MỤC LỤC

MỤC LỤC **1**

DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT **3**

DANH MỤC CÁC BẢNG **4**

DANH MỤC CÁC HÌNH VẼ, ĐỒ THỊ **5**

TÓM TẮT LUẬN VĂN **6**

MỞ ĐẦU **7**

Lý do lựa chọn đề tài **7**

Mục tiêu đề tài **9**

Phương pháp nghiên cứu **9**

Kết cấu luận văn **9**

CHƯƠNG 1 : LÝ THUYẾT TỔNG QUAN **11**

1.1 Giới thiệu chung **11**

1.2 Cấu trúc CGRA **11**

1.3 Vấn đề cần giải quyết **14**

CHƯƠNG 2 : THIẾT KẾ CHI TIẾT CỦA MUSRA **16**

2.1 Đặc tả kỹ thuật **16**

2.2 Cấu trúc mảng phần cứng có thể tái cấu hình **21**

2.2.1 Cấu trúc tổng thể của MUSRA 21

2.2.2 Mảng RCA 22

CHƯƠNG 3 : KẾT QUẢ MÔ PHỎNG VÀ THỬ NGHIỆM **39**

3.1 Mô hình mô phỏng của MUSRA **39**

3.2 Kịch bản kiểm chứng **40**

3.2.1 Phép tổng sai phân (chênh lệch) tuyệt đối (SAD) 40

3.2.2 Tổng chuyển động (Moving Sum) 40

3.2.3 Nhân vô hướng hai vector 41

3.2.4 Tích chập 42

3.3 Kết quả thực nghiệm và đánh giá **43**

3.3.1 Kết quả tổng hợp phần cứng 43

3.3.2 Kết quả mô phỏng 44

KẾT LUẬN 47

TÀI LIỆU THAM KHẢO 48

DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT

TT	Thuật ngữ viết tắt	Thuật ngữ viết đầy đủ	Ý nghĩa
1.	ASIC	Application-Specific Integrated Circuit	Mạch tích hợp chuyên dụng
2.	CGRA	Coarse Grain Reconfigurable Architectures	Cấu trúc tái cấu hình lõi thô
3.	CPU	Central Processing Unit	Đơn vị xử lý trung tâm
4.	DMA	Direct Memory Access	Truy cập bộ nhớ trực tiếp
5.	DFG	Data Flow Graph	Sơ đồ luồng dữ liệu
6.	FIFO	First In, First Out	
7.	FPGA	Field-Programmable Gate Array	Mảng cổng lập trình được dưới dạng trường
8.	MIMD	Multiple Instruction, Multiple Data	Xử lý đa lệnh đa dữ liệu
9.	MURSA	Multimedia Specific Reconfigurable Architecture	Mảng các phần tử xử lý có thể tái cấu hình kiến trúc thô ứng dụng cho xử lý đa phương tiện
10.	PE	Processing Element	Phần tử xử lý
11.	SIMD	Single Instruction, Multiple Data	Xử lý đơn lệnh, đa dữ liệu
12.	SoC	System on Chip	Hệ thống trên chip
	RCA	Reconfigurable Cell Array	Mảng phần tử tái cấu hình
13.	RTL	Register Transfer Level	Mức chuyển giao thanh ghi
14.	VLIW	Very Long Instruction Word	Từ lệnh rất dài

DANH MỤC CÁC BẢNG

BẢNG 2- 1: CÁC PHÉP TÍNH ĐƯỢC HỖ TRỢ BỞI RCA. 18
BẢNG 2-2: TÍN HIỆU VÀO RA CỦA KHỐI RCA8×8. 23
BẢNG 2-3 TÍN HIỆU VÀO RA CỦA KHỐI RC_LINE. 25
BẢNG 2-4: MÔ TẢ CÁC TÍN HIỆU CỦA RC. 27
BẢNG 2-5: ĐỊNH NGHĨ THÔNG TIN CẤU HÌNH NGUỒN DỮ LIỆU LỐI VÀO MỖI RC. 29
BẢNG 2-6: MÔ TẢ CÁC TÍN HIỆU CỦA PE. 30
BẢNG 2-7: MÔ TẢ CÁC TÍN HIỆU VÀO RA CỦA DATAPATH. 31
BẢNG 2-8: MÔ TẢ CÁC PHÉP TOÁN ĐƯỢC THỰC HIỆN TRÊN KHỐI ALU ...32
BẢNG 2-9: MÔ TẢ TÍN HIỆU CỦA THANH GHI CỤC BỘ LOR. 37
BẢNG 2-10: MÔ TẢ CÁC TÍN HIỆU CỦA ROUTER_A. 37
BẢNG 2-11: MÔ TẢ CÁC TÍN HIỆU CỦA ROUTER_B. 38
BẢNG 3- 1 KẾT QUẢ TỔNG HỢP MẢNG RCA8×8 TRÊN CÔNG NGHỆ FPGA VIRTEX-7 ((XC7VX485T). 44
BẢNG 3- 2 THỜI GIAN THỰC THI CÁC VÒNG LẶP KERNEL TRÊN CÁC NỀN TẢNG TÍNH TOÁN KHÁC NHAU. 45

DANH MỤC CÁC HÌNH VẼ, ĐỒ THỊ

HÌNH 1- 1 CẤU TRÚC CHUNG CỦA MỘT MẢNG PHẦN CỨNG TÁI CẤU HÌNH CẤU TRÚC THÔ. 12
HÌNH 2- 1: BIỂU DIỄN DFG CHO MỘT VÒNG LẶP ĐƠN GIẢN. 17
HÌNH 2- 2: LẬP LỊCH SỰ CẤU HÌNH VÀ THỰC THI CỦA MỘT VÒNG LẶP TRÊN MUSRA 17
HÌNH 2- 3: CẤU TRÚC CỦA MUSRA. 21
HÌNH 2- 4: TỔ CHỨC CỦA FIFO. 22
HÌNH 2- 5: CẤU TRÚC TOP-DOWN CỦA MẢNG RCA. 23
HÌNH 2- 6: CẤU TRÚC CỦA MỘT PHẦN TỬ RC. 26
HÌNH 2- 7: ĐỊNH DẠNG THÔNG TIN CẤU HÌNH CÁC PHẦN A, B, C. 28
HÌNH 2- 8: CẤU TRÚC CỦA MỘT PE. 30
HÌNH 2- 9: GIAO DIỆN VÀO/RA CỦA DATAPATH. 31
HÌNH 2- 10. SƠ ĐỒ THIẾT KẾ CÁC KHỐI THỰC HIỆN CÁC PHÉP TÍNH TRÊN DATAPATH. 32
HÌNH 2- 11: KHỐI ALU. 33
HÌNH 2- 12: ĐƠN VỊ CHỨC NĂNG ADD/SUB THỰC HIỆN PHÉP TÍNH CỘNG VÀ TRỪ HAI SỐ 16-BIT. 33
HÌNH 2- 13: CẤU TRÚC CỦA BỘ CỘNG LAI GHÉP HBD_ADDER. 34
HÌNH 2- 14: BỘ CỘNG CLA 4-BIT. 34
HÌNH 2- 15: SƠ ĐỒ CẤU TRÚC PHẦN CỨNG CỦA BỘ NHÂN BAUGH-WOOLEY [15] 35
HÌNH 2- 16. BỘ NHÂN MUL 4 BÍT [15] 36
HÌNH 2- 17: SƠ ĐỒ THỰC HIỆN KHỐI ABS 36
HÌNH 3- 1. MÔ HÌNH MÔ PHỎNG RCA CỦA MUSRA TRONG MÔI TRƯỜNG MODELSIM. 39
HÌNH 3- 2. (A) DFG VÀ (B) TỔ CHỨC DỮ LIỆU CHO QUÁ TRÌNH TÍNH TOÁN TRÊN MUSRA 40
HÌNH 3- 3. ÁNH XẠ TỔNG CHUYỂN ĐỘNG TRÊN MỘT CỬA SỔ TRƯỢT VỚI ĐỘ DÀI N=10. 41
HÌNH 3- 4. DFG (A), ÁNH XẠ CỦA DFG TRÊN MUSRA (B), VÀ SỰ THỰC THI ĐƯỢC ĐƯỜNG ỐNG HÓA (C) CỦA PHÉP NHÂN MA TRẬN – VECTƠ 42
HÌNH 3- 5: DFG THỰC HIỆN MỘT BỘ LỌC FIR BẬC 4. 43
HÌNH 3- 6. KẾT QUẢ MÔ PHỎNG CỦA KHỐI ƯỚC LƯỢNG CHUYỂN ĐỘNG (A) VÀ BỘ LỌC FIR (B) SỬ DỤNG MÔ HÌNH RTL CỦA MUSRA. 45

TÓM TẮT LUẬN VĂN

Luận văn mô tả thiết kế một cấu trúc tái cấu hình cấu trúc thô ứng dụng cho xử lý đa phương tiện gọi tắt là MUSRA (Multimedia Specific Reconfigurable Architecture). Cấu trúc này được sử dụng để tăng tốc độ tính toán cho các nhiệm vụ tính toán chuyên sâu trong một thuật toán bằng việc khai thác nhiều mức cơ chế song song trong một thuật toán. Cấu trúc hỗ trợ khả năng tái cấu hình động bằng việc cho phép kết cấu phần cứng tái cấu hình lại để thực hiện các chức năng khác nhau ngay cả khi hệ thống đang làm việc. Cấu trúc đề xuất được mô hình hoá ở mức truyền thanh ghi RTL (Register Transfer Level) sử dụng ngôn ngữ VHDL. Một vài ví dụ benchmark cũng đã được ánh xạ lên cấu trúc MUSRA để đánh giá độ linh hoạt và hiệu năng cao của hệ thống. Thiết kế đã được mô hình hóa bằng ngôn ngữ VHDL (trong đó RCA của MUSRA được thiết kế dưới dạng RTL) và tiến hành mô phỏng, so sánh với các phương thức thực hiện khác. Các kết quả thực nghiệm chỉ ra rằng thiết kế đáp ứng được yêu cầu cơ bản đặt ra ban đầu: như tăng tốc độ tính toán cho các vòng lặp; khả năng tái hình linh hoạt các vòng lặp khác nhau có thể sử dụng cho một số phép toán thường dùng trong xử lý đa phương tiện trong truyền thông. Các module được tham số hóa, dễ dàng mở rộng thiết kế theo các phương án kết nối khác nhau, trong đó lõi RCA của MUSRA được thiết kế với khả năng có thể mở rộng kích thước theo cả 2 chiều.

MỞ ĐẦU

Lý do lựa chọn đề tài

Xu hướng phát triển khoa học công nghệ những năm qua chỉ ra rằng các thiết bị di động cầm tay ngày càng trở nên thông minh hơn, mật độ tích hợp các ứng dụng chức năng ngày càng cao hơn. Các thiết bị này nói chung đều yêu cầu khả năng xử lý các chức năng tính toán chuyên sâu như truyền thông, chụp ảnh, quay phim, xem truyền hình, dịch vụ định vị toàn cầu,… theo thời gian thực. Thực hiện phần cứng cho các thiết bị như vậy luôn là một thách thức đối với nhà thiết kế bởi các yêu cầu khắt khe như giảm kích thước và công suất tiêu thụ của chip, tăng hiệu năng xử lý, rút ngắn thời gian thiết kế và triển khai sản phẩm, đơn giản hóa quá trình nâng cấp thiết bị sau bán hàng,… Thêm vào đó khả năng hỗ trợ đa chuẩn (truyền thông hoặc mã hóa) của thiết bị cũng là yêu cầu ngày càng phổ biến bởi nó cho phép giảm giá thành tích hợp sản phẩm cũng như cho phép khách hàng có thể nhận được nhiều loại hình dịch vụ từ các nhà cung cấp dịch vụ khác nhau trên cùng một thiết bị. Nói chung, trong các hệ thống nhúng truyền thống có hai phương pháp chủ yếu được sử dụng cho việc thực thi một chức năng mong muốn. Một phương pháp là sử dụng các vi mạch tích hợp chuyên dụng ASIC (Application Specific Integrated Circuit). Phương pháp thứ hai là sử dụng các bộ vi xử lý (Processor) có thể lập trình bằng phần mềm. Tuy nhiên, cả hai phương pháp trên đều không thể thỏa mãn được tất cả các yêu cầu như chỉ ra ở trên trong việc thực thi các ứng dụng đa phương tiện thế hệ tiếp theo. Một giải pháp rất hứa hẹn cho việc giải quyết vấn đề nêu trên là các hệ thống tính toán có thể tái cấu hình (Reconfigurable Computing System)[1]. Điểm khác biệt quan trọng của một hệ thống như vậy với các hệ thống xử lý thông thường là nó sử dụng các kết cấu phần cứng có thể tái cấu hình (Reconfigurable Hardware) cho việc tăng tốc độ thực thi các phần tiêu tốn nhiều thời gian tính toán trong một thuật toán. Phần cứng có thể tái cấu hình thường được tổ chức thành một mảng các đơn vị xử lý có thể tái cấu hình RPU (Reconfigurable Processing Units)[2]. Các chức năng tính toán chuyên sâu của một thuật toán có thể được hoán chuyển vào hoặc ra khỏi mảng RPU hoặc ở thời gian chạy (tức sự cấu hình động) hoặc ở thời gian biên dịch (tức sự cấu hình tĩnh). Ưu điểm lớn nhất của khả năng có thể tái cấu hình động hệ thống là nó cho phép tăng mật độ chức năng hiệu dụng của các ứng dụng được ánh xạ lên một đơn vị tài nguyên phần cứng[3]. Nói cách khác, kỹ thuật

này cho phép hệ thống xử lý có thể thực hiện cùng một số lượng ứng dụng với lượng tài nguyên phần cứng ít hơn khi dùng các mạch ASIC riêng biệt. Việc tăng mật độ chức năng của phần cứng đạt được bằng việc lập lịch các nhiệm vụ tính toán để chia sẻ theo thời gian cùng một tài nguyên phần cứng giống như việc quản lý bộ nhớ ảo trong máy tính. Điều đặc biệt là sau khi mảng RPU được cấu hình cho một chức năng nào đó nó sẽ hoạt động giống như một đơn vị phần cứng chuyên dụng cho chức năng đó. Vì thế, hệ thống xử lý sử dụng các kết cấu phần cứng có thể tái cấu hình thường đạt được sự dung hòa giữa hiệu năng tính toán và tính mềm dẻo. Điều này là bởi vì phần cứng có thể tái cấu hình kết hợp được khả năng lập trình lại sau chế tạo (post-fabrication programmability) của bộ vi xử lý với phong cách tính toán song song hiệu năng cao của một vi mạch ASIC.

FPGA (Field-Programmable Gate Array) là thiết bị hỗ trợ các kết cấu phần cứng có thể tái cấu hình ở mức lõi tinh (Fine-grained fabric). FPGA có thể được cấu hình để thực hiện hầu như bất kỳ chức năng phần cứng số nào. Tuy nhiên nhược điểm của FPGA là kích thước, công suất tiêu thụ và trễ lan truyền cao[4]. Điều này hạn chế khả năng ứng dụng của FPGA cho các thiết bị cầm tay. Nhằm vượt qua các giới hạn của các bộ vi xử lý và các thiết bị có thể tái cấu hình lõi tinh (cụ thể là FPGA), các cấu trúc phần cứng có thể tái cấu hình động lõi thô CGRA (Coarse-grained Reconfigurable Architecteture) đã được nghiên cứu và phát triển. Các cấu trúc CGRA thường được đề xuất cho một miền ứng dụng cụ thể ([5], [6] và [7]), chẳng hạn các ứng dụng xử lý đa phương tiện và truyền thông, thay vì hướng tới tới một ứng dụng bất kỳ như FPGA. CGRA được sử dụng để tăng tốc độ tính toán cho các nhiệm vụ tính toán chuyên sâu trong một thuật toán bằng việc khai thác nhiều mức cơ chế song song như DLP (Data Level Parallelism), ILP (Instruction Level Parallelism), TLP (Task Level Parallelism) trong một thuật toán [8]. Cấu trúc của CGRA cũng cần hỗ trợ khả năng tái cấu hình động bằng việc cho phép kết cấu phần cứng tái cấu hình lại để thực hiện các chức năng khác nhau ngay cả khi hệ thống đang làm việc. Bằng việc cấu hình động lại phần cứng như vậy, nhiều chức năng khác nhau được ánh xạ tới cùng một kết cấu phần cứng, do đó dẫn đến giảm được kích thước, giá thành cũng như công suất tiêu thụ của cả hệ thống.

Xuất phát từ thực tế nêu trên, luận văn tập trung nghiên cứu với đề tài **"Mô hình hóa mức RTL và thực thi mảng phần cứng có thể tái cấu hình cấu trúc thô cho các ứng dụng xử lý đa phương tiện".**

Mục tiêu đề tài

Mô hình hoá ở mức truyền thanh ghi RTL (Register Transfer Level) và thực thi một cấu trúc phần cứng có thể tái cấu hình động lõi thô ứng dụng trong lĩnh vực xử lý đa phương tiện, gọi tắt là MUSRA (Multimedia Specific Reconfigurable architecture). MUSRA được sử dụng để tăng tốc độ tính toán cho các nhiệm vụ tính toán chuyên sâu trong một thuật toán bằng việc khai thác nhiều mức cơ chế song song trong một thuật toán. Các phần tử xử lý có thể được cấu hình độc lập để thực hiện một chức năng tính toán trên dữ liệu có dấu hoặc không dấu 16 bit. Cấu trúc của MUSRA cũng cần hỗ trợ khả năng tái cấu hình động bằng việc cho phép kết cấu phần cứng tái cấu hình lại để thực hiện các chức năng khác nhau ngay cả khi hệ thống đang làm việc.

Phương pháp nghiên cứu

Để thực hiện mục tiêu trên, phương pháp nghiên cứu được sử dụng gồm:

- *Phương pháp nghiên cứu lý thuyết*: Nghiên cứu tìm hiểu các kỹ thuật mô hình hóa một chức năng phần cứng ở mức RTL bằng ngôn ngữ mô tả phần cứng VHDL từ đó nghiên cứu, hiểu rõ cấu trúc mảng MUSRA đã được đề xuất bởi nhóm nghiên cứu ở PTN SIS, Trường ĐHCN-ĐHQGHN.
- *Phương pháp thiết kế*: Phát triển và mô hình hoá cấu trúc phần cứng tái cấu hình ở mức RTL. Tổng hợp phần cứng với công nghệ FPGA; tiến hành đánh giá hiệu năng, giá thành phần cứng của cấu trúc MUSRA.
- *Phương pháp mô phỏng và kiểm chứng*: Mô phỏng thiết kế MUSRA với một số ứng dụng benchmark trên phần mềm ModelSIM nhằm đánh giá khả năng tái cấu hình linh hoạt và hiệu năng cao của cấu trúc MUSRA.
- *Phương pháp kiểm thực*: Kiểm nghiệm thiết kế trên nền tảng vi mạch FPGA.

Kết cấu luận văn

Nội dung luận văn được tổ chức thành các phần sau:

- **Chương 1: Lý thuyết tổng quan:** giới thiệu chung về CGRA, đưa ra phương án tiếp cận thiết kế hệ thống CGRA ứng dụng xử lý đa phương tiện.
- **Chương 2: Thiết kế chi tiết của MUSRA:** trình bày thiết kết chi tiết của cấu trúc phần cứng có thể tái cáu hình MUSRA (bao gồm các khối chức năng, giao diện ghép nối, mô tả chức năng hoạt động, …).
- **Chương 3: Kết quả mô phỏng và thử nghiệm**: trình bày phương án mô phỏng và đánh giá MUSRA trên phần mềm ModelSIM. Các kết quả mô

phỏng và một số đánh giá cũng được trình bày trong chương này.

- **Kết luận:** Trình bày ưu, nhược điểm của thiết kế, đưa ra các kết quả thu được từ việc thực hiện đề tài và phương hướng phát triển tiếp theo.

CHƯƠNG 1 : LÝ THUYẾT TỔNG QUAN

1.1 Giới thiệu chung

Khi xem xét một đối tượng xử lý dữ liệu, nếu nhìn ở mức hệ thống sẽ thấy phần tử xử lý dữ liệu là các bộ xử lý, khối DSP hoặc là các máy tính đơn lẻ trong một hệ thống lớn và phức tạp. Tuy nhiên, nếu trong quá trình xem xét đối tượng xử lý dữ liệu, ta quan tâm tới các khối tính toán ở mức thấp hơn, mức logic chẳng hạn, thì sẽ thấy các phần tử xử lý dữ liệu có thể là các tế bào logic (logic cell) cấu trúc tinh trong FPGA (Field Programmable Gate Array) hay các tế bào cấu trúc thô (như ALU, bộ nhân, ..) trong CGRA (Coarse-grained Reconfigurable Architecture). Một cách tổng quan, các khối tính toán ở mức thấp hơn có thể được phân chia thành 2 kiểu cấu trúc cơ bản:

- Cấu trúc tập trung xử lý dữ liệu ở mức bit: được gọi là cấu trúc tinh (fine – grained) (ví dụ các chip FPGA của Xilinx[9] hoặc Altera[10])
- Cấu trúc tập trung xử lý dữ liệu theo nhóm bit với các khối chức năng phức tạp: được gọi là cấu trúc thô (coarse – grained) (ví dụ [5], [6], [7]).

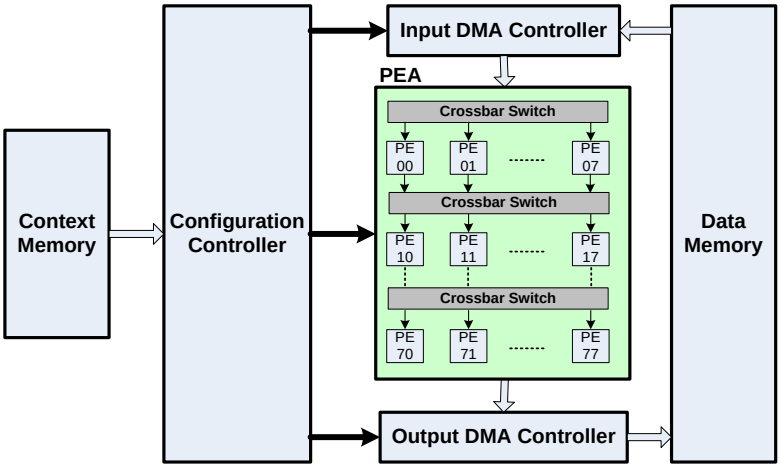
Vi mạch FPGA là một kiểu ứng dụng phổ biến, trực quan cho người thiết kế khi muốn sử dụng cấu trúc tinh để xây dựng lên các thiết kế của mình. Ưu điểm nổi bật của kiểu cấu trúc này là tính mềm dẻo và linh hoạt trong thiết kế. Tuy nhiên, vì can thiệp tới mức bit, nên hệ thống kết nối trên vi mạch dạng này chiếm một tài nguyên đáng kể [4]. Điều này là một khó khăn cho những người thiết kế khi phải thiết kế và làm việc trên các ứng dụng yêu cầu tối ưu về mặt tài nguyên như: các thiết bị di động. Nhược điểm thứ hai của cấu trúc dạng này là hiệu suất sử dụng năng lượng bị giảm nếu so với các vi mạch ASIC [4]. Cấu trúc thô ra đời là một giải pháp cho vấn đề này. Cấu trúc dạng này vừa đảm bảo được sự linh hoạt trong thiết kế, vừa giải quyết được bài toán tối ưu về tài nguyên và năng lượng. Phần tiếp theo sẽ trình bày rõ hơn về mặt cấu trúc tổng thể của một cấu trúc CGRA đã được nghiên cứu trên thế giới.

1.2 Cấu trúc CGRA

Tùy từng hướng ứng dụng cụ thể mà có những định hướng nghiên cứu khác nhau về CGRA. Có thể kể đến một số cấu trúc tiêu biểu như: Cấu trúc REMUS[5] và ADRES[6]. Cấu trúc ADRES (Architecture for Dynamically Reconfigurable

Embedded System) kết hợp bộ xử lý VLIW và ma trận tái cấu hình lõi thô vào thành một cấu trúc đơn trong đó ma trận tái cấu hình lõi thô đóng vai trò như một đơn vị đồng xử lý với VLIW. Ngược lại cấu trúc REMUS–II (Reconfigurable Muiltimedia System version II) thiết kế CGRA như một lõi IP được gắn vào bus hệ thống của bộ xử lý. REMUS-II chứa từ một hoặc hai đơn vị xử lý cấu trúc thô có khả năng tái cấu hình động một mảng các bộ vi xử lý RISC ghép với một bộ xử lý ARM thông qua bus AHB. Việc thiết kế CGRA như vậy giúp REMUS dễ dàng sử dụng lại trong các hệ thống khác nhau.

Về tổng thể, cấu trúc chung của CGRA thường bao gồm một mảng các phần tử xử lý PEA (Processing Element Array), các bộ điều khiển truy nhập trực tiếp bộ nhớ DMA vào/ra, bộ nhớ cấu hình, bộ nhớ dữ liệu và bộ điều khiển cấu hình như đã chỉ ra trong Hình 1- 1. Chức năng của các khối có thể tóm lược như sau:



Hình 1- 1 Cấu trúc chung của một mảng phần cứng tái cấu hình cấu trúc thô.

- *Bộ điều khiển cấu hình* (*Configuartion Controller*) thực hiện chức năng nạp các thông tin cấu hình (context) từ bộ nhớ cấu hình, sau đó giải mã chúng thành các thông tin để thiết lập chức năng của các khối tái cấu hình. Thời gian cần để cấu hình các khối tái cấu hình gọi là thời gian cấu hình. Tối ưu hóa thời gian này chính là đối tượng chính khi thực hiện thiết kế các bộ phân tích ngữ nghĩa của thông tin cấu hình. Một vài kỹ thuật như kỹ thuật nén context, kỹ thuật song song hóa hoạt động của bộ phân tích ngữ nghĩa context khi PEA thực thi đều là những kỹ thuật có thể giúp làm giảm thời gian cấu hình khỏi hiện tượng quá tải.

- *Mảng các phần tử xử lý có thể tái cấu hình* (*PEA)* thường được tổ chức thành một mảng có quy tắc các khối có khả năng tái cấu hình như là các tế bào logic có khả năng tái cấu hình (chẳng hạn bảng LUT (Look-up Table) trong FPGA) hoặc các phần tử xử lý có khả năng tái cấu hình (ví dụ như PACT XPP-III[7] hoặc REMUS[5]). Các khối cấu hình được kết nối với nhau thông qua mạng định tuyến có thể tái cấu hình. Mạng định tuyến này được thiết kế dựa trên kỹ thuật chuyển mạch-mạch (circuit-switching technique) hoặc kỹ thuật chuyển mạch gói (packet-switching technique). Các tham số quan trọng của một PEA có thể kể đến đó là cấu trúc kết nối hình học (topology), cấu trúc mức lõi của PEA (ví dụ 4-bit, 8-bit hay 16-bit), các PE có cấu trúc đồng nhất hoặc không đồng nhất, độ sâu cấu hình, mô hình thực thi,…
- *Bộ nhớ* là thành phần chính của bất kì hệ thống xử lý nào. Cách tổ chức và dung lượng của bộ nhớ ảnh hưởng trực tiếp đến hiệu năng, sự tiêu thụ công suất và diện tích chip bán dẫn dùng để thực thi của hệ thống cần thiết kế. Đặc biệt đối với các tác vụ tính toán chuyên sâu cần thực hiện song song một số lượng lớn các tính toán thì thông lượng truy xuất bộ nhớ thường là nguyên nhân gây nên tình trạng thắt nút cổ chai trong hoạt động của hệ thống. Bộ nhớ trung tâm dựa trên các bus truyền thống về cơ bản không thỏa mãn được các yêu cầu về băng thông truy xuất dữ liệu của các hệ thống tính toán tái cấu hình. Một bộ nhớ được phân tán dựa trên vi mạng trên chip (Network-on-Chip: NoC) cho phép nhiều phần tử xử lý thực hiện truy xuất đọc/ghi bộ nhớ đồng thời là một giải pháp hiệu quả cho vấn đề này. Băng thông thích nghi, cấu trúc hệ thống bộ nhớ, và cơ chế truy xuất bộ nhớ là các vấn đề chính khi thực hiện thiết kế bộ nhớ phân tán trên chip.

Ở đây cần làm rõ sự khác biệt giữa cấu trúc và nguyên tắc hoạt động của một CGRA và một bộ xử lý đa lõi. Trong các bộ xử lý đa lõi, mỗi một lõi là một bộ xử lý hoàn chỉnh bao gồm đơn vị điều khiển (control unit) và khối xử lý dữ liệu (datapath). Ngược lại, mỗi một phần tử xử lý của CGRA chỉ chứa một khối xử lý dữ liệu và toàn bộ PEA được trang bị một đơn vị điều khiển dùng chung. Điều này giúp làm giảm sự quá tải khi triển khai đơn vị điều khiển. Các thông tin cấu hình đóng vai trò như các lệnh của bộ xử lý quy định hoạt động của CGRA. Thông tin cấu hình chỉ ra các hoạt động cụ thể của PEA (ví dụ hoạt động của từng tế bào tái cấu hình (Reconfigurable cell: RC), các kết nối bên trong giữa các RC, nguồn dữ

liệu vào/ra cho PEA,..) cũng như là các tham số điều khiển hoạt động của PEA. Tương tự như một chu trình lệnh của bộ xử lý, một chu trình context cũng bao gồm ít nhất 3 pha đó là pha nạp thông tin cấu hình (context fetching), giải mã thông tin cấu hình (context decoding) và thực thi. Tuy nhiên, sự khác nhau ở đây chính là CGRA chỉ cần được cấu hình một lần cho việc thực thi nhiều chu kì. Một khi đã được cấu hình, CGRA hoạt động như một phần cứng dành riêng cho sự tính toán được định nghĩa trước. CGRA chỉ được cấu hình lại khi có một nhu cầu tính toán khác xuất hiện bằng cách nạp thông tin cấu hình mới. Ngược lại, bộ xử lý luôn luôn phải thực hiện tất cả các pha của một chu kì lệnh cho tất cả các lệnh thậm chí nếu mã lệnh của lệnh đó không thay đổi. Kết quả là, hiệu năng của CGRA cao hơn bộ xử lý bởi vì thời gian cho việc thực thi hai pha nạp và giải mã được giảm đi.

Nhận xét chung:

- Có ba thành phần chính cấu thành nên một hệ thống có CGRA: Bộ điều khiển cấu hình, bộ nhớ và mảng phần tử xử lý có thể tái cấu hình PEA. Ở mức hệ thống, các CGRA có thể được thực hiện như một đơn vị đồng xử lý trong đơn vị xử lý trung tâm (thường là các bộ vi xử lý) hoặc cũng có thể được sử dụng như một lõi IP ghép nối với đơn vị xử lý trung tâm qua hệ thống bus dùng chung.
- Bộ nhớ được quản lý thành 2 phần riêng biệt là bộ nhớ cấu hình và bộ nhớ dữ liệu.
- Hệ thống sử dụng dây nối, bus hoặc các bộ ghép kênh có thể tái cấu hình để gắn kết các phần tử xử lý với nhau. Việc xây dựng các cấu trúc định tuyến này tùy thuộc vào từng loại ứng dụng cụ thể và nhà thiết kế.

1.3 Vấn đề cần giải quyết

Khi thiết kế mảng CGRA ứng dụng cho xử lý đa phương tiện và truyền thông cần giải quyết các nội dung sau:

- Cấu trúc của CGRA cần được thiết kế hướng tới việc tăng tốc độ tính toán cho các vòng lặp. Bằng việc ánh xạ tính toán trong thân của một vòng lõi lên CGRA, CGRA chỉ cần thực hiện cấu hình một lần cho việc thực hiện nhiều lần cả vòng lặp, vì vậy nó có thể giảm một cách hiệu quả thời gian thực thi của vòng lặp
- Thông tin cấu hình cho CGRA được tổ chức dưới dạng các gói tin gọi là

context. Mỗi context bao gồm các từ cấu hình dùng để xác định hoạt động của mảng các phần tử xử lý có thể tái cấu hình (Reconfigurable Cell Array: RCA) (chẳng hạn chức năng của mỗi RC (Reconfigrable Cell), kết nối giữa các RC, nguồn dữ liệu lối vào, đích của kết quả, v.v.) cũng như các tham số điều khiển cho bộ điều khiển của CGRA. Quá trình tái cấu hình và hoạt động của các đơn vị phần cứng cần được tổ chức và lập lịch theo phương thức đường ống hóa (pipe line) nhằm giảm thời gian dùng cho tái cấu hình.

CHƯƠNG 2 : THIẾT KẾ CHI TIẾT CỦA MUSRA

2.1 Đặc tả kỹ thuật

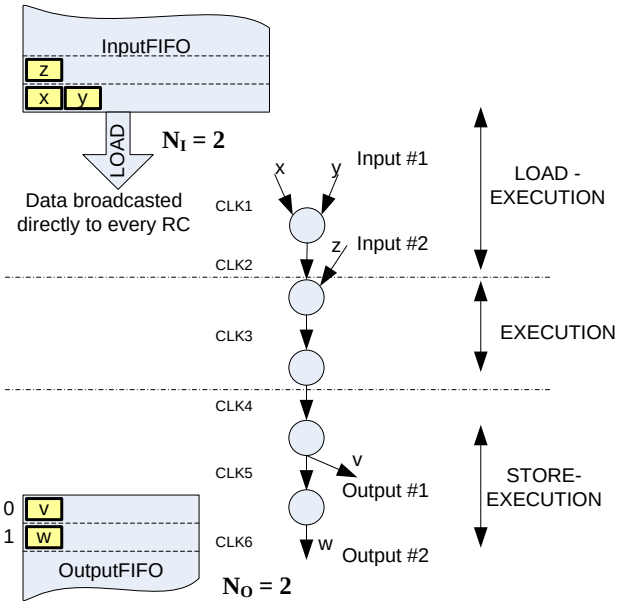
Trong phần này sẽ mô tả các đặc tả kỹ thuật cho một mảng các phần tử xử lý có thể tái cấu hình cấu trúc thô ứng dụng cho xử lý đa phương tiện gọi tắt là MUSRA (Multimedia Specific Reconfigurable Architecture) [11] được nghiên cứu và đề xuất bởi nhóm nghiên cứu ở phòng thí nghiệm Hệ thống tích hợp thông minh thuộc Trường Đại học công nghệ - Đại học Quốc gia Hà Nội.

Mô hình thực thi của MUSRA là mô hình đa lệnh – đa dữ liệu được đường ống hóa (pipelined Multi-Instruction - Multi-Data model), trong đó mỗi phần tử tái cấu hình RC (Reconfigurable Cell) có thể được cấu hình để thực hiện một tác vụ độc lập. Mỗi tầng của đường ống sẽ tương đương với một hàng RC. Các vòng lặp lõi trong ứng dụng được thực hiện thông qua việc ánh xạ thân vòng lặp lên MUSRA. Như vậy, chỉ cần cấu hình MUSRA một lần cho việc tính toán lặp lại của một vòng lặp. Nhiều lần lặp của một vòng lặp sẽ được lập lịch để thực thi đồng thời trong đường ống. Điều này làm giảm một cách đáng kể thời gian thực thi từ đó tăng tốc độ tính toán các vòng lặp lõi trong các thuật toán.

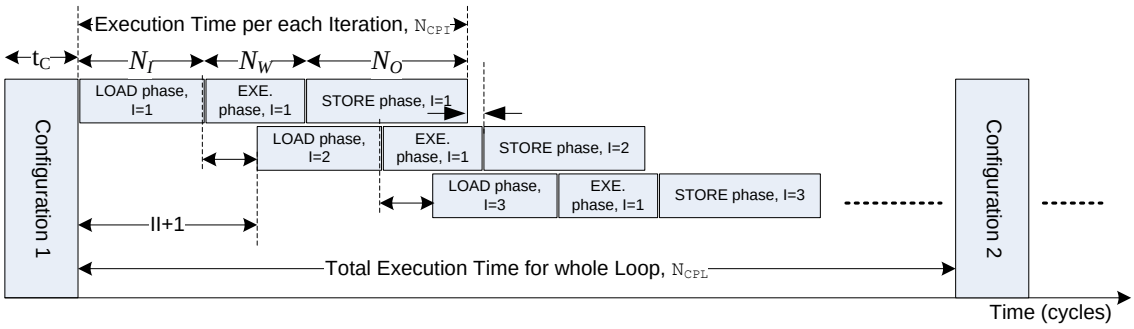
Để ánh xạ một vòng lặp lõi lên MUSRA, thân của vòng lặp sẽ được biểu diễn bằng lưu đồ luồng dữ liệu DFG (Data-Flow Graph) như được chỉ ra trong Hình 2 - 1. Các DFG này sau đó sẽ được ánh xạ lên MUSRA bằng việc tạo ra các thông tin cấu hình về việc gán một nốt của DFG tới RC nào và sườn nào của DFG tới các kết nối trên mảng MUSRA. Cuối cùng, DFG được lập lịch để thực thi tự động trên MUSRA bằng việc tạo ra các tham số điều khiển tương ứng cho bộ điều khiển MUSRA. Ngay khi đã được cấu hình cho một vòng lặp nhất định, MUSRA sẽ hoạt động giống như một phần cứng chuyên dụng cho vòng lặp đó. Khi tất cả vòng lặp đã được hoàn thành, vòng lặp sẽ được loại bỏ khỏi MUSRA và một vòng lặp mới sẽ được ánh xạ lên MUSRA.

Như chỉ ra trong Hình 2 - 1 một lần lặp của MUSRA được bắt đầu bằng pha LOAD-EXECUTION và sau đó là pha EXECUTION, cuối cùng được kết thúc bằng pha STORE-EXECUTION. Pha LOAD-EXECUTION và STORE-EXECUTION bao hàm rằng sự thực thi xảy ra song song với việc nạp và lưu dữ liệu một cách tương ứng. Trong khi đó, pha EXECUTION có nghĩa rằng trong suốt

pha này không có bất cứ thao tác đọc hoặc xuất dữ liệu nào diễn ra. Sự thực thi của một vòng lặp trên MUSRA được lập lịch sao cho các pha khác nhau của các lần lặp liên tiếp được xếp chồng lên nhau ở mức nhiều nhất có thể (Hình 2-2). Lập lịch cũng cần đảm bảo không có bất cứ xung đột nào xảy ra giữa các tài nguyên khi nhiều pha diễn ra đồng thời.



Hình 2 - 1: Biểu diễn DFG cho một vòng lặp đơn giản.



Hình 2 - 2: Lập lịch sử cấu hình và thực thi của một vòng lặp trên MUSRA.

Trong mô hình này, MUSRA có thể bắt đầu tính toán ngay khi dữ liệu của lần nhập dữ liệu đầu tiên xuất hiện ở lối vào của MUSRA, vì vậy pha LOAD và pha EXECUTION của cùng một lần lặp có thể tiến hành song song. Nói cách khác, mô hình thực thi cho phép ba pha LOAD, EXECUTION, STORE được thực hiện gối lên nhau theo phương thức đường ống ở mức cao nhất. Ở khía cạnh khác, nó cũng

cho phép dữ liệu của lần lặp tiếp theo được nạp đồng thời với dữ liệu của lần lặp hiện tại, vì vậy mô hình này có thể không chỉ tối đa hóa mức độ xếp chồng giữa các lần lặp liên tiếp mà còn cải thiện cả khả năng để khai thác dữ liệu có thể dùng lại giữa các lần lặp [11].

Các đặc tính chủ yếu của MUSRA được đề xuất:

+ *Mô hình thực thi của một ứng dụng trên MUSRA:*
  - Đa lệnh đa dữ liệu được đường ống hóa (Pipelined MIMD (Multi-instruction Multi-Data)

+ *Mô hình cấu hình:*
  - Thông tin cấu hình cho MUSRA được tổ chức dưới dạng các gói tin gọi là context. Mỗi context bao gồm các từ cấu hình 32-bit dùng để xác định hoạt động của mảng RCA (chẳng hạn chức năng của mỗi RC, kết nối giữa các RC, nguồn dữ liệu lối vào, đích của kết quả, v.v.) cũng như các tham số điều khiển cho bộ điều khiển của MUSRA.

Mô hình cấu trúc của MUSRA

- Các phần tử xử lý có thể tái cấu hình RC (Reconfigurable Cell) được tổ chức thành mảng 2 chiều 8×8;
- Các phép tính được hỗ trợ bởi mỗi RC: Hỗ trợ các phép tính trên dữ liệu dấu phẩy tĩnh có dấu hoặc không dấu (signed/unsigned fixed-point) như chỉ ra trong Bảng 2- 1;
- Mỗi RC có 3 toán hạng nguồn N-bit (N có thể là 8 hoặc 16). Sở dĩ cần sử dụng 3 toán hạng nguồn vì trong một số thuật toán xử lý đa phương tiện ta cần phải thực hiện các phép toán cộng-cộng hoặc nhân-cộng (nhân ma trận);
- Có những đơn vị chức năng đặc biệt thực hiện nhiệm vụ load/store dữ liệu tới/ra khỏi các RC;
- Có những đơn vị chức năng đặc biệt thực hiện nhiệm vị read/write để chuyển dữ liệu từ/tới bộ nhớ dữ liệu của MUSRA.

Bảng 2- 1: Các phép tính được hỗ trợ bởi RCA.

Phân loại	Mã lệnh	Phép tính	Ý nghĩa	Từ gợi nhớ

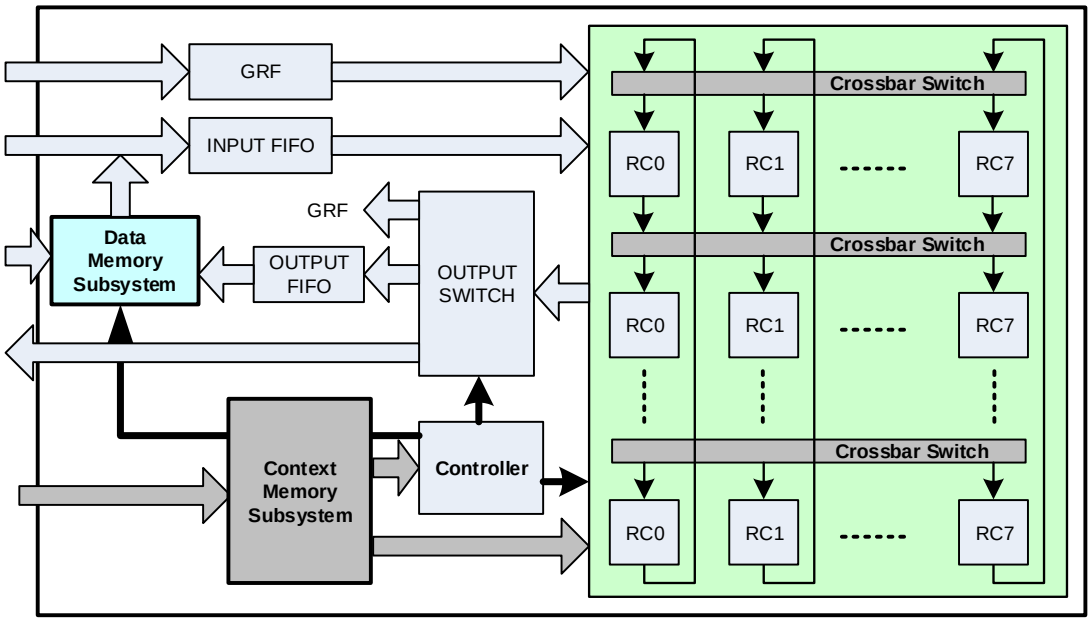
Arithmetic operations	5’b00000 (0)	A+B	Phép cộng A và B	ADD
	5’b00001 (1)	A−B	Phép trừ đi B đơn vị từ A	SUB
Shift operations	5’b00010 (2)	A>>B[3:0]	Phép dịch phải A đi B[3:0] đơn vị	BSR
	5’b00011 (3)	A<<B[3:0]	Phép dịch trái A đi B[3:0] đơn vị	BSL
	5’b00100 (4)	(A+((1<<B)>>1))>>B	Dịch phải và Vòng	SRR
Bypass A	5’b00101 (5)	A	Bypass A	A
Bitwise operations	5’b00110 (6)	A&B	Toán tử AND	AND
	5’b00111 (7)	A|B	Toán tử OR	OR
	5’b01000 (8)	A^B	Toán tử XOR	XOR
	5’b01001 (9)	~(A^B)	Toán tử NOT XOR	NXOR
Absolute operation	5’b01010 (10)	|A−B|	Chênh lệch tuyệt đối (Absolute Difference)	ASD
Comparison Operations	5’b01011 (11)	A−B>0	A có lớn hơn B không?	TGT
	5’b01100 (12)	A−B=0	A có bằng B không?	TEQ
	5’b01101 (13)	A−B>=0	A có lớn hơn hoặc bằng B không?	TGE
Saturation operator 1 (Clip A between 0 and B)	5’b01110 (14)	A<0?0:(A>B?B:A)	Nếu A<0 thì kết quả bằng 0. Ngược lại, nếu A>B thì kết quả bằng B. Ngược lại kết quả bằng A.	CLIP
	5’b01111 (15)	Max(A, B)	Nếu A<B thì kết quả bằng B. Ngược lại kết quả bằng A.	MAXB

Condition operation 2 (Multiplexer)	5’b10000 (16)	C?A:B	Nếu C=1 thì kết quả bằng A. Ngược lại kết quả bằng B.(‘C’ is the LOR_Input that is result of RC that is in the immediately above row and the same column with the current RC)	MUX
Multiplication	5’b10001 (17)	A×B	Nhân 2 số 8 bit chứa trong các byte thấp hơn của A và B	MUL
	5’b10010 (18)	**Reserved**		
Asymmetric Arithmetic operators	5’b10011 (19)	B−A	Trừ đi A đơn vị từ B.	
	5’b10100 (20)	B−A>0	A có nhỏ hơn B không?	
	5’b10101 (21)	B−A>=0	A có nhỏ hơn hoặc bằng B không?	
Condition operation 2	5’b10110 (22)	C?B+A:B−A	Nếu C=1 thì kết quả bằng A+B. Ngược lại, kết quả bằng B-A.‘C’ là LOR_Input từ cùng dòng và cột với RC hiện tại.	
Saturation operator 2	5’b10111 (23)	Min(A, B)	Nếu A>B thì kết quả bằng B, ngược lại kết quả bằng A.	MIN
	5’b11000 (24)	**Reserved**		
Bypass B	5’b11001 (25)	B	Bypass B	B
3-operand operators	5’b11010 (26)	Result + B	Result:= Result + B	ACC

	5’b11011 (27)	C+|A−B|	Result := C+ |A−B|	SADC
	5’b11100 (28)	C+(A+B)	Result := C+A+B	SUM3
	5’b11101 (29)	B + |C−A|	Result := B+|C−A|	SADB
	5’b11110 (30)	A*B + C	Result := A*B+C	MAC
	Others		**Reserved**	

2.2 Cấu trúc mảng phần cứng có thể tái cấu hình

2.2.1 Cấu trúc tổng thể của MUSRA

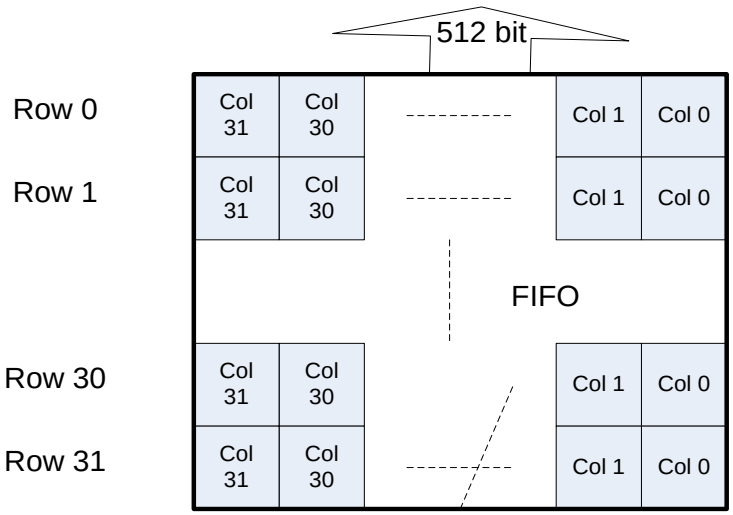


Hình 2 - 3: Cấu trúc của MUSRA.

Hình 2 - 3 mô tả cấu trúc của phần cứng có thể tái cấu hình lõi thô CGRA (Coarse-grained Reconfigurable architecture) được đề xuất cho các ứng dụng xử lý đa phương tiện và truyền thông gọi tắt là MUSRA. MUSRA bao gồm các khối chính như mảng tính toán có thể tái cấu hình RCA (Reconfigurable Computing Array), các bộ đệm FIFO (Input_FIFO/ Output_FIFO) cho việc nhập/xuất dữ liệu, tệp các thanh ghi toàn cục (GRFs: Global Register Files), bộ nhớ và hệ thống mạch

điều khiển cho dữ liệu (Data memory) và thông tin cấu hình (Context memory), và đơn vị điều khiển (Controller).

Dữ liệu vào/ra của mảng RCA được xếp hàng đợi bởi các bộ đệm FIFO độ sâu 32 hàng với băng thông là 256 bit (có thể tổ chức thành 32 byte hoặc 16 từ 16 bit) (như chỉ ra trong Hình 2 - 4). Thông qua hệ thống chuyển mạch (crossbar switch) trong RCA, dữ liệu từ INPUT_FIFO có thể quảng bá tới mọi RC, trong khi thông qua OUTPUT_SWITCH dữ liệu đã được xử lý bởi RCA được xuất tới OUTPUT_FIFO hoặc ghi vào GRF.



Hình 2 - 4: Tổ chức của FIFO.

Bộ nhớ dữ liệu là bộ nhớ lưu trữ dữ liệu đầu vào và các kết quả tính toán được thực hiện trên mảng RCA. Bộ nhớ cấu hình là bộ nhớ lưu trữ các thông tin cấu hình của các thành phần trong RCA, ví dụ như: opcode, router A, router B, router-config,... Bộ điều khiển là một máy trạng thái được tham số hóa cho phép tạo ra các tín hiệu điều khiển hoạt động của RCA một cách chính xác theo các thông tin cấu hình được nạp tới nó.

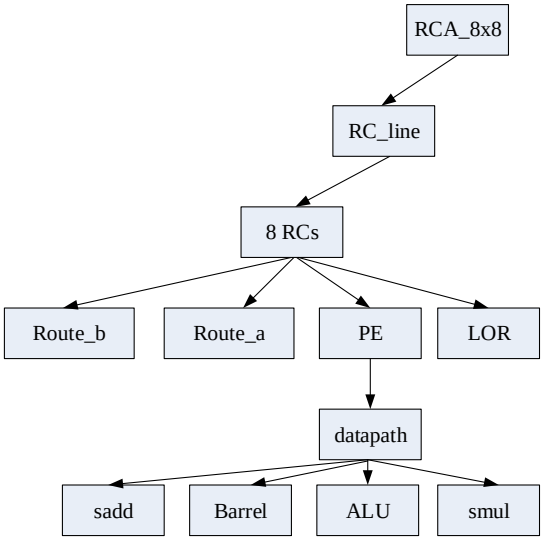
Mục tiêu của luận văn này là mô hình hóa và thực thi mảng RCA, do đó cấu trúc của RCA sẽ được trình bày chi tiết trong phần tiếp theo.

2.2.2 **Mảng RCA**

Thành phần quyết định tới khả năng có thể tái cấu hình của CGRA là mảng RCA được tổ chức thành một ma trận 8×8 phần tử RC (Reconfigurable Cell). Mỗi phần

tử RC của mảng có thể được cấu hình một cách độc lập tới một chức năng riêng biệt ở thời gian chạy. Nhiều RC có thể được kết hợp với nhau theo một mô hình DFG nào đó để thực hiện một nhiệm vụ tính toán chuyên sâu (Computation-intensive algorithms). Phân tích cấu trúc TOP-DOWN của mảng RCA8×8 được chỉ ra trong Hình 2 - 5.

Dựa trên các phân tích về tính cục bộ dữ liệu và cơ chế song song có sẵn trong các vòng lặp[12], kết cấu truyền thông của RCA được thiết kế nhằm khai thác tối đa khả năng tính toán theo phương thức đường ống hóa (pipelining) tính toán của các vòng lặp, cũng như khai thác tối đa dữ liệu giữa các lần lặp qua đó giảm băng thông dữ liệu vào RCA. Các RC trong hai hàng liên tiếp được kết nối với nhau thông qua hệ thống các chuyển mạch crossbar (Crossbar Switch). Thông qua hệ thống chuyển mạch này một RC có thể nhận kết quả tính toán từ một RC bất kỳ ở hàng ngay trên nó, đặc biệt RC trong hàng đầu tiên có thể lấy kết quả tính toán từ RC hàng cuối cùng.



Hình 2 - 5: : Cấu trúc TOP-DOWN của mảng RCA.

Các cổng vào/ra của mảng RCA được định nghĩa như chỉ ra trong Bảng 2-2.

Bảng 2-2: Tín hiệu vào ra của khối RCA8×8.

Tín hiệu	Hướng vào/ra	Độ rộng (bits)	Ý nghĩa

CLK	IN	1	Tín hiệu xung nhịp đồng bộ
ENABLE	IN	1	Tín hiệu Enable đồng bộ, tích cực mức cao
Reset_N	IN	1	Tín hiệu reset không đồng bộ, tích cực mức thấp, xóa nội dung các thanh ghi bên trong.
FIFO_WIRE_IN	IN	32*8	Nguồn dữ liệu là INPUT FIFO
GRF	IN	32*8	Tập các thanh ghi toàn cục (Global Register File)
ROUTER_A_CONFIG	IN	8*8*10	Thông tin cấu hình cho ROUTER toán hạng A
ROUTER_B_CONFIG	IN	8*8*10	Thông tin cấu hình cho ROUTER toán hạng B
ROUTER_LOR_CONFIG	IN	8*8*10	Thông tin cấu hình cho ROUTER của thanh ghi LOR
OPCODE_IN	IN	8*8*5	Mã toán tử (một thao tác datapath có thể xử lý)
MUX_C_CONFIG	IN	8*8*1	Thông tin cấu hình cho bộ hợp kênh chọn lối vào C
BIT16_8_DATA_A	IN	8*8*1	Chọn mức lõi của dữ liệu toán hạng A là 8 bit hay 16 bit
BIT16_8_DATA_B	IN	8*8*1	Chọn mức lõi của dữ liệu toán hạng B là 8 bit hay 16 bit
BIT16_8_DATA_LOR	IN	8*8*1	Chọn mức lõi của dữ liệu thanh ghi LOR là 8 bit hay 16 bit
BIT16_8_GRF	IN	8*8*1	Chọn mức lõi của dữ liệu từ thanh ghi GRF là 8 bit hay 16 bit
BIT16_8_OPERATOR	IN	8*8*1	Mức lõi của PE là 8 bit hay 16 bit (các Operator của PE

			thực hiện trên dữ liệu 8 hay 16 bit)
PE_OUT	OUT	8*8*16	Cổng xuất kết quả tính toán từ PE
LOR_OUT	OUT	8*8*16	Cổng xuất dữ liệu từ thanh ghi cục bộ LOR

A) RC_LINE

Một RC_LINE là một hàng gồm 8 RC trong cấu trúc mảng RCA. Chức năng các cổng vào/ra của RC_LINE được định nghĩa trong Bảng 2-3.

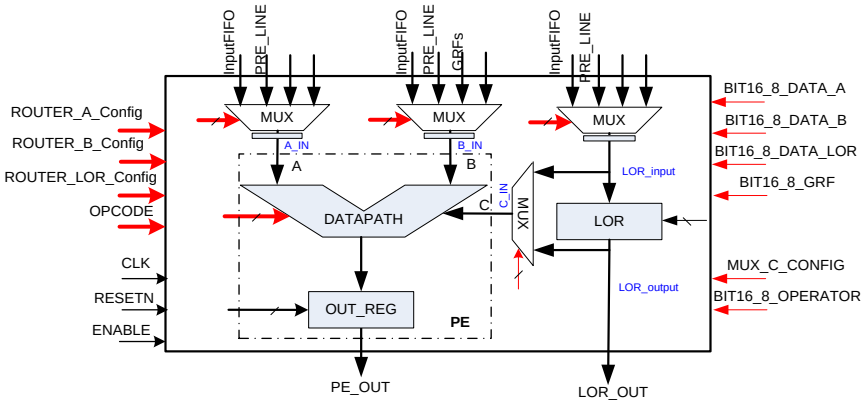
Bảng 2-3 Tín hiệu vào ra của khối RC_LINE.

Tín hiệu	**Hướng vào/ra**	**Độ rộng (bits)**	**Ý nghĩa**
CLK	IN	1	Tín hiệu xung nhịp đồng bộ
ENABLE	IN	1	Tín hiệu Enable đồng bộ, tích cực mức cao
Reset_N	IN	1	Tín hiệu reset không đồng bộ, tích cực mức thấp, xóa nội dung các thanh ghi bên trong.
PRELINE_OUT_IN	IN	8*2*16	Dữ liệu vào được lấy từ hàng RC trước. (Hàng 0 lấy dữ liệu từ hàng 7)
FIFO_WIRE_IN	IN	32*8	Nguồn dữ liệu là INPUT FIFO
GRF	IN	32*8	Tập các thanh ghi toàn cục (Global Register File)
ROUTER_A_CONFIG	IN	8*10	Thông tin cấu hình cho ROUTER toán hạng A
ROUTER_B_CONFIG	IN	8*10	Thông tin cấu hình cho ROUTER toán hạng B
ROUTER_LOR_CONFIG	IN	8*10	Thông tin cấu hình cho ROUTER của thanh ghi LOR
OPCODE_IN	IN	8*5	Mã toán tử (một thao tác datapath có thể xử lý))
MUX_C_CONFIG	IN	8*1	Thông tin cấu hình cho bộ hợp kênh chọn lối vào C
BIT16_8_DATA_A	IN	8*1	Chọn mức lõi của dữ liệu toán hạng A là 8 bit hay 16 bit

BIT16_8_DATA_B	IN	8*1	Chọn mức lõi của dữ liệu toán hạng B là 8 bit hay 16 bit
BIT16_8_DATA_LOR	IN	8*1	Chọn mức lõi của dữ liệu thanh ghi LOR là 8 bit hay 16 bit
BIT16_8_GRF	IN	8*1	Chọn mức lõi của dữ liệu từ thanh ghi GRF là 8 bit hay 16 bit
BIT16_8_OPERATOR	IN	8*1	Mức lõi của PE là 8 bit hay 16 bit (các Operator của PE thực hiện trên dữ liệu 8 hay 16 bit)
PE_OUT	OUT	8*16	Cổng xuất kết quả tính toán từ các PE trong RC
LOR_OUT	OUT	8*16	Cổng xuất dữ liệu từ thanh ghi cục bộ LOR của các RC trong hàng

B) Phần tử RC

RC (Hình 2-6) là đơn vị xử lý có khả năng tái cấu hình cơ sở của mảng RCA. Mỗi RC có một phần tử xử lý PE (Processing Element) có thể thực hiện các phép tính trên 2 hoặc 3 toán hạng 8/16 bit có dấu, chẳng hạn như các phép tính logic và số học, phép nhân, và các phép tính được dùng nhiều trong xử lý đa phương tiện và truyền thông (như dịch barrel, dịch làm tròn, tổng sai số tuyệt đối, v.v). Mỗi RC cũng có một thanh ghi dữ liệu cục bộ LOR (LOcal Register), dùng cho việc hiệu chỉnh chu kỳ làm việc của đường ống khi ánh xạ vòng lặp lên RCA, hoặc để lưu các hệ số trong suốt thời gian RCA làm việc. Dữ liệu đầu vào của mỗi RC có thể từ INPUT_FIFO, từ RC hàng ngay trên nó hoặc từ thanh ghi GRF. Chức năng của các cổng vào/ra của mỗi RC được mô tả trong Bảng 2-4.



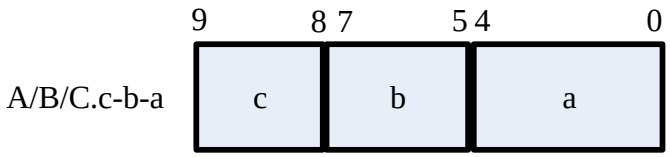
Hình 2 - 6: Cấu trúc của một phần tử RC.

Bảng 2-4: Mô tả các tín hiệu của RC.

Tín hiệu	Hướng vào/ra	Độ rộng (bits)	Ý nghĩa
PRELINE_OUT	IN	16*16 = 256	Nguồn dữ liệu từ hàng RC ngay trên nó, dữ liệu có thể là kết quả của PE cũng có thể là dữ liệu lối ra của thanh ghi cục bộ LOR
FIFO_WIRE	IN	32*8	Nguồn dữ liệu là INPUT FIFO
GRF	IN	32*8	Tệp các thanh ghi toàn cục (Global Register File)
ROUTER_A_CONFIG	IN	10	Thông tin cấu hình cho ROUTER toán hạng A
ROUTER_B_CONFIG	IN	10	Thông tin cấu hình cho ROUTER toán hạng B
ROUTER_LOR_CONFIG	IN	10	Thông tin cấu hình cho ROUTER của thanh ghi LOR
OPCODE_IN	IN	5	Mã toán tử (một thao tác datapath có thể xử lý))
MUX_C_CONFIG	IN	1	Thông tin cấu hình cho bộ hợp kênh chọn lối vào C
BIT16_8_DATA_A	IN	1	Chọn mức lõi của dữ liệu toán hạng A là 8 bit hay 16 bit
BIT16_8_DATA_B	IN	1	Chọn mức lõi của dữ liệu toán hạng B là 8 bit hay 16 bit
BIT16_8_DATA_LOR	IN	1	Chọn mức lõi của dữ liệu thanh ghi LOR là 8 bit hay 16 bit
BIT16_8_GRF	IN	1	Chọn mức lõi của dữ liệu từ thanh ghi GRF là 8 bit hay 16 bit
BIT16_8_OPERATOR	IN	1	Mức lõi của PE là 8 bit hay 16 bit (các Operator của PE thực hiện trên dữ liệu 8 hay 16 bit)
PE_OUT	OUT	16	Cổng xuất kết quả tính toán từ PE
LOR_OUT	OUT	16	Cổng xuất dữ liệu từ thanh ghi

			cục bộ LOR
CLK	IN	1	Tín hiệu xung nhịp đồng bộ
ENABLE	IN	1	Tín hiệu Enable đồng bộ, tích cực mức cao
Reset_N	IN	1	Tín hiệu reset không đồng bộ, tích cực mức thấp, xóa lối ra Result_REG

Ý nghĩa các bit cho các phần ROUTER_A_CONFIG, ROUTER_B_CONFIG, ROUTER_LOR_CONFIG (để đơn giản trong phần mô tả tiếp theo ký hiệu là A, B, C một cách tương ứng) là tương tự và được định nghĩa như sau (Hình 2 - 7):



Hình 2 - 7: Định dạng thông tin cấu hình các phần A, B, C.

a: 5 bit, ý nghĩa của a phụ thuộc vào giá trị của c, cụ thể:

- khi c = 2'b00, a xác định địa chỉ cột của dữ liệu trong FIFO (xác định vị trí cụ thể trong số 32Byte sau mỗi lần FIFO xuất dữ liệu).
- khi c = 2'b01, chỉ 4 bit LSB của a được dùng để chọn kết quả lối ra từ một PE hay LOR từ hàng ngay trên nó.
- khi c = 2'b10, a xác định địa một giá trị hằng số từ thanh ghi GRF.

b: 3 bit, chưa sử dụng.

c: 2 bit, nguồn dữ liệu lối vào của RC.

- [00] input FIFO,
- [01] hàng RC ngay phía trên,
- [10] GRF,
- [11] không nhận dữ liệu mới.

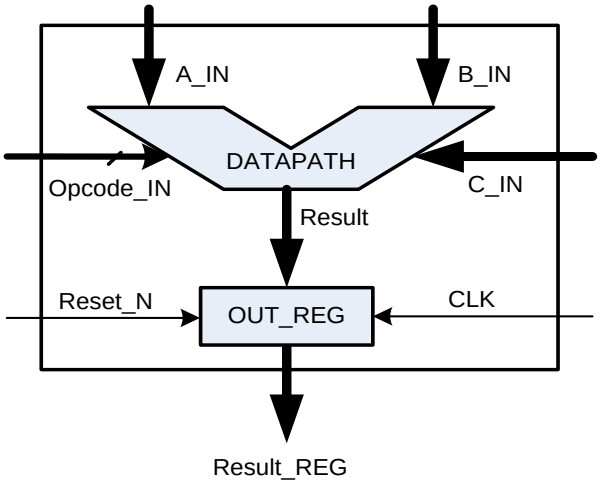
Chi tiết xem Bảng 2-5

Bảng 2-5: Định nghĩ thông tin cấu hình nguồn dữ liệu lối vào mỗi RC.

Thông tin cấu hình lối vào c	Thông tin cấu hình lối vào a	Nguồn dữ liệu lối vào
Khi c = 2'b01 nguồn dữ liệu lối vào là từ lối ra của RC hàng trước đó? (Nếu hàng hiện tại là hàng 0 (hàng đầu tiên) hàng trên nó sẽ là hàng 7 (hàng cuối cùng)).	8'bxxxx0000	Lối ra của PE[x][0]
	8'bxxxx0001	Lối ra của PE[x][1]
	8'bxxxx0010	Lối ra của PE[x][2]
	8'bxxxx0011	Lối ra của PE[x][3]
	8'bxxxx0100	Lối ra của PE[x][4]
	8'bxxxx0101	Lối ra của PE[x][5]
	8'bxxxx0110	Lối ra của PE[x][6]
	8'bxxxx0111	Lối ra của PE[x][7]
	8'bxxxx1000	Lối ra của LOR[x][0]
	8'bxxxx1001	Lối ra của LOR[x][1]
	8'bxxxx1010	Lối ra của LOR[x][2]
	8'bxxxx0011	Lối ra của LOR[x][3]
	8'bxxxx1100	Lối ra của LOR[x][4]
	8'bxxxx1101	Lối ra của LOR[x][5]
	8'bxxxx1110	Lối ra của LOR[x][6]
	8'bxxxx1111	Lối ra của LOR[x][7]
Khi c = 2'b00 nguồn dữ liệu lối vào là từ Input FIFO	Cả 5 bit của a	Dữ liệu nằm ở cột thứ mấy của Input FIFO
Khi c = 2'b10 nguồn dữ liệu lối vào là từ một trong hai thanh ghi hằng số	a[4..0]	0~31: chọn thanh ghi nào trong GRF
Khi c = 2'b11 không nhận lối vào	Không dùng	Không dùng

C) Phần tử xử lý PE (Processing Element)

Phần tử xử lý PE bao gồm đơn vị thực thi tính toán (Datapath) thực hiện tất các các chức năng tính toán và một thanh ghi kết quả lối ra. Giao diện vào/ra và chức năng của từng cổng được mô tả trong Bảng 2-6.

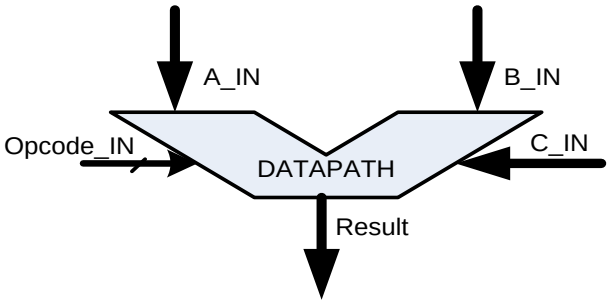


Hình 2 - 8: Cấu trúc của một PE.

Bảng 2-6: Mô tả các tín hiệu của PE.

Tín hiệu	**Hướng vào/ra**	**Độ rộng (bits)**	**Ý nghĩa**
A_IN	IN	16	Cổng nhập dữ liệu cho toán hạng A
B_IN	IN	16	Cổng nhập dữ liệu cho toán hạng B
C_IN	IN	16	Cổng nhập dữ liệu cho toán hạng C
OPCODE_IN	IN	5	Mã toán tử (một thao tác datapath có thể xử lý)
RESULT	OUT	16	Cổng xuất kết quả tính toán không đồng bộ (không qua thanh ghi)
RESULT_REG	OUT	16	Cổng xuất kết quả tính toán đồng bộ (qua thanh ghi)
BIT16_8_OPERATOR	IN	1	Mức lõi của PE là 8 bit hay 16 bit (các Toán tử của PE thực hiện trên dữ liệu 8 hay 16 bit)
CLK	IN	1	Tín hiệu xung nhịp đồng bộ
Reset_N	IN	1	Tín hiệu reset tích cực mức thấp, không đồng bộ, dung xóa lối ra Result_REG

D) Đơn vị thực thi tính toán (Datapath)

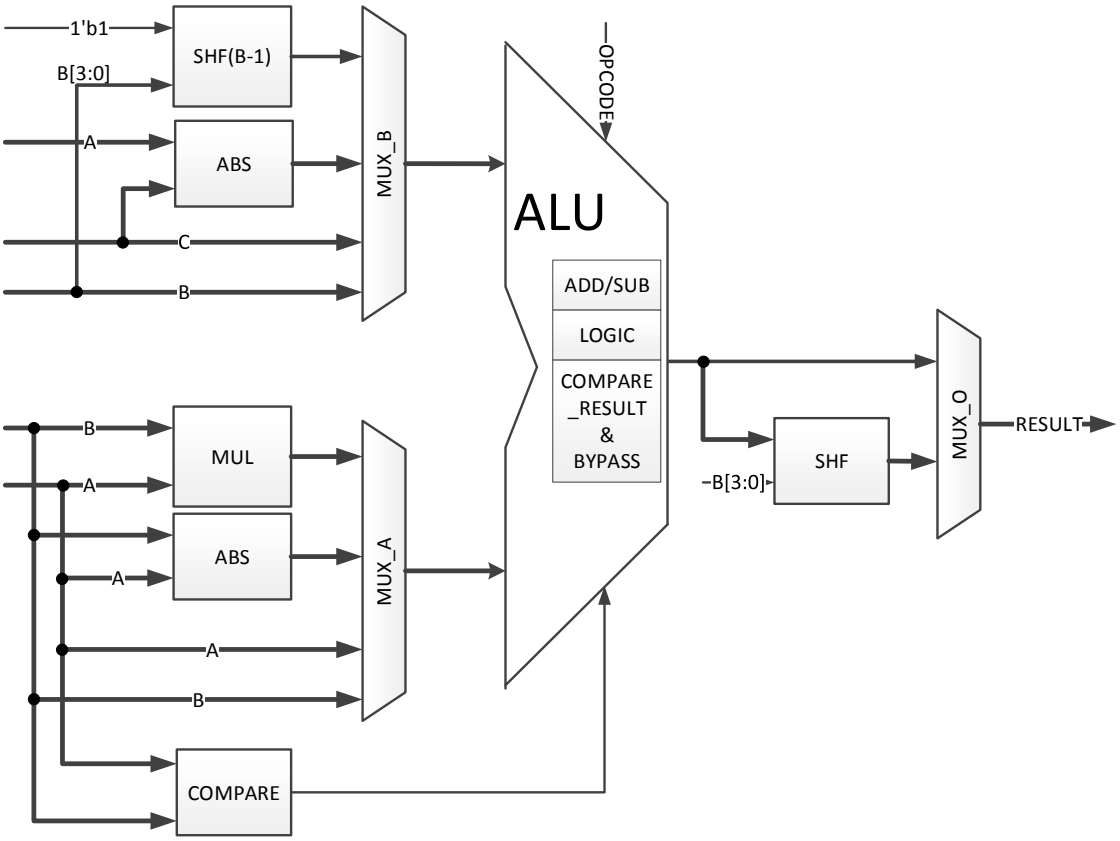


Hình 2 - 9: Giao diện vào/ra của datapath.

Bảng 2-7: Mô tả các tín hiệu vào ra của Datapath.

Tín hiệu	Hướng vào/ra	Độ rộng (bits)	Ý nghĩa
A_IN	IN	16	Cổng nhập dữ liệu cho toán hạng A
B_IN	IN	16	Cổng nhập dữ liệu cho toán hạng B
C_IN	IN	16	Cổng nhập dữ liệu cho toán hạng C
OPCODE_IN	IN	5	Mã toán tử (một thao tác datapath có thể xử lý)
RESULT	OUT	16	Cổng xuất kết quả tính toán

Đơn vị thực thi tính toán (Datapath) thực hiện các phép tính được liệt kê trong Bảng 2-1. Các khối thực hiện các phép toán được thiết kế như Hình 2 - 10. Khối ALU được thiết kế để thực hiện các phép tính số học và logic cơ bản. Các khối thực hiện phép dịch một bit, dịch 3 bit, tính trị tuyệt đối của một hiệu (ABS), phép so sánh được đặt ngoài khối ALU. Thông qua việc kết hợp một cách hợp lý các khối này với khối ALU sẽ cho phép phần tử xử lý PE thực hiện các phép tính còn lại, nhờ đó giúp tối giản hóa diện tích thực thi các PE.



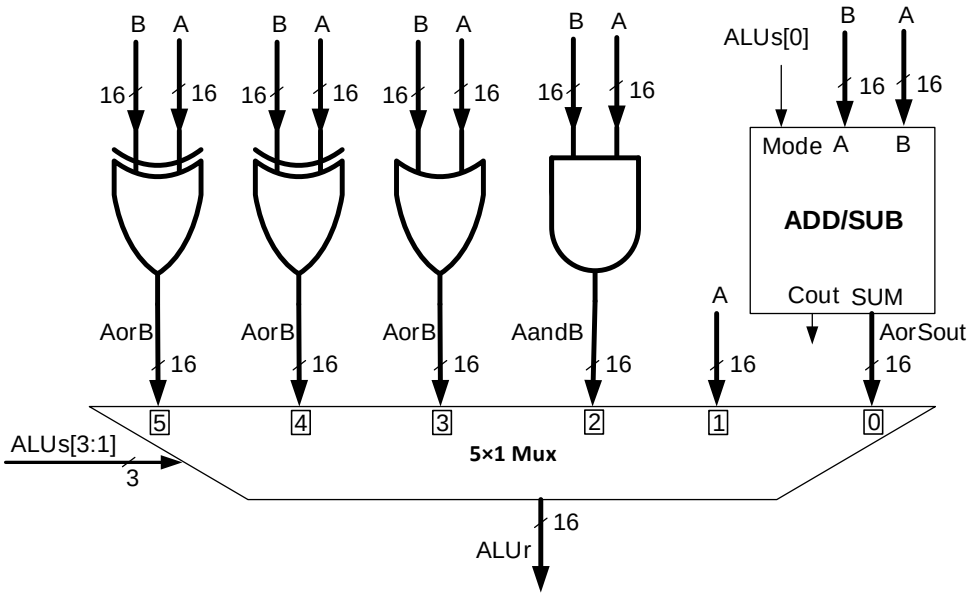
Hình 2 - 10. Sơ đồ thiết kế các khối thực hiện các phép tính trên datapath.

(1) Khối ALU

Khối ALU (Hình 2 - 11) được thiết kế để thực hiện các phép tính số học và logic cơ bản: bộ cộng, bộ trừ, bộ truyền qua (by pass) và bộ logic cơ bản (AND, OR, XOR, NOT, NXOR) như mô tả trong bảng 2-8 dưới đây.

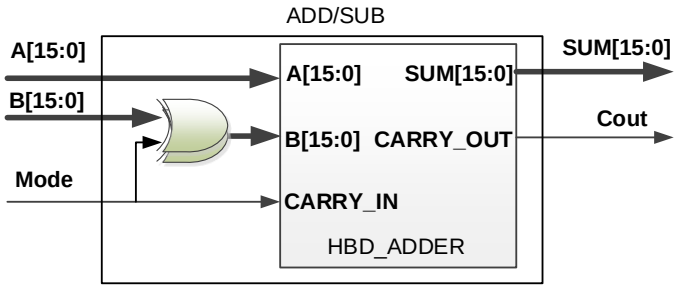
Bảng 2-8: Mô tả các phép toán được thực hiện trên khối ALU

Phân loại	Opcode	Phép tính	Ý nghĩa	Từ gợi nhớ
Arithmetic operations	5'b00000 (0)	A+B	Phép cộng A và B	ADD
	5'b00001 (1)	A−B	Phép trừ đi B đơn vị từ A	SUB
Bypass A	5'b00101 (5)	A	Bypass A	A
Bitwise operations	5'b00110 (6)	A&B	Toán tử AND	AND
	5'b00111 (7)	A|B	Toán tử OR	OR
	5'b01000 (8)	A^B	Toán tử XOR	XOR
	5'b01001 (9)	~(A^B)	Toán tử NOT XOR	NXOR



Hình 2 - 11: Khối ALU.

Hình 2 - 12 chỉ ra cấu trúc phần cứng đơn vị chức năng ADD/SUB thực hiện cả phép tính cộng và trừ được xây dựng trên cơ sở một bộ cộng hai số 16-bit và một số mạch logic bổ trợ. Tín hiệu Mode được dùng để chọn chế độ hoạt động của bộ ADD/SUB là cộng (Mode = 0) hay trừ (Mode = 1).

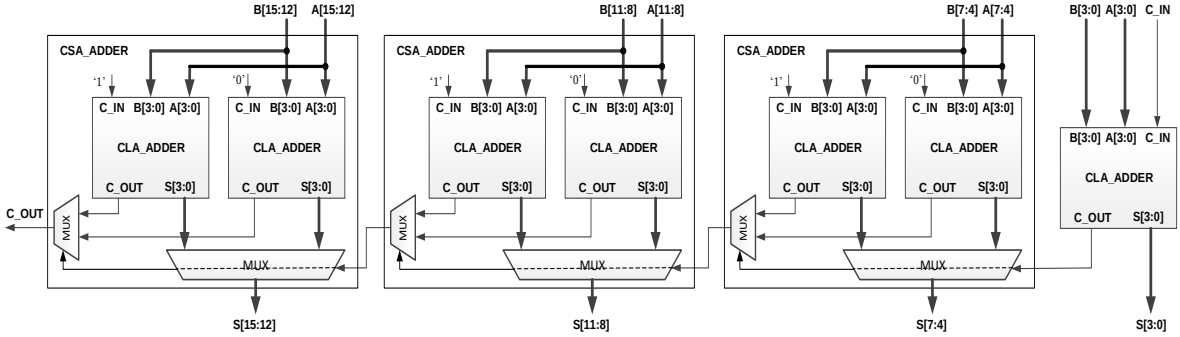


Hình 2 - 12: Đơn vị chức năng ADD/SUB thực hiện phép tính cộng và trừ hai số 16-bit.

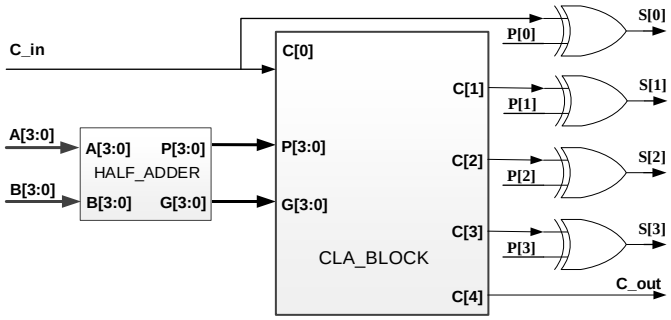
Trong đơn vị ADD/SUB cần một bộ cộng để thực hiện phép tính cộng hoặc trừ. Nói chung, có 2 phương pháp chính để thực hiện bộ cộng là: bộ cộng lan truyền cờ nhớ (Ripple carry adder: RCA và bộ cộng với cờ nhớ được tính trước (Carry Lookahead Adder: CLA) [13]. Bộ cộng RCA có ưu điểm là đơn giản, dễ thực hiện, giá thành thực thi phần cứng thấp. Tuy nhiên nhược điểm của bộ cộng RCA là trễ lan truyền cờ nhớ lớn, dẫn đến tốc độ tính toán chậm. Bộ cộng CLA cải thiện tốc độ tính toán bằng cách thực hiện các mạch

logic cho phép tính toán đồng thời tất cả các cờ nhớ ở giá thành phần cứng cao. Mạch logic để thực hiện bộ cộng CLA n-bit sẽ đặc biệt phức tạp với giá trị n lớn. Trên thực tế phương pháp tính toán trước cờ nhớ chỉ được dùng để thực hiện các mô-đun cộng hai số 4-bit.

Do đó, để cân bằng giữa yêu cầu về tốc độ thực hiện phép cộng và chi phí phần cứng, luận văn chọn phương pháp thực hiện lai ghép (HBD_ADDER) giữa bộ cộng CLA và RCA. Các số hạng lối vào A, B được tách thành các nible 4-bit và được thực hiện tính tổng đồng thời bởi 4 module bộ cộng CLA 4-bit. Cờ nhớ từ các khối CLA 4-bit được lan truyền giống như cách thực hiện bộ cộng RCA và sẽ được dùng để lựa chọn cờ nhớ để tính kết quả tổng của A và B cuối cùng. Sơ đồ khối bộ cộng 16-bit lai ghép được chỉ ra trong Hình 2-13. Mỗi khối CLA_ADDER là một bộ cộng hai số 4-bit theo phương pháp CLA có cấu trúc như được chỉ ra trong Hình 2 - 13



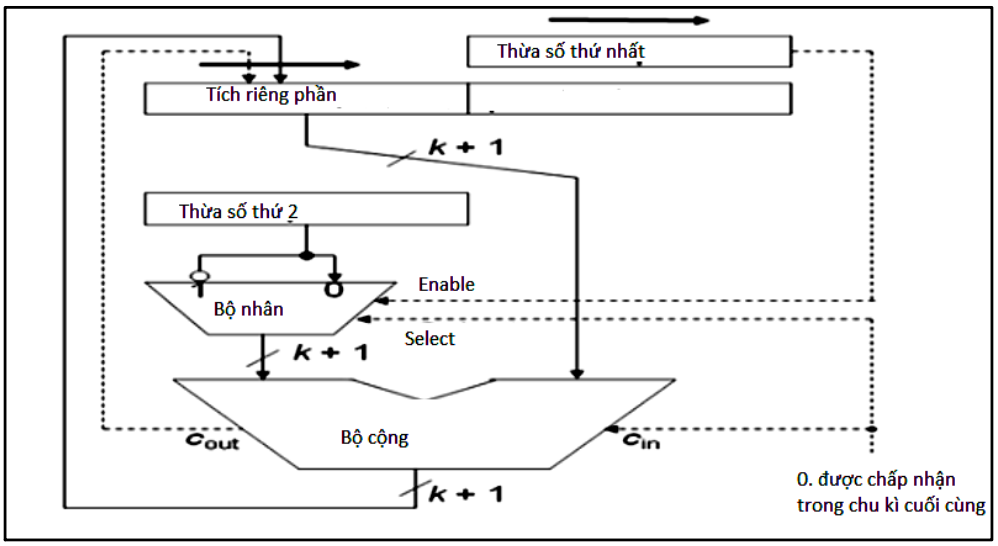
Hình 2 - 13: Cấu trúc của bộ cộng lai ghép HBD_ADDER.



Hình 2 - 14: Bộ cộng CLA 4-bit.

(2) Khối MUL

Khối MUL được thiết kế theo phương pháp Baugh-Wooley [15] . Cấu trúc phần cứng của bộ nhân Baugh-Wooley được mô tả như trong Hình 2 - 15



Hình 2 - 15: Sơ đồ cấu trúc phần cứng của bộ nhân Baugh-Wooley [15]

Phương pháp Baugh-Wooley áp dụng cho cả phép nhân có dấu và không dấu. Toán tử có dấu được biểu diễn dưới dạng số bù hai để đảm bảo chắc chắn rằng dấu của các tích riêng phần luôn là dấu dương. Các tích riêng phần được điều chỉnh sao cho dấu âm được chuyển đến bước cuối cùng. Cấu trúc bộ nhân 4 bit Baugh-Wooley được thể hiện như trong Hình 2 - 16 .

Về mặt toán học có thể mô tả phương pháp nhân Baugh-Wooley như sau:

Giả sử có hai số bù hai 4 bít X và Y:

$$X = -2^3 x_3 + \sum_{i=0}^{2} x_i 2^i \qquad Y = -2^3 y_3 + \sum_{i=0}^{2} y_i 2^i \qquad (2\text{-}1)$$

Tích của X và Y được thực hiện như sau:

$$XY = x_3 y_3 2^6 - \sum_{i=0}^{2} x_i y_3 2^{i+3} - \sum_{j=0}^{2} x_3 y_j 2^{j+3} + \sum_{i=0}^{2}\sum_{j=0}^{2} x_i y_j 2^{i+j} \qquad (2\text{-}2)$$

Đối với số bù hai ta luôn có:

$$-\sum_{i=0}^{3} x_i 2^i = -2^4 + \sum_{i=0}^{3} \overline{x_i 2^i} + 1 \qquad (2\text{-}3)$$

Do đó tích trên trở thành:

$$XY = x_3y_3 2^6 + \sum_{i=0}^{2} \overline{x_i y_3} 2^{i+3} + 2^3 - 2^6 + \sum_{i=0}^{2} \overline{x_3 y_j} 2^{j+3} + 2^3 - 2^6 + \sum_{i=0}^{2}\sum_{j=0}^{2} x_i y_j 2^{i+j}$$

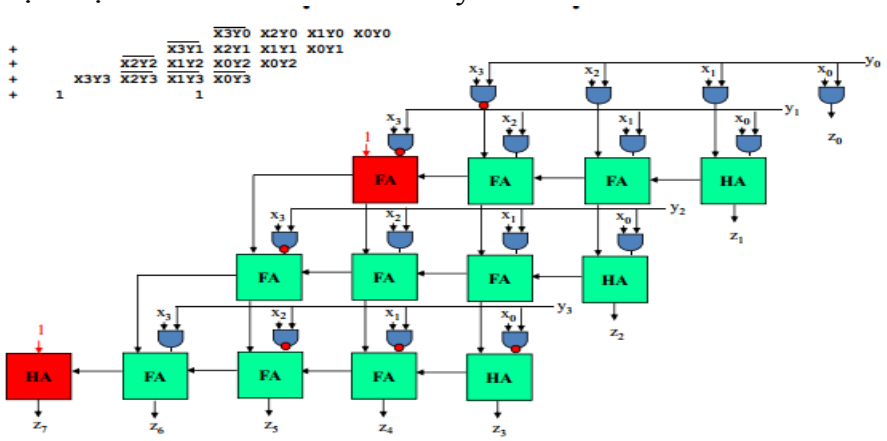
$$= x_3y_3 2^6 + \sum_{i=0}^{2} \overline{x_i y_3} 2^{i+3} + \sum_{i=0}^{2} \overline{x_3 y_j} 2^{j+3} + \sum_{i=0}^{2}\sum_{j=0}^{2} x_i y_j 2^{i+j} + 2^4 - 2^7$$

$$= -2^7 + x_3y_3 2^6 + (\overline{x_2y_3} + \overline{x_3y_2})2^5 + (\overline{x_1y_3} + \overline{x_3y_1} + x_2y_2 + 1)2^4 \qquad (2\text{-}4)$$

$$+(\overline{x_0y_3} + \overline{x_3y_0} + x_1y_2 + x_2y_1)2^3 + (x_0y_2 + x_1y_1 + x_2y_0)2^2 +$$

$$(x_0y_1 + x_1y_0)2^1 + x_0y_0\, 2^0$$

Mô hình thực hiện như Hình 2 - 16 dưới đây.



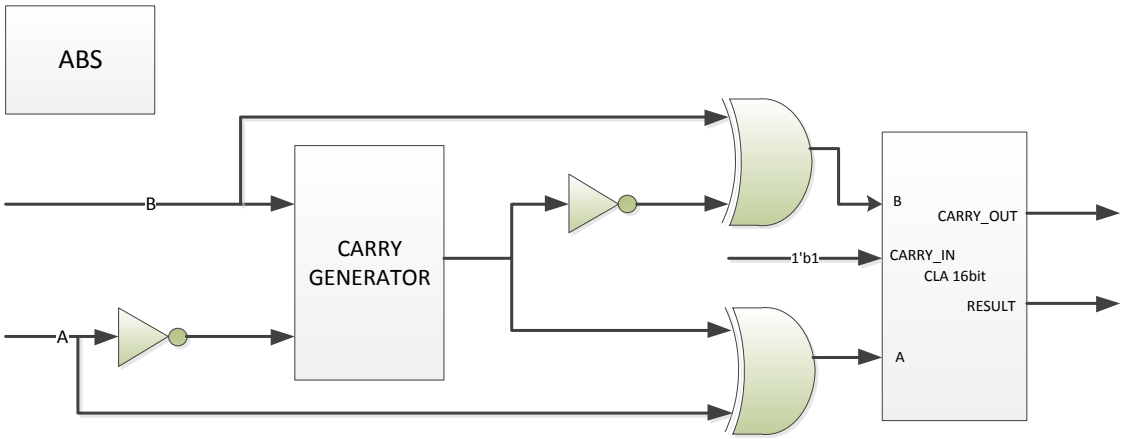
Hình 2 - 16. Bộ nhân MUL 4 bít [15]

(3) Khối ABS

Đối với số nhị phân, ta luôn có:

$$|A-B| = \begin{cases} A-B \text{ if } A>B \\ B-A \text{ if } A<B \end{cases} = \begin{cases} A+\overline{B}+1 & \text{if } A>B \\ \overline{A}+B+1 & \text{if } A<B \end{cases} \qquad (2\text{-}5)$$

Mạch logic thực hiện khối như sau:



Hình 2 - 17: Sơ đồ thực hiện khối ABS

Bộ Carry generator sẽ phát hiện giá trị nhỏ hơn trong hai giá trị A, B. Trong trường hợp B nhỏ hơn A, không có giá trị cờ nhớ được tạo ra, B sẽ bị đảo, ngược lại A sẽ bị đảo. Điều này được thực hiện bằng mạch XOR phía sau. Do đó, giá trị $(A, \overline{B})$ hoặc $(\overline{A}, \mathrm{B})$ sẽ được truyền tới bộ CLA

E) Thanh ghi cục bộ LOR

Bảng 2-9: Mô tả tín hiệu của thanh ghi cục bộ LOR.

Tín hiệu	Hướng vào/ra	Độ rộng (bits)	Ý nghĩa
LOR_IN	IN	16	Cổng nhập dữ liệu
LOR_OUT	OUT	16	Cổng xuất dữ liệu
CLK	IN	1	Tín hiệu xung nhịp đồng bộ
Reset_N	IN	1	Tín hiệu reset tích cực mức thấp, xóa lối ra Result_REG

F) Bộ định tuyến Router

Có 2 loại bộ định tuyến với số lượng các cổng vào khác nhau được sử dụng để lấy dữ liệu từ các nguồn khác nhau vào RC.

- ROUTER_A: Bộ định tuyến cho cổng toán hạng A của datapath và lối vào thanh ghi LOR (Local Register) nội bộ của RC. ROUTER_A chỉ có 2 nguồn dữ liệu là Input_FIFO và PRE_LINE (hàng RC trên)

- ROUTER_B: Bộ định tuyến cho cổng toán hạng B của datapath: Có 3 nguồn dữ liệu là Input_FIFO, Tệp thanh ghi toàn cục GRF và PRE_LINE (hàng RC trên)

Bảng 2-10: Mô tả các tín hiệu của Router_A.

Tín hiệu	Hướng vào/ra	Độ rộng (bits)	Ý nghĩa
PRELINE_OUT	IN	16*16 = 256	Nguồn dữ liệu từ hàng RC ngay trên nó, dữ liệu có thể là kết quả của PE cũng có thể là dữ liệu lối ra của thanh ghi cục bộ LOR
FIFO_WIRE	IN	32*8	Nguồn dữ liệu là INPUT FIFO
ROUTER_CONFIG	IN	10	Thông tin cấu hình cho ROUTER
BIT16_8_DATA	IN	1	Mức lõi của dữ liệu từ INPUT

			FIFO là 8 bit hay 16 bit
A_OUT	OUT	16	Cổng lối ra của ROUTER
CLK	IN	1	Tín hiệu xung nhịp đồng bộ
ENABLE	IN	1	Tín hiệu cho phép xuất kết quả tính toán từ Datapath tới lối ra Result_REG (qua thanh ghi)
Reset_N	IN	1	Tín hiệu reset tích cực mức thấp, xóa lối ra Result_REG

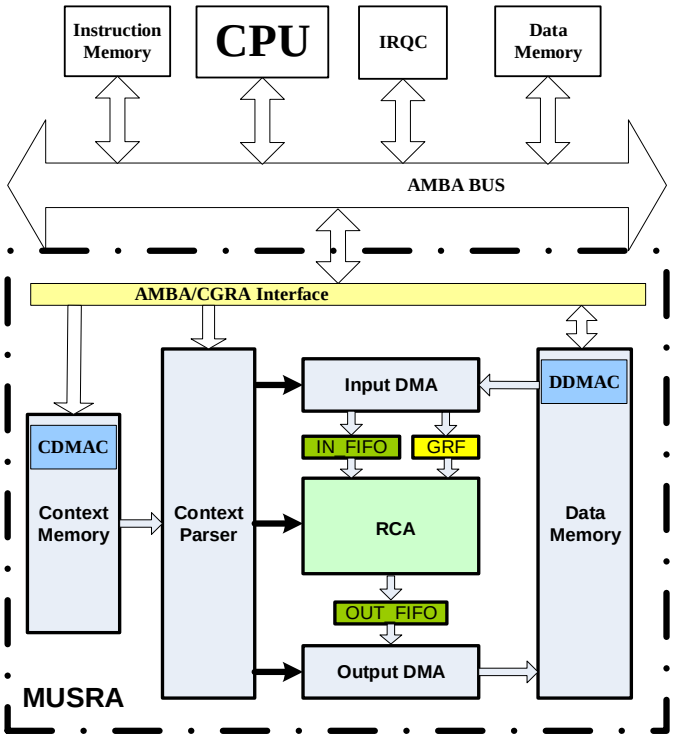
Bảng 2-11: Mô tả các tín hiệu của Router_B.

Tín hiệu	**Hướng vào/ra**	**Độ rộng (bits)**	**Ý nghĩa**
PRELINE_OUT	IN	16*16 = 256	Nguồn dữ liệu từ hàng RC ngay trên nó, dữ liệu có thể là kết quả của PE cũng có thể là dữ liệu lối ra của thanh ghi cục bộ LOR
FIFO_WIRE	IN	32*8	Nguồn dữ liệu là INPUT FIFO
GRF	IN	32*8	Tệp các thanh ghi toàn cục (Global Register File)
ROUTER_CONFIG	IN	10	Thông tin cấu hình cho ROUTER
BIT16_8_DATA	IN	1	Chọn mức lõi của dữ liệu từ INPUT FIFO là 8 bit hay 16 bit
BIT16_8_GRF	IN	1	Chọn mức lõi của của dữ liệu từ GRF là 8 bit hay 16 bit
B_OUT	OUT	16	Cổng lối ra của ROUTER
CLK	IN	1	Tín hiệu xung nhịp đồng bộ
ENABLE	IN	1	Tín hiệu cho phép xuất kết quả tính toán từ Datapath tới lối ra Result_REG (qua thanh ghi)
Reset_N	IN	1	Tín hiệu reset tích cực mức thấp, xóa lối ra Result_REG

CHƯƠNG 3 : KẾT QUẢ MÔ PHỎNG VÀ THỬ NGHIỆM

3.1 Mô hình mô phỏng của MUSRA

Cấu trúc MUSRA đề xuất đã được mô hình hóa ở mức RTL bằng ngôn ngữ VHDL. Ở mức hệ thống, một bộ mô phỏng có độ chính xác mức chu kì cũng được xây dựng nhằm đánh giá và kiểm thực sự thực thi của các ứng dụng khác nhau trên MUSRA. Bên cạnh MUSRA, bộ mô phỏng cũng sử dụng một bộ xử lý LEON3 và một số lõi IP khác từ thư viện Gaisler [14] như chỉ ra trong Hình 3-1. Sự kết nối giữa các lõi IP được thực thi qua bus AMBA 32-bit của ARM. LEON3 đóng vai trò như một đơn vị xử lý trung tâm quản lý và lập lịch tất cả các hoạt động của hệ thống. Bộ nhớ ngoài được sử dụng để giao tiếp dữ liệu giữa các tác vụ trên CPU và các tác vụ trên RCA. Sự đồng bộ giữa RCA, CPU và các DMA được thực thi theo cơ chế ngắt. Sau khi MUSRA hoàn tất mọi tác vụ đã được ấn định, nó sẽ tạo ra một ngắt qua khối IRQC để báo hiệu cho CPU và trả lại quyền điều khiển bus cho CPU. Các ngữ cảnh cấu hình cho các ứng dụng benchmark phải được lưu trước trong bộ nhớ cấu hình của MUSRA.



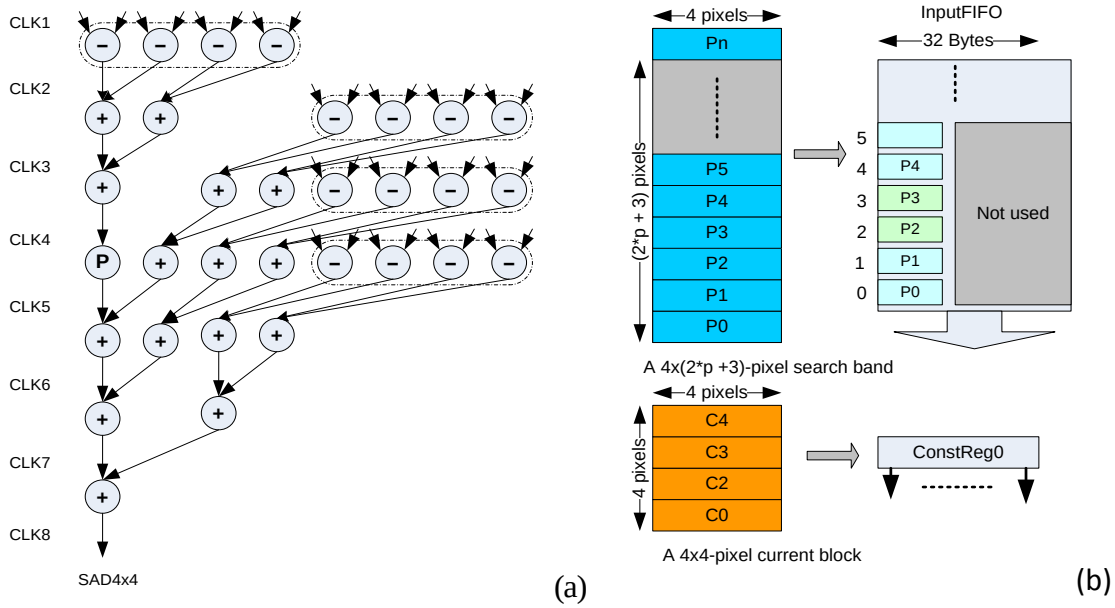
Hình 3- 1. Mô hình mô phỏng của MUSRA trong môi trường ModelSim.

3.2 Kịch bản kiểm chứng

Phần này sẽ trình bày việc ánh xạ một số ví dụ benchmark lên cấu trúc MUSRA đã được đề xuất. Các ví dụ benchmark được sử dụng bao gồm phép tính tổng các sai khác tuyệt đối (sum of absolute differences: SAD), phép tính tổng chuyển động (moving sum), phép nhân chập (convolution) và nhân ma trận-vectơ trong các thuật toán xử lý tín hiệu số. Mặc dù cấu trúc thực tế của RCA là 8×8, tuy nhiên để đơn giản trong việc trình bày trong phần này một số hình vẽ chỉ minh họa mảng RCA với kích thước 4×4.

3.2.1 Phép tổng sai phân (chênh lệch) tuyệt đối (SAD)

SAD được sử dụng rộng rãi trong xử lý số tín hiệu ảnh/video để đo lường sự tương quan giữa khối điểm ảnh hiện tại và khối điểm ảnh tham chiếu[13]. Trong Hình 3- 2(a) đưa ra một DFG cho tính tổng SAD của một khối 4×4 trên cấu trúc RCA. Trong sơ đồ này, sai khác tuyệt đối của mỗi hàng của từng khối 4×4 được thực hiện một cách độc lập trên từng tầng của đường ống. Tổ chức dữ liệu cho tính toán như chỉ ra trong Hình 3- 2(b), các pixel của khối mã hóa được sử dụng trong tất cả các lần tính toán do đó sẽ được dùng để khởi tạo thanh ghi hằng số RCA ở lần tính toán đầu tiên; các pixel của khối tham chiếu kích thước 4×(2×p+3) được nạp vào mảng RCA thông qua INPUT_FIFO, mỗi hàng 4 pixel (ký hiệu là P_i) một lần. Bằng cách sử dụng DFG và tổ chức dữ liệu như vậy, dữ liệu trùng lặp giữa hai ứng viên được khai thác để giảm đi số lần truy xuất bộ nhớ, đồng thời đảm bảo tất các RC của đường ống luôn ở trạng thái hoạt động.



Hình 3- 2: (a) DFG và (b) Tổ chức dữ liệu cho quá trình tính toán trên MUSRA.

3.2.2 Tổng chuyển động (Moving Sum)

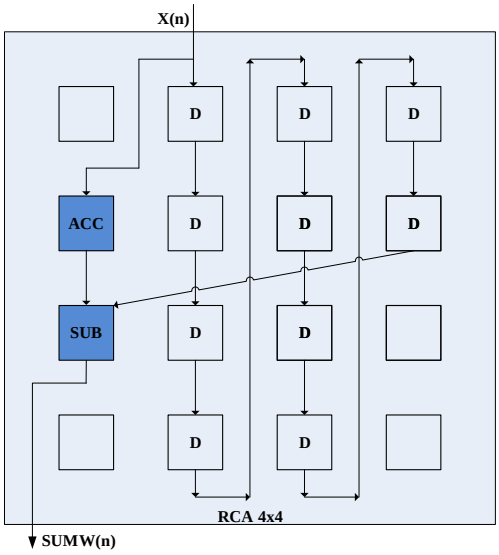
Thuật toán này được sử dụng phổ biến với các chuỗi dữ liệu biến đổi theo thời

gian để làm trơn các thăng giáng trong từng đoạn ngắn. Tổng chuyển động trên một cửa sổ trượt độ dài N ngang qua các thành phần lân cận của một chuỗi x(n) có công thức như sau[13]:

$$SUMW(n) = \sum_{k=0}^{N-1} x(n-k) \qquad (3\text{-}1)$$

Hoặc:
$$SUMW(n) = \left[SUMW(n-1) + x(n)\right] - x(n-N) \qquad (3\text{-}2)$$

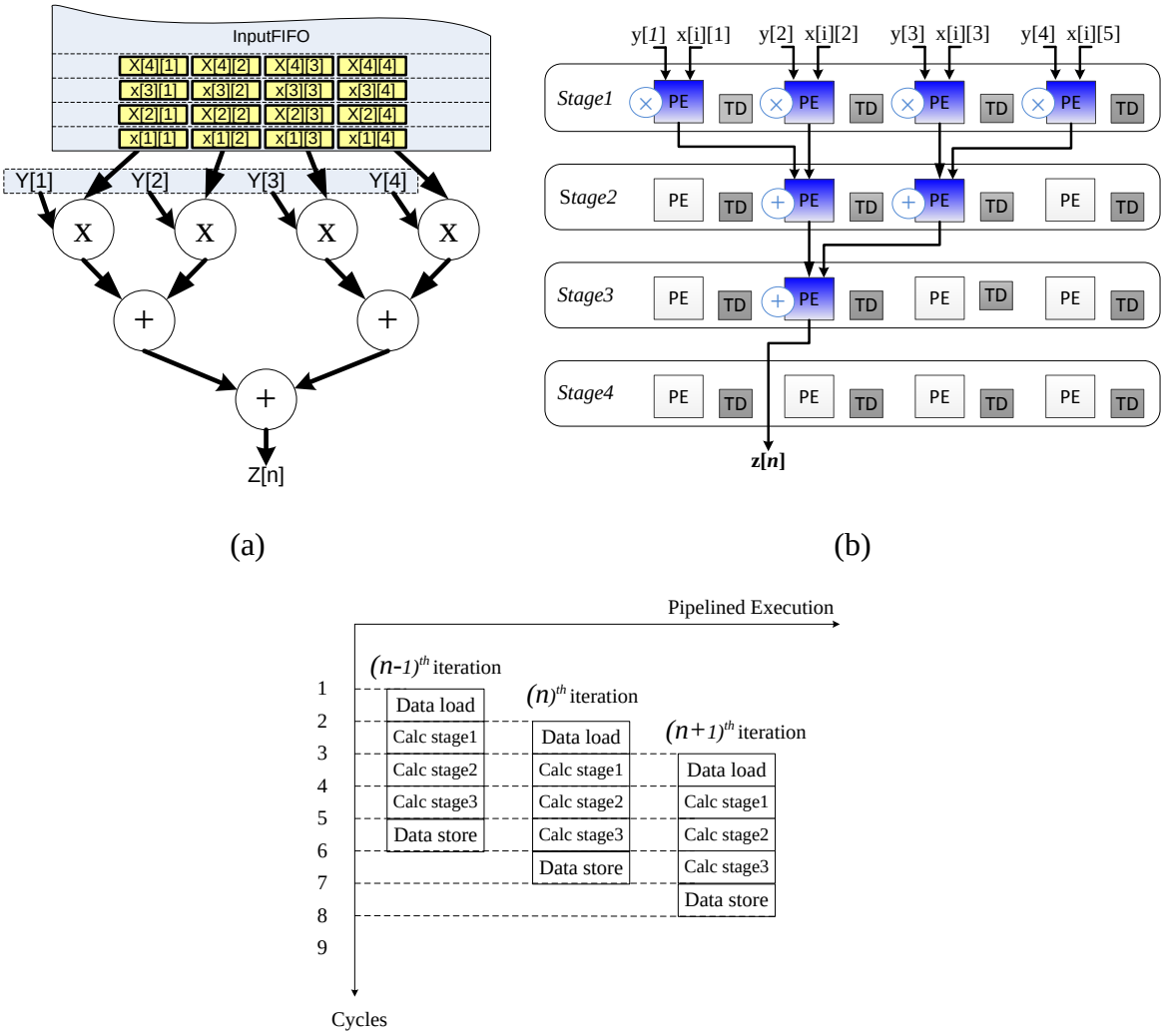
Hình 3- 3 chỉ ra việc ánh xạ công thức (2) lên MUSRA trong trường hợp N =10. Công việc này cũng minh họa vai trò và cách sử dụng thanh ghi LOR như một đơn vị trễ để tạo ra các phần tử *x(n-N).*



Hình 3- 3. Ánh xạ tổng chuyển động trên một cửa sổ trượt với độ dài N=10.

3.2.3 **Nhân vô hướng hai vector**

Hình 3- 4 (a) chỉ ra DFG cho phép nhân vô hướng một vector kích thước 4×4 với một vectơ 4×1. Ánh xạ của DFG tới mảng RCA và quá trình thực thi đường ống hóa được chỉ ra trong Hình 3- 4(b-c).



Hình 3- 4. DFG (a), Ánh xạ của DFG trên MUSRA (b), và sự thực thi được đường ống hóa (c) của phép nhân vô hướng hai vectơ.

Để ý rằng trong phép nhân vô hướng hai vectơ, vectơ đầu vào được sử dụng lặp lại để tính mỗi giá trị của vectơ lối ra, do đó các giá trị của vectơ lối ra sẽ được nạp vào một trong các thanh ghi GRF đã được định nghĩa trong RCA trước khi bắt đầu quá trình tính toán. Bằng cách thực hiện như vậy sẽ giảm đáng kể lượng truy xuất bộ nhớ.

3.2.4 **Tích chập**

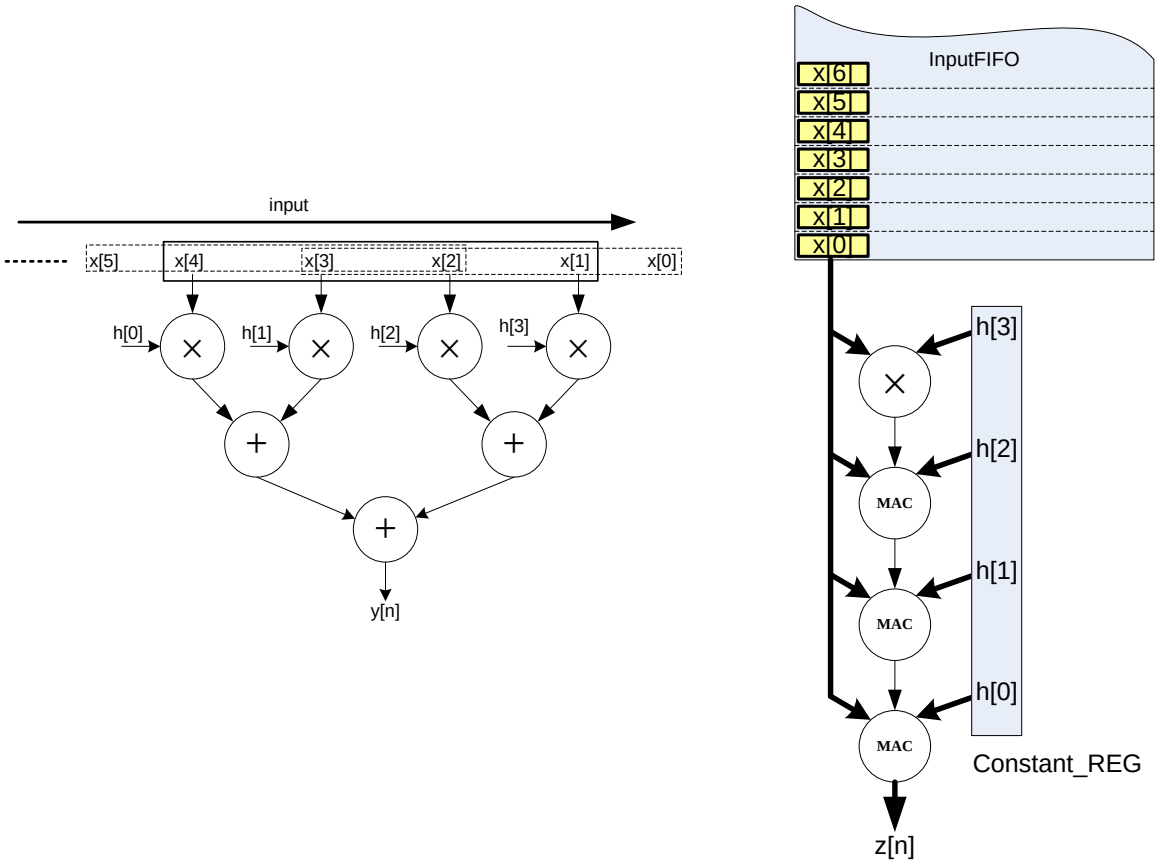
Tích chập (Convolution) thực hiện phép nhân trên hai hàm lối vào để tạo ra hàm lối ra theo công thức (3-3) [13]:

$$y[n] = \sum_{k=0}^{N-1} h[k] * x[n-k] \qquad (3\text{-}3)$$

Trong đó, x[k] là các mẫu của tín hiệu lối vào bộ lọc (x[k]=0 ∀k<0); h[k] là các hệ số của đáp ứng tần số của bộ lọc FIR; và y[n] các mẫu của tín hiệu lối ra của bộ lọc.

Tích chập được ứng dụng khá phổ biến trong lĩnh vực xử lý tín hiệu và hình ảnh. Một ứng dụng được biết đến khá rộng rãi của tích chập là bộ lọc FIR (Finite impulse response). Hình 3- 5(a) chỉ ra DFG dạng thực hiện trực tiếp của một bộ lọc FIR bậc 4.

Có thể thấy từ Hình 3- 5(a) rằng DFG cho bộ lọc FIR bậc 3 là tương tự DFG cho phép nhân vô hướng hai vectơ. Tuy nhiên cần chú ý rằng, luôn có 3 giá trị của chuỗi x[n] được sử dụng lại khi tính toán 2 giá trị liên tiếp của chuỗi y[n]. Do đó để khai thác được dữ liệu dùng lại giữa các lần lặp liên tiếp này, DFG như trong Hình 3- 5(b) sẽ được sử dụng. Trong DFG này, các mẫu x[n] sẽ được nhân với các hệ số h[n] tương ứng và cộng tích lũy với nhau để tạo ra mẫu lối ra z[n] theo phương thức tính toán đường ống 4 tầng. Các mẫu x[n] sẽ được "phun" lần lượt từ INPUT_FIFO và được quảng bá tới tất các RC. Như vậy tại một thời điểm chỉ có một mẫu x[n] được xuất ra bởi INPUT_FIFO và mỗi mẫu chỉ xuất một lần do đó sẽ giảm đi một lượng đáng kể băng thông truy xuất bộ nhớ.



(a) (b)

Hình 3- 5: DFG thực hiện một bộ lọc FIR bậc 3.

3.3 Kết quả thực nghiệm và đánh giá

3.3.1 Kết quả tổng hợp phần cứng

Mô hình RTL của mảng RCA8×8 đã được tổng hợp trên công nghệ FPGA Virtex-7

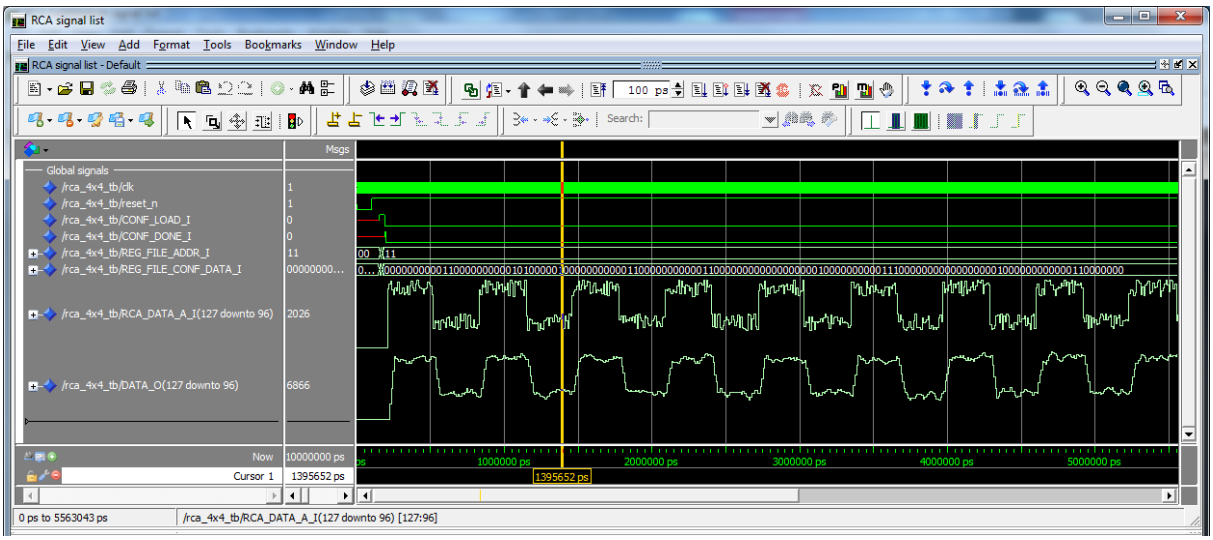
(XC7VX485) bằng bộ công cụ thiết kế Vivado Design Suite của hãng Xilinx. Bảng 3-1 chỉ ra tài nguyên phần cứng của chip FPGA được sử dụng cho thực hiện mảng RCA8×8. Chỉ có 2 loại tài nguyên phần cứng của chip FPGA được sử dụng cho thực hiện mảng RCA8×8 là Flip-Flop và LUT với tỷ lệ sử dụng là 0,84% và 23,13% một cách tương ứng. Với lượng tài nguyên sử dụng tương đối thấp như vậy, nó là khả thi để tích hợp mảng RCA8×8 trong các hệ thống tính toán hiệu năng cao.

Bảng 3- 1 Kết quả tổng hợp mảng RCA8×8 trên công nghệ FPGA Virtex-7 ((xc7vx485t).

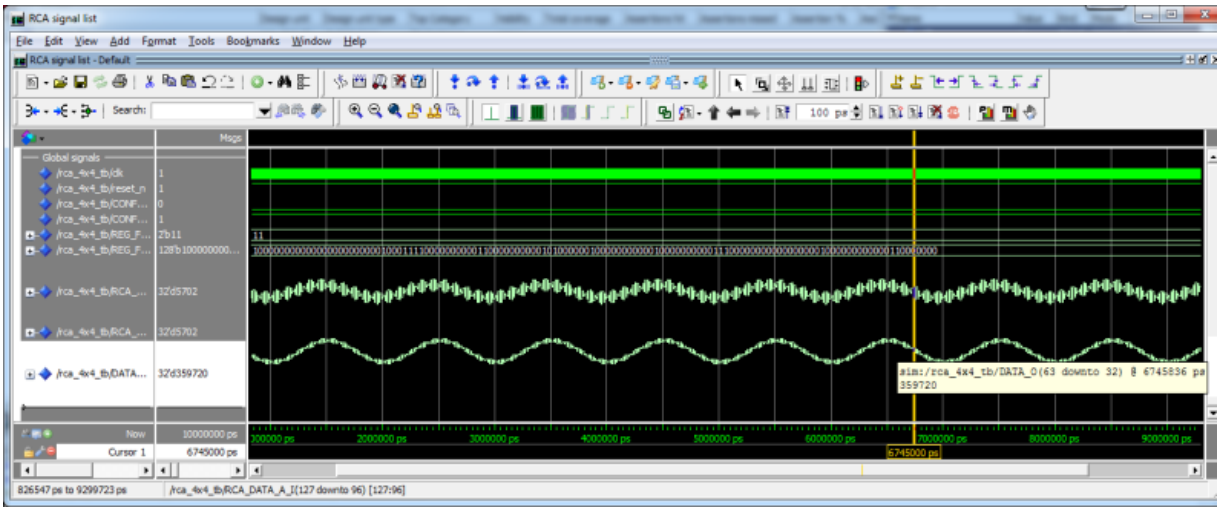
Loại tài nguyên	**Tài nguyên được sử dụng**	**Tài nguyên có sẵn**	**Tỷ lệ sử dụng tài nguyên (%)**
Flip-Flop	5120	607200	0,84
LUTs	70209	303600	23,13

3.3.2 **Kết quả mô phỏng**

Các ví dụ benchmark được ánh xạ lên mô hình RTL của MUSRA và được mô phỏng qua công cụ EDA Modelsim từ hãng Mentor Graphic. Kết quả mô phỏng của khối ước lượng chuyển động (a) và bộ lọc FIR (b) sử dụng mô hình RTL của MUSRA được chỉ ra trong Hình 3- 6. Việc ánh xạ các vòng lặp lên MUSRA là khá đơn giản và linh hoạt. Các chức năng của MUSRA có thể được thay đổi dễ dàng theo từng vòng lặp bằng việc nạp thông tin cấu hình tương ứng vào phần cứng tái cấu hình của MUSRA.



(a)



(b)

Hình 3- 6. Kết quả mô phỏng của khối ước lượng chuyển động (a) và bộ lọc FIR (b) sử dụng mô hình RTL của MUSRA.

Để đánh giá và so sánh hiệu năng của MUSRA, các ví dụ benchmark cũng được ánh xạ lên các nền tảng tính toán khác nhau. Kết quả có trong Bảng 3- 2 dưới đây. Bảng 3- 2 tổng hợp thời gian thực thi (tính theo số chu kì) trên các nền tảng phần cứng khác nhau, gồm: MUSRA, TI C6678 DSP, REMUS, và bộ xử lý LEON. Trong đó, LEON là bộ xử lý đa chức năng 32-bit, REMUS là một cấu trúc tái cấu hình lõi thô dựa trên mảng 8×8 tế bào tái cấu hình RC, TI C6678DSP là một bộ xử lý tín hiệu số 32-bit được sử dựng trong các ứng dụng cần xử lý tín hiệu hiệu năng cao. Mô phỏng được thực hiện với chuỗi dữ liệu đầu vào có chiều dài 1024, 2048, 4096 byte. Để đảm bảo công bằng, tất các các nền tảng đánh giá sử dụng bus hệ thống 32-bit.

Bảng 3- 2 Thời gian thực thi các vòng lặp kernel trên các nền tảng tính toán khác nhau.

Vòng lặp Kernel	**Độ dài chuỗi dữ liệu (bytes)**	**Bộ xử lý LEON [14]**	**TI C6678 DSP** (non-optimized code)	**TI C6678 DSP** (Optimized code)	**REMUS [5] CGRA**	**MUSRA CGRA**
(SAD)	1.024	10.125	NA	NA	2.027	**260**
	2.048	20.370	NA	NA	4.075	**516**
	4.096	40.860	NA	NA	8.171	**1.028**
Độ dài cửa sổ trượt (Window length N=8)	1.024	29.148	24.117	NA	3.074	**1.026**
	2.048	58.142	48.181	NA	6.146	**2.050**
	4.096	116.128	96.301	NA	12.290	**4.098**

Nhân vô hướng 2 véc-tơ có độ dài-4	1.024	64 622	30.827	555	386	386
	2.048	128.864	61.827	1.067	770	770
	4.096	257.344	122.987	2.083	1.538	1.538
8-tap FIR	1.024	407.765	NA	7.216	3.075	**1.032**
	2.048	816.384	NA	14.384	6.147	**2.056**
	4.096	1.641.778	NA	29.736	12.291	**4.104**

Như chỉ ra trong Bảng 3- 2, hiệu năng thực thi của các vòng lặp trên bộ xử lý LEON và DSP là tương đối thấp bởi vì phương pháp thực thi kiểu tuần tự. Phân tích sự thực thi của các vòng lặp trên bộ xử lý LEON ở mức lệnh hợp ngữ cho thấy: tại mỗi lần lặp, bộ xử lý phải dùng một lượng lớn các chu trình lệnh để điều khiển vòng lặp. Ngược lại, các cấu trúc CGRA như REMUS và MUSRA có thể cải thiện đáng kể hiệu năng của các vòng lặp vì chúng có khả năng thực hiện nhiều phép tính và nhiều lần lặp song song. So với mô hình REMUS, mô hình đề xuất trong luận văn này có hiệu năng tốt hơn vì mô hình này khai thác dữ liệ u cục bộ dựa theo các vòng lặp liên tiếp, do đó nó không chỉ giảm băng thông dữ liệu đầu vào mà còn khai thác hiệu quả cơ chế xử lý song song nhiều mức trong mỗi vòng lặp. Như thấy trong Bảng 3- 2, bằng việc tái sử dụng các dữ liệu lặp giữa hai lần lặp liền kề nhau trong các thuật toán SAD, tính tổng chuyển động, vòng lặp FIR, mô hình đề xuất cho phép giảm đáng kể số lần truy nhập bộ nhớ khi chiều dài chuỗi dữ liệu đầu vào lớn, do đó có khả năng giảm đáng kể băng thông truy xuất bộ nhớ. Hơn nữa, vì chỉ tối đa 4 byte dữ liệu cần phải đọc trong mỗi chu kì nên làm giảm đáng kể băng thông truy cập bộ nhớ được đòi hỏi để nạp dữ liệu từ bộ nhớ bên ngoài vào bộ nhớ trong của MUSRA. Như vậy là, bằng cách đảm bảo luồng dữ liệu cấp cho RCA liên tục mô hình đề xuất đạt được hiệu năng cao hơn do sử dụng 100% tài nguyên đường ống cho tính toán.

KẾT LUẬN

Trong thời gian tìm hiểu và nghiên cứu dưới sự giúp đỡ tận tình của thầy hướng dẫn TS.Nguyễn Kiêm Hùng, đến nay toàn bộ nội dung của luận văn đã được hoàn thành đáp ứng đầy đủ các yêu cầu đã đặt ra. Qua quá trình tìm hiểu thực hiện đề tài, tôi đã thu được những kết quả chính như sau:

- Tìm hiểu xu hướng nghiên cứu CGRA trên thế giới,
- Thiết kế đã được mô hình hóa bằng ngôn ngữ VHDL (trong đó RCA của MUSRA được thiết kế dưới dạng RTL) và tiến hành mô phỏng, so sánh với các phương thức thực hiện khác. Các kết quả thực nghiệm chỉ ra rằng thiết kế đáp ứng được yêu cầu cơ bản đặt ra ban đầu: như tăng tốc độ tính toán cho các vòng lặp; khả năng tái hình linh hoạt các vòng lặp khác nhau có thể sử dụng cho một số phép toán thường dùng trong xử lý đa phương tiện trong truyền thông
- Các module được tham số hóa, dễ dàng mở rộng thiết kế theo các phương án kết nối khác nhau, trong đó lõi RCA của MUSRA được thiết kế với khả năng có thể mở rộng kích thước theo cả 2 chiều.

Tuy nhiên, vẫn còn một số hạn chế như:

- Các phép toán mô tả trong DATAPATH còn chưa được tối ưu.
- Chưa thể thực hiện được mô phỏng kiểm tra toàn diện các khối.
- Các bit cấu hình lựa chọn chế độ 16/8 bit đã được bỏ quả.

Kết quả thu được của luận văn hoàn thành ở mức xây dựng một cấu trúc MUSRA hoàn chỉnh, trong đó phần lõi RCA được thiết kế ở mức RTL.

TÀI LIỆU THAM KHẢO

[1] Christophe Bobda, "Introduction to Reconfigurable Computing – Architectures, Algorithms, and Applications", Springer, 2007. doi: 10.1007/978-1-4020-6100-4.

[2] DeHon, A. (2015). Fundamental underpinnings of reconfigurable computing architectures. Proceedings of the IEEE, 103(3), 355-378.

[3] A. Shoa and S. Shirani, "Run-Time Reconfigurable Systems for Digital Signal Processing Applications: A Survey", Journal of VLSI Signal Processing, Vol. 39, pp.213–235, 2005, Springer Science.

[4] G. Theodoridis, D. Soudris and S. Vassiliadis, "A Survey of Coarse-Grain Reconfigurable Architectures and Cad Tools Basic Definitions, Critical Design Issues and Existing Coarse-grain Reconfigurable Systems", Springer, 2008.

[5] X. N. LIU, C. MEI, P. CAO, M. ZHU, and L. X. SHI: "Date Flow Optimization of Dynamically Coarse Grain Reconfigurable Architecture for Multimedia Applications", IEICE Trans. on Information and Systems, Vol. E95-D, No. 2, pp. 374-382, 2013.

[6] Frank Bouwens, Mladen Berekovic, Bjorn De Sutter, and Georgi Gaydadjiev: "Architecture Enhancements for the ADRES Coarse-Grained Reconfigurable Array" HiPEAC 2008, LNCS 4917, pp. 66–81, 2008.

[7] X. Technologies, "XPP-III Processor Overview", White Paper, July 13 2006.

[8] João M. P. Cardoso, Pedro C. Diniz: "Compilation Techniques for Reconfigurable Architectures", Springer, 2009.

[9] http://www.xilinx.com/products/silicon-devices/soc/zynq-7000.htm.

[10] Altera (2014): Intel® Arria® 10 SoC FPGA devices.

[11] Hung K. Nguyen, Quang-Vinh Tran, and Xuan-Tu Tran, "Data Locality Exploitation for Coarse-grained Reconfigurable Architecture in a Reconfigurable Network-on-Chip", The 2014 International Conference on Integrated Circuits, Design, and Verification (ICDV 2014).

[12] Kathryn S. McKinley, Steve Carr, Chau-Wen Tseng, "Improving Data Locality with Loop Transformations", ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 18, Issue 4, July 1996, pp. 424 - 453.

[13] Meher, Pramod Kumar, and Thanos Stouraitis. Arithmetic Circuits for DSP Applications. John Wiley & Sons, 2017.

[14] Gaisler Research, "GRLIB IP Core User's Manual", Version 1.3.0-b4133, August 2013.

[15] Indrayani Patle, Akansha Bhargav, Prashant Wanjari, "Implementation of Baugh-Wooley Multiplier Based on Soft-Core Processor", IOSR Journal of Engineering (IOSRJEN) e-ISSN: 2250-3021, p-ISSN: 2278-8719 Vol. 3, Issue 10 (October. 2013), ||V3|| PP 01-07