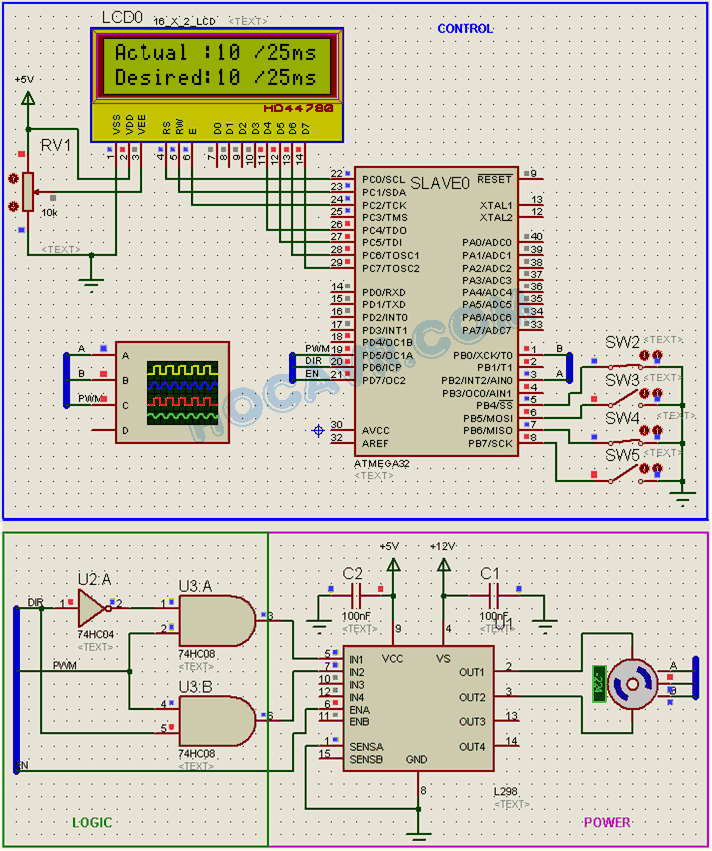
**I. Giới thiệu**

     Điều khiển động cơ DC (DC Motor) là một ứng dụng thuộc dạng cơ bản nhất của điều khiển tự động vì DC Motor là cơ cấu chấp hành (actuator) được dùng nhiều nhất trong các hệ thống tự động (ví dụ robot). Điều khiển được DC Motor là bạn đã có thể tự xây dựng được cho mình rất nhiều hệ thống tự động. Khái niệm Servo mà tôi dùng trong bài học này để chỉ một hệ thống hồi tiếp. DC servo motor là động cơ DC có bộ điều khiển hồi tiếp.  
     Bài này là một bài tổng hợp nhiều vấn đề ứng dụng AVR bao gồm nhận dữ liệu từ người dùng, điều khiển motor, đọc encoder, hiển thị LCD, cả giải thuật điều khiển PID và mạch công suất cho Motor…Do đó, ít nhất bạn phải nắm được các vấn đề cơ bản như Timer-Counter, TexLCD, mạch cầu H. Phần còn lại tôi sẽ giải thích trong lúc học bài này. Có 2 phương pháp điều khiển động cơ DC là analog và digital. Mục đích chính của chúng ta là dùng AVR điều khiển động cơ DC nên phương pháp số mà cụ thể là phương pháp điều rộng xung (PWM) sẽ được giới thiệu. Ngoài ra, khi nói đến điều khiển động cơ DC có 2 đại lương điều khiển chính là vị trí (số vòng quay) và vận tốc. Trong phần giải thích về bộ điều khiển PID tôi sẽ điều khiển vị trí làm ví dụ, tuy nhiên trong phần ví dụ lập trình cho AVR chúng ta sẽ thực hiện điều khiển vận tốc cho DC Motor. Bằng cách này, bạn có thể tự tin để mở rộng ví dụ để điều khiển cho cả 2 đại lượng. Vì là điều khiển một cách tự động nên chúng ta cần đọc về đại lượng điều khiển (cụ thể là vị trí hoặc vận tốc motor) và hồi tiếp (feedback) về để “hiệu chỉnh” PWM cấp cho động cơ. Chúng ta sẽ dùng incremental optical encoder để đọc số vòng quay và hồi tiếp về cho AVR. Bộ điều khiển PID sẽ được dùng và vận hành bởi AVR. Tổng quát, bài học này bao gồm:  
       - AVR phát PWM điều chỉnh vận tốc động cơ: phần này bạn xem lại bài 4 về Timer-Counter. Điều cơ bản cần nắm là bằng cách thay đổi độ rộng của xung PWM chúng ta sẽ thay đổi được vận tốc Motor.  
       - Xung PWM không trực tiếp làm quay động cơ mà thông qua một mạch công suất gọi là dirver. Driver cho DC Motor chính là mạch cầu H mà chúng ta đã tìm hiểu trong bài “Mạch cầu H”. Trong bài học này, tôi giới thiệu một chip có tích hợp sẵn mạch cầu H, chip L298D.  
      - Để việc điều khiển chip driver L298D dễ dàng, chúng ta sẽ tạo một mạch logic dùng các cổng NOT và AND.  
      - Động cơ DC mà chúng ta sử dụng có tích hợp sẵn một encoder 3 ngõ ra, chúng ta sẽ dùng AVR để đọc số xung (hay số vòng quay) và tính ra vận tốc của Motor. Việc đọc encoder sẽ được thực hiện bằng ngắt ngoài.  
      - Một giải thuật PID được xây dựng trong AVR để hiệu chỉnh vận tốc động cơ.  
      - Người dùng sẽ nhập vận tốc cần điều khiển vào AVR thông qua các switches. Vận tốc mong muốn và vận tốc thực của động cơ được hiển thị trên Text LCD.   
      Mạch điện ví dụ được trình bày trong hình 1.

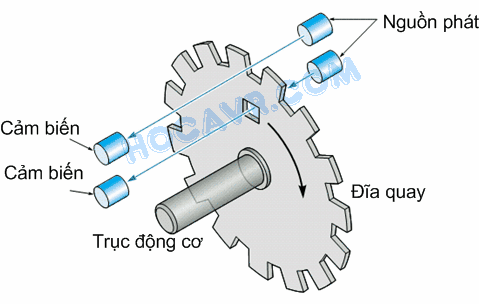


Hình 1. Hệ thống điều khiển động cơ DC servo.

     Trong mạch điện hình 1, tôi chia hệ thống thành 3 nhóm: nhóm CONTROL bao gồm AVR vận hành giải thuật điều khiển PID và việc nhập, xuất. Nhóm LOGIC thực hiện việc biến đổi các tín hiệu điều khiển để tạo ra các tín hiệu phù hợp cho chip driver. Nhóm POWER bao gồm chip driver L298D và DC Motor. Ngoài ra còn có một Encoder được tích hợp sẵn trên DC Motor.  
      Phần tiếp theo chúng ta sẽ tìm hiểu riêng từng nhóm, cuối cùng là viết chương trình cho AVR điều khiển hệ thống DC Servo Motor

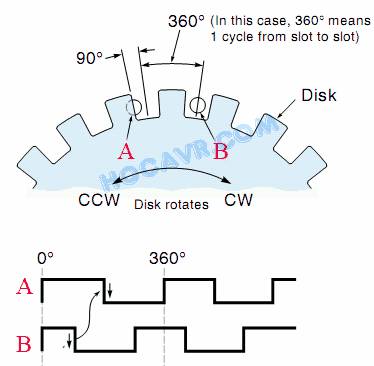
**II. Incremental Optical Encoder**

      Để điều khiển số vòng quay hay vận tốc động cơ thì chúng ta nhất thiết phải đọc được góc quay của motor. Một số phương pháp có thể được dùng để xác định góc quay của motor bao gồm tachometer (thật ra tachometer đo vận tốc quay), dùng biến trở xoay, hoặc dùng encoder. Trong đó 2 phương pháp đầu tiên là phương pháp analog và dùng optiacal encoder (encoder quang) thuộc nhóm phương pháp digital. Hệ thống optical encoder bao gồm một nguồn phát quang (thường là hồng ngoại – infrared), một cảm biến quang và một đĩa có chia rãnh. Optical encoder lại được chia thành 2 loại: encoder tuyệt đối (absolute optical encoder) và encoder tương đối (incremental optical encoder). Trong đa số các DC Motor, incremental optical encoder được dùng và mô hình động cơ servo trong bài này cũng không ngoại lệ. Từ bây giờ khi tôi nói encoder tức là incremental encoder. Hình 2 là mô hình của encoder loại này.



Hình 2. Optical Encoder (trích từ [1]).

      Encoder thường có 3 kênh (3 ngõ ra) bao gồm kênh A, kênh B và kênh I (Index). Trong hình 2 bạn thấy hãy chú ý một lỗ nhỏ bên phía trong của đĩa quay và một cặp phat-thu dành riêng cho lỗ nhỏ này. Đó là kênh I của encoder. Cữ mỗi lần motor quay được một vòng, lỗ nhỏ xuất hiện tại vị trí của cặp phát-thu, hồng ngoại từ nguồn phát sẽ xuyên qua lỗ nhỏ đến cảm biến quang, một tín hiệu xuất hiện trên cảm biến. Như thế kênh I xuất hiện một “xung” mỗi vòng quay của motor. Bên ngoài đĩa quay được chia thành các rãnh nhỏ và một cặp thu-phát khác dành cho các rãnh này. Đây là kênh A của encoder, hoạt động của kênh A cũng tương tự kênh I, điểm khác nhau là trong 1 vòng quay của motor, có N “xung” xuất hiện trên kênh A. N là số rãnh trên đĩa và được gọi là độ phân giải (resolution) của encoder. Mỗi loại encoder có độ phân giải khác nhau, có khi trên mỗi đĩa chĩ có vài rãnh nhưng cũng có trường hợp đến hàng nghìn rãnh được chia. Để điều khiển động cơ, bạn phải biết độ phân giải của encoder đang dùng. Độ phân giải ảnh hưởng đến độ chính xác điều khiển và cả phương pháp điều khiển. Không được vẽ trong hình 2, tuy nhiên trên các encoder còn có một cặp thu phát khác được đặt trên cùng đường tròn với kênh A nhưng lệch một chút (lệch M+0,5 rãnh), đây là kênh B của encoder. Tín hiệu xung từ kênh B có cùng tần số với kênh A nhưng lệch pha 90o. Bằng cách phối hợp kênh A và B người đọc sẽ biết chiều quay của động cơ. Hãy quan sát hình 3.

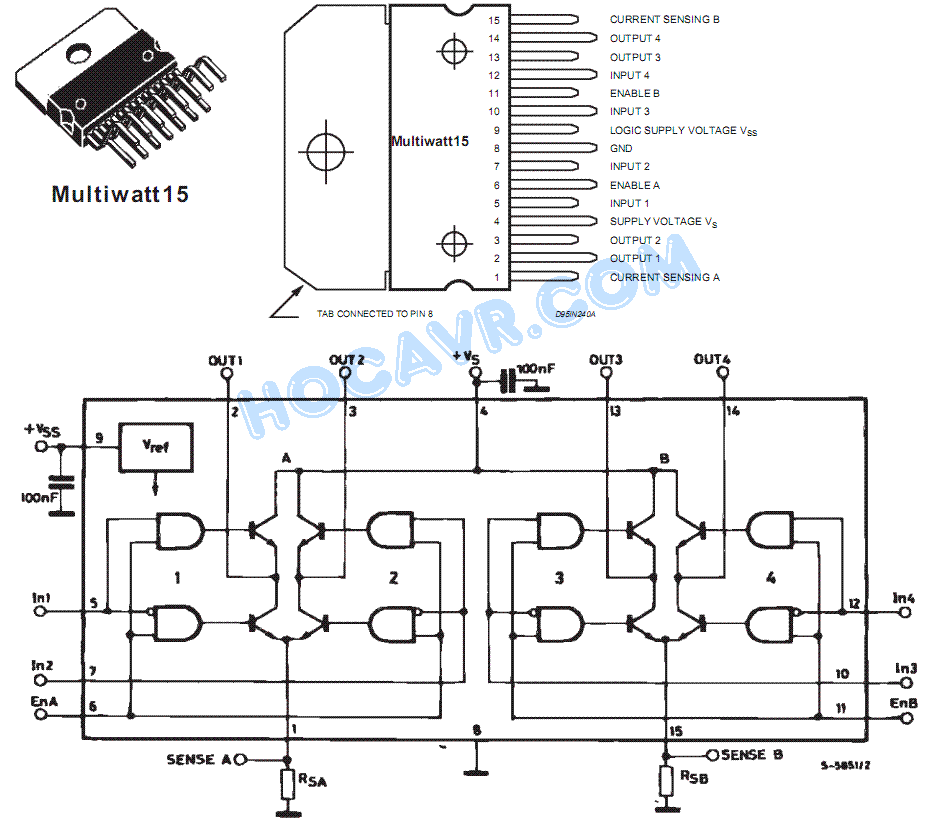


Hình 3. Hai kênh A và B lệch pha trong encoder (trích từ [1])

       Hình trên cùng trong hình 3 thể hiện sự bộ trí của 2 cảm biến kênh A và B lệch pha nhau. Khi cảm biến A bắt đầu bị che thì cảm biến B hoàn toàn nhận được hồng ngoại xuyên qua, và ngược lại. Hình thấp là dạng xung ngõ ra trên 2 kênh. Xét trường hợp motor quay cùng chiều kim đồng hồ, tín hiệu “đi” từ trái sang phải. Bạn hãy quan sát lúc tín hiệu A chuyển từ mức cao xuống thấp (cạnh xuống) thì kênh B đang ở mức thấp. Ngược lại, nếu động cơ quay ngược chiều kim đồng hồ, tín hiệu “đi” từ phải qua trái. Lúc này, tại cạnh xuống của kênh A thì kênh B đang ở mức cao. Như vậy, bằng cách phối hợp 2 kênh A và B chúng ta không những xác định được góc quay (thông qua số xung) mà còn biết được chiều quay của động cơ (thông qua mức của kênh B ở cạnh xuống của kênh A).   
     Câu hỏi bây giờ là làm thế nào để đọc encoder bằng AVR?  
       Tùy theo đại lượng điều khiển (vị trí hay vận tốc) và đặc điểm encoder (độ phân giải) chúng ta có các giải pháp sau để đọc encoder bằng AVR  
     - **Dùng input capture**: một số bộ timer-counter trên AVR có chức năng Input capture, hiểu nôm na như sau. Cứ mỗi lần có một tín hiệu (cạnh lên hoăc cạnh xuống) trên chân ICP (Input Capture Pin), giá trị thời gian của timer được tự động gán cho thanh ghi ICR (Input capture Register). So sánh giá trị thanh ghi ICR trong 2 lần liên tiếp sẽ đọc được chu kỳ của tín hiệu kích chân ICP. Từ đó suy ra tần số tín hiệu. Nếu một kênh của encoder được nối với chân ICP thì chúng ta có thể đo được tần số tín hiệu của kênh này. Nói cách khác, chúng ta sẽ tính được vận tốc của động cơ. Chúng ta có thể dùng ngắt Input capture và khi ngắt xảy ra, có thể đếm số thêm số xung để biết được góc quay motor, cũng có thể xác định được hướng quay thông qua xác định mức kênh B trong trình phục vụ ngắt input capture. Đây là một phương pháp hay, nhưng có nhược điểm là khá phức tạp khi sử dụng chức năng input capture của AVR. Mặc khác trên các chip AVR từ mega32 trở xuống, Input capture chỉ có ở timer 1, trong khi Timer này thường dùng để tạo PWM điều khiển động cơ.  
     - **Dùng chức năng counter**: đặt các kênh của encoder vào các chân đếm (T0, T1…) của các bộ timer chúng ta sẽ đếm được số lượng xung của các kênh. Đây là phương pháp sử dụng ít tài nguyên nhất (ít tốn thời gian cho encoder). Nhược điểm lớn nhất của phương pháp này là không xác định được chiều quay, mặc khác phương pháp này không ổn định khi vận tốc động cơ có sự thay đổi lớn.  
     - **Cuối cùng là sử dụng ngắt ngoài**: đây là phương pháp dễ nhưng chính xác để đọc encoder và cũng là phương pháp được dùng trong bài học này. Ý tưởng của phương pháp rất đơn giản, chúng ta nối kênh A của encoder với 1 ngắt ngoài (INT2 chẳng hạn) và kênh B với một chân nào đó bất kỳ (không phải chân ngắt). Cứ mỗi lần ngắt ngoài xảy ra, tức có 1 xung xuất hiện trên ở kênh A thì trình phục vụ ngắt ngoài tự động được gọi. Trong trình phục vụ ngắt này chúng ta kiểm tra mức của kênh B, tùy theo mức của kênh B chúng ta sẽ tăng biến đếm xung lên 1 hoặc giảm đi 1. Tuy nhiên, bạn cần phải tính toán rất cẩn thận khi sử dụng phương pháp này. Ví dụ trường hợp encoder có độ phân giải 2000 xung/vòng, motor bạn quay với vận tốc 100 vòng/s thì tần số xung trên kênh A của encode là 2000x100=200KHz, nghĩa là cứ mỗi 5 us ngắt ngoài xảy ra một lần. Tần số ngắt như thế là quá cao cho AVR, điều này có nghĩa là AVR chỉ tập trung cho mỗi việc “đếm xung”, không có đủ thời gian để thực thi các việc khác. Trong bài này, chúng ta chọn độ phân giải của encoder là 112 (112 xung trên mỗi vòng quay). Vận tốc tối đa của động cơ được chọn vào khoảng 30 vòng/s nên tần số xung lớn nhất từ encoder là 112x30=3.36KHz. Giá trị này hợp lí vì tần số cho AVR trong bài này được chọn 8MHz. Kênh A của encoder được nối với ngắt INT2 của chip atmega32, kênh B được nối với chân PB0, chúng ta không sử dụng kênh I (xem hình 1).  
     **Chú ý**: các ngõ ra trên đa số (gần như tất cả) các encoder có dạng cực góp hở (Open collector), muốn sử dụng chúng cần mắc điện trở kéo lên VCC (5V).

**III. Chip driver L298D**

      L298D là một chip tích hợp 2 mạch cầu H trong gói 15 chân. Tất cả các mạch kích, mạch cầu đều được tích hợp sẵn. L298D có điện áp danh nghĩa cao (lớn  nhất 50V) và dòng điện danh nghĩa lớn hơn 2A nên rất thích hợp cho các các ứng dụng công suất nhỏ như các động cơ DC loại nhỏ và vừa. Vì là loại “all in one” nên là lựa chọn hoàn hảo cho những người chưa có nhiều kinh nghiệm làm mạch điện tử. Trong bài học này tôi dùng chip L298D để làm driver cho motor. Hình 4 thể hiện mô hình thật của chip và cấu trúc bên trong chip.



Hình 4. Chip L298D

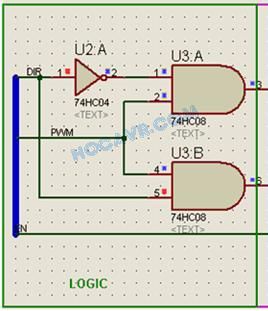
     Hình phía trên là hình dáng bên ngoài và tên gọi các chân của L298D. Hình phía dưới là cấu trúc bên trong chip. Có 2 mạch cầu H trên mỗi chip L298D nên có thể điều khiển 2 đối tượng chỉ với 1 chip này. Mỗi mạch cầu bao gồm 1 đường nguồn Vs (thật ra là đường chung cho 2 mạch cầu), một đường  current sensing (cảm biến dòng), phần cuối của mạch cầu H không được nối với GND mà bỏ trống cho người dùng nối một điện trở nhỏ gọi là sensing resistor. Bằng cách đo điện áp rơi trên điện trở này chúng ta có thể tính được dòng qua điện trở, cũng là dòng qua động cơ (xem hình 4). Mục đích chính của việc đo dòng điện qua động cơ là để xác định các trường hợp nguy hiểm xảy ra trong mạch, ví dụ quá tải. Nếu việc đo dòng động cơ không thật sự cần thiết bạn có thể nối đường current sensing này với GND (trong mạch điện của bài này, tôi nối chân current sensing với GND). Động cơ sẽ được nối với 2 đường OUT1, OUT2 (hoặc OUT3, OUT4 nếu dùng mạch cầu bên phải). Một chân En (EnA và EnB cho 2 mạch cầu) cho phép mạch cầu hoạt động, khi chân En được kéo lên mức cao, mạch cầu sẵn sang hoạt động. Các đường kích mỗi bên của mạch cầu được kết hợp với nhau và nhưng mức điện áp ngược nhau do một cổng Logic NOT. Bằng cách này chúng ta có thể tránh được trường hợp 2 transistor ở cùng một bên được kích cùng lúc (ngắn mạch). Như vậy, sẽ có 2 đường kích cho mỗi cầu H gọi là In1 và In2 (hoặc In3, In4). Để motor hoạt động chúng ta phải kéo 1 trong 2 đường kích này lên cao trong khi đường kia giữ ở mức thấp, ví dụ In1=1, In2=0. Khi đảo mức kích của 2 đường In, động cơ sẽ đảo chiều quay. Tuy nhiên, do L298D không chỉ được dùng đề đảo chiều động cơ mà còn điều khiển vận tốc động cơ bằng PWM, các đường In cần được “tổ hợp lại” bằng các cổng Logic (xem phần tiếp theo). Ngoài ra, trên chip L298D còn có các đường Vss cấp điện áp cho phần logic (5V) và GND chung cho cả logic và motor.   
     Trong thực tế, công suất thực mà L298D có thể tải nhỏ hơn so với giá trị danh nghĩa của nó (V=50V, I=2A). Để tăng dòng điện tải của chip lên gấp đôi, chúng ta có thể nối 2 mạch cầu H song song với nhau (các chân có chức năng như nhau của 2 mạch cầu được nối chung).

**II. Mạch logic cho L298D**

      Thông thường, khi thiết kế một mạch driver cho motor người ta thường dành 3 đường điều khiển đó là PWM dùng điều khiển vận tốc, DIR điều khiển hướng và En cho phép mạch hoạt động. Chip L298D đã có sẵn đường En nhưng 2 đường điều khiển In1 và In2 không thật sự chức năng như chúng ta mong muốn. Vì thế, chúng ta sẽ thiết kế một mạch logic phụ với 2 ngõ vào là PWM và DIR trong khi 2 ngõ ra là 2 đường điều khiển In1 và In2. Bảng chân trị của mạch logic cần thiết kế được trình bày trong bảng 1.  
Bảng 1. bảng chân trị của mạch logic cho driver L298D.

|  |  |  |  |
| --- | --- | --- | --- |
| PWM | DIR | In1 | In2 |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

      Từ bảng chân trị này, chúng ta có thể viết hàm bool cho 2 ngõ In1 và In2:  
      In1=PWM.NOT(DIR)  
      In2=PWM.DIR  
      Mạch logic vì thế sẽ có dạng như trong hình 5.

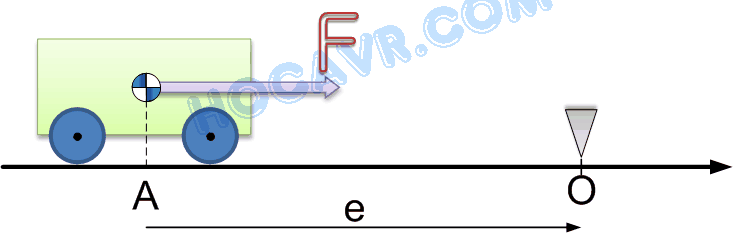


Hình 5. Mạch logic cho L239

     Tôi sẽ không giải thích chi tiết phần này, tuy nhiên điều bạn cần nắm là với mạch logic này, đường DIR có chức năng đảo chiều động cơ trong khi đường PWM điều khiển vận tốc động cơ bằng tín hiệu PWM.

**V. Giải thuật điều khiển PID**

      PID là cách viết tắc của các từ Propotional (tỉ lệ), Integral (tích phân) và Derivative (đạo hàm). Tuy xuất hiện rất lâu nhưng đến nay PID vẫn là giải thuật điều khiển được dùng nhiều nhất trong các ứng dụng điều khiển tự động. Để giúp bạn có cái hiểu rõ hơn bản chất của giải thuật PID tôi sẽ dùng một ví dụ điều khiển vị trí của một car (xe) trên đường thẳng. Giả sử bạn có một xe (đồ chơi...) có gắn một động cơ DC. Động cơ sinh ra một lực để đẩy xe chạy tới hoặc lui trên một đường thẳng như trong hình 6.

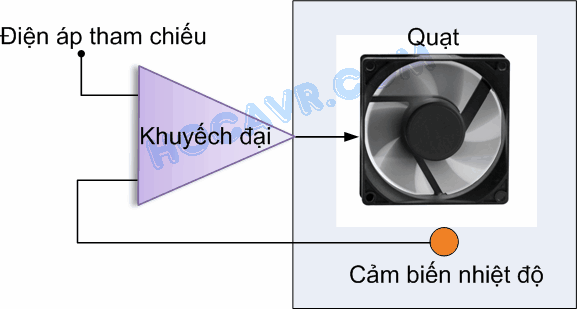


Hình 6. Ví dụ điều khiển vị trí xe trên đường thẳng

     Gọi F là lực do động cơ tạo ra điều khiển xe. Ban đầu xe ở vị trí A, nhiệm vụ đặt ra là điều khiển lực F (một cách tự động) để đẩy xe đến đúng vị trí O với các yêu cầu: chính xác (accurate), nhanh (fast response), ổn định (small overshot).   
     Một điều rất tự nhiên, nếu vị trí hiện tại của xe rất xa vị trí mong muốn (điểm O), hay nói cách khác sai số(error) lớn, chúng ta cần tác động lực F lớn để nhanh chóng đưa xe về O. Một cách đơn giản để công thức hóa ý tưởng này là dùng quan hệ tuyến tính:  
     F=Kp\*e                                                                                                                            (1)  
     Trong đó Kp là một hằng số dương nào đó mà chúng ta gọi là hệ số P (Propotional gain), e là sai số cần điều khiển tức khoảng cách từ điểm O đến vị trí hiện tại của xe. Mục tiêu điều khiển là đưa e tiến về 0 càng nhanh càng tốt. Rõ ràng nếu Kp lớn thì F cũng sẽ lớn và xe rất nhanh chóng tiến về vị trí O. Tuy nhiên, lực F quá lớn sẽ gia tốc cho xe rất nhanh (định luật II của Newton: F=ma). Khi xe đã đến vị trí O (tức e=0), thì tuy lực F=0 (vì F=Kp\*e=F=Kp\*0) nhưng do quán tính xe vẫn tiếp tục tiến về bên phải và lệch điểm O về bên phải, sai số e lại trở nên khác 0, giá trị sai số lúc này được gọi là overshot (vượt quá). Lúc này, sai số e là số âm, lực F lại xuất hiện nhưng với chiều ngược lại để kéo xe về lại điểm O. Nhưng một lần nữa, do Kp lớn nên giá trị lực F cũng lớn và có thể kéo xe lệch về bên trái điểm O. Quá trình cứ tiếp diễn, xe cứ mãi dao động quanh điểm O. Có trường hơp xe dao động càng ngày xàng xa điểm O. Bộ điều khiển lúc này được nói là không ổn định. Một đề xuất nhằm giảm overshot của xe là sử dụng một thành phần “thắng” trong bộ điều khiển. Sẽ rất lý tưởng nếu khi xe đang ở xa điểm O, bộ điều khiển sinh ra lực F lớn nhưng khi xe đã tiến gần đến điểm O thì thành phần “thắng” sẽ giảm tốc độ xe lại. Chúng ta đều biết khi một vật dao động quanh 1 điểm thì vật đó có vận tốc cao nhất ở tâm dao động (điểm O). Nói một cách khác, ở gần điểm O sai số e của xe thay đổi nhanh nhất (cần phân biệt: e thay đổi nhanh nhất  không phải e lớn nhất). Mặt khác, tốc độ thay đổi của e có thể tính bằng đạo hàm của biến này theo thời gian. Như vậy, khi  xe từ A tiến về gần O, đạo hàm của sai số e tăng giá trị nhưng ngược chiều của lực F (vì e đang giảm nhanh dần). Nếu sử dụng đạo hàm làm thành phần “thắng” thì có thể giảm được overshot của xe. Thành phần “thắng” này chính là thành phần D (Derivative) trong bộ điều khiển PID mà chúng ta đang khảo sát. Thêm thành phần D này vào bộ điều khiển P hiện tại, chúng ta thu được bộ điều khiển PD nhu sau:  
     F=Kp\*e + Kd\*(de/dt)                                                                                                        (2)  
     Trong đó (de/dt) là vận tốc thay đổi của sai số e và Kd là một hằng số không âm gọi là hệ số D (Derivative gain).   
     Sự hiện diện của thành phần D làm giảm overshot của xe, khi xe tiến gần về O, lực F gồm 2 thành phần Kp\*e > =0 (P) và Kd\*(de/dt) <=0 (D). Trong một số trường hợp thành phần D có giá trị lớn hơn thành phần P và lực F đổi chiều, “thắng” xe lại, vận tốc của xe vì thế giảm mạnh ở gần điểm O. Một vấn đề nảy sinh là nếu thành phần D quá lớn so với thành phần P hoặc bản thân thành phần P nhỏ thì khi xe tiến gần điểm O (chưa thật sự đến O), xe có thể dừng hẳn, thành phần D bằng 0 (vì sai số e không thay đổi nữa), lực F = Kp\*e. Trong khi Kp và e lúc này đều nhỏ nên lực F cũng nhỏ và có thể không thắng được lực ma sát tĩnh. Bạn hãy tưởng tượng tình huống bạn dùng sức của mình để đẩy một xe tải nặng vài chục tấn, tuy lực đẩy tồn tại nhưng xe không thể di chuyển. Như thế, xe sẽ đứng yên mãi dù sai số e vẫn chưa bằng 0. Sai số e trong tình huống này gọi là steady state error (tạm dịch là sai số trạng thái tĩnh). Để tránh steady state error, người ta thêm vào bộ điều khiển một thành phần có chức năng “cộng dồn” sai số. Khi steady state error xảy ra, 2 thành phần P và D mất tác dụng, thành phần điều khiển mới sẽ “cộng dồn” sai số theo thời gian và làm tăng lực F theo thời gian. Đến một lúc nào đó, lực F đủ lớn để thắng ma sát tĩnh và đẩy xe tiến tiếp về điểm O. Thành phần “cộng dồn” này chính là thành phần I (Integral - tích phân) trong bộ điều khiển PID. Vì chúng ta điều biết, tích phân một đại lượng theo thời gian chính là tổng của đại lượng đó theo thời gian. Bộ điều khiển đến thời điểm này đã đầy đủ là PID:  
     F=Kp\*e + Kd\*(de/dt)+Ki\*∫edt                                                                                                            (3)  
     (chú ý: ∫edt là tích phân của biến e theo t)

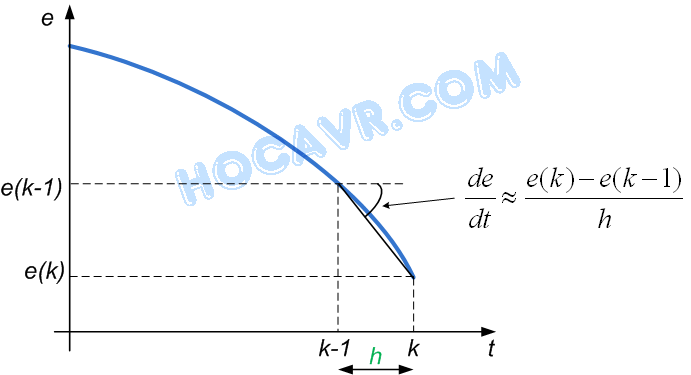
     Như vậy, chức năng của từng thành phần trong bộ điều khiển PID giờ đã rõ. Tùy vào mục đích và đối tượng điều khiển mà bộ điều khiển PID có thể được lượt bớt để trở thành bộ điều khiển P, PI hoặc PD. Công việc chính của người thiết kế bộ điều khiển PID là chọn các hệ số Kp, Kd và Ki sao cho bộ điều khiển hoạt động tốt và ổn định (quá trình này gọi là PID gain tuning). Đây không phải là việc dễ dàng vì nó phụ thuộc vào nhiều yếu tố. Tôi tóm tắt một kinh nghiệm cơ bản khi chọn các hệ số cho PID như sau:  
     - Chọn Kp trước: thử bộ điều khiển P với đối tượng thật (hoặc mô phỏng), điều chỉnh Kp sao cho thời gian đáp ứng đủ nhanh, chấp nhận overshot nhỏ.  
     - Thêm thành phần D để loại overshot, tăng Kd từ từ, thử nghiệm và chọn giá trị thích hợp. Steady state error có thể sẽ xuất hiện.  
     - Thêm thành phần I để giảm steady state error. Nên tăng Ki từ bé đến lớn để giảm steady state error đồng thời không để cho overshot xuất hiện trở lại.  
     Có một phương pháp rất phổ biến dùng để chọn các hệ số cho bộ điều khiển PID gọi là Ziegler–Nichols, bạn quan tâm có thể tự tìm hiểu thêm.

**Điều khiển PID số**  
     Công thức của bộ điều khiển PID trình bày trong (3) là dạng hàm liên tục của biến e, trong đó có cả thành phần tuyến tính, đạo hàm và tích phân. Tuy nhiên, hệ thống máy tính và vi điều khiển lại là hệ thống số. Muốn xây dựng bộ điều khiển PID trên máy tính hay trên vi điều khiển chúng ta phải biết cách xấp xỉ phương trình liên tục thành dạng rời rạc. Để thực hiện “số hóa” bộ điều khiển PID trước hết tôi nói sơ qua thế nào là hệ thống số (digital) so với hệ thống liên tục hay hệ thống tương tự (analog). Hãy quan sát hệ thống điều chỉnh nhiệt độ đơn giản như trong hình 7.



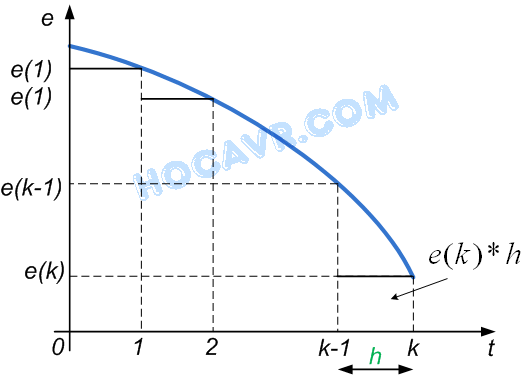
Hình 7. Tự động điều chỉnh nhiệt độ

      Giả sử chúng ta cần điều chỉnh nhiệt độ trong phòng ở một mức nào đó (tùy theo giá trị tham chiếu) bằng quạt. Cảm biến đo nhiệt độ và hồi tiếp về bộ khuyếch đại vi sai (so sánh và khuyếch đại). Nếu có sai số giữa giá trị tham chiếu và giá trị đo từ cảm biếm, bộ khuyếch đại vi sai sẽ tự động khuyếch đại sai số này và làm tăng hay giảm vận tốc của quạt để điều chỉnh nhiệt độ. Quá trình này xảy ra một cách liên tục. Bộ khuyếch đại vi sai trong trường hợp này chính là bộ điều khiển tương tự (analog controller). Bộ khuyếch đại này là một mạch điện tử thông thường như Opamp chẳng hạn. Nếu chúng ta thay bộ khuyếch đại này bằng một vi điều khiển AVR thì quá trình hiệu chỉnh không còn xảy ra liên tục nữa mà theo một chu kỳ nào đó. Ví dụ cứ mỗi 10 ms chúng ta đọc giá trị từ cảm biến một lần để tính toán sai số và xuất giá trị điều khiển quạt. Bộ điều khiển do AVR thực hiện gọi là bộ điều khiển số (digital controller) và khoảng thời gian 10ms này gọi là thời gian lấy mẫu (sampling time), đó là khoảng cách giữa 2 lần điều khiển liên tiếp. Rõ ràng thời gian lấy mẫu càng nhỏ (tấn số cao) thì việc hiệu chỉnh càng tiến gần đến sự “liên tục” và chất lượng điều khiển sẽ tốt hơn. Trong các bộ điều khiển số, thời gian lấy mẫu là một yếu tố rất quan trọng. Cần tính toán để thời gian này không quá lớn nhưng cũng đừng quá nhỏ, vì như thế sẽ hao phí thời gian thực thi.   
      Vì bộ điều khiển PID xây dựng trong AVR sẽ là bộ điều khiển số, chúng ta cần xấp xỉ công thức của bộ điều khiển này theo các khoảng thời gian rời rạc. Trước hết, thành phần P tương đối đơn giản vì đó là quan hệ tuyến tính Kp\*e, chúng ta chỉ cần áp dụng trực tiếp công thức này mà không cần bất kỳ xấp xỉ nào. Tiếp đến là xấp xỉ cho đạo hàm của biến e. Vì thời gian lấy mẫu cho các bộ điều khiển thường rất bé nên có thể xấp xỉ đạo hàm bằng sự thay đổi của e trong 2 lần lấy mẫu liên tiếp:  
      de/dt =(e(k) – e(k-1))/h.  
      Trong đó e(k) là giá trị hiện tại của e, e(k-1) là giá trị của e trong lần lấy mẫu trước đó và h là khoảng thời gian lấy mẫu (h là hằng số).

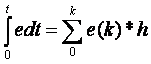


Hình 8. Xấp xỉ đạo hàm của biến sai số e

     Thành phần tích phân được xấp xỉ bằng diện tích vùng giới hạn bởi hàm đường biểu diễn của e và trục thời gian. Do việc tính toán tích phân không cần quá chính xác, chúng ta có thể dùng phương pháp xấp xỉ đơn giản nhất là xấp xỉ hình chữ nhật (sai số của phương pháp này cũng lớn nhất). Ý tưởng được trình bày trong hình 9.

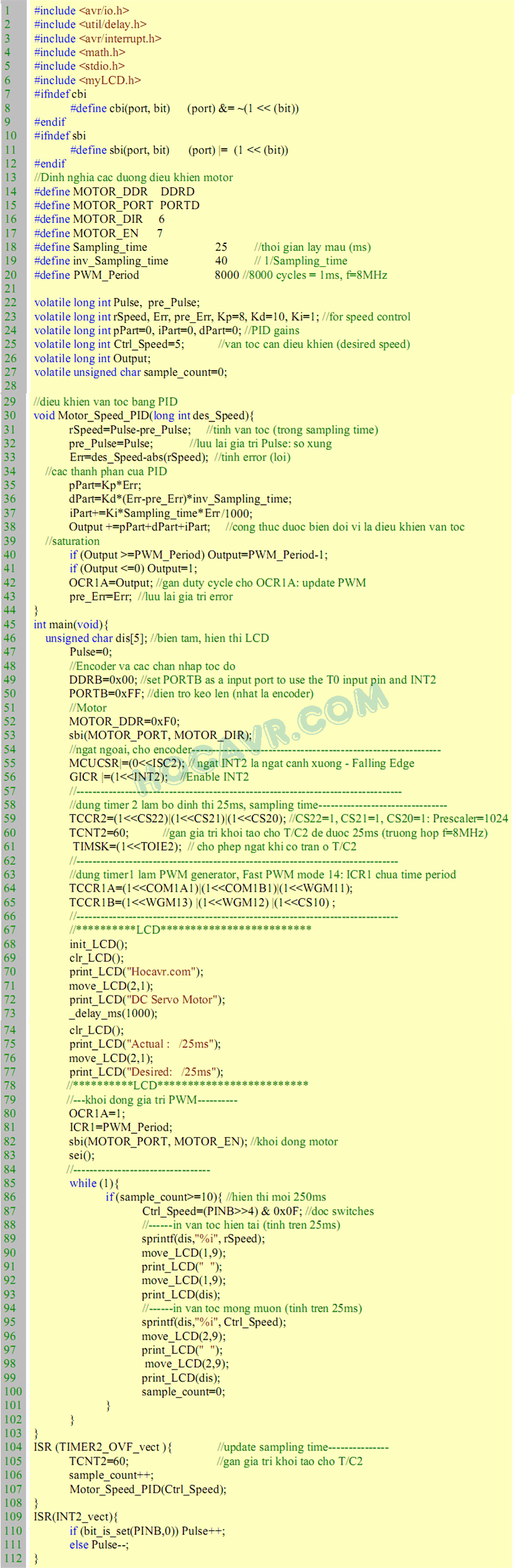


Hình 9. Xấp xỉ tích phân của biến sai số e

     Tích phân của biến e được tính bằng tổng diện tích các hình chữ nhật tại mỗi thời điểm đang xét. Mỗi hình chữ nhật có chiều rộng bằng thời gian lấy mẫu h và chiều cao là giá trị sai số e tại thời điểm đang xét. Tổng quát:  
                                                                                                                 (4)  
     Tổng hợp các xấp xỉ, công thức của bộ điều khiển PID số được trình bày trong (5)  
     http://www.hocavr.com/writing/docs/images/dcmotor/image020.gif                                                                       (5)  
     Trong đó u là đại lượng output từ bộ điều khiển. Để đơn giản hóa việc tính thành phần tích phân, chúng ta nên dùng phương pháp “cộng dồn” (hay đệ quy):  
     http://www.hocavr.com/writing/docs/images/dcmotor/image022.gif                                                                                                   (6)  
     Với I(k) là thành phần tích phân hiện tại và I(k-1) là thành phần tích phân trước đó.   
     Các công thức (5) và (6) rất dễ dàng để thực hiện bằng AVR. Do đó, đến lúc này chúng ta đã sẵn sàng để đưa ý tưởng vào lập trình cho chip.

**VI. Điều khiển DC Motor bằng AVR**

     Phần này chúng ta sẽ vận dụng tất cả phần lý thuyết giới thiệu ở trên để viết chương trình cho AVR. Mục đích là điều khiển vận tốc của DC Motor bằng giải thuật PID. Mạch điện mô phỏng được trình bày trong hình 1. Mô hình Motor dùng trong ví dụ là loại 12V có vận tốc không tải tối đa là 720rpm (revolute per minute) tức 20 vòng/s. Encoder dùng cho motor được chọn có độ phân giải 112 pulse/vòng. Kênh A của encoder được nối với ngắt ngoài INT2 để đếm xung, kênh B nối với chân PB0 (chân 1) của chip Atmega32 để xét hướng quay. Bốn switches được nối với 4 bit cao của PORTB để cài đặt vận tốc mong muốn cần điều khiển. Một Text LCD dùng hiển thị vận tốc thực của motor đọc từ Encoder (Actual speed) và vận tốc cài đặt (Desired speed). Do Text LCD được nối với PORTC nên nếu bạn muốn dùng chương trình này cho ứng dụng thật thì phải nạp lại fuses để vô hiệu hóa JTAG. Giải thuật PID số được vận hành bởi AVR trong thời gian lấy mẫu là 25ms. Timer 2 được dùng để tạo khoảng thời gian 25ms. Timer 1 (16 bit) là bộ tạo PWM điều khiển vận tốc động cơ. Toàn bộ nội dung chương trình được trình bày trong list 1.  
List 1. Điều khiển vận tốc động cơ DC



     Các dòng từ 14 đến 17 chúng ta định nghĩa các chân điều khiển DC Motor, chân DIR điều khiển hướng và EN kích hoạt hoặc dừng Motor (thực ra là dừng L298D). Do mục đích của chúng ta là điều khiển vận tốc động cơ, 2 chân này chỉ được “kích” một lần duy nhất trong chương trình chính (không cần đổi hướng quay của Motor). Dòng 18 định nghĩa thời gian lấy mẫu, Sampling\_time là 25 ms (.025s). Biến inv\_Sampling\_time ở dòng 19 là nghịch đảo của Sampling\_time, 1/0.025 = 40, vì đây cũng là hằng số, chúng ta định nghĩa trước để sau này không cần thực hiện phép nghịch đảo trong chương trình chính (tiết kiệm thời gian thực thi). PWM dùng điều khiển động cơ được chọn có tần số 1KHz nên chu kỳ la 1ms. Do chúng ta dùng nguồn xung giữ nhịp 8MHz, để tạo thời gian 1ms cần 8000 xung, giá trị này được định nghĩa trong dòng 20 và sẽ được gán cho thanh ghi ICR1 (TOP của PWM, xem lại bài Timer-Counter, Timer1, Fast PWM) trong chương trình chính (dòng 81). Các dòng code từ 22 đến 27 khai báo một số biến toàn cục dùng trong chương trình chính. Do các biến này sẽ được dùng cả trong trình phục vụ ngắt và chương trình chính nên cần khai báo đặc tính volatile, kiểu biến là long int tức số nguyên 32 bit (để tránh bị tràn khi tính toán sau này). Biến Pulse và pre\_Pulse là số xung hiện tại và lần lấy mẫu trước đó đọc từ encoder. Các biến trong dòng 23 và 24 dùng cho bộ điều khiển PID, biến Ctrl\_Speed là vận tốc mong muốn (set point) toàn cục và biến Output chứa giá trị tính được từ bộ điều khiển PID.   
     Trước khi đi tìm hiểu chương trình con chứa giải thuật PID, chúng ta sẽ khảo sát nội dung chương trình main và các trình phục vụ ngắt trước để hiểu tổng quan cách thức thực hiện. Chương trình chính bắt đầu từ dòng 45 và kết thúc ở dòng 103. Phần đầu của chương trình chính (ngoài vòng lặp while) khai báo và khởi tạo các module được sử dụng. 2 dòng 49 và 50 cài đặt hướng cho PORTB, do PORT này dùng đọc encoder và các switches chúng ta cần set nó là input và có điện trở kéo lên. Hai dòng 52 và 53 set hướng cho động cơ và sẽ giữ hướng này không đổi trong suất quá trình điều khiển sau này. Hai dòng 55 và 56 khai báo ngắt ngoài INT2 dùng đếm xung kênh A của encoder. Chú ý là INT2 chỉ có 2 mode là cạnh xuống và cạnh lên nên chỉ có 1 bit sense ISC2 để chọn mode. Bit ISC2 không nằm trong thanh ghi điều khiển MCUCR như các ngắt khác mà nằm trong thanh ghi điều khiển-trạng thái MCUCSR. Khi ISC2=0 thì chế độ ngắt cạnh xuống của INT2 được chọn (xem dòng 55). Sau đó INT2 được cho phép hoạt động ở dòng 56. Hãy tạm thời di chuyển đến dòng 109 để xem trình phục vụ ngắt INT2. Chức năng của INT2 trong bài này là “đếm xung encoder” vì thế trình phục vụ sẽ làm việc này. Khi có một ngắt INT2 xảy ra tức có 1 xung từ encoder vào thì trình phục vụ ngắt ISR(INT2 vect) tự động được gọi ra, dòng 110 trong trình phục vụ ngắt kiểm tra trạng thái chân PB0, tức kenh B của encoder. Nếu PB0=1 thì tăng biến xung đếm được Pulse lên 1, ngược lại nếu PB0=0 thì giảm Pulse đi 1 trong dòng 111. Quay về giải thích chương trình chính ở dòng 59, đây là các khai báo cho timer 2. Chúng ta sẽ dùng timer 2 tạo ra một khoảng thời gian lấy mẫu 25 ms, cứ sau 25 ms thì sẽ có ngắt tràn timer2 một lần và trong trình phục vụ ngắt tràn của timer2 chúng ta thực hiện tính toán PID. Dòng 59 chúng ta set các bit CS để chọn bộ chia tần số, bộ chia Prescaler=1024 được chọn vì 25 ms khá lớn so với thời gian 1 chu kỳ xung giữ nhịp (1/8 micro giây). Prescaler = 1024 nghĩa là sau 1024 nhịp của xung giữ nhịp, tức sau 128 micro giây (1024 \*1/8=128 us) thì thanh ghi giá trị TCNT2 mới tăng 1 đơn vị. Do chúng ta muốn tạo khoảng thời gian 25 ms tương đương 25000/128=195 đơn vị đếm của thanh ghi TCNT2, chúng ta sẽ gán giá trị khởi tạo cho TCNT2 là 255-195=60 (timer 2 sẽ tràn một lần khi TCNT2 đếm đến 255, xem lại bài Timer-Counter). Điều này thực hiện ở dòng 60 TCNT2=60. Dòng 61 cho phép ngắt tràn timer2. Hai dòng 64 và 65 khởi động Timer 1 dùng như một bộ tạo xung Fast PWM, mode 14, trong đó thanh ghi ICR1 chứa chu kỳ PWM và 2 thanh ghi OCR1A, OCR1B chứa duty cycle (khoảng ON) của PWM. Các dòng từ 68 đến 70 ghi texts lên LCD. Các dòng từ 80 đến 83 khởi động PWM cho DC Motor và cho phép ngắt toàn cục sei();. Trong vòng lặp while chủ yếu là công việc kiểm tra và hiển thị, biến sample\_count đếm số lần ngắt tràn timer2 xảy ra, nó được tăng 1 đơn vị khi có một ngắt tràn (xem dòng 106) tức sau 25ms. Dòng 86, chúng ta kiểm tra biến sample\_count, việc hiển thị chỉ đượcthực hiện mỗi 250 ms một lần (sample\_count=10) vì việc này tốn khá nhiều thời gian. Trong dòng 87 chúng ta kiểm tra các swiches để xem người dùng cho muốn thay đổi vận tốc tham chiếu cho điều khiển. Các dòng tiếp theo in biến rSpeed là số lượng xung đếm được từ encoder trong vòng 25 ms (cho tới hiện tại) ở dong 1 của LCD và in biến Ctrl\_Speed là số xung/25ms mà người dùng mong muốn motor đạt được. Nội dung quan trọng nhất của list 1, tuy nhiên, không nằm trong chương trình chính mà nằm ở các trình phục vụ ngắt  và chương trình con **Motor\_Speed\_PID(long int des\_Speed).**  
     Trước hết, trình phục vụ ngắt **ISR(TIMER2\_OVF\_vect)** được tự động gọi sau mỗi 25ms, trong trình này chúng ta cần set lại giá trị khởi động cho thanh ghi giá trị TCNT2 (xem lại bài Timer-counter) ở dòng 105. Sau đó tăng biến đếm sample\_count lên 1 (cùng cho việc đếm thời gian để hiển thị, đã nói ở trên). Cuối cùng là gọi chương trình con tính toán giải thuật **PID Motor\_Speed\_PID(long int des\_Speed)**. Đây là đoạn chương trình tính toán giải thuật PID và xuất giá trị điều khiển Motor. Hãy quay lại dòng 30 để tìm hiểu chương trình con này. Do biến Pulse chứa tổng số xung đọc từ encode (trong ISR(INT2\_vect) ), chúng ta lấy giá trị này trừ đi giá trị pre\_Pulse, tức số lượng xung ở thời điểm 25 ms trước đó, để thu được tổng số xung thu được trong 25 ms qua. Đây chính là vận tốc motor tính trên 25 ms:**rSpeed=Pulse-pre\_Pulse**. Sau khi tính được “vận tốc” rSpeed chúng ta gán lại giá trị Pulse cho pre\_Pulse để lần lấy mẫu sau dùng đến (dòng 32). Sai số vận tốc được đặt tên là Err, biến này được tính bằng bằng cách lấy vận tốc mong muốn trừ vận tốc hiện tại: **Err=des\_Speed-abs(rSpeed)** ở dòng 33. Dòng 34 tính thành phần P của bộ điều khiển**pPart=Kp\*Err**. Dòng 35 tính thành phần D của bộ điều khiển, như chúng ta đã thảo luận trong công thức (2) thì thành phần D được tính là: dPart=Kd\*(Err-pre\_Err)/Sampling\_time, trong đó pre\_Err là giá trị sai số ở lần lấy mẫu trước được lưu lại. Do 1/Sampling\_time = inv\_Sampling\_time nên chúng ta có thể thay dòng tính dPart bằng công thức trong dòng 35: **dPart=Kd\*(Err-pre\_Err)\*inv\_Sampling\_time**. Dòng 36 tính thành phần I (iPart), sử dụng phương pháp “cộng dồn” (đệ quy) chúng ta thu được iPart bằng iPart trước đó cộng với diện tích hình chữ nhật sai số hiện tại:**iPart+=Ki\*Sampling\_time\*Err/1000**. Chúng ta phải chia iPart cho 1000 vì Sampling\_time được tính theo ms trong khi đơn vị tính toán chuẩn trong là s. Cộng các thành phần này lại chúng ta được giá trị Output tổng hợp trong dòng 37. Tuy nhiên, theo lẽ thường thì công thức dòng 37 phải là Output=pPart+dPart+iPart nhưng ở đây lại là :**Output+=pPart+dPart+iPart**(để ý dấu + trước dấu =), nghĩa là Output được cộng dồn thay vì là tổng tức thời như chúng ta đã thảo luận trong phần giải thuật PID. Thật ra việc này cũng dễ hiểu. Trong bài toán điều khiển vị trí, khi sai số bằng 0 chúng ta có thể dừng bộ điều khiển (u=0) nhưng trong bài toán điều khiển vận tốc, khi sai số bằng 0 thì giá trị u vẫn phải được giữ là giá trị trước đó.Vì vậy, trong bài toán điều khiển vận tốc giá trị Output được cộng dồn thay vì gán trực tiếp, bạn phải ghi nhớ điều này trong các ứng dụng điều khiển của mình. Hai dòng 40 và 41 xét trường hợp bão hòa (saturation) khi Output vượt quá giới hạn cho phép của PWM (xén 2 đầu). Cuối cùng là gán giá trị tính toán được từ PID cho thanh ghi OCR1A để tăng hoặc giảm duty cycle của PWM trên chân OC1A (nối với PWM của Motor) và gán gái trị sai số Err cho biến pre\_Err cho lần lấy mẫu sau dùng đến.   
      Chạy mô phỏng: toàn bộ chương trình và cả mạch điện mô phỏng đã được tôi tạo sẵn. Người đọc chỉ cần đọc hiểu và chạy mô phỏng mạch điện. Chú khi chạy mô phỏng hãy thay đổi các switches để thay đổi vận tốc cần điều khiển. Gái trị vận tốc thực chất là số xung encoder trong 25 ms, người đọc hãy tự tính ra số vòng /s. Do mô hình motor trong phần mềm mô phỏng không hoàn hảo lắm nên đáp ứng bộ điều khiển hơi chậm, bạn có thể phải chờ một khoảng thời gian để thấy vận tốc Motor đạt đến vận tốc yêu cầu. Hay thay giá trị Kd trong dòng 23 thành 1 hoặc 0, biên dịch lại chương trình và mô phỏng để quan sát và so sánh ovetshot (sự vượt quá) của hệ thống.