ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

PHẠM ĐỨC HIẾU

CÔNG NGHỆ LẬP TRÌNH FPGA VÀ ỨNG DỤNG XỬ LÝ DỮ LIỆU ĐA PHƯƠNG TIỆN

LUẬN VĂN THẠC SĨ

Hà Nội – 2016

ĐẠI HỌC QUỐC GIA HÀ NỘI

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

PHẠM ĐỨC HIẾU

CÔNG NGHỆ LẬP TRÌNH FPGA VÀ ỨNG DỤNG XỬ LÝ DỮ LIỆU ĐA PHƯƠNG TIỆN

Ngành: Công nghệ thông tin

Chuyên ngành: Kỹ thuật phần mềm

Mã số: 60480103

NGƯỜI HƯỚNG DẪN KHOA HỌC: TS. LÊ QUANG MINH

Hà Nội 2016

LỜI CAM ĐOAN

Tôi cam đoan đây là công trình nghiên cứu do chính tôi thực hiện.

Các số liệu, kết quả nêu trong luận văn là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác.

Hà Nội, ngày 26 tháng 5 năm 2016

Tác giả

Phạm Đức Hiếu

LỜI CẢM ƠN

Trước tiên tôi muốn gửi lời cảm ơn đến thầy giáo TS. Lê Quang Minh, người trực tiếp hướng dẫn tôi thực hiện luận văn này. Tôi cũng mong muốn bày tỏ long biết ơn đến các thầy, cô giáo Trường Đại học Công nghệ - Đại học Quốc gia Hà Nội đã tận tình dạy dỗ và tạo mọi điều kiện học tập thuận lợi cho tôi trong suốt khóa học qua.

Tôi xin gửi lời cảm ơn đến gia đình, đặc biệt là bố mẹ, anh, chị và vợ tôi những người luôn hết mình yêu thương, dùi dắt và ủng hộ tôi trong cuộc sống.

Cuối cùng tôi xin cảm ơn ban Giám hiệu trường Cao đẳng y tế Lạng Sơn, các anh chị em đồng nghiệp đã tạo điều kiện cho tôi tham gia và hoàn thành khóa học. Tôi xin cảm ơn các bạn của tôi, những người luôn bên cạnh động viên, giúp đỡ và đóng góp nhiều ý kiến thiết thực trong quá trình học tập và thực hiện luận văn./.

Hà Nội, ngày 25 tháng 6 năm 2016

Tác giả

Phạm Đức Hiếu

DANH MỤC CÁC KÝ HIỆU, TỪ VIẾT TẮT

Từ viết tắt	**Nghĩa tiếng anh**
FPGA	: Field-Programmable Gate Array
DSP	: Digital Storage Oscilloscop
RAM	: Ramdom Access Memory
ROM	: Read – only Memory
DLL	: Delay Locked Loop
ADC	: Analog-to-Digital Converter
ASIC	: Application-Specific Integrated Circuit
CPLD	: Complex Programmable Logic Device
DAC	: Digital - to - Analog Converter
DRAM	: Dynamic Ramdom Access Memory
EEPROM	: Electrically Erasable Programmable Read – Only Memory
FIFO	: First In First Out
HDL	: Hardware Description Language
I/O	: Input/Output
LAB	: Logic Array Block
LUT	: Look Up Table
PLA	: Programmable Logic Array
SPLD	: Simple Programmable Devices
SRAM	: Static Ramdom Access Memory
VHDL	: VHSIC hardware description language
VHSIC	: Very High Speed Itergrated Circuit
RTL	: Register Transfer Level
Avalon-ST	: Avalon Streaming

DANH MỤC BẢNG

DANH MỤC HÌNH

MỤC LỤC

LỜI MỞ ĐẦU

Trong những năm gần đây với sự phát triển của công nghệ bán dẫn trong việc tạo ra những bộ vi xử lý và vi điều khiển, các hệ thống đo lường và điều khiển ngày càng thông minh hơn, giải quyết được nhiều bài toán phức tạp hơn. Tuy nhiên các hệ thống ngày càng hoàn thiện hơn, thông minh hơn thì vi xử lý và vi điều khiển chưa thể đáp ứng hết nhu cầu. Sự xuất hiện các thiết bị có thể lập trình như FPGA (Filed Progammable Gate Array đã tạo ra bước đột phá. Với công nghệ FPGA đã có rất nhiều công trình nghiên cứu giải quyết các bài toán lớn như bài toán xử lý tín hiện số, mật mã và nhận dạng. Các nghiên cứu hiện nay chủ yếu đi theo hướng kết hợp các bộ FPGA với những bộ xử lý thông thường trong một chip tạo ra hệ thống mới RCS (reconfigurable Computing System). Đây là một mô hình mới trong thiết kế các hệ thống có khả năng tính toán rất mạnh, thay thế được cho những máy tính lớn. Không những thế, các thiết bị này có khả năng tái lập trình nên các hệ thống này có độ linh hoạt cao, có thể thay đổi lại cấu hình để đáp ứng nhiều thuật toán hay các yêu cầu khác nhau về phần cứng trong quá trình hệ thống đang hoạt động.

Cùng với sự bùng nổ của các mạng internet, mạng di động là các nhu cầu giải trí, truyền thông đa phương tiện. Để tăng chất lượng dịch vụ của các dịch vụ đa phương tiện thì việc xử lý các dữ liệu đa phương tiện là rất cần thiết. Hiện nay các giải pháp xử lý dữ liệu đa phương tiện đều có chi phí rất cao. Do vậy mục đích của nghiên cứu này là ứng dụng công nghệ FPGA vào việc xử lý dữ liệu đa phương tiện một cách hiệu quả.

Trong những năm gần đây, giám sát tự động là môt trong những lĩnh vực được quan tâm và phát triển rộng rãi. Một trong những lý do khiến giám sát tự động phát triển mạnh mẽ là do sự tiến bộ của khoa học và khả năng ứng dụng rộng khắp của hệ thống. Tuy nhiên, các hệ thống sử dụng các camera giám sát do các điều kiện khách quan như ánh sáng thay đổi, … mà các hình ảnh, video thu được thường xuất hiện các nhiễu và chất lượng hình ảnh không được tốt. Vì vậy, một hệ thống xử lý nâng cao chất lượng dữ liệu hình ảnh, video thu được từ

camera là cần thiết giúp cho hệ thống giám sát hoạt động hiệu quả hơn. Do đó, luận văn lựa chọn xây dựng thử nghiệm hệ thống xử lý ảnh trên công nghệ FPGA nhằm nâng cao chất lượng hình ảnh trong hệ thống camera giám sát.

Đối tượng và phạm vi nghiên cứu:

1. Công nghệ FPGA: các lĩnh vực ứng dụng, các công cụ phát triển

2. Kỹ thuật xử lý ảnh.

3. Ứng dụng công nghệ FPGA xử lý dữ liệu hình ảnh

Những nội dung chính: Nội dung của luận văn gồm phần đặt vấn đề, 3 chương, kết luận và tài liệu tham khảo.

Chương I: Tổng quan về công nghệ FPGA

Chương này trình bày tổng quan về công nghệ FPGA, các lĩnh vực ứng dụng của công nghệ này và các công cụ phát triển, hỗ trợ lập trình trên FPGA.

Chương II Một số kỹ thuật xử lý hình ảnh

Chương này trình bày các khái niệm cơ bản về hình ảnh, các loại nhiễu và giới thiệu một số phương pháp xử lý hình ảnh, đánh giá hiệu quả của các phương pháp xử lý dữ liệu hình ảnh.

Chương III Ứng dụng công nghệ FPGA xử lý dữ liệu đa phương tiện dạng ảnh

Thực hiện thiết kế lõi IP xử lý nâng cao chất lượng hình ảnh, cụ thể lõi IP sẽ xác định các điểm ảnh bị lỗi và sửa chúng, xác định màu bằng phương pháp nội suy, sửa ma trận màu.

Phần kết luận của luận văn trình bày các kết quả đạt được và những hạn chế của luận văn, hướng phát triển của luận văn trong các nghiên cứu tiếp theo.

CHƯƠNG I: TỔNG QUAN VỀ CÔNG NGHỆ FPGA

1.1. Lịch sử ra đời FPGA

Năm 1984 Ross Freeman là người đầu tiên thiết kế PFGA và cũng là người sáng lập công ty Xilinx. Kiến trúc mới của FPGA cho phép tích hợp số lượng lớn các phần tử bán dẫn vào một vi mạch so với kiến trúc trước đó là CPLD. FPGA có khả năng chứa từ 100.000 đến vài tỷ cổng logic, trong khi CPLD chỉ chứa từ 10.000 đến 100.000 cổng logic, con số này đối với PAL vá PLA còn thấp hơn rất nhiều chỉ đạt vài nghìn đến 10.000 cổng logic.

SPLD thường là một mảng logic AND/OR lập trình được có kích thước xác định và chứa một số lượng hạn chế các phần tử nhớ đồng bộ (*clocked register*). Cấu trúc này hạn chế khả năng thực hiện những hàm phức tạp và thông thường hiệu xuất làm việc của vi mạch phụ thuộc vào cấu trúc cụ thể của vi mạch hơn là vào yêu cầu bài toán.

Kiến trúc của FPGA là kiến trúc mảng các khối logic, khối logic nhỏ hơn nhiều nếu đem so sánh với một khối SPLD, ưu điểm này giúp FPGA có thể chứa nhiều hơn các phần tử logic và phát huy tối đa khả năng lập trình của các phần tử logic và hệ thống mạch kết nối, để đạt được mục đích này thì kiến trúc của FPGA phức tạp hơn nhiều so với CPLD. Một điểm khác biệt với CPLD là trong những FPGA hiện đại được tích hợp nhiều những bộ logic số học đã sơ bộ tối ưu hóa hỗ trợ RAM, ROM tốc độ cao, hay các bộ nhân.

Ngoài khả năng tái cấu trúc vi mạch toàn cục, FPGA hiện tại còn hỗ trợ tái cấu trúc một bộ phận riêng lẻ trong khi vẫn đảm bảo hoạt động bình thường cho các bộ phận khác.

1.2. Sự phát triển của FPGA

Các thiết bị lập trình được, gọi chung là các thiết bị khả trình, có vai trò rất quan trọng trong thiết kế phần cứng số. Chúng là các chíp đa dụng có thể được cấu hình theo nhiều cách cho nhiều ứng dụng khác nhau.

+ Loại đầu tiên của thiết bị khả trình được sử dụng rộng rãi là Programmable read-only Memory-PROM. PROM là thiết bị lập trình chỉ được một lần gồm một dãy các ô nhớ chỉ đọc. PROM có thể thực hiện bất kỳ một hàm logic theo bảng sự thật nào đó, bằng cách sử dụng các đường địa chỉ như các ngõ nhập vào và ngõ xuất được xác định nội dung các bít nhớ. Có hai loại PROM cơ bản là Mask-Programmable và Field-Programmable.

- Mask-Programmable là loại thiết bị được lập trình bởi nhà sản xuất. Các chíp này thường sản xuất các chíp lôgíc tốc độ cao vì các kết nối bên trong thiết bị được thực hiện bằng phần cứng ngay từ khi sản xuất.

- Field-Programmable là thiết bị được lập trình bởi người dùng. Các kết nối bên trong của Field-Programmable luôn cần đến một số chuyển mạch lập trình được (cầu chì, transistor truyền…) vì vậy tốc độ truyền chậm hơn của thiết bị nối cứng *(Mask-Programmable)*. Tuy nhiên nó có nhiều ưu điểm như:

. Các chíp Field-Programmable có thể lập trình trong thời gian ngắn (khoảng vài phút hay vài giờ đồng hồ) còn các chíp Mask-Programmable khi sản xuất phải thực hiện trong thời gian dài (hàng tuần hay hàng tháng).

. Các chíp Field-Programmable rẻ hơn nhiều so với Mask-Programmable khi sản xuất với số lượng nhỏ.

Hai biến thể của PROM là EPROM chúng đều có chung ưu điểm là có khả năng xoá và lập trình lại nhiều lần.

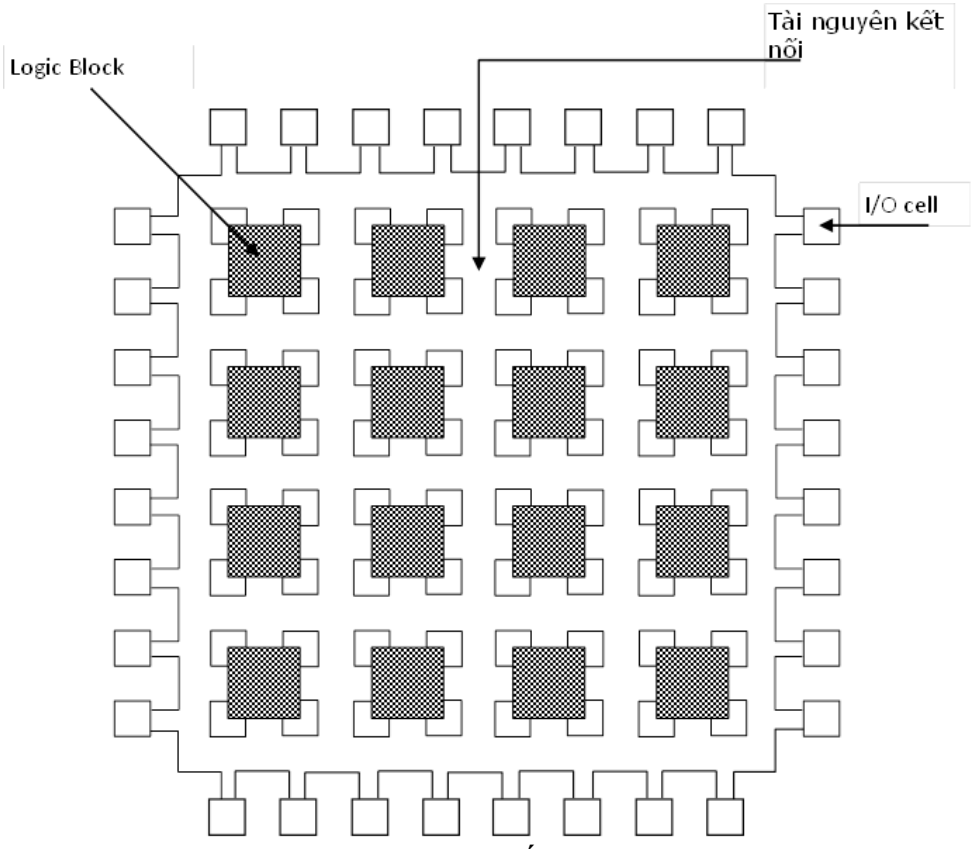
Tiếp đến là các thiết bị PLD, chúng thông thường có cấu tạo gồm một dãy các cổng AND được nối với một dãy các cổng OR. Loại cơ bản của PLD là PAL, PAL gồm một khối các cổng AND lập trình được nối đến các khối cổng OR cố định. Một loại nữa của PLD linh hoạt hơn PAL là PLA. PLA cũng có cấu trúc giống PAL nhưng các kết nối khả trình. Trong PLA cũng có hai loại là Mask-Programmable và Field-Programmable.

Cả hai loại PLD trên cho phép thực hiện các mạch logic có tốc độ cao tuy nhiên cấu trúc đơn giản của chúng chỉ cho phép thực hiện được các mạch logic cỡ nhỏ.

Loại thiết bị khả trình tổng quát nhất gồm dãy các phần tử rời rạc có thể kết nối với nhau theo mô tả của người sử dụng, được gọi là Mask-Programmable Gate Array (MPGA) chúng có cấu trúc cơ bản nhất là gồm các hàng transistor có thể được kết nối với nhau để thực hiện các mạch logic. Các kết nối do người dùng định nghĩa này có thể trong cả hàng và cả cột. Ưu điểm chính của MPGA so với PLD là nó cung cấp các kiến trúc tổng quát cho phép thực hiện các mạch logic lớn hơn. Vì cấu trúc của chúng có thể mở rộng lớn hơn cùng với số lượng logic lớn hơn.

FPGA đã kết hợp khả năng lập trình của PLD và kết cấu nối có thể mở rộng của MPGA. Do đó các thiết bị lập trình loại này có mật độ logíc cao hơn. FPGA được công ty Xilinx giới thiệu lần đầu tiên vào năm 1985 và đến nay đã có nhiều công ty phát triển như: Actel, Altera, Plessey, Plus logic, Quick...

1.2.1. Cấu trúc FPGA



Hình 1-1. Cấu trúc FPGA

FPGA là mạch tích hợp chứa nhiều (64 đến hơn 10.000) ô logic (logic cell) giống nhau có thể xem là các thành phần chuẩn. Mỗi ô logic giữ một hay một số chức năng độc lập (Hình 1-1). Các ô giống nhau được kết nối bởi một ma trận đường dẫn và các chuyển mạch khả trình. Người thực hiện thiết kế bằng các đặc trưng lôgic đơn của mỗi ô và lựa chọn đóng các chuyển mạch trong ma trận kết nối. Mảng của các ô lôgic và kiểu kết nối là kết cấu xây dựng khối cơ bản trong mạch lôgic. Các thiết kế phức tạp được tạo ra bằng cách kết hợp các khối cơ bản để tạo ra các mạch được mô tả.

Mô hình tổng quát của FPGA gồm một dãy hai chiều các khối lôgic (*logic block*) có thể được kết nối bằng các nguồn kết nối chung. Các nguồn kết nối gồm các đoạn kết nối (*segment*) có thể có chiều dài khác nhau. Bên trong các kết nối là các chuyển mạch lập trình được dùng để nối các khối lôgic với các đoạn dây, các khối vào/ra hay các đoạn dây với nhau. Mạch lôgic cài đặt trong FPGA bằng cách ánh xạ lôgic vào các khối lôgic riêng rẽ và sau đó nối các khối lôgic cấu hình (*Configurable logic Block*) cần thiết qua các chuyển mạch. Các khối CLB cung cấp các phần tử chức năng với cấu trúc sử dụng logic. Các khối vào/ra (*I/O Block*) cung cấp giao diện giữa các gói chân và các đường tín hiệu bên trong. Tài nguyên kết nối khả trình cung cấp các bộ phận truyền dẫn tới kết nối đầu vào và đầu ra của các CLB và các IOB trong mạng riêng.

Vậy cấu trúc FPGA gồm ba phần tử chính: Các khối lôgic cấu hình (CLB), các khối vào/ra (IOB) và các kết nối.

a. Các khối lôgic cấu hình:

Cấu trúc và nội dung của logic block được gọi theo kiến trúc của nó. Kiến trúc của khối lôgic có thể thiết kế theo nhiều cách khác nhau, có thể là các cổng AND 2 ngõ nhập, các bộ dồn kênh (*Multiplexer*) hay các bảng tìm kiếm (*Lock-up Table*). Ngoài ra có thể chứa các Flip-Flop để hỗ trợ cho việc thực hiện một cách tuần tự.

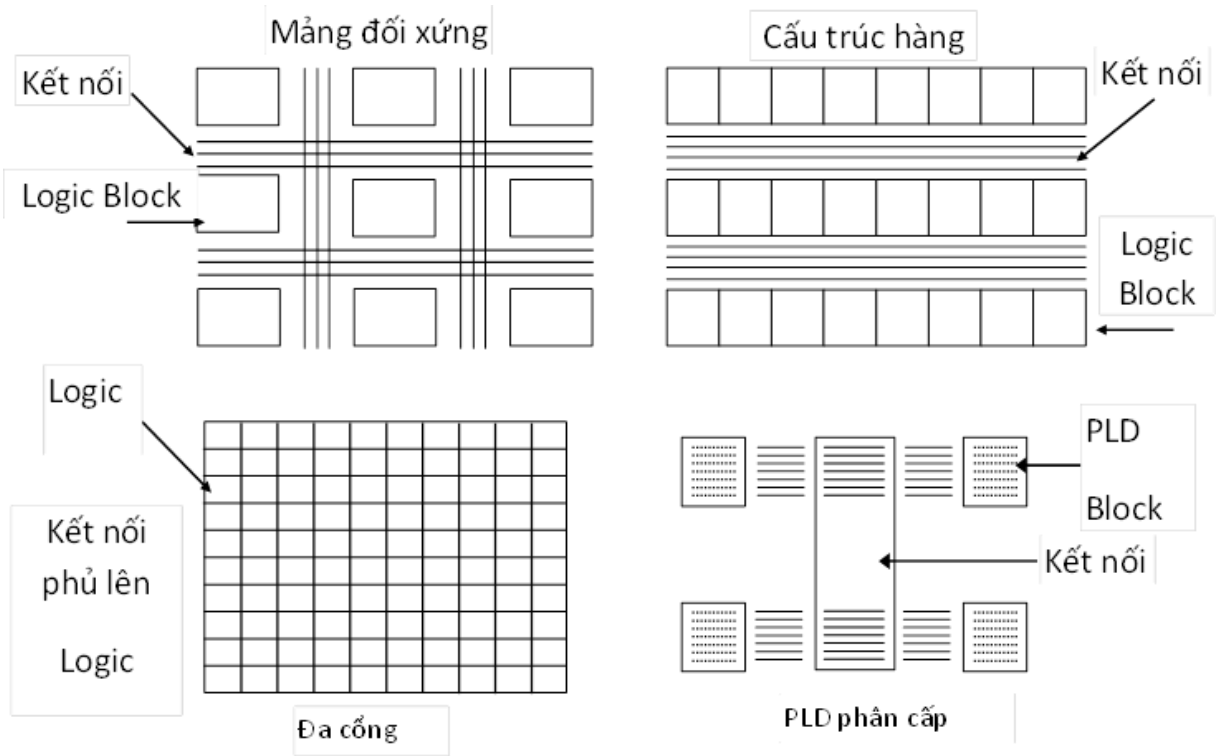
b. Các nguồn kết nối :

Các nguồn kết nối có cấu trúc và nội dung được gọi là kiến trúc đường

(*Routing Architecture*). Kiến trúc Routing gồm các đoạn đây nối và các chuyển mạch khả trình. Các chuyển mạch khả trình có cấu tạo khác nhau như pass-transistor, được điều khiển bởi các cell SRAM, các phần tử cầu chì nghịch, EPROM transistor và EEROM transitor. Giống như các khối lôgic có nhiều cách khác nhau để thiết kế các kiến trúc routing. Một số FPGA cung cấp nhiều kết nối đơn giản giữa các khối logic, một số khác cung cấp ít kết nối hơn nên routing phức tạp hơn.

1.2.2. Phân loại FPGA

FPGA có nhiều loại khác nhau có cấu trúc và đặc tính riêng tuỳ theo từng hãng sản xuất, tuy nhiên chúng có bốn loại chính sau: cấu trúc mảng đối xứng (*Symmetrical Array*), cấu trúc PLD phân cấp *(hierachircal PLD)*, cấu trúc hàng *(Row base)* và cấu trúc đa cổng *(Sea of Gate)* mô tả dưới đây.



Hình 1.2 Các loại cấu trúc FPGA

1.2.3. Các công nghệ lập trình FPGA

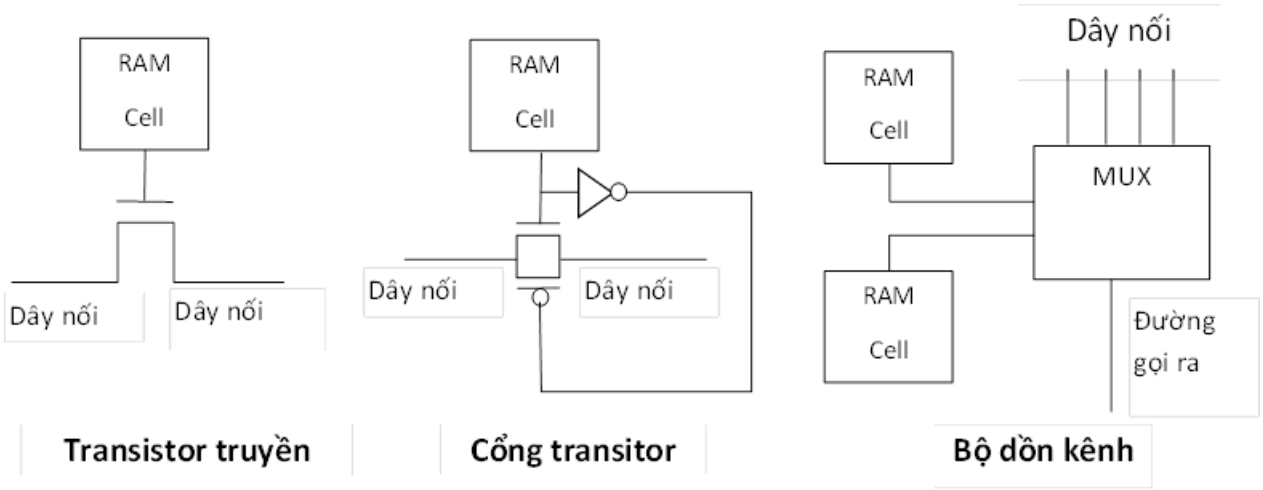
Có nhiều cách thực hiện các phần tử lập trình, các công nghệ lập trình hiện đang sử dụng là: RAM tĩnh, cầu chì nghịch (anti-fuse), EPROM transistor và EEROM transistor. Mặc dù công nghệ lập trình khác nhau, tất cả các phần tử lập trình đều có chung tính chất là có thể cấu hình được trong một trong hai trạng thái ON hoặc OFF. Các phần tử lập trình được dùng để thực hiện các kết nối lập trình được giữa các khối lôgic của các FPGA, còn FPGA thông thường có thể hơn 100.000 phần tử lập trình. Vì vậy các phần tử lập trình phải có những tính chất sau:

- Chiếm càng ít diện tích của chíp càng tốt.
- Có trở kháng thấp khi ở trạng thái ON và trở kháng cao khi ở trạng thái OFF.
- Có điện dung ký sinh thấp khi kết nối các đoạn dây.
- Có thể chế tạo một cách tin cậy số lượng lớn phần tử lập trình trên một chíp.

Có thể tuỳ thuộc vào ứng dụng cụ thể và có các số lượng phần tử lập trình có thể có các đặc tính khác. Về mặt chế tạo, các phần tử lập trình nếu có thể chế tạo theo công nghệ CMOS chuẩn là tốt nhất. Dưới đây sẽ trình bày chi tiết các công nghệ lập trình FPGA.

a. Công nghệ lập trình dùng RAM tĩnh

Công nghệ lập trình dùng RAM tĩnh (SRAM) sử dụng công nghệ CMOS tiêu chuẩn. Các kết nối lập trình được điều khiển bằng các transistor khác trên chíp hoặc bật (On) các transistor truyền dẫn cũng như các cổng transistor để tạo một kết nối hay tắt (Off) để ngắt kết nối.



Hình 1.3 Công nghệ lập trình ram tĩnh

Trong trường hợp transistor truyền dẫn và cổng transistor như hình trên, phần tử RAM Cell điều khiển cổng truyền bật hoặc tắt. Khi tắt giữa hai dây nối với cổng truyền dẫn sẽ có một trở kháng rất cao. Khi bật nó sẽ tạo một trở kháng thấp kết nối giữa hai dây nối. Đối với bộ dồn kênh, SRAM Cell điều khiển ngõ nhập nào của bộ dồn kênh sẽ được nối với ngõ ra của nó. Cách này thường dùng để kết nối tuỳ chọn từ một hay nhiều ngõ nhập của một khối logic.

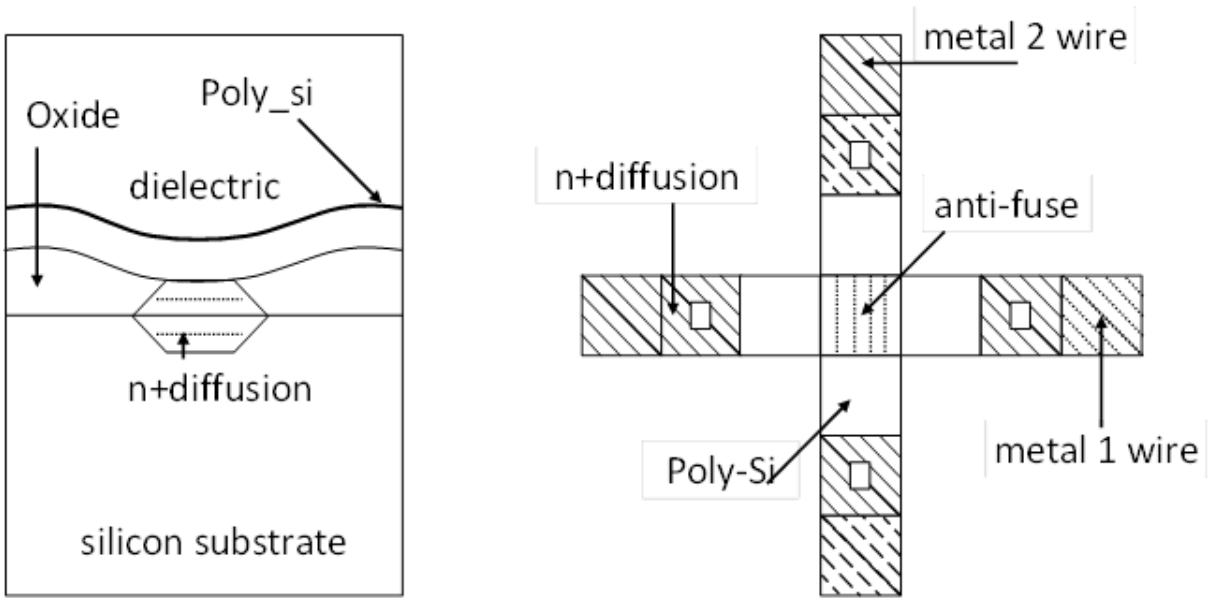
Trong các FPGA sử dụng công nghệ lập trình SRAM, các khối logic có thể được kết hợp với nhau qua cách kết hợp cả bộ dồn kênh (*Multiplexer*) và cổng truyền dẫn (*pass-gate*). Vì SRAM là bộ nhớ bay hơi, các FPGA này phải được tái cấu hình mỗi khi cấp nguồn cho chíp. Điều này có nghĩa là hệ thống sử dụng các chíp này phải có một số cơ chế lưu trữ thường trực cho các bit của RAM Cell, chẳng hạn ROM hay đĩa từ. Các bit của RAM Cell có thể được nạp vào FPGA một cách tuần tự hay định địa chỉ như một phần tử của mảng (theo cách thông thường của một RAM).

Các chíp được thực hiện theo công nghệ SRAM có diện tích khá lớn, bởi vì cần ít nhất 5 transistor cho mỗi RAM Cell cũng như các transistor cần thêm cho cổng truyền dẫn hay bộ dồn kênh. Ưu điểm của kỹ thuật này là cho phép FPGA có thể được tái cấu hình ngay trên mạch rất nhanh và nó có thể được chế tạo bằng công nghệ CMOS chuẩn.

b. Các thiết bị lập trình cầu chì nghịch (Anti-fuse)

Công nghệ lập trình anti-fuse được sử dụng trong các FPGA của Actel-Corp, Quick Logic và Cross Point Solution. Tuy anti-fuse được sử dụng trong các loại FPGA này có cấu tạo khác nhau, nhưng chức năng của chúng là như nhau. Một anti-fuse bình thường sẽ ở trạng thái cao, nhưng có thể bị "nóng chảy" thành trạng thái điện trở thấp khi được lập trình ở điện thế cao. Dưới đây sẽ giới thiệu cấu tạo của các anti-fuse của Actell và Quick Logic.

Anti-fuse của Actell được gọi là PLICE. Nó cấu trúc hình chữ nhật gồm 3 lớp: Lớp dưới cùng chứa các silic mang nhiều điện tích dương (n+diffusion), lớp giữa là một lớp điện môi (Oxy-Nitơ-Oxy cách điện), và lớp trên cùng là Poly-Silic.



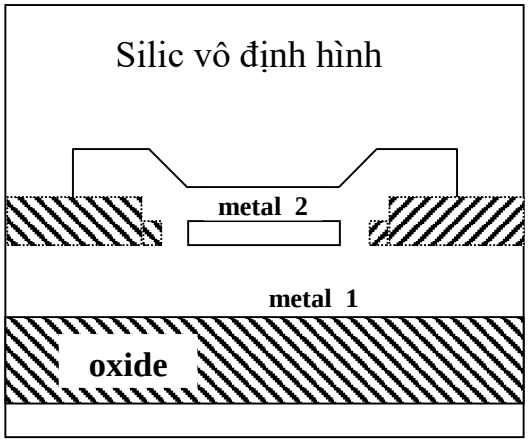
Hình 1.4 Công nghệ lập trình cầu chì nghịch PLICE

Anti-fuse PLICE được lập trình bằng cách đặt một điện thế cao thích hợp (18V) giữa hai đầu của anti-fuse và dòng điều khiển khoảng 5mA qua thiết bị. Dòng và áp này tạo ra một nhiệt lượng đủ nóng bên trong lớp điện môi làm nó nóng chảy và tạo ra một liên kết dẫn điện giữa các điện cực. Các transistor chịu được các điện thế cao được chế tạo bên trong FPGA để đáp ứng cho dòng và điện áp đủ

lớn. Cả hai lớp dưới cùng và trên cùng của cầu chì nghịch được nối với các dây kim loại để khi được lập trình cầu chì nghịch sẽ tạo ra một kết nối có trở kháng thấp (300Ω đến 500Ω) giữa hai dây kim loại.

Anti-fuse của Quick-Logic được gọi là ViaLink. Nó tương tự như PLICE cũng có ba lớp kim loại. Tuy nhiên, ViaLink sử dụng kim loại mức 1 cho lớp dưới cùng, một hợp chất vô dịnh hình cho lớp giữa và kim loại mức 2 cho lớp trên cùng. Khi ở trạng thái không được lập trình, anti-fuse có trở kháng hàng gigaôm, nhưng khi được lập trình nó sẽ tạo ra một kết nối giữa hai lớp kim loại trở kháng khoảng 80Ω. Anti-fuse được chế tạo bằng cách thêm 3 mặt nạ đặc biệt trong quy trình chế tạo CMOS thông thường.

ViaLink anti-fuse được lập trình bằng cách đặt một điện thế 10V giữa các đầu của nó, dòng được cấp đủ, trạng thái của Silic vô định hình sẽ thay đổi và tạo ra một liên kết điện giữa hai lớp kim loại. Diện tích các chíp sử dụng kỹ thuật anti-fuse rất nhỏ so với công nghệ khác. Tuy nhiên, bù lại cần phải có không gian lớn cho các transistor điện thế cao cần để giữ cho dòng và áp cao lúc lập trình. Nhược điểm của anti-fuse là quy trình chế tạo chúng phải thay đổi so với quy trình chế tạo SMOS.

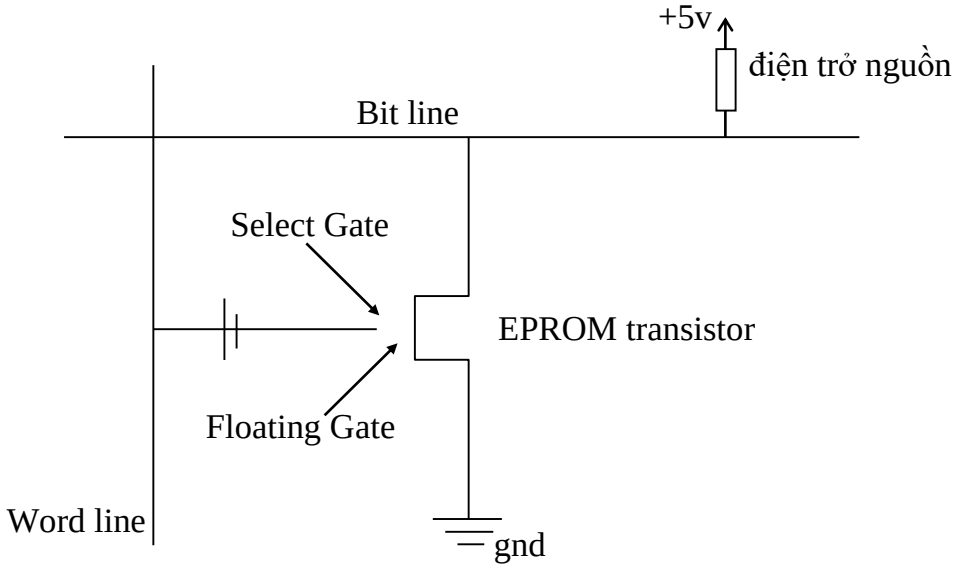


Hình 1.5 Công nghệ lập trình cầu chì nghịch ViaLink

c. Công nghệ lập trình dùng EPROM và EEROM

Công nghệ được dùng trong các FPGA của Altera Corp, và Plus Logic. Công nghệ này giống như sử dụng trong bộ nhớ EPROM. Không giống CMOS

transistor đơn giản, một EPROM transistor gồm hai cổng, một cổng treo (*floating-gate*) và một cổng chọn (*select-gate*). Cổng treo được đặt giữa cổng chọn và kênh dẫn của transitor, cổng này được gọi như thế vì nó không có kết nối điện đến bất kỳ mạch nào.



Hình 1.6 Công nghệ lập trình EPROM transistor

Ở trạng thái bình thường không được lập trình, không có điện tích giữa cổng treo (*floating-gate*) và transitor có thể chuyển sang trạng thái ON một cách bình thường bằng cổng chọn (*select-gate*). Khi transistor được lập trình bằng một dòng điện lớn chạy giữa nguồn và kênh, một điện tích được giữ lại ở cổng treo. Điện tích này làm transistor chuyển sang trạng thái OFF. Bằng cách này, EPROM transistor có thể có chức năng của một phần tử lập trình được. Một EPROM transistor có thể được tái lập trình bằng cách huỷ bỏ lớp điện tích được giữ lại ở cổng treo (phơi dưới ánh sáng cực tím sẽ kích hoạt các electron chuyển từ cổng vào chất nền của transistor).

EPROM transistor được sử dụng trong FPGA theo cách khác với SRAM và anti-fuse thay vì dùng cho lập trình kết nối hai dây, EPROM transistor được sử dụng để "kéo xuống" các ngõ nhập của logic-block.

Như hình vẽ (1.6), một đường dây gọi là "word line" (theo thuật ngữ bộ nhớ) được nối với cổng chọn của EPROM transistor, khi transistor chưa được lập trình ở trạng thái ON. "Word line" có thể làm cho "bit line" không được nối với ngõ nhập của logic-block vì bị kéo về mức logic không. Nhiều EPROM transistor ứng với nhiều "word line" được nối với cùng một "bit line", khi một điện kéo lên nguồn nối với "bit line", mô hình không những cho EPROM transistor thực hiện các kết nối mà còn thực hiện các chức năng logic AND nối dây (wired-AND). Nhược điểm của phương pháp này là các điện trở tiêu tốn năng lượng cố định.

Một ưu điểm của EPROM transistor là chúng có thể tái lập trình mà không cần bộ nhớ bên ngoài. Tuy nhiên, không giống SRAM, EPROM transistor không thể được tái lập trình ngay trên bo mạch.

Phương pháp dùng EEPROM (được sử dụng trong các FPGA của Advanced Micro Device-AMD) tương tự như công nghệ EPROM, ngoại trừ EEPROM transistor tốn gấp đôi diện tích chíp so với EPROM transistor và cần nhiều nguồn điện thế (để tái lập trình) mà các loại khác không cần.

Các công nghệ lập trình FPGA được tóm tắt trong bảng dưới đây:

Công nghệ lập trình	**Tính bay hơi**	**Có thể lập trình**	**Diện tích chíp**	**R(Kohm)**	**C(pf)**
Static RAM Cell	Có	Trong mạch	Lớn	1-2	10-20
PLICE Anti-fuse	Không	Không	Anti-fuse nhỏ Số transistor lớn	300-500	3-5
ViaLink Anti-fuse	Không	Không	Anti-fuse nhỏ Số transistor lớn	50-80	1-3
EPROM	Không	Ngoài mạch	Nhỏ	2-4	10-20
EEPROM	Không	Trong mạch	2xEPROM	2-4	10-20

Bảng 1.1 Các đặc tính của công nghệ lập trình

1.3. Các lĩnh vực ứng dụng của công nghệ FPGA

FPGA là thế hệ sau của IC khả trình nên chúng có thể ứng dụng trong hầu hết các ứng dụng của hiện đang dùng MPGA, PLD và các mạch tích hợp loại nhỏ (SSI).

a. Các mạch tích hợp là ứng dụng đặc biệt

FPGA là thiết bị tổng quát nhất để thực hiện các mạch lôgic số. Chúng đặc biệt thích hợp cho các mạch tích hợp chuyên dụng đặc biệt (ASIC) như bộ cộng, bộ điều khiển lôgic Flip-Flop...

b. Thiết kế mạch ngẫu nhiên

Mạch lôgic ngẫu nhiên thường được thực hiện bằng PAL. Nếu tốc độ của mạch không đòi hỏi khắt khe (các PAL nhanh hơn hầu hết các FPGA) thì mạch có thể thực hiện bằng FPGA. Hiện nay một FPGA cần từ 10 đến 20 PAL.

c. Thay thế các chíp SSI cho mạch ngẫu nhiên

Các mạch hiện tại trong các sản phẩm thương mại thường chứa nhiều chíp SSI. Trong nhiều trường hợp có thể thay thế bằng FPGA để giảm diện tích bo mạch.

d. Chế tạo mẫu

FPGA rất lý tưởng cho việc tạo mẫu các sản phẩm. Giá thành thực hiện thấp và thời gian thực hiện thiết kế vật lý ngắn, cung cấp các ưu điểm hơn nhiều so với các phương tiện truyền thống khác để chế tạo mẫu phần cứng. Các mẫu ban đầu có thể thực hiện rất nhanh và những thay đổi sau đó được thực hiện rất nhanh và ít tốn kém.

e. Máy tính dựa trên FPGA

Một loại máy tính dựa trên FPGA có thể tái lập trình ngay trên FPGA. Các máy này có một bo mạch chứa các FPGA với các chân nối với các chíp lân cận giống như thông thường. Ý tưởng là là một chương trình phần mềm có thể được "biên dịch" (sử dụng kỹ thuật tổng hợp mức cao, mức lôgic và mức sơ đồ bằng tay) vào ngay phần cứng. Phần cứng này sẽ được thực hiện bằng cách lập trình bo mạch FPGA. Phương pháp này có hai ưu điểm chính: một là không cần

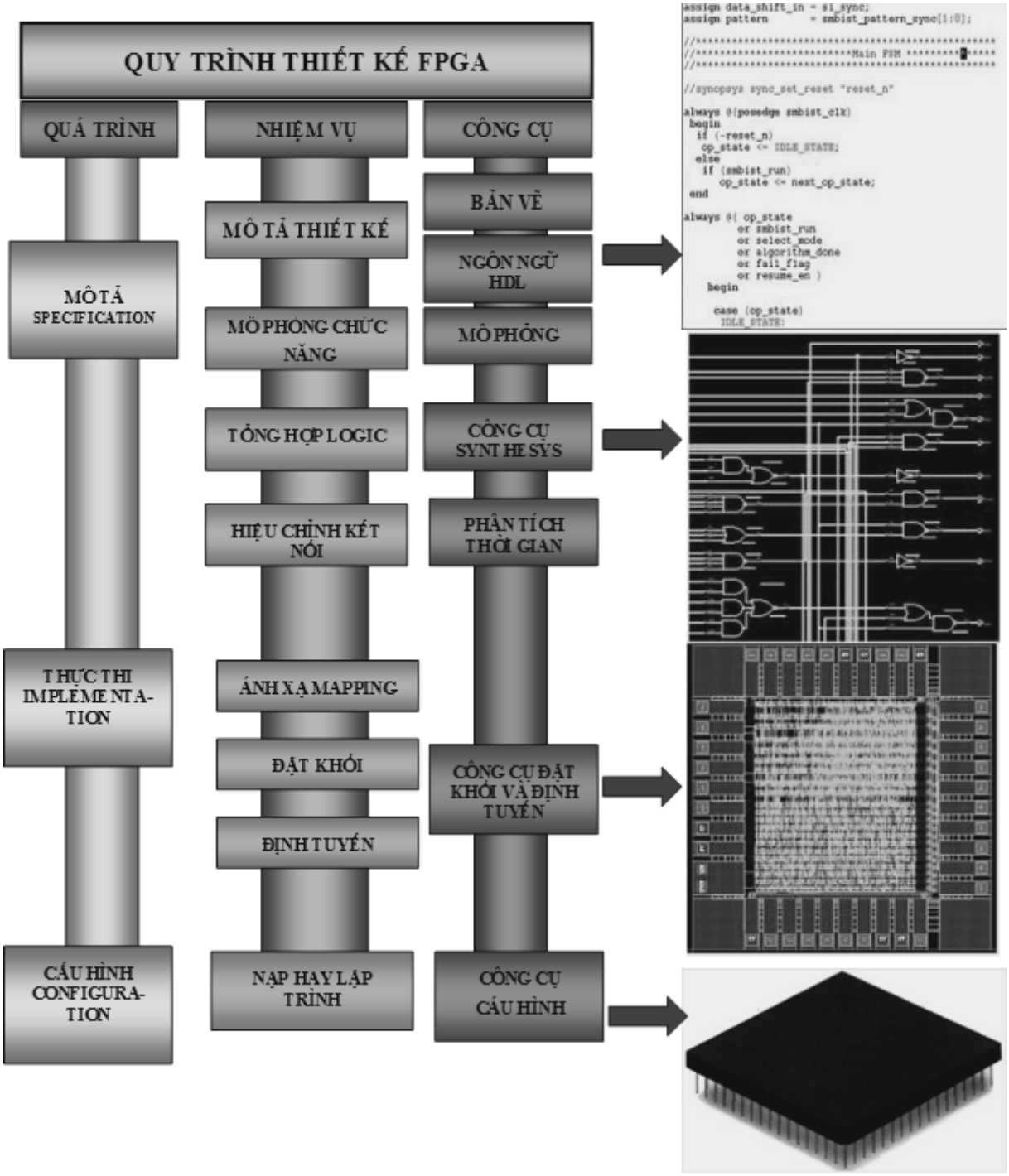
quá trình lấy lệnh như các bộ xử lý truyền thống vì phần cứng đã gộp cả lệnh. Kết quả là tốc độ có thể tăng lên hàng trăm lần. Hai là, môi trường tính toán có thể thực hiện song song mức cao, làm tăng tốc thêm nữa.

f. Tái cấu hình thành phần trực tiếp

FPGA cho phép có thể thay đổi theo mong muốn cấu trúc của một máy đang hoạt động. Một ví dụ là các thiết bị máy tính từ xa có thể thay đổi trực tiếp để khắc phục sự cố hay có lỗi thiết kế. Kiểu FPGA thích hợp nhất cho ứng dụng này là những FPGA có các chuyển mạch lập trình được.

1.4. Qui trình thiết kế FPGA

Quy trình thiết kế tổng quát trên FPGA được trình bày trong sơ đồ dưới đây [1,19]:



Hình 1-7 Sơ đồ quy trình thiết kế FPGA

- **Mô tả thiết kế (Design specification)**

Trong bước này, từ những yêu cầu của thiết kế và dựa trên khả năng của công nghệ hiện có, người thiết kế kiến trúc sẽ xây dựng nên toàn bộ kiến trúc tổng quan cho thiết kế. Nghĩa là trong bước này người thiết kế kiến trúc phải mô tả được những vấn đề sau:

- Thiết kế có những khối nào?
- Mỗi khối có chức năng gì?
- Hoạt động của thiết kế và của mỗi khối ra sao ?
- Phân tích các kỹ thuật sử dụng trong thiết kế và các công cụ, phần mềm hỗ trợ thiết kế.

Một thiết kế có thể được mô tả sử dụng ngôn ngữ mô tả phần cứng, như VHDL hay Verilog HDL hoặc có thể mô tả qua bản vẽ mạch (schematic capture). Một thiết kế có thể vừa bao gồm bản vẽ mạch mô tả sơ đồ khối chung, vừa có thể dùng ngôn ngữ HDL để mô tả chi tiết cho các khối trong sơ đồ.

- **Mô phỏng chức năng (Function simulation):**

Sau khi mô tả thiết kế, người thiết kế cần mô phỏng tổng thể thiết kế về mặt chức năng để kiểm tra thiết kế có hoạt động đúng với các chức năng yêu cầu.

- ***Tổng hợp logic (Logic Synthesis).***

Tổng hợp logic là quá trình tổng hợp các mô tả thiết kế thành sơ đồ bố trí mạch (netlist). Quá trình chia thành 2 bước: chuyển đổi các mã RTL, mã HDL thành mô tả dưới dạng các biểu thức đại số Boolean và dựa trên các biểu thức này kết hợp với thư viện tế bào chuẩn sẵn có để tổng hợp nên một thiết kế tối ưu.

- ***Hiệu chỉnh các kết nối (Datapath Schematic).***

Nhập netlist và các ràng buộc về thời gian vào một công cụ phân tích thời gian (timing analysic). Công cụ phân tích này sẽ tách rời tất cả các kết nối của thiết kế, tính thời gian trễ của các kết nối dựa trên các ràng buộc. Dựa trên kết quả phân tích (report) của công cụ phân tích, xác định các kết nối không thỏa mãn về thời gian. Tùy theo nguyên nhân dẫn đến không thỏa mãn mà ta có thể viết lại mã và tiến hành lại tổng hợp logic hoặc hiệu chỉnh lại các ràng buộc.

Thực thi (Implementation).

Ta đã có sơ đồ bố trí netlist mô tả tổng thể thiết kế tại mức cổng (chỉ gồm các cổng logic cơ bản và các mạch logic khác như: MUX). Quá trình này sẽ đặt sơ đồ netlist này lên chip, gọi là quá trình thực thi (Device Implementation).

Quá trình này gồm các bước:

Ánh xạ (mapping hay còn gọi fitting - ăn khớp) : chuẩn bị dữ liệu đầu vào, xác định kích thước các khối. Các khối này sẽ phải phù hợp với cấu trúc của 1 tế bào cơ bản của FPGA. (gồm nhiều cổng logic) và đặt chúng vào các vị trí tối ưu cho việc chạy dây.

Đặt khối và định tuyến (Place & Route):

+ Đặt khối: đặt các khối ánh xạ vào các tế bào (cell) ở vị trí tối ưu cho việc chạy dây.

+ Định tuyến: Bước này thực hiện việc nối dây các tế bào. Để thực hiện việc này, chúng ta cần có các thông tin sau:

- Các thông tin vật lý về thư viện tế bào, ví dụ kích thước tế bào, các điểm để kết nối, định thời, các trở ngại trong khi đi dây.
- Một netlist được tổng hợp sẽ chỉ ra chi tiết các instance và mối quan hệ kết nối bao gồm cả các đường dẫn bị hạn chế trong thiết kế.
- Tất cả các yêu cầu của tiến trình cho các lớp kết nối, bao gồm các luật thiết kế cho các lớp chạy dây, trở kháng và điện dung, tiêu thụ năng lượng, các luật về sự dẫn điện trong mỗi lớp.

- ***Quá trình Nạp (download) và lập trình (program).***

- Sau quá trình thực hiện, thiết kế cần được nạp vào FPGA dưới dạng dòng bit (bit stream).

- Quá trình nạp thiết kế (download) vào FPGA thường nạp vào bộ nhớ bay hơi, ví dụ như SRAM. Thông tin cấu hình sẽ được nạp vào bộ nhớ. Dòng bit được truyền lúc này sẽ mang thông tin định nghĩa các khối logic cũng như kết nối của thiết kế. Tuy nhiên, lưu ý rằng, SRAM sẽ mất dữ liệu khi mất nguồn nên thiết kế sẽ không lưu được đến phiên làm việc kế tiếp.

- Lập trình (program) là thuật ngữ để mô tả quá trình nạp chương trình cho các bộ nhớ không bay hơi, ví dụ như PROM. Như vậy, thông tin cấu hình vẫn sẽ được lưu trữ khi mất nguồn.

1.5. Ngôn ngữ lập trình trong FPGA

Có nhiều ngôn ngữ có thể lập trình cho FPGA như VHDL, Verilog, C, … Mỗi ngôn ngữ lại có ưu điểm và nhược điểm riêng. Ví dụ như Verilog là ngôn ngữ được phát triển và sử dụng chủ yếu ở Mỹ. Đây là ngôn ngữ rất gần với C, chính vì vậy sẽ rất thuận tiện cho ai đó đã quen với lập trình ngôn ngữ C. Ở châu Âu thì người ta lại quen dùng VHDL hơn, ưu điểm của ngôn ngữ này giúp người làm việc với nó sẽ có cái nhìn rất thấu đáo về phần cứng.

1.5.1. Ngôn ngữ VHDL

VHDL là ngôn ngữ mô tả phần cứng cho các mạch tích hợp tốc độ rất cao, là một loại ngôn ngữ mô tả phần cứng được phát triển dung cho chương trình VHSIC (*Very High Speed Itergrated Circuit*) của bộ quốc phòng Mỹ. Mục tiêu của việc phát triển VHDL là có được một ngôn ngữ mô phỏng phần cứng tiêu chuẩn và thống nhất cho phép thử nghiệm các hệ thống số nhanh hơn, cũng như cho phép dễ dàng đưa các hệ thống đó vào ứng dụng trong thực tế. Ngôn ngữ VHDL được ba công ty IBM, Intermetics, Texas Instruments bắt đầu nghiên cứu và phát triển vào tháng 7 năm 1983. Phiên bản đầu tiên được công bố vào tháng 8 năm 1985. Sau đó VHDL được đề xuất tổ chức IEEE xem xét thành một tiêu chuẩn chung. Năm 1987 đã đưa ra tiêu chuẩn về VHDL (tiêu chuẩn IEEE-1076-1987) [3,7].

VHDL được phát triển để giải quyết các khó khăn trong việc phát triển, thay đổi và lập tài liệu cho các hệ thống số. Như ta đã biết, một hệ thống số có rất nhiều tài liệu mô tả. Để có thể vận hành bảo trì sửa chữa một hệ thống ta cần tìm hiểu kỹ lưỡng tài liệu đó. Với việc mô phỏng phần cứng tốt việc xem xét các tài liệu mô tả trở nên dễ dàng hơn vì bộ tài liệu đó có thể được thực thi để mô phỏng hoạt động của hệ thống . như thế ta có thể xem xét toàn bộ các phần tử của hệ thống hoạt động trong một mô hình thống nhất.

VHDL được phát triển như một ngôn ngữ độc lập không gắn với bất kỳ một phương pháp thiết kế, một bộ mô tả hay công nghệ phần cứng nào. Người thiết kế có thể tự do lựa chọn công nghệ, phương pháp thiết kế trong khi chỉ sử dụng một ngôn ngữ duy nhất, và khi đem so sánh với các ngôn ngữ mô phỏng phần cứng khác đã kể ra ở trên ta thấy VHDL có một số ưu điểm hơn hẳn các ngôn ngữ khác:

- Tính công cộng: VHDL được phát triển dưới sự bảo trợ của chính phủ Mỹ và hiện nay là một chuẩn của IEEE. VHDL được hỗ trợ của nhiều nhà sản xuất thiết bị cũng như nhiều nhà cung cấp công cụ thiết kế mô phỏng hệ thống.
- Khả năng hỗ trợ nhiều công nghệ và phương pháp thiết kế: VHDL cho phép thiết kế bằng nhiều phương pháp. Ví dụ: phương pháp thiết kế từ trên xuống, hay từ dưới lên dựa vào các thư viện có sẵn. VHDL cũng hỗ trợ cho nhiều loại công cụ xây dựng mạch như sử dụng công nghệ đồng bộ hay không đồng bộ, sử dụng ma trận lập trình được hay sử dụng mảng ngẫu nhiên.
- Tính độc lập với công nghệ: VHDL hoàn toàn độc lập với công nghệ chế tạo phần cứng. Một mô tả hệ thống dùng VHDL thiết kế ở mức cổng có thể chuyển thành các bản tổng hợp mạch khác nhau tùy thuộc công nghệ chế tạo phần cứng mới ra đời, nó có thể được áp dụng ngay cho các hệ thống đã thiết kế.

- Khả năng mô tả rộng: VHDL cho phép mô tả hoạt động của phần cứng từ mức hệ thống số cho đến mức cổng. VHDL có khả năng mô tả hoạt động của hệ thống trên nhiều mức nhưng chỉ sử dụng một cú pháp chặt chẽ thống nhất cho mọi mức. Như thế ta có thể mô phỏng một bản thiết kế bao gồm cả các hệ thống con được mô tả chi tiết.
- Khả năng trao đổi kết quả: VHDL là một tiêu chuẩn được chấp nhận, nên một mô hình VHDL có thể chạy trên mọi bộ mô tả đáp ứng được tiêu chuẩn VHDL. Các kết quả mô tả hệ thống có thể được trao đổi giữa các nhà thiết kế sử dụng công cụ thiết kế khác nhau nhưng cùng tuân theo tiêu chuẩn VHDL. Cũng như một nhóm thiết kế có thể trao đổi mô tả mức cao của các hệ thống con trong một hệ thống lớn, trong đó các hệ thống con đó được thiết kế độc lập.
- Khả năng hỗ trợ thiết kế mức lớn và khả năng tái sử dụng lại các thiết kế: VHDL được phát triển như một ngôn ngữ lập trình bậc cao, vì vậy nó có thể được sử dụng để thiết kế một hệ thống lớn với sự tham gia của một nhóm nhiều người. Bên trong ngôn ngữ VHDL có nhiều tính năng hỗ trợ việc quản lý, thử nghiệm và chia sẻ thiết kế và nó cũng cho phép ta tái sử dụng lại các phần đã có sẵn.

Cấu trúc mô hình hệ thống mô tả bằng VHDL

Trong phần này em giới thiệu sơ qua về cấu trúc khung cơ bản của VHDL khi mô tả cho một mô hình thiết kế thực. thông thường mô hình VHDL bao gồm ba phần: Thực thể (*entity*), kiến trúc (*architecture*), các cấu hình, đôi khi ta sử dụng các gói (*packages*) và mô hình kiểm tra hoạt động của hệ thống (*testbench*).[15]

- **Thực thể (*entity*)**

Đây là nơi chứa các khai báo thực thể (là các port giao tiếp giữa FPGA và các tín hiệu bên ngoài các port này được sử dụng như là lớp vỏ của kiến trúc

thiết kế) và có thể bao gồm các tùy chọn “generic” là khai báo chung có thể dễ dàng sửa đổi khi cần.

- **Kiến trúc (*architecture*)**

Phần thứ hai trong mô hình VHDL là khai báo kiến trúc của chương trình. Mỗi một khai báo thực thể đều phải đi kèm với ít nhất một kiến trúc tương ứng. VHDL cho phép tạo ra hơn một kiến trúc cho một thực thể. Phần khai báo kiến trúc có thể bao gồm các khai báo về các tín hiệu bên trong, các phần tử bên trong hệ thống, hay các hàm và thủ tục mô tả hoạt động của hệ thống. Tên của kiến trúc là nhãn được đặt tùy theo người xử dụng. có hai cách mô tả kiến trúc của một phần tử (hoặc hệ thống) đó là mô hình hoạt động (*Behaviour*) hay mô tả theo mô hình cấu trúc (*Structure*). Tuy nhiên một hệ thống có thể bao gồm cả mô tả theo mô hình hoạt động và mô tả theo mô hình cấu trúc.

- **Mô tả kiến trúc theo mô hình hoạt động**

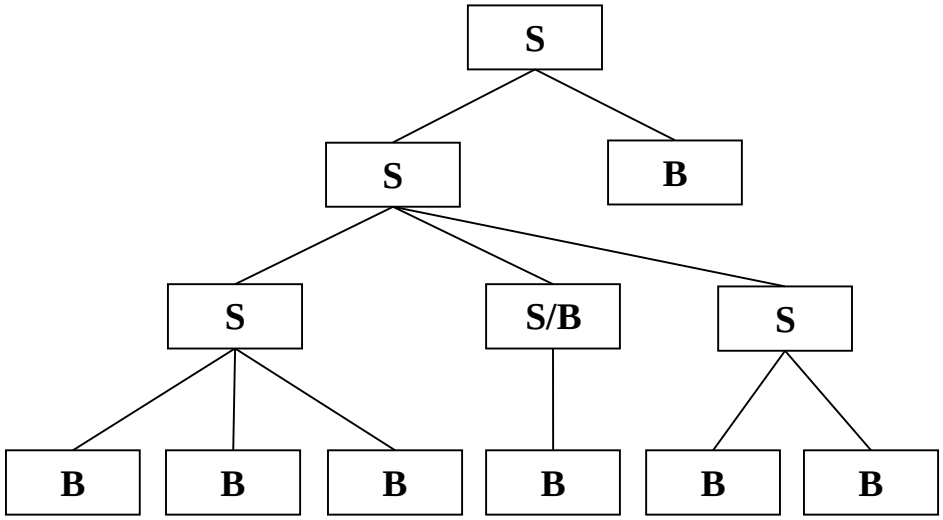
Mô hình hoạt động mô tả các hoạt động của hệ thống (hệ thống đáp ứng với các tín hệu vào như thế nào và đưa ra kết quả gì ở đầu ra) dưới dạng các cấu trúc ngôn ngữ lập trình bậc cao. Cấu trúc đó có thể là PROCESS, WAIT, IF, CASE, FOR-LOOP…

- **Mô tả kiến trúc theo mô hình cấu trúc**

Mô hình cấu trúc của một phần tử (hoặc hệ thống) có thể bao gồm nhiều cấp cấu trúc, bắt đầu từ một cổng logic đơn giản đến xây dụng mô tả cho một hệ thống hoàn thiện. Thực chất của việc mô tả theo mô hình cấu trúc là mô tả các phần tử con bên trong hệ thống và sự kết nối của các phần tử con đó. Ví dụ: mô tả mô hình cấu trúc một flip-flop RS gồm hai cổng NAND có thể mô tả cổng NAND được định nghĩa tương tự như ví dụ cổng NOT, sau đó mô tả sơ đồ móc nối các phần tử NAND tạo thành trigo RS.

- **Cấu trúc process**

Process là khối cơ bản của việc mô tả theo hoạt động. Process được xét đến như là một chuỗi các hoạt động đơn trong suốt quá trình dịch.



Hình 1-8 Cấu trúc process

S: Mô hình cấu trúc

B: Mô hình hoạt động

S/B: Mô hình kết hợp

Cấu trúc tổng quát

[*Process label*]

Process [(*sensitive_lish*)]

Process declarative part

Begin

...

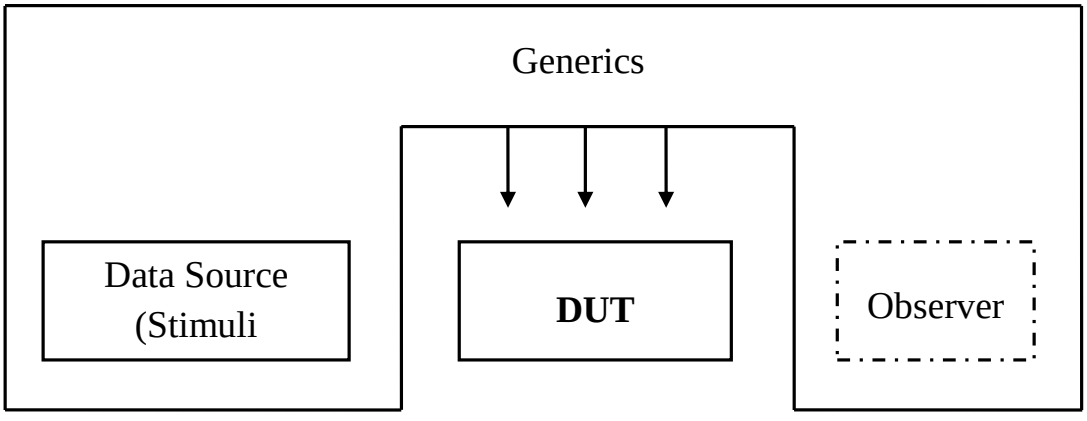
End Process

Trong đó các phần đặt trong dấu [ ] thì có thể có hoặc không.

- *Process label:* (nhãn lệnh) là tùy thuộc người lập trình đặt tên.
- *sensitive_lish:* Danh sách các yếu tố kích thích hoạt động.
- **Môi trường kiểm tra (*testbench*)**

Một trong các nhiệm vụ rất quan trọng là kiểm tra bản mô tả thiết kế. kiểm tra một mô hình VHDL được thực hiện bằng cách quan sát hoạt động của nó trong khi mô phỏng và các giá trị thu được có thể đem so sánh với yêu cầu thiết kế.

Môi trường kiểm tra có thể hiểu như một mạch kiểm tra ảo. môi trường kiểm tra sinh ra các tác động lên bản thiết kế và cho phép quan sát hoặc so sánh kết quả hoạt động của bản mô tả thiết kế. Thông thường thì các bản mô tả đều cung cấp chương trình thử. Nhưng ta cũng có thể tự xây dựng chương trình thử (*testbench*). Mạch thử thực chất là sự kết hợp của tổng hợp nhiều thành phần. Nó gồm ba thành phần: Mô hình VHDL đã qua kiểm tra, nguồn dữ liệu và bộ quan sát. Hoạt động của mô hình VHDL được kích thích bởi các nguồn dữ liệu và kiểm tra tính đúng đắn thông qua bộ quan sát.



Hình 1-9: sơ đồ khối của testbench

Trong đó: DUT: (*device under test*) mô hình VHDL cần kiểm tra.

Observer: Khối quan sát kết quả.

Data source: Nguồn dữ liệu (Khối tạo ra các tín hiệu kích thích).

1.5.2. Ngôn ngữ Verilog

- Behavior Modeling: là một thành phần được mô tả bởi đáp ứng Input/output của nó.
- Structural Modeling: là một thành phần được mô tả bởi các kết nối mức thấp giữa các thành phần con của mạch.

- **Behavior Modeling**

Trong Behavior Modeling, bạn sẽ mô tả chức năng của mạch chứ không phải cấu trúc của mạch. Hành vi output được mô tả theo mối quan hệ với các input. Phía dưới là một ví dụ về HDL code hướng hành vi. Ở đây, mô tả thao tác dịch bit. Kiểu mô hình này dựa vào synthesis engine để tạo ra mạch thực hiện đúng như hành vi đã được mô tả.

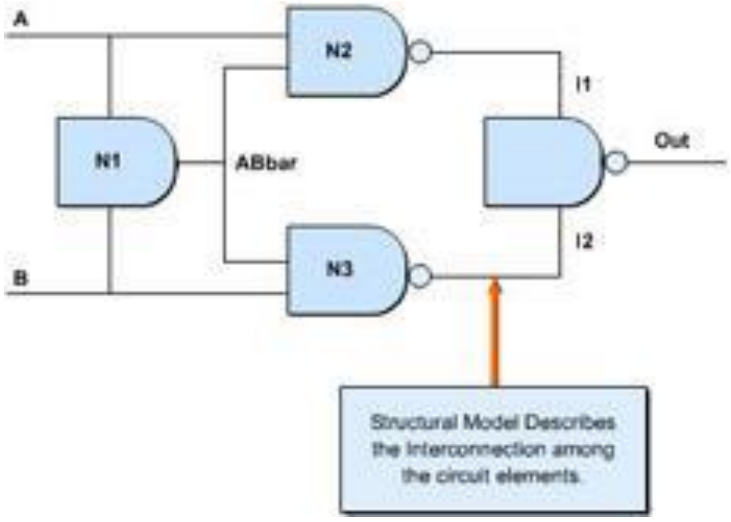
Mã:

```
if (en_shf) begin
  data_out <= data_in << 2;
end
else begin
  data_out <= data_out_next;
end
```

- **Structural Modeling**

Theo hướng cấu trúc, cả chức năng và cấu trúc của mạch được xác định rõ, các kỹ sư viết HDL code sẽ gọi ra các thành phần phần cứng thực tế và nối dây chúng lại với nhau. Các thành phần phần cứng có thể đơn giản chỉ là cổng AND hoặc cổng OR, hay thậm chí cũng có thể là một module biểu diễn các mức

trừu tượng khác. Trong một thiết kế hiện đại điển hình, bạn sẽ tìm các thành phần của cả mô hình hướng hành vi và mô hình hướng cấu trúc.



Hình 1-10 Mô hình cấu trúc mô tả kết nối các thực thể

- **RTL Synthesis**

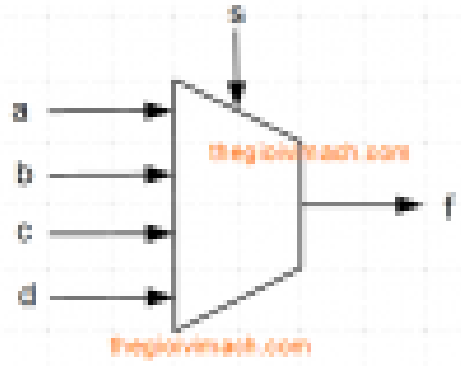
Hình dưới minh hoạ tiến trình tổng hợp RTL. Ở bước đầu tiên, synthesis engine chuyển đổi code của bạn thành phần cứng bằng cách sử dụng các thành phần kiến trúc được có sẵn trong thư viện. Sau đó nó sẽ đến bước tối ưu hoá để đảm bảo bạn có mô tả mạch có thể hiện thực tốt nhất.

Xem xét một ví dụ bên dưới, khi người kỹ sư viết mô tả cho một mạch như bên dưới (mã verilog), thì sau đó các công cụ synthesis đầu tiên sẽ tổng hợp ra một mạch với các thành phần có sẵn trong thư viện, bước kế tiếp là các công cụ synthesis sẽ sử dụng các thuật toán tối ưu dựa trên các ràng buộc mà người kỹ sư đưa vào để tối ưu lại mạch như trong Hình 3.4, dễ dàng nhận ra đoạn mã Verilog trên mô tả cho một bộ MUX như hình bên dưới.

Mã:

```
always @(a, b, c, d, s) begin
  case (s)
```

```
    2'b00: begin
      f = a;
    end
    2'b01: begin
      f = b;
    end
    2'b10: begin
      f = c;
    end
    2'b11: begin
      f = d;
    end
  endcase
end
```



Hình 1-11 Mô hình bộ dồn kênh

Trong ví dụ này, câu lệnh case được thực thi trong một khối always điển hình cho một multiplexer. Phần dưới em sẽ trình bày rõ về những loại cấu trúc như thế.

- **RTL Synthesis & RTL Simulation Flow điển hình**

Mô hình Verilog được viết có thể được gửi thông qua 2 nhánh khác nhau, synthesis hoặc simulation. Theo synthesis flow, trình biên dịch synthesis (ví dụ Synplify của Synopsys hay Native Synthesis Engine của Altera) sẽ sử dụng công nghệ có sẵn trong thư viện thiết bị để chuyển đổi và tối ưu hoá mô hình Verilog

của bạn thành một công nghệ netlist cụ thể mà có thể được sử dụng cho Timing Analysis hay P&R cho thiết bị cụ thể. Theo simulation flow, sử dụng cho ví dụ Mentor Graphics Modelsim, trình biên dịch simulation sẽ mô phỏng mô hình Verilog với các bổ sung được cung cấp từ Verilog testbench hay thông qua các test vector. Hầu hết các synthesis tool cũng xuất ra một tập tin post-synthesis Verilog sao cho các kỹ sư có thể kiểm tra kế quả synthesis trước khi thực hiện việc sắp xếp và đi dây (P&R). Trong trường hợp này, tập tin Verilog được xuất ra bởi synthesis tool đều có thể thay thế tập tin Verilog ban đầu trong trình mô phỏng. Testbench và test vector có thể được sử dụng cho việc kiểm tra này. Thêm vào đó, các tập tin mô hình simulation có thể được yêu cầu phụ thuộc vào synthesis tool. Chúng có thể được cung cấp bởi nhà cung cấp synthesis tool hoặc được tạo bởi synthesis tool cùng với mô hình post-synthesis.

CHƯƠNG II: MỘT SỐ KỸ THUẬT XỬ LÝ ẢNH

2.1. Kỹ thuật nâng cấp ảnh

Nâng cao chất lượng ảnh là một bước quan trọng, tạo tiền đề cho xử lý ảnh. Mục đích chính là làm nổi bật một số đặc tính của ảnh như thay đổi độ tương phản, lọc nhiễu, nổi biên, làm trơn biên ảnh … Có nhiều loại nhiễu khác và cũng có nhiều phương pháp lọc nhiễu khác nhau. Tuy nhiên tùy từng loại nhiễu mà áp dụng phương pháp lọc cho phù hợp.

Các thuật toán triển khai việc nâng cao chất lượng ảnh hầu hết dựa trên các kỹ thuật trên miền điểm, không gian và tần số.

2.1.1. Toán tử điểm

Xử lý điểm ảnh thực chất là biến đổi giá trị của một điểm ảnh dựa vào giá trị của chính nó mà không dựa vào các điểm ảnh khác.

Có hai cách tiếp cận với phương pháp này: Cách thứ nhất dùng một hàm biến đổi thích hợp với mục đích hoặc yêu cầu đặt ra để biến đổi giá trị mức xám của điểm ảnh sang một giá trị mức xám khác. Cách thứ hai là dùng lược đồ mức xám (Gray Histogram). Về mặt toán học, toán tử điểm là một ánh xạ từ giá trị cường độ ánh sáng u(m,n) tại tọa độ (m,n) sang giá trị cường độ ánh sáng khác v(m,n) thông qua hàm f(.) tức là:

$$v(m,n) = f(u(m,n))$$

Ứng dụng chính của toán tử điểm là biến đổi độ tương phản của ảnh. Một số dạng toán tử điểm được giới thiệu như sau:

2.1.1.1 Kỹ thuật biến đổi âm bản

$$O(m,n) = 255 – I(m,n)$$

Với O(m,n): Giá trị điểm ảnh đầu ra tại vị trí (m,n). I(m,n): Giá trị điểm ảnh đầu vào tại vị trí (m,n).

2.1.1.2 Kỹ thuật thay đổi độ xám

O(m,n) = I(m,n) + C

C = const, C_{max}= 255 và C_{min}= -255 Nếu C dương : Tăng độ sáng

Nếu C âm: Giảm độ sáng

2.1.1.3 Thay đổi độ tương phản

Trước tiên ta cần làm rõ khái niệm độ tương phản. Ảnh số là tập hợp các điểm, mà mỗi điểm có giá trị độ sáng khác nhau. Ở đây độ sáng để mắt người dễ cảm nhận ảnh song không phải là quyết định. Thực tế chỉ ra rằng hai đối tượng có cùng độ sáng nhưng đặt trên hai nền khác nhau sẽ cho cảm nhận khác nhau. Như vậy độ tương phản biểu diễn sự thay đổi độ sáng của đối tượng so với nền hay độ tương phản là độ nổi của điểm ảnh hay vùng ảnh so với nền. Với định nghĩa này nếu ảnh của ta có độ tương phản kém, ta có thể thay đổi tùy theo ý muốn.

Ta có công thức thay đổi độ tương phản

như sau: O(m,n) = a*I(m,n) + C

2.1.2 Toán tử không gian

Đây là toán tử khi tác động vào điểm ảnh thì nó quan tâm tới các điểm lân cận.

Toán tử được thực hiện thông qua một phép nhân chập và mẫu.

Giả sử ta có ảnh I(x,y), một mẫu T(k,l), khi đó ảnh I nhân chập với mẫu T được định nghĩa như sau:

2.2. Một số kỹ thuật lọc nhiễu

Thường là ảnh thu nhận có nhiễu cần phải loại bỏ nhiễu hay ảnh không sắc nét bị mờ hoặc cần làm tõ các chi tiết như đường biên ảnh. Các toán tử không gian dùng trong kỹ thuật tăng cường ảnh được phân nhóm theo công dụng: làm trơn nhiễu, nổi biên. Để làm trơn nhiễu hay tách nhiễu, người ta sử

dụng các bộ lọc tuyến tính (lọc trung bình, thông thấp) hay lọc phi tuyến (trung vị, giả trung vị, lọc đồng hình). Từ bản chất của nhiễu (thường tương ứng với tần số cao) và từ cơ sở lý thuyết lọc là: bộ lọc chỉ cho tín hiệu có tần số nào đó thông qua do đó, để lọc nhiễu người ta thường dùng lọc thông thấp (theo quan điểm tần số không gian) hay lấy tổ hợp tuyến tính để san bằng (lọc trung bình). Để làm nổi cạnh (ứng với tần số cao), người ta dùng các bộ lọc thông cao, lọc Laplace. Để áp dụng được các phương pháp lọc nhiễu phù hợp trước hết cần phải hiểu thế nào là nhiễu, có những loại nhiễu nào và đặc điểm của từng loại nhiễu.

Nhiễu là những phần tử ảnh mà giá trị của nó trội so với các phần tử xung quanh. Xét theo tần số, các phần tử nhiễu có tần số cao hơn so với các điểm ảnh xung quanh. Nhiễu thường gặp trong ảnh gồm các loại nhiễu sau:

Nhiễu cộng : nhiễu cộng thường phân bố khắp ảnh. Nếu ta gọi ảnh quan sát(ảnh thu được) là X_qs, ảnh gốc là X_gốc và nhiễu là η. Ảnh thu được có thể biểu diễn bởi:

X_qs = X_gốc + η.

Nhiễu nhân : Nhiễu nhân thường phân bố khắp ảnh. Nếu ta gọi ảnh quan sát(ảnh thu được) là X_qs, ảnh gốc là X_gốc và nhiễu là η. Ảnh thu được có thể biểu diễn bởi:

X_qs = X_gốc × η.

Nhiễu xung: Là sự kết hợp của nhiễu muối và nhiễu tiêu. Có 2 loại: Nhiễu xung đơn cực và nhiễu xung lưỡng cực. Nhiễu xung lưỡng cực có hàm phân bố là:

$$P(z) = \begin{cases} P_a & z = a \\ P_b & z = b \\ 0 \quad Trường\ còn\ lại \end{cases}$$

Nếu b>a, mức xám b xuất hiện như là điểm sáng của ảnh, a là điểm tối. Nếu a>b, ngược lại. a=b=0 là nhiễu xung đơn cực.

Nhiễu muối tiêu (Salt-pepper noise) - một ví dụ điển hình nhất của loại nhiễu xung này – sẽ cho thấy rõ hơn tính chất "đột biến" của nó. Các điểm ảnh bị nhiễu (noise pixel) có thể nhận các giá trị cực đại hoặc cực tiểu trong khoảng giá trị [0, 255]. Với ảnh mức xám (gray scale), nếu một điểm ảnh có giá trị cực đại (tức cường độ sáng bằng 255) thì nó sẽ tạo ra một đốm trắng trên ảnh, trông giống như hạt "muối". Và ngược lại nếu một điểm ảnh có giá trị cực tiểu (tức cường độ sáng bằng 0) thì sẽ tạo ra một đốm đen, giống như "tiêu". Vậy nên còn gọi là ảnh muối tiêu. Thông thường, khi nói một ảnh nhiễu muối tiêu 30% nghĩa là trong đó tỉ lệ các điểm ảnh nhiễu mang giá trị cực tiểu là 15% và cực đại là 15%.

Nhiễu Gaussian: Bởi vì khả năng dễ ứng dụng toán của nó trong cả lĩnh vực không gian và tần số, nhiễu Gaussian được sử dụng phổ biến trong thực tiễn. Có hàm phân bố là:

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(z-\mu)^2/2\sigma^2}$$

Trong đó, z biểu diễn mức xám, μ là giá trị trung bình của z,σ là độ lệch tiêu chuẩn, σ^2 là phương sai của z.

trong đó:

$$\mu = \frac{a+b}{2}$$

$$\sigma^2 = \frac{(b-a)^2}{12}.$$

Nhiễu Uniform: Được cho bởi:

$$p(z) = \begin{cases} \frac{1}{b-a} & \text{if } a \le z \le b \\ 0 & \text{otherwise.} \end{cases}$$

2.2.1. Kỹ thuật lọc trung bình

Với lọc trung bình, mỗi điểm ảnh được thay thế bằng trung bình trọng số của các điểm lân cận.

Tư tưởng của thuật toán lọc trung bình: ta sử dụng một cửa sổ lọc (ma trận 3x3) quét qua lần lượt từng điểm ảnh của ảnh đầu vào input. Tại vị trí mỗi điểm ảnh lấy giá trị của các điểm ảnh tương ứng trong vùng 3x3 của ảnh gốc "lấp" vào ma trận lọc. Giá trị điểm ảnh của ảnh đầu ra là giá trị trung bình của tất cả các điểm ảnh trong cửa sổ lọc. Việc tính toán này khá đơn giản với hai bước gồm tính tổng các thành phần trong cửa sổ lọc và sau đó chia tổng này cho số các phần tử của cửa sổ lọc.

Lọc trung bình có trọng số chính là thực hiện chập ảnh đầu vào với nhân chập H. Nhân chập H trong trường hợp này có dạng:

$$H = \frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Trong lọc trung bình, thường người ta ưu tiên cho các hướng để bảo vệ biên của ảnh khỏi bị mờ khi làm trơn ảnh. Các kiểu mặt nạ được sử dụng tùy theo các trường hợp khác nhau. Các bộ lọc trên là bộ lọc tuyến tính theo nghĩa là điểm ảnh ở tâm cửa số sẽ được thay bởi tổ hợp các điểm lân cận chập với mặt nạ. Giả sử đầu vào biểu diễn bởi ma trận I:

$$\mathrm{I} = \begin{bmatrix}4 & 7 & 3 & 7 & 1\\ 5 & 7 & 1 & 7 & 1\\ 6 & 6 & 1 & 8 & 3\\ 5 & 7 & 5 & 7 & 1\\ 5 & 7 & 6 & 1 & 2\end{bmatrix}$$

Ảnh số thu được bởi lọc trung bình Y=H⊗I có dạng:

$$\mathrm{Y} = \frac{1}{9}\begin{bmatrix}23 & 26 & 31 & 19 & 16\\ 35 & 39 & 46 & 31 & 27\\ 36 & 43 & 49 & 34 & 27\\ 36 & 48 & 48 & 34 & 22\\ 24 & 35 & 33 & 22 & 11\end{bmatrix}$$

Trong phương pháp lọc trung bình, với các nhiễu ảnh dàn đều trên toàn ảnh có khả năng làm mờ nhiễu, giảm sự khác biệt về mức sáng giữa các điểm ảnh của ảnh gốc và các điểm ảnh bị nhiễu tác động. Bộ lọc trung bình có vai trò làm trơn ảnh có thể xem như bộ lọc thông cao, có xu hướng cân bằng cường độ sáng các điểm ảnh. Phương pháp lọc trung bình cho kết quả nhanh hơn lọc trung vị vì không phải sắp xếp thứ tự các điểm ảnh. Phương pháp này phù hợp với các loại nhiễu ngẫu nhiên như nhiễu Gaussian hoặc nhiễu Uniform.

2.2.2. Kỹ Thuật lọc trung vị

Các bộ lọc phi tuyến cũng hay được dùng trong kỹ thuật tăng cường ảnh. Trong kỹ thuật này, người ta dùng bộ lọc trung vị, giả trung vị, lọc ngoài. Với lọc trung vị, điểm ảnh đầu vào sẽ được thay thế bởi trung vị các điểm ảnh còn lọc giả trung vị sẽ dùng trung bình cộng của 2 giá trị "trung vị" (trung bình cộng của max và min). Trung vị được viết với công thức:

$V(m,n) = Trungvi(y(m-k,n-l)$ với $\{k,l\} \in W$

Kỹ thuật này đòi hỏi giá trị các điểm ảnh trong cửa sổ phải xếp theo thứ tự tăng hay giảm dần so với giá trị trung vị. Kích thước cửa số thường được chọn sao cho số điểm ảnh trong cửa số là lẻ. Các cửa sổ hay dùng là cửa sổ có kích thước 3x3, hay 5x5 hay 7x7. Thí dụ:

Nếu y(m) = {2, 3, 8, 4, 2} và cửa sổ W=(-1, 0, 1), ảnh thu được sau lọc trung vị sẽ là: v(m) = (2, 3, 4, 4, 2).

do đó:

v[0] = 2 <giá trị biên> v[1] = trungvi(2,3,8)=3 v[2]=Trungvi(3,4,8)=4

v[3]=Trungvi(8,4,2)=4 v[4] = 2 <Giá trị biên>.

Tính chất của lọc trung vị:

- Lọc trung vị là loại lọc phi tuyến.
- Có lợi cho việc loại bỏ các điểm ảnh hay các hàng mà vẫn bảo tòan độ phân giải.
- Hiệu quả giảm khi số điểm trong cửa sổ lớn hay bằng một nửa số điểm trong cửa sổ.

Phương pháp lọc trung vị phù hợp với nhiễu sạn, do nhiễu sạn có các phần tử của nhiễu nhỏ nên xác xuất điểm nằm gần điểm nhiễu trở thành trung vị là rất cao. Do vậy, các điểm nhiễu hầu như sẽ được thay thế bởi thông tin của các điểm lân cận. Phương pháp lọc trung vị có hiệu qủa khá cao và áp dụng được với nhiều loại ảnh có nhiễu khác nhau. Quá trình lọc nhiễu không làm ảnh hưởng nhiều tới ảnh gốc, ít làm mờ ảnh so với các bộ lọc làm trơn tuyến tính. Tuy nhiên phương pháp này đòi hỏi các điểm ảnh phải được sắp xếp thứ tự tăng dần hoặc giảm dần.

2.3. Kỹ thuật phân ngưỡng

2.3.1. Kỹ thuật phân ngưỡng tự động

Cơ sở của kỹ thuật này dựa theo nguyên lý trong vật lý. Dựa vào entropy (nguyên lý thống kê), dựa vào toán học, dựa vào các điểm cực trị địa phương để tách.

- Giả sử có ảnh I(MxN)

- G là số mức xám của ảnh (trên lý thuyết).

- Gọi t(g) là số điểm ảnh có mức xám ≤ g momen quán tính trung bình có mức xám nhỏ hơn hoặc bằng các mức xám g.

$$M(g) = \frac{1}{t(g)} \sum_{i=0}^{g} ih(i)$$

$$T(g) = \sum_{i=0}^{g} H(i)$$

Hàm f: g -> f(g)

Hàm được tính như sau:

$$f(g) = \frac{t(g)}{MxN - t(g)} [M(g) - M(G-1)]^2$$

Tìm ra một giá trị θ nào đó sao cho f đạt max khi đó θ là ngưỡng cần tìm (f(θ) = max => θ là ngưỡng).

2.3.2. Kỹ Thuật sử dụng các điểm biên

Điểm biên là điểm mà ở đó có sự thay đổi đột ngột về giá trị mức xám. Nó là điểm nằm ở biên giới của các đối tượng ảnh hay giữa các đối tượng ảnh và nền.

Do mức xám của các điểm biên sẽ thể hiện được các vùng tốt hơn nên biểu đồ mức xám của các điểm biên sẽ cho kết quả chính xác hơn so với biểu đồ mức xám tổng thể.

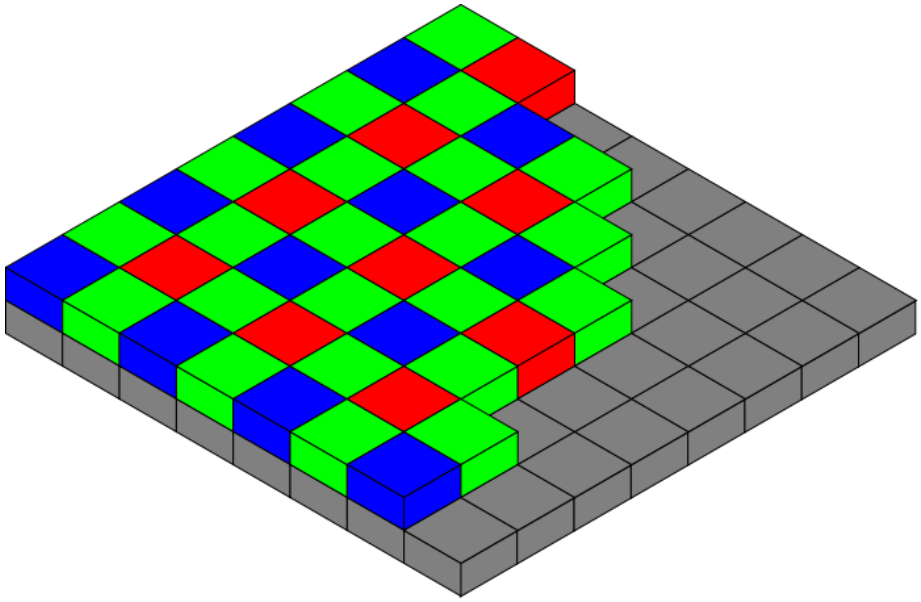
Việc xác định ngưỡng dựa trên toán tử dò biên vô hướng laplace. Ngưỡng được xác định trước hết bằng cách tính laplace của ảnh đầu vào. Cách đơn giản nhất là nhân chập với mặt nạ sau đây:

$$H = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Lúc này ta có một biểu đồ mức xám của ảnh ban đầu mà ta chỉ quan tâm tới các điểm ảnh có giá trị laplace lớn, những điểm ảnh trong nhóm 85% hoặc lớn hơn sẽ nằm trong biểu đồ này, còn các điểm khác thì không. Ngưỡng vừa sử dụng sẽ được tìm trong biểu đồ mức xám vừa tìm được

2.4. Bộ lọc màu Bayer Pattern và thuật toán Demosaic

Bộ lọc Bayer được sử dụng trong hầu hết trong các camera kỹ thuật số, camcorder và máy quét để tạo một ảnh màu. Lưới lọc bayer chỉ cho phép 3 màu đi qua: Red (R), Green (G), Blue (B) đi qua trong đó 50% là màu Green, 25% màu Red và 25% màu Blue



Hình 2-1 : Bộ lọc bayer trên cảm biến ảnh.

- **Giải thuật Demosaicing của bộ lọc Bayer:**

Demosaicing là quá trình chuyển lưới màu Bayer thành hình ảnh gồm thông tin đầy đủ về tất cả các màu ở mỗi điểm ảnh. Một cách để nhận diện đó là

xem một ô 2x2 gồm màu Red, Green và Blue là một khoang màu đầy đủ. Cách này cũng là một cách hay tuy nhiên hầu hết máy ảnh đều thực hiện thêm các bước để thu được thêm thông tin hình ảnh từ lưới lọc màu này. Nếu máy ảnh nhận diện màu theo ô 2x2 thì chỉ có thể đạt được độ phân giải bằng một nửa ở cả chiều dọc và ngang so với độ phân giải thực sự của các cảm biến.

Cũng có thuật toán demosaic khác có thể thu được độ phân giải cao hơn, đem tới hình ảnh ít bị nhiễu hơn hoặc tính toán chính xác hơn hình ảnh tại mỗi vị trí.

2.5. Ma trận sửa màu

Do có nhiều biến đổi khác nhau do khó khăn trong việc tái hiện màu trong các hệ thống ảnh, cụ thể:

- Các đặc trưng quang phổ của ống kính, bộ lọc.
- Thay đổi nguồn chiếu sáng như ánh sáng tự nhiên, đèn huỳnh quang, đèn sợi đốt.
- Các đặc điểm của các bộ lọc màu của cảm biến.

Ma trận sửa màu cung cấp một phương pháp sửa dữ liệu ảnh cho các biển đổi trên, ma trận này hoạt động trên hệ RGB. Ví dụ một trong 3 màu trong hệ thống xử lý ảnh từ nguồn ánh sáng ban đầu là mầu xanh dương (blue). Màu này là một sự kết hợp của các photon màu xanh nhân với sự đáp ứng tương đối của bộ lọc màu xanh, nhân với sự đáp ứng tương đối của silicon với các photon màu xanh. Tuy nhiên, sự đáp ứng của bộ lọc này và silicon khá khác nhau so với sự đáp ứng của mắt người, do đó màu xanh của cảm biến khá khác biệt so với màu xanh của con người.

Sự khác biệt này có thể sửa được và tái tạo gần chính xác với cái nhìn của con người. Lõi ma trận sửa màu nhan giá trị điểm ảnh với một hệ số tăng cường hoặc suy yếu để tạo ra một kết quả hiệu quả. Đồng thời màu xanh cũng được pha trộn thêm màu xanh là và màu đỏ. Màu xanh được tính theo công thức sau:

[out_b] = a1*R + a2*G + a3*B

Trong đó a1, a2, a3 là các tỉ lệ pha trộn của các màu đỏ, xanh lá, và xanh dương để tạo ra màu xanh ở đầu ra.

Mở rộng khái niệm này, một ma trận 3x3 được áp dụng cho mỗi màu một cách song song cùng lúc. Ma trận này có tỉ lệ pha trộn màu được xác định rõ.

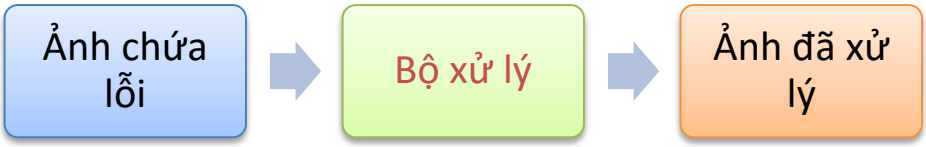
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} x \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

CHƯƠNG III: ỨNG DỤNG CÔNG NGHỆ FPGA XỬ LÝ DỮ LIỆU ĐA PHƯƠNG TIỆN DẠNG ẢNH

Mục tiêu của chương này là Thiết kế lõi IP xử lý ảnh theo chuẩn giao tiếp Avalon của hãng Altera. Lõi IP gồm các chức năng: xác định các điểm ảnh lỗi và sửa lại các điểm ảnh bị lỗi đó bằng kỹ thuật mặt nạ trung bình, Dùng thuật toán Demosaicing để nội suy màu và sửa màu bằng ma trân sửa màu. Tất cả các module của lõi IP này đều được thiết kế bằng ngôn ngữ Verilog HDL. Lõi IP có thể tích hợp trong hệ xử lý dữ liệu đa phương tiện nhằm tăng tốc độ xử lý của hệ thống. Do thời gian nghiên cứu còn ngắn, thiếu thiết bị nên em chỉ thực hiện chạy demo trên Kit DE2 với loại dữ liệu ảnh tĩnh, chưa thực hiện được trên hệ thống thời gian thực.

3.1. Thiết lập bài toán thử nghiệm:

Trong hệ thống xử lý ảnh gồm những thành phần cơ bản bộ thu thập dữ liệu, bộ xử lý và bộ hiển thị hình ảnh. Các dữ liệu hình ảnh có thể được thu trực tiếp từ các camera, những hình ảnh dữ liệu thu được luôn xuất hiện những điểm ảnh lỗi (nhiễu) theo nhiều mức độ khác nhau. Các nhiễu nhiễu này có thể là nhiễu ngẫu nhiên, nhiễu cố định hoặc nhiễu dải. Nhiễu ngẫu nhiên thường khó loại bỏ mà không làm giảm chất lượng ảnh, máy tính thường khó khăn khi phát hiện nhiễu ngẫu nhiên. Nhiễu cũng được tạo ra từ thành phần khác là màu sắc và độ sáng. Các loại nhiễu có thể loại bỏ bằng các thuật toán.



Xử lý ảnh hoặc video trên máy tính thường không mang lại hiệu quả cao mặc dù có thể thực hiện xử lý ảnh bằng phương pháp song song. Vì xử lý ảnh hay video yêu cầu phải xử lý lượng dữ liệu lớn với tốc độ cao. Tuy nhiên, việc xử lý dữ liệu lớn với tốc độ cao lại rất thích hợp với FPGA có khả năng xử lý song song. FPGA có thể xử lý dữ liệu lớn với vài chu kỳ clock.

Vì những lý do trên trong luận văn này em thực hiện thiết kế hệ thống xử lý ảnh bằng công nghệ FPGA.

3.2. Lựa chọn phần cứng và công cụ thử nghiệm:

3.2.1. Giới thiệu kit DE2:

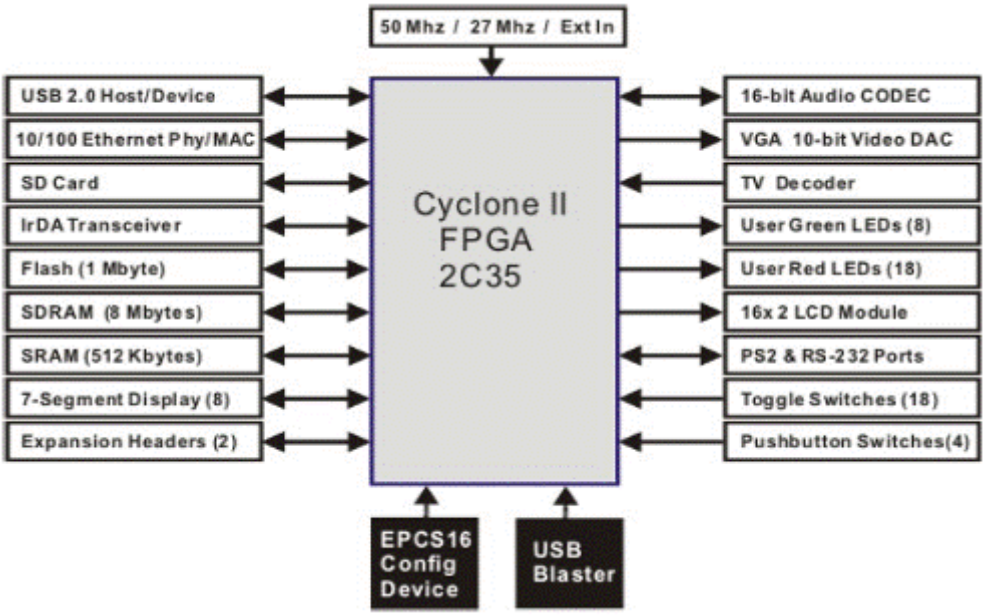
Kit DE2 cung cấp một phương tiện tối ưu để nghiên cứu về kĩ thuật số, cấu trúc máy tính và FPGA. Kit này sử dụng những công nghệ mới nhất cả về

phần cứng lẫn công cụ CAD (computer Aid Design) giúp nghiên cứu được nhiều ứng dụng khác nhau. Kit cung cấp nhiều đặc điểm phù hợp cho công việc nghiên cứu cũng như phát triển những hệ thống số thông thường hay những hệ thống multimedia phức tạp. Kit DE2 gồm những linh kiện chính sau:

- Chip Cyclone II 2C35 FPGA 672 pins. Tất cả những linh kiện trên kit đều được kết nối sẵn với những pin của FPGA, điều này cho phép người sử dụng có thể điều khiển tất cả những linh kiện cũng như ứng dụng của chúng.
- Rom EPCS16 – Dùng thiết lập cấu hình ban đầu cho thiết bị, hoạt động nối tiếp.
- USB Blaster – Dùng để cài đặt chương trình từ computer cho FPGA, hỗ trợ 2 chế độ: JTAG và AS (Active Serial).
- 512 – Kbyte SRAM
- 8 – Mbyte SDRAM
- – Mbyte Flash memory
- Khe cắm thẻ nhớ SD card.
- 18 toggle switches
- push-button switches
- 18 red LEDs
- 9 green LEDs
- LED 7 đoạn (7-segments displays)
- LED hiển thị kí tự dùng LCD (16x2 character displays)
- Nguồn xung clock 50 MHz và 27 MHz.
- 24-bit CD-quality audio CODEC với những đầu cắm line-in, line-out, và
- microphone-in.
- VGA DAC (10-bit high-speed triple DACs) với đầu cắm VGA-out.
- TV Decoder (NTSC/PAL) với đầu cắm TV-in.
- Giao tiếp chuẩn RS-232 với đầu cắm 9 pin.
- Giao tiếp chuẩn PS/2 cho chuột và bàn phím.
- Giao tip USB 2.0 (cả host lẫn device).
- Giao tiếp Ethernet 10/100
- Giao tiếp hồng ngoại (IrDA)
- Hai cổng kết nối header dùng để giao tiếp với những thiết bị ngoại vi khác mà người sử dụng muốn kết nối vào kit.

Đi kèm với những đặc tính phần cứng, Altera cũng cung cấp những giao tiếp I/O chuẩn và bảng điều khiener việc truy xuất những linh kiện trên KIT dựa trên phần mềm DE2 Control Panel.

Hình dưới mô tả các thành phần trên Kit DE2 của hãng Altera.



Hình 3-1. Sơ đồ khối Kit DE2

- Chip Cyclone II 2C23 FPGA gồm các bộ phận:
  - 33216 Les
  - 105 khối nhớ M4K
  - 483840 bít nhớ
  - 35 embeded multipliers
  - PLLs
  - 475 I/O pins
  - Fineline BGA 672 – pin package.
- Serial configuration device và USB blaster circuit:

Rom EPCS16 swerial configuration device, USB blaster for programing và user API control, JTAG và AS programming modes

- SRAM

512 Kbyte SRAM memory chip. Được tổ chức 256K x 16 bits, Có thể truy cập như là bộ nhớ cho vi xử lí Nios II hoặc truy cập thông qua bảng điều khiển Control Panel

- SDRAM:

8 Mbyte single data rate synchronous dynamic RAM, Được tổ chức 1M x 16 bits x 4 Banks. Có thể truy cập như là bộ nhớ cho vi xử lí Nios II hoặc truy cập thông qua bảng điều khiển Control Panel.

- Flash memory:

4 Mbyte NOR flash memory, 8 bit data bus. Có thể truy cập như là bộ nhớ cho vi xử lí Nios II hoặc truy cập thông qua bảng điều khiển Control Panel

- Khe cắm thẻ nhớ SD card:

Truy xuất SD card bằng mode SPI, có thể truy cập như là bộ nhớ cho vi xử lí Nios II với DE2 SD card driver.

- Pushbutton switches:

4 pushbutton switches, hồi phục lại tín hiệu bằng mạch Schmitt trigger. Ở trọng thái bình thường, tín hiệu ở mức cao; khi switch được nhấn, tín hiệu tạo ra một xung tích cực mức thấp và hồi phục lại trạng thái bính thường mức cao.

- Toggle switches:

18 toggle switches. Khi switch ở vị down thì tín hiệu ở mức thấp, ngược lại thì tín hiệu ở mức cao.

- Clock Inputs:

- Nguồn xung clock 50MHz, Nguồn xung clock 27MHz. Có thể sử dụng nguồn xung clock ngoài thông qua chân SMA.

- Audio Codec:

Wolfson WM8731 24-bit sigma-delta audio CODEC, Đầu cắm Line-in, Line-out, Microphone-in. Tần số lấy mẫu: 8 – 96 KHz, được ứng dụng cho Mp3 players, recorders, PDAs, smart phones, voice recorders…

- VGA output:

Sử dụng ADV7123 240 MHz triple 10 bit high speed video DAC. Với đầu cắm 15 pin high density D-sub, hỗ trợ độ phân giải 1600x1200 tại 100-Hz refresh rate. Có thể kết hợp với Cyclone II FPGA để thực thi một TV encoder tốc độ cao. NTSC/PAL TV decoder circuit: Sử dụng ADV7181B Multi-format SDTV video decoder, hỗ trợ NTSC (M,J,4.43), PAL (B,D,G,H,I,M,N),

SECAM. Tích hợp 3 ADC 9 bit 54MHz, hoạt động với nguồn xung clock 27 MHz, hỗ trợ composite video, hỗ trợ ngõ ra digital (8bit/16bit.

- Bộ điều khiển 10/100 ethernet:

Tích hợp MAC và PHY với giao tiếp vi xử lí thông thường. Hỗ trợ đường truyền 100 Base – T và 10 Base – T, hỗ trợ hoạt dộng kép tại 10Mb/s và 100Mb/s với auto – MDIX, hoàn toàn tương thích với cấu hình IEEE 802.3u, hỗ trợ IP/TCP/UDP checksum generation và checking.

- USB host/slave controller:

USB 2.0, hỗ trợ truyền dữ liệu tốc độ cao và tốc độ thấp, hỗ trợ USB chủ/khách, hai cổng USB. Cung cấp giao tiếp song song đến bộ vi xử lí, hỗ trợ Nios II bởi Terasic driver, hỗ trợ Programmed I/O (PIO) và Direct Memory Access (DMA).

- Cổng nối tiếp:

Gồm 2 cổng: một cổng giao tiếp RS-232, một cổng giao tiếp PS/2

- Cổng giao tiếp hồng ngoại:

Bộ truyền nhận tín hiệu 115.2kb/s, dòng điều khiển LED 32mA, tín hiệu ngõ vào được xác nhận bởi tích cực cạnh.

Hai đầu nối mở rộng:

2x40 pin của 2 đầu nối mở rộng được kết nối với 72 pin của Cyclone II I/O và 8 pin power và mass. Đầu nối 40 pin có thể tương thích với cable chuẩn 40 pin dùng cho ổ cứng IDE.

3.2.2. Giới thiệu phần mềm Quartus II

Quartus II là công cụ phần mềm phát triển của hãng Altera, cung cấp môi trường thiết kế toàn diện cho các thiết kế SOPC (hệ thống trên 1 chip khả trình - system on a programmable chip). Đây là phần mềm đóng gói tích hợp đầy đủ phục vụ cho thiết kế logic với các linh kiện logic khả trình PLD, FPGA của Altera, gồm các dòng APEX, Cyclone, FLEX, MAX, Stratix...Quatus II cung cấp các khả năng thiết kế sau:

– Môi trường thiết kế gồm các bản vẽ, sơ đồ khối, công cụ soạn thảo các ngôn ngữ: AHDL, VHDL, và Verilog HDL.

– Thiết kế LogicLock.

– Là công cụ mạnh để tổng hợp logic.

– Khả năng mô phỏng chức năng và thời gian.

– Phân tích thời gian.

– Phân tích logic nhúng với công cụ phân tích SignalTap® II.

– Cho phép xuất, tạo và kết nối các file nguồn để tạo ra các file chương trình.

– Tự động định vị lỗi.

– Khả năng lập trình và nhận diện linh kiện.

– Phần mềm Quartus II sử dụng bộ tích hợp NativeLink® với các công cụ thiết kế cung cấp việc truyền thông tin liền mạch giữa Quartus với các công cụ thiết kế phần cứng EDA khác.

– Quartus II cũng có thể đọc các file mạch (netlist) EDIF chuẩn, VHDL và Verilog HDL cũng như tạo ra các file netlist này.

– Quartus II có môi trường thiết kế đồ họa giúp nhà thiết kế dễ dàng viết mã, biên dịch, soát lỗi, mô phỏng...

Với Quartus có thế kết hợp nhiều kiểu file trong 1 dự án thiết kế phân cấp. Có thể dùng bộ công cụ tạo sơ đồ khối (Quartus Block Editor) để tạo ra sơ đồ khối mô tả thiết kế ở mức cao, sau đó dùng các sơ đồ khối khác, các bản vẽ như: AHDL Text Design Files (.tdf). EDIF Input Files (.edfh VHDL Design Files (.vhd). và Verilog HDL Design Files đế tạo ra thành phần thiết kế mức thấp.

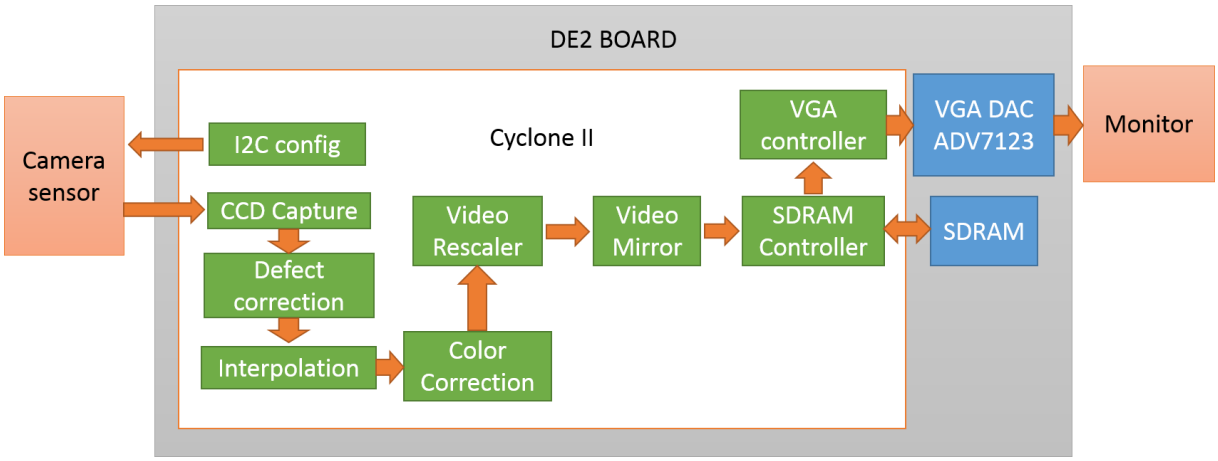
Quartus II cho phép làm việc với nhiều file ở cùng thời điểm, soạn thảo file thiết kế trong khi vấn có thế biên dịch hay chạy mô phỏng các dự án khác. Công cụ biên dịch Quartus II nằm ở trung tâm hệ thống, cung cấp quy trình thiết kế mạnh cho phép tùy biến đế đạt được thiết kế tối ưu trong dự án. Công cụ định vị lỗi tự động và các bàn tin cảnh báo khiến việc phát hiện và sửa lỗi trở nên đơn giản hơn.

3.3. Xây dựng thử nghiệm

3.3.1. Cách tiến hành

Hệ thống xử lý hình ảnh được xây dựng trên FPGA gồm 3 khối chính: khối thu thập dữ liệu, khối xử lý hình ảnh và khối hiển thị hình ảnh được biểu

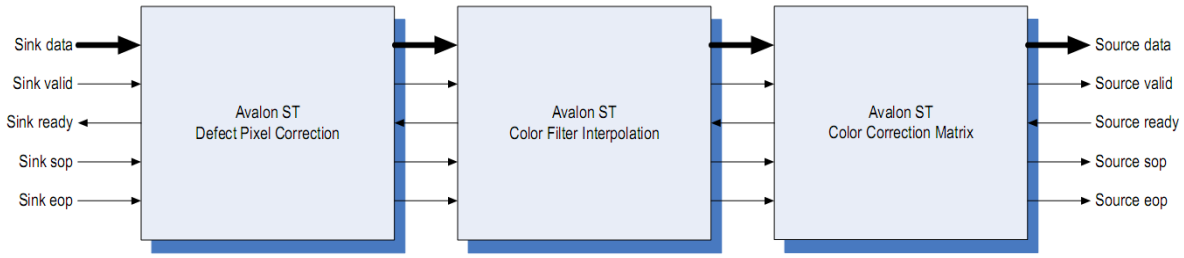
diễn trong hình 3-2. Hệ thống này có thể cho phép thay đổi độ phân giải hình ảnh đầu vào đầu ra phù hợp với độ phân giải của màn hình hiển thị hình ảnh, hỗ trợ các dữ liệu đầu vào với các chế độ 8 bit, 10 bit và 12 bit. Tương thích với chuẩn Avalon Streaming với ready_latency = 1. Tốc độ xung clock lên đến trên 100 MHz, tốc độ lấy mẫu dữ liệu lên đến trên 30 Msampe/s.



Hình 3-2. Mô hình hệ thống thử nghiệm

Khối thu thập dữ liệu từ camera và ghi vào SRAM, tuy nhiên do còn thiếu thiết bị nên trong phần mô phỏng trên kit DE2 em sử dụng ảnh tĩnh được chuyển thành định dạng RAW, sau đó được nạp vào SRAM thông qua Control panel DE2.

Khối xử lý ảnh được thiết kế thành 3 module: Defect pixel correction có chức năng khử nhiễu, interpolation có chức năng nội suy màu và color correction có chức năng sửa màu. Đầu tiên dữ liệu được lấy từ SRAM vào FIFO sau đó được khử nhiễu thông qua module defect correction và lại ghi vào FIFO. Tiếp theo dữ liệu được đọc từ FIFO và nội suy màu bằng module color Interpolation sau đó ghi vào FIFO. Cuối cùng dữ liệu được xử lý màu qua module Color correction rồi ghi vào FIFO. Dữ liệu được truyền theo chuẩn Avalon-ST, các module được kích hoạt bằng các switch trên Kit DE2. Thiết kế của khối xử lý hình ảnh được đưa ra như trong hình 3-3.



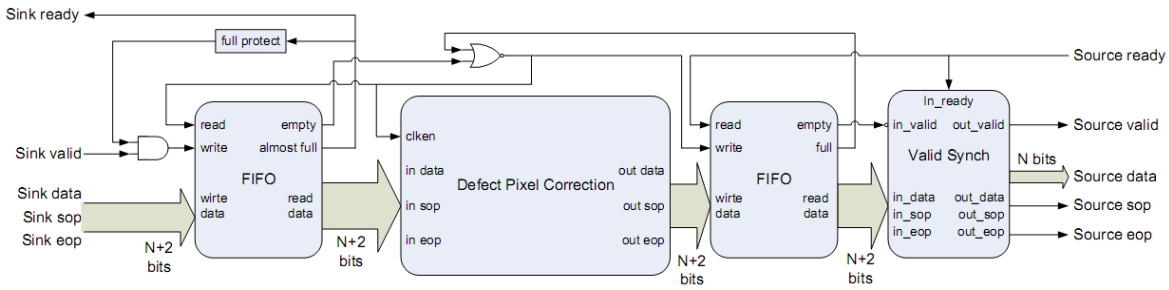
Hình 3-3. Sơ đồ thiết kế bộ xử lý ảnh

Tất cả các module đều được thiết kế theo chuẩn Avalon-ST do đó các module đều có các tín hiệu vào ra như nhau cụ thể:

Tên tín hiệu	Độ rộng (bit)	Hướng	Mô tả
clock	1	in	Avalon Clock and Reset
resetn	1	in	Avalon Clock and Reset
enable	1	In	Avalon Conduit
Sink_data_in	3*DW	in	Avalon ST – Sink - data
Sink_sop_in	1	in	Avalon ST – Sink – start of packet
Sink_eop_in	1	in	Avalon ST – Sink – end of packet
Sink_valid_in	1	in	Avalon ST – Sink – data valid
Sink_ready_out	1	out	Avalon ST – Sink – data ready
Source_data_out	3*DW	out	Avalon ST – Source – data
Source_sop_out	1	out	Avalon ST – Source – start of packet
Source_eop_out	1	out	Avalon ST – Source – end of packet
Source_valid_out	1	out	Avalon ST – Source – data valid
Source_ready_in	1	in	Avalon ST – Source – data ready

Bảng 3-1. Bảng mô tả các tín hiệu vào ra theo chuẩn Avalon-ST

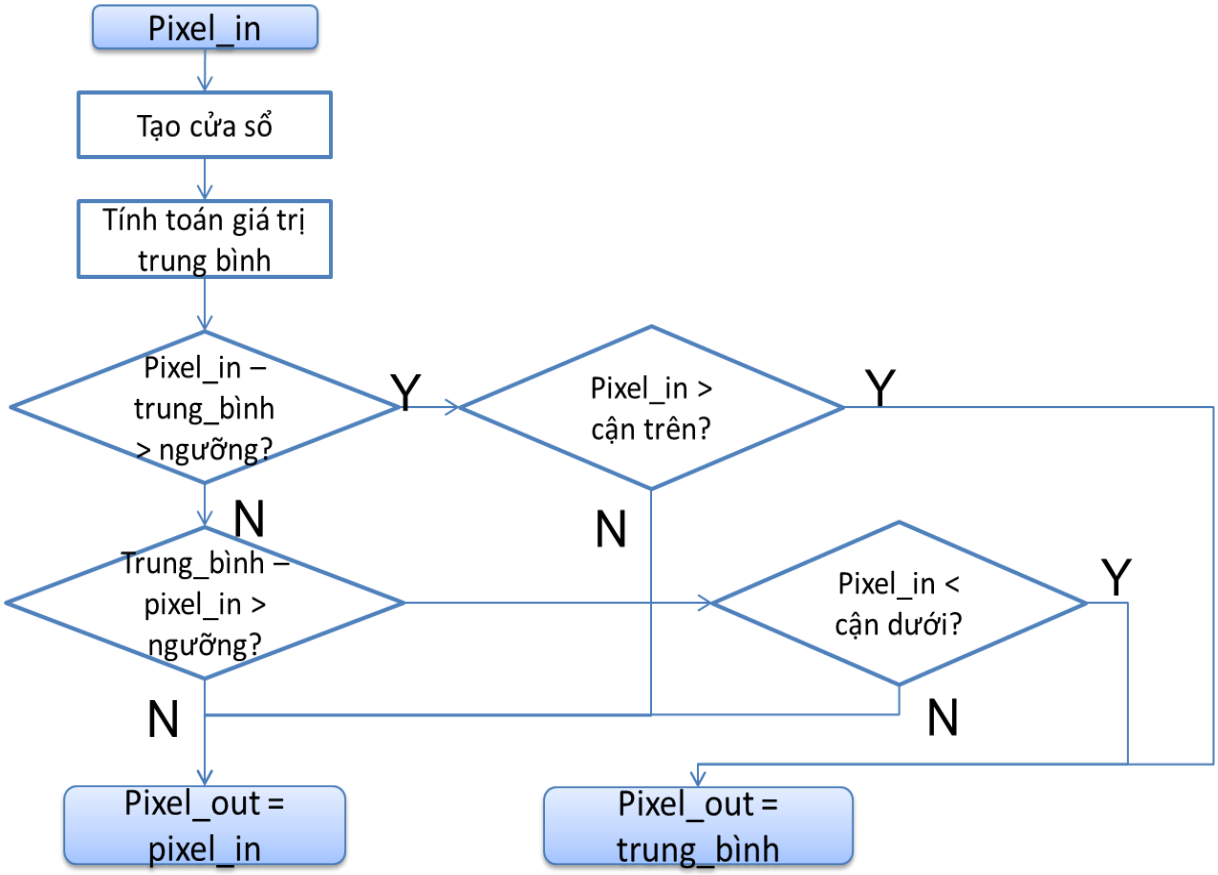
Dữ liệu được đọc từ SRAM vào FIFO và dữ liệu này được chuyển sang khối xử lý dữ liệu ảnh để loại bỏ các điểm ảnh chết bằng module Defect Pixel Correction. Hình 3-4 mô tả cụ thể dữ liệu được truyền từ FIFO sang module Defect Pixel Correction.



Hình 3-4. Sơ đồ truyền dữ liệu module Defect Pixel Correction

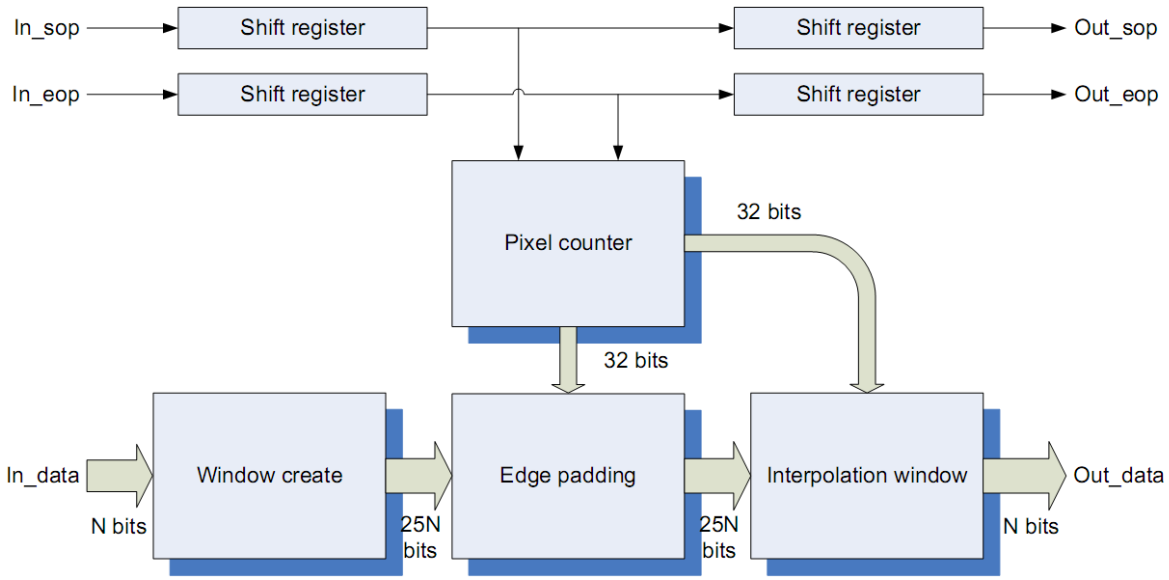
Tín hiệu Sink valid và full protect dùng để đáp ứng yêu cầu ghi dữ liệu vào FIFO, nếu được đáp ứng thì sẽ ghi N+2 bit vào FIFO. Tín hiệu clken điều khiển quá trình truyền dữ liệu từ FIFO sang module Defect Pixel correction. Nếu FIFO ở đầu vào của module rỗng hoặc FIFO bên đầu ra của module đầy sẽ dừng việc truyền dữ liệu. Tín hiệu Source ready in sẽ đáp ứng yêu cầu đọc dữ liệu của FIFO phía đầu ra của module. Sau đó dữ liệu sẽ được đồng bộ hóa bởi thành phần Valid Synch.

Dữ liệu được truyền vào module là các điểm ảnh, module Defect Pixel Correction có chức năng xác định các điểm ảnh lỗi và sửa lại theo thuật toán như trong hình 3-5 dưới đây:



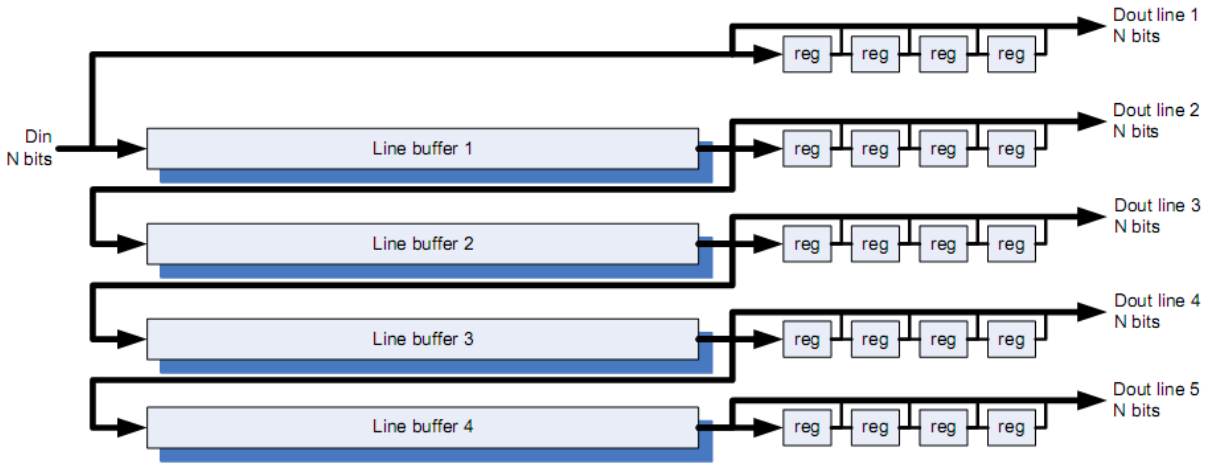
Hình 3-5: Sơ đồ khối giải thuật xác định điểm ảnh lỗi

Giải thuật đề nghị sử dụng cửa sổ trung bình và so sánh ngưỡng, kết hợp so sánh với cận trên và cận dưới. Nếu điểm ảnh ngõ vào có giá trị gần cận trên (hoặc gần cận dưới) và có độ sai lệch với giá trị trung bình của điểm ảnh xung quanh vượt ngưỡng thì xác định đó là điểm ảnh lỗi và được chỉnh sửa bằng giá trị trung bình của điểm ảnh xung quanh. Với ảnh 24 bit màu (mỗi kênh màu 8 bit) thì ngưỡng cận trên được chọn là 245 và ngưỡng cận dưới được chọn là 10. Ngưỡng so sánh với điểm ảnh lân cận có thể được chọn trong khoảng 40 - 80. Sơ đồ chi tiết được thể hiện trên hình 3-6.



Hình 3-6 Sơ đồ chi tiết module Defect Pixel Correction.

In_data được truyền vào theo 5 line và được tính toán lại theo thuật toán đã trình bày ở trên với cửa sổ được sử dụng có kích thước 5x5. Quá trình tạo cửa sổ 5x5 được trình bày ở hình 3-7.



Hình 3-7: Sơ đồ window creator

Các dữ liệu ảnh này sau khi được loại bỏ các điểm ảnh bị lỗi được truyền sang FIFO theo chuẩn avalon-ST để tiếp tục xử lý.

Sau khi kết thúc quá trình loại bỏ điểm ảnh lỗi dữ liệu được truyền sang module Color Interpolation theo chuẩn Avalon-ST. Tại module này dữ liệu ảnh sẽ được sửa các màu bằng bộ lọc trung bình.

Mặt nạ trung bình có độ phức tạp thấp và tốc độ xử lý nhanh hơn trong FPGA. Do đó đề tài chọn giải thuật mặt nạ lọc trung bình và so sánh ngưỡng để thực hiện thiết kế.

Việc áp dụng mặt nạ lọc sẽ làm giảm chất lượng ảnh, đặc biệt là tại các vùng cạnh (edge). Do đó cần biện pháp để hạn chế việc áp dụng mặt nạ lọc lên toàn bộ ảnh. Như đã phân tích ở trên, các điểm ảnh bị lỗi có 2 trường hợp là điểm nóng (là điểm bị kẹt ở mức cao) và điểm lạnh (điểm tín hiệu trả về luôn ở mức thấp) do đó ta có thể kết hợp so sánh với cận trên và dưới để áp dụng mặt nạ lọc cho hợp lý.

Một lưu ý nữa là các sắp xếp các điểm ảnh đặc trưng cho các màu là khác nhau đối với Bayer pattern. Do đó tùy từng màu ta sẽ có các cửa sổ trung bình khác nhau. Cụ thể ta có 2 cửa sổ cho mặt nạ lọc trung bình:

1		1		1
1				1
1		1		1

Mặt nạ trung bình cho điểm ảnh R&B

1		1		1
	2		2	
1				1
	2		2	
1		1		1

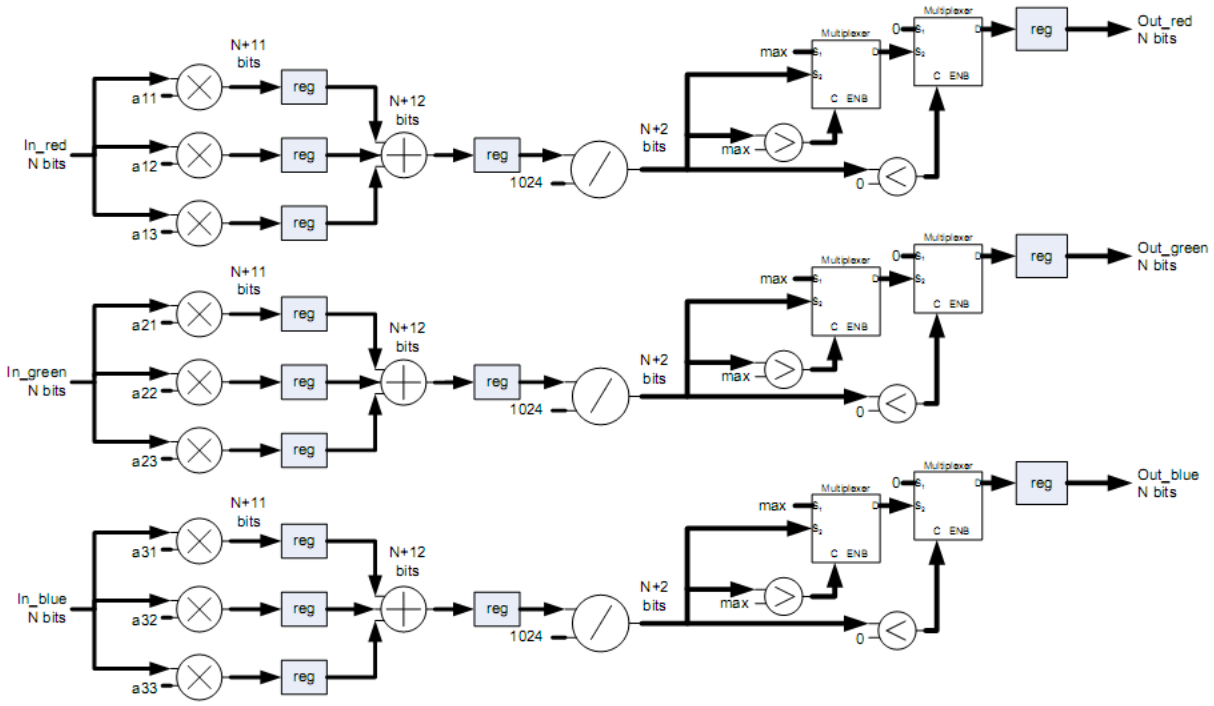
Mặt nạ trung bình cho điểm ảnh G

Tổng trọng số của cả 2 cửa sổ được chọn để có được tổng là lũy thừa của 2, để tiện cho việc tính toán trong FPGA.

Cuối cùng dữ liệu ảnh được truyền sang module Matrix Color Correction để xử lý. Đây là quá trình xử lý cuối cùng trước khi ảnh được chuyển sang bộ hiện thị ảnh. Trong quá trình này các điểm ảnh sẽ được sửa lại màu sắc cho phù hợp với cảm nhận màu của mắt người trước khi hiển thị ra màn hình. Các màu được tính theo công thức sau:

$$\begin{bmatrix} R\_out \\ G\_out \\ B\_out \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} x \begin{bmatrix} R\_in \\ G\_in \\ B\_in \end{bmatrix}$$

Trong module này các phép cộng, nhân sử dụng kỹ thuật pipeline, còn phép chia được thực hiện bằng phép dịch bit để tăng tốc độ xử lý của module. Hình 3-8 mô tả chi tiết cách thực hiện sử dụng ma trận sửa màu.



Hình 3-8. Sơ đồ chi tiết Color Correction Matrix

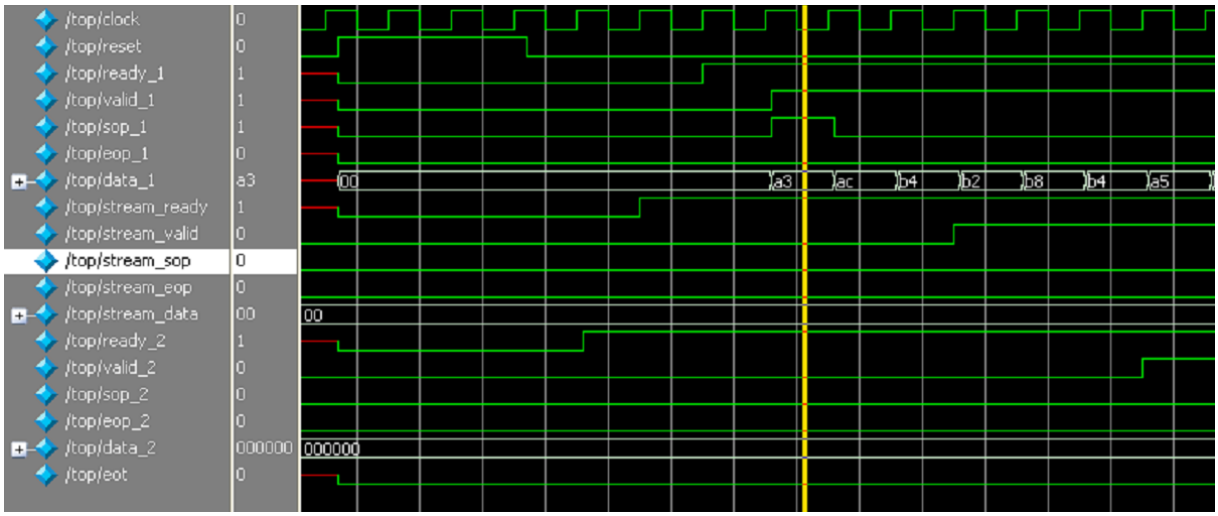
Sau nhiều thực nghiệm với các ma trận có trọng số khác nhau, ma trận cho ra màu sắc tốt nhất với các trọng số cụ thể như sau:

$$\begin{bmatrix} 1.6 & -0.3 & -0.3 \\ -0.3 & 1.6 & -0.3 \\ -0.3 & -0.3 & 1.6 \end{bmatrix}$$

Khối hiển thị: Chuyển dữ liệu ảnh từ hệ màu RGB sang hệ màu CMY và hiện thị lên màn hình.

Việc kiểm tra chức năng của hệ thống xử lý ảnh được thực hiện trên modelsim để đảm bảo các module chạy đúng theo chuẩn giao tiếp Avalon-ST.

Sau khi reset hệ thống và tín hiệu ready đạt được mức tích cực (nghĩa là sink sẵn sàng nhận dữ liệu), tín hiệu valid và startofpacket đạt tích cực, dữ liệu lúc này bắt đầu được truyền đi được thể hiện trong hình 3-6 dưới đây:



Hình 3-9. Kết quả mô phỏng theo từng tín hiệu

Khi ready mất tích cực, nghĩa là sink không sẵn sàng nhận dữ liệu), tín hiệu valid và data cũng mất sau 1 chu kỳ clock (vì ready latency = 1). Khi ready tích cực trở lại, tín hiệu valid lên mức tích cực và data tiếp tục truyền.

3.3.2. Thực hiện Demo trên Kit DE2.

Kết nối Kit DE2 với máy tính qua cổng USB và màn hình hiển thị với Kit DE2 qua cổng VGA.

Đầu tiên file hình ảnh gốc được biến đổi thành file ảnh chứa các điểm ảnh lỗi bằng thuật toán trên matlab và được lưu thành file dữ liệu.

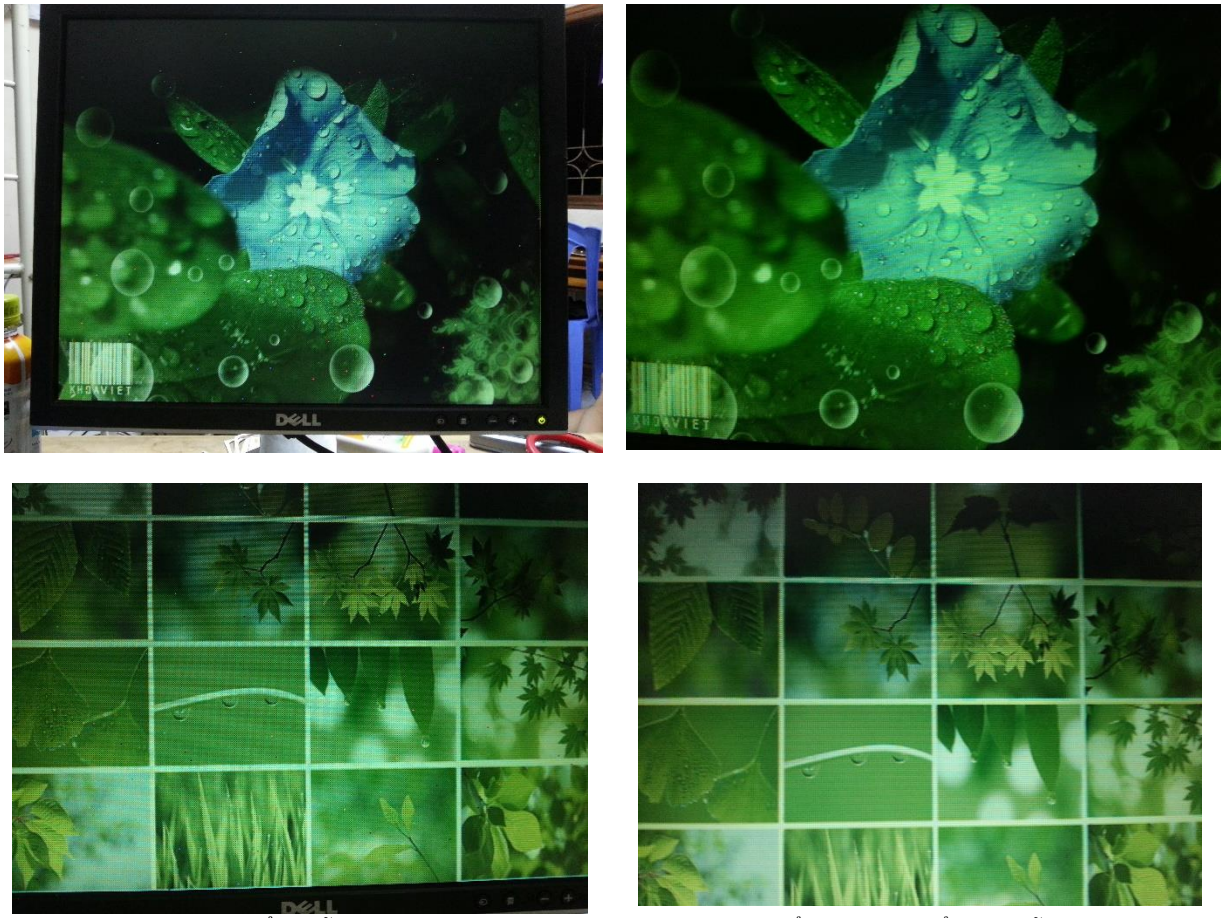
Sau đó sử dụng phần mềm điều khiển Control Panel DE2 để nạp file dữ liệu này lên SRAM của Kit DE2.

Chạy chức năng Programer trong Quartus II. Nạp file hệ thống demo được tạo DE2_TV.sof xuống Kit.

Chọn SW[0], SW[1], SW[2] để bật tắt chức năng sửa điểm ảnh lỗi, nội suy màu và sửa màu. Có thể sử dụng từng chức năng riêng hoặc sử dụng kết hợp các chức năng cùng lúc.

3.3.3. Đánh giá kết quả:

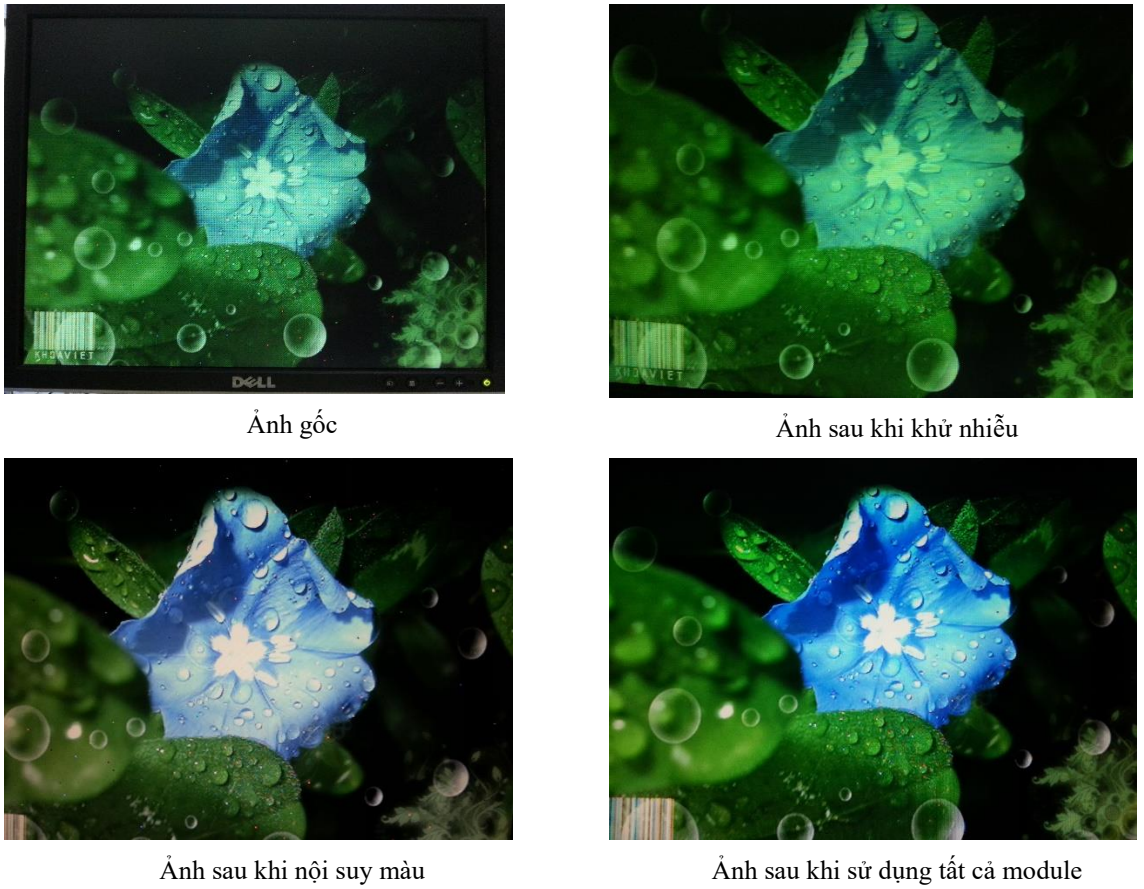
Kết quả thử nghiệm module Defect Pixel Correction trên Kit DE2 với một số ảnh đầu vào khác nhau:



Ảnh lỗi Ảnh đã xóa điểm ảnh lỗi

Các hình ảnh sau khi được xử lý bằng module Defect Pixel Correction hoàn toàn loại bỏ được các điểm ảnh lỗi. Các ảnh sau khi xử lý gần như không thay đổi so với ảnh gốc.

Hình 3-10 trình bày kết quả thử nghiệm trên Kit DE2



Ảnh gốc | Ảnh sau khi khử nhiễu

Ảnh sau khi nội suy màu | Ảnh sau khi sử dụng tất cả module

Hình 3-10. Kết quả xử lý ảnh

Sau khi thử nghiệm với một số ảnh đầu vào khác nhau, kết quả chương trình thực hiện đúng như mong đợi, các thuật toán lọc nhiễu, phục hồi ảnh đã làm việc đúng và kết quả hiển thị trên màn hình, bằng mắt thường có thể đánh giá các lỗi hoàn toàn đã được khắc phục.

KẾT LUẬN

Trong luận văn này, em đã nghiên cứu các kỹ thuật xử lý dữ liệu đa phương tiện, công nghệ lập trình FPGA và tìm hiểu kỹ thuật pipeline trong công nghệ FPGA, ngôn ngữ mô tả phần cứng verilog. Trong quá trình thực hiện luận văn em đã thu được những kết quả sau:

- Bước đầu đã nắm được kiến thức cơ bản về công nghệ PPGA, hiểu được tư tưởng luồng thiết kế trên công nghệ FPGA, khả năng xử lý dữ liệu của công nghệ FPGA.
- Hiểu và lập trình được bằng ngôn ngữ mô tả phần cứng Verilog, sử dụng được ngôn ngữ verilog để thiết kế lõi IP xử lý một trong nhiều loại dữ liệu đa phương tiện là xử lý hình ảnh.
- Nắm được cách sử dụng các công cụ lập trình như phần mềm Quartus II, phần mềm mô phỏng ModelSim

Những hạn chế và hướng phát triển của đề tài:

- Do thời gian thực hiện đề tài có hạn nên em mới chỉ thực hiện được xử lý ảnh trên các công cụ mô phỏng chưa thiết kế được vi mạch, chưa hoàn toàn hiểu hết các chức năng của các công cụ.
- Trong thời gian tới em sẽ tiếp tục hoàn thiện đề tài của mình để có được một sản phẩm hoàn chỉnh được thiết kế trên công nghệ FPGA.

PHỤ LỤC

1. Code file top điều khiển hoạt động của 3 module sửa các điểm ảnh lỗi, nội suy màu và sửa màu:

```
module image_processing(
        // System
        clock,
        resetn,
        enable,
        // Sink side
        sink_data_in,
        sink_valid_in,
        sink_ready_out,
        sink_sop_in,
        sink_eop_in,
        // Source side
        source_data_out,
        source_valid_out,
        source_ready_in,
        source_sop_out,
        source_eop_out
);
/*****************************************************************************
 *                           Parameter Declarations                          *
*****************************************************************************/
```

```
parameter               DW = 8;
parameter               WIDTH = 640;
parameter               HEIGHT = 480;
parameter               THRESHOLD = 100;
/*****************************************************************************
*************
 *                      Port Declarations                       *
*****************************************************************************
*************/
// System
input                   clock;
input                   resetn;
input                   enable;
// Sink side
input  [DW-1:0]    sink_data_in;
input                   sink_valid_in;
output                      sink_ready_out;
input                   sink_sop_in;
input                   sink_eop_in;
// Source side
output [3*DW-1:0]       source_data_out;
output                      source_valid_out;
input                   source_ready_in;
output                      source_sop_out;
output                      source_eop_out;
```

```
/*****************************************************************************
*************

 *                    Constant Declarations                    *

*****************************************************************************
*************/

/*****************************************************************************
*************

 *              Internal wires and registers Declarations              *

*****************************************************************************
*************/

// Internal Wires

wire   [DW-1:0]    avalon_1_data;

wire                        avalon_1_ready;

wire                        avalon_1_valid;

wire                        avalon_1_sop;

wire                        avalon_1_eop;

wire   [3*DW-1:0] avalon_2_data;

wire                        avalon_2_ready;

wire                        avalon_2_valid;

wire                        avalon_2_sop;

wire                        avalon_2_eop;

// Internal Registers

// State Machine Registers

// Integers


// Avalon Defect Pixel Correction
```

```
avalon_st_defect_pixel_correction      avalon_st_defect_pixel_correction_0(
        // System
        .clock(clock),
        .resetn(resetn),
        .enable(enable),
        // Sink side
        .sink_data_in(sink_data_in),
        .sink_valid_in(sink_valid_in),
        .sink_ready_out(sink_ready_out),
        .sink_sop_in(sink_sop_in),
        .sink_eop_in(sink_eop_in),
        // Source side
        .source_data_out(avalon_1_data),
        .source_valid_out(avalon_1_valid),
        .source_ready_in(avalon_1_ready),
        .source_sop_out(avalon_1_sop),
        .source_eop_out(avalon_1_eop)
);
defparam
        avalon_st_defect_pixel_correction_0.DW = DW,
        avalon_st_defect_pixel_correction_0.WIDTH = WIDTH,
        avalon_st_defect_pixel_correction_0.HEIGHT = HEIGHT,
        avalon_st_defect_pixel_correction_0.THRESHOLD = THRESHOLD;
// Avalon Color Filter Interpolation
avalon_st_color_filter_interpolation    avalon_st_color_filter_interpolation_0(
```

```
        // System
        .clock(clock),
        .resetn(resetn),
        .enable(enable),
        // Sink side
        .sink_data_in(avalon_1_data),
        .sink_valid_in(avalon_1_valid),
        .sink_ready_out(avalon_1_ready),
        .sink_sop_in(avalon_1_sop),
        .sink_eop_in(avalon_1_eop),
        // Source side
        .source_data_out(avalon_2_data),
        .source_valid_out(avalon_2_valid),
        .source_ready_in(avalon_2_ready),
        .source_sop_out(avalon_2_sop),
        .source_eop_out(avalon_2_eop)
);
defparam
        avalon_st_color_filter_interpolation_0.DW = DW,
        avalon_st_color_filter_interpolation_0.WIDTH = WIDTH,
        avalon_st_color_filter_interpolation_0.HEIGHT = HEIGHT;
// Avalon Color Correction Matrix
avalon_st_color_correction_matrix     avalon_st_color_correction_matrix_0(
        // System
        .clock(clock),
```

```
        .resetn(resetn),
        .enable(enable),
        // Sink side
        .sink_data_in(avalon_2_data),
        .sink_valid_in(avalon_2_valid),
        .sink_ready_out(avalon_2_ready),
        .sink_sop_in(avalon_2_sop),
        .sink_eop_in(avalon_2_eop),
        // Source side
        .source_data_out(source_data_out),
        .source_valid_out(source_valid_out),
        .source_ready_in(source_ready_in),
        .source_sop_out(source_sop_out),
        .source_eop_out(source_eop_out)
);
defparam
        avalon_st_color_correction_matrix_0.DW = DW;
endmodule
```

TÀI LIỆU THAM KHẢO

[1]. PGS.Vũ Quý Điềm (chủ biên), Phạm Văn Tuân, Nguyễn Thúy Anh, Đỗ Lê Phú, Nguyễn Ngọc Văn – *Cơ sở kỹ thuật đo lường điện tử* - NXB Khoa Học và Kỹ Thuật 2006.

[2]. Trần Ngọc Phụng, Trần Thị Điểm, Cao Trần Bảo Thương, Huỳnh Hữu Thuận, - *thực hiện một số thuật toán dò tìm chuyển động trên FPGA* - Hội Nghị Khoa Học Trường Đại Học KHTN lần 5, (2006).

[3] Mani B. Srivastava. VHDL tutorial. UCLA – EE.

[4] System Generator for DSP (Getting started Guide, Reference Guide, User Guide). Xilinx.

[5] Nguyễn Trọng Hải - *Bài giảng Verilog* - ĐH Kỹ thuật công nghệ TPHCM.

[6] Nguyễn Viết Kính, Trịnh Anh Vũ - *Thông tin số* - NXBGD.

[7] Jim Lewis, Coding a 40x40 Pipelined Multiplier in VHDL (version online:http://www.synthworks.com/papers/VHDL_RTL_Pipelined_Multiplier_MAPLD_2002_S_BW.pdf)

[8] ISO/IEC 15444-1,Information Technology-JPEG2000 Image Coding System, Part 1: Core Coding System, 2000

[9] PENG Zhou, ZHAO Bao-jun, High-throughout hardware architecture of MQ arithmetic coder, International Conference on Signal Processing (ICSP), October, 2010, pp. 430-433

[10] Michael Dyer, David Taubman, Saeid Nooshabadi, Improved throughput arithmetic coder for JPEG2000, International Conference on Image Processing, vol. 4, October, 2004, pp. 2817-2820

[11] Manjunath Gangadhar, Dinesh Bhatia, FPGA based EBCOT architecture for JPEG 2000, International Conference on Field-Programmable Technology (FPT), December, 2003, pp. 228-233

[12] Taoufik Saidani, Mohamed Atri, Rached Tourki, Implementation of JPEG 2000 MQ-Coder, International Conference on Design and Technology of Integrated Systems in Nanoscale Era, March, 2008, pp. 1-4

[13] Kai Liu, Yu Zhoub,Yun Song Li,Jian Feng Ma, A high performance MQ encoder architecture in JPEG2000, the VLSI Journal, vol. 43, no. 3, June, 2010, pp. 305-317

[14] Minsoo Rhu, In-Cheol Park, Optimization of Arithmetic Coding for JPEG2000, IEEE Transactions on Circuits and Systems for Video Technology, vol. 20, no. 3, March, 2010, pp. 446-451 118 Tạp chí Khoa học và Kỹ thuật - Học viện KTQS số 153 (4-2013)

[15] Tinku Acharya, Ping-Sing Tai, JPEG2000 Standard for Image Compression: Concepts, Algorithms and VLSI Architectures, New Jersey, U.S.A: John Wiley & Sons, chapter 5, 2005, pp. 185-195

[16] Wael M. El-Sharkasy, Mohamed E. Ragab, Hardware modelling of JPEG2000 MQ-encoder, International Conference on Intelligent and Advanced Systems (ICIAS), vol. 2, June, 2012, pp. 707-712

[17] Altera, Avalon Interface Specifications, California, U.S.A, 2011, pp. 35 - 44

[18] Altera, Stratix III Device Handbook, California, U.S.A, 2011

[19] Lê Quang Minh, Tài liệu bài giảng môn "Công nghệ lập trình nhúng", K20-CNPM, Trường Đại học Công nghệ - ĐHQGHN.

[20] Giáo trình: Công nghệ phần mềm nhúng, Nguyễn Ngọc Bình, NXB ĐHQGHN, Hà Nội, 2013.