**Trường Đại học Khoa Học Tự Nhiên**

**Khoa Điện Tử - Viễn Thông**

**---------------🙢🕮🕮🕮🙠---------------**

Ảnh có chứa vẽ

Mô tả được tạo tự động

**Seminar Tốt Nghiệp**

**Chủ đề:**

**Giảng viên: ThS. Đỗ Quốc Minh Đăng**

**Sinh viên thực hiện: Nguyễn Quốc An**

**Mã số sinh viên: 1720024**

*Tp. Hồ Chí Minh, ngày 20 tháng 06 năm 2021*

**LỜI CẢM ƠN**

Đầu tiên em xin chân thành cảm ơn các thầy các cô của khoa Điện tử - Viễn Thông đã cho em các kiến thức cần thiết trong quá trình 4 năm đại học vừa qua để em thực hiện đồ án tốt nghiệp lần này và có thêm nền tảng vững vàng giúp ích cho công việc của em sau này.

Tiếp đến em đặc biệt cảm ơn thầy Đỗ Quốc Minh Đăng người đã nhận lời dẫn dắt em thực hiện đồ án lần này, dưới sự hướng dẫn của thầy em đã hoàn thành được những mục tiêu đã đề ra với đồ án của mình và tích lũy thêm được nhiều kiến thức hữu ích. Một lần nữa em xin chân thành cảm ơn thầy Đăng và tất cả các thầy cô của khoa Điện tử - Viễn thông, những người đã diều dắt em trong chặng đường đại học của mình.

**MỤC LỤC**

[Chương 1: Tổng Quan Về RAM. 1](#_Toc73107942)

[1.1 Tổng quan về RAM. 1](#_Toc73107943)

[1.1.1 RAM là gì ? 1](#_Toc73107944)

[1.1.2 Quá trình hoạt động và phân loại RAM 2](#_Toc73107945)

[Chương 2: Tổng Quanvề SDRAM 4](#_Toc73107946)

[2.1 DRAM là gì ? 4](#_Toc73107947)

[2.2 Tổng quan về SDRAM 6](#_Toc73107948)

[2.2.1 SDRAM là gì ? 6](#_Toc73107949)

[2.2.2 Lịch sử phát triển SDRAM 6](#_Toc73107950)

[2.2.3 Phân loại SDRAM 7](#_Toc73107951)

[2.2.4 Nguyên lý hoạt động SDRAM 9](#_Toc73107952)

[2.2.5 Mảng bộ nhớ của SDRAM 10](#_Toc73107953)

[2.2.6 Bank trong DRAM 11](#_Toc73107954)

[2.2.7 Khái niệm DIMM (Dual In-line Memory Module) 12](#_Toc73107955)

[Chương 3: Tổ chức bộ nhớ và cách truy xuất SDRAM 15](#_Toc73107956)

[3.1 Các tín hiệu điều khiển 15](#_Toc73107957)

[3.2 Lệnh trong SDRAM 16](#_Toc73107958)

[3.3 Quá trình truy xuất dữ liệu trong SDRAM 19](#_Toc73107959)

[Chương 4: Giới thiệu phần mềm mô phỏng modelsim,ngôn ngữ thiết kế phần cứng verilog và giới hạn đề tài. 21](#_Toc73107960)

[4.1 Sơ lược về phần mềm mô phỏng modelsim. 21](#_Toc73107961)

[4.2 Sơ lược về ngôn ngữ thiết kế phần cứng Verilog. 22](#_Toc73107962)

[4.3 Giới hạn đề tài 22](#_Toc73107963)

[Chương 5: Thiết kế SDRAM với ngôn ngữ Verilog 23](#_Toc73107964)

[5.1 Thế kế một SDRAM điển hình (DUT) 23](#_Toc73107965)

**DANH MỤC HÌNH ẢNH**

Hình 1‑1 RAM trên một hệ thống SoC 2

Hình 2‑1 Một dòng DRAM của hãnh Hynix 4

Hình 2‑2 DRAM trong tổ chức PC 5

Hình 2‑3 Một dòng SDRAM của hãng Micron 6

Hình 2‑4 SDR SDRAM với khe cắm 3 múi 7

Hình 2‑5DDR SDRAM với khe cắm được chia thành 2 múi 8

Hình 2‑ 2‑6DDR2 SDRAM cũng thương tự DDR SDRAM 8

Hình 2‑7 Một thanh DDR3 SDRAM bọc vỏ tản nhiệt 8

Hình 2‑8 Chip RAM 9

Hình 2‑9 Một thanh Rambus với 2 rãnh liền trên khe cắm 9

Hình 2‑10 Tổ chức cơ DRAM. Mảng bộ nhớ DRAM là một lưới các ô lưu trữ, trong đó một bit dữ liệu là được lưu trữ tại mỗi giao điểm của một hàng và một cột. 10

Hình 2‑11 Cấu trúc DRAM với 4 mảng 11

Hình 2‑12 Hình mình họa một DIMM 12

Hình 2‑13 Hình minh họa một DIMM thực tế 13

Hình 3‑1 Tổng quan tổ chức bộ nhớ của SDRAM 15

Hình 4‑1 Phần mềm mô phỏng Mô Delsim 21

Hình 4‑2 Ngôn ngữ phần cứng verilog 22

Hình 5‑1 Thiết kế tổng quan phần cứng SDRAM 23

Hình 5‑2 Sơ đồ máy trạng thái cua bộ SDRAM controler 24

Hình 5‑3 Môi trường testbench để kiểm tra DUT thiết kế SDRAM 26

Hình 6‑1 Timing ghi giá trị vào SDRAM 27

Hình 6‑2 Timing đọc giá trị từ SDRAM 27

**DANH MỤC CÁC TỪ VIẾT TẮT**

|  |  |  |
| --- | --- | --- |
| **Từ viết tắt** | **Diễn giải** | **Dịch nghĩa** |
| **Data input** | Data input | Dữ liệu đầu ra |
| **Data output** | Data output | Dữ liệu đầu vào |
| **FSM** | Finite-state Machine | Sơ đồ máy trạng thái |
| **RAM** | Random Access Memory | Bộ nhớ truy cập ngẫu nhiên |
| **DRAM** | Dynamic random-access memory | Bộ nhớ truy cập ngẫu nhiên động |
| **SRAM** | Synchronous dynamic random access memory | Bộ nhớ truy cập ngẫu nhiên động đồng bộ |
| **DIMM** | Dual In-line Memory Module | Mô đun bộ nhớ theo hàng |
| **SoC** | System on chip | Hệ thống trên một chip |
|  |  |  |
|  |  |  |
|  |  |  |

# Chương I. Tổng Quan Về Đề Tài

## Giới thiệu đề tài

Ngày này với sự phát triển của internet đã kéo theo sự phát triển mạnh mẽ của ngành thương mại điện tử, với nhịp sống hối hả thì việc dành thời gian để ra ngoài mua sắm trở nên là một điều quá xa sỉ, việc mua sắm online càng trở nên quan trọng.

Đề tài em chọn là xây dựng một website thương mại điện tử, nó cho phép các tổ chức hoặc cá nhân có nhu cầu bán hàng có thể đăng tải sản phẩm của mình lên website, các cá nhân có nhu cầu mua hàng thì có thể tham khảo lựa chọn mua các sản phẩm được ra bán trên website. Website được xây dựng theo mô hình MVC, sử dụng ASP.NET framework.

## Mục đích nghiên cứu

Hệ thống lại các kiến thức đã có và thích lũy thêm các kiến thức mới, rèn luyện khả năng giải quyết vấn đề.

## Mục tiêu đề tài

Xây dựng được một website với các tính năng cơ bản mà một trang thương mại điện tử cần có.

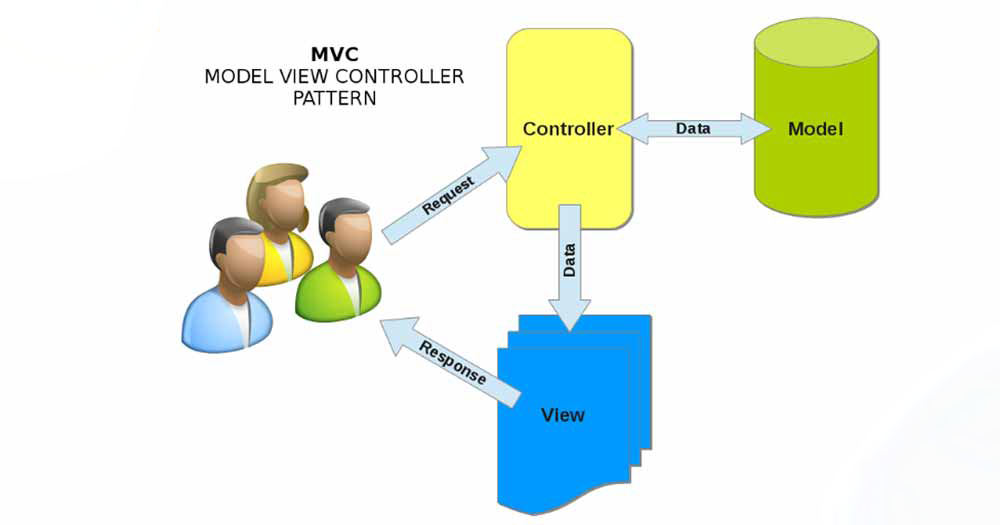
# Chương II. Cơ Sở Lý Thuyết

## Tổng quan về ASP.NET Framework

### Giới thiệu về ASP.NET Framework

ASP.NET framework là một nền tảng ứng dụng web (web application framework) được phát triển và cung cấp bởi Microsoft, cho phép những người lập trình tạo ra những trang web động, những ứng dụng web và những dịch vụ web. Lần đầu tiên được đưa ra thị trường vào tháng 2 năm 2002 cùng với phiên bản 1.0 của .NET framework, là công nghệ nối tiếp của Microsoft's Active Server Pages(ASP). ASP.NET được biên dịch dưới dạng Common Language Runtime (CLR), cho phép những người lập trình viết mã ASP.NET với bất kỳ ngôn ngữ nào được hỗ trợ bởi .NET language, trong đồ án lần này thì ngôn ngữ em sử dụng là C# và .NET framework phiên bản 4.8.

### Mô hình MVC trong ASP.NET

**MVC** là từ viết tắt bởi 3 từ **Model – View – Controller**. Đây là mô hình thiết kế sử dụng trong kỹ thuật phần mềm. Mô hình source code thành 3 phần, tương ứng mỗi từ. Mỗi từ tương ứng với một hoạt động tách biệt trong một mô hình.

##### Hình 1-1 Mô hình MVC và các thành phần bên trong

**Trong đó:**

**Model (M):** Là bộ phận có chức năng lưu trữ toàn bộ dữ liệu của ứng dụng. Bộ phận này là một cầu nối giữa 2 thành phần bên dưới là View và Controller. Model thể hiện dưới hình thức là một cơ sở dữ liệu hoặc có khi chỉ đơn giản là một [file XML](https://monamedia.co/file-xml-la-gi/) bình thường. Model thể hiện rõ các thao tác với cơ sở dữ liệu như cho phép xem, truy xuất, xử lý dữ liệu,…

**View (V):** Đây là phần giao diện (theme) dành cho người sử dụng. Nơi mà người dùng có thể lấy được thông tin dữ liệu của MVC thông qua các thao tác truy vấn như tìm kiếm hoặc sử dụng thông qua các website.Thông thường, các ứng dụng web sử dụng MVC View như một phần của  hệ thống, nơi các thành phần [HTML](https://monamedia.co/html-va-css-la-gi/) được tạo ra. Bên cạnh đó, View cũng có chức năng ghi nhận hoạt động của  người dùng để tương tác với Controller. Tuy nhiên, View không có mối quan hệ trực tiếp với Controller, cũng không được lấy dữ liệu từ Controller mà chỉ hiển thị yêu cầu chuyển cho Controller mà thôi.

Ví dụ: Nút “delete” được tạo bởi View khi người dùng nhấn vào nút đó sẽ có một hành động trong Controller.

**Controller (C):** Bộ phận có nhiệm vụ xử lý các yêu cầu người dùng đưa đến thông qua view. Từ đó, C đưa ra dữ liệu phù hợp với người dùng. Bên cạnh đó, Controller còn có chức năng kết nối với model.

## Tổng quan về SQL server và Entity framework

### SQL server

Trong đồ án lần này em chọn SQL server để quản lý database vì có sự tương thích cao với công nghệ .NET do cùng được phát triển bởi Microsoft.

SQL Server chính là một hệ quản trị dữ liệu quan hệ sử dụng câu lệnh SQL để trao đổi dữ liệu giữa máy cài SQL Server và máy Client. Một Relational Database Management System – RDBMS gồm có: databases, datase engine và các chương trình ứng dụng dùng để quản lý các bộ phận trong RDBMS và những dữ liệu khác.

### Entity framework

Entity Framework được phát hành lần đầu tiên vào năm 2008, nó là phương tiện tương tác chính giữa các ứng dụng .NET và cơ sở dữ liệu quan hệ.

Entity Framework là một ánh xạ quan hệ đối tượng (ORM), là một loại công cụ giúp đơn giản hóa ánh xạ giữa các đối tượng trong phần mềm với các bảng và cột của cơ sở dữ liệu quan hệ.

* Entity Framework (EF) là một framework ORM mã nguồn mở độc lập với .NET Framework.
* Entity Framework đảm nhiệm việc tạo các kết nối cơ sở dữ liệu và thực thi các lệnh, cũng như lấy kết quả truy vấn và tự động ánh xạ các kết quả đó thành các đối tượng trong ứng dụng của bạn.
* Entity Framework giúp theo dõi các thay đổi của đối tượng và cập nhật các thay đổi đó trở lại cơ sở dữ liệu cho bạn.

## Ajax

**AJAX là chữ viết tắt của Asynchronous JavaScript and XML,**AJAX = Asynchronous JavaScript and XML**.** Đây là một công nghệ giúp chung ta tạo ra những Web động mà hoàn toàn không reload lại trang nên rất mượt và đẹp. Trong đó:

* **Asynchronous**, hay nói ngắn hơn là *Async* – *bất đồng bộ*. Bất đồng bộ có nghĩa là một chương trình có thể xử lý không theo tuần tự các hàm. Sẽ không có quy trình, có thể nhảy đi bỏ qua bước nào đó. Ích lợi dễ thấy nhất của bất đồng bộ là chương trình có thể xử lý nhiều công việc một lúc.
* **JavaScript**là một ngôn ngữ lập trình nổi tiếng. Trong số rất nhiều chức năng của nó là khả năng quản lý nội dung động của website và hỗ trợ tương tác với người dùng.
* **XML** là một dạng của ngôn ngữ markup như HTML, chữ đầy đủ của nó là eXtensible Markup Language. Nếu HTML được dùng để hiển thị dữ liệu, XML được thiết kế để chứa dữ liệu

Ajax là cách mà chúng ta xử lý dữ liệu tại một số phần nhỏ trên ứng dụng web mà không cần phải load lại toàn bộ trang web

Cả **JavaScript** và **XML** đều hoạt động bất đồng bộ trong **AJAX**. Kết quả là, nhiều ứng dụng web có thể sử dụng AJAX để gửi và nhận data từ server mà không phải toàn bộ trang.

## Html

HTML viết tắt của chữ HyperText Markup Language (ngôn ngữ đánh dấu siêu văn bản), nó được dùng để tạo ra các tài liệu hiện thị được trên trình duyệt. HTML gọi là ngôn ngữ nhưng nó không giống như các ngôn ngữ kịch bản, ngôn ngữ lập trình (sử dụng các kịch bản để thi hành các tác vụ), ngôn ngữ đánh dấu (Markup) HTML lại sử dụng các THẺ để xác định nội dung của trang

HTML sử dụng cách đánh dấu (markup) để chú thích cho các thành phần (phần tử HTML) như văn bản, hình ảnh... các phần tử HTML tạo thành trang tài liệu hiện thị được trong các trình duyệt. Có rất nhiều phần tử HTML như <p>, <a>, <img>, <title>, <body> ...

Một phần tử HTML bắt đầu bằng mở thẻ (viết tên thẻ như p, a, img ... giữa < và >), đến nội dung phần tử và kết thúc bằng đóng thẻ (viết tên thẻ giữa </ và >

## Css

CSS là chữ viết tắt của Cascading Style Sheets, nó là một ngôn ngữ được sử dụng để tìm và định dạng lại các phần tử được tạo ra bởi các ngôn ngữ đánh dấu (ví dụ như HTML). Bạn có thể hiểu đơn giản rằng, nếu HTML đóng vai trò định dạng các phần tử trên website như việc tạo ra các đoạn văn bản, các tiêu đề, bảng,…thì CSS sẽ giúp chúng ta có thể thêm một chút “phong cách” vào các phần tử HTML đó như đổi màu sắc trang, đổi màu chữ, thay đổi cấu trúc,…rất nhiều.

Phương thức hoạt động của CSS là nó sẽ tìm dựa vào các vùng chọn, vùng chọn có thể là tên một thẻ HTML, tên một ID, class hay nhiều kiểu khác. Sau đó là nó sẽ áp dụng các thuộc tính cần thay đổi lên vùng chọn đó.

Tác dụng của CSS

* Hạn chế tối thiểu việc làm rối mã HTML của trang Web bằng các thẻ quy định kiểu dáng (chữ đậm, chữ in nghiêng, chữ có gạch chân, chữ màu), khiến mã nguồn của trang Web được gọn gàng hơn, tách nội dung của trang Web và định dạng hiển thị, dễ dàng cho việc cập nhật nội dung.
* Tạo ra các kiểu dáng có thể áp dụng cho nhiều trang Web, giúp tránh phải lặp lại việc định dạng cho các trang Web giống nhau.

# Chương III. Phân Tích & Thiết Kế Hệ Thống

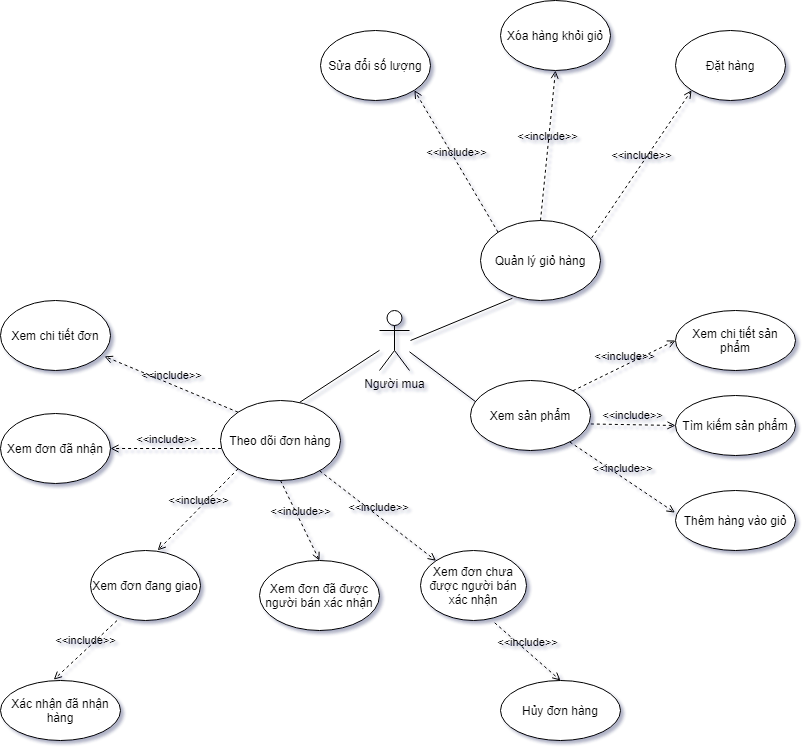
## Mô tả chức năng

Trang web sẽ có các thành phần user bao gồm: người mua hàng, người bán hàng và người thủ kho.

* Người mua: có thểm thêm hàng vào giỏ hàng, đặt hàng, theo dõi đơn hàng, hủy đơn hàng khi đơn hàng chưa được người bán xác nhận gửi đi.
* Người bán: có chức năng đăng sản phẩm, các thông tin liên quan về sản phẩm lên web, duyệt các đơn hàng mới khi bắt đầu gửi hàng về kho tập kết.
* Thủ kho: sẽ có chức năng duyệt đơn hàng và xuất file excel về thông tin, chi tiết đơn hàng khi mà đã nhận được hàng của người bán gửi về kho tập kết hàng.

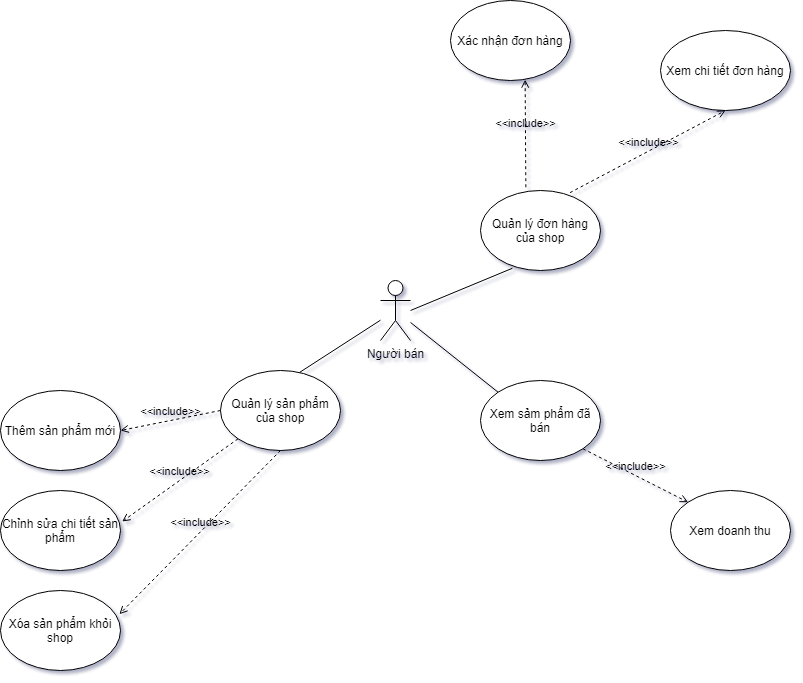
## Sơ đồ Use case

### Use case người mua



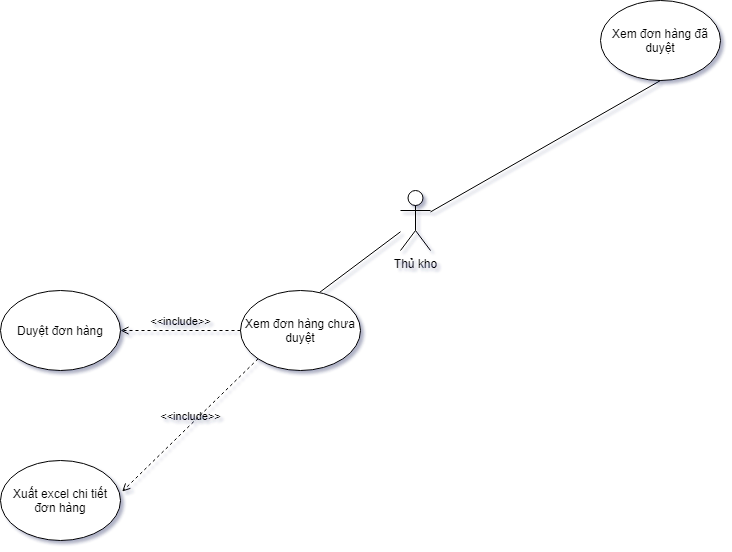
##### Hình 1.1 Sơ đồ Use Case mô tả chức năng người mua hàng

### Use case người bán



##### Hình 1.2 1 Sơ đồ Use Case mô tả chức năng người bán hàng

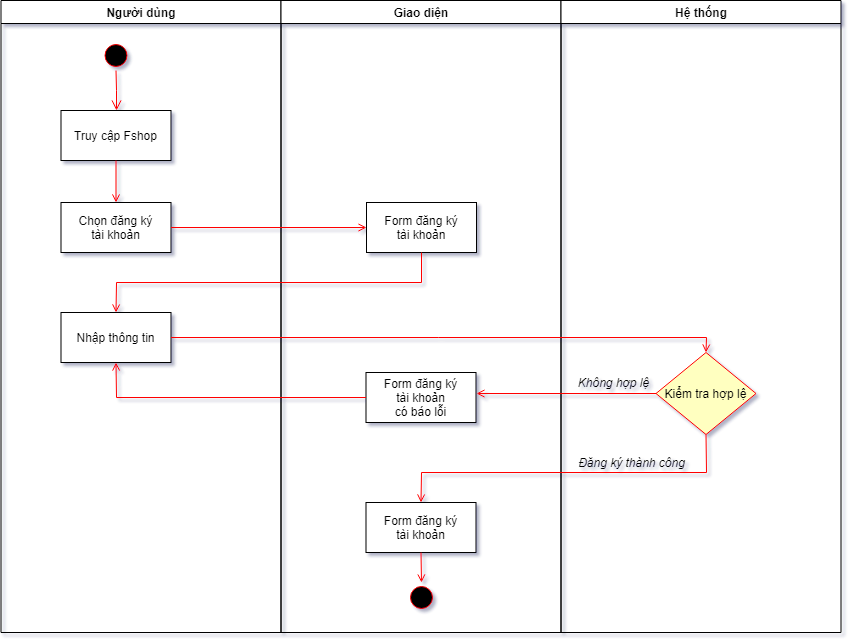
### Use case người thủ kho



##### Hình 1.3 Sơ đồ Use case mô tả chức năng người thủ kho

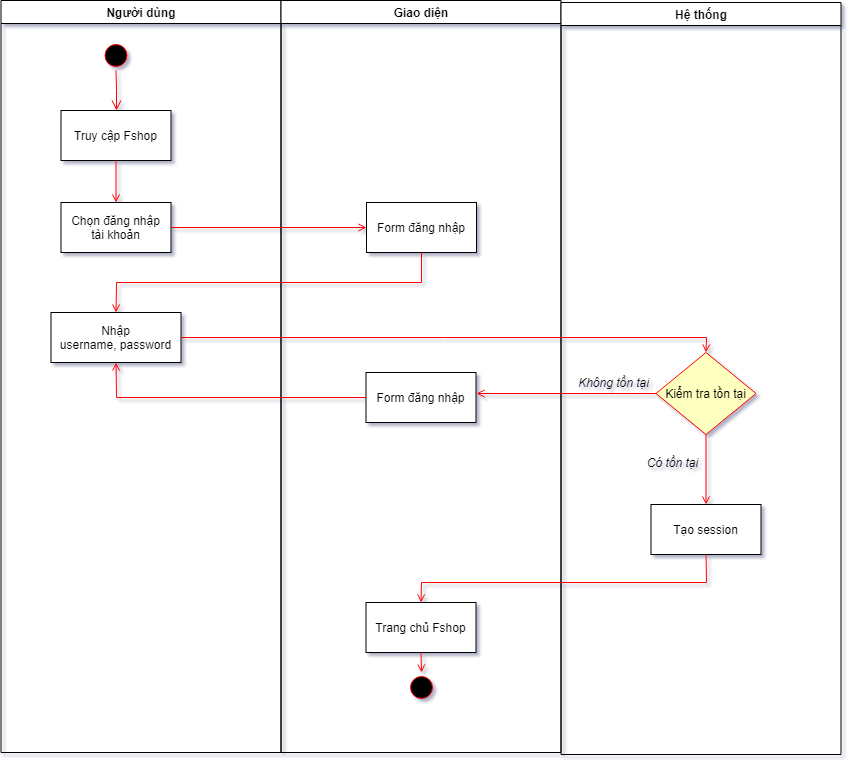
## Sơ đồ Activity

### Activity đăng ký



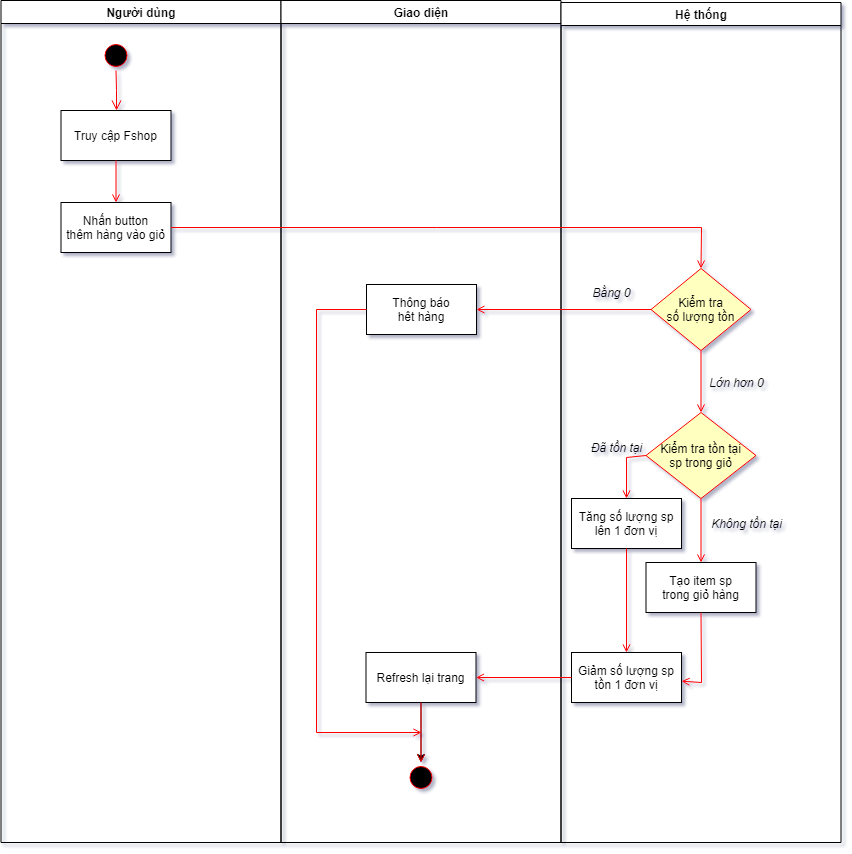
##### Hình 1.4 Sơ đồ activity chức năng đăng ký tài khoản

### Activity đăng nhập



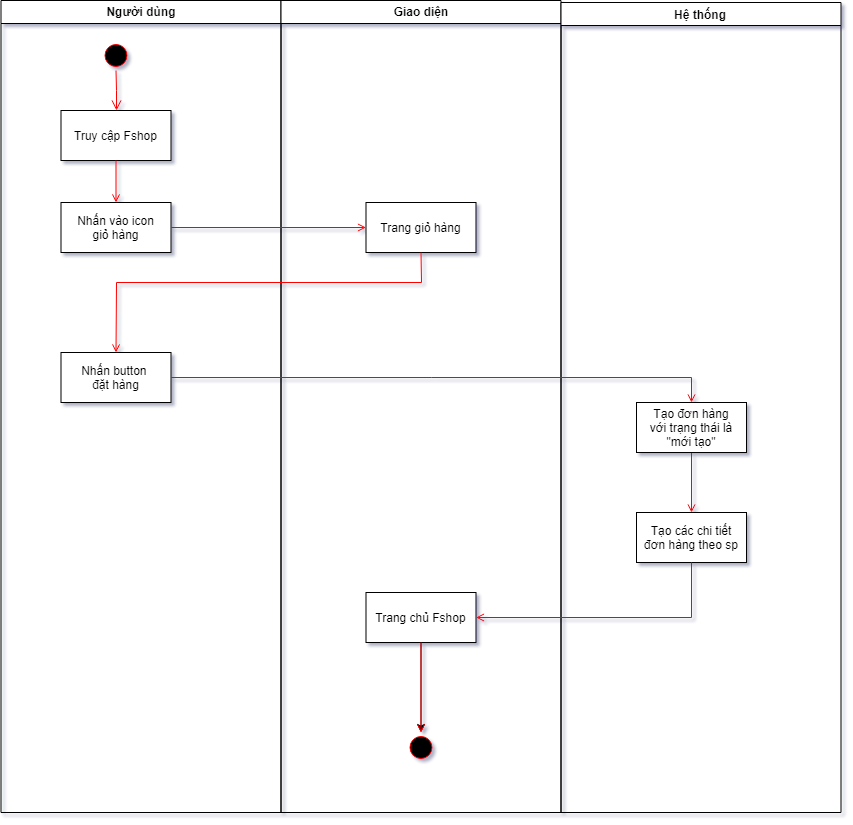
##### Hình 1.5 Sơ đồ activity chức năng đăng nhập

### Activity thêm hàng vào giỏ hàng



##### Hình 1.6 Sơ đồ activity chức năng thêm hàng vào giỏ hàng

### Activity đặt hàng



##### Hình 1.7 Sơ đồ activity đặt hàng

(Random Access Memory) là một loại bộ nhớ chính của máytính. RAM được gọi là bộ nhớ truy câp ngẫu nhiên vì đặc tính thờigian thực hiện thao tác đọc hoặc ghi đối với mỗi ô nhớ là như nhau, cho dùđang ở bất kỳ vị trí nào trong bộ nhớ là một byte (8 bit) tuy nhiên hệ thốnglại có thể đọc ghi vào nhiều byte (2, 4, 8 byte).

RAM khác biệt với các thiết bị bộ nhớ tuần tự (sequential memorydevice) chẳng hạn như các băng từ, đĩa các loại thiết bị này bắt buộcmáy tính phải di chuyển cơ học một cách tuần tự để truy cập dữ liệu. Cóthể lấy ví dụ có 100 ngôi nhà được đánh địa chỉ từ 1 đến 100m lần lượttheo chiều dọc. Với cách truy cập tuần tự muốn đi tới nhà thứ 99, cần phảiđi qua từng nhà từ nhà 1, 2, 3……. cho đến nhà thứ 99. Nhưng vớiphương thức truy cập ngẫu nhiên, có thể đi ngay đến ngôi nhà thứ 99 màkhông cần phải đi qua các ngôi nhà trước đó.Bởi vì các chip RAM có thể đọc ghi dữ liệu nên thuật ngữ RAM cũngđược hiểu như là một bộ nhớ đọc ghi, trái ngược với bộ nhớ chỉ đọc ROM(read-only memory).

RAM thông thường được sử dụng cho bộ nhớ chính (main memory)trong máy tính để lưu trữ các thông tin thay đổi, và các thông tin được sử dụng cho nhiều mục đích. RAM được tìm thấy ở bất kỳ đâu trong các thiết bị điện tử như máy chủ, PC, máy tính bảng, điện thoại thông minh hay máy in.



##### Hình 1‑1 RAM trên một hệ thống SoC

### Quá trình hoạt động và phân loại RAM

Trong điện thoại, máy tính, bộ nhớ RAM dùng để phối hợp với bộ nhớ máy tính điều khiển, truy cập, và sử dụng dữ liệu. Lúc này CPU chuyển dữ liệu từ ổ đĩa vào RAM để lưu trữ tạm thời, các vùng nhớ đã chiếm chỗ trên RAM sẽ được trả lại khi người dùng tắt ứng dụng hoặc tắt máy.

RAM được chia làm 2 loại SRAM và DRAM. SRAM hay còn gọi là RAM tĩnh (Static RAM) loại RAM này không bị mất nội dung sau khi nạp trừ khi khởi động máy tính, nó được ứng dụng vào lưu trữ dữ liệu khởi động. Khác với SRAM, DRAM được sử dụng để lưu trữ dữ liệu tạm thời khi chạy ứng dụng và có thể bị trả lại vùng nhớ chẳng hạn như với RAMtrên maý tính khi đóng ứng dụng hoặc shutdown hoặc turn off điện thoại hoặc máy tính thì dữ liệu trong các vùng RAM này mất đi và vùng nhớ đó trong RAM được giải phóng.

SRAM chạy nhanh hơn DRAM. Nhiều người có thể lầm lẫn là DRAM là "dynamic" cho nên ưu việt hơn. Điều đó không đúng. Trên thực tế, chế tạo SRAM tốn kém hơn hơn DRAM và SRAM thường có kích cỡ lớn hơn DRAM, nhưng tốc độ nhanh hơn DRAM vì không phải tốn thời gian refresh nhiều lần. Sự ra đời của DRAM chỉ là một lối đi vòng để hạ giá sản xuất của SRAM.

Bảng tóm tắt ưu và nhược điểm của DRAM và SRAM như bên dưới

|  |  |  |
| --- | --- | --- |
|  | SRAM | DRAM |
| Ưu điểm | * Thông tin ổn định. * Tốc độ nhanh. * Dung lượng IC nhỏ. | * Giá thành thấp |
| Nhược điểm | * Giá thành cao | * Thông tin không ổn định. * Tốc độ chậm. * Dung lượng IC lớn. |

# Tổng Quanvề SDRAM

## DRAM là gì ?

DRAM là viết tắt của Dynamic random-access memory (Bộ nhớ truy cập ngẫu nhiên động hay RAM động một loại bộ nhớ được sử dụng rộng rãi trên các hệ thống máy tính như là bộ nhớ chính. Xét về công suất, nó có thể đạt được 8GB cho mỗi chip trong IC hiện đại.

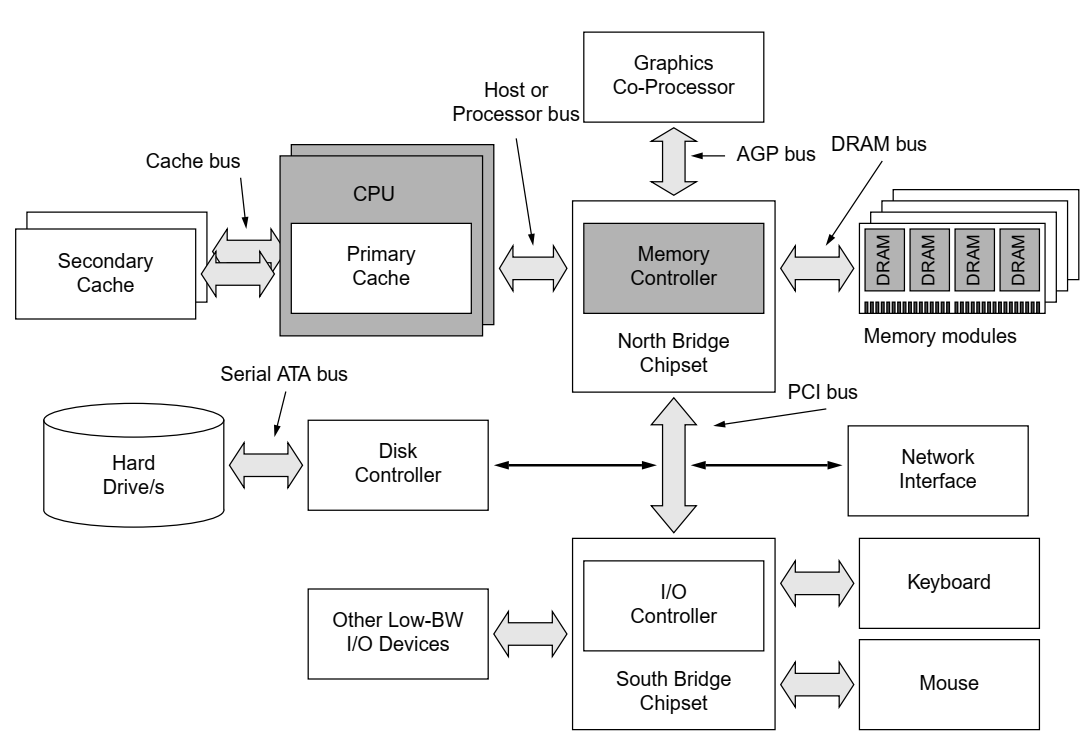
Về mặt vật lý, DRAM lưu trữ mỗi bit dữ liệu trong một tụ điện riêng bên trong mạch tích hợp. Tụ điện có thể được nạp hoặc xả, hai trạng thái này đại diện cho hai trạng thái của một bit (thường được gọi là 0 và 1). DRAM được gọi là động vì nó cần được làm tươi (refresh) hoặc nạp điện sau khoảng vài mili giây để bù đắp cho sự rò rỉ điện từ tụ điện. Vì tụ điện bị rò điện nên nếu không được làm tươi hoặc nạp điện lại đều đặn thì các bit dữ liệu lưu trên DRAM sẽ mất dần.

RAM truyền thống trên máy tính đều là DRAM. Những máy tính mới hơn sử dụng DDR (Dual Data Rate - tạm dịch: tốc độ dữ liệu kép) DRAM để nâng cao hiệu suất. SDRAM cũng làm một dạng của công nghệ DRAM.



##### Hình 2‑1 Một dòng DRAM của hãnh Hynix

Bên dưới là hình ảnh của một tổ chức PC điển hình mà DRAM là một phần trong tổng thể phức tạp đó. Có thể thấy rằng DRAM giao tiếp với CPU và các ngoại vi (chuột, bàn phím , đĩa cứng...) thông qua các bus 2 chiều.

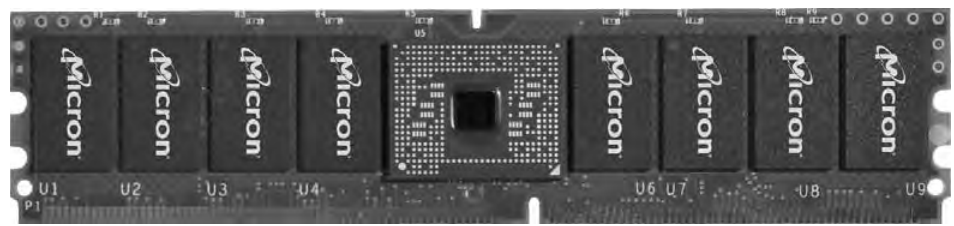


##### Hình 2‑2 DRAM trong tổ chức PC

## Tổng quan về SDRAM

### SDRAM là gì ?

SDRAM là viết tắt của Synchronous dynamic random access memory (Bộ nhớ truy cập ngẫu nhiên động đồng bộ). SDRAM là DRAM được đồng bộ hóa với bus hệ thống. Đây là tên chung cho những loại DRAM khác nhau được đồng bộ hóa với tốc độ xung nhịp tối ưu của vi xử lý. SDRAM có thể chạy ở tốc độ xung nhịp cao hơn nhiều so với bộ nhớ thông thường. Được đồng bộ với bus của CPU và có khả năng chạy ở mức 133MHz, nhanh hơn khoảng 3 lần so với RAM FPM thông thường, nhanh hơn DRAM EDO và DRAM BEDO khoảng 2 lần. SDRAM đang dần thay thế DRAM EDO trong nhiều máy tính đời mới.



##### Hình 2‑3 Một dòng SDRAM của hãng Micron

### Lịch sử phát triển SDRAM

Ý tưởng cơ bản đằng sau SDRAM đã tồn tại trong nhiều năm. Những ý tưởng đầu tiên xuất hiện ngay từ những năm 1970. Khái niệmSDRAM cũng được sử dụng trong một số bộ vi xử lý Intel đời đầu.

Một trong những sản phẩm SDRAM thương mại đầu tiên là KM48SL2000 được Samsung giới thiệu vào năm 1993. Mặc dù không nhận được sự chấp nhận rộng rãi ngay lập tức nhưng sự hấp thụ tương đối nhanh chóng. Do tốc độ được cải thiện của SDRAM nên khoảng đầu thế kỷ, tức là năm 2000, SDRAM hầu như đã thay thế công nghệ DRAM tiêu chuẩn trong hầu hết các ứng dụng máy tính.

Để đảm bảo công nghệ SDRAM có thể hoán đổi cho nhau, JEDEC, cơ quan công nghiệp về tiêu chuẩn bán dẫn, đã thông qua tiêu chuẩn SDRAM đầu tiên vào năm 1993. Điều này tạo điều kiện cho một tiêu chuẩn chung mở để phát triển SDRAM. Nó cũng cho phép các nhà phát triển có thể có phương tiện sử dụng sản phẩm từ nhiều nhà sản xuất và có tùy chọn nguồn thứ hai khả thi.

Từ khi SDRAM cơ bản được thiết lập, các bước phát triển tiếp theo đã diễn ra. Một dạng SDRAM được gọi là tốc độ dữ liệu gấp đôi, DDR SDRAM xuất hiện vào năm 2000 với bản phát hành 1 được cập nhật lên bản phát hành 2 vào tháng 5 năm 2002 và sau đó là bản phát hành C vào tháng 3 năm 2003.

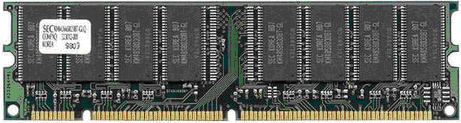
DDR SDRM được theo sau bởi phiên bản tiếp theo có tên là DDR2 SDRAM. Nó được giới thiệu lần đầu tiên vào giữa năm 2003 khi có hai tốc độ xung nhịp: 200 MHz (gọi tắt là PC2-3200) và 266 MHz (PC2-4200). Các sản phẩm đầu tiên của DDR2 SDRAM kém hơn so với DDR SDRAM trước đó, nhưng vào cuối năm 2004, hiệu suất của nó đã được cải thiện vượt xa các định dạng DDR.

Sau đó, phiên bản tiếp theo của SDRAM đã được đưa ra. Được biết đến với cái tên DDR3 SDRAM, các nguyên mẫu đầu tiên được công bố vào đầu năm 2005. Tuy nhiên, phải đến giữa năm 2007, các bo mạch chủ máy tính đầu tiên sử dụng DDR3 mới có mặt.Những phát triển tiếp theo của SDRAM là DDR4 SDRAM.

### Phân loại SDRAM

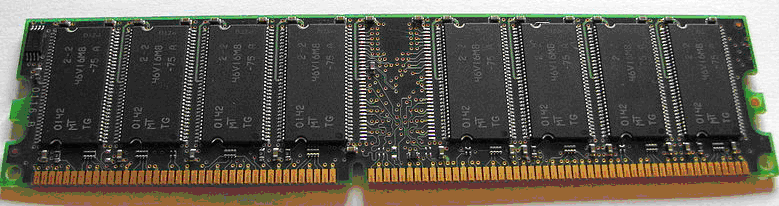
SDRAM lại gồm 3 loại chính là SDRRAM, DDRRAM và một số thế hệ sau của DDR.

***- SDR SDRAM*** (Single Data Rate SDRAM) thường được gọi là “SDR”, có 168 chân. Được dùng trong các máy vi tính cũ, có bus speed chạy cùng với clock speed của memory chip nay đã nỗi thời.



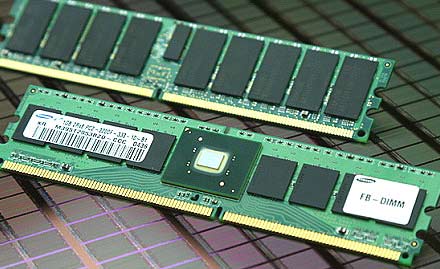
##### Hình 2‑4 SDR SDRAM với khe cắm 3 múi

**- DDR SDRAM** (Double Data Rate SDRAM) còn được gọi tắt là “DDR”, có 184 chân. DDR là cải tiến của SDR, có tốc độ gấp đôi SDR nhờ vào việc truyền tải 2 lần trong một chu kỳ bộ nhớ.



##### Hình 2‑5DDR SDRAM với khe cắm được chia thành 2 múi

**- DDR2 SDRAM** (Double Data Rate 2 SDRAM) còn gọi tắt là DDR2, có 240 chân. Là thế hệ sau của DDR, có tốc dộ truyền tải gấp đôi DDR vì bus speed cao gấp đôi clock speed. DDR2 sử dụng mức điện áp 1,8V.



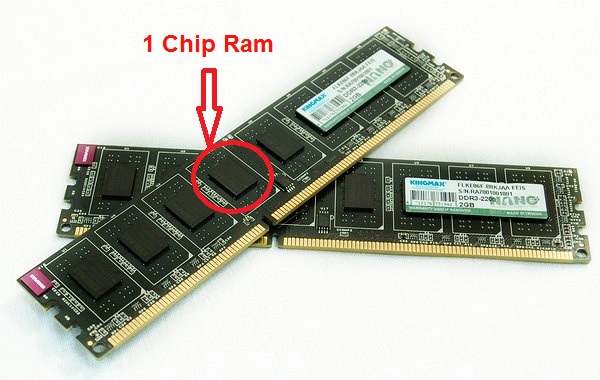
##### Hình 2‑ 2‑6DDR2 SDRAM cũng thương tự DDR SDRAM



##### Hình 2‑7 Một thanh DDR3 SDRAM bọc vỏ tản nhiệt

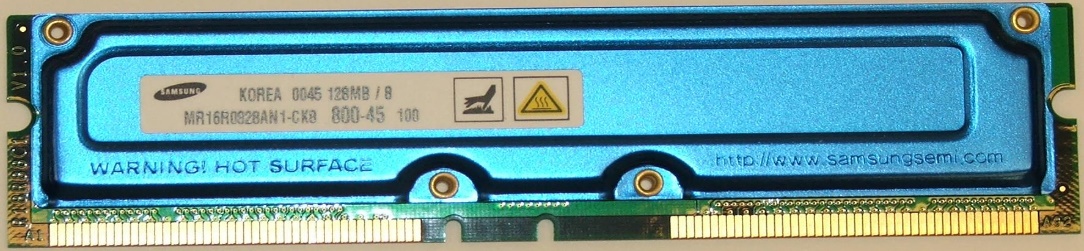
**- DDR3 SDRAM** (Double Data Rate 3 SDRAM) cũng có 240 chân như DDR2 SDRAM song nó lại có tốc độ bus cao hơn DDR2 nhiều. Mức điện áp quy đinh đối với DDR3 là 1,5V. Có tốc độ bus là 800/1066/1333/1600 Mhz, số bit dữ liệu là 64bits. Chuẩn DDR3 cho phép sử dụng với các chip nhớ có dung lượng từ 512M tới 8G.

**- RDRAM** **(*RAMbus Dynamic RAM*)**: Được gọi tắt là "**Rambus**". Đây là một loại DRAM được thiết kế kỹ thuật hoàn toàn mới so với kỹ thuật SDRAM. RDRAM hoạt động đồng bộ theo một hệ thống lặp và truyền dữ liệu theo một hướng. Một kênh bộ nhớ RDRAM có thể hỗ trợ đến 32 chip DRAM. Mỗi chip được ghép nối tuần tự trên một module gọi là RIMM (RAMbus Inline Memory Module) nhưng việc truyền dữ liệu được thực hiện giữa các mạch điều khiển và từng chip riêng biệt chứ không truyền giữa các chip với nhau.



##### Hình 2‑8 Chip RAM

Bus bộ nhớ RDRAM là đường dẫn liên tục đi qua các chip và module trên bus, mỗi module có các chân vào và ra trên các đầu đối diện. Do đó, nếu các khe cắm không chứa RIMM sẽ phải gắn một module liên tục để đảm bảo đường truyền được nối liền. Tốc độ RAMbus đạt từ 400-800MHz. RAMbus tuy không nhanh hơn SDRAM là bao nhưng lại đắt hơn rất nhiều nên có rất ít người dùng. RDRAM phải cắm thành cặp và ở những khe trống phải cắm những thanh RAM giả cho đủ.



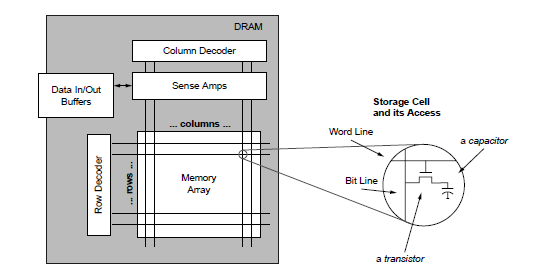
##### Hình 2‑9 Một thanh Rambus với 2 rãnh liền trên khe cắm

### Nguyên lý hoạt động SDRAM

Bộ nhớ truy cập ngẫu nhiên (RAM) sử dụngmột cặp tụ điện- transistor duy nhất cho mỗi bit. Ở hình bên phải bên dưới cho thấymạch cho ô lưu trữ trong DRAM. Vì các tụ điện lưu trữ các electronkhông phải là thiết bị hoàn hảo và sẽ rò rỉ trong quá trình nạp xả vì thế để giữ lại thông tin được lưu trữ ở đó, mỗitụ điện trong DRAM phải được làm mới định kỳ(tức là đọc và viết lại). Điều này lí giải cho việc đọc xóa liên tục của DRAM. DRAM có thể chứa một hoặc nhiều bộ nhớmảng, với mỗi ô (cell) lưu giữ một bit dữ liệu.

Một ví dụ để hiểu hoạt động của DRAMcó thể hình dung đường phốvới các đại lộ chạybắc - nam và các đường phố chạy theo hướng đông - tây.Khi một người muốn chỉ định một địa điểm hẹn trongmột thành phố như vậy, người ta chỉ đơn giản chỉ định giao lộ củamột đường phố và một đại lộ cũng tương tự như việc mảng bộ nhớ được tổ chứcthành hàng và cột.

Bằng cách xác định giao điểm của một hàngvà một cột (bằng cách chỉ định địa chỉ hàng và cộtđịa chỉ tới DRAM), bộ điều khiển bộ nhớ có thểtruy cập vào một ô lưu trữ riêng lẻ bên trong một chip DRAMđể đọc hoặc ghi dữ liệu được lưu giữ ở đó.



##### Hình 2‑10 Tổ chức cơ DRAM. Mảng bộ nhớ DRAM là một lưới các ô lưu trữ, trong đó một bit dữ liệu là được lưu trữ tại mỗi giao điểm của một hàng và một cột.

### Mảng bộ nhớ của SDRAM

Các DRAM hiện đại được thiết kế không chỉ gồm 1 bộ nhớ (memory) riêng rẽ là mà tập hợp của nhiều bộ nhớ mà ta gọi là mảng bộ nhớ (memrory array). Các mảng bộ nhớ trongchip nhớ có thể hoạt động theo nhiều cách khác nhau có thể hoạt động đồng bộ (unison), có thể hành động hoàn toàn độc lập (completely independently),hoặc có thể hoạt động cùng lúc theo cả 2 cách trên.

Nếu mảng bộ nhớđược thiết kế để hoạt động đồng bộ, chúng hoạt động như một đơn vị,và chip bộ nhớ thường truyền hoặc nhậnmột số bit bằng số mảng mỗithời gian bộ điều khiển bộ nhớ truy cập DRAM. Ví dụ trong một tổ chức đơn giản, DRAM x4 cho biết DRAM có lúcít nhất bốn mảng bộ nhớ và chiều rộng cột là4 bit (mỗi cột đọc hoặc ghi truyền 4 bitdữ liệu). Trong phần DRAM x4, bốn mảng, mỗi mảng đọc 1 dữ liệubit đồng thời và một phần gửi ra 4 bit dữ liệumỗi lần bộ nhớ được truy xuất. Ngày nay chúng ta cũng có nhiều định dạng khác như DRAMx4, DRAMx8, DRAMx16 ... được phổ biến và sử dụng rộng rãi.



##### Hình 2‑11 Cấu trúc DRAM với 4 mảng

### Bank trong DRAM

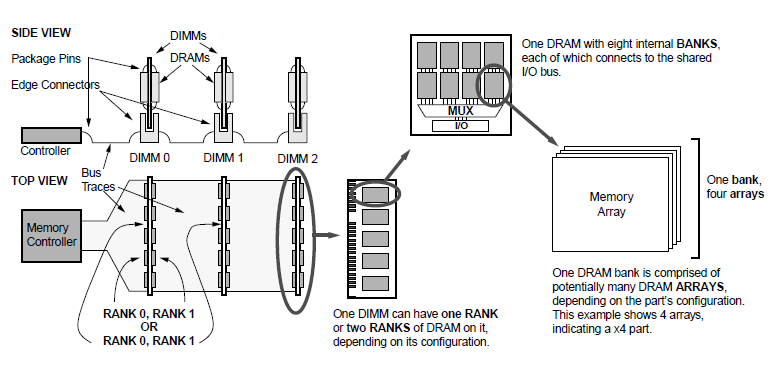
Các mảng bộ nhớ được truy xuất thông qua các bank. Bank chính là một khái niệm phát minh tuyệt vời được sinh ra để quản lí các mảng bộ nhớ giúp tăng băng thông bus thay .*Mỗi bộ mảng bộ nhớ hoạt động độc lập với tập hợp được gọi là một ngân hàng,* không phải một mảng. Mỗi bank hoạt động đôc lậptheo một giản đồ trang thái.Một số trạng thái trong giản đồ như activated, precharged, read ... v.v. cùng lúc đó các bank khác (trên cùng một thiết bị DRAM hoặc trêncác thiết bị DRAM khác) cũng đang được activated, precharged... tương tự.

Việc sử dụng nhiều bankđể quản lí các bộ nhớ độc lậpđã là một thực tế phổ biến trong thiết kế máy tínhkể từ khi DRAM được phát minh mục đích để đạt được các bus bộ nhớ băng thông cao và tận dụng tối đa hiệu suất sử dụng DRAM. Người ta còn đưa ra khái niệm hiệu suất sử dụng DRAM (tính theo phần trăm) để đo hiệu quả của việc truy cập.

### Khái niệm DIMM (Dual In-line Memory Module)

Một mô-đun bộ nhớ dual inline (DIMM) là một bảng mạch nhỏ chứa chip nhớ trên bo mạch chủ. DIMM kết hợp một loạt các bộ nhớ được gọi là bộ nhớ truy cập ngẫu nhiên động (DRAM), cung cấp lưu trữ chính, bộ nhớ chính mà liên tục đọc và thực thi được lưu trữ hướng dẫn hoặc dữ liệu trực tiếp tới CPU.

Các bank được quản lý bởi DIMM.Một hệ thống có thể có nhiều DIMM.Mỗi DIMM có thể chứa một hoặc nhiều bank độc lập và DIMM được xếp hạng theo thứ tự.Mỗi thứ hạng là một tập hợp các thiết bị DRAM được phân nhóm, mỗi thiết bị trong số đó có nhiều ngân hàng tiềm năng. Mỗi bank lại quản lí các mảng bộ nhớ tùy thuộc vào chiều rộng của bus dữ liệu.

Bên dưới là hình ảnh tổ chức của các DIMM trên lý thuyết và hình ảnh trong thực tế.

##### Hình 2‑12 Hình mình họa một DIMM



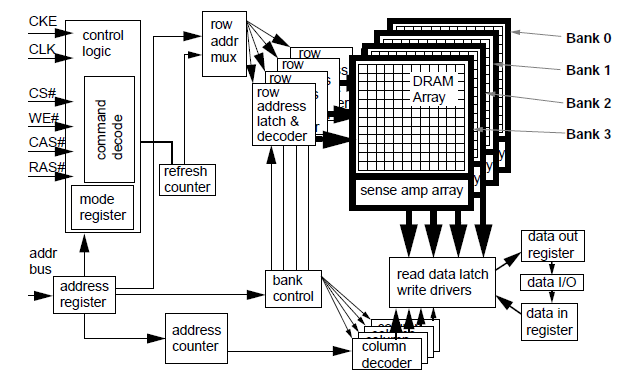
##### Hình 2‑13 Hình minh họa một DIMM thực tế

Các DIMM tiêu chuẩn phổ biến nhất, với chiều dài thông thường là 5,5 inch (14cm) và cao 1,18 inch (3cm)là:

* Unbuffered DIMM (UDIMM) - Được sử dụng chủ yếu trên máy tính để bàn và máy tính xách tay. Mặc dù chúng chạy nhanh và có chi phí thấp hơn, nhưng UDIM không ổn định như registered memory (RAM với một buffer giữa bộ nhớ và bus tới CPU). Các lệnh đi trực tiếp từ memory controller nằm trong CPU đến mô đun bộ nhớ.
* Fully-buffered DIMM (FB-DIMM) - Thường được sử dụng làm bộ nhớ chính trong các hệ thống yêu cầu dung lượng lớn, chẳng hạn như máy chủ và máy trạm. FB-DIMM sử dụng chip AMB để tăng độ tin cậy, duy trì tính toàn vẹn tín hiệu và cải thiện các phương pháp để phát hiện lỗi. Bus AMB được chia thành bus đọc 14 bit và bus ghi 10 bit. Có một bus đọc/ghi chuyên dụng nghĩa là việc đọc và ghi có thể xảy ra cùng một lúc, điều này dẫn đến hiệu suất tăng lên.
* Registered DIMM (RDIMM) - RDIMM thường được sử dụng trong các máy chủ, cũng như những ứng dụng đòi hỏi sự mạnh mẽ và ổn định khác. RDIMM có các thanh ghi bộ nhớ (memory register) trên bo mạch, được đặt giữa bộ nhớ và memory controller.
* Load-reduced DIMM (LR-DIMM) - Sử dụng công nghệ iMB (Isolation Memory Buffer). Chip iMB cô lập tất cả các electrical load (bộ phận hoặc phần của mạch tiêu thụ điện), bao gồm tín hiệu dữ liệu của các chip DRAM trên DIMM từ memory controller. Do đó, memory controller chỉ nhìn thấy iMB chứ không phải là các chip DRAM. Bộ nhớ đệm sau đó xử lý tất cả các lần đọc và ghi vào chip DRAM, giúp nâng cao dung lượng và tốc độ.
* SO-DIMM - Mặc dù DIMM tiêu chuẩn có dạng hình chữ nhật với chiều dài khoảng 14cm, nhưng SO-DIMM lại chỉ có kích thước bằng khoảng một nửa, với chiều dài 2,74 inch (7cm). SO-DIMM chủ yếu được sử dụng cho các thiết bị như máy tính xách tay và máy tính bảng. Nó khác với DIMM tiêu chuẩn ở chỗ DDR4 SO-DIMM có 260 pin (còn DIMM DRR4 có tới 288 pin). DIMM tiêu chuẩn được sử dụng trong PC và máy chủ.

# Tổ chức bộ nhớ và cách truy xuất SDRAM

Bên dưới là hình minh họa tổ chức một SDRAM thế hệ đầu tiên.|Ở những thế hệ sau như DDR2, DDR3, DDR4... cách tổ chức có thể khác đi chút ít nhưng vẫn thừa kế từ tổ chức ban đầu của SDRAM



##### Hình 3‑1 Tổng quan tổ chức bộ nhớ của SDRAM

## Các tín hiệu điều khiển

* **CKE (clock enable)**: Tín hiệu xung nhịp đồng bộ cho các tin hiệu trong khối SDRAM. Có thể đồng bộ ở cạnh lên hoặc cãnh xuống của xung clock tùy theo công nghệ SDRAM.
* **CS (chip select )** : tín hiệu chọn chip. Khi tín hiệu này ở mức cao, chip sẽ bỏ qua tất cả các đầu vào khác (ngoại trừ CKE) và hoạt động như thể nhận được lệnh NOP.
* **WE (Write enable):** Tín hiệu cho biết dữ liệu đang mong muốn ghi vào RAM.
* **RAS (Row Address Strobe):** Là tín hiệu để xác định địa chỉ nhớ theo hàng.
* **CAS (Column Address Strobe):** là tín hiệu để xác định địa chỉ nhớ theo cột.
* **Addr Bus (Address bus):** là đường truyền tín hiệu RAS và Cas.
* **Data Bus:** trong phạm vi hình vẽ trên không có signal data bus. Nhưng bus dữ liệu một phần mặc định luôn phải có khi truyền dữ liệu giữa Memory Controler và chip nhớ.
* **DQM (data mask):** (Chữ Q xuất hiện bởi vì, tuân theo các quy ước logic kỹ thuật số, các đường dữ liệu được gọi là đường "DQ".) Khi ở mức cao, các tín hiệu này sẽ triệt tiêu dữ liệu I / O. Khi dữ liệu ghi kèm theo, dữ liệu không thực sự được ghi vào DRAM. Khi xác nhận hai chu kỳ cao trước một chu kỳ đọc, dữ liệu đọc không được xuất ra từ chip. Có một dòng DQM trên 8 bit trên chip nhớ x16 hoặc DIMM. Khi cần truy xuất đến 1 địa chỉ nhớ bất kì Memory Controler sẽ gửi các tín hiệu RAS và CAS tương xứng đến Chip nhớ tương ứng với dữ liệu cần lấy.

## Lệnh trong SDRAM

SDRAM được truy xuất dựa theo trạng thái của các pin đầu vào (CS,RAS,SAC,WE) và các bus address (có thể khác nhau về độ rộng) để lựa chọn hàng cột như bên dưới. Tất cả các thế hệ SDRAM (SDR và ​​DDRx) về cơ bản sử dụng các lệnh giống nhau, với các thay đổi là:

* Các bit địa chỉ bổ sung để hỗ trợ các thiết bị lớn hơn
* Các bit chọn ngân hàng bổ sung
* Thanh ghi chế độ rộng hơn (DDR2 trở lên sử dụng 13 bit, A0 – A12)
* Các thanh ghi chế độ mở rộng bổ sung (được chọn bởi các bit địa chỉ ngân hàng)
* DDR2 xóa lệnh kết thúc cụm; DDR3 chỉ định lại nó là "hiệu chuẩn ZQ"
* DDR3 và DDR4 sử dụng A12 trong khi đọc và ghi lệnh để chỉ ra "burst chop", truyền dữ liệu nửa độ dài.
* DDR4 thay đổi mã hóa của lệnh kích hoạt . Một tín hiệu mới ACT điều khiển nó, trong đó các dòng điều khiển khác được sử dụng như các bit địa chỉ hàng 16, 15 và 14. Khi ACT ở mức cao, các lệnh khác tương tự như trên.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| CS | RAS | CAS | WE | BAn | A10 | An | Lệnh |
| H | X | X | X | X | X | X | **IDLE** (không hoạt động) |
| L | H | H | H | X | X | X | **NOP** (không hoạt động) |
| L | H | H | L | X | X | X | Chấm dứt liên tục: dừng quá trình đọc liên tục hoặc ghi liên tục đang diễn ra |
| L | H | L | H | BANK | L | COL | **READ**: đọc một loạt dữ liệu từ hàng hiện đang hoạt động |
| L | H | L | H | BANK | H | COL | **READ**: Đọc với tính năng tự động nạp trước: như trên và nạp trước (đóng hàng) khi hoàn |
| L | H | L | L | BANK | L | COL | **WRITE**: ghi một loạt dữ liệu vào hàng hiện đang hoạt động |
| L | H | L | L | BANK | H | COL | **WRITE** : ghi tính năng tự động nạp trước: như trên và nạp trước (đóng hàng) khi thực hiện xong hàng |
| L | L | H | H | BANK |  | | **Active** (kích hoạt): mở một hàng cho các lệnh đọc và ghi |
| L | L | H | L | BANK | L | X | **Precharge**: hủy kích hoạt (đóng) hàng hiện tại của bank đã chọn |
| L | L | H | L | X | H | X | **Precharge all:** hủy kích hoạt (đóng) dòng hiện tại hàng của tất cả các bank |
| L | L | L | H | X | X | X | **REFRESH**: làm mới một hàng của mỗi bank, sử dụng bộ đếm nội bộ. Tất cả các bank phải được tính phí trước. |
| L | L | L | L | 00 | MODE | | Thanh ghi chế độ tải: A0 đến A9 được tải để cấu hình chip DRAM. |

## Quá trình truy xuất dữ liệu trong SDRAM

Việc truy xuất SRAM dựa vào một bộ control logic như hình trên. Từ các tín hiệu đầu vào thông qua một bộ control logic sẽ tạo ra các command (lệnh) : READ, WRITE, ACTIVE, REFRESH, PRECHARGE kết hợp với bus đia chỉ để đóng mở các bank được sắp xếp theo hàng và cột. Việc đóng mở các bank này theo một trình tư sẽ giúp truy xuất dữ liêu memory trong SDRAM.

Đầu tiên lệnh ACTIVE sẽ kích hoạt một bank không hoạt động. Kết hợp với đia chỉ bank BAn và một địa chỉ hàng An bit. Thao này còn được gọi là "mở" bank. Thao tác này có tác dụng phụ là làm mới các ô lưu trữ bộ nhớ động (điện dung) của hàng đó.

Khi hàng đã được kích hoạt hoặc "mở", có thể READ (đọc) và WRITE (ghi) lệnh cho hàng đó. Việc kích hoạt yêu cầu một khoảng thời gian tối thiểu, được gọi là độ trễ hàng-cột, hoặc tRCD. Thời gian này, được làm tròn đến bội số tiếp theo của chu kỳ xung clock, chỉ định số chu kỳ chờ tối thiểu giữa một lệnh hoạt động và một lệnh đọc hoặc ghi. Trong các chu kỳ chờ này, các lệnh bổ sung có thể được gửi đến các bank khác; vì mỗi bank hoạt động hoàn toàn độc lập.

Khi lệnh READ được đưa ra, SDRAM sẽ tạo ra dữ liệu đầu ra tương ứng trên các dòng DQ tại cạnh lên của xung clock sau vài chu kỳ xung clock sau đó, tùy thuộc vào độ trễ CAS được cấu hình. Các word (32 bit) tiếp theo của cụm sẽ được tạo ra tương ứng tại cạnh lên của xung clock đồng hồ.

Một lệnh WRITE đi kèm với dữ liệu được ghi vào các dòng DQ trong cùng một cạnh lên của xung clock. Bộ control logic có nhiệm vụ đảm bảo rằng SDRAM không điều khiển dữ liệu đọc và ghi trên các dòng DQ cùng lúc mà nó cần điều khiển để tránh sự cạnh tranh. Điều này có thể được thực hiện bằng cách đợi cho đến khi một loạt đọc kết thúc, bằng cách kết thúc một loạt đọc hoặc bằng cách sử dụng dòng điều khiển DQM.

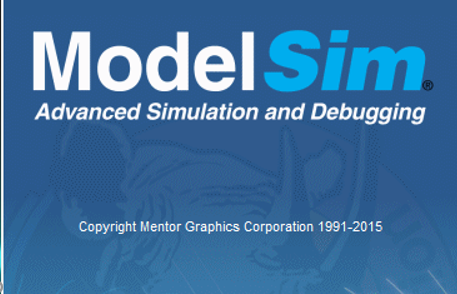
Khi bộ điều khiển bộ nhớ cần truy cập một hàng khác, trước tiên nó phải đưa bank đó về trạng thái nhàn rỗi để sẵn sàng nhận biết hàng tiếp theo. Đây được gọi là thao tác "nạp trước" Precharge hoặc "đóng" hàng. Việc nạp trước có thể được lệnh một cách rõ ràng, hoặc nó có thể được thực hiện tự động khi kết thúc thao tác đọc hoặc ghi. Một lần nữa, có một thời gian tối thiểu, độ trễ nạp trước hàng tRP , phải trôi qua trước khi hàng đó hoàn toàn "đóng" và do đó bank không hoạt động để nhận lệnh kích hoạt khác trên cùng bank đó.

Mặc dù làm mới một hàng REFRESH là một tác dụng phụ tự động của việc kích hoạt nó nhưng có một thời gian tối thiểu để điều này xảy ra, điều này yêu cầu thời gian truy cập hàng tối thiểu tRAS độ trễ giữa một lệnh hoạt động mở một hàng và lệnh nạp trước tương ứng để đóng nó. Giới hạn này thường bị thu hẹp bởi các lệnh đọc và ghi mong muốn vào hàng, vì vậy giá trị của nó có ít ảnh hưởng đến hiệu suất điển hình.

# Giới thiệu phần mềm mô phỏng modelsim,ngôn ngữ thiết kế phần cứng verilog và giới hạn đề tài.

## Sơ lược về phần mềm mô phỏng modelsim.

ModelSim là chương trình mô phỏng và debug cho các thiết kế ASIC và FPGA cực kỳ mạnh mẽ và hiệu quả, khi thiết kế FPGA chúng ta có thể dùng Modelsim để mô phỏng, kiểm tra, test chương trình…

ModelSim là môi trường debug cho các ngôn ngữ như Verilog HDL, VHDL, System C. ModelSim hiện nay có các phiên bản là PE,LE,SE,XE. ModelSim PE và LE dùng cho các thiết kế nhỏ và trung bình, trong khi bản SE dùng cho các thiết kế FPGA lớn hơn. Các phiên bản PE và SE có trả phí, phiên bản XE và phiên bản Student Edition là miễn phí.

##### Hình 4‑1 Phần mềm mô phỏng MoDelsim

## Sơ lược về ngôn ngữ thiết kế phần cứng Verilog.

Verilog là ngôn ngữ mô tả phần cứng (Hardware Description Language) được sử dụng trong việc thiết kế các các hệ thống số, các mạch tích hợp: như bộ nhớ RAM, bộ vi xử lý hoặc đơn giản là D-FlipFlop...Thiết kế số FPGA bằng ngôn ngữ verilog cũng giống như bạn viết chương trình cho vi điều khiển bằng ngôn ngữ ChoặcAssembly. Học Verilog không khó nếu bạn đẫ có nền tản về một ngôn ngữ nào đó, nó cũng giống tương tự C nên sẽ không khó khăn khi bắt đầu học Verilog. Ngoài Verilog còn có một ngôn ngữ mô tả phần cứng khá phổ biến khác là VHDL.

Verilog và VHDL là 2 ngôn ngữ mô tả phần cứng thông dụng nhất hiện nay, bạn có thể chọn một trong 2 ngôn ngữ để có thể học được FPGA hoặc cao hơn là ASIC. Trong phạm vi đồ án này sử dũng ngôn ngữ Verilog vì ngôn ngữ này tiếp cận khá dễ và đặc biệt cú pháp rất giống C, một khi học được Verilog bạn có thể học VHDL dễ dàng.



##### Hình 4‑2 Ngôn ngữ phần cứng verilog

## Giới hạn đề tài

Trong phạm vi đề tài này chủ yếu hướng đến việc đọc hiểu, nghiên cứu về SDRAM và thiết kết mô hình SDRAM sử dụng ngôn ngữ verilog sau đó chạy mô phỏng dạng sóng, timing trên phần mềm modelsim. Chưa tiến hành bước nap chạy trên system thực như các board, kit ASIC, FPGA...

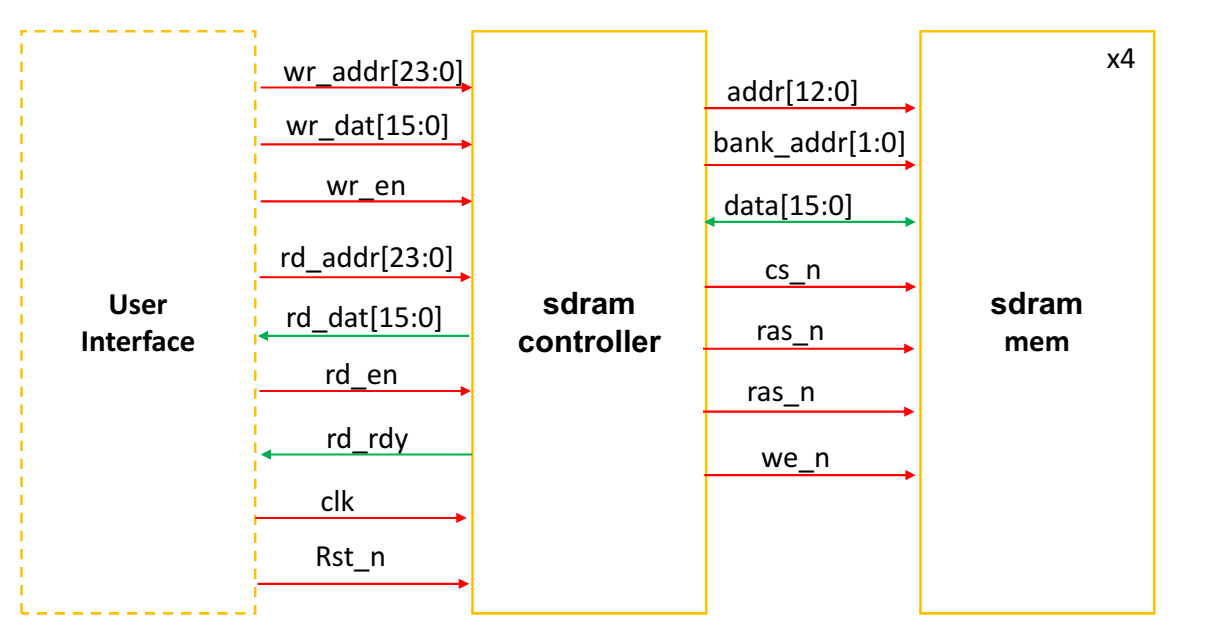
Chương sau là phần trình bài thiết kế SDRAM một cách chi tiết với ngôn ngữ Verilog.

# Thiết kế SDRAM với ngôn ngữ Verilog

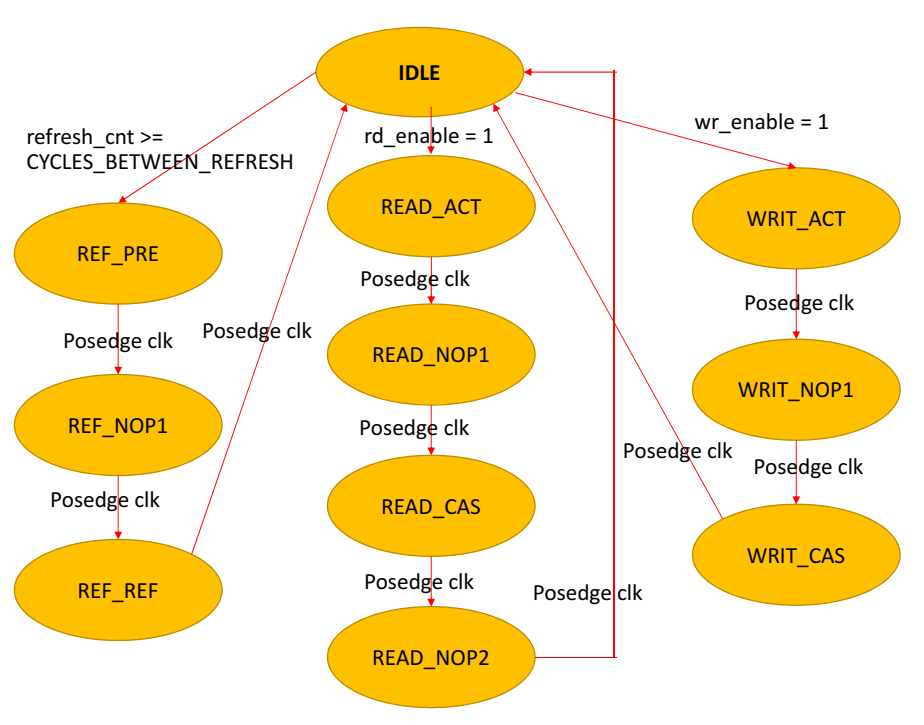
## Thế kế một SDRAM điển hình (DUT)

Bên dưới là thiết kế một SDRAM được sử dụng trong đồ án này. Một bộ điều khiển (controler) nhận thông tin và dữ liệu đầu từ phía user thông qua một sơ đồ máy trạng thái (FSM) để tạo thành các chỉ lệnh (command) các chỉ lệnh này sẽ giúp đóng mở các bank, đọc ghi dữ liệu trong khối bộ nhớ SDRAM (SDRAM memory). Trong thiết kế này chọn lựa số bank trong SDRAM là 4, dữ liệu đầu vào là 16 bit, độ rộng bus địa chỉ là 24 bit.

Đây không phải là cách thiết kế duy nhất trên thực tế có nhiều cách thiết kế và sự lựa chọn dữ liệu cũng như thông tin đầu vào rất khác nhau. Nhưng tất cả thiết kế đều hướng đến mục tiêu là truy xuất (đọc , ghi) SDRAM một cách chính xác.



*Hình 5‑1 Thiết kế tổng quan phần cứng SDRAM*

Bên dưới là sơ đồ máy trạng thái trong khối SDRAM Controler.

##### Hình 5‑2 Sơ đồ máy trạng thái cua bộ SDRAM controler

## Thiết kế môi trường mô phỏng test bench

Môi trường mô phỏng (simulation environment) hay môi trường kiểm tra (verification environment) của thiết kế là một tổ hợp các thành phần (bao gồm cả thiết kế) cho phép công cụ mô phỏng có thể sử dụng để thực thi tính toán các dữ liệu đầu ra của thiết kế dựa trên các dữ liệu đầu vào được môi trường cung cấp. Dữ liệu đầu ra có thể hiện dưới nhiều dạng khác nhau như một file dữ liệu, hoặc hiển thị trên ngõ ra chuẩn (màn hình).

Hiện nay, với sự phức tạp ngày càng tăng của các thiết kế môi trường mô phỏng cũng được xây dựng dựa trên các giao thức và phương pháp được chuẩn hóa ví dụ như OVM, VMM, UVM,... Các giao thức này phân chia môi trường thành nhiều thành phần với tên gọi khác nhau nhưng vẫn không nằm ngoài 2 mục đích là tạo dữ liệu đầu vào và giám sát dữ liệu đầu ra của thiết kế.

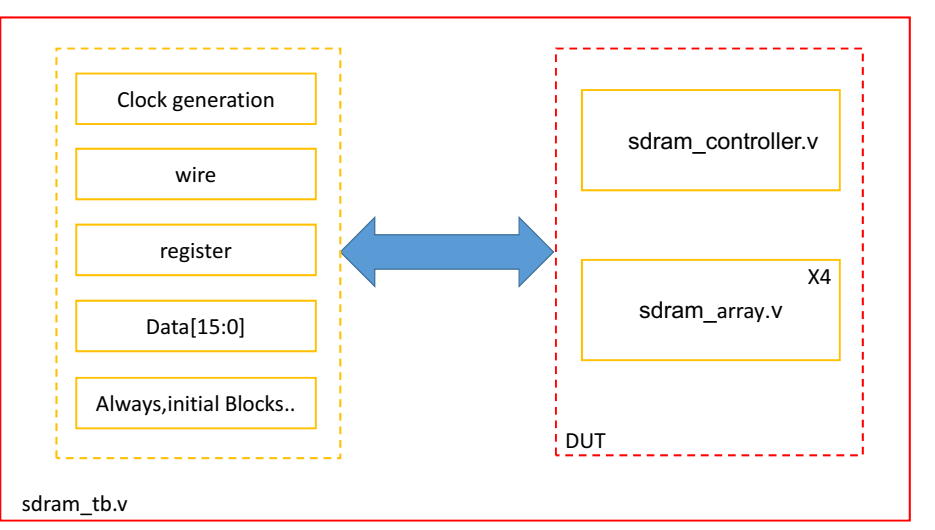
Trong phạm vi project này file testtop miêu tả môi trường mô phỏng là **sdram\_tb.v.**

Các thành phần trong môi trường test bench này gồm có:

* Clock generation: xung clock sử dụng trong toàn bộ thiết kế.
* Các khối giả lập tạo tín hiệu input đầu vào cho DUT (khối thiết kế SDRAM) được đặt trong các block alway , initial...
* Các biến (register) và các dây (wire) tạo ra để phục vụ cho viêc giả lập dữ liệu đầu vào.
* DUT (Device under test): Đây chính là thiết kế SDRAM cần được verify kiểm thử (file **sdram\_top.v**).

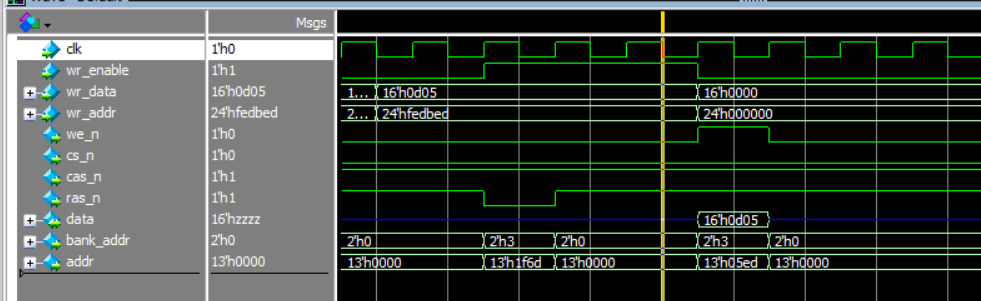
Trong phạm vi đồ án này lựa chọn data input dữ liệu đầu vào 16 bit, có nhiều cách để tạo dữ liệu này. Có thể tạo trực tiếp trong mội trường tesh bench hoặc gọi các giá trị 16 bit này từ một file log , text … hoặc data có thể được tạo khi force (gán) trực tiếp trong model sim có nhiều cách để thực hiện việc tạo data này. Trong đồ án này lựa chọn phương án tạo dữ liệu data input trực tiếp trong file test top (file **sdram\_tb.v**).

Về tính chính xác của môi trường kiểm thử này chỉ dừng lại ở mức độ so sánh kết quả ghi vào và kết quả đọc ra trở lạ từ SDRAM bằng cách xem và phân tích dạng sóng (wave form).

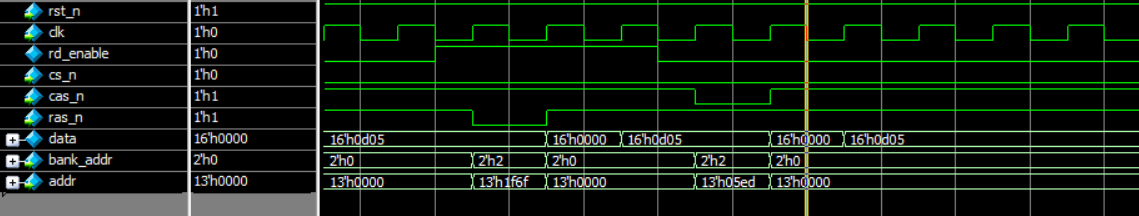


##### Hình 5‑3 Môi trường testbench để kiểm tra DUT thiết kế SDRAM

# Kết quả mô phỏng thiết kế SDRAM trên model sim.



##### Hình 6‑1 Timing ghi giá trị vào SDRAM



##### Hình 6‑2 Timing đọc giá trị từ SDRAM

Phân tích 2 biểu đồ timing của viêc ghi và đọc dữa liệu từ SDRAM ra cho cùng giá trị (như hình là 16’h0d05). Như đã nêu ở phần trước trong đồ án chỉ dừng lại ở việc sử dụng cách phân tích dạng sóng và timing trên model sim để kiểm tra thiết kế. Việc ghi kết qua vào SDRAM và đọc ra ghi nhận đúng với kết quả đã ghi vào cho thấy thiết kế đã đạt yêu cầu.

# Tổng Kết

Ngày RAM là một trong những thành phần phần cứng có tác động tích cực trong trong PC , các hệ thống SoC.. RAM có mặt hầu hết trong mọi thiết bị thông minh trong nhiều lĩnh vực khác nhau. Vì vậy việc tìm hiểu và bổ sung kiến thức và lĩnh vực này là vô cùng cần thiết đối với những sinh viên như chúng em.

Với việc thực hiện đề tài này đã giúp em có được những khái niệm cơ bản nhưng vô cùng cần thiết điển hình là các nguyên lý hoạt động, tổ chức bộ nhớ, cách truy cập vào một SDRAM nói riêng và RAM nói chung và còn nhiều kiến thức bổ ích khác. Những kiến thức tuy cơ bản nhưng nó là một nền tảng vững chắc để có thể tiếp tục nghiên cứu sâu hơn sau khi tốt nghiệp.

Qua đề tài này đã giúp em tìm hiểu được cách làm thế nào design và sử dụng các ngôn ngữ phần cứng để hiện thực các thiết kế (trong project này là ngôn ngữ verilog) cũng như cách sử dụng các phần mêm mô phỏng như model sim để hiện thực hóa các mô phỏng trước khi đưa vào các hệ thống chạy trên thực tế. Cũng qua đề tài đã giúp em có thêm các kĩ năng về phân tích timming, code tuy đây chưa phải là một dự án lớn và chưa quá phức tạp nhưng là tiền đề quan trọng sau này để có thể thực hiện các thiết kế phức tạp hơn ngoài thực tế.

Thực tế trong phạm vi của đồ án này chỉ dừng lại ở mức thiết kế và giả lập một SDRAM đơn giản với độ rộng đầu vào các bus, số bank , hàng , cột , số mảng bộ nhớ là tùy chọn. Cũng như việc thoải mái trong cách thiết kế bộ điều khiển SDRAM. Trên thực tế việc thiết kế và tạo ra các SDRAM để ứng dụng vào thực tế đòi hỏi một sự gắt gao và phức tạp hơn rất nhiều.

# Phụ Lục.

## Code module sdram\_controller.v :

module sdram\_controller (

/\* HOST INTERFACE \*/

wr\_addr,

wr\_data,

wr\_enable,

rd\_addr,

rd\_data,

rd\_ready,

rd\_enable,

busy,

rst\_n,

clk,

/\* SDRAM SIDE \*/

addr,

bank\_addr,

data,

cs\_n,

ras\_n,

cas\_n,

we\_n

);

/\* Internal Parameters \*/

parameter ROW\_WIDTH = 13;

parameter COL\_WIDTH = 9;

parameter BANK\_WIDTH = 2;

parameter SDRADDR\_WIDTH = ROW\_WIDTH > COL\_WIDTH ? ROW\_WIDTH : COL\_WIDTH;

parameter HADDR\_WIDTH = BANK\_WIDTH + ROW\_WIDTH + COL\_WIDTH;

parameter CLK\_FREQUENCY = 133; // Mhz

parameter REFRESH\_TIME = 32; // ms (how often we need to refresh)

parameter REFRESH\_COUNT = 8192; // cycles (how many refreshes required per refresh time)

// clk / refresh = clk / sec

// , sec / refbatch

// , ref / refbatch

localparam CYCLES\_BETWEEN\_REFRESH = ( CLK\_FREQUENCY

\* 1\_000

\* REFRESH\_TIME

) / REFRESH\_COUNT;

// STATES - State

localparam IDLE = 5'b00000;

localparam INIT\_NOP1 = 5'b01000, //8

INIT\_PRE1 = 5'b01001, //9

INIT\_NOP1\_1=5'b00101, //5

INIT\_REF1 = 5'b01010, //10

INIT\_NOP2 = 5'b01011, //11

INIT\_REF2 = 5'b01100, //12

INIT\_NOP3 = 5'b01101, //13

INIT\_LOAD = 5'b01110, //14

INIT\_NOP4 = 5'b01111; //15

localparam REF\_PRE = 5'b00001, //1

REF\_NOP1 = 5'b00010, //2

REF\_REF = 5'b00011, //3

REF\_NOP2 = 5'b00100; //4

localparam READ\_ACT = 5'b10000, //16

READ\_NOP1 = 5'b10001, //17

READ\_CAS = 5'b10010, //18

READ\_NOP2 = 5'b10011, //19

READ\_READ = 5'b10100; //20

localparam WRIT\_ACT = 5'b11000, //24

WRIT\_NOP1 = 5'b11001, //25

WRIT\_CAS = 5'b11010, //26

WRIT\_NOP2 = 5'b11011; //27

// Commands CCRCWBBA

// ESSSE100

localparam CMD\_PALL = 8'b1001\_0001,

CMD\_REF = 8'b1000\_0000,

CMD\_NOP = 8'b1011\_0000,

CMD\_MRS = 8'b1000\_000x,

CMD\_BACT = 8'b1001\_0xxx,

CMD\_READ = 8'b1010\_0xx1,

CMD\_WRIT = 8'b1011\_1xx1;

/\* Interface Definition \*/

/\* HOST INTERFACE \*/

input [HADDR\_WIDTH-1:0] wr\_addr;

input [15:0] wr\_data;

input wr\_enable;

input [HADDR\_WIDTH-1:0] rd\_addr;

output [15:0] rd\_data;

input rd\_enable;

output rd\_ready;

output busy;

input rst\_n;

input clk;

/\* SDRAM SIDE \*/

output [SDRADDR\_WIDTH-1:0] addr;

output [BANK\_WIDTH-1:0] bank\_addr;

inout [15:0] data;

output cs\_n;

output ras\_n;

output cas\_n;

output we\_n;

/\* I/O Registers \*/

reg [HADDR\_WIDTH-1:0] haddr\_r;

reg [15:0] wr\_data\_r;

reg [15:0] rd\_data\_r;

reg busy;

reg [SDRADDR\_WIDTH-1:0] addr\_r;

reg [BANK\_WIDTH-1:0] bank\_addr\_r;

reg rd\_ready\_r;

wire [15:0] data\_output;

assign rd\_data = rd\_data\_r;

/\* Internal Wiring \*/

reg [3:0] state\_cnt;

reg [9:0] refresh\_cnt;

reg [7:0] command;

reg [4:0] state;

// TODO output addr[6:4] when programming mode register

reg [7:0] command\_nxt;

reg [3:0] state\_cnt\_nxt;

reg [4:0] next;

wire clock\_enable;

assign {clock\_enable, cs\_n, ras\_n, cas\_n, we\_n} = command[7:3];

// state[4] will be set if mode is read/write

assign bank\_addr = (state[4]) ? bank\_addr\_r : command[2:1];

assign addr = (state[4] | state == INIT\_LOAD) ? addr\_r : { {SDRADDR\_WIDTH-11{1'b0}}, command[0], 10'd0 };

assign data = (state == WRIT\_CAS) ? wr\_data\_r : 16'bz;

assign rd\_ready = rd\_ready\_r;

// HOST INTERFACE

// all registered on posedge

always @ (posedge clk)

if (~rst\_n)

begin

state <= INIT\_NOP1;

command <= CMD\_NOP;

state\_cnt <= 4'hf;

haddr\_r <= {HADDR\_WIDTH{1'b0}};

wr\_data\_r <= 16'b0;

rd\_data\_r <= 16'b0;

busy <= 1'b0;

end

else

begin

state <= next;

command <= command\_nxt;

if (!state\_cnt)

state\_cnt <= state\_cnt\_nxt;

else

state\_cnt <= state\_cnt - 1'b1;

if (wr\_enable)

wr\_data\_r <= wr\_data;

if (state == READ\_READ)

begin

rd\_data\_r <= data;

rd\_ready\_r <= 1'b1;

end

else

rd\_ready\_r <= 1'b0;

busy <= state[4];

if (rd\_enable)

haddr\_r <= rd\_addr;

else if (wr\_enable)

haddr\_r <= wr\_addr;

end

// Handle refresh counter

always @ (posedge clk)

if (~rst\_n)

refresh\_cnt <= 10'b0;

else

if (state == REF\_NOP2)

refresh\_cnt <= 10'b0;

else

refresh\_cnt <= refresh\_cnt + 1'b1;

/\* Handle logic for sending addresses to SDRAM based on current state\*/

always @\*

begin

bank\_addr\_r = 2'b00;

addr\_r = {SDRADDR\_WIDTH{1'b0}};

if (state == READ\_ACT | state == WRIT\_ACT)

begin

bank\_addr\_r = haddr\_r[HADDR\_WIDTH-1:HADDR\_WIDTH-(BANK\_WIDTH)];

addr\_r = haddr\_r[HADDR\_WIDTH-(BANK\_WIDTH+1):HADDR\_WIDTH-(BANK\_WIDTH+ROW\_WIDTH)];

end

else if (state == READ\_CAS | state == WRIT\_CAS)

begin

// Send Column Address

// Set bank to bank to precharge

bank\_addr\_r = haddr\_r[HADDR\_WIDTH-1:HADDR\_WIDTH-(BANK\_WIDTH)];

// Examples for math

// BANK ROW COL

// HADDR\_WIDTH 2 + 13 + 9 = 24

// SDRADDR\_WIDTH 13

// Set CAS address to:

// 0s,

// 1 (A10 is always for auto precharge),

// 0s,

// column address

addr\_r = {

{SDRADDR\_WIDTH-(11){1'b0}},

1'b1, /\* A10 \*/

{10-COL\_WIDTH{1'b0}},

haddr\_r[COL\_WIDTH-1:0]

};

end

else if (state == INIT\_LOAD)

begin

// Program mode register during load cycle

// B C SB

// R A EUR

// S S-3Q ST

// T 654L210

addr\_r = {{SDRADDR\_WIDTH-10{1'b0}}, 10'b1000110000};

end

end

// Next state logic

always @\*

begin

state\_cnt\_nxt = 4'd0;

command\_nxt = CMD\_NOP;

if (state == IDLE)

// Monitor for refresh or hold

if (refresh\_cnt >= CYCLES\_BETWEEN\_REFRESH)

begin

next = REF\_PRE;

command\_nxt = CMD\_PALL;

end

else if (rd\_enable)

begin

next = READ\_ACT;

command\_nxt = CMD\_BACT;

end

else if (wr\_enable)

begin

next = WRIT\_ACT;

command\_nxt = CMD\_BACT;

end

else

begin

// HOLD

next = IDLE;

end

else

if (!state\_cnt)

case (state)

// INIT ENGINE

INIT\_NOP1:

begin

next = INIT\_PRE1;

command\_nxt = CMD\_PALL;

end

INIT\_PRE1:

begin

next = INIT\_NOP1\_1;

end

INIT\_NOP1\_1:

begin

next = INIT\_REF1;

command\_nxt = CMD\_REF;

end

INIT\_REF1:

begin

next = INIT\_NOP2;

state\_cnt\_nxt = 4'd7;

end

INIT\_NOP2:

begin

next = INIT\_REF2;

command\_nxt = CMD\_REF;

end

INIT\_REF2:

begin

next = INIT\_NOP3;

state\_cnt\_nxt = 4'd7;

end

INIT\_NOP3:

begin

next = INIT\_LOAD;

command\_nxt = CMD\_MRS;

end

INIT\_LOAD:

begin

next = INIT\_NOP4;

state\_cnt\_nxt = 4'd1;

end

// INIT\_NOP4: default - IDLE

// REFRESH

REF\_PRE:

begin

next = REF\_NOP1;

end

REF\_NOP1:

begin

next = REF\_REF;

command\_nxt = CMD\_REF;

end

REF\_REF:

begin

next = REF\_NOP2;

state\_cnt\_nxt = 4'd7;

end

// REF\_NOP2: default - IDLE

// WRITE

WRIT\_ACT:

begin

next = WRIT\_NOP1;

state\_cnt\_nxt = 4'd1;

end

WRIT\_NOP1:

begin

next = WRIT\_CAS;

command\_nxt = CMD\_WRIT;

end

WRIT\_CAS:

begin

next = WRIT\_NOP2;

state\_cnt\_nxt = 4'd1;

end

// WRIT\_NOP2: default - IDLE

// READ

READ\_ACT:

begin

next = READ\_NOP1;

state\_cnt\_nxt = 4'd1;

end

READ\_NOP1:

begin

next = READ\_CAS;

command\_nxt = CMD\_READ;

end

READ\_CAS:

begin

next = READ\_NOP2;

state\_cnt\_nxt = 4'd1;

end

READ\_NOP2:

begin

next = READ\_READ;

end

// READ\_READ: default - IDLE

default:

begin

next = IDLE;

end

endcase

else

begin

// Counter Not Reached - HOLD

next = state;

command\_nxt = command;

end

end

endmodule

## Code module sdram\_array.v

module sdram\_array(

clk,

rst\_n,

addr,

bank\_addr,

idata,

odata,

cs\_n,

ras\_n,

cas\_n,

we\_n

);

parameter ROW\_WIDTH = 13;

parameter COL\_WIDTH = 9;

parameter BANK\_WIDTH = 2;

parameter SDRADDR\_WIDTH = ROW\_WIDTH > COL\_WIDTH ? ROW\_WIDTH : COL\_WIDTH;

parameter HADDR\_WIDTH = BANK\_WIDTH + ROW\_WIDTH + COL\_WIDTH;

input clk;

input rst\_n;

input [SDRADDR\_WIDTH-1:0] addr;

input [BANK\_WIDTH-1:0] bank\_addr;

input [15:0] idata;

output [15:0] odata;

input cs\_n;

input ras\_n;

input cas\_n;

input we\_n;

wire rd\_cond;

assign rd\_cond = ras\_n & cas\_n & (!cs\_n) & (!we\_n);

wire wr\_cond;

assign wr\_cond = ras\_n & cas\_n & (!cs\_n) & (we\_n);

reg [16-1:0] mem\_ram;

always @ (posedge clk)

begin

if (!rst\_n)

begin

mem\_ram <= 16'd0;

end

else

begin

mem\_ram <= wr\_cond ? idata : mem\_ram;

end

end

reg [15:0] odata = 16'd0;

always @ (posedge clk) odata = rd\_cond ? mem\_ram : 16'd0;

endmodule

## Code module sdram\_tb.v

module sdram\_tb();

//vlog\_tb\_utils vlog\_tb\_utils0();

/\* HOST CONTROLLS \*/

reg [23:0] haddr;

reg [15:0] data\_input;

wire [15:0] data\_output;

wire busy;

reg rd\_enable, wr\_enable, rst\_n, clk;

/\* SDRAM SIDE \*/

wire [12:0] addr;

wire [1:0] bank\_addr;

wire [15:0] idata,data;

wire cs\_n, ras\_n, cas\_n, we\_n;

reg [15:0] data\_r;

assign idata = data\_r;

initial

begin

haddr = 24'd0;

data\_input = 16'd0;

rd\_enable = 1'b0;

wr\_enable = 1'b0;

rst\_n = 1'b1;

clk = 1'b0;

data\_r = 16'hzzzz;

end

always

#1 clk <= ~clk;

initial

begin

#3 rst\_n = 1'b0;

#3 rst\_n = 1'b1;

#120 haddr = 24'hfedbed;

data\_input = 16'd3333;

#3 wr\_enable = 1'b1;

#6 wr\_enable = 1'b0;

haddr = 24'd0;

data\_input = 16'd0;

#120 haddr = 24'hbedfed;

#3 rd\_enable = 1'b1;

#6 rd\_enable = 1'b0;

haddr = 24'd0;

#8 data\_r = 16'hbbbb;

#2 data\_r = 16'hzzzz;

#1000 $finish;

end

///////////////////////////////////////////////////////////////////////////////////////

//DUT instantial : Contrer + array SDRAM

sdram\_controller sdram\_controller (

/\* HOST INTERFACE \*/

.wr\_addr (haddr),

.wr\_data (data\_input),

.wr\_enable (wr\_enable),

.rd\_addr (haddr),

.rd\_data (data\_output),

.rd\_ready (),

.rd\_enable (rd\_enable),

.busy (busy),

.rst\_n (rst\_n),

.clk (clk),

/\* SDRAM SIDE \*/

.addr (addr),

.bank\_addr (bank\_addr),

.data (idata),

.cs\_n (cs\_n),

.ras\_n (ras\_n),

.cas\_n (cas\_n),

.we\_n (we\_n)

);

sdram\_array sdram\_array

(

.clk (clk),

.rst\_n (rst\_n),

.addr (addr),

.bank\_addr (bank\_addr),

.idata (idata),

.odata (data),

.cs\_n (cs\_n),

.ras\_n (ras\_n),

.cas\_n (cas\_n),

.we\_n (we\_n)

);

endmodule

## Tài liệu tham khảo

* Book : Memory Systems Cache DRAM Disk 2008 (David Wang · Bruce Jacob)
* Internet : Design and Implementation of DDR SDRAM Controller using Verilog

<https://www.ijsr.net/archive/v2i1/IJSROFF130201036.pdf>

* Internet:

<https://vi.abadgar-q.com/wiki/Synchronous_dynamic_random-access_memory>