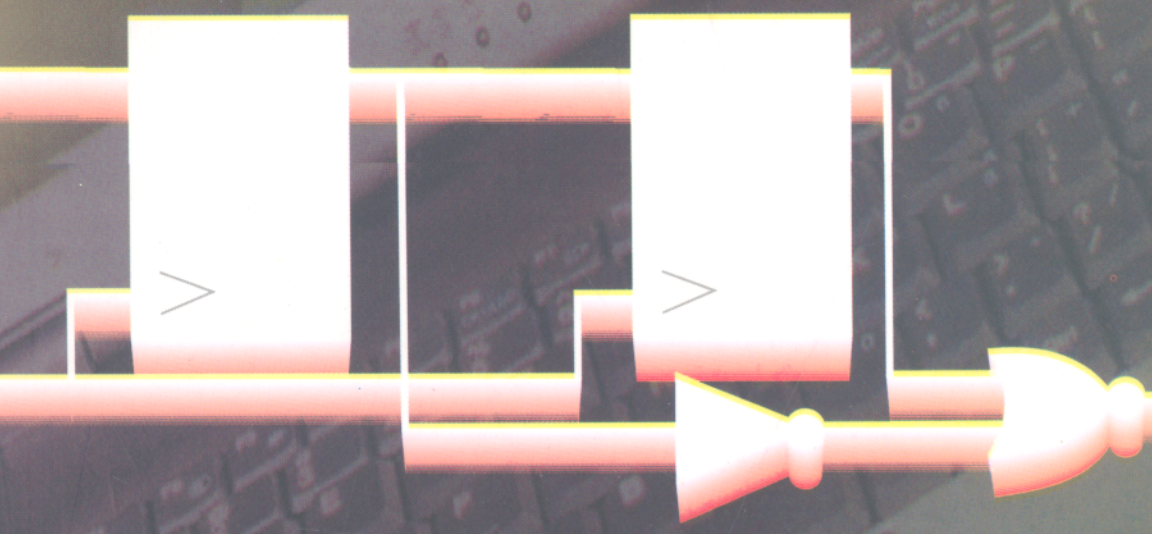
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
NGUYỄN LINH GIANG

# THIẾT KẾ MẠCH BẰNG MÁY TÍNH



NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

NGUYỄN LINH GIANG

# THIẾT KẾ MẠCH BẰNG MÁY TÍNH

• Giáo trình cho sinh viên Công nghệ Thông tin, Điện tử Viễn thông... các trường đại học, cao đẳng kỹ thuật... thuộc các hệ đào tạo

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
HÀ NỘI

6-6C2-01-6T7.3

KHKT-03

113-244-03

# Lời nói đầu

*Cùng với sự tiến bộ của khoa học kỹ thuật, việc tự động hoá thiết kế các mạch điện tử đóng một vai trò quan trọng trong việc thúc đẩy sự phát triển kỹ thuật tính toán. Từ những năm 50 của thế kỷ 20, quá trình phát triển kỹ thuật điện tử đã qua nhiều giai đoạn: từ những bóng đèn điện tử đến những bóng bán dẫn, qua những mạch tích hợp nhỏ rồi tới những mạch với mức độ tích hợp lớn và siêu lớn. Những mạch tích hợp loại này có thể chứa hàng triệu linh kiện bán dẫn trên một mạch. Ví dụ điển hình là các bộ vi xử lý. Việc thiết kế những mạch với độ tích hợp lớn và siêu lớn không thể thực hiện một cách thủ công, mà phải có sự giúp đỡ của máy tính.*

*Sau một số năm giảng dạy môn học "Thiết kế mạch nhờ máy tính" tại Khoa Công nghệ Thông tin, Trường Đại học Bách khoa Hà Nội, trên cơ sở tham khảo kinh nghiệm giảng dạy của các đồng nghiệp và các tài liệu thuộc lĩnh vực này, chúng tôi biên soạn giáo trình cho môn học này, nhằm cung cấp tài liệu tham khảo cho sinh viên chuyên ngành Công nghệ Thông tin, Điện tử Viễn thông và những ai quan tâm tìm hiểu về môn học trên.*

*Trong cuốn sách này chúng tôi trình bày các giai đoạn quan trọng trong quá trình thiết kế và sản xuất các mạch tích hợp trong công nghiệp. Quá trình này bao gồm các bước mô hình hóa mạch trên các mức độ chi tiết khác nhau, xây dựng mạch và kiểm nghiệm mạch. Chúng tôi không tham vọng trình bày tất cả các vấn đề liên quan tới tất cả các giai đoạn của quá trình sản xuất mà chủ yếu tập trung vào một số công đoạn: thiết kế mô hình hóa mạch và kiểm nghiệm tính đúng đắn của thiết kế. Trên giai đoạn mô hình hóa, trong thực tế có nhiều ngôn ngữ có thể được sử dụng như SPICE, VERILOG, VHDL..., song chúng tôi tập trung vào trình bày ngôn ngữ VHDL trong giáo trình này, bởi ngôn ngữ VHDL là một ngôn ngữ có tính cấu trúc cao. Một đặc điểm quan trọng của ngôn ngữ VHDL là nó cho phép mô tả thiết kế theo nhiều mức độ chi tiết khác nhau - từ mức kiến trúc đến các cấu trúc và dòng dữ liệu. Với những ưu điểm này, ngôn ngữ VHDL cho phép xây dựng các thiết kế mạch từ tổng quát đến chi tiết, cho phép nhà thiết kế có thể nhìn một cách tổng thể quá trình thiết kế và giúp cho quá trình kiểm tra tính đúng đắn của thiết kế được dễ dàng. Chính nhờ vậy mà ngôn ngữ VHDL*

*được sử dụng rộng rãi trong công nghiệp chế tạo mạch điện tử có độ tích hợp cao.*

*Nhân dịp cuốn sách được xuất bản, chúng tôi xin chân thành cảm ơn sự góp ý chân tình của các bạn đồng nghiệp trong Bộ môn Kỹ thuật Máy tính, Khoa Công nghệ Thông tin Trường Đại học Bách khoa Hà Nội. Đồng thời, chúng tôi cũng chân thành cảm ơn sự khuyến khích và tạo điều kiện của Nhà xuất bản Khoa học và Kỹ thuật để cuốn sách này sớm được ra đời. Cuối cùng chúng tôi mong nhận được sự đóng góp quý báu của các bạn đồng nghiệp và bạn đọc xa gần để lần tái bản sau cuốn sách được hoàn chỉnh hơn.*

**Tác giả**

**TS. Nguyễn Linh Giang**

## MỤC LỤC

# CHƯƠNG I. MỞ ĐẦU VÀO THIẾT KẾ MẠCH VI ĐIỆN TỬ

## §1.1. Các phân đoạn trong thiết kế các mạch tích hợp

Sự ra đời của các mạch vi điện tử đã làm cơ sở phát triển phần cứng và phần mềm của các hệ thống tính toán trong những thập kỷ gần đây. Việc tăng liên tục mức độ tích hợp của các mạch điện tử trên một nền đơn đã đưa tới việc chế tạo những hệ thống với độ phức tạp ngày càng tăng. Công nghệ chế tạo mạch tích hợp trên cơ sở các chất bán dẫn phát triển vũ bão. Tới giữa những năm 80 của thế kỷ 20 người ta đã có thể chế tạo được những mạch tích hợp chứa tới hàng triệu linh kiện điện tử trên một tinh thể chất bán dẫn. Những mạch đó được gọi là mạch tích hợp cao(VLSI) hoặc là mạch vi điện tử. Việc ra đời của những mạch vi điện tử đã làm nảy sinh sự cần thiết phải có một phương pháp luận và quy trình thiết kế, chế tạo thích hợp.

Trong công nghiệp, việc chế tạo các mạch tích hợp được thực hiện qua bốn giai đoạn:

- Giai đoạn thiết kế.
- Giai đoạn chế tạo.
- Giai đoạn kiểm tra.
- Giai đoạn đóng gói.

Ở giai đoạn thiết kế, từ các chức năng mà mạch sẽ thực hiện, chúng ta xây dựng mô hình của mạch trên nhiều mức độ chi tiết khác nhau. Các mức độ chi tiết có thể được chia thành mức kiến trúc, mức lôgic, mức vật lý. Kết quả của giai đoạn thiết kế là các mô hình của mạch đã được xác nhận không chứa lỗi trên phương diện thiết kế.

Giai đoạn thứ hai là giai đoạn chế tạo. Ở giai đoạn này, mạch tích hợp sẽ được chế tạo theo các công nghệ cấy ghép các phần tử mạch lên các tinh thể chất bán dẫn bằng phương pháp mặt nạ che phủ và công nghệ xây dựng các mạch nhiều lớp. Kết quả của giai đoạn này là những vi mạch thực hiện những chức năng như trong thiết kế.

Giai đoạn ba là giai đoạn kiểm tra. Ở giai đoạn này, những mạch đã chế tạo sẽ được kiểm nghiệm ngẫu nhiên để khẳng định rằng mạch không chứa lỗi về mặt chế tạo. Trong trường hợp có những lỗi gặp nhiều lần có thể rút ra

kết luận lỗi đó có thể là lỗi trong quá trình chế tạo. Dựa vào việc kiểm tra quy trình công nghệ ta có thể rút ra kết luận về các khâu có thể sinh ra lỗi.

Giai đoạn cuối cùng là giai đoạn đóng gói. Lúc đó các vi mạch sẽ được phân tách và được tạo vỏ bọc.

Trong chương trình ta sẽ nghiên cứu kỹ giai đoạn đầu tiên là giai đoạn thiết kế. Quá trình thiết kế các mạch vi điện tử trong công nghiệp được chia làm ba phân đoạn:

- Mô hình hóa.
- Tổng hợp và tối ưu hoá.
- Kiểm nghiệm và phê chuẩn.

Trong đó chúng ta tập trung vào bài toán mô hình hoá mạch và tổng hợp, tối ưu hoá mạch.

**Phân đoạn đầu tiên: Mô hình hoá**

Ở giai đoạn này, nhà thiết kế xây dựng các mô hình cấu trúc mạch và các chức năng mà mạch sẽ thực hiện. Các mô hình mạch là công cụ biểu diễn các ý tưởng thiết kế. Mô hình hoá đóng vai trò quan trọng trong thiết kế mạch vi điện tử bởi vì các mô hình là những phương tiện mang thông tin về các mạch sẽ được xây dựng một cách cô đọng và chính xác. Do đó mô hình cần phải chính xác, chặt chẽ cũng như có mức độ tổng quát, trong suốt và dễ hiểu đối với người thiết kế và máy. Với sự phát triển của các kỹ thuật mô phỏng, mô hình mạch có thể được xây dựng trên cơ sở các ngôn ngữ mô tả phần cứng HDL (hardware description languages). Trong nhiều trường hợp, các mô hình đồ họa như biểu đồ dòng thông tin, sơ đồ mạch và mô tả hình dạng hình học của các đối tượng cũng như cách sắp xếp chúng trên bản mạch đều có thể được dùng để biểu diễn mạch. Đối với những mạch có độ tích hợp siêu lớn do độ phức tạp của mạch rất cao nên việc xây dựng mô hình mạch thường theo các mức độ chi tiết khác nhau. Điều đó cho phép người thiết kế tập trung vào từng phần của mô hình tại từng giai đoạn thiết kế.

**Phân đoạn hai: Tổng hợp và tối ưu hóa**

Tổng hợp mạch là giai đoạn sáng tạo thứ hai của quá trình thiết kế. Giai đoạn đầu tuân theo các ý tưởng của nhà thiết kế hình thành dần các khái

niệm về mạch và xây dựng những mô hình sơ bộ đầu tiên về mạch. Mục đích chính của giai đoạn tổng hợp mạch là xây dựng mô hình chi tiết của mạch, ví dụ như các chi tiết về dạng hình học phục vụ cho công đoạn lắp ráp và tạo vỏ bọc cho mạch. Điều này đạt được thông qua quá trình xây dựng và chính xác hoá thiết kế từng bước trong đó mô hình trừu tượng ban đầu được người thiết kế chi tiết hoá từng bước lặp đi lặp lại. Khi thực hiện quá trình tổng hợp mạch theo các bước cải tiến mô hình, người thiết kế cần nhiều thông tin liên quan tới các công nghệ chế tạo và các phong cách thiết kế mong muốn. Ta có thể thấy các chức năng của mạch có thể độc lập với các chi tiết thực hiện, trong khi đó các dạng biểu diễn hình học của mạch hoàn toàn phụ thuộc vào các đặc tính của công nghệ ví dụ như kích thước của các dây dẫn trong mạch phụ thuộc vào công nghệ chế tạo.

Bài toán tối ưu mạch luôn kết hợp chặt chẽ với bài toán tổng hợp mạch. Quá trình tối ưu đòi hỏi phải lựa chọn những chi tiết xác định của mạch với mục đích làm tăng khả năng của mạch về phương diện thiết kế tương ứng với những độ đo xác định. Vai trò của tối ưu là nâng cao chất lượng của mạch điện như tối ưu về chức năng, về diện tích, về tính dễ kiểm nghiệm và phát hiện lỗi. Chức năng liên quan tới thời gian để thực hiện một quá trình xử lý thông tin cũng như số lượng thông tin có thể được xử lý trong một đơn vị thời gian. Các tính năng của mạch là ảnh hưởng lớn tới khả năng cạnh tranh của mạch trên thị trường. Vấn đề chất lượng của mạch cũng liên quan tới kích thước cũng như diện tích của mạch. Diện tích cũng là đối tượng của tối ưu mạch. Kích thước nhỏ của mạch cho phép có thể phân bố nhiều mạch trên một lớp, điều đó làm giảm giá thành chế tạo và đóng gói. Trong công nghiệp chế tạo chúng ta mong muốn có những thiết kế cho phép phát hiện lỗi và xác định vị trí lỗi của mạch sau khi chế tạo. Khả năng này, trong nhiều trường hợp, ảnh hưởng lớn tới chất lượng của mạch. Một thông số quan trọng trong vấn đề phát hiện lỗi của mạch là phần trăm lỗi có thể được phát hiện đối với một bộ giá trị thử nghiệm. Nói chung, người thiết kế mong muốn có những mạch dễ kiểm nghiệm, điều đó làm giảm giá thành chung của quá trình sản xuất.

**Phân đoạn 3: Kiểm nghiệm và phê chuẩn**

Quá trình phê chuẩn mạch là việc đạt được ở một mức độ chắc chắn hợp lý rằng mạch điện sẽ làm việc đúng với giả thiết không có lỗi chế tạo. Trên

phân đoạn này mục đích đặt ra là phải loại bỏ mọi lỗi thiết kế có thể có trước khi đưa vào sản xuất. Quá trình phê chuẩn mạch bao gồm việc xây dựng mô hình mô phỏng mạch dựa trên thiết kế và thực hiện kiểm tra. Mô phỏng mạch bao gồm phân tích các diễn biến hành vi của mạch điện theo thời gian đối với một hoặc nhiều bộ giá trị đầu vào. Quá trình mô phỏng có thể áp dụng trên nhiều mức thiết kế khác nhau tuỳ theo các mức trừu tượng của mô hình.

## §1.2. Mô hình hoá mạch điện

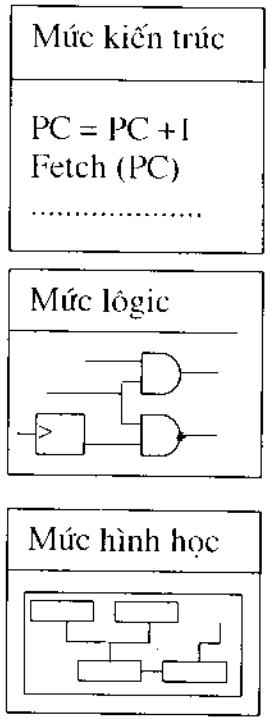
Mô hình mạch là biểu diễn trừu tượng trong đó chỉ ra những đặc tính thích hợp mà không có những chi tiết tương ứng. Quá trình tổng hợp mạch là quá trình tạo mô hình mạch bắt đầu từ những biểu diễn sơ lược nhất.

Các mô hình được phân loại theo các mức độ mô tả trừu tượng và các góc độ quan sát.

- Các mức độ mô tả trừu tượng được chia làm ba mức như sau:
  - Mức kiến trúc

    Ở mức kiến trúc, mạch điện được thể hiện qua tập hợp các thao tác như các tính toán trên dữ liệu, các phép chuyển đổi và truyền thông tin. Ví dụ, trên mức kiến trúc, mạch có thể được biểu diễn qua những mô hình trên các ngôn ngữ mô tả phần cứng, những biểu đồ luồng thông tin.
  - Mức lôgic

    Ở mức lôgic, mạch điện được thể hiện như tập hợp các chức năng lôgic và được chuyển thành các hàm lôgic. Ví dụ, trên mức lôgic mạch có thể được biểu diễn thông qua các biểu đồ chuyển trạng thái, các sơ đồ mạch lôgic.



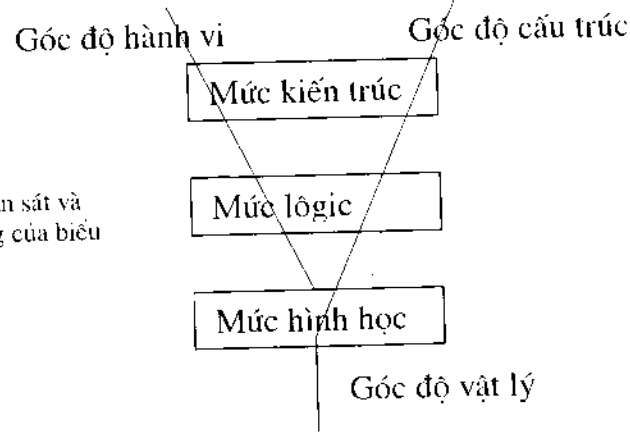

**Hình 1.1** Ba mức độ trừu tượng biểu diễn mạch điện.

- Mức hình học

Ở mức hình học, mạch có thể được biểu diễn như tập hợp các đối tượng hình học. Ví dụ đơn giản của biểu diễn hình học có thể là các lớp trong mạch nhiều lớp, dáng vẻ bề ngoài và phân bố của các phần tử cấu thành mạch.

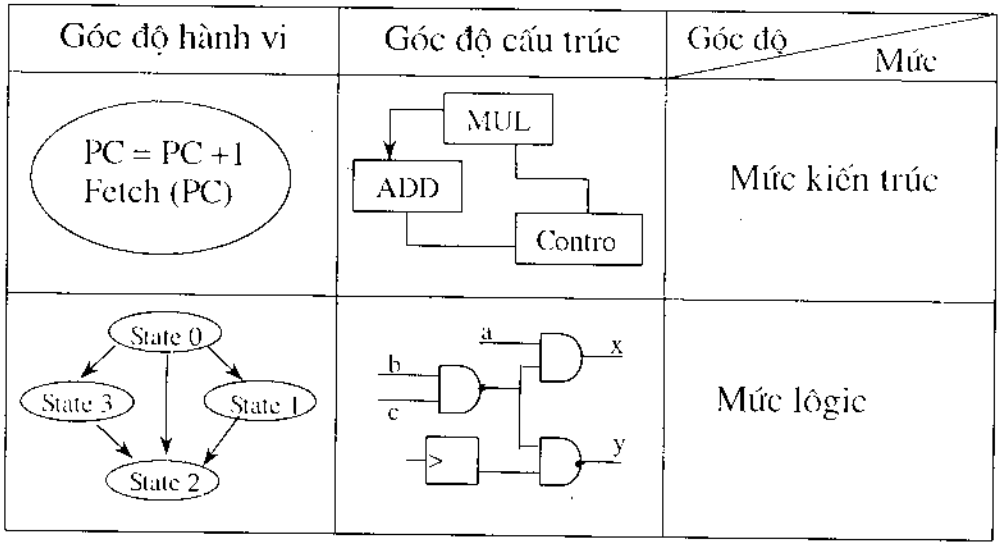
- Các góc độ quan sát cũng được chia thành ba góc độ:
  - Góc độ hành vi
    Góc độ hành vi mô tả các chức năng của mạch mà không quan tâm tới việc thực hiện các chức năng đó.
  - Góc độ cấu trúc
    Góc độ cấu trúc mô tả mô hình mạch bằng các thành phần cơ bản của mạch và các liên kết giữa các thành phần đó.
  - Góc độ vật lý
    Góc độ vật lý có liên quan tới các đối tượng vật lý xuất hiện trong thiết kế.

Các mô hình có các mức mô tả trừu tượng khác nhau và có thể được quan sát theo những góc độ khác nhau. Ví dụ, ở mức kiến trúc theo góc độ hành vi thì mạch điện là tập hợp các phép toán và sự liên quan giữa chúng với nhau, còn theo góc độ cấu trúc thì mạch là tập hợp các khối cơ sở và các liên kết, ghép nối giữa các khối cơ sở đó. Nếu xét trường hợp thiết kế các mạch đồng bộ thì với các mô hình trên mức lôgic, góc độ hành vi có thể là các lưu đồ chuyển trạng thái, còn góc độ cấu trúc là các phần tử lôgic và các kết nối giữa các phần tử đó. Mối quan hệ giữa các mức độ trừu tượng và các góc độ quan sát của mô hình được biểu diễn bằng sơ đồ chữ Y của Gajski-Kuhn.




**Hình 1.2** Các góc độ quan sát và các mức mô tả trừu tượng của biểu diễn mô hình mạch.

Hình 1.2 cho ta thấy mối liên hệ giữa các mức biểu diễn mô hình và các góc độ quan sát. Ở mức kiến trúc và lôgic, mô hình mạch thường được biểu diễn theo các góc độ hành vi và cấu trúc, còn ở mức hình học mô hình thường được biểu diễn qua góc độ cấu trúc và vật lý. Trên hình 1.3 cho ta ví dụ về các mức biểu diễn của mô hình bộ xử lý và các góc độ quan sát tương ứng. Ở mức kiến trúc, theo góc độ hành vi, mô hình là các dòng lệnh trên ngôn ngữ mô tả phần cứng HDL; theo góc độ cấu trúc, mô hình bao gồm tập hợp các khối cơ sở như bộ tính toán số học, bộ điều khiển, ... và các liên kết giữa các phần tử đó. Ở mức lôgic, theo góc độ hành vi, mô hình bao gồm các sơ đồ chuyển trạng thái của các ôtômat biểu diễn hoạt động của mạch; theo góc độ cấu trúc, mô hình được biểu diễn bằng các sơ đồ mạch lôgic giữa các phần tử lôgic cơ bản.




**Hình 1.3** Các mức biểu diễn mô hình và các góc độ quan sát tương ứng.

## §1.3. Tổng hợp và tối ưu hoá mạch dùng máy tính

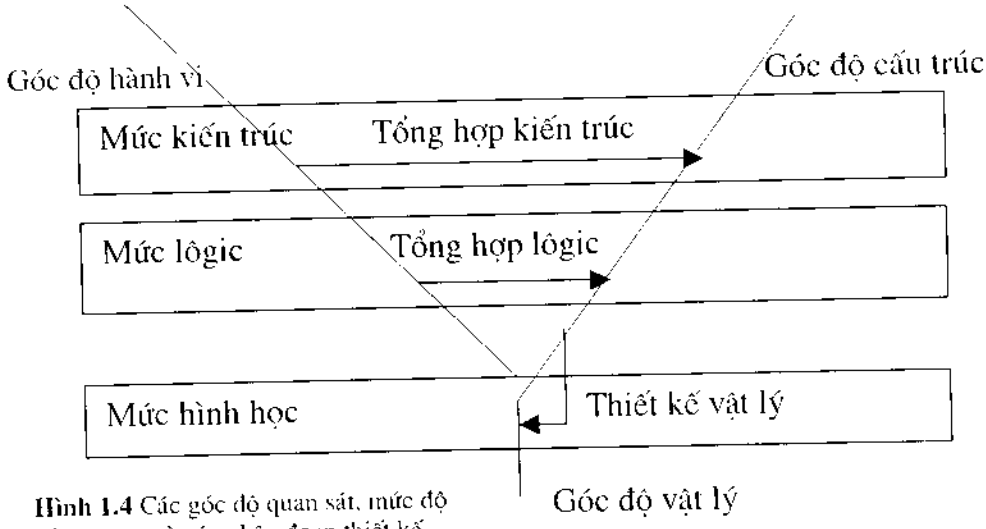
Các công cụ trợ giúp thiết kế bằng máy tính cho phép nâng cao năng suất thiết kế. Các kỹ thuật thiết kế cho phép giảm thời gian, nâng cao chu trình thiết kế và giảm công sức con người. Các kỹ thuật tối ưu làm tăng chất lượng thiết kế. Do đó các kỹ thuật tổng hợp và tối ưu hoá mạch với sự trợ

giúp của máy tính được sử dụng trong hầu hết các quá trình thiết kế mạch điện tử số.

## 1. Tổng hợp mạch điện

Việc phân loại các mô hình thành các mức trừu tượng và các góc độ quan sát cho chúng ta phương pháp xây dựng các thiết kế trên giai đoạn tổng hợp mạch. Giai đoạn tổng hợp mạch có thể được phân chia thành các phân đoạn sau:

- Tổng hợp ở mức kiến trúc bao gồm việc tạo ra góc độ cấu trúc của mô hình ở mức kiến trúc. Điều này tương đương với việc xác định và phân các chức năng của mạch thành các phép toán. Các phép toán này được gọi là tài nguyên của thiết kế. Trong mô hình cũng bao gồm cả các kết nối giữa các phép toán và trình tự thực hiện. Phân đoạn này thường được gọi là tổng hợp ở mức cao hay tổng hợp cấu trúc vì ở đó người thiết kế phải xác định các cấu trúc vĩ mô ( trên mức độ các sơ đồ khối ) của mạch.
- Tổng hợp ở mức lôgic là phân đoạn tạo ra góc độ cấu trúc của mô hình ở mức lôgic. Tổng hợp lôgic bao gồm các thao tác sử dụng kỹ thuật lôgic để tạo nên mô hình lôgic. Mô hình này gồm có các phần tử lôgic cơ bản và kết nối giữa các phần tử đó. Như vậy bước tổng hợp lôgic là bước xác định cấu trúc vi mô ( ở mức các phần tử lôgic cơ bản ) của mạch. Công việc chuyển đổi mô hình lôgic thành các kết nối giữa các phần tử được mô tả trong thư viện các phần tử cơ sở thường gọi là ánh xạ công nghệ hay là liên kết theo thư viện.
- Tổng hợp ở mức hình học bao gồm việc tạo ra góc độ vật lý của mô hình ở mức hình học. Nói cách khác, ở mức này mô hình được mô tả thông qua các đặc tính của tất cả các mẫu hình học tạo nên dạng của các mạch, phân bố các mạch trên bản mạch. Phân đoạn này thường được gọi là thiết kế vật lý.




**Hình 1.4** Các góc độ quan sát, mức độ trừu tượng và các phân đoạn thiết kế.

### a. Tổng hợp kiến trúc

Mô hình hành vi ở mức kiến trúc có thể được thể hiện qua tập hợp các phép toán và mối quan hệ phụ thuộc giữa chúng. Tổng hợp kiến trúc yêu cầu phải xác định các tài nguyên phần cứng cần được sử dụng để thực hiện các phép toán, ấn định trình tự thực hiện các phép toán và gắn kết chúng với các tài nguyên.

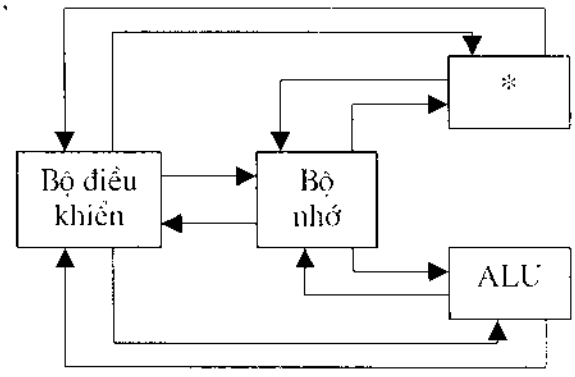
Việc thực hiện xây dựng mạch sau này phụ thuộc nhiều vào bước này. Thực vậy, các thiết kế về mặt kiến trúc sẽ xác định mức độ thực hiện song song của các phép toán. Thực hiện tối ưu hoá mạch trên mức này đóng vai trò hết sức quan trọng trong quá trình thiết kế.

Ta hãy xét ví dụ sau: ta thiết kế mạch thực hiện việc giải phương trình vi phân $y'' + 3xy' + 3y = 0$ trên đoạn $[0, a]$ bằng phương pháp số với bước dịch $dx$ và các giá trị ban đầu $x(0)=x$, $y(0)=y$, $y'(0)=u$.

Mạch điện có thể được biểu diễn bằng ngôn ngữ mô tả phần cứng như sau:

```
Diffeq{
        Read( x, y, dx, a );
        Repeat {
                x1 = x + dx;
                u1 = u - 3*x*u*dx - 3*y*dx;
```

```
        y1 = y + u*dx;
        c = x < a;
        x = x1;
        u = u1;
        y = y1;
    }
    Until (c );
    Write( y );
}
```




**Hình 1.5.** Góc độ cấu trúc ở mức kiến trúc.

Trong ví dụ này, để đơn giản, ta có thể coi các đường dữ liệu của mạch xuất phát từ hai nguồn: từ bộ nhân và từ bộ tính toán số học ALU. Bộ ALU này có thể thực hiện các phép toán cộng, trừ và so sánh. Mạch sẽ gồm có các thanh ghi, bộ xác định địa chỉ và bộ điều khiển. Góc độ cấu trúc của mô hình mạch ở mức kiến trúc cho ta thấy các cấu trúc vĩ mô của thiết kế.

### b. Tổng hợp lôgic

Mô hình ở mức lôgic của mạch có thể được biểu diễn bằng các sơ đồ chuyển trạng thái của các ôtômat hữu hạn, bằng các sơ đồ lôgic hoặc bằng các ngôn ngữ mô tả phần cứng HDL. Các mô hình này được nhà thiết kế đưa ra hoặc được tổng hợp từ các mô hình ở mức kiến trúc.

Các thao tác ở mức lôgic có thể khác nhau tuỳ theo tính chất của mạch (như mạch tổ hợp hoặc mạch tuần tự) và dạng biểu diễn ban đầu (biểu đồ chuyển trạng thái hay sơ đồ lôgic). Vấn đề tối ưu hoá đóng vai trò hết sức quan trọng. Nó gắn liền với quá trình tổng hợp trong việc xác định các cấu trúc vi mô của mạch. Kết quả cuối cùng của tổng hợp ở mức lôgic là biểu diễn cấu trúc đầy đủ, ví dụ như bằng các phần tử lôgic cơ bản và kết nối giữa những phần tử đó.

Trong ví dụ trước, các dòng dữ liệu được đưa về bộ nhớ và được đưa từ bộ nhớ tới ALU và bộ nhân. Hoạt động của bộ điều khiển được biểu diễn bằng sơ đồ chuyển trạng thái và sơ đồ lôgic.

### c. Thiết kế vật lý

Thiết kế vật lý là quá trình tạo ra sơ đồ bố trí của *chip* điện tử. Các lớp bố trí mạch tương ứng với các mặt nạ dùng trong quá trình chế tạo *chip*. Do đó các bố trí hình học là mục tiêu cuối cùng của thiết kế các mạch vi điện tử. Các thao tác chính trên phân đoạn này là bố trí mạch, đi dây, định tuyến. Trong cuốn sách này chúng ta không đi sâu vào phân đoạn này.

## 2. Tối ưu hoá mạch điện

Bài toán tối ưu hoá mạch luôn đi đôi với bài toán tổng hợp mạch. Tối ưu hoá mạch không những để đạt được ở mức độ cao nhất về chất lượng mạch mà còn tạo ra những mạch có tính cạnh tranh cao. Chúng ta chỉ xét các vấn đề tối ưu hai độ đo chất lượng quan trọng: diện tích và hoạt động của mạch. Ngoài ra một độ đo chất lượng quan trọng nữa là khả năng để kiểm tra và phát hiện lỗi của mạch.

Diện tích của mạch được xác định bằng tổng diện tích của các phần tử mạch. Do đó diện tích có thể xác định được thông qua góc độ cấu trúc của mạch nếu ta biết diện tích của từng thành phần mạch. Thông thường các phần tử cơ bản của mạch lôgic là các phần tử lôgic, các thanh ghi, các phần tử này có diện tích biết trước tuỳ thuộc vào từng loại thiết kế. Diện tích các dây nối đóng vai trò quan trọng và không thể bỏ qua. Các thành phần diện tích này có thể xác định từ mô hình mạch trên góc độ vật lý hoặc ước lượng từ các mô hình theo góc độ cấu trúc theo các phương pháp thống kê.

Hiệu năng của mạch được tối ưu hoá dựa trên thời gian trễ, thời gian đồng bộ, cạnh tranh trên các phần tử, ... Để tính toán độ đo hoạt động của mạch cần thiết phải phân tích cấu trúc và hành vi của mạch. Vấn đề này khác nhau đối với các loại mạch khác nhau.

Hiệu năng của các mạch tổ hợp được xác định thông qua thời gian trễ truyền từ đầu vào đến đầu ra. Thông thường để giảm độ phức tạp của tính toán, ta luôn giả thiết rằng các giá trị đầu vào xuất hiện trong cùng một thời điểm và hiệu năng của mạch được tính qua thời gian trễ truyền theo đường dữ liệu dài nhất.

Đối với các mạch tuần tự đồng bộ, độ đo hiệu năng có thể được xác định thông qua thời gian quay vòng của mạch. Thời gian này là chu kỳ đồng bộ nhanh nhất có thể đặt vào mạch. Ta nhận thấy rằng thời gian trễ truyền của

thành phần mạch tổ hợp của mạch tuần tự là cận dưới của thời gian quay vòng.

Khi ta xét mô hình trên mức kiến trúc như tập hợp các phép toán, đối với các mạch tuần tự đồng bộ, một trong những độ đo hiệu năng là thời gian cần thiết để thực hiện các phép toán. Thời gian này có thể được ước lượng thông qua các chu kỳ thời gian. Tích của thời gian quay vòng và thời gian thực hiện cho ta thời gian thực hiện tổng cộng của mạch. Thông thường thời gian quay vòng và thời gian thực hiện được tối ưu hoá riêng rẽ để đơn giản hoá quá trình tối ưu và thoả mãn các yêu cầu đặt ra đối với thiết kế.

Các mạch đồng bộ có thể được thực hiện dãy các phép toán theo chế độ dây chuyền ( pipeline ), trong đó mạch sẽ thực hiện các phép toán song song trên những tập hợp dữ liệu khác nhau. Như vậy hiệu năng của mạch còn có thể được thể hiện qua khả năng xử lý dữ liệu, lượng dữ liệu mà mạch có thể xử lý. Độ đo đó gọi là thông lượng của mạch. Đối với những mạch không thực hiện qua kỹ thuật dây chuyền, thông lượng bị giới hạn bởi nghịch đảo của tích giữa thời gian quay vòng và thời gian thực hiện. Kỹ thuật dây chuyền cho phép mạch tăng thông lượng dữ liệu được xử lý vượt qua giới hạn nói trên.

Với những độ đo nói trên, tối ưu hoá hiệu năng của mạch bao gồm việc giảm thiểu thời gian trễ truyền đối với mạch tổ hợp, thời gian quay vòng và thời gian thực hiện đối với mạch tuần tự đồng bộ; làm tăng tối đa thông lượng của mạch đối với những mạch thực hiện theo kỹ thuật dây chuyền.

Ngoài những bài toán tối ưu hoá về kích thước và thời gian nói trên, hiệu năng của mạch còn liên quan tới khả năng phát hiện lỗi và định vị vị trí lỗi trong mạch. Bài toán xây dựng những mạch cho phép dễ dàng tìm lỗi đóng một vai trò quan trong trong quá trình thiết kế và tối ưu hoá mạch. Những mạch dễ kiểm tra cho phép giảm thời gian sinh các bộ giá trị thử nghiệm và giảm số lượng các bộ giá trị thử nghiệm. Vấn đề đầu tiên được giải quyết bằng cách tìm ra những thuật toán tổng hợp mạch có hiệu quả; còn vấn đề thứ hai liên quan tới việc tìm ra những thuật toán tìm lỗi nhanh với mục tiêu giảm thời gian phát hiện lỗi và vị trí lỗi tương ứng với từng bộ giá trị thử nghiệm.

Tóm lại bài toán tối ưu hoá thiết kế được đưa về kết hợp hai bài toán: giảm thiểu diện tích thực tế của mạch và tăng hiệu năng của mạch với khả năng cao nhất có thể có. Bài toán tối ưu hoá có thể phụ thuộc vào các ràng buộc ví dụ như giới hạn trên về diện tích và giới hạn dưới về hiệu năng. Bài

toán tối ưu hoá có thể được biểu diễn trong không gian vectơ như sau. Tập hợp các cấu trúc có thể có của mạch sẽ được thiết kế tạo thành một không gian. Không gian này gọi là không gian thiết kế và chứa một số hữu hạn các điểm trong đó mỗi điểm tương ứng với một thiết kế cụ thể. Mỗi điểm ( tương ứng là thiết kế ) sẽ có các giá trị diện tích và hiệu năng tương ứng. Ta sẽ lập hàm giá trị trên cơ sở các đối tượng như diện tích, thời gian trễ, thời gian thực hiện, thời gian quay vòng, thông lượng. Bài toán tối ưu hoá mạch trở thành bài toán tìm kiếm điểm xác định trong không gian thiết kế sao cho các đối tượng đạt giá trị tối ưu.

Như vậy, trong chương này chúng ta đã nghiên cứu các bước cơ bản trong quá trình thiết kế tổng hợp mạch.

# CHƯƠNG II. CƠ SỞ TOÁN HỌC

Trong chương này chúng tôi nhắc lại một số vấn đề toán học làm cơ sở cho các chương tiếp theo. Các kiến thức toán chủ yếu xoay quanh cơ sở xây dựng các mạch số dựa trên các dạng chuẩn tắc của các biểu thức lôgic. Những vấn đề liên quan tới quá trình tìm lỗi trong các mạch lôgic liên quan tới các phương pháp mã hoá và lý thuyết đồ thị. Do đó chương này được chia làm hai phần chính, một phần liên quan tới vấn đề tối thiểu hoá các biểu thức lôgic, phần thứ hai liên quan tới lý thuyết đồ thị và mã hoá.

## § 2.1 Đại số Bool và lý thuyết chuyển mạch

### 1. Đại số Bool và lý thuyết tập hợp

Lý thuyết chuyển mạch là cơ sở thiết kế các hệ thống số hiện đại. Lý thuyết này dựa trên lôgic ký tự do nhà toán học Bool sáng tạo nên. Lĩnh vực lôgic ký tự là phát triển của lôgic học khi ta đưa vào các ký hiệu hình thức và các thao tác đại số hình thức. Đại số Bool được định nghĩa là một hệ đại số thoả mãn hệ các tiên đề.

**Định nghĩa:** Đối với tập hợp B = { a, b, ...} và hai toán tử '+' và '.', nếu bốn tiên đề sau thoả mãn thì hệ thống đại số gọi là đại số Bool:

1) $\forall a, b \in B$, $a + b = b + a$, $a \cdot b = b \cdot a$: Tính chất giao hoán; (2.1)

2) $\forall a, b, c \in B$,
$a + (b \cdot c) = (a + b) \cdot (a + c)$, $a \cdot (b + c) =$
$(a \cdot b) + (a \cdot c)$: Tính chất phân phối; (2.2)

3) $\exists 1 \in B, \exists 0 \in B$:
$\forall a \in B$, $a + 0 = a$, $a \cdot 1 = a$; Tồn tại các phần tử đơn vị; (2.3)

4) $\exists \bar{a} \in B$: $\forall a \in B$, $a + \bar{a} = 1$, $a \cdot \bar{a} = 0$: Phần bù. (2.4)

**Các định lý của đại số Bool:**

1. $a + a = a$; (2.5)
2. $a \cdot a = a$; (2.6)
3. $a + 1 = 1$; (2.7)

4. $a \cdot 0 = 0$; (2.8)

5. $\bar{\bar{a}} = a$ (2.9)

6. $(a \cdot b) \cdot c = a \cdot (b \cdot c)$; (2.10)

7. $(a + b) + c = a + (b + c)$; (2.11)

8. $a + a \cdot b = a$; (2.12)

9. $a \cdot (a + b) = a$; (2.13)

10. $\overline{a + b} = \bar{a}.\bar{b}$ ; (2.14)

11. $\overline{a.b} = \bar{a} + \bar{b}$ ; (2.15)

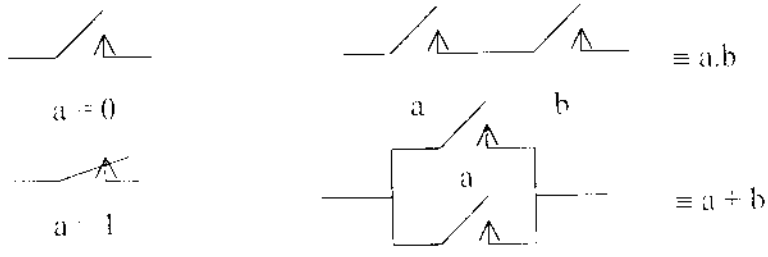
Với hệ tiên đề của đại số Bool, ta có thể chứng minh các định lý trên. Các định lý của đại số Bool có thể được thể hiện dưới dạng lý thuyết tập hợp như sau.

Giả thiết có tập hợp A, xét tập hợp S là tập các tập con của A. Đối với hai phần tử bất kỳ của tập S xác định phép hợp $\cup$ và phép giao $\cap$. Do S là tập của các tập con của A nên nếu tập con $a \in S$ thì phần bù của a trong A là $\bar{a}$ cũng thuộc S. Với các khái niệm tập hợp A và S, ta có thể minh họa các định lý của đại số Bool.

## 2. Đại số chuyển mạch

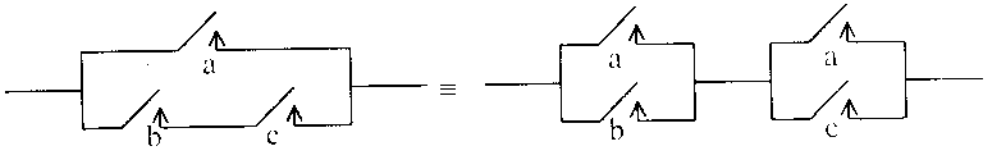
Một ví dụ của đại số Bool khi tập hợp B = {0, 1}. Khi đó ta có hệ đại số Bool đơn giản nhất. Mối liên hệ giữa đại số Bool nói trên với lý thuyết đóng ngắt mạch điện được Shannon đưa ra vào những năm 50 của thế kỷ 20. Hệ đại số bao gồm hai phần tử {0, 1} còn được gọi là đại số chuyển mạch. Các tiên đề và các định lý của đại số Bool hoàn toàn được áp dụng cho đại số chuyển mạch.

Để thiết lập mối tương quan giữa lý thuyết đóng ngắt mạch điện với các tiên đề và định lý của đại số chuyển mạch, ta đưa ra các sơ đồ mạch được xây dựng từ những sơ đồ cơ sở:




**Hình 2.1** Đại số chuyển mạch và các mạch khoá.

Trong các sơ đồ mạch đó, trạng thái đóng mạch của khoá tương ứng với giá trị 1; và trạng thái ngắt mạch tương ứng với giá trị 0. Theo cách quy định giá trị như trên, phép toán tuyển có thể được biểu diễn như đoạn mạch mắc song song các khoá, trong khi đó phép toán hội sẽ tương ứng với đoạn mạch mắc nối tiếp các khoá. Với các quy ước đó, tiên đề hai của đại số Bool có thể được biểu diễn thành các mạch tương đương như sau:

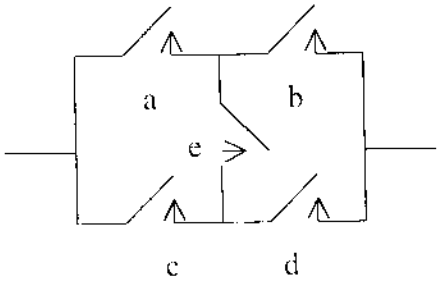



**Hình 2.2** Sơ đồ mạch đóng ngắt minh hoạ cho tính chất phân phối.

Sự tương đương giữa hai sơ đồ mạch khoá này có thể được kiểm chứng bằng các bảng chân lý tương ứng.

Dựa vào sự tương thích giữa các biểu thức lôgic và các sơ đồ mạch khoá ta có thể tạo ra các mạch khoá tương đương các biểu thức lôgic với những độ phức tạp khác nhau dựa vào các phép biến đổi trong mạch điện. Mặc dù vậy có một số loại mạch không thể biểu diễn được thành kết hợp giữa các tổ hợp mạch song song và nối tiếp, trong những trường hợp đó ta phải xây dựng mạch dựa theo bảng chân lý hoặc sử dụng phương pháp tạo tập hợp các liên kết.

Chúng ta xét trường hợp mạch theo liên kết mạch cầu: các đường đi từ phần phía trái mạch sang phần phía phải mạch bao gồm các đường {ab, aed, ceb, cd}.Phương pháp tạo tập hợp các liên kết thể hiện như sau: nếu trên bất kỳ đường đi từ phần bên trái mạch cầu sang phần bên phải ta đóng tất cả các khoá, khi đó mạch điện sẽ ở trạng thái đóng, còn trong trường hợp trên mỗi đường có ít nhất một khoá mở thì mạch sẽ ở trạng thái mở. Ví dụ nếu trên đường *aed* ta đóng tất cả các khoá *a, e, d* mạch sẽ ở trạng thái đóng. Như vậy đối với mạch trên, biểu thức lôgic tương đương sẽ là:




**Hình 2.3** Mạch cầu.

a . b + a . e . d + c . e . b + c . d

Khi sử dụng lý thuyết chuyển mạch trong biểu diễn các biểu thức lôgic, các định lý của đại số chuyển mạch có thể rút ra từ các tiên đề, nguyên lý đối ngẫu của lý thuyết mạch có vai trò khá quan trọng.

**Nguyên lý đối ngẫu**: nguyên lý đối ngẫu dựa trên cơ sở xây dựng các biểu thức đối ngẫu.

- Đối ngẫu của một biểu thức đại số Bool là một biểu thức lôgic trong đó các biến $a$ của biểu thức ban đầu được thay bằng $\bar{a}$, '*1*' thay bằng '*0*', '*0*' thay bằng '*1*', phép tuyển được thay bằng phép hội, phép hội được thay bằng phép tuyển. Khi thành lập biểu thức đối ngẫu ta phải sử dụng các dấu ngoặc để đảm bảo trật tự tính toán biểu thức.
- Nguyên lý đối ngẫu: giá trị của một biểu thức đại số Bool bằng bù của giá trị bbiểu thức đối ngẫu tương ứng.

Ta có thể chứng minh nguyên lý đối ngẫu bằng phương pháp quy nạp toán học và sử dụng các định lý de Morgan.

## § 2.2. Các hàm lôgic và dạng chuẩn tắc

### 1. Các hàm lôgic và khối n-chiều

Hàm lôgic $n$ biến được định nghĩa là ánh xạ không gian $n$-chiều $\mathbf{B}^n$ vào $\mathbf{B}$ :

$$\mathbf{B}^n \rightarrow \mathbf{B} \quad \mathbf{B}=\{0,1\} \tag{2.16}$$

Nếu $x_i$ là phần tử thuộc $\mathbf{B}$, khi đó $\boldsymbol{x}=(x_1,...,x_n)$ là một vectơ của không gian vectơ $n$ chiều $\mathbf{B}^n$. Hàm lôgic trên không gian $\mathbf{B}^n$ có thể được viết dưới dạng $f(x_1,...,x_n)$. Không gian vectơ $\mathbf{B}^n$ chứa $2^n$ điểm và một vectơ $\boldsymbol{x}$ thuộc không gian $\mathbf{B}^n$ được biểu diễn là một trong số $2^n$ điểm của không gian $n$ chiều $\mathbf{B}^n$. Các điểm này có thể được đặt tương ứng với các đỉnh của một hình khối có $2^n$ đỉnh trong không gian $\mathbf{B}^n$ . Hàm lôgic $f(x_1,...,x_n)$ sẽ đặt tương ứng mỗi đỉnh của hình khối với các giá trị *0* hoặc *1*.

Trong lĩnh vực thiết kế các mạch số không nhất thiết phải đặt giá trị *0* hoặc *1* cho tất cả các đỉnh của khối - các vectơ *x*. Một cách chính xác hơn, hàm lôgic *f* được định nghĩa là ánh xạ của tập con **A** của không gian $\mathbf{B}^n$ vào **B**.

$$f: \mathbf{A} \subseteq \mathbf{B}^n \rightarrow \mathbf{B} \quad \mathbf{B}=\{\mathbf{0},\mathbf{1}\} \tag{2.17}$$

Khi ta xét một tập con $\mathbf{A} \subseteq \mathbf{B}^n$ và hàm *f* có miền xác định là **A**.những điểm thuộc $\mathbf{B}^n$ và không thuộc **A** ( $x \in \mathbf{B}^n \setminus \mathbf{A}$ ) là đỉnh ta không quan tâm tới và ký hiệu là *đỉnh-d*; các đỉnh thuộc **A** và tương ứng với các giá trị *0* hoặc *1* sẽ được gọi lần lượt là *đỉnh-0* và *đỉnh-1*. Với cách ký hiệu các đỉnh của hình khối $2^n$ đỉnh như vậy, một hàm lôgic *f* sẽ ánh xạ các điểm trong không gian $\mathbf{B}^n$ vào tập hợp *{0,1,d}* :

$$f(x) : \mathbf{B}^n \rightarrow \{0,1,d\} \tag{2.18}$$

Nếu một đỉnh của hình khối *n*-chiều biểu diễn dưới dạng biểu thức lôgic, ta nhận được một biểu thức hội. Ví dụ trong không gian *3*-chiều, đỉnh *011* sẽ tương ứng với biểu thức lôgic $x_1 x_2 x_3$. Biểu thức lôgic có thể nhận được từ các đỉnh của hình khối $2^n$ chiều bằng cách ghi $x_i$ nếu giá trị của toạ độ thứ *i* tương ứng là '*1*' hoặc $\bar{x}_i$ nếu giá trị tọa độ thứ *i* là '*0*'. Biểu thức hội chỉ nhận giá trị '*1*' tại một điểm *x* duy nhất. Như vậy toạ độ của đỉnh sẽ tương ứng với một phần tử trong biểu thức hội.

Một cách tổng quát ta đưa vào khái niệm khối cấp *m* như sau:

Một vectơ *k* chiều : $c = (c_1,...,c_k)$, $c_i \in \{0,1,x\}$, $k<n$ được gọi là *khối* và một *khối* chứa *m* ký tự *x* sẽ gọi là *khối cấp m*.

Ví dụ: (011) là khối cấp 0; (01x)- khối cấp 1;

Khái niệm *khối* là tổng quát hóa của khái niệm đỉnh của khối *n*-chiều. *Khối cấp m* là khối con *m* chiều có chứa $2^m$ đỉnh. Ví dụ khối *(01x)* chứa hai đỉnh *(010)* và *(011)*. Nói một cách khác ký tự *x* có thể nhận giá trị '*0*' hoặc '*1*'. Một khối cấp *m* tương ứng với biểu thức hội chứa *n-m* biến. Biểu thức hội này có thể nhận được nếu ghi $\bar{x}_i$ khi giá trị tương ứng của tọa độ thứ *i* là '*0*' và $x_i$ khi giá trị tọa độ đó là '*1*' trong đó vị trí của ký tự *x* không được tính đến. Nói cách khác ký tự *x* có thể nhận giá trị '*0*','*1*' và khối cấp *m* sẽ tương đương với biểu thức lôgic gồm *(n-m)* biến.

Một khối cấp *0* sẽ tương ứng với đỉnh duy nhất của khối *n*-chiều, khối cấp *m* sẽ biểu diễn $2^m$ đỉnh, như vậy một khối cấp *m* sẽ bao phủ $2^m$ khối bậc

*0*. Một cách tổng quát, với hai tập hợp các khối $\mathbf{C}_1$ và $\mathbf{C}_2$, nếu tập hợp đỉnh bao phủ mỗi khối thuộc nhóm $\mathbf{C}_1$ là tập con của tập hợp đỉnh bao phủ các khối của $\mathbf{C}_2$, khi đó ta nói rằng $\mathbf{C}_2$ bao phủ $\mathbf{C}_1$.

## 2. Các dạng chuẩn tắc của hàm lôgic

Nếu hàm lôgic *f* không chứa *đỉnh-d*, khi đó để xác định hàm ta cần đưa ra tập hợp các *đỉnh-1* và *đỉnh-0*. Do không chứa các *đỉnh-d* nên tập hợp các *đỉnh-1*($V_1(f)$) và tập hợp các *đỉnh-0* ($V_0(f)$) bù nhau. Một đỉnh *c* (khối cấp *0*) tương ứng với một biểu thức hội *P(c)*, khi đó hàm *f(x)* sẽ được biểu diễn thông qua tập hợp các *đỉnh-1* $V_1(f)$ như sau:

$$f(x) = \bigvee_{c \in V_1(f)} P(c) \qquad (2.19)$$

*P(c)* gọi là biểu thức hội cực tiểu hay ngắn gọn là tích cực tiểu.

Như trên đã đề cập tới, một khối cấp *m* đại diện cho $2^m$ đỉnh, biểu thức hội cực tiểu sẽ tương ứng với số lượng cực tiểu các đỉnh tham gia vào biểu thức hội và sẽ nhận giá trị *1* tại những đỉnh này. Biểu thức (2.19) là biểu thức tuyển của các biểu thức hội cực tiểu và được gọi là dạng chuẩn tắc tuyển của biểu thức lôgic.

Ta xác định hàm bù lôgic ( gọi tắt là hàm bù )của một hàm *f* là một hàm nhận giá trị '*0*' tại những đỉnh mà *f* nhận giá trị '*1*' và nhận giá trị '*1*' tại những đỉnh mà *f* nhận giá trị '*0*'; hàm này được ký hiệu bằng $\bar{f}$

$$\bar{f}(x) = 1 \oplus f(x) \qquad (2.20)$$

Ta có thể nhận được hàm $\bar{f}$ nếu thay $V_1(P(c))$ bằng $V_0(P(c))$ trong biểu thức (2.19).

$$\bar{f}(x) = \bigvee_{c \in V_0(f)} P(c) \qquad (2.21)$$

Từ đó ta có thể nhận được biểu thức của *f(x)* bằng cách áp dụng công thức: $\bar{\bar{f}} \equiv f$.

$$f(x) = \bigwedge_{c \in V_0(f)} S(c) \qquad (2.22)$$

Các biểu thức *S(c)* nhận được từ *P(c)* bằng thay $x_i$ thành $\bar{x}_i$, $\bar{x}_j$ thành $x_j$ và thay phép hội ($\wedge$) thành phép tuyển ($\vee$). Ta nhận thấy *S(c)* sẽ tương ứng

với $(2^n - 1)$ đỉnh ngoại trừ đỉnh tương ứng với $P(c)$ và được gọi là biểu thức tuyển cực đại. Cách biểu diễn hàm lôgic $f$ thông qua phép hội của các biểu thức tuyển cực đại gọi là dạng chuẩn tắc hội.

Tiếp theo để có thể khảo sát một dạng chuẩn tắc nữa, chúng ta định nghĩa hàm loại trừ lôgic XOR như sau: XOR là phép toán hai ngôi cho giá trị '*1*' nếu chỉ một trong hai toán hạng nhận giá trị '*1*' và nhận giá trị '*0*' trong những trường hợp còn lại.

$$x \oplus y \equiv \bar{x}\, y \wedge x\, \bar{y} \tag{2.23}$$

Trong biểu thức trên ta thấy nếu x hoặc y luôn nhận giá trị 1 thì biểu thức sẽ nhận giá trị tương ứng theo $\bar{y}$ hoặc theo $\bar{x}$, có nghĩa là $1 \oplus x = \bar{x}$. Thêm vào đó phép toán XOR $\oplus$, cũng giống như phép cộng, thỏa mãn tính chất giao hoán, kết hợp và phân phối với phép nhân. Dựa vào các tính chất trên ta có thể đưa ra dạng chuẩn tắc theo phép toán XOR như sau.

Một hàm lôgic bất kỳ có thể được biểu diễn theo hệ thức sau:

$f(x_1, x_2, \ldots, x_n) = f(x_1, x_2, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)\bar{x}_i + f(x_1, x_2, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)x_i$

Cứ tiếp tục như vậy ta sẽ nhận được:

$$f(x_1, x_2, \ldots, x_n) = f(0, \ldots, 0)\bar{x}_1\bar{x}_2\ldots\bar{x}_n \vee f(1, 0, \ldots, 0)x_1\bar{x}_2\ldots\bar{x}_n \vee \ldots \vee f(1, 1, \ldots, 1)x_1x_2\ldots x_n. \tag{2.24}$$

Mỗi số hạng trong biểu thức trên là một biểu thức hội cực tiểu do đó chỉ có thể có một đỉnh nhận giá trị '*1*' đối với từng bộ giá trị của $x_1, x_2, \ldots x_n$. Thay phép toán $\vee$ bằng phép toán $\oplus$ và với mỗi $x_i$ thay bằng biểu thức $(1 \oplus x_i)$ ta nhận được biểu thức lôgic tương đương chỉ chứa phép hội và phép loại trừ lôgic.

$$f(x_1, x_2, \ldots, x_n) = f(0, \ldots, 0)(1 \oplus x_1)(1 \oplus x_2)\ldots(1 \oplus x_n) \oplus f(1, 0, \ldots, 0)x_1(1 \oplus x_2)\ldots(1 \oplus x_n) \oplus \ldots \oplus f(1, 1, \ldots, 1)x_1x_2\ldots x_n. \tag{2.25}$$

Mở các dấu ngoặc và áp dụng các tính chất kết hợp, phân phối với phép hội, ta nhận được:

$$\begin{aligned} f(x_1, x_2, \ldots, x_n) = {} & a_0 \oplus \\ & a_1x_1 \oplus a_2x_2 \oplus \ldots \oplus a_nx_n \oplus \\ & a_{12}x_1x_2 \oplus a_{13}x_1x_3 \oplus \ldots \oplus a_{1n}x_1x_n \oplus \\ & a_{23}x_2x_3 \oplus \ldots \oplus a_{(n-1)n}x_{n-1}x_n \oplus \\ & \ldots\ldots \\ & a_{123\ldots n}x_1x_2x_3\ldots x_n \end{aligned} \tag{2.26}$$

Các hệ số $a_0, a_1, ..., a_{12...n}$ có thể tìm được bằng cách tương ứng các biểu thức (2.25) và (2.26). Để biểu diễn các hệ số $a_0, a_1, ..., a_{12...n}$ ta đưa vào khái niệm vi phân của hàm lôgic.

Ta định nghĩa vi phân lôgic $\frac{\partial}{\partial x_i} f(x_1, x_2, ..., x_n)$ của hàm $f(x_1, x_2, ..., x_n)$ là biểu thức:

$$\frac{\partial}{\partial x_i} f(x_1, x_2, ..., x_n) = f(x_1, x_2, ..., x_{i-1}, 1, x_{i+1}, ..., x_n) \oplus f(x_1, x_2, ..., x_{i-1}, 0, x_{i+1}, ..., x_n) \qquad (2.27)$$

Do phép toán $\oplus$ biểu diễn đồng thời phép lấy tổng và phép lấy hiệu nên biểu thức trên còn gọi là sai phân lôgic. Vi phân lôgic có các tính chất của toán tử tuyến tính. Thêm vào đó kết quả của phép toán lấy vi phân theo biến $x_i$ của một tích lôgic sẽ bằng '*0*' nếu biến $x_i$ không tham gia vào tích và sẽ bằng chính tích lôgic loại trừ đi $x_i$ nếu $x_i$ tham gia vào biểu thức.

Ví dụ:
$$\frac{\partial}{\partial x_3} x_1x_2x_3x_4 = x_1x_2 1x_4 \oplus x_1x_2 0x_4 = x_1x_2x_4$$

$$\frac{\partial}{\partial x_5} x_1x_2x_3x_4 = x_1x_2x_3x_4 \oplus x_1x_2x_3x_4 = 0$$

Sử dụng khái niệm vi phân lôgic, các hệ số $a_{\alpha_1\alpha_2...\alpha_m}$ trong đó $\alpha_i \in [1,...,n]$ của biểu thức (2.26) sẽ được viết dưới dạng

$$a_{\alpha_1\alpha_2...\alpha_m} = \left[ \frac{\partial}{\partial x_{\alpha_1}} \frac{\partial}{\partial x_{\alpha_2}} \cdots \frac{\partial}{\partial x_{\alpha_m}} f(x_1, \cdots, x_n) \right]$$

$$i=1..m \ , \qquad \alpha_i \in [1,...,n] \qquad (2.28)$$

Từ biểu thức này suy ra hệ số $a_{\alpha_1\alpha_2...\alpha_m}$ bằng tổng loại trừ lôgic của các giá trị tại các đỉnh bao phủ một khối $m$ chiều. Khối $m$-chiều này chứa $x$ tại các vị trí tương ứng với $\alpha_i$ và '*0*' tại tất cả các vị trí còn lại.

## § 2.3. Tối thiểu hoá các biểu thức lôgic

Nội dung của mục này bàn tới các phương pháp tối thiểu hoá các biểu thức lôgic cơ bản khi thiết kế các mạch số. Việc tối thiểu hoá các biểu thức lôgic làm các biểu thức đó trở nên đơn giản hơn. Điều đó làm giảm kích thước và tăng hiệu năng mạch ( trên phương diện thời gian trễ và thời gian thực hiện ) được tổng hợp. Ngoài ra đối với việc phát hiện lỗi và thử nghiệm thiết kế, những mạch càng đơn giản cho phép giảm thời gian tìm các bộ giá trị thử nghiệm và giảm thời gian phát hiện lỗi. Do đó bài toán tối thiểu hoá các biểu thức lôgic đóng vai trò quan trọng trong quá trình tổng hợp và tối ưu mạch.

### 1. Nguyên lý chung để tối thiểu hóa các biểu thức lôgic

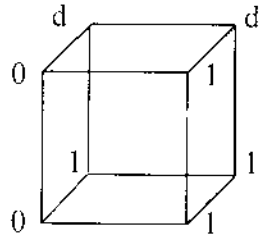
Cho hàm lôgic $n$ biến $f(x_1, x_2, ..., x_n)$ biểu diễn dưới dạng chuẩn tắc tuyển (tổng các tích lôgic). Tối thiểu hoá số lượng các tích lôgic của hàm $f$ và số lượng các biến lôgic trong từng tích lôgic dẫn tới việc làm giảm giá thành thể chế tạo mạch: số lượng các phần tử cơ sở, số lượng các đầu vào của mạch và các đầu vào của các phần tử mạch, diện tích của mạch, giảm thời gian kiểm nghiệm mạch.

Hàm $f$ được biểu diễn bởi tập hợp các *đỉnh-1* $\mathbf{V}_1(f)$ và tập hợp các *đỉnh-d* $V_d(f)$. Giữa các tích cực tiểu của hàm $f$ và các khối có mối tương quan một - một: khối cấp $m$ sẽ tương ứng với một tích lôgic (biểu thức hội) gồm $(n - m)$ biến.

Định nghĩa ***tích tối giản*** : khối $c$ được gọi là tích tối giản của hàm $f$ nếu:

- Tập hợp V(c) của các đỉnh ( khối bậc 0 ).
- $V(c) \cap V_1(f) \neq \varnothing$;
- $V(c) \subset V_1(f) \cup V_d(f)$;
- $\exists$ khối c': $V(c') \subset V_1(f) \cup V_d(f)$ và $V(c) \subseteq V(c')$.

Ví dụ: Giả sử ta có hàm $f(x_1,x_2,x_3)$ có tập hợp đỉnh $\mathbf{V}_0(f)$, $\mathbf{V}_1(f)$ và $\mathbf{V}_d(f)$.




$V_0(f) = \{(0,0,0),(1,0,0)\}$

$V_1(f) = \{(0,1,0),(0,1,1),(0,0,1),(1,1,0)\}$

$V_2(f) = \{(1,0,1),(1,1,1)\}$

**Hình 2.5** Các tích tối giản $(x1x)$ và $(xx1)$.

Các đỉnh được biểu diễn bằng khối (x10) chứa trong $\mathbf{V}_1(f) \cup \mathbf{V}_d(f)$. Mặt khác các đỉnh được biểu diễn bằng khối (x1x) cũng chứa trong $\mathbf{V}_1(f) \cup \mathbf{V}_d(f)$ và khối (x1x) bao phủ bởi khối (x10). Do đó (x10) không phải là tích tối giản. Khối duy nhất bao phủ (x1x) là (xxx) nhưng tập hợp biểu diễn khối (xxx) không nằm trong $\mathbf{V}_1(f) \cup \mathbf{V}_d(f)$, do đó (x1x) là tích đơn giản của hàm $f$, ta cũng có (xx1) cũng là tích tối giản.

Ta ký hiệu P là tập hợp tất cả các tích tối giản của hàm lôgic $f$, khi đó ta sẽ có định lý về bao phủ cực tiểu sau:

**Định lý về bao phủ cực tiểu:**

Nếu tập hợp $V(\mathbf{C})$ của các đỉnh biểu diễn tập hợp các khối **C** được cho dưới dạng:

$$V(C) = \bigcup_{a \in C} V(a)$$

khi đó tập hợp đầy đủ các tích tối giản P sẽ nhận trọng số nhỏ nhất trong tất cả các khả năng có thể có của tập C thoả mãn hệ thức:

$$\mathbf{V}_1(f) \subset \mathbf{V}(\mathbf{C}) \subset \mathbf{V}_1(f) \cup \mathbf{V}_d(f)$$

Hàm trọng số được giả thiết là dương và đơn điệu tăng với các biến độc lập là số lượng các biến tham gia vào các tích lôgic của từng khối và số lượng tất cả các khối.

Để chứng minh định lý này chúng ta dùng phương pháp phản chứng. Giả sử tồn tại tập hợp các khối **C'** có trọng số cực tiểu và không thoả mãn yêu cầu định lý, khi đó trong các khối đó tồn tại ít nhất một khối không phải là tích tối giản. Khối này được ký hiệu là $c'$, khi đó ta xét tập hợp:

$$\mathbf{C''} = (\ \mathbf{C'} - \{c'\}\ ) \cup \{c''\}.$$

trong đó $c''$ là tích tối giản bao phủ $c'$. Ta có số lượng các khối trong tập hợp **C'** và **C''** bằng nhau. Vì $c''$ là tích tối giản ($c'' \neq c'$) và bao phủ $c'$ nên trọng số **C''** nhỏ hơn **C'**. Điều đó là mâu thuẫn vì ta giả thiết **C'** có trọng số nhỏ nhất.

Quá trình tối thiểu hóa các hàm lôgic dựa trên cơ sở định lý về bao phủ tối thiểu có thể được chia làm hai giai đoạn sau:

a) Xác định tất cả các tích tối giản của hàm lôgic $f$ ( tập hợp các tích tối giản ký hiệu là **P** )

b) Tìm trong tập hợp các tập con **C** của tập **P** tập hợp có trọng số cực tiểu các khối $\mathbf{C}_{min}$ thỏa mãn hệ thức $\mathbf{V}_1(f) \subset \mathbf{V}(c)$.

## 2. Tìm các tích tối giản trên cơ sở biểu diễn trực quan. Phương pháp bảng Karnaugh

Các tích tối giản có thể được tìm bằng những sơ đồ trực quan khi số biến của hàm $f$ nhỏ. Trong phần trên chúng ta đã chỉ ra rằng, hàm lôgic $f$ của $n$ biến độc lập có thể được cho bằng cách gán các giá trị 0,1 và $d$ cho các đỉnh của khối $n$ chiều. Giữa các khối n chiều và các tích tối giản có một mối liên hệ đơn giản. Theo định nghĩa tích tối giản có thể biểu diễn bằng một khối. Mặt khác một khối cấp $m$ là khối con $m$ chiều nằm trong khối $n$ chiều và tích tối giản là khối lớn nhất nằm trong tập hợp $\mathbf{V}_1(f) \cup \mathbf{V}_d(f)$. Khái niệm khối lớn nhất trong trường hợp này có nghĩa là chứa tất cả các đỉnh của khối và không tồn tại các khối có chiều lớn hơn nằm trong $\mathbf{V}_1(f) \cup \mathbf{V}_d(f)$

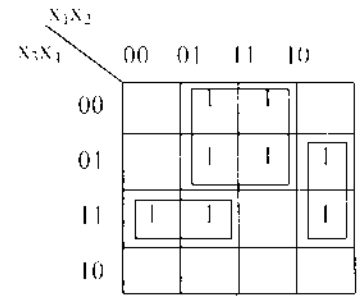
Thủ tục tìm kiếm tất cả các tích tối giản của hàm lôgic $f$ dựa vào tính chất nói trên đưa tới việc tìm tập hợp các khối có cấp lớn nhất có thể được với $m < n$ bao phủ tất cả các *đỉnh-1* và *đỉnh-d* của hàm $f$. Nếu $n \leq 3$ thuật toán có thể được thực hiện tường minh trên bản vẽ các khối; trong trường hợp $n \geq 4$ việc áp dụng phương pháp trực tiếp trở nên khó khăn.

**Phương pháp bảng Karnaugh.**

- Các đỉnh của các khối là các ô của bảng. Các ô của bảng có toạ độ như các đỉnh của khối. Giá trị trong ô là giá trị của hàm $f$ tại các đỉnh tương ứng.
- Các đỉnh được kết hợp theo quy tắc sau:

  Hai đỉnh lân cận ( hai khối bậc 0) tạo thành một khối bậc 1. Các đỉnh nằm trên biên của bảng cũng là các đỉnh lân cận với nhau. Các tọa độ độc lập được ký hiệu là $x$.

  - Bốn đỉnh lân cận có thể kết hợp tạo thành một khối bậc 2 chứa hai tọa độ độc lập.
  - Tám đỉnh lân cận tạo thành khối bậc 3.




**Hình 2.6** Phương pháp bảng Karnaugh.

Ví dụ: hàm lôgic được biểu diễn qua tập hợp các *đỉnh-1*

$$f = \mathbf{V}_1(3, 4, 5, 7, 9, 11, 12, 13) =$$

$$x_1x_2x_3\bar{x}_4 \vee \bar{x}_1x_2x_3\bar{x}_4 \vee x_1\bar{x}_2x_3x_4 \vee \bar{x}_1x_2\bar{x}_3x_4 \vee x_1\bar{x}_2\bar{x}_3x_4$$

$$\vee\ x_1\bar{x}_2x_3\bar{x}_4 \vee x_1x_2\bar{x}_3x_4 \vee x_1x_2x_3\bar{x}_4$$

$$= x_2\bar{x}_3 \vee x_1\bar{x}_2x_4 \vee x_1x_3x_4$$

* Để tối thiểu hóa các hàm ở dạng chuẩn tắc tuyển ta dùng các *đỉnh-1* và *đỉnh-d*.
* Với các hàm chuẩn tắc hội ta dùng các *đỉnh-0* và *đỉnh-d*.

## 3. Phương pháp tạo bảng theo các bước lặp. Phương pháp Quine - McCluskey

Tìm kiếm các tích tối giản theo bảng Karnaugh trở nên rất phức tạp và mất tính trực quan nếu số lượng các biến độc lập của hàm lôgic vượt quá năm. Ưu điểm của phương pháp bảng Karnaugh là ở chỗ cho phép xác định một cách trực quan tính liền kề của các ô. Một phương pháp khá quan trọng trong các bài toán tối thiểu hoá các biểu thức lôgic là phương pháp Quine-McCluskey. Phương pháp Quine - McCluskey cho phép xác định sự liền kề của các đỉnh bằng cách lập bảng.

* Hàm lôgic được cho bởi tập hợp các khối cấp 0. Các khối này được chia theo nhóm. Số lượng đơn vị trong các khối liền kề chênh lệch nhau một. Trong bảng của các khối cấp 0, ta sắp xếp các khối theo số lượng đơn vị và nhóm các khối có cùng số lượng đơn vị một cách tách biệt.

Ví dụ: cho hàm *f* biểu diễn bằng tập hợp các *đỉnh-1* và tập hợp các *đỉnh-d*

$$f(x_1, x_2, x_3, x_4) = \mathbf{V}_1(0, 2, 7, 8, 9, 13)$$

$$\mathbf{V}_0(f) = (3, 12, 10)$$

* Quá trình kiểm tra tính liền kề của các đỉnh được thực hiện với tất cả các tổ hợp các khối đối với hai nhóm lân cận. Nếu hai khối khác nhau bởi 1 và 0 chỉ ở một vị trí, vị trí đó được đặt *x* và nhận được một khối có bậc lớn hơn. Khối này được đưa vào bảng mới và đánh dấu '*v*' vào vị trí của các khối ban đầu được bao phủ bởi khối có cấp lớn hơn. Sau khi thực hiện bước này đối với tất cả các khối ta nhận được bảng mới có một nhóm ít hơn so với bảng ban đầu. Nếu trong bảng

mới này có hai hoặc nhiều hơn các khối giống nhau thì chúng sẽ bị loại bỏ.

- Quá trình trên sẽ được lặp lại với các khối cấp 1 cho tới khi chúng ta nhận được bảng đối với các khối cấp 2. Tiếp tục quá trình với các khối cấp 2, cấp 3, ..v.v cho tới khi ta không còn nhận được các bảng chứa dấu kiểm tra 'v'.

| Số lượng đơn vị | Khối cấp 0 | Kiểm tra |
|---|---|---|
| 0 | 0000 | v |
| 1 | 0010 | v |
| | 1000 | v |
| 2 | 0011 | v |
| | 1001 | v |
| | 1010 | v |
| | 1100 | v |
| 3 | 0111 | v |
| | 1101 | v |
| | **Khối cấp 1** | |
| 0 | 00x0 | v |
| | x000 | v |
| 1 | 001x | |
| | 100x | v |
| | 10x0 | v |
| | x010 | v |
| | 1x00 | v |
| 2 | 0x11 | |
| | 1x01 | v |
| | 110x | v |
| | **Khối cấp 2** | |
| 0 | x0x0 | |
| 1 | 1x0x | |

**Hình 2.7** Phương pháp Quine-McCluskey.

- Trong các bảng nhận được từ phương pháp lặp nối trên, những khối không được đánh dấu bởi ký hiệu 'v' là những tích tối giản của hàm lôgic ban đầu. Trong ví dụ ở hình bên, những tích tối giản là (001*x*), ( 0*x*11), (*x*0*x*0), (1*x*0*x*).
- Khi trong biểu diễn hàm có những *đinh-d*, tất cả những đinh đó sẽ được sử dụng như những *đinh-1* và nếu sau khi thực hiện còn lại một khối bao phủ tất cả các *đinh-d* thì khối đó có thể loại bỏ.

Phương pháp Quine-McCluskey dựa trên các phép lặp do đó cho phép ta có thể xây dựng các chương trình trên máy tính thực hiện tối thiểu hoá các hàm lôgic đối với những hàm có số biến lớn.

Ngoài những phương pháp nêu trên còn những phương pháp tối thiểu hoá các biểu thức lôgic dựa trên tích tổng quát, phương pháp tối thiểu hoá hệ các hàm lôgic, các phương pháp heuristic. Các phương pháp này cho phép giảm thời gian tối thiểu hoá các hàm lôgic phức tạp với sự trợ giúp của máy tính.

## 4. Phương pháp tìm các tích tối giản thông qua tích kết hợp

Phương pháp trực quan cũng như phương pháp dùng bảng để tìm các tích

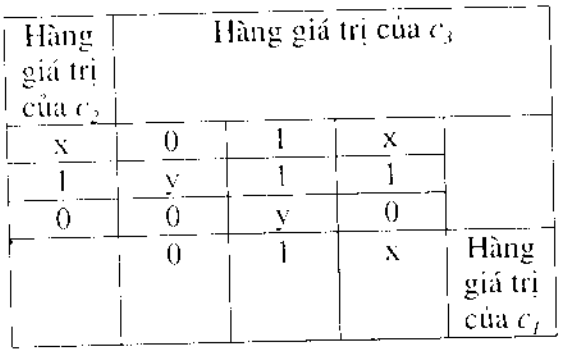
tối giản yêu cầu phải biểu diễn hàm lôgic ban đầu bằng những khối cấp 0. Khi số biến độc lập của hàm lôgic tăng lên, số lượng các khối cấp 0 sẽ tăng theo tỷ lệ hàm mũ, do đó để tìm những tích tối giản theo các khối có cấp tuỳ ý, trong kỹ thuật thường áp dụng các phương pháp đại số. Sau đây chúng ta nghiên cứu phương pháp tối thiểu hoá dựa trên tính toán các khối. Phương pháp này là tổng quát hoá phương pháp Quine-McCluskey.

**Định nghĩa tích kết hợp:** Tích kết hợp $c_3$ của hai khối $c_1$, $c_2$ là khối có cấp cực đại thoả mãn:

$$V(c_3) \subset V(c_1) \cup V(c_2),\ V(c_3) \neq V(c_1),\ V(c_3) \neq V(c_2).$$

Để có thể nhận được tích kết hợp ta thực hiện các phép tính theo hình 2.8 đối với những hàng giá trị cùng tên của các khối. Các kết quả sẽ được thể hiện như sau:

- Nếu trong kết quả của các thao tác đó ký hiệu $y$ xuất hiện trong hai hàng hoặc nhiều hơn thì không tồn tại tích kết hợp. Trong trường hợp này hai đỉnh bất kỳ tương ứng nằm trong $V(c_1)$ và $V(c_2)$ không có chung cạnh trong khối $n$ chiều.




**Hình 2.8** Các phép toán tìm tích kết hợp.

- Nếu $y$ không xuất hiện điều đó có nghĩa là có một khối che phủ khối kia và không phải là tích tối giản. Trong trường hợp $y$ xuất hiện đúng tại một vị trí, nếu thay $y$ bởi $x$ ta nhận được một khối thoả mãn các điều kiện của định nghĩa tích kết hợp. Do khối nhận được chứa $y$ nên khối đó thoả mãn điều kiện: $V(c_3) \neq V(c_1)$, $V(c_3) \neq V(c_2)$; ngoài ra do kết quả của phép toán giữa $x$ và 1 cũng như giữa $x$ và 0 bằng 1 và 0 tương ứng, khối nhận được thoả mãn điều kiện $V(c_3) \subset V(c_1) \cup V(c_2)$. Theo kết quả của các thao tác, ta nhận được khối có bậc cực đại.

Việc tìm các tích tối giản trong tập hợp bất kỳ các khối C có thể diễn ra theo sơ đồ sau:

- Loại bỏ khối $c_1$ ra khỏi tập hợp C nếu $V(c_1) \subset V(c_2)$, $c_1, c_2 \in C$.

- Thực hiện tìm tích kết hợp đối với $c_1, c_2 \in \mathbf{C}$ và hợp vào $\mathbf{C}$ trong trường hợp cho phép.

Thủ tục này được thực hiện cho đến khi ta không thể thêm vào tập $\mathbf{C}$ các khối mới. Do thủ tục này bao hàm cả thủ tục hợp các khối liền kề đã được mô tả ở phần trên, kết quả ta nhận được các tích tối giản.

Tóm lại trong chương hai, chúng ta đã nghiên cứu cơ sở lôgic để xây dựng các mạch số. Cơ sở này bao gồm đại số chuyển mạch - đại số Bool, biểu diễn các hàm lôgic bằng các dạng chuẩn tắc và các phương pháp tối thiểu hoá các biểu thức lôgic. Những vấn đề này nằm trong nền tảng cơ bản của kỹ thuật thiết kế các mạch lôgic.

# CHƯƠNG III. CƠ SỞ CỦA THIẾT KẾ LÔGIC

Các mạch tích hợp xử lý các thông tin được biểu diễn trong hệ nhị phân. Khi xây dựng mạch, các phép toán thường được biểu diễn qua các hàm lôgic, do đó biểu diễn các hàm lôgic trong các mạch tích hợp bằng những phương tiện đặc thù là cơ sở của thiết kế lôgic. Thông thường, các hàm lôgic có thể thực hiện dưới dạng các mạch tổ hợp, nhưng trong phần lớn các trường hợp các mạch có độ tích hợp cao thực hiện những chức năng lặp lại theo thời gian. Để lưu trữ các kết quả trung gian của các phép toán và thực hiện quá trình lặp theo thời gian các chức năng tổ hợp được thực hiện trên những mạch thao tác tuần tự. Cơ sở thiết kế các mạch tác động tuần tự dựa là mô hình các ôtômat hữu hạn. Trong chương này chúng ta sẽ nghiên cứu vấn đề thiết kế các mạch tổ hợp và thiết kế các mạch tác động tuần tự.

## §3.1. Đặc điểm của quá trình thiết kế mạch máy tính

Trong chương này chúng ta nghiên cứu các phương pháp thiết kế các khối chức năng của các thiết bị tính toán. Các thiết bị tính toán này thực hiện các vi thao tác đối với các tín hiệu tác động. Các vi thao tác tương ứng với thang trật tự thấp nhất trong thang phân cấp các phương pháp biểu diễn các thiết bị tính toán.

Cấu trúc của các khối chức năng phức tạp hơn cấu trúc của các phần tử lôgic. Do đó việc nghiên cứu các hoạt động của các khối đó trên cơ sở mạch điện tử nói chung không thể thực hiện được. Trong kỹ thuật thiết kế, người ta nghiên cứu hoạt động của các khối chức năng một cách gần đúng mà không cần thiết phải tìm hiểu các sơ đồ mạch điện trong trường hợp khối là những cấu trúc lôgic. Các cấu trúc lôgic này được xây dựng từ những phần tử lôgic lý tưởng.

### 1. Đánh giá thời gian trễ trong các mạch lôgic

Trong quá trình thiết kế các thiết bị tính toán, nhà thiết kế không chỉ quan tâm đến chức năng thực hiện các phép toán lôgic của mạch mà còn cần

thiết tính đến cả thời gian trễ của tín hiệu khi đi qua các phần tử lôgic và các đoạn mạch. Thời gian trễ này ảnh hưởng lớn đến hoạt động của mạch trong thực tế. Do đó việc mô tả và xử lý các giá trị thời gian trễ trong các thiết bị tính toán đóng vai trò quan trọng.

Trong mô hình đơn giản nhất và phổ biến nhất của các phần tử lôgic, một thuộc tính của thời gian truyền tín hiệu qua mạch là thời gian trễ thuần tuý $t_d$. Trong trường hợp này, thời gian trễ của mạch gồm các phần tử chức năng mắc nối tiếp sẽ bằng tổng các thời gian trễ của các phần tử chức năng và thời gian trễ của các phần tử liên kết. Thông thường thời gian trễ trong các phần tử là những đại lượng ngẫu nhiên, do đó việc tính đến các giá trị thời gian trễ phải sử dụng các phương pháp thống kê.

Thông thường các nhà sản xuất đưa ra giá trị cực đại của thời gian trễ. Đôi khi để cung cấp đầy đủ thông tin hơn về thời gian trễ, người ta có thể đưa ra giá trị cực đại và những giá trị tiêu biểu. Một cách đầy đủ hơn ta có thể cung cấp giá trị cực đại và cực tiểu của thời gian trễ. Trong những trường hợp cần độ chính xác cao người ta cần phải cung cấp những đặc tính thống kê của thời gian trễ như kỳ vọng toán học E, phương sai $\sigma^2$ và mô tả sự phụ thuộc của thời gian trễ vào những điều kiện môi trường bên ngoài như nhiệt độ, độ ẩm, độ dẫn điện, ...

Nếu chỉ biết giá trị cực đại của thời gian trễ, đối với một mạch truyền tín hiệu bất kỳ, ta chỉ có thể khẳng định được rằng tín hiệu khi đi qua mạch sẽ bị trễ **không lớn hơn** tổng các giá trị cực đại của thời gian trễ. Từ đó một hệ quả quan trọng nảy sinh là chúng ta *không có khả năng so sánh thời gian lan truyền tín hiệu qua các đoạn mạch khác nhau*. Trong mọi đoạn mạch giá trị thời gian trễ có thể nhỏ tuỳ ý.

Nếu chúng ta biết giá trị cực đại và cực tiểu của thời gian trễ, khi đó trong mọi đoạn mạch ta có thể xác định giá trị cực đại và cực tiểu của thời gian trễ. Trong trường hợp này, ta có thể so sánh được các thời gian trễ nhưng kết quả có thể xác định hoặc không xác định. Điều đó phụ thuộc vào các tham số thời gian trễ của các phần tử và số lượng các phần tử trong đoạn mạch.

Trong những trường hợp giới hạn đã biết, thời gian trễ thường được xác định bằng cách tính đến những khả năng xấu nhất trong quá trình truyền tín hiệu. Cách tính này dựa trên giả thiết như sau: thời gian trễ của tín hiệu khi đi qua các phần tử có thể được xác định thông qua những trở ngại khi đảm bảo những chế độ làm việc xác định của phần tử. Với chế độ làm việc lý

tưởng, thời gian trễ có thể coi như gần bằng không. Khi không thể đảm bảo được chế độ làm việc lý tưởng do ảnh hưởng từ bên ngoài, thời gian trễ của tín hiệu sẽ tăng lên. Trong trường hợp đơn giản nhất khi tín hiệu không bị phần tử làm méo, thời gian trễ lan truyền có thể được coi là độ lệch pha của tín hiệu ra tương ứng với tín hiệu vào. Nói chung phương pháp này chỉ tính đến những trường hợp có khả năng xảy ra nhỏ nhất trong mạch. Các điều kiện ràng buộc có thể được xác định như sau.

Ta xét hai đường truyền tín hiệu, một đường chứa $N_L$ phần tử, đường thứ hai chứa $N_S$ phần tử. Giả thiết rằng $N_L > N_S$. Đối với những mạch tốc độ cao, ta cần phải tính đến thời gian trễ của cả những mạch liên kết. Tổng thời gian trễ trong những mạch liên kết được ký hiệu là $t_d$ và sẽ được tính gộp với thời gian trễ của phần tử.

Thời gian trễ tối thiểu trên đường tín hiệu chứa $N_L$ nhóm phần tử sẽ bằng

$$t_{L\min} = N_L . t_{d\min};$$

Thời gian trễ tối đa trên đường tín hiệu chứa $N_S$ nhóm phần tử sẽ bằng:

$$t_{S\max} = N_S . t_{d\max}$$

Trong quá trình thiết kế ta cần thoả mãn điều kiện $t_L \geq t_S$ hay là:

$$N_L / N_S > t_{d\max} / t_{d\min}.$$

Như vậy, nếu hệ thức trên thoả mãn thì trong trường hợp xấu nhất tín hiệu truyền theo đường có ít phần tử sẽ nhanh hơn theo đường chứa nhiều phần tử.

Nếu tính đến những đặc tính thống kê của thời gian trễ chúng ta có thể có những đánh giá chính xác hơn về thời gian trễ của tín hiệu khi đi qua các phần tử so với phương pháp đánh giá theo khả năng xấu nhất.

Giả thiết rằng, các giá trị thời gian trễ là các đại lượng ngẫu nhiên độc lập và có phân bố xác suất gần với dạng phân bố Gauss. Giả thiết này được dựa trên cơ sở các nghiên cứu đặc tính của các phần tử lôgic.

Với những điều kiện đó, thời gian trễ trong một đoạn mạch gồm các phần tử mắc nối tiếp sẽ được coi là đại lượng ngẫu nhiên có phân bố xác suất dạng Gauss với kỳ vọng toán học E và phương sai $\sigma^2$ là tổng của các kỳ vọng toán học $E_i$ và tổng của các phương sai $\sigma_i^2$ của phân bố xác suất của các phần tử thành phần. Khi đó hiệu $\Delta$ của thời gian trễ trên đoạn mạch có nhiều phần tử với thời gian trễ trên đoạn mạch có ít phần tử hơn cũng là đại lượng ngẫu nhiên có phân bố xác suất dạng Gauss với kỳ vọng toán học E bằng:

$$E(\Delta) = E(t_L) -$$

trong đó E( $t_L$ )và E( $t_S$ ) là kỳ vọng toán học của thời gian trễ trên đoạn mạch L và S; và phương sai

$$D(\Delta) = D(t_L) + D(t_S)$$

trong đó $D(t_L)$ và $D(t_S)$ là phương sai của thời gian trễ trên đoạn mạch L và S.

Để tránh xung đột thì tín hiệu trên đường có nhiều phần tử phải đến chậm hơn tín hiệu trên đường có ít phần tử hơn một khoảng thời gian lớn hơn hoặc bằng $t_0$.

Xác suất để điều kiện này bị phá vỡ có thể được xác định như sau: ta xác định đại lượng:

$$n = \frac{E(\Delta) - t_0}{\sigma(\Delta)}$$

trong đó, $\sigma(\Delta) = \sqrt{D(\Delta)}$

Xác suất điều kiện tránh xung đột bị phá vỡ là xác suất của trường hợp đại lượng ( $\Delta - t_0$ ) lệch khỏi $E(\Delta)$ một đoạn bằng $n$ độ lệch quy chuẩn. Với giá trị $n > 3$, xác suất này được tính gần đúng theo công thức:

$$P \approx \left\{\exp\left[-n^2/2\right]\right\} / n\sqrt{2\pi}$$

Khi xác định giá trị thời gian trễ, ta cần phải tính đến ảnh hưởng của nhiệt độ, của tải, ... lên hoạt động của phần tử.

Các tính toán nêu trên trong trường hợp thời gian trễ là đại lượng ngẫu nhiên được sử dụng cho trường hợp mạch được xây dựng từ những phần tử riêng biệt. Khi mạch được tạo một cách đồng nhất trên một tinh thể, sự tản mạn tương đối của thời gian trễ giảm đi do sự tương quan giữa các phần tử mạch, thời gian trễ trở nên gần như tất định.

## 2. Các mạch tổ hợp và các mạch tuần tự

Sự phân chia các mạch số thành các mạch tổ hợp và các mạch tuần tự xuất phát từ các điểmkhác biệt cơ bản giữa các đặc tính của chúng.

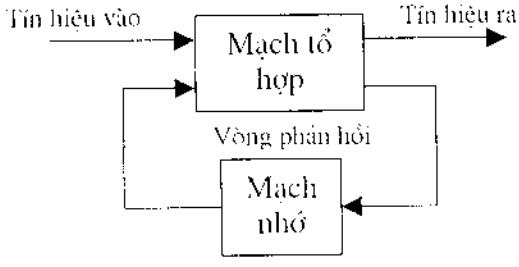
Các biến đầu ra của các mạch tổ hợp chỉ phụ thuộc vào các tác động vào mạch tại thời điểm hiện tại.

Các mạch tuần tự tính toán các giá trị ra dựa vào các giá trị đầu vào không chỉ tại thời điểm hiện tại mà còn phụ thuộc cả vào những trạng thái của mạch tính từ thời điểm đang xét trở về trước. Các trạng thái của mạch

tuần tự được lưu trữ vào các phần tử nhớ trong thành phần của mạch. Trạng thái của mạch tại một thời điểm là hàm số của các trạng thái của mạch và các giá trị đầu vào tại các thời điểm trước đó. Như vậy mạch tuần tự biến đổi một chuỗi các giá trị của các tín hiệu vào thành chuỗi các giá trị của tín hiệu ra. Các mạch tuần tự được cấu tạo bởi hai phần: các bộ phận nhớ để lưu trữ các trạng thái của mạch; và mạch tổ hợp dùng để điều khiển các phần tử nhớ và hình thành các giá trị tín hiệu ra.

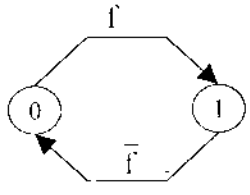
Trong kỹ thuật tính toán, các mạch tổ hợp là các mạch mã hoá, giải mã, bộ so sánh tín hiệu, bộ cộng. Các mạch tuần tự là các trigơ, các mạch nhớ, thanh ghi, bộ đếm, ... Các phương pháp tổng hợp và phân tích các mạch tổ hợp đơn giản hơn so với mạch tuần tự.

Trong quá trình thiết kế, các mạch số thường được biểu diễn bằng nhiều phương pháp, ví dụ như bằng các bảng, ma trận, đồ thị hặc bằng các ôtômat.




**Hình 3.1** Biểu diễn mạch số bằng ôtômat.

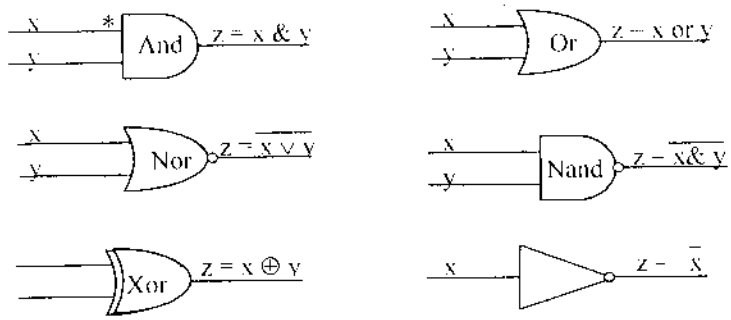
Tuỳ thuộc vào kích thước và độ phức tạp của mạch mà chúng ta lựa chọn phương pháp thích hợp. Một cách tổng quát, cả mạch tổ hợp lẫn mạch tuần tự có thể được biểu diễn bằng các ôtômat. Các ôtômat có thể được biểu diễn bằng sơ đồ kết hợp giữa mạch nhớ và mạch tổ hợp. Mạch nhớ dùng để lưu trữ trạng thái còn mạch tổ hợp dùng để tính các trạng thái mới và các tín hiệu ra mới dựa vào tín hiệu đầu vào, các trạng thái cũ. Ví dụ, một mạch tổ hợp bất kỳ có thể được biểu diễn bằng một ôtômat có hai trạng thái '1' và '0'; tín hiệu đầu ra được xác định theo trạng thái của ôtômat; hàm chuyển trạng thái vào trạng thái '1' chính là hàm lôgic biểu diễn chức năng mạch; từ trạng thái '1' hệ thống chuyển về trạng thái '0' bằng tín hiệu xác định bằng hàm đảo của hàm chức năng.



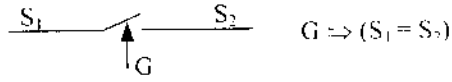

**Hình 3.2** Sơ đồ ôtômat biểu diễn mạch tổ hợp.

## §3.2. Các phần tử lôgic cơ bản

Trong quá trình thiết kế các mạch tích hợp có một số phần tử lôgic cơ bản được sử dụng phổ biến. Việc thực hiện các phần tử lôgic này phụ thuộc vào công nghệ sản xuất linh kiện điện tử như công nghệ transistor CMOS, công nghệ transistor trường, TTL, TTLS, ...v.v. Các phần tử lôgic cơ bản gồm phần tử AND, OR, NOT, XOR, NOR, NAND, ngoài ra trong nhiều trường hợp phần tử đóng ngắt cũng được coi là phần tử cơ bản. Trên hình 3.1 đưa ra ký hiệu các phần tử cơ bản với hai đầu vào.

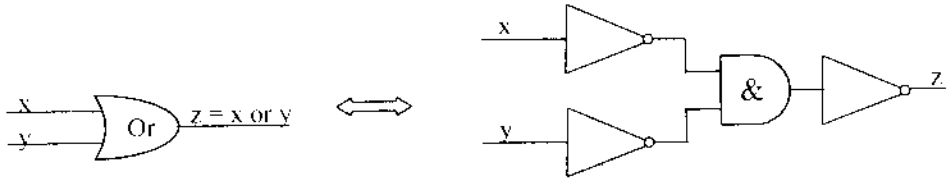



**Hình 3.3** Các phần tử lôgic cơ bản.




x, y: các đầu tín hiệu vào, z: đầu tín hiệu ra, G: đường tín hiệu điều khiển, $S_1$, $S_2$: các tín hiệu.

Trên quan điểm về khả năng xây dựng các hàm lôgic bất kỳ, một số phần tử cơ bản hợp thành hệ đầy đủ. Điều đó có nghĩa là với các hàm cơ bản tham




**Hình 3.4** Xây dựng phần tử OR bằng các phần tử NOT và AND.

gia vào hệ đầy đủ, ta có thể xây dựng mọi hàm lôgic. Ta có hệ các phần tử AND, OR, NOT tạo thành một hệ đầy đủ vì ta có thể xây dựng mọi hàm

lôgic theo các dạng chuẩn tắc tuyển hoặc hội với sự tham gia của các phần tử nói trên. Dưới đây ta xét một số hệ đầy đủ các phần tử lôgic cơ bản.

- Hệ phép toán bao gồm các phần tử NOT và AND.

Ta nhận thấy phép toán OR có thể được biểu diễn như sau qua NOT-AND :

$$z = x \vee y = \overline{\bar{x} \wedge \bar{y}} \qquad (3.1)$$

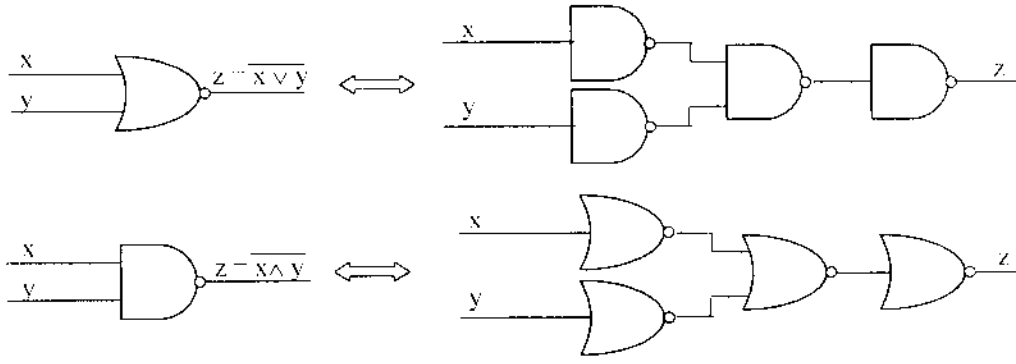
Do đó hệ hàm gồm các phần tử NOT và AND tạo thành một hệ đầy đủ.

- Hệ phép toán bao gồm các phần tử NOT và OR.

  Ta có phép toán AND có thể được xây dựng trên cơ sở của phép toán NOT và OR theo hệ thức dưới đây. Do đó hệ hàm bao gồm phần tử NOT và OR cũng tạo thành hệ đầy đủ.

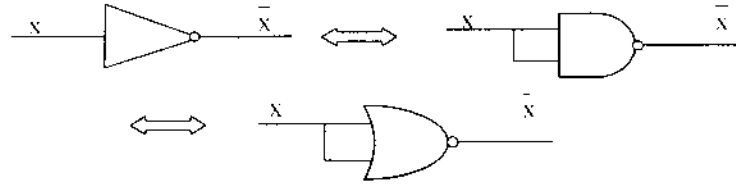
$$z = x \wedge y = \overline{\bar{x} \vee \bar{y}} \qquad (3.2)$$

- Hệ NAND và NOR:




**Hình 3.5** Xây dựng phần tử NOR/NAND trên cơ sở phần tử NAND/NOR.

Trong kỹ thuật thiết kế các mạch số, các phần tử trong hệ NOT-AND ( NOT-OR ) được kết hợp lại thành phần tử NAND ( NOR ). Trên hình 3.5




**Hình 3.6** Xây dựng phần tử NOT trong hệ NAND/NOR.

là cơ sở xây dựng phần tử NOR (NAND) trong hệ phần tử NAND ( NOR ). Ta nhận thấy, trong hệ phần tử NAND, cũng giống như trong hệ phần tử NOR, phần tử NOT có thể nhận được một cách đơn giản từ phần tử NAND hoặc NOR như trong hình 3.6. Các phần tử AND và OR cũng dễ dàng nhận được từ các phần tử NAND và NOR qua các hệ thức lôgic đơn giản sau:

- Trong hệ phần tử NAND.

$$x \wedge y = \overline{\overline{x \wedge y}} \qquad (3.3)$$

$$x \vee y = \overline{\overline{x} \wedge \overline{y}} \qquad (3.4)$$

- Trong hệ phần tử NOR.

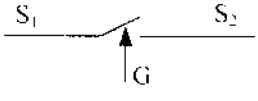
$$x \wedge y = \overline{\overline{x} \vee \overline{y}} \qquad (3.5)$$

$$x \vee y = \overline{\overline{x \vee y}} \qquad (3.6)$$

Như vậy hệ phép toán chỉ có một phần tử NAND hoặc một phần tử NOR là một hệ đầy đủ.

- Phần tử đóng ngắt.

Về khía cạnh lôgic phần tử này là phần tử truyền tín hiệu. Phần tử đóng ngắt thực hiện chức năng như một rơle. Nếu trên đường điều khiển G tín hiệu nhận giá trị '1', khi đó khoá đóng và tín hiệu $S_1$ được truyền tới đường $S_2$. Nếu giá trị tín hiệu G bằng '0', khoá mở và tín hiệu không truyền qua đường $S_1S_2$. Do phần tử này có chức năng như một mạch đóng ngắt, như đã chỉ ra trong mục 2.1 của chương 2, ta có thể xây dựng các hàm lôgic chỉ từ các mạch đóng ngắt. Trong trường hợp này ta cần có đường tín hiệu $\bar{x}$ đi đôi với đường tín hiệu của biến x, do đó những mạch lôgic được xây dựng từ những phần tử đóng ngắt thường được gọi là mạch lôgic hai đường. Trong những mạch này các đường tín hiệu thường xuất hiện theo cặp $x$ và $\bar{x}$. Trong công nghệ chế tạo những mạch có độ tích hợp siêu lớn VLSI, những mạch một đường thường được sử dụng. Khi đó giá trị $\bar{x}$ được tạo ra bằng phần tử NOT. Do đó ta có thể nói rằng các phần tử đóng ngắt và phần tử NOT tạo thành một hệ đầy đủ các phép toán.

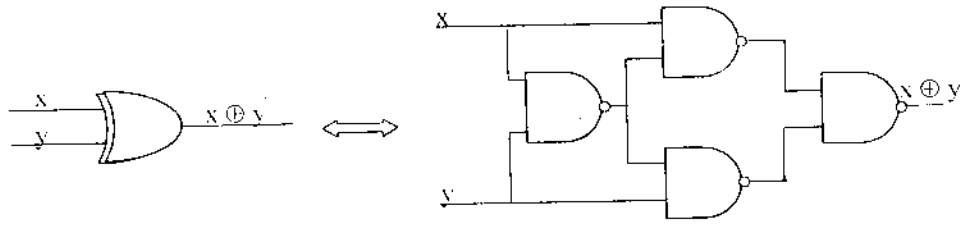


Hình 3.7 Phần tử đóng ngắt.

- Phần tử XOR.

Phần tử XOR thực hiện phép toán loại trừ lôgic. Phần tử này đứng riêng biệt không thể tạo thành hệ đầy đủ của các phép toán lôgic. Trong mục 2.2 của chương 2 chúng ta đã thấy một hàm lôgic có thể được tạo thành từ các

phép toán AND và XOR. Như vậy hệ các phép toán AND và XOR tạo thành hệ đầy đủ của các phép toán. Phần tử NOT có thể được tạo thành từ phần tử




**Hình 3.8** Xây dựng phần tử XOR từ các phần tử NAND.

XOR bằng cách cho một đường tín hiệu luôn nhận giá trị '1'. Phần tử XOR được xây dựng như sau trên các phần tử NAND.

Ngoài các phần tử lôgic hai đầu vào nêu trên, trong công nghệ còn sử dụng những phần tử có nhiều hơn hai đầu vào. Các phần tử có nhiều đầu vào có thể được biểu diễn như ghép nối nhiều lớp các phần tử lôgic có số lượng đầu vào ít hơn hoặc như một phần tử duy nhất. Các phương pháp xây dựng các mạch đó được lựa chọn dựa vào các tiêu chuẩn tối ưu về diện tích tinh thể bán dẫn của mạch, năng lượng mà mạch tiêu thụ và thời gian trễ truyền của tín hiệu khi đi qua mạch.

## §3.3. Thiết kế các mạch tổ hợp

### 1. Tổng hợp mạch theo biểu thức lôgic

Thông thường các hàm lôgic được biểu diễn bằng những biểu thức lôgic chứa những phép toán AND, OR và XOR, NOT. Những biểu thức đó có thể được thực hiện thành mạch thông qua những phần tử lôgic cơ sở.

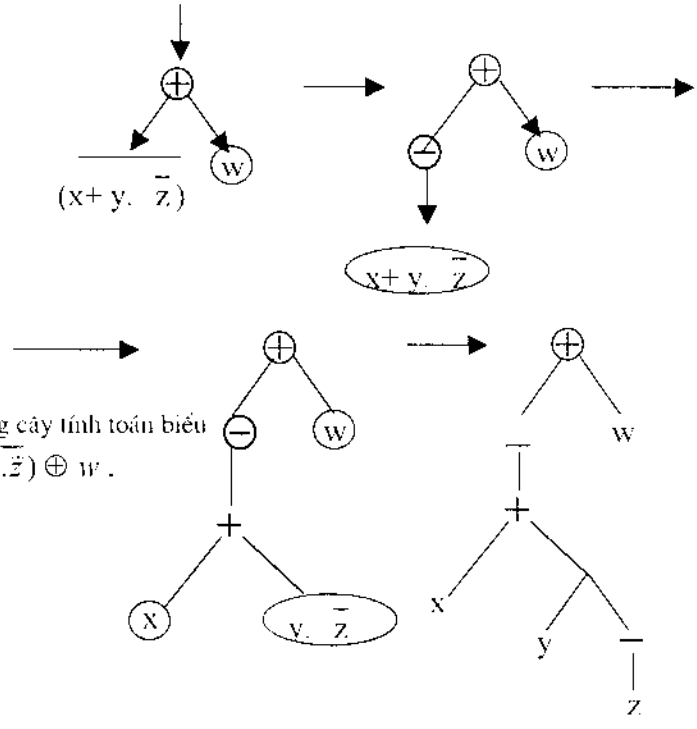
Biểu thức có thể được phân tách dưới dạng các cây tính toán tương ứng với các dấu ngoặc và mức độ ưu tiên của các phép toán. Điều này có thể đạt được nếu ta áp dụng phương pháp phân tích biểu thức toán học bằng cách viết Balan và bỏ các dấu ngoặc. Sau đó dựa vào cách viết Balan của biểu thức ta xây dựng cây biểu diễn việc tính toán biểu thức. Dựa vào cây tính

toán ta xây dựng sơ đồ mạch bằng cách thay thế mỗi biến bằng một đường tín hiệu, thay thế mỗi phép toán bằng một phần tử lôgic cơ bản. Sau đó ta xây dựng mạch theo các hệ đầy đủ được trình bày trong phần trước. Mỗi phần tử cơ bản được thay thế bằng sơ đồ tương đương trong hệ đầy đủ được lựa chọn và giản ước mạch bằng những phép biến đổi lôgic, ví dụ như hai phần tử NOT mắc nối tiếp có thể loại bỏ.

Ta hãy xét một ví dụ thiết kế mạch thực hiện biểu thức lôgic: $f = \overline{(x + y.\bar{z})} \oplus w$.

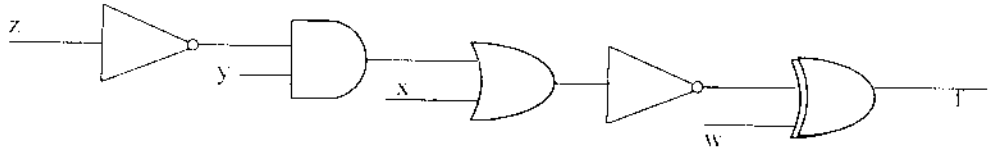
Biểu thức này được viết dưới dạng cách viết Balan như sau: $f = wxyz\ \bar{}\ \cdot + \oplus$ . Các bước phân tách được thực hiện lần lượt cho đến khi nhận được một cây hoàn chỉnh, với các nút trong là các phép toán và các nút lá là các biến .

Trong ví dụ trên : ta sẽ có các bước xây dựng cây như sau:



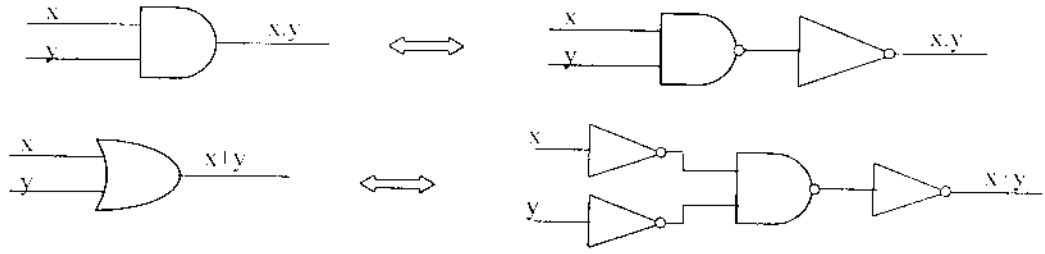

**Hình 3.9** Xây dựng cây tính toán biểu thức $f = \overline{(x + y.\bar{z})} \oplus w$ .

Theo cây biểu diễn trên ta có thể xây dựng sơ đồ biểu diễn mạch theo các phép toán cơ sở: XOR, AND, OR và NOT.

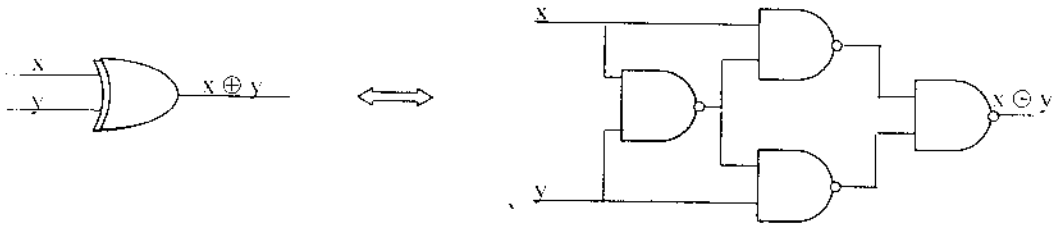



**Hình 3.10** Xây dựng sơ đồ mạch theo cây biểu diễn từ các phần tử AND, OR, XOR, NOT.

Để xây dựng sơ đồ trên các phần tử cơ sở NAND và NOT hoặc NOR và NOT thì ta sẽ xây dựng các phần tử AND, OR, XOR từ các phần tử của hệ đầy đủ trên:

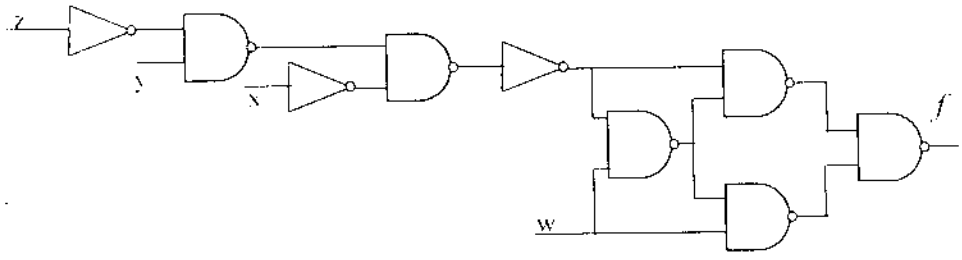



**Hình 3.11a** Biểu diễn tương đương của các phần tử AND, OR, XOR trong hệ đầy đủ NAND.



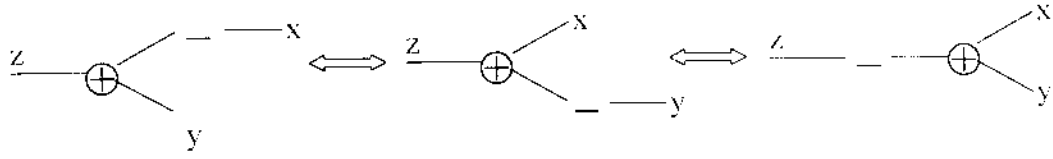

**Hình 3.11b** Biểu diễn tương đương của phần tử XOR trong hệ đầy đủ NAND.

Sau khi áp dụng các biểu diễn tương đương của các phần tử lôgic trong các hệ đầy đủ khác nhau, ta nhận được sơ đồ sau:




**Hình 3.12** Sơ đồ mạch thực hiện biểu thức $f = \overline{(x + y.\bar{z})} \oplus w$ trên hệ NAND và NOT.

Nếu phần tử XOR tham gia vào trong thiết kế, ta có thể sử dụng các hệ thức tương đương trên hình 3.11. Các hệ thức này được suy ra trực tiếp từ




**Hình 3.13** Các sơ đồ tương đương của phép loại trừ lôgic XOR.

Điều đó có nghĩa là toán tử NOT trong biểu thức của tổng XOR có thể nằm ở mọi vị trí.

Tóm lại phương pháp xây dựng mạch trực tiếp từ các biểu thức lôgic như đã nêu trên là phương pháp đơn giản. Với phương pháp này ta có thể xây dựng được mọi hàm lôgic với những độ phức tạp khác nhau. Phương pháp này có một nhược điểm lớn liên quan tới thời gian trễ của tín hiệu đi qua mạch và độ dài của tín hiệu. Từ ví dụ trên ta thấy, đối với các đầu vào x, y, z, w, các tín hiệu tác động phải có độ dài khác nhau. Điều đó là do cấu trúc phân tầng của mạch. Tín hiệu $z$ để đi ra được đầu ra $f$ phải đi qua bảy phần tử, trong khi đó tín hiệu $w$ chỉ cần đi qua ba phần tử. Như vậy ta phải tính toán độ dài các tín hiệu tác động và thời điểm tác động sao cho nhận được giá trị đúng ở đầu ra $f$.

## 2. Thực hiện các mạch hai tầng

Ta có thể mở rộng phương pháp trên và dựa trên cơ sở các phần tử có nhiều đầu vào, ta có thể xây dựng các sơ đồ dưới dạng chính tắc: AND-OR, OR-AND, AND XOR. Trong các phương pháp thiết kế các mạch hai tầng trên cơ sở các phần tử AND-OR và OR-AND, bên cạnh biến $x_i$ ta cần thêm

các giá trị của $x_i$. Khi đó cấu trúc mạch trở thành ba tầng do sử dụng thêm phần tử NOT để tạo nên phần tử bù.

Phương pháp xây dựng mạch trực tiếp từ các dạng chuẩn tắc được mô tả trong phần trước trên thực tế không có hiệu quả. Trong trường hợp biểu diễn hàm lôgic dưới dạng tổng các tích lôgic, trường hợp xấu nhất ta cần ( $2^n - 1$ ) tích cực tiểu và khi đó cần ( $2^n - 1$ ) phần tử AND với $n$ đầu vào và một phần tử OR với ( $2^n - 1$ ) đầu vào. Trong trường hợp tham gia $x_i$ và $\bar{x}_i$ khi đó cần thêm nhiều nhất là $n$ phần tử NOT trong trường hợp xây dựng mạch một đường.

Nếu tích cực tiểu của $n$ biến thực hiện như ở phần trước, khi đó cần tổng cộng ( n – 1 ) phần tử AND với hai đầu vào hoặc một phần tử AND với $n$ đầu vào. Tương tự một phần tử OR với $m$ đầu vào có thể thay bởi ( m – 1 ) phần tử OR với hai đầu vào. Khi xây dựng mạch các phần tử lôgic có $n$ đầu vào cần nhiều diện tích trên tinh thể bán dẫn hơn khi xây dựng trên những phần tử có hai đầu vào. Do đó giá thành của mạch cũng phụ thuộc vào số lượng đầu vào của các phần tử lôgic.

Đối với việc thiết kế các mạch tổ hợp sử dụng sơ đồ hai tầng thông qua các phần tử AND và OR, điều quan trọng là giảm được số lượng các phần tử AND và OR. Bên cạnh việc giảm số lượng các phần tử, một điều quan trọng là giảm số lượng đầu vào của các phần tử đó.

Ta có: tập hợp các khối có giá thành nhỏ nhất cho ta sơ đồ hai tầng có giá thành nhỏ nhất trên cơ sở AND-OR.

Chúng ta hãy xét một ví dụ: ta thiết kế mạch thực hiện hàm

$$f = V\,(0,1,2,5,7,8,10,15)$$

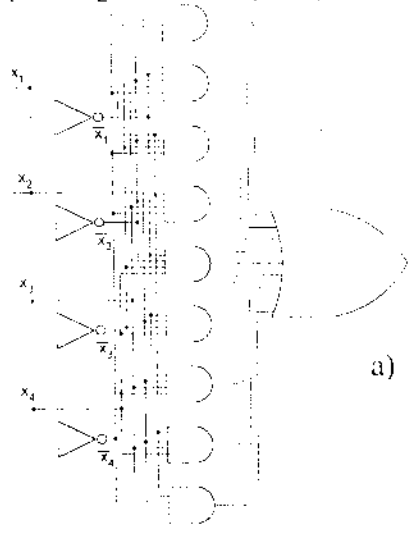
Sau khi tối thiểu hóa, ta có:

$$f = \{x0x0,\ 0x01,\ 111x,\ 01x1\}$$

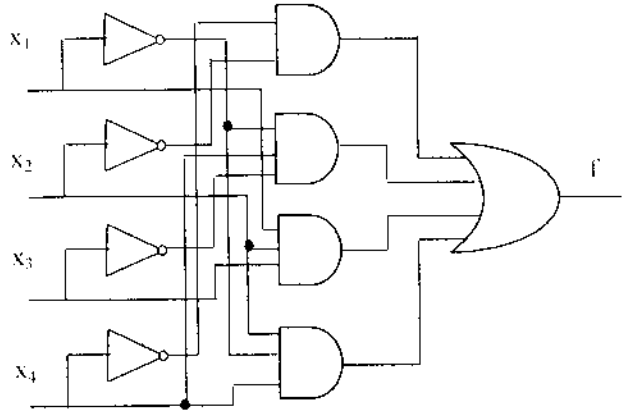
Sơ đồ thiết kế mạch thực hiện hàm $f$ được đưa ra trên hình 3.12 theo hai thiết kế: thiết kế theo dạng chuẩn tắc tuyển trực tiếp và thiết kế sau khi tối thiểu hoá. Ta nhận thấy thiết kế trực tiếp phức tạp hơn thiết kế sau khi đã tối thiểu hoá cả về số lượng các kết nối, cả về số lượng đầu vào của các phần tử lôgic.

Khi ta tối thiểu hoá các hàm lôgic, trọng số của tập hợp các khối được xác định bằng đại lượng đơn điệu tăng theo số vị trí biến trong khối ngoại trừ vị trí chứa x – tức là các vị trí của khối $m$ chiều ( $n - m$ ), và số lượng các khối tham gia vào tập hợp. Từ tập hợp khối này ta có thể đưa ra trực tiếp biểu thức lôgic dưới dạng chuẩn tắc tuyển. Mỗi khối tương ứng với một tích lôgic

và có thể được xây dựng trên phần tử AND có $n - m$ đầu vào. Biến $x_i$ tương ứng với giá trị '1' trong khối sẽ nối trực tiếp với phần tử AND còn biến $x_i$ tương ứng với giá trị '0' sẽ nối với phần tử lôgic AND qua phần tử NOT. Khi ta xây dựng biểu thức lôgic dưới dạng chuẩn tắc tuyển, mạch được thiết kế sẽ có dạng AND-OR. Số lượng đầu vào của phần tử OR sẽ bằng số lượng các khối tham gia vào tập hợp khối nói trên. Từ đó suy ra khi ta thiết kế mạch tổ hợp theo các mạch hai tầng AND-OR, tập hợp khối có trọng số nhỏ nhất sẽ tương ứng với mạch có giá thành thực hiện nhỏ nhất.

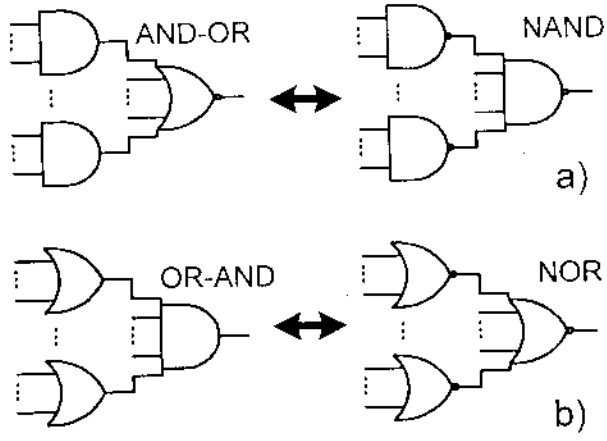



**Hình 3.14a** So sánh hai thiết kế hàm *f* trên mạch AND-OR. Trong hình a, thiết kế theo dạng chuẩn tắc.




**Hình 3.14b** So sánh hai thiết kế hàm *f* trên mạch AND-OR. Trong hình b, mạch được thiết kế với giá thành nhỏ nhất.

Khi ta biểu diễn biểu thức lôgic dưới dạng chuẩn tắc hội, sơ đồ sẽ được thực hiện dưới dạng các phần tử OR-AND. Giữa các sơ đồ dạng AND-OR và OR-AND có sự chuyển tiếp về các dạng NAND và NOR như trên hình 3.15. Điều đó được suy ra trực tiếp từ định lý de Morgan: $f_1 + f_2 \equiv \bar{f}_1 \cdot \bar{f}_2$.

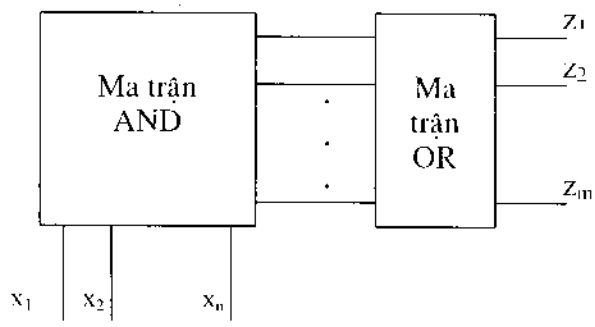



**Hình 3.15** a. Sự tương đương giữa mạch AND-OR và NAND.
b. Sự tương đương giữa mạch OR-AND và NOR.

Trong phần này ta không quan tâm tới vấn đề giới hạn số lượng đầu vào của các phần tử cơ bản ( giới hạn theo các nhánh đi vào đầu vào ) và giới hạn theo tải đặt vào đầu ra ( số lượng rẽ nhánh ở đầu ra ) của chúng. Biện pháp thông thường để loại bỏ các giới hạn này là tăng số lượng các phần tử lôgic và tăng số lượng các lớp phần tử.

### 3. Thực hiện mạch tổ hợp trên cơ sở các PLA/ROM

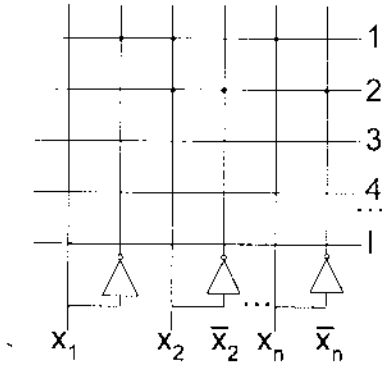
Ma trận lôgic lập trình là các khối phần tử vĩ mô được sử dụng để thiết kế những mạch LSI, VLSI theo cấu trúc các mạch hai tầng. Các PLA bao gồm ma trận AND và ma trận OR được mắc nối tiếp. Ma trận thứ nhất là ma trận AND, ma trận thứ hai là ma trận OR. Từ mục trước ta thấy rằng cấu trúc hai tầng AND - OR tương đương với cấu trúc hai tầng của các phần tử NAND. Theo nguyên lý đối ngẫu, cấu trúc đó cũng tương đương với cấu trúc hai tầng NOR – NOR. Do đó trong kỹ thuật thiết kế các cấu trúc hai tầng NAND-NAND và NOR-NOR được sử dụng rộng rãi.




**Hình 3.16** PLA theo cấu trúc AND-OR.

Như ta thấy các PLA thực hiện hệ các hàm lôgic. Ta có hệ thống các hàm lôgic có thể có được biểu diễn dưới dạng tập hợp các khối được đánh dấu. Nếu tập hợp **T** các khối được đánh dấu là cho trước thì việc lập trình cho PLA trở nên đơn giản hơn.

Trên hình 3.15 giới thiệu sơ đồ thực hiện ma trận AND. Ma trận AND được tạo bởi những tuyến dữ liệu sắp xếp theo những đường thẳng đứng và những đường ngang. Các tuyến dữ liệu vào đặt thẳng đứng thể hiện các đầu vào. Thông thường trong thiết kế hai đường, ta có $n$ đầu vào dành cho $x_i$ và $n$ đầu cho $\overline{x_i}$. Còn trong thiết kế một đường ta chỉ có n đường dữ liệu vào cho $x_i$, các giá trị đảo của $x_i$ $\overline{x_i}$ được tạo ra bởi lớp các phần tử NOT. Các đầu ra của ma trận AND sẽ là các đầu vào cho ma trận OR và được sắp xếp nằm ngang.




**Hình 3.17** Sơ đồ ma trận AND của PLA.

PLA được lập trình bằng cách đưa ra các điểm nối vào giao điểm của các đường. Mỗi đầu ra của ma trận AND thể hiện một tích lôgic. Như vậy sử dụng PLA ta có thể xây dựng các mạch lôgic từ các dạng chuẩn tắc tuyển hoặc hội một cách đơn giản. Để giảm độ phức tạp của thiết kế trong công nghệ thường sử dụng các phương pháp tối thiểu hoá hệ các hàm lôgic.

Trong ví dụ ở hình bên, các tuyến dữ liệu ra sẽ tương ứng với các biểu thức hội sau:

$$1: \bar{x}_1 x_2 \cdots x_n$$
$$2: \bar{x}_2 x_3 \cdots \bar{x}_n$$
$$\ldots$$
$$L: x_1 x_2 \ldots \bar{x}_n$$

Ma trận OR được lập trình tương tự như ma trận AND.

Diện tích mà PLA chiếm trong mạch VLSI tỷ lệ với giá trị $(2n + m).l$, trong đó $n$: số đầu vào, $l$ – số các tích lôgic trong dạng chuẩn tắc tuyển, $m$ – số lượng các đầu ra. Các giá trị $n$ và $m$ đặc trưng cho hệ hàm lôgic đã cho, còn $l$ xác định số lượng các phần tử trong tập hợp các khối được đánh dấu sẽ sử dụng.

Một phần tử hay được sử dụng của cấu trúc VLSI và cũng cho phép lập trình được như PLA là ROM. ROM khác PLA ở chỗ ROM là cấu trúc cho phép lập trình các giá trị 1 và 0 đối với các tích cực tiểu một cách tuỳ ý. Có $n$ đầu vào, mạch ROM về cơ bản không khác PLA với $2^n$ tuyến dữ liệu của các tích lôgic. Nhược điểm chính của ROM so với PLA là ROM có hiệu suất sử dụng diện tích tinh thể thấp. Điểm đặc biệt của ROM là khả năng lập trình tương ứng với tập hợp $V_1(f)$ hoặc bảng chân lý. So với PLA, ROM có độ mềm dẻo cao hơn trên quan điểm thay đổi các hàm lôgic trong hệ hàm, do đó ROM được sử dụng trong kỹ thuật tính toán rộng rãi hơn PLA. Đôi khi, để thay thế ROM, người ta có thể sử dụng các bộ nhớ truy cập ngẫu nhiên trong đó có ghi sẵn các bảng chân lý.

## §3.4. Những vấn đề khi thiết kế mạch tổ hợp

### 1. Những giai đoạn thiết kế mạch tổ hợp

Quá trình thiết kế mạch tổ hợp thường được thực hiện theo những bước sau:

- Khảo sát những đặc điểm về chức năng của mạch tổ hợp, những liên kết của mạch với những mạch khác theo đầu vào/đầu ra, thiết lập các quan hệ tương ứng với các biến lôgic.

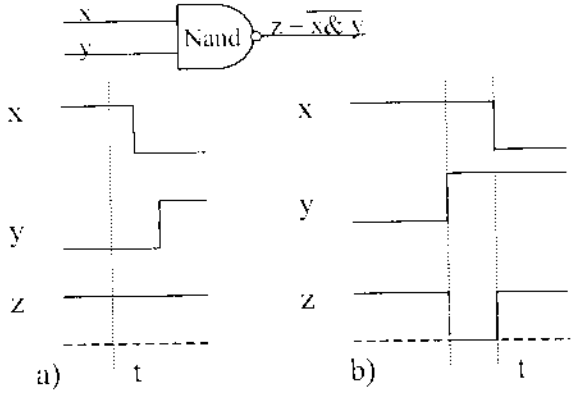
- Đánh giá kích thước của bài toán và giải quyết vấn đề phân chia mạch tổ hợp thành các phân hệ theo mức độ cần thiết.
  Giai đoạn này giải quyết vấn đề tiếp cận mạch tổ hợp cần thiết kế theo tổng thể hoặc phân chia thành các phân hệ. Việc phân chia mạch thành các phân hệ sẽ làm giảm độ phức tạp của thiết kế nếu số lượng phần tử mạch quá lớn. Ví dụ như khi thiết kế các khối chức năng xử lý các từ máy, một cách tự nhiên người ta phân chia mạch thành những phần tương ứng với từng hàng bit của từ máy.
- Xây dựng bảng thể hiện các chức năng của mạch tổ hợp.
- Tối thiểu hoá mạch.
  Các bảng, biểu đồ chức năng là các nguồn thông tin cho quá trình tối thiểu hoá. Ở đây, chúng ta phải hiểu quá trình tối thiểu hoá theo nghĩa rộng. Điều đó có nghĩa là chúng ta tối ưu hoá không chỉ biểu diễn các hàm lôgic mà còn tối ưu hoá mọi khâu trong toàn bộ quá trình thực hiện thiết kế hàm theo những tiêu chuẩn xác định.
- Lựa chọn các phần tử lôgic và biểu diễn hàm lôgic theo các hệ cơ sở đã lựa chọn.
  Trên giai đoạn này, những hàm lôgic đã tối thiểu hoá sẽ được biểu diễn dưới dạng chuẩn tắc hội hoặc chuẩn tắc tuyển. Sau đó các hàm lôgic sẽ được biểu diễn bằng những hệ đầy đủ NAND và NOR. Các thiết kế tương ứng sẽ phụ thuộc vào các hệ đầy đủ được lựa chọn.

## 2. Ảnh hưởng của thời gian trễ tới hoạt động của các mạch tổ hợp

Ảnh hưởng của thời gian trễ tới hoạt động của mạch có thể làm thay đổi hoàn toàn chức năng mạch. Đối với các mạch tổ hợp, thời gian trễ không chỉ làm giảm tốc độ hoạt động của mạch mà còn có thể sinh ra những giá trị nhất thời bị sai ở đầu ra của mạch. Điều đó sẽ làm hoạt động của toàn hệ thống có thể bị thay đổi. Theo thời gian những giá trị này sẽ biến mất và đầu ra của mạch sẽ nhận được các giá trị được tính theo các hàm lôgic đã thiết kế. Nhưng các giá trị sai này rất nguy hiểm trong những trường hợp khi mạch tổ hợp được nối với các mạch nhớ dùng lưu trữ các trạng thái của hệ thống. Khi đó sẽ xuất hiện các trạng thái không dự đoán trước và hoạt động của toàn hệ thống có thể bị sai hoàn toàn. Những trường hợp này gọi là các rủi ro trong mạch.

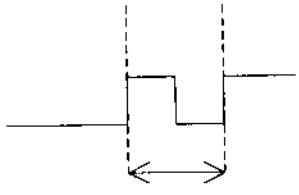
Trong kỹ thuật thường phân biệt hai loại rủi ro: rủi ro tĩnh và rủi ro động.

– Rủi ro tĩnh xuất hiện khi trạng thái của tín hiệu ra cuối cùng không thay đổi nhưng có thể xuất hiện những thay đổi trong khoảng thời gian ngắn. Chúng ta xét mạch thực hiện phép toán $z = \overline{xy}$. Do đối với từng đường tín hiệu, giá trị thời gian trễ có thể khác nhau. Khi tín hiệu đầu vào chuyển trạng thái từ '10' sang '01' tín hiệu đầu ra của phần tử NAND phải không thay đổi. Trong trường hợp trên hình 3.18 a), khi tín hiệu ở hai đầu $x$ và $y$ cùng bị trễ, tín hiệu đầu ra $z$ vẫn không thay đổi giá trị. Trong khi đó ở hình b) ở đầu ra của mạch xuất hiện giá trị tín hiệu z = '0' do hiện tượng trễ tín hiệu trên hai đường $x$ và $y$. Giá trị này không được hàm chức năng của mạch dự đoán và chỉ xuất hiện trong một khoảng thời gian ngắn. Đó chính là giá trị rủi ro tĩnh xuất hiện trong phần tử NAND.

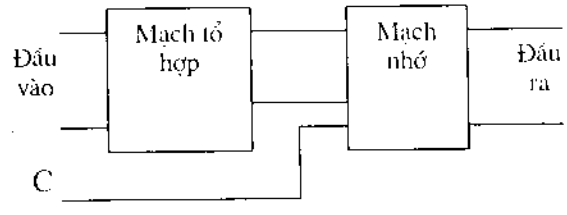



**Hình 3.18** Rủi ro tĩnh trong mạch NAND.

– Trường hợp thứ hai là trường hợp xuất hiện rủi ro động. Trong trường hợp này quá trình chuyển tiếp đầu tiên và cuối cùng luôn trùng với quá trình chuyển tiếp theo thuật toán của hoạt động mạch tổ hợp.




**Hình 3.19** Trường hợp xuất hiện rủi ro động.




**Hình 3.20** Loại bỏ rủi ro bằng các mạch nhớ.

Để loại trừ khả năng lỗi xuất hiện do rủi ro, trong các mạch tổ hợp người ta sử dụng đồng bộ quá trình nhận thông tin bằng các mạch nhớ nối với đầu ra của mạch tổ hợp. Thông tin được nhận vào mạch nhớ thông qua tín hiệu đồng bộ C. Tín hiệu này được tác động vào mạch nhớ sau khi

các quá trình quá độ trong mạch tổ hợp kết thúc. Như vậy các tín hiệu sai sẽ không tác động đến phần tử nhớ và do đó không xuất hiện trên đầu ra của mạch.

## §3.5. Thiết kế các mạch tuần tự

Các mạch tuần tự trong kỹ thuật thường được thiết kế theo cấu trúc bao gồm các mạch tổ hợp liên kết với các mạch nhớ. Các mạch tổ hợp sẽ thực hiện các tính toán theo các hàm lôgic, còn các mạch nhớ dùng để lưu trữ các kết quả trung gian. Do đó ta có thể biểu diễn hoạt động của mạch theo thời gian. Trong mục này chúng ta sẽ khảo sát các mạch nhớ và các phương pháp liên kết chúng với các mạch tổ hợp thành các mạch tuần tự.

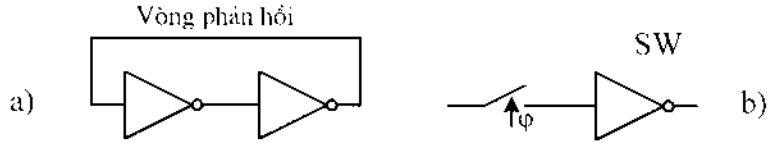
### 1. Nguyên lý của các mạch nhớ

Các mạch tổ hợp cho phép thực hiện một số mạch phức tạp, ví dụ như mạch nhân nhanh, nhưng đối với một số thao tác xử lý dữ liệu phức tạp hơn yêu cầu ghi nhớ các kết quả tính toán trung gian và thực hiện những thao tác lặp tương ứng với trình tự tính toán. Trong lĩnh vực xử lý số, các dữ liệu được biểu diễn dưới dạng nhị phân do đó cần thiết những mạch cho phép nhớ lại hai trạng thái '*0*' và '*1*'.

Có hai loại sơ đồ nhớ kinh điển đó là các mạch nhớ động và mạch nhớ tĩnh. Trong mục này ta xem xét các dạng mạch nhớ được xây dựng trên những phần tử lôgic đơn giản nhất – phần tử NOT. Việc xây dựng các phần tử nhớ đều dựa trên các nguyên lý chung. Đó là hoặc phải xây dựng các vòng phản hồi tín hiệu trong mạch để tự duy trì giá trị của ô nhớ; hoặc là dùng thiết bị phụ trợ để duy trì giá trị ô nhớ.

Hai nguyên lý trên được minh hoạ trên hình 3.21. Trong hình a, mạch nhớ được xây dựng từ hai phần tử NOT mắc nối tiếp và một vòng phản hồi. Khi mạch ở trạng thái ổn định đầu ra của hai phần tử NOT lần lượt là '1', '0' hoặc '0', '1' . Những giá trị đầu ra của các phần tử NOT cùng với vòng phản

hồi có tác dụng duy trì trạng thái của phần tử nhớ. Như vậy mạch này có tác dụng lưu trữ các giá trị dữ liệu '1' và '0'. Mạch nhớ này gọi là mạch nhớ tĩnh.




**Hình 3.21** Các dạng mạch nhớ: a. mạch nhớ tĩnh; b. mạch nhớ động.

Trên sơ đồ b) phần tử khóa SW nối tiếp với phần tử NOT. Phần tử nhớ này lưu trữ giá trị dữ liệu của ô nhớ bằng phần tử điện dung ký sinh tại đầu vào của phần tử lôgic NOT. Khi tín hiệu điều khiển $\varphi = 1$, khóa SW đóng và điện dung ký sinh tại đầu vào phần tử NOT được tích điện. Khi $\varphi = 0$, khóa SW mở và điện tích ở đầu vào phần tử NOT bị cô lập với mạch ngoài. Do đó phần tử NOT cần phải có giá trị trở kháng đầu vào cao. Ở trường hợp này, trong sản xuất thường xây dựng mạch theo công nghệ MOS. Thời gian lưu trữ được xác định theo thời gian lưu giữ điện tích của phần tử điện dung đầu vào. Thông thường thời gian này phụ thuộc vào nhiệt độ và có giá trị trong khoảng từ 1 giây đến $10^{-6}$ giây. Do đó muốn lưu trữ lâu dài giá trị của ô nhớ cần phải có những chu kỳ nạp lại điện tích trên phần tử điện cảm. Loại phần tử nhớ này gọi là phần tử nhớ động.

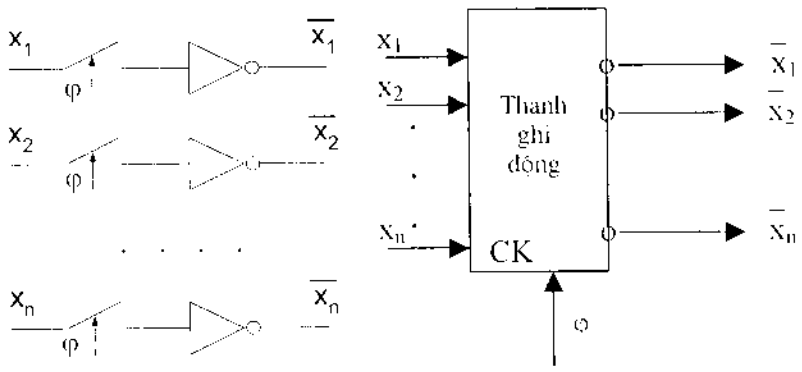
Trong cả hai trường hợp, dữ liệu được nhớ như là điện tích trên đầu vào của phần tử NOT. Nhưng đối với phần tử nhớ tĩnh điện tích luôn được nạp lại do đường tín hiệu phản hồi. Để sản xuất phần tử nhớ tĩnh người ta có thể sử dụng công nghệ MOS cũng như công nghệ mạch hai cực và đối với phần tử nhớ tĩnh không có giới hạn về thời gian nhớ.

## 2. Các sơ đồ thanh ghi và trigơ

Sơ đồ nhớ động cho phép lưu trữ trực tiếp một bit thông tin. Để có.thể ghi nhớ đồng thời được $n$ bit thông tin người ta dùng song song $n$ phần tử nhớ động. Thiết bị đó được gọi là thanh ghi động $n$ bit ( hình 3.22 ).

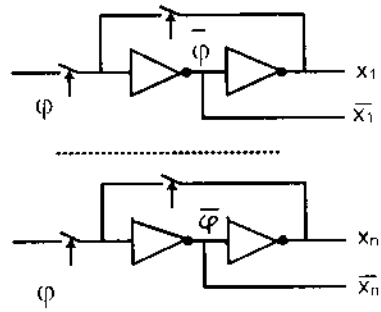
Đối với phần tử nhớ tĩnh vì trong đó không có mạch thu nhận dữ liệu, trong kỹ thuật thường dùng hai phương pháp xây dựng thanh ghi như sau.

- *Phương pháp 1:*




**Hình 3.22** Sơ đồ cấu trúc thanh ghi động.

Trong mạch đầu vào và mạch phản hồi ta dùng các bộ khóa SW (hình 3.23). Các khoá này được điều khiển sao cho chúng luôn ở hai trạng thái ngược nhau. Để đạt được điều đó các tín hiệu điều khiển sẽ là φ và $\overline{\varphi}$. Khi tín hiệu φ = 0, $\overline{\varphi}$ = 1 mạch vào sẽ ngắt và vòng phản hồi đóng lại mạch ở trạng thái nhớ. Khi φ = 1, $\overline{\varphi}$ = 0 mạch vào đóng và vòng phản hồi bị ngắt, mạch ở trạng thái ghi. Mạch sẽ có hai đầu ra: đầu giá trị thuận và đầu giá trị đảo. Đặc điểm của mạch là thời gian lưu trữ không bị giới hạn. Khi sử dụng song song $n$ mạch nhớ ta sẽ có thanh ghi tĩnh $n$ bit. Do trong mạch sử dụng các khoá nên phương pháp này không thích hợp với các mạch bán dẫn hai cực.



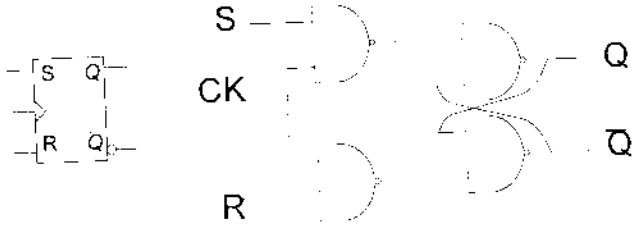

**Hình 3.23** Thanh ghi tĩnh

*Phương pháp 2:*

Thay vì dùng các phần tử NOT trong sơ đồ của phần tử nhớ, ta dùng các phần tử NAND hoặc NOR. Đó là các trigơ RS, D, T, JK. Trên hình 3.24 ta có sơ đồ cấu trúc và hoạt động của RS trigơ. Ta có thể xây dựng các phần tử trigơ khác dựa trên cơ sở trigơ RS. Trong hình 3.24 đưa ra cách xây dựng trigơ D từ trigơ RS.

Trong sơ đồ trigơ RS ta thấy khi tín hiệu CK = '1' và hai tín hiệu R với S cùng bằng '1', đầu ra Q và $\overline{Q}$ cùng bằng '1'. Khi CK chuyển sang giá trị '0', trạng thái này trở nên không ổn định và do tính bất đối xứng của mạch (thời gian trễ truyền trên các phần tử mạch không như nhau ) và các nhiễu động trong mạch mà một trong hai đường tín hiệu sẽ có giá trị '0'. Quá trình


**Hình 3.24** Các mạch trigơ RS và trigơ D.



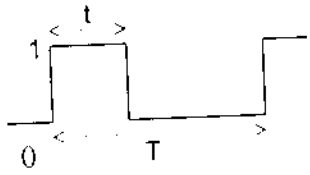

CK = 1
S=1, R=0 : ghi
S=0, R=1 : xóa
S=0, R=0 : giữ trạng thái
S=1, R=1 : không xác định, trạng thái cấm.
CK = 0, mạch giữ trạng thái.

chuyển tiếp này diễn ra trong khoảng thời gian rất ngắn sau khi CK chuyển trạng thái. Như vậy Q hoặc $\overline{Q}$ sẽ nhận giá trị '0' một cách ngẫu nhiên và không thể định trước được khoảng thời gian chuyển trạng thái, điều này dẫn tới những tính toán sai trong mạch và trạng thái này thường là trạng thái bị cấm.

Các phần tử trigơ có thể dùng để lưu trữ các bit thông tin và nếu ghép nối song song n phần tử trigơ ta sẽ nhận được thanh ghi *n* bit. Thanh ghi này là thanh ghi tĩnh và hoạt động tương tự như thanh ghi tĩnh xây dựng bằng các phần tử NOT.

### 3. Các chế độ đồng bộ

Tín hiệu điều khiển các đầu vào của mạch nhớ thường cung cấp bởi các mạch đồng bộ. Đầu vào tín hiệu đồng bộ tương ứng của mạch gọi là các đầu vào đồng bộ. Tín hiệu đồng bộ được xác định bởi độ dài khoảng thời gian *t* mà tín hiệu ở trạng thái '1' và chu kỳ T. Các mạch nhớ sử dụng tín hiệu đồng bộ trong các mục trước thực hiện các thao tác đọc và ghi dữ liệu khi đầu vào đồng bộ nhận giá trị 1. Các mạch loại này gọi là các mạch làm việc trong chế độ đồng bộ theo mức. Khi thiết kế các mạch làm việc theo chế độ đồng bộ, độ dài của tín hiệu đồng bộ đóng một vai trò quan trọng trong hoạt động của mạch. Nếu các trigơ D đồng bộ theo mức được ghép nối nối tiếp và nối chung các đầu tín hiệu đồng bộ ta sẽ có thanh ghi làm việc theo chế độ như sau. Nếu tín hiệu đồng bộ có độ dài đủ lớn thì sau một khoảng thời gian tất

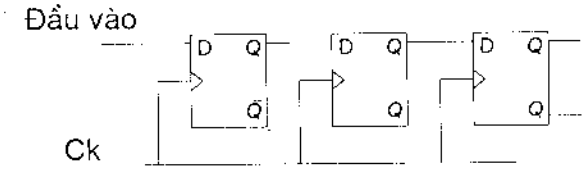



**Hình 3.25** Tín hiệu đồng bộ.

cả các trigơ sẽ ghi cùng một giá trị. Trong trường hợp ngược lại, khi độ dài $t$ của tín hiệu đồng bộ thoả mãn hệ thức:

$$\tau_{max} < t < d_{min}, \qquad (3.7)$$

trong đó $\tau_{max}$ là giá trị cực đại trong các độ dài cực tiểu của các xung đảm bảo việc ghi dữ liệu; $d_{min}$ là thời gian trễ cực tiểu khi tín hiệu lan truyền trong mạch nhớ từ đầu vào tới đầu ra. Khi đó thanh ghi thực hiện việc dịch dữ liệu sang phải một lớp sau mỗi lần nhận tín hiệu đồng bộ. Chế độ đồng bộ thoả mãn bất đẳng thức 3.7 gọi là đồng bộ hẹp.

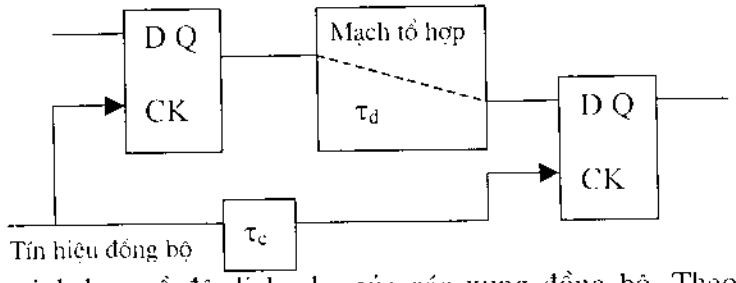



**Hình 3.26** Thanh ghi dịch sử dụng đồng bộ hẹp.

Ưu điểm của chế độ đồng bộ hẹp là đơn giản, dễ thực hiện và để xây dựng mạch phát xung đồng bộ không cần nhiều thiết bị. Nhưng nói chung chế độ đồng bộ này ít được sử dụng trong các mạch VLSI bởi vì trên tất cả các phần của mạch ta khó có thể đạt được bất đẳng thức 3.7. Trong nhiều trường hợp để có thể kiểm soát được hành vi của các mạch người ta sử dụng một biến thể của chế độ đồng bộ hẹp, đó là chế độ làm việc theo sườn.

Đồng bộ theo sườn dốc là chế độ làm việc của các mạch với tín hiệu đồng bộ có độ rộng của xung lớn và tác động thực hiện theo sườn lên hoặc sườn xuống của xung. Ở chế độ này các tác động của tín hiệu đồng bộ đối với mạch xảy ra cũng tương tự như trong chế độ đồng bộ hẹp. Tốc độ hoạt động của mạch trong trường hợp này được xác định bởi tốc độ hoạt động của chính các phần tử mạch do đó điều kiện về bất đẳng thức đồng bộ hẹp được thỏa mãn.

Nhược điểm của phương pháp đồng bộ theo sườn dốc và đồng bộ hẹp là miền dịch pha cho phép của xung đồng bộ. Độ dịch tương đối của các xung là hiện tượng xuất hiện độ dịch pha của các xung trên đầu vào đồng bộ của các phần tử nhớ trong mạch VLSI. Độ dịch pha của các xung đồng bộ xuất hiện chủ yếu do kết quả của sự phân bố các thời gian trễ trong các mạch phân phối tín hiệu đồng bộ.



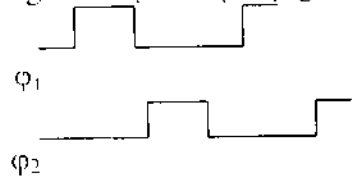

**Hình 3.27** Minh hoạ độ dịch pha của tín hiệu đồng bộ.

Trên hình 3.27 minh hoạ về độ dịch pha của các xung đồng bộ. Theo mạch này phần tử trigơ D thứ hai nhận tín hiệu đồng bộ CK trễ so với trigơ thứ nhất là $\tau_c$, dữ liệu được truyền từ đầu ra Q của trigơ thứ nhất đến đầu vào D của trigơ thứ hai thông qua một mạch tổ hợp có thời gian trễ $\tau_d$. Sự khác biệt trong thời gian xử lý gắn liền với thực hiện bất đẳng thức.

$$\tau_d < \tau_c \qquad (3.8)$$

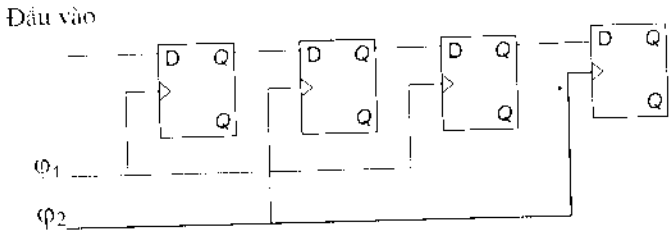
Bất đẳng thức này xác định tương quan giữa thời gian trễ theo mạch truyền dữ liệu giữa các mạch nhớ ( $\tau_d$ ) và độ dịch theo thời gian của xung đồng bộ ( $\tau_c$ ). Độ dịch của xung đồng bộ tăng theo kích thước của mạch VLSI, do đó trong công nghiệp thường xây dựng những mạch sử dụng hệ thống đồng bộ với vùng hoạt động rộng.

Trong các mạch hoạt động theo các chế độ đồng bộ theo mức, đồng bộ hẹp người ta chỉ dùng một tín hiệu thực hiện chức năng đồng bộ. Dạng đồng bộ đó gọi là chế độ đồng bộ một pha. Trong nhiều trường hợp, để tăng độ tin cậy của mạch, người ta sử dụng nhiều tín hiệu đồng bộ lệch pha để điều khiển mạch. Những chế độ làm việc như vậy gọi là chế độ đồng bộ nhiều pha.




**Hình 3.28** Tín hiệu đồng bộ hai pha.

Trên hình 3.28 minh họa các tín hiệu đồng bộ của hệ thống đồng bộ hai pha. Tại đây ta sử dụng hai tín hiệu đồng bộ không phủ nhau tại trạng thái "1" là $\varphi_1$ và $\varphi_2$. Trong thiết kế của thanh ghi dịch, hai tín hiệu đồng bộ này được tác động lần lượt. Độ rộng của xung và chu kỳ của các tín hiệu $\varphi_1$ và $\varphi_2$ có thể được xác định sao cho trạng thái '1' của chúng không phủ nhau, do đó dữ liệu sẽ dịch từ một lớp trigơ sang lớp trigơ sau chính xác theo từng tín hiệu đồng bộ. Nhưng trong trường hợp này trong hai mạch nhớ có thể lưu giữ một bit dữ liệu và so với mạch đồng bộ một pha thì cần một số lượng các tầng nhớ nhiều hơn hai lần.

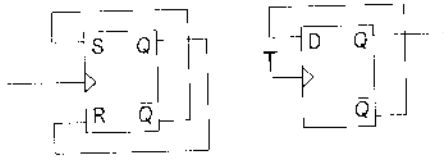



**Hình 3.29** Thanh ghi dịch làm việc theo chế độ đồng bộ hai pha.

Đồng bộ hai pha được thiết kế sao cho không có sự chồng nhau của các xung đồng bộ. Nhưng trong nhiều trường hợp có thể cho phép sự chồng nhau của các xung theo thời gian nhỏ hơn thời gian trễ $d_{min}$ từ đầu vào đến đầu ra của một mạch nhớ. Thông thường giá trị thời gian trễ $d_{min}$ lớn hơn thời gian trễ trên một phần tử lôgic, do đó $\varphi_2$ có thể nhận được từ $\varphi_1$ qua một mạch NOT sao cho đoạn giá trị phủ lên $\varphi_1$ nhỏ hơn $d_{min}$. Nói cách khác, đối với những phần không lớn của mạch ta có thể sử dụng tín hiệu $\overline{\varphi_1}$ thay cho $\varphi_2$.

## 4. Các trigơ sử dụng chế độ đồng bộ hẹp và đồng bộ hai pha

Trong kỹ thuật, nhiều mạch nhớ sử dụng các trigơ hoạt động theo các chế độ đồng bộ theo sườn lên hoặc làm việc theo chế độ đồng bộ hai pha.

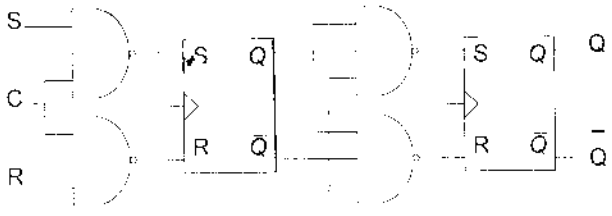



**Hình 3.30** Sơ đồ cấu trúc trigơ T.

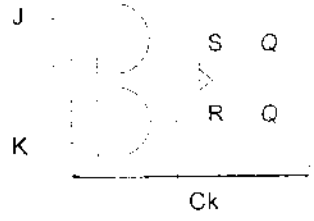
Các chế độ đồng bộ này cho phép giải quyết vấn đề cạnh tranh giữa các phần tử lôgic trong hoạt động của các trigơ. Điều này làm cho các mạch nhớ hoạt động tin cậy hơn và làm giảm tính bất định của mạch. Để minh họa vấn đề này ta hãy xét sơ đồ T-trigơ.

Về mặt cấu trúc và hoạt động trigơ T có thể được xây dựng từ trigơ $\overline{R}S$ bằng cách nối vòng đầu vào R với đầu ra Q và đầu vào S với đầu ra $\overline{Q}$. Trigơ T cũng có thể được xây dựng từ trigơ D bằng cách nối vòng đầu ra $\overline{Q}$ với đầu vào D. Cấu trúc đó không đảm bảo chế độ hoạt động ổn định. Trong mạch này phần tử nhớ trong thành phần của trigơ T đóng vai trò hai mặt. Về hoạt động ta nhận thấy rằng phần tử nhớ đóng hai vai trò: vừa là nguồn thông tin và vừa là bộ phận nhận thông tin. Thực hiện hai chức năng đó đồng thời là không thể được vì khi mạch nhận thông tin mới thì đồng thời xoá bỏ thông tin cũ. Nếu các phần tử lôgic có tốc độ tác dụng lý tưởng thì sơ đồ trên không làm việc. Sự tồn tại của thời gian trễ trong mạch cho phép thực hiện chế độ đếm trong một số điều kiện, nhưng do thời gian trễ trong mạch là đại lượng ngẫu nhiên độ tin cậy của mạch thấp.

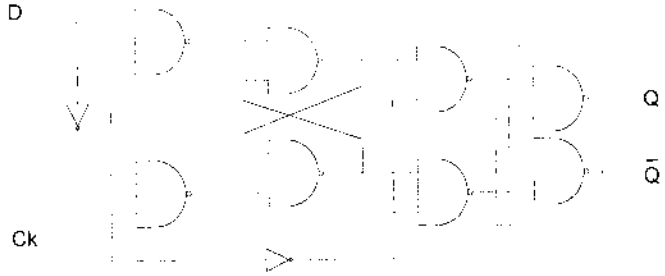
Để tăng độ tin cậy của mạch ta sử dụng hai phần tử nhớ trong đó một phần tử nhận thông tin vào trong khi trạng thái cũ vẫn giữ nguyên ở tầng ra. Khi các giá trị mới được hình thành ở lớp vào và thông tin cũ không cần thiết, dữ liệu sẽ được truyền từ tầng vào tới tầng ra. Các trigơ loại này làm



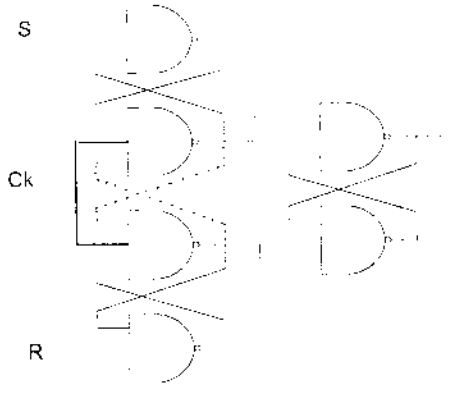

**Hình 3.31** Mạch trigơ RS làm việc theo chế độ đồng bộ hai pha.



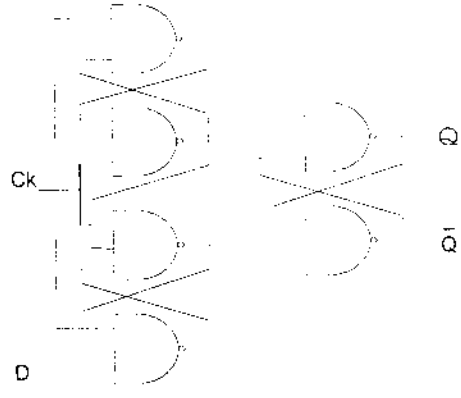

**Hình 3.32** Xây dựng trigơ JK từ trigơ RS.




**Hình 3.33** Mạch trigơ D làm việc theo chế độ đồng bộ hai pha.




**Hình 3.34** Mạch trigơ RS làm việc theo sườn lên.




**Hình 3.35** Mạch trigơ D làm việc theo sườn lên.

việc trong một chu trình. Nếu quan sát kỹ, ta sẽ thấy mạch trigơ loại này làm việc theo chế độ đồng bộ hai pha.

Các trigơ còn có thể được thiết kế để hoạt động theo sườn của tín hiệu đồng bộ. Trên hình 3.34, 3.35 trình bày thiết kế của trigơ RS và trigơ D làm việc theo sườn lên của tín hiệu đồng bộ. Trong các trigơ này ta có thể thay đổi các tín hiệu vào theo các mức bất kỳ của tín hiệu đồng bộ. Tín hiệu đồng bộ chỉ được kích hoạt trong khoảng thời gian nhỏ lân cận sườn lên hoặc sườn xuống.

## 5. Thiết kế các mạch tuần tự bằng các ôtômat hữu hạn

Trong mạch tổ hợp, các giá trị đầu ra được hoàn toàn xác định theo các tín hiệu đầu vào tại thời điểm hiện tại. Nhưng các mạch tuần tự có các giá trị đầu ra được xác định theo dãy các giá trị đầu vào tác động vào mạch tại những thời điểm trước thời điểm hiện tại. Trong kỹ thuật, việc thiết lập một dãy các giá trị đầu vào vô hạn là không thực hiện được với một số hữu hạn các phần tử lôgic nên các mạch tuần tự thường được biểu diễn thông qua một số hữu hạn các trạng thái.

Các mạch với số trạng thái hữu hạn có thể được biểu diễn theo những bảng hoặc sơ đồ chuyển trạng thái.Trong các mạch tuần tự, trạng thái ở một thời điểm phụ thuộc vào trạng thái ở những thời điểm trước đó của mạch. hay nói cách khác mạch tuần tự là mạch có nhớ. Thông thường số lượng các trạng thái của mạch tuần tự là hữu hạn. Do đó mạch dãy có thể được biểu diễn bằng những ôtômat có trạng thái hữu hạn.

Trong các mạch có số trạng thái hữu hạn, trạng thái bên trong phụ thuộc vào dãy các tín hiệu đầu vào tác động vào hệ thống. Cụ thể là giá trị đầu vào và trạng thái xác định giá trị đầu ra và trạng thái tiếp theo. Do số lượng các trạng thái bên trong là hữu hạn nên các dãy tín hiệu đầu vào có thể được phân chia thành các lớp tương đương. Nói cách khác những dãy tín hiệu đầu vào tạo ra sự chuyển trạng thái của mạch về cùng một trạng thái thì nằm trong một nhóm. Như vậy những mạch có số trạng thái hữu hạn về thực chất không phải là mô hình đầy đủ của mạch tuần tự. Trong những mạch tuần tự tín hiệu ra được xác định hoàn toàn theo dãy các tín hiệu đầu vào. Nói chung các mạch có số trạng thái hữu hạn có thể dùng để mô hình hoá các mạch tuần tự và mô hình càng chính xác nếu ta tăng số lượng các trạng thái bên trong. Mô hình này được ứng dụng rộng rãi trong quá trình thiết kế mạch.

Trong lý thuyết tính toán hai mô hình ôtômat hữu hạn hay được sử dụng nhất là ôtômat Moore và ôtômat Mealy.

Ôtômat Moore là một ôtômat hữu hạn bao gồm tập hợp các tín hiệu vào là { $x_1,x_2,...x_n$ }; tập hợp các tín hiệu ra là { $y_1,y_2,...,y_m$}; tập hợp các trạng thái { $a_1,a_2,...,a_k$ }. Trong ôtômat Moore, sự chuyển trạng thái phụ thuộc vào trạng thái trước đó và tín hiệu vào ở thời điểm trước đó. Quan hệ đó có thể được biểu diễn bằng hệ thức sau.

$$Z(t+1) = \Phi\ [Z(t), X(t)]; \qquad (3.9)$$

Tín hiệu đầu ra của ôtômat Moore chỉ phụ thuộc trạng thái:

$$Y(t) = \Psi\ [Z(t)]; \qquad (3.10)$$

Đối với ôtômat Mealy, sự chuyển trạng thái phụ thuộc vào trạng thái trước đó và tín hiệu vào ở thời điểm trước đó.

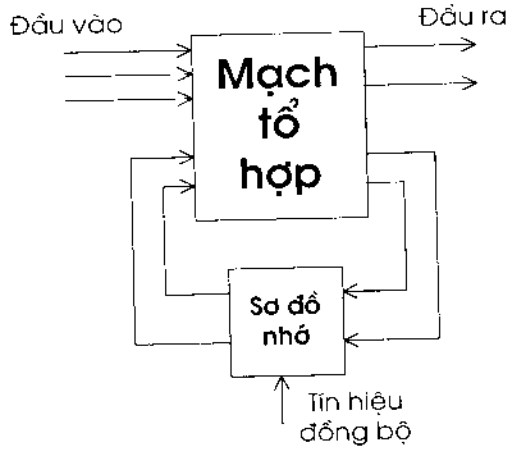
$$Z(t+1) = \Phi\ [Z(t), X(t)]; \qquad (3.11)$$

Tín hiệu đầu ra của ôtômat Mealey phụ thuộc vào trạng thái và tín hiệu đầu vào:

$$Y(t) = \Psi\ (Z(t), X(t)); \qquad (3.12)$$

trong đó: $\Phi(z,x)$ : hàm chuyển trạng thái;

$\Psi(z)$, $\Psi(z,x)$ : hàm xác định giá trị đầu ra của ôtômat.



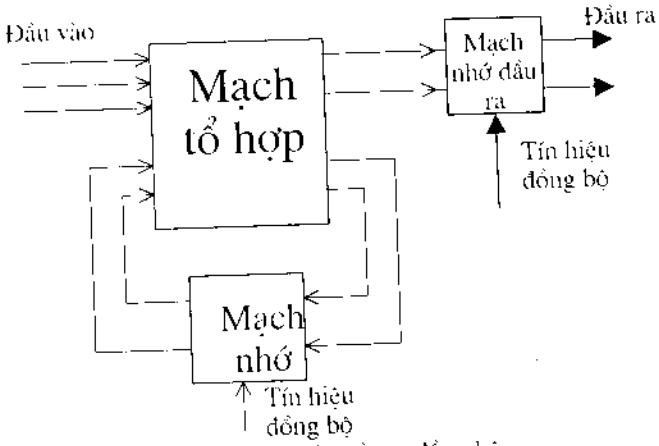

**Hình 3.36** Cấu trúc mạch tuần tự đồng bộ hình thành đầu ra không đồng bộ.

Các ôtômat có thể được biểu diễn trực quan bằng các sơ đồ chuyển trạng thái hoặc bằng các ma trận chuyển trạng thái. Các sơ đồ tương ứng với ôtômat Moore sẽ có các đỉnh là các cặp trạng thái / tín hiệu ra và các cung thể hiện quá trình chuyển trạng thái và được đánh dấu bằng tín hiệu vào tương ứng. Các sơ đồ chuyển trạng thái của ôtômat Mealey có các đỉnh là các trạng thái còn các cung thể hiện quá trình chuyển trạng thái và được đánh dấu bằng cặp giá trị tín hiệu vào / tín hiệu ra.

Các mạch tuần tự có thể được xây dựng qua các phần tử nhớ và các mạch tổ hợp theo các sơ đồ đồng bộ hoặc không đồng bộ. Trong trường hợp thiết kế mạch tuần tự theo chế độ không đồng bộ, các tín hiệu được hình thành

trên đầu ra của mạch tổ hợp có thể thay đổi và tác động vào mạch nhớ để thay đổi trạng thái của mạch. Như vậy có một khoảng thời gian chuyển tiếp của các giá trị trên đầu ra của mạch tuần tự không dự đoán được. Cấu trúc không đồng bộ của mạch tuần tự được biểu diễn trên hình 3.36.

Để khắc phục tình trạng bất định ở đầu ra của mạch tại những thời điểm chuyển trạng thái, người ta thêm vào mạch một hệ mạch nhớ nối trực tiếp với đầu ra của mạch tuần tự. Hệ mạch nhớ này có tác dụng lưu giữ giá trị đầu ra của mạch tuần tự khi mạch đang ở giai đoạn hình thành trạng thái mới. Khi mạch đã ở trạng thái ổn định mới, các giá trị đầu ra mới được tính toán. Sau khi toàn bộ quá trình chuyển trạng thái và hình thành giá trị ra mới kết thúc, dưới tác dụng của tín hiệu đồng bộ mạch nhớ sẽ ghi giá trị mới và đưa giá trị đó tới đầu ra. Khi tín hiệu đồng bộ tắt, mạch nhớ ở trạng thái lưu trữ dữ liệu và mạch sẵn sàng thực hiện các thao tác mới.




**Hình 3.37** Cấu trúc mạch tuần tự đồng bộ hình thành đầu ra đồng bộ.

## 6. Tối thiểu hoá các bảng chuyển trạng thái của các ôtômat

Trong quá trình xây dựng các mạch tuần tự, số lượng các trạng thái của các ôtômat ảnh hưởng rất lớn tới độ phức tạp của mạch. Trong nhiều trường hợp các ôtômat chứa những trạng thái dư thừa, do đó một trong những bước quan trọng trong khi thiết kế các ôtômát là tối thiểu hoá số lượng các trạng thái của ôtômat. Quá trình tối thiểu hoá trạng thái thực hiện bằng cách nhóm hai hoặc nhiều trạng thái tương đương của ôtômat. Để thực hiện điều này chúng ta sử dụng bảng chuyển trạng thái – một cách biểu diễn tương đương với sơ đồ chuyển trạng thái của ôtômat. Mỗi dòng của bảng chuyển trạng thái tương ứng với một trạng thái, mỗi cột của bảng sẽ xác định giá trị

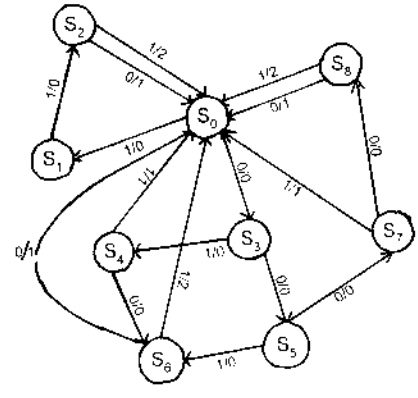
| Trạng thái | Đầu vào | |
|---|---|---|
| | 0 | 1 |
| $S_0$ | $0/S_3$ | $0/S_1$ |
| $S_1$ | $0/S_4$ | $0/S_2$ |
| $S_2$ | $1/S_0$ | $2/S_0$ |
| $S_3$ | $0/S_5$ | $0/S_4$ |
| $S_4$ | $0/S_6$ | $1/S_0$ |
| $S_5$ | $0/S_7$ | $0/S_6$ |
| $S_6$ | $1/S_0$ | $2/S_0$ |
| $S_7$ | $0/S_8$ | $1/S_0$ |
| $S_8$ | $1/S_0$ | $2/S_0$ |

**Hình 3.38** Bảng chuyển trạng thái.

đầu ra và trạng thái tiếp theo của ôtômat tương ứng với giá trị đầu vào đang xét.

Các trạng thái tương đương của ôtômat có thể được định nghĩa như sau. Hai trạng thái $S_i$ và $S_j$ được coi là tương đương nếu với một chuỗi tín hiệu vào bất kỳ tác động vào ôtômat khi ôtômat đang ở trạng thái $S_i$ hoặc $S_j$ chúng ta cùng nhận được một chuỗi tín hiệu ra như nhau.

Định nghĩa này có một ý nghĩa thực tế xác định bởi vì nếu ta quan sát ôtômat từ bên ngoài ta chỉ thu nhận được các tín hiệu ra. Điều kiện đủ của sự tương đương các trạng thái được biết đến như điều kiện ColdWell. Tương ứng với điều kiện này, hai dòng bất kỳ "đầu ra / trạng thái tiếp sau" giống nhau được coi là tương đương. Trong hình 3.39, các trạng thái $S_2$, $S_6$, $S_8$ thỏa mãn điều kiện ColdWell và do đó được coi là tương đương. Nhưng thậm chí cả khi những trạng thái tiếp sau phân biệt nhưng lại là tương đương thì hai dòng cũng được coi là tương đương. Như vậy ta phải thực hiện những bước kiểm tra tính tương đương của các trạng thái trong bảng theo những bước lặp, và sử dụng tính hội tụ của thủ tục kiểm tra.




**Hình 3.39** Sơ đồ chuyển trạng thái của ôtômat tương ứng với bảng chuyển trạng thái trên hình 3.38.

Phương pháp phân nhóm các trạng thái.

Các tiêu chuẩn ColdWell có thể được chia làm hai điều kiện.

- Điều kiện tương thích đầu ra. Điều kiện này là tiên quyết.
- Điều kiện trùng hợp của những trạng thái tiếp sau.

Quá trình phân nhóm các các dòng trong bảng chuyển trạng thái được thực hiện như sau:

- Nhóm các dòng trong bảng chuyển trạng thái theo các tín hiệu ra để nhận được các phân nhóm. Ví dụ, trong hình 3.38, các phân nhóm theo tín hiệu ra sẽ là:

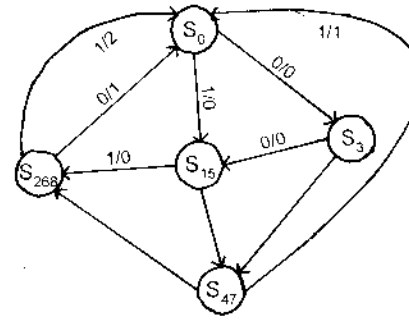
$$G_1 = \{ ( S_0, S_1, S_3, S_5 ), ( S_2, S_6, S_8 ), ( S_4, S_7 ) \}.$$

Ta thấy $G_1$ thể hiện nhóm các trạng thái tạo ra những giá trị tín hiệu ra như nhau với dãy tín hiệu vào có độ dài một. Các trạng thái $S_4$ và $S_7$ nằm trong cùng một nhóm bởi vì với đầu vào '0', $S_4$ chuyển về $S_6$ còn $S_7$ chuyển về $S_8$, nhưng $S_6$ và $S_8$ lại nằm trong cùng một nhóm tương đương; với đầu vào '1', $S_4$ và $S_7$ cùng chuyển về trạng thái $S_0$.

| Trạng thái | Đầu vào | |
|---|---|---|
| | 0 | 1 |
| $S_0$ | $0/S_3$ | $0/S_{15}$ |
| $S_{15}$ | $0/S_{47}$ | $0/S_{268}$ |
| $S_3$ | $0/S_{15}$ | $0/S_{47}$ |
| $S_{268}$ | $1/S_0$ | $2/S_0$ |
| $S_{47}$ | $0/S_{268}$ | $1/S_0$ |

**Hình 3.40** Bảng chuyển trạng thái sau khi đã tối thiểu hoá của ôtômat trên hình 3.39.

- Kiểm tra sự trùng hợp của những trạng thái ở thời điểm tiếp đối với các trạng thái ở trong từng nhóm. Các trạng thái tiếp theo phải nằm trong cùng một nhóm. Nếu các điều kiện này không thoả mãn thì nhóm sẽ được phân chia nhỏ. Trong ví dụ trước, nhóm thứ nhất của $G_1$ là ($S_0$, $S_1$, $S_3$, $S_5$) chứa các trạng thái $S_1$ và $S_5$. Các trạng thái này chuyển về những nhóm giống nhau: khi tín hiệu vào là '0', hệ thống sẽ chuyển về các trạng thái trong nhóm thứ ba: $S_4$ và $S_7$; khi tín hiệu vào là '1', hệ thống sẽ chuyển về trạng thái trong nhóm thứ hai: $S_2$ và $S_6$. Các trạng thái khác chuyển vào những nhóm khác nhau. Do đó nhóm ( $S_0$, $S_1$, $S_3$, $S_5$ ) sẽ được phân chia thành các nhóm ( $S_0$ ), ( $S_1$, $S_5$ ), ( $S_3$ ). Các nhóm còn lại của $G_1$ không phân tách được. Cuối cùng chúng ta nhận được phân nhóm tương đương như sau:




**Hình 3.41** Sơ đồ ôtômat trên hình 3.39 sau khi đã tối thiểu hoá.

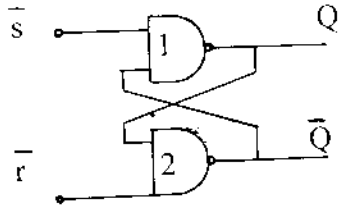
$$G_2 = \{ (S_0), (S_1, S_5), (S_3), (S_2, S_6, S_8), (S_4, S_7) \}.$$

Phân nhóm này được kiểm tra với các chuỗi đầu vào có độ dài hai. Thủ tục vừa trình bày ở trên sẽ được lặp lại cho tới khi đạt được sự hội tụ. Chúng ta không thể tiếp tục phân nhóm $G_2$, do đó kết quả nhận được có thể được coi là lời giải bài toán tối thiểu hoá.

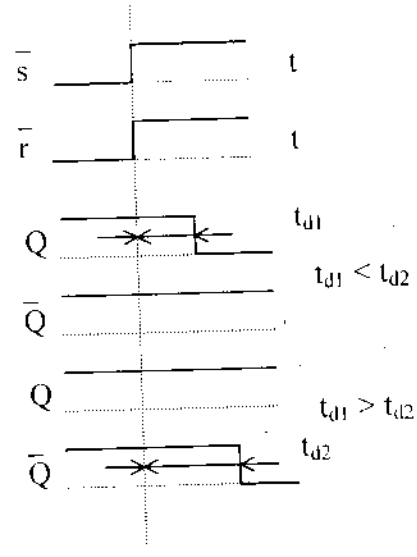
## §3.6. Những vấn đề khi thiết kế các mạch tuần tự

### 1. Hiện tượng cạnh tranh trong các mạch tuần tự

Khác với các mạch tổ hợp, trạng thái của các mạch tuần tự được xác định không chỉ từ các tín hiệu vào mà còn phụ thuộc vào các trạng thái trước đó của mạch. Việc lưu trữ các trạng thái trước đó của mạch được thực hiện trên các phần tử nhớ như thanh ghi, trigơ, ... Như vậy mạch tuần tự có thể biểu diễn bằng những ôtômat có số trạng thái hữu hạn. Việc phân tích và tổng hợp các ôtômat là một phần quan trọng trong lý thuyết ôtômat. Trong mục này chúng ta chỉ đề cập tới một số vấn đề khi thiết kế các ôtômat.




**Hình 3.42** Mạch trigơ RS.




**Hình 3.43** Ảnh hưởng của cạnh tranh trong phần tử trigơ RS.

Trong các ôtômat, do ảnh hưởng của thời gian trễ tín hiệu trên các phần tử mạch, người ta quan sát thấy những trạng thái trung gian chuyển tiếp giữa các trạng thái trong thiết kế. Những trạng thái chuyển tiếp này có ảnh hưởng lớn đến hoạt động của mạch. Sự cạnh tranh trong quá trình truyền tín hiệu

giữa các phần tử mạch có thể dẫn đến việc thiết lập những trạng thái sai của các phần tử nhớ, do đó có thể dẫn đến hoạt động sai của mạch. Khi thiết kế các ôtômat, sự cạnh tranh trong các phần tử mạch phải được loại trừ.

Chúng ta khảo sát ảnh hưởng của sự cạnh tranh giữa các phần tử mạch trong ví dụ đơn giản về hoạt động của trigơ RS. Ta nhận thấy, nếu $\overline{S} = \overline{R} = 1$, trigơ giữ nguyên trạng thái và giá trị trên hai đường tín hiệu ra Q và $\overline{Q}$ ngược nhau. Ví dụ, khi Q = 1 thì hai đầu vào của phần tử $NAND_2$ sẽ nhận giá trị '1', khi đó giá trị đầu ra $\overline{Q}$ sẽ bằng '0'. Giá trị '0' này sẽ duy trì giá trị '1' tại đầu ra Q. Khả năng nhớ của phần tử được duy trì bởi vòng phản hồi trên hai phần tử NAND.

Nếu các đầu vào $\overline{S}$ = '0' , $\overline{R}$ = '1', Giá trị các đầu ra sẽ là Q = '1', $\overline{Q}$ = '0'. Các giá trị này được duy trì ngay cả sau khi $\overline{S}$ = '1'.

Điều này cũng xảy ra tương tự khi $\overline{R}$ = '0', $\overline{S}$ = '1', các đường tín hiệu ra Q = '0', $\overline{Q}$ = '1'.

Trạng thái khi $\overline{S}$ = '0' , $\overline{R}$ = '0' là trạng thái cấm. Trạng thái này không thể sử dụng được vì khi các giá trị trên hai đầu vào $\overline{S}$ , $\overline{R}$ chuyển sang nhận các giá trị $\overline{S}$ = '1' , $\overline{R}$ = '1', kết quả sẽ phụ thuộc vào sự cạnh tranh giữa hai phần tử $NAND_1$ và $NAND_2$. Ta nhận thấy khi $\overline{S}$ = '0' , $\overline{R}$ = '0', hai đầu ra Q và $\overline{Q}$ cùng nhận giá trị Q = '1' , $\overline{Q}$ = '1'. Trạng thái này là trạng thái xác định mặc dù nó phá vỡ quan hệ $Q_{NAND2} = \overline{Q}_{NAND1}$. Nhưng khi trigơ chuyển ra khỏi trạng thái đó bằng các giá trị $\overline{S}$ = '1' , $\overline{R}$ = '1', quá trình chuyển trạng thái sẽ diễn ra như trên hình 3.34. Sau khi các giá trị '1' xuất hiện trên hai đầu vào của phần tử trigơ, trên hai đầu vào của hai phần tử $NAND_1$ và $NAND_2$ sẽ nhận đồng thời hai giá trị '1'. Như vậy hai đầu ra sẽ phải cùng nhận giá trị '0'. Nhưng điều đó không thể xảy ra, bởi vì sự xuất hiện giá trị '0' ở một đầu ra sẽ làm cho đầu ra còn lại nhận giá trị '1' và trạng thái này sẽ được duy trì. Như vậy một trong hai phần tử NAND sẽ thắng trong cuộc đua và trạng thái sẽ xác định theo phần tử NAND đó. Ví dụ, nếu thời gian trễ của phần tử $NAND_1$ lớn hơn thời gian trễ của phần tử $NAND_2$, $t_{d1} > t_{d2}$, tín hiệu '0' sẽ xuất hiện ở đầu ra của phần tử $NAND_2$ trước và trạng thái sẽ được thiết lập sẽ là Q = '1' , $\overline{Q}$ = '0'. Trong trường hợp, nếu $t_{d1} < t_{d2}$, thì các giá trị ra sẽ bằng Q = '0' , $\overline{Q}$ = '1'. Mặt khác giá trị thời gian trễ là một đại lượng ngẫu nhiên do đó có thể kết luận rằng trong trường hợp này giá trị đầu ra không xác định. Điều này làm cho cặp giá trị $\overline{S}$ = '0', $\overline{R}$ = '0' là các giá trị cấm.

Trong nhiều trường hợp cơ chế cạnh tranh trong các mạch tuần tự xảy ra như sau. Các trạng thái của ôtômat, do các tín hiệu thiết lập thay đổi lại phụ thuộc vào trạng thái của các phần tử nhớ. Do hiện tượng trễ tín hiệu trong các phần tử lôgic, sự thiết lập các giá trị của ô nhớ không xảy ra đồng thời. Điều đó có thể ảnh hưởng tới tín hiệu thiết lập của các phần tử khác và do đó có thể dẫn đến những trạng thái không lường trước.

Phương pháp quan trọng để ngăn chặn hiện tượng cạnh tranh giữa các phần tử trong mạch là sử dụng các chế độ đồng bộ. Trong các chế độ này, việc nhận các giá trị tín hiệu vào các ô nhớ và đưa các giá trị đó tới đầu ra được thực hiện vào những thời điểm xác định sau khi các quá trình quá độ trong mạch kết thúc.

Trên quan điểm về tổ chức các quá trình thời gian các mạch số có thể chia thành các mạch đồng bộ và các mạch không đồng bộ.

Trong những thiết bị không đồng bộ, quá trình chuyển mạch của các phần tử xảy ra không theo sự điều khiển từ bên ngoài. Tốc độ thực hiện của các phép toán trong các thiết bị đó phụ thuộc vào thời gian trễ của các phần tử và là những đại lượng ngẫu nhiên. Do thời gian truyền tín hiệu trong các đoạn mạch không được biết trước, điều đó làm cho vấn đề giải quyết cạnh tranh giữa các phần tử mạch trở nên khá phức tạp. Những phương pháp giải quyết cạnh tranh hiện có khá phức tạp và không được ứng dụng rộng rãi trong kỹ thuật. Trên thực tế chỉ có một số phần nhỏ của mạch được thực hiện theo chế độ không đồng bộ.

Trong những thiết bị đồng bộ, quá trình xử lý thông tin được sắp xếp theo thời gian bởi những tín hiệu đồng hồ do những thiết bị phát xung đồng hồ tạo ra.Việc nhận dữ liệu vào các phần tử nhớ được thực hiện tại những thời điểm thời gian xác định, sau khi các tín hiệu tại đầu vào các phần tử nhớ đạt được giá trị xác lập. Những giá trị xác lập này do cơ chế hoạt động của mạch xác định. Trong những mạch đồng bộ, tín hiệu đồng hồ được tạo ra sau những khoảng thời gian cố định với độ dài đủ lớn để cho các quá trình quá độ trong các đoạn mạch kết thúc ngay cả trong trường hợp thời gian trễ tín hiệu cực đại. Trên thực tế các quá trình quá độ trong phần lớn các phần tử mạch kết thúc sớm hơn dự tính, do đó trong toàn bộ độ dài của tín hiệu đồng bộ, các phần tử mạch ở trạng thái "nghỉ". Điều này là kết quả của quá trình chờ đợi khoảng thời gian đồng bộ kết thúc tính cho trường hợp xấu nhất khi thời gian trễ trong mạch cực đại. Mức độ không sử dụng hết tốc độ của phần tử mạch phụ thuộc vào sự phân tán tham số thời gian trễ trên các phần tử và

được xác định bởi tỷ số giữa giá trị thời gian trễ cực đại và giá trị trung bình của chúng.

## 2. Thiết kế những mạch tuần tự đồng bộ

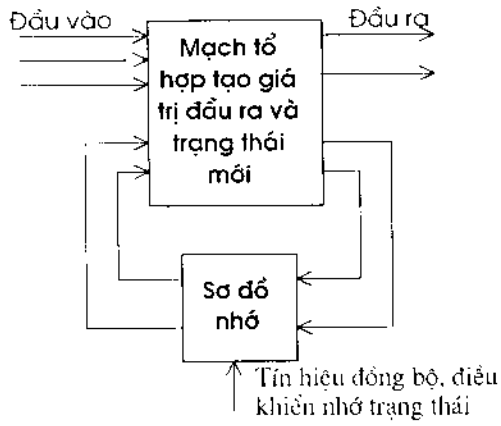
| | Các giá trị | Mã hoá nhị phân |
|---|---|---|
| Giá trị đầu vào | 0 | 0 |
| | 1 | 1 |
| Giá trị đầu ra | 0 | 00 |
| | 1 | 01 |
| | 2 | 11 |
| Các trạng thái | $S_0$ | 000 |
| | $S_3$ | 001 |
| | $S_{15}$ | 010 |
| | $S_{47}$ | 011 |
| | $S_{268}$ | 100 |

**Hình 3.44** Mã hoá nhị phân các giá trị vào, giá trị ra và các trạng thái.

Để có thể xây dựng các ôtômat từ các sơ đồ chuyển trạng thái hoặc bảng chuyển trạng thái, ta cần phải mã hoá nhị phân cho các giá trị đầu vào, đầu ra và các trạng thái. Một số nhị phân n bit có thể biểu diễn $2^n$ giá trị đầu vào, đầu ra và trạng thái. Đối với ôtômat đã tối thiểu hoá trong mục trước (hình 3.40, 3.41) các giá trị đầu vào, đầu ra và trạng thái sẽ được mã hoá như sau.

Ở đây chúng ta chú ý rằng, khi mã hoá các giá trị đầu ra, đầu vào và đặc biệt là các trạng thái, ta cần chọn những tổ hợp mã sao cho khoảng cách mã giữa hai trạng thái chuyển tiếp là ít nhất. Nói cách khác, ta cần phải mã hóa các trạng thái $S_i$ và $S_j$ sao cho khi chuyển từ trạng thái $S_i$ vào trạng thái $S_j$ theo sơ đồ chuyển trạng thái, số lượng bit cần phải chuyển trạng thái là ít nhất.

Chúng ta coi rằng sự chuyển trạng thái xác định với mọi đầu vào. Trên thực tế, trong các mạch tuần tự, có thể xảy ra các trường hợp khi giá trị của đầu ra và trạng thái tiếp theo không xác định với mọi giá trị đầu vào. Như vậy, trong mạch sẽ xuất hiện tình trạng không xác định. Trong những trường hợp này, cũng giống như các mạch tổ hợp, chúng ta có thể đơn giản hoá chức năng hoạt động của mạch, nhưng khi đó ta không thể sử dụng phương pháp tối thiểu hoá trạng thái theo các tiêu chuẩn ColdWell như trong mục §3.5.6. Người ta thường sử dụng các phương pháp dựa trên kinh nghiệm.




**Hình 3.45** Cấu trúc mạch tuần tự hình thành đầu ra không đồng bộ.

Khi ta mã hoá các trạng thái, với các chế độ đồng bộ thích hợp, chúng ta sẽ nhận được mạch tuần tự với sơ đồ dạng chính tắc như trên hình 3.45.

Các mạch nhớ có thể được thực hiện dưới dạng các sơ đồ dạng thanh ghi sử dụng những thiết kế mạch khác nhau với những chế độ đồng bộ thích hợp. Trên hình vẽ 3.45, các mạch nhớ có chức năng lưu trữ các trạng thái của mạch tuần tự. Mạch tổ hợp thực hiện chức năng các hàm lôgic thiết lập giá trị trạng thái mới và các giá trị đầu ra theo giá trị các đầu

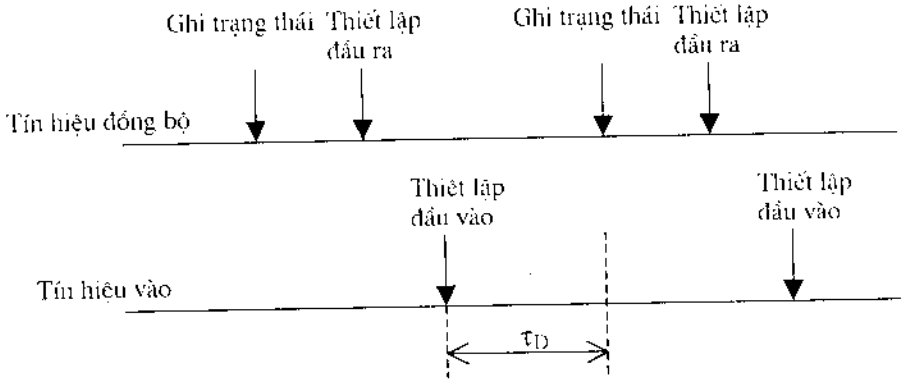
| Giá trị đầu vào | Trạng thái | Giá trị đầu ra | Giá trị ra đối với trạng thái tiếp theo |
|---|---|---|---|
| 0 | 000 | 00 | 00x |
| 1 | 000 | 00 | 0x0 |
| 0 | 010 | 00 | 0xx |
| 1 | 010 | 00 | x00 |
| 0 | 001 | 00 | 0x0 |
| 1 | 001 | 00 | 0xx |
| 0 | 100 | 0x | 000 |
| 1 | 100 | xx | 000 |
| 0 | 011 | 00 | x00 |
| 1 | 011 | 0x | 000 |

**Hình 3.46** Hàm lôgic nhiều đầu ra tương ứng với bảng chuyển trạng thái trên hình 3.40 và bảng mã hoá trạng thái trên hình 3.45.

vào và các trạng thái hiện thời ghi trong mạch nhớ của ôtômat. Mạch tổ hợp này có thể được xây dựng trên các phần tử lôgic cơ bản hoặc trên những PLA, ROM và dễ dàng được xây dựng từ những bảng chuyển trạng thái và các sơ đồ mã hoá trạng thái và tín hiệu vào / ra. Ví dụ, bảng giá trị trên hình

3.46 thể hiện hàm lôgic có nhiều đầu ra tương ứng với bảng chuyển trạng thái 3.40 và bảng mã hoá trên hình 3.44. Bảng này nhận được bằng cách chuyển sang hệ biểu diễn nhị phân các trạng thái, đầu vào và đầu ra. Trong bảng này, ký hiệu x tương ứng với giá trị đơn vị.

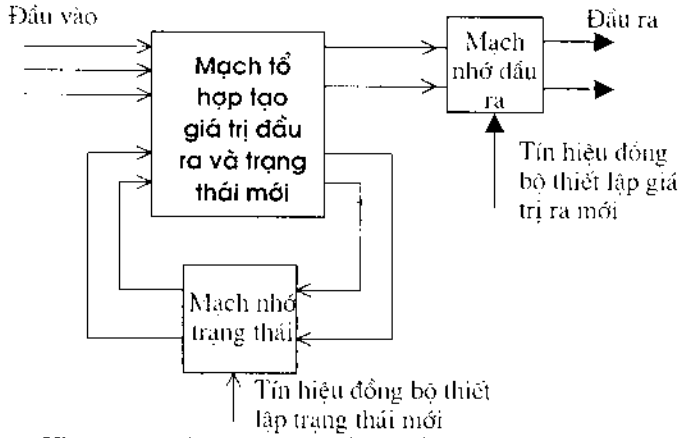
Khi thực hiện mạch tuần tự dưới dạng mạch đồng bộ theo sơ đồ trên hình 3.45, chúng ta cần phải quan tâm đặc biệt đến quan hệ thời gian giữa các tín hiệu đồng bộ và các tín hiệu đầu vào. Quan hệ này được mô tả trên hình 3.47. Vai trò của tín hiệu đồng bộ, không phụ thuộc vào dạng đồng bộ, là thực hiện chức năng ghi nhớ các giá trị trạng thái và đồng bộ hoá việc thiết lập các giá trị dữ liệu ra dựa vào trạng thái được nhớ. Các dữ liệu đưa vào bộ nhớ cần phải được thiết lập trước thời điểm thực hiện lưu trữ. Theo sơ đồ




**Hình 3.47** Quan hệ thời gian giữa tín hiệu đồng bộ và tín hiệu đầu vào.

thiết kế dạng đầu ra không đồng bộ (hình 3.45), các giá trị tín hiệu đặt vào hệ mạch nhớ là kết quả của các phép tính toán trong mạch tổ hợp đối với các giá trị đầu ra của hệ mạch nhớ (chính là các trạng thái) và các giá trị đầu vào hệ thống. Từ đó suy ra các giá trị đầu vào cần phải được thiết lập trước thời điểm được đưa vào hệ mạch nhớ để lưu trữ một khoảng thời gian lớn hơn hoặc bằng $\tau_D$ – giá trị thời gian trễ của mạch tổ hợp. Do đó chu kỳ của tín hiệu đồng bộ phải lớn hơn hoặc bằng $\tau_D$. Thêm vào đó, trong sơ đồ mạch tuần tự với đầu ra không đồng bộ, tồn tại thời gian trễ giữa thời điểm thiết lập các giá trị đầu vào và các giá trị đầu ra của hệ mạch tổ hợp. Do đó khi thiết kế các đoạn mạch mắc nối tiếp như trên hình 3.45 (đầu ra của một đoạn mạch ghép nối nối tiếp với đầu vào của đoạn mạch tiếp theo), thời gian thiết lập giá trị đầu vào sẽ càng trễ khi tín hiệu chuyển tiếp từ lớp mạch này sang lớp mạch tiếp theo.

Trong mục trước chúng ta đã thấy rằng giữa các phần tử trong mạch có hiện tượng chạy đua trong quá trình truyền tín hiệu. Nguyên nhân của hiện tượng này là tham số thời gian trễ tín hiệu trên các phần tử của mạch là những đại lượng ngẫu nhiên. Hiện tượng này ảnh hưởng rất lớn đến hoạt động của mạch. Do hiện tượng chạy đua này trong mạch có thể xuất hiện những trạng thái không được tính đến từ trước và điều đó có thể dẫn tới những hành vi không lường trước được của mạch.

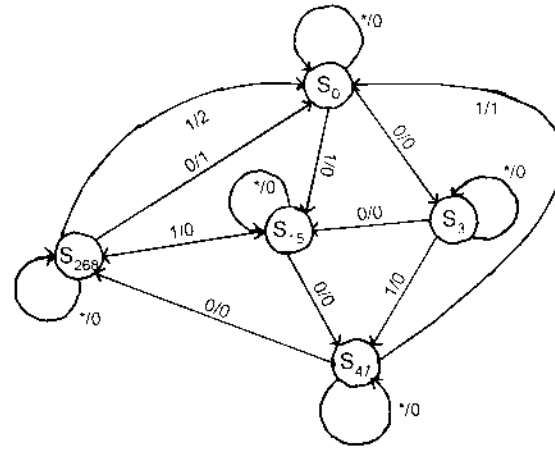



**Hình 3.48** Cấu trúc mạch tuần tự đồng bộ hình thành đầu ra đồng bộ.

Để loại trừ những thuộc tính ngẫu nhiên trong hành vi của mạch, những ôtômat hữu hạn thường được thiết kế theo chế độ đồng bộ việc thiết lập giá trị đầu ra. Trên đầu ra của mạch trong hình 3.45, chúng ta thêm vào một hệ mạch nhớ. Hệ mạch nhớ này có tác dụng lưu trữ giá trị của tín hiệu đầu ra. Khi trong mạch tổ hợp và hệ mạch nhớ trạng thái đang diễn ra quá trình thiết lập trạng thái mới thì quá trình chuyển tiếp này không ảnh hưởng tới đầu ra. Khi quá trình thiết lập trạng thái kết thúc, hệ mạch tổ hợp sẽ xác định giá trị đầu ra mới. Khi toàn bộ các quá trình hình thành giá trị này kết thuc, tín hiệu đồng bộ của hệ nhớ giá trị đầu ra sẽ điều khiển việc nhập giá trị đầu ra mới vào hệ mạch nhớ. Như vậy việc hình thành giá trị đầu ra sẽ chậm hơn việc hình thành trạng thái một chu kỳ đồng bộ, nhưng khi đó quan hệ thời gian theo như hình 3.47 có thể được thoả mãn không phụ thuộc vào số lượng các tầng trong đoạn mạch nối tiếp. Ta nhận thấy rằng sơ đồ ôtômat như hình 3.48 có thể được coi là sơ đồ ôtômat Moore do thông tin chứa trong hệ mạch nhớ giá trị đầu ra có thể coi như chỉ phụ thuộc hoàn toàn vào trạng thái.

Một vấn đề quan trọng khi thiết kế các ôtômat dưới dạng những mạch đồng bộ là quan hệ giữa thời điểm tác động của tín hiệu đầu vào của ôtômat và thời điểm tác động của tín hiệu đồng bộ. Đối với ôtômat biểu diễn bằng

sơ đồ chuyển trạng thái trong hình 3.39, một cách đơn giản, chúng ta có thể tổ chức quá trình đồng bộ sao cho mỗi khi xuất hiện tín hiệu đầu vào, ta thực hiện việc chuyển trạng thái sang trạng thái mới. Do đó nếu thực hiện mạch theo chế độ không đồng bộ đầu ra (hình 3.45), khi xuất hiện tín hiệu đầu vào tại đầu vào của ôtômat, chúng ta phải phát ra một tín hiệu đồng bộ. Nếu khoảng thời gian giữa những lần xuất hiện tín hiệu vào không như nhau, khi đó chu kỳ đồng bộ sẽ không đều. Trong nhiều trường hợp điều này sẽ làm việc thiết kế chế độ đồng bộ phức tạp lên nhiều. Các sơ đồ đồng bộ theo dạng này gọi là các sơ đồ đồng bộ xung. Trong những mạch đồng bộ xung, người ta sử dụng những xung đồng bộ có chu kỳ xác định, nếu tín hiệu đầu vào xuất hiện trong những khoảng thời gian biến thiên, khi đó chúng ta sẽ đưa vào những giá trị tín hiệu vào mới. Những giá trị tín hiệu này gọi là "giá trị không có tín hiệu" và ký hiệu là giá trị '*'. Nếu tính đến những trường hợp này, ôtômat trên hình 3.41 sẽ chuyển thành ôtômat trên hình 3.49. Nếu đầu vào của ôtômat nhận giá trị '*', khi đó tại tất cả các trạng thái của mạch thêm chu tình chờ. Chu trình này giữ nguyên trạng thái của mạch tại thời điểm hiện thời. Điều này không làm tăng số các trạng thái của ôtômat, nhưng số lượng đầu vào tăng lên một đường. Như vậy các giá trị tín hiệu vào tương ứng với ví dụ hình 3.41 sẽ được mã hoá bằng hai bit.




**Hình 3.49** Sơ đồ ôtômat trên hình 3.40 sau khi đã được đồng bộ hoá và thêm giá trị vào không xác định '*'.

Các sơ đồ mã hoá trạng thái của ôtômat bằng các biểu diễn nhị phân có thể ảnh hưởng tới giá thành thực hiện phần mạch tổ hợp. Việc lựa chọn những bộ mã tối ưu cho ta giá thành tối thiểu, nhưng các thuật toán mã hoá tối ưu phụ thuộc vào từng trường hợp cụ thể và hiện nay chưa có những thuật toán tìm mã tối ưu một cách tự động. Tuy vậy, nếu chú ý tới quá trình tối

thiểu hoá các hàm lôgic, ta sẽ nhận thấy rằng, những trạng thái gần nhau trong sơ đồ chuyển trạng thái (các trạng thái liên kết với một trong những trạng thái trước và một trong những trạng thái tiếp sau) nên được mã hoá bằng những tổ hợp mã nhị phân gần nhau. Tiêu chuẩn gần nhau ở đây có thể dựa trên khái niệm khoảng cách Hamming.

Như vậy trong chương này chúng ta đã nghiên cứu các phương pháp thiết kế mạch số bao gồm các vấn đề liên quan tới thiết kế các mạch tổ hợp và thiết kế các mạch tuần tự. Do sự khác biệt đặc thù về chức năng và hoạt động nên các mạch tổ hợp và các mạch dãy được xây dựng với những đặc điểm khác biệt. Những vấn đề quan trọng khi thiết kế các mạch số là lựa chọn các chế độ đồng bộ và xử lý các cạnh tranh trong mạch.

**Bài tập cho chương 3**

1. Hãy thiết kế thiết bị điều khiển cho mạch cộng hai số nhị phân dấu phẩy tĩnh 8 bit.

2. Hãy thiết kế thiết bị xử lý dữ liệu cho mạch cộng hai số nhị phân dấu phẩy tĩnh 8 bit.

3. Hãy thiết kế mạch thực hiện phép cộng hai số nhị phân dấu phẩy tĩnh 8 bit.

4. Hãy thiết kế thiết bị điều khiển cho mạch cộng hai số trong mã BCD với 4 hàng chữ số thập phân.

5. Hãy thiết kế thiết bị xử lý dữ liệu cho mạch cộng hai số trong mã BCD với 4 hàng chữ số thập phân.

6. Hãy thiết kế mạch thực hiện phép cộng hai số trong mã BCD với 4 hàng chữ số thập phân.

# CHƯƠNG IV. NHỮNG KHÁI NIỆM CHUNG VỀ MÔ HÌNH HOÁ PHẦN CỨNG

## §4.1. Mô hình hoá phần cứng

Mô hình phần cứng là những biểu diễn phần cứng trên các mức độ trừu tượng khác nhau. Mô hình cho ta thấy những phần tử liên quan mà không chỉ rõ những chi tiết của chúng. Trong quá trình thiết kế, mô hình được sử dụng để đặc trưng cho mạch, thể hiện các tính chất của mạch và là phương tiện để trao đổi thông tin về thiết kế giữa những người thiết kế hoặc giữa người thiết kế và máy tính. Các đặc tả mạch là các mô hình mô tả chi tiết mạch sẽ được xây dựng. Các đặc tả này luôn đi đôi với các ràng buộc về mặt thiết kế như hiệu năng, diện tích.

Các mô hình mạch bao giờ cũng được định nghĩa bằng các qui tắc cú pháp, ngữ nghĩa và được mô tả trong những ngữ cảnh sử dụng khác nhau. Cũng như các biểu thức toán học, cú pháp của mạch bao gồm các mô tả những thành phần cơ sở của mạch và mô tả mối liên kết giữa các thành phần đó trong mạch. Ngữ nghĩa của mạch là các mô tả về hoạt động của mạch, hành vi, chức năng của mạch. Ngữ cảnh của mạch là mô tả về hoạt động của mạch khi được ghép nối với những mạch khác, những nguồn tín hiệu tác động từ bên ngoài khác nhau. Các mô tả đó cung cấp một cách toàn diện thông tin về mạch để loại trừ mọi trường hợp mơ hồ trong thiết kế.

Mạch có thể được mô hình hóa theo những cách khác nhau tương ứng với các mức độ trừu tượng (mức kiến trúc, mức lôgic, mức hình học), theo các góc độ quan sát (góc độ hành vi, góc độ cấu trúc và góc độ vật lý) và tương ứng với các phương pháp mô hình hóa được sử dụng trong quá trình thiết kế (các ngôn ngữ mô tả thiết kế, các sơ đồ mạch hoặc các mô hình toán học).

Hiện nay trong quá trình thiết kế mạch, để biểu diễn các mô hình mạch, người ta thường sử dụng rộng rãi các ngôn ngữ mô hình hoá phần cứng ( HDL). Việc mô tả mạch bằng các ngôn ngữ mô hình hoá phần cứng cũng giống như việc viết chương trình bằng các ngôn ngữ lập trình phần mềm. Tính súc tích và dễ hiểu của các ngôn ngữ HDL giúp cho việc mô tả mạch bằng các ngôn ngữ đó được ưa chuộng hơn việc biểu diễn bằng các biểu đồ,

sơ đồ trạng thái, sơ đồ lôgic, mặc dù trong nhiều trường hợp một số sơ đồ cho phép ta thấy một cách trực quan các chức năng của mạch điện.

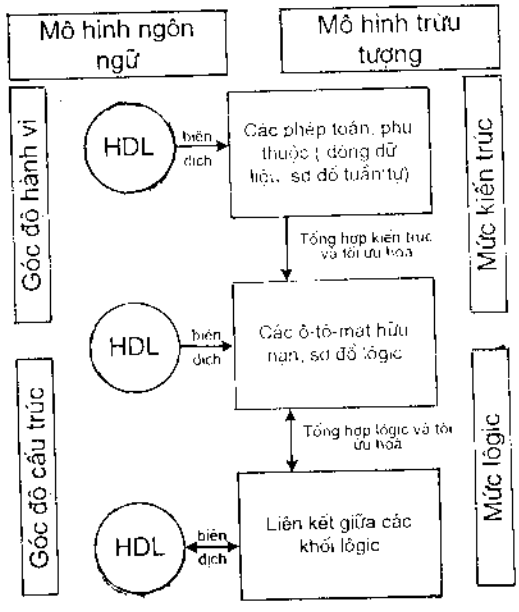
Trong chương này chúng tôi sẽ đề cập tới các mô hình trừu tượng của mạch. Các mô hình này sẽ được nghiên cứu ở mức kiến trúc và mức lôgic. Các mô hình trừu tượng là các mô hình toán học dựa trên cơ sở đồ hình và đại số Bool.

Ở mức kiến trúc, hành vi của mạch điện sẽ được trừu tượng hóa thông qua tập hợp các phép toán và mối quan hệ phụ thuộc giữa chúng. Các phép toán có thể là các phép toán số học, đại số, lôgic. Hành vi của các mạch tuần tự có thể được mô tả thông qua các ôtômat có trạng thái hữu hạn, hoặc các hàm lôgic trong trường hợp các mạch tổ hợp. Góc độ cấu trúc của mô hình sẽ được biểu diễn thông qua các liên kết giữa các khối lôgic, các mạch đóng ngắt (nếu xét trên mức lôgic) hoặc các tài nguyên (nếu xét trên mức kiến trúc).

Các mô hình trừu tượng đủ mạnh để có thể nắm được các đặc điểm của mạch thông qua các ngôn ngữ mô hình hoá HDL và sơ đồ. Đồng thời các mô hình đó cũng khá đơn giản để có thể kiểm chứng các hoạt động của mạch.

## §4.2. Các ngôn ngữ mô hình hóa phần cứng

Trong lĩnh vực thiết kế mạch, các ngôn ngữ mô hình hoá phần cứng (HDL) xuất hiện do nhu cầu phải có công cụ mô tả chính xác mạch về cấu trúc cũng như hành vi. Một số ngôn ngữ mô tả phần cứng được thiết kế với những đặc điểm và những mục đích khác nhau, thậm chí một số ngôn ngữ mô tả phần cứng được phát triển từ các ngôn ngữ lập trình, nhưng giữa các ngôn ngữ lập trình và ngôn ngữ mô hình hoá phần cứng có những điểm khác biệt quan trọng. Những khác nhau này là do sự khác biệt giữa những đối tượng mà các ngôn ngữ HDL và ngôn ngữ lập trình mô tả.




**Hình 4.1** Các mô hình mạch, quá trình tổng hợp và tối ưu mạch.

Sau đây chúng ta sẽ xem xét những đặc điểm đó.

- Các mạch phần cứng có thể thực hiện những phép toán có mức độ song song (đồng thời) lớn còn các phần mềm thì ngược lại, trên những máy đơn xử lý chỉ thực hiện được những phép toán tuần tự. Về mặt này, các ngôn ngữ mô tả phần cứng sẽ gần giống với các ngôn ngữ lập trình cho các máy tính xử lý song song.
- Các mô hình phần cứng luôn phải chứa những thông tin về cấu trúc. Ví dụ, sự tiếp xúc của mạch với các mạch khác làm nảy sinh yêu cầu phải mô tả các cổng vào ra của mạch và khuôn dạng dữ liệu được trao đổi qua những cổng đó. Do đó các ngôn ngữ mô hình hoá phần cứng cần phải hỗ trợ việc mô tả thiết kế cả trên góc độ hành vi và cấu trúc để có thể biểu diễn các đặc trưng của mạch một cách có hiệu quả.
- Việc xác định thời gian và thời điểm thực hiện của các phép toán hết sức quan trọng trong phần cứng do các tương tác giữa các thành phần phần cứng với nhau. Trong khi đó vấn đề tương tác theo thời gian ít ảnh hưởng tới việc thực hiện các phép toán trong các chương trình phần mềm, trừ một số trường hợp trong những ứng dụng thời gian thực.

Mạch điện có thể được mô tả dưới những góc độ quan sát khác nhau, do đó các ngôn ngữ HDL với những đặc trưng tương ứng cũng được phát triển. Trên thực tế, khi nghiên cứu các mô hình ở mức kiến trúc và lôgic của quá trình mô hình hóa, người ta sử dụng các mô tả mạch theo các góc độ hành vi và góc độ cấu trúc được sử dụng. Một số ngôn ngữ còn giúp ta có một cách nhìn tổng hợp, điều này cho phép nhà thiết kế có thể đặc tả các chi tiết của các phần mạch cũng như từng bước xây dựng các mô hình cấu trúc từ các mô hình hành vi. Các ngôn ngữ mô hình hoá phần cứng còn phục vụ cho mục đích truyền tải và trao đổi các khuôn dạng của thiết kế giữa nhà thiết kế và công cụ.

Các ngôn ngữ HDL không chỉ được phát triển do yêu cầu của bài toán mô hình hoá phần cứng. Để có thể phê chuẩn được thiết kế, các mô hình phần cứng phải được kiểm nghiệm qua mô phỏng hoặc bằng các phương pháp kiểm tra. Các phương pháp tổng hợp mạch sử dụng các mô hình trên các ngôn ngữ HDL làm điểm xuất phát. Các mô hình này phải đáp ứng được từng mục đích cụ thể của quá trình thiết kế mạch. Ví dụ như ngôn ngữ mô hình hoá phần cứng VHDL được phát triển với mục đích để mô tả những

mạch có độ tích hợp siêu lớn, ngôn ngữ Verilog được thiết kế để nâng cao hiệu quả của quá trình mô phỏng mạch.

Việc đáp ứng nhiều mục tiêu như vậy của các ngôn ngữ HDL không thể đạt được nếu ta chỉ sửa đổi các ngôn ngữ lập trình phần mềm để thoả mãn các nhu cầu của quá trình mô tả phần cứng. Các ngôn ngữ lập trình có thể dùng để mô phỏng hoạt động của một số thiết bị tính toán, nhưng không thể dùng trong quá trình xây dựng thiết kế.

## 1. Những đặc điểm khác biệt của các ngôn ngữ mô tả phần cứng

Các ngôn ngữ được đặc trưng nhờ các quy tắc cú pháp, ngữ nghĩa và thực tế sử dụng. Cú pháp liên quan tới các cấu trúc của ngôn ngữ và có thể được thể hiện qua các quy tắc ngữ pháp. Ngữ nghĩa chỉ ra ý nghĩa của các thành phần ngôn ngữ. Các quy tắc ngữ nghĩa tác động tương ứng tới những thành phần ngôn ngữ thoả mãn các quy tắc cú pháp. Thực tế sử dụng ngôn ngữ liên quan tới những khía cạnh khác của ngôn ngữ, bao gồm cả vấn đề sử dụng và thực hiện ngôn ngữ.

Về đại thể có thể chia ngôn ngữ làm hai loại: ngôn ngữ thủ tục (procedural) và ngôn ngữ mô tả khai báo (declarative).

- Trong các ngôn ngữ thủ tục, các chương trình thể hiện các tác động mong muốn bằng cách mô tả dãy các bước cần thiết để thực hiện các tác động đó.
- Đối với ngôn ngữ khai báo, các mô hình thể hiện các vấn đề sẽ được giải quyết bằng tập hợp các đặc tả, khai báo mà không đưa ra chi tiết các phương pháp giải quyết . Do đó trình tự mô tả các khối cơ sở không quan trọng trong những ngôn ngữ khai báo.

Các ngôn ngữ mô hình hoá phần cứng được phân loại dựa trên cơ sở góc độ quan sát các đối tượng được mô tả. Ví dụ, những ngôn ngữ mô tả thiết kế ở mức vật lý sẽ được hỗ trợ nhờ các đặc tả những đối tượng hình học nguyên thuỷ, các thao tác trên các đối tượng đó. Chúng ta chỉ tập trung vào nghiên cứu biểu diễn mô hình mạch tương ứng với các góc độ quan sát hành vi và cấu trúc. Theo khía cạnh đó ta cũng nghiên cứu những ngôn ngữ thích hợp tương ứng. Phần lớn các ngôn ngữ HDL đều hỗ trợ cả hai dạng mô tả mô hình theo các góc độ hành vi và cấu trúc.

Các ngôn ngữ HDL thường được phát triển kèm theo các bộ mô phỏng. Tốc độ thực hiện là một trong những yêu cầu đối với bộ mô phỏng. Các

thuật toán mô phỏng theo sự kiện được sử dụng rộng rãi bởi vì chúng cho phép bộ mô phỏng giảm thiểu số lượng các tính toán và do đó làm giảm thời gian thực hiện mô phỏng.

Quá trình mô phỏng bao gồm việc tính các giá trị của tín hiệu trong một khoảng thời gian xác định. Khoảng thời gian này được chia thành các khung thời gian. Trong mỗi khung thời gian, chu trình mô phỏng bao gồm các bước sau.

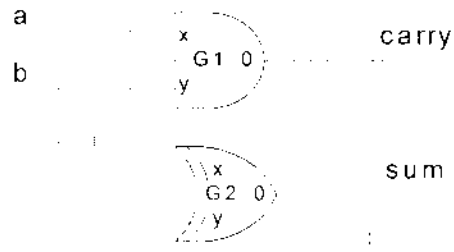
- Tín hiệu được lan truyền trong mạch và được cập nhật.
- Tất cả các quá trình nhạy cảm với các sự kiện được tính toán cho tới khi chúng được dừng lại.
- Khi tất cả các quá trình tính toán đều dừng lại, thời gian trong bộ mô phỏng được chuyển sang khung tiếp theo và hình thành chu trình mô phỏng mới.

Phương pháp mô hình hoá mạch nói trên đủ tổng quát để cho phép mô tả các mạch đồng bộ và không đồng bộ. Trong các mạch đồng bộ tất cả các quá trình có thể được kích hoạt tại mỗi chu kỳ đồng hồ. Nếu thời gian trễ của các thành phần mạch đã biết, chúng có thể được đặc tả như một thuộc tính của tín hiệu và bộ mô phỏng có thể mô tả chính xác hành vi theo thời gian của mạch. Trong trường hợp các thời gian trễ là chưa biết, người thiết kế hoặc phải tìm những thông tin về thời gian trễ từ các mô nhình hành vi, hoặc đưa ra các giả thiết và ràng buộc về thời gian trễ để thực hiện các phép toán trong khung thời gian xác định.

## 2. Các ngôn ngữ mô tả cấu trúc phần cứng

Mô hình được mô tả bằng các ngôn ngữ cấu trúc thể hiện các kết nối giữa các phần tử. Do đó các ngôn ngữ này có sức mạnh biểu cảm tương tự như các sơ đồ mạch mặc dù những đặc điểm của ngôn ngữ cho phép cung cấp những mô tả khái quát hơn. Các hệ thống thứ bậc trong ngôn ngữ cho phép tạo các mô hình có tính môđun hoá và ngắn gọn. Các thành phần cơ sở của ngôn ngữ cấu trúc cho phép xếp các ngôn ngữ này vào nhóm các ngôn ngữ khai báo ( declarative ), mặc dù một số ngôn ngữ mô tả cấu trúc có chứa những thành phần thủ tục. Các biến trong ngôn ngữ tương ứng với các cổng của các phần tử.

Ta hãy xét ví dụ mô tả mạch nửa tổng bằng ngôn ngữ VHDL.

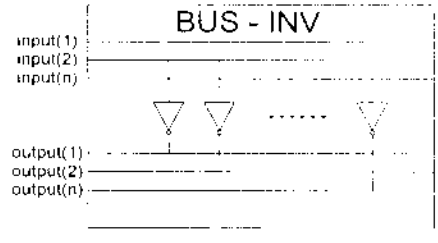



**Hình 4.2** Cấu trúc của bộ nửa tổng.

```
Architechture STRUCTURE of Half_Adder is
        Component AND2
                Port ( x, y: in bit; 0: out );
        End component
        Component EXOR2
                Port ( x, y: in bit ; 0: out );
        End component
Begin
        G1: AND2
                Port map ( a, b, carry );
        G2: EXOR2
                Port map ( a, b, sum );
End STRUCTURE;
```

Trong mô hình này chứa hai khai báo của mô hình khác là AND2 và EXOR2 và hai khởi tạo mô hình là G1 và G2. Thông tin cụ thể về các mô hình AND2 và EXOR2 được khai báo ở một vị trí khác, ví dụ như trong các thư viện chuẩn.

Dạng khác của biến là các siêu biến. Các biến này được dùng để làm mô hình mạch gọn hơn. Ví dụ về các




**Hình 4.3** Biểu diễn cấu trúc của mảng các bộ đảo tín hiệu.

biến loại này có thể là các chỉ số của mảng. Các loại biến này không biểu diễn trực tiếp các thành phần của phần cứng và được loại trừ khỏi mô hình sau bước dịch đầu tiên.

Ví dụ về các siêu biến: Ta xây dựng mô hình của mảng 32 bộ đảo tín hiệu nối giữa hai tuyến tín hiệu bằng ngôn ngữ VHDL. Từ khóa *generate* cho ta nhiều phiên bản của biến vòng lặp *i*:

```
Architecture STRUCTURE of BUS_INV is
Component INVERTER
        Port ( i1 : in bit ; 01 : out bit );
End component
Begin
  G: for i in 1 to 32 generate
  INV: INVERTER port map ( inputs( i ), output( i ));
  End generate;
End STRUCTURE;
```

### 3. Ngôn ngữ mô tả chức năng phần cứng

Các mạch tổ hợp có thể được mô tả bằng tập hợp các phần tử lôgic và tập hợp các phương trình. Các cấu trúc này liên kết các biến thành các biểu thức lôgic. Phương thức khai báo ứng dụng tốt nhất cho trường hợp mô tả các mạch tổ hợp - những mạch được mô tả không cần bộ nhớ. Thật vậy, các mạch tổ hợp có thể coi là các ghép nối (về mặt cấu trúc) của các toán tử, trong đó mỗi toán tử xác định một hàm lôgic. Các mô hình này khác với các mô hình cấu trúc ở chỗ không có tương quan "một - một" giữa các biểu thức và các cổng lôgic, bởi vì đối với một số biểu thức, sẽ không tồn tại phần tử lôgic thực hiện biểu thức đó.

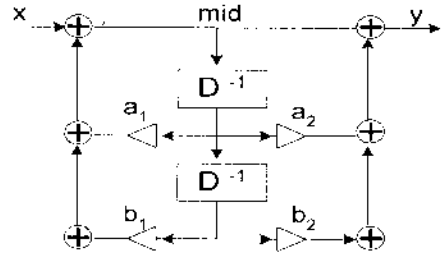
Các ngôn ngữ thủ tục có thể sử dụng để mô tả các mạch tổ hợp. Phần lớn các ngôn ngữ cấu trúc cho phép thực hiện phép gán nhiều lần với một biến. Để tránh sự mập mờ về giá trị biến, trong các ngôn ngữ mô tả chức năng có các cơ chế giải quyết mập mờ, ví dụ như những phép toán sau sẽ xoá bỏ tác động của phép toán trước.

Ta xét ví dụ như sau: mô tả bộ nửa tổng trên VHDL dùng mô hình hành vi:

```
Architecture BEHAVIOR of HALF_ADDER is
        Process
        Begin
                Carry ⇐ ( a and b );
                Sum ⇐ ( a xor b );
        End process;
End BEHAVIOR;
```

Trong ví dụ này, bên trong khối giới hạn bởi cặp từ khoá **Process** và **End** Process các biểu thức được thực hiện một cách tuần tự. Hai phép gán tương ứng với hai cấu trúc trong mô hình bộ nửa tổng.

Trong trường hợp mô hình hóa các mạch đồng bộ, phong cách mô hình hóa chịu ảnh hưởng nặng nề bởi cơ chế điều hoà thời gian của ngôn ngữ. Những phương pháp mô hình hóa các mạch tổ hợp có thể được mở rộng cho các mạch tuần tự. Các mạch tuần tự có thể được mô tả dựa vào tập hợp các phép gán của các biểu thức. Các đối số trong các biểu thức là các biến cùng với thời gian trễ đồng bộ. Mô hình này thường được sử dụng cho các bộ xử lý tín hiệu số hoặc các đường truyền dữ liệu đồng bộ.



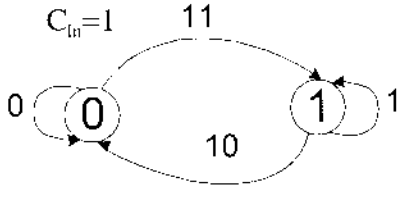

**Hình 4.4** Mạch lọc số truy hồi.

Ta xét ví dụ về mạch lọc số truy hồi. Các biến chứa giá trị thời gian trễ được ký hiệu bởi @ và đi theo sau là giá trị thời gian trễ. Khi đó mạch lọc có thể biểu diễn như sau:

```
Function IIR (a1, a2, b1, b2, x : num)
/* Giá trị trả lại */      y: num =
Begin
        y = mid + a2* mid@1 +b2 *mid@2;
        mid = x + a1*mid@1 +b1 *mid@2;
end.
```

Mạch tuần tự cũng thường được mô hình hóa bằng các ngôn ngữ thủ tục. Các ôtômat có trạng thái hữu hạn biểu diễn hoạt động của mạch có thể được mô tả bằng những mô hình thủ tục trong đó các biến lưu giữ các thông tin trạng thái. Khi đó các thao tác của ôtômat hữu hạn được mô tả bằng những bước lặp đồng bộ theo xung nhịp đồng hồ, với những phân nhánh chuyển trạng thái tương ứng với trạng thái hiện thời.

Đây là ví dụ mô tả ôtômat hữu hạn thực hiện việc nhận biết các bit '1' liên tiếp ở dòng dữ liệu vào. Mô tả được viết trên ngôn ngữ VHDL.




**Hình 4.5** Đồ hình chuyển trạng thái của ôtômat hữu hạn nhận biết dãy các bít '1' liên tiếp.

```
architecture BEHAVIOR of ReC is
        type STATE_TYPE is ( STATE_ZERO, STATE_ONE);
        signal STATE : STATE_TYPE := STATE_ZERO;
        process
                begin
                        wait until clock 'event' and clock = '1';
                        if ( in  = '1' ) then
                                case STATE is
                                when ⇒ STATE_ZERO
                                        STATE ⇐ STATE_ONE;
                                        out ⇐ '0';
                                when ⇒ STATE_ONE
                                        STATE ⇐ STATE_ONE;
                                        out ⇐ '1';
                                end case
                        else
                                STATE ⇐ STATE_ZERO;
                                out ⇐ '0';
                        end if
                end process
end BEHAVIOR.
```

Ta xét trường hợp mô hình hóa hành vi ở mức kiến trúc bằng những ngôn ngữ thủ tục. Phương thức chung là biểu diễn các phép gán tuần tự cho các biến bằng những cấu trúc điều khiển luồng thông tin như rẽ nhánh, vòng

lập, gọi các mô hình ( trong ngữ cảnh của các ngôn ngữ HDL, các mô hình tương đương với những chương trình con trong các ngôn ngữ lập trình phần mềm ). Điều này cho ta thấy các hành vi của mạch có thể được mô hình hóa như là tập hợp các phép toán và quan hệ. Các phép toán sẽ tương ứng với phép gán, nhóm các gán hoặc các phép gọi mô hình. Sự phụ thuộc của các dòng dữ liệu và dòng điều khiển sẽ xác định những ràng buộc về trình tự của các phép toán và mức độ đồng thời thực hiện chúng trong biểu diễn mô hình. Ví dụ như khi ta mô tả tập hợp các lệnh máy bằng ngôn ngữ VHDL, ta có thể nhận được đoạn chương trình sau.

```
ir ⇐ fetch ( pc );
case ir is
        when ⇒ AND
                acc ⇐ rega and regb;
        when ⇒ OR
                acc ⇐ rega and regb;
        when ⇒ XOR
                acc ⇐ rega and regb;
        when ⇒ ADD
                acc ⇐ rega and regb;
end case;
pc ⇐ pc +1;
```

Đoạn chương trình này mô tả ba bước thực hiện.

- Gọi hàm *fetch*, hàm này trả lại giá trị của thanh ghi *ir*.
- Các lựa chọn giá trị cho *acc* tương ứng với các phép toán được ghi trong thanh ghi lệnh *ir* với các toán hạng nằm trong thanh ghi *rega* và *regb*.
- Tăng bộ đếm chương trình *pc* lên 1.

Phân tích các quá trình thực hiện lệnh ta thấy dòng lệnh phép toán cuối cùng có thể thực hiện song song với hai phép trên.

Các mô hình hành cho phép biểu diễn các khung thời gian thực hiện phép toán với một mức độ tự do nhất định. Việc kiểm soát trình tự thực hiện và hành vi của mạch theo thời gian cũng có thể được mô hình hành vi cung

cấp. Ta xét mô hình ôtômat hữu hạn đã nêu ở ví dụ trước trước. Việc kiểm soát về thời gian được thực hiện dựa vào các toán tử đồng bộ dưới dạng biểu thức của câu lệnh *wait*. Do mô hình chỉ có một toán tử *wait* nên tất cả các phép toán sẽ thực hiện trong một vòng lặp thời gian. Việc điều khiển về thời gian cho phép ta kiểm soát trình tự thực hiện các phép toán của mô hình. Trong ví dụ trước, hai phép toán đầu phải thực hiện tuần tự, còn phép toán thứ ba có thể song song với một trong hai phép toán trên. Do đó ta cần thêm các câu lệnh *wait* để kiểm soát trình tự thực hiện chúng:

```
wait until clock' event and clock = '1';
ir ⇐ fetch (pc);
wait until clock' event and clock = '1'
case ir is
        when ⇒ ADD
                acc ⇐ rega + regb;
        when ⇒ AND
                acc ⇐ rega & regb;
end case
pc ⇐ pc +1;
```

Trong trường hợp này, mô hình tác động vào bộ mô phỏng và bộ mô phỏng sẽ làm trễ quá trình thứ hai và thứ ba.

## §4.3. Các mô hình trừu tượng

Trong mục này, chúng ta mô tả các mô hình trừu tượng dùng để biểu diễn mạch trên các mức độ phân tích và theo các góc độ quan sát khác nhau trên mức lôgic và mức kiến trúc. Các mô hình này thường dựa trên cách biểu diễn bằng các đồ hình.

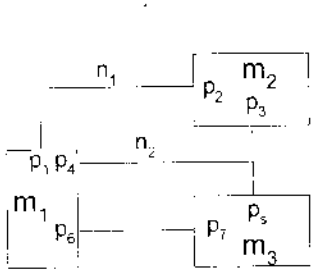
### 1. Các cấu trúc

Cấu trúc của mạch có thể mô hình hóa dựa vào các cấu trúc liên kết. Các cấu trúc liên kết bao gồm tập hợp các môđun, tập hợp các mạng kết nối và

quan hệ liên kết giữa các môđun và mạng kết nối. Mô hình cấu trúc có thể được biểu diễn bằng nhiều cách.

- Có thể biểu diễn mô hình cấu trúc một cách đơn giản bằng các siêu đồ thị, trong đó mỗi đỉnh của đồ thị tương ứng với các môđun và cung tương ứng với mạng liên kết. Quan hệ liên kết giữa các môđun và mạng được mô tả bằng các ma trận liên kết. Ta có siêu đồ thị tương đương với một đồ thị hai phần có tập hợp đỉnh chia làm hai phần, trong đó một phần tương đương với các môđun, phần còn lại tương đương với các mạng.
- Một cách biểu diễn khác của cấu trúc là biểu diễn mỗi môđun bằng các điểm đầu cuối hay là cổng và mô tả sự kết nối giữa các mạng với các cổng của môđun.

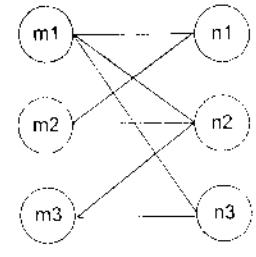
Thông thường các ma trận liên kết rất tản mạn, khi đó sử dụng danh sách mạng có hiệu quả hơn để mô tả cấu trúc. Trong danh sách mạng ta đánh số tất cả các mạng nối với từng môđun ( danh sách loại này gọi là danh sách mạng hướng môđun ) hoặc đánh số tất cả các môđun kết nối với một mạng ( danh sách mạng hướng mạng ). Ví dụ, trên hình 4.6 ta biểu diễn cấu trúc của một đối tượng gồm ba khối cơ sở. Các cấu trúc kết nối được biểu diễn bằng ma trận liên kết và đồ thị:



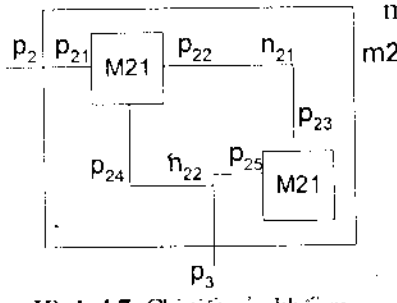

Ma trận liên kết &
danh sách liên kết:

$m_1$ 1 1 1
$m_2$ 1 1 0
$m_3$ 0 1 1
$n_1$ $n_2$ $n_3$

m1: n1, n2, n3
m2: n1, n2
m3: n2, n3




**Hình 4.6** Biểu diễn mô hình cấu trúc.




**Hình 4.7** Chi tiết của khối $m_2$.

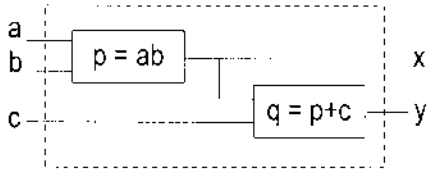
Các cấu trúc liên kết có thể tạo thành trật tự như sau: những môđun nằm tại các nút lá là các đối tượng cơ sở cùng tập hợp các cổng. Các môđun không nằm tại nút lá là tập hợp các môđun con. Tập hợp các mạng và các cấu trúc liên kết sẽ gắn kết các

mạng với các cổng của các môđun hoặc với các cổng của các môđun con. Trong ví dụ trước, các môđun có thể có chứa các môđun con như khối $m_2$. Khối $m_2$ chứa các khối $m_{21}$ và $m_{22}$ và các cổng $p_{21}$, $p_{22}$, $p_{23}$, $p_{24}$, $p_{25}$.

## 2. Mạng lôgic

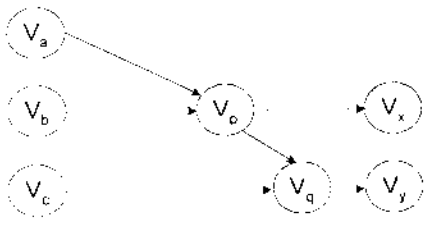
Mạng lôgic tổng quát là một cấu trúc trong đó những môđun tại lá tương ứng với các hàm lôgic tuần tự hoặc tổ hợp. Khái niệm này là khá tổng quát và mạnh. Ta giới hạn hai trường hợp đối với mạng cấu trúc: mạng lôgic tổ hợp và mạng lôgic đồng bộ.

– Mạng lôgic tổ hợp hay còn gọi là mạng Bool là một cấu trúc phân cấp, trong đó:
  - Mỗi môđun tại lá tương ứng với một hàm lôgic có nhiều đầu vào và một đầu ra. Hàm này gọi là hàm cục bộ.
  - Các cổng vào ra được chia làm hai nhóm: các cổng vào và các cổng ra. Các cổng không thuộc các môđun con cũng được chia làm hai nhóm: các đầu vào và các đầu ra sơ cấp.
  - Mỗi mạng có một cổng tách biệt gọi là cổng nguồn và có định hướng từ cổng nguồn tới các cổng khác. Các cổng nguồn của mạng có thể hoặc là đầu vào sơ cấp hoặc là đầu ra sơ cấp của môđun thuộc mức thấp hơn. Trên thực tế, cổng nguồn còn có thể là các đầu ra của những hàm cục bộ.
  - Quan hệ giữa các mạng trong môđun là quan hệ được sắp một phần.



**Hình 4.8** Sơ đồ mạng lôgic.

Một mạng lôgic có thể được biểu diễn bằng các đồ thị. Để đơn giản, ta xem xét trường hợp mạng không phân cấp. Đồ thị mạng lôgic $\mathbf{G_n(V,E)}$ là đồ thị có hướng trong đó tập hợp các đỉnh **V** chứa các phần tử là tương ứng "một-một" với các đầu vào sơ cấp, các hàm cục bộ ( là các môđun ) và các đầu ra sơ cấp. Tập hợp các cung có hướng **E** biểu diễn sự phân tách mạng nhiều cực thành các mạng hai cực. Theo định nghĩa, đồ thị nhận được sẽ không chứa mạch vòng bởi vì mạng là được sắp




**Hình 4.9** Sơ đồ mạng lôgic.

một phần đối với các môđun. Hình 4.9 thể hiện đồ thị mạng lôgic của cấu trúc ở hình 4.8, trong đó:

- $v_a$,$v_b$,$v_c$ là ba đỉnh vào;
- $v_x$,$v_y$: hai đỉnh ra;
- $v_p$,$v_q$: hai đỉnh trung gian tương ứng với các hàm lôgic.

Mạng lôgic là biểu diễn hỗn hợp theo cấu trúc và hành vi, vì các cơ chế liên kết cung cấp các cấu trúc, còn các hàm lôgic đưa ra hành vi tại các cổng của các môđun nằm tại nút lá. Trong một số trường hợp, các hàm lôgic không thể thực hiện được như vậy vì các giới hạn kỹ thuật. Ví dụ hàm AND 64 đầu vào có thể biểu diễn bằng các hàm cục bộ của mạng lôgic. Nhưng trong kỹ thuật hàm AND đó có thể được phân tách thành tập hợp các phần tử AND hai cổng và do đó có cấu trúc khác với ban đầu.

Trong những trường hợp mà mạng lôgic dùng để mô tả các liên kết giữa các cổng lôgic, ta có biểu diễn cấu trúc thuần tuý. Phần lớn, mạng lôgic sử dụng để biểu diễn các hàm lôgic có nhiều đầu vào-đầu ra bằng phương pháp cấu trúc. Thật vậy, các mạng lôgic có tương quan "một – một" với hàm vào / ra lôgic tổ hợp. Chúng ta có thể nhận được hàm này bằng cách kết hợp các hàm cục bộ với nhau để xác định các đầu ra sơ cấp theo các đầu vào sơ cấp. Thông thường các hàm vào/ra không thể dễ dàng biểu diễn dưới dạng chuẩn tắc, ví dụ như dạng chuẩn tắc tuyển hoặc hội, hoặc dưới dạng các sơ đồ quyết định nhị phân do kích thước và độ phức tạp của hàm. Đó là một trong các nguyên nhân sử dụng mô hình mạng lôgic. Mô hình này cũng không phải là biểu diễn duy nhất của các mạch tổ hợp.

Các mạng lôgic có đồng bộ là trường hợp tổng quát hóa của các mạng lôgic tổ hợp. Mạng lôgic loại này dùng để mô tả các mạch tuần tự làm việc theo chế độ đồng bộ. Trong mô hình này các môđun tại lá có thể dùng để mô tả các hàm lôgic tổ hợp có nhiều đầu vào với một đầu ra hoặc các phần tử trễ đồng bộ. Mạng không nhất thiết phải được sắp từng phần tương ứng với các môđun. Tuy nhiên, một mạng con của mạng đang xét, trong đó các phần tử nguồn không phải là phần tử trễ đồng bộ, có thể được sắp từng phần để mô hình hoá những trường hợp đặc biệt. Những trường hợp đặc biệt này yêu cầu trong những mạch đồng bộ không chứa những vòng phản hồi qua các mạch tổ hợp.

## 3. Các ôtômat và sơ đồ trạng thái

Trên phương diện hành vi, ở mức lôgic, các mạch dãy có thể được biểu diễn bằng các ôtômat hữu hạn. Các ôtômat hữu hạn được định nghĩa là một bộ năm $\mathbf{A} = < \mathbf{X}, \mathbf{Y}, \mathbf{S}, \delta, \lambda >$, trong đó:

- **X** là tập hợp các ký tự vào của ôtômat.
- **Y** là tập hợp các ký tự ra của ôtômat.
- **S** là tập hợp các trạng thái của ôtômat.
- Tập hợp các hàm chuyển trạng thái : $\delta : \mathbf{X} \times \mathbf{S} \rightarrow \mathbf{S}$
- Tập hợp các hàm ra : $\lambda : \mathbf{X} \times \mathbf{S} \rightarrow \mathbf{Y}$ : ôtômat Mealy
  $\lambda : \mathbf{S} \rightarrow \mathbf{Y}$ : ôtômat Moore.
- Và $s_0$ là trạng thái khởi động của ôtômat.

Bảng chuyển trạng thái của ôtômat chứa các trạng thái chuyển tiếp và những hàm ra. Tương ứng với các bảng chuyển trạng thái, khi biểu diễn ôtômat bằng đồ thị ta có sơ đồ chuyển trạng thái.

Sơ đồ chuyển trạng thái là một đồ thị có hướng $\mathbf{G_t(V,E)}$, trong đó:

- Tập hợp các đỉnh của đồ thị **V** tương ứng "một - một" với tập hợp các trạng thái **S** của ôtômat.
- Tập hợp các cung có hướng **E** tương ứng "một – một" với sự chuyển tiếp trạng thái. Những sự chuyển tiếp trạng thái này được đặc trưng bởi hàm chuyển tiếp δ. Đặc biệt, cung $( v_i, v_j )$ tồn tại nếu tồn tại ký tự vào $x \in \mathbf{X}$ sao cho
  $$\delta(x, s_i) = s_j , \forall\ i,j = 1, 2, ..., |\ S\ |.$$

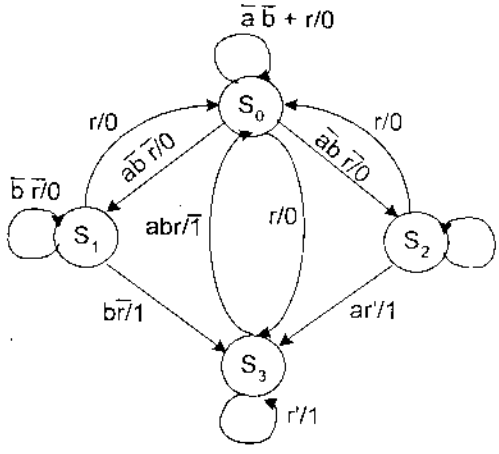
Trong mô hình ôtômat Mealy mỗi cung được gắn nhãn bởi cặp ký tự $x/\lambda(x, s_i)$. Còn trong mô hình ôtômat Moore mỗi cung được gắn nhãn bởi $x$, và mỗi đỉnh $v_i \in \mathbf{S}$ được dán nhãn tương ứng với hàm ra $\lambda(s_i)$.

Ta hãy xét ví dụ ôtômat Mealy thực hiện việc đồng bộ giữa hai tín hiệu. Các đầu vào sơ cấp là *a* và *b*, tín hiệu khởi tạo *r*. Ôtômat có một đầu ra sơ cấp *o*. Đầu ra *o* nhận giá trị '1' nếu hai tín hiệu *a* và *b* cùng đồng thời nhận giá trị *true* hoặc trong trường hợp một tín hiệu nhận giá trị *true* còn tín hiệu kia nhận giá trị *true* trong thời điểm trước đó. Ôtômat có bốn trạng thái sau:

- Trạng thái khởi động $s_0$.
- Trạng thái nhớ $s_1$ khi a là *true* và b là *false*.

– Trạng thái nhớ $s_2$ khi b là *true* và a là *false*.
– Trạng thái nhớ $s_3$ khi a và b cùng *true*.

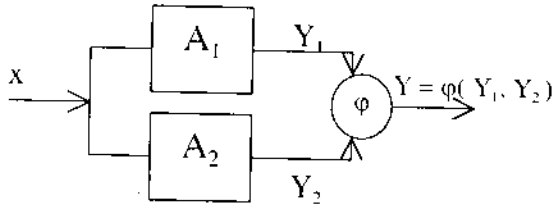
Trong một số trường hợp, để tiện lợi khi biểu diễn các sơ đồ ôtômat hữu hạn người ta phân tách ôtômat thành các sơ đồ con. Mỗi sơ đồ con, ngoại trừ sơ đồ gốc, có các trạng thái vào và trạng thái ra. Các trạng thái này tương ứng với một hoặc nhiều đỉnh của sơ đồ ở những mức cao hơn. Các đỉnh ở mức cao hơn này gọi là đỉnh tham chiếu. Mỗi sự chuyển trạng thái tới đỉnh tham chiếu tương ứng với sự chuyển trạng thái tới trạng thái vào của sơ đồ ôtômat hữu hạn tương ứng tại mức thấp hơn. Sự chuyển trạng thái tới các trạng thái ra tương ứng với việc quay về trạng thái tham chiếu.




**Hình 4.10** Sơ đồ chuyển trạng thái của ôtômat Mealey thực hiện việc đồng bộ giữa hai tín hiệu.

Các sơ đồ phân cấp thường được dùng trong quá trình tổng hợp thiết kế để ghép nối các ôtômat hữu hạn theo con đường môđun hoá. Các phương pháp này thường dùng để mô tả những mạch có kích thước lớn.

Trong kỹ thuật mô hình hoá, người ta thường sử dụng các phương pháp xây dựng các ôtômat phức tạp từ những ôtômat đơn giản. Các phương pháp này dựa trên việc ghép nối các ôtômat theo phương pháp ghép nối nối tiếp, ghép nối song song hoặc hỗn hợp của cả hai cách ghép nối trên.




**Hình 4.11** Ghép nối song song hai ôtômat.

- Ghép nối song song các ôtômat.

Hai ôtômat $A_1$ và $A_2$ được gọi là ghép nối song song thành một ôtômat A nếu hai đầu vào của hai ôtômat $A_1$ và $A_2$ được nối chung với đầu vào

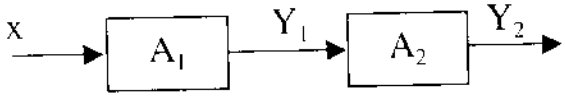
của ôtômat A. Điều kiện đặt ra ở đây là hai tập hợp tín hiệu đầu vào của hai ôtômat $A_1$ và $A_2$ phải hoàn toàn giống nhau. Các đầu ra của $A_1$ và $A_2$ được nối vào bộ tổng hợp tín hiệu. Bộ tổng hợp này thực hiện hàm φ lên hai đầu vào của hai ôtômat ban đầu và hình thành đầu ra của ôtômat A. Sơ đồ ghép nối song song được đưa ra trên hình 4.11. Các thông số của ôtômat A được xác định theo các thông số của các ôtômat $A_1$ và $A_2$ như sau:

$$\mathbf{A_1} = < \mathbf{X}, \mathbf{Y_1}, \mathbf{S_1}, \delta_1, \lambda_1 >$$
$$\mathbf{A_2} = < \mathbf{X}, \mathbf{Y_2}, \mathbf{S_2}, \delta_2, \lambda_2 >$$
$$\mathbf{A} = < \mathbf{X}, \mathbf{Y}, \mathbf{S}, \delta, \lambda >$$

trong đó

$$\mathbf{S} = \mathbf{S_1} \times \mathbf{S_2} = \{ (s_1, s_2) | s_1 \in \mathbf{S_1}, s_2 \in \mathbf{S_2} \};$$
$$\mathbf{X} = \{x_1, x_2, ..., x_n\};$$
$$\mathbf{Y} = \varphi(\mathbf{Y_1} \times \mathbf{Y_2});$$
$$\delta(\mathbf{x}, \mathbf{s}) = (\delta_1(\mathbf{x}, \mathbf{s_1}), \delta_2(\mathbf{x}, \mathbf{s_2}));$$
$$\lambda(\mathbf{x}, \mathbf{s}) = \varphi(\lambda_1(\mathbf{x}, \mathbf{s_1}), \lambda_2(\mathbf{x}, \mathbf{s_2})).$$

- Ghép nối nối tiếp hai ôtômat.




**Hình 4.12** Ghép nối nối tiếp hai ôtômat.

Hai ôtômat $A_1$ và $A_2$ được gọi là ghép nối nối tiếp nếu các tín hiệu đầu ra của ôtômat $A_1$ là các tín hiệu đầu vào của ôtômat $A_2$. Ở đây chúng ta phải giả thiết rằng khi ghép nối nối tiếp hai ôtômat, các tính chất ban đầu của các ôtômat không bị thay đổi. Hai ôtômat được ghép nối nối tiếp sẽ tương đương với một ôtômat $A = < \mathbf{X}, \mathbf{Y}, \mathbf{S}, \delta, \lambda >$, trong đó,

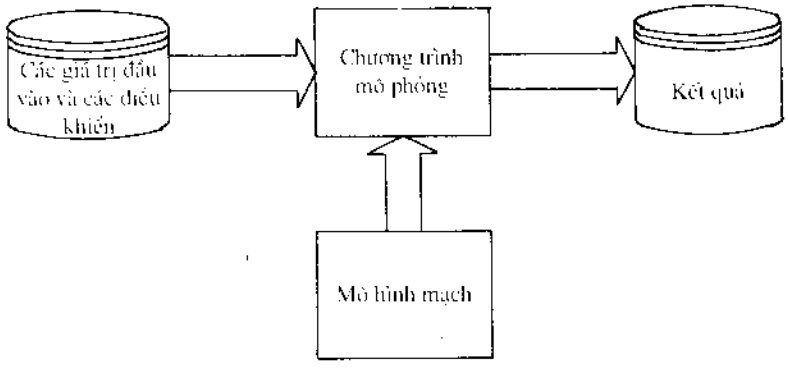
$$S = S_1 \times S_2 = \{s = (s_1, s_2) | s_1 \in \mathbf{S_1}, s_2 \in \mathbf{S_2}\}$$
$$X = X_1.$$
$$Y = Y_2.$$
$$\delta(\mathbf{x}, \mathbf{s}) = (\delta_1(\mathbf{x}, \mathbf{s_1}), \delta_2(\lambda_1(\mathbf{x}, \mathbf{s_1}), \mathbf{s_2}));$$
$$\lambda(\mathbf{x}, \mathbf{s}) = \lambda_2(\lambda_1(\mathbf{x}, \mathbf{s_1}), \mathbf{s_2}).$$

Tóm lại, trong chương này chúng ta nghiên cứu một số khái niệm cơ bản trong bài toán mô hình hoá mạch. Trong công nghiệp thiết kế các vi mạch, mạch điện được thiết kế trên các mô hình phần cứng và sau đó được cụ thể

hoá bằng những ngôn ngữ mô tả phần cứng. Các ngôn ngữ này có đặc điểm khác với các ngôn ngữ lập trình truyền thống ở các khía cạnh mô tả cấu trúc và hành vi mạch theo thời gian. Trong chương tiếp sau, chúng ta sẽ nghiên cứu các vấn đề liên quan tới bài toán mô hình hoá lôgic.

# CHƯƠNG V. CÁC PHƯƠNG PHÁP MÔ HÌNH HÓA LÔGIC

Mô hình hoá lôgic là hình thức kiểm tra thiết kế sử dụng các mô hình của mạch đã được thiết kế. Quá trình mô hình hoá lôgic và mô phỏng thiết kế có thể được biểu diễn theo sơ đồ trên hình 5.1. Chương trình mô phỏng sẽ




**Hình 5.1** Sơ đồ biểu diễn quá trình mô hình hoá lôgic và mô phỏng.

biểu diễn các tín hiệu vào và tín hiệu điều khiển, phát triển quá trình tính toán trên các tín hiệu theo thời gian và hình thành các giá trị đầu ra dựa trên mô hình của mạch.

Việc kiểm chứng thiết kế lôgic là quá trình kiểm tra thiết kế mạch trên phương diện hoạt động về chức năng và theo thời gian. Quá trình này được thực hiện dựa trên sự so sánh các kết quả nhận được qua quá trình mô phỏng với những giá trị được tính toán từ trước dựa vào chức năng. Thêm vào đó, mô hình hoá lôgic còn có thể sử dụng để kiểm chứng các tính chất sau của hoạt động của mạch được thiết kế:

- Sự độc lập của các trạng thái ban đầu;
- Sự nhạy cảm của các biến (tín hiệu) vào tham số thời gian trễ của các phần tử;
- Trong hoạt động của mạch không tồn tại sự chạy đua giữa các phần tử, sự dao động, các điều kiện đầu vào không thích hợp hoặc các trạng thái treo.

Thông thường nhà thiết kế xây dựng những phiên bản mẫu của mạch theo thiết kế và kiểm tra hoạt động của mẫu. Việc kiểm tra này sẽ cho phép tìm ra những lỗi tiềm ẩn trong thiết kế. Ưu điểm của việc tạo mẫu là chúng cho phép nhà thiết kế kiểm nghiệm thiết kế theo tốc độ tính toán thực tế. Nhưng việc tạo mẫu có một nhược điểm là giá thành xây dựng phiên bản mẫu thử nghiệm tốn thời gian và có giá thành cao. Mô hình hoá lôgic và mô phỏng thay thế việc xây dựng mẫu thử bằng các phần mềm. Điều này cho phép nhà thiết kế phân tích, kiểm nghiệm và hiệu chỉnh mô hình một cách dễ dàng. So với quá trình kiểm nghiệm trên mẫu, việc kiểm nghiệm thiết kế bằng mô hình có những ưu điểm sau:

- Cho phép kiểm tra các điều kiện sinh ra lỗi (ví dụ như các mâu thuẫn trên đường tín hiệu);
- Cho phép thay đổi tham số thời gian trễ của các phần tử trong mô hình để kiểm tra những trường hợp xấu nhất về điều phối thời gian trong mạch;
- Kiểm tra những giá trị do nhà thiết kế xác định trong quá trình mô phỏng;
- Cho phép mạch được mô phỏng bắt đầu hoạt động tại bất kỳ trạng một thái nào;
- Cho phép kiểm soát một cách chính xác việc điều phối thời gian đối với những sự kiện không đồng bộ;
- Có khả năng tự động kiểm tra hoạt động của mạch được thiết kế trong môi trường liên kết với những mạch khác.

Mặc dù các mô hình mô phỏng chạy chậm hơn các mẫu phần cứng nhưng việc kiểm nghiệm dùng mô phỏng cho phép nhà thiết kế dừng quá trình mô phỏng tại những thời điểm xác định và hiển thị các giá trị tín hiệu kể cả tại những đường tín hiệu không thể quan sát trực tiếp trong phần cứng. Do đó việc sử dụng các mô hình mô phỏng trong quá trình thiết kế mạch được thực hiện rộng rãi. Trong chương này chúng ta sẽ nghiên cứu một số phương pháp mô hình hoá lôgic và xây dựng mô hình mô phỏng trong công nghệ thiết kế mạch với độ tích hợp cao. Các phương pháp này tạo cơ sở cho việc xây dựng và hoạt động của các ngôn ngữ mô hình hoá phần cứng HDL.

## §5.1. Cơ sở mô hình hóa lôgic

### 1. Các phương pháp mô hình hóa và các hệ mô phỏng

Trong kỹ thuật thiết kế các mạch lôgic, người ta phân ra hai phương pháp chính để mô hình hoá mạch. Các phương pháp này được xây dựng dựa trên cơ sở các mô hình nội tại mà hệ mô phỏng sẽ xử lý. Các hệ chương trình mô phỏng thực hiện các mô hình lôgic được dịch từ các ngôn ngữ mô hình hoá phần cứng và được gọi là hệ mô phỏng bằng biên dịch. Các mã biên dịch được tạo ra từ những mô hình trên mức thanh ghi, từ các mô hình chức năng hoặc mô hình cấu trúc. Các hệ mô phỏng biểu diễn các mô hình dựa trên các cấu trúc dữ liệu được gọi là các hệ mô phỏng bằng cách lập bảng. Các cấu trúc dữ liệu được xây dựng từ các mô hình trên mức thanh ghi truyền đạt hoặc từ các mô hình cấu trúc. Việc thể hiện hoạt động của mô hình được kiểm soát bằng cách đặt những tác động vào mạch và gọi những chương trình con thực hiện những chức năng của các toán tử cơ sở ( đối với mô hình trên mức thanh ghi truyền đạt ) hoặc chức năng của các phần tử cơ sở ( đối với mô hình cấu trúc ).

Giả sử chúng ta khảo sát mạch điện khi mạch hoạt động và quan sát những tín hiệu thay đổi giá trị tại những thời điểm thời gian bất kỳ. Các tín hiệu này được gọi là những tín hiệu kích hoạt. Tỷ lệ giữa số lượng các tín hiệu kích hoạt và tổng số các tín hiệu trong mạch gọi là hoạt tính của mạch. Trung bình hoạt tính của mạch thường nằm trong khoảng từ 1% đến 5%. Điều này là cơ sở của phương pháp mô phỏng theo hoạt tính của mạch. Theo phương pháp này, hệ thống chỉ mô phỏng những phần hoạt động của mạch.

Trong mạch điện, sự thay đổi giá trị của tín hiệu trên một đường truyền tín hiệu được gọi là một sự kiện. Như vậy mỗi khi có một sự kiện xuất hiện trên đường tín hiệu $i$, chúng ta nói rằng phần tử mạch nhận đường tín hiệu $i$ làm đầu vào được kích hoạt. Quá trình tính toán các giá trị đầu ra của phần tử được gọi là quá trình xác định giá trị tín hiệu. Phương pháp mô phỏng theo hoạt tính của mạch chỉ xác định giá trị tín hiệu đối với những phần tử được kích hoạt. Những phần tử được kích hoạt sẽ thay đổi các giá trị tín hiệu trên đầu ra của chúng và tạo ra các sự kiện mới. Như vậy, hoạt tính của mạch được xác định bởi các sự kiện trên các đường tín hiệu, do đó phương pháp

mô phỏng theo hoạt tính còn được gọi là phương pháp mô phỏng hướng sự kiện. Để có thể truyền các sự kiện theo các đường liên kết trong mạch giữa các phần tử, hệ thống mô phỏng hướng sự kiện cần phải biết mô hình cấu trúc của mạch. Do đó mô hình hoá lôgic và mô phỏng hướng sự kiện thường dựa trên cách lập bảng.

Phương pháp mô hình hoá lôgic và mô phỏng bằng biên dịch phần lớn chỉ quan tâm tới việc kiểm chứng chức năng hoạt động của mạch mà không quan tâm tới việc điều khiển và điều phối các quá trình tính toán theo thời gian của mạch. Do đó, phương pháp mô hình hoá lôgic và mô phỏng thích hợp với những mạch đồng bộ, trong đó, việc điều phối các tiến trình tính toán theo thời gian có thể được kiểm tra tách rời với việc kiểm tra chức năng của mạch. Ngược lại, phương pháp mô hình hoá lôgic và mô phỏng hướng sự kiện tập trung chủ yếu vào các mô hình điều khiển tiến trình tính toán theo thời gian và có thể làm việc với những mô hình thời gian chính xác. Như vậy, phương pháp mô hình hoá lôgic và mô phỏng hướng sự kiện có tính tổng quát cao hơn và có thể áp dụng cho cả những mạch không đồng bộ.

Trước đây, phương pháp biên dịch được sử dụng khá phổ biến trong kỹ thuật nhưng với những nhược điểm trong việc xử lý thời gian trễ nên phương pháp này tỏ ra không thích hợp với việc phân tích hành vi của mạch theo thời gian. Do đó tại thời điểm hiện tại, phương pháp mô hình hoá lôgic và mô phỏng bằng biên dịch trở nên ít được sử dụng một cách độc lập mà thường được sử dụng kết hợp với những phương pháp khác. Nói chung, phương pháp biên dịch cũng tỏ ra khá thuận lợi trong việc mô hình hoá các mạch tổ hợp và trong một số trường hợp trong cả việc xây dựng mô hình cho các mạch tuần-tự đồng bộ.

Phương pháp mô hình hoá lôgic và mô phỏng hướng sự kiện có thể thao tác với những đầu vào thời gian thực. Điều đó có nghĩa là những đầu vào có số lần thay đổi trạng thái độc lập với hoạt tính của mạch được mô phỏng. Vấn đề này đóng vai trò quan trọng trong việc kiểm chứng thiết kế bởi vì phương pháp hướng sự kiện cho phép mô phỏng một cách chính xác những sự kiện không đồng bộ như các sự kiện ngắt hoặc các cạnh tranh trong yêu cầu sử dụng tuyến dữ liệu. Phương pháp mô hình hoá lôgic và mô phỏng bằng biên dịch chỉ cho phép các đầu vào thay đổi giá trị khi trạng thái mạch ổn định. Điều này thích hợp khi các tác động đầu vào là các vectơ được đặt vào mạch theo những tốc độ cố định. Ta chú ý rằng, những đầu vào thời gian thực bao gồm cả những vectơ đầu vào với tốc độ cố định.

Trong kỹ thuật hai phương pháp mô hình hoá và mô phỏng nói trên thường được sử dụng một cách kết hợp, trong đó những thủ tục hướng sự kiện sẽ truyền các sự kiện trên các đường tín hiệu qua các phần tử mạch còn những phần tử được kích hoạt sẽ thực hiện các thao tác lên tín hiệu bằng các mô hình xây dựng từ những mã biên dịch.

Các mức mô hình hoá lôgic và mô phỏng tương ứng với các mức biểu diễn hệ thống. Chúng ta có thể có các mức mô phỏng sau:

- Mô hình hoá trên mức thanh ghi: hệ thống sẽ được mô tả hoàn toàn trên mức thanh ghi truyền đạt hoặc như liên kết giữa những thành phần của mô hình trên thanh ghi;
- Mô hình hoá trên mức chức năng: hệ thống được mô tả bằng các thành phần cơ bản và liên kết giữa các thành phần đó;
- Mô hình hoá trên mức các phần tử lôgic;
- Mô hình hoá trên mức các transistor;
- Mô hình hoá hỗn hợp.

## 2. Các giá trị lôgic không xác định

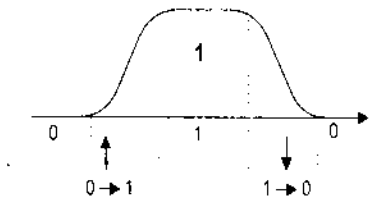
Trong quá trình mô hình hóa lôgic, để có thể mô tả chức năng và hoạt động của mạch theo thời gian, ta phải mô tả được một cách chính xác các quá trình diễn ra trong mạch. Để đạt được điều đó, chúng ta phải mô tả được sự biến thiên giá trị tín hiệu trên các đường tín hiệu trong mạch. Một trong những vấn đề phức tạp là mô tả được các quá trình quá độ diễn ra trong mạch mỗi khi xuất hiện các sự kiện trên những đường truyền tín hiệu. Do hoạt động của các mạch số dựa trên cơ sở của lôgic chuyển mạch, nhưng với hai giá trị { 0, 1 } ta không thể mô tả được sự chuyển tiếp giá trị tín hiệu từ một mức sang một mức khác. Điều đó dẫn tới việc ta phải mở rộng miền xác định của các phép toán lôgic truyền thống và mở rộng các phép toán này trên miền xác định mới.

Trong quá trình mô hình hoá, để mô tả các giá trị tín hiệu trên các đường tín hiệu trong các quá trình tính toán, người ta thường sử dụng hai dạng giá trị tín hiệu sau:

- Các giá trị tín hiệu thực - các đối tượng được mô hình hóa trực tiếp và tương ứng với các giá trị tín hiệu trong sơ đồ thực.

- Các giá trị ảo - các giá trị chỉ tồn tại trong mô hình mạch khi thực hiện quá trình mô phỏng.

Trong lôgic nhị phân, các giá trị tín hiệu thực không chỉ là '0' và '1'. Trong nhiều trường hợp nhà thiết kế cần thiết phải chỉ ra các trạng thái biến đổi giá trị từ '0' → '1' - ký hiệu là '↑' và từ '1'→ '0' – ký hiệu là '↓' của tín hiệu. Các trạng thái này thể hiện các quá trình quá độ trong mạch trên các đường truyền tín hiệu. Ngoài ra đối với những phần tử có ba trạng thái ta còn phải sử dụng thêm giá trị 'Z' để chỉ trạng thái trở kháng cao. Như vậy miền xác định của các phép toán lôgic được mở rộng từ tập hợp { 0, 1 } sang tập hợp {0, 1, ↑, ↓, Z}.




**Hình 5.2** Các giá trị tín hiệu 0, 1, ↑, ↓.

Các giá trị ảo được sử dụng trong những trường hợp trong quá trình mô hình hóa khi ta không thể thiết lập giá trị tín hiệu như là kết quả của các phép toán trên những giá trị thực hoặc khi ta phải mô hình hóa các phần tử của mạch trong điều kiện giá trị thời gian trễ của các phần tử không xác định. Ví dụ, khi mô phỏng hoạt động của phần tử NAND có hai đầu vào bằng các giá trị tín hiệu thực { 0, 1, ↑, ↓, Z }, chúng ta không thể thiết lập giá trị đầu ra bằng các giá trị tín hiệu thực nêu trên khi giá trị đầu vào là '↑' và '↓'. Trong những trường hợp như vậy chúng ta phải sử dụng thêm giá trị tín hiệu không xác định – ký hiệu là 'U'.

Nói chung, đáp ứng của các mạch tuần tự đối với các tác động phụ thuộc vào các giá trị khởi tạo ban đầu, do đó ở giai đoạn đầu của quá trình mô phỏng, chúng ta cần xác định những giá trị tín hiệu tương ứng với trạng thái ban đầu của mạch. Tuy nhiên khi mạch bắt đầu được cung cấp năng lượng, do giá trị thời gian trễ của các phần tử mạch là các đại lượng ngẫu nhiên nên trạng thái của các phần tử trigơ, thanh ghi, ô nhớ không xác định. Đó là nguyên nhân vì sao trước khi mạch bắt đầu thực hiện các chức năng tính toán thông thường, chúng ta thường đưa mạch về trạng thái ban đầu bằng một chuỗi các thao tác khởi tạo 'reset'. Như vậy, trong quá trình mô phỏng, khi cung cấp năng lượng cho mạch, ở thời điểm ban đầu tín hiệu thường được gán giá trị 'X'. Nếu giá trị của một tín hiệu là 'X' tại một thời điểm xác định, điều đó có nghĩa là tín hiệu có thể nhận giá trị hoặc '0' hoặc '1' tại thời điểm đó.

Thực chất các giá trị 'U' và 'X' có ý nghĩa khác nhau mặc dù chúng cùng là những giá trị không xác định. Giá trị 'X' chỉ sự bất định về trạng thái của mạch tại thời điểm ban đầu. Do có sự chạy đua giữa những phần tử lôgic thành phần và sự biến thiên ngẫu nhiên của tham số thời gian trễ của chúng nên ở thời điểm ban đầu, các phần tử nhớ có thể nhận các giá trị '0' hoặc '1' một cách ngẫu nhiên. Khi đó ta biểu thị trạng thái của mạch là 'X'. Giá trị 'U' xuất hiện khi chúng ta làm các phép toán trên các tín hiệu. Ví dụ, tại thời điểm t, giá trị tín hiệu $s_1$ là dãy "↑1", giá trị tín hiệu $s_2$ là "↓0". Các dãy tín hiệu này biểu thị các quá trình quá độ xảy ra trên đường tín hiệu $s_1$ và $s_2$ tại thời điểm t khi giá trị tín hiệu chuyển từ '0' sang '1' và từ '1' về '0'. Nếu các đường tín hiệu $s_1$ và $s_2$ là các đầu vào của phần tử AND, khi đó phần tử AND sẽ thực hiện phép toán *and*( '↑', '↓' ). Kết quả của phép toán này là không xác định. Trên đầu ra của phần tử AND tín hiệu sẽ không nhận giá trị '0' cũng như '1'. Trong trường hợp này chúng ta biểu thị giá trị tín hiệu bằng ký hiệu 'U'.

| AND | '0' | '1' | 'X' |
|---|---|---|---|
| '0' | '0' | '0' | '0' |
| '1' | '0' | '1' | 'X' |
| 'X' | '0' | 'X' | 'X' |

| OR | '0' | '1' | 'X' |
|---|---|---|---|
| '0' | '0' | '1' | 'X' |
| '1' | '1' | '1' | '1' |
| 'X' | 'X' | '1' | 'X' |

| NOT | |
|---|---|
| '0' | '1' |
| '1' | '0' |
| 'X' | 'X' |

**Hình 5.3** Mở rộng các phép toán lôgic sang hệ biểu diễn *3-giá trị*.

Nếu số lượng các giá trị thực và ảo của tín hiệu trong quá trình mô hình hoá lôgic và mô phỏng bằng *n* thì ta nhận được mô hình mô phỏng *n-giá trị* và hệ các phép toán lôgic tương ứng phải được mở rộng thành hệ lôgic *n-giá trị*. Ví dụ,

- Nếu n = 3, tập hợp các giá trị mà tín hiệu có thể nhận được trong mô hình mô phỏng của mạch sẽ là { 0, 1, X };
- Nếu n = 5, tập hợp các giá trị mà tín hiệu có thể nhận được trong mô hình mô phỏng của mạch sẽ là { 0, 1, ↑, ↓, U };
- Nếu n = 7, tập hợp các giá trị mà tín hiệu có thể nhận được trong mô hình mô phỏng của mạch sẽ là { 0, 1, ↑, ↓, U, X, Z };

Các phép toán trong trường hợp lôgic *3-giá trị* { 0, 1, X } được mở rộng như trên hình 5.3.

| NAND | '0' | '1' | '↑' | '↓' | 'U' |
|---|---|---|---|---|---|
| '0' | '1' | '1' | '1' | '1' | '1' |
| '1' | '1' | '0' | '↓' | '↑' | 'U' |
| '↑' | '1' | '↓' | '↓' | 'U' | 'U' |
| '↓' | '1' | '↑' | 'U' | '↑' | 'U' |
| 'U' | '1' | 'U' | 'U' | 'U' | 'U' |

| NOR | '0' | '1' | '↑' | '↓' | 'U' |
|---|---|---|---|---|---|
| '0' | '1' | '0' | '↓' | '1' | '1' |
| '1' | '0' | '0' | '0' | '0' | '0' |
| '↑' | '↓' | '0' | '↓' | 'U' | 'U' |
| '↓' | '↑' | '0' | 'U' | '↑' | 'U' |
| 'U' | 'U' | '0' | 'U' | 'U' | 'U' |

**Hình 5.4** Các phép toán NAND và NOR trong hệ lôgic *5-giá trị*.

Nếu ta sử dụng lôgic *5-giá trị* { 0, 1, ↑, ↓, U }, các phép toán NAND và NOR sẽ được mở rộng như trên hình 5.4.

Trong kỹ thuật thiết kế mạch với độ tích hợp lớn, rất ít khi quá trình mô hình hoá lôgic và mô phỏng được thực hiện chỉ với hai giá trị tín hiệu thực là { '0', '1' }. Thông thường chúng ta phải sử dụng ít nhất ba giá trị tín hiệu. Chúng ta xét một trường hợp đơn giản khi sử dụng lôgic *3-giá trị* đồng thời khảo sát những vấn đề nảy sinh khi mở rộng hệ lôgic và ảnh hưởng của chúng lên quá trình mô hình hoá lôgic và mô phỏng.

Như ta đã đề cập tới ở trên, khi bắt đầu cung cấp nguồn cho mạch, trong mạch xuất hiện sự chạy đua giữa các phần tử mạch. Điều đó dẫn tới việc trạng thái của mạch không xác định tại thời điểm ban đầu. Trạng thái của mạch sẽ nhận một cách ngẫu nhiên một trong hai giá trị '0' hoặc '1' tuỳ thuộc vào giá trị tham số trễ trong mạch. Trạng thái của mạch tại thời điểm này không thể dự đoán trước và được ký hiệu là 'X'. Giá trị 'X' này tương ứng với '0' hoặc '1' và có thể được tham gia vào quá trình thực hiện các phép toán lôgic cùng với các giá trị lôgic nhị phân khác trong quá trình mô hình hoá lôgic và mô phỏng. Sau khi thêm giá trị 'X', hệ các phép toán lôgic *2-giá trị* được mở rộng thành hệ các phép toán lôgic *3-giá trị*.

Giá trị 'X' biểu thị một trong hai giá trị của tập hợp {0,1}. Tương tự ta có thể coi giá trị '0' tương ứng với tập hợp {0}; giá trị '1' tương ứng với tập hợp {1}. Phép toán lôgic B giữa hai giá trị $p$ và $q$ trong đó

$$p,q \in \{ 0, 1, X \}$$

được xác định thông qua phép toán giữa các tập hợp biểu diễn giá trị của $p$ và $q$. Kết quả của phép toán B được xác định bằng hợp của tập hợp kết quả của mọi khả năng thực hiện phép toán $B$ với các thành phần của hai tập hợp tương ứng với hai toán hạng.

Ví dụ,

And( 0, X ) = And({ 0 }, { 0, 1 } ) =

= { And( 0, 0 ), And( 0, 1 ) } = { 0, 0 } = 0;

Or( 0, X ) = Or( { 0 }, { 0, 1 } ) =

={ Or( 0, 0 ), Or( 0, 1 ) } = { 0, 1 } = X

NOT( X ) = NOT( { 0, 1 } ) =

= { NOT( 0 ), NOT( 1 ) } = { 1, 0} = { 0, 1 } = X

Tất cả các kết quả của ba phép toán AND, OR, NOT trong hệ lôgic *3-giá trị* được đưa ra trên hình 5.3. Việc xác định giá trị của một hàm tổ hợp bất kỳ $f( x_1, x_2, ..., x_n )$ đối với một tập hợp các đầu vào $( v_1, v_2, ..., v_n )$ nhận giá trị trong tập hợp { 0, 1, X } được thực hiện như sau:

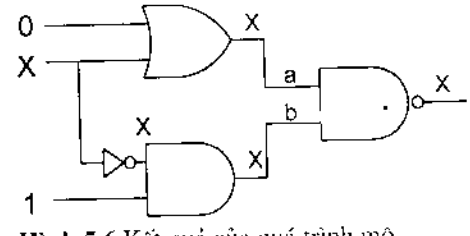
- Xây dựng khối Fx $( v_1, v_2, ..., v_n / x )$;
- Sử dụng phép toán giao mở rộng tìm giao của khối Fx với khối cơ sở của hàm *f*. Nếu tìm thấy tập hợp giao, giá trị của hàm *f* sẽ bằng giá trị nằm tại vị trí bên phải nhất; trong trường hợp ngược lại, giá trị hàm *f* sẽ bằng 'X'.

| ∩ | 0 | 1 | *x* | X |
|---|---|---|---|---|
| 0 | 0 | ∅ | 0 | ∅ |
| 1 | ∅ | 1 | 1 | ∅ |
| *x* | 0 | 1 | *x* | X |
| X | ∅ | ∅ | X | X |

**Hình 5.5** Biến thể của phép toán giao trong trường hợp lôgic *3-giá trị*.

Để hiểu sơ đồ tính toán này, với ký hiệu *x* trong một khối cơ sở chúng ta hiểu rằng ta không quan tâm tới giá trị của biến. Giá trị 'X' sẽ tương ứng với đỉnh *x* trong một khối cơ sở. Tuy nhiên, giá trị đầu vào nhị phân xác định trong khối cơ sở là cần thiết, do đó giá trị 'X' ở đầu vào không thể sinh ra giá trị đầu ra tương ứng. Ví dụ, đối với phần tử AND có hai đầu vào, khối ( X0l*x* ) tương thích với một khối cơ sở, trong khi đó khối ( X1l*x* ) không tương thích.

Khi sử dụng lôgic *3-giá trị* chúng ta sẽ bị mất thông tin trong biểu diễn chức năng và hành vi của mạch. Điều đó có thể thấy khi ta khảo sát trong bảng chân lý của phép toán NOT. Khi giá trị tại đầu vào của phần tử NOT là 'X', ta bị mất quan hệ nghịch đảo giữa giá trị tín hiệu vào và tín hiệu ra của phần tử -

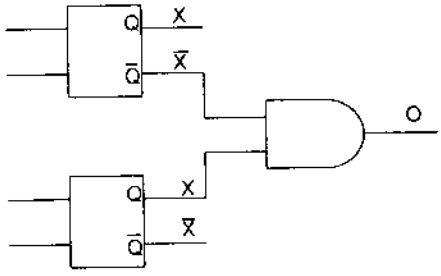



**Hình 5.6** Kết quả của quá trình mô phỏng trong hệ lôgic *3-giá trị*.

NOT( X ) = X. Điều đó cũng xảy ra khi ta xét giá trị đầu đảo $\overline{Q}$ của phần tử trigơ có trạng thái là 'X'. Vấn đề này dẫn tới những kết quả sai trong quá trình mô hình hoá và mô phỏng. Ví dụ, trong mạch trên hình 5.6, nếu tính toán theo lôgic *3-giá trị*, giá trị tín hiệu trên đầu ra của phần tử NAND phải là 'X'. Trên thực tế thì đường tín hiệu này nhận giá trị '1' do tín hiệu trên hai đường *a* và *b* luôn ngược nhau. Giá trị '1' trong trường hợp này là tất định không phụ thuộc vào giá trị trên đường tín hiệu vào 'X'. Để giải quyết vấn đề đó chúng ta đưa vào đi kèm với giá trị 'X' là giá trị $\overline{X}$ thoả mãn các hệ thức:

$$X.\ \overline{X} = 0$$
$$X + \overline{X} = 1$$

Việc đưa thêm giá trị $\overline{X}$ vào lôgic *3-giá trị* cũng chỉ giải quyết được một phần vấn đề nêu trên khi trong mạch ta chỉ có một biến trạng thái nhận giá trị 'X'. Trong những trường hợp khác điều này có thể đưa đến lỗi trong quá trình mô hình hoá lôgic và mô phỏng như trong ví dụ mạch trên hình 5.7. Trong ví dụ này, hai phần tử trigơ là độc lập. Đối với hai đầu ra Q và $\overline{Q}$ của cùng một trigơ, giá trị tín hiệu là đảo nhau nên nếu đầu ra Q có giá trị 'X' thì đầu ra $\overline{Q}$ sẽ nhận giá trị ' $\overline{X}$'. Nhưng đối với hai phần tử trigơ độc lập thì giữa đầu ra Q của một phần tử và đầu ra $\overline{Q}$ của phần tử thứ hai không có quan hệ nghịch đảo đó. Do đó, nếu tính theo lôgic *3-giá trị* và giá trị ' $\overline{X}$' thì đầu ra của phần tử AND có giá trị '0' ( X and $\overline{X}$ = 0 ), nhưng trên thực tế giá trị tại đầu ra của phần tử AND là 'X'. Như vậy chỉ sử dụng hai giá trị X và $\overline{X}$ không giải quyết triệt để vấn đề về tính toàn vẹn thông tin.




**Hình 5.7** Trường hợp sử dụng lôgic 3-giá trị cùng giá trị $\overline{X}$ cho kết quả sai.

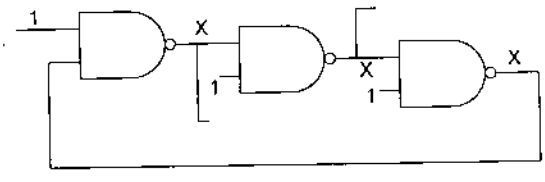
Để giải quyết vấn đề nêu trên,chúng ta có thể sử dụng nhiều tín hiệu không xác định khác nhau $X_1$, $X_2$,..., $X_k$ tương ứng với các nguồn tín hiệu độc lập ( các tín hiệu độc lập này có thể là các biến trạng thái của các phần tử mạch ) và các quy tắc tính toán độc lập:

$$X_i.\ \overline{X}_i = 0\ ;\ X_i + \overline{X}_i = 1$$

Đối với những mạch có độ tích hợp lớn thì phương pháp này không dùng được bởi vì các giá trị trên một đường tín hiệu có thể phải được biểu diễn bằng một biểu thức lôgic lớn với nhiều biến độc lập .

Thông thường phép toán do một phần tử chức năng thực hiện được xác định bằng cách mã hóa các giá trị trên nhóm các đường tín hiệu điều khiển. Vấn đề nảy sinh trong quá trình mô hình hoá lôgic và mô phỏng khi phần tử chức năng cần thực hiện và một số đường tín hiệu điều khiển nhận giá trị 'X'. Một cách tổng quát, nếu $k$ đường tín hiệu điều khiển nhận giá trị 'X', phần tử chức năng có thể thực hiện một trong $2^k$ phép toán. Lời giải chính xác sẽ nhận được khi ta thực hiện toàn bộ $2^k$ phép toán và kết quả của phép toán là hợp của tất cả các kết quả riêng phần của từng phép toán trong $2^k$ phép toán nói trên. Như vậy, nếu một biến số nhận giá trị '0' trong một số phép toán, và nhận giá trị '1' trong một số phép toán khác, kết quả sau khi tổng hợp lại sẽ là { 0, 1 } = 'X'. Phương pháp này chỉ khả thi khi $2^k$ là một số nguyên nhỏ. Ví dụ, ta giả thiết rằng hai bit trong địa chỉ của bộ nhớ ROM nhận giá trị 'X'. Điều này sẽ dẫn tới bốn giá trị địa chỉ, trong đó mỗi giá trị cho phép truy cập tới những từ máy khác nhau trong bộ nhớ. Kết quả tại đầu ra sẽ nhận giá trị nhị phân $b$ tại những vị trí bit trong đó mỗi từ được truy cập nhận giá trị b, và nhận giá trị 'X' tại những vị trí mà từ được truy cập không tương thích với địa chỉ.

Trong những mạch không đồng bộ, sự xuất hiện của giá trị 'X' có thể dẫn tới dao động trong mạch. Điều này có thể thấy trong hình 5.8. Các tín hiệu tham gia vào dao động tần số cao có thể được coi là hiệu điện thế giữa các mức tương ứng với giá trị lôgic '0' và '1'. Như vậy, bổ sung thêm vào các giá trị không xác định tĩnh giá trị 'X' còn có thể biểu diễn giá trị không xác định động hoặc giá trị lôgic trung gian.




**Hình 5.8** Hiện tượng dao động trong mạch khi sử dụng lôgic *3-giá trị*.

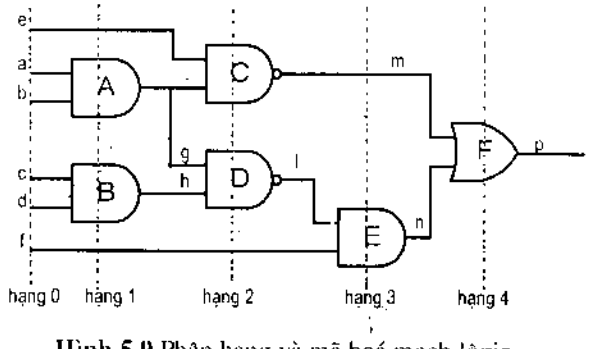
## §5.2. Phương pháp mô hình hoá biên dịch

Phương pháp mô hình hoá lôgic và mô phỏng bằng biên dịch là phương pháp mô hình hóa trong đó các tín hiệu được xác định giá trị bằng cách tạo cho mỗi phần tử của mạch một mã lệnh tương ứng với các phép toán mà phần tử đó cần thực hiện. Các mã lệnh nhận được sẽ được biểu diễn theo một trật tự tương ứng.

Trong phương pháp mô hình hoá và mô phỏng bằng biên dịch, mô hình bằng các mã lệnh là một thành phần của hệ thống mô phỏng. Trong những trường đặc biệt, hệ mô phỏng chính là các mô hình trên các mã lệnh. Nói chung, mô hình trên mã lệnh được kết nối với hệ mô phỏng, trong đó mỗi tiến trình bao gồm việc đọc các vectơ đầu vào, thực hiện mô hình với từng vectơ đầu vào và hiển thị kết quả.

Quá trình mô hình hoá bằng phương pháp biên dịch sẽ có những bước sau:

- Bước một: phân hạng các phần tử của mạch cần mô hình hoá theo trật tự thực hiện các phép toán sao cho không có mâu thuẫn nảy sinh.
- Bước hai: tạo các mã lệnh tương ứng với các phép toán do các phần tử thực hiện.




**Hình 5.9** Phân hạng và mã hoá mạch lôgic.

Để phân hạng các phần tử của mạch, đầu tiên chúng ta ngắt các vòng phản hồi trong mạch nếu có. Các điểm ngắt được xác định tương ứng với các chức năng của mạch. Trong trường hợp mô hình hoá các mạch tuần tự đồng bộ việc ngắt mạch phản hồi thực hiện trên những phần tử trực tiếp nhận các tín hiệu đồng bộ.

Giả sử $l(k)$ là hạng của phần tử $k$, khi đó quá trình phân hạng được thực hiện như sau:

- Các đầu vào của mạch có hạng '0'.
- Nếu $k_1, k_2, ..., k_p$ là các phần tử được nối với các đầu vào của phần tử $k$, hạng của phần tử $k$ sẽ bằng:

$$l(k) = 1 + \max(l(k_1), ..., l(k_p)).$$

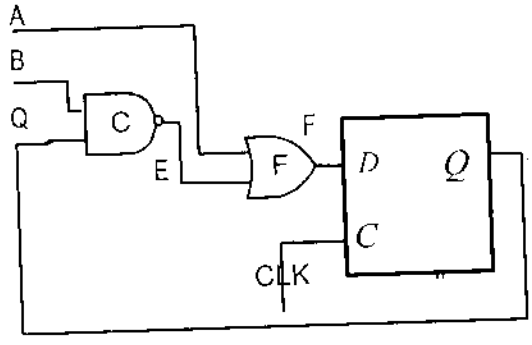
Ví dụ, xét mạch trên hình 5.9, theo phương pháp phân hạng nêu trên ta sẽ nhận được hạng của các phần tử trong mạch.

Tạo mã các lệnh là việc thay thế các phép toán mà phần tử thực hiện được bằng những lệnh mà máy hiểu được. Ví dụ, trong sơ đồ trên hình 5.9, các phần tử sẽ được mã hóa bằng những lệnh sau:

1. g = And (a,b)
2. h = And (c,d)
3. m = Nand (e,g)
4. l = Nand (g,h)
5. n = Nand (l,f)
6. p = Or (m,n)

Trình tự mã hóa được thực hiện theo thứ tự tăng dần hạng của các phần tử mạch. Dãy lệnh này sẽ được biên dịch và thực hiện với những giá trị vào cho trước.

Chúng ta xét trường hợp mạch tuần tự đồng bộ như trên hình 5.10. Mạch tuần tự này được điều khiển bằng tín hiệu đồng hồ CLK. Giả thiết rằng sau khi vectơ đầu vào mới được tác động vào mạch, ta có đủ thời gian để giá trị trên đường dữ liệu vào của phần tử trigơ được ổn định ít nhất là một thời gian $t_s$ trước khi phần tử trơg được kích hoạt bằng tín hiệu đồng hồ. Điều đó có nghĩa là thời gian $t_s$ là thời gian thiết lập của phần tử trigơ. Nếu điều kiện này được thoả mãn, ta có thể bỏ qua những giá trị thời gian trễ của từng phần tử lôgic trong quá trình mô hình hoá và mô phỏng, bởi vì thời điểm chính xác khi tín hiệu thay đổi giá trị trong mạch tổ hợp không quan trọng. Do đó đối với mỗi vectơ giá trị đầu vào, quá trình mô phỏng chỉ cần tính giá trị tĩnh của đường tín hiệu F và truyền giá trị đó tới đầu ra Q. Theo phương pháp phân hạng thì các đầu vào A, B sẽ nằm trên hạng '0', nhưng tín hiệu trạng thái của trigơ cũng nằm trên hạng '0' nếu ta giả thiết giá trị khởi tạo của trạng thái được biết trước. Trong trường hợp này hệ thống mô phỏng sẽ thực hiện mô hình mạch như trong trường hợp mô hình hoá mạch tổ hợp bằng cách tính các giá trị tín hiệu trên đầu ra của tất cả các phần tử đối với mỗi vectơ giá trị tín hiệu



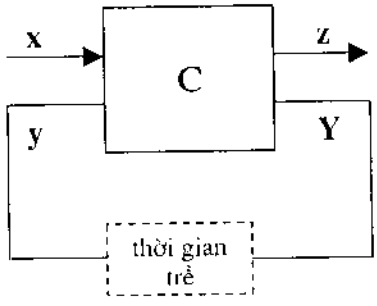

**Hình 5.10** Phân hạng và mã hoá mạch tuần tự đồng bộ.

đầu vào. Trong trường hợp giá trị của phần tử trigơ không được biết trước, hệ thống mô phỏng phải thực hiện tính toán trên các giá trị '0', '1' và 'X'. Các giá trị này được mã hoá hai bit như sau: '0' – "00"; '1' – "11"; 'X' – "01". Các phép toán lôgic hai ngôi AND ( OR ) giữa các chuỗi hai bit được thực hiện trên cơ sở các phép toán một bit tương ứng, nhưng phép toán NOT không thể thực hiện trên bit được vì giá trị NOT( X ) cho ta kết quả chuỗi "10" – là giá trị không có trong bảng mã. Vấn đề này được giải quyết bằng cách đổi chỗ hai bit sau khi đảo bit.

Từ những điều đưa ra ở trên, ta thấy, phương pháp biên dịch không tính tới ảnh hưởng của thời gian trễ khi tín hiệu được truyền qua mạch. Điều này là do quá trình xây dựng mô hình mạch được thực hiện theo hạng.

- Nếu chúng ta chỉ giới hạn trường hợp thời gian trễ là các thời gian trễ lan truyền thì việc mô hình hoá hoạt động của mạch có thể thực hiện được theo cách phân hạng. Các thời gian trễ lan truyền sẽ được tính đến một cách tường minh trong quá trình phân hạng và truyền tín hiệu qua từng lớp phân hạng.
- Đối với các dạng thời gian trễ khác như thời gian trễ ngẫu nhiên hoặc thời gian trễ quán tính thì việc mô hình hoá mạch theo phương pháp biên dịch không thể thực hiện được.
- Do ta phải ngắt các vòng phản hồi trong mạch, phương pháp biên dịch chỉ có thể sử dụng trong những trường hợp khi ý nghĩa của việc ngắt vòng phản hồi rõ ràng. Ví dụ như trong trường hợp các mạch đồng bộ.

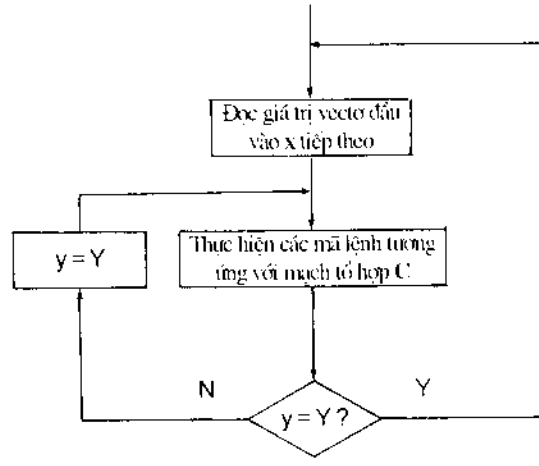
Mô hình hoá mạch bằng phương pháp biên dịch có thể được thực hiện khá nhanh. Nhưng phương pháp này có nhược điểm quan trọng là không tính thời gian trễ trong mạch. Nếu ta coi thời gian trễ trên tất cả các phần tử mạch được coi là bằng nhau thì việc phân hạng sẽ thể hiện được sự trễ của tín hiệu khi đi qua mạch. Trong trường hợp thời gian trễ có những dạng phức tạp thì phương pháp biên dịch không thể thực hiện được chính xác. Khi mô hình hoá các mạch đồng bộ, việc ngắt các vòng phản hồi cũng làm cho ngữ nghĩa của mạch bị thay đổi. Trong trường




**Hình 5.11** Mô hình mạch không đồng bộ với tham số thời gian trễ tập trung trên đường phản hồi.

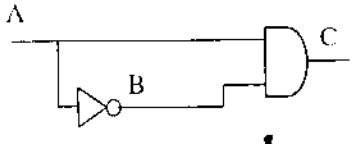
hợp đối với những mạch không đồng bộ việc không tính đến thời gian trễ có thể dẫn tới những kết quả sai.

Để giải quyết một phần những khó khăn trên, đối với những mạch không đồng bộ, ta phải giả thiết giá trị thời gian trễ chỉ tập trung trên đường phản hồi ( hình 5.11 ). Để đáp ứng với vectơ đầu vào ***x***, mạch phải trải qua một chuỗi các lần chuyển trạng thái. Quá trình chuyển trạng thái này được biểu diễn bằng sự thay đổi giá trị của biến trạng thái ***y***. Ta giả thiết rằng vectơ tín hiệu đầu vào chỉ tác động khi trạng thái của mạch ổn định ***y*** = **Y**. Đường tín hiệu phản hồi có hạng '0' phải được xác định trước khi các mã lệnh tương ứng với mạch tổ hợp **C** được tạo ra. Trên hình 5.12 chúng ta có thuật toán mô hình hoá lôgic và mô phỏng mạch không đồng bộ. Khi thực hiện mô phỏng mô hình, quá trình mô hình hoá sẽ thực hiện tính giá trị tín hiệu **z** và **Y** dựa trên giá trị **x** và ***y***.



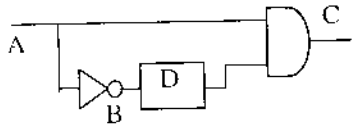

**Hình 5.12** Mô phỏng mạch không đồng bộ bằng mô hình mã lệnh biên dịch.

Phương pháp mô hình hoá và mô phỏng nói trên không chính xác khi thực hiện đối với các mạch không đồng bộ trong đó các thao tác tính toán đều dựa trên tham số thời gian trễ của các phần tử mạch. Ví dụ, ta xét mạch tạo xung biểu diễn trên hình 5.13, khi tín hiệu trên đường A có sự thay đổi giá trị từ '0' sang '1', nếu mạch đảo B không làm trễ tín hiệu thì tín hiệu trên đường C sẽ luôn nhận giá trị '0' do tín hiệu trên hai đường A và B luôn ngược nhau. Nếu phần tử B làm trễ tín hiệu, trong khoảng thời gian có độ dài bằng giá trị thời gian trễ tín hiệu qua phần tử B, hai đầu vào của phần tử AND sẽ có cùng giá trị '1' và trên đường tín hiệu C sẽ xuất hiện xung '0' → '1' → '0' có độ rộng bằng giá trị tham số trễ của phần tử đảo B. Nếu chúng ta không xây dựng mô hình bằng phương



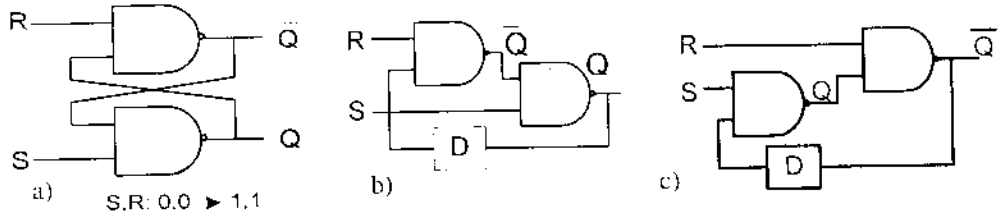

**Hình 5.13** Mạch tạo xung.

pháp biên dịch một cách cẩn thận, xung '1' này sẽ không xuất hiện trên đường tín hiệu C bởi vì phương pháp biên dịch chỉ quan tâm tới hành vi tĩnh của mạch ( nếu không quan tâm đến sự trễ tín hiệu, giá trị trên đường C luôn bằng '0' ). Nếu chúng ta thêm vào phần tử trễ D vào đường tín hiệu tại đầu ra của phần tử B, mạch sẽ được mô phỏng một cách đúng đắn. Trong trường hợp này phần tử trễ D đóng vai trò phần tử trễ tập trung trên đường phản hồi. Nói chung ta không thể tạo ra những mô hình đúng đắn đối với mạch không đồng bộ một cách tự động mà cần phải có các kinh nghiệm của các nhà thiết kế.




**Hình 5.14** Mô hình mạch tạo xung kèm tham số trễ.

Trong những trường hợp, khi cấu trúc của mạch cho phép xác định một cách tường minh các vòng phản hồi, ta vẫn có thể tạo ra các mô hình biên dịch khác nhau để mô tả cùng một mạch. Trong mỗi mô hình khác nhau của mạch, ta có những giả thiết khác nhau về vị trí của phần tử trễ tập trung nên những mô hình này có thể phản ứng khác nhau đối với cùng một tác động. Ta hãy xét trường hợp mô hình hoá lôgic và mô phỏng mạch trigơ RS khi giá trị vectơ tín hiệu đầu vào RS chuyển tiếp từ "00" sang "11". Nếu ta sử dụng mô hình với đường tín hiệu Q là đường tín hiệu phản hồi ( hình 5.15 b. ), ta thấy đường tín hiệu $\bar{Q}$ chuyển từ '1' sang '0' trong khi giá trị trên đường Q luôn bằng '1' đối với cả hai vectơ đầu vào. Tuy nhiên nếu ta sử dụng mô hình với đường tín hiệu $\bar{Q}$ là đường tín hiệu phản hồi, ta sẽ có đường tín hiệu Q sẽ chuyển trạng thái từ '1' sang '0', trong khi đường tín hiệu $\bar{Q}$ vẫn giữ nguyên giá trị '1'. Hiện tượng này phát sinh từ nguyên nhân sau: hai vectơ này tạo ra sự chạy đua giữa các phần tử NAND, trong đó kết quả phụ thuộc vào thời gian trễ thực tế trên các phần tử đó. Như vậy, ví dụ này cho ta thấy phương pháp mô hình hoá lôgic và mô phỏng bằng phương pháp biên

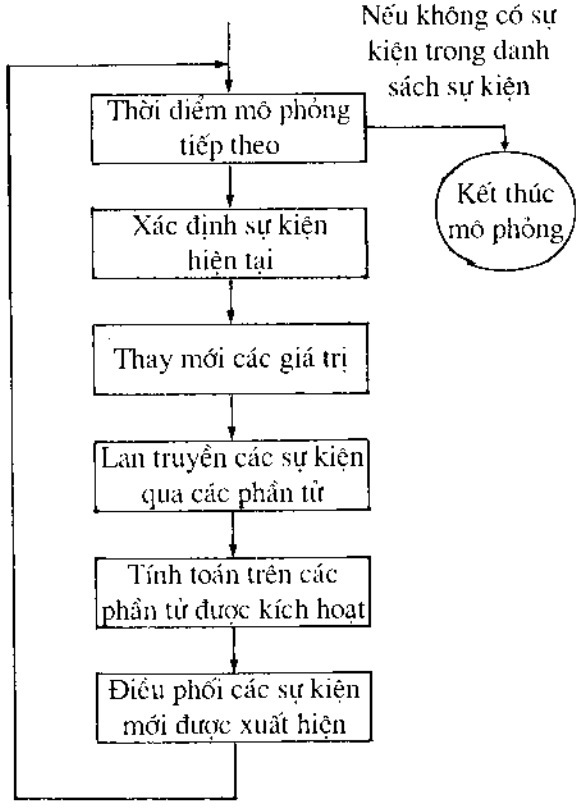



**Hình 5.15** Mạch trigơ RS và phương pháp mô hình hoá biên dịch.

dịch theo sơ đồ trên hình 5.12 không thích hợp với những mạch làm việc trong những chế độ có sự chạy đua giữa các phần tử hoặc rủi ro (hazard). Các hiện tượng này thường thấy trong các mạch không đồng bộ.

## §5.3. Phương pháp mô hình hoá hướng sự kiện

Phương pháp mô hình hoá hướng sự kiện cho phép xây dựng những mô hình mô phỏng trên máy tính với độ chính xác cao, trong đó ta có thể mô tả toàn bộ các liên kết trong mạch và dựa vào đó xây dựng những mô hình hoạt động của mạch theo tiến trình thời gian.




**Hình 5.16** Sơ đồ thực hiện mô hình hoá lôgic theo sự kiện.

Trong quá trình mô hình hoá lôgic và mô phỏng hướng sự kiện, chúng ta quan sát quá trình truyền các sự kiện xuất hiện trên các đường tín hiệu của mạch từ đầu vào đến đầu ra. Trong trường hợp này, ta giả thiết rằng vùng ảnh hưởng gây nên bởi sự biến thiên của một tín hiệu không lớn. Trong những mạch thông thường, sự thay đổi của tín hiệu đầu vào sẽ ảnh hưởng tới khoảng 10% các đường tín hiệu trong mạch. Như vậy, chúng ta thực hiện mô hình hoá lôgic và mô phỏng chỉ khi xuất hiện các sự thay

đổi giá trị tín hiệu trong mạch, hay nói cách khác là khi xuất hiện các sự kiện trong mạch. Điều này làm tăng hiệu suất của quá trình mô hình hoá và mô phỏng.

Hệ thống mô hình hoá lôgic và mô phỏng hướng sự kiện sử dụng mô hình cấu trúc của mạch để lan truyền các sự kiện. Sự thay đổi giá trị tại các đầu vào chính của mạch được xác định bằng các vectơ tín hiệu kích hoạt. Mọi sự kiện trên các đường tín hiệu khác của mạch được tính toán theo các phần tử bị kích hoạt.

Các sự kiện xuất hiện tại những thời điểm thời gian mô phỏng xác định. Cơ chế điều phối thời gian của quá trình mô phỏng điều khiển sự xuất hiện của các sự kiện theo một trật tự xác định. Các tác động vào mạch sẽ được biểu diễn bằng dãy các sự kiện xuất hiện trên các đường tín hiệu tại các thời điểm thời gian xác định. Các sự kiện sẽ xuất hiện trong tương lai sẽ phải chờ và được lưu trong danh sách các sự kiện. Những sự kiện trong danh sách sự kiện sẽ được điều phối và xử lý tại những thời điểm mô phỏng.

Sơ đồ của quá trình mô hình hoá lôgic và mô phỏng hướng sự kiện được thể hiện trên hình 5.16. Thời điểm mô phỏng hiện thời được chuyển tiếp thành thời điểm tiếp theo đối với những sự kiện đang ở trạng thái chờ và thời điểm tiếp theo này chuyển thành thời điểm mô phỏng hiện tại. Sau đó, hệ thống mô phỏng sẽ duyệt danh sách sự kiện và chọn những sự kiện được điều phối để xuất hiện trong thời điểm hiện tại đồng thời cập nhật lại giá trị của những tín hiệu kích hoạt. Hệ mô phỏng sẽ duyệt danh sách các đường rẽ nhánh của tín hiệu kích hoạt để xác định các phần tử sẽ được kích hoạt. Trong mạch thực, quá trình truyền sự kiện tại các điểm rẽ nhánh được thực hiện song song. Quá trình tính toán trên các phần tử bị kích hoạt có thể sinh ra các sự kiện mới và những sự kiện mới này sẽ được điều phối để xuất hiện tại những thời điểm tiếp theo tương ứng với tham số thời gian trễ của các phép toán liên quan tới phần tử đó. Hệ thống mô phỏng sẽ đưa những sự kiện mới được sinh ra này vào danh sách sự kiện và quá trình mô hình hoá lôgic và mô phỏng sẽ tiếp tục cho đến khi còn có trạng thái kích hoạt lôgic trong mạch, hay nói cách khác là cho đến khi mà danh sách sự kiện rỗng.

Quá trình tính toán trong phần tử M có thể sinh ra các sự kiện thay đổi trạng thái của phần tử. Khi những sự kiện thay đổi trạng thái xuất hiện, sự kiện này chỉ kích hoạt phần tử M và phần tử này sẽ thực hiện các tính toán. Để đơn giản chúng ta giả thiết rằng tất cả các sự kiện được xác định các tác động kích hoạt mạch được đưa vào danh sách sự kiện trước khi mô hình hoá

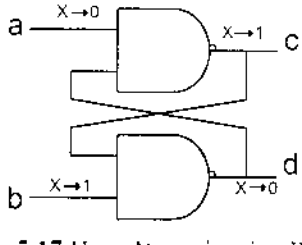
lôgic và mô phỏng. Trên thực tế, hệ mô phỏng định kỳ đọc các tệp chứa tác động kích hoạt mạch và kết hợp các sự kiện tại đầu vào với các sự kiện được sinh ra trong quá trình hoạt động của mạch. Thêm vào đó, ngoài các sự kiện sinh ra sự tính toán các giá trị tín hiệu trong mạch, trong kỹ thuật mô hình hoá lôgic hệ thống mô phỏng còn có thể thêm các sự kiện điều khiển quá trình. Các sự kiện loại này khởi tạo các hoạt động khác nhau của mạch tại các thời điểm thời gian xác định. Dưới đây là một số tác động của các sự kiện điều khiển:

- Hiển thị giá trị của những tín hiệu xác định;
- Kiểm tra giá trị của các tín hiệu cụ thể và có thể dừng quá trình mô phỏng nếu có sai sót trong quá trình tính toán.

Để mô tả rõ quá trình mô hình hoá lôgic và mô phỏng hướng sự kiện, chúng ta cụ thể hoá thuật toán mô phỏng hướng sự kiện như sau.

Nếu ký hiệu tập hợp các cặp "giá trị tín hiệu - đường truyền tín hiệu thay đổi ở thời điểm thời gian hiện tại" bằng $\mathbf{V}_0$ và tập hợp các cặp "giá trị tín hiệu - đường truyền tín hiệu thay đổi dưới tác dụng của $\mathbf{V}_0$" bằng $\mathbf{V}_1$. Khi đó thuật toán mô hình hóa lôgic hướng sự kiện thực hiện theo trình tự như sau:

1. Thiết lập hai tập hợp $\mathbf{V}_a$ và $\mathbf{V}_b$ và gán $a \leftarrow 0$ , $b \leftarrow 1$
2. Thiết lập giá trị của trạng thái ban đầu của mạch.
3. Đọc các giá trị đầu vào. Xét các đường tín hiệu mà trên đó xuất hiện các sự kiện ( sự thay đổi giá trị tín hiệu ). Các giá trị của sự kiện xảy ra này được ghi vào tập hợp $\mathbf{V}_a$.
4. Kiểm tra tập hợp $\mathbf{V}_a$, nếu $\mathbf{V}_a$ rỗng ta quay về bước 3; nếu $\mathbf{V}_a$ không rỗng, thực hiện bước 5.
5. Thực hiện mô hình hóa lôgic đối với các giá trị trong tập hợp $\mathbf{V}_a$; các đường dữ liệu trên đó xuất hiện sự kiện và giá trị của chúng được ghi vào $\mathbf{V}_b$.
6. Gán $a \leftarrow 1$ , $b \leftarrow 0$ , vai trò của $\mathbf{V}_a$ và $\mathbf{V}_b$ thay đổi. Kiểm tra sự xuất hiện dao động trong mạch. Nếu không có dao động, quay về bước 4.



**Hình 5.17** Hoạt động của trigơ RS.

Như vậy, chúng ta thấy phương pháp mô hình hoá lôgic hướng sự kiện khác với phương pháp mô hình hoá bằng biên dịch ở chỗ: ta không cần thực hiện việc phân hạng các phần tử. Thêm vào đó với sự có mặt của vòng lặp, ta có thể thực

hiện việc mô hình hóa với độ phức tạp tuỳ ý.

Ví dụ, ta mô hình hoá quá trình hoạt động của trigơ RS:

1. B1: Thiết lập $V_A$ và $V_B$ ( A ← 0 , B ← 1) ;
2. B2: Do trạng thái ban đầu của mạch không xác định, giá trị trên tất cả các đường tín hiệu được gán bằng 'X';
3. B3: Đặt các giá trị đầu vào : a = 0, b = 1. Ở đây ta có $V_0$ = { ( a, 0 ), ( b, 1 ) };
4. B4: Ta thấy tập hợp $V_0$ không rỗng, ta chuyển sang bước 5.
5. B5: Giá trị trên đường tín hiệu ra *c* thay đổi từ 'X' sang '1', tập hợp $V_1$ = { ( c, 1 ) };
6. B6: Gán $V_1$ vào $V_0$. Tập hợp $V_0$ nhận giá trị $V_0$ = { ( c, 1) } và quay lại bước 4.
7. B4: Ta thấy tập hợp $V_0$ không rỗng ta chuyển tới bước 5.
8. B5: Giá trị tín hiệu trên đường *d* chuyển từ 'X' sang '0'. $V_1$ = { ( d, 0 ) }
9. B6: Gán $V_1$ vào $V_0$. Tập hợp $V_0$ nhận giá trị $V_0$ = { ( d, 0 ) } và quay về bước 4.
10. B4: Ta thấy tập hợp $V_0$ không rỗng, ta sang bước 5.

    B5: Do không có tín hiệu vào và không có tín hiệu thay đổi khi d = 0, $V_1 = \varnothing$.
11. B6: Ta gán $V_1$ vào $V_0$, khi đó $V_0 = \varnothing$ và chuyển sang bước 4.
12. B4: Ta thấy tập hợp $V_0$ rỗng nên quay về bước 3.
13. B3: Đọc các giá trị tiếp theo ở đầu vào.

Như vậy với giá trị đầu vào a = '0', b = '1', ta nhận được các giá trị đầu ra c = '1', d = '0'.

Như vậy chúng ta đã thấy nguyên lý của phương pháp mô hình hoá hướng sự kiện. Trong trường hợp cần phải tính tới ảnh hưởng của tham số thời gian trễ của các các phần tử, ta không thể chỉ sử dụng hai trạng thái $V_0$, $V_1$ . Trong trường hợp này ta có thể đưa thêm vào quá trình mô phỏng các trạng thái $V_j$ tương ứng với các giá trị thời gian trễ và sắp xếp theo trình tự các trạng thái $V_j$ .

## §5.4. Mô hình hoá quá trình trễ tín hiệu trong các phần tử mạch

Khi chúng ta thực hiện mô hình hóa lôgic và mô phỏng hoạt động của mạch trên mức các phần tử lôgic, một nhân tố quyết định mức độ chính xác của mô hình so với mạch thực tế là tham số thời gian trễ của các phần tử lôgic. Nếu chúng ta biểu diễn hoạt động của mạch không tương ứng với những tình huống thực tế sẽ diễn ra trong các phần tử lôgic, khi đó sẽ xuất hiện sự không chính xác trong các mô hình hoạt động của mạch chủ yếu theo những quan hệ về thời gian. Khi thực hiện quá trình mô hình hóa hướng sự kiện theo sơ đồ trong mục 5.3 ( hình 5.16 ) chúng ta có thể nhận được những biến thể của thuật toán. Sự khác nhau căn bản trong những biến thể đó phụ thuộc chủ yếu vào các mô hình trễ tín hiệu tương ứng với hoạt động của những phần tử lôgic tham gia vào mạch. Việc mô hình hóa quá trình trễ tín hiệu là nhân tố cơ bản trong việc xác định tính chính xác của mô hình mạch và mức độ phức tạp của thuật toán mô hình hóa.

### 1. Mô hình hóa quá trình trễ tín hiệu qua các phần tử lôgic

Mỗi phần tử lôgic đều tác động lên các tín hiệu vào và làm trễ các tín hiệu đó. Trong kỹ thuật, ta thường sử dụng hai mô hình trễ qua các phần tử lôgic: mô hình trễ lan truyền và mô hình trễ quán tính. Ngoài hai loại trễ nói trên, trong một số trường hợp để mô tả được hoạt động của mạch, người ta còn sử dụng các mô hình trễ ngẫu nhiên.

- Trễ lan truyền: là sự trễ tín hiệu phát sinh khi ta cho tín hiệu đi qua phần tử mạch.
- Trễ quán tính: là sự trễ tín hiệu gắn liền với năng lượng để kích hoạt phần tử mạch.

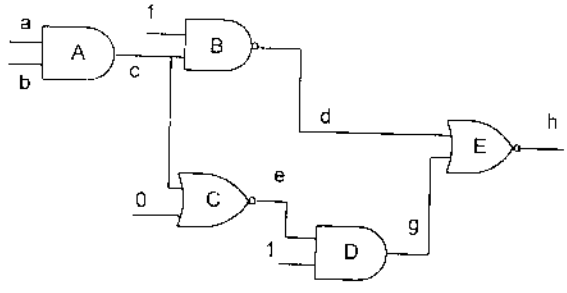
### a. Trễ lan truyền

Thời gian trễ lan truyền là thời gian mà tín hiệu ra của mạch chậm pha so với tín hiệu vào mạch. Mô hình trễ lan truyền là mô hình cơ bản, trong đó xác định khoảng thời gian $\Delta_T$ cách biệt giữa các sự kiện xuất hiện tại đầu ra với các sự kiện xuất hiện tại đầu vào sinh ra chúng. Để đơn giản, trong các

thuật toán mô hình hoá và mô phỏng, giá trị thông số thời gian trễ sử dụng trong mô phỏng là các số nguyên.

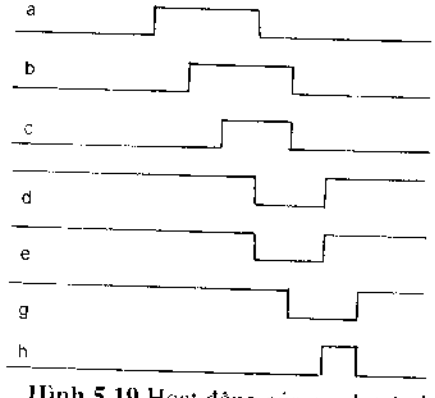
Trong một số trường hợp, khi mô hình hoá mạch ta không tính đến thời gian trễ tín hiệu khi đi qua các phần tử. Lúc đó, chúng ta coi thời gian trễ của phần tử bằng không. Như vậy trong trường hợp này, chúng ta sử dụng mô hình với thời gian trễ tín hiệu bằng không. Phương pháp mô hình hoá lôgic và mô phỏng bằng biên dịch sử dụng mô hình thuộc tính trễ này. Mô hình các phần tử với thời gian trễ bằng không chỉ được sử dụng để mô hình hóa các giá trị lôgic trong những mạch tổ hợp và mạch tuần tự đồng bộ.

Mô hình của mạch trong đó thời gian trễ của tất cả các phần tử lôgic bằng nhau gọi là mô hình với thuộc tính trễ thuần nhất. Do các phần tử lôgic có thời gian trễ khác không, như vậy chúng ta có khả năng mô hình hóa và xử lý các mạch không đồng bộ có phản hồi. Mô hình này là mô hình chứa thông số trễ đơn giản nhất. Trong trường hợp này, ta có thể chọn giá trị thông số trễ làm đơn vị với tỷ lệ một. Như vậy, chúng ta có mô hình trễ gọi là mô hình trễ đơn vị.

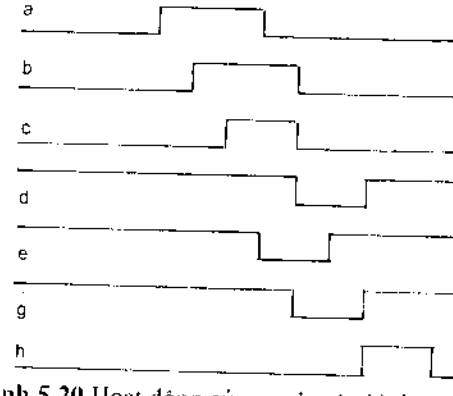



**Hình 5.18** Ví dụ sơ đồ mạch cho trường hợp trễ thuần nhất và trễ phân tán.

Trong trường hợp tổng quát, các phần tử của mạch có thể nhận những thông số thời gian trễ khác nhau. Để thực hiện quá trình mô hình hoá hoạt




**Hình 5.19** Hoạt động của mạch trên hình 5.18 với mô hình trễ thuần nhất.

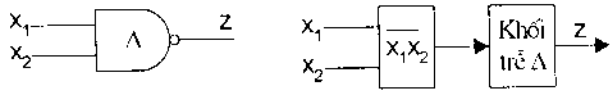



**Hình 5.20** Hoạt động của mạch trên hình 5.18 với mô hình trễ phân tán. $\Delta_0 = \Delta_1 = 2$, trên những phần tử khác $\Delta = 1$.

động của mạch, ta sử dụng đơn vị thời gian để đo giá trị thông số thời gian trễ là ước số chung lớn nhất $\Delta_T$ của các thông số thời gian trễ trên các phần tử và phân phối các giá trị thông số thời gian trễ theo tỷ lệ tương ứng với $\Delta_T$. Mô hình đó gọi là mô hình hoạt động với thông số thời gian trễ phân tán. Mô hình là mô hình phổ biến để biểu diễn các thuộc tính trễ lan truyền.

Ví dụ, đối với mạch trên hình 5.18, nếu ta thực hiện mô hình hóa mạch với mô hình quá trình trễ tín hiệu với thông số thời gian trễ thuần nhất, ta sẽ nhận được sơ đồ hoạt động theo thời gian như trên hình 5.19. Khi thông số thời gian trễ của các phần tử B và E bằng 2, những phần tử còn lại nhận có thông số thời gian trễ bằng 1, ta có sơ đồ hoạt động của mạch theo thời gian thể hiện trên hình 5.20.

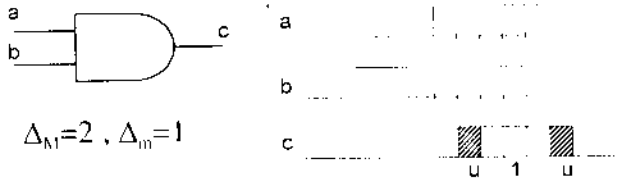
Khi mô hình hoá hành vi của các các phần tử lôgic, chúng ta thường phân tách chức năng tính toán lôgic và cơ chế điều phối thời gian của chúng.




**Hình 5.21** Mô hình phân tách chức năng lôgic và thuộc tính trễ.

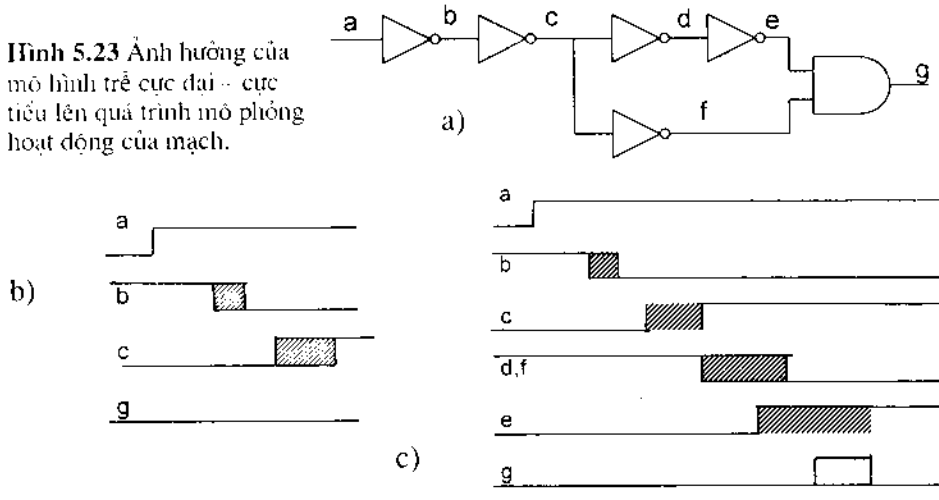
Như vậy, một phần tử với thông số thời gian trễ là $\Delta_T$ sẽ tương ứng với phần tử lôgic với thông số thời gian trễ bằng không và phần tử trễ có thông số thời gian trễ là $\Delta_T$. Trong quá trình mô hình hoá hành vi của phần tử lôgic, đầu tiên phần tử bị kích hoạt sẽ được tính toán theo chức năng lôgic, sau đó các thuộc tính trễ sẽ được tính toán và thể hiện qua quá trình điều phối thời gian.

Việc chỉ ra giá trị duy nhất của thông số thời gian trễ của phần tử rất phức tạp cả trên khía cạnh cấu trúc mạch và trên khía cạnh sản xuất. Ta có thể thực hiện đo đạc trực tiếp trên các mạch thực, nhưng đối với những mạch được thiết kế dưới dạng các mạch tích hợp VLSI thì khả năng đo trực tiếp là không thể có được. Trong trường hợp này để nâng cao tính chính xác của quá trình mô hình hóa và mô phỏng, ta chỉ ra khoảng biến thiên của thông số thời gian trễ. Mô hình quá trình trễ tín hiệu này được gọi là mô hình với thời gian trễ không xác định. Nếu trong mô hình với thời




**Hình 5.22** Sự xuất hiện giá trị không xác định 'u' trong hoạt động của phần tử AND với mô hình quá trình trễ tín hiệu với thời gian trễ cực đại - cực tiểu.

gian trễ không xác định, chúng ta chỉ rõ giá trị cực đại $\Delta_M$ và giá trị cực tiểu $\Delta_m$ của thông số thời gian trễ, mô hình sẽ được gọi là mô hình với thời gian trễ cực đại - cực tiểu. Ví dụ, thông số thời gian trễ của một loại phần tử NAND có thể biến thiên trong khoảng từ 5 ns đến 10 ns, như vậy đối với loại phần tử NAND này, $\Delta_m = 5$ ns và $\Delta_M = 10$ ns. Để khảo sát mô hình này, ta có thể xem xét hoạt động của phần tử AND có hai đầu vào ( hình 5.22 ). Trong ví dụ này, phần tử AND có giá trị thời gian trễ cực đại - cực tiểu: $\Delta_M = 2$, $\Delta_m = 1$. Ta thấy tín hiệu trên đường *c* chuyển từ trạng thái '0' sang trạng thái '1' khi tín hiệu trên đầu vào *a* chuyển từ '0' sang '1'. Ta thấy rằng trong khoảng thời gian trước $\Delta_m$ tính từ thời điểm xuất hiện sự kiện trên đường *a*, trên đường *c* không có sự thay đổi giá trị tín hiệu. Cũng tương tự, sau khoảng thời gian $\Delta_M$ tính từ thời điểm xuất hiện sự kiện trên đường *a*, trên đường *c* thiết lập giá trị '1'. Như vậy, trong khoảng thời gian từ $\Delta_m$ đến $\Delta_M$ sau khi xuất hiện sự kiện trên đường *a*, tín hiệu trên đường *c* là không xác định, ta sẽ dùng giá trị 'U' để biểu thị giá trị tín hiệu. Tóm lại, khi ta sử dụng mô hình trễ không xác định hoặc mô hình trễ cực đại – cực tiểu, trong hoạt động của mạch sẽ xuất hiện những hành vi không xác định và biểu thị qua những giá trị 'U'.



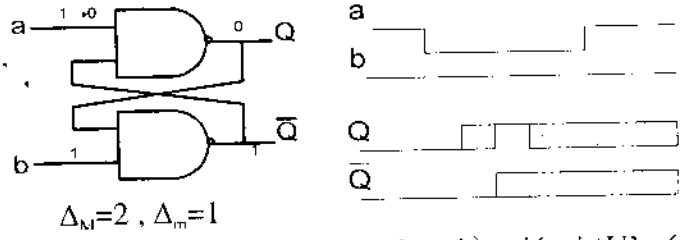

**Hình 5.23** Ảnh hưởng của mô hình trễ cực đại – cực tiểu lên quá trình mô phỏng hoạt động của mạch.

Với mô hình trễ cực đại – cực tiểu, ta có thể mô hình hoá phần lớn các trường hợp phức tạp xuất hiện trong hoạt động của mạch trên thực tế. Nhưng trong mô hình này nếu chúng ta giả thiết khoảng biến thiên của thời gian trễ lớn, trong quá trình mô hình hoá ta có thể truyền toàn các giá trị 'U' hoặc

trong mạch có thể xuất hiện những dạng tín hiệu không có trong sơ đồ thực. Để minh họa vấn đề này, chúng ta xét sơ đồ trên hình 5.23. Giả thiết rằng các phần tử mạch có thông số thời gian trễ $\Delta_M = 3$ và $\Delta_m = 2$. Theo sơ đồ lôgic ta phải có giá trị trên đường tín hiệu *g* luôn bằng '0'.

- Nếu bắt đầu quá trình mô phỏng từ đường tín hiệu *a*, ta nhận được giá trị trên đường tín hiệu *g* sẽ bằng 'U' ( theo hình 5.23.c ). Vì giá trị tín hiệu trên đường *g* luôn bằng '0', do đó sự xuất hiện giá trị 'U' có nghĩa là giá trị đó có thể là '1' hoặc '0'. Sự xuất hiện giá trị '1' trên đường *g* có nghĩa là trong mạch có thể xảy ra sự cố. Điều này được quá trình mô hình hoá dự đoán, nhưng trong thực tế không bao giờ xảy ra.
- Nếu bắt đầu quá trình mô phỏng từ đường tín hiệu *c*, ta sẽ nhận được g = 0 ( theo hình 5.23.b ), bởi vì độ trễ cực tiểu trên đường tín hiệu *c* – *d* – *e* có giá trị bằng 4, trong khi đó độ trễ cực đại trên đường *c* – *f* bằng 3. Như vậy, trên hai đường tín hiệu *e* và *f* không bao giờ có cùng giá trị '1' đồng thời và điều đó có nghĩa là đường *g* luôn nhận giá trị '0'.

Khi ta mô hình hoá mạch với mô hình trễ không xác định tương ứng với




**Hình 5.24** Hiện tượng mở rộng miền giá trị 'U' khi sử dụng mô hình trễ không xác định.

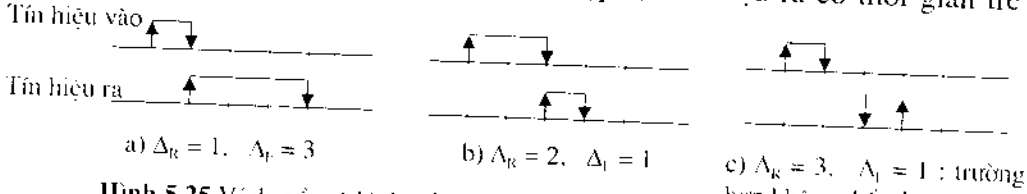
trường hợp xấu nhất, mỗi khi tín hiệu đi qua mạch, miền giá trị 'U' có thể sẽ bị mở rộng. Ta hãy xét hoạt động của mạch trigơ RS với hai phần tử NAND có cùng thông số thời gian trễ ( hình 5.24 ). Ta sử dụng mô hình trễ cực đại – cực tiểu với $\Delta_M = 2$ và $\Delta_m = 1$. Giả thiết, trạng thái ban đầu Q = '0' , $\bar{Q}$ = '1' và $a = b$ = '1'. Trong trường hợp khi giá trị trên đường *a* thay đổi từ '1'→ '0'. Khi *a* thay đổi, ở đầu ra Q xuất hiện giá trị 'U'. Giá trị 'U' này sẽ sinh ra giá trị 'U' tại đầu ra $\bar{Q}$. Việc tiếp tục quá trình tính toán với các giá trị 'U' dẫn tới tình trạng sau: bắt đầu từ một thời điểm nào đó, giá trị của Q và $\bar{Q}$ trở thành 'U'. Như vậy mô hình trễ không xác định không thể áp dụng được trong trường hợp này.

Trong các mạch số, việc tính giá trị thông số thời gian trễ sẽ phụ thuộc

vào hai vấn đề:

- Sự phụ thuộc của thời gian trễ vào hướng truyền tín hiệu;
- Độ chính xác của việc xác định thời gian trễ.

Trong một số thiết bị, thời gian trễ phụ thuộc vào hướng truyền của tín hiệu ra. Đối với những thiết bị như vậy, thời gian để tín hiệu ra tăng từ '0' → '1' và giảm từ '1' → '0' khác nhau khá xa. Ví dụ trong những mạch MOS, thời gian để thiết lập sườn xuống của tín hiệu ( thời gian trễ của sườn xuống $\Delta_F$ ) lớn hơn thời gian thiết lập sườn lên ( thời gian trễ của sườn lên $\Delta_R$ ) khoảng ba lần. Để mô tả những trạng thái này, ta đưa ra giá trị thời gian trễ của sườn lên (rise) và thời gian trễ của sườn xuống (fall) $\Delta_R$ và $\Delta_F$ đối với từng phần tử. Trong những trường hợp như vậy khi mô hình hoá hoạt động của mạch, độ dài của xung sẽ tăng. Thêm vào đó, đối với những giá trị thời gian trễ sườn lên và xuống khác nhau, có thể xuất hiện những trường hợp không có trên thực tế khi mô hình hoá mạch. Trường hợp này có thể thấy trên hình 5.25. Trong hình 5.25, trường hợp a), tín hiệu ra có thời gian trễ




a) $\Delta_R = 1$, $\Delta_F = 3$ b) $\Delta_R = 2$, $\Delta_F = 1$ c) $\Delta_R = 3$, $\Delta_F = 1$ : trường hợp không thể xảy ra.

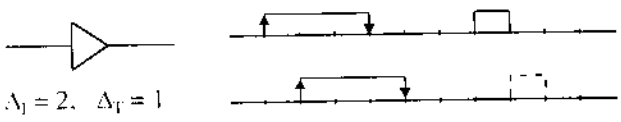
**Hình 5.25** Ví dụ về mô hình trễ sườn lên và sườn xuống.

sườn xuống $\Delta_F$ bằng ba còn thời gian trễ sườn lên bằng một, trong trường hợp này độ dài của tín hiệu ra tăng. Trường hợp b), tín hiệu ra có thời gian trễ sườn xuống $\Delta_F$ bằng một còn thời gian trễ sườn lên $\Delta_R$ bằng hai, trong trường hợp này độ dài của tín hiệu ra giảm. Trong trường hợp c), tín hiệu ra có thời gian trễ sườn xuống $\Delta_F$ bằng một còn thời gian trễ sườn lên $\Delta_R$ bằng ba, tín hiệu ra có sườn xuống vượt trước sườn lên - điều không thể xảy ra trong thực tế. Những trường hợp như vậy phải được loại bỏ trong quá trình mô hình hoá và mô phỏng.

### b. Trễ quán tính

Mọi mạch điện để cần năng lượng để chuyển trạng thái. Năng lượng của tín hiệu là hàm số của biên độ và độ dài tín hiệu. Nếu độ dài của tín hiệu quá ngắn, tín hiệu không thể kích hoạt để phần tử chuyển trạng thái. Độ dài tối thiểu của tín hiệu đầu vào đủ để kích hoạt mạch chuyển trạng thái được gọi

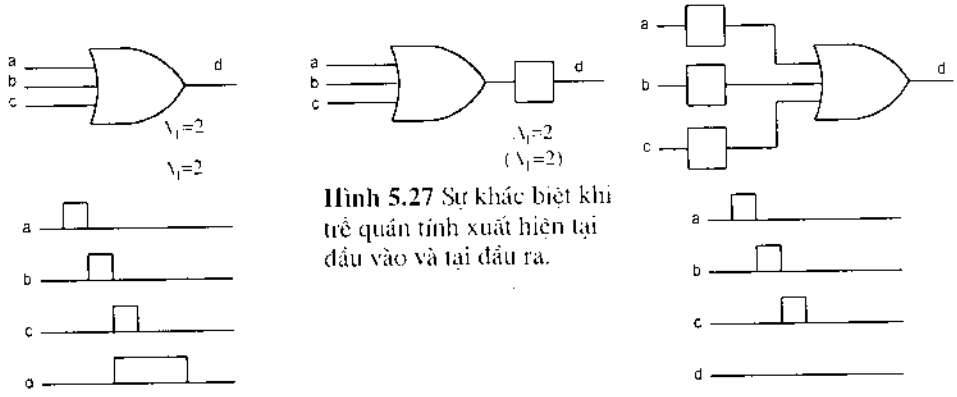
là thời gian trễ quán tính đầu vào của phần tử và ký hiệu là $\Delta_I$. Những tín hiệu có độ dài nhỏ hơn $\Delta_I$ sẽ được gọi là xung nhọn và sẽ không được phần tử cho đi qua. Nếu độ dài của tín hiệu vào lớn hơn hoặc bằng $\Delta_I$, tín hiệu sẽ được đi qua mạch với độ trễ bằng thời gian trễ lan truyền của phần tử. Việc đưa trễ quán tính đầu vào cho phép mô hình hoá những trường hợp đặc biệt trong các sơ đồ thực khi phần tử ngừng làm việc với những xung rất hẹp.




**Hình 5.26** Minh họa cho hiện tượng trễ quán tính đầu vào.

Hình 5.26 đưa ra minh họa trường hợp trễ quán tính đầu vào với xung có độ rộng bằng ba và xung có độ rộng bằng một. Phần tử có độ trễ quán tính đầu vào $\Delta_I = 2$ và trễ lan truyền $\Delta_T = 1$. Khi độ rộng của xung bằng một thì tín hiệu không tới được đầu ra.

Trong nhiều trường hợp, kết quả của việc mô hình hoá trễ quán tính phụ thuộc vào việc trễ quán tính xuất hiện ở đầu vào hoặc đầu ra của phần tử. Trễ quán tính xuất hiện ở đầu ra đặc trưng cho việc đầu ra của phần tử không thể đưa ra tín hiệu có độ dài nhỏ hơn $\Delta_I$. Xung tín hiệu tại đầu ra có thể được xung tín hiệu đầu vào sinh ra, nhưng cũng có thể bị loại bỏ nếu có độ dài

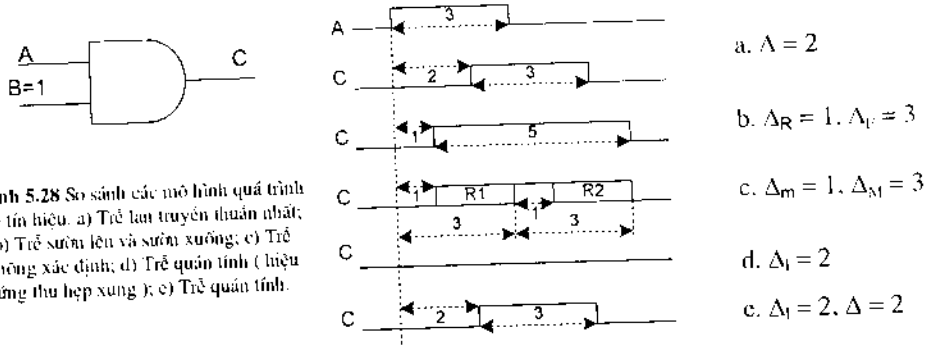



**Hình 5.27** Sự khác biệt khi trễ quán tính xuất hiện tại đầu vào và tại đầu ra.

quá nhỏ. Ta hãy xét phần tử OR ba đầu vào với thông số thời gian trễ lan truyền $\Delta_T = 2$ và $\Delta_I = 2$. Các tín hiệu đầu vào đều có độ dài bằng 1 và tác động lần lượt với độ trễ bằng 1. Trong trường hợp trễ quán tính được đặt tại đầu ra, ba tín hiệu đầu vào lần lượt sinh ra các xung giá trị '1' có độ dài bằng 1 liên tiếp. Các xung này hợp thành tín hiệu giá trị '1' có độ dài bằng 3

tại đầu ra. Tín hiệu này sẽ xuất hiện trên đầu ra vì có độ dài lớn hơn thông số trễ quán tính đầu ra ( $3 > 2 = \Delta_I$ ). Trong trường hợp thứ hai khi trễ quán tính được đặt tại đầu vào, ba tín hiệu đầu vào đều có độ dài nhỏ hơn thông số trễ quán tính do đó không thể kích hoạt phần tử. Điều này dẫn tới việc tín hiệu đầu ra không được hình thành. Như vậy việc mô hình hoá quá trình trễ quán tính gắn liền với các đặc tính vật lý của phần tử và cần thận trọng khi nghiên cứu. Hình 5.28 cho ta thấy những ảnh hưởng khác nhau của các mô hình trễ tới việc hình thành tín hiệu đầu ra đối với phần tử AND hai đầu vào.

Để quá trình mô hình hoá mạch được chính xác ta cần phải nghiên cứu tổ hợp những dạng khác nhau của thời gian trễ. Nhưng điều này làm tăng thời



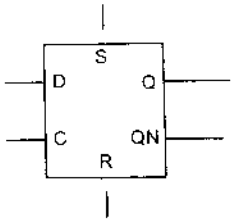

**Hình 5.28** So sánh các mô hình quá trình trễ tín hiệu. a) Trễ lan truyền thuần nhất; b) Trễ sườn lên và sườn xuống; c) Trễ không xác định; d) Trễ quán tính ( hiệu ứng thu hẹp xung ); e) Trễ quán tính.

gian thực hiện mô hình hoá và mô phỏng, điều này không thể chấp nhận được trong kỹ thuật. Như vậy, để thực hiện mô hình hoá quá trình trễ tín hiệu và không làm ảnh hưởng tới thời gian mô phỏng ta có thể sử dụng trình tự sau:

- Kiểm nghiệm lôgic của mạch sử dụng mô hình chỉ gồm các trễ thuần nhất;
- Thực hiện mô hình hoá và mô phỏng tính tới sự khác biệt của thời gian trễ sườn lên và sườn xuống;
- Sử dụng mô hình trễ không xác định trong quá trình mô hình hoá và mô phỏng;
- Thực hiện quá trình mô hình hoá sử dụng các phương pháp thống kê đối với những tổ hợp khác nhau của quá trình trễ tương ứng.

## 2. Mô hình hóa quá trình trễ tín hiệu qua các phần tử chức năng và thanh ghi

Các chức năng lôgic và các đặc tính thời gian của các phần tử chức năng phức tạp hơn so với các phần tử lôgic cơ bản. Ta hãy xét ví dụ mô tả hành vi




| q | S | R | C | D | Q | QN | TgTrễ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | x | x | 1 | 0 | $\Delta_{S/Q} = 4$; $\Delta_{S/QN} = 3$ |
| 1 | 1 | 0 | x | x | 0 | 1 | $\Delta_{R/Q} = 3$; $\Delta_{R/QN} = 4$ |
| 1 | 1 | 1 | ↑ | 0 | 0 | 1 | $\Delta^f_{C/Q} = 8$; $\Delta^r_{C/QN} = 6$ |
| 0 | 1 | 1 | ↑ | 1 | 1 | 0 | $\Delta^r_{C/Q} = 6$; $\Delta^f_{C/QN} = 8$ |
| x | 0 | 0 | x | x | u | u | |

**Hình 5.29** Minh hoạ độ trễ vào/ra đối với phần tử chức năng trigơ D.

hoạt động của phần tử trigơ D làm việc theo sườn lên với hai đường tín hiệu không đồng bộ thiết lập S ( set ) và khởi tạo R ( reset ) ( hình 5.29 ). Trong bảng trên hình 5.29, ký hiệu "$\Delta_{I/O}$" chỉ độ trễ của đáp ứng tại đầu ra 'O' đối với tác động tới đầu vào 'I'; các ký hiệu $\Delta^r$ và $\Delta^f$ chỉ độ trễ sườn lên và sườn xuống của phần tử. Ta xét một dòng của bảng, ví dụ dòng thứ ba. Dòng này chỉ ra rằng: nếu trạng thái ban đầu của phần tử $q$ = '1' và tại các đường tín hiệu S, R không có tác động ( S = '1', R = '1' ), sự chuyển trạng thái của xung đồng hồ C từ '0' sang '1' sẽ làm cho đầu ra Q nhận giá trị của đường D với thời gian trễ sườn xuống $\Delta^f_{C/Q} = 8$; cũng tương tự như vậy, đầu ra QN chuyển từ '0' sang '1' với độ trễ $\Delta^r_{C/QN} = 6$. Dòng cuối cùng của bảng chỉ ra rằng, nếu các đầu vào R và S nhận giá trị cấm "00", cả hai đầu ra sẽ cùng nhận giá trị 'U'.

Cũng tương tự như đối với phần tử lôgic, khái niệm độ trễ quán tính đầu vào có thể mở rộng cho phần tử trigơ nếu ta đưa vào các thông số chỉ độ rộng cực tiểu của xung tác động đối với các đầu vào C, S, R. Những đường tín hiệu này có tác dụng kích hoạt quá trình chuyển trạng thái của trigơ.

Đối với phần tử trigơ D nói trên, trong nhiều trường hợp, ta cần đưa ra những yêu cầu về độ dài của tín hiệu để tránh hiện tượng chạy đua giữa các đường tín hiệu C và D. Hai thông số thời gian khởi tạo (setup time) và thời gian tồn tại (hold time) là những thông số chỉ khoảng thời gian nhỏ nhất trước và sau thời điểm chuyển trạng thái trên đường C trong đó giá trị trên

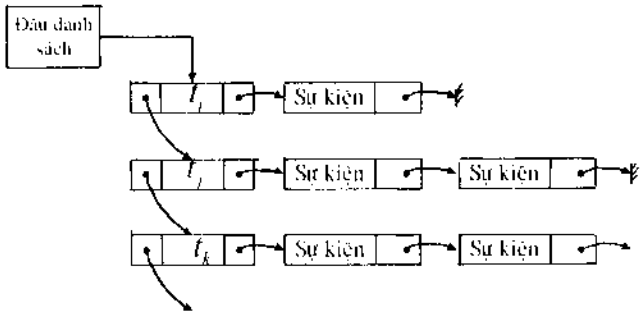
đường tín hiệu D phải ổn định ( không thay đổi ) để trigơ hoạt động chính xác. Các quá trình mô hình hoá lôgic và mô phỏng phải phản ánh được các yêu cầu này.

Đối với những phần tử phức tạp hơn như các thanh ghi, quá trình trễ tín hiệu được mô tả ít chi tiết hơn. Trong nhiều trường hợp, các mô hình trên mức thanh ghi sử dụng các chu trình thời gian lặp và nếu trong đó có xác định các thông số trễ, các thông số này sẽ được coi là độ trễ lan truyền.

## §5.5. Mô hình hóa trên mức các phần tử lôgic

Các phương pháp điều khiển quá trình thể hiện trình tự thực hiện mô hình hoá và những phương pháp xử lý gắn liền với quá trình mô phỏng mạch. Trong phương pháp biên dịch, trình tự thực hiện mô hình hoá được xác định bằng việc phân hạng các phần tử. Như vậy, việc ngắt các vòng phản hồi trong những mạch tuần tự và phân hạng phần tử chính là phương pháp điều khiển quá trình mô hình hoá. Đó chính là nguyên nhân dẫn tới việc không thể sử dụng các mô hình trễ trừ mô hình với độ trễ bằng không. Trong mục này chúng ta nghiên cứu các phương pháp điều khiển trong quá trình mô hình hoá hướng sự kiện như một phương tiện mô hình hoá mạch với độ trễ khác không.

Khi sử dụng phương pháp mô hình hoá hướng sự kiện nếu giá trị tín hiệu $v'(j)$ tại thời điểm thời gian $t_0$ khác giá trị $v(j)$ tại thời điểm trước, điều đó có nghĩa là xuất hiện sự kiện, ta sẽ coi rằng tín hiệu thay đổi giá trị vào thời điểm $t_0 + \Delta_T$ , trong đó $\Delta_T$ là độ trễ lan truyền của phần tử đang xét. Để mô tả sự xuất hiện sự kiện trên các đường tín hiệu trong mạch theo thời

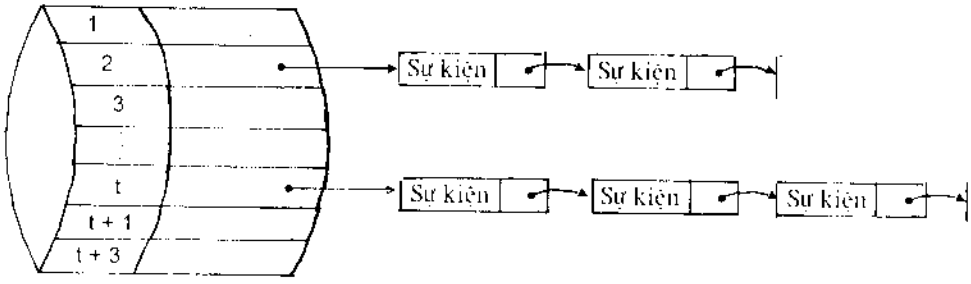



**Hình 5.30** Minh hoạ danh sách sự kiện.

gian ta chứa các sự kiện vào danh sách tuyến tính có dạng biểu diễn trên hình 5.30 và gọi là danh sách sự kiện. Tương ứng với mỗi một thời điểm thời gian , ta sẽ có một danh sách các sự kiện xuất hiện vào thời điểm đó.

Khi có một sự kiện mới xuất hiện, sự kiện này được đưa vào danh sách sự kiện gắn liền với một thời điểm thời gian xác định. Ta lưu ý một điều như sau, sự xuất hiện của một thời điểm thời gian được đánh dấu trong quá trình mô hình hoá không phụ thuộc vào chuỗi thời gian. Do đó, để xác định vị trí của các thời điểm đã được tính trước trong danh sách sự kiện, chúng ta cần phải có chuỗi thời gian tương ứng với danh sách được xử lý. Điều này có thể được thực hiện dựa vào các ánh xạ thời gian.

Khi ta sử dụng phép ánh xạ thời gian, dãy các thời điểm thời gian được xác định với khoảng thời gian cách đều Δ, mọi sự kiện trong hệ thống sẽ được xác định dựa vào các thời điểm thời gian này. Khoảng thời gian Δ được xác định bằng ước số chung lớn nhất của của thời gian truyền tín hiệu và độ trễ của các phần tử trong mạch. Do giới hạn về bộ nhớ dùng để thực hiện quá




**Hình 5.31** Vòng các thời điểm thời gian mô phỏng.

trình mô hình hoá mạch, ta xác định giá trị giới hạn của thời điểm thời gian cực đại bằng M. Như vậy, các thời điểm thời gian sau thời điểm cực đại sẽ quay lại bắt đầu từ thời điểm ban đầu và tạo thành vòng quay thời gian (bánh xe thời gian).

Thông thường, khoảng thời gian Δ tương thích với độ trễ của các phần tử trong mạch. Trong những trường hợp khi trong mạch có những phần tử với độ trễ lớn, khoảng thời gian Δ có thể vượt quá thời gian giới hạn M, điều này dẫn tới việc thời điểm mô phỏng vượt ra ngoài vòng thời gian và gây nên hiện tượng tràn. Để giải quyết vấn đề này, ta xây dựng danh sách bổ trợ để lưu trữ các thời điểm tràn và gọi là danh sách tràn. Trong trường hợp sử dụng danh sách tràn, mỗi khi bánh xe thời gian thực hiện được một vòng quay, ta cần thực hiện thao tác trả các thời điểm trong danh sách tràn về chu trình xử lý theo vòng thời gian.

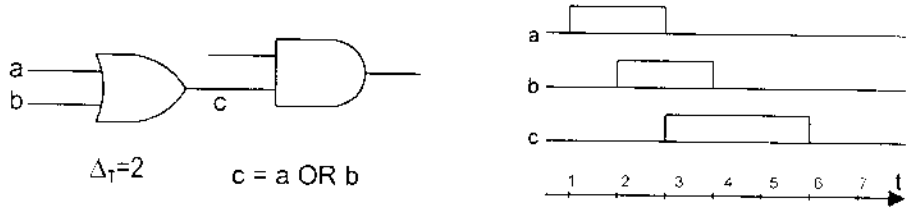
Khi thực hiện quá trình mô hình hoá theo vòng thời gian, các bước của quá trình mô phỏng có thể được biểu diễn dưới dạng sau:

- B1: Thiết lập các giá trị ban đầu của tín hiệu tại thời điểm $t \leftarrow t_0$ và đặt các giá trị đó thành giá trị hiện thời.

  B2: Đọc các giá trị đầu vào. Nếu giá trị này khác các giá trị hiện thời, ta đưa chúng vào tập hợp $L_\tau$ trong đó $\tau = t + \Delta_T$.

  B3: Nếu $L_\tau = \varnothing$, ta chuyển tới bước B5, trong trường hợp ngược lại, ta chuyển tới bước B4.
- B4: Nếu giá trị trong $L_\tau$ trùng với giá trị hiện thời, ta giữ nguyên giá trị hiện thời; nếu các giá trị này khác nhau, ta làm cập nhật giá trị hiện thời và đưa vào $L_\tau$, $\tau = t + \Delta_T$.
- Gán $t \leftarrow t + l$ và quay về bước B2..

Ta hãy xét một ví dụ minh họa thuật toán mô hình hoá hướng sự kiện đối với mạch biểu diễn trên hình 5.32. Các giá trị '1' và '0' trên một đường tín hiệu bất kỳ, ví dụ là trên đường *a*, sẽ được ký hiệu là ( a; 1 ) và ( a; 0 ). Giả thiết rằng các tín hiệu đầu vào thay đổi theo giản đồ thời gian trên hình 5.32, phần tử OR có độ trễ lan truyền $\Delta_T = 2$. Trong trường hợp này quá trình mô hình hoá hướng sự kiện sẽ được thực hiện theo trình tự dưới đây.




**Hình 5.32** Minh họa thuật toán mô hình hoá hướng sự kiện.

1. B1: Tại thời điểm ban đầu $t = t_0$ ( $t_0 = 1$ ); Các giá trị ban đầu: (a;0), (b;0), (c;0);
2. B2: Đọc giá trị đầu vào : ( a; 1 ); $L_{\tau+\Delta T} = L_3 = \{ ( c; 1 ) \}$;
3. B3: Kiểm tra $L_1$, $L_1 = \varnothing$, chuyển tới bước B5;
4. B5: $t \leftarrow t + 1$ ( $t = 2$ ); Quay lại bước B2;
5. B2: Đọc giá trị đầu vào ( b; 1 ); $L_{\tau+\Delta T} = L_4 = \{ ( c; 1 ) \}$;
6. B3: Kiểm tra $L_2$, $L_2 = \varnothing$ , chuyển tới bước B5;
7. B5: $t \leftarrow t + 1$ ( $t = 3$ ); Quay lại bước B2;
8. B2: Đọc giá trị đầu vào ( a; 0 );
9. B3: Kiểm tra $L_3$, $L_3 = \{ ( c; 1 ) \}$, $L_3 \neq \varnothing$; Thực hiện bước B4;
10. B4: Giá trị hiện thời được thiết lập bằng ( c; 1 );

11. B5: t ← t + 1 ( t = 4 ); Quay lại bước B2;
12. B2: Giá trị đầu vào ( b; 0 ); $L_6$ = { ( c; 0 ) };
13. B3: Kiểm tra $L_4$, $L_4$ = { ( c; 1 ) }, $L_4 \neq \varnothing$; Thực hiện bước B4;
14. B4: Do $L_4$ trùng với giá trị hiện thời ( c; 1 ), thực hiện bước B5;
15. B5: t ← t + 1 ( t = 5 ); Quay lại bước B2;
16. B2: Các tín hiệu vào không thay đổi giá trị; Thực hiện B3;
17. B3: Kiểm tra $L_5$, $L_5 = \varnothing$, chuyển tới bước B5;
18. B5: t ← t + 1 ( t = 6 );
19. B2: Các tín hiệu vào không thay đổi giá trị; Thực hiện B3;
20. B3: Kiểm tra $L_6$, $L_6$ = { ( c; 0 ) }, $L_6 \neq \varnothing$ . Thực hiện bước B4;
21. B4: Giá trị hiện thời được thiết lập bằng ( c; 0 );

Cuối cùng, ta nhận được giản đồ thời gian như trên hình 5.32.

Trong trường hợp khi quá trình phân tích và mô hình hóa mạch có sử dụng các mô hình trễ không xác định, để thực hiện các tính toán cho việc xây dựng danh sách sự kiện, ta cần phải hình thành được các sự kiện xuất hiện trong khoảng thời gian giữa các giá trị $\Delta_m$ và $\Delta_M$. Như trong ví dụ đối với mạch trên hình 5.32, nếu phần tử OR có độ trễ không xác định với giá trị cực tiểu $\Delta_m = 2$ và giá trị cực đại $\Delta_M = 3$, khi đó bước thứ hai trong trật tự tính toán nêu trên sẽ được ghi như sau:

$$L_3 = \{ ( c; U ) \}, L_4 = \{ ( c; 1 ) \};$$

Sau đó quá trình mô hình hóa và mô phỏng hướng sự kiện sẽ được thực hiện như bình thường chỉ cần tính đến giá trị 'U'. Thêm vào đó, do trong quá trình mô hình hóa ta phải liên tục tạo thêm những danh sách sự kiện mới, thời gian thực hiện mô hình hóa sẽ tăng lên khi phải xử lý các danh sách sự kiện.

Nếu trong quá trình phân tích và mô hình hóa mạch cần phải sử dụng mô hình với trễ quán tính, ta cần phải biến đổi thuật toán và thêm vào đó những thao tác tính toán tương ứng. Hơn nữa, do việc cần thiết phải sử dụng lại những sự kiện đã được xử lý, ta cần phải tăng chi phí về mặt bộ nhớ và thời gian tính toán.

Để nâng cao độ chính xác của quá trình phân tích mạch theo thời gian và mô hình hóa hướng sự kiện, chúng ta phải thu nhỏ độ lớn của lượng tử thời gian Δ. Điều này sẽ dẫn đến việc tăng dung lượng bộ nhớ để lưu trữ các sự kiện trong trường hợp ta giữ cận trên về thời gian mô phỏng M không đổi. Để giải quyết vấn đề này ta phải đưa ra những biến thể thích hợp của phương pháp ánh xạ thời gian và phương pháp danh sách sự kiện.

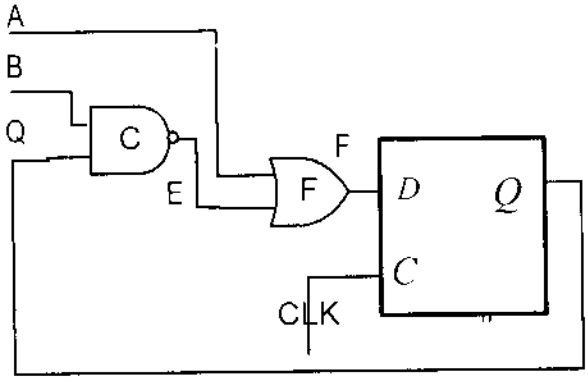
Trên thực tế, một trong những phương pháp để tăng tốc độ cho quá trình mô hình hoá và mô phỏng lôgic là phương pháp mô hình hoá hướng sự kiện. Việc tính toán đối với các phần tử lôgic chỉ được thực hiện khi giá trị tín hiệu khác với những giá trị đã được tính. Ưu điểm của phương pháp mô hình hoá hướng sự kiện là quá trình tính toán chỉ thực hiện khi có các sự kiện xuất hiện trong mạch, trong khi đó khi các giá trị đầu vào thay đổi, trung bình chỉ có 2,5% các tín hiệu trong mạch thay đổi giá trị. Điều đó làm tăng một cách đáng kể tốc độ quá trình mô hình hoá nói chung.

Khi thực hiện mô hình hoá trên mức các phần tử lôgic, ta có thể xác định một cách dễ dàng sự thay đổi giá trị trên các đường tín hiệu ra của các phần tử dựa vào các giá trị đầu vào của phần tử đó. Ví dụ, nếu trên một trong những đầu vào của phần tử AND giá trị tín hiệu không đổi và bằng '0', khi đó giá trị tín hiệu trên đầu ra sẽ không thay đổi không phụ thuộc vào sự thay đổi của giá trị tín hiệu trên các đầu vào khác của phần tử. Đối với phần tử OR, giá trị không đổi bằng '1' trên một trong những đầu vào sẽ đảm bảo không có sự thay đổi giá trị tín hiệu trên đầu ra. Nếu chú ý tới những tính chất của phần tử lôgic, ta có thể đơn giản hoá việc tính toán trong quá trình mô hình hoá mỗi khi có các sự kiện xuất hiện trong mạch.

Như vậy, trong chương này chúng ta đã nghiên cứu những vấn đề liên quan tới quá trình mô hình hoá và mô phỏng mạch. Những vấn đề quan trọng là thể hiện được hoạt động của mạch theo thời gian và hành vi làm trễ tín hiệu của các phần tử lôgic trong mạch.

**Bài tập cho chương 5**

1. Hãy sử dụng phương pháp mô phỏng bằng biên dịch 3 giá trị mô tả hoạt động của mạch trên hình B5.1. Mạch hoạt động với trạng thái ban đầu Q = u đối với vectơ giá trị đầu vào: "01" và "11".




**Hình B5.1**

2. Hãy sử dụng phương pháp mô hình hoá bằng biên dịch mô tả hoạt động mạch trên hình 5.15a đối với các vectơ đầu vào "00" và "11". Giả thiết rằng trong mô hình, hai đường Q và NQ đều được coi là các đường tín hiệu phản hồi với độ trễ bằng nhau.

3. Hãy lập chương trình trên ngôn ngữ lập trình truyền thống ( Pascal, C, C++, ...) mô tả cấu trúc và chức năng của phần tử trigơ RS master-slave theo phương pháp hướng sự kiện.

4. Hãy lập chương trình trên ngôn ngữ lập trình truyền thống ( Pascal, C, C++, ...) mô tả cấu trúc và chức năng của thanh ghi dịch 4 bit theo phương pháp hướng sự kiện.

5. Hãy lập chương trình trên ngôn ngữ lập trình truyền thống ( Pascal, C, C++, ...) mô tả cấu trúc và chức năng của bộ cộng 4 bit theo phương pháp hướng sự kiện.

# CHƯƠNG VI. NGÔN NGỮ MÔ HÌNH HÓA VHDL

## §6.1. Mở đầu ngôn ngữ VHDL

### 1. Những khái niệm chung về ngôn ngữ VHDL

Các phương pháp thiết kế dựa trên cơ sở của các ngôn ngữ HDL ngày càng trở nên phổ biến. Các ngôn ngữ mô tả phần cứng HDL được các nhà thiết kế mạch sử dụng chủ yếu để mô tả cấu trúc hoặc hành vi của các hệ thống số cho quá trình mô phỏng hoặc thiết kế.

Phương pháp thiết kế trên cơ sở các ngôn ngữ HDL so với các phương pháp thiết kế truyền thống trên cơ sở của các cổng lôgic có các ưu điểm sau:

- Các phương pháp này cho phép tăng năng suất thiết kế, nó cho phép nhà thiết kế tốn ít thời gian hơn và cho phép những người không cần kiến thức sâu về phần cứng có thể thiết kế phần cứng.
- Phương pháp thiết kế dựa trên các ngôn ngữ HDL khá cơ động với những công nghệ khác nhau. Các mô tả trên các ngôn ngữ HDL cung cấp các tài liệu độc lập với phần cứng của mạch điện. Sử dụng các chương trình tiện ích hỗ trợ thiết kế ta có thể chuyển đổi các biểu diễn trên các ngôn ngữ HDL thành nhiều mức ứng dụng cho những công nghệ khác nhau.

Ngôn ngữ VHDL ( VHSIC Hardware Description Language ) là ngôn ngữ được sử dụng phổ biến trong công nghệ chế tạo các mạch VLSI. VHDL được công nhận là ngôn ngữ tiêu chuẩn trong mô tả phần cứng của IEEE và của Bộ Quốc phòng Mỹ.

Về mặt cú pháp, ngôn ngữ VHDL là một ngôn ngữ được định kiểu chặt chẽ và có một tập hợp lớn các câu lệnh. Ngôn ngữ VHDL hỗ trợ các phương pháp mô tả nhiều lớp trong đó các thành phần cấu trúc hoặc mạng lưới các phần tử có thể đi đôi với các mô tả hành vi hoặc các thuật toán.

VHDL cung cấp khả năng mô tả của mạch số trên những mức độ trừu tượng khác nhau: mức thuật toán; mức các thanh ghi, hàm truyền đạt; mức các cổng lôgic. Nhà thiết kế có thể sử dụng chiến lược thiết kế từ trên xuống, đầu tiên mô tả thiết kế trên mức kiến trúc, sau đó chi tiết hoá từng bước thiết kế. Ví dụ ta có thể mô tả mạch so sánh một bit bằng ngôn ngữ VHDL theo

những mức độ chi tiết khác nhau. Mạch này bao gồm hai đầu vào và một đầu ra với các tín hiệu tại các đầu này là các tín hiệu số. Như vậy mạch sẽ tương ứng với một thực thể có hai đầu vào và một đầu ra. Kiến trúc của mạch đặc tả quan hệ giữa đầu vào và đầu ra của mạch và có thể mô tả theo hành vi , dòng truyền dữ liệu qua mạch, hoặc theo cấu trúc của mạch.

- **Mô tả trên mức thực thể**
  Trên mức thực thể, ta mô tả về số lượng các cổng vào ra của mạch và các dạng tín hiệu tại các cổng đó. Trong ví dụ về mạch so sánh một bit, mạch được mô tả bằng một thực thể bao gồm hai cổng vào và một cổng ra. Các dữ liệu tại các cổng này là các bit.

```
entity COMPARE is
        port ( A, B: in BIT; C: out BIT );
end COMPARE;
```

- **Mô tả mạch bằng hành vi**
  Để mô tả mạch bằng hành vi, trong ngôn ngữ VHDL người ta dùng cấu trúc process. Khi mô tả bằng hành vi, ta không cần thiết phải cung cấp chi tiết về việc thực hiện thiết kế. Trong ví dụ này ta thấy mạch so sánh được mô tả bằng một quá trình process. Quá trình này chịu tác động của hai tín hiệu là A và B.

```
architechture BEHAVIOR of COMPARE is
begin
        process ( A, B )
        begin
                if ( A = B ) then C <= '1';
                else C <= '0';
                end if
        end process;
end BEHAVIOR;
```

- **Mô tả bằng dòng dữ liệu**
  Theo biểu diễn bằng dòng dữ liệu, hệ thống được biểu diễn như các luồng chuyển động của các tín hiệu điều khiển và các dữ liệu. Theo

phương pháp này, hoạt động của mạch được biểu diễn như các mạch lôgic tổ hợp, như mạch cộng, mạch so sánh, mạch giải mã. Đối với ví dụ về mạch so sánh, ta thấy tín hiệu C được gán giá trị của biểu thức lôgic của A và B sau một thời gian 10 ns sau khi có sự thay đổi giá trị tín hiệu trên các cổng A, B.

```
architechture DATAFLOW of COMPARE is
begin
        C <= not ( A xor B ) after 10 ns;
end DATAFLOW;
```

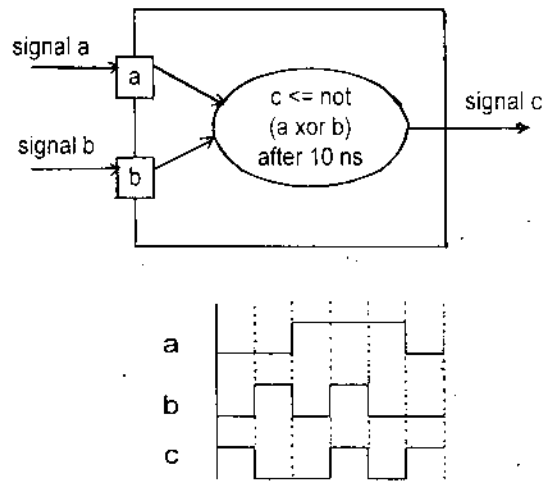
- **Mô tả mạch bằng cấu trúc**

Đối với cách mô tả mạch qua cấu trúc, ta phải mô tả các thành phần cấu trúc và mối liên kết giữa các thành phần đó. Trong ví dụ về mạch so sánh cấu trúc của mạch gồm một phần tử XOR có hai đầu vào I0, I1, kết nối với một phần tử NOT. Đầu ra của phần tử XOR nối với đầu vào của phần tử NOT. Cấu trúc kết nối trên được mô tả bằng đoạn chương trình VHDL.

```
architechture STRUCTURE of COMPARE is
component XOR_Gate
        port ( I0, I1 : in BIT; O : out BIT );
end component;
component NOT_Gate
        port ( I0 : in BIT; O: out BIT );
end component;
signal NET_I : BIT;
begin
U0: XOR_Gate port map ( I0 => A, I1 => B, O => NET_I );
U1: NOT_Gate port map ( I0 => NET_I, O => C );
end STRUCTURE;
```

## 2. Quá trình mô phỏng

Như đã đề cập tới trong chương 4, các ngôn ngữ HDL luôn đi kèm với các bộ mô phỏng. Các thiết kế trên ngôn ngữ VHDL sẽ thực hiện trên bộ mô phỏng VHDL để kiểm tra hành vi của hệ thống được mô hình hoá. Để tái tạo hoạt động của mô hình, nhà thiết kế cần cung cấp tập hợp các tác động vào mô hình. Chương trình mô phỏng sẽ gắn các tác động đó tới đầu vào của mô hình tại những thời điểm thời gian xác định và dựa vào mô hình đưa ra các đáp ứng của mạch. Các kết quả đó được nhà thiết kế sử dụng để kiểm tra mức độ thích hợp của thiết kế.




**Hình 6.1** Mô hình bộ so sánh một bit và các tín hiệu qua mô phỏng.

Ta có thể thực hiện mô phỏng trên bất kỳ giai đoạn nào của quá trình thiết kế. Tại các mức bao quát nhất của thiết kế, mô phỏng cung cấp cho ta thông tin về hoạt động của mạch đang thiết kế. Thông thường mô phỏng ở mức này thực hiện rất nhanh và không cung cấp cho ta những thông tin chi tiết về hoạt động của mạch và chế độ định thời gian. Khi quá trình thiết kế đi xuống các mức thấp hơn, mô phỏng sẽ chiếm thời gian dài hơn. Mô phỏng ở các mức thấp chiếm nhiều thời gian hơn nhưng cung cấp cho ta nhiều thông tin chi tiết về hoạt động của mạch, chế độ thời gian đồng hồ. Ngôn ngữ VHDL cho phép sử dụng cơ chế mô phỏng hỗn hợp, trong đó một số thành phần có thể được mô tả ở mức cao còn một số thành phần khác có thể được mô tả chi tiết. Ưu điểm của phương pháp mô phỏng hỗn hợp là cho phép nhà thiết kế tập trung vào xây dựng những phần mạch quan trọng. Để giảm giá thành thực hiện mô phỏng ở các mức thấp, các bộ mô phỏng cần được dùng để phát hiện các lỗi thiết kế càng sớm càng tốt. Trong quá trình mô phỏng

chương trình VHDL, các nhà thiết kế cần cung cấp tập hợp các giá trị thử nghiệm tại những thời điểm mô phỏng xác định.

## 3. Tổng hợp mạch

Tổng hợp mạch là quá trình xây dựng các mô tả thiết kế từ một mức trừu tượng nào đó sang một mức trừu tượng thấp hơn. Quá trình này có thể là sự biến đổi từ hành vi này sang hành vi khác hoặc từ hành vi sang cấu trúc. Quá trình biến đổi này tương tự như quá trình biên dịch chương trình phần mềm viết trên các ngôn ngữ bậc cao sang các mã assembly. Các đầu vào của các công cụ hỗ trợ thiết kế thường là các mô tả trên các ngôn ngữ HDL, các cơ chế điều khiển thời gian, các mục tiêu tối ưu, các thư viện kỹ thuật. Đầu ra của hệ thống hỗ trợ thiết kế là các danh sách mạng lưới đã tối ưu, các hiệu năng của mạch, diện tích của thiết kế được xây dựng. Sau đây chúng ta mô tả ngắn gọn về quá trình tổng hợp hành vi, tổng hợp ở mức thanh ghi truyền đạt và tổng hợp lôgic.

- Tổng hợp hành vi là quá trình biến đổi các mô tả bằng những ngôn ngữ thủ tục sang các mô tả ở mức thanh ghi truyền đạt. Thiết kế ở mức thanh ghi truyền đạt thường bao gồm các đường truyền dữ liệu, các mạch nhớ và các bộ điều khiển. Quá trình tổng hợp hành vi thường được gọi là quá trình tổng hợp ở mức cao hay còn gọi là tổng hợp kiến trúc. Bước này thường bao gồm tổng hợp các đường dữ liệu, tổng hợp các mạch nhớ và tổng hợp các mạch điều khiển.
- Tổng hợp ở mức thanh ghi truyền đạt là quá trình tạo ra cấu trúc mạng cho các mạch tuần tự từ tập hợp các hàm truyền đạt thanh ghi. Các trạng thái tương ứng với chế độ đồng hồ cũng được xác định ở mức này. Các thao tác ở mức thanh ghi truyền đạt có thể mô tả bằng các ôtômat hữu hạn hoặc tập hợp các phương trình truyền đạt ở mức thanh ghi. Các thành phần của quá trình bao gồm: tối ưu hoá trạng thái, mã hoá trạng thái, tối ưu hoá lôgic, ánh xạ công nghệ.
- Tổng hợp lôgic là quá trình chuyển các biểu diễn mạch bằng các biểu thức lôgic sang các mạch lôgic. Quá trình tối ưu hoá mạch lôgic thiết kế được chia làm hai giai đoạn: các quá trình tối ưu mạch không phụ thuộc vào công nghệ; và các ánh xạ các sơ đồ

và các phần tử vào các liên kết của các phần tử mô tả trong bộ thư viện.

Như vậy chúng ta đã thấy những khái niệm chung về ngôn ngữ VHDL và các giai đoạn trong quá trình thiết kế sử dụng các ngôn ngữ mô tả phần cứng.

## §6.2. Các cấu trúc cơ sở trong VHDL

Mỗi một hệ thống mạch số được thiết kế như một hệ phân cấp các môđun. Mỗi môđun tương ứng với một thực thể của thiết kế trên ngôn ngữ VHDL. Thực thể thiết kế thể hiện một đối tượng của thiết kế phần cứng. Đối tượng này có các đầu vào và đầu ra được xác định rõ ràng, đồng thời thực thể thiết kế cũng phải chỉ ra được chức năng của đối tượng thông qua các phép toán được định nghĩa trước. Mỗi thực thể thiết kế gồm có hai phần: phần khai báo thực thể và kiến trúc thực thể.

- Phần khai báo thực thể mô tả dạng bên ngoài của thực thể, các giao diện của thực thể với các thực thể khác. Giao diện này được thể hiện qua các cổng vào và cổng ra của thực thể.
- Kiến trúc thực thể mô tả những thành phần bên trong của thực thể.
- Ngoài ra chúng ta còn có thể dùng các gói tiện ích để hỗ trợ thiết kế. Các gói tiện ích xác định các thông tin chung của một số thực thể.
- Các thực thể còn có các cấu hình. Cấu hình là các dạng tương đương của thiết kế. Ví dụ như mạch nửa tổng có thể coi là một thực thể thiết kế. Thực thể này có thể được xây dựng trên các phần tử NAND hoặc trên các phần tử NOR. Khi đó ta nói thực thể này có hai cấu hình. Các cấu hình cho phép ta khảo sát các biến thể khác nhau của thiết kế và lựa chọn những bộ thiết bị tối ưu.
- Các thư viện là tập hợp những thực thể được mô tả sẵn. Các thực thể này được biên dịch và lưu trữ thường trực. Các thực thể trong thư viện sẽ được sử dụng tuỳ theo mức độ quan trọng trong thiết kế.

Các cấu trúc cơ sở của ngôn ngữ VHDL bao gồm:

- Các thực thể (entity)
- Các kiến trúc (architecture)
- Các gói (package)

- Các cấu hình (configuration)
- Các thư viện (**library**)

## 1. Mô tả các thực thể

Các khai báo thực thể cho ta cái nhìn đối với phần tử mạch cần được mô tả từ mặt bên ngoài. Bằng các khai báo thực thể, phần tử mạch sẽ được mô tả bằng số lượng và chức năng của các cổng giao tiếp và các tính chất của dữ liệu tại các cổng này theo phương diện từ ngoài vào. Các khai báo thực thể không cho chúng ta thấy cách thức xây dựng phần tử mạch.

Ngôn ngữ VHDL được mô tả theo những quy tắc cú pháp. Đối với các mô tả thực thể, cú pháp có dạng như sau.

**entity** *tên_thực_thể* **is**
[ *lệnh_khai_báo*_**generic** ]
[ *các_luật_tại_cổng* ]
{ *các_thành_phần_khai_báo_thực_thể* }
[ **begin**
*các_thành_phần_biểu_thức_thực_thể* ]
**end** [ *tên_thực_thể* ];

Lệnh khai báo **generic** dùng để khai báo các tham số được sử dụng để kiểm soát cấu trúc hoặc hành vi của thực thể. Các hằng số này gọi là các tham số chung. Các tham số chung có phạm vi sử dụng trong toàn bộ thực thể. Câu lệnh **generic** có cú pháp như sau.

**generic** (
*tên_hằng*: *kiểu_con* [ := *giá_trị_khởi_tạo* ]
{; *tên_hằng*: *kiểu_con* [ := *giá_trị_khởi_tạo* ]}
);

*tên_hằng*: là tên của tham số chung.
*kiểu*: kiểu dữ liệu của tham số.
*giá_trị_khởi_tạo*: giá trị khởi tạo của tham số.

Các luật cổng đặc tả cho các kết giao tiếp của thực thể và có quy tắc cú pháp như sau.

**port** ( *tên_cổng*: [ *mode* ] *kiểu_con* [ := *giá_trị_khởi_tạo* ]

{; *tên_cổng*: [ *mode* ] *kiểu_con* [ := *giá_trị_khởi_tạo* ]}
);

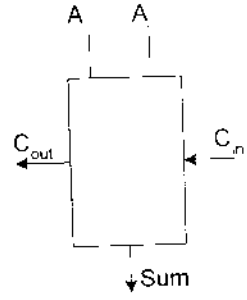
*tên_cổng*: tên của cổng được mô tả.
*mode*: chỉ hướng của tín hiệu tại cổng.
*kiểu_con*: kiểu dữ liệu tại cổng hoặc các tham số chung.
*giá_trị_khởi_tạo*: giá trị khởi tạo cho cổng.

Trong phần khai báo, thực thể và các cổng của thực thể luôn được đặt tên hoặc đánh định danh. Tên, định danh trong ngôn ngữ VHDL không phân biệt chữ hoa và thường. Một số định danh là các từ khóa của ngôn ngữ như: **entity**, **port**, **is**, **end**, ... Những từ này có ý nghĩa cố định và không thể thay đổi trong toàn bộ ngôn ngữ.

Các cổng là các tín hiệu kết nối thực thể với các thực thể khác Những tín hiệu tại cổng được đặt tương ứng với các dạng: **in**, **out**, **buffer**, **inout** và các kiểu dữ liệu. Ý nghĩa của các dạng cổng như sau.

- Cổng có dạng **in** là cổng chỉ dùng để đọc. Trong các mạch số các cổng dạng **in** chỉ được sử dụng làm cổng tín hiệu vào.
- Cổng dạng **out** là cổng chỉ để gán các giá trị. Trong thiết kế mạch, cổng dạng **out** chỉ sử dụng làm cổng tín hiệu ra.
- Cổng dạng **buffer** là cổng có thể cho phép cả hai thao tác đọc và gán dữ liệu. Nhưng trong từng ngữ cảnh cổng chỉ có thể nhận một trong hai chức năng đọc hoặc gán dữ liệu. Ví dụ về cổng dạng **buffer** là hai đường tín hiệu ra Q và $\overline{Q}$của RS trigơ. Đối với thực thể trigơ, hai đường tín hiệu này là hai tín hiệu ra. Nhưng đối với hai phần tử NAND hợp thành trigơ, các đường tín hiệu này là các tín hiệu vào. Trong các chương trình trên ngôn ngữ VHDL, các tín hiệu dạng **buffer** thường được sử dụng trong các trường hợp khi ta muốn đọc và ghi dữ liệu ở bên trong chương trình nhưng từ bên ngoài chương trình thì chỉ có thể đọc dữ liệu mà không thể ghi vào đường tín hiệu này.
- Cổng dạng **inout** là cổng có thể vừa đọc và vừa gán giá trị. Các cổng loại này cho phép có nhiều điều khiển dữ liệu đồng thời trong mọi ngữ cảnh. Các ví dụ về cổng dạng **inout** có thể là các tuyến dữ liệu vào/ra của bộ xử lý.

Trong phần *các_thành_phần_khai_báo_thực_thể* của thực thể chứa khai báo các hằng số, các kiểu hoặc tín hiệu có thể sử dụng trong quá trình xây dựng thực thể. Phần *các_thành_phần_biểu_thức_thực_thể* chứa các biểu thức thực hiện đồng thời. Các biểu thức này được sử dụng để kiểm tra các điều kiện ràng buộc các phép toán trong hành vi của thực thể cần thiết kế.

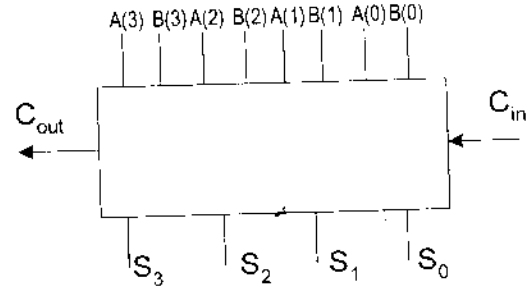



**Hình 6.2** Biểu diễn của bộ cộng một bit.

Ta hãy xét ví dụ mô tả mạch cộng một bit. Mạch cộng có tên là FULL_ADDER và có ba cổng vào là A, B và Cin. Hai cổng ra của mạch cộng là Sum và Cout. Các cổng vào và ra đều có kiểu tín hiệu là BIT, trong đó kiểu BIT là một kiểu được định nghĩa trước của ngôn ngữ VHDL. Đoạn chương trình mô tả thực thể mạch cộng một bit được viết như sau.

```
entity FULL_ADDER is
        port ( A, B, Cin : in BIT;
                    Sum, Cout : out BIT);
end FULL_ADDER
```

Chúng ta có thể kiểm soát cấu trúc và chế độ định thời gian của thực thể bằng cách sử dụng các tham số chung. Ví dụ, nếu chúng ta muốn xây dựng mạch cộng N bit, ta sẽ dùng tham số chung N để chỉ số bit của mạch cộng và tham số chung M để đặc trưng cho hành vi theo diễn biến thời gian của thực thể. Trong trường hợp mô tả bộ cộng bốn bit, ban đầu N được khởi tạo giá trị 4. Trong quá trình mô phỏng hoặc tổng hợp mạch giá trị thực của các tham số chung có thể thay đổi tuỳ theo nhu cầu. Đoạn chương trình VHDL sau cho ta ví dụ mô tả mạch cộng bốn bit.




**Hình 6.3** Biểu diễn của bộ cộng bốn bit.

```
entity ADDER is
  generic ( N : INTEGER := 4;
            M : TIME := 10 ns );
  port ( A, B : in BIT_VECTOR ( N - 1 downto 0 );
         Cin : in BIT;
         SUM : out BIT_VECTOR( N - 1 downto 0 );
         Cout : out BIT ) ;
  end ADDER;
```

## 2. Các kiến trúc

Sau khi các phần tử mạch được mô tả từ mặt ngoài bằng các khai báo thực thể, một hoặc nhiều cách thực hiện phần tử sẽ được mô tả bằng các kiến trúc. Trong ngôn ngữ VHDL, các kiến trúc cung cấp cái nhìn bên trong của thực thể. Kiến trúc của thực thể xác định mối quan hệ giữa các đầu vào và các đầu ra của thực thể và có thể biểu diễn theo hành vi, theo dòng vận chuyển dữ liệu hoặc theo cấu trúc.

Kiến trúc xác định các chức năng của thực thể. Kiến trúc chứa phần khai báo, trong đó bao gồm các khai báo của tín hiệu, khai báo kiểu, khai báo hằng, khai báo các thành phần và các chương trình con. Theo sau phần khai báo là phần thân của kiến trúc. Trong phần thân của kiến trúc chứa các câu kết cấu thực hiện đồng thời. Các kết cấu thực hiện đồng thời có thể là: các khối gán tín hiệu song song, các khối cấu trúc và các thành phần khởi tạo. Các kết cấu thực hiện đồng thời thể hiện tính chất thực hiện đồng thời của các thành phần phần cứng trong thiết kế khi có sự thay đổi trạng thái tín hiệu tác động vào mạch. Các kết cấu thực hiện đồng thời tương tác với nhau thông qua các tín hiệu. Mỗi kết cấu thực hiện đồng thời xác định một phần tử tính toán. Phần tử này đọc tín hiệu, thực hiện các phép toán trên các tín hiệu và gán những giá trị tính được cho tín hiệu. Các kết cấu này biểu diễn các khối phần cứng và cách thức liên kết giữa chúng theo dạng cấu trúc hoặc theo dạng hành vi. Bằng cách đó các kết cấu thực hiện đồng thời sẽ mô tả tổng thể cấu trúc và hành vi của thực thể được thiết kế. Các kết cấu đồng thời được thực hiện song song không phụ thuộc vào trình tự xuất hiện của chúng trong mô tả kiến trúc.

Kiến trúc được mô tả theo quy tắc cú pháp sau.

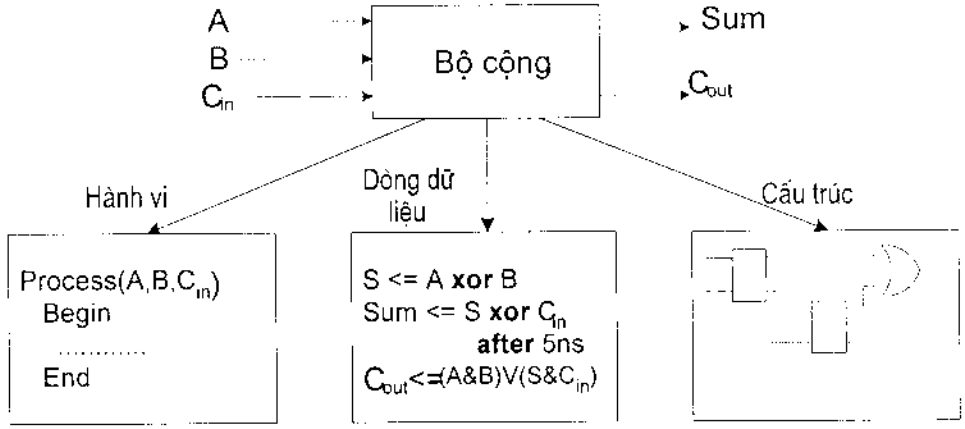
**architecture** *tên_kiến_trúc* **of** *tên_thực_thể* **is**
{ *phần_khai_báo_của_kiến_trúc* }
**begin**
{ *các_kết_cấu_thực_hiện_đồng_thời* }
**end** [ *tên_kiến_trúc* ];

Trong đó, *tên_thực_thể* là tên của đối tượng sẽ xây dựng. Tên này phải trùng với tên của thực thể tương ứng với kiến trúc đã khai báo trong phần khai báo thực thể.

Phần khai báo *phần_khai_báo_của_kiến_trúc* chứa những khai báo trong mô tả kiến trúc. Hai dấu { } có nghĩa là có thể không chứa một khai báo nào hoặc có chứa nhiều khai báo.

Phần nằm giữa hai từ khoá **begin** và **end,** *các_kết_cấu_thực_hiện_đồng_thời* xác định các khối phần cứng theo dạng cấu trúc hoặc hành vi. Các thành phần *signals* dùng để kết nối những khối riêng biệt của kiến trúc. Mỗi tín hiệu tương ứng với một kiểu dữ liệu. Các kiểu đó xác định dạng của dữ liệu được truyền trên các đường tín hiệu.

Một thực thể có thể có nhiều kiến trúc. Nhà thiết kế có thể xây dựng mô hình thết kế sử dụng các phương pháp thực hiện khác nhau theo các góc độ quan sát khác nhau hoặc ở các mức độ trừu tượng khác nhau. Thông thường, ta có thể biểu diễn kiến trúc của một thực thể ở ba phương diện: phương diện hành vi, phương diện dòng dữ liệu và phương diện cấu trúc. Một kiến trúc cũng có thể có hỗn hợp cả ba phương diện biểu diễn thiết kế. Ta hãy xét ví dụ biểu diễn kiến trúc thiết kế của mạch cộng một bit. Theo hành vi, mạch cộng có thể được biểu diễn như một hệ hàm lôgic tác động lên ba biến độc lập là A, B và Cin để hình thành các tín hiệu ra là Sum và Cout. Theo cách biểu diễn bằng dòng dữ liệu, mạch cộng thực hiện các tác động.



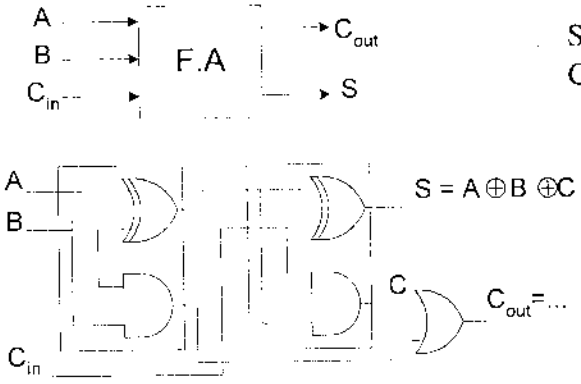

**Hình 6.4** Mô tả thực thể mạch cộng và các kiến trúc.

### a. Biểu diễn kiến trúc theo hành vi

Biểu diễn kiến trúc của thực thể theo hành vi là mô tả chức năng của hệ thống tương tự như các chương trình phần mềm bằng các quá trình tính toán. Trong biểu diễn này ta không cung cấp chi tiết việc thực hiện thiết kế. Trong ngôn ngữ VHDL, để biểu diễn kiến trúc theo hành vi, cấu trúc chính của hành vi sẽ là quá trình. Một quá trình có thể coi như là một chương trình và được xây dựng từ những cấu trúc thủ tục và có thể cho phép gọi các chương trình con giống như trong các ngôn ngữ lập trình truyền thống.

Ta hãy xét ví dụ mô tả kiến trúc của mạch cộng một bit bằng hành vi. Trong mô tả chứa một quá trình với ba tham số là các tín hiệu A, B, Cin. Các tín hiệu này được chứa trong danh sách các tín hiệu tác động vào quá trình. Việc thực hiện quá trình sẽ dừng lại nếu không có các sự kiện xảy tra trên các đường tín hiệu xuất hiện trong danh sách hay nói một cách khác là các tín hiệu trong danh sách các tín hiệu tác động không thay đổi giá trị. Mỗi khi có một sự kiện xảy ra trên các đường tín hiệu, quá trình sẽ được kích hoạt và các câu lệnh bên trong cấu trúc sẽ được thực hiện tuần tự.

```
architecture BEHAVIOR of FULL_ADDER is
begin
        process ( A, B, Cin )
        begin
                if ( A = '0' and B = '0' and C = '0' ) then
                        Sum <= '0' ;
                        Cout <= '0' ;
                elsif   ( A = '0' and B = '0' and Cin = '1' ) or
                        ( A = '0' and B = '1' and Cin = '0' ) or
                        ( A = '1' and B = '0' and Cin = '0' ) then
                                Sum <= '1' ;
                                Cout <= '0' ;
                elsif ( A = '1' and B = '1' and Cin = '1' ) then
                        Sum <= '1' ;
                        Cout  <= '1' ;
                end if
        end process;
end BEHAVIOR;
```

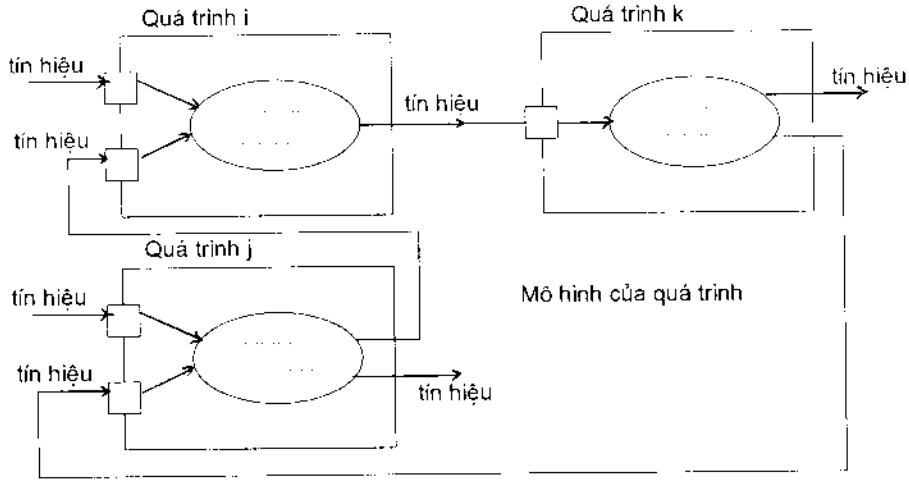



**Hình 6.5** Cấu trúc và hành vi bộ cộng một bit.

Một quá trình mô tả hành vi của một phần hoặc toàn bộ thiết kế. Quá trình xác định các khối mã lệnh tuần tự độc lập. Các khối này có thể được kích hoạt ứng với các biến đổi trạng thái của tín hiệu. Khi có nhiều hơn một quá trình trong kiến trúc thì những quá trình đó được thực hiện đồng thời.

### i. Mô hình hành vi

Thiết kế số được mô hình hóa như một nhóm các phép toán tác động lên các giá trị dữ liệu truyền qua hệ thống. Trong mô hình hành vi của VHDL




**Hình 6.6** Mô hình hoạt động của một quá trình **process**.

mỗi phép toán được gọi là một quá trình và các giá trị dữ liệu đi qua hệ thống được gọi là tín hiệu. Một hệ thống có thể coi là tập hợp các quá trình và các quá trình tương tác với nhau thông qua các tín hiệu. Tất cả các quá trình trong mô hình thực hiện đồng thời và các tín hiệu được dùng để định vị các quá trình song song.

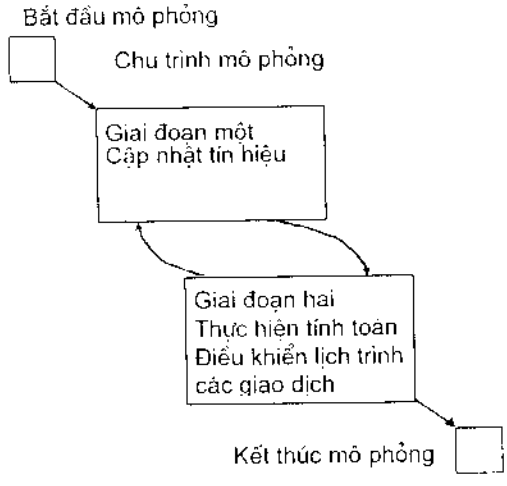
Chúng ta có thể coi việc thực hiện một quá trình như một vòng lặp vô hạn. Vòng lặp này bắt đầu từ việc thực hiện dòng lệnh đầu tiên, lần lượt đến dòng lệnh thứ hai, thứ ba, ...cho đến dòng lệnh cuối cùng và lại quay trở lại dòng lệnh đầu tiên. Việc thực hiện các câu lệnh trong quá trình được thực hiện cho đến khi gặp câu lệnh **wait**. Khi bị dừng lại, quá trình có thể được tiếp tục thực hiện trở lại. Điều kiện để quá trình thực hiện trở lại phụ thuộc vào thời gian thời gian ngừng cực đại trong câu lệnh **wait** trôi qua. Trong nhiều trường hợp quá trình có thể được kích hoạt trở lại tuỳ thuộc vào sự thay đổi trạng thái của các tín hiệu tác động hoặc khi các điều kiện đặt ra được thỏa mãn. Ngôn ngữ VHDL cung cấp khả năng mô tả những tín hiệu tác động vào các quá trình bằng danh sách các tín hiệu tác động. Khi giá trị của các tín hiệu trong danh sách tác động bị thay đổi, quá trình được kích hoạt và bắt đầu thực hiện.

Trên hình 6.6 ta có mô hình kiến trúc của hệ thống gồm ba quá trình *i*, *j*, *k*. Mỗi quá trình *i* và *j* có hai tín hiệu tác động còn quá trình *k* chỉ có một. Đầu ra của quá trình *i* nối với đầu vào của quá trình *k* và đầu ra của quá trình *j* nối với đầu vào của quá trình *i*. Cả ba quá trình cùng thực hiện đồng thời và các tín hiệu tác động dùng để kiểm soát tiến trình thực hiện các quá trình.

### ii. Mô hình điều khiển thời gian

Sự thay đổi giá trị tín hiệu tại những thời điểm thời gian xác định được thể hiện qua "thời điểm mô phỏng" của hệ thống. Thời điểm mô phỏng trong ngôn ngữ VHDL là thời điểm tại đó có các sự kiện xuất hiện trên đường tín hiệu. Khái niệm thời điểm mô phỏng trên thực tế là khác với khái niệm thời gian đồng hồ nội tại. Các quá trình trong VHDL được kích hoạt lại mỗi khi có sự thay đổi giá trị của tín hiệu trong danh sách tín hiệu tác động. Khi quá trình tạo nên giá trị cho tín hiệu đầu ra, hệ thống mô phỏng sẽ chỉ định lượng tử thời gian trước khi giá trị được gửi ra đầu ra. Lúc đó ta nói rằng hệ mô phỏng thực hiện việc định chương trình cho giao tác sau thời điểm mô phỏng xác định. Hệ thống mô phỏng của VHDL cũng cho phép định chương trình

cho một số bất kỳ các giao tác đối với đường tín hiệu ra. Tập hợp tất cả các giao tác đối với tín hiệu trong một quá trình gọi là
VHDL cung cấp mô hình hai giai đoạn: chu trình mô phỏng.




**Hình 6.7** Chu trình mô phỏng của quá trình.

Trong giai đoạn 1: giá trị của các tín hiệu sẽ thực hiện giao dịch trong thời gian hiện tại được làm mới

- Trong giai đoạn 2: những process nhận thông tin tại những tín hiệu tác động sẽ thực hiện tính toán cho tới khi bị treo. Kết thúc giai đoạn 2, thời điểm mô phỏng sẽ nhận giá trị mới và chu trình thực hiện lại.

Nhà thiết kế có thể chỉ định thời gian tính từ thời điểm hiện thời mà giá trị sẽ được gửi tới tín hiệu ra trong các câu lệnh gán tín hiệu. Nếu trong câu lệnh gán tín hiệu không chỉ rõ giá trị thời gian trễ hoặc giá trị này bằng '0' thì thời gian trễ mặc định **delta** của bộ mô phỏng sẽ được sử dụng để định lịch trình cho thao tác. Thời gian trễ này không thay đổi thời điểm xuất hiện tín hiệu đồng hồ mô phỏng nhưng được sử dụng để kết thúc một chu trình mô phỏng và bắt đầu một chu trình mới. Nếu giá trị mới được gán cho tín hiệu khác với giá trị cũ, trên đường tín hiệu xuất hiện sự kiện.

### b. Biểu diễn kiến trúc theo dòng dữ liệu

Phong cách mô tả kiến trúc theo dòng dữ liệu đặc tả hệ thống như các biểu diễn song song của các dòng dữ liệu và dòng các tín hiệu điều khiển. Theo phương pháp này chúng ta mô tả các dòng thông tin, hành vi luân chuyển các dòng dữ liệu theo thời gian của các hàm lôgic tổ hợp như mạch cộng, mạch so sánh, mạch giải mã và các phần tử lôgic cơ sở. Ví dụ, ta có thể biểu diễn kiến trúc của mạch cộng FULL_ADDER theo phong cách dòng dữ liệu như sau.

```
architecture DATAFLOW of FULL_ADDER is
        signal S : BIT ;
begin
        S <= A xor B ;
        Sum <= S xor Cin affer 10 ns;
        Cout <= ( A and B ) or ( S and Cin ) affer 5 ns ;
end DATAFLOW;
```

Trong ví dụ này ta có ba biểu thức gán tín hiệu song song. Mỗi biểu thức gán này có thể hiểu như một quá trình với tín hiệu về bên phải của phép gán là các tín hiệu nằm trong danh sách các tín hiệu tác động của quá trình. Ví dụ, trong phép gán thứ nhất, đường tín hiệu S sẽ nhận giá trị A **xor** B sau khi có sự kiện xuất hiện trên đường A, hoặc trên đường B hoặc trên cả hai đường A và B. Ở đây chúng ta không đặt giá trị tường minh của thời gian trễ. Do đó để mô phỏng tiến trình thực hiện quá trình theo thời gian, hệ thống sử dụng thời gian trễ mặc định **delta**. Trong phép gán thứ hai, đường tín hiệu Sum nhận giá trị biểu thức S **xor** Cin sau 10ns so với thời điểm có các sự kiện trên các đầu vào S và Cin của phần tử Xor. Cũng tương tự như vậy, đường tín hiệu Cout sẽ nhận giá trị

(( A **and** B ) **or** ( S **and** Cin ))

sau 5 **ns** so với thời điểm có sự kiện trên các đường tín hiệu A, B, S hoặc Cin.

Các hằng số cơ sở có thể được dùng như các tham số trễ. Trong ví dụ này, chúng ta thấy các khai báo của tham số trễ của phần khai báo thực thể và việc sử dụng chúng trong kiến trúc.

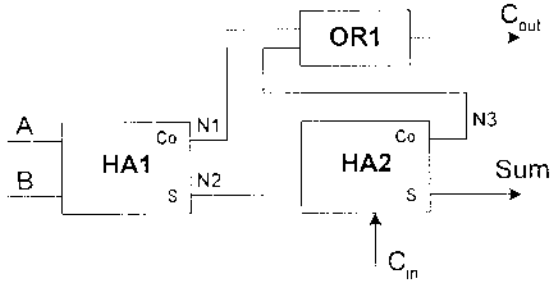
Ví dụ,

```
entity FULL_ADDER is
        generic ( N: TIME := 5 ns );
        port (A, B, Cin : in BIT; Sum, Cout : out BIT);
end FULLADDER;
architecture DATAFLOW of FULL_ADDER is
        signal S: BIT;
```

```
begin
        S <=A xor B ;
        Sum <= S xor Cin affer 2*N ;
        Cout <= (A and B) or ( S and Cin ) affer N ;
end DATAFLOW;
```

### c. Biểu diễn kiến trúc bằng cấu trúc

Phong cách mô tả kiến trúc thông qua cấu trúc xác định cấu trúc của thực thể sử dụng các khai báo của các phần tử thành phần và các phiên bản của các phần tử thành phần. Các mô tả cấu trúc chứa danh sách các phần tử hoạt động đồng thời và liên kết giữa các phần tử đó. Ví dụ, khi ta mô tả cấu trúc của mạch cộng một bit FULL_ADDER. Mạch FULL_ADDER được thiết kế trên cơ sở các thành phần là HALF_ADDER và OR_GATE. Cấu trúc được thiết kế của mạch cộng một bit chứa hai mạch nửa tổng và một phần tử lôgic cơ bản là OR_Gate. Các phần tử này liên kết với nhau bằng các tín hiệu.




**Hình 6.8** Cấu trúc ở mức kiến trúc của mạch cộng một bit của hai tín hiệu.

```
architecture STRUCTURE of FULL_ADDER is
        component HALF_ADDER
          port ( L1,L2 : in BIT;
                 Carry,Sum : out BIT );
        end component;
        component OR_GATE
```

```
        port ( L1, L2: in BIT;
               O: out BIT );
    end component;
    signal N1,N2,N3 : BIT;
begin
    HA1 : HALF_ADDER port map ( A, B, N1, N2 );
    HA2 : HALF_ADDER port map ( N2, Cin, N3, Sum );
    OR1 : OR_GATE port map ( N1, N3, Cout );
end STRUCTURE;
```

Khi độ phức tạp của thiết kế tăng lên, nhà thiết kế thường phân tách hệ thống thành những hệ thống con. Các hệ thống con này liên kết chặt chẽ theo chức năng trong thành phần của hệ thống tổng thể. Mỗi hệ thống con lại có thể được phân tách thành những phân hệ ở mức thấp hơn nữa. Trong ngôn ngữ VHDL, ở mức cao nhất của thiết kế người ta sử dụng mô hình kiến trúc theo phong cách cấu trúc của thực thể. Cấu trúc này sẽ gồm tập hợp phiên bản của các phân hệ kết nối với nhau bằng các đường tín hiệu.

Mỗi phiên bản của thành phần mạch được mô tả bằng các hộp đen trong biểu diễn cấu trúc với các liên kết đầu vào và liên kết đầu ra được mô tả rõ ràng. Các phiên bản thành phần phải tương thích với các thực thể. Các thực thể này sẽ mô tả các chức năng của thành phần mạch bằng các mô hình kiến trúc theo biểu diễn cấu trúc hoặc hành vi. Ví dụ, đối với mạch cộng một bit của hai tín hiệu, ta có mạch cộng sẽ được xây dựng từ hai mạch nửa tổng và một phần tử OR. Khi đó theo các mô tả kiến trúc của các thực thể bằng các mô tả cấu trúc, thực thể FULL_ADDER được tạo bởi hai thực thể HALF_ADDER và một thực thể OR_GATE. Trong đó thực thể HALF_ADDER có thể được xây dựng từ các phần tử XOR và AND.

Mô tả thực thể HALF_ADDER theo cấu trúc từ các phần tử AND và XOR.

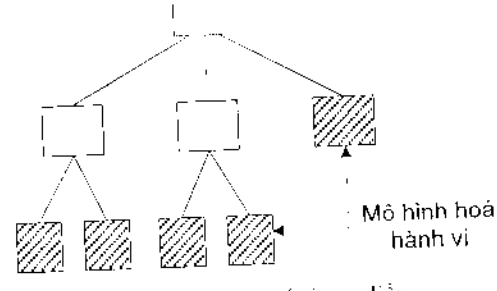
```
entity HALF_ ADDER is
port ( I0, I1 : in BIT; S, C0 : out BIT );
end HALF_ ADDER;
architecture STRUCTURE of HALF_ADDER is
    component XOR_GATE
        port ( I0, I1:  in BIT; O: out BIT );
```

```
        end component;
        component AND2_GATE
                port ( I0, I1 : in BIT; O: out BIT );
        end component;
begin
        U1: XOR_GATE port map ( I0, I1, S );
        U2: AND2_GATE port map ( I0, I1, C0 );
end STRUCTURE;
```

Mỗi thành phần của thực thể nêu trên có thể xây dựng từ các thực thể khác mô tả các chức năng của chúng. Ví dụ phần tử XOR_GATE có thể được mô tả theo hành vi như sau.

```
entity XOR-GATE;
        port ( I0, I1 : in BIT; S, C0 : out BIT );
end XOR_GATE;
architecture BEHAVIOR of XOR_GATE is
begin
        O <= I0 xor I1 affer 10 ns;
end BEHAVIOR;
```

Biểu diễn cấu trúc của các phân cấp thiết kế ảnh hưởng tới quá trình phân tách thiết kế. Điều này xuất phát từ các đặc điểm của hệ thống được thiết kế. Ở một mức phân cấp bất kỳ, hệ thống được cấu tạo bởi các liên kết của những thành phần ở mức đang xét. Biểu diễn cấu trúc của kiến trúc chứa danh sách các hộp đen. Ở mức thấp nhất của quá trình phân tách, ta phải mô tả hành vi của các phần tử nằm trong thiết kế ở mức này. Quá trình phân cấp có thể biểu diễn dưới dạng cây phân cấp ( hình 5.9 ). Tại mức thấp nhất của phân cấp, ta phải mô tả hành vi của các thực thể theo trình tự mà mô hình mạch sẽ được mô phỏng.




**Hình 6.9** Cây phân cấp biểu diễn kiến trúc của mô hình thiết kế.

### 3. Các gói thiết kế

Mục đích chính của các gói là tập hợp các phần tử có thể dùng chung giữa hai hoặc nhiều đơn vị thiết kế. Một gói bao gồm hai thành phần: phần khai báo gói và phần thân gói.

- Phần khai báo gói chứa tất cả các khai báo của mọi tên. Những tên này sẽ được các đơn vị thiết kế dùng đến khi sử dụng gói. Thông thường phần khai báo chứa một số kiểu dữ liệu chung, các hằng và mô tả của các chương trình con.
- Phần thân gói bao gồm các phần thân của các chương trình con mô tả trong phần khai báo gói. Phần thân này là ẩn đối với bên ngoài. Phần thân của gói không bắt buộc phải có nếu không có chương trình con được mô tả trong gói.

Ví dụ, ta có khai báo gói như sau. Gói này khai báo một số kiểu, biến, hằng và chương trình con.

```
package EX_PKG is
    subtype INT8 is INTEGER range 0 to 255
    constant ZERO : INT8 :=0 ;
    constant MAX : INT8 :=100 ;
    procedure Increcment ( variable count : inout INT8 )
    end EX-PKG.
```

Do trong khai báo có thủ tục Increcment nên ta cần phải có thân của gói tương ứng với khai báo gói trên.

```
package body EX-PKG is
    procedure Increcment ( variable Data: inout INT8 ) is
        begin
            if ( Count >= MAX ) then
                Count := ZERO;
            else
                Count := Count + 1;
        end if
        end Increcment;
    end EX-PKG;
```

## 4. Các cấu hình

Một thực thể có thể có một vài kiến trúc. Trong quá trình thiết kế, ta có thể cần 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 biểu diễn cấu trúc của thiết kế.

Cú pháp của mô tả cấu hình:

**Configuration** *tên_cấu hình* **of** *tên_thực_thể* **is**
{ *phần_khai_báo_của_cấu_hình* }
**for** *đặc_tả_của_khối*
{ *mệnh_đề*_**use** }
{ *các_phần_tử_của_cấu_hình* }
**end for;**

Vị từ *phần_khai_báo_của_cấu_hình* cho phép cấu hình sử dụng các phần tử trong các gói và các thư viện.
Vị từ *đặc_tả_của_khối* xác định cấu hình cho kiến trúc của thực thể.
Ví dụ

```
Configuration FADD_CONFIG of FULL_ADDER is
        for STRUCTURE
        for HA1, HA2: HALF_ADDER
                use entity WORK.HALF_ADDER(STRUCTURE)
        for OR1: OR_GATE use entity WORK.OR_GATE;
        end for
end FADD_CONFIG;
```

trong cấu hình này, chúng ta thấy:
STRUCTURE chỉ tới kiến trúc của thực thể FULL_ADDER được đặt cấu hình.
HA1 và HA2: là các thực thể gắn với thực thể HALF_ADER của kiến trúc STRUCTURE trong thư viện WORK.

Phiên bản OR1 gắn với thực thể OR_GATE trong thư viện WORK.

### 5. Các thư viện thiết kế

Phân tích VHDL là quá trình kiểm tra thiết kế VHDL cho đúng cú pháp và ngữ nghĩa. Sau khi phân tích VHDL, các đơn vị thiết kế sẽ được lưu giữ trong các thư viện để sử dụng sau này. Thư viện thiết kế có thể chứa những phần tử thư viện sau:

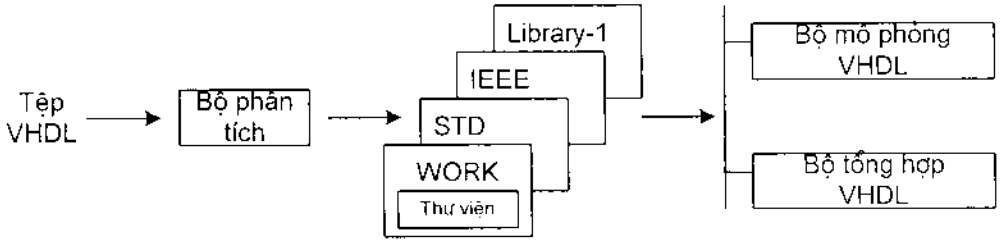
Gói : là những mô tả, khai báo được dùng chung.
Thực thể : là những mô tả thiết kế được dùng chung.
Kiến trúc : những thiết kế chi tiết được dùng chung.
Cấu hình : là những phiên bản của thực thể được dùng chung.

Các đơn vị thư viện là các cấu trúc VHDL có thể được phân tích riêng rẽ theo trình tự nhất định. Ví dụ, thực thể phải được phân tích trước kiến trúc của chúng; các gói phải được phân tích trước khi được đơn vị thiết kế sử dụng.

Trong ngôn ngữ VHDL có thư viện thiết kế đặc biệt có tên là "WORK". Khi chúng ta biên dịch một chương trình viết trên ngôn ngữ VHDL nhưng không chỉ rõ thư viện đích, chương trình này sẽ được biên dịch và chứa vào thư viện "WORK". Ví dụ, lệnh

**vc** My-Design.vhd

sẽ kiểm tra cú pháp chương trình nằm trong tệp "My-Design.vhd", dịch




**Hình 6.10** Sơ đồ biểu diễn quá trình biên dịch và mô phỏng từ chương trình thành các thư viện của ngôn ngữ VHDL, sau đó các thư viện sẽ được đưa vào hệ mô phỏng.

chương trình đó rồi chứa vào thư viện "WORK". Hình 6.10 chỉ ra các phương thức sử dụng các thư viện thiết kế trong ngôn ngữ VHDL.

## §6.3. Các kiểu dữ liệu

Ngôn ngữ VHDL có ba dạng đối tượng: biến, tín hiệu và hằng. Phần khai báo trong các cấu trúc ngôn ngữ sẽ liệt kê các đối tượng sẽ sử dụng, các kiểu của các đối tượng đó và giá trị ban đầu mà chúng sẽ nhận trong quá trình mô phỏng.

### 1. Các đối tượng dữ liệu

Trong ngôn ngữ VHDL người ta phân biệt ba loại đối tượng dữ liệu: hằng, biến và tín hiệu. Các đối tượng được đặc tả dựa vào các từ khoá. Những từ khoá này phải xuất hiện ở phần đầu của phần khai báo đối tượng.

#### a. Hằng

Hằng là đối tượng được khởi tạo bằng những giá trị nhất định khi được tạo nên trong quá trình thực hiện và sau đó giá trị của hằng không thay đổi. Hằng có thể được khai báo trong các gói, thực thể, kiến trúc, chương trình con, khối và quá trình.

Cú pháp khai báo hằng:

**constant** *tên_hằng* { , *tên_hằng* }: *kiểu* [:= *giá_trị* ];

Ví dụ,

```
constant CHAR7 : BIT_VECTOR ( 4 downto 0 ) := "00111";
constant MSB : INTEGER := 5;
```

#### b. Biến

Biến là đối tượng dữ liệu dùng để chứa những kết quả trung gian. Biến chỉ có thể được khai báo bên trong các quá trình hoặc chương trình con. Biến luôn đi đôi với kiểu, do đó biến phải được khai báo kiểu, xác định khoảng giới hạn hoặc giá trị khởi tạo ban đầu. Một cách mặc định, giá trị khởi tạo

của biến là giá trị thấp nhất trong các giá trị thuộc miền xác định của kiểu. Biến có cú pháp khai báo như sau.

**variable** *tên_biến* {, *tên_biến*}: *kiểu* [:= *giá_trị_khởi_ tạo* ];

Ví dụ,

**variable** Temp: BIT_VECTOR ( 8 **downto** 0 )
**variable** Delay: INTEGER **range** 0 **to** 15 :=0;

### c. Tín hiệu

Tín hiệu là đối tượng dữ liệu dùng để kết nối giữa các quá trình hoặc đồng bộ các quá trình. Khai báo tín hiệu sẽ tạo tín hiệu mới có các giá trị của kiểu xác định. Tín hiệu có thể được khai báo trong phần khai báo gói ( khi đó tín hiệu sẽ là tín hiệu toàn cục ), khai báo thực thể ( khi đó tín hiệu là tín hiệu toàn cục của thực thể ), khai báo kiến trúc ( tín hiệu sẽ là tín hiệu toàn cục của kiến trúc ) và trong khối. Các tín hiệu có thể được sử dụng nhưng không thể được khai báo trong các quá trình và các chương trình con. Có thể giải thích điều này như sau, các quá trình và chương trình con là các thành phần cơ sở của mô hình và được coi là các hộp đen. Các tín hiệu sẽ tác động vào các hộp đen đó từ bên ngoài. Các đáp ứng của hộp đen sẽ ảnh hưởng đến đường tín hiệu ra. Các tín hiệu có cú pháp khai báo như sau.

**signal** *tên_tín_hiệu* {, *tên_tín_hiệu* }: *kiểu* [:= *giá_trị_khởi_tạo* ];

Ví dụ,

**signal** Beep: BIT := '0';
**signal** Res : INTEGER **range** 0 **to** 1023;

## 2. Các kiểu dữ liệu

Mọi đối tượng dữ liệu trong ngôn ngữ VHDL đều phải được định nghĩa bởi các kiểu dữ liệu. Ngôn ngữ VHDL cho phép sử dụng các kiểu cơ sở để tạo nên các đối tượng phức tạp hơn.

Kiểu phải được khai báo trước khi sử dụng. Khai báo kiểu xác định tên kiểu và miền xác định của kiểu. Các khai báo kiểu có thể nằm trong phần khai báo của gói, khai báo thực thể, khai báo kiến trúc, khai báo chương trình con và khai báo các quá trình.

Các kiểu dữ liệu chính trong ngôn ngữ VHDL:

- Kiểu kiệt kê.
- Kiểu số nguyên
- Kiểu được định nghĩa trước của VHDL
- Kiểu mảng
  Kiểu bản ghi
- Kiểu STD_LOGIC.
- SIGNED và UNSIGNED.
- Các kiểu con.

### a. Kiểu liệt kê

Kiểu liệt kê được định nghĩa bằng cách liệt kê tất cả các giá trị có thể có của kiểu. Các giá trị này do người sử dụng xác định và có thể là các tên hoặc những ký tự.

Cú pháp của kiểu liệt kê

**type** *tên_kiểu* **is** ( *giá_trị_liệt_kê* {, *giá_trị_liệt_kê* } );

Trong ngôn ngữ VHDL kiểu liệt kê có đặc điểm khác với kiểu liệt kê của các ngôn ngữ lập trình khác. Mỗi giá trị trong thành phần của kiểu có thể xuất hiện trong hai hoặc nhiều hơn kiểu liệt kê.

Ví dụ,
**type** Color **is** (Red, Orange, Yellow, Green, Blue, Purple);
**type** Light **is** ( Red, Yellow, Green );
**type** STD_LOGIC **is** ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
**variable** X: Color;
**signal** Y: STD_LOGIC;

Trong các ngôn ngữ lập trình truyền thống, các liệt kê không được phép chứa những phần tử giống nhau. Trong ví dụ trên, trong các ngôn ngữ lập

trình truyền thống các khai báo hai kiểu "Color" và "Light" là sai về mặt cú pháp. Điều đó là do nếu một biến X có kiểu là "Color", thì khi thực hiện phép gán

X := Red;

chương trình dịch sẽ không xác định được giá trị 'Red' thuộc miền xác định của kiểu nào! Nếu giá trị 'Red' đó thuộc kiểu "Color" thì phép gán là đúng. Trong trường hợp giá trị 'Red' thuộc kiểu "Light" thì phép gán đó là sai.

Để giải quyết tình trạng mập mờ đó, trong ngôn ngữ VHDL tồn tại cơ chế định kiểu. Cơ chế này cho phép xác định một cách tường minh giá trị đang xét thuộc kiểu nào.

Mỗi đối tượng xuất hiện trong kiểu có một vị trí nhất định và được xác định bởi trật tự xuất hiện. Một cách mặc định, đối tượng đầu tiên có vị trí 0, đối tượng tiếp theo có vị trí 1, v.v. Trong quá trình tổng hợp phần cứng, kích thước của tín hiệu hoặc biến kiểu liệt kê được xác định bởi số lượng bit nhỏ nhất để mã hóa số lượng các giá trị liệt kê.

### b. Kiểu số nguyên

Kiểu nguyên là miền xác định của các số nguyên. Tất cả các phép toán toán học thông thường đều áp dụng được cho số nguyên. Các phép toán thực hiện trên kiểu nguyên là : +, -, *, / . Quy tắc cú pháp khai báo kiểu nguyên có dạng như sau:

**type** *tên_kiểu* **is range** *miền_số_nguyên* ;
*miền_số_nguyên* là miền con của tập hợp số nguyên.

Ví dụ,
**type** Integer **is range** -2147483647 **to** 2147483647;
**type** Twos_complement_Integer **is range** -32768 to 32767;

### c. Các kiểu được định nghĩa trước trong VHDL

Theo tiêu chuẩn IEEE 1076 – 1987 người ta xác định hai gói chuẩn: gói Standard và gói Textio trong thư viện STD. Mỗi gói chứa tập hợp các kiểu và toán tử. Trong gói Standard chứa tất cả các tệp nguồn của ngôn ngữ VHDL

với các mệnh đề ấn. Sau đây là một số kiểu chuẩn được mô tả trong gói Standard.

- **Boolean**: kiểu liệt kê có hai giá trị **false** và **true** với quan hệ **false** < **true**. Các phép tác động lên đối tượng kiểu **Boolean** là phép toán lôgic và quan hệ. Các phép toán này trả lại giá trị kiểu **Boolean**.
- **Bit**: kiểu liệt kê với hai giá trị '0' và '1'. Các phép toán lôgic có thể thực hiện trên các đối tượng kiểu **Bit** và trả lại giá trị kiểu **Bit**.
- **Character**: kiểu liệt kê với miền xác định là tập hợp các ký tự ASCII. Các ký tự không hiện được biểu diễn bằng tên chứa ba ký tự. Các ký tự hiện hình được biểu diễn giữa hai dấu móc đơn.

  **Integer**: kiểu số nguyên với những giá trị dương hoặc âm. Miền xác định của kiểu **Integer** trong gói Standard là: từ -2,147,483,647 đến 2,147,483,647. Các phép toán trên các đối tượng kiểu **Integer** là các phép toán số học: +, - , * , / .
- **Natural**: là một kiểu con của kiểu số nguyên và dùng để chỉ các số nguyên không âm – số tự nhiên.
- **Positive**: là kiểu con của kiểu số nguyên sử dụng để biểu diễn các số dương.

  **Bit_vector**: là kiểu biểu diễn mảng các **Bit**.

  **String**: kiểu dữ liệu bao gồm mảng các **Charater**. Một giá trị kiểu **String** được chứa trong một cặp móc kép.
- **Real**: mô tả các số thực trong giới hạn: từ -1.0E + 38 đến 1.0E + 38.
- Physical type **Time**: kiểu **Time** được sử dụng để biểu diễn các giá trị thời gian dùng trong quá trình mô phỏng.

### d. Kiểu mảng

Tương tự như trong các ngôn ngữ lập trình truyền thống, trong ngôn ngữ VHDL, phần tử kiểu mảng là nhóm các phần tử cùng kiểu và được truy cập tới như một đối tượng. Phần tử kiểu mảng trong ngôn ngữ VHDL có những đặc điểm như sau.

Các phần tử của mảng có thể là mọi kiểu trong ngôn ngữ VHDL.

- Số lượng các chỉ số của mảng ( nói cách khác là số chiều của mảng ) có thể nhận mọi giá trị dương.
- Mảng chỉ có một và chỉ một chỉ số dùng để truy nhập tới phần tử.

- Miền biến thiên của chỉ số xác định số phần tử của mảng và hướng sắp xếp chỉ số trong mảng: từ các chỉ số cao xuống các chỉ số thấp hoặc ngược lại.
- Kiểu của chỉ số có thể là kiểu nguyên hoặc kiểu liệt kê.

Khác với các ngôn ngữ lập trình truyền thống, trong ngôn ngữ VHDL mảng được chia làm hai loại: mảng có ràng buộc và mảng không ràng buộc.

Mảng có ràng buộc là mảng trong đó kiểu của chỉ số có miền xác định được quy định một cách tường minh. Điều đó có nghĩa là sau khi được khai báo, mảng có kích thước cố định. Như vậy mảng có ràng buộc tương tự với đối tượng kiểu mảng trong các ngôn ngữ lập trình truyền thống.

Cú pháp của mảng có ràng buộc:

**type** *tên_mảng* **is array** ( *khoảng_của_chỉ_số* ) **of** *kiểu_của_phần_tử*.

Trong đó,

*khoảng_của_chỉ_số* là miền con của kiểu nguyên hoặc kiểu liệt kê.

*kiểu_của_phần_tử* là kiểu của phần tử mảng.

Ví dụ, khai báo mảng của 64 phần tử nguyên.

**type** Arr **is array** ( 0 **to** 63 ) **of Integer**;

- Mảng không ràng buộc là kiểu mảng trong đó miền xác định của kiểu chỉ số và hướng sắp xếp của các chechỉ số không xác định mặc dù số lượng chiều của mảng được chỉ rõ. Như vậy đối tượng mảng không ràng buộc có thể có số lượng phần tử không xác định, nói cách khác mảng không ràng buộc có số kích thước biến thiên. Như vậy dung lượng bộ nhớ sử dụng cho một mảng không ràng buộc có thể thay đổi tuỳ theo nhu cầu sử dụng. Điều này khác với các ngôn ngữ lập trình truyền thống như C, Pascal, ....

Cú pháp của mảng không ràng buộc:

**type** *tên_mảng* **is array** ( *tên_kiểu_chỉ_số* **range** < > ) **of** *kiểu_phần_tử*.

*tên_kiểu_chỉ_số* là kiểu con xác định chỉ số.

Ví dụ.

```
type BIT_VECTOR is array ( Natural range < > ) of BIT;
type String is array ( postive range < > ) of character;
subtype B4 is BIT_VECTOR ( 3 downto 0 );
variable V5 : BIT_VECTOR ( 4 downto 0 );
```

Mảng thường dùng để biểu diễn cấu trúc tuyến tính như thanh ghi, RAM, ROM. Các phần tử mảng được đánh địa chỉ bằng các chỉ số, do đó chỉ số của mảng phải có kiểu rời rạc như kiểu số nguyên, kiểu liệt kê.

Ví dụ: mô tả bộ nhớ có kích thước 16 x 10:

```
subtype Int4 is Integer range 0 to 15;
subtype Int10 is Integer range 0 to 1023;
type Memory is array ( Int4 ) of Int10;
```

Ta có thể truy cập tới các phần tử của mảng theo chỉ số.

Ví dụ, để truy cập tới các phần tử của mảng X theo chỉ số I.

```
variable X: Memory;
variable I: Int4;
variable Y: Int10;
...
Y:= X( I + 3 );
```

Ngôn ngữ VHDL cho phép khai báo các mảng nhiều chiều. Ví dụ về truy cập mảng nhiều chiều được đưa ra dưới đây.

Ví dụ,

```
type Twoarray is array ( 0 to 7, 0 to 3 ) of BIT;
constant TwoROM : Twoarray:=
        (( '0', '0' , '0' , '1' ) , ( '0' , '0' , '1' , '1' ),
         ( '0', '1' , '0' , '1' ) , ( '0' , '1' , '0' , '1' ),
         ( '0', '0' , '1' , '1' ) , ( '1' , '0' , '0' , '1' ),
         ( '1', '1' , '0' , '1' ) , ( '1' , '1' , '1' , '1' ));
....
X := TwoROM (2,3);
```

### e. Kiểu bản ghi

Bản ghi là nhóm của một hoặc nhiều phần tử thuộc những kiểu khác nhau và được truy cập tới như một đối tượng. Bản ghi có những đặc điểm sau.

- 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 bản ghi.

Ví dụ kiểu bản ghi:

```
type Date_Type is ( Sun, Mon, Tue, Wed, Thr, Fri, Sat );
type Holyday is
        record
                Year  : Integer range 1900 to 1999;
                Month : Integer range 1 to 12;
                Day   : Integer range 1 to 31;
                Date  : Date_Type;
        end record;
```

Các phần tử của bản ghi được truy nhập theo tên bản ghi và tên trường. Hai thành phần này ngăn cách nhau bởi dấu '.'.

Ví dụ,

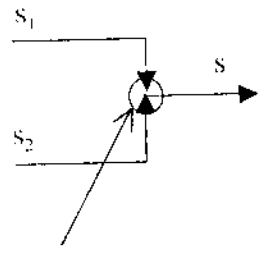
```
signal S: Holiday;
variable T1:  Integer range ( 1900 to 2099 );
variable T2: Date_Type;
...
T1 := S.Year;
T2 := S.Data;
```

### f. Các kiểu trong gói Standard Logic

Để mô hình hoá các tín hiện có thể nhận nhiều hơn hai giá trị, trong ngôn ngữ VHDL xác định gói Standard Logic ( tương ứng với chuẩn IEEE

Std 1164 – 1993 ). Hai kiểu dữ liệu cơ sở trong gói này là STD_ULOGIC và STD_LOGIC.

STD_ULOGIC xác định kiểu dữ liệu gồm chín giá trị. Các giá trị tín hiệu kiểu STD_ULOGIC không thể tham gia vào các phép toán lôgic và số học cơ bản. Trong trường hợp này ta phải mở rộng các phép toán số học và lôgic cơ bản bằng cách cung cấp các hàm quyết định. Trong những trường hợp khi tín hiệu trên một đường có thể nhận được từ nhiều nguồn tín hiệu khác nhau, chúng ta cũng phải cung cấp các hàm quyết định. Ví dụ, trong trường hợp trên hình 5.11, đường tín hiệu s là kết nối của hai đường tín hiệu độc lập $s_1$ và $s_2$ , khi đó tại điểm kết nối của hai đường tín hiệu ta cần phải xác định giá tín hiệu trên đường s sẽ nhận được từ giá trị tín hiệu trên đường $s_1$ và $s_2$ như thế nào. Thông thường giá trị trên đường s được xác định bằng các phép toán OR, AND hoặc XOR. Trong trường hợp giá trị của tín hiệu được mô tả bằng một trong chín giá trị của kiểu STD_ULOGIC, ta cần phải mở rộng các phép toán lôgic cho các giá trị không thuộc tập hợp { '0', '1'} hay { 'false', 'true' }. Những phép toán mở rộng đó được biểu diễn bằng các hàm quyết định.




**Hình 6.11** Trường hợp sử dụng hàm quyết định.

Các giá trị của kiểu STD_ULOGIC:

**type** STD_ULOGIC **is** ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-' );

STD_LOGIC là kiểu được xác định. Các hàm quyết định của kiểu này được cung cấp bởi gói Standand Logic.

Tương tự như BIT và BIT_VECTOR, ngôn ngữ VHDL cung cấp hai kiểu STD_ULOGIC_VECTOR và STD_LOGIC_VECTOR.

Ví dụ,

```
entity ADDER is
   port (  A, B: in STD_LOGIC_VECTOR ( 3 downto 0 );
           Cin  : in STD_LOGIC;
           S    : out STD_LOGIC_VECTOR ( 3 downto 0 );
           Cout: out STD_LOGIC;  );
end ADDER;
```

### g. Kiểu signed và unsigned

Kiểu có dấu **signed** và không dấu **unsigned** được sử dụng đối với những đối tượng sẽ được truy cập tới như các bit và như các số nguyên. Hai kiểu này được định nghĩa trong hai gói NUMERIC_BIT và NUMERIC_STD như sau:

**type signed is array** ( **natural range** <> ) **of** BIT/ STD_LOGIC;
**type unsigned is array** ( **natural range** <> ) **of** BIT/ STD_LOGIC;

Ví dụ,

Card_Input: **unsigned** ( 3 **downto** 0 );
Score: **signed** ( 4 **downto** 0 ) := "00100";

Các phép toán số học được mở rộng cho những kiểu dữ liệu này và được mô tả trong các gói NUMERIC_BIT và NUMERIC_STD. Các đối tượng kiểu **unsigned** được biểu diễn như các số nguyên nhị phân không dấu.

### h. Kiểu con

Các kiểu con là một tập hợp con của các kiểu đã được định nghĩa khác. Phép khai báo kiểu con có thể nằm ở mọi vị trí cho phép khai báo kiểu. Sự khác biệt giữa kiểu con và kiểu là ở chỗ kiểu con chỉ là tập con của một kiểu hoặc một kiểu con đã được định nghĩa trước.

Kiểu con thường dùng để giới hạn các dạng dữ liệu trong các phép gán và trên các đường dữ liệu. Với việc khai báo các kiểu con, các dạng dữ liệu có thể được nhận các giá trị có nghĩa. Các kiểu con kế thừa mọi toán tử và chương trình con tác động lên các kiểu cha. Ví dụ, các kiểu **natural** và **positive** là các kiểu con của kiểu nguyên **integer**. Các khai báo kiểu con có dạng sau:

**subtype** Int4 **is Integer range** 0 **to** 15;
**subtype** BIT_VECTOR6 **is** BIT_VECTOR ( 5 **downto** 0 );

## §6.4 Toán tử và biểu thức

Trong ngôn ngữ VHDL, các biểu thức là các công thức. Các công thức này xác định các tác động tính toán lên các đối tượng dữ liệu. Các biểu thức

thực hiện các tính toán số học và lôgic sử dụng các các toán tử với một số các toán hạng. Các toán tử đặc trưng cho phép toán sẽ được thực hiện còn các toán hạng là các nguồn dữ liệu cho các phép toán.

## 1. Các toán tử

Các toán tử được phân chia theo các mức độ ưu tiên và trật tự tính toán. Trong bảng 5.1 đưa ra các nhóm phép toán với mức độ ưu tiên tăng dần. Các quy ước về trật tự thực hiện các phép toán trong biểu thức được thể hiện như sau:

Trong biểu thức các phép toán có mức độ ưu tiên lớn hơn sẽ được thực hiện trước. Các dấu ngoặc đơn cũng giúp xác định đúng trật tự tính toán biểu thức.

Các phép toán trong nhóm với cùng một mức độ ưu tiên sẽ được thực hiện từ trái qua phải trong các biểu thức.

**Bảng 6.1.** Các toán tử và mức độ ưu tiên.

| | |
|---|---|
| Các phép toán lôgic | and, or, nand, nor, xor |
| Các phép toán quan hệ | =, /=, <, <=, >, >= |
| Các phép toán cộng | +, -, & |
| Toán tử dấu | +, - |
| Các phép toán nhân | *, /, mod, rem |
| Các phép toán khác | **, abs, not |

### a. Các phép toán lôgic

Trong ngôn ngữ VHDL, các phép toán lôgic gồm có **and**, **or**, **nand**, **nor**, **xor** và **not**. Các phép toán này tác động lên các dữ liệu kiểu BIT, **Boolean** và mảng một chiều các BIT. Đối với các phép toán hai ngôi **and**, **or**, **nand**, **nor**, **xor**, các toán hạng phải cùng kiểu. Trong trường hợp các toán hạng là các mảng một chiều các BIT thì các mảng phải có cùng độ dài. Các phép toán nhị phân xác định các hàm trên các bit của những phần tử mảng có cùng chỉ số, kết quả sẽ là một mảng có cùng độ dài và giới hạn các chỉ số. Phép toán một ngôi **not** xác định phép đảo bit đối với toán hạng của nó. Trong trường hợp toán hạng là một mảng thì phép toán **not** tác động lên tất cả các phần tử của mảng.

Ví dụ.

```
signal A, B, C: BIT_VECTOR ( 6 downto 0 );
...
A <= B and C;
```

### b. Các phép toán quan hệ

Các toán tử quan hệ gồm có các phép toán so sánh "=" , "/=", "<" , "<=" , ">" , ">=". Các phép toán so sánh là các phép toán hai ngôi và các toán hạng phải có cùng kiểu. Kết quả so sánh nhận kiểu **Boolean**.

Phép toán "=" kiểm tra quan hệ "bằng"; "/=" – kiểm tra quan hệ "khác" , "<" – quan hệ "nhỏ hơn"; "<=" – nhỏ hơn hoặc bằng , ">" – "lớn hơn" , ">=" – "lớn hơn hoặc bằng". Kết quả nhận giá trị **true** nếu quan hệ được nghiệm đúng và nhận giá trị **false** trong trường hợp ngược lại.

Các quan hệ "=" và "/=" xác định với mọi kiểu dữ liệu. Còn các quan hệ còn lại "<" , "<=" , ">" , ">=" chỉ xác định với các kiểu liệt kê, số nguyên, mảng một chiều các liệt kê hoặc số nguyên.

Sự sắp xếp các giá trị trong miền xác định của kiểu xác định kết quả của các phép toán so sánh. Các số nguyên được sắp xếp theo thứ tự tự nhiên từ các số âm đến các số dương. Các mảng sẽ được so sánh bằng cách so sánh từng phần tử của mảng bắt đầu từ cận trái của miền xác định chỉ số mảng.

Ví dụ,

```
signal A, B: unsigned ( 6 downto 0 );
signal C, D, E: Boolean;
...
C <= ( A = B );
```

### c. Các phép toán cộng

Các phép toán cộng bao gồm "+", "-" , "&". Các phép toán "+", "-" thực hiện trên các đối tượng kiểu **integer**. Phép toán nối "&" áp dụng với các đối tượng là mảng thanh ghi. Phép toán này xây dựng mảng mới bằng cách ghép nối hai mảng nằm trong toán hạng với nhau. Mỗi toán hạng của phép toán "&" có thể là một mảng hoặc phần tử của mảng. Các phép toán này cũng thực hiện với các toán hạng có kiểu **signed** và **unsigned**.

Ví dụ,

```
signal W: BIT_VECTOR ( 3 downto 0 );
signal X: integer range 0 to 15;
signal Y, Z: unsigned ( 3 downto 0 );
...
Z <= X + Y + Z;
Y <= Z ( 2 downto 0 ) & W( 1 );
```

**d. Các phép toán định dấu**

Các phép toán một ngôi "+" , "-" , "**abs**" thực hiện với các toán hạng dạng số và trả lại giá trị cùng kiểu.

**e. Các phép toán nhân**

Các phép toán nhân "**" , "*" , "/" , **mod** , **rem** thực hiện trên các kiểu **integer**. Phép toán "**" thể hiện phép nâng lên luỹ thừa; "*" – phép nhân; "/" – phép chia; **mod** – lấy môđun; **rem** – lấy phần dư.

Ví dụ,

```
signal A, B, C: integer range 0 to 31;
...
A <= B * B;
D <= E mod 4;
```

**2. Các toán hạng**

Trong một biểu thức, các toán tử dùng các toán hạng để tính các giá trị của chúng. Thông thường trong ngôn ngữ VHDL có nhiều dạng toán hạng. Các toán hạng cũng có thể là chính các biểu thức.

Các dạng toán hạng trong ngôn ngữ VHDL bao gồm:

- Các hằng, ký hiệu như 'x' , "1001" , 345;
- Các tên, định danh;
- Các chỉ số;

- Các tên ngắn và biệt danh;
- Các tên thuộc tính;
- Các nhóm;
- Các biểu thức định kiểu;
- Các phép gọi hàm;
- Các biểu thức chuyển đổi kiểu.

### a. Các hằng và giá trị ký hiệu

Các hằng ký và giá trị ký hiệu hoặc là giá trị số, giá trị ký tự, các giá trị liệt kê , các giá trị xâu.

- Các giá trị số là các hằng giá trị nguyên. Các giá trị số được biểu diễn tuỳ thuộc theo hệ cơ số mà giá trị đó biểu diễn.
  - Trong hệ cơ số mười các giá trị số được viết như bình thường.
  - Trong hệ cơ số khác từ 2 → 16, ta viết dưới dạng

    *cơ_số # giá_trị_hằng*

    Ví dụ, 2 # 1101
- Các hằng ký tự là các ký tự riêng lẻ được viết trong hai ngoặc đơn. Ví dụ, 'Z'.
- Các hằng liệt kê là các hằng được xác định trong định nghĩa kiểu liệt kê. Trong ngôn ngữ VHDL, các giá trị liệt kê của các kiểu liệt kê khác nhau có thể trùng nhau.

  Ví dụ,

  **type** Color **is** ( Red, Green, Blue );

  **type** Light **is** ( Red, Yellow, Green );
- Các giá trị xâu ký tự là biểu diễn của mảng ký tự một chiều các ký tự. Có hai dạng hằng xâu: các xâu ký tự và xâu các bit.
  - Các xâu ký tự là chuỗi các ký tự nằm giữa hai dấu móc kép. Ví dụ, "Demo", "1001001".
  - Xâu các bit được biểu diễn tương tự như xâu ký tự, nhưng phân biệt các các dạng xâu nhị phân, octal và hexa bằng các ký tự chỉ hệ cơ số. Ví dụ, B "10011", O "277", X "4C".

### b. Các tên và định danh

Các định danh đôi khi còn gọi một cách đơn giản là tên. Định danh là các tên đối với hằng, biến, tín hiệu, thực thể, của các cổng, chương trình con và của các khai báo tham số. Các từ khoá trong ngôn ngữ VHDL cũng là các định danh. Các tên phải bắt đầu từ chữ cái và chỉ chứa chữ cái, chữ số và dấu gạch nối '_' . Dấu gạch nối '_' không thể là ký tự sau cùng trong một định danh. Trong ngôn ngữ VHDL các tên và định danh không phân biệt theo chữ hoa và chữ thường. Một số định danh như **entity**, **port**, **is** và **end** là các từ khoá trong ngôn ngữ VHDL. Mỗi từ khoá có ý nghĩa xác định trong ngôn ngữ và không thể sử dụng trong các mục đích khác.

### c. Tên được chỉ số hóa

Những tên này xác định một phần tử của đối tượng mảng. Cú pháp mô tả của tên chỉ số như sau:

*tên_mảng* ( *biểu_thức* )

Trong đó, *biểu_thức* phải trả lại giá trị là chỉ số của phần tử mảng trong miền xác định của chỉ số.

Ví dụ,

```
type Memory is array ( 0 to 7 ) of integer range 0 to 1023;
variable Data_Array: Memory;
variable ADDR: integer range 0 to 7;
variable Data: integer range 0 to 1023;
...
Data := Data_Array ( ADDR );
```

### d. Tên ngắn và biệt danh

Tên ngắn xác định chuỗi các phần tử của đối tượng mảng. Hướng của các chỉ số phần tử mảng là **to** hoặc **downto**. Tuy nhiên chiều chỉ số của tên ngắn phải tương thích với chiều chỉ số của kiểu mảng tương ứng. Tên ngắn có thể sử dụng cùng với các biệt danh ( alias ).

Biệt danh tạo ra một tên mới cho một phần hoặc toàn bộ mảng. Biệt danh cung cấp một cơ chế truy cập khác tới phần tử của mảng.

Ví dụ,

```
variable Org  : BIT_VECTOR ( 7 downto 0 );
alias A1      : BIT_VECTOR ( 0 to 3 ) is ORG ( 7 downto 4 );
alias A2      : BIT_VECTOR ( 107 downto 100 ) is ORG;
alias A3      : BIT is ORG ( 7 );
```

### c. Thuộc tính

Thuộc tính là dữ liệu gắn liền với đối tượng trong ngôn ngữ VHDL. Thuộc tính trong ngôn ngữ VHDL của các biến hoặc tín hiệu tương ứng với những giá trị cụ thể và được xác định theo quy tắc cú pháp sau:

*tiền_tố' thuộc_tính*

Các thuộc tính được định nghĩa trước trong ngôn ngữ VHDL là **left, right, low, high, range, reverse-range, length.**

- Các thuộc tính **left** hoặc **right** trả lại chỉ số của phần tử bên trái nhất hoặc bên phải nhất của kiểu dữ liệu.
- Các thuộc tính **high, low** trả lại chỉ số của phần tử cao nhất hoặc thấp nhất của kiểu dữ liệu.
- Các thuộc tính **range** và **reverse_range** xác định khoảng của chỉ số.
- Thuộc tính **length** đưa ra số lượng các phần tử của một BIT_VECTOR.

Ví dụ,

```
subtype Index_range is integer range 10 downto 0;
variable Vect1 : BIT_VECTOR ( Indx_range );
```

khi đó ta sẽ có:

```
Vect1' left ↔ Indx_range' left == 10;
Vect1' right ↔ Indx_range' right == 0;
```

Vect1' **high** ↔ Indx_range' **high** == 10;
Vect1' **low** ↔ Indx_range' **low** == 0;
Vect1' **range** ↔ Indx_range == 10 **downto** 0;
Vect1' **reverse_range** ↔ Indx_range == 0 **to** 10;
Vect1' **length** ↔ 11.

- Các thuộc tính **event** và **stable** chỉ có đối với các tín hiệu. Các thuộc tính này chỉ rằng trên đường tín hiệu đang xét có xuất hiện sự kiện hay giá trị trên đường tín hiệu ổn định tại thời điểm hiện tại. Trong quá trình tổng hợp mạch, các thuộc tính này thường dùng với các lệnh **wait** và lệnh **if**.

Trong ngôn ngữ VHDL còn có một số thuộc tính được định nghĩa trước khác. Phần lớn các thuộc tính này được dùng cho quá trình mô phỏng. Ví dụ như các thuộc tính **delayed**, **active**, **behavior**, **structure**, **last_even**, **last_active**.

### f. Các nhóm

Các nhóm kết hợp một hoặc nhiều giá trị vào một giá trị kết hợp của kiểu mảng hoặc kiểu bản ghi. Nhóm được dùng để gán giá trị cho đối tượng kiểu mảng hoặc bản ghi khi khởi tạo hoặc trong các biểu thức gán. Việc đánh chỉ số các phần tử được đặc tả theo tên hoặc theo vị trí.

- Đặc tả theo tên: sự tương ứng giữa các phần tử của nhóm và phần tử của đối tượng được gán được chỉ rõ theo tên của từng phần tử và giá trị của chúng.
  Ví dụ,

```
type Color_list is ( Red, Orange, Blue, White );
type Color_array is array (Color_list)
                        of BIT_VECTOR ( 1 downto 0 );
variable X : Color_array;
...
X := ( Red => "00", Blue => "10", Orange => "01", White => "11" );
```

- Đặc tả theo vị trí : mỗi phần tử nhận giá trị của chúng trong biểu thức theo trật tự.
  Ví dụ, như trên trừ phép gán cuối :

```
type Color_list is ( Red, Orange, Blue, White );
type Color_array is array (Color_list)
                        of BIT_VECTOR ( 1 downto 0 );
variable X : Color_array;
...
X := ( "00" , "10" , "01" , "11" );
```

Khi gán giá trị cho nhóm, ta có thể không cần thiết đặt giá trị cho tất cả các phần tử của nhóm.

Ví dụ,

```
subtype BV7 is BIT_VECTOR ( 7 downto 0 );
variable X : BV7;
...
X := ( '0' , '0' , '0' , '1' , '1' , '1' , '1' , '1' );
X := ( '0' , '0' , '0' , others => '1' );
X := BV7' ( '0' , '0' , '0' , others => '1' );
```

### g. Các biểu thức định kiểu

Các biểu thức định kiểu là những biểu thức hoặc nhóm dùng để xác định rõ những tình trạng mập mờ. Ví dụ, như trong trường hợp hai kiểu liệt kê có những giá trị liệt kê giống nhau. Cú pháp của biểu thức định kiểu như sau:

*tên_kiểu*' ( *biểu_thức* )

*biểu_thức* phải có kiểu trùng với *tên_kiểu*.

Ví dụ, trong trường hợp hai liệt kê có những giá trị giống nhau.

```
type Color1 is ( Red, Orange, Yellow, Green, Blue, Purple );
type Color2 is ( Green, Black, White, Yellow );
variable X : Color1;
```

...
X := Color1' ( Yellow );

Giá trị 'Yellow' được xác định tường minh là thuộc kiểu Color1. Như vậy phép gán trên là đúng với cú pháp.

**h. Phép chuyển đổi kiểu**

Phép chuyển đổi kiểu cung cấp phương tiện biến đổi giá trị của những đối tượng thuộc những kiểu có quan hệ chặt chẽ với nhau ví dụ như kiểu **real** và kiểu **integer**. Cú pháp của phép biến đổi kiểu như sau:

*tên_kiểu* ( *biểu_thức* )

Ví dụ,

**signal** X: STD_LOGIC_VECTOR ( 3 **downto** 0 );
**signal** Y: STD_ULOGIC_VECTOR ( 3 **downto** 0 );
...
Y <= STD_ULOGIC_VECTOR ( X );

## §6.5. Các cấu trúc tuần tự

Trong ngôn ngữ VHDL, kiến trúc xác định chức năng của thực thể. Trong kiến trúc chứa phần khai báo các kiểu, các tín hiệu, các hằng, các thành phần và các chương trình con. Theo sau phần khai báo là các cấu trúc thực hiện đồng thời. Các cấu trúc thực hiện đồng thời có thể là các biểu thức gán tín hiệu song song, các khối và các lệnh khởi tạo phiên bản của thành phần. Các lệnh thực hiện đồng thời được kết nối với nhau bằng những tín hiệu. Mỗi khối lệnh thực hiện đồng thời trong một kiến trúc xác định một đơn vị tính toán bao gồm các thao tác tính toán như: đọc tín hiệu vào, thực hiện các tính toán trên các tín hiệu đó và gán những giá trị tính được cho các tín hiệu ra.

Trong ngôn ngữ VHDL, một cấu trúc thực hiện đồng thời là quá trình ( **process** ). Quá trình là một cấu trúc quan trọng được sử dụng để mô tả

hành vi hoạt động của mạch. Trong một kiến trúc, tất cả các quá trình sẽ được thực hiện đồng thời khi mô phỏng.

Một quá trình được xây dựng từ những cấu trúc tuần tự – hay còn gọi là các lệnh tuần tự. Trong thời gian mô phỏng, các lệnh tuần tự trong một quá trình sẽ được thực hiện lần lượt trong một chu trình vô hạn bắt đầu từ lệnh thứ nhất đến lệnh thứ *n* và sau đó việc thực hiện quá trình lại quay trở lại lệnh đầu. Việc thực hiện một quá trình trong quá trình mô phỏng trên ngôn ngữ VHDL bị dừng lại khi gặp câu lệnh **wait** và được kích hoạt lại khi có sự thay đổi trạng thái của ít nhất một trong các tín hiệu nằm trong danh sách các tín hiệu tác động.

Các lệnh tuần tự trong ngôn ngữ VHDL gồm có:

- Câu lệnh gán cho biến;
- Câu lệnh gán cho tín hiệu;
- Câu lệnh **if**;
- Câu lệnh **case**;
- Câu lệnh rỗng **null**;
- Các lệnh vòng lặp.

## 1. Phép gán biến

Trong ngôn ngữ VHDL, phép gán biến có tác dụng tương tự phép gán ở các ngôn ngữ lập trình truyền thống. Phép gán biến thiết lập giá trị mới cho biến. Cú pháp của phép gán biến như sau:

*biến* := *biểu_thức*;

Vế trái của phép gán phải là biến đã được khai báo từ trước. Vế phải của phép gán là biểu thức. Để phép gán có thể thực hiện được, biểu thức ở vế trái và biến ở vế phải của phép gán phải cùng kiểu.

Khi một biến được gán giá trị, phép gán được thực hiện với thời gian mô phỏng bằng không. Điều đó có nghĩa là sự thay đổi giá trị của biến được xảy ra tức thời ngay tại thời điểm mô phỏng hiện tại.

Các biến chỉ được khai báo trong các quá trình hoặc chương trình con và được sử dụng để lưu trữ các kết quả trung gian. Một biến được khai báo bên trong một quá trình hoặc chương trình con sẽ tồn tại cục bộ trong quá trình hoặc chương trình con đó và không thể được truy cập tới từ các cấu trúc song song khác.

Ví dụ phép gán biến,

```
subtype Int16 is integer range 0 to 65635;
signal S1, S2 : Int16;
signal GT: boolean;
process (S1, S2)
        variable A, B : Int16;
        constant C: Int16 := 100;
begin
        A := S1 + 1;
        B := S2 *2 - C;
        GT <= A > B;
end process;
```

## 2. Phép gán tín hiệu

Trong ngôn ngữ VHDL, tín hiệu là một dạng đối tượng đặc biệt. Phép gán tín hiệu dùng để thay đổi giá trị của tín hiệu. Các tín hiệu luôn được biểu diễn kết hợp với diễn biến thời gian. Phép gán tín hiệu thay đổi giá trị của tín hiệu tương ứng theo thời gian và phụ thuộc vào các mô hình quá trình trễ trong các phần tử mạch. Khi tín hiệu được gán giá trị, giá trị mới đó không được gắn với tín hiệu một cách tức thời mà phải sau một thời gian được định lịch trình trước trong thời điểm mô phỏng tiếp theo tương ứng với mô hình trễ. Phép gán tín hiệu có cú pháp như sau:

*tín_hiệu_đích* <= [ **transport** ] *biểu_thức* [ **after** *thời_gian* ]

*biểu_thức* xác định các giá trị gán. Kiểu của *biểu_thức* phải trùng với kiểu của *tín_hiệu_đích*.
*thời_gian* là biểu thức có kiểu TIME.

Trong một quá trình, việc gán giá trị của biểu thức cho tín hiệu sẽ được làm trễ khi chu trình mô phỏng đang thực hiện và được kiểm soát bởi toán tử **wait**. Chúng ta hãy xét ví dụ về phép gán tín hiệu trong đoạn chương trình sau:

```
process
begin
        S1 <= not CLK after 30 ns ;
        S2 <= DIn after 0 ns ;
        S3 <= S1 and S2 ;
        wait on CLK ;
end;
```

Trong đoạn chương trình này, thao tác gán kết quả của phép toán "**not** CLK" cho biến S1 sẽ được thực hiện sau 30 **ns** tính từ thời điểm mô phỏng hiện tại. Trong câu lệnh thứ hai, thời gian trễ của phép gán được cho bằng '0', khi đó việc gán giá trị của tín hiệu DIn cho tín hiệu S2 sẽ được thực hiện sau khoảng thời gian **delta** tính từ thời điểm mô phỏng hiện tại. Tương tự như vậy, trong phép gán tín hiệu thứ ba, kết quả của phép toán lôgic **and** thực hiện trên các giá trị hiện thời của tín hiệu S1 và S2 sẽ được gán cho tín hiệu S3 sau khoảng thời gian **delta** tính từ thời điểm mô phỏng hiện thời. Khi xuất hiện sự kiện trên đường tín hiệu CLK ( giá trị trên đường tín hiệu CLK chuyển từ '0' sang '1' ), tất cả các giao dịch đã được định lịch trình trước thời điểm xuất hiện sự kiện sẽ được thực hiện bởi bộ mô phỏng.

Khi tín hiệu được gán giá trị trong quá trình, phép gán sẽ xác định một bộ điều khiển tín hiệu. Trong một quá trình, một tín hiệu chỉ có thể có một điều khiển, điều đó có nghĩa là bên trong một quá trình tín hiệu chỉ có thể xuất phát từ một nguồn. Nếu tín hiệu được gán giá trị trong nhiều quá trình khác nhau, chúng ta nói rằng tín hiệu có nhiều điều khiển. Trong ngôn ngữ VHDL, trong trường hợp một tín hiệu có nhiều điều khiển, các hàm quyết định được xây dựng để xác định giá trị các tín hiệu có nhiều điều khiển.

Như đã đề cập tới trong chương ba, trong thiết kế mạch, chúng ta phân biệt hai hai dạng thời gian trễ khi ta điều khiển các thao tác trên tín hiệu theo thời gian: thời gian trễ quán tính và thời gian trễ lan truyền.

- Thời gian trễ quán tính được thể hiện mặc định trong ngôn ngữ VHDL. Giá trị thời gian trễ quán tính là độ dài giới hạn cần thiết của tín hiệu tác động để thiết có thể phản ứng với sự xuất hiện tín hiệu đầu vào. Nếu thời gian tồn tại của tín hiệu đầu vào không vượt quá giá trị thời gian trễ quán tính thì mạch sẽ không phản ứng với sự thay đổi của tín hiệu. Thông số thời gian trễ quán tính được sử dụng trong

quá trình mô hình hóa các thiết bị số để loại trừ sự xuất hiện những xung nhọn tại đầu vào.

- Thời gian trễ lan truyền là thời gian trễ xuất hiện khi tín hiệu đi qua mạch.

Từ khóa **transport** được dùng trong trường hợp thời gian trễ trong phép gán là thời gian trễ lan truyền. Nếu không sử dụng từ khóa **transport** trong phép gán tín hiệu, thời gian trễ sẽ được coi là thời gian trễ quán tính một cách mặc định. Việc sử dụng trễ quán tính và trễ lan truyền tạo nên những hiệu ứng khác nhau lên quá trình mô phỏng. Chúng ta hãy xét quá trình sau:

```
signal S: BIT := '0';
process
begin
        S <= '1' after 5 ns ;
        S <= '0' after 10 ns ;
end process;
```

Trong câu lệnh thứ nhất, hệ thống mô phỏng sẽ định lịch trình cho phép gán tín hiệu S giá trị '1' sau 5 ns tính từ thời điểm mô phỏng hiện thời. Đối với câu lệnh thứ hai, hệ mô phỏng sẽ định lịch trình cho phép gán giá trị '0' cho tín hiệu S sau 10 ns tính từ thời điểm mô phỏng hiện thời. Như vậy, câu lệnh thứ hai sẽ xoá bỏ kết quả của phép gán thứ nhất bởi vì giá trị thời gian trễ là thời gian trễ quán tính. Câu lệnh thứ nhất chỉ ra rằng thời gian trễ quán tính của phép gán là 5 **ns**, trong khi đó câu lệnh thứ hai khẳng định thời gian trễ quán tính của phép gán là 10 **ns**.

Nếu đổi chỗ hai phép gán trong đoạn chương trình trên, chúng ta nhận được quá trình dưới đây,

```
signal S: BIT := '0';
process
begin
        S <= '0' after 10 ns ;
        S <= '1' after 5 ns ;
        wait on CLK ;
end process;
```

Trong trường hợp này, câu lệnh gán thứ nhất sẽ xác định một phép gán giá trị '0' cho tín hiệu S sau 10 **ns** tính từ thời điểm mô phỏng hiện thời còn phép gán thứ hai sẽ xác định thao tác gán giá trị '1' cho S sau 5 **ns** so với thời điểm hiện thời. Như vậy, khi bộ mô phỏng bắt đầu thực hiện, các giao tác của phép gán thứ hai được thực hiện thay cho các giao tác của phép gán thứ nhất. Điều đó có nghĩa là, chỉ có phép gán giá trị '1' cho tín hiệu S tại thời điểm 5 **ns** tính từ thời điểm mô phỏng hiện thời sẽ được thực hiện.

Nếu trong đoạn chương trình trên, chúng ta thay thời gian trễ quán tính bằng thời gian trễ lan truyền như trong đoạn chương trình dưới đây, kết quả thực hiện của quá trình sẽ thay đổi.

```
signal S: BIT := '0';
process
begin
        S <= transport '1' after 5 ns ;
        S <= transport '0' after 10 ns ;
        wait on CLK ;
end process;
```

Trong đoạn chương trình này, việc gán giá trị '1' cho tín hiệu S sẽ được làm trễ 5 **ns** tính từ thời điểm mô phỏng. Còn phép gán tín hiệu thứ hai sẽ thêm một thao tác mới cho điều khiển của tín hiệu S. Điều này có nghĩa là sẽ có hai thao tác được thực hiện: một thao tác gán giá trị xảy ra sau 5 **ns** tính từ thời điểm mô phỏng và thao tác gán thứ hai sẽ được thực hiện sau 10 ns tính từ thời điểm mô phỏng. Kết quả là trên đường tín hiệu S sẽ xuất hiện một xung bằng '1' có độ dài 5 **ns** tại thời điểm 5 **ns** tính từ thời điểm hiện thời.

Trong ví dụ đang xét ở trên, chúng ta đổi chỗ hai phép gán như sau:

```
signal S: BIT := '0';
process
begin
        S <= transport '0' after 10 ns ;
        S <= transport '1' after 5 ns ;
        wait on CLK ;
end process;
```

Trong quá trình mô phỏng, phép gán thứ nhất xác định thao tác gán giá trị '0' cho tín hiệu S sau 10 **ns** tính từ thời điểm mô phỏng hiện thời. Trong khi đó, phép gán thứ hai sẽ đặt cho tín hiệu S giá trị '1' sau 5 **ns** tính từ thời điểm mô phỏng hiện thời. Như vậy, trong quá trình mô phỏng, các thao tác thực hiện phép gán thứ hai sẽ đè lên các thao tác của phép gán thứ nhất và quá trình trên sẽ tương ứng với việc gán cho tín hiệu S giá trị '1' sau 5 ns tính từ thời điểm mô phỏng hiện thời.

Để chỉ rõ sự khác biệt giữa phép gán tín hiệu và phép gán biến, chúng ta hãy xét ví dụ đoạn chương trình sau,

```
signal S1, CLK: BIT ;
Main:  process
begin
        S1 <= '1';
        S1 <= '0';
        wait until CLK'event and CLK = '1';
end process Main;
```

Trong đoạn chương trình này, các câu lệnh gán giá trị '1' và '0' cho tín hiệu S1 được đặt trước câu lệnh **wait**. Phép gán thứ hai sẽ loại bỏ tác động của phép gán thứ nhất và do đó tín hiệu S1 sẽ được gán giá trị '0' sau khi quá trình mô phỏng thực hiện lệnh **wait**. Trong đoạn chương trình tiếp theo dưới đây,

```
signal S1, S2: integer ;
Main:  process
begin
        S1 <= 5;
        wait until CLK'event and CLK ='1';
        S1 <= 10;
        S2 <= S1;
        wait until CLK'event and CLK = '1';
end process Main;
```

thao tác gán giá trị '10' cho tín hiệu S1 được định trước với thời gian trễ **delta** tính từ thời điểm mô phỏng hiện tại. Trong khi đó phép gán giá trị hiện thời của S1 ( bằng '5' ) cho tín hiệu S2 được định trước với thời gian trễ

**delta** tính từ thời điểm mô phỏng hiện tại. Như vậy, sau câu lệnh **wait** thứ hai, các tín hiệu S1 và S2 sẽ được gán giá trị là '10' và '5' một cách tương ứng. Điều đó có nghĩa là tín hiệu S2 sẽ nhận được giá trị bằng '5' - giá trị của S1 trước khi phép gán "S1 <= 10;" được thực hiện. Chúng ta xét thêm một ví dụ nữa với đoạn chương trình sau,

```
signal S2: integer ;
Main: process
        variable V1 : interger;
begin
        V1 := 10;
        S2 <= V1;
        wait until CLK'event and CLK = '1';
end process Main;
```

Trong ví dụ này, khi phép gán "V1 := 10;" được thực hiện, giá trị 10 sẽ được gán cho biến V1 một cách tức thời tại thời điểm mô phỏng hiện tại. Sau đó thao tác gán giá trị của V1 cho tín hiệu S2 sẽ được định lịch trình và sau khi câu lệnh **wait** được thực hiện, tín hiệu S2 sẽ được gán giá trị bằng '10'.

### 3. Câu lệnh if

Câu lệnh **if** tạo nên phân nhánh trong khi thực hiện chương trình. Tùy theo kết quả của biểu thức điều kiện mà có thể hoặc một số lệnh hoặc không có lệnh nào sẽ được thực hiện. Câu lệnh **if** có cấu trúc cú pháp như sau:

```
if < điều_kiện > then
        {< câu_lệnh_tuần_tự >}
{elsif < điều_kiện >then}
        {< câu_lệnh_tuần_tự >}
[else {< câu_lệnh_tuần_tự >}]
end if;
```

Trong đó, < *điều_kiện* > là biểu thức kiểu **boolean**.

Trong mỗi nhánh của toán tử **if** có thể chứa một hoặc nhiều câu lệnh tuần tự. Đầu tiên biểu thức điều kiện được tính toán, nếu kết quả cho giá trị **true**, các câu lệnh tuần tự nằm sau từ khóa **then** sẽ được thực hiện tức thời. Trong trường hợp ngược lại, nếu biểu thức điều kiện sau từ khóa **elsif** cho giá trị **true**, các câu lệnh tuần tự sau từ khóa **then** tiếp theo sẽ được thực hiện, ...vv. Đoạn chương trình dưới đây sử dụng câu lệnh **if** để mô tả phần tử AND với hai đầu vào:

```
signal Input1, Input2, Output: STD_LOGIC;
And_process: process ( Input1, Input2 );
begin
        if Input1 = '0'  or  Input2 = '0' then
                Output <= '0';
        elsif Input1 = 'X' or Input2 = 'X' then
                Output <= 'X';
        else    Output <= '1';
        end if
end process;
```

## 4. Câu lệnh Case

Câu 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 hiện nếu biểu thức kiểm soát có giá trị bằng giá trị tương ứng của lựa chọn. Câu lệnh **case** có cấu trúc cú pháp như sau:

```
case < biểu_thức > is
        when < lựa_chọn > =>
                {< câu_lệnh_tuần_tự >}
        { when < lựa_chọn > =>
                {< câu_lệnh_tuần_tự >} }
end case;
```

Trong đó,
< *biểu_thức* > sau khi tính phải có kiểu: nguyên, kiểu liệt kê hoặc mảng một chiều các ký tự như BIT_VECTOR;
< *lựa_chọn* > hoặc là những biểu thức cố định ( ví dụ như hằng số ) hoặc một khoảng cố định;
mỗi giá trị trong miền xác định của < *biểu_thức* > phải được phủ bởi một và chỉ một < *lựa_chọn* >;
< *lựa_chọn* > cuối cùng có thể là **others**. Từ khóa **others** tương ứng với tất cả các trường hợp còn lại của < *lựa_chọn* > trong miền xác định của biểu thức kiểm soát. Đối với các < *lựa_chọn* >, ngôn ngữ VHDL đưa ra các giới hạn sau:

- < *lựa_chọn* > phải có cùng kiểu với < *biểu_thức* >.
- mỗi giá trị của < *biểu_thức* > phải được biểu diễn bằng một và chỉ một < *lựa_chọn* >.
- nếu không có lựa chọn **others** thì mọi giá trị có thể có được của < *biểu_thức* > phải được phủ bởi tất cả các khả năng có thể có của < *lựa_chọn* >.

Ví dụ sử dụng câu lệnh **case**,

```
signal S1: integer range 0 to 7 ;
signal I1, I2, I3 : BIT ;
select_process: process ( S1, I1, I2, I3 )
begin
        case S1 is
                when 0 | 2  =>
                        OU <= '0' ;
                when 1 =>
                        OU <= I1 ;
                when 3 to 5 =>
                        OU <= I2 ;
                when others =>
                        OU <= I3 ;
        end case;
end process select_process;
```

## 5. Câu lệnh rỗng null

Câu lệnh rỗng có cú pháp như sau:

**null;**

Trong ngôn ngữ VHDL, khi chương trình mô phỏng gặp câu lệnh **null**, nó sẽ bỏ qua lệnh này và thực hiện lệnh tiếp sau. Thông thường, câu lệnh **null** dùng để chỉ trường hợp không thực hiện lệnh một cách tường minh khi có các điều kiện trả lại giá trị **true**. Do đó, câu lệnh **null** thường được dùng trong các câu lệnh **case** đối với những lựa chọn không cần thao tác.

Ví dụ,

```
variable Sel : integer range 0 to 31;
variable V : integer range 0 to 31;
case Sel is
        when 0 to 15 =>
                V := Sel ;
        when others =>
                null;
end case;
```

## 6. Các lệnh vòng lặp

Lệnh lặp **loop** chứa thân vòng lặp bao gồm dãy các câu lệnh sẽ được thực hiện không hoặc nhiều lần. Câu lệnh **loop** có quy tắc cú pháp 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* >: trong ngôn ngữ VHDL có một số dạng vòng lặp với các sơ đồ lặp khác nhau như: vòng lặp với sơ đồ lặp **for**, vòng lặp **while**, và vòng lặp không chứa 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 ngôn ngữ VHDL, câu lệnh **next** cũng có thể được dùng để thay đổi trình tự thực hiện thân của vòng lặp ( giống câu lệnh **continue** trong ngôn ngữ C ).

Ví dụ,

```
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;
```

Vòng lặp chứa < *sơ_đồ_lặp* > dạng **for** là một dạng khác của vòng lặp. Vòng lặp **for** là câu lệnh tuần tự nằm trong quá trình **process**, và cho phép thân của vòng lặp thực hiện theo số lượng xác định các lần lặp. Ví dụ,

```
for I in 1 to 10 loop
        I_square ( I ) := I * I;
end loop;
```

Biến điều khiển vòng lặp I không cần thiết phải khai báo trước và không thể được gán giá trị trong thân vòng lặp. Trong những trường hợp để chỉ ra rằng biến lặp nhận khoảng lùi, chúng ta có thể sử dụng từ khóa **downto**. Ví dụ,

```
for I in X downto Y loop
        I_square ( I ) := I * I;
end loop;
```

Sơ đồ lặp **while** là sơ đồ lặp trong đó quá trình lặp được thực hiện nếu biểu thức điều kiện lặp nhận giá trị **true**. Vòng lặp sẽ dừng lại khi giá trị của biểu thức điều kiện trở thành **false** hoặc quá trình thực hiện thân vòng lặp gặp lệnh **exit**. Cũng tương tự như vòng lặp **for**, câu lệnh **next** cũng có thể được dùng để thay đổi trật tự lặp. Ví dụ,

```
process
variable A, B, C, D: integer;
begin
        while ( ( A + B ) > ( C + D ) ) loop
                A := A - 1;
                C := C + B;
                next when ( B < 10 );
                B := B - D;
        end loop;
end process;
```

Đối với các vòng lặp, trong các nhánh chứa phép gán tín hiệu phải có ít nhất một câu lệnh **wait**. Nếu không thỏa mãn điều kiện này, quá trình mô phỏng có thể khác đi. Chúng ta hãy xét ví dụ đoạn chương trình sau,

```
signal S: integer range 0 to 10;
process
variable I: integer range 0 to 10;
begin
        wait until ( CLK'event and CLK = '0' );
        I := 0;
        while ( I < 10 ) loop
                S <= I;
                I := I + 1;
        end loop;
end process;
```

Trong ví dụ này, tín hiệu S được cập nhật 10 lần trong vòng lặp. Hành vi này sẽ không được bộ mô phỏng thể hiện. Tuy nhiên, các kết quả tức thời có thể được hiển thị trong quá trình mô phỏng kết quả tổng hợp mạch.

### 7. Câu lệnh next

Lệnh **next** chỉ dùng trong các vòng lặp. Lệnh này có tác dụng loại bỏ việc thực hiện các câu lệnh nằm giữa câu lệnh **next** và cuối vòng lặp khi điều kiện trong câu lệnh được nghiệm đúng ( tương tự lệnh **continue** trong ngôn ngữ C ). Lệnh **next** có cấu trúc cú pháp như sau,

**next** [< *nhãn_vòng_lặp* >][ **when** < *điều_kiện* >];

Trong trường hợp có các vòng lặp lồng nhau thì việc thực hiện lệnh **next** sẽ được xác định một cách tường minh bằng < *nhãn_vòng_lặp* >. Nếu không có nhãn vòng lặp trong câu lệnh, lệnh **next** sẽ tác dụng lên vòng lặp trong cùng chứa lệnh **next**. Ví dụ,

```
L1: while I < 10 loop
        L2: while J < 10 loop
                .....
                next L1 when I = J;
            end loop L2;
    end loop L1;
```

### 8. Câu lệnh exit

Câu lệnh **exit** có thể được dùng bên trong các vòng lặp. Câu lệnh này có tác dụng bỏ qua các lệnh còn lại của vòng lặp và thực hiện ngay lệnh tiếp sau vòng lặp vừa kết thúc. Lệnh **exit** có cấu trúc cú pháp như sau,

**exit** [< *nhãn_vòng_lặp* >][ **when** < *điều_kiện* >];

### 9. Câu lệnh wait

Lệnh **wait** điều khiển bộ mô phỏng ngừng việc thực hiện các quá trình hoặc các chương trình con cho tới khi điều kiện bên trong câu lệnh được

nghiệm đúng. Ta có thể nói rằng điều kiện trong câu lệnh **wait** chỉ có thể được nghiệm đúng khi xuất hiện các sự kiện trên đường tín hiệu. Như vậy, các đối tượng dữ liệu tham gia trong trong điều kiện phải là các tín hiệu. Các điều kiện để tiếp tục quá trình bị dừng có thể được biểu thị dưới ba dạng sau đây trong ngôn ngữ VHDL:

**wait**
[ **on** < *tên_tín_hiệu* > {, < *tên_tín_hiệu* >}]
[ **until** < *biểu_thức_lôgic* >]
[ **for** < *biểu_thức_thời_gian* >];

- Câu lệnh **wait on**: chỉ cho chúng ta danh sách các đường tín hiệu mà bộ mô phỏng sẽ chờ sự kiện ( sự thay đổi trạng thái các tín hiệu ). Ví dụ, đối với câu lệnh

  **wait on** A,B;

  quá trình mô phỏng sẽ dừng lại cho đến khi có xuất hiện sự kiện trên đường tín hiệu A, hoặc B. Sau đó bộ mô phỏng sẽ tiếp tục thực hiện câu lệnh đứng sau lệnh **wait**.
- Câu lệnh **wait until** sẽ dừng việc thực hiện quá trình cho tới khi biểu thức lôgic nhận giá trị **true**. Câu lệnh **wait** loại này sẽ tạo ra một danh sách ngầm định các tín hiệu tác động trong biểu thức lôgic. Mỗi khi có bất kỳ một sự kiện xuất hiện trên đường tín hiệu trong danh sách này, biểu thức lôgic sẽ được tính. Trong trường hợp lệnh **wait until** không chứa biểu thức lôgic, chúng ta hiểu rằng câu lệnh sẽ là **wait until true**. Ví dụ sử dụng lệnh **wait until,**

  **wait until** x < 10;

  quá trình thực hiện sẽ dừng lại cho tới khi x < 10.
- Câu lệnh **wait for** sẽ dừng việc mô phỏng quá trình một thời gian bằng giá trị thời gian được chỉ định bên trong điều kiện. Sau khoảng thời gian được chỉ định, bộ mô phỏng thực hiện lệnh tiếp theo sau lệnh **wait**. Nếu biểu thức thời gian không có, chúng ta hiểu rằng lệnh wait có ý nghĩa như sau:

  **wait for time'high;**

điều này có nghĩa là chúng ta không có chỉ định tường minh về thời gian chờ. Ví dụ về sử dụng câu lệnh **wait for**,

**wait for** 10 **ns**;

- Các lựa chọn trên có thể được sử dụng đồng thời, như trong ví dụ dưới đây:

**wait on** A, B **until** ( x < 10 ) **for** 10 **ns**;

điều này có nghĩa là bộ mô phỏng sẽ chờ khi có thay đổi tín hiệu A hoặc B và sẽ tiếp tục chỉ khi ( x < 10 ) tại thời điểm xuất hiện sự kiện, hoặc cho tới khi 10 ns đã trôi qua.

Trong quá trình mô hình hóa mạch, lệnh **wait** có thể được dùng để thiết lập đồng hồ cho chế độ đồng bộ. Trong các mô hình thiết kế bằng ngôn ngữ VHDL, lệnh **wait** ngừng quá trình thực hiện cho tới xuất hiện sườn dương hoặc sườn âm trên đường tín hiệu. Ví dụ, khi mô hình hóa phần tử trigơ D làm việc trong chế độ đồng bộ theo sườn dương, chúng ta có thể nhận được đoạn chương trình VHDL như sau,

```
entity D_FF is
port ( CLK: in BIT; D: in BIT; Q: buffer BIT );
end D_FF;

architecture Behavior of D_FF is
begin
        process
                wait until CLK'event and CLK = '1';
                Q <= D;
        end process;
end Behavior;
```

## 10. Phép gọi chương trình con và lệnh return

Trong ngôn ngữ VHDL có hai dạng chương trình con:

- Thủ tục: **procedure** có thể trả lại nhiều giá trị;
- Hàm: **function** chỉ trả lại một giá trị và có thể tham gia vào các biểu thức.

Câu lệnh **return** dùng để kết thúc hoạt động của các chương trình con và chỉ được sử dụng trong hàm hoặc thủ tục. Đối với hàm, sự có mặt của lệnh **return** là bắt buộc còn trong thủ tục thì không bắt buộc. Lệnh **return** có cấu trúc cú pháp như sau:

**return** [< *biểu_thức* >];

## §6.6. Các cấu trúc song song

Trong ngôn ngữ VHDL, một kiến trúc có thể chứa một hoặc nhiều các cấu trúc song song. Mỗi cấu trúc song song xác định một đơn vị tính toán bao gồm các thao tác đọc tín hiệu, thực hiện các tính toán trên các giá trị tín hiệu và gán những giá trị tính được cho tín hiệu ra. Các cấu trúc song song xác định các thành phần và các quá trình liên kết những thành phần đó bằng những cấu trúc và hành vi của các thực thể. Các cấu trúc song song sẽ được thực hiện đồng thời trong quá trình mô phỏng không phụ thuộc vào trật tự xuất hiện của chúng trong kiến trúc.

Trong ngôn ngữ VHDL có các cấu trúc song song sau:

- Quá trình **process**;
- Các phép gán tín hiệu song song;
- Phép gán tín hiệu có điều kiện;
- Phép gán tín hiệu có lựa chọn;
- Khối;
- Phép gọi chương trình con song song.

### 1. Các quá trình process

Quá trình tính toán **process** được tạo thành từ một tập hợp các câu lệnh tuần tự. Tất cả các quá trình **process** trong một thiết kế được thực hiện một cách song song. Tuy vậy, tại một thời điểm xác định chỉ có một câu lệnh

tuần tự được thực hiện trong mỗi quá trình **process**. Một quá trình **process** liên kết với phần còn lại của thiết kế thông qua các thao tác đọc các giá trị từ các tín hiệu đầu vào, các cổng được khai báo ngoài quá trình hoặc ghi giá trị vào các tín hiệu, cổng đó. Một quá trình tính toán **process** được mô tả theo quy tắc cú pháp sau:

[< *nhãn* >:] **process** [(< *danh_sách_các_tín_hiệu_tác_động* >)]
{< *phần_khai_báo* >}
**begin**
{< *lệnh_tuần_tự* >}
**end process** [< *nhãn* >];

< *phần_khai_báo* > định nghĩa các đối tượng tồn tại cục bộ trong **process** bao gồm:

- Các khai báo biến, khai báo hằng, khai báo kiểu, kiểu con;
- Thân chương trình con, khai báo các biệt danh, luật **use**.

Nếu quá trình chứa < *danh_sách_các_tín_hiệu_tác_động* > thì lúc đó quá trình này sẽ tương tự như quá trình không chứa danh sách tín hiệu tác động nhưng lại chứa lệnh **wait** ở vị trí câu lệnh cuối cùng trong quá trình:

**wait on** < *danh_sách_các_tín_hiệu_tác_động* >;

Những quá trình đó không cần có sự xuất hiện tường minh của lệnh **wait.**

Việc thực hiện một quá trình **process** bao gồm việc thực hiện lặp lại các cấu trúc tuần tự chứa bên trong thân quá trình. Sau khi câu lệnh tuần tự cuối cùng được thực hiện, việc mô phỏng quá trình sẽ được bắt đầu lại từ câu lệnh tuần tự đầu tiên của quá trình. Điều này làm cho việc mô phỏng hoạt động của quá trình giống như một vòng lặp vô hạn bao gồm tất cả các câu lệnh tuần tự bên trong quá trình. Việc thực hiện mô phỏng quá trình **process** có thể bị dừng lại bằng câu lệnh **wait** và có thể được kích hoạt lại khi xuất hiện sự kiện trên các đường tín hiệu trong danh sách tín hiệu tác động. Chúng ta hãy xét ví dụ mô hình hoá bộ mã hoá 8→3. Quá trình biểu diễn bộ mã hoá này sẽ có danh sách tín hiệu tác động gồm các tín hiệu y1, y2, y3, y4, y5, y6, y7. Điều này cũng tương đương với một quá trình không có danh sách tín hiệu tác động nhưng chứa câu lệnh:

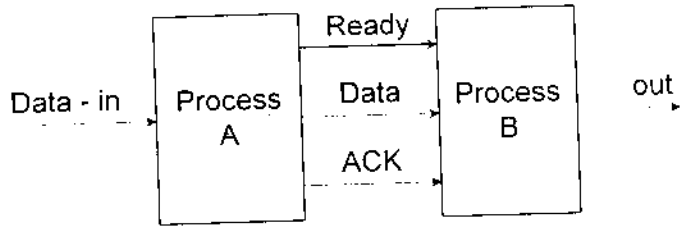
```
wait on y1, y2, y3, y4, y5, y6, y7;
```

tại vị trí câu lệnh cuối cùng trong quá trình. Bộ mã hoá 8 -> 3 được mô tả bằng đoạn chương trình VHDL dưới đây:

```
entity Encoder is
        port ( y1, y2, y3, y4, y5, y6, y7: in BIT;
                Vec: out BIT_VECTOR ( 2 downto 0 ) );
end Encoder;
architecture Behavior of Encoder is
begin
        process( y1, y2, y3, y4, y5, y6, y7)
        begin
                if ( y7 = '1' ) then Vec <= "111";
                elsif ( y6 = '1' ) then Vec <= "110";
                elsif ( y5 = '1' ) then Vec <= "101";
                elsif ( y4 = '1' ) then Vec <= "100";
                elsif ( y3 = '1' ) then Vec <= "011";
                elsif ( y2 = '1' ) then Vec <= "010";
                elsif ( y1 = '1' ) then Vec <= "001";
                else    Vec <= "000";
                end if;
        end process;
end Behavior;
```

Chúng ta xét một ví dụ về sự tương tác giữa hai quá trình. Ở đây chúng ta có một quá trình gửi thông tin và quá trình thứ hai nhận thông tin. Quá trình gửi thông tin có danh sách tín hiệu tác động gồm hai tín hiệu CLK và Ack. Quá trình nhận có hai tín hiệu trong danh sách tín hiệu tác động CLK và Ready. Hai quá trình này đồng bộ hoá hoạt động của chúng bằng lệnh Ack và Ready. Đoạn chương trình VHDL dưới đây mô tả hoạt động của hai quá trình nói trên.

```
entity HandShake is
port(  CLK   : in BIT;
       DIn   : in integer;
       DOut: out integer );
end HandShake;
```




**Hình 6.12.** Mô hình tương tác giữa hai quá trình process.

```
architecture Protocol of HandShake is
        signal Ready, Ack : BIT;
        signal Data: integer;
begin
        Send: process
        begin
                Ready <= '1';
                Data <= DIn;
                wait until CLK'event and CLK = '1' and Ack = '1';
                Ready <= '0';
                wait until CLK'event and CLK = '1' and Ack = '0';
        end process Send;
        Receive: process
        begin
                Ack <= '0';
                wait until CLK'event and CLK = '1' and Ready = '1';
                DOut <= Data;
                Ack <= '1';
                wait until CLK'event and CLK = '1' and Ready = '0';
        end process Receive;
end Protocol;
```

## 2. Các phép gán tín hiệu song song

Một dạng khác của phép gán tín hiệu trong ngôn ngữ VHDL là phép gán tín hiệu song song. Phép gán này được sử dụng bên ngoài các **process** và bên trong các kiến trúc. Dạng đơn giản nhất của phép gán tín hiệu song song có cấu trúc cú pháp như sau:

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

trong đó, < *tín_hiệu_đích* >là tín hiệu nhận giá trị của < *biểu_thức* >. Cũng giống như trường hợp phép gán tín hiệu tuần tự, luật **after** sẽ được bộ tổng hợp mạch bỏ qua.

Phép gán tín hiệu song song tương đương với một quá trình **process** chứa phép gán tín hiệu. Chúng ta hãy xét ví dụ với hai phép gán tín hiệu song song trong một kiến trúc.

Ví dụ,

```
architecture Description1 of Example is
        signal I1, I2, I3, I4, And_Out, Or_Out: BIT;
begin
        And_Out <= I1 and I2 and I3 and I4;
        Or_Out <= I1 or I2 or I3 or I4;
end Description1;
```

Đoạn chương trình này tương đương với chương trình VHDL với các **process** chứa các phép gán tín hiệu tuần tự sau,

```
architecture Description2 of Example is
        signal I1, I2, I3, I4, And_Out, Or_Out: BIT;
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 Description2;
```

Đoạn chương trình trên cũng tương đương với đoạn chương trình dưới đây, trong đó các quá trình **process** không có danh sách các tín hiệu tác động nhưng chứa các lệnh **wait**.

```
architecture Description3 of Example is
        signal I1, I2, I3, I4, And_Out, Or_Out: BIT;
begin
        process
        begin
                And_Out <= I1 and I2 and I3 and I4;
                wait on I1, I2, I3, I4;
        end process;
        process
        begin
                Or_Out <= I1 or I2 or I3 or I4;
                wait on I1, I2, I3, I4;
        end process;
end Description3;
```

### 3. Phép gán tín hiệu có điều kiện

Phép gán tín hiệu có điều kiện là câu lệnh song song thực hiện phép gán giá trị của các biểu thức cho một tín hiệu đích tùy theo các điều kiện đặt ra. Trong các biểu thức của lệnh gán, ngoại trừ biểu thức cuối cùng, những biểu thức khác được đi kèm với điều kiện gán. Khi một điều kiện nào đó nhận giá trị bằng **true**, biểu thức tương ứng với điều kiện sẽ được lựa chọn và kết quả của việc tính toán biểu thức sẽ được gán cho tín hiệu đích. Tại một thời điểm thời gian, chỉ có một biểu thức được sử dụng cho phép gán. Phép gán tín hiệu có điều kiện được mô tả theo quy tắc cú pháp sau:

*< tín_hiệu_đích >* <= {*< biểu_thức >*[**after** *< biểu_thức_thời_gian >*]
**when** *< điều_kiện >* **else**}
*< biểu_thức >*[**after** *< biểu_thức_thời_gian >*];

Mọi phép gán tín hiệu có điều kiện tương đương với một quá trình chứa lệnh **if**. Đoạn chương trình dưới đây cho ta ví dụ về sự tương đương đó.

Ví dụ,

```
architecture Description1 of Example is
        signal A, B, X, Z: BIT;
        Z <= A when ( X > 10 ) else
             B when ( X > 5 ) else
             C ;
end Description1;
```

Đoạn chương trình này sẽ tương đương với quá trình **process** sau:

```
architecture Description2 of Example is
        signal A, B, X, Z : BIT;
        constant C: BIT := '0' ;
        process ( A, B, X )
        begin
                if ( X > 10 ) then
                        Z <= A ;
                elsif ( X > 5 ) then
                        Z <= B ;
                else    Z <= C ;
                end if
        end process;
end Description2;
```

### 4. Phép gán tín hiệu theo lựa chọn

Phép gán tín hiệu theo lựa chọn thực hiện phép gán cho một tín hiệu đích

với biểu thức **with**. Giá trị của biểu thức lựa chọn nằm sau từ khóa **with** được sử dụng giống như lệnh **case**. Phép gán được thực hiện mỗi khi có xuất hiện sự kiện làm thay đổi giá trị của biểu thức lựa chọn. Cú pháp của câu lệnh được biểu diễn 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* >;

Phép gán tín hiệu theo lựa chọn tương đương với một quá trình **process** chứa câu lệnh **case**. Ví dụ, phép gán tín hiệu theo lựa chọn sau đây sẽ tương đương với một quá trình **process** chứa lệnh **case**,

```
with Sel select
        Z <= A when 0 | 1 | 2,
             B when 3 to 100,
             C when others;
```

cấu trúc của **process** tương đương:

```
process( Sel, A, B, C )
begin
        case Sel is
                when 0 | 1 | 2 =>
                        Z <= A ;
                when 3 to 100 =>
                        Z <= B ;
                when others
                        Z <= C ;
        end case;
end process;
```

## 5. Khối

Khối bao gồm một tập hợp các câu lệnh song song. Một kiến trúc có thể được phân tách thành một số các cấu trúc lôgic. 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. Khối được biểu diễn theo quy tắc cú pháp sau,

< *nhãn* >: **block**
    {< *phần_khai_báo* >}
**begin**
    {< *câu_lệnh_song_song* >}
**end block** [< *nhãn* >];

< *phần_khai_báo* > xác định các đối tượng tồn tại cục bộ trong khối và có thể có các dạng sau:

- Khai báo hằng, kiểu, kiểu con và tín hiệu;
- Thân chương trình con;
- Khai báo các biệt danh;
- Khai báo các thành phần;
- Luật **use**.

Trình tự của mỗi < *câu_lệnh_song_song* > trong khối không quan trọng bởi vì các câu lệnh luôn được kích hoạt. Các < *câu_lệnh_song_song* > luôn truyền thông tin thông qua các tín hiệu. Các đối tượng được khai báo trong một khối **block** thì xác định trong toàn bộ khối, bao gồm cả các khối con. Nếu trong một khối con có khai báo một đối tượng trùng tên với một đối tượng ở khối bao nó, khi đó khai báo của đối tượng ở khối con sẽ che lấp đối tượng ở khối bên ngoài. Chúng ta hãy xét một ví dụ,

```
architecture Behavior of Example is
        signal Ou1: integer;
        signal Ou2: BIT;
begin
        B1 : block
                signal S : integer;
```

```
        begin
                Ou1 <= S ;
        end block B1;
        B2 : block
        begin
                Ou2 <= S ;
        end block B2;
end Behavior;
```

### 6. Gọi chương trình con song song

Phép gọi chương trình con song song tương đương với các quá trình **process** bao gồm các phép gọi chương trình con tuần tự tương ứng. Mỗi phép gọi chương trình con song song tương đương với một quá trình **process** không chứa dãy danh sách các tín hiệu tác động, phần khai báo rỗng và phần thân chứa một phép gọi chương trình con, tiếp theo là một câu lệnh **wait**.

## §6.7. Các chương trình con và các gói chương trình

### 1. Các chương trình con

Trong ngôn ngữ VHDL có hai dạng chương trình con là **procedure** và **function**. Các chương trình con có thể được sử dụng tại mọi vị trí trong mô tả VHDL. Các gói chương trình xác định tên của mọi đối tượng có thể được chia sẻ giữa các thực thể.

- Các thủ tục **procedure** sẽ được gọi đến như một câu lệnh và có thể trả lại nhiều giá trị. Một thủ tục **procedure** được phép thay đổi giá trị các đối tượng tương ứng với các tham số hình thức của **procedure**. Như vậy, các tham số của một thủ tục **procedure** có thể có các dạng **in**, **out**, và **inout**.
- Các hàm **function** sẽ được sử dụng như một biểu thức và chỉ được phép trả lại duy nhất một giá trị. Hàm **function** thường

được sử dụng để thực hiện tính toán trên các giá trị của tham số và không có mục đích thay đổi giá trị của các đối tượng được gắn kết với tham số. Do đó các tham số của hàm phải có dạng **in** và thuộc nhóm các tín hiệu **signal** hoặc hằng **constant**. Đối với hàm **function**, chúng ta phải mô tả kiểu của giá trị trả lại.

Một mô tả chương trình con được phân chia thành hai phần: phần khai báo chương trình con và thân của chương trình con. Phần khai báo đưa ra các thông tin về giao diện của chương trình con ( có nghĩa là các đầu vào và đầu ra của chương trình con ) và phần thân chương trình con mô tả các chức năng của chương trình con.

Phần khai báo chương trình con được mô tả theo quy tắc cú pháp sau:

< *khai_báo_chương_trình_con* >:: =
**procedure** < *tên_thủ_tục* >< *danh_sách_tham_số* > |
**funtion** < *tên_hàm* >< *danh_sách_tham_số* >
**return** < *kiểu_giá_trị_trả_lại* > ;
< *danh_sách_tham_số* > ::= [**class**] < *danh_sách_tên* >
[**mode**] < *kiểu* >[:= < *biểu_thức* >];

trong đó,

nhóm **class** : **constant, variable, signal**;

dạng **mode**: **in, out, inout**.

Nếu không được chỉ rõ, tham số sẽ được coi rằng có **mode** là **in** một cách mặc định. Cũng tương tự như vậy, tham số có dạng **in** sẽ có **class** là hằng **constant** một cách mặc định và tương ứng của dạng **out** và **inout** là biến **variable**. Ví dụ dưới đây cho chúng ta thấy khai báo của chương trình con,

Ví dụ,

**funtion** Incrementer ( Count: **integer** ) **return integer**;

Phần thân của chương trình con được mô tả theo quy tắc cú pháp sau:

< *khai_báo_chương_trình_con* > **is**
{< *phần_khai_báo_của_chương_trình_con* >}

```
begin
        {< lệnh_tuần_tự >}
end [< tên_chương_trình_con >];
```

Ví dụ sau đây cho ta thấy mô tả của thân chương trình con. Chú ý rằng, khi gọi hàm **function**, phép gọi hàm sẽ bị ngắt khi thực hiện đến câu lệnh **return**. Câu lệnh **return** xác định giá trị trả lại cho phép gọi chương trình con. Ví dụ,

```
funtion Incrementer ( Count: integer) return integer is
        variable Temp : integer ;
        begin
                if ( Count >= 255 ) then
                        Temp := 0;
                else
                        Temp := Count + 1 ;
                end if;
                return ( Temp );
        end Incrementer;
```

Khi gọi chương trình con, các đối tượng thực tế tương ứng với tham số hình thức lớp **variable** phải là các biến; tương ứng với lớp **constant** phải là hằng số hoặc biểu thức và tương ứng với lớp **signal** phải là tín hiệu. Các hằng số và biến được truyền theo giá trị, còn tín hiệu được truyền theo địa chỉ. Do đó đối với lớp đối tượng tín hiệu thì mọi tác động lên tham số truyền vào thân chương trình con cũng chính là tác động trực tiếp lên tín hiệu được truyền vào. Chúng ta hãy xét ví dụ chương trình con sau:

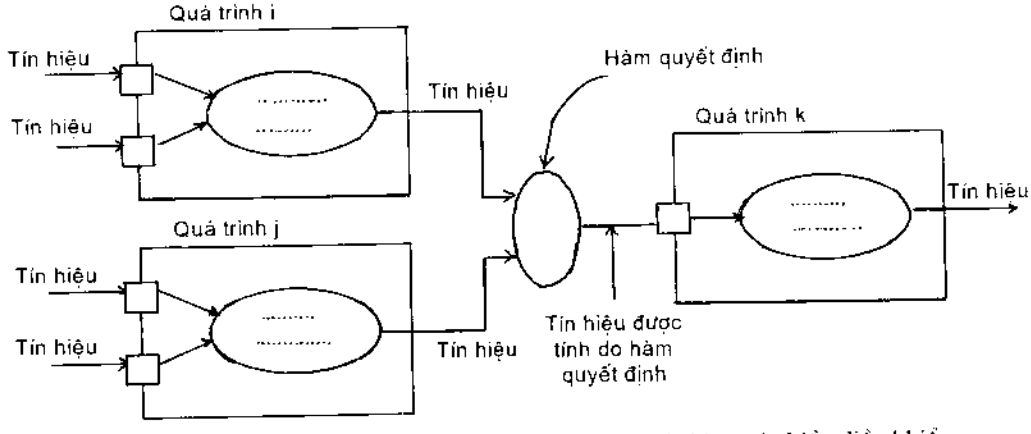
Ví dụ,

```
procedure Send ( signal CLK: BIT ; Data: integer;
                 signal Ack: BIT; signal Ready: out BIT;
                 signal Wire: out integer ) is
  begin
        wait until ( CLK'event and CLK = '1' and Ack = '1' );
        wire <= Data;
        Ready <= '1';
```

```
        wait until ( CLK'event and CLK = '1' and Ack = '1' );
        Ready <= '0';
    end Send;
```

## 2. Các hàm quyết định

Chúng ta biết rằng, mỗi tín hiệu đều xuất phát từ một nguồn. Nói cách khác là mỗi tín hiệu có một điều khiển. Trong nhiều trường hợp, ví dụ như khi các đường tín hiệu bị chập lại, tín hiệu đi ra khỏi nút chập sẽ được tổng hợp từ các tín hiệu đi vào nút theo một luật xác định. Trong trường hợp này, chúng ta nói rằng tín hiệu đi ra khỏi nút chập có nhiều điều khiển. Ngôn ngữ VHDL cho phép chúng ta có thể xác định các tín hiệu xuất phát từ nhiều nguồn nếu sử dụng những hàm quyết định. Những hàm quyết định này dùng để xác định giá trị của đường tín hiệu từ những giá trị nhận được từ nhiều nguồn điều khiển. Ví dụ,

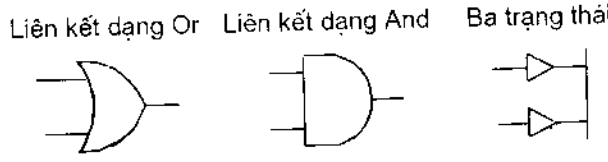



**Hình 6.13.** Sơ đồ hành vi của các quá trình sinh ra tín hiệu có nhiều điều khiển.

Trong ví dụ này, cả hai quá trình **process** i và **process** j cùng điều khiển một tín hiệu, do đó chúng ta cần có hàm quyết định để xác định giá trị cho tín hiệu nhận được.

Đối với quá trình mô phỏng, các hàm quyết định có thể là mọi hàm bao gồm các chương trình viết trên ngôn ngữ VHDL. Trong những mạch thực, kết quả của các hàm quyết định là liên kết các tín hiệu và cùng cho qua một phần tử lôgic đặc biệt có một đầu ra. Trong kỹ thuật, chỉ có một số giới hạn

các Trong các phần cứng chỉ có một vài kiểu thực hiện các hàm quyết định như liên kết dạng Or, liên kết dạng And hoặc liên kết dạng ba trạng thái ( hình 6.14).




**Hình 6.14.** Các biểu diễn phần cứng của các hàm quyết định: hàm quyết định dạng Or, dạng And và ba trạng thái.

Các tín hiệu được các hàm quyết định tạo ra nếu khai báo của tín hiệu chứa cả các hàm quyết định hoặc khai báo kiểu con của tín hiệu chứa hàm quyết định. Ví dụ,

```
signal NODE: WIRE_AND BIT;
subtype RESOLVED_STD is WIRE_OR STD_ULOGIC;
```

Trong ví dụ này, khai báo thứ nhất cho chúng ta thấy tín hiệu NODE là tín hiệu được xác định với hàm quyết định là WIRE_AND. Mỗi khi xuất hiện sự kiện trên đường tín hiệu NODE, hàm WIRE_AND được gọi và trả lại giá trị kiểu BIT – là giá trị của tín hiệu NODE. Khai báo thứ hai xác định kiểu con được quyết định RESOLVED_STD. Các tín hiệu được khai báo thuộc kiểu RESOLVED_STD là những tín hiệu được quyết định. Dưới đây chúng ta thấy cách để xác định và sử dụng các tín hiệu được quyết định:

- Xác định kiểu tín hiệu ( nếu cần thiết );
- Xác định hàm quyết định. Hàm này sẽ nhận các tín hiệu vào và trả lại tín hiệu thuộc kiểu đã được xác định;
- Khai báo kiểu con của kiểu tín hiệu kèm với hàm quyết định;
- Khai báo và sử dụng các tín hiệu được quyết định.

Trong ví dụ dưới đây, chúng ta thấy tín hiệu Z được khai báo là thuộc kiểu được quyết định WIREOR_STD, trong đó hàm quyết định của kiểu tín hiệu là WIRE_OR. Trong kiến trúc, chúng ta có hai phép gán tín hiệu đồng thời. Cả hai phép gán đều gán giá trị cho tín hiệu Z. Như vậy tín hiệu Z sẽ nhận

giá trị được xác định qua phép toán or giữa hai đầu vào của hàm quyết định: ( I1 **and** I2 ) và ( I3 **xor** I4 ).

Ví dụ:

```
architecture Resolved_Arch of WOR is
        function  WIRE_OR( Din: in STD_ULOGIC_VECTOR )
                                        return STD_ULOGIC is
        begin
          .....
              return( Din( 0 ) or Din( 1 ) );
        end WIRE_OR;
        subtype WIREOR_STD is WIRE_OR STD_ULOGIC;
        signal I1, I2, I3, I4: STD_ULOGIC;
        signal Z: WIREOR_STD;
begin
        Z <= I1 and I2;
        Z <= I3 xor I4;
end WOR;
```

Các gói chương trình STD_LOGIC_1164 của thư viện IEEE chứa các kiểu dữ liệu được quyết định là STD_LOGIC và STD_LOGIC_VECTOR. Dạng quyết định của kiểu STD_LOGIC là dạng quyết định theo ba trạng thái. Các tín hiệu có nhiều nguồn điều khiển sẽ được gán cho tín hiệu kiểu STD_LOGIC và giá trị được quyết định bởi các hàm ba trạng thái.

### 3. Các gói chương trình

Các kiểu dữ liệu, hằng số và chương trình con có thể được khai báo bên trong các khai báo thực thể hoặc bên trong thân của các kiến trúc. Các khai báo này là cục bộ trong những kiến trúc tương ứng và thực thể khác không thể truy cập tới những đối tượng đó. Tuy nhiên trong nhiều trường hợp những kiến trúc khác nhau cũng có thể phải chia sẻ những đối tượng chung nào đó. Các gói chương trình ( Packages ) trong ngôn ngữ VHDL cho phép chúng ta thực hiện các khai báo chung cho nhiều thực thể khác nhau. Một gói chương trình của ngôn ngữ VHDL được biểu diễn thành hai phần: phần khai báo gói và thân của gói.

Phần khai báo gói mô tả các giao diện của gói và có cấu trúc cú pháp như sau,

**package** < *tên_của_gói* > **is**
{< *các_khai_báo_thuộc_gói* >}
**end** [< *tên_gói* >];

Các khai báo thuộc gói có thể là các :

- Khai báo kiểu, kiểu con, tín hiệu, hằng, biệt danh;
- Khai báo thành phần, chương trình con;
- Luật **use**;
- Và có thể chứa cả các package khác.

Các khai báo tín hiệu trong gói tạo nên một số vấn đề trong quá trình tổng hợp mạch bởi vì một tín hiệu không thể phân phối giữa hai thực thể. Cách giải quyết thông dụng vấn đề này là tín hiệu sẽ được khai báo như một tín hiệu cục bộ. Nói cách khác, nếu hai thực thể sử dụng cùng một tín hiệu trong gói, mỗi thực thể sẽ nhận được một phiên bản sao chép của tín hiệu này.

Ví dụ về khai báo gói:

```
library IEEE
use IEEE.NUMERIC_BIT.all;
package Watch_Pkg is
        subtype Mouth_type is interger range 0 to 12;
        subtype Date_type is interger range 0 to 31;
        subtype BCD4_type is unsigned ( 3 downto 0 );
        subtype BCD5_type is unsigned ( 4 downto 0 );
        constant BCD5_1: BCD5_type := B"0_0001";
        constant BCD5_7: BCD5_type := B"0_0001";
        function BCD_Inc ( L : in BCD4_type ) return BCD5_type;
end Watch_Pkg;
```

Thân của một gói xác định các hành vi của gói. Thân của một gói luôn phải cùng tên với khai báo gói. Phần thân gói được bắt đầu bởi từ khóa **package body**. Các thông tin trong trong thân của gói không thể nhìn thấy

được từ các thiết kế hoặc thực thể sử dụng gói đó. Như vậy từ các thiết kế và các thực thể chúng ta chỉ có thể truy cập tới các đối tượng trong gói thông qua các giao diện được xác định trong phần khai báo gói mà không thể can thiệp trực tiếp vào bên trong gói. Nói một cách khác gói là một hộp đen với các giao diện được đưa ra trong phần khai báo. Thân của gói được mô tả theo quy tắc cú pháp như sau:

**package body** < *tên_của_gói* > **is**
    {< *các_khai_báo_trong_thân_gói* >}
**end** [< *tên_của_gói* >];

Trong phần khai báo trong thân gói có thể chứa các:

- Khai báo kiểu, kiểu con, hằng;
- Thân chương trình con;
- Luật **use**.

Ví dụ dưới đây cho chúng ta thấy việc mô tả thân của gói trong ngôn ngữ VHDL:

```
package body WATCH_PKG is
function BCD_INC ( L: in BCD4_TYPE )
                        return BCD5_TYPE is
    variable V1, V2, V: BCD5_TYPE;
begin
    V1 := L + BCD5 - 1 ;
    V2 := L + BCD5 - 7 ;
    case V2( 4 ) is
        when '0' => V := V1;
        when '1' => V := V2;
    end case;
    return ( V );
end BCD_INC;
end WATCH_PKG;
```

Các phần tử được mô tả trong phần khai báo gói không thể nhìn thấy được một cách tự động từ các thư viện khác. Luật **use** đứng trước một đơn vị chương trình sẽ làm cho tất cả các phần tử trong phần khai báo gói có thể được truy cập từ đơn vị chương trình đó. Ví dụ, giả thiết rằng, gói chương

trình WATCH_PKG nêu trên được biên dịch và đưa vào thư viện MY_LIBRARY. Để có thể sử dụng các khai báo trong gói WATCH_PKG, chúng ta cần đưa chúng vào thư viện chương trình hiện thời sử dụng đoạn lệnh sau:

```
library MY_LIBRARY ;
use MY_LIBRARY. WATCH_PKG.all;
```

Hội đồng chuẩn hoá IEEE xác định hai thư viện cho ngôn ngữ VHDL là STD và IEEE. Mỗi thư viện chứa một số gói như:

- STANDARD và TEXTIO đối với thư viện STD. Gói STANDARD xác định các kiểu dữ liệu quan trọng như **integer**, **boolean**, BIT, ...
- STD_LOGIC_1164, NUMERIC_BIT, NUMERIC_STD đối với thư viện IEEE. Các gói này chứa các kiểu và những hàm quan trọng cho quá trình tổng hợp và mô phỏng bằng ngôn ngữ VHDL.

Trong chương này chúng ta đã khảo sát những đặc điểm cơ bản của ngôn ngữ VHDL - một ngôn ngữ mô tả phần cứng điển hình. Ngôn ngữ này được dùng để mô hình hoá mạch trong công nghệ thiết kế, chế tạo mạch với độ tích hợp cao và siêu cao. Những thành phần chính của ngôn ngữ như các lệnh tuần tự, các lệnh song song, các chương trình con, các phương pháp mô tả thực thể và phương pháp luận biểu diễn mạch theo cấu chúc, hành vi hoặc theo dòng dữ liệu đã được đề cập tới. Trong chương tiếp theo chúng ta sử dụng ngôn ngữ VHDL để xây dựng các mô hình những mạch lôgic cơ bản.

**Bài tập cho chương 6**

1. Hãy mô tả hành vi của bộ chuyển mạch 2 - 1.

2. Hãy mô tả hành vi của bộ giải mã 2 – 4 bằng câu lệnh **case**.

3. Hãy mô tả hành vi của bộ giải mã 2 – 4 bằng câu lệnh **if**.

4. Bộ ALU 8 bit thực hiện các phép toán:
   - phép cộng khi đầu vào điều khiển nhận giá trị "00";
   - phép trừ khi đầu điều khiển nhận giá trị "01";
   - phép toán **and** khi đầu điều khiển nhận giá trị "10";
   - phép toán **or** khi đầu điều khiển nhận giá trị "11".

   Hãy viết chương trình trên ngôn ngữ VHDL mô tả hành vi của bộ tính toán số học – lôgic nói trên.

5. Viết chương trình trên ngôn ngữ VHDL mô tả cấu trúc và chức năng của bộ chuyển đổi mã BCD thành hiển thị 7 – mảnh.

6. Hãy viết chương trình con thực hiện phép cộng hai số theo mã BCD.

7. Hãy viết chương trình con kiểm tra tính chẵn lẻ của chuỗi 16 bit.

8. Viết chương trình con thực hiện việc chuyển đổi số nguyên trong khoảng từ 0 đến 255 thành vectơ 8 bit và ngược lại.

# CHƯƠNG VII. MÔ HÌNH HÓA MẠCH BẰNG NGÔN NGỮ VHDL

## §7.1. Mô hình hóa trên mức cấu trúc

Các mạch số thường được biểu diễn theo một cấu trúc phân cấp các thành phần. Mỗi thành phần có tập hợp các cổng để liên kết với các thành phần khác. Một thiết kế bao gồm một tập hợp những phiên bản của mạch xét trên những mức độ chi tiết khác nhau. Một phiên bản mạch trên một mức chi tiết cụ thể của thiết kế được tạo thành từ các mô tả những phần tử trên mức chi tiết đó và những phần tử này được liên kết thông qua các tín hiệu tại các cổng của chúng xét trên mức chi tiết này. Khi mô tả mạch bằng ngôn ngữ VHDL, việc phân cấp thiết kế được thực hiện thông qua các khai báo thành phần mạch và các biểu thức mô tả phiên bản của thành phần.

Khi mô tả hành vi của mạch, thành phần cơ sở của hành vi là các câu lệnh mô tả quá trình ( **process** ), còn khi mô tả cấu trúc, đơn vị cơ sở sẽ là các câu lệnh mô tả phiên bản của thành phần. Những câu lệnh mô tả quá trình và các câu lệnh mô tả phiên bản thành phần cần phải được gói vào trong thân của mô tả kiến trúc. Các mô tả kiến trúc sẽ có chức năng phân tách các đơn vị thiết kế. Một trong những đặc tính quan trọng của VHDL là ngôn ngữ này cho phép mô hình hoá thiết kế trên những mức độ trừu tượng và chi tiết khác nhau. Nói một cách khác, một biểu diễn kiến trúc có thể bao gồm các câu lệnh mô tả quá trình, các câu lệnh mô tả thành phần.

### 1. Khai báo thành phần

Một kiến trúc có thể sử dụng những thực thể đã được mô tả một cách độc lập. Những thực thể này được chứa trong thư viện thiết kế và có thể được sử dụng trong các khai báo thành phần và những câu lệnh mô tả các phiên bản của thành phần. Trong phần mô tả của thiết kế, mỗi câu lệnh khai báo thành phần tương ứng với một thực thể. Để có thể mô phỏng hoặc tổng hợp một thiết kế nào đó, thực thể và các mô tả kiến trúc của tất cả các thành phần mạch phải được biên dịch trước trong thư viện thiết kế.

Trong trình tự mô phỏng hoặc tổng hợp thiết kế, thực thể và mô tả kiến trúc đối với mọi thành phần phải được biên dịch trong thư viện thiết kế.

Cách thức mô tả cấu trúc của thành phần cũng tương tự như cách thức mô tả thực thể. Trước hết để mô tả cấu trúc của thành phần, chúng ta phải xác định rõ các giao diện của thành phần. Các giao diện này chính là các đường tín hiệu vào và ra khỏi thành phần. Trước khi được sử dụng, các thành phần phải được khai báo một cách tường minh theo quy tắc cú pháp như sau:

**component** < tên_thành_phần >
[**port** (< khai_báo_cổng_cục_bộ >)]
**end component**;

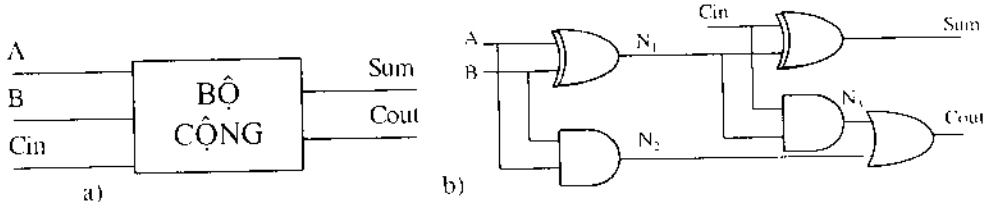
Ví dụ, chúng ta mô tả thành phần của bộ cộng đầy đủ FullAdder. Bộ cộng đầy đủ một bit có ba đường tín hiệu vào:

- Đường vào A, B là đường tín hiệu tương ứng với hàng chữ số thứ *k* của hai số hạng;
  Đường vào Cin là đường tín hiệu tương ứng với hàng giá trị nhớ từ hàng chữ số thứ *k-1* vào hàng chữ số thứ *k*;

và hai đường tín hiệu ra:

- Đường tín hiệu Sum là tổng của các đường tín hiệu A, B, Cin;
  Đường tín hiệu Cout là hàng giá trị nhớ từ hàng chữ số thứ *k* vào hàng chữ số thứ *k+1*.

Về mặt cấu trúc, nếu xét trên mức lôgic, bộ cộng đầy đủ 1 bit được tạo thành từ các phần tử AND, OR và XOR. Trên sơ đồ cấu trúc, bộ cộng này được mô tả trên hình 7.1b. Như vậy, bộ cộng gồm có hai phần tử XOR, hai phần tử AND và một phần tử OR. Các phần tử này đều là những phần tử có hai đầu vào và mỗi phần tử lôgic sẽ được mô tả bằng các khai báo thành phần như dưới đây.




**Hình 7.1.** Giao diện và cấu trúc bộ cộng đầy đủ một bit.

```
component Or2_gate
        port (I0, I1: in STD_LOGIC ; O : out STD_LOGIC);
end component;

component Xor_gate
        port (I0, I1: in STD_LOGIC ; O : out STD_LOGIC);
end component;

component And2_gate
        port (I0, I1: in STD_LOGIC ; O : out STD_LOGIC);
end component;
```

## 2. Mô tả các phiên bản của thành phần

Các thành phần mô tả trong kiến trúc có thể được khởi tạo bằng các câu lệnh tạo phiên bản thành phần. Tại các vị trí được tạo ra, phiên bản của thành phần được biểu diễn thông qua các thuộc tính bên ngoài như tên, kiểu hoặc hướng tín hiệu tại các cổng vào/ra, nhưng các tín hiệu bên trong thành phần không được biểu diễn tường minh. Các câu lệnh khởi tạo sinh ra các phiên bản của thành phần và kết nối chúng để tạo nên một danh sách liên kết của thiết kế.

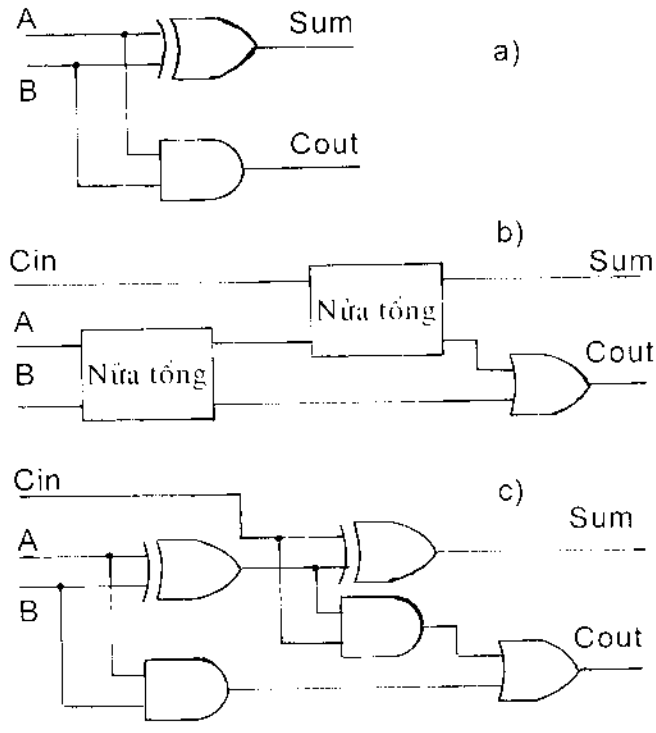
Câu lệnh tạo phiên bản thành phần có cấu trúc cú pháp như sau:

```
< nhãn_khởi_tạo >:< tên_thành_phần >
port map ( [ < tên_cổng_cục_bộ > => ] < biểu thức >
{, [ < tên_cổng_cục_bộ > => ] < biểu thức > } );
```

Trong đó, < nhãn_khởi_tạo > là tên của phiên bản vừa được tạo ra của thành phần. Mỗi câu lệnh tạo phiên bản thành phần luôn phải được kèm với một nhãn_khởi_tạo. Các nhãn này sẽ được tham chiếu tới trong các cấu hình của thành phần.

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. Các cổng được mô tả trong khai báo thành phần là những cổng cục bộ. Trong khi đó, những cổng của các phiên bản thành phần được gọi là cổng

thực. Trong các câu lệnh tạo phiên bản thành phần, cấu trúc **port map** đặt tương ứng mỗi cổng thực của phiên bản với một cổng cục bộ của thành phần. Mỗi cổng thực phải là đối tượng dạng *tín hiệu*. Trong ngôn ngữ VHDL đưa ra hai phương pháp ánh xạ các cổng cục bộ vào các cổng thực: ánh xạ theo vị trí và ánh xạ theo tên.




**Hình 7.2.** Cấu trúc phân cấp của bộ cộng một bit: a) cấu trúc bộ nửa tổng một bit; b) xây dựng bộ cộng đầy đủ một bit từ các bộ nửa tổng; c) chi tiết bộ cộng đầy đủ một bit từ các phần tử lôgic AND, OR, XOR.

- 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 mỗi cổng với các tín hiệu thực:

< tên_cổng_cục_bộ > ⇒ < tên_tín_hiệu_thực >.

Chúng ta hãy xét ví dụ mô tả cấu trúc của bộ nửa tổng một bit, trong đó bộ cộng được tạo thành từ các phần tử lôgic AND, XOR theo sơ đồ trên hình 7.2a.

```
architecture Inst of HalfAdder is
        component Xor2
                port ( I0, I1: in STD_LOGIC; O: out STD_LOGIC );
        end component;
        component And2
                port ( I0, I1: in STD_LOGIC; O: out STD_LOGIC );
        end component;
        begin
                G1: Xor2 port map ( A, B, Sum );
                G2: And2 port map ( I0 ⇒ A, I1 ⇒ B, O ⇒ Cout );
        end Inst;
```

Trong ví dụ này, phiên bản G1 của phần tử Xor2 có các cổng được ánh xạ theo vị trí vào các đường tín hiệu A, B và Sum; còn phiên bản G2 của phần tử And2 được ánh xạ tường minh theo tên của cổng cục bộ và đường tín hiệu.

Thông thường, mỗi thành phần sẽ thể hiện một thực thể độc lập của thiết kế, như vậy, các phiên bản của thành phần sẽ tạo nên các lớp phân cấp của thiết kế. Điều này được minh họa trên hình 7.2. Trong ví dụ này chúng ta thấy cấu trúc của bộ cộng đầy đủ một bít được xây dựng trên cơ sở của bộ nửa tổng một bít và phần tử OR ( hình 7.2b ). Trong khi đó, bộ nửa tổng một bít cũng có thể được coi là một thực thể độc lập và có cấu trúc được tạo thành từ các phần tử AND và XOR ( hình 7.2a ). Như vậy thiết kế bộ cộng có thể được biểu diễn theo hai lớp: lớp các bộ nửa tổng với phần tử OR và lớp chi tiết với các phần tử logic AND, OR, XOR.

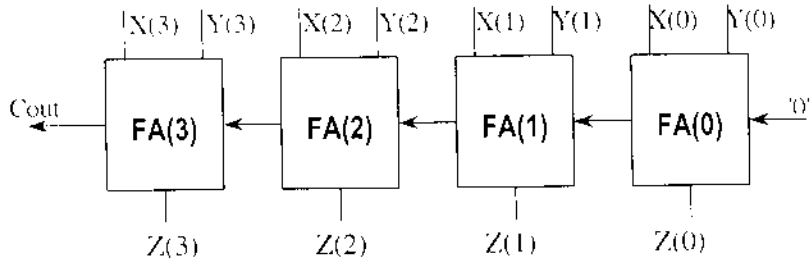
### 3. Câu lệnh Generate

Câu lệnh khởi tạo Generate là câu lệnh song song được định nghĩa bên trong các kiến trúc và được dùng để mô tả các phiên bản thành phần. Câu lệnh Generate có cấu trúc cú pháp như sau:

```
< nhãn_khởi_tạo > : < sơ_đồ_khởi_tạo > generate
{< các_câu_lệnh_song_song >}
end generate [< nhãn_khởi_tạo >];
```

Trong ngôn ngữ VHDL, có hai loại sơ đồ khởi tạo: sơ đồ khởi tạo **for** và sơ đồ khởi tạo **if**.

Sơ đồ **for** dùng để mô tả các cấu trúc có tính quy luật. Sơ đồ này đưa ra các tham số khởi tạo và các bước lặp. Các tham số khởi tạo không cần phải khai báo trước. Giá trị của chúng không thể gán hoặc truyền ra ngoài cấu trúc khởi tạo mặc dù VHDL cho phép sử dụng chúng bên trong cấu trúc.

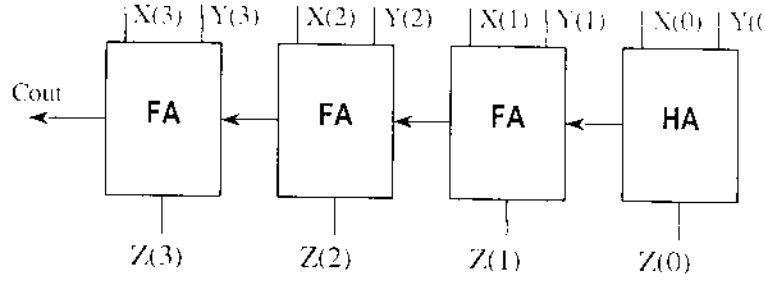



**Hình 7.3.** Xây dựng bộ cộng bốn bit từ các bộ cộng đầy đủ một bit.

Ví dụ, bộ cộng bốn bit có thể được xây dựng từ bốn bộ cộng đầy đủ một bit theo sơ đồ trên hình 7.3. Nếu sử dụng câu lệnh khởi tạo với sơ đồ **for** cấu trúc của bộ cộng bốn bit sẽ được mô tả như sau:

```
architecture Gen_For of FullAdder4 is
        signal X, Y, Z : STD_LOGIC_VECTOR ( 3 downto 0 );
        signal Cout    : STD_LOGIC;
        signal Tmp     : STD_LOGIC_VECTOR ( 4 downto 0 );
        component FullAdder
                port( A, B, C: in STD_LOGIC;
                      S, C0: out STD_LOGIC);
        end component;
        begin
            Tmp ( 0 ) <= '0';
            G: for I in 0 to 3 generate
                FA: FullAdder port map ( X( I ), Y( I ),
                        Tmp( I ), Z( I ), Tmp( I+1 ) );
                end generate;
            Cout <= Tmp( 4 );
        end Gen_For ;
```

Thông thường trong nhiều thiết kế sự xuất hiện và liên kết giữa các phần tử không tuân theo quy luật lặp giống trong ví dụ trên. Đối với những trường hợp này, ngôn ngữ VHDL cung cấp khả năng mô tả mạch sử dụng sơ đồ **if**. Khác với câu lệnh **if** tuần tự, câu lệnh **if** khởi tạo không chứa các nhánh **else** và **elsif**. Điều này có thể hiểu được vì trong một mạch phần cứng xác định không thể nào tồn tại lựa chọn cấu trúc dựa vào dữ liệu bên ngoài.




**Hình 7.4.** Xây dựng bộ cộng bốn bit từ ba bộ cộng đầy đủ một bit và một bộ nửa tổng một bit.

Ví dụ, chúng ta xây dựng bộ cộng bốn bit từ ba bộ cộng đầy đủ một bit và một bộ nửa tổng. Cấu trúc này tương tự cấu trúc trên hình 7.3, chỉ khác tại một điểm là không có đầu vào "0" cho phần tử đầu tiên. Nếu sử dụng cấu trúc **if**, cấu trúc của bộ cộng bốn bit sẽ được mô tả như sau:

```
architecture Gen_for of FullAdder4 is
        signal X, Y, Z : STD_LOGIC_VECTOR ( 3 downto 0 );
        signal Cout    : STD_LOGIC;
        signal Tmp     : STD_LOGIC_VECTOR ( 4 downto 1 );
        component HalfAdder
            port( A, B: in STD_LOGIC; S, C0: out STD_LOGIC );
        end component;
        begin
          G0: for I in 0 to 3 generate
            G1: if I = 0 generate
                HA: HalfAdder port map ( X( I ), Y( I ),
                                Z( I ), Tmp( I + 1 ) );
```

```
            end generate;
        G2 : if I > 1  and I <= 3 generate
            FA: HalfAdder port map ( X( I ), Y( I ),
                                Tmp( I ), Z( I ), Tmp( I + 1 ) );
            end generate;
    end generate;
  Cout <= Tmp(4) ;
end Gen_for;
```

Bất kỳ một cấu trúc song song của VHDL như các quá trình, các lệnh gán tín hiệu song song, các khối hoặc các lệnh gọi chương trình con song song và các lệnh khởi tạo khác có thể nằm bên trong các câu lệnh khởi tạo. Khi sử dụng các câu lệnh khởi tạo lồng nhau, chúng ta có thể xây dựng những cấu trúc có tính quy luật như sau:

```
Nested_Gen: block
    L1: for I in 0 to 3 generate
    L2: for J in 0 to 3 generate
    FA: CELL port map ( A( I ), B( I ), C( 2 * I + J), D( I + 2 * J ) );
    end generate
    end generate
end block Nested_Gen
```

## 4. Các cấu hình

Các khai báo thành phần và các phiên bản của thành phần chỉ chứa mô tả bên ngoài của một phần tử mạch trên một mức chi tiết nhất định. Các thực thể này không thể sử dụng được trong quá trình mô phỏng bởi vì chúng không chứa những thông tin mô tả về chức năng của phần tử. Như vậy, để có thể thực hiện mô phỏng hoạt động của phần tử, chúng ta cần một cơ chế liên kết các mô tả bên ngoài và các cấu trúc bên trong của phần tử. Trong ngôn ngữ VHDL, cơ chế liên kết này được thể hiện trong các *cấu hình* của phần tử. Chúng ta hãy xét ví dụ mô tả cấu trúc của bộ cộng đầy đủ một bit FullAdder trên hình 7.2. Bộ cộng này được mô tả bằng đoạn chương trình sau:

```
libraly IEEE;
use IEEE.STD_LOGIC_1164.all;
entity FullAdder is
    port ( A, B, Cin: in STD_LOGIC; Sum, Cout: out STD_LOGIC);
end FullAdder;
architecture IMP1 of FullAdder is
    component Xorgate;
            port ( I0, I1: in STD_LOGIC; O: out STD_LOGIC );
    end component;
    component And2gate;
            port( I0, I1: in STD_LOGIC; O: out STD_LOGIC );
    end component;
    component Or2gate;
            port ( I0, I1: in STD_LOGIC; O: out STD_LOGIC );
    end component;
    signal S1, S2, S3 : STD_LOGIC;
    begin
            U1 : Xorgate port map ( A, B, S1 );
            U2 : And2gate port map ( A, B, S2 );
            U3 : And2gate port map ( Cin, S1, S3 );
            U4 : Xorgate port map ( Cin, S1, Sum );
            U5 : Or2gate port map ( S2, S3, Cout );
     end IMP1;
```

Trong đoạn chương trình mô tả ở trên, chúng ta chỉ khai báo về kiểu của thành phần và những phương thức kết nối giữa chúng mà không chỉ rõ các chức năng của từng thành phần. Như vậy, để thực hiện và mô phỏng đoạn chương trình nói trên, chúng ta cần phải mô tả hành vi của từng thành phần. Trong đoạn chương trình dưới đây, chúng ta mô tả những thực thể tham gia vào cấu trúc của bộ cộng đầy đủ một bit. Các mô tả thực thể này bao gồm phần khai báo thực thể và mô tả hành vi.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity And2gate is
```

```
        port ( I0, I1: in STD_LOGIC; O: out STD_LOGIC );
end And2gate;
architecture BHV of And2gate is
begin
        O <= I0 and I1;
end BHV;

library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity Or2gate is
        port ( I0, I1: in STD_LOGIC; O: out STD_LOGIC );
end Or2gate;
architecture BHV of Or2gate is
begin
        O <= I0 or I1;
end BHV;

libraly IEEE;
use IEEE.STD_LOGIC_1164.all;
entity Xorgate is
        port ( I0, I1: in STD_LOGIC; O: out STD_LOGIC );
end Xorgate;
architecture BHV of Xorgate is
begin
        O <= I0 xor  I1;
end BHV;
```

Sau khi tất cả các thực thể và kiến trúc của từng phần tử được biên dịch và chứa trong các thư viện, chúng ta có thể tiến hành biên dịch mô hình của bộ cộng đầy đủ một bit FullAdder nêu trên. Trong khi biên dịch thiết kế của bộ cộng đầy đủ, chương trình dịch sẽ đối sánh từng định danh của các thành phần trong mô tả thiết kế với các thực thể trong thư viện đang sử dụng. Nếu ứng với mỗi thành phần trong thiết kế có thể tìm thấy thực thể có cùng tên trong các thư viện, chương trình biên dịch sẽ xây dựng thực thể mô phỏng cho mô hình thiết kế của bộ cộng FullAdder. Ví dụ: [illegible] kiến trúc với tên

IMP1 của bộ cộng FullAdder có ba thành phần là Xorgate, Or2gate và And2gate. Nếu trong thư viện WORK chứa các thực thể có cùng tên, chương trình mô phỏng sẽ gắn kết kiến trúc được biên dịch sau cùng ( trong trường hợp thực thể có nhiều hơn một kiến trúc ) với những thành phần tương ứng. Như vậy nhà thiết kế không nhất thiết phải đặc tả một cách tường minh sự gắn kết những phiên bản của thành phần và các cặp thực thể - kiến trúc chứa trong thư viện thiết kế. Chúng ta gọi sự gắn kết này là gắn kết mặc định.

Trong trường hợp một thực thể có nhiều kiến trúc thì việc chọn mặc định kiến trúc được dịch sau cùng có thể không lựa chọn được kiến trúc cần thiết, do đó cần phải biểu diễn cấu hình một cách tường minh.

## 5. Biểu diễn cấu hình

Một thực thể có thể có nhiều kiến trúc. Mỗi kiến trúc có thể là mô hình thuật toán, kiến trúc khác có thể là mức thanh ghi truyền đạt hoặc có thể là mô hình cấu trúc. Trên khía cạnh thiết kế, chúng ta cần lựa chọn các kiến trúc của phần tử một cách thích hợp và phải chỉ rõ các khai báo thực thể và các kiến trúc sẽ được lựa chọn. Như vậy, một cấu hình sẽ cung cấp cho chúng ta cách lựa chọn những khai báo thực thể và kiến trúc đối với từng phiên bản của thành phần được mô tả trong thân kiến trúc của thiết kế.

Cấu hình của một đặc tả thành phần được biểu diễn theo quy tắc cú pháp sau:

**for** < danh_sách_phiên_bản >: < tên_thành_phần > **use** < đặc_tả_gắn_kết >;

Trong đó:

< danh_sách_phiên_bản > chỉ ra danh sách phiên bản của thành phần được đặt cấu hình( có tên được chỉ ra bằng < tên_thành_phần > ). Phiên bản có thể được đặc tả bằng < nhãn_phiên_bản > hoặc dùng các từ khóa **others** hoặc **all**.

- Từ khóa **others** chỉ tới các phiên bản của thành phần có tên là < tên_thành_phần > và chưa được đặt cấu hình.
- Từ khóa **all** chỉ tới mọi phiên bản của thành phần có tên là < tên_thành_phần >.

< đặc_tả_gắn_kết > đưa ra các ánh xạ giữa các kiến trúc được lựa chọn, dựa theo đó phiên bản sẽ được xây dựng. Thông tin này sẽ được mô tả theo quy tắc cú pháp sau:

**entity** < tên_thư_viện.tên_thực_thể > [ ( < tên_kiến_trúc > ) ];

Nếu thực thể chỉ có một kiến trúc thì tên của kiến trúc trong phần < đặc_tả_gắn_kết > có thể được bỏ qua.
Ví dụ, ta có thể đặc tả bộ cộng đầy đủ bằng cách sử dụng các cấu hình như sau:

```
libraly IEEE;
use IEEE.STD_LOGIC_1164.all;
entity FullAdder is
        port( A, B, Cin: in STD_LOGIC;
              Sum, Cout: out STD_LOGIC);
end FullAdder ;
architecture IMP1 of FullAdder is
  component Xorgate2;
       port ( I0, I1: in STD_LOGIC; O: out STD_LOGIC );
  end component;
  component Andgate2;
       port ( I0, I1: in STD_LOGIC; O: out STD_LOGIC );
  end component;
  component Orgate2;
       port ( I0, I1: in STD_LOGIC; O: out STD_LOGIC );
  end component;
  signal S1, S2, S3 : STD_LOGIC;
  for U1: Xorgate2 use entity work.Xorgate( BHV );
  for others: Xorgate2 use entity work.Xorgate(BHV) ;
  for all: Andgate2 use entity work.And2gate;
  for U5: Orgate2 use entity work.Or2gate;
begin
  U1: Xorgate2 port map ( A, B, N1 );
  U2: Andgate2 port map ( A, B, N2 );
  U3: Andgate2 port map ( Cin, N1, N3 );
  U4: Xorgate2 port map ( Cin, N1, Sum );
  U5: Orgate2 port map ( N3, N2, Cout );
end IMP1;
```

Trong ví dụ này, phiên bản có nhãn U1 của phần tử Xor sẽ có mô tả tương ứng với thực thể Xor2gate chứa trong thư viện WORK. Việc thiết lập liên kết giữa phiên bản và mô tả thực thể được thể hiện tường minh bằng câu

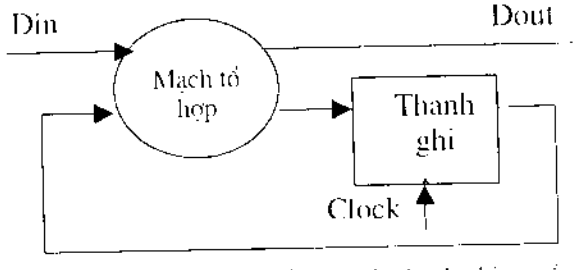
lệnh gắn kết **for**. Các phần tử Xor có nhãn khác U1 sẽ được gắn kết theo dòng lện **for others** tiếp theo. Tất cả các phần tử Andgate2 sẽ sử dụng cấu hình tương ứng với thực thể And2gate. Đối với phần tử Or việc thực hiện gắn kết cấu trúc cũng xảy ra tương tự.

## §7.2. Mô hình hóa trên mức thanh ghi truyền đạt

Một thiết kế trên mức thanh ghi truyền đạt bao gồm một tập hợp các thanh ghi liên kết với các mạch tổ hợp. Trong mục này, chúng tôi sẽ trình bày mối quan hệ giữa các cấu trúc trên mức thanh ghi trong ngôn ngữ VHDL và mạch lôgic sẽ được tổng hợp.

### 1. Mô hình hóa mạch tổ hợp

Một quá trình không chứa câu lệnh **if** với tín hiệu điều khiển theo sườn lên hoặc sườn xuống hoặc không chứa câu lệnh **wait** với các sự kiện của tín hiệu gọi là quá trình tổ hợp. Một quá trình tổ hợp sẽ được tổng hợp bằng các mạch tổ hợp. Ví dụ sau đây sẽ mô tả mạch tổ hợp tính giá trị nhớ của bộ cộng đầy đủ với các đầu vào A, B, Cin.



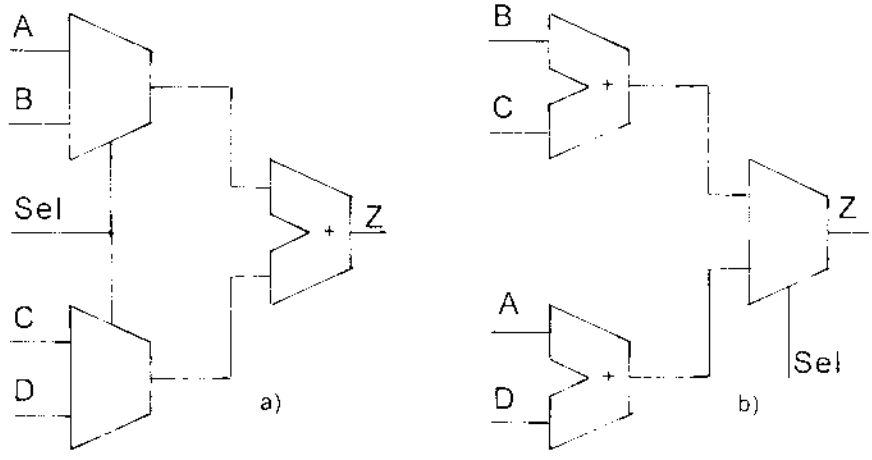
**Hình 7.5.** Biểu diễn thiết kế trên mức thanh ghi truyền đạt.

```
signal A, B, Cin, Cout: BIT;
process( A, B, Cin )
begin
    Cout <= ( A and B ) or ( ( A or B ) and Cin );
end;
```

Trong ví dụ này, tất cả các tín hiệu vào phải được ghi trong danh sách tín hiệu tác động. Danh sách tín hiệu tác động chỉ ra rằng, quá trình tính toán đang xét sẽ được thực hiện khi có sự thay đổi của các tín hiệu nằm trong danh sách.

Để mô tả mạch lôgic tổ hợp, các biến và tín hiệu trong một quá trình **process** không được nhận giá trị gán khởi tạo trước bởi vì mạch tổ hợp không chứa các phần tử nhớ. Khi trong mô hình mạch có các biến hoặc tín hiệu được khởi tạo giá trị trước, điều này sẽ tương đương với việc trong mạch phải có những phần tử lưu trữ các giá trị khởi tạo. Như vậy khi mô hình hóa cấu trúc, chương trình mô phỏng sẽ sinh ra các phần tử nhớ để lưu trữ các giá trị khởi tạo. Mạch trở thành mạch có nhớ. Thêm vào đó, trong các mô hình mạch tổ hợp, các tín hiệu và biến cần phải được gán giá trị trước khi được sử dụng.

Mọi câu lệnh tuần tự trừ các lệnh **wait**, **loop** và **if** với những tín hiệu điều khiển theo sườn đều có thể dùng để mô tả các mạch lôgic tổ hợp. Các phép toán số học như +, -, *, ...; các phép toán quan hệ và các phép toán lôgic đều có thể được sử dụng trong biểu thức.




**Hình 7.6.** Hai kết quả tổng hợp mạch khác nhau.

Công cụ tổng hợp có thể thực hiện chia sẻ tài nguyên nếu có các thao tác loại trừ trong biểu diễn thiết kế. Ví dụ, hai phép toán cộng trong trong đoạn chương trình dưới đây là hai phép cộng loại trừ vì chúng không thể thực hiện đồng thời. Chúng ta có thể gán các phép toán đó cho hai khối chức năng khác nhau hoặc gán cho cùng một khối chức năng tùy thuộc vào các ràng buộc về tài nguyên. Nếu điều kiện ràng buộc về tài nguyên yêu cầu sử dụng

một bộ cộng, mạch tổng hợp được sẽ có dạng trên hình 7.6a. Tuy nhiên, nếu điều kiện ràng buộc yêu cầu hai bộ cộng, mạch nhận được sẽ có dạng trên hình 7.6b.

Ví dụ,

```
process ( A, B, Sel )
begin
    if ( Sel = '1' )
            then Z <= B + C;
            else Z <= A + D;
    end if;
end process;
```

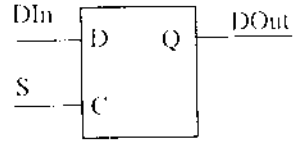
Các phép gán tín hiệu song song có thể được sử dụng ngắn gọn đối với một số loại mạch lôgic tổ hợp. Ví dụ, trong quá trình sinh ra giá trị nhớ của bộ cộng đầy đủ một bit có thể được viết dưới dạng một phép gán tín hiệu song song như trong đoạn chương trình dưới đây.

Ví dụ,

```
architecture DataFlow of FullAdder is
        signal A, B, Cin, Cout: BIT;
        .....
        Cout <= ( A and B ) or ( ( A or B ) and Cin );
end DataFlow;
```

## 2. Các mạch trigơ điều khiển theo mức ( mạch lật )

Các mạch lật là các trigơ làm việc với chế độ đồng bộ theo mức. Các trigơ làm việc theo sườn và trigơ làm việc theo mức thường được sử dụng trong các phần tử nhớ một bit. Ta xét ví dụ mô tả mạch trigơ D làm việc theo mức.




**Hình 7.7.** Mạch tri-gơ D làm việc với chế độ đồng bộ theo mức.

Ví dụ, mô tả hành vi mạch trigơ D làm việc theo mức bằng ngôn ngữ VHDL:

```
signal S, DIn, DOut: BIT;
.....
process ( S, DIn )
begin
        if ( S = '1' ) then
            DOut ⇐ Din;
        end if;
end process;
```

Chúng ta thấy, danh sách các tín hiệu nhạy cảm chứa các tín hiệu S và DIn. Các tín hiệu này cần thiết để phần tử trigơ D hoạt động đúng. Như vậy, khi tín hiệu trên đường tín hiệu S và DIn thay đổi giá trị, quá trình sẽ được chương trình mô phỏng thực hiện. Chúng ta thấy việc gán giá trị cho tín hiệu DOut được ẩn trong câu lệnh điều kiện **if**, giá trị tín hiệu trên đường tín hiệu DOut sẽ không thay đổi nếu giá trị trên đường S bằng '0'. Nếu giá trị trên đường tín hiệu S bằng '1', giá trị trên đường tín hiệu DOut sẽ thay đổi và bằng DIn mỗi khi tín hiệu trong danh sách tín hiệu nhạy cảm thay đổi giá trị. Đó là hành vi hoạt động của phần tử trigơ làm việc theo mức. Như vậy, đối với các mạch trigơ làm việc theo mức, chúng ta phải chỉ rõ danh sách các tín hiệu tác động. Khi các tín hiệu tác động thay đổi thì quá trình mô tả hành vi của mạch sẽ được thực hiện.

Nói chung, hành vi của các mạch trigơ làm việc theo mức sẽ được xây dựng trên những câu lệnh điều kiện **if** không đầy đủ. Có thể hiểu câu lệnh điều kiện **if** không đầy đủ là câu lệnh **if** chỉ có một nhánh **then** và không chứa nhánh **else**. Như vậy, mọi tín hiệu hoặc biến không được điều khiển bởi tất cả các khả năng có thể có của điều kiện đều được mô phỏng thành những phần tử trigơ làm việc theo mức. Các phần tử trigơ làm việc theo mức thường xuất hiện trong quá trình mô phỏng khi gặp các lệnh **if** và **case** có cấu trúc không đầy đủ. Đoạn chương trình trong ví dụ vừa nêu trên tương ứng với phần tử trigơ làm việc theo mức vì được biểu diễn bằng câu lệnh **if** có cấu trúc không đầy đủ.

Để tránh sự xuất hiện của các phần tử trigơ làm việc theo mức không mong muốn, chúng ta phải gán tín hiệu với tất cả các khả năng có thể có của điều kiện trong các câu lệnh rẽ nhánh. Ví dụ, nếu ta bổ sung các nhánh vào câu lệnh điều kiện trong đoạn chương trình nêu trên, câu lệnh **if** sẽ được biểu

diễn đầy đủ. Khi đó đoạn chương trình sẽ được chương trình mô phỏng tổng hợp như một phần tử AND.

Ví dụ,

```
signal S, DIn, DOut: BIT;
process ( S, DIn )
begin
        if ( S = '1' ) then
                DOut <= DIn;
        else
                DOut <= '0';
        end if
end process;
```

Chúng ta có thể mô tả những phần tử trigơ làm việc theo mức với hai đầu tín hiệu thiết lập giá trị '0' hoặc '1' không đồng bộ. Đoạn chương trình dưới đây sẽ biểu diễn phần tử trigơ làm việc theo mức có giá trị sẽ thiết lập về '0' khi tín hiệu đầu vào không đồng bộ RST nhận giá trị bằng ' '. Như vậy trong ví dụ này, tín hiệu RST là tín hiệu kích hoạt với với mức gi. trị tín hiệu cao.

Ví dụ, nếu RST =1 ⇒ thiết lập trạng thái của trigơ về '0'.

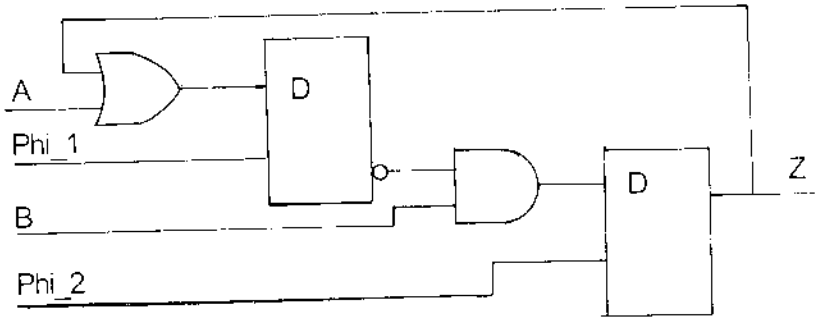
```
signal S, RST, DIn, DOut: BIT;
process ( S, RST, DIn )
begin
        if ( RST = '1' ) then
                DOut ⇐ '0';
        elsif ( S = '1' ) then
                DOut <= DIn;
        end if;
end process;
```

Nếu chúng ta muốn đổi tín hiệu RST thành tín hiệu kích h ạt với mức giá trị thấp, điều kiện trong câu lệnh **if** sẽ chuyển từ ( RST = '1' ) thành ( RST = '0' ).

### 3. Xây dựng mạch đồng hồ hai pha

Các mạch đồng hồ hai pha có thể được mô tả bằng cách sử dụng các trigơ làm việc theo mức. Mạch đồng hồ hai pha được biểu diễn bang hai quá trình. Trong đó, một quá trình mô tả mạch tổ hợp và mạch lật tầng một, quá trình khác mô tả mạch tổ hợp và mạch lật tầng thứ hai.

Ví dụ, thiết kế của mạch đồng hồ hai pha sẽ được mô tả bằng đoạn chương trình trên ngôn ngữ VHDL như sau:




**Hình 7.8.** Thiết kế của mạch đồng hồ hai pha.

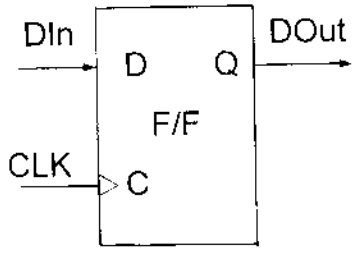
```
entity TwoPhase is
        port( A, B: in BIT; Ph1, Ph2: in BIT; Z: buffer BIT );
end TwoPhase;
architecture Implement of TwoPhase is
        signal D: BIT;
begin
        process( A, Z, Phi_1 )
        begin
                if ( Phi_1 = '1' ) then
                        D <= A or Z;
                end if;
        end process;
        process( B, D, Phi_2 )
        begin
                if ( Phi_2 = '1' ) then
                        Z <= B and ( not D );
```

```
            end if
        end process;
    end Implement;
```

### 4. Các mạch trigơ làm việc theo sườn ( flip-flop )

Các quá trình chứa các tín hiệu **if** hoặc **wait** điều khiển theo sườn lên ( hoặc sườn xuống ) là các quá trình được định giờ. Các mạch trigơ điều khiển theo sườn lên hoặc sườn xuống của tín hiệu sẽ được tạo ra từ mô tả trên ngôn ngữ VHDL nếu phép gán tín hiệu ( hoặc phép gán biến ) được thực hiện theo sườn lên hoặc sườn xuống của các tín hiệu điều khiển. Khi tìm ra sườn của tín hiệu đồng hồ, chương trình mô phỏng có thể xác định được vị trí để đưa phần tử trigơ vào mô hình. Như vậy mô hình nhận được sẽ hoạt động theo dự đoán trong giai đoạn thiết kế.

Thuộc tính **event** của tín hiệu được dùng để biểu diễn sự biến thiên của tín hiệu. Khi chúng ta cần xác định sự biến thiên của tín hiệu, thuộc tính **event** sẽ cho ra giá trị lôgic tuỳ theo trên đường tín hiệu có xuất hiện sự kiện hay không. Thuộc tính **stable** cũng cho giá trị lôgic và có ý nghĩa ngược lại với thuộc tính **event**.




**Hình 7.9.** Trigơ D điều khiển theo sườn tín hiệu đồng hồ.

Trong ví dụ dưới đây, chúng ta mô tả trigơ D điều khiển theo sườn lên bằng ngôn ngữ VHDL. Biểu diễn của mạch bao gồm một quá trình **process** có danh sách tín hiệu tác động chứa tín hiệu đồng hồ CLK. Quá trình này chứa một câu lệnh **if** có biểu thức điều kiện kiểm tra sườn lên của tín hiệu đồng hồ CLK. Trong thời gian tín hiệu thiết lập sườn lên, giá trị trên đường tín hiệu DIn được gán cho đường tín hiệu DOut.

Ví dụ mô tả hoạt động của trigơ D làm việc theo sườn lên của tín hiệu đồng hồ CLK.

```
signal CLK, DIn, DOut: BIT;
.....
process( CLK )
begin
```

```
        if ( CLK'event and CLK = '1' ) then
            DOut <= DIn;
        end if;
    end process;
```

Trong các chương trình trên ngôn ngữ VHDL, các biến cũng có thể sinh ra các trigơ điều khiển theo sườn tín hiệu. Chúng ta biết rằng, khi một biến được khai báo trong một quá trình **process**, giá trị của biến sẽ không rời khỏi quá trình (có nghĩa là giá trị của biến được khai báo bên trong một quá trình sẽ không được sử dụng ở bên ngoài quá trình). Như vậy, thời điểm mà một biến sẽ sinh ra một trigơ điều khiển theo sườn trong quá trình mô phỏng là thời điểm khi biến được sử dụng trước lúc được gán giá trị bên trong quá trình được điều khiển theo tín hiệu đồng hồ. Trong ví dụ dưới đây, đoạn mã chương trình VHDL sẽ sinh ra hai phần tử trigơ làm việc theo sườn khi được mô phỏng.

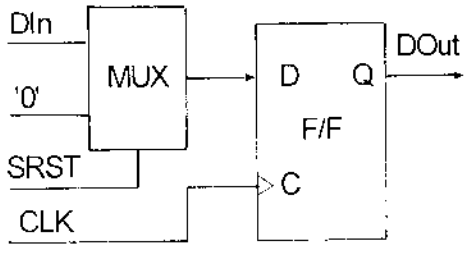
Ví dụ: đoạn chương trình tạo ra hai phần tử trigơ điều khiển theo sườn trong quá trình mô phỏng.

```
signal CLK, DIn, DOut: BIT;
process(CLK)
variable TMP: BIT;
begin
        if (CLK'event and CLK = '1') then
            DOut <= TMP;
            TMP := DIn;
        end if;
end process;
```

Trong ví dụ này, biến TMP được sử dụng trước khi được gán giá trị. Như vậy, giá trị chứa trong biến TMP là giá trị có được sau lần lặp trước của trình tự thực hiện quá trình và chúng ta cần phải có một trigơ để lưu trữ giá trị này. Nếu chúng ta đổi chỗ hai phép gán trong đoạn chương trình nói trên, biến TMP sẽ tương ứng đơn thuần với một dây dẫn. Khi đó chỉ có một phần tử trigơ sẽ được tạo ra trong quá trình mô hình hóa mạch.

## 5. Thiết lập và xóa giá trị đồng bộ và không đồng bộ trong các mạch trigơ điều khiển theo sườn tín hiệu( flip-flop )

Giá trị của các phần tử flip-flop có thể được thiết lập hoặc xoá một cách đồng bộ dựa vào sự xuất hiện tín hiệu trên các đầu vào tương ứng của phần tử. Phép gán giá trị chỉ được thực hiện khi xuất hiện sườn của tín hiệu đồng bộ. Còn trong các thời điểm không có tín hiệu đồng bộ, sự thay đổi giá trị tín hiệu trên các đầu thiết lập hoặc xóa không ảnh hưởng tới trạng thái của phần tử nhớ.




**Hình 7.10.** Mạch D flip-flop với đầu xóa trạng thái đồng bộ.

Tất cả các phép gán bên trong câu lệnh **if** phản ứng với sườn( lên hoặc xuống ) của tín hiệu sẽ có tác dụng thiết lập trạng thái của phần tử nhớ trong toàn bộ thời gian hình thành sườn tín hiệu đồng hồ. Phần tử flip-flop với các đầu thiết lập hoặc xóa trạng thái có thể được mô hình hóa bằng cách xác định thời điểm đầu vào đồng bộ sẽ được thiết lập theo sự xuất hiện sườn tín hiệu đồng hồ. Đoạn chương trình dưới đây sẽ tương đương với mạch flip-flop có cấu trúc được mô tả trên hình 7.10.

Ví dụ:

```
signal CLK, DIn, DOut, SRST: BIT;
process( CLK );
        if ( CLK'event and CLK = '1' ) then
                if ( SRST = '1' ) then
                        DOut ← '0';
                else
                        DOut ← DIn;
                end if
        end if;
end process;
```

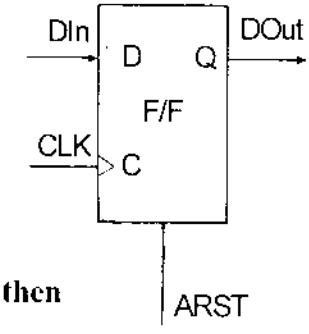
Trong đoạn chương trình trên, chúng ta thấy các tín hiệu DIn và SRST không cần thiết phải nằm trong danh sách các tín hiệu tác động. Các phép

gán chỉ xảy ra khi xuất hiện sự thay đổi giá trị của tín hiệu đồng hồ CLK (được kiểm tra bằng điều kiện CLK'**event**) từ '0' sang '1' (được kiểm tra bằng điều kiện CLK = '1').

Trong nhiều trường hợp, việc thiết lập hoặc xoá giá trị của phần tử nhớ không phụ thuộc vào thời gian đồng hồ. Ví dụ, trong các phần tử trigơ D điều khiển theo sườn tín hiệu (D flip-flop) đối với nhiều trường hợp, cần thiết thiết lập giá trị hoặc xoá giá trị không phụ thuộc vào thời gian đồng hồ. Đoạn chương trình VHDL dưới đây mô tả phần tử D flip-flop có các đầu vào xoá giá trị không đồng bộ.

Ví dụ D flip-flop với đầu xoá giá trị không đồng bộ:

```
signal CLK, ARST, DIn, DOut: BIT;
    ..................
    process( CLK, ARST )
    begin
        if ( ARST = '1' ) then
            DOut ← '0';
        elsif ( CLK'event and CLK = '1' ) then
            DOut ← DIn;
        end if;
    end process;
```




**Hình 7.11.** Mạch D flip-flop với đầu xóa trạng thái không đồng bộ.

Trong ví dụ này, chúng ta thấy thực thể được mô tả có một cổng vào bổ sung – cổng xoá giá trị không đồng bộ ARST so với trường hợp xoá giá trị đồng bộ. Các tín hiệu CLK và ARST phải nằm trong danh sách tín hiệu điều khiển. Mỗi khi có một sự kiện xuất hiện trên các đường tín hiệu CLK hoặc ARST, đoạn chương trình trong thân của **process** sẽ được thực hiện. Trong ví dụ này, tín hiệu ARST có mức độ ưu tiên cao hơn tín hiệu CLK. Khi tín hiệu trên đường ARST nhận giá trị '1', đầu ra của D flip-flop nhận giá trị '0'. Trong những trường hợp khác, D flip-flop sẽ gán giá trị tín hiệu đầu vào DIn cho đầu ra DOut tương ứng với sườn lên của tín hiệu đồng hồ CLK.

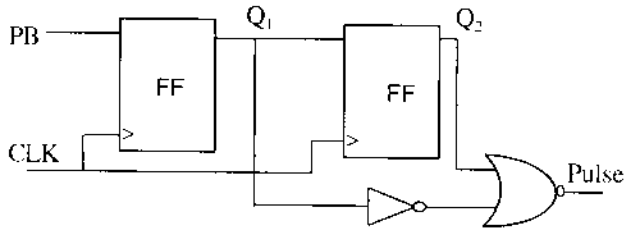
### 6. Sơ đồ chung mô tả mạch có nhớ

Nếu xét trên quan điểm cấu trúc và chức năng, mạch có nhớ luôn được chia thành hai phần: các thành phần tổ hợp và các mạch nhớ đồng bộ. Tương ứng với cách phân chia chức năng và cấu trúc như vậy, mô hình của mạch có nhớ bao gồm hai phần: một phần mô tả các cấu trúc và chức năng tổ hợp; một phần mô tả các cấu trúc và chức năng mạch nhớ đồng bộ.

- Trong phần mô tả các thành phần tổ hợp, chúng ta biểu diễn những thành phần mạch có hành vi phụ thuộc vào sự thay đổi giá trị tín hiệu nằm trong danh sách tín hiệu tác động. Tất cả các tín hiệu được tham chiếu tới trong phần mô tả các thành phần tổ hợp phải được liệt kê trong danh sách các tín hiệu tác động. Các thao tác đối với tín hiệu trong phần này không đi kèm với các phép kiểm tra sự xuất hiện sườn tín hiệu ( tức là không chứa các thuộc tính '**event** ).
- Trong phần mô tả các mạch nhớ đồng bộ, chúng ta biểu diễn các thành phần mạch có hành vi phụ thuộc vào sự xuất hiện các sườn của tín hiệu đồng bộ. Các thao tác đối với tín hiệu trong phần này chứa các phép kiểm tra sự kiện trên đường tín hiệu và kiểm tra sườn tín hiệu ( như CLK'**event and** CLK = '1' ).

Ví dụ, đoạn chương trình sau mô tả mạch tạo xung có sơ đồ thiết kế ở mức lôgic như trên hình 7.12.

```
entity Pulser is
    port ( CLK, PB: in BIT; Pulse: out BIT );
end Pulser;
```




**Hình 7.12.** Mạch tạo xung.

```
architecture BHV of Pulser is
        signal Q1, Q2: BIT;
begin
        process ( CLK, Q1, Q2 )
        begin
```

```
            if ( CLK'event and CLK = '1' ) then
                Q1 <= PB;
                Q2 <= Q1;
            end if;
            Pulse <= ( not Q1 ) nor Q2;
        end process;
    end BHV;
```

Trong ví dụ này, phần chức năng tổ hợp được biểu diễn bằng phép gán:

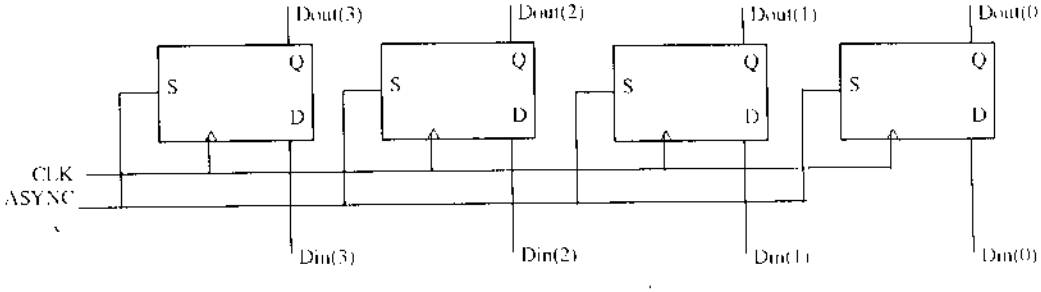
```
Pulse <= ( not Q1 ) nor Q2;
```

Còn phần chức năng nhớ đồng bộ theo sườn lên được biểu diễn bằng câu lệnh **if** kiểm tra sự kiện:

```
if ( CLK'event and CLK = '1' ) then
    Q1 <= PB;
    Q2 <= Q1;
end if;
```

## 7. Các thanh ghi

Các dạng thanh ghi khác nhau cũng thường được sử dụng trong các mạch có nhớ. Trên hình 7.13 chúng ta biểu diễn sơ đồ của thanh ghi bốn bit làm việc trong chế độ đồng bộ bằng tín hiệu đồng hồ CLK. Thanh ghi được thiết lập giá trị ban đầu bằng "1111" khi xuất hiện tín hiệu thiết lập không đồng bộ ASYNC. Khi xuất hiện sườn lên của tín hiệu đồng bộ CLK, các bit




**Hình 7.13.** Sơ đồ cấu trúc thanh ghi 4-bit.

đầu vào của thanh ghi được truyền tới đầu ra đồng thời. Với thanh ghi như

trên hình 7.13, chúng ta có đoạn chương trình trên ngôn ngữ VHDL tương ứng mô tả hành vi của mạch.

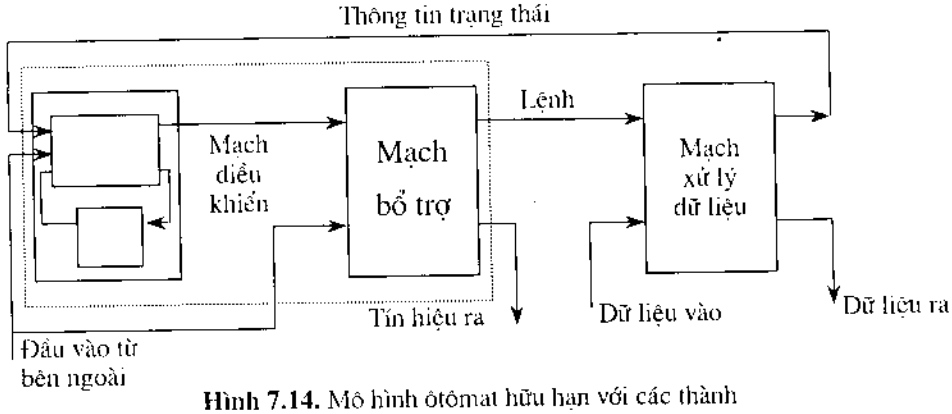
Ví dụ mô tả hành vi của thanh ghi bốn bit theo sơ đồ trên hình 7.13.

```
signal CLK, ASYNC: BIT;
signal DIn, DOut: BIT_VECTOR ( 3 downto 0 );
process ( CLK, ASYNC )
begin
        if ( ASYNC = '1' ) then
                DOut <= "1111";
        elsif ( CLK'event and CLK = '1' ) then
                DOut <= DIn;
        end if;
end process;
```

Với sơ đồ chung để mô tả các mạch có nhớ, chúng ta có thể biểu diễn các dạng thanh ghi khác trên ngôn ngữ VHDL một cách tương tự.

## §7.3. Mô hình hóa các ôtômat hữu hạn

Một thiết kế mạch số có thể được chia làm hai thành phần: bộ xử lý dữ




**Hình 7.14.** Mô hình ôtômat hữu hạn với các thành phần điều khiển và thành phần xử lý dữ liệu.

liệu và bộ điều khiển. Mối quan hệ giữa bộ điều khiển và bộ xử lý dữ liệu trong mạch được biểu diễn trên hình 7.14. Bộ xử lý dữ liệu thực hiện các thao tác đối với dữ liệu chứa trong các phần tử nhớ theo các lệnh do bộ điều khiển đưa ra. Bộ điều khiển đưa ra các lệnh điều khiển thích hợp cho bộ xử lý dữ liệu tại mỗi thời điểm thời gian. Với những lệnh điều khiển từ bộ điều khiển bộ xử lý dữ liệu có thể thực hiện những chức năng và thao tác thích hợp để xác định được các tín hiệu đầu ra cần thiết. Bộ điều khiển nhận những thông tin phản hồi từ bộ xử lý dữ liệu - các thông tin trạng thái xử lý dữ liệu. Các thông tin phản hồi này được sử dụng làm các biến quyết định để xác định dãy chuyển trạng thái của mạch.

Bộ điều khiển là những mạch tuần tự có các trạng thái được dùng để xác định các lệnh điều khiển cho hệ thống. Tại một trạng thái hiện thời, dựa vào những thông tin trạng thái và thông tin đầu vào (thông tin đầu vào do bộ xử lý dữ liệu cung cấp hoặc là các tín hiệu thiết lập từ bên ngoài) bộ điều khiển chuyển sang trạng thái mới và khởi tạo các lệnh điều khiển mới và tạo ra các giá trị ra tương ứng. Bộ điều khiển thường được xây dựng từ các mạch tuần tự - các thanh ghi trạng thái và các mạch tổ hợp. Các thanh ghi trạng thái lưu giữ các trạng thái hiện thời, còn mạch tổ hợp tạo ra các lệnh điều khiển và trạng thái mới dựa trên cơ sở trạng thái hiện thời và các tín hiệu đầu vào. Các mạch tuần tự có một số hữu hạn các trạng thái gọi là các ôtômat hữu hạn. Các ôtômat hữu hạn là những công cụ hữu hiệu để thực hiện các cơ chế điều khiển và quyết định trong các mạch số.

Máy ôtômat hữu hạn là một bộ sáu $< X, Y, S, s_0, \delta, \lambda >$, trong đó:

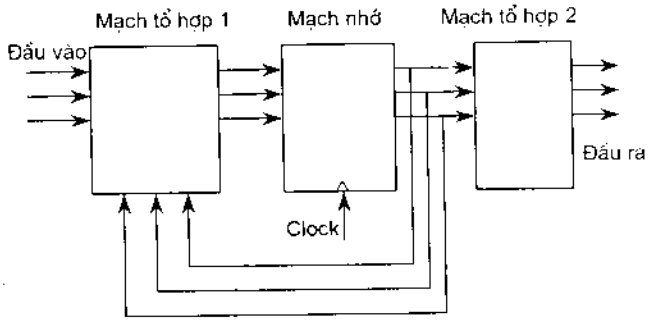
- X - tập hợp các tín hiệu vào của ôtômat:
  $X = \{ x_1(t),..., x_n(t) \}$
- Y - tập hợp các tín hiệu ra của ôtômat:
  $Y = \{ y_1(t),..., y_m(t) \}$
- S - tập hợp các trạng thái của ôtômat:
  $S = \{ s_1(t),..., s_s(t) \}$
- $s_0$ - trạng thái ban đầu của ôtômat:
  $s_0 (t) \in S$
- Hàm $\delta ( s, x )$ - hàm chuyển trạng thái của ôtômat;
- Hàm $\lambda ( s, x )$ - hàm đầu ra của ôtômat.

Tương ứng với các phương pháp tính toán hàm chuyển trạng thái và hàm ra, chúng ta có các loại ôtômat khác nhau. Hai dạng ôtômat hữu hạn thông

dụng là ôtômat Moore và ôtômat Mealy. Chúng ta sử dụng ngôn ngữ VHDL để mô tả hai mô hình ôtômat này.

## 1. Mô hình hóa ôtômat Moore

Trong chương 3, chúng ta đã mô tả hình thức ôtômat Moore, do đó tại chương này, chúng ta sẽ chỉ nhắc lại một cách ngắn gọn.




**Hình 7.15.** Sơ đồ khối của ôtômat Moore.

Ôtômat Moore là một ôtômat hữu hạn có hàm chuyển trạng thái và hàm ra có dạng như sau:

$s(t) = \delta(s(t-1), x(t))$

$y(t) = \lambda(s(t))$ $t = 1, 2, ...$

Như vậy trong ôtômat Moore, tín hiệu đầu ra ở thời điểm hiện thời chỉ phụ thuộc vào trạng thái hiện thời của ôtômat; còn trạng thái ở thời điểm hiện thời sẽ được tính thông qua tín hiệu đầu vào tại thời điểm hiện thời và trạng thái trước đó của ôtômat. Theo sơ đồ khối trên hình 7.15, ôtômat Moore có thể được biểu diễn bao gồm một mạch tổ hợp dùng để xác định trạng thái mới của ôtômat thông qua tín hiệu vào và trạng thái trước đó; một hệ mạch nhớ để lưu giữ trạng thái. Đối với ôtômat Moore, tín hiệu đầu ra chỉ phụ thuộc vào trạng thái hiện thời, do đó, chúng ta cần một mạch tổ hợp nữa để xác định tín hiệu ra. Mạch tổ hợp xác định tín hiệu ra sẽ được nối trực tiếp với hệ mạch nhớ trạng thái bởi vì tín hiệu ra chỉ phụ thuộc vào trạng thái của ôtômat ở thời điểm hiện thời. Các phần tử nhớ của ôtômat Moore thường được điều khiển bằng tín hiệu đồng hồ, do đó các tín hiệu ra cũng được đồng bộ hóa theo đồng hồ. Như vậy ôtômat Moore là một ôtômat với đầu ra đồng bộ. Vì lẽ đó chúng ta không cần quan tâm tới những khó khăn có thể xuất hiện do các quá trình quá độ hoặc cạnh tranh giữa các phần tử trong mạch của ôtômat gây ra (những vấn đề này đã được đề cập tới trong chương 3). Các tín hiệu ra được tính qua mạch tổ hợp nhờ các tín hiệu từ hệ nhớ các trạng thái.

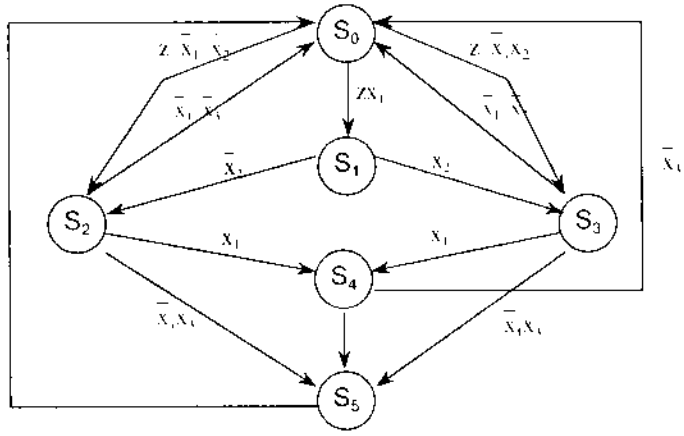
Chúng ta có thể biểu diễn ôtômat Moore trên ngôn ngữ VHDL tương tự như khi biểu diễn các mạch có nhớ trên mức thanh ghi. Quá trình biểu diễn ôtômat hữu hạn sẽ được chia thành hai phân hệ: phân hệ tổ hợp và phân hệ mạch tuần tự. Tín hiệu xóa trạng thái không đồng bộ khởi tạo giá trị cho các thanh ghi và đưa ôtômat về trạng thái ban đầu. Cùng với sự xuất hiện sườn của tín hiệu đồng bộ CLK, giá trị của trạng thái mới được gán cho trạng thái hiện thời. Phân hệ mạch tổ hợp mô tả mạch tổ hợp tính toán trạng thái mới và mạch tổ hợp tính toán giá trị đầu ra. Đối với ôtômat Moore, tín hiệu đầu ra chỉ phụ thuộc vào trạng thái hiện thời nên mạch tổ hợp xác định đầu ra không kết nối với các tín hiệu vào. Dưới đây chúng ta xét ví dụ xây dựng chương trình trên ngôn ngữ VHDL mô tả ôtômat Moore điều khiển mạch cộng hai số nguyên dấu phẩy tĩnh.

Ví dụ, xây dựng mạch điều khiển phép toán cộng hai số nguyên. Sơ đồ thuật toán được đưa ra trên hình 7.17. Các số dương được lưu trữ dưới dạng mã trực tiếp còn các số âm được lưu trữ theo dạng mã bù hai. Theo sơ đồ thuật toán, chúng ta xây dựng được sơ đồ chuyển trạng thái của ôtômat Moore tương ứng (hình 7.16). Trên hình 7.17, các trạng thái của ôtômat Moore nằm tương ứng với các khối thực hiện phép toán và không ghi trong dấu ngoặc đơn.

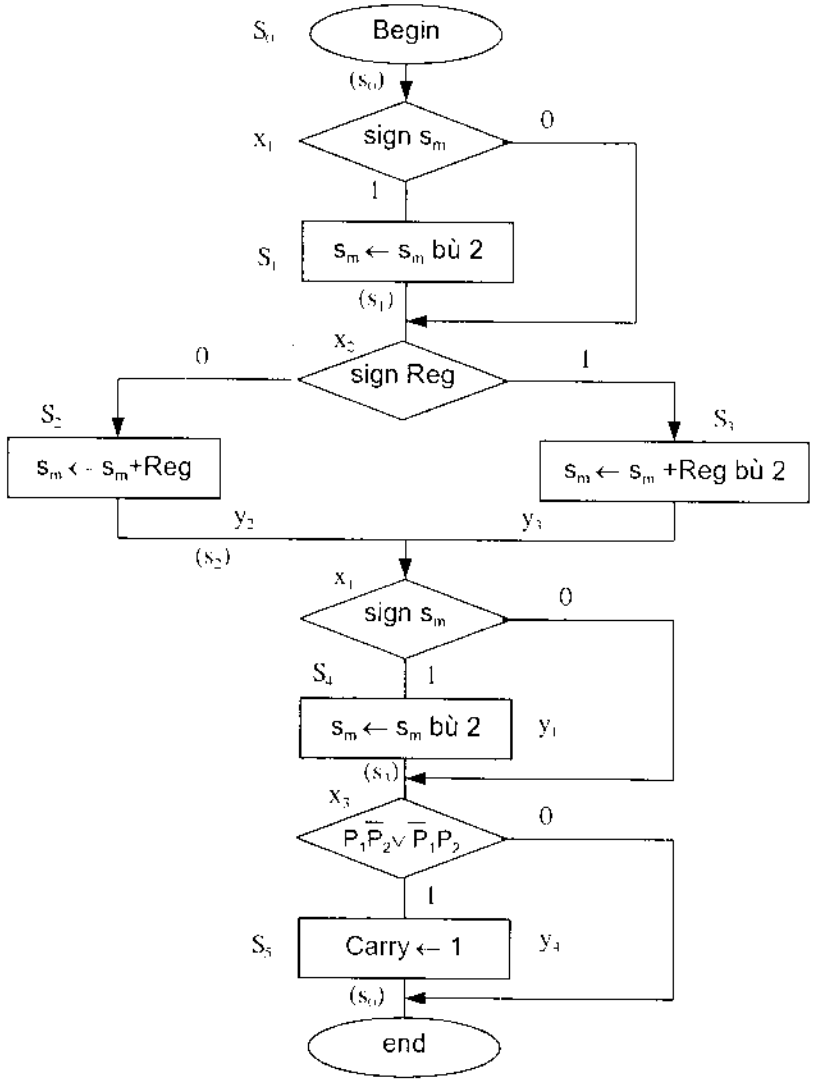
Ôtômat Moore này có các tín hiệu vào là: $X = \{ x_1 , x_2 , x_3 \}$;

các tín hiệu ra là: $Y = \{ y_1, y_2, y_3, y_4 \}$;

các trạng thái là: $S = \{ S_0 , S_1 , S_2 , S_3 , S_4 , S_5 \}$;




**Hình 7.16.** Sơ đồ chuyển trạng thái của ôtômat Moore tương ứng với thuật toán trên hình 7.17.




**Hình 7.17.** Sơ đồ thuật toán thực hiện phép cộng hai số có dấu phẩy tĩnh.

Đoạn chương trình dưới đây mô tả một phần của ôtômat Moore nói trên.

```
entity MooreSum is
        port (  CLK, RST: in BIT;
                X: in BIT-VECTOR ( 3 downto 1 );
                Y: out BIT-VECTOR ( 4 downto 1 ) );
end MooreSum;
```

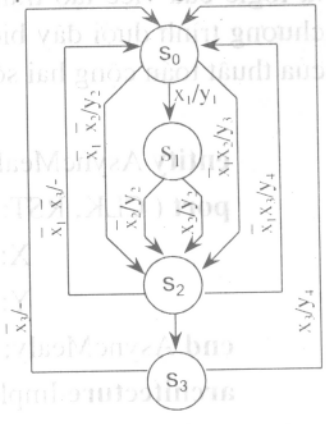
```
architecture Implement of MooreSum is
begin
        process ( CLK, RST, X )
                type StateType is ( S0, S1, S2, S3, S4, S5 )
                variable State, NextState: StateType;
        begin
                if ( RST = '1' ) then
                        for I in 1 to 4 loop
                                Y(I) <= '0';
                        end loop;
                        S <= S0;
                elsif ( CLK'event and CLK = '1' ) then
                        State := NextState;
                end if
                case State is
                        when S0 =>
                                for I in 1 to 4 loop
                                        Y( I ) <= '0';
                                end loop;
                                if ( X( 1 ) = '1' ) then
                                        NextState := S1;
                                elsif ( X( 2 ) = '1' ) then
                                        NextState := S3;
                                else NextState := S2;
                                end if;
                        when S1 =>
                                Y( 1 ) <= '1';
                                if ( X( 2 ) = '1' ) then
                                        NextState := S3;
                                else NextState := S2;
                                end if;
                        ....
```

## 2. Mô hình hóa ôtômat Mealy

Ôtômat Mealy có các hàm chuyển trạng thái và hàm ra được biểu diễn theo hệ thức sau:

$$s(t) = \delta(s(t-1), x(t))$$
$$y(t) = \lambda(s(t), x(t)) \qquad t=1, 2, ...$$

Đối với ôtômat Mealy, các tín hiệu đầu ra là phụ thuộc vào trạng thái và tín hiệu đầu vào ở thời điểm hiện thời. Như vậy, tín hiệu đầu ra có thể thay đổi nếu tín hiệu đầu vào bị thay đổi trong thời gian xuất hiện xung nhịp đồng hồ. Điều đó làm cho tín hiệu đầu ra của ôtômat Mealy có thể tức thời nhận giá trị không dự đoán trước do có sự trễ tín hiệu xét từ thời điểm tín hiệu vào thay đổi đến thời điểm mà giá trị đầu ra của phần tử flip-flop thay đổi. Để ngăn chặn sự thay đổi giá trị đầu ra trong thời gian tồn tại của xung đồng bộ, chúng ta phải đồng bộ hóa hoạt động của ôtômat Mealy không đồng bộ. Để đạt được điều này, tín hiệu đi vào hệ mạch nhớ trạng thái phải được đồng bộ bằng xung đồng hồ và khi đó tín hiệu đầu ra phải được xác định chỉ trong thời gian thiết lập sườn của xung đồng hồ.



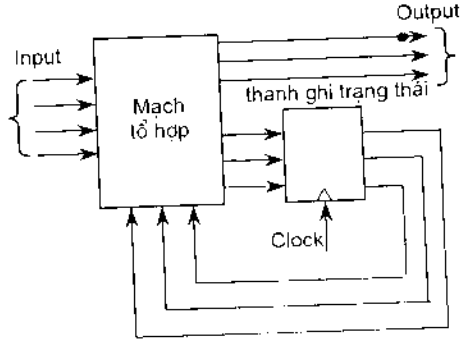

**Hình 7.18.** Sơ đồ chuyển trạng thái của ôtômat Mealy tương ứng với thuật toán trên hình 7.17.

Cũng giống như ôtômat Moore, ôtômat Mealy cũng được mô tả trên ngôn ngữ VHDL bằng hai phân hệ: phân hệ mạch tổ hợp xác định hàm ra với hàm chuyển trạng thái và phân hệ mạch nhớ đồng bộ. Ôtômat Mealy có thể được xây dựng theo hai dạng: dạng mạch không đồng bộ đầu ra và dạng mạch đồng bộ đầu ra. Chúng ta hãy xét ví dụ xây dựng mạch điều khiển thực hiện phép cộng hai số dấu phẩy tĩnh đã nêu ở mục trước. Ôtômat Mealy tương ứng với thuật toán trên hình 7.17 có giản đồ chuyển trạng thái mô tả trên hình 7.18. Ôtômat Mealy nhận được sẽ có các thông số sau:

Các tín hiệu vào là: $X = \{ x_1, x_2, x_3 \}$;
Các tín hiệu ra là: $Y = \{ y_1, y_2, y_3, y_4 \}$;
Các trạng thái là: $S = \{ S_0, S_1, S_2, S_3 \}$;

Như vậy chúng ta thấy rằng, với cùng một thuật toán, thiết kế theo mô hình Mealy sẽ tiết kiệm trạng thái hơn so với thiết kế theo mô hình Moore.

Nếu xây dựng ôtômat Mealy theo thiết kế không đồng bộ, sơ đồ của ôtômat sẽ có dạng như trên hình 7.19. Trong phần hệ mạch nhớ đồng bộ, việc kích hoạt trạng thái không đồng bộ (asynchoronous reset) đặt giá trị đầu cho các thanh ghi và đưa máy về trạng thái ban đầu. Mỗi lần xuất hiện sườn của tín hiệu CLK giá trị của trạng thái mới trở thành trạng thái hiện thời. Phần hệ tổ hợp của ôtômat mô tả lôgic của việc tạo trạng thái mới và tín hiệu đầu ra của hệ thống. Đoạn chương trình dưới đây biểu diễn ôtômat Mealy theo thiết kế không đồng bộ của thuật toán cộng hai số có dấu phẩy tĩnh đã nêu ở mục trước.

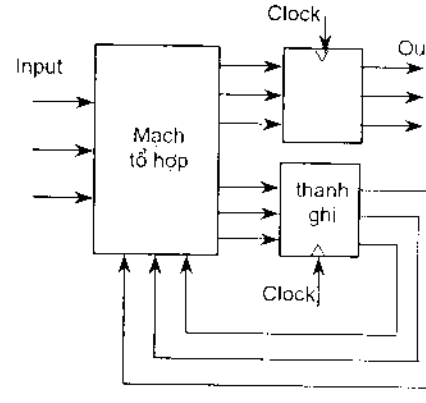



**Hình 7.19.** Sơ đồ nguyên lý của ôtômat Mealy hoạt động theo chế độ không đồng bộ đầu ra.

```
entity AsyncMealy is
port ( CLK, RST: in BIT;
                X: in BIT-VECTOR ( 3 downto 1 );
                Y: out BIT-VECTOR ( 4 downto 1 ) );
end AsyncMealy;
architecture Implement of AsyncMealy is
begin
        process ( CLK, RST, X )
                type StateType is ( S0, S1, S2, S3 );
                variable State, NextState: StateType ;
        begin
                if ( RST = '1' ) then
                        State := S0;
                elsif ( CLK'event and CLK = '1' ) then
                        State := NextState;
```

```
end if;
case State is
        when S0 =>
                if ( X( 1 ) = '1' ) then
                        Y( 1 ) = '1';
                        NextState := S1;
                elsif ( X( 2 ) = '1' ) then
                        Y( 3 ) = '1';
                        NextState := S2;
                else    Y( 2 ) = '1';
                        NextState := S2;
                end if;
        when S1 =>
                if ( X( 2 ) = '1' ) then
                        Y( 3 ) = '1';
                        NextState := S2;
                else    Y( 2 ) = '1';
                        NextState := S2;
                end if;
        ....
```

Chúng ta thấy rằng, trong ôtômat Mealy không đồng bộ, sự xuất hiện tín hiệu đầu ra không đồng bộ với tín hiệu đầu vào. Điều này có thể sẽ dẫn tới những kết quả thực hiện phép tính sai. Để khắc phục nhược điểm của ôtômat Mealy không đồng bộ, chúng ta sẽ thêm vào trong mạch một hệ nhớ đồng bộ. Hệ nhớ này nằm giữa mạch tổ hợp tính toán hàm ra với đầu ra của ôtômat. Thông thường các phần tử của mạch nhớ đồng bộ này là các phần tử flip-flop. Các flip-flop này




**Hình 7.20.** Sơ đồ nguyên lý của ôtômat Mealy hoạt động theo chế độ đồng bộ đầu ra.

được đồng bộ theo sườn giống như các thanh ghi trạng thái. Khi xuất hiện xung đồng hồ, trong thời gian thiết lập sườn của tín hiệu đồng hồ, hệ mạch nhớ đầu ra nhận giá trị do mạch tổ hợp xác định hàm ra tính toán được. Trong toàn bộ khoảng thời gian sau khi xung đồng hồ thiết lập sườn, hệ mạch nhớ không thay đổi trạng thái. Điều này làm cho trong khoảng thời gian tồn tại của xung đồng hồ, giá trị của các tín hiệu đầu ra không thay đổi cho dù tín hiệu đầu vào có thể bị thay đổi do những lý do khách qua. Chúng ta có thể mô tả các hàm chuyển trạng thái trong phần đồng bộ của chương trình VHDL. Đối với ôtômat Mealy theo thuật toán trên hình 7.17, thiết kế với mạch đồng bộ đầu ra sẽ được biểu diễn bằng ngôn ngữ VHDL như dưới đây.

```
entity SyncMealy is
port ( CLK, RST: in BIT;
        X: in BIT-VECTOR ( 3 downto 1 );
        Y: out BIT-VECTOR ( 4 downto 1 ) );
end SyncMeal;
architecture Implement of SyncMeal is
begin
        process ( CLK, RST, X )
                type StateType is ( S0, S1, S2, S3, S );
                variable State : StateType;
        begin
                if ( RST = '1' ) then
                        State := S0;
                elsif ( CLK'event and CLK = '1' ) then
                case State is
                        when S0 =>
                                if ( X( 1 ) = '1' ) then
                                        Y( 1 ) <= '1';
                                        State := S1;
```

```
elsif ( X( 2 ) = '1' ) then
        Y( 3 ) <= '1';
        State := S3;
else    Y( 2 ) <= '1';
        State := S2;
end if;
```

....

Như vậy, trong chương này chúng ta đã đưa ra những phương pháp mô hình hóa cấu trúc và chức năng của những mạch lôgic cơ bản - các mạch tổ hợp và các mạch tuần tự sử dụng những cấu trúc cơ bản trong ngôn ngữ VHDL.

**Bài tập cho chương 7**

1. Hãy mô tả hành vi của bộ mã hoá 16 – 4 bằng ngôn ngữ VHDL.

2. Mô tả cấu trúc và chức năng của bộ đếm thuận – nghịch, đồng bộ, mã NBCD bằng ngôn ngữ VHDL.

3. Mô tả cấu trúc và chức năng của bộ đếm không đồng bộ, modul 10 bằng ngôn ngữ VHDL.

4. Mô tả cấu trúc và chức năng của bộ đếm đặt lại trạng thái bằng ngôn ngữ VHDL.

5. Mô tả cấu trúc và chức năng của bộ đếm nhị phân không đồng bộ bằng ngôn ngữ VHDL.

6. Mô tả cấu trúc và chức năng của mạch cộng, trừ mã bù 2 bằng ngôn ngữ VHDL.

7. Mô tả cấu trúc và chức năng của mạch cộng, trừ mã bù 1 bằng ngôn ngữ VHDL.

8. Mô tả cấu trúc và chức năng của mạch cộng, trừ mã BCD bằng ngôn ngữ VHDL.

9. Mô tả cấu trúc và chức năng của mạch cộng 2 số nhị phân 8 bit bằng ngôn ngữ VHDL.

10. Mô tả cấu trúc và chức năng của mạch cộng nhanh 2 số nhị phân 8 bit bằng ngôn ngữ VHDL.

11. Mô tả cấu trúc và chức năng của các mạch tổ hợp, thực hiện các biểu thức toán học, logic bằng ngôn ngữ VHDL.

# CHƯƠNG VIII. CÁC PHƯƠNG PHÁP KIỂM TRA LỖI MẠCH LÔGIC

Trong chương này chúng tôi trình bày một số phương pháp phát hiện các lỗi của các mạch lôgic. Việc phát hiện lỗi được thực hiện dựa vào các bộ giá trị thử nghiệm phát hiện lỗi. Ở đây việc phát hiện các lỗi thiết kế có thể thực hiện trên các mô hình biểu diễn bằng các ngôn ngữ mô hình hóa phần cứng. Điều này đóng một vai trò quan trọng trong quá trình sản xuất và làm giảm được chi phí kiểm tra. Trong chương này chúng tôi tập trung chủ yếu vào các phương pháp phát hiện lỗi và tạo bộ giá trị thử nghiệm cho các mạch tổ hợp. Đối với những mạch tuần tự, do cơ chế hoạt động phức tạp của chúng nên việc phát hiện lỗi được thực hiện dựa trên những phương pháp phức tạp hơn đối với các mạch tổ hợp và vượt ra ngoài khuôn khổ cuốn sách này.

## §8.1. Các mô hình lỗi lôgic

Bài toán phát hiện lỗi trong các mạch lôgic là bài toán xác định sơ đồ lôgic được thiết kế thực hiện được các chức năng đã đề ra. Để giải quyết bài toán này chúng ta cần phải xây dựng mô hình của mạch lôgic như một đối tượng kiểm tra, sau đó xây dựng phương pháp phát hiện lỗi và cuối cùng là xây dựng mô hình lỗi ( có nghĩa là chỉ ra được các đặc trưng của lỗi ). Ta đã thấy rằng theo quan điểm phân loại mạch dựa vào hoạt động của chúng, các mạch lôgic được chia thành thành các mạch tổ hợp và các mạch tuần tự.

Như ta đã thấy trong các mục trước, trên quan điểm về hoạt động, các mạch tổ hợp được thể hiện ở chỗ trạng thái của các đầu ra ở mọi thời điểm thời gian được xác định hoàn toàn bằng các trạng thái của các đầu vào tại cùng một thời điểm. Nếu xét trên quan điểm cấu trúc, trong các mạch tổ hợp hoàn toàn không chứa các vòng tín hiệu phản hồi. Nếu xét trên khía cạnh phát hiện lỗi, các mạch tổ hợp là những đối tượng nghiên cứu khá đơn giản.

Nếu xét trên quan điểm hành vi, trong hoạt động của các mạch tuần tự xuất hiện các trạng thái bên trong; còn xét trên quan điểm cấu trúc, các mạch này còn chứa các vòng phản hồi. Điều đó làm cho việc phát hiện lỗi trong các mạch tuần tự là một bài toán vô cùng phức tạp.

Các bộ nhớ và các bộ vi xử lý về lý thuyết cũng thuộc nhóm các mạch tuần tự, nhưng vì số lượng các trạng thái trong của chúng rất lớn, do đó, nói

chung, các mạch này không thể xếp vào nhóm các mạch tuần tự. Với những đặc điểm khác biệt như vậy về cấu trúc và hành vi, việc phát hiện lỗi đối với các bộ nhớ và bộ vi xử lý cần phải tính đến các đặc trưng của chúng về mặt cấu trúc và hành vi.

Như vậy chúng ta thấy rằng không thể có phương pháp chung để phát hiện lỗi trong các mạch lôgic. Để có thể thực hiện được công việc đó ta cần phải lựa chọn các phương pháp phát hiện lỗi tính tới các đặc trưng của mạch.

Một bài toán quan trọng xuất hiện khi xây dựng các phương pháp phát hiện lỗi là lựa chọn điểm đặt cho các tín hiệu thử nghiệm và các điểm quan sát thích hợp. Trước đây, khi kiểm tra các mạch lôgic phức tạp, chúng ta quan sát các tín hiệu thông qua các điểm nút bên trong của mạch. Khi mức độ tích hợp của mạch tăng lên tới mức như trong các mạch LSI, việc đặt thiết bị kiểm tra các nút bên trong mạch trở nên phức tạp hơn nhiều. Điều đó làm cho việc đặt và quan sát các tín hiệu thử nghiệm tại những điểm bất kỳ của mạch trở nên vô cùng khó khăn và nhiều khi không thể thực hiện nổi. Như vậy, phương pháp phát hiện lỗi đối với những mạch LSI chỉ được sử dụng các chân vào và chân ra của mạch.

Khi độ tích hợp của mạch tăng từ LSI đến VLSI, tỷ lệ giữa số lượng chân bên ngoài của mạch và số lượng phần tử trên tinh thể bán dẫn giảm, điều đó làm cho bài toán phát hiện lỗi đối với những mạch VLSI trở nên phức tạp hơn nhiều so với việc phát hiện lỗi cho các mạch LSI. Việc tăng mức độ tích hợp của mạch cho phép xây dựng những mạch có chức năng phức tạp, ví dụ như các bộ vi xử lý, điều này làm tăng độ phức tạp của bài toán phát hiện lỗi. Việc tìm ra các phương pháp giải quyết bài toán phát hiện lỗi là chìa khoá cho việc thực hiện các mạch VLSI và đóng vai trò vô cùng quan trọng trong thời đại của các mạch VLSI.

Vì mạch cần được kiểm tra là mạch lôgic, nên ta giả thiết rằng khi trong mạch có lỗi, mạch vẫn thực hiện các chức năng như một mạch lôgic. Các lỗi thoả mãn điều kiện này đều được gọi là các lỗi lôgic.

Các lỗi lôgic biểu hiện ảnh hưởng của các lỗi vật lý lên hành vi của các hệ thống được mô hình hóa. Vì trong quá trình mô hình hóa các phần tử mạch chúng ta tách biệt các chức năng lôgic và hành vi thời gian, cho nên ta sẽ phân chia các lỗi thành các nhóm lỗi sau:

- Nhóm các **lỗi ảnh hưởng tới chức năng** lôgic của phần tử.
- Nhóm các **lỗi ảnh hưởng tới độ trễ** tín hiệu đi qua phần tử. Các lỗi loại này sẽ ảnh hưởng tới thời gian tính toán của phần tử và do đó cũng ảnh hưởng tới toàn mạch.

Khi ta mô tả các lỗi vật lý như các lỗi lôgic, ta sẽ được những lợi điểm như sau:

Thứ nhất là, bài toán phân tích lỗi trở thành bài toán lôgic hơn là bài toán vật lý. Điều này làm độ phức tạp của bài toán giảm vì nhiều lỗi vật lý khác nhau có thể được mô hình hóa bằng cùng một lỗi lôgic.

Thứ hai là một số lỗi lôgic trở nên không phụ thuộc vào công nghệ theo nghĩa: cùng một mô hình lỗi có thể sử dụng trong nhiều công nghệ khác nhau. Do đó các phương pháp kiểm tra và chuẩn đoán được xây dựng cho một mô hình lỗi sẽ không thay đổi ngay cả khi có sự thay đổi về công nghệ.

Thứ ba là các bộ giá trị thử nghiệm để phát hiện các lỗi lôgic có thể được sử dụng đối với các lỗi vật lý có hành vi trong mạch chưa hoàn toàn được hiểu rõ hoặc quá phức tạp để có thể phân tích.

Một mô hình lỗi có thể là mô hình ẩn hoặc mô hình tường minh.

- **Mô hình lỗi tường minh** xác định một không gian lỗi, trong đó từng lỗi được xác định độc lập, và do đó lỗi sẽ phân tích có thể được xác định một cách tường minh. Trên thực tế mô hình lỗi tường minh có thể được mở rộng tuy nhiên phải đảm bảo được kích thước của không gian lỗi không quá lớn.
- **Mô hình lỗi ẩn** xây dựng không gian lỗi bằng cách xác định có lựa chọn các lỗi tuỳ theo mức độ quan tâm chủ yếu thông qua các tính chất của chúng.

Nếu cho trước lỗi lôgic và mô hình mạch, về nguyên lý, chúng ta sẽ xác định được chức năng lôgic của mạch với sự tồn tại của lỗi này trong mạch. Như vậy, quá trình mô hình hoá lỗi liên quan mật thiết tới loại mô hình mạch đang sử dụng. Tuỳ theo mô hình mạch các lỗi lôgic có thể được chia sơ bộ thành các loại sau:

- Các lỗi được xác định gắn liền với mô hình cấu trúc được gọi là các lỗi cấu trúc. Ảnh hưởng của các lỗi cấu trúc là làm thay đổi sự liên kết giữa các thành phần mạch.
- Các lỗi được xác định gắn liền với mô hình chức năng của mạch được gọi là các lỗi chức năng. Ví dụ, ảnh hưởng của các lỗi chức

năng có thể làm thay đổi bảng chân lý của thành phần mạch hoặc dẫn tới việc cấm các phép toán trên thanh ghi.

Mặc dù các lỗi ngẫu nhiên hoặc lỗi đột biến có mặt thường xuyên trong mạch, việc mô hình hóa những lỗi đó yêu cầu các dữ liệu thống kê về sự xuất hiện theo xác suất của chúng. Những dữ liệu này cần thiết để xác định số lần kiểm tra một cách độc lập cần lặp lại để cực đại hóa xác suất phát hiện những lỗi chỉ xuất hiện vài lần trong mạch với những giá trị thử nghiệm. Thông thường chúng ta không có đầy đủ những thông tin về mặt thống kê, do đó đối với những đột biến hoặc xuất hiện không thường xuyên, tốt nhất là sử dụng các phương pháp kiểm nghiệm trực tuyến. Trong giáo trình này chúng ta chỉ nghiên cứu tới **các lỗi thường trực**.

Nếu không được nhắc tới một cách tường minh, chúng ta giả thiết rằng tại một thời điểm thời gian, trong mạch chỉ có nhiều nhất là một lỗi. Việc làm đơn giản hoá bài toán phát hiện lỗi bằng **giả thiết về lỗi đơn lẻ** được biện minh bởi *chiến lược kiểm tra thường xuyên*. Chiến lược kiểm tra thường xuyên có thể được phát biểu như sau: chúng ta cần kiểm tra hệ thống một cách thường xuyên sao cho xác suất xuất hiện nhiều hơn một lỗi giữa hai lần kiểm tra liên tiếp là đủ nhỏ. Do đó nếu khoảng thời gian hoạt động của mạch càng lớn, ta sẽ có nhiều khả năng gặp trường hợp xuất hiện nhiều lỗi trong mạch một lúc. Trên thực tế vẫn xuất hiện những trường hợp mà chiến lược kiểm tra thường xuyên không đủ để ngăn chặn trường hợp trong mạch có thể xuất hiện nhiều lỗi một lúc. Những trường hợp có thể xuất hiện nhiều hơn một lỗi trong mạch có thể là:

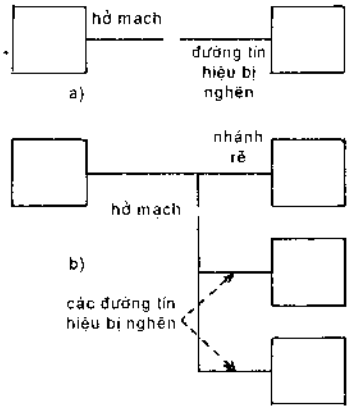
- Các lỗi vật lý có thể xuất hiện trong mạch giữa hai lần kiểm tra là. Trong những lỗi vật lý đó, một số lỗi có thể tương ứng với nhiều lỗi lôgic. Điều này có khả năng xảy ra lớn đối với những mạch có độ tích hợp cao trong đó nhiều lỗi vật lý có thể ảnh hưởng tới bề mặt tinh thể trên đó có một số các thành phần mạch.
- Trong những mạch mới được sản xuất, thì trong những lần thử đầu những lỗi kép có thể xuất hiện.
- Trong trường hợp những phép thử không phát hiện được hết những lỗi đơn lẻ, tại bất kỳ lúc nào, mạch có thể chứa những lỗi chưa được phát hiện (lỗi ẩn). Những lỗi ẩn này, khi xuất hiện những lỗi đơn lẻ thứ hai giữa hai lần kiểm tra, sẽ tạo ra nhiều lỗi kép trong mạch.

Tuy vậy, mặc dù trong mạch có thể xuất hiện nhiều lỗi đồng thời, nhưng những bộ giá trị thử nghiệm dùng để phát hiện các lỗi đơn lẻ có thể dùng để

tìm những lỗi kép. Sở dĩ có thể thực hiện được như vậy là do trong phần lớn các trường hợp, *những lỗi kép có thể được phát hiện khi kiểm nghiệm bằng những bộ giá trị thử nghiệm thiết kế cho việc phát hiện những lỗi đơn lẻ hợp thành lỗi kép nói trên.*

Nói chung, *các mô hình lỗi cấu trúc giả thiết rằng: các thành phần không có lỗi và chỉ có những đường kết nối chúng là có thể có lỗi.* Các lỗi đặc trưng do các đường kết nối tạo nên thường là: ngắn mạch hoặc hở mạch.

- ***Các lỗi ngắn mạch*** ( chập mạch ) là những lỗi xuất hiện khi những đường truyền không được phép liên kết bị chập. Ví dụ, trong nhiều công nghệ, sự ngắn mạch giữa dây đất hoặc nguồn với các đường tín hiệu có thể làm cho đường tín hiệu bị chập có mức điện thế cố định. Khi đó lỗi lôgic tương ứng với sự chập mạch sẽ bao gồm những tín hiệu nhận những giá trị cố định $v \in \{0,1\}$. Những lỗi này được gọi là những **lỗi hằng số** và được ký hiệu là *s-a-v*. Sự chập mạch giữa hai đường tín hiệu thường sinh ra những hàm lôgic mới và được gọi là các **lỗi bắc cầu**. Tương ứng với những hàm lôgic được sinh ra do chập mạch, chúng ta phân biệt lỗi bắc cầu AND và lỗi bắc cầu OR.
- ***Các lỗi mạch hở*** là kết quả của sự đứt các kết nối. Trong nhiều công nghệ, sự hở mạch trên những đường tín hiệu một chiều với một nhánh phân kỳ sẽ làm cho đường tín hiệu vào đó trở thành bị ngắt và nhận một giá trị lôgic cố định ( như trên hình 8.1a ) và do đó trên đường tín hiệu này xuất hiện một lỗi hằng số *s-a-v*. Những lỗi này cũng có thể là kết quả của những lỗi vật lý bên trong phần tử điều khiển đường tín hiệu. Nếu chúng ta không kiểm tra giá trị trên hai đầu của đường tín hiệu thì không thể phân biệt được hai trường hợp nói trên.



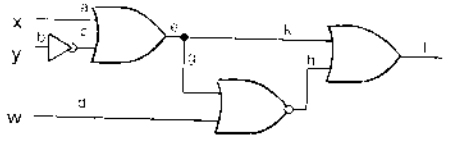

**Hình 8.1.** Lỗi hở mạch:
a) lỗi hở mạch đơn; b) lỗi hở mạch kép.

Như vậy chúng ta thấy rằng, một lỗi lôgic đơn lẻ, ví dụ như lỗi hằng số nhận giá trị $a \in \{0, 1\}$ trên đường tín hiệu $i$ có thể biểu diễn các lỗi vật lý hoàn toàn khác nhau: lỗi do đường tín hiệu $i$ bị ngắt; lỗi do đường tín hiệu $i$ bị chập với nguồn hoặc đất; bất kỳ một lỗi vật lý bên trong phần tử có đường tín hiệu $i$ là đầu ra và lỗi này làm cho đường tín hiệu $i$ luôn giữ giá trị bằng $a$.

Sự hở mạch trên một đường tín hiệu có rẽ nhánh có thể sinh ra nhiều lỗi hằng số trên các nhánh thành phần (ví dụ như trên hình 8.1b). Nếu chỉ giới hạn nghiên cứu mô hình lỗi hằng số đơn lẻ, chúng ta phải xem xét mọi lỗi hằng số đơn lẻ trên các nhánh rẽ của đường tín hiệu tách biệt với lỗi trên nhánh chính.

Khi ta thực hiện mô hình hoá mạch theo từng bước phân cấp, mỗi thành phần mạch sẽ được biểu diễn theo các mô hình cấu trúc bên trong của chúng. Tuy nhiên, nếu các thành phần mạch được kiểm tra độc lập trước khi chúng được lắp ráp, khi đó chúng ta chỉ cần thiết kiểm tra các lỗi ảnh hưởng tới các đường kết nối giữa các phần tử đó. Như vậy chúng ta sẽ không xem xét các lỗi xuất hiện bên trong các thành phần mạch mà chỉ giới hạn việc khảo sát các lỗi gắn với các chân vào/ra của thành phần. Giả thiết về giới hạn các lỗi tại chân vào/ra gọi là *mô hình lỗi tại chân vào/ra*. Sau đây chúng ta xem xét một số ví dụ về ảnh hưởng của các dạng lỗi nói trên đến chức năng lôgic của các mạch.

Trong các mạch lôgic, lỗi hằng số là lỗi hay gặp. Khi trên một đường tín

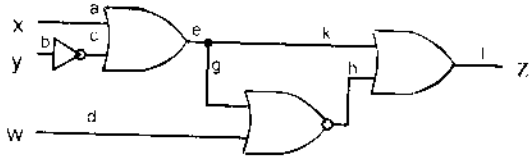



$$z = \overline{(x + \bar{y})} + \bar{w} + (x + y) = x + \bar{y} + \bar{w}$$

**Hình 8.2** Ví dụ sơ đồ mạch lôgic.

hiệu *a* xuất hiện lỗi hằng số và làm cho giá trị tín hiệu cố định bằng *k* thì lỗi này sẽ được gọi là lỗi hằng số giá trị *k* và được biểu diễn trên sơ đồ mạch là *a/k*. Ta hãy xét ví dụ mạch trên hình 8.2.

Nếu trong mạch có lỗi hằng số *d/1* thì đầu ra *z* sẽ bị thay đổi và hàm ra của mạch sẽ bằng:




**Hình 8.2** Ví dụ mạch lôgic.

$$z = f_{d\,1}(w,x,y) = x + \bar{y}$$

Trong trường hợp có lỗi hằng số *g/0* thì hàm ra sẽ bằng:

$$z = f_{g0}(w,x,y) = x + \bar{y} + \bar{w}$$

Nếu các lỗi hằng số xuất hiện trên đường tín hiệu có rẽ nhánh, ta cần chú ý xét ảnh hưởng lẫn nhau của các lỗi trên các phân nhánh của đường tín hiệu. Như trong ví dụ trên, trên đường tín hiệu *g* xuất hiện lỗi *g/0*, vấn đề đặt ra là lỗi

này có ảnh hưởng tới đường tín hiệu *k* không. Trong một số trường hợp, nếu chỉ xét tới các liên kết giữa các đường tín hiệu thì một cách tự nhiên, lỗi *g/0* sẽ gây ra lỗi *k/0*; nhưng nếu xét trên quan điểm vật lý thì điều đó không chắc chắn. Nói chung, để đảm bảo tính tổng quát, chúng ta cần phải giả thiết rằng lỗi *g/0* và *k/0* là những lỗi khác nhau. Như vậy, trong trường hợp trong mạch xuất hiện lỗi *k/0*, hàm ra sẽ cho giá trị *z* bằng:

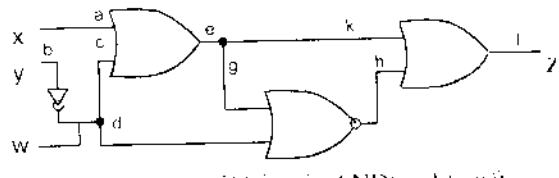
$$z = f_{k/0}(w,x,y) = \bar{w}\,\bar{x}\,y \neq f_{g/0}$$

Trong trường hợp này, hàm ra khác với trường hợp lỗi *g/0*.

Trong trường hợp trong mạch xuất hiện lỗi *e/0* , một cách tự nhiên, chúng ta có thể coi lỗi này là tập hợp của hai lỗi *g/0* và *k/0*. Như vậy chúng ta thấy có sự phân nhánh và lan truyền lỗi theo các đường tín hiệu liên kết.

Tóm lại, khi có sự phân nhánh đường tín hiệu, chúng ta cần chú ý tới những khả năng phát sinh lỗi trên các nhánh và khả năng xét các đường tín hiệu phân nhánh: các đường tín hiệu nằm sau điểm rẽ nhánh có thể được coi là những đường tín hiệu khác nhau, hoặc tất cả những đường tín hiệu liên kết với nhau có thể được coi là cùng một đường.

Bây giờ chúng ta xét các lỗi chập mạch. Như trên đã đề cập tới, trong nhiều trường hợp các lỗi chập mạch tạo nên những lỗi bắc cầu và được đặc trưng bởi việc thực hiện các hàm lôgic AND hoặc OR. Khi trong mạch có sự chập hai đường tín hiệu *a* và *b* và sinh ra các lỗi bắc cầu, tương ứng với các hàm lôgic mà chúng thể hiện là AND hoặc OR, các lỗi này sẽ được ký hiệu tương ứng là AND(a-b), OR(a-b). Ta hãy xét ví dụ mạch trên hình 8.2, nếu đường tín hiệu *c* và *d* bị chập và sinh ra lỗi chập mạch bắc cầu AND( c-d ), hàm ra sẽ có dạng như sau:




**Hình 8.3** Lỗi bắc cầu AND( c-d ) xuất hiện trong mạch trên hình 8.2.

$$f_{AND(c-d)}(w,x,y) = x + \bar{x}y + w\,\bar{y} + \bar{x}\,\bar{w}$$

Một trong những bài toán cơ bản gắn liền với mô hình lỗi lôgic là trả lời câu hỏi: mô hình lỗi lôgic có thể phản ánh các lỗi vật lý xuất hiện trong các mạch với độ trung thực nào. Để có thể nhận được sự tương ứng một cách khách quan với các lỗi vật lý, chúng cần phải nghiên cứu một số lượng lớn các mô hình lỗi lôgic. Nhưng khi đó, một khó khăn gặp phải là điều này làm cho các thuật toán tạo các bộ giá trị thử nghiệm dùng để phát hiện lỗi trở

nên phức tạp hơn nhiều. Đối với một số mô hình lỗi lôgic như mô hình lỗi hằng số, việc tạo các bộ giá trị thử nghiệm để phát hiện lỗi tương đối đơn giản. Thêm vào đó một điểm yếu của mô hình lỗi hằng số là mô hình lỗi này không phản ánh một cách đầy đủ các đặc tính của các lỗi vật lý.

## §8.2. Bài toán phát hiện lỗi

Trong mục này chúng ta đưa ra một số khái niệm về bài toán mô hình hoá lỗi và phát hiện lỗi trong các mạch lôgic. Vì chúng ta chia các mạch lôgic làm hai nhóm: các mạch tổ hợp và các mạch tuần tự dựa theo hành vi hoạt động của mạch. Các loại mạch này có điểm khác biệt ở sự tồn tại của bộ nhớ và vòng phản hồi. Do đó các phương pháp tạo các bộ giá trị thử nghiệm đối với hai loại mạch nói trên cũng có những điểm khác biệt. Chúng ta sẽ xét riêng từng loại mạch; một số phương pháp sử dụng cho các mạch tổ hợp cũng có thể phát triển sang trường hợp các mạch tuần tự với những sửa đổi thích hợp.

### 1. Phát hiện lỗi trong các mạch tổ hợp

#### a. Bài toán phát hiện lỗi

Cho $Z(x)$ là hàm lôgic của mạch tổ hợp $N$ với $x$ là vectơ giá trị đầu vào bất kỳ và $Z(x)$ biểu diễn ánh xạ thực hiện bởi mạch $N$. Với một vectơ giá trị đầu vào cụ thể $t = (x_1,x_2,...,x_n)$, ta sẽ có $Z(t)$ là đáp ứng của mạch $N$ đối với vectơ $t$. Đối với những mạch có nhiều đầu ra, $Z(t)$ cũng là vectơ.

Nếu trong mạch xuất hiện lỗi $f$, mạch $N$ sẽ chuyển thành mạch $N_f$. Ta giả thiết rằng mạch $N_f$ cũng là mạch tổ hợp với hàm chức năng $Z_f(x)$. Mạch được kiểm nghiệm bằng cách đặt dãy $T$ của các vectơ giá trị thử nghiệm $t_1,t_2,...,t_m$ lên các đầu vào của mạch và so sánh những giá trị thu được trên đầu ra theo lý thuyết của mạch $N$ tương ứng với những vectơ đầu vào đó: $Z(t_1)$, $Z(t_2)$,..., $Z(t_m)$ với những giá trị thu được trên thực tế.

Chúng ta đưa ra định nghĩa như sau: vectơ giá trị kiểm nghiệm $t$ được gọi là phát hiện lỗi $f$ nếu: $Z_f(t) \neq Z(t)$. Khi áp dụng định nghĩa trên, chúng ta phải chú ý những điểm dưới đây:

- Các vectơ giá trị kiểm nghiệm trong dãy $T$ có thể được sử dụng không phụ thuộc vào trình tự áp dụng, đối với mạch tổ hợp $N$, dãy $T$ được gọi là tập hợp các vectơ giá trị kiểm nghiệm.
- Định nghĩa này không áp dụng được nếu mạch chứa lỗi $N_f$ trở thành mạch tuần tự.
- Trong định nghĩa này chúng ta giả thiết rằng việc kiểm tra lỗi bằng cách đặt các giá trị thử nghiệm và thu nhận các kết quả thông qua các chân của phần tử và so sánh hoàn toàn của các kết quả nhận được.

Chúng ta xác định hàm:

$$F_f(t) \equiv F_f(x_1,x_2,...,x_n) = Z(x_1,x_2,..,x_n) \oplus Z_f(x_1,x_2,..,x_n)$$

Như vậy, nếu $Z(t) \neq Z_f(t)$ ta sẽ có $F_f(t)$ nhận giá trị '1'; còn nếu $Z(t) = Z_f(t)$ bằng nhau, tức là $F_f(t)$ nhận giá trị '0', ta nói rằng vectơ giá trị thử nghiệm $t$ không phát hiện được lỗi $f$. Hàm $F_f(t)$ còn được gọi là hàm khoảng cách lỗi. Vectơ giá trị đầu vào $t = (x_1,x_2,..,x_n)$ làm cho $F_f(t) = 1$ để phát hiện được lỗi $f$ và gọi là vectơ giá trị kiểm nghiệm phát hiện lỗi $f$.

Nếu trong mạch có p lỗi và nếu $F_1(t), F_2(t), \ldots, F_p(t)$ là các hàm khoảng cách tương ứng với từng lỗi, khi đó các vectơ giá trị kiểm nghiệm để phát hiện những lỗi đó được tạo thành tập hợp:

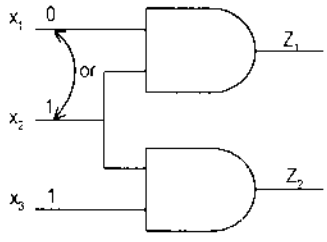
$$T = \bigcup_{i=1}^{p} \{t' \mid F_i(t') = 1\}$$

Tập hợp này gọi là tập các giá trị kiểm nghiệm để phát hiện lỗi.

Ta hãy xét một ví dụ, trên mạch hình 8.4 xuất hiện lỗi $f$ là lỗi bắc cầu OR giữa hai đường tín hiệu $x_1$ và $x_2$. Lỗi bắc cầu này đưa các hàm ra tại $Z_1$ và $Z_2$ trở thành:

$$Z_{1f} = x_1 + x_2 \text{ ( thay cho } Z_1 = x_1x_2 \text{ )}$$

$$Z_{2f} = ( x_1 + x_2 ) x_3 \text{ ( thay cho } Z_2 = x_2x_3 \text{ )}$$




**Hình 8.4** Ví dụ trường hợp lỗi bắc cầu logic OR.

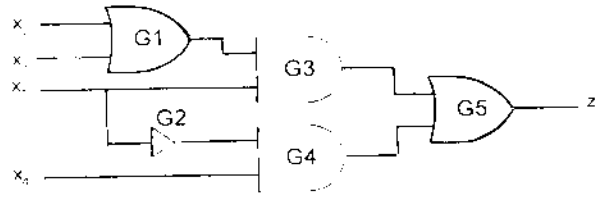
Vectơ giá trị đầu vào $t$ = "011" cho phép phát hiện lỗi này vì vectơ giá trị đầu ra khi mạch không lỗi bằng

$$Z(\,011\,) = 01\,;$$

trong khi đó khi xuất hiện lỗi $f$:

$$Z_f(\,011\,) = 11.$$

Như vậy, ta có $Z(\,t\,) \neq Z_f(\,t\,)$ và $F_f(\,t\,) = 1$.




**Hình 8.5** Ví dụ về lỗi hằng số $x_1/0$.

Ta xét thêm ví dụ về các lỗi hằng số đối với mạch trên hình 8.5. Mạch này thực hiện hàm lôgic:

$Z = ( x_2 + x_3 )x_1 + \bar{x}_1 x_4$

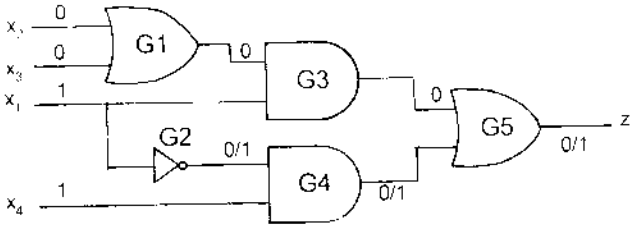
Giả thiết rằng trong mạch xuất hiện lỗi hằng số *1* giá trị '0' *s-a-0* trên đường vào $x_1$ ( lỗi $x_1/0$ ). Với sự xuất hiện của lỗi $x_1/0$, ta luôn có $\bar{x}_1 x_4 = 1$ và hàm ra tại đường *z* trở thành: $Z_f = ( x_2 + x_3 )x_1$. Điều kiện phát hiện lỗi $F_f( t ) = 1$ trở thành

$$F_f( t ) = Z( t ) \oplus Z_f( t ) = \bar{x}_1 x_4 = 1$$

Như vậy mọi vectơ giá trị đầu vào có $x_1$ và $x_4$ thoả mãn điều kiện trên - tức là $x_1 = 0$ và $x_4 = 1$, là vectơ giá trị kiểm nghiệm phát hiện lỗi $x_1/0$. Suy ra vectơ giá trị kiểmnghiệm phát hiện lỗi $x_1/0$ có thể là một trong những vectơ sau:( 0001, 0011, 0101, 0111 ) .

### b. Độ mẫn cảm với lỗi

Ta xét ví dụ mạch trên hình 8.5. Ta đặt vào mạch vectơ giá trị vào "1001" và quan sát hành vi của mạch trong trường hợp không có lỗi và có lỗi cố định *s-a-1* trên đầu ra của phần tử $G_5$. Trên hình vẽ đưa ra giá trị tín hiệu trên các đường tín hiệu trong hai trường hợp không lỗi và có lỗi dưới dạng $v/v_f$, trong đó *v* là những giá trị tương ứng với trường hợp không lỗi và $v_f$ - có lỗi. Lỗi được phát hiện trên đầu ra giá trị $v=0 \neq v_f = 1$ .




**Hình 8.6** Khảo sát độ mẫn cảm với lỗi của mạch.

Ví dụ trên minh hoạ hai khái niệm cơ sở trong bài toán phát hiện lỗi của các mạch lôgic:

- Vectơ giá trị kiểm nghiệm *t* dùng để phát hiện lỗi *f* đã kích hoạt lỗi *f* trong quá trình mô phỏng, ( còn có thể nói là khởi tạo lỗi hoặc tạo hiệu ứng lỗi ) bằng cách tạo ra những giá trị khác nhau $v$, $v_f$ trên các đường tín hiệu tính từ vị trí lỗi.
- Vectơ giá trị kiểm nghiệm *t* truyền lỗi *f* tới đầu ra $w$, và làm cho tất cả những đường tín hiệu trên ít nhất một đường dẫn giữa vị trí bị lỗi và đầu ra $w$ phải có những giá trị $v$ và $v_f$ khác nhau. Trong ví dụ trên hình 8.6, lỗi được truyền theo đường qua các phần tử ( $\mathbf{G_2}$, $\mathbf{G_4}$, $\mathbf{G_5}$ ).

Đường truyền bị vectơ giá trị kiểm nghiệm *t* thay đổi giá trị khi xuất hiện lỗi *f* trong mạch gọi là mẫn cảm với lỗi *f* qua vectơ kiểm nghiệm *t*. Đường dẫn được tạo bởi những đường mẫn cảm sẽ được gọi là đường dẫn mẫn cảm.

Phần tử lôgic có đầu ra mẫn cảm với lỗi *f* sẽ có ít nhất một đầu vào cũng được làm cho mẫn cảm với lỗi *f*. Các giá trị đảo $v_f$ và giá trị điều khiển $v_c$ của phần tử lôgic AND, OR, NAND, NOR có thể được thể hiện như sau:

- Giá trị đầu vào được gọi là giá trị điều khiển $v_c$ nếu giá trị đó xác định đầu ra của phần tử không phụ thuộc vào các đầu vào khác.
- **Giá trị đảo $v_f$: Các phần tử lôgic And, Or, Nand, Nor nhận các giá trị đảo $v_f$ và giá trị điều khiển $v_c$ như trong bảng sau:**

| | $v_c$ | $v_f$ | | $v_c$ | $v_f$ | | $v_c$ | $v_f$ | | $v_c$ | $v_f$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| And | 0 | 0 | Or | 1 | 0 | Nand | 0 | 1 | Nor | 1 | 1 |

- **Đối với các phần tử lôgic cơ bản là And, Or, Nand, Nor, giá trị đầu ra được tính theo các giá trị đảo và giá trị điều khiển và bằng $v_c \oplus v_f$.**

Như vậy, mọi phần tử lôgic với 3 đầu vào sẽ có các khối cơ sở như sau:

| | |
|---|---|
| $v_c$ X X | $v_c \oplus v_f$ |
| X $v_c$ X | $v_c \oplus v_f$ |
| X X $v_c$ | $v_c \oplus v_f$ |
| $\overline{v_c}$ $\overline{v_c}$ $\overline{v_c}$ | $\overline{v_c} \oplus v_f$ |

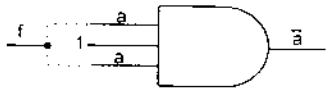
Với cách biểu diễn thông qua các giá trị $v_f$ và $v_f$, mệnh đề về tính mẫn cảm với lỗi của các phần tử lôgic có thể được phát biểu như sau:

**Mệnh đề về tính mẫn cảm lỗi**

Cho **G** là phần tử lôgic với giá trị nghịch đảo $v_i$ và giá trị điều khiển $v_c$ có đầu ra nhạy cảm với lỗi $f$ (tương ứng với vectơ giá trị kiểm nghiệm $t$ phát hiện lỗi), ta có:

- Tất cả các đầu vào của **G** mẫn cảm với lỗi $f$ sẽ nhận cùng một giá trị (ví dụ là $a$).
- Tất cả những đầu vào của **G** không mẫn cảm với lỗi $f$ (nếu ít nhất một đầu vào nào đó) sẽ nhận giá trị $\overline{v}_c$.
- Đầu ra của **G** sẽ nhận giá trị $a \oplus v_i$.

**Chứng minh mệnh đề về tính mẫn cảm lỗi**.




**Hình 8.7** Phần tử NAND thoả mãn mệnh đề về tính mẫn cảm lỗi, $v_c = 0$, $v_i = 1$.

Chúng ta chứng minh mệnh đề này bằng phương pháp phản chứng như sau:

- Giả thiết rằng có hai đầu vào nhạy cảm với lỗi trong phần tử **G** là $k$, $l$. Các đầu vào này nhận các giá trị khác nhau và giả sử đầu vào $k$ nhận giá trị điều khiển của phần tử $v_c$. Khi trên phần tử xuất hiện lỗi $f$, hai đường tín hiệu $k$ và $l$ thay đổi giá trị và đường $l$ nhận giá trị điều khiển. Nhưng điều này có nghĩa là phần tử **G** vẫn nhận giá trị cũ, độc lập với sự xuất hiện của lỗi $f$. Điều này mâu thuẫn với giả thiết đầu ra của **G** phải mẫn cảm với lỗi – tức là phải thay đổi giá trị khi trong mạch xuất hiện lỗi $f$. Như vậy, tất cả những đầu vào mẫn cảm của **G** cần phải nhận cùng một giá trị ( ví dụ là $a$ ).
- Đầu ra của **G** không thể thay đổi khi xuất hiện lỗi $f$ nếu một trong các đầu vào không nhạy cảm với lỗi $f$ nhận giá trị $v_c$; do đó tất cả các đầu vào của **G** không nhạy cảm với lỗi $f$ ( nếu có ) cần phải nhận giá trị $\overline{v}_c$.
- Nếu $a = v_c$, đầu ra của phần tử lôgic nhận giá trị bằng $v_c \oplus v_i$. Nếu $a = \overline{v}_c$ , tất cả các đầu vào của **G** sẽ nhận giá trị $\overline{v}_c$ và do đó đầu ra của phần tử sẽ nhận giá trị bằng $\overline{v}_c \oplus v_i$. Như vậy, trong cả hai trường hợp đầu ra của phần tử lôgic **G** đều nhận giá trị bằng $a \oplus v_i$.

Giá trị $\bar{v}_i$ gọi là giá trị kích hoạt vì nó cho phép các lỗi lan truyền qua phần tử tới đầu ra. Trên hình 8.7, đưa ra ví dụ về phần tử NAND thoả mãn mệnh đề trên. Phần tử NAND này sẽ có tất cả các đầu vào nhận giá trị '1' hoặc tất cả những đầu vào mẫn cảm nhận giá trị '0' và những đầu còn lại được đặt giá trị '1'.

Nếu ta áp dụng phần 3 của mệnh đề trên lặp lại nhiều lần, chúng ta sẽ nhận được hệ quả sau về các đường dẫn mẫn cảm với lỗi:

**Hệ quả**

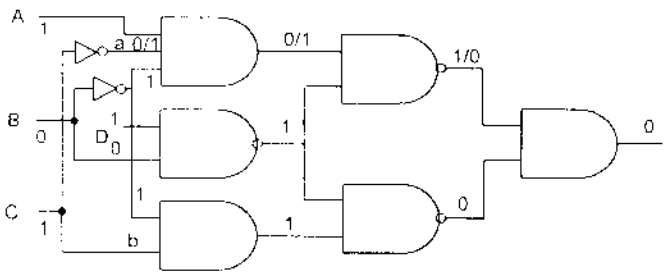
Giả thiết đường tín hiệu *j* mẫn cảm với lỗi hằng số giá trị *v s-a-v f* (tương ứng với vectơ giá trị kiểm nghiệm *t* phát hiện lỗi ) và *p* là giá trị nghịch đảo của đường mẫn cảm lỗi giữa *f* và *j*.

- Khi đó giá trị của *j* trong vectơ *t* là $\bar{v} \oplus p$.
- Nếu có một số đường tín hiệu nhạy cảm giữa *f* và *j* thì tất cả các đường đó đều cùng nhận giá trị nghịch đảo *p*.

### c. Khả năng phát hiện lỗi

Do các điểm quan sát chỉ giới hạn tại các đầu vào và đầu ra ( có nghĩa là việc kiểm tra chỉ được thực hiện theo giá trị của hàm ra ) nên có thể xảy ra trường hợp có những lỗi không phát hiện được hoặc có những lỗi không phân biệt được. Một lỗi *f* được gọi là có thể phát hiện được nếu tồn tại vectơ giá trị kiểm nghiệm *t* phát hiện được *f*. Trong trường hợp ngược lại, lỗi *f* được coi là không thể phát hiện được.

Những lỗi *f* không thể phát hiện được là những lỗi làm cho giá trị hàm ra không thay đổi so sánh với trường hợp trong mạch không có lỗi. Như vậy ta có $Z_f(t) = Z(t)$, hay có thể nói cách khác $F_f(t) = 0$. Khi đó không thể có vectơ giá trị kiểm nghiệm nào có thể kích hoạt quá trình truyền lỗi *f* và đồng thời tạo ra đường dẫn mẫn cảm với lỗi *f* đến đầu ra chính của mạch. Những lỗi không thể phát hiện được gọi là những **lỗi dư thừa** và những mạch tổ hợp chứa những lỗi hằng số dư thừa được được gọi là **mạch dư thừa**.

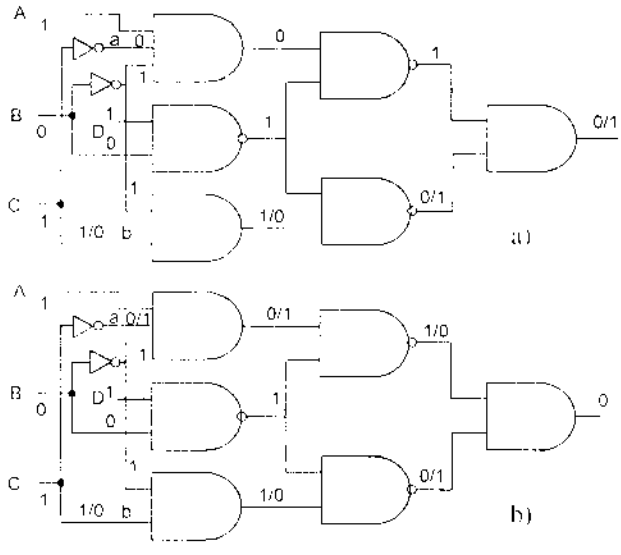



**Hình 8.8.** Minh họa trường hợp không phát hiện được lỗi *a/1* trong mạch.

Ví dụ, với mạch trên hình 8.8, lỗi hằng số trên đường tín hiệu *a s-a-1 a/1* là lỗi không thể phát hiện được.

Những lỗi không phát hiện được nói trên có thể bỏ qua vì chúng không làm thay đổi chức năng lôgic của mạch nhưng sự có mặt của các lỗi này có thể phá vỡ giả thiết về điều kiện trong mạch chỉ có một lỗi và có thể dẫn đến những kết quả sai. Nếu dựa trên chiến lược kiểm tra thường xuyên, có thể giả thiết rằng chúng ta sẽ phát hiện được lỗi trước khi lỗi thứ hai xuất hiện. Nhưng điều này có thể không thực hiện được nếu lỗi thứ nhất không được phát hiện.

Khi chúng ta tạo ra các vectơ giá trị thử nghiệm cho một mạch, mục đích chính là tạo được một tập hợp đầy đủ các vectơ giá trị kiểm nghiệm để phát hiện lỗi. Tập hợp này sẽ cho phép phát hiện mọi lỗi có thể phát hiện được. Tuy nhiên, tập hợp này cũng không đủ để phát hiện mọi lỗi có thể phát hiện được nếu trong mạch tồn tại những lỗi chưa được phát hiện. Ví dụ, trong mạch trên hình 8.9a, lỗi hằng số trên đường tín hiệu *b s-a-0* làm cho giá trị đường là '0' *b/0* có thể được phát hiện nếu sử dụng vectơ giá trị kiểm nghiệm *t* = "1101". Lỗi này trở thành không phát hiện được nếu




**Hình 8.9.** Ảnh hưởng của lỗi *a/1* không phát hiện được trong mạch tới việc phát hiện lỗi thứ hai *b/0*.

trong mạch có lỗi hằng số *a/1 s-a-1*( hình 8.9b ).

Trường hợp tồn tại một lỗi không phát hiện được ngăn chặn việc tìm các lỗi khác trong mạch không chỉ giới hạn trong việc tìm những lỗi cùng loại ( như trường hợp các lỗi hằng số ảnh hưởng tới các lỗi hằng số ) mà còn có thể đúng đối với những lỗi khác loại, ví dụ như: một lỗi bắc cầu, không phát hiện được có thể làm cho việc phát hiện lỗi cố định khác không thể thực hiện được. Ta hãy xét ví dụ mạch trên hình 8.10.

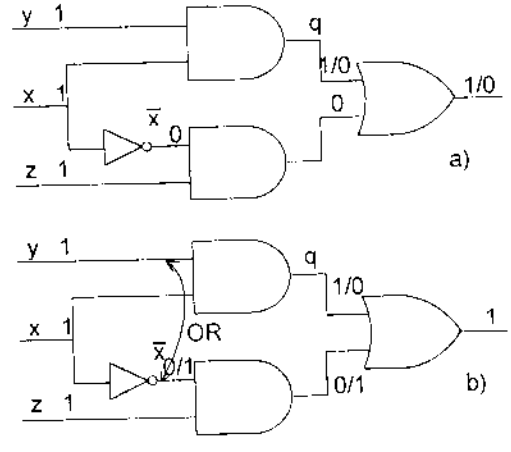
Mạch trên hình 8.10 thực hiện hàm lôgic:

$$w = xy + \bar{x}z$$

Lỗi bắc cầu OR giữa đường tín hiệu $y$ và $\bar{x}$ là không phát hiện được vì:

$$w_f = xy + yz + \bar{x}z = xy + \bar{x}z$$

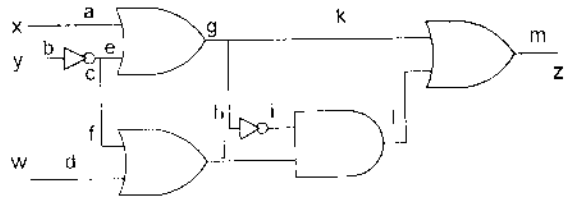
Trên hình 8.10a, ta thấy vectơ giá trị kiểm nghiệm $t$ = "111" cho phép phát hiện lỗi hằng số *s-a-0* trên đường tín hiệu *q q/0*. Khi trong mạch xuất hiện lỗi bắc cầu OR nói trên thì vectơ giá trị kiểm nghiệm này không thể phát hiện được lỗi *q/0* (hình 8.10b). Ta nhận thấy tập hợp các vectơ giá trị kiểm nghiệm **T** = {111, 010, 001, 101} là tập hợp đầy đủ để phát hiện lỗi hằng số đơn và vectơ $t$ = "111" là vectơ giá trị kiểm nghiệm duy nhất phát hiện lỗi *q/0*. Như vậy tập hợp **T** trở thành không đầy đủ khi trong mạch xuất hiện lỗi bắc cầu không phát hiện được nói trên.




**Hình 8.10.** Trường hợp lỗi bắc cầu OR không phát hiện được ảnh hưởng tới việc phát hiện lỗi hằng số *q/0*.

Những mạch tổ hợp dư thừa có thể được đơn giản hóa bằng cách loại bỏ ít nhất một phần tử lôgic hoặc một đầu vào của phần tử. Ví dụ, giả thiết rằng lỗi *f s-a-1* xuất hiện tại đầu vào *i* của phần tử AND là không thể phát hiện được. Do chức năng của mạch không thay đổi khi xuất hiện lỗi *f* nên ta có thể đặt giá trị '1' thường trực tại đầu vào *i* này. Do phần tử AND với *n* đầu

vào trong đó có một đầu vào *i* nhận giá trị '1' sẽ tương đương với phần tử AND với (*n* - 1) đầu vào. Trong đó phần tử AND với (*n* - 1) đầu vào nhận được từ phần tử *n* đầu vào ban đầu bằng cách loại bỏ đầu vào *i* đó. Tương tự như vậy, nếu phần tử AND có lỗi *s-a-0* tại đầu vào là không thể tìm được thì phần tử AND đó có thể loại bỏ và thay thế bằng tín hiệu '0'.




**Hình 8.11.** Ví dụ về lỗi dư thừa và lỗi tương đương.

Nếu hai lỗi *f* và *g* khi xuất hiện trong mạch làm cho các hàm ra của mạch $Z_f(x)$ và $Z_g(x)$ thoả mãn hệ thức $Z_f(x) \equiv Z_g(x)\ \forall x$, khi đó những lỗi này được gọi là tương đương về mặt chức năng.

Vectơ giá trị đầu vào *t* làm hai lỗi *f* và *g* phân biệt được nếu $Z_f(t) \neq Z_g(t)$ và hai lỗi *f* và *g* sẽ được gọi là phân biệt được. Chúng ta không thể tìm được vectơ giá trị kiểm nghiệm *t* cho phép phân biệt hai lỗi tương đương về mặt chức năng. Quan hệ tương đương về mặt chức năng giữa các lỗi cho phép phân hoạch tập hợp các lỗi thành các lớp tương đương. Để phân tích lỗi, chúng ta chỉ cần khảo sát một lỗi đại diện cho từng lớp tương đương. Ta hãy xét ví dụ mạch tổ hợp trên hình 8.11. Hàm ra của mạch có dạng:

$$z = Z(x, y, w) = x + \bar{y} + \bar{x}\,y\,\bar{w}$$

Nếu trong mạch xuất hiện lỗi hằng số *f/0* ta sẽ có :

$$Z_{f/0}(x, y, w) = Z(x, y, w)$$

Vậy lỗi *f/0* là lỗi dư thừa. Nếu trong mạch xuất hiện lỗi hằng số *d/1* hoặc *f/1* thì:

$$Z_{d/1}(x, y, w) = Z_{f/1}(x, y, w) = x + \bar{y}$$

Vậy các lỗi hằng số *d/1* và *f/1* là những lỗi tương đương. Tương tự như vậy, các lỗi hằng số *j/0* và *l/0* cũng là những lỗi tương đương.

Các lỗi dư thừa và tương đương là hệ quả của việc xét mạch là một hộp đen và chỉ quan sát hành vi của mạch qua các đầu vào và đầu ra của mạch. Nếu ta có thể quan sát giá trị tín hiệu tại những đường truyền bên trong mạch thì những lỗi này sẽ phát hiện được. Nếu chúng ta đưa ra phương pháp

xác định được không những sự tồn tại của lỗi mà còn và chỉ ra được vị trí lỗi, thì phương pháp đó gọi là phương pháp chuẩn đoán lỗi.

Đối với hai lỗi bất kỳ *f* và *g*, chúng ta xác định hàm khoảng cách giữa hai lỗi đó như sau:

$$F_{fg}(x) = F_f(x) \oplus F_g(x) = Z_f(x) \oplus Z_g(x)$$

Nếu vectơ giá trị thử nghiệm *t* làm cho $F_{fg}(t) = 1$, ta nói rằng vectơ giá trị thử nghiệm *t* phân biệt hai lỗi *f*, *g* và được ký hiệu là $t^{fg}$.

Giả thiết rằng **P** là tập hợp các lỗi có thể có và mạch được thiết kế không chứa lỗi. Nếu chúng ta xác định tập hợp tất cả những vectơ giá trị kiểm nghiệm phân biệt hai lỗi bất kỳ $\mathbf{T} = \{ t^{fg} \}$ đối với mọi phần tử thuộc tập hợp **P** thì tập hợp $\mathbf{T} = \{ t^{fg} \}$ trở thành tập hợp các vectơ giá trị thử nghiệm cho phép thực hiện việc chuẩn đoán lỗi và được gọi là tập hợp các vectơ chuẩn đoán lỗi. Nói chung, khi xuất hiện các lỗi tương đương, chúng ta cần xác định tập hợp các vectơ chuẩn đoán đối với những lỗi đại diện cho các lớp lỗi tương đương về mặt chức năng.

Ví dụ, chúng ta xét mạch biểu diễn trên hình 8.12:

Nếu xác định các lỗi hằng số *s-a-0*, *s-a-1* trên các đường truyền tín hiệu của mạch, ta có các phân nhóm lỗi như sau:

$$Z_{a1} = Z_{b0} = Z_{c1} = Z_{d1} = Z_{k1} = Z_{l1} = Z_{h1} = Z_{i0l} = 1;$$

$$Z_{a0} = \bar{y} + \bar{w};$$

$$Z_{b1} = Z_{c0} = x + \bar{w};$$

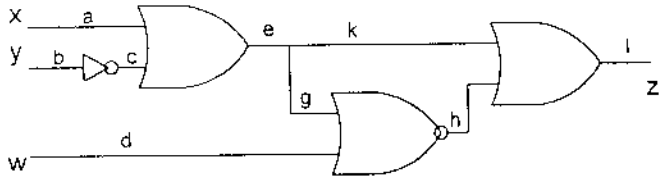
$$Z_{d1} = Z_{g1} = Z_{h0} = x + \bar{y};$$

$$Z_{e0} = \bar{w};$$

$$Z_{k0} = \bar{x}\, y\, \bar{w};$$

$$Z_{l0} = 0;$$

$$Z_{g0} = x + \bar{y} + \bar{w};$$




**Hình 8.12.** Ví dụ minh hoạ cho việc chuẩn đoán lỗi bằng các vectơ giá trị thử nghiệm.

trong đó lỗi *g/0* là lỗi dư thừa vì $Z_{g/0}(x) \equiv Z(x)$. Như vậy chúng ta thấy rằng chúng ta có bảy lỗi đại diện. Các vectơ giá trị kiểm nghiệm để chuẩn đoán các lỗi đại diện được đưa ra trong bảng sau:

| Đầu vào | | | Lỗi | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| x | y | w | a/1 | a/0 | b/1 | d/1 | e/0 | k/0 | l/0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |

Từ bảng này ta thấy, đối với mỗi cặp lỗi $f$, $g$ ta đều có thể tìm được vectơ giá trị đầu vào $t$ phân biệt chúng ( có nghĩa là $F_{fg}(t) = 1$ ). Do đó, tập hợp các vectơ giá trị kiểm nghiệm trong bảng là tập hợp chuẩn đoán các lỗi hằng số trên các đường tín hiệu trong mạch tổ hợp. Bảng chứa các giá trị của hàm khoảng khoảng cách lỗi $F_f(x)$ đối với các vectơ giá trị thử nghiệm $t$ và lỗi $f$ gọi là bảng chuẩn đoán lỗi.

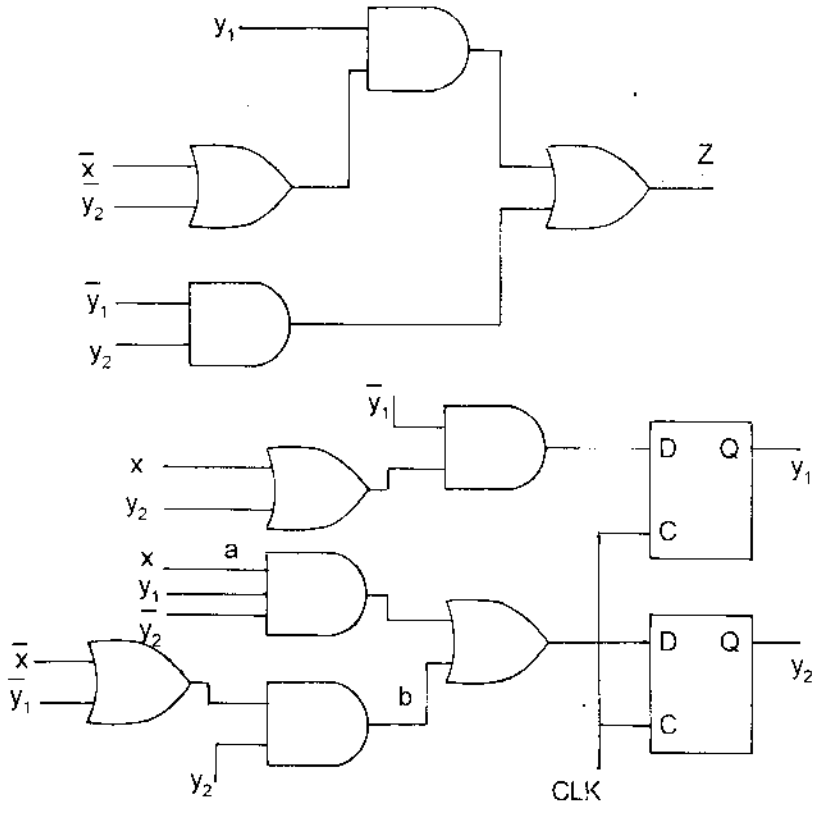
**2. Phát hiện lỗi trong mạch tuần tự**

Kiểm tra các mạch tuần tự phức tạp hơn so với trường hợp mạch tổ hợp. Để có thể phát hiện được lỗi, chúng ta cần phải sử dụng chuỗi các vectơ giá trị kiểm nghiệm và đáp ứng của mạch tuần tự sẽ là hàm đối với các trạng thái ban đầu của mạch. Chúng ta sẽ đưa ra một số minh hoạ quá trình phát hiện lỗi cho mạch tuần tự.

Giả thiết $\boldsymbol{T}$ là chuỗi vectơ giá trị thử nghiệm và $R(q, \boldsymbol{T})$ là đáp ứng của mạch tuần tự $N$ đối với chuỗi $\boldsymbol{T}$ bắt đầu từ trạng thái $q$. Giả thiết rằng khi trong mạch $N$ xuất hiện lỗi $f$, mạch $N$ trở thành mạch $N_f$ và $R_f(q_f, \boldsymbol{T})$ là đáp ứng của mạch chứa lỗi $N_f$ đối với chuỗi $\boldsymbol{T}$ bắt đầu từ trạng thái $q_f$. Chúng ta đưa ra định nghĩa sau:

Chuỗi $T$ các vectơ giá trị kiểm nghiệm **chắc chắn phát hiện** lỗi $f$ trong mạch tuần tự nếu và chỉ nếu các đáp ứng đầu ra $R(q, T)$ và $R_f(q_f, T)$ khác nhau với mỗi cặp trạng thái ban đầu $q$ và $q_f$.

Trên hình 8.13 đưa ra mạch tuần tự với bảng chuyển trạng thái, dãy giá trị tín hiệu đầu ra là đáp ứng của mạch đối với dãy tín hiệu đầu vào $T = 10111$ trong trường hợp mạch không có lỗi. Chúng ta xét mạch trong những trường hợp mạch chứa lỗi hằng số $f$ *s-a-1* trên đường $a$ *a/1* , trường hợp và lỗi hằng số $g$ *s-a-0* trên đường $b$ *b/0*. Ta nhận thấy tất cả các chuỗi đáp ứng của mạch khi chứa lỗi $g$ đều khác với các đáp ứng của mạch không chứa lỗi đối với cùng một tác động, như vậy chuỗi $T$ nói trên chắc chắn phát hiện được lỗi $g$. Mặt khác, mọi đáp ứng của mạch không chứa lỗi đối với chuỗi $T$ xét từ trạng thái khởi tạo $B$ đều trùng với đáp ứng của mạch chứa lỗi $f$ xét tại cùng trạng thái bắt đầu $B$, như vậy chuỗi $T$ không chắc chắn phát




**Hình 8.13.** Minh hoạ trường hợp phát hiện lỗi cho mạch tuần tự.

hiện được lỗi *f*.

Ví dụ nói trên cho thấy rằng, mặc dù chuỗi vectơ giá trị thử nghiệm $\boldsymbol{T}$ chắc chắn phát hiện được lỗi *g*, nhưng chúng ta xác định được các dấu hiệu của lỗi một cách không đơn giản. Thật vậy, chúng ta không thể khẳng định rằng tại một vị trí nào đó trong dãy đáp ứng, mạch không bị lỗi sẽ cho ra giá trị '1' và mạch chứa lỗi sẽ cho ra giá trị '0' hoặc ngược lại. Thay vào đó, chúng ta phải liệt kê ra mọi đáp ứng có thể có của mạch bình thường và mạch chứa lỗi. Nhưng điều này là không thực tế vì đối với mạch tuần tự với *n* phần tử nhớ sẽ có $2^n$ khả năng của trạng thái khởi động. Thêm nữa, chúng ta cần phải xác định quy trình kiểm tra cho nhà thiết kế. Nhà thiết kế phải thực hiện các phép thử qua các chân vào ra của mạch và so sánh các đáp ứng đo đạc được với những đáp ứng mong muốn trên lý thuyết đối với từng vectơ giá trị kiểm nghiệm. Như vậy các đáp ứng mong muốn phải được xác định trước và suy ra cả những đáp ứng của mạch chứa lỗi cũng phải được xác định hoặc dự đoán được. Để thực hiện điều này, chúng ta đưa ra định nghĩa đã được làm yếu đi của khái niệm phát hiện lỗi đối với mạch tuần tự như sau:

Chuỗi $\boldsymbol{T}$ các vectơ giá trị kiểm nghiệm phát hiện được lỗi *f* nếu và chỉ nếu đối với mỗi cặp trạng thái khởi động có thể có *q* và $q_f$, các chuỗi đáp ứng $R(q, \boldsymbol{T})$ và $R_f(q, \boldsymbol{T})$ khác biệt nhau đối với một vectơ $t_i$ cụ thể nào đó trong chuỗi $\boldsymbol{T}$.

Để xác định được vectơ $t_i$ khi lỗi f có thể quan sát được tại đầu ra của mạch, một cách độc lập với các trạng thái khởi động *q* và $q_f$, các phép kiểm nghiệm thường được thực hiện qua hai giai đoạn khác nhau:

- Trong giai đoạn thứ nhất, chúng ta đặt chuỗi vectơ giá trị khởi động $\boldsymbol{T}_1$ vào mạch sao cho sau khi thực hiện đến cuối chuỗi $\boldsymbol{T}_1$, mạch *N* và $N_f$ được đưa về hai trạng thái đã biết $q_1$ và $q_{1f}$. Các đáp ứng của hai mạch *N* và $N_f$ trong khi áp dụng chuỗi vectơ $\boldsymbol{T}_1$ được bỏ qua vì chúng không dự đoán được.
- Trong giai đoạn thứ hai, chúng ta đặt chuỗi vectơ $\boldsymbol{T}_2$ và các đáp ứng

| | $x$ | | $y_1$ | $y_2$ |
|---|---|---|---|---|
| A | A, 0 | D, 0 | 0 | 0 |
| B | C, 1 | C, 1 | 0 | 1 |
| C | B, 1 | A, 0 | 1 | 1 |
| D | A, 1 | B, 1 | 1 | 0 |

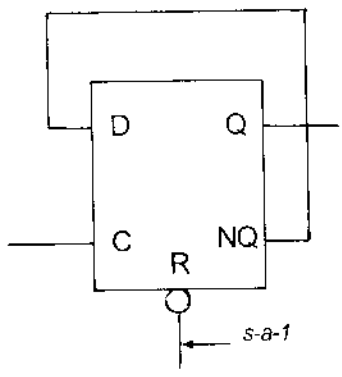
| Trạng thái ban đầu | Chuỗi tín hiệu đầu ra | | |
|---|---|---|---|
| | Không lỗi | lỗi *f a/1* | lỗi *g b/0* |
| A | 01011 | 01010 | 01101 |
| B | 11100 | 11100 | 11101 |
| C | 00011 | 00010 | 01010 |
| D | 11001 | 10010 | 11010 |

**Hình 8.14.** Bảng chuyển trạng thái và bảng xác định chuỗi tín hiệu ra trong trường hợp không lỗi và có lỗi *f* hoặc *g* của mạch trên hình 8.13.

$R(q_0, T_p)$ và $R_f(q_0, T_p)$ là các chuỗi dự đoán được. Thông thường vectơ giá trị kiểm nghiệm $t_i$ là vectơ đầu tiên trong chuỗi $T_p$ mà khi đặt vào mạch sẽ phát hiện được lỗi.

Phương pháp nói trên để kiểm nghiệm và phát hiện lỗi cho mạch tuần tự dựa trên cơ sở giả thiết rằng tồn tại chuỗi khởi tạo $T_I$. Đối với hầu hết các mạch sử dụng trong thực tế đều có thể tìm được chuỗi vectơ giá trị khởi tạo, chuỗi này cho phép mạch bắt đầu hoạt động từ một trạng thái đã biết. Thông thường các mạch sẽ được thiết kế sao cho chúng có thể dễ dàng được khởi tạo. Phương pháp thông thường nhất là sử dụng các đường tín hiệu khởi tạo và đặt giá trị đầu cho từng phần tử trigơ. Tuy nhiên, chuỗi vectơ giá trị khởi tạo cho mạch không chứa lỗi $N$ có thể không đáp ứng với việc khởi tạo mạch $N_f$ chứa lỗi $f$ nào đó. Những lỗi $f$ như vậy được gọi là lỗi ngăn chặn khởi tạo.

Ta hãy xét một ví dụ về những lỗi ngăn chặn khởi tạo đã đề cập tới. Trên hình 8.15 đưa ra sơ đồ của trigơ D được kết nối để tạo thành bộ đếm 1 bit. Trên đường tín hiệu R xuất hiện lỗi hằng số *s-a-1*. Chúng ta thấy rằng mạch không chứa lỗi $N$ hoàn toàn được khởi tạo bởi tín hiệu $R$ = '*0*' , nhưng với mạch chứa lỗi hằng số *R/1* thì không thể khởi tạo mạch về trạng thái đó.




**Hình 8.15.** Minh họa lỗi ngăn chặn khởi tạo.

Chúng ta có thể sinh ra dãy các vectơ thử nghiệm $T$ để phát hiện lỗi ngăn chặn khởi tạo trạng thái bằng cách phân tích một cách riêng biệt mỗi trạng thái khởi tạo khi xuất hiện lỗi, nhưng nói chung, phương pháp này không thể áp dụng được trên thực tế. Do đó các lỗi ngăn chặn khởi tạo trạng thái phải được coi là không phát hiện được nếu chỉ dùng các phép kiểm tra qua các chân vào/ra của mạch với các phép so sánh trên toàn bộ các đầu ra. Tuy nhiên không giống như trong trường hợp đối với mạch tổ hợp, điều này không có nghĩa là mạch điện có tính dư thừa. Chúng ta cũng nên biết rằng các lỗi ngăn chặn khởi tạo trạng thái có thể được phát hiện bằng những phương pháp kiểm nghiệm khác.

## §8.3. Các phương pháp thuật toán tổng hợp các giá trị thử nghiệm

Như chúng ta đã đề cập tới trong mục trước, hành vi của mạch khi xuất hiện lỗi phụ thuộc nhiều vào chức năng của các mạch lôgic khi không bị lỗi. Độ phức tạp của việc tìm những bộ giá trị thử nghiệm phát hiện lỗi phụ thuộc nhiều vào loại mạch: mạch tổ hợp hoặc mạch tuần tự. Những mạch tuần tự là những mạch chứa các vòng phản hồi nên việc mô hình hoá chức năng của mạch cũng như việc xây dựng các bộ giá trị thử nghiệm phức tạp hơn rất nhiều so với trường hợp các mạch tổ hợp. Độ phức tạp của các phương pháp tạo các bộ giá trị phát hiện lỗi cũng phụ thuộc rất nhiều vào loại lỗi đang nghiên cứu. Trong khuôn khổ giáo trình này, chúng tôi chỉ giới hạn trong việc nghiên cứu các phương pháp tạo các bộ giá trị thử nghiệm phát hiện lỗi đối với các mạch tổ hợp và cũng giới hạn trong việc xét các lỗi hằng số.

### 1. Phương pháp dựa trên sự kích hoạt đường truyền

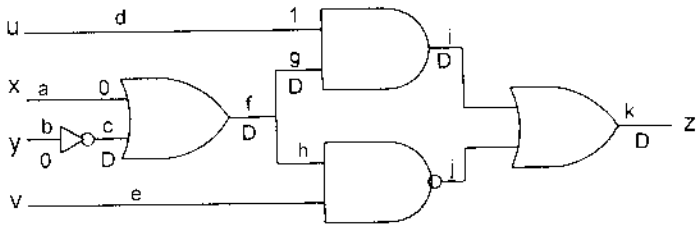
Các phương pháp tạo dãy giá trị thử nghiệm có thể chia làm hai nhóm:

- Các phương pháp tìm các vectơ giá trị đầu vào có thể phát hiện một lỗi cho trước.
- Các phương pháp tìm các lỗi có thể được phát hiện sử dụng một vectơ giá trị kiểm nghiệm cho trước. Các phươgn pháp thuộc nhóm này còn được gọi là các phương pháp mô hình hoá lỗi.

Trong mục này chúng ta nghiên cứu các phương pháp loại thuộc nhóm thứ nhất. Đối tượng được kiểm nghiệm trong phần này là các mạch tổ hợp và chúng ta chỉ xét các mô hình lỗi hằng số. Các phương pháp tạo vectơ giá trị thử nghiệm cho các lỗi có thể có loại trừ các lỗi dư thừa được gọi là các phương pháp thuật toán. Như vậy, mô hình hoá lỗi không thể gọi là phương pháp thuật toán vì nếu quá trình mô hình hoá không được thực hiện với mọi khả năng có thể có của các vectơ giá trị đầu vào, chúng ta không thể đảm bảo được việc tìm ra các vectơ giá trị kiểm nghiệm cho tất cả các lỗi có thể được phát hiện.

Các phương pháp thuật toán tạo các vectơ giá trị thử nghiệm để phát hiện lỗi được sử dụng hiện nay đều dựa trên khái niệm kích hoạt đường truyền.

Kích hoạt đường truyền là phương pháp xác định các vectơ giá trị thử nghiệm dựa trên việc tìm đường mà theo đó sự khác biệt giữa các giá trị tín hiệu khi mạch chứa lỗi và khi mạch không có lỗi được truyền ra bên ngoài. Nói cách khác là chúng ta phải xác định được các đường dẫn mẫn cảm với lỗi. Ví dụ, xét mạch tổ hợp trên hình 8.16, giả thiết rằng trong mạch xuất hiện lỗi hằng số *s-a-0* trên đường *c* là *c/0*. Để phát hiện được lỗi này ta cần xác định vectơ giá trị đầu vào sao cho trên đường *c* nhận giá trị là *1*( tức là giá trị nghịch đảo với giá trị lỗi ). Điều này sẽ dẫn tới giá trị trên đường *b* phải bằng '0' suy ra giá trị trên đường vào *y* cũng phải bằng '0':




**Hình 8.16.** Minh họa khái niệm kích hoạt đường truyền.

$$c = \text{'1'} \Rightarrow b = \text{'0'} \Rightarrow y = \text{'0'}.$$

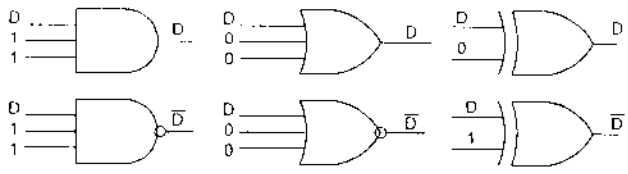
Đại lượng có giá trị bằng '1' khi mạch không lỗi và bằng '0' trong trường hợp mạch có lỗi được ký hiệu là *D* ( defect - lỗi ). Trở lại ví dụ trên hình 8.16, chúng ta thấy khi tín hiệu trên đường *y* có giá trị bằng *'0'*, đường tín hiệu *c* sẽ nhận giá trị bằng '1'. Trong trường hợp xuất hiện lỗi hằng số *s-a-0*, đường *c* sẽ nhận giá trị cố định bằng *'0'*, nói cách khác trên đường tín hiệu *c* xuất hiện giá trị *D*. Như vậy, chúng ta có một số nhận xét như sau:

- Nếu đường tín hiệu $x$ = '0' thì đường *f* nhận giá trị *D*, suy ra đường tín hiệu *g* và *h* cùng nhận giá trị *D*;
- Nếu đường tín hiệu $u$ = '1' thì đường *i* nhận giá trị *D*;
- Để giá trị *D* xuất hiện ở đầu ra *z* của mạch ( trên đường tín hiệu *k* ) thì đường tín hiệu *j* phải nhận giá trị '0' suy ra đường tín hiệu *e* phải nhận giá trị '1' và đường tín hiệu *v* nhận giá trị '1'.

Chúng ta thấy, với các giá trị tín hiệu đầu vào $x$ = '0', $y$ = '0', $u$ = '1', $v$ = '1', giá trị *D* xuất hiện trên đường tín hiệu *c* sẽ lan truyền tới được đầu ra. Như vậy đường tín hiệu *k* sẽ phản ánh đầy đủ mọi biến động trên đường *c*, nói cách khác đường dẫn *c-f-g-i-k* là đường dẫn mẫn cảm với lỗi. Như vậy, các giá trị tín hiệu $x$ = '0', $y$ = '0', $u$ = '1', $v$ = '1' tạo thành vectơ giá trị thử nghiệm phát hiện lỗi hằng số *c/0* và đường tín hiệu ra nhận giá trị '1' trong trường hợp mạch không bị lỗi và nhận giá trị '0' khi xuất hiện lỗi *c/0*.

Trong trường hợp trong mạch xuất hiện lỗi hằng số *s-a-1* làm cho đường tín hiệu nhận giá trị cố định '1', trong định nghĩa của đại lượng $D$ các giá trị '0' và '1' đổi chỗ cho nhau và thay cho khái niệm $D$ ta đưa vào khái niệm $\bar{D}$. Các giá trị $D$ và $\bar{D}$ nhận được trên những đường bị lỗi gọi là các *khối-D*. Quá trình truyền các *khối-D* tới đầu ra của mạch gọi là phép truyền $D$ và thủ tục thực hiện phép truyền $D$ gọi là thủ tục chuyển $D$.

Đối với từng phần tử lôgic cơ sở, quá trình thiết lập và truyền $D$ là quá trình xác định và đặt các giá trị tương ứng tại các đầu vào còn lại của phần tử khi một trong các đầu vào của phần tử bị lỗi và nhận giá trị $D$ hoặc $\bar{D}$, sao cho giá trị $D$ ( hoặc $\bar{D}$ ) được truyền nguyên vẹn tới đầu ra của phần tử. Trên hình 8.17, ta thấy đối với các phần tử AND và NAND, phép truyền $D$ là thiết lập trên những đầu vào không bị lỗi những giá trị bằng '1'; đối với các phần tử OR và NOR - thiết lập giá trị '0'; trong khi đó đối với phần tử XOR, phép truyền D tương ứng với việc thiết lập trên đầu vào còn lại giá trị bất kỳ trong tập hợp { 0, 1 }.Chúng ta nhận thấy đối với các phần tử NAND, NOR, XOR, giá trị trên đầu ra bị đảo lại:



**Hình 8.17.** Thủ tục chuyển D qua các phần tử lôgic cơ bản.

$$D \rightarrow \bar{D},\ \bar{D} \rightarrow D$$

Dựa trên khái niệm *khối-D*, các phương pháp tạo vectơ giá trị kiểm nghiệm phát hiện các lỗi hằng số dựa trên sự kích hoạt đường truyền tín hiệu và thiết lập đường dẫn mẫn cảm với lỗi được thực hiện qua ba giai đoạn:

1. Đối với những lỗi hằng số trên một đường tín hiệu xác định, chúng ta thiết lập *khối-D* trên phần tử nhận đường tín hiệu làm đầu vào và xác định các giá trị thoả mãn *khối-D* đó ( thủ tục thiết lập $D$ ). Kết quả của bước này là xác định được các giá trị đầu vào trên phần tử trực tiếp chịu ảnh hưởng của lỗi.
2. Giá trị của *khối-D* được đưa ra đầu ra của toàn mạch bằng thủ tục truyền $D$. Kết quả của bước này là xác định được đường dẫn mẫn cảm với lỗi để truyền giá trị $D$ tới đầu ra. Trong một số mạch có thể xuất hiện nhiều đường dẫn mẫn cảm với lỗi ( một trường hợp dễ thấy khi các phần tử lôgic có thể có nhiều hơn hai đầu vào ).

3. Xác định các giá trị đầu vào tương thích với các giá trị '0' và '1' nhận được trong quá trình thiết lập *D* và truyền *D*.

Các bước 2 và bước 3 được thực hiện dựa vào những thao tác chính như sau:

- Thao tác truyền-*D*.
- Phép suy diễn.

  Phép quay lui.

Phép suy diễn cần thiết để xác định giá trị trên những đường truyền tín hiệu một cách duy nhất theo giá trị trên những đường tín hiệu khác. Điều này có thể được minh hoạ bằng ví dụ trên hình 8.16, nếu giá trị tại đường tín hiệu *f* được thiết lập bằng '0' thì suy ra giá trị trên đường *h* và *g* cũng được thiết lập bằng '0'. Theo chức năng của phần tử AND, nếu đường *g* được thiết lập bằng '0', suy ra đường *i* sẽ được thiết lập bằng '0'. Mặt khác để thiết lập được đường *f* giá trị '0', giá trị trên đường *a* và *c* chắc chắn phải bằng '0'. Giá trị trên đường *c* bằng '0' sẽ suy ra đường *b* phải nhận giá trị '1'. Ta có thể tổng kết quá trình suy diễn trên như sau:

$$f = 0 \Rightarrow h = 0,\ g = 0;\ g = 0 \Rightarrow i = 0;$$

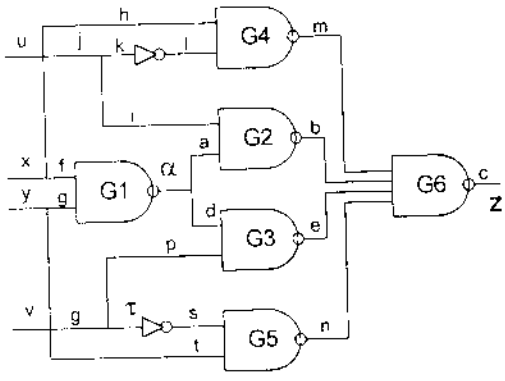
$$f = 0 \Rightarrow a = 0,\ c = 0;\ c = 0 \Rightarrow b = 1.$$

Phép quay lui là phép toán thiết lập giá trị trên các đầu vào của mạch sao cho không có mâu thuẫn với các giá trị tại các đường truyền trong mạch. Phép toán này khác với phép suy diễn nói trên là kết quả có thể không duy nhất. Ví dụ như trên hình 8.16, nếu đường *j* nhận giá trị bằng '0' thì theo phép quay lui ta thiết lập trên đường *h* và *e* các tổ hợp có thể có là ( 1,0 ), ( 0, 1 ), ( 1, 1 ). Việc lựa chọn các giá trị phù hợp phụ thuộc vào từng trường hợp cụ thể và dựa vào các luật lựa chọn. Các luật lựa chọn có thể có những đặc điểm như sau:

- Các luật lựa chọn có thể là những luật như sau:
  - Chọn đường tín hiệu gần với đầu vào nhất;
  - Trong trường hợp giá trị tín hiệu bằng '1', lựa chọn đường truyền chứa phần tử OR;
  - Trong trường hợp giá trị tín hiệu bằng '0', lựa chọn đường truyền chứa phần tử AND.

  Việc đưa các luật lựa chọn cần thiết ngay cả đối với quá trình truyền *D*. Trong trường hợp nếu ở đầu ra của phần tử có rẽ nhánh thì sẽ không duy nhất trong việc chọn đường. Trong nhiều trường hợp kinh nghiệm cho thấy nên chọn đường tín hiệu gần đầu ra nhất.

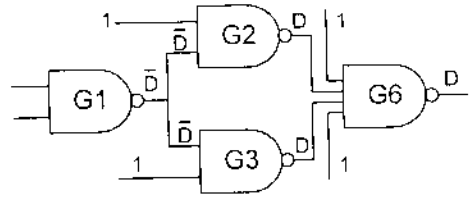
Trong quá trình truyền $D$ thông thường sẽ không đầy đủ nếu chỉ xét một đường truyền. Để giải quyết các vấn đề nảy sinh khi chỉ xét một đường truyền $D$ duy nhất, trong nhiều trường hợp chúng ta cần phải sử dụng nhiều đường truyền $D$ khác nhau một cách đồng thời. Trong ví dụ dưới đây, trong mạch có thể xuất hiện lỗi, nhưng lỗi này không thể phát hiện được nếu chỉ xét một đường truyền $D$. Do đó, nếu các đường tín hiệu có phân nhánh, nói chung chúng ta phải xem xét việc truyền giá trị $D$ theo nhiều đường dẫn khác nhau. Ta xét ví dụ trên hình 8.18, trong mạch xuất hiện lỗi hằng số *s-a-1* trên đường tín hiệu $\alpha$ - lỗi $\alpha/1$. Trong ví dụ này chúng ta thấy, để thiết lập *khối-D* tương ứng với lỗi $\alpha/1$ giá trị trên đường $\alpha$ phải được thiết lập bằng '0' trong trường hợp không có lỗi, điều này sẽ tương đương với việc đặt giá trị $x = y =$ '1'. Bây giờ ta sẽ xét các đường truyền giá trị $D$ khác nhau tới đầu ra $z$ của mạch:




**Hình 8.18.** Thiết lập và truyền D theo hai đường dẫn mẫn cảm lỗi khác nhau cho phép phát hiện lỗi $\alpha/1$.

- Nếu lựa chọn đường truyền $D$ qua các đường tín hiệu *a-b-c* thì trên phần tử **G6** theo thủ tục truyền $D$, chúng ta nhận được:

  - Các đường tín hiệu *e*, *m*, *n* phải nhận giá trị '1' ( khi đó giá trị truyền tới đầu ra z sẽ là $\bar{D}$ );
  - Đường *e* phải nhận giá trị '1' sẽ dẫn tới đường *p* phải nhận giá trị '0' vì đường *d* đã nhận giá trị $D$ và chúng ta không truyền $D$ qua đường *d-e-c*. Điều này dẫn tới đường *v* phải nhận giá trị '0'.
  - Đường *n* phải nhận giá trị '1' sẽ dẫn tới đường *s* phải nhận giá trị '0' vì đường y đã nhận giá trị '1' và suy ra đường *t* cũng




**Hình 8.19.** Vectơ giá trị kiểm nghiệm lỗi $\alpha/1$ sẽ tồn tại nếu giá trị D được truyền qua đồng thời cả hai đường a-b-c và d-e-c.

nhận giá trị '1'. Đường s phải nhận giá trị '0' sẽ suy ra đường *v* phải nhận giá trị '1'. Điều này dẫn tới mâu thuẫn với kết luận vừa nhận được với đường *e*.

- Chúng ta sẽ nhận được kết quả tương tự nếu chọn đường truyền *D* là đường dẫn đi qua các đường truyền tín hiệu *d-e-c* do tính đối xứng của mạch.
- Nếu ta chọn đồng thời việc truyền *D* qua cả hai đường *a-b-c* và *d-e-c* thì như đã chỉ ra trên hình 8.19, hai đường tín hiệu còn lại của phần tử **G6** sẽ cùng nhận giá trị '1'. Điều này sẽ dẫn tới hai đường *u* và *v* sẽ đồng thời nhận giá trị '1'. Như vậy với bộ giá trị đầu vào

$$x = y = u = v = \text{'1'}$$

chúng ta sẽ truyền được giá trị D tới đầu ra của mạch, hay nói cách khác chúng ta sẽ phát hiện được lỗi *a/1* với vectơ giá trị đầu vào $(x, y, u, v) = (1, 1, 1, 1)$.

## 2. Thuật toán D

Thuật toán **D** là phương pháp thuật toán đầu tiên dùng để tìm các vectơ giá trị kiểm nghiệm để phát hiện các lỗi hằng số dựa trên phương pháp kích hoạt đường truyền. Trên khía cạnh về khả năng xác định các vectơ giá trị kiểm nghiệm tìm ra mọi lỗi có thể phát hiện được, vấn đề quan trọng nhất là nguyên lý truyền D theo nhiều đường dẫn khác nhau. Nếu xét theo khía cạnh về phương pháp tổng hợp các vectơ giá trị kiểm nghiệm thì vấn đề quan trọng là cần phải kết hợp quá trình truyền D và quay lui như thế nào. Chúng ta sẽ xem xét một số điểm quan trọng.

- Nếu một số đường tín hiệu trong mạch bị rẽ nhánh, khi đó trong quá trình truyền *D* chúng ta sẽ phải lựa chọn đường tín hiệu để truyền giá trị *D* tới đầu ra của mạch.

  Tập hợp các phần tử qua đó có khả năng truyền giá trị *D* (tức là những phần tử với đầu vào có thể nhận các giá trị *D* hoặc $\bar{D}$ còn giá trị đầu ra chưa xác định) gọi là ranh giới *D*. Ranh giới *D* thường được biểu diễn bằng hàng đợi. Những phần tử có khả năng truyền *D* sẽ được đưa vào đầu hàng đợi. Khi phần tử đó không còn khả năng thực

hiện quá trình truyền $D$ thì sẽ lựa chọn phần tử tiếp theo trong hàng đợi.

Phép toán quay lui trong thuật toán **D** gọi là phép toán xác định đầu vào. Để xác định giá trị các đầu vào của những phần tử chưa được xử lý, trong phép toán xác định đầu vào, chúng ta sử dụng phương pháp lựa chọn một cách tuần tự những tổ hợp có thể có của đầu vào. Nếu trong khi lựa chọn ta thiết lập một mức độ ưu tiên, khi đó ta sử dụng những nguyên tắc tương ứng với luật chọn, những khả năng lựa chọn chưa sử dụng đến được lưu trong một hàng đợi gọi là hàng đợi xác định đầu vào.

Trên cơ sở của những nhận xét trên, thuật toán **D** có thể biểu diễn qua các bước như sau:

Thuật toán **D**:

1. Trong tập hợp các lỗi, chúng ta lựa chọn một lỗi hằng số *s-a-v* chưa khảo sát. Lỗi này thiết lập giá trị *v* trên đường tín hiệu *s* và ký hiệu là *s/v*. Khối $D$ tương ứng với lỗi trên đường *s* ký hiệu là $D^v$ ( tức là $D^0$ tương ứng với giá trị $D$ còn $D^1$ tương ứng với $\overline{D}$ ). Phần tử có đầu vào nối với đường tín hiệu *s* được đưa vào hàng đợi ranh giới $D$.
2. Đối với khối $D$, chúng ta xác định giá trị các đầu vào sử dụng thao tác thiết lập $D$. Nếu trong trường hợp này cần phải lựa chọn đường truyền $D$, thì chúng ta lựa chọn đường truyền theo luật **A**:

   **Luật A**: *trong quá trình truyền D, các đường tín hiệu gần với các đầu vào sẽ được lựa chọn trước.*

   Các đường tín hiệu không được lựa chọn còn lại sẽ được đưa vào hàng đợi thiết lập. Trong trường hợp không nhận được những giá trị đầu vào xác định, lỗi được coi là dư thừa và quay về bước 1.
3. Thực hiện phép suy diễn đối với đường tín hiệu trong mạch có thể tham gia vào quá trình thiết lập và truyền $D$. Nếu xuất hiện mâu thuẫn giữa các giá trị nhận được của các đường tín hiệu trong quá trình truyền $D$, thì chúng ta ngừng quá trình truyền $D$, lựa chọn giá trị đầu tiên trong hàng đợi xác định trước và quay về bước 2.
4. Thực hiện phép truyền $D$ đối với giá trị đầu tiên của hàng đợi ranh giới $D$. Đối với phần tử tiếp theo lại nhận giá trị D trên đầu vào và chúng ta lại thực hiện phép lan truyền $D$ qua phần tử này. Các đầu vào của phần tử sẽ được lựa chọn giá trị. Nếu cần thiết phải lựa chọn phần tử để được truyền $D$, chúng ta sử dụng luật **B**:

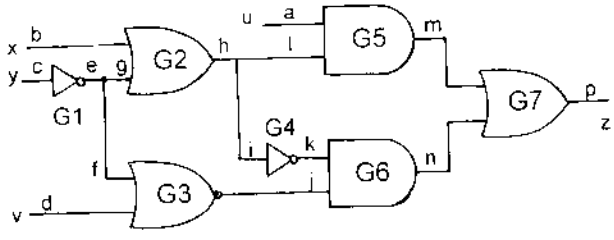
**Luật B:** *Các phần tử gần với những đầu ra nhất sẽ được lựa chọn trước tiên trong quá trình lan truyền giá trị D tới đầu ra.*

Những phần tử không được chọn còn lại được đưa vào hàng đợi. Nếu không thể thực hiện phép truyền $D$, chúng ta lựa chọn giá trị ban đầu tiếp theo. Nếu hàng đợi ranh giới $D$ rỗng, chúng ta cho rằng lỗi đang xét được coi là lỗi dư thừa và quay về bước 1.

5. Thực hiện phép suy diễn đối với tất cả những đường truyền có thể có. Nếu xuất hiện mâu thuẫn trong quá trình truyền D và quay lui, chúng ta quay lại bước 4.
6. Nếu theo kết quả của quá trình truyền $D$, giá trị $D$ hoặc $\bar{D}$ đạt được tới đầu ra, thì chúng ta lựa chọn tiếp phần tử sau của hàng đợi ranh giới $D$. Nếu không lựa chọn được phần tử nào, chúng ta sẽ quay về bước 4.
7. Áp dụng phép toán xác định đầu vào đối với tất cả các phần tử có đầu ra đã xác định và có các đầu vào chưa được xác định và lựa chọn những giá trị đầu vào dùng luật **A** và hàng đợi phép toán xác định đầu vào. Phép toán này được thực hiện lặp lại liên tiếp khi không gặp mâu thuẫn trong giá trị các đầu vào của mạch. Nếu nhận được tập hợp không rỗng các giá trị đầu vào, thì chúng ta coi các vectơ giá trị đầu vào đó đó là vectơ giá trị kiểm nghiệm để phát hiện lỗi *s/v* và sau đó quay về bước 1. Trong trường hợp phép toán xác định đầu vào không thể thực hiện được, thì chúng ta quay về bước 4.

Trong thuật toán **D** được mô tả ở trên, các luật A và B chỉ xác định trình tự lựa chọn đường tín hiệu với các phần tử liên quan và không đóng vai trò quan trọng.

Ta hãy xét ví dụ mạch trên hình 8.20:




**Hình 8.20.** Ví dụ minh hoạ thuật toán **D** đối với mạch có chứa lỗi hằng số khi xuất hiện lỗi h/0.

Chúng ta xác định vectơ giá trị kiểm nghiệm để phát hiện lỗi hằng số *s-a-0* trên đường *h h/0* sử dụng thuật toán **D**.

1. B1. Lựa chọn lỗi *h/0* trong danh sách lỗi; gán cho đường *h* giá trị *D*. Các phần tử nối với đường *h* là **G4** và **G5** được đưa vào hàng đợi ranh giới *D*. Do phần tử **G5** gần đầu ra hơn phần tử **G4** nên sẽ được lựa chọn trước trong quá trình truyền *D* theo luật **B** nên hàng đợi ranh giới *D* là [ **G5**, **G4** ];
2. B2. Để đường *h* có giá trị '1', chúng ta cần phải có hoặc đường *b* nhận giá trị '1' hoặc đường g nhận giá trị '1'. Theo luật **A**, đường *b* gần với đầu vào hơn nên trước tiên chúng ta chọn giá trị đường *b* bằng '1':

   $h = 1 \Rightarrow b = 1 \vee g = 1$; b gần đầu vào hơn, chọn b = 1;

   Do đường b là đường tín hiệu vào nên chúng ta không cần thực hiện bước B3 của thuật toán **D** mà chuyển sang thực hiện bước B4;
3. B4. Thực hiện thao tác truyền *D* qua phần tử **G5**. Giá trị *D* sẽ xuất hiện trên đường tín hiệu *m* và trên đường *a* tín hiệu phải nhận giá trị '1'. Do phần tử **G7** gần với đầu ra hơn so với phần tử **G4** nên phần tử **G7** sẽ được đưa vào đầu hàng đợi ranh giới *D*. Hàng đợi ranh giới D sẽ bằng [ **G7**, **G4** ];
4. B5. Thực hiện phép suy diễn chúng ta nhận được giá trị $\bar{D}$ trên đường tín hiệu *k*;
5. B6. Vì đường tín hiệu *m* không phải là đầu ra của mạch nên quay lại bước B4;
6. B4. Thực hiện thao tác truyền *D* qua phần tử **G7**, đường tín hiệu *p* sẽ nhận giá trị *D* và đường tín hiệu *n* phải nhận giá trị '*0*'. Đường tín hiệu *p* là đầu ra của mạch;
7. B6. Vì giá trị *D* được truyền ra đầu ra của mạch ta chuyển tới thao tác xác định tiếp sau;
8. B7. Trên đường truyền *n* là đối tượng của phép toán xác định đầu vào cho phần tử **G7**, ta có đường *n* phải nhận giá trị '*0*'. Điều này sẽ suy ra trên phần tử **G6**, đường tín hiệu *j* phải nhận giá trị '0' vì đường tín hiệu *k* đã nhận giá trị $\bar{D}$ do phép suy diễn qua phần tử **G4**. Thực hiện thao tác xác định đầu vào đối với phần tử **G6** và giá trị đường tín hiệu *j* bằng '*0*', chúng ta nhận được những tổ hợp giá trị của các đường tín hiệu *d*, *f* là ( 1, 0 ); ( 1, 1 ); ( 0, 1 ).

   Nếu lựa chọn *d* = '1', *f* = '0' suy ra *e* = '1' và phép toán xác định đầu vào trên phần tử **G1** cho ta *c* = '0'.

Như vậy bộ giá trị thử nghiệm phát hiện lỗi h/0 là

x = 1, y = 0, u = 1, v = 1.

Chúng ta xét tiếp ví dụ tìm vectơ giá trị thử nghiệm phát hiện lỗi hằng số s-a-0 trên đường *f f/0* đối với mạch trên hình 8.20.

1. B1. Đường tín hiệu *f* được gán giá trị *D*, hàng đợi ranh giới *D* bao gồm [ G3 ];
2. B2. Với giá trị đường *f* bằng '1', phép toán xác định đầu vào cho ta giá trị trên đường *e* bằng '1' và suy ra đường *c* nhận giá trị bằng '0';
3. B3. Áp dụng phép suy diễn ta nhận được giá trị trên các đường *g*, *h*, *l*, *i*, *k*, *n* như sau:

   g = 1, h = 1, *l* = 1, i = 1, k = 0, n = 0;
4. B4. Thực hiện truyền *D* qua phần tử **G3**, chúng ta nhận được trên đường *j* giá trị *D* và trên *d* giá trị '0'; hàng đợi ranh giới *D* chứa [ G6 ];
5. B6. Do đường tín hiệu *j* không phải là đầu ra nên chúng ta quay lại bước B4;
6. B4. Do giá trị trên đường tín hiệu *k* bằng '0', cho nên chúng ta không thể thực hiện việc truyền *D* qua phần tử **G6**. Mặt khác hàng đợi ranh giới *D* rỗng, điều này dẫn tới kết luận lỗi *f/0* là lỗi dư thừa.

Ta có thể thấy rằng sử dụng thuật toán *D* chúng ta có thể xác định được mọi bộ giá trị thử nghiệm để tìm mọi lỗi hằng số có thể phát hiện được. Thêm vào đó, thuật toán *D* có thể mở rộng để áp dụng sang các mô hình lỗi khác.

### 3. Thuật toán PODEM ( Path-Oriented Decision Making )

Trong thuật toán **D**, đầu tiên giá trị *D* được đưa tới đầu ra của mạch bằng phép truyền *D*, sau đó dùng phép toán xác định đầu vào để tìm giá trị trên các đầu vào. Nếu khi xác định giá trị đầu vào gặp mâu thuẫn thì thực hiện quay lui và lựa chọn đường truyền *D* khác tức là từ bước B7 của thuật toán quay lại bước B4. Trong trường hợp gặp nhiều mâu thuẫn trong phép xác định đầu vào, thời gian tổng hợp vectơ giá trị kiểm nghiệm tăng. Ví dụ, như khi xuất hiện nhiều phần tử XOR trong mạch, các thao tác truyền *D* khá đơn giản nhưng việc áp dụng phép toán xác định đầu vào thường kết thúc không

thành công và quá trình truyền $D$ phải quay lui. Điều này làm cho việc tổng hợp bộ giá trị thử nghiệm bằng thuật toán **D** phải tiêu tốn thêm nhiều thời gian.

Để tăng hiệu quả của quá trình truyền $D$ và xác định đầu vào, nếu tính tới những tình huống đã nêu trên, trong kỹ thuật thường sử dụng phương pháp **PODEM**. Trong phương pháp này, khi thực hiện quay lui ta dùng thủ tục dịch chuyển ngược. Chúng ta có thể trình bày ngắn gọn thủ tục dịch chuyển ngược như sau.

Giả thiết rằng trên đường tín hiệu $x$ thiết lập giá trị $v$. Tại vị trí được đánh nhãn đầu tiên khi đường tín hiệu $x$ được gán giá trị $v$, chúng ta lựa chọn một đường dẫn theo hướng tới đầu vào. Giá trị của nhãn được đảo khi đi qua những phần tử loại NAND, NOR, NOT. Trong quá trình dịch chuyển ngược theo một đường, giá trị nhãn được truyền theo hướng từ đầu ra của mạch tới các đầu vào. Do đó đối với những phần tử có nhiều đầu vào, chúng ta cần thiết phải lựa chọn các đường vào tương ứng với các luật lựa chọn. Những đường tín hiệu không được lựa chọn được đưa vào hàng đợi quay lui và sẽ được sử dụng trong quá trình quay lui.

Các luật lựa chọn đường tín hiệu vào có dạng như sau:

Luật lựa chọn **A**:

1) Lựa chọn đường dẫn gần đầu vào nhất;
2) Nếu giá trị của nhãn bằng '0', chúng ta lựa chọn đường truyền là đầu ra của phần tử AND hoặc NOR. Nếu giá trị của nhãn bằng '1', chúng ta lựa chọn đường truyền là đầu ra của OR hoặc NAND.

Trên cơ sở của thủ tục dịch chuyển ngược nói trên và luật **A** lựa chọn đường truyền, thuật toán PODEM có thể được trình bày như sau:

Thuật toán **PODEM**

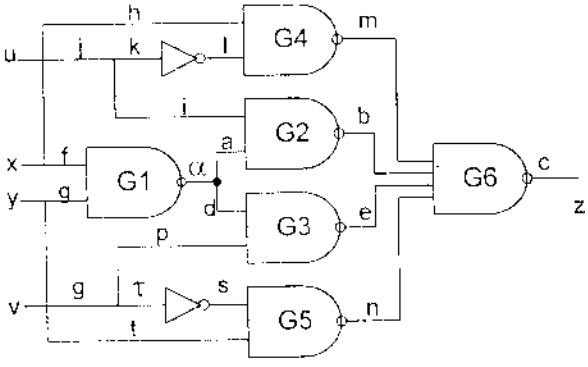
1. Trong tập hợp các lỗi, lựa chọn lỗi hằng số chưa sử dụng $x$-$a$-$v$ trên đường tín hiệu $x$ làm cho đường tín hiệu nhận giá trị $v$ - lỗi $x/v$. Trên đường $x$ thiết lập nhãn $D^v$ và đưa vào hàng đợi ranh giới $D$ những phần tử nhận đường $x$ làm đầu vào. Nếu không còn lỗi trong tập hợp các lỗi, chúng ta dừng quá trình tổng hợp bộ giá trị thử nghiệm.
2. Trên đường tín hiệu $x$, thực hiện thao tác thiết lập $D^v$ và xác định các giá trị đầu vào. Nếu xuất hiện khả năng lựa chọn đường dẫn, sử dụng

luật $\mathfrak{A}$, những phần tử còn lại sẽ được đưa vào hàng đợi thiết lập phục vụ quá trình quay lui.

3. Đối với những giá trị đầu vào nhận được từ kết quả thực hiện thao tác thiết lập $D^{-1}$, chúng ta thực hiện phép suy diễn.
4. Đối với giá trị đầu tiên trong hàng đợi ranh giới $D$, chúng ta thực hiện phép truyền $D$. Phần tử nhận tiếp theo trong phép truyền $D$ nhận giá trị $D$ tại đầu vào được đưa vào đầu hàng đợi ranh giới $D$. Trong quá trình truyền $D$, nếu cần thiết phải lựa chọn phần tử trên đường dẫn mẫn cảm, chúng ta áp dụng luật $\mathfrak{B}$. Thông thường chúng ta có thể sử dụng luật **B** trong thuật toán **D** làm luật $\mathfrak{B}$ của thuật toán **PODEM**. Sau khi đã lựa chọn phần tử trên đường truyền $D$ tới đầu ra, các phần tử còn lại được đưa vào hàng đợi ranh giới $D$. Trong trường hợp thao tác truyền $D$ tới đầu ra không thể kết thúc được, chúng ta sử dụng phần tử tiếp theo trong hàng đợi ranh giới $D$. Nếu hàng đợi ranh giới $D$ rỗng, lỗi được coi là lỗi dư thừa và việc thực hiện thuật toán được quay lại bước 1.
5. Thực hiện phép suy diễn.
6. Thực hiện thủ tục dịch chuyển ngược đối với những đường tín hiệu trên đó giá trị đầu ra của các phần tử được xác định còn các giá trị đầu vào chưa xác định. Trong trường hợp cần thiết phải lựa chọn đường tín hiệu, chúng ta thực hiện lựa chọn đường sử dụng luật $\mathfrak{A}$. Những đường tín hiệu chưa được lựa chọn được đưa vào hàng đợi quay lui.
7. Thực hiện phép suy diễn, sau khi đã gán cho các đầu vào những giá trị nhãn nhận được từ phép quay lui. Kiểm tra tính đúng đắn của giá trị đầu của nhãn. Nếu không nhận được giá trị đúng, chúng ta kiểm tra tính đúng đắn của giá trị nghịch đảo. Nếu việc kiểm tra giá trị cho kết quả đúng, chúng ta chuyển sang bước 6 và sử dụng đường tín hiệu đầu tiên trong hàng đợi quay lui. Trong trường hợp việc kiểm tra cho kết quả sai, như vậy thao tác truyền $D$ tới đầu ra không thể thực hiện được, chúng ta quay về bước 4.
8. Nếu tồn tại đường cho phép thực hiện bước 6, chúng ta thực hiện các bước 6 và bước 7.
9. Nếu giá trị $D$ tới được đầu ra, bộ giá trị đầu vào nhận được sẽ là vectơ giá trị thử nghiệm để phát hiện lỗi *s/v* và chúng ta quay lại

bước 1. Trong trường hợp thao tác truyền $D$ không tới được đầu ra, chúng ta quay lại bước 4.

Chúng ta minh họa thuật toán **PODEM** qua mạch trên hình 8.21. Giả thiết rằng trong mạch xuất hiện lỗi hằng số $s$-$a$-$1$ trên đường $\alpha - \alpha/1$. Tổng hợp vectơ giá trị kiểm nghiệm đối với lỗi $\alpha/1$ theo thuật toán **PODEM** được thực hiện như sau:




**Hình 8.21.** Minh họa thuật toán PODEM trong việc tổng hợp giá trị thử nghiệm phát hiện lỗi hằng số $s$-$a$-$1$ trên đường $\alpha$ $\alpha/1$.

1) B1. Gán cho đường $\alpha$ giá trị $D$, hàng đợi ranh giới $D$ sẽ chứa những phần tử sau:[ G2, G3 ];
2) B2. Để đường tín hiệu $\alpha$ nhận giá trị '0', chúng ta cần thiết đường tín hiệu $f$ và $g$ nhận giá trị bằng '1';
3) B3. Thực hiện phép suy diễn đối với đường f và g, chúng ta nhận được hai đường tín hiệu $h$ và $t$ cũng nhận giá trị '1';
4) B4. Thực hiện thao tác truyền $D$ qua phần tử **G2**, chúng ta nhận được giá trị đường $b$ = D và đường $i$ phải nhận giá trị '1'. Từ đó suy ra hàng đợi ranh giới $D$ sẽ bao gồmcác phần tử [ G6, G3 ];
5) B5. Áp dụng phép suy diễn ta nhận được $j = k$ = '1'; $l$ = '0', $m$ = '1'. Do $j$ là đầu vào của mạch nên các bước từ bước B6 đến B8 của thuật toán **PODEM** được bỏ qua và chúng ta sang bước B9;
6) B9. Do đường tín hiệu $b$ không phải là đầu vào của mạch, chúng ta chuyển sang lại bước B4;
7) B4. Thực hiện thao tác truyền $D$ qua phần tử **G6**. Để thực hiện được điều này chúng ta cần có đường $e$ nhận giá trị '1', đường $n$ cũng phải nhận giá trị '1'. Hàng đợi ranh giới $D$ sẽ chỉ có phần tử [ G3 ];
8) B6. Thực hiện thủ tục dịch chuyển ngược với giá trị đường $e$ bằng '1', chúng ta nhận được giá trị đường $p = q$ = '0' và điều này suy ra đường $v$ nhận giá trị '0';
9) B7. Thực hiện phép suy diễn đối với giá trị $v$ = '0' sẽ cho ta giá trị $s$ = '1', $n$ = '0', $e$ = '1'. Như vậy chúng ta không nhận được giá trị

đầu ra đúng ( tức là ta phải nhận được $n$ = '1', $e = D$ ). Chúng ta nghịch đảo giá trị của $v$ $v$ = '1'. Với giá trị này, đường tín hiệu $e = \bar{D}$, như vậy chúng ta cũng không nhận được giá trị cần thiết $e$ = '1'. Kết luận rằng chúng ta không thể thực hiện được phép truyền $D$ qua phần tử **G6** tới đầu ra, do đó phải quay lại bước B4;

10) B4. Thực hiện phép truyền $D$ qua phần tử **G3**, chúng ta có giá trị trên đường $e = D$ ( vì giá trị đường $v$ đã được nghịch đảo lại bằng '1' ), đường $p$ = '1'. Hàng đợi ranh giới $D$ bao gồm phần tử [ G6 ];
11) B5. Thực hiện phép suy diễn với những giá trị vừa nhận được, ta có $q = r$ = '1', $s$ = '0', $u$ = '1';
12) B9. Do không nhận được giá trị D ở đầu ra, chúng ta quay lại bước B4;
13) B4. Thực hiện thao tác truyền $D$ qua phần tử **G6** chúng ta nhận được $e = D$, các bước từ bước B5 đến bước B8 được loại bỏ;
14) B9. Do giá trị $D$ được truyền tới đầu ra, bộ giá trị thử nghiệm nhận được sẽ là: $x = y = u = v$ = '1'.

Như vậy chúng ta thấy rằng thuật toán **PODEM** khảo sát tất cả các tổ hợp có thể có của đầu vào cho nên có thể áp dụng thuật toán này để tổng hợp các bộ giá trị thử nghiệm đối với mọi lỗi hằng số có thể phát hiện được. Bằng cách áp dụng các luật lựa chọn khác nhau, chúng ta có thể sửa đổi thuật toán **PODEM** để có thể nhận được các thuật toán hiệu quả hơn. Cũng giống như thuật toán **D**, thuật toán **PODEM** có thể áp dụng để tổng hợp các bộ giá trị thử nghiệm phát hiện các lỗi bắc cầu.

## §8.4. Phương pháp mô hình hóa lỗi

### 1. Bài toán mô hình hoá lỗi

Việc tạo các vectơ giá trị thử nghiệm để tìm lỗi bằng các phương pháp thuật toán cho phép ta thực hiện đối với mọi lỗi có thể phát hiện được. Như vậy có thể nói rằng phương pháp thuật toán là phương pháp tạo vectơ giá trị

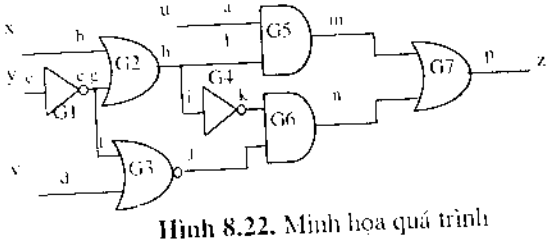
thử nghiệm tìm lỗi đầy đủ. Khi độ phức tạp của mạch tăng, thời gian để tạo bộ giá trị phát hiện lỗi sẽ tăng nhanh. Theo lý thuyết, thời gian tạo bộ giá trị thử nghiệm sẽ tăng theo luật hàm mũ đối với số lượng phần tử lôgic tham gia vào mạch. Đối với những mạch trong thực tế, thời gian tạo bộ giá trị thử nghiệm sẽ tỷ lệ bậc hai - bậc ba so với sự tăng lên của số lượng các phần tử lôgic cơ bản trong mạch.

Để giải quyết vấn đề về độ phức tạp của thuật toán, trong kỹ thuật chúng ta còn có thể sử dụng các phương pháp gọi là mô hình hóa lỗi. Phương pháp mô hình hoá lỗi cho phép trong một thời gian ngắn phát hiện được lỗi với một tỷ lệ khá cao.

Phương pháp mô hình hóa lỗi có thể được trình bày ngắn gọn như sau: chúng ta thực hiện mô hình hóa lôgic mạch chứa lỗi tương ứng với một vectơ giá trị đầu vào nào đó. Nếu nhận được các giá trị đầu ra của mạch trong trường hợp xuất hiện lỗi và trong trường hợp không bị lỗi khác nhau, như vậy vectơ giá trị đầu vào được coi là vectơ giá trị thử nghiệm phát hiện lỗi đang xét. Nói cách khác, phương pháp mô hình hóa lỗi là phương pháp xác định các lỗi có thể được phát hiện sử dụng một vectơ giá trị đầu vào cho trước.

Chúng ta thấy, phương pháp mô hình hoá lỗi về cơ bản dựa trên quá trình mô hình hóa lôgic mạch trong hai trường hợp: chứa lỗi và không chứa lỗi. Tuy nhiên đối với phiên bản bị lỗi của mạch, chúng ta có thể sử dụng những thủ pháp khác nhau để thực hiện quá trình mô hình hoá lôgic. Trong mục này chúng ta sẽ nghiên cứu những phương pháp chính để thực hiện mô hình hoá lỗi: phương pháp mô hình hóa lỗi song song, phương pháp mô hình hóa lỗi suy diễn và phương pháp mô hình hóa lỗi cạnh tranh.

## 2. Phương pháp mô hình hoá lỗi song song

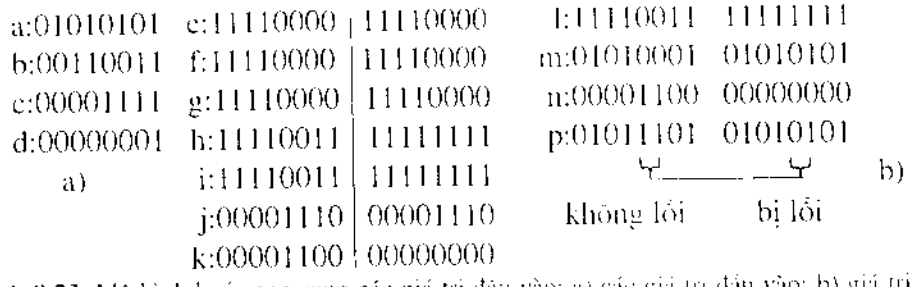



**Hình 8.22.** Minh họa quá trình mô hình hoá lỗi song song.

Cũng tương tự như quá trình mô hình hóa lôgic, khi thực hiện mô hình hóa lỗi, giá trị trên các đường tín hiệu được mô tả bằng một từ máy. Do đó nếu một từ máy có chứa $n$ hàng chữ số nhị phân, thì chúng ta có thể thực hiện được quá trình

mô hình hóa đối với $n$ vectơ giá trị đầu vào một cách song song.

Ta hãy xét ví dụ mạch trên hình 8.22. Đối với mạch này, với một từ máy gồm 16-bit, ta có thể thực hiện quá trình mô hình hóa song song đối với tất cả các vectơ giá trị đầu vào. Trong trường hợp này, các lỗi có thể được biểu diễn bằng cách thiết lập cố định các giá trị '0', '1' trên các đường tín hiệu bị lỗi, không phụ thuộc vào giá trị đầu vào. Đối với mạch trên hình 8.22, chúng ta thực hiện quá trình mô hình hóa lỗi với các vectơ giá trị đầu vào được biểu diễn trên các từ máy 8-bit; các giá trị trên các đường tín hiệu và trên đầu ra được biểu diễn trên hình 8.23. Quá trình mô hình hóa đối với lỗi hằng số *s-a-1* trên đường *h* *h/1* cho chúng ta thấy: tín hiệu đầu ra của mạch là $p$ = "01010101". Trong khi đó đầu ra trong trường hợp mạch không lỗi là: $p$ = "01011101". Như vậy bit thứ tư tính từ bên phải đã bị thay đổi do lỗi *h/1*. Từ đó suy ra bộ giá trị ( *a, b, c, d* ) = ( 0, 0, 1, 0 ) là bộ giá trị thử nghiệm phát hiện lỗi *h/1*. Một cách tương tự, ta có thể thực hiện quá trình mô hình hoá đối với mọi lỗi có thể phát hiện được khác. Nếu giá trị đầu ra trong trường hợp mạch chứa lỗi khác với đầu ra của mạch trong trường hợp mạch không bị lỗi thì chúng ta có thể kết luận rằng bộ giá trị đầu vào tương ứng là bộ giá trị thử nghiệm phát hiện lỗi.




**Hình 8.23.** Mô hình hoá song song các giá trị đầu vào: a) các giá trị đầu vào; b) giá trị trên các đường tín hiệu trong trường hợp mạch không lỗi và có chứa lỗi.

Khi mô hình hóa theo phương pháp trên, chúng ta thấy các giá trị đầu vào tham gia vào quá trình mô hình hoá một cách đồng thời trên tất cả các bit của từ máy do đó phương pháp này gọi là *mô hình hóa lỗi với đầu vào song song*. Khác với cách làm trên, chúng ta có thể thực hiện quá trình mô hình hóa trên toàn bộ các lỗi chỉ sử dụng một bộ giá trị đầu vào. Phương pháp này gọi là mô hình hóa lỗi song song.

Chúng ta hãy xét ví dụ mô hình hoá lỗi song song với sơ đồ mạch trên hình 8.22. Tương ứng với phương pháp mô hình hoá lỗi song song, chúng ta thực hiện quá trình mô hình hóa một cách đồng thời với các lỗi:

g/0, g/1, h/0, j/0, j/1, k/0, k/1, m/1;

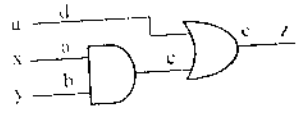
tương ứng với bộ giá trị đầu vào: ( a, b, c, d ) = ( 1, 0, 1, 0 ). Trên hình 8.24, chúng ta thấy bộ giá trị đầu vào như nhau ( cùng bằng "1010" ) đối với tất cả các lỗi nêu trên. Các giá trị được sắp xếp trong từ máy tương ứng với các lỗi theo trình tự nêu trên. Sau khi thực hiện quá trình mô hình hoá lôgic với tất cả các lỗi đã cho, chúng ta nhận được từ máy biểu diễn các giá trị đầu ra $p$ = "11101011". Ta nhận thấy rằng, trong trường hợp nếu xuất hiện các lỗi

| | | | |
|---|---|---|---|
| a:11111111 | e:00000000 | i:01000000 | m:01000001 |
| b:00000000 | f:00000000 | j:11101111 | n:10101011 |
| c:11111111 | g:01000000 | k:10111011 | p:11101011 |
| d:00000000 | h:01000000 | l:01000000 | |

**Hình 8.24.** Mô hình hoá lỗi song song.

hằng số *j/0* hoặc *k/0* giá trị tín hiệu ở đầu ra $p$ khác với giá trị đầu ra của mạch trong trường hợp không có lỗi ( trong trường hợp không có lỗi, đầu ra $p$ nhận giá trị bằng '1' tương ứng với vectơ giá trị đầu vào "1010", còn trong trường hợp xuất hiện lỗi *j/0* hoặc *k/0*, đầu ra $p$ nhận giá trị bằng '0' ). Từ đó suy ra vectơ giá trị đầu vào ( 1, 0, 1, 0 ) là vectơ giá trị kiểm nghiệm để phát hiện các lỗi hằng số *j/0*, *k/0*.

Trong những điều kiện thông thường, việc tạo ra các vectơ giá trị kiểm nghiệm đối với một số lượng lớn lỗi sử dụng phương pháp mô hình hóa lỗi song song và sau đó áp dụng phương pháp mô hình hoá lỗi với đầu vào song song cho những lỗi chưa được phát hiện còn lại là một thuật toán tạo các vectơ giá trị kiểm nghiệm để phát hiện lỗi có hiệu quả cao.




**Hình 8.25.** Ví dụ mạch đơn giản.

Khi thực hiện phương pháp mô hình hóa lỗi cũng như khi thực hiện quá trình mô hình hóa lôgic ta có thể sử dụng phương pháp biên dịch. Ví dụ, trong mạch trên hình 8.25, đối với phần tử AND chúng ta thấy, biểu thức lôgic $P_a$ cho đường tín hiệu $a$ có dạng:

$$P_a = xa_n + a_l;$$

trong đó:

$a_n = 1$ trong trường hợp giá trị đúng trên đường $a$;

$a_n = 0$ trong trường hợp ngược lại;

$a_l = 1$ khi thiết lập giá trị '1' trên đường $a$;

$a_l = 0$ trong những trường hợp khác.

Như vậy, khi thiết lập giá trị '0' trên đường tín hiệu $a$ ta có

$$a_1 = a_n = 0 \Rightarrow P_a = 0.$$

Tương tự như vậy, trên đường tín hiệu *b*:

$$P_b = yb_n + b_1;$$

Chúng ta thấy rằng giá trị trên đường tín hiệu *c* bằng $P_aP_b$, từ đó suy ra giá trị trên đường tín hiệu *c* có thể viết dưới dạng biểu thức lôgic sau:

$$P_c = ( P_aP_b )c_n + c_1;$$

Đối với phần tử OR, biểu thức lôgic sẽ có dạng:

$$P_c = ( P_c + P_d )c_n + c_1;$$

$$P_d = ud_n + 1;$$

Quá trình mô hình hoá nêu trên được thực hiện bằng cách xây dựng những biểu thức lôgic đối với những trạng thái lỗi của các phần tử lôgic và tính các giá trị của chúng. Khi tính các biểu lôgic bằng những phép toán trên bit với các từ có độ dài *n* của giá trị các biến, ta có thể thực hiện mô hình hoá song song.

Ví dụ, đối mạch trên hình 8.25, để thực hiện mô hình hoá đầu vào song song đối với lỗi hằng số *c/0*, chúng ta cần đưa ra các giá trị của trạng thái lỗi và trên cơ sở đó xác định biểu thức lôgic của mạch ( hình 8.26 ). Trong trường hợp mô hình hoá lỗi song song ta cần phải đưa ra giá trị của trạng thái lỗi ( hình 8.27 ). Trong ví dụ đối với mạch trên hình 8.25, chúng ta thực hiện quá trình mô hình hóa lỗi song song cho các lỗi: *a/0*, *a/1*, *b/0*, *b/1*, *c/0*, *c/1*, *d/0*, *d/1* đối với vectơ giá trị đầu vào: $x = y =$ '1', $u =$ '0'.

Mô hình hóa đầu vào song song:

| | | |
|---|---|---|
| u 01010101 | a 11111111 | 00000000 |
| x 00110011 | b 11111111 | 00000000 |
| y 00001111 | c 00000000 | 00000000 |
| | d 11111111 | 00000000 |
| | e 11111111 | 00000000 |

**Hình 8.26.** Mô hình hoá lỗi với các đầu vào song song cho mạch trên hình 8.25.

| | | |
|---|---|---|
| u 00000000 | a 00111111 | 01000000 |
| x 11111111 | b 11001111 | 00010000 |
| y 11111111 | c 11110011 | 00000100 |
| | d 11111100 | 00000001 |
| | e 11111111 | 00000000 |

**Hình 8.27.** Mô hình hoá lỗi song song cho mạch trên hình 8.25 với vectơ đầu vào ( *u*, *x*, *y* ) = ( *0*, *1*, *1* ) cho các lỗi *a/0*, *a/1*, *b/0*, *b/1*, *c/0*, *c/1*, *d/0*, *d/1*.

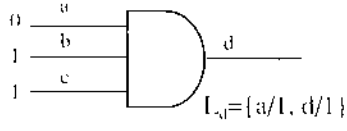
Khi sử dụng phương pháp biên dịch trong quá trình mô hình hoá lỗi, chúng ta cần phân hạng các phần tử lôgic và thực hiện theo trình tự của các biểu thức lôgic.

### 3. Mô hình hóa lỗi suy diễn

Trong phương pháp mô hình hóa lỗi song song đã nêu trên, đầu tiên chúng ta đưa ra các vectơ giá trị đầu vào, các lỗi và sau đó kiểm tra khả năng phát hiện các lỗi đưa ra bằng các vectơ giá trị đầu vào đang xét. Trong trường hợp thực hiện kiểm tra thành công, vectơ giá trị đầu vào đang xét được coi là vectơ giá trị kiểm tra phát hiện lỗi đã cho; nếu thực hiện mô hình hoá lỗi không thành công, chúng ta phải thực hiện quá trình mô hình hoá lỗi cho bộ giá trị đầu vào và lỗi khác. Như vậy, trong phương pháp mô hình hoá lỗi nói trên, mối quan hệ lôgic giữa các lỗi và các vectơ giá trị đầu vào không được thiết lập và do đó hiệu quả của quá trình tạo các bộ giá trị kiểm nghiệm phát hiện lỗi không cao.

Để tăng hiệu quả của thuật toán mô hình hóa, chúng ta đưa ra phương pháp, trong đó tương ứng với một vectơ giá trị đầu vào, phải phát hiện tất cả các lỗi có thể được phát hiện bằng bộ giá trị này. Để thực hiện được điều đó, chúng ta phải sử dụng các phép toán suy diễn tập hợp để thực hiện quá trình mô hình hóa và do đó phương pháp này được gọi là phương pháp mô hình hóa lỗi suy diễn.

Để đơn giản và có thể thấy rõ được cơ chế suy diễn, chúng ta xét ví dụ phần tử AND có ba đầu vào. Nếu các đầu vào nhận giá trị $a$ = '0', $b$ = '1', $c$ = '1', đầu ra $d$ sẽ nhận giá trị $d$ = '0'. Từ đó suy ra các lỗi hằng số do vectơ giá trị đầu vào ( $a$, $b$, $c$ ) = ( 0, 1, 1 ) phát hiện được sẽ là: $a$/1, $d$/1 ( chúng ta sẽ viết gọn ký hiệu lỗi hằng số $s$-$a$-$v$ trên đường tín hiệu $s$ là $s$/$v$ thành $s_v$ ). Nếu các đầu vào là $a$ = '1', $b$ = '1', $c$ = '1', ta có $d$ = '1'. Từ đó suy ra, tập hợp các lỗi hằng số có thể phát hiện được bằng vectơ giá trị đầu vào ( $a$, $b$, $c$ ) = ( 1, 1, 1 ) sẽ là { $a_0$, $b_0$, $c_0$, $d_0$ }. Chúng ta có thể nhận được tập hợp này theo những luật sau:




**Hình 8.28.** Ví dụ mô hình hoá lỗi suy diễn với phần tử AND có ba đầu vào.

**Luật suy diễn C:** các phép toán tập hợp đối với phần tử **A** và các giá trị đầu vào của phần tử đó:

- Đường tín hiệu vào $a$ của phần tử là đường vào từ bên ngoài. Nếu giá trị đường tín hiệu này bằng '1', ta có $\mathbf{L}_a = \{ a_0 \}$; nếu giá trị đường tín hiệu bằng '0', thì $\mathbf{L}_a = \{ a_1 \}$;
- Hàm ra của phần tử **A** được biểu diễn dưới dạng tổng hoặc tích. Nếu giá trị đúng của đường tín hiệu $a$ bằng '0', thì ký hiệu $a$ trong biểu thức sẽ được thay bằng $\mathbf{L}_a$, ký hiệu $\bar{a}$ được thay bằng $\bar{\mathbf{L}}_a$. Nếu giá trị đúng của đường tín hiệu $a$ bằng '1' thì thì ký hiệu $a$ trong biểu thức sẽ được thay bằng $\bar{\mathbf{L}}_a$ và $\bar{a}$ được thay bằng $\bar{\mathbf{L}}_a$.
- Tích lôgic (.) và tổng lôgic (+) trong biểu thức được thay bằng các phép toán tập hợp $\cap$ và $\cup$. Nếu giá trị đúng của đầu ra bằng '0' thì biểu thức được thay bằng $\mathbf{L}$, nếu là '1' - $\bar{\mathbf{L}}$.

  Trong trường hợp khi giá trị đúng của đường tín hiệu ra $f$ bằng '0' thì ta hợp tập hợp $\{ f_1 \}$ vào $\mathbf{L}$; nếu giá trị đầu ra bằng '1', ta sẽ hợp tập hợp $\{ f_0 \}$ vào $\mathbf{L}$. Kết quả ta nhận được tập hợp $\mathbf{L}_f$.

Các phần tử trong tập hợp $\mathbf{L}_f$ là các lỗi hằng số có thể được phát hiện bằng vectơ giá trị đầu vào đang xét. Ví dụ, đối với phần tử AND trên hình 8.28, ta có $d = abc$, với $a$ = '0', $b$ = '1', $c$ = '1', từ đó suy ra:

$$\mathbf{L}_d = \left(\mathbf{L}_a \cap \bar{\mathbf{L}}_b \cap \bar{\mathbf{L}}_c\right) \cup \{d_1\};$$

trong đó $\mathbf{L}_a = \{ a_1 \}$, $\mathbf{L}_b = \{ b_0 \}$, $\mathbf{L}_c = \{ c_0 \}$; sau một số phép biến đổi, nhận được:

$$\mathbf{L}_d = \{ a_1, d_1 \}.$$

Nếu các đầu vào nhận giá trị: $a$ = '1', $b$ = '1', $c$ = '1', ta sẽ có:

$$\bar{\mathbf{L}} = (\mathbf{L}_a \cap \mathbf{L}_b \cap \bar{\mathbf{L}}_c) \Rightarrow \mathbf{L}_d = \overline{(\bar{\mathbf{L}}_a \cap \bar{\mathbf{L}}_b \cap \bar{\bar{\mathbf{L}}}_c)} \cup \{d_0\} = (\mathbf{L}_a \cup \mathbf{L}_b \cup \mathbf{L}_c) \cup \{d_0\}$$

trong đó $\mathbf{L}_a = \{ a_0 \}$, $\mathbf{L}_b = \{ b_0 \}$, $\mathbf{L}_c = \{ c_0 \}$. Như vậy ta nhận được

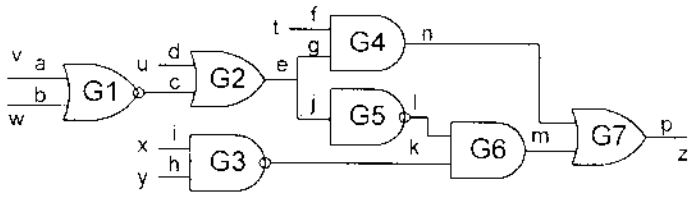
$$\mathbf{L}_d = \{ a_0, b_0, c_0, d_0 \}.$$

Trong những trường hợp khi đầu vào $a$ của phần tử không phải là đầu vào từ bên ngoài, $\mathbf{L}_a$ là tập hợp các lỗi được phát hiện trước mức đang xét. Do đó, khi đi từ các đầu vào tính từ bên ngoài tuần tự theo mức của sơ đồ lôgic và áp dụng nhiều lần luật đã nêu trên, ta có thể xác định được tập hợp lỗi được phát hiện bằng bộ giá trị đầu vào đã cho.

Thuật toán nêu trên được gọi là thuật toán mô hình hoá lỗi suy diễn. Chúng ta hãy xét ví dụ áp dụng phương pháp mô hình hoá lỗi suy diễn đối

với mạch trên hình 8.29. Bài toán đặt ra là xác định các lỗi có thể được phát hiện với vectơ giá trị đầu vào:

$$x = y = t = u = v = w = 1$$




**Hình 8.29.** Ví dụ minh hoạ phương pháp mô hình hoá lỗi suy diễn.

Áp dụng một cách tuần tự các luật bắt đầu từ đầu vào, chúng ta nhận được các tập hợp sau:

$\mathbf{L}_a = \{ a_0 \}$;
$\mathbf{L}_b = \{ b_0 \}$;
$\mathbf{L}_d = \{ d_0 \}$;
$\mathbf{L}_f = \{ f_0 \}$;

$\mathbf{L}_h = \{ h_0 \}$, $\mathbf{L}_i = \{ i_0 \}$;

$\mathbf{L}_c = ( \mathbf{L}_a \cap \mathbf{L}_b ) \cup \{ c_1 \} = \{ c_1 \}$;

$\mathbf{L}_e = (\mathbf{L}_d \cap \bar{\mathbf{L}}_c) \cup \{e_0\} = \{d_0, e_0\}$;

$\mathbf{L}_g = \mathbf{L}_e \cup \{ g_0 \} = \{ d_0, e_0, g_0 \}$;

$\mathbf{L}_j = \mathbf{L}_e \cup \{ j_0 \} = \{ d_0, e_0, j_0 \}$;

$\mathbf{L}_k = ( \mathbf{L}_i \cup \mathbf{L}_h ) \cup \{ k_1 \} = \{ i_0, h_0, k_1 \}$;

$\mathbf{L}_l = \mathbf{L}_j \cup \{ l_1 \} = \{ d_0, e_0, j_0, l_1 \}$;

$\mathbf{L}_m = ( \mathbf{L}_k \cap \mathbf{L}_l ) \cup \{ m_1 \} = \{ m_1 \}$;

$\mathbf{L}_n = ( \mathbf{L}_g \cup \mathbf{L}_f ) \cup \{ n_0 \} = \{ d_0, e_0, g_0, f_0, n_0 \}$;

$\mathbf{L}_p = (\mathbf{L}_n \cap \bar{\mathbf{L}}_m) \cup \{p_0\} = \{ d_0, e_0, g_0, f_0, n_0, p_0 \}$.

Như vậy vectơ giá trị đầu vào $x = y = t = u = v = w =$ '1' cho phép phát hiện mọi lỗi nằm trong tập hợp $\mathbf{L}_p = \{ d_0, e_0, g_0, f_0, n_0, p_0 \}$.

Nếu giá trị đầu vào $y$ thay đổi thành '0', thì sẽ có những thay đổi sau trong dãy suy diễn nói trên:

$y = 0 \Rightarrow h = 0, k = 1$;

$h = 0 \Rightarrow \mathbf{L}_h = \{ h_1 \}$;

$k = 1 \Rightarrow \mathbf{L}_k = (\mathbf{L}_h \cap \bar{\mathbf{L}}_i) \cup \{k_0\} = \{h_1, k_0\}$;

Vì $m = 1 \Rightarrow \mathbf{L}_m = ( \mathbf{L}_k \cap \mathbf{L}_l ) \cup \{ m_0 \} = \{ c_1, d_0, e_1, h_1, j_1, l_0, k_0, m_0 \}$;

Vì $p = 1 \Rightarrow \mathbf{L}_p = (\mathbf{L}_m \cap \bar{\mathbf{L}}_n) \cup \{p_0\} = \{ h_1, j_1, k_0, l_0, m_0, p_0 \}$;

Từ ví dụ trên chúng ta thấy rằng khi sử dụng phương pháp mô hình hoá lỗi suy diễn, chúng ta luôn sử dụng các phép toán tập hợp. Điều đó dẫn đến

việc cần thiết xây dựng phương pháp thực hiện có hiệu quả các phép toán tập hợp trên máy tính. Chúng ta thấy rằng các phép toán tập hợp phụ thuộc vào các giá trị đầu vào nên việc áp dụng các phép xử lý song song không thể thực hiện được trong trường hợp này.

## 4. Mô hình hóa lỗi cạnh tranh

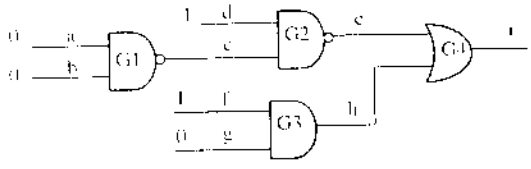
Trong phương pháp mô hình hoá suy diễn đã trình bày ở trên, danh sách các lỗi có thể phát hiện bằng một vectơ giá trị đầu vào cho trước được tìm thấy bằng cách lan truyền tuần tự lỗi theo các mức phân hạng. Trong phương pháp này ta cần phải thực hiện các phép toán tập hợp khó thực hiện trên máy tính.

Tương tự với tư tưởng truyền danh sách lỗi, chúng ta dùng phương pháp xây dựng danh sách bằng cách mô hình hóa thay cho các phép toán tập hợp. Phương pháp này gọi là phương pháp mô hình hóa lỗi cạnh tranh.

Chúng ta xét ví dụ trên hình 8.30 với các đầu vào của mạch là:

$$a = b = 0,\ c = 1,\ f = 1,\ g = 0.$$

Với $a = b =$ '0' ta có giá trị đúng tại $e =$ '1'. Những lỗi có thể được phát hiện nhờ những giá trị đầu vào và đầu ra đó nhận các giá trị đảo: $a/1$, $b/1$, $e/0$ , ta có thể viết gọn là các lỗi $a_1$, $b_1$, $e_0$. Các lỗi $a_0$, $b_0$ trong trường hợp này không thể phát hiện được bởi vì chúng làm cho đầu ra có giá trị trùng với giá trị thực.




$a = b = 0,\ d = 1,\ f = 1,\ g = 0$

**Hình 8.30.** Ví dụ mô hình hoá lỗi sử dụng phương pháp mô hình hoá lỗi cạnh tranh.

- Đối với phần tử **G2**, nếu giá trị trên đường $d$ bằng '1' suy ra giá trị đúng tại đường $e$ bằng '0'. Như vậy các lỗi có thể được phát hiện là: $c_0$, $d_0$, $e_1$. Vì các lỗi này thiết lập giá trị đường $e$ bằng '1', cho nên khả năng phát hiện các lỗi đó bằng bộ giá trị đầu vào đang xét được đảm bảo. Từ đó suy ra, với các giá trị đầu vào $a = b =$ '0', $d =$ '1' thì các lỗi $\mathbf{L}_e = \{ c_0, d_0, e_1 \}$ có thể được phát hiện.

- Nếu giá trị trên đường tín hiệu $f =$ '1', $g =$ '0', ta sẽ có giá trị đúng trên đầu ra $h$ của phần tử **G3** là '0', do đó các lỗi $\{ f_0, g_0, h_1 \}$ có thể được

phát hiện. Mặt khác, khi trong mạch có lỗi $f_0$, đầu ra $h$ = '0' như vậy, danh sách lỗi cuối cùng sẽ được phát hiện là: $\mathbf{L_h} = \{ g_1, h_1 \}$;

Đối với phần tử **G4**, giá trị đúng trên các đường $e$, $h$ bằng '0' do đó giá trị đúng trên đường $i$ cũng bằng '0'. Như vậy các lỗi có thể phát hiện được sẽ là $h_1$, $e_1$. Mặt khác sự xuất hiện của lỗi $e_1$ được sinh ra từ danh sách $\mathbf{L_e}$, còn lỗi $h_1$ nằm trong tập hợp $\mathbf{L_h}$. Kết luận, tập hợp các lỗi sẽ được phát hiện trên đầu ra $z$ là:

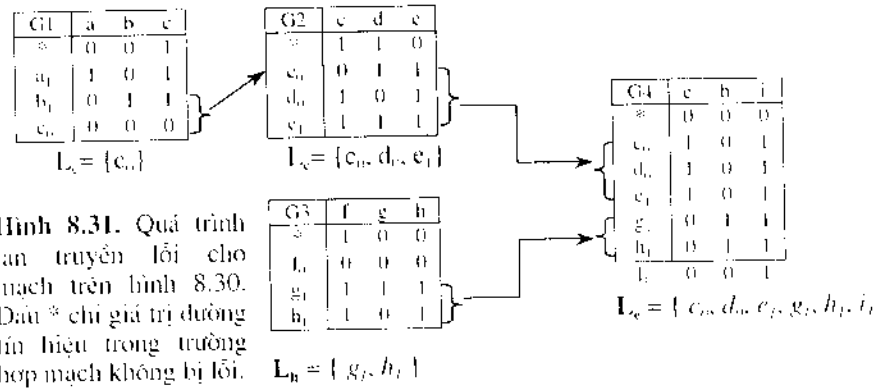
$$\mathbf{L_i} = \{ c_0, d_0, e_1, g_1, h_1, i_1 \}.$$

Trên hình 8.31 là sơ đồ trình tự truyền và nhận các lỗi có thể được phát hiện qua các phần tử lôgic trong mạch trên hình 8.30.

Dựa vào các suy luận nêu trên, chúng ta có thể đưa ra các quy tắc sau:

**Luật D**:

1. Đối với phần tử lôgic **G1** có tất cả các đầu vào là từ bên ngoài: chúng ta ký hiệu các đầu vào là $a$,$b$, đầu ra là $c$; giá trị tại các đầu vào ký hiệu là $\alpha$, $\beta$ và đầu ra tương ứng là $\gamma$. Trong danh sách các lỗi sẽ phát hiện: $\{ a_\alpha, b_\beta, c_{\bar\gamma} \}$, các lỗi được coi là được phát hiện nếu chúng thiết lập trên đầu ra giá trị $\bar\gamma$. Những lỗi này gọi là lỗi lan truyền.

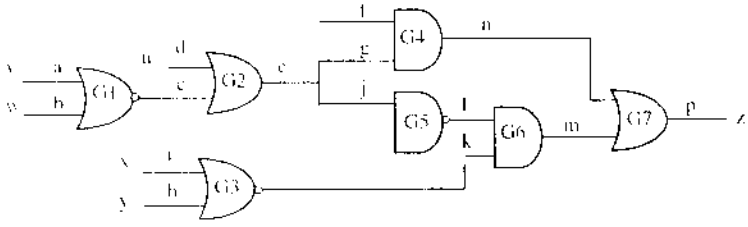



**Hình 8.31.** Quá trình lan truyền lỗi cho mạch trên hình 8.30. Dấu * chỉ giá trị đúng tín hiệu trong trường hợp mạch không bị lỗi.

2. Đối với phần tử lôgic $\mathbf{G_X}$ có đầu vào từ bên ngoài và đầu vào từ bên trong tương ứng là $a$ và $b$, đầu ra là $c$. Giá trị trên đường vào $a$ là $\alpha$ , trên đường truyền $b$ là $\beta$, giá trị đúng trên đầu ra $c$ là $\gamma$. Từ danh sách các lỗi có thể phát hiện $\{ a_\alpha, L_b(\beta), c_{\bar\gamma} \}$, những lỗi chính thức được tìm thấy sẽ là

những lỗi thiết lập trên đầu ra $c$ giá trị $\bar{\gamma}$. Những lỗi này tạo thành danh sách lỗi lan truyền $\mathbf{L}_c(\gamma)$.

3. Đối với phần tử logic $\mathbf{G}_{\mathrm{XI}}$ chỉ có các đầu vào bên trong là $a$, $b$ và đầu ra $c$, các đường tín hiệu $a$, $b$ được đặt giá trị đúng $\alpha$ và $\beta$, đường c sẽ nhận giá trị đúng là $\gamma$. Từ danh sách $\{ \mathbf{L}_a(\alpha), \mathbf{L}_b(\beta), c_{\bar{\gamma}} \}$ các lỗi sẽ phát hiện




**Hình 8.32.** Ví dụ thực hiện phương pháp mô hình hoá lỗi cạnh tranh.

được là các lỗi đặt giá trị trên đường $c$ bằng $\bar{\gamma}$. Những lỗi này tạo thành danh sách lỗi lan truyền qua phần tử $\mathbf{G}_{\mathrm{XI}}$ $\mathbf{L}_c(\gamma)$.

4. Các lỗi lan truyền $\mathbf{L}_c(\gamma)$ tìm thấy trên đầu ra bên ngoài của mạch $c$ tạo thành tập hợp các lỗi phát hiện được nhờ bộ giá trị đầu vào đã cho.

Chúng ta xét ví dụ đối với mạch trên hình 8.32, sau khi áp dụng **luật D**, tập hợp các lỗi nhận được sẽ là:

$$\mathbf{L}_{\mathbf{p}} = \{ d_0, e_0, g_0, f_0, w_0, p_0 \}$$

Cũng tương tự như phương pháp mô hình hoá lỗi suy diễn, việc thực hiện xử lý song song để có thể mô hình hoá đồng thời trên các vectơ đầu vào tương đối phức tạp.

Như vậy, chúng ta đã nghiên cứu những phương pháp mô hình hoá lỗi. Đối tượng áp dụng của những phương pháp này là phát hiện các lỗi hằng số cho các mạch tổ hợp. Chúng ta đã thấy trong chương 5, khi thực hiện mô hình hoá lôgic cho mạch, chúng ta có thể tính đến độ trễ của các phần tử cũng như hình dạng của tín hiệu. Do đó chúng ta cũng có thể xét đến những mô hình lỗi tương ứng. Nói chung, quá trình mô hình hoá lỗi nếu được tính tới các thông số về thời gian xử lý sẽ phức tạp hơn rất nhiều.

## §8.5. Một số phương pháp làm đơn giản hóa quá trình kiểm tra phát hiện lỗi

Với sự phát triển của công nghệ VLSI, kích thước và độ phức tạp của các mạch số tăng nhanh. Khi số lượng các phần tử mạch tăng, thời gian cần thiết để tạo các bộ giá trị thử nghiệm phát hiện lỗi tăng. Đối với những mạch phức tạp như các bộ vi xử lý, bài toán kiểm tra lỗi trở nên hết sức phức tạp. Để giải quyết vấn đề này, chúng ta không thể chỉ giới hạn ở việc tìm ra những phương pháp kiểm tra có hiệu quả mà còn phải tìm cách thiết kế những mạch cho phép làm đơn giản hoá bài toán phát hiện lỗi ngay từ đầu.

Để làm đơn giản hóa quá trình kiểm tra và phát hiện lỗi, chúng ta cần phải giải quyết hai bài toán:

- Bài toán thứ nhất là giảm thời gian tạo các vectơ giá trị thử nghiệm. Bài toán này được giải quyết bằng cách xây dựng những thuật toán tạo vectơ giá trị thử nghiệm hiệu quả hơn.
- Bài toán thứ hai là làm giảm số lượng các phép kiểm tra và nhằm mục đích là làm giảm thời gian kiểm tra mạch.

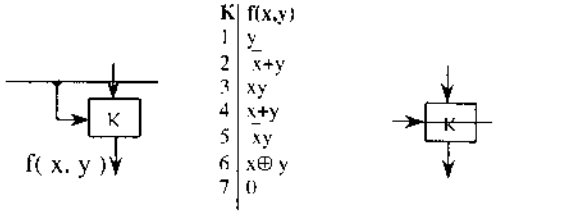
Thêm vào đó, khi thiết kế những mạch dễ kiểm tra theo các mục đích nêu trên lại nảy sinh ra các vấn đề mới:

Tăng số lượng thiết bị để làm đơn giản hóa quá trình kiểm tra mạch.

Giảm tốc độ làm việc do hệ quả của việc làm đơn giản hóa quá trình kiểm tra.

Như vậy, thiết kế những mạch dễ kiểm tra có thể coi là thiết kế mạch lôgic với khả năng thêm một số lượng tối thiểu thiết bị bổ sung vào mạch mà không làm thay đổi chức năng và các tham số của mạch ban đầu kết hợp với khả năng thực hiện kiểm tra mạch bằng một số lượng nhỏ các vectơ giá trị kiểm nghiệm nhận được từ những phương pháp đơn giản. Trong thực tế, để nhận được một phương pháp thoả mãn tất cả các yêu cầu nêu trên là không đơn giản, do đó, trong quá trình thiết kế những mạch dễ kiểm tra, nhà thiết kế phải đưa ra được cách giải quyết những vấn đề nói trên.

## 1. Các giới hạn về cấu trúc của sơ đồ




Mô tả chức năng phần tử — Các hàm chức năng của phần tử — Ký hiệu phần tử

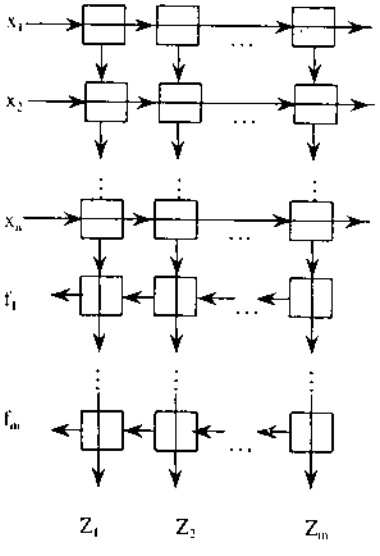
**Hình 8.33.** Mô tả cấu trúc và chức năng của phần tử cơ bản.

Trước tiên, chúng ta nghiên cứu những phương pháp thiết kế những mạch dễ kiểm tra bằng dựa trên việc xây dựng những mạch lôgic có một dạng nhất định. Những mạch đó thường có cấu trúc dạng ma trận hai chiều. Vì cấu trúc của mạch có tính quy luật cao cho nên chúng ta có thể tổng hợp các vectơ giá trị kiểm nghiệm cho những mạch loại này một cách tương đối dễ dàng.

Chúng ta xét ví dụ một mạch kinh điển: ma trận các phần tử cơ bản với những giá trị được giản lược. Phần tử cơ bản của mạch là phần tử có hai đầu vào và một đầu ra. Phần tử này thực hiện một trong bảy hàm lôgic cơ bản như trong mô tả trên hình 8.33. Mạch tổng thể sẽ có cấu trúc được mô tả dưới dạng ma trận như trên hình 8.34. Các chức năng lôgic được thực hiện trên cơ sở kết hợp ma trận thuận và ma trận nghịch đảo. Nếu chọn hàm chức năng của phần tử trong ma trận thuận là $xy$ hoặc $\bar{x}y$, thì hàm chức năng của phần tử lôgic trong ma trận nghịch đảo sẽ có dạng $( x + y )$ và được thể hiện dưới dạng chuẩn tắc tuyển. Từ đó suy ra, trong trường hợp tổng quát, chúng ta có thể biểu diễn và xây dựng được mọi hàm lôgic.

Trong mục này chúng ta chỉ xét các lỗi hằng số trên các đường vào và đường ra của các phần tử. Quá trình phát hiện lỗi sẽ được thực hiện theo trình tự sau:

1. Xác định giá trị $X_j^l$ của tập hợp các biến đầu vào $( x_1, x_2, ..., x_n )$ sao cho trong mỗi cột, giá trị đầu ra $z_j$ trùng với $y_j$.

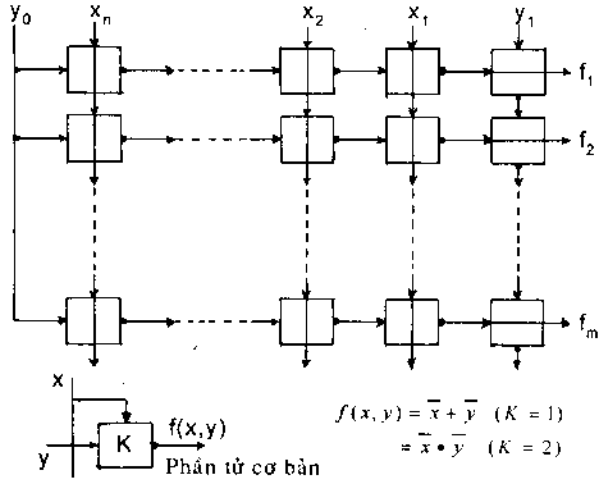



**Hình 8.34.** Ma trận các phần tử với các chức năng giản lược.

2. Nếu khi thay đổi giá trị $y_j$ đối với bộ giá trị đầu vào $X_i^l$ chúng ta nhận được sự thay đổi giá trị của $z_j$, như vậy có thể kết luận rằng không có lỗi trên các đầu vào và đầu ra của mạch. Điều này được thực hiện với mọi cột trong sơ đồ cấu trúc của mạch.
3. Đối với phần tử trên giao điểm của cột thứ $j$ và dòng thứ $i$, chúng ta lựa chọn các giá trị $x_1,..., x_{i-1}, x_{i+1}, ..., x_n$ và $y_j$ ( $X_i^l$, $y^j$ ) sao cho đường truyền từ đầu vào $x_i$ tới đầu ra $z_j$ được kích hoạt.
4. Nếu khi thay đổi giá trị $x_i$ đối với ( $X_i^l$, $y^j$ ) giá trị của $z_j$ cũng thay đổi theo, như vậy chúng ta có thể kết luận rằng phần tử thứ $i$ và thứ $j$ hoạt động đúng. Điều này sẽ được nghiệm đúng với mọi phần tử trong mạch ma trận.

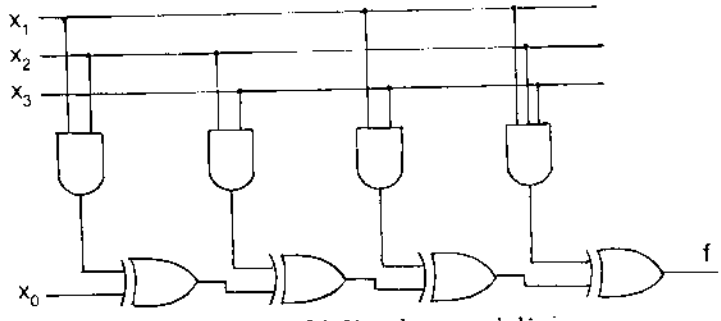
Theo các bước thực hiện nêu trên, chúng ta có thể dễ dàng kiểm tra hoạt động của toàn bộ mạch ma trận thuận. Để có thể kiểm tra hoạt động của ma trận nghịch đảo, chúng ta cũng có thể dễ dàng xây dựng các vectơ giá trị thử nghiệm bằng cách khởi tạo các đường truyền kích hoạt nhạy cảm với lỗi một cách tuần tự.

Chúng ta nhận thấy, cấu trúc của mạch cũng giữ nguyên khi xây dựng các hàm khác nhau, chỉ có các chức năng của các phần tử là thay đổi. Do đó, việc xây dựng các bộ giá trị thử nghiệm trong trường hợp này không phức tạp. Như vậy, mạch có thể được coi là dễ kiểm tra. Tuy nhiên, khi xây dựng mạch, chúng ta cần sử dụng số lượng phần tử cơ bản nhiều hơn so với quá trình thiết kế lôgic thông thường. Ngoài ra, đối với ma trận kích thước $n$ x $m$, số lượng các tầng bằng $2m + n$, như vậy tốc độ làm việc của mạch có thể bị giảm. Vì những lý do trên, việc thực hiện xây dựng các mạch lôgic theo phương pháp nêu trên gặp nhiều khó khăn trong thực tế. Số lượng các vectơ giá trị thử nghiệm có thể đạt tới $2m( n + 2 )$ đối với một hàm lôgic.




**Hình 8.35.** Ma trận NOR-NAND.

Trong ma trận nói trên, chúng ta sử dụng bảy hàm lôgic cơ bản, tuy nhiên, một hàm bất kỳ có thể được biểu diễn bằng hai hàm cơ bản: NAND hoặc NOR. Những ma trận đó được gọi là các ma trận dạng NAND-NOR. Khi xây dựng $k$ hàm $n$ biến bằng các ma trận dạng NAND-NOR, số lượng các vectơ giá trị thử nghiệm bằng $k(n + 2)$, và số lượng các tầng cực đại bằng $n + k$. Như vậy cúng ta thấy rằng vấn đề về số lượng kiểm tra chưa được giải quyết triệt để.




**Hình 8.36.** Xây dựng mạch lôgic từ những phần tử AND và XOR.

Mọi hàm lôgic có thể biểu diễn dưới dạng tổ hợp các phép toán XOR của các tích lôgic ( xem chương 2 ). Biểu diễn hàm dưới dạng như vậy cho phép thực hiện dưới dạng mạch hai tầng của các phép toán lôgic AND và XOR - dạng mạch AND-XOR. Ví dụ hàm lôgic:

$$f(x_1, x_2, x_3) = x_1x_2 \oplus x_2x_3 \oplus x_1x_3 \oplus x_1x_2x_3$$

sẽ được biểu diễn dưới dạng mạch hai tầng như trên hình 8.36. Các vectơ giá trị thử nghiệm cho những mạch loại này có cấu trúc khá đơn giản. Cấu trúc này bao gồm giá trị '0' đối với một biến và giá trị '1' đối với các biến còn lại.

| | $x_0$ | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|---|
| $T_A$ | 0 | 0 | 0 | 0 |
| | 0 | 1 | 1 | 1 |
| | 1 | 0 | 0 | 0 |
| | 1 | 1 | 1 | 1 |
| $T_X$ | d | 0 | 1 | 0 |
| | d | 1 | 0 | 1 |
| | d | 1 | 1 | 1 |

d - trong những điều kiện thông thường, nhận giá trị '0' hoặc '1'; $T_A$ - kiểm tra phần tử AND; $T_X$ - kiểm tra phần tử XOR.

**Hình 8.37.** Các bộ giá trị thử nghiệm T đối với mạch AND-XOR.

Nếu số lượng các tích của $n$ biến lôgic bằng $p$, chúng ta có thể thấy rằng số lượng các vectơ giá trị thử nghiệm sẽ bằng ( $2n + 4$ ) và số lượng cực đại các tầng mạch sẽ bằng $p$. Nếu so sánh với các phương pháp xây dựng các vectơ giá trị thử nghiệm thông thường, số lượng các vectơ giá trị thử nghiệm không lớn.

Một điểm khác biệt là số lượng các tầng vượt xa số lượng tầng nếu so sánh với xây dựng mạch bằng phương pháp thông thường.

## 2. Biến thể của sơ đồ mạch và các phần tử thêm mới

Nói chung, để đơn giản hóa bài toán phát hiện lỗi trong các mạch lôgic, chúng ta cần đơn giản hóa quá trình quan sát các phần bên trong của mạch và giảm nhẹ việc thiết lập các giá trị tuỳ ý tại các điểm bên trong của mạch. Nói theo cách khác, chúng ta cần tăng tính dễ quan sát và khả năng kiểm soát của mạch. Với mục đích đó, chúng ta đưa vào trong mạch những phần tử bổ sung và thực hiện việc biến đổi sơ đồ của mạch sao cho mạch không thay đổi về chức năng. Ở đây, một vấn đề quan trọng là chúng ta phải xác định được mục đích chính khi nghiên cứu xây dựng những mạch để kiểm tra.

Nếu xuất phát điểm là số lượng cực tiểu các bộ giá trị thử nghiệm, như vậy chúng ta có thể biến đổi mạch điện sao cho có thể phát hiện được lỗi chỉ cần ít nhất là ba bộ giá trị thử nghiệm. Số lượng này được đưa ra đựa vào nhận xét như sau: để có thể phát hiện lỗi của những phần tử AND và OR có hai đầu vào, chúng ta cần ít nhất ba bộ giá trị kiểm nghiệm, vì vậy có thể nói ba bộ giá trị thử nghiệm là cận dưới theo lý thuyết.

Để phát hiện lỗi cho mạch AND có hai đầu vào, chúng ta cần phải kiểm tra phần tử với ba bộ giá trị đầu vào: ( 0, 1 ), ( 1, 0 ), ( 1, 1 ). Do đó, phần tử AND có thể được kiểm tra bởi bất kỳ hai trong số các dãy giá trị đầu vào sau:

$$S_{\wedge} = \{ 011, 101, 110 \}$$

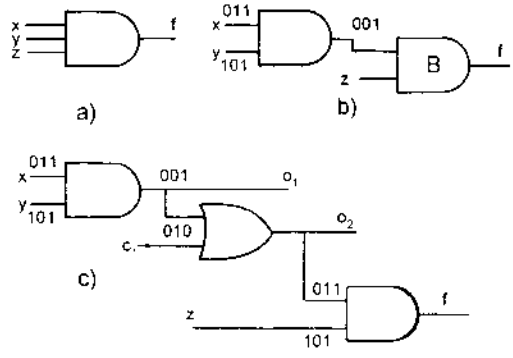
Trong trường hợp phần tử OR, việc kiểm tra có thể được bảo đảm bởi bất kỳ hai trong số các dãy giá trị đầu vào sau:

$$S_{\vee} = \{ 001, 010, 100 \}$$

Để có thể phát hiện được lỗi dùng các dãy giá trị đầu vào nêu trên thì mạch cần phải được biến đổi như sau:

1. Mạch phải được chuyển đổi thành mạch chỉ chứa các phần tử AND, OR, NOT.

2. Phần tử AND ( hoặc OR ) có số lượng đầu vào là ba hoặc hơn sẽ được khai triển thành tổ hợp liên kết tuần tự của các phần tử AND ( hoặc OR ) có hai đầu vào. Ví dụ, phần tử AND có ba đầu vào trên hình 8.38a sẽ được khai triển thành mạch gồm có các kết nối của các phần tử AND hai đầu vào mô tả trên hình 8.38b.




**Hình 8.38.** Quá trình phân rã mạch cho phép phát hiện lỗi chỉ với ba bộ giá trị đầu vào.

3. Bắt đầu từ các đầu vào, chúng ta chọn các bộ giá trị thử nghiệm từ trong các chuỗi $S_A$ và $S_O$ tương ứng với dạng của phần tử. Truyền các giá trị này tới đầu ra của mạch.

4. Nếu trong quá trình truyền giá trị, không nhận được các dãy tín hiệu từ $S_A$ hoặc $S_O$, chúng ta cần phải thêm vào đường tín hiệu phần tử AND hoặc OR có hai đầu vào.

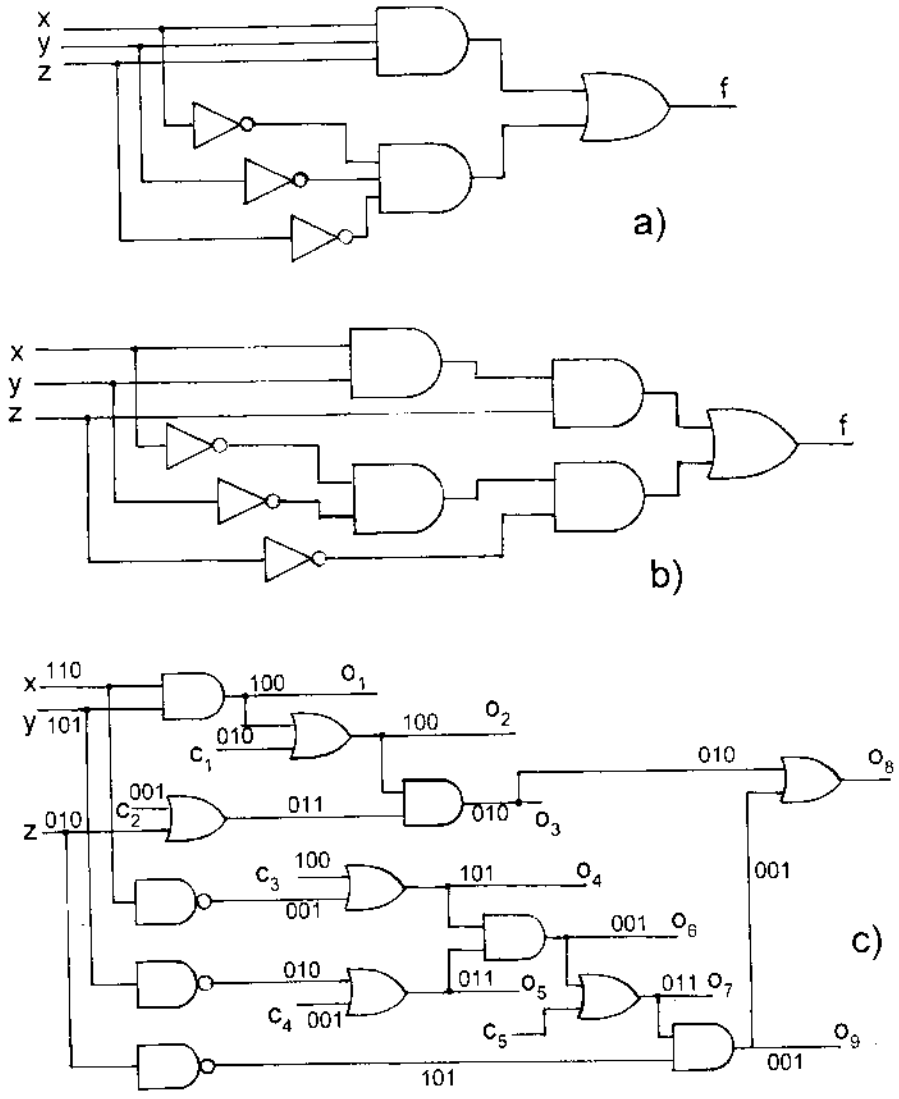
   Ví dụ, nếu dãy các giá trị được lựa chọn như trên hình 8.38b, khi đó tại đầu vào của phần tử AND **B** sẽ xuất hiện dãy tín hiệu "001". Nếu thêm phần tử OR như trên hình 8.38c và lựa chọn bộ giá trị tương ứng, quá trình phát hiện lỗi tại tất cả các phần tử trong mạch có thể được thực hiện chỉ bằng ba bộ giá trị thử nghiệm.

5. Lặp bước B3 và B4 cho tới khi trên tất cả các phần tử trong mạch thiết lập được các giá trị thử nghiệm.

Ví dụ về phương pháp này được đưa ra trên hình 8.39. Việc áp dụng bước 1 và 2 đối với mạch trên hình 8.39a sẽ dẫn tới mạch trên hình 8.39b. Sau đó bằng cách áp dụng tuần tự các bước 3 và 4 đối với mạch trên hình 8.39b, chúng ta nhận được mạch trên hình 8.39c.

Phương pháp nói trên có ý nghĩa lý thuyết trên quan điểm xây dựng mạch với số lượng của bộ giá trị kiểm nghiệm tối thiểu. Trong thực tế không ứng dụng được vì phải tăng số lượng các phần tử lên nhiều.

Trên quan điểm thực hành thì ta cần phải tăng tính hiệu quả của quá trình tổng hợp các giá trị kiểm nghiệm với sự tăng tối thiểu số lượng các phần tử mạch.



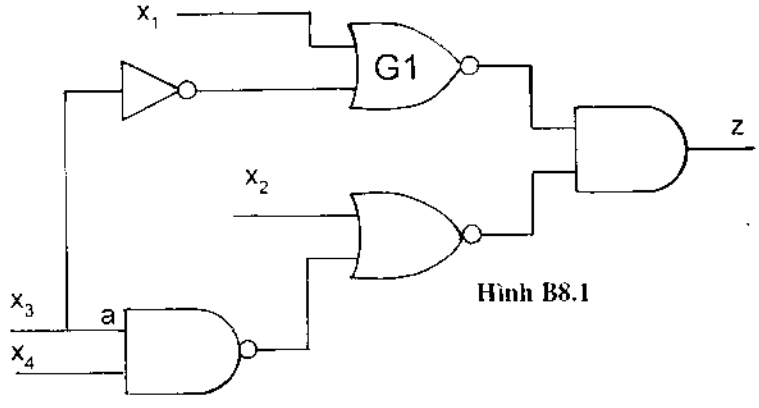

**Hình 8.39.** Phân rã các phần tử nhiều đầu vào thành các phần tử hai đầu vào: a) ví dụ mạch tổ hợp; b) xây dựng mạch theo cấu trúc dạng cây; c) phân rã thành các phần tử hai đầu vào.

**Bài tập cho chương 8**

1. Tìm mạch số có thể chứa một lỗi ngắt mạch không phát hiện được.

2. Có tồn tại một mạch tổ hợp C với một đường tín hiệu S nào đó và bộ giá trị thử nghiệm $t$ sao cho bộ giá trị $t$ có thể phát hiện cả hai lỗi s-a-0 và s-a-1 trên đường tín hiệu S ? Hãy cho ví dụ hoặc chứng minh rằng điều đó là không thể xảy ra.

3. Cho mạch số trên hình B8.1. Hãy xác định hàm ra của mạch đối với những lỗi sau:
   a) Lỗi bắc cầu AND giữa các đầu vào của phần tử G1.
   b) Lỗi kép {$x_3$ s-a-1, $x_2$ s-a-0 }.



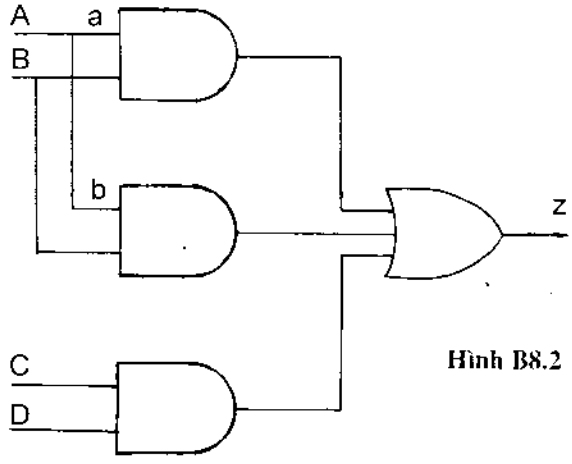
**Hình B8.1**

4. Với mạch điện trên hình B8.1, những bộ giá trị nào trong các bộ giá trị sau sẽ phát hiện lỗi $x_1$ s-a-0:
   a. ( 0, 1, 1, 1 );
   b. ( 1, 1, 1, 1 );
   c. ( 1, 1, 0, 1 );
   d. ( 1, 0, 1, 0 ).

5. Đối với mạch điện trên hình B8.1, hãy xác định biểu thức lôgiccho tất cả các bộ giá trị thử nghiệm phát hiện lỗi:
   a. $x_3$ s-a-0;
   b. $x_2$ s-a-0;
   c. $x_2$ s-a-1;

6. Hãy thực hiện những thao tác sau đối với mạch điện trên hình B8.2:

- Tìm tập hợp tất cả các bộ giá trị thử nghiệm phát hiện lỗi a: s-a-0;
- Tìm tập hợp tất cả các bộ giá trị thử nghiệm phát hiện lỗi b: s-a-0;
- Tìm tập hợp tất cả các bộ giá trị thử nghiệm phát hiện lỗi kép {a s-a-0, b s-a-0}




**Hình B8.2**

**Tài liệu tham khảo**


1. Saveliev. A. Ya. **Prikladnaya Theorya Tsiphrovykh Avtomatov.** Utsebnik dlya VUZOV po spets EVM. M Vysshaya Shkola 1987. 272 pp.
2. Saveliev. A. Ya, Ovtsinnikov. V. A. Konstruirovanie EVM i system. Utsebnik dlya VUZOV. M Vysshaya Shkola 1984. 248 pp.
3. Norenkov. I. P. Manitsev. V. B. **Systemy avtomatizirovannogo Proektirovaniya elektronnoi i vytsislitelnoi apparatury.** M Vysshaya Shkola 1994. 272 pp.
4. Ugryumov E. P. Proektirovanie elementov i uzlov EVB. M Vysshaya Skola 1987. 315pp.
5. K. Kinosita. **Logitsheckoe proektirovanhie SBIS.** M Mir 1989. 310 pp
6. R. S. Sandige. **Modern Digital Design.** McGraw-Hill, Inc. 1990. 741 pp.
7. Y. Hsu, K. F. Tsai, J. T. Liu, E. S. Lin. **VHDL modeling for Digital design synthesis.** Kluwer Academic Publishers 1995. 356pp.
8. Đặng Văn Chuyết. **Kỹ thuật điện tử số.** NXBGD, 2000.
9. Nguyễn Thúy Vân. **Kỹ thuật số**. NXBKHKT. 1999.
10. G. De Micheli. **Synthesis and Optimization of Digital Circuits.** Mc Graw-Hill, Inc. 1994. 580pp.
11. Z. Navabi. **VHDL. Analysis and Modeling of Digital Systems.** Mc Graw-Hill, Inc. 1998. 632pp.
12. M. Abramovici, Melvin A. Breuer, Arthur D. Friedman. **Digital Systems Testing and Testable Design.** Wiley-IEEE Press 1993. 653 pp.

# THIẾT KẾ MẠCH BẰNG MÁY TÍNH

*Tác giả:* TS. NGUYỄN LINH GIANG

*Chịu trách nhiệm xuất bản :* PGS. TS TÔ ĐĂNG HẢI
*Biên tập :* ThS. NGUYỄN HUY TIẾN
*Sửa bài :* NGỌC LINH
*Trình bày bìa :* HƯƠNG LAN

**NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT**
**70 TRẦN HƯNG ĐẠO – HÀ NỘI**

In 1000 cuốn khổ 16 x 24 cm, tại: Xưởng in NXB Văn hoá Dân tộc
Giấy phép xuất bản số: 113-244-24/4/2003
In xong và nộp lưu chiểu tháng 7 năm 2003