**REPORT**

**Thiết kế và thực hiện bộ tăng tốc cho tính nhân chập 2-D bằng phương pháp HLS**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Họ và tên (Full name)** | **Mã SV (ID)** | **Đóng góp (Contribution)** |
| Thành viên 1 (Member 1) | obi | 22029076 |  |
| Thành viên 2  (Member 2) | Lê Trần Em CK | 22029041 |  |
| Tên/Địa chỉ Repo trên Github hoặc Google Drive | [matrixConv/MatConv.h at main · ltrnnhdng/matrixConv](https://github.com/ltrnnhdng/matrixConv) | | |

|  |
| --- |
| **Abstract (from 5 to 10 lines)** |
|  |

|  |
| --- |
| **Keywords** |
|  |

|  |
| --- |
| **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 | 11/04/2025 | Nguyễn Kiêm Hùng | Original Version |
| V2.0 | 24/5/2025 | MCK ft.Obito | Code |
| V3.0 | 1/6/2025 | MCK Prod.Phongkin | Chỉnh sửa phần 5, thêm phần 6,7  Thêm chú thích các hình |

Mục lục

[Document History 4](#_Toc199671860)

[1. Giới thiệu 6](#_Toc199671861)

[2. Yêu cầu 7](#_Toc199671862)

[2.1. Yêu cầu đối với thiết kế: 7](#_Toc199671863)

[3. Thuật toán. 8](#_Toc199671864)

[Kết quả synthesis của code ban đầu, không có dẫn hướng và ràng buộc: 8](#_Toc199671865)

[4. Mô hình C và testbench 10](#_Toc199671866)

[5. Tối ưu thiết kế để nâng cao hiệu năng 10](#_Toc199671867)

[a. Thực hiện partition các ma trận thành phần: 10](#_Toc199671868)

[b. Thực hiện pipeline 2 vòng lặp ngoài: 10](#_Toc199671869)

[c. Thực hiện unroll 2 vòng lặp trong: 11](#_Toc199671870)

[d. Kết quả 11](#_Toc199671871)

[e. So sánh unroll hoàn toàn và pipeline hoàn toàn cho các vòng lặp 11](#_Toc199671872)

[6. Mô phỏng/thực thi và đánh giá 12](#_Toc199671873)

[7. Kết luận 13](#_Toc199671874)

[List of Figures 17](#_Toc199671875)

# 1. 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 đểthiết 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:

Và kernel có kích thước 3×3 như sau:

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

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.

# 2. Yêu cầu.

## 2.1. 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

# 3. Thuật toán.

Thuật toán tích chập ma trận được triển khai trong file MatConv.cpp.Thuật toán được sử dụng là sử dụng 4 vòng lặp lồng nhau dựa trên kích thước của ma trận đầu vào và kernel:

* 2 vòng lặp ngoài duyệt qua các phần tử của ma trận đầu ra
  + 2 vòng lặp trong duyệt qua các phần tử của kernel cùng với các phần tử đầu vào tương ứng để tính tổng và ghi vào vị trí đang xét của ma trận đầu ra

Psudo-code:

for each element [i][j] in output matrix:

output[i][j] = 0;

for each element [x][y] in kernel matrix:

output[i][j] = kernal[x][y] \* input[i+x][j+y];

Các kích thước của ma trận được định nghĩa trong file MatConv.h:

* Kích thước kernel: KER\_ROWS = 5, KER\_COLS = 5.
* Kích thước đầu vào: INP\_ROWS = 15, INP\_COLS = 15.

Kích thước đầu ra được tính toán dựa trên kích thước đầu vào và kernel:

OUTP\_ROWS = INP\_ROWS - KER\_ROWS + 1 = 15 - 5 + 1 = 11.

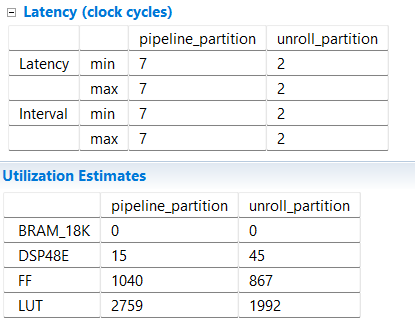
OUTP\_COLS = INP\_COLS - KER\_COLS + 1 = 15 - 5+ 1 = 11.

Kiểu dữ liệu được sử dụng:

* mat\_ker: char cho ma trận kernel.
* mat\_inp: char cho ma trận đầu vào.
* mat\_outp: short cho ma trận đầu ra.

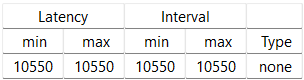
## Kết quả synthesis của code ban đầu, không có dẫn hướng và ràng buộc:

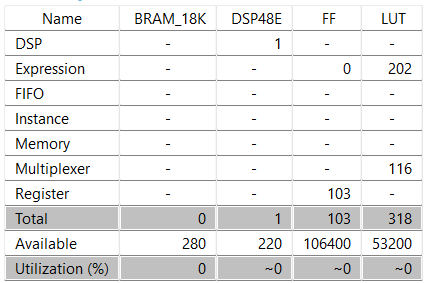
Khi sử dụng ma trận inp là 5x5 và ma trận ker là 3x3 thì kết quả cho thấy rằng unroll hết các vòng lặp sẽ tối ưu hơn pipeline hết các vòng lặp vì lí do do kích thước các ma trận nhỏ làm cho tổng lượng tài nguyên nhỏ nên đáp ứng tốt nhất khi unroll tất cả. ngược lại pipeline cố gắng đạt được II=1 thì lại tạo ra nhiều tài nguyên hơn mặc dù kích thước các ma trận đều nhỏ

****

Hình 2: Kết quả tổng hợp ban đầu với ma trận input và kernel lần lượt là 5x5 và 3x3

Để thấy rõ hơn ảnh hưởng của các dẫn hướng đối với lượng tài nguyên của Board, nhóm đã tăng kích thước của các ma trận thành: inp là 15x15 và ker là 5x5:

****

****

Hình 3: Kết quả tổng hợp ban đầu với ma trận input và kernel lần lượt 15x15 và 3x3.

# 4. Mô hình C và testbench

Code: [Github](https://github.com/ltrnnhdng/matrixConv/blob/main/src/MatConv_tb.cpp).

- Khởi tạo dữ liệu: Sinh ngẫu nhiên ma trận kernel và input. Seed ngẫu nhiên cố định (123456) đảm bảo khả năng tái tạo.

- Tính toán tham chiếu: Tính toán kết quả tích chập chính xác bằng phần mềm C++ (sw\_res).

- Gọi HLS Function: Nếu HW\_COSIM được định nghĩa (trong MatConv.h), gọi hàm MatConv (mô phỏng phần cứng) để nhận kết quả (hw\_res).

- So sánh và báo cáo: So sánh hw\_res với sw\_res từng phần tử một. Mọi sự không khớp được đánh dấu \* và tăng biến đếm lỗi (err\_cnt). Cuối cùng, báo cáo tổng số lỗi hoặc thông báo thành công.

# 5. Tối ưu thiết kế để nâng cao hiệu năng

## a. Thực hiện partition các ma trận thành phần:

set\_directive\_array\_partition -type complete -dim 0 "MatConv" ker

set\_directive\_array\_partition -type complete -dim 0 "MatConv" inp

set\_directive\_array\_partition -type complete -dim 0 "MatConv" outp

- Việc partition các mảng ker, inp, và outp theo chiều 0 với kiểu complete giúp tách các phần tử trong mảng thành các phần tử rời rạc trong phần cứng.

- Mục đích là cho phép **truy cập song song tới từng phần tử** trong mảng. Nếu không có dẫn hướng này, HLS sẽ chỉ tạo một cổng đọc hoặc ghi cho toàn bộ mảng dẫn đến điểm thắt cổ chai, giảm tốc độ xử lý do giới hạn của tốc độ đọc và ghi.

- Dẫn hướng này được đặt luôn vào file MatConv.cpp để tối ưu cho các dẫn hướng sau

## b. Thực hiện pipeline 2 vòng lặp ngoài:

set\_directive\_pipeline "MatConv/Output\_Cols"

set\_directive\_pipeline "MatConv/Output\_Rows"

- Áp dụng cho vòng lặp Output\_Cols (và/hoặc Output\_Rows). Mục tiêu là đạt II tối ưu nhất để tạo ra một pixel đầu ra mỗi chu kỳ clock.

- Khi pipeline vòng lặp Output\_Cols, cho phép hệ bắt đầu thực hiện vòng lặp tiếp mà không cần đợi vòng lặp hiện tại hoàn tất.

- Lý do sử dụng pipeline ở 2 vòng lặp ngoài vì đối với các vòng lặp lớn thì pipeline cho thông lượng tốt (tức là tạo ra nhiều kết quả đầu ra mỗi chu kỳ).

## c. Thực hiện unroll 2 vòng lặp trong:

set\_directive\_unroll "MatConv/Ker\_Cols"

set\_directive\_unroll "MatConv/Ker\_Rows"

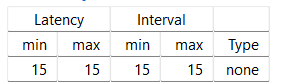
- Áp dụng cho các vòng lặp **Ker\_Rows** và **Ker\_Cols**. Vì kernel là 5x5, việc unroll hoàn toàn sẽ tạo ra 25 bộ nhân và bộ cộng hoạt động song song để tính toán một điểm đầu ra.

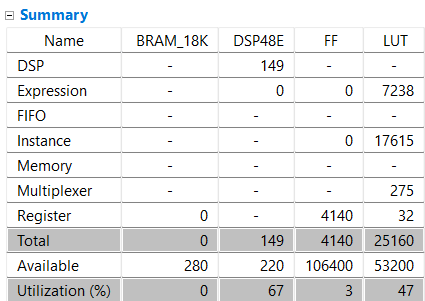
- Lý do sử dụng unroll ở 2 vòng lặp trong vì đối với vòng lặp nhỏ và cố định như kernel thì unroll mang lại hiệu quả tốt về độ trễ.

# 6. Mô phỏng/thực thi và đánh giá.

## a. Kết quả tổng hợp.

- Đối với ma trận kernel 5x5 và đầu vào 15x15 thì kết quả tổng hợp là:





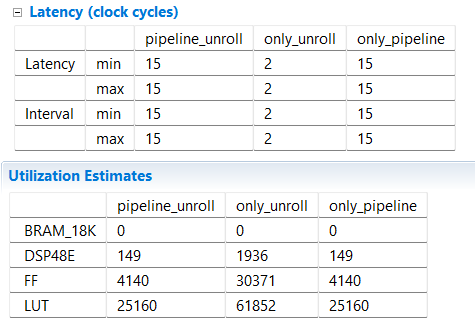
Hình 4: Kết quả tổng hợp khi sử dụng tối ưu của nhóm.

- Có thể thấy sự đánh đổi giữa độ trễ với tài nguyên phần cứng. Khi áp dụng các phương pháp chia mảng và unroll, pipeline, độ trễ đạt được là 15 (giảm xấp xỉ 700 lần so với ban đầu). Tuy nhiên, phần cứng chiếm dụng cũng tăng rất nhiều lần, đặc biệt với DSP để xử lý phép nhân, cộng song song.

## b. So sánh unroll hoàn toàn và pipeline hoàn toàn cho các vòng lặp.

Khi tăng kích thước ma trận lên, ta thấy rõ ràng trong việc pipeline hoàn toàn các vòng lặp với kích thước ma trận lớn sẽ cải thiện tài nguyên hơn rõ rệt so với unroll hoàn toàn các vòng lặp.

Kết quả cho thấy unroll hoàn toàn các vòng lặp sẽ đạt được độ trễ tốt nhất nhưng sẻ vượt quá lượng tài nguyên có sẵn của FPGA và pipeline đạt được độ trễ phù hợp và tài nguyên tối ưu tốt. tương đương với phương pháp mà nhóm đề xuất.

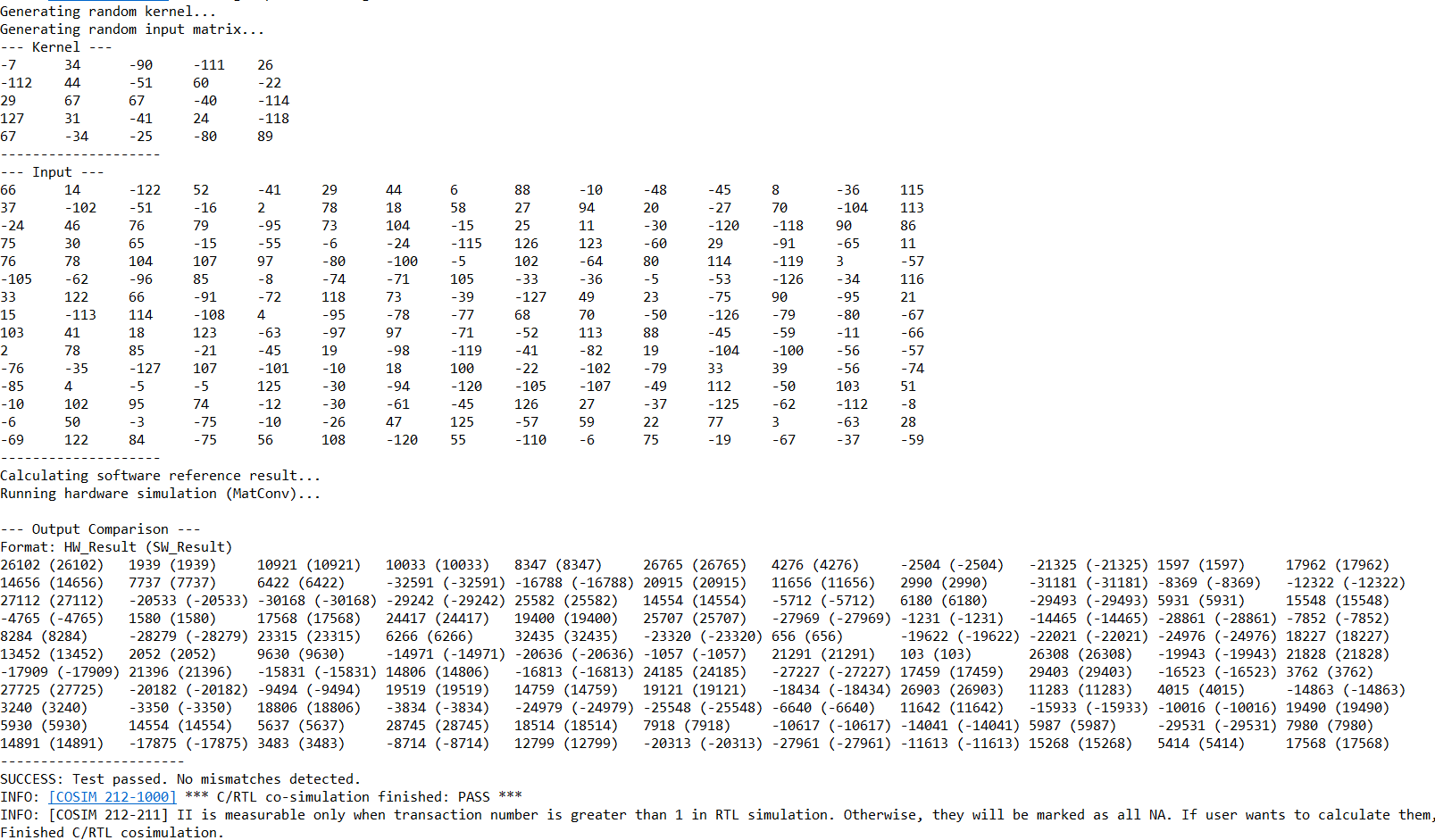


Hình 5: So sánh 3 cách tối ưu thiết kế.

## c. Thực thi.

Để kiểm chứng hoạt động chính xác của phần cứng sau khi tổng hợp bằng Vivado HLS, nhóm đã tiến hành mô phỏng đồng thời C/RTL (C/RTL Co-simulation). Quá trình này được thực hiện thông qua tệp MatConv\_tb.cpp với các bước chi tiết như sau:

* **Khởi tạo dữ liệu:** Testbench tạo ngẫu nhiên các giá trị cho ma trận kernel và ma trận đầu vào. Để đảm bảo khả năng tái tạo kết quả mô phỏng, một seed ngẫu nhiên cố định (123456) đã được sử dụng.
* **Tính toán tham chiếu:** Kết quả tích chập chính xác được tính toán bằng phần mềm C++ với hàm (sw\_res) để làm cơ sở so sánh.
* **Gọi hàm HLS:** Hàm MatConv (đã được tổng hợp bằng HLS) được gọi để nhận kết quả từ mô phỏng phần cứng (hw\_res).
* **So sánh và báo cáo:** Testbench thực hiện so sánh từng phần tử giữa hw\_res và sw\_res. Bất kỳ sự không khớp nào đều được đánh dấu bằng dấu '\*' và biến đếm lỗi (err\_cnt) sẽ được tăng lên. Cuối cùng, testbench sẽ báo cáo tổng số lỗi hoặc thông báo thành công nếu không có lỗi nào được phát hiện.



Hình 6: Kết quả đồng mô phỏng C/RTL [Github](https://github.com/ltrnnhdng/matrixConv)

Kết quả cho thấy thiết kế phần cứng hoạt động chính xác và đáp ứng đúng yêu cầu chức năng so với mô hình C.

# 7. Kết luận

Thiết kế đã được tối ưu với các dẫn hướng và ràng buộc do nhóm đề xuất:

* Phân vùng mảng: các ma trận Các ma trận ker, inp, và outp đã được phân vùng hoàn toàn theo chiều 0. Điều này cho phép truy cập song song tới từng phần tử trong mảng, loại bỏ các điểm thắt cổ chai do giới hạn tốc độ đọc/ghi và cải thiện đáng kể khả năng xử lý đồng thời.
* Pipeline các vòng lặp ngoài (set\_directive\_pipeline): Các vòng lặp ngoài (Output\_Rows, Output\_Cols) đã được pipeline. Mục tiêu là tăng thông lượng, cho phép hệ thống bắt đầu thực hiện vòng lặp tiếp theo mà không cần đợi vòng lặp hiện tại hoàn tất, từ đó đạt được Interval (II) tối ưu cho việc tạo ra một pixel đầu ra mỗi chu kỳ xung nhịp. Phương pháp này đặc biệt hiệu quả với các vòng lặp lớn, mang lại thông lượng cao.
* Unroll các vòng lặp trong: Hai vòng lặp trong (Ker\_Rows, Ker\_Cols) đã được unroll hoàn toàn. Với kernel kích thước 5×5, việc unroll tạo ra 25 bộ nhân và bộ cộng hoạt động song song để tính toán một điểm đầu ra. Phương pháp này mang lại hiệu quả vượt trội về độ trễ, đặc biệt phù hợp với các vòng lặp nhỏ và có số lần lặp cố định như vòng lặp kernel.

Kết quả tổng hợp cho thấy sự cải thiện rất lớn trong thiết kế: giảm từ 10550 chu kỳ xung nhịp của thiết kế ban đầu xuống còn 15 chu kỳ (giảm xấp xỉ 700 lần) với sự đành đổi của tài nguyên hệ thống sử dụng 149 DSP, 4140 FF và 25160 LUT.

So sánh các phương pháp tối ưu, nhóm nhận thấy:

* Giải pháp "*only\_unroll*" đạt độ trễ tốt nhất (2 chu kỳ) nhưng đòi hỏi lượng tài nguyên rất lớn (1936 DSP48E, 30371 FF, 61852 LUT), dẫn đến quá tải tài nguyên.
* Giải pháp "*only\_pipeline*" và giải pháp kết hợp "pipeline\_unroll" có độ trễ cao hơn (15 chu kỳ) nhưng tối ưu hơn nhiều về tài nguyên (149 DSP48E, 4140 FF, 25160 LUT).

Từ đó, có thể kết luận rằng phương pháp kết hợp giữa pipeline các vòng lặp ngoài và unroll các vòng lặp trong, cùng với phân vùng mảng, đã mang lại sự cân bằng giữa hiệu năng (độ trễ thấp) và chi phí tài nguyên cho nhân chập ma trận. Quá trình đồng mô phỏng C/RTL đã xác nhận tính đúng đắn của thiết kế, đảm bảo rằng phần cứng hoạt động chính xác theo yêu cầu.

# List of Figures

[Hình 1:Ví dụ minh họa phép tính tích chập 2D. 7](#_Toc199674411)

[Hình 2: Kết quả tổng hợp ban đầu với ma trận input và kernel lần lượt là 5x5 và 3x3 9](#_Toc199674412)

[Hình 3: Kết quả tổng hợp ban đầu với ma trận input và kernel lần lượt 15x15 và 3x3. 9](#_Toc199674413)

[Hình 4: Kết quả tổng hợp khi sử dụng tối ưu của nhóm. 11](#_Toc199674414)

[Hình 5: So sánh 3 cách tối ưu thiết kế. 12](#_Toc199674415)

[Hình 6: Kết quả đồng mô phỏng C/RTL Github 13](#_Toc199674416)

References

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