TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

----------&&&&&---------

LẬP TRÌNH NHÚNG

PIC 16F877A

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

----------&&&&&---------

LẬP TRÌNH NHÚNG

CHƯƠNG 4:

LẬP TRÌNH CCS CHO VI ĐIỀU KHIỂN PIC16F877A

Tự tạo mẫu chương trình

```
#include <16F877A.H>
#device *=8 ADC=10
#FUSES NOWDT, HS, NOPUT, NOPROTECT, NODEBUG, NOBROWNOUT
#use delay(clock=20000000)
#use rs232(baud=9600,parity=N,xmit=PIN_B5,rcv=PIN_B2,bits=9)
#use i2c(Master,Fast,sda=PIN_B1,scl=PIN_B4)
// Phần khai báo biến toàn cục
int8 x;
//Phần khai báo, định nghĩa các chương trình con
#int_xxx // Khai bao chuong trinh ngat
xxx_isr() {
// Code here
}
void Ten_chuong_trinh_con(Ten_Bien) {
// Code here
}
void main() {
// Enter code here!
}
```

Khai báo biến

- Các loại biến được hỗ trợ khi lập trình cho VĐK
  - *int1 (short)* số 1 bit có giá trị là 0 hay 1
  - *int8 (byte, int)* số nguyên 1 byte (8 bit)
  - *int16(long)* số nguyên 16 bit
  - *int32* số nguyên 32 bit
  - *char* ký tự 8 bit
  - *float* số thực 32 bit

  Thêm signed hoặc unsigned phía trước để chỉ số có dấu hay không. Khai báo như trên là mặc định không dấu.
- Phạm vi biến:
  - *int8*: 0, 255 *signed int8*: -128, 127
  - *int16*: 0, 2^16-1 *signed int16*: -2^15, 2^15-1
  - *int32*: 0, 2^32-1 *signed int32*: -2^31, 2^31-1

Khai báo biến

- Khai báo biến:
  - *int8 a;* *//Số a là số 8 bit không dấu*
  - *signed int8 a;* *//Số a là số có dấu (bit 7 là bit dấu)*
- Khai báo hằng
  - *int8 const a = 124;*
  - *signed int8 const b = -102;*
- Khai báo mảng hằng số
  - *int8 const a[5] = {5, 1, 2, 4, 3};*

    //Mảng a gồm 5 phần tử,

    //chỉ số bắt đầu từ 0: a[0] = 5

Khai báo biến

- Cách sử dụng biến
  - Tràn số và làm tròn:
    - *int8 a = 275;* *//a = 275-256 = 19*
    - *int8 const a = 275;* *//a = 19*
    - *int8 a = 40, b = 7, c;*
    - *c = a*b;* *//c = 280-256 = 24*
    - *c = a/b;* *//c = 5*
  - Ép kiểu để tiết kiệm RAM và thời gian tính
    - *int16 c;*
    - *int8 a = 40, b = 7;*
    - *c = (int16)a*b;* *//c = 280*
- 123: Số trong hệ cơ số 10
- 0x123: Số trong hệ cơ số 16
- 0b01000101: Số trong hệ cơ số 2

CẤU TRÚC CÂU LỆNH TRONG CCS

Cấu trúc lệnh	Ví dụ

while (biểu thức)
{ // Các lệnh;
}
	while (count<20)
	{ Output_B(count);
	Count=count+1;
	}
	Chú ý: while(1) sẽ thực hiện các lệnh trong khối while này mãi mãi

for (biểu thức 1, biểu thức 2, biểu thức 3)
{//Các lệnh;
}
	for (i =1;i <=10;++i)
	{A=a+i;
	}

switch (biến) { case giá trị 1: lệnh 1; break; case giá trị 2: lệnh 2; break; …… case giá trị n: lệnh n break; [default: lệnh n+1; break;] }	***switch (cmd) {*** ***case 0: printf("cmd 0");*** ***break;*** ***case 1: printf("cmd 1");*** ***break;*** ***default: printf("bad cmd");*** ***break; }***

CẤU TRÚC CÂU LỆNH TRONG CCS

Cấu trúc lệnh	Ví dụ

If (biểu thức)
Lệnh1;
Else
lệnh2;
	if (x==25)
	x=1;
	else
	x=x+1;

return [expr];	***return (5);***
goto label;	***goto loop;***
label: stmt;	***loop: I++;***
break;	***break;***
continue;	***continue;***
expr;	***i=1;***
{[stmt]}	***{a=1;*** ***b=1;}***

CÁC TOÁN TỬ TRONG CCS

+	Addition Operator
+=	Addition assignment operator, x+=y, is the same as x=x+y
&=	Bitwise and assignment operator, x&=y, is the same as x=x&y
&	Address operator
&	Bitwise and operator
^=	Bitwise exclusive or assignment operator, x^=y, is the same as x=x^y
^	Bitwise exclusive or operator
|=	Bitwise inclusive or assignment operator, x|=y, is the same as x=x|y
|	Bitwise inclusive or operator
?:	Conditional Expression operator
- -	Decrement
/=	Division assignment operator, x\=y, is the same as x=x/y
/	Division operator
==	Equality
>	Greater than operator
>=	Greater than or equal to operator
++	Increment
*	Indirection operator

CÁC TOÁN TỬ TRONG CCS

!=	Inequality
<<=	Left shift assignment operator, x<<=y, is the same as x=x<<y
<	Less than operator
<<	Left Shift operator
<=	Less than or equal to operator
&&	Logical AND operator
!	Logical negation operator
||	Logical OR operator
%=	Modules assignment operator x%=y, is the same as x=x%y
%	Modules operator
*=	Multiplication assignment operator, x*=y, is the same as x=x*y
*	Multiplication operator
~	One's complement operator
>>=	Right shift assignment, x>>=y, is the same as x=x>>y
>>	Right shift operator
->	Structure Pointer operation
-=	Subtraction assignment operator
-	Subtraction operator
sizeof	Determines size in bytes of operand

CÁC HÀM XỬ LÝ SỐ

- Bao gồm các hàm:
  - *sin() cos() tan() asin() acos() atan()*: các hàm lượng giác
  - *abs()*: lấy giá trị tuyệt đối
  - *ceil()*: làm tròn theo hướng tăng
  - *floor()*: làm tròn theo hướng giảm
  - *exp()*: tính giá trị e^x
  - *log()*: tính giá trị *ln()*
  - *log10*: tính giá trị *lg()*
  - *pow()*: tính luỹ thừa
  - *sqrt()*: tính giá trị căn bậc 2

CÁC HÀM XỬ LÝ BIT

- shift_left(address, byte,value)
- shift_right(address,byte,value)

  Dịch trái hoặc dịch phải 1 bit vào mảng hay 1 cấu trúc. Địa chỉ có thể là mảng hay địa chỉ con trỏ tới cấu trúc (kiểu như &data). Bit 0 byte thấp nhất là LSB
- rotate_right(address, byte), rotate_left(address, byte)

  Quay 1 bit trong 1 mảng hay 1 cấu trúc. Địa chỉ có thể là mảng hay địa chỉ con trỏ tới cấu trúc (kiểu như &data). Bit 0 byte thấp nhất là LSB
- bit_test(var, bit)

  Dùng để kiểm tra vị trí bit trong biến *var*. Hàm trả về 0 hay 1 là giá trị bit trong biến *var*.
  - *var*: Biến 8, 16, hoặc 32 bit bất kỳ.
  - *bit*: vị trí bit trong biến *var*

CÁC HÀM XỬ LÝ BIT

- bit_clear(var, bit), bit_set(var, bit)
  - *bit_clear* dùng để xoá (*set* = 0) bit được chỉ định bởi vị trí bit trong biến *var*.
  - *bit_set* dùng để *set* = 1 bit được chỉ định bởi vị trí bit trong biến *var*.
  - *var*: Biến 8, 16, hoặc 32 bit bất kỳ.
  - *bit*: vị trí *clear* (*set*), có thể 0-7 (biến 8 bit); 0-15 (biến 16 bit); 0-31 (biến 32 bit).
  - Hàm không trả về giá trị.
- swap(var)
  - *var*: Biến 8 bit.
  - Hàm này tráo vị trí 4 bit trên và 4 bit dưới của *var*, hàm không trả về giá trị

CÁC HÀM XỬ LÝ BIT

- make8(var, offset)
  - Hàm này trích 1 byte từ biến *var*.
  - *var*: Biến 8, 16, 32 bit
  - *offset*: vị trí của byte cần trích (0, 1, 2, 3)
  - Hàm trả về giá trị byte cần trích.
  - VD:

    Int16 x = 1234; //x = 0x4D2

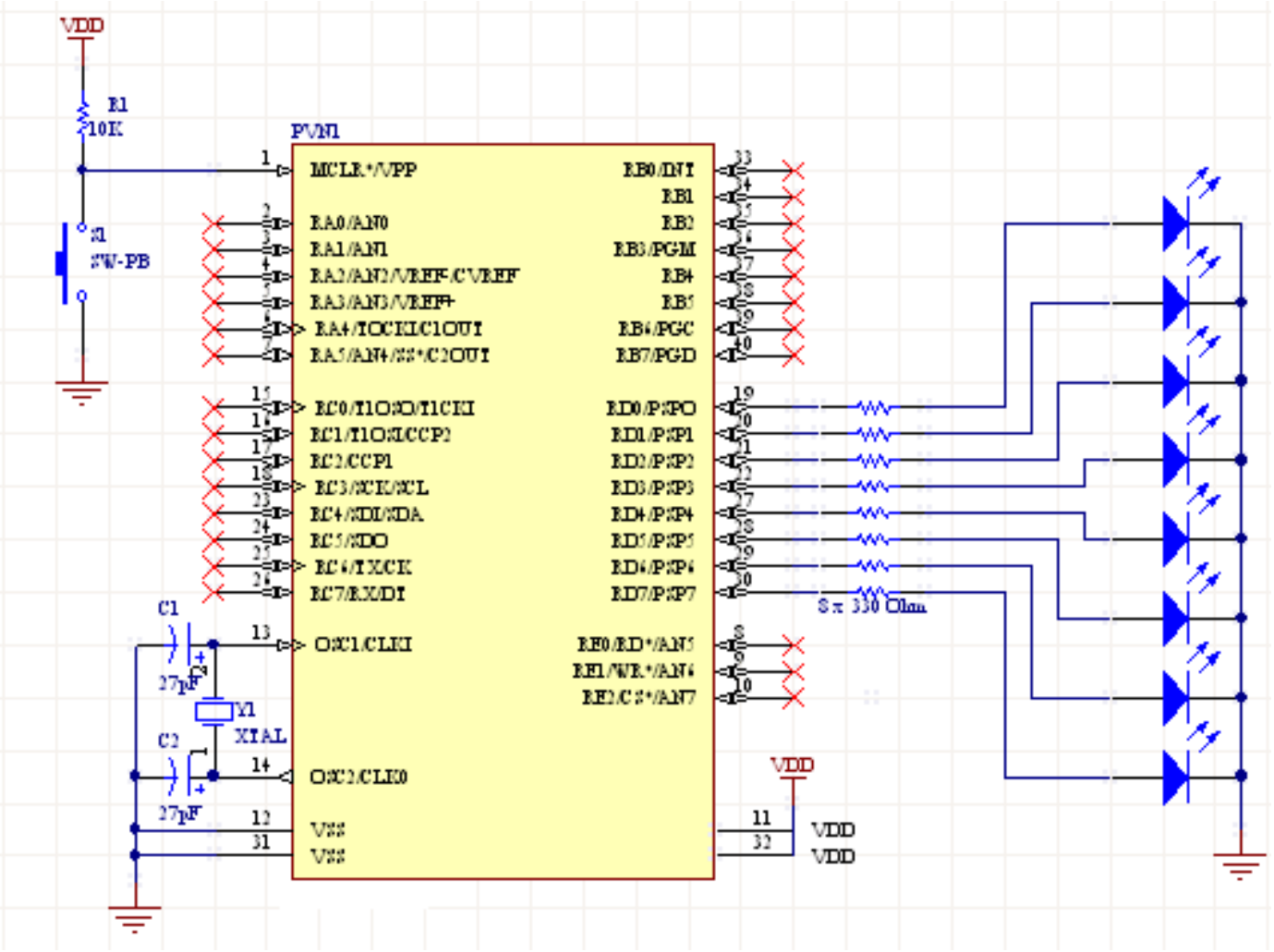
    Y = make8(x, 1) //Y = 4 = 0x04
- make16(varhigh, varlow)
  - Trả về giá trị 16 bit kết hợp từ 2 biến 8 bit *varhigh* và *varlow*, byte cao là *varhigh*, byte thấp là *varlow*.

CÁC HÀM XỬ LÝ BIT

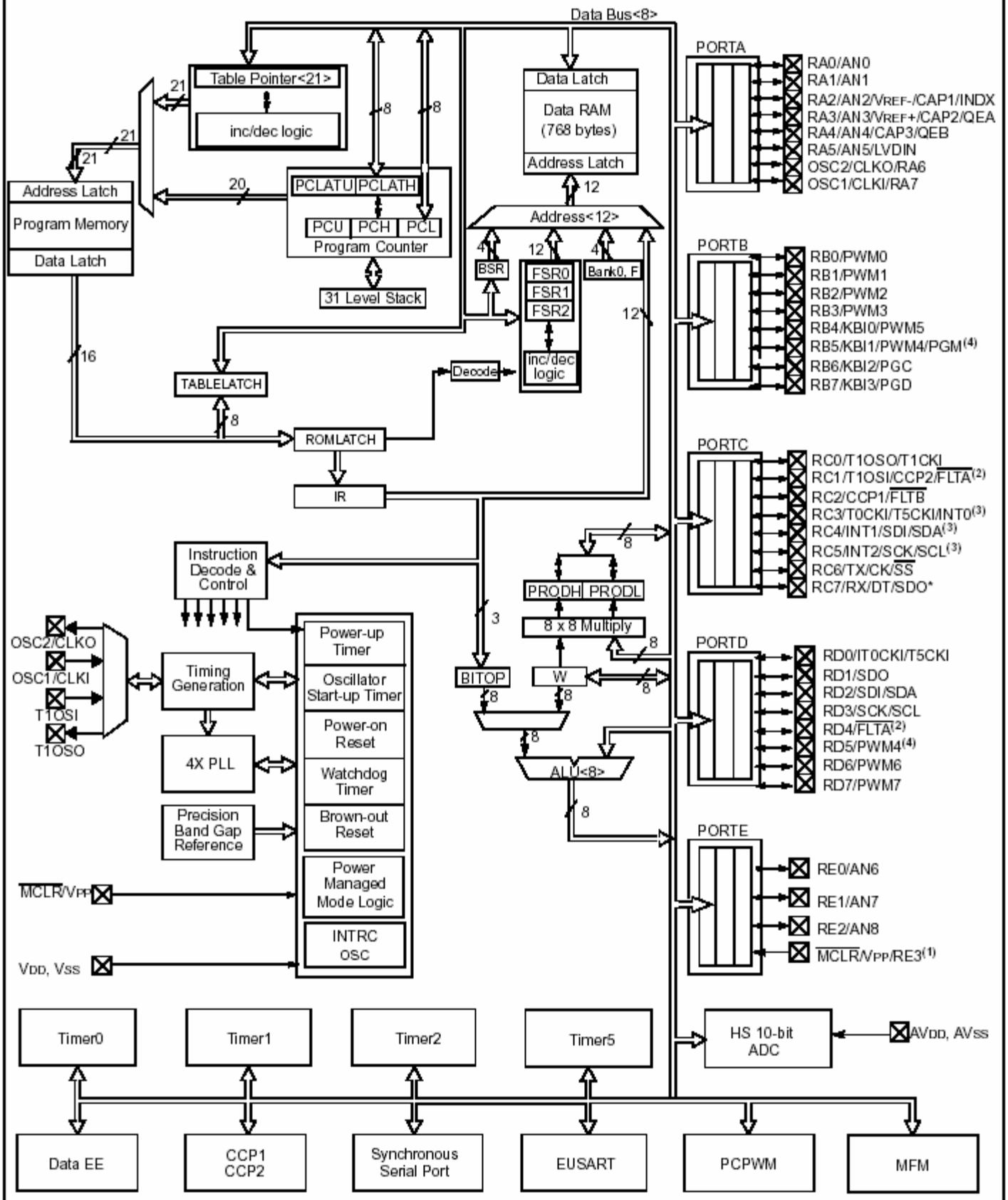
- make32(var1, var2, var3, var4)
  - Trả về giá trị 32 bit kết hợp từ các giá trị 8 bit hay 16 bit từ *var1* tới *var4*. Trong đó *var2* đến *var4* có thể có hoặc không. Giá trị *var1* sẽ là MSB, kế tiếp là *var2*, ... Nếu tổng số bit kết hợp ít hơn 32 bit thì 0 được thêm vào MSB cho đủ 32 bit.
  - VD:

```
Int8 a = 0x01, b = 0x02, c = 0x03, d = 0x04;//Các giá trị Hex
Int32 e;
e = make32(a, b, c, d);      //e = 0x01020304
e = make32(a, b, c, 5);      //e = 0x01020305
e = make32(a, b, 8);         //e = 0x00010208
e = make32(a, 0x1237);       //e = 0x00011237
```

CÁC HÀM VÀO/RA DỮ LIỆU



```
1.  #include<16F877A.h>
2.  #FUSES XT,NOWDT,PUT,NOPROTECT,NOBROWNOUT,NOLVP
3.  /* XT: su dung thach anh tan so thap (=4mhz)
4.  NOWDT:khong su dung che do Watch Dog Timer
5.  PUT(power up timer): chon che do lam viec cho pic khi dien ap da on dinh
6.  NOPROTECT : khong cho phep doc lai ma chuong trinh trong chip
7.  NOBROWNOUT : khong reset lai pic khi bi sut ap
8.  NOLVP(low voltage programming) : ?? */
9.  #USE DELAY(CLOCK=4000000)
10. #USE FAST_IO(D)
11. void main()
12. { set_tris_b(0);
13.       while(true) //tao vong lap vo tan
14.         { output_high(PIN_D0); delay_ms(500);
15.           output_low(PIN_D0); delay_ms(500);
16.           output_D(255); delay_ms(500);
17.            output_D(0); delay_ms(500);
18.          }
19.  }
```



Note 1: RE3 is available only when $\overline{\text{MCLR}}$ is disabled.

2: RD4 is the alternate pin for $\overline{\text{FLTA}}$.

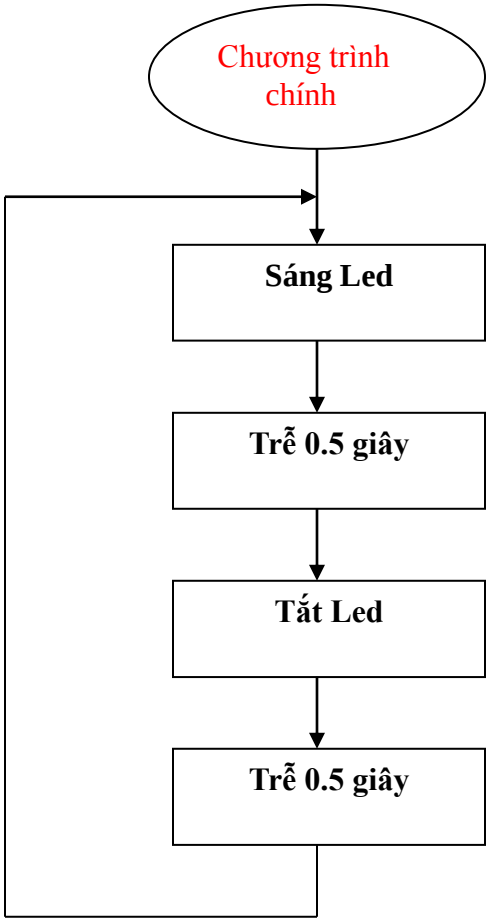
3: RC3, RC4 and RC5 are alternate pins for T0CKI/T5CKI, SDI/SDA, SCK/SCL respectively.

4: RD5 is the alternate pin for PWM4.

CÁC HÀM VÀO/RA DỮ LIỆU

- Output_low(pin), Output_high(pin)
  - Dùng thiết lập mức 0 (low, 0V) hay mức 1 (high, 5V) cho chân IC, pin chỉ vị trí chân. Hàm này sẽ đặt pin làm đầu ra, dài 2-4 chu kỳ máy.
  - VD: Chương trình sau xuất xung vuông chu kỳ 1s, duty = 50% ra chân B0, Nối B0 với 1 LED sẽ làm nhấp nháy LED.

```
#Include<16F877A.h>        //Khai bao su dung vi dieu khien PIC16F877A
#Use delay(clock=12000000)//Su dung bo dao dong thach anh 12MHz
Void Main()                    //Chuong trinh chinh
{   While(1)              //Vong lap vo han
    {         Output_high(Pin_B0); //Sang LED noi voi chan B0
              Delay_ms(500);                    //delay 500ms
              Output_low(Pin_B0); //Tat LED noi voi chan B0
              Delay_ms(500);                    //delay 500ms
}
}
```


Chương trình chính
Sáng Led
Trễ 0.5 giây
Tắt Led
Trễ 0.5 giây

CÁC HÀM VÀO/RA DỮ LIỆU

- Output_bit(Pin, Value)
  - *Pin*: Tên chân
  - *Value*: Giá trị 0 Hay 1
  - Hàm này cũng xuất giá trị 0 hoặc 1 trên Pin tương tự như 2 hàm trên. Thường nó dùng khi giá trị ra tuỳ thuộc giá trị biến nào đó hay muốn xuất đảo giá trị ngõ ra trước đó.

```
#Include<16F877A.h>                  //Khai bao su dung vi dieu khien PIC16F877A
#Use delay(clock=12000000)            /Su dung bo dao dong thach anh 12MHz
Void Main()                           //Chuong trinh chinh
{   While(1)                          //Vong lap vo han
    {       Output_bit(Pin_B0,1);      //Sang LED noi voi chan B0
            Delay_ms(500);            //delay 500ms
            Output_bit(Pin_B0,0);     //Tat LED noi voi chan B0
            Delay_ms(500);            //delay 500ms
}
}
```

CÁC HÀM VÀO/RA DỮ LIỆU

- Output_bit(Pin, Value)
  - VD: Chương trình xuất xung vuông chu kỳ 1s, duty = 50%

```
#Include<16F877A.h>
#Use delay(clock=12000000)
Int1 x=1;
Void Main()        //Chuong trinh chinh
{
   While(1)        //Vong lap vo han
   {
         Output_bit(Pin_B0, !x);//Dao trang thai chan B0
         Delay_ms(500);         //delay 500ms
   }
}
```

CÁC HÀM VÀO/RA DỮ LIỆU

- Input(Pin)

  Hàm này trả về giá trị 0 hay 1 là trạng thái của chân vi điều khiển, giá trị là 1 bit.

  - VD:

    X=input(Pin_B0); //X có giá trị 1 bit : 0 hoặc 1

- Output_X(Value)

  - X: Tên port có trên vi điều khiển,
  - Value: Giá trị 1 byte.

  Hàm này xuất ra giá trị 1 byte ra port, tất cả chân của port đó đều là ngõ ra.

  - VD:

    Output_B(135); //Xuất giá trị 0b10000111 ra port B

CÁC HÀM VÀO/RA DỮ LIỆU

- Input_X()
  - X: Tên port có trên vi điều khiển,
  - Hàm này trả về giá trị 8 bit là giá trị đang hiện hữu của port đó.
  - VD:

```
Int8 m;
m = Input_C(); //m là dữ liệu 8 bit
```

CÁC HÀM VÀO/RA DỮ LIỆU

- Set_tris_X(Value)
  - Hàm này định nghĩa chân IO cho 1 port là ngõ vào hay ngõ ra. Chỉ được dùng với ***#Use fast_IO( ).***
  - Value: Giá trị 8 bit, mỗi bit đại diện 1 chân và bit = 1 sẽ đặt chân đó là ngõ vào, bit = 0 đặt chân đó là ngõ ra.
  - VD:

```
Set_tris_B(0xE2) //0xE2 = 0b11100010: B1, B5, B6, B7 là Input
                 //còn lại là Output
```

  - Chú ý
    - Set_tris_B(0) //PortB =0b00000000: Tất cả là Output
    - Set_tris_B(255)//PortB =0b11111111: Tất cả là Input
    - Set_tris_B(1) //PortB=0b00000001: Chỉ chân B0 là Input, còn lại là Output

#Use fast_IO( )

- *Trong CCS, nếu không sử dụng "fast_io(PORT) " - (với cấu trúc #use fast_io(PORT)) thì khi sử dụng các câu lệnh input, output nó sẽ tự động dùng lệnh set_tris_x() trước khi xuất hoặc đọc logic trên các chân.*
- *Cách này có ưu điểm là không cần dùng lệnh set_tris_x() mà chương trình vẫn đúng ý người viết nhưng nhược điểm là tốn thêm dung lượng chương trình (do chèn thêm lệnh set_tris_x). Cách này thường dùng cho người mới học khi chưa hình dung hoặc quản lý chặt được những gì mình viết.*
- *Còn cách sử dụng fast_io() thì khi sử dụng các câu lệnh input, output phải set_tris_x() đúng thì chương trình mới chạy đúng. Ưu điểm là tốn ít dung lượng chương trình hơn, câu lệnh xử lý nhanh hơn. Nhược điểm là phải hiểu rõ cơ chế hoạt động IO của PIC thì mới quản lý được.*

CÁC HÀM DELAY

- Để sử dụng các hàm Delay, cần có khai báo tiền xử lý ở đầu chương trình.
- Ví dụ: Sử dụng bộ dao động thạch anh có tần số 12MHz, thì cần phải khai báo

  #Use delay(clock = 12000000)
- Có 3 hàm Delay sau:
  - Delay_cycles(Count)
  - Delay_us(Time)
  - Delay_ms(Time)

CÁC HÀM DELAY

- Delay_cycles(Count)
  - *Count*: Hằng số từ 0-255, là số chu kỳ lệnh (1 chu kỳ lệnh = 4 chu kỳ máy).
  - Hàm không trả về giá trị, hàm dùng để Delay 1 số chu kỳ lệnh cho trước.
  - VD:

```
Delay_cycles(30);          //Với thạch anh 12MHz,
                           //hàm này delay 30*4/12000000=10us
```

CÁC HÀM DELAY

- Delay_us(Time)
  - *Time*: là biến số thì từ 0-255, là hằng số thì từ 0-65535.
  - Hàm không trả về giá trị, hàm này cho phép delay khoảng thời gian Time theo đơn vị *us*
  - VD:

```
Int8 a=125;
Delay_us(a);        //Delay 125us
Delay_us(5000);  //Delay 5000us = 5ms
```

CÁC HÀM DELAY

- Delay_ms(Time)
  - *Time*: là biến số thì từ 0-255, là hằng số thì từ 0-65535.
  - Hàm không trả về giá trị, hàm này cho phép delay khoảng thời gian Time theo đơn vị *ms*
  - VD:

```
Int8 a=125;
Delay_ms(a);       //Delay 125ms
Delay_ms(5000); //Delay 5000ms = 5s
```

Nguyên lý hoạt động cơ bản của một Timer

- **Chế độ định thời:**

– Mỗi bộ timer có một hoặc nhiều thanh ghi chứa giá trị đếm của nó (tùy thuộc vào độ dài của timer), ta giả sử tên thanh ghi là TMR có độ dài là n byte, hay giá trị đếm tối đa là 2^{8n}-1. Khi giá trị của TMR = 2^{8n}, vi điều khiển sẽ set bit cờ (Flat) của bộ timer đó lên mức 1. Đồng thời TMR sẽ tự động xóa về giá trị 0. Do đó bằng cách kiểm tra bit cờ ta có thể biết Timer đếm bị tràn hay chưa.

– *Trong chế độ định thời, Giá trị của thanh ghi TMR sẽ tự động tăng lên 1 đơn vị sau mỗi chu kì lệnh của vi điều khiển.*

- Giả sử vi điều khiển dùng thạch anh tần số 4MHz(ck máy=1μs), như vậy để định thời 200(μs) *dùng Timer0 (8 bit; n=1; giá trị tối đa là 255) ta cho TMR0= 256-200=56* rồi bắt đầu cho dếm lên.

Nguyên lý hoạt động cơ bản của một Timer

- **Chế độ bộ đếm:**

– Trong chế độ này, một chân chức năng trên VĐK sẽ trở thành chân đầu vào xung của bộ đếm.

Ví dụ: chân RA4 đối với Timer0 và RC0 dối với Timer1.

– *Khi được cài đặt hoạt động trong chế độ bộ dếm, Giá trị của thanh ghi TMR sẽ tự động tăng lên 1 đơn vị khi có một xung vào chân đầu vào xung của timer đó. Khi giá trị của TMR tràn, vi điều khiển sẽ set bit cờ (Flat) của bộ timer đó lên mức 1 và TMR bị xóa, TMR=0.*

– Dạng xung được xác định là suờn âm hay sườn dương phụ thuộc vào việc cài đặt bit chọn dạng xung tương ứng trên thanh ghi của vi điều khiển.

CÁC HÀM SỬ DỤNG TIMER

- Timer0: 8 bit hoặc 16 bit
- Các hàm liên quan khi sử dụng Timer0:
  - *Setup_timer_0(Mode)*: Thiết lập các tham số cho Timer0.
  - *Set_timer0(Value)* hoặc *Set_rtcc(Value)*: Thiết lập giá trị khởi điểm của Timer0, Value có thể là 8 bit hoặc 16 bit tuỳ thuộc vào vi điều khiển.
  - *Value = Get_timer0*: Trả ra giá trị của Timer0.
- ***Các thanh ghi liên quan đến TIMER0:***
- ✓ TMR0 (địa chỉ….) : chứa giá trị đếm của Timer0.
- ✓ INTCON (địa chỉ ……): cho phép ngắt hoạt động (GIE và PEIE).
- ✓ OPTION_REG (địa chỉ ……): điều khiển prescaler.

Các thanh ghi liên quan đến TIMER0:

TABLE 11-1: REGISTERS ASSOCIATED WITH TIMER0

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
TMR0L	Timer0 Module Low Byte Register								xxxx xxxx	uuuu uuuu
TMR0H	Timer0 Module High Byte Register								0000 0000	0000 0000
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
T0CON	TMR0ON	T016BIT	T0CS	T0SE	PSA	T0PS2	T0PS1	T0PS0	1111 1111	1111 1111
TRISA	RA7(1)	RA6(1)	PORTA Data Direction Register						1111 1111	1111 1111

Legend: x = unknown, u = unchanged, – = unimplemented locations read as '0'. Shaded cells are not used by Timer0.
Note 1: RA6 and RA7 are enabled as I/O pins depending on the Oscillator mode selected in Configuration Word 1H.

- Thanh ghi:
- INTCON có địa chỉ là 0xFF2h
- T0CON có địa chỉ là 0xFD5h

TABLE 5-1: SPECIAL FUNCTION REGISTER MAP FOR PIC18F2331/2431/4331/4431 DEVICES

Address	Name	Address	Name	Address	Name	Address	Name	Address	Name
FFFh	TOSU	FDFh	INDF2	FBFh	CCPR1H	F9Fh	IPR1	F7Fh	PTCON0
FFEh	TOSH	FDEh	POSTINC2	FBEh	CCPR1L	F9Eh	PIR1	F7Eh	PTCON1
FFDh	TOSL	FDDh	POSTDEC2	FBDh	CCP1CON	F9Dh	PIE1	F7Dh	PTMRL
FFCh	STKPTR	FDCh	PREINC2	FBCh	CCPR2H	F9Ch	—	F7Ch	PTMRH
FFBh	PCLATU	FDBh	PLUSW2	FBBh	CCPR2L	F9Bh	OSCTUNE	F7Bh	PTPERL
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	ADCON3	F7Ah	PTPERH
FF9h	PCL	FD9h	FSR2L	FB9h	ANSEL1	F99h	ADCHS	F79h	PDC0L
FF8h	TBLPTRU	FD8h	STATUS	FB8h	ANSEL0	F98h	—	F78h	PDC0H
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	T5CON	F97h	—	F77h	PDC1L
FF6h	TBLPTRL	FD6h	TMR0L	FB6h	QEICON	F96h	TRISE	F76h	PDC1H
FF5h	TABLAT	FD5h	T0CON	FB5h	—	F95h	TRISD	F75h	PDC2L
FF4h	PRODH	FD4h	—	FB4h	—	F94h	TRISC	F74h	PDC2H
FF3h	PRODL	FD3h	OSCCON	FB3h	—	F93h	TRISB	F73h	PDC3L
FF2h	INTCON	FD2h	LVDCON	FB2h	—	F92h	TRISA	F72h	PDC3H
FF1h	INTCON2	FD1h	WDTCON	FB1h	—	F91h	PR5H	F71h	SEVTCMPL
FF0h	INTCON3	FD0h	RCON	FB0h	SPBRGH	F90h	PR5L	F70h	SEVTCMPH
FEFh	INDF0	FCFh	TMR1H	FAFh	SPBRG	F8Fh	—	F6Fh	PWMCON0
FEEh	POSTINC0	FCEh	TMR1L	FAEh	RCREG	F8Eh	—	F6Eh	PWMCON1
FEDh	POSTDEC0	FCDh	T1CON	FADh	TXREG	F8Dh	LATE	F6Dh	DTCON
FECh	PREINC0	FCCh	TMR2	FACh	TXSTA	F8Ch	LATD	F6Ch	FLTCONFIG
FEBh	PLUSW0	FCBh	PR2	FABh	RCSTA	F8Bh	LATC	F6Bh	OVDCOND
FEAh	FSR0H	FCAh	T2CON	FAAh	BAUDCTL	F8Ah	LATB	F6Ah	OVDCONS
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEADR	F89h	LATA	F69h	CAP1BUFH
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F88h	TMR5H	F68h	CAP1BUFL
FE7h	INDF1	FC7h	SSPSTAT	FA7h	EECON2	F87h	TMR5L	F67h	CAP2BUFH
FE6h	POSTINC1	FC6h	SSPCON	FA6h	EECON1	F86h	—	F66h	CAP2BUFL
FE5h	POSTDEC1	FC5h	—	FA5h	IPR3	F85h	—	F65h	CAP3BUFH
FE4h	PREINC1	FC4h	ADRESH	FA4h	PIR3	F84h	PORTE	F64h	CAP3BUFL
FE3h	PLUSW1	FC3h	ADRESL	FA3h	PIE3	F83h	PORTD	F63h	CAP1CON
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC	F62h	CAP2CON
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB	F61h	CAP3CON
FE0h	BSR	FC0h	ADCON2	FA0h	PIE2	F80h	PORTA	F60h	DFLTCON

CÁC HÀM SỬ DỤNG TIMER

- Timer0
  - VD: Sử dụng PIC16F877A, thạch anh 12MHz

```
Int8 Time;
Setup_timer_0(RTCC_INTERNAL|RTCC_DIV_2|RTCC_8_BIT);
//Sử dụng Timer0 với 8bit và Prescale=2,tần số=tần số thạch anh
//Thời gian tràn Timer0 =2*256*(4/12000000)s=170.667us
Set_timer0(0);     //Thiết lập giá trị đầu của Timer0 là 0
Time = Get_timer0();//Đọc giá trị của Timer0 gán cho biến Time
```

- Tỉ lệ chia tần số là một trong những từ sau: tương ứng với tỉ lệ chia

RTCC_DIV_1,

RTCC_DIV_2, RTCC_DIV_4,

RTCC_DIV_8, RTCC_DIV_16,

RTCC_DIV_32, RTCC_DIV_64,

RTCC_DIV_128, RTCC_DIV_256

Ví dụ 1 Timer0

- Ví dụ sử dụng thạch anh 4 MHz, timer 0 (8 bit), bộ chia tần 4, trễ 200us thì ta tính theo công thức sau :

$$256 - \frac{200us}{4.\frac{4}{4MHz}} = 206$$

Giải thích các số trên như sau :

- 256 vì đây là Timer 8 bit.

- 200 : đếm 200us

- 4 : bộ chia tần 4 = tỷ lệ 1:4 nghĩa là 4 chu kỳ máy thì mới tăng bộ đếm 1 đv

- 4MHz: thạch anh

Như vậy chúng ta phải cài đặt giá trị cho Timer 0 như sau :

```
setup_timer_0(RTCC_INTERNAL|RTCC_DIV_4);
set_timer0(206);
```

Ví dụ 2 Timer0

- Ví dụ sử dụng thạch anh 12 MHz, timer 0 (8 bit), bộ chia tần 4, trễ 200us thì ta tính theo công thức sau :

$$256 - \frac{200us}{4.\frac{4}{12MHz}} = 106$$

Giải thích các số trên như sau :

- 256 vì đây là Timer 8 bit.

- 200 : đếm 200us

- 4 : bộ chia tần 4 = tỷ lệ 1:4 nghĩa là 1 chu kỳ máy thì mới tăng bộ đếm 1 đv

- 12MHz: thạch anh

Như vậy chúng ta phải cài đặt giá trị cho Timer 0 như sau :

```
setup_timer_0(RTCC_INTERNAL|RTCC_DIV_4);
set_timer0(106);
```

Ví dụ 2 Timer0

- Ví dụ sử dụng thạch anh 12 MHz, timer 0 (8 bit), bộ chia tần 64, trễ 2000us=2ms thì ta tính theo công thức sau :

$$256 - \frac{2000us}{64.\frac{4}{12MHz}} = \underline{\mathbf{93.75}} \rightarrow \mathbf{256 - \frac{x}{a.\frac{4}{12MHz}} = b(chẵn) \rightarrow x = ?}$$

Giải thích các số trên như sau:

- 256 vì đây là Timer 8 bit.

- 200 : đếm 200us

- 64 : bộ chia tần 64 = tỷ lệ 1:44 nghĩa là 64 chu kỳ máy thì mới tăng bộ đếm 1 đv

- 12MHz: thạch anh

CÁC HÀM SỬ DỤNG TIMER

- Counter0
  - Các hàm liên quan khi sử dụng Counter0:

1/Setup_timer_0 (chế độ|tỉ lệ chia tần số)

- Chế độ có thể là một trong những từ sau:

- *RTCC_EXT_L_TO_H*: chế độ bộ đếm với xung dạng suờn lên
- *RTCC_EXT_H_TO_L:* chế độ bộ đếm với xung dạng suờn xuống

- Tỉ lệ chia tần số là một trong những từ sau: tương ứng với tỉ lệ chia

RTCC_DIV_2, RTCC_DIV_4, RTCC_DIV_1,
RTCC_DIV_8, RTCC_DIV_16,
RTCC_DIV_32, RTCC_DIV_64,
RTCC_DIV_128, RTCC_DIV_256

2/ Set_timer0(Value)* hoặc *Set_rtcc(Value)

***3/ Value = Get_timer0*: Trả ra giá trị của Timer0.**

CÁC HÀM SỬ DỤNG TIMER

- Timer1
  - Các hàm liên quan khi sử dụng Timer1:
    - *Setup_timer_1(Mode)*: Thiết lập các tham số cho Timer1.
    - *Set_timer1(Value)*: Thiết lập giá trị khởi điểm của Timer1.
    - *Value = Get_timer1*: Trả ra giá trị của Timer1.
  - VD: Sử dụng PIC16F877A, thạch anh 12MHz

```
Int8 Time;
Setup_timer_1(T1_INTERNAL|T1_DIV_8);
   //Sử dụng Timer1 với Prescale = 8, tần số = tần số thạch anh
   //Thời gian tràn Timer1 =8*65536*(4/12000000)s=174.763ms
Set_timer1(0);          //Thiết lập giá trị đầu của Timer1 là 0
Time = Get_timer1();//Đọc giá trị của Timer1 gán cho biến Time
```

CÁC HÀM SỬ DỤNG TIMER

- Timer2
  - Các hàm liên quan khi sử dụng Timer1:
    - *Setup_timer_2(Mode, Period, Postscale)*: Thiết lập các tham số cho Timer2.
    - *Set_timer2(Value)*: Thiết lập giá trị khởi điểm của Timer2.
    - *Value = Get_timer2*: Trả ra giá trị của Timer2.
  - VD: Sử dụng PIC16F877A, thạch anh 12MHz

```
Int8 Time;
Setup_timer_2(T2_DIV_4, 0xC0, 2);
   //Sử dụng Timer2 với Prescale = 4, tần số = tần số thạch anh,
   //Period = 0xC0=192, Postscale = 2,
   //Thời gian tràn Timer2
   //=4*(192+1)*2*(4/12000000)s=514.667us
Set_timer2(0);          //Thiết lập giá trị đầu của Timer2 là 0
Time = Get_timer2();//Đọc giá trị của Timer2 gán cho biến Time
```

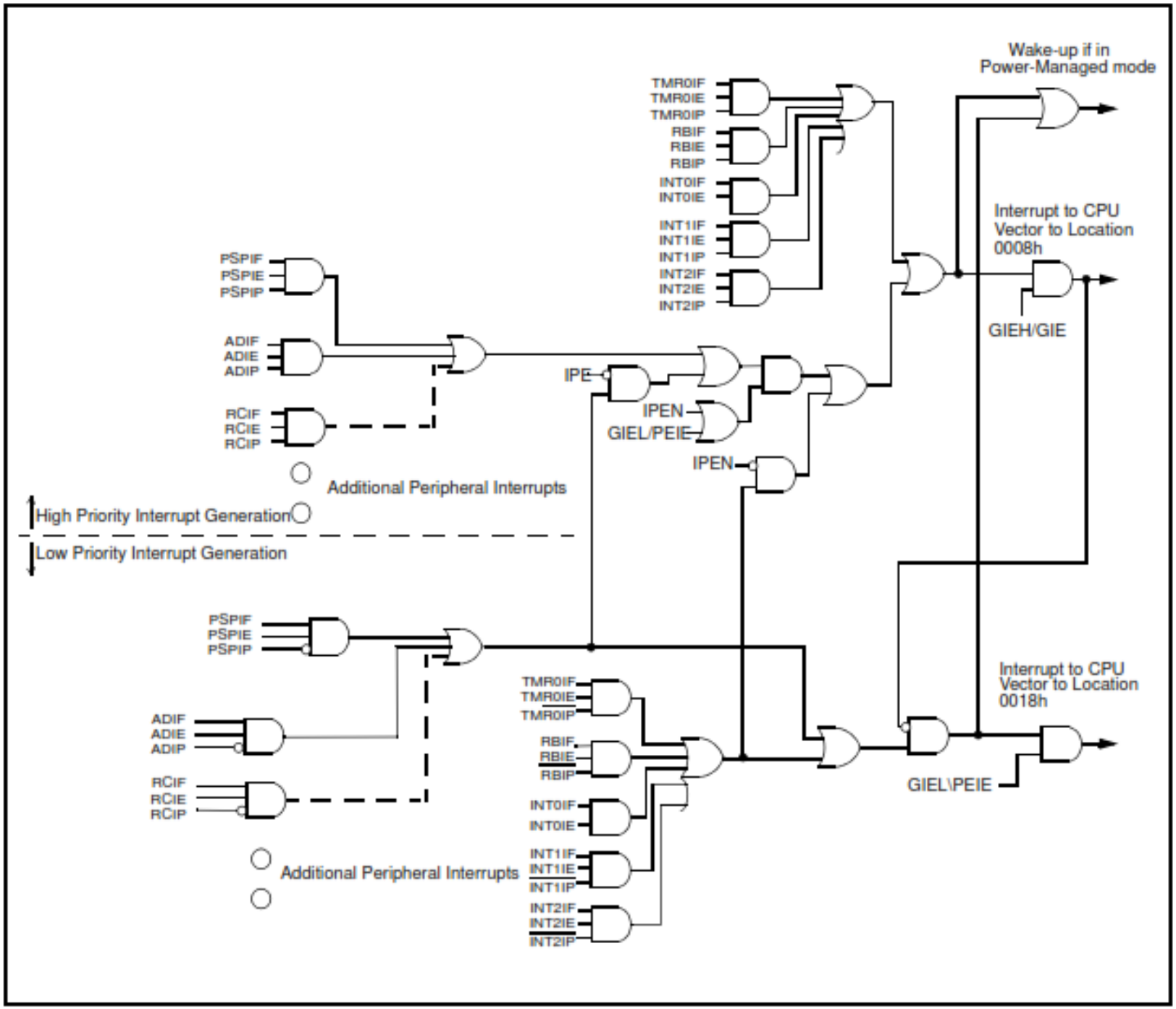
CÁC HÀM SỬ DỤNG NGẮT

- Khai báo ngắt
  - Mỗi dòng vi điều khiển có số lượng nguồn ngắt khác nhau: PIC14 có 14 ngắt, PIC18 có 35 ngắt.
  - Muốn biết CCS hỗ trợ những ngắt nào cho vi điều khiển có thể xem trong file *.h tương ứng, ở cuối file là các ngắt mf CCS hỗ trợ nó.
  - Hoặc có thể vào CCS -> View -> Valid interrups tiếp đó chọn vi điều khiển muốn xem nó sẽ hiển thị danh sách ngắt có thể có cho vi điều khiển đó.
  - Hàm đi kèm phục vụ ngắt không cần tham số vì tham số không có tác dụng. Sử dụng NOCLEAR sau #INT_XXX để CCS không xoá cờ ngắt của hàm đó.
  - Để cho phép ngắt đó hoạt động phải dùng lệnh **Enable_interrupts(INT_XXX)** và **Enable_interrupts(GLOBAL)**. Khoá FAST theo sau #INT_XXX để cho ngắt đó ưu tiên cao, chỉ được 1 ngắt và chỉ có ở PIC18 và dsPIC.
  - VD: ***#INT_TIMER0*** *FAST NOCLEAR*

CÁC HÀM SỬ DỤNG NGẮT

- Các ngắt trong PIC16F877A:
  - GLOBAL : cho phép ngắt toàn cục
  - INT_TIMER0 : ngắt do tràn TIMER0
  - INT_TIMER1 : ngắt do tràn Timer1
  - INT_TIMER2 : ngắt do tràn Timer2
  - INT_RB : có thay đổi 1 trong các chân RB4 _ RB7
  - INT_EXT : ngắt ngoài trên chân RB0
  - …..

FIGURE 9-1: INTERRUPT LOGIC



CÁC HÀM SỬ DỤNG NGẮT

- Enable_interrupts(Level)
  - Level: Tên các ngắt đã cho ở Bảng trên hay là GLOBAL để cho phép ngắt ở cấp toàn cục.
  - Mọi ngắt của vi điều khiển đều có 1 bit cờ ngắt, 1 bit cho phép ngắt. Khi có ngắt thì bit cờ ngắt bị set=1, những ngắt có hoạt động được hay không tuỳ thuộc vào bit cho phép ngắt. Enable_interrupts(INT_XXX) sẽ bật bit cho phép ngắt. Nhưng tất cả các ngắt đều không thể thực thi nếu bit cho phép ngắt toàn cục = 0, Enable_interrupts(GLOBAL) sẽ bật bit này.
  - VD: Để cho phép ngắt Timer0 và Timer1 hoạt động:

    Enable_interrupts(INT_TIMER0);

    Enable_interrupts(INT_TIMER1);

    Enable_interrupts(GLOBAL);//Chỉ cần dùng 1 lần trừ phi muốn có //thay đổi đặc biệt

CÁC HÀM SỬ DỤNG NGẮT

- Disable_interrupts(Level)
  - Level giống như trong hàm Enable_interrupts()
  - Hàm này vô hiệu 1 ngắt bằng cách set bit cho phép ngắt = 0,
  - Disable_interrupts(global) set bit cho phép ngắt toàn cục = 0 (cấm tất cả các ngắt)
  - ***Không dùng hàm này trong hàm phục vụ ngắt vì không có tác dụng, cờ ngắt luôn bị xoá tự động.***
- Clear_interrupt(Level)
  - Level không có GLOBAL
  - Hàm này xoá cờ ngắt của ngắt được chỉ định bởi Level

CÁC HÀM SỬ DỤNG NGẮT

- Ext_int_edge(Source, Edge)
  - Hàm này thiết lập **nguồn ngắt ngoài EXTx** là cạnh lên hay xuống.
  - *Source*: Nguồn ngắt, trên PIC18 có 3 nguồn ngắt trên 3 chân EXT0, EXT1, EXT2 ứng với source = 0, 1, 2. Các PIC khác chỉ có 1 nguồn ngắt EXT nên source = 0.
  - *Edge*: Chọn cạnh kích ngắt, Edge = L_TO_H nếu chọn cạnh lên hay H_TO_L nếu chọn cạnh xuống.

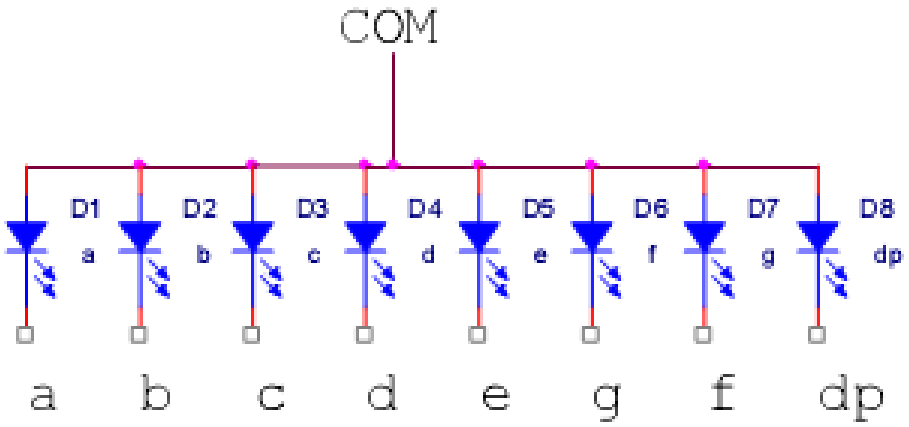
CÁC HÀM SỬ DỤNG NGẮT

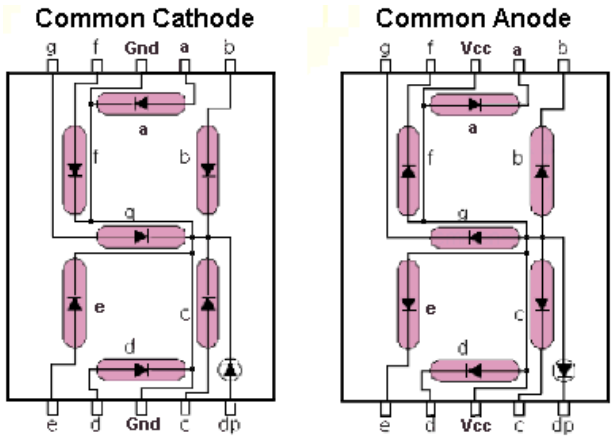
```
//Ngắt 1s trên Timer0
#include <16F877A.h>
#device 18F4680*=16 ADC=10
#fuses NOWDT,PUT,XT,NOPROTECT
#use delay(clock=4000000)
#use fast_io(c)
#byte portc=0x07
int16 count;
int8 a;
#int_timer0 //Chuong trinh ngat TMR0
void timer0()
{set_timer0(56); // T = 2*(256 - 56)*1us = 400us
(nếu dùng thạch anh 20M thi thay 1us thành 0,2us)
++count;
if(count == 2500)
 // 2500*400us = 1000000us = 1s
{count=0;
rotate_left(&a,1);
}
}
```

```
//Chuong trinh dich led
void main()
{ set_tris_c(0);
enable_interrupts(int_timer0);
setup_timer_0(RTCC_INTERNAL|RTC
C_DIV_2);
enable_interrupts(global);

a = 0x01;
while(true)
{
portc = a;
}
}
```

LED 7 đoạn-Loại Anode chung





Bảng mã cho Led Anode chung (a là MSB, dp là LSB):

Số	a	b	c	d	e	f	g	dp	Mã hex
0	0	0	0	0	0	0	1	1	03h
1	1	0	0	1	1	1	1	1	9Fh
2	0	0	1	0	0	1	0	1	25h
3	0	0	0	0	1	1	0	1	0Dh
4	1	0	0	1	1	0	0	1	99h
5	0	1	0	0	1	0	0	1	49h
6	0	1	0	0	0	0	0	1	41h
7	0	0	0	1	1	1	1	1	1Fh
8	0	0	0	0	0	0	0	1	01h
9	0	0	0	0	1	0	0	1	09h

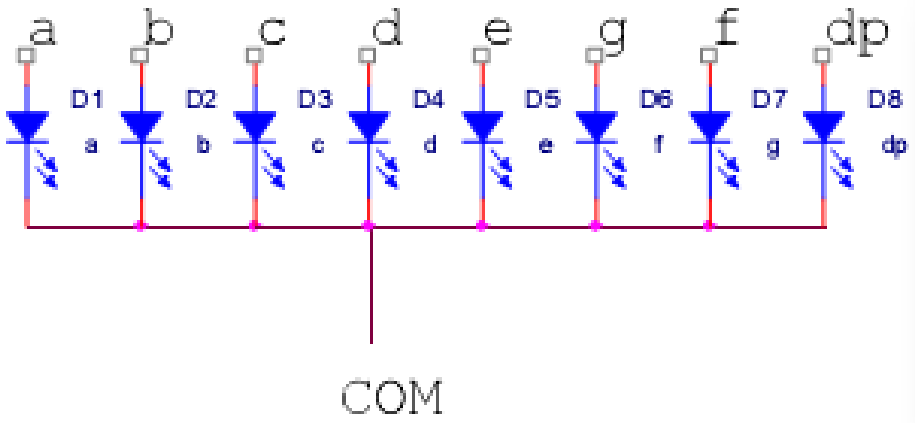
Bảng mã cho Led Anode chung (a là LSB, dp là MSB):

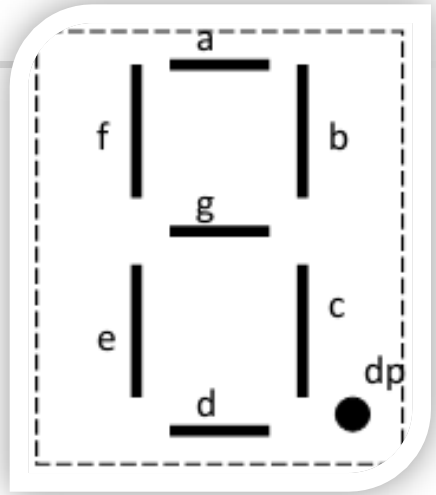
Số	dp	g	f	e	d	c	b	a	Mã hex
0	1	1	0	0	0	0	0	0	0C0h
1	1	1	1	1	1	0	0	1	0F9h
2	1	0	1	0	0	1	0	0	0A4h
3	1	0	1	1	0	0	0	0	0B0h
4	1	0	0	1	1	0	0	1	99h
5	1	0	0	1	0	0	1	0	92h
6	1	0	0	0	0	0	1	0	82h
7	1	1	1	1	1	0	0	0	0F8h
8	1	0	0	0	0	0	0	0	80h
9	1	0	0	1	0	0	0	0	90h

Giao tiếp nút nhấn

```
1.  #INCLUDE<16F877A.H>
2.  #FUSES XT,NOWDT,PUT,NOPROTECT,NOBROWNOUT,NOLVP
3.  #USE DELAY(CLOCK=4000000)
4.  #USE FAST_IO(B)
5.  #BYTE PORTB=0x06
6.  #BIT BUTTON=PORTB.7
7.  #BIT LED=PORTB.6
8.  VOID MAIN()
9.  { SET_TRIS_B(0b10111111);
10.         WHILE(TRUE)
11.         {
12.             IF(BUTTON==0) //KIEM TRA NUT BAM
13.                 {
14.                 LED=1; //BAT DEN
15.                 DELAY_MS(1000);
16.                 LED=0; //TAT DEN
17.                 }
18.         }
19. }
```

LED 7 đoạn-Loại Cathode chung





Mã cho LED Cathode chung(a là MSB, dp là LSB)

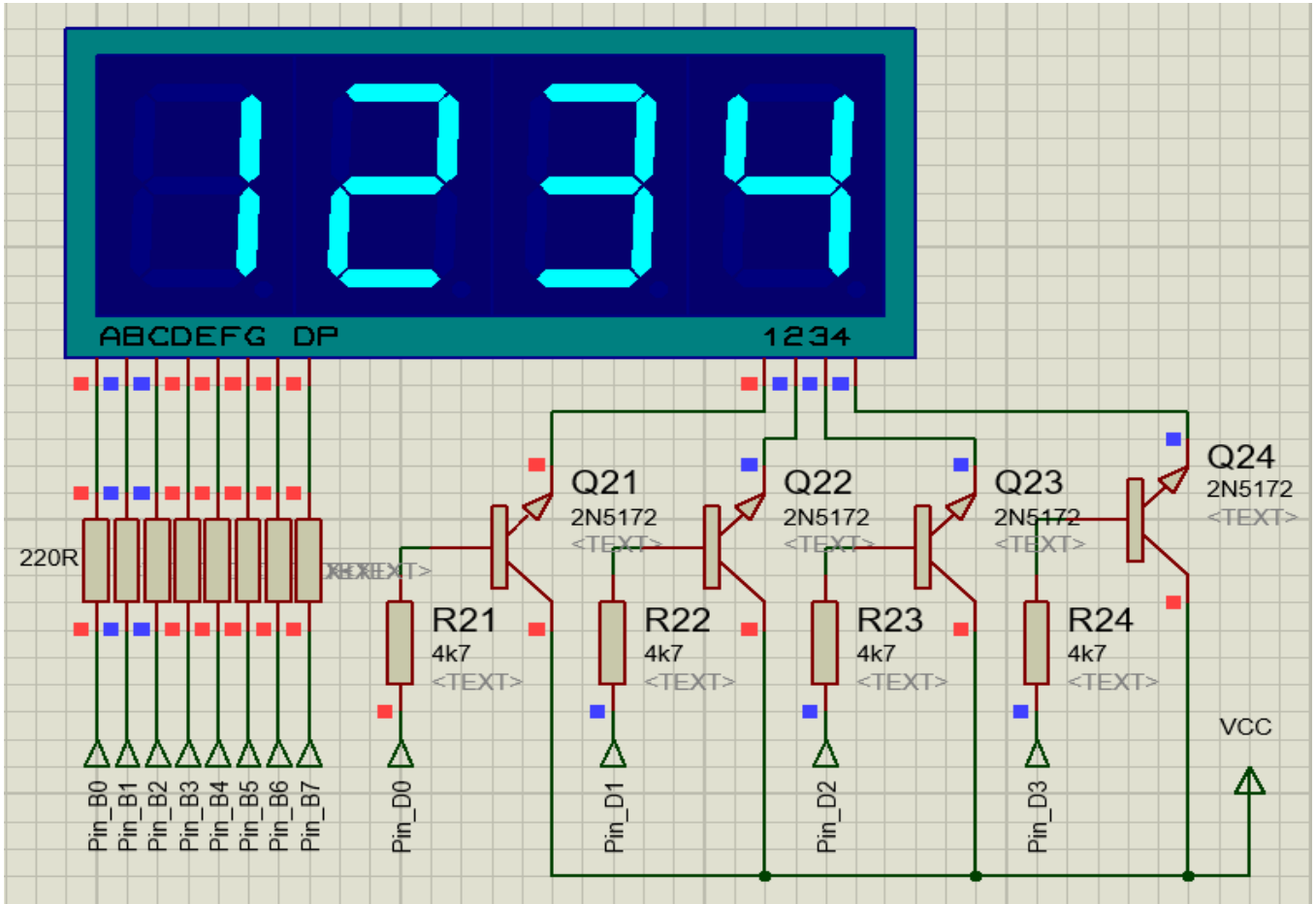
Số	a	b	c	d	e	f	g	dp	Mã hex
0	1	1	1	1	1	1	0	0	0FCh
1	0	1	1	0	0	0	0	0	60h
2	1	1	0	1	1	0	1	0	0DAh
3	1	1	1	1	0	0	1	0	0F2h
4	0	1	1	0	0	1	1	0	66h
5	1	0	1	1	0	1	1	0	0B6h
6	1	0	1	1	1	1	1	0	0BEh
7	1	1	1	0	0	0	0	0	0E0h
8	1	1	1	1	1	1	1	0	0FEh
9	1	1	1	1	0	1	1	0	0F6h

Mã cho LED Cathode chung(a là LSB, dp là MSB)

Số	dp	g	f	e	d	c	b	a	Mã hex
0	0	0	1	1	1	1	1	1	3Fh
1	0	0	0	0	0	1	1	0	06h
2	0	1	0	1	1	0	1	1	5Bh
3	0	1	0	0	1	1	1	1	4Fh
4	0	1	1	0	0	1	1	0	66h
5	0	1	1	0	1	1	0	1	6Dh
6	0	1	1	1	1	1	0	1	7Dh
7	0	0	0	0	0	1	1	1	07h
8	0	1	1	1	1	1	1	1	7Fh
9	0	1	1	0	1	1	1	1	6Fh

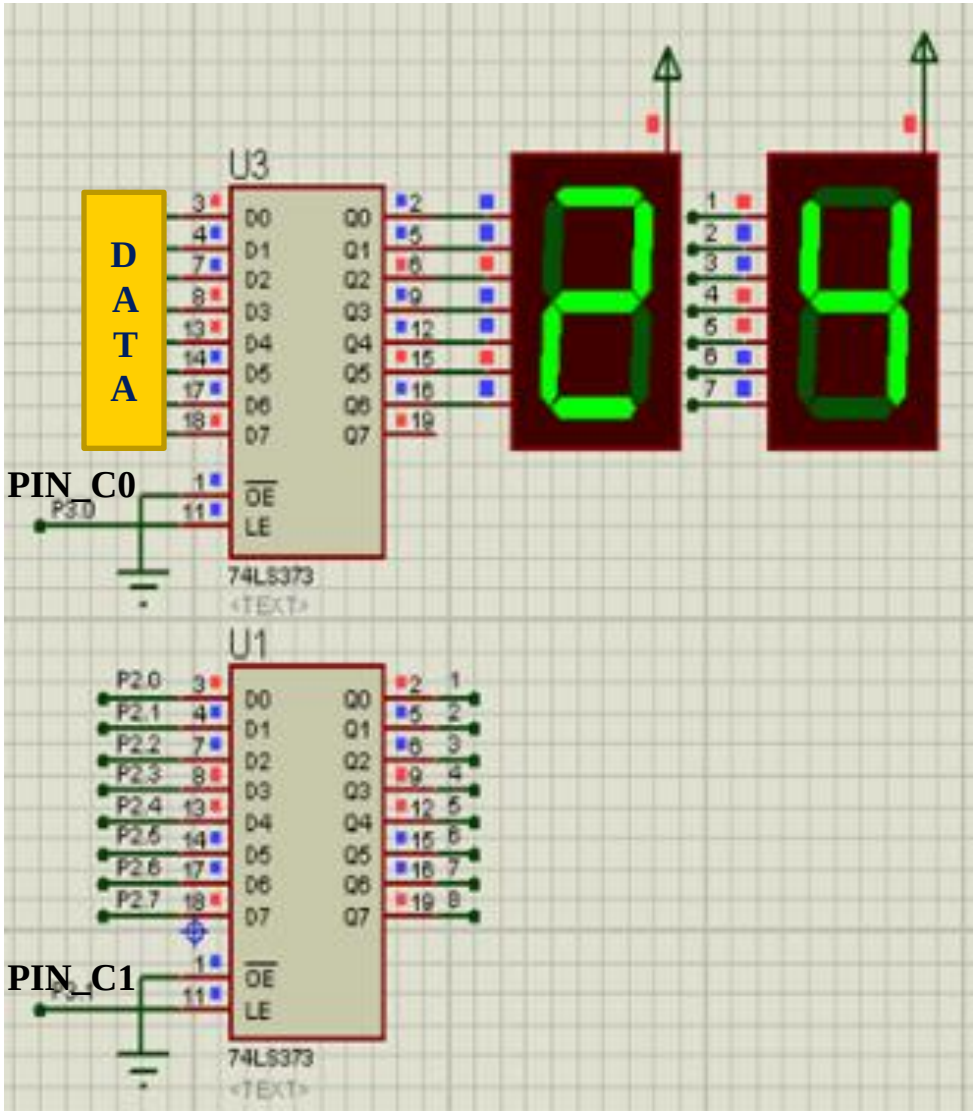
PHƯƠNG PHÁP QUÉT LED

Khi kết nối chung các đường dữ liệu của Led 7 đoạn, các Led không thể sáng đồng thời (do ảnh hưởng lẫn nhau giữa các Led) mà phải thực hiện quét Led, nghĩa là tại mỗi thời điểm chỉ sáng một Led và tắt các Led còn lại. Do hiện tượng lưu ảnh của mắt, ta sẽ thấy các Led sáng đồng thời.



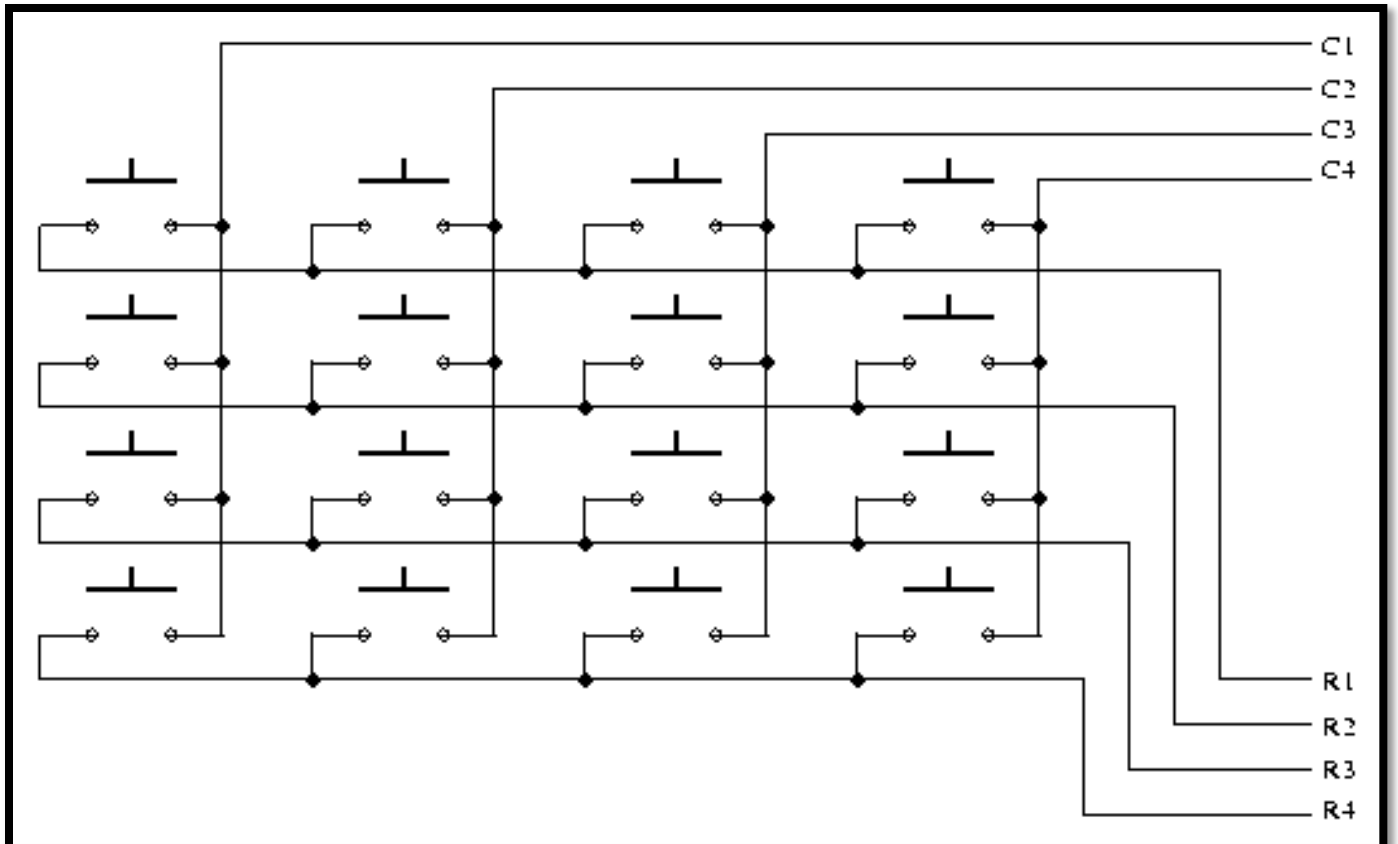
ĐIỀU KHIỂN LED 7 ĐOẠN

- Phương pháp quét
- Phương pháp chốt
  - IC 74LS374,
  - GHI DỊCH 74LS595



Ma trận phím 4x4

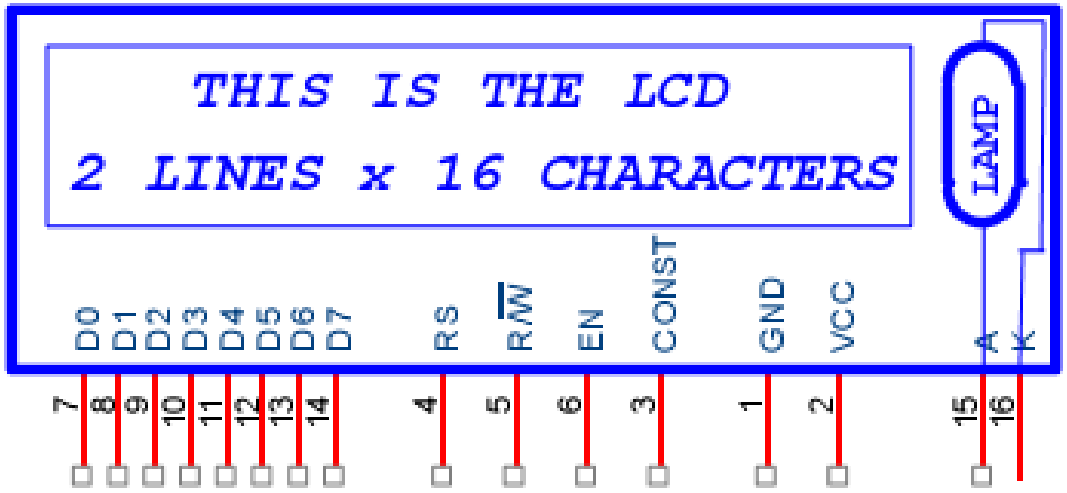
- Quét phím:
  - *Cho các cột =1; cho từng hàng =0.*
  - *Kiểm tra xem cột nào =0 → phím giao của hàng=0 và cột =0 tương ứng đó được nhấn.*



ĐIỀU KHIỂN LCD

- Về phân loại, LCD có nhiều loại :

- 8 character *1 line
- 16 character *1 line (2 line hoặc 4 line)
- 20 character * .....



LCD có 16 chân:

CONST (contrast): chỉnh độ tương phản (độ sáng của hình ảnh trên LCD).

EN (Enable): cho phép đọc/ghi dữ liệu. Trong chế độ đọc, EN tác động bằng xung dương (cạnh lên) và trong chế độ ghi, EN tác động bằng xung âm (cạnh xuống).

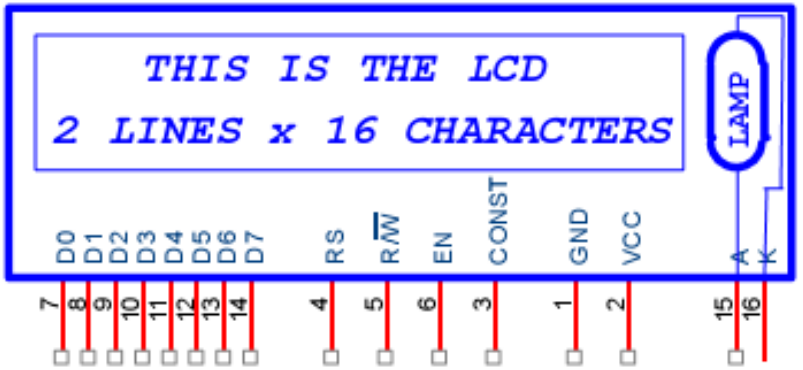
RS (register selection): chọn thanh ghi lệnh (RS = 0) hoặc thanh ghi dữ liệu (RS = 1)

R/W: đọc (R/W = 1) hay ghi (R/W = 0)

D7 – D4: bus dữ liệu (chế độ 8 bit: 4 bit cao, chế độ 4 bit: dùng cho truyền 4 bit cao và 4 bit thấp). Ngoài ra, bit D7 còn dùng làm ngõ ra cho cờ Busy.

D3 – D0: 4 bit thấp trong chế độ 8 bit hay bỏ trống trong chế độ 4 bit.

A, K: anode và cathode đèn nền của LCD.



CÁC HÀM PHỤC VỤ ADC

- Setup_ADC(Mode)
  - Hàm này dùng để thiết lập hoạt động của module ADC, hàm không trả về giá trị.
  - Tham số Mode tuỳ thuộc vào file **.h* có tên tương ứng trên chip đang sử dụng, nằm trong thư mục DEVICES của CCS.
  - VD:

```
Setup_ADC(ADC_CLOCK_INTERNAL);
   //Thời gian lấy mẫu bằng xung clock của
   //vi điều khiển (khoảng 2-6us)
Setup_ADC(ADC_CLOCK_DIV_8);
   //Thời gian lấy mẫu bằng xung clock /8
   //(khoảng 0.8us với thạch anh 40MHz
```

CÁC HÀM PHỤC VỤ ADC

- Setup_ADC_ports(Value)
  - Hàm này xác định chân lấy tín hiệu tương tự.
  - Tuỳ thuộc bố trí chân trên chip, số chân, chân nào dùng cho ADC và số chức năng ADC mỗi chip mà Value có thể có những giá trị khác nhau.
  - Các giá trị của Value có thể tham khảo trong file **.h* có tên tương ứng trên chip đang sử dụng.
  - VD:

```
Setup_ADC_ports(1); //Thiết lập chân A0 nhận tín hiệu tương tự
Setup_ADC_ports(8); //Thiết lập chân A3 nhận tín hiệu tương tự
Setup_ADC_ports(ANALOG_RA3_REF);//Tất cả các chân nhận tín
   //hiệu  tương tự và chân A3 điện áp so sánh
Setup_ADC_ports(RA0_RA1_RA3_ANALOG);
//Chỉ chân A0, A1 và A3 là chân nhận tín hiệu tương tự
```

CÁC HÀM PHỤC VỤ ADC

- Set_ADC_channel(Channel)
  - Chọn chân để đọc vào giá trị tín hiệu tương tự.
  - Hàm không trả về giá trị.
  - Giá trị Channel tuỳ theo số chân chức năng ADC của mỗi vi điều khiển.
  - Với 16F877A Channel có giá trị từ 0-8:

    0-chân A0 1-chân A1 2-chân A2 3-chân A3

    4-chân A4 5-chân A5 6-chân E0 7-chân E1

    8-chân E2
  - **Chú ý**:
    - Delay 10us sau hàm này rồi mới dùng hàm Read_ADC() để đảm bảo kết quả đúng.
    - Hàm chỉ hoạt động khi vi điều khiển có hỗ trợ phần cứng A/D.

CÁC HÀM PHỤC VỤ ADC

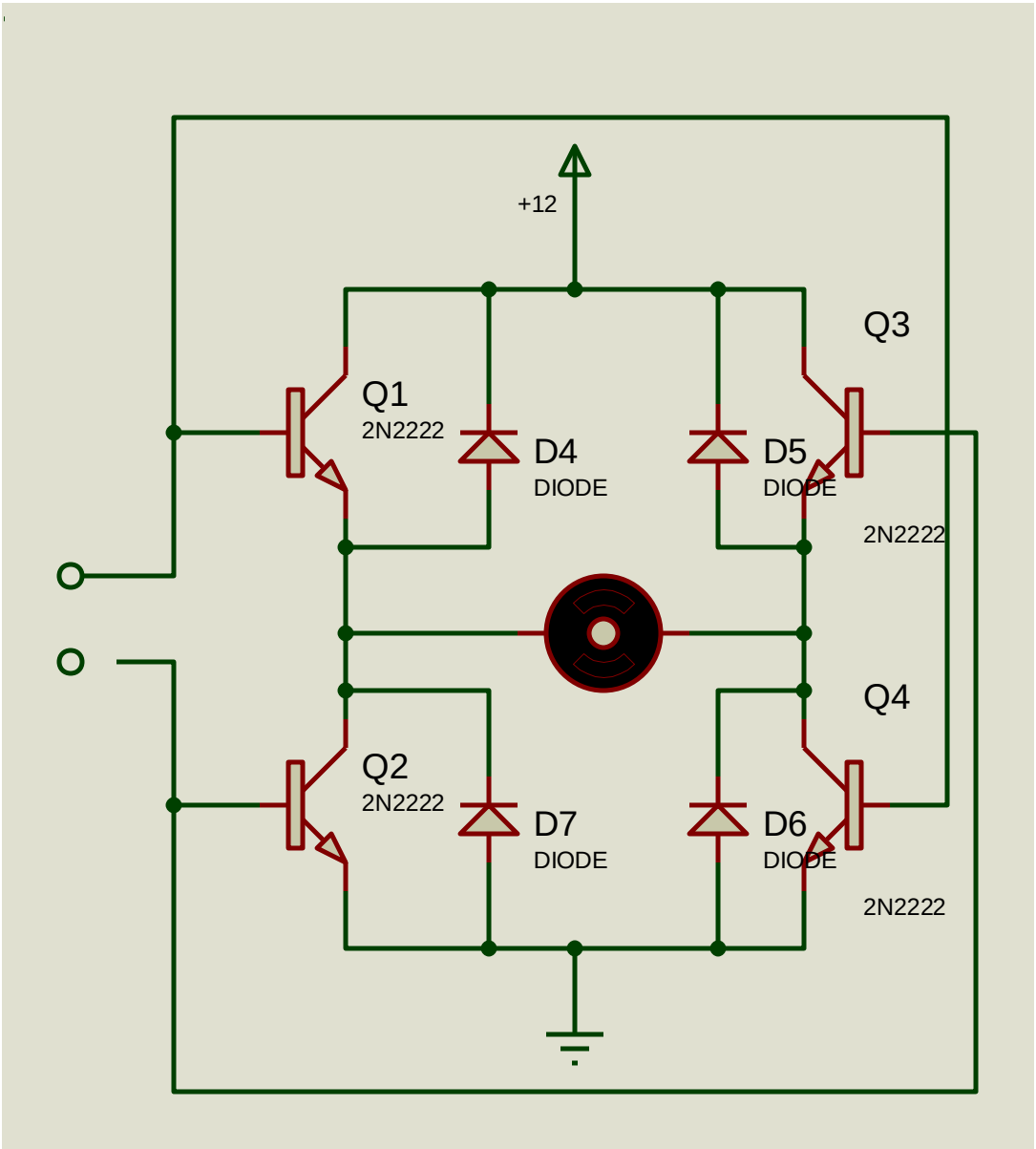
- Value = Read_ADC (Mode)
  - *Mode* là tham số tuỳ chọn. Có thể chọn 1 trong các thông số như sau:
    - ADC_START_AND_READ (Giá trị mặc định)
    - ADC_START_ONLY (Bắt đầu chuyển đổi và trả về giá trị)
    - ADC_READ_ONLY (Đọc kết quả chuyển đổi cuối cùng)
  - Hàm đọc giá trị tín hiệu số theo tín hiệu tương tự vào bộ ADC. Kết quả *Value* trả về có thể là giá trị 8 bit hay 16 bit tuỳ thuộc vào khai báo *#DEVICE ADC= Number* như bảng sau:

#DEVCE	8 bit	10 bit	11 bit	16 bit
ADC=8	00-FF	00-FF	00-FF	00-FF
ADC=10	x	0-3FF	x	x
ADC=11	x	x	0-7FF	x
ADC=16	0-FF00	0-FFC0	0-FFEO	0-FFFF

x: Không xác định

  - Hàm chỉ hoạt động khi vi điều khiển có hỗ trợ phần cứng A/D.

Động cơ DC


+12
Q1
2N2222
D4
DIODE
Q3
D5
DIODE
2N2222
Q2
2N2222
D7
DIODE
Q4
D6
DIODE
2N2222

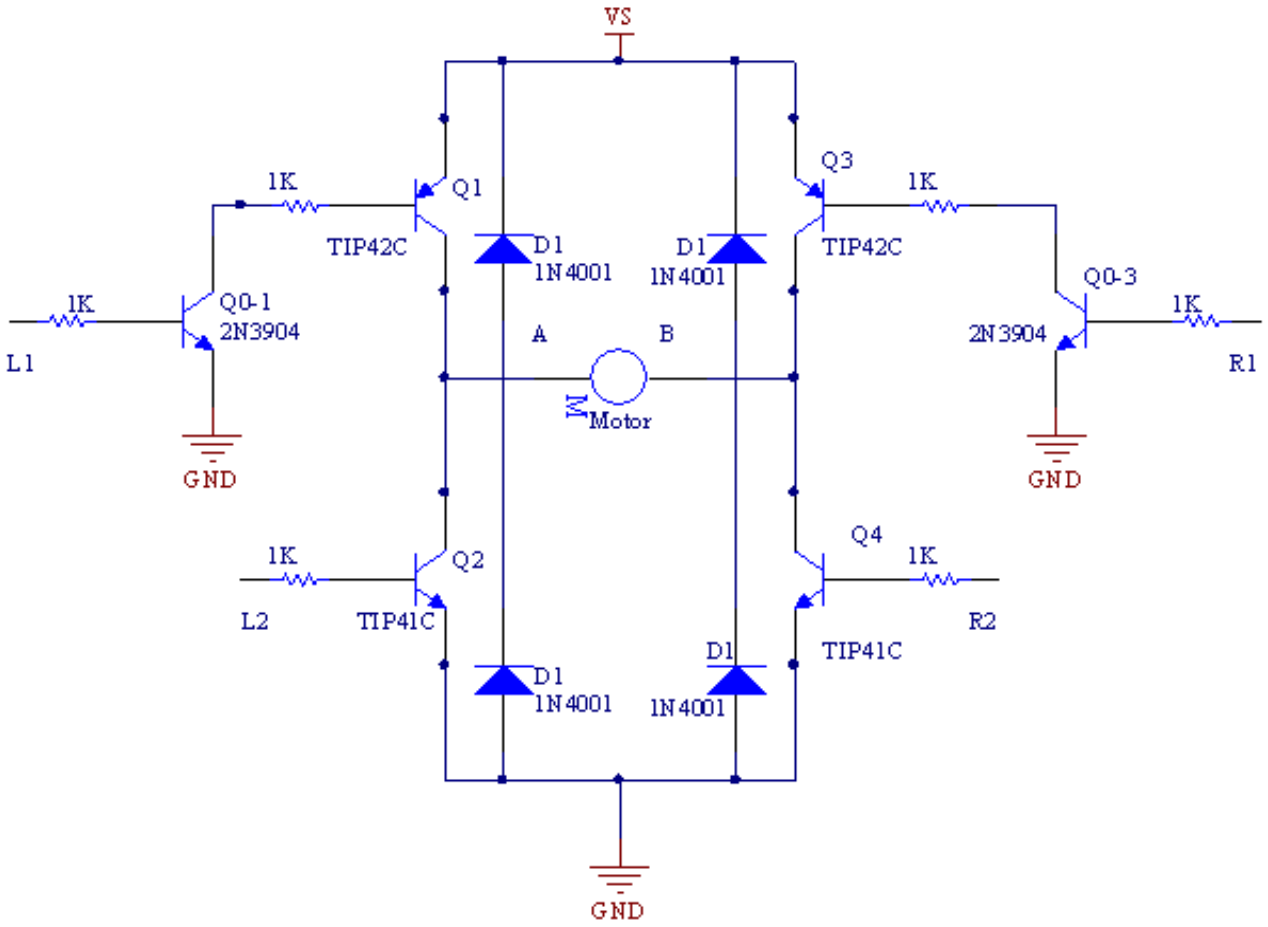
Điều khiển động cơ DC

- Mạch cầu H: được cấu tạo bởi 4 transitor BJT hay là Fet.
- Tác dụng của transitor BJT và Fet là các van đóng mở dẫn dòng điện từ nguồn xuống tải với công suất lớn.
- IC bán dẫn được tích hợp luôn cả cầu H trong đó ta chỉ cần cấp xung điều khiển, có bảo vệ dòng.

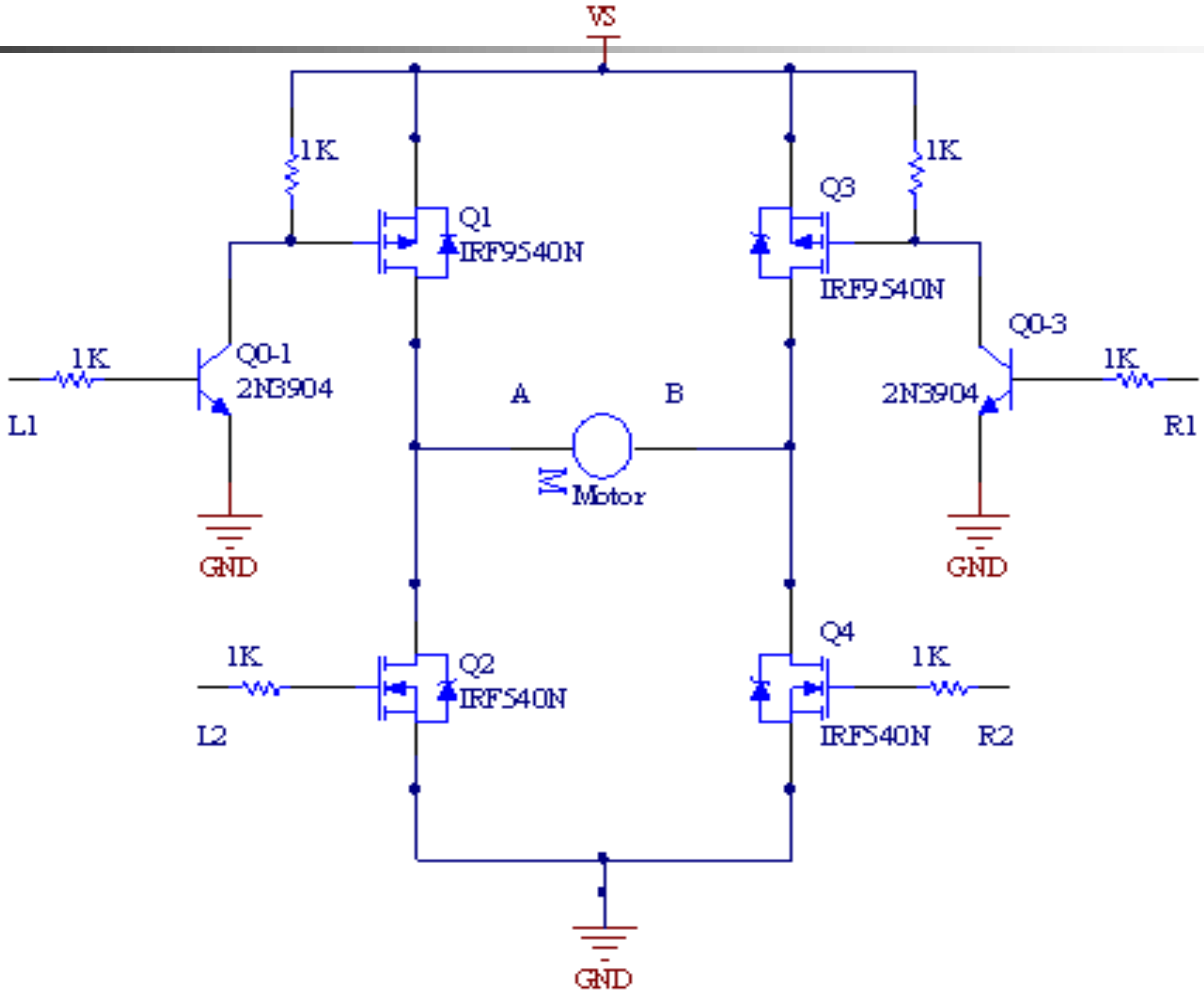
  + L293 : Với điện áp đầu vào là 36V và dòng điện đỉnh qua nó là 1.2A.

  + L298 : Với điện áp đầu vào là 46V và dòng điện đỉnh qua nó là 4A

Điều khiển động cơ DC

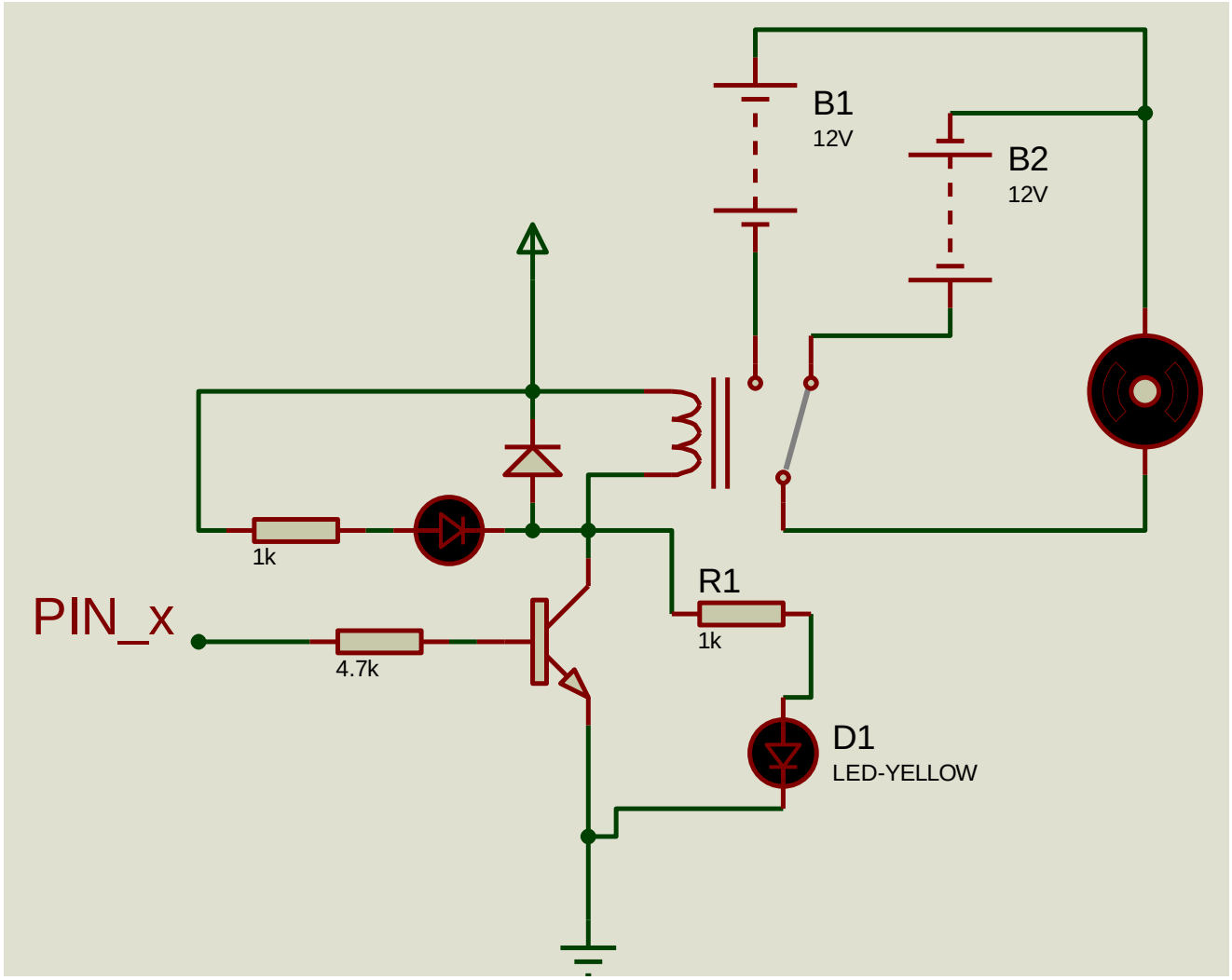


Điều khiển động cơ DC



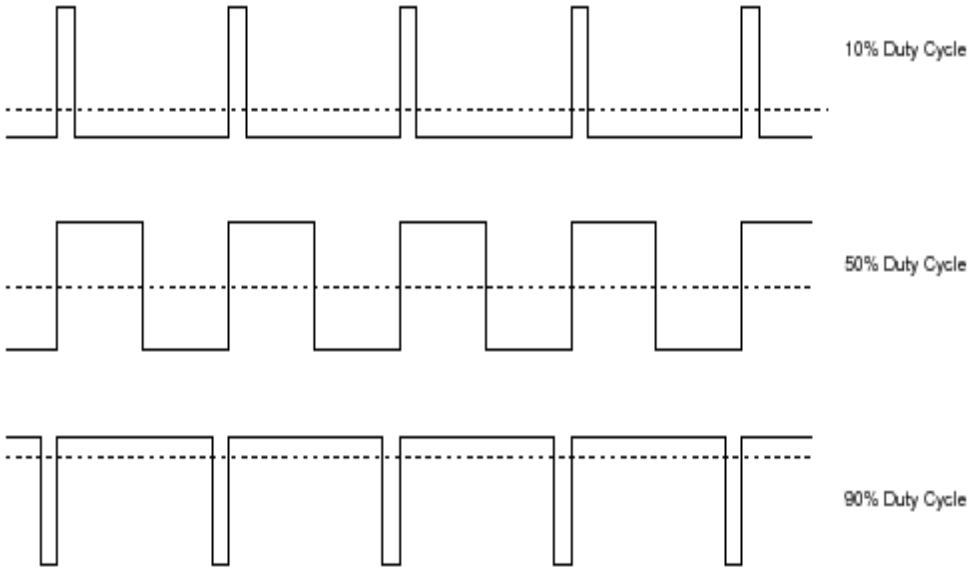
Không dùng diode để tránh dòng ngược vì trong con Fet nó đã có diode rồi.

Điều khiển động cơ DC



PHƯƠNG PHÁP PWM

- Phương pháp PWM là PP thay đổi điện áp trung bình cấp cho phần ứng.



- *PIC16F877A* có hỗ trợ 2 kênh điều xung bằng **phần cứng** ở 2 chân **C1(CCP2)** và **C2(CCP1)** sử dụng **TIMER2**.

MODULE PWM/CAPTURE/COMPARATOR

Các hàm hỗ trợ việc điều xung bằng phần cứng của CCS:

Ghi chú: Chỉ đề cập đến các đối số của các hàm được phục vụ cho việc điều xung PWM.

- setup_timer_2 (mode, period, postscale)
  - *mode:* T2_DIV_BY_1, T2_DIV_BY_4, T2_DIV_BY_16
  - *period:* 0-255
  - *postscale:* 1
- Tần số điều xung PWM:

$$f = fosc / [4*mode*(period+1) * postscale]$$

MODULE PWM/CAPTURE/COMPARATOR

Các hàm hỗ trợ việc điều xung bằng phần cứng của CCS:

- **setup_ccp1(mode) và setup_ccp2(mode)**
  - **mode:** - *CCP_PWM*= chọn chế độ PWM.
    - *CCP_OFF*= tắt chế độ PWM.
- **set_pwm1_duty(value) và set_pwm2_duty(value)**

Nếu value là giá trị kiểu *int 8bit:*

duty_cycle = value / (period+1)

Nếu value là giá trị *long int 16bit*:

duty_cycle = value&1023/[4*(period+1)]

Nếu không cần điều xung quá "mịn" thì nên điều xung ở giá trị value 8bit cho đơn giản.

MODULE PWM/CAPTURE/COMPARATOR

Các hàm hỗ trợ việc điều xung bằng phần cứng của CCS:

Ví dụ: ***Ta muốn điều xung PWM với tần số 10kHz với tần số thạch anh (fosc) sử dụng là 20MHz (value 8bit).***

*f=fosc/[4*mode*(period+1)]*

→ *10000 =20000000/[4*mode*(period+1)]*

→ *mode(period+1) = 500*

Với mode = [1,4,16] và period = 0-255 ta có thể chọn:

mode = 4; period = 124

mode = 16; period = 32

Để cho việc điều xung được "mịn" (chọn được nhiều giá trị duty cycle) ta chọn mode = 4 và period = 124.

Như vậy, để duty_cycle từ 0% đến 100% ta cho value từ 0 đến 125.

value = 30 →duty_cycle = 30 / (124+1) = 0.32 = 32%

value = 63 →duty_cycle = 63 / (124+1) = 0.504 = 50.4%

value = 113 →duty_cycle = 113 / (124+1) = 0.904 = 90.4%

ví dụ: Code:

```
setup_timer_2(T2_DIV_BY_4, 124,1);
setup_ccp1(CCP_PWM);
set_pwm1_duty(30);
```

MODULE PWM/CAPTURE/COMPARATOR

Các hàm hỗ trợ việc điều xung bằng phần cứng của CCS:

MODULE PWM/CAPTURE/COMPARATOR

- Chế độ PWM
  - CCP_PWM Bật chế độ PWM
  - CCP_PWM_PLUS_1
  - CCP_PWM_PLUS_2
  - CCP_PWM_PLUS_3
  - Set_CCPx_duty(Value)
    - *Value*: Biến hay hằng và có thể là số 8bit hay 16bit
    - *x*=0,1,2,…: Tên của chân CCPx
    - Hàm này dung để thiết lập giá trị duty của xung trong chế độ PWM. Hàm này ghi 10bit giá trị vào thanh ghi CCPx. Nếu *Value* chỉ là 8 bit, thì hàm sẽ tự động dịch thêm 2bit để đủ 10bit nạp vào thanh ghi CCPx. Tùy độ phân giải mà giá trị của *Value* không phải lúc nào cũng đạt tới 1023. Do đó, *Value*=512 không có nghĩa là duty=50%.

MODULE PWM/CAPTURE/COMPARATOR

- CCS luôn tạo sẵn các tên danh định C như là các biến con trỏ tới CCP1 và CCP2 là:
  - CCP_1 (16bit),
  - CCP_2 (16bit),
  - CCP_1_HIGH,
  - CCP_1_LOW,
  - CCP_2_HIGH,
  - CCP_2_LOW.
- Không cần khai báo dùng luôn các tên đó để lấy giá trị khi dùng module Capture hay gán giá trị khi dùng Compare.

MODULE PWM/CAPTURE/COMPARATOR

- Setup_CCPx(Mode)
  - Dùng trước tiên để thiết lập chế độ hoạt động hay vô hiệu tính năng *CCPx* = 1, 2, ... tên chân CCP có trên vi điều khiển.
  - *Mode* là 1 trong các hằng số sau: (Các hằng số có thể xem trong file **.h* và tuỳ vi điều khiển)
  - CCP_OFF Tắt chức năng CCP, các chân này sẽ là chân I/O
- Chế độ Capture
  - CCP_CAPTURE_FE Capture khi có cạnh xuống
  - CCP_CAPTURE_RE Capture khi có cạnh lên
  - CCP_CAPTURE_DIV_4 Chỉ Capture sau khi đếm đủ 4 cạnh lên
  - CCP_CAPTURE_DIV_16 Chỉ Capture sau khi đếm đủ 16 cạnh lên
  - Xử dụng để kéo giãn thời gian làm việc của vi điều khiển để dành thời gian cho công việc khác.

MODULE PWM/CAPTURE/COMPARATOR

- Chế độ Compare
  - CCP_COMPARE_SET_ON_MATCH //Xuất xung mức cao khi TMR1=CCPx
  - CCP_COMPARE_CLR_ON_MATCH //Xuất xung mức thấp khi TMR1=CCPx
  - CCP_COMPARE_INT // Ngắt khi TMR1=CCPx

MODULE PWM CHUYÊN DỤNG

- Module PWM chuyên dụng trên được tích hợp sẵn trên một số PIC18 và dsPIC.
- Các vi điều khiển này có từ 6 tới 8 chân PWMx phục vụ cho điều chế độ rộng xung, chuyên dụng cho điều khiển động cơ AC và DC.

MODULE PWM CHUYÊN DỤNG

- Các hàm dùng trong Module PWMx chuyên dụng:
  - set_power_pwm_override(pwm, override, value)
    - Giá trị *pwm* lựa chọn chân PWM cần thiết lập. *pwm* là hằng số từ 0 đến 7
    - *Override* xác định khi nào tín hiệu ra được xác định bởi thanh ghi OVDCONS và khi nào tín hiệu ra được xác định bởi thanh ghi PDC. Khi *Override* là *false*, thanh ghi PDC xác định tín hiệu ra. Khi *Override* là *true*, tín hiệu ra được xác định bởi giá trị lưu trên thanh ghi OVDCONS.
    - Khi *value* là giá trị 1, thì chân PWM được kích hoạt trong chu kỳ tiếp theo. Nếu là 0 thì chân PWM sẽ không được kích hoạt.
    - VD:

```
set_power_pwm_override(1, true, 1); //PWM1 được kích hoạt
set_power_pwm_override(1, false, 0); //PMW1 không được kích hoạt
```

MODULE PWM CHUYÊN DỤNG

- Các hàm dùng trong Module PWMx chuyên dụng:
  - set_power_pwmX_duty(duty)
    - *X* là giá trị 0, 2, 4, hoặc 6
    - *Duty* là số nguyên từ 0 đến 16383.
    - Hàm này lưu giá trị của duty vào trong thanh ghi PDCXL/H. Giá trị của duty là khoảng thời gian mà giá trị đầu ra PWM ở trạng thái kích hoạt.
    - VD:

```
set_power_pwmx_duty(4000);
```

MODULE PWM CHUYÊN DỤNG

- Các hàm dùng trong Module PWMx chuyên dụng:
  - setup_power_pwm_pins(module0,module1,module2,module3)
    - Hàm này xác định chế độ hoạt động cho các cặp của module PWM, các chế độ bao gồm:
      - PWM_PINS_DISABLED: cả 2 chân đều OFF
      - PWM_ODD_ON : chỉ có chân lẻ ON
      - PWM_BOTH_ON :cả 2 chân đều ON
      - PWM_COMPLEMENTARY: ngõ ra 2 chân bù nhau
    - VD:

```
setup_power_pwm_pins(PWM_PINS_DISABLED, PWM_PINS_DISABLED, PWM_PINS_DISABLED, PWM_PINS_DISABLED);
setup_power_pwm_pins(PWM_COMPLEMENTARY, PWM_COMPLEMENTARY, PWM_PINS_DISABLED,PWM_PINS_DISABLED);
```

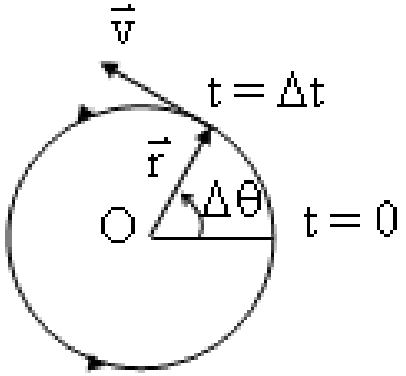
MODULE PWM CHUYÊN DỤNG

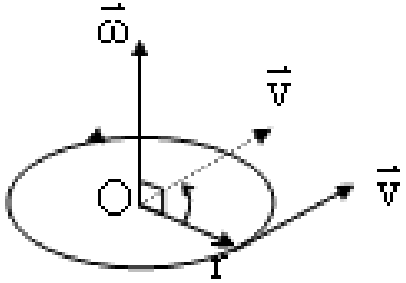
- Các hàm dùng trong Module PWMx chuyên dụng:
  - setup_power_pwm(***modes***, ***postscale***, ***time_base***, ***period***, ***compare***, ***compare_postscale***, ***dead_time***)
    - ***Modes***: PWM_CLOCK_DIV_4, PWM_CLOCK_DIV_16,
      PWM_CLOCK_DIV_64, PWM_CLOCK_DIV_128
      PWM_DISABLED, PWM_FREE_RUN, PWM_SINGLE_SHOT,
      PWM_UP_DOWN, PWM_UP_DOWN_INT
      PWM_OVERRIDE_SYNC
      PWM_UP_TRIGGER,
      PWM_DOWN_TRIGGER
      PWM_UPDATE_DISABLE, PWM_UPDATE_ENABLE
      PWM_DEAD_CLOCK_DIV_2,
      PWM_DEAD_CLOCK_DIV_4,
      PWM_DEAD_CLOCK_DIV_8,
      PWM_DEAD_CLOCK_DIV_16
  - VD:

MODULE PWM CHUYÊN DỤNG

- Các hàm dùng trong Module PWMx chuyên dụng:
  - set_pwmx_duty (value)
    - *Value*: Biến hay hằng và có thể là số 8bit hay 16bit
    - *x*=0,1,2,…: Tên của chân PWMx
    - Hàm này dung để thiết lập giá trị duty của xung trong chế độ PWM. Giá trị duty cycle được xác định như sau:
    - ***duty cycle = value / [4 * (PR2 +1)]***
    - Trong đó: *PR2* là giá trị lớn nhất của Timer2 sẽ đếm từ trước khi chốt giá trị của tín hiệu ra.
    - VD: Với bộ tạo dao động 20 MHz, tần số xung 1.2 kHz, t2DIV = 16, PR2 = 200. Đoạn chương trình sau thiết lập giá trị duty = 50% (hay 416 us).

```
long duty;
duty = 408; // [408/(4*(200+1))]=0.5=50%
set_pwm1_duty(duty);
```





Hình 4 : Vận tốc góc

Giả sử ở thời điểm ban đầu vị trí của chất điểm được xác định bởi góc θ và sau khoảng thời gian Δt vị trí của chất điểm được xác định bởi góc θ+Δθ, cũng tương tự người ta định nghĩa *vận tốc góc tức thời*

$$\omega = \lim_{\Delta t \to 0} \frac{\Delta\theta}{\Delta t} = \frac{d\theta}{dt} \qquad (I.10)$$

Đơn vị đo ω là rad/s.

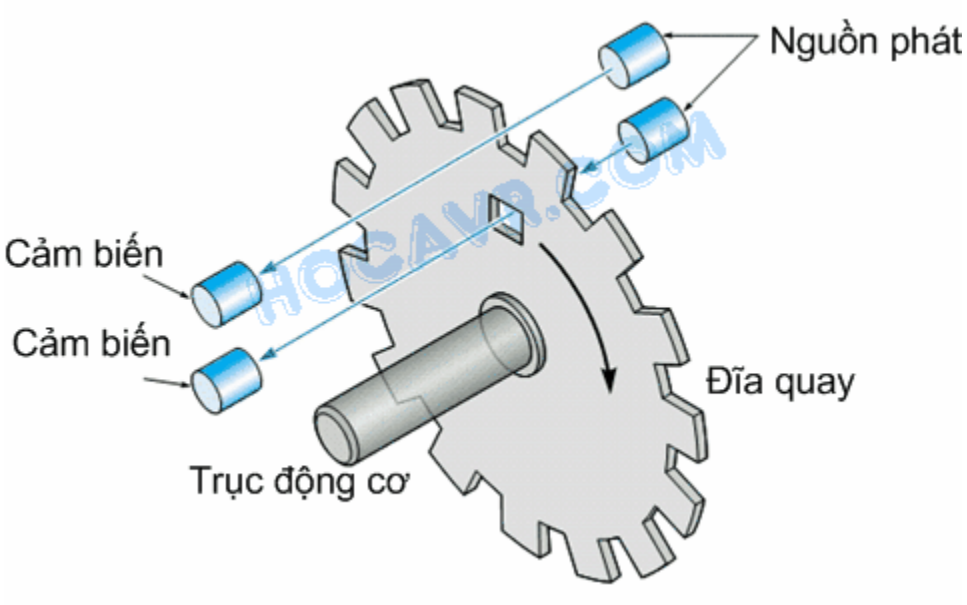
Vận tốc góc $\vec{\omega}$ cũng là một vectơ, phương và chiều của nó được xác định bởi biểu thức sau :

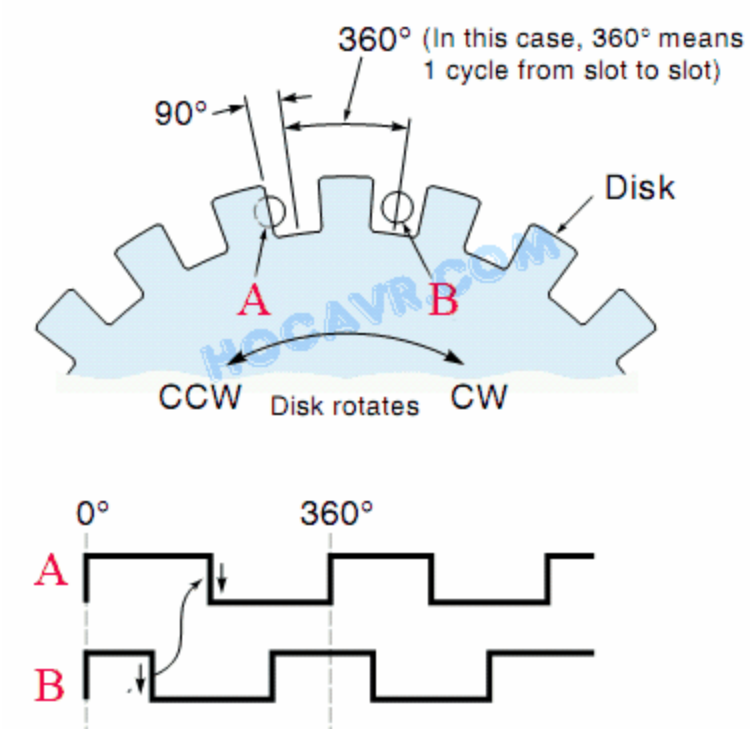
$$\vec{\omega} = \frac{(\vec{r} \times \vec{v})}{r^2} \qquad (I.11)$$

trong đó $(\vec{r} \times \vec{v})$ là ký hiệu của tích vectơ (tích hữu hướng) của hai vectơ $\vec{r}$ và $\vec{v}$.

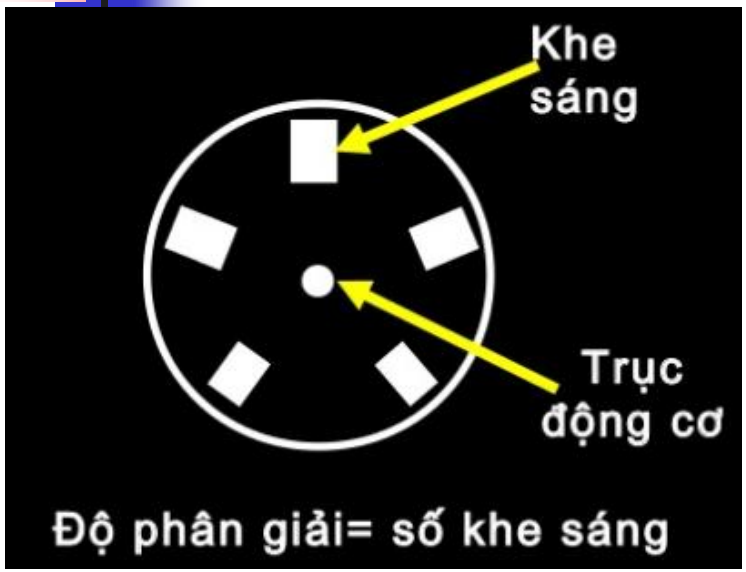
ENCODER

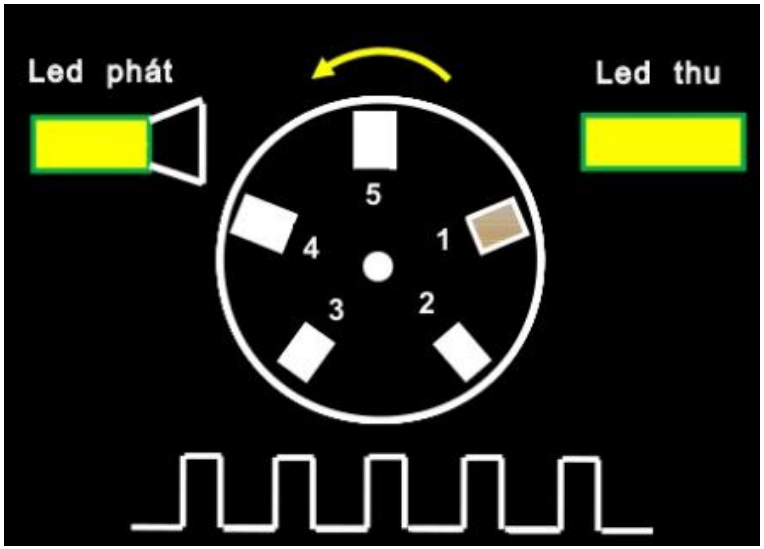
- Encoder thường có 3 kênh (3 ngõ ra) bao gồm kênh A, kênh B và kênh I (Index)





Phương pháp 1





Thời gian: 10s

Số xung đo được: 10 xung

Độ phân giải: 5xung/vòng

=> Số vòng quay: 2

Công thức tính vận tốc

$$N = 60 \frac{X}{Y} \quad \text{(Vòng/phút)}$$

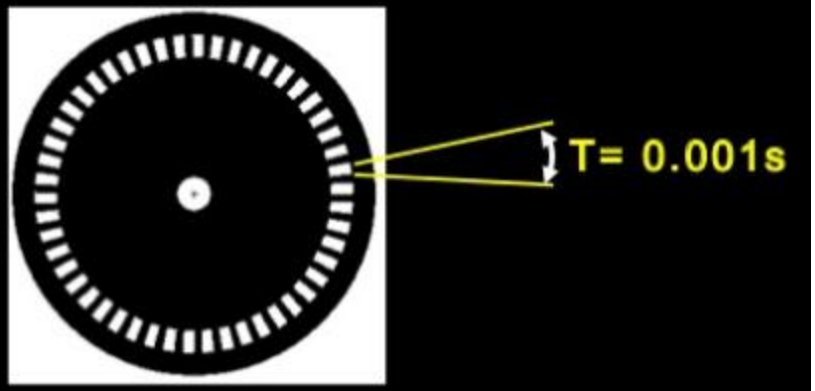
X: *Số xung đếm trong 1s*

Y: *độ phân giải encoder (xung/vòng)*

Phương pháp 2

VD: 1 xung ,T= 1ms =0.001s

Độ phân giải : 100 xung/vòng



1 vòng quay hết : 0.001s x 100=0.1s

Công thức tính vận tốc

$$N = \frac{60}{T \times đpg} \text{ (Vòng/phút)}$$

T: *thời gian 1 xung (giây)*

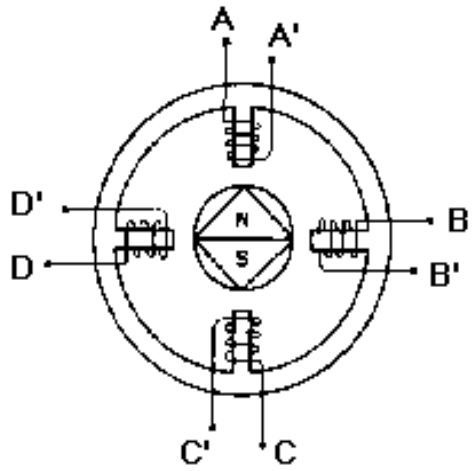
Đpg: *độ phân giải encoder (xung/vòng)*

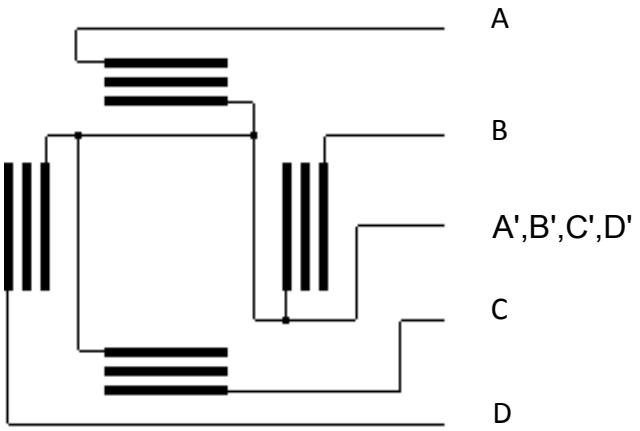
	Phương pháp 1	Phương pháp 2
Đặc điểm	Sử dụng bộ đếm số xung trong 1s	Xác định thời gian 1 chu kỳ t
Hiển thị	Dễ quan sát	Giá trị thay đổi liên tục
Độ chính xác	Không cao	cao

- Trên PIC có chân ngắt ngoài, bạn có thể dùng nó để đọc xung từ encoder rồi dùng timer để suy ra vận tốc của động cơ.

  Nếu như dùng timer thì bạn có thể set timer0 chế độ counter rồi đếm xung encoder, chân còn lại encoder nối vô chân IO bất kì để biết chiều quay của động cơ. Sau đó dùng timer1 set khoảng thời gian định trước rồi đọc số xung do timer0 đếm được => vận tốc động cơ.

Động cơ bước





loại 2 pha , 6 dây (4 cuộn dây , 2 common)

Hoạt động của động cơ bước:

- Góc bước G. $$G = \frac{N_r * N_s}{N_r - N_s}$$
- *Nr*: số răng của rotor động cơ bước
- *Ns*: số răng của stator động cơ bước
- Tuỳ thuộc vào nhà sản xuất góc bước thường có giá trị:

$$G = 0{,}72^0;\ 0{,}9^0;\ 1{,}8^0;\ 3{,}6^0;\ 7{,}5^0;\ 15^0;\ 30^0$$

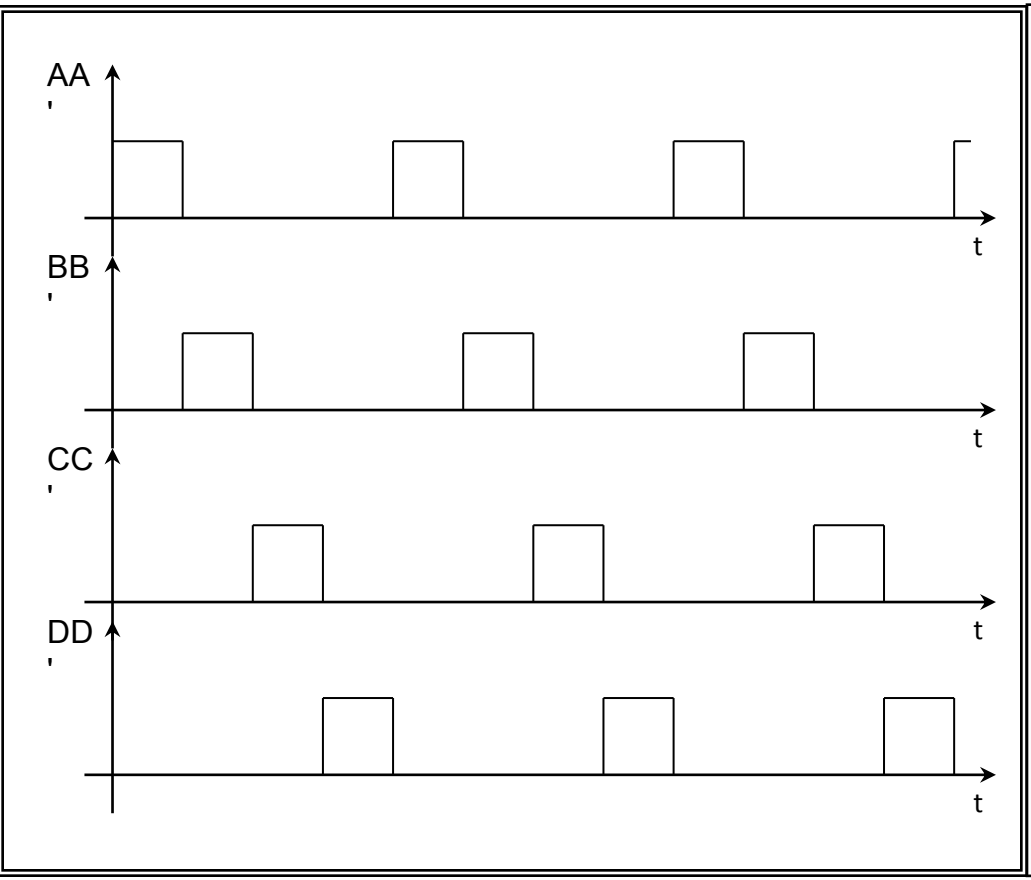
- Góc bước càng nhỏ thì độ phân giải của động cơ bước càng cao,tức động cơ quay càng đỡ bị giật,giá thành của động cơ bước càng cao.
- Số bước là số xung điều khiển cần thiết để Rotor động cơ quay được một vòng biểu thị bằng thông số S :

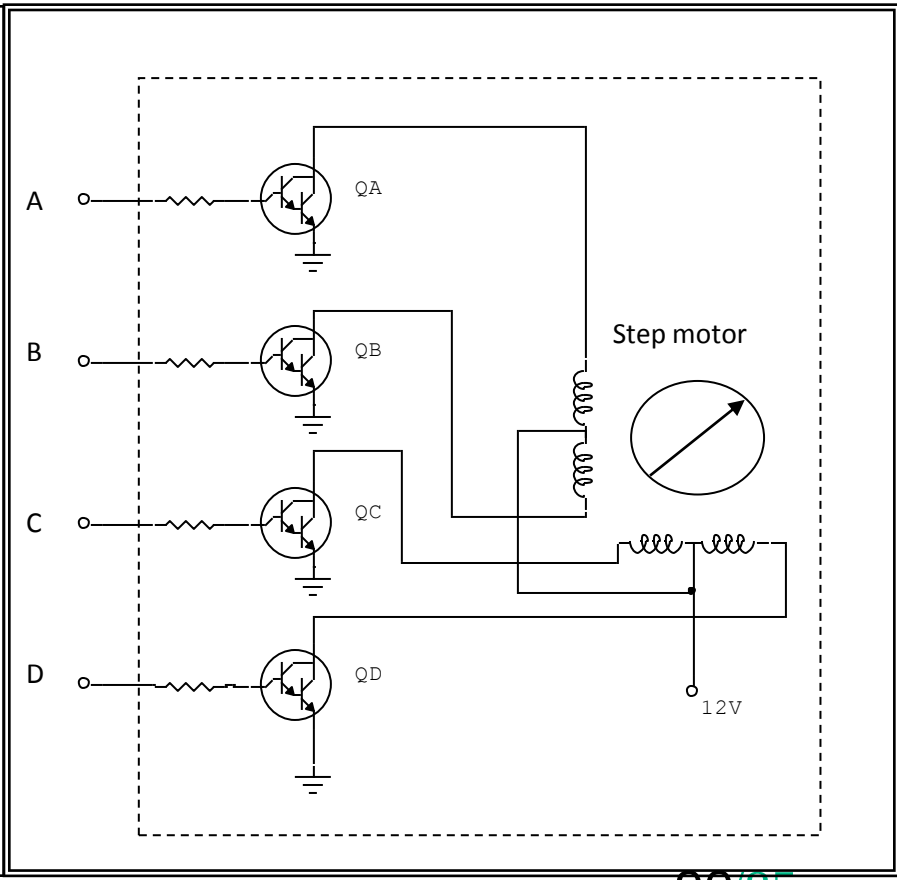
$$S = \frac{360}{G}$$

Điều khiển động cơ bước

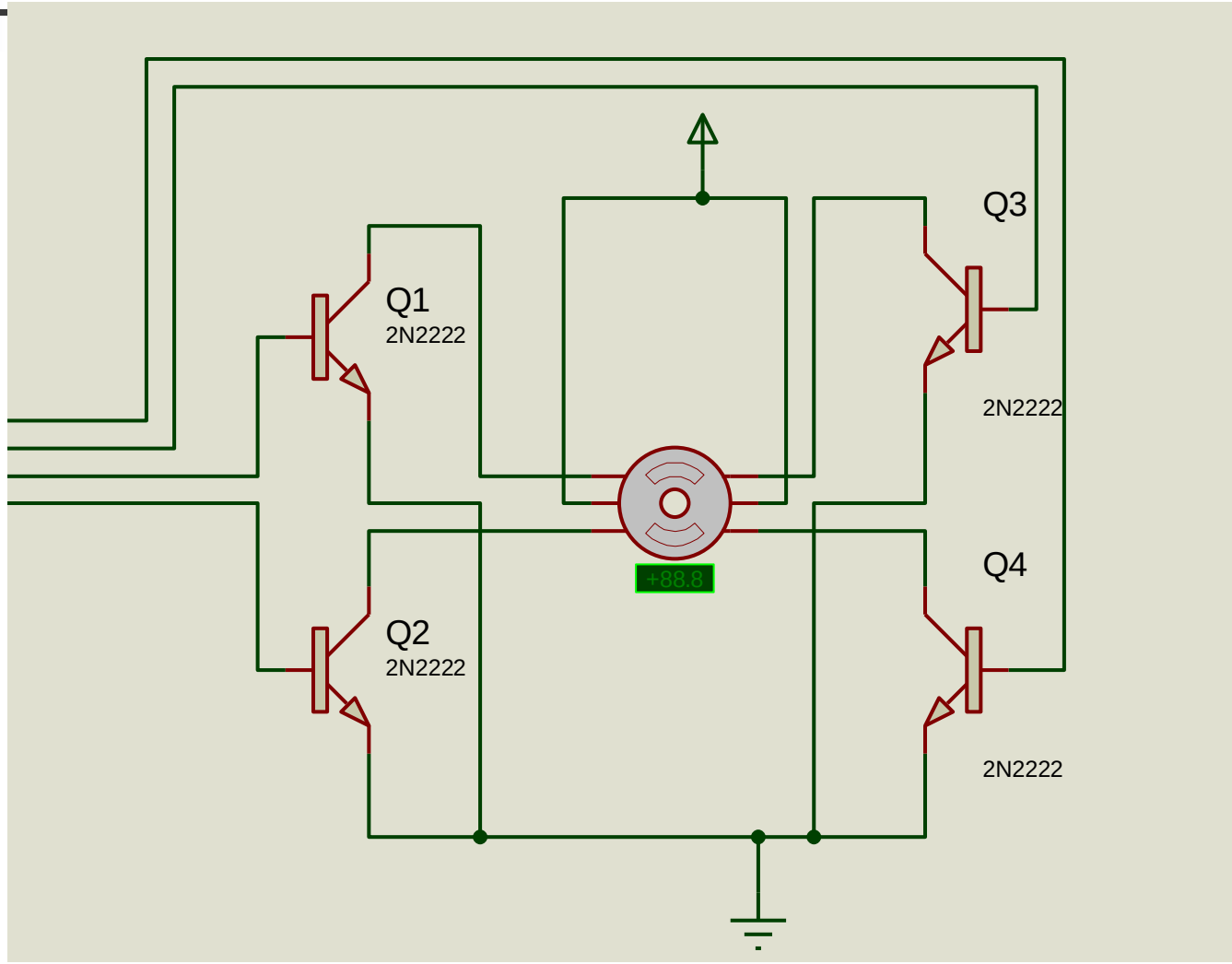
- "ABCD"= "0001" → "0010" → "0100" → "1000" → "0001"

→ "ABCD" thực hiện lệnh dịch 4 lần, động cơ quay 1 vòng.





Điều khiển động cơ bước



loại 2 pha , 6 dây (4 cuộn dây , 2 common)

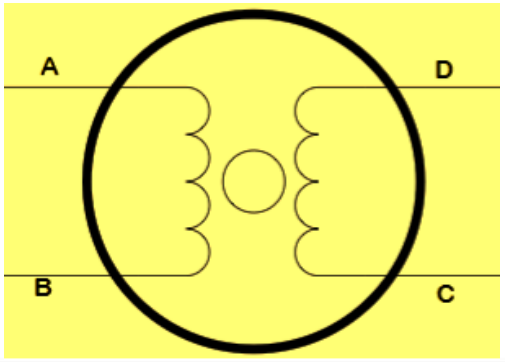
Bảng 4.5 - Điều khiển một bước

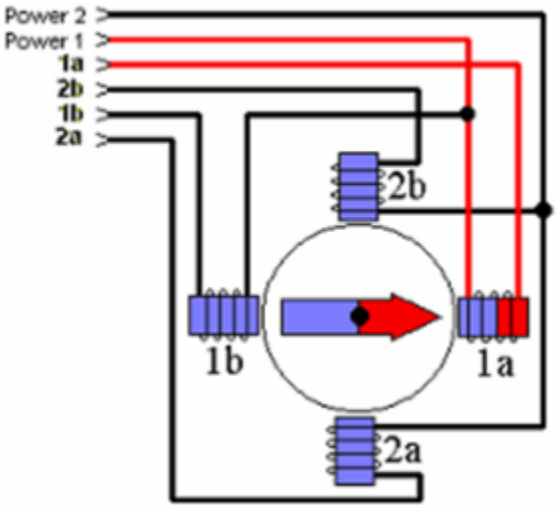
Ngược				Thuận			
1	**2**	**3**	**4**	**1**	**2**	**3**	**4**
1	0	0	0	1	0	0	0
0	1	0	0	0	0	0	1
0	0	1	0	0	0	1	0
0	0	0	1	0	1	0	0
1	0	0	0	1	0	0	0

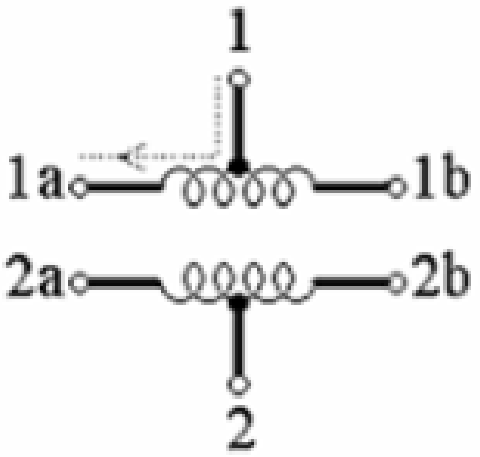
Bảng 4.6 - Điều khiển nửa bước

Ngược				Thuận			
1	**2**	**3**	**4**	**1**	**2**	**3**	**4**
1	0	0	1	1	0	0	1
1	0	0	0	0	0	0	1
1	1	0	0	0	0	1	1
0	1	0	0	0	0	1	0
0	1	1	0	0	1	1	0

Xung điều khiển				Tên gọi	Mô tả
1a	2a	1b	2b		
1	0	0	0	Đủ bước, một pha	Công suất thấp, trong một thời điểm chỉ có một pha được cấp điện.
0	1	0	0		
0	0	1	0		
0	0	0	1		
1	1	0	0	Đủ bước, hai pha	Mô-men quay lớn, trong một thời điểm có 2 pha được cấp điện.
0	1	1	0		
0	0	1	1		
1	0	0	1		
1	0	0	0	Nửa bước	Chỉ quay nửa bước mỗi xung, nhưng môn-men quay không đồng bộ (lúc lớn lúc nhỏ). Cấp xung kiểu này làm cho động cơ "bước" nhỏ đi nhưng làm giảm sự liên tục của động cơ, và có thể làm động cơ mất bước.
1	1	0	0		
0	1	0	0		
0	1	1	0		
0	0	1	0		
0	0	1	1		
0	0	0	1		
1	0	0	1		

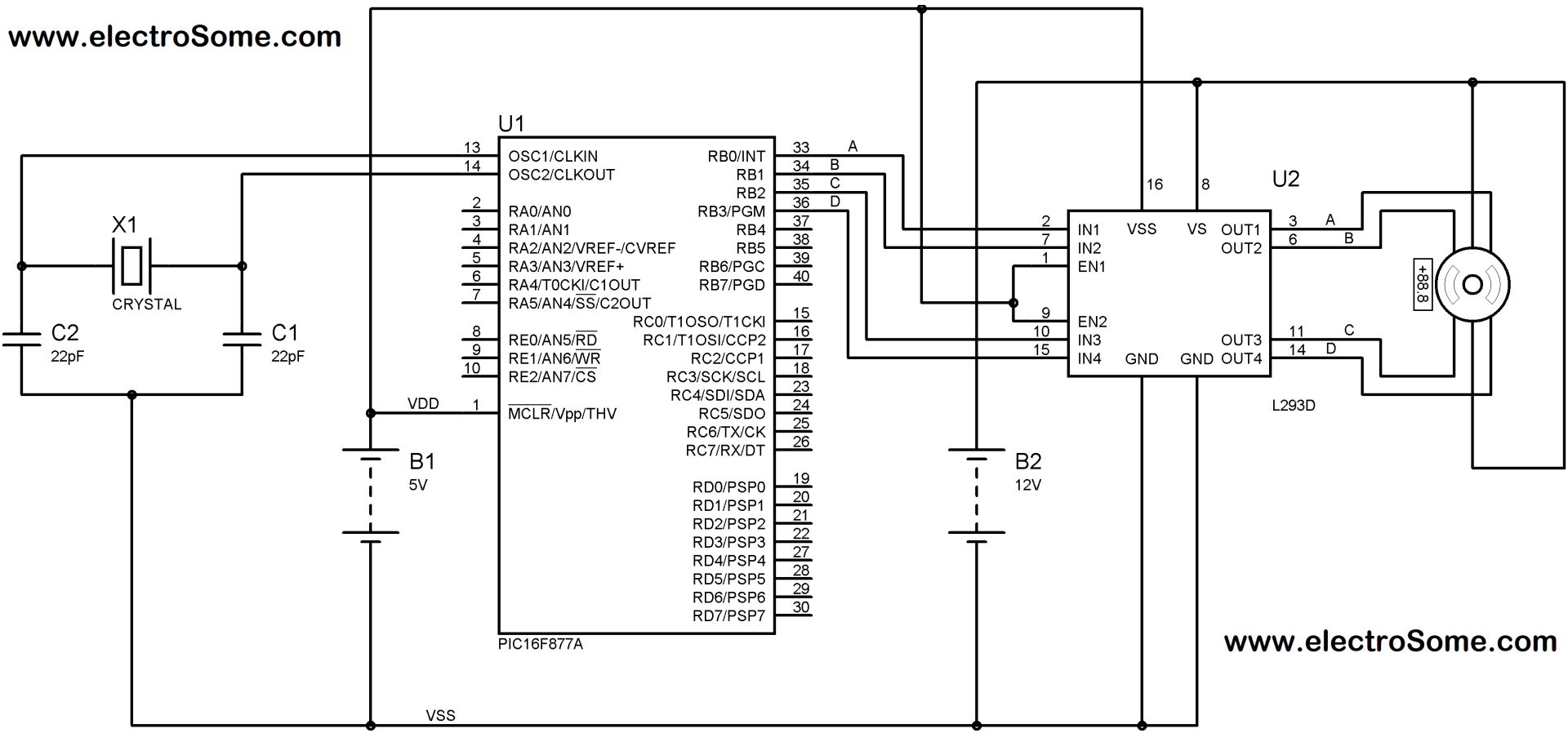






A B C D

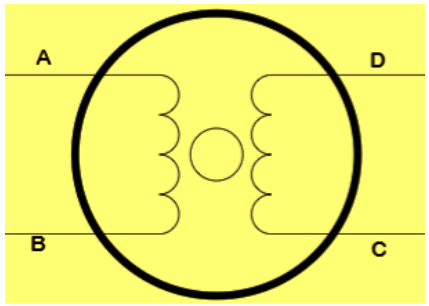
Giao tiếp với L293D



Giao tiếp với L298

Full Drive Stepping Sequence				
Step	A	B	C	D
1	1	1	0	0
2	0	1	1	0
3	0	0	1	1
4	1	0	0	1

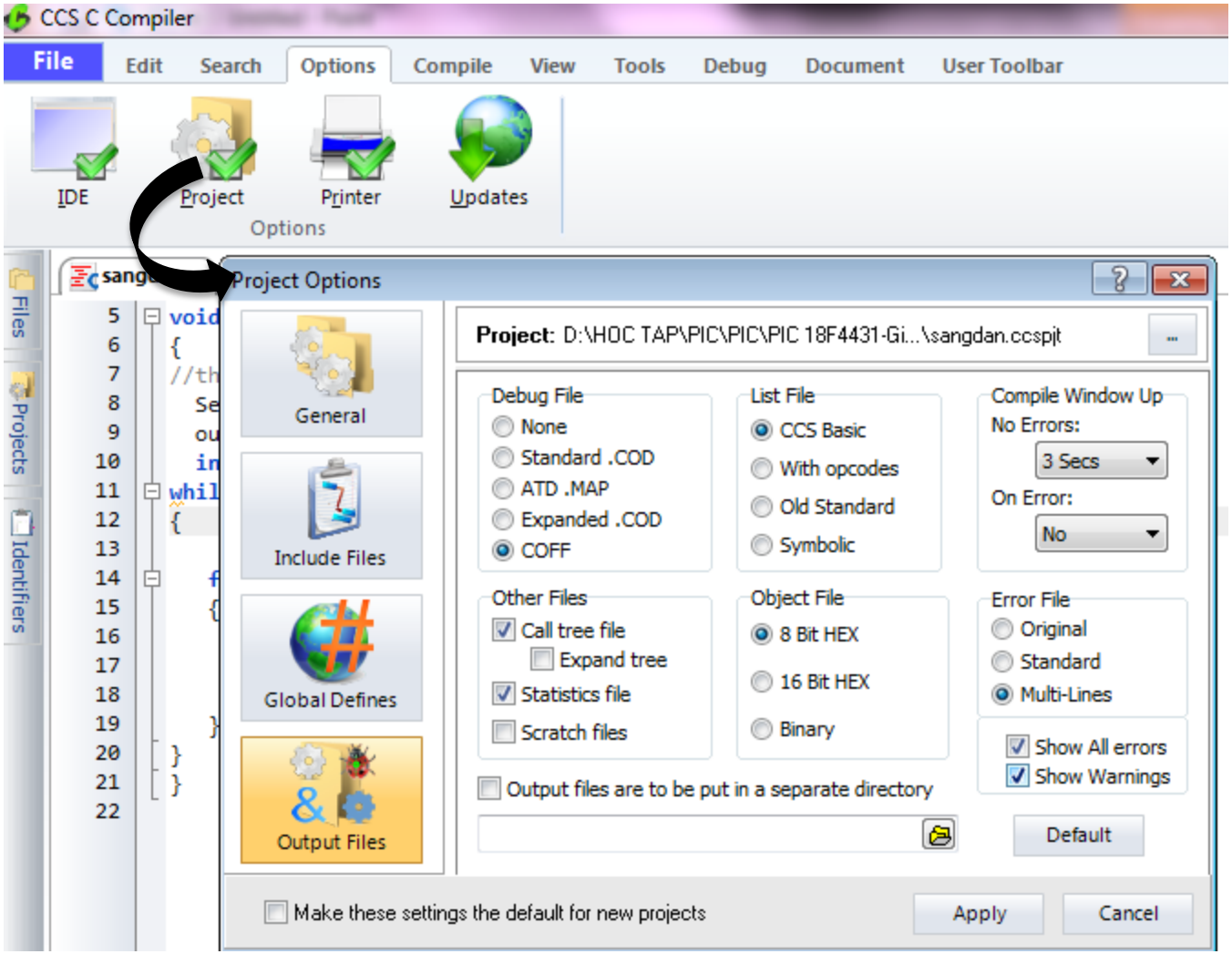
Half Drive Stepping Sequence				
Step	A	B	C	D
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1



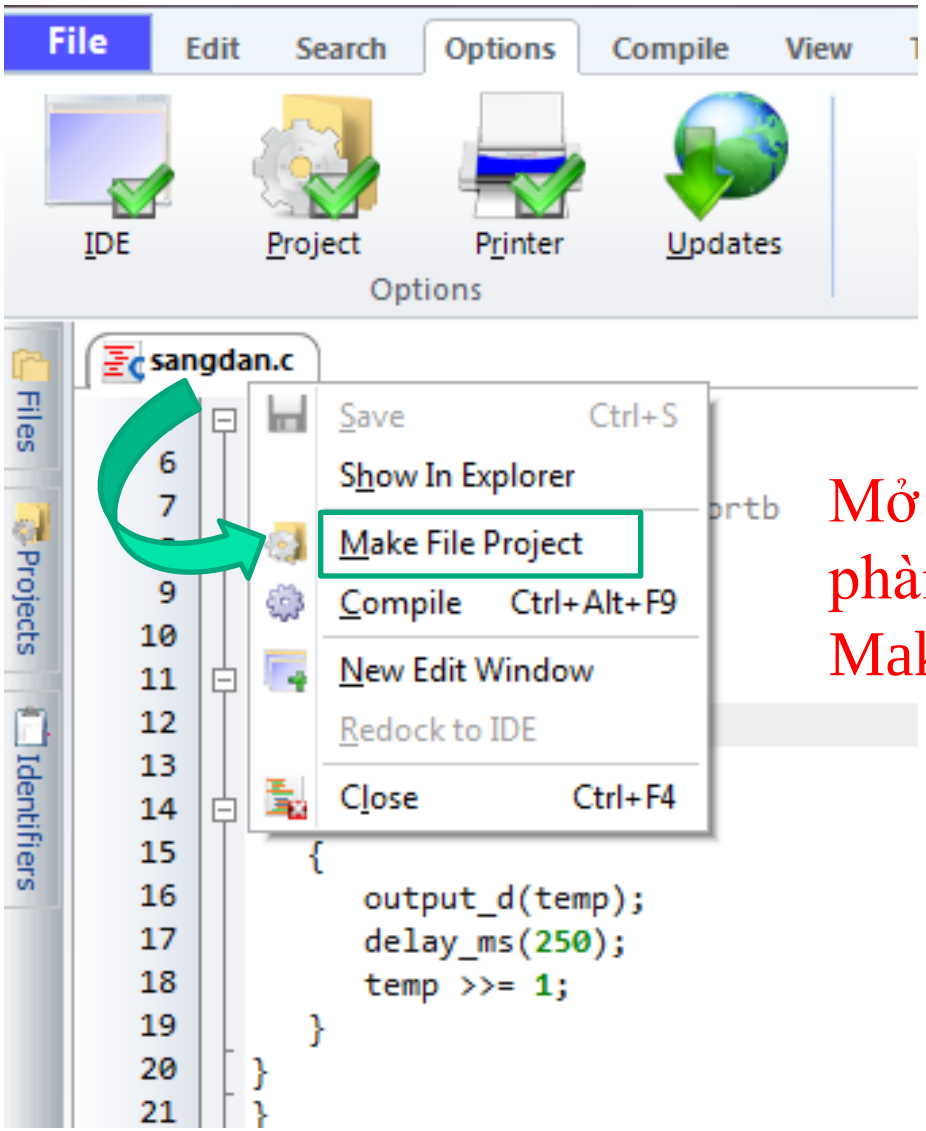
CÁC HÀM PHỤC VỤ CÁC MODULE KHÁC

- Ngoài những hàm được trình bày chi tiết như trên, CCS còn hỗ trợ rất nhiều các hàm khác như:
  - giao tiếp nối tiếp RS232;
  - giao tiếp LCD;
  - giao tiếp I2C;
  - …
- Tất cả các hàm này được trình bày chi tiết trong phần Help của phần mềm CCS.

Tạo đường dẫn file Hex



Tạo nhanh Project từ code.c



Mở một file.c sau đó click phải trên tab tên file, chọn Make file project

TABLE 5-1: SPECIAL FUNCTION REGISTER MAP FOR PIC18F2331/2431/4331/4431 DEVICES

Address	Name	Address	Name	Address	Name	Address	Name	Address	Name
FFFh	TOSU	FDFh	INDF2	FBFh	CCPR1H	F9Fh	IPR1	F7Fh	PTCON0
FFEh	TOSH	FDEh	POSTINC2	FBEh	CCPR1L	F9Eh	PIR1	F7Eh	PTCON1
FFDh	TOSL	FDDh	POSTDEC2	FBDh	CCP1CON	F9Dh	PIE1	F7Dh	PTMRL
FFCh	STKPTR	FDCh	PREINC2	FBCh	CCPR2H	F9Ch	—	F7Ch	PTMRH
FFBh	PCLATU	FDBh	PLUSW2	FBBh	CCPR2L	F9Bh	OSCTUNE	F7Bh	PTPERL
FFAh	PCLATH	FDAh	FSR2H	FBAh	CCP2CON	F9Ah	ADCON3	F7Ah	PTPERH
FF9h	PCL	FD9h	FSR2L	FB9h	ANSEL1	F99h	ADCHS	F79h	PDC0L
FF8h	TBLPTRU	FD8h	STATUS	FB8h	ANSEL0	F98h	—	F78h	PDC0H
FF7h	TBLPTRH	FD7h	TMR0H	FB7h	T5CON	F97h	—	F77h	PDC1L
FF6h	TBLPTRL	FD6h	TMR0L	FB6h	QEICON	F96h	TRISE	F76h	PDC1H
FF5h	TABLAT	FD5h	T0CON	FB5h	—	F95h	TRISD	F75h	PDC2L
FF4h	PRODH	FD4h	—	FB4h	—	F94h	TRISC	F74h	PDC2H
FF3h	PRODL	FD3h	OSCCON	FB3h	—	F93h	TRISB	F73h	PDC3L
FF2h	INTCON	FD2h	LVDCON	FB2h	—	F92h	TRISA	F72h	PDC3H
FF1h	INTCON2	FD1h	WDTCON	FB1h	—	F91h	PR5H	F71h	SEVTCMPL
FF0h	INTCON3	FD0h	RCON	FB0h	SPBRGH	F90h	PR5L	F70h	SEVTCMPH
FEFh	INDF0	FCFh	TMR1H	FAFh	SPBRG	F8Fh	—	F6Fh	PWMCON0
FEEh	POSTINC0	FCEh	TMR1L	FAEh	RCREG	F8Eh	—	F6Eh	PWMCON1
FEDh	POSTDEC0	FCDh	T1CON	FADh	TXREG	F8Dh	LATE	F6Dh	DTCON
FECh	PREINC0	FCCh	TMR2	FACh	TXSTA	F8Ch	LATD	F6Ch	FLTCONFIG
FEBh	PLUSW0	FCBh	PR2	FABh	RCSTA	F8Bh	LATC	F6Bh	OVDCOND
FEAh	FSR0H	FCAh	T2CON	FAAh	BAUDCTL	F8Ah	LATB	F6Ah	OVDCONS
FE9h	FSR0L	FC9h	SSPBUF	FA9h	EEADR	F89h	LATA	F69h	CAP1BUFH
FE8h	WREG	FC8h	SSPADD	FA8h	EEDATA	F88h	TMR5H	F68h	CAP1BUFL
FE7h	INDF1	FC7h	SSPSTAT	FA7h	EECON2	F87h	TMR5L	F67h	CAP2BUFH
FE6h	POSTINC1	FC6h	SSPCON	FA6h	EECON1	F86h	—	F66h	CAP2BUFL
FE5h	POSTDEC1	FC5h	—	FA5h	IPR3	F85h	—	F65h	CAP3BUFH
FE4h	PREINC1	FC4h	ADRESH	FA4h	PIR3	F84h	PORTE	F64h	CAP3BUFL
FE3h	PLUSW1	FC3h	ADRESL	FA3h	PIE3	F83h	PORTD	F63h	CAP1CON
FE2h	FSR1H	FC2h	ADCON0	FA2h	IPR2	F82h	PORTC	F62h	CAP2CON
FE1h	FSR1L	FC1h	ADCON1	FA1h	PIR2	F81h	PORTB	F61h	CAP3CON
FE0h	BSR	FC0h	ADCON2	FA0h	PIE2	F80h	PORTA	F60h	DFLTCON