

Tuần	Nội dung	Giáo trình	BT, TN,...
1	Chương I: Khái niệm về Điều khiển Logic 1.1 Khái niệm về Điều khiển Logic 1.2 Mô hình hóa các hệ thống rời rạc 1.2.1 Đại số Bool 1.2.2 Automat hữu hạn		
2	1.2.3 Petri net 1.2.4 State Charts 1.2.5 StateFlow 1.2.6 GRAFCET		
3	1.3 Chuẩn IEC 61131 và các bộ điều khiển lập trình được 1.3.1 PLC và ngôn ngữ lập trình theo chuẩn 1.3.2 Các công cụ đặt cấu hình 1.3.3 Đơn vị tổ chức chương trình 1.3.4 Phương pháp cấu hình đặc biệt 1.3.5 Tổ chức PLCopen		
4	Chương II: Mạch logic tổ hợp 2.1 Định nghĩa và phân loại 2.2 Tổng hợp mạch logic tổ hợp 2.2.1 Phương pháp đại số 2.2.2 Phương pháp ma trận Các nô		Làm bài tập
5	2.2.3 Phương pháp Quine Mc. Clusky		
6	Chương III: Mạch logic tuần tự 3.1 Khái niệm cơ bản về mạch logic tuần tự		
7	3.2 Tổng hợp mạch logic tuần tự 3.2.1 Phương pháp ma trận trạng thái		Làm bài tập
8	3.2.1 Phương pháp ma trận trạng thái (tiếp) 3.2.2 Phương pháp GRAFCET		Làm bài tập

9	3.2.2 Phương pháp GRAFCET		Làm bài tập
10	Chương IV: Tổng quan về PLC 4.1 Giới thiệu về PLC 4.2 Cấu trúc phần cứng 4.3 Hoạt động của PLC		Làm thí nghiệm
11	4.4 Các lệnh trong PLC		Làm thí nghiệm
12	Chương V: Kỹ thuật lập trình PLC 5.1 Thiết kế chương trình dựa vào lưu đồ 5.2 Thiết kế chương trình dựa vào trạng thái		Làm bài tập và thí nghiệm
13	5.3 Kỹ thuật ghi dịch 5.4 Sử dụng biểu đồ chức năng tuần tự (SFC)		Làm bài tập và thí nghiệm
14	Chương VI: Ghép nối và truyền thông với PLC 6.1 Các thiết bị vào ra		Làm thí nghiệm
15	6.2 Ghép nối với PLC 6.3 Truyền thông với PLC		
	Chương VII: Mô hình hóa các sự kiện rời rạc 7.1 FSM – Done 7.2 Petrinet - Done 7.3 State Charts & State Flow		

CHƯƠNG 1: KHÁI NIỆM CHUNG VỀ ĐIỀU KHIỂN LOGIC

1.1 Khái niệm về ĐKLG

Một hệ thống có thể được coi là một tổ hợp các bộ phận tương tác lẫn nhau, được tổ chức để thực hiện một mục tiêu nào đó, thông thường là để đạt được giá trị gia tăng - thông qua quá trình thay đổi các tính chất vật lý, hóa học hay sinh học, sắp đặt lại vị trí, trao đổi thông tin – trên các loại nguyên liệu thô, năng lượng, bán thành phẩm, sức lao động – trong các điều kiện môi trường nhất định. Nếu như các quá trình xảy ra trong các điều kiện nhất định đều tuân theo những quy luật hóa lý hay sinh học của chính nó thì để đảm bảo kết quả của quá trình là các sản phẩm mong muốn ta có thể tác động vào các điều kiện này. Việc tác động vào các điều kiện để quá trình xảy ra như mong muốn gọi là điều khiển. Để thực hiện việc điều khiển cần phải theo dõi các đại lượng thay đổi liên quan đến bản thân quá trình và các tham số thể hiện các điều kiện mà quá trình đang xảy ra thông qua các thiết bị đo như các sensor. Các kết quả mong muốn được thể hiện như các lượng đặt. Như vậy điều khiển liên quan đến việc tiếp nhận các giá trị lượng đặt, các giá trị của các tham số, các giá trị liên quan đến đại lượng thay đổi theo quá trình, xử lý các thông tin này theo một quy luật nào đó, sau đó đưa ra các tín hiệu tác động lên quá trình.

Nếu theo dõi quá trình theo thời gian có thể phân loại quá trình thành loại liên tục và loại rời rạc. Quá trình là liên tục nếu có thể xác định hoặc mô tả được các đại lượng liên quan đến quá trình ở mọi điểm theo thời gian, ví dụ quá trình thay đổi nhiệt độ, thay đổi áp suất, các phản ứng hóa học, ... Quá trình là rời rạc nếu chỉ có thể biết được giá trị các đại lượng liên quan đến quá trình ở những thời điểm nhất định theo thời gian hoặc thậm chí không biết được thời điểm xuất hiện của chúng. Loại sau này gọi là các sự kiện. Sự kiện chỉ cho thấy sự hiện diện của chúng khi chúng xảy ra.

Để có thể điều khiển, khống chế được các quá trình, một vấn đề quan trọng là có thể có được mô tả toán học của chúng. Sử dụng các công cụ toán học để mô tả các quá trình gọi là mô hình hóa. Đây chính là quá trình dùng tư duy trừu tượng để mô tả các quá trình. Với tư duy trừu tượng một số quá trình có bản chất rất khác nhau lại có thể được mô tả bởi các mô hình giống nhau, do đó được nghiên cứu bằng các công cụ toán học giống nhau.

Các quá trình liên tục có thể được mô tả bởi hệ phương trình vi phân. Nhờ phép tính vi phân có thể mô tả được sự thay đổi của các đại lượng quan tâm trong những khoảng thời gian nhỏ tùy ý. Các hệ thống liên tục là đối tượng nghiên cứu của lý thuyết điều khiển tự động. Đặc biệt đối với lớp các hệ thống tuyến tính, được mô tả bởi hệ phương trình vi phân tuyến tính, các phương pháp và công cụ nghiên cứu đã được phát triển và ứng dụng từ lâu nay.

Đối với các hệ thống rời rạc, với cảm nhận ban đầu, có thể nghĩ rằng việc mô tả và nghiên cứu chúng sẽ dễ dàng hơn. Một trong những cách tiếp cận ngây thơ nhất đối với hệ thống rời rạc là có thể liệt kê được hết những đáp ứng có khả năng xảy ra. Ví

dụ một chiếc quạt điện thông thường có thể ở trạng thái chạy hoặc dừng. Quạt có thể chạy nếu ta đóng công tắc cấp nguồn điện cho động cơ quạt. Quạt sẽ dừng nếu ta cắt công tắc cấp điện cho động cơ. Hình dung về chiếc quạt sẽ phức tạp dần lên nếu bổ xung thêm những trạng thái thực tế khác. Chế độ chạy có thể cần phân biệt thêm chạy ở tốc độ nào, đơn giản nhất có thể là quạt có ba cấp tốc độ, 1, 2, 3. Khi trong chế độ chạy có thể có chế độ thay đổi hướng gió. Giữa các cấp tốc độ cố định có thể có chế độ chuyển giữa tốc độ này sang tốc độ khác và ngược lại để phông theo chế độ gió tự nhiên, gây cảm giác dễ chịu. Nếu hình dung chiếc quạt này lắp đặt ở một vị trí xa người sử dụng hoặc đây là một chiếc quạt để tạo nên sự thay đổi áp suất không khí trong một dây chuyền công nghiệp sẽ cần có một bộ phận cảm biến gió để nhận biết quạt có thực sự hoạt động hay không khi công tắc nguồn đã đóng. Để tránh bị ảnh hưởng khi điện áp lưới có sự thay đổi bất thường tín hiệu từ bộ cảm biến gió (cảm biến áp suất không khí) chỉ thực sự được xử lý sau một khoảng thời gian nhất định, ví dụ sau 30s. Những biến động nháy, mất điện áp nguồn ngắn 5 – 10s sẽ không ảnh hưởng gì đến trạng thái hoạt động của quạt. Rõ ràng là để mô tả hệ thống xác thực hơn, số lượng các trạng thái sẽ tăng lên nhanh chóng. Thực tế, một trong những khó khăn lớn khi nghiên cứu các hệ rời rạc là số lượng các trạng thái quá lớn, đến mức chỉ cần liệt kê ra chúng đã là không thể, ngay cả với sự trợ giúp của các máy tính hiện đại.

Khái niệm về điều khiển logic (Logic Control) liên quan đến các hệ thống rời rạc, trong đó đáp ứng của quá trình chỉ có thể biết được ở những thời điểm nhất định theo thời gian hoặc khi những sự kiện xảy ra. Thời điểm sự kiện xảy ra có thể hoàn toàn không biết trước được. Đáp ứng của hệ thống có thể chịu sự ảnh hưởng của tình trạng trước đó cũng như của các sự kiện nên các hệ thống loại này còn gọi là hệ phản ứng hay hệ tương tác (Reactive Systems). Điều khiển logic có thể hiểu là sự lựa chọn hay sự đưa ra quyết định, trong những hoàn cảnh nhất định, trong những điều kiện nhất định, cách tác động vào quá trình để có được đáp ứng như mong muốn.

Trong thực tế những tác động vào các hệ rời rạc có phạm vi rất rộng, đặt ra những vấn đề cần giải quyết cho quá trình điều khiển, có thể kể ra sau đây:

- Hệ thống có thể phải ở trong các chế độ làm việc khác nhau, tuân theo lệnh điều khiển từ bên ngoài. Chế độ đơn giản nhất có thể thấy là chạy (Start), dừng (Stop).
- Hệ thống có thể phải chuyển từ chế độ này sang chế độ khác theo một trình tự nhất định, xác định bởi một số điều kiện nhất định. Cách thức hoạt động này rất phổ biến đối với các quá trình công nghiệp. Ví dụ để khởi động một dây chuyền công nghệ luôn đòi hỏi những trật tự chặt chẽ, bộ phận nào phải chạy trước, bộ phận nào phải chạy sau, cũng như tuân thủ quá trình tăng tốc theo quy luật nhất định. Điều tương tự cũng xảy khi cần dừng dây chuyền lại.
- Đảm bảo quá trình xảy ra theo các trình tự về thời gian nhất định. Đảm bảo tính đồng bộ là ví dụ về nhiệm vụ kiểu này.

- Tương tác giữa các bộ phận. Trong một hệ thống các bộ phận có tương tác lẫn nhau, chế độ hoạt động ở khâu này có thể tạo ra những điều kiện nhất định quy định hoạt động của các bộ phận khác. Đảm bảo những mối quan hệ này trong các hệ thống phân tán, kích cỡ lớn là một nhiệm vụ không hề dễ dàng.
- Phản ứng tức thời trước một số sự kiện. Sự kiện có thể tham gia vào các hoạt động bình thường, ví dụ như một chuyển động đến một công tắc cuối hành trình, cần phải dừng lại hoặc chuyển động ngược lại, lệnh điều khiển đến khi một nút bấm chức năng nào đó tác động, ... Sự kiện thường tham gia đảm bảo an toàn cho quá trình như khi bánh răng truyền chuyển động bị vỡ, vật lạ thâm nhập khu vực nguy hiểm, hoặc can thiệp tức thời của người vận hành ấn nút dừng khẩn cấp khi thấy có hiện tượng bất thường.

Điều khiển logic được nghiên cứu trong nhiều lĩnh vực:

- Kỹ thuật tính toán (Computer Science);
- Lập trình (Programming);
- Mô phỏng (Simulation);
- Truyền thông (Communication);
- Các hệ thống điều khiển công nghiệp (Industrial Control).

Ở đây sẽ chỉ quan tâm đến những vấn đề của điều khiển logic trong các hệ thống điều khiển công nghiệp, trong đó người kỹ sư cần thiết kế hệ thống điều khiển để đảm bảo quá trình xảy theo đúng quy trình công nghệ yêu cầu, hoạt động an toàn, tin cậy, với hiệu quả cao nhất. Tính hiệu quả được thể hiện bởi sự kết hợp một cách tiết kiệm hệ thống trang thiết bị trong khi đảm bảo được xác suất giữa những lần dừng máy nhỏ (Mean Time Between Failures), máy móc dễ vận hành, dễ sửa chữa, tiết kiệm năng lượng, ... Những thước đo hiệu quả của hệ thống thiết bị máy móc có thể không dễ đánh giá ngay từ khâu thiết kế ban đầu nhưng có tính định hướng cho người kỹ sư khi tiếp cận các vấn đề đối với một hệ thống điều khiển công nghiệp.

Nhiệm vụ thiết kế một hệ thống điều khiển logic tuân theo các bước giống như các nhiệm vụ thiết kế bất kỳ nào, đó là:

1. Phân tích quá trình để làm rõ các yêu cầu về công nghệ mong muốn;
2. Trên cơ sở các yêu cầu công nghệ mong muốn cần mô tả được hệ thống bằng ngôn ngữ kỹ sư phù hợp. Ngôn ngữ kỹ sư ở đây được hiểu là một công cụ mô tả đặc thù nào đó mà dựa trên kết quả mô tả hệ thống, gọi là mô hình, có thể chuyển mô hình này sang dạng ứng dụng được thông qua các thiết bị phần cứng cùng với các phần mềm cần thiết;
3. Mô phỏng hệ thống. Với những hệ thống phức tạp hoặc quan trọng không thể dễ dàng xây dựng ngay các thiết bị thử nghiệm, có thể là do quá đắt tiền hoặc quá

nguy hiểm trong thử nghiệm. Các công cụ mô phỏng trên máy tính ngày nay cho phép thử nghiệm trên mô hình gần với các đặc điểm thực tế, từ đó có được những đặc tính của hệ thống cần quan tâm.

4. Triển khai hệ thống điều khiển trên các thiết bị thực. Quá trình này ngày nay thường đi cùng với việc thiết kế phần cứng và xây dựng các phần mềm.

Trong quá trình thiết kế cần phân biệt đặc thù của các phương pháp kỹ sư với các phương pháp nghiên cứu cơ bản khác. Phương pháp kỹ sư nhất thiết phải đi đến được các ứng dụng mà không nhất thiết phải được giải thích ngọn nguồn bằng các suy luận chặt chẽ, như trong tư duy kiểu toán học.

1.2 Mô hình hóa các hệ thống rời rạc

Mô tả các hệ thống rời rạc tỏ ra phức tạp hơn nhiều so với các hệ thống liên tục. Mức độ phức tạp thể hiện ở chỗ ta có thể rời rạc hóa các quá trình đến mức độ nào. Ở mức độ tiếp cận ban đầu quá trình có thể chỉ phân chia ra theo một vài chế độ làm việc, ví dụ như chạy và dừng như với chiếc quạt đã nói đến trên đây. Sau đó quá trình rời rạc có thể lại tiếp tục để phản ánh thực tế một cách chi tiết hơn. Giữa các trạng thái làm việc lại có thể có mối quan hệ với nhau, có thể xảy ra đồng thời hoặc theo một trật tự nhất định nào đó. Khi số lượng các trạng thái nhiều lên cùng với các mối quan hệ giữa chúng, rõ ràng việc xác định được các đặc tính, các tính chất của hệ thống trở nên khó khăn hơn nhiều. Nếu như mục đích của việc mô hình hóa hệ thống là để suy luận ra được, đoán trước được các đáp ứng của hệ thống thì việc đưa ra quá nhiều các khả năng đáp ứng xảy ra cũng không giúp ích được gì nhiều cho quá trình thiết kế hệ thống điều khiển.

1.1.1 Đại số Bool

Một trong những công cụ toán học có cơ sở đầy đủ nhất đã được nghiên cứu kỹ lưỡng là đại số Bool, theo tên của nhà toán học giữa thế kỷ XIX George Boole, hay còn gọi là đại số logic [1]. Trong đại số Bool khái niệm trừu tượng đưa ra có dạng đơn giản nhất là coi sự kiện chỉ có thể là có hoặc không, tồn tại hoặc không tồn tại, đúng hoặc sai. Khái niệm đơn giản đúng hoặc sai được thể hiện bằng hai chữ số 1 và 0 tạo nên cơ sở hệ đếm cơ số 2, biến tất cả các phép toán số học cơ bản chỉ còn lại là phép cộng. Về mặt vật lý đúng hay sai có thể được thực hiện bằng một mạch điện đóng hay mở hay việc tạo nên mức điện áp giữa 0 – 5 V. Các khóa bán dẫn dùng tranzito có thể dễ dàng thực hiện được các quá trình vật lý này đã tạo nên cuộc cách mạng lớn là sự ra đời của các thiết bị điện tử số, sau đó là máy tính điện tử mà ảnh hưởng của chúng đến mọi mặt đời sống, kinh tế, kỹ thuật thì mọi người đều biết.

Mặc dù dựa trên khái niệm cơ bản nhất là đúng hoặc sai, 0 hoặc 1, đại số Bool có khả năng mô tả hàng loạt các quá trình thực tế. Các mối quan hệ logic phức tạp như

sự lựa chọn các tác động cần thiết trong một tổ hợp rất lớn các tín hiệu đầu vào thể hiện qua các hàm logic. Các công cụ phân tích của đại số Bool cho phép tối thiểu hóa các phần tử logic cần thiết để xây dựng nên các bộ điều khiển logic hữu hiệu là cơ sở cho những ứng dụng thực tế của lý thuyết này.

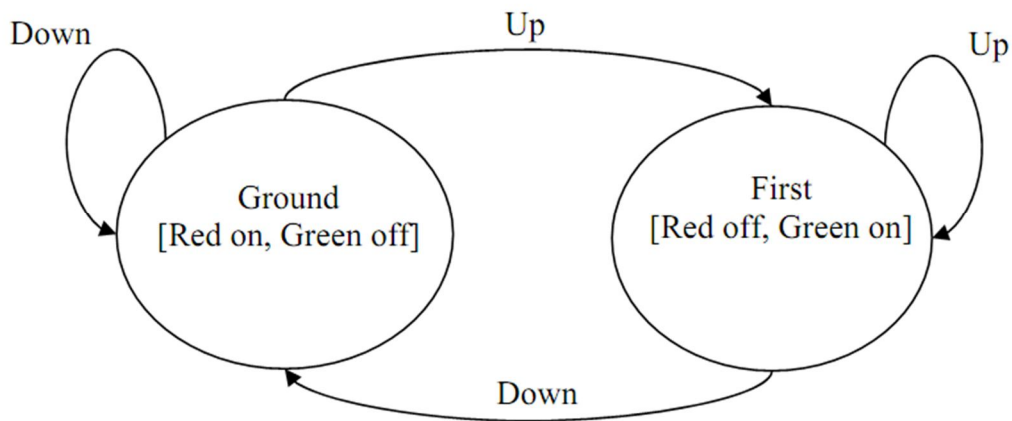
Đại số Bool là cơ sở để xây dựng nên các mạch điện tính toán, các vi mạch logic độ phức hợp cao (Complex Programmable Logic Device – CPLD) hay các mạch tổ hợp logic có thể lập trình được (Flexible Programmable Gate Array – FPGA), ngày càng được ứng dụng rộng rãi.

Tuy nhiên trừu tượng hóa đến mức coi một cái gì đó chỉ là đúng hoặc sai sẽ là quá khiên cưỡng khi mô tả các quá trình thực tế hoặc sẽ dẫn đến phải phân chia quá trình ra quá nhiều mức độ chi tiết đến mức không thể phân tích nổi. Để bù đắp những khiếm khuyết này lĩnh vực đại số logic đã phát triển lên các hướng chuyên sâu mới là logic mờ (Fuzzy Logic) và mạng nơron. Các bước phát triển cao hơn của logic đã tỏ ra có những ứng dụng thực tế quan trọng và cần đến những nghiên cứu chuyên sâu, ngoài phạm vi muốn đề cập đến ở đây.

Với sự phát triển của kỹ thuật máy tính con người đã có những công cụ hữu hiệu khác để mô tả một cách trừu tượng những quá trình thực tế mà không phải dựa trên tư duy kiểu toán học. Đó là những công cụ tư duy bằng ngôn ngữ hay bằng hình ảnh. Ở đây muốn nói đến các công cụ để mô hình hóa các quá trình một cách trực giác thông qua các loại đồ thị.

1.1.2 Automát hữu hạn (Finite State Machine - FSM)

Automat hữu hạn hay là máy trạng thái hữu hạn dùng ngôn ngữ hình ảnh, dưới dạng đồ thị để mô tả các quá trình [2]. Các trạng thái (state), thể hiện dưới dạng đồ thị là một hình khép kín, chữ nhật hay tròn, có thể gọi chung là quả bóng. Nối giữa các trạng thái là các mũi tên chỉ khả năng chuyển từ bước này sang bước khác (Transition). Đồ thị này được gọi là đồ thị trạng thái (State graph). Mỗi mũi tên ứng với một điều kiện logic cần kiểm tra hay là các tín hiệu đầu vào. FMS hoạt động bắt đầu từ một trạng thái ban đầu, qua các bước chuyển phụ thuộc vào các điều kiện logic có cho phép hay không, đến một trạng thái bất kỳ nào đó nếu có thể. Những trạng thái có thể đến được từ trạng thái ban đầu trong đồ thị gọi là trạng thái được phép. FMS có thể đưa ra tác động đầu ra phụ thuộc vào các đầu vào và trạng thái hiện tại. Nếu đánh số các trạng thái và gán cho các điều kiện logic các mức logic 0,1, có thể biểu diễn FMS dưới dạng bảng, rất thuận lợi cho biểu diễn dưới dạng ký tự, có thể chuyển từ môi trường soạn thảo này sang môi trường khác một cách dễ dàng.



Hình Error! No text of specified style in document..1 Ví dụ về FSM mô hình bộ điều khiển thang máy.

Ví dụ FSM cho trên hình 1.1. Giả sử có bộ điều khiển thang máy, chỉ có hai tầng, tầng 1: First; và tầng ngầm: Floor. Có hai lệnh chuyển động, lên: Up; và xuống: Down. Bộ điều khiển có hai đèn chỉ thị, đèn đỏ chỉ tầng ngầm: Red; và đèn xanh chỉ tầng 1: Green. Mỗi đèn sẽ sáng khi thang ở tầng tương ứng. Đồ thị trên hình 1.1 gồm hai quả bóng chỉ hai trạng thái Floor và First. Tín hiệu đầu vào là hai lệnh Up, Down. Hành động cần thực hiện khi ở các trạng thái là làm sáng đèn tương ứng và tắt đèn kia. Ở mỗi thời điểm bộ điều khiển sẽ theo dõi tín hiệu đầu vào và thực hiện bật, tắt đèn như mong muốn.

Nếu coi đầu vào: Up = 1; Down = 0;

Trạng thái: Floor = 0; First = 1;

Đèn (tín hiệu ra): On = 1; Off = 0,

Có thể lập bảng trạng thái như sau:

Bảng Error! No text of specified style in document.-1 Bảng trạng thái của FSM trên hình 1.1.

Trạng thái	Đầu vào	Trạng thái tiếp theo	Đèn đỏ Red	Đèn xanh Green
0	0	0	1	0
0	1	1	0	1
1	0	0	1	0
1	1	1	0	1

FSM có thể mô tả hệ thống một cách hữu hiệu, và được ứng dụng trong thiết kế các mạch điện tử số, các thủ tục truyền thông [3]. Các phần mềm thiết kế hệ thống số ngày nay cho phép khai báo các FMS phức tạp và tự động hóa quá trình chuyển từ mô

hình sang ứng dụng trên mạch điện tử rất thuận tiện. Trong các hệ thống tự động hóa phức tạp State graph rất hay được dùng, có thể không phải để thiết kế mà là để mô tả hoạt động và các tương tác giữa các bộ phận của hệ thống.

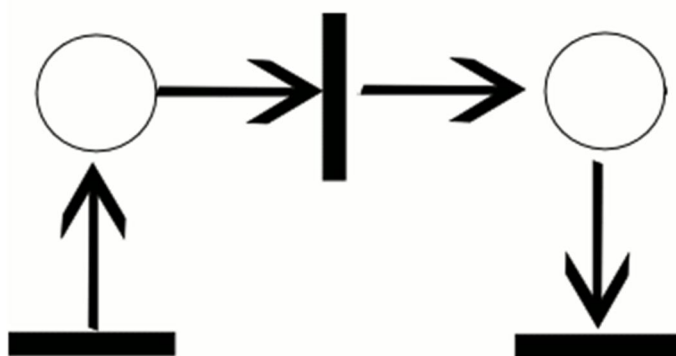
Mặc dù FSM có công cụ toán học cơ sở là lý thuyết đồ thị, nó cũng tỏ ra còn nhiều hạn chế, đó là mỗi trạng thái được coi là không chia nhỏ ra được nữa, dẫn đến số trạng thái có thể là rất lớn cũng như là các bước chuyển giữa chúng.

1.1.3 Petri Net

Petri Net, được phát minh bởi Carl Adam Petri năm 1939 dùng để mô tả các quá trình hóa học, là một ngôn ngữ toán học mô hình hóa cho các hệ thống phân tán [7]. Petri Net là một đồ thị có hướng, trong đó mỗi nốt của đồ thị thể hiện bước chuyển (transition, các sự kiện có thể xảy ra), ký hiệu bằng một gạch đứng, và các vị trí (nghĩa là các điều kiện), ký hiệu bằng vòng tròn nhỏ. Những cung có hướng, ký hiệu bằng các mũi tên, nối các nốt với các vị trí và ngược lại, thể hiện các vị trí đã chuẩn bị các điều kiện cho các bước chuyển xảy ra. Các cung không bao giờ nối giữa cùng các vị trí cũng như cùng các bước chuyển.

Các cung nối từ vị trí đến bước chuyển gọi là cung đầu vào. Các cung nối bước chuyển với vị trí gọi là các cung đầu ra.

Tại các vị trí có một số lượng các token nào đó. Sự phân bố các token tại các vị trí gọi là sự đánh dấu. Bước chuyển sẽ xảy ra nếu có token ở tất cả các cung đầu vào. Khi chuyển các token sẽ bị đưa sang đầu cuối của tất cả các cung đầu ra. Khi có khả năng chuyển, tất cả các bước chuyển đều có khả năng chuyển như nhau, tuy nhiên không thể biết là cái nào sẽ chuyển vì mỗi lần chỉ có một token được chuyển đi. Vì các token có thể phân bố ở các vị trí khác nhau, thậm chí nhiều token tại một vị trí, nên do tính không thể biết trước của mô hình PetriNet mô hình này có thể mô tả rất tốt tình trạng tranh chấp (concurrent) của hệ thống phân tán.



Hình Error! No text of specified style in document..2 Ví dụ về Petri Net.

Giống như các ngôn ngữ dùng hình ảnh khác Petri Net cho phép mô tả bằng đồ thị các quá trình nhảy cấp, bao gồm lựa chọn, suy luận và các hoạt động tranh chấp.

Petri Net có định nghĩa toán học chính xác cho các ngữ nghĩa của nó và có nền tảng lý thuyết đầy đủ để phân tích nó.

1.1.4 Statecharts và stateflows

○ Statecharts

Statecharts mở rộng khả năng của FSM do David Harel đưa ra trong những năm 1980 và công bố 1987 [4, 5], có khả năng mô tả những hệ thống rời rạc kiểu phản ứng theo sự kiện phức tạp như hệ tính toán nhiều CPU, các thủ tục truyền thông và các hệ điều khiển điện tử trong tự động hóa. Statecharts sử dụng FSM cơ bản nhưng bổ xung thêm ba khái niệm cơ bản, đó là phân cấp, tranh chấp và quảng bá truyền thông. Nhờ đó Statecharts vẫn giữ nguyên được những ưu điểm do tính trực giác của FSM mang lại nhưng mở rộng khả năng diễn tả với các mức độ chi tiết khác nhau, làm cho đặc tính của những hệ rất lớn được mô tả một cách dễ hiểu và quản lý được. Khi sử dụng công cụ đồ họa máy tính Statecharts có thể trở thành một môi trường mô tả đứng riêng, trong đó có thể phân chia hệ thống thành các bộ phận chức năng và các đặc tính dòng số liệu liên quan đến mỗi bộ phận và xem xét được tương tác giữa các bộ phận với nhau.

Ví dụ về Statecharts cho trên hình 1.2 về chế độ của đồng hồ bấm giờ. Trong chế độ bấm giờ ta muốn đồng hồ chỉ thị thời gian theo phút, chính xác đến phần trăm giây, dưới dạng: mm:ss.cc. Chế độ bấm giờ có thể chạy lặp lại (Lap) hoặc chỉ bấm giờ một lần (Lap_stop). Đồ thị gồm hai trạng thái Stop (dừng) và Run (chạy). Trong Stop có hai trạng thái con là Reset (xóa hết) và Lap_stop (dừng vòng lặp). Trong trạng thái Run có hai trạng thái con là Running và Lap, ngoài ra còn chứa một đồ thị trạng thái TIC thực hiện việc đếm giờ chính xác đến phần trăm giây (cent). Các trạng thái được kích hoạt nhờ nút bấm đặt chế độ, tương ứng với các tín hiệu logic chạy (START) và chạy lặp lại (LAP). Các nút bấm này nếu bấm tiếp sẽ về chế độ trước đó. Ví dụ đang trong trạng thái Reset, bấm nút START sẽ chuyển sang Running. Nếu đang trong Running bấm START lần nữa sẽ về chế độ xóa hết Reset. Tín hiệu START cũng tác động tương tự giữa trạng thái vòng lặp (Lap) và xóa vòng lặp (Lap_stop). Trong các trạng thái có thể gắn với các hoạt động (activities). Ví dụ trạng thái Running có hoạt động trong quá trình trạng thái này tích cực, ký hiệu là:

during:	/trong khi
disp_cent=cent;	/chỉ thị phần trăm giây
disp_sec=sec;	/chỉ thị giây
disp_min=min ;	/chỉ thị phút
...	

- Các tác động và các điều kiện gắn với các trạng thái và các điều kiện để xác định hành vi của đồ thị trạng thái.

Với Stateflow có thể mở rộng khả năng của đồ thị trạng thái bằng cách:

- Thêm vào các cấu trúc phân cấp;
- Mô hình các trạng thái làm việc song song (tranh chấp);
- Xác định các hàm chức năng bằng đồ thị, bằng các thủ tục chèn vào hoặc bằng các bảng trạng thái.
- Sử dụng logic thời gian để lập lịch cho các sự kiện.
- Xác định các vector, ma trận và các kiểu dữ liệu.

Hơn nữa, Stateflow sẽ tiến hành mô phỏng mô hình đã được xây dựng để nghiên cứu các đặc tính cần quan tâm của hệ thống. Khả năng rất mạnh của Stateflow là tự động chuyển sang dạng mã chương trình C dùng Stateflow[®] Coder[™], tích hợp cùng phần mã C của Simulink ra dạng có thể cài đặt vào các bộ điều khiển. Điều này rút ngắn rất nhiều thời gian lao động của các kỹ sư trong quá trình thiết kế hệ thống điều khiển.

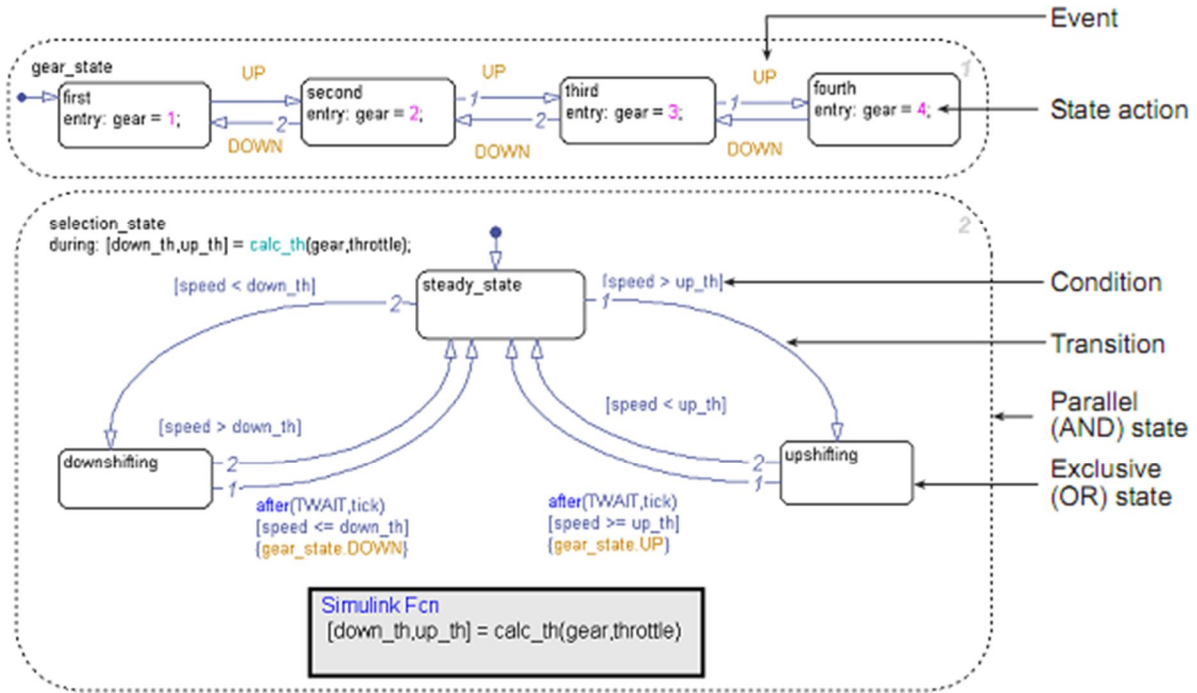
Ví dụ về đồ thị trạng thái của Stateflow cho trên hình 1.4. Đây là mô hình một hộp số tự động điều khiển ô tô. Hộp số sẽ tự động chuyển số tùy theo tốc độ của xe và độ mở của đường vào ga cho động cơ.

- Mô hình có hai trạng thái có thể hoạt động song song là: gear_state (trạng thái hộp số) và selection_state (lựa chọn số). Các trạng thái song song, cùng xảy ra (parallel AND states) có vành bao bên ngoài bằng đường nét đứt. Trong mỗi hai trạng thái song song này có các trạng thái con. Các trạng thái con có vành ngoài nét liền là những trạng thái loại trừ nhau, nghĩa là ở mỗi thời điểm chỉ có một trong chúng được kích hoạt.

- Các trạng thái nối với nhau bằng các cung chỉ sự kiện (UP, DOWN) hoặc các điều kiện (speed > down_th: tốc độ lớn hơn ngưỡng thấp; speed < down_th: tốc độ nhỏ hơn ngưỡng thấp; speed > up_th: tốc độ lớn hơn ngưỡng tốc độ cao; speed < up_th: tốc độ nhỏ hơn ngưỡng tốc độ cao; ...).

- Với mỗi trạng thái có thể có các tác động (action) gây nên các hoạt động cần thiết khi hệ thống đang ở trong một trạng thái nào đó. Các tác động có thể là loại kích hoạt khi vào trạng thái, entry; khi đang ở trong trạng thái, during, khi ra khỏi trạng thái, exit.

- Trong trạng thái có thể có hàm tính toán, như hàm calc_th: [down_th, up_th] = calc_th(gear, throttle). Hàm calc_th tính toán các giá trị ngưỡng tốc độ thấp, tốc độ cao tự vị trí hộp số gear và độ mở của đường đưa ga vào buồng đốt throttle.



Hình Error! No text of specified style in document..4 Ví dụ Stateflow điều khiển hộp số tự động xe ô tô.

Mặc dù không có phân ngữ nghĩa chặt chẽ như UML [6, 9], điều có thể dẫn tới không tối ưu về mã phát sinh, lỗi chương trình rất khó phát hiện và gỡ rối, nhưng Stateflow rất phù hợp cho những người phát triển ứng dụng trong các hệ thống điều khiển, trong đó người kỹ sư tập trung vào các nhiệm vụ đảm bảo chức năng điều khiển cho hệ thống chứ không phải là các chuyên gia về lập trình hay mô hình hóa.

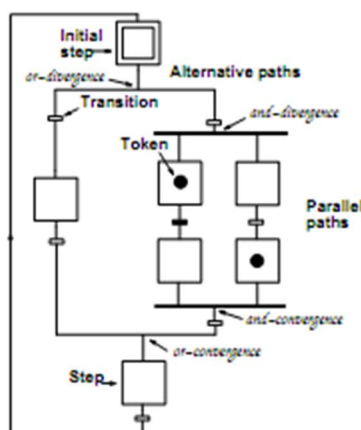
1.1.5 GRAFCET

GRAFCET cũng là một ngôn ngữ đồ thị được phát triển ở Pháp từ 1977, như là một phương pháp mô tả đặc tính cho các bộ điều khiển logic [10]. Từ năm 1988 GRAFCET được công nhận như một chuẩn quốc tế bởi IEC và mang tên đồ thị hàm tuần tự (Sequential Function Charts – SFC). Ngày nay GRAFCET/SFC là một phần trong chuẩn quốc tế ngôn ngữ lập trình IEC 61131 cho các bộ điều khiển logic lập trình được (Programmable Logic Controller – PLC). GRAFCET được dùng phổ biến trong công nghiệp vì giao diện đồ thị rõ ràng của nó. GRAFCET có cơ sở toán học từ mạng Petri Net.

GRAFCET có cú pháp đồ thị, ví dụ cho trên hình 1.5. Nó bao gồm các trạng thái (steps), thể hiện bằng một hình chữ nhật, và các bước chuyển (transitions), thể hiện bằng các gạch ngang. Lưu ý đối với GRAFCET trạng thái (state) được gọi là step. Trạng thái ban đầu (Initial step) thể hiện bởi hình vuông nét đôi sẽ được kích

hoạt trước tiên khi hệ thống bắt đầu hoạt động. Các trạng thái có thể được kích hoạt lần lượt hoặc đồng thời.

Trạng thái (steps): trạng thái có thể tích cực hoặc không. Trạng thái tích cực được đánh dấu bằng một token trong nó. Các trạng thái tích cực xác định trạng thái của hệ thống tại một thời điểm nào đó. Gắn với mỗi trạng thái có thể có một hay vài hành động. Các hành động được thực hiện khi trạng thái tích cực.



Hình *Error! No text of specified style in document.*5 Cú pháp của GRAFCET.

Các bước chuyển (transitions): dùng để kết nối các trạng thái. Mỗi bước chuyển có một độ hấp thụ nhất định. Bước chuyển là được phép nếu tất các trạng thái phía trên nó đều tích cực. Khi độ hấp thụ của một bước chuyển được phép trở nên là đủ thì bước chuyển được kích hoạt ngay lập tức. Khi bước chuyển kích hoạt các trạng thái trên nó trở nên thụ động và các trạng thái tiếp sau nó trở nên tích cực.

Hành động (actions): gồm hai loại, loại theo mức và loại theo xung. Loại theo mức kéo dài một khoảng thời gian hữu hạn theo một biến logic nào đó và giữ nguyên trạng thái tác động chừng nào trạng thái gắn với nó còn tích cực. Hành động theo mức có thể là có điều kiện hoặc không điều kiện. Loại hành động theo xung có nhiệm vụ thay đổi giá trị một biến nào đó. Biến có thể là logic nhưng không nhất thiết như vậy. Một hành động theo xung được thực hiện ngay khi trạng thái của nó trở nên tích cực. Có thể đưa vào các biến phụ thuộc thời gian để tạo nên các hành động có trễ hoặc xảy ra trong khoảng thời gian nhất định. Một hành động theo mức bao giờ cũng có thể chuyển thành một hành động theo xung.

Độ hấp thụ (receptivities). Mỗi bước chuyển có một độ hấp thụ nhất định. Độ hấp thụ có thể là một điều kiện logic, một sự kiện hay điều kiện logic kết hợp với sự kiện. Trên đồ thị sự kiện thể hiện bằng một chữ cái (biến) bên cạnh trái của nó có mũi tên lên hoặc xuống, ví dụ $\uparrow x$, $\downarrow y$. Mũi tên lên chỉ sự kiện xảy ra ở sườn lên của xung, mũi tên xuống chỉ sự kiện gắn với sườn xuống.

Ngôn ngữ GRAFCET được phát triển với mục đích sử dụng cho các bộ PLC. Ngôn ngữ được dùng trước hết với mục đích mô tả hệ thống bằng đồ thị để có thể

phân tích, đánh giá sự hoạt động một cách trực giác. Việc ứng dụng thực tế được thực hiện bằng cách chuyển đồ thị GRAFCET sang một ngôn ngữ thông dụng khác là đồ thị dạng bậc thang (Ladder Diagrams – LD). LD là ngôn ngữ đơn giản, thừa hưởng từ các sơ đồ điều khiển logic dùng rơle trước đây. Các sơ đồ LD rất dễ thực hiện đối với các hệ thống nhỏ nhưng khi hệ thống trở nên phức tạp hơn thì sơ đồ LD sẽ rất khó quản lý.

Về mặt ứng dụng GRAFCET thường đi cùng với một loại PLC do một hãng phát triển. Do đó sử dụng GRAFCET chỉ hiệu quả khi ta định xây dựng hệ thống điều khiển trên loại PLC đó, khi đó phần mềm soạn thảo chương trình sẽ chuyển tự động hệ thống mô tả bởi GRAFCET sang ngôn ngữ LD, cài đặt trên PLC. Nếu không có những điều kiện này thì việc chuyển GRAFCET sang dạng cài đặt trên PLC sẽ rất khó khăn, đến mức mà những ưu thế của ngôn ngữ đồ thị cũng không còn giá trị gì.

1.3 Chuẩn IEC 61131 và các bộ điều khiển lập trình được

1.3.1 PLC và ngôn ngữ lập trình theo chuẩn

PLC là bộ điều khiển logic lập trình được, được nhiều nhà sản xuất phát triển với kích cỡ từ nhỏ đến lớn và được ứng dụng rộng rãi trong công nghiệp. Tuy nhiên tính phổ biến của PLC làm nảy sinh hàng loạt vấn đề. Mặc dù vẫn mang một số đặc điểm chung về phần cứng, phần mềm nhưng giữa các loại PLC có nhiều điểm khác nhau, đặc biệt là về phần mềm mà cụ thể là ngôn ngữ lập trình điều khiển. Các hãng khác nhau có thể phát triển những tập lệnh khác nhau, các hàm khác nhau và cách sử dụng các chức năng quan trọng như bộ đếm, bộ định thời, truyền thông cũng có nhiều điểm khác biệt. Ngay cả giữa các loại PLC của cùng một hãng cũng có thể có sự khác nhau rất lớn. Sự bất tương thích gây ra nhiều khó khăn cho người sử dụng khi muốn chuyển đổi chương trình điều khiển có sẵn trên một loại PLC này sang một loại PLC khác khi nâng cấp hoặc sửa chữa hệ thống, khi muốn sử dụng phối hợp nhiều loại PLC khác nhau trong cùng một thiết bị máy móc. Ngay cả việc học và sử dụng các loại PLC khác nhau cũng sẽ tiêu tốn nhiều công sức và thời gian.

Do những lý do trên mà tổ chức IEC (International Electrotechnical Commission) đề ra chuẩn cho ngôn ngữ lập trình mang tên IEC 61131 [12]. IEC 61131 không phải là một ngôn ngữ mà là các chuẩn để ngôn ngữ cụ thể phải tuân theo. Chuẩn IEC 61131 đưa ra các quy định về bộ điều khiển khả trình, trong đó có PLC, và các thiết bị ngoại vi đi kèm, từ phần cứng (cơ khí, điện, điện tử, khí nén, thủy lực, ...), truyền thông đến phần mềm và ngôn ngữ lập trình. Bộ tiêu chuẩn này gồm nhiều phần, mỗi phần xét đến một khía cạnh nhất định, và do đó tạo thành một chuẩn “con” bên trong bộ tiêu chuẩn IEC 61131 và thường được kí hiệu bởi số thứ tự của phần (ví dụ IEC 61131-1, IEC 61131-2,...). Trong đó tiêu chuẩn IEC 61131-3, tức phần 3 của bộ tiêu chuẩn

IEC 61131, với tên gọi “Programming Languages”, qui định về các ngôn ngữ lập trình cũng như cách thức lập trình điều khiển cho tất cả các thiết bị, các quá trình và các bộ điều khiển. Những qui định trong tiêu chuẩn IEC 61131-3 đem lại một cách nhìn nhận mới về lập trình cho các hệ thống điều khiển, đảm bảo tính hiệu quả cao và sức mạnh.

Chuẩn IEC 61131-3 nói riêng và bộ tiêu chuẩn IEC 61131 nói chung đã vượt ra ngoài giới hạn là một bộ tiêu chuẩn về PLC mà trở thành một bộ tiêu chuẩn cho các thiết bị điều khiển khả trình (Programmable Controller) nói chung. Hiện nay, phần lớn các bộ điều khiển trong thực tế đều là các bộ điều khiển khả trình. Bởi vậy, phạm vi áp dụng của bộ tiêu chuẩn IEC 61131 trở nên rất rộng lớn. Một điểm cần chú ý nữa là phần nhiều các qui định trong bộ tiêu chuẩn IEC 61131 ở dạng khuyến cáo, nghĩa là nên được tuân theo chứ không bắt buộc. Bởi vậy mặc dù nhiều sản phẩm của các hãng khác nhau được nói là “tuân theo chuẩn IEC 61131” nhưng vẫn có thể không thực hiện đầy đủ và hoàn toàn đúng như những qui định đề ra trong chuẩn.

Dưới đây sẽ chỉ ra những đặc điểm quan trọng nhất mà chuẩn IEC 61131-3 đem lại cho các ngôn ngữ lập trình điều khiển.

1.3.2 Không phụ thuộc vào một phần cứng cụ thể nào.

Điều này cần thiết để chương trình có thể mang đi được (Portable). Do tính có thể mang đi được có thể tạo nên các thư viện gồm các chương trình nhỏ, được xây dựng cho các ứng dụng phổ biến, có thể ghép nối vào các ứng dụng lớn hơn. Điều này cũng có nghĩa là các đoạn mã có thể sử dụng lại (reuseable). Rõ ràng là thời gian và công sức của những người phát triển ứng dụng sẽ được giảm đáng kể.

1.3.3 Dùng nhiều ngôn ngữ trong cùng một chương trình điều khiển

Đây là một trong những ưu điểm nổi bật nhất của chuẩn. Việc sử dụng kết hợp nhiều ngôn ngữ lập trình khác nhau trong cùng một chương trình là điều mong mỏi của hầu hết các lập trình viên. Trong cùng một chương trình điều khiển, người lập trình có thể sử dụng đồng thời và trực tiếp nhiều ngôn ngữ lập trình khác nhau. Điều này giúp tăng tính linh hoạt và hiệu quả của việc lập trình bởi có thể tận dụng tối đa các ưu điểm của từng ngôn ngữ lập trình.

IEC 61131-3 định nghĩa 5 ngôn ngữ lập trình: Ladder (LD), Function block diagram (FBD), Sequential function chart (SFC), Structure Text (ST), Instruction List (IL) phục vụ cho một dải rộng các ứng dụng. Người lập trình PLC mọi nơi sẽ sử dụng cùng ngôn ngữ lập trình, ngân sách đào tạo sẽ giảm đi, nhất là khi dùng thiết bị của nhiều hãng khác nhau tuân theo chuẩn này, khi cần thiết chỉ cần bổ sung thêm một vài kiến thức về một bộ điều khiển mới. Chuẩn giúp tăng hiệu suất cũng như giảm thời gian thực hiện một dự án tự động hóa bằng cách tái sử dụng các thành phần (chương trình) đã được phát triển trước trong các dự án khác hoặc bởi những người khác.

Tuy nhiên không phải khi nào cũng sử dụng được 5 ngôn ngữ lập trình này, còn phụ thuộc vào tốc độ vi xử lý, loại PLC hay mức độ hỗ trợ của hãng. Thông thường thì việc sử dụng ngôn ngữ như sau đây:

- SFC dùng cho quá trình xử lý lặp đi lặp lại, có liên động hay những hoạt động đồng thời.
- LD được chấp nhận rộng rãi bởi người lập trình PLC khắp toàn cầu, dùng cho những ứng dụng vào ra số, xử lý cơ bản, khá dễ dàng cho việc thay thế code về sau.
- IL có tốc độ xử lý nhanh do sát với ngôn ngữ máy và hay được sử dụng ở châu Âu.
- ST là ngôn ngữ hay được dùng ở châu Âu, dùng cho những người quen với lập trình bậc cao, thực hiện được các phép toán phức tạp, nếu lập trình bằng IL và ST gây khó khăn trong việc phát hiện và sửa lỗi sau này.
- FBD dùng cho các vào ra số hay những xử lý cơ bản, tuy nhiên lại tốn diện tích màn hình quan sát khi lập trình.

1.3.4 Các công cụ đặt cấu hình

Ở các phương pháp cấu trúc hóa các chương trình PLC truyền thống, các ứng dụng được gói gọn trong các khối với các đặc tính runtime (khi đang chạy) không rõ ràng, việc cấu hình đơn thuần là chọn PLC và phân công vào ra, sau đó lập trình dựa theo phân công ban đầu này, sẽ là rất khó khăn khi chương trình của chúng ta dài hàng trăm trang. IEC 61131-3 cung cấp các phương tiện chuẩn hóa và tinh vi để tháo gỡ khó khăn này. Vì chương trình là độc lập với phần cứng, nên việc cấu hình cần phải xác định đặc tính runtime cho chương trình (PROGRAMs) và khối chức năng (FBs), giao tiếp giữa các cấu hình và gán các biến cho địa chỉ phần cứng PLC cụ thể. Giờ đây chúng ta có thể cấu hình bằng chương trình nên thoát khỏi gò bó lúc lập trình và cũng không phải nhớ nhiều nữa. Quản lý dự án có thể chia chương trình thành các module cho nhiều người làm sau đó tổng hợp lại một cách dễ dàng.

1.3.5 Lập tài liệu dự án một cách tiện lợi và nhanh chóng

Chuẩn cũng cho phép chương trình cũng như thiết bị tạo điều kiện để giám sát, chẩn đoán hệ thống và thu thập dữ liệu phục vụ cho việc tổng kết và lập kế hoạch chính xác, dễ dàng.

1.3.6 An toàn và tiện lợi hơn khi dùng các biến và kiểu dữ liệu

Với PLC và phương pháp lập trình thông thường thì việc truy cập dữ

liệu được thực hiện bởi các biến toàn cục hay là địa chỉ tuyệt đối (phần cứng), thông thường là địa chỉ vào ra, cờ, khối dữ liệu chẳng hạn như I0.0, I0.1, Q0.0,... Người lập trình phải tự phân bổ vị trí của chúng và phải hết sức cẩn thận nếu không sẽ xảy ra trường hợp các phần của chương trình ảnh hưởng lẫn nhau (ví dụ ghi đè dữ liệu).

Khi lập trình theo chuẩn IEC 61131-3 thì không dùng địa chỉ phần cứng trực tiếp mà thay vào đó là việc sử dụng các biến được đặt tên rõ ràng. Người lập trình cũng không phải xác định chúng cần được lưu giữ ở đâu mà chương trình tự động sắp xếp. Mỗi biến có một kiểu dữ liệu cụ thể như Bool, Byte, WORD, DWORD, LWORD, Integer (SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT), REAL, LREAL, TIME, DATE, TOD hay TIME_OF_DAY, DT hay DATE_AND_TIME, WString. Biến cục bộ chỉ có ý nghĩa trong phạm vi một đơn vị tổ chức chương trình, nên không lo ảnh hưởng khi sử dụng ở đơn vị khác.

1.3.7 Đơn vị tổ chức chương trình

Đối với các chương trình điều khiển lớn và phức tạp, phương pháp lập trình có cấu trúc được sử dụng thay cho phương pháp lập trình tuyến tính. Trong các hệ lập trình cho PLC trước kia, phương pháp lập trình có cấu trúc được hỗ trợ thông qua việc tổ chức chương trình thành các khối (block), với ý nghĩa như là các thành phần nhỏ nhất xây dựng nên chương trình (vì vậy gọi là các building block). Các khối này được phân loại theo nội dung chứa bên trong khối, thông thường bao gồm: khối tổ chức chương trình (OB - Organisation Block), khối chức năng (FB - Function Block), khối dữ liệu (DB - Data Block). Có thể thấy là ở đây khối chức năng và dữ liệu của khối chức năng được tách rời nhau tương ứng trong các khối FB và DB. Cách phân chia như thế này gợi nhớ đến các ngôn ngữ lập trình quen thuộc như Pascal hay C với chương trình chính, hàm và dữ liệu toàn cục (global data).

Chuẩn IEC 61131-3 đã tiến một bước xa hơn khi phân chia các khối, được gọi tên là các đơn vị tổ chức chương trình (**POU** - Program Organisation Unit), theo chức năng và sự phân cấp cấu trúc. Có ba loại POU được quy định là: Hàm (FUN - FUNction), khối chức năng (FB - Function Block) và chương trình (PROG - PROGram). Một điểm đáng chú ý là dữ liệu của các khối chức năng không nằm riêng nữa mà được đưa vào ngay trong chính khối đó. Đó chính là sự đóng gói dữ liệu, một trong những nguyên tắc cơ bản của phương pháp lập trình hướng đối tượng. Việc sử dụng POU giúp hạn chế được chủng loại khối được sử dụng, thống nhất hóa và đơn giản hóa để người sử dụng dễ dàng hơn. Hơn nữa, POU được thiết kế hoàn toàn độc lập với đối tượng, chính vì thế mà có thể tái sử dụng chúng trên nhiều nền khác nhau.

1.3.8 Phương pháp cấu hình đặc biệt

Thông thường khi lập trình PLC việc cấu hình là chọn loại PLC và phân công

vào ra, sau đó viết chương trình dựa theo phân bố ban đầu này. Các ứng dụng được đưa vào các khối (ví dụ OB) và không có đặc tính lúc vận hành rõ ràng.

IEC 61131-3 áp dụng những tiến bộ của công nghệ kỹ thuật mới cho phép việc mô hình một dự án PLC gồm những ứng dụng có nhiều vi xử lý. Mô hình phần mềm theo chuẩn giúp người dùng cấu trúc hóa các ứng dụng hướng tới thực tiễn bằng cách xây dựng các khối tổ chức chương trình POU để tạo điều kiện dễ dàng cho việc bảo trì, thu thập dữ liệu và tăng khả năng chẩn đoán của PLC. Một phần mềm đồng nhất là rất cần thiết cho việc tăng tính linh hoạt của các ứng dụng. Các tài nguyên của PLC (các vi xử lý hay khối CPU) được gán đặc tính vận hành khi cấu hình và điều đó làm cho chương trình của ta độc lập với phần cứng.

1.3.9 Tổ chức PLCopen

IEC 61131 ra đời mang lại nhiều lợi ích cho cả người sử dụng cuối và cả nhà sản xuất thiết bị logic khả trình. Hiện nay hầu hết các hãng đều cam kết sản xuất sản phẩm tuân theo chuẩn này, có thể là một phần hay toàn phần. Cũng phải nói rằng chuẩn không chỉ áp dụng cho PLC, mà cho hầu hết các thiết bị điều khiển khả trình như PAC, PLC, các bộ điều khiển quá trình,.. Nhiều tổ chức đang hoạt động nhằm phổ biến rộng rãi chuẩn, trong đó có việc ra đời của tổ chức PLCopen, tổ chức khuyến khích các thành viên theo chuẩn. PLCopen đã có rất nhiều thành viên, mà đặc biệt là các công ty tập đoàn lớn như Siemens, ABB, Allen Bradley, OMRON, Mitsubishi Electric, ... Hiện tổ chức không chỉ hoạt động ở châu Âu mà có cả ở Mỹ, Á.

Công ty Smart software solution, cũng là thành viên của PLCopen, phát triển phần mềm CoDeSys và thành lập tổ chức automation alliance với hơn 100 thành viên cam kết sử dụng phần mềm của hãng. CoDeSys không chỉ được chấp nhận ở châu Âu mà đang dần phát triển trên toàn thế giới, là các công ty ở Đức, Thụy sĩ, Italia, Áo, Pháp, Anh, Bỉ, Phần Lan, Thụy Điển, Nga, Mỹ và Trung Quốc với nhiều loại sản phẩm như PLC, CNC/PLC combination, Bộ truyền động thông minh, DCS (decentralized control system), Panel PLC, Lõi PLC, các module vào ra thông minh,... Phần mềm CoDeSys lập trình cho thiết bị điều khiển khả trình hoàn toàn tuân theo chuẩn IEC 61131-3. Với một phần mềm lập trình chung theo chuẩn và các hãng sản xuất thiết bị phần cứng cam kết sản phẩm hoàn toàn hỗ trợ thì việc lập trình sẽ thống nhất và đơn giản hơn cho người sử dụng.

Ngoài các hãng được kể trên còn rất nhiều hãng lớn nhỏ nữa sản xuất các sản phẩm hỗ trợ theo tiêu chuẩn IEC 61131 như BECKHOFF, Rexroth, Danfoss, Schneider, Yokogawa, Emerson,... Qua đó đủ thấy rằng chuẩn đã được chấp nhận rộng rãi như thế nào. Có một thực tế là nhiều hãng vẫn dùng phần mềm riêng cho PLC hay PAC và các bộ điều khiển khả trình của họ, do vậy vẫn chưa phải là hoàn toàn theo chuẩn nên chưa thể tận dụng hết được những ưu điểm của lập trình theo chuẩn, cũng chưa thể tiết kiệm được tối đa thời gian và công sức cho việc lập trình. Chẳng hạn, khi mua PLC của nhiều hãng, chúng ta vẫn phải học khá nhiều, cả phần cứng lẫn phần mềm, cũng

chưa thể lấy chương trình lập trình với phần mềm của hãng này đổ sang PLC của hãng khác được. Trong tương lai, những hạn chế trên sẽ được khắc phục vì tiêu chuẩn này chắc chắn sẽ còn được chấp nhận rộng hơn nữa và lập trình theo.

CHƯƠNG 2: MẠCH LOGIC TỔ HỢP

2.1 Cơ sở toán học về đại số logic

2.1.1 Hàm và biến logic

a. Biến logic

Biến số x được gọi là biến logic nếu x thuộc tập hợp B chỉ gồm 2 phần tử ký hiệu là 0 và 1. Nghĩa là biến logic x chỉ nhận hai giá trị 0 và 1.

$$x \in B = \{0;1\}$$

b. Hàm logic

Hàm số f của các biến x_1, x_2, \dots, x_n được gọi là hàm logic khi và chỉ khi các biến x_1, x_2, \dots, x_n là các biến logic và giá trị của hàm số f cũng là giá trị logic, tức là f cũng chỉ có 2 giá trị 0 và 1.

$$f(x_1, x_2, \dots, x_n) \in B = \{0;1\} \text{ với } x_1, x_2, \dots, x_n \in B = \{0;1\}.$$

c. Các phép toán logic cơ bản

- Phép nghịch đảo (NOT)

Ký hiệu: nghịch đảo của một biến logic x ký hiệu là \bar{x} .

Phép nghịch đảo được định nghĩa thông qua bảng giá trị 2.1 như sau

Bảng 2.1. Bảng giá trị của phép nghịch đảo

x	$f(x) = \bar{x}$
0	1
1	0

- Phép cộng logic (OR)

Phép cộng logic được thực hiện với 2 biến logic x và y ký hiệu là $x+y$

Phép cộng logic được định nghĩa thông qua bảng giá trị 2.2 như sau:

Bảng 2.2. Bảng giá trị của phép cộng logic

x	y	$f(x,y) = x+y$
0	0	0
0	1	1
1	0	1
1	1	1

Kết quả của phép cộng logic hai biến logic chỉ nhận giá trị 0 khi và chỉ khi cả hai biến logic có giá trị 0

- Phép nhân logic (AND)

Phép nhân logic được thực hiện với 2 biến logic x và y ký hiệu là $x.y$ hoặc $x*y$ hoặc đơn giản là xy

Phép nhân logic được định nghĩa thông qua bảng giá trị 2.3 như sau:

Bảng 2.3. Bảng giá trị của phép nhân logic

x	y	$f(x,y) = x.y$
0	0	0
0	1	0
1	0	0
1	1	1

Kết quả của phép nhân logic hai biến logic chỉ nhận giá trị 1 khi và chỉ khi cả hai biến logic có giá trị 1

2.1.2 Các tính chất và định luật logic cơ bản

a. Các tính chất của phép toán logic

- * Tính chất giao hoán

Kết quả của phép tính giao hoán là không đổi nếu ta đổi chỗ hai biến logic cho nhau

$$x+y = y+x$$

$$x.y = y.x$$

- * Tính chất kết hợp

$$x+y+z = (x+y)+z = x+(y+z)$$

$$x.y.z = (x.y).z = x.(y.z)$$

- * Tính chất phân phối

$$x.(y+z) = x.y + x.z$$

$$x+(y.z) = (x+y).(x+z)$$

- * Luật De Morgan

$$\overline{x_1 + x_2 + \dots + x_n} = \overline{x_1} . \overline{x_2} . \dots . \overline{x_n}$$

$$\overline{x_1 . x_2 . \dots . x_n} = \overline{x_1} + \overline{x_2} + \dots + \overline{x_n}$$

Phủ định của một tổng các biến logic bằng tổng các phủ định các biến logic.

Phủ định của một tích các biến logic bằng tích các phủ định của các biến logic

- * Tính đối ngẫu

Nếu trong một hệ thức ta thay phép cộng bằng phép nhân, phép nhân bằng phép cộng, thay 0 bằng 1 và 1 bằng 0 thì sẽ thu được một hệ thức mới gọi là hệ thức đối ngẫu. Nếu hệ thức ban đầu đúng thì hệ thức đối ngẫu của nó cũng đúng.

b. Một số hệ thức logic cơ bản

- $x+0 = x$; $x.1 = x$
- $x.0 = 0$; $x+1 = 1$
- $x+x = x$; $x.x = x$
- $x + \bar{x} = 1$; $x.\bar{x} = 0$
- $x+xy = x$; $x.(x+y) = x$
- $xy + x\bar{y} = x$; $(x + y)(x + \bar{y}) = x$

2.1.3 Các cách biểu diễn hàm logic tổ hợp

a. Biểu diễn bằng bảng chân lý

Bảng chân lý là bảng liệt kê tất cả các tổ hợp các giá trị của các biến và giá trị tương ứng của hàm số với mỗi tổ hợp biến đó. Như vậy, với một hàm n biến thì ta sẽ có 2^n tổ hợp biến. Bảng 2.4 cho ta một ví dụ về bảng chân lý với hàm 3 biến

Bảng 2.4 Bảng chân lý biểu diễn hàm 3 biến

Giá trị thập phân của tổ hợp biến	x1	x2	x3	f(x1,x2,x3)
0	0	0	0	1
1	0	0	1	0
2	0	1	0	“x”
3	0	1	1	“x”
4	1	0	0	0
5	1	0	1	1
6	1	1	0	“x”
7	1	1	1	1

Ghi chú:

Những chỗ đánh dấu “x” là giá trị hàm không xác định (có thể là 0 hoặc 1), có nghĩa là giá trị hàm ứng với các tổ hợp giá trị biến đầu vào không ảnh hưởng đến vai trò của hàm logic này.

Ưu điểm của cách biểu diễn này là dễ nhìn, ít nhầm lẫn vì tất cả các khả năng có thể của hàm số đã được liệt kê ra hết. Nhược điểm của phương pháp này là cồng kềnh, đặc biệt là khi biến số lớn.

b. Biểu diễn bằng biểu thức đại số

Hàm logic có thể được biểu diễn bằng biểu thức đại số sử dụng các phép toán cộng và nhân logic. Bằng cách biểu diễn hàm logic bằng biểu thức đại số, có thể dùng các thiết bị logic để thực hiện hàm logic một cách dàng.

Một hàm logic n biến bất kỳ bao giờ cũng có thể biểu diễn thành tổng chuẩn đầy đủ và tích chuẩn đầy đủ.

- * Cách viết hàm dưới dạng tổng chuẩn đầy đủ
 - + Chỉ quan tâm đến tổ hợp các giá trị của biến làm cho hàm có giá trị 1. Số lần hàm bằng 1 chính bằng số tích của các tổ hợp biến này.
 - + Trong mỗi tích ứng với một tổ hợp biến làm cho hàm có giá trị bằng 1, các biến có giá trị 1 thì giữ nguyên, các biến có giá trị 0 thì được lấy giá trị đảo
 - + Hàm tổng chuẩn đầy đủ sẽ là tổng các tích đó

Ví dụ: Cho hàm logic 2 biến như bảng 2.5

Bảng 2.5: Hàm logic 2 biến số

x1	x2	y=f(x1,x2)
0	0	1
0	1	0
1	0	0
1	1	1

Ta nhận thấy có 2 tổ hợp giá trị biến làm cho hàm có giá trị 1 là $(x1,x2) = (0,0)$ và $(x1,x2) = (1,1)$. Với tổ hợp $(x1,x2) = (0,0)$, vì hai biến đều nhận giá trị 0 nên tích các biến tương ứng sẽ là $\overline{x1}.\overline{x2}$. Với tổ hợp $(x1,x2) = (1,1)$, vì hai biến đều nhận giá trị 1 nên tích các biến tương ứng sẽ là $x1.x2$. Do đó hàm logic có thể được biểu diễn dưới dạng tổng chuẩn đầy đủ như sau

$$y = f(x1,x2) = \overline{x1}.\overline{x2} + x1.x2$$

- * Cách viết hàm dưới dạng tích chuẩn đầy đủ
 - + Chỉ quan tâm đến tổ hợp các giá trị của biến làm cho hàm có giá trị 0. Số lần hàm bằng 0 chính bằng số tổng của các tổ hợp biến này.
 - + Trong mỗi tổng ứng với một tổ hợp biến làm cho hàm có giá trị bằng 0, các biến có giá trị 0 thì giữ nguyên, các biến có giá trị 1 thì được lấy giá trị đảo
 - + Hàm tích chuẩn đầy đủ sẽ là tích các tổng đó

Ví dụ: Cho hàm logic 2 biến như bảng 2.5

Ta nhận thấy có 2 tổ hợp giá trị biến làm cho hàm có giá trị 0 là $(x1,x2) = (0,1)$ và $(x1,x2) = (1,0)$. Với tổ hợp $(x1,x2) = (0,1)$, vì biến x1 nhận giá trị 0 nên được giữ nguyên, biến x2 nhận giá trị 1 nên phải lấy giá trị đảo, tổng các biến tương ứng sẽ là $x1 + \overline{x2}$. Với tổ hợp $(x1,x2)=(1,0)$, vì biến x1 nhận giá trị 1 nên phải lấy giá trị đảo, biến x2 nhận giá trị 0 nên được giữ nguyên, tổng các biến tương ứng sẽ là $\overline{x1}+x2$. Do đó hàm logic có thể được biểu diễn dưới dạng tích chuẩn đầy đủ như sau

$$y=f(x1, x2)=(x1 + \overline{x2})(\overline{x1}+x2)$$

Để viết các hàm logic dưới dạng ngắn gọn, người ta có thể viết các hàm logic dưới dạng như sau:

+ Dạng tổng chuẩn đầy đủ

Hàm logic ở bảng 2.4 có thể viết như sau:

$$f(x_1, x_2, x_3) = \Sigma(0, 5, 7) \text{ với } N = 2, 3, 6$$

Trong đó các số 0, 5, 7 là giá trị thập phân của tổ hợp biến (theo thứ tự $x_1 x_2 x_3$) làm cho hàm có giá trị bằng 1; và 2, 3, 6 là các giá trị thập phân của tổ hợp biến làm cho hàm có giá trị không xác định.

+ Dạng tích chuẩn đầy đủ

Hàm logic ở bảng 2.4 có thể viết như sau:

$$f(x_1, x_2, x_3) = \Pi(1, 4) \text{ với } N = 2, 3, 6$$

Trong đó các số 1, 4 là giá trị thập phân của tổ hợp biến (theo thứ tự $x_1 x_2 x_3$) làm cho hàm có giá trị bằng 0; và 2, 3, 6 là các giá trị thập phân của tổ hợp biến làm cho hàm có giá trị không xác định.

c. Biểu diễn bằng bảng Các ô

Nguyên tắc xây dựng bảng Các ô như sau:

- + Để biểu diễn một hàm logic có n biến số, cần lập một bảng có 2^n ô, mỗi ô tương ứng với một tổ hợp biến.
- + Các ô cạnh nhau hoặc đối xứng nhau chỉ cho phép khác nhau về 1 giá trị của một biến
- + Trong các ô ghi các giá trị của hàm tương ứng với tổ hợp biến ứng với ô đó

Ví dụ: Bảng Các ô biểu diễn hàm logic trong bảng 1.5

Bảng 2.6 Bảng Các ô biểu diễn hàm logic trong bảng 2.5

x_2	0	1
x_1		
0	1	0
1	0	1

Bảng Các ô biểu diễn hàm logic trong bảng 2.4

$x_2 x_3$	00	01	11	10
x_1				
0	1	0	“X”	“X”
1	0	1	1	“X”

Chú ý: với các ô ứng với các tổ hợp biến làm cho hàm có giá trị không xác định thì có thể điền “X” hoặc bỏ trống

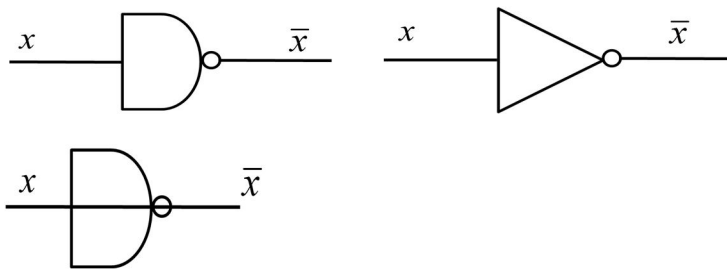
Bảng Các ô cho hàm 5 biến

$\begin{matrix} \text{x3x4x5} \\ \text{x1x2} \end{matrix}$	000	001	011	010	110	111	101	100

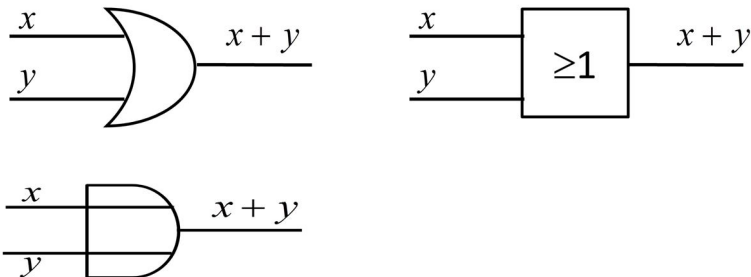
2.1.4 Các ký hiệu mạch logic

Để mô tả các mạch logic thực hiện các hàm logic, người ta thường dùng các ký hiệu của các phần tử logic thực hiện các phép toán logic cơ bản. Các ký hiệu đó bao gồm:

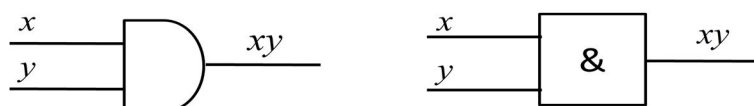
a. Phép phủ định (nghịch đảo) NOT



b. Phép cộng logic OR

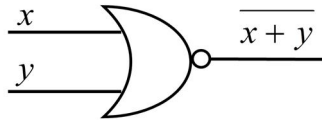


c. Phép nhân logic AND

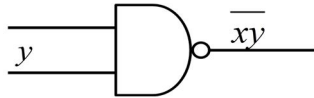


d. Phép NOR và NAND

- Phép NOR










- Phép NAND



2.1.5 Mạch role-tiếp điểm

Mạch logic cũng có thể được biểu diễn bằng sơ đồ role- tiếp điểm. Một số ký hiệu được sử dụng trong mạch role –tiếp điểm bao gồm:

- Nút ấn thường mở: . Bình thường khi không bị tác động thì hai cực của nút ấn được cách ly. Nếu ấn nút thì hai cực của nút ấn sẽ được nối với nhau và có thể cho phép dòng điện chạy qua.
- Nút ấn thường đóng: . Bình thường khi không bị tác động thì hai cực của nút ấn được nối với nhau và có thể cho phép dòng điện chạy qua. Nếu ấn nút thì hai cực của nút ấn sẽ được cách ly và không cho dòng điện chạy qua.
- Công tắc hành trình thường mở: . Bình thường khi không bị tác động thì hai cực của công tắc hành trình được cách ly. Nếu bị tác động thì hai cực của công tắc hành trình sẽ được nối với nhau và có thể cho phép dòng điện chạy qua.
- Công tắc hành trình thường đóng: . Bình thường khi không bị tác động thì hai cực của công tắc hành trình được nối với nhau và có thể cho phép dòng điện chạy qua. Nếu bị tác động thì hai cực của công tắc hành trình sẽ được cách ly và không cho dòng điện chạy qua.
- Rơ le gồm 2 phần cuộn dây , ký hiệu là một hình chữ nhật đứng  và các tiếp điểm. Tiếp điểm có 2 loại cơ bản là tiếp điểm thường mở và thường đóng. Tiếp điểm thường mở, ký hiệu . Bình thường khi cuộn dây không có điện thì 2 điểm cực của tiếp điểm bị cách ly và không cho phép dòng điện được chạy qua. Nếu cấp điện cho cuộn dây thì cuộn dây này sẽ làm cho hai cực của tiếp điểm được nối với nhau và có thể cho phép dòng điện chạy qua. Tiếp điểm thường đóng, ký hiệu . Bình thường khi cuộn dây không có điện thì 2 điểm cực của tiếp điểm được nối với nhau và có thể cho phép dòng điện chạy qua. Nếu cấp điện cho cuộn dây thì cuộn dây này sẽ làm 2 điểm cực của tiếp điểm bị cách ly và không cho phép dòng điện được chạy qua.

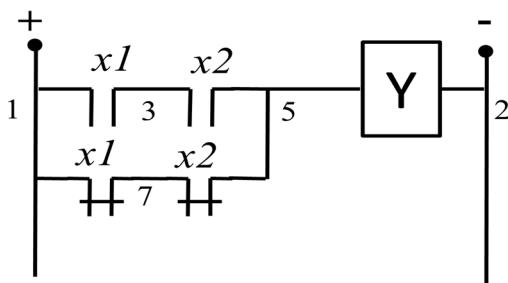
Mạch logic có thể được biểu diễn bằng sơ đồ rơ le tiếp điểm như sau:

- Sơ đồ gồm 2 dây thể hiện dây nguồn cấp cho mạch
- Tùy thuộc vào thiết bị vật lý tương ứng với các biến logic mà ta có thể biểu diễn các biến dưới dạng nút ấn, công tắc hành trình hay các tiếp điểm
- Các biến ở trạng thái bình thường được biểu diễn bằng các phần tử ở trạng thái

- thường mở
- Các biến ở trạng thái đảo thì được biểu diễn bằng các phần tử ở trạng thái thường đóng.
- Đầu ra được biểu diễn bằng cuộn dây của role. Cuộn dây có điện thì có nghĩa là đầu ra nhận giá trị 1, không có điện thì nhận giá trị 0. Khi cuộn dây thay đổi trạng thái có điện hoặc không có điện thì các tiếp điểm (rơ le) tương ứng cũng thay đổi trạng thái theo (coi là tức thời)
- Phép cộng logic được biểu diễn bằng cách nối hai phần tử song song với nhau
- Phép nhân logic được biểu diễn bằng cách nối hai phần tử nối tiếp với nhau.
- Đầu ra được biểu diễn bằng cách nối cuộn dây với tổ hợp kết nối các phần tử đầu vào.

Ví dụ mạch role-tiếp điểm thực hiện hàm logic

$$y=f(x1,x2)= \overline{x1}.\overline{x2} + x1.x2$$



Khi $x1 = x2 = 0$ nghĩa là cuộn dây $x1$ và $x2$ (không vẽ ở đây) không có điện, do đó các tiếp điểm ở trạng thái như hình vẽ. Khi đó dòng điện có thể đi từ 1-7-5-2, cuộn dây Y có điện, nghĩa là $y = f(x1,x2) = 1$.

Khi $x1=x2 = 1$ nghĩa là cuộn dây $x1$ và $x2$ có điện. Khi đó tiếp điểm $x1$ (1-3), $x2$ (3-5) đóng lại, các tiếp điểm $x1$ (1-7), $x2$ (7-5) mở ra. Khi đó dòng điện có thể đi từ 1-3-5-2, cuộn dây Y có điện, nghĩa là $y = f(x1,x2) = 1$.

Khi $x1 = 0, x2 = 1$ nghĩa là cuộn dây $x1$ không có điện và cuộn dây $x2$ có điện. Khi đó các tiếp điểm $x1$ giữ nguyên trạng thái như hình vẽ. Tiếp điểm $x2$ (3-5) đóng lại, $x2$ (7-5) mở ra. Cuộn dây Y bị cách ly khỏi nguồn điện, nghĩa là $y = f(x1,x2) = 0$.

Khi $x1 = 1, x2 = 0$ nghĩa là cuộn dây $x1$ có điện và cuộn dây $x2$ không có điện. Khi đó các tiếp điểm $x2$ giữ nguyên trạng thái như hình vẽ. Tiếp điểm $x1$ (1-3) đóng lại, $x2$ (1-7) mở ra. Cuộn dây Y bị cách ly khỏi nguồn điện, nghĩa là $y = f(x1,x2) = 0$.

2.2 Định nghĩa và phân loại

Mạch tổ hợp là mạch mà trạng thái đầu ra của mạch chỉ phụ thuộc vào tổ hợp các trạng thái đầu vào mà không phụ thuộc vào trình tự tác động của các đầu vào.

Theo quan điểm điều khiển thì mạch tổ hợp là mạch hở, hệ thống không có phản hồi,

nghĩa là trạng thái đóng mở của các phần tử trong mạch hoàn toàn không bị ảnh hưởng của trạng thái tín hiệu đầu ra.

Về mặt toán học, giả thiết một mạch tổ hợp có n đầu vào x_i ($i=1, \dots, n$) và m đầu ra y_j ($j=1, \dots, m$), ta ký hiệu

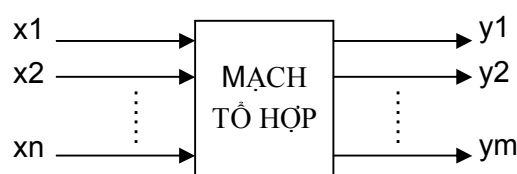
$X = \{x_1, x_2, \dots, x_n\}$ là tập các tín hiệu vào

$Y = \{y_1, y_2, \dots, y_m\}$ là tập các tín hiệu ra

Thì mạch tổ hợp được biểu diễn bởi m phương trình đại số Boole như sau:

$$y_j = f_j(x_1, x_2, \dots, x_n) \text{ với } j = 1, \dots, m$$

Có thể biểu diễn mô hình toán học của mạch tổ hợp theo sơ đồ khối như ở hình dưới đây



2.3 Tổng hợp mạch logic tổ hợp

Việc tổng hợp mạch logic tổ hợp thực chất là thiết kế mạch tổ hợp. Nhiệm vụ chính ở đây là thiết kế được mạch tổ hợp thỏa mãn yêu cầu kỹ thuật nhưng mạch phải tối giản. Bài toán tổng hợp là bài toán phức tạp, vì ngoài các yêu cầu về chức năng logic, việc tổng hợp mạch còn phụ thuộc vào việc sử dụng các phần tử. Với mỗi loại phần tử được sử dụng thì ngoài nguyên tắc chung về mạch logic còn đòi hỏi phải bổ sung những nguyên tắc riêng lúc tổng hợp và thiết kế mạch. Tuy nhiên, phạm vi được đề cập ở đây chỉ tập trung vào việc đáp ứng các chức năng logic và tối thiểu hóa mạch logic.

Như đã đề cập trong phần biểu diễn hàm logic trong chương I, ta có thể biểu diễn hàm logic dưới dạng biểu thức đại số từ bảng chân lý. Từ biểu thức đại số đó ta có thể sử dụng các phần tử logic để thực hiện mạch logic. Tuy nhiên biểu thức đại số mà ta nhận được trong chương I thường không tối thiểu, nghĩa là số lượng biến và phép toán là không tối ưu. Do vậy số lượng phần tử logic cần thiết để thực hiện hàm logic sẽ lớn, gây ra tăng chi phí và phức tạp trong việc gỡ rối. Ở phần dưới đây sẽ đề cập đến một số phương pháp tối thiểu hóa hàm logic tổ hợp.

- Phương pháp đại số

- Phương pháp ma trận các nô
- Phương pháp Quine Mc Clusky

2.3.1 Phương pháp đại số

Phương pháp đại số là phương pháp dùng các biến đổi đại số để rút gọn hàm logic. Một số biểu thức đại số thường hay được sử dụng bao gồm:

- | | | |
|-----------------------|---|----------------------------|
| • $x+0 = x$ | ; | $x.1 = x$ |
| • $x.0 = 0$ | ; | $x+1 = 1$ |
| • $x+x = x$ | ; | $x.x = x$ |
| • $x + \bar{x} = 1$ | ; | $x.\bar{x} = 0$ |
| • $x+xy = x$ | ; | $x.(x+y) = x$ |
| • $xy + x\bar{y} = x$ | ; | $(x + y)(x + \bar{y}) = x$ |

Ví dụ:

Rút gọn hàm $y = f(x_1, x_2) = x_1\bar{x}_2 + x_1x_2 + \bar{x}_1x_2$

$$\begin{aligned}
 f(x_1, x_2) &= x_1\bar{x}_2 + x_1x_2 + \bar{x}_1x_2 \\
 &= (x_1\bar{x}_2 + x_1x_2) + (\bar{x}_1x_2) \\
 &= x_1 + x_2
 \end{aligned}$$

Như vậy từ một hàm logic 2 biến với 2 phép cộng và 3 phép nhân logic, ta có thể rút gọn thành một hàm logic tương đương với chỉ một phép cộng logic. Cũng có thể thấy rằng biểu diễn này là tối thiểu.

Ưu điểm:

Phương pháp đại số có ưu điểm là khá trực quan.

Nhược điểm:

Tuy nhiên nhược điểm cơ bản của phương pháp này là rất khó có thể đánh giá được kết quả thu được đã là tối ưu hay chưa.

2.3.2 Phương pháp ma trận các nô

Phương pháp bảng Các nô biểu diễn hàm logic dùng bảng Các nô sau đó dùng các tính chất của bảng Các nô để nhóm các tổ hợp biến thích hợp lại với nhau để rút gọn hàm logic. Các bước để thực hiện phương pháp bảng Các nô như sau:

- Bước 1:

Biểu diễn hàm đã cho dưới dạng bảng Các nô

- Bước 2:

Nhóm các ô có giá trị 1 hoặc “x” (không xác định) cạnh nhau hoặc đối xứng nhau thành các vòng. Việc nhóm này phải thỏa mãn các điều kiện sau:

- + Số ô trong một vòng có dạng 2^m ($1 \leq m \in \mathbb{N}$) với m lớn nhất có thể
- + Các vòng có thể giao nhau nhưng không được trùm lên nhau
- + Các vòng phải phủ hết các ô có giá trị 1 (không phải phủ hết các ô không xác định).
- + Số vòng phải là ít nhất

- Bước 3:

Mỗi vòng sẽ tương ứng với tích các biến mà giá trị các biến đó là không thay đổi trong các ô trong vòng đó. Hàm rút gọn bằng tổng các tích tương ứng với các vòng

Việc nhóm các ô có giá trị 1 cạnh nhau hoặc đối xứng nhau thành một nhóm thực chất chính là áp dụng tính chất $xy + x\bar{y} = x$ của phép toán logic. Theo cách sắp xếp của bảng các nô thì hai ô cạnh nhau hoặc đối xứng nhau chỉ khác nhau một giá trị biến, ví dụ là y. Nên khi ghép hai ô đó lại với nhau thì sẽ rút gọn được biến y, và biểu thức tương ứng với nhóm 2 ô đó sẽ chỉ còn lại tổ hợp các biến không đổi trị, ở đây là x. Tính chất này có thể mở rộng ra cho 2^m ô cạnh nhau hoặc đối xứng nhau (qua trục phân chia vùng nhận giá trị 0 và vùng nhận giá trị 1 của 1 biến). Chú ý rằng trong nhóm 2^m ô thì sẽ có đúng m biến bị đổi trị. Giá trị của m phải là tối đa và số vòng phải là tối thiểu để đảm bảo kết quả thu được là tối ưu.

Ví dụ 1:

Tối thiểu hóa hàm logic $f(x_1, x_2, x_3) = \Sigma(0, 2, 5, 6, 7)$

Biểu diễn hàm logic bằng bảng Các nô như dưới đây

$x_2 x_3$	00	01	11	10
x_1				
0	1	0	0	1
1	0	1	1	1

Có thể nhóm các ô với nhau như sau:

- Ô $\overline{x_1}.\overline{x_2}.\overline{x_3}$ (000) và $\overline{x_1}.x_2.\overline{x_3}$ (010), 2 ô này chỉ có biến x_2 là đổi trị nên tổ hợp rút gọn tương ứng còn $\overline{x_1}.\overline{x_3}$
- Ô $x_1.\overline{x_2}.x_3$ (101) và $x_1.x_2.x_3$ (111), 2 ô này chỉ có biến x_2 là đổi trị nên tổ hợp rút gọn tương ứng còn $x_1.x_3$
- Ô $x_1.x_2.x_3$ (111) và $x_1.x_2.\overline{x_3}$ (110), 2 ô này chỉ có biến x_3 là đổi trị nên tổ hợp rút gọn tương ứng còn là $x_1.x_2$

Như vậy hàm rút gọn là $f(x_1, x_2, x_3) = \overline{x_1}.\overline{x_3} + x_1.x_3 + x_1.x_2$

Ta cũng có thể nhóm các ô theo cách khác như sau

x_2x_3	00	01	11	10
x_1				
0	1	0	0	1
1	0	1	1	1

- Ô $\overline{x_1}.\overline{x_2}.\overline{x_3}$ (000) và $\overline{x_1}.x_2.\overline{x_3}$ (010), 2 ô này chỉ có biến x_2 là đổi trị nên tổ hợp rút gọn tương ứng còn $\overline{x_1}.\overline{x_3}$
- Ô $x_1.\overline{x_2}.x_3$ (101) và $x_1.x_2.x_3$ (111), 2 ô này chỉ có biến x_2 là đổi trị nên tổ hợp rút gọn tương ứng còn $x_1.x_3$
- Ô $\overline{x_1}.x_2.\overline{x_3}$ (010) và $x_1.x_2.\overline{x_3}$ (110), 2 ô này chỉ có biến x_1 là đổi trị nên tổ hợp rút gọn tương ứng còn là $x_2.\overline{x_3}$

Như vậy hàm rút gọn là $f(x_1, x_2, x_3) = \overline{x_1}.\overline{x_3} + x_1.x_3 + x_2.\overline{x_3}$

Ta nhận thấy rằng có thể có nhiều kết quả rút gọn khác nhau và đều là kết quả tối ưu. Điều này hoàn toàn hợp lý vì tuy cách biểu diễn có khác nhau nhưng giá trị hàm số ứng với cùng 1 tổ hợp biến là như nhau và số phép toán để thực hiện hai biểu diễn khác nhau này là như nhau (3 phép nhân logic và 2 phép cộng logic) và là tối thiểu

Ví dụ 2:

Tối thiểu hóa hàm logic $f(x_1, x_2, x_3) = \Sigma(0, 5, 7)$ với $N = 2, 3, 6$

Biểu diễn hàm logic bằng bảng Các ô như dưới đây

x_2x_3	00	01	11	10
x_1				

0	<u>1</u>	0	"x"	<u>"x"</u>
1	0	<u>1</u>	<u>1</u>	"x"

Trong ví dụ này có cả các tổ hợp không xác định nên khi nhóm các ô lại với nhau ta có thể kết hợp các ô có giá trị 1 và các ô không xác định để được nhóm có số ô là tối đa. Chú ý rằng không nhất thiết các vòng phải phủ hết các ô không xác định. Trong ví dụ này có thể nhóm các ô như sau:

- Hai ô $\overline{x_1}.\overline{x_2}.\overline{x_3}$ (000) và $\overline{x_1}.x_2.\overline{x_3}$ (010), 2 ô này chỉ có biến x_2 là đổi trị nên tổ hợp rút gọn tương ứng còn $\overline{x_1}.x_3$
- Hai ô $x_1.\overline{x_2}.x_3$ (101) và $x_1.x_2.x_3$ (111), 2 ô này chỉ có biến x_2 là đổi trị nên tổ hợp rút gọn tương ứng còn $x_1.x_3$

Sau khi nhóm như trên ta thấy các ô có giá trị 1 đã được nhóm hết với số vòng là ít nhất. Một số ô không xác định không nhất thiết phải thuộc một vòng nào đó. Như vậy hàm rút gọn là $f(x_1, x_2, x_3) = \overline{x_1}.\overline{x_3} + x_1.x_3$

Phương pháp bảng Các nô khá trực quan, tuy nhiên khi số biến tăng lên thì việc nhóm các ô trở nên phức tạp và việc chọn số tối đa trong một nhóm cũng như số nhóm là tối thiểu trở nên khó khăn. Thêm vào đó, việc thực thi phương pháp này trên các công cụ tính toán rất khó khăn.

2.3.3 Phương pháp Quine Mc Clusky

Cũng như phương pháp bảng Các nô, phương pháp Quine Mc Clusky cũng áp dụng tính chất $xy + x\overline{y} = x$ của phép toán logic. Tuy nhiên phương pháp này xem xét lần lượt khả năng rút gọn của tất cả các kết hợp của hai tổ hợp biến ban đầu và cả sau từng bước rút gọn. Bằng cách này việc rút gọn được trình bày dưới dạng thuật toán và có thể lập trình cho máy tính thực hiện.

Các bước thực hiện thuật toán Quine Mc Clusky như sau:

- Bước 1: Ghi các tổ hợp biến làm cho hàm có giá trị bằng 1 theo mã nhị phân. Các biến bị đảo thì ghi thành 0, các biến được giữ nguyên thì ghi thành 1. Ví dụ $x_1\overline{x_2}x_3$ sẽ ghi thành 101
- Bước 2: Nhóm các tổ hợp biến theo số chữ số 1 trong biểu diễn nhị phân của tổ hợp biến. Đặt tên nhóm i là nhóm có i chữ số 1 trong biểu diễn nhị phân. Ghi các tổ hợp biến này trong 1 cột.
- Bước 3: Xem xét khả năng ghép của mỗi tổ hợp của nhóm thứ i với từng tổ hợp của nhóm thứ $i+1$ trong cùng một cột. Hai tổ hợp chỉ được ghép với nhau khi biểu diễn nhị phân của chúng chỉ khác nhau 1 bit ở cùng 1 vị trí. Khi ghép 2 tổ hợp với nhau ta sẽ ghi sang cột bên cạnh tổ hợp mới hình thành bằng cách giữ nguyên các phần giống nhau và thay phần khác nhau bằng dấu gạch ngang (-). Sau khi đã xem xét tất cả các khả năng ghép của toàn bộ các tổ hợp, đánh dấu sao (*) vào các tổ

hợp biến đã tham gia ghép và dấu v (✓) vào các tổ hợp không thể ghép.

- Bước 4: Lập lại bước trên với cột vừa mới hình thành cho đến khi không kết hợp được nữa. Chú ý, hai tổ hợp có dấu (-) chỉ ghép được với nhau khi mà chỉ có 1 vị trí bit khác nhau và bit đó phải là 0 và 1, không thể là “-”.
- Bước 5: Lập bảng phủ tối thiểu để chọn số tổ hợp không thể ghép tối thiểu có thể phủ hết được số tổ hợp ban đầu. Bảng phủ tối thiểu có các cột tương ứng với số tổ hợp nguyên gốc ban đầu. Các hàng tương ứng với số tổ hợp không thể ghép được nữa. Trên một hàng, nếu tổ hợp ứng với hàng đó có thể “phủ” tổ hợp ứng với cột (nếu thay ở tổ hợp rút gọn dấu “-” bằng số 0 hoặc 1 thì sẽ được tổ hợp nguyên gốc ứng với cột) thì ô ứng với cột đó đánh dấu “x”. Từ bảng phủ đã đánh dấu sẽ chọn được số tổ hợp tối thiểu.
- Bước 6: Sau khi đã tìm được số tổ hợp tối thiểu thì chuyển các tổ hợp mã nhị phân thành tổ hợp biến tương ứng (các biến ứng với vị trí có dấu “-” sẽ bị rút gọn trong biểu diễn). Ví dụ cho hàm 3 biến $f(x_1, x_2, x_3)$ thì 1-1 ứng với tổ hợp x_1x_3 . Vị trí x_2 tương ứng với dấu “-” nên bị rút gọn trong biểu diễn.

Ví dụ: Rút gọn hàm $f(x_1, x_2, x_3) = \Sigma(0, 1, 4, 5, 7)$

Ta có bảng thể hiện thuật từ bước 1 đến 3 như sau:

Nhóm	Giá trị thập phân tương ứng	Tổ hợp I	Tổ hợp II	Tổ hợp III
0	0	000*	00-*(0&1) -00* (0&4)	-0- ✓(0&1;4&5) (0&4;1&5)
1	1	001*	-01* (1&5)	
	4	100*	10-* (4&5)	
2	5	101*	1-1✓ (5&7)	
3	7	111*		

Tổ hợp 000(0) và 001(1) chỉ khác nhau 1 bit cuối cùng nên có thể ghép chúng được với nhau và tổ hợp mới hình thành là 00-. Tương tự với các tổ hợp khác ta thu được cột tổ hợp II. Trong tổ hợp II ta thấy 00- và 10- có cũng chỉ khác nhau bit đầu tiên nên có thể ghép được với nhau và được tổ hợp -0-. Tương tự với các tổ hợp khác. Riêng tổ hợp 1-1 không ghép được với tổ hợp nào nên ta đánh dấu (✓). Các tổ hợp đã tham gia ghép được đánh dấu (*). Đến cột tổ hợp III thì không ghép được nữa nên quá trình ghép dừng ở đây.

Tiếp theo là lập bảng phủ như dưới đây

	000	001	100	101	111
-0-	X	x	X	x	
1-1				x	x

Ta thấy 2 tổ hợp -0- và 1-1 đã là tối thiểu.

Kết luận hàm tối thiểu sẽ là $f(x_1, x_2, x_3) = \overline{x_2} + x_1.x_3$

Ví dụ 2:

Rút gọn hàm $f(x_1, x_2, x_3, x_4) = \Sigma(0, 4, 5, 6, 7, 8, 9, 10, 13, 15)$

Nhóm	Giá trị thập phân tương ứng	Tổ hợp I	Tổ hợp II	Tổ hợp III
0	0	0000*	0-00√(0&4) -000√(0&8)	
1	4	0100*	010-*(4&5) 01-0*(4&6)	01--√(4&5;6&7) (4&6;5&7)
	8	1000*	100-√(8&9) 10-0√(8&10)	
2	5	0101*	01-1*(5&7)	-1-1√(5&7;13&15) (5&13;7&15)
	6	0110*	011-*(6&7)	
	9	1001*	-101*(5&13)	
	10	1010*	1-01√(9&13)	
3	7	0111*	-111*(7&15)	
	13	1101*	11-1*(13&15)	
4	15	1111*		

Bảng phủ:

	0000	0100	0101	0110	0111	1000	1001	1010	1101	1111
0-00	x	X								
-000	x					x				
100-						x	x			
10-0						x		x		
1-01							x		x	
01--		X	X	X	x					
-1-1			X		x				x	x

Có một số gợi ý để chọn ra các tổ hợp rút gọn từ bảng phủ như sau:

Trong các cột, tìm các cột chỉ có 1 ô có dấu “x”, ở đây là 3 cột 0110, 1010 và 1111. Ta thấy rằng các tổ hợp ứng với các dấu “x” tìm ở trên phải được chọn vì chỉ có những tổ hợp rút gọn đó mới phủ được các tổ hợp tương ứng với các cột tìm thấy. Như vậy ở đây ta chọn 3 tổ hợp rút gọn là 01-- (ứng với 0110), 10-0 (ứng với 1010) và -1-1 (ứng với 1111). Ba tổ hợp rút gọn đã phủ hết các tổ hợp 0100, 0101, 0110, 0111, 1000, 1010, và 1111. Ta còn các tổ hợp nguyên gốc chưa bị phủ gồm có 0000, 1001 và 4 tổ hợp rút gọn chưa dùng đến. Đến đây việc chọn tổ hợp rút gọn để phủ 3 tổ hợp nguyên gốc trở nên đơn giản hơn nhiều. Ví dụ ta có thể chọn 0-00, khi đó ô 0000 bị phủ, và tổ hợp -000 là không cần thiết, vì các tổ hợp mà tổ hợp -000 phủ đã bị phủ hết. Tiếp theo ta chọn tổ hợp rút gọn 100- để phủ tổ hợp nguyên gốc còn lại. Như vậy các tổ hợp nguyên gốc đã được phủ hết. Cuối cùng ta thu được kết quả

$$f(x_1, x_2, x_3, x_4) = \overline{x_1} \cdot \overline{x_3} \cdot \overline{x_4} + x_1 \cdot \overline{x_2} \cdot \overline{x_3} + x_1 \cdot \overline{x_2} \cdot \overline{x_4} + \overline{x_1} \cdot x_2 + x_2 \cdot x_4$$

CHƯƠNG 3: MẠCH LOGIC TUẦN TỰ

3.1 Khái niệm cơ bản về mạch logic tuần tự

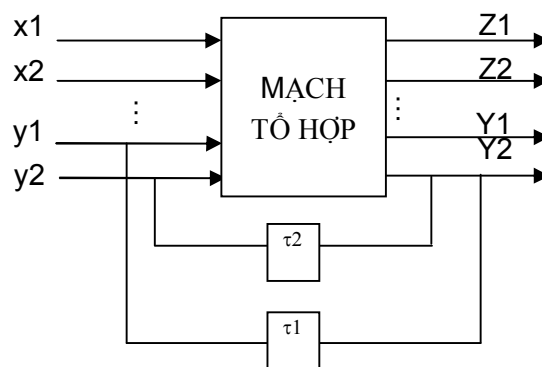
3.1 Khái niệm cơ bản về mạch logic tuần tự

3.1.1 Giới thiệu chung

Mạch logic tuần tự là mạch mà trong đó trạng thái của tín hiệu ra không những phụ thuộc vào tín hiệu vào mà còn phụ thuộc vào cả trình tự tác động của tín hiệu vào, nghĩa là phụ thuộc vào các trạng thái trước đó của mạch hay là mạch có nhớ các trạng thái. Như vậy, về mặt thiết bị thì ở mạch tuần tự không những chỉ có các phần tử đóng mở logic mà còn có các phần tử nhớ.

Sơ đồ cấu trúc cơ bản của mạch trình tự như ở hình 3.1. Nét đặc trưng ở đây là mạch có phản hồi thể hiện qua các biến nội bộ (Y_1, Y_2 và y_1, y_2)

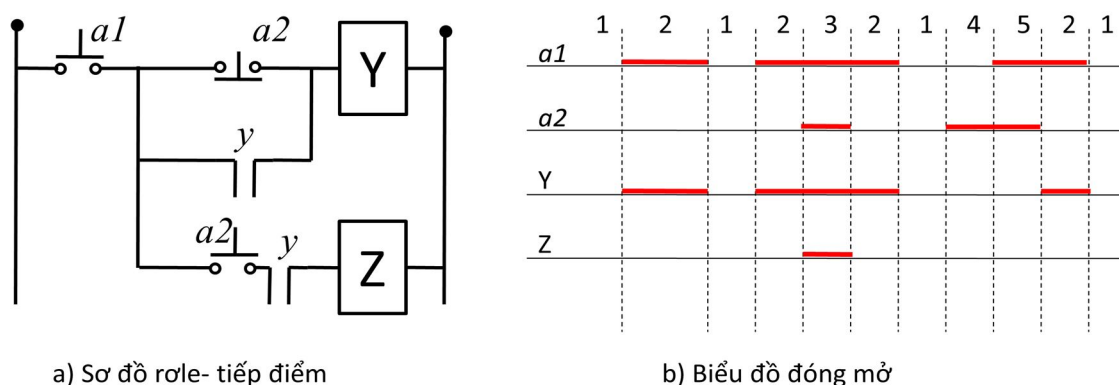
Hoạt động trình tự của mạch được thể hiện ở sự thay đổi của biến nội bộ Y . Trong quá trình làm việc, do sự thay đổi của các tín hiệu vào X (X_1, X_2, \dots) sẽ dẫn đến thay đổi các tín hiệu ra Z (Z_1, Z_2, \dots) và cả tín hiệu nội bộ Y (Y_1, Y_2, \dots). Sự thay đổi của biến Y (Y_1, Y_2, \dots) sẽ dẫn đến thay đổi biến y (y_1, y_2, \dots) sau thời gian (τ_1, τ_2, \dots). Sự thay đổi của các biến y (y_1, y_2, \dots) lại có thể dẫn đến thay đổi các tín hiệu ra Z , kể cả Y , rồi lại sự thay đổi của Y lại dẫn đến sự thay đổi của $y \dots$ Quá trình nếu cứ tiếp tục lâu dài như vậy sẽ làm cho hệ mất ổn định, nghĩa là mạch không làm việc được. Yêu cầu đặt ra là phải làm cho mạch ổn định, nghĩa là khi mạch tuần tự có sự thay đổi của tín hiệu vào sẽ chuyển từ một trạng thái ổn định này sang trạng thái ổn định khác và trải qua một số trạng thái trung gian không ổn định.



Hình 3.1 Cấu trúc cơ bản của mạch trình tự

3.1.2 Mô tả hoạt động của mạch tuần tự

Một trong những công cụ để mô tả hoạt động của mạch tuần tự là biểu đồ đóng mở theo thời gian. Ví dụ ta có sơ đồ mạch rơle-tiếp điểm (hình 3.2a) và biểu đồ đóng mở (hình 3.2b) dưới đây



Hình 3.2: Sơ đồ mạch rơ le tiếp điểm và biểu đồ đóng mở

Trong biểu đồ đóng mở, chiều ngang biểu thị thời gian. Các số biểu thị tên trạng thái của hệ thống. Nét đậm biểu thị tại thời điểm đó giá trị biến là 1 (nút bị ấn hoặc cuộn dây có điện). Nét mảnh biểu thị giá trị 0 (nút ấn ở trạng thái không bị tác động, cuộn dây chưa có điện).

Trạng thái 1 là trạng thái ban đầu, tức là chưa tác động gì vào các biến đầu vào. Khi đó, các nút chưa bị tác động, $a1=a2=0$ và các cuộn dây chưa có điện, $Y=Z=0$.

Nếu ta chỉ ấn nút $a1$, $a1=1$, thì cuộn dây Y có điện, $Y=1$ (trạng thái 2). Nếu ta nhả tay ra không ấn nút $a1$ nữa, $a1=0$, cuộn dây Y mất điện, $Y=0$, hệ thống quay về trạng thái 1.

Tiếp theo, nếu ta chỉ ấn nút $a1$ thì hệ thống ở trạng thái 2. Khi đó, cuộn dây Y có điện làm các tiếp điểm y đóng lại. Sau đó trong lúc vẫn giữ nút $a1$ và ấn tiếp nút $a2$, $a2=1$. Nút ấn thường đóng $a2$ mở ra, tuy nhiên cuộn dây Y vẫn có điện vì tiếp điểm y song song với $a2$ đã đóng lại trước đó duy trì điện cho cuộn dây Y . Nút ấn $a2$ thường mở bây giờ đóng lại, cộng với tiếp điểm y đã đóng làm cho cuộn dây Z có điện $Z=1$ (trạng thái 3). Bây giờ ta nhả tay khỏi nút ấn $a2$, $a2=0$, (vẫn ấn nút $a1$) thì tiếp điểm $a2$ thường mở sẽ trở về trạng thái thường mở và làm cho cuộn dây Z mất điện, $Z=0$. Cuộn dây vẫn có điện, $Y=1$. Hệ thống trở lại trạng thái 2. Và nếu ta nhả tay khỏi nút $a1$ thì trạng thái lại trở về trạng thái 1.

Khi hệ thống đang ở trạng thái 1, nếu ta chỉ ấn nút $a2$, $a2=1$, khi đó cả cuộn dây Y và Z cũng đều không có điện, $Y=Z=0$ (trạng thái 4). Nếu ta vẫn giữ $a2$ và ấn thêm $a1$ thì cuộn dây Y và Z cũng vẫn không có điện. Vì khi đó nút ấn thường đóng $a2$ bị ấn trước nên mạch bị hở. Khi ấn nút $a1$ cũng không làm Y có điện. Vì Y không có điện nên tiếp điểm y vẫn mở, làm cho Z cũng không có điện. Đây là trạng thái số 5.

Nếu sau đó ta vẫn giữ nút a1 và thả nút a2 ra thì hệ thống quay về trạng thái 2. Và nếu thả nút a1 ra thì từ trạng thái 2 hệ thống sẽ trở về trạng thái 1.

Có thể thấy rõ ở trong ví dụ này khác biệt cơ bản của mạch logic tuần tự và mạch tổ hợp. Trong mạch logic tổ hợp, hàm chỉ có 1 giá trị với 1 tổ hợp biến. Tuy nhiên trong mạch logic tuần tự thì khác. Trong ví dụ này trạng thái 3 và 5 có cùng tổ hợp biến đầu vào ($a_1=a_2=1$), tuy nhiên đầu ra của trạng thái 3 là $Y=Z=1$, khác với đầu ra của trạng thái 5 là $Y=Z=0$. Giá trị của mạch logic tuần tự không những phụ thuộc vào tổ hợp đầu vào mà còn phụ thuộc vào trình tự thực hiện, tức là các trạng thái trước đó nữa, hay nói một cách khác là mạch có nhớ.

3.2 Tổng hợp mạch logic tuần tự

Bài toán tổng hợp mạch tuần tự là bài toán khó, hơn nữa từ một yêu cầu đề ra lại có nhiều cách giải quyết khác nhau, do vậy vấn đề chung ở đây là phải dựa vào một chỉ tiêu tối ưu nào đó, đồng thời để tìm được lời giải tối ưu thì ngoài các suy đoán logic người thiết kế còn phải tận dụng các kinh nghiệm thực tế rất phong phú và đa dạng. Dưới đây chỉ đề cập đến một số bước thực hiện chung và các ví dụ cụ thể để minh họa phương pháp tổng hợp mạch tuần tự. Các phương pháp tổng hợp mạch tuần tự được giới thiệu:

- Phương pháp ma trận trạng thái
- Phương pháp Grafset

3.2.1 Phương pháp ma trận trạng thái

Thông thường trong thực tế, người thiết kế sẽ nhận được một yêu cầu tổng hợp nên mạch logic để thực hiện một yêu cầu công nghệ được đặt ra nào đó. Do đó bước đầu tiên để là phải mã hóa bài toán dưới dạng các biến logic, sau đó thì sẽ dùng các công cụ toán học để tổng hợp ra các hàm logic thỏa mãn yêu cầu công nghệ đặt ra. Sau khi đã mã hóa bài toán rồi có thể sử dụng phương pháp ma trận trạng thái để tổng hợp.

Trình tự tổng hợp mạch logic tuần tự sử dụng phương pháp ma trận trạng thái như sau:

Bước 1:

Xác định các trạng thái của hệ và xây dựng graph chuyển trạng thái. Đầu tiên ta phải liệt kê được tất cả các trạng thái mà ta cần quan tâm của hệ và phải chỉ ra được mối liên hệ giữa các trạng thái đó bằng các cung tròn có hướng. Một cung tròn có hướng từ trạng thái i đến trạng thái j cho ta biết hệ thống có thể chuyển từ trạng thái i sang trạng thái j.

Bước 2:

Lập bảng chuyển trạng thái MI. Bảng chuyển trạng thái MI có số hàng bằng số trạng thái của hệ, mỗi hàng ứng với một trạng thái. Số cột chia làm 2 vùng, vùng các biến đầu vào và vùng các biến đầu ra. Tại vùng các biến đầu ra, số cột bằng số đầu ra, và ta điền giá trị của các biến đầu ra ứng với các trạng thái của hệ vào các ô tương ứng với hàng của trạng thái đó.

Đối với vùng biến đầu vào, số cột bằng số tổ hợp biến đầu vào (N biến đầu vào sẽ có 2^N cột). Mỗi cột ứng với một tổ hợp biến đầu vào và được viết theo thứ tự giống như với bảng Các nô. Việc điền vào các ô ở vùng này như sau (dựa vào graph chuyển trạng thái)

- Tại ô trong hàng của trạng thái i , thuộc cột ứng với đầu vào của trạng thái i ta điền (i) (số và có vòng tròn bao quanh). Ta gọi đây là các trạng thái i ổn định.
- Nếu trạng thái i có thể chuyển sang trạng thái j thì tại ô trong hàng của trạng thái i , thuộc cột đầu vào của trạng thái j ta điền số j . Ta gọi là trạng thái trung gian.

Bước 3:

Rút gọn bảng chuyển trạng thái MI thành MII. Nguyên tắc rút gọn bảng chuyển MI là ta ghép các hàng của bảng MI. Các hàng trong bảng MI có thể ghép được với nhau khi chúng thỏa mãn các yêu cầu sau:

- Không quan tâm đến các đầu ra của các trạng thái ứng với các hàng
- Tất cả các ô cùng cột phải được điền cùng một số (không quan tâm đến ổn định hay trung gian), hoặc là ô trống. Nếu chỉ cần một cột có 2 ô trong cột đó điền số khác nhau thì không ghép được.

Nguyên tắc ghép 2 hay nhiều hàng trong bảng MI thành một hàng trong bảng MII như sau:

- Nếu ghép các ô có trạng thái ổn định thì ghi trạng thái ổn định
- Nếu không có trạng thái ổn định thì ghi trạng thái trung gian (nếu có) hoặc để trống.

Chú ý: Trong bảng MII không có vùng đầu ra, vì khi ghép các hàng vào thì một hàng trong MII có thể ứng với nhiều đầu ra.

Bước 4:

Xác định và mã hóa biến trung gian. Từ bảng MII ta sẽ xác định số biến trung gian cần thiết. Nếu số hàng của MII là H thì số biến trung gian là S , là số nhỏ nhất sao cho $2^S \geq H$. Sau khi chọn số biến trung gian, ta sẽ phân chia các hàng (các trạng thái) ứng với các tổ hợp biến trung gian một cách hợp lý.

Bước 5:

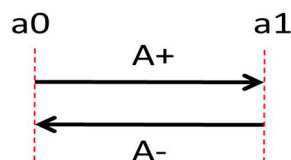
Lập bảng Các nô để xác định hàm logic cho các biến trung gian và đầu ra. Bảng Các nô sẽ tương tự như MII với số cột bằng số cột của MII ứng với các tổ hợp của biến đầu vào, số hàng là 2^S ứng với các tổ hợp của các biến trung gian. Việc điền vào các ô của bảng Các nô như sau:

- Nếu các ô trong MII là ô trống thì các ô tương ứng trong bảng Các nô cũng là các ô trống (ô trống tương đương với trạng thái không xác định)
- Nếu các ô trong bảng Các nô không có ô tương ứng trong bảng MII (do trong trường hợp $2^S > H$, sẽ có một số hàng có trong bảng Các nô nhưng không có trong MII) thì các ô này cũng để trống
- Nếu các ô trong bảng MII ghi trạng thái ổn định thì các ô tương ứng trong bảng Các nô của biến ra hoặc biến trung gian ghi giá trị của biến đó ứng với trạng thái được ghi trong ô.
- Nếu các ô trong bảng MII ghi trạng thái trung gian thì các ô tương ứng trong bảng Các nô của biến ra để trống, các ô tương ứng trong bảng Các nô của biến trung gian ghi giá trị của biến trung gian đó ứng với trạng thái được ghi trong ô.

Sau khi đã lập được bảng Các nô cho các biến trung gian và đầu ra thì ta sẽ tối thiểu hóa các bảng Các nô đó và tìm được hàm logic cho biến ra và biến trung gian. Như vậy các biến trung gian và đầu ra là hàm của đầu vào và các biến trung gian.

Ví dụ 3.1

Cho công nghệ như hình vẽ dưới đây

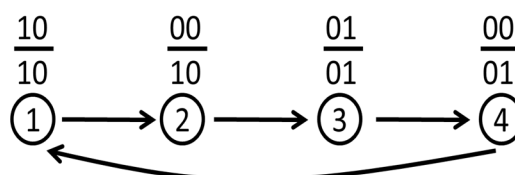


Ban đầu thiết bị chạm vào a0 và di chuyển sang phải. Sau đó thiết bị chạm vào a1 và di chuyển sang trái. Tiếp theo thiết bị lại chạm vào a0 và chu trình được lặp lại. Chú ý rằng Khi thiết bị rời khỏi vị trí của cảm biến thì cảm biến lại trở về trạng thái không tác động.

Trình tự tổng hợp mạch thực hiện công nghệ này như sau:

Bước 1:

- Xác định biến vào là a0 và a1; biến ra là A+ (sang phải) và A- (sang trái)
- Số trạng thái và graph chuyển trạng thái: vào/ra = a0a1/A+A-



Trạng thái 1 là trạng thái cảm biến a_0 tác động, $a_0 = 1$, a_1 chưa tác động, $a_1 = 0$. Khi thiết bị chạy sang phải, $A^+ = 1$, $A^- = 0$ ($a_0 a_1 / A^+ A^- = 10/10$). Khi thiết bị chạy khỏi vị trí cảm biến thì a_0 trở về trạng thái không tác động ($a_0 = 0$), thiết bị vẫn tiếp tục chạy sang phải, ta có trạng thái 2 (00/10). Khi thiết bị chạy và tác động vào cảm biến a_1 , $a_1 = 1$, thì thiết bị chuyển hướng chạy sang trái ($A^+ = 0$; $A^- = 1$). Ta có trạng thái 3 (01/01). Khi thiết bị chạy khỏi vị trí cảm biến thì a_1 trở về trạng thái không tác động ($a_1 = 0$), thiết bị vẫn tiếp tục chạy sang trái, ta có trạng thái 4 (00/01). Thiết bị sau đó sẽ gặp a_0 và chu trình sẽ lặp lại.

Bước 2:

- Lập bảng chuyển trạng thái MI

Trạng thái	Tín hiệu vào: $a_0 a_1$				Tín hiệu ra	
	00	01	11	10	A^+	A^-
① (sang phải)	2			①	1	0
② (trên đường sang phải)	②	3			1	0
③ (sang trái)		③			0	1
④ (trên đường sang trái)	④			1	0	1

Trạng thái 1 có đầu vào là 10 nên ở hàng 1, cột ứng với tổ hợp 10 ta ghi (1). Tương tự ta ghi các trạng thái ổn định khác như (2) ở hàng 2, cột 00; (3) ở hàng 3 cột 01; và (4) ở hàng 4, cột 00.

Trạng thái 1 có thể chuyển sang trạng thái 2 nên ở hàng 1, cột ứng với tổ hợp 00 là tổ hợp ứng với trạng thái 2, ta ghi số 2. Tương tự ta ghi 3 ở hàng 2, cột 01; 4 ở hàng 3, cột 00 và 1 ở hàng 4 cột 10.

Bước 3:

- Rút gọn bảng chuyển trạng thái MI thành MII

Ta thấy hàng 1 và 2 có thể ghép với nhau, hàng 3 và 4 có thể ghép với nhau. Bảng chuyển MII sẽ như sau

	a_0			
① + ②	② ¹⁰	3		① ¹⁰
③ + ④	④ ⁰¹	③ ⁰¹		1

Ô ở cột 1 của hàng 1 là 2 ổn định, ở hàng 2 là 2 trung gian nên ghép thành 2 ổn định

Ô ở cột 2 hàng 1 là ô trống, ở hàng 2 là 3 trung gian nên ghép thành 3 trung gian

Ô ở cột 3 hàng 1 và 2 đều là ô trống nên ghép thành ô trống

Ô ở cột 4 hàng 1 là 2 trung gian, ở hàng 2 là ô trống nên ghép thành 2 trung gian.

Tương tự ta có kết quả ghép hàng 3 và 4 như trong bảng.

Các chỉ số nhỏ chỉ tổ hợp đầu ra với trạng thái ổn định đó

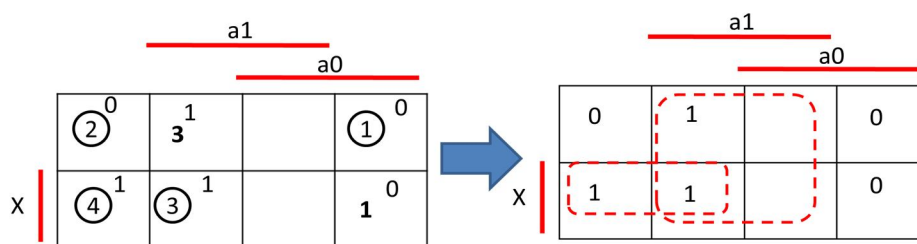
Bước 4:

- Xác định và mã hóa biến trung gian. Ta thấy MII có 2 hàng nên ta cần 1 biến trung gian, đặt là X. Ta phân chia hàng 1 của MII (là trạng thái 1 và 2) ứng với $X = 0$, và hàng 2 của MII (là trạng thái 3 và 4) ứng với $X = 1$.

Bước 5:

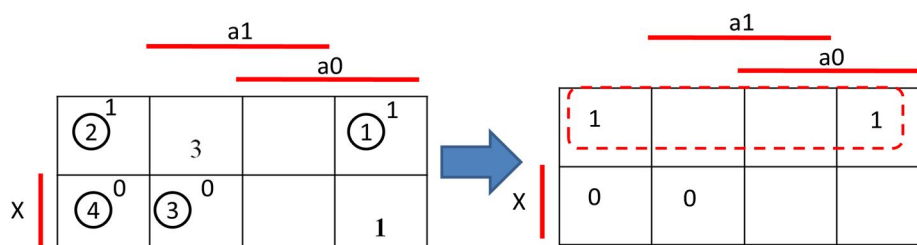
- Lập các bảng Các nô và tìm hàm logic cho các biến trung gian và biến ra

a) Với biến trung gian X



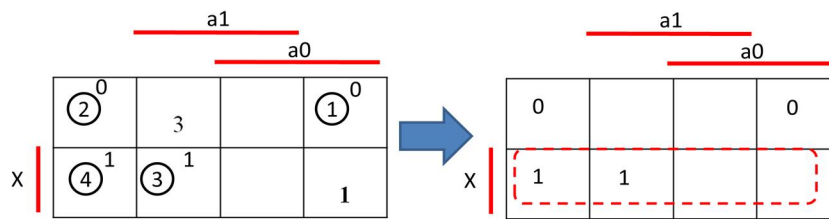
Các trạng thái 1 và 2 ứng với $X = 0$ nên trong các ô trong bảng Các nô ứng với các ô trong MII có ghi trạng thái 1 và 2 (ổn định và trung gian) ta điền giá trị 0. Các trạng thái 3 và 4 ứng với $X = 1$ nên trong các ô trong bảng Các nô ứng với các ô trong MII có ghi trạng thái 3 và 4 (ổn định và trung gian) ta điền giá trị 1. Cuối cùng ta được bảng Các nô như hình vẽ và tối thiểu hóa được hàm sau : $X = a1 + \overline{a0}X$

b) Với biến đầu ra A+



Các trạng thái 1 và 2 ứng với $A+ = 1$, trạng thái 3 và 4 ứng với $A+ = 0$ nên trong các ô trong bảng Các nô ứng với các ô trong MII có ghi trạng thái 1 và 2 ổn định ta điền giá trị 1, ghi trạng thái 3 và 4 ổn định ta điền 0, ghi các trạng thái trung gian ta để trống. Cuối cùng ta được bảng Các nô như hình vẽ và tối thiểu hóa được hàm sau $A+ = \overline{X}$.

c) Với biến đầu ra A-



Các trạng thái 1 và 2 ứng với $A^- = 0$, trạng thái 3 và 4 ứng với $A^- = 1$ nên trong các ô trong bảng Các nô ứng với các ô trong MII có ghi trạng thái 1 và 2 ổn định ta điền giá trị 0, ghi trạng thái 3 và 4 ổn định ta điền 1, ghi các trạng thái trung gian ta để trống. Cuối cùng ta được bảng Các nô như hình vẽ và tối thiểu hóa được hàm sau $A^- = X$.

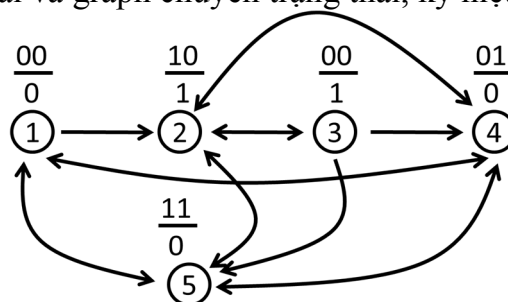
Trong ví dụ 3.1 ta không định nghĩa trạng thái có đầu vào là $a_0a_1 = 11$. Phụ thuộc vào yêu cầu công nghệ mà ta phải xem xét tất cả các khả năng có thể của hệ thống hay chỉ xem xét những trạng thái mà ta coi là quan trọng và không quan tâm đến các trạng thái có thể khác. Về nguyên tắc, việc xem xét đến tất cả các trạng thái có thể của hệ thống đảm bảo hệ thống hoạt động chính xác và tin cậy.

Ví dụ 3.2:

Cho 2 nút ấn m và d để điều khiển thiết bị T. Nếu ấn nút m sẽ cấp điện cho T, ấn nút d sẽ cắt điện của T. Các nút ấn d và m đều là nút ấn dạng xung. Tổng hợp mạch điều khiển thiết bị T

Bước 1

- Xác định biến vào: m và d; biến ra T. Khi bị ấn nút thì các biến vào có giá trị 1, khi không bị tác động thì có giá trị 0. Khi T được cấp điện thì $T = 1$, khi không có điện thì $T = 0$.
- Xác định số trạng thái và graph chuyển trạng thái, ký hiệu vào/ra = md/T



Ở đây ta xét tất cả các khả năng có thể xảy ra của hệ. Trạng thái 1 là trạng thái ban đầu. Nếu ta ấn nút m thì hệ thống chuyển sang trạng thái 2 ($T=1$). Khi ta nhả tay khỏi nút m thì T vẫn có điện ($T=1$) và ta có trạng thái 3. Nếu hệ đang ở trạng thái 3 mà ta

ấn nút d thì hệ chuyển sang trạng thái 4, T mất điện ($T=0$). Sau đó ta nhả tay ra khỏi nút d thì hệ quay về trạng thái 1.

Ngoài ra ta cũng xem xét các khả năng từ trạng thái 1 chuyển sang trạng thái 4, từ trạng thái 3 chuyển về trạng thái 2, và trạng thái 2 và 4 chuyển qua lại với nhau.

Ta cũng xét đến trạng thái 5 là trạng thái cả 2 nút m và d bị ấn cùng lúc, khi đó ta quyết định đầu ra $T = 0$. Ngoài ra ta cũng xét đến mối liên hệ của trạng thái 5 và 4 trạng thái khác của hệ, như đã chỉ ra trong graph chuyển trạng thái.

Bước 2

- Lập bảng chuyển trạng thái MI

Trạng thái	Tín hiệu vào: md				Tín hiệu ra T
	00	01	11	10	
①	①	4	5	2	0
②	3	4	5	②	1
③	③	4	5	2	1
④	1	④	5	2	0
⑤	1	4	⑤	2	0

Bước 3

Rút gọn bảng chuyển trạng thái MI thành MII: ta thấy hàng 1, 4, và 5 có thể ghép được với nhau; hàng 2 và 3 cũng có thể ghép được với nhau. Bảng chuyển MII sẽ như sau:

① + ④ + ⑤	① ⁰	④ ⁰	⑤ ⁰	1 ¹ 2
② + ③	③ ¹	0 ⁰ 4	0 ⁰ 5	② ¹

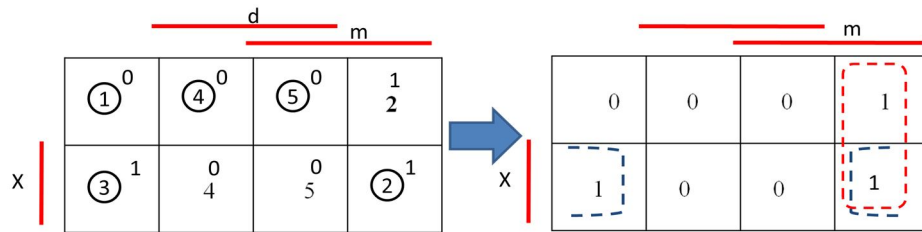
Bước 4

- Xác định và mã hóa biến trung gian

Ta thấy MII có 2 hàng nên ta cần 1 biến trung gian là X. Ta phân chia hàng 1 của MII (là các trạng thái 1, 4, và 5) ứng với $X = 0$, và hàng 2 của MII (là các trạng thái 3 và 4) ứng với $X = 1$.

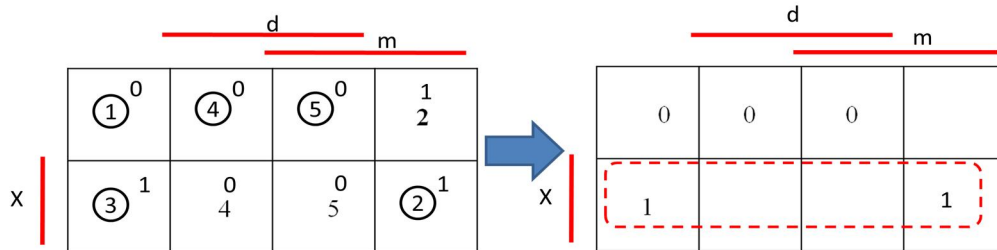
Bước 5

- Lập bảng Các nô và tìm hàm logic cho các biến trung gian và biến ra
 - Với biến trung gian X



$$X = m\bar{d} + X\bar{d} = (m + X)\bar{d}$$

b) Với biến ra T



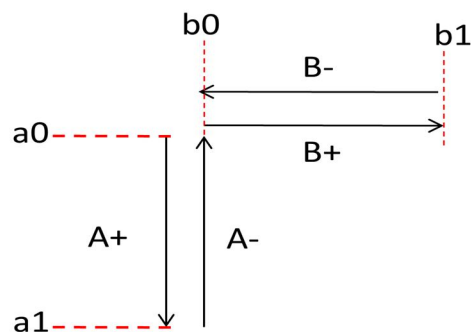
$$T = X$$

Chú ý: Ở cả hai ví dụ này ta thấy biến trung gian trùng với 1 biến ra. Ta nhận thấy rằng khi các trạng thái được ghép vào trong một hàng mà có cùng giá trị đầu ra thì có thể chọn biến đầu ra bằng biến trung gian. Ví dụ ở trong ví dụ 3.1 trạng thái 1 và 2 có các hàng tương ứng được ghép với nhau đều có đầu ra $A^+ = 1$, $A^- = 0$; trạng thái 3 và 4 đều có đầu ra $A^+ = 0$, $A^- = 1$ nên ta có thể chọn $X = A^-$. Tương tự ở ví dụ 3.2 trạng thái 1, 4, và 5 có cùng giá trị $T = 0$, trạng thái 2 và 3 có cùng giá trị $T = 1$ và các hàng tương ứng của chúng được ghép với nhau nên ta có thể chọn $X = T$.

Đối với phương pháp ma trận trạng thái, khi số đầu vào tăng lên thì việc tổng hợp trở nên phức tạp do gặp khó khăn trong việc ghép các hàng một cách tối ưu và bảng Các nô có kích thước quá lớn. Trong nhiều trường hợp ta có thể rút gọn được số biến đầu vào như trong ví dụ dưới đây.

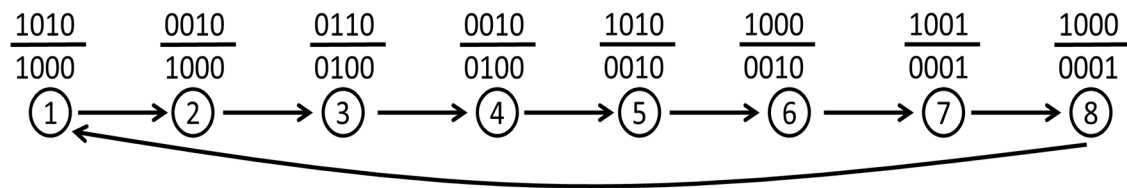
Ví dụ 3.3

Cho công nghệ như hình vẽ dưới đây



Hệ thống gồm 2 cơ cấu chuyển động lên-xuống và phải-trái. Đầu tiên cơ cấu lên xuống sẽ thực hiện chuyển động đi xuống (cơ cấu chuyển động phải-trái đứng im). Khi gặp cảm biến a1 thì sẽ thực hiện chuyển động lên (cơ cấu chuyển động phải-trái vẫn đứng im). Khi gặp cảm biến a0 thì cơ cấu lên-xuống dừng, cơ cấu phải-trái thực hiện chuyển động sang phải. Khi gặp cảm biến b1 thì thực hiện chuyển động sang trái. Khi gặp b0 thì cơ cấu phải trái dừng và cơ cấu lên xuống thực hiện chuyển động đi xuống và chu trình sẽ được lặp lại.

Ở đây ta có 4 biến đầu vào (a0a1b0b1) và 4 biến đầu ra (A+A-B+B-). Bình thường graph chuyển trạng thái sẽ gồm 8 trạng thái như dưới đây

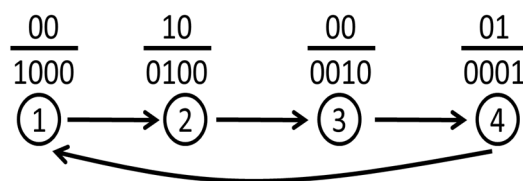


Tuy nhiên ta có thể rút gọn số biến đầu vào và số trạng thái của hệ như sau:

Chọn biến a thay cho a0 và a1 sao cho: a0 tác động làm cho biến a được thiết lập giá trị 0 duy trì giá trị 0 khi a0 đã ngừng tác động. Và biến a chỉ được thiết lập giá trị 1 khi biến a1 tác động và duy trì giá trị 1 khi a1 ngừng tác động. Có thể thấy rằng a0 đóng vai trò nút d, a1 đóng vai trò nút m và a đóng vai trò T trong ví dụ 3.2, do đó ta có mối quan hệ sau $a = (a1 + a)\overline{a0}$.

Tương tự ta chọn biến b thay cho b0 và b1, và ta có mối quan hệ $b = (b1 + b)\overline{b0}$

Sau khi chọn 2 biến mới a và b làm biến đầu vào thì ghraph chuyển trạng thái của hệ mới với quan hệ vào/ra là ab/A+A-B+B- sẽ như sau



Bây giờ bài toán trở nên đơn giản hơn với 2 đầu vào và 4 trạng thái.

Kết quả

- Biến trung gian $X = a + \overline{b}X$
- Các biến ra: $A+ = \overline{b}X$; $A- = a$; $B+ = \overline{a}X$; $B- = b$

3.2.2 Phương pháp Grafcet

Grafcet là từ viết tắt của tiếng Pháp “Graphe fonctionnel de commande étape transition” là một đồ hình chức năng cho phép mô tả các trạng thái làm việc của hệ

thống và biểu diễn quá trình điều khiển với các trạng thái chuyển biến từ trạng thái này sang trạng thái khác, đó là một graph định hướng và được xác định bởi các phần tử sau

$$G := \{E, T, A, M\}$$

Trong đó

- $E = \{E_1, E_2, \dots, E_m\}$ là tập hữu hạn các trạng thái (giai đoạn) của hệ thống. Mỗi trạng thái ứng với những tác động nào đó của phần điều khiển và trong một trạng thái các hành vi điều khiển là không thay đổi. Một trạng thái có hai khả năng là hoạt động và không hoạt động. Điều khiển chính là thực hiện các mệnh đề logic chứa các biến vào và các biến ra để hệ thống có được một trạng thái xác định trong hệ và đó cũng chính là một trạng thái của grafcet.
- $T = \{t_1, t_2, \dots, t_p\}$ là tập hữu hạn các chuyển tiếp (chuyển trạng thái). Hàm Boole gắn với một chuyển tiếp được gọi là “một tiếp nhận”. Giữa hai trạng thái luôn luôn tồn tại một chuyển tiếp.
- $A = \{a_1, a_2, \dots, a_n\}$ là tập các cung định hướng nối giữa một trạng thái này với một chuyển tiếp hoặc giữa một chuyển tiếp với một trạng thái
- $M = \{m_1, m_2, \dots, m_m\}$ là tập các giá trị 0 và 1. Nếu $m_i = 1$ thì trạng thái i là hoạt động, nếu $m_i = 0$ thì trạng thái i là không hoạt động.

Grafcet cho một quá trình luôn luôn là một đồ hình khép kín từ trạng thái đầu đến trạng thái cuối và từ trạng thái cuối đến trạng thái đầu.

a. Một số ký hiệu dùng trong grafcet

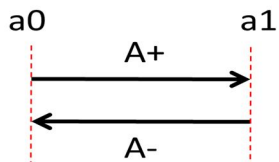
- Một trạng thái được biểu diễn bằng một hình chữ nhật có đánh số. Gắn liền với biểu tượng trạng thái là một hình chữ nhật bên cạnh, trong hình chữ nhật này có ghi các tác động của trạng thái đó.
 - + Trạng thái khởi đầu được thể hiện bằng hai hình chữ nhật lồng vào nhau
 - + Trạng thái đang hoạt động có thêm dấu “•” ở trong hình chữ nhật trạng thái
- Một chuyển tiếp được biểu diễn bằng đường gạch “-“, bên cạnh ghi các tác nhân kích thích (biến vào) liên quan đến chuyển tiếp đó.

b. Quy tắc vượt qua chuyển tiếp (quy tắc hoạt động của grafcet)

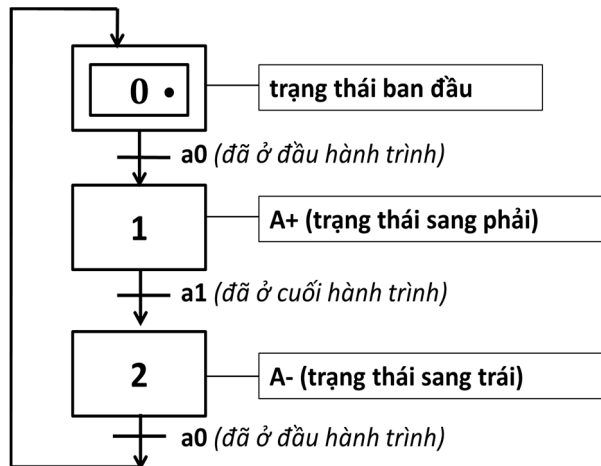
- Một chuyển tiếp là hợp cách (hoặc chuẩn) khi tất cả các trạng thái đầu vào của nó (các trạng thái có cung định hướng nối theo hướng từ trạng thái đó đến chuyển tiếp) là hoạt động. Một chuyển tiếp chỉ được vượt qua khi nó là chuẩn và tiếp nhận gắn với chuyển tiếp là đúng.
- Việc vượt qua một chuyển tiếp sẽ làm hoạt động trạng thái kế tiếp (trạng thái có cung định hướng đi từ chuyển tiếp đến nó) và khử bỏ hoạt động của trạng thái đầu vào của chuyển tiếp

Ví dụ 1

Cho công nghệ như hình dưới đây



Grafcet ứng với công nghệ này như hình dưới đây



Trạng thái 0 (S0) là trạng thái ban đầu (ký hiệu là 2 hình chữ nhật lồng vào nhau)

Trạng thái 1 (S1) và 2 (S2) là các trạng thái của hệ. Trạng thái 1 ứng với chuyển động sang phải và tương ứng với biến $S1 = A+$. Trạng thái 2 ứng với chuyển động sang trái và tương ứng với biến $S2 = A-$. Giữa trạng thái 0 và trạng thái 1 có một chuyển tiếp $t1$ với tiếp nhận $t1 = a0$. Giữa trạng thái 1 và trạng thái 2 có một chuyển tiếp $t2$ với tiếp nhận $t2 = a1$. Giữa trạng thái 2 và trạng thái 0 có một chuyển tiếp $t3$ với tiếp nhận $t3 = a0$.

Ta có các cung định hướng từ S0 tới $t1$, $t1$ tới S1; S1 tới $t2$, $t2$ tới S2; S2 tới $t3$, $t3$ tới S0.

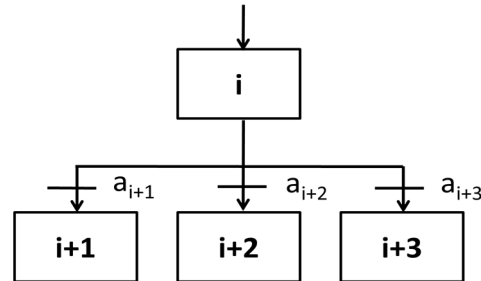
Giả sử hệ thống đang ở trạng thái ban đầu (có một dấu “•” ở trạng thái S0). Khi đó chuyển tiếp $t1$ là chuẩn. Nếu tiếp nhận $t1 = a0$ đúng ($t1 = a0 = 1$) thì chuyển tiếp $t1$ được vượt qua. Khi đó trạng thái 1 sẽ hoạt động và trạng thái ban đầu sẽ ngừng hoạt động. Dấu “•” sẽ chuyển từ trạng thái S0 sang trạng thái S1. Điều này tương ứng với việc hệ chuyển từ trạng thái ban đầu sang trạng thái sang phải. Và khi đó chuyển tiếp $t2$ là chuẩn.

Vật chuyển động sang phải và tác động lên cảm biến $a1$, khi đó tiếp nhận $t2 = a1 = 1$ đúng và chuyển tiếp $t2$ được vượt qua. Khi đó trạng thái 1 sẽ ngừng hoạt động (ngừng chuyển động sang phải) và trạng thái 2 sẽ hoạt động (chuyển động sang trái). Dấu “•” sẽ chuyển từ trạng thái S1 sang trạng thái S2.

Tiếp theo là tiếp nhận t_2 là chuẩn và khi cảm biến a_0 bị tác động thì hệ sẽ chuyển sang trạng thái S_0 và ngay sau đó chuyển sang S_1 .

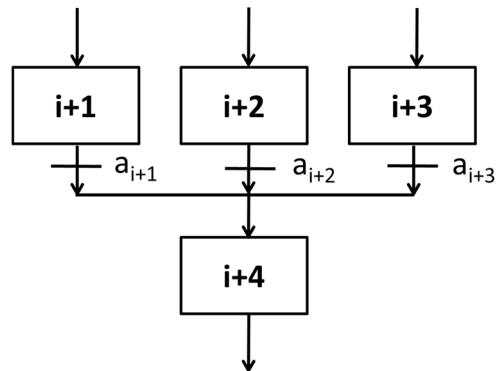
c. *Phân nhánh trong grafcet*

- Phân kỳ “HOẶC”



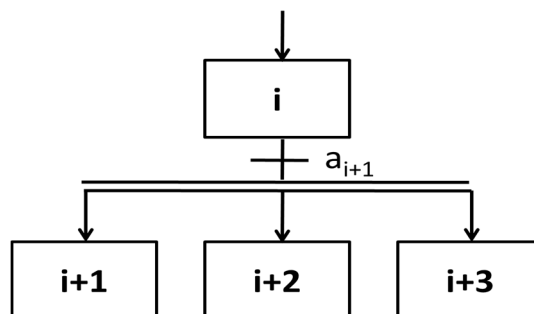
Nếu trạng thái S_i hoạt động và chuyển tiếp a_{i+1} (hoặc a_{i+2} hoặc a_{i+3}) đúng thì hệ chuyển sang trạng thái S_{i+1} (hoặc S_{i+2} hoặc S_{i+3}).

- Hội tụ “Hoặc”



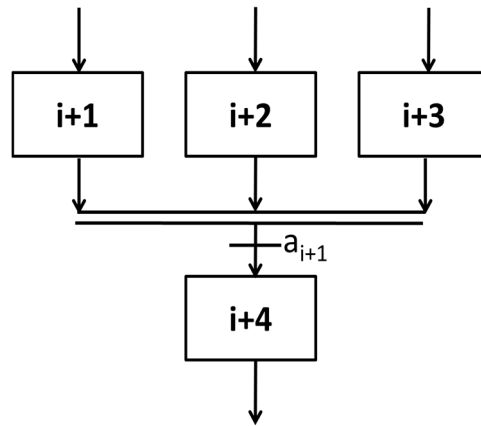
Nếu trạng thái S_{i+1} hoạt động và chuyển tiếp a_{i+1} đúng thì hệ chuyển sang hoạt động ở trạng thái S_{i+4} . Tương tự như vậy nếu trạng thái S_{i+2} (hoặc S_{i+3}) hoạt động và chuyển tiếp a_{i+2} (hoặc a_{i+3}) đúng thì hệ chuyển sang trạng thái S_{i+4} .

- Phân kỳ “VÀ”



Nếu trạng thái S_i hoạt động và chuyển tiếp a_{i+1} đúng thì tất cả trạng thái S_{i+1} , S_{i+2} và S_{i+3} đồng thời hoạt động.

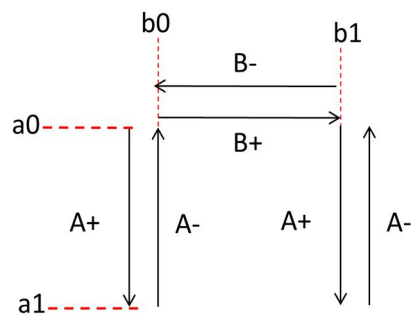
- Hội tụ “VÀ”



Nếu trạng thái S_{i+1} , S_{i+2} và S_{i+3} cùng hoạt động và chuyển tiếp a_{i+1} đúng thì trạng thái S_{i+4} sẽ hoạt động, đồng thời 3 trạng thái trên cũng ngừng hoạt động.

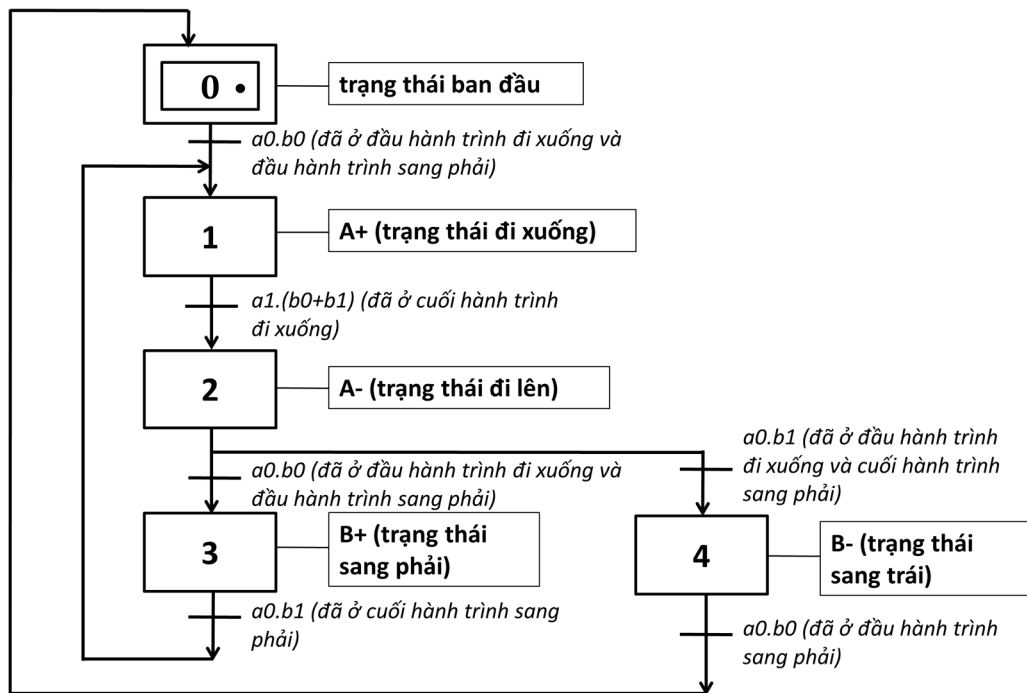
Ví dụ 2:

Cho công nghệ như hình vẽ dưới đây



Hệ thống gồm 2 cơ cấu chuyển động lên-xuống và phải-trái. Đầu tiên cơ cấu lên xuống sẽ thực hiện chuyển động đi xuống (cơ cấu chuyển động phải-trái đứng im). Khi gặp cảm biến a1 thì sẽ thực hiện chuyển động lên (cơ cấu chuyển động phải-trái vẫn đứng im). Khi gặp cảm biến a0 thì cơ cấu lên-xuống dừng, cơ cấu phải-trái thực hiện chuyển động sang phải và mang theo cả cơ cấu lên-xuống. Khi gặp cảm biến b1 cơ cấu phải-trái dừng và cơ cấu lên xuống hoạt động thực hiện chuyển động xuống thì thực hiện chuyển động sang trái. Khi gặp cảm biến a1 thì sẽ thực hiện chuyển động lên (cơ cấu chuyển động phải-trái vẫn đứng im). Khi gặp cảm biến a0 thì cơ cấu lên-xuống dừng, cơ cấu phải-trái thực hiện chuyển động sang trái và mang theo cả cơ cấu lên-xuống. Khi gặp b0 thì cơ cấu phải trái dừng và cơ cấu lên xuống thực hiện chuyển động đi xuống và chu trình sẽ được lặp lại.

Grafcet ứng với công nghệ này như hình dưới đây



d. Grafcet và hàm logic tương ứng

Sau khi đã diễn tả công nghệ bằng GRAFCET, ta có thể sử dụng một số công cụ để vận hành GRAFCET để thực hiện công nghệ yêu cầu. Một trong những công cụ này là sử dụng PLC với ngôn ngữ lập trình SFC, sẽ được giới thiệu ở phần sau. Ở phần này ta sẽ xem xét cách chuyển đổi từ GRAFCET thành các hàm logic để thực hiện công nghệ yêu cầu qua các phần tử logic cơ bản.

Ta nhận thấy với mỗi trạng thái S_i của hệ sẽ phải có 2 hàm: hàm đóng, tức là hàm làm cho trạng thái bắt đầu làm việc, ký hiệu là S_i^+ ; và hàm cắt, tức là hàm làm cho trạng thái ngừng hoạt động, ký hiệu là S_i^- .

Theo quy tắc vượt qua chuyển tiếp, trạng thái S_i sẽ bắt đầu hoạt động khi trạng thái trước nó (trạng thái đầu vào, giả sử là S_{i-1}) là hoạt động và tiếp nhận tín hiệu gắn với chuyển tiếp nằm giữa S_{i-1} và S_i là đúng. Từ đó ta có hàm đóng $S_i^+ = t_i.S_{i-1}$

Cũng theo quy tắc vượt qua chuyển tiếp, khi trạng thái S_{i+1} (là trạng thái phía sau, trạng thái đầu ra của trạng thái S_i) hoạt động, nó sẽ khử bỏ hoạt động của trạng thái S_i . Do đó ta có hàm cắt $S_i^- = S_{i+1}$.

Như vậy ta đã có thể thành lập được hàm đóng và hàm cắt của trạng thái S_i từ các trạng thái ngay trước và ngay sau, và các tiếp nhận liên quan. Từ hàm đóng và hàm cắt này ta có thể xác định được hàm logic của trạng thái S_i dựa vào kết quả ở phần trước như sau:

$$S_i = (S_i^+ + S_i^-) \overline{S_i^-}$$

Chú ý: nếu ta thực hiện mạch logic bằng các phần tử RS Flip-Flop thì S_i^+ sẽ ứng với đầu S, S_i^- ứng với đầu R của Flip-Flop.

Đối với các mạch phân nhánh ta có các hàm logic tương ứng như sau

- Mạch phân kỳ “HOẶC”:

$$S_i^- = S_{i+1} + S_{i+2} + S_{i+3}$$

$$S_{i+1}^+ = a_{i+1} S_i$$

$$S_{i+2}^+ = a_{i+2} S_i$$

$$S_{i+3}^+ = a_{i+3} S_i$$
- Mạch hội tụ “HOẶC”

$$S_{i+1}^- = S_{i+2}^- = S_{i+3}^- = S_{i+4}$$

$$S_{i+4}^+ = a_{i+1} S_{i+1} + a_{i+2} S_{i+2} + a_{i+3} S_{i+3}$$
- Mạch phân kỳ “VÀ”

$$S_i^- = S_{i+1} \cdot S_{i+2} \cdot S_{i+3}$$

$$S_{i+1}^+ = S_{i+2}^+ = S_{i+3}^+ = S_i$$
- Mạch hội tụ “VÀ”

$$S_{i+1}^- = S_{i+2}^- = S_{i+3}^- = S_{i+4}$$

$$S_{i+4}^+ = a_{i+1} S_{i+1} S_{i+2} S_{i+3}$$

1.3.11 4.1.2 Khả năng của PLC

Cùng với sự ra đời, PLC mang theo nhiều khả năng để ứng dụng trong nhiều hệ thống khác nhau. PLC có thể thực hiện chức năng của các rơ le, các bộ định thời, bộ đếm, các bản mạch điều khiển và mạch điện tử logic. PLC hỗ trợ khả năng tùy biến điều khiển tự động và điều khiển bằng tay một cách linh hoạt. Trong các hệ thống điều khiển liên tục, PLC có thể thực hiện các phép tính toán số học, đọc và xuất các đại lượng tương tự, thực hiện các thuật toán điều khiển PID, Fuzzy ... Tuy nhiên, bản chất của quá trình điều khiển và tính toán vẫn là điều khiển số. Trong hệ thống, PLC còn có chức năng điều khiển tổng thể, cảnh báo, ghép nối mở rộng hệ thống.

PLC mang lại sự thuận tiện trong việc rút ngắn thời gian thi công, lắp đặt hệ thống, dễ dàng thay đổi hoạt động với tổn thất nhỏ. Đồng thời với việc sử dụng PLC, các thiết kế có thể được tính toán tương đối chính xác giá trị thi công, dễ dàng thay đổi hoạt động nhờ thay đổi phần mềm, ứng dụng điều khiển phạm vi rộng, độ tin cậy cao và hoạt động ổn định trong các môi trường khắc nghiệt do được chế tạo tuân thủ các tiêu chuẩn trong công nghiệp.

1.3.12 4.1.3 Phân loại PLC

PLC có rất nhiều chủng loại do nhiều nhà sản xuất cung cấp. Một số nhà sản xuất và tích hợp hệ thống sử dụng PLC cho chính họ tạo ra. Một số nhà sản xuất cung cấp PLC mang tính phổ biến cho người thiết kế và tích hợp hệ thống khác. PLC được sử dụng rộng rãi với rất nhiều ứng dụng từ đơn giản đến phức tạp, phù hợp với các yêu cầu khác nhau của từng hệ thống. Việc phân loại PLC được dựa vào khả năng của PLC về tốc độ xử lý, bộ nhớ, khả năng quản lý vào/ra. Về cơ bản được chia thành ba loại: loại nhỏ, loại vừa và loại lớn.

PLC loại nhỏ có dung lượng bộ nhớ dưới 2KB, quản lý số điểm vào/ra dưới 128, được sử dụng trong các công nghệ có yêu cầu đơn giản. Thường được gọi với tên Small, Micro. PLC loại vừa có bộ nhớ lên đến 32KB. PLC cỡ lớn có bộ nhớ cỡ MB, và quản lý tới hàng nghìn điểm vào/ra.

Về hình dạng, PLC thường được chế tạo thành 2 loại: khối cố định (Compact, Fixed) và các khối chức năng riêng biệt (Modular). Loại PLC dạng khối cố định thường là loại nhỏ và vừa, có khối MAIN gồm có CPU, số lượng đầu vào/ra cố định, nguồn cấp, các cổng truyền thông để có thể hoạt động độc lập. Với các yêu cầu mở rộng, nhà sản xuất cung cấp các khối mở rộng có chức năng đặc biệt: vào/ra số, vào/ra tương tự, vào/ra tốc độ cao, truyền thông ... ghép nối với khối MAIN. Loại PLC dạng các khối chức năng riêng biệt thường là PLC cỡ vừa và lớn. Các khối cơ bản là: nguồn, CPU, vào/ra số, vào/ra tương tự ... Với cấu trúc này, PLC được sử dụng một cách mềm dẻo và linh hoạt. Người sử dụng có thể dễ dàng lựa chọn cấu hình cho

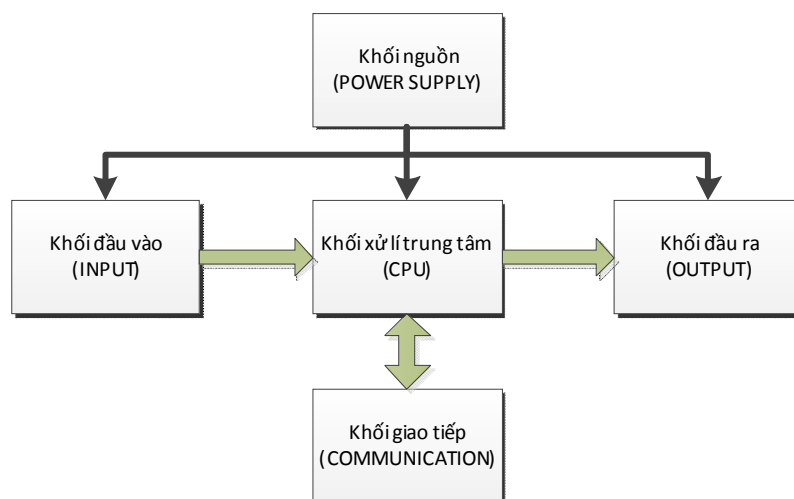
mình. Các khối thường được lắp vào các khe cắm (Slot) trên các bản mạch (Bus module, Backplane).



Hình 4.2: PLC dạng khối cố định

4.2 Cấu trúc phần cứng

PLC có rất nhiều hình dạng khác nhau, dạng khối cố định hoặc khối chức năng riêng biệt. Tuy nhiên, các hệ PLC luôn có thành phần tương tự nhau: khối nguồn, khối xử lý trung tâm, khối vào/ra và giao tiếp. Sơ đồ khối cơ bản của PLC được thể hiện trên hình ...



Hình 4.3: Sơ đồ khối cơ bản của PLC

1.3.13 4.2.1. Khối nguồn

Khối nguồn có chức năng cung cấp nguồn một chiều ổn định cho các khối khác trong hệ thống. Thông thường, điện áp nguồn đầu vào cho khối nguồn là điện áp xoay chiều 120VAC đến 220VAC (cho hầu hết PLC) hoặc điện áp một chiều 24VDC (cho một số loại PLC nhỏ). Điện áp đầu ra của khối nguồn là điện áp một chiều. Đối với khối nguồn độc lập thì điện áp đầu ra thường là 24VDC, trong một số trường hợp sẽ

có các đầu ra điện áp khác nhau: $\pm 5\text{VDC}$, $\pm 15\text{VDC}$, 24VDC . Công suất của loại này có thể lựa chọn từ nhỏ đến lớn tùy vào công suất tiêu thụ của các thiết bị, khối chức năng được cung cấp. Đối với các khối nguồn nằm trong khối MAIN, điện áp đầu ra sẽ có nhiều giá trị khác nhau và có công suất nhỏ.



Hình 4.4: Khối nguồn độc lập

1.3.14 4.2.2. Khối xử lý trung tâm (CPU)

Khối xử lý trung tâm là bộ não của PLC, có chức năng nhận dữ liệu từ các module vào, thực hiện chương trình, đưa ra kết quả và điều khiển các thiết bị được nối vào module ra. CPU có thể được chế tạo thành khối độc lập hoặc nằm trong khối MAIN. Các thành phần chính của CPU gồm có: bộ vi xử lý, bộ nhớ (MEMORY), bus ...

a. Bộ vi xử lý:

Bộ vi xử lý là hạt nhân của CPU, quyết định tốc độ xử lý, khả năng quản lý ngoại vi của PLC. Tùy vào hãng sản xuất và loại PLC, các bộ vi xử lý được sử dụng có thể là 8 bit, 16 bit hoặc 32 bit ... Một số loại PLC được dạng Modular có thể dùng nhiều CPU trong một hệ thống. Việc có nhiều CPU giúp tăng khả năng và tốc độ xử lý, tính toán, truyền thông ... trong khi các CPU có thể chia sẻ các tài nguyên khác và dự phòng khi một trong các CPU bị lỗi.

b. Bộ nhớ:

Bộ nhớ là thiết bị lưu trữ các thông tin: chương trình, dữ liệu, tham số hệ thống, cấu hình hệ thống ... Việc tổ chức và quản lý bộ nhớ do nhà sản xuất quy định. Bộ nhớ có thể được chia thành hai loại: bộ nhớ duy trì (non-Volatile) và bộ nhớ không duy trì (Volatile). Nội dung của bộ nhớ duy trì sẽ không bị mất khi mất điện. Tuy nhiên, tốc độ đọc/ghi (truy cập) bộ nhớ chậm. Vì vậy, bộ nhớ duy trì thường được dùng làm bộ nhớ chương trình, bộ nhớ lưu dữ liệu. Bộ nhớ không duy trì sẽ mất nội dung khi mất điện nhưng lại có tốc độ truy cập cao. Bộ nhớ không duy trì thường được dùng làm bộ nhớ đệm để CPU xử lý, tính toán, lưu các giá trị biến trung gian. Trong một số trường

hợp, bộ nhớ không duy trì cũng được sử dụng làm bộ nhớ dữ liệu khi có thêm nguồn nuôi cho bộ nhớ (thường là pin).

Một số loại bộ nhớ dùng làm bộ nhớ cho PLC: ROM, EEPROM, RAM, SRAM, DRAM, FLASH ...

Việc đọc/ghi thông tin với bộ nhớ được gọi là truy cập bộ nhớ. Bộ nhớ có thể được truy cập theo từng bit (với dữ liệu nhị phân 1, 0); theo từng Byte (dữ liệu 8 bit); theo từng Word (16 bit) và Double Word (DWord) (32 bit).

c. BUS:

BUS là hệ thống đường dẫn thông tin giữa các phần trong CPU. BUS trong PLC được chia thành ba loại: BUS địa chỉ (Address BUS), BUS điều khiển (Control BUS), BUS dữ liệu (Data BUS). BUS địa chỉ chứa thông tin địa chỉ của ô nhớ, byte nhớ ... BUS dữ liệu được sử dụng truyền nội dung bộ nhớ. BUS điều khiển dùng để truyền các tín hiệu quy định đọc/ghi bộ nhớ.

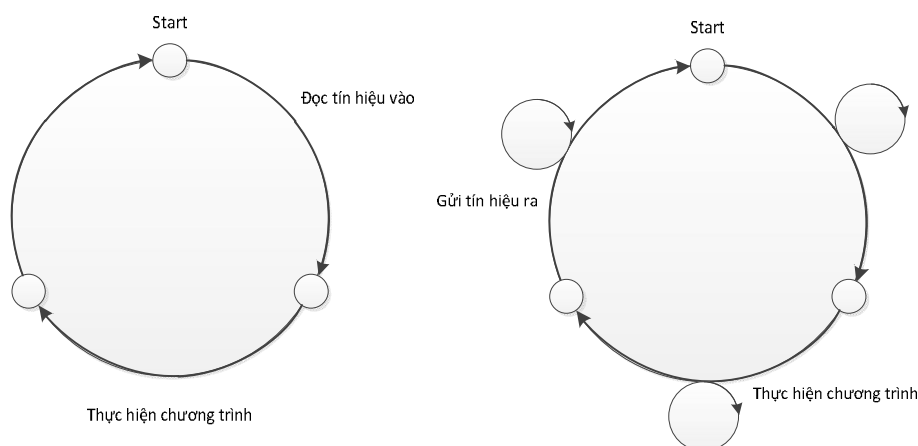
1.3.15 4.2.3. Khối vào/ra

Các module vào/ra là các thiết bị mà từ đó CPU trao đổi thông tin, dữ liệu với thế giới bên ngoài. Các module vào nhận tín hiệu từ các thiết bị vào, biến đổi thành dữ liệu gửi đến CPU. Các module ra nhận dữ liệu từ CPU, biến đổi thành tín hiệu điều khiển các thiết bị ra. Do nguồn tín hiệu vào và ra rất đa dạng về chủng loại, nên các module vào/ra cũng có rất nhiều loại. Các loại vào/ra cơ bản được chia thành: các module vào/ra rời rạc, module vào/ra tương tự, module vào/ra đặc biệt, ... Các module vào ra cũng giống CPU, nguồn, được chế tạo theo chuẩn và tương thích với CPU và tương thích với nhau qua các khe cắm.

Việc trao đổi dữ liệu giữa CPU và các module vào/ra nhờ thao tác đọc/ghi qua BUS. Mỗi lần trao đổi dữ liệu 8 bit hoặc 16 bit tùy vào loại PLC. Hệ thống quản lý đầu vào/ra theo địa chỉ. Các địa chỉ phụ thuộc vào vị trí lắp các module vào/ra trong hệ thống, loại của module vào/ra.

4.3 Hoạt động của PLC

PLC hoạt động theo nguyên tắc quét vòng (SCAN). Mỗi vòng quét (chu kỳ quét – Scan Cycle) được thực hiện qua 3 giai đoạn cơ bản.



Hình 4.5. Chu kỳ hoạt động của PLC

Giai đoạn đầu tiên, PLC thực hiện đọc các trạng thái tín hiệu ở các module vào, lưu giữ vào vùng ảnh đầu vào để làm dữ liệu thực hiện chương trình. Giai đoạn tiếp theo, chương trình lưu trong bộ nhớ được thực hiện. Kết quả sẽ được lưu trữ trong bộ nhớ để sử dụng trong các chu kỳ quét sau hoặc gửi ra các đầu ra. Ở giai đoạn thứ 3, PLC gửi dữ liệu đến vùng ảnh đầu ra và biến đổi thành tín hiệu điều khiển các cơ cấu chấp hành, các thiết bị thực được nối với module ra. PLC sẽ tiếp tục vòng quét sau và quá trình này được lặp đi lặp lại liên tục. Ngoài ra còn một số chương trình con, ngắt sẽ tạo ra các chu kỳ quét phụ của PLC.

Thời gian thực hiện 1 vòng quét có ảnh hưởng lớn đến tốc độ xử lý và khả năng xử lý thời gian thực của PLC. Vì vậy, PLC hầu hết chỉ được sử dụng trong các bài toán điều khiển có chu kỳ làm việc và thời gian điều khiển lớn. Chu kỳ quét của PLC phụ thuộc vào các nhân tố sau: tốc độ xử lý của vi xử lý của CPU, độ dài chương trình, số lượng các đầu vào, ra. Ngoài ra, chu kỳ quét còn phụ thuộc vào số lượng và nội dung các chu kỳ quét phụ. Đối với một hệ PLC, để giảm chu kỳ quét, cần thực hiện tối ưu chương trình và lựa chọn các loại PLC có khả năng phù hợp với từng ứng dụng cụ thể. Nguyên tắc hoạt động quét vòng của PLC hạn chế khả năng xử lý tức thời của PLC. Tuy nhiên, các PLC hiện đại ngày càng được trang bị và tăng cường khả năng xử lý ngắt. Vì vậy, các PLC đã dần khắc phục được nhược điểm về chu kỳ quét.

4.4 Các lệnh trong PLC (Các ngôn ngữ lập trình PLC)

4.4.1 Ladder

Ngôn ngữ lập trình giản đồ thang LD (Ladder Diagram) được người Mỹ phát minh từ một vài thế kỷ trước để thay thế điều khiển role và là ngôn ngữ lập trình PLC phổ biến nhất ở đây hiện nay, với khoảng 95% ứng dụng. Ngôn ngữ này đã được chấp

nhận rộng rãi khắp nơi trên thế giới, hầu hết người lập trình điều khiển đều có thể hiểu và sử dụng. Một cách trực quan, ngôn ngữ này giống hệt một mạch điều khiển gồm các đầu vào cần thiết để điều khiển một hay một vài đầu ra. Chính vì nó giống như mạch điện thật nên chỉ cần có kiến thức về mạch điện là có thể đọc hiểu và lập trình bằng ngôn ngữ này.

Thành phần đầu tiên phải có trong ngôn ngữ LD là các thanh nguồn (Power rail) thẳng đứng bên trái và bên phải, nơi mà nguồn (tượng tượng) chạy vào và ra. Riêng thanh bên phải có thể hiện hoặc ẩn. Trạng thái của thanh trái được coi là ON tại mọi thời điểm.

Các đường nối nằm ngang và thẳng đứng nối các phần tử tiếp điểm và cuộn dây cùng các khối hàm để truyền trạng thái từ trái sang phải, trạng thái là ON hoặc OFF tương ứng mức logic 1 hoặc 0. Theo đường ngang là nối theo quan hệ logic VÀ, còn đường nối theo phương đứng thể hiện quan hệ logic HOẶC.

Các tiếp điểm để truyền trạng thái từ trái qua phải của nó và đóng vai trò như tiếp điểm VÀ về logic. Các tiếp điểm chuẩn và kí hiệu gồm:

- Tiếp điểm thường mở ($--|^{***}|--$), khi trạng thái của biến ứng với tiếp điểm ON (tên biến đặt tại kí hiệu $***$, phía dưới đây không ghi lại kí hiệu này nữa) thì sẽ cho trạng thái của đường nối sau đó là ON và ngược lại.

- Tiếp điểm thường đóng ($--|/|--$), khi trạng thái của biến ứng với tiếp điểm OFF thì mới cho trạng thái đường nối sau nó là ON và ngược lại.

- Tiếp điểm phát xung sườn lên ($--|P|--$), khi phát hiện có trạng thái từ OFF lên ON ở biến tương ứng và phía trước ON thì sẽ cho ra đằng sau một tín hiệu ON tức thì, còn tại mọi thời điểm khác phía sau luôn OFF.

- Tiếp điểm phát xung sườn xuống ($--|N|--$), khi phát hiện có trạng thái chuyển từ ON xuống OFF ở biến tương ứng và phía trước ON thì cho đằng sau ON tức thì, còn lại mọi thời điểm khác đằng sau đều OFF.

Cuộn dây sao lại trạng thái đường nối bên trái sang bên phải đồng thời lưu lại trạng thái vào biến tương ứng. Các cuộn dây và kí hiệu chuẩn:

- Cuộn dây thường ($--(^{***})--$), trạng thái ở bên trái được sao lưu vào biến tương ứng (ở kí hiệu $***$) và sang phải.

- Cuộn dây đảo ($--(/)--$), trạng thái bên trái được sao lại sang bên phải, nghịch đảo của trạng thái bên trái được lưu vào biến tương ứng.

- Cuộn dây SET ($--(S)--$), trạng thái của biến tương ứng được set lên ON khi trạng

thái trước là ON và duy trì trạng thái này cho đến khi được reset bằng cuộn RESET.

- Cuộn dây RESET (--(R)--), trạng thái của biến tương ứng được reset xuống OFF khi có trạng thái ON phía trước, và duy trì trạng thái này cho đến khi được set bằng cuộn SET.

- Cuộn dây phát hiện xung lên (--(P)--), trạng thái của biến logic tương ứng là ON khi có chuyển trạng thái OFF lên ON phía trước, trạng thái bên trái luôn được sao lại sang bên phải.

- Cuộn dây phát hiện xung xuống (--(N)--), trạng thái của biến logic tương ứng là ON khi có chuyển trạng thái ON xuống OFF ở phía trước, trạng thái của bên trái luôn được sao lại bên phải.

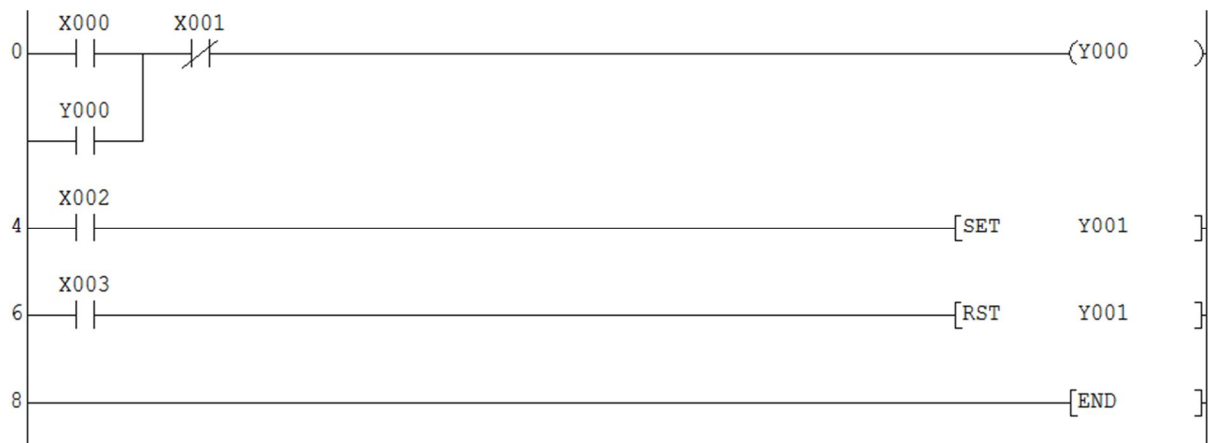
- Ngoài ra có các cuộn dây nhớ (---(M)---), cuộn dây SET nhớ (---(SM)---), cuộn dây RESET nhớ (---(RM)---), tác động giống hệt như cuộn dây thường, cuộn SET và cuộn RESET tương ứng ở trên.

Ví dụ sử dụng ngôn ngữ LD lập trình cho PLC thực hiện hàm logic:

$$Y0 = (X0 + Y0) \cdot X1$$

$$Y1^+ = X2$$

$$Y1^- = X3$$



4.4.2 IL

Ngôn ngữ lập trình Instruction List (IL) giống như ngôn ngữ Assembler lập trình cho vi xử lý, với nhiều hàng câu lệnh mà mỗi câu lệnh thể hiện cho một hoạt động. Nếu viết theo chuẩn IEC hoàn toàn thì việc chuyển phần mềm dùng cho các phần cứng khác nhau thật dễ dàng. Do là ngôn ngữ bậc thấp gần ngôn ngữ máy nên chương

trình viết bằng IL sẽ giúp bộ điều khiển thực thi nhanh hơn, hơn nữa chương trình gọn nhẹ chiếm ít bộ nhớ. Tuy nhiên với dung lượng bộ nhớ hiện nay cũng như tốc độ xử lý của thiết bị khả trình là lớn nên ưu điểm trên cũng không bù lại nhược điểm của nó là gây khó khăn cho người lập trình vì xét về mặt trực quan nó rất khó hiểu so với các ngôn ngữ đồ họa. Bên cạnh đó, lập trình với IL là một công việc nhàm chán với một số người. Khi cần chúng ta có thể dùng kết hợp ngôn ngữ này với các ngôn ngữ khác. IL là một chuỗi các câu lệnh chỉ dẫn, mỗi chỉ dẫn được viết trên một dòng mới, có thể có một hoặc nhiều toán hạng cách nhau bằng dấu phẩy. Bắt đầu chỉ dẫn có thể dùng nhân theo sau là dấu hai chấm. Chú thích đặt cuối dòng, sau câu lệnh và đặt trong (* và *). Các toán tử dùng trong ngôn ngữ IL và chức năng của chúng thể hiện trong bảng:

Bảng 4.1. Các toán tử sử dụng trong ngôn ngữ IL

Thứ tự	Toán tử	Ý nghĩa
1	LD	Đặt giá trị hiện tại cho toán hạng, nghịch đảo là LDN
2	ST	Đưa giá trị hiện tại tới địa chỉ toán hạng
3	S	Đặt toán hạng loại logic lên 1
4	R	Đặt lại logic 0 cho toán hạng
5	AND	Logic AND, nghịch đảo là ANDN
6	&	Logic AND
7	OR	Logic OR, nghịch đảo là ORN
8	XOR	Hoặc loại trừ
9	NOT	Logic nghịch đảo
10	ADD	Cộng
11	SUB	Trừ
12	MUL	Nhân
13	DIV	Chia
14	MOD	Phép chia lấy dư
15	GT	So sánh lớn hơn
16	GE	So sánh lớn hơn hoặc bằng
17	EQ	So sánh bằng
18	NE	So sánh khác nhau
19	LE	So sánh nhỏ hơn hoặc bằng
20	LT	So sánh nhỏ hơn
21	JMP	Nhảy tới nhãn

22	CAL	Gọi khối chức năng
23	RET	Trở về từ gọi hàm, khối chức năng hay chương trình

Ví dụ:

```

Start: LD      %IX1.0      (* Start push button*)
      AND      %MX5        (* CONDITION*)
      ST       %QX2        (*Turn on Motor*)

```

Giữa các câu lệnh chỉ dẫn cũng có thể chèn vào các hàng trống. Khi sử dụng hàm và khối chức năng trong IL cũng phải dùng các từ khóa tương ứng của chúng. Các từ khóa này được trình bày ở phần nói về hàm và khối chức năng.

4.4.3 STL

Với càng vòng lặp IF...THEN, câu lệnh lựa chọn và kết thúc mỗi câu lệnh là dấu chấm phẩy, ngôn ngữ lập trình ST giống các ngôn ngữ lập trình bậc cao như Pascal, C do đó đối với sinh viên và những người làm quen với lập trình với các ngôn ngữ này sẽ hiểu và dùng ST nhanh chóng. Với ST có thể xây dựng chương trình gọn nhất, có thể đưa chú thích vào dễ dàng, tiện lợi cho việc cấu trúc hóa chương trình, Chương trình viết bằng ST sẽ giúp thiết bị chạy nhanh hơn các ngôn ngữ đồ họa, khi copy và chỉnh sửa chương trình cho một gói dự án mới tương tự cũng không mấy vất vả. Việc dùng kết hợp ngôn ngữ ST với các ngôn ngữ khác thật tiện lợi, nhất là ta dùng kết hợp với SFC. Tuy không dễ nhìn thấy như ngôn ngữ đồ họa nhưng ST cũng không khó hiểu lắm. Nói chung nếu dùng riêng ngôn ngữ ST thì có vẻ không thú vị cho người lập trình, nhưng khi kết hợp thì sẽ nhanh hơn và đỡ tốn diện tích quan sát cho những chỗ không cần thiết thể hiện bằng đồ họa.

a. Biểu thức

Biểu thức (expression) bao gồm toán tử và toán hạng, khi tính toán sẽ cho ra kết quả thuộc một kiểu dữ liệu nào đó. Toán hạng là biến, hàm hoặc các biểu thức khác. Toán tử có mở đóng ngoặc (), hàm tính toán như Ln(x), Max(x,y), số mũ ** hay EXPT, nghịch đảo – hay NOT, nhân *, chia /, modulo MOD, cộng +, trừ -, so sánh <, >, <=, >=, bằng nhau = hay khác nhau <>, và logic & hay AND, XOR, OR.

Độ dài tối đa cho phép của một biểu thức tùy thuộc vào thiết bị của các hãng cung cấp.

b. Câu lệnh

Câu lệnh (statement) viết trong ngôn ngữ ST được kết thúc bằng dấu chấm phẩy (;), các loại câu lệnh:

- Lệnh gán: dùng “:=” để gán, ví dụ $A:=B$; $C:=\sin(x)$; $i:=i+1$;
- Lệnh điều khiển hàm và khối chức năng gồm các cơ chế cho việc kích hoạt các khối chức năng và cho việc trả về điều khiển các thực thể đang được kích hoạt trước khi kết thúc một hàm hay khối chức năng. Khối chức năng được kích hoạt bằng câu lệnh bằng tên khối chức năng và theo đó là các tham số vào có gán giá trị để trong dấu ngoặc, ví dụ: `CMD_TMR(IN:=%IX5, PT:=T#300ms)`; thứ tự tham số vào ở trong đó không quan trọng, cũng không bắt buộc phải gán giá trị khi đó thì giá trị gán trước đó hoặc giá trị gán khởi đầu (nếu trước đó không gán) sẽ được áp dụng. Câu lệnh RETURN sẽ giúp thoát khỏi hàm hay khối chức năng hiện thời.

- Lệnh lựa chọn gồm có câu lệnh IF hoặc lệnh CASE. Loại thứ nhất có cấu trúc: *IF biểu thức điều kiện THEN tập các câu lệnh ELSE tập các câu lệnh END_IF*; Lệnh CASE gồm một biểu thức để lựa chọn có kiểu INT và sau đó là một tập các câu lệnh với mỗi câu lệnh ứng với một số nguyên hoặc dải số nguyên, khi không có lệnh ứng với số nguyên nào trong nhóm được chọn thì sẽ thực hiện lệnh sau ELSE, hoặc cũng có thể không lệnh nào được thực hiện nếu không có giá trị thỏa mãn, kết thúc là `END_CASE`;

- Câu lệnh lặp: gồm các lệnh với cấu trúc:

FOR biến chạy có gán giá trị đầu TO giá trị cuối BY giá trị nhảy DO lệnh END_FOR; biến chạy hay là biến điều khiển, cùng các giá trị đầu và giá trị cuối phải cùng một kiểu dữ liệu trong kiểu nguyên. Việc kiểm tra điều kiện kết thúc vòng lặp được thực hiện đầu các vòng lặp, do đó không bao giờ thực hiện vượt quá giá trị cuối.

WHILE biểu thức DO câu lệnh END_WHILE; khi điều kiện biểu thức logic được kiểm tra là sai thì mới dừng vòng lặp.

REPEAT câu lệnh UNTIL biểu thức điều kiện END_REPEAT; các câu lệnh được thực hiện lặp đi lặp lại cho đến khi biểu thức điều kiện đúng thì dừng vòng lặp. Nếu không có điều kiện dừng vòng lặp không đảm bảo thì sẽ có lỗi.

4.4.4 FBD

Ngôn ngữ lập trình biểu đồ khối chức năng (function block diagram FBD) được ứng dụng rộng rãi chỉ sau LD, và có dạng đồ họa cũng như LD. Các khối chức năng được nối với nhau thành một chuỗi rất dễ dàng theo dõi, về mặt trực quan thì dễ hiểu hơn nếu như người lập trình chưa hiểu nhiều về các role logic. Các hàm và khối chức năng được lấy từ thư viện chuẩn hoặc do người lập trình xây dựng.

4.4.5 SFC

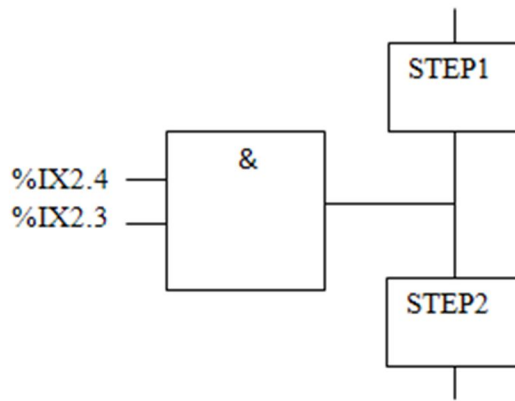
Ngôn ngữ lập trình biểu đồ hàm tuần tự (sequential function chart – SFC) cho phép sử dụng các thuật toán phức tạp trong một chuỗi các bước và các chuyển. Việc lập trình theo ngôn ngữ SFC được thực hiện giống như phương pháp tổng hợp mạch logic Grafcet, đối với những người đã quen với phương pháp tổng hợp mạch này thì đây là ngôn ngữ rất thuận tiện để viết chương trình, lại gắn bó chặt chẽ với các quá trình trong công nghiệp. Theo ngôn ngữ này, một hộp tác động khởi đầu được theo sau bởi một loạt các chuyển và các bước tác động. Một hộp tác động với phần mã bên trong được viết bằng bất kì ngôn ngữ nào của chuẩn ở trạng thái hoạt động cho đến khi bước chuyển tiếp ở ngay sau nó được kích hoạt thì hộp tác động ở trước hết tác động và hộp tác động phía sau được cho phép. Bước chuyển tiếp cũng có các mã chương trình để kiểm tra các điều kiện xem có cho chương trình chuyển sang bước tiếp theo hay không.

Do các phần tử SFC đòi hỏi cho việc lưu trữ thông tin trạng thái nên khối tổ chức chương trình có thể sử dụng để xây dựng các phần tử này là chương trình hoặc khối chức năng, còn hàm không lưu được thông tin trạng thái nên không thể sử dụng ở đây. Nếu bất kì một phần nào của khối tổ chức chương trình được chia thành các phần tử SFC, thì khi đó thì toàn bộ khối tổ chức chương trình cũng bị chia nhỏ. Nếu ko có phần SFC nào được gắn vào thì toàn bộ khối tổ chức chương trình được xem là một tác động đơn và được điều khiển bởi thực thể kích hoạt.

c. Bước nhảy

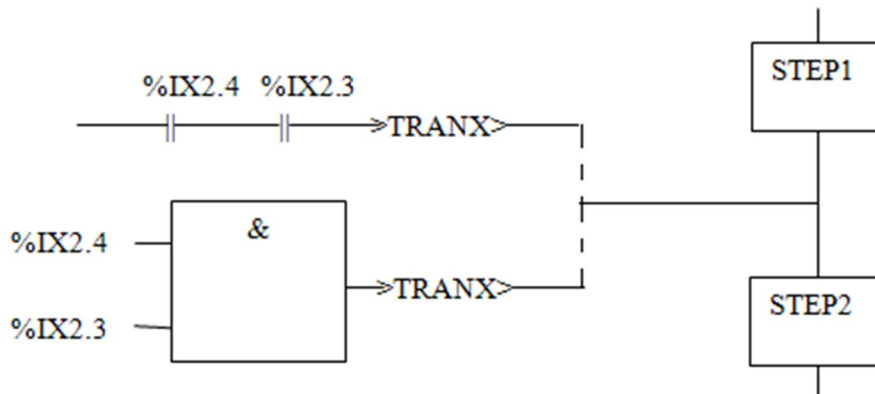
Bước nhảy (step) thể hiện các trạng thái mà hoạt động của khối tổ chức chương trình và các đầu vào ra tương ứng tuân theo các luật của các tác động đi kèm của bước nhảy đó.

Bước nhảy có thể dạng khối với tên bước nhảy ở trong đó. Bước khởi đầu kí hiệu vẽ hai đường nét bao quanh, với giá trị logic 1 mặc định, còn bước theo sau thì chỉ một đường, giá trị mặc định ban đầu là logic 0. Bước nhảy cũng có thể được khai báo dạng chữ STEP...END_STEP. Kết nối đầu vào và đầu ra của một bước là đường thẳng đứng phía trên và dưới của khối đó, hoặc dùng chữ TRANSITION...END_TRANSITION. Cờ bước nhảy thể hiện trạng thái hoạt động hay không của bước được thể hiện dưới dạng logic *****.X** trong đó ******* là tên bước nhảy. Biến này có giá trị logic 1 khi bước tương ứng đang được kích hoạt, và có giá trị 0 khi bước không



Hình 4.3. Chuyển viết bằng ngôn ngữ FBD

- Đưa một mạng LD hoặc FBD cho đầu cuối qua một bộ nối (connector) nối sang đường nối thẳng đứng.



Hình 4.4. Chuyển viết bằng FBD hoặc LD dùng bộ nối

- Dùng cấu trúc TRANSITION ...END_TRANSITION theo ngôn ngữ ST, sau từ khóa TRANSITION FROM là bước phía trước, sau từ khóa TO sẽ là bước phía sau, biểu thức gán “:=” là biểu thức logic viết bằng ST thể hiện điều kiện chuyển.

STEP STEP1: END_STEP

TRANSITION FROM STEP1 TO STEP2

:= %IX2.4 & %IX2.3 ;

END_TRANSITION

STEP STEP2: END_STEP

- Dùng cấu trúc TRANSITION ...END_TRANSITION theo ngôn ngữ IL, sau TRANSITION FROM là bước phía trước, sau từ khóa TO là bước phía sau chuyển, Các biểu thức viết bằng ngôn ngữ IL sẽ cho kết quả cho điều kiện chuyển.

STEP STEP1: END_STEP

TRANSITION FROM STEP1 TO STEP 2:

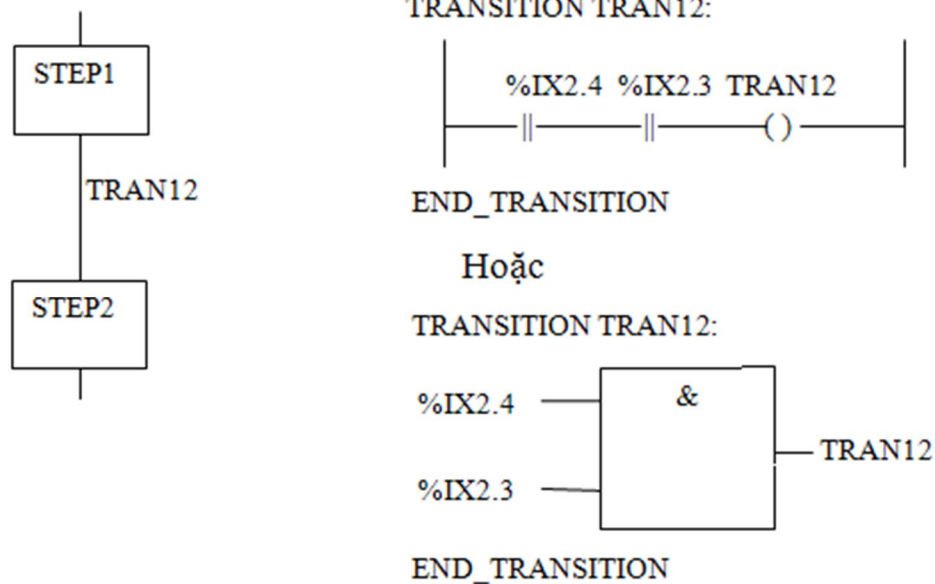
LD %IX2.4

AND %IX2.3

END_TRANSITION

STEP STEP2: END_STEP

- Dùng tên chuyển (TRANSITION name) ở bên phải đường nối thẳng đứng với vai trò như là bộ nhận dạng cho chuyển, bộ này giống như cấu trúc TRANSITION ...END_TRANSITION cho kết quả logic vào biến có tên chuyển để làm điều kiện chuyển từ : một mạng LD hay FBD hoặc các chỉ dẫn viết theo IL hoặc gán biểu thức logic viết theo ST như dưới đây:



Hình 4.5. Chuyển bằng tên và lập trình bằng LD hoặc FBD

Hoặc viết bằng IL cho tên chuyển:

TRANSITION TRAN12:

LD %IX2.4

AND %IX2.3

END_TRANSITION

Hoặc viết bằng ST cho tên chuyển:

TRANSITION TRAN12

:= %IX2.4 & %IX2.3 ;

END_TRANSITION

Chuyển chỉ có ý nghĩa cục bộ trong phạm vi đơn vị tổ chức chương trình chứa chuyển đó. Số lượng chuyển cho phép trong một chương trình SFC và trong một bước phụ thuộc vào thiết bị thực tế.

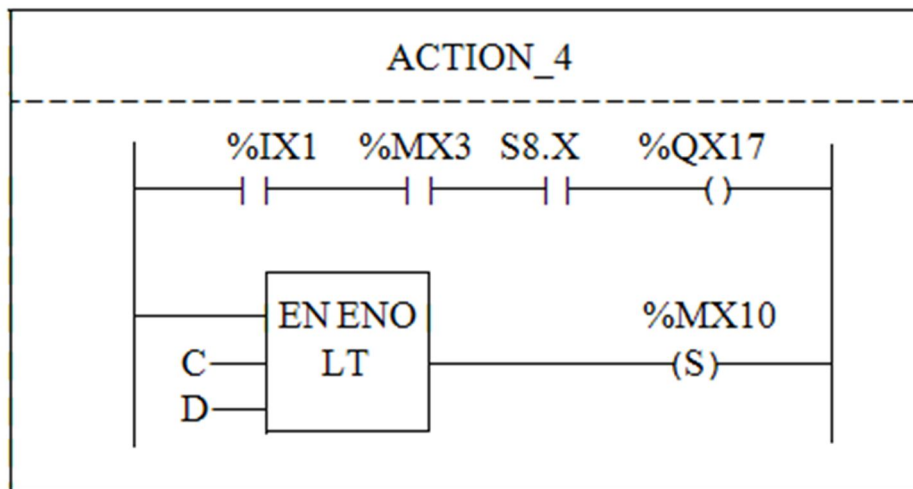
e. Tác động

Mỗi bước có thể không có hoặc có nhiều tác động, bước nhảy không có tác động

đi kèm được coi là có chức năng ĐỔI, nghĩa là chỉ chờ điều kiện chuyển đúng để chuyển sang bước tiếp theo. Một tác động có thể là một biến logic, một tập các lệnh của ngôn ngữ IL, hoặc một tập câu lệnh theo ngôn ngữ ST, hay một nấc thang của ngôn ngữ LD, hoặc các mạng FBD hoặc một mạng SFC. Việc khai báo tác động có thể theo các cách tương ứng với kiểu biến mà nó có thể như trên và chỉ có ý nghĩa trong phạm vi của khối tổ chức chương trình chứa nó. Ví dụ khai báo tác động:

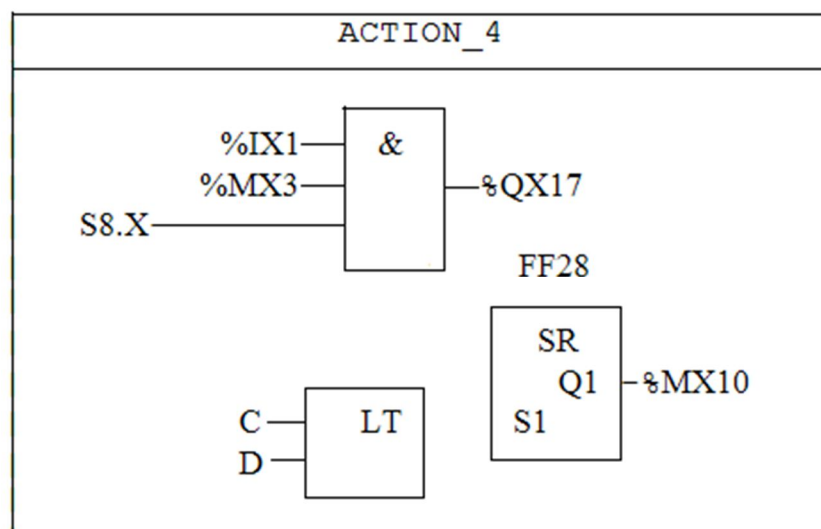
- Nếu dạng biến logic thì được khai báo trong khối VAR hoặc VAR_OUTPUT hoặc dạng đồ họa tương tự như thế.

- Dạng LD:



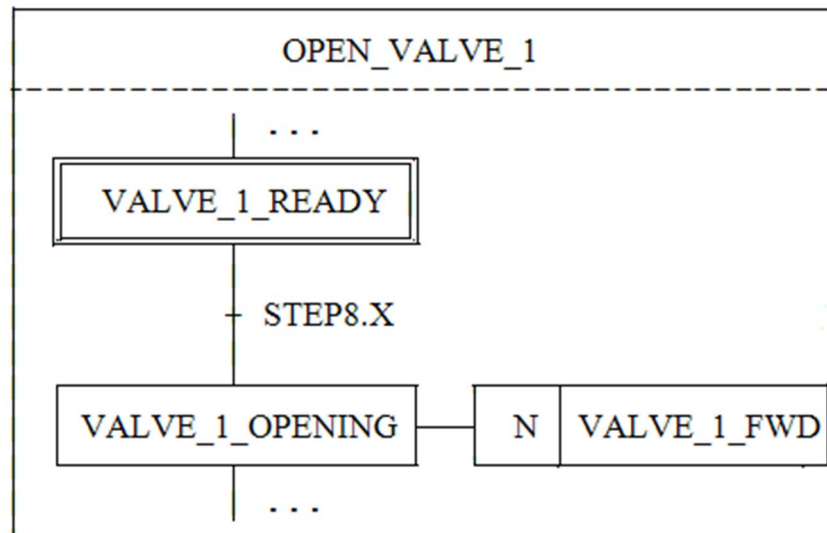
Hình 4.6. Dạng tác động viết bằng ngôn ngữ LD

- Dạng FBD:



Hình 4.7. Dạng tác động viết bằng ngôn ngữ FBD

- Ví dụ khai báo SFC:



Hình 4.8. Dạng tác động bằng SFC

- Khai báo dạng ST:

```

ACTION ACTION_4:
    %QX17 := %IX1 & %MX3 & S8.X ;
    FF28(S1 := (C<D));
    %MX10 := FF28.Q;
END_ACTION

```

- Khai báo dạng IL:

```

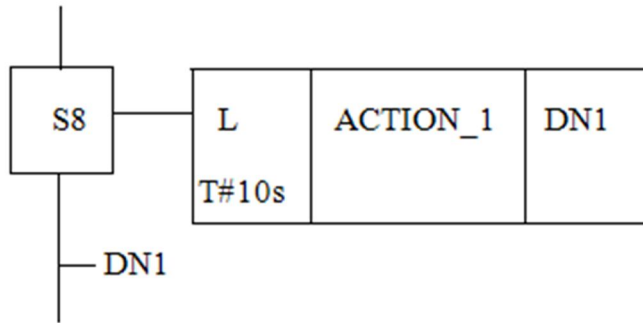
ACTION ACTION_4:
LD    S8.X
AND    %IX1
AND    %MX3
ST     %QX17
LD     C
LT     D
S1     FF28
LD     FF28.Q
ST     %MX10
END_ACTION

```

Trong ví dụ, cờ S8.X được dùng với mục đích là khi S8 không được kích hoạt thì cho %QX17:=0.

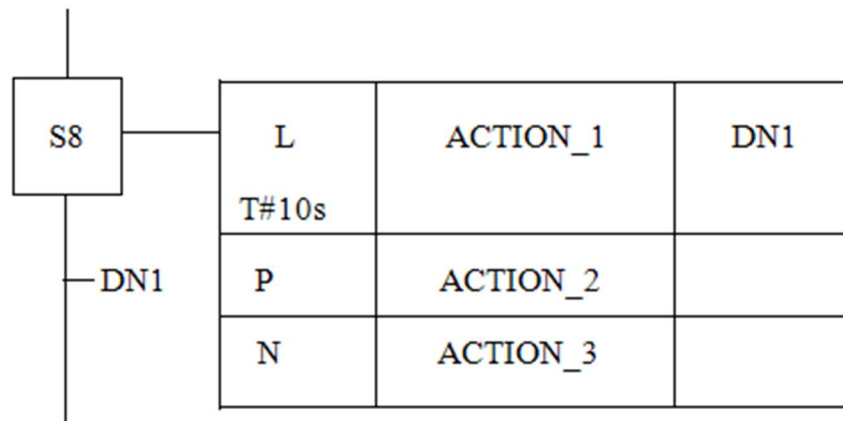
Một bộ điều khiển khả trình có cung cấp công cụ lập trình theo SFC sẽ có các cơ chế ghép các tác động với bước nhảy như sau:

- Khởi tác động về mặt logic hoặc vật lý được đi kèm sát cạnh bước nhảy, ví dụ:



Hình 4.9. Dạng khối tác động đi kèm bước

- Có nhiều khối tác động móc vào nhau đi kèm với bước, ví dụ:



Hình 4.10. Dạng nhiều khối tác động đi kèm bước

- Trong bước nhảy dạng ngôn ngữ, ví dụ:

STEP S8:

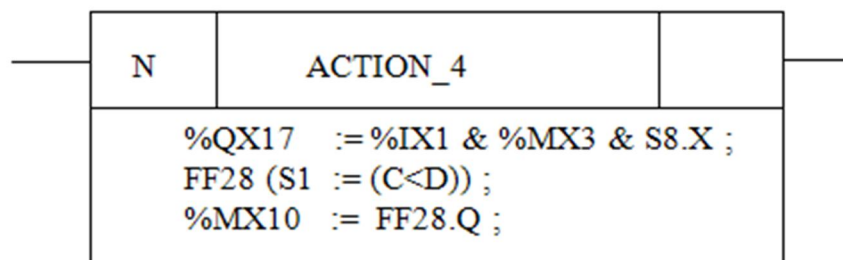
ACTION_1 (L, t#10, DN1);

ACTION_2 (P);

ACTION_3 (N);

END_STEP

- Dùng khối tác động, là phần tử đồ họa kết hợp biến logic cùng các tên đặc tính tác động để tạo điều kiện cho phép cho tác động xảy ra, ví dụ:



Hình 4. 11. Khối tác động kết hợp điều kiện đi kèm với bước

f. Các quy tắc tiến triển

Vị trí đầu tiên của mạng SFC được đặc trưng bởi bước nhảy đầu tiên, và được kích hoạt khi khởi động chương trình hay khối chức năng có chứa mạng đó. Việc tiến triển trạng thái tác động của các bước diễn ra theo đường nối trực tiếp các bước khi có các chuyển tác động. Chuyển được cho phép khi các bước phía trước nối với nó ở trạng thái hoạt động. Tác động của chuyển xảy ra khi chuyển được cho phép và điều kiện chuyển đúng, và nó làm cho các bước ngay trước chuyển hết tác động và nhường chỗ cho các bước ngay sau đó.

Việc thay đổi giữa bước nhảy sang chuyển rồi chuyển sang bước nhảy được duy trì trong việc kết nối các phân tử SFC, không bao giờ có hai bước nhảy nối trực tiếp mà phải qua một chuyển, và hai chuyển cũng phải có một bước nhảy chia tách ra. Một chuyển có thể làm cho nhiều bước phía sau được kích hoạt đồng thời, sau đó việc tiến triển của các chuỗi phía sau độc lập với nhau, người ta đưa ra khái niệm phân kì và hội tụ của các chuỗi và dùng hai đường nằm ngang song song (đường kép nằm ngang) khi chia ra hoặc hợp lại diễn ra đồng thời: nếu là phân kì (hay hội tụ) đồng thời thì chỉ có một chuyển và chuyển đó nằm trước (hoặc sau) đường kép nằm ngang tương ứng. Nếu phân kì theo kiểu lựa chọn thì các chuyển được đặt sau đường nằm ngang, mỗi chuyển là điều kiện cho bước tiếp theo tương ứng với nó, còn hội tụ lựa chọn thì các chuyển đặt trước đường nằm ngang. Trong trường hợp theo kiểu lựa chọn thì việc chuyển có thể thực hiện theo ưu tiên do người sử dụng mức ưu tiên cho các chuyển ở các nhánh, không phải tại một thời điểm có nhiều tác động chuyển được.

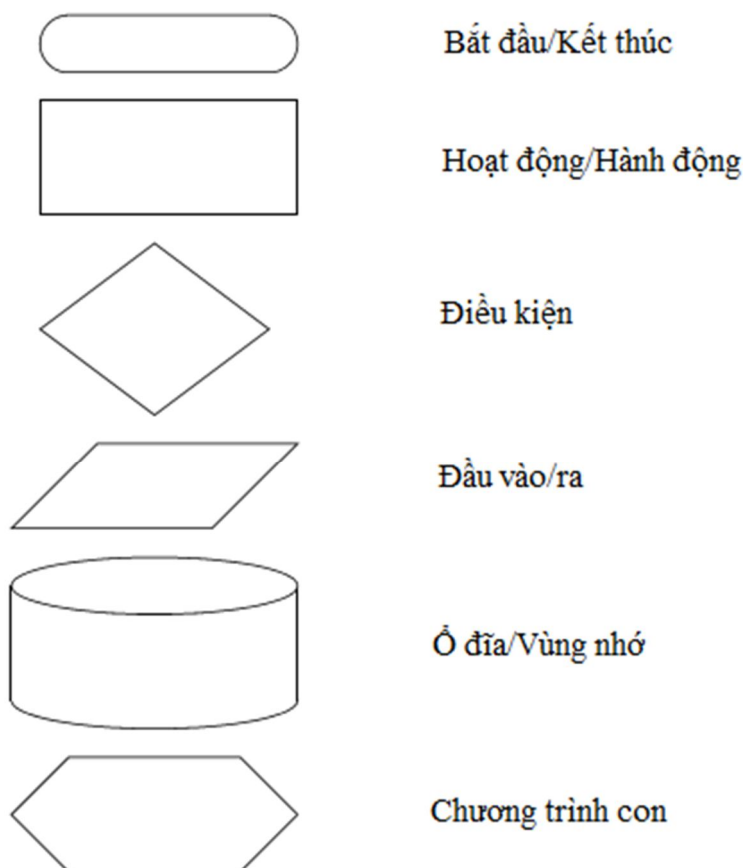
CHƯƠNG 5: KỸ THUẬT LẬP TRÌNH PLC

5.1 Thiết kế chương trình dựa vào lưu đồ

5.1.1 Giới thiệu về lưu đồ chương trình

Lưu đồ là sơ đồ đại diện cho một quá trình gồm các trạng thái (bước) liên tiếp của quá trình. Mỗi bước sẽ được thực hiện bởi một yêu cầu đơn giản có thể được coi là kết quả của một vài điều kiện đơn giản.

Các khối cơ bản trong lưu đồ:

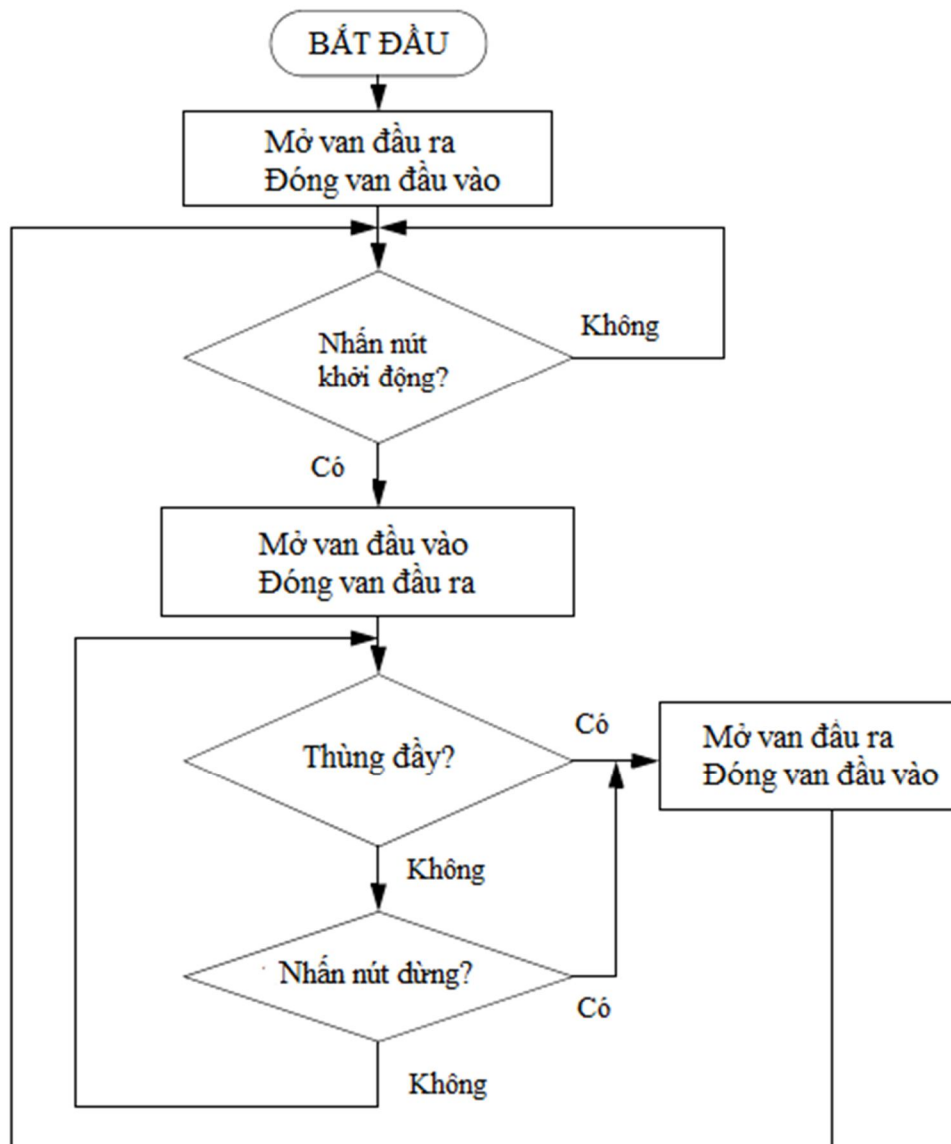


Hình 5.1: Ký hiệu trong lưu đồ

Ví dụ:

Lưu đồ thể hiện hệ thống điều khiển cho thùng nước với yêu cầu:

- Khi nhấn nút *khởi động* thùng nước bắt đầu được đổ đầy nước, van đầu ra được đóng lại.
- Khi thùng nước đầy, nút ấn *dừng* được ấn và van đầu ra được mở và van đầu vào đóng lại.



Hình 5.2: Lưu đồ thùng chứa

Hoạt động đầu tiên của hệ là *mở van đầu ra, đóng van đầu vào*. Tiếp theo là khối điều kiện được sử dụng để mô tả cho trạng thái chờ *nhấn nút khởi động*. Khi nút khởi động được nhấn, hệ sẽ đi theo hướng *Có*, van đầu vào sẽ mở và van đầu ra sẽ đóng. Lưu đồ sẽ đi vào một vòng lặp với hai khối điều kiện để chờ *thùng đầy* hoặc *nút dừng được ấn*. Nếu một trong hai trường hợp xảy ra đều sẽ dẫn tới hành động *đóng van đầu vào và mở van đầu ra*. Lưu đồ lại quay về trạng thái chờ nhấn nút khởi động một lần nữa.

Mục đích mô tả hệ thống nhờ lưu đồ chương trình:

- Hiểu sự hoạt động của hệ thống.
- Xác định các hoạt động cơ bản (được biểu diễn bởi các khối).
- Xác định chuỗi các hoạt động (trình tự được thực hiện bởi các mũi tên)

- Khi nào chuỗi hoạt động thay đổi nhờ sử dụng các khối điều kiện.

Khi lập được lưu đồ chương trình của hệ thống ta có thể chuyển lưu đồ sang ngôn ngữ lập trình giản đồ thang để lập trình cho PLC. Phần sau ta sẽ nghiên cứu việc chuyển từ lưu đồ chương trình sang ngôn ngữ giản đồ thang.

5.1.2 Chuyển từ lưu đồ chương trình sang giản đồ thang.

Để hiểu rõ việc chuyển từ lưu đồ chương trình sang giản đồ thang ta xét một ví dụ cụ thể là hệ thống điều khiển thùng chứa như trên.

Bước 1:

- Liệt kê đầu vào ra: Dựa vào yêu cầu công nghệ ta liệt kê tất cả các đầu vào ra cho PLC.

Bảng 5.1: Liệt kê đầu vào ra

STT	Tên tín hiệu	Ký hiệu	Vào/Ra		Số/Tương tự	
			Vào	Ra	Số	Tương tự
1	Nút khởi động	Start	1		1	
2	Nút dừng	Stop	1		1	
3	Cảm biến báo thùng đầy	Full	1		1	
4	Mở/đóng van đầu vào	Inlet		1	1	
5	Mở/đóng van đầu ra	Outlet		1	1	

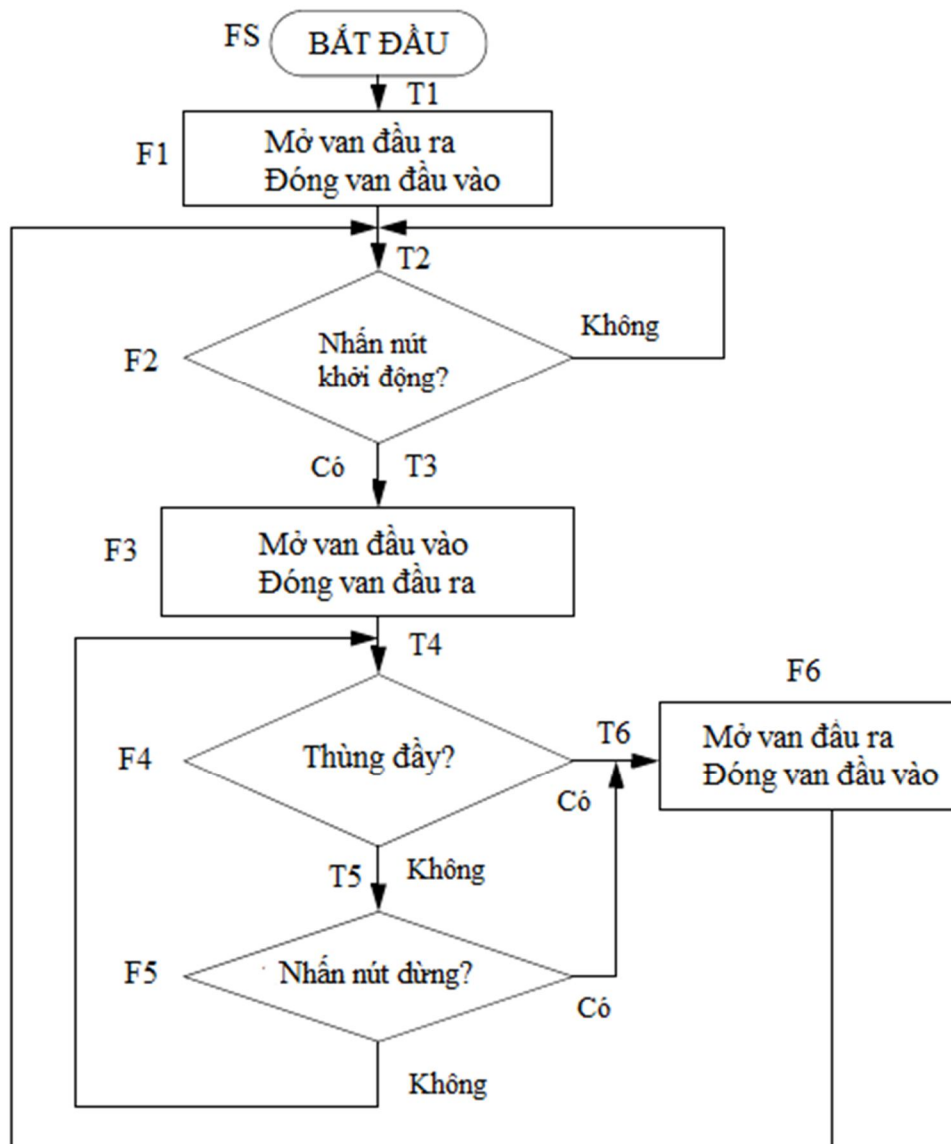
- Sử dụng ký hiệu đánh dấu tất cả các phần tử trong lưu đồ chương trình (khởi hoạt động, điều kiện, mũi tên chuyển hướng). Trong đó các phần tử thường được đánh dấu như sau:

Bảng 5.2: Một số ký hiệu đánh dấu thường gặp trong lưu đồ

STT	Các khối trong lưu đồ	Ký hiệu đánh dấu	Vị trí đánh dấu
1	Bắt đầu/kết thúc	FSi	Phía trên/bên phải
2	Hoạt động/hành động	Fi	Phía trên/bên phải
3	Điều kiện	Fi	Phía trên/bên phải
4	Chuyển tiếp giữa các khối (mũi tên)	Ti	Trên mũi tên chuyển tiếp

Các ký hiệu đánh dấu sẽ là các rơ le trung gian (nếu thực hiện bằng mạch rơ le tiếp điểm) hoặc các ô nhớ trong PLC (nếu lập trình sử dụng PLC).

Đối với lưu đồ trong ví dụ điều khiển thùng chứa ta sẽ đánh dấu như sau:



Hình 5.3: Lưu đồ điều khiển chương trình đã được đánh dấu

Bước 2: Giảm đồ thang cho các chuyển tiếp

Một yếu tố quan trọng trong lưu đồ điều khiển là các mũi tên chuyển tiếp trong lưu đồ (cũng là điều kiện chuyển trạng thái của hệ thống thực tế), và ta phải thực hiện chương trình hóa (giảm đồ thang) các chuyển tiếp.

Quy tắc: Mỗi chuyển tiếp được thực hiện khi có ít nhất một trong các đầu vào của nó là đúng. Tức là:

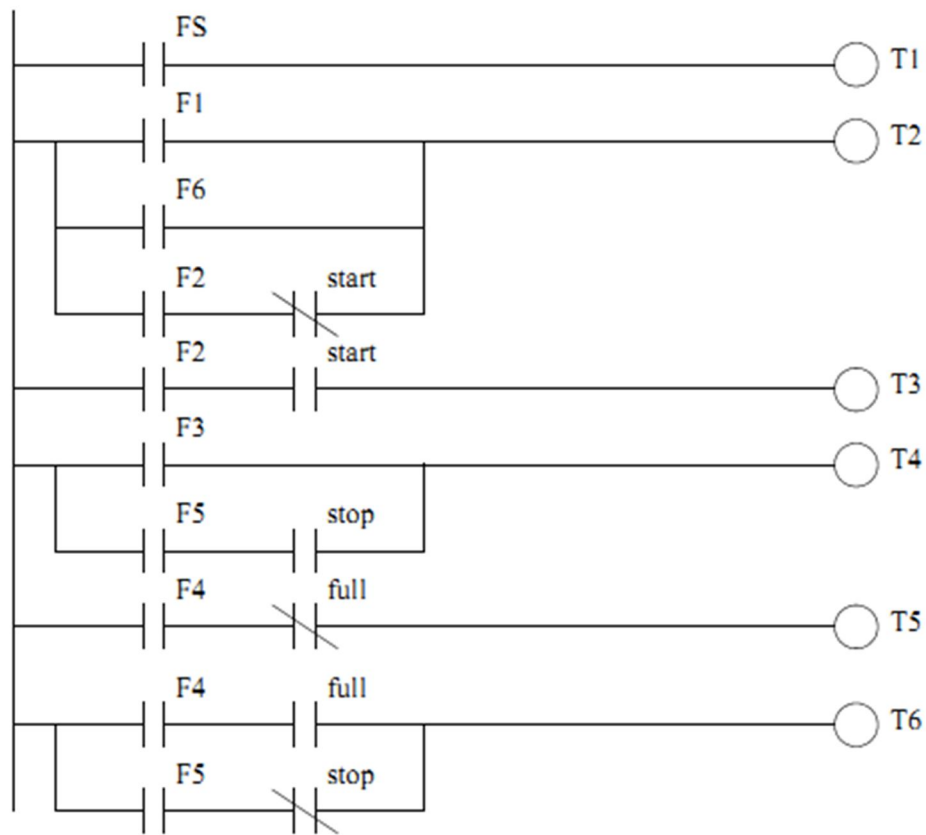
- Hàm đúng của các chuyển tiếp là logic hoặc của các đầu vào chuyển tiếp.
- Khi chuyển tiếp được thực hiện thì các đầu vào sẽ tự hủy (do chuyển sang trạng thái tiếp theo) nên chuyển tiếp sẽ tự cắt.

Theo quy tắc đó, với ví dụ trên ta có:

Bảng 5.3: Hàm logic của các chuyển tiếp

STT	Chuyển tiếp	Hàm logic
1	T1	$T1 = FS$
2	T2	$T2 = F1 \text{ or } F6 \text{ or } (F2 \text{ and Not Start})$
3	T3	$T3 = F2 \text{ and Start}$
4	T4	$T4 = F3 \text{ or } (F5 \text{ and Start})$
5	T5	$T5 = F4 \text{ and (not Full)}$
6	T6	$T6 = (F4 \text{ and Full}) \text{ or } (F5 \text{ and not Stop})$

Do đó giản đồ thang của các chuyển tiếp trong ví dụ điều khiển thùng chứa như sau:



Hình 5.4: Giản đồ thang cho các chuyển tiếp

Bước 3: Giản đồ thang cho các hoạt động, điều kiện

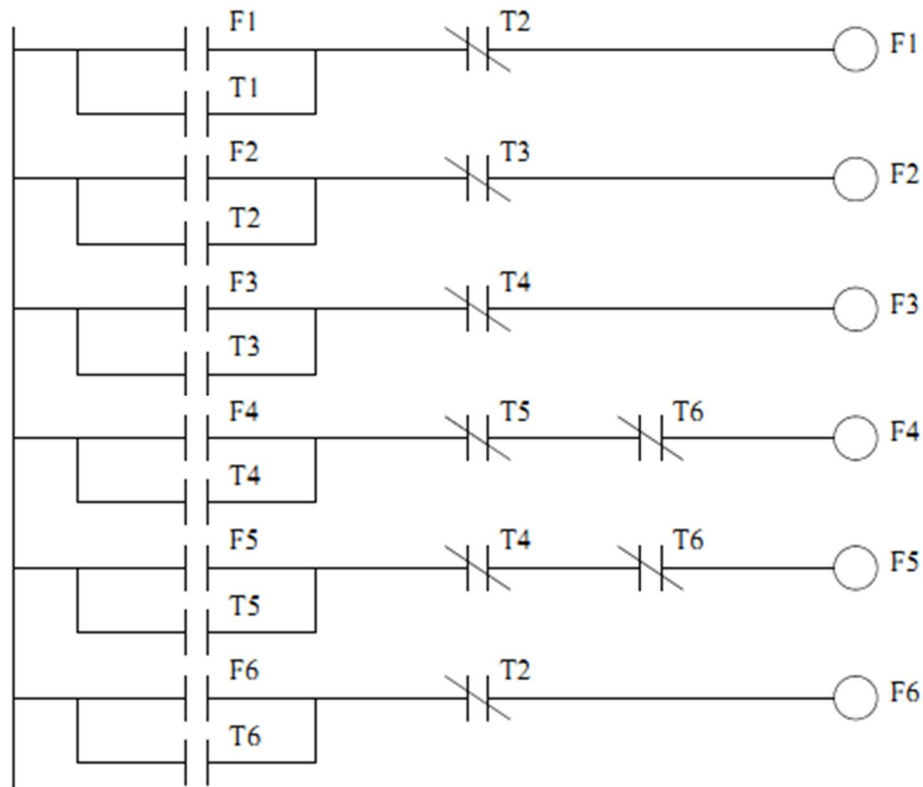
Quy tắc: một hoạt động hay điều kiện được kích hoạt khi các chuyển tiếp tới nó là đúng (sau đó trạng thái của nó phải tự duy trì) và được xóa bỏ khi chuyển tiếp đi ra từ nó là được thực hiện.

Vậy hàm đóng của các hoạt động là các chuyển tiếp tới nó.

Hàm cắt của các hoạt động là chuyển tiếp ra khỏi nó.

Ví dụ: Hoạt động F1 được kích hoạt khi chuyển tiếp T1 đúng và được hủy hoạt động bởi chuyển tiếp T2.

Từ đó ta có giản đồ thang cho các hoạt động của hệ thống như sau:



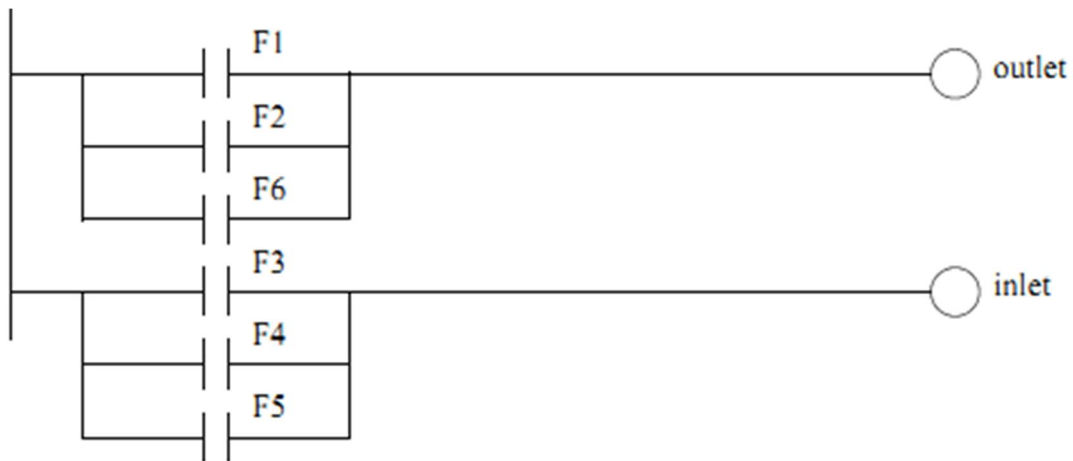
Hình 5.5: Giản đồ thang cho các hoạt động

Bước 4: Giản đồ thang cho các đầu vào ra

Từ các trạng thái hoạt động của hệ thống, ta đưa ra giản đồ thang cho các đầu ra theo quy tắc: *Các đầu ra được kích hoạt khi các hoạt động của hệ thống liên quan tới nó cũng được kích hoạt.*

Van đầu ra được kích hoạt khi các hoạt động: F1, F2, F6 được kích hoạt.

Tương tự ta có giản đồ thang cho các đầu ra như sau:



Hình 5.6: Giải đồ thang cho đầu ra

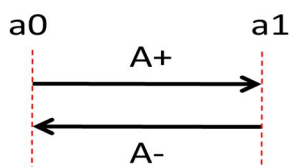
5.2 Thiết kế chương trình dựa vào trạng thái

Việc thiết kế chương trình dựa vào trạng thái chính là việc sử dụng phương pháp tổng hợp mạch logic tuần tự ma trận trạng thái. Các bước cụ thể như sau:

Bước 1

Tìm hàm logic của các biến trung gian, đầu vào đầu ra theo phương pháp ma trận trạng thái.

Xét lại ví dụ 3.1 chương 3



Theo phương pháp ma trận trạng thái ta đã có hàm logic như sau:

$$X = a1 + \overline{a0}X$$

$$A+ = \overline{X}.$$

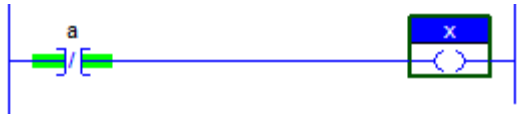
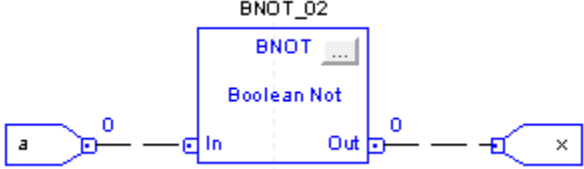
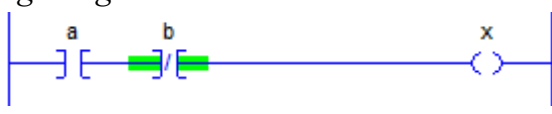
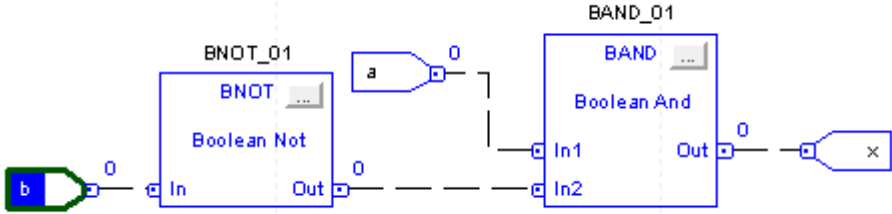
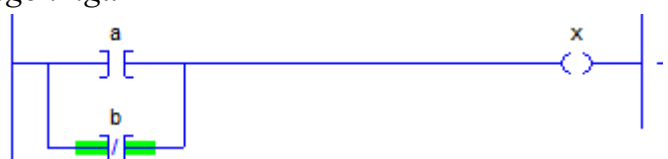
$$A- = X.$$

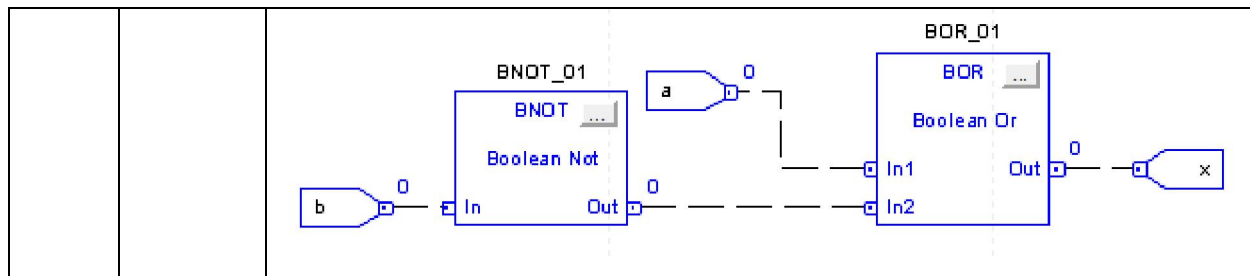
Bước 2

Chuyển các hàm logic sang ngôn ngữ PLC tương ứng.

Các hàm logic cơ bản và ngôn ngữ tương ứng trong PLC

STT	Hàm logic	Lệnh trong PLC
1	$x = \text{not } a$	<p>1. Ngôn ngữ IL</p> <p>LDN a (*Đưa giá trị a vào thanh ghi dịch*)</p> <p>ST x (*Ghi kết quả vào x*)</p> <p>2. Ngôn ngữ ST</p> <p>x := NOT a;</p>

		<p>3. Ngôn ngữ LD</p>  <p>4. Ngôn ngữ FBD</p> 
2	$x = a \cdot \bar{b}$	<p>1. Ngôn ngữ IL</p> <p>LD a (*Đưa giá trị a vào thanh ghi dịch*)</p> <p>ANDN b (*logic VÀ với \bar{b}*)</p> <p>ST x (*Ghi kết quả vào x*)</p> <p>2. Ngôn ngữ ST</p> <p>$x := a \& (\text{NOT } b);$</p> <p>3. Ngôn ngữ LD</p>  <p>4. Ngôn ngữ FBD</p> 
3	$x = a + \bar{b}$	<p>1. Ngôn ngữ IL</p> <p>LD a (*Đưa giá trị a vào thanh ghi dịch*)</p> <p>ORN b (*logic VÀ với \bar{b}*)</p> <p>ST x (*Ghi kết quả vào x*)</p> <p>2. Ngôn ngữ ST</p> <p>$x := a \text{ OR } (\text{NOT } b);$</p> <p>3. Ngôn ngữ LD</p>  <p>4. Ngôn ngữ FBD</p>



Với hàm logic đã có:

$$X = a1 + \overline{a}0X$$

$$A+ = \overline{X}.$$

$$A- = X.$$

Ta có các chương trình PLC như sau:

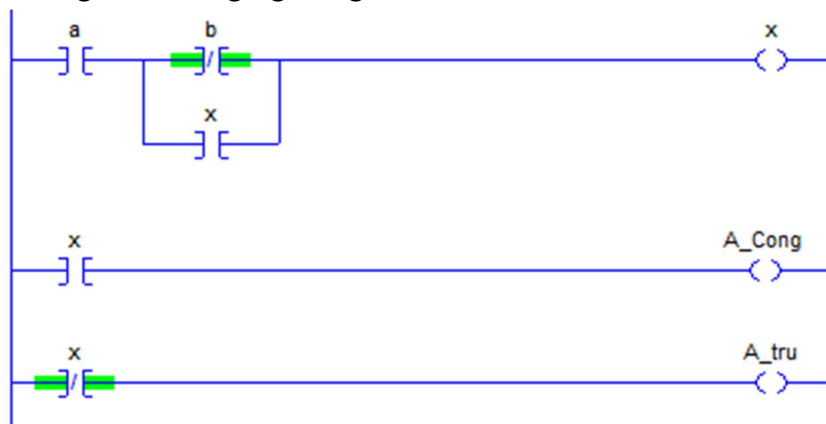
- Chương trình bằng ngôn ngữ IL

```
LDN  a0
AND  X
OR   a1
ST   X
LDN  X
ST   A_cong
LD   X
ST   A_tru
```

- Chương trình bằng ngôn ngữ ST

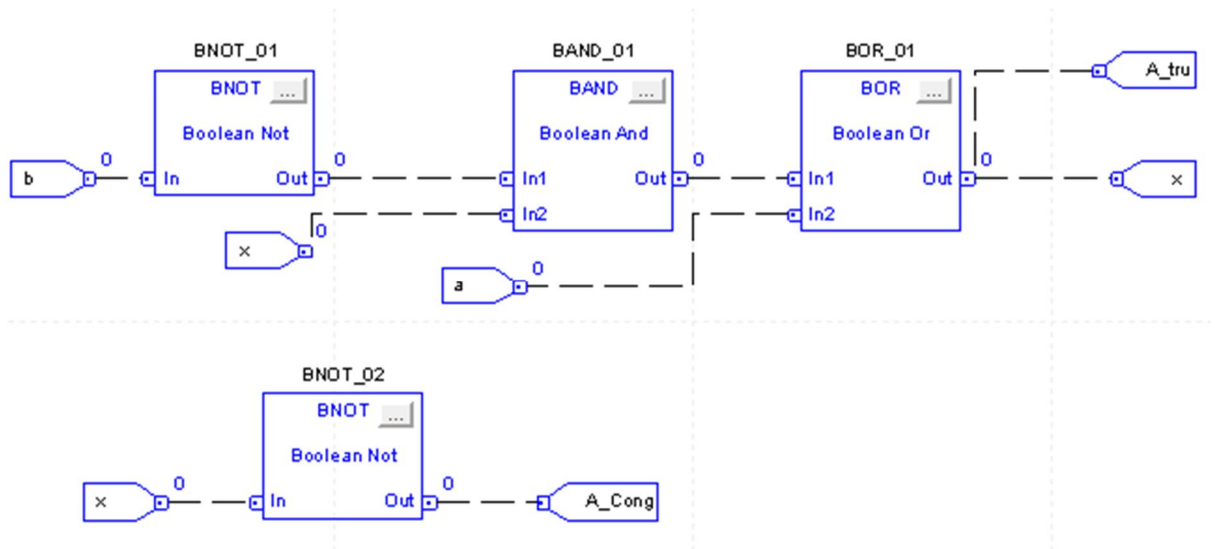
```
X := a1 OR (NOT a0 and X);
A_cong := X;
A_tru := NOT X;
```

- Chương trình bằng ngôn ngữ LD



Hình 5.7: Ví dụ ngôn ngữ lập trình LD

- Chương trình bằng ngôn ngữ FBD



Hình 5.8: Ví dụ ngôn ngữ lập trình FBD

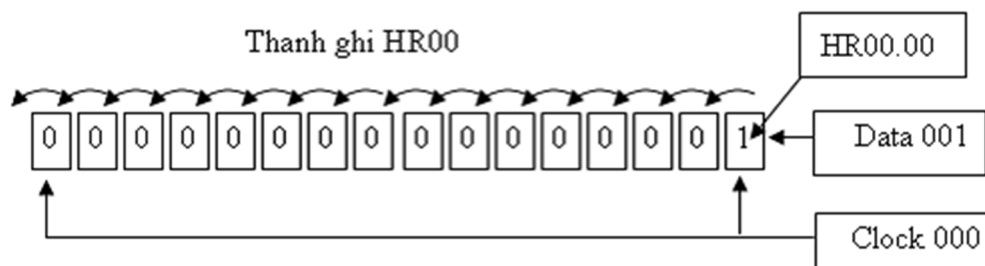
5.3 Kỹ thuật thanh ghi dịch

5.3.1 Giới thiệu về thanh ghi dịch trong PLC

Bộ ghi dịch của PLC có chức năng GHI số liệu vào ô nhớ dữ liệu và DỊCH các bit trong ô nhớ khi có xung điều khiển dịch bit. Do đó ta có thể ứng dụng thanh ghi dịch để thực hiện việc thiết kế chương trình điều khiển tuần tự:

- Mỗi bit sẽ tương ứng với một công đoạn trong chu trình công nghệ.
- Khi thực hiện xong sẽ xóa dữ liệu trong thanh ghi dịch để đưa hệ thống về trạng thái ban đầu.

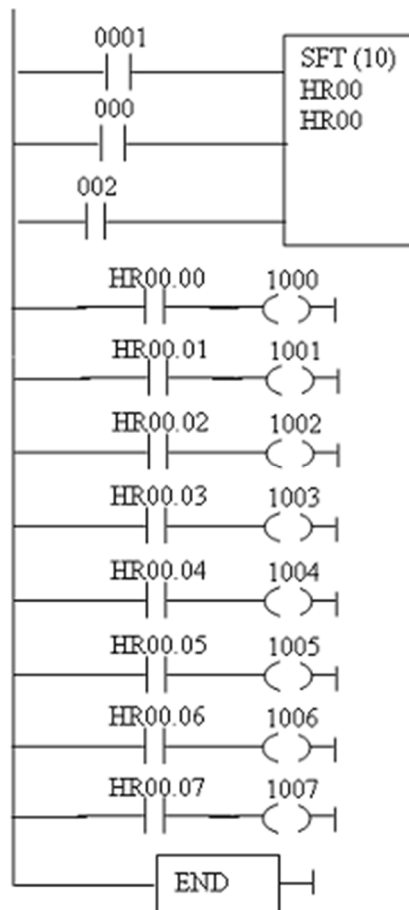
Ví dụ thanh ghi dịch của Omron



Hình 5.9: Thanh ghi dịch Omron

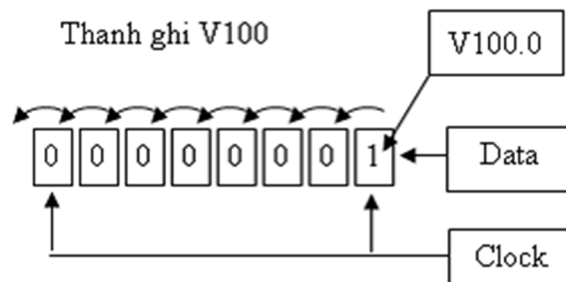
Khi đầu vào Data = 1 (On) và có xung clock thì dữ liệu sẽ đẩy vào ô nhớ thứ nhất của thanh ghi là HR00.00 và mỗi lần có xung clock sau đó dữ liệu sẽ đẩy vào các ô nhớ tiếp theo. Mỗi ô nhớ sẽ điều khiển một công đoạn trong quá trình công nghệ.

Lệnh giản đồ thang như sau:



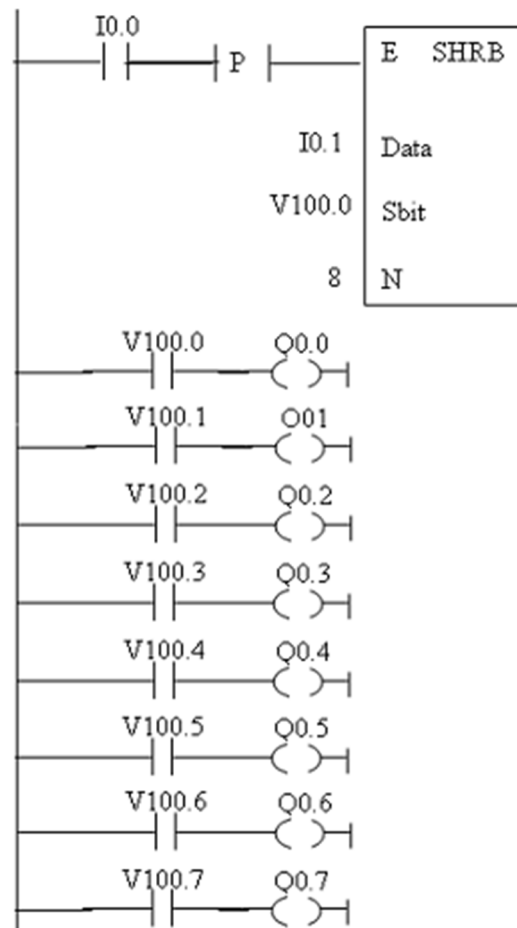
Hình 5.10: Lệnh thanh ghi dịch Omron

Tương tự cho thanh ghi dịch Siemens S7 200



Hình 5.11: Thanh ghi dịch Siemens S7 200

Lệnh thanh ghi dịch Siemens



Hình 5.12: Lệnh thanh ghi dịch Siemens S7 200

5.3.2 Thông số thanh ghi dịch thường dùng

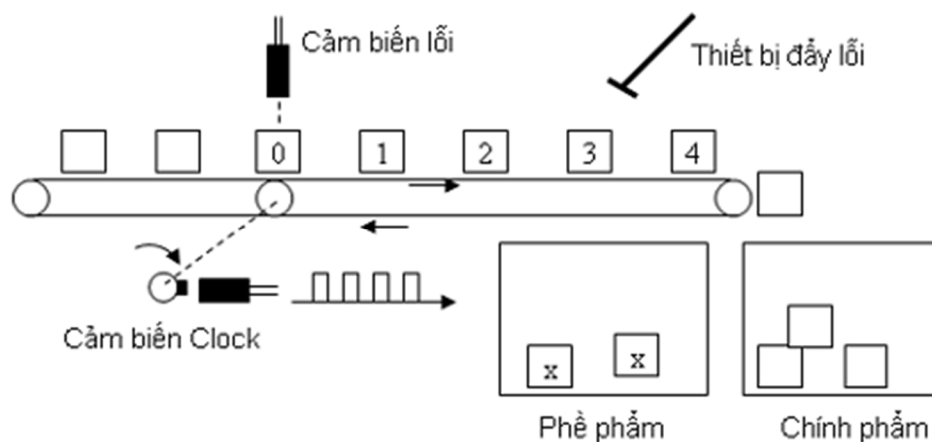
- Thông số thanh ghi dịch của PLC Omron CPM1
 - + Số lượng thanh ghi: 16 (HR00 đến HR15)
 - + Số lượng bit: 16
- Thông số thanh ghi dịch PLC S7 200
 - + Số lượng thanh ghi: Rất nhiều tùy thuộc vùng nhớ của PLC cụ thể
 - + Số lượng bit: tùy thuộc lệnh thanh ghi dịch sử dụng

Bảng 5.4: Các loại thanh ghi dịch của S7 200

STT	Lệnh thanh ghi	Số bit
1	SHRB (SHLB)	8
2	SHRW (SHLW)	16
3	SHRD (SHLD)	32

5.3.3 Ứng dụng thanh ghi dịch giải quyết một công nghệ cụ thể

- Công nghệ phân loại sản phẩm



Hình 5.13 Máy phân loại sản phẩm

Nguyên lý hoạt động:

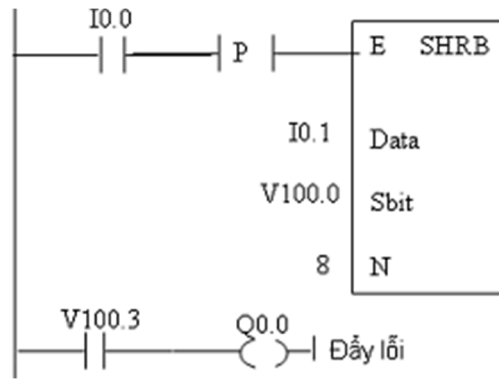
- + Cảm biến lỗi sẽ nhận biết chất lượng sản phẩm, nếu sản phẩm lỗi nó sẽ ON, và tín hiệu này sẽ được đưa vào thanh ghi dịch.
 - + Cảm biến clock dùng để phát xung clock vào thanh ghi dịch mỗi lần có xung clock sẽ đẩy sản phẩm di chuyển một bước trên dây truyền.
 - + Khi sản phẩm bị lỗi thanh ghi dịch sẽ ghi nhớ nó và tại vị trí của thiết bị đẩy lỗi nó sẽ bị đẩy ra khỏi dây truyền.
- Phân công vào ra cho PLC

Bảng 5.5: Bảng phân công vào ra cho PLC S7 200

STT	Tên tín hiệu	Loại tín hiệu	Đầu vào/Ra		Địa chỉ
			Vào	Ra	
1	Cảm biến lỗi	Số	1		I0.0
2	Cảm biến xung clock	Số	1		I0.1
3	Cần đẩy sản phẩm lỗi	Số		1	Q0.0

- Chương trình PLC S7 200 cho máy đẩy lỗi như sau:
 - + Tùy theo thiết kế cơ khí vị trí đẩy lỗi cách vị trí cảm biến báo lỗi là bao nhiêu xung clock mà ta sử dụng loại thanh ghi dịch nào (byte - 8 bit, word - 16 bit, hay DWord - 32 bit)
 - + Thông thường để đạt độ tin cậy cơ khí nên để nhỏ hơn 8 xung clock.

Chương trình giản đồ thang



Hình 5.14: Giản đồ thang chương trình máy phân loại sản phẩm.

Chương trình dưới dạng STL

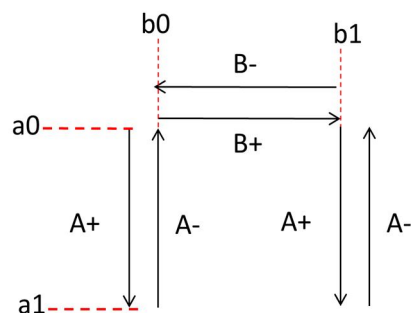
Network1		
LD	I0.0	//Đầu vào Cảm biến Clock
EU		//Lệnh vi phân sườn lên
SHRB	I0.1, V100.0, 8	//Bộ ghi dịch, I0.1 đầu vào cảm biến lỗi
Network2		
LD	V100.3	//Bit thứ 3 của V100
=	Q0.0	//Đầu ra đẩy lỗi

5.4 Sử dụng biểu đồ chức năng tuần tự SFC

Trong chương 3 (Mục 3.2.2) và chương 4 (Mục 4.4) ta đã nghiên cứu việc thiết kế chương trình bằng phương pháp Grafcet và ngôn ngữ lập trình SFC cho PLC. Trong phần này ta sẽ tìm hiểu cách thức thiết kế một chương trình cho một công nghệ cụ thể sử dụng biểu đồ chức năng tuần tự SFC.

Ví dụ ở đây ta nghiên cứu là ví dụ 2 mục 3.2.2 chương 3

Cho công nghệ như hình vẽ dưới đây



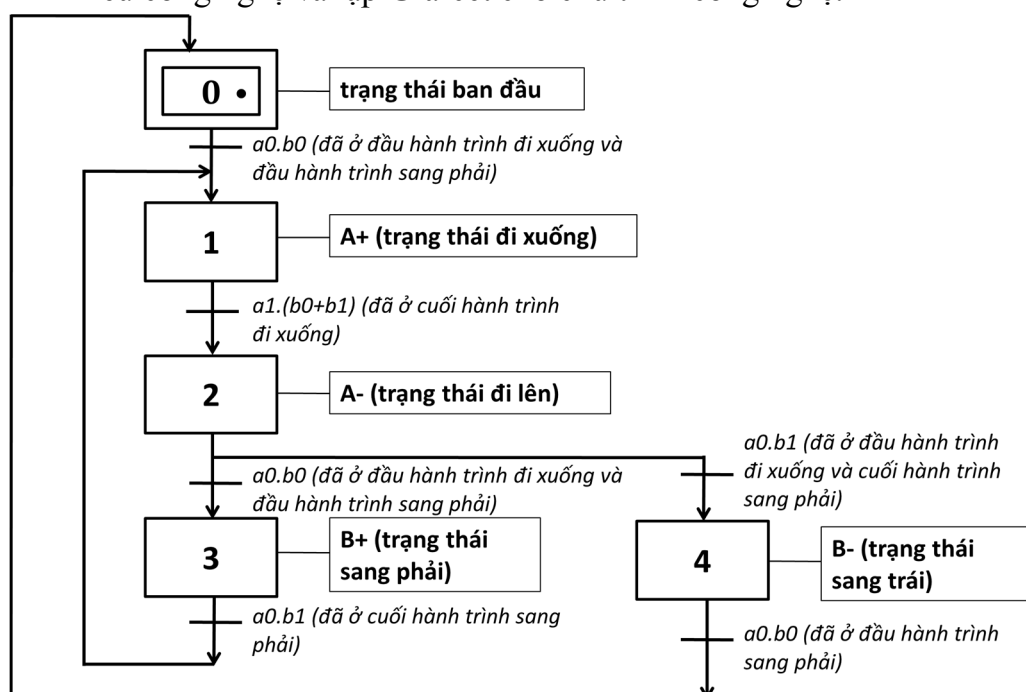
Hình 5.15: Công nghệ khoan 2 lỗ

Hệ thống gồm 2 cơ cấu chuyển động lên-xuống và phải-trái. Đầu tiên cơ cấu lên xuống sẽ thực hiện chuyển động đi xuống (cơ cấu chuyển động phải-trái đứng im). Khi gặp cảm biến a1 thì sẽ thực hiện chuyển động lên (cơ cấu chuyển động phải-trái vẫn đứng im). Khi gặp cảm biến a0 thì cơ cấu lên-xuống dừng, cơ cấu phải-trái thực

hiện chuyển động sang phải và mang theo cả cơ cấu lên-xuống. Khi gặp cảm biến b1 cơ cấu phải-trái dừng và cơ cấu lên xuống hoạt động thực hiện chuyển động xuống thì thực hiện chuyển động sang trái. Khi gặp cảm biến a1 thì sẽ thực hiện chuyển động lên (cơ cấu chuyển động phải-trái vẫn đứng im). Khi gặp cảm biến a0 thì cơ cấu lên-xuống dừng, cơ cấu phải-trái thực hiện chuyển động sang trái và mang theo cả cơ cấu lên-xuống. Khi gặp b0 thì cơ cấu phải trái dừng và cơ cấu lên xuống thực hiện chuyển động đi xuống và chu trình sẽ được lặp lại.

Bước 1:

Tìm hiểu công nghệ và lập Grafcet cho chu trình công nghệ.



Hình 5.16: Grafcet của công nghệ khoan 2 lỗ

Bước 2: Liệt kê đầu vào ra cho PLC

Bảng 5.6: Đầu vào ra cho công nghệ khoan 2 lỗ

STT	Tên tín hiệu	Loại tín hiệu	Đầu vào/Ra		Địa chỉ
			Vào	Ra	
1	Nút ấn khởi động	Số	1		
2	Nút ấn dừng	Số	1		
3	Công tắc hành trình (CTHT) a0	Số	1		
4	Công tắc hành trình (CTHT) a1	Số	1		
5	Công tắc hành trình (CTHT) b0	Số	1		
6	Công tắc hành trình (CTHT) b1	Số	1		
7	Đầu ra điều khiển xi lanh A đi xuống A+	Số		1	
8	Đầu ra điều khiển xi lanh A đi	Số		1	

	lên A-				
9	Đầu ra điều khiển xi lanh B sang trái B+			1	
10	Đầu ra điều khiển xi lanh B sang phải B-			1	

Bước 3: Lựa chọn PLC và lập trình điều khiển

- Dựa vào số lượng đầu vào ra ta chọn PLC và chú ý loại PLC phải hỗ trợ ngôn ngữ SFC. Ở đây ta sử dụng PLC của hãng Allen Bradley và lập trình bằng RSLogic5000.

- Gắn địa chỉ cho các tín hiệu vào ra

Bảng 5.7: Phân công vào ra cho PLC

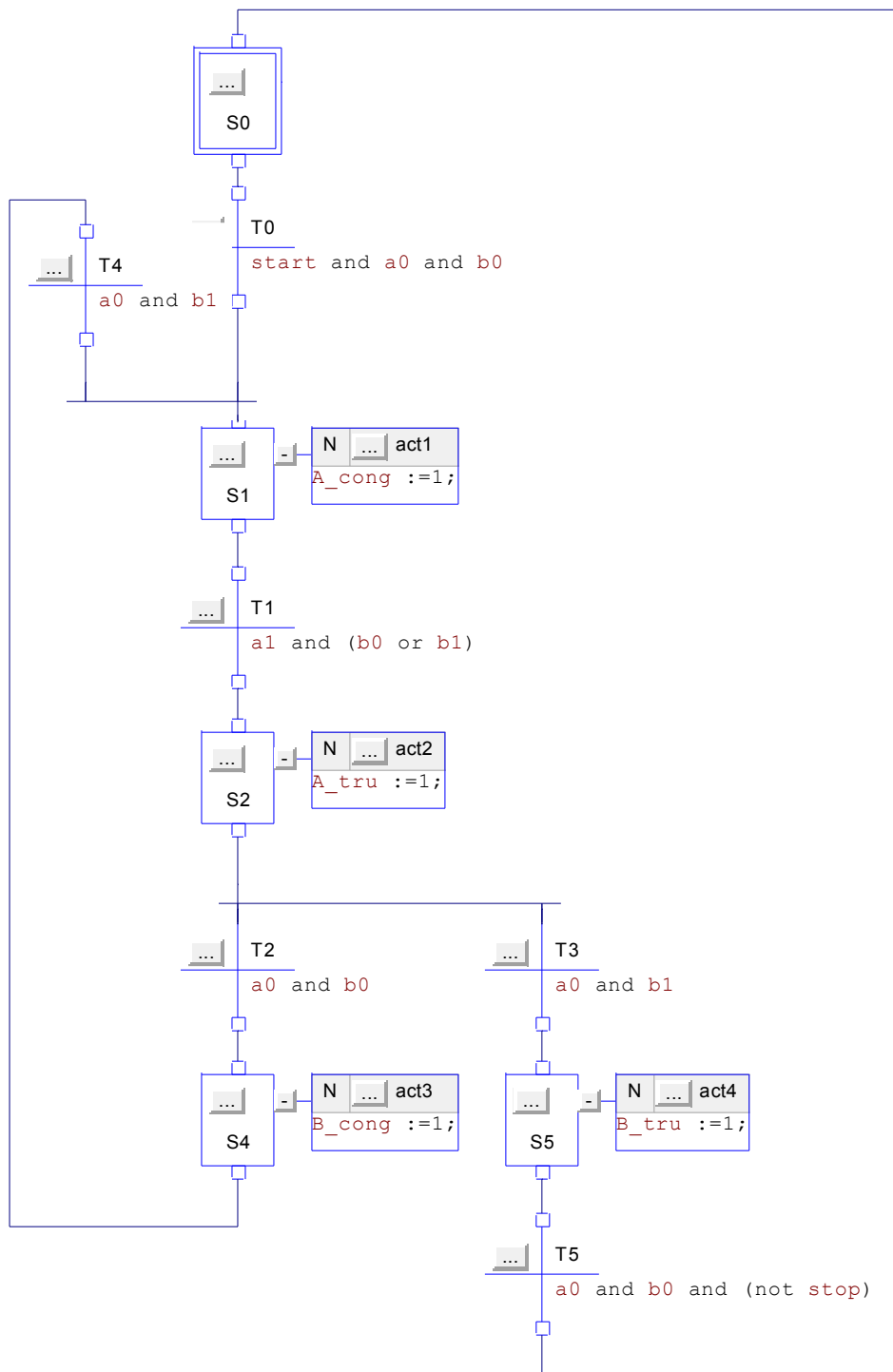
STT	Tên tín hiệu	Loại tín hiệu	Đầu vào/Ra		Địa chỉ
			Vào	Ra	
1	Nút ấn khởi động	Số	1		Local:2:I.Data.0
2	Nút ấn dừng	Số	1		Local:2:I.Data.1
3	Công tắc hành trình (CTHT) a0	Số	1		Local:2:I.Data.2
4	Công tắc hành trình (CTHT) a1	Số	1		Local:2:I.Data.3
5	Công tắc hành trình (CTHT) b0	Số	1		Local:2:I.Data.4
6	Công tắc hành trình (CTHT) b1	Số	1		Local:2:I.Data.5
7	Đầu ra điều khiển xi lanh A đi xuống A+	Số		1	Local:3:O.Data.0
8	Đầu ra điều khiển xi lanh A đi lên A-	Số		1	Local:3:O.Data.1
9	Đầu ra điều khiển xi lanh B sang trái B+	Số		1	Local:3:O.Data.2
10	Đầu ra điều khiển xi lanh B sang phải B-	Số		1	Local:3:O.Data.3

Gắn tag cho PLC và các trạng thái, chuyển tiếp của Grafcet (phần mềm RSLogic5000):

Scope: MainProgram Show... Show All					
	Name	Alias For	Base Tag	Data Type	Style
	start	Local:2:I.Data.0(C)	Local:2:I.Data.0(C)	BOOL	Decimal
	stop	Local:2:I.Data.1(C)	Local:2:I.Data.1(C)	BOOL	Decimal
	a0	Local:2:I.Data.2(C)	Local:2:I.Data.2(C)	BOOL	Decimal
	a1	Local:2:I.Data.3(C)	Local:2:I.Data.3(C)	BOOL	Decimal
	b0	Local:2:I.Data.4(C)	Local:2:I.Data.4(C)	BOOL	Decimal
	b1	Local:2:I.Data.5(C)	Local:2:I.Data.5(C)	BOOL	Decimal
	A_cong	Local:3:O.Data.0(C)	Local:3:O.Data.0(C)	BOOL	Decimal
	A_tru	Local:3:O.Data.1(C)	Local:3:O.Data.1(C)	BOOL	Decimal
	B_cong	Local:3:O.Data.2(C)	Local:3:O.Data.2(C)	BOOL	Decimal
	B_tru	Local:3:O.Data.3(C)	Local:3:O.Data.3(C)	BOOL	Decimal
	S0			SFC_STEP	
	T0			BOOL	Decimal
	S1			SFC_STEP	
	act1			SFC_ACTION	
	T1			BOOL	Decimal
	act2			SFC_ACTION	
	S2			SFC_STEP	
	T2			BOOL	Decimal
	T3			BOOL	Decimal
	S4			SFC_STEP	
	act3			SFC_ACTION	
	S5			SFC_STEP	
	T4			BOOL	Decimal
	act4			SFC_ACTION	
	T5			BOOL	Decimal

Hình 5.19: Địa chỉ vào/ra và các trạng thái, chuyển tiếp

- Chuyển từ Grafcet sang ngôn ngữ SFC:



Hình 5.18: Chương trình PLC bằng ngôn ngữ SFC của công nghệ khoan 2 lỗ

CHƯƠNG 6

GHÉP NỐI VÀ TRUYỀN THÔNG VỚI PLC

6.1 Các thiết bị vào ra

Trong hệ thống trang bị điện công nghiệp nói chung và sử dụng PLC nói riêng, có rất nhiều thiết bị khác nhau. Trong đó có các thiết bị cũng cấp tín hiệu vào: nút nhấn, công tắc hành trình, các cảm biến ... ; các thiết bị ra: đèn báo, rơ le, LED 7 thanh ... và các thiết bị khác ghép nối với PLC: biến tần, HMI, các bộ biến đổi tín hiệu trung gian ... Để có thể thực hiện ghép nối chính xác các thiết bị vào/ra với PLC, người sử dụng cần nắm rõ về nguyên lý hoạt động, dạng tín hiệu vào/ra của thiết bị và sơ đồ nguyên lý đầu tín hiệu vào/ra của PLC. Phần này sẽ trình bày về một số thiết bị vào/ra cơ bản.

Các thiết bị vào

a. Công tắc và nút nhấn

Công tắc và nút nhấn là hai thiết bị vào cơ bản và phổ biến nhất trong các hệ thống điều khiển. Trong các hệ thống không sử dụng các phần tử điều khiển như PLC, dòng điện và điện áp lớn được đặt trực tiếp vào thiết bị. Tuy nhiên trong hệ thống điều khiển có PLC, các công tắc và nút nhấn hầu hết được nối vào PLC và chỉ có dòng điện nhỏ chạy qua. Công tắc và nút nhấn cung cấp các tín hiệu ra dạng tiếp điểm. Điều này giúp thuận lợi và dễ dàng cho việc ghép nối với các thiết bị khác.



Hình 6.1: Nút ấn

Ký hiệu của các loại nút ấn trong bản vẽ như trong bảng 6.1

Bảng 6.1: Ký hiệu nút ấn

Mô tả	Loại	US/Canada	Quốc tế/Anh	Đức
Nút nhấn (Push button)	Thường hở (NO)			
	Thường kín (NC)			

Công tắc (Switch)	2 vị trí		
	3 vị trí		

b. Công tắc hành trình: (cơ khí, từ tính, quang điện)

Trong quá trình hoạt động của các hệ thống công nghiệp, có nhiều cơ cấu phải thực hiện các chuyển động. Để xác định quá trình và vị trí của các chuyển động đó, cần có các công tắc hành trình. Các công tắc hành trình có thể đặt tại vị trí đầu, cuối của hành trình hoặc đặt tại các vị trí giữa hành trình để có thể nhận biết và điều khiển cơ cấu theo yêu cầu công nghệ. Công tắc hành trình được chế tạo theo nhiều nguyên tắc. Tuy nhiên, ba nguyên tắc chính dùng trong công tắc hành trình là: cơ khí, từ tính, quang điện.

Công tắc hành trình được chế tạo theo nguyên tắc cơ khí bao gồm 2 phần chính: cơ cấu tác động và tiếp điểm. Cơ cấu tác động được chế tạo theo các hình dạng, kích thước khác nhau để phù hợp với các vị trí ứng dụng khác nhau. Tín hiệu ra của công tắc hành trình là tiếp điểm. Tiếp điểm có 3 loại: thường kín (NC), thường hở (NO), cặp tiếp điểm (NO+NC có chung 1 chân COM). Trong công tắc hành trình có thể có 1 tiếp điểm, 2 tiếp điểm hoặc 1 cặp tiếp điểm.



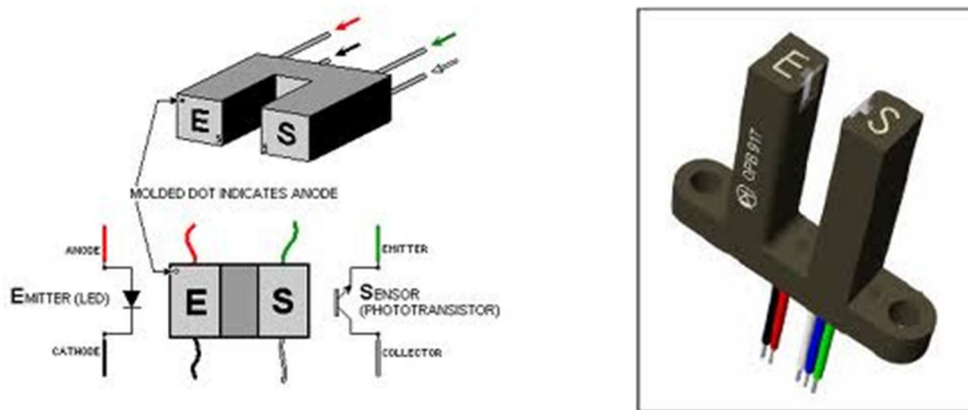
Hình 6.2: Một số loại công tắc hành trình theo nguyên tắc cơ khí

Công tắc hành trình theo nguyên tắc từ tính gồm 2 phần: nam châm vĩnh cửu và công tắc từ (tiếp điểm). Nam châm vĩnh cửu thường được lắp trên cơ cấu thụ động, phần công tắc từ lắp tại cơ cấu gắn với hệ thống điều khiển. Công tắc hành trình theo nguyên tắc từ tính không gây cản trở cho sự di chuyển của cơ cấu.



Hình 6.3: Một số loại công tắc từ trong thực tế.

Công tắc hành trình theo nguyên tắc quang điện được cấu tạo gồm phần tử phát và phần tử thu. Hai phần tử có thể được thiết kế đối diện nhau. Tín hiệu ra nằm trong phần tử thu và phần tử thường được sử dụng là phototransistor.



Hình 6.4: Nguyên lý và hình dạng CTHT theo nguyên tắc quang điện.

Trên cơ cấu chấp hành được gắn 1 phần tử đặc biệt mà khi di chuyển đến vị trí đặt CTHT, phần tử đó sẽ che hoặc để hở bên phát và bên thu tạo ra sự thay đổi tín hiệu.

c. Cảm biến tiệm cận: (quang điện, cảm ứng từ, điện dung)

Cảm biến tiệm cận là thiết bị dùng để xác định vị trí của cơ cấu chấp hành hoặc sản phẩm, vật liệu với khoảng cách nhất định. Đối với cảm biến tiệm cận, các nguyên tắc thường được sử dụng là quang điện, cảm ứng từ, điện dung. Đầu ra của cảm biến tiệm cận có thể chọn loại đầu ra tiếp điểm hoặc đầu ra transistor. Một số loại cảm biến tiệm cận cho phép cài đặt khoảng cách tác động và dừng tác động.



Hình 6.5: Hình ảnh cảm biến tiệm cận cảm ứng từ và điện dung

d. Cảm biến tốc độ: (phát tốc, quang điện - encoder)

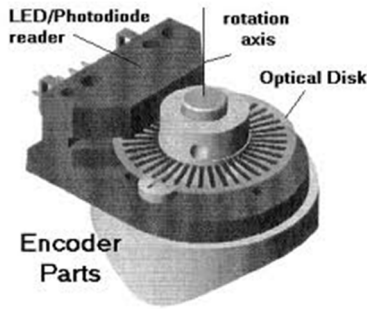
Trong các hệ thống sản xuất, cơ cấu quay rất phổ biến. Để thực hiện cơ cấu này có thể dùng nhiều phương án khác nhau như động cơ khí nén, động cơ điện. Cũng như các cơ cấu di chuyển khác, các cảm biến vị trí cũng được sử dụng. Tuy nhiên, trong công nghệ điều khiển, để đảm bảo an toàn, độ ổn định, chính xác, hầu hết các cơ cấu quay cần được xác định tốc độ. Vì vậy, thiết bị để đo và phản hồi tốc độ của cơ cấu rất quan trọng. Hai thiết bị chủ yếu được dùng trong công nghiệp là phát tốc và encoder.



Hình 6.6: Hình ảnh phát tốc và encoder

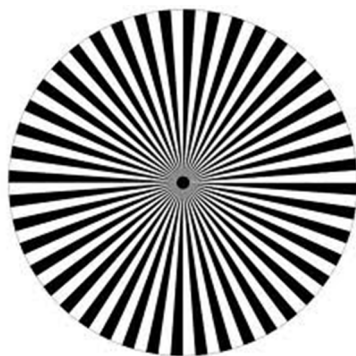
Với hình dạng bên ngoài, phát tốc và encoder không khác nhau nhiều vì cùng được chế tạo để dễ dàng lắp ráp đo tốc độ cơ cấu quay. Tuy nhiên, thường encoder sẽ nhỏ và nhẹ hơn do cấu tạo. Phát tốc có cấu tạo như động cơ điện một chiều và cho tín hiệu ra điện áp tỉ lệ với tốc độ động cơ. Vì vậy, tín hiệu từ phát tốc được sử dụng với các module tương tự.

Encoder là thiết bị cho tín hiệu ra dạng xung. Encoder có 2 loại: encoder tuyệt đối và encoder tương đối. Encoder tuyệt đối thường dùng đo vị trí cơ cấu quay. Encoder tương đối dùng đo tốc độ và cũng có thể xác định vị trí cơ cấu quay. Tuy nhiên, cấu tạo cơ bản của encoder luôn gồm 2 phần : đĩa quay và phần tử quang điện. Đĩa quay sẽ được đặt vào giữa phần phát và thu của phần tử quang điện. Trên đĩa quay sẽ có các rãnh (hoặc vạch đen). Khi đĩa quay hoạt động, phần thu trên phần tử quang điện sẽ thay đổi trạng thái và tạo ra tín hiệu xung. Ngoài ra encoder còn có các thành phần khác : trục quay, vỏ và có thể có mạch điện tử để ổn định tín hiệu xung ra.

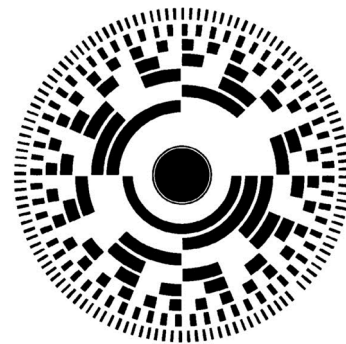


Hình 6.7 Cấu tạo cơ bản của encoder

Encoder tương đối có đĩa quay làm bằng kim loại hoặc nhựa và được khắc các rãnh hoặc vạch cách đều trên đó. Số lượng các vạch thể hiện độ phân giải của encoder. Ví dụ : đĩa quay có 100 vạch \Rightarrow encoder có độ phân giải 100 xung/vòng (100 ppr – 100 pulse per round). Khác với encoder tương đối, encoder tuyệt đối có đĩa quay được khắc tín hiệu mã hóa nhị phân trên đó. Điều này cũng quyết định độ phân giải của encoder. Ví dụ : trên đĩa được khắc 10 vòng khác nhau thì sẽ có độ phân giải $2^{10} = 1024$ vị trí/vòng.



a. Encoder tương đối



b. Encoder tuyệt đối

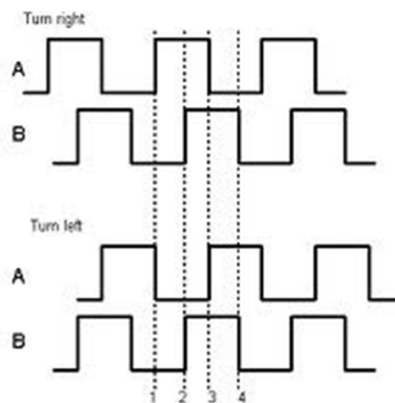
Hình 6.8: Đĩa quay của encoder

a. Encoder tương đối

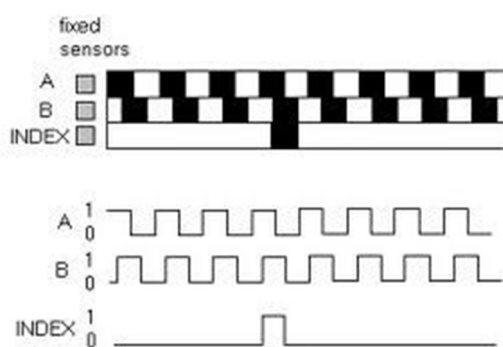
b. Encoder tuyệt đối

Encoder tuyệt đối có số đầu dây ra bằng số bit được mã hóa trên đĩa quay và dây nối cấp nguồn. Encoder tương đối thì thường có loại 3 dây 1 đầu ra ; 4 dây 2 đầu ra ; 5 dây 3 đầu ra (vì luôn có 2 chân cấp nguồn). Các tín hiệu ra được kí hiệu là A, B, C. Loại encoder 1 tín hiệu ra chỉ cho biết tốc độ quay mà không thể xác định được chiều quay của cơ cấu. Đồng thời chỉ cho phép tăng độ phân giải lên 2 lần khi đọc cả sườn lên và sườn xuống của xung ra. Ví dụ : lắp encoder có thông số 500ppr thì có thể đọc được 1000 xung trong 1 vòng quay của encoder. Với encoder 2 tín hiệu ra A và B, dựa vào góc lệch pha giữa 2 tín hiệu, có thể xác định chiều quay của encoder từ đó xác định chiều chuyển động của cơ cấu. Đồng thời, nếu sử dụng chế độ đọc 2 xung và đọc đủ sườn lên, sườn xuống thì độ phân giải tăng 4 lần. Đây là loại encoder được sử

dụng phổ biến nhất. Tuy nhiên, nhược điểm của loại 1 tín hiệu và 2 tín hiệu ra là không xác định vị trí lấy làm mốc. Vì vậy cần có thêm công tắc hành trình hoặc cảm biến định vị trí gốc nếu sử dụng trong hệ thống cần xác định vị trí. Encoder 5 dây 3 tín hiệu ra cho phép xác định vị trí gốc bằng cách sử dụng thêm 1 tín hiệu C.



Hình 6.10: Xung ra của encoder 2 tín hiệu ra



Hình 6.11: Xung ra của encoder 3 tín hiệu ra

6.1.1 Các thiết bị ra

a. Đèn báo

Đèn báo là một trong các thiết bị ra phổ biến nhất trong hệ thống trang bị điện. Các đèn báo thường được lắp đặt tại vị trí vận hành, tủ điều khiển để giúp nhân viên vận hành nhận biết các trạng thái hoạt động của hệ thống. Đèn báo trong công nghiệp thường sử dụng hai loại: đèn báo DC và đèn báo AC. Tùy từng hệ thống cụ thể, kỹ sư thiết kế sẽ sử dụng loại phù hợp.



Hình 6.12: Đèn báo

b. Rơ le:

Để thực hiện đóng cắt các thiết bị điện trong hệ thống như van điện – khí nén, công tắc tơ ... và đảm bảo cách li với các thiết bị điều khiển (PLC), rơ le điện từ luôn được sử dụng. Các đầu ra của PLC thường có khả năng chịu dòng điện nhỏ, khi có sự cố trong hệ thống sẽ dễ dẫn đến hỏng, việc thay thế sửa chữa khó khăn, vì vậy không được dùng để đóng điện trực tiếp với dòng điện lớn mà phải dùng thông qua rơ le điện từ làm trung gian. Rơ le điện từ dùng trong công nghiệp thường có 3 loại khi phân loại theo điện áp cuộn hút: 12VDC, 24VDC, 220VAC. Số cặp tiếp điểm tùy vào từng loại rơ le, thông thường có 2 cặp tiếp điểm, 4 cặp tiếp điểm. Các rơ le trung gian thường được cắm trên đế cắm để dễ dàng lắp đặt, thay thế, sửa chữa.

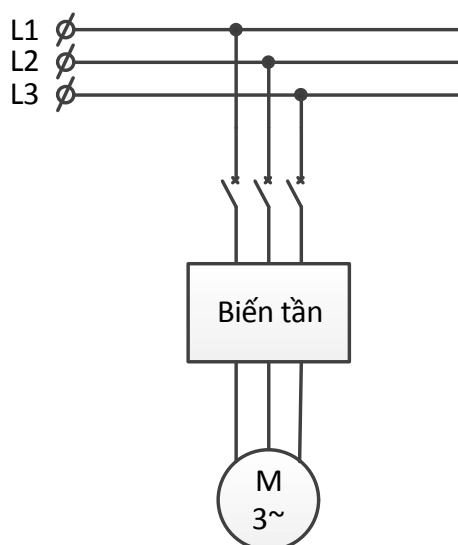


Hình 6.13: Rơ le trung gian

6.1.2 Các thiết bị khác

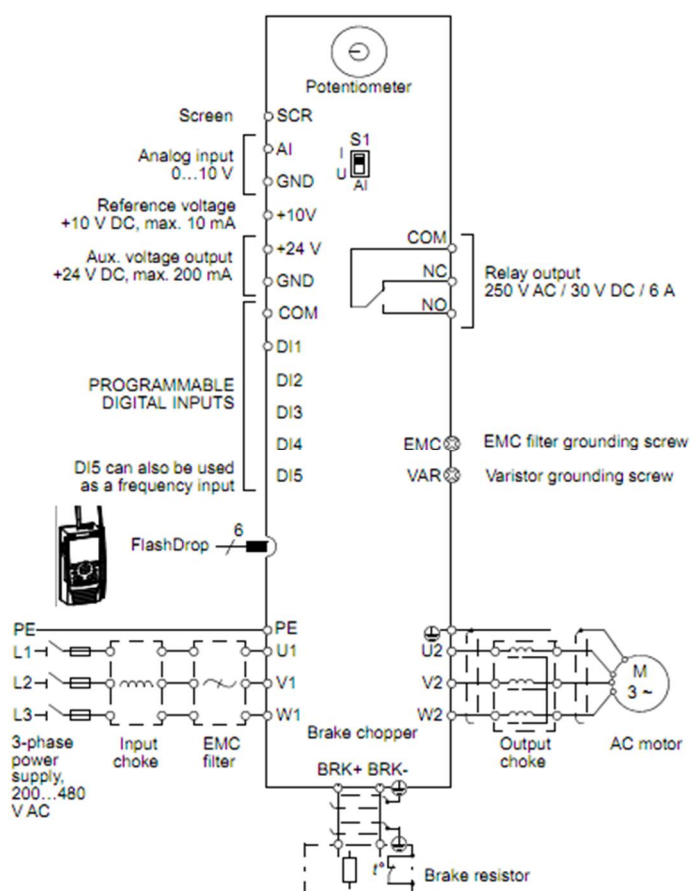
a. Biến tần

Trong các hệ thống sản xuất, động cơ điện được sử dụng rất rộng rãi. Công tắc tơ là thiết bị thường được sử dụng để đóng cắt nguồn cấp cho động cơ điện. Tuy nhiên, với các vị trí sử dụng động cơ có công suất lớn, không thể đóng điện trực tiếp. Ngoài ra, phụ thuộc vào yêu cầu công nghệ, các động cơ còn có thể được ổn định tốc độ, điều chỉnh tốc độ, tăng tốc, giảm tốc theo thời gian v.v... Khi đó biến tần sẽ được lắp giữa nguồn và động cơ.



Hình 6.14: Vị trí biến tần trong hệ thống

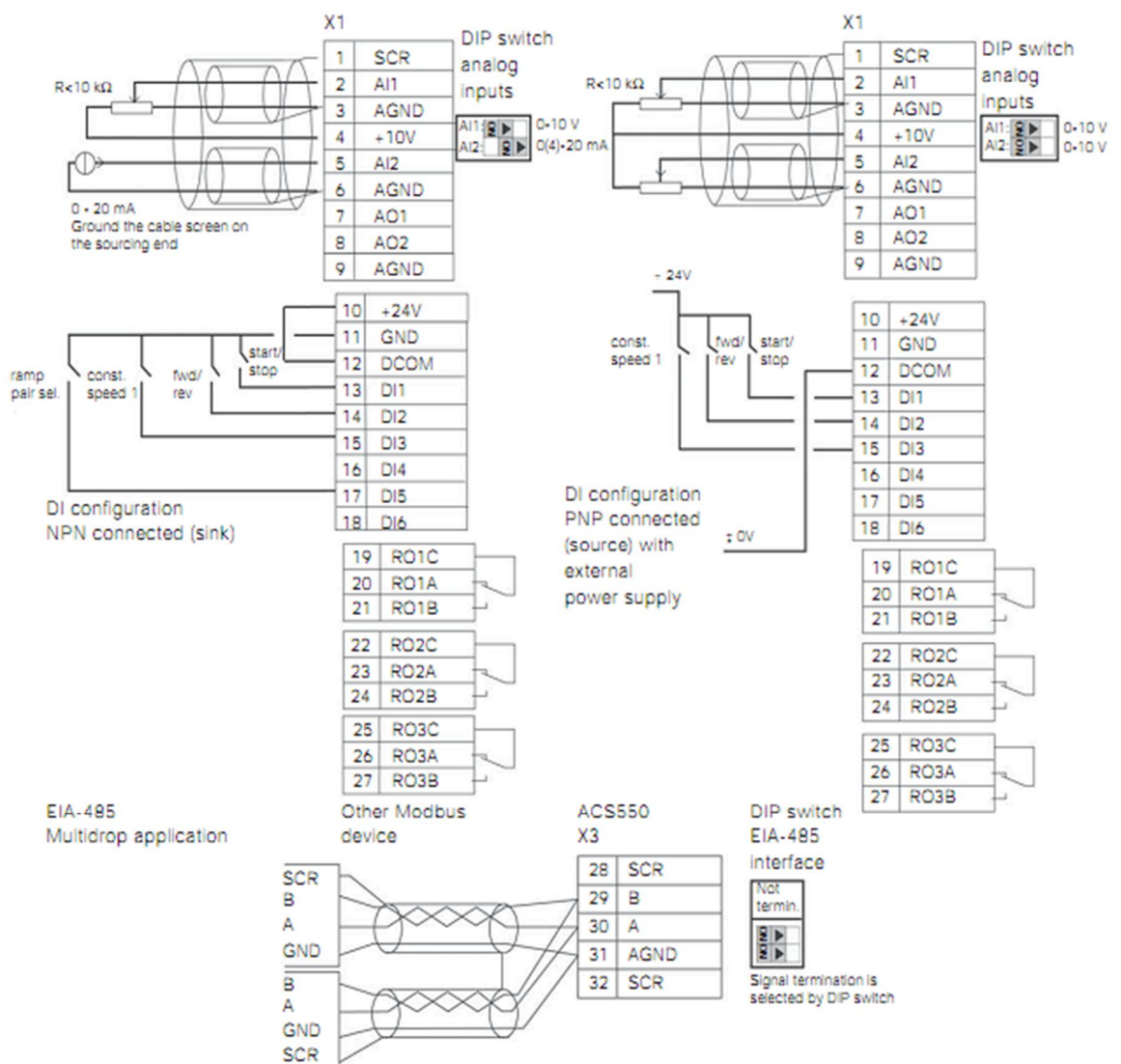
Trên thực tế có rất nhiều loại biến tần khác nhau được sản xuất bởi nhiều hãng khác nhau. Tuy nhiên, cũng giống PLC, biến tần cũng tuân thủ các tiêu chuẩn công nghiệp về vào/ra. Vì vậy, hầu hết các biến tần có cấu trúc tương tự nhau.



Hình 6.15: Sơ đồ đấu nối cơ bản cho biến tần ABB-ACS150

Việc điều khiển biến tần có thể trực tiếp bằng tay bởi người vận hành hoặc thông qua hệ thống điều khiển, PLC. Hệ thống điều khiển, PLC ghép nối với biến tần qua các cổng truyền thông, đầu vào/ra số, vào/ra tương tự. Nói chung, các biến tần có các đầu vào/ra cơ bản sau: đầu vào nguồn cấp, đầu ra động cơ, các đầu vào/ra số, vào/ra tương tự, cổng truyền thông. Ví dụ về các đầu vào/ra của biến tần được thể hiện trong hình ... với biến tần ABB-ACS150. Đầu vào nguồn: U1, V1, W1, PE; đầu ra động cơ: U2, V2, W2; đầu vào tương tự: AI; đầu vào số: DI1, DI2, DI3, DI4, DI5, COM; đầu ra số: COM, NC, NO.

Sơ đồ đầu nối vào/ra đối với các tín hiệu được thể hiện trong hình ... Đây là ví dụ với biến tần ABB-ACS550. Các loại biến tần khác cũng có cách đầu nối tương tự.



Hình 6.16: Sơ đồ đầu nối tín hiệu với biến tần ABB-ACS550

b. HMI:

HMI (Human Machine Interface) là thiết bị giao diện giữa người và máy sản xuất (hệ thống sản xuất). Đây là các màn hình giao diện và màn hình cảm ứng được ghép nối với các thiết bị điều khiển để thực hiện chức năng hiển thị trạng thái hoạt

động của hệ thống: cảnh báo, số liệu thực tế, số liệu đặt ... và nhận các thao tác từ người điều khiển truyền xuống thiết bị điều khiển. Các HMI dùng trong công nghiệp được chế tạo theo tiêu chuẩn công nghiệp nên có các đặc điểm chung: các tính năng về giao diện như độ phân giải, màu sắc thường thấp hoặc vừa phải; độ bền cao, hoạt động tốt trong các môi trường khắc nghiệt; ngoài các cổng giao tiếp thông thường còn có cổng giao tiếp theo chuẩn công nghiệp như RS232, RS485; giá thành cao; được chế tạo với nhiều kích thước khác nhau để phục vụ nhiều mục đích sử dụng.



Hình 6.17: HMI của Siemens

6.2 Ghép nối với PLC

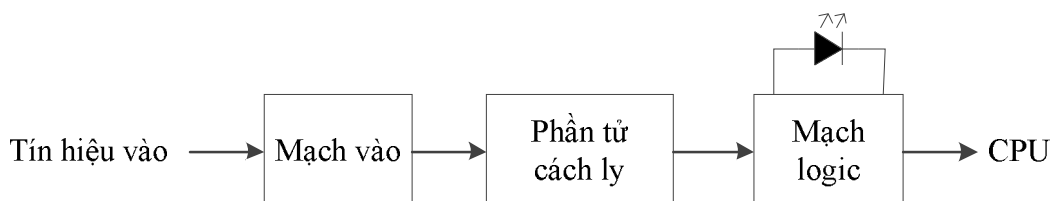
6.2.1 Vào/ra số

Trong các đầu vào/ra của PLC, các đầu vào/ra số (Digital I/O) được sử dụng phổ biến nhất. Chúng thực hiện chức năng ghép nối với các tín hiệu vào/ra dạng ON/OFF. Các đầu vào/ra số có nhiệm vụ chuyển đổi tín hiệu từ thiết bị ngoại vi thành dạng tín hiệu logic đưa vào CPU hoặc chuyển tín hiệu từ CPU thành dạng thích hợp với các thiết bị ngoại vi. Đồng thời các đầu vào/ra số còn có chức năng cách li giữa PLC và các thiết bị ngoại vi để đảm bảo an toàn cho PLC khi hoạt động.

a. Đầu vào số

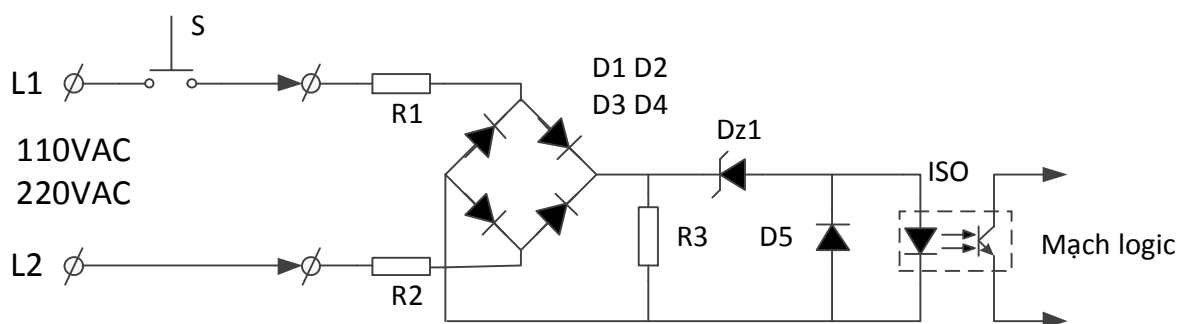
Các đầu vào số được ghép nối với các thiết bị vào như: chuyển mạch, nút nhấn, công tắc hành trình, các cảm biến đầu ra ON/OFF, các đầu ra ON/OFF của các bộ điều khiển nhiệt độ, khối lượng Mỗi thiết bị vào sẽ được nối với một đầu vào có vị trí và tên gọi xác định (phụ thuộc vào từng loại PLC). Mỗi đầu vào sẽ tương ứng với 1 bit nhớ. Bit nhớ sẽ có giá trị 0 hoặc 1 phụ thuộc vào trạng thái của thiết bị vào hay nói cách khác là tín hiệu đưa vào đầu vào tương ứng. Trong thực tế sản xuất, có nhiều loại tín hiệu vào khác nhau: 24VDC, 110VAC, 220VAC ... Để phù hợp với các tín hiệu vào khác nhau, các nhà sản xuất đã chế tạo ra các đầu vào có nguyên lý khác nhau.

Tuy nhiên, sơ đồ khối cơ bản của đầu vào số của PLC được mô tả trong hình 6.18 gồm có: mạch vào, phần tử cách ly (thường là cách ly quang), mạch logic và đèn báo.



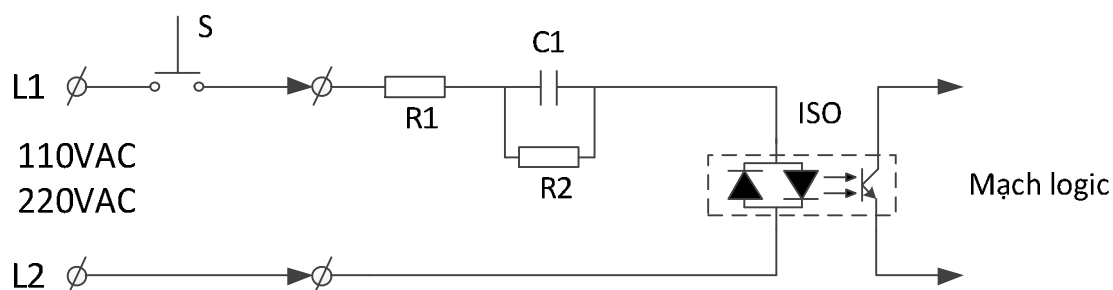
Hình 6.18: Sơ đồ khối chức năng của đầu vào số

Tín hiệu vào sẽ được đưa vào mạch vào có nguyên lý phù hợp, tín hiệu đó sẽ được đưa đến phần tử cách ly rồi chuyển đến mạch logic. Mạch logic có nhiệm vụ bật LED báo và chuyển đổi thành tín hiệu phù hợp với CPU để xử lý. Như vậy với mỗi loại tín hiệu vào khác nhau, sơ đồ nguyên lý của mạch vào sẽ khác nhau. Sơ đồ nguyên lý của mạch vào với tín hiệu vào 110VAC hoặc 220VAC được mô tả trong hình 6.19 với S là nút nhấn; R1, R2, R3, D1, D2, D3, D4, D5, Dz1 tạo ra mạch vào; ISO là phần tử cách ly quang (Opto Coupler)



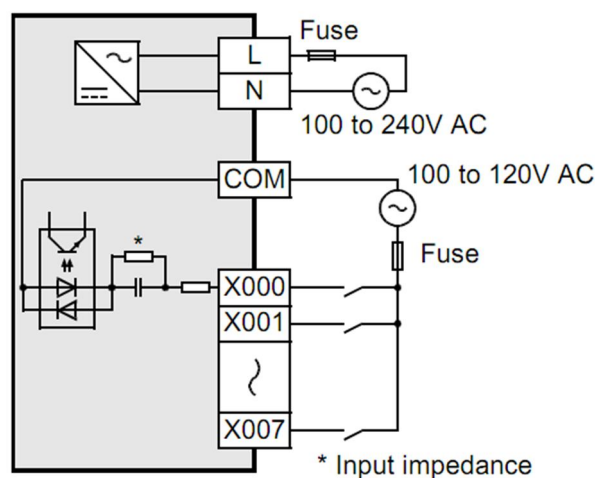
Hình 6.19: Sơ đồ nguyên lý mạch vào 110VAC – 220VAC

Tuy nhiên, sơ đồ này cũng không đúng hoàn toàn với tất cả các PLC, có sơ đồ khác cũng phù hợp với tín hiệu vào xoay chiều và được thể hiện trong hình 6.20.



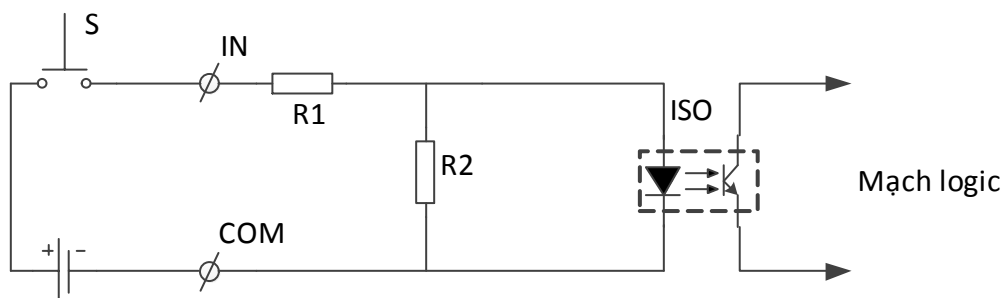
Hình 6.20: Sơ đồ nguyên lý mạch vào 110VAC – 220VAC

PLC Mitsubishi FX3U sử dụng mạch vào xoay chiều theo nguyên lý 2 và có sơ đồ ghép nối như hình 6.21

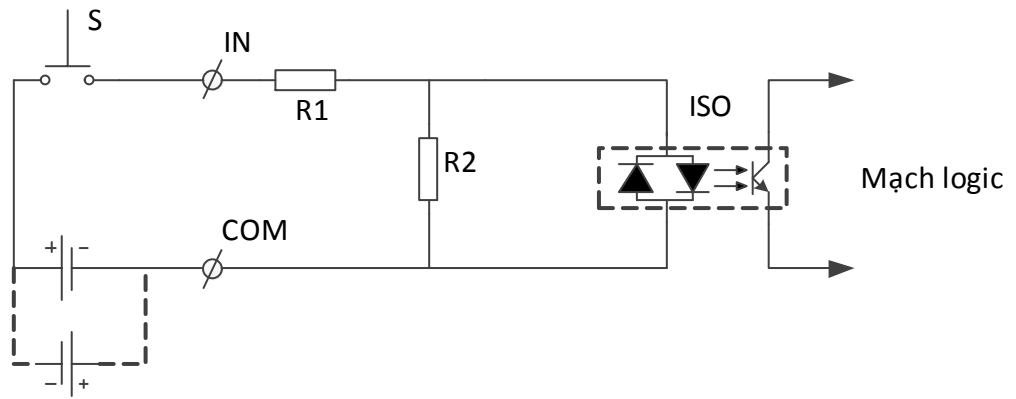


Hình 6.21: Sơ đồ ghép nối đầu vào tín hiệu 110VAC-120VAC của PLC FX3U

Trong các ứng dụng thực tế, tín hiệu vào 24VDC được sử dụng phổ biến hơn vì tính an toàn cho sử dụng, vận hành và điều kiện hoạt động của các thiết bị vào (đặc biệt là các loại cảm biến). Đồng thời sơ đồ nguyên lý của mạch vào 24VDC đơn giản hơn nhiều so với mạch vào 110VAC hay 220VAC. Tuy nhiên, khi làm việc với nguồn DC, chiều của nguồn điện rất quan trọng. Do đó, có 2 sơ đồ nguyên lý của mạch vào với tín hiệu vào DC được thể hiện trong hình 6.22



a, Sơ đồ sử dụng nguồn DC cố định



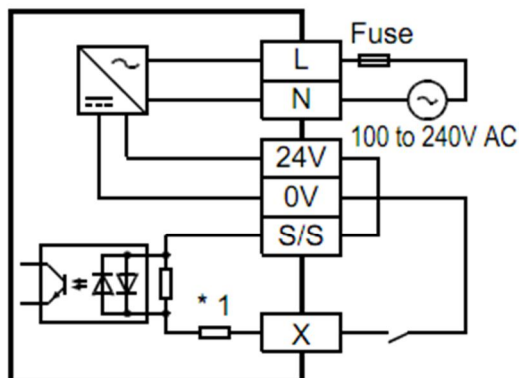
b, Sơ đồ sử dụng nguồn DC chiều tùy ý

Hình 6.22: Sơ đồ nguyên lý mạch vào 24VDC

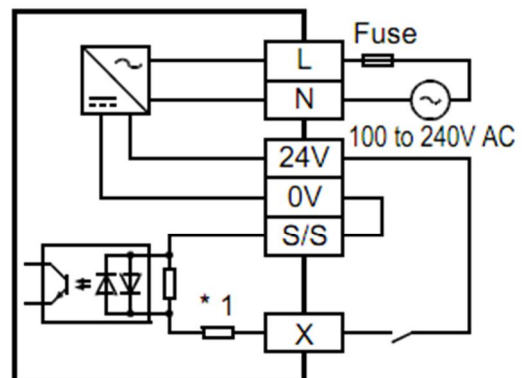
Trong sơ đồ a, chỉ sử dụng phần tử cách ly với tín hiệu vào đi theo một chiều xác định nên nguồn DC chỉ lắp được theo một chiều. Trong sơ đồ b, do sử dụng phần tử cách ly cho dòng điện chạy theo hai chiều nên có thể tùy ý đấu nguồn DC. Sơ đồ b cũng là sơ đồ được sử dụng trong hầu hết các PLC hiện nay. Ví dụ ghép nối thiết bị vào 24VDC với PLC FX3U của Mitsubishi được thể hiện trong hình 6.23

- AC power supply type

Sink input wiring

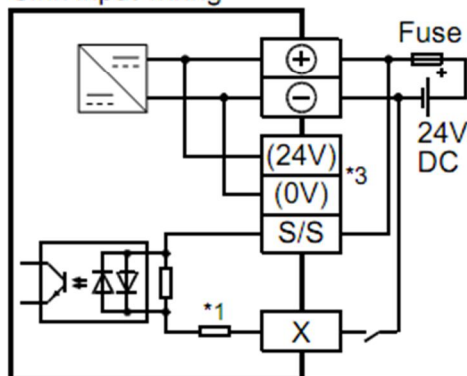


Source input wiring

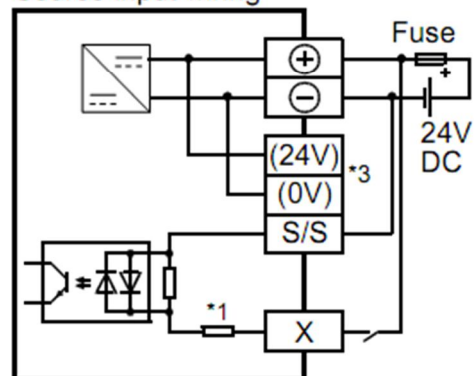


- DC power supply type

Sink input wiring



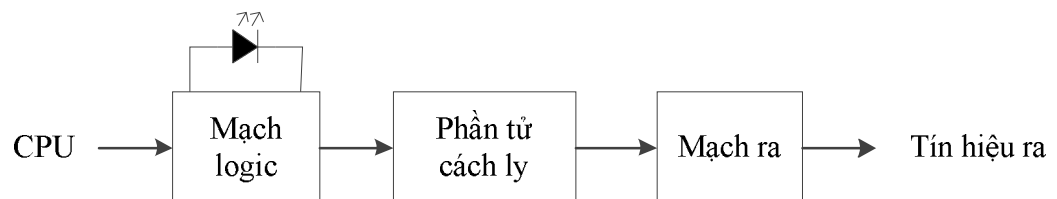
Source input wiring



Hình 6.23: Sơ đồ nguyên lý ghép nối thiết bị vào 24VDC với đầu vào PLC FX3U

b. Đầu ra số:

Các đầu ra số được ghép nối với các thiết bị ra nhận tín hiệu ON/OFF (đóng/cắt, 24VDC và 0VDC, ...) như rơ le điện từ, đèn báo, công tắc tơ, van, ... Mỗi thiết bị ra sẽ kết nối với một đầu ra cố định của PLC và được định địa chỉ bit. Khi thực hiện thao tác điều khiển ghi/xóa bit nhớ tương ứng thì trạng thái đầu ra thay đổi theo. Cũng giống đầu vào số, các đầu ra có thể được kết nối với thiết bị ra sử dụng điện xoay chiều, một chiều nên sẽ có các sơ đồ nguyên lý khác nhau. Tuy nhiên sơ đồ khối vẫn không thay đổi và được thể hiện trong hình 6.24



Hình 6.24: Sơ đồ khối mạch ra

6.2.2 Vào/ra tương tự

Trong khả năng của PLC, ngoài làm việc với các hệ thống logic tuần tự, PLC còn được sử dụng rất phổ biến trong các hệ điều khiển quá trình với số lượng lớn các tín hiệu tương tự từ các cảm biến, biến tần, đồng hồ đo ... Các đầu vào/ra tương tự có thể được chế tạo trên Main module hoặc được chế tạo thành các module mở rộng.

a. Vào tương tự:

PLC muốn xử lý cả tín hiệu tương tự cần phải được kết nối với các đầu vào tương tự. Chức năng của các đầu vào tương tự là biến đổi tín hiệu từ dạng tương tự thành số để lưu vào vùng nhớ nhất định để PLC xử lý. Trong công nghiệp, các tín hiệu tương tự được chuẩn hóa dưới hai dạng là dòng điện và điện áp. Chuẩn tín hiệu dòng điện là các dải: 0mA đến 20mA, 4mA đến 20mA. Chuẩn tín hiệu điện áp có thể là điện áp lưỡng cực hoặc đơn cực và có các dải giá trị: (-5V, +5V), (-10V, +10V), (0V, +5V), (0V, +10V). Ngoài các đầu vào tương tự được chế tạo chuẩn, còn có một số đầu vào tương tự được chế tạo chuyên dụng cho một loại cảm biến nhất định. Ví dụ : đầu vào tương tự cho điện trở nhiệt Pt100, cặp nhiệt ngẫu, ...

Để lựa chọn các đầu vào tương tự của PLC phù hợp với hệ thống và yêu cầu điều khiển, thiết bị thì cần chú ý tới các vấn đề sau :

- Số kênh trên 1 module.
- Dạng và dải tín hiệu vào: dòng điện hay điện áp, dải hoạt động để chọn dải chuẩn.
- Trở kháng vào.
- Độ phân giải: 8bit, 10bit, 12bit tùy vào từng module. Thông số này quyết định khả năng xử lý dữ liệu của PLC đối với tín hiệu tương tự.
- Tốc độ biến đổi.

b. Đầu ra tương tự

Để điều khiển các thiết bị trường như biến tần, PLC có thể được trang bị các đầu ra tương tự. Thông thường, đầu ra tương tự không được chế tạo tích hợp trên Main module mà chỉ được chế tạo trong các module mở rộng. Các module ra tương tự có chức năng biến đổi từ giá trị số trong vùng nhớ nhất định thành dạng tín hiệu ra tương tự. Cũng giống như các đầu vào tương tự, các đầu ra tương tự cũng được chuẩn hóa tín hiệu theo tiêu chuẩn công nghiệp. Tín hiệu ra cũng có hai dạng là dòng điện và điện áp. Các dải tín hiệu và các điều cần chú ý cũng giống đầu vào tương tự.

Trên thực tế, nhà sản xuất cũng chế tạo các module vào/ra tương tự hỗn hợp. Tức là trên cũng một module vừa có các đầu vào và đầu ra tương tự.

6.2.3 Vào/ra tốc độ cao

Để sử dụng đếm các tín hiệu tốc độ cao như các xung từ encoder, xung tín hiệu khác, PLC phải được ghép với các đầu vào tốc độ cao và có bộ đếm tốc độ cao (HSC – High Speed Counter) bên trong. Vì chu kỳ quét của PLC thường vào cỡ ms nên không thể đáp ứng được sự thay đổi của tín hiệu vào. Bộ đếm tốc độ cao HSC được sử dụng hoạt động độc lập với CPU, tự động cập nhật giá trị xung tín hiệu đếm được vào bộ nhớ và CPU sẽ lấy ra sử dụng. Các tần số thường có với bộ đếm tốc độ cao là 20kHz, 50kHz, 100kHz, 200kHz và có thể lên tới 500kHz.

Các đầu vào ra tốc độ cao có thể được tích hợp trên một số dòng Main Module nhất định. Tuy nhiên, các đầu vào HSC tích hợp thường có tần số thấp, dạng tín hiệu vào bị hạn chế (thường chỉ dùng cho loại tín hiệu 3 dây hoặc 4 dây từ encoder). Khi muốn sử dụng với tần số cao và đầy đủ khả năng về phần cứng mà phần mềm, PLC sẽ được ghép nối với module mở rộng HSC.

Ngoài các đầu vào/ra đã được đề cập đến, có rất nhiều các module vào/ra đặc biệt khác được chế tạo chuyên dụng cho một mục đích : module điều khiển động cơ

bước, module điều khiển động cơ AC servo, module truyền thông mở rộng, module GSM-GPRS (truyền dữ liệu qua mạng điện thoại di động) module đọc mã code ...

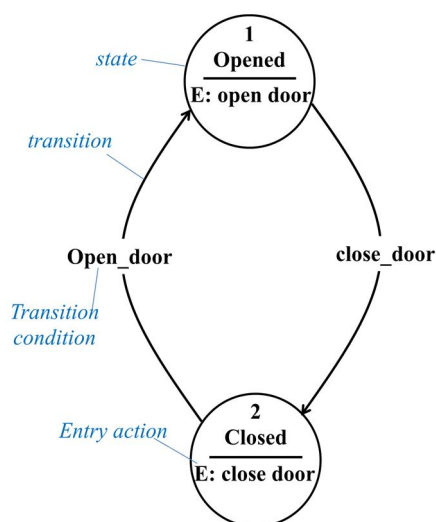
CHƯƠNG 7

MÔ PHỎNG HỆ THỐNG CÁC SỰ KIỆN RỜI RẠC

7.1 Aptomat hữu hạn - Finite State Machine (FSM)

7.1.1. Giới thiệu chung

Automat hữu hạn là một khái niệm toán học trừu tượng được dùng để thiết kế các mạch logic hoặc các chương trình máy tính. Nó là một mô hình ứng xử (behavior model) bao gồm một số lượng hữu hạn các trạng thái (state), các chuyển tiếp (transition) giữa các trạng thái đó, và các hành động, tương tự như lưu đồ mà trong đó người ta có thể giám sát cách logic được thực hiện khi một điều kiện nào đó thỏa mãn. Automat hữu hạn có bộ nhớ trong hữu hạn, đặc tính đầu vào là đọc các ký hiệu theo tuần tự, mỗi lần một ký hiệu và không quay trở lại được; đặc tính đầu ra ở dạng giao diện người dùng, mỗi khi mô hình được thi hành. Sự vận hành của automat hữu hạn bắt đầu từ một trạng thái (được gọi là trạng thái bắt đầu – start state), đi qua các chuyển tiếp phụ thuộc vào đầu vào tới các trạng thái khác nhau và có thể kết thúc bất cứ lúc nào có thể. Tuy nhiên chỉ một tập hợp nào đó của các trạng thái đánh dấu sự thành công của quá trình vận hành đó (được gọi là các trạng thái chấp nhận- accept states)



Hình 7.1: Ví dụ về một automat hữu hạn

Automat hữu hạn có thể giải quyết được rất nhiều vấn đề trong tự động hóa, thiết kế điện tử, thiết kế các giao thức truyền thông, phân tích cú pháp và các ứng dụng kỹ thuật khác. Trong các nghiên cứu về sinh học và trí tuệ nhân tạo, automat hữu hạn hoặc là các phân cấp của automat hữu hạn đôi khi được sử dụng để mô tả các hệ thống thần kinh và trong ngôn ngữ để mô tả ngữ pháp của các ngôn ngữ tự nhiên.

7.1.2. Các khái niệm

Một trạng thái (state) hiện tại được xác định bởi các trạng thái trước đó của hệ thống. Có thể hiểu là nó ghi nhớ các thông tin về quá khứ, nghĩa là, nó phản ánh sự thay đổi đầu vào của hệ thống từ khi bắt đầu đến thời điểm hiện tại.

Một chuyển tiếp (transition) chỉ ra sự thay đổi trạng thái và được mô tả bởi một điều kiện cần được thi hành để kích hoạt trạng thái.

Một hành động (action) là một mô tả của một hoạt động được thực hiện tại một thời điểm cho trước. Có một vài kiểu hành động sau:

- Hành động đi vào (entry action): được thực hiện khi chuyển vào một trạng thái
- Hành động thoát ra (exit action): được thực hiện khi thoát ra khỏi trạng thái
- Hành động đầu vào (input action): được thực hiện phụ thuộc vào trạng thái hiện tại và các điều kiện đầu vào.
- Hành động chuyển tiếp (transition action): được thực hiện khi đang thực hiện chuyển tiếp đó

Một automata hữu hạn có thể được biểu diễn sử dụng giản đồ trạng thái (state diagram) như ở trong hình 1. Bên cạnh đó, một vài bảng chuyển trạng thái được sử dụng. Một cách biểu diễn phổ biến nhất là như ở bảng dưới đây: sự kết hợp của trạng thái hiện tại (trạng thái B) và đầu vào (Y) đưa đến trạng thái tiếp theo (trạng thái C). Toàn bộ các thông tin hành động có thể được thêm vào sử dụng các ghi chú.

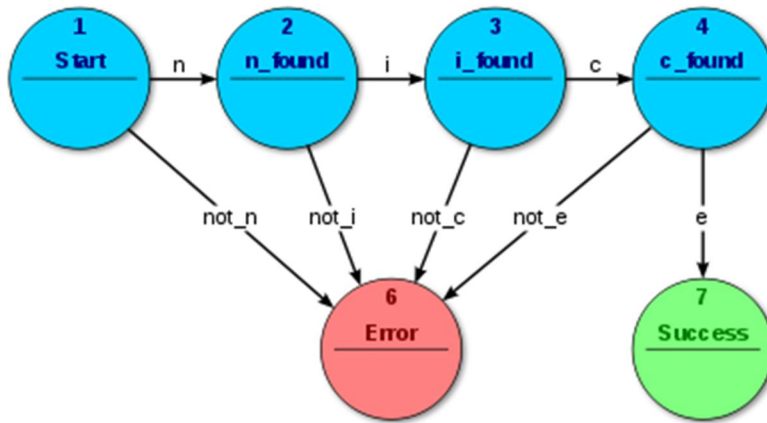
Bảng 7.1: Bảng chuyển trạng thái

Bảng chuyển trạng thái			
Trạng thái hiện tại → Đầu vào ↓	Trạng thái A	Trạng thái B	Trạng thái C
Đầu vào X
Đầu vào Y	...	Trạng thái C	...
Đầu vào Z

7.1.3. Phân loại

Có hai loại automata hữu hạn: Loại acceptors/Recognizer và Transducer

a. Acceptors/Recognizer



Hình 7.2: Automat hữu hạn loại Acceptor: Phân tích cú pháp từ "nice"

Acceptors/recognizers (hay gọi là bộ dò tuần tự - **sequence detectors**) tạo ra một đầu ra nhị phân, thể hiện câu trả lời có (yes) hoặc không (no) cho câu hỏi liệu một đầu vào được máy chấp nhận hay không được chấp nhận. Tất cả các trạng thái của automat hữu hạn này được cho là chấp nhận hay không chấp nhận. Tại thời điểm khi tất cả các đầu vào được xử lý, nếu trạng thái hiện tại là một trạng thái chấp nhận, thì đầu vào được chấp nhận; nếu không, nó bị loại bỏ. Theo luật thì đầu vào là các biểu tượng (các ký tự); các hành động không được dùng. Ví dụ trong hình 2 ở trên chỉ ra một automat hữu hạn chấp nhận từ "nice". Từ trạng thái số 1 là "Start", nếu đầu vào là chữ "n" thì sẽ chuyển sang trạng thái số 2 là "n_found" (đã tìm thấy chữ n). Nếu không phải là chữ "n" thì sẽ chuyển sang trạng thái "Error" (báo lỗi là không phải từ nice). Tương tự automat hữu hạn sẽ xét các đầu vào tiếp theo xem có đúng lần lượt là các ký tự "i", "n", và "e" hay không. Nếu không đúng thì máy sẽ chuyển đến trạng thái "Error", nếu tất cả các đầu vào là đúng và lần lượt theo thứ tự, thì máy sẽ chuyển tới trạng thái số 7 là "Success", nghĩa là từ được nhập vào đúng là từ "nice". Trong automat hữu hạn này, chỉ có một trạng thái chấp nhận là trạng thái số 7

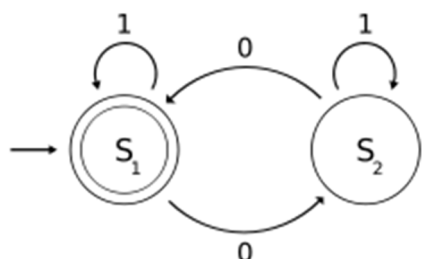
Automat hữu hạn cũng có thể được mô tả như là một định nghĩa một ngôn ngữ, chứa tất cả các từ được chấp nhận bởi automat hữu hạn mà không bị loại bỏ. Khi đó ta nói rằng ngôn ngữ đó được chấp nhận bởi automat hữu hạn. Theo định nghĩa, ngôn ngữ đó được gọi là "regular languages". Một ngôn ngữ được gọi là regular language nếu như có một automat hữu hạn nào đó chấp nhận nó.

Ở đây có thêm một số khái niệm mới như sau:

- Trạng thái bắt đầu (start state): Trạng thái bắt đầu thường được vẽ với một mũi tên chỉ vào trạng thái đó (có thể từ bất cứ phía nào)

- **Trạng thái chấp nhận (accept state):** Trạng thái chấp nhận là trạng thái mà tại đó automata hữu hạn đã thực hiện thành công các thủ tục của nó. Trạng thái chấp nhận được biểu thị bằng hai vòng tròn lồng vào nhau

Hình 7.3 là một ví dụ về trạng thái chấp nhận trong một automata hữu hạn xác định nếu đầu vào nhị phân chứa số chẵn các chữ số 0



Hình 7.3: Một automata hữu hạn xác định một số nhị phân có số chữ số 0 là chẵn với S_1 là trạng thái chấp nhận.

S_1 biểu diễn trạng thái tại đó số chẵn chữ số 0 đã được đưa vào và do đó được định nghĩa là một trạng thái chấp nhận. S_1 đồng thời cũng là trạng thái bắt đầu, do ban đầu không có chữ số nào được nhập vào. Khi đang ở trạng thái S_1 , nếu có ký tự 0 vào, thì số chữ số 0 tăng lên 1, nghĩa là số chữ số 0 là một số lẻ, và máy chuyển sang trạng thái S_2 là trạng thái có số chữ số 0 là lẻ. Nếu đang ở trạng thái S_2 mà có ký tự 0 vào, thì số chữ số 0 tăng lên 1 và số chữ số 0 bây giờ là một số chẵn. Do đó máy chuyển từ trạng thái S_2 sang trạng thái S_1 . Nếu có ký tự 1 vào thì số chữ số 0 là không thay đổi, do đó máy ở nguyên trạng thái hiện thời.

Automata hữu hạn này sẽ đưa ra một trạng thái kết thúc đúng nếu số nhị phân chứa số chẵn các chữ số 0 bao gồm cả dãy không có chữ số 0 nào

b. Transducers

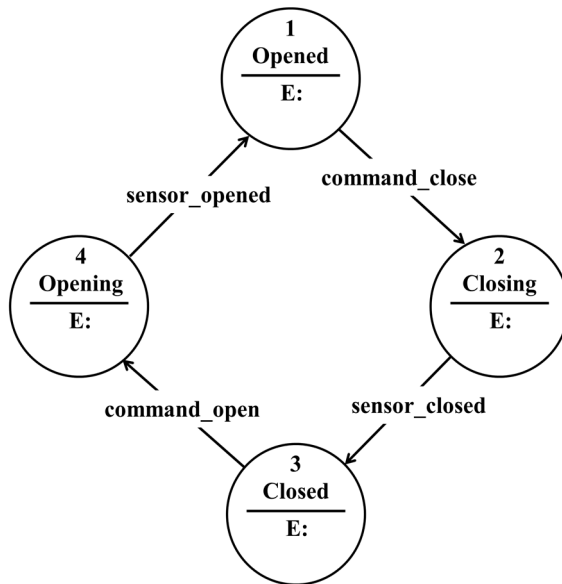
Transducers phát ra đầu ra dựa vào đầu vào và trạng thái sử dụng các hành động

Transducer được dùng trong các ứng dụng điều khiển và trong lĩnh vực ngôn ngữ tính toán. Có hai loại transducer nổi bật:

- **Moore machine:**

Moore machine chỉ sử dụng các hành động đi vào (entry action), nghĩa là đầu ra chỉ phụ thuộc vào trạng thái. Ưu điểm của mô hình Moore là tính đơn giản của các ứng xử. Xét ví dụ về cửa thang máy. Automata hữu hạn nhận ra hai lệnh “command_open” (mở cửa) và “command_close” (đóng cửa) để thay đổi trạng thái. Hành động đi vào (E:) trong trạng thái “Opening” (Đang mở) khởi động động cơ để mở cửa ra, hành động đi vào trong trạng thái “Closing” (Đang đóng) khởi động động

cơ theo hướng ngược lại để đóng cửa lại. Các trạng thái “Opened” (đã mở) và “Closed” (đã đóng) dừng động cơ khi cửa đã mở hoàn toàn (sensor_opened) hoặc đóng hoàn toàn (sensor_closed). Chúng gửi tín hiệu ra ngoài (hoặc tới các automat hữu hạn khác) tính hướng “cửa đã được mở” hoặc “cửa đã được đóng”.

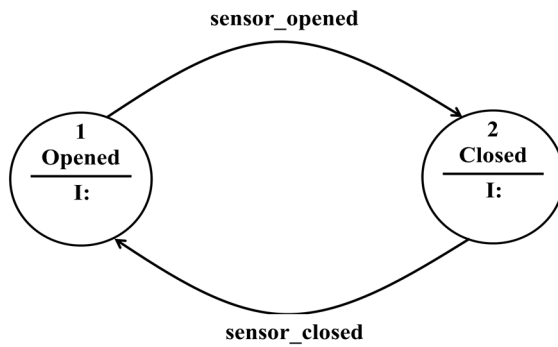


Hình 7.4: Moore machine cho mô hình cửa thang máy

- Mealy machine

Mealy machine chỉ dùng các hành động đầu vào, nghĩa là đầu ra phụ thuộc vào đầu vào và trạng thái. Việc sử dụng Mealy machine thường dẫn đến việc giảm bớt số lượng trạng thái. Ví dụ như trong hình 4 là một Mealy machine thực hiện một cách ứng xử tương tự như ví dụ về Moore machine. Ở đây có hai hành động đầu vào (I:): “khởi động động cơ để đóng cửa nếu có lệnh đóng (command_closed) đến” ở trạng thái đã mở (Opened) và “khởi động động cơ để mở cửa nếu có lệnh mở (command_opened) đến” ở trạng thái đã đóng (Closed). Khi đang ở trạng thái đã đóng, nếu có lệnh mở cửa (commanded_opened), thì hành động đầu vào (I:) sẽ được thực hiện và cửa mở ra. Khi cửa mở hoàn toàn (sensor_opened), hệ sẽ chuyển sang trạng thái đã mở (Opened). Khi đang ở trạng thái đã mở, nếu có lệnh đóng cửa (commanded_closed), thì hành động đầu vào (I:) sẽ được thực hiện và cửa đóng lại. Khi cửa đóng hoàn toàn (sensor_closed), hệ sẽ chuyển sang trạng thái đã đóng (Closed). Các trạng thái trung gian “opening” (đang mở) và “closing” (đang đóng) không xuất hiện trong mô hình này.

Trong thực tế thì mô hình kết hợp giữa Moore và Mealy machine thường hay được sử dụng.



Hình 7.5: Mealy machine cho mô hình cửa thang máy

7.1.4. Mô hình toán học

a. Acceptor

Một automata hữu hạn loại Acceptor gồm 5 thành phần $(\Sigma, S, s_0, \delta, F)$, với

- Σ : tập các ký tự đầu vào (tập hữu hạn khác rỗng các biểu tượng)
- S : tập hữu hạn khác rỗng các trạng thái.
- s_0 là trạng thái ban đầu, một thành phần của S .
- δ là hàm chuyển trạng thái: $\delta : S \times \Sigma \rightarrow S$
- F : là tập các trạng thái cuối cùng, một tập con của S .

Hàm δ không nhất thiết phải định nghĩa cho tất cả các tổ hợp của (trạng thái, đầu vào). Nếu như một automata hữu hạn đang ở trạng thái q , ký tự tiếp theo là x và $\delta(q, x)$ không được định nghĩa, thì M có thể phát ra một lỗi (nghĩa là loại bỏ đầu vào)

b. Transducer

Một automata hữu hạn loại transducer gồm 6 thành phần $(\Sigma, \Gamma, S, s_0, \delta, \omega)$ trong đó

- Σ : tập các ký tự đầu vào (tập hữu hạn khác rỗng các biểu tượng).
- Γ : tập các ký tự đầu ra (tập hữu hạn khác rỗng các biểu tượng).
- S : tập hữu hạn khác rỗng các trạng thái..
- s_0 : là trạng thái ban đầu, một thành phần của S .
- δ : là hàm chuyển trạng thái: $\delta : S \times \Sigma \rightarrow S$.
- ω : là hàm đầu ra.

Nếu hàm đầu ra là hàm của trạng thái và ký tự đầu vào ($\omega : S \times \Sigma \rightarrow \Gamma$), định nghĩa đó tương ứng với mô hình **Mealy**. Nếu hàm đầu ra chỉ phụ thuộc vào trạng thái ($\omega : S \rightarrow \Gamma$) định nghĩa đó tương đương với mô hình **Moore**. Automata hữu hạn không có hàm đầu ra được gọi là hệ thống chuyển.

7.2 Petri Net

7.2.1. Các định nghĩa cơ bản

a. Cấu trúc Petri net

Petri net bao gồm 4 phần: tập các vị trí (place) P , tập các chuyển tiếp (transition) T , một hàm đầu vào I và một hàm đầu ra O . Các hàm đầu vào và đầu ra định nghĩa mối quan hệ giữa các vị trí và các chuyển tiếp. Hàm đầu vào I là một ánh xạ từ một chuyển tiếp t_j tới một bộ các vị trí $I(t_j)$, được gọi là các vị trí đầu vào của chuyển tiếp đó. Hàm đầu ra O là một ánh xạ từ một chuyển tiếp t_j tới một bộ các vị trí $O(t_j)$, được gọi là các vị trí đầu ra của chuyển tiếp đó.

Cấu trúc của một Petri net được định nghĩa bằng các vị trí, các chuyển tiếp và các hàm đầu vào, đầu ra.

Định nghĩa 1 Một cấu trúc Petri net C gồm 4 thành phần $C = (P, T, I, O)$

$P = \{p_1, p_2, \dots, p_n\}$ là tập hữu hạn các vị trí (place), $n \geq 0$; $T = \{t_1, t_2, \dots, t_m\}$ là tập hữu hạn các chuyển tiếp (transition), $m \geq 0$, $P \cap T = \emptyset$.

$I: T \rightarrow P^\infty$ là hàm đầu vào, một ánh xạ từ các chuyển tiếp tới túi (bag) các vị trí.

$O: T \rightarrow P^\infty$ là hàm đầu ra, một ánh xạ từ các chuyển tiếp tới túi (bag) các vị trí.

Một vị trí p_i là một vị trí đầu vào của một chuyển tiếp t_j nếu $p_i \in I(t_j)$; p_i là vị trí đầu ra của chuyển tiếp nếu $p_i \in O(t_j)$. Các đầu vào và đầu ra của một chuyển tiếp là túi (bag) của các vị trí. Túi là một khái niệm tổng quát của tập hợp, nó cho phép sự xuất hiện nhiều lần của một thành phần trong một túi. Việc sử dụng khái niệm túi ở đây để cho phép một vị trí có thể là đầu vào nhiều lần hoặc đầu ra nhiều lần của một chuyển tiếp. Số lần xuất hiện của một vị trí p_i trong túi đầu vào của một chuyển tiếp t_j được ký hiệu là $\#(p_i, I(t_j))$. Tương tự, số lần xuất hiện của một vị trí p_i trong túi đầu ra của một chuyển tiếp t_j được ký hiệu là $\#(p_i, O(t_j))$. Nếu các hàm đầu vào và đầu ra là các tập (chứ không phải là túi) thì số lần xuất hiện sẽ chỉ là 0 hoặc 1.

Ví dụ 7.1:

Cấu trúc của một Petri net $C = (P, T, I, O)$ với

$$P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$$

$$T = \{t_1, t_2, t_3, t_4\}$$

$$I(t_1) = \{p_1\}$$

$$O(t_1) = \{p_2, p_3, p_5\}$$

$$I(t_2) = \{p_2, p_3, p_5\}$$

$$O(t_2) = \{p_5\}$$

$$I(t3) = \{p4\}$$

$$O(t3) = \{p4\}$$

$$I(t4) = \{p4\}$$

$$O(t4) = \{p2, p3\}$$

Ví dụ 7.2:

Cấu trúc của một Petri net $C = (P, T, I, O)$ với

$$P = \{p1, p2, p3, p4, p5, p6\}$$

$$T = \{t1, t2, t3, t4, t5\}$$

$$I(t1) = \{p1\}$$

$$O(t1) = \{p2, p3\}$$

$$I(t2) = \{p2, p3\}$$

$$O(t2) = \{p3, p5, p5\}$$

$$I(t3) = \{p2, p3\}$$

$$O(t3) = \{p2, p4\}$$

$$I(t4) = \{p4, p5, p5, p5\}$$

$$O(t4) = \{p4\}$$

$$I(t5) = \{p2\}$$

$$O(t5) = \{p6\}$$

7.2.2. Petri net graph

Hầu hết các nghiên cứu lý thuyết về Petri net đều dựa trên định nghĩa chuẩn tắc của cấu trúc Petri net ở trên. Tuy nhiên, việc biểu diễn đồ họa của cấu trúc Petri net sẽ làm cho việc mô tả khái niệm Petri net trở nên dễ dàng hơn.

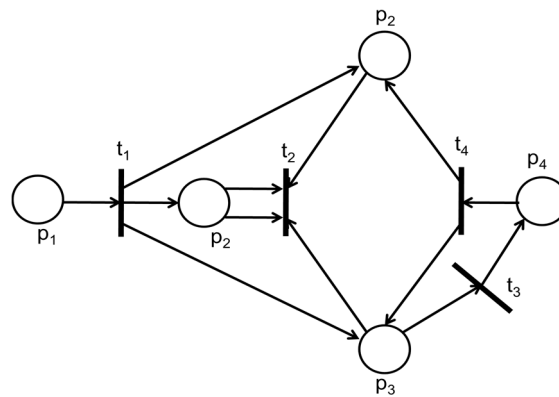
Cấu trúc Petri net bao gồm các vị trí và các chuyển tiếp. Tương ứng với các khái niệm đó, Petri net graph có hai loại nút. Một nút dạng hình tròn biểu thị một vị trí, một thanh hình chữ nhật biểu thị cho một chuyển tiếp. Các cung tròn có hướng kết nối các vị trí và các vị trí và các chuyển tiếp, với một số cung có hướng từ các vị trí tới các chuyển tiếp và một số có hướng từ các chuyển tiếp tới các vị trí. Một cung tròn có hướng từ một vị trí p_i tới chuyển tiếp t_j định nghĩa vị trí p_i là một đầu vào của chuyển tiếp t_j . Một vị trí đầu ra được biểu thị bởi một cung tròn từ một chuyển tiếp tới vị trí đó.

Petri net là một multigraph, do nó cho phép nhiều cung tròn từ một nút của graph tới một nút khác. Thêm vào đó, do các cung tròn là có hướng, Petri net là một multigraph có hướng (directed multigraph). Các nút của graph có thể được phân chia thành 2 tập (tập các vị trí và tập các chuyển tiếp) sao cho mỗi cung tròn có hướng từ một phần tử của một tập (vị trí hoặc chuyển tiếp) tới một phần tử của tập còn lại (chuyển tiếp hoặc vị trí), Petri net là một multigraph có hướng song phương (bipartite directed multigraph). Ta có thể gọi một cách ngắn gọn là Petri net graph.

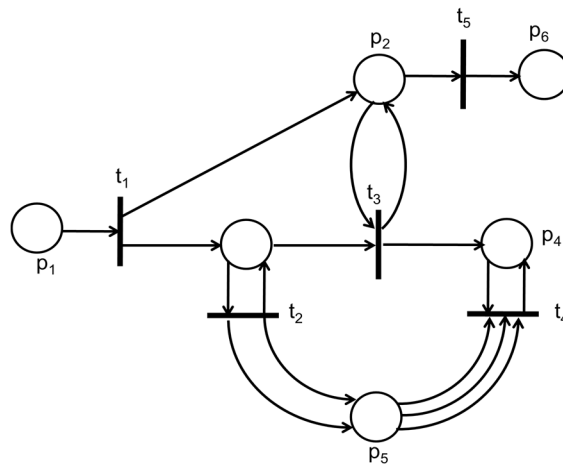
Định nghĩa 2

Một Petri net graph G là một multigraph có hướng song phương, $G = (V, A)$, trong đó $V = \{v_1, v_2, \dots, v_s\}$ là tập các đỉnh và $A = \{a_1, a_2, \dots, a_r\}$ là một túi các cung tròn có hướng, $a_i = (v_j, v_k)$, với $v_j, v_k \in V$. Tập V có thể được phân chia thành hai tập P và T tách rời nhau sao cho $V = P \cup T$, $P \cap T = \emptyset$, và với mỗi cung tròn $a_i \in A$, nếu $a_i = (v_j, v_k)$, thì hoặc $v_j \in P$ và $v_k \in T$ hoặc $v_j \in T$ và $v_k \in P$.

Hình 7.6 và 7.7 là các Petri net graph tương ứng với các cấu trúc Petri net trong ví dụ 1 và ví dụ 2



Hình 7.6 Petri net ứng với ví dụ 1



Hình 7.7 Petri net ứng với ví dụ 2

7.2.3 Đánh dấu Petri net

Một đánh dấu (marking) μ là một ấn định các thẻ (token) vào các vị trí (place) của một Petri net. Thẻ (token) là một khái niệm nguyên thủy cho Petri net (cũng như các khái niệm vị trí và chuyển tiếp). Các thẻ được ấn định, và có thể được coi là cư trú trong các vị trí (place) của Petri net. Các thẻ được dùng để định nghĩa sự hoạt động của Petri net.

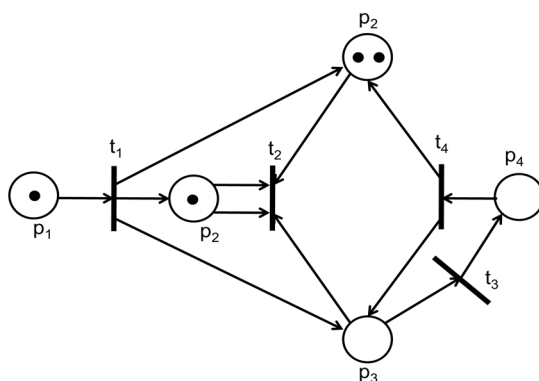
Định nghĩa 3

Một đánh dấu μ của một Petri net $C = (P, T, I, O)$ là một hàm số từ tập các vị trí P tới tập các số nguyên không âm N . $\mu = P \rightarrow N$

Đánh dấu μ cũng có thể được định nghĩa là một véc tơ n phần tử, $\mu = (\mu_1, \mu_2, \dots, \mu_n)$, trong đó $n = |P|$ và $\mu_i \in N$, $i = 1, 2, \dots, n$. Véc tơ μ cho biết số lượng nhãn trong mỗi vị trí p_i của Petri net. Số thẻ trong vị trí p_i là μ_i , $i = 1, 2, \dots, n$. Việc định nghĩa đánh dấu như là một hàm số và như một véc tơ có mối quan hệ rất tường minh $\mu(p_i) = \mu_i$.

Một Petri net được đánh dấu $M = (C, \mu)$ bao gồm một cấu trúc Petri net $C = (P, T, I, O)$ và một đánh dấu μ . Cũng có thể viết dưới dạng $M = (P, T, I, O, \mu)$.

Trong Petri net graph, các thẻ được biểu diễn bằng các chấm nhỏ (\bullet) trong các vòng tròn biểu diễn các vị trí (place) của Petri net. Hình 1.3 là một ví dụ về biểu diễn hình họa một Petri net được đánh dấu.



Hình 7.8: Petri net được đánh dấu.

Cấu trúc Petri net trong hình 7.8 tương tự như trong hình 7.6. Đánh dấu là $(1, 2, 0, 0, 1)$

Do số lượng thẻ có thể gán cho một vị trí của Petri net là không giới hạn, các đánh dấu của Petri net có thể là vô cùng. Tập tất cả các đánh dấu cho một Petri net với n vị trí đơn giản là tập của tất cả các véc tơ n phần tử, N^n .

7.2.4 Các luật hoạt động của Petri net

Sự hoạt động của Petri net được điều khiển bằng số lượng và sự phân bố của các thẻ trong Petri net. Các thẻ cư trú trong các vị trí và điều khiển sự hoạt động của mạng. Petri net hoạt động bằng cách chạy (fire) các chuyển tiếp. Một chuyển tiếp chạy bằng cách xóa bỏ các thẻ trong các vị trí đầu vào và tạo ra các thẻ mới và phân bố chúng tới các vị trí đầu vào của nó.

Một chuyển tiếp có thể chạy nếu nó sẵn sàng. Một chuyển tiếp là sẵn sàng nếu mỗi vị trí đầu vào của nó có số thẻ lớn hơn hoặc bằng số cung tròn từ vị trí tới chuyển tiếp. Ví dụ, nếu đầu vào cho một chuyển tiếp t_4 là hai vị trí p_1 và p_2 , thì t_4 là sẵn sàng nếu p_1 có ít nhất một thẻ và p_2 có ít nhất một thẻ. Với chuyển tiếp t_7 với túi đầu vào là $\{p_6, p_6, p_6\}$, vị trí p_6 phải có ít nhất 3 thẻ để chuyển tiếp t_7 có thể sẵn sàng.

Định nghĩa 4

Một chuyển tiếp $t_j \in T$ trong một Petri net được đánh dấu $C = (P, T, I, O)$ với đánh dấu μ là sẵn sàng nếu với tất cả các vị trí $p_i \in P$,

$$\mu(p_i) \geq \#(p_i, I(t_j))$$

Một chuyển tiếp chạy (fire) bằng cách xóa bỏ các thẻ khỏi các vị trí đầu vào của nó, số thẻ bị xóa bỏ khỏi một vị trí đầu vào bằng số cung tròn từ vị trí đó tới chuyển tiếp chạy, và gửi thêm vào các vị trí đầu ra của nó số thẻ bằng số cung tròn nối từ chuyển tiếp tới vị trí đầu ra.

Nói chung việc chạy một chuyển tiếp sẽ thay đổi đánh dấu μ của Petri net thành một đánh dấu mới μ' . Chú ý rằng do chỉ những chuyển tiếp sẵn sàng mới có thể chạy, số thẻ trong mỗi vị trí luôn là không âm khi một chuyển tiếp chạy. Việc chạy một chuyển tiếp sẽ không bao giờ cố gắng xóa bỏ một thẻ không có tại một vị trí. Nếu như không có đủ số thẻ tại một vị trí đầu vào của một chuyển tiếp, thì chuyển tiếp đó sẽ không sẵn sàng và không thể chạy.

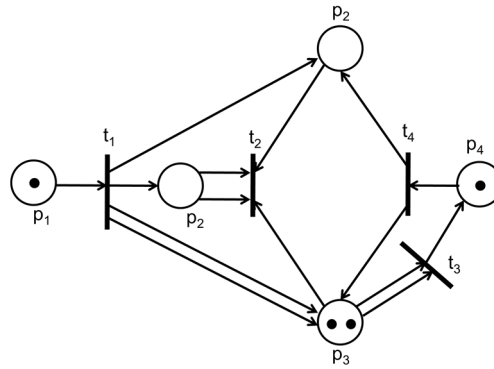
Định nghĩa 5

Một chuyển tiếp t_j trong một Petri net được đánh dấu với đánh dấu μ có thể chạy khi nó sẵn sàng. Việc chạy một chuyển tiếp sẵn sàng sẽ dẫn đến một đánh dấu mới μ' được định nghĩa như sau:

$$\mu'(p_i) = \mu(p_i) - \#(p_i, I(t_j)) + \#(p_i, O(t_j))$$

Ví dụ:

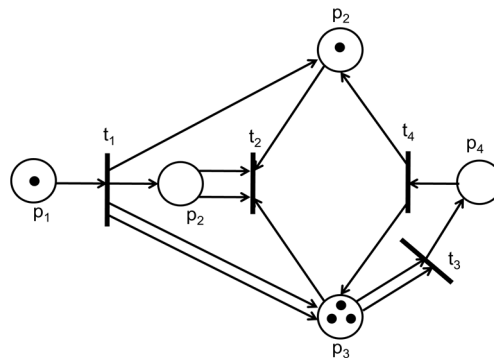
Mạng Petri net được đánh dấu như hình 7.9.



Hình 7.9: Petri net đánh dấu

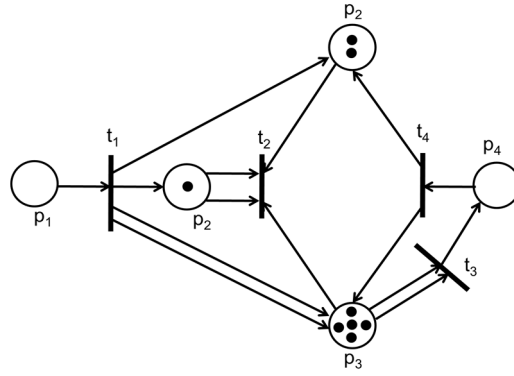
Hình 7.9 Một Petri net được đánh dấu dùng để mô tả luật chạy. Các chuyển tiếp t_1 , t_3 và t_4 đều sẵn sàng

Với đánh dấu như trong hình vẽ, có 3 chuyển tiếp sẵn sàng là t_1 , t_3 , và t_4 . Chuyển tiếp t_2 không sẵn sàng bởi vì không có thẻ nào trong vị trí p_2 hay p_3 , là các vị trí đầu vào của chuyển tiếp t_2 . Do các chuyển tiếp t_1 , t_3 , và t_4 sẵn sàng, bất cứ chuyển tiếp nào cũng có thể chạy. Nếu chuyển tiếp t_4 chạy, nó sẽ xóa bỏ một thẻ khỏi mỗi vị trí đầu vào (là p_4) và thêm một thẻ vào mỗi vị trí đầu ra (là p_2 và p_3). Khi đó đánh dấu mới của có được từ việc chạy t_4 sẽ như hình 7.10.



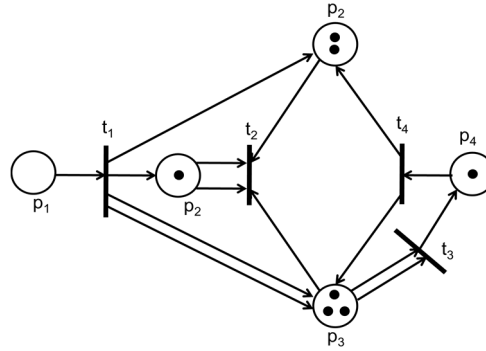
Hình 7.10: Kết quả đánh dấu từ việc chạy chuyển tiếp t_4 trong hình 7.9

Trong mạng Petri net được đánh dấu như trong hình 7.10, chỉ có chuyển tiếp t_1 và t_3 là sẵn sàng. Việc chạy chuyển tiếp t_1 sẽ xóa bỏ thẻ trong vị trí p_1 và thêm các thẻ vào các vị trí p_2 , p_3 và p_4 (thêm hai thẻ vào p_4 do p_4 là đầu ra kép của chuyển tiếp t_1). Việc này sẽ tạo ra một đánh dấu như trong hình 7.11.



Hình 7.11: Kết quả đánh dấu từ việc chạy chuyển tiếp t_1 trong hình 7.10

Trong mạng Petri net này, các chuyển tiếp t_2 và t_3 sẵn sàng. Chạy chuyển tiếp t_3 sẽ tạo ra đánh dấu như trong hình 7.12, ở đó hai thẻ đã bị xóa bỏ khỏi vị trí p_4 và một thẻ được thêm vào vị trí p_5 .



Hình 7.12: Kết quả đánh dấu từ việc chạy chuyển tiếp t_3 trong hình 7.11

Việc chạy các chuyển tiếp có thể tiếp tục khi mà vẫn có ít nhất một chuyển tiếp sẵn sàng. Khi không có chuyển tiếp sẵn sàng nào, việc hoạt động của Petri net bị dừng.

7.2.5. Không gian trạng thái Petri net

Trạng thái của Petri net được định nghĩa bằng đánh dấu của nó. Việc chạy một chuyển tiếp thể hiện sự thay đổi trạng thái của Petri net bằng sự thay đổi đánh dấu của mạng. Không gian trạng thái của một Petri net với n vị trí là tập hợp của tất cả các đánh dấu, là N^n . Sự thay đổi trạng thái gây ra bởi chạy một chuyển tiếp được định nghĩa bởi hàm thay đổi δ được gọi là hàm trạng thái tiếp theo. Khi được áp dụng cho một đánh dấu (trạng thái) μ và một chuyển tiếp t_j , hàm này sẽ sinh ra một đánh dấu (trạng thái) mới nhận được bằng cách chạy chuyển tiếp t_j với đánh dấu μ . Do t_j chỉ có thể chạy nếu nó sẵn, $\delta(\mu, t_j)$ không được định nghĩa nếu t_j không sẵn sàng với đánh

đánh dấu μ . Nếu t_j sẵn sàng, $\delta(\mu, t_j) = \mu'$ với μ' là đánh dấu nhận được từ việc xóa bỏ các thẻ khỏi các đầu vào của t_j và thêm các thẻ vào các đầu ra của t_j .

Định nghĩa 6

Hàm trạng thái tiếp theo $\delta: N^n \times T \rightarrow N^n$ của một Petri net $C = (P, T, I, O)$ với đánh dấu μ và chuyển tiếp $t_j \in T$ được định nghĩa khi và chỉ khi

$$\mu(p_i) \geq \#(p_i, I(t_j))$$

với tất cả $p_i \in P$. Nếu $\delta(\mu, t_j)$ được định nghĩa, thì $\delta(\mu, t_j) = \mu'$, trong đó

$$\mu'(p_i) = \mu(p_i) - \#(p_i, I(t_j)) + \#(p_i, O(t_j))$$

với tất cả $p_i \in P$.

Cho một Petri net $C = (P, T, I, O)$ và một đánh dấu ban đầu μ^0 , chúng ta có thể chạy Petri net bằng cách chạy liên tiếp các chuyển tiếp. Chạy một chuyển tiếp sẵn sàng t_j với đánh dấu ban đầu sẽ tạo ra một đánh dấu mới $\mu^1 = \delta(\mu^0, t_j)$. Trong đánh dấu mới này, chúng ta có thể chạy bất cứ một chuyển tiếp sẵn sàng nào, ví dụ t_k , sẽ được một đánh dấu mới $\mu^2 = \delta(\mu^1, t_k)$. Quá trình này vẫn có thể tiếp tục khi vẫn có ít nhất một chuyển tiếp sẵn sàng với mỗi đánh dấu. Nếu chúng ta đạt đến một đánh dấu mà không có chuyển tiếp nào sẵn sàng, khi đó không có chuyển tiếp nào có thể chạy, hàm trạng thái tiếp theo không được định nghĩa cho tất cả các chuyển tiếp, và sự hoạt động của Petri net bị dừng.

Sự hoạt động của Petri net sinh ra hai chuỗi: chuỗi các đánh dấu ($\mu^0, \mu^1, \mu^2, \dots$) và chuỗi các chuyển tiếp đã chạy ($t_{j0}, t_{j1}, t_{j2}, \dots$). Hai chuỗi này liên hệ với nhau bằng mối quan hệ $\delta(\mu^k, t_{jk}) = \mu^{k+1}$, với $k = 1, 2, 3, \dots$. Với một chuỗi chuyển tiếp và một đánh dấu μ^0 , chúng ta có thể dễ dàng nhận được một chuỗi các đánh dấu cho sự hoạt động của Petri net, và ngoại trừ một vài trường hợp thoái hóa, với một chuỗi các đánh dấu, chúng ta có thể nhận được một chuỗi các chuyển tiếp. Cả hai chuỗi này do đó cung cấp một bản ghi sự hoạt động của Petri net.

Với một đánh dấu μ , có một tập các chuyển tiếp sẵn sàng và có thể chạy. Kết quả của việc chạy một chuyển tiếp với đánh dấu μ là một chuyển tiếp mới μ' . Chúng ta nói rằng μ' là có thể đạt được ngay lập tức từ μ , nghĩa là chúng ta có thể ngay lập tức nhận được trạng thái μ' từ trạng thái μ .

Định nghĩa 7

Với một Petri net $C = (P, T, I, O)$ với đánh dấu μ , một đánh dấu μ' là có thể đạt đến được ngay lập tức từ μ nếu tồn tại một chuyển tiếp $t_j \in T$ sao cho $\delta(\mu, t_j) = \mu'$.

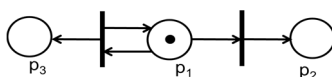
Chúng ta có thể mở rộng khái niệm này để định nghĩa tập các đánh dấu có thể đạt đến với một mạng Petri net cho trước. Nếu μ' là có thể đạt đến được ngay lập tức từ μ và μ'' là có thể đạt đến được ngay từ μ' , thì khi đó ta nói rằng μ'' là có thể đạt đến từ μ . Chúng ta định nghĩa tập có thể đạt đến $R(C, \mu)$ của Petri net C với đánh dấu μ là tất cả các đánh dấu có thể đạt đến được từ μ . Một đánh dấu μ' là thuộc về $R(C, \mu)$ nếu có một chuỗi nào đó các chuyển tiếp chạy có thể thay đổi đánh dấu μ thành đánh dấu μ' .

Định nghĩa 8

Tập có thể đạt đến $R(C, \mu)$ của một mạng Petri net $C = (P, T, I, O)$ với đánh dấu μ là tập nhỏ nhất của các đánh dấu thỏa mãn

- $\mu \in R(C, \mu)$
- Nếu $\mu' \in R(C, \mu)$ và $\mu'' = \delta(\mu', t_j)$ với $t_j \in T$, thì $\mu'' \in R(C, \mu)$.

Với Petri net như trong hình 7.13, và với đánh dấu là $\mu = (1,0,0)$, hai đánh dấu có thể đạt đến được ngay là $(0,1,0)$ và $(1,0,1)$. Từ đánh dấu $(0,1,0)$ sẽ không đến được một đánh dấu nào nữa do không có chuyển tiếp nào sẵn sàng. Tuy nhiên từ đánh dấu $(1,0,1)$, ta có thể đến được $(0,1,1)$ và $(1,0,2)$.



Hình 7.13 Một Petri net được đánh dấu

7.2.6 Biểu diễn ma trận của Petri net

Có thể định nghĩa một Petri net dùng hai ma trận D^+ và D^- để biểu diễn hàm đầu vào và đầu ra. Mỗi ma trận có kích thước $m \times n$ (m là số chuyển tiếp và n là số vị trí). Ta có thể định nghĩa ma trận đầu vào D^- và ma trận đầu ra như sau:

$$D^-[j,i] = \#(p_i, I(t_j))$$

$$D^+[j,i] = \#(p_i, O(t_j))$$

Việc định nghĩa Petri net dùng ma trận (P, T, D^-, D^+) tương đương với dạng định nghĩa chuẩn mà ta đã sử dụng ở trên, và cho phép việc định nghĩa được viết lại dưới dạng ma trận và véc tơ.

Gọi $e[j]$ là một véc tơ m thành phần trong đó các thành phần đều có giá trị 0 trừ thành phần thứ j có giá trị 1. Chuyển tiếp t_j được biểu diễn bằng véc tơ $e[j]$ đó.

Khi đó một chuyển tiếp t_j là sẵn sàng với đánh dấu μ nếu

$$\mu \geq e[j].D^-$$

Và kết quả của việc chạy chuyển tiếp t_j với đánh dấu μ (nếu chuyển tiếp sẵn sàng), được tính như sau

$$\begin{aligned}\delta(\mu, t_j) &= \mu - e[j]D^- + e[j]D^+ \\ &= \mu + e[j](-D^- + D^+) \\ &= \mu + e[j]D\end{aligned}$$

Với D được định nghĩa là ma trận thay đổi tổng hợp $D = D^+ - D^-$.

Với một dãy các chuyển tiếp chạy $\sigma = t_{j1}, t_{j2}, \dots, t_{jk}$ ta có

$$\begin{aligned}\delta(\mu, \sigma) &= \delta(\mu, t_{j1} t_{j2} \dots t_{jk}) \\ &= \mu + e[j1]D + e[j2]D + \dots + e[jk]D \\ &= \mu + (e[j1] + e[j2] + \dots + e[jk])D \\ &= \mu + f(\sigma)D\end{aligned}$$

Véc tơ $f(\sigma) = e[j1] + e[j2] + \dots + e[jk]$ được gọi là véc tơ chạy của dãy chuyển tiếp $t_{j1}, t_{j2}, \dots, t_{jk}$. Thành phần thứ i của $f(\sigma)$, $f(\sigma)_i$ là số lần chuyển tiếp t_i chạy trong dãy chuyển tiếp $t_{j1}, t_{j2}, \dots, t_{jk}$. Véc tơ chạy do đó là véc tơ các số nguyên không âm.

7.2.7 Mô hình hóa với Petri net

Petri net có thể được dùng để mô hình hóa các tính chất cơ bản của hệ thống động học được điều khiển theo sự kiện một cách hiệu quả

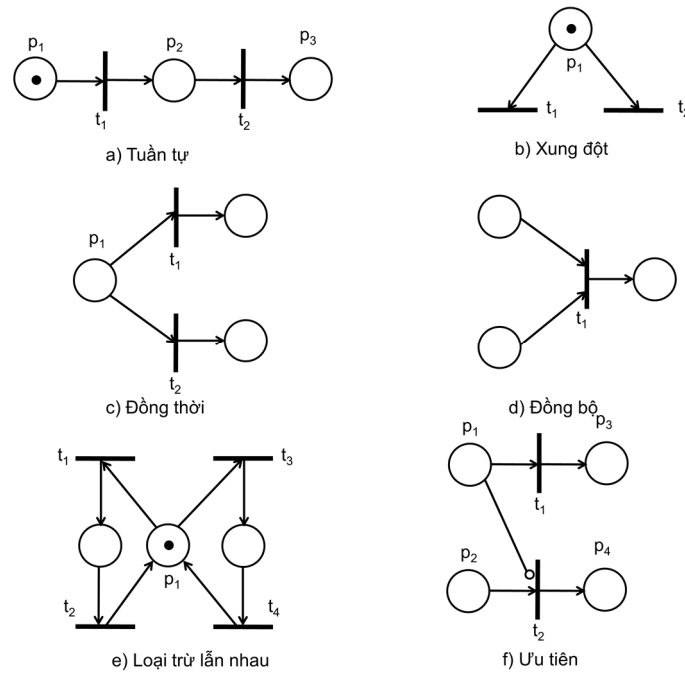
- Hoạt động tuần tự (sequential execution): Trong hình 7.14(a), chuyển tiếp t_2 chỉ chạy được sau khi chuyển tiếp t_1 đã chạy. Điều này tạo ra một ràng buộc về thứ tự “ t_2 sau t_1 ”. Ràng buộc thứ tự này là điển hình của quá trình hoạt động của các phần trong một hệ động học. Đồng thời cấu trúc Petri net này cũng mô phỏng mối liên hệ tự nhiên giữa các hành động.
- Xung đột (conflict): chuyển tiếp t_1 và t_2 xung đột với nhau trong hình 7.14(b). Cả hai chuyển tiếp đều sẵn sàng, tuy nhiên việc chạy một chuyển tiếp sẽ làm vô hiệu tính sẵn sàng của chuyển tiếp còn lại. Tình huống này sẽ xảy ra, ví dụ, khi một máy phải lựa chọn một bộ phận trong các loại bộ phận khác nhau, hoặc là một bộ phận phải lựa chọn một máy trong các máy khác nhau. Vấn đề xung đột có thể được giải quyết bằng cách lựa chọn hoàn toàn ngẫu nhiên hoặc theo xác suất bằng cách thiết lập các xác suất chạy cho mỗi chuyển tiếp.
- Tính đồng thời (concurrency): trong hình 7.14(c), chuyển tiếp t_1 và t_2 có tính chất đồng thời. Tính đồng thời là một thuộc tính quan trọng của các tương tác hệ

thống. Chú ý rằng điều kiện cần thiết cho các chuyển tiếp là đồng thời là tồn tại một chuyển tiếp phân nhánh gửi một thẻ vào trong hai hoặc nhiều hơn vị trí.

- Đồng bộ hóa (synchronization): Trong một hệ thống động học, việc một sự kiện yêu cầu nhiều nguồn tài nguyên khác nhau là khá phổ biến. Việc đồng bộ hóa các nguồn khác nhau có thể được thực hiện bằng các chuyển tiếp như trong hình 7.14(d). Ở đây, chuyển tiếp t_1 sẵn sàng chỉ khi mỗi vị trí p_1 và p_2 đều nhận một thẻ. Việc các thẻ đến mỗi vị trí có thể là kết quả của một dãy phức tạp các hoạt động ở một nơi nào đó trong phần còn lại của mô hình Petri net. Về bản chất, chuyển tiếp t_1 mô phỏng hành động kết hợp.

- Loại trừ lẫn nhau (mutually exclusive): Hai quá trình được gọi là loại trừ lẫn nhau nếu chúng không thể được thực hiện tại cùng một thời điểm, do các ràng buộc về việc sử dụng các nguồn tài nguyên bị chia sẻ. Hình 7.14(e) chỉ ra một cấu trúc loại trừ lẫn nhau. Ví dụ, một robot có thể được chia sẻ bởi hai máy cho việc lắp tải và gỡ tải. Hai cấu trúc này là loại trừ lẫn nhau song song và loại trừ lẫn nhau tuần tự.

- Ưu tiên: Petri net được nói tới ở trên không có một cơ chế nào để biểu thị sự ưu tiên. Để thực hiện được việc này, người ta giới thiệu khái niệm cung chặn (inhibitor arc). Cung chặn kết nối một vị trí đầu vào tới một chuyển tiếp, và được biểu thị một cách hình ảnh là một cung tròn kết thúc bởi một hình tròn nhỏ. Sự hiện diện của cung chặn kết nối vị trí đầu vào với một chuyển tiếp làm thay đổi điều kiện sẵn sàng của chuyển tiếp. Khi đó, một chuyển tiếp là sẵn sàng nếu mỗi vị trí đầu vào nối với chuyển tiếp bằng cung tròn có hướng bình thường chứa ít nhất số nhãn bằng số kết nối, và không có nhãn nào ở vị trí đầu vào được kết nối với chuyển tiếp bằng cung chặn. Luật chạy một chuyển tiếp, đối với các vị trí đầu vào bình thường thì không thay đổi. Tuy nhiên việc chạy đó không làm thay đổi sự đánh dấu của các vị trí được kết nối bằng cung chặn. Hình 7.14(f) mô tả một Petri net với cung chặn. Chuyển tiếp t_1 sẵn sàng nếu vị trí p_1 chứa một nhãn, trong khi t_2 sẵn sàng nếu p_2 chứa một nhãn và không có nhãn nào ở p_1 . Điều này tạo ra sự ưu tiên của t_1 so với t_2 .



Hình 7.14: Petri net mô tả các đặc tính của hệ thống

7.2.8 Petri net bậc cao

Trong Petri net chuẩn, không có sự phân biệt giữa các nhãn. Vì điều này mà Petri net có nhiều hạn chế trong việc mô phỏng các cấu hình rất lớn và không cấu trúc. Để khắc phục điều này, các Petri net bậc cao (High level Petri net) đã được phát triển. Sau đây sẽ giới thiệu sơ lược một số loại Petri net bậc cao.

Petri net màu (Colored Petri nets)

Được giới thiệu bởi Kurt Jensen, Petri net màu (CPN) có các thẻ được đi kèm với các màu để chỉ ra sự nhận dạng của các thẻ. Thêm vào đó, mỗi vị trí và mỗi chuyển tiếp được gán một tập các màu. Một chuyển tiếp có thể chạy ứng với mỗi màu của nó. Bằng việc chạy các chuyển tiếp, các thẻ được xóa bỏ khỏi các vị trí đầu vào và được thêm vào các vị trí đầu ra tương tự như với Petri net thông thường, ngoại trừ sự phụ thuộc được chỉ ra giữa màu chạy chuyển tiếp và màu của các thẻ được bao gồm. Màu gắn với một thẻ có thể được thay đổi bởi việc chạy các chuyển tiếp và nó thường biểu thị một giá trị dữ liệu phức. Petri net màu dẫn tới các mô hình mạng cô đọng bằng cách dùng khái niệm màu.

Petri net thời gian (Timed Petri Nets)

Sự cần thiết của việc bao gồm các biến thời gian trong các mô hình của rất nhiều loại hệ thống động học là rất rõ ràng do các hệ thống này bản chất tự nhiên là thời gian thực. Trên thực tế, hầu hết một sự kiện đều liên quan đến yếu tố thời gian. Khi một Petri net chứa một biến thời gian, nó trở thành Petri net thời gian (Timed Petri Net). Định nghĩa của Petri net thời gian bao gồm ba yếu tố sau:

- Cấu trúc hình học topo
- Đánh nhãn cấu trúc
- Các luật hoạt động

Cấu trúc hình học topo của Petri net thời gian nói chung là giống với Petri net thông thường. Việc đánh nhãn của Petri net thời gian bao gồm việc thiết lập các giá trị bằng số cho một hoặc nhiều các thứ sau:

- Các chuyển tiếp
- Các vị trí
- Các cung tròn kết nối các vị trí và các chuyển tiếp

Các luật hoạt động được định nghĩa khác nhau tùy theo cách Petri net được đánh nhãn với các biến thời gian. Các luật hoạt động định nghĩa cho một Petri net thời gian điều khiển quá trình di chuyển của các thẻ.

Có nhiều kiểu Petri net thời gian, hai trong những loại Petri net thời gian được sử dụng rộng rãi là Petri net thời gian xác định (Deterministic timed Petri net) và Petri net thời gian ngẫu nhiên (stochastic timed Petri net), trong đó các biến thời gian được kết hợp với các chuyển tiếp.

Petri net thời gian xác định (Deterministic timed Petri nets)

Việc giới thiệu các nhãn thời gian xác định vào Petri net đầu tiên được thực hiện bởi Ramchandani (Ramchandani 1974). Ở đây, nhãn thời gian được đặt vào mỗi chuyển tiếp, chỉ ra một thực tế rằng các chuyển tiếp thường được sử dụng để mô phỏng các hành động, và các hành động thì cần một khoảng thời gian để hoàn thành. Petri net mở rộng này được gọi là Petri net thời gian xác định (DTPN) và được dùng như một mô hình mở rộng để phân tích quá trình thực hiện của hệ thống. Phương pháp này thích hợp với một lớp giới hạn các hệ thống được gọi là mạng tự quyết định (decision-free nets). Về mặt cấu trúc, mỗi vị trí là một đầu vào của không nhiều hơn một chuyển tiếp, và chỉ là đầu ra của không nhiều hơn một chuyển tiếp.

Petri net thời gian ngẫu nhiên (Stochastic timed Petri nets)

Petri net thời gian ngẫu nhiên (STPN) là Petri net mà trong đó thời gian chạy ngẫu nhiên được gắn với các chuyển tiếp. Petri net thời gian ngẫu nhiên thực chất là một mô hình bậc cao tạo ra một quá trình ngẫu nhiên. Việc đánh giá quá trình hoạt động dựa trên Petri net thời gian ngẫu nhiên về cơ bản bao gồm việc mô hình hóa hệ thống bằng Petri net thời gian ngẫu nhiên và tạo ra một cách tự động một quá trình ngẫu nhiên không chế các đáp ứng của hệ thống. Quá trình ngẫu nhiên này sau đó được phân tích sử dụng các kỹ thuật đã biết. Petri net thời gian ngẫu nhiên là một mô hình đồ họa và mang lại rất nhiều tiện lợi cho người mô phỏng trong việc đạt được một mô hình bậc cao đáng tin cậy của hệ thống.

7.3 Statechart và stateflow

7.3.1 Nhược điểm của FSM

Như ta đã biết máy trạng thái hữu hạn FSM là một công cụ để mô tả hệ thống bằng cách đơn giản hóa các mô hình phức tạp sử dụng các giả thiết đơn giản. Cụ thể:

- + Hệ thống được mô hình hóa ở trong những điều kiện hữu hạn, gọi là trạng thái.
- + Hệ thống chỉ tồn tại trong các trạng thái xác định trước với các điều kiện thích hợp.
- + Hệ thống hoạt động bằng cách chuyển giữa các trạng thái qua các đường khác nhau gọi là chuyển tiếp
- + Các chuyển tiếp được thực hiện trong hệ thống nhờ sự tác động của các sự kiện.
- + Các chuyển tiếp

Nhược điểm của FSM truyền thống như sau:

- + Khó khăn khi mô tả hệ thống phức tạp.
- + Hỗ trợ rất kém trong việc truy cập đồng thời.

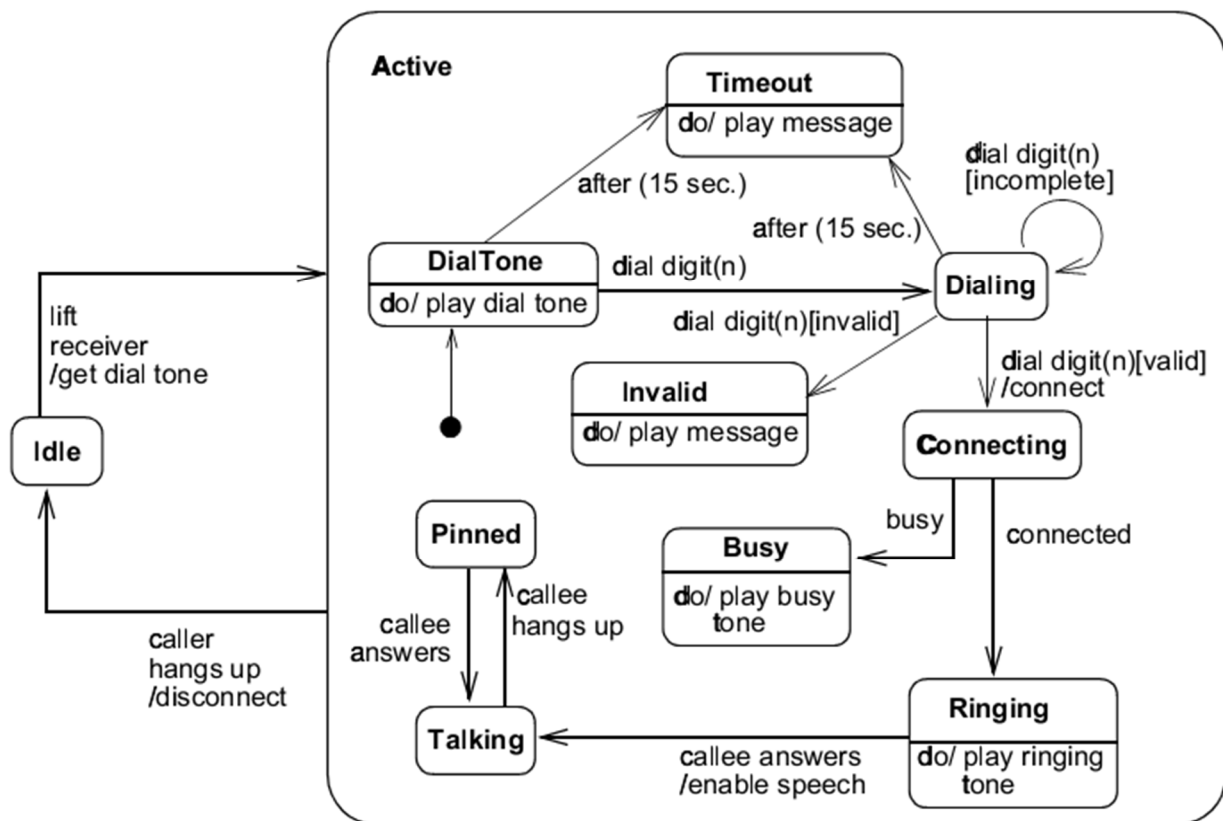
Để khắc phục những nhược điểm này biểu đồ trạng thái state chart đã ra đời.

7.3.2 Khái niệm state chart

Tương tự như FSM, State chart chính là máy trạng thái hữu hạn FSM được bổ sung thêm các trạng thái phân cấp, trạng thái đồng thời qua đó có thể mô tả các hệ thống phức tạp một cách đơn giản hơn.

State chart sử dụng để mô tả mọi trạng thái có thể của một hệ thống và cũng như của một đối tượng khi có các sự kiện tác động lên nó.

Ví dụ về state chart cho một điện thoại đơn giản:



Hình 7.15: Ví dụ về Statechart

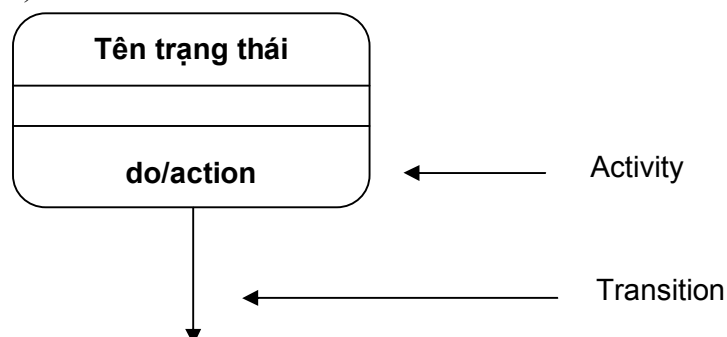
7.3.3 Trạng thái (state)

a. Khái niệm

Một trạng thái là một trạng thái trong chu kỳ sống của đối tượng, hay một tương tác của đối tượng thỏa mãn một số những điều kiện nhất định, thực hiện một số hành động hoặc chờ đợi một sự kiện nào đó.

b. Ký hiệu

Các trạng thái của đối tượng được biểu diễn bằng hình vuông với các góc tròn (như hình 7.16)



Hình 7.16: Ký hiệu trạng thái của đối tượng

Một biểu đồ trạng thái bắt đầu bằng "trạng thái ban đầu" và kết thúc bởi "trạng thái kết thúc". Tuy nhiên chúng ta không bắt buộc phải sử dụng chính xác 1 trạng thái đầu và 1 trạng thái kết thúc, trong biểu đồ trạng thái chúng ta có thể bắt đầu với nhiều "trạng thái ban đầu" và có thể không có "trạng thái kết thúc".

Điều này là hoàn toàn bình thường vì chúng ta có thể có những hệ thống mà hoạt động của nó không bao giờ kết thúc.

Trạng thái bắt đầu được ký hiệu bởi vòng tròn đặc màu đen, trạng thái kết thúc được ký hiệu bởi hai vòng tròn lồng vào nhau, vòng tròn trong đặc màu đen (hình 7.17)

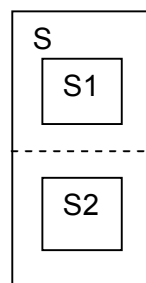


Hình 7.17: Trạng thái ban đầu và trạng thái kết thúc

c. Các trạng thái And-state và Or-state

- And-states: Bao gồm hai thành phần quan hệ "and" với nhau có nghĩa là khi trạng thái And-state được kích hoạt thì cả hai trạng thái con đều được kích hoạt

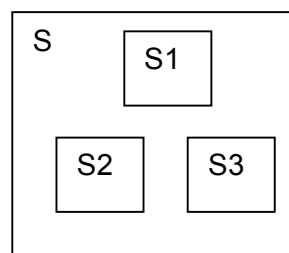
And-state được biểu diễn bằng hình chữ nhật và phân biệt hai trạng thái bằng đường nét đứt.



$S \text{ hoạt động } \Leftrightarrow S_1 \text{ hoạt động và } S_2 \text{ hoạt động}$

- Or-states: Bao gồm các thành phần có quan hệ Or với nhau. Có nghĩa là khi trạng thái Or-state được kích hoạt thì chỉ một trong các trạng thái con được kích hoạt.

Or-state được biểu diễn bởi hình vuông bên trong chứa các trạng thái con



S hoạt động \Leftrightarrow S1 hoạt động hoặc S2 hoạt động hoặc S3 hoạt động

7.3.4 Chuyển tiếp (Transition)

a. Khái niệm

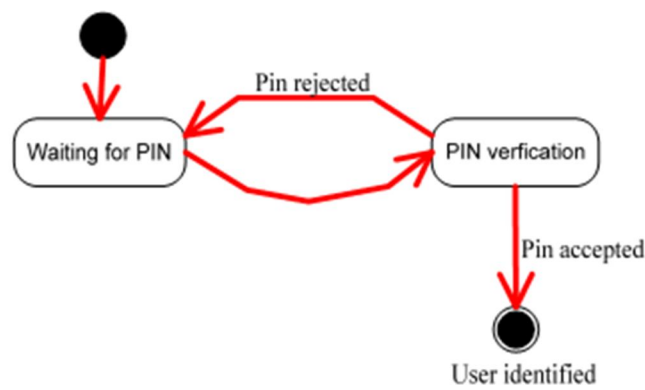
Transition biểu hiện mối quan hệ giữa hai trạng thái của một đối tượng. Khi đối tượng chuyển sang trạng thái tiếp theo thì transition sẽ bị hủy (fire)

b. Ký hiệu

Chuyển tiếp được ký hiệu bởi một mũi tên liền từ trạng thái nguồn đến trạng thái đích và thường được đánh dấu bởi nhãn có cấu trúc như sau:

event [điều kiện chờ]/biểu thức hoạt động

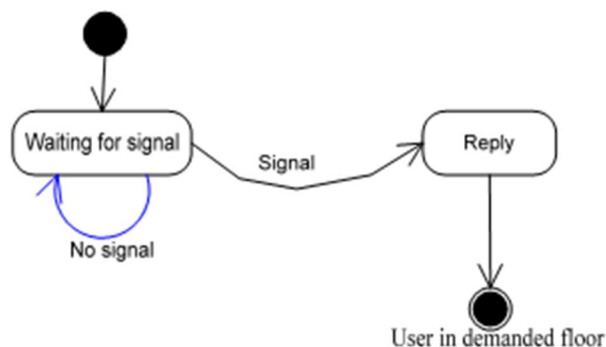
c. Ví dụ



Hình 7.18: Ví dụ về Transition

d. Các loại Transition

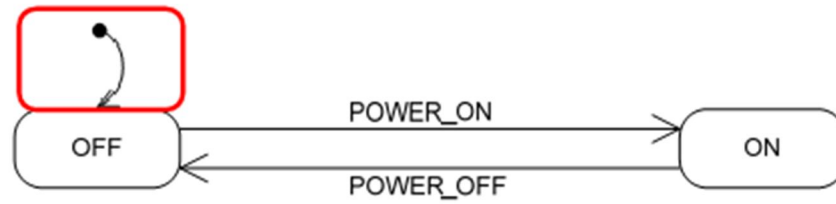
+ Self-Transition (Tự chuyển): là chuyển tiếp chuyển về chính trạng thái ban đầu:



Hình 7.19: Tự chuyển (Self-Transition)

+ Chuyển tiếp mặc định (default- transition)

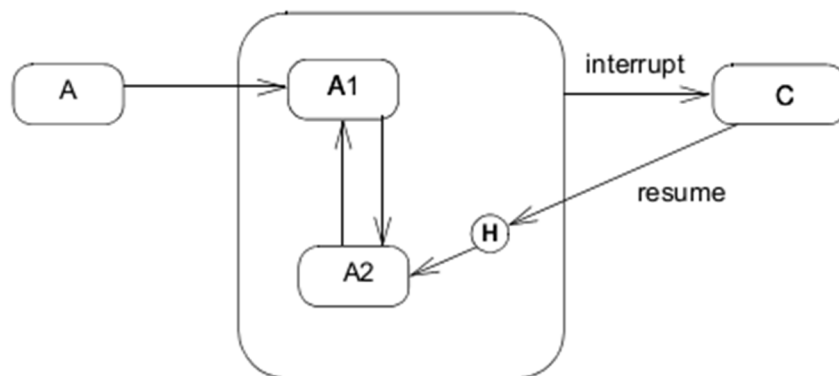
Là chuyển tiếp từ trạng thái ban đầu.



Hình 7.20: Chuyển tiếp mặc định (Default-Transition)

+ Chuyển tiếp có tính lịch sử

Một chuyển tiếp chuyển từ trạng thái ngoài của trạng thái tổng hợp về trạng thái tổng hợp cũ được gọi là chuyển tiếp có tính lịch sử và được ký hiệu bởi chữ H trong vòng tròn.



Hình 7.21: Chuyển tiếp có tính lịch sử

7.3.5 Sự kiện (Events)

a. Khái niệm

Một event là một tín hiệu tại một thời điểm xác định, các event có thể là tín hiệu nhận biết theo sườn (lên hoặc xuống), tín hiệu so sánh hoặc ngắt. Event có thể được tạo ra từ bên ngoài hoặc do nội tại trạng thái sinh ra.

Nguồn sinh ra các sự kiện nội tại có thể là: hoạt động trong trạng thái, sự kiện thời gian hoặc dùng cảm biến để xác định diễn biến của state, action, condition và data-item.

Event có thể được hình thành bằng cách kết hợp nhiều event và nhiều condition khác nhau.

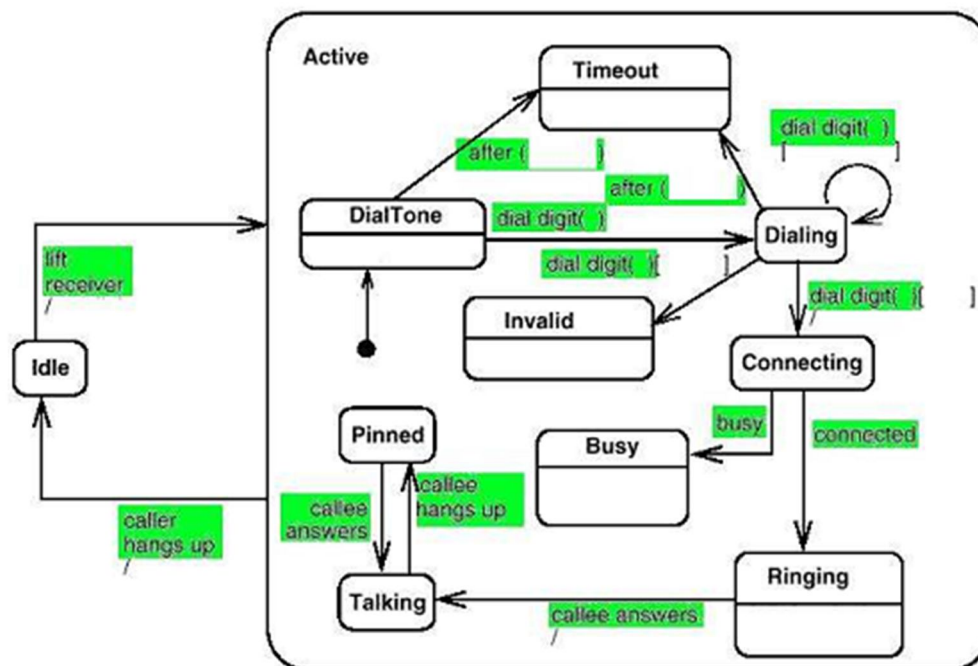
b. Ký hiệu

Một event được khai báo sử dụng cấu trúc như sau:

tên sự kiện (danh sách các tham số cách nhau bởi dấu ",")

Và được khai báo trên đường mũi tên của transition hoặc trong phạm vi của trạng thái như trên hình như: after(15 sec) hay dial digit(n).

c. Ví dụ



Hình 7.22: Cách khai báo event

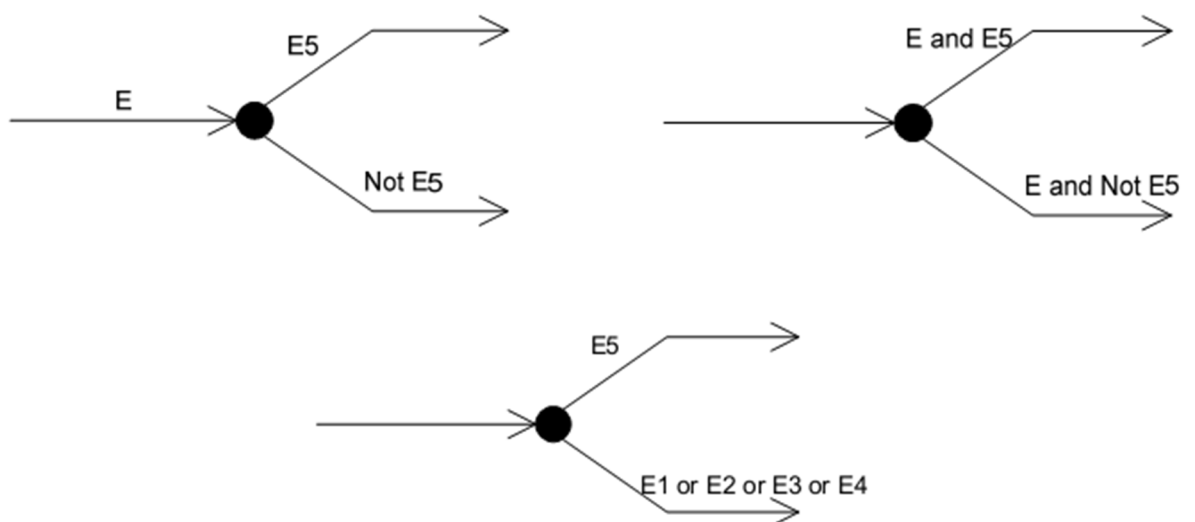
d. Các tính chất của event

- Phủ định của một event:

Để phủ định một event ta sử dụng toán tử NOT. Phủ định của event đã xác định có nghĩa là event đó không xảy ra.

- Các biểu thức của event được tính toán xác định theo các quy tắc ưu tiên thông thường của phép toán logic và dấu ngoặc đơn.

Ví dụ về phủ định một sự kiện



Hình 7.23: Phủ định sự kiện

Các toán tử ANY được sử dụng để phát hiện khi có một trong các event xảy ra trong nhiều event

Toán tử ALL được sử dụng khi cần phải có điều kiện tất cả các event cùng xảy ra.

7.3.6 Trigger

Triggers là các phần tử động trong biểu đồ trạng thái statechart. Trigger là nguồn gốc của các chuyển tiếp giữa các trạng thái hoặc các phản ứng tĩnh. Các sự kiện, các điều kiện hoặc sự kết hợp giữa chúng sẽ tạo nên các Trigger.

Trigger được ký hiệu bằng ký hiệu có hướng (các mũi tên) kết nối giữa hai trạng thái.

7.3.7 Điều kiện (Condition)

a. Khái niệm

Condition là các biểu thức logic, có giá trị ĐÚNG (TRUE) hoặc SAI (FALSE) trong suốt thời gian mà điều kiện được tồn tại. Các điều kiện có thể là sườn tín hiệu hoặc mức tín hiệu.

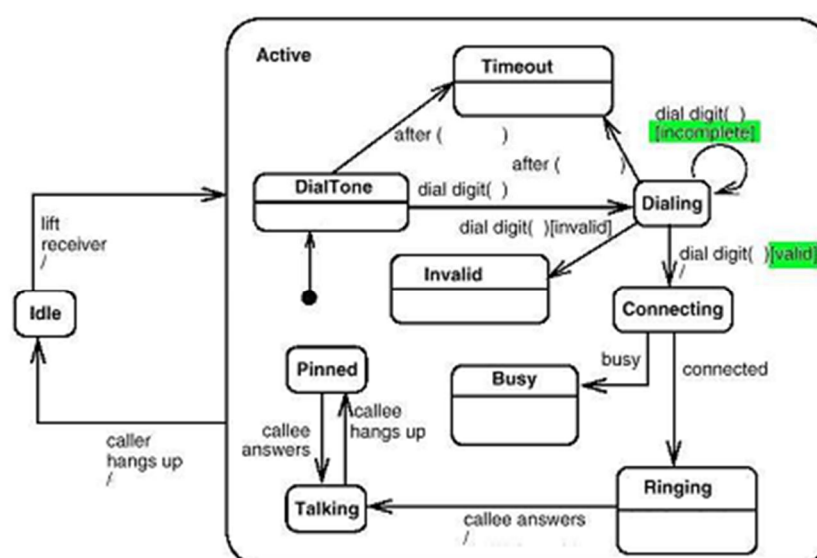
Các điều kiện có thể là các phần tử cơ bản (cảm biến, ...) hoặc là sự kết hợp của các hàm logic.

b. Ký hiệu

Các điều kiện được ký hiệu bởi cấu trúc cơ bản như sau và nằm trên mũi tên của transition hoặc trong vùng của trigger:

Event[Condition]

c. Ví dụ về condition



Hình 7.24: Ví dụ về ký hiệu condition

7.3.8 Biểu thức có mục dữ liệu (Data-item Expression)

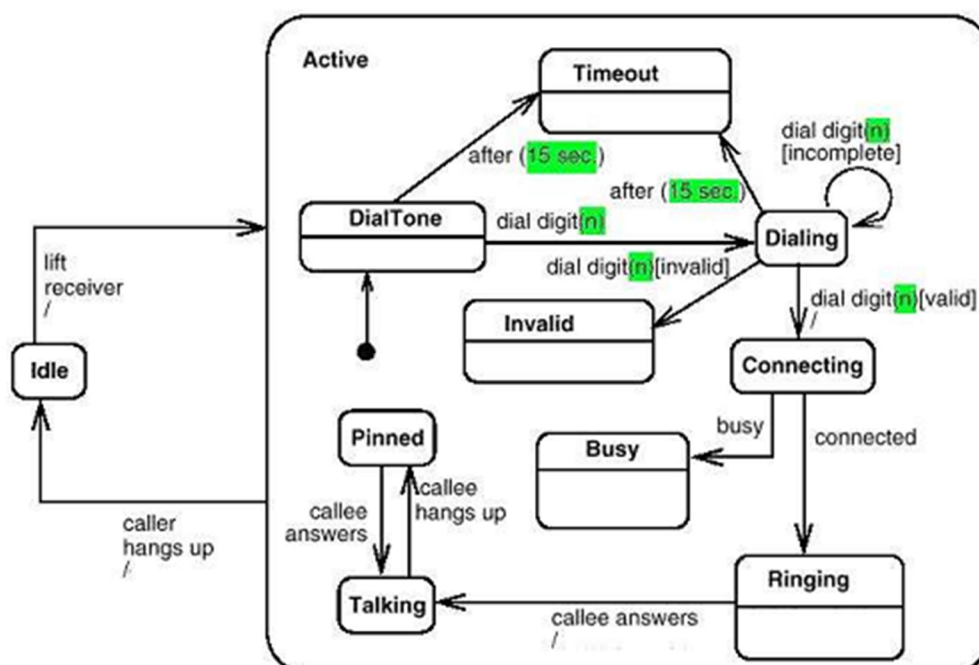
a. Khái niệm

Một mục dữ liệu (data-item) là một đơn vị thông tin mà ta có thể giả định cho giá trị thay đổi của kiểu và cấu trúc.

Chúng cũng hoàn toàn tương tự như các thành phần dữ liệu thông thường trong ngôn ngữ lập trình như: biến, hằng, ...

Mục dữ liệu có thể có nhiều kiểu giống nhau như: số liệu (số nguyên, số thực, bit, mảng bit), chuỗi và cấu trúc. Biểu thức kiểu số liệu cho phép người dùng sử dụng các hàm chức năng để tính toán.

b. Ví dụ



Hình 7.25: Ví dụ về sử dụng chỉ mục dữ liệu

7.3.9 Hoạt động (Action)

a. Khái niệm

Action là các hoạt động tức thời, kết quả của một vài trigger. Sự thay đổi giá trị của một điều kiện, mục dữ liệu hay hệ quả của các hoạt động khác là ví dụ cho các action.

Một action có thể là chuỗi nối tiếp của các action xảy ra đồng thời mà không phân biệt trình tự xuất hiện của nó.

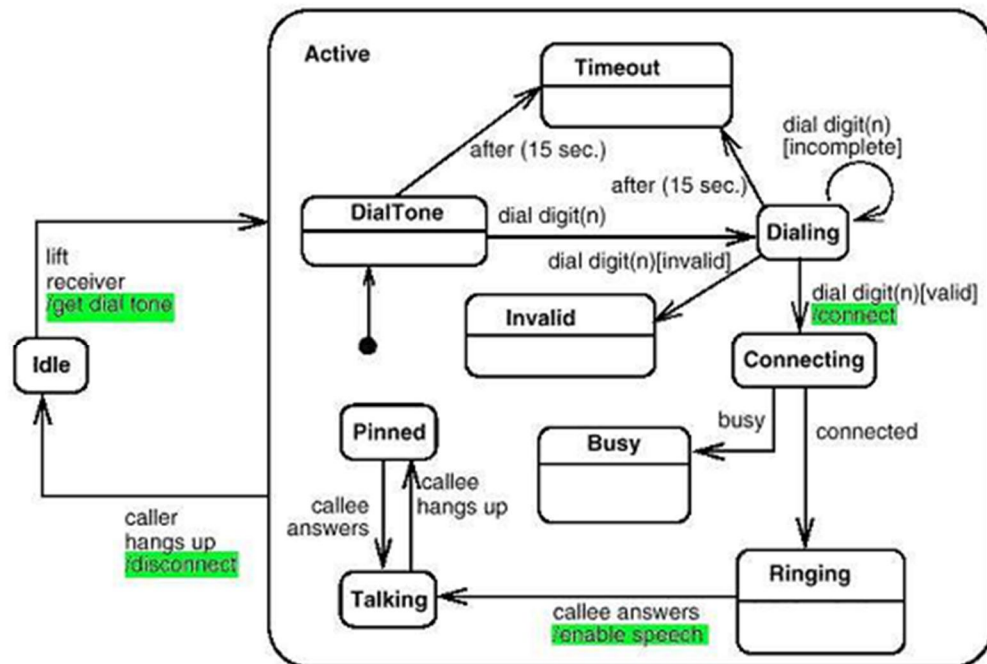
Các action có tính logic điều kiện có thể được mô tả dựa trên các điều kiện hoặc các event đồng thời.

b. Ký hiệu

Action được biểu diễn dưới dạng nhãn chữ số nằm trên đường transition hoặc trong phần action của trạng thái và có cấu trúc như sau:

event[condition]/action

c. Ví dụ



Hình 7.26: Ký hiệu action

7.3.10 Activities

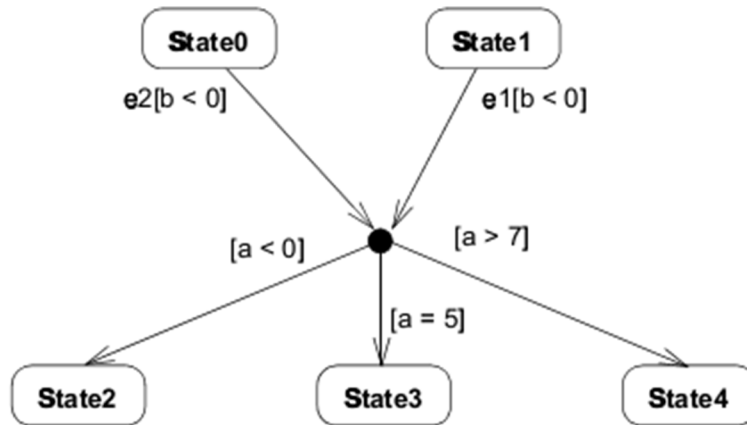
Activities là các hoạt động xảy ra trong khoảng thời gian gần như bằng không và có thể được điều khiển (bắt đầu hoặc kết thúc) thông qua các action cũng như có thể bị giám sát trạng thái.

Activities không được biểu diễn trong StateChart, nó xuất hiện như một phần của các phần tử khác

7.3.11 Sự kết hợp các chuyển tiếp

Các chuyển tiếp có thể kết hợp theo nhiều kiểu tùy thuộc vào các điều kiện khác nhau. Sau đây là các loại kết hợp.

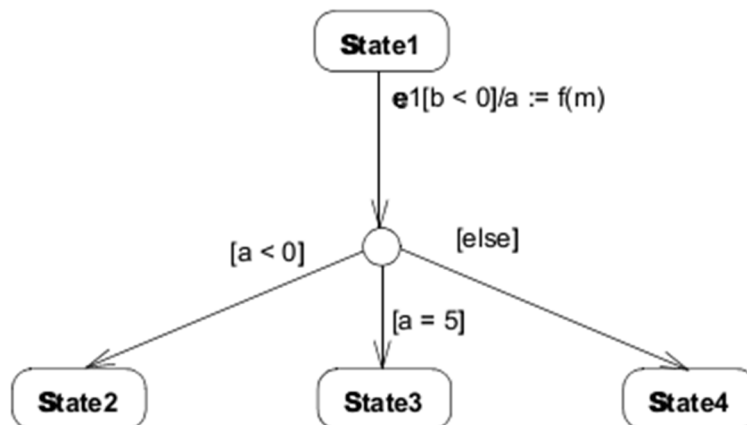
a. Kết hợp sử dụng điểm kết nối



Hình 7.27: Kết hợp chuyển tiếp sử dụng điểm kết nối

Nếu đối tượng đang ở trạng thái 1 và $b < 0$ khi chuyển tiếp $e1$ xảy ra, chuyển tiếp sẽ được thực hiện nếu một trong 3 điều kiện chờ xảy ra ($a < 0$ hoặc $a = 5$ hoặc $a > 7$). Nếu $a = 6$ sẽ không có chuyển tiếp nào được thực hiện.

b. Điểm lựa chọn linh hoạt



Hình 7.28: Kết hợp chuyển tiếp sử dụng điểm lựa chọn linh hoạt

Nếu đối tượng đang ở trạng thái 1, sự kiện $e1$, điều kiện chờ là $b < 0$, biểu thức kích hoạt chuyển tiếp là $a := f(m)$. Nếu điều kiện chờ $b < 0$ thỏa mãn. Sự kiện $e1$ xảy ra, các điều kiện của biểu thức sẽ lựa chọn 3 điều kiện tiếp theo. Tùy thuộc

$a < 0$: chuyển tiếp sẽ chuyển sang trạng thái 2

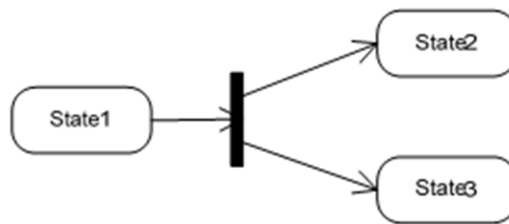
$a = 5$: chuyển tiếp sẽ chuyển sang trạng thái 3

Nếu không thì chuyển tiếp sẽ chuyển sang trạng thái 4

c. Sử dụng Fork

Khi từ một transition trạng thái được chuyển sang hai hoặc nhiều hơn các trạng thái khác nhau chúng ta sử dụng Fork

Fork trong statechart được ký hiệu bởi thanh thẳng đứng màu đen.

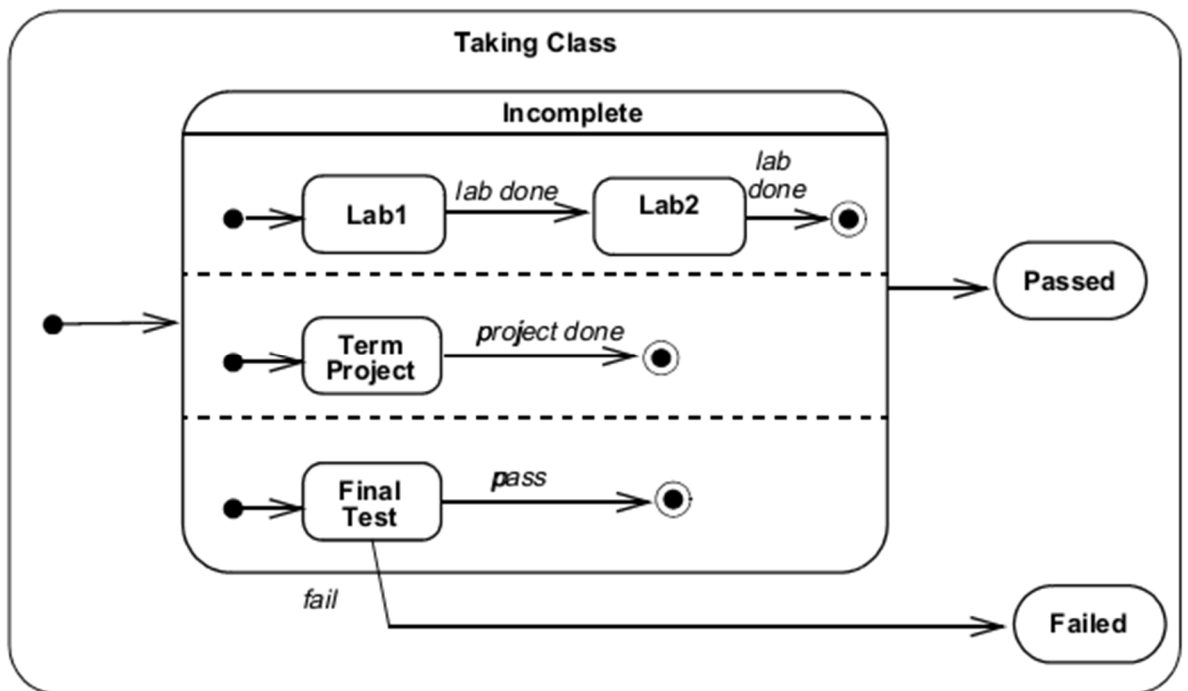


Hình 7.29: Sử dụng Fork

7.3.12 Trạng thái đồng thời (Concurrence state)

Như ta đã biết, Statechart cải thiện so với FSM là đã bổ sung các trạng thái tương đương. Các trạng thái tương đương là các trạng thái xảy ra đồng thời. Chúng được phân tách nhau bởi đường nét đứt.

Ví dụ các trạng thái Concurrence state

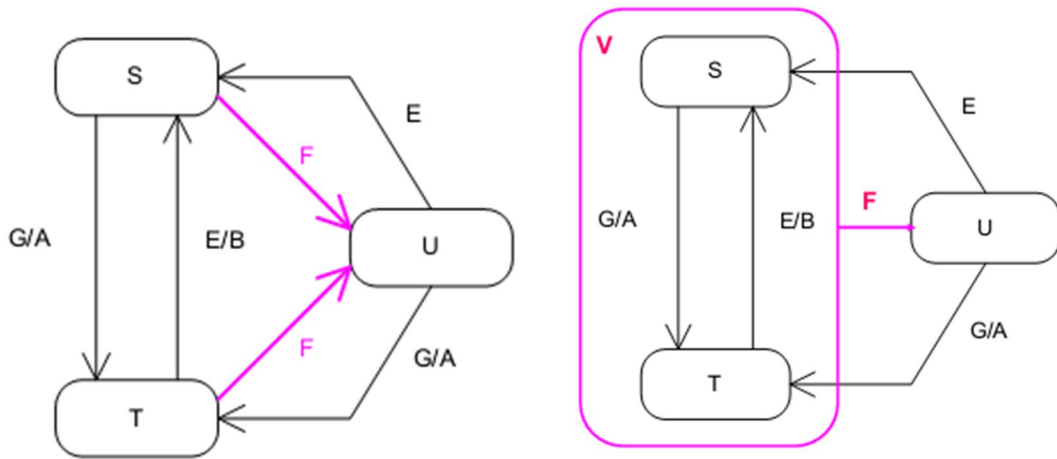


Hình 7.30: Các trạng thái đồng thời

7.3.13 Sự phân cấp trạng thái

Một biểu đồ statechart có thể rất phức tạp với nhiều trạng thái state và chuyển tiếp transition. Việc phân cấp trạng thái sẽ giúp biểu diễn những hệ thống phức tạp này bằng các biểu đồ đơn giản.

Ví dụ về phân cấp trạng thái



Hình 7.31: Statechart chưa phân cấp trạng thái

a. Statechart không phân cấp trạng thái

b. Statechart có phân cấp trạng thái

Trên hình là một statechart chưa phân cấp trạng thái, còn hình b là statechart của cùng đối tượng đã sử dụng phân cấp trạng thái.

Trạng thái V là trạng thái tổng hợp của hai trạng thái S và T ($V = S \text{ Or } T$), S và T là hai trạng thái con của V. Với cách này ta đã giảm được 1 transition so với cách ở hình a.

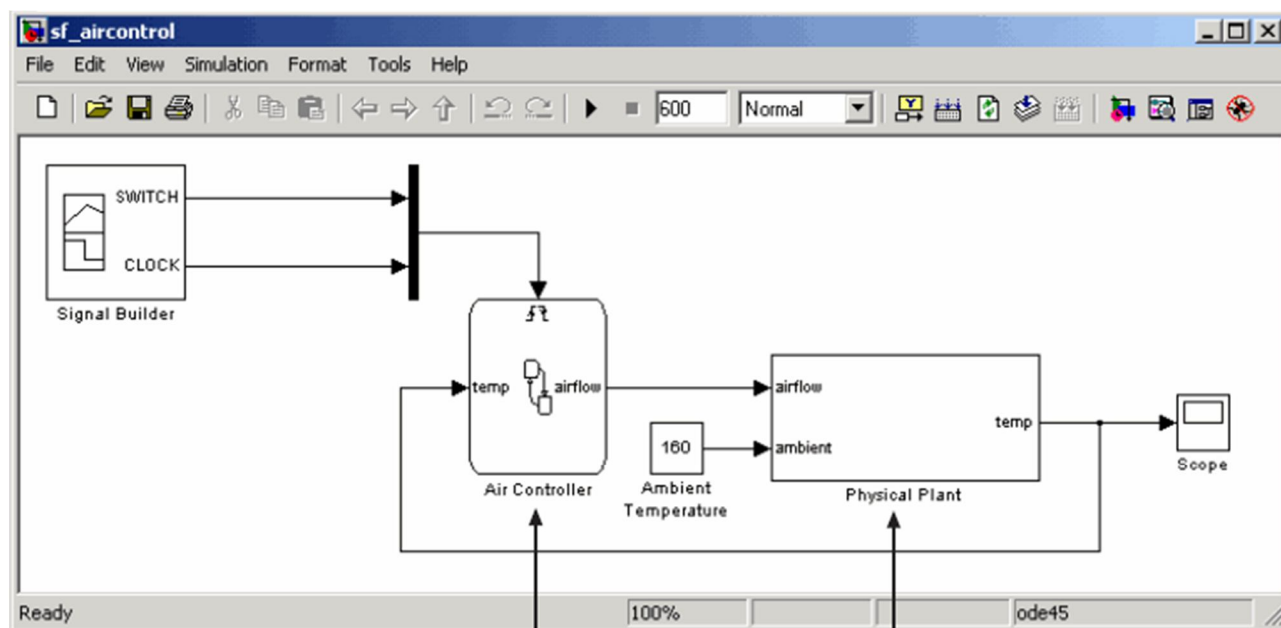
7.3.14 Công cụ stateflow

Hiện nay có khá nhiều công cụ để sử dụng lý thuyết statechart để mô phỏng kiểm tra các hệ thống. Một trong các công cụ đó là Stateflow của Matlab simulink. Dưới đây là trình tự triển khai một hệ thống sử dụng stateflow.

Yêu cầu hệ thống:

Giữ cho đối tượng ở nhiệt độ thấp hơn 120°C bằng cách sử dụng 2 quạt làm mát. Nếu nhiệt độ nhỏ hơn 120°C thì cả hai quạt không hoạt động. Nếu nhiệt độ từ $120-150^{\circ}\text{C}$ thì một quạt hoạt động, nếu nhiệt độ trên 150°C thì hai quạt cùng hoạt động.

Mô hình Simulink như sau:



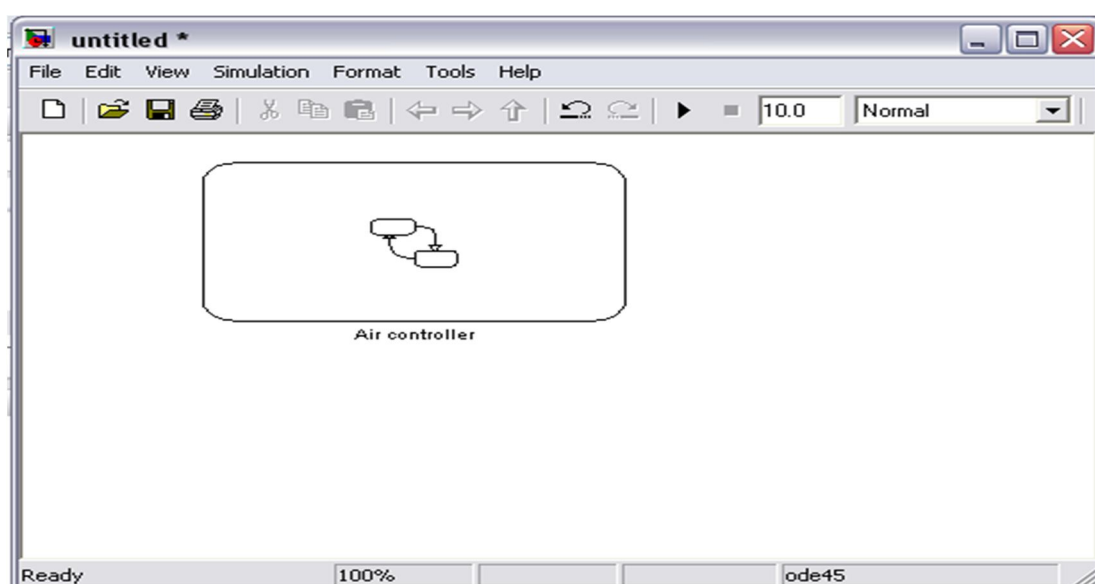
Stateflow chart Simulink subsystem
 Hình 7.32: Mô hình simulink hệ thống làm mát

Trong đó:

Khối Air Controller hệ thống điều khiển nhiệt độ mô phỏng bằng statechart
 Khối Physical Plant là đối tượng cần làm mát, các thông số là: đầu vào nhiệt độ môi trường, và lưu lượng khí của hệ thống quạt. Đầu ra là nhiệt độ của đối tượng.

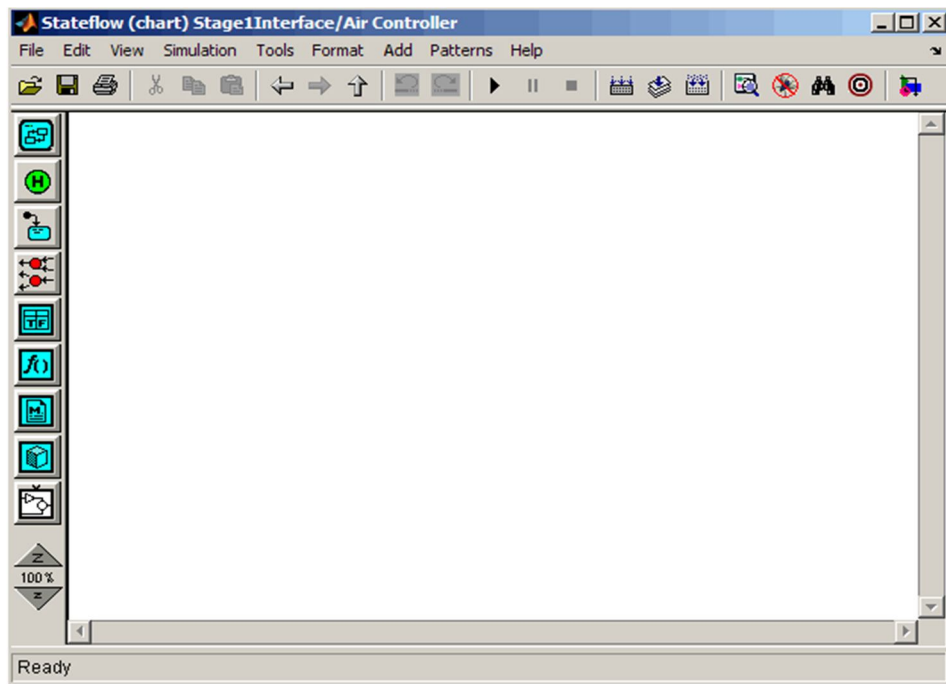
Cách tạo khối Air Controller như sau:

Bước 1 : Mở chương trình Matlab , chọn phần Start -> Simulink -> More -> Stateflow -> New chart. Đặt tên khối là Air controller



Hình 7.33: Mô hình ban đầu

Nháy chuột vào khối Air controller để mở cửa sổ Stateflow editor (chart) :

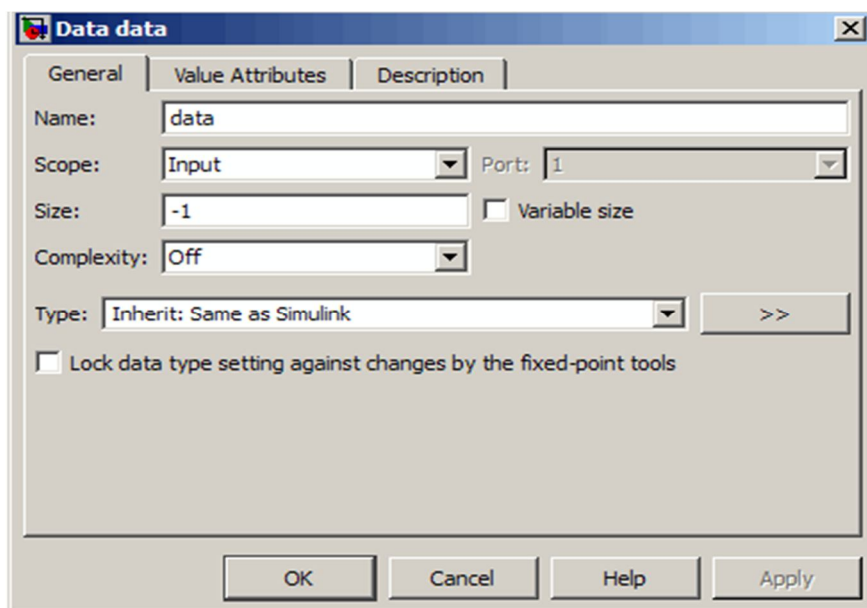


Hình 7.34: Cửa sổ Stateflow (chart)

Gán dữ liệu đầu vào (input) và đầu ra (output) cho khối Air controller

+ **Nhập giá trị Input** : Vào phần Add -> data -> Input from simulink

Cửa sổ Data sẽ hiện ra như hình dưới



Hình 7.35: Cửa sổ Input data

Ở phần Name, chuyển thành Temp do dữ liệu chúng ta cần nhập vào là nhiệt độ môi trường .

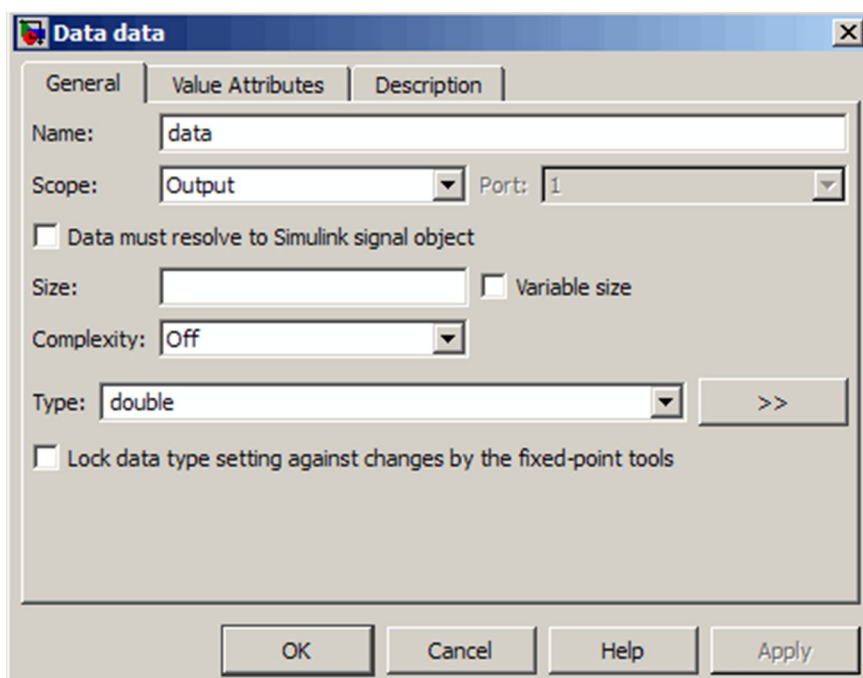
Scope để Input để nhận giá trị bên ngoài Simulink (khối Physical Plant)

Size, Type để nguyên do đã thỏa mãn điều kiện thừa kế tín hiệu đầu vào từ simulink. Complexity để OFF do ta chỉ xét đến các giá trị dạng số đơn giản .

Ở phần Value Attribute , chọn watch in debugger để kiểm tra giá trị của temp ở mỗi điểm dừng (breakpoint) .

+ **Nhập giá trị Output:** Vào Add -> data -> Out to simulink

Cửa sổ Data sẽ hiện ra như hình dưới





Hình 7.36: Cửa sổ Output data

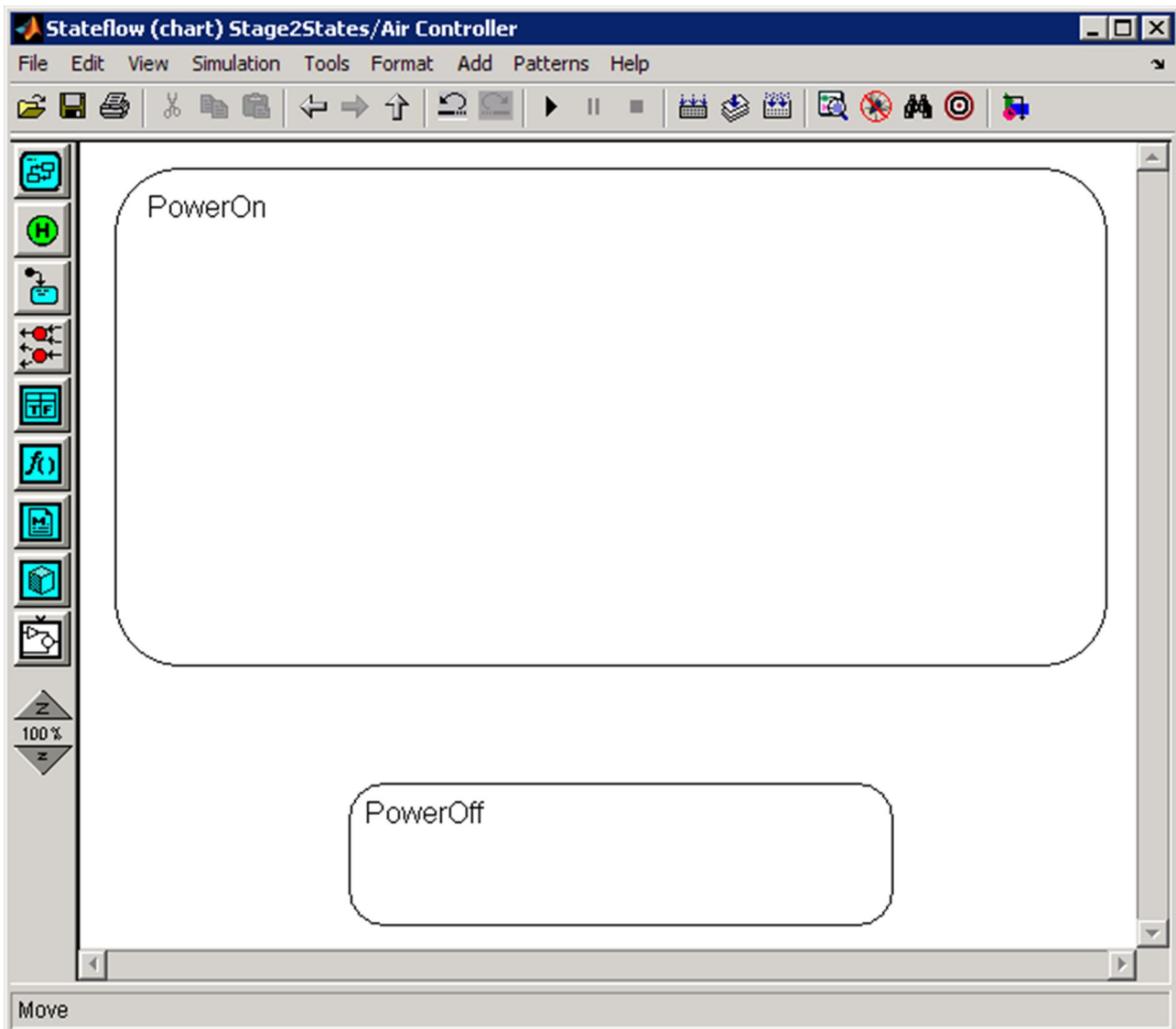
Ở phần Name , ta sửa lại thành airflow .

Ở phần Type , chuyển thành kiểu dữ liệu 8-bit (uint8)

Vào phần Value Attributes , chọn Watch in debugger và điền hai giá trị 0 và 2 vào phần minimum và maximum của giá trị Limit range .

Bước 2: Tạo các khối trạng thái: Ở bảng stateflow editor, chọn state tool icon  và tạo ra 2 khối trạng thái (state) ở phần bảng vẽ, 1 ở trên và 1 ở góc dưới bảng vẽ. Đặt tên cho khối trên là PowerOn và khối dưới là PowerOff. Nhấp chuột vào hình, rồi kéo

chuột ra góc của hình cho đến khi hình  xuất hiện, nhấp và giữ để kéo dẫn hình cho đến độ lớn cần thiết. Biểu đồ lúc này sẽ có dạng như sau :



Hình7.37: Các khối PowerOn và PowerOff

(Lưu ý: khi tạo mới 1 khối state, góc trái của khối là nơi chúng ta đặt tên khối đó)

2 khối trên biểu thị 2 khối trạng thái cơ bản nhất của hệ thống: Bật hệ thống (PowerOn) và tắt hệ thống (PowerOff). Ngoài 2 trạng thái trên, chúng ta cần phải xác định cụ thể xem có những trạng thái hoạt động nào khác liên quan đến quá trình mà chúng ta đang nghiên cứu.

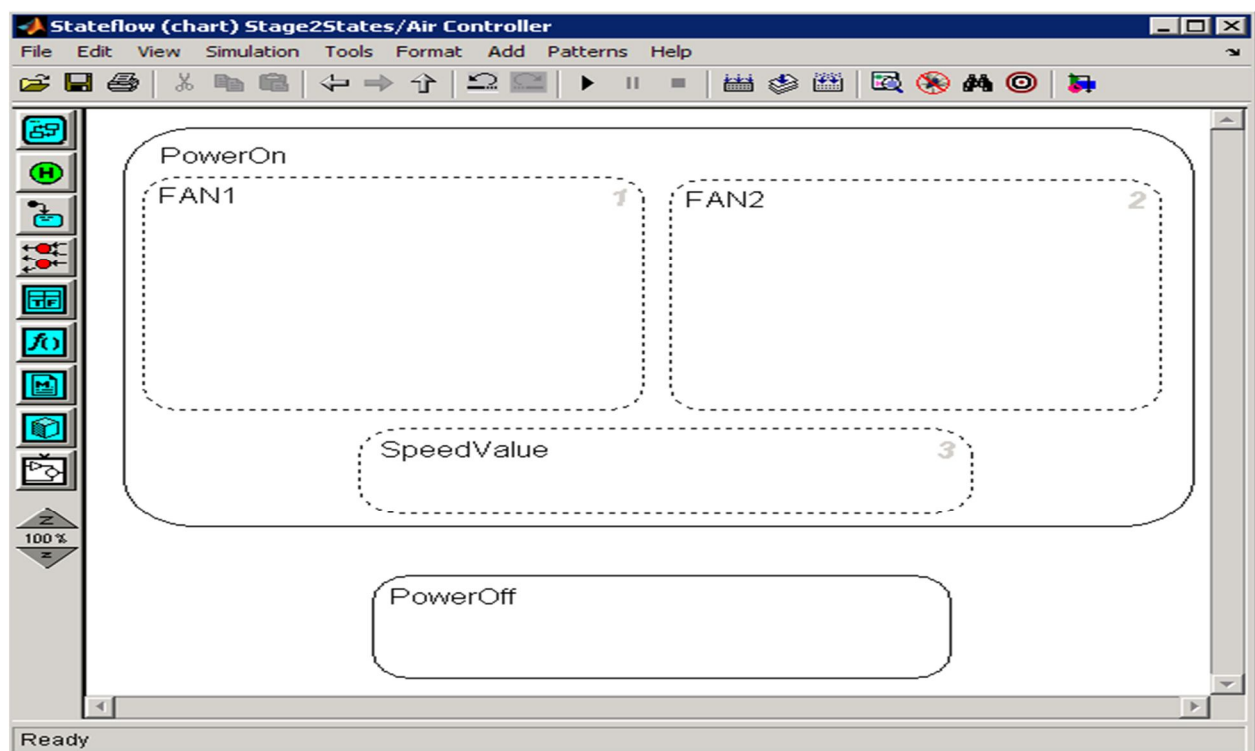
Dựa theo chế độ làm việc , ta có những trạng thái sau :

- + PowerOff : Tắt hệ thống, quạt không hoạt động
- + PowerOn : Bật hệ thống, có thể có 1 , 2 hoặc không có quạt nào hoạt động
- + Trạng thái PowerOff và PowerOn là các trạng thái hoặc)

- + Fan1 , Fan2 : chỉ quạt số 1 và số 2 , là trạng thái phụ thuộc vào trạng thái PowerOn , có thể diễn ra đồng thời với nhau(trạng thái And).
- + Fan1.on, Fan1.off: phụ thuộc vào 2 trạng thái trên nó là Fan1 và PowerOn, chỉ có 1 trạng thái diễn ra tại 1 thời điểm xác định (trạng thái OR)
- + Fan2.on, Fan2.off: phụ thuộc vào 2 trạng thái trên nó là Fan2 và PowerOn, chỉ có 1 trạng thái diễn ra tại 1 thời điểm xác định (trạng thái OR)
- + SpeedValue: là trạng thái quan sát dựa trên các giá trị được đưa ra của Fan1 và Fan2 để đưa ra tín hiệu đầu ra chính xác nhất .

Sau khi đã xác định được các trạng thái hoạt động của quá trình, ta tiếp tục tiến hành việc xây dựng các khối trạng thái còn lại.

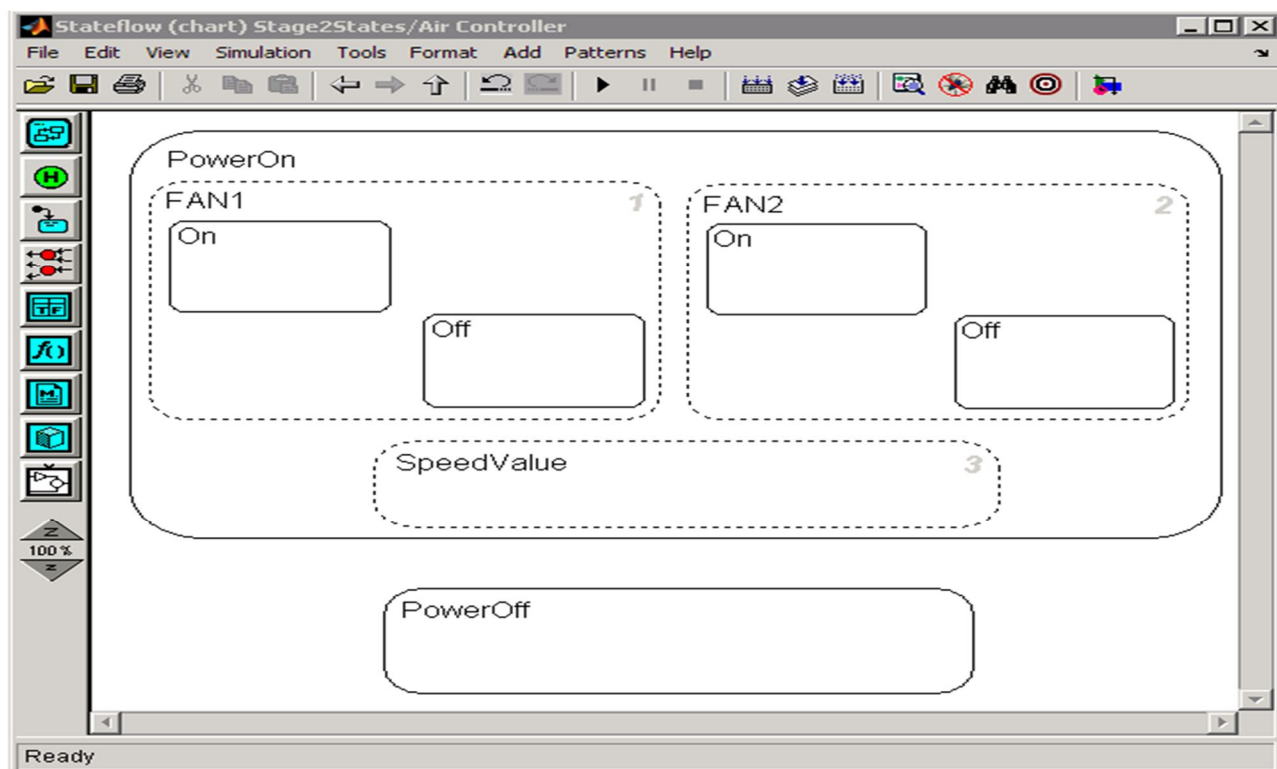
- + Chuột phải vào khối PowerOn , chọn **Decomposition => Parallel (AND)** => những state được tạo bên trong PowerOn đều là các Parallel State .
- + Tiếp tục chọn state icon tool và tạo mới thêm 3 state bên trong , lần lượt là Fan1 , Fan2 , SpeedValue .



Hình 7.38: Thêm vào các khối Fan1, Fan2, SpeedValue

Ta cần chú ý là các khối song song (Parallel) khi được tạo ra đều được đánh số dựa theo thứ tự được tạo. Vì thế để đảm bảo đúng thứ tự hoạt động mà chúng ta đã đề ra từ trước, vào **File => Chart Properties**, chọn phần **User specified state/transition execution order**. Lựa chọn này cho phép ta được tự ý chỉnh sửa thứ tự hoạt động của các state song song theo ý muốn. Để làm việc này, phải chuột vào 1 state song song bất kỳ, chọn **Execution order** và chọn thứ tự hoạt động mà mình muốn .

- + Để tạo các trạng thái On , Off của mỗi quạt. Chọn state icon tool và tạo thêm bên trong khối Fan1 và Fan2, mỗi khối 2 trạng thái On và Off .



Hình 7.39: Tạo các khối trạng thái On, Off cho Fan1 và Fan2

Bước 3 : Gán hành động cho trạng thái:

Chúng ta có 3 loại hành động, đó là:

Entry (en): diễn ra vào lúc bắt đầu trạng thái, sử dụng 1 lần duy nhất. Có dạng:

Entry: một hoặc nhiều hành động, hay

en : một hay nhiều hành động;

During (du): diễn ra khi trạng thái được kích hoạt, và không có bất kì tín hiệu truyền xác định (valid transition) nào tới những trạng thái khác, được dùng đến trong mỗi bước thời gian.

Exit: diễn ra trước khi chuyển sang một trạng thái khác , chỉ cần sử dụng 1 lần .

Theo như trên thì trong biểu đồ này , chỉ có trạng thái PowerOff và SpeedValue là cần phải gán hành động vào.

Cách thực hiện:

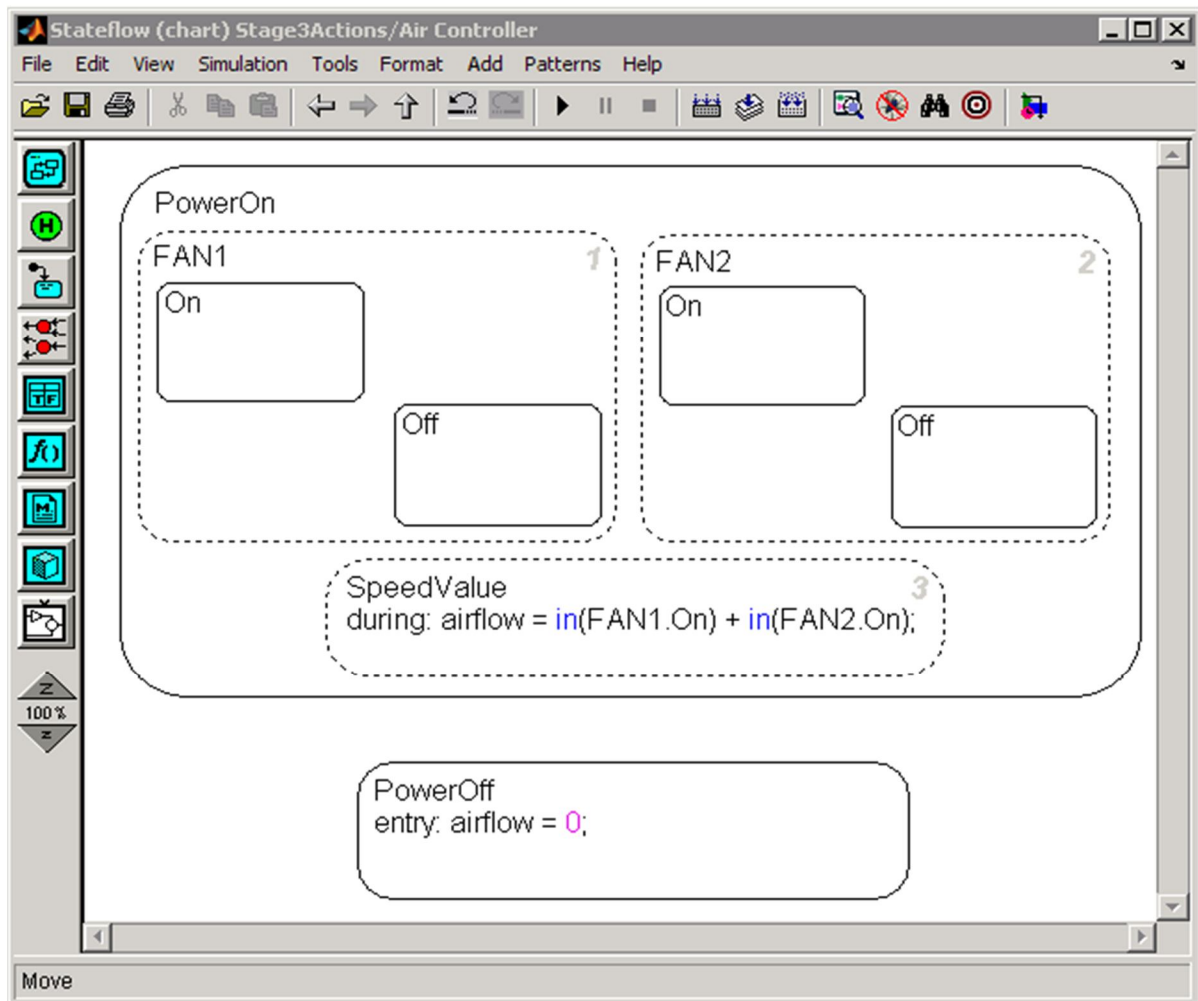
Với khối PowerOff, bấm vào phần tên của khối, xuống dòng (bấm Enter) và đánh dòng chữ:

entry: airflow = 0 ;

Với khối SpeedValue, bấm vào phần tên của khối, xuống dòng và đánh dòng chữ:

during: airflow = in(FAN1.On) + in(FAN2.On);

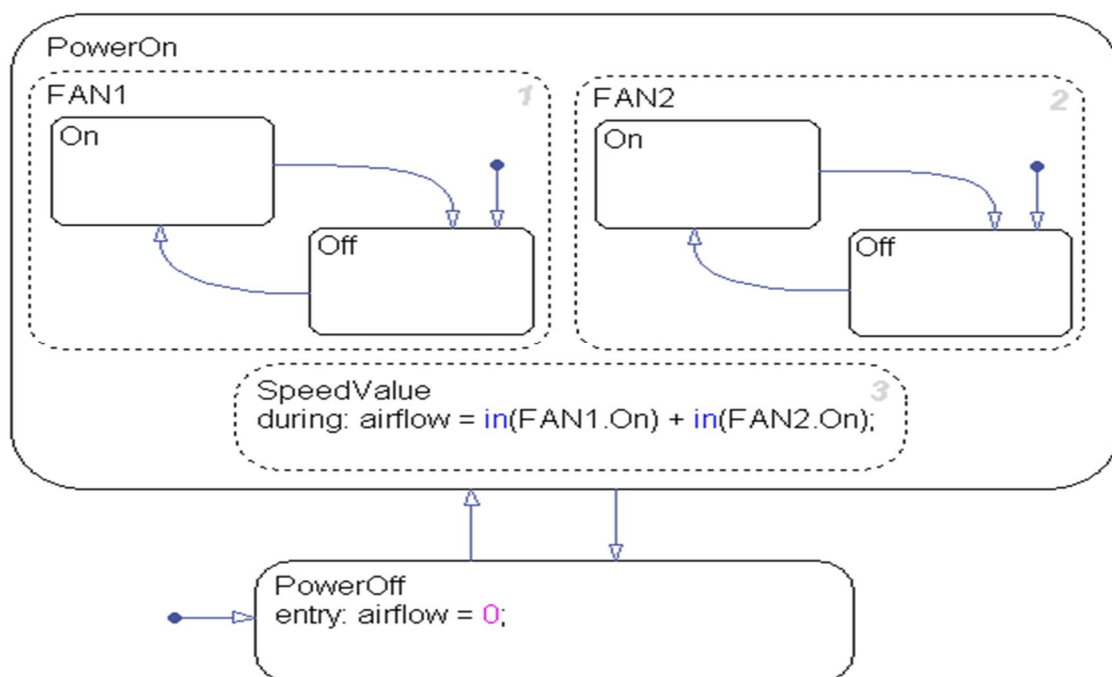
Kết quả cuối cùng sẽ giống như sau :



Hình 7.40: Viết các hành động .

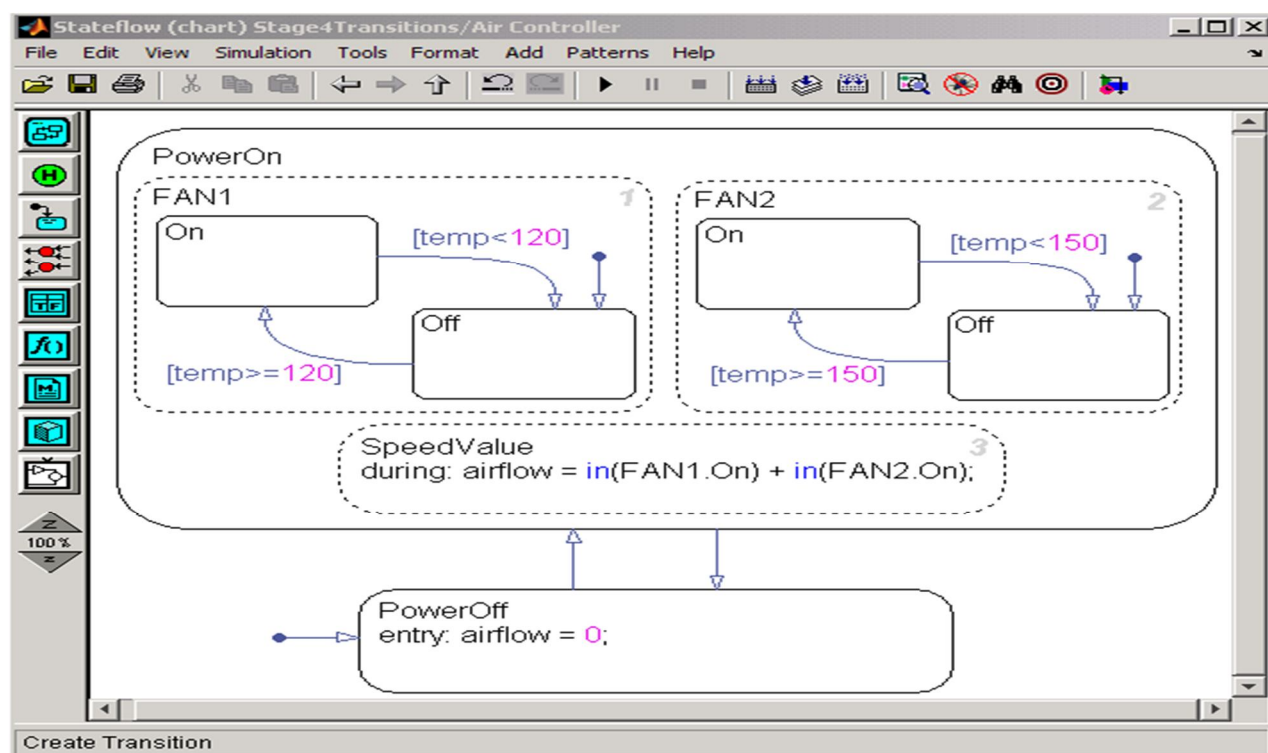
Bước 4: Kết nối các khối và bổ sung các điều kiện cần thiết .

+ Bổ sung các Transition ta được:



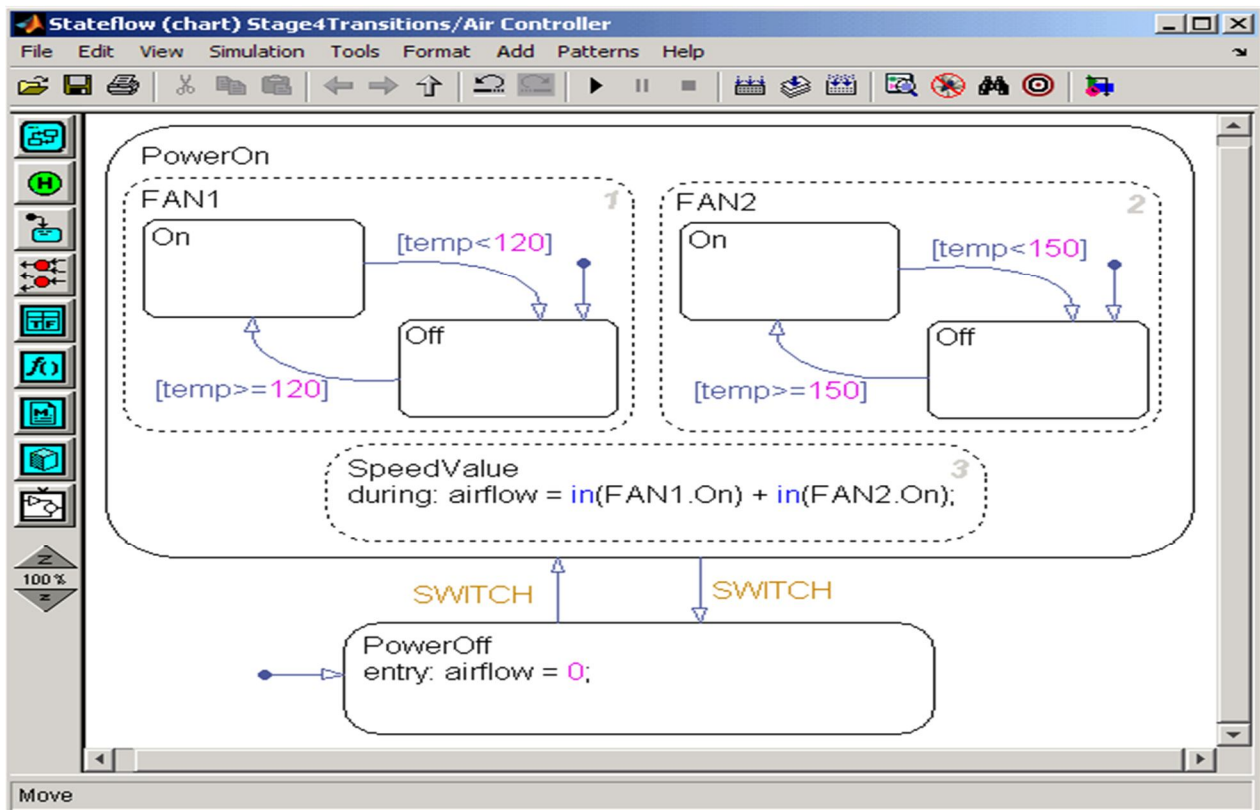
Hình 7.41: Thêm các transition.

+ Bổ sung các Condition:



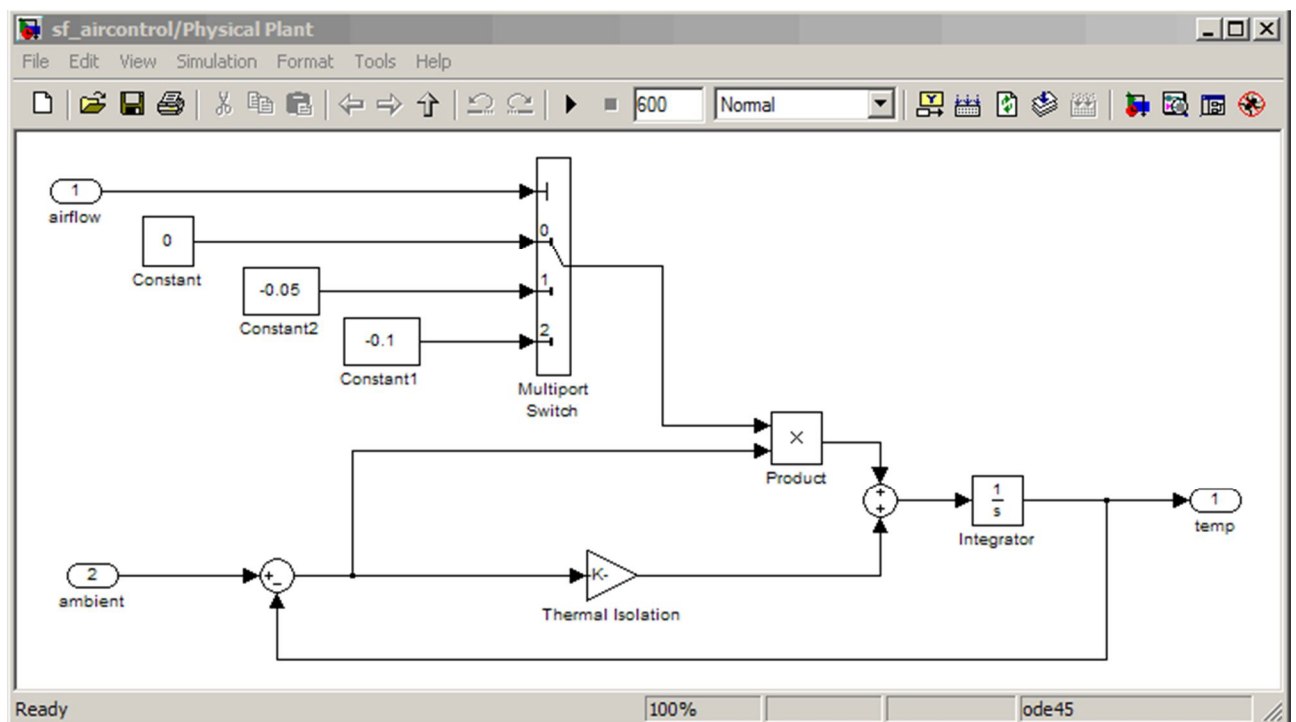
Hình 7.42: Đặt các Condition.

+ Đặt các Trigger



Hình 7.43: Thêm các Trigger

+ Khối Physical Plant



Hình 7.44: Khối Physical Plant