

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ - ĐHQGHN

VIỆN TRÍ TUỆ NHÂN TẠO



Thiết kế và thực hiện khối 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)	Vũ Minh Đức	22022587	Xây dựng thuật toán. Xây dựng bản mô tả mức cao cho thuật toán. Xây dựng testbench để kiểm chứng thuật toán.
Thành viên 2 (Member 2)	Đào Duy Hưng	22022589	Tổng hợp bản mô tả C thành bản mô tả RTL. Mô phỏng C/RTL co-simulation để kiểm chứng và phân tích hoạt động của phần cứng. Tối ưu mô hình C, cải thiện hiệu năng của thiết kế Thực hiện thiết kế trên ZynQ-7000.
Github	hardware-design-for-deep-learning		

Lời mở đầu

Báo cáo này trình bày quá trình thiết kế, tối ưu hóa và triển khai thuật toán tính toán tích chập 2D trên nền tảng FPGA ZynQ-7000, nhằm đáp ứng các yêu cầu về hiệu suất và tài nguyên cho các ứng dụng thời gian thực, đặc biệt trong lĩnh vực xử lý ảnh và tín hiệu. Thuật toán tích chập 2D là một trong những phép toán quan trọng trong nhiều ứng dụng, bao gồm nhận dạng hình ảnh, phân tích tín hiệu, và các hệ thống xử lý dữ liệu thời gian thực. Để tối ưu hóa thuật toán cho phần cứng, thuật toán ban đầu được phát triển bằng ngôn ngữ C, sau đó kiểm chứng và tối ưu hóa thông qua việc sử dụng các kỹ thuật cao cấp trong Vivado HLS, như pipelining, unrolling và phân vùng mảng (array partitioning), nhằm giảm độ trễ và tăng tốc độ xử lý.

Quá trình phát triển bao gồm việc sử dụng testbench để kiểm tra tính chính xác của thuật toán C, sau đó mô hình hóa và tổng hợp mã C thành mô hình RTL (Register Transfer Level) bằng Vivado HLS. Tiếp theo, mô hình RTL đã được đồng mô phỏng (C/RTL co-simulation) để đảm bảo tính chính xác và hiệu suất của thiết kế. Cuối cùng, thiết kế được triển khai trên FPGA ZynQ-7000 để đánh giá hiệu năng thực tế, bao gồm mức độ sử dụng tài nguyên phần cứng như LUTs, flip-flops, DSPs và BRAM, cũng như tốc độ xử lý và độ trễ của thuật toán.

Kết quả triển khai trên ZynQ-7000 cho thấy thiết kế đạt được sự cải thiện đáng kể về tốc độ xử lý, độ trễ thấp và sử dụng tài nguyên hiệu quả, đáp ứng các yêu cầu của các ứng dụng yêu cầu tính toán thời gian thực, đặc biệt là trong lĩnh vực xử lý ảnh và nhận dạng hình ảnh. Các kỹ thuật tối ưu hóa phần cứng đã giúp tăng cường khả năng xử lý song song, đồng thời tối thiểu hóa việc sử dụng tài nguyên mà không ảnh hưởng đến hiệu suất.

Keywords

Tích chập 2D, FPGA, ZynQ-7000, Vivado HLS, Pipelining, Song song hóa, Tăng tốc phần cứng, Xử lý ảnh

MỤC LỤC

1. Giới thiệu	4
2. Yêu cầu	5
3. Thuật toán	6
4. Mô hình C và testbench	6
5. Tối ưu thiết kế để nâng cao hiệu năng	8
6. Mô phỏng/thực thi và đánh giá	10
7. Kết quả triển khai phần cứng trên ZynQ	11
8. Kết luận	12

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 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:

$$I = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

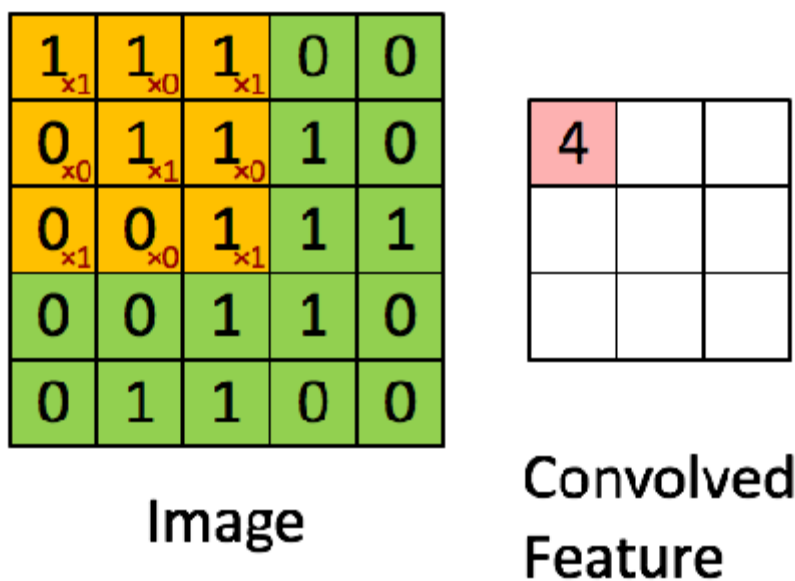
Và kernel có kích thước 3×3 như sau:

$$K = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

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

$$C = \begin{pmatrix} 4 & 3 & 4 \\ 2 & 4 & 3 \\ 2 & 3 & 4 \end{pmatrix}$$

Quá trình tính toán được minh họa trong Hình 1.



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

Lưu ý rằng pixel có tọa độ (r, c) – (hàng, cột) – trong 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

Xây dựng thuật toán thực hiện tính nhân chập 2D: Thuật toán nhân chập 2D được xây dựng để xử lý hình ảnh hoặc tín hiệu dưới dạng ma trận, áp dụng bộ lọc (kernel) lên ma trận đầu vào (input) và tính toán ma trận đầu ra (output).

Xây dựng bản mô tả mức cao cho thuật toán bằng ngôn ngữ C: Mã C được viết để mô phỏng thuật toán nhân chập 2D với các cấu trúc dữ liệu là mảng 2D. Mô hình này có thể được chuyển đổi sang mô hình phần cứng với Vivado HLS.

Xây dựng testbench để kiểm chứng thuật toán bằng C simulation: Testbench sẽ mô phỏng các ma trận đầu vào và kiểm tra kết quả từ hàm `conv2D` để đảm bảo tính đúng đắn của thuật toán.

Tổng hợp bản mô tả C thành bản mô tả RTL bằng VHDL/Verilog bằng Vivado HLS: Sử dụng Vivado HLS để chuyển đổi mô tả C sang mô hình RTL, cho phép triển khai trên phần cứng FPGA.

Mô phỏng C/RTL co-simulation để kiểm chứng và phân tích hoạt động của phần cứng sau tổng hợp HLS: Tiến hành mô phỏng đồng thời giữa mô hình C và RTL để kiểm tra tính chính xác của phần cứng sau khi tổng hợp và tối ưu.

Tối ưu mô hình C và thêm vào các dẫn hướng để cải thiện hiệu năng của thiết kế: Sử dụng các chỉ thị HLS để tối ưu hóa hiệu suất của mã C, bao gồm các chỉ thị như **PIPELINE**, **UNROLL** và **ARRAY_PARTITION** nhằm tối ưu hóa phần cứng.

Thực hiện thiết kế trên ZynQ-7000 để đánh giá lượng tài nguyên sử dụng, hiệu năng: Triển khai thiết kế lên ZynQ-7000 và đánh giá các chỉ số như mức độ sử dụng tài nguyên, độ trễ và hiệu suất xử lý.

3. Thuật toán

Thuật toán nhân chập 2D áp dụng trong báo cáo này có thể được mô tả như sau:

Mô tả thuật toán:

Thuật toán áp dụng một bộ lọc (kernel) có kích thước cố định ($KERNEL_SIZE \times KERNEL_SIZE$) vào một cửa sổ di động trên ma trận đầu vào. Với mỗi vị trí của cửa sổ, một phép nhân giữa phần tử của ma trận đầu vào và kernel được thực hiện, sau đó các giá trị được cộng lại để tạo ra phần tử tương ứng trong ma trận đầu ra.

Thuật toán này sử dụng các chỉ thị tối ưu hóa phần cứng, như **PIPELINE** và **UNROLL**, nhằm giảm thiểu độ trễ và tối đa hóa khả năng xử lý đồng thời.

Các chỉ thị tối ưu hóa:

#pragma HLS PIPELINE: Cải thiện hiệu suất bằng cách thiết lập pipeline cho vòng lặp, giúp thực hiện các phép tính đồng thời.

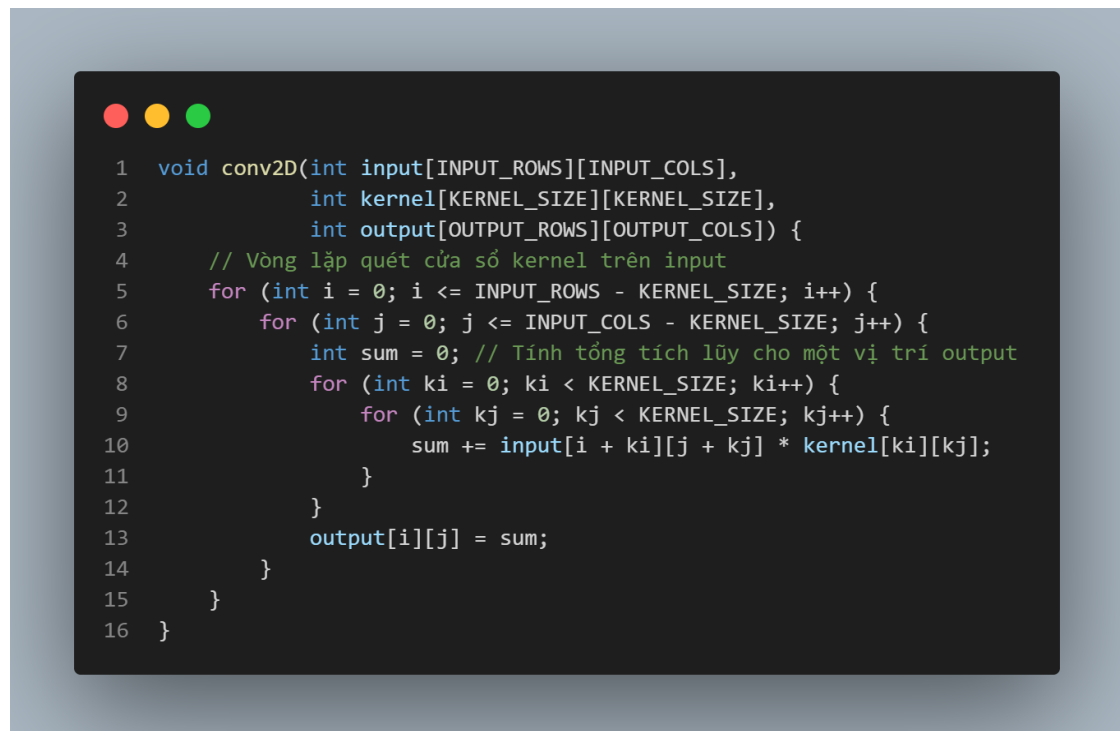
#pragma HLS UNROLL: Mở rộng vòng lặp để thực hiện nhiều phép toán song song, giảm thiểu thời gian xử lý.

#pragma HLS ARRAY_PARTITION: Phân mảng input và kernel thành các phần tử độc lập, giúp tối ưu hóa việc truy cập bộ nhớ và tăng hiệu suất.

4. Mô hình C và testbench

Mô hình C: Mô hình C được viết cho thuật toán nhân chập 2D và sử dụng các chỉ thị HLS để tối ưu hóa thiết kế phần cứng. Cấu trúc chính của thuật toán bao gồm ba vòng

lặp: hai vòng lặp để di chuyển cửa sổ kernel trên ma trận đầu vào và một vòng lặp bên trong để tính toán tổng tích lũy cho mỗi phần tử đầu ra.

A screenshot of a code editor with a dark background and light-colored text. The code is written in C++ and implements a 2D convolution function. It features three nested loops: an outer loop for rows, a middle loop for columns, and an inner loop for the kernel. The code calculates the sum of products for each output pixel. The function signature is void conv2D(int input[INPUT_ROWS][INPUT_COLS], int kernel[KERNEL_SIZE][KERNEL_SIZE], int output[OUTPUT_ROWS][OUTPUT_COLS]). The code is numbered from 1 to 16 on the left side of the editor.

```
1 void conv2D(int input[INPUT_ROWS][INPUT_COLS],  
2             int kernel[KERNEL_SIZE][KERNEL_SIZE],  
3             int output[OUTPUT_ROWS][OUTPUT_COLS]) {  
4     // Vòng lặp quét cửa sổ kernel trên input  
5     for (int i = 0; i <= INPUT_ROWS - KERNEL_SIZE; i++) {  
6         for (int j = 0; j <= INPUT_COLS - KERNEL_SIZE; j++) {  
7             int sum = 0; // Tính tổng tích lũy cho một vị trí output  
8             for (int ki = 0; ki < KERNEL_SIZE; ki++) {  
9                 for (int kj = 0; kj < KERNEL_SIZE; kj++) {  
10                    sum += input[i + ki][j + kj] * kernel[ki][kj];  
11                }  
12            }  
13            output[i][j] = sum;  
14        }  
15    }  
16 }
```

Hình 2. Minh họa hàm tính tích chập 2D

Testbench: Testbench được xây dựng để mô phỏng các ma trận đầu vào khác nhau và xác minh kết quả trả về từ hàm `conv2D`. Testbench cũng kiểm tra tính chính xác của việc thực hiện phép toán nhân chập trên các bộ dữ liệu mẫu.

```

1  #include <stdio.h>
2  #include "conv2D.h"
3
4  int main() {
5      // Khai báo ma trận đầu vào
6      int input[INPUT_ROWS][INPUT_COLS] = {
7          {1, 2, 3, 0, 1},
8          {4, 5, 6, 1, 0},
9          {7, 8, 9, 0, 1},
10         {1, 2, 1, 2, 3},
11         {0, 1, 0, 2, 1}
12     };
13
14     // Kernel tích chập (bộ lọc Sobel ngang)
15     int kernel[KERNEL_SIZE][KERNEL_SIZE] = {
16         {1, 0, -1},
17         {1, 0, -1},
18         {1, 0, -1}
19     };
20
21     // Kết quả output sau tích chập
22     int output[INPUT_ROWS - KERNEL_SIZE + 1][INPUT_COLS - KERNEL_SIZE + 1] = {0};
23
24     // In ma trận đầu vào
25     printf("Input Matrix:\n");
26     printMatrix(INPUT_ROWS, INPUT_COLS, input);
27
28     // Thực hiện tích chập
29     conv2D(input, kernel, output);
30
31     // In kết quả sau khi tính tích chập
32     printf("\nOutput Matrix after Convolution:\n");
33     printMatrix(INPUT_ROWS - KERNEL_SIZE + 1, INPUT_COLS - KERNEL_SIZE + 1, output);
34
35     return 0;
36 }

```

Hình 3. Minh họa testbench

5. Tối ưu thiết kế để nâng cao hiệu năng

Việc tối ưu hóa thiết kế tập trung vào nâng cao hiệu năng thông qua các kỹ thuật xử lý song song, tối ưu hóa vòng lặp, và quản lý bộ nhớ. Các kỹ thuật cụ thể được áp dụng trong thuật toán **2D Convolution** bao gồm:

Parallelism (Xử lý song song)

Thuật toán tận dụng xử lý song song để thực hiện đồng thời các phép toán nhân và cộng trong quá trình quét kernel trên ma trận đầu vào. Điều này giúp giảm thời gian xử lý và tăng hiệu suất thiết kế.

Pipelining (Đường ống hóa)

Pipelining được áp dụng cho các vòng lặp, cho phép các giai đoạn xử lý chồng chéo nhau. Điều này giúp tối đa hóa việc sử dụng tài nguyên phần cứng và giảm độ trễ tổng thể.

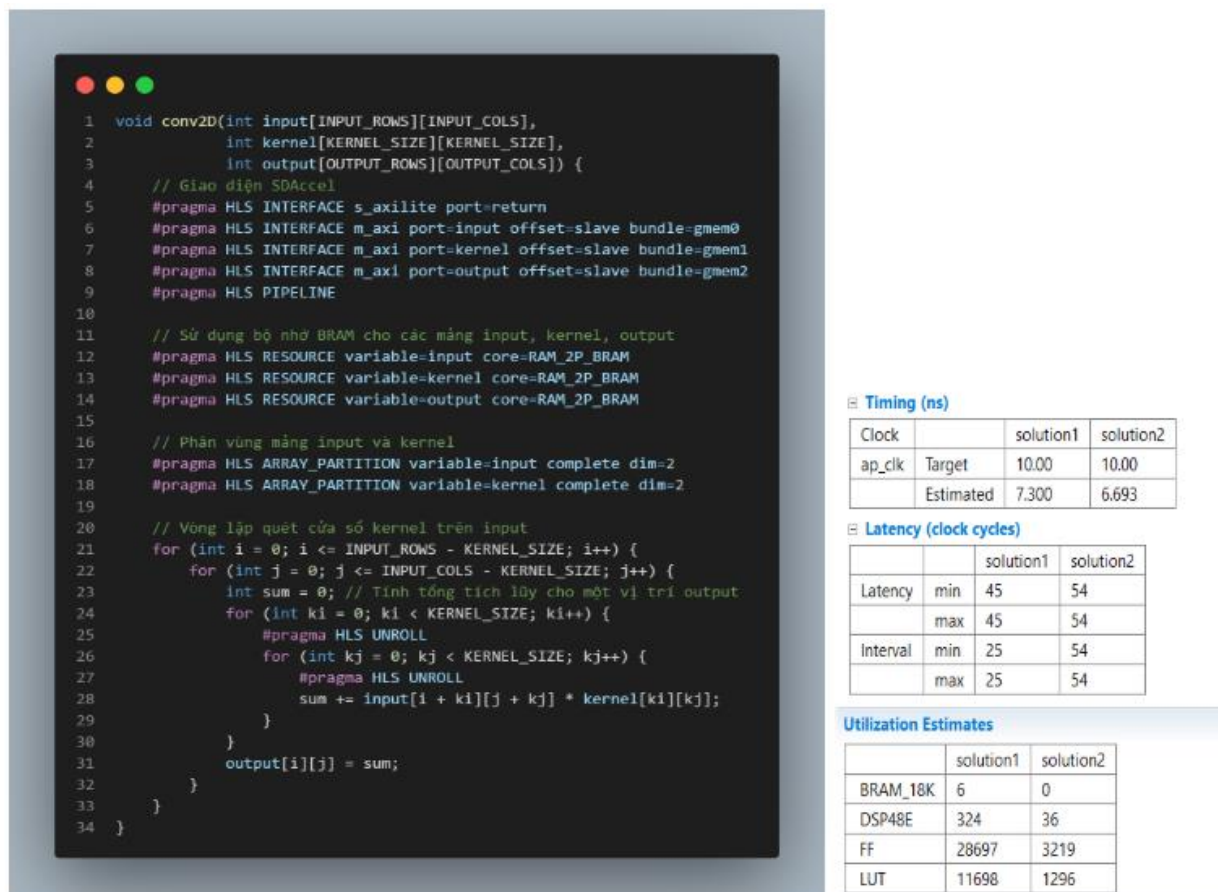
Unrolling (Mở rộng vòng lặp)

Vòng lặp bên trong, nơi thực hiện các phép toán giữa kernel và ma trận đầu vào, được mở rộng hoàn toàn (unrolled). Kỹ thuật này tăng khả năng xử lý song song và giảm số chu kỳ cần thiết.

Partitioning (Phân vùng mảng)

Các mảng dữ liệu (input, kernel, output) được phân vùng để tăng khả năng truy cập đồng thời và giảm tắc nghẽn bộ nhớ. Điều này cải thiện đáng kể hiệu suất truy cập dữ liệu.

Nhờ các kỹ thuật trên, thiết kế đạt được tốc độ xử lý cao hơn, sử dụng hiệu quả tài nguyên phần cứng và đảm bảo đáp ứng yêu cầu thời gian thực trong các ứng dụng phức tạp.



Hình 4. Hàm tính tích chập 2D được tối ưu thiết kế và sự chênh lệch về hiệu suất

6. Mô phỏng/thực thi và đánh giá

Mô phỏng C

Trước khi tổng hợp sang phần cứng, mã C được mô phỏng để kiểm tra tính đúng đắn của thuật toán. Các giá trị đầu ra được tính toán bởi thuật toán sẽ được so sánh với kết quả mong đợi để đảm bảo tính chính xác.

```
7 INFO: [APCC 202-3] Tmp directory is apcc_db
8 INFO: [APCC 202-1] APCC is done.
9   Generating csim.exe
10 Input Matrix:
11 1 2 3 0 1
12 4 5 6 1 0
13 7 8 9 0 1
14 1 2 1 2 3
15 0 1 0 2 1
16
17 Output Matrix after Convolution:
18 -6 14 16
19 -4 12 12
20 -2 7 5
21 INFO: [SIM 1] CSim done with 0 errors.
22 INFO: [SIM 3] ***** CSIM finish *****
```

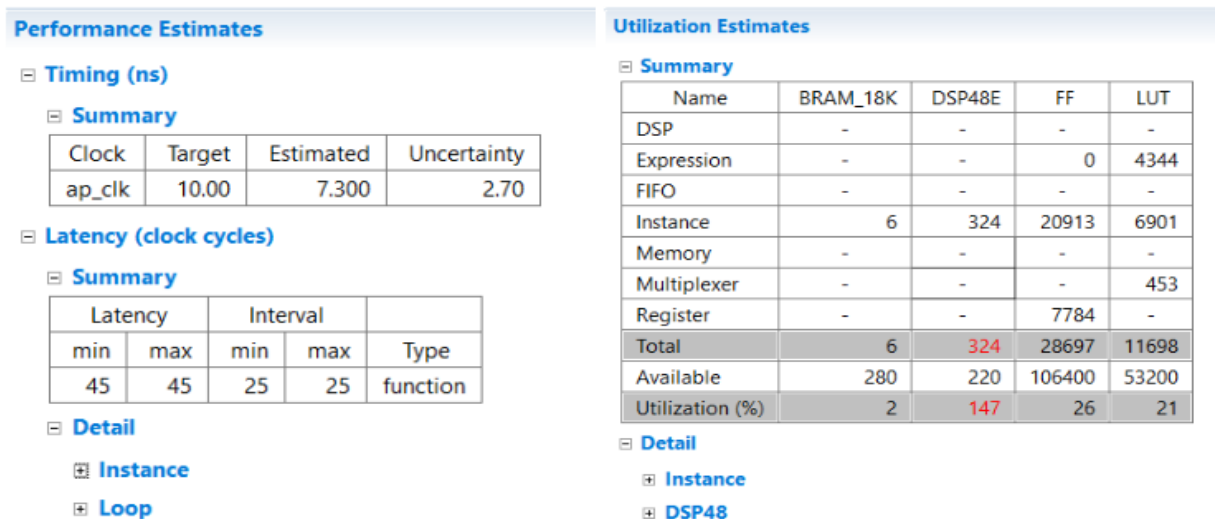
Hình 5. Kết quả thu được sau khi Run C Simulation (conv2D_csim.log)

Tổng hợp C (C Synthesis)

Mã C được tổng hợp sang mô tả phần cứng (RTL) bằng công cụ Vivado HLS. Quá trình này tạo ra các báo cáo chi tiết về:

- **Sử dụng tài nguyên phần cứng:** Bao gồm số lượng LUTs, FFs, DSPs và BRAM.
- **Độ trễ (Latency):** Đo thời gian thực hiện thuật toán trên phần cứng.
- **Tốc độ xử lý (Throughput):** Đánh giá khả năng xử lý song song trong thiết kế.

Kết quả từ quá trình tổng hợp C là cơ sở để tiếp tục kiểm chứng và tối ưu hóa thiết kế.



Hình 6. Kết quả thu được sau khi Run C Synthesis

Mô phỏng C/RTL Co-simulation

Sau khi mã C được tổng hợp thành mô tả RTL, quá trình mô phỏng đồng thời giữa mô hình C và RTL được thực hiện. Điều này nhằm kiểm tra xem thiết kế phần cứng có cho ra kết quả chính xác và khớp với mô hình phần mềm hay không. Đồng thời, quá trình này cũng cung cấp thông tin về hiệu suất của thiết kế sau khi tổng hợp.

Đánh giá hiệu suất

Thiết kế được đánh giá dựa trên các tiêu chí sau:

- **Độ trễ (Latency):** Đo thời gian xử lý cho mỗi phép toán tích chập.
- **Sử dụng tài nguyên:** Đánh giá mức độ sử dụng logic, bộ nhớ (BRAM), và các thành phần phần cứng khác.
- **Tốc độ xử lý:** Phân tích khả năng thực hiện các phép tính nhanh chóng trong các bài toán thời gian thực.

Kết quả từ các mô phỏng và đánh giá cho thấy thiết kế đạt hiệu suất cao, sử dụng tài nguyên hợp lý, và phù hợp với yêu cầu của ứng dụng.

7. Kết quả triển khai phần cứng trên ZynQ

Sau khi mô phỏng và tối ưu hóa, thiết kế được triển khai trên FPGA ZynQ-7000. Quá trình này bao gồm việc tạo file bitstream từ Vivado và tải chương trình vào phần cứng. Kết quả triển khai cung cấp các báo cáo về mức độ sử dụng tài nguyên như LUTs, flip-flops, DSPs và BRAM.

Đánh giá tài nguyên

- **LUTs và Flip-Flops:** Được sử dụng cho các phép toán logic và trạng thái.
- **DSPs:** Dùng cho phép toán nhân, tăng hiệu suất.
- **BRAM:** Lưu trữ dữ liệu đầu vào và kết quả, giúp giảm độ trễ.

Đánh giá hiệu năng

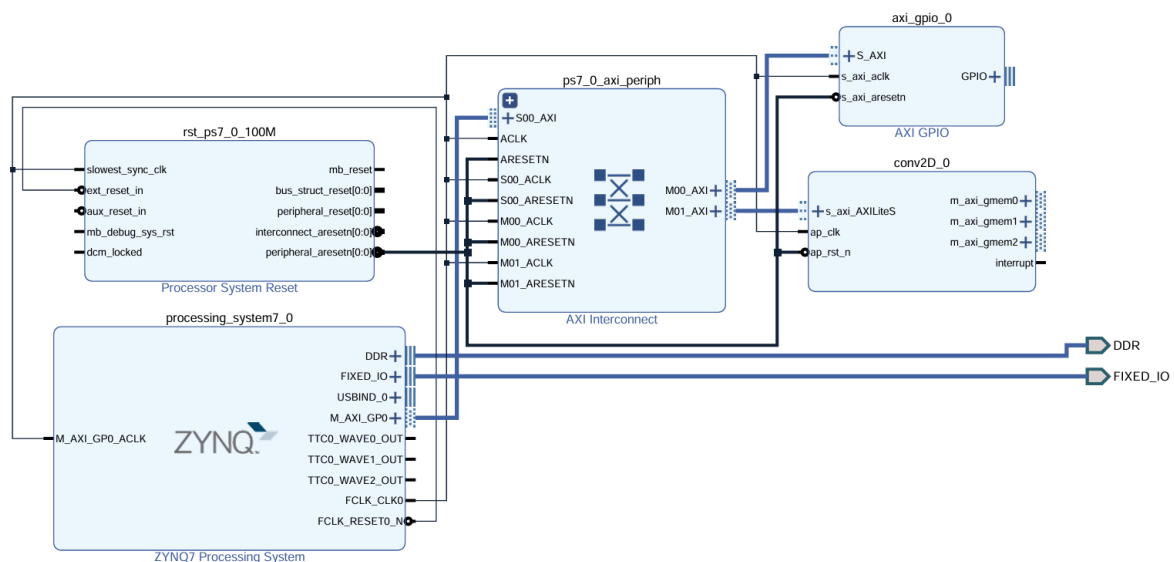
- **Thời gian thực hiện:** Đáp ứng yêu cầu về thời gian thực với tốc độ xử lý nhanh.
- **Độ trễ thấp:** Đảm bảo hiệu quả trong các ứng dụng thời gian thực.

Block Diagram

Các thành phần chính bao gồm:

- **BRAM:** Lưu trữ dữ liệu đầu vào, kernel và kết quả.
- **Processing Unit (DSP/Logic):** Thực hiện phép toán nhân.

Kết quả triển khai cho thấy thiết kế tối ưu tài nguyên và hiệu suất trên ZynQ-7000.



Hình 7. Block Design

8. Kết luận

Trong nghiên cứu này, chúng tôi đã thực hiện quá trình thiết kế, tối ưu hóa và triển khai thuật toán nhân chập 2D trên nền tảng FPGA ZynQ-7000. Quá trình này bao gồm các bước từ việc phát triển thuật toán bằng ngôn ngữ C, tổng hợp mã C sang RTL (VHDL/Verilog), mô phỏng C/RTL co-simulation, và triển khai thiết kế phần cứng trên FPGA để đánh giá hiệu quả và tài nguyên.

Kết quả thiết kế và tối ưu hóa:

- Thuật toán nhân chập 2D được triển khai và tối ưu hóa bằng các kỹ thuật như pipelining, unrolling và partitioning. Những kỹ thuật này giúp tối đa hóa hiệu suất của hệ thống, giảm độ trễ và tăng khả năng xử lý song song.
- **Pipelining** đã giúp giảm thiểu độ trễ giữa các chu kỳ tính toán, trong khi **unrolling** mở rộng các vòng lặp để thực hiện nhiều phép toán đồng thời, từ đó cải thiện tốc độ tính toán. **Partitioning** giúp tối ưu hóa bộ nhớ bằng cách chia nhỏ các mảng đầu vào và kết quả, giảm tắc nghẽn trong truy cập bộ nhớ.

Kết quả triển khai phần cứng trên ZynQ-7000:

- Việc triển khai trên ZynQ-7000 đã cho thấy rằng thiết kế có thể chạy hiệu quả và đáp ứng yêu cầu về tốc độ xử lý và tài nguyên phần cứng. Các tài nguyên như LUTs, flip-flops, DSPs và BRAM được sử dụng một cách hợp lý và không vượt quá giới hạn của FPGA, đảm bảo sự ổn định trong hoạt động.
- Hiệu suất tính toán đạt yêu cầu, với độ trễ thấp và khả năng xử lý song song các phép toán tích chập 2D, đáp ứng các yêu cầu về xử lý trong thời gian thực.

Mô phỏng C/RTL Co-simulation:

- Việc thực hiện mô phỏng C/RTL Co-simulation đã chứng minh tính chính xác của thiết kế. Kết quả đầu ra từ mô hình C và RTL hoàn toàn khớp nhau, khẳng định rằng quá trình tổng hợp và chuyển đổi từ mã C sang RTL không gây ra sai sót về logic.

Đánh giá hiệu suất và tài nguyên:

- Kết quả từ báo cáo tài nguyên cho thấy thiết kế sử dụng một lượng tài nguyên hợp lý của FPGA ZynQ-7000, bao gồm các DSPs cho phép toán nhân, BRAM cho việc lưu trữ dữ liệu, và LUTs cho phép toán logic. Mức sử dụng tài nguyên này là hợp lý và đáp ứng yêu cầu cho các ứng dụng có độ phức tạp vừa phải đến cao.
- Tốc độ xử lý và độ trễ của hệ thống được tối ưu hóa, cho phép xử lý các ma trận lớn trong thời gian ngắn, đáp ứng yêu cầu về xử lý thời gian thực.

Kết luận chung:

Quá trình triển khai thuật toán nhân chập 2D trên ZynQ-7000 đã thành công trong việc tối ưu hóa hiệu suất và sử dụng tài nguyên phần cứng. Các kỹ thuật tối ưu hóa như pipelining, unrolling và partitioning đã giúp giảm độ trễ và tăng hiệu suất, làm cho thiết kế có thể xử lý nhanh và hiệu quả các tác vụ phức tạp. Việc sử dụng ZynQ-7000, với khả năng tính toán song song cao và khả năng xử lý mạnh mẽ, cho thấy đây là một nền tảng thích hợp cho các ứng dụng cần xử lý tín hiệu thời gian thực, như trong các hệ thống nhận dạng hình ảnh và xử lý video.