REPORT

Thiết kế và thực hiện khối tính nhân chập 2-D bằng phương pháp HLS

Ver 1.0

08/11/2024

|  | Họ và tên (Full name) | Mã SV (ID) | Đóng góp (Contribution) |
| --- | --- | --- | --- |
| Thành viên 1 (Member 1) | Quách Đắc Chính | 22022518 | Xây dựng, code cho bài toán tính nhân chập 2-D, viết báo cáo |
| Thành viên 2  (Member 2) | Hoàng Ngọc Hào | 22022668 | Tối ưu bài toán, viết báo cáo |
| Tên/Địa chỉ Repo trên Github | <https://github.com/ChinhQuach303/CONV2D_Project.git> | | |

| Abstract |
| --- |
| Bài báo cáo với chủ đề xây dựng và thiết kế khối nhân chập 2-D qua Vivado HLS sử dụng ngôn ngữ C của nhóm. Sau đó tối ưu làm giảm độ trễ và tối ưu hiệu năng  bằng các phương pháp đã được học.Cuối cùng là thực hiện triển khai phần cứng trên ZynQ7 |

| Keywords |
| --- |
| Vivado HLS, 2-D Convolution |

| Hướng dẫn (Guide) |
| --- |
| Sinh viên điền vào báo cáo theo mẫu đính kèm. Sinh viên điền các mục:   * Thông tin sinh viên, mã số sinh viên * Mục *Đóng Góp* điền các công việc đã làm tương ứng của từng sinh viên. * Tên/Địa chỉ Repo trên Github hoặc Google Drive   Ngoại trừ phần thông tin sinh viên, mã số sinh viên và tên/địa chỉ Repo trên Github hoặc Google Drive ở đầu, sinh viên cần hoàn thành các phần nội dung (theo các mục đã được gợi ý – nhưng không hạn chế) trong phần báo cáo để mô tả các công việc nhóm đã thực hiện và các kết quả đã đạt được.  Sinh viên làm theo nhóm 2 người, chỉ cần 1 sinh viên đại diện nộp 1 bản báo cáo, sửa tên file thành tên của các thành viên trong nhóm (viết có dấu).  Sinh viên nộp lại báo cáo này trước khi tới trình bày kết quả, muộn nhất trước ngày thi hết môn một ngày. Ngày thi, SV cần mang máy tính laptop và sản phẩm để chạy demo!  Lưu ý: Nghiêm cấm mọi hình thức copy bài (bao gồm cả report và mã nguồn) của nhau. Nếu phát hiện sự giống nhau giữa 2 bài thì tùy mức độ mà có thể sẽ bị trừ điểm hoặc chia lấy điểm trung bình làm điểm của project. |

Document History

| Version | Time | Revised by | Description |
| --- | --- | --- | --- |
| V1.0 | 08/11/2024 | Nguyễn Kiêm Hùng | Original Version |
| V2.0 | 18/12/2024 | Quách Đắc Chính | Report version |
|  |  |  |  |

MỤC LỤC

[Document History 3](#_gjdgxs)

[MỤC LỤC 4](#_30j0zll)

[1. Giới thiệu 5](#_1fob9te)

[2. Yêu cầu 6](#_2et92p0)

[*2.1.*](#_tyjcwt) *Yêu cầu đối với thiết kế: 6*

[3. Thuật toán 6](#_3dy6vkm)

[4. Mô hình C và testbench 7](#_1t3h5sf)

[5. Tối ưu thiết kế để nâng cao hiệu năng 7](#_4d34og8)

[6. Mô phỏng/thực thi và đánh giá 7](#_2s8eyo1)

[7. Kết luận 7](#_17dp8vu)

[Appendix A: 8](#_3rdcrjn)

[Appendix B: VHDL Code 9](#_26in1rg)

[(đóng gói thành tệp nén và gửi kèm báo cáo) 9](#_lnxbz9)

[Appendix C: 10](#_35nkun2)

[List of Figures 11](#_1ksv4uv)

[List of Tables 12](#_44sinio)

[References 13](#_2jxsxqh)

# Giới thiệu

*(Introduction to the motivation, Objectives, and main Contents of the project)*

Mục tiêu: Vận dụng các kiến thức, kỹ năng đã được học để thiết kế, mô phỏng và thực thi một mô-đun phần cứng thực hiện tính tích chập J = 2DConV(I, K) giữa hình ảnh lối vào I với một ma trận kernel K ([1] ). Trong đó, mỗi pixel trong hình ảnh tích phân J đại diện cho tổng tích lũy của tích điểm-điểm giữa ma trận K với một ma trận cùng kích thước được trích xuất từ ma trận đầu vào I. Phép tính tích chập được mô tả bằng ví dụ sau.

*Ví dụ:* nếu hình ảnh đầu vào I là ma trận có kích thước 5×5 như sau:

I =

Và kernel có kích thước 3×3 như sau:

K =

thì kết quả tính toán của khối 2D-Convolution trả về hình ảnh J có kích thước 3×3 như sau:

C =

Quá trình tính toán được minh họa trong Hình 1.

A picture containing shape

Description automatically generated

Hình 1. Ví dụ minh họa phép tính tích chập 2D.

Lưu ý rằng pixel có tọa độ (r,c) – (hàng, cột) – tronh ảnh lối ra được tính bằng cách nhân chập điểm – điểm giữa ma trận kernel và ma trận 3×3 có tâm nằm ở vị trí có tọa độ (r + 1, c + 1) trong ảnh lối vào.

# Yêu cầu

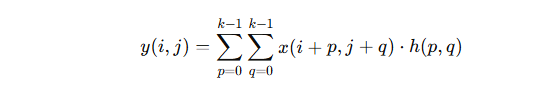
## Yêu cầu đối với thiết kế:

* Xây dựng thuật toán thực hiện tính nhân chập 2D.
* Xây dựng bản mô tả mức cao cho thuật toán bằng ngôn ngữ C,
* Xây dựng testbench để kiểm chứng thuật toán bằng C simulation
* Tổng hợp bản mô tả C thành bản mô tả RTL bằng VHDL/Verilog bằng Vivado HLS
* Mô phỏng C/RTL co-simulation để kiểm chứng và phân tích hoạt động của phần cứng sau tổng hợp HLS
* Tối ưu mô hình C và thêm vào các dẫn hướng để cải thiện hiệu năng của thiết kế
* Thực hiện thiết kế trên ZynQ-7000 để đánh giá lượng tài nguyên sử dụng, hiệu năng.

# Thuật toán

Thuật toán được sử dụng để cài đặt trong bài báo cáo nhằm tính khối nhân chập 2D với đầu vào là 1 ảnh và sử dụng ma trận kernel.

### Công thức nhân chập 2D:



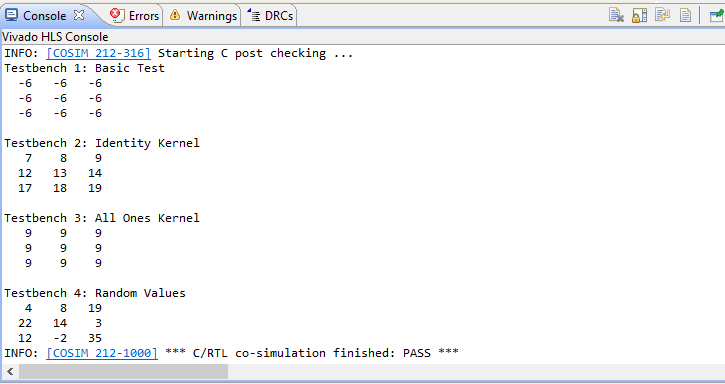
Trong đó:

* x là ma trận đầu vào (ảnh 2D) m x n
* h là ma trận kernel(nhân chập), kích thước k x k
* y là ma trận đầu ra sau khi áp dụng nhân chập n-k+1 x m-k+1

# Mô hình C và testbench

Mô hình C: Sau khi có đã hiểu về thuật toán, nhóm em áp dụng vào để cài đặt chương trình C. Với đầu vào và đầu ra đã nêu, thuật toán sử dụng 4 vòng lặp trong đó  
- 2 vòng lặp bên trên để duyệt các pixel của ma trận đầu ra  
- 2 vòng lặp bên dưới để thực hiện phép nhân ma trận kernel và vùng được duyệt bởi ma trận đầu vào

Testbench: Ở file testbench, sẽ kiểm tra các trường hợp để chắc chắn rằng code chạy thành công bao gồm 4 trường hợp gồm thông thường, ma trận kernel đơn vị, ma trận kernel các phần tử đều là 1 và cuối cùng là ma trận đầu vào ngẫu nhiên.



*Hình 2. Mô tả kết quả chạy thành công file testbench*

# Tối ưu thiết kế để nâng cao hiệu năng

Sau khi có kết quả, nhận thấy code hiện tại có thể cải thiện bằng cách tối ưu truy cập bộ nhớ (sử dụng caching hoặc partitioning), song song hóa các vòng lặp (loop unrolling/pipelining), và tận dụng phần cứng (dùng SIMD hoặc các directive như #pragma HLS).  
Với ý tưởng trên thì nhóm đã triển khai như sau:

#pragma HLS ARRAY\_PARTITION:

Chia nhỏ các mảng đầu vào, kernel và đầu ra với dim=2.

* + Khi một mảng lớn được truy cập tuần tự, sẽ có độ trễ do truy cập bộ nhớ ngoài (DRAM).
  + Việc chia nhỏ mảng sẽ lưu trữ từng phần của mảng trong các thanh ghi nội bộ hoặc BRAM, cho phép truy cập song song vào các phần tử. Điều này giúp giảm độ trễ truy cập bộ nhớ và tăng tốc độ xử lý.

**#**pragma HLS PIPELINE:

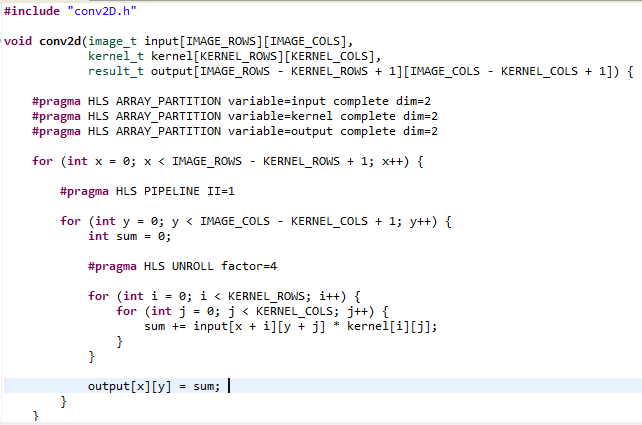
Áp dụng pipelining cho vòng lặp ngoài (vòng x).

* + Mỗi lần tính toán một pixel đầu ra sẽ cần thực hiện nhiều phép tính nhân và cộng qua các vòng lặp lồng nhau. Nếu không sử dụng pipelining, vòng lặp sẽ phải đợi toàn bộ các phép tính của vòng trước hoàn tất mới chuyển sang vòng kế tiếp.
  + Pipelining cho phép xử lý chồng lấn các phép tính (ở các giai đoạn khác nhau) trên phần cứng, đảm bảo rằng mỗi chu kỳ xung nhịp có thể bắt đầu tính toán một pixel mới. Điều này giúp tăng thông lượng (throughput).

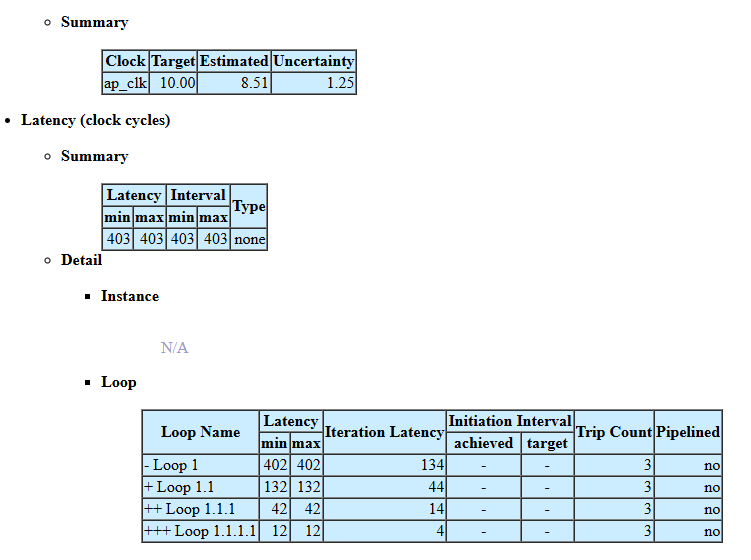
#pragma HLS UNROLL:

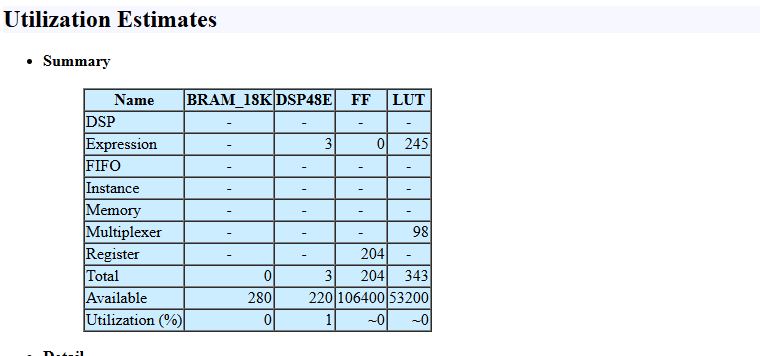
Giải unroll vòng lặp xử lý kernel (vòng i và j) với hệ số 4.

* + Thay vì xử lý từng giá trị của kernel tuần tự, việc unroll cho phép xử lý đồng thời nhiều giá trị trong một chu kỳ xung nhịp. Ví dụ, với hệ số unroll là 4, bốn phép tính nhân và cộng có thể được thực hiện song song.
  + Điều này đặc biệt hiệu quả khi kernel có kích thước nhỏ (ví dụ: 3x3 hoặc 5x5), giúp giảm đáng kể số chu kỳ cần thiết để tính tích chập.

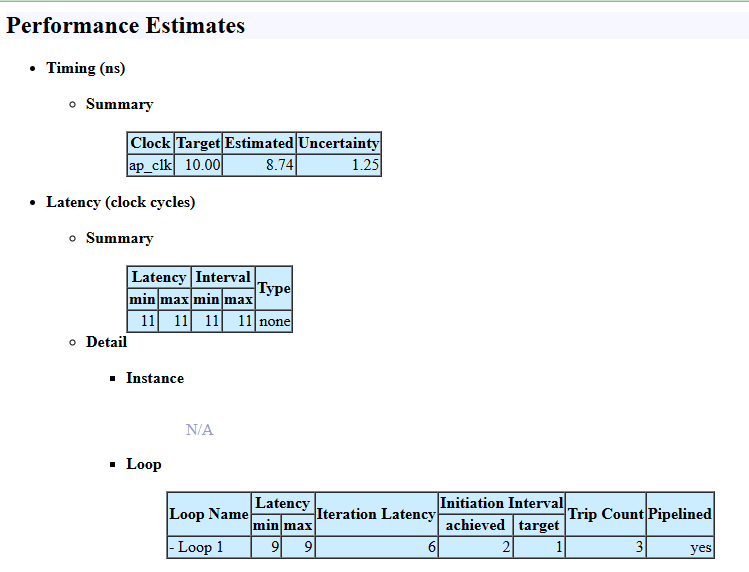


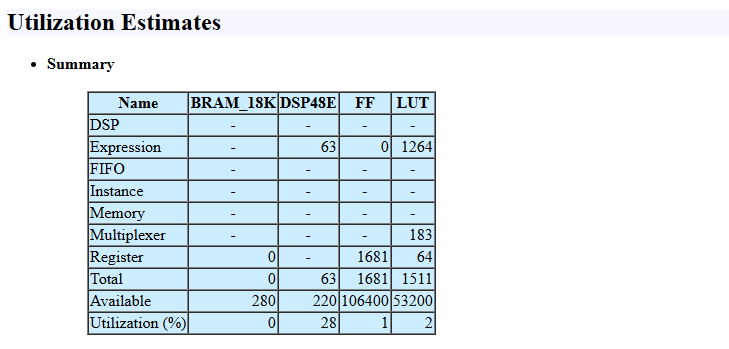
# Mô phỏng/thực thi và đánh giá





*Hình 3.1. Kết quả trước khi được sửa đổ*i





*Hình 3.2. Kết quả sau khi đã cải tiến*

Qua 2 kết quả thu được từ Hình 3.1 và Hình 3.2, với kết quả chạy hàm mô phỏng C( C/Simulation) thì đều đã chạy thành công các testbench.

Sau khi áp dụng các dẫn hướng đã giúp giảm độ trễ(latency) từ 403 xuống còn 11 chu kì xung nhịp. Tuy nhiên thì việc thêm dẫn hướng cũng làm cho việc sử dụng tài nguyên trở nên nhiều hơn:  
- Thanh ghi (Registers):

Tăng nhiều Phân chia mảng ARRAY\_PARRTION lưu trữ nhiều phần tử trong các thanh ghi nội bộ thay vì truy cập DRAM hoặc BRAM. Kết quả từ 204 lên đến 1681.

-DSP (Digital Signal Processing Units):

Sử dụng tăng cao: Các phép tính nhân trong vòng lặp unroll sẽ yêu cầu nhiều DSP hơn do thực hiện song song các phép tính. Kết quả tăng từ 3 lên 81.

-LUT (Look Up Table):

Tăng hơn: Tương tự như DSP tăng từ 245 lên 1264.

Nhìn chung, mặc dù việc sử dụng tài nguyên tăng cao, hiệu quả về mặt giảm độ trễ và tăng tốc độ tính toán khá tốt. Những thay đổi này mở ra tiềm năng ứng dụng cao cho các hệ thống đòi hỏi hiệu suất vượt trội, đặc biệt trong xử lý tín hiệu và hình ảnh thời gian thực.

# Kết quả triển khai phần cứng trên ZynQ

# Kết luận

Nhóm đã đưa ra được cách thiết kế cũng như tối ưu cho bài toán khối nhân chập 2D(2D-Convolution) sử dụng Vivado HLS. Từ kết quả thiết kế sau khi được hiệu chỉnh đã làm giảm độ trễ từ 403 xuống còn 11.

Việc tối ưu đã giúp tăng mức sử dụng tài nguyên tuy nhiên đó vẫn chưa thực sự tối ưu. Ở bài toán này thì kích thước đầu vào vẫn tương đối nhỏ nên đây là một kết quả tương đối ổn định. Nếu mở rộng bài toán với kích thước đầu vào lớn hơn thì cần xem xét kỹ hơn các phương pháp tối ưu khác để giúp cân bằng giữa hiệu năng và mức tiêu thụ tài nguyên.

Appendix A:

Appendix B: VHDL Code

(đóng gói thành tệp nén và gửi kèm báo cáo)

Appendix C:

Compress and email to hungnvnu@gmail.com

List of Figures

[Hình 1. Ví dụ minh họa quá trình sắp xếp một dãy gồm 4 phần tử theo trật tự tăng dần. 6](#_3znysh7)

[Hình 2. Giao diện ghép nối I/O của đơn vị Sorting Unit. 7](#_3j2qqm3)

[Hình 3: FSMD. 8](#_1y810tw)

[Hình 4: Datapath. 8](#_4i7ojhp)

[Hình 5: FSM of controller. 8](#_2xcytpi)

[Hình 5: Block diagram of whole sorting unit. 9](#_1ci93xb)

List of Tables

[Bảng 1: Mô tả các tín hiệu vào ra. 7](#_3whwml4)

References

1. https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1