THIẾT KẾ VÀ CHẾ TẠO HỆ THỐNG ĐIỀU KHIỂN TỪ XA QUA MẠNG PSTN

1. Tính cấp thiết của đề tài

Ngày nay, cuộc sống của con người ngày càng hiện đại, để có thể giải phóng một phần sức lao động và tiết kiệm thời gian con người không ngừng cải tiến, phát minh mới các ứng dụng KHKT. Một trong số các ứng dụng KHKT đó là hệ thống điều khiển từ xa. Hệ thống này không những được sử dụng trong an ninh quốc phòng, các dự án nghiên cứu khoa học, trong sản xuất mà con phục vụ cả nhu cầu của mọi người dân lao động.

Đối với một số hệ thống điều khiển từ xa bằng tia hồng ngoại, bằng cơ, bằng từ... có một điểm hạn chế đó là khoảng các điều khiển. Ngược lại với mạng điện thoại ngày càng được mở rộng trên toàn thế giới thì giới hạn xa không phụ thuộc vào khoảng cách, do đó đã mở ra một hướng mới trong lĩnh vực tự động điều khiển.

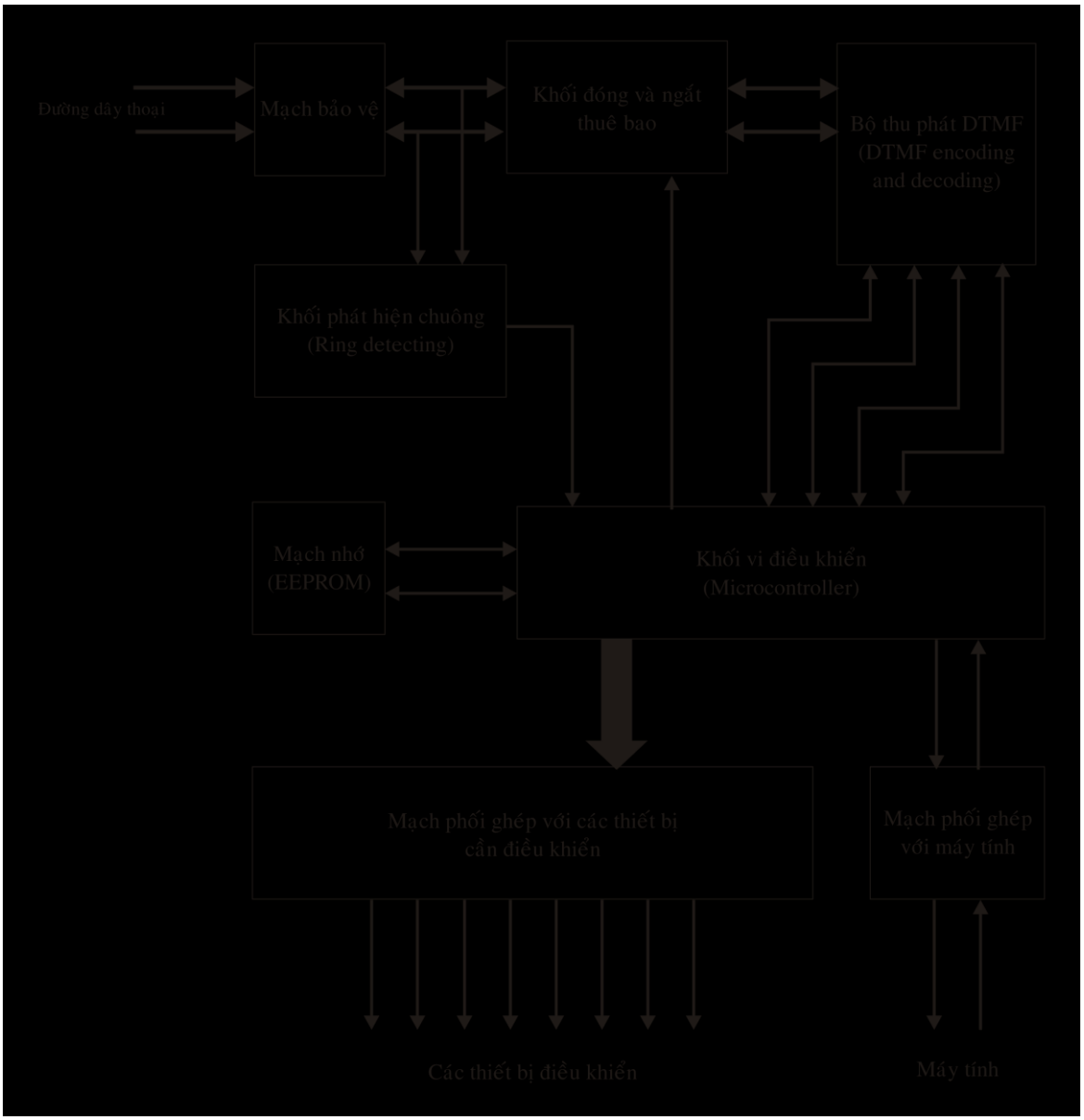
Với điện thoại đang trở nên phổ biến trong hầu hết các hộ gia đình. Cùng với khả năng truyền tải thông tin dạng tiếng nói, mạng điện thoại còn cho phép quay số đa âm tần, là cơ sở cho khả năng điều khiển từ xa.

Trên thế giới cũng đã có nhiều công trình nghiên cứu và ứng dụng chế tạo các hệ thống điều khiển từ xa qua mạng điện thoại có thể nói là rất hoàn thiện, tuy nhiên các sản phẩm này mới chỉ được ứng dụng ở những công trình lớn, với giá thành cao như trong các công trình quân sự, các thiết kế "ngôi nhà thông minh",…Tại Việt Nam cũng đã có một số công trình nghiên cứu, song vẫn còn có khiết điểm như sử dụng biến áp để ghép nối...do vậy sản phẩm chưa phổ biến tới tay người tiêu dùng.

Với mục đích tìm hiểu và học tập nhóm sinh viên chúng tôi mạnh dạn đăng ký thực hiện đề tài "Thiết kế và chế tạo hệ thống điều khiển từ xa qua mạng PSTN". Để tạo ra sản phẩm phổ biến tới tay người tiêu dùng với giá thành thấp, an toàn tin cậy, dễ sử dụng.

2. Nội dung báo cáo: Mô tả sơ đồ khối,nguyên tắc hoạt động và kết quả thu được.

a. Sơ đồ khối của hệ thống



Hình 1: Sơ đồ khối của hệ thống điều khiển từ xa qua mạng PSTN

b. Nguyên tắc hoạt động

Khi muốn điều khiển ta chỉ việc gọi về số máy của máy điện thoại được kết nối với bộ phận điều khiển ở nơi cần điều khiển thì tín hiệu chuông của tổng đài sẽ cấp cho thuê bao nếu thuê bao đó không bận. Mạch điều khiển được mắc song song vào đường dây của thuê bao. Lúc này, khối phát hiện chuông sẽ phát hiện tín hiệu này và ngõ ra thay đổi mức logic từ cao xuống thấp. Sự thay đổi mức logic này tác động vào khối xử lý trung tâm. Khối xử lý

sẽ định thời gian đợi chuông. Sau một khoảng thời gian không ai nhấc máy tức vẫn còn tín hiệu chuông thì khối xử lý sẽ tác động vào khối kết nối thuê bao. Khối kết nối thuê bao sẽ đóng tải giả, lúc này tổng đài ngưng cấp tín hiệu chuông và kết nối cho thông thoại.

Khi đã thông thoại, ta sẽ bấm mã mật khẩu để thâm nhập vào hệ thống điều khiển, hệ thống sẽ báo lại bằng tiếng bip, bip... để báo cho người điều khiển biết mạch đã làm việc và chờ lệnh điều khiển. Sau đó người điều khiển sẽ bấm lệnh điều khiển mở hay tắt, tín hiệu này tác động đến khối đóng ngắt relay.

Việc nhận dạng phím nào bấm, được khối giải mã DTMF quyết định. Khi người điều khiển nhấn phím, một cặp tần số DTMF truyền trên đường dây thoại. Tần số này nằm trên dải thông của tín thiệu thoại, một tần số cao và một tần số thấp nên không thể trùng lấp với tín hiệu người nói. Khi giải mã DTMF và hiển thị số được nhấn, 4 bit được giải mã được đưa vào khối xử lý trung tâm để xử lý.

Khi không ấn phím, sau một thời gian đợi mà không có phím ấn thì khối xử lý sẽ ngưng kết nối thuê bao. Lúc này tổng đài sẽ giải tỏa thuê bao. Người điều khiển có thể gác máy bất cứ lúc nào muốn ngừng điều khiển, mạch sẽ tự động ngắt kết nối thuê bao sau một thời gian nhất định để giải tỏa thuê bao.

c. Kết quả thu được

- Mạch điện thiết kế và hoàn thành đúng tiến độ với giá thành hợp lý.
- Mạch điện hoạt động tốt có thể điều khiển hoạt động của 2 thiết bị điện khác nhau cùng một lúc.
- Có thể nâng số lượng thiết bị điều khiển lên tối đa là 8 thiết bị.
- Mạch điện dễ sử dụng: có thể điều khiển thiết bị trực tiếp tại nhà hoặc bất kỳ nơi nào bằng điện thoại để bàn hoặc điện thoại di động.
- Chương trình cho phép điều khiển cả từ trên máy vi tính với thao tác cài đặt đơn giản.
- Giao diện thân thiện dễ sử dụng, có thông tin trợ giúp, hoàn toàn bằng tiếng Việt.

3. Phương pháp nghiên cứu

Để thực hiện được đề tài, chúng em cần phải xác định được phương pháp nghiên cứu với trình tự như sau:

a. *Nghiên cứu và tìm hiểu lý thuyết:*

- Nghiên cứu tìm hiểu về DTMF(Dual-Tone Multi-Frequency): DTMF là tập hợp của hai cặp tần số. Một là tần số thấp có giá trị từ 697đến 941Hz, và một là tần số cao từ 1209 đến 1633Hz. Sự kết hợp chính xác này tạo ra các tone 1,2,...,9,0,A,B,C,D và #,*.
- Chức năng, cách sử dụng các linh kiện chính được sử dụng:

- Linh kiện có chức năng thu phát tín hiệu DTMF là IC MT8880C: Đây là linh kiện tích hợp có khả năng giải mã thu và phát và có các ngõ ra tín hiệu BCD. Trong mạch sử dụng cấu hình một lối vào.
- Linh kiện có chức năng điều khiển là IC AT89C52: là linh kiện có khả năng lập trình được. Đây chính là trung tâm điều khiển cả hệ thống.
- Ngoài ra còn có các linh kiện khác như: IC 4N35 dùng để các ly điện áp giữa hệ thống và đường dây, EEPROM 24C16A cho phép chứa đựng mật khẩu và các thiết bị điều khiển và có thể thay đổi được nhằm tạo ra sự mềm dẻo cho hệ thống, IS RS MAX232 dùng để phối ghép với máy tính.

b. *Lập sơ đồ khối theo mục tiêu của đề tài:* Đó là, điều khiển được các thiết bị khi người vắng nhà hoặc có nhà...

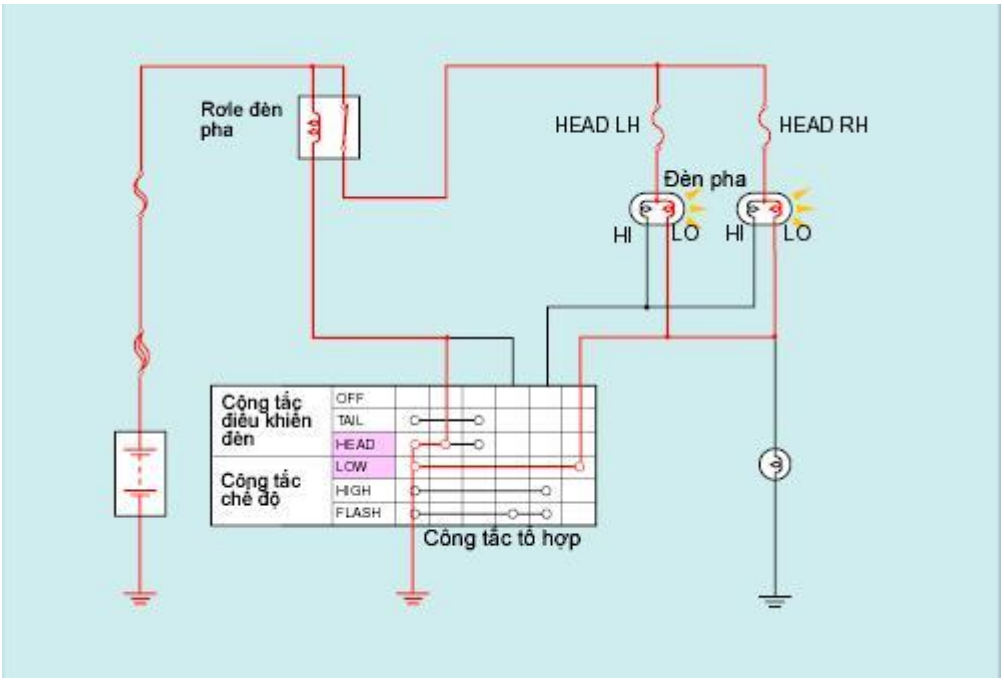
c. *Tính toán thiết kế phần cứng.*

d. *Thiết kế phần mềm cho khối xử lý trung tâm*: Dùng ngôn ngữ lập trình Asembler.

e. *Thiết kế giao diện trên máy vi tính*: Dùng ngôn ngữ lập trình VisualBasic.

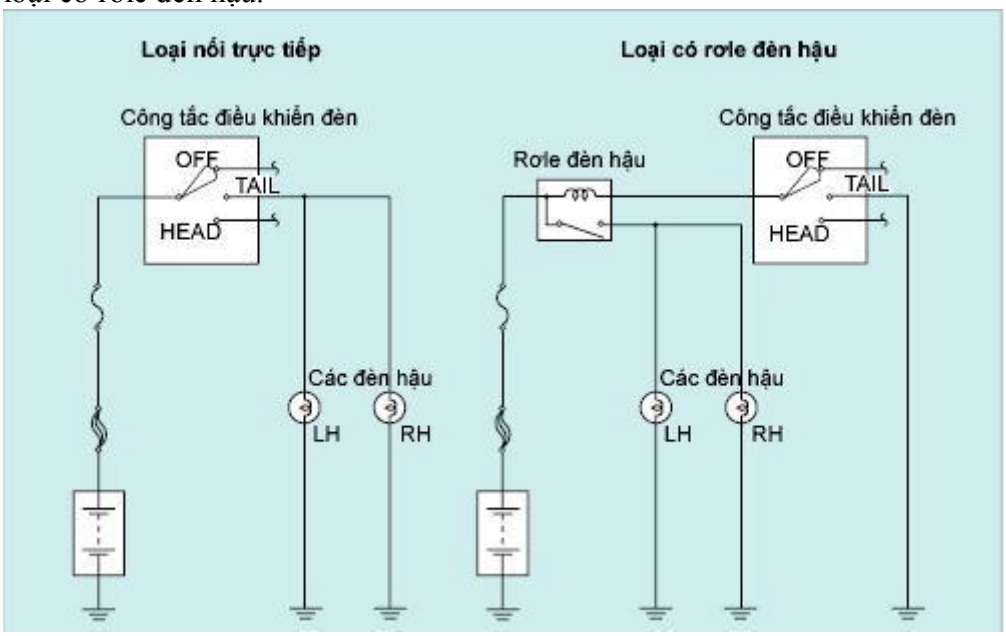
4. Khả năng ứng dụng trong thực tiễn

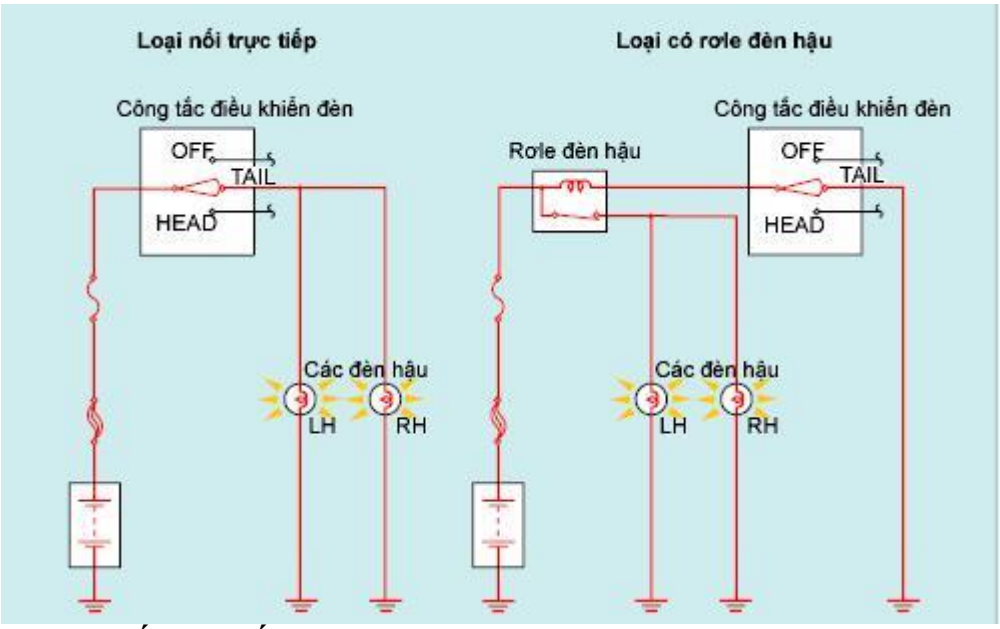
Hệ thống có thể được ứng dụng trong các lĩnh vực khác nhau, nhưng chủ yếu là sản phẩm có thể phục vụ nhu cầu của người lao động với mục đích giải phóng một phần sức lao động và tiết kiệm thời gian.



1. Hệ thống đèn hậu

Có hai loại hệ thống đèn hậu: loại đèn hậu được nối trực tiếp vào công tắc điều khiển đèn và loại có rơle đèn hậu.





(1) Loại nối trực tiếp

Khi công tắc điều khiển đèn được vặn về vị trí "TAIL", thì các đèn hậu bật sáng.

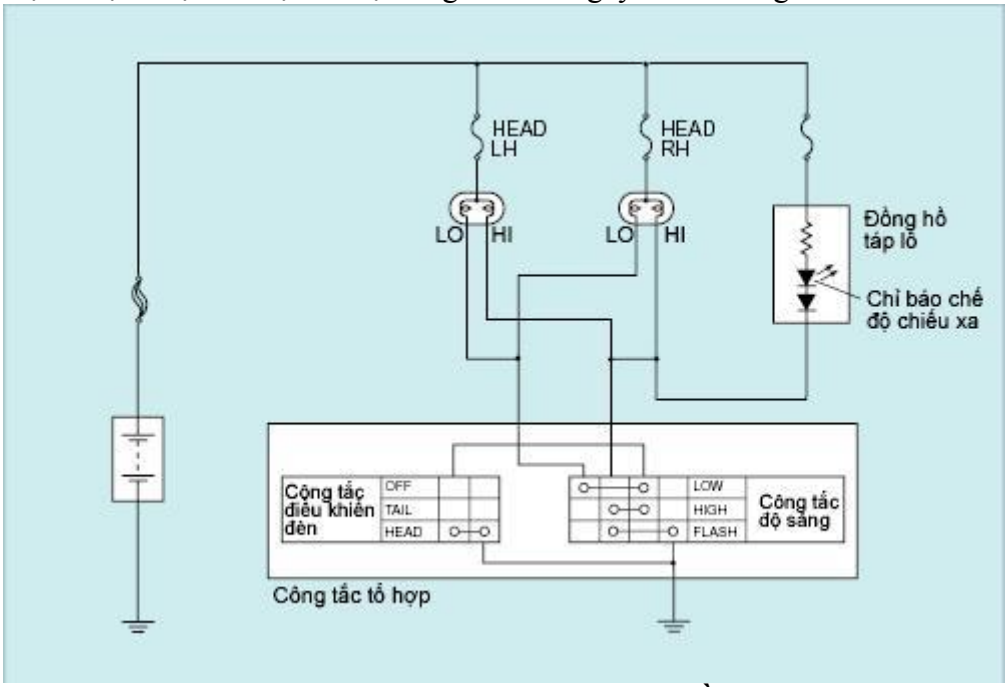
(2) Loại có rơle đèn hậu

Khi công tắc điều khiển đèn vặn về vị trí "TAIL", thì dòng điện đi vào phía cuộn dây của rơ le đèn hậu. Rơle đèn hậu được bật lên và đèn sáng.

Một số xe có hệ thống đèn hậu được trang bị chỉ báo đèn hậu.

2. Hệ thống đèn pha

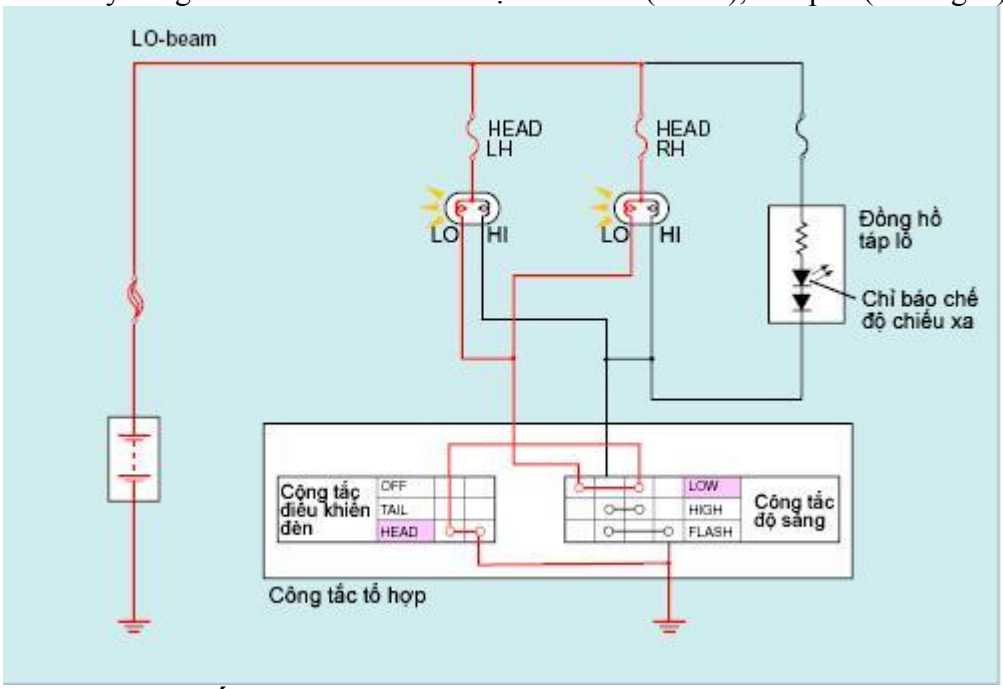
Có hai loại hệ thống đèn pha khác nhau tuỳ theo chúng có thiết bị điện như rơle đèn pha và rơle điều chỉnh độ sáng. Nhìn chung khi công tắc điều chỉnh độ sáng ở vị trí "FLASH", thì mạch điện được cấu tạo để bật sáng các đèn ngay cả khi công tắc điều khiển đèn ở vị trí OFF.



(1) Loại không có rơ le đèn pha và không có rơle điều chỉnh độ sáng

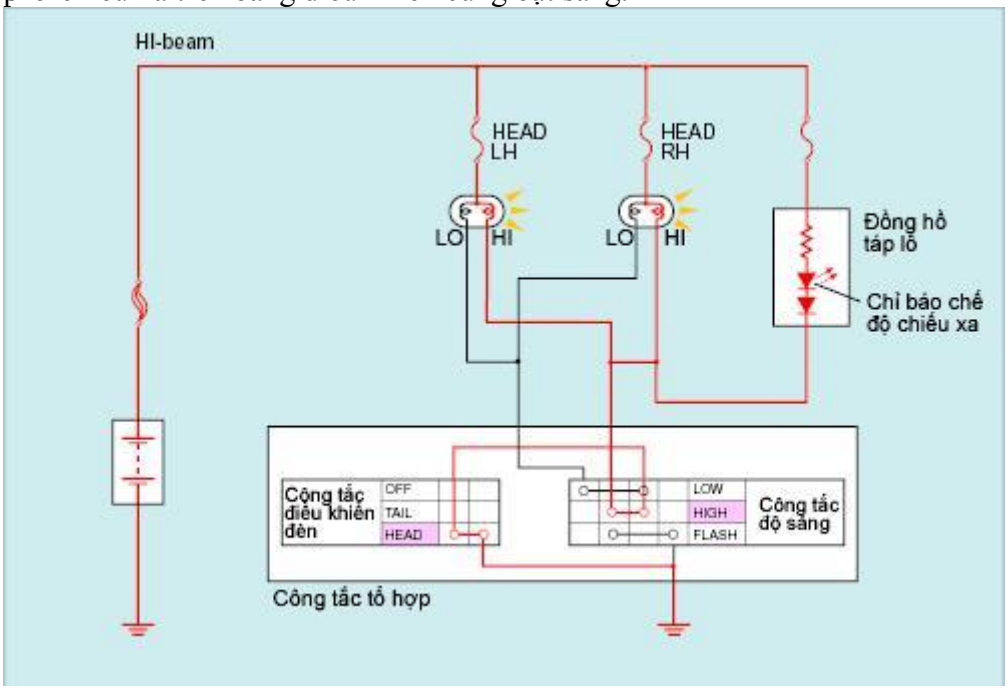
+ Đèn pha (Chiếu gần LO – Bearn)

Khi xoay công tắc điều khiển đèn về vị trí HEAD (LOW), đèn pha (chiếu gần) bật sáng.



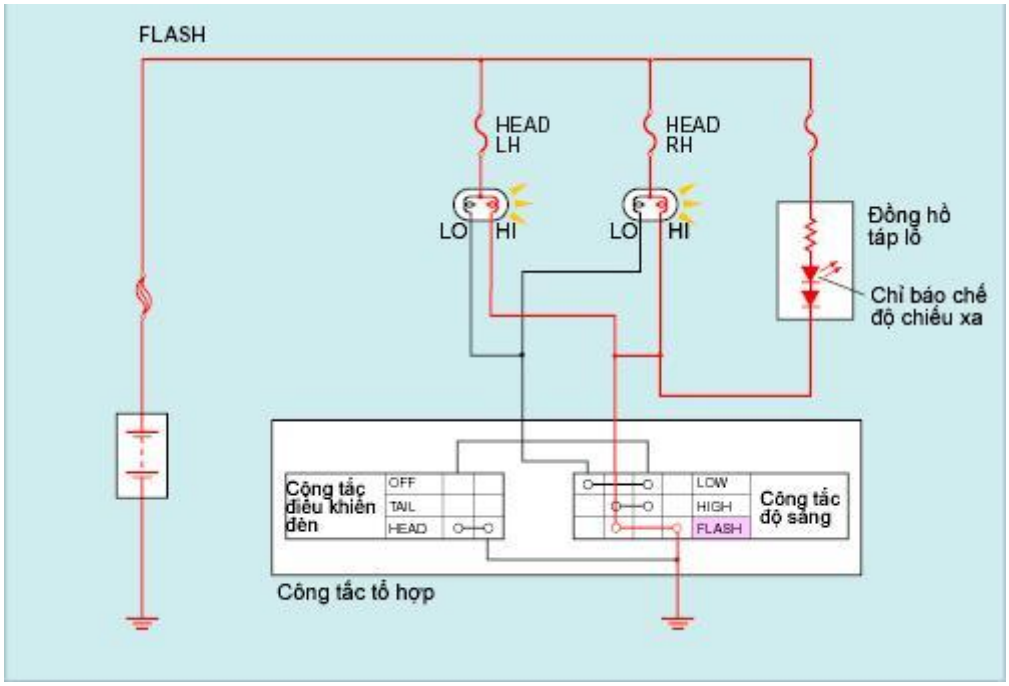
+ Đèn pha (Chiếu xa "High – Bearn")

Khi xoay công tắc về vị trí HEAD (HIGH), thì đèn pha-chiếu xa bật sáng và đèn chỉ báo đèn pha-chiếu xa trên bảng điều khiển cũng bật sáng.

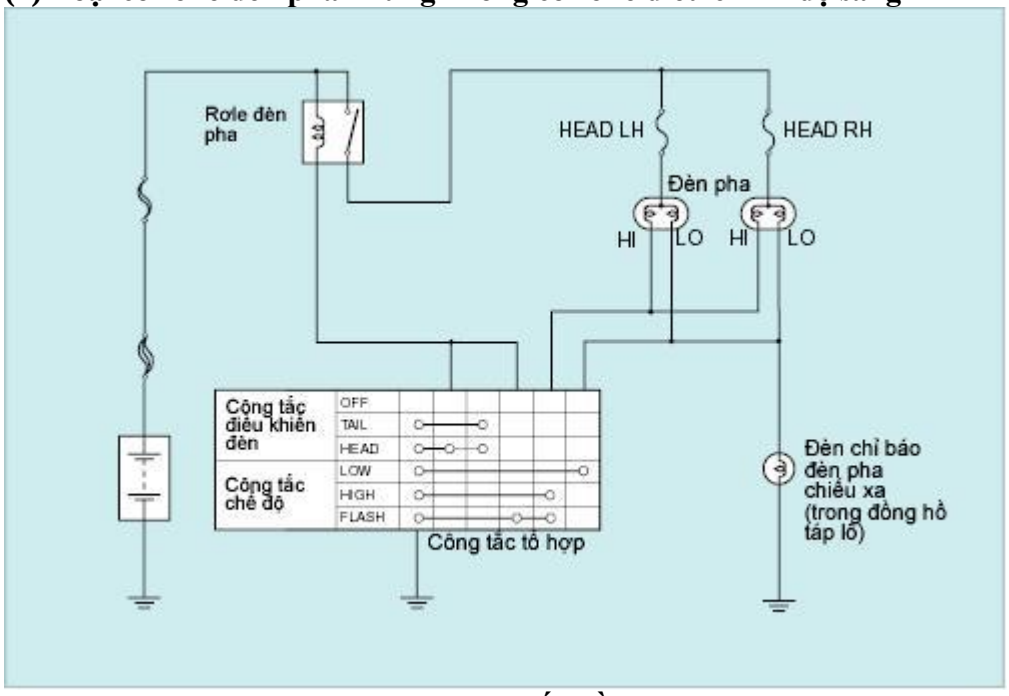


+ Đèn pha FLASH (Nháy pha)

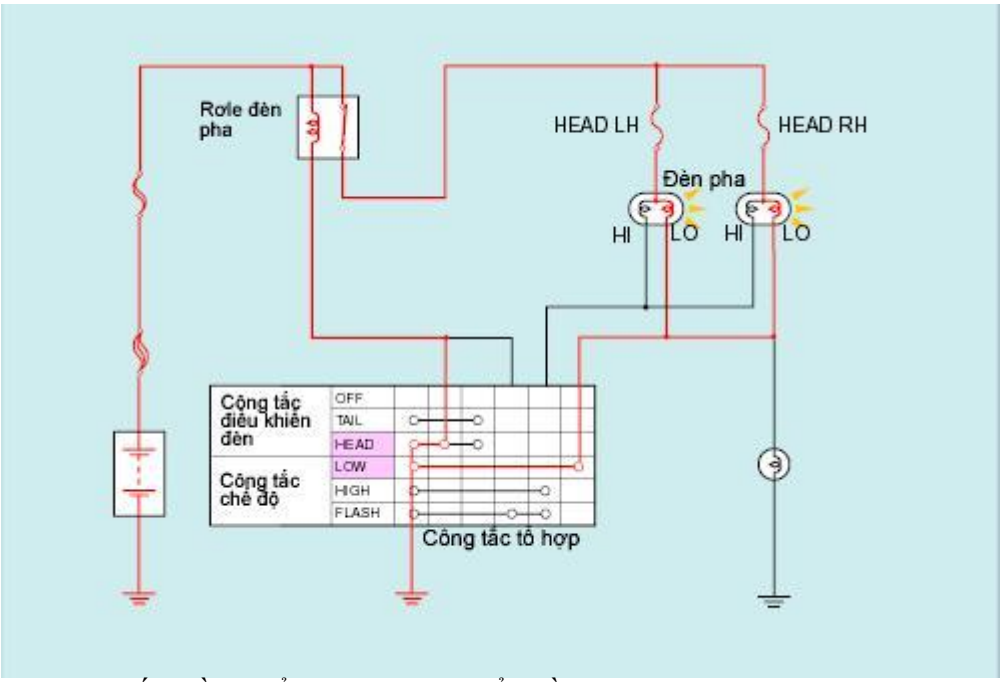
Khi công tắc điều khiển đèn dịch chuyển về vị trí FLASH thì đèn pha chiếu xa sẽ bật sáng.



(2) Loại có rơ le đèn pha nhưng không có rơ le điều chỉnh độ sáng

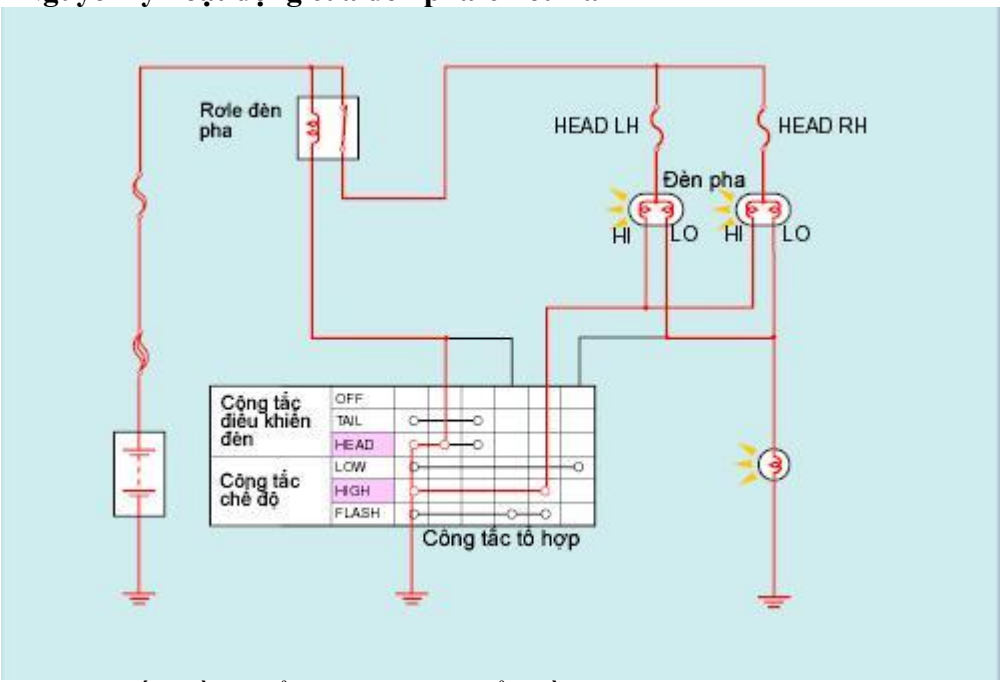


+ Nguyên lý hoạt động của đèn pha-chiếu gần



Khi công tắc điều khiển đèn dịch chuyển về vị trí HEAD (LOW), thì rơle đèn pha được bật lên và đèn pha-chiếu gần sáng lên.

+Nguyên lý hoạt động của đèn pha-chiếu xa



Khi công tắc điều khiển đèn dịch chuyển về vị trí HEAD (HIGH), thì rơ le đèn pha bật đèn pha-chiếu xa và đèn chỉ báo đèn pha-chiếu xa trên đồng hồ táp lô cũng bật sáng.

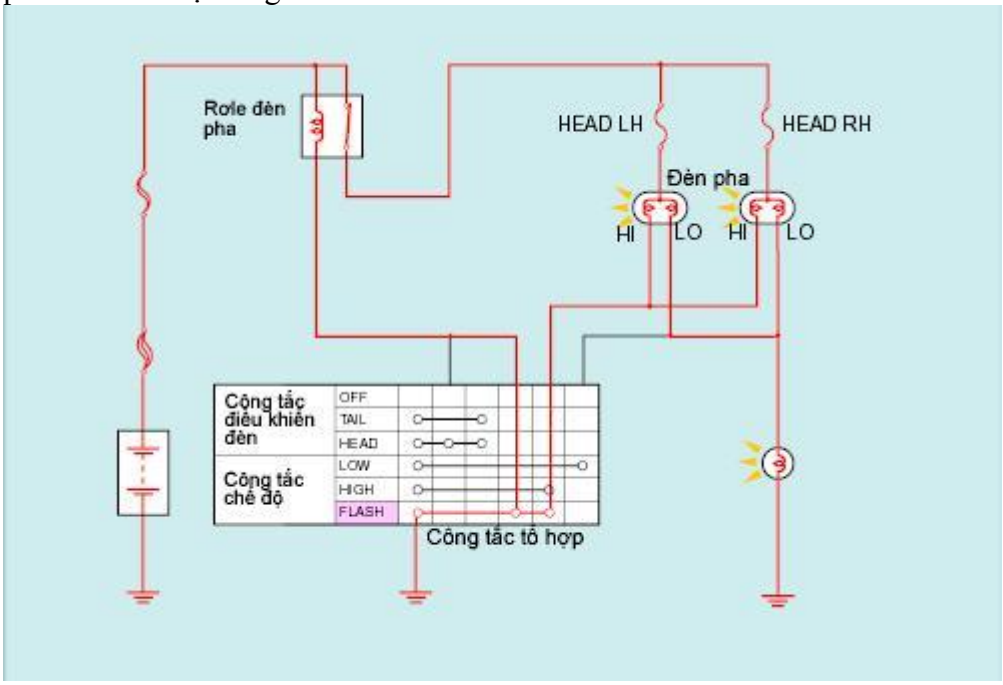
Loại mắc nối tiếp với đèn chỉ báo

Dòng điện đi từ đèn pha-chiếu gần đến đèn chỉ báo đèn pha-chiếu xa và đèn chỉ báo bật sáng. Dòng điện đi đến đèn pha-chiếu gần, nhưng vì điện trở và dòng điện nhỏ nên chúng không sáng.

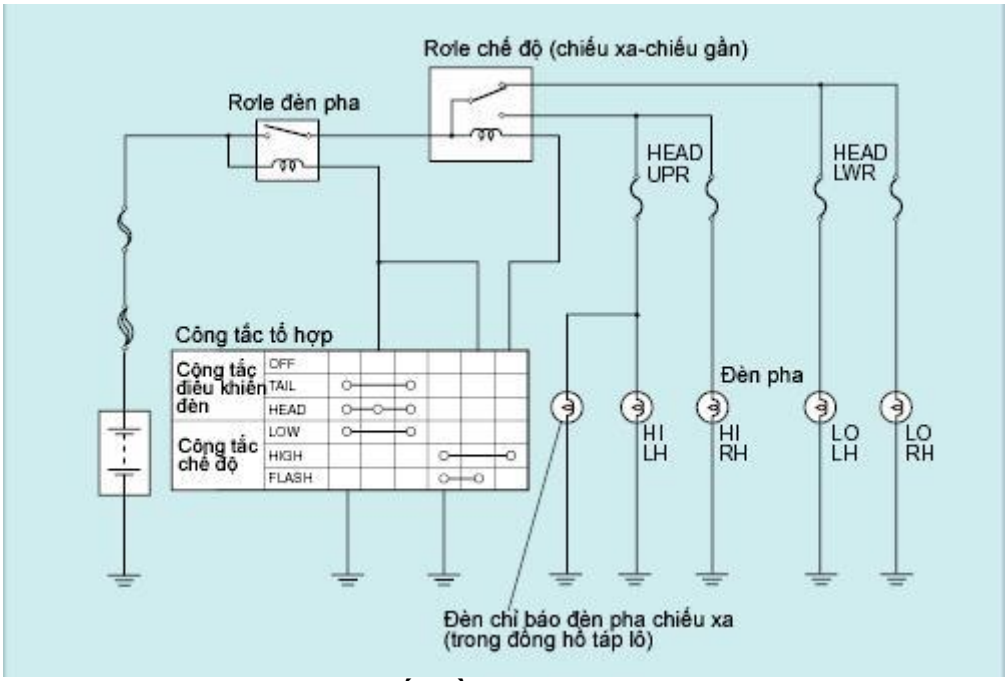
+ Hoạt động nháy đèn pha

Khi công tắc điều khiển đèn dịch chuyển về vị trí FLASH, thì rơ le đèn pha bật lên và các đèn

pha-chiếu xa bật sáng.

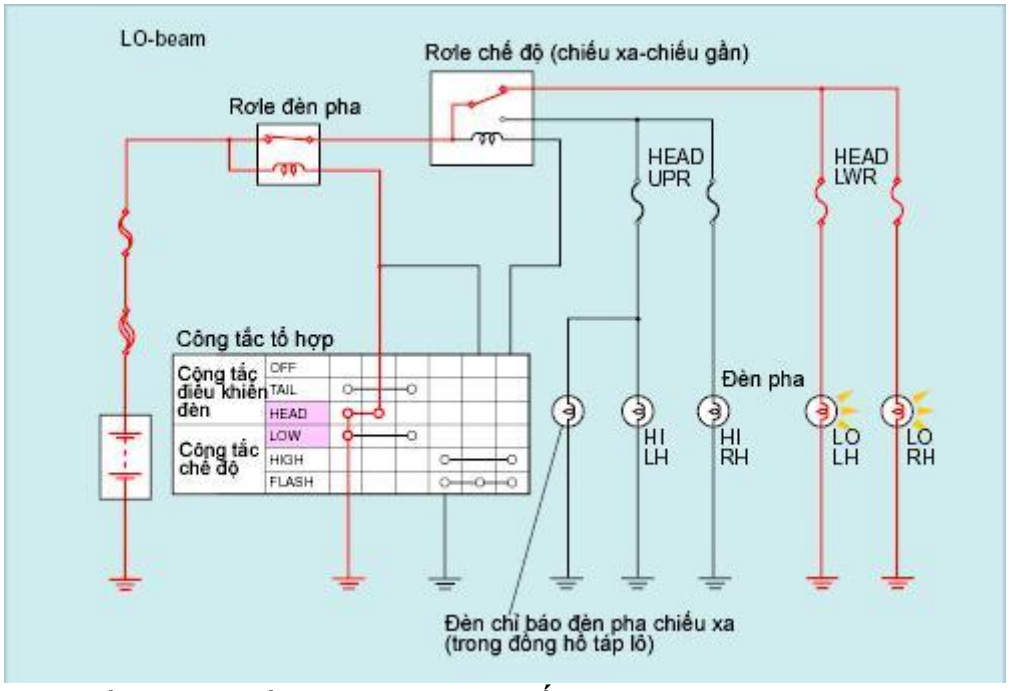


(3) Loại có cả rơle đèn pha và rơle điều chỉnh độ sáng



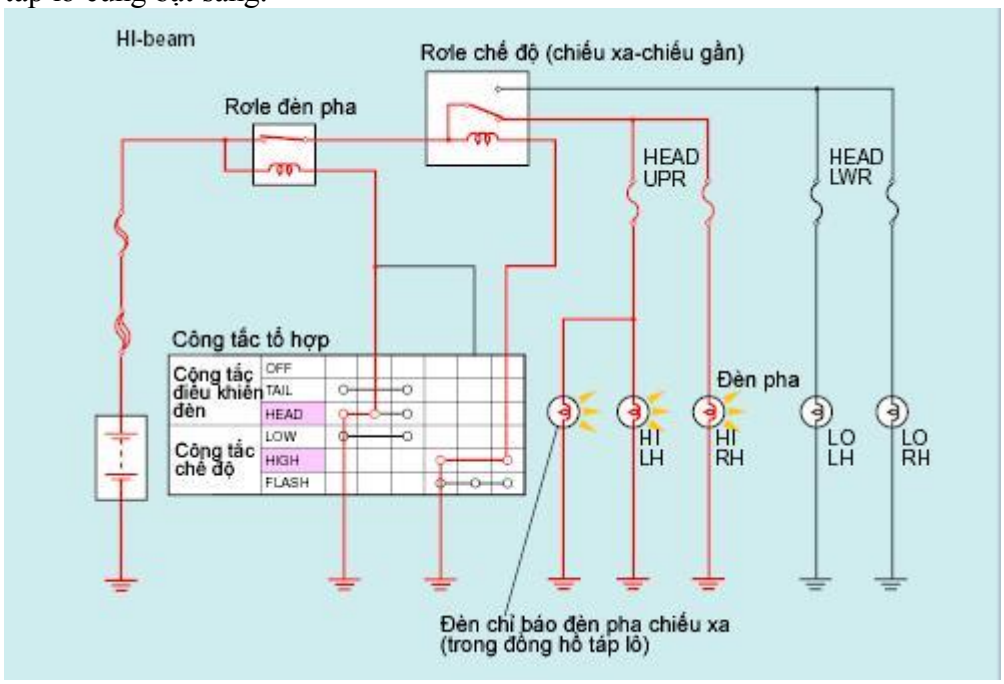
+ Hoạt động của đèn pha-chiếu gần

Khi công tắc điều khiển đèn dịch chuyển về vị trí HEAD (LOW), thì rơ le đèn pha bật lên và các đèn pha-chiếu gần bật sáng.



+ Nguyên lý hoạt động của đèn pha-chiếu xa

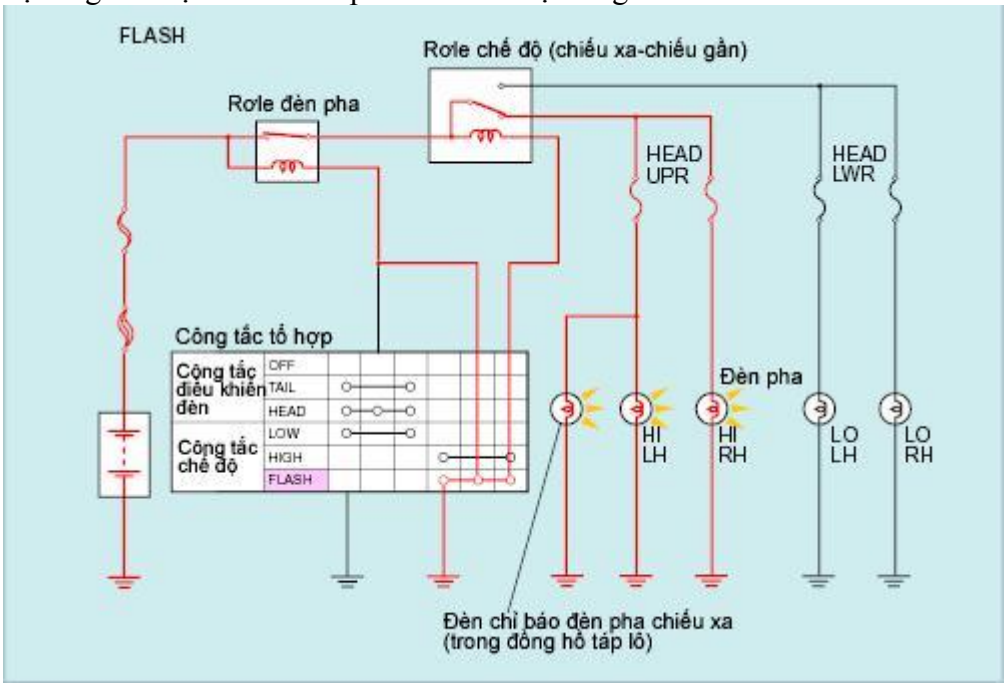
Khi công tắc đèn pha dịch chuyển tới vị trí HEAD (HIGH), thì các rơle đèn pha và điều chỉnh độ sáng đèn bật lên, các đèn pha-chiếu xa bật sáng và đèn chỉ báo đèn pha-chiếu xa trên bảng táp lô cũng bật sáng.



+ Nguyên lý hoạt động nháy pha FLASH

Khi công tắc điều khiển đèn dịch chuyển về vị trí FLASH, thì các rơle đèn pha và điều chỉnh

độ sáng đèn bật lên và đèn pha-chiếu xa bật sáng



Nguồn Oto-hui

Bài viết liên quan

- 17/12/2010 -- Cấu tạo hệ thống chiếu sáng xe nói chung (0)
- 24/12/2010 -- HT chiếu sáng ban ngày trên xe (0)
- 21/12/2010 -- Lich Thay Một Số Phụ Tùng Quan Trọng Cho Ô Tô (1)
- 09/05/2010 -- Hệ thống chống trộm xe hơi – Phần 3 (2)
- 18/04/2010 -- Cách khởi động khi ắc quy hết điện (3)
- 29/03/2010 -- Chuẩn đoán hư hỏng ắc quy – Phần 2 (2)
- 28/03/2010 -- Chuẩn đoán hư hỏng ắc quy – Phần 1 (0)
- 24/02/2010 -- Dây đai an toàn (0)
- 18/12/2010 -- Hệ thống Common Rail Diesel (0)
- 30/08/2010 -- Khác nhau cơ bản giữa : Chế hòa khí và phun xăng điện tử (0)

Trong chuyên mục: Bài viết mới,Lý thuyết,Đèn - Điện Từ khóa : ac quy, chế độ, chieu san, chiếu sáng, cong tac, den hau, den pha, dieu khien,

MỘT SỐ CHƯƠNG TRÌNH KHẢO SÁT, THIẾT KẾ HỆ THỐNG ĐIỀU KHIỂN TỰ ĐỘNG

(Nếu bạn nào quan tâm đến các chương trình thì liên hệ với PQT)

1. Chương trình 1:

Viết chương trình xác định hàm truyền vòng kín có khâu hồi tiếp đơn vị.

2. Chương trình 2:

Viết chương trình tìm cực và zero của hàm truyền.

3. Chương trình 3:

Viết chương trình khảo sát tính ổn định của hệ tuyến tính liên tục dùng giản đồ Bode.

4. Chương trình 4:

Tạo ra lệnh hurwitz để xét tính ổn định của hệ thống tuyến tính liên tục theo tiêu chuẩn Hurwitz.

5. Chương trình 5:

Viết chương trình tự động vẽ giản đồ Bode, biểu đồ Nyquist, quỹ đạo nghiệm của hệ tuyến tính liên tục.

6. Chương trình 6:

Viết chương trình để tìm các chỉ tiêu trong miền thời gian của hệ bậc 2.

7. Chương trình 7:

Viết chương trình để thực hiện bổ chính cho một hệ thống tuyến tính liên tục bằng giản đồ Bode.

8. Chương trình 8:

Viết chương trình khảo sát ảnh hưởng của khâu PID vào hệ thống tuyến tính bậc 2.
trong các tập tin này chương trình sẽ không thực hiện được.

9. Chương trình 9:

Viết lệnh dùng để khảo sát tính ổn định của hệ thống tuyến tính gián đoạn theo tiêu chuẩn Jury.

11. Chương trình 11:

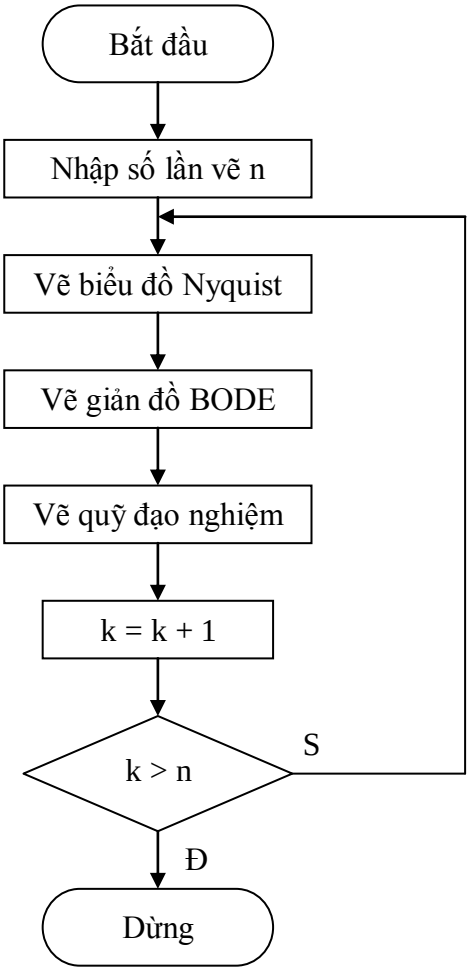
Viết chương trình đồ họa để vẽ các đáp ứng tần số và đáp ứng thời gian bằng cách chọn trong menu.

Chương trình được soạn thảo trong 2 tập tin dohoa.m và action.m và hệ thống trong chương trình này có hàm truyền là:

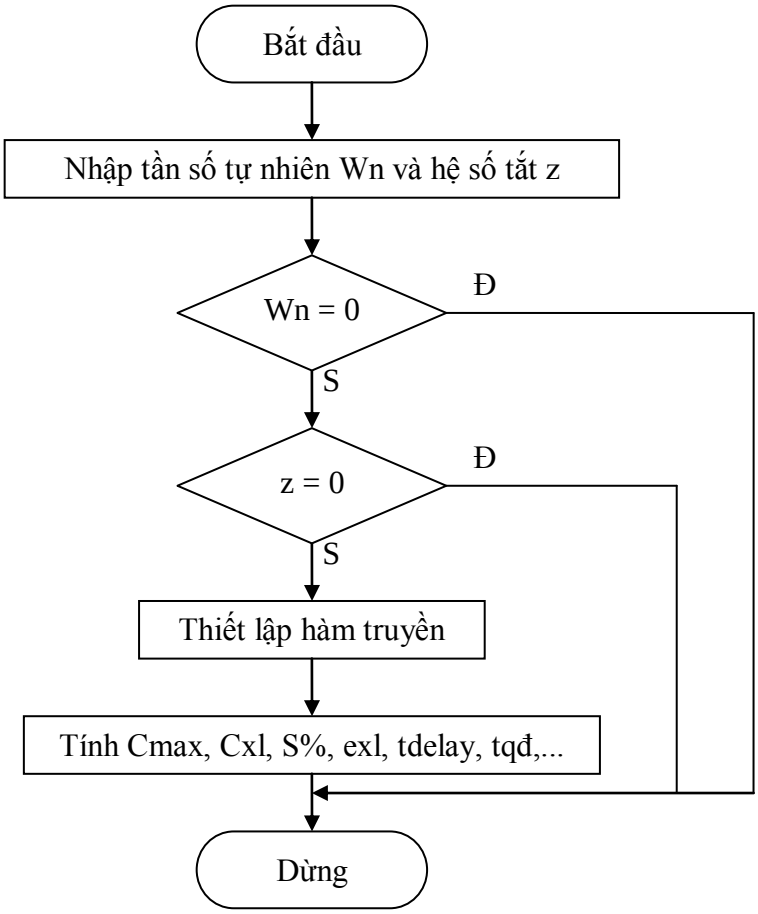
$$G(s) = \frac{1}{s(s\ \blacksquare 4)(s\ \blacksquare 6)}$$

PHỤ CHƯƠNG: LƯU ĐỒ CÁC CHƯƠNG TRÌNH

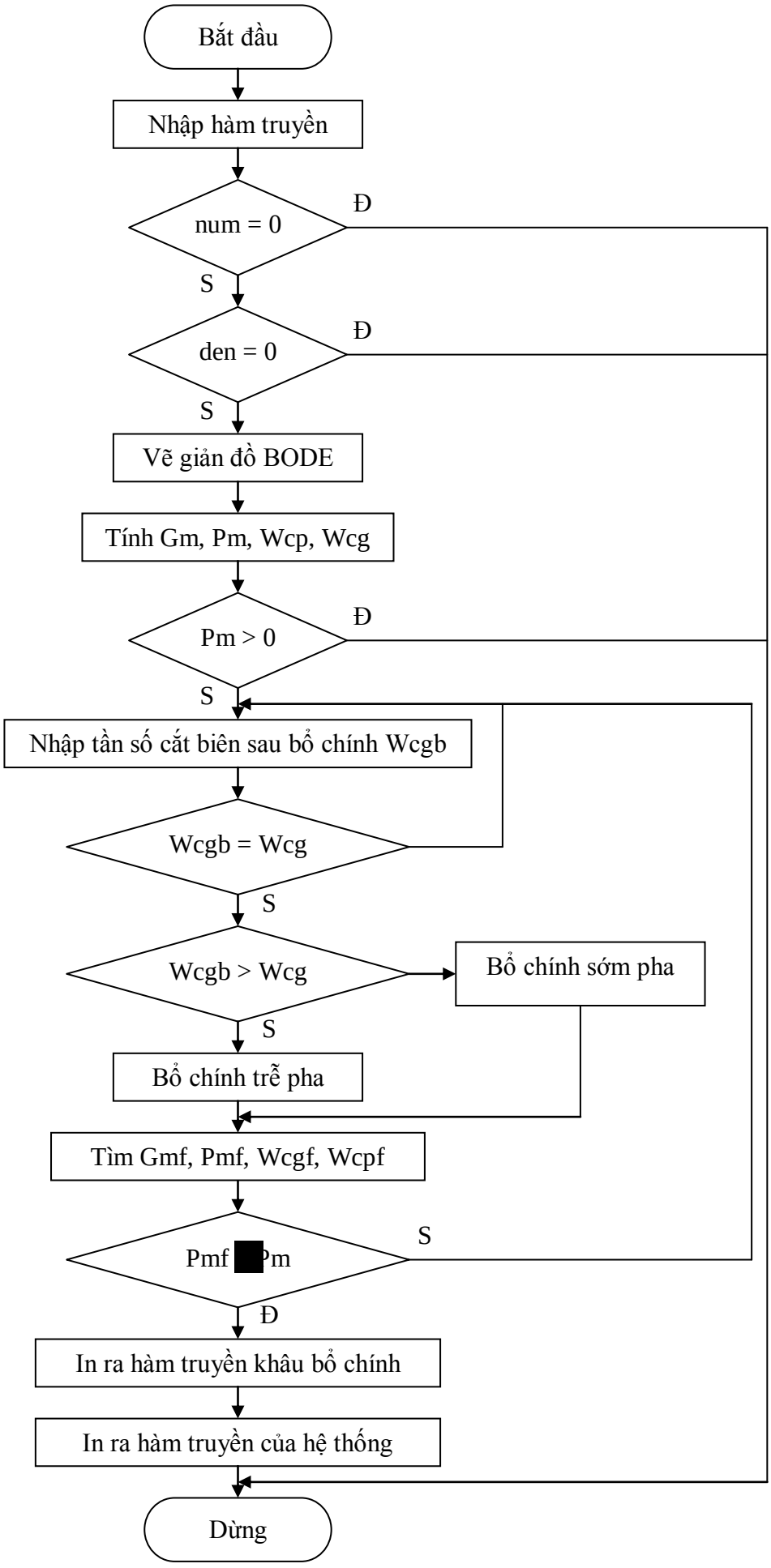
Lưu đồ chương trình tự động vẽ biểu đồ Nyquist, giản đồ Bode và quỹ đạo nghiệm


Bắt đầu
Nhập số lần vẽ n
Vẽ biểu đồ Nyquist
Vẽ giản đồ BODE
Vẽ quỹ đạo nghiệm
k = k + 1
k > n
S
Đ
Dừng

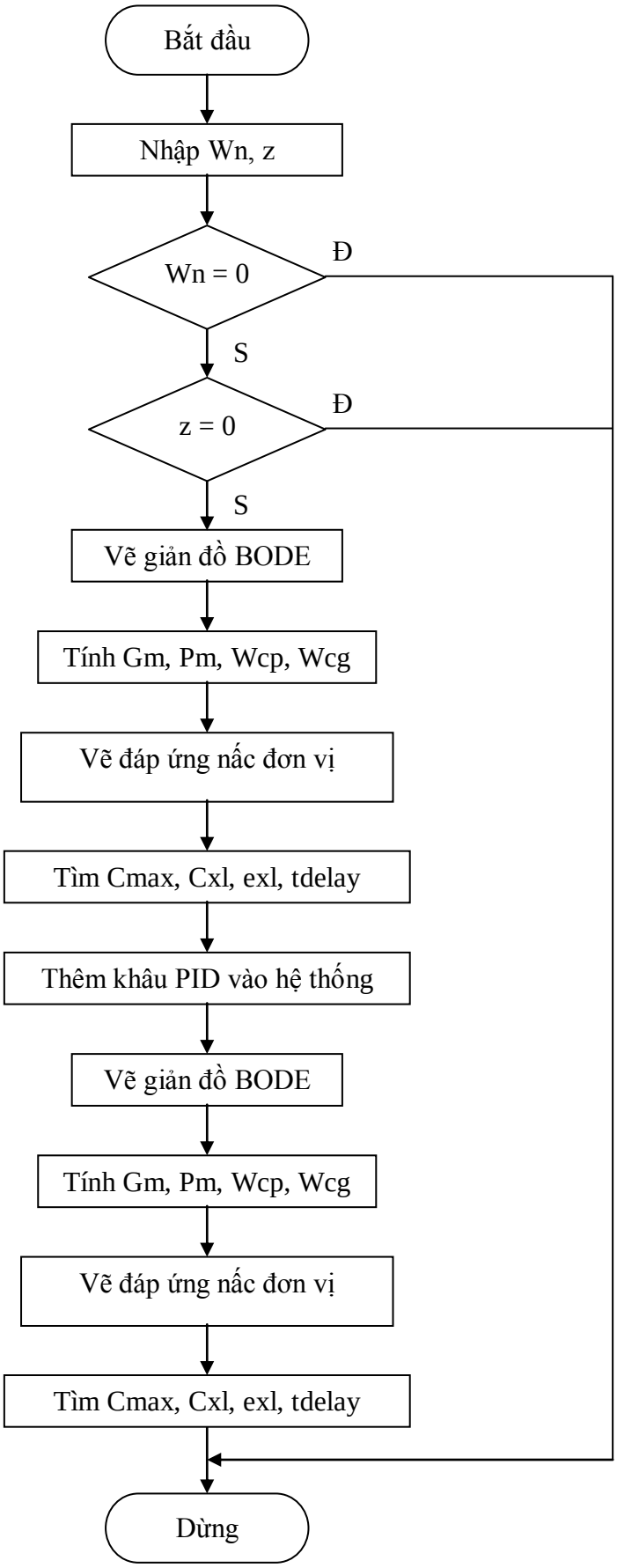
Chương trình tìm các chỉ tiêu trong miền thời gian của hệ bậc 2



Chương trình bổ chính cho hệ thống tuyến tính liên tục


Bắt đầu
Nhập hàm truyền
num = 0
Đ
S
den = 0
Đ
S
Vẽ giản đồ BODE
Tính Gm, Pm, Wcp, Wcg
Pm > 0
Đ
S
Nhập tần số cắt biên sau bổ chính Wcgb
Wcgb = Wcg
S
Wcgb > Wcg
Bổ chính sớm pha
S
Bổ chính trễ pha
Tìm Gmf, Pmf, Wcgf, Wcpf
Pmf Pm
S
Đ
In ra hàm truyền khâu bổ chính
In ra hàm truyền của hệ thống
Dừng

Chương trình khảo sát ảnh hưởng của khâu PID vào hệ thống



THIẾT KẾ HỆ THỐNG ĐIỀU KHIỂN HỘP CHẠY DAO

I. NHIỆM VỤ CHUNG:

Hệ thống điều khiển hộp chạy dao có nhiệm vụ thay đổi các cơ cấu truyền động trong hộp chạy dao để cắt được các loai ren khác nhau. Quá trình thay đổi các đường truyền thông qua việc đóng mở các ly hợp. Qua việc tham khảo máy chuẩn T620 ở đây ta bố trí 2 nhóm tay gạt 1 và 2 để thực hiện nhiệm vụ trên.

1. Đối với nhóm I (Tay quay tổ hợp I)

- Nhóm này có nhiệm vụ thay đổi các bước tp khi cắt mỗi loại ren

- Thay đổi vị trí ăn khớp của bánh răng Z36 ăn khớp với 1 trong 7 bánh của bộ Noóctông để thực hiện các bước ren trong các cột cơ sở.

- Thay đổi vị trí của khối bánh răng di trượt Z18+28 trên trục XIII và Z (28-48) trên trục XV để thực hiện các bước ren trong các cột cơ sở.

2. Đối với nhóm II (tay quay đơn)

- Nhóm này dùng để thay đổi chuyển động khi cắt các loại ren khác nhau theo yêu cầu.

Đối với mỗi loại ren khác nhau thì khi cắt tay gạt này có vị trí tương ứng khác nhau.

Cụ thể: + vị trí tiện ren quốc tế và môdun

+ vị trí tiện ren và pit

+ vị trí tiện ren chính xác

+ vị trí tiện ren mặt đầu

+ vị trí tiện trơn

Để thực hiện các yêu cầu trên nhóm gạt II phải điều khiển sự ăn khớp ra vào của bốn ly hợp M_3- M_3- M_4 – M_5 và bánh răng di trượt trên trục XI có Z=35. Như vậy nhóm I và II không thể thay thế lẫn nhau được. Vì trong cùng một lúc không thể cắt được 2 loại ren mà chỉ cắt được 1 loại ren 1 loại ren được cắt phải gạt 2 tay gạt.

II. CẤU TẠO- NGUYÊN LÝ- CÁCH TÍNH TOÁN HỆ THỐNG TAY GẠT

Nhóm I

Cấu tạo giống như nhóm I ở máy chuẩn.

Nguyên lý:

Điều khiển nhóm cơ sở:

+ Kéo tay quay tổ hợp I giá trị H thông qua hệ thống đòn và tỷ lệ cánh tay đòn và thông qua hệ thống này làm cho $Z_đ$ quay quanh O_2 1 góc β.

+ Tay quay tổ hộp ở trạng thái kéo ra và quay đi Để $Z_đ$ di trượt và lần luợt ăn khớp với các Z_n. Lúc đó cụm điều khiển kéo càng (lắp bánh $Z_đ$) nhờ chốt chạy trên rành xiên song song với độ côn của bánh nooctong. Vị trí ăn khớp được xác định bởi 1 trong 7 lỗ trên rãnh. Khi đẩy tay quay vào thì quá trình xảy ra ngược lại làm bánh đệm Z36 ăn khớp với 1 trong 7 bánh của bộ nooctong kết thúc nhóm điều khiển cơ sở.

Điều khiển nhóm gấp bội

Khi kéo tay quay tổ hợp ra chốt sẽ đi vào 1 trong 4 lỗ bánh răng Z36 khi quay tay điều khiển nhóm cơ sở thì chốt sẽ làm quay bánh răng này và 2 banhs răng nữa, bánh răng quay làm chốt dịch chuyển tác động vào càng gạt gạt khối 2 bậc Z28-48 tới vị trí ăn khớp của nó còn các bánh răng kia sẽ chuyển tác động đén càng làm dịch chuyển khối bánh răng 2 bậc Z (18-28) trên trục XIII.

1.Tính toán nhóm I (tay quay tổ hợp)

+ Những điều cần chú ý khi nghien cứu máy chuẩn.

Độ nâng a của rãnh A bằng bao nhiêu để khi lắc thì bánh đệm Z36 thoát ra hoàn toàn khỏi khối nooctong tạo ra 1 khoảng hở để khi bộ bánh đệm chuyển động dọc trục không bị va chạm.

Góc α có trị số bao nhiêu để phù hợp hành trình gạt và chốt chạy được dễ dàng trong rãnh.

Để trình độ nâng a cần tìm độ lắc yêu cầu của bánh đệm Z36 ăn khớp với bộ nooctong và tỷ số lắc (tỷ số lắc hằng,
tỷ số giữa khoảng dịch chuyển của điểm tiếp xúc trên Z36 với khối nooctong và khoảng dịch chuyển của chốt Q).

Qua hình vẽ ta thấy khi khối bánh răng đệm Z36 ăn khớp với Z48 thì độ lắc yêu cầu phải lơn hơn chiều cao răng 1 lượng nào đó để khi gạt không bị ảnh hưởng va đập.

Khi đó khoảng cách từ tâm quayO của cần lắc p tới tâm chốt B là nhỏ nhất. Do đó llắc là nhỏ nhất.

Khi bánh răng đệm ăn khớp với Z1(Z26) thì yêu cầu độ lắc là: $X = h_r + (Z7 - Z).m/2$ (h_r : h răng)

Độ lắc lúc này là lớn nhất nhưng khi đó khoảng cách từ tâm O của cần p tới chốt B lúc này cũng lơns nhất.

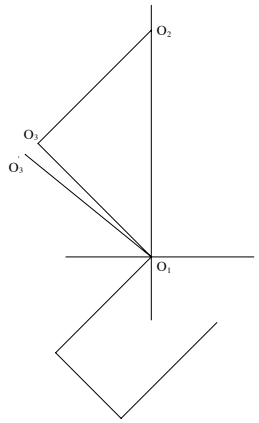
Như vậy cứ kéo từ Z1 đến Z7 của bộ nooctong thì độ nâng tăng dần lên và độ lắc cũng tăng dần và tỷ số i min và x min sẽ lấy với bánh Z7. Trong khi đó độ nâng a của rãnh A trên thanh n tỷ lệ thuận với với độ lắc x nhưng tỷ lệ nghịch voứi tỷ số lắc i. Do đó khi tính độ nâng a ta phải tinh ở hai vị trí tương ứng với bánh răng ăn khớp Z1và Z7 để chọn a nào lớn hơn.

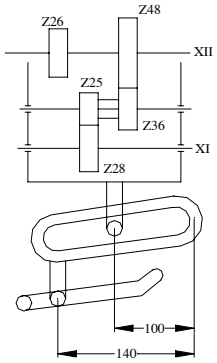
Tính toán độ nâng a khi ăn khớp với Z4

Từ số răng và modun của bánh răng ta tính được khoảng tâm lúc ăn khớp : $O_1 O_2 = 126$

$$o_1o_3 = \frac{m}{2}.(Z25 + Z28) = \frac{2,5}{2}(25 + 28) = 66,2mm$$

$$o_2o_3 = \frac{m}{2}.(Z7 + Z36) = \frac{2,5}{2}(48 + 36) = 84mm$$





Để lắc khối đệm Z36 tách ra khỏi vị trí ăn khớp với bộ nooctong thì khoảng cách các tâm như sau:

$O_1O_2 = 126; O_1O_3 = 66{,}2$

$O_2O_3^{'} = 84 + h_r = 84 + 2.m = 88$

Gọi góc lắc của O_3 là β thì $\beta = \hat{O}_1^{'} - \hat{O}_1$ .Ta có:

$$\cos\lambda = \frac{b^2 + c^2 - a^2}{2bc}$$

$$\Rightarrow \cos\hat{O}_1^{'} = \frac{\overline{O_1O_2}^2 + \overline{O_1O_3}^2 - \overline{O_2O_3}^2}{2.\overline{O_1O_2}.\overline{O_1O_3}} = \frac{126^2 + 66{,}2 - 88^2}{2.126.66{,}2} = 0{,}72$$

$$\Rightarrow \hat{O}_1^{'} = 43^0 3''$$

$$\cos\hat{O}_1 = \frac{\overline{O_1O_2}^2 + \overline{O_3O_1}^2 - \overline{O_2O_3}^2}{2.\overline{O_1O_2}.\overline{O_1O_3}} = \frac{126^2 + 66{,}2^2 - 84^2}{2.126.66{,}2} = 0{,}465$$

$$\Rightarrow \hat{O}_1 = 40^0 2'' \Rightarrow \beta = \hat{O}_1^{'} - \hat{O}_1 = 3^0 2'.$$

Khi $\hat{O}_3$ quay quanh O_1 một góc β thì T cũng quay quanh O_1 1 góc $\beta = 3^0 2'$. Như vậy chốt T dịch chuyển 1 đoạn bằng t, vì góc nâng nhỏ nên coi t bằng cung quay được

$$t = \frac{2\Pi.75}{360}.3^0 2' = \frac{2\Pi.75.181}{360.60} = 3{,}96$$

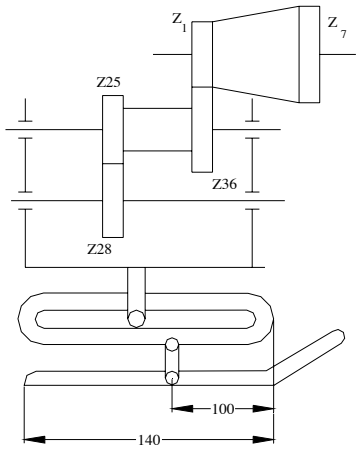
Như vậy chốt B phải dịch chuyển một đoạn đường bằng a (a bằng độ nâng của rãnh trên thanh n)

a=3,96/50.150=7,92mm

-Tính độ nâng a khi bánh đệm Z36 ăn khớp với Z1 của bộ nooctong

Góc cần thiết khi gạt là $\beta = \hat{O}_1' - \hat{O}_1$

Tương tự như trên ta có: $\hat{O}_1' = 43^0 3'$



$\cos\hat{O}_1 = 12,6^2 + 66,2^2 - 62^2/(2.12666,2) = 0,95 \Rightarrow \hat{O}_1 = 18^0 48'$

$\beta = \hat{O}_1' - \hat{O}_1 = 43^0 3' - 18^0 48' = 24^0 15'$

Như vậy chốt T phải quay 1 góc $\beta = 24^0 15'$

Tương ứng với chốt T phải dịch chuyển 1 đoạn t

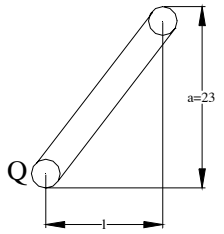
$$t = \frac{2\Pi.75.24^0 15'}{360} = \frac{2.360.75.1450}{360.60} = 31,8 \approx 32$$

Độ nâng $a = \frac{32.100}{140} = 22,8 \approx 23(mm)$

So sánh 2 vị trí ta có thể chọn a=23(mm)

- Tính góc nâng α của rãnh A

Nếu α càng nhỏ thì chốt Q chuyển động trong rãnh cũng dễ nhưng nếu nhỏ quá thì thanh n và rãnh A phải có kích thước dài để dễ chuyển động. Nếu α lớn thì thanh n ngắn dẫn đến chuyển động của thanh ngắn.



Sau khi nghiên cứu cách bố trí không gian và kích thước máy ta tính α rồi so sánh. Nếu $\alpha < 48^0$ là được.

Theo kinh nghiệm $\alpha = 38^0$ là tốt nhất.

Theo máy chuẩn ta chọn được các kích thước

$L = \frac{a}{tg\alpha} = \frac{23}{tg38^0} = 29,5mm$ lấy L=30mm.

Khi thay đổi tỷ số truyền ở nhóm cơ sở phải kéo trục ra để xoay theo máy chuẩn khoảng cách được tạo ra khi đó ≈ 30(mm). Ta chọn ≈ 33mm. Do đó khi chuyển động kéo ra thì $O \to O_1$; $O' \to O_1'$

$OO_1 = O'O_1' = 33mm$ .Tức là thanh n chuyển động được 1 đoạn là 33mm.Từ đó rút ra được $\alpha_{\min}$.

$$tg\alpha_{\min} = \frac{a}{L} = \frac{23}{30} = 0{,}78 \Rightarrow \alpha_{\min} = 58^0$$

Lấy α trong khoảng từ 38^0 - 45^0

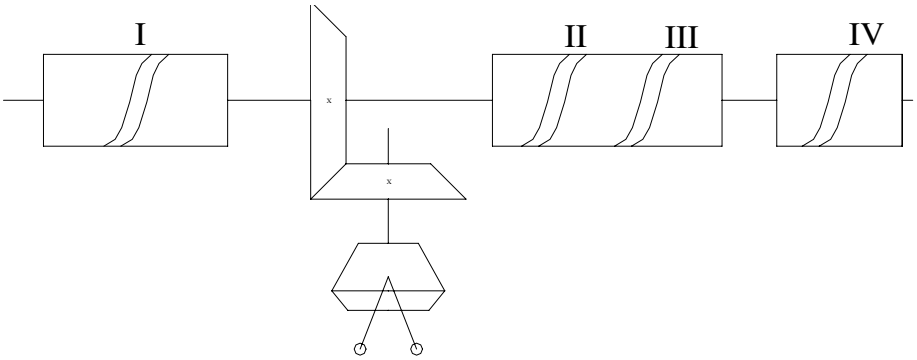
- Tính góc xoay cần thiết để dịch chuyển bánh đệm ăn khớp với bộ nooctong lúc này là rut tay quay ra và xoay 1 góc nhất định.

2.Tính toán nhóm II (tay quay đơn)

Tay quay II điều khiển trục mang 4 cam thùng I, II, III, IV

CamI: điều khiển ly hợp M_2 và bánh răng Z35 trên trục x

Cam II: Điều khiển ly hợp M_3



Cam III: Điều khiển ly hợp M_4

Cam IV: điều khiển ly hợp M_5

Nhiệm vụ các cam thùng trên làm nhiệm vụ đóng mở

các ly hợp để cắt các loại ren khác nhau: quốc tế- modun-pit chính xác

Phân tích các chuyển động khi cắt các loại ren ta có các vị trí các ly hợp $M_2; M_3; M_4; M_5$ và bánh Z35 (bánh F).

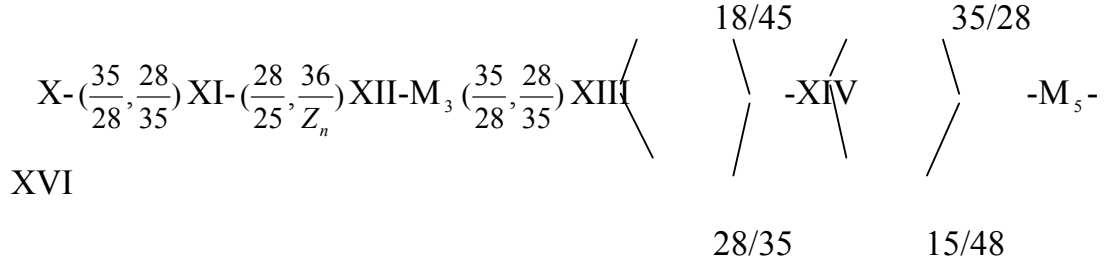
Khi cắt ren quốc tế và môđuyn (đường Nooctong chủ động)

$$\Rightarrow \text{X-M}_2\text{-XII-}(\frac{Z_n}{36} - \frac{25}{28})\text{-XI-M}_4\text{-XIII} \begin{cases} \diagup \ 18/45 \ \diagdown \\ \diagdown \ 28/35 \ \diagup \end{cases} \text{XIV-} \begin{cases} \diagup \ 35/28 \ \diagdown \\ \diagdown \ 15/48 \ \diagup \end{cases} \text{M}_5\text{-XV}$$

khi đó M_2-T ; M_4-T

M_3-P ; M_5-P

Khi cắt ren Anh+Pít (đường Nooctong bị động)



Khi đó M_2-P ; M_4-P

M_3-P ; M_5-P

Khi cắt ren chính xác:

Trục X- M_2-XII- M_3-XV- M_5-XVII

Khi đó M_2-T ; M_4-G

M_3-T ; M_5-P

Khi cắt ren mặt đầu:đường truyền giống như ren quốc tế chỉ khác là nối trục XV không nối vào vít me XVII mà qua tỉ số truyền 28/56-XVI (không qua M_6-ly hợp siêu việt).

Khi đó M_2-T ; M_4-T

M_3-P ; M_5-G

Tiện trơn: đường truyền giống ren quốc tế chỉ khác ở chỗ nối trục XV không nối vào trục vít me mà qua 28/56 M_6-XVI.

Khi đó M_2-T ; M_4-T

M_3-P ; M_5-G

Nhận xét: Khi tay gạt I quay 1 vòng thì nó sẽ phải thực hiện được việc điều chỉnh cắt tất cả các loại ren theo yêu cầu thiết kế máy.Do đó nhóm gạt II phải có 5 vị trí tương ứng với 5 loại ren kể trên.Tính lượng nâng thông qua hành trình gạt L:

Ly hợp M_2 :khi gạt để làm việc thì đồng thời phải cắt sự ăn khớp của bánh răng 35/28 ; L_2 =B+f=12+2=14mm

Ly hợp M_3: L_3=B+f=12+2=14mm

Ly hợp M_4 :L_4 =B+f=9+2=11mm

Ly hợp M_5 :L_5 =B+g=7+1=8mm

SƠ ĐỒ KHAI TRIỂN RÃNH CAM

Góc quay tay gạt (độ)	Loại ren	Cam I (M2)		Cam II (M3)		Cam III (M4)		Cam IV (M5)	
		Vị trí tay gạt	Dạng rãnh	Vị trí tay gạt	Dạng rãnh	Vị trí tay gạt	Dạng rãnh	Vị trí tay gạt	Dạng rãnh
0	Quốc tế+Môđuyn	T		P		T		P	
12	Anh-Pít	P		P		P		P	
144	Chính xác	T		T		G		P	
236	Mặt đầu	T		P		T		T	
308	Tiện tron	T		P		T		G	
360		T		P		T		P	

ư

MỤC LỤC

TÀI LIỆU THAM KHẢO

[1]-**Máy công cụ I**

Phạm Đắp, Nguyễn Hoa Đăng

[2]-**Thiết kế máy công cụ (tập 2)**

Nguyễn Anh Tuấn, Phạm Đắp

[3]-**Tính toán thiết kế máy cắt kim loại**

Phạm Đắp, Nguyễn Đức Lộc, Phạm Thế Trường, Nguyễn Tiến Lưỡng

[4]-**Tính toán thiết kế hệ dẫn động cơ khí (tập 1)**

Trịnh Chất, Lê Văn Uyển

[5]-**Chi tiết máy (tập 1,2)**

Nguyễn Trọng Hiệp

[6]-**Sổ tay công nghệ chế tạo máy**

Nguyễn Đắc Lộc, Lê Văn Tiến, Ninh Đức Tốn, Trần Xuân Việt

[7]-**Tập bản vẽ thiết kế máy tiện,khoan,phay**

MÔ TẢ PLC

THIẾT KẾ HỆ THỐNG ĐIỀU KHIỂN SỬ DỤNG PLC

I. TỔNG QUAN VỀ PLC:

1. Xuất xứ:

PLC viết tắt của từ Progammable Logic Control, là thiết bị điều khiển logic khả trình xuất hiện lần đầu tiên vào năm 1969 tại một hãng ôtô của Mỹ. Bắt đầu chỉ đơn giản là một bộ logic thuần tuý ứng dụng để điều khiển các quá trình công nghệ, chủ yếu là điều khiển ON/OFF giống như hệ thống rơle, công tắc tơ thông thường mà không điều khiển chất lượng hệ.
Từ khi xuất hiện PLC đã được cải tiến với tốc độ rất nhanh .
- năm 1974 PLCđã xử dụng nhiều bộ vi xử lý như mạchđịnh thời gian, bộ đếm dung lượng nhớ.
- Năm 1976 đã giới thiệu hệ thống đưa tín hiệu vào từ xa.
- Năm 1977 đã dùng đến vi xử lý.
- Năm 1980 PLC phát triển các khối vào ra thông minh nâng cao điều khiển thuận lợiqua viễn thông, nâng cao phát triển phần mềm, lập trình dùng máy tính cá nhân.
- Năm 1985 PLC đã được ghép nối thành mạngPLC.

Ngày nay PLCđã được cải tiến nhiều và đáp ứng tất cả các yêu cầu điều củakhiển như:
+ Điều khiế số lượng (ON/OFF).
+Điều khiển chất luợng(thực hiện các mạch vòng phản hồi: U, I,ω, S…).
Thực chất PLC là một máy tính công nghiệp mà quá trình điều khiển được thể hiện bằng chương trình. PLC thay thế hoàn toàn các phương pháp điều khiển truyền thống dùng rơ le, công tắc tơ.
Chính vì vậy PLC đướcử dụng rộng rãi trong công nghiệp, nó được xem như là một giải pháp điều khiển lý tưởng các quá trình sản xuất.

2. Vị trí, nhiệm vụ của bộ PLC trong hệ thống điều khiển:

Trong hệ thống điều khiển PLC là một khâu trung gian có nhiệm vụ xử lý các thông tin đầu vào rồi đưa tín hiệu ra tới các thiết bị chấp hành.

3. Ưu nhược điểm của PLC.

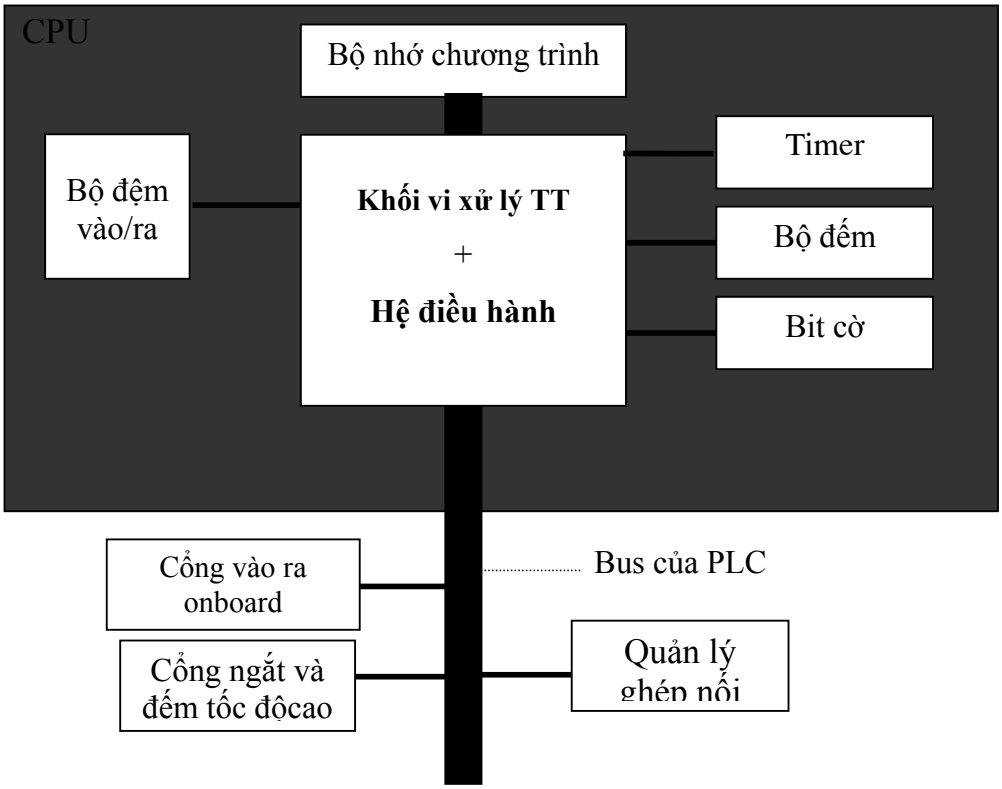
Ngày nay hầu hết các máy công nghiệp được thay thế các hệ điều khiển rơ le thông thường, sử dụng bán dẫn bằng các bộ điều khiển lập trình.
- Nó có các ưu điểm sau:

Giảm bớt quá trình ghép nối dây vì thế giảm giá thành đầu tư . Giảm diện tích lắp đặt, ít hỏng hóc, làm việc tin cậy, tốc độ quá trình điều khiển nhanh, khả năng chống nhiễu tốt, bảo trì bảo dưỡng tốt hơn vì nó có module chuẩn hoá.
- Nhược điểm:

Chưa thích hợp cho quá trình điều khiển nhỏ (một vài đầu ra) vì thế nếu dùng giá thành rất cao.

Ngôn ngữ hệ đóng (ngôn ngữ của các hãng riêng) nên khó thay thế.

4. Cấu trúc PLC:



Hình 3.1 Nguyên lý cấu trúc chung của bộ PLC.

5. Phần mềm PLC:

Phần mềm PLC có các loại ngôn ngữ khác nhau như:

+ Danh sách lệnh: STL.
+ Sơ đồ bậc thang: LAD DE R.
+ Sơ đồ khối chức năng: Block Function.
+ Ngôn ngữ bậc cao.

II. GIỚI THIỆU VỀ BỘ PLC S7-300:

PLC là loại thiết bị cho phép thực hiện linh hoạt các thuật toán điều khiển số thông qua một ngôn ngữ lập trình. PLC là một bộ điều khiển số nhỏ gọn, dễ thay đổi thuật toán và đặc biệt dễ trao đổi thông tin với môi trường

xung quanh (với PLC khác hoặc với máy tính). Toàn bộ chương trình điều khiển được lưu trong bộ nhớ của PLC dưới dạng các khốichương trình (Khối OB,FC hoăcFB) và được thực hiện theo chu kỳ vòng quét.

Đẻ thực hiện được chương trình thì PLC phải có tính năng như một máy tính, nghĩa là phải có CPU, hệ điều hành, bộ nhớ, các cổng vào ra. Ngoài ra PLC còn phải có thêm các khối chức năng đặc biệt như counter, timer... và nhữngkhối hàm chuyên dụng như bộ đệm vào /ra, bit cờ… .

1. Các module của PLCS7 - 300:

Thông thường, để tăng tính mềm dẻo trong ứng dụng thực tế mà ở đó phần lớn các đối tượng điều khiển có số tín hiệu đầu vào, đầu ra cũng như chủng loại tín hiệu đầu vào/ra khác nhau mà các bộ điều khiển PLC được thiết kế cũng khác nhau về cấu hình. Chúng được chia nhỏ thành các Module. Số các số Module được sử dụng nhiều hay ít tùy theo từng bài toán, song tối thiểu bao giờ cũng phải có một Module chính là Module CPU. Các Module còn lại là những Module nhận/truyền tín hiệu với số lượng điều khiển, các Module chức năng chuyên dụng như PID, điều khiển động cơ. Chúng được gọi chung là Module mở rộng. Tất cả các Module được gá trên những thanh ray (Rack).

a. Module CPU:

Module CPU là loại Module có chứa bộ vi xử lý, hệ điều hành, bộ nhớ, các bộ thời gian, bộ đếm, cổng truyền thông (RS315) và có thể có một vài cổng vào /ra số. Các cổng vào ra số có trên Module CPU được gọi là cổng vào ra onboard.

Trong họ PLCS7 – 300 có nhiều loại Module CPU khác nhau. Nói chung chúng được đặt tên theo bộ vi xử lý có trong nó như Module CPU 312, Module CPU 314, Module CPU 315… .

Những Module cùng sử dụng một loại bộ vi xử lý, nhưng khác nhau về cổng vào/ ra onboard cũng như các khối hàm đặc biệt được tích hợp sẵn trong thư viện của hệ điều hành phục vụ việc sử dụng các cổng vào/ra onboard này sẽ được phân biệt với nhau trong tên gọi bằng thêm cụm chữ cái IFM (viết tắt của Intergrated Function Module). Ví dụ như Module CPU 312 IFM, Module CPU 314 IFM, Module 315 IFM ….

Ngoài ra còn có các loại Module với 2 cổng truyền thông, trong đó cổng truyền thông thứ 2 có chức năng chính là phục vụ việc nối mạng phân tán. Tất nhiên kèm theo cổng truyền thông thứ 2 này là những phần mềm tiện dụng thích hợp cũng đã được cài sẵn trong hệ điều hành. Các loại Module CPU được phân biệt với những Module CPU khác bằng thêm cụm từ DP (Distributed Port) trong tên gọi. Ví dụ: Module CPU 312 DP…

b. Module mở rộng:

Các Module mở rộng được chia thành 5 loại chính:

1. PS (*Power Supply*) : Module nguồn nuôi, có 3 loại 2A, 5A, và 10A.

2. SM (*Signal Module*) : Module mở rộng cổng tín hiệu vào/ra, bao gồm:

- DI *(Digital Input*): Module mở rộng các cổng vào số. Số các cổng vào số mở rộng có thể là 8, 16 hoặc 32 tùy thuộc vào từng loại Module.
- DO (*Digital Output*): Module mở rộng các cổng ra số,Số cáccổng ra số mở rộng có thể là 8, 16 hoặc 32 tùy thuộc vào từng loại Module.
- DI/DO (*Digital Input/Digital Output*): Module mở rộng các cổng vào/ra số. Số các cổng vào/ra số mở rộng có thể là 8 vào/8 ra hoặc 16 vào/16 ra tùy thuộc vào từng loại Module.
- AI (*Analog Input*): Module mở rộng các cổng vào tương tự. Về bản chất chúng chính là những bộ chuyển đổi tương tự số 12bits (AD), tức là mỗi tín hiệu tương tự được chuyển thành tín hiệu số (nguyên) có độ dài 12bits. Số các cổng vào tương tự có thể là 2, 4 hoặc 8 tùy thuộc vào từng loại Module.
- AO (*Analog Output*): Module mở rộng các cổng ra tươngtự. Chúng chính là bộ (DA). Số các cổng vào tương tự có thể là 2 hoặc 4 tùy thuộc vào từng loại Module.
- AI/AO (*Analog Input/ Analog Output*): Module mở rộng các cổng vào/ra tương tự. Số các cổng vào/ra tương tự có thể là 4 vào/2 ra hoặc 4 vào/ 4 ra tùy thuộc vào từng loại Module.

3. IM (*Interface Module*): Module ghép nối. Đây là loại Module chuyên dùng có nhiệm vụ nối từng nhóm các Module mở rộng lại với nhau thành 1 khối và được quản lý chung bởi 1 Module CPU. Thông thường các Module mở rộng được gá liền nhau trên một thanh đỡ gọi là rack. Trên mỗi rack chỉ có thể gá được nhiều nhất 8 Module mở rộng (không kể Module CPU, Module nguồn nuôi). Một Module CPU S7 – 300 có thể làm việc trực tiếp được nhiều nhất 4 racks và các racks này phải được nối với nhau bằng Module IM.

4. FM (*Function module*): Module có chức năng điều khiển riêng, ví dụ như module điều khiển động cơ bước, Module điều khiển động cơ servo, Module PID, Module điều khiển vòng kín,…

5. CP (*Communication Module*): Module phục vụ truyền thông trong mạng giữa các PLC với nhau hoặc PLC với máy tính.

2. Kiểu dữ liệu và phân chia bộ nhớ:

a. Kiểu dữ liệu:

Một chương trình ứng dụng S7 – 300 có thể sử dụng các kiểu dữ liệu sau:

- BOOL: Với dung lượng 1 bit và có giá trị là 0 hoặc 1 (đúng hoặc sai). Đây là kiểu dữ liệu cho biến hai trị.
- BYTE: Gồm 8 bits, thường được dùng để biểu diễn một số nguyên dương trong khoảng từ 0 đến 255 hoặc mã ASCII của một ký tự.
- WORD: Gồm 2 bytes để biểu diễn 1 số nguyên dương từ 0 đến 65535.

- INT: Cũng có dung lượng là 2 bytes, dùng để biểu diễn số nguyên trong khoảng – 32768 đến 32767.
- DINT: gồm 4 bytes, dùng để biểu diễn một số nguyên từ – 2147483648 đến 2147483647.
- REAL: gồm 4 byte dùng để biểu diễn một số thực dấu phẩy động.
- S5T (hay S5TIME): khoảng thời gian, được tính theo giờ/phút/giây/mini giây.
- TOD: Biểu diễn giá trị thời gian tính theo giờ/phút/giây.
- DATE: Biểu diễn giá trị thời gian tính theo năm/tháng/ngày.
- CHAR: Biểu diễn một hoặc nhiều ký tự (nhiều nhất là 4 ký tự)

b. Cấu trúc bộ nhớ của CPU:

Bộ nhớ của S7–300 được chia làm 3 vùng chính:

***** Vùng chứa chương trình ứng dụng

Vùng nhớ chương trình được chia thành 3 miền:

-OB *(Organisation Block*): Miền chứa chương trình tổ chức.

- FC (*Function*): Miền chứa chương trình con được tổ chức thành hàm có biến hình thức để trao đổi dữ liệu với chương trình đã gọi nó.

- FB (*Function Block*): Miền chứa chương trình con, được tổ chức thành hàm và có khả năng trao đổi dữ liệu với bất cứ một khối chương trình nào khác. Các dữ liệu này phải được xây dựng thành khối dữ liệu riêng (gọi là DB – Data Block).

***** Vùng chứa tham số của hệ điều hành và chương trình ứng dụng

+ I *(Process image input*): Miền bộ đệm các dữ liệu cổng vào số. Trước khi bắt đầu thực hiện chương trình PLC sẽ đọc giá trị logic của tất cả các cổng đầu vào và cất giữ chúng tại vùng nhớ I, thông thường chương trình ứng dụng không đọc trực tiếp trạng thái logic của cổng vào số mà chỉ lấy dữ liệu của cổng vào từ bộ đệm I.

+. Q (*Process image output*): Miền bộ nhớ đệm các dữ liệu cổng ra số. Kết thúc giai đoạn thực hiện chương trình, PLC sẽ chuyển giá trị logic của bộ đệm Q tới các cổng ra số. Thông thường chương trình không trực tiếp gán giá trị tới cổng ra mà chỉ chuyển chúng vào bộ nhớ đệm Q.

+ M (Miền các biến cờ): Chương trình ứng dụng sử dụng vùng nhớ này để lưu giữ các tham số cần thiết và có thể truy cập nó theo bit(M) byte(MB), từ (MW) hay từ kép (MD).

+ T : Miền nhớ phục vụ bộ thời gian (Timer) bao gồm việc lưu trữ giá trị thời gian đặt trước (PV - Preset value) ,giá trị đếm thời gian tức thời (CV - Current value) cũng như giá trị logic đầu ra của bộ thời gian.

+ C : Miền nhớ phục vụ bộ đếm (Counter) bao gồm việc lưu giữ giá trị đặt trước (PV - Preset value),giá trị đếm tức thời (CV - current value) và giá trị logic đầu ra của bộ đếm.

+ PI: Miền địa chỉ cổng vào các module tương tự (I/O: external input).Các giá trị tương tự tại cổng vào của module tương tự sẽ được module đọc và

chuyển tự động theo những địa chỉ. Chương trình ứng dụng có thể truy cập miền nhớ PI theo từng byte (PIB), từng từ (PIW) hoặc từng kép (PID).

+ PQ: Miền địa chỉ cổng ra cho các module tương tự (I/O - external output). Các giá trị theo những địa chỉ này sẽ được module tương tự chuyển tới các cổng ra tương tự. Chương trình ứng dụng có thể truy cập miền nhớ PQ theo từng byte (PQB), từng từ kép (PQD).

c Vùng chứa các khối dữ liệu: Chia thành hai loại:

***** DB (*Data block*). Miền chứa các dữ liệu được tổ chức thành khối, kích thước cũng như số lượng, khối do người sử dụng qui định và phù hợp với từng bài toán điều khiển. Chương trình có thể truy cập miền này theo từng bit (DBX), byte (DBB), từ (DBW), từ kép (DBD).

***** L *(Local data block*): Đây là miền dữ liệu địa phương được các khối chương trình OB, FC, FB tổ chức và sử dụng cho các biến nháp tức thời và trao đổi dữ liệu của biến hình thức với những khối chương trình đã gọi nó. Nội dung của một số dữ liệu trong miền nhớ này sẽ bị xoá khi kết thúc chương trình tương ứng trong OB, FC. FB.

Miền nhớ này có thể truy nhập từ chương trình theo bit (L), byte (LB), từ (LW) hoặc từ kép (LD).

d. Trao đổi dữ liệu giữa CPU và các module mở rộng:

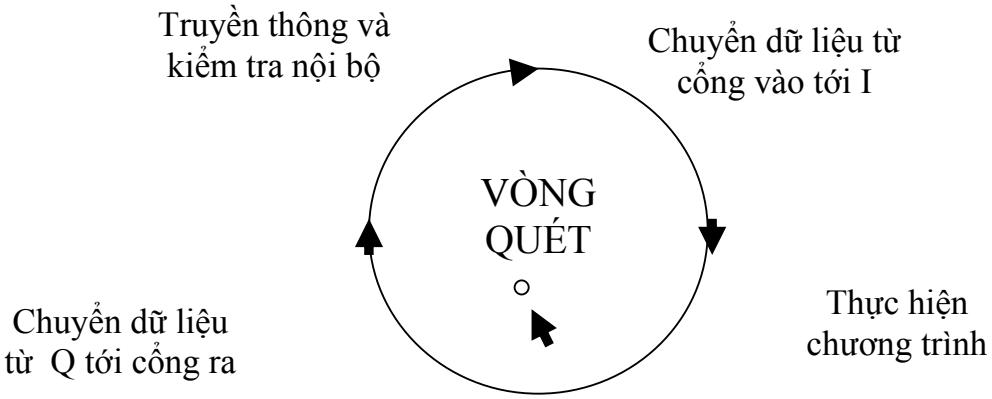
Trong trạm PLC luôn có sự trao đổi dữ liệu giữa CPU với các module mở rộng thông qua bus nội bộ. Ngay tại vòng quét, các dữ liệu tại cổng vào của các module số (DI) đã được CPU chuyển tới bộ đệm vào số (process image input table - I). Cuối mỗi vòng quét nội dung của bộ đệm ra số (process image input table - Q) lại được CPU chuyển tới cổng ra của các module ra số (DO). Việc thay đổi nội dung hai bộ đệm này được thực hiện bởi chương trình ứng dụng (user program). Điều này cho thấy nếu trong chương trình ứng dụng có nhiều lệnh đọc giá trị cổng vào số thì cho dù giá trị logic thực có của cổng vào này có thể đã bị thay đổi trong quá trình thực hiện vòng quét, chương trình sẽ vẫn luôn đọc được cùng một giá trị từ I và giá trị đó chính là giá trị của cổng vào có tại thời điểm đầu vòng quét. Cũng như vậy, nếu chương trình ứng dụng nhiều lần thay đổi giá trị cho một cổng ra số thì do nó chỉ thay đổi giá trị cho một cổng ra số thì do nó chỉ thay đổi nội dung bit nhớ tương ứng trong Q nên chỉ có giá trị ở lần thay đổi cuối cùng mới thực sự được đưa tới cổng ra vật lý của module DO.

Khác hẳn với việc đọc/ ghi cổng số, việc truy nhập cổng vào/ ra tương tự lại được CPU thực hiện trực tiếp với module mở rộng (AI/AO). Như vậy mỗi lệnh đọc giá trị từ địa chỉ thuộc vùng PI sẽ thu được một giá trị đúng bằng giá trị thực có ở cổng tại thời điểm thực hiện lệnh. Tương tự khi thực hiện lệnh gửi một giá trị (số nguyên 16 bits) tới địa chỉ của vùng PQ (Periphenal Output), giá trị đó sẽ được gửi ngay tới cổng ra tương tự của module. Do sự phân chia địa chỉ và đặc thù về tổ chức bộ nhớ của

S7-300 chỉ có các module vào/ ra số mới có bộ đệm còn các module vào/ ra tương tự thì không, chúng chỉ được cung cấp địa chỉ để truy nhập (địa chỉ PI và PQ). Tuy nhiên PI và PQ được cung cấp nhiều hơn AI/AO nên tạo khả năng kết nối các cổng vào / ra số với những địa chỉ dôi ra trong PI/PQ giúp chương trình ứng dụng có thể truy nhập trực tiếp các module DI/DO mở rộng để có giá trị tức thời tại cổng mà không cần thông qua bộ đệm I,Q.

3. Vòng quét chương trình:

PLC thực hiện chương trình theo chu trình lặp, mỗi vòng lặp được gọi là vòng quét (scan), mỗi vòng quét được bắt đầu bằng giai đoạn chuyển dữ liệu từ các cổng vào số tới vùng bộ đệm ảo I, tiếp theo là giai đoạn thực hiện chương trình.Trong từng vòng quét chương trình được thực hiện từ lệnh đầu tiên đến lệnh kết thúc của khối OB1 (Block End). Sau giai đoạn thực hiện chương trình là giai đoạn chuyển các nội dung của bộ đếm ảo Q tới các cổng ra số, vòng quét được kết thúc bằng giai đoạn truyền thông nội bộ và kiểm soát lỗi.



Hình 3.2. Vòng quét chương trình.

Bộ đệm I và Q không liên quan tới các cổng vào/ra tương tự nên các lệnh truy cập cổng tương tự được thực hiện trực tiếp với cổng vật lý chứ không thông qua bộ đệm.

Thời gian cần thiết để PLC thực hiện được một vòng quét gọi là thời gian vòng quét (Scan time). Thời gian vòng quét không cố định tức là không phải vòng quét nào cũng thực hiện trong khoảng thời gian như nhau. Có vòng quét thực hiện lâu có vòng quét thực hiện nhanh tuỳ thuộc vào số lệnh trong chương trình được thực hiện, vào khối dữ liệu được truyền thông trong vòng quét đó.

Như vậy giữa việc đọc dữ liệu từ đối tượng để xử lý, tính toán và việc gửi tín hiệu điều khiển tới các đối tượng có một khoảng thời gian trễ đúng

bằng thời gian vòng quét. Nói cách khác thời gian vòng quét quyết định tính thời gian thực của chương trình điều khiển trong PLC. Thời gian vòng quét càng ngắn thì tính thời gian thực của chương trình càng cao.

Nếu sử dụng các khối chương trình đặc biệt có chế độ ngắt như khối OB40, OB80 thì chương trình của các khối đó sẽ được thực hiện trong vòng quét khi xuất hiện tín hiệu báo ngắt cùng chủng loại. Các khối chương trình này có thể được thực hiện tại mọi điểm trong vòng quét chứ không bị gò ép là phải trong giai đoạn thực hiện chương trình. Ví dụ như một tín hiệu báo ngắt xuất hiện khi PLC đang ở giai đoạn truyền thông và kiểm tra nội bộ PLC sẽ tạm dừng công việc truyền thông và kiểm tra để thực hiện khối chương trình tương ứng với tín hiệu báo ngắt. Với hình thức xử lý tín hiệu báo ngắt như vậy thì thời gian vòng quét lớn khi trong vòng quét có nhiều tín hiệu ngắt. Do đó để nâng cao tính thời gian thực cho chương trình điều khiển thì tuyệt đối không viết chương trình xử lý ngắt quá dài hoặc quá lạm dụng việc sử dụng chế độ ngắt trong chương trình điều khiển.

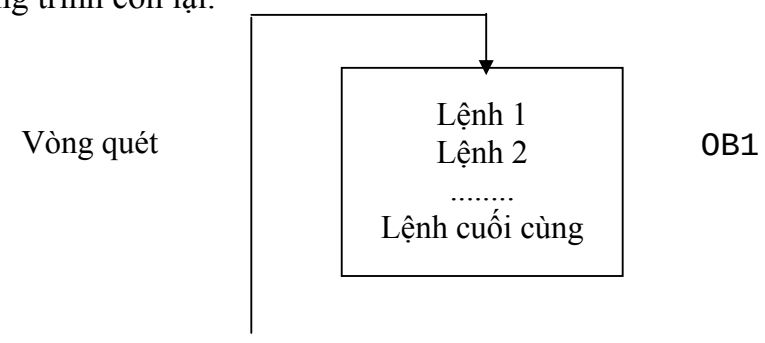
Tại thời điểm thực hiện lệnh vào/ ra thông thường lệnh không làm việc trực tiếp với cổng vào /ra mà chỉ thông qua bộ đệm ảo của cổng trong vùng nhớ tham số. Việc truyền thông giữa bộ đệm ảo với ngoại vi trong các giai đoạn 1 và 3 do hệ thống điều hành CPU quản lý**. Ở** một số module CPU khi gặp lệnh vào/ ra, ngay lập tức hệ thống cho dừng mọi công việc khác ngay cả chương trình xử lý ngắt để thực hiện trực tiếp với cổng vào/ ra.

4. Cấu trúc chương trình:

Chương trình cho S7-300 được lưu trong bộ nhớ của PLC ở vùng dành riêng cho chương trình và có thể được lập với 2 dạng cấu trúc khác nhau.

a. Lập trình tuyến tính:

Toàn bộ chương trình điều khiển nằm trong một khối trong bộ nhớ. Loại hình cấu trúc tuyến tính này phù hợp với bài toán tự động nhỏ, không phức tạp. Khối được chọn phải là khối OB1 là khối mà PLC luôn quét và thực hiện các lệnh trong nó thường xuyên từ lệnh đầu tiên đến lệnh cuối cùng và quay lại lệnh đầu tiên. Các khối OB khác không tham gia vào vòng quét mà được gọi bằng những tín hiệu báo ngắt. Mỗi tín hiệu báo ngắt như vậy chỉ có khả năng gọi một loại khối OB nhất định. Mỗi khi xuất hiện một tín hiệu báo ngắt hệ thống sẽ tạm dừng công việc đang thực hiện, chuyển sang chương trình xử lý ngắt trong các khối OB tương ứng. Sau khi thực hiện xong hệ thống mới trở về thực hiện tiếp chương trình còn lại.



Hình 3.3 Lập trình tuyến tính.

b. Lập trình có cấu trúc:

Chương trình được chia thành các phần nhỏ với từng nhiệm vụ riêng, các phần này nằm trong những khối chương trình khác nhau. Loại hình cấu trúc này phù hợp với những bài toán điều khiển nhiều nhiệm vụ và phức tạp. PLC S7-300 có 4 loại khối cơ bản.

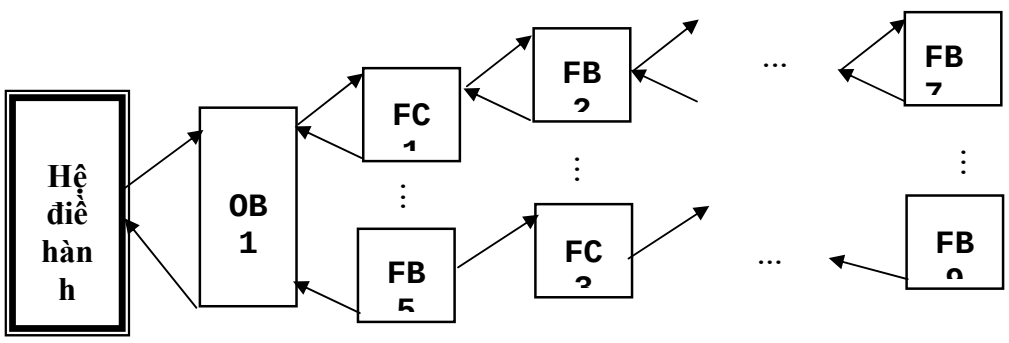
- Loại khối OB *(organization Block)*: Là khối tổ chức và quản lý chương trình điều khiển. Có nhiều loại khối OB với những chức năng khác nhau, chúng đuợc phân biệt với nhau bằng một số nguyên đi sau khối ký tự OB. Ví dụ như: OB1, OB35, OB80...

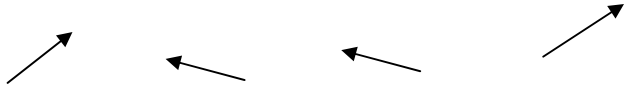
- Loại khối FC *(Progam Block)*: Khối chương trình với những chức năng riêng giống như một chương trình con hoặc một hàm (Chương trình con có biến hình thức). Một chương trình ứng dụng có thể có nhiều khối FC và các khối FC này được phân biệt với nhau bằng một số nguyên sau nhóm ký tự FC. Ví dụ như: FC1, FC2 ...

- Loại khối FB *(Function Block)*: Là loại khối FC đặc biệt có khả năng trao đổi một lượng dữ liệu lớn với các khối chương trình khác, các dữ liệu này phải tổ chức thành khối dữ liệu riêng có tên gọi là Data Block. Một chương trình ứng dụng có thể có nhiều khối FB và các khối FB này được phân biệt với nhau bằng một số nguyên sau nhóm ký tự FB. Chẳng hạn như FB1, FB2 ...

- Loại khối DB *(Data Block)*: Khối chứa các dữ liệu cần thiếtt để thực hiện chương trình. Các tham số của khối do người dùng tự đặt. Một chương trình ứng dụng có thể có nhiều khối DB và các khối DB này được phân biệt với nhau bằng một số nguyên sau nhóm ký tự DB. Vó dụ như: DB1, DB2 ...

Chương trình trong các khối được liên kết với nhau bằng các lệnh gọi khối, chuyển khối. Xem những phần chương trình trong các khối như là các chương trình con thì S7-300 cho phép gọi chương trình con lồng nhau, tức là từ chương trình con này gọi một chương trình con khác và từ chương trình con được gọi lại gọi tới một chương trình con thứ 3... . Một khối chương trình con không thể gọi đến chính nó. Số các lệnh gọi lồng nhau phụ thuộc vào từng chủng loại module CPU mà ta sử dụng. Ví dụ như đối với module CPU 314 thì số lệnh gọi lồng nhau nhiều nhất có thể cho phép là 8. Nếu số lần gọi khối lồng nhau mà vượt quá con số giới hạn cho phép, PLC sẽ chuyển sang chế độ STOP và đặt cờ báo lỗi.





Số lệnh gọi lồng nhau nhiều nhất cho phép phụ thuộc vào từng loại

Hình 3.4. Lập trình có cấu trúc

Khối OB1 luôn được PLC quét và thực hiện các lệnh từ lệnh đầu tiên đến lệnh cuối cùng và quay lại lệnh đầu tiên. Như vậy toàn bộ các khối chương trình được quản lý chặt chẽ bởi khối OB1.

c. Những khối OB đặc biệt:

Trong khi khối OB1 được thực đều đặn ở từng vòng quét trong giai đoạn thực hiện chương trình (giai đoạn 2) thì các khối OB khác chỉ được thực hiện khi xuất hiện tín hiệu báo ngắt tương ứng, nói cách khác chương trình viết cho các khối OB này là chương trình xử lý tín hiệu ngắt (event). Chúng bao gồm:

1. OB10 (*Time of Day interrupt*): Chương trình trong khối OB10 sẽ được thực hiện khi giá trị của đồng hồ thời gian thực nằm trong một khoảng thời gian đã được quy định. OB10 có thể được gọi một lần, nhiều lần cách đều nhau từng phút, thừng giờ, từng ngày ... Việc quy định khoảng thời gian hay số lần gọi OB10 được thực hiện nhờ chương trình hệ thống SFC 28 hoặc trong bảng tham số của Module CPU nhờ phần mềm Step7.

2. OB20 (*Time Delay interrupt*): Chương trình trong khối OB20 sẽ được thực hiện sau một khoảng thời gian trễ đặt trước kể từ khi gọi chương trình hệ thống SFC32 để đặt thời gian trễ.

3. OB35 (*Cyclic interrupt*): Chương trình trong OB35 sẽ được thực hiện cách đều nhau một khoảng thời gian cố định. Mặc định, khoảng thời gian này sẽ là 100 ms, song ta có thể thay đổi nó trong bảng tham số của Module CPU nhờ phần mềm Step.

4 . OB40 (*Hardware interrupt*): Chương trình trong OB40 sẽ được thực hiện khi xuất hiện 1 tín hiệu báo ngắt từ ngoại vi đưa vào Module CPU thông qua các cổng vào ra số onboard đặc biệt, hoặc thông qua các Module SM, CP, FM.

5. OB80 (*Cyle Time Fault*): Chương trình trong khối OB 80 sẽ được thực hiện khi thời gian vòng quét (Scan Time) vượt quá khoảng thời gian cực đại đã quy định hoặc khi có một tín hiệu ngắt gọi một khối OB nào đó mà khối OB này chưa kết thúc ở lần gọi trước. Mặc định, Scan time cực đại là 150ms, nhưng có thể thay đổi nó thông qua bảng tham số của module CPU nhờ phần mềm STEP 7.

6. OB 81 *(Power Supply Fault):*Module CPU sẽ gọi chương trình trong khối OB81 khi phát hiện thấy có lỗi về nguồn nuôi.

7. OB82 *(Diagnostic Interrupt):* Chương trình trong OB82 được gọi khi CPU phát hiện có sự cố từ các module vào/ ra mở rộng. Các module mở rộng này phải là những module có khả năng tự kiểm tra mình (diagnostic cabilities).

8. OB85 (*Not Load Fault*): CPU sẽ gọi khối OB85 khi phát hiện thấy chương trình ứng dụng có sử dụng chế độ ngắt nhưng chương trình xử lý tín hiệu ngắt lại không có trong khối OB tương ứng.

9. OB87 *(Communication Fault):* Khối OB87 sẽ được gọi khi CPU phát hiện thấy lỗi trong truyền thông .Ví dụ như không có tín hiệu trả lời từ đối tác.

10. OB100 (*Start Up Information*): Khối OB100 sẽ được thực hiện một lần khi CPU chuyển trạng thái từ STOP (dừng) sang RUN (chạy).

11. OB101 *(Cold Start Up Information - Chỉ có với S7-400):* Khối OB101 sẽ được thực hiện một lần khi công tắc nguồn của CPU chuyển trạng thái từ OFF sang ON.

12. OB121 *(Synchronous error)*: Khối OB121sẽ được thực hiện khi CPU phát hiện thấy lỗi logic trong chương trình như đổi sai kiểu dữ liệu hoặc lỗi truy nhập khối DB, FC, FB không có trong bộ nhớ của CPU.

13. OB122 (*Synchronous error*): Khối OB122 sẽ được thực hiện khi CPU phát hiện thấy lỗi truy nhập module trong chương trình. Ví dụ chương trình có lệnh truy nhập module vào ra mở rộng nhưng lại không tìm thấy module này.

5. Tổ chức bộ nhớ CPU:

Bộ nhớ CPU bao gồm:

- Vùng nhớ chứa các thanh ghi.
- Vùng System memory.
- Vùng Load memory.
- Vùng word memory.

Kích thước của các vùng nhớ này phụ thuộc vào chủng loại của từng module CPU.

Load memory: Là vùng nhớ chứa chương trình ứng dụng (do người sử dụng viết) bao gồm tất cả các khối chương trình ứng dụng OB, FC, FB, các khối chương trình trong thư viện hệ thống được sử dụng (SFC, SFB) và các khối dữ liệu DB. Vùng nhớ này được tạo bởi một phần bộ nhớ RAM của CPU và EEPROM (nếu có EEPROM). Khi thực hiện động tác xóa bộ nhớ (MRES) toàn bộ các khối chương trình và khối dữ liệu nằm trong RAM sẽ bị xoá. Cũng như vậy khi chương trình hay khối dữ liệu được đổ (download), từ thiết bị lập trình (PG, máy tính) vào module CPU, chúng sẽ được ghi lên phần RAM của vùng nhớ Load memory.

Work memory: Là vùng nhớ chứa các khối DB đang được mở, khối chương trình (OB, FC, DB, SFC hoặc SFB) đang được CPU thực hiện và phần bộ nhớ cấp phát cho những tham số hình thức để các khối chương trình này

trao đổi tham trị với hệ điều hành và với các khối chương trình khác (local block). Tại một thời điểm nhất định vùng Work memory chỉ chứa một khối chương trình. Sau khi khối chương trình đó được thực hiện xong thì hệ điều hành sẽ xoá nó khỏi Work memory và nạp vào đó khối chương trình kế tiếp đến lượt thực hiện.

System memory: Là vùng nhớ chứa các bộ đệm vào/ ra số (Q, I), các biến cờ (M), thanh ghi C - Word, PV, T- bit của Timer, thanh ghi C - Word, PV, C - bit của Counter. Việc truy cập, sửa đổi dữ liệu những ô nhớ thuộc vùng nhớ này được phân chia hoặc bởi hệ điều hành của CPU hoặc do chương trình ứng dụng.

Trong các vùng nhớ không có vùng nhớ nào được dùng làm bộ đệm cho các cổng vào/ ra tương tự hay nói cách khác các cổng vào/ ra tương tự không có bộ đệm và như vậy mỗi lệnh truy nhập module tương tự (đọc hoặc gửi giá trị) đều có tác dụng trực tiếp tới cổng vật lý của module.

*** Ý nghĩa các vùng nhớ của CPU S7 - 300:**

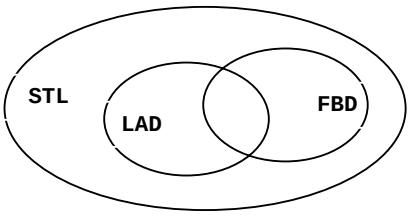
Tên gọi	Kích thước truy cập	Kích thước tối đa (phụ thuộc CPU)	Ý nghĩa
Procces image Iput (I) Bộ đệm vào số	I IB TW ID	0.0 ÷ 1277 0 ÷ 127 0 ÷ 126 0 ÷ 124	Đầu mỗi vòng quét hệ điều hành sẽ ghi vào phần nhớ này các giá trị được lấy từ cổng vào số (digital input) vật lý của module mở rộng.
Procces image Output (Q) Bộ đệm ra số	Q QB QW QD	0.0 ÷ 1277 0 ÷ 127 0 ÷ 126 0 ÷ 124	Cuối mỗi vòng quét, hệ điều hành sẽ đọc nội dung của miền nhớ này và chuyển ra cổng ra số (digital input) của các module mở rộng.
Bit Memory (M) Vùng nhớ cờ	M MB MW MD	0.0 ÷ 255.7 0 ÷ 255 0 ÷ 254 0 ÷ 252	Được sử dụng như một miền các biến cờ cho chương trình ứng dụng.
Timer (T)	T0 ÷ T255		Miền nhớ lưu giữ các giá trị PV, CV và T - bit của Timer. Được truy nhập để sửa đổi bởi hệ điều hành và chương trình ứng dụng.
Counter (C)	C0 ÷ C255		Miền nhớ lưu giữ các giá trị PV, CV và C - bit của Counter. Được truy nhập để sửa đổi bởi hệ điều hành và chương trình ứng dụng.
Data block (DB) Khối dữ liệu share	DB X DBB DBW DBD	0.0 ÷ 65535.7 0 ÷ 65535 0 ÷ 65534 0 ÷ 65532	Được mở bằng lệnh "OPN DB"
Data lock (DI) Khối dữ liệu Instance	DI X DIB DIW DID	0.0 ÷ 65535.7 0 ÷ 65535 0 ÷ 65534 0 ÷ 65532	Là khối DB.Được mở bằng lệnh "OPN DI"
Local block (L)	L	0.0 ÷ 65535.7	Miền nhớ được cấp phát cho các khối FC, FB mỗi

Miền nhớ địa phương cho các tham số hình thức	LB LW LD	0 ÷ 65535 0 ÷ 65534 0 ÷ 65532	khi khối này được gọi để thực hiện. Miền nhớ này cũng sẽ được giải phóng khi thực hiện xong các khối chương trình đó.
Peripheral input (PI)	PIB PIW PID	0 ÷ 65535 0 ÷ 65534 0 ÷ 65532	Chỉ có địa chỉ truy cập để đọc. Không có phần bộ nhớ thực sự.
Peripheral output (PQ)	PQB PQW PQD	0 ÷ 65535 0 ÷ 65534 0 ÷ 65532	Chỉ có địa chỉ truy cập để ghi. Không có phần bộ nhớ thực sự.

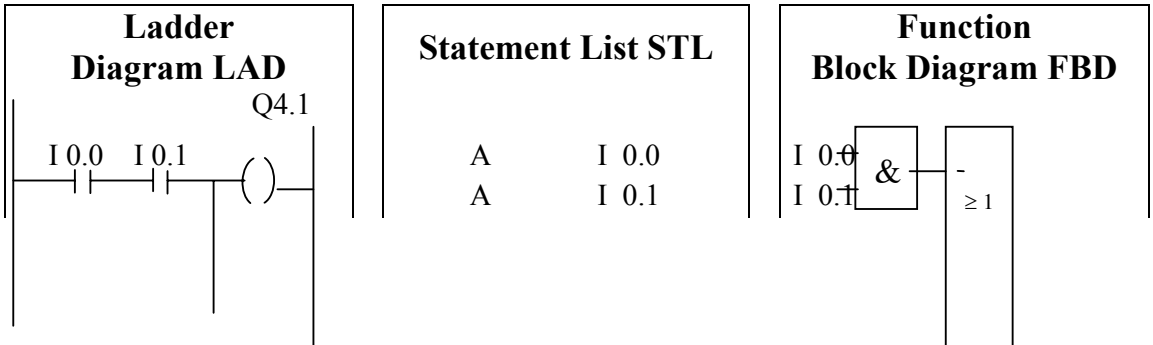
6. Ngôn ngữ lập trình STL:

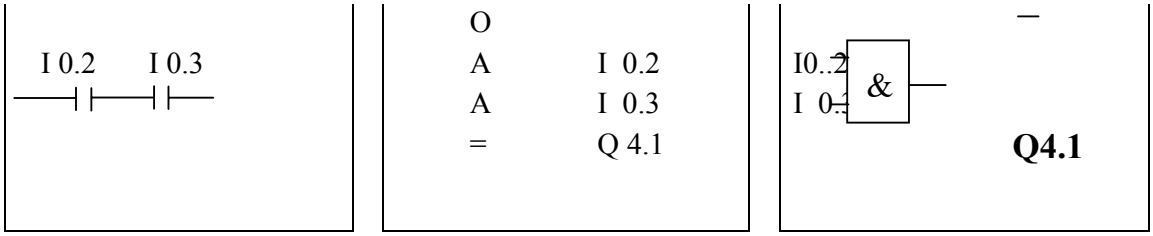
Các loại PLC nói chung thường có nhiều ngôn ngữ lập trình nhằm phục vụ các đối tượng sử dụng khác nhau. PLC S7 - 300 có ba ngôn ngữ lập trình cơ bản. Đó là:

- Ngôn ngữ "liệt kê lệnh", ký hiệu là STL *(Statement list)*: Đây là dạng ngôn ngữ lập trình thông thường của máy tính. Một chương trình được phép bởi nhiều câu lệnh theo một thuật toán nhất định, mỗi lệnh chiếm một hàng và đều có cấu trúc chung "tên lệnh" + "toán dạng".
- Ngôn ngữ "hình thang", ký hiệu là LAD *(Ladder logic)*: Đây là dạng ngôn ngữ đồ hoạ thích hợp với những người quen thiết kế mạch điều khiển logic.
- Ngôn ngữ "hình khối", ký hiệu là FBD *(Function block diagram)*: Đây cũng là kiểu ngôn ngữ đồ hoạ dành cho người có thói quen thiết kế mạch điều khiển số.



Hình 3.5. STL là ngôn ngữ mạnh nhất trong ba loại ngôn ngữ lập trình cho S7-300.





Hình 3.6. Ba kiểu ngôn ngữ lập trình cho S7-300

Một chương trình viết trên LAD hoặc FBD có thể chuyển sang được dạng STL, nhưng ngược lại thì không. Trong STL có nhiều lệnh không có trong LAD hay FBD (hình 3.5). Cũng chính vì lý do đó, tôi chọn STL làm ngôn ngữ lập trình minh họa trong đồ án này.

Tạp chí Tin học và Điều khiển học, T.20, S.3 (2004), 205–218

DESIGNING A LIFT CONTROL SYSTEM

PHAM TRAN NHU[1], NGUYEN VAN TRUONG[2]

[1]*Institute of Information Technology,*
[2]*Pedagogical University-Thai Nguyen University*

Abstract. In this paper, we present an application of syntactical approach given in a formal design technique for real-time embedded systems. The technique is the model of discretization at the state level and the approximation of continuous state variables by discrete ones. The lift system presented in this paper shall be monitored and controlled by a computing system that shall respect the components, handle the events, and satisfy the usual procedures and invariants. The Duration Calculus with Iteration is used in the paper to specify requirements of the system.

Tóm tắt. Trong bài báo này chúng tôi trình bày một ứng dụng - hệ thống điều khiển thang máy - theo cách tiếp cận hình thức cho các hệ nhúng. Kỹ thuật thiết kế được dùng là mô hình hoá sự rời rạc và xấp xỉ các biến trạng thái liên tục bởi các biến trạng thái rời rạc. Hệ thống thang máy được giám sát và điều khiển thông qua một hệ thống tính toán nhằm quản lý các bộ phận, điều khiển các sự kiện và làm thoả mãn các thủ tục cùng những bất biến đặc trưng cho hệ thống. Tính Toán khoảng lặp được dùng trong bài viết để đặc tả các yêu cầu của hệ thống.

1. INTRODUCTION

The lift control system belongs among real-time control systems. The system consists of some physical plant, in permanent interaction with its environment, for which a suitable controller has to be constructed such that the controlled plant exhibits the desired time dependent behavior. Many authors have proposed approachs for designing the lift control system (e.g. [2, 10]). However, some approach is just a postulate - it has not yet been widely tested, so a failure in the reaction of the plant may appear. The problem is to use suitable technique for specifying and reasoning about the design of the system.

For any real-time systems in general, and for our lift control system in particular, the continuous model (real numbers) is suitable for specifying the continuous behavior of the states of the environment and those of the plant, which can change at any time according to the laws of physics. However, the state of a digital program changes only at discrete time points at ticks of a computer clock, so the discrete model (natural numbers) should be considered for implementation of the system. Therefore, it is appropriate to combine two models into the same formalism such that the design and its correctness can accurately be reasoned about in an uniform manner.

Using formal methods is a key solution for buiding a correct system. In this paper, we apply Duration Calculus with Iteration (DC^*), a logic obtained by extending Duration Calculus (DC) (cf. e.g. [12]) with the iteration operator (*) [1], to model of our lift control system.

This makes for a logical framework that can handle both continuous time and discrete time models for the system.

The design process can be formalised as follows. Firstly, a state variables model of the system should be defined. The state variables model comprises continuous state variables (modeling the behavior of continuous components) and discrete state variables (modeling the behavior of discrete components). Secondly, the requirement of the system is formalized as a DC formula Req over continuous state variables. A design decision must be established in order to the requirement of the system will be met and refined into a detailed design Des over continuous state variables such that $A \vdash Des \Rightarrow Req$, where A stands for some assumptions about the behavior of the environment and the relationship between continuous state variables. Finally, the discretization step follows. We approximate continuous state variables by discrete ones and formalize the relationship between them based on the general behavior of the sensors and actuators. The control requirement is derived from the detailed design and refined into a $simpleDC^*$ formula $Cont$ over discrete state variables such that $A_c \vdash Cont \Rightarrow Des$, where A_c stands for some assumptions about the behavior of the environment and the relationship between continuous state variables, and relationship between discrete state variables and continuous ones. The discrete formula $Cont$ is the formal specification of the controller.

The remaining of the paper is organized as follows. In Section 2 we give a brief summary of DC^*. The discretization technique is peresented in Section 3. Some refinement and verification rules are given in Section 4. The formal design process of the lift control system, the main part of the paper, is contained in Section 5. Section 6 concludes the paper.

2. DURATION CALCULUS WITH ITERATION

In this section we give a brief summary of DC^*. The readers are referred to [1] for more details on the calculus.

A language for DC^* is built starting from the following sets of *symbols*: a set of *constant symbols* $\{a, b, c, \}$, a set of *individual variables* $\{x, y, z, ...\}$, a set of *state variables* $\{P, Q, ...\}$, a set of *temporal variables* $\{u, v, ...\}$, a set of *function symbols* $\{f, g, ...\}$, a set of *relation symbols* $\{R, U, ...\}$, and a set of *temporal propositional letters* $\{A, B, ...\}$.

A DC^* language definition is essentially that of the sets of *state expressions* S, *terms* t and *formulas* φ of the language. These sets can be defined inductively by the following BNFs:

$S \triangleq \mathbf{0} | P | \neg S | S \vee S$

$t \triangleq c | x | u | \int S | f(t, ..., t)$

$\varphi \triangleq A | R(t, ..., t) | \neg\varphi | (\varphi \vee \varphi) | (\varphi^\frown\varphi) | (\varphi^*) | \exists x\varphi$

A state variable P is interpreted as a function $I(P) : IR^+ \to \{0, 1\}$ (a state). $I(P)(t) = 1$ means that state P is present at time t, and $I(P)(t) = 0$ means that P is not present at time t. We assume that a state has finite variability in any finite time interval. A state expression is interpreted as a function which is defined by the interpretations for the state variables and boolean operators.

For an arbitrary state expression S, its duration is denoted by $\int S$. Given an interpretation I of the state variables and an interval, duration $\int S$ is interpreted as the accumulated length

of time within the interval at which S is present. So for any interval $[t, t']$, the interpretation $I(\int S)([t, t'])$ is defined as $\int_t^{t'} I(S)(t)dt$.

A formula φ is satisfied by an interpretation in an interval $[t, t']$ when it evaluates to true for that interpretation over that time interval. This is written as $I, [t, t'] \models \varphi$.

Given an interpretation I, a formula $\varphi^\frown\varphi'$ is true for $[t, t"]$ if there exists a t' such that $t \leq t' \leq t"$ and φ and φ' are true for $[t, t']$ and $[t', t"]$, respectively.

We consider the following abbreviations:

$\ell \stackrel{\wedge}{=} \int \mathbf{1}$, $\lceil S \rceil \stackrel{\wedge}{=} (\int S = \ell) \wedge (\ell > 0)$, $\diamond\varphi \stackrel{\wedge}{=} (true^\frown\varphi^\frown true)$, and $\Box\varphi \stackrel{\wedge}{=} \neg\diamond\neg\varphi$. We assume that boolean connectives bind more tightly than $^\frown$. Besides, we use some other symbols as abbreviations in the usual way.

The proof system for DC^* consists of a complete Hilbert-style proof system for first order logic (cf. e.g. [8]), axioms and rules for interval logic, Duration Calculus axioms and rules and axioms about iteration (cf. e.g. [12]). We only recall here some axioms and rules of the proof system of DC^*.

$(DC1)$ $\int \mathbf{0} = 0$

$(DC2)$ $\int \mathbf{1} = \ell$

$(DC3)$ $\int S \geq 0$

$(DC4)$ $\int S_1 + \int S_2 = \int(S_1 \vee S_2) + \int(S_1 \wedge S_2)$

$(DC5)$ $(\int S = x^\frown \int S = y) \Rightarrow \int S = x + y$

$(DC6)$ $\int S_1 = \int S_2$ if $S_1 \Leftrightarrow S_2$ in propositional calculus.

$$[\ell = 0/A]\varphi \ \varphi \Rightarrow [A^\frown\lceil S\rceil/A]\varphi$$

(IR1)
$$\frac{\varphi \Rightarrow [A^\frown\lceil\neg S\rceil/A]\varphi}{[true/A]\varphi}$$

$$[\ell = 0/A]\varphi \ \varphi \Rightarrow [\lceil S\rceil^\frown A/A]\varphi$$

(IR2)
$$\frac{\varphi \ \Rightarrow [\lceil\neg S\rceil^\frown A/A]\varphi}{[true/A]\varphi}$$

(ω)
$$\frac{\forall k < \omega[(\lceil S\rceil \vee \lceil\neg S\rceil)^k/A]\varphi}{[true/A]\varphi}$$

(DC_1^*) $\ell = 0 \Rightarrow \varphi^*$

(DC_2^*) $(\varphi^\frown\varphi^*) \Rightarrow \varphi^*$

(DC_3^*) $(\varphi^* \wedge \varphi'^\frown true) \Rightarrow (\varphi' \wedge \ell = 0^\frown true) \vee (((\varphi^* \wedge \neg\varphi'^\frown\varphi) \wedge \varphi')^\frown true)$

The proof system of DC^* is complete for sentences where iteration is allowed only for a restricted class of formulas called *simple*.

Definition 1. Simple DC^* formulas are defined inductively by the following BNF

$$\varphi \stackrel{\wedge}{=} \ell = 0 | \lceil S\rceil | a \leq \ell | \ell \leq a | (\varphi \vee \varphi) | (\varphi \wedge \varphi) | (\varphi^\frown\varphi) | \varphi^*$$

Definition 2. Given a simple DC^* formula φ, we define a simple DC^* formula $PREF(\varphi)$ as follows.

1. $PREF(\lceil S \rceil) \stackrel{\wedge}{=} \lceil S \rceil^*$
2. $PREF(a \leq \ell) \stackrel{\wedge}{=} \ell \geq 0$
3. $PREF(\ell \leq a) \stackrel{\wedge}{=} \ell \leq a$
4. $PREF(\varphi \vee \varphi') \stackrel{\wedge}{=} PREF(\varphi) \vee PREF(\varphi')$
5. $PREF(\varphi \wedge \varphi') \stackrel{\wedge}{=} PREF(\varphi) \wedge PREF(\varphi')$
6. $PREF(\varphi^\frown \varphi') \stackrel{\wedge}{=} PREF(\varphi) \vee (\varphi^\frown PREF(\varphi'))$
7. $PREF(\varphi^*) \stackrel{\wedge}{=} \varphi^{*\frown} PREF(\varphi)$

Intuitively, $PREF(D)$ is a simple formula that holds for all prefixes of an interval that validates D. It follows immediately from the definition that

Proposition 1. $\varphi \Rightarrow \neg(\neg PREF(\varphi)^\frown true)$.

The class of simple DC^* formulas plays an important role in our design process presented in section 5. The following section presents the discretization technique.

3. DISCRETE INTERFACE

A model of real-time control systems is depicted in figure 1. The *plant* denotes the continuous componets of the system. The *controller* is a discrete component denoting a control program executed by a computer. The *sensors* sample the states of the plant. The *actuators* receive commands from the controller and control the plant accordingly. The sensors and the actuators constitute the continuous-to-discrete and discrete-to-continuous interfaces respectively.

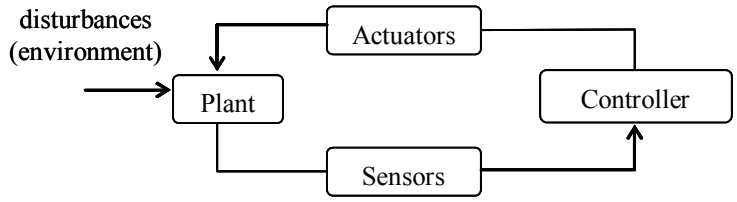


Figure 1. A model of controlled system

In the following part of this section we defined three concepts for formalising the relationship between continuous state variables and discrete ones.

Definition 3. (Stability) Given a state variable s and a positive real number δ, we say s is $\delta - stable$ iff the following formula is satisfied by any interval

$$\delta - stable(s) \stackrel{\wedge}{=} \Box(\lceil \neg s \rceil^\frown \lceil s \rceil^\frown \lceil \neg s \rceil \Rightarrow \lceil \neg s \rceil^\frown (\lceil s \rceil \wedge \ell > \delta)^\frown \lceil \neg S \rceil)$$

The *stability* means that a state should not change quickly in order to be observable at discrete time.

Definition 4. (Control state) Given two state variables r and s, and a non-negative real number δ, we say r $\delta - controls$ s iff the following formula is satisfied by any interval

$$r \rhd_\delta s \stackrel{\wedge}{=} \Box(\lceil r \rceil \wedge \ell > \delta \Rightarrow (\ell \leq \delta)^\frown \lceil s \rceil)$$

The concept of *control state* is used for formalising the behaviour of actuators. Let r be a state variable modeling a program command, and s a state of the plant. Then the relation $r \rhd_\delta s$ means that whenever the controller issues the command r, the plant gets into state s within at most δ time units. So the maximum response time is δ time units.

Definition 5. (Observation state) Given two state variables r and s, and a non-negative real number δ, we say r $\delta - observes$ s iff the following formula is satisfied by any interval.

$$r \stackrel{\rightarrow}{\approx}_\delta s \stackrel{\wedge}{=} (s \rhd_\delta r) \wedge (\neg s \rhd_\delta \neg r)$$

The concept of *observation state* can be used for formalizing the behavior of the sensors. Let r be a state variable modeling a discrete program variable, and s a state of the environment. Then the relation $r \stackrel{\rightarrow}{\approx}_\delta s$ means that any change (stable enough) in s is observed by the controller within δ time units. So the sampling step is δ time units. Note that the definition say nothing about unstable change of s.

We will assume that environment state variables are stable enough to be observable by the controller, otherwise there is no way to observe them in discrete time.

For formalising the discrete interface, for any continuous state variable s, we consider a discrete state variable s_c used by the control program to observe s via the sensors. The relationship between s and its sampling s_c is formalised by $s_c \stackrel{\rightarrow}{\approx}_\delta s$ for some non-negative real number δ. Similarly, for any state t of the plant we consider a command t_c, a discrete state for requesting (via the actuators) the plant getting into state t. The ralationship between t and t_c is formalised by $t_c \rhd_\tau t$ for some non-negative real number τ.

4. REFINEMENT AND VERIFICATION RULES

Some rules given in this section are useful for both the refinement and the verification. The proofs of some rules and more details are given in [7].

Transitivity rules

$$\text{Rule 1} \quad \frac{(r \rhd_\delta s)(s \rhd_\tau t)}{r \rhd_{(\delta+\tau)} t} \qquad \text{Rule 2} \quad \frac{(r \stackrel{\rightarrow}{\approx}_\delta s)(s \stackrel{\rightarrow}{\approx}_\tau t)}{r \stackrel{\rightarrow}{\approx}_{(\delta+\tau)} s}$$

These rules say that the accuracy is deteriorated through sequential samplings of a state. They are helpful for the design of distributed systems comprising many sensors, as well as how to use the sensors efficiently.

Observation rules

Rule 3

$$\frac{(r \stackrel{\rightarrow}{\approx}_\delta s)}{(\lceil s \rceil \wedge \ell > \delta)^\frown(\lceil \neg s \rceil \wedge \ell > \delta) \Rightarrow \ell \leq \delta^\frown \lceil r \rceil^\frown \lceil \neg r \rceil^\frown true}$$

Rule 3 allows to capture the change of state from 0 to 1 or from 1 to 0 by observation.

State Distance

Rule 4a

$$\frac{(r \overrightarrow{\approx}_{\delta} s)\ \delta - stable(s)}{(\delta + \tau) - stable(r) \Rightarrow \tau - stable(s)}$$

Rule 4b

$$\frac{(r \overrightarrow{\approx}_{\delta} s)\ \delta - stable(r)}{(\delta + \tau) - stable(s) \Rightarrow \tau - stable(r)}$$

These rules define a necessary condition for the stability of a continuous state, which is the stability of its sampling and vice-versa. It is useful for refinement.

State Occurrence

Rule 5a

$$\frac{(s \rhd_{\tau} t)}{\Box(\lceil t \rceil \Rightarrow \ell \leq \tau) \Rightarrow \Box(\lceil s \rceil \Rightarrow \ell \leq \delta + \tau)}$$

Rule 5b

$$\frac{(r \overrightarrow{\approx}_{\delta} s)}{\Box(\lceil r \rceil \Rightarrow \ell \leq \tau) \Rightarrow \Box(\lceil s \rceil \Rightarrow \ell \leq \delta + \tau)}$$

These rules are helpful for both refinement and verification. It define how fast the control program should be to satisfy a time constraint about the occurrence of the state.

Duration of state

Rule 6 $PREF(\lceil r \rceil^{*\frown}(\lceil \neg r \rceil \frown (\lceil r \rceil \wedge \ell > \delta))^{*\frown} \lceil \neg r \rceil) \Rightarrow \delta - stable(r)$

Rule 7 $PREF(\lceil \neg r \rceil^{*\frown}((\lceil r \rceil \wedge \ell \leq \delta) \frown \lceil \neg r \rceil)^{*\frown}(\lceil r \rceil \wedge \ell \leq \delta)) \Rightarrow \Box(\lceil r \rceil \Rightarrow \ell \leq \delta)$

Invariant Rule for loop

$$\varphi \Rightarrow \neg(true \frown \neg\alpha)\ \ \ell = 0 \Rightarrow \alpha$$

$$\varphi \Rightarrow \neg(\neg\beta \frown true)\ \ \ell = 0 \Rightarrow \beta$$

Rule 8

$$\frac{\alpha \frown \varphi^{*} \frown \beta \Rightarrow \chi\ \ \varphi \Rightarrow \Box\chi}{\varphi^{*} \Rightarrow \Box\chi}$$

Invariant Rule for sequential concatenation

$$\psi \Rightarrow \Box\chi\ \ \varphi \Rightarrow \Box\chi\ \ \alpha \frown \beta \Rightarrow \chi$$

Rule 9

$$\frac{\psi \Rightarrow \neg(\neg\beta \frown true)\ \varphi \Rightarrow \neg(true \frown \neg\alpha)}{\varphi \frown \psi \Rightarrow \Box\chi}$$

Trivial parallel composition

Rule 10

$$\frac{A \Rightarrow \Box\psi\ B \Rightarrow \Box\varphi}{A \wedge B \Rightarrow \Box(\psi \wedge \varphi)}$$

Monotonicity

Rule 11a $\dfrac{r \rhd_{\tau} s\ \tau \leq \delta}{r \rhd_{\delta} s}$

Rule 11b If $r \Rightarrow s$ then $r \rhd_0 s$

Rule 11c $\dfrac{(r \rhd_{\delta} s)(t \rhd_{\delta} u)}{(r \wedge t) \rhd_{\delta} (s \wedge u)}$

Rule 11d $\dfrac{(r \overrightarrow{\approx}_{\delta} s)(t \overrightarrow{\approx}_{\delta} u)}{(r \wedge t) \overrightarrow{\approx}_{\delta} (s \wedge u)}$

5. DESIGN A SINGLE LIFT CONTROL SYSTEM

5.1 Problem domain description

The logical control of a lift system studied in this paper consists of a simple, single lift system. It allows movement of a single lift cage between a finite number of floors. The starting and stopping of the lift [cage] and the opening and closing of floor doors are made by the pressing of floor call, door close and cage send buttons.

Components: The lift system has the following immediate components: a lift *cage* with send buttons, one for each floor; a *motor*; $N+1$ *floors*, each with a *floor door*, a *call button* and a *close button*; *sensors* and *actuators*; a *controller*.

We identify floors by natural numbers, numbered 0 to N, and assume that the lift can carry any number of clients!

The system state is made up from the above components with their attributes.

Attributes: The system and its components have the following attributes.

+ The fift cage is either stopped at floor j for j lying between 0 and N inclusive, or is moving up (or down) between floors i and $i+1$ (i and $i-1$), for i lying between 0 and $N-1$(N and 1).

+ A floor door is either open or closed.

+ The motor is either running up (or down) or is stopped.

+ The motor, when running, runs at a constant speed-which causes the lift cage to move between immediately neighbouring floors in t_m time units.

Events: We consider only the following events.

+ A send button is pressed for floor k, for $k = 0, ..., N$.

+ A call button on floor k is pressed, for $k = 0, ..., N$.

+ A close button is pressed for the door at floor k, for $k = 0, ..., N$.

+ The opening (and closing) of floor doors.

+ The starting and stopping of the motor-implying the same for the cage.

For the sake of simplicity we do not identify explicitly two journeys of the lift cage: upward one and downward one.

Procedure: A lift journey is procedurally described.

+ *Servicing* a floor k means that a send button is pressed for floor k, or a call button on floor k is pressed, or the lift cage is running upwardly or downwardly (towards floor k).

+ There is a *request* on floor j, means that a call or a send button at floor j is pressed, iff there does not exist any services of floors and the floor door is closed; or a close button at floor j is pressed when the floor door is open. This implies that the lift system services floors succesively. This dogma makes our design simple.

Invariants: The above plus the invariants fully describe expectations.

+ There are at least two floors (a component invariant).

+ The cage has exactly one send button for each floor (a component invariant).

+ Pressing a call button at floor i or pressing a send button for floor i causes the lift to service that floor within t_s time units (a procedural, functional invariant).

+ A floor door may only be open if the lift cage is at that floor (a component safety invariant).
+ The floor door is open for at least t_0 time units and at most $t_{\max}$ time units (a procedural, functional invariant).

The lift system presented in this paper shall be monitored and controlled by a controller that shall respect the components, handle the events, and satisfy the usual procedures and invariants enumerated above.

5.2. Formalizing the requirements of the system

We introduce the following continuous state variables: variable c_i holds if the call button on floor i is pressed, variable s_i holds if the send button for floor i is pressed, variable d_i holds if the door at floor i is open, and variable f_i holds if the lift is at floor i, for i ranges over interval $[0, ..., N]$; variable *motor* hodls if motor is on (and this makes the lift cage move); variable $close_i$ holds if the close button on floor i is pressed (at the time when the door at floor i is open). We do not model lift positions between floors.

The requirements of the system are defined by

$$Req \stackrel{\wedge}{=} \Box(SafetyReq \wedge FunctReq)$$

The safety property for the lift control system is: for every floor, the door must only be opened if the lift is at that floor. This is equivalent to stating that "if the lift is not at floor i, then door i must be closed".

$$SafetyReq \stackrel{\wedge}{=} \lceil d_i \rceil \Rightarrow \lceil f_i \rceil$$

The function requirement is the following conjunction

$$FunctReq \stackrel{\wedge}{=} F_1 \wedge F_2 \wedge F_3$$

Pressing a send button causes the lift to service the corresponding floor within t_s time units.

$$F_1 \stackrel{\wedge}{=} \lceil s_i \rceil^\frown true \Rightarrow \ell \leq t_s \vee (\ell \leq t_s^\frown \lceil d_i \rceil^\frown true)$$

This requirement states that for every observation interval for which s_i hodls initially, i.e. the send button for the $i'th$ floor is pressed, either the interval is shorter than or equal to t_s or it may be diveded into three subintervals where the first lasts at most t_s, in the second the door at floor i is opened, and a final subinterval which is unconstrained.

A similar condition must hold when pressing a call button: pressing a call button causes the lift to service the corresponding floor within t_s time units.

$$F_2 \stackrel{\wedge}{=} \lceil c_i \rceil^\frown true \Rightarrow \ell \leq t_s \vee (\ell \leq t_s^\frown \lceil d_i \rceil^\frown true)$$

The system must guarantee that when a floor is serviced, the door is open for at least t_0 time units and at most $t_{\max}$ time units.

$$F_3 \stackrel{\wedge}{=} (\lceil \neg d_i \rceil^\frown \lceil d_i \rceil^\frown \lceil \neg d_i \rceil \Rightarrow \ell \geq t_0) \wedge (\lceil d_i \rceil \Rightarrow \ell \leq t_{\max})$$

Having defined the requirements, we now present a design decision which implements the requirements.

5.3. Design decision

We define the design decision by the predicate Des

$$Des \stackrel{\wedge}{=} \Box(D1 \wedge D2 \wedge D3 \wedge D4 \wedge D5 \wedge D6 \wedge D7)$$

The following formula is derived directly from the assumptions of the behaviour of the system as described in section 5.1

$$\begin{aligned} D1 \stackrel{\wedge}{=} & ((\ell = a \wedge \lceil s_i \vee c_i \rceil)^\frown(\ell = b \wedge (\lceil \neg d_i \rceil^\frown \lceil d_i \rceil)) \\ & \Rightarrow (\ell = a)^\frown(\ell \leq b \wedge \lceil \neg(s_i \vee c_i \vee s_j \vee c_j)\rceil)^\frown true) \wedge \lceil \neg(close_i \wedge \neg d_i)\rceil \\ & \wedge \lceil \neg(c_i \wedge d_i)\rceil \wedge \lceil \neg(c_i \wedge s_i)\rceil \wedge \lceil \neg(s_i \wedge d_i)\rceil \wedge \lceil \neg(c_i \wedge d_j)\rceil \wedge \lceil \neg(c_i \wedge c_j)\rceil \\ & \wedge \lceil \neg(c_i \wedge s_j)\rceil \wedge \lceil \neg(s_i \wedge s_j)\rceil \wedge \lceil \neg(s_i \wedge d_j)\rceil \end{aligned}$$

If for every interval for which a send button for floor i is pressed initially, and the lift is at floor j and $j \neq i$, then the interval may be divided into three subintervals where the first lasts at most θ time units, in the second the motor is on, and an unconstrained final subinterval; in the condition of $i = j$, then the door at floor i must be opened within θ time units, where θ stands for a response time of the system. A similar condition must hold when pressing a call button at floor i.

$$\begin{aligned} D2 &\stackrel{\wedge}{=} \lceil (s_i \vee c_i) \wedge f_j \rceil^\frown true \Rightarrow (\ell \leq \theta)^\frown \lceil motor \rceil^\frown true \\ D3 &\stackrel{\wedge}{=} \lceil (s_i \vee c_i) \wedge f_i \rceil^\frown true \Rightarrow (\ell \leq \theta)^\frown \lceil d_i \rceil^\frown true \end{aligned}$$

If a send button for floor i is pressed while lift is at floor j, the lift may reach the destination floor and then the motor is off and the door at the floor is opened within θ time units. A similar condition must hold when pressing a call button at floor i.

$$\begin{aligned} D4 \stackrel{\wedge}{=} & \lceil (s_i \vee c_i) \wedge f_j \rceil^\frown true \\ & \Rightarrow (\ell \leq \theta)^\frown \ell = |i-j| t_m^\frown \lceil (\ell \leq \theta^\frown \lceil d_i \rceil) \wedge \lceil f_i \rceil \rceil^\frown true) \wedge \lceil (s_i \vee c_i) \wedge f_j \rceil^\frown true \\ & \Rightarrow (\ell \leq \theta)^\frown \ell = |i-j| t_m^\frown \lceil (\ell \leq \theta^\frown \lceil \neg motor \rceil) \wedge \lceil f_i \rceil \rceil^\frown true) \end{aligned}$$

If the close button at floor i is pressed it may make the door at the floor open within θ time units.

$$D5 \stackrel{\wedge}{=} \lceil close_i \rceil^\frown true \Rightarrow (\ell \leq \theta)^\frown \lceil \neg d_i \rceil^\frown true$$

Two following formulas will help satisfy the requirement of the maximum and minimum time units for which a door is open.

$$\begin{aligned} D6 &\stackrel{\wedge}{=} \lceil \neg d_i \rceil^\frown \lceil d_i \rceil^\frown true \Rightarrow \lceil \neg d_i \rceil^\frown \ell < t_0^\frown \lceil \neg close_i \rceil^\frown true \\ D7 &\stackrel{\wedge}{=} (\lceil d_i \rceil \wedge t_{\max} - 1 \leq \ell \leq t_{\max})^\frown true \Rightarrow \lceil d_i \rceil^\frown \lceil \neg d_i \rceil^\frown true \end{aligned}$$

Inittially the lift is idle at the ground floor with the doors close, motor stops, and no requests for the lift.

$$Init \stackrel{\wedge}{=} \lceil \neg motor \wedge f_0 \wedge \neg d_i \wedge \neg s_i \wedge \neg c_i \wedge \neg close_i \rceil^\frown true \vee \lceil \rceil$$

The maximum time it may take to service a floor corresponds to the time it takes to move across $N+1$ floors and the response time it takes to open doors.

$$A1 \stackrel{\wedge}{=} (t_s \geq (n+1)t_m + 2\theta)$$

The following formula is derived directly from the attribute of the motor as described in section 5.1

$$A2 \stackrel{\wedge}{=} \lceil (s_i \vee c_i) \wedge f_j \rceil^\frown true \Rightarrow \ell \leq \theta^\frown \ell = |i-j| t_m^\frown \lceil f_j \rceil^\frown true$$

We assume that the response time θ is small enough (compare to the speed of the motor) in order to the lift, having reached the destination floor, can be at the floor within at least θ time units while the motor is still running.

$$A3 \stackrel{\wedge}{=} \lceil \neg f_i \rceil^\frown \lceil f_i \wedge motor \rceil^\frown true \Rightarrow \lceil \neg f_i \rceil^\frown \ell \leq \theta \wedge \lceil f_i \rceil^\frown true$$

$$A4 \stackrel{\wedge}{=} (t_0 + \theta + 1 \leq t_{\max})$$

Let $A \stackrel{\wedge}{=} Init \wedge A1 \wedge A2 \wedge A3 \wedge A4$

The following theorem says that the design Des implies the specification Req, under the assumption A.

Theorem 5.3.1. $A \vdash Des \Rightarrow Req$

Proof. See Appendix.

We will find a discrete specification for the controller as follows.

5.4. Discrete design

For any continuous state variable s, let s_c be the discrete state variable used by the controller to observe s via the sensors. Then the relationships between continuous state variables and discrete state ones are formalised as following formulas: $f_{ic} \stackrel{\rightarrow}{\approx}_\delta f_i, c_{ic} \stackrel{\rightarrow}{\approx}_\delta c_i, d_{ic} \stackrel{\rightarrow}{\approx}_\delta d_i, s_{ic} \stackrel{\rightarrow}{\approx}_\delta s_i$, and $close_{ic} \stackrel{\rightarrow}{\approx}_\delta close_i$, where δ is the sampling step.

Let $Dopen_i, Dclose_i, Mon$, and $Moff$ be discrete state variables, which hold when the controller requests the actuators to open the door at floor i, close the door at floor i, start the motor, and stop the motor, respectively. The relationship between them and the continuous state variables d_i and $motor$ are expressed by $Dopen_i \triangleright_\tau d_i, Dclose_i \triangleright_\tau d_i, Mon \triangleright_\tau motor$, and $Moff \triangleright_\tau \neg motor$, where τ stands for the response time of the plant via the actuators. Besides, we also introduce a symbol ℓ as one described above, but its value can be calculated only by some computer clock.

Let

$$\varphi 1 \stackrel{\wedge}{=} (\lceil (c_{ic} \vee s_{ic}) \wedge f_{jc} \wedge Mon \rceil \wedge \ell = \tau)^\frown (|i-j| t_m \leq \ell \leq \delta + |i-j| t_m)^\frown \lceil Moff \rceil$$

$$\varphi 2 \stackrel{\wedge}{=} (\lceil (c_{ic} \vee s_{ic}) \wedge f_{jc} \wedge Mon \rceil \wedge \ell = \tau)^\frown (|i-j| t_m \leq \ell \leq \delta + |i-j| t_m)^\frown \lceil Dopen_i \rceil$$

$$\varphi 3 \stackrel{\wedge}{=} (\lceil (c_{ic} \vee s_{ic}) \wedge f_{ic} \wedge Dopen_i \rceil \wedge (\ell = \tau)$$

$$\phi 1 \stackrel{\wedge}{=} (\ell \geq t_0 - \tau)^\frown \lceil close_{ic} \wedge Dclose_i \rceil$$

$$\phi 2 \stackrel{\wedge}{=} (t_{\max} - \theta - 1 \leq \ell \leq t_{\max} - \theta)^\frown \lceil Dclose_i \rceil$$

$$\mu 1 \stackrel{\wedge}{=} \neg(\lceil(c_{ic} \vee s_{ic}) \wedge \neg f_{ic}\rceil^\frown(\ell \leq \theta + |i-j|t_m)^\frown\lceil c_{ic} \vee s_{ic} \vee c_{jc} \vee s_{jc}\rceil)$$

$$\mu 2 \stackrel{\wedge}{=} \lceil\neg(c_{ic} \wedge d_{jc})\rceil \wedge \lceil\neg(c_{ic} \wedge c_{jc})\rceil \wedge \lceil\neg(c_{ic} \wedge s_{jc})\rceil \wedge \lceil\neg(s_{ic} \wedge s_{jc})\rceil \wedge$$
$$\lceil\neg(s_{ic} \wedge d_{jc})\rceil \wedge \lceil\neg(close_{ic} \wedge \neg d_{ic})\rceil \wedge \lceil\neg(c_{ic} \wedge d_{ic})\rceil \wedge \lceil\neg(c_{ic} \wedge s_{ic})\rceil \wedge \lceil\neg(s_{ic} \wedge d_{ic})\rceil$$

$$\varphi \stackrel{\wedge}{=} \lceil\neg c_{ic} \wedge \neg s_{ic}\rceil^{*\frown}((\varphi 1 \wedge \varphi 2) \vee \varphi 3^\frown\phi 1 \vee \phi 2)^*$$

$$\varphi \stackrel{\wedge}{=} \lceil\neg c_{ic} \wedge \neg s_{ic}\rceil^* \vee \lceil\neg c_{ic} \wedge \neg s_{ic}\rceil^\frown((\varphi 1 \wedge \varphi 2) \vee \varphi 3^\frown\phi 1 \vee \phi 2)^*$$

A discrete design for the controller is difined as following formula

$$Cont \stackrel{\wedge}{=} (\varphi^{*\frown}\phi \vee PREF(\varphi^*)) \wedge \mu 1 \wedge \mu 2$$

Let

$$A_c \stackrel{\wedge}{=} A \wedge (f_{ic} \stackrel{\rightarrow}{\approx}_\delta f_i) \wedge (c_{ic} \stackrel{\rightarrow}{\approx}_\delta c_i) \wedge (d_{ic} \stackrel{\rightarrow}{\approx}_\delta d_i) \wedge (s_{ic} \stackrel{\rightarrow}{\approx}_\delta s_i) \wedge (close_{ic} \stackrel{\rightarrow}{\approx}_\delta close_i) \wedge$$
$$(Dopen_i \rhd_\tau d_i) \wedge (Dclose_i \rhd_\tau \neg d_i) \wedge (Mon \rhd_\tau motor) \wedge (Moff \rhd_\tau \neg\ motor) \wedge$$
$$(\theta \geq \tau + \delta) \wedge (\lceil Dopen_i\rceil \Rightarrow \ell \leq \theta) \wedge (\lceil Dclose_i\rceil \Rightarrow \ell \leq \theta)$$

The following theorem shows the correctness of the discrete design, under the assumption A_c.

Theorem 5.4.1. $A_c \vdash Cont \Rightarrow Des$

Proof. See Appendix.

6. CONCLUSION

We have designed a lift control system by using the technique of modelling discretization at the state level and the approximating continuous state variables by discrete ones. We consider DC^* as specification language to reason about the design of the system. DC^* has been used successfully in many case studies, see e.g. [1, 5, 7], except for our system described above. Using the technique will make our system design move closer to real world compares to one given in [10], and it is useful for programmers to implement it in some programming language. Besides, it is not difficult to prove the correctness of the design of the system by using the technique.

In general, we just have considered the system with simple requirements and the dogmatic assumption. In our future work, we will use the technique to design more complex and practical systems.

Acknowledgment. The authors would like to thank Mr. Dang Van Hung for his useful references.

REFERENCES

[1] Dang Van Hung, Dimitar P. Guelev, Completeness and Decidability of a Fragment of Duration Calculus with Iteration, *Technical Report 163*, UNU/IIST, P.O.Box 3058, Macau, 1999.

[2] Derek N. Dyck, Peter E. Caines, The Logical Control of an Elevator,*IEEE Transactions on Automatic Control* **40** (1995) 480–486.

[3] Doron A.Peled, *Software Reliability Methods*, Springer-Verlag, USA, 2001.

[4] Edward A. Lee, Embedded Software, *Advances In Computers* **56** (2002) 55–90.

[5] Fancois Siewe, Dang Van Hung, Formal Design Technique For Real-Time Embedded Systems, *Technical Report, Science Conference on the Occasion of 25th foundation year*, Institute of Information Technology, Hanoi, Vietnam, 2001.

[6] Francois Siewe, Dang Van Hung, Deriving Real-Time Programs from Duration Calculus Specifications, *Technical Report 222,* UNU/IIST, P.O. Box 3058, Macau, 2000.

[7] Fanõois Siewe, Dang Van Hung, From Continuous Specification to Discrete Design, *Technical Report 182,* UNU/IIST, P.O. Box 3058, Macau, 2002.

[8] J. Shoenfield, *Mathematical Logic,* Addison-Wesley, Massachusetts, 1967.

[9] Jan Vytopil, *Formal Techniques in Real-Time And Fault-Tolerant Systems,* Kluwer Academic Publishers, USA, 1993.

[10] Kirsten Mark Hansen, Angers Peter Ravn, and Hans Rischel, Designing Verified Real-time Systems, *Technical Report,* Dept. of Computer Science, Technical Uni. of Denmark, EuroMicro '92, Paris, 1992.

[11] Michael Schiebe, *Saskia Pferrer, Real-Time Systems Engineering And Applications,* Kluwer Academic Publishers, USA, 1992.

[12] Michael R. Hansen, Zhou Chaochen, Duration Calculus: Logical Foundations, *Formal Aspects of Computing* **9** (1997) 283–330.

[13] Rajesh Kumar Gupta, *Co-synthesis of Hardware And Software For Digital Embedded Systems*, Kluwer Academic Publishers, USA, 1995.

[14] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, Design of Embedded Systems: Formal Models, Validation, and Synthesis, *Proceedings of the IEEE,* **85** (1997).

Appendix

A Proof of Theorem 5.3.1

We shall prove that $A \vdash Des \Rightarrow Req$.
Observation:

+ If $T \vdash A \Rightarrow B$ and $T \vdash C \Rightarrow D$ then $T \vdash A \wedge C \Rightarrow B \wedge D$
+ $A \vdash \Box B \Rightarrow C \Leftrightarrow A, B \vdash C$ (Deduction theorem).
+ $\Box(\phi \Rightarrow \varphi) \Rightarrow (\Box\phi \Rightarrow \Box\varphi)$

Therefore, we can complete the above theorem if we can prove four following theorems.

Theorem 5.3.1.a $A, D1 \wedge D2 \wedge D3 \wedge D4 \vdash \lceil d_i \rceil \Rightarrow \lceil f_i \rceil$
For every interval at which d_i holds, there always exists a smallest interval that contains that one and it makes only c_i, or only s_i hold initially. Assume that c_i holds initially, we will give the brief proof of the theorem as follows.

1. $\lceil c_i \rceil^\frown \lceil \neg d_i \rceil^\frown \lceil d_i \rceil$ $\{assumption\}$
2. $\lceil c_i \rceil^\frown true$ $\{1, DC^*\}$
3. $\lceil c_i \rceil^\frown true \Rightarrow (\lceil c_i \wedge f_i \rceil^\frown true) \wedge (\lceil c_i \wedge \neg f_i \rceil^\frown true) \vee (\lceil c_i \wedge f_i \rceil^\frown true) \wedge (\lceil c_i \wedge \neg f_i \rceil^\frown true)$ $\{D1, DC*\}$

4. $(\lceil c_i \wedge f_i \rceil^\frown true) \wedge (\lceil c_i \wedge \neg f_i \rceil^\frown true)$
$\Rightarrow (\ell \leq \theta)^\frown(\ell = |i-j|t_m)^\frown\lceil(\ell \leq \theta^\frown\lceil d_i\rceil \wedge \lceil f_i\rceil)\rceil^\frown true \wedge tt$ $\{D2, D3, D4, A\}$
$\Rightarrow (\ell \leq \theta)^\frown(\ell = |i-j|t_m)^\frown\lceil(\ell \leq \theta^\frown\lceil d_i\rceil) \wedge \lceil f_i\rceil\rceil^\frown true$ $\{DC^*\}$
5. $(\lceil c_i \wedge f_i \rceil^\frown true) \wedge (\lceil c_i \wedge \neg f_i \rceil^\frown true)$
$\Rightarrow (\ell \leq \theta^\frown\lceil d_i\rceil) \wedge \lceil f_i\rceil^\frown true$ $\{D2, D4, D3, A3, DC^*\}$
6. $\lceil c_i\rceil^\frown true \Rightarrow (\ell \leq \theta^\frown\lceil d_i\rceil) \wedge \lceil f_i\rceil^\frown true \vee$
$(|i-j|t_m) \leq \ell \leq \theta + |i-j|t_m^\frown\lceil(\ell \leq \theta^\frown\lceil d_i\rceil) \wedge \lceil f_i\rceil\rceil^\frown true$ $\{2, 3, 4, 5\}$
7. $\lceil d_i\rceil \Rightarrow \lceil f_i\rceil$ $\{1, 2, 6, DC^*\}$
We can prove the theorem if si holds in the same way.

Theorem 5.3.1.b $A, D1 \wedge D2 \wedge D3 \wedge D4 \wedge \lceil s_i\rceil^\frown true \vdash \ell \leq t_s \vee (\ell \leq t_s^\frown\lceil d_i\rceil^\frown true)$
For every observation interval for which s_i holds initially and the interval is longer than t_s, we present the brief proof of the theorem as follows.
1. $\lceil s_i\rceil^\frown true$ $\{hypothesis\}$
2. $\lceil s_i\rceil^\frown true \Rightarrow (\lceil s_i \wedge f_i\rceil^\frown true) \vee (\lceil s_i \wedge \neg f_i\rceil^\frown true)$ $\{DC^*\}$
3. $\lceil s_i \wedge f_i\rceil^\frown true \Rightarrow \ell \leq \theta^\frown\lceil d_i\rceil^\frown true$ $\{D3\}$
4. $\lceil(s_i \vee c_i) \wedge f_i\rceil^\frown true \Rightarrow (\ell \leq \theta)^\frown\lceil motor\rceil^\frown true$ $\{D2\}$
$\Rightarrow (\ell \leq \theta)^\frown(\ell = |i-j|t_m)^\frown\ell \leq \theta^\frown\lceil d_i\rceil^\frown true$ $\{A2, D4\}$
$\Rightarrow (\ell \leq t_s)^\frown\lceil d_i\rceil^\frown true$ $\{A1, Arithmetic\}$
5. $(\ell \leq \theta)^\frown\lceil d_i\rceil^\frown true \vee (\ell \leq t_s^\frown\lceil d_i\rceil^\frown true)$ $\{1, 2, 3, 4, DC^*\}$
6. $\ell \leq t_s^\frown\lceil d_i\rceil^\frown true$ $\{5, A1, DC^*\}$

Theorem 5.3.1.c $A, D1 \wedge D2 \wedge D3 \wedge D4 \wedge \lceil c_i\rceil^\frown true \vdash \ell \leq t_s \vee (\ell \leq t_s^\frown\lceil d_i\rceil^\frown true)$
The proof of this theorem can be induced from that of theorem 5.3.1.b.

Theorem 5.3.1.d $A, D7 \wedge D6 \wedge D5 \vdash F3$
1. $\lceil\neg d_i\rceil^\frown\lceil d_i\rceil^\frown true \Rightarrow \lceil\neg d_i\rceil^\frown\ell < t_0^\frown\lceil\neg close_i\rceil^\frown true$ $\{D6\} \Rightarrow \lceil\neg d_i\rceil^\frown(\ell < t_0^\frown\ell \leq \theta) \wedge \lceil d_i\rceil^\frown true$ $\{D5\}$
2. $(\lceil\neg d_i\rceil^\frown\lceil d_i\rceil^\frown\lceil\neg d_i\rceil) \wedge (\ell \leq t_0) \Rightarrow$ $ff\{1, DC^*\}$
3. $\lceil d_i\rceil \Rightarrow \ell \leq t_{\max}$ $\{D7 and DC^*\}$
4. $F3$ $\{A, 2, 3, obs.\text{and}DC^*\}$

B. Proof of Theorem 5.4.1
We shall prove that $A_c \vdash Cont \Rightarrow Des$
Observation:

$$(\varphi \vee \phi^*)^* \Leftrightarrow (\varphi^*{}^\frown\phi^*)^*$$
$$A \vdash B \Rightarrow C \Leftrightarrow A, B \vdash C$$

Because of the space litmit, we don' t give the complete proof of the theorem and we just prove the following theorem.
Theorem $A_c \vdash Cont \Rightarrow D7$

1. $\varphi^*{}^\frown\phi$ {hypothesis}
2. $(\lceil\neg c_{ic}\wedge\neg s_{ic}\rceil^*{}^\frown((\varphi1\wedge\varphi2)\vee\varphi3{}^\frown\phi1\vee\phi2)^*)^*{}^\frown(\lceil\neg c_{ic}\wedge\neg s_{ic}\rceil^*\vee\lceil\neg c_{ic}\wedge\neg s_{ic}\rceil{}^\frown((\varphi1\wedge\varphi2)\vee\varphi3{}^\frown\phi1\vee\phi2)^*)^*$ $\{1, DC^*\}$
3. $(\lceil\neg c_{ic}\wedge\neg s_{ic}\rceil\vee((\varphi1\wedge\varphi2)\vee\varphi3{}^\frown\phi1\vee\phi2))^*{}^\frown(\lceil\neg c_{ic}\wedge\neg s_{ic}\rceil^*\vee\lceil\neg c_{ic}\wedge\neg s_{ic}\rceil{}^\frown((\varphi1\wedge\varphi2)\vee\varphi3{}^\frown\phi1\vee\phi2)^*)^*$ $\{2, obs^*\}$
4. $(\varphi1\wedge\varphi2)\vee\varphi3{}^\frown\phi1\vee\phi2$
$\Rightarrow\lceil(c_{ic}\vee s_{ic})\wedge f_{jc}\wedge Mon\rceil\wedge\ell=\tau{}^\frown(|i-j|t_m\leq\ell\leq\delta+|i-j|t_m){}^\frown(\lceil Dopen_i\rceil)$
$\vee\lceil(c_{ic}\vee s_{ic})\wedge f_{ic}\wedge Dopen_i\rceil\wedge\ell=\tau{}^\frown((\ell\geq t_0-\tau){}^\frown\lceil close_{ic}\wedge Dclose_i\rceil)$
$\vee(t_{\max}-\theta-1\leq\ell\leq t_{\max}-\theta){}^\frown\lceil Dclose_i\rceil$ $\{DC^*\}$
5. $\lceil Dopen_i\rceil\Rightarrow\ell\leq\tau{}^\frown\lceil d_i\rceil$ $\{A_c, Def.4, A_c\}$
6. $\lceil Dclose_i\rceil\Rightarrow\ell\leq\tau{}^\frown\lceil\neg d_i\rceil$ $\{A_c, Def.4, A_c\}$
7. $\lceil d_i\rceil\wedge(t_{\max}-1\leq\ell\leq t_{\max}){}^\frown true$ $\{ass.\}$
8. $(\varphi1\wedge\varphi2)\vee\varphi3{}^\frown\phi1\vee\phi2$
$\Rightarrow(\lceil(c_{ic}\vee s_{ic})\wedge f_{jc}\wedge Mon\rceil\wedge\ell=\tau{}^\frown(|i-j|t_m\leq\ell\leq\delta+|i-j|t_m){}^\frown\ell\leq\tau{}^\frown(t_{\max}-1\leq\ell\leq t_{\max})\wedge(\lceil d_i\rceil)\vee(\ell\leq\tau{}^\frown(t_{\max}-1\leq\ell\leq t_{\max})\wedge\lceil d_i\rceil){}^\frown\lceil\neg d_i\rceil\wedge(\ell\leq\delta)$ $\{4,5,6,7, A_c, Rule5a, Arithmetic\}$
9. $(\varphi1\wedge\varphi2)\vee\varphi3{}^\frown\phi1\vee\phi2$
$\Rightarrow(\lceil d_i\rceil\wedge(t_{\max}-1\leq\ell\leq t_{\max}){}^\frown true\Rightarrow(\ell\geq t_{\max}-1\wedge\ell\leq t_{\max}){}^\frown\lceil\neg d_i\rceil{}^\frown true)$ $\{8, DC^*\}$
10. $\ell=0\Rightarrow\Box D7$ $\{DC^*\}$
11. $(\lceil\neg c_{ic}\wedge\neg s_{ic}\rceil\vee\lceil\neg c_{ic}\wedge\neg s_{ic}\rceil^*\Rightarrow\ D7\{10, A_c, DC^*\}$
12. $A_c, ((\varphi1\wedge\varphi2)\vee\varphi3{}^\frown\phi1\vee\phi2)^*\wedge\mu1\wedge\mu2, (\lceil d_i\rceil\wedge t_{\max}-1\leq\ell\leq t_{\max}){}^\frown true\vdash\lceil d_i\rceil{}^\frown\lceil\neg d_i\rceil{}^\frown true$ $\{9,10,11, DC^*, obs.\}$
13. $A_c, (\lceil\neg c_{ic}\wedge\neg s_{ic}\rceil\vee\lceil\neg c_{ic}\wedge\neg s_{ic}\rceil^*)\wedge\mu1\wedge\mu2, (\lceil d_i\rceil\wedge t_{\max}-1\leq\ell\leq t_{\max}){}^\frown true\vdash\lceil d_i\rceil{}^\frown\lceil\neg d_i\rceil{}^\frown true$ $\{11, DC^*, obs.\}$
14. $A_c, (\varphi^*{}^\frown\phi)\wedge\mu1\wedge\mu2, (\lceil d_i\rceil\wedge t_{\max}-1\leq\ell\leq t_{\max}){}^\frown true\vdash\lceil d_i\rceil{}^\frown\lceil\neg d_i\rceil{}^\frown true$ $\{1,2,3,10,12,13, obs., Rule9\}$
15. $PREF(\varphi^*)$ $\{hypothesis\}$
16. $(\lceil\neg c_{ic}\wedge\neg s_{ic}\rceil^*{}^\frown((\varphi1\wedge\varphi2)\vee\varphi3{}^\frown\phi1\vee\phi2)^*)^*{}^\frown((\ell=0\vee\lceil\neg c_{ic}\wedge\neg s_{ic}\rceil)\vee\lceil\neg c_{ic}\wedge\neg s_{ic}\rceil^*{}^\frown(\varphi1\wedge\varphi2)\vee\varphi3{}^\frown\phi1\vee\phi2)^*$ $\{13, Def.of PREF()\}$
17. $A_c, PREF(\varphi^*)\wedge\mu1\wedge\mu2, (\lceil d_i\rceil\wedge t_{\max}-1\leq\ell\leq t_{\max}){}^\frown true\vdash\lceil d_i\rceil{}^\frown\lceil\neg d_i\rceil{}^\frown true$ $\{10,12,13, DC^*, obs.\}$
18. $A_c\vdash Cont\Rightarrow D7$ $\{14,17, obs.\}$

Nhận bài ngày 11 - 8 - 2003

THIẾT KẾ CÁC HỆ THỐNG ĐIỀU KHIỂN SỐ SỬ DỤNG VI ĐIỀU KHIỂN (MICROCONTROLLER) VÀ MÁY TÍNH CÁ NHÂN (PC)

Nguyễn Thanh Sơn
Bộ môn Thiết bị điện-điện tử, Khoa Điện, Đại học Bách khoa Hà Nội

***Tóm tắt*-Điều khiển số là một nhánh của lý thuyết điều khiển gắn liền với việc sử dụng các máy tính số. Tùy theo mức độ và yêu cầu điều khiển, một hệ thống điều khiển số có thể được xây dựng từ các vi điều khiển hoặc kết hợp giữa vi điều khiển với máy tính cá nhân. Bài báo này trình bày các bước thiết kế một hệ thống điều khiển số bằng cách kết hợp giữa vi điều khiển và máy tính cá nhân. Hệ thống điều khiển bao gồm phần cứng được xây dựng từ các vi điều khiển thông dụng giá rẻ AT89S51 và phần mềm được lập trình bằng ngôn ngữ Visual Basic. Với giao người sử dụng bằng đồ họa, người sử dụng có thể dễ dàng thay đổi các thông số của hệ thống điều khiển. Hy vọng bài báo sẽ là nguồn tham khảo hữu ích cho sinh viên chuyên ngành Thiết bị điện-điện tử, Khoa Điện, Đại học Bách khoa Hà Nội trong việc thiết thiết kế các hệ thống điều khiển số quy mô vừa và nhỏ.**

***Từ khóa*-Điều khiển số, vi điều khiển AT89S51, Visual Basic.**

I. Giới thiệu

Trong mười năm qua, nhờ giá thành thấp và độ tin cậy cao nên các máy tính số đã được sử dụng rộng rãi trong nhiều hệ thống điều khiển. Hiện tại, trên thế giới có khoảng 100 triệu hệ thống điều khiển số sử dụng máy tính. Nếu chỉ tính riêng các hệ thống điều khiển phức tạp như điều khiển trong ngành hàng không thì có khoảng có khoảng 20 triệu hệ thống điều khiển bằng máy tính [1].

Chúng ta có thể gặp các hệ thống điều khiển số trong nhiều ứng dụng như điều khiển quá trình, điều khiển giao thông, điều khiển máy bay, điều khiển rada, máy công cụ,... Ưu điểm của các hệ thống điều khiển số là độ chính xác cao và tính khả trình linh hoạt của chúng. Cụ thể, các thuật toán điều khiển dễ dàng được xây dựng và sửa đổi nhờ các công cụ chuyên dụng để lập trình cho các phần cứng.

Vi điều khiển AT89S51 là vi điều khiển 8 bit với bộ nhớ chớp nhoáng khả trình trong hệ thống của hãng Atmel với dung lượng bộ nhớ 4 Kbytes. Vi điều khiển này được sản xuất sử dụng công nghệ lưu trữ thông tin không mất mát (non-volatile memory). Vi điều khiển AT89S51 tương thích với tập lệnh chuẩn công nghiệp và các chân ra của họ vi điều khiển 80C51. Với tổ hợp trong một chip của bộ xử lý trung tâm 8 bit và bộ nhớ chớp nhoáng, vi điều khiển AT89S51 thực sự là một bộ vi điều khiển mạnh, linh hoạt và kinh tế cho hàng loạt ứng dụng điều khiển số quy mô vừa và nhỏ.

Ngôn ngữ lập trình Visual Basic là một ngôn ngữ lập trình bậc cao theo luồng các sự kiện của hãng Microsoft. Ngôn ngữ lập trình này được bắt nguồn từ ngôn ngữ Basic và cho phép người sử dụng phát triển nhanh các ứng dụng của giao diện người sử dụng đồ họa, truy cập vào các cơ sở dữ liệu, các điều khiển ActiveX,...Do đó, trong bài báo này Visual Basic được chọn để lập trình các phần mềm điều khiển với giao diện tiện lợi cho quá trình thay đổi các tham số của hệ thống điều khiển.

Để giúp sinh viện chuyên ngành Thiết bị điện-điện tử Khoa Điện, Đại học Bách khoa Hà Nội có thể hiểu tường tận và thiết kế được các hệ thống điều khiển số quy mô vừa và nhỏ, tập thể các cán bộ trong nhóm Điều khiển của bộ môn Thiết bị điện-điện tử đã dành thời gian tổng hợp lý thuyết về điều khiển số, xây dựng các hệ thống điều khiển số sử dụng máy tính cá nhân và vi điều khiển AT89S51 để điều khiển các thiết bị điện phổ cập như động cơ điện, máy phát điện,...Nội dung của bài báo được trình bày với kết hợp giữa lý thuyết với thực hành ở mức độ đơn giản phù hợp với kiến thức của sinh viên chuyên ngành Thiết bị điện-điện tử ở các năm cuối đã được trang bị các kiến thức như Điều khiển số, Kỹ thuật vi xử lý, Điện tử công suất. Bài báo được bố cục với các phần sau:

-Phần 2 của bài báo giới thiệu vắn tắt về các hệ thống điều khiển số và biến đổi z.

-Phần 3 giới thiệu về cách xác định hàm truyền của một số bộ điều khiển số thông dụng. Cụ thể, phần này giới thiệu về việc xác định hàm truyền của bộ điều khiển "dead-beat" và bộ điều khiển Dahlin.

-Phần 4 giới thiệu về nguyên tắc chuyển các hàm truyền của bộ điều khiển số ở dạng biến đổi z sang dạng phù hợp với quá trình thực thi bằng máy tính số. Cụ thể là các hệ thống có hàm truyền bậc nhất, bậc hai và bộ điều khiển tỷ lệ-tích phân-vi phân (PID) được biểu diễn ở dạng lấy mẫu tại các thời điểm khác nhau.

-Phần 5 trình bày các bước để thiết kế các mạch vào ra giao tiếp với máy tính sử dụng vi điều khiển AT89S51, phần mềm điều khiển xây dựng bằng ngôn ngữ Visual Basic.

-Phần 6 là kết luận với các hướng phát triển tiếp theo trong tương lai.

Ngoài ra bài báo còn bao gồm các phụ lục cần thiết cho việc tham khảo để thiết kế phần cứng và xây dựng các phần mềm điều khiển bằng máy tính.

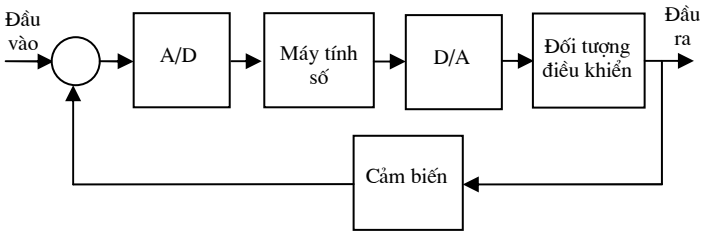
II. Điều khiển số và biến đổi z

Các hệ thống điều khiển số hay còn được gọi là các hệ thống điều khiển với tín hiệu được lấy mẫu với sơ đồ khối như trên hình 1 được xây dựng từ các phần tử sau:

-Bộ chuyển đổi tương tự sang số (A/D converter): làm nhiệm vụ chuyển đổi tín hiệu phản hồi ở dạng liên tục sang dạng số để thuận tiện cho việc xử lý bằng máy tính số.

-Máy tính số: chứa chương trình điều khiển chính.

-Bộ chuyển đổi số sang tương tự (D/A converter): làm nhiệm vụ chuyển tín hiệu số đầu ra của máy tính sang dạng liên tục điều khiển các mạch chấp hành để đóng mở các van bán dẫn như tiristo, triac hay tranzito.



Hình 1: Sơ đồ khối một hệ thống điều khiển số.

Trong điều khiển số, quá trình lấy mẫu có thể được mô tả như là quá trình đóng cắt của một công tắc sau mỗi chu kỳ T được tính bằng giây. Tập hợp của tất các tín hiệu lấy mẫu từ tín hiệu liên tục $r(t)$ được mô tả bằng công thức sau:

$$r^*(t) = \sum_{n=0}^{\infty} r(nT)\delta(t-nT) \quad (1)$$

Trong công thức (1), $r(nT)$ là biên độ của tín hiệu lấy mẫu tại chu kỳ thứ n, $\delta(t-nT)$ là xung đơn vị tại chu kỳ thứ n. Biến đổi Laplace phương trình (1) ta có:

$$R^*(p) = \sum_{n=0}^{\infty} r(nT)e^{-pnT} \quad (2)$$

Phương trình (2) được gọi là phương trình trong mặt phẳng p của tín hiệu được lấy mẫu $r(t)$. Đồng thời phương trình (2) còn được xem như là một chuỗi vô tận của các lũy thừa e^{-pnT}.

Trong lý thuyết điều khiển số, biến đổi z được định nghĩa như sau:

$$z = e^{pT} \quad (3)$$

Biến đổi z của phương trình (2) được ký hiệu $Z\left[r(t)\right] = R(z)$ và được xác định như sau:

$$R(z) = \sum_{n=0}^{\infty} r(nT)z^{-n} \quad (4)$$

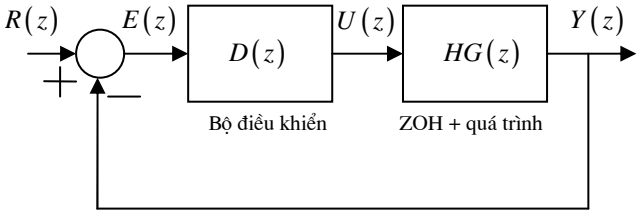
Từ phương trình (4) ta thấy, biến đổi z bao gồm một chuỗi các biến z. Mặt khác, phương trình (4) có thể được viết lại như sau:

$$R(z) = r(0) + r(T)z^{-1} + r(2T)z^{-2} + r(3T)z^{-3} + \ldots \quad (5)$$

Trong đó $r(nT)$ là các hệ số của chuỗi lũy thừa tại các các thời điểm lấy mẫu khác nhau.

III. Các bộ điều khiển số

Một cách tổng quát, chúng ta có thể sử dụng sơ đồ khối như hình 2 khi thiết kế một bộ điều khiển số. Trong đó, $R(z)$ là đầu vào tham chiếu hay còn gọi là giá đặt, $E(z)$ là tín hiệu sai lệch giữa tín hiệu đặt và tín hiệu phản hồi, $U(z)$ là đầu ra của bộ điều khiển cần được thiết kế và $Y(z)$ là đầu ra của hệ thống. $HG(z)$ đặc trưng cho hàm truyền của đối tượng điều khiển đã được số hóa kết hợp với giữ mẫu bậc không.



Hình 2: Hệ thống điều khiển thời gian rời rạc.

Hàm truyền của hệ kín như trên hình 2 có thể được viết như sau:

$$\frac{Y(z)}{R(z)} = \frac{D(z)HG(z)}{1+D(z)HG(z)} \quad (6)$$

Chúng ta ký hiệu hàm truyền của hệ kín là $T(z)$. Do đó ta có:

$$T(z) = \frac{Y(z)}{R(z)} \quad (7)$$

Từ phương trình (6) và (7) ta xác định được hàm truyền của bộ điều khiển cần phải được thiết kế như sau:

$$D(z) = \frac{1}{HG(z)}\left[\frac{T(z)}{1-T(z)}\right] \quad (8)$$

Phương trình (8) có nghĩa là hàm truyền của bộ điều khiển có thể xác định được nếu chúng ta biết mô hình hay hàm truyền của quá trình. Bộ điều khiển $D(z)$ phải được thiết kế sao cho hệ là ổn định và có thể thực thi bằng các phần cứng. Sau đây chúng ta sẽ quan khảo sát hai bộ điều khiển số được thiết kế theo phương trình (8). Đó là bộ điều khiển "dead-beat" và bộ điều khiển Dahlin.

a) Bộ điều khiển "dead-beat"

Bộ điều khiển "dead-beat" là một bộ điều khiển mà tín hiệu đầu ra có dạng nhảy cấp giống như tín hiệu đầu vào nhưng trễ so với đầu vào một hoặc vài chu kỳ lấy mẫu. Hàm truyền của hệ kín khi đó sẽ là:

$$T(z) = z^{-k} \qquad k \geq 1 \quad (9)$$

Từ phương trình (8), hàm truyền của bộ điều khiển cần được thiết kế là:

$$D(z) = \frac{1}{HG(z)}\left[\frac{z^{-k}}{1-z^{-k}}\right] \quad (10)$$

Ví dụ chúng ta cần thiết kế bộ điều khiển cho một hệ thống với đối tượng điều khiển có hàm truyền như sau:

$$G(p)=\frac{e^{-2p}}{1+10p}$$

Hàm truyền của hệ kín với giữ mẫu bậc không được xác định như sau:

$$HG(z)=Z\left\{\frac{1-e^{-pT}}{p}G(p)\right\}=\left(1-z^{-1}\right)Z\left\{\frac{e^{-2p}}{p(1+10p)}\right\}$$

Giả thiết chu kỳ lấy mẫu T= 1 giây ta có:

$$HG(z)=\left(1-z^{-1}\right)z^{-2}Z\left\{\frac{1/10}{p(1/10+p)}\right\}$$

$$HG(z)=\left(1-z^{-1}\right)z^{-2}\frac{z\left(1-e^{-0,1}\right)}{(z-1)\left(z-e^{-0,1}\right)}=z^{-3}\frac{\left(1-e^{-0,1}\right)}{1-e^{-0,1}z^{-1}}$$

$$HG(z)=\frac{0,095z^{-3}}{1-0,904z^{-1}}$$

Do đó ta có:

$$D(z)=\frac{1-0,904z^{-1}}{0,095z^{-3}}\left[\frac{z^{-k}}{1-z^{-k}}\right]$$

Giả thiết $k\geq 3$ ta có:

$$D(z)=\frac{1-0,904z^{-1}}{0,095z^{-3}}\left[\frac{z^{-3}}{1-z^{-3}}\right]=\frac{z^3-0,904z^2}{0,095\left(z^3-1\right)}$$

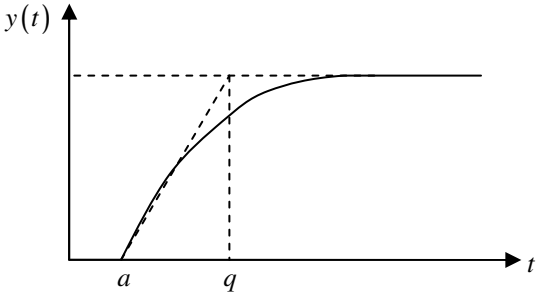
a) Bộ điều khiển Dahlin

Bộ điều khiển Dahlin là sự biến cải của bộ điều khiển "dead-beat" và tạo nên phản ứng theo hàm mũ trơn hơn phản ứng của bộ điều khiển "dead-beat".

Phản ứng yêu cầu của hệ thống trong mặt phẳng p có thể được viết như sau:

$$Y(p)=\frac{1}{p}\left(\frac{e^{-ap}}{1+pq}\right) \qquad (11)$$

Trong đó a và q được chọn để đạt được phản ứng theo mong muốn như trên hình 3.



Hình 3: Phản ứng đầu ra của bộ điều khiển Dahlin.

Dạng tổng quát của hàm truyền của bộ điều khiển Dahlin là [1]:

$$D(z)=\frac{1}{HG(z)}\frac{z^{-k-1}\left(1-e^{-T/q}\right)}{1-e^{-T/q}z^{-1}-\left(1-e^{-T/q}z^{-1}\right)z^{-k-1}} \qquad (12)$$

Ví dụ thiết kế bộ điều khiển Dahlin cho một hệ thống với với thời gian lấy mẫu T=1 giây và đối tượng điều khiển có hàm truyền như sau:

$$G(p)=\frac{e^{-2p}}{1+10p}$$

Như đã trình bày trong ví dụ trên hàm truyền của hệ đối tượng điều khiển với giữ mẫu bậc không có dạng như sau:

$$HG(z)=\frac{0,095z^{-3}}{1-0,904z^{-1}}$$

Giả thiết ta chọn $q=10$, khi đó hàm truyền của bộ điều khiển sẽ có dạng như sau:

$$D(z)=\frac{1}{HG(z)}\left[\frac{T(z)}{1-T(z)}\right]$$
$$=\frac{1-0,904z^{-1}}{0,095z^{-3}}\frac{z^{-k-1}\left(1-e^{-0,1}\right)}{1-e^{-0,1}z^{-1}-\left(1-e^{-0,1}z^{-1}\right)z^{-k-1}}$$

$$D(z)=\frac{1-0,904z^{-1}}{0,095z^{-3}}\frac{0,095z^{-k-1}}{1-0,904z^{-1}-0,095z^{-k-1}}$$

Giả sử ta chọn $k=2$ ta có:

$$D(z)=\frac{0,095z^3-0,0858z^2}{0,095z^3-0,0858z^2-0,0090}$$

Tóm lại, với giả thiết là các hàm truyền của đối tượng điều khiển đã biết trước, chúng ta có thể dễ dàng xây dựng được các hàm truyền của các bộ điều khiển theo vòng kín. Tuy nhiên trong thực tế, việc thiết lập được mô hình chính xác của các đối tượng điều khiển là hết sức khó khăn. Do đó chúng ta sẽ xét đến bộ điều khiển tỷ lệ-tích phân-vi phân hay còn được gọi là các bộ điều khiển PID được sử dụng phổ biến trong công nghiệp ở phần tiếp theo.

IV. Thực thi các bộ điều khiển số

Các thuật toán điều khiển số ở dạng biến đổi z cần thiết phải được chuyển sang dạng phương trình phù hợp để thực thi với các phần cứng hay máy tính cá nhân. Một hàm truyền của một bộ điều khiển số ở dạng biến đổi z có thể được thực thi bằng nhiều phương pháp khác nhau. Về mặt toán học các phương pháp này là tương đương. Tuy nhiên, các phương pháp khác nhau sẽ có các hệ số tính toán khác nhau, độ nhạy khác nhau đối với tín hiệu sai lệch và cách lập trình khác nhau. Phần này sẽ trình bày các bước để thực thi các bộ điều khiển số theo phương pháp sơ đồ song song.

Hàm truyền của một bộ điều khiển số có thể được biểu diễn ở dạng tổng của hàm truyền bậc nhất và hàm truyền bậc hai như sau:

$$D(z) = \alpha_0 + D_1(z) + D_2(z) \tag{13}$$

Trong đó hàm truyền bậc nhất có dạng như sau:

$$D_1(z) = \frac{\alpha}{1+\beta z^{-1}} = \frac{\alpha R(z)}{E(z)} \tag{14}$$

Trong đó

$$\frac{R(z)}{E(z)} = \frac{1}{1+\beta z^{-1}} \tag{15}$$

Từ phương trình (15) ta có xác định được $R(z)$ có dạng như sau:

$$R(z) = E(z) - \beta R(z) z^{-1} \tag{16}$$

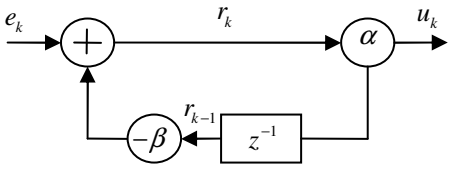
Trong điều khiển số z^{-1} chính là phần tử trễ đơn vị hay là trễ sau một chu kỳ lấy mẫu. Do đó từ công thức (16) ta có thể biểu diễn các giá trị $R(z)$ và $E(z)$ ở dạng lấy mẫu tại các thời điểm lấy mẫu k khác nhau như sau:

$$r_k = e_k - \beta r_{k-1} \tag{17}$$

Trong đó r_k là giá trị của $r(t)$ tại thời điểm lấy mẫu thứ k, r_{k_1} là giá trị của $r(t)$ tại thời điểm lấy mẫu chậm sau thời điểm lấy mẫu k một chu kỳ. Cuối cùng, e_k là giá trị của $e(t)$ tại thời điểm lấy mẫu k. Tín hiệu đầu ra điều khiển u_k được tính như sau:

$$u_k = \alpha(e_k - \beta r_{k-1}) \tag{18}$$

Phương trình (18) có thể biểu diễn bằng sơ đồ như trên hình 4. Sơ đồ này được gọi là sơ đồ song song.



Hình 4: Thực thi hàm truyền bậc nhất theo sơ đồ song song.

Hàm truyền bậc hai có dạng như sau:

$$D_2(z) = \frac{a_0 + a_1 z^{-1}}{1 + b_1 z^{-1} + b_2 z^{-2}} = \frac{U(z)}{E(z)} \tag{19}$$

Hay

$$U(z) = a_0 R(z) + a_1 z^{-1} R(z) \tag{20}$$

Trong đó

$$R(z) = \left(\frac{1}{1 + b_1 z^{-1} + b_2 z^{-2}}\right) E(z) \tag{21}$$

Phương trình (20) là đầu ra của hàm truyền bậc hai ở dạng biến đổi z. Ở dạng lấy mẫu tại các thời điểm k khác nhau ta có thể viết lại phương trình (20) như sau:

$$u_k = a_0 r_k + a_1 r_{k-1} \tag{22}$$

Trong đó u_k là giá trị đầu ra $u(t)$ của hàm truyền tại thời điểm lấy mẫu thứ k, r_k là giá trị của $r(t)$ tại thời điểm lấy mẫu thứ k, r_{k-1} là giá trị của $r(t)$ tại thời điểm lấy mẫu chậm sau thời điểm lấy mẫu thứ k một chu kỳ.

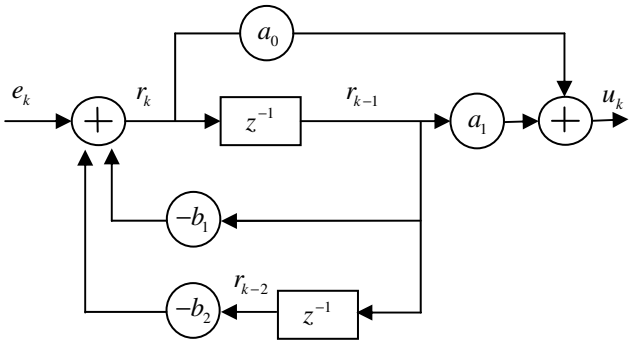
Mặt khác, phương trình (21) có thể được viết lại như sau:

$$R(z) = E(z) - b_1 z^{-1} R(z) - b_2 z^{-2} R(z) \tag{23}$$

Phương trình (23) là phương trình ở dạng biến đổi z. Phương trình (23) có thể biểu diễn ở dạng lấy mẫu tại các thời điểm k khác nhau như sau:

$$r_k = e_k - b_1 r_{k-1} - b_2 r_{k-2} \tag{24}$$

Trong đó r_{k-2} là giá trị của $r(t)$ tại thời điểm lấy mẫu chậm sau thời điểm lấy mẫu thứ k hai chu kỳ và e_k là giá trị của $e(t)$ tại thời điểm lấy mẫu thứ k.



Hình 5: Thực thi hàm truyền bậc hai theo sơ đồ song song.

Sau khi đã làm quen được với các thao tác chuyển các hàm truyền đơn giản ở dạng biến đổi z sang dạng phù hợp với việc thực thi bằng máy tính số, chúng ta có thể thực thi được các bộ điều khiển được sử dụng phổ biến trong công nghiệp như là bộ điều khiển tỷ lệ-tích phân-vi phân hay còn gọi là bộ điều khiển PID.

Phương trình đầu ra của bộ điều khiển PID có dạng như sau:

$$u(t) = K_p \left[e(t) + \frac{1}{T_i} \int_0^t e(t)\,dt + T_d \frac{de(t)}{dt} \right] \tag{25}$$

Trong đó $u(t)$ là tín hiệu đầu ra của bộ điều khiển, $e(t)$ là tín hiệu đầu vào của bộ điều khiển, K_p là hệ số tỷ lệ, T_i là thời

gian tích phân, T_d là thời gian vi phân. Mặt khác, biến đổi Laplace của phương trình (25) có dạng như sau:

$$U(p)=\left(K_p+\frac{K_p}{T_i p}+K_p T_d p\right)E(p) \tag{26}$$

Biến đổi z phương trình (26) có dạng như sau:

$$U(z)=\left(K_p+\frac{K_p}{T_i}\frac{T}{1-z^{-1}}+K_p T_d\frac{1-z^{-1}}{T}\right)E(z) \tag{27}$$

Trong đó T là chu kỳ lấy mẫu.

Nếu đặt $K_p=a$, $\frac{K_p}{T_i}T=b$ và $K_p T_d=c$ thì hàm truyền của bộ điều khiển có dạng như sau:

$$U(z)=aE(z)+P(z)+Q(z) \tag{28}$$

Trong đó

$$P(z)=\frac{b}{1-z^{-1}}E(z) \tag{29}$$

$$Q(z)=c\left(1-z^{-1}\right)E(z) \tag{30}$$

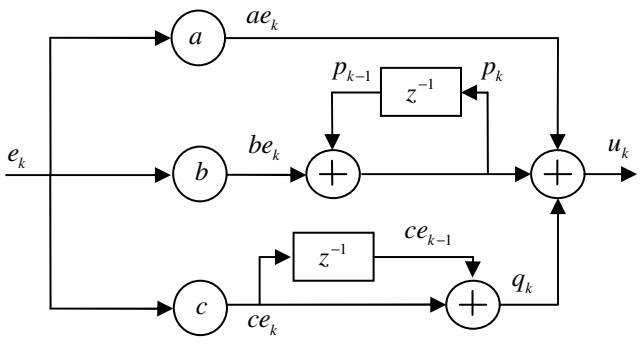
Lưu ý rằng $P(z)$ và $Q(z)$ chỉ là các biến trung gian. Phương trình (29) và (30) có thể được viết dưới dạng lấy mẫu tại các thời điểm lấy mẫu k khác nhau như sau:

$$p_k=be_k+p_{k-1} \tag{31}$$

$$q_k=c\left(e_k+e_{k-1}\right) \tag{32}$$

$$u_k=ae_k+p_k+q_k \tag{33}$$

Các phương trình (31), (32) và (33) là các phương trình được sử dụng để thực thi bộ điều khiển PID sử dụng máy tính số. Các phương trình này tương đương với sơ đồ song song như hình 3.

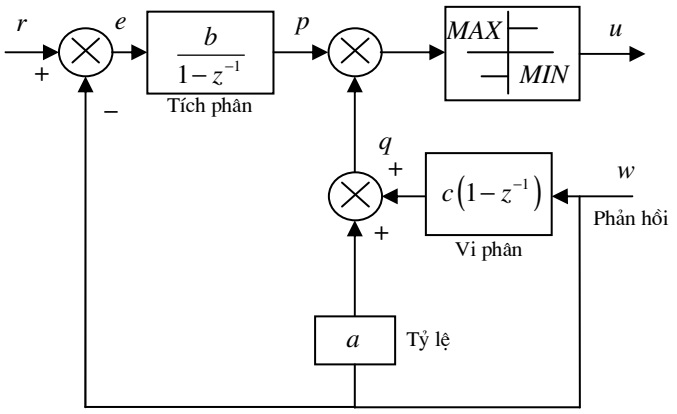


Hình 6: Thực thi hàm truyền của bộ điều khiển PID theo sơ đồ song song.

Một trong những vấn đề của bộ điều khiển PID theo sơ đồ như trên hình 6 là quá trình tích phân đến cùng (integral windup) của bộ điều khiển gây nên hiện tượng quá hiệu chỉnh trong thời gian dài đối với phản ứng đầu ra của hệ thống. Để tránh hiện tượng này chúng ta phải khống chế đầu ra của bộ điều khiển nằm trong phạm vi cho phép từ giá trị nhỏ nhất đến giá trị lớn nhất.

Vấn đề thứ hai của bộ điều khiển PID theo sơ đồ như trên hình 6 xuất phát từ quá trình vi phân của bộ điều khiển khi giá trị đặt thay đổi đáng kể làm cho tín hiệu sai lệch cũng thay đổi theo. Trong trường hợp như vậy, thành phần vi phân sẽ gây nên hiện tượng giật (kick) của đầu ra bộ điều khiển. Để khắc phục hiện tượng này, chúng ta cần thiết chuyển thành phần vi phân tới vòng phản hồi như hình 7. Thành phần tỷ lệ cũng có thể gây nên hiện tượng tương tự nên thành phần này cũng được chuyển tới vòng phản hồi.

Khi thiết kế các bộ điều khiển số, chúng ta cần phải quan tâm đến việc chọn khoảng thời gian lấy mẫu. Một cách đơn giản, chúng ta có thể chọn các mẫu với tốc độ càng nhanh càng tốt. Tuy nhiên, tốc độ lấy mẫu nhanh có thể gây nên một sự lãng phí không cần thiết cho phần cứng. Có nhiều quy tắc thực nghiệm để chọn chu kỳ lấy mẫu. Ví dụ, đối với một hệ thống có phản ứng vòng hở được làm gần đúng theo phương pháp Ziegler-Nichols thì chu kỳ lấy mẫu nên nhỏ hơn 1/4 thời gian tăng T_1.



Hình 7: Sơ đồ thực hành bộ điều khiển PID trong thực tế.

V. Độ ổn định của các hệ thống điều khiển số

Giống như các hệ thống điều khiển tương tự, chúng ta có thể sử dụng một số tiêu chuẩn để xét độ ổn định của các hệ thống điều khiển số. Trong bài báo này, chúng ta sẽ xem xét tiêu chuẩn ổn định Jury dùng để xét độ ổn định của các hệ thống điều khiển số có bậc hai và ba. Tiêu chuẩn Jury sẽ trở nên phức tạp nếu bậc của hệ thống là lớn.

Giả thiết chúng ta có hàm truyền của một hệ mạch vòng kín như sau:

$$\frac{Y(z)}{R(z)}=\frac{G(z)}{1+GH(z)} \tag{34}$$

Ở đây $F(z)=1+GH(z)=0$ được gọi là phương trình đặc tính của hệ thống. Độ ổn định của hệ thống phụ thuộc vào vị trí của các cực của hàm truyền. Đối với các hệ thống liên tục, hệ được xem là ổn định nếu các cực nằm bên trái mặt phẳng p. Bằng cách ánh xạ mặt phẳng p vào mặt phẳng z, một hệ thống điều khiển số được xem là ổn định nếu các cực nằm trong vòng tròn đơn vị.

Đối với phương trình đặc tính của hệ thống bậc hai có dạng:

$$F(z) = a_2 z^2 + a_1 z + a_0 = 0 \quad (35)$$

thì hệ được gọi là ổn định nếu:

$$F(1) > 0\ ,\ F(-1) > 0 \text{ và } |a_0| < a_2 \quad (36)$$

Đối với phương trình đặc tính của hệ bậc ba có dạng:

$$F(z) = a_3 z^3 + a_2 z^2 + a_1 z + a_0 = 0 \quad (37)$$

thì hệ được gọi là ổn định nếu:

$$F(1) > 0\ ,\ F(-1) < 0\ ,\ |a_0| < a_3$$
$$\text{và} \quad \left|\det\begin{bmatrix} a_0 & a_3 \\ a_3 & a_0 \end{bmatrix}\right| > \left|\det\begin{bmatrix} a_0 & a_1 \\ a_3 & a_2 \end{bmatrix}\right| \quad (38)$$

Ngoài ra chúng ta còn có thể sử dụng các phương pháp khác để xét ổn định của các hệ thống điều khiển số như:

-Tiêu chuẩn Routh-Hurwitz.

-Phương pháp quỹ tích gốc (root locus).

-Tiêu chuẩn Nyquist.

-Đồ thị Bode (Bode diagrams).

Các phương pháp trên có thể tham khảo một số tài liệu tiếng Việt.

VI. Thực nghiệm

Trong phần này chúng ta sẽ quan tâm đến việc ứng dụng vi điều khiển và máy tính cá nhân để xây dựng các hệ thống điều khiển số.

a) Phát triển các ứng dụng đo lường điều khiển sử dụng vi điều khiển

Ngày nay, vi điều khiển được sử dụng rộng rãi để phát triển các ứng dụng về điều khiển. Để lợi cho sinh viên trong quá trình học và phát triển các ứng dụng thật, các hãng đã tung ra thị trường các công cụ vừa có khả năng lập trình cho vi điều khiển và tiện lợi cho việc phát triển các ứng dụng điều khiển số như trên hình 8. Với chức năng của các công cụ này, người sử dụng có thể phát triển các ứng dụng về điều khiển số, đo lường các đại lượng vật lý, truyền thông,..

Hình 8 là bo mạch dùng để phát triển các ứng dụng của vi điều khiển họ 8051 của hãng MikroElectronika. Với bo mạch này người sử dụng có thể phát triển các ứng dụng với các loại vi điều khiển sau của hãng Atmel:

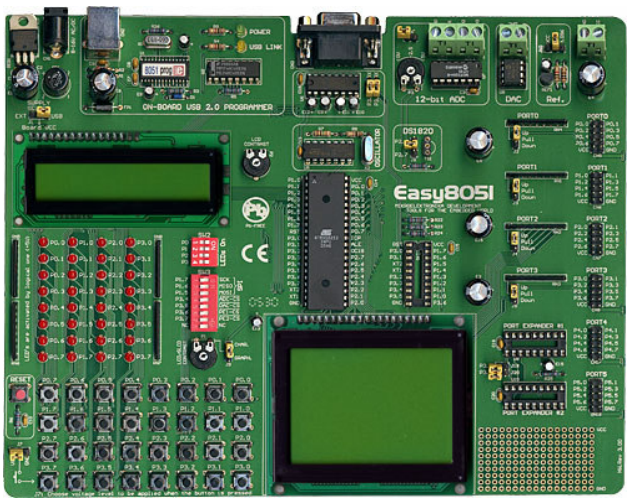
-AT89S51

-AT89S52

-AT89S53

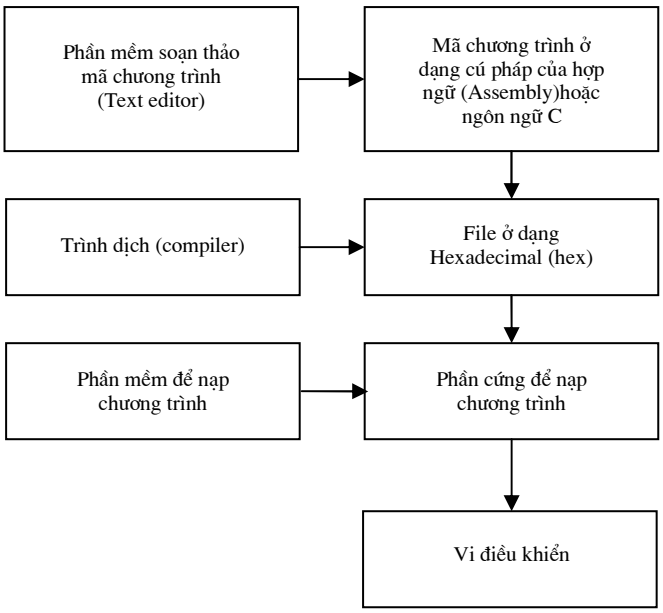
-AT89S8252

-AT89S8253

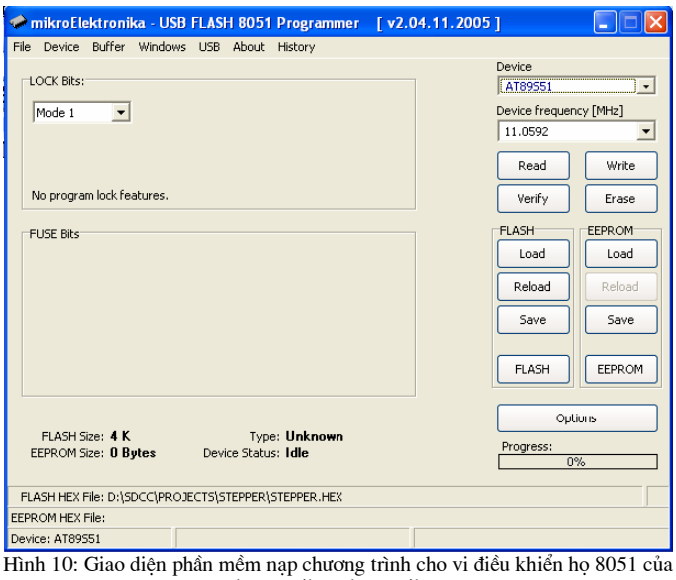


Hình 8: Bo mạch lập trình và phát triển các ứng dụng điều khiển sử dụng vi điều khiển họ 8051 của hãng MikroElectronika..

Quy trình lập trình cho vi điều khiển được trình bày trên hình 9. Trước tiên, chúng ta cần phải sử dụng một chương trình soạn thảo văn bản để viết chương trình điều khiển. Chương trình soạn thảo văn bản đơn giản để có thể viết được chương trình là Notepad. Tùy theo cú pháp của ngôn ngữ được sử dụng để lập trình cho vi điều khiển như hợp ngữ hay C mà các mã chương trình có thể lưu lại ở dạng là "tên_file. a51" hay "tên_file. c". Tiếp đó, chúng ta phải sử dụng một chương trình được gọi là trình dịch (compiler) để chuyển mã chương trình sang dạng file dạng hex (file này còn được gọi là phần mềm nhúng) để nạp vào bộ nhớ của vi điều khiển. Đối với lập trình bằng hợp ngữ, chúng ta có thể sử dụng trình dịch miễn phí ASEM-51 (http://plit.de/asem-51/final13.htm). Đối với lập trình bằng ngôn ngữ C, chúng ta có thể sử dụng trình dịch miễn phí SDCC (http://sdcc.sourceforge.net). Sau khi đã chuyển từ mã cương trình sang dạng file hex, chúng ta phải sử dụng phần cứng (hình 8) và phần mềm để nạp chương trình cho bộ vi điều khiển (hình 10).



Hình 9: Các bước lập trình cho vi điều khiển.



Hình 10: Giao diện phần mềm nạp chương trình cho vi điều khiển họ 8051 của hãng MikroElectronika.

b) Phần cứng giao diện với máy tính

Để số hóa tín hiệu phản hồi thuận tiện cho việc xử lý bằng máy tính, chúng ta phải sử dụng các bộ chuyển đổi tương tự sang số. Các bộ chuyển đổi tượng tự sang số có thể được chia làm hai loại. Loại thứ nhất được gọi là các bộ chuyển đổi tượng tự sang số đầu ra song song có nghĩa là các bộ chuyển đổi này này có dạng tín hiệu số đầu ra ở dạng các bit song song. Loại thứ hai được gọi là các bộ chuyển đổi tượng tự sang số nối tiếp tức là đầu ra số của các bộ chuyển đổi này là các bit nối tiếp.

Trong bài báo này chúng ta quan tâm đến việc sử dụng bộ chuyển đổi tương tự sang số song song ADC0809 [2]. Vi điều khiển này bao một bộ dồn kênh 8 đầu vào và được địa chỉ hóa bởi 3 bit. Đầu ra của bộ chuyển đổi bao gồm 8 bit song song. Bố trí chân ra và cách mắc mạch ngoài của bộ chuyển đổi tương tự sang số ADC0809 có thể tham khảo trong phần phụ lục.

Đầu ra 8 bit của bộ chuyển đổi tương tự sang số ADC0809 được ghép nối với một cổng của một vi điều khiển AT98S51 ví dụ như là cổng P1. Vi điều khiển AT89S51 này được lập trình để giao tiếp với cổng nối tiếp của máy tính theo chuẩn RS-232. Tương tự để gửi một tín hiệu từ máy tính đến các thiết bị chấp hành tương tự, chúng ta phải sử dụng các bộ chuyển đổi từ số sang tương tự. Ví dụ chúng ta có thể sử dụng bộ chuyển đổi số sang tương tự DAC0808 [3].

Tốc độ truyền thông nối tiếp phụ thuộc vào phương pháp sử dụng các bộ định thời của vi điều khiển. Khi bộ định thời 1 (Timer 1) được sử dụng ở chế độ 2 (mode 2) [4], tốc độ truyền thông nối tiếp ký hiệu là BR tính theo đơn vị baud (số bit trên giây) được xác định như sau:

$$BR = \frac{f}{32.12.(256 - TH1)} \tag{39}$$

Trong đó f là tần số dao động của mạch và $TH1$ là giá trị của Timer 1. Từ công thức (39), ta dễ dàng suy ra:

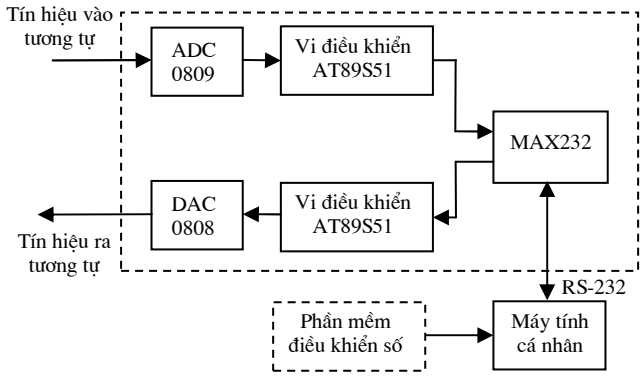
$$TH1 = 256 - \frac{f}{384.BR} \tag{40}$$

Giả sử chúng ta cần ta cần tạo nên tốc độ truyền dữ liệu nối tiếp là 9600 baud và tần số dao động cấp cho vi điều khiển bằng thạch anh là 11,0592 thì giá trị của Timer 1 khi đó sẽ là:

$$TH1 = 256 - \frac{11059200}{384.9600} = 253 = FD(H)$$

Như vậy là giá trị của TH1 là 253 ở hệ 10 (decimal) hay FD ở hệ mười sáu (hexadecimal).

Trong một hệ thống đo lường và điều khiển bằng sử dụng vi điều khiển chúng ta có thể phải sử dụng nhiều vi điều khiển. Mỗi vi điều khiển đảm nhận một chức năng khác nhau. Ví dụ quá trình thu thập dữ liệu sẽ yêu cầu một vi điều khiển riêng. Tương tự quá trình gửi truyền một tín hiệu điều khiển từ máy tính đến các thiết bị ngoại vi sẽ đòi hỏi phải sử dụng một vi điều khiển khác. Sơ đồ khối của phần cứng giao tiếp với máy tính được trình bày như trên hình 11.



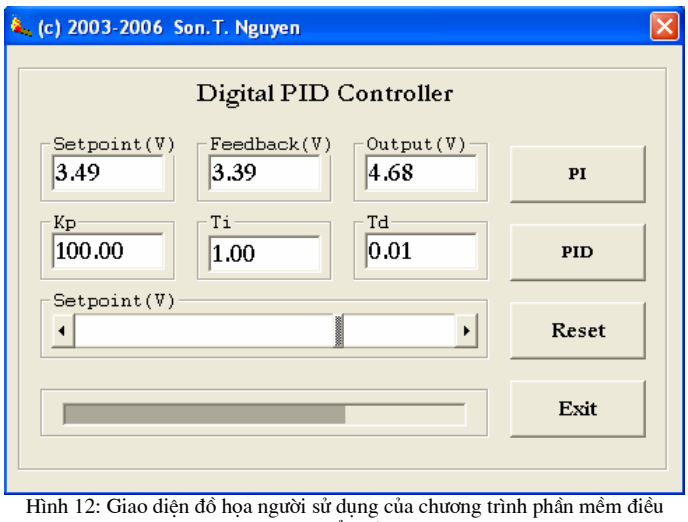
Hình 11: Sơ đồ khối của phần cứng giao tiếp với máy tính cá nhân.

Đối với các hệ máy tính xách tay đời mới không có cổng nối tiếp, chúng ta phải sử dụng cáp chuyển đổi USB sang RS-232. Các cáp chuyển đổi này đi kèm với những driver điều khiển.

b) Phần mềm điều khiển số

Phần mềm điều khiển được xây dựng với ngôn ngữ Visual Basic. Trong bài báo này chúng ta quan tâm đến việc xây dựng phần mềm cho bộ điều khiển PI và PID. Phần mềm bao gồm hai phần:

- Phần thứ nhất là giao diện người sử dụng bằng đồ họa cho phép người sử dụng có thể quan sát giá trị phản hồi, đầu ra của bộ điều khiển như trên hình 12. Đồng thời giao diện cũng cho phép người sử dụng thay đổi dễ dàng các thông số của hệ thống điều khiển như giá trị đặt (setpoint) bằng thanh trượt ngang hay hệ số tỷ lệ K_p, hằng số thời gian tích phân T_i và hằng số thời gian vi phân T_d của bộ điều khiển PID có thể được nhập trực tiếp từ bàn phím.
- Phần thứ hai là mã chương trình điều khiển bao gồm chương trình con thu thập tín hiệu phản hồi sau khi được số hóa bởi bộ chuyển đổi tương tự sang số và chương trình con của các bộ điều khiển PI và PID. Thuật toán để xây dựng bộ điều khiển PID đã được để cập trong phần IV. Giá trị của các đại lượng đo được có thể được hiển thị tùy theo từng ứng dụng điều khiển cụ thể.



Hình 12: Giao diện đồ họa người sử dụng của chương trình phần mềm điều khiển số.

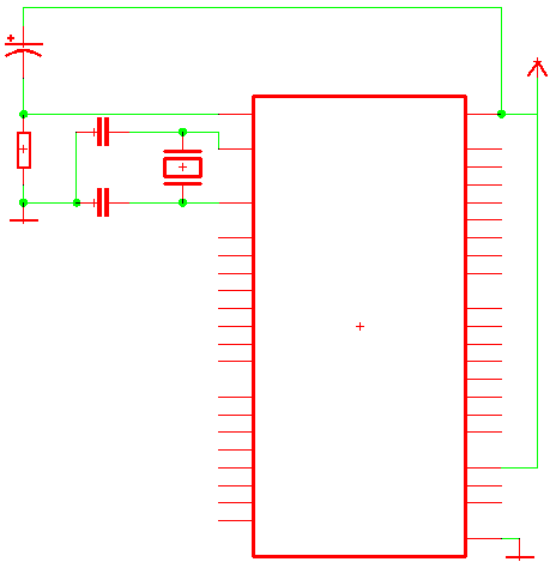
VII. Kết luận

Bài báo đã tổng hợp lại các kiến thức cơ bản trong điều khiển số ở mức độ đơn giản nhưng đủ để thực thi với các vi điều khiển hiện có tại Việt Nam. Hướng phát triển tương lai của ứng dụng này là thiết kế các bộ điều khiển số cho các đối tượng điều khiển cụ thể.

Phụ lục

Phụ lục 1: Mạch ngoài của vi điều khiển AT89S51

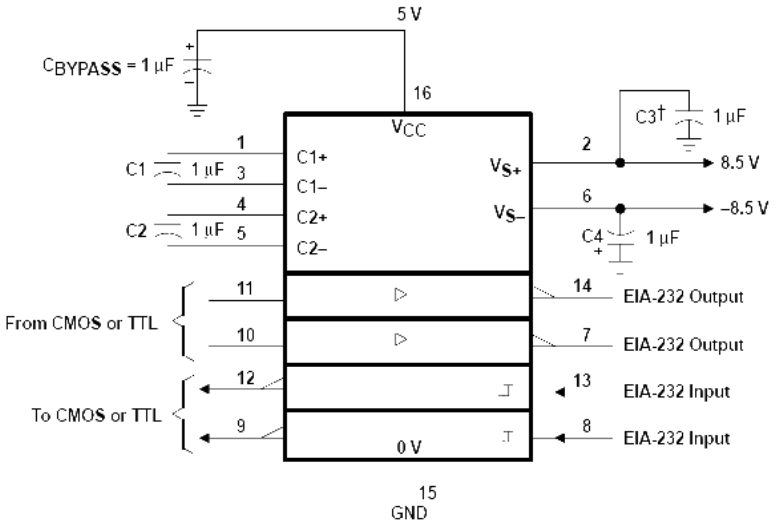
Để vi điều khiển có thể làm việc sau khi được lập trình, chúng ta phải mắc mạch ngoài cho vi điều khiển. Ngoài ra tùy theo từng ứng dụng chúng ta có thể ghép nối các cổng của vi điều khiển với các bộ chuyển đổi tương tự sang số hay số sang tương tự để thu thập dữ liệu và điều khiển. Hình 1 là sơ đồ mạch ngoài tối thiểu để vi điều khiển có thể làm việc được.



Hình 1: Mạch ngoài của vi điều khiển AT89S51.

Phụ lục 2: Chuyển đổi mức tín hiệu từ TTL sang RS-232 và ngược lại

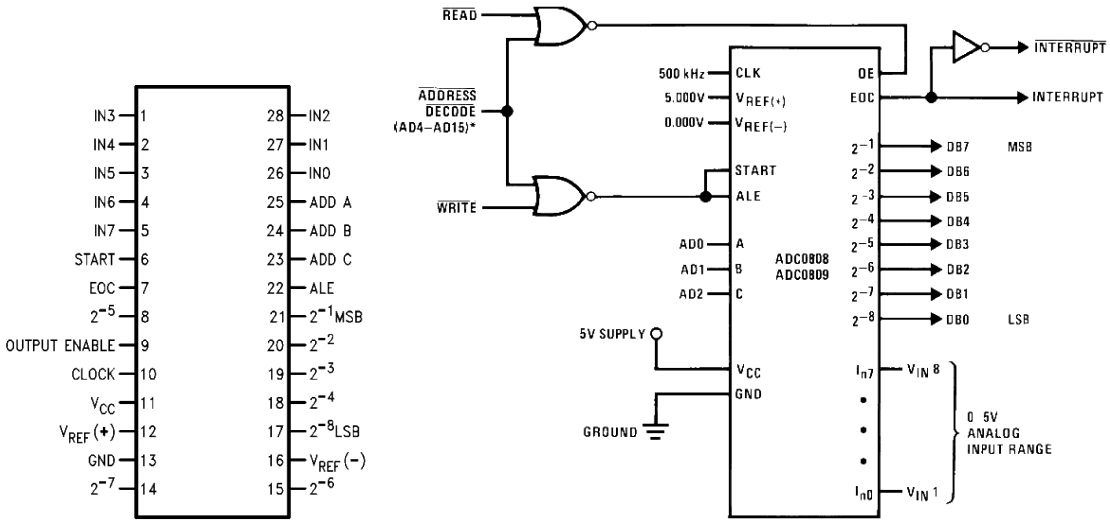
Tiêu chuẩn RS-232 định nghĩa các mức điện áp từ ±3 đến ±15 V tương ứng với 1 logic và 0 logic. Giải giá trị điện áp gần 0 V được xem như là không có hiệu lực. Mức 1 logic được định nghĩa là một điện áp âm còn không logic được định nghĩa là một điện áp dương. Mặt khác, vi điều khiển AT89S51 làm việc với tín hiệu 1 logic tương ứng với 5 V và 0 logic tương ứng với 0 V. Do đó, để vi điều khiển có thể giao tiếp được với máy tính qua cổng nối tiếp theo chuẩn RS-232, chúng ta cần phải sử dụng IC MAX232 để chuyển đổi mức tín hiệu như trên hình 1.



Hình 2: IC MAX232 dùng để chuyển đổi mức tín hiệu từ TTL sang RS-232 và ngược lại.

Phụ lục 3: Bộ chuyển đổi tương tự sang số ADC 0809

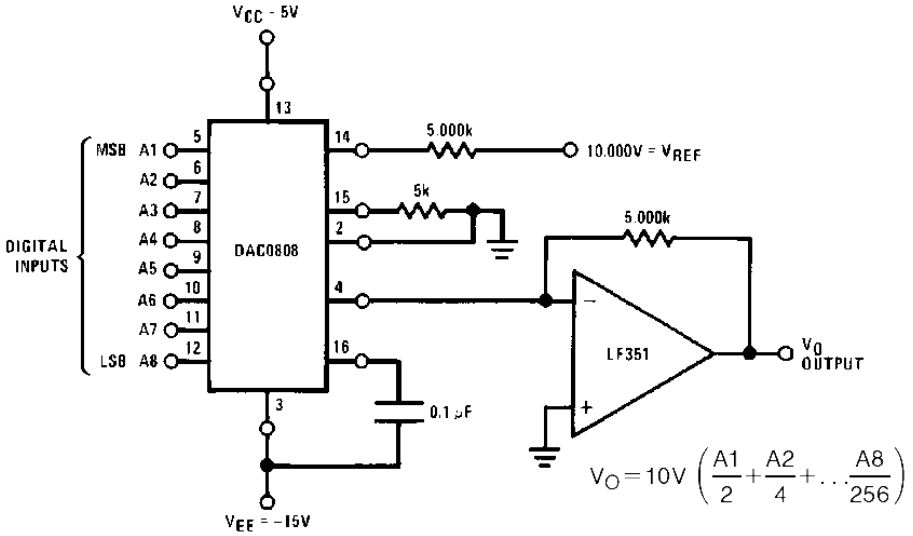
Bộ chuyển đổi số sang tương tự được sử dụng để chuyển đổi tín hiệu tương tự trong dải 0 đến 5 V sang tín hiệu số 8 bit song song. Sơ đồ chân ra và mạch ngoài để sử dụng được trình bày trên hình 3.



Hình 3: Bộ trí chân ra (bên trái) và sơ đồ mắc mạch ngoài của bộ chuyển đổi tương tự sang số 0808/0809.

Phụ lục 4: Bộ chuyển đổi số sang tương tự DAC 0808

Bộ chuyển đổi số sang tương tự dùng để chuyển đổi tín hiệu số 8 bit sang tín hiệu tương tự. Sơ đồ mạch ngoài để sử dụng của DAC 0808 được trình bày trên hình 4.



Hình 4: Sơ đồ mạch ngoài của bộ chuyển đổi tương tự sang số 0808/0809.

Phụ lục 5: Thiết kế phần cứng giao diện với máy tính cá nhân sử dụng vi điều khiển AT89S51với truyền thông nối tiếp

Trong kỹ thuật truyền thông và khoa học máy tính, truyền thông nối tiếp là quá trình gửi hoặc nhận 1 bit tại một thời điểm, khác với truyền thông song song nhận hoặc gửi nhiều bit cùng một lúc. Trong truyền thông nối tiếp, người ta đã đưa ra một số tiêu chuẩn như RS-232, RS-423, RS-485,...Phần này sẽ giới thiệu cách lập trình truyền thông nối tiếp sử dụng chuẩn RS-232 và vi điều khiển AT89S51.

RS-232 là tiêu chuẩn cho các tín hiệu dữ liệu ở dạng nhị phân nối tiếp kết nối giữa một thiết bị đầu cuối (data terminal equipment) và một thiết bị truyền thông dữ liệu (data communications equipment). Dạng truyền thông này chủ yếu sử dụng cổng nối tiếp của máy tính (serial computer port). Thiết bị đầu cuối có thể là một máy tính còn thiết bị truyền thông dữ liệu có thể là một modem.

Để giao tiếp với máy tính qua cổng nối tiếp, chúng ta sử dụng vi điều khiển AT89S51 với bộ định thời Timer1. Chương trình viết bằng hợp ngữ để truyền và nhận dữ liệu từ máy tính đến cổng P1 của vi điều khiển AT89S51 được trình bày dưới đây. Chương trình bao gồm hai phần. Phần thứ nhất dùng để định nghĩa chức năng của Timer 1 ở chế độ 2 (mode 2) với truyền thông nối tiếp 8 bit, tốc độ truyền 9600 baud khi tần số dao động là 11,0592MHz. Phần thứ hai là quá trình truyền và nhận dữ liệu sử dụng thanh ghi SBUF.

a) Chương trình nhận dữ liệu 8 bit sử dụng vi điều khiển AT89S51 với tần số dao động 11,0592 MHz:

```
ACALL   START

RS232:
            MOV     TMOD,#20h           ;Set Timer 1 for auto reload - mode 2
            MOV     TCON,#41h           ;Run Timer 1
            MOV     TH1,#0FDh           ;Set Timer 1 = 253 for 9600 baud with XTAL = 11.0592MHz
            MOV     SCON,#50h           ;8 bit data, mode 1
            ANL     PCON,#7Fh           ;Clear SMOD
            RET

RECEIVE:
            CLR     RI
            MOV     A,SBUF
WAIT:       JNB     RI,WAIT
            RET

START:
            ACALL   RS232
LOOP:
            ACALL   RECEIVE
            MOV     P1,A
            AJMP    LOOP
END
```

b) Chương trình truyền dữ liệu 8 bit sử dụng vi điều khiển AT89S51 với tần số dao động 11,0592 MHz:

```
ACALL   START

RS232:
            MOV     TMOD,#20h           ;Set Timer 1 for auto reload - mode 2
            MOV     TCON,#41h           ;Run Timer 1
            MOV     TH1,#0FDh           ;Set Timer 1 = 253 for 9600 baud with XTAL = 11.0592MHz
            MOV     SCON,#50h           ;8 bit data, mode 1
            ANL     PCON,#7Fh           ;Clear SMOD
            RET

SEND:
```

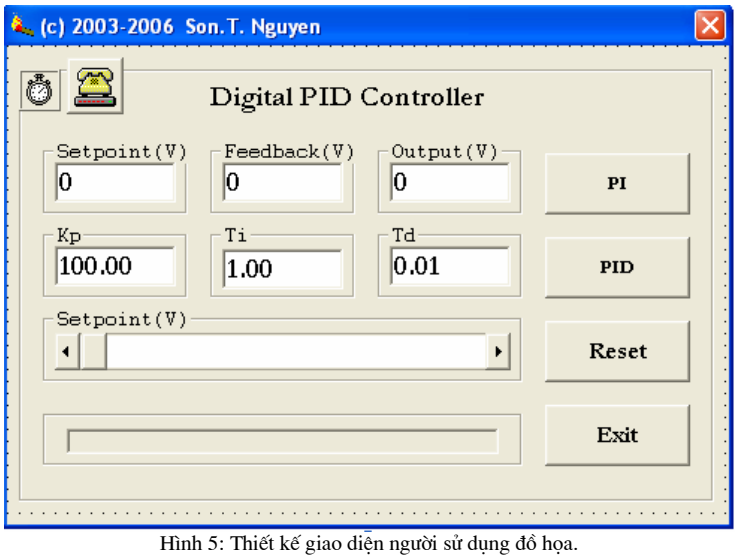
```
            CLR     TI
            MOV     SBUF,A
WAIT:       JNB     TI,WAIT1
            RET

START:
            ACALL   RS232
LOOP:
            MOV     A,P1
            ACALL   SEND
            AJMP    LOOP
END
```

Đối với một số dòng máy tính xách tay đời mới chỉ có cổng USB mà không có cổng nối tiếp, chúng ta phải sử dụng cáp chuyển đổi từ USB sang RS232.

Phụ lục 6: Chương trình phần mềm của bộ điều khiển số PID và giao diện người sử dụng viết bằng ngôn ngữ Visual Basic

a) Giao diện người sử dụng:

Giao diện người sử dụng được xây dựng từ các hộp thoại của ngôn ngữ Visual Basic. Các hộp thoại này có thể được điều khiển bằng mã chương trình. Hình 4 là giao diện người sử dụng được thiết kế với các hộp thoại.



Hình 5: Thiết kế giao diện người sử dụng đồ họa.

b) Phần mã chương trình:

Phần mã chương trình bao gồm các chương trình con điều khiển việc truyền và nhận tín hiệu theo chuẩn RS-232. Bên cạnh đó mã chương trình còn bao gồm chương trình con của bộ điều khiển PI và PID. Mã chương trình được trình bày ở dưới đây.

```
'----------------------------------------------------------------------
' Digital Control System Design based on Ziegler-Nichols algorithm:
'
'   -P controller:      Kp = T1/K/Td
'
'   -PI controller:     Kp = 0.9*T1/K/Td
'                       Ti = 3.3*Td
'
'   -PID controller:    Kp = 1.2*T1/K/Td
'                       Ti = 2*Td
'                       Tde = 0.5*Td
'
' Implementation of digital PID controller
' In s-domain, the output signal of the PID controller
' can be expressed as follows:
'           /                \
'          |      1           |
' E(s)= Kp |1 + ----- + Td*s |E(s)                              (1)
'          |     Ti*s         |
'           \                /
' Using the the following properties of the z-transform:
'
'   1        z           1
' ----- = ------- = ------------
'   s      z - 1     1 - z^(-1)
'
' We can derive the z-form of the PID controller:
'           /                                    \
'          |           T             Td*[1 - z^(-1)] |
' U(z)= Kp |1 + ----------------- + -----------------|E(z)      (2)
'          |     Ti*[1 - z^(-1)]             T       |
```

```
'           \                                       /
'
' Kp is the propotional constant
' Ti is the integral time constant
' Ts id the derivative time constant
' T is the sampling interval
'
' Let:
'                Kp*T           Kp*Td
' a = Kp   b = ------      c = ------
'                 Ti              T
'
' Equation (2) can therefore be expressed as follows
'
'                          1
' U(z)= a*E(z) + b*------------*E(z) + c*[1 - z^(-1)]*E(z)      (3)
'                   [1 - z^(-1)]
'
' U(z)= a*E(z) + P(z) + Q(z)                                   (4)
'
'where
'
'               1
' P(z)= b*-------------*E(z)                                   (5)
'          [1 - z^(-1)]
'
' Q(z) = c*[1 - z^(-1)]*E(z)                                   (6)
'
'
' From (5), we can write
'
' P(kT) = P(kT-T) + b*E(kT)                                   (7)
'
' From (6), we have
'
' Q(kT) = c*[E(kT) - E(kT-T)]                                 (8)
'
'
'Reference: page 251 "Microcontroller Based Applied Digital Control"
'-D. Ibrahim
'(c) 2006 John Wiley & Sons, Ltd
'
'
' Copyright (c) 2004 - 2005 Son Nguyen
' University of Technology, Sydney
' E-mail: Son.NguyenThanh@uts.edu.au
'--------------------------------------------------------------------------
Option Explicit
Dim Serial_Data As String
Dim feedback As String
Dim High_Byte As Long
Dim Low_Byte As Long
Dim Two_Byte As Long
Dim Kp As Single
Dim Tin As Single
Dim Tde As Single
Dim Ts As Single
Dim a As Single
Dim b As Single
Dim c As Single
Dim rkt As Single
Dim ykt As Single
Dim ykt_1 As Single
Dim ekt As Single
Dim ekt_1 As Single
Dim pkt As Single
Dim pkt_1 As Single
Dim qkt As Single
Dim qkt_1 As Single
Dim ukt As Single
Dim Vmax As Integer
Dim Vmin As Integer

Private Sub Command2_Click()
    Unload Me
End Sub

Private Sub Form_Load()
    MSComm1.RThreshold = 2
    MSComm1.InputLen = 2
    MSComm1.CommPort = 4
    MSComm1.Settings = "9600,N,8,1"
    MSComm1.PortOpen = True
    HScroll1.Min = 0
    HScroll1.Max = 255
    Picture1.AutoRedraw = True
    Picture1.ScaleLeft = 0
    Picture1.ScaleWidth = 255
End Sub

Private Sub MSComm1_OnComm()
```

```
    If MSComm1.CommEvent = comEvReceive Then
        Serial_Data = MSComm1.Input             ' Get data
        High_Byte = Asc(Mid$(Serial_Data, 1, 1)) ' Get 1st byte
        Low_Byte = Asc(Mid$(Serial_Data, 2, 1))  ' Get 2nd byte
        Two_Byte = JoinHighLow(High_Byte, Low_Byte)
    End If
End Sub

Private Sub PID_parameters(m As Integer)
    Kp = Format(Val(Text3.Text), "0.00")
    Tin = Format(Val(Text4.Text), "0.00")
    Tde = Format(Val(Text5.Text), "0.00")
    Ts = 0.05
End Sub

Private Sub PI_Click()
    a = Kp
    b = (Kp * Ts) / Tin
    c = 0
    ekt_1 = 0
    pkt_1 = 0
    qkt_1 = 0
End Sub

Private Sub PID_Click()
    a = Kp
    b = (Kp * Ts) / Tin
    c = (Kp * Tde) / Ts
    ekt_1 = 0
    pkt_1 = 0
    qkt_1 = 0
End Sub

Private Sub Reset_Click()
    a = 0
    b = 0
    c = 1
    ekt_1 = 0
    pkt_1 = 0
    qkt_1 = 0
End Sub

Private Sub Timer1_Timer()
    PID_parameters (1)
    feedback = Low_Byte
    rkt = HScroll1.Value
    Text9.Text = Format(rkt / 51, "0.00")
    Text1.Text = Format(feedback / 51, "0.00")
    DrawScale CStr(HScroll1.Value)
    Standard_PID (1)
End Sub

Private Function JoinHighLow(lHigh As Long, lLow As Long) As Long
  JoinHighLow = (lHigh * &H100) Or lLow 'Join High Byte and Low Byte
End Function

Private Sub DrawScale(Variable As Long)
    Picture1.Cls
    Picture1.Line (0, 0)-(Variable, 400), vb3DShadow, BF
End Sub

Private Sub Form_Unload(Cancel As Integer)
    MSComm1.PortOpen = False
End Sub

Private Sub Standard_PID(m As Integer)
    'Zero Order Holding approximation(Standard PID controller)
    ykt = feedback
    ekt = rkt - ykt             'the Error
    pkt = b * ekt + pkt_1       'the I term
    qkt = c * (ekt - ekt_1)     'the D term
    ukt = a * ekt + pkt + qkt   'the PID output
    'Prevent the controller from the "integral windup"
    Vmax = 2550
    Vmin = 0
    If ukt > Vmax Then
        pkt = pkt_1
        ukt = Vmax
    End If
    If ukt < Vmin Then
        pkt = pkt_1
        ukt = Vmin
    End If
    'Save the variables
    pkt_1 = pkt
    ekt_1 = ekt
    'Output the control signal
    Text2.Text = Format(ukt / 510, "0.00")
    MSComm1.Output = Chr(Round(ukt / 10))
End Sub
```

Tài liệu tham khảo

[1] D. Ibrahim, "Microcontroller Based Applied Digital Control," *John Wiley & Sons, Ltd*, 2006.
[2] National, "ADC0808/ADC0809 8-Bit μP Compatible A/D Converters with 8-Channel Multiplexer," 1999.
[3] National, "DAC0808/DAC0807/DAC0806 8-Bit D/A Converters," 1995.
[4] Atmel, "8051 Flash Microcontroller-Memory Organization."