LỜI NÓI ĐẦU

Ngày nay, với những ứng dụng của khoa học kỹ thuật tiên tiến, thế giới của chúng ta đã và đang ngày một thay đổi, văn minh và hiện đại hơn. Trong đó, sự phát triển của kỹ thuật tự động hóa đã tạo ra hàng loạt những thiết bị với các đặc điểm nổi bật như sự chính xác, bảo mật cao, tốc độ nhanh, gọn nhẹ là nhữnh yếu tố rất cần thiết cho sự tiện lợi trong cuộc sống.

Ý tưởng đề tài xuất phát từ bài toàn thực tế. Một thiết bị vừa có thể đo nhiệt độ phòng tại một thời điểm xác định vừa có thể điều khiển thiết bị (động cơ). Với một giá trị nhiệt độ khác nhau mà hệ thống sẽ điều khiển tắt hay bật và thay đổi tốc độ động cơ. Đồng thời người dùng có thể thiết lập các giá trị ngưỡng theo đúng yêu cầu riêng.

Đề tài ***"Thiết kế hệ thống điều khiển động cơ DC bằng nhiệt độ"*** là sự kết hợp của nhiều mạch điện tử cơ bản cũng như sử dụng phần tử vi điều khiển trong chương trình giảng dạy, là sự tổng hợp kiến thức từ các môn cơ sở ngành và kỹ năng thực hành trong môn Vi Điều Khiển.

Đề tài của em gồm 3 phần:

Chương 1. Tổng quan về các phần tử

Chương 2. Thiết kế hệ thống điều khiển

Chương 3. Chương trình điều khiển

Để thực hiện được đồ án này em xin gửi lời cảm ơn chân thành nhất đến tất cả các thầy cô giáo, các cán bộ nhân viên trường Đại học dân lập Hải phòng nói chung và các thầy cô giáo trong khoa Điện – Điện tử nói riêng đã đạy dỗ, và giúp đỡ em suốt thời gian em học tại trường.

Trong quá trình làm đề tài, do sự hạn chế về thời gian, tài liệu và trình độ có hạn nên không tránh khỏi có thiếu sót. Em rất mong được sự đóng góp ý kiến của các thầy cô và các bạn để đồ án tốt nghiệp của em được hoàn thiện hơn. Em xin chân thành cảm ơn.

Hải Phòng, tháng 6 năm 2013

Sinh viên thực hiện

Mạc Minh Đức

Chương 1. TỔNG QUAN VỀ CÁC PHẦN TỬ

1.1. Tổng quan về PIC 16F887A

Thông thường có 4 họ vi điều khiển 8 bit chính là 6811 của Motorola, 8051 của Intel, z8 của Xilog và Pic 16 của Microchip Technology. Mỗi một loại trên đây đều có một tập lệnh và thanh ghi riêng duy nhất, nên chúng thường không tương thích lẫn nhau. Ngoài ra cũng có những bộ vi điều khiển 16 bit và 32 bit được sản xuất bởi các hãng khác nhau. Với tất cả những bộ vi điều khiển khác nhau thì tiêu chuẩn để lựa chọn là:

* Đáp ứng được nhu cầu tính toán của bài toán một cách hiệu quả, đầy đủ chức năng cần thiết và thấp nhất về mặt giá thành. Trong khi phân tích các nhu cầu của một dự án dựa trên bộ vi điều khiển chúng ta phải biết bộ vi điều khiển nào là 8 bit, 16 bit hay 32 bit có thể đáp ứng tốt nhất nhu cầu của bài toán một cách hiệu quả. Những tiêu chuẩn đó là:

- Tốc độ: tốc độ lớn nhất mà vi điều khiển hỗ trợ là bao nhiêu.

- Kiểu đóng vỏ: Đóng vỏ kiểu DIP 40 chân hay QFP. Đây là yêu cầu quan trọng xét về không gian, kiểu lắp ráp và tạo mẫu thử cho sản phẩm cuối cùng.

- Công suất tiêu thụ: Điều này đặc biệt khắt khe đối với các sản phẩm dùng pin, ắc quy.

- Dung lượng bộ nhớ Rom và Ram trên chíp.

- Số chân vào ra và bộ định thời trên chíp.

- Khả năng dễ dàng nâng cấp cho hiệu suất cao hoặc giảm công suất tiêu thụ.

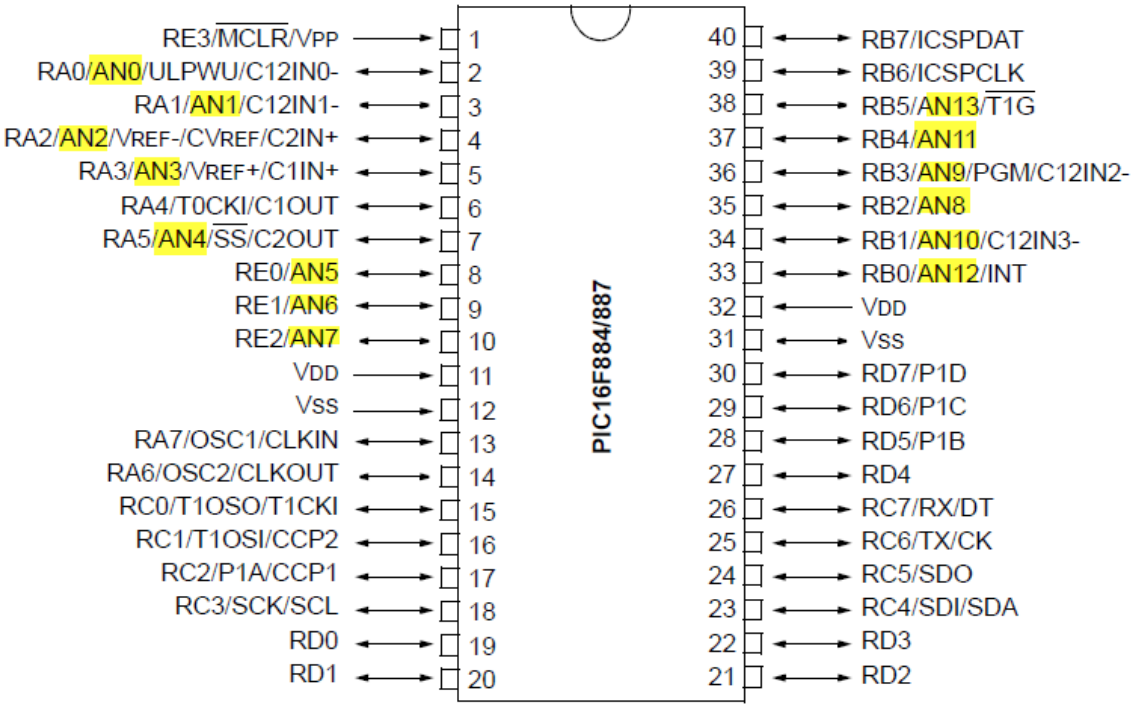
- Giá thành cho một đơn vị: Điều này quan trọng quyết định giá thành sản phẩm mà một bộ vi điều khiển được sử dụng.

*) Có sẵn các công cụ phát triển phần mềm như các trình biên dịch, trình hợp ngữ và gỡ rối.

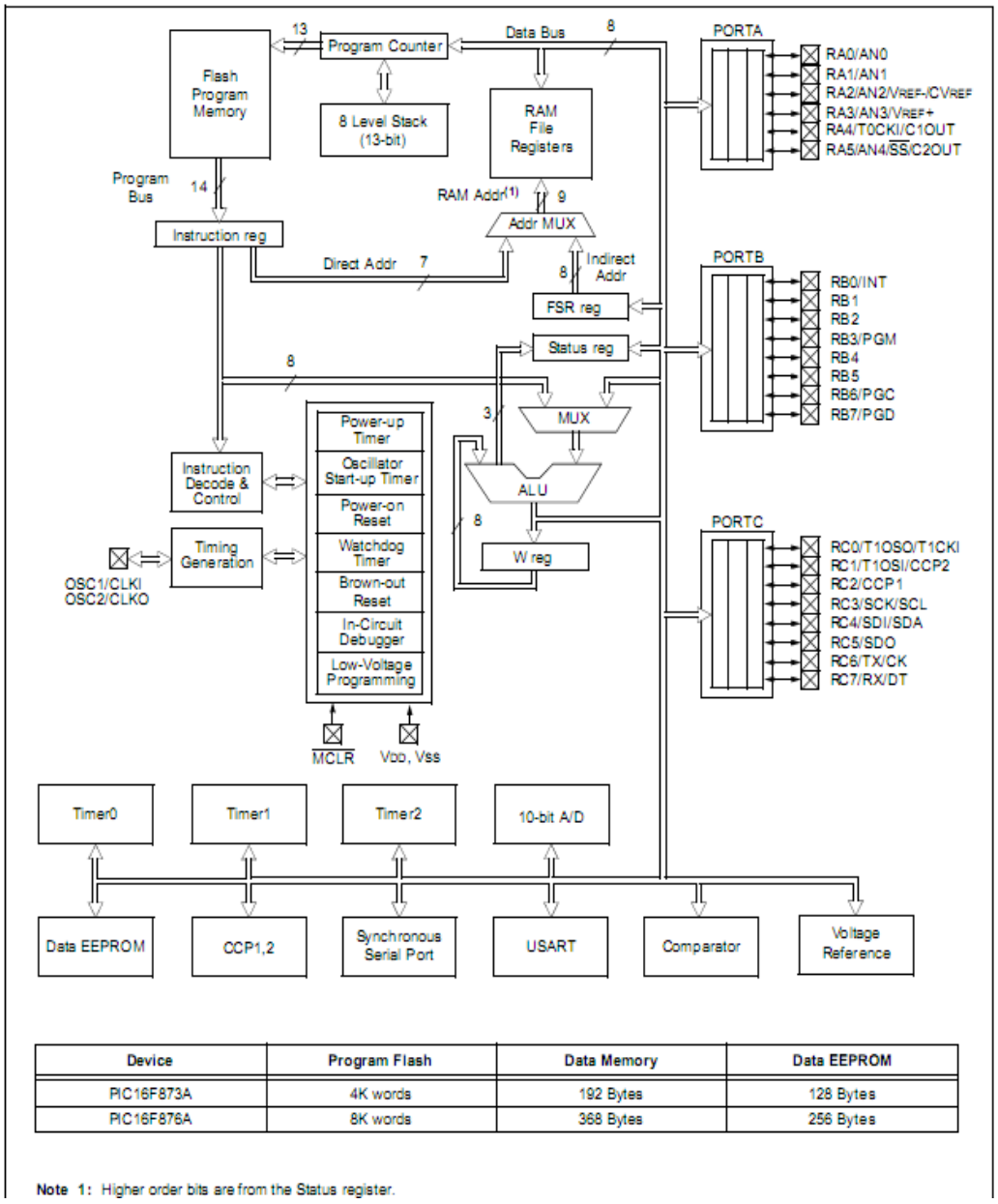
*) Nguồn các bộ vi điều khiển sẵn có nhiều và tin cậy. Khả năng sẵn sàng đáp ứng về số lượng trong hiện tại tương lai.

Hiện nay các bộ vi điều khiển 8 bit họ 8051 là có số lượng lớn nhất các nhà cung cấp đa dạng như Intel, Atmel, Philip… Nhưng về mặt tính năng và công năng thì có thể xem PIC vượt trội hơn rất nhiều so với 89 với nhiều module được tích hợp sẵn như ADC10 BIT, PWM 10 BIT, PROM 256 BYTE, COMPARATER, VERF COMPARATER, một đặc điểm nữa là tất cả các vi điều khiển PIC sử dụng thì đều có chuẩn PI tức chuẩn công nghiệp thay vì chuẩn PC (chuẩn dân dụng). Ngoài ra PIC còn được rất nhiều nhà sản xuất phần mềm tạo ra các ngôn ngữ hỗ trợ cho việc lập trình ngoài ngôn ngữ Asembly ra còn có thể sử dụng ngôn ngữ C thì sử dụng CCSC, HTPIC hay sử dụng Basic thì có MirkoBasic… và còn nhiều chương trình khác nữa để hỗ trợ cho việc lập trình bên cạnh ngôn ngữ kinh điển là asmbler. Nên trong đề tài này em lựa chọn sử dụng vi điều khiển PIC làm bộ điều khiển chính, và ở đây là PIC16F887A.

1.1.1. Sơ đồ khối và bảng mô tả chức năng các chân của PIC16F887A



Hình 1.1. PIC 16F887A



Device	Program Flash	Data Memory	Data EEPROM
PIC16F873A	4K words	192 Bytes	128 Bytes
PIC16F876A	8K words	368 Bytes	256 Bytes

Note 1: Higher order bits are from the Status register.

Hình 1.2. Sơ đồ khối của PIC16F887A

Bảng mô tả chức năng các chân của PIC16F887A

Pin Name	DIP Pin#	PLCC Pin#	QFT Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	13	14	30	1	ST/CMOS(4)	Đầu vào của xung dao động thạch anh/ngõ vào xung clock ngoại

OSC2/CLKOUT	1	2	18	O	-	Đầu ra của xung dao động thạch anh. Nối với thạch anh hay cộng hưởng trong chế độ dao động của thạch anh.Trong chế độ RC, ngõ ra của chân OSC2.
$\overline{MCLR}$ /V_{pp}	1	2	18	I/P	ST	Ngõ vào của Master Clear(Reset) hoặc ngõ vào điện thế được lập trình. Chân này cho phép tín hiệu Reset thiết bị tác động ở mức thấp.
RA0/AN0	2	3	19	I/O	TTL	PORTA là port vào ra hai chiều. RA0 có thể làm ngõ vào tương tự thứ 0.
RA1/AN1	3	4	20	I/O	TTL	RA1 có thể làm ngõ vào tương tự thứ 1
RA2/AN2/VREF –	4	5	21	I/O	TTL	RA2 có thể làm ngõ vào tương tự 2 hoặc điện áp chuẩn tương tự âm.
RA3/AN3/VREF +	5	6	22	I/O	TTL	RA3 có thể làm ngõ vào tương tự 3 hoặc điện áp chuẩn tương tự dương.
RA4/T0CKI	6	7	23	I/O	ST	RA4 có thể làm ngõ vào xung clock cho bộ định thời Timer0.
RA5/ $\overline{SS}$ /AN4	7	8	24	I/O	TTL	RA5 có thể làm ngõ vào tương tự thứ 4
RB0/INT RB1 RB2	33 34 35	36 37 38	8 9 10	I/O I/O I/O	TTL/ST(1) TTL TTL	PORTB là port hai chiều. RB0 có thể làm chân ngắt ngoà
RB3/PGM	36	39	11	I/O	TTL	RB3 có thể làm ngõ vào của điện thế được lập trình ở mức thấp.

RB4	37	41	14	I/O	TTL	Interrupt-on-change pin.
RB5	38	42	15	I/O	TTL	Interrupt-on-change hoặc In-Crcuit Debugger pin.
RB6/PGC	39	43	16	I/O	TTL/ST(2)	Serial programming clock.
RB7/PGD	40	44	17	I/O	TTL/ST(3)	Interrupt-on-change pin hoặc In-Crcuit Debugger pin . Serial programming data .
RC0/T1OSO/T1CKI	15	16	32	I/O	ST	PORTC là port vào ra hai chiều. RC0 có thể là ngõ vào của bộ dao động Timer1 hoặc ngõ xung clock cho Timer1
RC1/T1OSI/CCP2	16	18	35	I/O	ST	RC1 có thể là ngõ vào của bộ dao động Timer1 hoặc ngõ vào Capture2/ngõ ra compare2/ngõ vào PWM2.
RC2/CCP1	17	19	36	I/O	ST	RC2 có thể ngõ vào capture1/ngõ ra compare1/ngõ vào PWM1
RC3/SCK/SCL	18	20	37	I/O	ST	RC3 có thể là ngõ vào xung
RC4/SDI/SDA	23	25	42	I/O	ST	Clock đồng bộ nội tiếp/ngõ ra trong cả hai chế độ SPI và I2C RC4 có thể là dữ liệu bên trong SPI(chế độ SPI) hoặc dữ liệu I/O(chế độ I^2C).
RC5/SDO	24	26	43	I/O	ST	RC5 có thể là dữ liệu ngoài SPI(chế độ SPI)
RC6/TX/CK	25	27	44	I/O	ST	RC6 có thể là chân truyền không đồng bộ USART hoặc đồng bộ với xung đồng hồ
RC7/RX/DT	26	29	1	I/O	ST	RC7 có thể là chân nhận không đồng bộ USART hoặc đồng bộ với dữ liệu.
RD0/PSP0	19	21	38	I/O	ST/TTL(3)	PORTD là port vào ra hai chiều hoặc là parallel slave port khi giao tiếp với bus của bộ vi xử lý.
RD1/PSP1	20	22	39	I/O	ST/TTL(3)	
RD2/PSP2	21	23	40	I/O	ST/TTL(3)	
RD3/PSP3	22	24	41	I/O	ST/TTL(3)	
RD4/PSP4	27	30	2	I/O	ST/TTL(3)	
RD5/PSP5	28	31	3	I/O	ST/TTL(3)	
RD6/PSP6	29	32	4	I/O	ST/TTL(3)	
RD7/PSP7	30	33	5	I/O	ST/TTL(3)	

RE0/$\overline{RD}$/AN5	8	9	25	I/O	ST/TTL(3)	PORTE là port vào ra hai chiều. RE0 có thể điều khiển việc đọc parrallel slave port hoặc là ngoc vào tương tự thứ 5.
RE1/$\overline{WR}$/AN6	9	10	26	I/O	ST/TTL(3)	RE1 có thể điều khiển việc ghi parallel slave port hoặc là ngõ vào tương tự thứ 6.
RE2/$\overline{CS}$/AN7	10	11	27	I/O	ST/TTL(3)	RE2 có thể điều khiển việc chọn parallel slave port hoặc là ngõ vào tương tự thứ 7
V_{ss} V_{DD}	12, 31 11, 32	13, 34 12, 35	7, 28 6, 29	P P		Cung cấp nguồn dương cho các mức logicvà những chân I/O.
NC		1,17,28, 40	12,13 33, 4			Những chân này không được nối bên trong và nó được để trống

Ghi chú: I = input; O = output; I/O = input/output; P = power

- = Not used; TTL = TTL input; ST = Schmitt Trigger input

1. Là vùng đệm có ngõ vào Trigger Schmitt khi được cấu hình như ngắt ngoài.

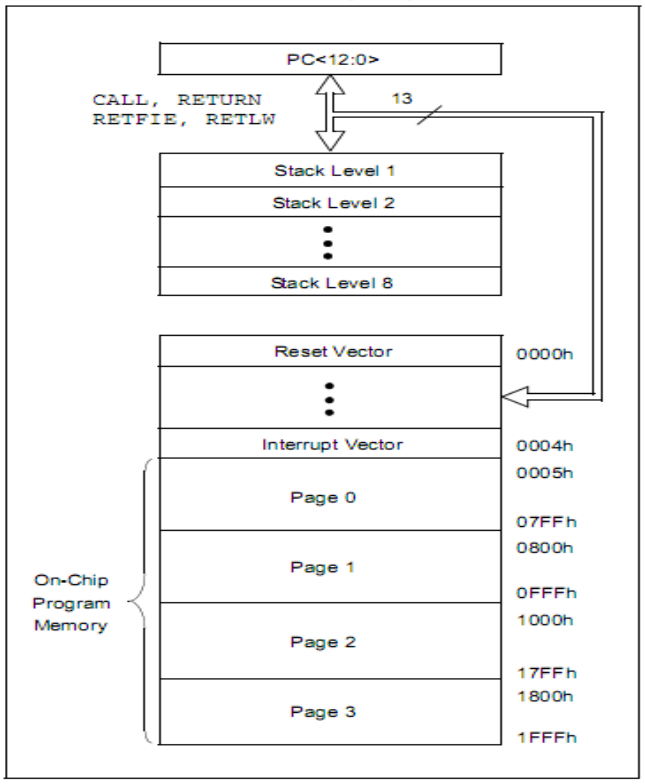
2. Là vùng đệm có ngõ vào Trigger Schmitt khi được sử dụng trong chế độ 9 Serial Programming.

3. Là vùng đệm có ngõ vào Trigger Schmitt khi được cấu hình như ngõ vào ra mục đích chung và là ngõ vào TTL khi sử dụng trong chế độ Parallel Slave Port (cho việc giao tiếp với các bus của bộ vi xử lý).

4. Là vùng đệm có ngõ vào Trigger Schmitt khi được cấu hình trong chế độ dao động RC và một ngõ vào CMOS khác.

1.1.2. Tổ chức bộ nhớ

Có 2 khối bộ nhớ trong các vi điều khiển họ PIC16F88X, bộ nhớ chương trình và bộ nhớ dữ liệu, với những bus riêng biệt để có thể truy cập đồng thời.



Hình 1.3. Ngăn xếp và bản đồ bộ nhớ chương trình PIC16F887A

1.1.2.1. Tổ chức của bộ nhớ chương trình

Các vi điều khiển họ PIC16F887A có bộ đếm chương trình 13 bit có khả năng định vị không gian bộ nhớ chương trình lên đến 8Kb.Các IC PIC16F887A có 8Kb bộ nhớ chương trình FLASH, các IC PIC16F873/874 chỉ có 4 Kb.Vectơ RESET đặt tại địa chỉ 0000h và vectơ ngắt tại địa chỉ 0004h.

1.1.2.2. Tổ chức bộ nhớ dữ liệu

Bộ nhớ dữ liệu được chia thành nhiều dãy và chứa các thanh ghi mục đích chung và các thanh ghi chức năng đặc biệt. BIT RP1 (STATUS <6>) và RP0 (STATUS <5>) là những bit dùng để chọn các dãy thanh ghi.

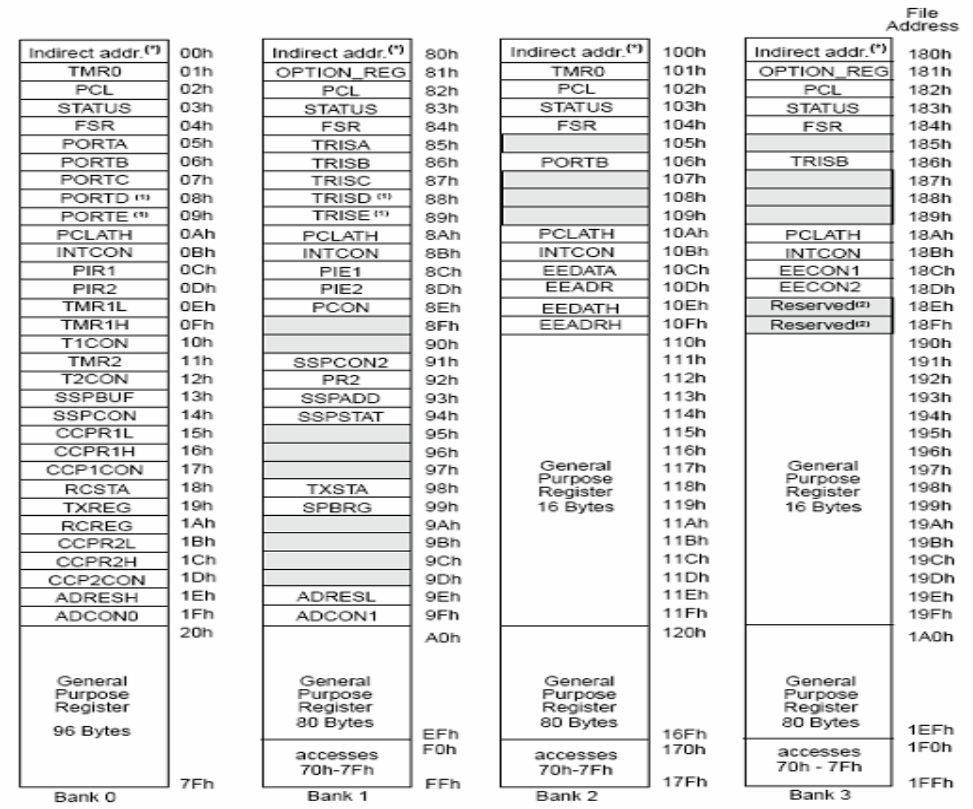
RP1:RP0	Bank
00	0
01	1

10	**2**
11	**3**

Chiều dài của mỗi dãy là 7Fh (128 byte). Phần thấp của mỗi dãy dùng để chứa các thanh ghi chức năng đặc biệt.Trên các thanh ghi chức năng đặc biệt là các thanh ghi mục đích chung, có chức năng như RAM tĩnh. Thường thì những thanh ghi đặc biệt được sử dụng từ một dãy và có thể được ánh xạ vào những dãy khác để giảm bớt đoạn mã và khả năng truy cập nhanh hơn.

1.1.2.3. Các thanh ghi mục đích chung

Các thanh ghi này có thể truy cập trực tiếp hoặc gián tiếp thông qua thanh ghi FSG (File Select Register).



Hình 1.4. Các thanh ghi của PIC16F887A

1.1.2.4. Các thanh ghi chức năng đặc biệt

Các thanh ghi chức năng đặc biệt (Special Function Resgister) được sử dụng bởi CPU và các bộ nhớ ngoại vi để điều khiển các hoạt động được yêu cầu của thiết bị. Những thanh ghi này có chức năng như RAM tĩnh. Danh sách những thanh ghi nay được trình bày ở bảng dưới. Các thanh ghi chức năng đặc biệt có thể chia thành hai loại: phần trung tâm (CPU) và phần ngoại vi.

1.1.2.5. Các thanh ghi trạng thái

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

bit 7 **IRP:** Register Bank Select bit (used for indirect addressing)
1 = Bank 2, 3 (100h-1FFh)
0 = Bank 0, 1 (00h-FFh)

bit 6-5 **RP1:RP0**: Register Bank Select bits (used for direct addressing)
11 = Bank 3 (180h-1FFh)
10 = Bank 2 (100h-17Fh)
01 = Bank 1 (80h-FFh)
00 = Bank 0 (00h-7Fh)
Each bank is 128 bytes.

bit 4 $\overline{TO}$: Time-out bit
1 = After power-up, CLRWDT instruction or SLEEP instruction
0 = A WDT time-out occurred

bit 3 $\overline{PD}$: Power-down bit
1 = After power-up or by the CLRWDT instruction
0 = By execution of the SLEEP instruction

bit 2 **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 **DC**: Digit carry/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
(for $\overline{borrow}$, the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0 **C**: Carry/$\overline{borrow}$ bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

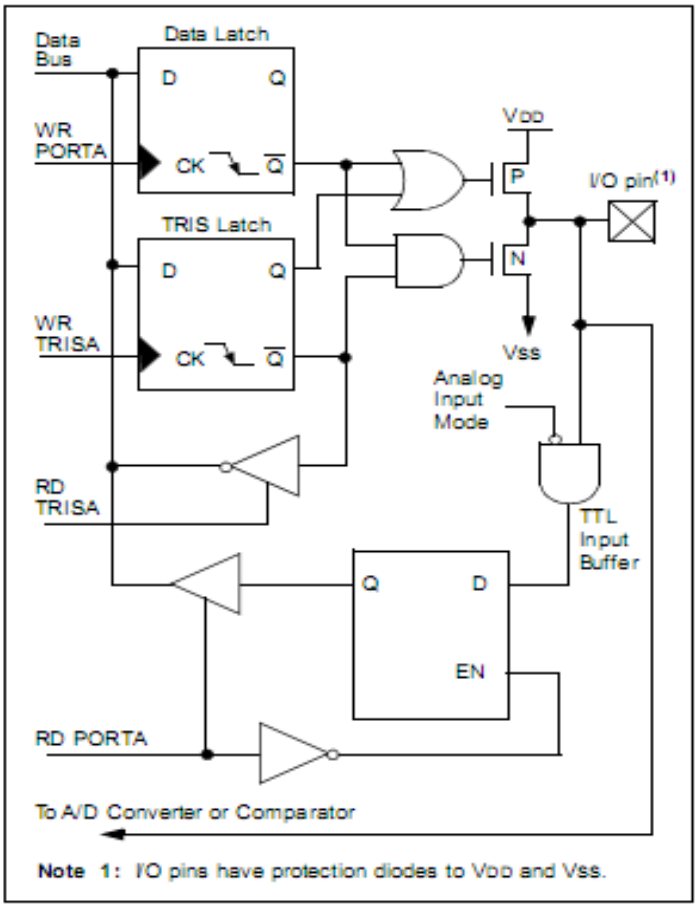
Hình 1.5. Thanh ghi trạng thái (địa chỉ 03h, 83h, 103h, 183h)

Thanh ghi trạng thái chứa các trạng thái số học của bộ ALU, trạng thái RESET và những bits chọn dãy thanh ghi cho bộ nhớ dữ liệu. Thanh ghi trạng thái có thể là đích cho bất kì lệnh nào, giống như những thanh ghi khác. Nếu thanh ghi trang thái là đích cho một lệnh mà ảnh hưởng đến các cờ Z, DC hoặc C, và sau đó những bit này sẽ được vô hiệu hoá. Những bit này có thể đặt hoặc xoá tuỳ theo trạng thái logic của thiết bị. Hơn nữa hai bit $\overline{TO}$ và PD thì không cho phép ghi, vì vậy kết quả của một tập lệnh mà thanh ghi trạng thái là đích có thể

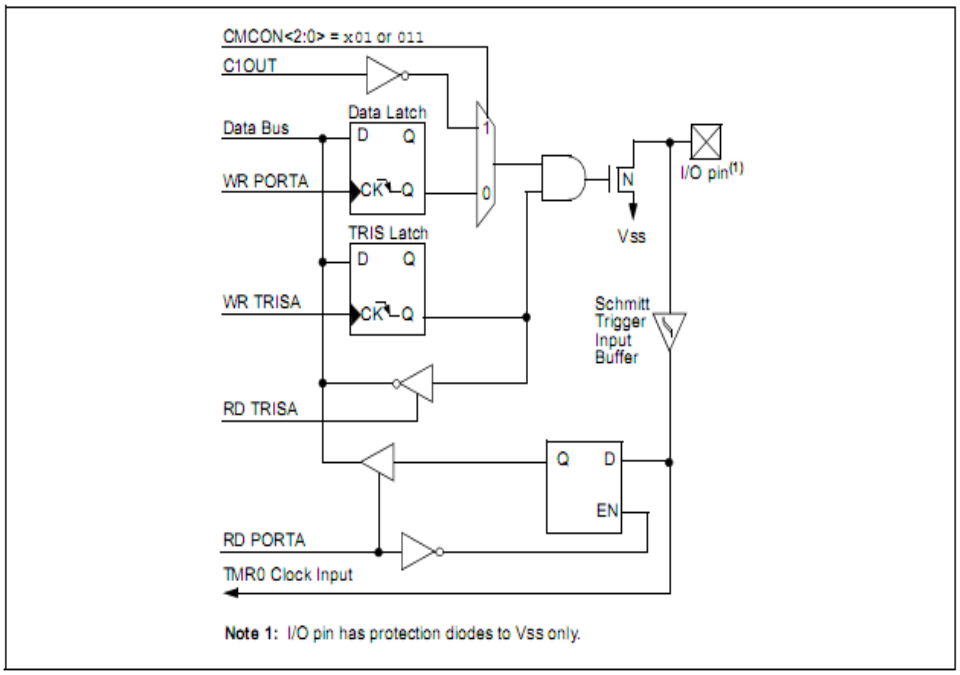
khác hơn dự định. Ví dụ, CLRF STATUS sẽ soá 3 bit cao nhất và đặt bit Z. Lúc này các bits của thanh ghi trạng thái là 000u u1uu (u = unchanged). Chỉ có các lệnh BCF, BSF, SWAPF và MOVWF được sử dụng để thay đổi thanh ghi trạng thái, bởi vì những lệnh này không làm ảnh hưởng đến các bit Z, DC hoặc C từ thanh ghi trạng thái. Đối với những lệnh khác thì không ảnh hưởng đến những bits trạng thái này.

1.1.3. Các cổng của PIC 16F887A

1.1.3.1. PORTA và thanh ghi TRISA



Hình 1.6. Sơ đồ khối của chân RA3:RA0 và RA5



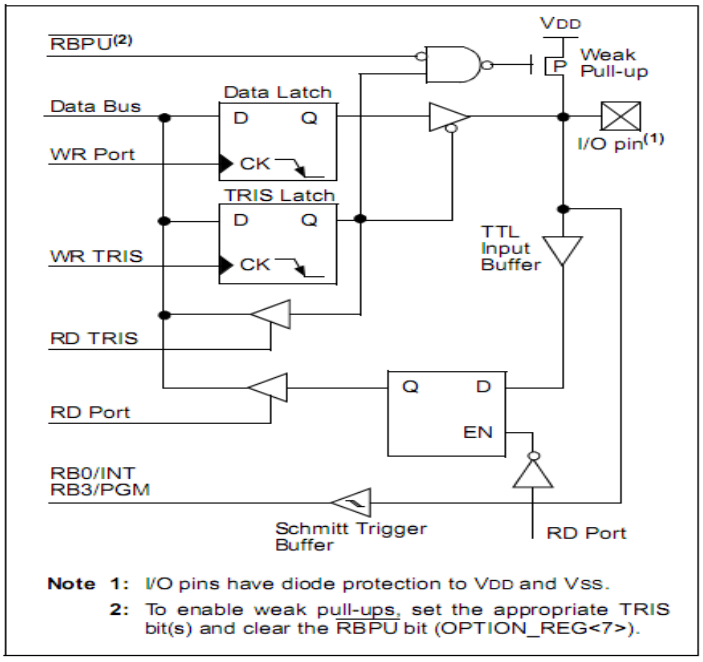
Hình 1.7. Sơ đồ khối của chân RA4/T0CKI

1.1.3.2. PORTB và thanh ghi TRISB

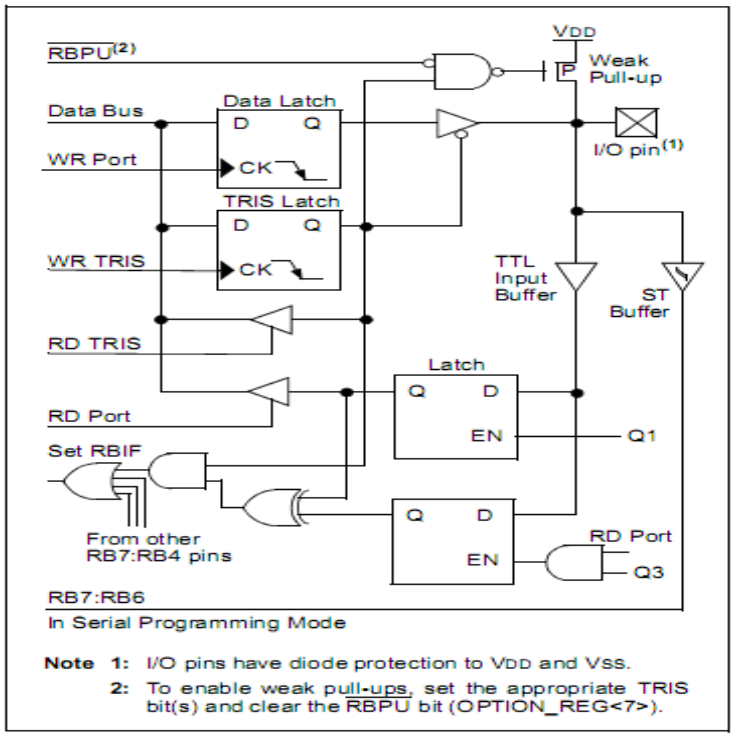
PORTB có độ rộng 8 bit, là port vào ra hai chiều. Ba chân của PORTB được đa hợp với chức năng lập trình mức điện thế thấp (Low Voltage Programming): RB3/PGM, RB6/PGC và RB7/PGD. Mỗi chân của PORTB có một điện trở kéo bên trong. Một bit điều khiển có thể mở tất cả những điện trở kéo này lên. Điều này được thực hiện bằng cách xoá bit $\overline{RBPU}$ (OPTION_REG<7>). Những điện trở này bị cấm khi có một Power-on Reset. Bốn chân của PORTB: RB7 đến RB4 có một ngắt để thay đổi đặc tính .Chỉ những chân được cấu hình như ngõ vào mới có thể gây ra ngắt này. Những chân vào (RB7:RB4) được so sánh với giá trị được chốt trước đó trong lấn đọc cuối cùng của PORTB. Các kết quả không phù hợp ở ngõ ra trên chân RB7:RB4 được OR với nhau để phát ra một ngắt Port thay đổi RB với cờ ngắt là RBIF (INTCON<0>). Ngắt này có thể đánh thức thiết bị từ trạng thái nghỉ (SLEEP). Trong thủ tục phục vụ ngắt người sử dụng có thể xoá ngắt theo cách sau:

a) Đọc hoặc ghi bất kì lên PORTB. Điều này sẽ kết thúc điều kiện không hoà hợp.

b) Xoá bit cờ RBIF.



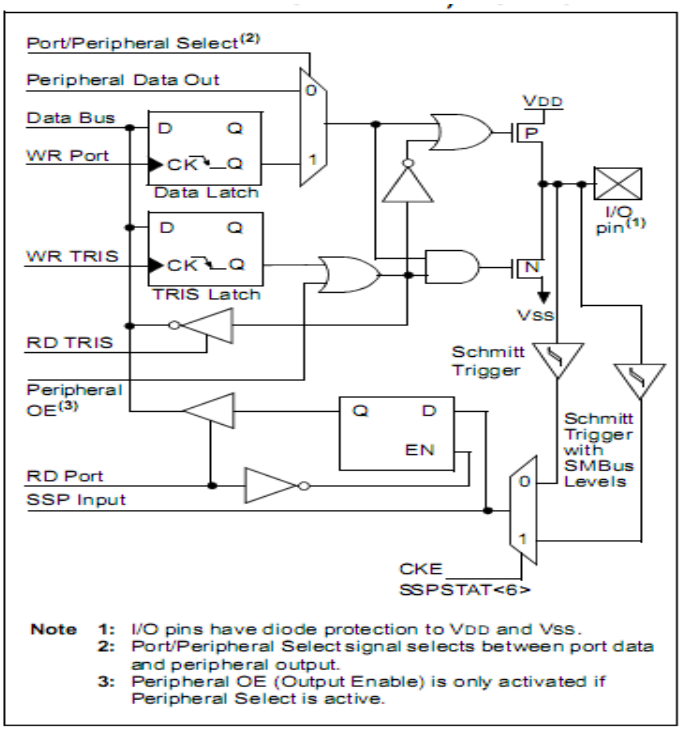
Hình 1.8. Sơ đồ khối các chân RB3:RB0



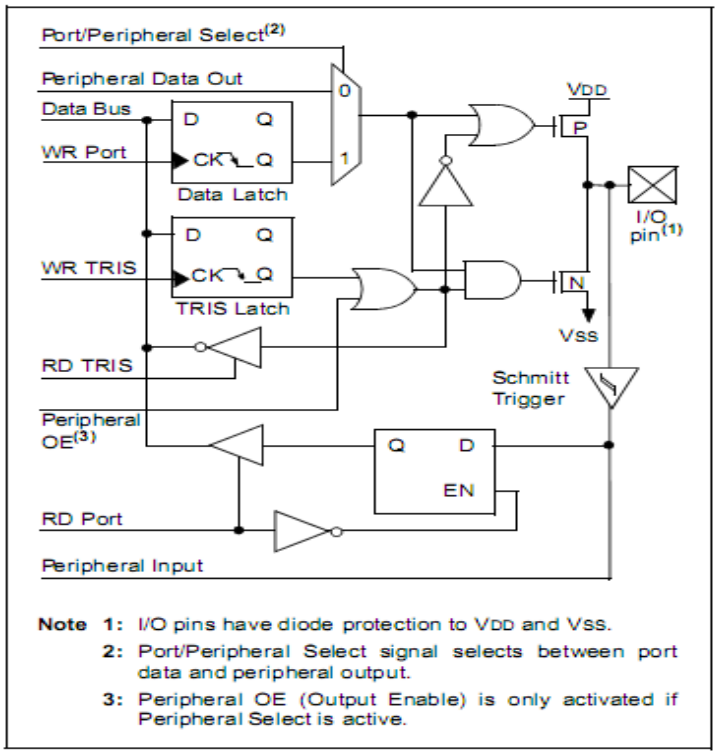
Hình 1.9. Sơ đồ khối các chân RB7:RB4

1.1.3.3. PORTC và thanh ghi TRISC

PORTC có độ rộng là 8 bit, là port hai chiều. Thanh ghi dữ liệu trực tiếp tương ứng là TRISC. Cho tất cả các bit của TRISC là 1 thì các chân tương ứng ở PORTC là ngõ vào. Cho tất cả các bit của TRISC là 0 thì các chân tương ứng ở PORTC là ngõ ra. PORTC được đa hợp với vài chức năng ngoại vi, những chân của PORTC có đệm Trigger Schmitt ở ngõ vào. Khi bộ I2C được cho phép, chân 3 và 4 của PORTC có thể cấu hình với mức I2C bình thường, hoặc với mức SMBus bằng cách sử dụng bit CKE (SSPSTAT<6>). Khi những chức năng ngoại vi được cho phép, chúng ta cần phải quan tâm đến việc định nghĩa các bits của TRIS cho mỗi chân của PORTC. Một vài thiết bị ngoại vi ghi đè lên bit TRIS thì tạo nên một chân ở ngõ ra, trong khi những thiết bị ngoại vi khác ghi đè lên bit TRIS thì sẽ tạo nên một chân ở ngõ vào. Khi những bit TRIS ghi đè bị tác động trong khi thiết bị ngoại vi được cho phép, những lệnh đọc thay thế ghi (BSF, BCF, XORWF) với TRISC là nơi đến cần phải được tránh. Người sử dụng cần phải chỉ ra vùng ngoại vi tương ứng để đảm bảo cho việc đặt TRIS bit là đúng.



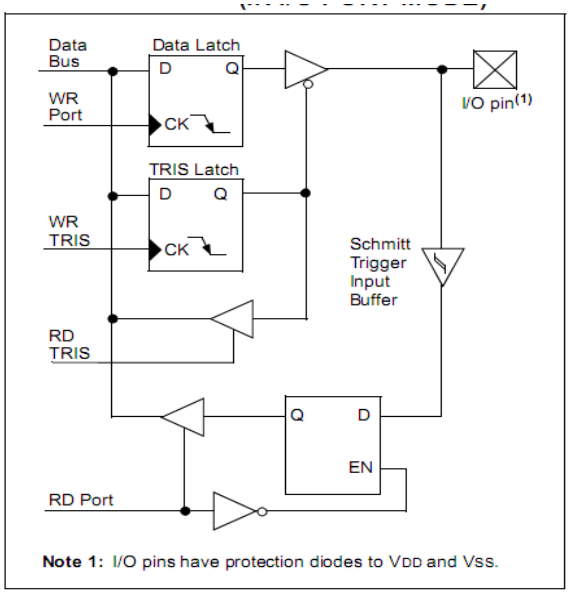
Hình 1.10. Sơ đồ khối của các chân RC<4:3>



Hình 1.11. Sơ đồ khối của các chân RC<2:0> và RC<7:5>

1.1.3.4. PORTD và thanh ghi TRISD

PORTD là port 8 bit với đệm Trigger Schmitt ở ngõ vào. Mỗi chân có thể được cấu hình riêng lẻ như một ngõ vào hoặc ngõ ra. PORTD có thể được cấu hình như port của bộ vi xử lý rộng 8 bit (parallel slave port) bằng cách đặt bit điều khiển PSPMIDE (TRISE <4>). Trong chế độ này, đệm ở ngõ vào là TTL.

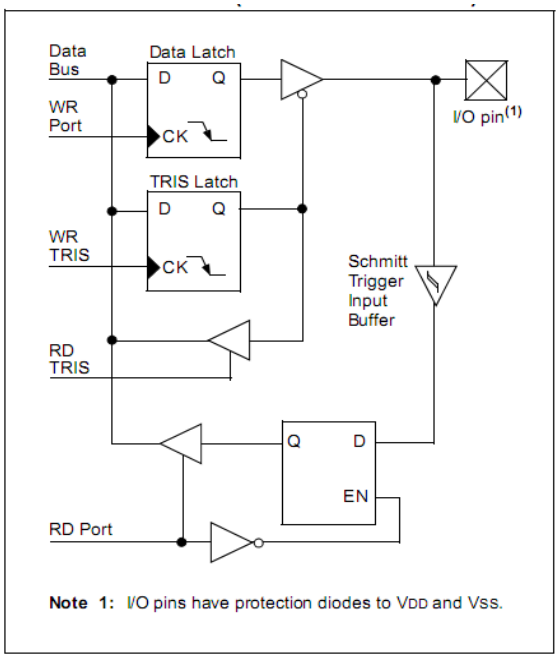


Hình 1.12. Sơ đồ khối của PORTD (trong chế độ là port I/O)

1.1.3.5. PORTE và thanh ghi TRISE

PORTE có ba chân (RE0/RD/AN5, RE1/WR/AN6, và RE2/CS/AN7) mỗi chân được cấu hình riêng lẻ như những ngõ vào hoặc những ngõ ra. Những chân này có đệm Trigger Schmitt ở ngõ vào. Những chân của PORTE đóng vai trò như những ngõ vào điều khiển vào ra cho Port của vi xử lý khi bit PSPMODE (TRISE <4>) được đặt. Trong chế độ này, người sử dụng cần phải chắc chắn rằng những bit TRISE <2:0> được đặt, và chắc rằng những chân này được cấu hình như những ngõ vào số. Cũng bảo đảm rằng ADCON1 được cấu hình cho vào ra số. Trong chế độ này, những đệm ở ngõ vào là TTL.

Những chân của PORTE được đa hợp với những ngõ vào tương tư, Khi được chọn cho ngõ vào tương tự, những chân này sẽ đọc giá trị "0". TRISE điều khiển hướng của những chân RE chỉ khi những chân này được sử dụng như những ngõ vào tương tự. Người sử dụng cần phải giữ những chân được cấu hình như những ngõ vào khi sử dụng chúng như những ngõ vào tương tự.



Hình 1.13. Sơ đồ khối của PORTE (trong chế độ I/O port)

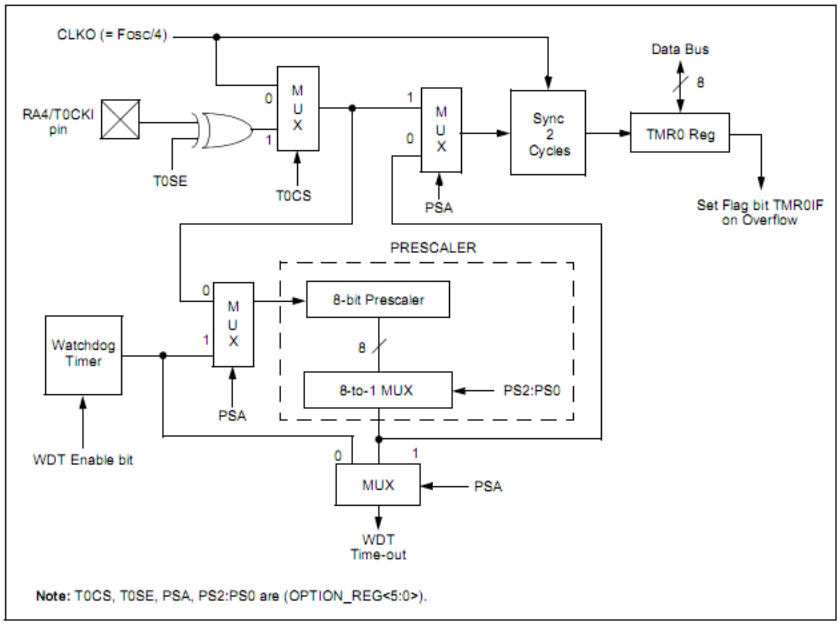
1.1.4. Hoạt động cuả định thời

1.1.4.1. Bộ định thời TIMER0

Bộ định thời/bộ đếm Timer0 có các đặc tính sau:

- Bộ định thời / bộ đếm 8 bit
- Cho phép đọc và ghi
- Bộ chia 8 bit lập trình được bằng phần mềm
- Chọn xung clock nội hoặc ngoại
- Ngắt khi có sự tràn từ FFh đến 00h
- Chọn sườn cho xung clock ngoài

Sơ đồ khối của bộ định thời Timer0 và bộ chia dùng chung với WDT được đưa ra trong hình 1.14.



Hình 1.14. Sơ đồ khối của bộ định thời Timer0 và bộ chia dùng chung với WDT

Chế độ định thời (Timer) được chọn bằng cách xoá bit T0CS (OPTION_REG<5>). Trong chế độ định thời, bộ định thời Timer0 sẽ tăng dần sau mỗi chu kì lệnh (không có bộ chia). Nếu thanh ghi TmR0 được ghi thì sự tăng sẽ bị ngăn lại sau hai chu kì lệnh.

Chế độ đếm (Counter) được chọn bằng cách xoá bit T0CS (OPTION_REG<5>). Trong chế độ đếm, Timer0 sẽ tăng dần ở mỗi cạnh lên xuống của chân RA4/T0CKI. Sự tăng sườn được xác định bởi bit Timer0 Source Edge Select, T0SE (OPTION_RE<4>). Bộ chia chỉ được dùng chung qua lại giữa bộ định thời Timer0 và bộ định thời Watchdog. Bộ chia không cho phép đọc hoặc ghi

Ngắt Timer0

Ngắt TMR0 được phát ra khi thanh ghi TMR0 tràn từ FFh đến 00h. Sự tràn này sẽ đặt bit T0IF (INTCON<2>). Ngắt này có thể được giấu đi bằng cách xóa bit T0IE (INTCON<5>) . Bit T0IF cần phải được xóa trong chương trình bởi thủ tục phục vụ ngắt của bộ định thời Timer0 trước khi ngắt này được cho phép lại.

Sử dụng Timer0 với xung clock ngoại

Khi bộ chia không được sử dụng, clock ngoài đặt vào thì giống như bộ chia ở ngõ ra. Sự đồng bộ của chân T0CKI với clock ngoài được thực hiện bằng cách lấy mẫu bộ chia ở ngõ ra trên chân Q2 và Q4. Vì vậy thực sự cần thiết để chân T0CKI ở mức cao trong ít nhất 2 chu kỳ máy và ở mức thấp trong ít nhất 2 chu kỳ máy.

Bộ chia

Thiết bị PIC16F87X chỉ có một bộ chia mà được dùng chung bởi bộ định thời TIMER0 và bộ định thời Watchdog. Bộ chia có các Hệ số chia dùng cho Timer0 hoặc bộ WDT. Các hệ số này không có khả năng đọc và khả năng viết. Để chọn hệ số chia xung vào Timer0 hoặc cho bộ WDT ta tiến hành xoá hoặc đặt bit PSA của thanh ghi OPTION_REG<3>.

Những bit PS2, PS1, PS0 của thanh ghi OPTION_REG<2:0> dùng để xác lập các hệ số chia.

1.1.4.2. Bộ định thời TIMER1

Bộ định thời TIMER1 là một bộ định thời/bộ đếm 16 bit gồm hai thanh ghi TMR1H (Byte cao) và TMR1L (byte thấp) mà có thể đọc hoặc ghi. Cặp thanh ghi này tăng số đếm từ 0000h đến FFFFh và báo tràn sẽ xuất hiện khi có sự chuyển số đếm từ FFFFh xuống 0000h. Ngắt, nếu được phép có thể phát ra khi có số đếm tràn và được đặt ở bit cờ ngắt TMR1IF. Ngắt có thể được phép hoặc cấm bằng cách đặt hoặc xoá bit cho phép ngắt TMR1IE.

Bộ định thời Timer1 có thể được cấu hình để hoạt động một trong hai chế độ sau:

- Định thời một khoảng thời gian (timer)
- Đếm sự kiện (Counter)

Việc lựa chọn một trong hai chế độ được xác định bằng cách đặt hoặc xoá bit điều khiển TMR1ON.

--	--	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	TMR1ON

Bit7 Bit0

Bit 7-6 Không được định nghĩa

Bit 5-4 bit chọn bộ chia clock cho timer1

Bit 3 bit điều khiển cho phép bộ dao động Timer1

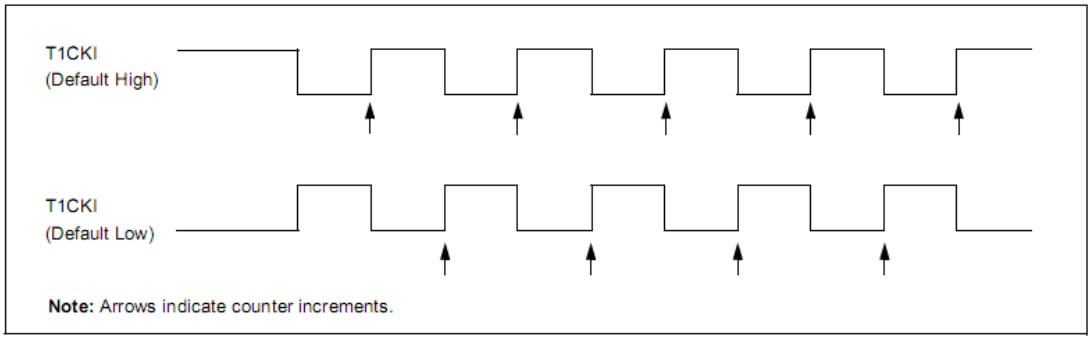
Bit 2 bit điều khiển clock ngoài Timer

Bit 1 bit chọn nguồn clock cho Timer1

Bit 0 bit điều khiển hoạt động của Timer1

Chế độ Timer

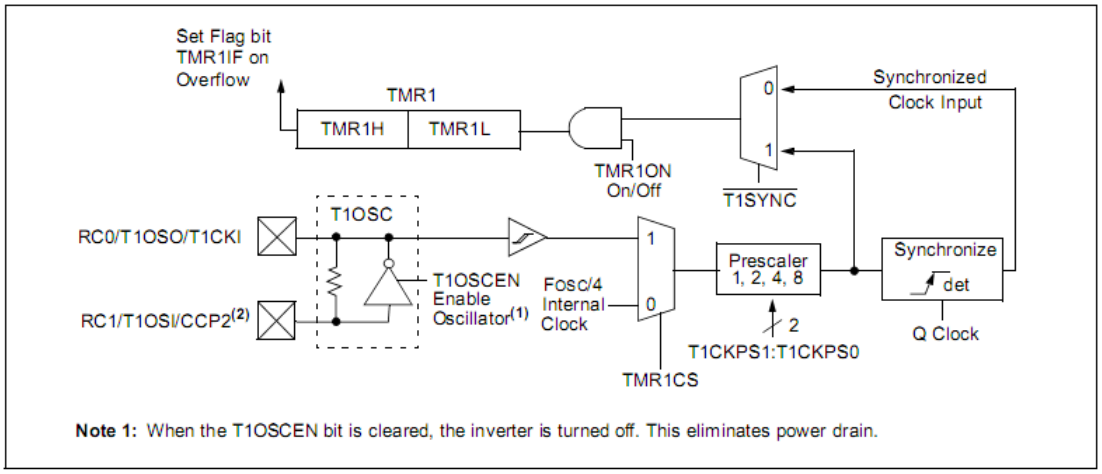
Chế độ Timer được chọn bằng cách xoá TMR1CS. Trong chế độ này, Nguồn clock đặt vào Timer là mạch dao động $F_{OSC}/4$. Bit điều khiển đồng bộ không bị tác động vì clock ngoài luôn luôn đồng bộ.



Hình 1.15. Cạnh tăng timer1

Chế độ counter

Trong chế độ này, bộ định thời tăng số đếm qua clock ngoài. Việc tăng xảy ra sau mỗi sườn lên của xung clock ngoài. Bộ định thời phải có một sườn lên trước khi việc đếm bắt đầu.



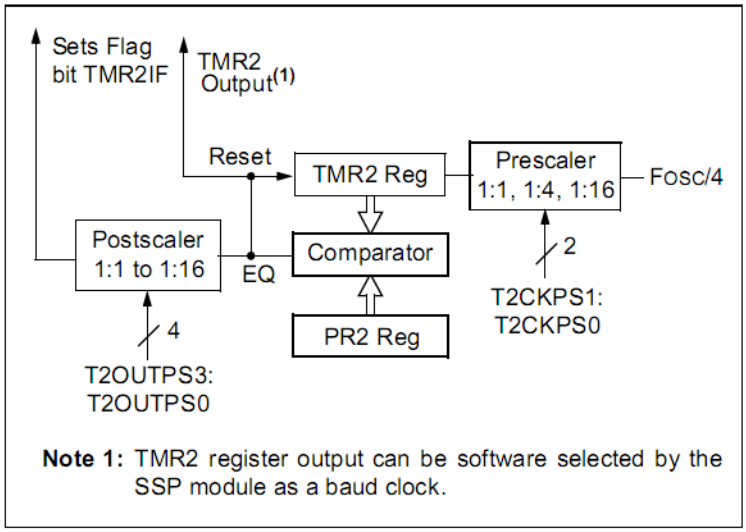
Hình 1.16. Sơ đồ khối bộ định thời timer1

1.1.4.3. Bộ định thời TIMER2

Bộ định thời TIMER2 là bộ định thời 8 bit với một bộ đếm và một bộ potscaler. Nó thường dùng chung với bộ CCP trong chế độ PWM (sẽ được đề cập ở phần sau). Thanh ghi TMR2 có thể đọc hoặc ghi và được xoá khi có bất kì tín hiệu reset nào của thiết bị

Bộ định thời TIMER2 có một thanh ghi chu kỳ 8 bit, PR2. Bộ định thời tăng số đếm lên từ 00h đến giá trị được ghi trong thanh ghi TR2 và sau đó reset lại giá trị 00h trong chu kỳ kế tiếp. PR2 là thanh ghi có thể đọc hoặc ghi.

Giá trị trùng hợp trong thanh ghi TMR2 được đi qua bộ postscaler 4 bit để phát ra một ngắt TMR2 (được đặt ở bit cờ ngắt TMR2IF). Bộ định thời TIMER2 có thể được tắt (không hoạt động) bằng cách xoá bít điều khiển TMR2ON để giản thiểu công suất tiêu tán nguồn.



Hình 1.17. Sơ đồ khối của TIMER2

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6-3 **TOUTPS3:TOUTPS0**: Timer2 Output Postscale Select bits
0000 = 1:1 postscale
0001 = 1:2 postscale
0010 = 1:3 postscale
•
•
•
1111 = 1:16 postscale

bit 2 **TMR2ON**: Timer2 On bit
1 = Timer2 is on
0 = Timer2 is off

bit 1-0 **T2CKPS1:T2CKPS0**: Timer2 Clock Prescale Select bits
00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

Hình 1.18. T2CON: Thanh ghi điều khiển Timer2 (địa chỉ 12h)

Một đặc điểm khác của vi điều khiển Pic16F887A là có bộ dao động chủ trên chip điều, nó sẽ giúp tránh được những sai số không cần thiết trong việc tạo

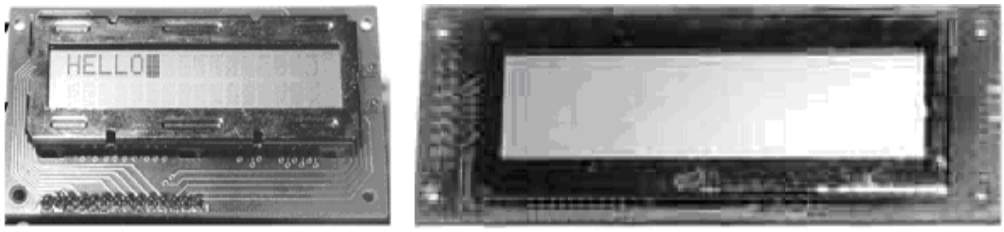
xung dao động, vi điều khiển Pic16F887A có khả năng tự Reset bằng bộ WDT, và có thêm 256 byte EEPROM. Nhưng giá thành của Pic đắt hơn so với 8051.

1.2. Thiết bị hiện thị LCD.

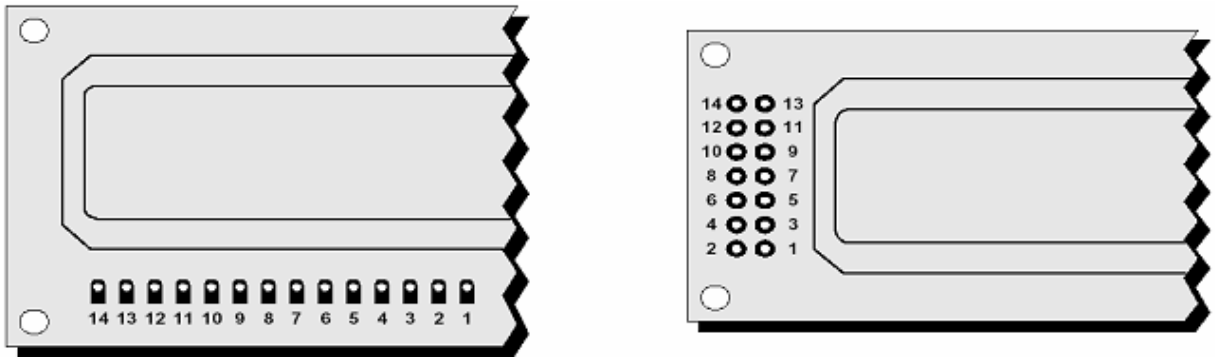
Ngày nay, thiết bị hiển thị LCD (Liquid Crystal Display) được sử dụng trong rất nhiều các ứng dụng của VĐK. LCD có rất nhiều ưu điểm so với các dạng hiển thị khác như nó có khả năng hiển thị kí tự đa dạng, trực quan (chữ, số và kí tự đồ họa), dễ dàng đưa vào mạch ứng dụng theo nhiều giao thức giao tiếp khác nhau, tốn rất ít tài nguyên hệ thống và giá thành rẻ ... Trong đề tài này tôi sử dụng HD44780 của Hitachi, một loại thiết bị hiển thị LCD rất thông dụng ở nước ta.

1.2.1. Hình dáng kích thước.

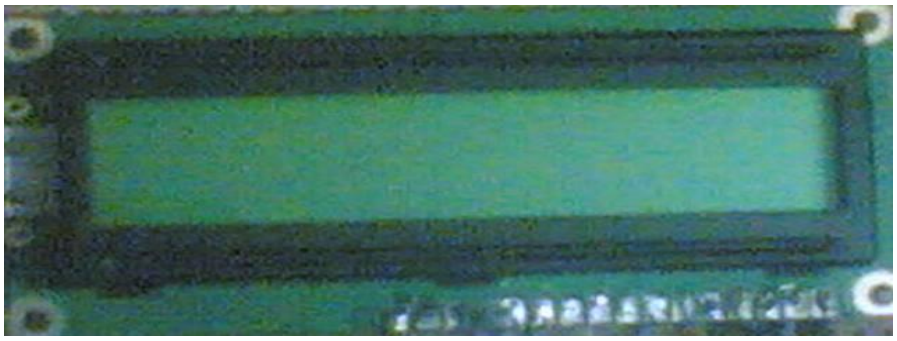
Có rất nhiều loại LCD với nhiều hình dáng và kích thước khác nhau, trên hình 1.19. là hai loại LCD thông dụng.



Hình 1.19. Hình hai loại LCD thông dụng.



Hình 1.20. Sơ đồ chân của LCD



Hình 1.21. LCD loại DM 1602A.

Khi sản xuất LCD, nhà sản xuất đã tích hợp chíp điều khiển (HD44780) bên trong lớp vỏ và chỉ đưa các chân giao tiếp cần thiết. Các chân này được đánh số thứ tự và đặt tên như hình 1.20.

1.2.2. Các chân chức năng.

Bảng 3.1. Các chân chức năng của HD44780.

Chân số	Tên	Chức năng
1	Vss	Chân nối đất cho LCD, khi thiết kế mạch ta nối chân này với GND của mạch điều khiển.
2	Vdd	Chân cấp nguồn cho LCD, khi thiết kế mạch ta nối chân này với 5V của mạch điều khiển.
3	Vo	Chân này dùng để điều chỉnh độ tương phản của LCD.
4	RS	Chân chọn thanh ghi (Register select). Nối chân RS với logic "0" (GND) hoặc logic "1" (Vcc) để chọn thanh ghi. + Logic "0": Bus DB0-DB7 sẽ nối với thanh ghi lệnh IR của LCD (ở chế độ "ghi" - write) hoặc nối với bộ đếm địa chỉ của LCD (ở chế độ "đọc" - read) + Logic "1": Bus DB0-DB7 sẽ nối với thanh ghi dữ liệu DR bên trong LCD.
5	RW	Chân chọn chế độ đọc/ghi (Read/Write). Nối chân R/W với logic "0" để LCD hoạt động ở chế độ ghi, hoặc nối với logic "1" để LCD ở chế độ đọc.
6	E	Chân cho phép (Enable). Sau khi các tín hiệu được đặt lên bus DB0-DB7, các lệnh chỉ được chấp nhận khi có 1 xung cho phép của chân E.

		+ Ở chế độ ghi: Dữ liệu ở bus sẽ được LCD chuyển vào (chấp nhận) thanh ghi bên trong nó khi phát hiện một xung (low-to-high transition) của tín hiệu chân E. + Ở chế độ đọc: Dữ liệu sẽ được LCD xuất ra DB0-DB7 khi phát hiện sườn lên (low-to-high transition) ở chân E và được LCD giữ ở bus đến khi nào chân E xuống mức thấp.
7÷14	DB0÷DB7	8 đường của bus dữ liệu dùng để trao đổi thông tin với MPU. Có 2 chế độ sử dụng 8 đường bus này: + Chế độ 8 bit: Dữ liệu được truyền trên cả 8 đường, với bit MSB là bit DB7. + Chế độ 4 bit: Dữ liệu được truyền trên 4 đường từ DB4 tới DB7, bit MSB là DB7.
15	A	15 là Catot, điện áp khoảng Uak=4,2V
16	K	Chân nối đất của đèn Back light

1.2.3. Sơ đồ khối của HD44780.

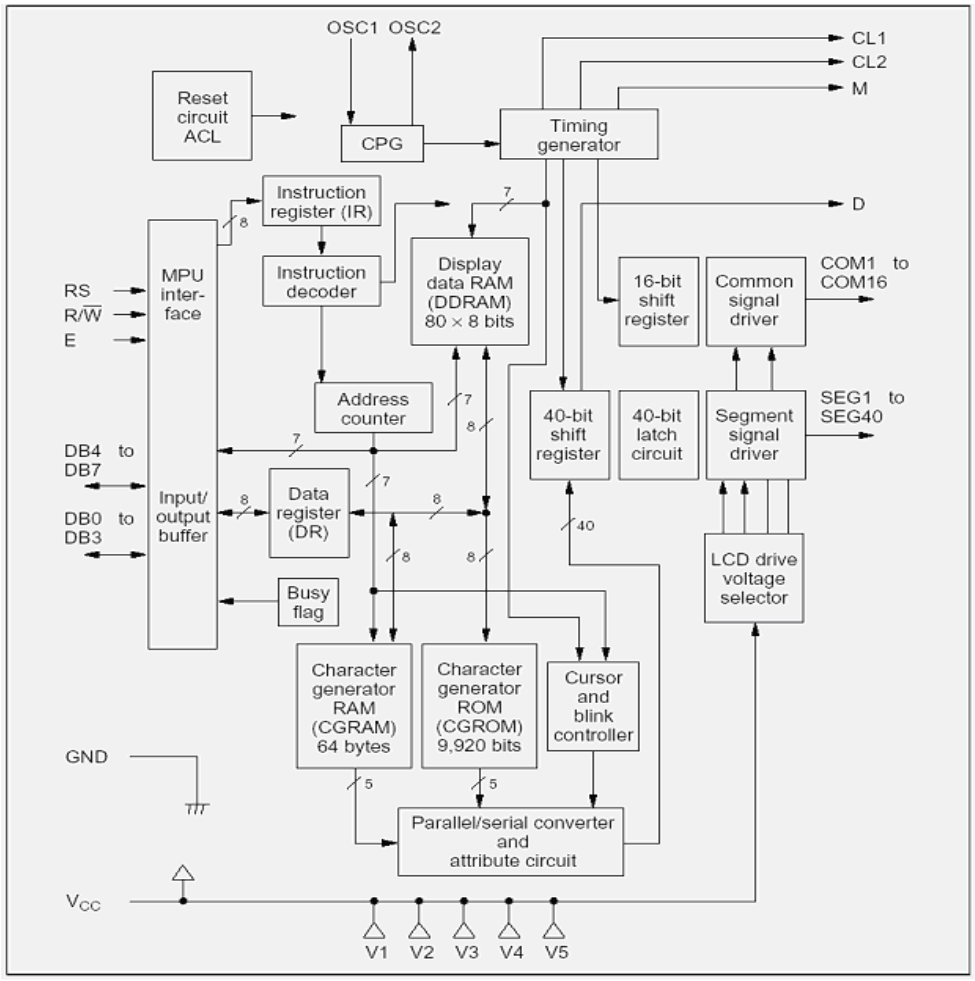
Để hiểu rõ hơn chức năng các chân và hoạt động của chúng, ta tìm hiểu sơ qua chíp HD44780 thông qua các khối cơ bản của nó.

*) Các thanh ghi:

Chíp HD44780 có 2 thanh ghi 8 bit quan trọng là: Thanh ghi lệnh IR (Instructor Register) và thanh ghi dữ liệu DR (Data Register).

- Thanh ghi IR: Để điều khiển LCD, người dùng phải "ra lệnh" thông qua tám đường bus DB0-DB7. Mỗi lệnh được nhà sản xuất LCD đánh địa chỉ rõ ràng. Người dùng chỉ việc cung cấp địa chỉ lệnh bằng cách nạp vào thanh ghi IR. Nghĩa là, khi ta nạp vào thanh ghi IR một chuỗi 8 bit, chíp HD44780 sẽ tra bảng mã lệnh tại địa chỉ mà IR cung cấp và thực hiện lệnh đó.

VD: Lệnh "hiển thị màn hình" có địa chỉ lệnh là 00001100 (DB7…DB0)



Hình 1.22. Sơ đồ khối của HD44780.

- Thanh ghi DR: Thanh ghi DR dùng để chứa dữ liệu 8 bit để ghi vào vùng RAM, DDRAM hoặc CGRAM (ở chế độ ghi) hoặc dùng để chứa dữ liệu từ 2 vùng RAM này gởi ra cho MPU (ở chế độ đọc). Nghĩa là, khi MPU ghi thông tin vào DR, mạch nội bên trong chíp sẽ tự động ghi thông tin này vào DDRAM hoặc CGRAM. Hoặc khi thông tin về địa chỉ được ghi vào IR, dữ liệu ở địa chỉ này trong vùng RAM nội của HD44780 sẽ được chuyển ra DR để truyền cho MPU. Vậy bằng cách điều khiển chân RS và R/W chúng ta có thể chuyển qua lại giữ 2 thanh ghi này trong khi giao tiếp với MPU. Bảng 3.2. tóm tắt lại các thiết lập đối với hai chân RS và R/W theo mục đích giao tiếp.

Bảng 3.2. Bảng chức năng chân RS và R/W theo mục đích sử dụng.

RS	RW	Ý nghĩa
0	0	Ghi vào thanh ghi IR để ra lệnh cho LCD (VD: cần display clear, …)
0	1	Đọc cờ bận ở DB7 và giá trị của bộ đếm địa chỉ ở DB0-DB6
1	0	Ghi vào thanh ghi DR
1	1	Đọc dữ liệu từ DR

*) Cờ báo bận BF (Busy Flag):

Khi thực hiện các hoạt động bên trong chíp, mạch nội bên trong cần một khoảng thời gian để hoàn tất. Khi đang thực thi các hoạt động bên trong chíp như thế, LCD bỏ qua mọi giao tiếp với bên ngoài và bật cờ BF (thông qua chân DB7 khi có thiết lập RS=0, R/W=1) lên để báo cho MPU biết nó đang "bận". Dĩ nhiên, khi xong việc, nó sẽ đặt cờ BF lại mức 0.

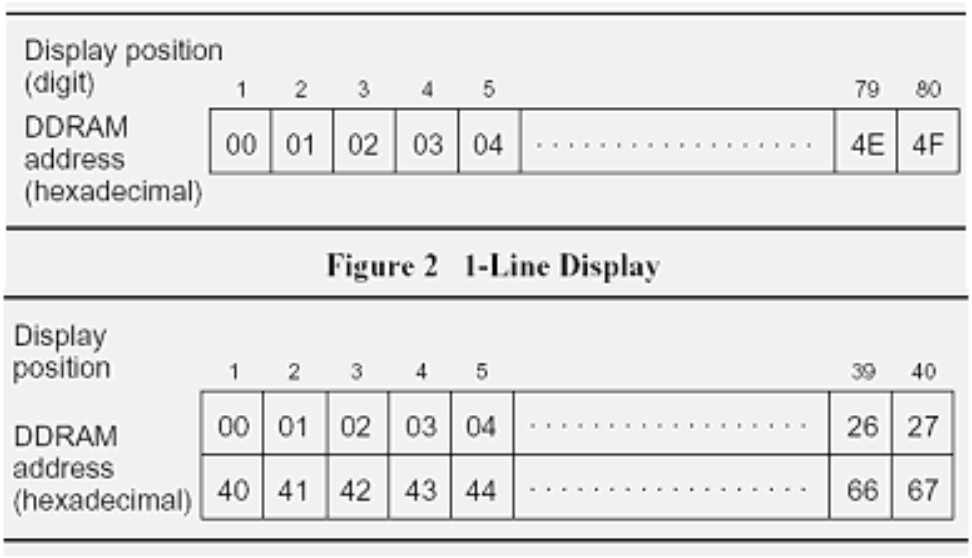
*) Bộ đếm địa chỉ AC (Address Counter):

Như trong sơ đồ khối, thanh ghi IR không trực tiếp kết nối với vùng RAM (DDRAM và CGRAM) mà thông qua bộ đếm địa chỉ AC. Bộ đếm này lại nối với 2 vùng RAM theo kiểu rẽ nhánh. Khi một địa chỉ lệnh được nạp vào thanh ghi IR, thông tin được nối trực tiếp cho 2 vùng RAM nhưng việc chọn lựa vùng RAM tương tác đã được bao hàm trong mã lệnh. Sau khi ghi vào (hoặc đọc từ) RAM, bộ đếm AC tự động tăng lên (hoặc giảm đi) 1 đơn vị và nội dung của AC được xuất ra cho MPU thông qua DB0-DB6 khi có thiết lập RS=0 và R/W=1 (xem bảng 3.2). Lưu ý: Thời gian cập nhật AC không được tính vào thời gian thực thi lệnh mà được cập nhật sau khi cờ BF lên mức cao (not busy), cho nên khi lập trình hiển thị, bạn phải delay một khoảng tADD khoảng 4µS-5µS (ngay sau khi BF=1) trước khi nạp dữ liệu mới.

*) Vùng RAM hiển thị DDRAM (Display Data RAM):

Đây là vùng RAM dùng để hiển thị, nghĩa là ứng với một địa chỉ của RAM là một ô kí tự trên màn hình và khi bạn ghi vào vùng RAM này một mã 8 bit,

LCD sẽ hiển thị tại vị trí tương ứng trên màn hình một kí tự có mã 8 bit mà bạn đã cung cấp như hình 1.23.



Hình 1.23. Mối liên hệ giữa địa chỉ của DDRAM và vị trí hiển thị của LCD.

Vùng RAM này có 80x8 bits nhớ, nghĩa là chứa được 80 kí tự mã 8 bits. Những vùng RAM còn lại không dùng cho hiển thị có thể dùng như vùng RAM đa mục đích. Lưu ý là để truy cập vào DDRAM, ta phải cung cấp địa chỉ cho AC theo mã HEX.

*) Vùng ROM chứa kí tự CGROM (Character Generator ROM):

Vùng ROM này dùng để chứa các mẫu kí tự loại 5x8 hoặc 5x10 điểm ảnh/kí tự, và định địa chỉ bằng 8 bit. Tuy nhiên, nó chỉ có 208 mẫu kí tự 5x8 và 32 mẫu kí tự kiểu 5x10 (tổng cộng là 240 thay vì 256 mẫu kí tự). Người dùng không thể thay đổi vùng ROM này.

Table 2 Example of Correspondence between EPROM Address Data and Character Pattern (5 × 8 Dots)

EPROM Address												Data					
A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	O4	O3	O2	O1	LSB O0	
								0	0	0	0	1	0	0	0	0	
								0	0	0	1	1	0	0	0	0	
								0	0	1	0	1	0	1	1	0	
								0	0	1	1	1	1	0	0	1	
								0	1	0	0	1	0	0	0	1	
								0	1	0	1	1	0	0	0	1	
								0	1	1	0	1	1	1	1	0	
0	1	1	0	0	0	1	0	0	1	1	1	0	0	0	0	0	← Cursor position
								1	0	0	0	0	0	0	0	0	
								1	0	0	1	0	0	0	0	0	
								1	0	1	0	0	0	0	0	0	
								1	0	1	1	0	0	0	0	0	
								1	1	0	0	0	0	0	0	0	
								1	1	0	1	0	0	0	0	0	
								1	1	1	0	0	0	0	0	0	
								1	1	1	1	0	0	0	0	0	

Hình 1.24. Mối liên hệ giữa địa chỉ của ROM và dữ liệu tạo mẫu kí tự.

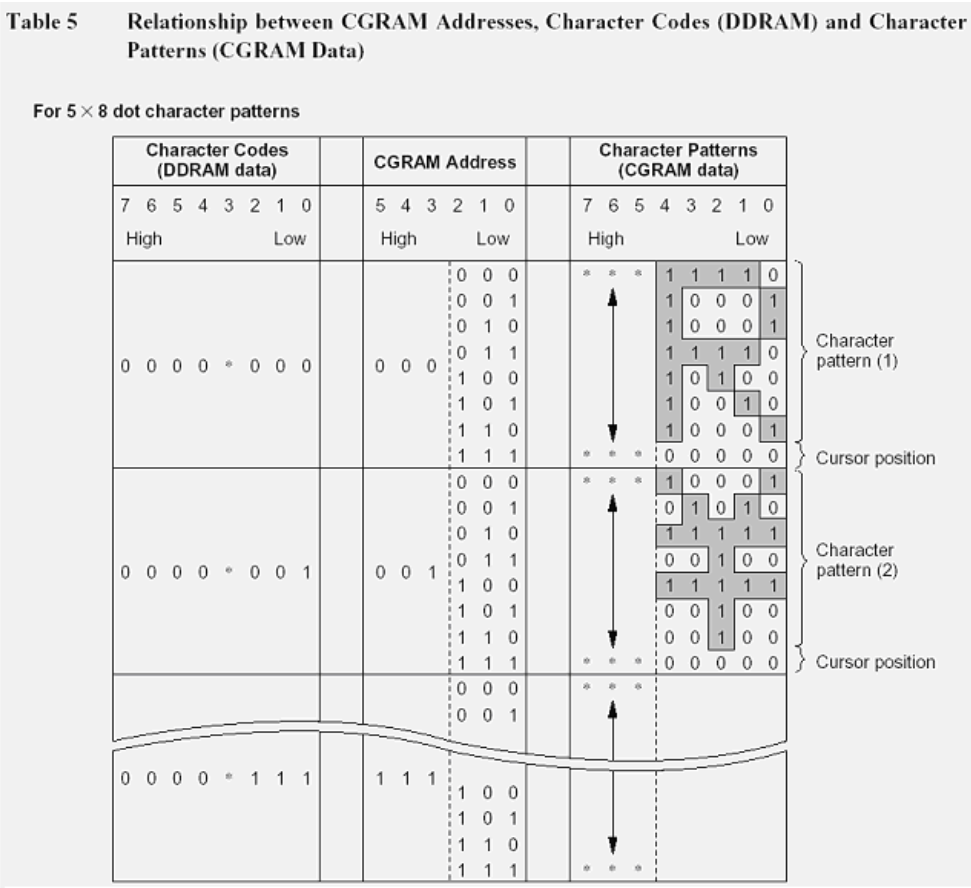
*) Vùng RAM chứa kí tự đồ họa CGRAM (Character Generator RAM):

Như trên bảng mã kí tự, nhà sản xuất dành vùng có địa chỉ byte cao là 0000h để người dùng có thể tạo các mẫu kí tự đồ họa riêng. Tuy nhiên dung lượng vùng này rất hạn chế: Ta chỉ có thể tạo 8 kí tự loại 5x8 điểm ảnh, hoặc 4 kí tự loại 5x10 điểm ảnh. Để ghi vào CGRAM, xem hình 1.24.

1.2.4. Tập lệnh của LCD.

Trước khi tìm hiểu tập lệnh của LCD, sau đây là một vài chú ý khi giao tiếp với LCD:

* Tuy trong sơ đồ khối của LCD có nhiều khối khác nhau, nhưng khi lập trình điều khiển LCD ta chỉ có thể tác động trực tiếp được vào 2 thanh ghi DR và IR thông qua các chân DBx, và ta phải thiết lập chân RS, R/W phù hợp để chuyển qua lại giữ 2 thanh ghi này. (xem bảng 3.2)



Hình 1.25. Mối liên hệ giữa địa chỉ của CGRAM, dữ liệu CGARM, và mã kí tự.

* Với mỗi lệnh, LCD cần một khoảng thời gian để hoàn tất, thời gian này có thể khá lâu đối với tốc độ của MPU, nên ta cần kiểm tra cờ BF hoặc đợi (delay) cho LCD thực thi xong lệnh hiện hành mới có thể ra lệnh tiếp theo.

* Địa chỉ của RAM (AC) sẽ tự động tăng (giảm) 1 đơn vị, mỗi khi có lệnh ghi vào RAM. (Điều này giúp chương trình gọn hơn)

* Các lệnh của LCD có thể chia thành 4 nhóm như sau:

- Các lệnh về kiểu hiển thị. VD : Kiểu hiển thị (1 hàng/2 hàng), chiều dài dữ liệu (8 bit/4 bit), …
- Chỉ định địa chỉ RAM nội.
- Nhóm lệnh truyền dữ liệu trong RAM nội.
- Các lệnh còn lại .

Bảng 3.3. Tập lệnh của LCD.

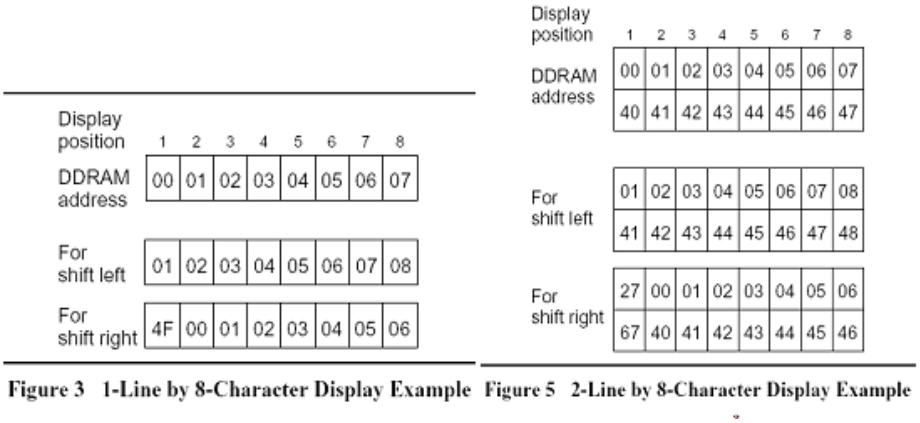
<table>
<tr><th>Tên lệnh</th><th>Hoạt động</th><th>Thời gian chạy</th></tr>
<tr><td>Clear Display</td><td>Mã lệnh: DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
DBx = 0 0 0 0 0 0 0 1
Lệnh Clear Display (xóa hiển thị) sẽ ghi một khoảng trống (mã hiển thị kí tự 20H) vào tất cả ô nhớ trong DDRAM, sau đó trả bộ đếm địa chỉ AC=0, trả lại hiển thị gốc nếu nó bị thay đổi, nghĩa là: Tắt hiển thị, con trỏ dời về góc trái (hàng đầu tiên), chế độ tăng AC.</td><td></td></tr>
<tr><td>Return home</td><td>Mã lệnh: DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
DBx = 0 0 0 0 0 0 1 *
Lệnh Return home trả bộ đếm địa chỉ AC về 0, trả lại kiểu hiển thị gốc nếu nó bị thay đổi. Nội dung của DDRAM không thay đổi.</td><td>1.52 ms</td></tr>
<tr><td>Entry mode set</td><td>Mã lệnh: DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
DBx = 0 0 0 0 0 1 [I/D] [S]
I/D: Tăng (I/D=1) hoặc giảm (I/D=0) bộ đếm địa chỉ hiển thị AC 1 đơn vị mỗi khi có hành động ghi hoặc đọc vùng DDRAM. Vị trí con trỏ cũng di chuyển theo sự tăng giảm này.
S: Khi S=1 toàn bộ nội dung hiển thị bị dịch sang phải (I/D=0) hoặc sang trái (I/D=1) mỗi khi có hành động ghi vùng DDRAM. Khi S=0: không dịch nội dung hiển thị. Nội dung hiển thị không dịch khi đọc DDRAM hoặc đọc/ghi vùng CGRAM.
 

Figure 3 1-Line by 8-Character Display Example Figure 5 2-Line by 8-Character Display Example
Hình 3.7. Hoạt động dịch trái và dịch phải nội dung hiển thị</td><td>37µs</td></tr>
<tr><td>Display on/off</td><td>Mã lệnh: DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
DBx = 0 0 0 0 1 [D] [C] [B]</td><td></td></tr>
</table>

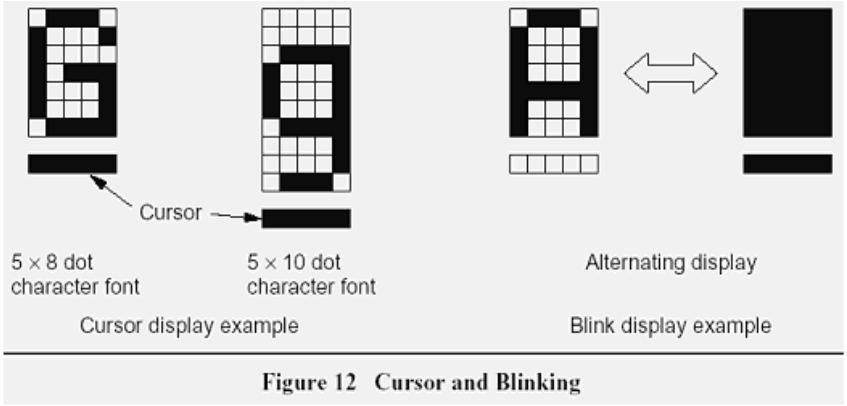
<table>
<tr>
<td>control</td>
<td>D: Hiển thị màn hình khi D=1 và ngược lại. Khi tắt hiển thị, nội dung DDRAM không thay đổi.
C: Hiển thị con trỏ khi C=1 và ngược lại. Vị trí và hình dạng con trỏ, xem hình 3.8.
B: Nhấp nháy kí tự tại vị trí con trỏ khi B=1 và ngược lại. Xem thêm hình 8. về kiểu nhấp nháy. Chu kì nhấp nháy khoảng 409,6ms khi mạch dao động nội LCD là 250kHz.
 

Figure 12 Cursor and Blinking
Hình 3.8. Kiểu con, kiểu kí tự và nhấp nháy kí tự</td>
<td>37µs</td>
</tr>
<tr>
<td>Cursor or display shift</td>
<td>Mã lệnh: DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
DBx = 0 0 0 1 [S/C] [R/L] * *
Lệnh Cursor or display shift dịch chuyển con trỏ hay dữ liệu hiển thị sang trái mà không cần hành động ghi/đọc dữ liệu. Khi hiển thị kiểu 2 dòng, con trỏ sẽ nhảy xuống dòng dưới khi dịch qua vị trí thứ 40 của hàng đầu tiên. Dữ liệu hàng đầu và hàng 2 dịch cùng một lúc. Chi tiết sử dụng xem bảng sau:
<table>
<tr><th>S/C</th><th>R/L</th><th>Hoạt động</th></tr>
<tr><td>0</td><td>0</td><td>Dịch vị trí con trỏ sang trái (Nghĩa là giảm AC một đơn vị).</td></tr>
<tr><td>0</td><td>1</td><td>Dịch vị trí con trỏ sang phải (Tăng AC lên 1 đơn vị).</td></tr>
<tr><td>1</td><td>0</td><td>Dịch toàn bộ nội dung hiển thị sang trái, con trỏ cũng dịch theo.</td></tr>
<tr><td>1</td><td>1</td><td>Dịch toàn bộ nội dung hiển thị sang phải, con trỏ cũng dịch theo.</td></tr>
</table></td>
<td>37µs</td>
</tr>
<tr>
<td>Function set</td>
<td>Mã lệnh: DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
DBx = 0 0 1 [DL] [N] [F] * *
DL: Khi DL=1, LCD giao tiếp với MPU bằng giao thức 8 bit (từ bit DB7 đến DB0). Ngược lại, giao thức giao tiếp là 4 bit (từ bit DB7 đến bit DB0). Khi chọn giao thức 4 bit, dữ liệu được truyền/nhận 2 lần liên tiếp</td>
<td></td>
</tr>
</table>

	với 4 bit cao gửi/nhận trước, 4 bit thấp gửi/nhận sau. N: Thiết lập số hàng hiển thị. Khi N=0: hiển thị 1 hàng, N=1: hiển thị 2 hàng. F: Thiết lập kiểu kí tự. Khi F=0: kiểu kí tự 5x8 điểm ảnh, F=1: kiểu kí tự 5x10 điểm ảnh. * Chú ý: • Chỉ thực hiện thay đổi Function set ở đầu chương trình. Và sau khi được thực thi 1 lần, lệnh thay đổi Function set không được LCD chấp nhận nữa ngoại trừ thiết lập chuyển đổi giao thức giao tiếp. • Không thể hiển thị kiểu kí tự 5x10 điểm ảnh ở kiểu hiển thị 2 hàng.	37µs
Set CGRAM address	Mã lệnh: DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx= 0 1 [ACG][ACG][ACG][ACG][ACG][ACG] Lệnh này ghi vào AC địa chỉ của CGRAM. Kí hiệu [ACG] chỉ 1 bit của chuỗi dữ liệu 6 bit. Ngay sau lệnh này là lệnh đọc/ghi dữ liệu từ CGRAM tại địa chỉ đã được chỉ định.	37µs
Set DDRAM address	Mã lệnh: DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = 1 [AD] [AD] [AD] [AD] [AD] [AD] [AD] Lệnh này ghi vào AC địa chỉ của DDRAM, dùng khi cần thiết lập tọa độ hiển thị mong muốn. Ngay sau lệnh này là lệnh đọc/ghi dữ liệu từ DDRAM tại địa chỉ đã được chỉ định. Khi ở chế độ hiển thị 1 hàng, địa chỉ có thể từ 00H đến 4FH. Khi ở chế độ hiển thị 2 hàng, địa chỉ từ 00h đến 27H cho hàng thứ nhất, và từ 40h đến 67h cho hàng thứ 2.	37µs
Read BF and address	Mã lệnh: DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx= [BF] [AC] [AC] [AC] [AC] [AC] [AC] [AC] (RS=0, R/W=1) Như đã đề cập trước đây, khi cờ BF bật, LCD đang làm việc và lệnh tiếp theo (nếu có) sẽ bị bỏ qua nếu cờ BF chưa về mức thấp. Cho nên, khi lập trình điều khiển, bạn phải kiểm tra cờ BF trước khi ghi dữ liệu vào LCD. Khi đọc cờ BF, giá trị của AC cũng được xuất ra các bit [AC]. Nó là địa chỉ của CG hay DDRAM là tùy thuộc vào lệnh trước đó.	0µs
Write ata to CG or DDRAM	Mã lệnh: DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = [Write data] (RS=1, R/W=0) Khi thiết lập RS=1, R/W=0, dữ liệu cần ghi được đưa vào các chân DBx từ mạch ngoài sẽ được LCD chuyển vào trong LCD tại địa chỉ được xác định từ lệnh ghi địa chỉ trước đó (lệnh ghi địa chỉ cũng xác định luôn	37µs tADD

	vùng RAM cần ghi). Sau khi ghi, bộ đếm địa chỉ AC tự động tăng/giảm 1 tùy theo thiết lập Entry mode. Lưu ý là thời gian cập nhật AC không tính vào thời gian thực thi lệnh.	4µs
Read data from CG or DDRAM	Mã lệnh: DBx = DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0 DBx = [Read data] (RS=1, R/W=1) Khi thiết lập RS=1, R/W=1,dữ liệu từ CG/DDRAM được chuyển ra MPU thông qua các chân DBx (địa chỉ và vùng RAM đã được xác định bằng lệnh ghi địa chỉ trước đó). Sau khi đọc, AC tự động tăng/giảm 1 tùy theo thiết lập Entry mode, tuy nhiên nội dung hiển thị không bị dịch bất chấp chế độ Entry mode.	37µs tADD 4µs

1.2.5. Đặc tính của các chân giao tiếp.

LCD sẽ bị hỏng nghiêm trọng, hoặc hoạt động sai lệch nếu bạn vi phạm khoảng đặc tính điện sau đây:

Bảng 3.4. Đặc tính điện làm việc điển hình.

Chân cấp nguồn (Vcc-GND)	Min:-0.3V , Max+7V
Các chân ngõ vào (DBx,E,…)	Min:-0.3V , Max:(Vcc+0.3V)
Nhiệt độ hoạt động	Min:-30C , Max:+75C
Nhiệt độ bảo quản	Min:-55C , Max:+125C

Đặc tính điện làm việc điển hình: (Đo trong điều kiện hoạt động Vcc = 4.5V đến 5.5V, T = -30 đến +75C).

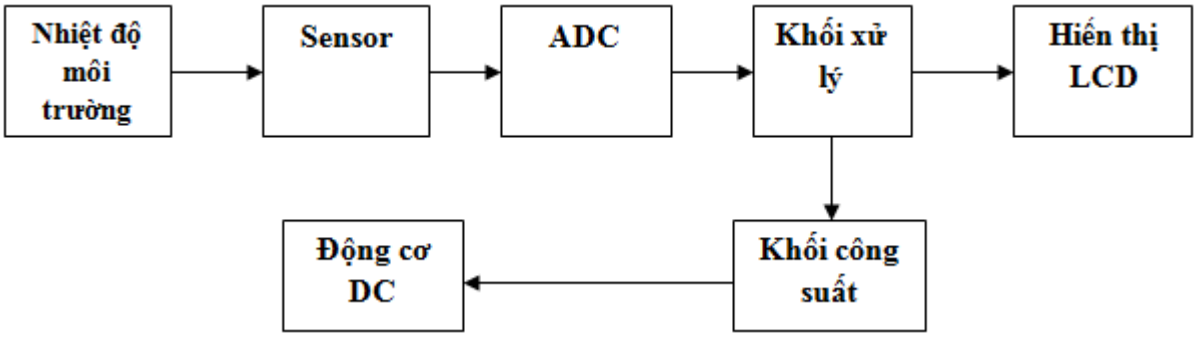
Bảng 3.5. Miền làm việc bình thường.

Chân cấp nguồn Vcc-GND	2.7V đến 5.5V
Điện áp vào mức cao V_{IH}	2.2V đến Vcc
Điện áp vào mức thấp V_{IL}	-0.3V đến 0.6V
Điện áp ra mức cao (DB0-DB7)	Min 2.4V (khi I_{OH} = -0.205mA)
Điện áp ra mức thấp (DB0-DB7)	Max 0.4V (khi I_{OL} = 1.2mA)
Dòng điện ngõ vào (input leakage current) I_{LI}	-1uA đến 1uA (khi V_{IN} = 0 đến Vcc)
Dòng điện cấp nguồn I_{CC}	350uA(typ.) đến 600uA
Tần số dao động nội f_{OSC}	190kHz đến 350kHz (điển hình là 270kHz)

Chương 2. THIẾT KẾ HỆ THỐNG ĐIỀU KHIỂN ĐỘNG CƠ DC BẰNG NHIỆT ĐỘ

2.1. Sơ đồ khối

Với yêu cầu của đề tài là thiết kế hệ thống điều khiển động cơ DC theo nhiệt độ, tức là từ nhiệt độ đo được từ môi trường, hệ thống điều khiển tốc độ động cơ DC quay nhanh hay chậm, ta có sơ đồ khối hệ thống trong hình 2.1.



Hình 2.1. Sơ đồ khối hệ thống điều khiển động cơ DC bằng nhiệt độ

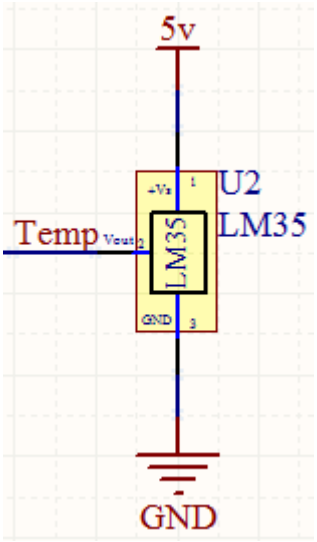
Với sơ đồ này ta sử dụng cảm biến nhiệt độ Sensor đo nhiệt độ môi trường. Bộ xử lý làm nhiệm vụ nhận, chuyển đổi ADC, từ đó điều khiển động cơ DC quay với tốc độ phù hợp. Trên sơ đồ sử dụng khối hiển thị để người sử dụng có thể theo dõi được các thông số và thao tác thực hiện.

2.2. Thiết kế các khối

2.2.1. Mạch đo nhiệt độ.

Nhiệt độ là một đại lượng vật lý vô hướng. Để đo đạc và tính toán giá trị của nó ta phải dùng các bộ cảm biến. Mạch đo nhiệt độ dùng các loại bộ cảm biến LM35. Các bộ cảm biến LM35 là bộ cảm biến nhiệt, mạch tích hợp chính xác cao mà điện áp đầu ra của nó tỷ lệ tuyến tính với nhiệt độ theo thang độ Celsius(*).

Bộ cảm biến LM35 cũng không yêu cầu cân chỉnh ngoài vì vốn chúng đã được cân chỉnh. Chúng đưa ra điện áp 10mV cho mỗi sự thay đổi 1^0C.

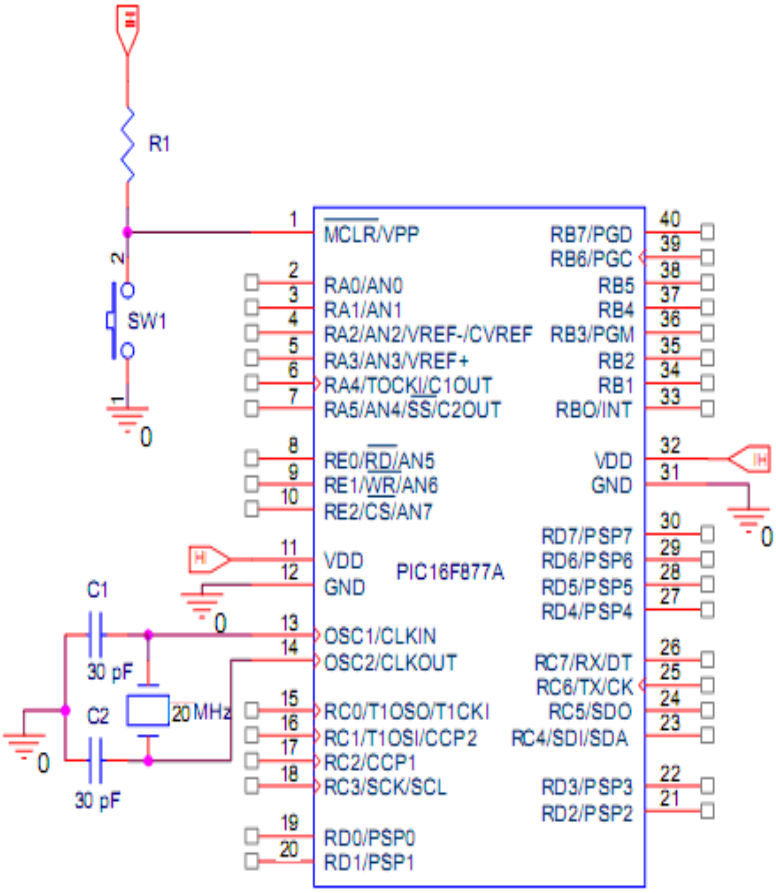


Hình 2.2 Cảm biến LM35

Như vậy chỉ cần 1 bộ cảm biến LM35 ta có thể tính được giá trị nhiệt độ tại thời điểm xác định dựa vào giá trị điện áp đầu ra LM35. Như đã nói ở trên, ứng với mỗi thay đổi 1^0C, giá trị đầu ra sẽ tăng thêm 10mV. Do đo, qua một bộ chuyển đổi tín hiệu tương tự (điện áp) sang tín hiệu số (bit) để hiển thị kết quả đo đạc và tính toán, xử lý kết quả.

2.2.2. Khối xử lý

Đây là khối quan trọng và quản lý toàn bộ hoạt động của mạch. Nhiệm vụ chính của khối là nhận về giá trị nhiệt độ, điều khiển động cơ. Giá trị đo từ LM35 được cho qua bộ ADC, mạch này ta có thể dùng IC ngoài hoặc được tích hợp trong một số dòng vi điều khiển. Ở đây chúng ta dùng PIC16F877A, vừa chuyển đổi ADC, hiển thị LCD, vừa điều khiển động cơ bằng phương pháp PWM. PIC16F877A là vi điều khiển có 40 chân, với 5 cổng vào ra là Port A (RA0÷RA5), Port B (RB0÷RB7), Port C (RC0÷RC7), Port D (RD0÷RD7), Port E (RE0÷RE2). Nó có 8K Flash ROM và 368 Bytes RAM. Sơ đồ sử dụng PIC16F877A như trong hình 2.3, Chân 1 được đấu để có thể RESET chương trình. Xung Clock dùng dao đông thạch anh 20MHz, đấu vào chân 13, 14.



Hình 2.3. Sơ đồ nguyên lý của PIC16F877A trong mạch

2.2.3. Chức năng ADC trong PIC16F887.

Trong PIC 16F887 có hỗ trợ bộ chuyển đổi ADC 10 bit 8 kênh. PIC16F887 có 8 ngõ vào Analog (RA4:RA0) và (RE2:RE0).

Kết quả chuyển đổi từ tín hiệu tương tự sang tín hiệu số là 10 bít tương ứng và được lưu trong thanh ghi ADRESH:ADRESL. Khi không sử dụng bộ chuyển đổi ADC, các thanh ghi này có thể được sử dụng như các thanh ghi thông thường. Khi quá trình chuyển đôi hoàn tất, kết quả sẽ được lưu vào hai thanh ghi ADRESH:ADRESL. Cờ ngắt ADIF được set.

Quá trình chuyển đổi tương tự sang số bao gồm các bước sau:

1. Thiết lập các thông số cho bộ chuyển đổi ADC

Chọn ngõ vào alalog, chọn điện áp mẫu.

Chọn kênh chuyển đổi AD (Thanh ghi ADCON0)

Chọn xung Clock cho kênh chuyển đổi AD.

Cho phép bộ chuyển đổi AD hoạt động.

2. Thiết lập các cờ ngắt cho bộ AD.

   Clear bit ADIF

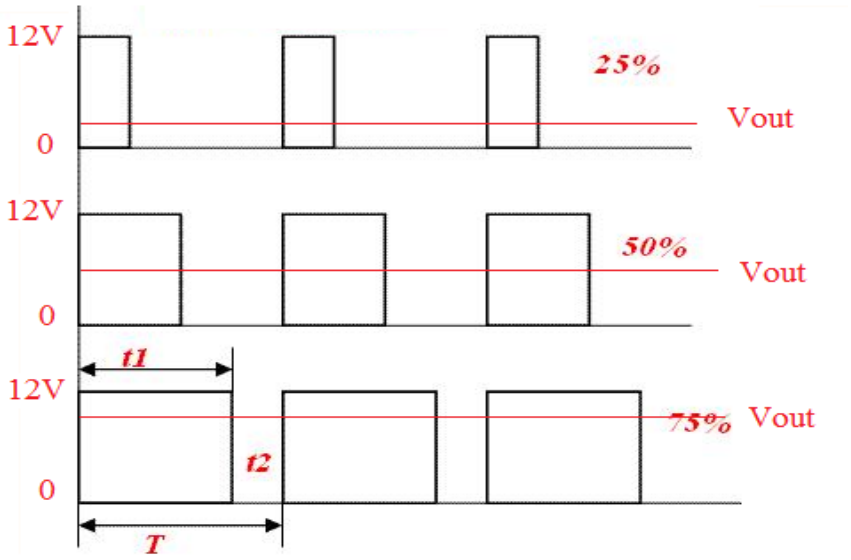
   Set bit ADIE

   Set bit PEIE

   Set bit GIE
3. Đợi cho tới khi quá trình lấy mẫu hoàn tất.
4. Bắt đầu quá trình chuyển đổi.
5. Đợi cho tới khi quá trình chuyển đổi hoàn tất.
6. Đọc kết quả chuyển đổi và xóa cờ ngắt, set bit.
7. Tiếp tục thực hiện các bước 1 và 2 cho quá trình chuyển đổi tiếp theo

2.2.3. Khối công suất.

Ở đây sử dụng phương pháp PWM (Pulse Width Modulation) để điều khiển tốc độ của động cơ DC. Phương pháp điều chế PWM 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. Các xung PWM khi biến đổi thì có cùng 1 tần số và khác nhau về độ rộng của sườn dương hay hoặc là sườn âm. Điều khiển động cơ sử dụng phương thức điều chế xung PWM là một trong các phương thức được sử dụng rất rộng rãi trong điều khiển động cơ ứng dụng trong công nghiệp, dân dụng cũng như trong nhiều ứng dụng khác, ngoài ra PWM còn tham gia và điều chế các mạch nguồn như là: boot, buck, nghịch lưu 1 pha và 3 pha ... Điều đặc biệt là PWM chuyên dùng để điều khiển các phần tử điện tử công suất có đường đặc tính là tuyến tính khi có sẵn 1 nguồn 1 chiều cố định. Như vậy PWM được ứng dụng rất nhiều trong các thiết bị điện, điện tử.



Hình 2.4. Xung PWM và điện áp đầu ra

Hình 2.4. là sơ đồ đặc tả xung PWM và cách thức tính điện áp đầu ra đưa tới động cơ. Nhìn vào sơ đồ ta có

- Chu kỳ của xung PWM là thời gian T.
- Thời gian phát xung PWM là t_1
- Thời gian nghỉ không phát xung là t_2

Công thức tính giá trị trung bình của điện áp ra tải :

Vout = Vin*duty

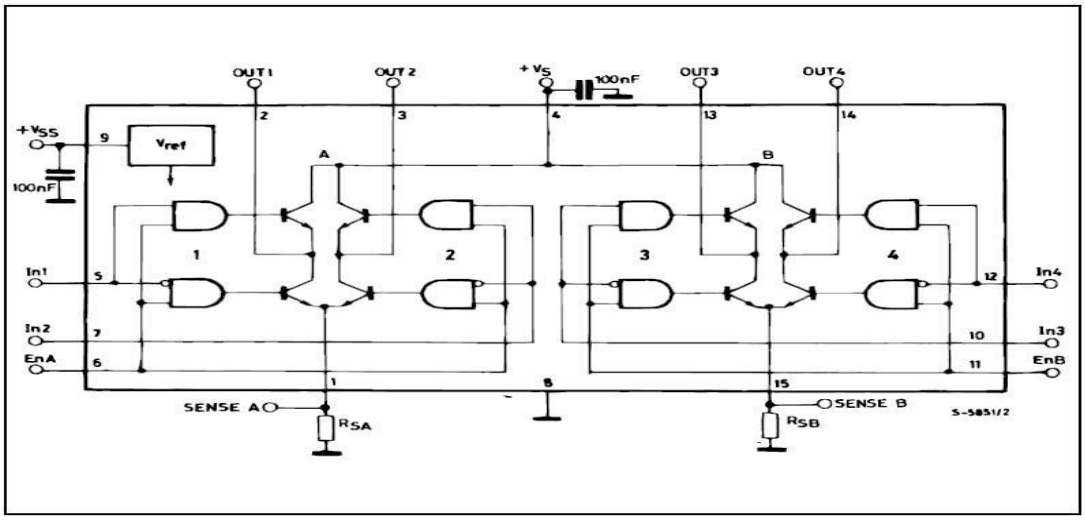
Trong đó:

- Vout là điện áp ra
- Vin là điện áp đầu vào
- Duty là % thời gian phát xung được tính bằng Duty = t_1/T*100%

Trong khối điều khiển, PIC16F887A điều khiển động cơ thông qua quá trình tạo xung PWM rồi đưa vào IC Driver L298D tạo nguồn nuôi động cơ. Bản chất của IC Driver L298D là hai bộ mạch cầu H được tích hợp trong cùng IC.

Mạch cầu H là một trong những mạch được sử dụng rộng rãi cho việc điều khiển động cơ.

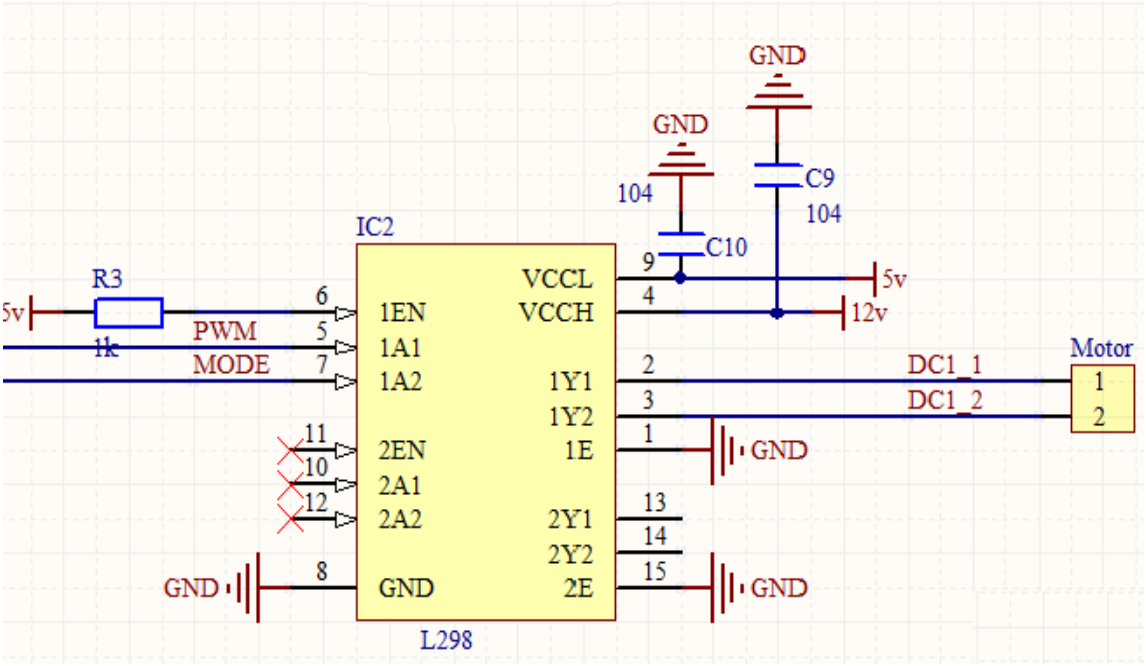
L298D là một chip tích hợp 2 mạch cầu H trong gói 15 chân. Tất cả các mạch kích, mạch cầu đều được tích hợp sẵn. L298D có điện áp danh nghĩa cao (lớn nhất 50V) và dòng điện danh nghĩa lớn hơn 2A nên rất thích hợp cho các các ứng dụng công suất nhỏ như các động cơ DC loại nhỏ và vừa.



Hình 2.5. Sơ đồ khối bên trong IC Driver L298D.

Hình 2.5. là sơ đồ khối bên trong Driver L298D. Có 2 mạch cầu H trên mỗi chip L298D nên có thể điều khiển 2 đối tượng chỉ với 1 chip. 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. 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 như quá tải). Nếu việc đo dòng động cơ không thật sự cần thiết ta có thể nối đường current sensing này 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 đông cơ 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 morto.

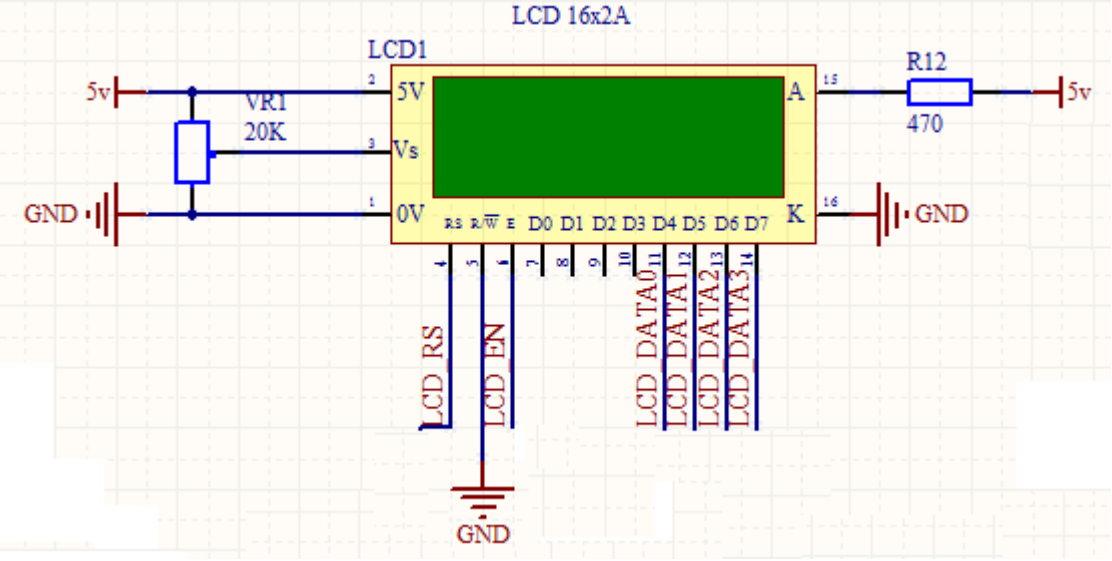


Hình 2.6. Sơ đồ nối chân L298 trong mạch

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).

2.2.4. Khối hiển thị

Để thuận tiện cho việc hiển thị kí tự và chế độ cài đặt trạng thái điều khiển, ở em đây sử dụng LCD_DM 16x2A.



Hình 2.7. Sơ đồ nguyên lý của LCD16x2A

LCD16x2A là loại 2 dòng, 16 kí tự, sử dụng nguồn nuôi thấp (từ 2,5 đến 5V). Có thể hoạt động ở hai chế độ 4 bit hoặc 8 bit (trong đề tài này em sử dụng chế độ 4 bit).

2.2.5. Motor DC

Cấu tạo và nguyên lý làm việc.

Cấu tạo của động cơ gồm có 2 phần: Stato đứng yên và rôto quay so với stato. Phần cảm tạo ra từ trường đi trong mạch từ, xuyên qua các vòng dây quấn của phần ứng. Khi có dòng điện chạy trong mạch phần ứng, các thanh dẫn phần ứng sẽ chịu tác động bởi các lực điện từ theo phương tiếp tuyến với mặt trụ rôto, làm cho rôto quay.

Hình 2.8. Động cơ DC

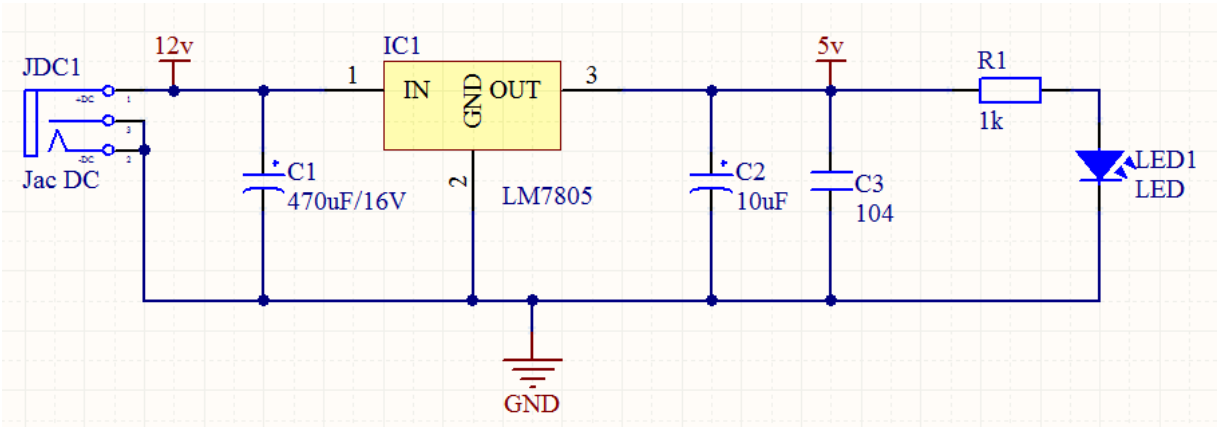
Dòng điện I đi qua mạch phần ứng của động cơ được biểu diễn bằng phương trình sau:

$$I = \frac{U - E}{R_a}$$

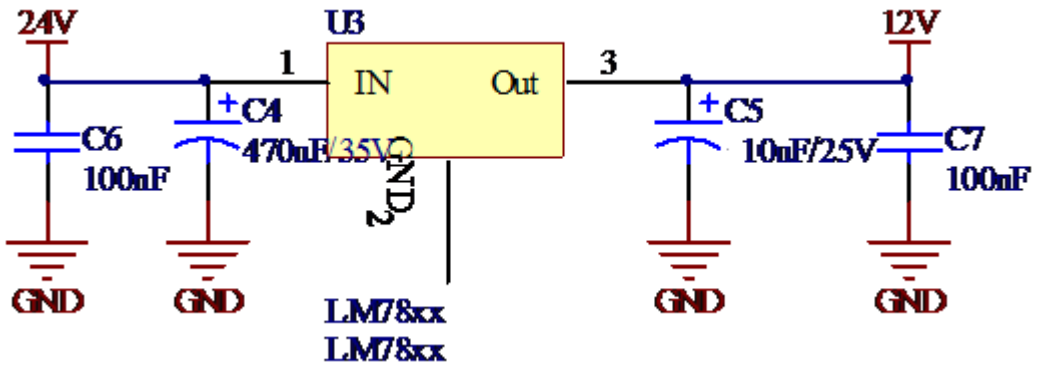
U là điện áp đặt vào mạch phần ứng, R_a là điện trở mạch phần ứng, và E là sức điện động phần ứng.

2.2.6. Khối nguồn

Cung cấp nguồn nuôi cho toàn bộ hệ thống. Sơ đồ khối trong hình 2.6.



Hình 2.9. Mạch biến áp 5V cấp nguồn cho vi điều khiển sử dụng IC LM7805

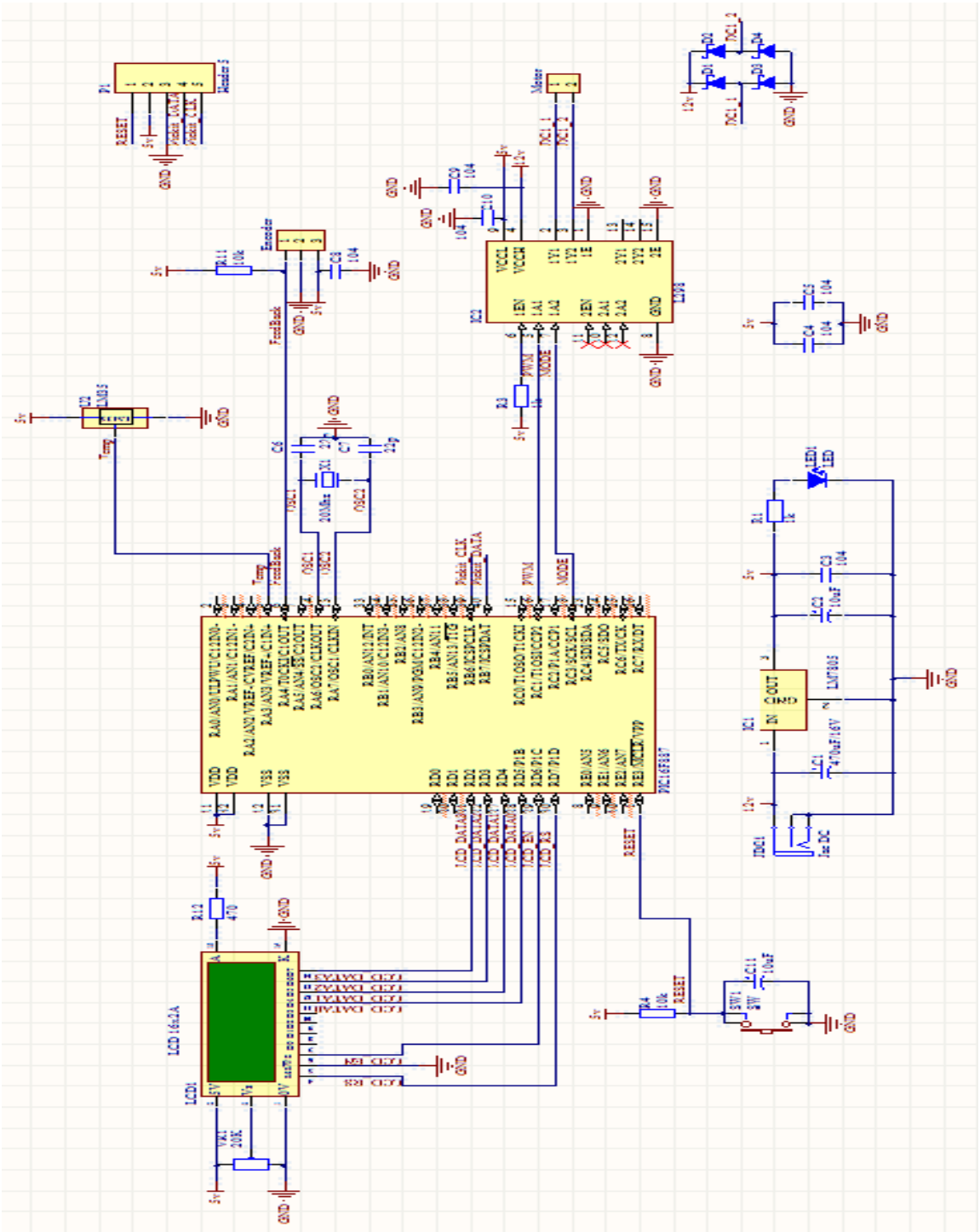


Hình 2.10. Mạch biến áp 12V cấp nguồn động cơ dùng IC LM7812

Ở đây bộ ổn áp dùng IC 7805, 7812 để tạo nguồn +5V cung cấp nguồn cho mạch vi điều khiển và IC 7812 và nguồn +12V cung cấp cho động cơ.

2.3. Sơ đồ mạch nguyên lý hệ thống

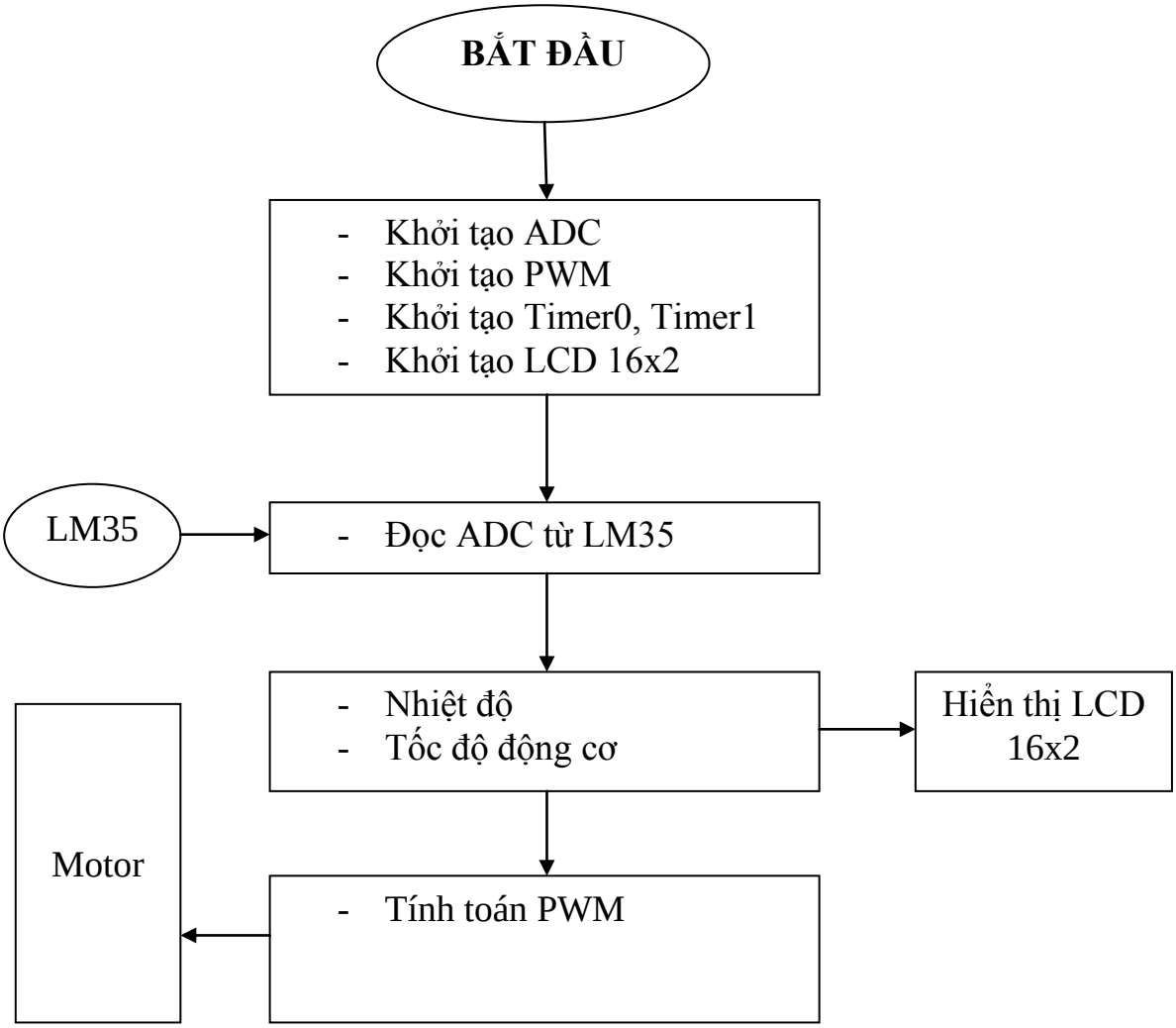
Sơ đồ mạch nguyên lý của hệ thống như trong hình 2.11.



Hình 2.11. Sơ đồ mạch nguyên lý hệ thống điều khiển động cơ DC bằng nhiệt độ

Chương 3. CHƯƠNG TRÌNH ĐIỀU KHIỂN

3.1. Lưu đồ thuật toán



3.2. Chương trình điều khiển

```
// Khai báo LCD
#ifndef _LCD_H_
#define _LCD_H_
#include <htc.h>
#include "timer.h"


#define LCD_RS RD7
//#define   LCD_RW RA2
#define LCD_EN RD6
#define LCD_D4 RD5
#define LCD_D5 RD4
#define LCD_D6 RD3
#define LCD_D7 RD2
#define LCD_PORT TRISD

void lcd_numb2(unsigned char number);
void lcd_numb4(unsigned int number);
/* write a byte to the LCD in 4 bit mode */

extern void lcd_write(unsigned char);

/* Clear and home the LCD */

extern void lcd_clear(void);

/* write a string of characters to the LCD */

extern void lcd_puts(const char * s);

/* Go to the specified position */

extern void lcd_goto(unsigned char colum, unsigned char row);

/* intialize the LCD - call before anything else */

extern void lcd_init(void);
```

```
extern void lcd_putch(char);

/*  Set the cursor position */

#define lcd_cursor(x)   lcd_write(((x)&0x7F)|0x80)

/* Display ON/OFF Control defines */
#define DON         0b00001111  /* Display on      */
#define DOFF        0b00001011  /* Display off     */
#define CURSOR_ON   0b00001111  /* Cursor on       */
#define CURSOR_OFF  0b00001101  /* Cursor off      */
#define BLINK_ON    0b00001111  /* Cursor Blink    */
#define BLINK_OFF   0b00001110  /* Cursor No Blink */

/* Cursor or Display Shift defines */
#define SHIFT_CUR_LEFT    0b00000100  /* Cursor shifts to the left   */
#define SHIFT_CUR_RIGHT   0b00000101  /* Cursor shifts to the right  */
#define SHIFT_DISP_LEFT   0b00000110  /* Display shifts to the left  */
#define SHIFT_DISP_RIGHT  0b00000111  /* Display shifts to the right */

/* Function Set defines */
#define FOUR_BIT   0b00101100  /* 4-bit Interface              */
#define EIGHT_BIT  0b00111100  /* 8-bit Interface              */
#define LINE_5X7   0b00110000  /* 5x7 characters, single line  */
#define LINE_5X10  0b00110100  /* 5x10 characters              */
#define LINES_5X7  0b00111000  /* 5x7 characters, multiple line */

#endif
//Chương trình LCD
typedef union _BYTE_VAL
{
   unsigned char Val;
   struct
   {
      unsigned char b0:1;
      unsigned char b1:1;
      unsigned char b2:1;
      unsigned char b3:1;
      unsigned char b4:1;
```

```
        unsigned char b5:1;
        unsigned char b6:1;
        unsigned char b7:1;
    } bits;
} _BYTE_VAL;

void lcd_write(unsigned char c)
{
    _BYTE_VAL TempByte;

    TempByte.Val = c;

    LCD_D4  = TempByte.bits.b4;
    LCD_D5  = TempByte.bits.b5;
    LCD_D6  = TempByte.bits.b6;
    LCD_D7  = TempByte.bits.b7;
    LCD_STROBE();
    LCD_D4  = TempByte.bits.b0;
    LCD_D5  = TempByte.bits.b1;
    LCD_D6  = TempByte.bits.b2;
    LCD_D7  = TempByte.bits.b3;
    LCD_STROBE();
}

void lcd_clear(void)
{
    LCD_RS = 0;
    lcd_write(0x01);
    __delay_ms(2);
}

/* write a string of chars to the LCD */

void lcd_puts(const char *s)
{
    LCD_RS = 1; // write characters
    while(*s)
        lcd_write(*s++);
}
```

```
/* write one character to the LCD */

void lcd_putch(char c)
{
   LCD_RS = 1; // write characters
   lcd_write( c );
}

void lcd_goto(unsigned char colum, unsigned char row)
{
   unsigned char pos[2] = {0x80,0xC0};
   LCD_RS = 0;
   lcd_write(pos[row] + colum);
}

/* initialise the LCD - put into 4 bit mode */
void lcd_init(void)
{
   LCD_PORT = 0x00;
   LCD_RS = 0;
   LCD_EN = 0;
   // reset LCD
   lcd_write(0x30);
   __delay_ms(20);
   lcd_write(0x30);
   __delay_ms(20);
   lcd_write(0x32);
   __delay_ms(20);
   lcd_write(0x28);
   __delay_ms(10); // wait 15mSec after power applied,
   lcd_write(0x28);
   __delay_ms(10); // wait 15mSec after power applied,
   lcd_write(0x28);
   __delay_ms(10); // wait 15mSec after power applied,

   lcd_write(0x28);    // Four bit mode,5x7
   __delay_ms(1);  // wait 15mSec after power applied,
   lcd_write(0x08);
```

```
    __delay_ms(1);  // wait 15mSec after power applied,
    lcd_clear();    // Clear screen
    __delay_ms(1);  // wait 15mSec after power applied,
    lcd_write(0x06); // Set entry Mode
    __delay_ms(1);  // wait 15mSec after power applied,
    lcd_write(0x0C); // Display On, Cursor On, Cursor Blink

}

void lcd_numb2(unsigned char number)
{
        lcd_putch(number/10 + '0');
        lcd_putch(number%10 + '0');
}

void lcd_numb4(unsigned int number)
{
        lcd_putch (((number/1000)%10)+ '0');
        lcd_putch (((number/100)%10)+ '0');
        lcd_putch (((number/10)%10)+ '0');
        lcd_putch((number%10)+ '0');
}
//Chương trình chính
#define _XTAL_FREQ 20000000

#include <htc.h>
#include "timer.h"
#include "lcd.h"
#include "main.h"
#include <string.h>
#include <stdio.h>

#define MODE        RC3
#define PWM         RC1
#define ENCODER     32

#define SW_SET      RA0
#define SW_INC      RA1
#define SW_DEC      RA2
```

```
enum {
   STOP = 0,
   LEFT = 1,
   RIGHT = 2
};
int speed ;
unsigned int duty_speed;

void Control_motor(unsigned char rotate,unsigned int m_speed)
{
   switch(rotate)
   {
      case STOP:{
         CCPR2L = 0;
         MODE = 0;
         CCP2CON &= 0x0F;
      }  break;
      case LEFT:{
         CCPR2L = m_speed;
         CCP2CON |= 0x30;
         MODE = 0;
      }  break;
      case RIGHT:{
         CCPR2L = 100 - m_speed;
         CCP2CON |= 0x30;
         MODE = 1;
      }  break;
      default: break;
   }
}

#define FASLE   0
#define TRUE    1

unsigned int value_adc, pulse_cnt = 0,nhietdo;
unsigned char TMR0over = 0 ,cnt_timer1 = 0, speed_flag, read_adc = 0;

void interrupt Timer(void)
```

```
{
   if (PEIE && ADIE && ADIF)
   {
      ADIF = 0;
      value_adc = (ADRESH<<8)|ADRESL;
      nhietdo = (300*value_adc)/1023;
      lcd_goto(6,0);
      lcd_numb2(nhietdo);
      if(nhietdo < 20)
      {
         duty_speed = 0;
         Control_motor(STOP,duty_speed);
      }
      else if(nhietdo >= 20 && nhietdo <=50)
      {
         duty_speed = (nhietdo*72)/2;
         Control_motor(LEFT,duty_speed);
      }
      else
      {
         duty_speed = 175;
         Control_motor(LEFT,duty_speed);
      }
   }
   if(TMR0IF)
   {
      TMR0over++;
      TMR0IF = 0;
   }
   if(TMR1IF)//1s
   {
      TMR1H = 0x3C;
      TMR1L = 0xAF;
      TMR1IF = 0;
      if(cnt_timer1++ >= 200)
      {
         pulse_cnt = (unsigned int)256*TMR0over + TMR0;
         speed_flag = TRUE;
         cnt_timer1 = 0;
```

```
            TMR0over = 0;
            TMR0 = 0;
        }
        if(read_adc++ >= 10)
        {
            read_adc = 0;
            GO_DONE = 1;
        }
    }
}
int cacul_speed_motor()
{
    float speed_motor;
    speed_motor = (pulse_cnt*70)/ENCODER; // vong/phut
    speed_flag = FASLE;
    return (int)speed_motor;
}

void main()
{
    TRISA = 0xFF;
    ADC_Init();
    PWM_Init();
    Timer0_Init();
    Timer1_Init();
    Enb_interrup();
    lcd_init();
    lcd_goto(0,0);
    lcd_puts("TEMP : ");
    lcd_goto(0,1);
    lcd_puts("SPEED: ");
    while(1)
    {
        __delay_ms(300);
        if(speed_flag == TRUE)
        {
            speed = cacul_speed_motor();
            lcd_goto(6,1);
            lcd_numb4(speed);
```

```
        }
    }

}
//Khai báo Timer
#ifndef _TIMER_H_
#define _TIMER_H_

#include <htc.h>

void Timer0_Init();
void Timer1_Init();
void ADC_Init();
void PWM_Init();
void Enb_interrup();

#endif
//Timer
#include "timer.h"
#include <stdio.h>

void Timer1_Init()
{
    TMR1CS  = 0;
    T1CKPS1 = 0;
    T1CKPS0 = 0;
    TMR1IF = 0;
    TMR1IE = 1;
    TMR1H = 0x3C;
    TMR1L = 0xAF;
    TMR1ON = 1;
}
void Timer0_Init()
{
    TMR0IF = 0 ;
    T0CS = 1;
    T0SE = 1;
    PS0 = 0;
    PS1 = 0;
```

```
    TMR0IE = 1;
    TMR0 = 0x00;

    TRISA4 = 1;
}
void ADC_Init()
{
    ANSEL = 0x80;       // ADC chanel 3
    ANSELH = 0x00;
    ADCS1 = 1;  //  F/32
    ADCS0 = 0;
    VCFG1 = 0;  //  Internal Vref
    VCFG0 = 0;
    CHS3 = 0;   // AN3
    CHS2 = 0;
    CHS1 = 1;
    CHS0 = 1;
    ADFM = 1;   //  ADFM = 1 Right justified
    ADON = 1;   // ON ADC
    ADIE = 1;    // ADC Interrupt Enable
    ADIF = 0;    // Clear ADC Interrupt Flag
}

void PWM_Init()
{
    TRISC1 = 0;
    TRISC3 = 0;
    CCPR2L = 100;
    CCP2CON = 0x3C;
    PR2 = 199;
    T2CON = 0x01;
    TMR2ON = 1;
}

void Enb_interrup()
{
    PEIE = 1 ;
    GIE = 1 ;
}
```

KẾT LUẬN

Đề tài "***Thiết kế hệ thống điều khiển động cơ DC bằng nhiệt độ***" không phải là đề tài mới mẻ và cũng không phải đề tài lớn, nhưng qua đó em đã bổ xung được rất nhiều kinh nghiệm và kiền thức có ích cho bản thân như:

- Hiểu được phương pháp điều khiển sử dụng vi điều khiển.
- Tìm hiểu thêm được về mạch công suất và Motor DC.
- Xây dựng hệ thống điều khiển cơ bản.
- Hiểu biết các cách lập trình C cho VĐK thành thạo hơn.
- Có nhiều kinh nghiệm trong việc thi công và lắp rắp mạch điện tử.
- Biết cách hiệu chỉnh các thông số mạch thực tế

Và qua thời gian làm đề tài cùng với sự hướng dẫn chỉ bảo của thầy em đã học hỏi, rèn luyện được tinh thần làm việc nghiêm túc và cách thức không ngừng nghiên cứu các kiến thức mới để bổ xung tích luỹ kiến thức cho mình.

Trong quá trình làm đề tài, do sự hạn chế về thời gian, tài liệu và trình độ có hạn nên không tránh khỏi có thiếu sót. Em rất mong được sự đóng góp ý kiến của thầy cô và các bạn để đề tài của em được hoàn thiện hơn.

Sau cùng em xin trân thành bày tỏ lòng biết ơn của mình đối với thầy NGUYỄN VĂN DƯƠNG và các thầy cô trong khoa đã giúp đỡ em hoàn thành đề tài này!

Ngày 20 tháng 6 năm 2013

Sinh viên

Mạc Minh Đức

TÀI LIỆU THAM KHẢO

1. Nguyễn Tăng C□ờng, Phan Quốc Thắng, Cấu trúc và lập trình họ Vi Điều khiển 8051, Nhà xuất bản khoa học và Kỹ Thuật.

2 Nguyễn Mạnh Giang, Cấu trúc, lập trình ghép nối và ứng dụng của Vi Điều Khiển, nhà xuất bản Lao Động – Xã Hội.

3. Phạm Minh Hà(2004), *Kỹ thuật mạch điện tử,* Nhà xuất bản Khoa học và kỹ thuật.

4. Ngô Diện Tập, Vi Điều Khiển trong đo l□ờng và điều khiển tự động, Nhà xuất bản Khoa Hoc và Kỹ Thuật, Hà Nội.

5. Họ Vi Điều Khiển 8051, Tống Văn ON, nhà Xuất bản Lao Động và Xã Hội.

6. Các bạn có thể truy cập các trang Web rất hay của Việt Nam nh□ :

www.dientuvietnam.net

www.picvietnam.com

www.dientuvienthong.net

www.vagam.dieukhien.net

www.duyphi.phpnet.us/index.htm