**REPORT**

**Thiết kế và thực hiện khối tính ảnh tích phân 2-D**

Ver 1.0

27/4/2025

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Họ và tên (Full name)** | **Mã SV (ID)** | **Đóng góp (Contribution)** |
| Thành viên 1 (Member 1) | Đỗ Quốc Việt | 21020951 | Thiết kế thuật toán, FSMD, datapath, controller. |
| Thành viên 2  (Member 2) | Hoảng Huy Tuấn |  | Viết code cho các thành phần con và testbench. |
| Tên/Địa chỉ Repo trên Github hoặc Google Drive | https://github.com/vjt081103/Digital\_Final\_Project | | |

|  |
| --- |
| **Tóm tắt (Abstract - from 5 to 10 lines)** |
| Khối tích phân ảnh 2D xử lý ảnh để thực hiện tính ma trận tích phân ảnh cho một ảnh có kích thước M x N được cho trước. Khối sẽ nhận các giá trị bao gồm giá trị kích thước của ảnh M và N đồng thời nhận 2 giá trị 32 bit là địa chỉ của ảnh và địa chỉ để lưu ma trận của ảnh trong bộ nhớ lưu được 1 byte cho mỗi ô địa chỉ. Sau khi khởi động, khối tích phân sẽ tính toán và lưu giá trị ma trận thu được vào trong bộ nhớ với giá trị địa chỉ cơ sở là địa chỉ đích đã nhận. Khối tích phân này cũng sẽ nhận được quyền đọc và ghi của bộ nhớ đã cho. |

|  |
| --- |
| **Từ khóa (Keywords)** |
| Integral, memory,… |

Document History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Time** | **Revised by** | **Description** |
| V0.1 | 27/04/2021 | Nguyễn Kiêm Hùng | Original Version |
| V0.2 | 18/5/2021 | Đỗ Quốc Việt | Thêm thuật toán ban đầu |
| V0.3 | 28/5/2021 | Đỗ Quốc Việt | Thêm datapath, danh sách các chân, controller |
| V0.4 | 1/6/2021 | Hoàng Huy Tuấn | Kết quả chạy đợt 1 |
| V0.5 | 5/6/2021 | Đỗ Quốc Việt | Chỉnh sửa lại thuật toán |
| V0.6 | 6/6/2021 | Hoàng Huy Tuấn | Kết quả chạy sau khi sửa thuật toán |
| V0.7 | 8/6/2021 | Hoàng Huy Tuấn | Hoàn thiện lần 1 |
| V0.8 | 11/6/2021 | Đỗ Quốc Việt | Hoàn thiện lần 2 |

MỤC LỤC

[Document History 3](#_Toc197372625)

[MỤC LỤC 4](#_Toc197372626)

[1. Giới thiệu 5](#_Toc197372627)

[2. Yêu cầu 6](#_Toc197372628)

[2.1. Yêu cầu đối với thiết kế: 6](#_Toc197372629)

[2.2. Định nghĩa giao diện vào/ra 8](#_Toc197372630)

[3. Thuật toán 8](#_Toc197372631)

[4. Thiết kế mức RTL 11](#_Toc197372632)

[4.1. Mô hình máy FSMD 11](#_Toc197372633)

[4.2. Đơn vị xử lý dữ liệu (Datapath) 13](#_Toc197372634)

[4.3. Đơn vị điều khiển (Control Unit) 15](#_Toc197372635)

[4.4. Sơ đồ khối tổng thể 10](#_Toc197372636)

[5. Mô hình hóa bằng VHDL 10](#_Toc197372637)

[6. Mô phỏng/thực thi và đánh giá 17](#_Toc197372638)

6.1. Chạy testbench trên ModelSim……………………………………………………………….17

6.2. Tích hợp trên vivado…………………………………………………………………………..18

[7. Kết luận 19](#_Toc197372639)

[Appendix A: Reference 20](#_Toc197372640)

[Appendix B: VHDL Code 21](#_Toc197372642)

[Appendix C: 22](#_Toc197372644)

[List of Figures 23](#_Toc197372645)

[List of Tables 24](#_Toc197372646)

[References 25](#_Toc197372647)

# 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 ảnh tích phân J = IntegralImage(I) (tương tự chức năng hàm integralImage() trong Matlab [1]) từ một ảnh gốc I. Trong đó, mỗi pixel trong hình ảnh tích phân J đại diện cho tổng tích lũy của một pixel đầu vào với tất cả các pixel ở phía trên và bên trái của pixel đầu vào trong ảnh gốc I. Phép chuyển đổi hình ảnh 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 =

thì kết quả tính toán của khối chuyển đổi IntegralImage trả về hình ảnh J có kích thước 6×6 như sau:

J =

Hình 1. Ví dụ minh họa phép chuyển đổi hình ảnh.

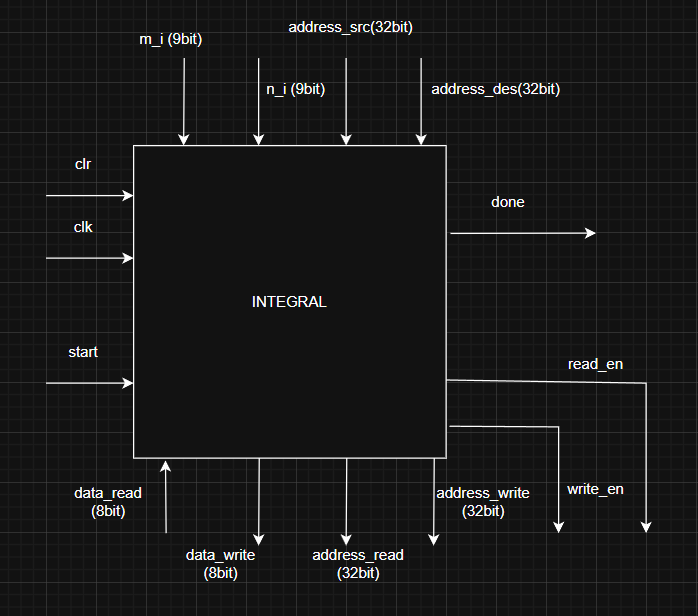
Lưu ý rằng pixel có tọa độ (r,c) – (hàng, cột) – tronh ảnh gốc được ánh xạ tới pixel có tọa độ (r + 1, c + 1) trong ảnh tích phân. Các bước tính hình ảnh tích phân của hình ảnh đầu vào I được minh họa như sau:

* Các pixel trong hàng và cột đầu tiên trong ảnh tích phân đều là 0
* Pixel trong ma trận ban đầu tại tọa độ (1, 1) có giá trị 17 không thay đổi trong ảnh tích phân vì không có pixel nào khác trong tổng. Do đó, pixel trong ảnh tích phân tại tọa độ (2, 2) có giá trị 17.
* Pixel trong ma trận ban đầu tại tọa độ (1, 2) ánh xạ tới pixel (2, 3) trong ảnh tích phân. Giá trị là tổng của giá trị pixel ban đầu (24), các pixel phía trên nó (0) và các pixel bên trái của nó (17): 24 + 17 + 0 = 41.
* Pixel trong ma trận ban đầu tại tọa độ (1, 3) ánh xạ tới pixel (2, 4) trong hình ảnh tích phân. Giá trị là tổng của giá trị pixel ban đầu (1), pixel phía trên nó (0) và các pixel bên trái của nó (đã được cộng lại bằng 41). Do đó, giá trị tại pixel (2,4) trong ảnh tích phân là 1 + 41 + 0 = 42.
* …

# Yêu cầu

## Yêu cầu đối với thiết kế:

* Khối IntegralImage có giao diện ghép nối tới CPU sao cho CPU có thể thiết lập địa chỉ cơ sở của hình ảnh gốc, kích thước N×M (5×5 - 256×256)của hình ảnh, địa chỉ cơ sở của vùng nhớ lưu hình ảnh tích phân trả về
* CPU kích hoạt quá trình tính toán của khối IntegralImage bằng các đặt tín hiệu Start = ‘1’.
* Sau khi quá trình tính hình ảnh tích phân hoàn thành, khối IntegralImage sẽ báo cho CPU biết bằng cách đặt tín hiệu Done = ‘1’;



Hình 2. Giao diện ghép nối I/O.

## Định nghĩa giao diện vào/ra

Mô đun này sẽ có các giá trị generic là :

+ Input\_width : độ dài đầu vào của kích thước ( mặc định là 9 bit ).

+ Data\_width : độ dài lưu trữ của bộ nhớ ( mặc định là 8 bit hay 1 ô nhớ lưu trữ 8 bit) .

+ Address\_width : độ dài địa chỉ của bộ nhớ ( mặc định là 32 bit) .

Bảng 1: Mô tả các tín hiệu vào ra.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TT** | **Port** | **Direction** | **Width** | **Meaning** |
| 1 | clr | IN | 1 | Tín hiệu clear hay reset |
| 2 | clk | IN | 1 | Tín hiệu clock |
| 3 | start | IN | 1 | Tín hiệu bắt đầu |
| 4 | done | OUT | 1 | Tín hiệu thông báo thực hiện xong |
| 5 | m\_i | IN | 9 (input\_width) | Số hàng của ma trận gốc |
| 6 | n\_i | IN | 9 (input\_width) | Số cột của ma trận gốc |
| 7 | address\_src\_i | IN | 32 (address\_width) | Địa chỉ cơ sở của ma trận gốc |
| 8 | address\_des\_i | IN | 32 (address\_width) | Địa chỉ cơ sở để lưu ma trận tích phân |
| 9 | read\_en | OUT | 1 | Tín hiệu yêu cầu đọc từ bộ nhớ |
| 10 | write\_en | OUT | 1 | Tín hiệu yêu cầu viết vào bộ nhớ |
| 11 | data\_read | IN | 8 (data\_width) | Dữ liệu đọc được từ ô nhớ |
| 12 | data\_write | OUT | 8 (data\_width) | Dữ liệu viết vào 1 ô nhớ |
| 13 | address\_read | OUT | 32 (address\_width) | Địa chỉ để đọc dữ liệu từ bộ nhớ |
| 14 | address\_write | OUT | 32 (address\_width) | Địa chỉ để viết dữ liệu vào bộ nhớ |

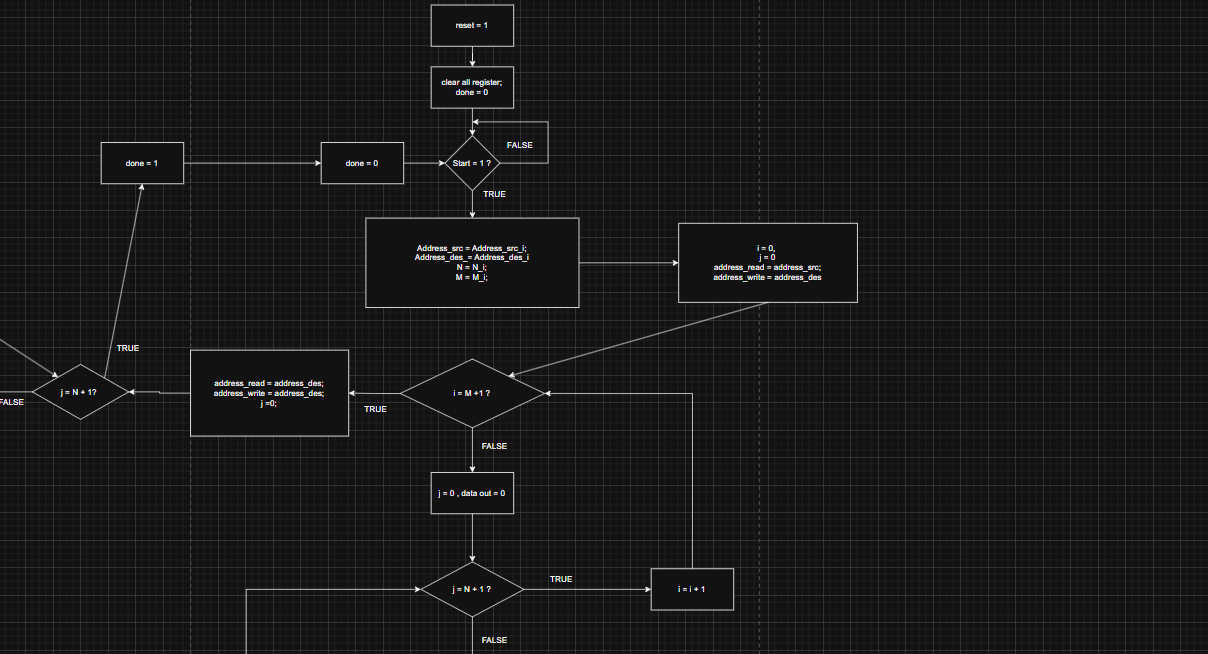
# Thuật toán

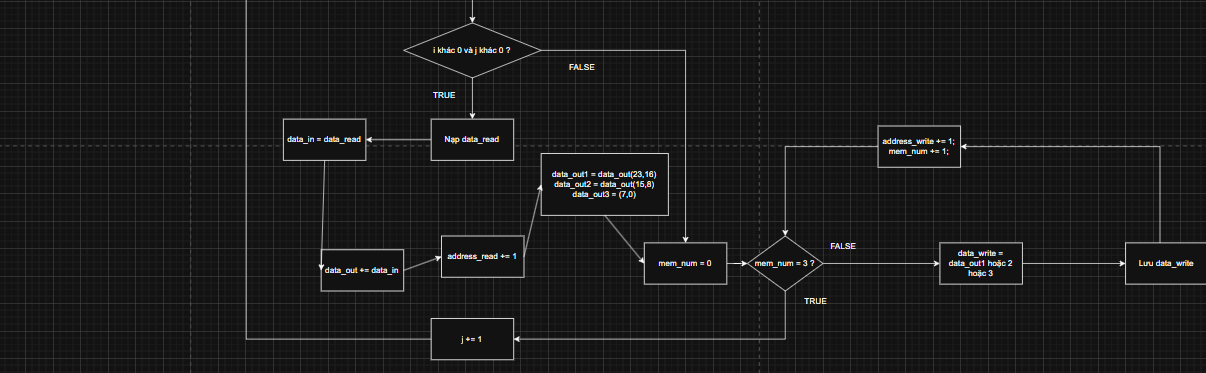
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | input m, n : 9 bit  input address\_des, address\_src : 32 bit  var i, j : 9 bit  External Memory: M[2 ^ 32][8]  var address\_read, address\_write : 32 bit  var data\_out, data\_in : 32 bit  var data\_out1, data\_out2, data\_out3, data\_out4 : 8 bit |
| 8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | Beginning  --GIAI ĐOẠN 1  wait for Start = ‘1’  Done = ‘0’  address\_read = address\_src ;  address\_write = address\_des ;  for i = 0 to m do  data\_out = 0 ;  for j = 0 to n do  if (i neq 0 and j neq 0) then  data\_in = X « 000000 » & M[address\_read] ;  data\_out = data\_out + data\_in ;  address\_read = address\_read + 1 ;  end if  data\_out1 = data\_out(31 downto 24) ;  data\_out2 = data\_out(23 downto 16) ;  data\_out3 = data\_out(15 downto 8) ;  data\_out4 = data\_out(7 downto 0) ;  M[address\_write + 0] = data\_out1 ;  M[address\_write + 1] = data\_out2 ;  M[address\_write + 2] = data\_out3 ;  M[address\_write + 3] = data\_out4 ;  address\_write = address\_write + 4 ;  end for ;  end for ;  --GIAI ĐOẠN 2  address\_write = address\_des ;  address\_read = address\_des ;  for j = 0 to n do  data\_out = 0 ;  address\_write = address\_des + 4 \* j ;  address\_read = address\_des + 4 \* j  for i = 0 to m do  data\_in = M[address\_read] & M[address\_read + 1] & M[address\_read +2] & M[address\_read +3] ;  data\_out = data\_out + data\_in ;  data\_out1 = data\_out(31 downto 24) ;  data\_out2 = data\_out(23 downto 16) ;  data\_out3 = data\_out(15 downto 8) ;  data\_out4 = data\_out(7 downto 0) ;  M[address\_write + 0] = data\_out1 ;  M[address\_write + 1] = data\_out2 ;  M[address\_write + 2] = data\_out3 ;  M[address\_write + 3] = data\_out4 ;  address\_write = address\_write + 4 \* n ;  address\_read = address\_read + 4 \* n ;  end for ;  end for ;  Done = ‘1’ ;  End ; |

# Thiết kế mức RTL

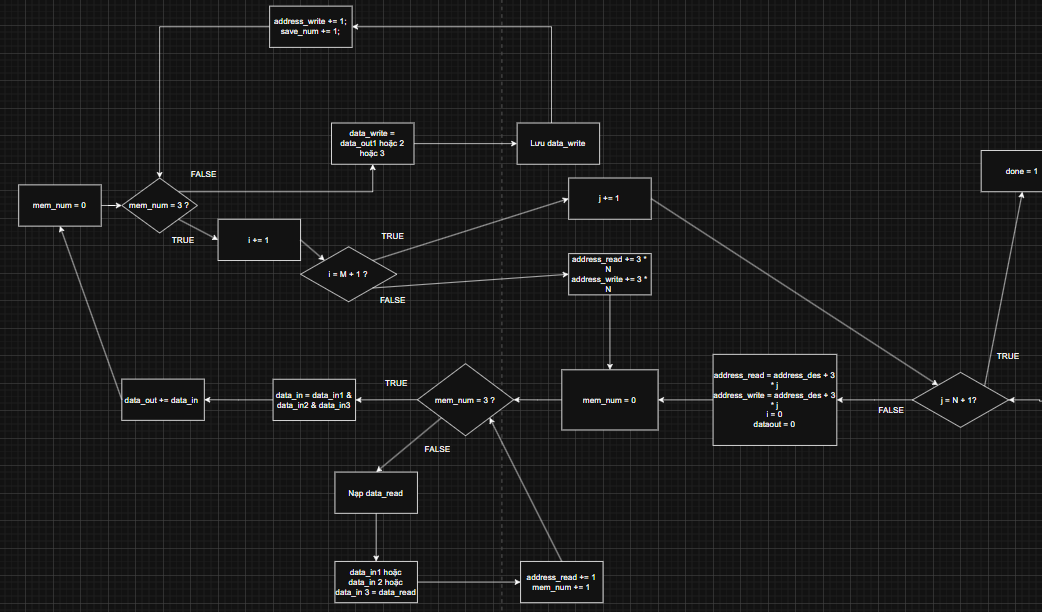
## Mô hình máy FSMD

Giai đoạn 1

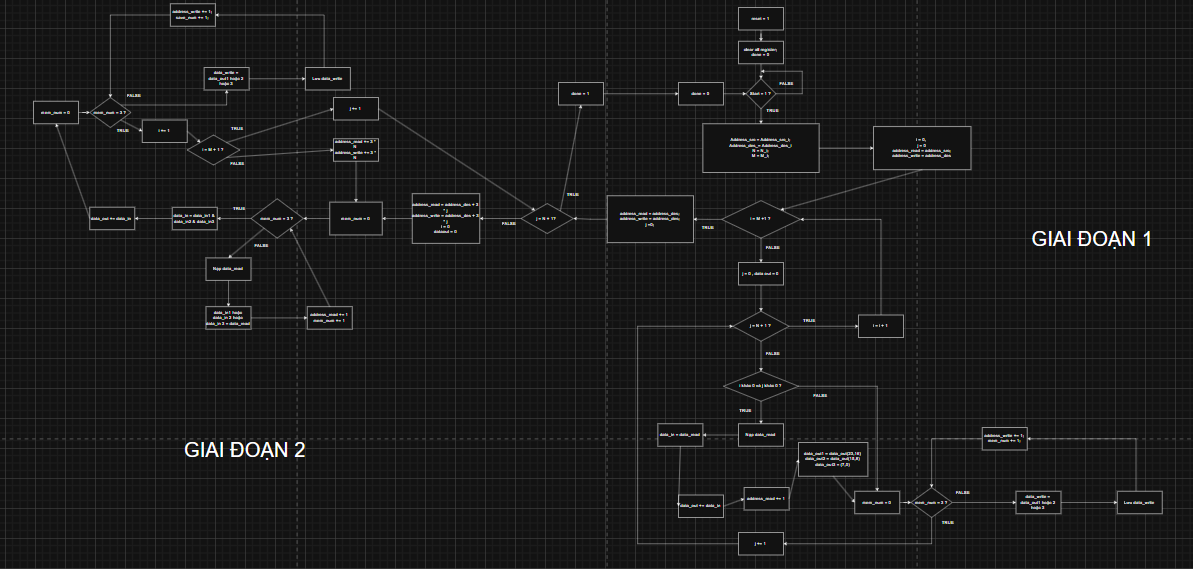




Giai đoạn 2

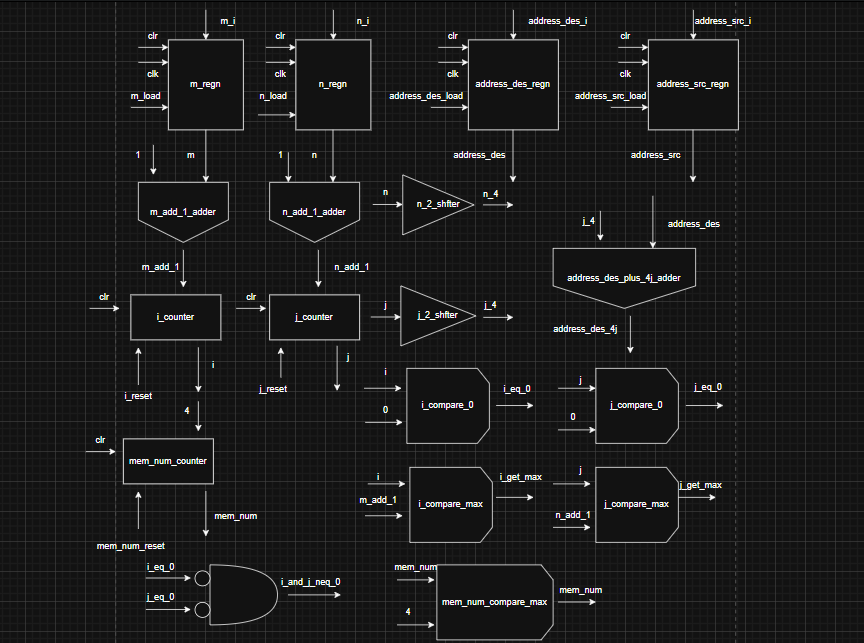


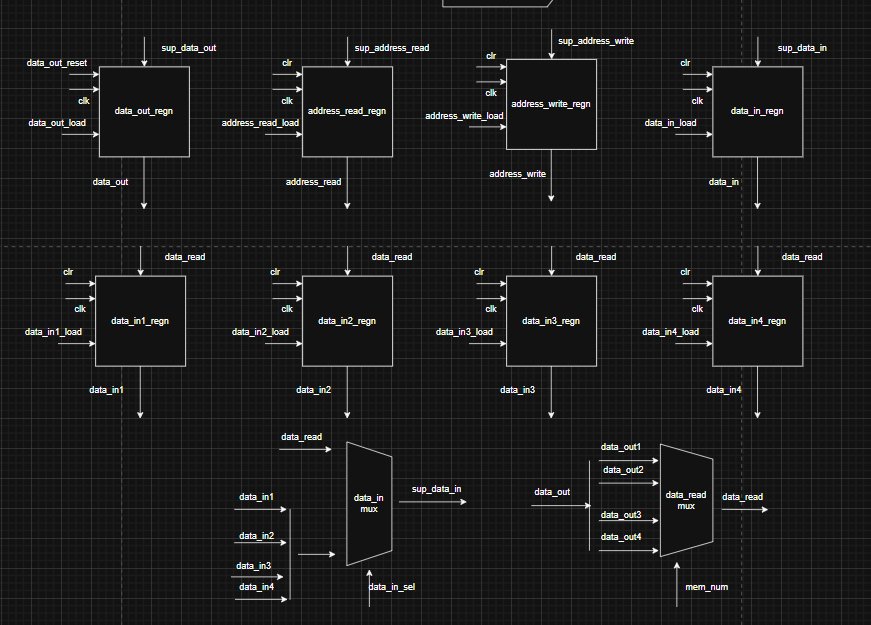
Sơ đồ tổng

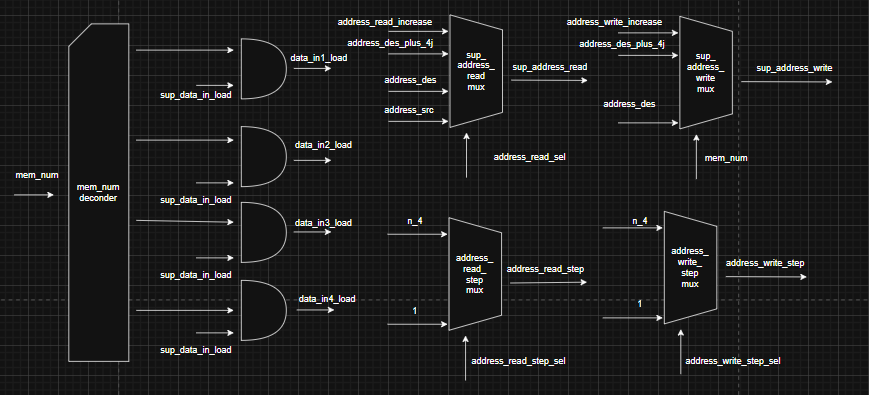


Hình 3: Mô hình máy FSMD.

## Đơn vị xử lý dữ liệu (Datapath)



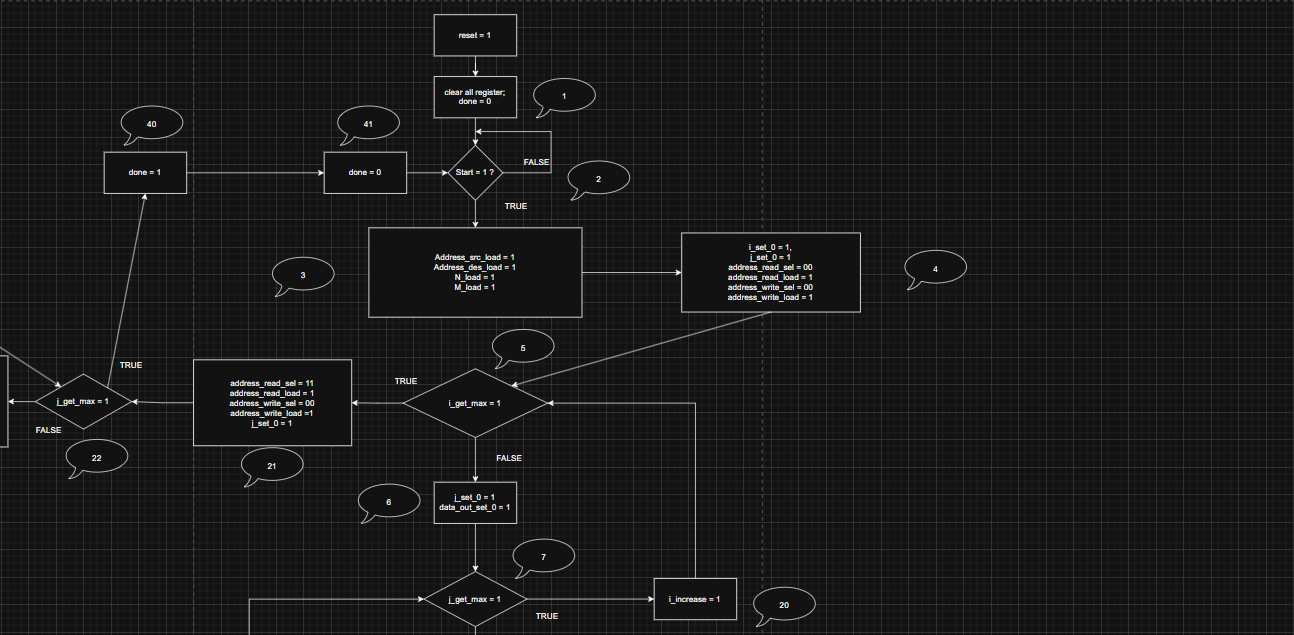


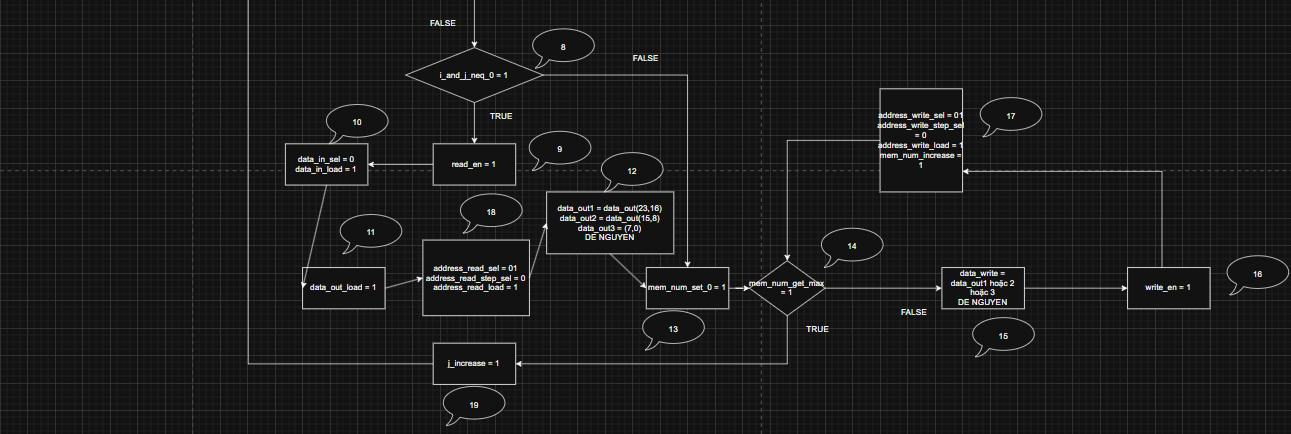


Hình 4: Cấu trúc của đơn vị xử lý dữ liệu Datapath.

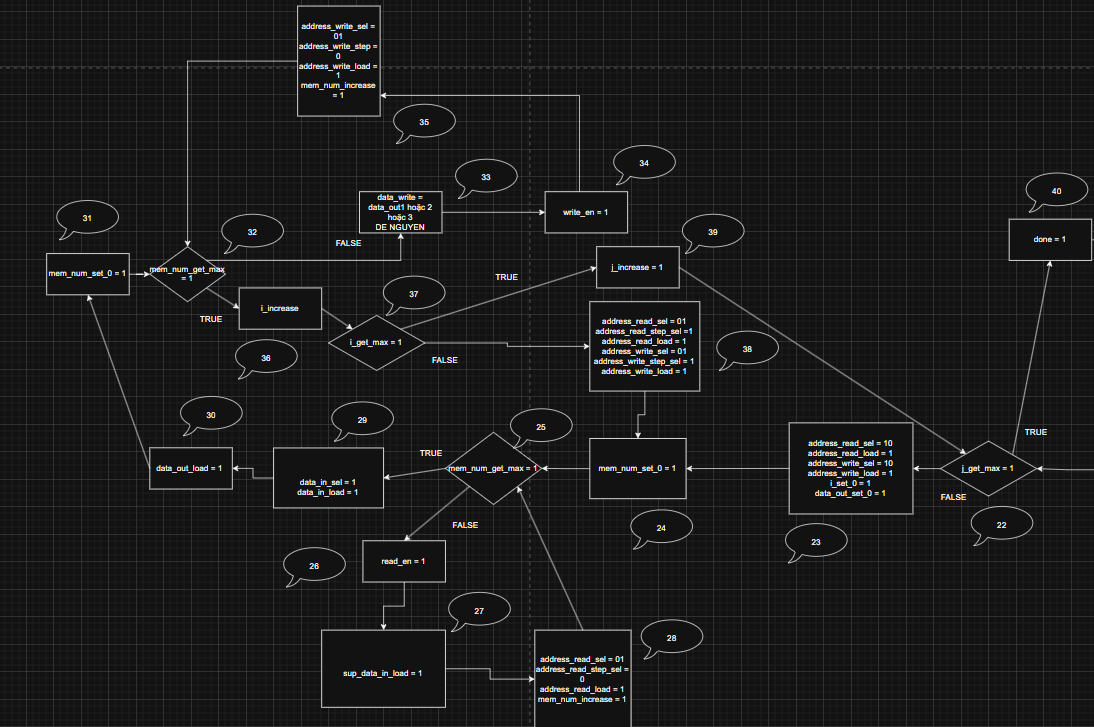
## Đơn vị điều khiển (Control Unit)

Giai đoạn 1

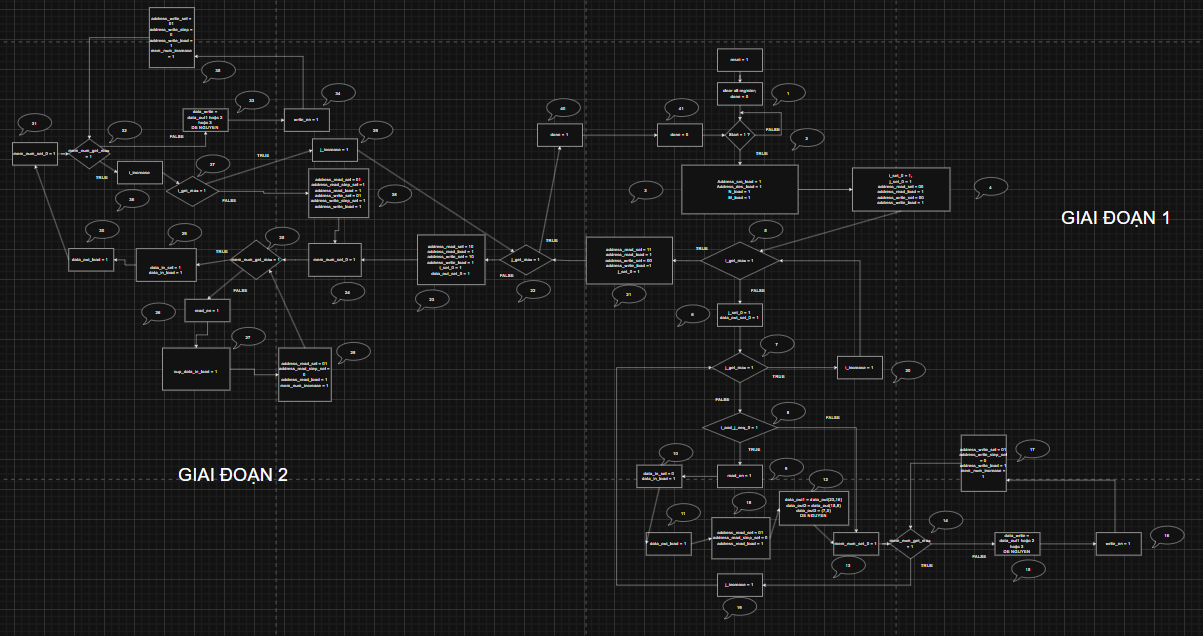




Giai đoạn 2:

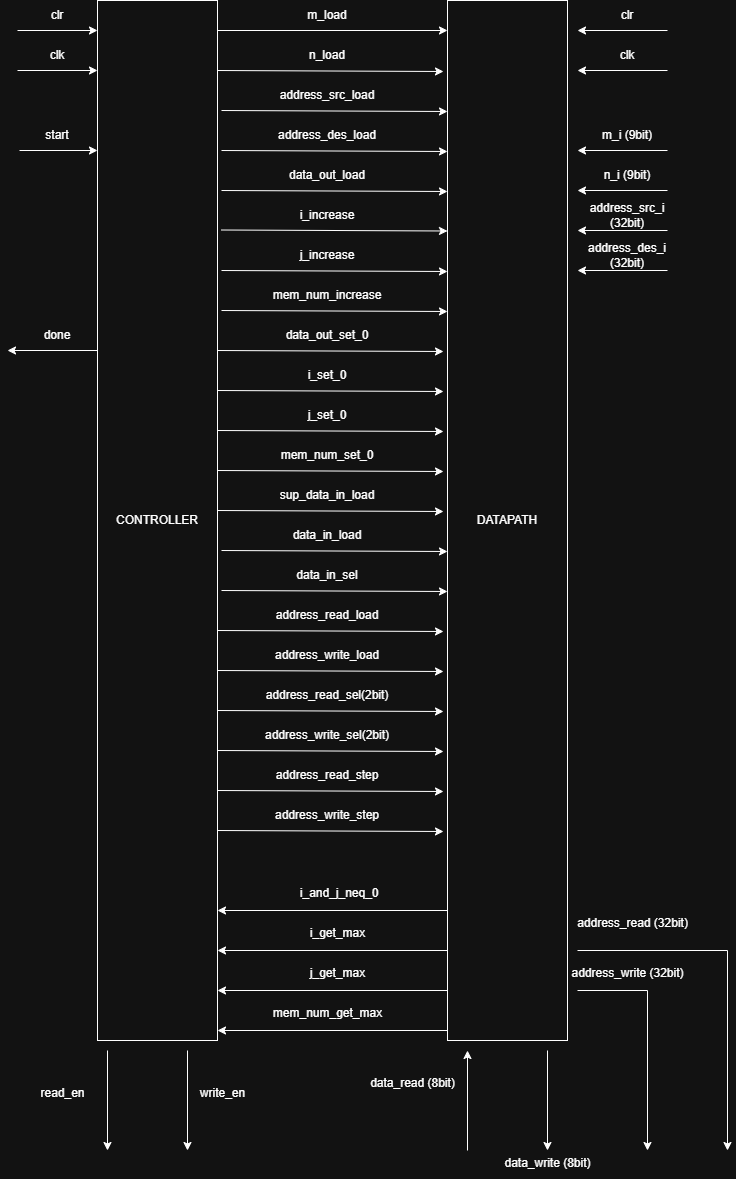


Sơ đồ tổng



Hình 5: Máy FSM của đơn vị điều khiển.

## Sơ đồ khối tổng thể



Hình 6: Sơ đồ khối tổng thể của bộ tính ảnh tích phân.

# Mô hình hóa bằng VHDL

# Mô phỏng/thực thi và đánh giá

## Chạy testbench trên ModelSim

Chạy testbench với một bộ nhớ đã được khởi tạo sẵn giá trị của 40 ô nhớ đầu như sau:

Bảng 2: Dữ liệu 40 địa chỉ đầu

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 121 | 226 | 181 | 226 | 149 | 174 | 197 | 62 |
| 35 | 4 | 245 | 179 | 112 | 41 | 140 | 230 |
| 163 | 142 | 119 | 234 | 134 | 104 | 206 | 90 |
| 85 | 214 | 90 | 124 | 88 | 27 | 70 | 190 |
| 92 | 30 | 195 | 219 | 181 | 255 | 52 | 204 |

Thực hiện chạy với address\_src = 0 và address\_des = 45 với:

* MxN = 3x3. Kết quả thu được chuỗi dãy số có giá trị như sau:

( 0 0 0 0

0 121 347 528

0 347 722 1077

0 544 981 1371 ).

Kết quả này ta thu được sau 1066 clock (tính từ lúc start = ‘1’ đến lúc done =’1’).

* MxN = 5x5. Kết quả thu được chuỗi dãy số có giá trị như sau:

( 0 0 0 0 0 0

0 121 347 528 754 903

0 295 718 961 1222 1375

0 540 1142 1497 1799 2092

0 770 1535 2032 2453 2980

0 904 1773 2476 2987 3599 ).

Kết quả này ta thu được sau 2399 clock (tính từ lúc start = ‘1’ đến lúc done =’1’).

* MxN = 5x7. Kết quả thu được chuỗi dãy số có giá trị như sau:

( 0 0 0 0 0 0 0 0

0 121 347 528 754 903 1077 1274

0 183 444 629 1100 1428 1714 1952

0 323 814 1162 1775 2222 2742 3114

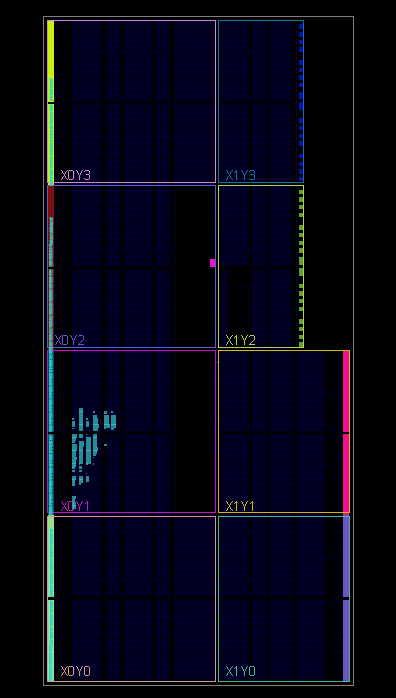
0 427 1124 1562 2260 2921 3531 4027

0 515 1239 1747 2635 3388 4028 4719 ).

Kết quả này ta thu được sau 3197 clock (tính từ lúc start = ‘1’ đến lúc done = ’1’).

## Tích hợp trên Vivaodo

Sau khi thực hiện Synthesis và Implement trên Vivado, mạch sẽ được tích hợp tại vị trí như sau:



Hình 7: Vị trí được tích hợp trên mạch (Vùng màu xanh dương)

Dựa vào timing report, thời gian dư thừa trong 1 chu kì khi ta cho chu kì clock là 10 ns là: 3,9837 ns. Vì vậy, bộ mô đun có thể hoạt động tốt khi chúng ta cho chu kì clock xuống nhỏ nhất là 6 ns hay tối đa là tần số 166 MHz.

# Kết luận

Mô đun đã có thể thực hiện được nhiệm vụ yêu cầu, tuy nhiên mô đun này có thuật toán chỉ phù hợp cho bộ nhớ 8 bit, khi độ dài bộ nhớ thay đổi phải điều chỉnh lại cho phù hợp.

Appendix A: Reference

1. https://www.mathworks.com/help/images/ref/integralimage.html

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 phép chuyển đổi hình ảnh. 5](#_Toc197372584)

[Hình 2. Giao diện ghép nối I/O. 7](#_Toc197372585)

[Hình 3: Mô hình máy FSMD. 12](#_Toc197372586)

[Hình 4: Cấu trúc của đơn vị xử lý dữ liệu Datapath. 14](#_Toc197372587)

[Hình 5: Máy FSM của đơn vị điều khiển. 16](#_Toc197372588)

[Hình 6: Sơ đồ khối tổng thể của bộ tính ảnh tích phân. 17](#_Toc197372589)

[Hình 7: Vị trí được tích hợp trên mạch (Vùng màu xanh dương)……………………………...19](#Hinh7)

List of Tables

[Bảng 1: Mô tả các tín hiệu vào ra. 7](#_Toc197372593)

[Bảng 2: Dữ liệu 40 địa chỉ đầu…………………………………………………………….….18](#Bang2)

References