HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
ThS. TRẦN THỊ THÚY HÀ - ThS. ĐỖ MẠNH HÀ

Giáo trình
Điện tử số

(TẬP 2)

NHÀ XUẤT BẢN THÔNG TIN VÀ TRUYỀN THÔNG
Hà Nội, 11-2009

LỜI NÓI ĐẦU

Cùng với sự tiến bộ của khoa học và công nghệ, các thiết bị điện tử đã, đang và sẽ tiếp tục được ứng dụng ngày càng rộng rãi và mang lại hiệu quả cao trong hầu hết các lĩnh vực kinh tế kỹ thuật cũng như đời sống xã hội.

Việc xử lý tín hiệu trong các thiết bị điện tử hiện đại đều dựa trên cơ sở nguyên lý số. Bởi vậy việc nắm vững kiến thức về điện tử số là yêu cầu bắt buộc đối với kỹ sư điện, điện tử, viễn thông và CNTT hiện nay. Kiến thức về Điện tử số không phải chỉ cần thiết đối với kỹ sư các ngành kể trên mà còn cần thiết đối với nhiều cán bộ kỹ thuật các chuyên ngành khác có ứng dụng điện tử.

Nhằm giới thiệu một cách hệ thống các khái niệm cơ bản về điện tử số, các cổng logic, các phần tử cơ bản, các mạch số chức năng điển hình, các phương pháp phân tích và thiết kế mạch logic ... Học viện Công nghệ Bưu chính Viễn thông phối hợp với Nhà xuất bản Thông tin và Truyền thông xuất bản cuốn sách "***Giáo trình Điện tử số***" (02 tập).

Giáo trình bao gồm các kiến thức cơ bản về cơ sở đại số logic, mạch cổng logic, mạch logic tổ hợp, các trigơ, mạch logic tuần tự, các mạch phát xung và tạo dạng xung, các bộ nhớ thông dụng. Giáo trình còn bao gồm các kiến thức cơ bản về cấu kiện logic khả trình và ngôn ngữ mô tả phần cứng VHDL. Đây là ngôn ngữ phổ biến hiện nay dùng để mô tả cho mô phỏng cũng như thiết kế các hệ thống số. Nội dung giáo trình gồm 02 tập có 9 chương: 7 chương đầu do ThS. Trần Thị Thúy Hà biên soạn, 2 chương cuối do ThS. Đỗ Mạnh Hà biên soạn. Trước và sau mỗi chương đều có phần giới thiệu và phần tóm tắt để giúp người học dễ nắm bắt kiến thức. Các câu hỏi ôn tập để người học kiểm tra mức độ nắm kiến thức sau khi học mỗi chương.

Giáo trình gồm 02 tập có 9 chương được bố cục như sau:

Tập 1 gồm:

Chương 1: Hệ đếm.

Chương 2: Đại số Boole.

Chương 3: Cổng logic TTL và CMOS.

Chương 4: Mạch logic tổ hợp.

Chương 5: Mạch logic tuần tự.

Tập 2 gồm:

Chương 6: Mạch phát xung và tạo dạng xung.

Chương 7: Bộ nhớ bán dẫn.

Chương 8: Cấu kiện logic khả trình (PLD).

Chương 9 : Ngôn ngữ mô tả phần cứng VHDL.

Trên cơ sở các kiến thức căn bản, giáo trình đã cố gắng tiếp cận các vấn đề hiện đại, đồng thời liên hệ với thực tế kỹ thuật. Tuy nhiên do thời gian biên soạn có hạn nên cuốn giáo trình có thể còn những thiếu sót, rất mong được bạn đọc góp ý. Các ý kiến xin gửi về Bộ môn Kỹ thuật Điện tử - Khoa Kỹ thuật Điện tử 1 - Học viện Công nghệ Bưu chính Viễn thông.

Xin trân trọng giới thiệu!

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

THUẬT NGỮ VIẾT TẮT

ALU	Arthmetic Logic Unit	Đơn vị tính logic và số học
ANSI	American National Standards Institude	Viện tiêu chuẩn Quốc gia Hoa Kỳ
ASIC	Application Specific Integrated Circuit	Mạch tích hợp ứng dụng đặc biệt
BCD	Binary Coded Decimal	Số thập phân mã hóa theo nhị phân
Bit	Binary Digit	Số nhị phân
Byte		Một nhóm gồm 8 bit
C, CLK	Clock	Xung đồng hồ (Xung nhịp)
Cache		Bộ nhớ trung gian
CAS	Column Address Select	Chọn địa chỉ cột
CLR	Clear	Xóa
CMOS	Complementary Metal Oxide Semiconductor	Vật liệu bán dẫn gồm hai linh kiện NMOS và PMOS mắc tổ hợp với nhau
CPU	Central Processing Unit	Đơn vị xử lý trung tâm
CPLD	Complex Programmable Logic Device	Cấu kiện logic khả trình phức tạp
Crumb	2 bit	
CS	Chip Select	Chọn chíp
DDL	Diode-Diode Logic	Cổng logic chứa các điốt
Deckle		10 bit
DLL	Delay Locked Loop	Vòng khoá pha trễ
DEMUX	DeMultiplexer	Bộ phân kênh
DRAM	Dynamic RAM	RAM động

DTL	Diode Transistor Logic	Cổng logic chứa các điốt và tranzito
Dynner		32 bit
ECL	Emitter Couple Logic	Cổng logic ghép cực Emitơ
EEPROM	Electrically Erasable ROM	ROM lập trình được và xóa được bằng điện
EPROM	Erasable ROM	ROM lập trình được và xóa được bằng tia cực tím
FET	Field Effect Transistor	Tranzito hiệu ứng trường
FPGA	Field Programmable Gate Array	Ma trận cổng lập trình được theo trường.
H	High	Mức logic cao
I^2L	Integrated Injection Logic	Mạch logic tích hợp phun
IC	Integrated Circuit	Mạch tích hợp
IEEE	Institude of Electrical and Electronics Engineers	Viện kỹ thuật Điện và điện tử
ISP	In System Programming	Lập trình trên hệ thống
L	Low	Mức logic thấp
Latch		Bộ chốt
LCD	Liquid Crystal Display	Hiển thị tinh thể lỏng
LED	Light Emitting Diode	Điốt phát quang
LSB	Least Significant Bit	Bit có ý nghĩa bé nhất
LUT	Look Up Table	Bảng ánh xạ
Maxterm		Thừa số lớn nhất
Minterm		Số hạng nhỏ nhất
MOSFET	Metal Oxide Semiconductor FET	FET có cực cửa cách ly bằng lớp ôxít kim loại
MROM	Mask ROM	ROM được chế tạo bằng phương pháp che mặt nạ
MSB	Most Significant Bit	Bit có ý nghĩa lớn nhất

MSI	Medium Scale Integrated	Mức độ tích hợp trung bình
MUX	Multiplexer	Bộ ghép kênh
Nibble		4 bit
NMOS	N – chanel MOS	Tranzito trường kênh dẫn N
PAL	Programmable Array Logic	Logic mảng khả trình
PLA	Programmable Logic Array	Mảng logic khả trình
PLD	Programmable Logic Device	Cấu kiện logic khả trình
Playte	16 bit	
PLS	Programmable Logic Sequence	Logic tuần tự khả trình
PMOS	P – chanel MOS	Tranzito trường kênh dẫn P
PRE	Preset	Tái lập
PROM	Programmable ROM	ROM khả trình
RAM	Random Access Memory	Bộ nhớ truy cập ngẫu nhiên
RAS	Row Address Select	Chọn địa chỉ hàng
RBI	Riple Blanking Input	Đầu vào xóa nối tiếp
RBO	Riple Blanking Output	Đầu ra xóa nối tiếp
ROM	Read Only Memory	Bộ nhớ chỉ đọc
RTL	Resistance Transistor Logic	Cổng logic dùng điện trở và tranzito
RTL*	Register Transfer Level (*Chương 9)	Mức luồng dữ liệu
SPLD	Simple Programmable Logic Device	Cấu kiện logic khả trình đơn giản
SRAM	Static RAM	RAM tĩnh
SSI	Small Scale Integrated	Mức độ tích hợp trung bình
TTL	Transistor – Transistor Logic	Cổng logic dùng Tranzito
VLSI	Very Large Scale Integrated	Mức độ tích hợp rất lớn

MỤC LỤC

Chương 6

MẠCH PHÁT XUNG VÀ TẠO DẠNG XUNG

GIỚI THIỆU

Hầu hết các hệ thống kỹ thuật số đều yêu cầu một vài loại dạng sóng định thời, ví dụ một nguồn xung của bộ dao động cần thiết cho tất cả các hệ thống tuần tự định thời. Trong các hệ thống kỹ thuật số, dạng sóng xung vuông thường được sử dụng nhất. Sự tạo ra các dạng sóng xung vuông được gọi là bộ đa hài.

Có ba loại bộ đa hài:

- Bộ dao động đa hài (chạy tự do).
- Bộ đa hài đơn ổn (một nhịp).
- Bộ đa hài hai trạng thái ổn định (trigơ).

Một bộ dao động đa hài chỉ là một bộ dao động để tạo ra dạng xung. Nó có hai trạng thái chuẩn mà không yêu cầu sự kích hoạt từ bên ngoài. Bộ này thường được dùng làm xung điều khiển cho các mạch tuần tự.

Một bộ đa hài đơn ổn chỉ có một trạng thái ổn định, tức là trong điều kiện trạng thái ổn định thì đầu ra của nó cố định. Đầu ra này ở trạng thái thấp hoặc ở trạng thái cao. Mạch này cần một xung kích khởi từ bên ngoài để cho mạch chuyển sang trạng thái khác. Mạch này vẫn giữ nguyên trạng thái cũ trong một khoảng thời gian, khoảng thời gian này phụ thuộc vào các thành phần được dùng trong mạch. Trạng thái của mạch này được xem là trạng thái ổn định bởi vì nó phục hồi trở về trạng thái ổn định mà không cần bất kỳ xung kích hoạt nào từ

bên ngoài. Độ rộng của xung kích hoạt rất nhỏ, độ rộng của xung đầu ra chỉ phụ thuộc vào khoảng thời gian mà mạch giữ lại ở trạng thái ổn định. Mạch này được gọi là mạch một nhịp (one-shot) bởi vì một xung kích hoạt chỉ tạo được một xung có độ rộng khác. Mạch này rất hữu dụng bởi vì nó có thể tạo ra một xung tương đối dài (hàng chục mili giây - ms) từ một xung hẹp, do đó nó còn được gọi là bộ giảm xung (pulse stretcher).

Ví dụ, một bộ vi xử lý có thể phát tín hiệu cho một thiết bị bên ngoài để in một nội dung nào đó bằng cách truyền qua một xung. Thiết bị đầu ra nói chung có tốc độ chậm hơn bộ vi xử lý, do đó nó yêu cầu một xung tín hiệu trong một khoảng thời gian lâu hơn. Điều này đạt được bằng một mạch giao tiếp có chứa bộ đa hài đơn ổn.

Một mạch đa hài trong đó cả hai trạng thái đều ổn định thì được gọi là mạch đa hài hai trạng thái ổn định hay trigơ. Mạch này thực hiện việc chuyển tiếp từ một trạng thái ổn định này sang một trạng thái ổn định khác chỉ lúc xung kích khởi được áp vào. Mạch này thường được dùng làm các thành phần của bộ nhớ trong các hệ thống kỹ thuật số và đã được thảo luận ở chương 5.

Chương này tập trung vào sơ đồ, nguyên tắc hoạt động, ứng dụng của các mạch dao động đa hài, mạch dao động đa hài đợi, trigơ Schmitt dựa trên các cổng TTL, CMOS và IC định thời 555. Sau chương này bạn đọc có thể tự thiết kế các mạch dao động theo các yêu cầu cơ bản cho các ứng dụng khác nhau.

6.1 MẠCH PHÁT XUNG

Trong mạch dãy đồng bộ, xung vuông của tín hiệu đồng hồ (đồng hồ) được dùng để điều khiển, phối hợp hoạt động của toàn bộ hệ thống. Vậy đặc tính của xung đồng hồ trực tiếp ảnh hưởng đến chất lượng hoạt động của hệ thống. Dựa vào hình 6.1 chúng ta tìm hiểu một số chỉ tiêu đánh giá dạng xung vuông.

- T: Chu kỳ xung. Đó là khoảng thời gian giữa 2 xung liền nhau trong loạt xung trùng lặp.

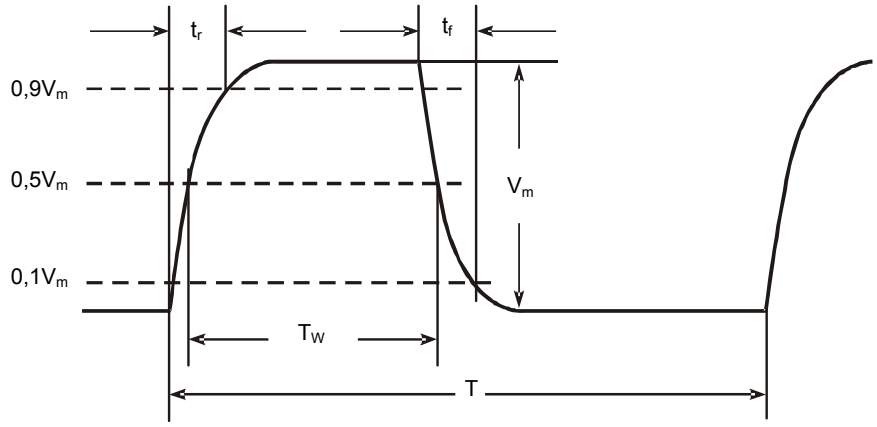
- $f = \frac{1}{T}$: Tần số xung. Đó là số xung trong 1 đơn vị thời gian.

- T_W : Độ rộng xung. Đó là khoảng thời gian từ sườn trước đến sườn sau của xung, tính từ mức $0.5V_m$.

- V_m: Biên độ xung. Đó là biên độ lớn nhất của điện áp xung.

- t_r: Sườn trước. Đó là khoảng thời gian sườn trước tăng từ $0.1V_m$ đến $0.9V_m$.

- t_f: Sườn sau. Đó là khoảng thời gian sườn sau giảm từ $0,9V_m$ đến $0,1V_m$.

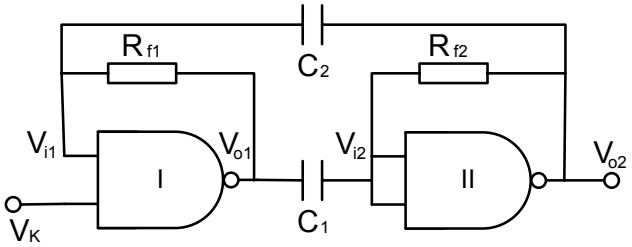


Hình 6.1: Đặc trưng kỹ thuật của xung vuông

Do vai trò quan trọng của xung đồng hồ nên các yêu cầu với chúng cũng rất chặt chẽ. Người ta đánh giá xung đồng hồ qua các chỉ tiêu: Độ ổn định tần số (hay độ không ổn định tương đối của tần số $\frac{\Delta f}{f}$), độ ổn định pha, độ ổn định biên độ.

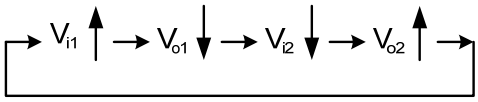
6.1.1 Mạch dao động đa hài cơ bản cổng NAND TTL

Cổng NAND khi làm việc trong vùng chuyển tiếp có thể khuếch đại mạnh tín hiệu đầu vào. Nếu 2 cổng NAND được ghép điện dung thành mạch vòng như hình 6.2 ta được bộ dao động đa hài. V_K là đầu vào điều khiển, khi ở mức cao mạch phát xung và khi ở mức thấp mạch ngừng phát.

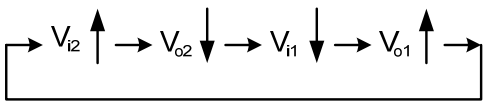


Hình 6.2: Bộ dao động đa hài cấu trúc bằng cổng NAND

Nếu các cổng I và II thiết lập điểm công tác tĩnh trong vùng chuyển tiếp và V_K = 1, thì mạch sẽ phát xung khi được nối nguồn. Nguyên tắc làm việc của mạch như sau: Giả sử do tác động của nhiễu làm cho V_{i1} tăng một chút, lập tức xuất hiện quá trình phản hồi dương sau:



Khi đó, cổng I nhanh chóng trở thành thông bão hoà, cổng II nhanh chóng ngắt, mạch bước vào trạng thái tạm ổn định. Lúc này, C_1 nạp điện và C_2 phóng điện theo mạch đơn giản hoá được thể hiện trong hình 6.2. C_1 nạp đến khi V_{i2} tăng đến ngưỡng thông V_T, trong mạch xuất hiện quá trình phản hồi dương như sau:



Kết quả quá trình này là: Cổng I nhanh chóng ngắt còn cổng II thông bão hoà, mạch điện bước vào trạng thái tạm ổn định mới. Lúc này C_2 nạp điện còn C_1 phóng cho đến khi V_{i1} bằng ngưỡng thông V_T làm xuất hiện quá trình phản hồi dương đưa mạch về trạng thái ổn định ban đầu. Mạch không ngừng dao động, khi bỏ qua điện trở đầu ra của các cổng NAND, dựa vào hình 6.3a giản đồ xung của mạch được thể hiện trên hình 6.3b.

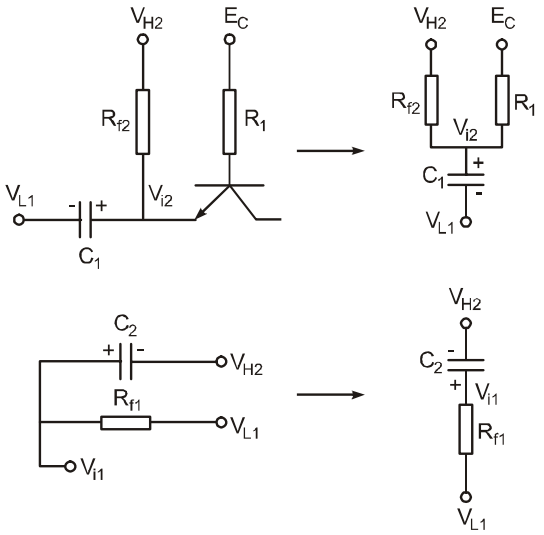
Vì thời gian nạp điện nhanh hơn thời gian phóng, nên thời gian duy trì trạng thái ổn định tạm thời phụ thuộc vào thời gian nạp điện của hai tụ điện C_1 và C_2. Từ hình 6.2 ta có thời gian nạp điện của tụ C_1 là $\tau_1 = (R_{f2} // R_1) C_1$, thời gian để V_{i2} nạp điện đến V_T là:

$$t_{M2} = \left(R_{f2} // R_1\right) C_1 \ln \frac{2V_{OH} - \left(V_T + V_{OL}\right)}{V_{OH} - V_T} \qquad (6.1)$$

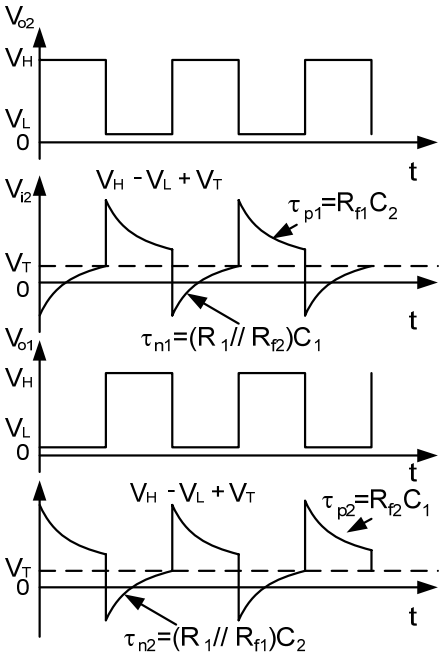
Nếu $R_{f1} = R_{f2} = R_f$, $C_1 = C_2 = C$, $V_{OH} = 3V$, $V_{OL} = 0{,}35V$, $V_T = 1{,}4V$ thì ta có:

$$T \approx 2\left(R_f // R_1\right) C \qquad (6.2)$$

T là chu kỳ của tín hiệu đa hài đầu ra.



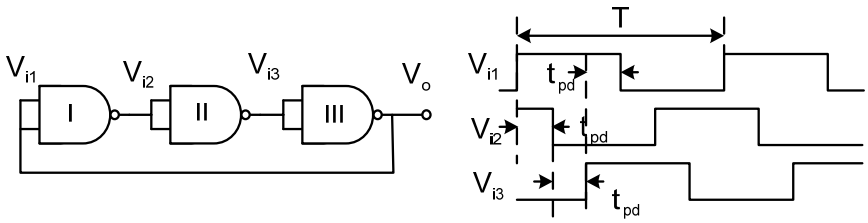
Hình 6.3a: Mạch vòng nạp phóng điện của tụ C_1, C_2



Hình 6.3b: Dạng sóng gần đúng của điện áp tại các điểm

trên mạch bộ dao động đa hài

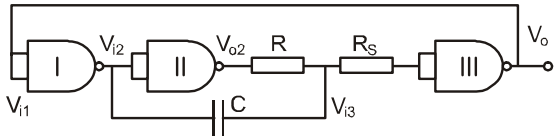
6.1.2 Mạch dao động đa hài vòng RC



Hình 6.4: Bộ dao động vòng và dạng sóng

Bộ dao động vòng có cấu trúc gồm 3 cổng NAND mắc nối tiếp như hình 6.4.

Do có sự phản hồi dương từ V_o đến V_{i1} làm cho mạch này không có trạng thái ổn định. Tần số của tín hiệu đầu ra phụ thuộc vào thời gian trễ của cổng NAND và không thể điều chỉnh được tần số này. Tần số của mạch phát sẽ điều chỉnh được khi một mạch trễ RC được mắc thêm vào mạch như hình 6.5. Tần số dao động của mạch điều chỉnh được thông qua giá trị của tụ điện C và điện trở R.

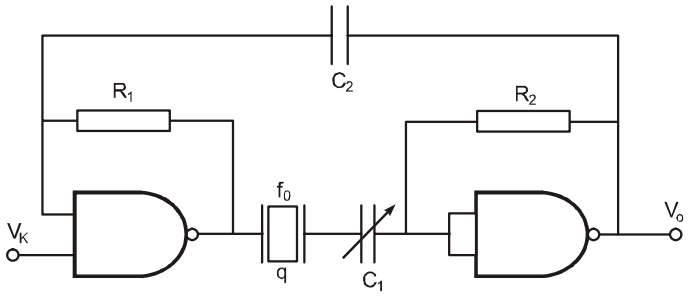


Hình 6.5: Bộ dao động đa hài có mạch RC

6.1.3 Mạch dao động đa hài thạch anh

Để có các tín hiệu đồng hồ (xung đồng hồ) có tần số chính xác và có độ ổn định cao, các mạch đa hài trình bày trên đây không đáp ứng được. Tinh thể thạch anh thường được sử dụng trong các trường hợp này. Thạch anh có tính ổn định tần số tốt, hệ số phẩm chất rất cao dẫn đến tính chọn lọc tần số rất cao. Hình 6.6 là một mạch dao động

đa hài điển hình sử dụng tinh thể thạch anh. Tần số của mạch dao động chỉ phụ thuộc vào tinh thể thạch anh mà không phụ thuộc vào giá trị các tụ điện và điện trở trong mạch.



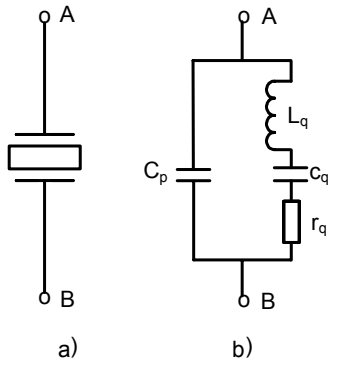
Hình 6.6: Mạch dao động đa hài thạch anh

Bộ tạo dao động đa hài thạch anh được thiết lập bằng cách thay một tụ điện trong bộ đa hài đã xét trên bằng thạch anh.

Thạch anh có tính chất áp điện, tương đương như một khung cộng hưởng.

Kí hiệu, sơ đồ tương đương về điện của thạch anh được trình bày trên hình 6.7.

Thạch anh có 2 tần số cộng hưởng: Tần số cộng hưởng nối tiếp f_q tương ứng với trở kháng của thạch anh $z_q = 0$ và tần số cộng hưởng song song f_p tương ứng với trở kháng của thạch anh $z_q = \infty$.



Hình 6.7a: Kí hiệu của thạch anh;
b) Sơ đồ tương đương và điện áp của thạch anh

Trong hình 6.6, tụ C_1 là tụ có thể biến đổi để thay đổi tần số cộng hưởng của thạch anh trong một phạm vi hẹp.

Tại tần số cộng hưởng nối tiếp của thạch anh f_q, trở kháng của thạch anh rất nhỏ, tín hiệu có tần số f_q đi qua thạch anh dễ dàng. Nên tần số dao động của mạch được quyết định bằng tần số cộng hưởng nối tiếp của thạch anh f_q mà không phụ thuộc vào các giá trị của RC của mạch.

6.1.4 Mạch dao động đa hài CMOS

Hình 6.8a là mạch dao động đa hài cơ bản sử dụng hai cổng NOR CMOS và các linh kiện định thời trở và tụ.

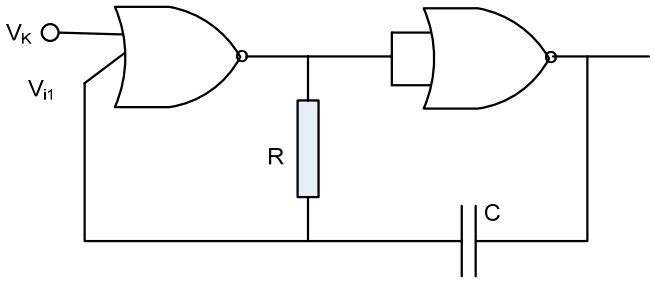
Giản đồ xung của mạch được thể hiện trên hình 6.8b.

Chu kỳ dao động của mạch được tính gần đúng như sau:

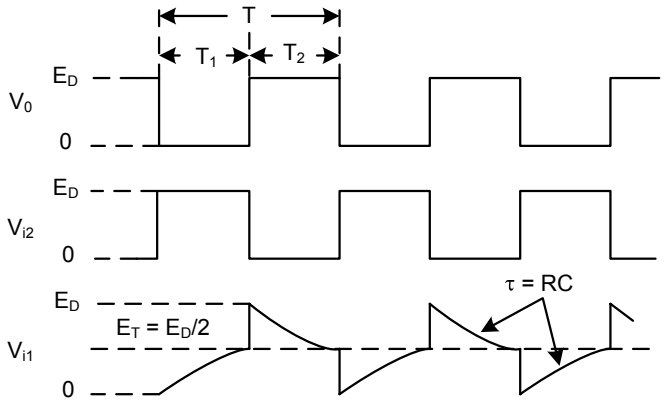
$$T = T_1 + T_2 = RC\ln\left(\frac{E_D}{E_D - V_T} + \frac{E_D}{V_T}\right) \tag{6.3}$$

Nếu giả thiết $V_T = E_D/2$ thì $T_1 = T_2$, khi đó

$$T = RC\ln 4 \approx 1,4RC \tag{6.4}$$



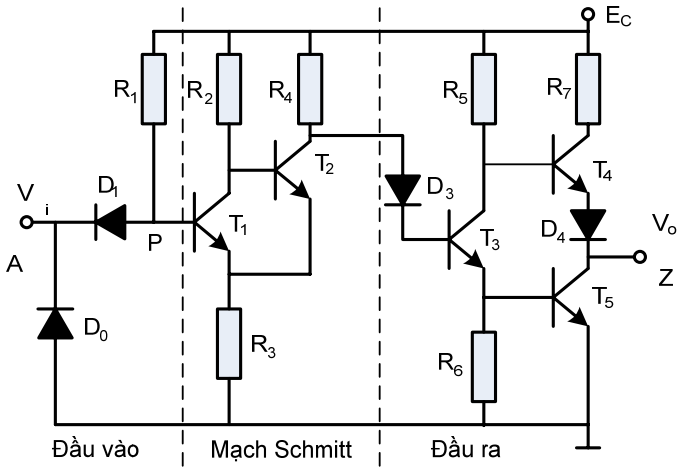
a)



b)

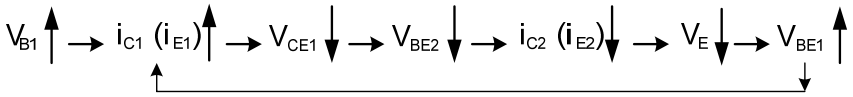
Hình 6.8: Bộ dao động đa hài dùng cổng NOR CMOS và giản đồ xung

6.2 TRIGƠ SCHMITT

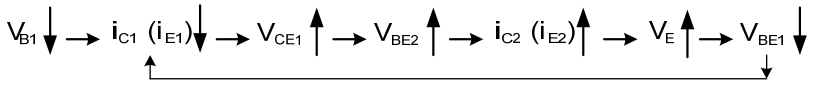


Hình 6.9: Sơ đồ nguyên lý của trigơ Schmitt

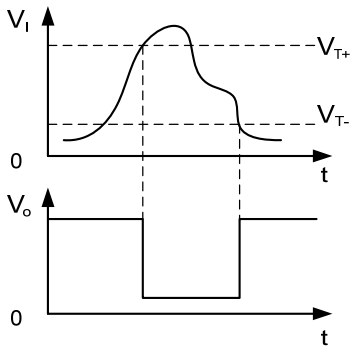
Hình 6.9 là sơ đồ nguyên lý của trigơ Schmitt, hay còn được gọi là bộ đảo pha trigơ Schmitt. Nó gồm 3 phần: mạch đầu vào, mạch Schmitt và tầng công suất đầu ra. Nguyên tắc làm việc của mạch như sau: Nếu V_{B1} ở mức thấp thì T_1 ngắt, T_2 thông bão hoà và ngược lại nếu V_{B1} ở mức cao thì T_1 thông bão hoà, T_2 ngắt. Khi V_{B1} tăng từ mức thấp lên mức cao đến trị số $V_{BE1} = V_{B1} - I_L R_3 = 0{,}5V$ thì T_1 bắt đầu chuyển từ trạng thái ngắt vào trạng thái khuếch đại. Do V_{B1} tiếp tục tăng nên $V_{CE1} = V_{BE2}$ giảm xuống. Sau khi T_2 rời khỏi trạng thái bão hoà mà V_{B1} tiếp tục tăng thì xảy ra quá trình phản hồi dương sau:



Nhờ phản hồi dương mạch điện nhanh chóng chuyển sang trạng thái T_1 thông bão hoà, T_2 ngắt. Nếu V_{B1} sau khi tăng đến cực đại thì bắt đầu giảm; khi V_{B1} giảm đến mức làm T_1 ra khỏi vùng bão hoà, T_2 ra khỏi vùng ngắt thì mạch điện lại xảy ra quá trình phản hồi dương sau:



Kết quả mạch điện nhanh chóng lật sang trạng thái T_1 ngắt, T_2 thông bão hoà. Chúng ta gọi giá trị điện áp đầu vào V_I trong quá trình tăng lên của nó đạt đến ngưỡng làm lật mạch Schmitt để đầu ra từ mức cao xuống mức thấp là ngưỡng trên V_{T+} và giá trị ngược lại là ngưỡng dưới của trigơ Schmitt V_{T-} (hình 6.10). Hiệu điện áp tương ứng với ngưỡng trên và ngưỡng dưới được gọi là độ chênh lệch điện áp chuyển mạch $\Delta V = V_{T+} - V_{T-}$.



Hình 6.10: Dạng sóng đầu vào V_i và đầu ra V_o của trigơ Schmitt

Trigơ Schmitt thực chất là một bộ so sánh hai ngưỡng nên nó được dùng ứng dụng khác nhau như: Các mạch dao động, các mạch so sánh, lọc nhiễu...

6.3 MẠCH ĐA HÀI ĐỢI

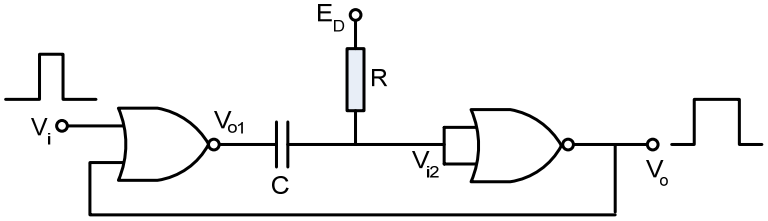
Mạch đa hài đợi có một trạng thái ổn định và một trạng thái tạm ổn định. Khi có tác dụng của xung ngoài, mạch có thể chuyển đổi từ trạng thái ổn định sang trạng thái tạm ổn định. Sau khi duy trì một thời gian, mạch sẽ tự động quay lại trạng thái ổn định. Thời gian tạm ổn định phụ thuộc vào các thông số của mạch mà không phụ thuộc vào xung kích. Mạch đa hài được ứng dụng trong các mạch định thời, tạo dạng xung, trễ…

6.3.1 Mạch đa hài đợi CMOS

6.3.1.1 Mạch đa hài đợi kiểu vi phân

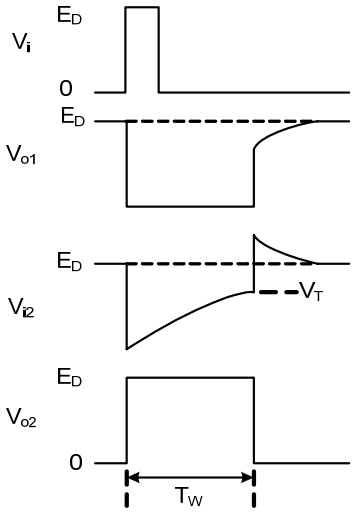
Hình 6.11 là sơ đồ nguyên lý của mạch đa hài đợi kiểu vi phân.

Tại trạng thái ổn định, $V_I = 0$ thì $V_{o1} = E_D$, $V_{i2} = E_D$, $V_{o2} = 0$. Khi có một xung kích thích đầu vào làm cho cổng 1 nhanh chóng cấm và đầu ra bằng 0, xem giản đồ 6.12.



Hình 6.11: Đa hài đợi kiểu vi phân dùng cổng NOR CMOS

Mạch điện RC sẽ nạp điện cho tụ điện C. Trong quá trình nạp, điện áp V_{i2} tăng dần đến ngưỡng V_T và làm cổng 2 đóng, điện áp $V_{o2} = 0$. Khi đó, cổng 1 nhanh chóng chuyển về trạng thái cấm và làm cho mạch đa hài đợi trở về trạng thái ổn định.



Hình 6.12: Dạng sóng của mạch đa hài đợi kiểu vi phân

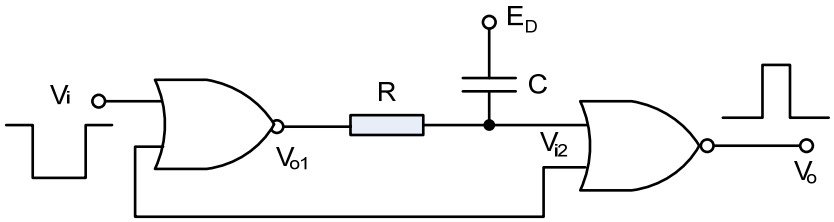
Độ rộng xung tại đầu ra của mạch được xác định bằng công thức sau:

$$T_W = (R + R_0)C * \ln\frac{E_D}{E_D - V_T} \quad (6.5)$$

trong đó R_0 là điện trở đầu ra của cổng 1, nếu $V_T = E_D/2$ thì:

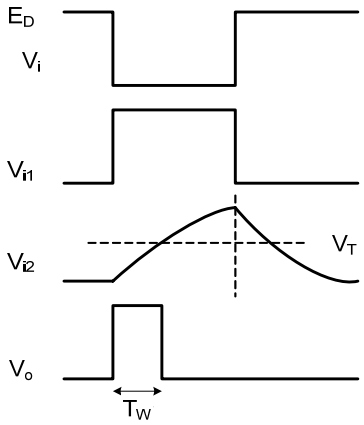
$$T_W = 0,7(R + R_0)C \quad (6.6)$$

6.3.1.2 Mạch đa hài đợi kiểu tích phân



Hình 6.13: Đa hài đợi kiểu tích phân dùng cổng NOR CMOS

Hình 6.13 biểu diễn sơ đồ nguyên lý của mạch đa hài đợi kiểu tích phân.



Hình 6.14: Dạng sóng của mạch đa hài đợi kiểu tích phân

Tại trạng thái ổn định, $V_i = 1$ thì $V_{o1} = 0$, $V_{i2} = 0$, $V_{o2} = 0$. Khi đầu vào V_i chuyển từ 1 xuống 0 đầu ra V_{o2} nhảy từ trạng thái 0 lên 1

và đồng thời mạch RC bắt đầu tích điện cho tụ điện C, khi điện áp $V_{i2} = V_T$ điện áp đầu ra V_{o2} chuyển xuống trạng thái 0. Sau khi hết xung đầu vào tụ điện phóng điện thông qua trở R và mạch trở về trạng thái ổn định.

Độ rộng xung đầu ra của mạch đa hài đợi (hình 6.14) được tính theo công thức:

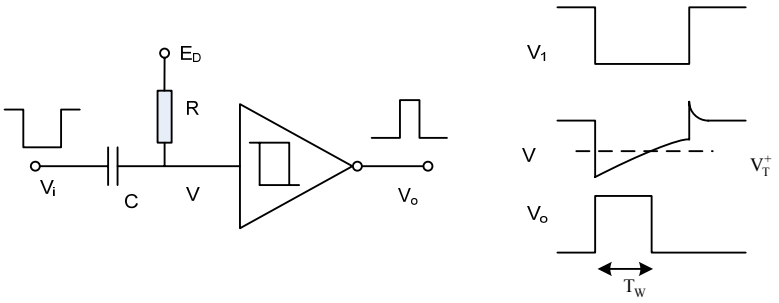
$$T_W = (R + R_0)C * \ln\frac{E_D}{E_D - V_T} \tag{6.7}$$

trong đó R_0 là điện trở đầu ra của cổng 1, nếu $V_T = E_D/2$ thì:

$$T_W = 0,7(R + R_0)C \tag{6.8}$$

6.3.1.3 Mạch đa hài đợi dùng trigơ Schmitt

Dựa vào đặc tính so sánh của trigơ Schmitt, mạch nguyên lý chỉ ra trên hình 6.15 là bộ đa hài đợi.



Hình 6.15: Sơ đồ nguyên lý và giản đồ thời gian của mạch đa hài dùng trigơ Schmitt

Độ rộng xung đầu ra phụ thuộc vào ngưỡng trên của trigơ Schmitt và giá trị của tụ điện C và điện trở R theo công thức sau:

$$T_W = RC * \ln\frac{E_D}{E_D - V_T^+} \tag{6.9}$$

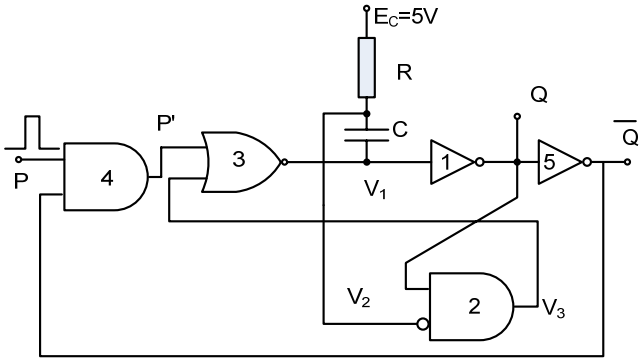
nếu $V_T = E_D/2$ thì: $T_W = 0,7RC$ (6.10)

6.3.2 Mạch đa hài đợi TTL

Hình 6.16 là sơ đồ nguyên lý mạch đa hài đợi họ TTL, trong đó các cổng 1, 2, 3 cấu trúc lên mạch flip-flop, cổng 4, 5 là mạch tạo dạng xung. Các cổng này thuộc họ TTL nên có mức logic 1 là 3,6V và logic 0 là 0,3V. Đầu vào V_2 biểu thị sử dụng mạch đảo. Mạch đảo này thông bão hoà thì $V_2 \sim 0,7V$, còn ngưỡng thông của nó cỡ 0,6V.

Tại trạng thái ổn định P = P' = 0. Mạch đảo đầu vào V_2 là bộ khuếch đại tranzito emitơ chung ở trạng thái bão hoà và khi đó $V_2 = 0,7V$, $V_3 = 0$, $V_1 = 1$, Q = 0, $\overline{Q} = 1$.

Khi có xung dương tác động ở đầu vào thì P = 1, P' = 1, $V_1 = 0$, Q = 1, $\overline{Q} = 0$, mạch ở trạng thái tạm ổn định. Do $\overline{Q} = 0$ khoá cổng 4, nên sau khi bị kích thích bởi sườn dương xung P thì mạch bị cách ly khỏi xung P.

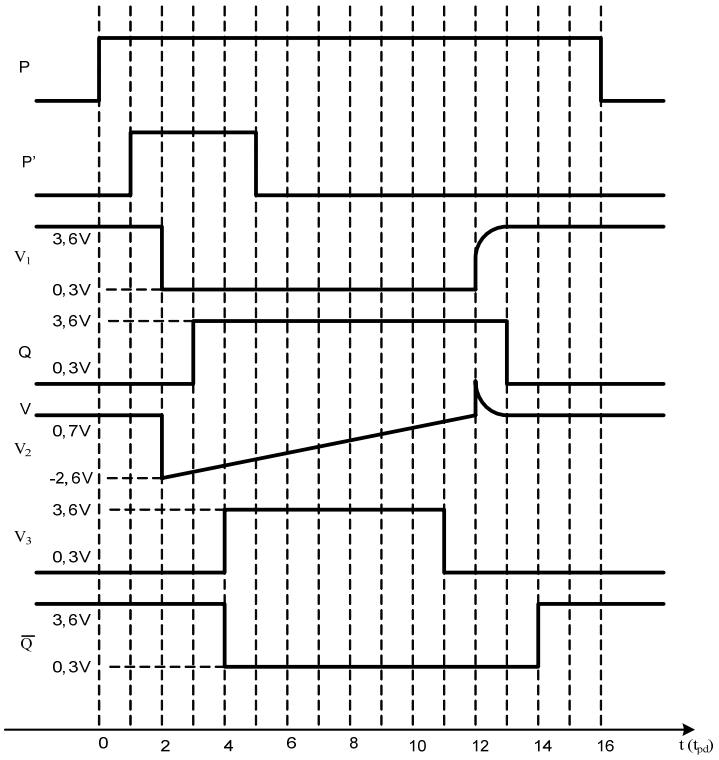


Hình 6.16: Sơ đồ nguyên lý mạch đa hài đợi họ TTL

Vì điện áp trên tụ C không tăng đột biến nên khi V_1 từ mức cao 3,6V đột biến xuống 0,3V thì V_2 từ mức 0,7V đột biến xuống -2,6V. Bắt đầu quá trình nạp điện của tụ điện C. V_2 tăng dần lên. Khi V_2 Tăng lên đến ngưỡng thông 0,6V thì sinh ra quá trình phản hồi sau:

$$V_2 \uparrow \rightarrow V_3 \downarrow \rightarrow V_1 \uparrow \rightarrow Q \downarrow$$

Quá trình này làm mạch nhanh chóng trở về trạng thái ổn định ban đầu $V_3 = 0$, $V_1 = 1$, $Q = 0$, $\overline{Q} = 1$. Tiếp đó tụ điện C phóng điện, V_2 dần dần hồi phục về 0,7V. Hình 6.17 chỉ ra giản đồ xung của mạch đa hài đợi họ TTL với giả thiết thời gian trễ truyền đạt của các cổng và bộ đảo pha đều bằng t_{pd}.



Hình 6.17: Giản đồ xung của mạch dao động đa hài đợi TTL với giả thiết độ trễ của các cổng là t_{pd}

Độ rộng xung ra được tính theo công thức $T_W = 0,7RC$. Mạch dao động đa hài đợi được thiết kế sẵn trong một số họ IC TTL như 74LS121, 74LS123… bằng cách thay đổi các giá trị tụ và trở mắc ngoài sẽ cho các xung đầu ra mong muốn.

6.4 IC ĐỊNH THỜI

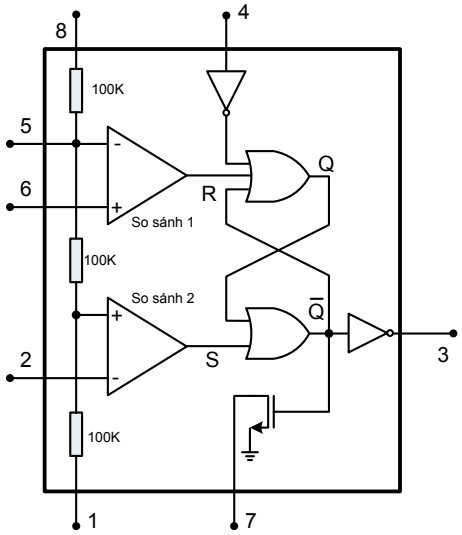
6.4.1 Mạch điện của IC 555

Bộ định thời 555 được sử dụng rất rộng rãi trong các bộ dao động đa hài, đa hài đợi và các bộ so sánh... Hình 6.18 là sơ đồ khối nguyên lý của IC định thời này, trong đó chức năng của các chân được chỉ ra trong bảng sau:

Bảng 6.1: Bảng mô tả chức năng của các chân trong IC

Chân	Chức năng	Chân	Chức năng
1	Đất – GND	5	Điện áp điều khiển
2	Chân kích thích	6	Chân ngưỡng
3	Đầu ra	7	Đầu phóng điện
4	Xoá – Reset	8	Nguồn – Vcc

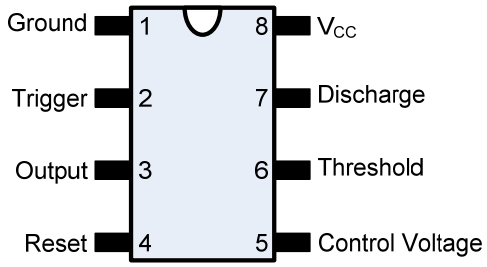
Sơ đồ nguyên lý của IC 555, gồm một mạch phân áp với 3 điện trở R (100kΩ) nối với chân 8. Ở đây chân 8 bao gồm cả nguồn nuôi của các bộ so sánh và các cổng logic trong mạch.



Hình 6.18: Sơ đồ khối nguyên lý của IC 555

Điện áp $+E_C$ nối với chân 8 có giá trị từ +5V ÷ +25V tuỳ theo mức biên độ của xung ở đầu ra.

Mạch gồm hai bộ so sánh (1) và (2). Điện áp đầu vào đảo của bộ so sánh thứ nhất có mức điện áp bằng $\frac{2}{3}E_C$, điện áp ở đầu vào thuận của bộ so sánh thứ hai bằng $\frac{1}{3}E_C$.



Hình 6.19: Sơ đồ chân của IC LMC555

Đầu ra của hai bộ so sánh được nối với đầu vào của Trigơ *RS,* Trigơ *RS* sử dụng trong mạch này dùng các cổng NOR, do đó mức tích cực là mức cao. Chân 4 được nối với đầu vào của một cổng NOR thông qua một cổng NOT có tác dụng điều khiển hoạt động của trigơ, điện áp chân 4 ở mức cao (mức "1") trigơ hoạt động bình thường, còn điện áp chân 4 ở mức thấp (mức "0") cấm trigơ hoạt động. Đầu ra $\overline{Q}$ của trigơ RS được đưa qua cổng NOT tới đầu ra ở chân 3, đồng thời $\overline{Q}$ nối với tranzito T để tạo đầu phóng điện.

Bảng chức năng của IC 555 được chỉ ra trên bảng 6.2.

Bảng 6.2: Bảng chức năng của IC 555

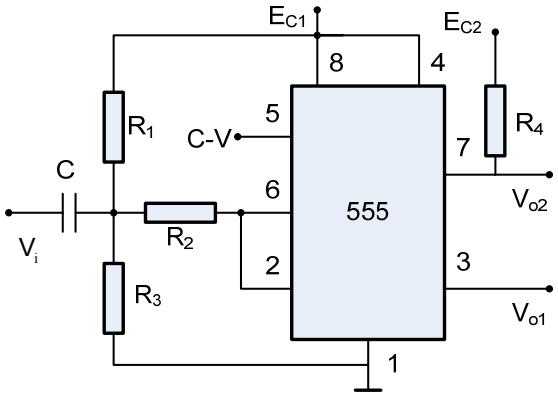
TH	$\overline{TRIG}$	$\overline{R}$	**OUT**	**DIS**
X	X	L	L	Thông
$>\frac{2}{3}E_C$	$>\frac{1}{3}E_C$	H	L	Thông

TH	$\overline{TRIG}$	$\overline{R}$	OUT	DIS
$< \frac{2}{3}E_C$	$> \frac{1}{3}E_C$	H	Không đổi	Không đổi
X	$> \frac{1}{3}E_C$	H	H	Ngắt

6.4.2 Một vài ứng dụng của IC định thời 555

6.4.2.1 Trigơ Schmitt

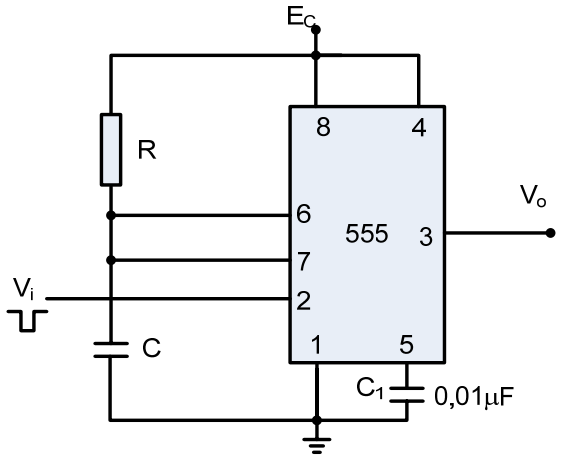
Hình 6.20 là sơ đồ nguyên lý của trigơ Schmitt dùng IC 555. Với sơ đồ này ngưỡng trên $V_{T+} = \frac{2}{3} * E_{C1}$ và ngưỡng dưới $V_{T-} = \frac{1}{3} * E_{C1}$. Độ chênh lệch điện áp $\Delta V = V_{T+} - V_{T-} = \frac{1}{3} * E_{C1}$. Nếu đưa điện áp vào đầu vào C-V thì có thể điều chỉnh được V_{T+}, V_{T-} và ΔV.



Hình 6.20: Mạch trigơ Schmitt dùng IC 555

6.4.2.2 Mạch đa hài đợi

Hình 6.21 trình bày sơ đồ khối của mạch đa hài đợi dùng IC 555.



Hình 6.21: Mạch đa hài đợi dùng IC 555

Trong sơ đồ hình 6.21, tụ C_1 tác dụng cùng điện trở R (trong sơ đồ nguyên lý hình 6.18) tạo thành mạch lọc thông thấp.

Giả sử lúc đầu tụ C chưa tích điện nên $V_C = 0$. Khi đóng mạch nguồn điện, tụ C tích điện, mạch tích điện từ $+E_C$ qua R, qua C xuống đất. Tụ C tích điện, điện áp u_C tăng lên, cho đến khi $u_C \geq \frac{2}{3}E_C$, điện áp ra của bộ so sánh (1) R = 0, dẫn đến $\overline{Q} = 1$. Vì $\overline{Q} = 1$ nên cho tranzito T mở, tụ C phóng điện qua R, qua tranzito xuống đất. Tụ C phóng điện đến khi $u_C = 0$. Đây là trạng thái ổn định bền (trạng thái đợi) của mạch.

Khi có xung vuông đưa tới đầu vào, nếu biên độ xung vào đủ lớn, điện áp ra của bộ so sánh (2) S = 1, do đó Q = 1 và $\overline{Q} = 0$, bắt đầu thời gian kéo dài xung ở đầu ra. Vì $\overline{Q} = 0$ nên T cấm, tụ C tích điện, mạch tích điện từ $+E_C$, qua R, qua C xuống đất, tụ C nạp điện với hằng số thời gian:

$$\tau_n = R.C \qquad (6.11)$$

Tụ C tích điện, điện áp u_C tăng, cho đến khi $u_C \geq \frac{2}{3}E_C$, đối với bộ so sánh (1) điện áp ra R = 0, dẫn đến $\overline{Q}$ = 1, chấm dứt thời gian kéo dài xung ở đầu ra. Vì $\overline{Q}$ = 1, T mở , tụ C phóng điện, qua R, qua T xuống đất, cho đến khi u_c= 0, trở lại trạng thái ban đầu.

Khi tụ C tích điện, điện áp trên tụ C tăng theo quy luật hàm mũ:

$$u_C = E_C\left[1 - \exp\left(-\frac{t}{t_{nap}}\right)\right] \tag{6.12}$$

Thời gian tụ nạp từ điện áp 0V đến $\frac{2}{3}E_C$ được trình bày như sau:

$$u_C = E_C\left(1 - e^{-\frac{t_x}{\tau}}\right) = \frac{2}{3}E_C \tag{6.13}$$

Suy ra:

$$1 - e^{-\frac{t_x}{\tau_{nap}}} = \frac{2}{3}$$

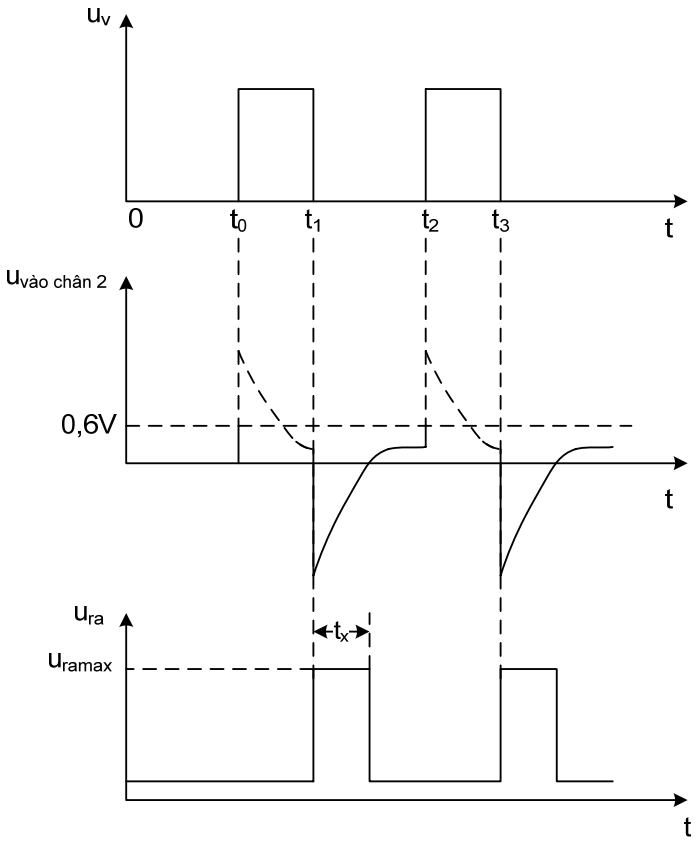
hay là:

$$\frac{1}{3} = e^{-\frac{t_x}{\tau_{nap}}};\ e^{\frac{t_x}{\tau_{nap}}} = 3$$

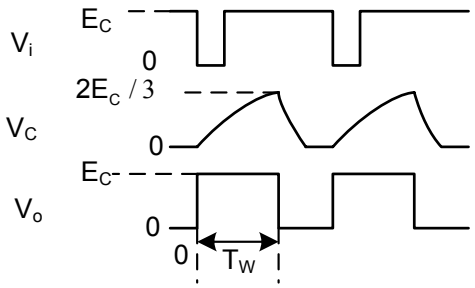
$$tx = \tau_{nạp}\ln 3 = 1,1\tau_{nạp}$$

$$t_x = 1,1RC \tag{6.14}$$

Giản đồ điện áp - thời gian minh họa hoạt động của mạch đa hài đợi dùng IC 555 được trình bày trên hình 6.22.



Hình 6.22: Giản đồ điện áp - thời gian của mạch đa hài đợi dùng IC 555



Hình 6.23: Mạch đa hài đợi dùng IC 555 và dạng sóng

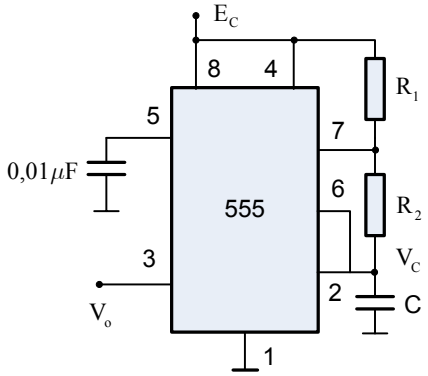
Hình 6.23 là sơ đồ nguyên lý và giản đồ thời gian của mạch đa hài đợi dùng IC 555, trong đó RC là mạch định thời. Độ kéo dài xung đầu ra được xác định bằng công thức:

$$T_W \simeq RC \ln 3 \simeq 1,1RC \qquad (6.15)$$

Mạch dao động đa hài đợi này yêu cầu độ rộng xung đầu vào nhỏ hơn độ rộng xung đầu ra, nếu nó lớn hơn thì yêu cầu dùng thêm mạch vi phân ở đầu vào.

6.4.2.3 Mạch đa hài

Sơ đồ nguyên lý mạch đa hài dùng IC 555 được trình bày trên hình 6.24.



Hình 6.24: Mạch đa hài dùng IC 555

Lúc đầu đóng mạch nguồn nuôi một chiều, tụ C chưa kịp nạp điện, điện áp trên tụ $U_C = 0$, do đó điện áp ra của bộ so sánh (1) R = 0, điện áp ra của bộ so sánh (2) S = 1, dẫn đến Q = 1, $\overline{Q} = 0$ và T cấm. Tụ C tích điện, mạch tích điện từ $+E_C$, qua R_1, R_2, qua C xuống đất, tụ C nạp điện với hằng số thời gian: nạp= $(R_1 + R_2).C$. Tụ C nạp điện, điện áp trên tụ C tăng lên, cho đến khi $u_C \geq \frac{2}{3}E_C$, điện áp ra của bộ so sánh (1) R = 1,

$\overline{Q}$ = 1, dẫn đến tranzito T thông, tụ C phóng điện, mạch phóng điện từ cực dương của tụ C, qua R_2, qua T xuống đất đến cực âm của tụ C. Tụ C phóng điện đến khi điện áp trên tụ $u_C \le \frac{1}{3}E_C$, điện áp ra của bộ so sánh (2) R = 0, dẫn đến Q = 1, $\overline{Q}$ = 0. Do $\overline{Q}$ = 0 nên tranzito T bị cấm, tụ C lại được nạp điện, quá trình được lặp lại như cũ. Hiện tượng này tiếp diễn liên tục và tuần hoàn.

Lưu ý:

Khi mới đóng điện nguồn tụ C nạp từ 0V đến $\frac{2}{3}E_C$, sau đó tụ phóng điện từ $\frac{2}{3}E_C$ xuống $\frac{1}{3}E_C$, rồi lại nạp từ $\frac{1}{3}E_C$ đến $\frac{2}{3}E_C$.

Thời gian nạp và phóng của tụ được tính theo công thức (6.16) và (6.17).

+ Thời gian nạp:

$$t_{nạp} = \tau_{nạp} \ln 2$$

$$t_{nạp} = 0,7(R_1 + R_2)C \qquad (6.16)$$

+ Thời gian phóng:

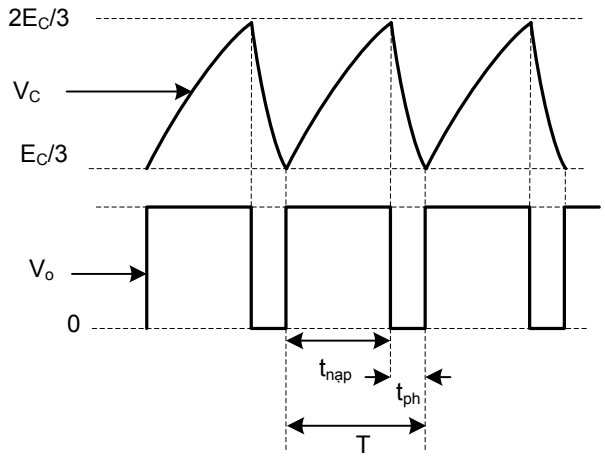
$$t_{ph} = \tau_{ph} \ln 2 \approx 0,7R_2C \qquad (6.17)$$

+ Chu kỳ của xung ở đầu ra:

$$T = t_{nạp} + t_{ph} = 0,7(R_1 + 2R_2)C \qquad (6.18)$$

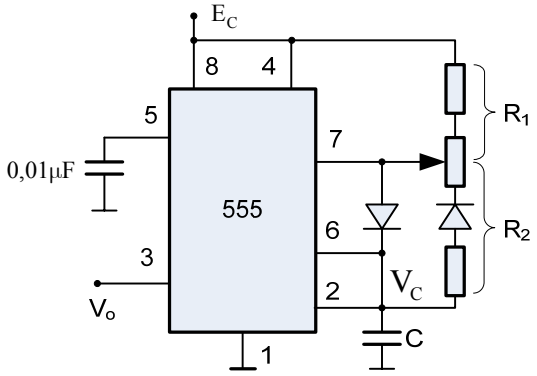
Dạng tín hiệu ở các cực của mạch được trình bày trên hình 6.25.

Do thời gian phóng, nạp không bằng nhau, nên xung vuông ra không đối xứng.



Hình 6.25: Dạng điện áp tại các chân của mạch đa hài dùng IC 555

$$f = \frac{1}{T} = \frac{1,43}{(R_1 + 2R_2)C} \tag{6.19}$$



Hình 6.26: Mạch đa hài điều chỉnh được độ lấp đầy xung dùng IC 555

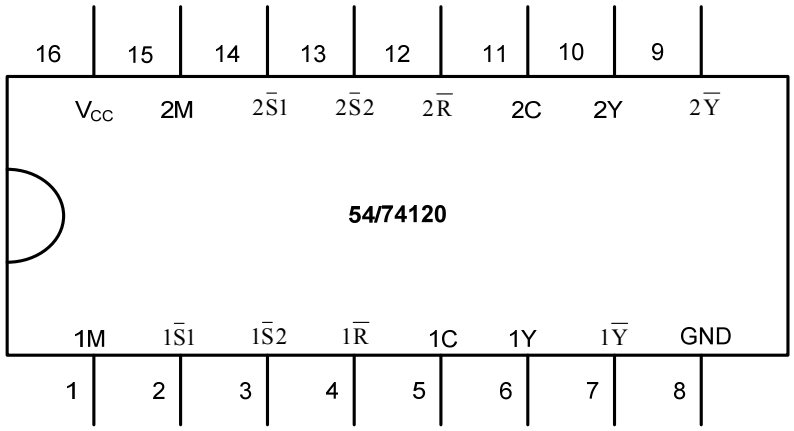
Như ta thấy xung đầu ra có độ lấp đầy phụ thuộc vào cả điện trở R_1 và R_2 và không thể tạo ra xung vuông với độ lấp đầy bằng 50% thông qua việc thay đổi giá trị R_1 và R_2. Để có được xung vuông với độ lấp đầy bằng 50%, người ta sử dụng mạch có thêm 2 điốt khi đó trở phóng và nạp điện cho tụ có thể thay đổi độc lập và tạo ra xung mong muốn.

Hình 6.26 là sơ đồ nguyên lý của mạch đa hài dùng IC 555 mà độ lấp đầy có thể thay đổi được.

6.5 MỘT SỐ IC THÔNG DỤNG

6.5.1 IC 54/74120

Hình 6.27 là sơ đồ chân của IC 54/74120. IC54/74120 là IC điều khiển/đồng bộ xung kép.



Hình 6.27: Sơ đồ chân IC54/74120

Loại IC này có thể phát ra một xung hoặc một dãy xung đồng bộ nhờ chức năng điều khiển. IC này rất hữu dụng trong các mạch điều khiển đồng bộ. Nó có chức năng chốt trạng thái hoạt động để các xung đầu ra không bị cắt.

Các xung đồng bộ đơn được thiết kế để đồng bộ các tín hiệu không đồng bộ. Các đầu vào $\bar{S}1$, $\bar{S}2$ hoặc $\bar{R}$ sẽ cho phép tạo xung ra. Đầu vào điều khiển M xác định xem một dãy xung vào hay chỉ một xung vào được đi qua mạch. Khi không có lệnh dừng thì xung điều khiển đồng hồ sẽ tiếp tục cho xung đồng hồ đi qua mạch miễn là đầu vào điều khiển chế độ vẫn ở mức thấp. Khi đầu vào điều khiển chế độ ở mức cao thì chỉ có một xung được đi qua.

Bảng chức năng của IC 74120 được chỉ ra trong bảng 6.3.

Bảng 6.3: Bảng chức năng của IC 74120

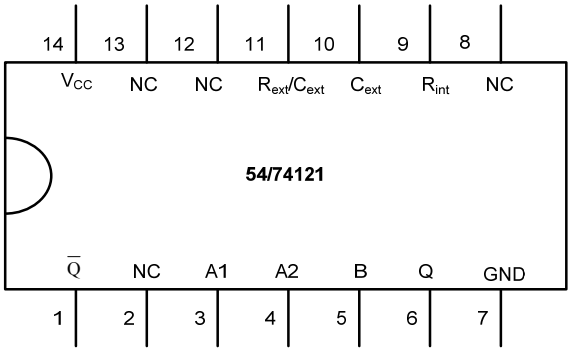
Đầu vào			Chức năng
$\overline{R}$	$\overline{S}1$	$\overline{S}2$	
X	L	X	Cho phép xung ra
X	X	L	Cho phép xung ra
L	H	H	Cấm xung ra
H	↓	H	Bắt đầu cho xung ra
H	H	↓	Bắt đầu cho xung ra
↓	H	H	Dừng không cho xung ra
H	H	H	Tiếp tục ↑

H: Mức cao; L: Mức thấp.

↓: Chuyển trạng thái từ H xuống L.

↑: Bắt đầu hoạt động bởi sự chuyển đổi trạng thái ↓ cuối cùng.

6.5.2 IC 54/74121



Hình 6.28: Sơ đồ chân IC 54/74121

IC 54/74121 là IC đơn ổn (đa hài đợi) có đầu vào là trigơ Schmitt. Hình 6.28 giới thiệu sơ đồ chân của IC này.

Độ rộng xung ra của IC này có thể điều khiển được:

+ Với R_{int} … 35ns.

+ Với R_{ext}/C_{ext} … 40ns đến 28s.

Bảng chức năng của IC 54/74121 được chỉ ra trên bảng 6.4.

Bảng 6.4: Bảng chức năng của IC 54/74121

Đầu vào			**Lối ra**	
A1	A2	B	Q	$\overline{Q}$
L	X	H	L	H
X	L	H	L↑	H↑
X	X	L	L↑	H↑
H	H	X	L↑	H↑
H	↓	H	⊓	⊔
↓	H	H	⊓	⊔
↓	↓	H	⊓	⊔
L	X	↑	⊓	⊔
X	L	↑	⊓	⊔

↓: Chuyển trạng thái từ H xuống L.

↑: Tín hiệu tại đầu vào A và B được thiết lập đủ dài để hoàn thành bất kỳ xung tới nào trước khi bắt đầu lại.

Đặc điểm của IC này là có hai đầu vào hoạt động tại mức tích cực âm và một đầu vào hoạt động tại mức tích cực dương. Các đầu vào này được sử dụng như là các đầu vào khống chế.

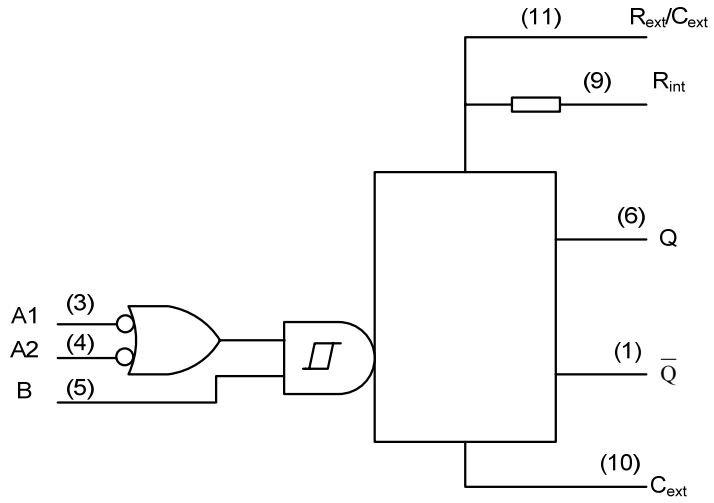
Trigơ Schmittt xuất hiện ở đầu vào B cho phép các tín hiệu dao động tự do với tốc độ thấp cỡ 1V/s đi qua. Do vậy mạch này có thể hoạt động với mức nhiễu cho phép là 1,2V.

Độ rộng xung ra có thể thay đổi từ 40ns đến 28s. bằng cách chọn các tham số định thời. Nếu không có các linh kiện định thời (R_{int} nối với V_{CC}; C_{ext} và C_{ext}/R_{ext} để hở) thì độ rộng xung ra đạt được 30÷35ns.

Độ rộng xung ra của IC này được xác định theo công thức:

$$T_W = 0,7C_{ext}.R_T$$

Hình 6.29 chỉ ra sơ đồ logic của IC 54/74121.



Hình 6.29: Sơ đồ logic của IC 54/74121

Lưu ý:

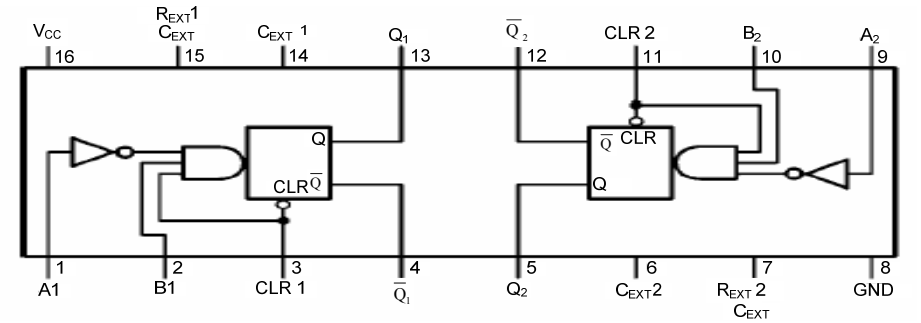
+ Khi nối tụ phụ bên ngoài ta nối giữa tụ C_{ext} và R_{ext}/C_{ext}.

+ Để sử dụng điện trở định thời bên trong IC ta nối điện trở R_{int} với V_{CC}. Để thay đổi độ rộng xung một cách chính xác và tuần hoàn, ta nối điện trở phụ bên ngoài nằm giữa R_{ext}/C_{ext} và $V_{CC,}$ điện trở R_{int} để hở.

6.5.3 IC 54/74123

IC 54/74 123 bao gồm hai bộ dao động một trạng thái ổn định độc lập nhau. Cả hai bộ này đều có thể kích khởi lại.

Hình 6.30 là sơ đồ logic của IC 54/74123.



Hình 6.30: Sơ đồ logic của IC 54/74123

Bảng 6.5 là bảng chức năng của IC 54/74123.

Bảng 6.5: Bảng chức năng của IC 54/74121

Lối vào			**Đáp ứng**
A	**B**	**CLR**	
X	X	L	Không kích khởi
↓	L	X	Không kích khởi
↓	H	H	Kích khởi
H	↑	X	Không kích khởi
L	↑	H	Kích khởi
L	H	↑	Kích khởi

Độ rộng xung ra được lựa chọn bởi điện trở R_x và tụ C_x và được tính bởi công thức (tụ $C_x > 1000pF$):

$$T_W \approx 0{,}28R_xC_x(1+0{,}7/R_x).$$

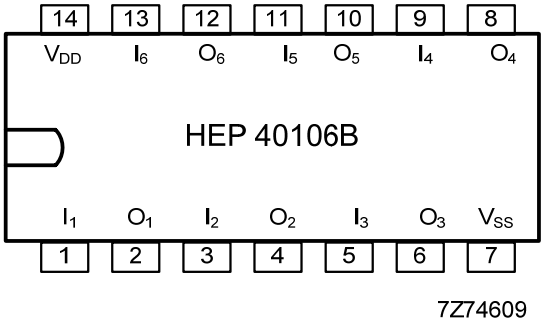
6.5.4 Trigơ Schmittt

IC HEF 40106B bao gồm sáu cổng đảo Schmittt.

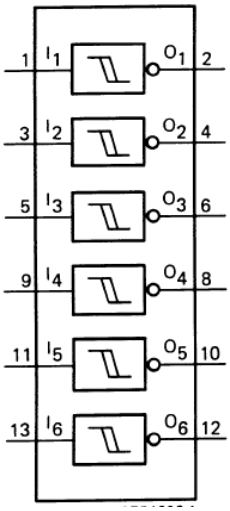
IC này được sử dụng để loại trừ nhiễu hoặc thay đổi dạng sóng thành xung vuông một cách chậm chạp.

Sự sai khác giữa điện áp ngưỡng cộng (V_P) và ngưỡng trừ (V_N) chính là điện áp trễ (V_H).

Hình 6.31 và 6.32 là sơ đồ chân và sơ đồ logic của IC HEF 40106B.



Hình 6.31: Sơ đồ chân của IC 40106B



Hình 6.32: Sơ đồ logic của IC 40106B

TÓM TẮT

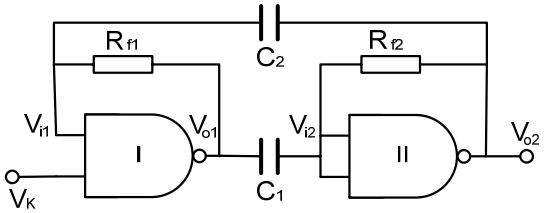
Trong chương này chúng ta đã tìm hiểu các mạch tạo xung. Mạch dao động xung tự kích không cần tín hiệu ngoài đưa vào; sau khi được cấp nguồn một chiều mạch tự động sinh ra xung vuông. Thuộc loại dao động tự kích này có các mạch: bộ dao động đa hài cơ bản cổng NAND họ TTL, bộ dao động vòng, bộ dao động thạch anh, bộ dao động đa hài cơ bản CMOS.

Mạch tạo dạng xung không tự động phát xung nhưng có thể biến tín hiệu đầu vào hình dạng khác thành xung vuông theo yêu cầu của mạch số. Trong số mạch tạo dạng xung, chúng ta đã tìm hiểu: trigơ Schmitt và đơn ổn.

Cách mạch phát xung và tạo dạng xung trên đây, ngoài dùng làm xung đồng hồ ra còn có ứng dụng vo cùng rộng rãi trong các hệ thống xung - số. Bộ dao động đa hài thường dùng làm bộ tạo xung chuẩn thời gian và chuẩn tần số. Mạch đơn ổn thường dùng để định thời và làm trễ xung. Trigơ Schmitt ngoài ứng dụng tạo dạng xung còn ứng dụng so sánh mức và giám sát mức…

CÂU HỎI ÔN TẬP

1. Trong mạch dao động đa hài cơ bản dùng cổng NAND họ TTL, hình BT 6.1, nếu giá trị điện trở $R_{f1} = 5*R_{f2} = 10$ kΩ, giá trị $C_1 = C_2 = 1\mu F$ thì mạch có hoạt động như thế nào? Dạng tín hiệu tương đối đầu ra sẽ như thế nào?

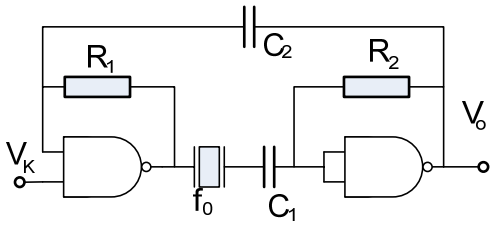


Hình BT 6.1: Bộ dao động đa hài cấu trúc bằng cổng NAND

2. Với câu hỏi như câu 1 và giả thiết $R_1 = 3$ kΩ, tính tần số dao động của mạch và vẽ dạng sóng đầu ra.

3. Đặc điểm nổi bật nhất của mạch dao động đa hài dùng thạch anh là gì?

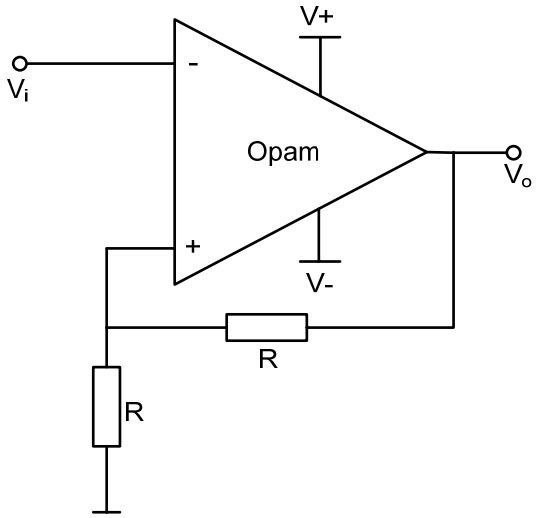
4. Trong mạch dao động đa hài dùng thạch anh như hình BT 6.4, nếu không có tụ C_1, đầu ra của thạch anh được nối trực tiếp với đầu vào của cổng NAND thứ hai thì mạch sẽ hoạt động như thế nào ?



Hình BT 6.4: Mạch dao động đa hài thạch anh

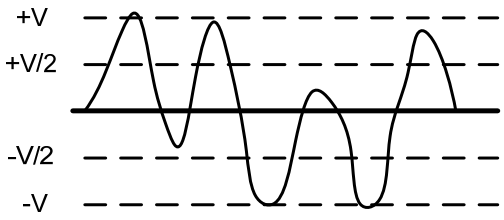
5. Đặc điểm quan trọng nhất của trigơ Schmitt là gì?

6. Mạch có sơ đồ nguyên lý như hình BT 6.6 có chức năng như thế nào?



Hình BT 6.6

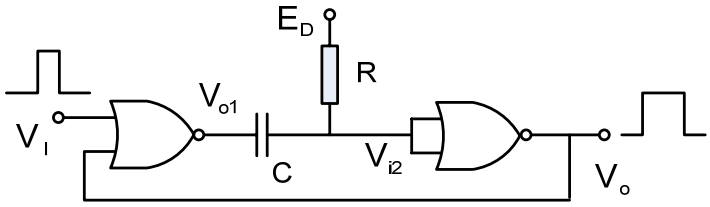
7. Với mạch điện như câu hỏi 6, nếu tín hiệu đầu vào có dạng tín hiệu như hình BT 6.7, tín hiệu đầu ra như thế nào?



Hình BT 6.7

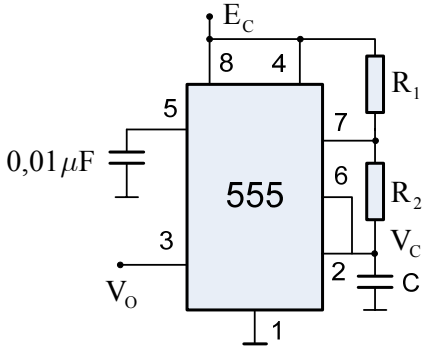
8. Chức năng của mạch đa hài đợi là gì?

9. Trong mạch đa hài đợi kiểu vi phân như hình BT 6.9, nếu xung điều khiển có độ rộng lớn hơn xung đa hài đợi đầu ra thì tín hiệu đầu ra như thế nào?



Hình BT 6.9

10. Trong mạch đa hài hình BT 6.10, nếu điện trở R_2 bị nối tắt mạch sẽ hoạt động như thế nào?



Hình BT 6.10

Chương 7

BỘ NHỚ BÁN DẪN

GIỚI THIỆU

Bộ nhớ bán dẫn thay thế các loại bộ nhớ bằng vật liệu từ. Các tiến bộ mới của công nghệ bán dẫn trong thời gian gần đây đã cung cấp nhiều mạch nhớ loại MSI và LSI có độ tin cậy cao và giá thành hạ. Vào đầu thập kỷ 60 của thế kỷ XX, giá thành thương phẩm của một bit nhớ vào khoảng 2USD. Đến nay (những năm đầu thế kỷ XXI), giá thương phẩm của 128 Mbyte vào khoảng 20 USD. Như vậy giá thành thương phẩm của một bit nhớ sau khoảng 40 năm đã giảm đi khoảng 105.10^6 lần. Bộ nhớ bán dẫn điển hình có các tế bào nhớ sắp xếp theo hình chữ nhật, gắn trong khối hộp nhỏ bằng nhựa dạng DIP (Dual in line package). Tế bào nhớ cơ bản là một mạch trigơ, tranzito hay mạch có khả năng tích trữ điện tích, tế bào nhớ này dùng để lưu trữ một bit thông tin.

Trong phần này giới thiệu một số bộ nhớ bán dẫn cơ bản.

7.1 KHÁI NIỆM CHUNG

7.1.1 Khái niệm

Bộ nhớ là một thiết bị có khả năng lưu trữ thông tin (nhị phân). Muốn sử dụng bộ nhớ, trước tiên ta phải viết dữ liệu và các thông tin cần thiết vào nó, sau đó lúc cần thiết phải lấy dữ liệu đã viết trước đó để sử dụng. Thủ tục viết vào và đọc ra phải được kiểm soát chặt chẽ, tránh nhầm lẫn nhờ định vị chính xác từng vị trí ô nhớ và nội dung của nó theo một mã địa chỉ duy nhất.

7.1.2 Những đặc trưng chính của bộ nhớ

7.1.2.1 Dung lượng của bộ nhớ

Dung lượng bộ nhớ là số bit thông tin tối đa có thể lưu trữ trong nó. Dung lượng cũng có thể biểu thị bằng số từ nhớ nbit. ***Từ nhớ nbit*** là số bit (n) thông tin mà ta có thể đọc hoặc viết đồng thời vào bộ nhớ. Ví dụ: Một bộ nhớ có dung lượng là 256bit; nếu nó có cấu trúc để có thể truy nhập cùng một lúc 8bit thông tin, thì ta cũng có thể biểu thị dung lượng bộ nhớ là 32 từ nhớ x 8bit = 32byte.

7.1.2.2 Cách truy nhập thông tin

Các bộ nhớ có thể có một trong hai cách truy nhập thông tin.

Truy nhập trực tiếp hay còn gọi là truy nhập ngẫu nhiên (random access). Ở cách này, không gian bộ nhớ được chia thành nhiều ô nhớ. Mỗi ô nhớ chứa được 1 từ nhớ nbit và có một địa chỉ xác định, mã hoá bằng số nhị phân kbit. Như vậy, người sử dụng có thể truy nhập trực tiếp thông tin ở ô nhớ có địa chỉ nào đó trong bộ nhớ. Mỗi bộ nhớ có kbit địa chỉ sẽ có 2^k ô nhớ và có thể viết được 2^k từ nhớ nbit.

Truy nhập liên tiếp (serial access) hay còn gọi là kiểu truy nhập tuần tự. Các đĩa từ, băng từ, trống từ, thanh ghi dịch… có kiểu truy nhập này. Các bit thông tin được đưa vào và lấy ra một cách tuần tự.

7.1.2.3 Tốc độ truy nhập thông tin

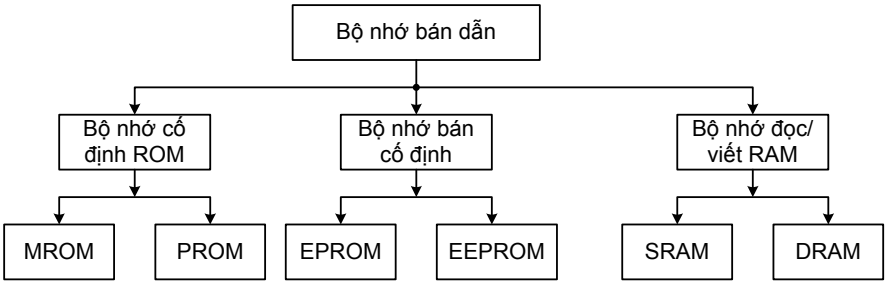
Đây là thông số rất quan trọng của bộ nhớ. Nó được đặc trưng bởi thời gian cần thiết để truy nhập thông tin.

Thời gian truy nhập thông tin ở các bộ nhớ truy nhập kiểu trực tiếp gồm thời gian tìm địa chỉ của ô nhớ và thời gian đọc/viết thông tin trên đó. Thời gian truy nhập thông tin phụ thuộc chủ yếu vào công nghệ chế tạo. Với công nghệ MOS thì thời gian truy nhập khoảng 30 đến vài trăm ns.

Ở các bộ nhớ truy nhập kiểu tuần tự, thời gian truy nhập phụ thuộc vào vị trí của thông tin cần truy nhập trong tập tin (file). Đối với các băng từ, đĩa từ thời gian truy nhập của nó được định nghĩa là thời gian trung bình hoặc cực đại để truy nhập một thông tin và nằm trong khoảng vài ms đến nhiều s.

7.1.3 Phân loại

Dựa trên thời gian viết và cách viết, có thể chia thành bộ nhớ cố định, bộ nhớ bán cố định và bộ nhớ đọc/viết được. Bộ nhớ có nội dung được viết sẵn một lần khi chế tạo được gọi là bộ nhớ cố định và được ký hiệu là ROM (Read Only Memory). Sau khi đã được viết (bằng mặt nạ-mask) từ nhà máy thì ROM loại này không viết lại được nữa đó chính là MROM. PROM là một dạng khác, các bit có thể được viết bằng thiết bị viết của người sử dụng trong một lần (Programmable ROM).



Hình 7.1: Sơ đồ phân loại bộ nhớ bán dẫn

Bộ nhớ có thể đọc/viết nhiều lần được gọi là RAM (Random Access Memory) gồm hai loại: bộ nhớ RAM tĩnh-SRAM (Static RAM) thường được xây dựng trên các mạch điện tử trigơ và RAM động-DRAM (Dynamic RAM) được xây dựng trên cơ sở nhớ các điện tích ở tụ điện; bộ nhớ này phải được hồi phục nội dung đều đặn, nếu không nội dung sẽ mất đi theo sự rò điện tích trên tụ. Giữa ROM và RAM có một lớp các bộ nhớ được gọi là EPROM (Erasable PROM), dữ liệu trong đó có thể xoá được bằng tia cực tím và viết lại được,

EEPROM (Electric EPROM) có thể xoá được bằng dòng điện. Các loại này còn được gọi là bộ nhớ bán cố định. Các bộ nhớ DRAM thường thoả mãn những yêu cầu khi cần bộ nhớ có dung lượng lớn; trong khi đó khi cần có tốc độ truy xuất lớn thì phải dùng các bộ nhớ SRAM có giá thành đắt hơn. Nhưng cả hai loại này đều có nhược điểm là thuộc loại "bay hơi" (volatile), thông tin sẽ bị mất đi khi nguồn nuôi bị ngắt. Do vậy, các chương trình dùng cho việc khởi động PC như BIOS thường phải nạp trên các bộ nhớ ROM.

7.1.4 Tổ chức của bộ nhớ

Bộ nhớ thường được tổ chức gồm nhiều vi mạch nhớ được ghép lại để có độ dài từ và tổng số từ cần thiết. Những chip nhớ được thiết kế sao cho có đầy đủ một số chức năng của bộ nhớ như:

- Một ma trận nhớ gồm các ô nhớ, mỗi ô nhớ ứng với một bit nhớ.
- Mạch logic giải mã địa chỉ ô nhớ.
- Mạch logic cho phép đọc nội dung ô nhớ.
- Mạch logic cho phép viết nội dung ô nhớ.
- Các bộ đệm vào, bộ đệm ra và bộ mở rộng địa chỉ.

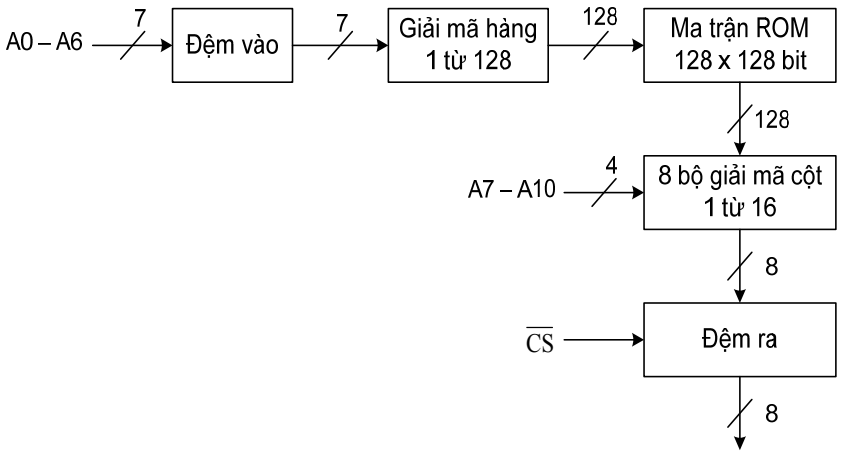
Cách tổ chức đơn giản nhất là tổ chức theo từ (word organized) với sự chọn tuyến tính. Một ma trận nhớ như vậy có độ dài của cột bằng số lượng từ (W) và độ dài của hàng bằng số lượng bit (B) trong một từ. Bộ chọn từ phải giải mã 1 từ (W), nghĩa là giải mã để có một đầu ra duy nhất cho một từ trong bộ nhớ. Phương pháp này có thời gian truy nhập ngắn nhưng cần một bộ giải mã lớn khi tổng số từ lớn, do đó làm tăng giá thành sản phẩm.

Kích thước của phần giải mã địa chỉ sẽ giảm đi khi tổ chức ma trận nhớ và phần logic chọn từ cho phép giải mã hai bước. Ma trận nhớ sử dụng giải mã hai bước ứng với từ vật lý và từ logic. Từ vật lý bao gồm số lượng bit trong một hàng của ma trận. Từ logic bao gồm số lượng bit tương ứng với một từ logic được nhận biết và gửi ra cùng

một lúc. Cần hai bộ giải mã: một bộ giải mã hàng để chọn một từ vật lý và một bộ giải mã cột gồm có một vài mạch hợp kênh chọn một từ logic từ một từ vật lý đã chọn. Một từ vật lý được chia thành S từ logic. Bộ giải mã hàng là bộ giải mã chọn 1 từ W mà B = W/S và bộ chọn cột chứa B bộ hợp kênh một đường từ S.

Ví dụ: Sơ đồ ROM dung lượng 2048 x 8 (2048 từ, mỗi từ chứa 8bit) tổ chức giải mã hai bước như hình 7.2.

Ma trận nhớ là 128 x 128, như vậy có 128 = 2^7 từ vật lý. Một từ vật lý được chọn bởi 7 đường địa chỉ từ A0 đến A6. Bộ giải mã hàng chọn 1 hàng từ 128 hàng. Một từ vật lý được chia thành 128/8 = 16 nhóm 8bit. Nhóm thứ nhất chứa những bit có trọng số cao nhất của 16 từ logic. Nhóm thứ hai chứa các bit cao tiếp theo của 16 từ logic…Nhóm cuối cùng chứa những bit thấp nhất của 16 từ logic, do đó S = 16. Như vậy, những bộ giải mã cột gồm 8 bộ hợp kênh một đường từ 16 đường để cung cấp một từ locgic ra 8bit. Những địa chỉ từ A7 đến A10 điều khiển các bộ giải mã cột. Trường hợp đặc biệt khi số phần tử trong một từ vật lý bằng số bit trong một từ vật lý thì đó là bộ nhớ tổ chức theo bit có nghĩa là mỗi từ logic có độ dài 1bit.



Hình 7.2: Ví dụ về bộ giải mã cho ma trận ROM 128 x 128

Các bộ đệm ra đảm bảo các mức logic mong muốn và cung cấp đủ dòng điện, ngoài ra nó còn có đầu ra colectơ hở hoặc 3 trạng thái cho phép nối chung đầu ra của một vài chip với nhau. Bộ đệm ra được điều khiển bởi một hay nhiều đầu vào như chọn mạch CS (Chip Select), cho phép mở CE (Chip Enable) hay cho phép mở đầu ba trạng thái OE (Output Enable).

7.2 BỘ NHỚ CỐ ĐỊNH - ROM

Để một máy tính điện tử tiến hành các hoạt động bình thường dù đơn giản hay phức tạp thì cần phải cung cấp cho máy trước tiên là các lệnh từng bước tiến hành. Tập hợp danh sách các lệnh được gọi là chương trình. Việc nhập một chương trình nào đó từ đĩa hay từ bàn phím vào máy tính phải tuân theo sự điều khiển của một chương trình đặc biệt: Chương trình khởi động thường trú trong máy. Chương trình này được cài sẵn ở một bộ nhớ chỉ đọc (ROM). Bộ nhớ ROM có những thuộc tính sau:

+ Chỉ cho phép đọc nội dung đã được viết sẵn từ trước chứa trong nó. Nội dung này do người điều hành thiết kế lập trình sẵn và viết vào nó bằng một phương pháp đặc biệt.

+ Người sử dụng về nguyên tắc là không thể hoặc rất khó thay đổi nội dung thông tin đã viết nhớ trong ROM.

+ Nội dung được viết trong ROM có tính chất cố định, không bị mất đi theo thời gian hay do mất nguồn năng lượng cung cấp cho toàn bộ hệ thống số trong đó có ROM.

Do những thuộc tính trên nên ROM được dùng khi:

+ Chứa các chương trình vận hành máy tính, chương trình hỗ trợ việc lập trình hay các chương trình giám sát việc điều hành theo các bước nghiêm ngặt cố định.

+ Các chương trình cài đặt cho máy tính chuyên dụng chỉ thực hiện một nhóm các công việc giới hạn, được lặp đi, lặp lại có quy luật

do công nghệ tự động hóa đòi hỏi (ví dụ: tự động điều khiển động cơ cho ô tô…).

+ Lưu giữ các bảng biểu quan trọng được sử dụng thường xuyên như công cụ cho việc tính toán, xử lý của máy tính và các thiết bị liên quan.

7.2.1 Cấu trúc chung của ROM

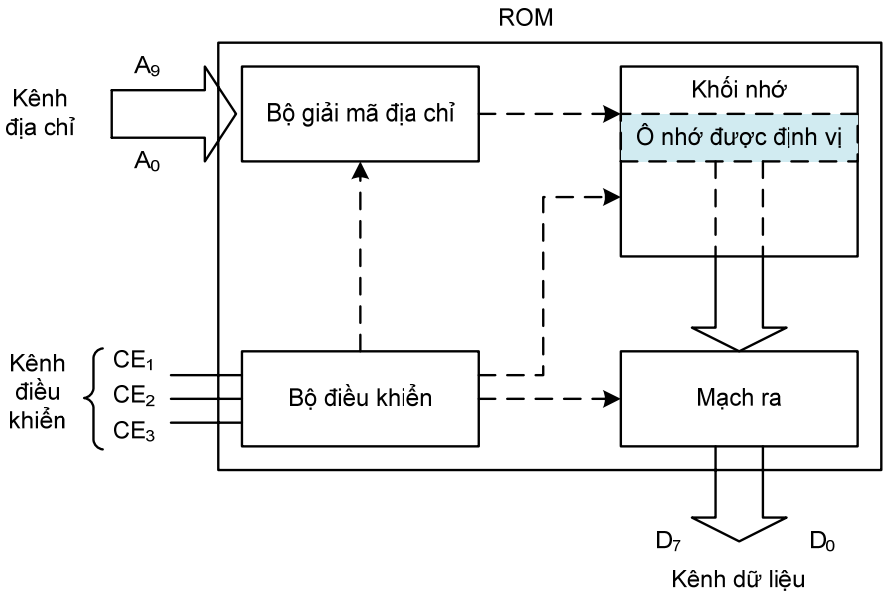
ROM bao gồm 4 khối cơ bản:

+ Bộ nhớ chứa các ô nhớ và trong các ô nhớ là các từ nhớ.

+ Mạch điều khiển tiếp nhận các tín hiệu vào từ kênh điều khiển.

+ Bộ giải mã địa chỉ dùng để định vị ô nhớ.

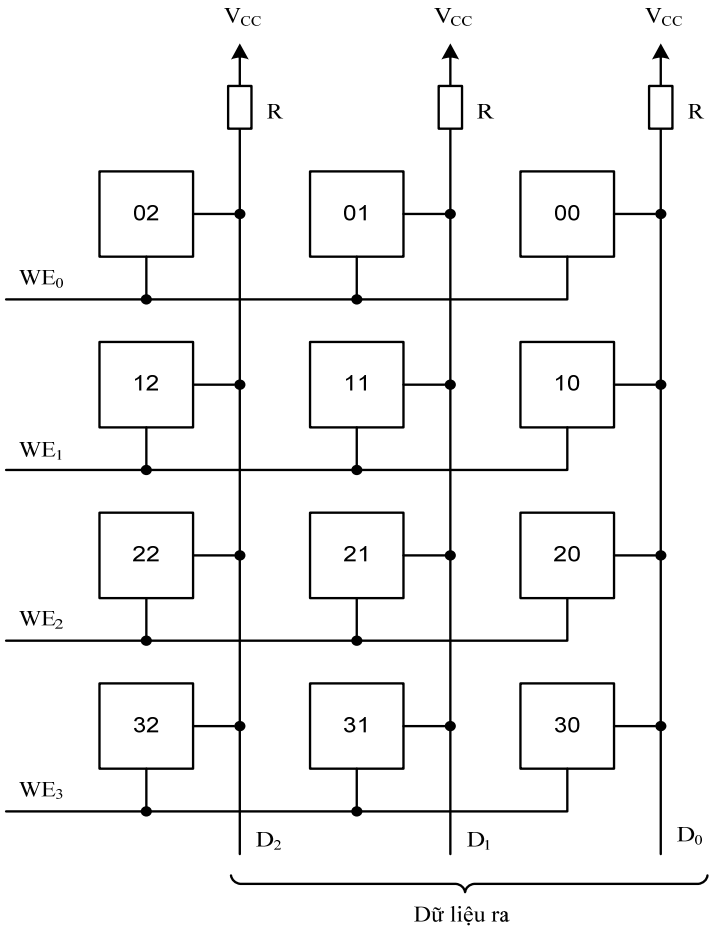
+ Mạch ra dùng để đưa nội dung ô nhớ tới các thiết bị có liên quan cần tiếp nhận nội dung này.



Hình 7.3: Cấu trúc cơ bản của bộ nhớ ROM

7.2.1.1 Khối nhớ

Mỗi ô nhớ nhị phân có chức năng lưu giữ một trong hai trạng thái 0 hoặc 1. Tập hợp một nhóm các ô nhớ xếp theo hàng tạo thành một vị trí nhớ có khả năng lưu giữ một từ nhị phân 8bit hoặc 16bit… Kiểu sắp xếp này được gọi là cấu trúc bộ nhớ kiểu mảng (array) tuyến tính. Hình 7.4 biểu diễn một cấu trúc bộ nhớ kiểu mảng tuyến tính với dung lượng 4 x 3.



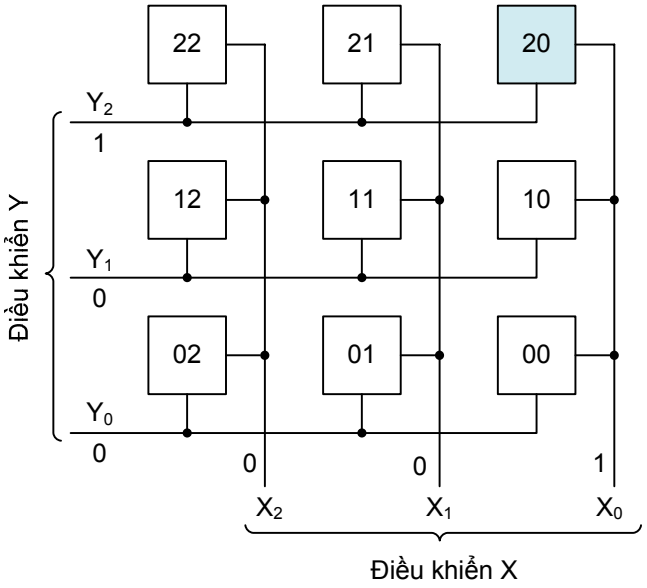
Hình 7.4: Cấu trúc bộ nhớ kiểu mảng tuyến tính

Trình tự đọc một từ nhớ như sau:

+ Kích hoạt đầu vào điều khiển WE (Word Enable) (tích cực ở mức thấp hay cao tùy theo cấu tạo của ô nhớ). Đây chính là đầu giải mã địa chỉ.

+ Các ô nhớ phải có đầu ra thuộc loại Colectơ hở hoặc đầu ra 3 trạng thái. Tất cả các ô nhớ trong một cột xác định có đầu ra nối chung với một đầu ra dữ liệu. Trong ví dụ này từ nhớ chỉ có 3bit nên có 3 đầu ra $D_2D_1D_0$ tương ứng với 3 cột, 4 hàng tương ứng 4 từ được điều khiển theo các xung WE0 WE1 WE2 và WE3. Điều này có nghĩa là mỗi một từ nhớ phải có một đầu điều khiển WE riêng. Đây chính là hạn chế của cấu trúc mảng tuyến tính vì khi số lượng địa chỉ tăng lên với các bộ nhớ dung lượng lớn thì cấu trúc bộ giải mã địa chỉ sẽ rất phức tạp.

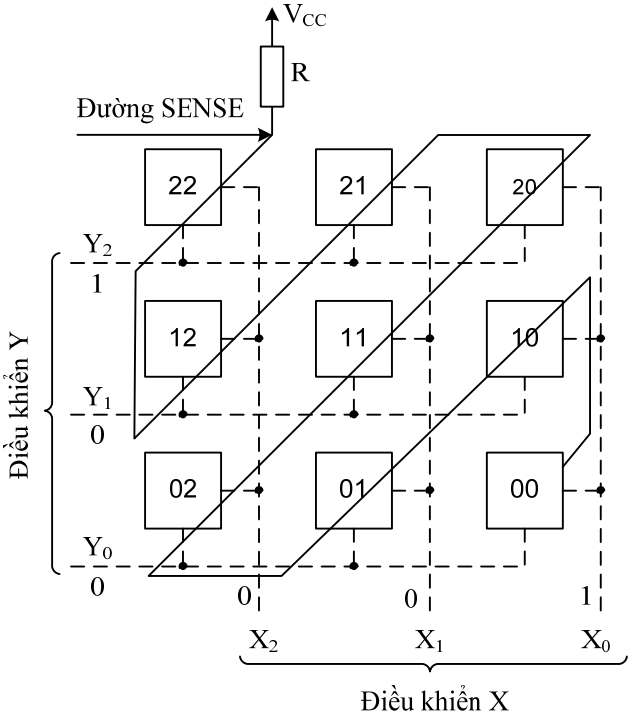
Dạng cấu trúc thứ hai của bộ nhớ là tổ chức theo kiểu ma trận, thích hợp với các bộ nhớ có dung lượng lớn (hình 7.5).



Hình 7.5a) Cấu trúc bộ nhớ kiểu ma trận hướng bit

Hình 7.5a là kết cấu mảng ma trận (3 x 3) các ô nhớ. Ví dụ này đủ tổng quát cho một ma trận *n* hàng có ký hiệu từ Y_0 đến Y_{n-1}, *m* cột

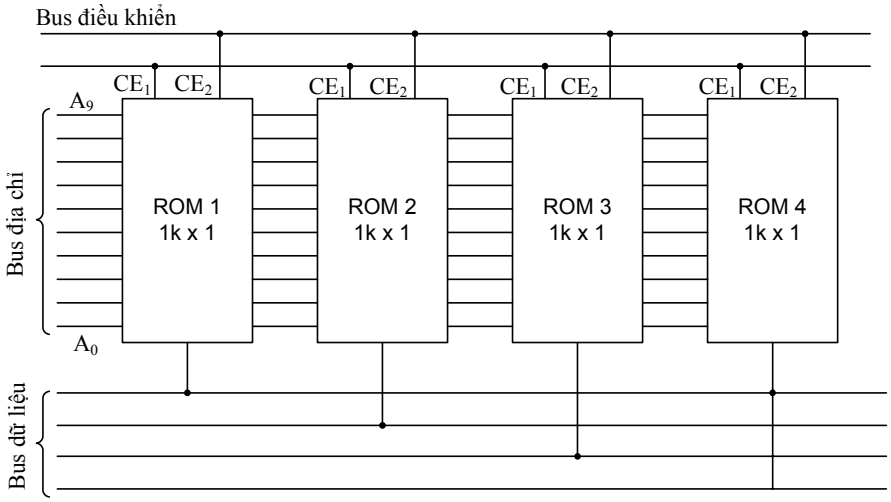
có ký hiệu từ X_0 đến X_{m-1}. Vậy bộ nhớ này có dung lượng là (*m x n*)bit nhớ. Mỗi ô nhớ của ma trận có hai đầu vào cho phép. Khi chúng đồng thời ở trạng thái tích cực thì ô nhớ mới xuất (đọc) được một bit lưu trong nó. Ví dụ để kích hoạt ô nhớ có ký hiệu là 20 thì các đầu X_0 và Y_2 phải ở trạng thái tích cực, ví dụ là tích cực cao ($X_0 = Y_2 = 1$), lúc này giá trị trên đầu cho phép phải là 2 tổ hợp: $X_2X_1X_0 = 001$ và $Y_2Y_1Y_0 = 100$. Với hai tổ hợp cho phép hàng Y_i và cột X_j này chỉ cho một ô nhớ duy nhất có địa chỉ (*i j*) được chuyển lên trạng thái tích cực.



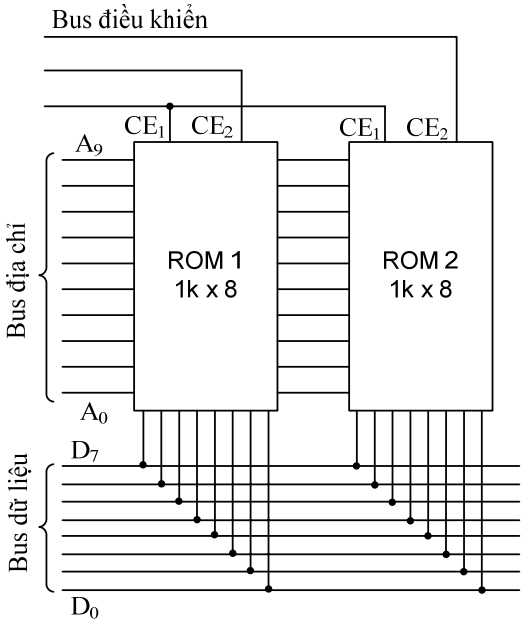
Hình 7.5b) Tín hiệu điều khiển SENSE tìm đến từng điểm nút ma trận là một ô nhớ đơn lẻ

Hình 7.5b mô tả đường vẽ cắt mặt của tất cả các ô nhớ gọi là đường SENSE nhấn mạnh tới việc chỉ một dây trên được sử dụng. Dây này dùng để dò tới ô tích cực theo hướng bit và dữ liệu chứa trong một ô nhớ được sẽ được đọc ra. Tức là ô nhớ tích cực chỉ nằm trên

dây SENSE. Mỗi một ô nhớ phải là cổng 3 trạng thái hở colectơ (trạng thái 0, trạng thái 1 và trạng thái trở kháng cao).



Hình 7.6a) Mở rộng độ dài từ cho ROM từ 1k x 1 thành 1k x 4



Hình 7.6b) Mở rộng dung lượng bộ nhớ ROM từ 1k x 8 thành 2k x 8

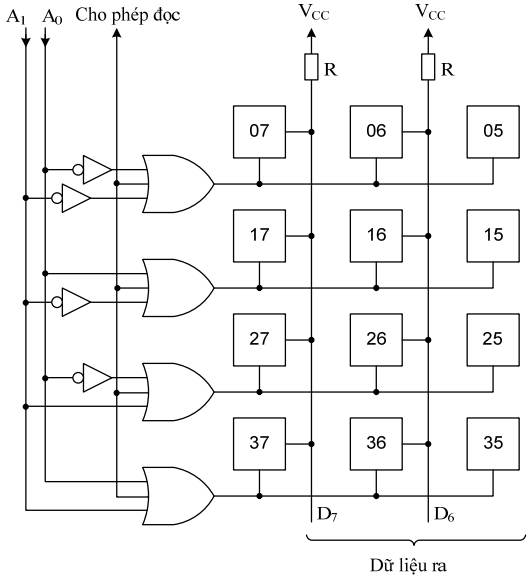
Dung lượng của bộ nhớ trên hình 7.5b là 9 x 1 (nghĩa là có 9 từ nhớ, mỗi từ dài 1bit). Muốn tạo ra một từ 4bit ta phải nối 4 mảng ma trận như vậy song song với nhau như hình 7.6a. Ở đây, mỗi mảng ma trận chứa 1024 ô nhớ, nên dung lượng tổng cộng là (1k x 4)bit. Hình 7.6b minh họa cách mở rộng dung lượng nhớ từ (1k x 8)bit thành (2k x 8)bit.

7.2.1.2 Bộ giải mã địa chỉ

Bộ giải mã địa chỉ là giao diện giữa kênh địa chỉ và khối nhớ. Nó có khả năng truyền rất nhiều địa chỉ trên một số ít đường truyền. Địa chỉ nhị phân phải được giải mã trước khi tác động tới mảng ô nhớ.

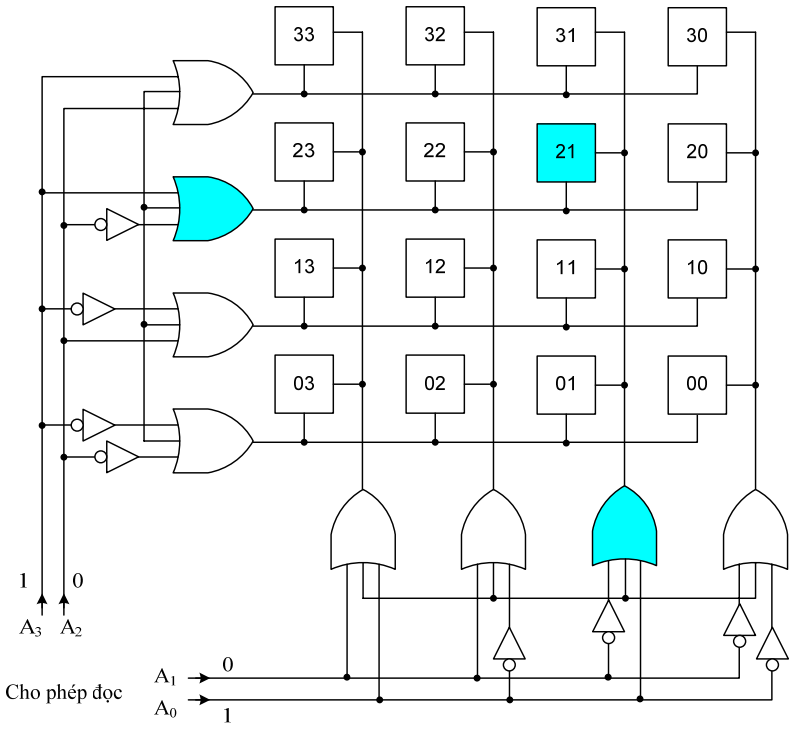
+ Trường hợp cấu trúc bộ nhớ dạng mảng tuyến tính:

Ví dụ với bộ giải mã địa chỉ 2 vào 8 ra thì A_1, A_0 là các đầu vào địa chỉ, còn 4 đầu ra của bộ giải mã chính là các đầu cho phép (WE) (hình 7.7). Tức là ứng với một tổ hợp mã A_1A_0 thì chỉ có một đầu ra WE ở trạng thái tích cực.



Hình 7.7: Bộ giải mã ROM 2 vào 8 ra cho 32 ô nhớ (4 hàng x 8 cột)

+ Trường hợp bộ nhớ có cấu trúc dạng ma trận thì việc tổ chức mạch giải mã địa chỉ theo hàng và cột của ma trận đơn giản hơn nhiều. Ví dụ với bộ giải mã địa chỉ 4 vào 16 ra : với việc chọn 2bit địa chỉ A_1A_0 cho địa chỉ cột của ma trận và 2bit còn lại A_3A_2 cho địa chỉ hàng của ma trận. Như vậy chỉ cần 8 cổng NOT đã tạo ra được một địa chỉ hàng *i* cột *j* xác định tương ứng với một ô nhớ trong ma trận (hình 7.8). Ví dụ A_3A_2 A_1A_0 = 1001, khi đó vị trí hàng 10, cột 01 (hàng 2 cột 1 trong biểu diễn thập phân của *i, j*), tức là ô 21 được chọn (chuyển lên trạng thái tích cực).

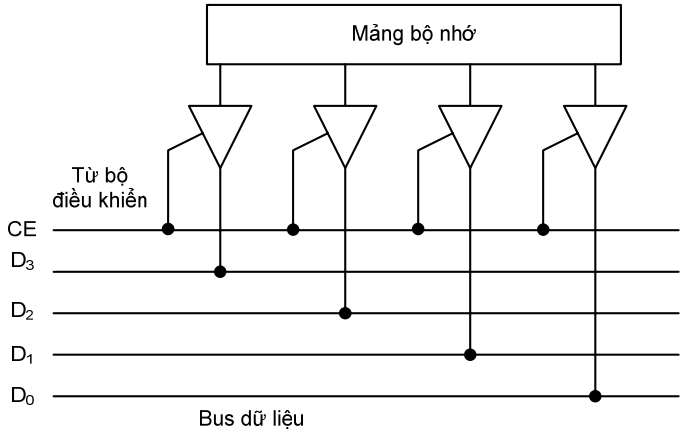


Hình 7.8: Bộ giải mã địa chỉ cho mảng ma trận ô nhớ (4 x 4t)bit

7.2.1.3 Mạch ra của bộ nhớ

Mạch ra có nhiệm vụ kết nối dữ liệu đã chọn với kênh dữ liệu vào lúc thích hợp. Trên hình 7.9, bộ đệm 3 trạng thái được sử dụng

với mục đích ngắt kết nối giữa bộ nhớ với kênh dữ liệu khi bộ nhớ ở trạng thái không tích cực (tức là không làm việc), các cổng 3 trạng thái phải có hệ số tải cao đủ để ghép kênh dữ liệu với vô số tải khác cùng nối với kênh.



Hình 7.9: Cổng 3 trạng thái dùng làm mạch đầu ra của ROM.

Tín hiệu CE từ bộ điều khiển cho phép hay không cho phép kết nối bộ nhớ với Bus dữ liệu.

7.2.1.4 Mạch điều khiển trong ROM

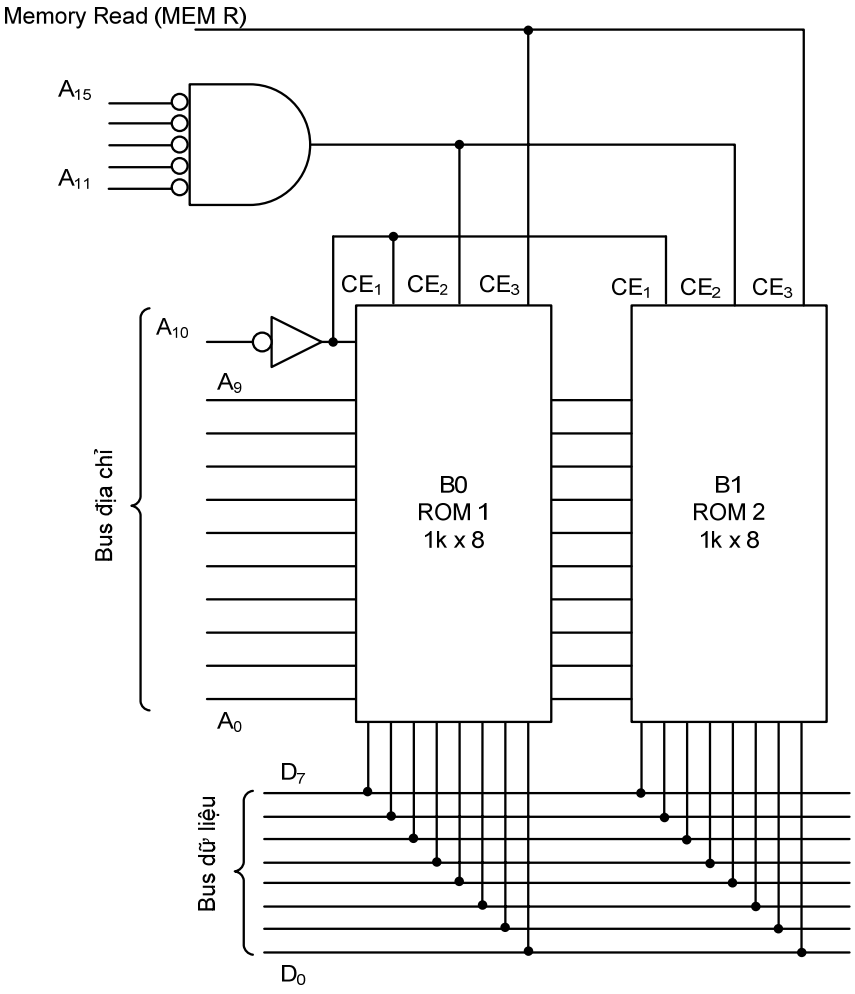
Mạch điều khiển trong ROM có chức năng khá đơn giản. Ví dụ hình 7.3 là cấu trúc bộ nhớ có 3 đầu điều khiển CE_1, CE_2 và CE_3 (Chip Enable) hay còn gọi là CS (Chip Select). Khi cả 3 tín hiệu này đều tích cực thì ROM mới được phép hoạt động (cần một cổng AND hay cổng NAND 3 đầu vào tùy tính chất từng mạch). Tín hiệu CE này có thể thực hiện các nhiệm vụ sau:

+ Chọn chip cần đọc (khi bộ nhớ có nhiều chip ghép song song).

+ Định giờ cho phép kết nối ROM với kênh dữ liệu.

+ Tham gia vào tiến trình giải mã địa chỉ để mở rộng khả năng kiểm soát và chọn địa chỉ cho chính xác.

Ví dụ hình 7.10 minh họa 3 chức năng điều khiển mà tín hiệu CE tham gia:



Hình 7.10: Dùng đầu vào điều khiển CE để chọn chip

Bộ nhớ có chứa 2 chip nhớ loại 1k. Bus địa chỉ của máy tính có 16bit ($A_0 \div A_{15}$) nên chúng dễ dàng xác định được 2k ô nhớ, các bit địa

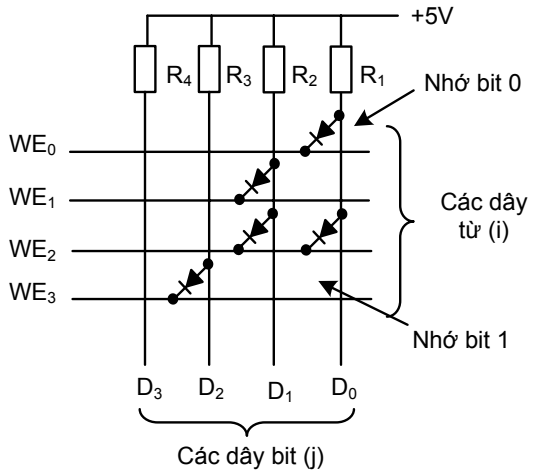
chỉ từ A_0 đến A_9 đã được nối song song và được giải mã đồng thời ở bên trong hai chip này. Tuy nhiên, chỉ có chip nào có cả ba đầu CE ở trạng thái tích cực thì mới cho phép xuất dữ liệu ra (ví dụ trong hình 7.10, bit A_{10} sẽ quyết định chip nào sẽ ở trạng thái tích cực). Chip B0 có khối địa chỉ nhị phân là 11bit, từ 0 00000 00000 đến 0 11111 11111, chip B1 sẽ có các mã địa chỉ ô nhớ từ 1 00000 00000 đến 1 11111 11111. Hệ thống địa chỉ này có được cho hai chip B0 và B1 với điều kiện các bit địa chỉ có trọng số cao hơn A_{10} (ví như $A_{11} \div A_{15}$) phải ở trạng thái không tích cực. Trong hình 7.10 cổng NOR đảm nhận nhiệm vụ này. CE_2 chỉ tích cực khi các địa chỉ thừa này bằng 0, còn nếu các địa chỉ này ở trạng thái tích cực thì CE_2 sẽ không tích cực, nghĩa là chip B0 và B1 sẽ không được chọn để tham gia quá trình xử lý.

Với 16bit địa chỉ thì có tối đa $2^6.2^{10}$ = 64k hay 64 chip 1k được gán địa chỉ. Các bit từ A_{10} đến A_{15} được tổ hợp để chọn 1 trong 64 chip 1k này lên trạng thái tích cực để xuất dữ liệu, còn các địa chỉ từ A_0 đến A_9 dùng để chọn vị trí địa chỉ một ô nhớ cụ thể trong chip đã được chọn. Đầu điều khiển CE_3 dùng để định giờ hoạt động cho ROM, khi tín hiệu MEM R xuất hiện thì dữ liệu đã được chọn được quyền xuất ra Bus dữ liệu.

7.2.2 MROM

ROM lập trình theo kiểu mặt nạ được gọi là MROM. Nó được chế tạo trên một phiến silic theo một số bước xử lý như quang khắc và khuếch tán để tạo ra những tiếp giáp bán dẫn có tính dẫn điện theo một chiều (như điốt, tranzito trường).

Người thiết kế định rõ chương trình muốn viết vào ROM, thông tin này được sử dụng để điều khiển quá trình làm mặt nạ. Hình 7.11 là một ví dụ đơn giản về sơ đồ MROM dùng điốt.



Hình 7.11: MROM dùng điốt

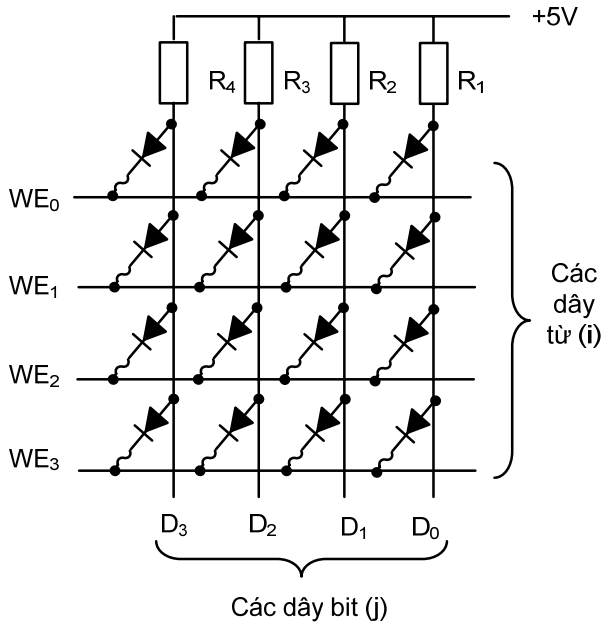
Chỗ giao nhau giữa các dây từ (hàng) và các dây bit (cột) tạo nên một phần tử nhớ (ô nhớ). Một điốt được đặt tại đó (hình 7.11) sẽ cho phép lưu trữ dữ liệu "0". Ngược lại những vị trí không có điốt thì sẽ cho phép lưu trữ dữ liệu "1" (hoặc ngược lại có điốt thì nhớ bit 1, không có thì nhớ bit 0-tùy tính chất mạch). Trong trường hợp hình 7.11, khi đọc một từ dữ liệu thứ *i* của ROM, bộ giải mã sẽ đặt dây từ đó xuống mức logic thấp, các dây còn lại ở mức cao. Do vậy chỉ những điốt nối với dây này được phân cực thuận, do đó nó sẽ dẫn làm cho điện thế đầu ra trên các dây bit tương ứng ở mức logic thấp, các dây bit còn lại sẽ giữ ở mức cao.

Cả hai công nghệ MOS và lưỡng cực được dùng để chế tạo MROM. Thời gian truy nhập của bộ nhớ lưỡng cực khoảng từ 50 - 90ns, bộ nhớ MOS lâu hơn khoảng 10 lần. Do đó ROM lưỡng cực nhanh hơn và có khả năng kích hoạt tốt hơn trong khi mạch nhớ MOS cùng dung lượng có kích thước nhỏ hơn và tiêu thụ năng lượng ít hơn. Tuy nhiên ảnh hưởng của nhiễu tĩnh điện lớn nên dễ làm mất hay sai lệch thông tin đã được viết nên cần có các biện pháp công nghệ chặt chẽ để đảm bảo an toàn thông tin được lưu trữ.

7.2.3 PROM

Để khắc phục nhược điểm của MROM là tính năng động kém do thời gian chế tạo phụ thuộc vào thời gian đặt hàng, hầu như không thay đổi được nội dung, giá thành tương đối cao nên người ta sử dụng loại ROM lập trình được – PROM.

PROM cũng gồm có các điốt như ở MROM nhưng chúng có mặt đầy đủ tại các vị trí giao nhau giữa dây từ và dây bit (hình 7.12). Mỗi điốt được nối với một cầu chì. Bình thường khi chưa lập trình, các cầu chì còn nguyên vẹn, nội dung của PROM sẽ toàn là 0. Khi định vị đến một bit bằng cách đặt một xung điện ở đầu ra tương ứng, cầu chì sẽ bị đứt và bit này sẽ bằng 1. Bằng cách đó ta có thể lập trình toàn bộ các bit trong PROM. Như vậy, việc lập trình đó có thể được thực hiện bởi người sử dụng chỉ một lần duy nhất, không thể sửa đổi được.



Hình 7.12: PROM dùng điốt

7.3 BỘ NHỚ BÁN CỐ ĐỊNH

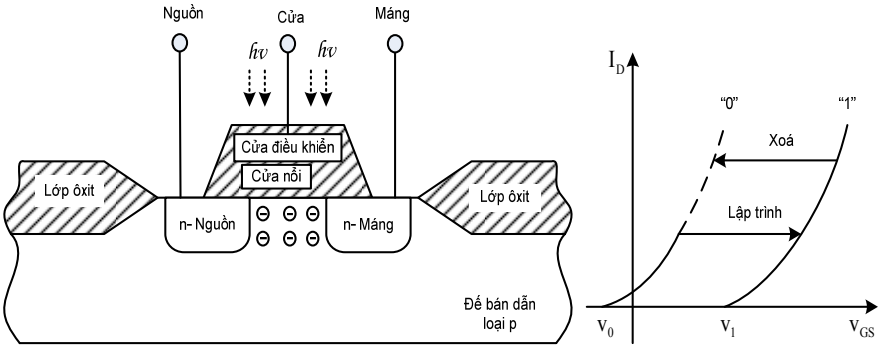
7.3.1 EPROM (Erasable PROM)

Dữ liệu vào có thể được viết vào bằng xung điện nhưng được lưu giữ theo kiểu không bay hơi. Đó là loại ROM có thể lập trình được và xóa được. Hình 7.13 chỉ ra cấu trúc của một tranzito dùng để làm một ô nhớ gọi là FAMOST (Floating gate avalanche injection MOS tranzito).

Trong ô nhớ dùng tranzito này, cực cửa được nối với đường từ, cực máng được nối với đường bit và cực nguồn được nối với nguồn chuẩn được coi là nguồn cho mức logic 1. Khác với tranzito MOS bình thường, tranzito loại này còn có thêm một cửa gọi là *cửa nổi* (floating gate); đó là một vùng vật liệu được thêm vào vào giữa lớp cách điện cao như ở hình 7.13. Nếu cửa nổi không có điện tích thì nó không ảnh hưởng gì đến cực cửa điều khiển và tranzito hoạt động như bình thường. Tức là khi dây từ được kích hoạt (cực cửa có điện thế dương) thì tranzito dẫn, cực máng và nguồn được nối với nhau qua kênh dẫn và dây bit có mức logic 1. Nếu cửa nổi có các điện tử trong đó với điện tích âm thì chúng sẽ ngăn trường điều khiển của cực cửa và dù dây từ được kích hoạt thì cũng không thể phát ra trường đủ mạnh với cực cửa điều khiển để làm thông tranzito. Lúc này đường bit không được nối với nguồn chuẩn và ô nhớ coi như được giữ giá trị 0.

Việc nạp các điện tử vào vùng cửa nổi, tức là tạo ra các ô nhớ mang giá trị 0 được thực hiện bởi xung điện có độ dài cỡ 50ms và độ lớn + 20V đặt giữa cực cửa và cực máng. Lúc đó những điện tích mang năng lượng lớn sẽ đi qua lớp cách điện giữa đế và cửa nổi. Chúng tích tụ trong vùng cửa nổi và được giữ ở đây sau khi xung lập trình tắt. Đó là do cửa nổi được cách điện cao với xung quanh và các điện tử không còn đủ năng lượng sau khi lạnh đi, để có thể vượt ra ngoài lớp cách điện đó nữa. Chúng sẽ được giữ ở đây trong một thời gian rất dài (ít nhất là 10 năm).

Để xoá các thông tin, tức là làm mất các điện tích điện tử trong vùng cửa nổi, phải chiếu ánh sáng tử ngoại UV vào chíp nhớ. Lúc này, những điện tử hấp thụ được năng lượng và sẽ nhảy lên các mức năng lượng cao và rời khỏi cửa nổi giống như cách mà chúng đã thâm nhập vào. Trong chip EPROM có một cửa sổ làm bằng thuỷ tinh thạch anh chỉ để cho ánh sáng tử ngoại đi qua khi cần xoá dữ liệu trong bộ nhớ.



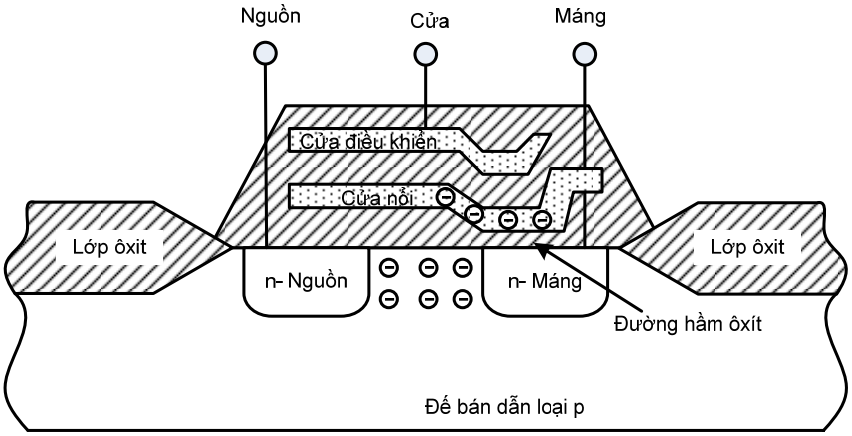
Hình 7.13: Cấu trúc của tranzito dùng làm ô nhớ trong EPROM

7.3.2 EEPROM (Electrically Erasable PROM)

Cửa sổ thạch anh có giá thành khá đắt và không tiện lợi nên những năm gần đây xuất hiện các chip PROM có thể xoá dữ liệu bằng phương pháp điện. Cấu trúc của ô nhớ giống như hình 7.14.

Việc nạp các điện tử cho cửa nổi được thực hiện như cách ở EPROM. Bằng một xung điện tương đối dài, các điện tích mang năng lượng cao được phát ra trong đế sẽ thấm qua lớp cửa ôxit và tích tụ trong cửa nổi. Để xoá EEPROM, một lớp kênh màng mỏng ôxit giữa vùng cửa nổi trải xuống dưới đế và cực máng giữ vai trò quan trọng. Các lớp cách điện không thể là lý tưởng được, các điện tích có thể thấm qua lớp phân cách với một xác suất thấp. Xác suất này tăng lên khi bề dày của lớp giảm đi và điện thế giữa hai điện cực ở hai mặt lớp cách điện tăng lên. Muốn phóng các điện tích trong vùng cửa nổi một điện thế (-20V) được đặt vào cực cửa điều khiển và cực máng. Lúc

này các điện tử âm trong cửa nổi được chảy về cực máng qua kênh màng mỏng ôxit và dữ liệu lưu giữ được xoá đi. Điều lưu ý là phải làm sao cho dòng điện tích này chảy không quá lâu vì nếu không vùng cửa nổi này lại trở nên tích điện dương làm cho hoạt động của tranzito không được trạng thái bình thường (mức nhớ 1).

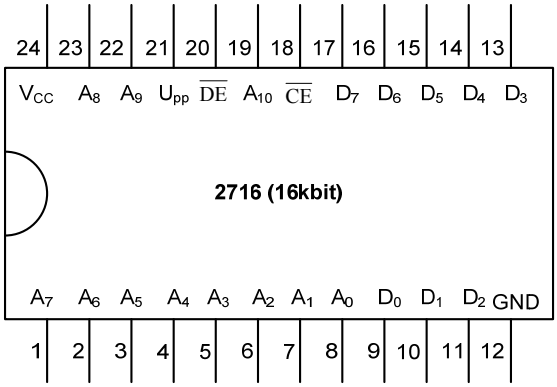


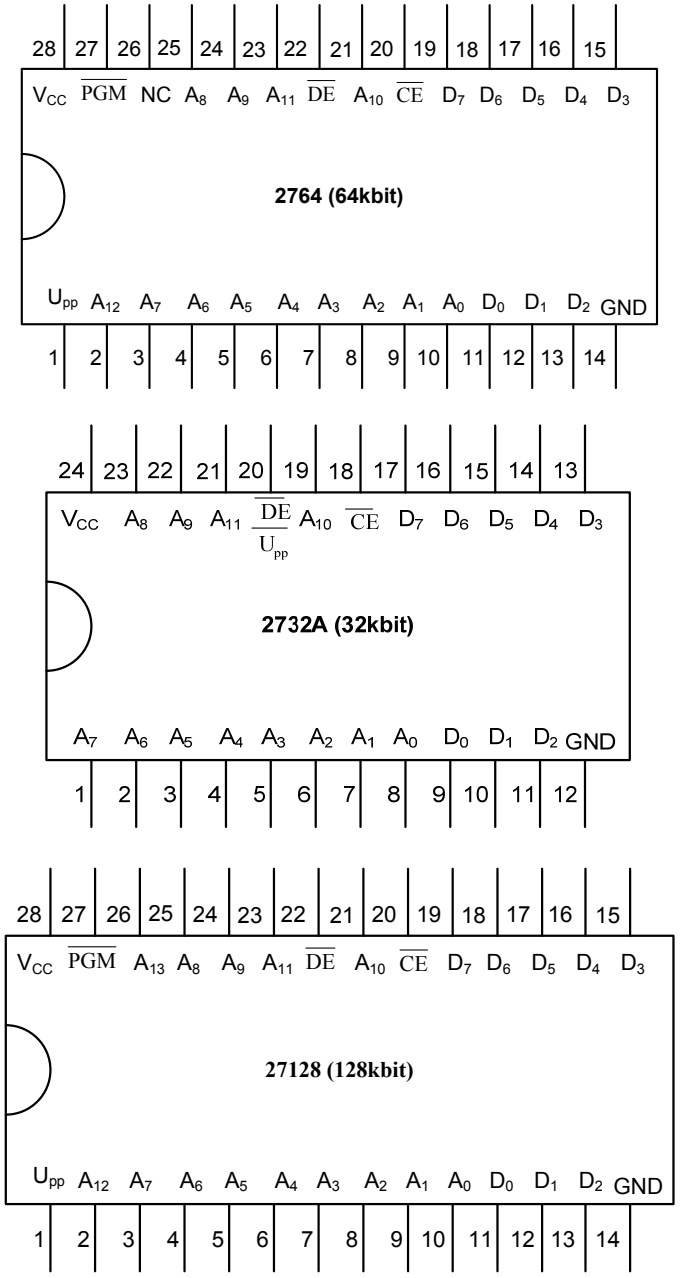
Hình 7.14: Cấu trúc của một ô nhớ trong EEPROM

Các chip ROM hiện nay có thời gian truy nhập từ 120ns đến 150ns dài hơn nhiều thời gian đó trong các chip nhớ RAM.

7.4 MỘT SỐ IC THƯỜNG GẶP

7.4.1 EPROM xóa bằng tia cực tím





Hình 7.15: Sơ đồ chân của EPROM thường gặp

Trong các IC trên có:

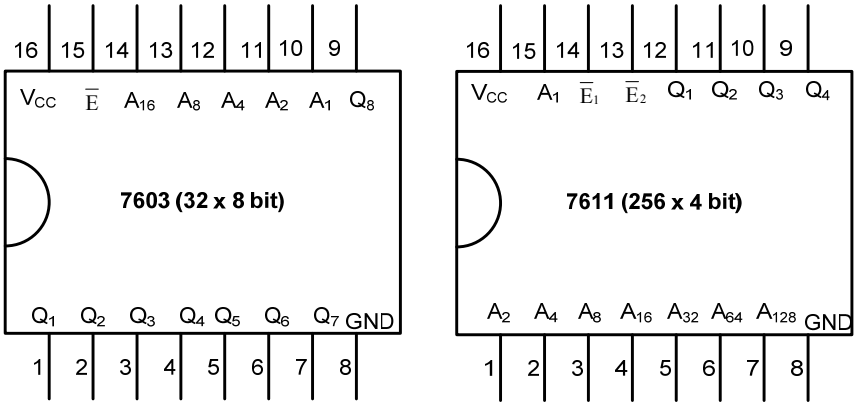
+ Chân U_{pp} dùng để đặt điện áp cao khi viết (nạp dữ liệu).

+ Tương thích với TTL, thường sử dụng trong CPU.

+ 27xx(x): 2 hay 3 số tiếp theo số 27 chỉ dung lượng nhớ. Ví dụ 2764 nghĩa là bộ nhớ có dung lượng 64kbit.

7.4.2 PROM

Hai loại IC PROM điển hình là 7603 và 7611.b



Hình 7.16: Sơ đồ chân của IC PROM

IC 7603 có dung lượng 32 x 8bit, tương thích với TTL. IC này có 5 đầu vào địa chỉ từ A_1 đến A_{16}, 8 đầu ra dữ liệu từ Q_1 đến Q_8. Khi $\overline{E} = 0$ thì IC cho phép xuất dữ liệu, khi $\overline{E} = 1$ thì các đầu ra Q bị thả nổi, nguồn +5V nạp dữ liệu bằng cách đốt cháy cầu chì loại đầu ra là cổng 3 trạng thái.

IC7611 có dung lượng 256 x 4bit, tương thích với TTL. IC này có 8 đầu vào địa chỉ từ A_1 đến A_{128}, 4 đầu ra dữ liệu từ Q_1 đến Q_4 (loại cổng 3 trạng thái). Khi $\overline{E_1} = \overline{E_2} = 0$ thì IC cho phép xuất dữ liệu, khi $\overline{E_1} = 0$ và $\overline{E_2} = 1$ thì các đầu ra Q bị thả nổi, nguồn +5V nạp dữ liệu bằng cách đốt cháy cầu chì.

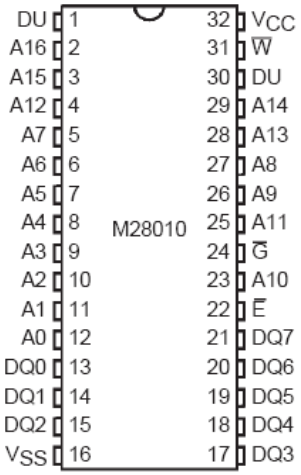
7.4.3 EEPROM

IC M28010 là bộ nhớ có dung lượng 128k x 8bit, có công suất thấp, nạp song song. Nguồn cấp cho IC này là 5V, 3V, hoặc 2V phụ thuộc vào ứng dụng được chọn. Đối với loại M28010 có nguồn nằm trong dải 4,5V ÷5,5V, đối với loại M28010-W có nguồn nằm trong dải 2,7V ÷3,6V, đối với loại M28010-R có nguồn nằm trong dải 1,8V ÷ 2,4V.

Bảng 7.1 là bảng mô tả chức năng của chân IC.

Bảng 7.1: Bảng mô tả chức năng chân của IC

Chân	Chức năng
A0 ÷ A16	Đầu vào địa chỉ
DQ0 ÷ DQ7	Đầu vào/ra dữ liệu
$\overline{W}$	Cho phép viết
$\overline{E}$	Chọn chip (khi $\overline{E}$ = 0 cho phép đọc/viết)
$\overline{G}$	Cho phép ra (điều khiển bộ đệm đầu ra, được sử dụng để thiết lập chế độ đọc)
V_{CC}	Nguồn cung cấp
V_{SS}	Đất

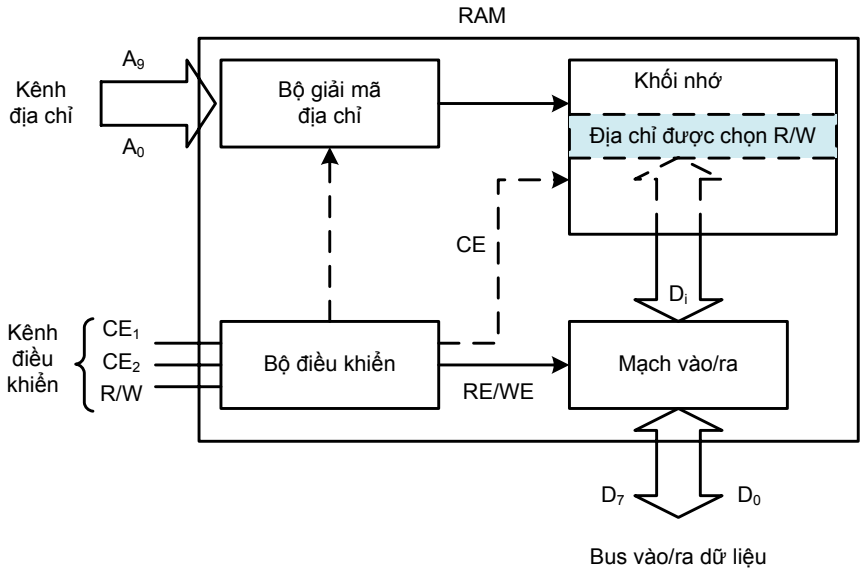


Hình 7.17: Sơ đồ chân của IC

7.5 RAM

RAM có khả năng cho phép viết lưu trữ dữ liệu thông tin tạm thời trong một thời gian, sau đó lại đọc thông tin đó để tiếp tục xử lý khi cần thiết nên nó có tên là bộ nhớ đọc/viết. Một đặc tính quan trọng khác của RAM là các dữ liệu trong RAM chỉ có tính chất tạm thời, dễ bị xóa khi mất nguồn năng lượng cấp cho nó nên cần phải phòng bị cho nó một nguồn phụ nhằm tránh hiện tượng mất nguồn đột ngột, trong khi quá trình xử lý chưa kết thúc, thông tin trong RAM còn có ích.

7.5.1 Cấu trúc khối của RAM



Hình 7.18: Cấu trúc 4 khối của một RAM có 8bit dữ liệu và 8bit địa chỉ

Cũng như ROM, RAM cũng có 4 phần chính như mô tả trên hình 7.18. Điểm khác biệt là:

+ Mạch điều khiển của RAM phải có thêm đầu vào R/W điều khiển hai quá trình cơ bản trong thao tác của RAM: viết dữ liệu thông tin vào nó và quá trình xuất (đọc) thông tin đã viết.

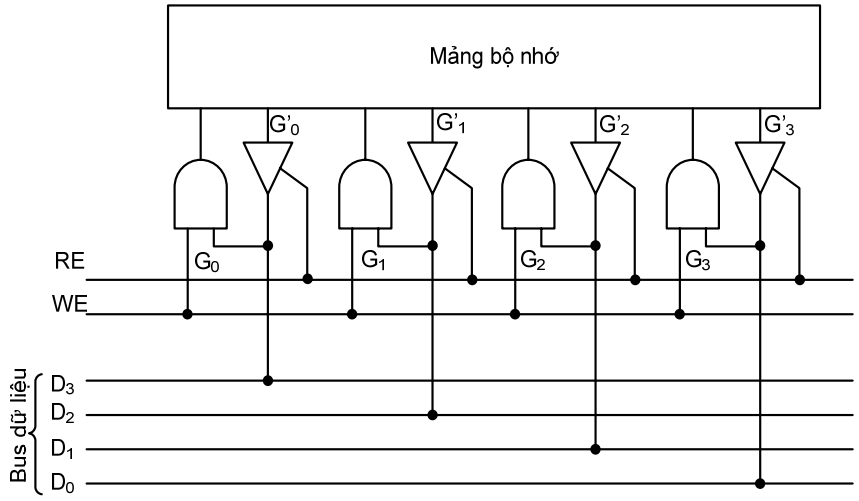
+ Mạch đầu ra có khả năng kiểm soát hai chiều trước khi cho phép giao tiếp với kênh dữ liệu. Quá trình này tuân theo nguyên tắc: (đồng bộ với việc điều khiển R/W) khi bộ nhớ đang đọc thì không được viết và ngược lại; trạng thái thứ ba có thể chờ quyết định.

7.5.1.1 Mạch vào/ra của RAM

Mạch vào ra của RAM phức tạp hơn ROM. Hình 7.19 mô tả một dạng mạch vào/ra với bốn dữ liệu từ D_0 đến D_3.

Các dữ liệu trên kênh dữ liệu G_0, G_1, G_2 và G_3 viết vào bộ nhớ khi tín hiệu cho phép viết WE = 1. Khi có tín hiệu điều khiển cho phép đọc RE = 1 thì các cổng G'_0, G'_1, G'_2 và G'_3 được điều khiển mở (đây là cổng 3 trạng thái) để xuất các dữ liệu ra kênh.

Một điều cần chú ý là các thao tác trên phải đồng bộ với việc chọn xong địa chỉ viết hay đọc trong bộ nhớ.



Hình 7.19: Mạch vào/ra 4bit dữ liệu của RAM

Cổng 3 trạng thái còn có khả năng đưa mức ra của G'_0, G'_1, G'_2 và G'_3 lên trạng thái trở kháng cao, do đó, bộ nhớ được đặt trong tình

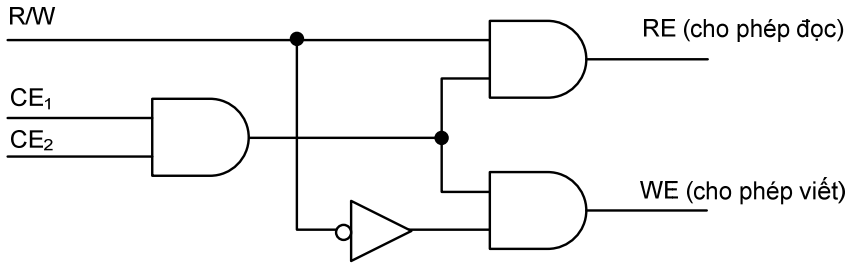
trạng chờ: không cho dữ liệu ra và cũng không thực hiện viết dữ liệu vào. Lúc này toàn bộ dữ liệu được cách ly ra khỏi kênh dữ liệu cho an toàn.

7.5.1.2 Mạch điều khiển của RAM

Hình 7.20 mô tả một dạng mạch đơn giản đưa tới RAM 3 chế độ hoạt động khác nhau từ 3 đầu vào điều khiển CE_1, CE_2 và R/W. Ở mỗi thời điểm chỉ có một trong ba chế độ được thiết lập:

+ Khi ở chế độ đọc, xung R/W ở mức logic 1. Đồng thời các tín hiệu cho phép chọn CE_1, CE_2 được kích hoạt ở mức 1 nên lúc này RE = 1, tức là chế độ đọc được thiết lập. Khi đó tín hiệu $\overline{R / W} = 0$ nên tín hiệu cho phép viết WE bằng 0 (cấm viết).

+ Khi ở chế độ viết, xung R/W ở mức logic 0, $\overline{R / W} = 1$, đồng thời các tín hiệu cho phép chọn CE_1, CE_2 được kích hoạt ở mức 1 nên lúc này WE = 1, tức là nó ở chế độ viết. Khi đó tín hiệu R/W = 0 nên tín hiệu cho phép đọc RE bằng 0 (cấm đọc).



Hình 7.20: Mạch điều khiển RAM có 3 chế độ: đọc/viết/chờ

+ Tín hiệu tích cực đồng thời $CE_1 = CE_2 = 1$ ở cả hai chế độ đọc và viết phải được chuyển cùng lúc tới mảng ô nhớ nhằm thông báo việc xuất (khi đọc) hay việc nhập (khi viết) dữ liệu tới địa chỉ ô nhớ đã được mạch giải mã chọn.

+ Khi tín hiệu $CE_1.CE_2 = 0$ (có ít nhất một tín hiệu CE ở trạng thái không tích cực) thì mạch điều khiển ở hình 7.20 sẽ chuyển bộ nhớ

sang chế độ chờ (Standby) bất chấp tín hiệu R/W có tích cực hay không, lúc này RE = 0 và WE = 0.

7.5.2 Cấu tạo của DRAM

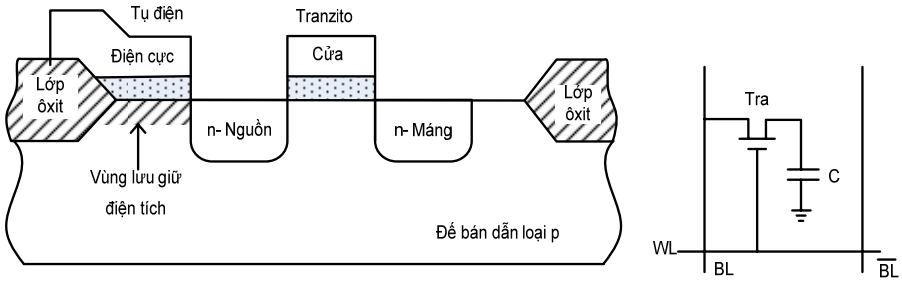
7.5.2.1 Cấu tạo của một ô nhớ

Các ô nhớ được sắp xếp theo hàng và cột trong một ma trận nhớ. Địa chỉ ô nhớ được chia thành hai phần: địa chỉ hàng và cột. Hai địa chỉ này được đọc vào bộ đệm một cách lần lượt. Xử lý kiểu này được gọi là hợp kênh, lý do là để giảm kích thước bộ giải mã, tức là giảm kích thước và giá thành vi mạch. Quá trình dồn kênh địa chỉ này được điều khiển bởi các tín hiệu RAS (Row Access Strobe) và CAS (Column Access Strobe).

Nếu $\overline{RAS}$ ở mức tích cực thấp thì DRAM nhận được địa chỉ đặt vào nó và sử dụng như địa chỉ hàng.

Nếu $\overline{CAS}$ ở mức tích cực thấp thì DRAM nhận được địa chỉ đặt vào nó và sử dụng như địa chỉ cột.

Một ô nhớ của DRAM gồm có một tranzito trường MOS có trở đầu vào rất lớn và một tụ điện C là linh kiện lưu trữ một bit thông tin tương ứng với hai trạng thái có hoặc không có điện tích trên tụ.



Hình 7.21: Cấu tạo một ô nhớ của DRAM

Tranzito hoạt động như một công tắc, cho phép nạp hay phóng điện tích của tụ khi thực hiện phép đọc hay viết. Cực cửa (Gate) của

tranzito được nối với dây hàng (còn gọi là dây từ-WL-Word Line) và cực máng (Drain) được nối với dây cột (còn được gọi là dây bit BL hoặc $\overline{BL}$ - Bit Line), cực nguồn (Source) được nối với tụ điện. Điện áp nạp trên tụ tương đối nhỏ, vì thế cần sử dụng khuếch đại nhạy trong mạch nhớ.

7.5.2.2 Nguyên lý hoạt động của tế bào DRAM

Như mục trên ta biết tế bào DRAM có cấu tạo hết sức đơn giản. Phần tử lưu giữ một bit dữ liệu là điện dung C, thường được tạo thành bằng điện dung tiếp giáp của MOSFET, có điện dung rất nhỏ, khoảng một vài pF. Do đó, thể tích của một tế bào DRAM bé hơn rất nhiều so với tế bào RAM tĩnh (SRAM). Nhờ thế, bộ nhớ DRAM có dung lượng nhớ rất lớn và giá thành thấp. Tuy nhiên, nhược điểm cơ bản của loại này là tụ điện không thể giữ được năng lượng đã nạp lâu dài. Trong DRAM luôn xảy ra sự tự phóng điện (dòng rò). Hiện tượng này làm điện áp trên tụ giảm dần theo thời gian được gọi là tự bay hơi. Vì vậy cứ sau 2 ms, nếu không được nạp bồi thì dữ liệu sẽ mất đi. Để duy trì dữ liệu cần phải định kỳ thực hiện nạp bồi. Quá trình này gọi là làm tươi (refresh) DRAM.

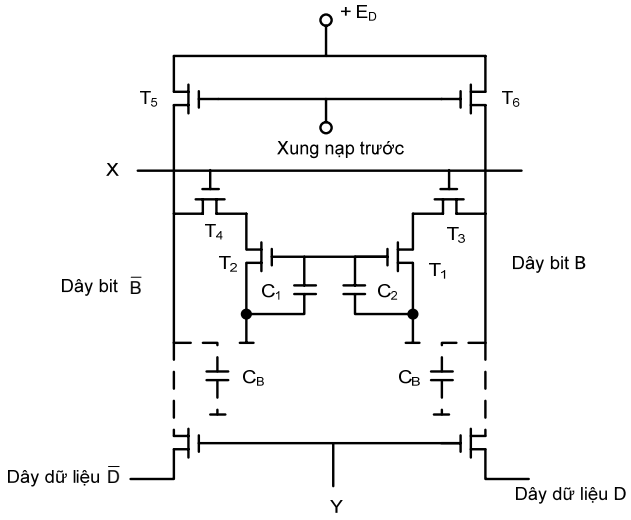
Hình 7.22 trình bày sơ đồ logic của một phần tử nhớ RAM động.

Trong mạch T_1 và T_2 ghép chéo nhau. Dữ liệu (điện tích) lưu giữ trên tụ C_1 và C_2. Điện áp trên C_1 và C_2 điều khiển T_1, T_2 thông hoặc ngắt. Khi C_1 nạp điện tích (điện áp trên C_1 lớn hơn điện áp cắt của T_1), C_2 không nạp điện tích (điện áp trên tụ C_2 nhỏ hơn điện áp cắt của T_2) thì T_1 thông T_2 ngắt, tương ứng với trạng thái 0 của phần tử nhớ. Khi C_2 nạp điện tích C_1 không nạp điện tích thì T_1 ngắt T_2 thông, tương ứng trạng thái 1 của phần tử nhớ.

Tranzito T_3, T_4 làm nhiệm vụ điều khiển nối thông phần tử nhớ và dây bit.

T_5, T_6 là mạch nạp trước của dây bit. Dùng chung cho tất cả các phần tử nhớ cùng cột trong ma trận nhớ.

Khi bắt đầu truy nhập, truy xuất bộ nhớ trên cực cửa của T_5 và T_6 có xung nạp trước nên T_5, T_6 nối thông, do đó các dây bit B và $\overline{B}$ có mức điện áp cao vì nối với nguồn E_D.



Hình 7.22: Phần tử nhớ động MOS

Sau khi kết thúc xung nạp trước T_5 và T_6 ngắt, dây bit cách ly khỏi nguồn E_D. Nhưng do tác dụng của điện dung phân bố C_B, $C_{\overline{B}}$. Khi đó mức điện áp cao của dây bit có thể duy trì thêm một thời gian nữa.

Trong thời gian này, giả sử tiến hành đọc dữ liệu, dây X có mức cao T_3 và T_4 thông, giả sử phần tử nhớ có trạng thái D (T_1 thông T_2 ngắt) lúc này C_B phóng điện qua T_1 và T_3, do đó dây bit B biến thành mức thấp.

Do T_2 ngắt dây bit $\overline{B}$ vẫn ở mức cao. Vì vậy dữ liệu nhớ trong phần tử được đọc ra dây bit B và $\overline{B}$. Nếu lúc này, dây Y cũng ở mức cao thì dữ liệu của ô nhớ sẽ đưa đến đầu ra của RAM qua dây dữ liệu D và $\overline{D}$.

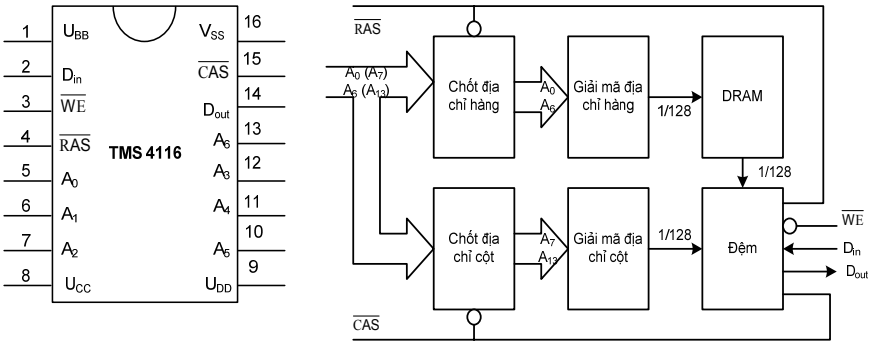
Tác dụng của mạch nạp trước của dây bit: Trong thời gian T_3, T_4 thông, nếu dây bit không được nạp điện trước thì mức cao có được của dây $\overline{B}$ chỉ do tụ C_1 phóng qua T_4 nạp vào $C_{\overline{B}}$. Do vậy, điện tích trên C_1 bị suy giảm, $C_{\overline{B}}$ có giá trị thậm chí lớn hơn C_1 (vì có nhiều phần tử nối vào dây bit). Vì vậy, G_1 (cực cửa của T_1) không giữ nguyên được mức cao sau 1 lần đọc tức là dữ liệu bị mất. Nhờ có mạch điện nạp trước, điện thế trên dây bit $\overline{B}$ còn cao hơn điện thế G_1 một chút. Vì vậy khi đọc dữ liệu, điện tích trên C_1 không bị suy giảm mà còn được làm tươi nhờ sự nạp điện thêm cho C_1 qua T_4.

Khi tiến hành viết, đầu vào dữ liệu của RAM sẽ làm thay đổi trạng thái phần tử nhớ, thông qua dây dữ liệu và dây bit, tức là đưa dữ liệu vào lưu trữ trong phần tử nhớ.

7.5.2.3. Một số IC DRAM

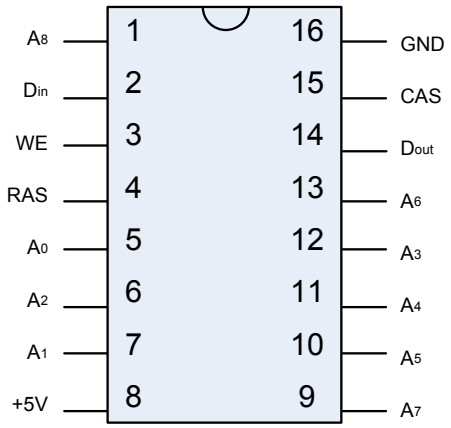
Một số loại chip DRAM thường gặp là: TMS 4116: có dung lượng 16k x 1bit; IC 41256 có dung lượng 256k x 1bit. Thời gian truy nhập thông tin khoảng 150ns, công suất tiêu thụ khoảng 280mW khi làm việc (khi chờ = 28mW).

Hình 7.23 là sơ đồ chân và sơ đồ khối của IC TMS 4116.



Hình 7.23: Sơ đồ chân và sơ đồ khối của IC TMS 4116

IC TMS 4116 là loại MOS có dung lượng 16k x 1bit có 2^{14} = 16384 ô nhớ, số đường địa chỉ yêu cầu là 14. Thực ra theo phương pháp chọn kênh địa chỉ chỉ cần dùng 7 đường chứa thông tin về hàng sau đó lại chứa thông tin về cột. Muốn vậy người ta sử dụng hai tín hiệu $\overline{RAS}$ và $\overline{CAS}$ tương ứng: Khi $\overline{RAS}$ = 0 thì thông tin trên các đường địa chỉ sẽ được mở qua mạch chốt địa chỉ hàng và khi $\overline{CAS}$ = 0 thì mở mạch địa chỉ cột. Cấm trạng thái $\overline{RAS}$ = $\overline{CAS}$ = 0 vì tín hiệu này sẽ làm rối loạn việc xác định địa chỉ ô nhớ. Khi đường $\overline{WE}$ = 0 thì cho phép viết để thông tin D_{in} được nhập vào địa chỉ đã chọn. Khi đọc thì đường $\overline{WE}$ = 1: Từ địa chỉ của ô nhớ được chọn thì thông tin sẽ được xuất ra đường D_{out}.



Hình 7.24: Sơ đồ chân của IC DRAM 41256

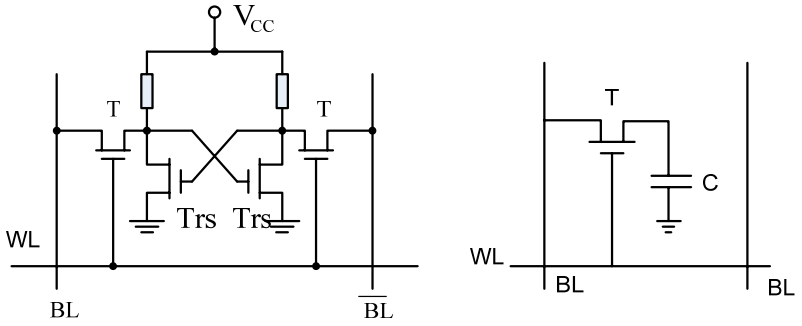
Hình 7.24 là vỏ của IC 41256 dung lượng 256k x 1bit. Mạch cần 18bit địa chỉ để mã hoá cho các địa chỉ hàng và cột; nhưng trên vỏ chỉ có 9 đường địa chỉ từ A_0 đến A_8.

Hai chân RAS, CAS hoạt động ở mức cao, dùng để điều khiển 9bit địa chỉ trên chip tới bộ giải mã địa chỉ hàng hay cột.

7.5.3 SRAM

7.5.3.1 Cấu tạo chung của SRAM

Một ô nhớ của SRAM giữ thông tin bởi trạng thái của mạch trigơ. Thuật ngữ "tĩnh" chỉ ra rằng khi nguồn nuôi chưa bị cắt thì thông tin của ô nhớ vẫn được giữ nguyên. Khác với ô nhớ DRAM, ở đây ô nhớ trigơ cung cấp một tín hiệu số mạnh hơn nhiều vì đã có các tranzito trong các ô nhớ, chúng có khả năng khuếch đại tín hiệu và do đó có thể cấp trực tiếp cho các đường bit. Trong DRAM, sự khuếch đại tín hiệu trong các bộ khuếch đại cần nhiều thời gian và do đó thời gian truy nhập dài hơn. Khi định địa chỉ trong các trigơ ở SRAM, các tranzito bổ sung cho các trigơ, các bộ giải mã địa chỉ… cũng được đòi hỏi như ở DRAM.



Hình 7.25: Cấu tạo một ô nhớ của SRAM và DRAM

Như trong DRAM, cực cửa của tranzito trong ô nhớ của SRAM cũng được nối với đường từ và cực máng nối với cặp đường bit. Nếu dữ liệu được đọc từ ô nhớ, khi đó bộ giải mã hàng kích hoạt đường dây từ WL tương ứng. Hai tranzito T dẫn và nối trigơ nhớ với cặp dây bit. Như vậy hai đầu ra Q và $\overline{Q}$ được nối với các đường bit và các tín hiệu được truyền tới bộ khuếch đại ở cuối đường dây này. Vì điện thế chênh lệch lớn nên xử lý khuếch đại như vậy sẽ nhanh hơn trong DRAM (cỡ 10ns hoặc ngắn hơn), do đó chip SRAM cần địa chỉ cột sớm hơn nếu thời gian truy nhập không được giảm. Như vậy SRAM

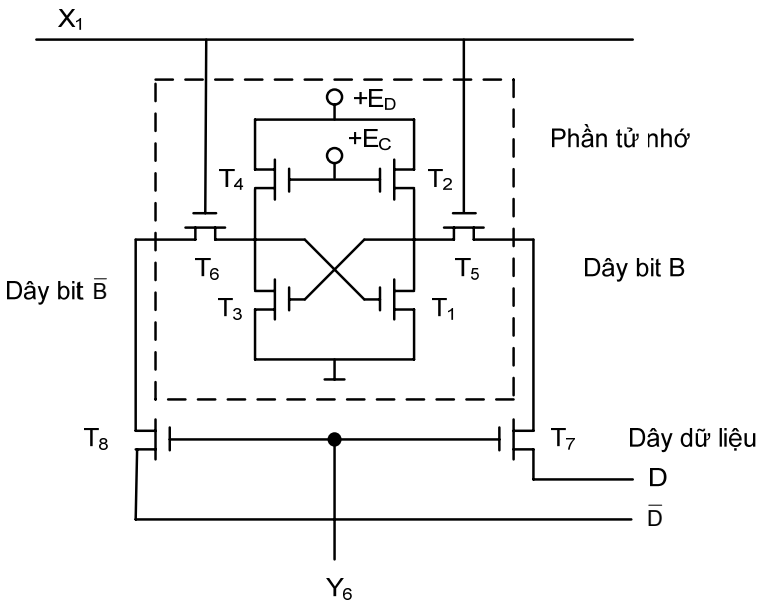
không cần thực hiện phân kênh các địa chỉ hàng và cột. Sau khi dữ liệu ổn định, bộ giải mã cột chọn cột phù hợp và cho ra tín hiệu dữ liệu tới bộ đệm dữ liệu ra và tới mạch ra.

Viết dữ liệu được thực hiện theo cách ngược lại. Qua bộ đệm vào và bộ giải mã cột, dữ liệu viết được đặt vào bộ khuếch đại phù hợp. Cùng lúc đó bộ giải mã hàng kích hoạt đường dây từ và làm tranzito T dẫn. Trigơ đưa dữ liệu được lưu trữ vào cặp dây bit. Tuy vậy, bộ khuếch đại nhạy hơn các tranzito nên nó sẽ cấp cho các đường bit một tín hiệu phù hợp với dữ liệu viết. Do đó, trigơ sẽ chuyển trạng thái phù hợp với dữ liệu mới hoặc giữ giá trị đã được lưu trữ phụ thuộc vào việc dữ liệu viết trùng với dữ liệu đã lưu trữ hay không.

7.5.3.2 Nguyên lý hoạt động

7.5.3.2.1 Phần tử nhớ RAM tĩnh dùng NMOS

Hình 7.26 trình bày sơ đồ phần tử nhớ RAM tĩnh.



Hình 7.26: Phần tử nhớ RAM tĩnh dùng NMOS

Trong mạch T_1 và T_2 làm thành bộ đảo pha, T_3 và T_4 cũng làm thành 1 bộ đảo pha khác. Hai bộ đảo pha này có đầu vào và đầu ra nối chéo nhau tạo ra một trigơ RS cơ bản, cấu trúc thành một phần tử nhớ. T_1 thông T_3 ngắt là trạng thái 0, T_1 ngắt T_3 thông là trạng thái 1. Dây X_i điều khiển T_5, T_6 thông hoặc ngắt, do đó điều khiển nối đầu ra của trigơ với dây bit.

Khi $X_i = 0$ thì T_5, T_6 thông, do đó trigơ nối với dây bit khi $X_i = 0$ thì T_5, T_6 ngắt, trigơ ngắt khỏi dây bit. T_7, T_8 điều khiển nối, ngắt giữa dây bit và dây dữ liệu, bằng tín hiệu dây Y_j. Khi $Y_j = 1$ thì T_7, T_8 thông dây bit nối với dây dữ liệu D và … $\overline{D}$, còn khi $Y_j = 0$ thì T_7, T_8 ngắt dây bit không nối với dây dữ liệu.

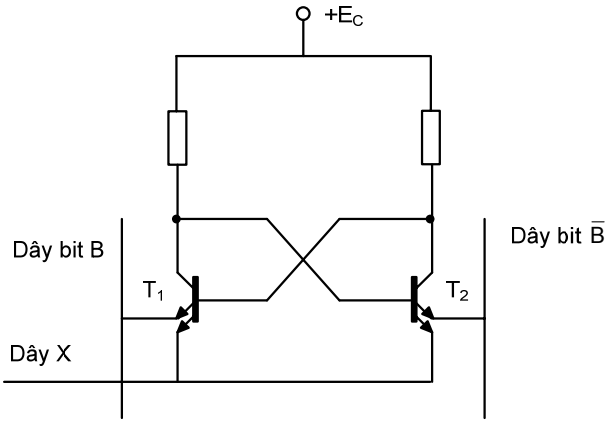
Không phải mỗi phần tử nhớ đều có dùng T_7, T_8; mà T_7, T_8 là dùng chung cho cả cột phần tử nhớ.

Vậy phần tử nhớ nào mà các giá trị X_i, Y_j của nó đều là 1 thì mới được nối thông với dây dữ liệu, nghĩa là khi đó mới có thể đọc hoặc viết với phần tử nhớ đó.

Bất kì thời điểm nào thì cũng chỉ có một X_i nào đó ở mức 1 và một Y_j nào đó ở mức 1.

7.5.3.2.2 Phần tử nhớ dùng tranzito lưỡng cực

Hình 7.27 là sơ đồ phần tử nhớ dùng tranzito lưỡng cực.



Hình 7.27: Phần tử nhớ lưỡng cực

Một phần tử nhớ gồm 2 tranzito nhiều emitơ và 2 điện trở, cấu trúc nên một trigơ. Một đầu emitơ nối với dây X, đầu emitơ còn lại nối riêng rẽ vào dây bit B,... Ở trạng thái lưu giữ dữ liệu, điện thế dây X cỡ 0,3V, điện thế dây bit cỡ 1,1V. Khi đó dòng điện của tranzito thông qua dây X, lớp tiếp giáp bán dẫn nối với dây bit phân cực ngược làm hở mạch giữa dây bit và trigơ.

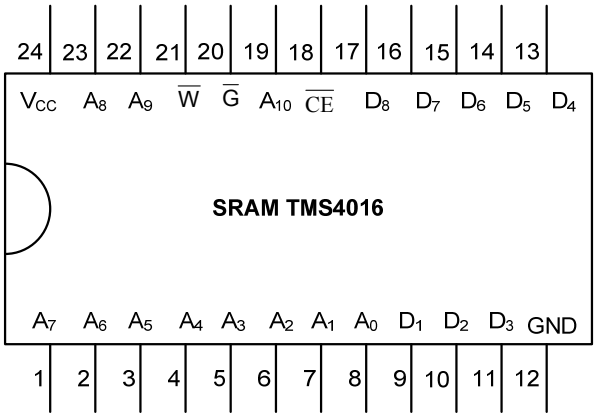
Trạng thái "phần tử nhớ có thể là 0 (T_1 thông, T_2 ngắt) hoặc có thể là 1 (T_1 ngắt, T_2 thông).

Khi phần tử nhớ được chọn (để đọc hoặc viết dữ liệu), điện thế dây X có mức cao hơn cỡ 2,2V, điện thế dây bit nhỏ hơn nên dòng điện qua tranzito thông sẽ chạy ra dây bit.

7.5.4 Một số IC SRAM

Một số IC SRAM thường gặp là IC 2148, 2114-2 của hãng Intel. Dung lượng 1k x 4bit. Thời gian truy nhập thông tin khoảng 200ns, công suất tiệu thụ 525mW.

IC TMS 4016 dung lượng 2k x 8bit. Hình 7.28 là sơ đồ chân của IC này:



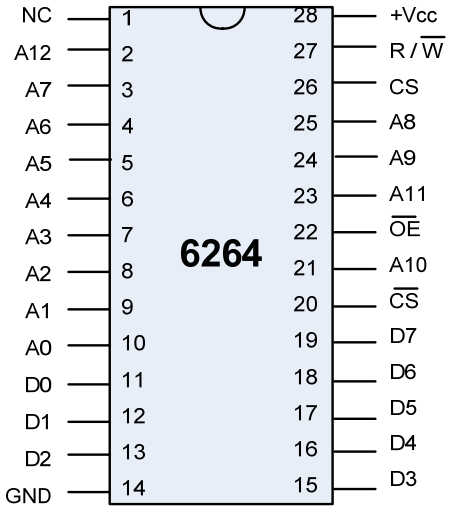
Hình 7.28: Sơ đồ chân của TMS4016

Bảng 7.2 là bảng chức năng của IC.

Bảng 7.2: Bảng chức năng của IC TMS 4016

Phương thức hoạt động	$\overline{W}$	$\overline{CE}$	$\overline{G}$	$D_1 - D_8$
Viết	L	L	X	Dữ liệu xác định
Đọc	H	L	L	Dữ liệu ra
Cấm chip	X	H	X	Trở kháng cao
Cấm ra	H	L	H	Trở kháng cao

IC HM 6116, họ CMOS, dung lượng 2kbyte, thời gian truy nhập là 120ns, công suất tiêu thụ khi làm việc là P = 180mW (khi chờ ≈ μW). Hình 7.29 giới thiệu IC 6264, dung lượng 8kbyte và bảng điều kiện thao tác của nó.



Hình 7.29: Sơ đồ chân của SRAM 6264

Bảng 7.3: Bảng chức năng của IC 6264

Phương thức hoạt động	$\overline{CS}$	CS	$\overline{WE}$	$\overline{OE}$
Không được chọn	H	X	X	X
Đọc	L	H	H	L
Đọc nhưng không xuất dữ liệu	L	H	H	H
Viết	L	H	L	L

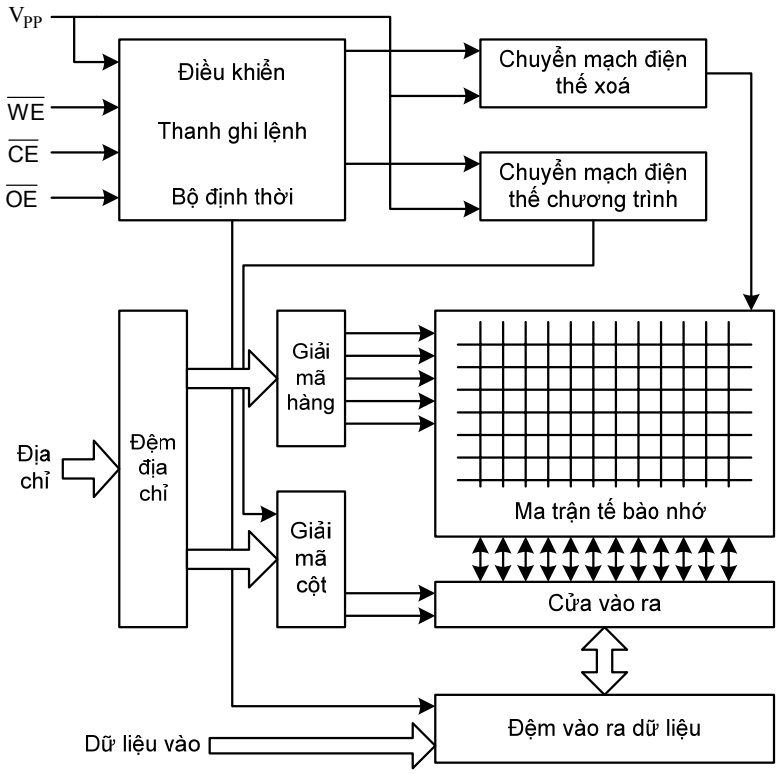
7.6 ĐĨA CỨNG SILICON- BỘ NHỚ FLASH

Trong những năm gần đây, một loại bộ nhớ không bay hơi mới đã xuất hiện trên thị trường, thường được sử dụng thay thế cho các ổ đĩa mềm và cứng trong những máy tính. Đó là bộ nhớ flash. Cấu trúc của chúng cơ bản như EEPROM, chỉ có lớp kênh ôxit ở các ô nhớ mỏng hơn. Do vậy chỉ cần điện thế cỡ 12V là có thể cho phép thực hiện 10.000 chu trình xoá và lập trình. Bộ nhớ flash có thể hoạt động gần mềm dẻo như DRAM và SRAM nhưng lại không bị mất dữ liệu khi bị cắt điện. Hình 7.30 chỉ ra sơ đồ khối của nó.

Phần chính là mạng nhớ bao gồm các ô nhớ FAMOST như được mô tả ở mục trên. Giống như SRAM, bộ nhớ flash không dồn phân kênh địa chỉ. Các bộ giải mã hàng và cột chọn một đường từ và một hoặc nhiều cặp đường bit. Dữ liệu đọc được đưa ra ngoài bộ đệm dữ liệu I/O hoặc được viết vào ô nhớ đã được định địa chỉ bởi bộ đệm này qua cổng I/O. Xử lý đọc được thực hiện với điện thế MOS thông thường là 5V. Để lập trình một ô nhớ, đơn vị điều khiển flash đặt một xung điện thế ngắn cỡ 10μs và 12V gây nên một sự chọc thủng thác lũ vào tranzito nhớ để nạp vào cửa nổi. Một chip nhớ flash 1Mbit có thể được lập trình trong khoảng 2s, nhưng khác với EEPROM việc xoá được thực hiện từng chip một. Thời gian xoá cho toàn bộ bộ nhớ flash khoảng 1s. Xử lý đọc, lập trình và xoá được điều khiển bởi các lệnh có độ dài 2byte được bộ xử lý viết vào các thanh viết lệnh của mạch điều khiển flash.

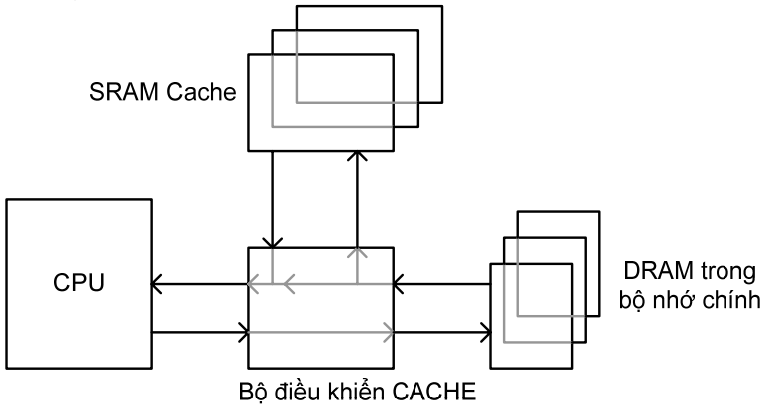
Mục đích sử dụng chính của bộ nhớ flash là để thay thế cho các ổ đĩa mềm và ổ đĩa cứng dung lượng nhỏ. Do nó là mạch tích hợp nên có ưu điểm là kích thước nhỏ và tiêu thụ năng lượng thấp, không bị ảnh hưởng của va đập. Các đĩa cứng chất rắn dựa trên cơ sở các bộ nhớ flash có lợi thế về công suất tiêu thụ cũng như giá thành có dung lượng tới vài MB. Các card nhớ loại này có ưu điểm là không gặp phải vấn đề mất thông tin như trường hợp RAM CMOS khi pin Ni-Cd bị hỏng. Thời gian lưu trữ thông tin trong bộ nhớ flash ít nhất là 10năm, thông thường là 100 năm, với khoảng thời gian này thì các đĩa mềm và cứng đã bị hỏng rồi.

Nhược điểm của bộ nhớ flash là chỉ có thể xoá theo kiểu lần lượt từng chip hoặc lần lượt từng trang.



Hình 7.30: Sơ đồ bộ nhớ FLASH

7.7 BỘ NHỚ CACHE



Hình 7.31: Nguyên lý của Cache

Với các máy tính có tốc độ nhanh (trên 33MHz), cần phải xen các trạng thái đợi khi truy xuất dữ liệu tới các DRAM rẻ tiền nhưng có thời gian thâm nhập chậm (60-120ns). Điều này làm giảm hiệu suất của máy. Có thể giải quyết bằng cách dùng các SRAM có thời gian thâm nhập ngắn hơn (20-25ns, thậm chí 12ns) nhưng giá thành lại rất đắt. Bộ nhớ Cache kết hợp được các lợi điểm nhanh của SRAM và rẻ của DRAM. Giữa CPU và bộ nhớ chính bằng DRAM, người ta xen vào một bộ nhớ SRAM nhanh có dung lượng nhỏ bằng 1/10 hoặc 1/100 lần bộ nhớ chính gọi là cache; dưới sự điều khiển của mạch điều khiển cache, bộ nhớ này sẽ lưu trữ tạm thời các dữ liệu thường được gọi và cung cấp nó cho CPU trong thời gian ngắn.

Cache chứa các thông tin mới vừa được CPU sử dụng gần đây nhất. Khi CPU đọc dữ liệu nó sẽ đưa ra một địa chỉ tới bộ điều khiển cache. Sau đó một trong hai quá trình sau sẽ xảy ra:

Cache hit: nếu địa chỉ đó đã có sẵn trong RAM cache.

Cache miss: ngược lại, nếu địa chỉ đó không có sẵn trong RAM cache.

Như vậy, cache hit tỷ lệ với truy xuất thông tin có sẵn trong bộ nhớ cache SRAM, còn cache miss lại tỷ lệ với truy xuất thông tin có trong bộ nhớ chính là các DRAM.

TÓM TẮT

Trong chương này chúng ta trình bày nguyên lý cấu tạo, các tính năng cơ bản của các loại bộ nhớ bán dẫn: ROM, PROM, EPROM, EEPROM, SRAM, DRAM, FLASH, CACHE.

Các chip RAM không thích hợp cho các chương trình khởi động do các thông tin trên đó bị mất khi tắt nguồn. Do vậy phải dùng đến ROM, trong đó các dữ liệu cần lưu trữ được viết một lần theo cách không bay hơi để nhằm giữ được mãi.

Trong những năm gần đây, một loại bộ nhớ không bay hơi mới đã xuất hiện trên thị trường, thường được sử dụng thay thế cho các ổ đĩa mềm và cứng trong những máy tính. Đó là bộ nhớ flash. Cấu trúc của chúng cơ bản như EEPROM, chỉ có lớp kênh ôxit ở các ô nhớ mỏng hơn.

CÂU HỎI ÔN TẬP

1. Bộ nhớ ROM về cơ bản khác bộ nhớ RAM ở điểm gì?
2. Linh kiện lưu giữ bit thông tin của DRAM là linh kiện gì?
3. Linh kiện lưu giữ bit thông tin của SRAM là linh kiện gì?
4. Linh kiện lưu giữ bit thông tin của EPROM là linh kiện gì?
5. Trong EPROM, việc nạp các điện tích vào vùng cửa nổi có nghĩa là làm gì cho nó?
6. SRAM 6264: Cần bao nhiêu bit địa chỉ cho nó khi dung lượng là 2048 từ x 8bit? Nó có thể nhớ bao nhiêu từ 32byte? Nếu ghép song song 4 IC này thì dung lượng bộ nhớ tổng cộng là bao nhiêu?
7. Làm tươi bộ nhớ DRAM là gì? Tại sao phải làm tươi DRAM?
8. Một DRAM có dung lượng nhớ là 64kbit thì cần bao nhiêu đầu vào/ra? Nó chứa được bao nhiêu từ nhị phân 8bit? Vẽ sơ đồ khối của bộ nhớ?

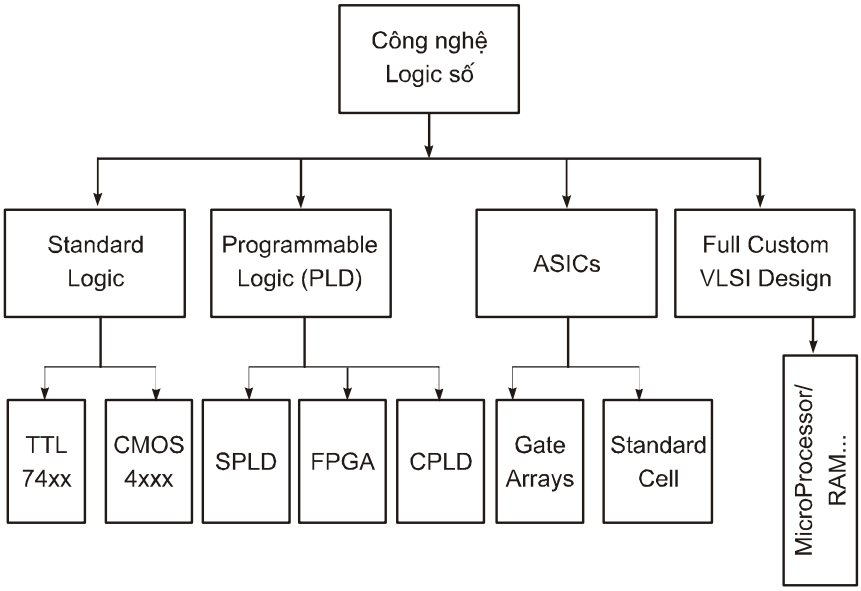
Chương 8 CẤU KIỆN LOGIC KHẢ TRÌNH (PLD)

GIỚI THIỆU

Trong các chương trước ngoài lý thuyết về phân tích, thiết kế hệ mạch số, cũng đã giới thiệu các ví dụ dùng công nghệ logic số sử dụng các IC có chức năng cố định để thiết kế mạch số. Chương này trình bày tổng quan hơn về các công nghệ logic số phổ biến có thể được sử dụng trong thực tế để xây dựng mạch, hệ thống số. Đặc biệt chương này sẽ trình bày kỹ hơn về công nghệ logic khả trình với các cấu kiện logic khả trình cho phép người dùng lập trình lại cấu hình của nó để có chức năng mới. Hai loại cấu kiện logic khả trình CPLD, FPGA hiện đang được sử dụng khá phổ biến giúp phát triển nhanh các ứng dụng từ đơn giản đến phức tạp. Lưu đồ thiết kế cho chúng sẽ được trình bày trong chương này.

8.1 GIỚI THIỆU VỀ CÔNG NGHỆ LOGIC SỐ

Ngày nay công nghệ logic số đã phát triển rất mạnh mẽ và là công nghệ chủ đạo trong việc phát triển các sản phẩm điện - điện tử công nghiệp và dân dụng, điều khiển - tự động, viễn thông và công nghệ thông tin. Hiện nay có có rất nhiều loại công nghệ logic số khác nhau được sử dụng để thực hiện các thiết kế logic số. Biểu đồ phân loại các công nghệ logic số như hình 8.1. Trong đó gồm có:



Hình 8.1: Phân loại công nghệ logic số

8.1.1 Công nghệ logic chuẩn (Standard Logic)

Đây là công nghệ logic truyền thống gồm có các IC số chức năng cố định, công nghệ này có 2 họ cấu kiện logic điển hình là TTL 74xx và CMOS 4xxx. Chức năng của mỗi cấu kiện là cố định do nhà sản xuất tạo ra, người sử dụng chỉ thực hiện kết nối chúng với nhau để xây dựng mạch ứng dụng. Các IC số theo công nghệ logic chuẩn này rất đa dạng từ IC chức năng thực hiện các phép toán logic căn bản đến IC thực hiện các chức năng phức tạp khác như: bộ ghép kênh, phân kênh, bộ cộng, so sánh, bộ mã hoá, giải mã, bộ đếm… Chúng là các IC số có chức năng cố định, tức là mỗi IC thực hiện một chứ năng chuyên biệt. Những cấu kiện này được sản xuất một số lượng lớn để đáp ứng nhu cầu ứng dụng phong phú. Để thiết kế một mạch số, nhà thiết kế có thể chọn từ các IC có sẵn phù hợp nhất cho mạch điện. Phần thiết kế này có thể được chỉnh sửa để đáp ứng các yêu cầu chuyên biệt của những linh kiện này.

Ưu điểm của việc sử dụng công nghệ logic chuẩn này là:

- Thực hiện thiết kế đơn giản.
- Chi phí phát triển ứng dụng thấp.
- Thay đổi nhanh bản thiết kế.
- Tương đối dễ thử nghiệm các mạch

Nhược điểm:

- Các yêu cầu về kích thước trong bảng mạch lớn.
- Khó chế tạo được những mạch ứng dụng phức tạp.
- Yêu cầu về điện lớn.
- Thiếu tính bảo mật (các bảng mạch có thể bị sao chép).
- Các yêu cầu về chi phí bổ sung, khoảng trống, điện… cần thiết để chỉnh sửa bản thiết kế hoặc bổ sung các tính năng khác.

8.1.2 Công nghệ ASIC

Để khắc phục những nhược điểm của việc thiết kế bằng cách sử dụng các IC chức năng cố định, các mạch tích hợp chuyên biệt ứng dụng (ASIC - Application Specific IC) đã được phát triển. Các ASIC đã được thiết kế để đáp ứng các yêu cầu chuyên biệt của một mạch và được giới thiệu bởi một nhà sản xuất IC. Các thiết kế này quá phức tạp không thể thực hiện bằng cách sử dụng các IC chức năng cố định được Người sử dụng có thể thiết kế cấu hình chúng theo chức năng mong muốn, nhưng cần thực hiện công đoạn sản xuất cuối cùng tại nhà máy để tạo ra một lại IC chuyên dụng cho mục đích riêng của người sử dụng. Hiện nay công nghệ ASIC gồm 2 loại là: ***Gate Arrays*** (mảng cổng) và ***Standard Cell*** (tế bào chuẩn).

Cấu kiện ***Gate Arrays*** được cấu tạo từ mảng các tế bào logic cố định đã được sản xuất trước. Mỗi tế bào logic (logic cell) gồm có vài cổng logic hoặc một Flip-Flop. Cần có một bước sản xuất cuối cùng

để thực hiện tạo ra lớp kết nối các logic cell này theo mẫu kết nối đã được tạo ra bởi người sử dụng khi thực hiện một thiết kế xác định.

Cấu kiện ***Standard Cell*** không có cấu trúc cố định, nhà sản xuất tạo ra mặt nạ riêng để xây dựng IC dựa vào những lựa chọn linh kiện của người sử dụng như các bộ điều khiển, ALU, RAM, ROM, vi xử lý... từ thư viện standard cell mà nhà sản xuất đã đưa ra. Các lớp mặt nạ này sẽ được thực hiện trong quá trình sản xuất tại nhà máy.

Ưu điểm của việc sử dụng công nghệ ASIC là:

- Giảm thiểu được kích thước thông qua việc sử dụng mức tích hợp cao.

- Giảm thiểu được yêu cầu về điện.

- Nếu được sản xuất theo một quy mô lớn thì chi phí giảm đáng kể.

- Việc thiết kế được thực hiện dưới dạng này thì hoàn toàn không thể sao chép được.

Nhược điểm:

- Chi phí phát triển ban đầu có thể cực kỳ lớn.

- Các phương pháp thử nghiệm phải được phát triển và điều này làm gia tăng chi phí và công sức.

8.1.3 Công nghệ logic khả trình (Programmable Logic)

Có một phương pháp khác có các ưu điểm của hai phương pháp trên là sử dụng các cấu kiện logic khả trình (PLD). Một cấu kiện logic khả trình là một IC số mà người dùng có thể cấu hình để chúng để có khả năng thực hiện các chức năng logic như mong muốn. Đây là một chip LSI có chứa một cấu trúc mà cho phép nhà thiết kế tạo tuỳ biến cho nó để dùng cho bất kỳ ứng dụng đặc biệt nào, tức là nó có thể được người dùng lập trình lại cấu hình để thực hiện một chức năng cần thiết cho ứng dụng của họ.

Các PLD có các ưu điểm sau:

- Thời gian thiết kế ứng dụng ngắn.
- Chi phí phát triển thấp.
- Giảm thiểu được yêu cầu khoảng trống trên bảng mạch.
- Giảm thiểu được yêu cầu về điện.
- Bảo đảm tính bảo mật của thiết kế.
- Tốc độ chuyển mạch nhanh hơn.
- Mật độ tích hợp cao.
- Chi phí sản xuất số lượng lớn thấp.
- PLD cũng cho phép nhà thiết kế có nhiều phương tiện linh động hơn để thử nghiệm với các bản thiết kế bởi vì chúng có thể được cấu hình lại trong vài giây.

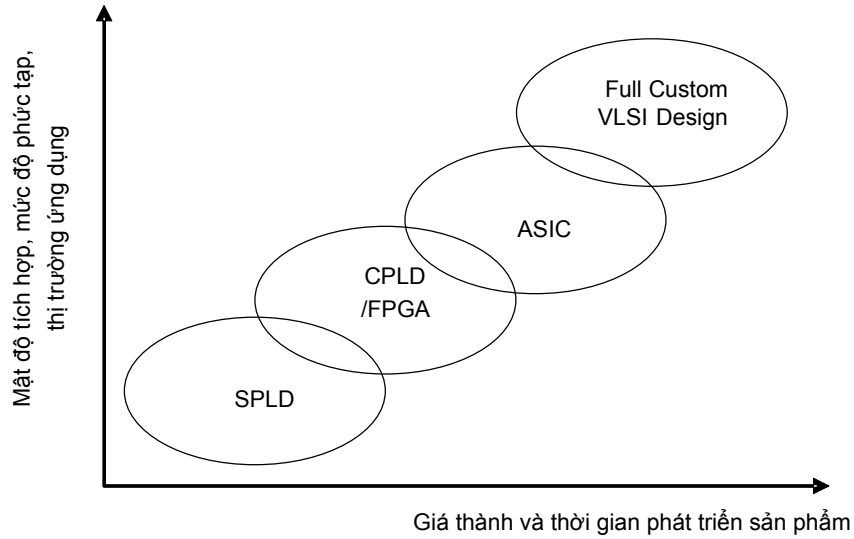
Với nhiều ưu điểm như vậy nên hiện nay có một số lượng lớn các PLD được các nhà sản xuất IC tạo ra với nhiều tính năng đa dạng và nhiều tuỳ chọn có sẵn để nhà thiết kế mạch có thể sử dụng một cách phổ biến.

Một số cấu trúc của PLD như: Mảng logic khả trình (PLA), logic mảng khả trình (PAL), Cấu kiện logic khả trình đơn giản (SPLD) và Mảng cổng có thể lập trình theo trường (FPGA) sẽ được đề cập kỹ hơn trong các phần sau.

8.1.4 Công nghệ thiết kế vi mạch số mật độ tích hợp lớn *(Full Custom VLSI Design)*

Công nghệ thực hiện việc thiết kế vi mạch số ở mức tranzito trên bề mạch của tinh thể bán dẫn, cho phép tạo ra vi mạch có chức năng mong muốn, thường sử dụng để thiết kế các vi mạch đa dụng có tính năng mạnh và có phạm vi ứng dụng lớn, điển hình như các họ vi xử lý (MicroProcessor), bộ nhớ RAM,... các vi mạch số có độ phức tạo cao, tính năng mạnh... Khi sử dụng các họ vi xử lý để phát triển ứng dụng,

thiết kế của người sử dụng được chương trình hoá và được cài đặt vào các thiết bị nhớ. Nhiệm vụ chính của việc thiết kế là lập trình cho các vi xử lý này theo bài toán đã đặt ra. Việc thay đổi thiết kế khá dễ dàng thích hợp với những bài toán ứng dụng có thuật toán phức tạp. Tuy nhiên với nhiều ứng dụng thiết kế logic số vẫn cần phải ghép nối vi xử lý với nhiều cấu kiện số khác, việc thiết kế mạch khá phức tạp. Đa phần hoạt động của các vi xử lý là các tiến trình tuần tự, do đó với các ứng dụng đòi hỏi tốc độ cao, xử lý song song thì việc sử dụng vi xử lý là khá khó khăn.



Hình 8.2: Sự cân bằng trong việc ứng dụng của công nghệ logic số

Sự phức tạp khi sử dụng, thời gian thiết kế, cũng như độ phức tạp của ứng dụng là khác nhau đối với mỗi loại công nghệ logic số. Biểu đồ về sự cân bằng giữa mật độ tích hợp, mức độ phức tạp, thị trường ứng dụng của các công nghệ logic số và giá thành cũng như thời gian phát triển sản phẩm được đưa ra trong hình 8.2. Việc sử dụng cấu kiện đa dụng như vi xử lý rất nhanh, tuy nhiên việc phát triển vi mạch VLSI đa dụng cho một thiết kế ở mức tranzito có thể

mất tới vài năm cho việc thiết kế và kiểm tra. Do đó công nghệ này phù hợp cho việc phát triển những loại cấu kiện có số lượng cao nhất, hiệu năng cao nhất, đa dụng nhất ví dụ như vi xử lý, bộ nhớ... sử dụng trong máy tính. Giá thành và thời gian phát triển sản phẩm dùng ASIC giảm hơn công nghệ thiết kế VLSI, tuy vậy vẫn cần thời gian và chi phí phát triển cho công đoạn sản xuất cuối cùng ở nhà máy. Việc kiểm tra được thực hiện bởi người sử dụng sau công đoạn sản xuất cuối cùng đó, nên nếu có bất kỳ lỗi thiết kế nào cũng sẽ dẫn đến việc tăng thời gian và giá thành phát triển. Công nghệ này thích hợp với phát triển các sản phẩm có số lượng và thời gian sống lớn. Công nghệ logic khả trình mà điển hình nhất là CPLD và FPGA rất thích hợp cho yêu cầu phát triển sản phẩm nhanh, số lượng không lớn và có mật độ tích hợp cũng độ phức tạp không quá lớn và đặc biệt tiện lợi trong giai đoạn nghiên cứu phát triển, chế tạo thử sản phẩm. Tuy nhiên sự cân bằng về hiệu năng và tính kinh tế giữa ASIC, CPLD, FPGA sẽ liên tục thay đổi với mỗi thế hệ cấu kiện và công cụ thiết kế mới.

Tóm lại trong số các công nghệ logic số hiện nay, công nghệ logic khả trình là lựa chọn khá thích hợp cho việc phát triển các hệ thống số không quá phức tạp, với số lượng không lớn, thời gian phát triển nhanh, nhất là ở những nước có nền công nghiệp điện tử cũng như sản xuất vi mạch mới phát triển như Việt Nam.

8.2 CẤU KIỆN LOGIC KHẢ TRÌNH (PLD)

Vào cuối thập kỷ 70, các thiết bị logic chuẩn xuất hiện ồ ạt, đi kèm với đó là sự xuất hiện mạch in. Người ta đặt ra câu hỏi: "Chuyện gì xảy ra nếu người thiết kế có thể thực hiện các kết nối khác nhau trong một thiết bị lớn hơn?" Điều này cho phép người thiết kế tích hợp nhiều thiết bị logic chuẩn trong một linh kiện. Để có thiết kế linh hoạt nhất, nhà sản xuất Ron Cline từ Signetics (sau này được Philips và thậm chí cả Xilinx) đưa ra ý tưởng dùng hai ma trận kết nối khả trình. Hai ma trận kết nối khả trình này có thể tổ hợp tuỳ ý giữa các cổng

AND và cổng OR, đồng thời cho phép nhiều cổng OR cùng sử dụng chung một cổng AND. Kiến trúc này rất linh hoạt, nhưng tại thời điểm đó, trễ lan truyền từ đầu vào tới đầu ra (T_{pd}) khá cao (do cấu trúc 10µm) nên thiết bị hoạt động tương đối chậm. Và dạng công nghệ logic khả trình đầu tiên xuất hiện chính là SPLD. Sau này công nghệ CPLD và FPGA ra đời có mật độ tích hợp cao hơn, cấu trúc linh hoạt hơn cho phép tạo ra nhiều mạch logic phức tạp hơn

Vi mạch lập trình, viết tắt là PLD (Programmable Logic Device), là loại cấu kiện điện tử có nhiều ưu điểm và hiện nay đang được phát triển rất mạnh. Về nguyên lý, chúng có cấu tạo rất giống với PROM. Việc lập trình cho PLD có thể được thực hiện bằng các công nghệ khác nhau, dựa trên cơ sở bẻ cầu chì hoặc chuyển mạch. Tuy nhiên, ứng dụng của PLD lại rất khác với PROM. Một PLD, được tạo thành bằng một số cổng AND, OR, XOR hoặc cả các trigơ, có thể thực hiện nhiều hàm Boole khác nhau.

8.2.1 SPLD

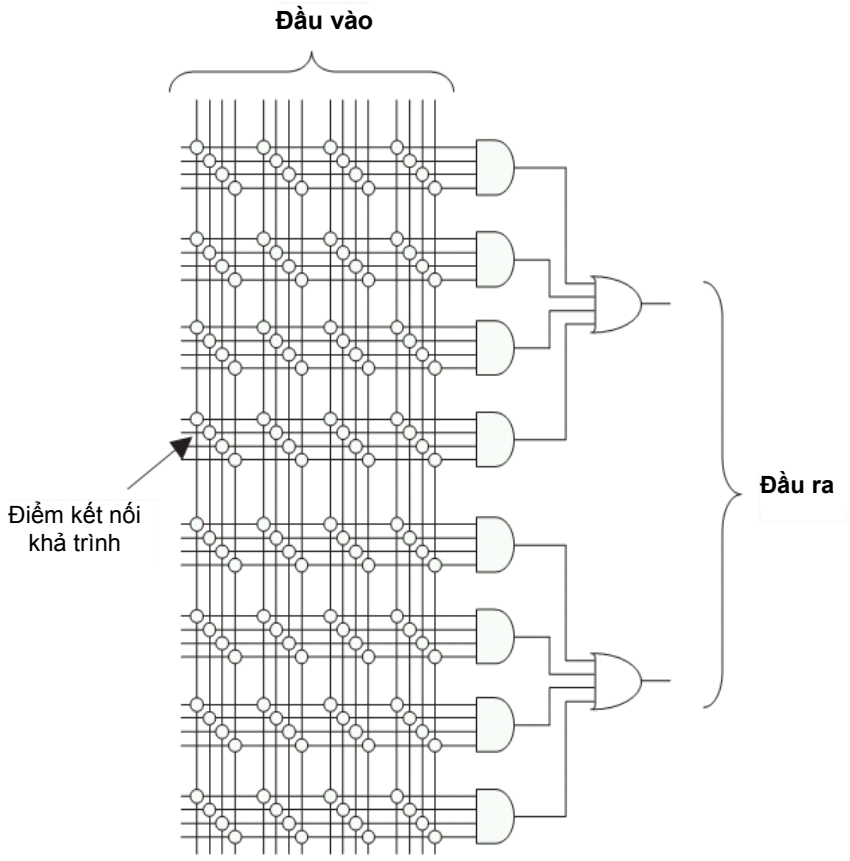
SPLD - cấu kiện logic khả trình đơn giản. Đây là loại cấu kiện số có nhiều ưu điểm và cũng đã được phát triển rất mạnh. Về nguyên lý, chúng có cấu tạo rất giống với PROM. Việc lập trình cho SPLD có thể được thực hiện bằng các công nghệ khác nhau, dựa trên cơ sở thực hiện các kết nối bằng cách sử dụng cầu chì hoặc chuyển mạch. Một SPLD, được tạo thành bằng một số mảng cổng AND, OR, XOR hoặc cả các trigơ, có thể thực hiện nhiều hàm Boole khác nhau.

Các SPLD đều có cấu tạo dựa trên một trong hai dạng cấu trúc chính: mảng logic khả trình PLA (Programmable Logic Array) và logic mảng khả trình PAL (Programmable Array Logic).

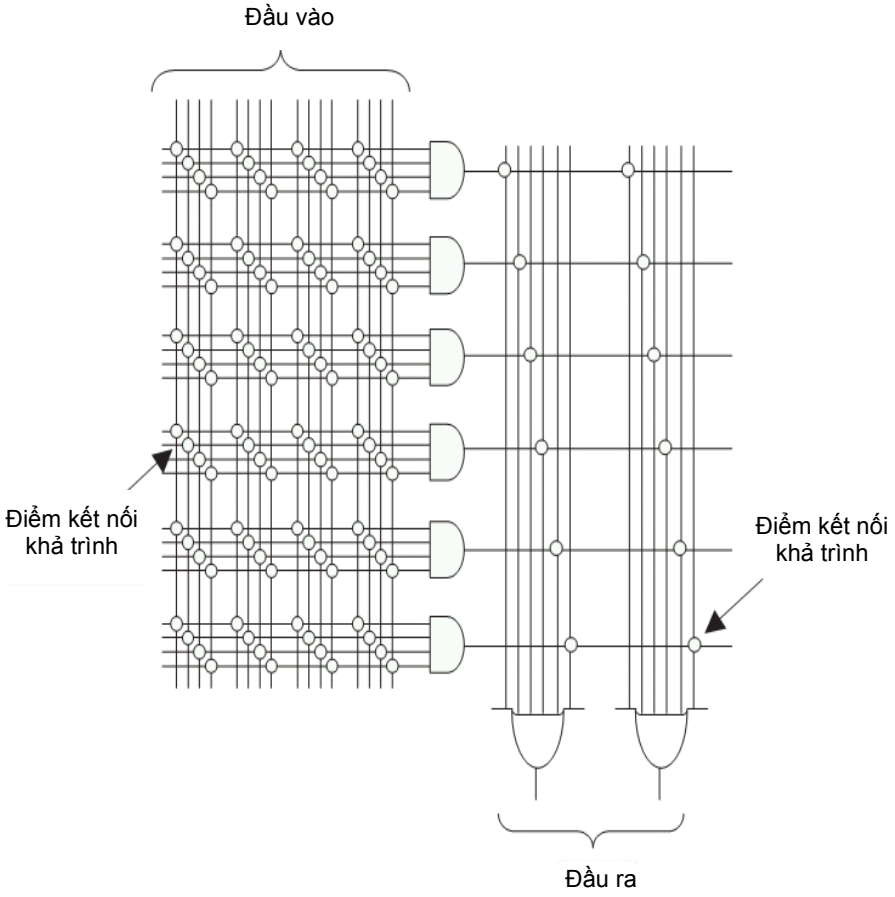
Thành phần cơ bản của PLA là một mảng AND và một mảng OR lập trình được. Mỗi mảng AND, OR gồm các hàng và các cột liên kết với nhau. Tại mỗi điểm giao giữa hàng và cột, có một cầu chì. Khi

cầu chì đóng, tại điểm đó có kết nối giữa hàng và cột, khi cầu chì ngắt, tại đó không có kết nối. Việc đóng ngắt cầu chì được thực hiện bằng phần mềm (do lập trình viên hoặc sử dụng công cụ lập trình trên hệ thống (ISP: In- System Programming).

Cấu trúc PLA tạo ra sự tổ hợp tùy ý giữa các cổng AND và OR, cho mật độ logic cao nhưng tốc độ chậm, số lượng cầu chì lớn. Vì vậy, sau này người ta đã đưa ra một kiểu kiến trúc khác là logic mảng khả trình PAL (Programmable Array Logic).



Hình 8.3: Kiến trúc PAL



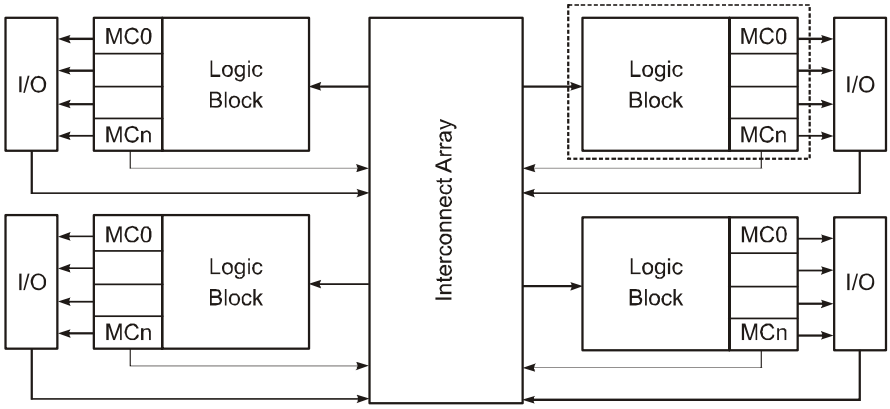
Hình 8.4: Kiến trúc PLA

Công nghệ PLD xuất hiện từ rất sớm với các công ty như Xilinx – sản xuất vi mạch CMOS công suất cực thấp dựa trên công nghệ Flash. PLD dựa trên công nghệ Flash cho phép lập trình và xoá vi mạch nhiều lần bằng điện, nhờ đó tiết kiệm được thời gian so với xoá vi mạch bằng tia cực tím.

8.2.2 CPLD (Complex PLD)

CPLD thuộc về dòng chip khả lập trình PLD nhưng thiết bị logic khả trình phức hợp (CPLD) có mật độ logic cao hơn so với các PLD

đơn giản như đã xét ở trên (PLA và PAL). CPLD bao gồm nhiều mạch logic, mỗi mạch có thể coi là một SPLD. Trong một mạch đơn chỉ thực hiện các chức năng logic đơn giản. Các chức năng logic phức tạp hơn cần số lượng khối nhiều hơn, sử dụng ma trận liên kết chung giữa các khối để tạo kết nối. CPLD thường dùng để điều khiển ghép cổng phức hợp ở tốc độ rất cao (5ns, tương đương với 200MHz). Kiến trúc cơ bản của CPLD được minh hoạ trong hình 8.5.



Hình 8.5: Kiến trúc chung của CPLD

CPLD có cấu trúc đồng nhất gồm nhiều khối chức năng ***"Function Block"*** được kết nối với nhau thông qua một ma trận kết nối trung tâm "***Interconnect Array".*** Mỗi khối chức năng gồm có một khối logic (***logic block***) - gồm các hạng tích AND và hạng tổng OR sắp xếp giống như PLA hoặc PAL, cho phép thực hiện các hàm logic tổ hợp và nhiều khối MC (Macrocell) có chứa tài nguyên là các Trigơ cho phép xây dựng mạch số tuần tự như các thanh ghi và mạch tuần tự... Phần lõi bên trong của CPLD được nối ra bên ngoài thông qua các khối vào/ra I/O cho phép thiết lập chức năng cho các chân của IC có chức năng vào hoặc ra hoặc vừa là chân vào vừa là chân ra, ngoài ra còn có thể thiết lập các chân I/O này làm việc ở các mức logic khác nhau, có điện trở pull-up hoặc pull-down...

Với cấu trúc đồng nhất, giá thành rẻ, tính năng khá mạnh, dễ sử dụng CPLD đã và đang được sử dụng rất rộng rãi trong thực tế, giúp cho nhà sản xuất phát triển nhanh sản phẩm của mình với giá thành rẻ. Đặc biệt hiện nay các hãng đã phát triển các họ CPLD với tính năng rất mạnh, công suất tiêu thụ thấp, chúng đang được sử dụng rất nhiều để phát triển các sản phẩm điện tử, viễn thông, công nghệ thông tin, nhất là trong các thiết bị cầm tay, di động…

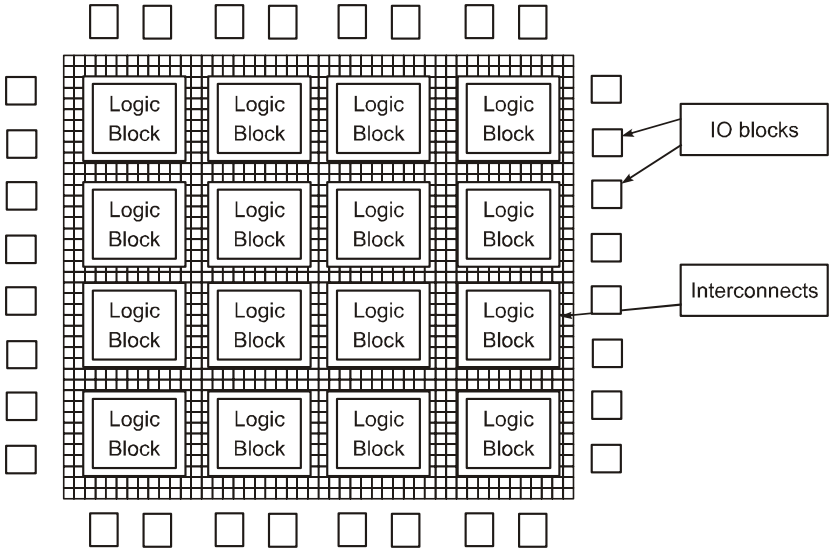
Trong thực tế rất có nhiều loại CPLD khác nhau, của các hãng khác nhau và đã được phát triển với nhiều chủng loại, thế hệ CPLD khác nhau. Cấu tạo, dung lượng, tính năng, đặc điểm, ứng dụng… của mỗi loại CPLD cũng rất khác nhau. Trong giáo trình này không đi sâu trình bày cấu tạo cụ thể của các họ CPLD, mà chỉ trình bày kiến trúc chung đơn giản nhất của CPLD. Khi sử dụng cụ thể loại CPLD nào, người học nên tham khảo các tài liệu khác, nhất là tham khảo các tài liệu kỹ thuật được cung cấp kèm theo cấu kiện do các hãng đưa ra. Các hãng điện tử nổi tiếng trên thế giới đang sở hữu, phát triển, cung cấp các loại cấu kiện CPLD là Xilinx, Altera…

8.2.3 FPGA

FPGA (Field Programmable Gate Array - Ma trận cổng lập trình được theo trường): có cấu trúc và hoạt động phức tạp hơn CPLD. Nó có thể thực hiện những chức năng phức tạp ưu việt hơn CPLD. Năm 1985, công ty Xilinx đưa ra ý tưởng hoàn toàn mới, đó là kết hợp thời gian hoàn thành sản phẩm và khả năng điều khiển được của PLD với mật độ và ưu thế về chi phí của GateArray. Từ đó, FPGA ra đời. Hiện nay, Xilinx vẫn là nhà sản xuất chip FPGA số một trên thế giới.

Cấu trúc FPGA đơn giản gồm các tế bào logic (Logic Cell hay logic Block), các khối cách đều nhau, liên kết nhờ các đường kết nối có thể thay đổi được theo yêu cầu của người thiết kế. Nghĩa là người thiết kế có quyền thiết kế, lập trình và thay đổi mạch điện. Hiện nay,

FPGA có mật độ khá cao, lên tới hàng trăm tỷ cổng và cấu trúc cũng đa dạng, phức tạp hơn. Nhiều chức năng phức tạp đã được tích hợp sẵn để tăng hiệu quả sử dụng FPGA. Ví dụ như ngoài những tế bào logic, nhiều họ FPGA đã được tích hợp thêm các khối chức năng khác như các bộ nhân cứng, khối nhớ, PLL, thậm chí cả một bộ vi xử lý mạnh…



Hình 8.6: Kiến trúc chung của FPGA

Có hai loại FPGA cơ bản: Loại lập trình lại được, dựa trên công nghệ SRAM và loại lập trình một lần.

Loại lập trình lại được (dựa trên SRAM):

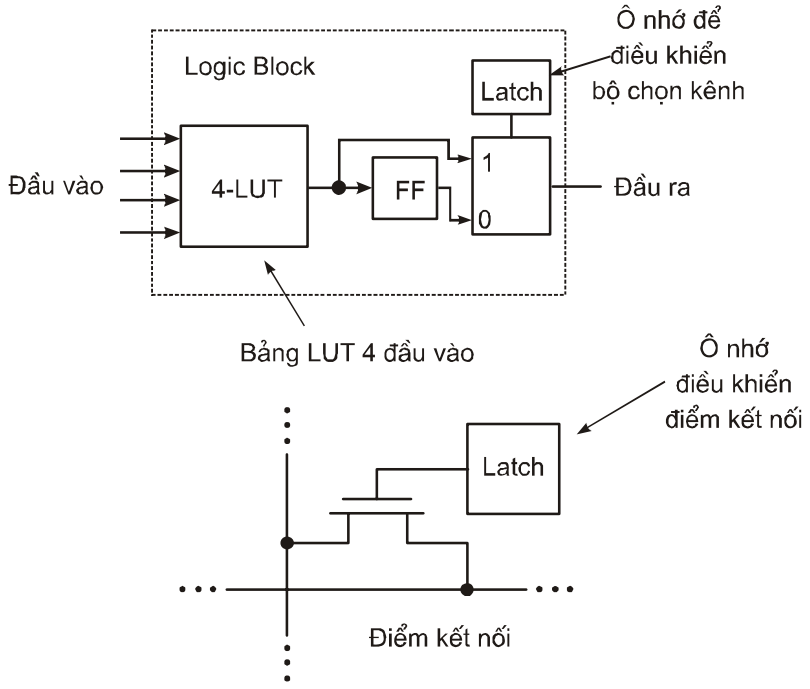
- SRAM xác định các kết nối

- SRAM định nghĩa các hàm logic trong bảng ánh xạ (LUT - Look Up Table)

Loại lập trình một lần:

- Kết nối dạng bẻ cầu chì

- Sử dụng các cổng logic truyền thống



Hình 8.7: Cấu trúc của tế bào logic đơn giản

Hai dạng này khác nhau về quy trình thực hiện tế bào logic và cơ chế được sử dụng để tạo kết nối trong thiết bị.

Chip FPGA lập trình một lần sử dụng phương pháp bẻ cầu chì (kết nối được tạo ra bằng cách đóng cầu chì) để tạo kết nối tạm thời trong chip, do đó không cần SPROM hoặc các phương tiện khác để nạp chương trình vào FPGA. Tuy nhiên, mỗi lần thay đổi thiết kế, phải bỏ hoàn toàn chip cũ đi. Tế bào logic OTP tương tự như PLD với các cổng và các trigơ định trước.

Dạng FPGA quan trọng hơn và được dùng phổ biến hơn cả là dạng lập trình lại được, dựa trên SRAM. Trên thực tế, FPGA SRAM được lập trình lại mỗi khi bật nguồn, vì FPGA là dạng chip nhớ tạm thời. Do đó, mỗi chip FPGA đều cần có một bộ nhớ PROM nối tiếp hoặc một bộ nhớ hệ thống.

Trong tế bào logic SRAM, thay vì các cổng thông thường, người ta sử dụng bảng ánh xạ (LUT). Bảng này xác định các giá trị đầu ra dựa trên các giá trị đầu vào, sử dụng để xây dựng các hàm logic tổ hợp. Trong sơ đồ "Tế bào logic SRAM" minh hoạ ở hình vẽ 8.7, 16 tổ hợp khác nhau của 4 đầu vào sẽ xác định giá trị của đầu ra). Các ô nhớ SRAM cũng được sử dụng để điều khiển kết nối.

Bảng 8.1: So sánh giữa CPLD và FPGA

CPLD	FPGA
- Cấu trúc theo mảng các hạng tích	- Cấu trúc dựa vào LUT
- Mảng kết nối trung tâm	- Ma trận kết nối 2 chiều X-Y
- Mật độ tích hợp trung bình	- Mật độ tích hợp cao
- Tỷ lệ số chân I/O trên microcell lớn	- Tỷ lệ số chân I/O trên microcell nhỏ
- Cấu hình được lưu lại khi mất điện và không thay đổi trong quá trình hoạt động	- Cấu hình nạp vào SRAM, khi mất điện sẽ không còn, cần có bộ nhớ cấu hình PROM, cấu hình có thể được nạp tự động trong quá trình hoạt động.
- Cấu trúc đồng nhất	– Cấu trúc không đồng nhất
	– Nhiều tài nguyên: DLL (Delay Locked Loop: Vòng khoá pha trễ), bộ nhớ, các bộ nhân
– Ứng dụng: Mã hoá và giải mã logic, các máy trạng thái hay các giao diện bus chuẩn (SPI, I2C, SMBus...), ưu điểm nổi bật khi thiết kế các mạch logic nhiều đầu vào.	- Ứng dụng: PCI (Peripheral Component Interface: Giao diện thành phần ngoại vi), giao tiếp nối tiếp tốc độ cao và các bộ vi xử lý nhúng,... ưu thế nổi bật khi thiết kế phức tạp, cần nhiều tài nguyên.

8.3 GIỚI THIỆU PHƯƠNG PHÁP THIẾT LẬP CẤU HÌNH CHO CPLD/FPGA

Để thiết kế và thiết lập cấu hình của các chip CPLD và FPGA, có hai phương pháp phổ biến: phương pháp dùng sơ đồ mô tả và phương pháp dùng ngôn ngữ mô tả phần cứng (HDL: Hardware Description Language). Hai phương pháp trên được tóm tắt dưới đây.

8.3.1 Phương pháp dùng sơ đồ mô tả

Sơ đồ mô tả là phương pháp truyền thống người thiết kế sử dụng để xác định ma trận cổng và các cấu kiện logic khả trình, nó cho phép xác định chính xác các cổng cũng như cách kết nối các cổng đó để được thiết kế mong muốn. Phương pháp sử dụng sơ đồ mô tả gồm 4 bước chính như sau:

Bước 1: Chọn thư viện cấu kiện và công cụ mô tả thiết kế. Sau đó, chọn các cổng cần cho thiết kế từ thư viện, có thể kết hợp tuỳ ý các cổng với nhau. Ở bước này, phải lựa chọn họ cấu kiện sẽ sử dụng, nhưng chưa phải quyết định sử dụng cấu kiện cụ thể nào trong họ để đáp ứng các yêu cầu về tốc độ và kích thước.

Bước 2: Thực hiện kết nối các cổng với nhau, sử dụng lưới hoặc dây nối. Người thiết kế có thể điều chỉnh kết nối giữa các cổng tuỳ ý theo mục đích thiết kế.

Bước 3: Gắn thêm và phân bố các bộ đệm đầu vào và đầu ra. Các bộ đệm này sẽ xác định các chân I/O cho thiết bị.

Bước 4: Bước cuối là tạo ra Netlist. Netlist là file mô tả mạch số dưới dạng text, được tạo bởi công cụ thiết kế. Bản mô tả thiết kế giúp các chương trình khác nắm được các cổng logic có trong mạch, cách kết nối các cổng đó và số các chân I/O. Chuẩn để viết file Netlist phổ biến nhất là dạng EDIF (Electronic Digital Interchange Format: Định dạng trao đổi điện tử số).

Phương pháp thiết kế dùng sơ đồ mô tả có hai nhược điểm chính:

- Khi thiết kế trở nên phức tạp và dùng nhiều cổng hơn, việc sử dụng phương pháp này trở nên khó khăn, việc sửa lỗi cũng khó khăn hơn, thậm chí không khả thi.

- Một nhược điểm khác của phương pháp thiết kế theo sơ đồ là khó thay đổi công nghệ hay nhà cung cấp. Giả sử ban đầu, sử dụng chip FPGA của nhà sản xuất X để thiết kế mạch 10 000 cổng, sau đó chuyển sang sử dụng ma trận cổng, sẽ phải thay đổi toàn bộ các trang sơ đồ có sử dụng ma trận cổng trong thư viện của nhà sản xuất X.

8.3.2. Phương pháp dùng ngôn ngữ mô tả phần cứng (HDL)

Để khắc phục các nhược điểm trên, người ta có thể dùng phương pháp dùng ngôn ngữ mô tả phần cứng (HDL). Có hai ngôn ngữ mô tả phần cứng phổ biến nhất: VHDL và Verilog.

Trong phương pháp này, thay vì dùng các sơ đồ mô tả và các kết nối, người lập trình dùng ngôn ngữ mô tả phần cứng để mô tả các tính năng và hoạt động của từng phần trong hệ thống. Các bước chính để thiết lập cấu hình cho PLD như sau:

Bước 1: Dùng ngôn ngữ mô tả phần cứng (HDL) để mô tả các tính năng và hoạt động của từng phần trong hệ thống. Đồng thời có thể dùng ngôn ngữ mô tả phần cứng HDL mô tả kết nối giữa các phần trong một hệ thống. Đầu ra của quá trình này là một file ở dạng text.

Bước 2: Dùng công cụ synthesis (tổng hợp) để tạo ra file Netlist từ file ở trên. Công cụ synthesis xác định các cổng được sử dụng dựa trên mô hình hoạt động (trong phương pháp thiết kế truyền thống, người thiết kế phải thực hiện thao tác này). Vì Netlist đặc trưng cho họ thiết bị và nhà sản xuất, nên phải sử dụng thư viện của nhà sản xuất tương ứng. Hầu hết các công cụ thiết kế đều cung cấp tên các nhà sản xuất mảng cổng, FPGA và CPLD.

Ngoài ra, khi lựa chọn hay sắp xếp ở mức cổng, người sử dụng có thể tối ưu hoá thiết kế để số lượng cổng sử dụng ít nhất; tối ưu hoá từng phần thiết kế để tăng tốc độ; sử dụng cấu hình cổng phù hợp để giảm công suất; sử dụng cấu hình thanh ghi cho máy trạng thái.

Người thiết kế có thể thử nghiệm với nhiều nhà cung cấp khác nhau, nhiều họ thiết bị khác nhau, nhiều phương pháp tối ưu khác nhau, từ đó có được nhiều giải pháp thiết kế thay vì chỉ có một giải pháp như với phương pháp sơ đồ.

Một ưu điểm nữa của phương pháp dùng ngôn ngữ mô tả phần cứng là rất dễ nâng cấp hệ thống. Sử dụng ngôn ngữ HDL rút ngắn thời gian thiết kế và khi thay đổi thiết kế cũng rất đơn giản. Người thiết kế chọn thư viện và các thông số tối ưu (về tốc độ, diện tích,…), công cụ synthesis sẽ xác định kết quả. Người thiết kế có thể lựa chọn phương án thiết kế tối ưu sau khi thử các phương án khác nhau.

Sau khi đã tạo được file Netlist, để lập trình chip PLD dựa trên cấu hình đã được thiết lập, người lập trình cần dùng các công cụ khác. Ngoài ra, người lập trình có thể chạy mô phỏng và kiểm tra thiết kế của mình. Các vấn đề này sẽ được đề cập rõ hơn trong các chương sau.

8.4 YÊU CẦU CHUNG KHI THIẾT KẾ VỚI CPLD/FPGA

(Ví dụ với CPLD/FPGA của Xilinx)

8.4.1 Chọn vi mạch CPLD hoặc FPGA phù hợp

Khi phát triển các hệ thống số sử dụng CPLD/FPGA bước đầu tiên cần được thực hiện là phân tích bài toán, lựa chọn vi mạch CPLD hoặc FPGA phù hợp. Việc chọn được vi mạch, công nghệ phù hợp nhất cho các tiêu chuẩn thiết kế, được tiến hành theo các yêu cầu sau:

Mật độ: Là mật độ logic dự tính của linh kiện, đặc trưng bởi khái niệm "số lượng cổng".

Số lượng thanh ghi: Phải tính được số thanh ghi cần cho bộ đếm, máy trạng thái, thanh ghi và bộ chốt. Số lượng macrocell trong vi mạch tối thiểu phải bằng số thanh ghi cần có.

Số lượng chân vào/ra: Phải xác định vi mạch thiết kế cần bao nhiêu đầu vào, bao nhiêu đầu ra.

Yêu cầu về tốc độ: Tuyến tổ hợp nhanh nhất sẽ xác định t_{pd} (trễ truyền trong vi mạch, tính theo ns). Mạch tuần tự nhanh nhất sẽ xác định tần số tối đa của vi mạch (f_{max}).

Đóng vỏ: Phải xác định vi mạch cần gọn nhất hay chỉ sử dụng dạng QFP (Quad Flat Package: Kết hợp sơ đồ bên trong) thông thường. Hoặc vi mạch thiết kế thuộc dạng có lắp chân cắm, trong trường hợp này là vi mạch PLCC (Programmable Logic Controller Circuit).

Công suất thấp: Phải xác định sản phẩm sẽ sử dụng nguồn pin hay năng lượng mặt trời, thiết kế có yêu cầu công suất tiêu thụ thấp hay không, vấn đề tổn hao nhiệt có quan trọng hay không?

Chức năng cấp hệ thống: Phải xác định bo mạch có bao gồm nhiều vi mạch đa mức điện áp hay không, giữa các vi mạch có phải chuyển mức hay không, có yêu cầu sửa dạng xung đồng bộ hay không, có yêu cầu giao tiếp giữa bộ nhớ và bộ vi xử lý hay không?

8.4.2 Chọn giải pháp cấu hình cho CPDL/FPGA

8.4.2.1 Lập trình ngay trên hệ thống

Các CPLD và FPGA của các hãng nói chung, của Xilinx nói riêng có thể được lập trình ngay trên hệ thống (vi mạch đã được hàn vào mạch ứng dụng) thông qua giao thức JTAG (Joint Test Advisory Group) đã được tích hợp sẵn trong IC. Người thiết kế sử dụng cáp nạp để nạp cấu hình cho CPLD hoặc FPGA. Xilinx đưa ra một số chuẩn cáp nạp như sau:

+ ***MultiLINX:*** Cáp nạp dựa trên giao chuẩn giao tiếp nối tiếp USB hoặc RS232, cáp nạp này có tốc độ truyền trong dải rộng và giao diện có điện áp điều chỉnh được để phù hợp với việc giao tiếp với các hệ thống và các chân I/O hoạt động ở các mức điện áp khác nhau 5V;

3,3V; 2,5V. Và được thiết kế để hỗ trợ để cho các phần mềm gỡ rối phần cứng trước kia, nay chúng đã trở lên lỗi thời khi có sự ra đời của công cụ gỡ rối phần cứng ChipScope ILA.

+ ***Parallel Cable IV:*** Cáp nạp sử dụng cổng giao tiếp song song của máy tính, được phát triển để thay thế cho chuẩn cáp nạp Parallel Cable III và cho phép tăng tốc độ lên hơn 10 lần và hỗ trợ cho tất các các vi mạch sử dụng mức điện áp I/O từ 5V xuống 1,5V. Hiện nay chuẩn cáp nạp này được dùng phổ biến hơn cả.

8.4.2.2 Lập trình bên ngoài

Các CPLD và FPGA của Xilinx cũng có thể được lập trình bên ngoài bởi bộ lập trình chip HW130 của Xilinx cũng như các bộ lập trình của các nhà phát triển khác. Điều này cũng thuận tiện cho việc sử dụng các chip được lập trình trước trong thời gian sản xuất.

Cấu hình của CPLD được nạp vào FLASH nên khi mất điện cấu hình không bị mất đi, trong khi đó cấu hình khi hoạt động của FPGA được ghi vào SRAM nên sẽ mất đi khi mất điện, vì vậy cần sử dụng FPGA và kết hợp với EEPROM lưu cấu hình phù hợp, mỗi khi bật nguồn, cấu hình sẽ nạp tự động từ PROM vào FPGA. Có thể sử dụng EEPROM nối tiếp hoặc song song, tuy nhiên thì loại EEPROM nối tiếp hay được sử dụng hơn cả. Khi thiết kế cần chọn loại EEPROM có dung lượng phù hợp với mật độ của các loại FPGA khác nhau.

Ngoài ra Xilinx còn cung cấp các giải pháp được thiết kế trước, dễ sử dụng để cấu hình cho tất cả CPLD và FPGA của Xilinx, nhất là khi thiết kế các hệ thống phức tạp. Tất cả các nội dung liên quan đến cấu hình, EEPROM cho FPGA hay ISP cho CPLD, đều được đưa ra. Các giải pháp sử dụng công cụ "3rd part boundary scan", các giải pháp phần mềm kèm theo, cáp ISP, thiết bị kiểm tra tự động ATE và hỗ trợ lập trình cũng như các thiết bị lưu trữ cấu hình.

8.4.2.3 Nhóm cấu hình hệ thống ACE

Giải pháp cấu hình hiện đại nhất là nhóm cấu hình Hệ thống ACE. Với giải pháp Hệ thống ACE, người thiết kế có thể dễ dàng sử dụng giao diện vi xử lí trong Hệ thống ACE để trực tiếp phối hợp cấu hình FPGA theo các yêu cầu của hệ thống. Giải pháp đầu tiên trong nhóm này là Hệ thống ACE CF, cung cấp công nghệ điều khiển ổ đĩa "Microdrive" kích thước một inch và "CompactFlash" cũng như bộ lưu trữ cấu hình có dung lượng 8Gbit. Ngoài ra, Hệ thống ACE CF cũng được thiết kế trước, cung cấp các đặc tính hiện đại để tận dụng khả năng cấu hình lại linh hoạt của FPGA, bao gồm:

- Cấu hình multi-board từ một nguồn duy nhất
- Quản lý luồng bit đa cấu hình
- Nâng cấp cấu hình qua mạng (IRL)
- Hot-swapping
- Khởi tạo trung tâm xử lí và lưu trữ phần mềm
- Mã hóa

Với hệ thống ACE CF, người thiết kế có thể thực hiện được gần như toàn bộ các yêu cầu cấu hình cho FPGA. Các khả năng bổ trợ hệ thống này cho phép người thiết kế sử dụng FPGA thỏa mãn các yêu cầu định trước về mặt thiết kế và thời gian xử lí lỗi. Ngoài ra, các cổng vi xử lí và cổng kiểm tra JTAG còn cho phép tích hợp Hệ thống ACE trong mọi hệ thống.

Một số đặc điểm của giải pháp cấu hình hệ thống ACE:

- Độ linh hoạt: Với hệ thống ACE CF, có thể sử dụng một thiết kế cho nhiều ứng dụng khác nhau, nhờ đó giảm đáng kể thời gian hoàn thành sản phẩm. Thay vì thiết kế vài bo mạch tương tự nhau phù hợp với các chuẩn khác nhau, giờ đây người thiết kế chỉ phải thiết kế một bo mạch duy nhất với nhiều cấu hình được lưu trữ trong bộ nhớ hệ thống ACE CF. Mỗi bo mạch có thể chọn các cấu hình phù hợp với

các chuẩn khác nhau bằng cách khởi tạo giá trị mặc định tương ứng được lưu trong bộ nhớ ACE. Hệ thống còn cho phép lưu nhiều cấu hình cho một thiết kế trong một hệ thống ACE CF đơn. Ví dụ như trong quá trình thiết kế mẫu, người thiết kế có thể lưu các cấu hình hoạt động, cấu hình kiểm tra và cấu hình gỡ rối trong bộ nhớ ACE, đồng thời có thể chọn các cấu hình khác để chạy thử bản thiết kế của mình.

Để hỗ trợ quản lý nhiều luồng bit và tích hợp điều khiển cấu hình FPGA với hoạt động của hệ thống, hệ thống ACE có một cổng vi xử lí trong hệ thống. Cổng này cho phép bộ xử lí của hệ thống thay đổi cấu hình mặc định, cấu hình lại trigơ, cấu hình lại từng FPGA hoặc một nhóm FPGA, truy nhập vào các file không cấu hình được lưu trong khối CompactFlash hoặc dùng khối CompactFlash làm bộ nhớ chung cho hệ thống.

Với các FPGA có trung tâm xử lí kèm theo, hệ thống ACE CF cung cấp giải pháp 3 trong 1 để quản lý phần cứng và phần mềm. Hệ thống ACE CF có thể cấu hình khung FPGA, khởi tạo trung tâm vi xử lí và cung cấp các ứng dụng phần mềm cho trung tâm này nếu cần mà không phải thêm bất cứ thiết bị phần cứng nào.

+ ***Mật độ:*** Với mật độ logic cao chưa từng thấy (trên 8Gbit), một Hệ thống ACE CF có thể cấu hình cho hàng trăm FPGA và có thể thay thế cho các mảng EEPROM cấu hình. Người thiết kế có thể lưu một số lượng lớn các thiết kế khác nhau cho một mảng FPGA trong cùng một khối nhớ. Hệ thống ACE CF sử dụng hệ thống file FAT (File Allocation Table: bảng sắp xếp file tiêu chuẩn), do đó người thiết kế có thể lưu cả những file không ở dạng luồng bit hoặc sử dụng bộ nhớ thừa làm bộ nhớ chuẩn cho hệ thống.

+ ***Khả năng quản lý tập trung:*** Hệ thống ACE CF được thiết kế để quản lý cấu hình theo yêu cầu. Một hệ thống ACE CF có thể cấu hình cho một hoặc nhiều bo mạch FPGA. Khả năng tập trung cho

phép đơn giản hóa quá trình quản lý và nâng cấp cấu hình. Để thay đổi hay nâng cấp cấu hình của một hệ thống, người thiết kế có thể vào khối nhớ, thực hiện các thay đổi cần thiết trên màn hình máy tính, chỉnh lại nội dung trong hệ thống qua cổng vi xử lí; hoặc tải cấu hình mới về qua mạng, sử dụng IRL.

8.4.3 Chọn công cụ phần mềm phù hợp

Xilinx đã cung cấp các công cụ thiết kế điện tử hoàn chỉnh, cho phép thực hiện thiết kế trên các thiết bị logic khả trình của Xilinx. Các công cụ này kết hợp công nghệ tiên tiến với giao diện đồ họa linh hoạt, dễ sử dụng để người thiết kế có được thiết kế tối ưu. Bộ công cụ phần mềm hiện đang được sử dụng rộng rãi là ISE với phiên bản mới nhất là 11.1 (27/4/2009).

Xilinx cũng cung cấp ISE dưới dạng các gói phần mềm có cấu hình khác nhau với giá thành khác nhau:

+ ISE WebPACK: Bản miễn phí có thể dùng để thiết kế cho tất cả các họ CPLD của Xilinx và một số loại FPGA dung lượng thấp và trung bình.

+ Gói phần mềm cơ bản BASEX: Có thể thiết kế cho các loại chip sau:

Virtex-4, FPGA LX15, LX25, SX25, FX12, Spartan-3 FPGA lên đến 1.500 ngàn cổng và tất cả các họ CPLD.

+ Gói phần mềm Foundation: Có thể thiết kế cho tất cả các loại FPGA và CPLD của Xilinx

Ngoài ra Xilinx còn phát triển các bộ công cụ phần mềm tiện ích khác như System Generator hỗ trợ cho các thiết kế DSP (Digital Signal Processor: Bộ xử lý tín hiệu số) hay EDK (Embedded Development Kit: Bộ phần mềm phát triển hệ thống) hỗ trợ cho các thiết kế nhúng.

ISE hay được dùng kết hợp với phần mềm mô phỏng ModelSim của Mentor Graphics phiên bản XE được phát triển riêng hỗ trợ cho các họ CPLD/FPGA của Xilinx.

8.5 LƯU ĐỒ THIẾT KẾ CHO CPLD/FPGA

8.5.1 Lưu đồ thiết kế cho CPLD

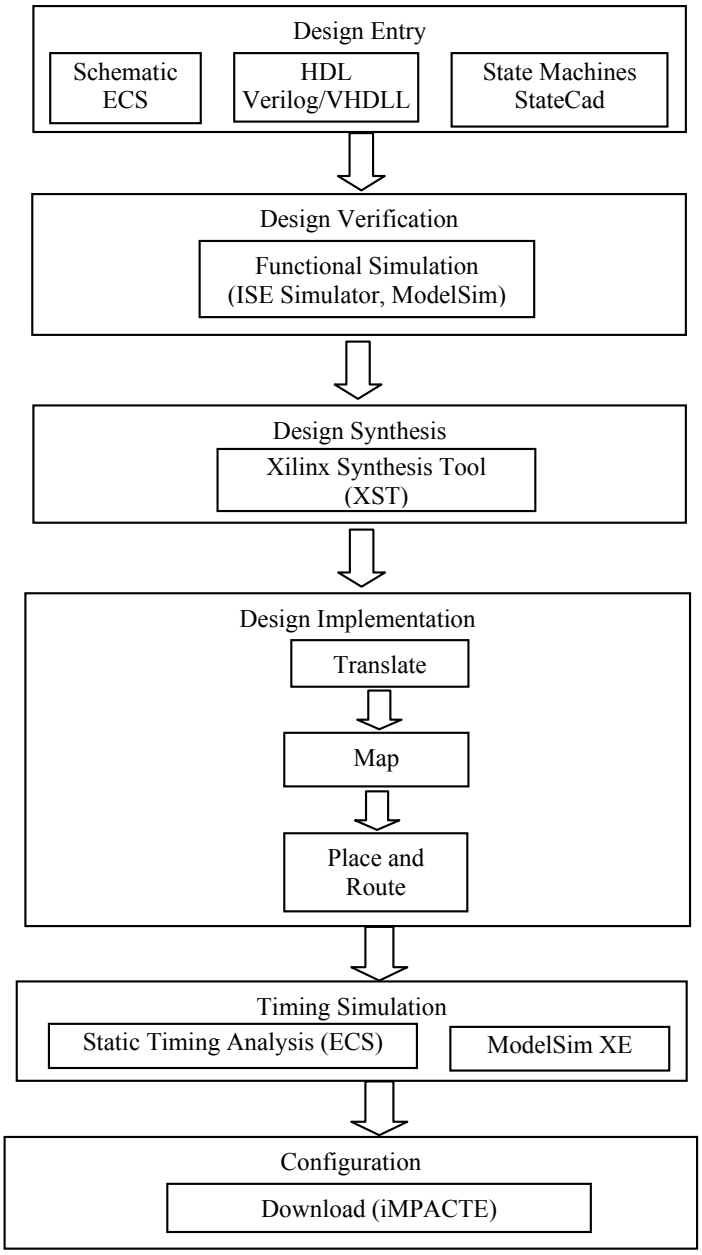
Quá trình thiết kế cho CPLD chủ yếu là thực hiện trên các công cụ phần mềm, lưu đồ thiết kế chung cho CPLD (Ví dụ sử dụng phần mềm ISE) như hình 8.8, bao gồm các bước như sau:

Bước 1: Nhập thiết kế (Design Entry)

Đây là bước đầu tiên và quan trọng nhất của quá trình thiết kế cho CPLD. Các công cụ thiết kế cho phép nhập thiết kế theo các cách sau:

- *Nhập thiết kế theo sơ đồ nguyên lý Schematic*, người thiết kế sử dụng các môđun đã có sẵn trong thư viện Schematic để ghép nối chúng với nhau tạo thành bản thiết kế theo yêu cầu, cách này có thể thực hiện thiết kế nhanh nhưng sẽ rất khó khăn và không tối ưu tài nguyên của CPLD khi thiết kế phức tạp và thiết kế không sử dụng sang công cụ thiết kế CPLD của các hãng khác. Từ sơ đồ nguyên lý đã thiết kế được công cụ phần mềm sẽ chuyển đổi sang file ngôn ngữ mô tả phần cứng HDL, mà phổ biến là VHDL hoặc Verilog.

- *Nhập thiết kế sử dụng ngôn ngữ mô tả phần cứng HDL* (VHDL, Verilog, ABEL, AHDL...). Người thiết kế có thể sử dụng chương trình soạn thảo để thực hiện việc mô tả toàn bộ bản thiết kế của mình dưới dạng ngôn ngữ HDL nào đó mà công cụ thiết kế có thể tổng hợp được. Có rất nhiều phương pháp mô tả, mức độ trừu tượng khác nhau khi thiết kế, mỗi cách mô tả khác nhau có thể tạo ra một cấu trúc mạch khác nhau trong CPLD mặc dù chúng có cùng chức năng.



Hình 8.8: Lưu đồ thiết kế CPLD

Do đó người thiết kế cần thực hiện phân tích bài toán, tìm hiểu tài nguyên, cấu trúc của CPLD, yêu cầu về thời gian thiết kế để sử dụng kiểu mô tả, mức độ trừu tượng phù hợp vừa đảm bảo yêu cầu về thời gian thiết kế vừa tối ưu được việc sử dụng tài nguyên của CPLD.

- *Nhập thiết kế dưới dạng sơ đồ*: Công cụ thiết kế còn cho phép nhập thiết kế vào dưới dạng sơ đồ mà điển hình là đồ hình trạng thái, sau đó chúng cũng được chuyển đổi sang HDL.

Việc nhập thiết kế rất linh hoạt, có thể sử dụng cả 3 cách trên để thực hiện các phần khác nhau của thiết kế.

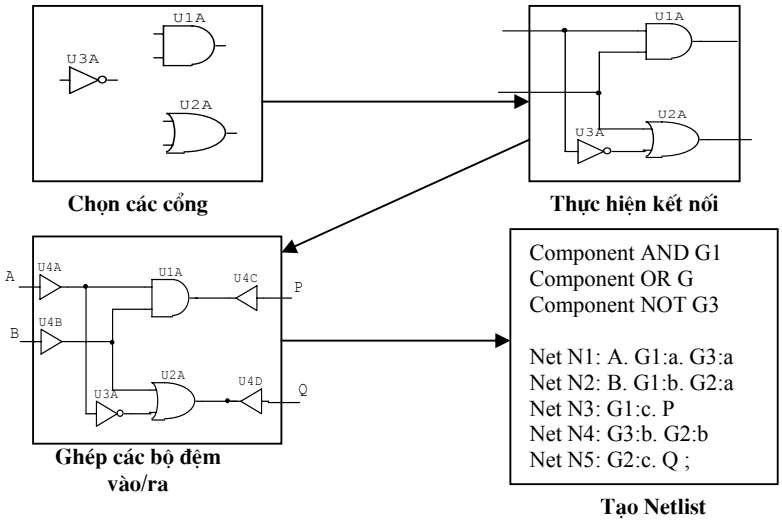
Bước 2: Kiểm tra, mô phỏng thiết kế (Design Verification)

Thực hiện kiểm tra, mô phỏng chức năng hoạt động của thiết kế HDL đã tạo ra ở trên. Các công cụ thiết kế đều hỗ trợ việc mô phỏng chức năng hoạt động của bản thiết kế HDL theo mô hình hoạt động (Behavioral Model), mức độ mô phỏng này độc lập với loại CPLD đã được lựa chọn. Bước này có thể không cần phải thực hiện trong khi thiết kế.

Bước 3: Tổng hợp thiết kế (Design Synthesis)

Sau khi hoàn thành mô phỏng thiết kế, bước tổng hợp tiếp theo có nhiệm vụ chuyển thiết kế dưới dạng file vản bản HDL thành dạng file Netlist, thực hiện mô tả mạch thực ở mức thấp dưới dạng cổng logic và kết nối giữa chúng với nhau. Có thể sử dụng các công cụ tổng hợp của các hãng khác nhau.

Mỗi công cụ có thể tạo ra file Netlist theo định dạng riêng (ví dụ của công cụ thổng hợp XST của Xilinx tạo ra: XNF-Xilinx Netlist Format) nhưng người thiết kế có thể đặt lựa chọn để tạo ra file Netlist dưới dạng định dạng chuẩn EDIF (Electronic Digital Interchange Format) mà tất cả các công cụ có thể hiểu được.



Hình 8.9: Ví dụ tổng hợp ra file Netlist

Bước 4: Thực hiện thiết kế (Design Implementation)

Sau khi có file Netlist, bước tiếp theo là thực hiện thiết kế, nghĩa là xây dựng cấu hình cho CPLD. Bước này sử dụng file Netlist và file ràng buộc "constraints file" (mô tả các nguyên tắc thiết kế, các ràng buộc về vật lý như gán vị trí cho các đầu vào/ra trên chip, các ràng buộc về tốc độ, thời gian, tần số…) để tạo thiết kế sử dụng tài nguyên có sẵn của CPLD. Bước này bao gồm các bước: Translate (biên dịch), Map (Phân bố bản thiết kế vào chip), Place and Route (Định vị và định tuyến kết nối).

*** Translate** (biên dịch)

Bước này nhằm thực hiện kiểm tra thiết kế và đảm bảo file Netlist phù hợp với kiến trúc của CPLD đã chọn, kiểm tra file ràng buộc "constraints file" của người sử dụng để phát hiện các lỗi mâu thuẫn với tham số của CPLD đã chọn. Biên dịch thường bao gồm các quá trình: tối ưu hoá, biên dịch thành các thành phần vật lý của cấu

kiện; kiểm tra ràng buộc thiết kế. Khi kết thúc bước biên dịch, sẽ có một bản báo cáo về các chương trình được sử dụng, danh sách các cổng I/O và các cấu kiện được sử dụng trong thiết kế, nhờ đó người thiết kế sẽ lựa chọn được phương án thiết kế tối ưu.

*** Map**

Tạo bản phân bố thiết kế tới các tài nguyên cụ thể trong CPLD. Nếu thiết kế quá lớn so với thiết bị được chọn, quy trình này không thể hoàn thành nhiệm vụ của mình. Quá trình Map có các tham số ràng buộc của thiết kế, ví dụ như tham số tốc độ, thời gian của thiết kế và đôi khi quyết định gắn thêm các thành phần logic để đáp ứng các yêu cầu về thời gian. Map có khả năng thay đổi thiết kế xung quanh các bảng ánh xạ để tạo khả năng thực hiện tốt nhất cho thiết kế. Quy trình này được thực hiện hoàn toàn tự động và cần rất ít tác động đầu vào từ người sử dụng. Bước này nhằm đưa mạch thiết kế vào một thiết bị cụ thể. Bước này cũng tạo ra báo cáo xác nhận các tài nguyên được sử dụng trong chip, mô tả chính xác các phần trong thiết kế được đặt ở vị trí nào trong chip thực tế.

*** Place and Route** (PAR - Định vị và định tuyến kết nối)

Place là quá trình lựa chọn vị trí phù hợp của mỗi khối chức năng trong thiết kế và đưa các cổng logic của phần đó vào các khối logic hay các môđun cụ thể trong CPLD trên cơ sở tối ưu việc kết nối và đảm bảo về các ràng buộc về thời gian. Những phần logic hoạt động tốc độ cao sẽ được xếp cạnh nhau để giảm độ dài đường kết nối. Route là quá trình tạo liên kết vật lý giữa các khối logic. Hầu hết các nhà sản xuất cung cấp công cụ Place and Route tự động cho người sử dụng. Ngoài công cụ tự động, người thiết kế có thể tự Place and Route trong khi thiết kế. Nhà sản xuất cũng cung cấp các công cụ, như "Floorplanner", để nâng cao hiệu suất quá trình Place and Route do người thiết kế thực hiện so với quá trình tự động.

Place and Route là quá trình phức tạp, do đó nó chiếm thời gian nhiều nhất. Tuy nhiên, bước này chỉ có thể hoạt động tốt nếu Chip đã chọn đáp ứng đủ các tuyến liên kết cho thiết kế. Nếu không, người thiết kế sẽ phải chọn chip có dung lượng lớn hơn. Sau bước này tạo ra được file cấu hình *.jed có thể được nạp vào cho CPLD.

Bước 5: Timing Simulation (Mô phỏng định thời)

Sau bước Place and Route người thiết kế có thể thực hiện mô phỏng thiết kế ở mức cổng logic đã được định vị trí và định tuyến trên CPLD, phần mềm sử dụng file cấu hình đã được tạo ra và kết hợp với thư viện về mô hình thời gian của các họ CPLD (ví dụ ISE của Xilinx thì dùng thư viện VITAL), để thực hiên mô phỏng hoạt động của thiết kế mà có tính đến các tham số thời gian trễ, thời gian thiết lập… của các cổng logic trong CPLD. Bước này rất quan trọng với những thiết kế phức tạp, tốc độ lớn.

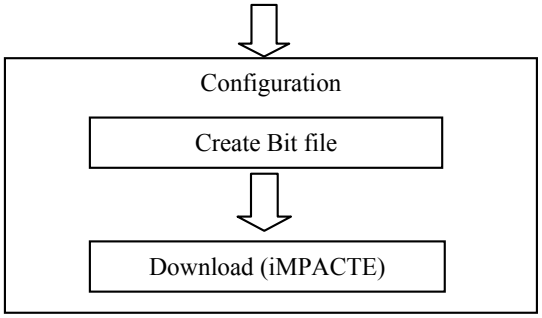
Bước 6: Configuration (Cấu hình) hay bước nạp cấu hình (Download)

Gọi chương trình điều khiển việc nạp cấu hình, thực kết nối thiết bị nạp (cáp nạp) đến CPLD và nạp file cấu hình cho CPLD. Với CPLD của hãng Xilinx, quá trình lập trình có thể thực hiện ngay trong hệ thống nhờ công cụ JTAG hoặc sử dụng bộ lập trình thiết bị chuyên dùng, ví dụ như công cụ JTAG Data I/O, theo chuẩn IEEE/ANSI 1149.1_1190. Công cụ JTAG là một bộ các nguyên tắc thiết kế, hỗ trợ quá trình kiểm tra, lập trình cho thiết bị và gỡ rối trên chip, trên bo mạch và trên hệ thống. Khả năng lập trình trên hệ thống là ưu điểm của CPLD, cho phép lập trình lại CPLD đã được hàn trực tiếp lên PCB. Nếu có thay đổi trong thiết kế, sẽ không phải tháo thiết bị ra khỏi bo mạch, mà đơn giản chỉ phải lập trình lại trên hệ thống.

8.5.2 Lưu đồ thiết kế cho FPGA

Lưu đồ thiết kế cho FPGA cũng tương tự như lưu đồ thiết kế cho CPLD, chỉ khác ở bước cuối cùng - bước Cấu hình cho FPGA. Ở bước

này, đối với FPGA có thêm bước "Create Bit file" để tạo ra file "luồng bit" để nạp vào bộ nhớ cấu hình trong FPGA thường là bộ nhớ tạm thời như SRAM. Dòng bit được nạp mang tất cả thông tin để định nghĩa các hàm logic và các liên kết trong thiết kế. Mỗi thiết kế khác nhau có một dòng bit khác nhau. Các thiết bị SRAM mất toàn bộ thông tin mỗi khi ngắt nguồn, do đó khi cần thiết phải nạp dòng bit cấu hình này vào trong EEPROM (thường sử dụng EEPROM nối tiếp). Mỗi khi thiết bị được bật nguồn file cấu hình từ EEPROM sẽ được nạp tự động vào bộ nhớ SRAM của FPGA và FPGA hoạt động theo cấu hình đã được nạp đó.



Hình 810: Lưu đồ thiết kế FPGA

TÓM TẮT

Chương này đã trình bày tổng quan về các công nghệ logic số, đặc biệt là công nghệ logic khả trình với hai họ cấu kiện phổ biến hiện nay là CPLD và FPGA. Giới thiệu các phương pháp lập trình cấu hình cho CPLD/FPGA, cũng như những vấn đề cần chú ý khi thiết kế ứng dụng dùng CPLD/FPGA. Lưu đồ thiết kế cho CPLD/FPGA cũng được trình bày khá chi tiết trong chương này giúp bạn đọc hiểu được toàn bộ quá trình cũng như các bước cơ bản cần được thực hiện để thiết lập cấu hình cho CPLD/FPGA theo yêu cầu của người thiết kế.

CÂU HỎI ÔN TẬP

1. Trình bày các công nghệ logic số cơ bản
2. Nêu đặc điểm của các phương pháp xây dựng mạch số sử dụng công nghệ logic chuẩn, logic khả trình, ASIC?
3. Trình bày cấu trúc của PAL, PLA, lấy ví dụ về xây dựng hàm logic sử dụng các cấu trúc này.
4. Trình bày cấu tạo chung của CPLD, FPGA và so sánh.
5. Trình bày các phương pháp chung để thiết kế cho CPLD/FPGA
6. Trình bày các yêu cầu chung khi thiết kế với CPLD/FPGA.
7. Trình bày lưu đồ thiết kế cho CPLD/FPGA.

Chương 9

NGÔN NGỮ MÔ TẢ PHẦN CỨNG VHDL

GIỚI THIỆU

Ngày nay, các mạch tích hợp có mật độ ngày càng lớn và ngày càng thực hiện được nhiều chức năng, hệ thống điện tử nói chung, hệ thống số nói riêng ngày trở nên phức tạp. Do đó, vấn đề thiết kế mạch càng trở nên phức tạp. Những phương pháp truyền thống như dùng phương pháp tối thiểu hoá hàm Boolean hay dùng sơ đồ các phần tử không còn đáp ứng được các yêu cầu đặt ra khi thiết kế. Nhược điểm lớn nhất của các phương pháp này là chúng chỉ mô tả được hệ thống dưới dạng mạng nối các phần tử với nhau. Người thiết kế cần phải đi qua hai bước thực hiện hoàn toàn thủ công: đó là chuyển từ các yêu cầu về chức năng của hệ thống sang biểu diễn theo dạng hàm Boolean, sau các bước tối thiểu hoá hàm này ta lại phải chuyển từ hàm Boolean sang sơ đồ mạch của hệ thống. Cũng tương tự khi phân tích một hệ thống người phân tích cần phải phân tích sơ đồ mạch của hệ thống, rồi chuyển nó thành các hàm Boolean, sau đó mới lập lại các chức năng, hoạt động của hệ thống. Tất cả các bước nói trên hoàn toàn phải thực hiện thủ công không có bất kỳ sự trợ giúp nào của máy tính. Người thiết kế chỉ có thể sử dụng máy tính làm công cụ hỗ trợ trong việc vẽ sơ đồ mạch của hệ thống và chuyển từ sơ đồ mạch sang công cụ tổng hợp mạch vật lý dùng công cụ tổng hợp "Synthesis". Một nhược điểm khác nữa của phương pháp thiết kế truyền thống là sự giới hạn về độ

phức tạp của hệ thống được thiết kế. Phương pháp dùng hàm Boolean chỉ có thể dùng để thiết kế hệ thống lớn nhất biểu diễn bởi vài trăm hàm. Còn phương pháp dựa trên sơ đồ chỉ có thể dùng để thiết kế hệ thống lớn nhất chứa khoảng vài nghìn phần tử.

Nhờ sự trợ giúp đắc lực của máy tính, phương pháp thiết kế, thử nghiệm, phân tích các hệ thống số sử dụng các ngôn ngữ mô tả phần cứng nổi bật lên với các ưu điểm hơn hẳn và sẽ dần thay thế các phương pháp truyền thống. Sự ra đời của ngôn ngữ mô phỏng phần cứng đã giải quyết được rất nhiều nhược điểm lớn của các phương pháp thiết kế trước đây: Nếu các phương pháp cũ đòi hỏi phải chuyển đổi từ mô tả hệ thống (các chỉ tiêu về chức năng) sang tập hợp các hàm logic bằng tay thì bước chuyển đó hoàn toàn không cần thiết khi dùng HDL. Hầu hết các công cụ thiết kế dùng ngôn ngữ mô phỏng phần cứng đều cho phép sử dụng đồ hình trạng thái cho các hệ thống tuần tự cũng như cho phép sử dụng bảng chân lý cho hệ thống mạch tích hợp. Việc chuyển đổi từ các biểu đồ trạng thái hay bảng chân lý sang mã ngôn ngữ mô phỏng phần cứng được thực hiện hoàn toàn tự động nhờ phần mềm thiết kế trên máy tính.

Nhờ tính dễ kiểm tra thử nghiệm hệ thống trong suốt quá trình thiết kế mà người thiết kế có thể dễ dàng phát hiện các lỗi thiết kế ngay từ những giai đoạn đầu, giai đoạn chưa đưa vào sản xuất thử, do đó tiết kiệm được lượng chi phí đáng kể bởi từ ý tưởng thiết kế đến tạo ra sản phẩm đúng như mong muốn là một việc rất khó tránh khỏi những khó khăn, thất bại.

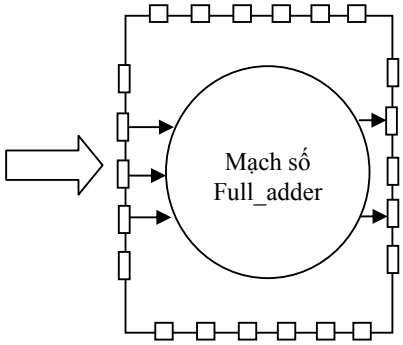
Khi mọi lĩnh vực của khoa học đều phát triển không ngừng thì sự phức tạp của hệ thống điện tử cũng ngày một tăng theo và gần như không thể tiến hành thiết kế thủ công mà không có sự trợ giúp của các loại máy tính hiện đại. Ngày nay, ngôn ngữ mô tả phần cứng HDL được dùng nhiều để thiết kế cho các thiết bị logic lập trình được PLD

từ loại đơn giản đến các loại phức tạp như FPGA, cũng như thiết kế hệ thống VLSI.

Trong toàn bộ lưu đồ thiết kế cho CPLD hoặc FPGA, bước nhập thiết kế là bước quan trọng và tốn nhiều công sức nhất, nó quyết định phần lớn đến kết quả của công việc thiết kế. Các công cụ thiết kế hỗ trợ nhiều phương pháp nhập thiết kế khác nhau, tuy nhiên phương pháp nhập thiết kế dùng ngôn ngữ mô tả phần cứng HDL là ưu việt hơn cả và được sử dụng chủ yếu trong quá trình thiết kế số nói chung và thiết kế cho CPLD/FPGA nói riêng. Hiện nay có nhiều ngôn ngữ HDL được sử dụng (như VHDL, Verilog, ABEL…), tuy nhiên trong giáo trình này chỉ giới thiệu phương pháp thiết kế dùng ngôn ngữ VHDL và giới thiệu những đặc điểm của VHDL khiến nó được trở thành một ngôn ngữ HDL không những được giảng dạy và sử dụng ở nhiều trường đại học trên thế giới mà còn được sử dụng bởi hãng điện tử lớn.

Tóm lại người sử dụng có thể dùng VHDL để mô tả cấu trúc và hoạt động của một mạch số, sau đó mã VHDL này có thể dùng để mô phỏng hoạt động của mạch số trên máy tính hay được tổng hợp thành file netlist rồi được ánh xạ đến một phần cứng thực dùng công nghệ logic số nào đó để thực hiện hệ mạch số như đã mô tả. Ví dụ dưới đây là mô tả VHDL cho mạch cộng toàn tổng.

```
-- Mã VHDL mô tả cấu trúc,
hoạt động của mạch số
ENTITY full_adder IS
PORT (a, b, cin: IN BIT;
 s, cout: OUT BIT);
END full_adder;
ARCHITECTURE dataflow OF
full_adder IS
BEGIN
s <= a XOR b XOR cin;
cout <= (a AND b) OR (a
AND cin) OR
 (b AND cin);
END dataflow;
```



9.1. LỊCH SỬ PHÁT TRIỂN CỦA VHDL

VHDL là ngôn ngữ mô tả phần cứng cho các mạch tích hợp tốc độ rất cao, là một loại ngôn ngữ mô tả phần cứng được phát triển dùng cho trương trình VHSIC (Very High Speed Intergrated Circuit: Mạch tích hợp tốc độ rất cao) của Bộ Quốc phòng Mỹ. Mục tiêu của việc phát triển VHDL là có được một ngôn ngữ mô phỏng phần cứng tiêu chuẩn và thống nhất cho phép thử nghiệm các hệ thống số nhanh hơn cũng như cho phép dễ dàng đưa các hệ thống đó vào ứng dụng trong thực tế. Ngôn ngữ VHDL được ba công ty Intermetics, IBM và Texas Instruments bắt đầu nghiên cứu phát triển vào tháng 7 năm 1983. Phiên bản đầu tiên được công bố vào tháng 8 năm1985. Sau đó VHDL được đề xuất để tổ chức IEEE xem xét thành một tiêu chuẩn chung. Năm 1987 IEEE đã đưa ra tiêu chuẩn về VHDL đâu tiên (tiêu chuẩn IEEE-1076-1987). Từ đó các phiển bản tiếp theo của tiêu chuẩn đã ra đời. Đa số các phần mềm thiết kế hiện nay đều hỗ trợ VHDL.

Tiêu chuẩn IEEE-1076-1987 quy định:

- Các kiểu dữ liệu cơ bản: số học (integer, real), kiểu logic (bit, boolean), kiểu ký tự (character), kiểu thời gian (time), mảng bit

(bit_vector), mảng ký tự (string), cùng các phép toán với các kiểu dữ liệu đó.

- Các cấu trúc lệnh tuần tự và song song.

- Các đơn vị thiết kế…

Để mô tả được các tín hiệu logic nhiều trạng thái, IEEE đã bổ sung tiêu chuẩn IEEE 1164 quy định kiểu dữ liệu logic có tới 9 trạng thái (*std_ulogic và kiểu mảng std_ulogic_vector*) cùng các phép toán với kiểu dữ liệu đó.

Năm 1993, IEEE đưa ra phiên bản IEEE-1016-1993 bổ sung thêm nhiều cấu trúc, cú pháp cho phép việc đặt tên linh hoạt hơn, bổ sung thêm lệnh XNOR, kiểu ký tự xuất ra máy in theo tiêu chuẩn ISO-8859-1…

Phiên bản tiêu chuẩn năm 2000 và 2002 bổ sung một số ý tưởng về VHDL hướng đối tượng (giống như C++) và bỏ một số hạn chế trong luật đặt tên cổng "port".

Ngoài tiêu chuẩn IEEE 1164, IEEE còn đưa ra một số tiêu chuẩn bổ sung cho VHDL như:

+ IEEE 1076.1 (VHDL-AMS) tiêu chuẩn VHDL mở rộng cho mô tả mạch tương tự và mạch tín hiệu hỗn hợp số - tương tự.

+ IEEE 1076.2 bổ sung một số kiểu dữ liệu phức (complex), thực (real).

+ IEEE 1076.3 bổ sung một số kiểu dữ liệu có dấu (signed) và không dấu (unsigned).

+ IEEE 1076.4 - VITAL ASIC (Application Specific Integrated Circuit) Modeling Specification bổ sung thư viện VITAL quy định đặc tính của mô hình mạch ASIC.

+ IEEE 1076.6 bổ sung thư viện cho phép tổng hợp hệ thống số theo mô hình luồng dữ liệu RTL

Năm 2006, Ủy ban kỹ thuật VHDL của Accellera được sự ủy nhiệm của IEEE để thực hiện các cập nhật tiếp theo cho tiêu chuẩn đã đưa phiên bản VHDL 3.0 (tiêu chuẩn VHDL-2006) ngoài việc giữ lại đầy đủ phiên bản tiêu chuẩn cũ còn thực hiện nhiều mở rộng cho phép việc viết và quản lý mã VHDL dễ dàng hơn. Những thay đổi quan trọng nhất là kết hợp (IEEE 1164, IEEE 1076.2, IEEE 1076.3) thành tiêu chuẩn chung IEEE 1076 -2006, mở rộng cú pháp linh hoạt hơn cho cấu trúc lệnh "case" và lệnh "generate", kết hợp VHPI (VHPI - interface to C/C++) và tập con của PSL (Property Specification Language). Những thay đổi này nâng cao chất lượng của các mã VHLD cho phép tổng hợp, thực hiện viết testbench linh hoạt hơn, cho phép sử dụng rộng rãi hơn việc sử dụng VHDL cho việc mô tả mức hệ thống.

Năm 2008, Accellera đã đưa ra phiên bản VHDL 4.0 (tiêu chuẩn VHDL 2008), đã giải quyết hơn 90 vấn đề còn tồn tại của phiên bản VHDL 2006 và thêm vào các kiểu generic nâng cấp và phiên bản này dự kiến được đưa ra dưới tiêu chuẩn IEEE 1076-2008.

Như vậy VHDL luôn luôn được chuẩn hóa, bổ sung, thay đổi, hoàn thiện… để nó có thể đáp ứng được sự phức tạp, yêu cầu ngày càng cao của hệ thống điện tử nói chung, hệ thống số nói riêng.

Trong chương này các mã VHDL được sử dụng chủ yếu là theo tiêu chuẩn IEEE-1076-2002.

9.2 NHỮNG ƯU ĐIỂM CỦA VHDL

VHDL được phát triển để giải quyết các khó khăn trong việc mô phỏng, thiết kế, phát triển, thay đổi và lập tài liệu cho các hệ thống số. Như ta đã biết, một hệ thống số có rất nhiều tài liệu mô tả. Để có thể vận hành bảo trì sửa chữa một hệ thống ta cần tìm hiểu kỹ lưỡng tài liệu đó. Với một ngôn ngữ mô phỏng phần cứng tốt việc xem xét các tài liệu mô tả trở nên dễ dàng hơn vì bộ tài liệu đó có thể được thực thi

để mô phỏng hoạt động của hệ thống. Như thế ta có thể xem xét toàn bộ các phần tử của hệ thống hoạt động trong một mô hình thống nhất.

VHDL được phát triển như một ngôn ngữ độc lập không gắn với bất kỳ một phương pháp thiết kế, một bộ mô tả hay công nghệ phần cứng nào. Người thiết kế có thể tự do lựa chọn công nghệ, phương pháp thiết kế trong khi chỉ sử dụng một ngôn ngữ duy nhất. Và khi đem so sánh với các ngôn ngữ mô phỏng phần cứng khác ta thấy VHDL có một số ưu điểm hơn hẳn các ngôn ngữ khác:

+ Thứ nhất là tính công cộng: VHDL được phát triển dưới sự bảo trợ của chính phủ Mỹ và hiện nay là một tiêu chuẩn của IEEE. VHDL được sự hỗ trợ của nhiều nhà sản xuất thiết bị cũng như nhiều nhà cung cấp công cụ thiết kế mô phỏng hệ thống.

+ Thứ hai là khả năng hỗ trợ nhiều công nghệ và phương pháp thiết kế. VHDL cho phép thiết kế bằng nhiều phương pháp, ví dụ phương pháp thiết kế từ trên xuống hay từ dưới lên dựa vào các thư viện sẵn có. VHDL cũng hỗ trợ cho nhiều loại công cụ thiết kế mạch như sử dụng công nghệ đồng bộ hay không đồng bộ, sử dụng ma trận lập trình được hay sử dụng mảng ngẫu nhiên.

+ Thứ ba là tính độc lập với công nghệ: VHDL hoàn toàn độc lập với công nghệ chế tạo phần cứng. Một mô tả hệ thống dùng VHDL thiết kế ở mức cổng logic có thể được chuyển thành các bản tích hợp mạch khác nhau tuỳ thuộc công nghệ chế tạo phần cứng mới ra đời nó có thể được áp dụng ngay cho các hệ thống đã thiết kế.

+ Thứ tư là khả năng mô tả mở rộng: VHDL cho phép mô tả hoạt động của phần cứng từ mức hệ thống cho đến mức cổng logic. VHDL có khả năng mô tả hoạt động của hệ thống trên nhiều mức nhưng chỉ sử dụng một cú pháp chặt chẽ thống nhất cho mọi mức. Như thế ta có thể mô phỏng một bản thiết kế bao gồm cả các hệ con được mô tả chi tiết.

+ Thứ năm là khả năng trao đổi kết quả: Vì VHDL là một tiêu chuẩn được chấp nhận, nên một mô hình VHDL có thể chạy trên mọi bộ mô tả đáp ứng được tiêu chuẩn VHDL. Các kết quả mô tả hệ thống có thể được trao đổi giữa các nhà thiết kế sử dụng công cụ thiết kế khác nhau nhưng cùng tuân theo tiêu chuẩn VHDL. Cũng như một nhóm thiết kế có thể trao đổi mô tả mức cao của các hệ thống con trong một hệ thống lớn (trong đó các hệ con đó được thiết kế độc lập).

+ Thứ sáu là khả năng hỗ trợ thiết kế mức lớn và khả năng sử dụng lại các thiết kế: VHDL được phát triển như một ngôn ngữ lập trình bậc cao, vì vậy nó có thể được sử dụng để thiết kế một hệ thống lớn với sự tham gia của một nhóm nhiều người. Bên trong ngôn ngữ VHDL có nhiều tính năng hỗ trợ việc quản lý, thử nghiệm và chia sẻ thiết kế. Và nó cũng cho phép dùng lại các phần đã có sẵn.

9.3 CẤU TRÚC NGÔN NGỮ CỦA VHDL

VHDL là ngôn ngữ cho phép mô tả các thiết bị phần cứng số trừu tượng, nó không dựa vào công nghệ thiết bị phần cứng số hay phương pháp được sử dụng để thiết kế thiết bị số cụ thể nào đó. Những khái niệm, mô hình trừu tượng của thiết bị phần cứng số được đưa ra như là nền tảng của ngôn ngữ. Do đó dùng VHDL cho phép mô tả được hầu hết các hệ thống phần cứng số. Các mô hình trừu tượng gồm:

Mô hình hành vi (a Model of Behavior).

Mô hình thời gian (a Model of Time).

Mô hình cấu trúc (a Model of Structure).

Để thực hiện mô tả cho một hệ thống số nào đó cần thực hiện theo các bước như sau:

- Phân tích yêu cầu của hệ thống số cần phải thiết kế hoặc cần phải mô tả.

- Phân tách hệ thống thành những khối con.

- Xác định mô hình mô tả phù hợp cho mỗi khối con hoặc cho cả hệ thống.

- Sử dụng ngôn ngữ VHDL để mô tả hệ thống số theo các mô hình đã xác định.

Như vậy việc nắm chắc cấu trúc, cú pháp, các mô hình mô tả của ngôn ngữ là rất quan trọng, quyết định chủ yếu đến thành công trong việc mô tả hệ thống số cần thiết kế.

VHDL cũng có nhiều điểm giống như một ngôn ngữ lập trình bậc cao, có cấu trúc, có cú pháp riêng, có cách tổ chức chương trình, có từ khóa, có phương pháp biểu diễn số liệu riêng...

Cấu trúc ngôn ngữ cơ bản của VHDL gồm:

Đối tượng: Quy định các dạng tín hiệu cố định, tín hiệu, cổng vào/ra hay tín hiệu đệm…

Các kiểu dữ liệu: Quy định các kiểu dữ liệu có thể được dùng để gán cho mỗi đối tượng.

Các phép toán: Quy định các phép toán sử dụng cho mỗi loại dữ liệu.

Các đơn vị thiết kế: Các thành phần cơ bản cấu trúc lên một chương trình mã mô tả dùng VHDL.

Các cấu trúc lệnh tuần tự: Cấu trúc câu lệnh thực hiện theo tiến trình tuần tự, thường dùng mô tả các cấu trúc mạch tuần tự của mạch số.

Các cấu trúc lệnh song song: Cấu trúc câu lệnh thực hiện song song, thường dùng mô tả các cấu trúc mạch tổ hợp.

Chú ý:

- Trong các đoạn mã mô tả VHDL trong chương các từ khóa đều được in đậm.

- Trong VHDL không phân biệt chữ hoa, chữ thường.

- Ghi chú trong VHDL dùng dấu "--" *(-- Ghi chú).*

9.3.1 Đối tượng trong VHDL

Trong ngôn ngữ VHDL gồm có 4 đối tượng là: tín hiệu **- signal**, biến **- variable**, hằng - **constant**, tham số chung – **generic.** Mỗi đối tượng được khai báo dựa vào từ khóa tương ứng và chúng có mục đích sử dụng khác nhau.

9.3.1.1 Tín hiệu – Signal

Là đối tượng để biểu diễn đường kết nối giữa các cổng vào/ra của thực thể (mạch số), giữa các cổng vào/ra của các khối thành phần phần cứng bên trong mạch số… Chúng là phương tiện truyền dữ liệu động giữa các thành phần của mạch số.

Tín hiệu có tính toàn cục rất cao, chúng có thể được khai báo trong **package** (tín hiệu toàn cục, được sử dụng bởi một số thực thể), khai báo trong thực thể - **Entity** (tín hiệu nội bộ dùng trong thực thể, có thể được tham chiếu bởi bất kỳ kiến trúc nào của thực thể đó), khai báo trong kiến trúc - **Architecture** (tín hiệu nội bộ dùng trong kiến trúc, có thể được sử dụng trong bất cứ cấu trúc lệnh nào trong kiến trúc). Các tín hiệu có thể được sử dụng nhưng không được khai báo trong tiến trình - **process**, trong thủ tục - **proceduce**, trong hàm - **function**, vì tiến trình và thủ tục, hàm là thành phần cơ sở của mô hình và chúng được coi như các hộp đen.

Cú pháp khai báo tín hiệu như sau:

```
Signal tên_tín_hiệu {,tên_tín_hiệu}:kiểu_dữ_liệu
[:=giá_trị_khởi_tạo];
```

```
Ví dụ: Signal a,b,c: Bit:='1'; -- Giá trị khởi tạo
là '1';
Signal y,reg: std_logic_vector(3 downto 0):="0000";
```

9.3.1.2 Biến - Variable

Là đối tượng cục bộ được sử dụng để chứa các kết quả trung gian. Biến chỉ được khai báo và sử dụng trong process và trong procedure và function.

Cú pháp khai báo của biến cũng tương tự như khai báo tín hiệu:

```
variable tên_biến {,tên_biến}: kiểu_dữ_liệu
[:=giá_trị_khởi_tạo];
```

```
Ví dụ: variable x: Bit:='1';
  variable Q: std_logic_vector(3 downto 0);
```

Nếu không được khởi tạo giá trị ban đầu biến sẽ nhận giá trị ban đầu là giá trị thấp nhất trong các giá trị thuộc miền xác định của kiểu dữ liệu.

Đối tượng tín hiệu - ***Signal*** cũng có thể chứa dữ liệu trung gian nhưng chúng lại không được sử dụng vì những lý do sau:

- Việc sử dụng biến hiệu quả hơn vì giá trị của biến được gán ngay lập tức trong ***process***, trong khi đó tín hiệu chỉ được lập kế hoạch để thực hiện và chỉ được cập nhật toàn bộ sau khi kết thúc process.

- Trong quá trình chạy mô phỏng hay tổng hợp, biến chiếm ít bộ nhớ hơn trong khi tín hiệu cần nhiều thông tin để có thể lập kế hoạch thực hiện cũng như để chứa các thuộc tính của tín hiệu.

- Sử dụng tín hiệu phải yêu cầu có lệnh ***wait*** để thực hiện đồng bộ phép gán tín hiệu với phép lặp thực hiện theo cách sử dụng quen thuộc.

9.3.1.3 Hằng - constant

Đối tượng hằng được gán cho các giá trị cụ thể của một kiểu dữ liệu khi được tạo ra và không đổi trong toàn bộ quá trình thực hiện. Hằng có thể dùng để mô tả cho tín hiệu không đổi (ví dụ tín hiệu GND, V_{CC},…). Hằng cũng có tính toàn cục giống như tín hiệu và có

thể được khai báo trong ***package, entity, architecture, proceduce, function, process...***

Cú pháp khai báo hằng:

```
constant tên_hằng {,tên_hằng}:
kiểu_dữ_liệu:=giá_trị_khởi_tạo;
```

Ví dụ:
```
constant GND: std_logic:='0';
constant PI: real:=3.1414;
constant datamemory: memory:= (('0','0','0','0'),
                               '0','0','0','1'),
                               '0','0','1','1'));
```

9.3.1.4 Tham số chung - Generic

Dùng để khai báo tham số cho mô hình mạch số và chỉ được khái báo trong phần ***Entity*** (Đề cập chi tiết trong phần 4.3.3).

Cú pháp khai báo tham số chung:

```
Generic(Tên_tham_số{,Tên_tham_số}:
        kiểu_dữ_liệu:=giá_trị_khởi_tạo);
```

Ví dụ:
```
entity Logic_AND is
 generic(delay: Time:=1ns);
  port(A, B: in std_logic;
 X: out std_logic);
end Logic_AND;
```

Tóm lại: Các đối tượng trong VHDL có mục đích sử dụng, phạm vi sử dụng khác nhau, nhưng chúng có cú pháp khai báo chung như sau:

```
Đối_tượng tên_đối_tượng: kiểu_dữ_liệu
{:=giá_trị_khởi_tạo}
```

Các đối tượng khi khai báo phải được xác định kiểu dữ liệu tương ứng. VDHL định nghĩa nhiều kiểu dữ liệu khác nhau để phù

hợp với việc mô tả, thiết kế, mô phỏng các hệ thống số khác nhau trong thực tế.

9.3.2 Kiểu dữ liệu trong VHDL

Trong VHDL có 4 dạng dữ liệu:

Vô hướng: gồm các dữ liệu có giá trị đơn như ***bit, boolean, integer, real, physical, character, std_logic và std_ulogic, enumerated*** (kiểu liệt kê)...

Kiểu ghép: các dữ liệu dưới dạng một nhóm các thành phần như mảng, bảng ghi (record). ***Bit_logic_vector, std_logic_vector*** và ***String*** đều là những dạng dữ liệu ghép đã được định nghĩa sẵn.

Mảng hai chiều (2-D Arrays): các dữ liệu có dạng mảng 2 chiều, được tạo nên từ 1 mảng của một mảng 1 chiều (hay một bản ghi).

Kiểu dữ liệu con (Subtypes): tập dữ liệu con của một dữ liệu đã có sẵn, được người dùng tự định nghĩa dựa trên những dạng có sẵn.

Các kiểu dữ liệu đã được định nghĩa trong gói dữ liệu chuẩn trong thư viện chuẩn Standard Library của VHDL là: ***bit, boolean, integer, real, physical, character, std_logic and std_ulogic, Bit_logic_vector, std_logic_vector và String*** và một số kiểu dữ liệu con. Do các kiểu dữ liệu trên đã được định nghĩa trong các thư viện chuẩn của VHD nên khi sử dụng các kiểu dữ liệu này chỉ cần khai báo thư viện tương ứng để có thể sử dụng chúng cũng như các phép toán tương ứng với chúng.

Cú pháp chung định nghĩa kiểu dữ liệu trong VHDL như sau:

```
Type Tên_kiểu is giới_hạn_giá_trị_của_kiểu
```

Các kiểu dữ liệu chuẩn được trình bày chi tiết sau đâu:

9.3.2.1 Kiểu vô hướng

- *Bit:* Kiểu liệt kê với 2 giá trị '0' và '1', đặc trưng cho 2 mức logic thấp và cao. Kiểu Bit đã được định nghĩa như sau:

```
Type Bit is ('0', '1');
```

- *Boolean:* Kiểu liệt kê với 2 giá trị false và true. Kiểu Boolean đã được định nghĩa như sau: `Type Boolean is (false, true);`

- *Integer:* Kiểu số nguyên với những giá trị dương hoặc âm, độ lớn mặc định là 32 bit với giới hạn giá trị: từ -2147483647 đến +2147483647. Khi sử dụng có thể giới hạn miền xác định theo giới hạn giảm dần dùng từ khóa ***downto*** hoặc tăng dần dùng từ khóa ***to***:

```
signal A: integer range 0 to 7;-- A số nguyên 3 bit
variable B: integer range 15 downto 0;-- B số nguyên 4 bit
signal C: integer range 15 downto -15;-- C số nguyên 5 bit
```

Các cách biểu diễn số nguyên dạng thập phân:

```
+ digit[underline]digit,
```

Ví dụ: `0, 1, 123_456_789, -123_5678…`

```
+ digit(E)digit,
```

Ví dụ:: `987E6 (=987.10^6)…`

Các cách biểu diễn dưới dạng cơ số xác định:

```
+ base#based_integer#[exponent],
 Ví dụ: 2#1100_0100#, 16#C4#, 4#301#E1, (=196).
```

- *Real:* Kiểu số thực có giới hạn từ -1.0E+38 đến 1.0E+38, khác với kiểu ***integer*** kiểu ***Real*** khi sử dụng thường được định nghĩa thành kiểu dữ liệu con riêng và có giới hạn miền xác định, ví dụ:

```
signal a: Real:=-123E-4;
type CAPACITY is range -25.0 to 25.0;
signal Sig_1: CAPACITY:= 3.0;
type PROBABILITY is range 1.0 downto 0.0;
```

```
constant P: PROBABILITY:= 0.5;
```

Các cách biểu diễn số thực:

+ Biểu diễn dưới dạng thập phân:

```
integer[.integer][exponent]
```

Ví dụ: `0.0, 0.5, 1.1234_5678, 12.4E-9…`

+ Biểu diễn dưới dạng cơ số xác định:

```
base#based_integer[.based_integer ]#[exponent]
```

Ví dụ: `2#1.111_1111_111#E+11 (=1,1023.10`^{3}`),`

```
  16#F.FF#E2 (=4095.0)
```

- Character: Kiểu ký tự - kiểu dữ liệu liệt kê với miền xác định là tập hợp các ký tự ASCII. Biểu diễn của giá trị Character: 'A', 'a', '*', ' ', NUL, ESC…

Kiểu Vật lý - Physical: được sử dụng để biểu diễn các đại lượng vật lý như khoảng cách, điện trở, dòng điện, thời gian… Kiểu vật lý cung cấp đơn vị cơ bản và các đơn vị kế tiếp được định nghĩa theo đơn vị cơ bản, đơn vị nhỏ nhất có thể biểu diễn được là đơn vị cơ bản. Trong thư viện chuẩn của VHDL kiểu Time (kiểu dữ liệu thời gian) là kiểu vật lý duy nhất đã được định nghĩa.

```
type Time is range <xác_định giới hạn>
units
fs; -- Đơn vị cơ bản
ps = 1000 fs;
ns = 1000 ps;
us = 1000 ns;
ms = 1000 us;
sec = 1000 ms;
min = 60 sec;
hr = 60 min;
End Units;
```

Ví dụ sử dụng:

```
constant Tpd: time:= 3ns;
...
Z <= A after Tpd;
```

- Std_logic và Std_ulogic: kiểu dữ liệu logic nhiều mức đã được định nghĩa trong gói ***std_logic_1164***, so với kiểu ***Bit*** thì chúng có thể mô tả chính xác và chi tiết hơn cho các phần cứng số, chúng còn xác định được cường độ khác nhau của các tín hiệu số. Đây là các kiểu dữ liệu rất quan trọng thường được sử dụng để mô tả cho các tín hiệu trong các hệ thống số cho mục đích thiết kế, tổng hợp, hiện thực hệ thống đó bằng mạch cứng.

```
type std_ulogic is
('U', -- Uninitialize
 'X', -- Forcing Unknown
 '0', -- Forcing Zero
 '1', -- Forcing One
 'Z', -- High Impedance
 'W', -- Weak Unknown
 'L', -- Weak Zero
 'H', -- Weak One
 '-' -- Don't Care
) ;
```

```
type std_logic is
('U', -- Uninitialize
 'X', -- Forcing Unknown
 '0', -- Forcing Zero
 '1', -- Forcing One
 'Z', -- High Impedance
 'W', -- Weak Unknown
 'L', -- Weak Zero
 'H', -- Weak One
 '-' -- Don't Care
) ;
```

Ý nghĩa của các mức logic như sau:

```
'U', -- Uninitialize: Mức khởi tạo
'X', -- Forcing Unknown: Mức không xác định chiếm ưu thế
'0', -- Forcing Zero: Mức logic thấp chiếm ưu thế
'1', -- Forcing One: Mức logic cao chiếm ưu thế
'Z', -- High Impedance: Mức trở kháng cao
'W', -- Weak Unknown: Mức không xác định yếu
```

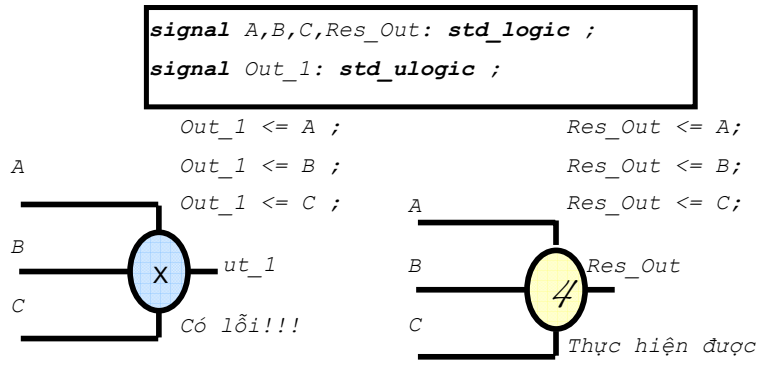
```
'L', -- Weak Zero : Mức logic thấp yếu
'H', -- Weak One : Mức logic cao yếu
'-' -- Don't Care : Không quan tâm đến mức logic
```

Các mức logic chiếm ưu thế bao giờ cũng có cường độ lớn hơn các mức logic yếu, ví dụ nếu đầu ra của một cổng logic có trạng thái cao được nối với tín hiệu GND, trong trường hợp này thì đầu ra chính là mức logic cao yếu 'H' còn tín hiệu GND có mức logic thấp chiếm ưu thế '0', như vậy mức trạng thái của đầu ra đó sẽ bị kéo xuống mức logic chiếm ưu thế '0', điều này có thể làm hỏng cổng logic.

Hai kiểu dữ liệu ***std_logic*** và ***std_ulogic*** tương tự nhau, chúng chỉ khác nhau ở chỗ là kiểu ***std_ulogic*** không có hàm phân giải (***unresolved***), còn kiểu ***std_logic*** có hàm phân giải – hàm quyết định giá trị tín hiệu, do đó sẽ có lỗi khi các tín hiệu kiểu ***std_ulogic*** được nối chung vào 1 điểm. Thư viện cũng cung cấp hàm phát hiện lỗi này của các tín hiệu kiểu ***std_ulogic.***

Nếu không có ràng buộc gì đặc biệt khi thiết kế hệ thống số thì nên dùng kiểu ***std_logic***. Vì thực thế hệ thống số hay được thiết kế theo kiểu sử dụng bus tín hiệu chung, nên sẽ có nhiều tín hiệu khác nhau của các khối con khác nhau cùng nối chung vào bus.

Ví dụ như cách mô tả sau cho ta thấy sự khác nhau về việc sử dụng kiểu dữ liệu ***std_logic*** và ***std_ulogic*** trong quá trình chạy mô phỏng hay tổng hợp từ mã VHDL. Theo mã mô tả thì tín hiệu A, B, C cùng nối chung vào tín hiệu Out_1, Reg_Out. Vì Reg_Out có kiểu ***std_logic***, nên chương trình sẽ không báo lỗi vì kiểu dữ liệu này có hàm phân giải để quyết định mức logic của Reg_Out theo các mức logic chiếm ưu thế trong 3 tín hiệu A, B, C.



(Ký hiệu "<=" dùng ở trên là lệnh gán tín hiệu (sẽ được trình bày chi tiết hơn ở phần sau) lệnh gán tín hiệu thực hiện được với 2 dữ liệu cùng kiểu, cùng độ lớn, giá trị của tín hiệu bên phải sẽ được gán cho tín hiệu bên trái).

So sánh 2 bảng chân lý sau để hiểu hơn về hàm phân giải:

Bảng 9.1: Bảng chân lý của hàm AND và phân giải ứng với Std_logic

	U	X	0	1	Z	W	L	H	-
U	U	U	0	U	U	U	0	U	
X	U	X	0	X	X	X	0	X	
0	0	0	0	0	0	O	0	0	
1	U	X	0	1	X	X	0	1	
Z	U	X	0	X	X	X	0	X	
W	U	X	0	X	X	X	0	X	
L	0	0	0	0	0	0	0	0	
H	U	X	0	1	X	X	0	1	
-	U	X	0	X	X	X	0	X	

	U	X	0	1	Z	W	L	H	-
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	X
1	U	X	X	1	1	1	1	1	X
Z	U	X	0	1	Z	W	L	H	X
W	U	X	0	1	W	W	W	W	X
L	U	X	0	1	L	W	L	W	X
H	U	X	0	1	H	W	W	H	X
-	U	X	X	X	X	X	X	X	X

- Kiểu dữ liệu liệt kê tự định nghĩa: Kiểu dữ liệu liệt kê, do người sử dụng tự định nghĩa, cho phép mô tả rất rõ ràng và linh hoạt cho các mô hình phần cứng số với mức độ trừu tượng cao. Kiểu dữ liệu này dùng nhiều mô tả đồ hình trạng thái (máy trạng thái), các hệ thống phức tạp…

Ví dụ:

```
type My_State is ( RST, LOAD, FETCH, STOR, SHIFT);
...
signal STATE, NEXT_STATE: My_State;
```

Kiểu dữ liệu My_State gồm 5 giá trị RST, LOAD, FETCH, STOR, SHIFT. Các chương trình phần mềm thiết kế thường mặc định mã hóa các giá trị đó bằng số nhị phân 3 bit. Tuy nhiên các chương trình phần mềm thiết kế thường cho phép gán giá trị bit nhị phân tùy ý cho các giá trị của kiểu dữ liệu liệt kê tự định nghĩa này.

9.3.2.2 Kiểu dữ liệu ghép

Tương tự các ngôn ngữ lập trình, VHDL cũng có các kiểu dữ liệu ghép là nhóm các phần tử dữ liệu theo dạng mảng (array) hoặc bảng ghi (record).

+ Mảng – Array:

Mảng là nhóm nhiều phần tử có cùng kiểu dữ liệu với nhau thành đối tượng duy nhất. Mỗi phần tử của mảng có thể được truy cập bằng một hoặc nhiều chỉ số của mảng. Cú pháp định nghĩa kiểu dữ liệu mảng như sau:

```
Type tên_mảng is array (khoảng _của _chỉ số) of
kiểu_của_phần_tử;
```

Ví dụ một số cách khai báo và sử dụng dữ liệu mảng:

```
type WORD is array (3 downto 0) of std_logic ;
signal B_bus: WORD ;
type DATA is array (3 downto 0) of integer range 0 to 9;
signal C_bus: DATA ;
```

Các kiểu dữ liệu mảng đã được định nghĩa trong thư viện chuẩn của VHDL (các kiểu dữ liệu vector) là:

Bit_logic_vector (mảng dữ liệu kiểu Bit).

Std_logic_vector (mảng dữ liệu kiểu std_logic)

Std_ulogic_vector (mảng dữ liệu kiểu std_ulogic).

String (mảng dữ liệu kiểu Chacracter).

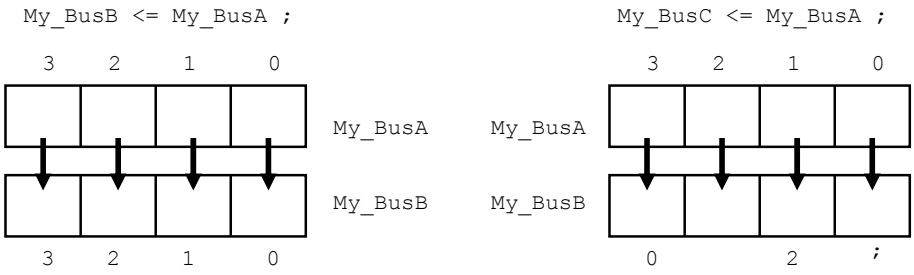
Ngoài ra có kiểu ***Signed*** (Kiểu logic có dấu) và ***Unsigned*** (Kiểu logic không dấu) được định nghĩa trong thư viện ***std_logic_arith.*** Chúng tương tự như kiểu ***std_logic_vector***. Với ***Signed*** thì bit đầu tiên là bit dấu và số âm được biểu diễn dưới dạng mã bù 2. Ví dụ "1101" nếu là kiểu ***Unsigned*** thì có giá trị thập phân là 13, còn nếu là kiểu ***Signed*** thì có giá trị thập phân là -3.

Một số ví dụ sử dụng các kiểu dữ liệu vector như sau:

```
signal My_BusA, My_BusB: bit_vector (3 downto 0);
signal My_BusC: bit_vector (0 to 3);
signal Data_Word: std_logic_vector (11 downto 0);
signal Data1: signed (0 to 7);
signal Data2: unsigned (8 downto 1);
variable Warning2: string(1 to 30):= " Unstable,
Aborting Now" ;
constant Warning3: string(1 to 20):= " Entering
FSM State2";
```

Một số phép toán thao tác với phần tử mảng:

- Phép gán cho mảng: 2 mảng phải cùng kiểu, cùng độ lớn, phép gán sẽ thực hiện gán theo từng phần tử theo thứ tự từ trái sang phải:




```
Data_Word <= "101001101111";
Data_Word <= X"A6F";
```

```
Data_Word <= O"5157";
Data_Word <= B"1010_0110_1111";
```

Cách biểu diễn số liệu ***bit_vector*** và ***std_logic_vector***: B|O|X "giá_trị" (dùng dấu nháy kép). Trong đó B: Binary -Kiểu nhị phân, O: Octal - kiểu bát phân, X: hexadecimal.

```
X"1AF"=B"0001_1010_1111"= B"000_110_101_111"=O"0657"
```

- Phép gộp ( ): cho phép nhóm cả dữ liệu vô hướng và dữ liệu mảng để thuận tiện cho các phép gán cho mảng:

```
signal H_BYTE, L_BYTE: std_logic_vector (0 to 7);
signal Q_Out: std_logic_vector (31 downto 0);
signal A, B, C, D: std_logic;
signal WORD: std_logic_vector (3 downto 0);
...
(A,B,C,D)<=WORD; -- A<=WORD[3], B<=WORD[2],
C<=WORD[1], D<=WORD[0].
```

Chú ý: Phép gộp ở vế bên trái chỉ dùng với kiểu dữ liệu vô hướng.

```
WORD <= ( 2 => '1', 3 => D, others => '0' ); --
WORD="D100"
Q_Out <= (others => '0');
WORD <= ( A, B, C, D );
H_Byte <= (7|6|0=>'1', 2 to 5 => '0' );-- Bit 7, 6, 0
của H_Byte
 -- được gán bằng '1', từ bit 2 đến bit 5 được gán
bằng '0'.
L_Byte <= (3=>'1', 1 to 2 => '0', 4 to 7 => '1');
```

Chú ý: ***"others"*** có thể được sử dụng khi gán mặc định, nó có ý nghĩa là các tất cả các phần tử còn lại được gán bằng một giá trị nào đó).

+ Bảng ghi - Record:

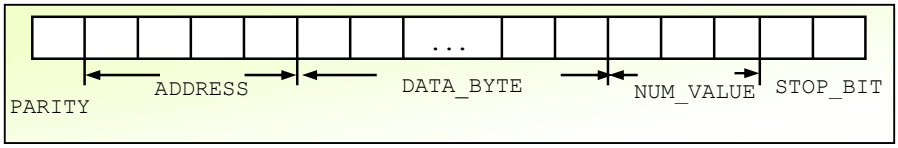
Bảng ghi là nhóm nhiều phần tử có kiểu dữ liệu khác nhau thành đối tượng duy nhất.

Mỗi phần tử của bản ghi được truy nhập tới theo tên trường.

Các phần tử của bản ghi có thể nhận mọi kiểu của ngôn ngữ VHDL kể cả mảng và bảng ghi.

Ví dụ định nghĩa kiểu dữ liệu bảng ghi như sau:

```
type OPCODE is record
PARITY: bit;
ADDRESS: std_logic_vector ( 0 to 3 );
DATA_BYTE: std_logic_vector ( 7 downto 0 );
NUM_VALUE: integer range 0 to 6;
STOP_BITS: bit_vector (1 downto 0);
end record ;
...
```




```
signal TX_PACKET, RX_PACKET: OPCODE;
```

Cách truy nhập và gán dữ liệu cho các trường của bảng ghi: Các phần tử của bảng ghi được truy nhập theo tên bảng ghi và tên trường, 2 thành phần này được ngăn cách bởi dấu '.'. Xét một số ví dụ sau để hiểu cách truy nhập và gán dữ liệu cho bảng ghi:

```
TX_PACKET <= ('1',"0011","11101010",5,"10");
TX_PACKET.ADDRESS <= ("0011");
TX_PACKET <= RX_PACKET;
TX_PACKET.ADDRESS <= RX_PACKET.ADDRESS;
```

9.3.2.3 Kiểu dữ liệu mảng 2 chiều (2-D Array)

Mảng 2 chiều là kiểu dữ liệu mảng của các phần tử mảng một chiều hay bảng ghi. Một số ví dụ định nghĩa và khai báo kiểu dữ liệu mảng 2 chiều như sau:

```
type Mem_Array is array (0 to 3) of std_logic_vector
(7 downto 0);
type Data_Array is array (0 to 2) of OPCODE;
...
signal My_Mem:Mem_Array;
signal My_Data:Data_Array;
-- Ví dụ ứng dụng dùng mảng 2 chiều khởi tạo một vùng
nhớ ROM
constant My_ROM:REM_Array:= (0 =>(others=>'1'),-- gán
cho phần tử thứ nhất
             1 => "10100010", -- gán cho phần tử thứ 2
             2 => "00001111", -- gán cho phần tử thứ 3
             3 => "11110000");-- gán cho phần tử thứ 4
```

9.3.2.4 Kiểu dữ liệu con

Là một tập hợp con của các kiểu dữ liệu đã được định nghĩa khác. Phép khai báo kiểu dữ liệu con có thể nằm ở mọi vị trí cho phép khai báo kiểu dữ liệu. Cú pháp khai báo chung:

```
Subtype Tên_kiểu_dữ_liệu_con is
xác_định_kiểu_dữ_liệu_con;
Ví dụ:subtype My_Int is integer range 0 to 255;
      subtype My_Small_Int is My_Int range 5 to 30;
      subtype word is bit_vector(31 downto 0);
```

9.3.2.5 Tổng kết

Trong VHDL có những lệnh chỉ dùng cho mục đích mô phỏng mà không thể tổng hợp được mạch cứng.

Bảng tổng kết sau đưa ra kiểu dữ liệu có thể được dùng để viết mã VHDL có thể tổng hợp được mạch cứng:

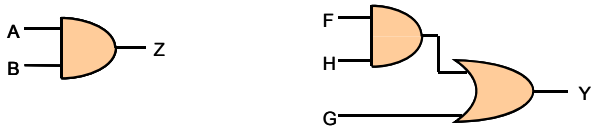
Bảng 9.2: Kiểu dữ liệu dùng để viết mã VHDL có thể tích hợp trên mạch

Kiểu dữ liệu	Giá trị có thể tổng hợp được
BIT, BIT_VECTOR	'0', '1'
STD_LOGIC, STD_LOGIC_VECTOR	'X', '0', '1', 'Z' (resolved)
STD_ULOGIC, STD_ULOGIC_VECTOR	'X', '0', '1', 'Z' (unresolved)
BOOLEAN	True, False
NATURAL	Từ 0 tới +2.147.483.647
INTEGER	Từ -2.147.483.647 đến +2.147.483.647
SIGNED	Từ -2.147.483.647 đến +2.147.483.647
UNSIGNED	Từ 0 to +2.147.483.647
Kiểu số nguyên tự định nghĩa	Tập con của INTEGER
Kiểu dữ liệu liệt kê	Các giá trị được liệt kê bởi người dùng
SUBTYPE	Tập dữ liệu con được định nghĩa
ARRAY	Mảng của bất kỳ kiểu dữ liệu nào ở trên
RECORD	Bảng ghi chứa bất kỳ kiểu dữ liệu nào ở trên

9.3.3 Các phép toán trong VHDL

9.3.3.1 Toán tử logic

Toán tử logic gồm có: **AND, OR, NAND, NOR, XOR, NOT, XNOR** được sử dụng cho các dạng dữ liệu là ***bit, boolean, bit_vector, std_logic_vector.***



Ví dụ: Z <= A ***and*** B; Y <= G ***or*** (F ***and*** H) ;

Chú ý:

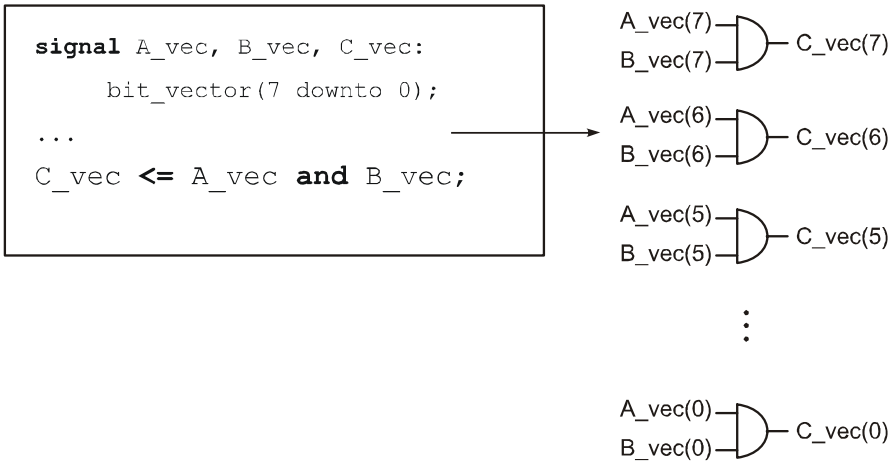
Toán tử logic dùng cho kiểu dữ liệu mảng có nguyên tắc thực hiện như sau:

+ Chỉ thực hiện với các mảng cùng kiểu, cùng độ lớn.

+ Phép toán logic thực hiện với từng phần tử của mảng và theo thứ tự từ trái sang phải.

+ Toán tử ***not*** có độ ưu tiên cao nhất:

Ví dụ: `Z <= not A and B -- = (not A) and B.`



9.3.3.2 Toán tử quan hệ

Toán tử quan hệ được sử dụng cho hầu hết các dạng dữ liệu, tất cả các toán tử quan hệ đều cho giá trị trả về dưới dạng boolean.

Toán tử quan hệ gồm có: =, /=, <, <=, >, >=.

Ví dụ: `signal FLAG_BIT: boolean ;`

```
signal A, B: integer ;
FLAG_BIT <=(A > B);
```

- Nguyên tắc thực hiện phép quan hệ với dữ liệu mảng:

+ Các mảng phải cùng kiểu, độ dài có thể khác nhau.

+ Mảng có độ dài khác nhau thì phép quan hệ thực hiện ưu tiên phần tử từ trái sang phải và so sánh theo giá trị ASCII.

```
signal A_vec: bit_vector ( 7 downto 0 ):= "11000110";
signal B_vec: bit_vector ( 5 downto 0 ):= "111001";
...
if ( A_vec > B_vec ) then
State <= Normal;
else
State <= Code_Red;
end if;
```

9.3.3.3 Toán tử số học

Toán tử số học được sử dụng cho kiểu dữ liệu ***Integer, Real, Signed, Unsigned, các dạng dữ liệu vật lý, Std_logic, Std_logic_vector, Bit, Bit_vector***. Cần chú ý rằng không phải tất cả toán tử số học đều có thể sử dụng cho kiểu dữ liệu mảng.

Các toán tử số học là: +, -, *, /, abs (trị tuyệt đối), ** (hàm mũ).

9.3.3.4 Toán tử dịch

Toán tử dịch là toán tử tác động lên toán hạng kiểu Bit_vector để tạo ra các phép dịch hoặc quay dữ liệu. Cú pháp của toán tử dịch:

Toán_hạng_trái Toán_Tử_dịch Toán hạng phải;

Trong đó: <Toán_hạng_trái> phải là kiểu Bit_vector sẽ được dịch hoặc quay dữ liệu, <Toán_hạng_phải> xác định số vị trí được dịch hoặc quay và phải có kiểu số nguyên mang giá trị dương hoặc âm, nếu là giá trị âm sẽ chỉ ra hướng ngược lại với giá trị dương. Mỗi phép dịch cho kết qủa cùng dạng và kích thước với toán hạng ban đầu.

Các toán tử dịch trong VHDL là: ***sll*** (dịch trái logic), ***srl*** (dịch phải logic), ***sla*** (dịch trái số học), ***sra*** (dịch phải số học), ***rol*** (quay trái), ***ror*** (quay phải).

Ví dụ:

```
signal A_vec: bit_vector (7 downto 0):= "11000110";
signal D_vec: bit_vector (7 downto 0);
```

`D_vec <= A_vec sll 2;`	`D_vec ="00011000"`	Dịch trái 2 bit và vị trí bên phải được điền giá trị 0.
`D_vec <= A_vec sra 2;`	`D_vec ="11110001"`	Dịch phải số học 2 bit và vị trí bên trái được giữ như giá trị ban đầu.
`D_vec <= A_vec ror 3;`	`D_vec ="11011000"`	Quay phải 3 bit.
`D_vec <= A_vec srl 2;`	`D_vec ="00110001"`	Dịch phải 2 bit và vị trí bên trái được điền giá trị 0
`D_vec <= A_vec sra -2;`	`D_vec ="00011000"`	Dịch phải số học -2 bit (dịch trái số học 2 bit).

9.3.3.5 Toán tử ghép nối

Toán tử ghép nối "&" cho phép ghép nối một cách linh hoạt các dữ liệu đơn và dữ liệu dạng mảng thành các mảng lớn hơn.

Ví dụ:

```
signal A_vector, B_vector: std_logic_vector (7 downto 0);
signal Z_vector: std_logic_vector (15 downto 0);
  Z_vector <= A_vector & B_vector;
```

9.3.3.6 Toán tử tách

Toán tử tách cho phép ta lấy ra một số thành phần của mảng, chiều chỉ số của phép tách phải cùng chiều đánh chỉ số đã định nghĩa cho mảng.

Ví dụ:

```
signal Z_vec: std_logic_vector (15 downto 0);
signal B_vec: std_logic_vector (7 downto 0);
B_vec <= Z_vec (12 downto 5);
```

9.3.3.7 Toán tử thuộc tính

Toán tử thuộc tính cho phép xác định thuộc tính dữ liệu của đối tượng biến và tín hiệu. Cú pháp chung:

```
Đối_tượng'thuộc_tính
```

- Các thuộc tính cho kiểu dữ liệu mảng trong VHDL là:

+ ***d'left, d'right:*** Trả lại chỉ số của phần tử bên trái nhất hoặc bên phải nhất của dữ liệu mảng d.

+ ***d'high, d'low:*** Trả lại chỉ số của phần tử cao nhất hoặc thấp nhất của kiểu dữ liệu mảng d.

+ ***d'range, d'reverse_range:*** Xác định khoảng của chỉ số của mảng d.

+ ***d'length:*** Trả về số lượng các phần tử của mảng d.

- Các thuộc tính cho kiểu dữ liệu liệt kê

+ ***d'val(pos):*** Trả về giá trị tại vị trí được xác định pos.

+ ***d'pos(value):*** Trả về vị trí của giá trị xác định value.

+ **d'leftof(value):** Trả về giá trị bên trái của giá trị xác định value.

+ ***d'val(row, column):*** Trả về giá trị tại ví trí có hàng, cột xác định.

- Các thuộc tính cho tín hiệu

+ ***s'event, s'stable:*** Trả về giá trị boolean, chỉ ra rằng trên đường tín hiệu s có xuất hiện sự kiện thay đổi hay giá trị trên đường tín hiệu ổn định tại thời điểm hiện tại. Các thuộc tính này dùng nhiều với lệnh ***wait*** và ***if.***

+ ***s'active***: Trả về giá trị "true" nếu s='1'.

+ ***s'quiet (time):*** Trả về giá trị "true" nếu không có sự kiện nào xảy ra đối với tín hiệu s trong khoảng thời gian xác định time.

+ ***s'last_event:*** Trả về thời gian kể từ khi có sự kiện mới nhất xảy ra đối với s.

+ ***s'last_active:*** Trả về thời gian kể từ khi s='1'

+ ***s'last_value:*** Trả về giá trị của s trước khi xảy ra sự kiện mới nhất.

Ví dụ sử dụng toán tử thuộc tính như sau:

```
signal clk: std_logic:='0';
...
PROCESS(clk)
TYPE bit4 IS ARRAY(0 TO 3) of BIT;
TYPE bit_strange IS ARRAY(10 TO 20) OF BIT;
VARIABLE len1, len2: INTEGER;
BEGIN
If (clk'event and clk='1')then -- sự kiện có sườn
dương của clk.
len1:= bit4'LENGTH; -- returns 4
len2:= bit_strange'LENGTH; -- returns 11
End if;
END PROCESS;
```

- Thuộc tính tự định nghĩa: Người sử dụng có thể tự định nghĩa hàm thuộc tính với cú pháp như sau (viết trong gói dữ liệu package của thư viện do người sử dụng tạo ra:

```
ATTRIBUTE Tên_thuộc_tính: Kiểu_thuộc_tính; -- Khai báo
chung
ATTRIBUTE Tên_thuộc_tính OF Tên_đối_tượng: Loại_đối_tượng
  IS giá_trị;-- Xác định giá trị của thuộc tính
```

Trong đó:

+ *Kiểu_thuộc_tính:* Kiểu dữ liệu trả về của thuộc tính (Boolean, Integer, std_logic_vector…)

+ *Loại_đối_tượng:* Kiểu đối tượng của thuộc tính (Type, Signal, Function…).

Ví dụ:

```
ATTRIBUTE number_of_inputs: INTEGER; -- Khai báo chung
ATTRIBUTE number_of_inputs OF nand3: SIGNAL IS 3; -- xác
định giá trị của thuộc tính
...
inputs <= nand3'number_of_pins; -- attribute call, returns 3
```

9.3.3.8 Biến đổi dữ liệu

VHDL không cho phép thực hiện các phép toán trực tiếp (số học, logic…) giữa các số liệu có kiểu dữ liệu khác nhau. Vì vậy cần có hàm biến đổi dữ liệu từ kiểu này sang kiểu khác theo 2 cách như sau:

- Viết một đoạn mã VHDL để thực hiện biến đổi dữ liệu

- Sử dụng hàm – Function trong các Packet đã được định nghĩa: Nếu các dữ liệu có quan hệ chặt chẽ với nhau (ví dụ như 2 toán hạng có cùng cơ sở "base", mặc dù được khai báo theo 2 lớp kiểu khác nhau), thì gói ***std_logic_1164*** của thư viện IEEE có cung cấp các hàm biến đổi dữ liệu trực tiếp giữa chúng giống như "ép kiểu" trong ngôn ngữ lập trình C/C++. Xét ví dụ dưới đây:

```
TYPE long IS INTEGER RANGE -100 TO 100;
TYPE short IS INTEGER RANGE -10 TO 10;
-- Kiểu long và short cùng cơ sở là kiểu INTEGER
SIGNAL x: short;
SIGNAL y: long;
...
y<= 2*x + 5; -- lỗi, vì kết quả vế trái và vế phải không
cùng kiểu
y<=long(2*x + 5); --OK, Dữ liệu vế phải được chuyển cùng
kiểu vế trái.
```

Một số hàm biến đổi dữ liệu được cung cấp trong gói std_logic_arith như sau:

+ ***Conv_integer(p):*** Biến đổi tham số p có kiểu dữ liệu integer, unsigned, signed, std_logic sang kiểu integer.

+ ***Conv_unsigned(p, b):*** Biến đổi tham số p có kiểu dữ liệu integer, unsigned, signed, std_logic sang kiểu unsigned kích thước b bit.

+ ***Conv_signed(p, b):*** Biến đổi tham số p có kiểu dữ liệu integer, unsigned, signed, std_logic sang kiểu signed kích thước b bit.

+ ***Conv_std_logic_vector(p, b):*** Biến đổi tham số p có kiểu integer, unsiged, signed hoặc std_logic sang kiểu std_logic_vector kích thước b bit.

9.3.3.9 Định nghĩa các toán tử chồng

VDHL cho phép định nghĩa các toán tử chồng theo các toán tử đã được định nghĩa giống như khái niệm "overloading" trong ngôn ngữ lập trình C++.

Ví dụ toán tử "+" cho dữ liệu kiểu Bit chưa được định nghĩa trong các thư viện chuẩn, người sử dụng có thể viết thêm hàm (FUNCTION – sẽ được đề cập chi tiết ở phần sau) định nghĩa cho toán tử "+" cho kiểu dữ liệu đó như sau:

```
FUNCTION "+" (a: INTEGER, b: BIT) RETURN INTEGER IS
BEGIN
IF (b='1') THEN RETURN a+1;
ELSE RETURN a;
END IF;
END "+";
```

S□ d□ng toán t□ "+" □ trên nh□ sau:

```
SIGNAL inp1, outp: INTEGER RANGE 0 TO 15;
SIGNAL inp2: BIT;
(...)
outp <= 3 + inp1 + inp2;
(...)
------------------------------
```

Trong biểu thức "outp<=3+inp1+inp2;'', phép "+" đầu tiên đã được định nghĩa trong thư viện chuẩn giữa 2 số Integer, còn phép cộng thứ 2 là toán tử "+" đã được định nghĩa theo kiểu "overloading" ở trên.

9.3.3.10 Tổng kết

Bảng tổng kết các phép toán trong VHDL như sau:

Bảng 9.3: Các phép toán trong VHDL

Phép toán	***Toán t□***	***Ki□u d□ li□u***
Phép gán	*<=,:=, =>*	*Bất kỳ kiểu dữ liệu nào*
Phép toán logic	*NOT, AND, NAND,* *OR, NOR, XOR, XNOR*	*BIT, BIT_VECTOR,* *STD_LOGIC, STD_LOGIC_VECTOR,* *STD_ULOGIC, STD_ULOGIC_VECTOR*

Phép toán số học	*+, -,*,/,**	*INTEGER, SIGNED, UNSIGNED*
	(mod, rem, abs)- chỉ dùng cho mô phỏng	
Phép quan hệ	*=, /=, <, >, <=, >=*	*Tất cả các kiểu dữ liệu ở trên*
Phép dịch	*sll, srl, sla, sra, rol, ror*	*BIT_VECTOR*
Phép gộp	*&, (,,,)*	*BIT, BIT_VECTOR,* *STD_LOGIC, STD_LOGIC_VECTOR,* *STD_ULOGIC, STD_ULOGIC_VECTOR*

9.3.4 Các đơn vị thiết kế trong VHDL

Các đơn vị thiết kế chính là thành phần cấu trúc chính (giống như khung chương trình) để viết mã lệnh mô tả VHDL cho mạch hay hệ thống số. Mỗi đơn vị thiết kế sẽ có các nhiệm vụ riêng biệt khác nhau.

VDHL sử dụng 6 đơn vị thiết kế gồm 2 loại: Đơn vị cơ bản và đơn vị thiết kế thứ cấp:

- Đơn vị thiết kế cơ bản:

Library: Cho phép tạo thư viện trong VHDL

Package: Tạo các gói giữ liệu trong Library, như các khai báo các đối tượng, khai báo thủ tục, hàm...

Entity (Thực thể): Cho phép khai báo các giao diện của một khối thiết kế số nào đó: như khai báo các cổng vào/ra, các tham số của khối mạch...

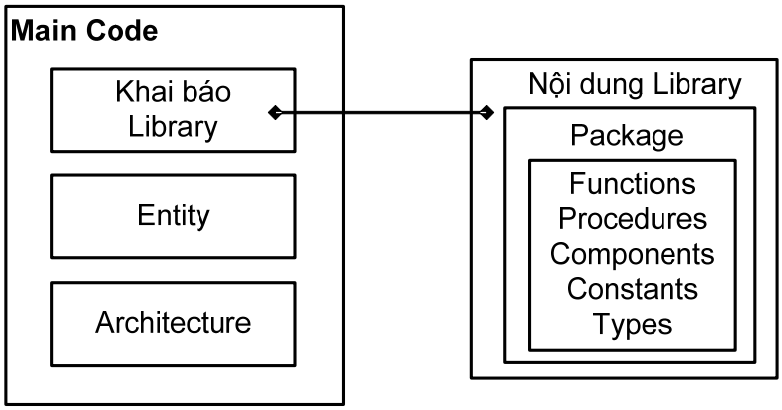
- Đơn vị thiết kế thứ cấp (phụ thuộc vào một đơn vị thiết kế cơ bản):

Architecture: Mô tả hoạt động bên trong của một ***Entity*** hay đây chính là phần mô tả hoạt động của khối mạch số.

Package Body: Mô tả chi tiết cho các khai báo trong ***Package*** như viết các hàm, các thủ tục...

Configuration: Đơn vị thiết kế cấu hình cho phép gắn các phiên bản của thực thể vào những kiến trúc khác nhau. Cấu hình cũng có thể được sử dụng để thay thế một cách nhanh chóng các phần tử của thực thể trong các biểu diễn cấu trúc của thiết kế.

Quan hệ và các đơn vị thiết kế có quan hệ như hình sau:



Hình 9.1 Các thành phần cơ bản của mã VHDL

9.3.4.1 Entity (Thực thể)

Đây là phần khai báo tham số, chỉ tiêu, giao diện cổng vào/ra cho phần tử, khối con hay cả hệ thống số. Ta có thể có tất cả các thông tin để kết nối phần tử mạch này vào phần tử mạch khác trong hệ thống hoặc thiết kế tác nhân đầu vào phục vụ cho mục đích mô tả hoạt động của phần tử và hệ thống số sau này. Hoạt động thật sự của mạch không nằm ở phần khai báo này mà được viết trong phần ***Architechture*** tương ứng.

Trong nhiều phần mềm thiết kế cho phép việc khai báo ***Entity*** này hoàn toàn tự động. Người sử dụng chỉ cần sử dụng một bảng số liệu vào nhập vào tên các cổng vào/ra, tham số cho phần tử hay hệ thống và gán kiểu dữ liệu tương ứng cho chúng. Phần mềm sẽ tự động tạo ra mã mô tả cho phần Entity.

Cú pháp khai báo chung của một Entity như sau:

```
entity Tên_thực_thể is
      generic(--Khai báo danh sách các tham số chung
 Tên_tham_số: [Kiểu_dữ_liệu] [:=giá_trị_khởi_tạo];
 ...
 );
port(-- Khai báo danh sách đối tượng các port vào/ra
 Tên_cổng: [mode] [Kiểu_dữ_liệu] [:=giá_trị_khởi_tạo];
...
 );
end Tên_thực_thể;
```

Trong khai báo trên:

+ Tham số khai báo trong phần ***generic*** dùng để kiểm soát, thay đổi cấu trúc, hoạt động của thực thể, chúng sẽ được truyền giá trị hoặc lấy giá trị mặc định ban đầu khi thực thể được khởi tạo. Tham số này rất hữu ích khi thiết kế theo kiểu cấu trúc, sẽ sử dụng nhiều thành phần cấu trúc cùng kiểu như khai báo của ***Entity*** nhưng có tham số về cấu trúc, hoạt động khác nhau.

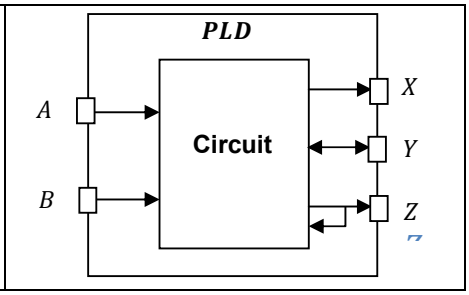
+ "--" Dấu đánh dấu dòng chú thích (comment) trong mã mô tả VHDL

+ [mode]: chỉ hướng tín hiệu của cổng có thể là: (***in, out, inout*** hoặc ***buffer***). Trong đó cổng dạng ***in*** chỉ dùng để đọc dữ liệu. Cổng dạng ***out*** chỉ dùng để gán giá trị dữ liệu. Cổng ***inout*** cho phép đồng thời vừa đọc vừa gán giá trị dữ liệu ở trong và ngoài chương trình. Cổng

dạng ***buffer*** cho phép cả 2 thao tác đọc và gán dữ liệu từ bên trong chương trình, nhưng chỉ cho phép đọc dữ liệu từ ngoài chương trình.

Ví dụ khai báo thực thể cho mạch số "Circuit" như hình vẽ bên:

```
entity Circuit is
generic(delay:Time:=1ns);
port(A, B: in std_logic;
X: out std_logic;
Y: inout std_logic;
Z: buffer std_logic);
end Circuit;
```



9.3.4.2 Architecture (Kiến trúc)

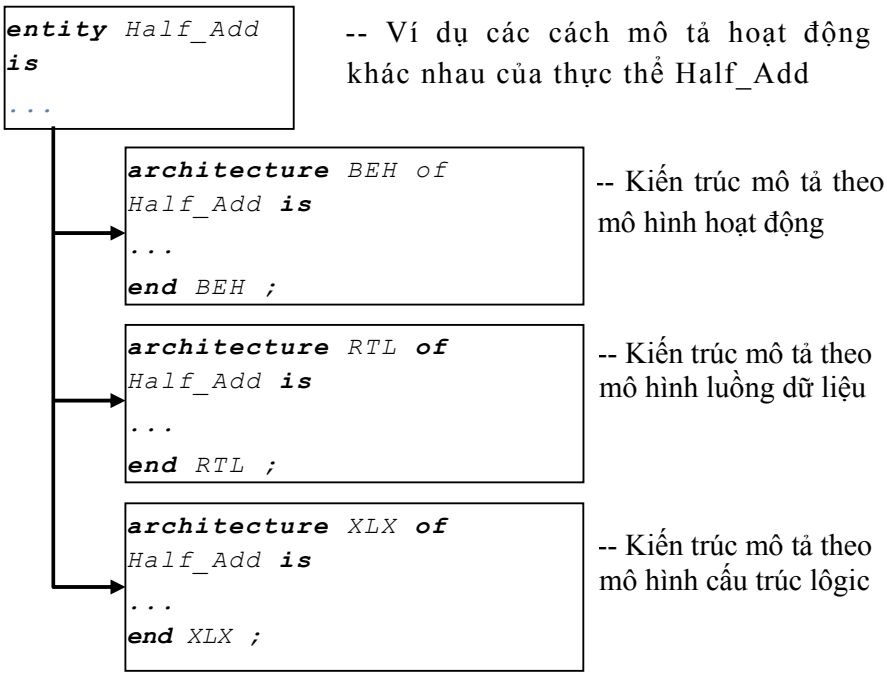
Cấu trúc này cho phép mô tả hoạt động bên trong của thực thể. Cú pháp chung của một ***Architecture:***

```
Architecture Tên_kiến_trúc of Tên_thực_thể is
-- Thực hiện các khai báo cho kiến trúc
...
Begin
-- Viết các mô tả hoạt động bên trong cho thực thể
...
End Tên_kiến_trúc;
```

Phần khai báo kiến trúc có thể bao gồm các khai báo về các đối tượng ***signal, constant, kiểu dữ liệu***, khai báo các phần tử bên trong hệ thống ***(component)*** hay các hàm ***(function)*** và thủ tục ***(proceduce)*** sẽ được sử dụng để mô tả hoạt động của hệ thống. Tên_kiến_trúc là nhãn được đặt tuỳ theo người sử dụng.

VHDL cho phép tạo ra nhiều mô tả ***Architecture*** cho một thực thể, cho phép thực hiện nhiều cách mô tả hoạt động khác nhau cho một thực thể. Mỗi cách mô tả hoạt động sẽ tối ưu về mặt thời gian thiết kế hay độ tin cậy hay tối ưu về tài nguyên sử dụng khi tổng hợp…

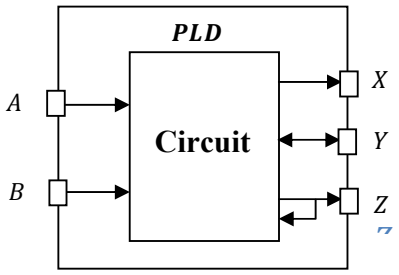
Thông thường có thể sử dụng 3 cách chính mô tả kiến trúc của một phần tử hay hệ thống số đó là: mô tả theo mô hình hành vi (Behaviour), mô tả theo mô hình cấu trúc logic (Structure) và mô hình luồng dữ liệu (RTL). Tuy nhiên để mô tả cho một hệ thống, trong một kiến trúc có thể kết hợp sử dụng 2 hoặc cả 3 mô hình mô tả trên để thực hiện cho từng thành phần con tương ứng của hệ thống số. Trong phần sau của cuốn sách này sẽ trình bày chi tiết hơn các phương pháp mô tả này.



Theo VHDL tiêu chuẩn thì một Entity có thể có nhiều Architechture mô tả hoạt động cho nó theo các cách khác nhau, tuy nhiên đa số các phần mềm thiết kế hiện nay trong một file mã mô tả VHDL thì chỉ cho phép viết một Architechture, nếu muốn viết nhiều cách mô tả hoạt động khác nhau thì có thể viết ở nhiều file khác nhau và với tên khác của Entity nhưng có giao diện hoàn toàn giống nhau.

Ví dụ mô tả hoạt động cho mạch "Circuit" như sau:

```
-- Khai báo giao diện
entity Circuit is
generic(delay: Time:=1ns);
port(A, B: in std_logic;
 X: out std_logic;
 Y: inout std_logic;
 Z: buffer std_logic);
end Circuit;
-- Mô tả hoạt động
architecture Behavioral of
Circuit is
 constant GND:
std_logic:='0';
 signal c: std_logic;
begin
 c<= A and B after delay;
 Y<=Y or Z; -- Y là inout
nên có thể --vừa ghi vừa
đọc được
 X<= c xnor Y;
end Behavioral;
```



+ Package và Package Body

Package (gói dữ liệu) là đơn vị thiết kế cơ bản dùng để chứa những khai báo cho các đối tượng, khai báo thủ tục ***procedure***, hàm function, kiểu dữ liệu, ***component*** có thể dùng chung cho những thiết kế, cấu trúc, dự án khác nhau…

Package Body là đơn vị thiết kế phụ thuộc được dùng để chứa những mô tả chi tiết cho các khai báo trong đơn vị thiết kế ***Package*** nào đó, mô tả chi tiết nội dung của các hàm, các thủ tục... ***Package Body*** thường được viết ngay sau ***Package***. Cú pháp chung các đơn vị thiết kế ***Package*** và ***Package Body:***

```
package My_Pack is
 constant...
 function bv_to_integer
(BV: bit_v..
  return integer
 component...
 subtype...
 end package My_pack;

package body My_Pack is
function bv_to_integer
(BV: bit_v..
  return integer is
 variable...
 begin
 for index in BV'range
 loop
 ....
 end function;
...
end My_Pack ;
```

```
-- Cách sử dụng
package trong file
mô tả VHDL.
library IEEE; --
Thư việc chuẩn
use
IEEE.std_logic_1164
.all;
...
-- Trong phần mềm
thiết kế ISE gói dữ
liệu do người sử
dụng tạo ra thường
được tổ chức mặc
định trong thư viện
"work"
```

+ Library (Thư viện)

Trong VHDL có các thư viện thiết kế chuẩn, ngoài ra người thiết kế có thể tạo các thư viện thiết kế riêng. Trong một thiết kế VHDL nhiều đoạn chương trình có thể được gọi từ các thư viện khác nhau.

Phân tích VHDL là một quá trình kiểm tra các đơn vị thiết kế VHDL để cho đúng cú pháp và ngữ nghĩa, các đơn vị thiết kế VHDL được lưu vào thư viện để sử dụng sau này. Thư viện thiết kế chứa các những phần tử thư viện sau:

Package: Chứa những mô tả khai báo được dùng chung.

Entity: Là những mô tả giao diện thiết kế được dùng chung.

Architecture: Những mô tả hoạt động thiết kế được dùng chung.

Configuration: Là những phiên bản của thực thể được dùng chung.

Các đơn vị thư viện là các cấu trúc VHDL có thể được phân tích riêng rẽ theo trình tự nhất định.

Trong VHDL có thư viện thiết kế đặc biệt có tên là "work". Khi người thiết kế biên dịch một chương trình viết trên VHDL nhưng không chỉ rõ thư viện đích, chương trình này sẽ được biên dịch và chứa vào thư viện "work".

Ví dụ cách gọi và sử dụng thư viện như sau:

```
library My_Lib ;
use My_Lib.Fast_Counters.all ;

entity Mod1 is
 port (...);
```

+ Configuration (Cấu hình)

Một thực thể có thể có một vài kiến trúc mô tả hoạt động cho nó. Trong quá trình thiết kế có thể phải thử nghiệm một vài biến thể của thiết kế bằng cách sử dụng các kiến trúc khác nhau. Cấu hình là thành phần cơ bản của đơn vị thiết kế. Cấu hình cho phép gắn các phiên bản của thực thể vào những kiến trúc khác nhau. Cấu hình cũng có thể được sử dụng để thay thế một cách nhanh chóng các phần tử của thực thể trong các biểu diễn cấu trúc của thiết kế.

Cú pháp của mô tả cấu hình như sau:

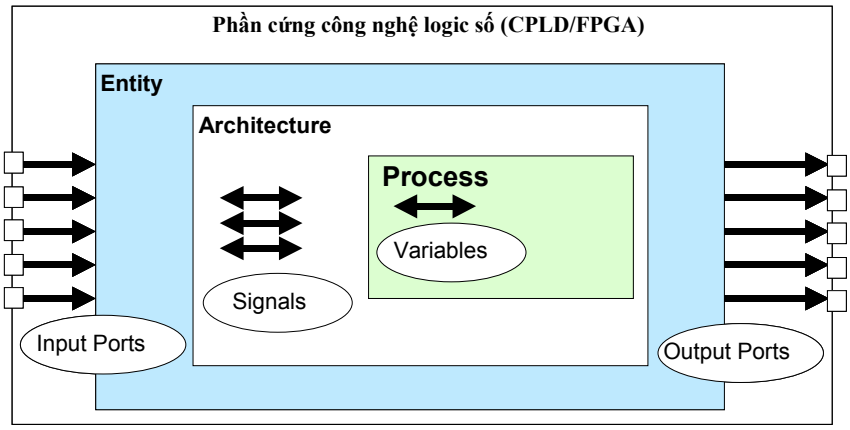
```
Configuration tên_cấu_hình of tên_thực_thể is
-- Phần khai báo của cấu hình (cho phép sử dụng
-- các phần tử trong package và library.
for đặc_tả_của_khối
{mệnh_đề_use}
{các_phần_tử_của_cấu_hình}
end for;
```

Ví dụ:

```
library ttl, work;
configuration v4_27_87 of processor is
use work.all;
for structure_view
for a1:alu
use configuration ttl.sn74ls181;
end for;
for m1,m2,m3: mux
use entity multiplex4 (behavior);
end for;
for all: latch -- use defaults
end for;
end for;
end configuration v4_27_87;
```

9.3.5 Cấu trúc chung của một chương trình mô tả VHDL

Mô hình cấu trúc mô tả phần cứng số và phạm vi sử dụng của các đối tượng trong VHDL có thể được tổng kết đơn giản như trong hình 9.2:



Hình 9.2: Cấu trúc mô tả phần cứng và các đối tượng trong VHDL

Sau đây là cấu trúc chung đơn giản của một chương trình mô tả VHDL:

```
-- Ví dụ cấu trúc 1 file mô tả cho một hệ thống phần cứng số dùng VHDL
-- Khai báo thư viện,(mặc định cần khai báo thư viện IEEE (thư viện
-- chuẩn đã được xây dựng).
library IEEE;...
-- Khai báo gói dữ liệu (package) trong thư viện cần sử dụng:
use IEEE.STD_LOGIC_1164.ALL;...
-- Khai báo thực thể
Entity Tên_thực_thể is
      -- Khai báo các tham số generic nếu cần:
 Generic(-- khai báo danh sách các tham số);
 Port(-- Khai báo danh sách các cổng vào/ra);
End Tên_thực_thể;
-- Bắt đầu viết
Architecture Tên_kiến_trúc of Tên_thực_thể is
{Khai báo:kiểu dữ liệu, các component,các đối tượng constant, signal}
Begin
      { Viết các mô tả dùng cấu trúc lệnh song song }
      ...
 Process(-- danh sách tín hiệu kích thích nếu cần)
      {Khai báo:kiểu dữ liệu, các đối tượng biến constant, variable }
 Begin
 { Viết các mô tả dùng cấu trúc lệnh tuần tự }
 End process;
      ...
 { Viết các mô tả dùng cấu trúc lệnh song song hay
```

```
process khác }
 ...
End Tên_kiến_trúc;
```

Ví dụ:

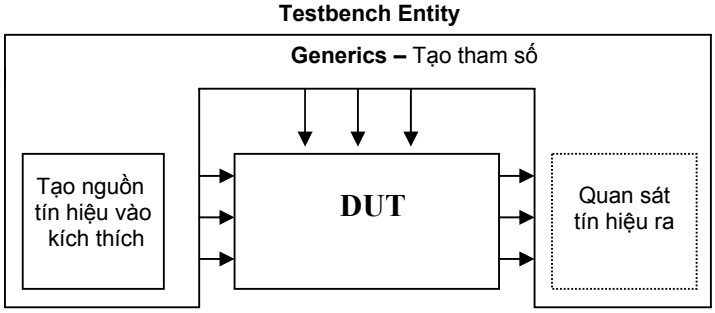
```
-- Khai báo thư viện chuẩn
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--------------------------------
-- Khai báo giao diện
entity Circuit is
 generic( delay: Time:=1ns);
      port(A, B: in std_logic ;
 X: out std_logic;
 Y: inout std_logic;
 Z: buffer std_logic) ;
end Circuit;
--------------------------------
-- Mô tả hoạt động
architecture Behavioral of Circuit is
 constant GND: std_logic:='0';
 signal c: std_logic;
begin
c<= A and B after delay;
Y<=(Y or Z) or GND; -- Y là inout nên có thể vừa ghi vừa
đọc được
X<= c xnor Y;
End Behavioral;
```

9.3.6 Môi trường kiểm tra "testbench"

Một trong các nhiệm vụ rất quan trọng là kiểm tra bản mô tả thiết kế. Kiểm tra một mô hình VHDL được thực hiện bằng cách quan sát hoạt động của nó trong khi mô phỏng và các giá trị thu được có thể đem so sánh với yêu cầu thiết kế.

Môi trường kiểm tra có thể hiểu như một mạch kiểm tra ảo. Môi trường kiểm tra sinh ra các tác động lên bản thiết kế và cho phép quan sát hoặc so sánh kết quả hoạt động của bản mô tả thiết kế. Thông thường thì các bản mô tả đều cung cấp chương trình thử. Nhưng ta cũng có thể tự xây dựng chương trình thử (testbench). Mạch thử thực chất là sự kết hợp của tổng hợp nhiều thành phần. Nó gồm ba thành phần. Mô hình VHDL cần kiểm tra, nguồn dữ liệu và bộ quan sát. Hoạt động của mô hình VHDL được kích thích bởi các nguồn dữ liệu và kiểm tra tính đúng đắn thông qua bộ quan sát. Hình 9.3 là sơ đồ tổng quát của một chương trình thử (Testbench).

Testbench được mô tả như một Entity không có đầu vào đầu ra, chỉ có tín hiệu bên trong được ghép tới khối DUT cần được kiểm tra theo kiểu cấu trúc. Người thiết kế sẽ mô tả các tín hiệu bên trong này tạo ra tín hiệu kích thích cho các đầu vào của DUT và đọc kết quả ra để quan sát... Để mô tả các tín hiệu vào kích thích thường dùng cách mô tả theo hành vi – behavioral.



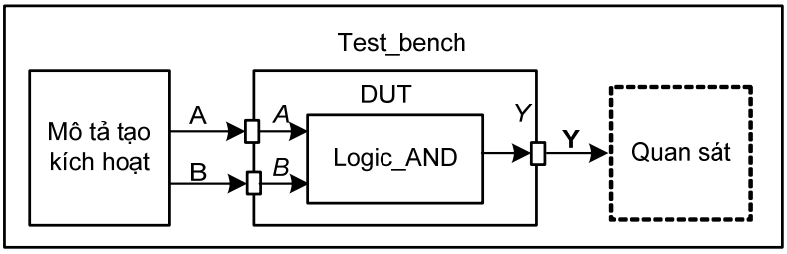
Trong đó: DUT: (device under test) mô hình VHDL cần kiểm tra

Hình 9.3: Sơ đồ tổng quát chương trình thử Testbench

Ví dụ mô tả cho mạch "Logic_AND" như sau:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
-- Khai báo giao diện
entity Logic_AND is
      port(A, B: in std_logic ;
 Y: out std_logic);
end Logic_AND;
-- Mô tả hoạt động
architecture Beh of Logic_AND is
begin
 Y<= A and B;
End Beh;
```

Viết Testbench cho thực thể "Logic_AND" đã mô tả ở trên:



Thực thể ***Test_bench*** không có các cổng vào/ra mà chỉ khai báo các tín hiệu nội bộ A, B, Y để nối tới khối DUT cần kiểm tra (Logic_AND). Trong phần kiến trúc mô tả hoạt động của Test_bench, coi khối Logic_AND như một component để tạo thành khối Test_bench. Toàn bộ mã mô tả cho Test_bench như sau:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-- Khai báo thực thể Test_bench;
```

```
ENTITY Test_bench IS
END Test_bench;
-- Mô tả kiến trúc của Test_bench
ARCHITECTURE behavior OF Test_bench IS
      COMPONENT Logic_AND
      PORT(
            A: IN std_logic;
            B: IN std_logic;
            Y: OUT std_logic
            );
      END COMPONENT;
      SIGNAL A: std_logic:='0';
      SIGNAL B: std_logic:='0';
      SIGNAL Y: std_logic;
BEGIN
-- Nối chân cổng vào/ra của DUT với các tín hiệu của
Test_bench
 Dut: Logic_AND PORT MAP(
            a => a,
            b => b,
            y => y
      );
 tb: PROCESS
 BEGIN
       -- Viết mô tả tạo kích thích
            A<= '1' after 10ns;
            B<= '1' after 20ns;
...
 END PROCESS;
-- *** End Test Bench - User Defined Section ***
END;
```

Trong các phần mềm thiết kế sau khi hoàn thành các mô tả cho Test_bench, người thiết kế sẽ chạy công cụ mô phỏng, các tín hiệu đầu ra của DUT sẽ được mặc tính đọc ra và cho phép người thiết kế quan sát dễ dàng dưới dạng giản đồ thời gian hay các file số liệu…

Người thiết kế có thể dễ dàng viết các mô tả kích thích để tạo ra các yêu cầu kiểm tra tùy ý cho bản thiết kế của mình. Nhiều chức năng mô phỏng, kiểm tra được hỗ trợ rất mạnh bởi các phần mềm thiết kế.

9.3.7 Các cấu trúc lệnh song song

Như đã trình bày trong phần cấu trúc chung của chương trình mô tả VHDL, trong mô tả một kiến trúc (Architecture) có chứa nhiều cấu trúc lệnh song song. Mỗi cấu trúc lệnh song song sẽ tương ứng với một thành phần phần cứng nào đó khi thực hiện tổng hợp mạch, mỗi cấu trúc song song có thể viết ở bất kỳ vị trí nào trong đoạn mô tả Architecture mà chức năng hoạt động của thực thể không thay đổi. Các cấu trúc lệnh song song có trong VHDL gồm:

+ Cấu trúc process.

+ Lệnh gán tín hiệu song song.

+ Lệnh gán có điều kiện.

+ Lệnh gán tín hiệu có lựa chọn.

+ Khối.

+ Phép gọi thủ tục, hàm song song.

9.3.7.1 Cấu trúc Process

Cấu trúc ***Process*** được tạo thành từ một tập hợp cấu trúc lệnh tuần tự (được trình bày chi tiết ở phần sau). Nó cũng là khối cơ bản của việc mô tả hoạt động của thực thể. Tất cả các ***Process*** trong một thiết kế được thực hiện song song. Mỗi một ***Process*** có thể tương

đương với một khối mạch nào đó. Chú ý là tại một thời điểm xác định chỉ có một câu lệnh tuần tự được thực hiện trong mỗi cấu trúc ***Process***. Cấu trúc tổng quát của Process như sau:

```
[Nhãn] Process [(Danh sách tín hiệu kích thích)]

[Khai báo:kiểu dữ liệu, các đối tượng biến constant,
variable]
 Begin
 {Viết các mô tả dùng cấu trúc lệnh tuần tự}
 End process;
```

Trong đó các phần đặt trong dấu [ ] có thể có hoặc không.

- Nhãn_lệnh: Tuỳ thuộc người thiết kế đặt tên.

- Danh sách tín hiệu kích thích: Danh sách các yếu tố kích thích hoạt động.

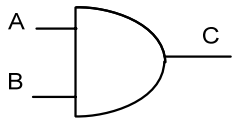
Nếu ***Process*** chứa (danh sách tín hiệu kích thích) thì lúc đó Process sẽ được thực hiện khi có bất kỳ sự thay đổi nào của bất kỳ tín hiệu nào trong danh sách tín hiệu kích thích. Điều này tương đương với ***Process*** không chứa danh sách tín hiệu kích thích nhưng lại chứa lệnh ***wait*** ở vị trí câu lệnh cuối cùng trong ***Process:***

```
Wait on <danh sách tín hiệu kích thích>
```

Khi tổng hợp mạch thì mỗi ***Process*** sẽ tương ứng với một khối mạch chức năng nào đó. Còn khi thực hiện mô phỏng, việc thực hiện một ***Process*** bao gồm việc thực hiện lặp lại các cấu trúc lệnh tuần tự chứa bên trong thân của ***Process***. Giống như một vòng lặp vô hạn và mỗi bước lặp được thực hiện mỗi khi có sự thay đổi của bất kỳ tín hiệu nào trong danh sách tín hiệu kích thích.

Ví dụ Process mô tả mạch logic AND như sau:

```
entity Logic_AND is
 Port ( A,B: in std_logic;
 C: out std_logic);
end Logic_AND;
architecture Behavioral of
Logic_AND is
begin
 Process(A,B)
 begin
       C<= A and B;
 end Process;
end Behavioral;
```

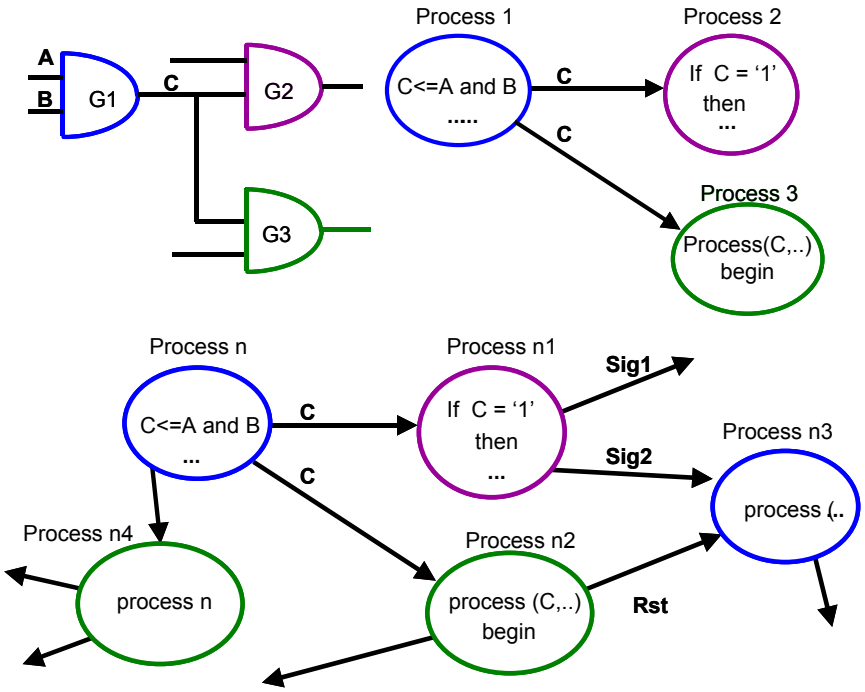


Khi dùng ***Process*** để mô tả khối mạch tổ hợp thì tất cả các tín hiệu vào của mạch phải có trong "danh sách tín hiệu kích thích".

Còn khi sử dụng Process để mô tả khối mạch có phần tử tuần tự (phần tử nhớ) thì nếu muốn thiết kế mạch hoạt động theo tín hiệu clock nào và không đồng bộ với tín hiệu vào nào (ví dụ hoạt động theo clk, không đồng bộ với tín hiệu Reset) thì những tín hiệu vào đó phải được đưa vào "danh sách tín hiệu kích thích".

Một ***Process*** liên kết với phần còn lại của thiết kế thông qua các thao tác đọc các giá trị từ các tín hiệu vào hay các cổng vào và ghi giá trị vào các tín hiệu ra hay cổng ra được khai báo ngoài ***Process***. Chú ý khi thiết kế là một tín hiệu có thể được đưa vào (được đọc giá trị trong Process) nhiều ***Process*** nhưng chỉ nên được ghi ra bởi một ***Process***.

Sự hoạt động đồng thời của mỗi ***Process*** và mô hình kết nối của chúng được mô tả như hình vẽ 9.4, trong đó tín hiệu sẽ truyền giá trị giữa những ***Process*** hoạt động đồng thời:



Hình 9.4: Mô hình kết nối của các Process

9.3.7.2 Các phép gán tín hiệu song song

Phép gán tín hiệu song song sử dụng bên trong các ***Architecture*** nhưng bên ngoài ***Process***. Dạng đơn giản nhất của phép gán tín hiệu song song có cú pháp như sau:

```
<tín_hiệu_đích><= <biểu_thức>[after<biểu_thức_thời_gian>];
```

Trong đó <tín_hiệu_đích> nhận giá trị của <biểu_thức>, chú ý là lệnh ***after*** chỉ dùng cho mô phỏng còn khi tổng hợp mạch nó sẽ được bỏ qua. Phép gán song song tương đương một Process chứa 1 phép gán tín hiệu.

Ví dụ mô tả mạch AND và OR có cùng 4 đầu vào như sau:

```
...
architecture Behavioral of logic1 is
 signal I1, I2, I3, I4, AND_out, OR_out: std_logic;
begin
...
 AND_out<= I1 and I2 and I3 and I4;
 OR_out<= I1 or I2 or I3 or I4;
...
end Behavioral;
```

Đoạn chương trình trên tương đương với đoạn chương trình VHDL với các ***Process*** chứa các phép gán tín hiệu tuần tự (lệnh tuần tự xem phần tiếp theo):

```
...
architecture Behavioral of logic1 is
 signal I1, I2, I3, I4, AND_out, OR_out: std_logic;
begin
...
 process(I1, I2, I3, I4)
 begin
            AND_out<= I1 and I2 and I3 and I4;
 end process;
 process(I1, I2, I3, I4)
 begin
            OR_out<= I1 or I2 or I3 or I4;
 end process;
...
end Behavioral;
```

9.3.7.3 Phép gán tín hiệu có điều kiện

Phép gán tín hiệu có điều kiện là cấu trúc lệch song song thực hiện phép gán giá trị của các biểu thức cho một tín hiệu đích tùy theo các điều kiện đặt ra. Cú pháp chung:

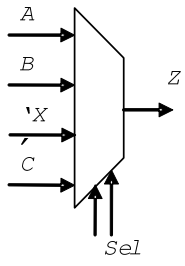
```
<tín_hiệu_đích> <= <biểu_thức>[after
<biểu_thức_thời_gian>] when
 <điều_kiện> else
       <biểu_thức>[after <biểu_thức_thời_gian>] when
 <điều_kiện> else
                  ...
 <biểu_thức>[after <biểu_thức_thời_gian>];
```

Cấu trúc phép gán tín hiệu có điều kiện có thể coi là cấu trúc song song của lệnh tuần tự ***If*** được thay thế tương đương với ***Process*** chứa lệnh tuần tự ***if.***

Ví dụ mô tả cấu trúc chọn kênh 4 đầu vào, 1 đầu ra Z và tín hiệu chọn kênh 2 bit Sel như sau:

```
architecture ...
 begin
   Z <= A when Sel="00"
else
    B when  Sel="10" else
    C when  Sel="11" else
    'X' ;
  end architecture;
```

```
architecture ...
 begin
 process  (A,B,C, SEL )
 begin
   case (SEL)  is
   when  "00" =>Z <= A;
   when  "10" =>Z <= B;
   when  "11" =>Z <= C;
   when others  =>Z<=
'X';
  end case;
 end process;
end architecture;
```



9.3.7.4 Phép gán tín hiệu theo lựa chọn

Phép gán tín hiệu theo lựa chọn thực hiện gán cho một tín hiệu đích với biểu thức ***with.*** Cấu trúc này có thể coi như là cấu trúc song song của lệnh tuần tự ***case,*** nó có thể thay thế tương đương với ***Process*** chứa lệnh tuần tự ***case.*** Cú pháp chung của lệnh ***with*** như sau:

```
With <biểu_thức_lựa_chọn> select
<tín_hiệu_đích> <= <biểu_thức> [after <biểu_thức_thời_gian>]
      when <giá_trị_lựa _chọn>,
       <biểu_thức> [after <biểu_thức_thời_gian>]
      when <giá_trị_lựa _chọn>,
      ...
      <biểu_thức> [after <biểu_thức_thời_gian>]
      when others;
```

Ví dụ mô tả cấu trúc chọn kênh 4 đầu vào, 1 đầu ra Z và tín hiệu chọn kênh 2 bit Sel như sau:

```
architecture...
with SEL select
 Z<= A when "00",
 B when "10",
 C when "11",
 'X' when others ;
end architecture;
```

↔

```
architecture...
 begin
 process (A,B,C, SEL)
 begin
 case SEL is
 when "00" => Z <= A ;
 when "10" => Z <= B ;
 when "11" => Z <= C ;
 when others => Z <= X';
 end case;
 end process;
end architecture;
```

9.3.7.5 Khối (Block)

Block bao gồm tập hợp các cấu trúc lệnh song song. Một kiến trúc có thể phân tách thành một số cấu trúc logic. Mỗi khối biểu diễn

một thành phần của mô hình và thường được sử dụng để tổ chức một tập hợp các cấu trúc song song phân cấp. Cú pháp chung:

```
<nhãn>: Block
            {<phần_khai_báo>}
begin
            {<câu_lệnh_song_song>} – có trình tự bất kỳ
end block;
```

<phần_khai_báo>: xác định các đối tượng tồn tại cục bộ trong khối và có thể là các khai báo sau:

- Khai báo hằng, kiểu dữ liệu, tín hiệu.
- Thân thủ tục.
- Khai báo component.
- Luật use – sử dụng trong ***Configuration.***

9.3.7.6 Gọi thủ tục, hàm song song

Phép gọi thủ tục, hàm song song tương đương với các Process bao gồm các phép gọi thủ tục tuần tự tương ứng. Mỗi phép gọi thủ tục, hàm tương đương với một ***Process*** không chứa dãy danh sách các tín hiệu kích thích, phần khai báo rỗng và phần thân chứa một phép gọi thủ tục, hàm, tiếp theo là một câu lệnh ***wait.*** Chi tiết hơn về thủ tục và hàm sẽ được trình bày trong các phần tiếp theo.

9.3.8 Cấu trúc lệnh tuần tự

Trong ngôn ngữ VHDL một cấu trúc song song quan trọng là ***Process.*** Cấu trúc này thường được sử dụng để mô tả hành vi hoạt động của mạch số. Trong kiến trúc, tất cả các **Process** sẽ được tổng hợp thành một khối mạch chức năng và thực hiện đồng thời khi mô phỏng. Một **Process** được xây dựng từ các cấu trúc lệnh tuần tự. Khi mô phỏng các lệnh tuần tự được thực hiện lần lượt trong một chu trình

vô hạn bắt đầu từ lệnh thứ nhất đến lệnh cuối và được kích hoạt trở lại thực hiện lệnh đầu mỗi khi có bất kỳ sự thay đổi nào trong danh sách tín hiệu kích thích hay trong danh sách tín hiệu trong câu lệnh ***wait.***

Các cấu trúc lệnh tuần tự cơ bản trong VHDL gồm:

+ Câu lệnh gán cho biến.

+ Câu lệnh gán cho tín hiệu.

+ Câu lệnh ***if.***

+ Câu lệnh ***case.***

+ Câu lệnh rỗng ***Null.***

+ Các lệnh lặp.

9.3.8.1 Phép gán biến

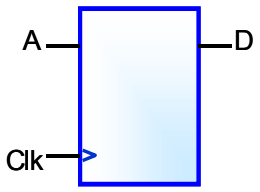
Cú pháp của phép gán biến như sau:

```
biến:= biểu_thức
```

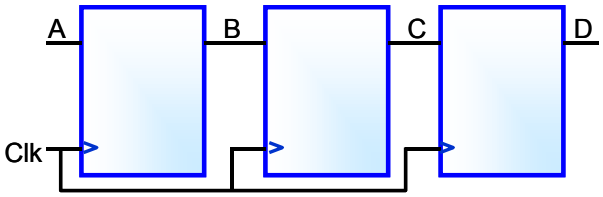
Phép gán biến được thực hiện với thời gian mô phỏng bằng 0 và giá trị biến sẽ được cập nhật ngay giá trị của biểu thức. Đối tượng <biến> chỉ được khai báo và sử dụng trong Process và thủ tục, hàm, nó được sử dụng để lưu trữ các kết quả trung gian.

So sánh kết quả mạch số tổng hợp được của mỗi ví dụ sau:

```
...
process( Clk )
 variable B, C, D: bit:= '1' ;
 begin
 If (Clk'event and Clk ='1') then
 B:= A ;
 C:= B ;
 D:= C ;
 end if ;
end process ;...
```



```
...
process( Clk )
 variable B, C, D: bit:= '1';
 begin
 If ( Clk'event and Clk ='1' ) then
 D:= C;
 C:= B;
 B:= A;
 end if;
end process ;
```



Chú ý trong 2 ví dụ trên, giá trị các biến được cập nhập tức thì, ví dụ thứ nhất khi tổng hợp mạch chỉ tạo ra một trigơ D. Còn với ví dụ thứ 2 thứ tự gán biến thay đổi, kết quả tạo ra 3 trigơ D. Trong ví dụ 1, nếu B,C,D là tín hiệu thì kết quả hoàn toàn khác. Xem ví dụ ở phần phép gán tín hiệu.

9.3.8.2 Phép gán tín hiệu

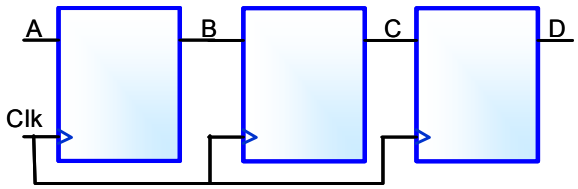
Cú pháp của phép gán biến như sau:

```
Tín_hiệu_đích<= biểu_thức [after giá_trị_thời_gian];
```

Khác với phép gán biến, phép gán tín hiệu trong ***Process*** không được cập nhập ngay tức thì mà phép gán đó chỉ được đặt kế hoạch thực hiện và kết quả chỉ được cập nhập sau khi kết thúc ***Process.***

Ví dụ:

```
Architecture Behavior of Trigo is
      signal Clk, A, B, C, D: bit:= '1';
Begin
 process( Clk )
 begin
 If (Clk'event and Clk ='1') then
 B <= A ;
 C <= B ;
 D <= C ;
 end if ;
end process ;
End Behavior;
```



9.3.8.3 Lệnh if

Lệnh này cho phép các phép toán được thực hiện trên một điều kiện nào đó. Có ba dạng cơ bản là:

+ Dạng 1:

```
if (Điều_kiện) then
      <Các_câu_lệnh_tuần_tự>;
end if;
```

+ Dạng 2:

```
if (Điều_kiện) then
      <Các_câu_lệnh_tuần_tự>;
else
      <Các_câu_lệnh_tuần_tự>;
end if;
```

+ *Dạng 3:*

```
if (Điều_kiện_1) then
        <Các_câu_lệnh_tuần_tự>;
elsif (Điều_kiện_2) then
        <Các_câu_lệnh_tuần_tự>;
elsif (Điều_kiện_3) then
        <Các_câu_lệnh_tuần_tự>;
else
        <Các_câu_lệnh_tuần_tự>;
end if;
```

Trong lệnh ***if/else***, ta phải chú ý một số điều sau:

+ Điều kiện đúng đầu tiên được tìm thấy sẽ được thực hiện.

+ Các điều kiện có thể chồng chất lên nhau.

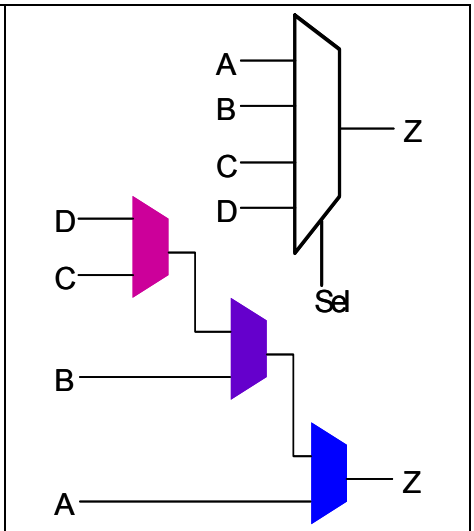
+ Điều kiện đầu tiên trong lệnh ***if/else*** được ưu tiên.

Ví dụ:

```
process (A, B, C, D, Sel)

begin
 If (Sel = "00") then
        Z <= A ;
 elsif (Sel = "01") then
        Z <= B ;
 elsif (Sel = "10") then
        Z <= C ;
 elsif (Sel = "11") then
        Z <= D ;
 end if;
end process ;
```

-- Với mô tả trên cấu trúc bên trong của mạch ghép kênh 4 đầu vào tổng hợp được thực sự được xây dựng từ 3 mạch ghép kênh 2 đầu vào.



9.3.8.4 Lệnh case

Lệnh ***case*** được sử dụng trong trường hợp có một biểu thức để kiểm soát nhiều rẽ nhánh trong chương trình VHDL. Các lệnh tương ứng với một trong các lựa chọn sẽ được thực hiệu nếu biểu thức kiểm soát có giá trị bằng giá trị tương ứng của lựa chọn đó. Có hai dạng cơ bản:

Dạng 1:

```
Case (biểu_thức_kiểm_soát) is
      When <giá_trị_lựa_chọn> =>
            <Các_câu_lệnh_tuần_tự>;
      When <giá_trị_lựa_chọn> =>
            <Các_câu_lệnh_tuần_tự>;
      ...
end case;
```

Dạng 2:

```
Case (selector expression) is
      When <giá_trị_lựa_chọn> =>
            <Các_câu_lệnh_tuần_tự>;
      When <giá_trị_lựa_chọn> =>
            <Các_câu_lệnh_tuần_tự>;
      ...
      When others =>
            <Các_câu_lệnh_tuần_tự>;
      end case;
```

Các chú ý khi dùng lệnh case:

+ Tất cả các giá trị của biểu thức lựa chọn phải được chỉ rõ.

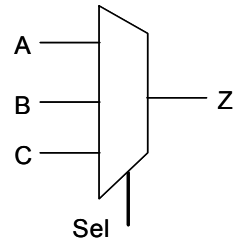
+ Không có các giá trị lựa chọn bị chồng chất lên nhau.

9.3.8.5 Câu lệnh rỗng Null

Câu lệnh rỗng có cú pháp như sau: ***Null;***

Trong VDHL khi chương trình mô phỏng gặp câu lệnh ***Null*** nó sẽ bỏ qua lệnh này và thực hiện lệnh tiếp theo sau. Thông thường lệnh ***Null*** dùng để chỉ trường hợp không thực hiện của lệnh một cách tường minh khi có các điều kiện trả lại giá trị true. Do đó lệnh ***Null*** thường được dùng trong các câu lệnh ***case*** đối với những giá trị lựa chọn không cần thao tác. Ví dụ:

```
process (A, B, C, D, Sel)
begin
case Sel is
 when "00" => Z <= A;
 when "01" => Z <= B;
 when "10" => Z <= C;
 when others => Null;
end case;
end process;
```



9.3.8.6 Các lệnh lặp

Lệnh lặp ***loop*** chứa thân vòng lặp bao gồm dãy các câu lệnh sẽ được thực hiện nhiều lần.

Cú pháp của lệnh lặp như sau:

```
[<nhãn>:] [<sơ_đồ_lặp>] loop
      {<lệnh_tuần_tự>}|
      {next [<nhãn>] [when <điều_kiện>];}|
      {exit [<nhãn>] [when <điều_kiện>];}
end loop [nhãn];
```

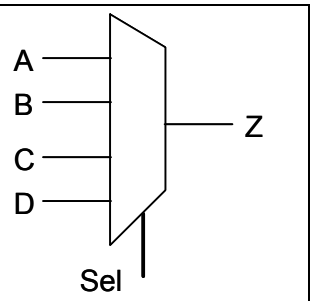
- <nhãn>: Nhãn của vòng lặp và thường được dùng để xây dựng những vòng lặp lồng nhau, trong đó mỗi vòng lặp được kết thúc bởi từ khóa ***end loop.***

- <sơ_đồ_lặp>: Vòng lặp với sơ đồ lặp ***for*** hoặc vòng lặp ***while*** và vòng lặp không chứa các sơ đồ lặp.

Với những vòng lặp không chứa [<sơ_đồ_lặp>], các lệnh trong dãy lệnh tuần tự sẽ được thực hiện cho tới khi được ngắt bởi câu lệnh ***exit.*** Trong đó câu lệnh ***next*** cũng được dùng để thay đổi trình tự thực hiện thân của vòng lặp.

Ví dụ vòng lặp không chứa sơ đồ lặp:

```
process (A, B, C, D, Sel)
begin
case Sel is
 when "00" => Z <= A ;
 when "01" => Z <= B ;
 when "10" => Z <= C ;
 when "11" => Z <= D ;
end case ;
end process ;
```

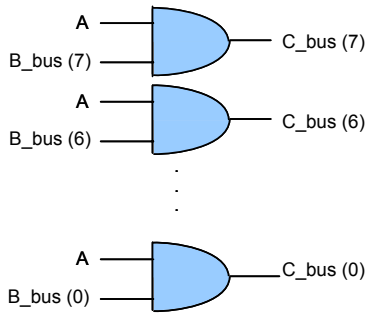


Ví dụ vòng lặp chứa sơ đồ lặp:

```
Count_down: Process
      Variable Min,Sec: integer range 0 to 60;
Begin
      L1: loop
            L2: loop
                  exit L2 when (Sec=0);
                  wait until CLK'event and CLK='1';
                  Sec:=Sec-1;
            End loop L2;
            Exit L1 when (Min=0);
            Min:=Min-1;
            Sec:=60;
      End loop L1;
End process Count_down;
```

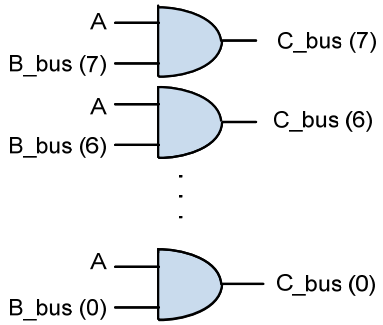
Ví dụ vòng lặp chứa <sơ_đồ_lặp> dạng ***for:***

```
process (A, B_bus)
begin
 for i in 7 downto 0
loop
 C_bus (i) <= A and
B_bus (i);
 end loop;
end process;
hoặc:
process (A, B_bus)
begin
 for i in 0 to 7 loop
 C_bus (i) <= A and
B_bus (i);
 end loop ;
End process;
```



Ví dụ vòng lặp chứa <sơ_đồ_lặp> dạng ***while*** như sau:

```
 process ( A, B_bus )
      variable
i:integer:=0;
begin
 while (i<8) loop
 C_bus (i) <= A and
B_bus (i);
 i:=i+1;
 end loop ;
End process;
```



9.3.9 Hàm và thủ tục

Thủ tục – Procedure và hàm – Function còn được gọi chung là chương trình con. Về mặt cấu trúc thì hàm và thủ tục khá giống với Process là chúng chỉ chứa các lệnh tuần tự. Về mặt ứng dụng thì giữa Process và chương trình con có sự khác nhau cơ bản là: Process được

mô tả ngay trong mã chương trình chính và không phải khai báo, còn chương trình con thường được khai báo và mô tả trong thư viện Library để được sử dụng chung và chia sẻ giữa các ứng dụng khác nhau, tuy nhiên chương trình con cũng có thể được khai báo và mô tả ngay trong mã chương trình chính nếu muốn.

9.3.9.1 Hàm (Function)

Hàm là một đoạn mã mô tả tuần tự, được sử dụng xây dựng các toán tử, phép toán mới cho các bài toán nào đó phải giải quyết mà hàm đó chưa có trong thư viện chuẩn. Các phép toán thường được viết là: hàm biến đổi dữ liệu, hàm logic, hàm số học, toán tử và hàm thuộc tính… Hàm được viết một lần và có thể được sử dụng lại nhiều lần trong 1 ứng dụng và chia sẻ giữa các ứng dụng, như vậy sẽ làm cho mã chương trình chính ngắn và dễ hiểu hơn.

Các cấu trúc lệnh tuần tự (như If, case, loop,.. trừ lệnh wait) có thể được sử dụng trong hàm. Chỉ có biến – Variable và hằng Constant và ***danh_sách_biến*** của hàm mới được khai báo và sử dụng trong hàm. Signal và Component bên ngoài ***không*** được phép sử dụng trong hàm.

Cú pháp để khai báo và mô tả hàm (phần Function body) như sau:

```
-- Khai báo
FUNCTION Tên_hàm [(danh_sách_biến)] RETURN kiểu_dữ_liệu;
```

```
-- Function Body (Mô tả hàm)
FUNCTION Tên_hàm [(danh_sách_biến)] RETURN kiểu_dữ_liệu
IS
 -- Khai báo CONSTANT, VARIABLE nếu có
BEGIN
 (Viết mô tả hàm dùng cấu trúc Lệnh tuần tự)
END Tên_hàm;
```

Trong đó: ***danh_sách_biến*** xác định tham số đầu vào cho hàm có thể được khai báo là đối tượng ***constant, signal*** của các kiểu dữ liệu có thể tổng hợp được như boolean, std_logic, integer, std_logic_vector…, nhưng không được sử dụng range với integer và không được sử dụng to/downto với kiểu dữ liệu vector.

Khi gọi hàm thì các đối tượng ***signal*** và ***constant*** sẽ được truyền vào hàm tương ứng theo ***danh_sách_biến.***

Chú ý: Hàm phải được trả về 1 giá trị duy nhất có kiểu_dữ_liệu như đã khai báo. Và trong ***danh_sách_biến*** từ khóa của hằng constant có thể được bỏ đi.

Ví dụ hàm xác định sườn dương của tín hiệu clk như sau:

```
------ Function body -------
FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN
BOOLEAN IS
BEGIN
 RETURN (s'EVENT AND s='1');
END positive_edge;
```

Khi sử dụng có thể gọi hàm trên như sau:

```
…
IF positive_edge(clk) THEN...
```

Ví dụ hàm biến đổi dữ liệu từ mảng nhị phân sang số nguyên như sau:

```
------ Function body -------------
FUNCTION conv_integer(SIGNAL vector: STD_LOGIC_VECTOR)
RETURN INTEGER IS
 VARIABLE result: INTEGER RANGE 0 TO 2**vector'LENGTH-
1;
BEGIN
IF (vector(vector'HIGH)='1') THEN result:=1;
```

```
    ELSE result:=0;
    END IF;
    FOR i IN (vector'HIGH-1) DOWNTO (vector'LOW) LOOP
     result:=result*2;
     IF(vector(i)='1') THEN result:=result+1;
     END IF;
    END LOOP;
    RETURN result;
END conv_integer;
```

Khi sử dụng có thể gọi hàm trên như sau:

```
Signal y: integer;
Signal a: Std_logic_vector (7 downto 0);
...
y<= conv_integer(a); -- Tín hiệu a được truyền vào hàm.
...
```

Hàm có thể được mô tả trong Package (+ Package body) của library hay trong chương trình chính.

Ví dụ cách mô tả hàm trong Package như sau:

```
-- Xây dựng PACKAGE
-- Khai báo thư viện chuẩn
LIBRARY ieee;
USE ieee.std_logic_1164.all;
---------------------------------------------
PACKAGE my_package IS
 -- Khai báo Hàm muốn mô tả
 FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN
BOOLEAN;
END my_package;
---------------------------------------------
PACKAGE BODY my_package IS
```

```
 FUNCTION positive_edge(SIGNAL s: STD_LOGIC) RETURN
BOOLEAN IS
 BEGIN
 RETURN s'EVENT AND s='1';
 END positive_edge;
END my_package;

-- Mã chương trình chính
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE work.my_package.all;
---------------------------------------------
ENTITY DFF IS
 PORT (d, clk, rst: IN STD_LOGIC;
 q: OUT STD_LOGIC);
END DFF;
 ---------------------------------------------
ARCHITECTURE my_arch OF dff IS
BEGIN
 PROCESS (clk, rst)
 BEGIN
 IF (rst='1') THEN q <= '0';
 ELSIF positive_edge(clk) THEN q <= d;
 END IF;
 END PROCESS;
END my_arch;
```

Nếu muốn mô tả hàm ngay trong chương trình chính thì toàn bộ hàm đó được viết trong phần khai báo của Architecture. Ví dụ cách dùng này như sau:

```
-- Khai báo thư viện chuẩn
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-- Khai báo thực thể
ENTITY DFF IS
 PORT ( d, clk, rst: IN STD_LOGIC;
 q: OUT STD_LOGIC);
END DFF;
--Mô tả hoạt động
ARCHITECTURE my_arch OF DFF IS
-- Phần khai báo của ARCHITECTURE
 FUNCTION positive_edge(SIGNAL s: STD_LOGIC)RETURN
BOOLEAN IS
 BEGIN
 RETURN s'EVENT AND s='1';
 END positive_edge;
-- Viết mô tả cho ARCHITECTURE
BEGIN
 PROCESS (clk, rst)
 BEGIN
 IF (rst='1') THEN q <= '0';
 ELSIF positive_edge(clk) THEN q <= d;
 END IF;
 END PROCESS;
END my_arch;
```

9.3.9.2 Thủ tục (Procedure)

Về cơ bản cú pháp, cách gọi, vị trí của thủ tục - Procedure tương tự như hàm – Function, nhưng thủ tục không được trả về giá trị.

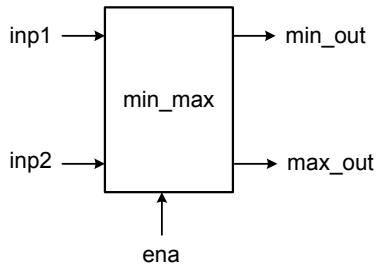
Cú pháp để khai báo và mô tả thủ tục (phần Procedure body) như sau:

```
-- Khai báo
PROCEDURE Tên_thủ_tục [(danh_sách_biến)];
```

```
-- PROCEDURE Body (Mô tả hàm)
PROCEDURE Tên_hàm [(danh_sách_biến)] IS
 -- Khai báo CONSTANT, VARIABLE nếu có
BEGIN
 (Viết mô tả thủ tục dùng cấu trúc Lệnh tuần tự)
END Tên_thủ_tục;
```

Ví dụ:

Thủ tục được viết ngay trong chương trình chính của ứng dụng mô tả bộ xác định giá trị nhỏ nhất min_out và giá trị lớn nhất max_out của 2 giá trị đầu vào inp1, inp2 như sau:




```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
-------------------------------------------------------
ENTITY min_max IS
 GENERIC (limit: INTEGER:= 255);
 PORT (ena: IN BIT; -- Tín hiệu cho phép enable
 inp1, inp2: IN INTEGER RANGE 0 TO limit;
 min_out, max_out: OUT INTEGER RANGE 0 TO limit);
END min_max;
 -------------------------------------------------------
ARCHITECTURE my_architecture OF min_max IS
```

```
 -- Phần khai báo của ARCHITECTURE
 PROCEDURE sort (SIGNAL in1, in2: IN INTEGER RANGE 0
TO limit;
  SIGNAL min, max: OUT INTEGER RANGE 0 TO limit) IS
 BEGIN
        IF (in1 > in2) THEN
        max <= in1;
        min <= in2;
        ELSE
        max <= in2;
        min <= in1;
        END IF;
 END sort;
 --------------------------
 BEGIN
 PROCESS (ena)
 BEGIN
 IF (ena='1') THEN sort(inp1, inp2, min_out, max_out);
 END IF;
 END PROCESS;
END my_architecture;
```

9.4 CÁC MỨC ĐỘ TRỪU TƯỢNG VÀ PHƯƠNG PHÁP MÔ TẢ HỆ THỐNG PHẦN CỨNG SỐ

Sử dụng VHDL cho phép mô tả hệ thống phần cứng số theo các mức độ trừu tượng khác nhau. Hình 9.5 trình bày các mức độ mô tả trừu tượng giảm dần khi sử dụng VHDL.

+ Mức mô tả theo mô hình hành vi *(Behavioral):* mức độ mô tả trừu tượng cao nhất, kiểu mô tả này thường dùng cho mô hình phần cứng dùng cho mô phỏng.

+ Mức mô tả theo mô hình luồng dữ liệu *RTL* (Register Tranfer Level) – gọi tắt là mô hình *RTL:* Kiểu mô tả hệ thống theo từng thành

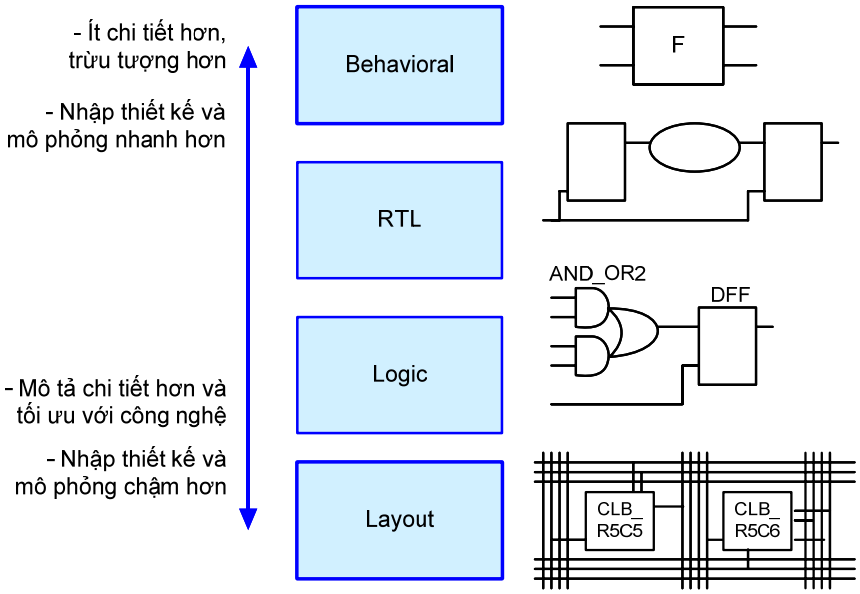
phần cấu trúc mạch tổ hợp và mạch tuần tự, kiểu mô tả này khá tối ưu và cho khả năng tổng hợp cao, độc lập với công nghệ.

+ Mức mô tả theo mô hình cấu trúc *logic:* Kiểu mô tả này thường sử dụng các cấu trúc logic đã được xây dựng sẵn hoặc chọn trong thư viện của nhà cung cấp phù hợp với loại công nghệ sử dụng.

+ Mức mô tả theo cấu trúc *layout.* Mức độ mô tả chi tiết nhất, mô tả chi tiết tới cấu trúc bên trong những tài nguyên đã sẵn có trong cấu kiện, cách này tối ưu cho việc tổng hợp trên loại cấu kiện, công nghệ đã sử dụng.

Trong giáo trình này các mã mô tả VHDL chỉ sử dụng theo 3 mức: hành vi, RTL và cấu trúc logic.

Chú ý: Trong một chương trình mô tả có thể chỉ sử dụng theo một cách mô tả hoặc cũng có thể phải dùng kết hợp cả 3 cách tùy theo độ phức tạp của hệ thống phần cứng số, yêu cầu về thời gian mô tả và thiết kế, yêu cầu về sự tối ưu phần cứng…



Hình 9.5: Các mức độ mô tả hệ thống phần cứng số

9.4.1 Phương pháp mô tả theo mô hình cấu trúc logic

Mô hình cấu trúc của một phần tử (hoặc hệ thống) có thể bao gồm nhiều cấp cấu trúc như bắt đầu từ một cổng logic đơn giản đến xây dựng mô tả cho một hệ thống hoàn thiện. Thực chất của việc mô tả theo mô hình cấu trúc là mô tả các phần tử con bên trong hệ thống và sự kết nối của các phần tử con đó. Cách thức mô tả cấu trúc của thành phần con cũng tương tự như cách thức mô tả thực thể. Trước hết để mô tả cấu trúc của thành phần con, chúng ta phải xác định rõ các giao diện của thành phần con. Các giao diện này chính là các đường tín hiệu vào và ra từ thành phần con.

Trước khi được sử dụng trong kiến trúc của cả hệ thống, các thành phần đã được xây dựng (gọi tắt là các *component*) phải được khai báo một cách tường minh theo cú pháp sau:

```
Component <tên_thành_phần>
      Port(<khai_báo_danh_sách_các_cổng_cục_bộ;>)
      -- Tương tự như khai báo trong thực thể
End component;
```

Chú ý: Các cổng vào/ra của mỗi thành phần con không được kết nối trực tiếp với nhau mà phải kết nối thông qua tín hiệu nội bộ có cùng kiểu, cùng độ lớn với các cổng vào/ra đó.

Cú pháp mô tả móc nối giữa các thành phần con như sau:

```
<nhãn_khởi_tạo>:<tên_thành_phần>
port map ([<tên_cổng_cục_bộ> =>] <biểu_thức>
  {[<tên_cổng_cục_bộ>=>]<biểu_thức>});
```

Cấu trúc *port map* ánh xạ các cổng của phần tử vào các tín hiệu. Ánh xạ này có thể hiểu như việc kết nối cổng tương ứng của phần tử vào đường tín hiệu. Cấu trúc *port map* đặt tương ứng mỗi cổng thực của phiên bản với một cổng cục bộ thành phần. Thực hiện nối các

cổng vào/ra của các thành phần con với các chân vào/ra của hệ thống hoặc nối với tín hiệu nội bộ trong hệ thống để kết nối tới các cổng vào/ra của các thành phần con khác. Ánh xạ được thực hiện theo vị trí theo tên:

+ Khi sử dụng ánh xạ theo vị trí, chúng ta đưa ra danh sách các tín hiệu tuân theo đúng trật tự mà cổng được khai báo.

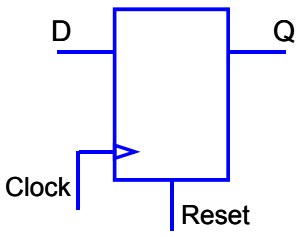
+ Đối với trường hợp ánh xạ theo tên, chúng ta sử dụng cấu trúc ánh xạ tường minh đặt tương ứng với mỗi cổng với các tín hiệu thực:

```
<tên_cổng_cục_bộ> => <tên_tín_hiệu_thực>
```

Ví dụ: Mô tả mô hình cấu trúc một thanh ghi 4 bít được xây dựng từ 4 trigơ DFF. Có thể mô tả trigơ DFF sau đó sau đó mô tả sơ đồ móc nối các phần tử trigơ DFF tạo thành thanh ghi.

Mô tả trigơ DFF như sau:

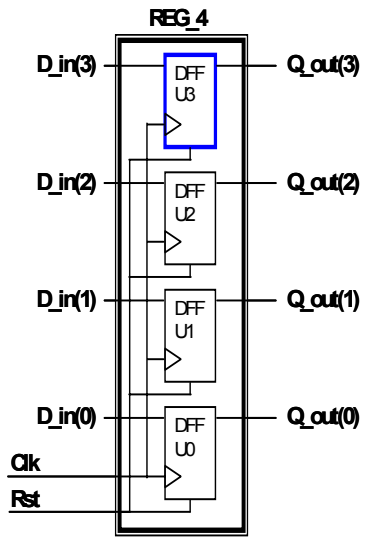
```
entity DFF is
 port ( D, Clock: in std_logic ;
 Reset: in std_logic ;
 Q: out std_logic) ;
end entity DFF ;
architecture RTL of DFF is
begin
 process (Clock, Reset)
 begin
 If (Reset = '1' ) then Q <= '0' ;
 elsif (Clock'event and Clock =
'1') then
 Q <= D ;
 end if;
 end process ;
```



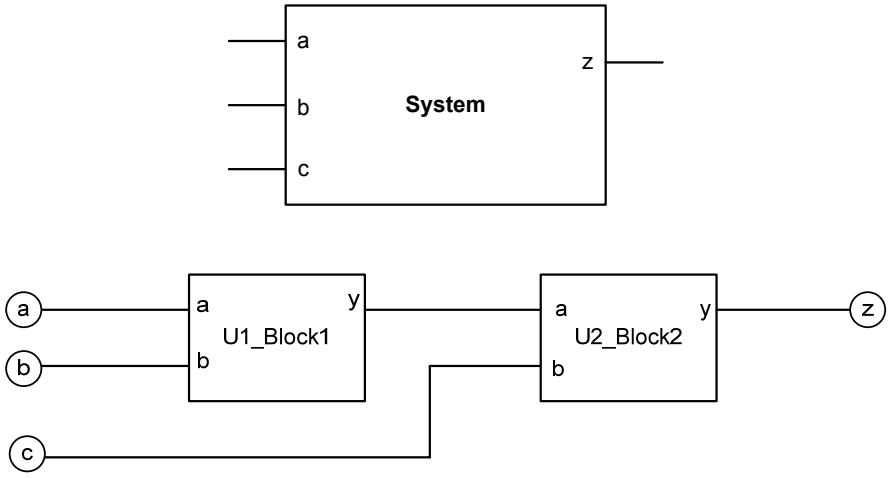
Như vậy trigơ DFF được mô tả ở trên là dạng trigơ có Reset mức tích cực cao và không đồng bộ.

- Ví dụ: Mô tả cấu trúc của thanh ghi 4 bit gồm 4 trigơ D ở trên theo phương pháp cấu trúc như sau:

```
entity REG_4 is
 port (D_in: in
std_logic_vector
  (3 downto 0);
 Clk, Rst: in std_logic;
 Q_out: out std_logic_vector
  (3 downto 0));
end REG_4;
-------------------------
architecture Structural of
REG_4 is
-- Khai báo component cần sử
dụng
 component DFF
       port ( D,Clock: in
std_logic;
  Reset: in std_logic;
  Q: out std_logic);
 end component;
begin
-- Ánh xạ theo vị trí:
U3:DFF port map(D_in(3), Clk,
Rst, Q_out(3));
U2:DFF port map(D_in(2), Clk,
Rst, Q_out(2));
U1:DFF port map(D_in(1), Clk,
Rst, Q_out(1));
-- Ánh xạ theo tên:
U0:DFF port map(Clock =>Clk, D
=>D_in(0),
  Reset =>Rst,Q =>Q_out(0));
end Structural;
```



Ví dụ: Mô tả hệ thống số "System" theo phương pháp cấu trúc từ các khối con như sau:




```
-- Block1.vhd: File mô tả khối con thứ nhất
...
entity Block1 is
 Port ( a: in STD_LOGIC;
 b: in STD_LOGIC;
 y: out STD_LOGIC);
end Block1;
architecture Behavioral of Block1 is
begin
 y<= (not a and b) or (a and not b);
end Behavioral;
```

```
-- Block2.vhd: File mô tả khối con thứ 2
...
entity Block2 is
 Port ( a: in STD_LOGIC;
 b: in STD_LOGIC;
 y: out STD_LOGIC);
end Block2;
architecture Behavioral of Block2 is
begin
 Y<= a xnor b;
end Behavioral;
```

```
-- System.vhd: Mô tả hệ thống "System" theo mô hình cấu trúc
...
entity System is
 Port ( a,b,c: in STD_LOGIC;
 z: out STD_LOGIC);
end System;
architecture Behavioral of System is
      -- Khai báo các khối con sẽ dùng trong hệ thống
 COMPONENT Block1
 PORT(
            a,b: IN std_logic;
            y: OUT std_logic
            );
 END COMPONENT;
 COMPONENT Block2
      PORT(
            a,b: IN std_logic;
            y: OUT std_logic
            );
```

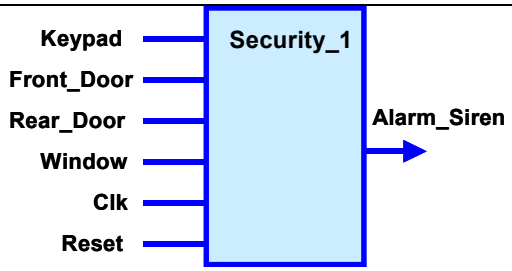
```
      END COMPONENT;
      signal x: std_logic; -- Tín hiệu nội bộ dùng để
nối giữa các khối
begin
 U1_Block1: Block1 port map(a,b,x);
 U2_Block2: Block2 port map(a=>x,b=>c,y=>z);
end Behavioral;
```

9.4.2 Phương pháp mô tả theo mô hình hành vi (Behavioral):

Đây là mức độ mô tả trừu tượng nhất, chủ yếu là mô tả theo chức năng của hệ thống số theo yêu cầu đầu vào và đáp ứng ra sử dụng các cấu trúc lệnh giống như của ngôn ngữ lập trình bậc cao như Process, wait, If, case, for-loop… Mô tả theo cách này tính ngữ nghĩa tự nhiên và giải thuật rất cao, nhập thiết kế rất nhanh, nhưng cấu trúc của phần cứng thường không rõ. Với những hệ thống phức tạp (nhất là có giải thuật phức tạp), yêu cầu cần thiết kế nhanh mà không cần yêu cầu về mức độ tối ưu phần cứng cao thường dùng cách mô tả này. Người thiết kế chỉ mô tả chức năng, hành vi được mong đợi của thiết bị dùng VHDL từ những mô tả hệ thống dạng văn bản và hay lưu đồ thuật toán. Phương pháp mô tả này rất hay dùng cho mục đích mô phỏng.

Ví dụ: Mô tả hệ thống cảnh báo theo mô hình hành vi "Hệ thống gồm có đầu vào từ các bộ cảm biến (*Front_Door, Rear_Door, Window*), đầu vào từ bàn phím bấm Keypad, tín hiệu *Clk, Reset* và đầu ra điều khiển còi báo động *Alarm_Siren*".

Chức năng hoạt động của hệ thống như sau: Nếu mỗi khi có một sensor nào đó được kích hoạt, thì hệ thống kiểm tra mã bàn phím. Nếu sau 20 giây mà không có mã bàn phím nhập đúng nhập vào thì còi báo động sẽ được bật lên.




```
entity Security_1 is
port (Clk, Reset: in std_logic ;
 Keypad: in std_logic_vector (3 downto 0) ;
 Front_Door, Rear_Door, Window: in boolean ;
 Alarm_Siren: out boolean ) ;
end Security_1 ;
-------------------------------
architecture Behavioral of Security_1 is
      constant Delay_Period: time:= 20 s;
begin
 process (Keypad,Front_Door,Rear_Door,Window)
 begin
 if (Front_Door or Rear_Door or Window ) then
 if (Keypad = "0011") then
 Alarm_siren <= false ;
 else
 Alarm_Siren <= true after Delay_Period ;
 end if ;
 end if ;
 end process ;
end Behavioral;
```

9.4.3 Phương pháp mô tả theo mô hình luồng dữ liệu RTL

Hệ thống được biểu diễn theo mô hình RTL bao gồm tập các thanh ghi (khối mạch có nhớ) và các phép toán được thực hiện trên dữ liệu số nhị phân được lưu trong các thanh ghi. Luồng dữ liệu và việc

xử lý dữ liệu thực hiện trên số liệu được chứa trong các thanh ghi được coi như là hoạt động chuyển đổi giữa các thanh ghi. Ví dụ: Mô hình RTL này được sử dụng để biểu diễn cấu trúc bộ vi xử lý. Hệ thống số được biểu diễn theo mô hình RTL khi chúng được xác định bởi 3 thành phần như sau:

- Tập các thanh ghi trong hệ thống (Các khối mạch nhớ, mạch tuần tự).

- Các phép toán được thực hiện trên dữ liệu được lưu trong các thanh ghi được xây dựng nhờ các mạch logic tổ hợp.

- Những điều khiển để giám sát chuỗi tuần tự các phép toán trong hệ thống (thường được xây dựng trên mô hình máy trạng thái).

Thanh ghi gồm nhóm các Trigơ chứa dữ liệu nhị phân. Một thanh ghi có thể nạp thông tin mới, dịch thông tin… Một bộ đếm được coi như là một thanh ghi có khả năng tăng, giảm giá trị tuần tự. Một Trigơ có thể coi như là thanh ghi 1 bit. Phần mạch gồm có các Trigơ và các cổng liên quan trong bất cứ mạch tuần tự nào có thể được gọi là những thanh ghi.

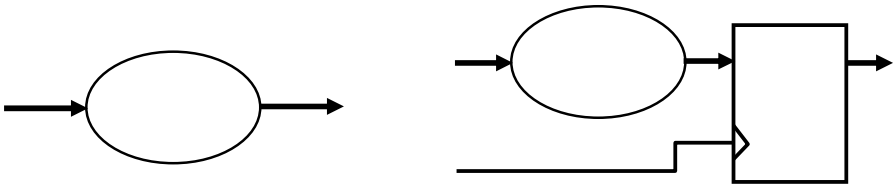
Các phép toán được thực hiện trên dữ liệu chứa trong các thanh ghi là những phép toán cơ bản có thể được thực hiện song song trên chuỗi bit trong một chu kỳ clock. Kết quả của phép toán có thể thay thế dữ liệu trước đó của thanh ghi hoặc kết quả có thể được chuyển đến thanh ghi khác. Có 4 kiểu phép toán như sau:

+ Phép chuyển đổi: Truyền dữ liệu từ thanh ghi này sang thanh ghi khác.

+ Phép toán số học.

+ Phép toán logic.

+ Phép dịch.



Combinatorial **Clocked Process**

Điều khiển khởi tạo chuỗi các phép toán bao gồm tín hiệu định thời cho phép thực hiện tuần tự các phép toán theo cách đã được mô tả trước. Có thể coi mô hình RTL là mô hình mô tả hành vi theo từng xung clock của hệ thống số.

Hệ thống số được mô tả bằng VHDL theo mô hình RTL có khả năng tổng hợp rất cao và rất dễ dàng trong việc trao đổi giữa các công cụ tổng hợp, thiết kế và có thể tổng hợp trên các công nghệ PLD khác nhau.

Tóm lại theo mô hình RTL hệ thống số được xây dựng trên cơ sở các khối thanh ghi và các khối mạch logic tổ hợp và chúng được mô tả bằng các tiến trình tổ hợp *(combinatorial process)* và các tiến trình hoạt động theo clock *(clocked process)*.

9.3.4.1 Mô tả mạch logic tổ hợp

Mạch logic tổ hợp có thể mô tả bằng các cấu trúc lệnh song, tuy nhiên thường dùng các *process* tổ hợp. Khi sử dụng *process* tổ hợp ***tất cả các tín hiệu vào*** của mạch tổ hợp phải được đưa vào danh sách tín hiệu kích thích.

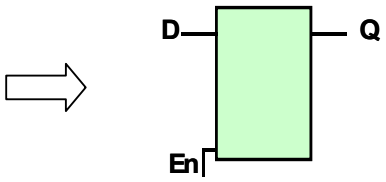
Ví dụ tiến trình tổ hợp như sau:

```
process(A,B)
begin
 Z <= A or B ;
end process;
```

```
process (D, En)
begin
-- gán mặc định đầu ra
 Q <= 0;
 if En = '1' then Q <= D ;
 end if ;
end process;
```

Chú ý: trong các *process* tổ hợp nên có phép gán giá trị mặc định cho đầu ra để tránh trường hợp mạch bị biến thành mạch chốt theo mức.

```
-- Trong process không
gán giá trị mặc định
process (D, En)
begin
 if En = '1' then Q <= D
;
 end if ;
end process ;
```



(T□ng h□p thành m□ch ch□t theo m□c)

Chú ý: Khi mô tả mạch logic tổ hợp các biến được khai báo trong *process* không được gán giá trị mặc định trước vì để mạch tổng hợp được không chứa các phần tử nhớ. Nếu trong mã mô tả có các biến được gán giá trị mặc định trước thì chương trình tổng hợp sẽ tạo ra các phần tử nhớ để lưu trữ các giá trị khởi tạo, mạch trở thành mạch có nhớ.

Mọi câu lệnh tuần tự trừ các lệnh *wait, loop, if* với những tín hiệu điều khiển theo sườn đều có thể dùng để mô tả các mạch tổ hợp. Các phép toán số học, logic, quan hệ đều có thể được sử dụng trong biểu thức.

9.3.4.2 Mô tả mạch tuần tự

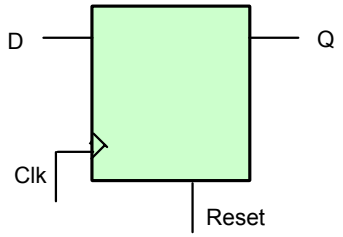
Các khối thanh ghi có thể được mô tả bằng tiến trình hoạt động theo clock theo 2 kiểu:

- Tiến trình đồng bộ với danh sách tín hiệu kích thích chỉ có duy nhất tín hiệu clock, mọi biến đổi của mạch được đồng bộ theo sườn clock)

- Hoặc tiến trình không đồng bộ với danh sách tín hiệu kích thích không chỉ có tín hiệu clk mà còn có các tín hiệu không đồng bộ khác.

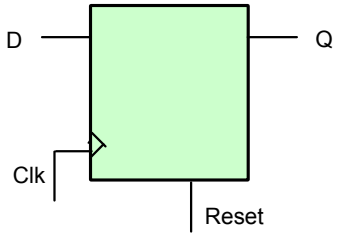
Ví dụ: Mô tả hoạt động của trigơ D làm việc theo sườn dương với các tín hiệu Reset không đồng bộ như sau:

```
process ( Clk, Reset )
 begin
 if reset = '1' then
 Q <= '0' ;
 elsif (Clk`event and Clk =
'1') then
 Q <= D ;
 end if ;
end process ;
```



Ví dụ: Mô tả hoạt động của trigơ D làm việc theo sườn dương với các tín hiệu Reset đồng bộ như sau:

```
process (Clk)
 begin
 if (Clk`event and Clk =
'1') then
      if reset = '1' then Q
<= '0' ;
      elsif then Q <= D ;
      end if ;
 end if ;
end process ;
```



Ví dụ: Mô tả hoạt động của Bộ đếm nhị phân tiến 4 bit đầu ra Q [3:0] hoạt động với sườn dương clock, có nạp đồng bộ giá trị cố định "1010" mức tích cực cao

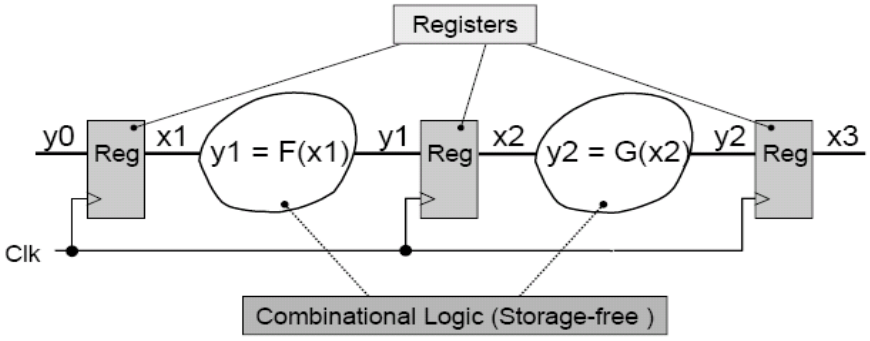
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity counter is
 port( Clk, SLOAD: in std_logic;
 Q: out std_logic_vector(3 downto 0));
end counter;
architecture archi of counter is
 signal tmp: std_logic_vector(3 downto 0);
 begin
 process (Clk)
 begin
 if (Clk'event and Clk='1') then
 if (SLOAD='1') then tmp <= "1010";
 else tmp <= tmp + 1;
-- Không thể gán trực tiếp Q<=Q+1,vì Q là tín hiệu ra
nên không được phép đọc vào ở biểu thức (Q+1).
 end if;
 end if;
 end process;
 Q <= tmp;
end archi;
```

Ví dụ: Mô tả hoạt động của Bộ đếm nhị phân lùi 4 bit đầu ra Q [3:0] hoạt động với sườn âm clock, có nạp không đồng bộ giá trị cố định "1111" mức tích cực thấp:

```
entity counter is
 port( Clk, S: in std_logic;
 Q: out std_logic_vector(3 downto 0));
end counter;
architecture archi of counter is
 signal tmp: std_logic_vector(3 downto 0);
 begin
 process (Clk,S)
 begin
 if (S='0' then tmp<= "1111";
 else if (Clk'event and Clk='0') then
```

```
 tmp <= tmp - 1;
 end if;
 end if;
 end process;
 Q <= tmp;
End archi;
```

Tóm lại biểu diễn hệ thống số theo mô hình RTL cần sử dụng các cấu trúc thanh ghi (Registers) và mạch tổ hợp (combinational logic). Một mô hình RTL điển hình chính là mô hình đường dữ liệu "datapath" như hình 9.6.



Hình 9.6: Một mô hình RTL

Mô tả VHDL cho mô hình đường dữ liệu ở trên trên có thể thực hiện theo 2 cách như sau:

Mô tả thành 2 process độc lập cho mạch tuần tự và mạch tổ hợp độc lập:

```
architecture SPLIT of DATAPATH is
      signal X1, Y1, X2, Y2:...
begin
REG: process (CLK)
begin
if (CLK'event and CLK = '1') then -- Resisters
 X1 <= Y0;
```

```
 X2 <= Y1;
 X3 <= Y2;
end if;
end process;
LOGIC: process (X1, X2) -- combinational logic
begin
 Y1 <= F(X1);
 Y2 <= G(X2);
end process;
end SPLIT;
```

- Cách viết kết hợp mạch tổ hợp và tuần tự trong một process:

```
architecture COMBINED of DATAPATH is
      signal X1, X2:...
begin
process (CLK) -- Registers
begin
if (CLK'event and CLK = '1') then
-- Khối mạch tổ hợp được tổng hợp từ biểu thức F,G.
 X2 <= F(X1);
 X3 <= G(X2);
 X1 <= Y0;
end if;
end process;
end COMBINED;
```

9.4.4 Phương pháp mô tả theo mô hình đồ hình trạng thái (máy trạng thái - State Machine)

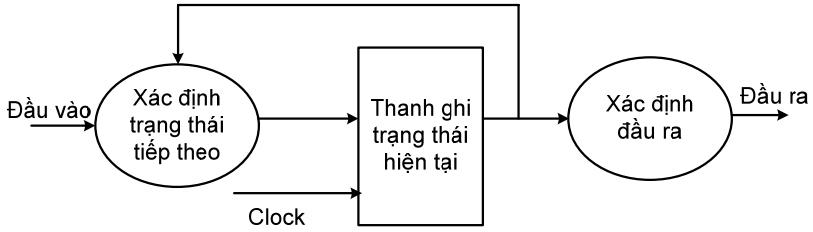
Hoạt động của một hệ thống số tuần tự có thể được mô tả dưới dạng đồ hình trạng thái Moore hoặc Mealy. Dùng VHDL có thể mô tả được đồ hình chuyển đổi trạng thái đó.

Bảng sau cho biết khả năng mô tả đồ hình trạng thái dùng VHDL:

STT	Yêu cầu mô tả	Sử dụng cấu trúc trong VHDL
1	Trạng thái logic hiện tại	**Process** hoạt động theo clock
2	Xác định trạng thái logic tiếp theo	**Process** tổ hợp
3	Xác định đầu ra	**Process** tổ hợp
4	Đặt tên cho các trạng thái	Kiểu dữ liệu liệt kê
5	Đánh giá mỗi trạng thái	Lệnh **Case**
6	Đánh giá các điều kiện đầu vào	Lệnh **if/else**

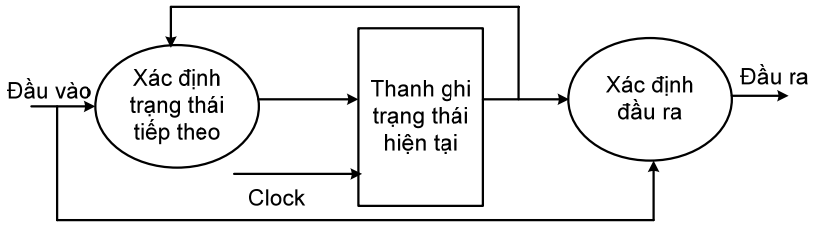
Tổng kết lại các kiểu đồ hình trạng thái có mô hình mạch số được xây dựng từ các khối mạch tổ hợp và khối mạch tuần tự như sau:

- *Mô hình Moore:* Kết quả đầu ra chỉ phụ thuộc vào trạng thái hiện tại.



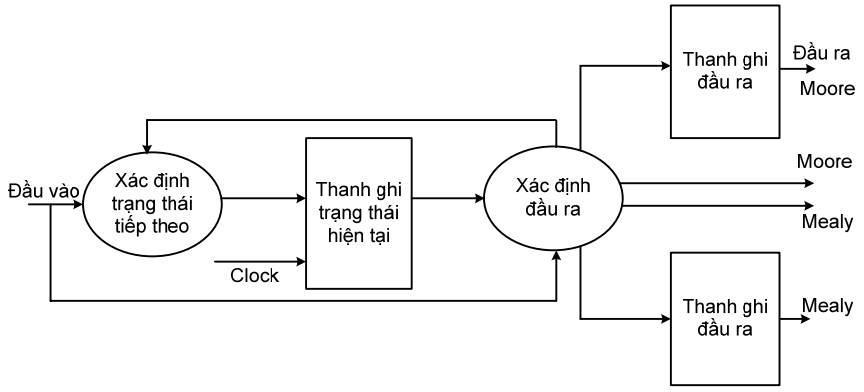
Hình 9.7: Mô hình máy trạng thái Moore

- *Mô hình Mealy:* Đầu ra phụ thuộc vào cả trạng thái hiện tại và tín hiệu vào.



Hình 9.8: Mô hình máy trạng thái Mealy

Trong thực tế hệ thống số thường được mô tả bằng việc kết hợp cả mô hình Moore và Mealy và sử dụng thêm thanh ghi đầu ra:



Hình 9.9: Mô hình máy trạng thái hỗn hợp Moore và Mealy

Cách sử dụng kiểu dữ liệu liệt kê để đặt tên cho các trạng thái như sau:

```
architecture RTL of FSM is
...
type My_State is ( Init, Load, Fetch, Stor_A, Stor_B);

signal Current_State, Next_State: My_State;
...
Begin
...
End RTL;
```

- Cách sử dụng hằng để mã hóa các trạng theo như mong muốn như sau:

```
subtype My_State is std_logic_vector( 0 to 5 ) ;
constant Init : My_State:= "111000" ;
constant Load : My_State:= "101010" ;
constant Init : My_State:= "000011" ;
signal Curr_State, Next_State: My_State ;
...
begin
...
```

- Để mô tả quá trình chuyển đổi trạng thái và cập nhật kết quả đầu ra ứng với mỗi trạng thái thông thường sử dụng cách mô tả bằng nhiều tiến trình

+ Tiến trình cập nhập trạng thái mới của hệ thống (tiến trình Sync).

```
Sync: process ( CLK, RST)
 begin
      ...
end process Sync ;
```

+ Tiến trình kiểm tra điều kiện chuyển đổi trạng thái (tiến trình Comb).

```
Comb: process ( Curr_State, In1, In2…)
 begin
      ...
end process Comb ;
```

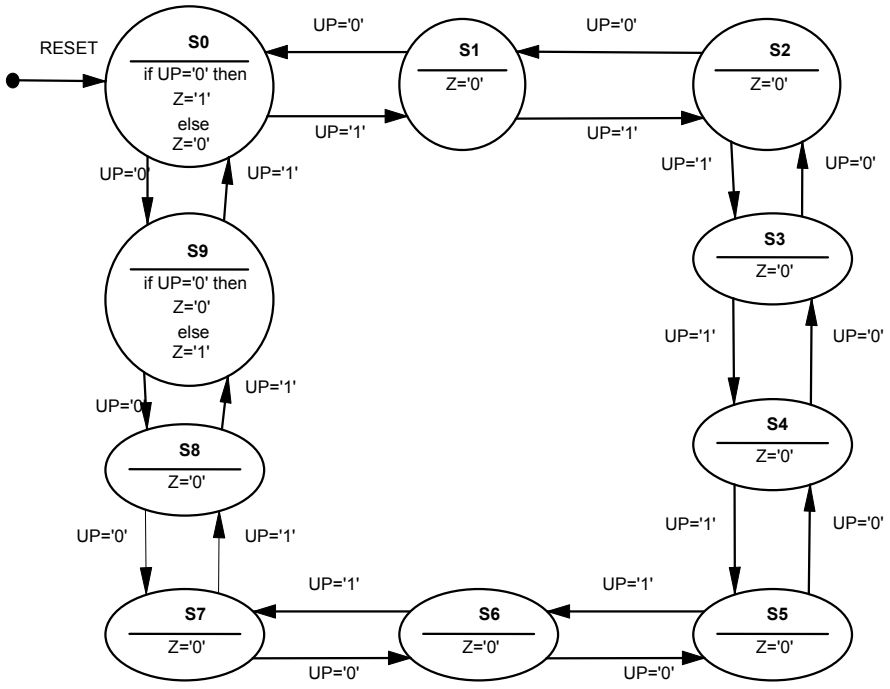
+ Tiến trình cập kết quả đầu ra ứng với mỗi trạng thái *(tiến trình Outputs).*

```
Outputs: process ( Curr_State, In1, In2…)
 begin
      ...
end process Outputs ;
```

Ví dụ: Bộ đếm thập phân thuận nghịch đồng bộ "Updown Counter" có mô hình vẽ sau:

Hình 9.10: Mô hình bộ đếm thuận nghịch

Hoạt động của bộ đếm có thể được mô tả theo đồ hình trạng thái (mô hình máy trạng thái Mealy) như hình 9.11.



Hình 9.11: Đồ hình trạng thái của bộ đếm thập phân thuận nghịch

Bộ đếm có tín hiệu RESET đồng bộ, mức tích cực cao: Khi RESET = '1' bộ đếm sẽ được xóa về trạng thái S0.

Khi UP='1': Bộ đếm thực hiện đếm tiến (đếm thuận), ngược lại UP='0' bộ đếm thực hiện đếm lùi (đếm nghịch).

Mô tả VHDL cho đồ hình trạng thái trên như sau:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY FSM IS
      PORT (CLK,RESET,UP: IN std_logic;
            Z: OUT std_logic);
END;
ARCHITECTURE BEHAVIOR OF FSM IS
      SIGNAL sreg: std_logic_vector (3 DOWNTO 0);
      SIGNAL next_sreg: std_logic_vector (3 DOWNTO 0);
      CONSTANT S0: std_logic_vector (3 DOWNTO
0):="0000";
      CONSTANT S1: std_logic_vector (3 DOWNTO
0):="0001";
      CONSTANT S2: std_logic_vector (3 DOWNTO
0):="0010";
      CONSTANT S3: std_logic_vector (3 DOWNTO
0):="0011";
      CONSTANT S4: std_logic_vector (3 DOWNTO
0):="0100";
      CONSTANT S5: std_logic_vector (3 DOWNTO
0):="0101";
      CONSTANT S6: std_logic_vector (3 DOWNTO
0):="0110";
      CONSTANT S7: std_logic_vector (3 DOWNTO
0):="0111";
      CONSTANT S8: std_logic_vector (3 DOWNTO
0):="1000";
      CONSTANT S9: std_logic_vector (3 DOWNTO
0):="1001";
      SIGNAL next_Z: std_logic;
```

```
BEGIN
      Sync: PROCESS (CLK) -- Cập nhật trạng thái mới
của bộ đếm
      BEGIN
            IF CLK='1' AND CLK'event THEN
             if RESET='1' then
             sreg<= S0;
             else
                  sreg <= next_sreg;
             end if;
            END IF;
      END PROCESS;

      Comb: PROCESS (sreg,UP) -- Kiểm tra điều kiện
chuyển trạng thái
      BEGIN
       CASE sreg IS
             WHEN S0 =>
                   IF ( UP='0' ) THEN
next_sreg<=S9;
                   ELSE  next_sreg<=S1;
                   END IF;
             WHEN S1 =>
                   IF ( UP='0' ) THEN
next_sreg<=S0;
                   ELSE  next_sreg<=S2;
                   END IF;
             WHEN S2 =>
                   IF ( UP='0' ) THEN
next_sreg<=S1;
                   ELSE  next_sreg<=S3;
                   END IF;
             WHEN S3 =>
```

```
                    IF ( UP='0' ) THEN
next_sreg<=S2;
                    ELSE  next_sreg<=S4;
                    END IF;
              WHEN S4 =>
                    IF ( UP='0' ) THEN
next_sreg<=S3;
                    ELSE  next_sreg<=S5;
                    END IF;
              WHEN S5 =>
                    IF ( UP='0' ) THEN
next_sreg<=S4;
                    ELSE  next_sreg<=S6;
                    END IF;
              WHEN S6 =>
                    IF ( UP='0' ) THEN
next_sreg<=S5;
                    ELSE  next_sreg<=S7;
                    END IF;
              WHEN S7 =>
                    IF ( UP='0' ) THEN
next_sreg<=S6;
                    ELSE  next_sreg<=S8;
                    END IF;
              WHEN S8 =>
                    IF ( UP='0' ) THEN
next_sreg<=S7;
                    ELSE  next_sreg<=S9;
                    END IF;
              WHEN S9 =>
                    IF ( UP='0' ) THEN
next_sreg<=S8;
                    ELSE  next_sreg<=S0;
```

```
                    END IF;
              WHEN OTHERS => next_sreg<=S0;
       END CASE;
      END PROCESS;
      Outputs: PROCESS (sreg,UP) --Tính kết quả đầu ra
      BEGIN
            IF UP='1' THEN
             if sreg=S9 then Z<= '1';
             else              Z<= '0';
             end if;
            ELSE
             if sreg=S0 then Z<= '1';
             else              Z<= '0';
             end if;
            END IF;
      END PROCESS;
END BEHAVIOR;
```

TÓM TẮT

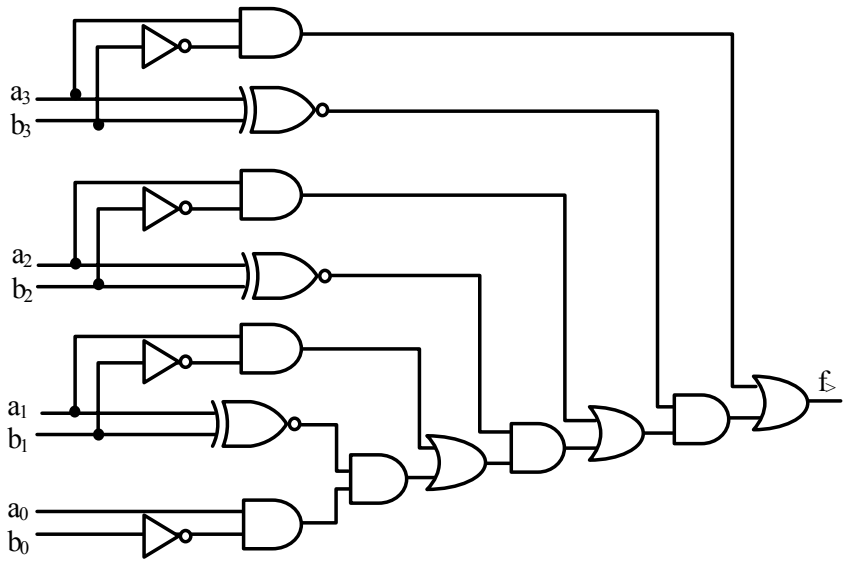
Chương này đã trình bày vai trò của ngôn ngữ mô tả phần cứng trong quá trình mô phỏng, thiết kế mạch số. Cấu trúc của ngôn ngữ VHDL được trình bày từ kiểu đối tượng, kiểu dữ liệu đến các cấu trúc lệnh song song và lệnh tuần tự… Trên cơ sở hiểu biến về cấu trúc của VHDL bạn đọc có thể thực hiện viết các mô tả cho cho hệ thống số từ đơn giản đến phức tạp theo các mô hình khác nhau: Mô hình hành vi, mô hình RTL, mô hình cấu trúc logic và mô tả theo đồ hình trạng thái.

CÂU HỎI ÔN TẬP

1. Viết mô tả VHDL cho hàm mạch tổ hợp có hàm logic như sau:

$$f_> = a_3.\overline{b_3} + \overline{a_3 \oplus b_3}.a_2.\overline{b_2} + \overline{a_3 \oplus b_3}.\overline{a_2 \oplus b_2}.a_1.\overline{b_1} + \overline{a_3 \oplus b_3}.\overline{a_2 \oplus b_2}.\overline{a_1 \oplus b_1}.a_0.\overline{b_0}$$

2. Viết mô tả VHDL cho mạch logic sau?



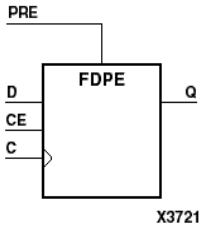
Hình BT.9.1

3. Viết mô tả VHDL cho các trigơ JK, RS, T có các đầu vào lập và xóa mức tích cực cao?
4. Viết mô tả VHDL cho bộ đếm thập phân tiến 2 digit?

5. Viết mô tả VHDL cho bộ đếm thập phân tiến lùi 2 digit?
6. Mô hình phần cứng nào tổng hợp được ứng với đoạn mô tả như sau:

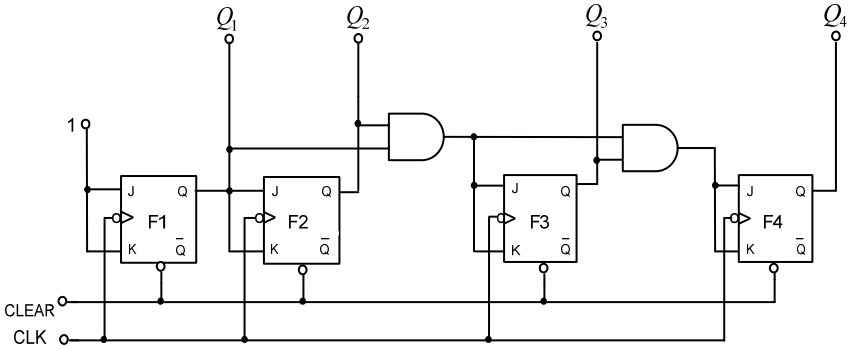
```
entity flop is
 port(C, D, CLR: in std_logic;
 Q : out std_logic);
end flop;
architecture archi of flop is
 begin
 process (C, CLR)
 begin
 if (CLR = '1')then
 Q <= '0';
 elsif (C'event and C='0')then
 Q <= D;
 end if;
 end process;
end archi;
```

7. Viết mô tả VHDL cho mạch giải mã BCD sang hiện thị LED7 đoạn?
8. Viết mô tả cho mạch điều khiển hiển thị cho cơ cấu chỉ thị số 4 digit có đầu vào là mã BCD 4 bit theo nguyên lý quét?
9. Viết mô tả VHDL cho mạch chọn địa chỉ 74138?
10. Viết mô tả VHDL cho mạch ALU đã học trong chương 4?
11. Viết mô tả VHDL cho mạch giải mã từ nhị phân sang thập phân, sang mã gray và ngược lại?
12. Viết mô tả VHDL cho IC ghi dịch 8 bit đã học trong chương 5?
13. Viết mô tả VHDL cho mô hình thanh ghi 4 bit hoạt động sườn dương của clock, có tín hiệu chốt clock và thiết lập không đồng bộ như sau:



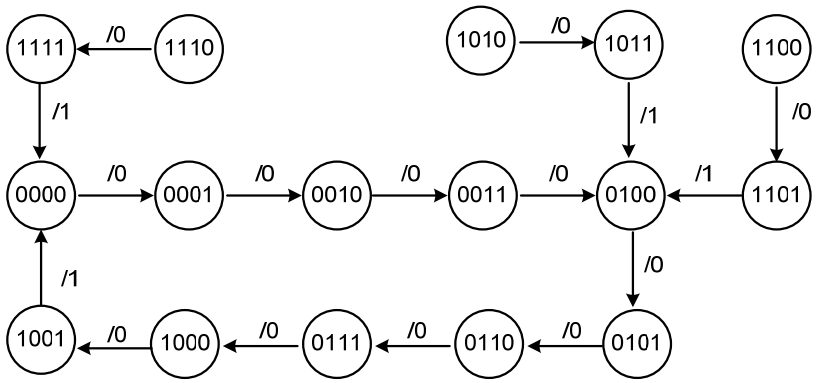
D[3:0]	Đầu vào dữ liệu 4 bit
C	Clock sườn dương
PRE	Tín hiệu thiết lập không đồng bộ mức tích cực cao

14. Viết mô tả VHDL cho mạch sau theo mô hình RTL



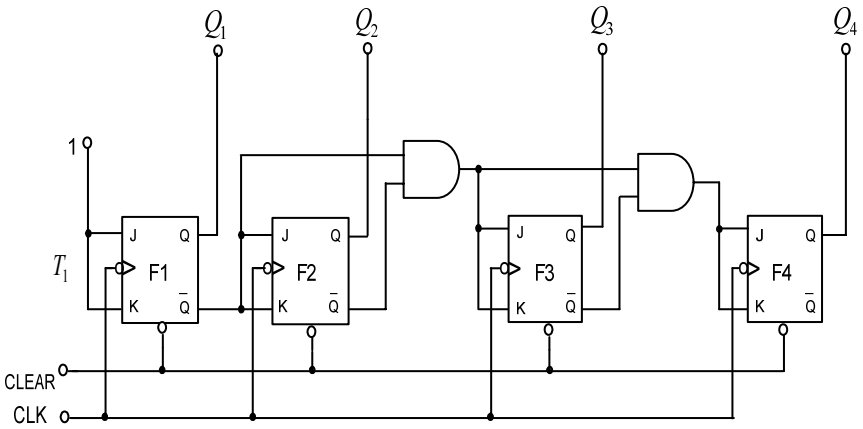
Hình BT.9.14

15. Viết mô tả VHDL cho đồ hình trạng thái sau:



Hình BT.9.15

16. Viết mô tả VHDL cho mạch sau theo mô hình RTL



Hình BT.9.16

17. Trình bày nguyên lý bắt phím, thiết kế mạch bắt phím và viết mô tả bằng VHDL?

18. Thiết kế mạch quét và nhận dạng cho ma trận phím bấm (Keypad) 4x4?

19. Thiết kế mô hình đèn giao thông tại một ngã tư và viết mô tả bằng VHDL?

20. Thiết kế mạch điều khiển ma trận LED 8x8 và viết mô tả bằng VHDL?

21. Thiết kế bộ điều khiển truyền thông nối tiếp UART?

TÀI LIỆU THAM KHẢO

1. *Giáo trình Kỹ thuật số* - Trần Văn Minh, NXB Bưu điện 2002.
2. *Cơ sở kỹ thuật điện tử số*, Đại học Thanh Hoa, Bắc Kinh, NXB Giáo dục 1996.
3. *Kỹ thuật số*, Nguyễn Thúy Vân, NXB Khoa học và Kỹ thuật 1994.
4. *Kỹ thuật điện tử số thực hành,* Bạch Gia Dương – Chử Đức Trình, Nhà xuất bản Đại học Quốc gia Hà Nội 2007.
5. *Giáo trình Kỹ thuật số,* Nguyễn Viết Nguyên, Nhà xuất bản Giáo dục 2004.
6. *Mạch logic kỹ thuật số,* Nguyễn Minh Đức, Nhà xuất bản Tổng hợp thành phố Hồ Chí Minh, 2004.
7. *Toán logic và kỹ thuật số,* Nguyễn Nam Quân – NXB Giáo dục 2004.
8. *Cấu trúc máy vi tính,* Trần Quang Vinh, NXB Đại học Quốc gia Hà nội, 2005.
9. *Fundamentals of logic design*, fourth edition, Charles H. Roth, Prentice Hall 1991.
10. *Lessons in Electric Circuits, Volume No 4-Digital,* Tony R. Kuphaldt, Tái bản lần thứ 4-2007.
11. *Digital engineering design*, Richard F.Tinder, Prentice Hall 1991.
12. *Digital design principles and practices*, John F.Wakerly, Prentice Hall 1990.

13. *VHDL for Programmable Logic* by Kevin Skahill, Addison Wesley, 1996
14. *The Designer's Guide to VHDL* by Peter Ashenden, Morgan Kaufmann, 1996.
15. *Analysis and Design of Digital Systems with VHDL* by Dewey A., PWS Publishing, 1993.

Giáo trình
Điện tử số

Chịu trách nhiệm xuất bản
NGUYỄN THỊ THU HÀ

Mã số: GD 20 HM 09

Biên tập:	NGÔ MỸ HẠNH
	THU CHÂU - THỌ VIỆT
Trình bày sách:	NGUYỄN BIÊN - NGUYỄN NAM
Sửa bản in:	THU CHÂU - THỌ VIỆT
Thiết kế bìa:	TRẦN HỒNG MINH

(Giáo trình này được ban hành kèm theo Quyết định số 47/QĐ-ĐT&KHCN ngày 20 tháng 02 năm 2009 của Giám đốc Học viện Công nghệ Bưu chính Viễn thông)

In 700 bản, khổ 16 x 24 cm tại Công ty Cổ phần In Khoa học Công nghệ mới
Số đăng ký kế hoạch xuất bản: 528-2009/CXB/2-202/TTTT
Số quyết định xuất bản: 226/QĐ-NXB TTTT ngày 18/11/2009
In xong và nộp lưu chiểu tháng 11 năm 2009.