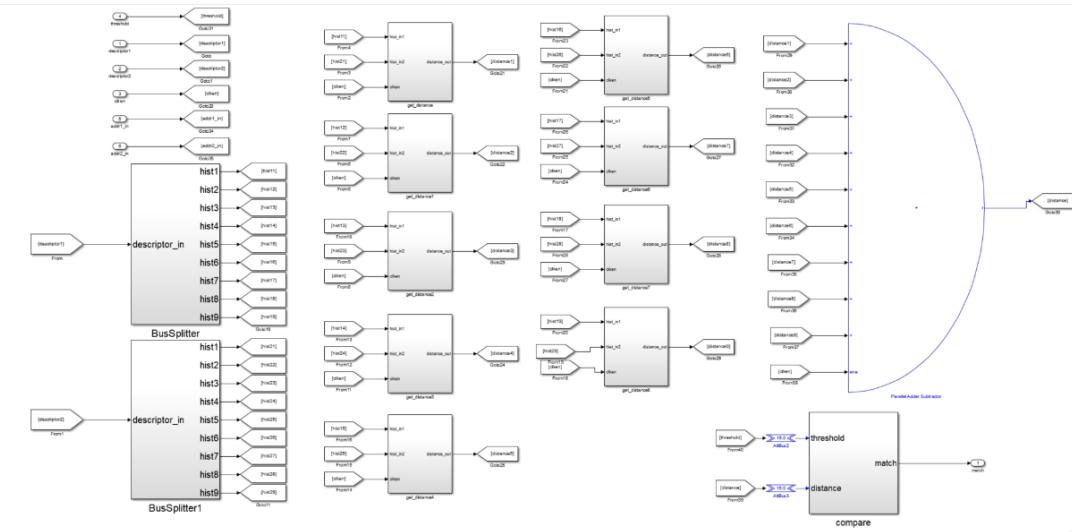


Hình 5 Thiết kế sử dụng 2 khối Ram lưu descriptor của từng ảnh

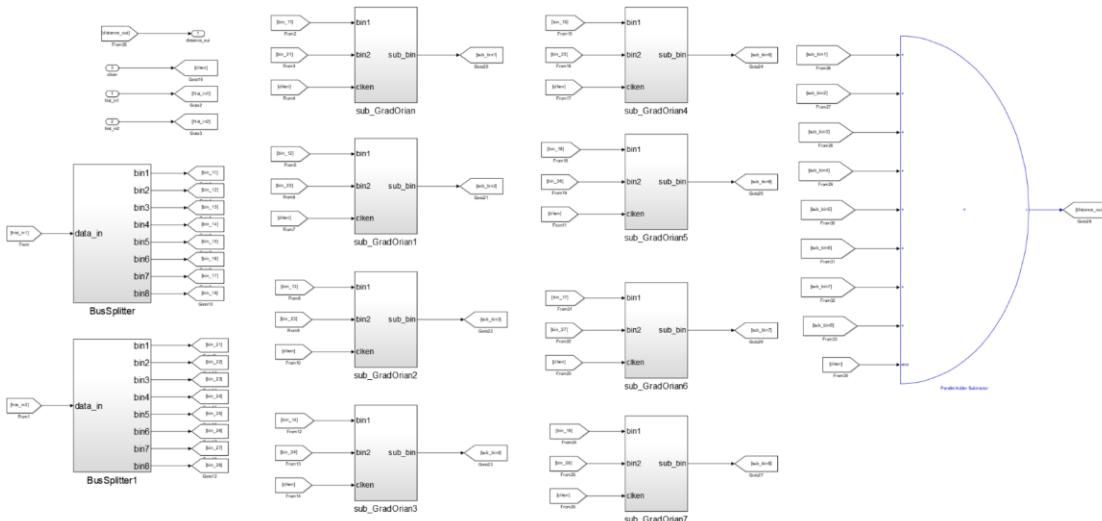


Hình 6 Thiết kế khói gen địa chỉ đọc descriptor từ 2 khói Ram

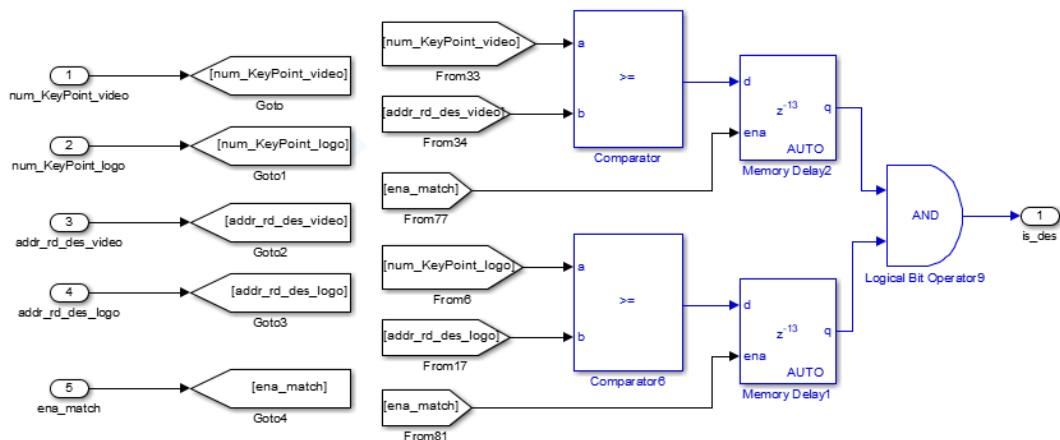
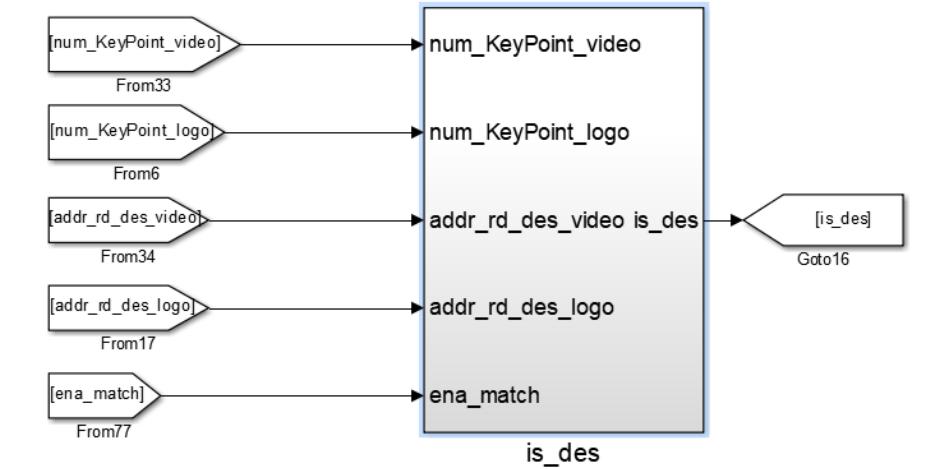
Nghiên cứu, thiết kế và triển khai thuật toán SIFT trên FPGA



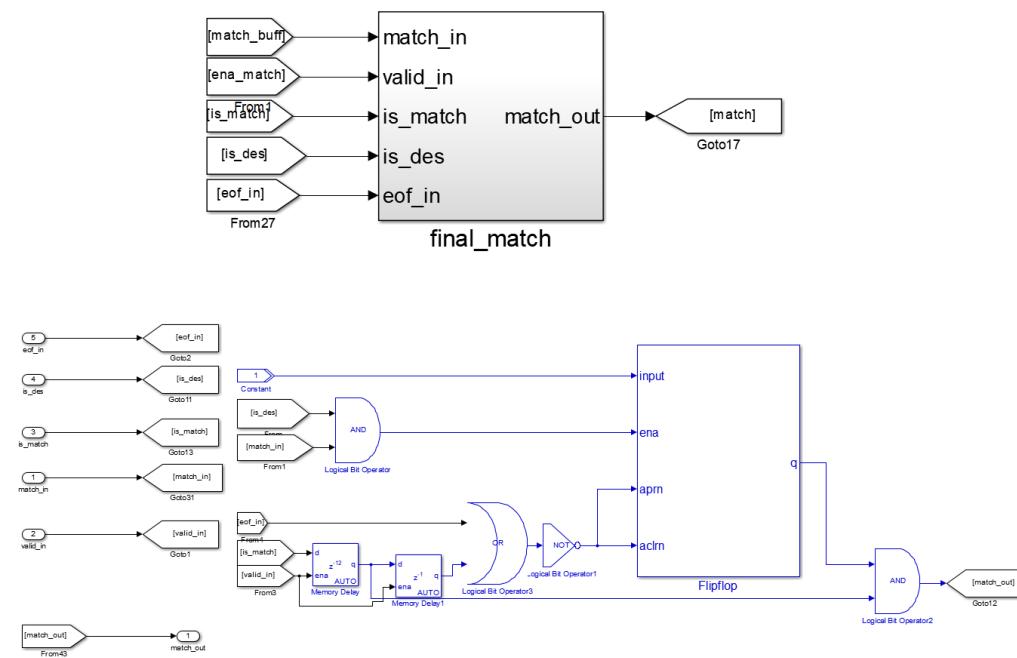
Hình 7 Thiết kế khối compare so sánh 2 descriptor



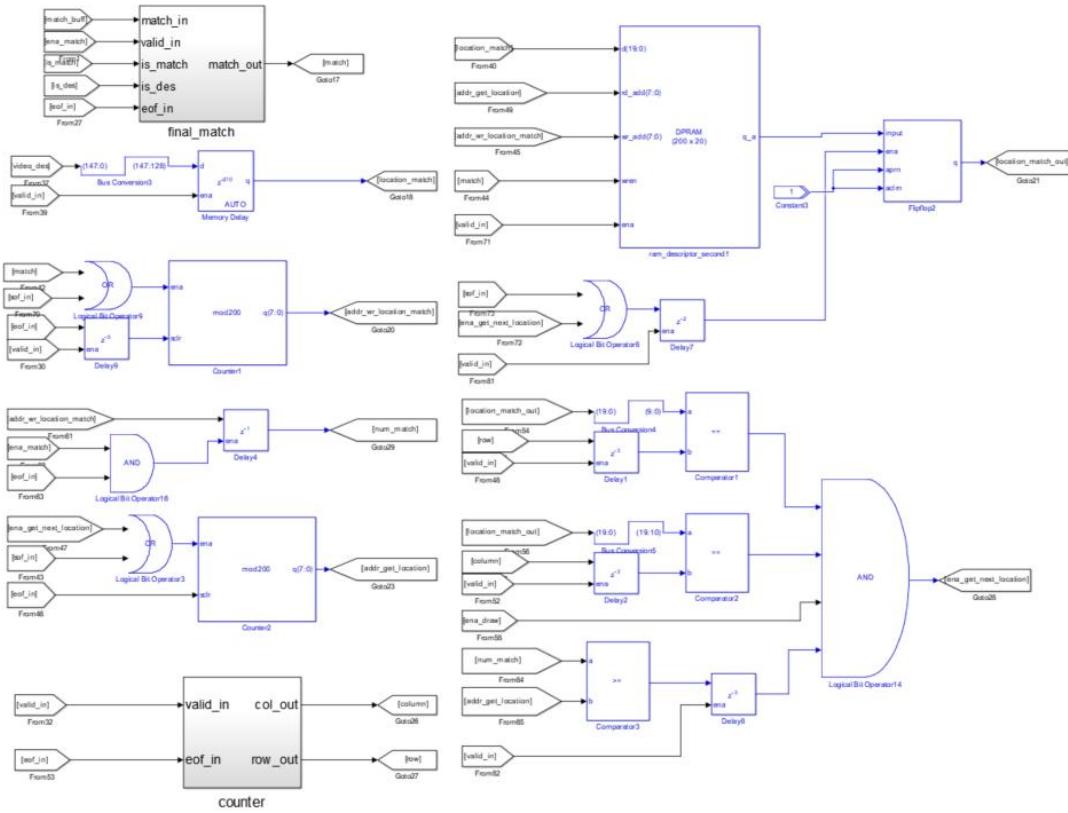
Hình 8 Thiết kế khối `get_distance`



Hình 9 Thiết kế khối *is_des* cho biết descriptor đang so sánh có phải descriptor không



Hình 10 Thiết kế khối *final_match*



Hình 11 Thiết kế đọc/ghi vị trí điểm hấp dẫn match vào ram