**REPORT**

**Thiết kế và thực hiện mạch tính tổng các sai khác tuyệt đối (Sum of Absolute Differences – SAD) giữa hai ma trận kích thước M×N**

Ver 1.0

20/04/2024

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Họ và tên (Full name)** | **Mã SV (ID)** | **Đóng góp (Contribution)** |
| Thành viên 1 (Member 1) | Ngạc Anh Kiệt | 21020690 | Datapath, Components, Code VHDL and Debug |
| Thành viên 2  (Member 2) | Phạm Lê Đức Thành | 21021637 | Controller, Testbench, Code VHDL |
| Tên/Địa chỉ Repo trên Github hoặc Google Drive |  | | |

|  |
| --- |
| **Tóm tắt (Abstract - from 5 to 10 lines)** |
| Bài báo cáo này sẽ thiết kế mức RTL một hệ thống có thể nhận vào hai ma trận và in ra Tổng sai khác tuyệt đối của hai ma trận đó. Từ các nguyên lý này, chúng tôi đã viết một chương trình VHDL hoàn chỉnh để thực thi. Ngoài ra, chúng tôi cũng thực hiện phân tích thuật toán, lưu đồ, sơ đồ liên quan và thực hiện chạy mô phỏng và đánh giá kiểm chứng kết quả. |

|  |
| --- |
| **Từ khóa (Keywords)** |
| SAD, RTL, VHDL |

Document History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Time** | **Revised by** | **Description** |
| V1.0 | 20/04/2024 | Nguyễn Kiêm Hùng | Problem definition |
| V1.2 |  |  |  |
|  |  |  |  |

|  |
| --- |
| **Hướng dẫn (Instructions)** |
| 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   Ngoại trừ phần thông tin sinh viên, mã số sinh viên và tên/địa chỉ Repo trên Github 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 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.** |

MỤC LỤC

[Document History 3](#_Toc117778198)

[MỤC LỤC](#_Toc117778199) 5

[1. Giới thiệu](#_Toc117778200) 6

[2. Yêu cầu](#_Toc117778201) 6

[2.1. Yêu cầu đối với thiết kế:](#_Toc117778202) 6

[2.2. Định nghĩa giao diện vào/ra](#_Toc117778203) 7

[3. Thuật toán](#_Toc117778204) 8

[4. Thiết kế mức RTL](#_Toc117778205) 9

[4.1. Mô hình máy FSMD](#_Toc117778206) 9

[4.2. Đơn vị xử lý dữ liệu (Datapath)](#_Toc117778207) 10

[4.3. Đơn vị điều khiển (Control Unit)](#_Toc117778208) 13

[4.4. Sơ đồ khối tổng thể](#_Toc117778209) 15

[5. Mô hình hóa bằng VHDL](#_Toc117778210) 15

[6. Mô phỏng/thực thi và đánh giá](#_Toc117778211) 16

[7. Kết luận](#_Toc117778212) 17

[List of Figures](#_Toc117778217) 20

[List of Tables](#_Toc117778218) 21

[References](#_Toc117778219) 22

# 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 tổng các sai khác tuyệt đối (Sum of Absolute Differences - SAD) giữa hai ma trận kích thước M×N. Trong đó, mỗi phần tử trong các ma trận là các số nguyên có dấu. Phép tính SAD được minh họa bằng chương trình C trong Hình 1.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | *// Hàm tính tổng của sai khác tuyệt đối (SAD) giữa hai ma trận*  int computeSAD(int\*\* mat1, int\*\* mat2, int M, int N)  {  int i, j;  int SAD = 0;  *// Duyệt qua từng phần tử của ma trận và tính SAD*  for (i = 0; i < M; i++) {  for (j = 0; j < N; j++) {  SAD += abs(mat1[i][j] - mat2[i][j]);  }  }  return SAD;  } |

Hình 1. Chương trình C minh họa phép tính SAD.

# Yêu cầu

## Yêu cầu đối với thiết kế:

* SV thiết kế mạch phần cứng gọi là computeSAD thực hiện tính SAD theo thuật toán được minh họa trong Hình 1
* Các ma trận đầu vào I có kích thước M×N có thể được thiết lập bởi CPU
* Khối computeSAD có giao diện ghép nối tới CPU sao cho CPU kích hoạt quá trình tính toán của khối computeSAD bằng các đặt tín hiệu Start = ‘1’.
* Sau khi quá trình tính SAD hoàn thành, khối computeSAD sẽ báo cho CPU biết bằng cách đặt tín hiệu Done = ‘1’;
* Khối computeSAD có 1 giao diện ghép nối tới tới bộ nhớ để đọc các ma trận đầu vào
* Khối computeSAD có 1 giao diện ghép nối tới tới bộ nhớ để ghi dữ liệu đầu ra

A diagram of a computer software

Description automatically generated

Hình 2. Giao diện ghép nối I/O.

## Định nghĩa giao diện vào/ra

Bảng 1: Mô tả các tín hiệu vào ra.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TT** | **Port** | **Direction** | **Width** | **Meaning** |
| 1 | CLK | Vào | 1 | Tín hiệu Clock |
| 2 | RST | Vào | 1 | Tín hiệu Reset |
| 3 | START | Vào | 1 | Tín hiệu Start |
| 4 | W\_en\_x | Vào | 1 | Tín hiệu Write Enable của MemoryX |
| 5 | W\_en\_y | Vào | 1 | Tín hiệu Write Enable của MemoryY |
| 6 | X\_in | Vào | 16 | Tín hiệu vào của ma trận X |
| 7 | Y\_in | Vào | 16 | Tín hiệu vào của ma trận Y |
| 8 | SAD\_o | Ra | 16 | Tín hiệu Tổng Sai số tuyệt đối giữa hai ma trận X, Y |
| 9 | Done | Ra | 1 | Tín hiệu Done báo hiệu kết thúc |

# Thuật toán :

Thuật toán sử dụng trong dự án đã được nêu ở Hình 1, tuy nhiên, làm việc với mảng hai chiều trong VHDL sẽ gây rất nhiều khó khăn, vậy nên chúng tôi sẽ sửa lại thuật toán. Cụ thể, mảng hai chiều kích cỡ chiều rộng m và chiều dài n sẽ được chuyển đổi thành một mảng một chiều có chiều dài là m nhân x. Thuật toán sẽ được viết lại như sau :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | *// Hàm tính tổng của sai khác tuyệt đối (SAD) giữa hai ma trận*  int computeSAD(int\*\* mat1, int\*\* mat2, int M, int N)  {  int i;  int SAD = 0;  *// Duyệt qua từng phần tử của ma trận và tính SAD*  for (i = 0; i < M\*N; i++) {  SAD += abs(mat1[i] - mat2[i]);  }  }  return SAD;  } |

Hình 3. Chương trình C minh họa phép tính SAD sau khi chỉnh sửa

Trước tiên, ta khởi tạo hai giá trị i và SAD = 0. Sau đó, ta dùng biến i để đồng thời lặp qua ma trận ảnh mat1 và mat2. Trong lúc lặp, ta thực hiện tính Sai số tuyệt đối (Lấy trị tuyệt đối phép trừ mat1[i] – mat2[i]), đồng thời cộng dồn vào giá trị SAD.

Sau khi thoát vòng lặp, ta trả về SAD, chính là tổng sai khác tuyệt đối giữa hai ma trận mat1 và mat2.

# Thiết kế mức RTL

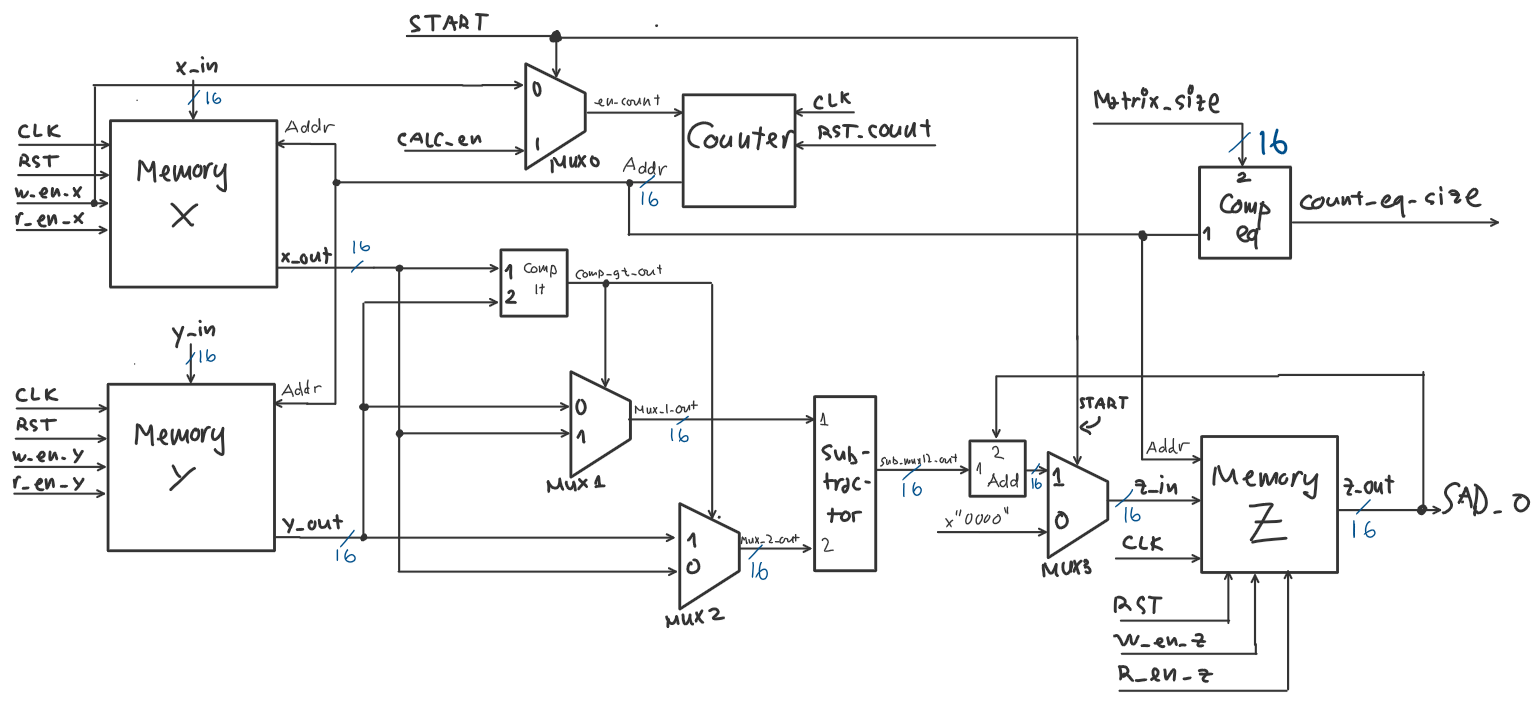
## Mô hình máy FSMD

A diagram of a algorithm

Description automatically generated

Hình 4: Mô hình máy FSMD.

## Đơn vị xử lý dữ liệu (Datapath)



Hình 5.1: Cấu trúc của đơn vị xử lý dữ liệu Datapath.

Trước khi tín hiệu start được gán bằng 1, ta sẽ nhập dữ liệu vào hai thanh ghi[1] (MemoryX, MemoryY), công việc này sẽ được thực hiện ở Stage S1 và S2. Để có thể nhập được dữ liệu, ta cần xung đếm từ bộ Counter, ở đây chúng tôi đã sử dụng chính xung Write Enable của thanh ghi X (W\_en\_x) làm một Counter enable cho Counter (En\_count). Tín hiệu Start lúc này đóng vai trò như một tín hiệu quyết định Mux0, khi việc tính toán chưa bắt đầu, tức Start = 0, Mux0 sẽ lựa chọn tín hiệu W\_en\_x, và lúc này, mỗi khi ta cho phép thanh ghi được ghi bằng sướn dương của W\_en\_x, Counter lúc này cũng sẽ được phép đếm và sẽ bắt đầu đếm. Giá trị đầu ra của counter lúc này chính là địa chỉ thanh ghi X. Sau khi hoàn thành việc ghi giá trị, ta ngưng cấp các xung W\_en\_x và gán Start bằng ‘1’.

A diagram of a diagram

Description automatically generated

Hình 5.2: Cấu trúc của bộ đếm Counter.

*[1]: Xem thêm ở APPENDIX A*

Khi Start = 1, hệ thống sẽ đi đến stage s3, ta gán tín hiệu Reset của Counter cho 1 (RST\_count = 1), lúc này Count sẽ xóa giá trị đếm hiện tại và trở về trạng thái ban đầu. Tiếp theo ta gán lần lượt các giá trị Read enable của thanh ghi X, Y bằng 1 để hệ thống thực hiện đọc các giá trị ta vừa ghi vào thanh ghi ở Stage trước. Tiếp đến là Read và Write enable của thanh ghi Z bằng 1, thanh ghi Z chính là thanh ghi đầu ra, vừa thực hiện công việc ghi các dữ liệu đầu ra, vừa thực hiện tính toán dựa trên các tín hiệu đó. Trong Stage này, mỗi một lần đếm, sẽ có 2 tín hiệu đầu ra X\_out và Y\_out của hai thanh ghi X, Y được đọc.

A diagram of a diagram

Description automatically generated

Hình 5.3: Cấu trúc của bộ tính toán sai khác tuyệt đối.

Tín hiệu X\_out sẽ đi vào chân 1 của bộ so sánh hơn, chân 1 của Mux1 và chân 0 của Mux2, còn tính hiệu Y\_out sẽ được nối vào chân 2 của bộ so sánh hơn, chân 0 của Mux1 và chân 1 của Mux2 như sơ đồ hình 5.3. Ở đây, bộ so sánh hơn sẽ so sánh liệu tín hiệu chân 1 có lớn hơn tín hiệu chân 2 không, nếu có thì đầu ra là 1, ngược lại đầu ra sẽ là 0. Tín hiệu đầu ra của bộ so sánh trên cũng đóng vai trò là tín hiệu quyết định của hai bộ Mux1 và Mux2 đã đề cập bên trên. Cuối cùng, hai tín hiệu đầu ra của hai Mux sẽ được đưa vào một bộ trừ đại số. Quá trình trên thực chất là việc so sánh xem tín hiệu X\_out hay Y\_out lớn hơn, nếu tín hiệu này lớn hơn tín hiệu kia thì lấy tín hiệu này trừ tín hiệu kia.

Tiếp theo, ta thực hiện tính toán cộng dồn các giá trị ta thu được sau khi trừ để ra được giá trị tổng mong muốn. Để thực hiện việc này, ta cần có một thanh ghi MemoryZ để thực hiện ghi giá trị. Ở giai đoạn này, ta trước hết cần kiểm tra tín hiệu start được nối với chân quyết định của Mux3, đây là tín hiệu được điều khiển từ controller và sẽ được kích hoạt khi bằng 0 ở ngay Stage 1. Tín hiệu này sẽ đi vào Mux3 và khởi tạo cho Z\_in giá trị bằng 0. Sau đó, tín hiệu Z\_in đã được khởi tạo bằng 0, sẽ được cộng lần lượt với các giá trị đầu ra từ sub\_mux12\_out, bởi vì tại thời điểm bắt đầu cộng, tín hiệu Start đã được gán 1, tức là ta sẽ chọn cổng số 1 của Mux3 chứ không còn khởi tạo giá trị như ban nãy nữa. SAD\_o sẽ liên tục được gán bằng Z\_out, với Z\_out chính là Z\_in tại địa chỉ Addr tương ứng.

A diagram of a computer hardware

Description automatically generated

Hình 5.4: Cấu trúc của bộ cộng dồn giá trị cùng bộ nhớ output

Cuối cùng, để có thể điều khiển được điểm dừng, ta sẽ so sánh giá trị địa chỉ hiện tại với kích cỡ ma trận, nếu bằng nhau, bộ so sánh sẽ trả về tín hiệu count\_eq\_size = ‘1’, và giá trị này sẽ được đưa vào controller, được sử dụng để đưa hệ thống đến stage cuối cùng. Lúc này, SAD\_o sẽ chứa giá trị Tổng sai số tuyệt đối cuối cùng.

A diagram of a complex code

Description automatically generated with medium confidence

Hình 5.5: Cấu trúc bộ so sánh thoát vòng lặp

## Đơn vị điều khiển (Control Unit)

A diagram of a flowchart

Description automatically generated

Hình 6: Máy FSM của đơn vị điều khiển.

Trước nhất, ta thực hiện gán RST = ‘1’, đây là bước “làm sạch” toàn bộ thanh ghi và biến được sử dụng trong toàn bộ quá trình tính toán.

Tại Stage S1, ta đồng thời gán các tín hiệu Done = ‘0’ – tín hiệu Done là một tín hiệu thông báo rằng việc tính toán đã hoàn thành hay chưa, ta thực hiện khởi tạo tín hiệu này tại Stage này, và RST\_count = ‘1’ – RST\_count là tín hiệu làm mới lại bộ đếm, bộ đếm được sử dụng để truy cập địa chỉ các thanh ghi.

Tại Stage S2, ta kiểm tra xem CPU đã gửi tín hiệu Start = ‘1’ về chưa. Tín hiệu Start = ‘1’ chỉ được gửi khi mà việc nhập dữ liệu vào hai thanh ghi MemoryX và MemoryY đã hoàn thành. Nếu Start = ‘1’, ta thực hiện sang Stage kế tiếp là Stage tính toán, còn nếu không ta tiếp tục lặp lại Stage S2 cho đến khi việc nhập hoàn thành.

Tại Stage S3, ta thực hiện gán RST\_count = ‘1’, mục đích là để làm mới bộ đếm, bởi ở các Stage trước, bộ đếm đã thực hiện đếm từ 0 đến kích cỡ của ma trận nhằm phục vụ việc ghi giá trị vào đúng địa chỉ thanh ghi, nên tại đây ta cần làm mới lại giá trị này để có thể thực hiện đếm khi đọc giá trị từ thanh ghi.

Ở Stage S4, ta gán R\_en\_x = ‘1’ và R\_en\_y = ‘1’. Đây là hai tín hiệu Read Enable của hai thanh ghi MemoryX và MemoryY, nhằm phục vụ việc đọc các giá trị từ hai thanh ghi ra để thực hiện tính toán.

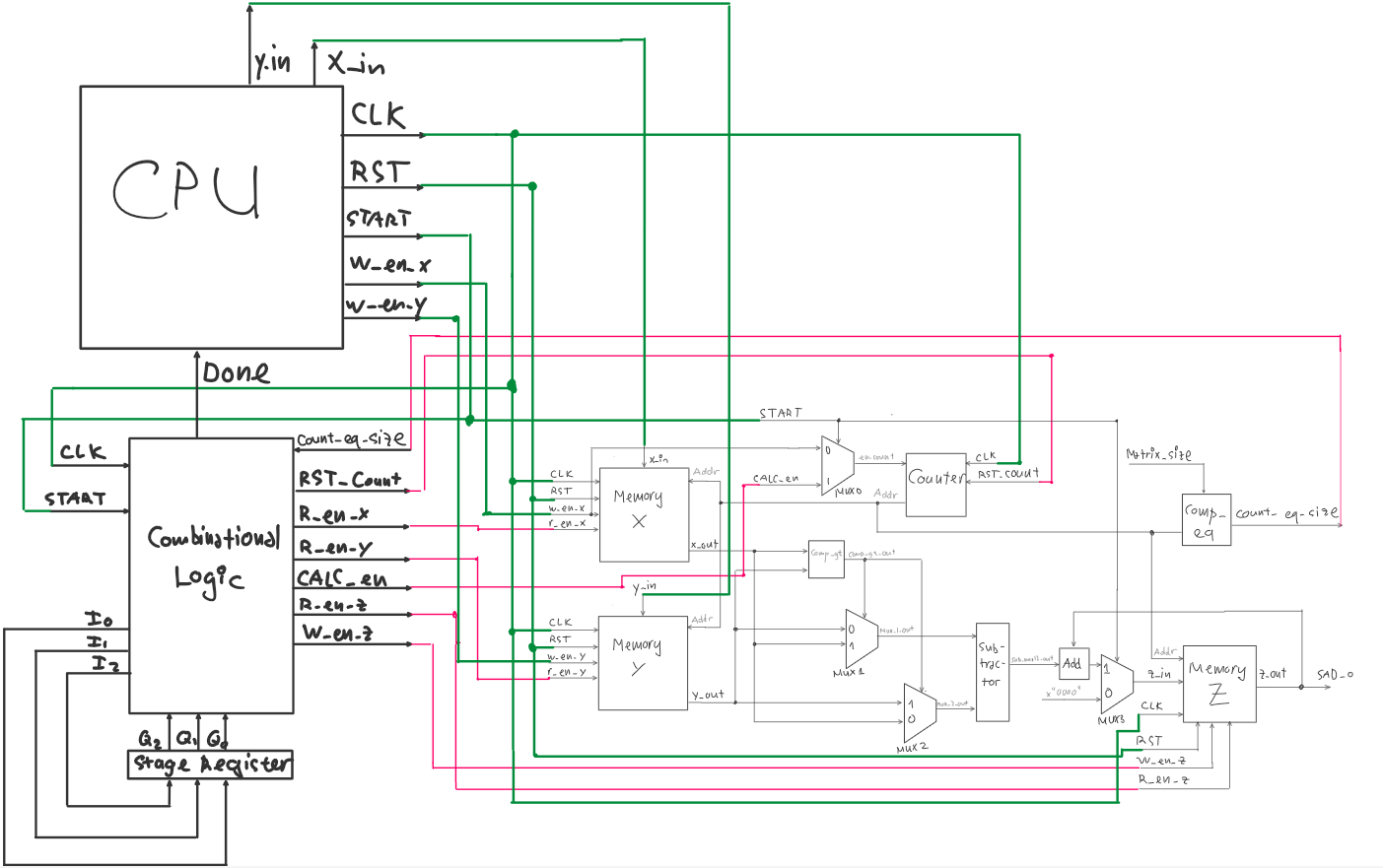
Stage S5 và S6 lần lượt là hai Stages gán W\_en\_z = ‘1’ và R\_en\_z = ‘1’, sở dĩ ta phải đưa hai stage này riêng biệt bởi ta thường không mong muốn việc Đọc và Ghi xảy ra trên cùng xung CLK, việc này có thể gây nên sự rối loạn, sai lệch không mong muốn.

Hoàn thành ba Stages S4, S5, S6 là ta đã hoàn thành việc lấy giá trị, tính toán và ghi giá trị vào MemoryZ và đọc giá trị đầu ra của MemoryZ vào tín hiệu đầu ra cuối cùng SAD\_o, ở Stage S7, ta gán tín hiệu Calc\_en = ‘1’ để bộ đếm thực hiện việc đếm tới giá trị địa chỉ tiếp theo.

Sau khi giá trị địa chỉ được cộng, tại Stage 8, ta sẽ kiểm tra xem giá trị địa chỉ đã bằng giá trị Matrix\_size chưa. Ở đây, Matrix\_size là tổng số ô nhớ trên thanh ghi (trong báo cáo này, là 16), tức thanh ghi sẽ có ô địa chỉ chạy từ 0 tới 15. Vậy nếu biến địa chỉ đạt tới địa chỉ ô nhớ thứ 16, chương trình sẽ đi tới Stage 9, còn nếu không, ta sẽ lặp lại Stage 4 để thực hiện tiếp tục tính toán các giá trị tiếp theo.

Cuối cùng, ở Stage 10, nếu nhận được tín hiệu Start = ‘1’ từ CPU, hệ thống sẽ quay lại Stage 1, làm mới lại các tín hiệu và thanh ghi để phục vụ tính toán ma trận mới. Còn nếu không, Stage 10 sẽ tiếp tục được lặp lại và hệ thống không tính nữa.

## Sơ đồ khối tổng thể



Hình 7: Sơ đồ khối tổng thể của thiết kế.

Hình trên gồm bộ Điều khiển, bộ Tính toán và CPU, cùng với các dây được nối từ cổng của bộ này sang cổng của bộ kia. Đây là một sơ đồ khối hoàn chỉnh của thiết kế.

# Mô hình hóa bằng VHDL

Tổ chức tệp mã nguồn như hình dưới đây.

A screenshot of a computer

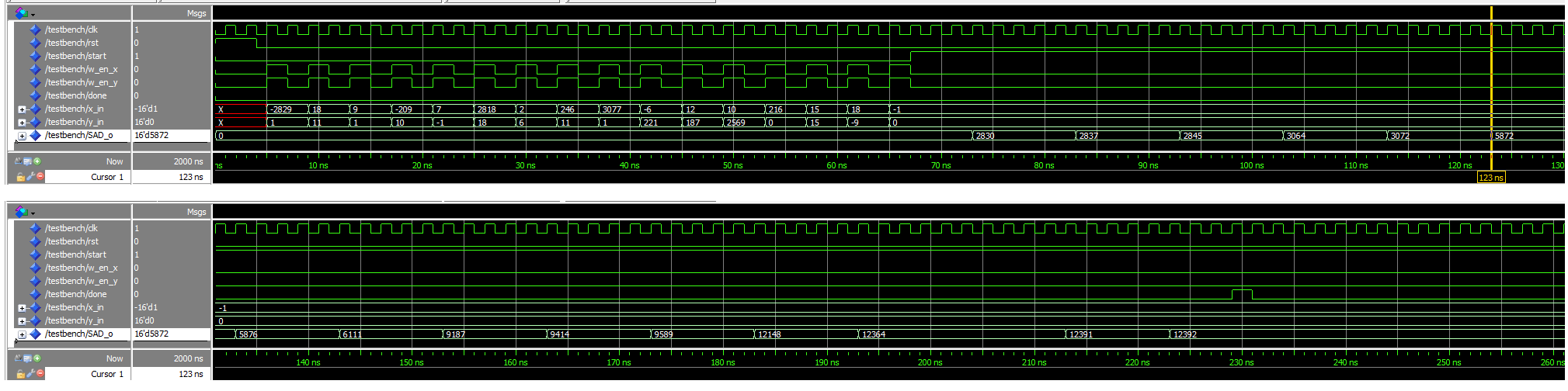
Description automatically generated

Hình 8: Tổ chức tệp của mã nguồn VHDL

Trong đó, testbench.vhd là tệp testbench, đại diện cho CPU của người dùng. Các tác vụ Clock, reset, start để được truyền từ đây vào datapath và controller. Tệp SADlibrary.vhd là một tệp header chứa tất cả các thực thể con trong nó, phục vụ sự thuận tiện trong khai báo. Tệp datapath.vhd và controller.vhd là hai tệp VHDL của datapath và controller của chương trình, đã được phân tích tại mục 4.2 và 4.3. Tệp Counter.vhd là tệp VHDL mô phỏng lại khối đếm cộng, còn tệp ArrayIn.vhd là tệp mô phỏng lại các thanh ghi. Tệp computeSAD là tệp mô hình tổng thể của toàn bộ hệ thống.

# Mô phỏng/thực thi và đánh giá:

Ta thực hiện chạy chương trình và mô phỏng sóng.



**Hình 9: Mô phỏng tín hiệu**

Có thể thấy, hai thanh ghi chứa dữ liệu hiện đang có kích cỡ là 16 ô (chạy từ 0 đến 15), mỗi giá trị có 15-bit và là có dấu.

Từ 0 ns đến 5 ns, tín hiệu RST = ‘1’ để thực hiện làm mới các thanh ghi và giá trị. Bắt đầu từ giây thứ 5 ns, ta bắt đầu đọc từng giá trị vào thanh ghi MemoryX và MemoryY, cụ thể cứ mỗi 1 xung CLK, tại sườn dương CLK, xung w\_en\_x và w\_en\_y sẽ bằng 1 và giá trị sẽ được ghi vào tại thời điểm sườn dương của hai tín hiệu Enable trên. Ở 67 ns, hai thanh ghi đã hoàn thành việc ghi giá trị, và tín hiệu Start được gán giá trị ‘1’, tức là hệ thống sẽ bắt đầu tính toán.

Từ thời điểm này đi, tín hiệu SAD\_o sẽ cộng dồn các giá trị sai khác tuyệt đối, tính từ địa chỉ số 0 của thanh ghi, ví dụ như tại vị trí số 0, đầu ra của MemoryX là -2829 (trong hệ 10), và đầu ra của MemoryY là 1. Sai khác tuyệt đối của hai giá trị này là |-2829 – 1| và bằng 2830. Giá trị 2830 cũng xuất hiện tại thời điểm 73 ns, chính là thời điểm hệ thống thực hiện tính toán xong và in ra SAD\_out tại thời điểm đó. Tiếp tục tính toàn, đến thời điểm 229 ns, SAD\_o lúc này bằng 12392 và tín hiệu Done được gán bằng ‘1’, báo hiệu rằng việc tính toán đã hoàn tất. Đầu ra cuối cùng của chương trình chính là 12392.

Kiểm chứng kết quả trên, ta thực hiện tính tay:

A black background with white numbers and symbols

Description automatically generated

Nhận thấy kết quả hoàn toàn khớp.

# Kết luận

Ở dự án này, chúng tôi đã xây dựng nên được một hệ thống tính toán tổng sai số tuyệt đối của hai ma trận đầu vào. Kích cỡ ma trận hiện đang cố định là 16 phần tử cũng như kích cỡ của một phần tử là 16 bits. Ngoài ra, mạch cũng hoạt động với cả số âm lẫn số dương, khiến cho việc tính toán trở nên đa dạng hơn. Các sơ đồ thiết kế Tổng thể, Controller và Datapath cũng đã được cung cấp đầy đủ.

Tuy nhiên, vấn đề Tràn bit hay Overflow[2], vẫn hoàn toàn có thể xảy ra, nếu dữ liệu đầu vào chưa hợp lý dẫn đến việc cộng dồn tổng bị quá lớn. Để xử lý vấn đề này, ta có thể tăng số bits của SAD\_o lên thành 32 và tích hợp thêm mạch Sign Extends để tăng số bits của giá trị. Thực tế, việc làm trên cũng không hoàn toàn giải quyết được vấn đề Overflow, bởi với một số lượng phần tử đủ nhiều và các cặp số có giá trị Sai khác tuyệt đối quá lớn, thì biến 32 bits vẫn có thể bị tràn.

*[2]: Xem thêm ở APPENDIX A*

**APPENDIX A: SỐ ÂM VÀ HIỆN TƯỢNG TRÀN BIT (OVERFLOW)**

Hiện tượng Tràn bit là hiện tượng xảy ra khi giá trị con số vượt quá kích cỡ mà nó được cung cấp, cụ thể ở đây ta xét hiện tượng tràn bit của số nhị phân có dấu. Để chuyển đổi từ số nhị phân không dấu (unsigned) sang số nhị phân có dấu (signed), ta thực hiện đảo toàn bộ các bit sau đó cộng với 1, đây còn được gọi là bù 2.

Ví dụ, số 12 được biểu diễn dưới dạng số nhị phân sẽ có dạng: 0000 1100

Vậy nếu quy đổi sang -12, số nhị phân này thành:

11110011 + 00000001

= 11110100

Nếu giờ ta cấp phát cho một biến số nhị phân 8 bits tất cả, vậy biến này sẽ có thể biểu diễn được từ -128 trong hệ thập phân (tức 10000000 trong hệ nhị phân) đến 127 trong hệ thập phân (tức 01111111 trong hệ nhị phân) Vậy thì tất cả các giá trị nằm ngoài khoảng này, biến nhị phân này sẽ không thể biểu diễn được và từ đó sẽ gây ra hiện tượng Tràn bit. Thực vậy, nếu ta thực hiện cộng một vào 01111111, tức ta thu lại 10000000, nếu xét có dấu thì đây là số -128, trong khi rõ ràng, 127 + 1 = 128, chứ không phải -128.

Ở dự án phía trên, chúng tôi cấp cho các biến dữ liệu có dấu của chúng tôi 16 bits, tức có thể biểu diễn từ -32768 đến 32767 trong hệ thập phân.

Ta nhận thấy một điểm đặc biệt, đó là tín hiệu đầu ra SAD\_o, là một giá trị luôn dương, bởi lẽ nó là tổng tích lũy của các số được lấy giá trị tuyệt đối. Vì vậy, tuy các biến của ma trận đầu vào phải là biến có dấu, thì tín hiệu SAD\_o này hoàn toàn có thể được khởi tạo là một tín hiệu không xét dấu, hay chính là tín hiệu Unsigned. Nếu làm như vậy, giá trị mà SAD\_o có thể biểu diễn được tăng lên đáng kể, ta có thêm một bit nữa để biểu diễn giá trị, khoảng giá trị hệ thập phân sẽ tăng từ (-32768 đến 32767) thành (0 đến 65535).

**APPENDIX B: MEMORY**

Memory hay chính là bộ nhớ, là một phân vùng được sử dụng để chứa dữ liệu, với mỗi một ô nhớ là một bit, mỗi ô nhớ đều tương ứng với một giá trị địa chỉ riêng. Hình dưới đây mình họa một thanh địa chỉ có 15 bit, địa chỉ được ghi bên dưới môi ô, dưới dạng số nhị phân 4 bits.



**Hình B.1: Thanh ghi 1**

Trong dự án này, chúng tôi sử dụng hệ Hex, tức một bit Hex bằng 4\_bit Binary, hay chính là 1 byte, các số Hex này là Hex có dấu (signed), nên thực tế một Hex cần 2 bytes, với một byte đầu tiên để mã hóa dấu, byte còn lại là giá trị. Dưới đây là hình ảnh thanh ghi minh họa, với thanh ghi phía trên là thanh ghi Binary 32 bits và thanh ghi bên dưới là Hex 8 bytes.

A close-up of a number

Description automatically generated

**Hình B.2: Thanh ghi 2**

Có thể thấy việc cấp phát bộ nhớ cho từng biến và từng bộ nhớ một cách chính xác là rất quan trọng, nếu ta cấp thiếu bộ nhớ, hệ thống sẽ gặp lỗi tràn bộ nhớ - là một lỗi nghiêm trọng (Fatal error) và sẽ dừng hoàn toàn quá trình chạy của chương trình, trong khi nếu cấp phát thừa bộ nhớ lại dẫn đến tốn tài nguyên.

List of Figures

[Hình 1. Chương trình C minh họa phép tính SAD.. 6](#_Toc496109799)

[Hình 2. Giao diện ghép nối I/O. 7](#_Toc496109800)

[Hình 3: Chương trình C minh họa phép tính SAD sau khi chỉnh sửa. 9](#_Toc496109801)

Hình 4: Mô hình máy FSMD. 10

Hình 5.1: Cấu trúc của đơn vị xử lý dữ liệu Datapath... 11

Hình 5.2: Cấu trúc của bộ đếm Counter. 11

[Hình](#_Toc496109803) 5.3: Cấu trúc của bộ tính toán sai khác tuyệt đối. 12

Hình 5.4: Cấu trúc của bộ cộng dồn giá trị cùng bộ nhớ output. 13

Hình 5.5: Cấu trúc bộ so sánh thoát vòng lặp. 13

Hình 6: Máy FSM của đơn vị điều khiển. 14

Hình 7: Sơ đồ khối tổng thể của thiết kế. 16

Hình 8: Tổ chức tệp của mã nguồn VHDL. 16

Hình 9: Mô phỏng tín hiệu. 17

Hình B.1: Thanh ghi 1. 20

Hình B.2: Thanh ghi 2. ……………………………………………………………………….20

List of Tables

[Bảng 1: Mô tả các tín hiệu vào ra. 8](#_Toc496109879)