

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN ĐIỆN TỬ-VIỄN THÔNG



====o0o====

BÁO CÁO THỰC TẬP KỸ THUẬT

**NHẬN DIỆN CHỮ CÁI SỬ DỤNG MATLAB VÀ
THỰC HIỆN MẠNG NEURAL TRÊN FPGA**

ESRC Lab - C9 420

GVHD : PGS.TS Phạm Ngọc Nam

ThS Nguyễn Thị Kim Thoa

Sinh viên thực hiện : Nguyễn Minh Hiếu

Mã số sinh viên : 20151336

Lớp : Điện tử 03 – K60

Hà Nội, tháng 8 năm 2018

MỤC LỤC

A. LỜI NÓI ĐẦU.....	5
B. NỘI DUNG	6
Chương 1. Giới thiệu chức năng, nhiệm vụ, cơ cấu tổ chức năng của đơn vị tiếp nhận	6
1.1. Trung tâm nghiên cứu và phát triển Điện tử - Viễn thông	6
1.2 Phòng thí nghiệm Hệ thống nhúng và Tính toán cấu hình lại (ESRC Lab)	8
Chương 2 . Nội dung thực tập.....	9
2.1 Giới thiệu mạng neural nhân tạo.....	9
2.2 Cấu trúc một neuron	11
2.3 Thuật toán lan truyền ngược (backpropagation algorithm)	12
2.4 Nhận diện chữ cái sử dụng Matlab	14
2.5 Khái quát về FPGA, ngôn ngữ Verilog và mạng neural 2:3:2	18
2.5.1 Giới thiệu chung về FPGA	18
2.5.2 Giới thiệu ngôn ngữ mô tả phần cứng Verilog.....	20
2.5.3 Tổng quan mạng neural 2:3:2.....	21
2.6 Thiết một neuron.....	22
2.6.1 Module mạch nhân	22
2.6.2 Module hàm kích hoạt.....	24
2.6.3 Mô phỏng một neuron	25
2.7 Thực hiện mạng neural 2:3:2	26
Chương 3. Nhận xét, đề xuất.....	27
3.1 Ưu điểm	27

3.2 Nhược điểm	27
3.3 Đề xuất.....	27
C. KẾT LUẬN	28
Tài liệu tham khảo.....	29
D. PHỤ LỤC	30

DANH MỤC HÌNH VẼ

Hình 2.1 Tế bào thần kinh sinh học	9
Hình 2.2 Cấu trúc mạng neural.....	10
Hình 2.3 Một neuron riêng lẻ	11
Hình 2.4 Minh họa 6 bước đầu nhận diện chữ cái bằng Matlab.....	15
Hình 2.5 Kết quả sau khi thực hiện bước 8.....	16
Hình 2.6 Công cụ nftool trong Matlab.....	17
Hình 2.7 Kiến trúc một chip FPGA	19
Hình 2.8 Mạng neural 2:3:2	21
Hình 2.9 Mạng neuron riêng lẻ.....	21
Hình 2.10 Lưu đồ thuật toán giải thuật Booth	22
Hình 2.11 Sơ đồ ASM giải thuật Booth	23
Hình 2.12 Tổng quát mạch nhân	23
Hình 2.13 Hàm sigmoid.....	24
Hình 2.14 Tổng quát mạch hàm kích hoạt.....	24
Hình 2.15 Mô phỏng một neuron	25
Hình 2.16 Cấu trúc cụ thể mạng neural 2:3:2	26
Hình 2.17 Mô phỏng mạng neural 2:3:2	26

A. LỜI NÓI ĐẦU

Học phần Thực tập kĩ thuật nằm trong chương trình đào tạo của viện Điện tử - Viễn thông, giúp sinh viên tiếp cận môi trường, có thêm kinh nghiệm làm việc thực tế ở một công ty thuộc lĩnh vực Điện tử - Viễn thông hoặc tham gia vào một phòng nghiên cứu cả trong và ngoài trường

Trong học kì hè 20173, em đã tham gia thực tập tại Phòng thí nghiệm Hệ thống và Tính toán cấu hình lại (Embedded System and Reconfigurable Compute Laboratory - ESRC Lab), thuộc Trung tâm Nghiên cứu và Phát triển, viện Điện tử Viễn Thông, trường Đại học Bách Khoa Hà Nội. Em đã được các anh chị khóa trên hướng dẫn nghiên cứu đề tài “Nhận diện chữ cái sử dụng Matlab và thực hiện mạng neural trên FPGA”, với mục tiêu làm quen với trí thông minh nhân tạo (AI – Artificial Intelligence) và luyện tập ngôn ngữ mô tả phần cứng Verilog. Vì lần đầu biết tới mạng neural nhân tạo (ANNs – Artificial Neural Networks) và ngôn ngữ Verilog nên đề tài không được hoàn chỉnh, bọn em dự định sẽ tiếp tục phát triển trong năm học tới.

Em xin cảm ơn Ban giám hiệu trường Đại học Bách Khoa Hà Nội và Ban lãnh đạo viện Điện tử - Viễn thông đã tạo điều kiện cho chúng em tham gia thực tập, cũng như thầy Phạm Ngọc Nam, các anh chị, các bạn trong Lab đã đồng hành và hỗ trợ em em hoàn thành tốt học phần thực tập.

B. NỘI DUNG

Chương 1. Giới thiệu chức năng, nhiệm vụ, cơ cấu tổ chức năng của đơn vị tiếp nhận

1.1. Trung tâm nghiên cứu và phát triển Điện tử - Viễn thông

Địa chỉ: Phòng 618, Thư viện Tạ Quang Bửu, Đại học Bách khoa Hà Nội, số 1 Đại Cồ Việt, quận Hai Bà Trưng, Hà Nội.

Trung tâm bao gồm 8 phòng thí nghiệm (PTN) chính:

- PTN Thiết kế vi mạch (IC Design Lab)
- PTN Thông tin vô tuyến (Wireless Communications Lab)
- PTN Mạng thế hệ mới (Future Internet Lab)
- PTN Quang dẫn và siêu cao tần (Microwaves and Photonics Lab)
- PTN Hệ thống nhúng và tính toán khả cấu hình (Embedded Systems and Reconfigurable Computing Lab)
- PTN Xử lý tín hiệu và thông tin (Signal and Information Processing Laboratory)
- PTN Đa phương tiện (Multimedia Lab)
- PTN Kỹ thuật điện tử Y sinh (Bio-medical Electronics Lab)

Các lĩnh vực nghiên cứu chính bao gồm:

- **Thông tin vô tuyến:** Các hướng nghiên cứu liên quan đến thiết bị di động thông minh; Các kỹ thuật xử lý tín hiệu tiên tiến trong thông tin di động như OFDM/OFDMA, MIMO-OFDM, CDMA; Thiết kế và mô phỏng mạng thông tin di động ở dạng mô hình; Nghiên cứu và thiết kế hệ thống thông tin dưới nước.
- **Mạng thế hệ mới và dịch vụ:** tập trung vào các chủ đề nghiên cứu liên quan đến mạng cố định và di động thế hệ mới, mạng Internet và các dịch vụ mạng như: Công nghệ ảo hóa, quản lý tài nguyên mạng, mạng tiết kiệm năng lượng, đảm bảo chất lượng dịch vụ và chất lượng trải nghiệm trong mạng; Mạng cảm biến vô tuyến và ứng dụng của mạng cảm biến trong môi trường, xây dựng, giao thông

vận tải; Hệ thống thông tin không dây đa chặng phục vụ các hoạt động hiện trường và tình huống khẩn cấp; Mạng đồng đẳng và mạng xếp chồng; Các công nghệ sử dụng trong mạng di động: tối ưu hóa tài nguyên mạng, quản lý di động, điều khiển chuyển giao, đảm bảo chất lượng dịch vụ và chất lượng trải nghiệm di động; Công nghệ theo định hướng dịch vụ giá trị gia tăng và môi trường thông minh.

- ***Quang dẫn và siêu cao tần:*** tập trung vào các chủ đề nghiên cứu liên quan đến công nghệ quang và siêu cao tần như: Nghiên cứu, tính toán và thiết kế các mạch quang học có kích thước nano: áp dụng cho các hệ thống thông tin quang, các phần tử tích cực, và đặc biệt là các thiết bị truyền dẫn năng lượng quang học kích thước nano; Phân tích, thiết kế và chế tạo các loại antenna thế hệ mới: nghiên cứu các loại antenna đa băng, băng rộng, kích thước nhỏ, hiệu suất bức xạ cao ứng dụng trong các hệ thống thông tin vô tuyến, các hệ thống vô tuyến cảm biến môi trường; Phân tích và thiết kế đường truyền sóng vô tuyến: phân tích và thiết kế đường truyền, phương thức truyền sóng thích hợp cho các hệ thống thông tin.
- ***Hệ thống nhúng và tính toán khả cấu hình:*** tập trung vào các chủ đề nghiên cứu liên quan đến thiết kế các hệ thống vi mạch khả trình và ứng dụng; Các ứng dụng của hệ thống nhúng trong viễn thông; Xây dựng và đăng ký sở hữu trí tuệ các bộ thư viện phần cứng phục vụ cho việc thiết kế các thiết bị di động như công nghệ CDMA, OFDM, .v.v.; Thiết kế các vi mạch mã hóa, giải mã video (MPEG4, H.264), mã hóa, giải mã ảnh (JPEG) và giải mã, mã hóa âm thanh (MP3, AAC); Các ứng dụng của hệ thống nhúng trong các hệ điều khiển và điện tử ứng dụng.
- ***Xử lý tín hiệu:*** tập trung vào các chủ đề nghiên cứu liên quan đến xử lý tín hiệu trong viễn thông, trong đa phương tiện, trong y sinh như: Lý thuyết xử lý tín hiệu phi tuyến và các ứng dụng trong viễn thông, bảo mật; Thông tin hỗn loạn, laser hỗn loạn, mạch hỗn loạn và mã hóa biểu tượng; Mạng thần kinh tế bào và các ứng dụng; Xử lý ảnh và ứng dụng của xử lý ảnh trong y tế, giao thông vận tải, bảo mật, ...

- **Thiết kế vi mạch:** thiết kế các IC số và tương tự, phát triển các công cụ kiểm tra vi mạch, IC cao tần và RFID.
- **Kỹ thuật y sinh:** tập trung vào các chủ đề: Xử lý thông tin y tế; Thiết kế chế tạo các thiết bị điện tử y sinh ứng dụng tại Việt Nam; Xây dựng và phát triển hệ thống mạng thông tin ứng dụng trong y tế.

1.2 Phòng thí nghiệm Hệ thống nhúng và Tính toán cấu hình lại (ESRC Lab)

Địa chỉ: Phòng 420, nhà C9, Đại học Bách khoa Hà Nội, số 1 Đại Cồ Việt, quận Hai Bà Trưng, Hà Nội.

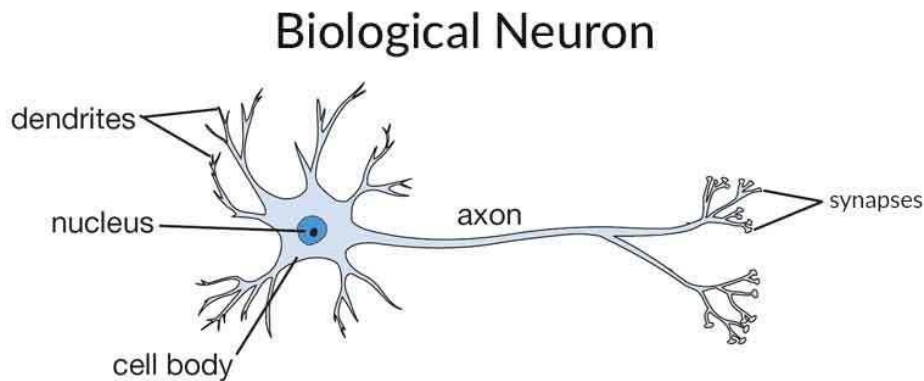
Cơ cấu tổ chức:

- Chủ nhiệm Lab: PGS.TS Phạm Ngọc Nam – Viện phó Viện Điện tử - Viễn thông, Phó trưởng bộ môn Điện tử và Kỹ thuật máy tính
- Nghiên cứu sinh: 2.
- Trưởng lab: Hoàng Huyền Trang – Điện tử 01 K59
- Số lượng sinh viên đang nghiên cứu và học tập: 30.
- Hướng nghiên cứu:
 - Thiết kế hệ thống nhúng.
 - Tính toán cấu hình lại.
 - Thực hiện và tối ưu các thuật toán xử lý tín hiệu trên FPGA.
 - Xử lý hình ảnh
- Các đề tài đang thực hiện:
 - HTTP streaming
 - Virtual Reality
 - Openflow
 - HTTP streaming over SDN
 - Power aware NetFPGA10G

Chương 2 . Nội dung thực tập

2.1 Giới thiệu mạng neural nhân tạo

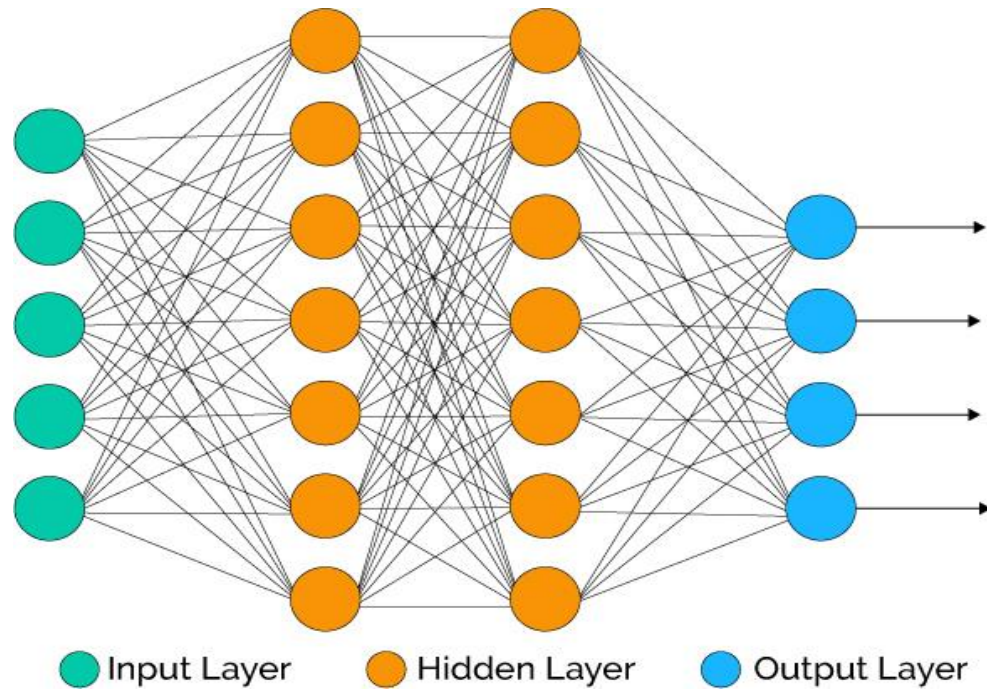
Thuật ngữ “neural” bắt nguồn từ đơn vị chức năng cơ bản của tế bào thần kinh (neuron) trong não người. Mạng neural sinh học là một mạng lưới liên kết hàng tỷ tế bào thần kinh với hàng nghìn tỷ kết nối giữa chúng. Hình 2.1 mô tả một tế bào thần kinh sinh học (biological neuron).



Hình 2.1 Tế bào thần kinh sinh học

Mạng neural nhân tạo là một hệ thống tính toán được xây dựng dựa trên mạng neural sinh học của não người. Mạng neural nhân tạo có rất nhiều ứng dụng trong thực tế: điều khiển hệ thống, chơi trò chơi điện tử, đưa ra quyết định, nhận diện hình mẫu, khuôn mặt, chẩn đoán y tế, khai thác dữ liệu giao dịch tự động,...

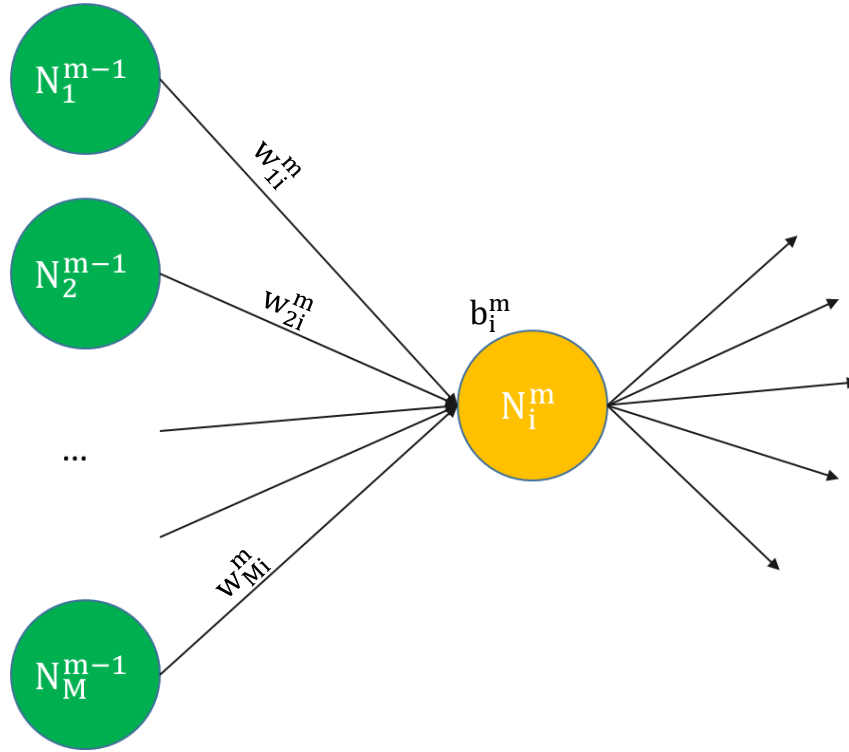
Mạng neural nhân tạo gồm một lớp vào (input layer), một hoặc nhiều lớp ẩn (hidden layer) và một lớp ra (output layer). Một lớp bao gồm nhiều neuron, các neuron ở một lớp sẽ được kết nối đầy đủ với tất cả các neuron ở lớp ngay trước thông qua hàm truyền (propagation function) với trọng số (weight) tương ứng (ngoài ra có thể có thêm phân độ lệch (bias)), sau đó được xử lý qua hàm kích hoạt (activation function) trước khi tiếp tục kết nối với tất cả các neuron lớp sau. Hình 2.2 mô tả một mạng neural.



Hình 2.2 Cấu trúc mạng neural

Mạng neural nhân tạo học bằng các điều chỉnh trong số và độ lệch, lặp đi lặp lại đến khi mang lại kết quả mong muốn. Mạng được huấn luyện trước bằng các sử dụng bộ quy tắc còn gọi là thuật toán học tập. Các thuật toán học tập phổ biến bao gồm: giảm gradient (gradient descent), lan truyền ngược (back propagation), luật Hebb (Hebb rule), luật Hopfield (Hopfield law), thuật toán trung bình tối thiểu (LMS - least mean square), học tập cạnh tranh (competitive learning). Quá trình học gồm nhiều kiểu học khác nhau: học có giám sát (supervised learning), học không giám sát (unsupervised learning), học tăng cường (reinforcement learning), học ngoại tuyến (offline learning), học trực tuyến (online learning).

2.2 Cấu trúc một neuron



Hình 2.3 Một neuron riêng lẻ

Hình 2.3 mô tả một neuron thứ i thuộc lớp thứ m , được kết nối với các neuron từ thứ 1 đến thứ M của lớp $m-1$ bằng hàm truyền như sau:

$$z_i^m = \sum_{j=1}^M a_j^{m-1} * w_{ji}^m + b_i^m \quad (1)$$

Trong đó z_i^m là đầu vào của neuron thứ i thuộc lớp thứ m , a_j^{m-1} là đầu ra của neuron thứ j thuộc lớp thứ $m-1$, w_{ji}^m là trọng số kết nối từ neuron thứ j thuộc lớp thứ $m-1$ đến neuron thứ i thuộc lớp thứ m , b_i^m là bias của neuron thứ i thuộc lớp thứ m

Mối quan hệ giữa đầu ra và đầu vào của 1 neuron được thể hiện qua hàm kích hoạt. Trong đề tài này sử dụng hàm sigmoid làm hàm kích hoạt:

$$a_i^m = \frac{1}{1+e^{z_i^m}} \quad (2)$$

Sau đó đầu ra của neuron lại được nối với các neuron lớp kế tiếp.

2.3 Thuật toán lan truyền ngược (backpropagation algorithm)

Các giá trị weight và bias sẽ được gán các giá trị ngẫu nhiên khác 0. Sau đó các vector trong tập huấn luyện sẽ lần lượt được đưa vào mạng để tính toán ra output và từ đó tính toán được hàm giá (cost function) theo công thức:

$$C = \sum_{i=1}^n C_i \quad (3)$$

Trong đó:

$$C_i = \frac{1}{2} \sum (t_j^i - y_j^i)^2 \quad (4)$$

Với i là chỉ số tương ứng với mẫu huấn luyện thứ i .

Sau khi tính toán xong giá trị của cost function sẽ là giai đoạn lan truyền ngược (back propagation). Ở giai đoạn này, các đạo hàm riêng của hàm Cost function theo các w và b sẽ được tính toán.

Ở lớp output, giá trị unit error sẽ được tính theo công thức:

$$\partial_j^h[p] = (y_j^p - t_j^p) \times f_{\text{act}}'(\text{net}_j^h) \quad (5)$$

Trong đó p là thứ tự của neuron output, y là activation của neuron output. Qua đó ta tính được:

$$\frac{\partial C_p}{\partial w_{i,j}^h} = a_i^{h-1} \times \partial_j^h[p] \quad (6)$$

$$\frac{\partial C_p}{\partial b_j^h} = \partial_j^h[p] \quad (7)$$

Đối với lớp hidden:

$$\partial_j^h[p] = f_{\text{act}}'(\text{net}_j^h) \times \sum_i \partial_i^{h+1}[p] \times w_{j,i}^{h+1} \quad (8)$$

Qua đó ta cũng tính được:

$$\frac{\partial C_p}{\partial w_{i,j}^h} = a_i^{h-1} \times \partial_j^h[p] \quad (9)$$

$$\frac{\partial C_p}{\partial b_j^h} = \partial_j^h[p] \quad (10)$$

Giá trị đạo hàm riêng phần sẽ được tính theo công thức:

$$\frac{\partial C}{\partial w_{ij}^h} = \sum_p \frac{\partial C_p}{\partial w_{ij}^h} \quad (11)$$

$$\frac{\partial C}{\partial b_j^h} = \sum_p \frac{\partial C_p}{\partial b_j^h} \quad (12)$$

Đến đây ta tính:

$$\Delta w_{ij}^h = -\mu \frac{\partial C}{\partial w_{ij}^h} \quad (13)$$

$$\Delta b_j^h = -\mu \frac{\partial C}{\partial b_j^h} \quad (14)$$

Trong đó μ là một số thực dương gọi là learning rate. Một điểm cực tiểu của hàm số tính chất, đạo hàm bên trái âm (hàm trong lân cận phía trái điểm này nghịch biến), đạo hàm bên phải dương (hàm trong lân cận phía phải điểm này đồng biến). Do đó nếu lỡ chọn tham số làm cho đạo hàm âm thì phải tăng tham số lên để nó sát về điểm cực tiểu, ngược lại nếu lỡ chọn tham số cho đạo hàm dương thì ta phải giảm tham số đi để nó sát về điểm cực tiểu, suy ra μ phải là số dương để đảm bảo ta luôn đi ngược chiều với đạo hàm, có thể chọn càng lặp càng nhỏ để tìm điểm sát với cực tiểu hơn.

Giá trị của các w và b sẽ được cập nhật theo công thức:

$$(\text{new})w_{ij}^h = (\text{old})w_{ij}^h + \Delta w_{ij}^h \quad (15)$$

$$(\text{new})b_j^h = (\text{old})b_j^h + \Delta b_j^h \quad (16)$$

Ta lặp đi lặp lại các bước trên đến khi nào các đạo hàm riêng sát với 0 thì dừng.

Thuật toán lan truyền ngược hiểu đơn giản là ban đầu tính unit's error của các neuron lớp đầu ra trước rồi tính unit's error của các lớp trước theo các lớp sau. Sau đó tính các đạo hàm theo unit's error. Giá trị μ trong biểu thức (13), (14) ảnh hưởng khá nhiều đến quá trình học của thuật toán. Thường giá trị này sẽ chọn khoảng từ 0 đến 1.

2.4 Nhận diện chữ cái sử dụng Matlab

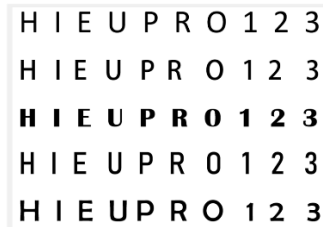
Matlab (Matrix Laboratory), là một bộ phần mềm toán học của hãng Mathworks để lập trình, tính toán số và có tính trực quan rất cao. Matlab làm việc chủ yếu với ma trận, với chuỗi kí tự Matlab cũng xem là một dãy các kí tự hay là dãy mã số của các ký tự. Matlab dùng để giải quyết các bài toán về giải tích số, xử lý tín hiệu số, xử lý đồ họa,... mà không phải lập trình cở điển. Hiện nay, Matlab có đến hàng ngàn lệnh và hàm tiện ích. Ngoài các hàm cài sẵn trong chính ngôn ngữ, Matlab còn có các lệnh và hàm ứng dụng chuyên biệt trong các Toolbox, để mở rộng môi trường Matlab nhằm giải quyết các bài toán thuộc các phạm trù riêng. Các Toolbox khá quan trọng và tiện ích cho người dùng như toán sơ cấp, xử lý tín hiệu số, xử lý ảnh, xử lý âm thanh, ma trận thưa, logic mờ,... Ở phần này ta sử dụng công cụ nftool của Matlab để nhận diện chữ cái.

Để thực hiện chương trình trên Matlab, ta đi theo 8 bước tiền xử lí ảnh sau:

1. Tải ảnh cần nhận diện
2. Chuyển ảnh về mức xám
3. Chuyển ảnh về mức nhị phân
4. Xác định cạnh
5. Tăng gấp đôi cạnh
6. Làm đầy các đối tượng

Hình 2.4 minh họa 6 bước trên khi thực hiện bằng Matlab.

1. Read Image
I=imread(Training.bmp);
imshow(I);



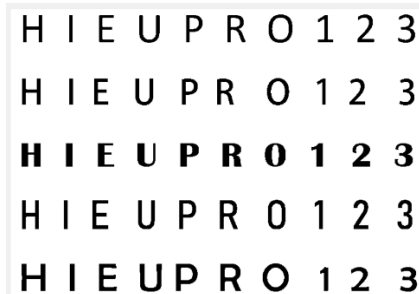
2. Convert to grayscale image

Igray=rgb2gray(I);
imshow(Igray);



3. Convert grayscale image to binary image

Ibw=im2bw(Igray,graythresh(Igray));
imshow(Ibw);



4. Edge detection

Iedge=edge(uint8(Ibw));
imshow(Iedge)



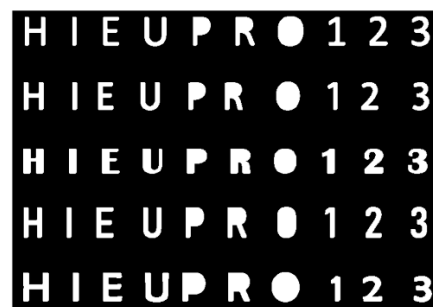
5. Image dilation

se=strel('square',2);
Iedge2=imdilate(Iedge, se);
imshow(Iedge2)



6. Image filling

Ifill= imfill(Iedge2,'holes');
imshow(Ifill)

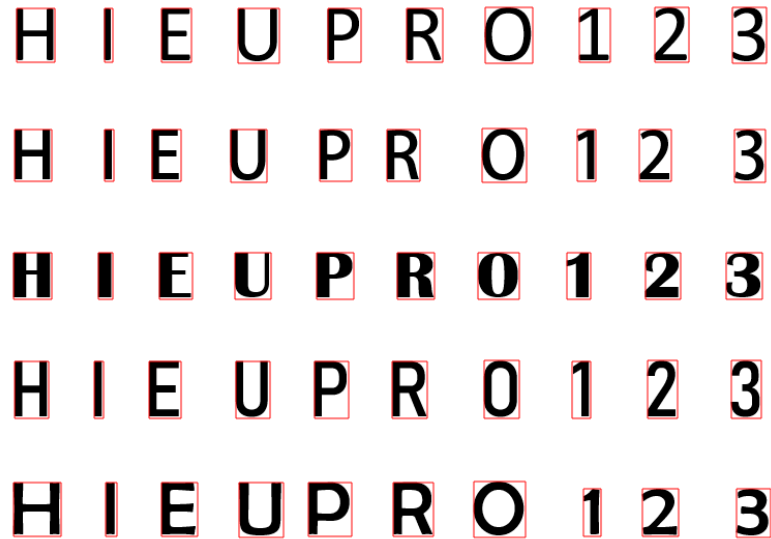


Hình 2.4 Minh họa 6 bước đầu nhận diện chữ cái bằng Matlab

Sau 6 bước trên, ta thực hiện 2 bước quan trọng là

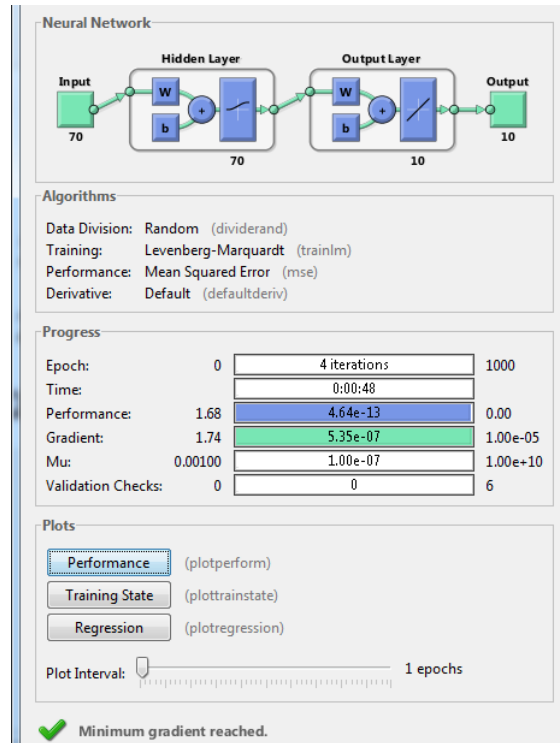
7. Đánh dấu các đối tượng, xác định vị trí các đối tượng
8. Vẽ hình chữ nhật bao quanh các đối tượng dựa trên ảnh gốc

Hình 2.5 là kết quả sau khi thực hiện bước 8.



Hình 2.5 Kết quả sau khi thực hiện bước 8

Trong chương trình này sử dụng dữ liệu đầu vào có 70 mẫu của 40 phần tử từ ảnh trong phần tiền xử lý. Còn 70 mẫu của 10 phần tử còn lại dùng để tạo mẫu test. Đầu tiên ta chọn đối tượng cần nhận diện, tiếp theo xử lý đối tượng và tạo dữ liệu kiểm tra, sau đó tạo mạng neural huấn luyện bằng công cụ nftool (hình 2.6).



Hình 2.6 Công cụ nftool trong Matlab

Cuối cùng là nhận diện kí tự và cho ra kết quả. Kết quả chạy cho thấy chương trình chưa thực sự nhận diện đúng kí tự, và chỉ nhận diện được các kí tự đã huấn luyện.

Chi tiết code và giao diện, video chạy Matlab có ở trong phần phụ lục.

2.5 Khái quát về FPGA, ngôn ngữ Verilog và mạng neural 2:3:2

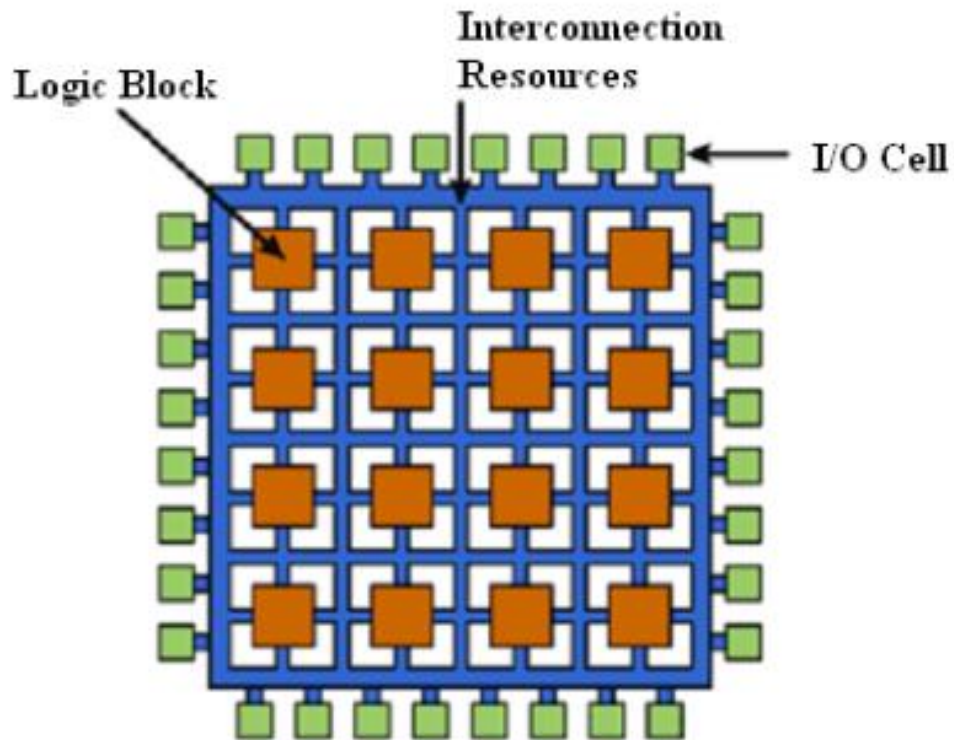
2.5.1 Giới thiệu chung về FPGA

FPGA (Field Programmable Gate Array – Mạch tích hợp có thể lập trình tại hiện trường) có ứng dụng rất lớn trong nhiều lĩnh vực như xử lý tín hiệu số, hàng không, vũ trụ, quốc phòng, xử lý ảnh... bởi tính linh động cao trong quá trình thiết kế, giúp người lập trình có thể xử lý những bài toán phức tạp mà trước kia chỉ thực hiện nhờ phần mềm. FPGA rất mạnh trong các lĩnh vực hoặc ứng dụng mà kiến trúc của nó yêu cầu xử lý song song, đặc biệt là mã hóa và giải mã. Hiện nay nghệ FPGA đang được sản xuất và hỗ trợ phần mềm bởi các hãng như: Xilinx, Altera, Actel, Atmel, ... trong đó Xilinx và Altera là hai hãng hàng đầu.

Các FPGA hiện nay có thể cho phép tái cấu hình lại từng phần, nghĩa là cho phép một phần của thiết kế được cấu hình lại trong khi những thiết kế khác vẫn tiếp tục hoạt động. Người thiết kế có thể tích hợp vào đó các bộ xử lý mềm hay vi xử lý tích hợp nhúng. Tính năng này chính là thể hiện khả năng lập trình tại “hiện trường” (field) của FPGA, mà không phụ thuộc dây chuyền sản xuất FPGA tại nhà máy. Các vi xử lý này có thể được thiết kế như các khối logic thông thường, mà mã nguồn do các hãng cung cấp, và có các ngoại vi được thiết kế linh động (khối giao tiếp UART (Universal asynchronous receiver-transmitter – Bộ thu phát không đồng bộ phổ quát), khối vào - ra đa chức năng (General-purpose input/output), Ethernet, ...). Các vi xử lý này cũng có thể được tính toán khả trình lại ngay trong khi đang chạy.

Kiến trúc cơ bản của một chip FPGA bao gồm:

- Các khối logic cơ bản lập trình được (Configurable logic blocks - CLBs)
- Hệ thống mạch liên kết lập trình được (Routing channels)
- Khối vào/ra (I/O Pads)



Hình 2.7 Kiến trúc một chip FPGA

Như trong hình 2.7, một ma trận các khối logic CLB được kết nối với nhau, phía ngoài được bao phủ bởi các khối vào ra I/O. Hệ thống kết nối bên trong chip gồm có các đường dây ngang và dọc có thể được xác định thông các chuyển mạch lập trình được. Việc sử dụng FPGA chính là việc lập trình các chuyển mạch ngắt hay đóng để kết nối các CLB với nhau. Ngoài các khối cơ bản một chip FPGA thường có thêm các thành phần: bộ nhân, bộ cộng nhân vi xử lý, bộ nhớ Block Ram, bộ quản lý xung đồng hồ.

2.5.2 Giới thiệu ngôn ngữ mô tả phần cứng Verilog

Ngôn ngữ mô tả phần cứng (Hardware description language - HDL) mô tả hành vi của mạch điện hoặc hệ thống, từ đó mạch điện vật lý hoặc hệ thống có thể được thực thi. 2 HDL phổ biến được sử dụng hiện nay là VHDL và VerilogHDL. VHDL được sử dụng phổ biến ở châu Âu, còn Verilog được sử dụng phổ biến ở Mỹ. Hai ngôn ngữ này khá tương đồng nhau, nếu đã biết một ngôn ngữ thì có thể dễ dàng học ngôn ngữ còn lại.

VHDL (Very High Speed Integrated Circuits HDL – VHSIC HDL) được sáng lập bởi Bộ Quốc phòng Mỹ trong những năm 80. Phiên bản đầu tiên là VHDL 87, lần nâng cấp sau đó có tên là VHDL 93. VHDL là ngôn ngữ mô tả phần cứng nguyên gốc đầu tiên được chuẩn hóa bởi Viện kỹ sư điện và điện tử Mỹ (Institute of Electrical and Electronics Engineers - IEEE), tới chuẩn IEEE 1076. VHDL được sử dụng để tự động hóa các thiết kế điện tử để mô tả các hệ thống kỹ thuật số và tín hiệu hỗn hợp. VHDL cũng được sử dụng như một ngôn ngữ lập trình song song đa chức năng.

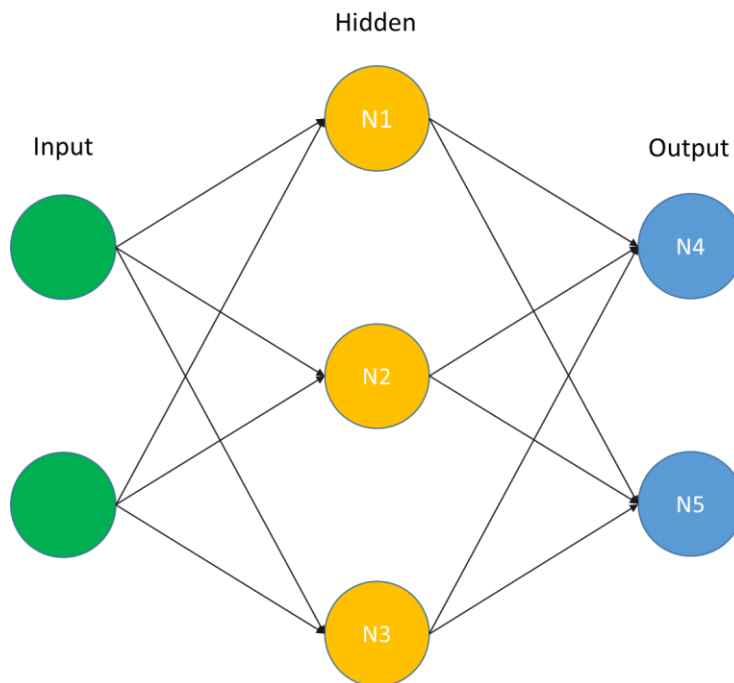
Verilog HDL được phát minh bởi Prabhu Goel, Phil Moorby, Chi-Lai Huang và Douglas Warmke khoảng cuối năm 1983, đầu năm 1984, trong dự án có tên là Automated Integrated Design Systems (sau đổi tên thành Gateway Design Automation vào năm 1985). Verilog có cấu trúc lỏng lẻo hơn VHDL và gần giống ngôn ngữ C rất gần gũi và được sử dụng phổ biến. Verilog có nhiều chuẩn: Verilog 97, Verilog 2001 và Verilog 2005. Verilog được sử dụng để mô hình hóa các hệ thống điện tử, phổ biến nhất trong việc thiết kế và kiểm tra mạch ở mức trừu tượng RTL (register-transfer level). Nó cũng được sử dụng trong việc xác minh các mạch tương tự và mạch tín hiệu hỗn hợp, cũng như trong thiết kế các mạch di truyền.

Trong VHDL và Verilog, các lệnh được thực hiện song song, chỉ khi các câu lệnh đặt trong khối process (đối với VHDL), always hoặc initial (đối với Verilog) thì các lệnh trong đó mới được thực hiện tuần tự.

Ở trong phần này ta sẽ sử dụng ngôn ngữ Verilog.

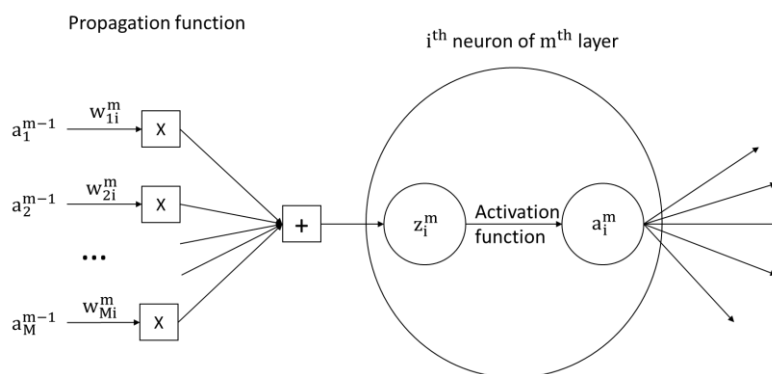
2.5.3 Tổng quan mạng neural 2:3:2

Hình 2.8 là một mạng neural 2:3:2 (2 neuron lớp đầu vào, 3 neuron lớp ẩn, 2 neuron lớp đầu ra). Ta chỉ quan tâm 5 neuron ở lớp ẩn và lớp đầu ra như ghi chú ở trên hình 2.8.



Hình 2.8 Mạng neural 2:3:2

Để thiết kế một mạng neural đầy đủ, ta thiết kế từng neuron riêng lẻ, sau đó nối các neuron với nhau. Trong một neuron, ta phải thiết kế các module mạch nhân và module hàm kích hoạt. Hình 2.9 là một neuron riêng lẻ.



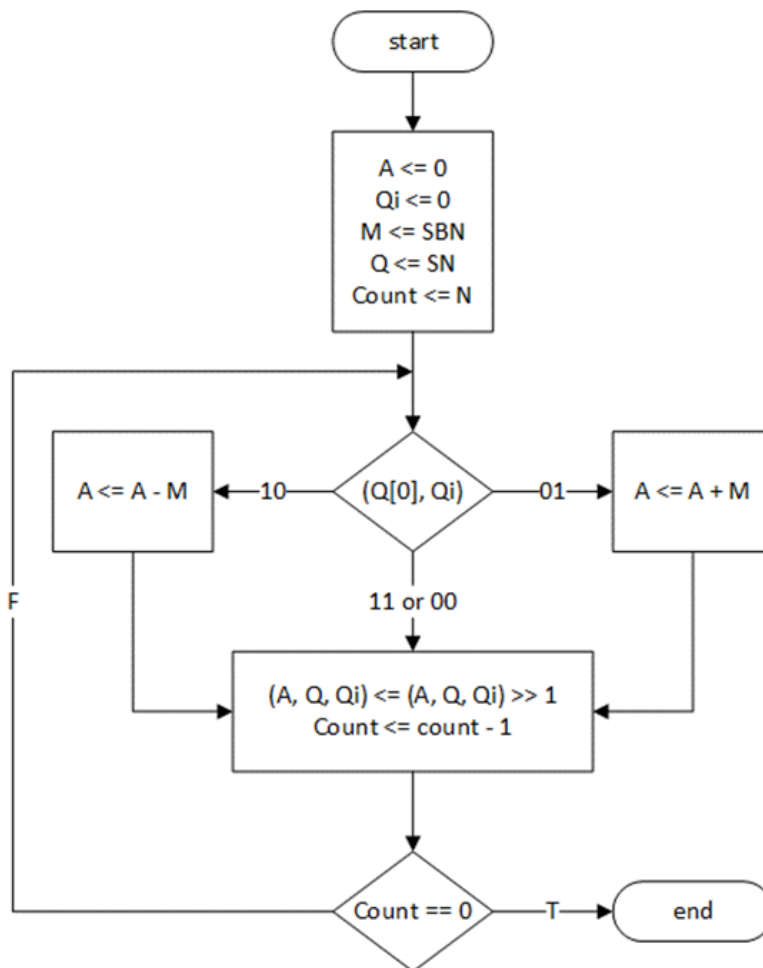
Hình 2.9 Mạng neuron riêng lẻ

2.6 Thiết một neuron

2.6.1 Module mạch nhân

Như đã thấy ở hình trên, một neuron được kết nối với M neuron ở lớp trước đó, chúng ta cần M bộ nhân 2 số thực, và $M/2$ bộ cộng 2 số thực (nếu M chẵn), $(M+1)/2$ bộ cộng 2 số thực (nếu M lẻ). Để thực hiện bộ cộng sử dụng toán tử cộng có sẵn trong Verilog.

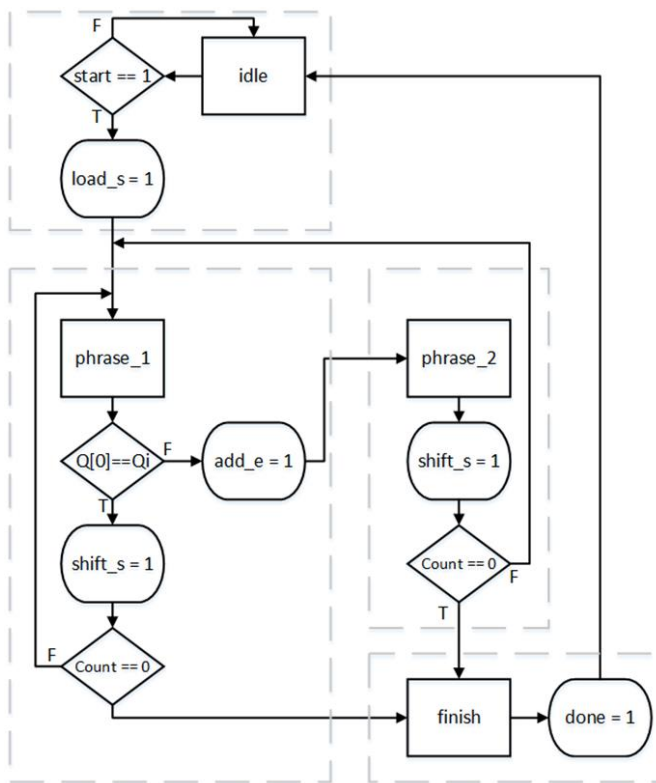
Mạch nhân số thực thực hiện chức năng nhân hai số thực có dấu dấu phẩy động. Quá trình nhân sẽ diễn ra trong nhiều chu kỳ đồng hồ. Cách thực hiện nhân ở đây là sử dụng giải thuật booth. Lưu đồ thuật toán được thể hiện ở hình 2.10.



Ban đầu gán A và $Q_i=0$, 2 biến M và Q lưu số bị nhân và số nhân, Q_i là bit dấu, count lưu số bit của 2 số cần nhân với nhau. Theo sơ đồ, trong mỗi vòng lặp, nếu giá trị của $Q[0]Q_i=01$ thì cộng thêm M vào A , 10 thì trừ M từ A , còn lại 11 hoặc 00 thì thực hiện quay phải toàn bộ dãy A, Q, Q_i . Sau mỗi vòng lặp giảm count đi 1, lặp đến khi count bằng 0 thì $\{A, Q\}$ chính là kết quả.

Hình 2.10 Lưu đồ thuật toán giải thuật Booth

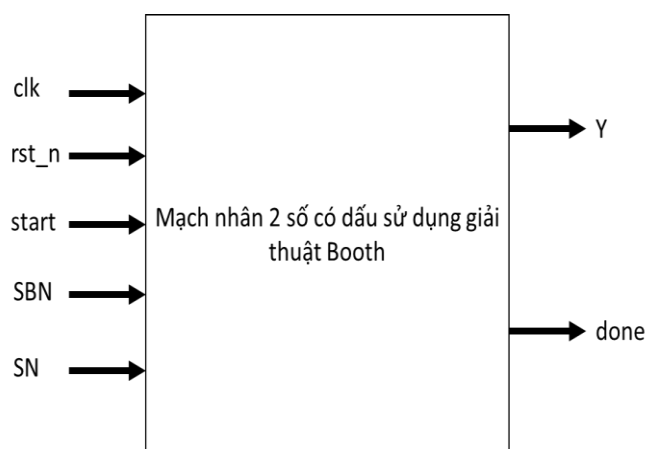
Sau đây ta xây dựng sơ đồ máy trạng thái thuật toán (Algorithmic state machine – ASM) (hình 2.11).



Hình 2.11 Sơ đồ ASM giải thuật Booth

Sơ đồ ASM được suy ra từ lưu đồ thuật toán. Mạch bao gồm các trạng thái: idle (trạng thái ban đầu), phrase_1, phrase_2, finish. Tín hiệu add_s cho phép chuyển trạng thái từ phrase_1 sang phrase_2 và cộng. Tín hiệu shift_s cho phép trừ bộ đếm và dịch các giá trị trong thanh ghi. Cuối cùng ở trạng thái finish, một tín hiệu done được xuất ra thông báo quá trình nhân hoàn tất.

Tổng quát mạch nhân như hình 2.12.



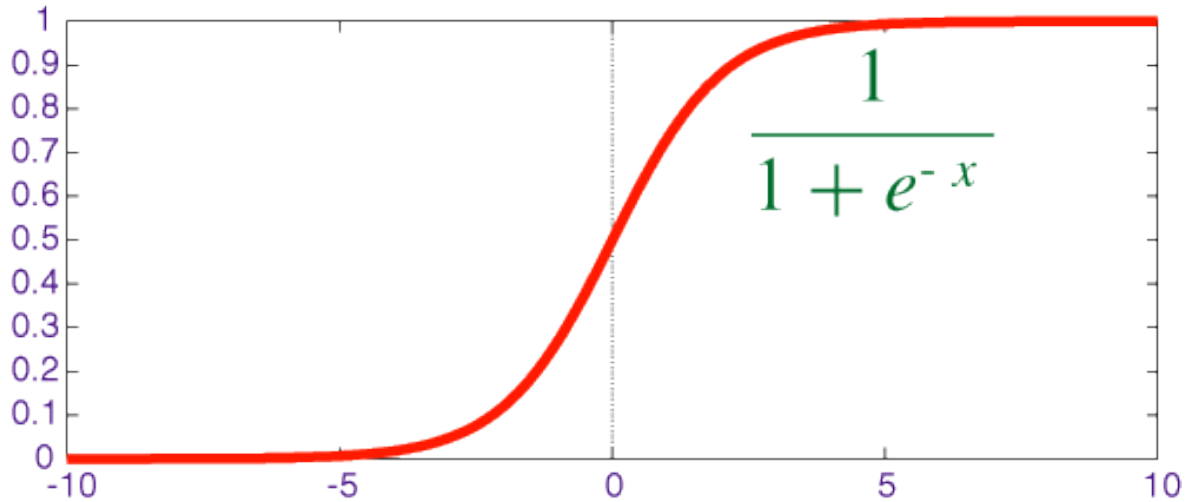
Hình 2.12 Tổng quát mạch nhân

Mạch có 5 tín hiệu vào: xung đồng hồ (clk), reset (rst_n), tín hiệu cho phép mạch chạy (start), số bị nhân (SBN), số nhân (SN); 2 tín hiệu ra: kết quả phép nhân (Y), tín hiệu báo hiệu quá trình nhân đã hoàn tất (done).

Chi tiết code mạch nhân có ở trong phần phụ lục.

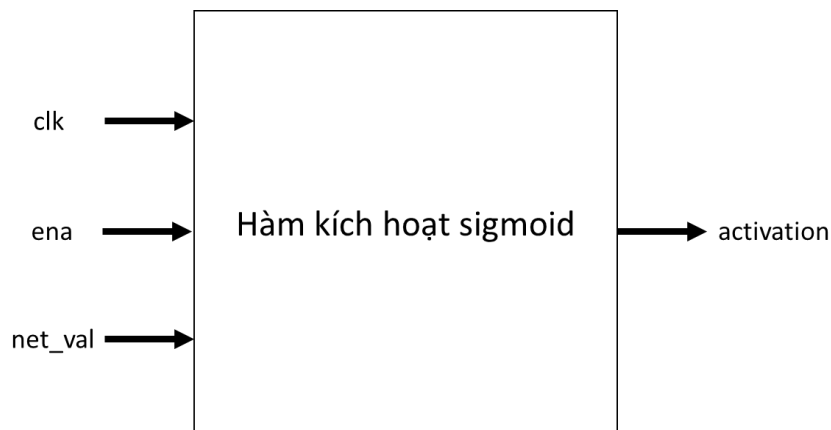
2.6.2 Module hàm kích hoạt

Hàm sigmoid là một hàm phi tuyến, nhận vào một giá trị thực bất kì và cho ra giá trị nằm trong khoảng từ 0 đến 1 (hình 2.13).



Hình 2.13 Hàm sigmoid

Hàm kích hoạt sigmoid được gen code bằng matlab, ta dùng công cụ HDL Coder trong Matlab để thay thế hàm Sigmoid với Lookup Table.

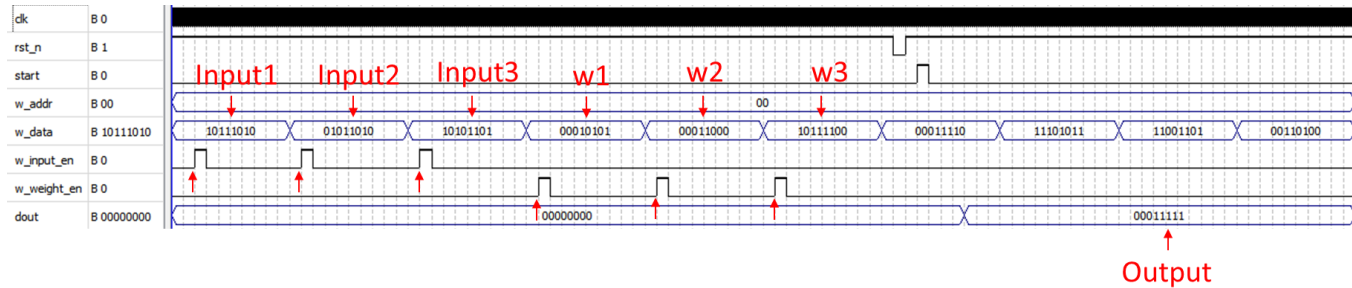


Hình 2.14 Tổng quát mạch hàm kích hoạt

Như hình 2.14, mạch hàm kích hoạt gồm 3 tín hiệu đầu vào: xung đồng hồ (clk), tín hiệu cho phép chạy module, giá trị vào neuron, tín hiệu đầu ra là giá trị cần tính của hàm sigmoid ra khỏi neuron. Code cụ thể đã tạo bằng matlab có trong phần phụ lục.

2.6.3 Mô phỏng một neuron

Hình 2.15 là kết quả mô phỏng một neuron thực hiện bằng công cụ Modelsim trong Quartus 13.

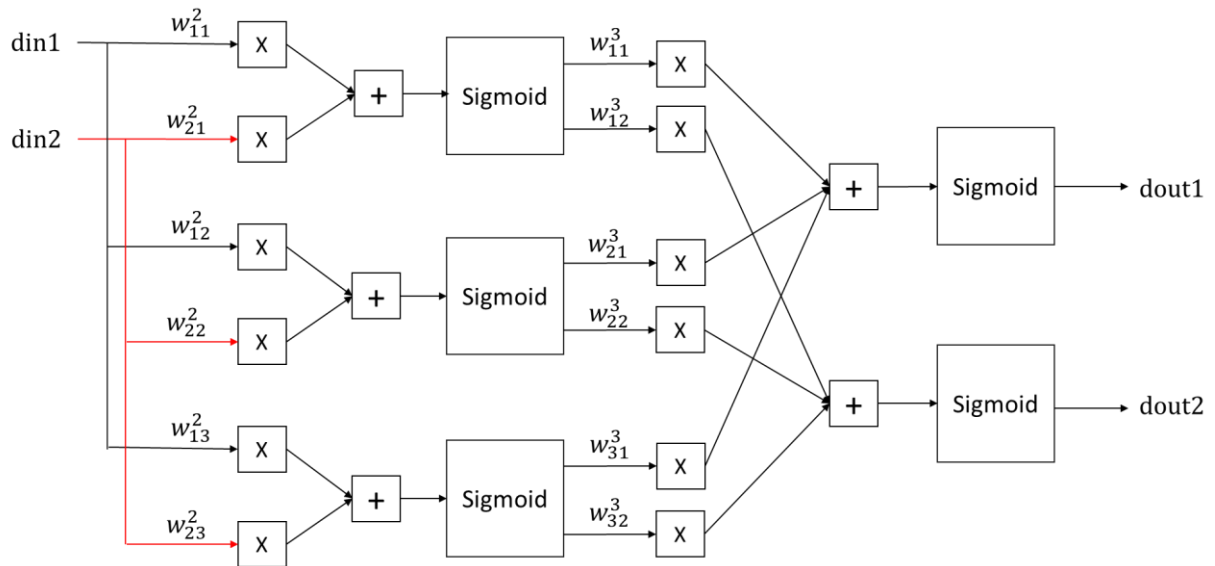


Hình 2.15 Mô phỏng một neuron

3 vector input được đưa vào tuần tự mỗi khi có tín hiệu w_input_en, tương tự với 3 trọng số khi có tín hiệu w_weight_en, tín hiệu start được kích hoạt, sau một loạt xung clock thì output hiện kết quả đầu ra. Kết quả này đã được kiểm tra là chính xác.

2.7 Thực hiện mạng neural 2:3:2

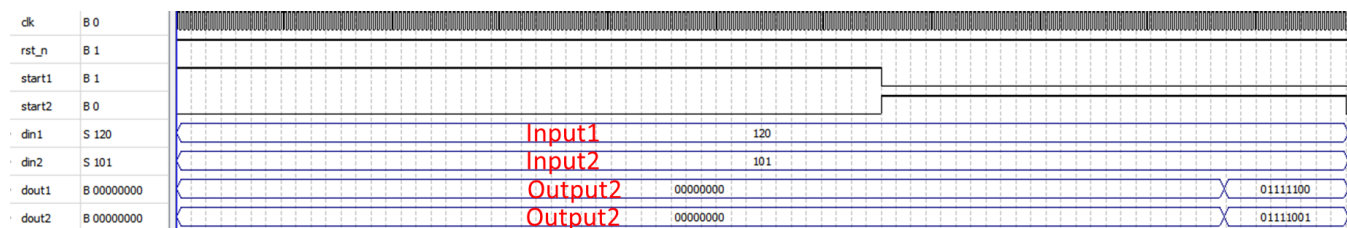
Cấu trúc cụ thể mạng neural 2:3:2 như hình 2.16.



Hình 2.16 Cấu trúc cụ thể mạng neural 2:3:2

Việc kết nối các neuron riêng lẻ với nhau không được kết quả như dự tính, bởi trình độ còn hạn chế, nên ở đây ta thiết kế mạng neural như một mạch tổ hợp với các module mạch nhân và module hàm kích hoạt đã có. Code toàn bộ mạch ở trong phần phụ lục.

Kết quả mô phỏng mạng neural 2:3:2 như hình 2.17.



Hình 2.17 Mô phỏng mạng neural 2:3:2

2 vector đầu vào din1 và din2 ở dạng thập phân, tín hiệu start1, start2 cho phép các mạch nhân từ layer1 sang layer2 và từ layer2 sang layer3 hoạt động, dout1 và dout2 là 2 vector đầu ra ở 2 neuron N4 và N5.

Chương 3. Nhận xét, đề xuất

3.1 Ưu điểm

Trí thông minh nhân tạo hiện nay có tính thực tế cao và được ứng dụng rất nhiều trong cuộc sống. Đề tài chúng em thực hiện là đề tài hay, giúp chúng em làm quen với mạng neural, sử dụng cả phần mềm (Matlab) và phần cứng (FPGA) để triển khai mạng neural. Trong thời gian thực tập tại ESRC Lab, em đã được trau dồi và luyện tập nhiều về các kỹ năng:

- Kỹ năng giao tiếp và làm việc nhóm
- Kỹ năng làm slide, thuyết trình, báo cáo

Các thành tựu đạt được sau khóa thực tập là:

- Sử dụng được Matlab thực hiện thành công nhân diện chữ cái bằng nftool
- Luyện tập Verilog, xây dựng được các module nhỏ và ghép lại thành module lớn
- Tìm hiểu về trí thông minh nhân tạo và mạng neural, triển khai mạng neural đơn giản trên FPGA

3.2 Nhược điểm

Vì làm việc theo nhóm nhiều người, khả năng lên kế hoạch còn hạn chế nên chúng em chưa có thời gian làm việc nhiều cùng nhau. Bên cạnh đó khả năng tiếng Anh còn hạn chế nên việc nghiên cứu sách vở mất rất nhiều thời gian. Trong đề tài thực hiện vẫn còn thiếu sót:

- Chưa ghép được các neuron riêng lẻ lại với nhau
- Chưa thực hiện được thuật toán lan truyền ngược

3.3 Đề xuất

Trong năm học bọn em sẽ tiếp tục nghiên cứu để hoàn thiện đề tài. Em hi vọng các khóa sinh viên sau cũng sẽ tham gia những khóa thực tập bổ ích như thế này.

C. KẾT LUẬN

Trong thời gian thực tập cùng các anh chị và các bạn, em đã học hỏi được nhiều kiến thức và kỹ năng thực tế, tích lũy thêm kinh nghiệm làm việc, các buổi thảo luận nhóm và lên thuyết trình giúp em tự tin hơn. Chúng em có thêm kiến thức mới về mạng neural, những ứng dụng của mạng neural, cách huấn luyện mạng neural, ngoài ra còn được luyện tập về Matlab và ngôn ngữ Verilog. Đề tài tuy còn nhiều thiếu sót nhưng là kết quả làm việc tích cực của cả nhóm. Một lần nữa em xin được chân thành cảm ơn viện Điện tử-viên thông đã giúp cho chúng em có học phần thực tập kỹ thuật, chân thành cảm ơn thầy Phạm Ngọc Nam chủ nhiệm phòng Lab đã tạo điều kiện để chúng có thể thực tập tại ESRC Lab, cảm ơn các anh chị và các bạn đã cùng em hoàn thành đề tài này.

Tài liệu tham khảo

- [1] Pong P, Chu, FPGA Prototyping by Verilog Example, A John Wiley & Sons, Inc., Publication, 2008
- [2] David Kriesel, A Brief Introduction to Neural Networks, dkriesel.com, 2005
- [3] Mentor: prof. Primož Potočnik, Student: Žiga Zadnik, Character Recognition: Handwritten character Recognition: Training a Simple NN for classification using MATLAB
- [4] Mrs. Rana D. Abdu-Aljabar, Design and Implementation of Neural Network in FPGA, Journal of Engineering and Development, Vol. 16, No.3, Sep. 2012 ISSN 1813-7822
- [5] Internet

D. PHỤ LỤC

- Chương trình nhận diện chữ cái sử dụng Matlab (file .m và file .fig)

<https://drive.google.com/open?id=1avAbi-VbZQnBkHtPHIDHqXEqXmxAzgjj>

- Video sử dụng Matlab nhận diện chữ cái

<https://drive.google.com/open?id=1WkoBrzeDwCspfNxjDy7SSIH4U8v2AFGF>

- Module mạch nhân

```
module multiplication_booth
    #(
        parameter N = 8
    )
    (
        input wire clk, rst_n,
        input wire start,
        input wire [N-1:0] SBN, SN,
        output wire [2*N-1:0] Y,
        output reg done
    );

    function integer log2;
    input integer n;

        for ( log2 = 0; n > 0; n = n >> 1)
            log2 = log2 + 1;

    endfunction

    // encode state
    localparam idle    = 2'b00;
    localparam phrase_1 = 2'b01;
    localparam phrase_2 = 2'b10;
    localparam finish   = 2'b11;

    reg load_s, add_e, shift_s;
    wire Q0_eq_Qi, count_is_0;

    reg [1:0] state_reg, state_next;
    reg [N-1:0] Q_reg, A_reg, M_reg;
    reg [N-1:0] Q_next, A_next, M_next;
    reg Qi_reg, Qi_next;
    reg [log2(N-1)-1:0] count_reg, count_next;
```

```

always @(posedge clk, negedge rst_n)
    if (!rst_n)
        state_reg <= 1'b0;
    else
        state_reg <= state_next;

always @*
begin
    state_next = state_reg;
    load_s     = 1'b0;
    add_e      = 1'b0;
    shift_s    = 1'b0;
    done       = 1'b0;

    case (state_reg)
        idle:
            if (start)
                begin
                    state_next = phrase_1;
                    load_s = 1'b1;
                end

            phrase_1:
                begin
                    if (!Q0_eq_Qi)
                        state_next = phrase_2;
                    else if (count_is_0)
                        state_next = finish;

                    if (!Q0_eq_Qi)
                        add_e = 1'b1;
                    else
                        shift_s = 1'b1;
                end

            phrase_2:
                begin
                    if (count_is_0)
                        state_next = finish;
                    else
                        state_next = phrase_1;

                    shift_s = 1'b1;
                end

            finish:
                begin
                    state_next = idle;
                end
    endcase
end

```

```

                                done = 1'b1;
                                end
                                endcase
end

always @(posedge clk, negedge rst_n)
    if (!rst_n)
        begin
            Q_reg    <= 1'b0;
            Qi_reg   <= 1'b0;
            A_reg    <= 1'b0;
            M_reg    <= 1'b0;
            count_reg <= 1'b0;
        end
    else
        begin
            Q_reg    <= Q_next;
            Qi_reg   <= Qi_next;
            A_reg    <= A_next;
            M_reg    <= M_next;
            count_reg <= count_next;
        end
    end

always @*
begin
    // default value
    Q_next    = Q_reg;
    Qi_next   = Qi_reg;
    A_next    = A_reg;
    M_next    = M_reg;
    count_next = count_reg;

    if (add_e) // ASM control
        if (Qi_reg) // check Qi to determine add or sub
            A_next = A_reg + M_reg;
        else
            A_next = A_reg - M_reg;

    if (shift_s)
        begin
            { A_next, Q_next, Qi_next } = { A_reg[N-1], A_reg[N-1:0], Q_reg };
            count_next = count_reg - 1'b1;
        end

    if (load_s)
        begin
            Q_next    = SN;
            Qi_next   = 1'b0;

```



```

        A_next    = 1'b0;
        M_next    = SBN;
        count_next = N-1;
    end
end

assign Q0_eq_Qi = Q_reg[0] == Qi_reg;
assign count_is_0 = count_reg == 0;

// output
assign Y = { A_reg, Q_reg };
endmodule

```

- Module hàm kích hoạt (gồm module mem_act và module activation, được gen code từ Matlab)

```
`timescale 1ns / 1ps
module mem_act(clk, din, addr, dout);
parameter DWIDTH=16;
parameter AWIDTH=8;
parameter DWIDTH_TMP=32;

input clk, din;
input [AWIDTH-1:0] addr;
output [DWIDTH-1:0] dout;

reg [DWIDTH-1:0] dout;
reg [DWIDTH_TMP-1:0] dout_tmp;
reg [DWIDTH_TMP-1:0] mem [0:(2**AWIDTH-1)];

initial begin
    mem[8'b1000_0000]=32'b0000_0000_0000_0000_0001_0101_1111_1010;//-8.0000
    mem[8'b1000_0001]=32'b0000_0000_0000_0000_0001_0111_0110_0101;
    mem[8'b1000_0010]=32'b0000_0000_0000_0000_0001_1000_1110_0111;
    mem[8'b1000_0011]=32'b0000_0000_0000_0000_0001_1010_1000_0010;
    mem[8'b1000_0100]=32'b0000_0000_0000_0000_0001_1100_0011_1000;
    mem[8'b1000_0101]=32'b0000_0000_0000_0000_0001_1110_0000_1001;
    mem[8'b1000_0110]=32'b0000_0000_0000_0000_0001_1111_1111_1001;
    mem[8'b1000_0111]=32'b0000_0000_0000_0000_0010_0010_0000_1000;
    mem[8'b1000_1000]=32'b0000_0000_0000_0000_0010_0100_0011_1010;
    mem[8'b1000_1001]=32'b0000_0000_0000_0000_0010_0110_1001_0000;
    mem[8'b1000_1010]=32'b0000_0000_0000_0000_0010_1001_0000_1100;
    mem[8'b1000_1011]=32'b0000_0000_0000_0000_0010_1011_1011_0001;
    mem[8'b1000_1100]=32'b0000_0000_0000_0000_0010_1110_1000_0010;
    mem[8'b1000_1101]=32'b0000_0000_0000_0000_0011_0001_1000_0010;
    mem[8'b1000_1110]=32'b0000_0000_0000_0000_0011_0100_1011_0010;
    mem[8'b1000_1111]=32'b0000_0000_0000_0000_0011_1000_0001_1000;
    //end at -7.06250
    mem[8'b1001_0000]=32'b0000_0000_0000_0000_0011_1011_1011_0101; //-7.0000
    mem[8'b1001_0001]=32'b0000_0000_0000_0000_0011_1111_1000_1110;
    mem[8'b1001_0010]=32'b0000_0000_0000_0000_0100_0011_1010_0110;
    mem[8'b1001_0011]=32'b0000_0000_0000_0000_0100_1000_0000_0010;
    mem[8'b1001_0100]=32'b0000_0000_0000_0000_0100_1100_1010_0101;
    mem[8'b1001_0101]=32'b0000_0000_0000_0000_0101_0001_1001_0101;
    mem[8'b1001_0110]=32'b0000_0000_0000_0000_0101_0110_1101_0110;
    mem[8'b1001_0111]=32'b0000_0000_0000_0000_0101_1100_0110_0010;
    mem[8'b1001_1000]=32'b0000_0000_0000_0000_0110_0010_0110_0010;
    mem[8'b1001_1001]=32'b0000_0000_0000_0000_0110_1000_1011_0111;
    mem[8'b1001_1010]=32'b0000_0000_0000_0000_0110_1111_0111_0101;
    mem[8'b1001_1011]=32'b0000_0000_0000_0000_0111_0110_1010_0010;
```

```

mem[8'b1001_1100]=32'b0000_0000_0000_0000_0111_1110_0100_0101;
mem[8'b1001_1101]=32'b0000_0000_0000_0000_1000_0110_0110_0110;
mem[8'b1001_1110]=32'b0000_0000_0000_0000_1000_1111_0000_1100;
mem[8'b1001_1111]=32'b0000_0000_0000_0000_1001_1000_0100_0000;
//end at -6.06250
mem[8'b1010_0000]=32'b0000_0000_0000_0000_1010_0010_0000_1100; //-6.0000
mem[8'b1010_0001]=32'b0000_0000_0000_0000_1010_1100_0111_1000;
mem[8'b1010_0010]=32'b0000_0000_0000_0000_1011_0111_1001_0000;
mem[8'b1010_0011]=32'b0000_0000_0000_0000_1100_0011_0101_1101;
mem[8'b1010_0100]=32'b0000_0000_0000_0000_1100_1111_1110_1101;
mem[8'b1010_0101]=32'b0000_0000_0000_0000_1101_1101_0100_1010;
mem[8'b1010_0110]=32'b0000_0000_0000_0000_1110_1011_1000_0011;
mem[8'b1010_0111]=32'b0000_0000_0000_0000_1111_1010_1010_0100;
mem[8'b1010_1000]=32'b0000_0000_0000_0001_0000_1010_1011_1110;
mem[8'b1010_1001]=32'b0000_0000_0000_0001_0001_1011_1101_1111;
mem[8'b1010_1010]=32'b0000_0000_0000_0001_0010_1110_0001_1000;
mem[8'b1010_1011]=32'b0000_0000_0000_0001_0100_0001_0111_1011;
mem[8'b1010_1100]=32'b0000_0000_0000_0001_0101_0110_0001_1011;
mem[8'b1010_1101]=32'b0000_0000_0000_0001_0110_1100_0000_1100;
mem[8'b1010_1110]=32'b0000_0000_0000_0001_1000_0011_0110_0011;
mem[8'b1010_1111]=32'b0000_0000_0000_0001_1001_1100_0011_0111;
//end at -5.06250
mem[8'b1011_0000]=32'b0000_0000_0000_0001_1011_0110_1001_1111; //-5.0000
mem[8'b1011_0001]=32'b0000_0000_0000_0001_1101_0010_1011_0110;
mem[8'b1011_0010]=32'b0000_0000_0000_0001_1111_0000_1001_0101;
mem[8'b1011_0011]=32'b0000_0000_0000_0010_0001_0000_0101_1010;
mem[8'b1011_0100]=32'b0000_0000_0000_0010_0011_0010_0010_0010;
mem[8'b1011_0101]=32'b0000_0000_0000_0010_0101_0110_0000_1111;
mem[8'b1011_0110]=32'b0000_0000_0000_0010_0111_1100_0100_0001;
mem[8'b1011_0111]=32'b0000_0000_0000_0010_1010_0100_1101_1110;
mem[8'b1011_1000]=32'b0000_0000_0000_0010_1101_0000_0000_1010;
mem[8'b1011_1001]=32'b0000_0000_0000_0010_1111_1101_1111_0000;
mem[8'b1011_1010]=32'b0000_0000_0000_0011_0010_1110_1011_1000;
mem[8'b1011_1011]=32'b0000_0000_0000_0011_0110_0010_1001_0010;
mem[8'b1011_1100]=32'b0000_0000_0000_0011_1001_1001_1010_1101;
mem[8'b1011_1101]=32'b0000_0000_0000_0011_1101_0100_0011_1010;
mem[8'b1011_1110]=32'b0000_0000_0000_0100_0001_0010_0111_0001;
mem[8'b1011_1111]=32'b0000_0000_0000_0100_0101_0100_1000_1001;
//end at -4.06250
mem[8'b1100_0000]=32'b0000_0000_0000_0100_1001_1010_1011_1111; //-4.0000
mem[8'b1100_0001]=32'b0000_0000_0000_0100_1110_0101_0101_0000;
mem[8'b1100_0010]=32'b0000_0000_0000_0101_0011_0100_1000_0000;
mem[8'b1100_0011]=32'b0000_0000_0000_0101_1000_1000_1001_0101;
mem[8'b1100_0100]=32'b0000_0000_0000_0101_1110_0001_1101_1000;
mem[8'b1100_0101]=32'b0000_0000_0000_0110_0100_0000_1001_0111;
mem[8'b1100_0110]=32'b0000_0000_0000_0110_1010_0101_0010_0100;
mem[8'b1100_0111]=32'b0000_0000_0000_0111_0000_1111_1101_0100;
mem[8'b1100_1000]=32'b0000_0000_0000_0111_1000_0001_0000_0010;

```

```

mem[8'b1100_1001]=32'b0000_0000_0000_0111_1111_1001_0000_0011;
mem[8'b1100_1010]=32'b0000_0000_0000_1000_0111_1000_0100_0011;
mem[8'b1100_1011]=32'b0000_0000_0000_1000_1111_1111_0100_0001;
mem[8'b1100_1100]=32'b0000_0000_0000_1001_1000_1110_0100_0001;
mem[8'b1100_1101]=32'b0000_0000_0000_1010_0010_0101_1100_0101;
mem[8'b1100_1110]=32'b0000_0000_0000_1010_1100_0110_0100_0011;
mem[8'b1100_1111]=32'b0000_0000_0000_1011_0111_0000_0011_0101;
//end at -3.06250
mem[8'b1101_0000]=32'b0000_0000_0000_1100_0010_0100_0001_1010; // -3.0000
mem[8'b1101_0001]=32'b0000_0000_0000_1100_1110_0010_0111_1000;
mem[8'b1101_0010]=32'b0000_0000_0000_1101_1010_1011_1101_0111;
mem[8'b1101_0011]=32'b0000_0000_0000_1110_1000_0000_1100_0110;
mem[8'b1101_0100]=32'b0000_0000_0000_1111_0110_0001_1101_0111;
mem[8'b1101_0101]=32'b0000_0000_0001_0000_0100_1111_1010_0000;
mem[8'b1101_0110]=32'b0000_0000_0001_0001_0100_1010_1011_1101;
mem[8'b1101_0111]=32'b0000_0000_0001_0010_0101_0011_1100_1101;
mem[8'b1101_1000]=32'b0000_0000_0001_0011_0110_1011_0111_0001;
mem[8'b1101_1001]=32'b0000_0000_0001_0100_1001_0010_0100_1111;
mem[8'b1101_1010]=32'b0000_0000_0001_0101_1100_1001_0000_1101;
mem[8'b1101_1011]=32'b0000_0000_0001_0111_0001_0000_0101_0110;
mem[8'b1101_1100]=32'b0000_0000_0001_1000_0110_1000_1101_0011;
mem[8'b1101_1101]=32'b0000_0000_0001_1001_1101_0011_0010_1110;
mem[8'b1101_1110]=32'b0000_0000_0001_1011_0101_0000_0001_0011;
mem[8'b1101_1111]=32'b0000_0000_0001_1100_1110_0000_0010_1001;
//end at -2.06250
mem[8'b1110_0000]=32'b000000000000111101000010000010101; // -2.0000
mem[8'b1110_0001]=32'b000000000001000000011110001111001;
mem[8'b1110_0010]=32'b000000000001000100000100111110010;
mem[8'b1110_0011]=32'b00000000000100011110110100010100;
mem[8'b1110_0100]=32'b000000000001001011110011001101100;
mem[8'b1110_0101]=32'b00000000000100111111011001111110;
mem[8'b1110_0110]=32'b000000000001010100001110111000000;
mem[8'b1110_0111]=32'b000000000001011000101110010011110;
mem[8'b1110_1000]=32'b000000000001011101011001101110000;
mem[8'b1110_1001]=32'b000000000001100010010001010000010;
mem[8'b1110_1010]=32'b000000000001100111010101000001000;
mem[8'b1110_1011]=32'b000000000001101100100101000100100;
mem[8'b1110_1100]=32'b000000000001110010000001011100000;
mem[8'b1110_1101]=32'b000000000001110111101010000101110;
mem[8'b1110_1110]=32'b000000000001111101011110111100100;
mem[8'b1110_1111]=32'b000000000010000011011111110111110;
//end at -1.06250
mem[8'b1111_0000]=32'b000000000010001001101100101011000; // -1.0000
mem[8'b1111_0001]=32'b000000000010010000000101000110011;
mem[8'b1111_0010]=32'b00000000001001011010101000110101100;
mem[8'b1111_0011]=32'b000000000010011101010111100000100;
mem[8'b1111_0100]=32'b000000000010100100010000101011000;
mem[8'b1111_0101]=32'b000000000010101011010011110100111;

```

```

mem[8'b1111_0110]=32'b00000000010110010100000011001111;
mem[8'b1111_0111]=32'b00000000010111001110101110001101;
mem[8'b1111_1000]=32'b00000000011000001010011010000001;
mem[8'b1111_1001]=32'b00000000011001000111000000110000;
mem[8'b1111_1010]=32'b00000000011010000100011100000000;
mem[8'b1111_1011]=32'b00000000011011000010100101000100;
mem[8'b1111_1100]=32'b00000000011100000001010100110011;
mem[8'b1111_1101]=32'b00000000011101000000100011111000;
mem[8'b1111_1110]=32'b0000000001111000000001010101010;
mem[8'b1111_1111]=32'b00000000011111000000000001010101;
//end at -0.06250
mem[8'b0000_0000]=32'b00000000010000000000000000000000; //0.0000
mem[8'b0000_0001]=32'b000000000100000111111111110101011;
mem[8'b0000_0010]=32'b00000000010000111111111101010110;
mem[8'b0000_0011]=32'b000000000100010111111011100001000;
mem[8'b0000_0100]=32'b000000000100011111110101011001101;
mem[8'b0000_0101]=32'b00000000010010011110101101011100;
mem[8'b0000_0110]=32'b000000000100101111011100100000000;
mem[8'b0000_0111]=32'b000000000100110111000111111010000;
mem[8'b0000_1000]=32'b00000000010011110101100101111111;
mem[8'b0000_1001]=32'b000000000101000110001010001110011;
mem[8'b0000_1010]=32'b000000000101001101011111100110001;
mem[8'b0000_1011]=32'b000000000101010100101100001011001;
mem[8'b0000_1100]=32'b000000000101011011101111010101000;
mem[8'b0000_1101]=32'b000000000101100010101000011111100;
mem[8'b0000_1110]=32'b000000000101101001010111001010100;
mem[8'b0000_1111]=32'b00000000010110111111010111001101;
//end at 0.93750
mem[8'b0001_0000]=32'b000000000101110110010011010101000; //1.0000
mem[8'b0001_0001]=32'b00000000010111100100000001000010;
mem[8'b0001_0010]=32'b000000000110000010100001000011100;
mem[8'b0001_0011]=32'b000000000110001000010101111010010;
mem[8'b0001_0100]=32'b00000000011000110111110100100000;
mem[8'b0001_0101]=32'b000000000110010011011010111011100;
mem[8'b0001_0110]=32'b00000000011001100010101011111000;
mem[8'b0001_0111]=32'b000000000110011101101110101111110;
mem[8'b0001_1000]=32'b000000000110100010100110010010000;
mem[8'b0001_1001]=32'b000000000110100111010001101100010;
mem[8'b0001_1010]=32'b000000000110101011110001001000000;
mem[8'b0001_1011]=32'b000000000110110000000100110000010;
mem[8'b0001_1100]=32'b000000000110110100001100110010100;
mem[8'b0001_1101]=32'b000000000110111000001001011101100;
mem[8'b0001_1110]=32'b000000000110111011111011000001110;
mem[8'b0001_1111]=32'b000000000110111111100001110000111;
//end at 1.93750
mem[8'b0010_0000]=32'b000000000111000010111101111101011; //2.0000
mem[8'b0010_0001]=32'b000000000111000110001111111010111;
mem[8'b0010_0010]=32'b000000000111001001010111111101101;

```

```

mem[8'b0010_0011]=32'b00000000111001100010110011010010;
mem[8'b0010_0100]=32'b00000000111001111001011100101101;
mem[8'b0010_0101]=32'b00000000111010001110111110101010;
mem[8'b0010_0110]=32'b00000000111010100011011011110011;
mem[8'b0010_0111]=32'b0000000011101011011011011010001;
mem[8'b0010_1000]=32'b00000000111011001001010010001111;
mem[8'b0010_1001]=32'b00000000111011011010110000110011;
mem[8'b0010_1010]=32'b00000000111011101011010101000011;
mem[8'b0010_1011]=32'b00000000111011111011000001100000;
mem[8'b0010_1100]=32'b00000000111100001001111000101001;
mem[8'b0010_1101]=32'b00000000111100010111111100111010;
mem[8'b0010_1110]=32'b00000000111100100101010000101001;
mem[8'b0010_1111]=32'b00000000111100110001110110001000;
//end at 2.93750
mem[8'b0011_0000]=32'b00000000111100111101101111100110; //3.0000
mem[8'b0011_0001]=32'b00000000111101001000111111001011;
mem[8'b0011_0010]=32'b00000000111101010011100110111101;
mem[8'b0011_0011]=32'b000000001111010111101101000111011;
mem[8'b0011_0100]=32'b00000000111101100111000110111111;
mem[8'b0011_0101]=32'b00000000111101110000000010111111;
mem[8'b0011_0110]=32'b00000000111101111000011110101101;
mem[8'b0011_0111]=32'b00000000111110000000011011110101;
mem[8'b0011_1000]=32'b00000000111110000111111011111110;
mem[8'b0011_1001]=32'b00000000111110001111000000101100;
mem[8'b0011_1010]=32'b00000000111110010101101011011100;
mem[8'b0011_1011]=32'b000000001111100110111111101101001;
mem[8'b0011_1100]=32'b00000000111110100001111000101000;
mem[8'b0011_1101]=32'b00000000111110100111011101101011;
mem[8'b0011_1110]=32'b00000000111110101100101110000000;
mem[8'b0011_1111]=32'b00000000111110110001101010110000;
//end at 3.93750
mem[8'b0100_0000]=32'b00000000111110110110010101000001; //4.0000
mem[8'b0100_0001]=32'b00000000111110111010101101110111;
mem[8'b0100_0010]=32'b00000000111110111110110110001111;
mem[8'b0100_0011]=32'b00000000111111000010101111000110;
mem[8'b0100_0100]=32'b00000000111111000110011001010011;
mem[8'b0100_0101]=32'b00000000111111001001110101101110;
mem[8'b0100_0110]=32'b00000000111111001101000101001000;
mem[8'b0100_0111]=32'b00000000111111010000001000010000;
mem[8'b0100_1000]=32'b00000000111111010010111111110110;
mem[8'b0100_1001]=32'b00000000111111010101101100100010;
mem[8'b0100_1010]=32'b00000000111111011000001110111111;
mem[8'b0100_1011]=32'b00000000111111011010100111110001;
mem[8'b0100_1100]=32'b00000000111111011100110111011110;
mem[8'b0100_1101]=32'b00000000111111011100110111011110;
mem[8'b0100_1110]=32'b00000000111111100000111101101011;
mem[8'b0100_1111]=32'b00000000111111100010110101001010;
//end at 4.93750

```

```

mem[8'b0101_0000]=32'b0000000011111100100100101100001; //5.0000
mem[8'b0101_0001]=32'b0000000011111100110001111001001;
mem[8'b0101_0010]=32'b0000000011111100111110010011101;
mem[8'b0101_0011]=32'b0000000011111101001001111110100;
mem[8'b0101_0100]=32'b0000000011111101010100111100101;
mem[8'b0101_0101]=32'b0000000011111101011111010000101;
mem[8'b0101_0110]=32'b0000000011111101101000111101000;
mem[8'b0101_0111]=32'b0000000011111101110010000100001;
mem[8'b0101_1000]=32'b0000000011111101111010101000010;
mem[8'b0101_1001]=32'b0000000011111110000010101011100;
mem[8'b0101_1010]=32'b0000000011111110001010001111101;
mem[8'b0101_1011]=32'b0000000011111110010001010110110;
mem[8'b0101_1100]=32'b000000001111111001100000010011;
mem[8'b0101_1101]=32'b0000000011111110011110010100011;
mem[8'b0101_1110]=32'b0000000011111110100100001110000;
mem[8'b0101_1111]=32'b0000000011111110101001110001000;
//end at 5.93750
mem[8'b0110_0000]=32'b0000000011111110101110111110100; //6.0000
mem[8'b0110_0001]=32'b0000000011111110110011111000000;
mem[8'b0110_0010]=32'b0000000011111110111000011110100;
mem[8'b0110_0011]=32'b0000000011111110111100110011010;
mem[8'b0110_0100]=32'b0000000011111111000000110111011;
mem[8'b0110_0101]=32'b0000000011111111000100101011110;
mem[8'b0110_0110]=32'b0000000011111111001000010001011;
mem[8'b0110_0111]=32'b00000000111111110010111101001001;
mem[8'b0110_1000]=32'b0000000011111111001110110011110;
mem[8'b0110_1001]=32'b0000000011111111010001110010010;
mem[8'b0110_1010]=32'b0000000011111111010100100101010;
mem[8'b0110_1011]=32'b0000000011111111010111001101011;
mem[8'b0110_1100]=32'b0000000011111111011001101011011;
mem[8'b0110_1101]=32'b0000000011111111011011111111110;
mem[8'b0110_1110]=32'b0000000011111111011110001011010;
mem[8'b0110_1111]=32'b0000000011111111100000001110010;
//end at 6.93750
mem[8'b0111_0000]=32'b0000000011111111100010001001011; //7.0000
mem[8'b0111_0001]=32'b0000000011111111100011111101000;
mem[8'b0111_0010]=32'b0000000011111111100101101001110;
mem[8'b0111_0011]=32'b0000000011111111100111001111110;
mem[8'b0111_0100]=32'b0000000011111111101000101111110;
mem[8'b0111_0101]=32'b0000000011111111101010001001111;
mem[8'b0111_0110]=32'b0000000011111111101011011110100;
mem[8'b0111_0111]=32'b0000000011111111101100101110000;
mem[8'b0111_1000]=32'b0000000011111111101101111000110;
mem[8'b0111_1001]=32'b0000000011111111101110111111000;
mem[8'b0111_1010]=32'b0000000011111111110000000000111;
mem[8'b0111_1011]=32'b0000000011111111110000111110111;
mem[8'b0111_1100]=32'b0000000011111111110001111001000;
mem[8'b0111_1101]=32'b0000000011111111110010101111110;

```

```

        mem[8'b0111_1110]=32'b0000000011111111110011100011001;
        mem[8'b0111_1111]=32'b0000000011111111110100010011011;
        //end at 7.93750
end

always@(posedge clk) begin
    if(din==1)
        begin
            dout_tmp = mem[addr];    //0000_0000.0000_0000_0000_0000_0000
            dout = dout_tmp[29:14];    //00_0000.0000_0000_00
        end
    end
endmodule

```

```

`timescale 1ns / 1ps
module activation
    #(
        parameter DWIDTH = 16,
        parameter AWIDTH = 8,
        parameter SP = 6
    )
    (
        input wire clock,
        input wire ena,
        input signed [DWIDTH-1:0] net_val,
        output wire [DWIDTH-1:0] activation
    );

    mem_act #(
        .DWIDTH ( DWIDTH ),
        .AWIDTH ( AWIDTH )
    ) lut
    (
        .clk ( clock ),
        .din ( ena ),
        .addr ( net_val[DWIDTH-1-SP+4:DWIDTH-AWIDTH-SP+4] ),
        .dout ( activation )
    );
endmodule

```


- Module mạng neural 2:3:2

```

module NN
#(
    parameter N=8;
    parameter Fix=5
)

(
    input wire clk,rst_n,
    input wire start1,start2,
    input wire [N-1:0] din1,din2,
    input wire [N-1:0] w01,w11,w02,w12,w03,w13,w14,w24,w34,w15,w25,w35,
    output wire [N-1:0] dout1,dout2
);

wire [2*N-1:0] z1,z2,z3,z4,z5,y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,a1,a2,a3,a4,a5;
wire [N-1:0] z11,z22,z33;
wire done1,done2,done3,done4,done5,done6,done7,done8,done9,done10,done11,done12,ena1,ena2;

multiplication_booth m1
(.clk(clk),.start(start1),.rst_n(rst_n),.SBN(din1),.SN(w01),.Y(y1),.done(done1));
multiplication_booth m2
(.clk(clk),.start(start1),.rst_n(rst_n),.SBN(din2),.SN(w11),.Y(y2),.done(done2));
assign z1=y1+y2;

multiplication_booth m3
(.clk(clk),.start(start1),.rst_n(rst_n),.SBN(din1),.SN(w02),.Y(y3),.done(done3));
multiplication_booth m4
(.clk(clk),.start(start1),.rst_n(rst_n),.SBN(din2),.SN(w12),.Y(y4),.done(done4));
assign z2=y3+y4;

multiplication_booth m5
(.clk(clk),.start(start1),.rst_n(rst_n),.SBN(din1),.SN(w03),.Y(y5),.done(done5));
multiplication_booth m6
(.clk(clk),.start(start1),.rst_n(rst_n),.SBN(din2),.SN(w13),.Y(y6),.done(done6));
assign z3=y5+y6;

assign ena1=done1 & done2 & done3 & done4 & done5 & done6;
activation ac1 (.clock(clk),.ena(ena1),.net_val(z1),.activation(a1));
activation ac2 (.clock(clk),.ena(ena1),.net_val(z2),.activation(a2));
activation ac3 (.clock(clk),.ena(ena1),.net_val(z3),.activation(a3));

assign z11=a1[N-1+Fix:Fix];
assign z22=a2[N-1+Fix:Fix];
assign z33=a3[N-1+Fix:Fix];

multiplication_booth m7 (.clk(clk),.start(start2),.rst_n(rst_n),.SBN(z11),.SN(w14),.Y(y7),.done(done7));
multiplication_booth m8 (.clk(clk),.start(start2),.rst_n(rst_n),.SBN(z22),.SN(w24),.Y(y8),.done(done8));

```

```

multiplication_booth m9 (.clk(clk),.start(start2),.rst_n(rst_n),.SBN(z33),.SN(w34),.Y(y9),.done(done9));
assign z4=y7+y8+y9;

multiplication_booth m10
(.clk(clk),.start(start2),.rst_n(rst_n),.SBN(z11),.SN(w15),.Y(y10),.done(done10));
multiplication_booth m11
(.clk(clk),.start(start2),.rst_n(rst_n),.SBN(z22),.SN(w25),.Y(y11),.done(done11));
multiplication_booth m12
(.clk(clk),.start(start2),.rst_n(rst_n),.SBN(z33),.SN(w35),.Y(y12),.done(done12));
assign z5=y10+y11+y12;

assign ena2=done7 & done8 & done9 & done10 & done11 & done12;
activation ac4 (.clock(clk),.ena(ena2),.net_val(z4),.activation(a4));
activation ac5 (.clock(clk),.ena(ena2),.net_val(z5),.activation(a5));

assign dout1=a4[N-1+Fix:Fix];
assign dout2=a5[N-1+Fix:Fix];

endmodule

```