BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC QUẢN LÝ VÀ CÔNG NGHỆ HẢI PHÒNG

ĐỒ ÁN TỐT NGHIỆP

NGÀNH ĐIỆN TỰ ĐỘNG CÔNG NGHIỆP

Sinh viên : Trịnh Đình Hảo

Giảng viên hướng dẫn : ThS Đỗ Anh Dũng

Hải Phòng -2022

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC QUẢN LÝ VÀ CÔNG NGHỆ HẢI PHÒNG

NGHIÊN CỨU THIẾT KẾ BỘ ĐIỀU KHIỂN TỐC ĐỘ ĐỘNG CƠ DC SERVO

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC HỆ CHÍNH QUY
NGÀNH ĐIỆN TỰ ĐỘNG CÔNG NGHIỆP

Sinh viên thực hiện: Trịnh Đình Hảo
Giảng viên hướng dẫn: ThS Đỗ Anh Dũng

Hải Phòng - 2022

BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC QUẢN LÝ VÀ CÔNG NGHỆ HẢI PHÒNG

NHIỆM VỤ ĐỀ TÀI TỐT NGHIỆP

Sinh viên : Trịnh Đình Hảo **- MSV** : 2013102007

Lớp : DCL 2401

Ngành: Điện Tự Động Công Nghiệp

Tên đề tài : Nghiên cứu thiết kế bộ điều khiển tốc độ động cơ DC Servo

NHIỆM VỤ ĐỀ TÀI

1.Nội dung và các yêu cầu cần giải quyết trong nhiệm vụ đề tài tốt nghiệp (về lý luận, thực tiễn, các số liệu cần tính toán và các bản vẽ).

……
……
……
……
……
……

2. Các số liệu cần thiết để tính toán.

……
……
……
……
……
……
……
……

3.Địa điểm thực tập tốt nghiệp.

……

CÁC CÁN BỘ HƯỚNG DẪN ĐỀ TÀI TỐT NGHIỆP

Họ và tên :

Học hàm, học vị :

Cơ quan công tác : Trường Đại học quản lý và công nghệ Hải Phòng

Nội dung hướng dẫn:

…….....

…….....

…….....

Đề tài tốt nghiệp được giao ngày 20 tháng 6 năm 2022

Yêu cầu phải hoàn thành xong trước ngày 10 tháng 9 năm 2022

Đã nhận nhiệm vụ ĐTTN	Đã giao nhiệm vụ ĐTTN
Sinh viên	*Giảng viên hướng dẫn*

Hải Phòng, ngày tháng năm 2022

TRƯỞNG KHOA

TS. Đoàn Hữu Chức

Cộng hòa xã hội chủ nghĩa Việt Nam

Độc lập - Tự do - Hạnh phúc

PHIẾU NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN TỐT NGHIỆP

Họ và tên giảng viên: Đỗ Anh Dũng

Đơn vị công tác: Trường Đại học Quản lý và Công nghệ Hải Phòng

Họ và tên sinh viên: Trịnh Đình Hảo

Chuyên ngành: Điện Tự Động Công Nghiệp

Nội dung hướng dẫn : Toàn bộ đề tài

1. Tinh thần thái độ của sinh viên trong quá trình làm đề tài tốt nghiệp

.....

.....

.....

.....

2. Đánh giá chất lượng của đồ án/khóa luận (so với nội dung yêu cầu đã đề ra trong nhiệm vụ Đ.T.T.N, trên các mặt lý luận, thực tiễn, tính toán số liệu...)

.....

.....

.....

3. Ý kiến của giảng viên hướng dẫn tốt nghiệp

Được bảo vệ ☐ Không được bảo vệ ☐ Điểm hướng dẫn ☐

Hải Phòng, ngày.....tháng.....năm 2022

Giảng viên hướng dẫn

(ký và ghi rõ họ tên)

Cộng hòa xã hội chủ nghĩa Việt Nam

Độc lập - Tự do - Hạnh phúc

PHIẾU NHẬN XÉT CỦA GIẢNG VIÊN CHẤM PHẢN BIỆN

Họ và tên giảng viên ……

Đơn vị công tác:.....

Họ và tên sinh viên: .....**Chuyên ngành:**.....

Đề tài tốt nghiệp: .....

.....

1. Phần nhận xét của giảng viên chấm phản biện

.....

.....

.....

.....

2. Những mặt còn hạn chế

.....

.....

.....

.....

3. Ý kiến của giảng viên chấm phản biện

Được bảo vệ ☐ Không được bảo vệ ☐ Điểm phản biện ☐

Hải Phòng, ngày.....tháng.....năm 2022

Giảng viên chấm phản biện

(ký và ghi rõ họ tên)

MỤC LỤC

CHƯƠNG 1 CẤU TẠO VÀ NGUYÊN LÝ HOẠT ĐỘNG ĐỘNG CƠ DC SERVO.

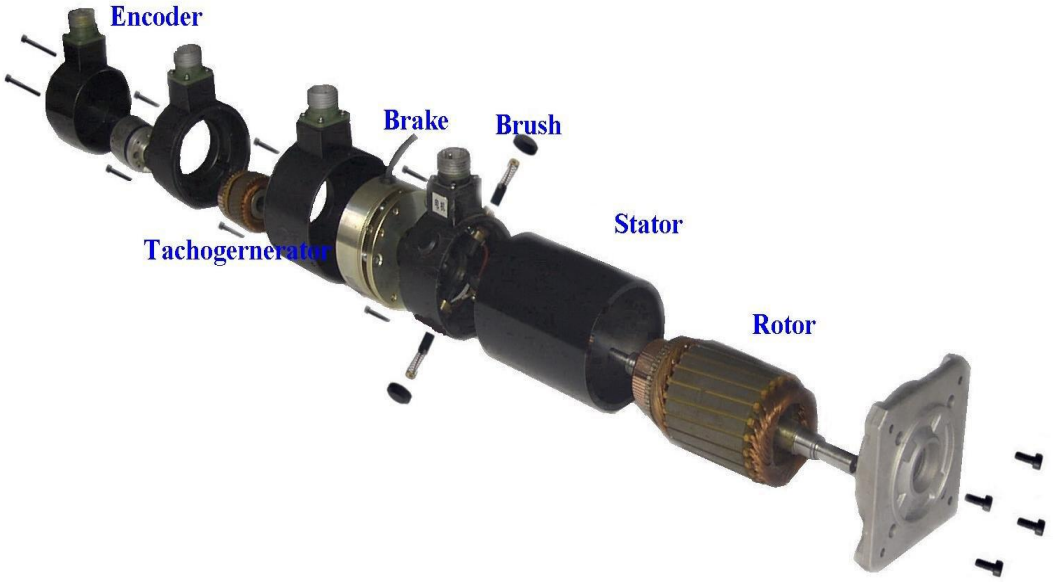
1.3 Cấu tạo động cơ DC SERVO.

Động cơ DC và động cơ bước vốn là những hệ hồi tiếp vòng hở - ta cấp điện để động cơ quay nhưng chúng quay bao nhiêu thì ta không biết, kể cả đối với động cơ bước là động cơ quay một góc xác định tùy vào số xung nhận được. Việc thiết lập một hệ thống điều khiển để xác định những gì ngăn cản chuyển động quay của động cơ hoặc làm động cơ không quay cũng không dễ dàng.

Hình 1.1: Một động cơ DC servo trong thực tế

Mặt khác, động cơ servo được thiết kế cho những hệ thống hồi tiếp vòng kín. Tín hiệu ra của động cơ được nối với một mạch điều khiển. Khi động cơ quay, vận tốc và vị trí sẽ được hồi tiếp về mạch điều khiển này. Nếu có bất kỳ lý do nào ngăn cản chuyển động quay của động cơ, cơ cấu hồi tiếp sẽ nhận thấy tín hiệu ra chưa đạt được vị trí mong muốn. Mạch điều khiển tiếp tục chỉnh sai lệch cho động cơ đạt được điểm chính xác.

Động cơ servo có nhiều kiểu dáng và kích thước, được sử dụng trong nhiều máy khác nhau, từ máy tiện điều khiển bằng máy tính cho đến các mô hình máy bay và xe hơi.



Hình 1.2: Các thành phần của động cơ DC servo.

Một động cơ DCservo tiêu biểu gồm có các thành phần chính sau:

- **Stator**: được gắn liền với vỏ động cơ
- **Rotor**: là thành phần tạo chuyển động quay
- **Chổi than và vành góp**: giúp đưa điện vào Rotor
- **Encoder:** hay còn gọi là bộ mã hóa vòng quay, phản hồi xung, đơn vị tính (xung/vòng)

Ngoài ra, DC servo còn có thể có thêm các thành phần sau:

- **Phanh điện từ:** giúp hãm động cơ trong trường hợp cần thiết
- **Tachometer :** là thành phần phản hồi tương tự, thực chất là một máy phát điện nhỏ, với điện áp phản hồi được tính bằng (vol/vòng quay)

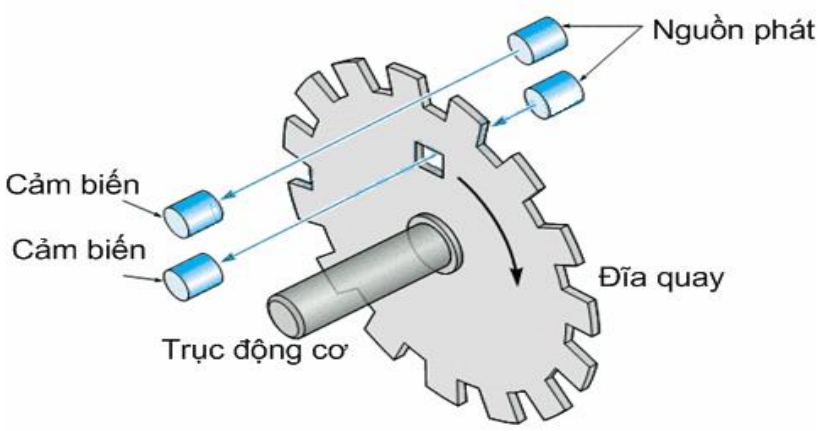
1.4 Nguyên lý điều khiển động cơ DC SERVO.

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, hoặc dùng biến trở xoay, hoặc dùng encoder. Trong đó 2 phương pháp đầu tiên là

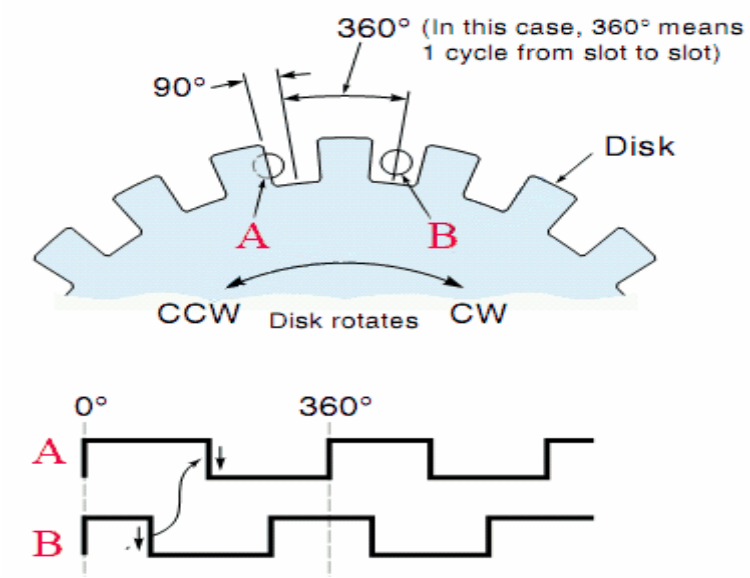
phương pháp analog và dùng optical encoder (encoder quang) thuộc nhóm phương pháp digital.

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 đa số.



Hình 1.3: Cấu tạo một encoder quang.

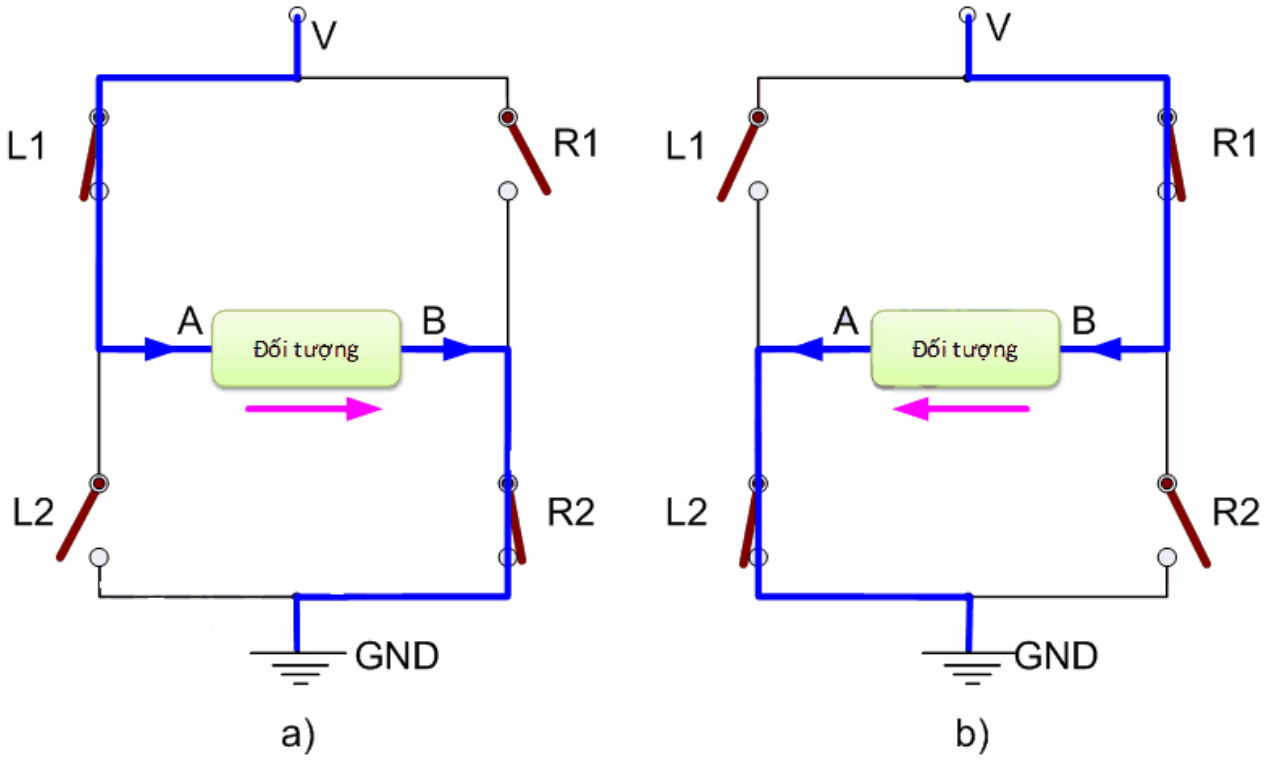
Encoder thường có 3 kênh (3 ngõ ra) bao gồm kênh A, kênh B và kênh I (Index). Trong hình trên chú ý rằng có một lỗ nhỏ bên phía trong của đĩa quay và một cặp phat-thu dành riêng cho lỗ nhỏ này. Đó là kênh I của encoder. Cữ mỗi lần motor quay được một vòng, lỗ nhỏ xuất hiện tại vị trí của cặp phát-thu, 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 1.2, 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 90^o. Bằng cách phối hợp kênh A và B người đọc sẽ biết chiều quay của động cơ. Hãy quan sát hình 3.



Hình 1.4: Hoạt động của một encoder quang.

Hình trên cùng trong hình 1.3 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).

Động cơ Dc servo được điều khiển bởi tín hiệu từ vi điều khiển theo nguyên lý điều khiển độ rộng xung (Pulse width modulation – PWM), sử dụng mạch cầu H



Hình 1.5: Hoạt động của mạch cầu H

Trong hình 1.5, hãy xem 2 đầu V và GND là 2 đầu (+) và (-) của ắc qui, "đối tượng" là động cơ DC mà chúng ta cần điều khiển, "đối tượng" này có 2 đầu A và B, mục đích điều khiển là cho phép dòng điện qua "đối tượng" theo chiều A đến B hoặc B đến A. Thành phần chính tạo nên mạch cầu H của chúng ta chính là 4 "khóa" L1, L2, R1 và R2 (L: Left, R:Right). Ở điều kiện bình thường 4 khóa này "mở", mạch cầu H không hoạt động.

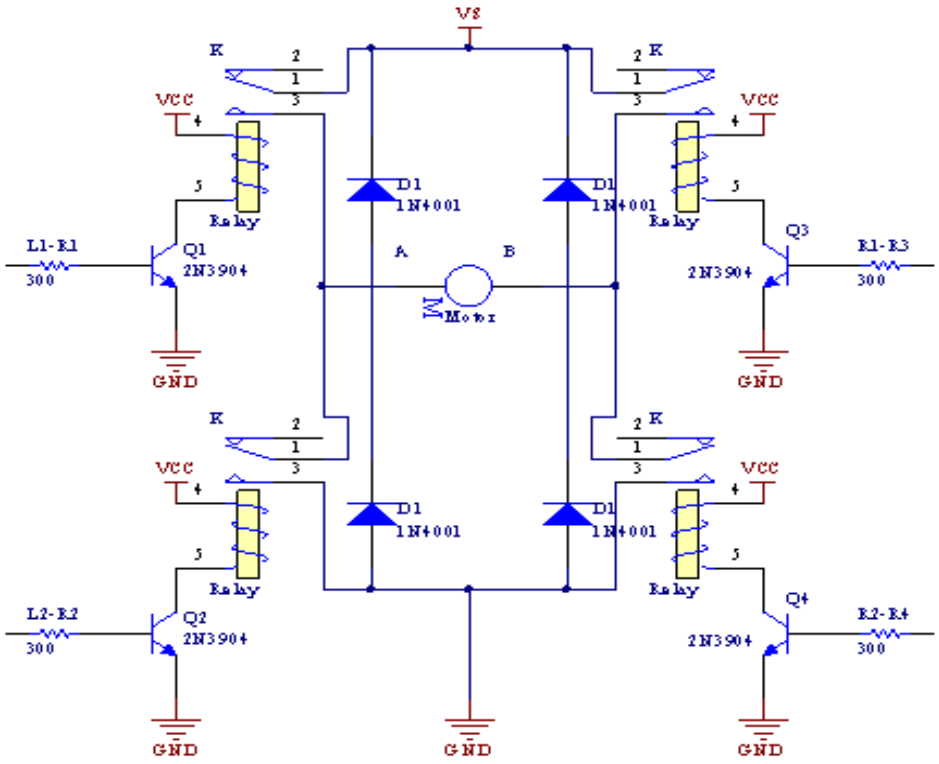
Giả sử bằng cách nào đó mà 2 khóa L1 và R2 được "đóng lại" (L2 và R1 vẫn mở), có một dòng điện chạy từ V qua khóa L1 đến đầu A và xuyên qua đối tượng đến đầu B của nó trước khi qua khóa R2 và về GND (như hình 2a). Như thế, với giả sử này sẽ có dòng điện chạy qua đối tượng theo chiều từ A đến B. Bây giờ hãy giả sử khác đi rằng R1 và L2 đóng trong khi L1 và R2 mở, dòng điện lại xuất hiện và lần này nó sẽ chạy qua đối tượng theo chiều từ B đến A như trong hình 2b (V->R1->B->A->L2->GND). Vậy là chúng ta có thể dùng mạch cầu H để đảo chiều dòng điện qua một "đối tượng" (hay cụ thể, đảo chiều quay động cơ)

Nếu đóng đồng thời 2 khóa ở cùng một bên (L1 và L2 hoặc R1 và R2) hoặc thậm chí đóng cả 4 khóa? Hiện tượng "ngắn mạch" (short circuit), V và GND gần như nối trực tiếp với nhau và hiển nhiên ắc qui sẽ bị hỏng hoặc nguy hiểm hơn là cháy nổ mạch xảy ra. Cách đóng các khóa như thế này sẽ làm hỏng mạch cầu H. Để tránh việc này xảy ra, người ta thường

dùng thêm các mạch logic để kích cầu H, chúng ta sẽ biết rõ hơn về mạch logic này trong các phần sau.

Giả thiết cuối cùng là 2 trường hợp các khóa ở phần dưới hoặc phần trên cùng đóng (ví dụ L1 và R1 cùng đóng, L2 và R2 cùng mở). Với trường hợp này, cả 2 đầu A, B của "đối tượng" cùng nối với một mức điện áp và sẽ không có dòng điện nào chạy qua, mạch cầu H không hoạt động. Đây có thể coi là một cách "thắng" động cơ (nhưng không phải lúc nào cũng có tác dụng).

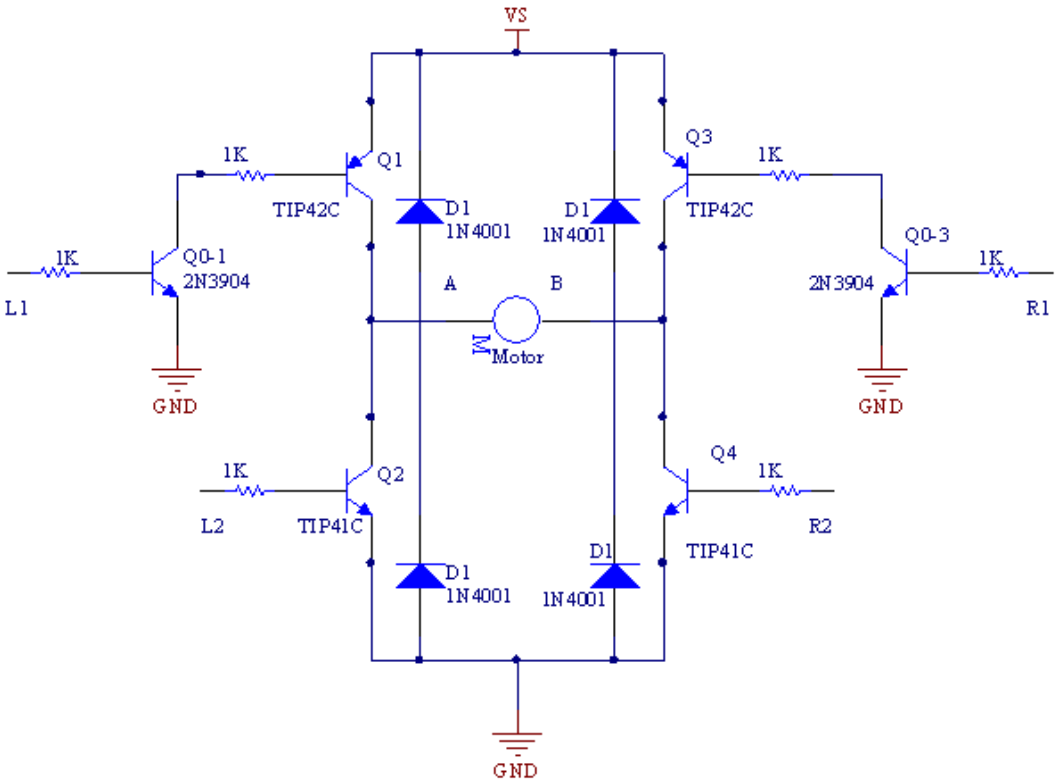
Đó là nguyên lý cơ bản của mạch cầu H. Như vậy thành phần chính của mạch cầu H chính là các "khóa", việc chọn linh kiện để làm các khóa này phụ thuộc vào mục đích sử dụng mạch cầu, loại đối tượng cần điều khiển, công suất tiêu thụ của đối tượng và cả hiểu biết, điều kiện của người thiết kế. Nhìn chung, các khóa của mạch cầu H thường được chế tạo bằng rờ le (relay), BJT (Bipolar Junction Transistor) hay MOSFET (Metal Oxide Semiconductor Field-Effect Transistor).



Hình 1.6: Mạch cầu H dùng Rơ le.

Trong mạch cầu H dùng rờ le ở hình 1.6, 4 diode được dùng để chống hiện tượng dòng ngược (nhất là khi điều khiển động cơ). Các đường kích solenoid không được nối trực tiếp với chip điều khiển mà thông qua các transistor, việc kích các transistor lại được thực hiện qua các điện trở.

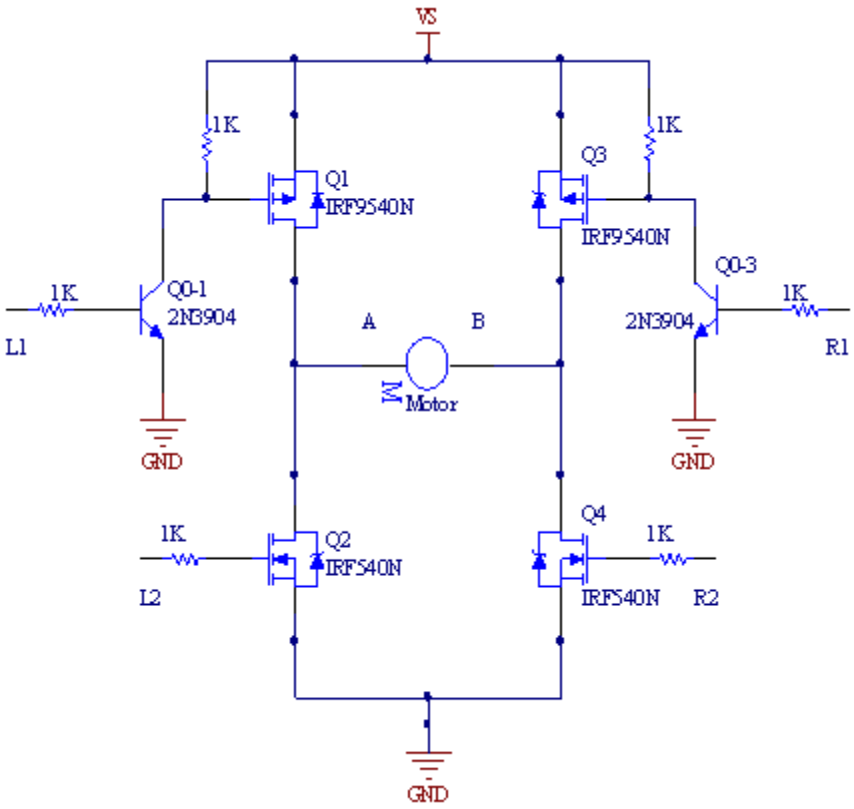
Mạch cầu H dùng rờ le có ưu điểm là dễ chế tạo, chịu dòng cao, đặc biệt nếu thay rờ le bằng các linh kiện tương đương như contactor, dòng điện tải có thể lên đến hàng trăm ampere. Tuy nhiên, do là thiết bị "cơ khí" nên tốc độ đóng/mở của rờ le rất chậm, nếu đóng mở quá nhanh có thể dẫn đến hiện tượng "dính" tiếp điểm và hư hỏng. Vì vậy, mạch cầu H bằng rờ le không được dùng trong phương pháp điều khiển tốc độ động cơ bằng PWM. Người ta thay thế rờ le trong mạch cầu H, bằng các tranzitor gọi là các "khóa điện tử" với khả năng đóng/mở lên đến hàng nghìn hoặc triệu lần trên mỗi giây



Hình 1.7: Mạch cầu H dùng BJT

Do BJT có thể được kích ở tốc độ rất cao nên ngoài chức năng đảo chiều, mạch cầu H dùng BJT có thể dùng điều khiển tốc độ motor bằng cách áp tín hiệu PWM vào các đường .

Nhược điểm lớn nhất của mạch cầu H dùng BJT là công suất của BJT thường nhỏ, vì vậy với motor công suất lớn thì BJT ít được sử dụng. Mạch điện kích cho BJT cần tính toán rất kỹ để đưa BJT vào trạng thái bão hòa, nếu không sẽ hỏng BJT. Mặt khác, điện trở CE của BJT khi bão hòa cũng tương đối lớn, BJT vì vậy có thể bị nóng…



Hình 1.8: Mạch cầu H dùng Mosfet

Hình trên minh họa một mạch cầu H dùng MOSFET điển hình với cặp IRF9540 và IRF540

Mạch cầu H dùng MOSFET, hoạt động tương tự như mạch cầu H dùng BJT, tuy nhiên do ưu điểm của các Mosfet là tốc độ đóng mở nhanh, dòng tải lớn do đó được dùng nhiều hơn trong thực tế.

2.3. Vi điều khiển AVR.

2.1.1. Giới thiệu về AVR.

AVR là họ Vi điều khiển khá mới trên thị trường cũng như đối với người sử dụng. Đây là họ vi điều khiển được chế tạo theo kiến trúc RISC (Reduced Instruction Set Computer) có cấu trúc khá phức tạp. Ngoài các tính năng như các họ vi điều khiển khác, nó còn tích hợp nhiều tính năng mới rất tiện lợi cho người thiết kế và lập trình.

Sự ra đời của AVR bắt nguồn từ yêu cầu thực tế là hầu hết khi cần lập trình cho vi điều khiển, thường dùng những ngôn ngữ bậc cao HLL (Hight Level Language) để lập trình ngay cả với loại chip xử lí 8 bit. Tuy nhiên khi biên dịch thì kích thước đoạn mã sẽ tăng nhiều so với dùng ngôn ngữ Assembly. Hãng Atmel nhận thấy rằng cần phải phát triển một cấu trúc đặc biệt để giãm thiểu sự chênh lệch kích thước mã đã nói trên. Và kết quả là họ vi điều khiển AVR ra đời với việc làm giãm kích thước đoạn mã khi biên dịch và thêm vào đó là thực hiện lệnh đúng chu kỳ máy với 32 thanh ghi tích lũy và đạt tốc độ nhanh hơn các họ vi điều khiển khác từ 4 đến 12 lần. Vì thế nghiên cứu AVR là một đề tài khá lý thú và giúp cho sinh viên biết thêm một họ vi điều khiển vào loại mạnh nhất hiện nay.

Vi điều khiển AVR do hãng Atmel (Hoa Kì) sản xuất được gới thiệu lần đầu năm 1996.

Họ vi điều khiển AVR là một họ vi điều khiển có cấu trúc hiện đại (so với 8051).

Có ba loại trong họ này đó là :

- Tinyavr.
- AVR (loại AVR).
- MegaAVR.

Hình 2.1. Các dòng AVR: tiny, AVR và AT mega.

Tất cả các thiết bị trong họ AVR đều có chung một tập lệnh, và tổ chức bộ nhớ giống nhau. Nhưng khi chuyển nghiên cứu từ một vi điều khiển AVR này sang loại khác thì thật là đơn giản. Cấu tạo AVR bao gồm: SRAM, EEPROM và giao tiếp SRAM mở rộng, bộ chuyển đổi tương tự số (ADC), cấu trúc nhiều tuyến, UART, USART…

2.1.2. Một số chíp AVR thông dụng.

AT90S1200 AT90S2313

AT90S2323 and AT90S2343 AT90S2333 and AT90S4433

AT90S4414 and AT90S8515 AT90S4434 and AT90S8535 AT90C8534

ATtiny10, ATtiny11 and ATtiny12 ATtiny15

ATtiny22 ATtiny26 ATtiny28

ATmega8/8515/8535 ATmega16 ATmega161 ATmega162 ATmega163 ATmega169

ATmega32 ATmega323 ATmega103

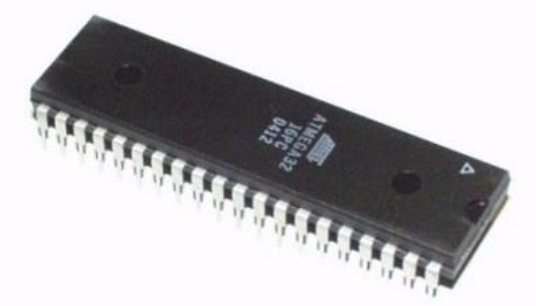
ATmega64/128/2560/2561 ATmega86RF401.

…

2.4. Chíp Atmega32.

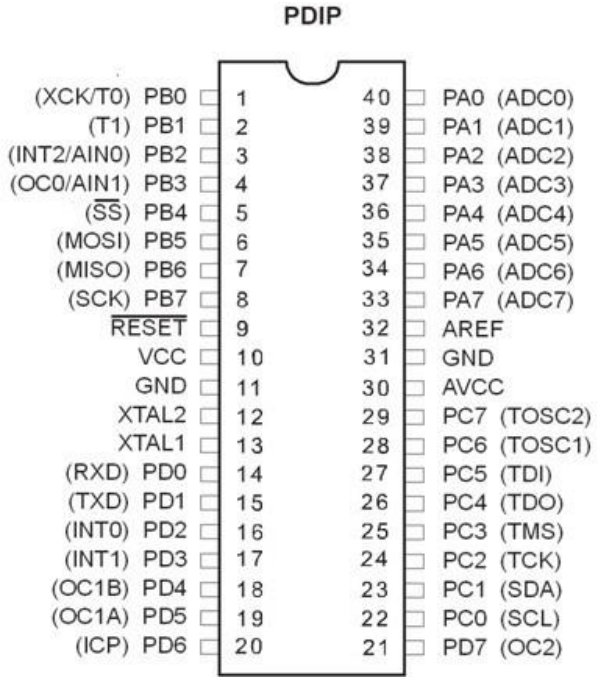
Atmega32 là vi điều khiển thuộc họ AVR của hãng Atmel, có 40 chân trong đó có 32 chân I/O, có 4 kênh điều xung PWM, sử dụng thạch anh ngoài 8MHz.

Nhân AVR kết hợp tập lệnh đầy đủ với 32 thanh ghi đa năng. Tất cả các thanh ghi liên kết trực tiếp với khối xử lý số học và logic (ALU) cho phép 2 thanh ghi độc lập được truy cập trong một lệnh đơn trong 1 chu kỳ đồng hồ. Kết quả là tốc độ nhanh gấp 10 lần các bộ vi điều khiển CISC thường. Chính vì điều đó em đã chon Atmega32 để làm đế tài nghiên cứu và ứng dụng.



Hình 2.2. Hình dạng thức tế ATMega32.

2.4.1. Cấu hình chân (pin configurations).



Hình 2.3. Cấu trúc chân của Atmega32.

2.4.2. Đặc tính của ATmega32.

- Được chế tạo theo kiến trúc RISC.
- Bộ lệnh gồm 118 lệnh, hầu hết đều thực thi chỉ trong một chu kì xung nhịp.
- 32x8 thanh ghi làm việc đa dụng.

- 32 KB Flash ROM lập trình được ngay trên hệ thống.

- Giao diện nối tiếp SPI cho phép lập trình ngay trên hệ thống.

- Cho phép 1000 lần ghi / xoá.

- Bộ EEPROM 1024 byte.

- Cho phép 100.000 ghi / xoá.

- Bộ nhớ SRAM 2 Kbyte.

- Bộ biến đổi ADC 8 kênh, 10 bit.

- 32 ngõ I/O lập trình được.

- Bộ truyền nối tiếp bất đồng bộ vạn năng UART.

- Vcc = 2.7V đến 6V.

- Tốc độ làm việc: 0 đến 16 Mhz.

- Tốc độ xử lí lệnh 16 MIPS ở 16 MHz (16 triệu lệnh trên giây).

- Bộ đếm thời gian thực (RTC) với bộ dao động và chế độ đếm tách biệt.

- 2 bộ Timer 8 bit và 2 bộ Timer 16 bit với chế độ so sánh và chia tần số tách biệt và chế độ bắt mẫu.

- Bốn kênh điều chế độ rộng xung PWM.

- Bộ định thời Watchdog lập trình được. Tự động reset khi treo máy.

- Bộ so sánh tương tự.

- Sáu chế độ ngủ: Chế độ rỗi (Idle), tiết kiệm điện (Power save), chế độ Power Down, chế độ ADC Noise Reduction, chế độ Standby và chế độ Extended Standby.

2.4.2.1. Mô tả ý nghĩa các chân (Pin descipsions).

- At mega32 gồm có 4 port: Port A, port B, port C và port D.

- Port A gồm 8 chân từ PA0 đến PA7: Là cổng vào tương tự cho chuyển đổi tương tự sang số. Nó cũng là cổng vào/ra hai hướng 8 bít trong trường hợp không sử sụng làm cổng chuyển đổi tương tự, có điện trở nối lên nguồn dương bên trong. Port A cung cấp đường địa chỉ dữ liệu vao/ra theo kiểu hợp kênh khi dùng bộ nhớ bên ngoài.

- Port B gồm 8 chân từ PB0 đến PB7: Là cổng vào/ra hai hướng 8 bít, có điện trở nối lên nguồn dương bên trong. Port B cung cấp các chức năng ứng với các tính năng đặc biệt của Atmega32.

- Port C gồm các chân từ PC0 đến PC7: Là cổng vào/ra hai hướng 8 bit, có điện trở nối lên nguồn dương bên trong, Port C cung cấp các địa chỉ lối ra khi sử dụng bộ nhớ bên ngoài và đồng thời cung cấp ứng với các tính năng đặc biệt của Atmega32.

- Port D gồm các chân từ PD0 đến PD7: Là cổng vào/ra hai hướng 8 bít, có điện trở nối lên nguồn dương bên trong. Port D cung cấp các chức năng ứng với các tính năng đặc biệt của Atmega32.

- Chân nguồn Vcc (chân số 10 và chân số 30): Điện áp nguồn nuôi của Atmega32 từ 4.5v đến 5.5v.

- Chân Reset (chân số 9): Lối vào đặt lại.

- Chân GND (chân số 11 và chân 31): Chân nối mát.

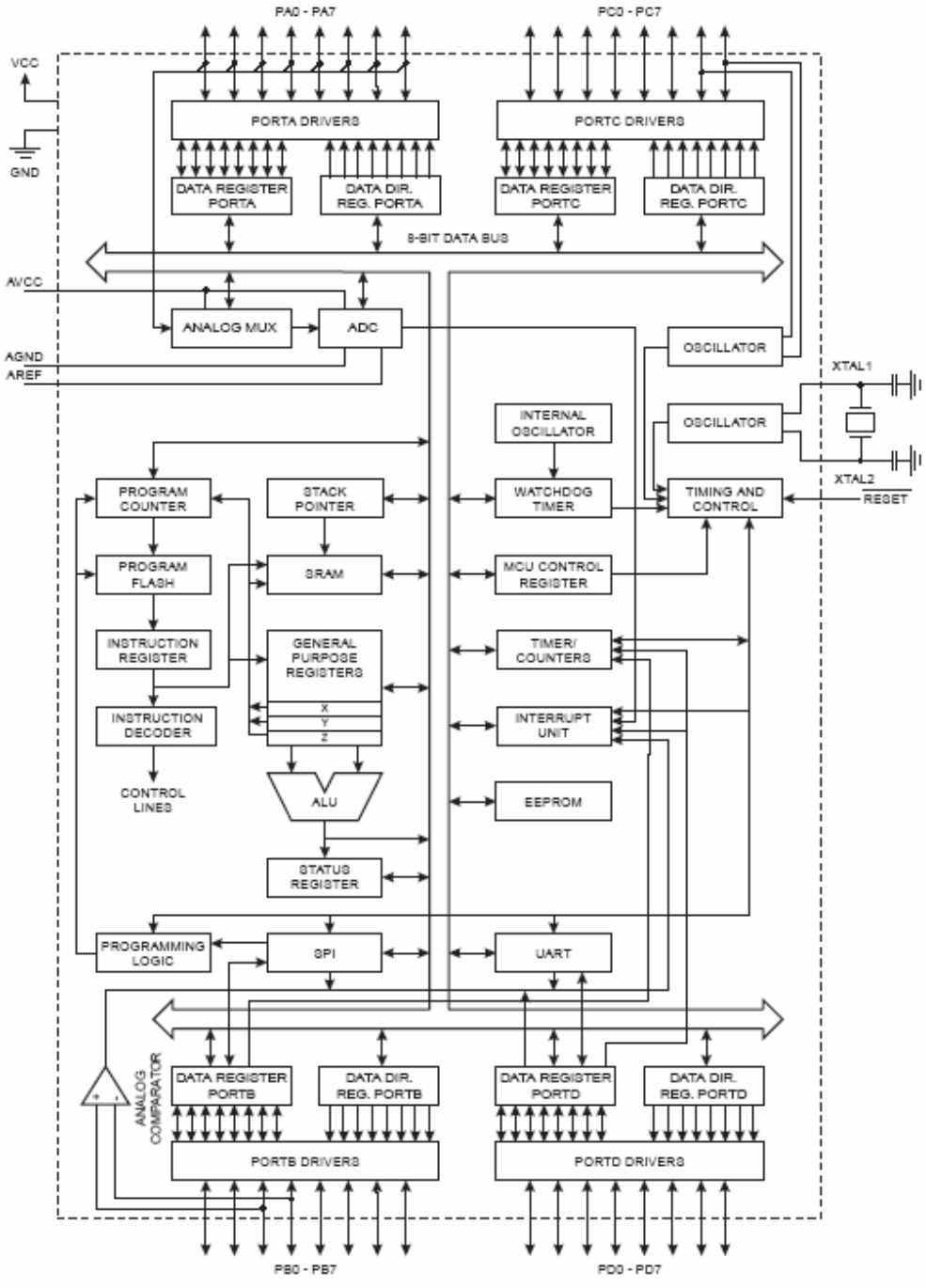
- Chân XTAL1, XTAL2 là hai chân nối thạch anh ngoài (chân số 12 và chân số 13). Atmega32 sử dụng thạch anh ngoài là 8MHz.

- Chân ICP (chân số 20): Là chân vào cho chức năng bắt tín hiệu cho bộ định thời/đếm 1.

- Chân OC1B (chân số 18): Là chân ra cho chức năng so sánh lối ra bộ định thời/đếm 1.

- Chân INT1(chân số 17): Chân ngõ vào ngắt.

2.4.2.2. Sơ đồ khối.

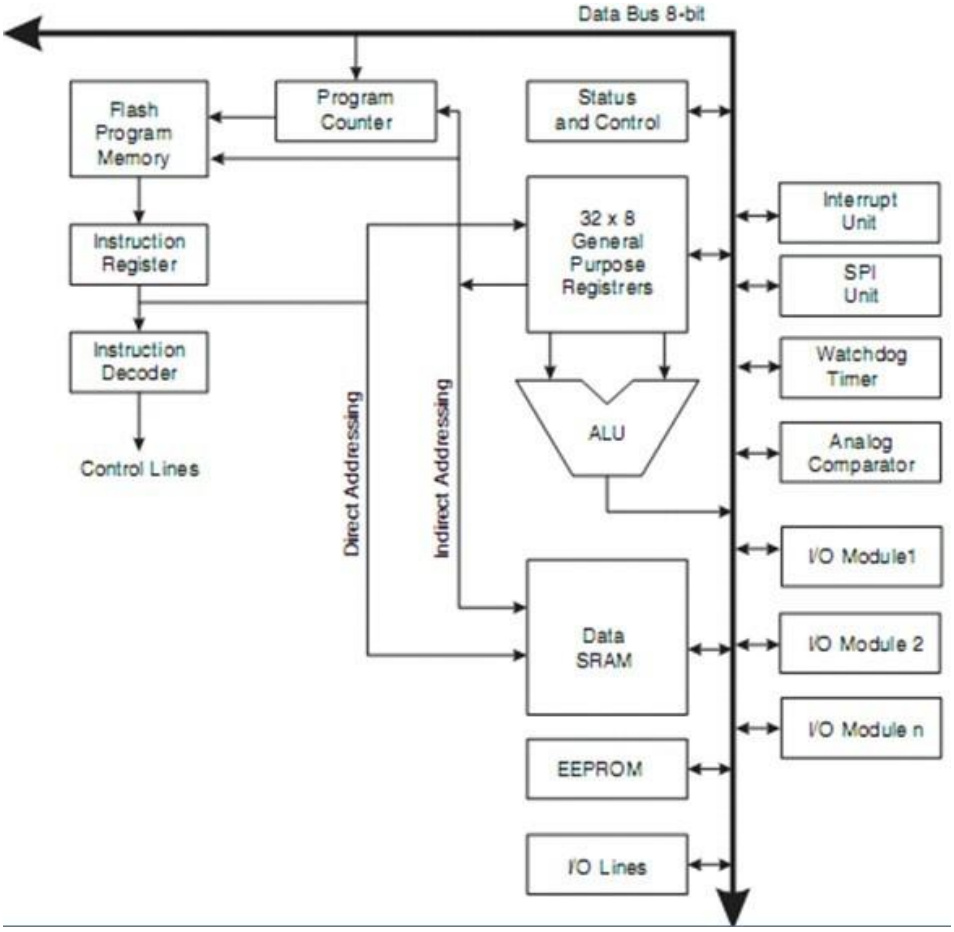


Hình 2.4. Sơ đồ khối Atmega32.

2.4.2.3. Cấu trúc nhân AVR.

Phần cốt lõi của AVR kết hợp tập lệnh phong phú về số lượng với 32 thanh ghi làm việc đa năng. Toàn bộ 32 thanh ghi đều được kết nối trực tiếp với ALU (Arithmeetic Logic Unit), cho phép truy cập hai thanh ghi độc lập bằng một chu kỳ xung nhịp. Kiến trúc đạt được có tốc độ xử lý nhanh gấp 10 lần vi điều khiển kiểu dạng CISC thông thường.

2.4.2.4. Cấu trúc tổng quát.



Hình 2.5. Sơ đồ cấu trúc CPU của Atmega32.

AVR sử dụng cấu trúc Harvard, tách riêng bộ nhớ và các bus cho chương trình và dữ liệu. Các lệnh được thực hiện chỉ trong một chu kỳ xung clock. Bộ nhớ chương trình được lưu trong bộ nhớ Flash.

2.4.2.5. ALU.

ALU làm việc trực tiếp với các thanh ghi chức năng chung. Các phép toán được thực hiện trong một chu kỳ xung clock. Hoạt động của ALU được chia làm 3 loại: Đại số, logic và theo bit.

2.4.2.6. Thanh ghi trạng thái.

Đây là thanh ghi trạng thái có 8 bit lưu trữ trạng thái của ALU sau các phép tính số học và logic.

BIT S3F(S5F)

7	6	5	4	3	2	1	0
I	**T**	**H**	**S**	**V**	**N**	**Z**	**C**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

Read/Write								
Intial Value								

Hình 2.6. Thanh ghi trạng thái SREG.

- C: Carry Flag; Cờ nhớ (Nếu phép toán có cờ nhớ sẽ được thiết lập).

- Z: Zero Flag; Cờ zero (Nếu kết quả phép toán bằng 0).

- N: Negative (Nếu kết quả phép toán là âm).

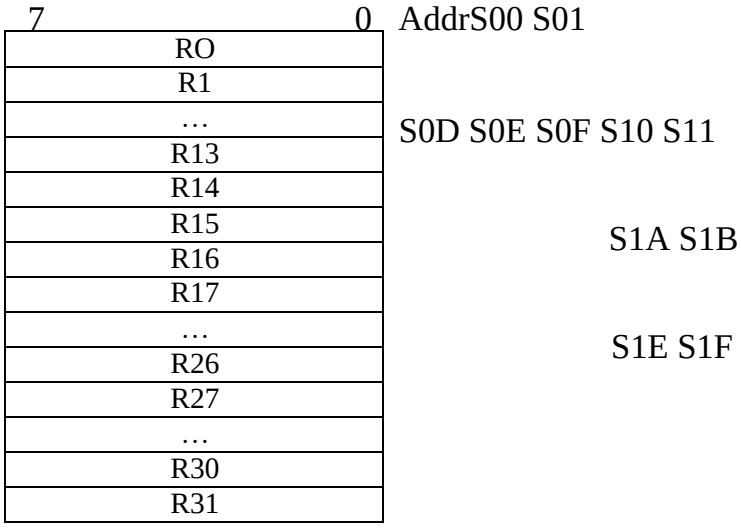
- V: Two'scomplement overflow (Cờ này được thiết lập khi tràn số bù 2) V, For signed tests (S=N XOR V) S:N.

- H: Half Carry Flag (Được sử dụng trong một số toán hạng sẽ được chỉ ra sau).

- T: Transfer bit used by BLD and BST intruction (Được sử dụng làm nơi chung gian trong các lệnh BLD, BST).

- I: Global Interrupt Enable/Disable Flag (Đây là bit cho phép toàn cục ngắt. Nếu bit này ở trạng thái logic 0 thì không có một ngắt nào được phục vụ).

2.4.2.7. Các thanh ghi chức năng chung.



Hình 2.7. Thanh ghi chức năng chung.

2.4.2.8. Con trỏ ngăn xếp (SP).

Là một thanh ghi 16 bit nhưng cũng có thể được xem như hai thanh ghi chức năng đặc biệt 8 bit. Có địa chỉ trong các thanh ghi chức năng đặc biệt là $3E (Trong bộ nhớ RAM là $5E). Có nhiệm vụ trỏ tới vùng nhớ trong RAM chứa ngăn xếp.

BIT	15	14	13	12	11	10	9	8
S3E(S5E)	-	-	-	-	-	-	-	-
S3D(S5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
	7	6	5	4	3	2	1	0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Intial Value	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0

Hình 2.8. Thanh ghi con chỏ ngăn xếp.

Khi chương trình phục vụ ngắt hoặc chương trình con thì con chỏ PC được lưu vào

ngăn xếp trong khi con trỏ ngăn xếp giảm hai vị trí. Và con trỏ ngăn xếp sẽ giảm 1 khi thực hiện lệnh push. Ngược lại khi thực hiện lệ POP thì con chỏ ngăn xếp sẽ tăng 1 và khi thực hiện lệnh RET hoặc RETI thì con chỏ ngăn xếp sẽ tăng 2. Như vậy con trỏ ngăn xếp cần được chương trình đặt trước giá trị khởi tạo ngăn xếp trước khi một chương trình con được gọi hoặc các ngắt được cho phép phục vụ. Và giá trị ngăn xếp ít nhất cũng phải lớn hơn hoặc bằng 60H (0x60) vì 5FH trỏ lại là các thanh ghi.

2.4.3. Quản lý ngắt.

Ngắt là một cơ chế cho phép thiết bị ngoại vi báo cho CPU biết vế tình trạng sẵn sàng cho đổi dữ liệu của mình. *Ví dụ:* Khi bộ truyền nhận UART nhận được một byte nó sẽ báo cho CPU biết thông của cờ RXC, hoặc khi nó đã truyền được một byte thì cờ TX được thiết lập…

Khi có tín hiệu báo ngắt CPU sẽ tạm dừng công việc đang thực hiện lại và lưu vị trí và thực hiện chương trình (con chỏ PC) vào ngăn xếp sau đó chỏ tới vector phục vụ ngắt và thực hiện chương trình phục vụ ngắt đó cho tới khi gặp lệnh RETI (return from interrup) thì CPU lại lấy PC từ ngăn xếp ra và tiếp tục thực hiện chương trình mà trước khi có ngắt nó đã thực hiện. Trong trường hợp mà có nhiều ngắt yêu cầu cùng một lúc thì CPU sẽ lưu các cờ báo ngắt đó lại và thực hiện lần lượt các ngắt theo bước ưu tiên. Trong khi đang thực hiện ngắt mà xuất hiện ngắt mới thí sẽ xảy ra hai trướng hợp. Trường hợp ngắt này có mức ưu tiên cao hơn thì xẽ được phục vụ. Còn nếu có mức ưu tiên thấp hơn thì sẽ bị bỏ qua.

Bộ nhớ ngăn xếp là vùng bất kỳ. Trong SRAM từ địa chỉ 0x60 trở lên. Để truy nhập vào SRAM thông thường thì dùng con chỏ X, Y, Z và để truy nhập vào SRAM theo kiểu ngăn xếp thì dùng con trỏ SP. Con chỏ này là một thanh ghi 16 bit và được truy nhập như hai thanh ghi 8 bit chung có địa chỉ SPL: 0x3D/0x5D(IO/SRAM) và SPH:0x3E/0x5E.

Khi chương trình phục vụ ngắt hoặc chương trình con thì con trỏ PC được lưu vào ngăn xếp trong khi con trỏ ngăn xếp giảm đi hai vị trí. Và con chỏ ngăn xếp sẽ giảm 1 khi thực hiện lệnh push. Ngược lại khi thực hiện lệnh POP thì con trỏ ngăn xếp sẽ tăng 1 và thực hiện lệnh RET hoặc RETI thì con chỏ ngăn xếp sẽ tăng 2. Như vậy con trỏ ngăn xếp cần được chương trình đặt trước giá trị khởi tạo ngăn xếp trước khi một chương trình con được gọi hoặc các ngắt được cho phép phục vụ. Và giá trị ngăn xếp ít nhất cũng phải lớn hơn hoặc bằng 60H (0x60) vì 5FH trỏ lại là các thanh ghi.

Bảng 2.1. Vector ngắt cho Atmega32

Vector	Program Address	Source	Interrupt Definition
1	$000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	INT2	External Interrupt Request 2
5	$008	TIMER2 COMP	Timer/Counter2 Compare Match
6	$00A	TIMER2 OVF	Timer/Counter2 Overflow
7	$00C	TIMER1 CAPT	Timer/Counter1 Capture Event

8	$00E	TIMER1 CAP	Timer/Counter1 Compare Match A
9	$010	TIMER1 CAPB	Timer/Counter1 Compare Match B
10	$012	TIMER1 OVF	Timer/Counter1 Overflow
11	$014	TIMER0 COMP	Timer/Counter0 Compare Match
12	$016	TIMER0 OVF	Timer/Counter0 Overflow
13	$018	SPI, STC	Serial Transfer Complete
14	$01A	USART, RXC	USART, Rx Complete
15	$01C	USART, UDRE	USART Data Rigister Empty
16	$01E	USART, TCX	USART, Tx Complete
17	$020	ADC	ADC Conversion Complete
18	$022	EE_RDY	EEPROM Ready
19	$024	ANA_COMP	Analog Comparator
20	$026	TWI	Two-wire Serial Interface
21	$028	SPM_RDY	Store Program Memory Ready

2.4.4. Cấu trúc bộ nhớ.

Cũng như mọi vi điều khiển khác AVR có cấu trúc Harvard tức là có bộ nhớ và đường bus riêng cho bộ nhớ chương trình và bộ nhớ dữ liệu.

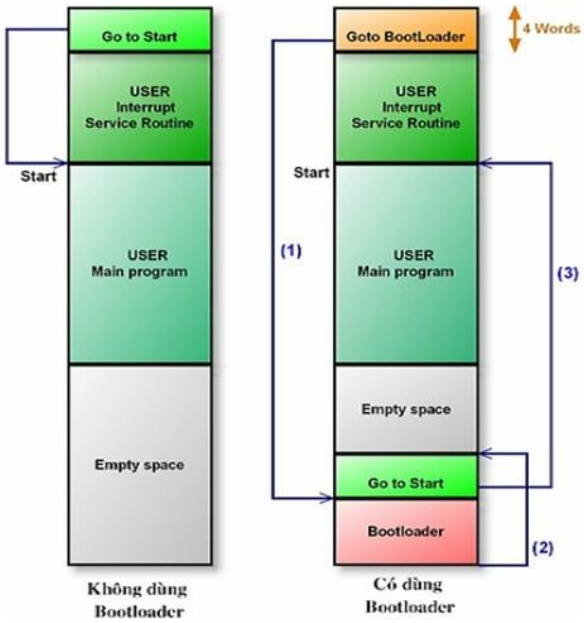
2.4.4.1. Bộ nhớ chương trình.

Bộ nhớ chương trình của AVR là bộ nhớ Flash có dung lượng 32 Kbytes. Bộ nhớ chương trình có độ rộng bus là 16 bit. Ở vi điều khiển ATmega32 bộ nhớ chương trình còn có thể được chia làm 2 phần: Phần boot loader (Boot loader program section) và phần ứng dụng (Applicationprogram section).

- Phần boot loader: Chứa chương trình boot loader.

- Phần ứng dụng (Application program section): Là vùng nhớ chứa chương trình ứng dụng của người dùng. Kích thước của phần boot loader và phần ứng dụng có thể tùy chọn.

Hình 1.9 thể hiện cấu trúc bộ nhớ chương trình có sử dụng và không sử dụng boot loader, khi sử dụng phần boot loader thấy 4 word đầu tiên thay vì chỉ thị cho CPU chuyển tới chương trình ứng dụng của người dùng (là chương trình có nhãn start) thì chỉ thị CPU nhảy tới phần chương trình boot loader để thực hiện trước, rồi mới quay trở lại thực hiện chương trình ứng dụng.



Hình 2.9. Bộ nhớ chương trình có và không có sử dụng boot loader.

2.4.4.2. Bộ nhớ dữ liệu.

Bộ nhớ dữ liệu của AVR chia làm 2 phần chính là bộ nhớ SRAM và bộ nhớ EEPROM. Tuy cùng là bộ nhớ dữ liệu nhưng hai bộ nhớ này lại tách biệt nhau và được đánh địa chỉ riêng.

- Bộ nhớ SRAM: Có dụng lượng 2 Kbytes.

Bảng 2.2. Địa chỉ của tất cả các port.

Tên PORT	Địa chỉ O/I	Địa chỉ SRAM
PORTA	$1B	$3B
DDRA	$1A	$3A
PINA	$19	$39
PORTB	$18	$38
DDRB	$17	$37
PINB	$16	$36
PORTC	$15	$35
DDRC	$14	$34
PINC	$13	$33
PORTD	$12	$32
DDRD	$11	$31
PIND	$10	$30

- Bộ nhớ EEPROM: Bộ nhớ EEPROM có kích thước là 1024 bytes. EEPROM được xem như là một bộ nhớ vào ra được đánh địa chỉ độc lập với SRAM, điều này có nghĩa là cần sử dụng các lệnh in, out … khi muốn truy xuất tới EEPROM.

Để ghi vào EEPROM cần thực hiện các bước sau:

- Chờ cho bit EEWE về 0.
- Cấm tất cả các ngắt.
- Ghi địa chỉ vào thanh ghi EEAR.
- Ghi dữ liệu mà cần ghi vào EEPROM vào thanh ghi EEDR.
- Set bit EEMWE thành 1.
- Set bit EEWE thành 1 .
- Cho phép các ngắt trở lại.

Đọc dữ liệu từ EEPROM: Việc đọc dữ liệu từ EEPROM đơn giản hơn ghi dữ liệu vào EEPROM, để đọc dữ liệu từ EEPROM thực hiện các bước sau:

- Chờ cho bit EEWE về 0.
- Ghi địa chỉ vào thanh ghi EEAR.
- Set bit EERE lên 1.

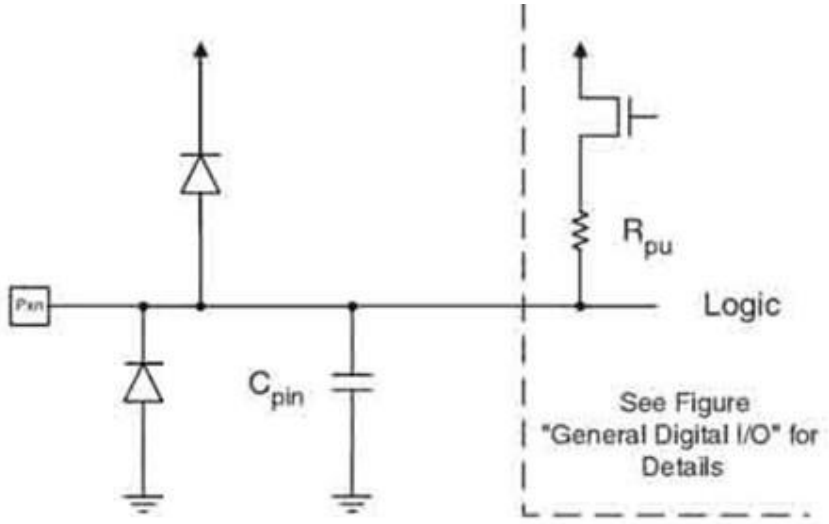
2.4.5. Cổng vào ra.

Cổng vào ra là một trong số các phương tiện để vi điều khiển giao tiếp với các thiết bị ngoại vi. Atmega32 có 4 cổng (port) vào ra 8 bit l: PortA, PortB, PortC, PortD tương ứng với 32 đường vào ra. Các cổng vào ra của AVR là cổng vào ra hai chiều có thể định hướng, tức có thể chọn hướng của cổng là hướng vào (input) hay hướng ra (output). Tất các các

cổng vào ra của AVR điều có tính năng Đọc – Chỉnh sửa Ghi (Read – Modify – write) khi sử dụng chúng như là các cổng vào ra số thông thường. Điều này có nghĩa là khi thay đổi hướng của một chân nào đó thì nó không làm ảnh hưởng tới hướng của các chân khác. Tất cả các chân của các cổng (port) đều có điện trở kéo lên (pull-up) riêng, có thể cho phép hay không cho phép điện trở kéo lên này hoạt động.

- Cách hoạt động.

Khi khảo sát các cổng như là các cổng vào ra số thông thường thì tính chất của các cổng (PortA, PortB, PortC, PortD) là tương tự nhau, nên chỉ cần khảo sát một cổng nào đó trong số 4 cổng của vi điều khiển là đủ. Mỗi một cổng vào ra của vi điều khiển được liên kết với 3 thanh ghi: PORTx, DDRx, PINx. (ở đây x là để thay thế cho A, B, C, D). Ba thanh ghi này sẽ được phối hợp với nhau để điều khiển hoạt động của cổng, chẳng hạn thiết lập cổng thành lối vào có sử dụng điện trở pull-up, ..v.v..



Hình 2.10. Cấu trúc chân của AVR.

Cấu trúc chân của AVR có thể phân biệt rõ chức năng (vào ra) trạng thái (0 1) từ đó có 4 kiểu vào ra cho một chân của AVR. Khác với AT89C51 là chỉ có 2 trạng thái duy nhất (0 1).

Để điều khiển các chân này có 2 thanh ghi.

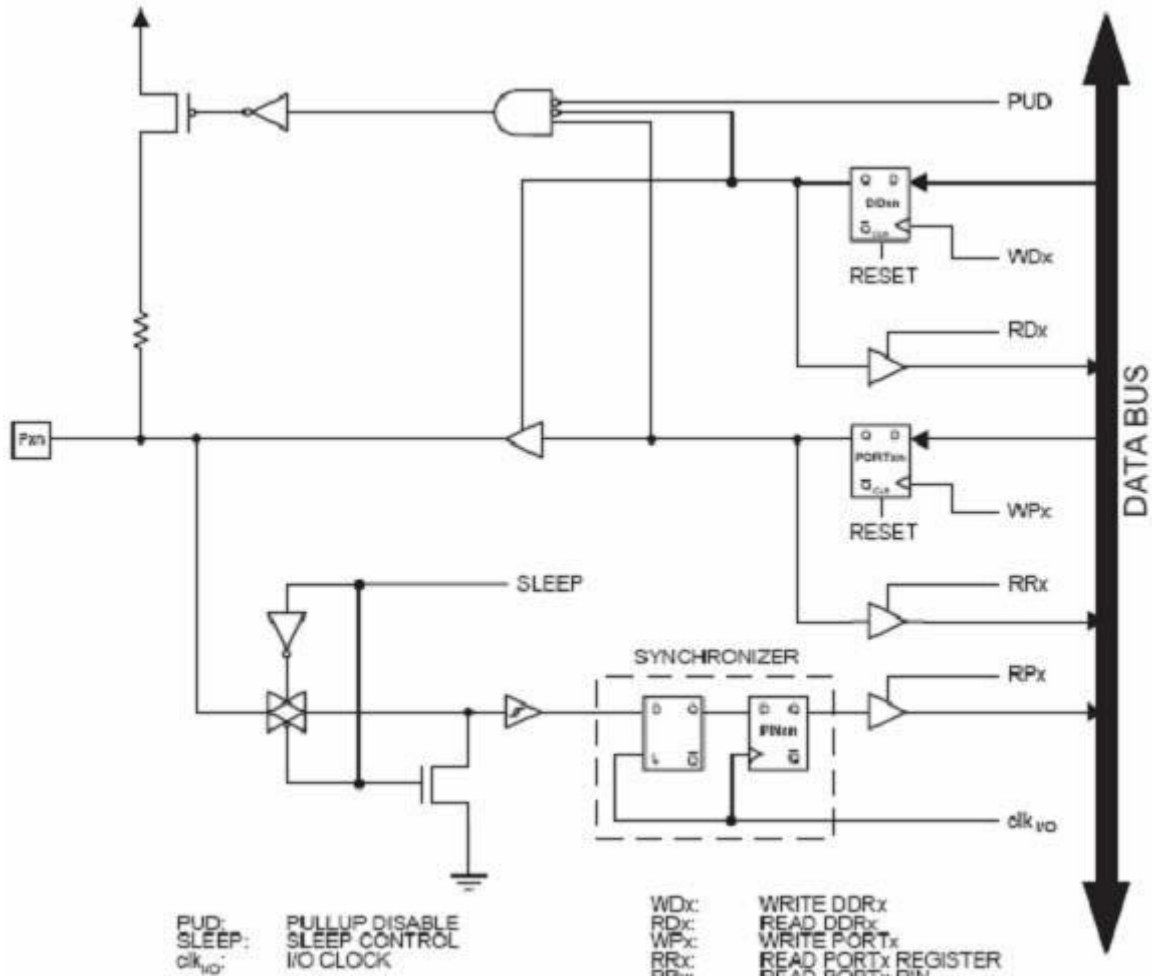
- PORTx: Giá trị tại từng chân (0 – 1) có thể truy cập tới từng bit PORTx.n.

- DDRx: Thanh ghi chỉ trạng thái của từng chân, vào hoặc là ra.

Bảng 2.3. Cấu hình cho các chân cổng.

DDxn	PORTxn	PUD(in SFIOR)	I/O	Pull	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext.Pulled low
0	1	1	Output	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

DDRxn là bit thứ n của thanh ghi DDRx. PORTxn là bit thứ n của thanh ghi PORTx. Dấu "x" ở cột thứ 3 để chỉ giá trị logic là tùy ý.



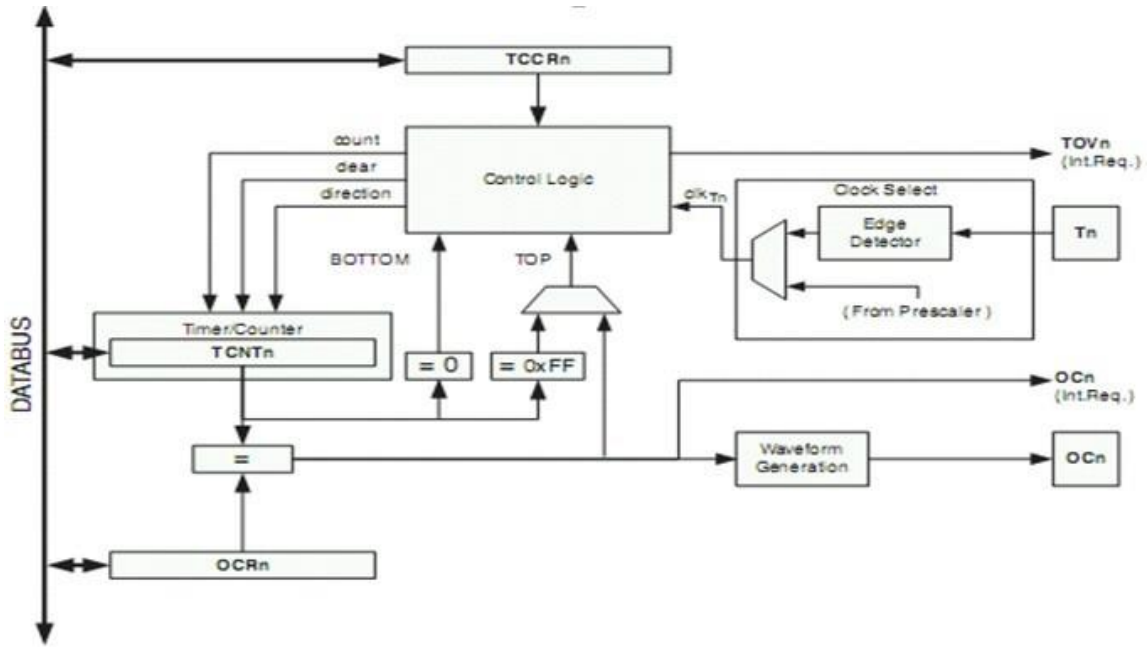
Hình 2.11. Sơ đồ một cổng vào ra.

Ở sơ đồ trên ngoài 2 bit của các thanh ghi DDRx và PORTx tham gia điều khiển điện trở treo (pull-up resistor), còn có một tín hiệu nữa điều khiển điện trở treo, đó là tín hiệu PUD, đây là bit nằm trong thanh ghi SFIOR, khi set bit này thành 1 thì điện trở kéo lên sẽ không được cho phép bất kể các thiết lập của các thanh ghi DDRx và PORTx. Khi bit này là 0 thì điện trở kéo lên được cho phép nếu {DDRxn, PORTxn} = {0, 1}.

2.4.6. Bộ định thời.

Bộ định thời (timer/counter0) là một module định thời/đếm 8 bit, có các đặc điểm sau:

- Bộ đếm một kênh.
- Xóa bộ định thời khi trong mode so sánh (tự động nạp).
- PWM.
- Tạo tần số.
- Bộ đếm sự kiện ngoài.
- Chia tần 10 bit.
- Nguồn ngắt tràn bộ đếm và so sánh. Sơ đồ cấu trúc của bộ định thời:



Hình 2.12. Sơ đồ cấu trúc bộ định thời.

2.4.6.1. Các thanh ghi.

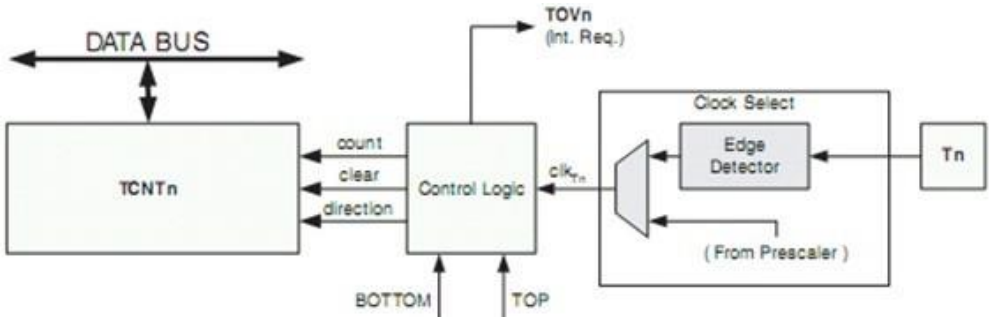
TCNT0 và OCR0 là các thanh ghi 8 bit. Các tín hiệu yêu cầu ngắt đều nằm trong thanh ghi TIFR. Các ngắt có thể được che bởi thanh ghi TIMSK.

Bộ định thời có thể sử dụng xung clock nội thông qua bộ chia hoặc xung clock ngoài trên chân T0. Khối chọn xung clock điều khiển việc bộ định thời/bộ đếm sẽ dùng nguồn xung nào để tăng giá trị của nó. Ngõ ra của khối chọn xung clock được xem là xung clock của bộ định thời (clkT0).

Thanh ghi OCR0 luôn được so sánh với giá trị của bộ định thời/bộ đếm.Kết quả so sánh có thể được sử dụng để tạo ra PWM hoặc biến đổi tần số ngõ ra tại chân OC0.

2.4.6.2. Đơn vị đếm.

Phần chính của bộ định thời 8 bit là một đơn vị đếm song hướng có thể lập trình được. Cấu trúc của nó như hình dưới đây:



Hình 2.13. Đơn vị đếm.

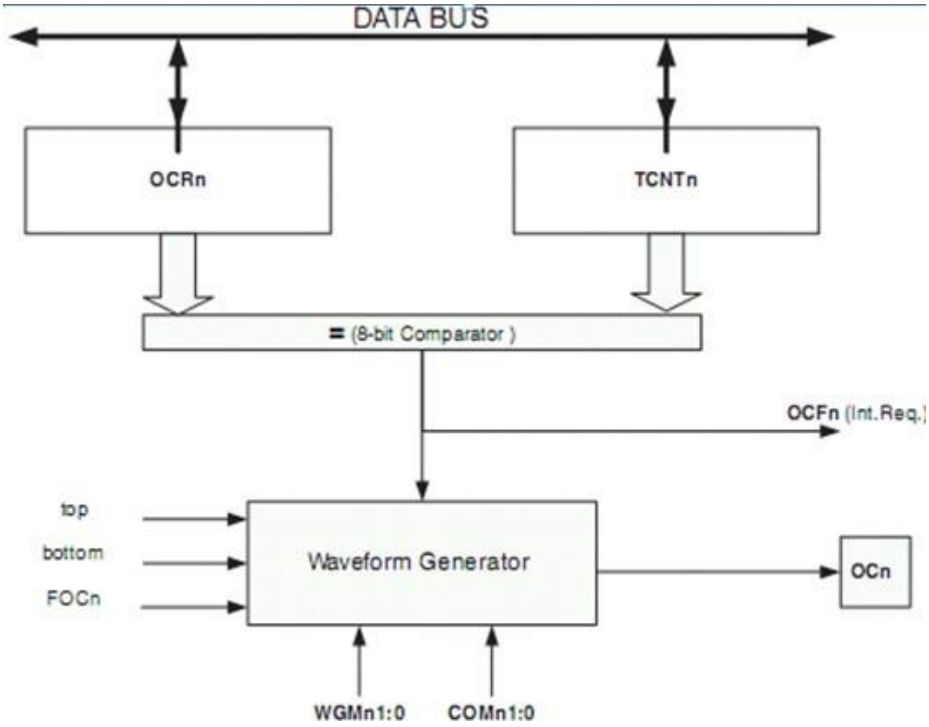
- Count: Tăng hay giảm TCNT0 1.
- Direction: Lựa chọn giữa đếm lên và đếm xuống.
- Clear: Xóa thanh ghi TCNT0.

- ClkT0: Xung clock của bộ định thời.

- TOP: Báo hiệu bộ định thời để tăng đến giá trị lớn nhất.

- BOTTOM: Báo hiệu bộ định thời để giảm đến giá trị nhỏ nhất .

2.4.6.3. Đơn vị so sánh ngõ ra.



Hình 2.14. Đơn vị so sánh ngõ ra.

Bộ so sánh 8 bit liên tục so sánh giá trị TCNT0 với giá trị trong thanh ghi so sánh ngõ ra(OCR0).Khi giá trị TCNT0 bằng với OCR0,bộ so sánh sẽ tạo một báo hiệu.Báo hiệu này sẽ đặt giá trị cờ so sánh ngõ ra(OCF0)lên 1 vào chu kỳ xung lock tiếp theo.Nếu được kích hoạt(OCIE0=1),cờ OCF0 sẽ tạo ra một ngắt so sánh ngõ ra và sẽ tự động được xóa khi ngắt được thực thi.Cờ OCF0 cũng có thể được xóa bằng phần mềm.

2.4.7. Mô tả các thanh ghi.

2.4.7.1. Thanh ghi điều khiển bộ định thời/bộ đếm TCCR0.

Bit	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Intial Value	0	0	0	0	0	0	0	0

Hình 2.15. Thanh ghi điều khiển bộ định thời.

- Bit 7 - FOC0: So sánh ngõ ra bắt buộc: Bit này chỉ tích cực khi bit WGM00 chỉ định chế độ làm việc không có PWM. Khi đặt bit này lên 1, một báo hiệu so sánh bắt buộc xuất hiện tại đơn vị tạo dạng sóng.

- Bit 6, 3 - WGM01:0: Chế độ tạo dạng sóng: Các bit này điều khiển đếm thứ tự của bộ đếm, nguồn cho giá trị lớn nhất của bộ đếm (TOP) và kiểu tạo dạng sóng sẽ được sử

dụng.

- Bit 5:4 - COM01:0: Chế độ báo hiệu so sánh ngõ ra: Các bit này điều khiển hoạt động của chân OC0. Nếu một hoặc cả hai bit COM01:0 được đặt lên 1, ngõ ra OC0 sẽ hoạt động.

- Bit 2:0: CS02:0: Chọn xung đồng hồ: Ba bit này dùng để lựa chọn nguồn xung cho bộ định thời/bộ đếm.

Bảng 2.4. Chọn nguồn xung cho bộ định thời.

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescalin)
0	1	0	$clk_{I/O}$/8(From prescalin)
0	0	1	$clk_{I/O}$/64(From prescalin)
1	1	0	$clk_{I/O}$/256(From prescalin)
1	0	1	$clk_{I/O}$/1024(From prescalin)
1	1	0	External clock source on T0 pin. Clock on falling edge
1	1	1	External clock source on T0 pin. Clock on falling edge

2.4.7.2. Thanh ghi bộ định thời/bộ đếm.

Bit	7	6	5	4	3	2	1	0
	TCNT0(7-0)							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Intial Value	0	0	0	0	0	0	0	0

Hình 1.16. Thanh ghi bộ định thời.

Thanh ghi bộ định thời/bộ đếm cho phép truy cập trực tiếp (cả đọc và ghi) vào bộ đếm 8 bit.

2.4.7.3. Thanh ghi so sánh ngõ ra - OCR0.

Bit	7	6	5	4	3	2	1	0
	OCR0(7-0)							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Intial Value	0	0	0	0	0	0	0	0

Hình 2.17. Thanh ghi so sánh ngõ ra.

Thanh ghi này chứa một giá trị 8 bit và liên tục được so sánh với giá trị của bộ đếm.

2.4.7.4. Thanh ghi mặt nạ ngắt.

Bit	7	6	5	4	3	2	1	0
	OCIF2	TOIE1	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Intial Value	0	0	0	0	0	0	0	0

Hình 2.18. Thanh ghi mặt nạ ngắt TIMSK

- Bit 1-OCIE0: Cho phép ngắt báo hiệu so sánh.
- Bit 0-TOIE0: Cho phép ngắt tràn bộ đếm.

2.4.7.5. Thanh ghi cờ ngắt bộ định thời.

Bit	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Intial Value	0	0	0	0	0	0	0	0

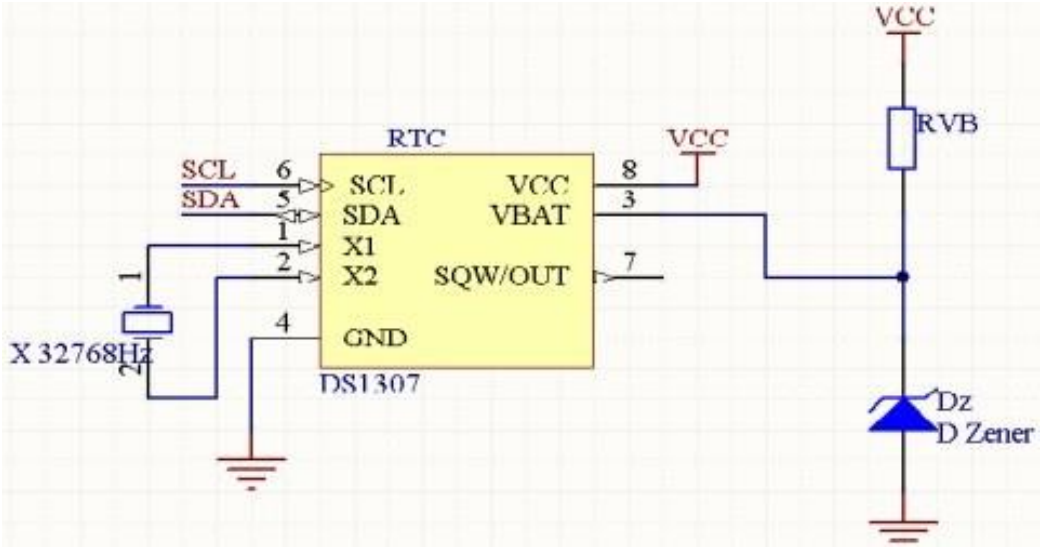
Hình 2.19. Thanh ghi cờ ngắt bộ định thời.

- Bit 1-OCF0: Cờ so sánh ngõ ra 0.
- Bit 0-TOV0: Cờ tràn bộ đếm.

Bit TOV0 được đặt lên 1 khi bộ đếm bị tràn và được xóa bởi phần cứng khi vector ngắt tương ứng được thực hiện. Bit này cũng có thể được xóa bằng phần mềm.

2.4.8. Giao tiếp với I2C

Bus của I2C từ DS1307 và 24Cxx được nối với jumper có thể kết nối với bất kỳ 2 bit của hai cổng bất kỳ cua AVR trên KIT bởi một dây nối.



Hình 2.20. Sơ đồ cấu trúc giao tiếp I2C

2.4.8.1. Thanh ghi:

TWI trên AVR được vận hành bởi 5 thanh ghi bao gồm thanh ghi tốc độ giữ nhịp TWBR, thanh ghi điều khiển TWCR, thanh ghi trạng thái TWSR, thanh ghi địa chỉ TWAR và thanh ghi dữ liệu TWDR.

- TWBR (TWI Bit Rate Register): là 1 thanh ghi 8 bit quy định tốc độ phát xung giữ nhịp trên đường SCL của chip Master.

7	6	5	4	3	2	1	0	
TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Hình 2.21. Thanh ghi quy định tốc độ phát xung.

$$SCL\ frequency = \frac{\text{CPU clock frequency}}{2a16 + 2(TWBR).4^{TWPS}}$$

Trong đó CPU Clock frequency là tần số hoạt động chính của AVR, TWBR là giá trị thanh thi TWBR và TWPS là giá trị của 2 bits TWPS1 và TWPS0 nằm trong thanh thi trạng thái TWSR. Hai bits này được gọi là bit prescaler, thông thường hay set TWPS1: 0 =00 để chọn Prescaler là 1 (40=1). Bảng 1 tóm tắt tốc độ xung giữ nhịp tạo ra trên SCL đối với các giá trị của tham số:

Bảng 2.5. Tốc độ xung giữ nhịp tham khảo.

CPU clock frequency[MHZ}	TWBR	TWPS	SCL frequency[khz]
16	12	0	400
16	72	0	100
14.4	10	0	400
14.4	64	0	100

12	10	0	~333
12	52	0	100
8	10	0	~222
8	32	0	100
4	12	0	100
3.6	10	0	100
2	10	0	~55
1	10	0	~28

- TWCR (TWI Control Register): là thanh ghi 8 bit điều khiển hoạt động của TWI.

7	6	5	4	3	2	1	0	
TWIN7	TWEA	TWSTA	TWSTO	TWWC	TWEB	-	TWIE	TWCR
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Hình 2.22. Thanh ghi điều chỉnh TWI.

Một điều cần chú ý là các bit trong thanh ghi TWCR không cần được set cùng lúc, tùy vào từng giai đoạn trong quá trình giao tiếp TWI các bit có thể được set riêng lẻ.

- TWSR (TWI Status Register): là 1 thanh ghi 8 bit trong đó có 5 bit chứa code trạng thái của TWI và 2 bit chọn prescaler.

7	6	5	4	3	2	1	0	
TWS7	TWS6	TWS5	TWS4	TWS3	TWS2	TWS1	TWS0	TWSR
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Hình 2.23. Thanh ghi trạng thái TWI.

Có rất nhiều bước, nhiều tình huống xảy ra khi giao tiếp bằng TWI cho cả Master và Slave. Ứng với mỗi trường hợp TWI sẽ tạo ra 1 code trong thanh ghi TWSR . Lập trình cho TWI cần xét code trong 5 bit cao của thanh ghi TWSR và đưa ra các ứng xử hợp lý ứng với từng code.

- TWDR (TWI Data Register): là thanh ghi dữ liệu chính của TWI. Trong quá trình nhận, dữ liệu nhận về sẽ được lưu trong TWDR. Trong quá trình gởi, dữ liệu chứa trong TWDR sẽ được chuyển ra đường SDA.

- TWAR (TWI Address Register): là thanh ghi chứa device address của chip Slave. Cấu trúc thanh ghi được trình bày trong hình dưới.

7	6	5	4	3	2	1	0	
TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGECE	TWAR
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Hình 2.24. Thanh ghi chứa device của slave.

Nhớ lại địa chỉ Slave được tạo thành từ 7 bits, trên thanh ghi TWAR 7 bits địa chỉ này nằm ở 7 vị trí cao. Trước khi sử dụng TWI như Slave, phải gán địa chỉ cho chip, việc viết địa chỉ thường được thực hiện bằng lệnh TWAR = (Device_address<<1)+TWGCE. Trong đó TWGCE (TWI General Call Enable) là bit

cho phép cuộc gọi chung. Đề cập bên trên, Slave co quyền cho phép Master thực hiện cuộc gọi chung với nó hay không. Nếu TWGCE=1, Slave sẽ đáp ứng lại cuộc gọi chung nếu có, nếu TWGCE=0 thì Slave sẽ bỏ qua cuộc gọi chung.

Hoạt động của TWI:

TWI trên AVR được gọi là byte-oriented (tạm dịch là hướng byte) và interrupt-based (dựa trên ngắt). Bất kỳ một sự kiện nào trong quá trình truyền/nhận TWI cũng có thể gây ra 1 ngắt TWI. TWI trên AVR vì thế hoạt động tương đối độc lập với chip. Tuy nhiên, cần khai thác ngắt trên AVR một cách hợp lý. Ví dụ, đối với Master, không cần sử dụng ngắt vì chip này hoàn toàn chủ động trong việc truyền và nhận. Riêng với Slave, sử dụng ngắt để tránh bỏ lỡ các cuộc gọi là cần thiết.

Tất cả các AVR trên mạng TWI đều có thể là Master hay Slave, cả Master và Slave đều có thể truyền và nhận dữ liệu. Vì thế, có tất cả 4 mode trong hoạt động của TWI trên AVR. Sẽ lần lượt khảo sát các mode này như sau: Master Transmitter (chip chủ truyền), Master Receiver (Chip chủ nhận), Slave Reicever (chip tớ nhận) và Slave Transmitter (Chip tớ truyền). Trước khi khảo sát các chế độ hoạt động của TWI qui ước một số ký hiệu thường dùng (đây cũng là các ký hiệu dùng trong datasheet của các chip AVR).

S: START condition – điều kiện bắt đầu Rs: REPEAT START – bắt đầu lặp lại

R: READ Bit, bit này bằng 1 được gởi kèm với gói địa chỉ W: WRITE Bit, bit này mang giá trị 0, gởi kèm gói địa chỉ

ACK: Ackowledge, bit xác nhận, chân SDA được kéo xuống 0 ở xung thứ 9

NACK: Not Acknowledge, không xác nhận, SDA ở mức cao ở bit thứ 9 Data: 8 bits dữ liệu

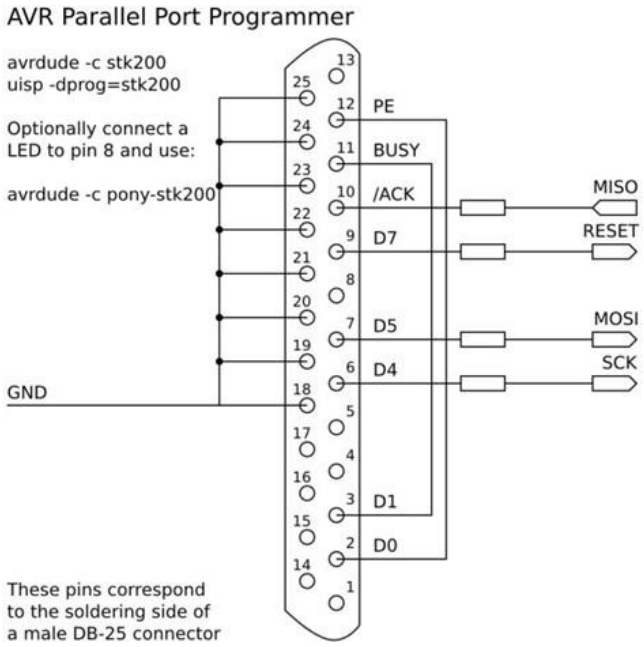
P: STOP condition – điều kiện kết thúc.

SLA: Slave address, địa chỉ của Slave cần giao tiếp.

2.4.9. Mạch nạp cho AVR.

Đây là mạch nạp thông dụng nhất (STK200/300) sử dụng cổng LPT, có thể nạp trực tiếp chương trình CodeVisionAVR. Một trong những ưu điểm lớn nhất của các chip AVR là tính đơn giản khi sử dụng trong đó có việc nạp chương trình cho chíp. AVR hỗ trợ khả năng nạp chương trình ngay trong hệ thống – ISP (In – System Programming), có thể nạp trực tiếp chương trình vào chip mà không cần tháo chip ra khỏi mạch ứng dụng. Mạch nạp cho AVR rất phong phú nhưng hầu hết đều rất đơn

giản. Dưới đây là loại mạch nạp STK200, cực kỳ đơn giản chỉ với một cổng DB25 male, và 4 điện trở vài trăm ohm là được.



Hình 2.25. Mạch nạp STK200 sử dụng DB25 và 4 điện trở.

Hiện nay trên thị trường cũng có bán rất nhiều các bộ nạp cho AVR giao tiếp qua cổng USB rất gọn và dễ dàng sử dụng. Tuy nhiên giá thành vẫn có thể đắt hơn các bộ nạp cũ. Các bộ nạp cũ vẫn có thể sử dụng qua các cáp chuyển đổi Com sang USB.

CHƯƠNG 3 THUẬT TOÁN ĐIỀU KHIỂN PID BẰNG CÁCH ĐIỀU CHẾ ĐỘ RỘNG PWM

3.3. Nguyên lý điều xung PWM.

PWM (pulse width modulation) là biến điệu độ rộng xung, là phương pháp điều chỉnh điện áp ra tải, hay nói cách khác là phương pháp điều chế dựa trên sự thay đổi độ rộng của chuỗi xung vuông dẫn đến sự thay đổi điện áp ra. Trong các thiết bị cơ điện tử, thường dùng PWM để điều tốc, mô-men của động cơ DC rất có hiệu quả.

PWM (Pulse Width Modulation) đang đóng vai trò gần như tuyệt đối trong các hệ công suất chuyển mạch SMPS (Switch Mode Power Supply) còn gọi là nguồn xung. Nói một cách khác, nhắc tới nguồn xung thì người ta nghĩ ngay đến PWM.

PWM tạo dựng trên nguyên tắc chuyển lải năng lượng từ A đến B dưới dạng các xung vuông toàn áp (biên độ xung gần với điện áp nguồn cung cấp) liên tiếp. Trong đó mức năng lượng tỷ lệ thuận với thời gian mở xung (độ rộng xung tính trên đơn vị thới gian.

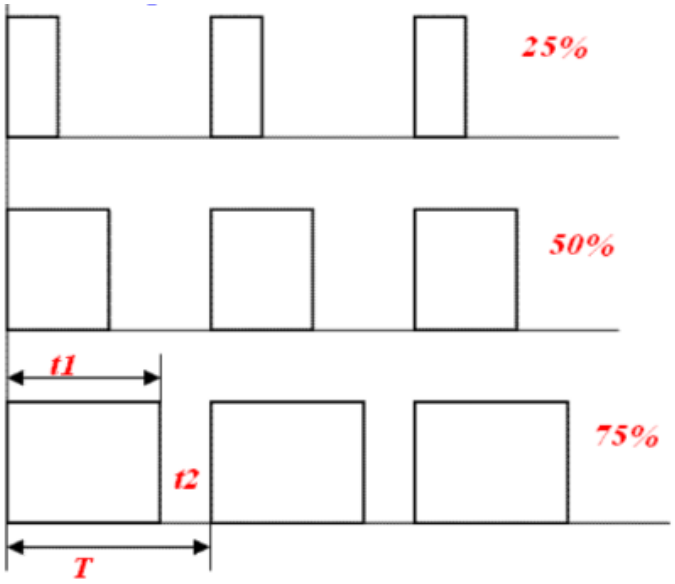
Tần số xung trong PWM có thể cố định hay biến đổi (thường là cố định tần số xung chuyển mạch). Với tần số cố định, chu kỳ t bằng tổng thời gian mở xung t(on) với thời gian tắt xung t(off).

$$t = t(on) + t(off) = t1 + t2$$

Tỷ lệ của thời gian mở trên chu kỳ xung chính là độ sâu điều biến độ rộng xung, đặc trưng thành thuật ngữ "% duty".

Ví dụ, tần số xung 1 KHz --> t = 1 ms.

Với t(on) = 0,5 ms --> ta có độ rộng xung T=(T1/T2).100% = 50%

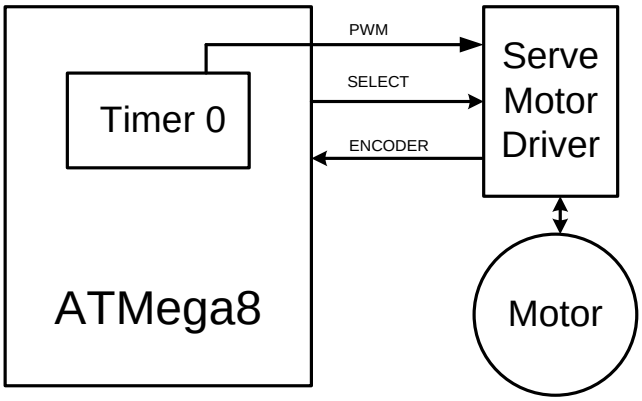


Hình 3.1. Tỷ lệ thời gian mở trên chu kỳ xung.

Như vậy

- Đối với PWM = 25% thì Ut = Umax.25%(V)
- Đối với PWM = 50% thì Ut = Umax.50% (V)
- Đối với PWM = 75% thì Ut = Umax.75% (V)

Trong vi điều khiển hiện đại như PIC hay AVR thường có sẵn module PWM.



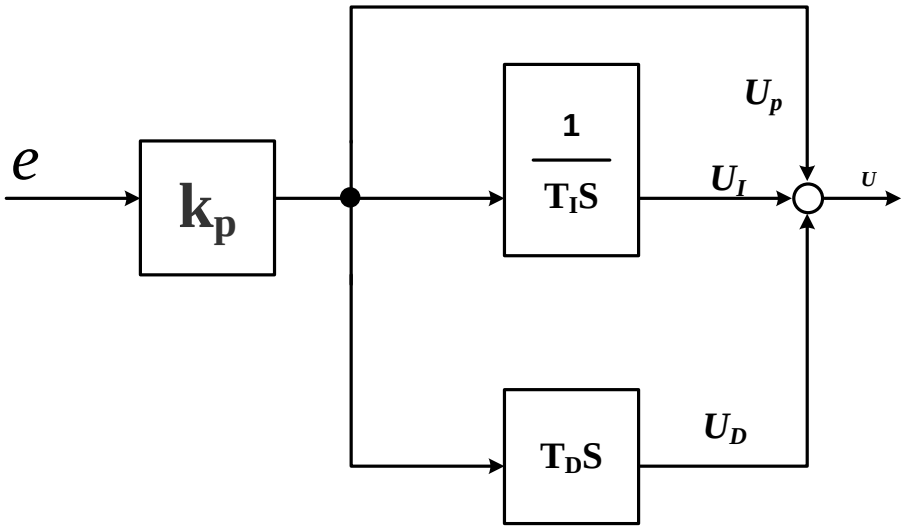
Hình 3.2. Module điều khiển motor bằng AVR.

Sử dụng các kênh PWM này, ta có thể điều khiển tốc độ động cơ theo ý muốn. Kênh PWM1 được dùng cho chiều thuận của động cơ, PWM2 được dùng cho chiều còn lại của động cơ.

3.4. Thuật toán PID

3.4.1. Khái quát về bộ điều khiển PID

Cấu trúc của bộ điều khiển PID (hình2.3) gồm 3 thành phần là khâu khuếch đại (P), khâu tích phân (I) và khâu vi phân (D). Khi sử dụng thuật toán PID nhấ thiết phải lựa chọn chế độ làm việc P, I hay D và sau đó đặt tham số cho các chế độ đã chọn. Một cách tổng quát, có ba thuật toán cơ bản được sử dụng là P, PI và PID.



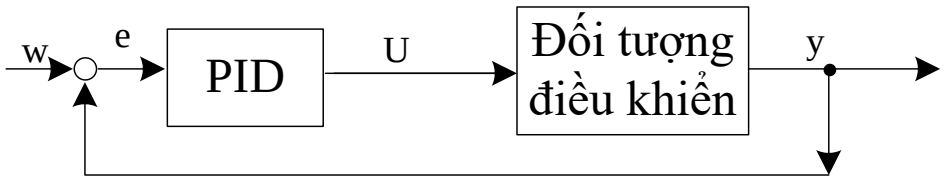
Hình 3.3. Cấu trúc bộ điều khiển PID. U_p U_I U_D

Bộ điều khiển PID có cấu trúc đơn giản, dễ sử dụng nên được sử dụng rộng rãi trong điều khiển các đối tượng SISO theo nguyên lý hồi tiếp (hình 4.4) bộ PID có nhiệm vụ đưa sai lệch e(t) của hệ thống về 0 sao cho quá trình quá độ thỏa mãn các yêu cầu cơ bản về chất lượng:

- Nếu sai lệch tĩnh e(t) càng lớn thì thông qua thành phần $u_P(t)$, tín hiệu điều chỉnh u(t) càng lớn.

- Nếu sai lệch e(t) chưa bằng 0 thì thông qua thành phần $u_I(t)$, PID vẫn còn tạo tín hiệu điều chỉnh.

- Nếu sự thay đổi của sai lệch e(t) càng lớn thì thông qua thành phần $u_D(t)$, phản ứng thích hợp của u(t) sẽ càng nhanh.



Hình 3.4 .Điều khiển hồi tiếp với bộ điều khiển PID

Bộ điều khiển PID được mô tả bằng mô hình vào-ra:

$$u(t) = k_p\left[e(t) + \frac{1}{T}\int_0^t e(\tau)d\tau + \mathrm{T_D}\frac{de(t)}{dt}\right]$$

Trong đó :

e(t) – tín hiệu đầu vào.

u(t) – tín hiệu đầu ra.

k_p – tín hiệu khuếc đại

T_I – hằng số tích phân.

T_D – hằng số vi phân.

Từ mô hình vào – ra trên, ta có được hàm truyền đạt của bộ điều khiển PID

$$R(s) = k_p\left(1 + \frac{1}{T_I s} + T_D s\right)$$

Có nhiều phương pháp xác định tham số của bộ điều khiển PID:

- **Phương pháp Ziegler-Nichols.**
- Phương pháp Chien-Hrones-Reswick.
- Phương pháp tổng T của kuhn.
- Phương pháp tối ưu modul và phương pháp tối ưu đối xứng.
- Phương pháp tối ưu theo sai lệch bám

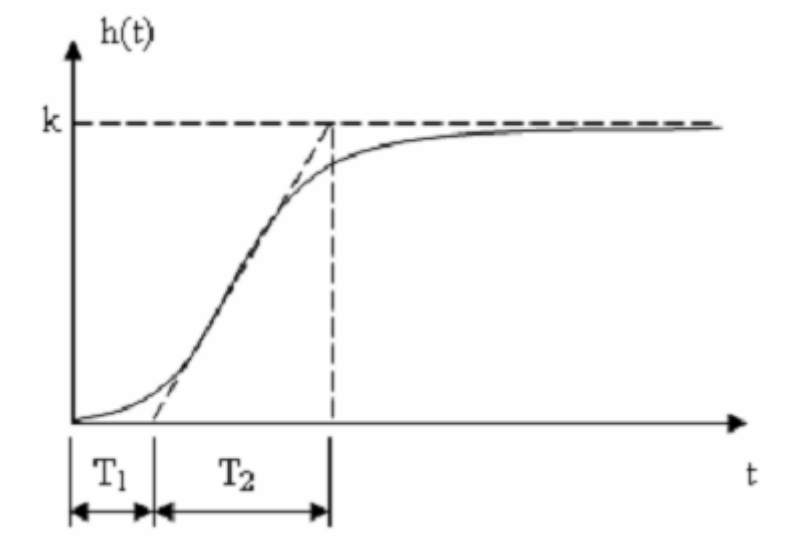
Đề tài sử dụng phương pháp Ziegler-Nichols nên tôi chỉ đi sâu vào phương pháp này, các bạn có thể tìm hiểu thêm các nguyên tắc khác như đã nêu ở trên.

3.4.2. Phương pháp Ziegler-Nichols.

Phương pháp Ziegler-Nichols là phương pháp thực nghiệm để xác định tham số bộ điều khiển P, PI hoặc PID bằng cách dựa vào đáp ứng quá độ của đối tượng điều khiển. Tùy theo đặc điểm của từng đối tượng, Ziegler và Nichols đưa ra hai phương pháp lựa chọn tham số của bộ điều khiển

a) Phương pháp Ziegler-Nichols thứ nhất:

Phương pháp này áp dụng cho các đối tượng có đáp ứng đối với tín hiệu vào là hàm nấc có dạng chữ S (hình 4.5) như nhiệt độ lò nhiệt, tốc độ động cơ…



Hình 3.5. Đáp ứng nấc của hệ hở có dạng chữ S.

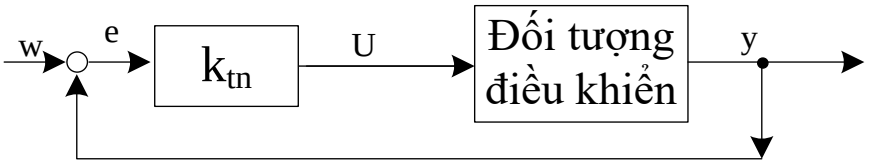
Thông số của bộ điều khiển được chọn theo bảng sau:

Bảng 3.1. Các tham số PID theo phương pháp Ziegler-Nichols thứ nhất

Thông số / BĐK	k_p	T_I	T_D
P	$T_2/(k.T_1)$	-	-
I	$0,9T_2/(k.T_1)$	$T_1/0,3$	-
D	1,2T2/(k.T1)	2T1	0,5T1

b) Phương pháp Ziegler-Nichols thứ hai.

Phương pháp này áp dụng cho đối tượng có khâu tích phân lý tưởng như mực chất lỏng trong bồn chứa, hệ truyền động dùng động cơ… đáp ứng quá độ của hệ hở của đối tượng tăng đến vô cùng. Phương pháp này được thực hiện nhưsau:

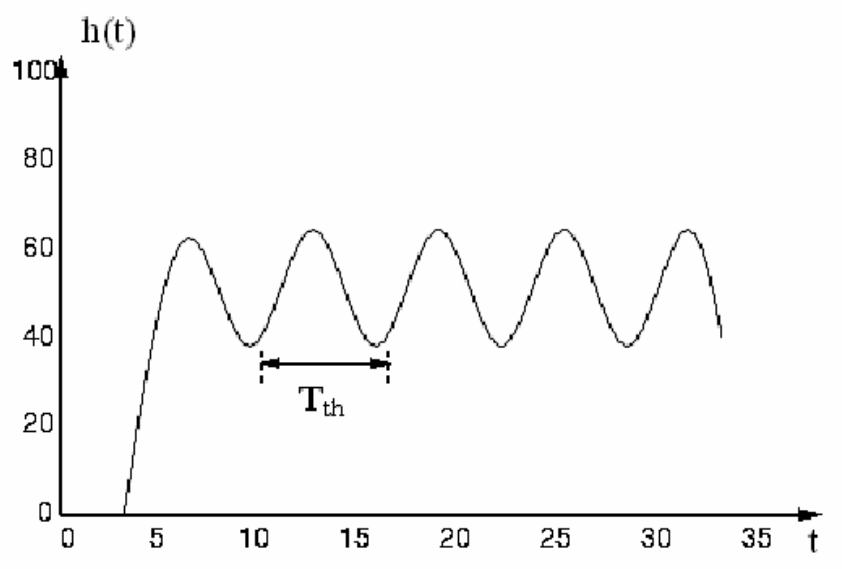


Hình 3.6. Xác định hằng số khuếch đại tới hạn

- Thay bộ điều khiển PID trong hệ kín bằng bộ khuếc đại (hình 4.6)

- Tăng hệ số khuếc đại tới giá trị tới hạn k_{tn} để hệ kín ở chế độ biên giới ổnđịnh, tức là h(t) có dạng dao động điều hòa.

- Xác định chu kỳ T_{th} của dao động.



Hình 3.7: Đáp ứng nấc của hệ kín khi k= k_{tn}

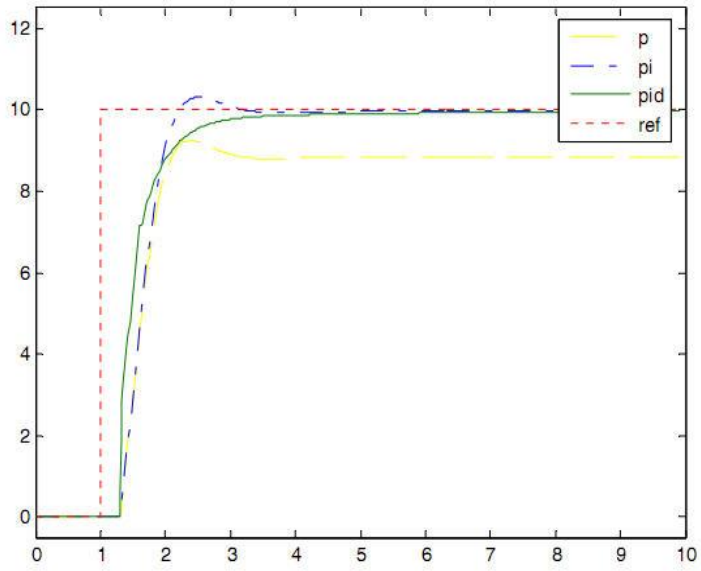
Thông số của các bộ điều khiển được chọn theo bảng sau:

Bảng 3.2: Các tham số PID theo phương pháp Ziegler-Nichols thứ hai.

Thông số / BĐK	k_p	T_I	T_D
P	$0,5k_{tn}$	-	-
I	$0,45\ k_{tn}$	$0,85\ T_{th}$	-
D	$0,6\ k_{tn}$	$0,5\ T_{th}$	$0,125\ T_{th}$

CHƯƠNG 4. THIẾT KẾ BỘ ĐIỀU KHIỂN PI CHO ĐỘNG CƠ DC SERVO

Hệ thống trên chỉ thực sự đạt hiệu quả khi ta thiết kế cho nó 1 bộ điều khiển phù hợp.Với kết cấu phần cứng và đặc tính điều khiển là điều khiển tốc độ thì bộ điều khiển PI số là thích hợp hơn cả, ta chỉ dùng thêm khâu vi phân nếu liên quan tới điều khiển vị trí.Do ứng dụng vi điều khiển ngày càng rộng rãi, chính vì vậy việc số hóa bộ điều khiển PI là rất quan trọng.Đi cùng với nó là các phương pháp cho ta số hóa một cách tương đối chính xác như Ziegler-Nichols…….v v.



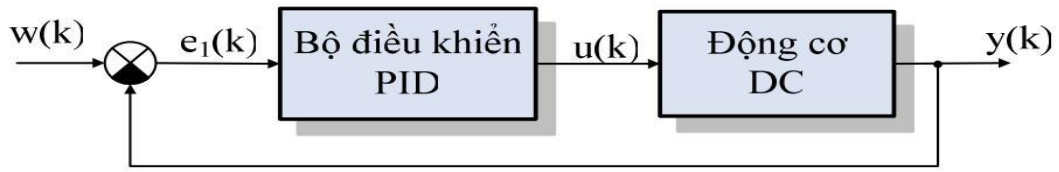
Hình 4.1. So sánh đặc tính P,PI,PD,PID

4.5. Thuật toán điều khiển.

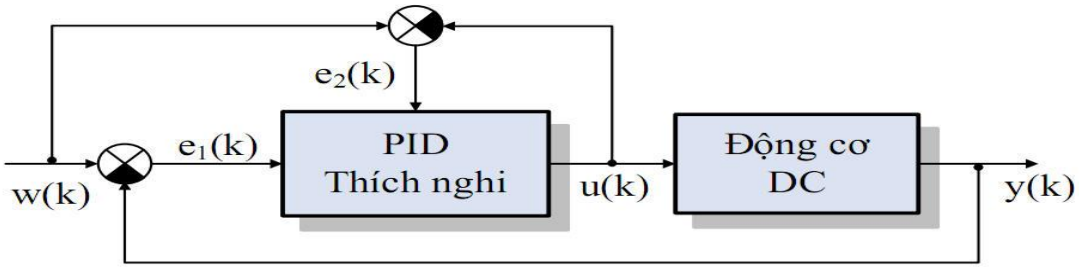
Về nguyên tắc khi xây dựng bộ điều khiển cho một đối tượng nào đó thì ta cần biết được cấu trúc, đặc tính, hàm truyền…của đối tượng.

Với động cơ 1 chiều bất kì thì ta không phải lúc nào cũng mổ xẻ động cơ ra rồi đo đạc các thông số của nó, chính vì vậy bộ điều khiển phải có nhiệm vụ tạo ra tín hiệu điều khiển phù hợp nhất, có khả năng điều khiển linh hoạt với nhiều động cơ khác nhau.

Với điều khiển PID thì có rất nhiều cách thức thực hiện, riêng bộ PID thì có PID thường và PID thích nghi



Hình 4.2. PID thường

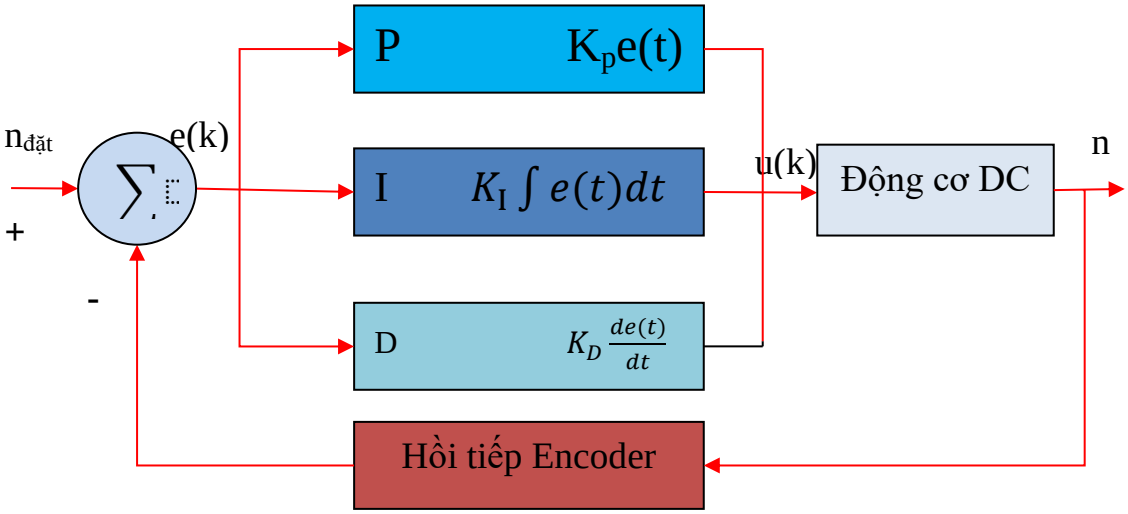


Hình 4.3. PID thích nghi

Với PID thường thì đầu ra chỉ phụ thuộc vào tín hiệu $e_1(k)$, còn với PID thích nghi thì còn phụ thuộc thêm vào $e_2(k)$.

Thực tế cho thấy PID thích nghi hoạt động tốt và ổn định hơn PID thường.

4.6. Phương trình toán học bộ PID.

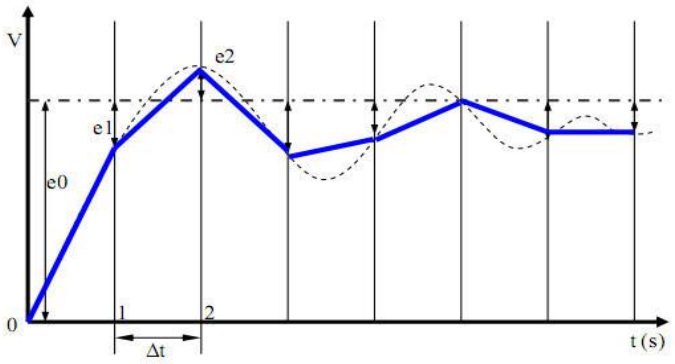


Hình 4.4. Sơ đồ khối bộ PID

$$U(t) = K_P * e(t) + K_I \int e(t) * dt + K_D \frac{de(t)}{dt}$$

Nếu bỏ thành phần vi phân đi ta chỉ còn PI:

$$U(t) = K_P * e(t) + K_I \int e(t) * dt$$



Chú thích: - đường chấm gạch biểu diễn vận tốc cần thiết v_set.
- đường gạch gạch biểu diễn vận tốc thực tế của động cơ.

Hình 4.5. Trích mẫu lấy tín hiệu

Δt là thời gian lấy mẫu (Samplingtime) là rất nhỏ.

Ta tính thành phần tích phân $\int_0^t e(t) * dt = \lim(\sum e(t) * \Delta t)_{\Delta t \to 0}$ Lấy gần đúng ta có

$\int_0^t e(t) * dt = \lim(\sum e(t) * \Delta t)_{\Delta t \to 0} = \sum e(i) * \Delta t$ trong đó i=0,1,2,…

Tóm lại ta có $U(t) = K_P * e + K_I * \sum e(i) \Delta t$

Đặt $e\_sum(i+1) = \sum e(i) = e\_sum(i) + e(i+1)$ Công thức trên được viết lại như sau:

$U(t) = K_P * e + K_I e\_sum$

Áp dụng vào phần cứng điều khiển ta có ngõ ra bộ PI là %duty (PWM).

Ngõ vào là $e\_sum = v\_cur - v\_set$

v_cur là vận tốc hiện tại

v_set là vận tốc đặt

Sự phụ thuộc của vận tốc vào %duty là gần như tuyến tính cho nên để đơn gian ta coi nó là tuyến tính.Vì vậy ta có thể điều khiển vận tốc thông qua %duty.

- **Giải thuật lập trình để tính PWM như sau**

  Kp,K_I là các hệ số xác định nhờ phương pháp Ziegler-Nichols.

  e_2 =(v_set-v_cur) là sai lệch đang xét

  e_1 là sai lệch trước đó

  e_sum=e_2+e_1 là tổng các sai lệch tới thời điểm đang xét

  e_sub=(e_2 - e_1) là độ biến thiên sai lệch

  v_cur là vận tốc hiện tại

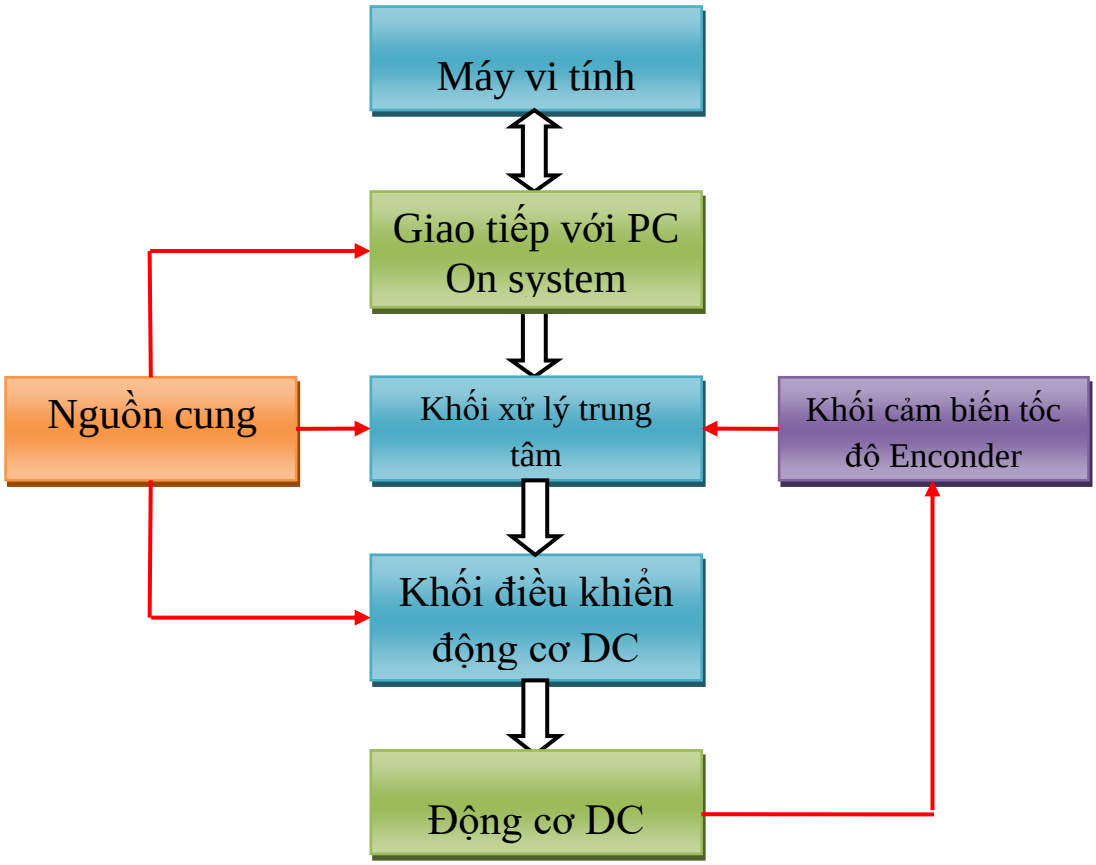
  v_set là vận tốc đặt

  Với encoder 100xung/vòng, ta trích mẫu 50ms, như vậy vận tốc động cơ tại thời điểm đang xét là v_cur = 12*INT0 (INT0 là số xung đếm được trong 50ms)

  Ta đi đến hệ thức tính PWM cho PI cuối cùng là:

$$\mathbf{PWM=PWM + Kp*e\_2 + K_I*e\_sum}$$

4.7. Sơ đồ khối mạch điều khiển.



Hình 4.6. Liên kết các khối trong hệ thống

Atmega8 có khả năng On system tức là nạp chương trình cho chip ngay cả khi chip đang hoạt động, chỉ cần kết nối các PIN MISO, MOSI, SCK, RESET vào mạch nạp hợp lý là nạp được.

4.8. Tính chọn hệ số Kp, K_I

Có nhiều phương pháp để chọn hệ số cho bộ điều khiển, nhưng trong đề tài tôi xét thấy phương pháp Ziegler-Nichols thứ hai là hợp lý hơn cả vì nó đáp ứng được yêu cầu của và mục tiêu của đề tài.

4.8.1. Tính chọn.

Ta dựa vào phương pháp thứ 2 của Ziegler-Nichols.Nội dung phương pháp như sau:

Tham số cho bộ điều khiển PID chọn theo bảng trên ta được:

Lấy Kgh=3; Tgh=0.2(s)

→ Kp=0.45*3=1.35=27/20

Ti=KI=0.85*0.2=0.17=17/100

Chú ý: giá trị chọn chỉ là giá trị ban đầu trong quá trình chạy thử động cơ đã có sự thay đổi để đạt được hiệu quả tốt hơn.

4.4.3. Code cho vi điều khiển.

- Nguyên tắc hoạt động của trương chình như sau:

Khi hận được chuỗi dữ liệu 24 bít từ PC chuyền đến

Tín hiệu có dạng:

“s” “ chr(n)” “@”(mỗi giá trị được mã hóa 8bit) động co sẽ quay theo chiều thuận.

Nhận giá trị và xác định chiều qua của dộng cơ và giá trị tốc độ quay chính là ev_1 lúc này ev_2 bằng 0

Khi đố vi điều khiển sẽ đặt giá trị đó vào công thức tính PWM và chuyền sung PWM ra chân 39 của VDK kết hợp với tín hiệu đảo chiều khi đó động cơ sẽ bắt đầu hoạt động.

Sau khi động cơ bắt đâu hoạt động cứ 20ms vi điều khiển lại đọc giá trị phản hồi từ encoder của động cơ để xác định tốc độ chính là ev_2 gửi về máy tính đồng thời cũng xác chuyền vào công thức PWM để can thiệp điều khiển sao cho tốc độ đạt giá trị người dùng đặt.

Khi người sử dụng tick vào nút đảo chiều trên giao diện người sử dụng PC sẽ gửi đến VDK một đoạn dữ liệu như sau: “ d” “i” “@” ngay lập tức VDK sẽ xuất ra PWM bằng 0 và lệnh dừng động cơ. Khi động cơ đã dừng VDK sẽ tiếp tục xuất sung PWM với giá trị giống khi quay chiều thuận.

Khi người dùng click chuột vào ô stop trên giao diện người sử dụng PC sẽ gửi đến VDK một đoạn mã như sau: “t” “t” “@” ngay lập tức VDK sẽ xuất PWM = 0 động cơ dừng hoạt động.

Mã code C cho AVR

```
/*****************************************************
Trinh Dinh Hao – Lop DCL2401 - HPU
Project :
Version :
Date    : 15/07/2022
Author  :
Company :
Comments:
Chip type           : ATmega8
Program type        : Application
Clock frequency     : 8.000000 MHz
Memory model        : Small
External RAM size   : 0
Data Stack size     : 256
*****************************************************/
#include <mega8.h>
#include <delay.h>
#define PWM OCR1AL
#define dc PORTB.0
unsigned char t=0,i=0,value_clock=0,ev_2=0,fb=0;
unsigned char buffer_vb[3];// Mang nhan 2 phan tu
bit enable_send_pc=0,enable_encoder=0;
int ev_1=0,clock_INT0=0,run=0;
// External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{
// Place your code here
if(enable_encoder==1)//Neu dc phep dem xung
{
   clock_INT0++;//Dem xung tu encoder
}
}
#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
```

```
#define UDRE 5
#define RXC 7
#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)
// USART Receiver buffer
#define RX_BUFFER_SIZE 8
char rx_buffer[RX_BUFFER_SIZE];
#if RX_BUFFER_SIZE<256
unsigned char rx_wr_index,rx_rd_index,rx_counter;
#else
unsigned int rx_wr_index,rx_rd_index,rx_counter;
#endif
// This flag is set on USART Receiver buffer overflow
bit rx_buffer_overflow;
// USART Receiver interrupt service routine
interrupt [USART_RXC] void usart_rx_isr(void)
{
char status,data;
status=UCSRA;
data=UDR;
if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
   {
   rx_buffer[rx_wr_index]=data;
   if (++rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;
   if (++rx_counter == RX_BUFFER_SIZE)
      {
      rx_counter=0;
      rx_buffer_overflow=1;
      };
   };
}
#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the USART Receiver buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
```

```
{
char data;
while (rx_counter==0);
data=rx_buffer[rx_rd_index];
if (++rx_rd_index == RX_BUFFER_SIZE) rx_rd_index=0;
#asm("cli")
--rx_counter;
#asm("sei")
return data;
}
#pragma used-
#endif
// USART Transmitter buffer
#define TX_BUFFER_SIZE 8
char tx_buffer[TX_BUFFER_SIZE];
#if TX_BUFFER_SIZE<256
unsigned char tx_wr_index,tx_rd_index,tx_counter;
#else
unsigned int tx_wr_index,tx_rd_index,tx_counter;
#endif

// USART Transmitter interrupt service routine
interrupt [USART_TXC] void usart_tx_isr(void)
{
if (tx_counter)
   {
   --tx_counter;
   UDR=tx_buffer[tx_rd_index];
   if (++tx_rd_index == TX_BUFFER_SIZE) tx_rd_index=0;
   };
}
#ifndef _DEBUG_TERMINAL_IO_
// Write a character to the USART Transmitter buffer
#define _ALTERNATE_PUTCHAR_
#pragma used+
void putchar(char c)
{
while (tx_counter == TX_BUFFER_SIZE);
#asm("cli")
```

```
if (tx_counter || ((UCSRA & DATA_REGISTER_EMPTY)==0))
   {
   tx_buffer[tx_wr_index]=c;
   if (++tx_wr_index == TX_BUFFER_SIZE) tx_wr_index=0;
   ++tx_counter;
}
else
   UDR=c;
#asm("sei")
}
#pragma used-
#endif
// Standard Input/Output functions
#include <stdio.h>
// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Reinitialize Timer 0 value
TCNT0=0x83;// Ngat 1ms
// Place your code here
t++;
if(t==20 && run==1)//Neu dat 20ms va dc nap toc do
{
   t=0;
   enable_encoder=0;// Chot giu gia tri xung
   fb=clock_INT0;// Copy xung encoder
   clock_INT0=0;// Xoa de dem lai tu dau
   enable_send_pc=1;//Cho phep gui len PC
   enable_encoder=1;//Cho phep dem xung encoder
   ev_1=value_clock-fb;// Sai lech hien tai
   PWM=PWM+0.1*ev_1+0.001*(ev_1+ev_2);// Tinh toan PWM
   ev_2=ev_1;// Sai lech 2
}
else if(t==20 && run==0)
{
   t=0;
   clock_INT0=0;// Xoa de dem lai tu dau
}
}
```

```
// Declare your global variables here

void main(void)
{
// Declare your local variables here
// Input/Output Ports initialization
// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=Out Func1=Out Func0=Out
// State7=T State6=T State5=T State4=T State3=T State2=0 State1=0 State0=0
PORTB=0x00;
DDRB=0x07;
// Port C initialization
// Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;
// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=P State1=T State0=T
PORTD=0x04;
DDRD=0x00;
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 125.000 kHz
TCCR0=0x03;
TCNT0=0x83;
// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 125.000 kHz
// Mode: Ph. correct PWM top=0x00FF
// OC1A output: Non-Inv.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0x81;
```

```
TCCR1B=0x03;
TCNT1H=0x00;
TCNT1L=0x00;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;
// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer2 Stopped
// Mode: Normal top=0xFF
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;
// External Interrupt(s) initialization
// INT0: On
// INT0 Mode: Rising Edge
// INT1: Off
GICR|=0x40;
MCUCR=0x03;
GIFR=0x40;
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x01;
// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud Rate: 9600
UCSRA=0x00;
UCSRB=0xD8;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x33;
// Analog Comparator initialization
```

```
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;
// ADC initialization
// ADC disabled
ADCSRA=0x00;
// SPI initialization
// SPI disabled
SPCR=0x00;
// TWI initialization
// TWI disabled
TWCR=0x00;
// Global enable interrupts
#asm("sei")
// chương trình chính
dc=1;
PWM=0;
while (1)
      {
      // Place your code here
      if(rx_counter>0) //  nếu có dữ liệu từ PC
      {
        buffer_vb[i]=getchar(); // nhận dữ liệu
        i++;
      }
      else if(rx_counter==0 && i==3)
      {
        //---------------Kiem tra du lieu nhan duoc-----------------
        if(buffer_vb[0]=='s' && buffer_vb[2]=='@' && buffer_vb[1]!='t' &&
buffer_vb[1]>0)//Neu yeu cai khoi dong dong co
        {
           value_clock=buffer_vb[1];// Copy gia tri toc do yeu cau tu PC
           run=1;
           for(i=0;i<3;i++)
           {
              buffer_vb[i]=0;//Xoa bo dem
           }
        i=0;
```

```
      }
      else if(buffer_vb[0]=='d' && buffer_vb[2]=='@'&& buffer_vb[1]=='i')
//Neu yeu cai dao chieu dong co
      {
         PWM=0;// Tat PWM
         run=0;// Dung dong co
         delay_ms(1000);// Tre
         dc=~dc;
         run=1;
         for(i=0;i<3;i++)
         {
            buffer_vb[i]=0;//Xoa bo dem
         }
      i=0;
      }
      elseif(buffer_vb[0]=='t'&&buffer_vb[2]=='@'&&buffer_vb[1]=='t')
//Neu yeu cai stop dong co
      {
      run=0;
      PWM=0;
      value_clock=0;// Xoa
      clock_INT0=0;// Xoa
      enable_encoder=0;// Chot giu gia tri xung
      for(i=0;i<20;i++)
         {
             putchar(PWM);// Gui toc do 0 len may tinh
         }
      i=0;
      }
      for(i=0;i<3;i++)
         {
         buffer_vb[i]=0;//Xoa bo dem
         }
      i=0;
     }
     if(enable_send_pc==1 && PWM>0 && run==1)// Neu yeu cau gui len Pc
     {
      putchar(fb);// Gui thong tin toc do len may tinh
      enable_send_pc=0;// Chi gui sau 20ms
```

```
    }
    };
}
```

Kết luận

Trong thời gian làm đồ án em đã tìm hiểu được những nội dung chính sau:

- Cấu tạo và nguyên lý hoạt động của động cơ DC Servo;
- Cấu trúc và tài nguyên của dòng vi điều khiển AVR;
- Thuật toán điều khiển PID và các phương pháp tính các hệ số P,I,D
- Thiết kế xây dựng mô hình điều khiển động cơ DC Servo theo thuật toán PID.

Em xin chân thành cảm ơn sự hướng dẫn tận tình của thầy Đỗ Anh Dũng và thầy Đoàn Hữu Chức để em hoàn thành đồ án tốt nghiệp với đề tài: "Nghiên cứu thiết kế bộ điều khiển tốc độ động cơ DC Servo" đúng thời gian quy định. Đồ án còn những thiếu sót kính mong các thầy và các bạn góp ý để đồ án được hoàn thiện hơn.

Em xin chân thành cảm ơn các thầy !

Hải phòng , ngày tháng năm 2022

Sinh viên thực hiện

Tài liệu tham khảo

1. Văn Thế Minh (2008), ***Kỹ thuật vi xử lý,*** Nhà xuất bản Đại học Bách Khoa Hà Nội.
2. Quách Tuấn Ngọc(2003). Ngôn Ngữ Lập Trình C, Nhà xuất bản Thống Kê.
3. Phạm Mnh Hà(2008), Kỹ thuật mạch điện tử, Nhà xuất bản Khoa Học Kỹ Thuật.
4. http://www.diendandientu.com.
5. http://www.dientuvietnam.net.
6. http://www.picvietnam.com.
7. http://www.hocavr.com.