GIÁO TRÌNH
VI ĐIỀU KHIỂN
AVR

Mục Lục

BÀI 1 : GIỚI THIỆU VỀ VI ĐIỀU KHIỂN AVR

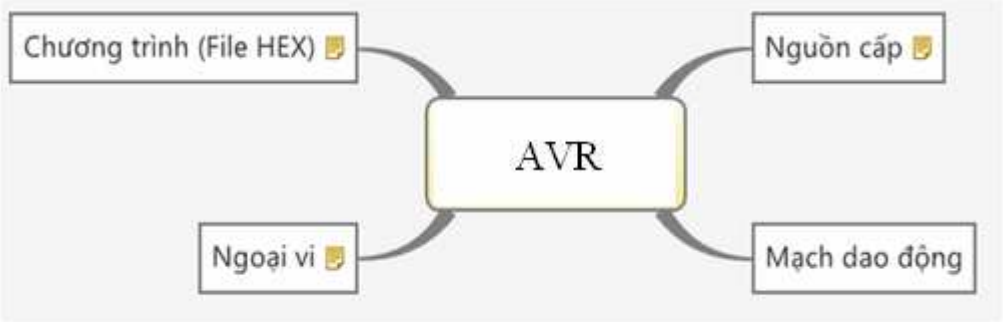
- *Giới thiệu chung về vi điều khiển.*
- *Giới thiệu về vi điều khiển Atmega32.*
- *Lập trình cho Atmega32.*

1. Giới thiệu về vi điều khiển

Khái niệm vi điều khiển (microcontroller – MC) đã khá quen thuộc với các sinh viên CNTT, điện tử, điều khiển tự động cũng như Cơ điện tử… Nó là một trong những IC thích hợp nhất để thay thế các IC số trong việc thiết kế mạch logic. Ngày nay đã có những MC tích hợp đủ tất cả các chức năng của mạch logic. Nói như vậy không có nghĩa là các IC số cũng như các IC mạch số lập trình được khác như PLC… không cần dùng nữa. MC cũng có những hạn chế mà rõ ràng nhất là tốc độ chậm hơn các mạch logic… MC cũng là một máy tính – máy tính nhúng vì nó có đầy đủ chức năng của một máy tính. Có CPU, bộ nhớ chương trình, bộ nhớ dữ liệu, có I/O và các bus trao đổi dữ liệu.

Cần phân biệt khái niệm MC với khái niệm vi xử lý (microprocessor – MP) như 8088 chẳng hạn. MP chỉ là CPU mà không có các thành phần khác như bộ nhớ I/O, bộ nhớ. Muốn sử dụng MP cần thêm các chức năng này, lúc này người ta gọi nó là hệ vi xử lý (microprocessor system). Do đặc điểm này nên nếu để lựa chọn giữa MC và MP trong một mạch điện tử nào đó thì tất nhiên người ta sẽ chọn MC vì nó sẽ rẻ tiền hơn nhiều do đã tích hợp các chức năng khác vào trong chip.

Vậy để một vi điều khiển chạy được thì cần những điều kiện gì :



- Thứ nhất là nguồn cấp, nguồn cấp là cái đầu tiên, cơ bản nhất trong các mạch điện tử, và vấn đề về nguồn là 1 trong những vấn đề rất đau đầu. Không có nguồn thì không thể gọi là 1 mạch điện được. Nguồn cấp cho vi điều khiển là nguồn 1 chiều.

- Thứ hai là mạch dao động, mạch dao động để làm gì ? Giả sử các bạn lập trình cho con AVR : đến thời điểm A làm 1 công việc gì đó, thế thì nó lấy cái gì để xác định được thời điểm nào là thời điểm A ? Đó chính là mạch dao động. Ví dụ như mọi người đều thống nhất vào một giờ chuẩn để làm việc. Cả hệ thống vi điều khiển cũng vậy, cả hệ thống khi đó đều lấy xung nhịp clock – xung nhịp mạch dao động làm xung nhịp chuẩn để hoạt động.
- Thứ ba là ngoại vi, ngoại vi ở đây là các thiết bị để giao tiếp với vi điều khiển để thực hiện 1 nhiệm vụ nào đó mà vi điều khiển đưa ra. Ví dụ như các bạn muốn điều khiển động cơ 1 chiều, nhưng vì vi điều khiển chỉ đưa ra các mức điện áp 0-5V, và dòng điều khiển cỡ mấy chục mA, với nguồn cấp này thì ko thể nối trực tiếp động cơ vào vi điều khiển để điều khiển, mà phải qua 1 thiết bị khác gọi là ngoại vi, chính xác hơn ở đây là driver, người ta dùng driver để có thể điều khiển được các dòng điện lớn từ các nguồn điện nhỏ. Các bàn phím, công tắc… là các ngoại vi.

- Thứ 4 là chương trình, ở đây là file .hex để nạp cho vi điều khiển, chương trình chính là thuật toán mà bạn triển khai thành các câu lệnh rồi biên dịch thành mã hex để nạp vào vi điều khiển.

Các công cụ để học AVR :

- Ngôn ngữ lập trình : C, ASM…
- Phần mềm lập trình : IAR, CodeVisionAVR…
- Mạch nạp : STK200/300/500, Burn-E…
- Mạch phát triển : Board trắng, phần mềm mô phỏng, kit…

2. Giới thiệu về vi điều khiển AVR

AVR là họ vi điều khiển 8 bit theo công nghệ mới, với những tính năng rất mạnh được tích hợp trong chip của hãng Atmel theo công nghệ RISC, nó mạnh ngang hàng với các họ vi điều khiển 8 bit khác như PIC, PSoC. Do ra đời muộn hơn nên họ vi điều khiển AVR có nhiều tính năng mới đáp ứng tối đa nhu cầu của người sử dụng, so với họ 8051, 89xx sẽ có độ ổn định, khả năng tích hợp, sự mềm dẻo trong việc lập trình và rất tiện lợi.

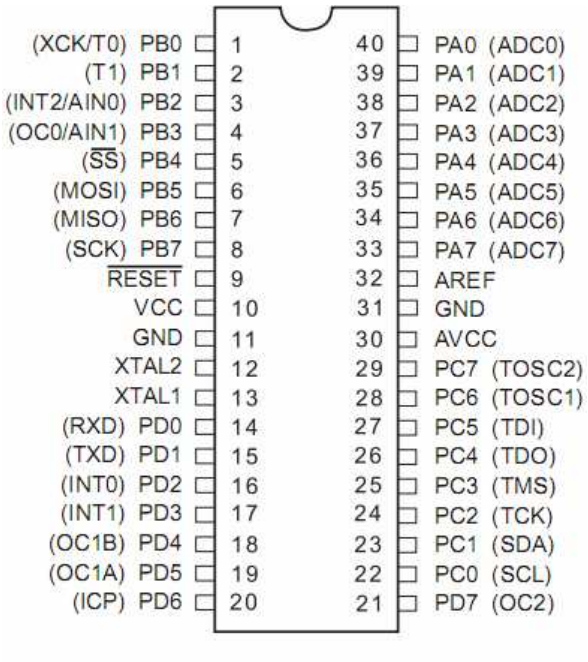
Các tính năng mới của họ AVR:

- ✓ Giao diện SPI đồng bộ.
- ✓ Các đường dẫn vào/ra (I/O) lập trình được.
- ✓ Giao tiếp I2C.
- ✓ Bộ biến đổi ADC 10 bit.
- ✓ Các kênh băm xung PWM.
- ✓ Các chế độ tiết kiệm năng lượng như sleep, stand by..vv.
- ✓ Một bộ định thời Watchdog.
- ✓ 3 bộ Timer/Counter 8 bit.
- ✓ 1 bộ Timer/Counter 16 bit.
- ✓ 1 bộ so sánh analog.
- ✓ Bộ nhớ EEPROM.
- ✓ Giao tiếp USART..vv.

Atmelga32 có đầy đủ tính năng của họ AVR, về giá thành so với các loại khác thì giá thành là vừa phải khi nghiên cứu và làm các công việc ứng dụng tới vi điều khiển. Tính năng :

- ✓ Bộ nhớ 32KB Flash có khả năng đọc, ghi 10000 lần
- ✓ 1024 byte EEPROM có khả năng đọc, ghi 100000 lần.
- ✓ 2KB SRAM.
- ✓ 8 kênh đầu vào ADC 10 bit.
- ✓ Đóng vỏ 40 chân , trong đó có 32 chân vào ra dữ liệu chia làm 4 PORT A,B,C,D. Các chân này đều có chế độ pull_up resistors.

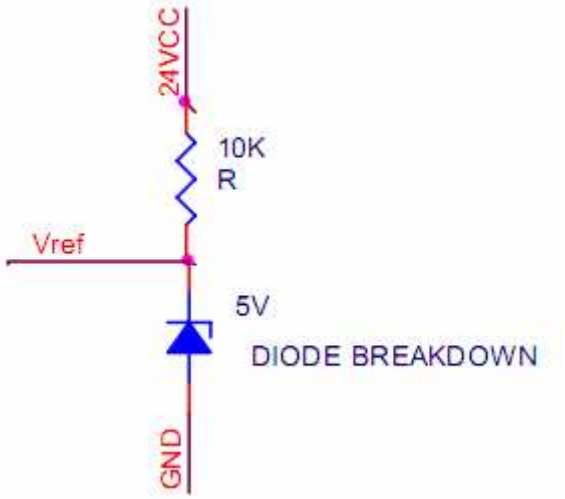
- ✓ Hỗ trợ các giao tiếp UART, SPI, I2C.
- ✓ 1 bộ so sánh analog, 4 kênh PWM.
- ✓ 2 bộ timer/counter 8 bit, 1 bộ timer/counter1 16 bit.
- ✓ 1 bộ định thời Watchdog.



Sơ đồ chân Atmega32

Mô tả chức năng các chân của atmega32

- Vcc và GND 2 chân cấp nguồn cho vi điều khiển hoạt động.
- Reset đây là chân reset cứng khởi động lại mọi hoạt động của hệ thống.
- 2 chân XTAL1, XTAL2 các chân tạo bộ dao động ngoài cho vi điều khiển, các chân này được nối với thạch anh (hay sử dụng loại 4M), tụ gốm (22p).
- Chân Vref thường nối lên 5v(Vcc), nhưng khi sử dụng bộ ADC thì chân này được sử dụng làm điện thế so sánh, khi đó chân này phải cấp cho nó điện áp cố định, có thể sử dụng diode zener:



- Chân Avcc thường được nối lên Vcc nhưng khi sử dụng bộ ADC thì chân này được nối qua 1 cuộn cảm lên Vcc với mục đích ổn định điện áp cho bộ biến đổi.

3. Lập trình cho AVR

Giới thiệu

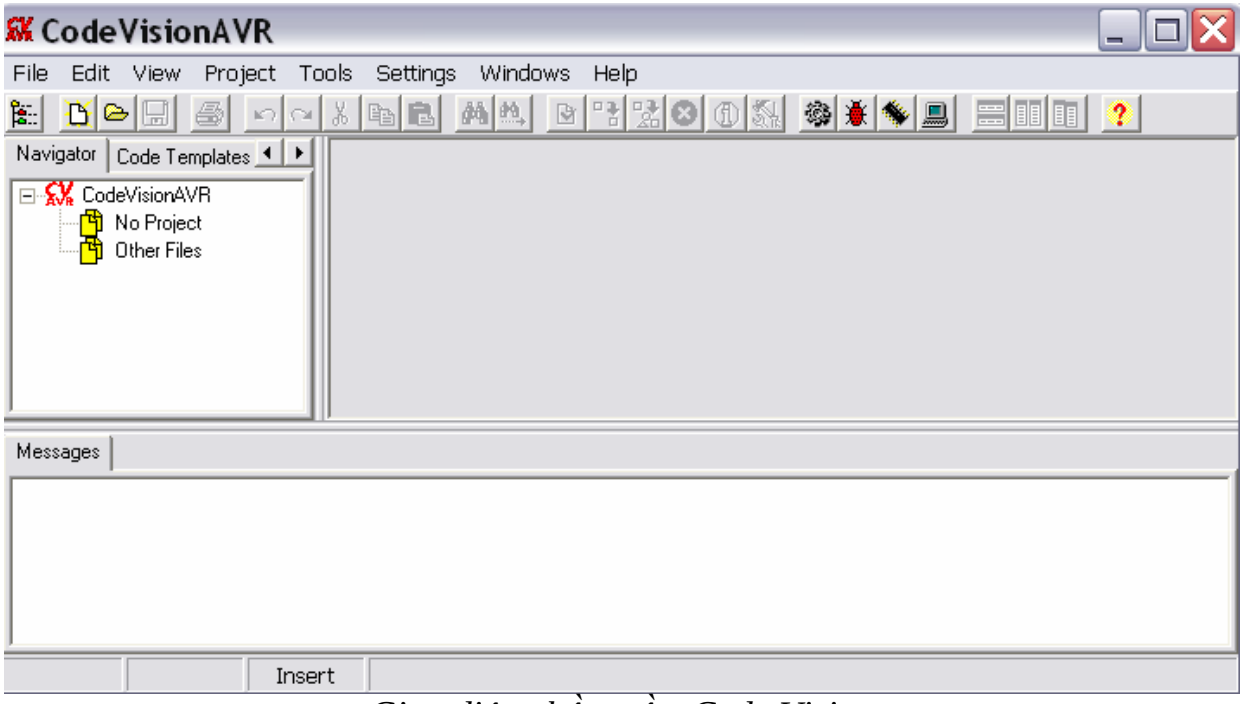
Để lập trình cho AVR, chúng ta có thể sử dụng 2 ngôn ngữ cơ bản là C và ASM. Nhìn chung, 2 ngôn ngữ này có những ưu và nhược điểm riêng.

Ngôn ngữ ASM có ưu điểm là gọn nhẹ, giúp người lập trình nắm bắt sâu hơn về phần cứng. Tuy nhiên lại có nhược điểm là phức tạp, khó triển khai về mặt thuật toán, không thuận tiện để xây dựng các chương trình lớn.

Ngược lại ngôn ngữ C lại dễ dung, tiện lợi, dễ debug, thuận tiện để xây dựng các chương trình lớn. Nhưng nhược điểm của ngôn ngữ C là khó giúp người lập trình hiểu biết sâu về phần cứng, các thanh ghi, tập lệnh của vi điều khiển, hơn nữa, xét về tốc độ, chương trình viết bằng ngôn ngữ C chạy chậm hơn chương trình viết bằng ngôn ngữ ASM.

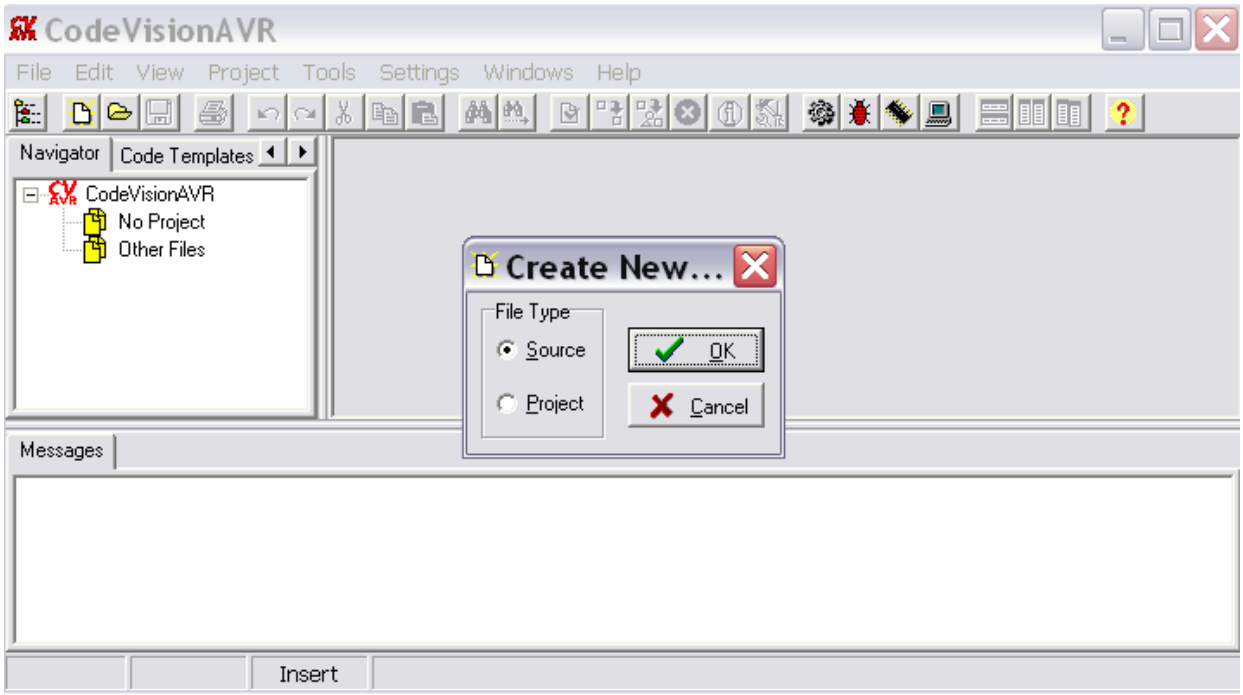
Tùy vào từng bài toán, từng yêu cầu cụ thể mà ta chọn lựa ngôn ngữ lập trình cho phù hợp.

Có rất nhiều phần mềm lập trình cho AVR, như Code Vision, IAR, AVRStudio…, trong đó Code Vision là một trong những phần mềm khá nổi tiếng và phổ biến. Trong khuôn khổ giáo trình này, chúng ta sẽ sử dụng phần mềm Code Vision để lập trình cho AVR.

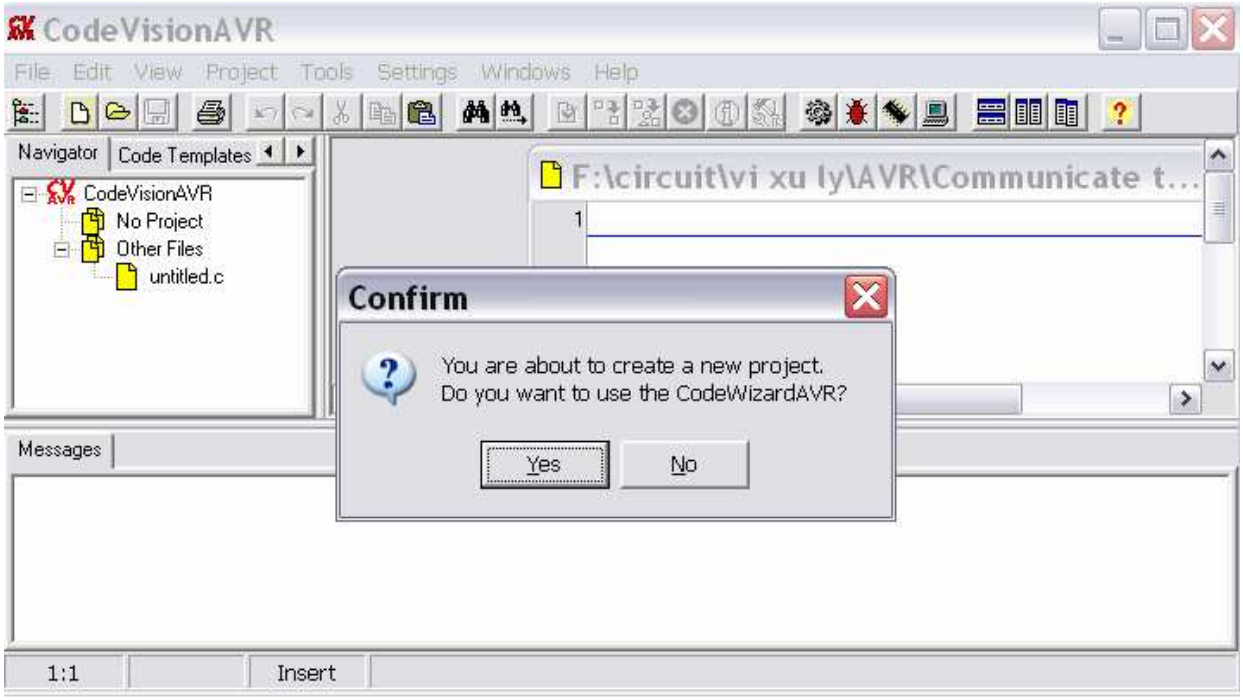


Giao diện phần mềm Code Vision

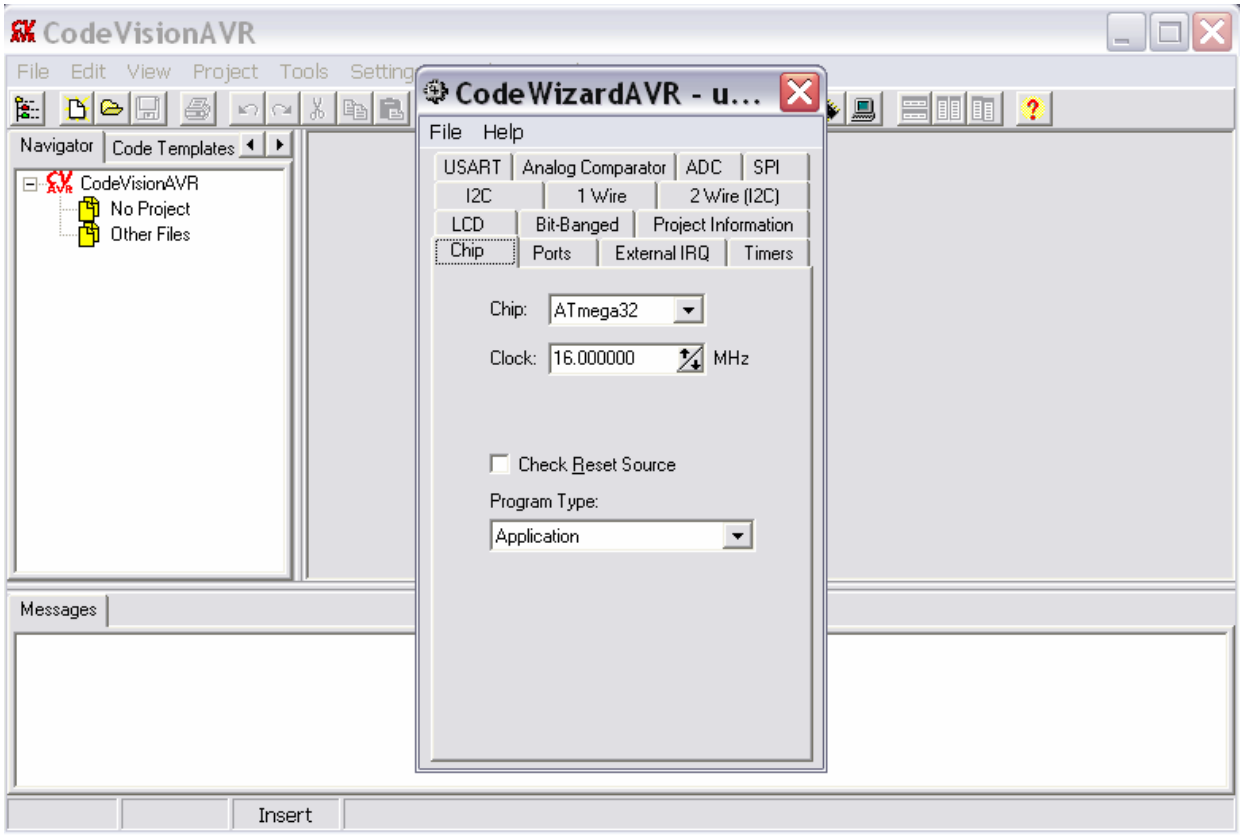
Tạo project trong Code Vision :
Để tạo Project mới chọn trên menu: File -> New được như sau:



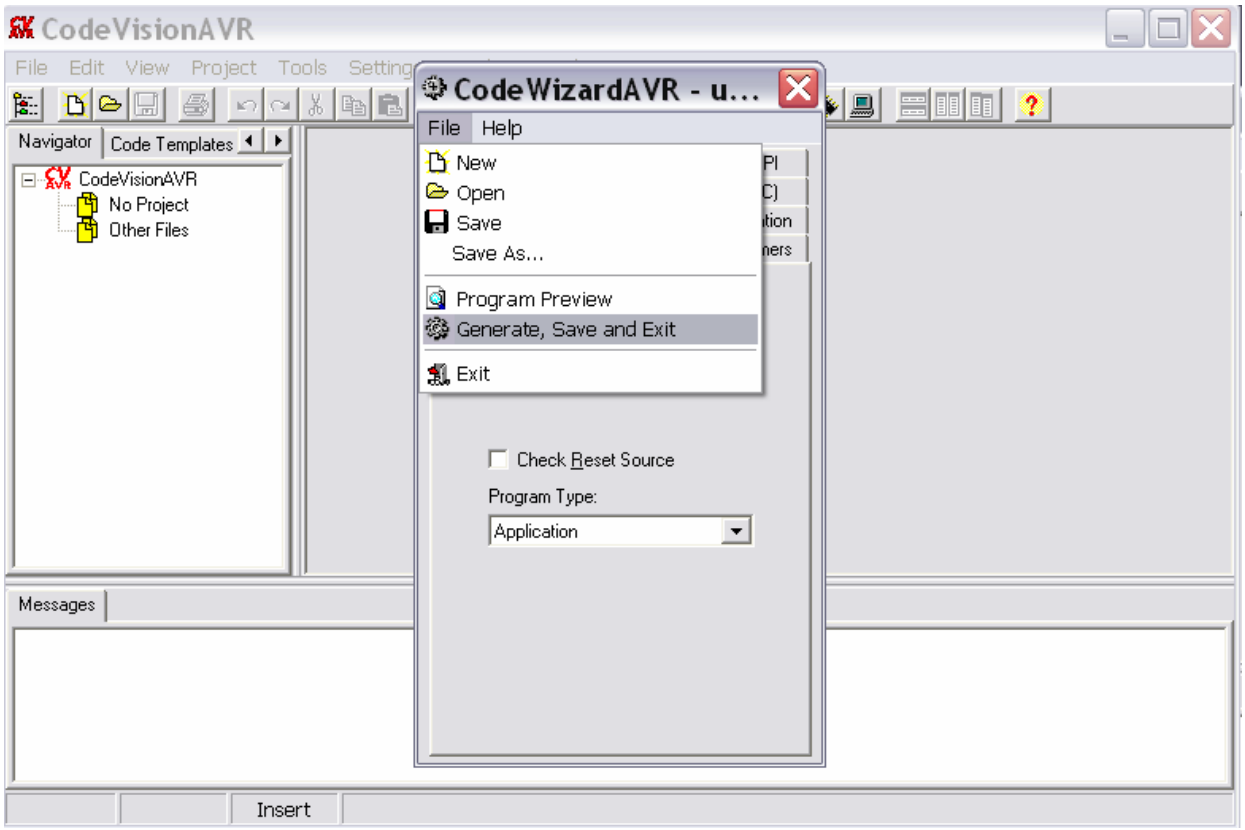
Chọn Project sau đó click chuột vào OK được cửa sổ hỏi xem có sử dụng Code Winzard không:



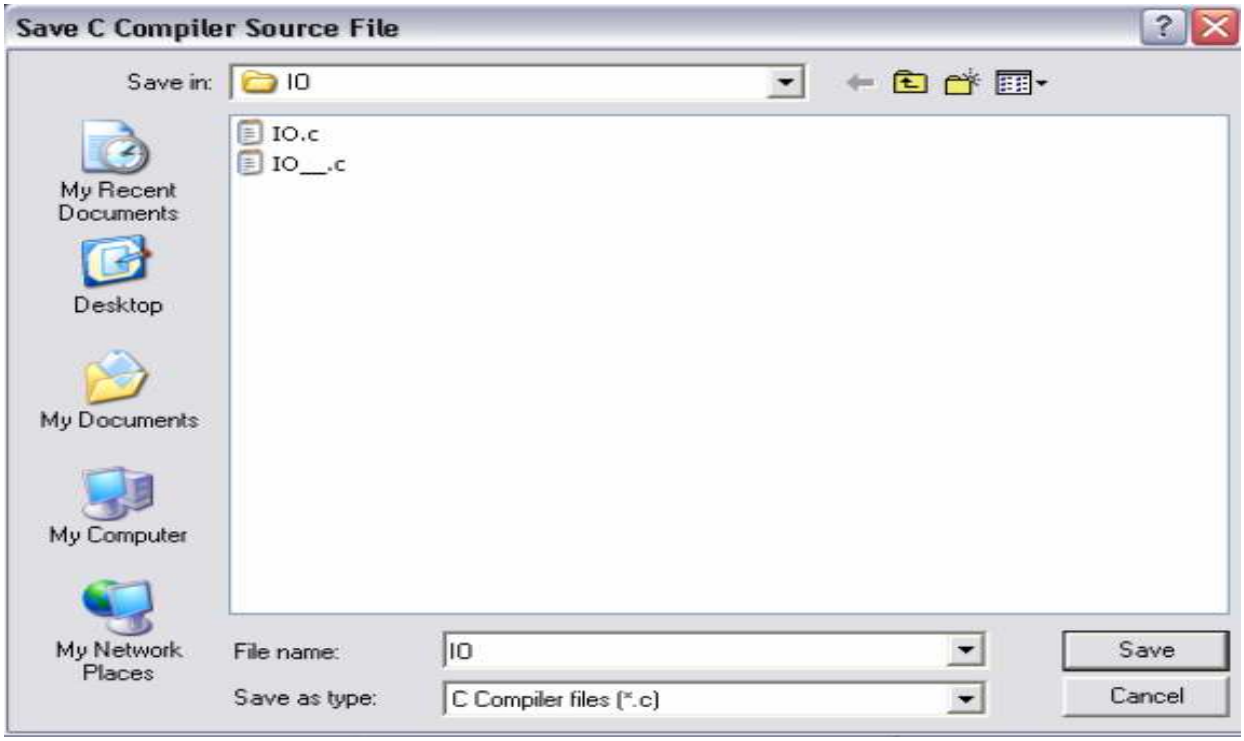
Chọn Yes được cửa sổ CodeWinzardAVR như sau :



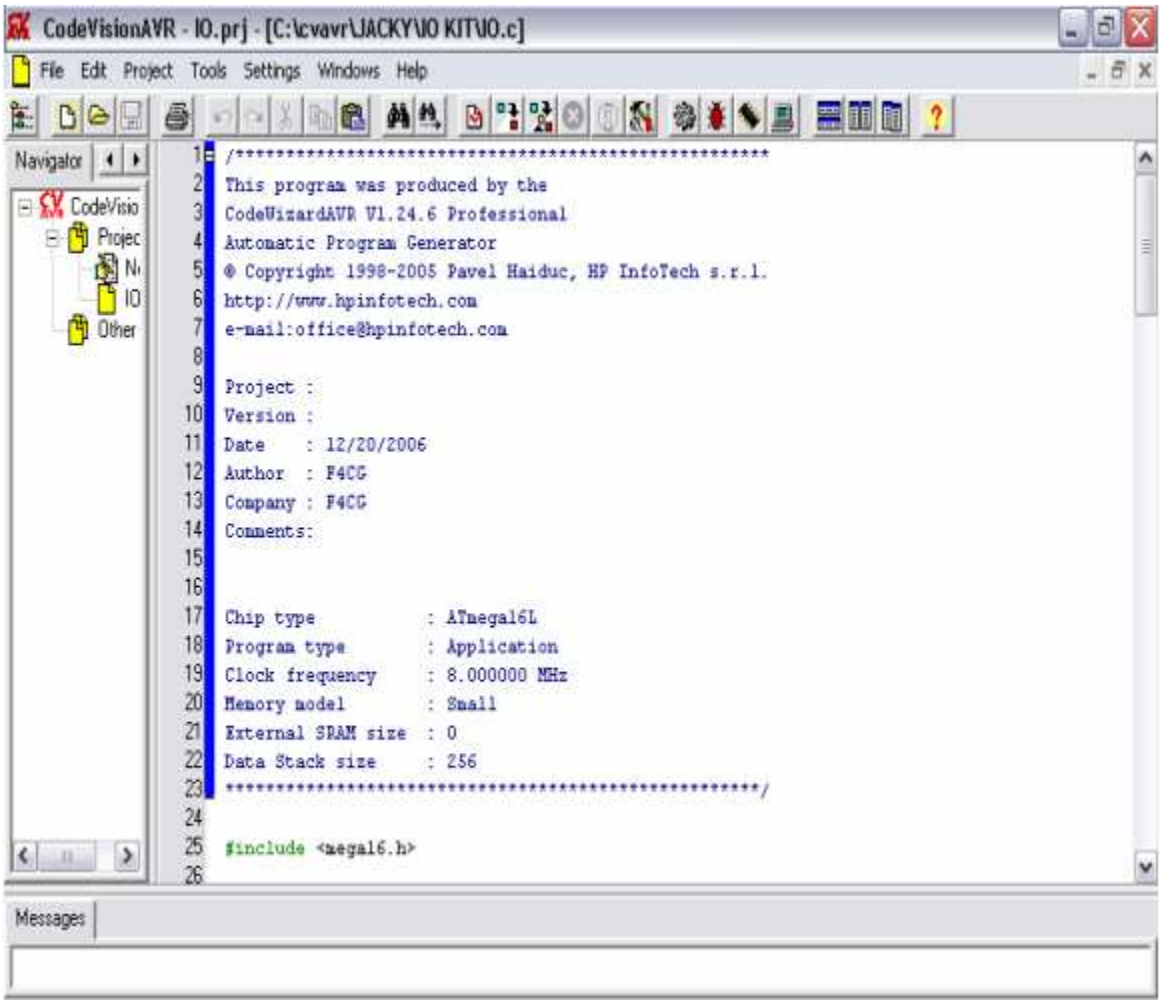
- Sử dụng chíp AVR nào và thạch anh tần số bao nhiêu ta nhập vào tab Chip. Để khởi tạo cho các cổng IO ta chuyển qua tab Ports.
- Các chân IO của AVR mặc định trạng thái IN, muốn chuyển thành trạng thái OUT để có thể đưa các mức logic ra ta click chuột vào các nút IN (mầu trắng) để nó chuyển thành OUT trong các Tab Port. Sau đó chọn *File -> Generate, Save and Exit.*



Sau đó ta save project lại :



Ta được như sau :



Như vậy là chúng ta đã tạo xong project trong Code Vision.

BÀI 2 : GIAO TIẾP VÀO RA I/O

- *Cơ bản về giao tiếp vào ra I/O*
- *Các cổng trong atmega32 và cơ bản về chức năng của các cổng*
- *Cách cấu hình vào ra I/O*
- *Viết chương trình nháy led*

1. Giới thiệu giao tiếp vào ra I/O

Lập trình I/O là lập trình đơn giản và cơ bản nhất, nhưng lại được sử dụng nhiều nhất, chúng ta điều khiển on/off bóng đèn, động cơ, hay 1 thiết bị nào đó cũng là 1 dạng của điều khiển I/O.

Để giảm bớt số chân ra, một số chân của AVR là các chân đa chức năng, nó phục vụ cho các thiết bị ngoại vi. Ở đây khái niệm thiết bị ngoại vi không có nghĩa là 1 chip khác mua rời bên ngoài mà là các mô đun được tích hợp sẵn trong chip như các mô đun ADC.... Khi các thiết bị ngoại vi này được enable thì các chân này không được sử dụng như các chân của các cổng I/O thông thường nữa.

2. Cách cấu hình chức năng IO

Atmega32 có 4 cổng vào ra là PORTA, PORTB, PORTC, PORTD. Khi xem xét đến các cổng I/O của AVR thì ta phải xét tới 3 thanh ghi DDxn,PORTxn,PINxn.

- Các bit DDxn để truy cập cho địa chỉ xuất nhập DDRx. Bit DDxn trong thanh ghi DDRx dùng để điều khiển hướng dữ liệu của các chân của cổng này.Khi ghi giá trị logic ‘0’ vào bất kì bit nào của thanh ghi này thì nó sẽ trở thành lối vào,còn ghi ‘1’ vào bit đó thì nó trở thành lối ra.
- Các bit PORTxn để truy cập tại địa chỉ xuất nhập PORTx. Khi PORTx được ghi giá trị 1 khi các chân có cấu tạo như cổng ra thì điện trở kéo là chủ động(được nối với cổng). Ngắt điện trở kéo ra, PORTx được ghi giá trị 0 hoặc các chân có dạng như cổng ra.Các chân của cổng là 3 trạng thái khi 1 điều kiện reset là tích cực thậm chí xung đồng hồ không hoạt động.
- Các bit PINxn để truy cập tại địa chỉ xuất nhập PINx. PINx là các cổng chỉ để đọc,các cổng này có thể đọc trạng thái logic của PORTx.PINx không phải là thanh ghi,việc đọc PINx cho phép ta đọc giá trị logic trên các chân của PORTx.chú ý PINx không phải là thanh ghi,việc đọc PINx cho phép ta đọc giá trị logic trên các chân của PORTx.
- Nếu PORTxn được ghi giá trị logic ‘1’ khi các chân của cổng có dạng như chân ra ,các chân có giá trị ‘1’.Nếu PORTxn ghi giá trị ‘0’ khi các chân của cổng có dạng như chân ra thì các chân đó có giá trị ‘0’.
- Các cổng của AVR đều có thể đọc, ghi. Để thiết lập 1 cổng là cổng vào, ra thì ta tác động tới các bit DDxn, PORTxn, PINxn. Ta có thể thiết lập để từng bit làm cổng vào, ra cứ không chỉ với cổng, như vậy ta có thể xử lý tới từng bit, đây chính là điểm mạnh của các dòng Vi điều khiển 8 bit.

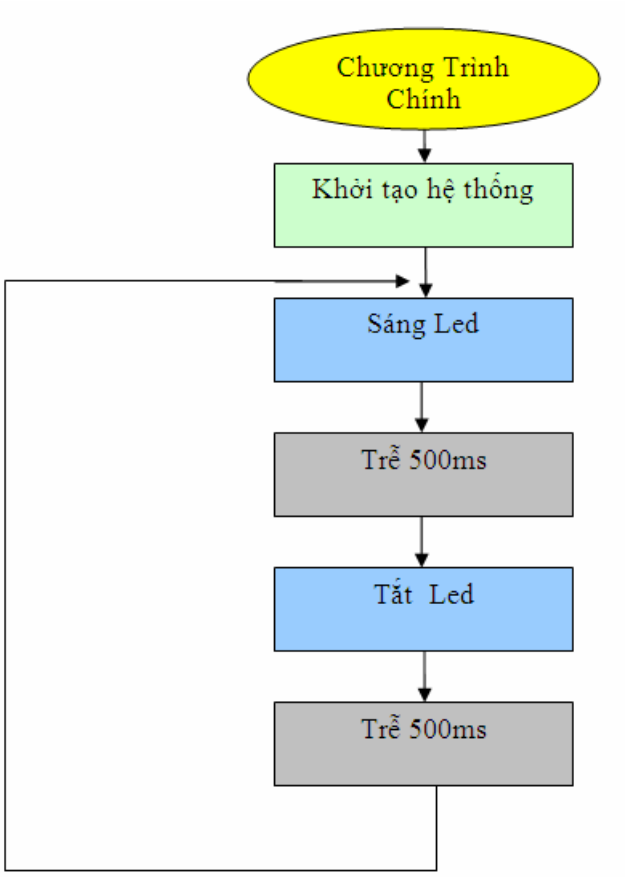
3. Ví dụ minh họa

Chương trình sau sẽ làm nhấp nháy cả 8 led, led nối vào port A.

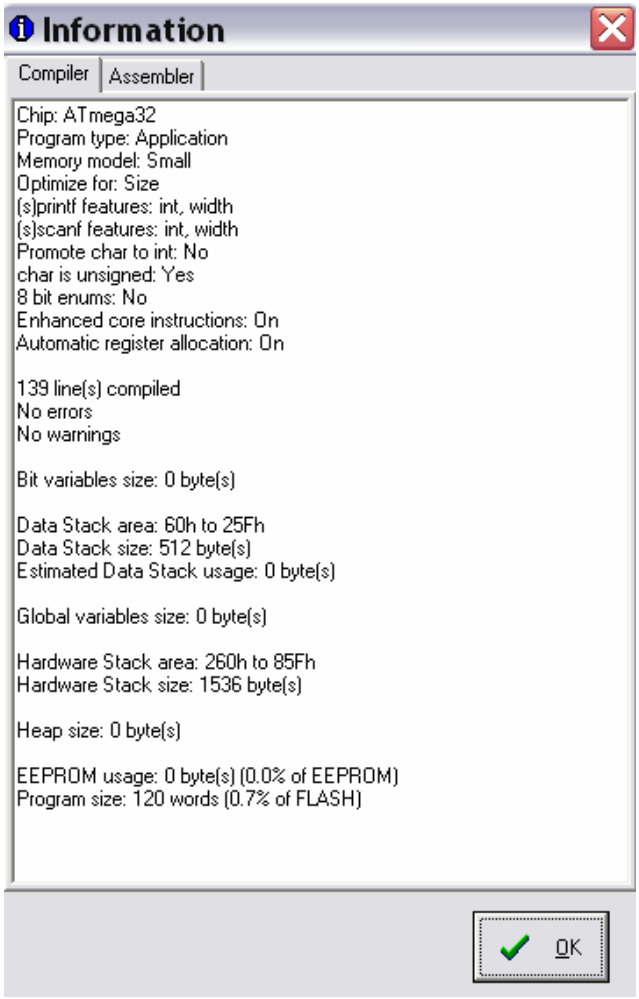
```
// Led blinking program
// Author : pk
// Date : 30/08/2009

#include <mega32.h>
#include <delay.h>

void main(){
        PORTA = 0x00;     // Gia tri ban dau cho cong A
        DDRA = 0xFF;      // Chon cong A la cong ra
        while(1){
                PORTA = 0;
                delay_ms(500);
                PORTA = 0xFF;
                delay_ms(500);
        }

}
```

Phân tích

Chương trình trên rất đơn giản, sơ đồ thuật toán của chương trình trên như sau :

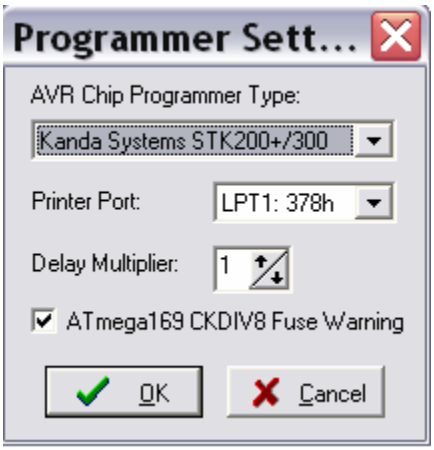


Sau khi viết xong chương trình, chúng ta nhấn Shift+F9 để biên dịch. Nếu chương trình không có lỗi và biên dịch thành công, sẽ có thông báo như sau :

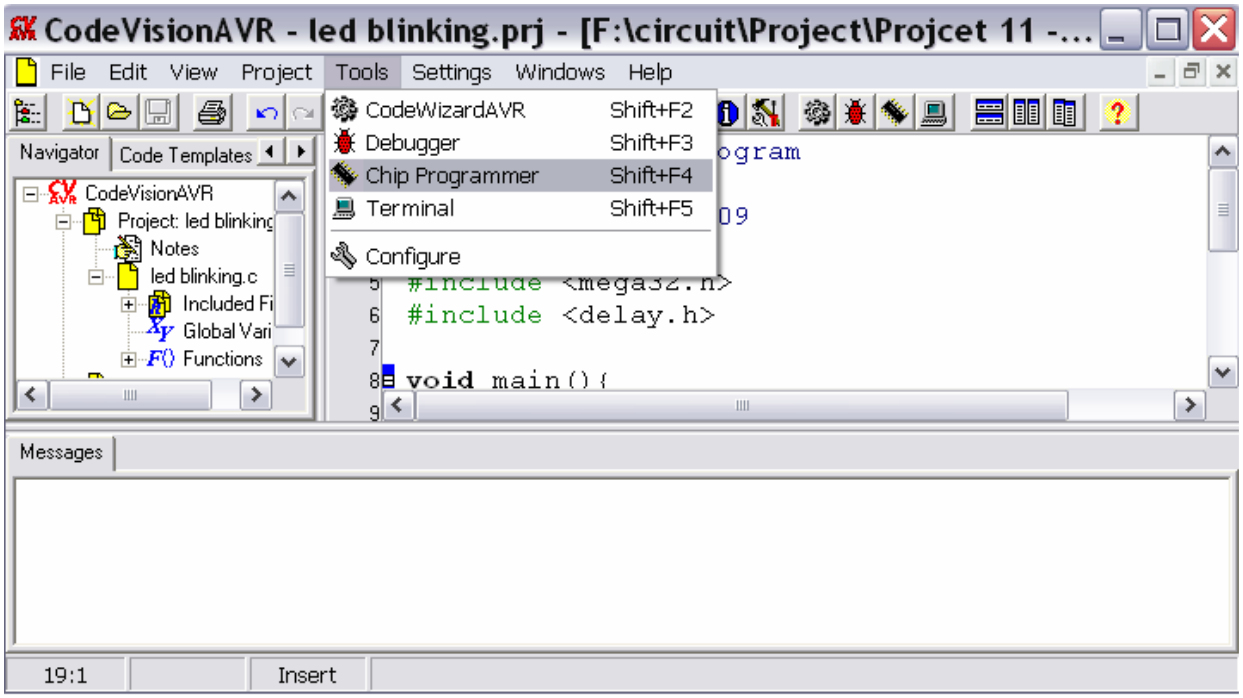


Để nạp chương trình các bạn cần cấu hình cho mạch nạp. Vào menu: Settings -> Programmer được cửa sổ như sau :

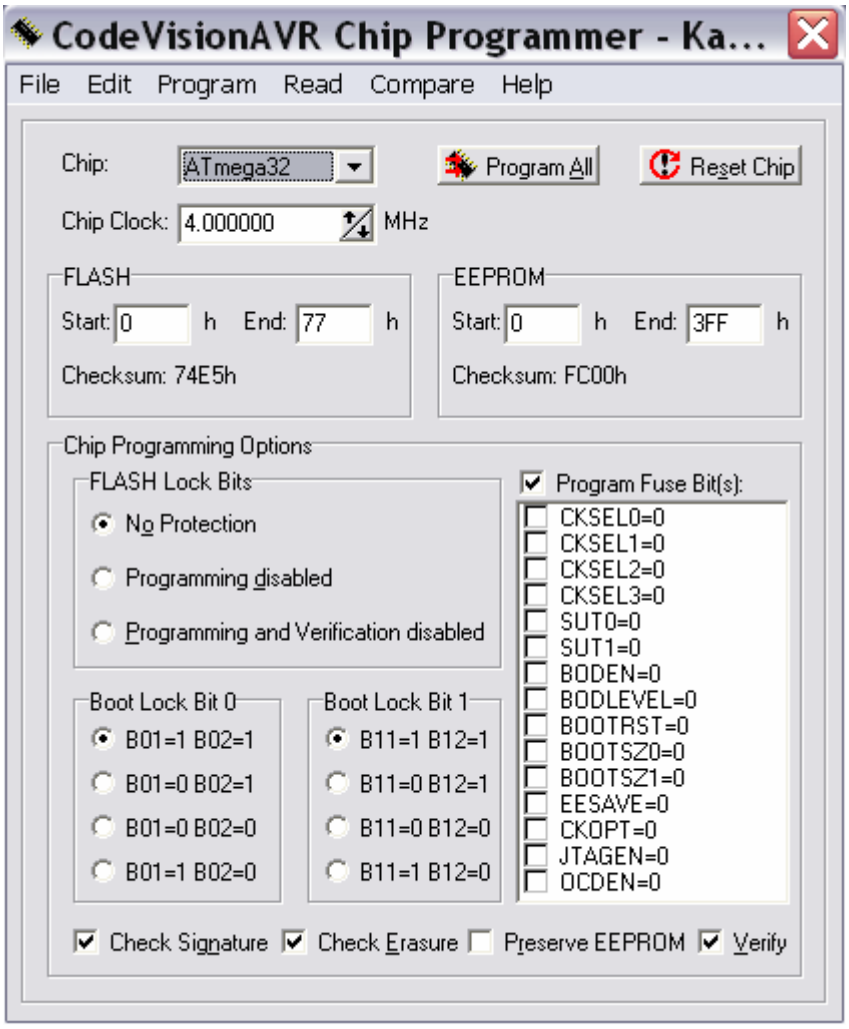
Mạch nạp ta dùng STK 200 do đó các bạn chọn Kanda *Systems STK200+/300*. Nhấp OK. Sau đó các bạn chọn trên menu : Projects -> Configure được cửa sổ như sau:



Sau đó bạn chọn Too/ Chip Programmer để nạp cho AVR :



Chúng ta được cửa sổ như sau :



Các bạn cấu hình các thông số cần thiết, như chọn thạch anh nội hay ngoại, cấu hình các fuse bit… rồi nhấn vào Program All để nạp chương trình.

BÀI 3 : GIAO TIẾP VỚI LED 7 THANH

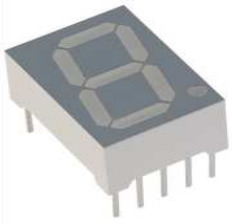
- *Cơ bản về led 7 thanh*
- *Nguyên lí lập trình led 7 thanh.*
- *Ví dụ minh họa*

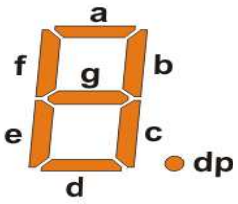
1. Cơ bản về led 7 thanh

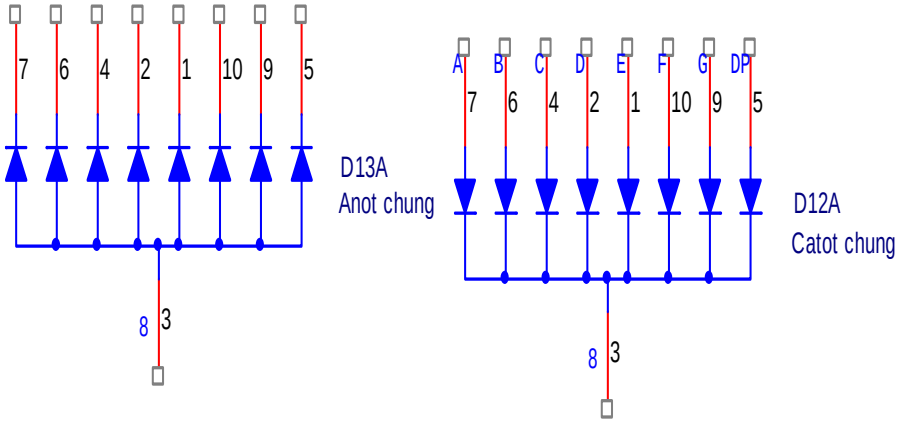
Ở bài học này, chúng ta sẽ học về giao tiếp giữa AVR và led 7 thanh ,các hiển thị số trên led 7 thanh , cũng như các giải thuật về quét led.

Led 7 thanh là linh kiện điện tử dùng để hiển thị số. Ưu điểm của led 7 thanh là giá thành rẻ, khoảng cách quan sát xa và dễ dàng trong lập trình. Nhược điểm là led 7 thanh chỉ hiển thị được 1 số kí tự nhất định.

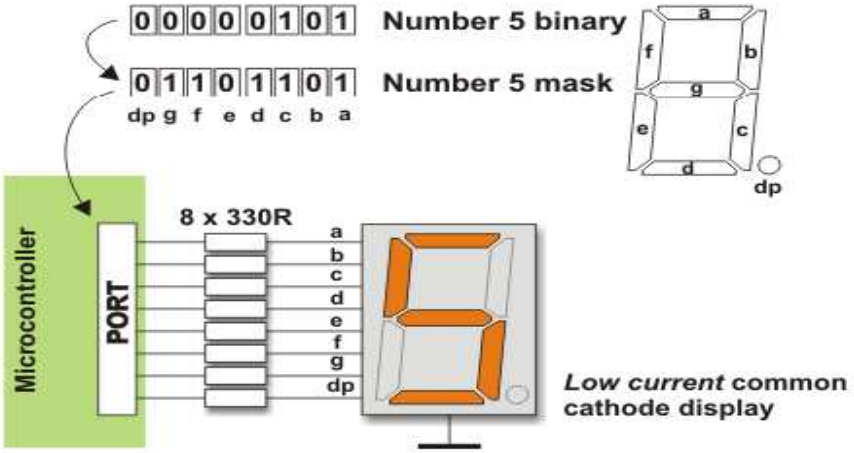
Led 7 thanh có 2 loại là anot chung và catot chung. Có hình dạng thực tế và hình dạng nguyên lí như hình sau :






7
6
4
2
1
10
9
5
8
3
D13A
Anot chung
A
B
C
D
E
F
G
DP
7
6
4
2
1
10
9
5
8
3
D12A
Catot chung

2. Nguyên lí lập trình cho led 7 thanh



Sơ đồ ghép nối với vi điều khiển

Led 7 thanh bao gồm 7 thanh a,b,c,d,e,f,g và 1 "thanh" dp, mỗi thanh là một led. Tùy vào cách nối chung anot hay catot giữa các thanh mà ta có 2 loại anot chung hoặc catot chung.

Như hình vẽ trên, led 7 thanh có dạng anot chung, muốn thanh nào sáng, chúng ta chỉ việc cấp điện áp dương vào chân tương ứng, khi đó led tương ứng với thanh đó sẽ được phân cực thuận và phát sáng.

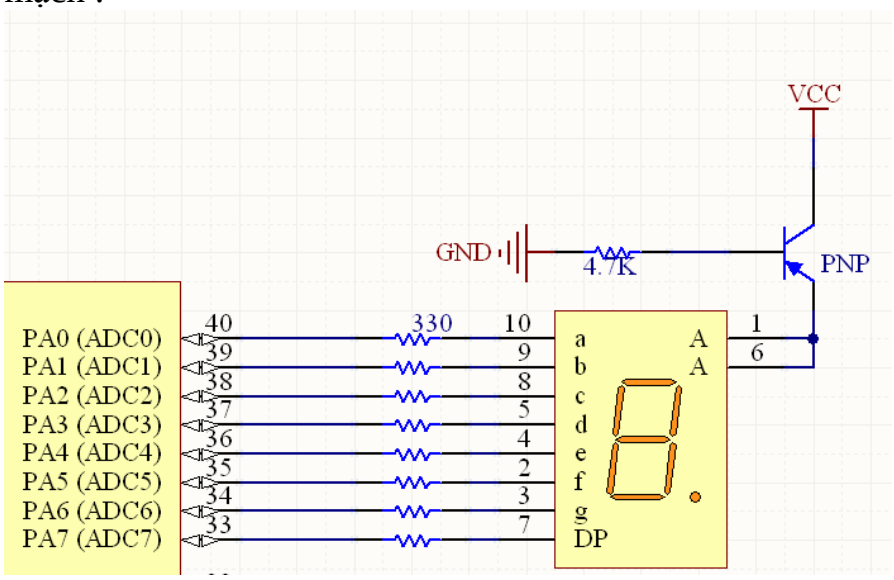
Ví dụ như hình vẽ trên, để sáng thành hình số 5, ta cần các thanh a,f,g,c,d sáng, khi đó ta cần cấp mức logic 1 (tương ứng với điện áp 5V) vào các chân tương ứng, và kết quả là ta được 1 chuỗi số nhị phân 10110110, hay ở dạng mã hex : 0xB6.

Bằng cách tương tự, ta cũng tạo được giá trị (mã) để xuất ra port của vi điều khiển để led sáng các số từ 0 đến 9. Người ta thường tạo ra 1 bảng mã như vậy như vậy để tiện sử dụng.

3. Ví dụ minh họa

Ở ví dụ sau, chúng ta sẽ hiển thị lần lượt các số từ 0 đến 9 lên led 7 thanh.

Sơ đồ mạch :



Bảng mã hóa các chữ số

Các số hiển thị	P1.7 dp	P1.6 g	P1.5 f	P1.4 e	P1.3 d	P1.2 c	P1.1 b	P1.0 a	Số nạp hex
0	1	1	0	0	0	0	0	0	0xC0
1	1	1	1	1	1	0	0	1	0xF9
2	1	0	1	0	0	1	0	0	0xA4
3	1	0	1	1	0	0	0	0	0xB0
4	1	0	0	1	1	0	0	1	0x99
5	1	0	0	1	0	0	1	0	0x92
6	1	0	0	0	0	0	1	0	0x82
7	1	1	1	1	1	0	0	0	0xF8
8	1	0	0	0	0	0	0	0	0x80
9	1	0	0	1	0	0	0	0	0x90

Chương trình

```
#include <mega32.h>
#include <delay.h>

void main(){
    unsigned char font[10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xf8,0x80,0x90};
    int i;
    DDRA = 0xFF;

    while(1){
        for(i = 0; i <=9; i++){
            PORTA = font[i];
            delay_ms(500);
        }
    }
}
```

Trong chương trình trên, các câu lệnh cấu hình tương tự như phần trước, chúng ta chỉ phân tích về thuật toán.

Biến font[] là một mảng số kiểu char, dùng để lưu trữ các mã của các số tương ứng, ví dụ số 0 sẽ có mã là phần tử đầu tiên của mảng : font[0] hay 0xC0, tương tự, số 1 sẽ có mã là font[1] hay 0xF9…

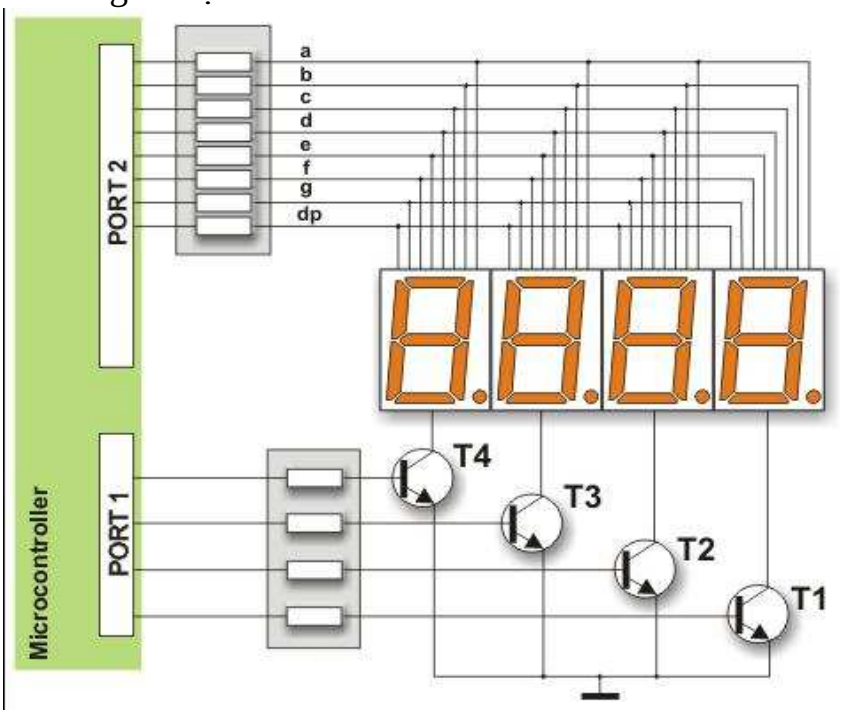
Lần lượt chúng ta xuất từng phần tử của mảng font[] ra cổng nối vào led (port B), khi chạy chương trình, chúng ta sẽ thấy led sáng từ 0 đến 9.

Cách giao tiếp với nhiều led

Chúng ta có thể sử dụng nhiều port để giao tiếp với nhiều led 7 thanh, mỗi led nối với 1 port khác nhau, tuy nhiên, vi điều khiển, ví dụ như dòng 16F887 chỉ có 4 port 8 bit, nếu làm như vậy, chúng ta chỉ có thể giao tiếp với nhiều nhất là 4 led 7 thanh.

Để giải quyết vấn đề trên, người ta sử dụng 1 phương pháp là quét led, tại một thời điểm chỉ có một led sáng, mỗi led sẽ sáng trong một khoảng thời gian nhất định, sau đó led đó tắt và led kế tiếp lại sáng. Làm như vậy, với khoảng thời gian sáng/tắt rất nhanh, mắt chúng ta không thể phân biệt được sự rời rạc đó và kết quả là chúng ta sẽ thấy led sáng liên tục.

Với phương pháp quét led, người ta chia ra làm 2 đường : đường điều khiển và đường dữ liệu, đường dữ liệu được nối vào các thanh a, b,c,d,e,f,g, đường điều khiển dùng để bật/tắt các led.



Ví dụ như hình vẽ trên, chúng ta chỉ cần dùng 2 port để điều khiển 4 led, port dữ liệu là port 2 và port điều khiển là port 1.

Bài tập

- Viết chương trình hiển thị số 1234 led 4 led 7 thanh theo như gợi ý trên.
- Viết chương trình đếm trong 1 khoảng bất kì nhỏ hơn 9999, ví dụ từ 1000 đến 65535. Số đếm được hiển thị lên 4 led 7 thanh.

BÀI 4 : GIAO TIẾP VỚI BÀN PHÍM

- *Cơ bản về phím bấm.*
- *Chương trình ví dụ giao tiếp với phím bấm*

1. Cơ bản về phím bấm

Bàn phím được sử dụng trong rất nhiều các thiết bị, để giúp người sử dụng lựa chọn các chức năng của thiết bị. Có thể nói giao tiếp bàn phím là một ứng dụng khá quan trọng.

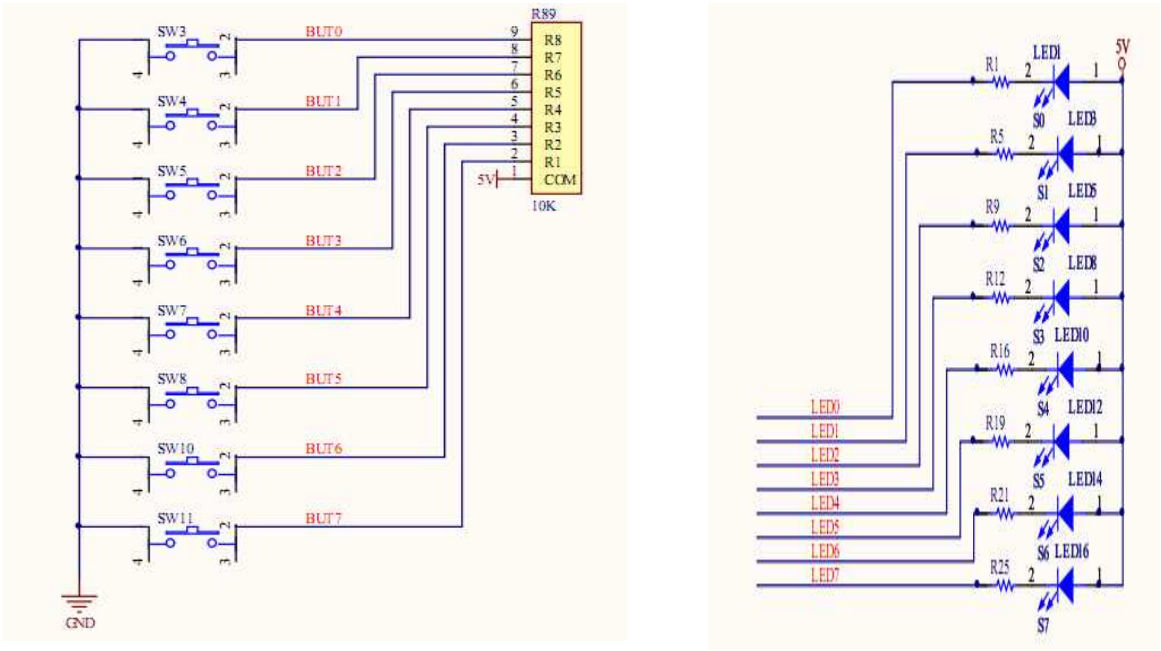
Phím bấm thông dụng nhất có cấu tạo gồm 2 đầu tiếp xúc, mỗi khi chúng ta bấm phím, 2 đầu này sẽ chạm vào nhau (xem hình vẽ ở sơ đồ nguyên lí bên dưới).

Ngoài ra còn nhiều loại phím bấm khác, và cấu tạo cũng khác, có thể là phím bấm thường đóng, khi ta bấm phím thì 2 đầu tiếp xúc không thông nhau. Hoặc cũng có loại phím bấm cảm ứng, dựa trên sự thay đổi điện trở của màng điện trở, hoặc dựa trên sự thay đổi điện dung hay điện cảm mỗi khi có tay người chạm vào.

2. Chương trình ví dụ

Ở ví dụ này ,chúng ta sẽ lập trình để dùng bàn phím điều khiển các con led bật tắt theo ý muốn.

Sơ đồ nguyên lý



Có 8 phím bấm, được nối với port D, các led đơn được nối vào port. Chúng ta sẽ lập trình để xem trạng thái của port D (trạng thái của các phím bấm) bằng cách quan sát trạng thái của led.

Chương trình :

```
#include <mega32.h>

void main(){

    DDRB = 0xFF;    // Chon cong RB la cong ra
    DDRD = 0;       // Chon cong RD la cong vao

    PORTB = 0;

    while(1){
        PORTB = PIND;
    }
}
```

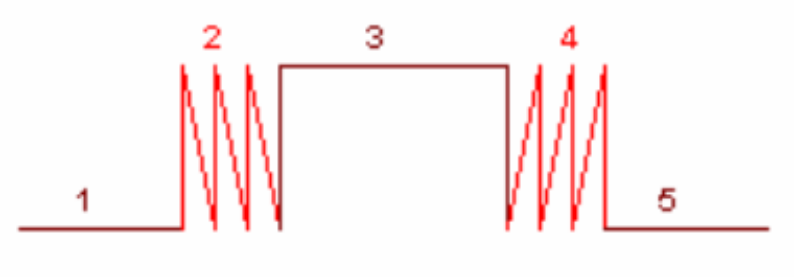
Phân tích chương trình :

Chương trình trên rất đơn giản, chúng ta set port B là port ra, port D là port vào, sau đó chúng ta liên tục lấy giá trị của port D gán cho giá trị của port B thông qua câu lệnh : `PORTB = PIND;`

3. Kĩ thuật chống rung bàn phím

Vì sao phải chống rung :

Bàn phím của chúng ta là bàn phím cơ học, bề mặt tiếp xúc của cơ cấu bên trong phím không phải là phẳng lí tưởng, do vậy, mỗi khi bấm phím hay nhả phím, xung vào vi điều khiển sẽ không phải là 1 xung thẳng đứng, mà là rất nhiều xung kim. Vì thời gian quét của vi điều khiển rất nhanh, nên tất cả các giá trị tại thời điểm rung đó đều được ghi lại. Chúng ta phải tìm cách sao cho vi điều khiển không lấy giá trị tại thời điểm rung.



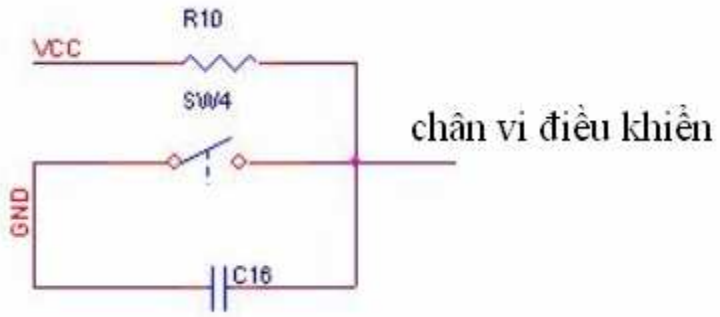
Sơ đồ xung khi bấm phím

Thời điểm 2 và 4 (xung màu đỏ) trong hình trên là thời điểm mà khi ta bắt đầu nhấn phím và khi bắt đầu nhả phím, thời điểm 1 và 5 là thời điểm phím ở trạng thái ổn định khi được nhả hoàn toàn, thời điểm 3 là thời điểm phím ở trạng thái ổn định khi đang được nhấn.

Có 2 phương pháp chống rung là chống rung bằng phần cứng và chống rung bằng phần mềm.

Chống rung bằng phần cứng

Chúng ta mắc thêm tụ nối song song với phím bấm, thường là tụ 104, tụ này có tác xụng hấp thụ các xung nhọn đi vào chân vi điều khiển, như vậy sẽ triệt tiêu hoàn toàn các xung kim.



Chống rung bằng phần mềm

Mỗi khi phát hiện có tín hiệu bấm phím, chúng ta cho vi điều khiển không đọc liên tục giá trị của phím nữa bằng cách cho delay một khoảng thời gian, khoảng trên 10ms, sau khoảng thời gian đó, chúng ta lại đọc phím như bình thường. Ví dụ code như sau :

```
If(phát hiện bấm phím){
    Delay_ms(10);
    Tiếp tục làm các công việc khác
    ……
}
```

Bài tập

- Viết chương trình giao tiếp với phím bấm và led 7 thanh, mỗi khi bấm phím, số trên led lại tăng lên 1 đơn vị. Khi số tăng đến 9 mà bấm tiếp thì số trở về 0.

-

BÀI 5 : BỘ CHUYỂN ĐỔI ADC

- *Giới thiệu về ADC.*
- *Cách cấu hình sử dụng module ADC trong Code Vision cho Atmega32*
- *Ví dụ*

1. Giới thiệu về ADC

Chúng ta biết rằng các tín hiệu ở thế giới xung quanh chúng ta toàn là các tín hiệu tương tự : dòng điện 220VAC, dòng điện 5V, sức gió, tốc độ động cơ, tuy nhiên vi điều khiển chỉ xử lí được các tín hiệu số : 10101, như vậy, cần phải có 1 thiết bị nào đó để chuyển đổi qua lại giữa 2 loại tín hiệu này, đó là lí do vì sao chúng ta có các bộ ADC và DAC.

ADC là 1 thiết bị dùng để chuyển đổi tín hiệu tương tự thành tín hiệu số. Còn DAC thì ngược lại, chuyển tín hiệu số thành tín hiệu tương tự.

Atmega32 có 8 chân của PORTA sử dụng làm 8 kênh đầu vào ADC. Để sử dụng tính năng ADC của Atmega32 chúng ta cần phải thiết kế phần cứng của Vi điều khiển như sau :

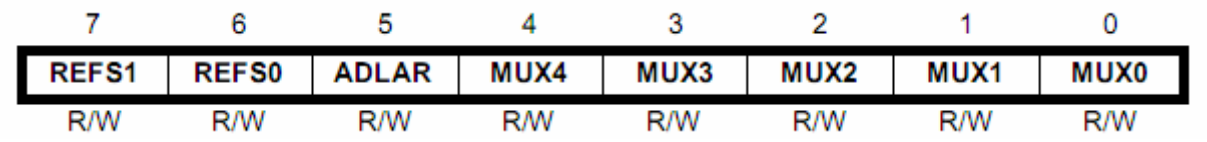
- Chân AVCC chân này bình thường khi thiết kế mạch chúng ta đưa lên Vcc(5V) nhưng khi trong mạch có sử dụng các kênh ADC của phần cứng thì chúng ta phải nối chân này lên Vcc qua 1 cuộn cảm nhằm mục đích cấp nguồn ổn định cho các kênh (đầu vào) của bộ biến đổi.
- Chân AREF chân này cần cấp 1 giá trị điện áp ổn định được sử dụng làm điện áp tham chiếu, chính vì vậy điện áp cấp vào chân này cần ổn định vì khi nó thay đổi làm giá trị ADC ở các kênh thu được bị trôi (thay đổi) không ổn định với 1 giá trị đầu vào chúng ta có công thức tính như sau:

  ADCx=(V_INT*1024)/ AREF

  Chúng ta thấy giá trị ADCx tỉ lệ thuận với điện áp vào V_INT. Giá trị ADC thu được từ các kênh được lưu vào 2 thanh ghi ADCH và ADCL khi sử dụng chúng ta phải đọc giá trị từ các thanh ghi này, khi sử dụng ở ché độ 8 bít thì chỉ lưu vào thanh ghi ADCL.

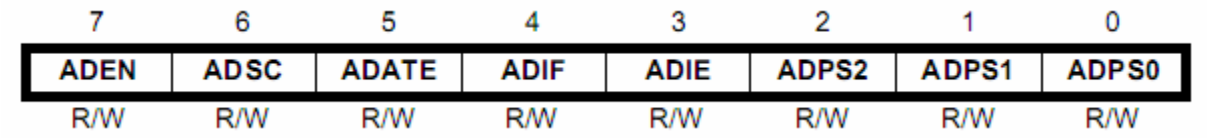
Các thanh ghi liên quan

ADMUX (ADC Mutiplexer Selection Register) : Là thanh ghi điều khiển việc chọn điện áp tham chiếu, kênh và chế độ hoạt động của ADC.



7	6	5	4	3	2	1	0
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ADCSRA (ADC Control and Status Register A) : Là thanh ghi điều khiển hoạt động và chứa trạng thái của module ADC.



7	6	5	4	3	2	1	0
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ADCL và ADCH (ADC Data Register) : Là 2 thanh ghi chứa giá trị của ADC sau quá trình chuyển đổi.

Cụ thể về ý nghĩa của các bit trong các thanh ghi này, các bạn có thể tham khảo trong datasheet của Atmega32.

2. Cách cấu hình ADC trong Code Vision cho Atmega32.

Sau đây là các bước cấu hình để module ADC hoạt động :

- Chọn điện áp tham chiếu, kênh đọc ADC (ADMUX)
- Cho phép module ADC hoạt động (ADCSRA)
- Cho phép quá trình chuyển đổi diễn ra và đọc giá trị sau khi chuyển đổi.

3. Ví dụ minh họa

Trong ví dụ sau, chúng ta sẽ đọc giá trị của ADC được nối vào chân A0, giá trị ADC sau khi chuyển đổi được xuất ra port B và D

Chương trình

```
#include <mega32.h>
#include <delay.h>

#define ADC_VREF_TYPE 0x00

// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input){
    ADMUX=adc_input|ADC_VREF_TYPE;
    // Start the AD conversion
    ADCSRA|=0x40;       // Start convert progress
    // Wait for the AD conversion to complete
    while ((ADCSRA & 0x10)==0);
    ADCSRA|=0x10;
    return ADCW;
}
```

```
void main(void){
    unsigned i = 0;

    PORTA = 0x00;
    DDRA = 0x00;

    PORTB = 0x00;
    DDRB = 0xFF;

    PORTD = 0x00;
    DDRD = 0xFF;

    ADMUX = ADC_VREF_TYPE;
    ADCSRA = 0x87;

    while (1){
        i = read_adc(0);
        PORTB = i&0xff;
        PORTD = i>>8;
        delay_ms(500);
    }
}
```

Bài tập

Bạn hãy phân tích chương trình trên và chỉ ra các chế độ hoạt động của module ADC được cấu hình như trên.

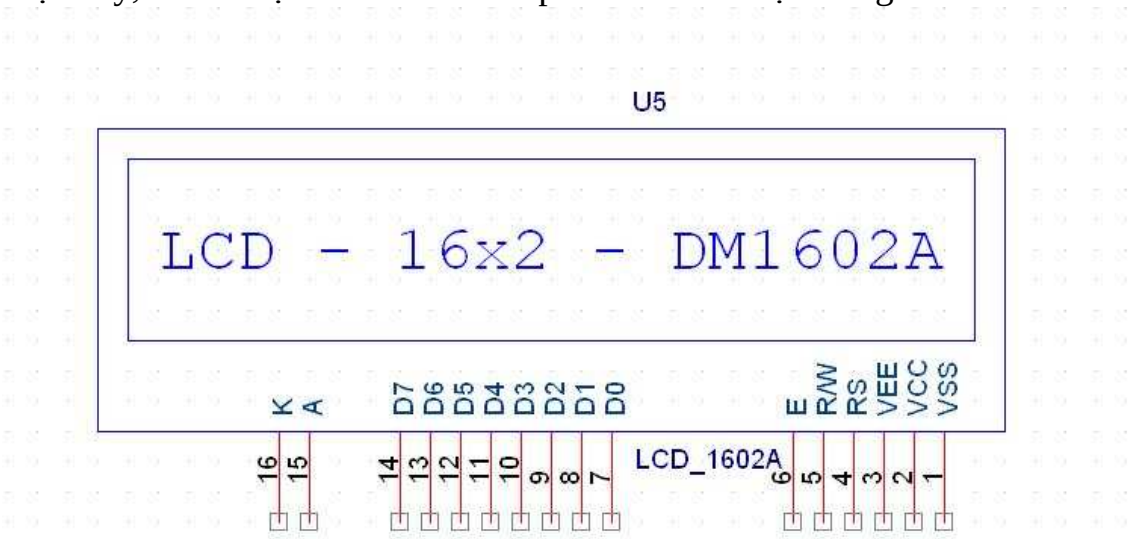
BÀI 6 : GIAO TIẾP LCD

- *Giới thiệu về LCD*
- *Cách cấu hình cho LCD trong Code Vision cho Atmega32*
- *Ví dụ minh họa*

1. Giới thiệu về LCD 16x2

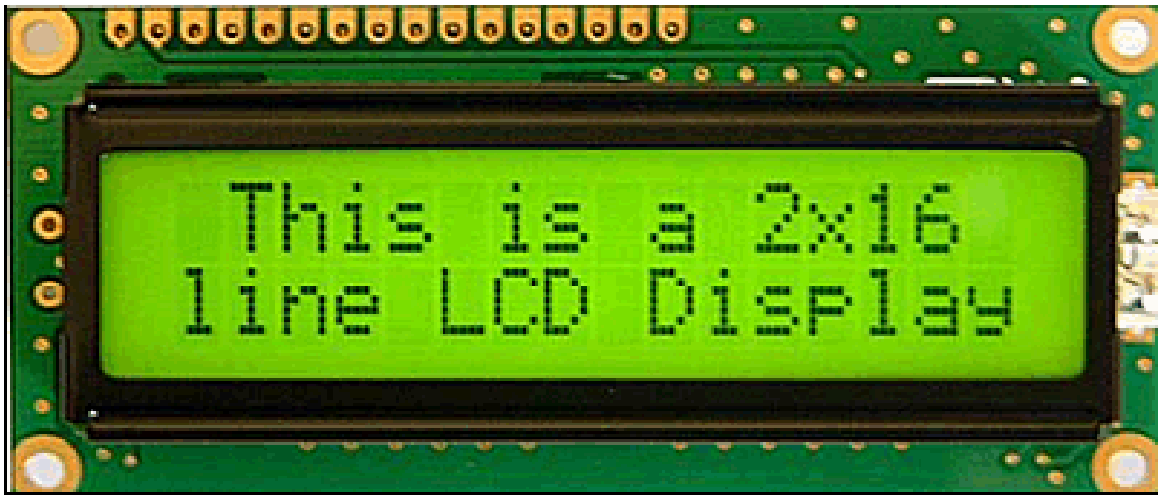
Giống như led 7 thanh, LCD là một thiết bị ngoại vi dùng để giao tiếp với người dùng, so với led 7 thanh thì LCD có ưu điểm là hiển thị được tất cả các kí tự trong bảng mã ascci, trong khi đó led 7 thanh chỉ hiển thị được một số kí tự, nhưng LCD lại có nhược điểm là giá thành cao và khoảng cách nhìn gần.

LCD là từ viết tắt của **L**iquid **C**rystal **D**isplay (màn hình tinh thể lỏng). Có nhiều loại màn hình LCD với các kích cỡ khác nhau, ví dụ như LCD 16x1 (16 cột và 1 hàng), LCD 16x2 (16 cột và 2 hàng), LCD 20x2 (20 cột và 2 hàng)… Trong bài học này, ta xét loại LCD 16x2 bán phổ biến trên thị trường.



Sơ đồ nguyên lí của LCD 16x2

Hình dạng thực của 1 LCD 16x2 (16 hàng, 2 cột).



Chức năng của các chân LCD :

Chân	Kí hiệu	Mức Logic	I/O	Chức năng
1	Vss	-	-	Nguồn (GND)
2	Vcc	-	-	Nguồn (+5V)
3	Vee	-	-	Chỉnh độ tương phản
4	RS	0/1	I	0 = Nhập lệnh 1 = Nhập dữ liệu
5	R/W	0/1	I	0 = Ghi dữ liệu 1 = Đọc dữ liệu
6	E	1, 1->0	I	Tín hiệu cho phép
7	DB0	0/1	I/O	Bus dữ liệu 0
8	DB1	0/1	I/O	Bus dữ liệu 1
9	DB2	0/1	I/O	Bus dữ liệu 2
10	DB3	0/1	I/O	Bus dữ liệu 3
11	DB4	0/1	I/O	Bus dữ liệu 4
12	DB5	0/1	I/O	Bus dữ liệu 5
13	DB6	0/1	I/O	Bus dữ liệu 6
14	DB7	0/1	I/O	Bus dữ liệu 7
15	Lamp-	-	-	Đèn LCD
16	Lamp+	-	-	Đèn LCD

Các chân Vcc, Vss và Vee

Chân Vcc cấp dương nguồn 5V, chân Vss nối đất, chân Vee được dùng để điều khiển độ tương phản của màn hình LCD.

RS (Register select)

Khi ở mức thấp, chỉ thị được truyền đến LCD như xoá màn hình ,vị trí con trỏ ….Khi ở mức cao, kí tự được truyền đến LCD

R/W (Read/Write)

Dùng để xác định hướng của dữ liệu được truyền giữa LCD và vi điều khiển. Khi nó ở mức thấp dữ liệu được ghi đến LCD và khi ở mức cao, dữ liệu được đọc từ LCD. Nếu chúng ta chỉ cần ghi dữ liệu lên LCD thì chúng ta có thể nối chân này xuống GND để tiết kiệm chân

E (Enable)

Cho phép ta truy cập/xuất đến LCD thông qua chân RS và R/W.Khi chân E ở mức cao (1) LCD sẽ kiểm tra trạng thái của 2 chân RS và R/W và đáp ứng cho phù hợp. Khi dữ liệu được cấp đến chân dữ liệu thì một xung mức cao xuống thấp phải được áp đến chân này để LCD chốt dữ liệu trên các chân dữ liêu. Xung này phải rộng tối thiểu là 450ns. Còn khi chân E ở mức thấp (0), LCD sẽ bị vô hiệu hoá hoặc bỏ qua tín hiệu của 2 chân RS và R/W.

Các chân **D0 - D7**

Đây là 8 chân dữ liệu 8 bít, được dùng để gửi thông tin lên LCD hoặc đọc nội dung của các thanh ghi trong LCD. Các kí tự được truyền theo mã tương ứng trong bảng mã ascii. Cũng có các mã lệnh mà có thể được gửi đến LCD để xoá màn hình hoặc đưa con trỏ về đầu dòng hoặc nhấp nháy con trỏ.

LCD có 2 chế độ giao tiếp, chế độ 4 bit (chỉ dùng 4 chân D4 đến D7 để truyền dữ liệu) và chế độ 8 bit (dùng cả 8 chân dữ liệu từ D0 đến D7), ở chế độ 4 bit, khi truyền 1 byte, chúng ta sẽ truyền nửa cao của byte trước, sau đó mới truyền nửa thấp của byte.

Trước khi truyền các kí tự ra màn hình LCD ta cần thiết lập cho LCD như chọn chế độ 4 bit hoặc 8 bit, 1 dòng hay 2 dòng ,bật/tắt con trỏ… Dưới đây là bảng tập lệnh của LCD :

Mã (Hex)	Lệnh đến thanh ghi của LCD
1	Xoá màn hình hiển thị
2	Trở về đầu dòng
4	Giả con trỏ (dịch con trỏ sang trái)
6	Tăng con trỏ (dịch con trỏ sang phải)
5	Dịch hiển thị sang phải
7	Dịch hiển thị sang trái
8	Tắt con trỏ, tắt hiển thị
A	Tắt hiển thị, bật con trỏ
C	Bật hiển thị, tắt con trỏ
E	Bật hiển thị, nhấp nháy con trỏ
F	Tắt con trỏ, nhấp nháy con trỏ
10	Dịch vị trí con trỏ sang trái
14	Dịch vị trí con trỏ sang phải
18	Dịch toàn bộ hiển thị sang trái
1C	Dịch toàn bộ hiển thị sang phải
80	Ép con trỏ Vũ đầu dòng thứ nhất
C0	Ép con trỏ Vũ đầu dòng thứ hai
38	Hai dòng và ma trận 5 × 7

Để đọc thanh ghi lệnh,ta phải đặt RS=0 và R/W =1 và xung cao xuống thấp cho bít E. Sau khi đọc thanh ghi lệnh,nếu bit D7(cờ bận) ở mức cao thì LCD bận, không có thông tin hay lệnh nào được xuất đến nó. Khi D7=0 mới có thể gửi lệnh hay dữ liệu đến LCD. Chúng ta nên kiểm tra bit cờ bận trước khi ghi thông tin lên LCD.

Bảng dữ liệu của LCD

Có thể di chuyển con trỏ đến vị trí bất kì trên màn hình LCD bằng cách nạp vào các giá trị tương ứng như bảng sau và gửi yêu cầu đến LCD:

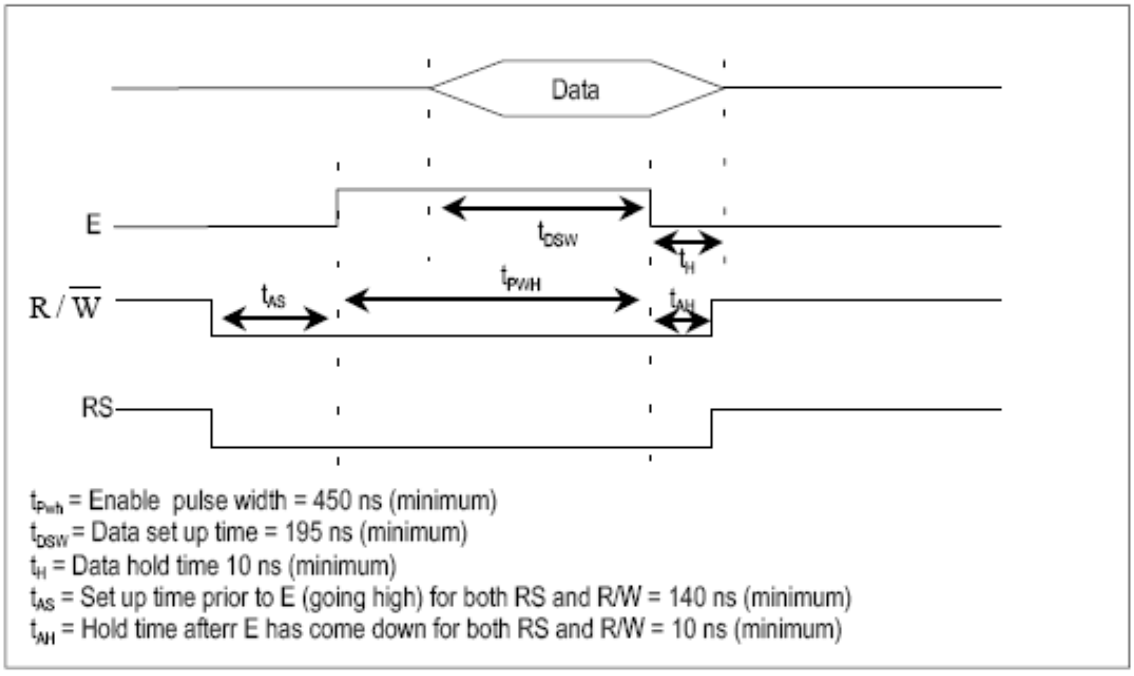
RS	E/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	A	A	A	A	A	A	A

	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Dòng 1 (min)	1	0	0	0	0	0	0	0
Dòng 1 (max)	1	0	1	0	0	1	1	1
Dòng 2 (min)	1	1	0	0	0	0	0	0
Dòng 2 (max)	1	1	1	0	0	1	1	1

Như vậy,0x80 đến 0x8F cho dòng lệnh 1.
0xC0 đến 0xCF cho dòng lệnh 2

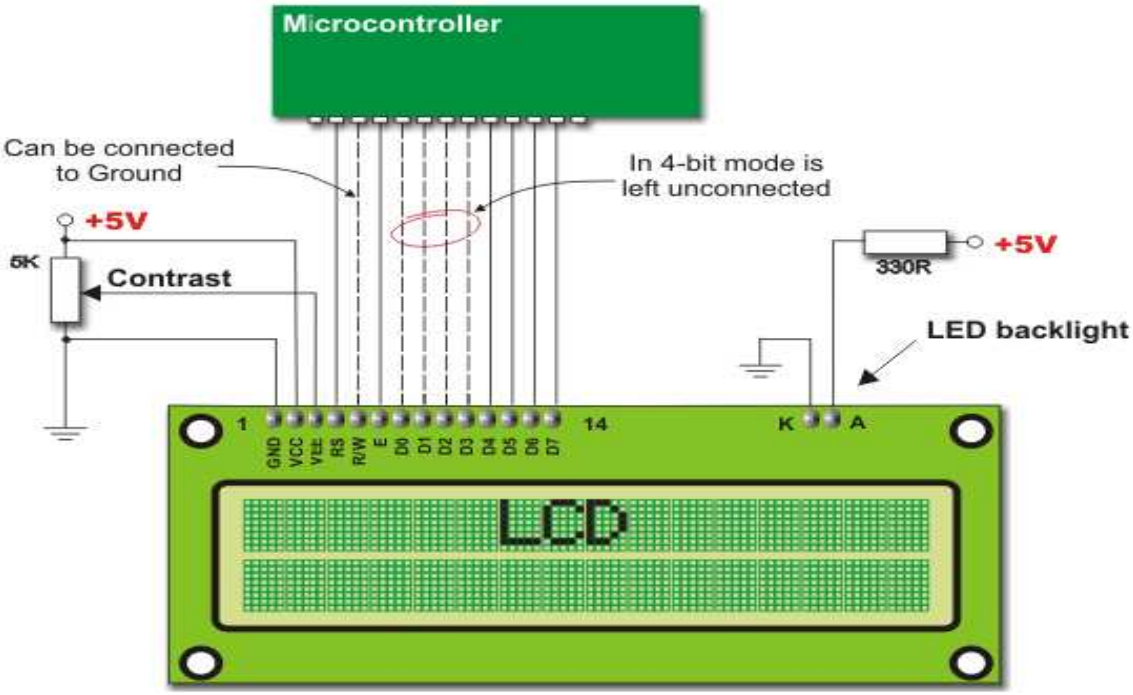
16 × 2 LCD	80	81	82	83	84	85	86	Through	8F
	C0	C0	C2	C3	C4	C5	C6	Through	CF
20 × 1 LCD	80	81	82	83	Through	93			
20 × 2 LCD	80	81	82	83	Through	93			
	C0	C0	C2	C3	Through	D3			
20 × 4 LCD	80	81	82	83	Through	93			
	C0	C0	C2	C3	Through	D3			
	94	95	96	97	Through	A7			
	D4	D5	D6	D7	Through	E7			
20 × 2 LCD	80	81	82	83	Through	A7			
	C0	C0	C2	C3	Through	E7			

Phân khe thời gian của LCD :



t_{Pwh} = Enable pulse width = 450 ns (minimum)
t_{DSW} = Data set up time = 195 ns (minimum)
t_H = Data hold time 10 ns (minimum)
t_{AS} = Set up time prior to E (going high) for both RS and R/W = 140 ns (minimum)
t_{AH} = Hold time afterr E has come down for both RS and R/W = 10 ns (minimum)

Sơ đồ kết nối với vi điều khiển :


Microcontroller
Can be connected to Ground
In 4-bit mode is left unconnected
+5V
5K
Contrast
330R
+5V
LED backlight
1
14
K
A
GND
VCC
VEE
RS
R/W
E
D0
D1
D2
D3
D4
D5
D6
D7
LCD

2. Cách cấu hình cho LCD trong Code Vision cho Atmega32

Code Vision đã hỗ trợ chúng ta thư viện giao tiếp với LCD qua chế độ 4 bit thông qua file lcd.h. Sau đây là các hàm thông dụng :

void lcd_write_byte(unsigned char addr, unsigned char data);
Hàm này gửi 1 byte tới LCD

void lcd_gotoxy(unsigned char x, unsigned char y);
Hàm này đặt giá trị con trỏ của LCD tới vị trí x,y

void lcd_clear(void);
Hàm này dùng để xóa màn hình hiển thị của LCD

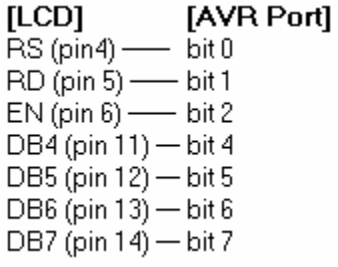
void lcd_putchar(char c);
Hàm này dùng để hiển thị 1 kí tự lên LCD

void lcd_puts(char *str);
Hàm này dùng để hiển thị 1 chuỗi kí tự (chứa trong RAM) lên LCD

void lcd_putsf(char flash *str);
Hàm này dùng để hiển thị 1 chuỗi kí tự (chứa trong Flash) lên LCD

unsigned char lcd_init(unsigned char lcd_columns);
Hàm này dùng để khởi tạo LCD

Sơ đồ kết nối giữa LCD với 1 port của vi điều khiển đã được mặc định trong thư viện như sau :



Các bước cấu hình cho LCD :

- Bước 1 : Định nghĩa các chân cho LCD
- Bước 2 : Khởi tạo LCD : *lcd_init();*
- Bước 3 : Viết lệnh cần thiết : *lcd_puts("…"), lcd_gotoxy(x,y),…*

3. Ví dụ

Trong ví dụ sau, chúng ta sẽ hiển thị dòng chữ "LOP HOC VKD AVR" lên LCD, LCD nối vào port B, sơ đồ kết nối đã chỉ ra như trên.

Chương trình :

```
#include <mega16.h>
#include <delay.h>
#include <lcd.h>

#asm
    .equ __lcd_port=0x18
#endasm


void main(){
    lcd_init(16);
    lcd_putsf("LOP HOC VDK AVR");

    while (1);
}
```

Bài tập

Các hàm có sẵn trong thư viện chỉ hỗ trợ chúng ta hiển thị kí tự lên LCD, bây giờ bạn hãy lập trình một hàm sao cho có thể hiển thị số thực, số nguyên lên LCD, đối số truyền vào là số cần hiển thị.

Gợi ý :

Các kí tự hiển thị lên LCD tuân theo chuẩn trong bảng mã ASCII, muốn hiển thị kí tự nào, chúng ta có thể truyền luôn kí tự đó vào hàm lcd_putc() hoặc có thể cho đối số truyền vào là vị trí của kí tự đó trong bảng mã ASCII.

Ví dụ số 1 có vị trí là 49 trong bảng mã ASCII, như vậy muốn hiển thị số 1 lên LCD, chúng ta dùng thể có 2 cách sau :

```
lcd_putchar ('1');
```

Hoặc

```
lcd_putchar (49);
```

Hay :

```
lcd_putchar (48 + 1);
```

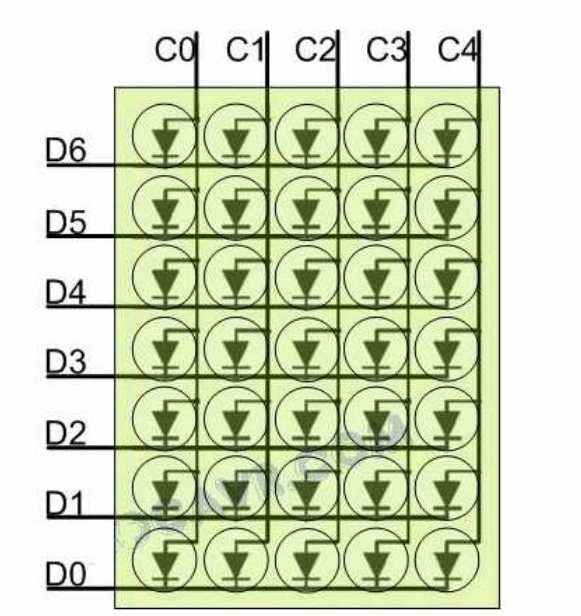
BÀI 7 : GIAO TIẾP VỚI LED MA TRẬN

- *Cơ bản về led ma trận*
- *Cách tạo font cho led ma trận*
- *Ví dụ minh họa*

1. Cơ bản về led ma trận

Led ma trận là một loạt các led đơn được sắp xếp thành các hàng và các cột dạng ma trận, các led có cùng hàng thì sẽ chung 1 chân, chân còn lại nối chung với các led nằm cùng cột.

Ma trận led được ứng dụng rất nhiều trong thực tế, điển hình là các bảng quang báo.



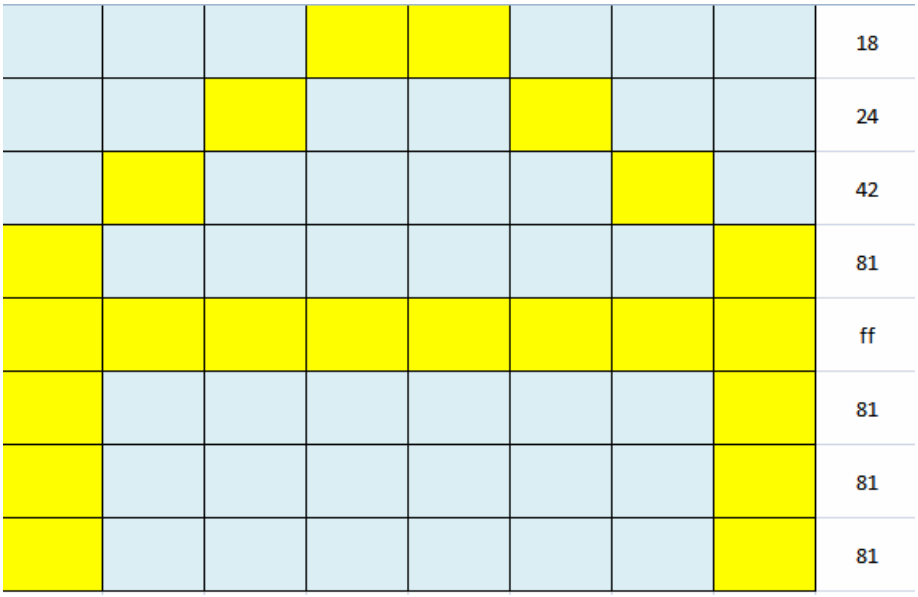
Ma trận LED 7x5.

Để điều khiển led ma trận sáng theo ý muốn, chúng ta sử dụng phương pháp quét led, lợi dụng tính năng lưu ảnh ở mắt người, trong các biển quảng cáo, chúng ta nhìn thấy led sáng liên tục, thực ra không phải vậy, mà là led nhấp nháy liên tục, nhưng do tốc độ cao nên mắt người không kịp phân biệt và kết quả là chúng ta nhìn thấy 1 hình ảnh liên tục.

Có 1 cách quét led ma trận là quét theo hàng và quét theo cột, ví dụ trong bài sẽ trình bày các quét theo hàng (ma trận led chúng ta sử dụng là ma trận kích cỡ 8x8), đây cũng là cách quét led phổ biến hiện nay.

2. Tạo font cho led ma trận

Có rất nhiều phần mềm hỗ trợ chúng ta tạo font cho led ma trận, tuy nhiên, sau đây tác giả sẽ hướng dẫn các bạn sử dụng phần mềm Excel nằm trong bộ Microsoft Office) để tạo bảng font, sau đây là font cho chữ A :



Phương pháp quét led như sau :

- Đầu tiên, chúng ta cho hàng thứ nhất active, các ô ở hàng thứ nhất có giá trị 0x18 (các ô màu vàng tương ứng có giá trị 1, các ô màu xanh nhạt có giá trị 0), như vậy 2 led ở hàng thứ nhất sẽ sáng (tương ứng với 2 ô màu vàng).
- Sau đó chúng ta un-active hàng thứ nhất, toàn bộ các led ở hàng thứ nhất tắt, và cho active hàng thứ 2, cũng tương tự như trên, chúng ta đưa giá trị là 0x24 cho các ô ở hàng thứ 2, kết quả là chúng ta cũng được 2 ô sáng (tương ứng với 2 ô màu vàng) ở hàng thứ 2.
- Tương tự, chúng ta cho sáng lần lượt các hàng với các giá trị như hình vẽ trên.
- Do tốc độ quét nhanh nên mắt chúng ta không phân biệt được sự chuyển động rời rạc của các led. Và kết quả là chúng ta nhìn thấy led sáng thành hình chữ A như hình vẽ.

Các kí tự khác cũng có thể tạo tương tự như trên.

3. Ví dụ minh họa.

Đoạn chương trình sau sẽ làm hiển thị chữ A lên led ma trận, các hàng và các cột được nối tương ứng vào các port B và D :

```
/*****************************************************************************
*       Chuong trinh giao tiep voi led ma tran
*       Tac gia     : pk
*       Mo ta       : Su dung vi dieu khien de giao tiep voi led ma tran
*****************************************************************************/
#include <mega32.h>
#include <delay.h>

/************************************ Define ***************************************/
#define wait_time 1
/************************************ Protype Function *****************************/
/************************************ Chuong trinh chinh ***************************/

char font[] = {0x18,0x24,0x42,0x81,0xFF,0x81,0x81,0x81,      // Chu A
               0xFE,0xC3,0xC3,0xFE,0xFE,0xC3,0xC3,0xFE};    // Chu B

void main(){
   int i,j;

   DDRB = 0xFF;
   DDRD = 0xFF;

   while(1){
      // Chu A
      j = 1;
      for(i = 0;i < 8; i++){
         PORTB = font[i];
         PORTD = ~j;
         j = j*2;
         delay_ms(wait_time);
      }
   }
}
// Het Chuong trinh
```

Bài tập

Dựa vào nguyên lí tạo chữ A ở trên, bạn hãy tạo và viết chương trình hiển thị các kí tự bất kì trong bảng chữ cái

BÀI 8: GIAO TIẾP MÁY TÍNH

- *Cơ bản về giao tiếp RS232.*
- *Cách cấu hình giao tiếp RS232 trong CCS cho PIC16F887*
- *Ví dụ minh họa*

1. Cơ bản về giao tiếp RS232

RS232 là một dạng giao thức, dùng để truyền dữ liệu giữa các thiết bị điện tử có hỗ trợ giao thức này. RS232 là một trong những giao thức ra đời sớm nhất và có thể nói là đơn giản nhất.

Cho đến nay, RS232 vẫn còn được ứng dụng rất nhiều do giao thức đơn giản, độ tin cậy cao, và khoảng cách truyền khá xa, tuy nhiên tốc độ truyền vẫn ở mức khá khiêm tốn so với các giao thức ra đời sau này như USB, SPI, I^2C…

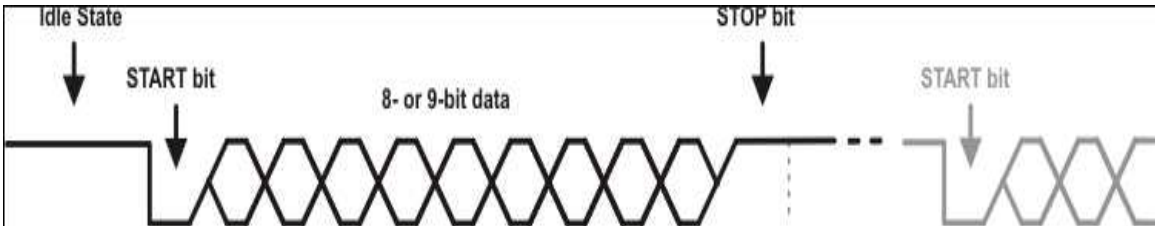
Để sử dụng được giao tiếp RS232, chúng ta sử dụng module UART có sẵn trong Atmega32.

UART là viết tắt của **U**niversal **A**synchronous **R**eceiver **T**ransmitter, là giao tiếp truyền nhận dị bộ, dị bộ ở đây có nghĩa là thiết bị truyền và thiết bị nhận không cùng chung xung nhịp clock.

Trong giao thức RS232, chúng ta quan tâm đến những thông số sau :

- Tốc độ baud : Là số bit truyền trên 1s, điển hình là 9600 bit/s
- Parity : có 2 loại parity là parity chẵn và parity lẻ, dùng để tăng tính kiểm soát lỗi trong 1 lần truyền, giả sử ta cấu hình parity là chẵn thì mỗi lần truyền, nếu số bit có mức logic 1 là lẻ thì module tự thêm 1 bit 1 vào cuối khung truyền, còn nếu số bit có mức logic 1 là chẵn thì không thêm bit 1 vào cuối khung truyền. Parity lẻ cũng tương tự như vậy.
- Số bit trên mỗi lần truyền : Là số bit dữ liệu (data) trên mỗi khung truyền, thường là 8 bit.

Một khung truyền UART có cấu trúc như sau :


Idle State
START bit
8- or 9-bit data
STOP bit
START bit

2. Cách cấu hình module UART trong Code Vision

Để cấu hình sử dụng module UART trong Code Vision, ta sử dụng 4 thanh ghi UCSRA, UCSRB, UCSRC, UBRRH, UBRRL :

UCSRA (USART Control and Status Register A)
UCSRB (USART Control and Status Register B)
UCSRC (USART Control and Status Register C)
UBRRL và **UBRRH** (USART Baud Rate Registers)

Cụ thể về các bit và ý nghĩa của các bit trong các thanh ghi này, các bạn có thể tham khảo phần help của Code Vision.

Một số hàm thông dụng :

char getchar(void)
Hàm này trả về một kí tự nằm trong bộ đệm nhận của Atmega32 (nếu có).

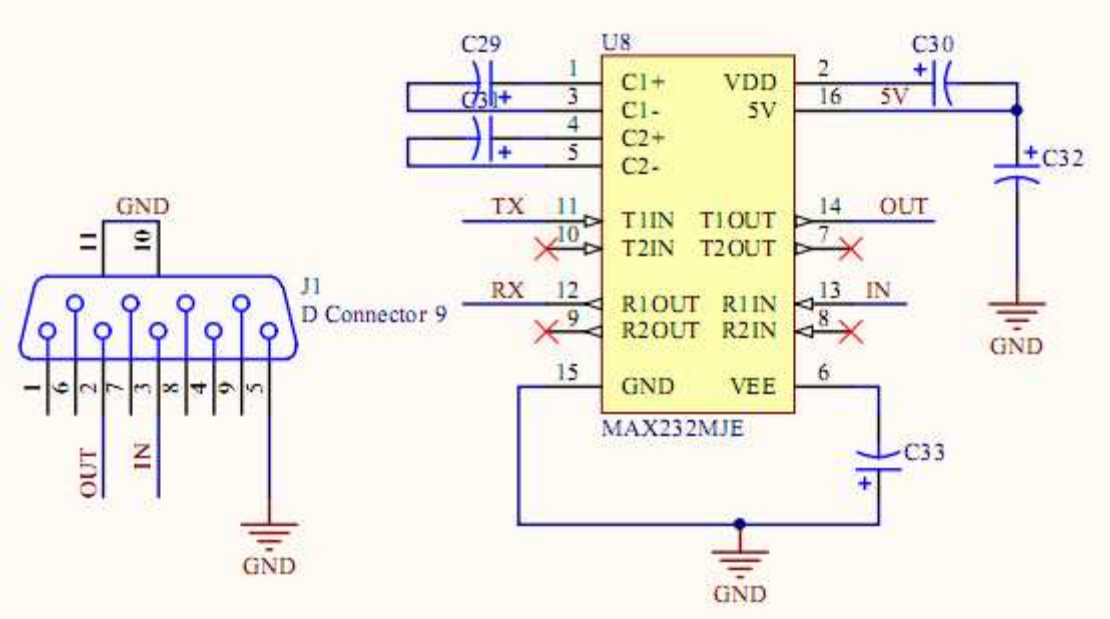
void putchar(char c)
Hàm này truyền một kí tự thông qua module UART

void printf(char flash *fmtstr [, arg1, arg2, ...])
Hàm này giống như hàm trong C, nhưng thay vì in các kí tự lên màn hình (trong C) thì hàm này sẽ gửi dữ liệu thông qua module UART.

3. Ví dụ.

Sau đây là một chương trình minh họa giao tiếp RS232, chương trình sẽ liên tục gửi chuỗi kí tự lên "*Chuong trinh giao tiep RS232*" lên PC.

Mạch nguyên lí :



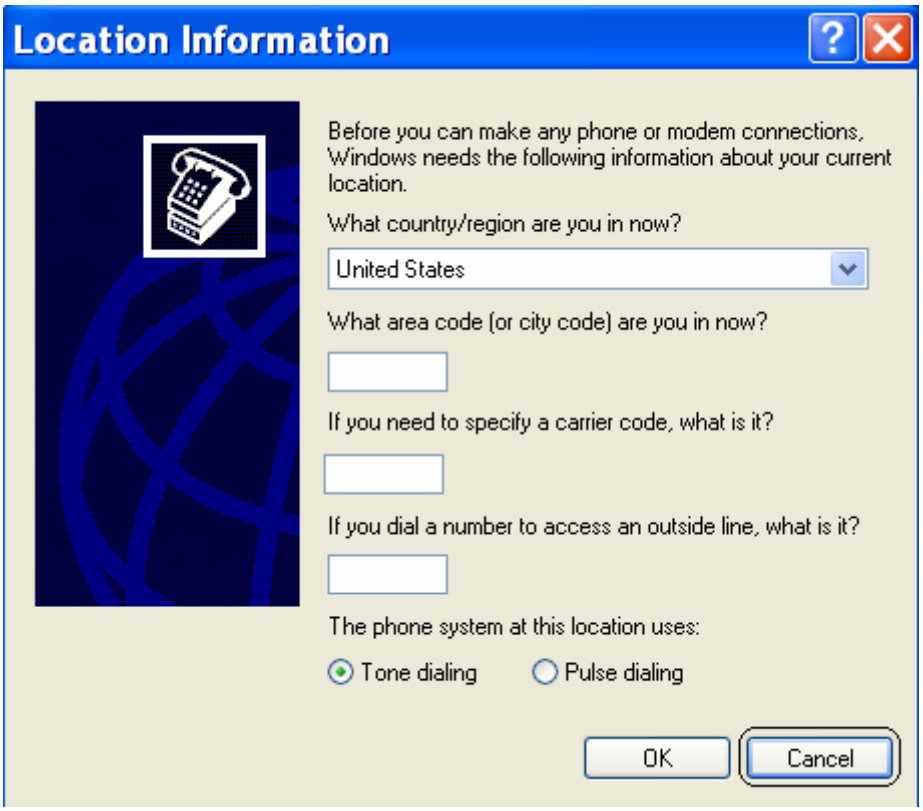
Trong mạch nguyên lí, chúng ta sử dụng thêm 1 IC max 232 để chuyển điện áp tương ứng với 2 mức logic 0 và 1 của vi điều khiển thành điện áp ở mức logic tương ứng với AVR, hai chân 11 và 12 của max232 được nối với 2 chân TX và RX của vi điều khiển.

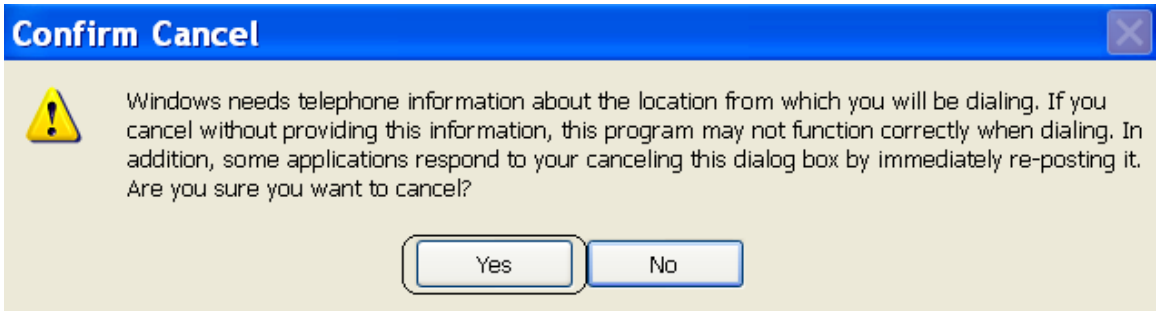
Chương trình :

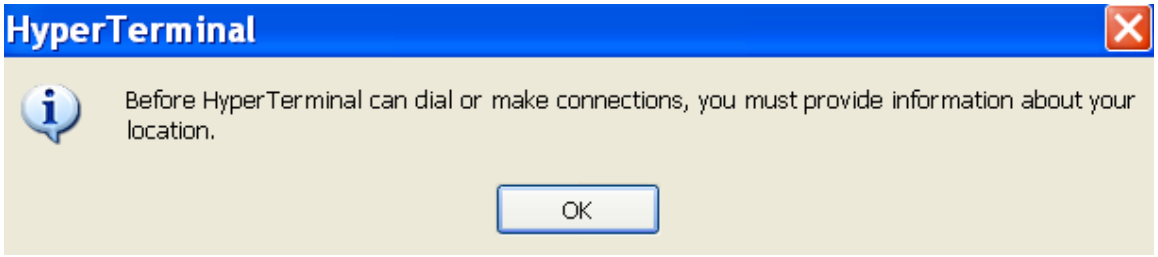
```
#include <mega32.h>
#include <delay.h>

// Standard Input/Output functions
#include <stdio.h>

void main(void)
{
    // USART initialization
    // Communication Parameters: 8 Data, 1 Stop, No Parity
    // USART Receiver: Off
    // USART Transmitter: On
    // USART Mode: Asynchronous
    // USART Baud rate: 9600

    UCSRA=0x00;
    UCSRB=0x08;
    UCSRC=0x86;
    UBRRH=0x00;
    UBRRL=0x67;

    while (1){
        printf("Chuong trinh giao tiep RS232");
        delay_ms(1000);
    }
}
```

Chương trình giao tiếp RS232 rất đơn giản. Để có thể quan sát kí tự được truyền lên PC, chúng ta có thể sử dụng 1 phần mềm có sẵn trong window là Hyper Terminal, để mở phần mềm này, chúng ta làm như sau :

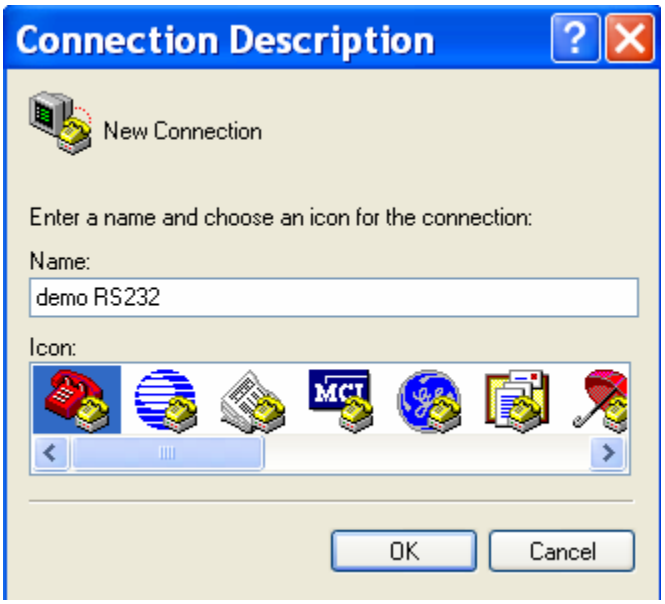
- Vào Start/All Program/Accessories/Communications/Hyper Terminal
- Tiếp đến, xuất hiện hộp thoại nhắc nhở nhập tên thông tin khu vực, chúng ta chọn cancel, sau đó chọn **yes** và **ok**





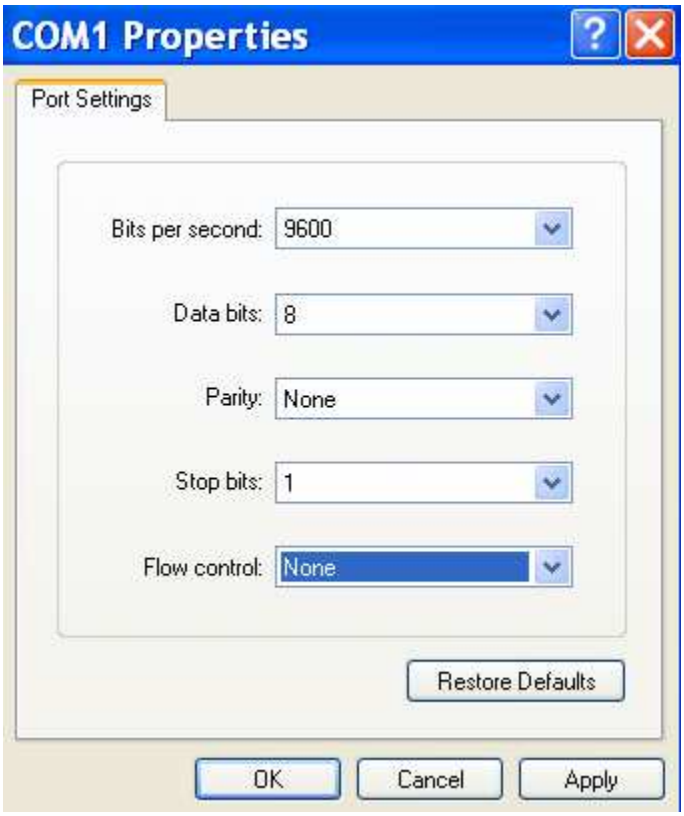


- Sau đó chúng ta nhập mô tả kết nối :



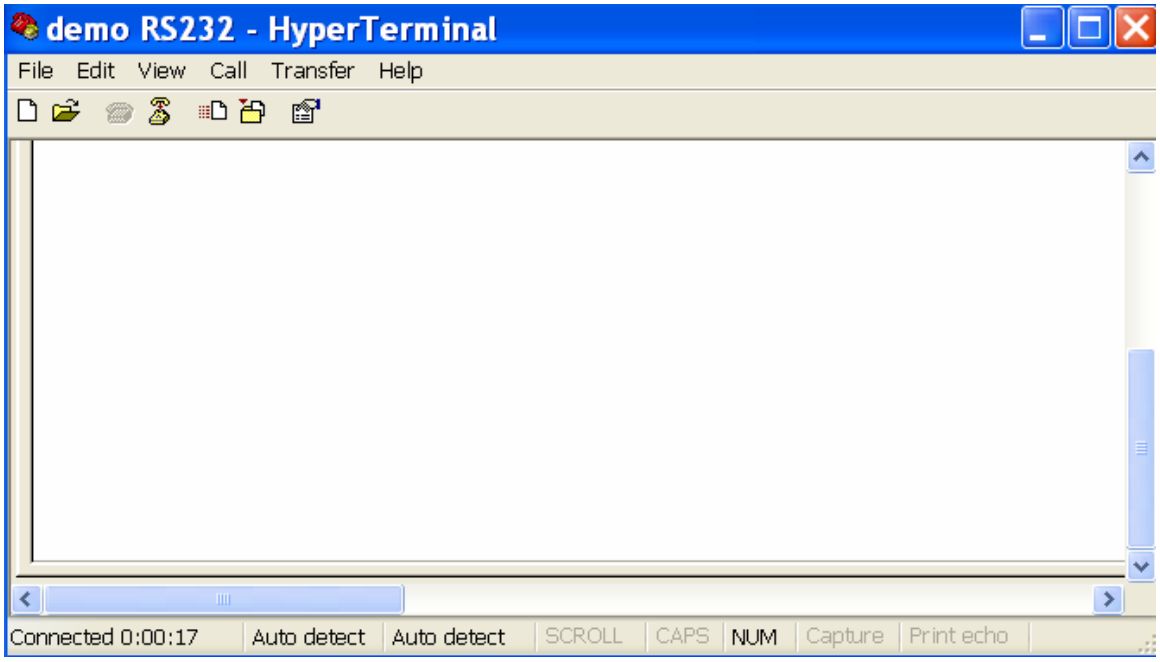
- Nếu lại xuất hiện hộp thoại nhắc nhở nhập tên thông tin khu vực, chúng ta làm như trên.
- Sau đó, chúng ta chọn các thông số để thiết lập kết nối :

Chọn cổng COM để kết nối

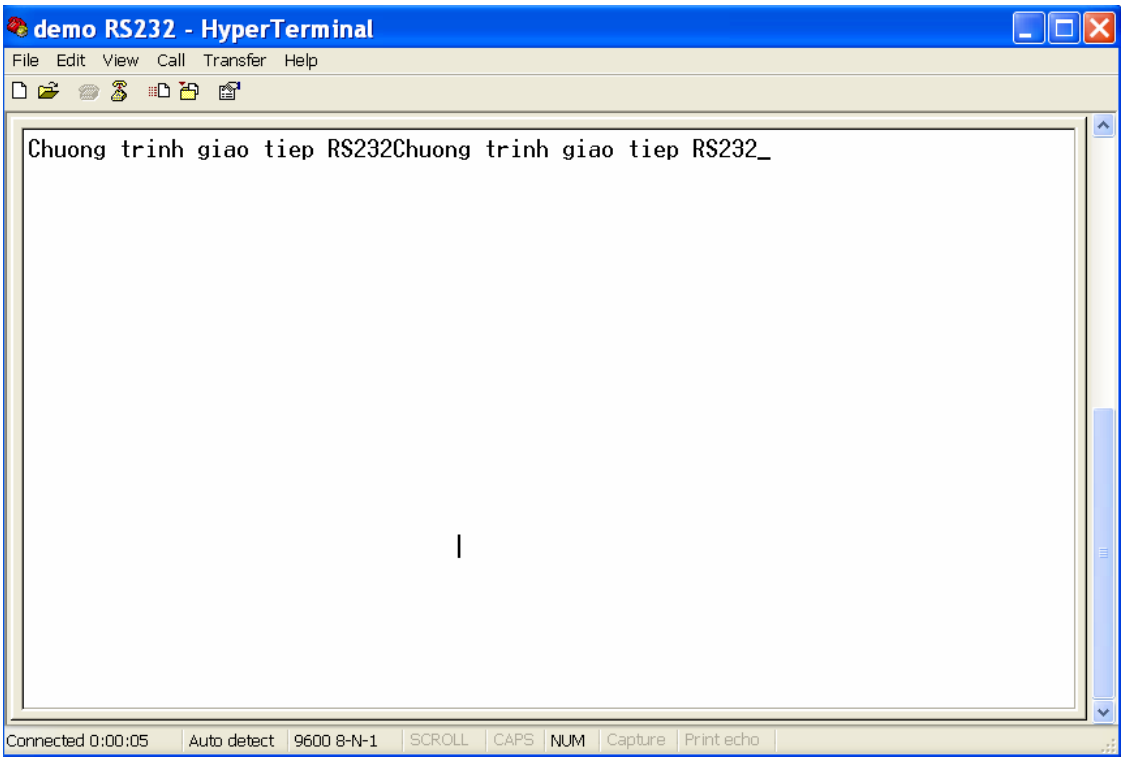


Chọn tốc độ baud, số bit dữ liệu trên 1 khung truyền và parity

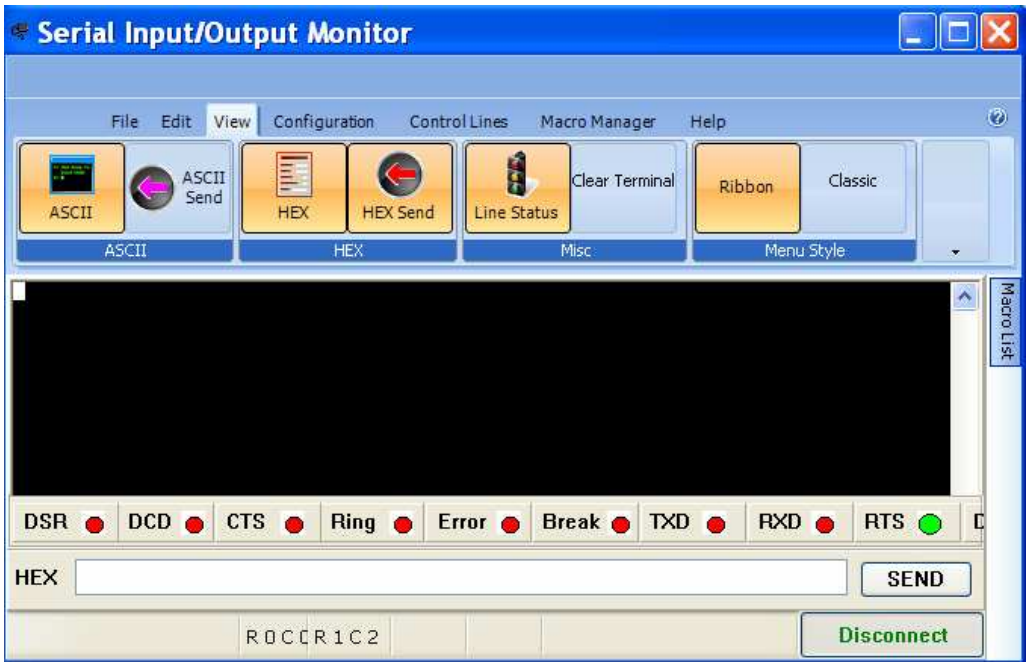
- Giao diện của chương trình như sau :

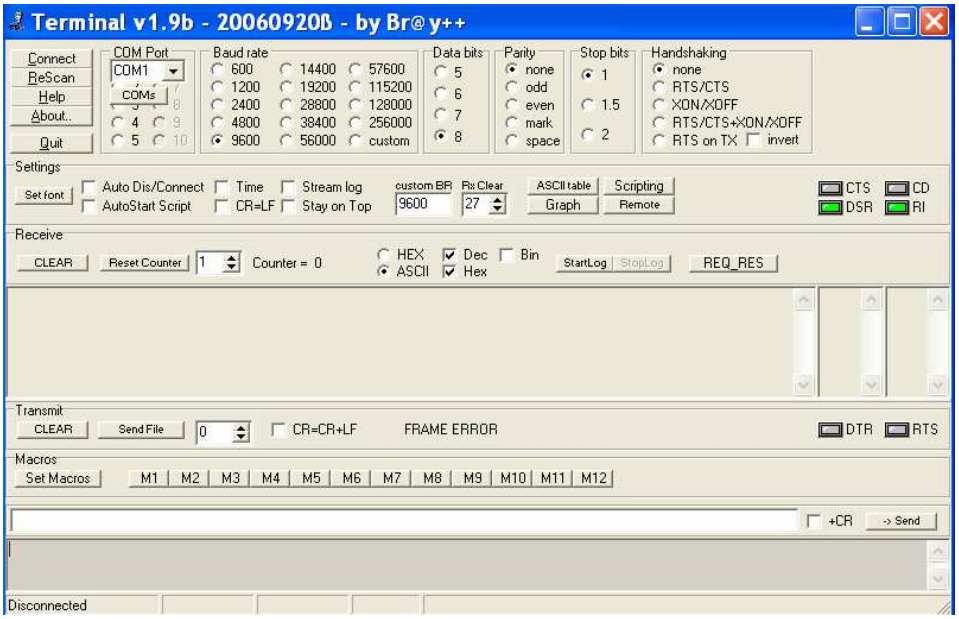


- Kết quả, chúng ta được như sau :



- Chú ý : Ngoài chương trình Hyper Terminal có sẵn trong window, chúng ta có thể sử dụng nhiều chương trình khác như **Terminal**, hay như chương trình có sẵn của CCS như siow (Serial Input/Output Monitor), giao diện các chương trình đó như sau :


Serial Input/Output Monitor
File
Edit
View
Configuration
Control Lines
Macro Manager
Help
ASCII
ASCII Send
HEX
HEX Send
Line Status
Clear Terminal
Ribbon
Classic
ASCII
HEX
Misc
Menu Style
Macro List
DSR
DCD
CTS
Ring
Error
Break
TXD
RXD
RTS
HEX
SEND
Disconnect


Terminal v1.9b - 20060920B - by Br@y++
Connect
ReScan
Help
About..
Quit
COM Port
COM1
COMs
Baud rate
600
1200
2400
4800
9600
14400
19200
38400
56000
57600
115200
128000
256000
custom
Data bits
Parity
none
odd
even
mark
space
Stop bits
1
1.5
2
Handshaking
RTS/CTS
XON/XOFF
RTS/CTS+XON/XOFF
RTS on TX
invert
Settings
Set font
Auto Dis/Connect
AutoStart Script
Time
CR=LF
Stream log
Stay on Top
custom BR
9600
Rx Clear
27
ASCII table
Graph
Scripting
Remote
CTS
DSR
CD
RI
Receive
CLEAR
Reset Counter
Counter = 0
HEX
ASCII
Dec
Hex
Bin
StartLog
StopLog
REQ_RES
Transmit
CLEAR
Send File
CR=CR+LF
FRAME ERROR
DTR
RTS
Macros
Set Macros
M1
M2
M3
M4
M5
M6
M7
M8
M9
M10
M11
M12
+CR
-> Send
Disconnected

Bài tập:

Ví dụ trên mô tả việc gửi dữ liệu lên PC, dựa vào các hàm có sẵn trong CCS như đã giới thiệu trước đó, bạn hãy viết chương trình đọc dữ liệu gửi về từ PC.

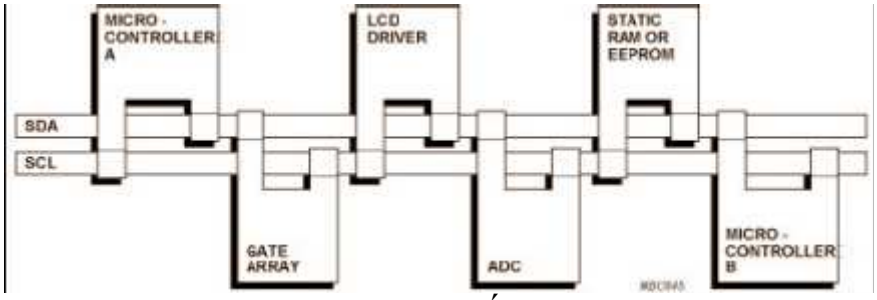
BÀI 9 : GIAO TIẾP I^2C

- *Giới thiệu về giao tiếp I^2C.*
- *Cách cấu hình giao tiếp I^2C trong Code Vision cho Atmega32.*
- *Ví dụ minh họa.*

1. Giới thiệu chung về I2C

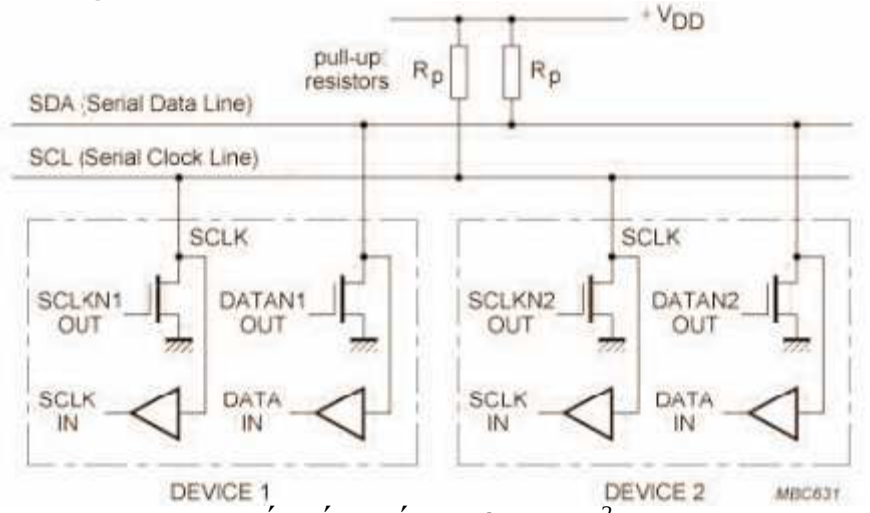
Ngày nay trong các hệ thống điện tử hiện đại, rất nhiều IC hay thiết bị ngoại vi cần phải giao tiếp với các IC hay thiết bị khác – giao tiếp với thế giới bên ngoài. Vói mục tiêu đạt được hiệu quả cho phần cứng tốt nhất với mạch điện đơn giản, Phillips đã phát triển một chuẩn giao tiếp nối tiếp 2 dây được gọi là I^2C. I^2C là tên viết tắt của cụm từ Inter Intergrated. I^2C có tốc độ truyền khá cao, cỡ Mbit/s, tuy nhiên khoảng cách truyền rất ngắn, chỉ khoảng trên board mạch.

I^2C mặc dù được phát triển bởi Philips, nhưng nó đã được rất nhiều nhà sản xuất IC trên thế giới sử dụng. I^2C trở thành một chuẩn công nghiệp cho các giao tiếp điều khiển, có thể kể ra đây một vài tên tuổi ngoài Philips như : Texas Intrument (TI), Maxim Dallas,analog Device, National emiconductor … Bus I^2C được sử dụng làm bus giao tiếp ngoại vi cho rất nhiều loại IC khác nhau như các loại vi điều khiển PIC, AVR, ARM, chip nhớ như RAM tĩnh (Static Ram), EEPROM, bộ chuyển đổi tương tự số (ADC), số tương tụ (DAC)…



Bus I2C và các thiết bị ngoại vi

Một giao tiếp I^2C gồm có 2 dây : Serial Data (SDA) và Serial Clock (SCL). SDA là đường truyền dữ liệu 2 hướng, còn SCL là đường truyền xung đồng hồ và chỉ theo một hướng.

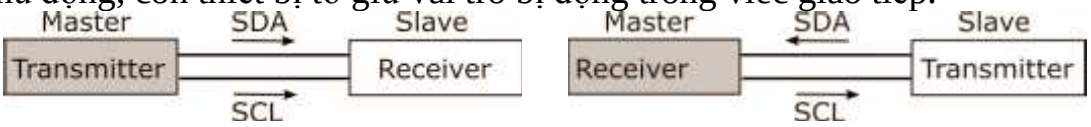


Kết nối thiết bị vào bus I^2C

Mỗi dây SDA hay SCL đều được nối với điện áp dương của nguồn cấp thông qua một điện trở kéo lên (pull up resistor). Sự cần thiết của các điện trở kéo này là vì chân giao tiếp I^2C của các thiết bị ngoại vi thường là dạng cực máng hở (open drain or open collector).Giá trị của các điện trở này khác nhau tùy vào từng thiết bị và chuẩn giao tiếp, thường dao động trong khoảng 1KΩ đến 4.7KΩ.

Ta thấy có rất nhiều thiết bị (ICs) cùng được kết nối vào một bus I^2C, tuy nhiên sẽ không xảy ra chuyện nhầm lẫn giữa các thiết bị, bởi mỗi thiết bị sẽ được nhận ra bởi một địa chỉ duy nhất với một quan hệ chủ/tớ tồn tại trong suốt thời gian kết nối. Mỗi thiết bị có thể hoạt đông như là thiết bị nhận dữ liệu hay có thể vừa truyền vừa nhận. Hoạt động truyền hay nhận còn tùy thuộc vào việc thiết bị đó là chủ (master) hay tớ (slave).

Một thiết bị hay một IC khi kết nối với bus I^2C, ngoài một địa chỉ (duy nhất) để phân biệt, nó còn được cấu hình là thiết bị chủ (master) hay tớ (slave). Tại sao lại có sự phân biệt này ? Vì trên một bus I^2C thì quyền điều khiển thuộc về thiết bị chủ (master). Thiết bị chủ nắm vai trò tạo xung đồng hồ cho toàn hệ thống, khi giữa hai thiết bị chủ/tớ giao tiếp thì thiết bị chủ có nhiệm vụ tạo xung đồng hồ và quản lý địa chỉ của thiết bị tớ trong suốt quá trình giao tiếp. Thiết bị chủ giữ vai trò chủ động, còn thiết bị tớ giữ vai trò bị động trong viêc giao tiếp.



Truyền nhận dữ liệu giữa chủ/tớ

Nhìn hình trên ta thấy xung đồng hồ chỉ có một hướng từ chủ đến tớ, còn luồng dữ liệu có thể đi theo hai hướng, từ chủ đến tớ hay ngược lại tớ đến chủ. Về dữ liệu truyền trên bus I^2C, một bus I^2C chuẩn truyền 8 bit dữ liệu có hướng trên đường truyền với tốc độ là 100Kbits/s – Chế độ chuẩn (Standard mode). Tốc độ truyền có thể lên tới 400Kbits/s – Chế độ nhanh (Fast mode) và cao nhất là 3,4Mbits/s – Chế độ cao tốc (High speed mode).

Một bus I2C có thể hoạt động ở nhiều chế độ khác nhau:

- Một chủ một tớ (one master – one slave)
- Một chủ nhiều tớ (one master – multi slave)
- Nhiều chủ nhiều tớ (Multi master – multi slave)

Dù ở chế độ nào, một giao tiếp I2C đều dựa vào quan hệ chủ/tớ. Giả thiết một thiết bị A muốn gửi dữ liệu đến thiết bị B, quá trình được thực hiện như sau :

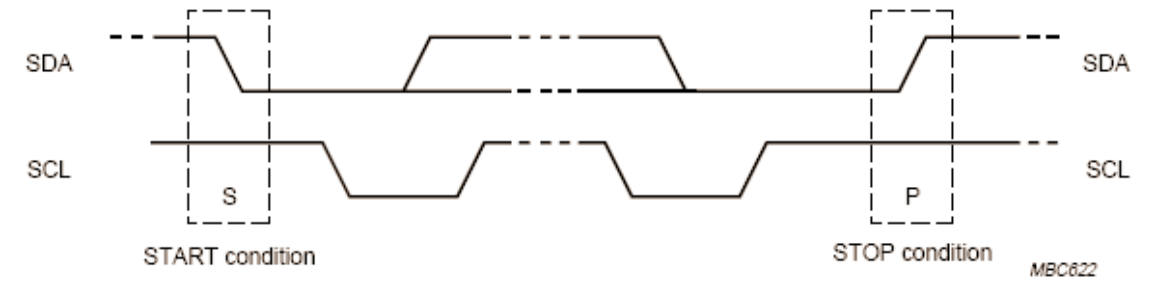
- Thiết bị A (Chủ) xác định đúng địa chỉ của thiết bị B (tớ), cùng với việc xác định địa chỉ, thiết bị A sẽ quyết định việc đọc hay ghi vào thiết bị tớ.
- Thiết bị A gửi dữ liệu tới thiết bị B
- Thiết bị A kết thúc quá trình truyền dữ liệu

Khi A muốn nhận dữ liệu từ B, quá trình diễn ra như trên, chỉ khác là A sẽ nhận dữ liệu từ B. Trong giao tiếp này, A là chủ còn B vẫn là tớ. Chi tiết việc thiết lập một giao tiếp giữa hai thiết bị sẽ được mô tả chi tiết trong các mục dưới đây.

START and STOP conditions

START và STOP là những điều kiện bắt buộc phải có khi một thiết bị chủ muốn thiết lập giao tiếp với một thiết bị nào đó trong mạng I^2C. START là điều kiện khởi đầu, báo hiệu bắt đầu của giao tiếp, còn STOP báo hiệu kết thúc một giao tiếp. Hình dưới đây mô tả điều kiện START và STOP.

Ban đầu khi chưa thực hiện quá trình giao tiếp, cả hai đường SDA và SCL đều ở mức cao (**SDA = SCL = HIGH**). Lúc này bus I2C được coi là dỗi ("**bus free**"), sẵn sàng cho một giao tiếp. Hai điều kiện START và STOP là không thể thiếu trong việc giao tiếp giữa các thiết bị I^2C với nhau



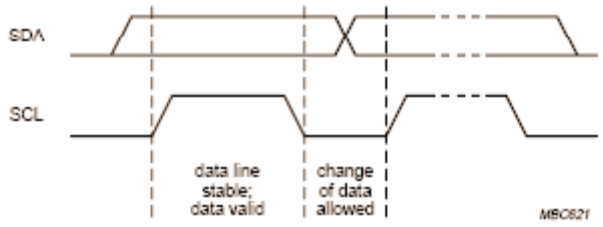
Điều kiện START và STOP của bus I^2C

Điều kiện START : một sự chuyển đổi trạng thái từ cao xuống thấp trên đường SDA trong khi đường SCL đang ở mức cao (cao = 1; thấp = 0) báo hiệu một điều kiện START Điều kiện STOP : Một sự chuyển đổi trạng thái từ mức thấp lên cao trên đường SDA trong khi đường SCL đang ở mức cao.

Cả hai điều kiện START và STOP đều được tạo ra bởi thiết bị chủ. Sau tín hiệu START, bus I^2C coi như đang trong trang thái làm việc (busy). Bus I^2C sẽ rỗi, sẵn sàng cho một giao tiếp mới sau tín hiệu STOP từ phía thiết bị chủ. Sau khi có một điều kiện START, trong qua trình giao tiếp, khi có một tín hiệu START được lặp lại thay vì một tín hiệu STOP thì bus I^2C vẫn tiếp tục trong trạng thái bận. Tín hiệu START và lặp lại START đều có chức năng giống nhau là khởi tạo một giao tiếp.

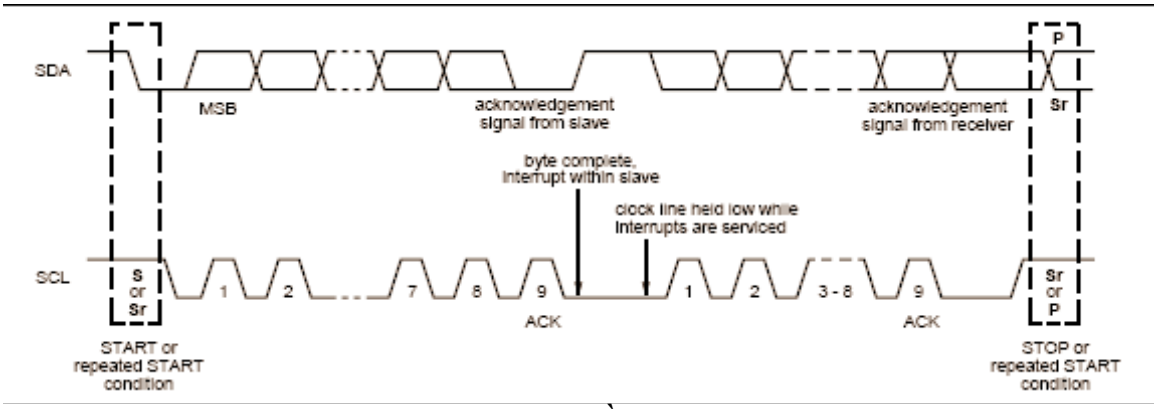
Định dạng dữ liệu truyền

Dữ liệu được truyền trên bus I^2C theo từng bit, bit dữ liệu được truyền đi tại mỗi sườn dương của xung đồng hồ trên dây SCL, quá trình thay đổi bit dữ liệu xảy ra khi SCL đang ở mức thấp.

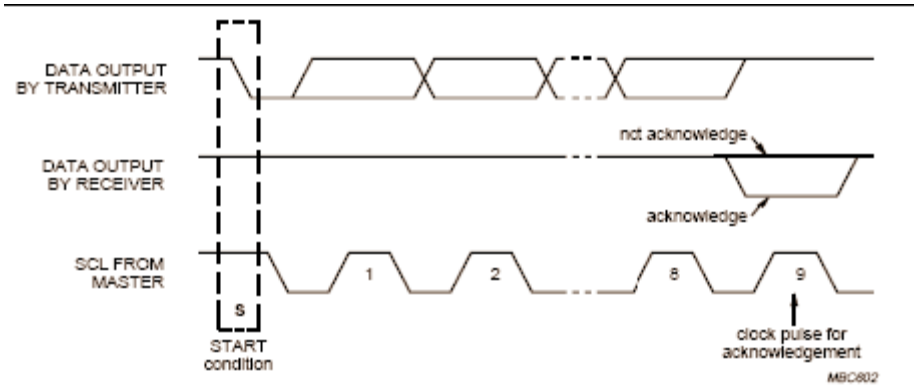


Quá trình truyền 1 bit dữ liệu

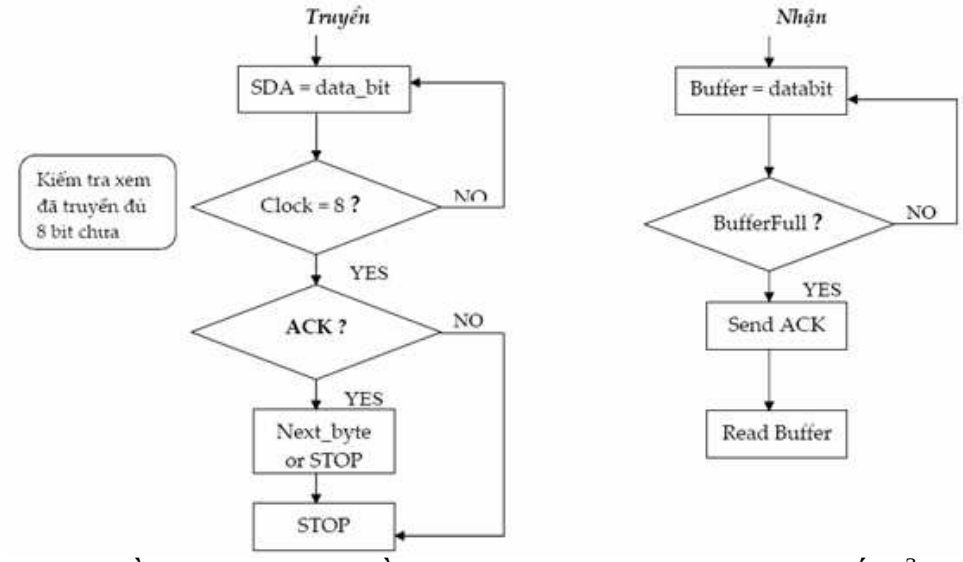
Mỗi byte dữ liệu được truyền có độ dài là 8 bits. Số lượng byte có thể truyền trong một lần là không hạn chế. Mỗi byte được truyền đi theo sau là một bit ACK để báo hiệu đã nhận dữ liệu. Bit có trọng số cao nhất (MSB) sẽ được truyền đi đầu tiên, các bít sẽ được truyền đi lần lượt. Sau 8 xung clock trên dây SCL, 8 bit dữ liệu đã được truyền đi. Lúc này thiết bị nhận, sau khi đã nhận đủ 8 bít dữ liệu sẽ kéo SDA xuống mức thấp tạo một xung ACK ứng với xung clock thứ 9 trên dây SDA để báo hiệu đã nhận đủ 8 bit. Thiết bị truyền khi nhận được bit ACK sẽ tiếp tục thực hiện quá trình truyền hoặc kết thúc.



Dữ liệu truyền trên bus I2C



Bit ACK trên bus I2C



Lưu đồ thuật toán truyền và nhận dữ liệu trong giao tiếp I^2C

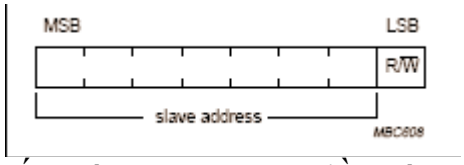
Một byte truyền đi có kèm theo bit ACK là điều kiên bắt buộc, nhằm đảm bảo cho quá trình truyền nhận được diễn ra chính xác. Khi không nhận được đúng địa chỉ hay khi muốn kết thúc quá trình giao tiếp, thiết bị nhận sẽ gửi một xung Not ACK (SDA ở mức cao) để báo cho thiết bị chủ biết, thiết bị chủ sẽ tạo xung STOP để kết thúc hay lặp lại một xung START để bắt đầu quá trình mới.

Định dạng địa chỉ thiết bị

Mỗi thiết bị ngoại vi tham gia vào bus I^2C đều có một địa chỉ duy nhất, nhằm phân biệt giữa các thiết bị với nhau. Độ dài địa chỉ là 7 – bit, điều đó có nghĩa là trên một bus I^2C ta có thể phân biệt tối đa 128 thiết bị. Khi thiết bị chủ muốn giao tiếp với ngoại vi nào trên bus I2C, nó sẽ gửi 7 bit địa chỉ của thiết bị đó ra bus ngay sau xung START. Byte đầu tiên được gửi sẽ bao gồm 7 bit địa chỉ và một bít thứ 8 điều khiển hướng truyền. Mỗi một thiết bị ngoại vi sẽ có một địa chỉ riêng do nhà sản xuất ra nó quy định. Địa chỉ đó có thể là cố định hay thay đổi. Riêng bit điều khiển hướng sẽ quy định chiều truyền dữ liệu. Nếu bit này bằng “0” có nghĩa là byte dữ liệu tiếp theo sau sẽ được truyền từ chủ đến tớ, còn ngược lại nếu bằng “1” thì các byte theo sau byte đầu tiên sẽ là dữ liệu từ con tớ gửi đến con chủ. Việc thiết lập giá trị cho bit này do con chủ thi hành, con tớ sẽ tùy theo giá trị đó mà có sự phản hồi tương ứng đến con chủ.

Truyền dữ liệu trên bus I2C, chế độ Master Slave

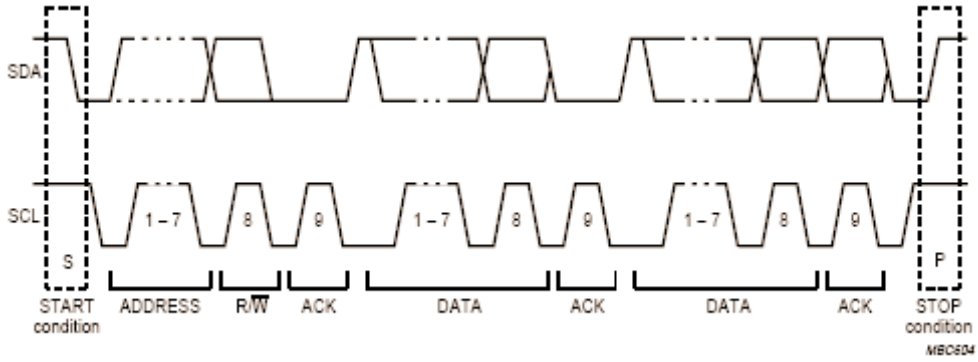
Dữ liệu truyền có thể theo 2 hướng, từ chủ đến tớ hay ngược lại. Hướng truyền được quy định bởi bit thứ 8 trong byte đầu tiên được truyền đi.



Cấu trúc byte dữ liệu đầu tiên

Mỗi một thiết bị ngoại vi sẽ có một địa chỉ riêng do nhà sản xuất ra nó quy định. Địa chỉ đó có thể là cố định hay thay đổi. Riêng bit điều khiển hướng sẽ quy định chiều truyền dữ liệu. Nếu bit này bằng “0” có nghĩa là byte dữ liệu tiếp theo sau sẽ được truyền từ chủ đến tớ, còn ngược lại nếu bằng “1” thì các byte theo sau byte đầu tiên sẽ là dữ liệu từ con tớ gửi đến con chủ. Việc thiết lập giá trị cho bit

này do con chủ thi hành, con tớ sẽ tùy theo giá trị đó mà có sự phản hồi tương ứng đến con chủ.



Quá trình truyền dữ liệu

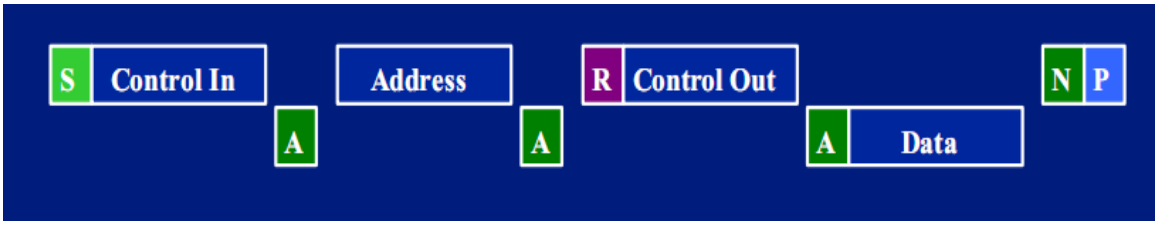
Truyền dữ liệu từ chủ đến tớ (ghi dữ liệu)

Quá trình thực hiện :

- Thiết bị chủ tạo tín hiệu START
- Thiết bị chủ gửi tín hiệu điều khiển (Control In) tới thiết bị tớ, báo hiệu quá trình tiếp theo sẽ là đọc hay ghi dữ liệu. Byte này được qui định bởi nhà sản xuất.
- Nếu nhận được tín hiệu ACK, có nghĩa là quá trình gửi Control In đã thành công, thiết bị chủ tiếp tục gửi địa chỉ cần ghi dữ liệu ở thiết bị tớ.
- Khi tiếp tục nhận được xung ACK báo đã nhận diện đúng thiết bị tớ, thiết bị chủ bắt đầu gửi dữ liệu đến thiết bị tớ theo từng byte một. Theo sau mỗi byte này đều là một xung ACK. Số lượng byte truyền là không hạn chế.
- Kết thúc quá trình truyền, thiết bị chủ sau khi truyền byte cuối sẽ tạo xung STOP báo hiệu kết thúc.

Truyền dữ liệu từ tớ đến chủ (đọc dữ liệu)

Thiết bị chủ muốn đọc dữ liệu từ thiết bị tớ, quá trình thực hiện như sau :



- Khi bus rỗi, thiết bị chủ tạo xung START, báo hiệu bắt đầu giao tiếp.
- Thiết bị chủ gửi tín hiệu điều khiển (Control In) tới thiết bị tớ, báo hiệu quá trình tiếp theo sẽ là đọc hay ghi dữ liệu. Byte này được qui định bởi nhà sản xuất.
- Nếu nhận được tín hiệu ACK, có nghĩa là quá trình gửi Control In đã thành công, thiết bị chủ tiếp tục gửi địa chỉ cần đọc dữ liệu ở thiết bị tớ.
- Sau xung ACK đầu tiên, thiết bị tớ sẽ gửi từng byte ra bus, thiết bị chủ sẽ nhận dữ liệu và trả về xung ACK. Số lượng byte không hạn chế
- Khi muốn kết thúc quá trình giao tiếp, thiết bị chủ gửi xung Not-ACK và tạo xung STOP để kết thúc.

Quá trình kết hợp ghi và đọc dữ liệu: giữa hai xung START và STOP, thiết bị chủ có thể thực hiện việc đọc hay ghi nhiều lần, với một hay nhiều thiết bị. Để thực hiện việc đó, sau một quá trình ghi hay đọc, thiết bị chủ lặp lại một xung START và lại gửi lại địa chỉ của thiết bị tớ và bắt đầu một quá trình mới.

Chế độ giao tiếp Master Slave là chế độ cơ bản trong một bus I2C, toàn bộ bus được quản lý bởi một master duy nhất. Trong chế độ này sẽ không xảy ra tình trạng xung đột bus hay mất đồng bộ xung clock vì chỉ có một master duy nhất có thể tạo xung clock.

Chế độ Multi-Master

Trên bus I^2C có thể có nhiều hơn một master điều khiển bus. Khi đó bus I^2C sẽ hoạt động ở chế độ Multi-Master.

2. Module I^2C trong Atmega32

Với những tiện ích đem lại, khối giao tiếp I^2C đã được tích hợp cứng trong khá nhiều loại vi điều khiển khác nhau. Với những loại Vi điều khiển không có hỗ trợ phần cứng giao tiếp I^2C, để sử dụng ta có thể dùng phần mềm lập trình, khi đó ta sẽ viết một chương trinh diều khiển 2 chân bất kỳ của Vi điều khiển để nó thực hiện giao tiếp I^2C (các hàm START, STOP, WRITE, READ).

Trong Code Vision đã có thư viện hỗ trợ giao tiếp I^2C, đó là file *I2C.H*.

Các hàm hỗ trợ :

void i2c_init(void);

Hàm này khởi tạo module I^2C.

unsigned char i2c_start(void);

Hàm này tạo ra tín hiệu Start cho module I^2C

void i2c_stop(void);

Hàm này tạo ra tín hiệu Start cho module I^2C

unsigned char i2c_read(unsigned char ack);

Hàm này đọc một byte từ bus.

Ack có thể bằng 1 hoặc 0, tương ứng là có trả lại tín hiệu acknowledgement sau khi nhận được byte hay không

unsigned char i2c_write(unsigned char data);

Hàm này ghi một byte lên bus.

Các bước cấu hình module I^2C

- ✓ Định nghĩa chân giao tiếp I^2C

Ví dụ :

```
/* the I2C bus is connected to PORTB */
/* the SDA signal is bit 3 */
/* the SCL signal is bit 4 */
#asm
    .equ __i2c_port=0x18
    .equ __sda_bit=3
    .equ __scl_bit=4
#endasm
```

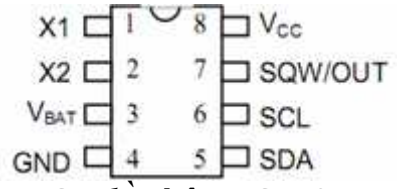
- ✓ Thêm thư viện I^2C vào chương trình
  include <i2c.h>
- ✓ Khởi tạo module I^2C thông qua hàm *i2c_init()*
- ✓ Viết các lệnh cần thiết : read, write, start, stop…

3. Ví dụ

Ví dụ sau sẽ sử dụng module I^2C có trong Atmega32 để giao tiếp với IC thời gian thực DS1307, khi khởi động chúng ta đặt giây trong DS1307 là 5s, sau đó chúng ta đọc lại từ DS1307 rồi hiển thị số giây lên LCD.

Sơ lược về DS1307

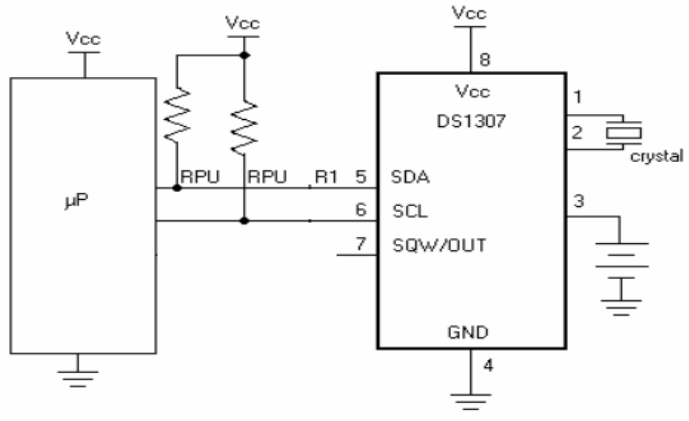
DS1307 là IC thời gian thực, sử dụng giao tiếp I^2C để giao tiếp với các thiết bị khác. Dữ liệu trong DS1307 như giờ, phút, giây… được đặt tại các địa chỉ cố định, được cung cấp bởi nhà sản xuất. Việc đọc hay ghi giờ, phút, giây… chúng ta sẽ đọc/ghi vào các địa chỉ tương ứng.



Sơ đồ chân DS1307

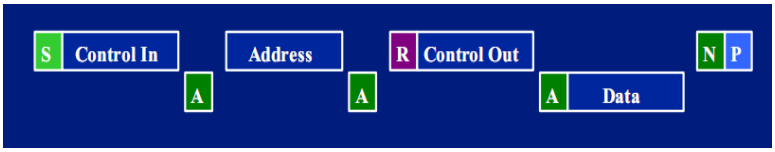
Chi tiết về chức năng và địa chỉ của các dữ liệu trong DS1307, các bạn có thể tham khảo trong datasheet.

Một điều lưu ý là dữ liệu trong DS1307 được lưu dưới dạng số BCD, trong khi đó dữ liệu dùng trong vi điều khiển lại ở dưới dạng số nhị phân, do vậy, trước khi đọc, ghi dữ liệu, chúng ta phải chuyển đổi giữa 2 loại số này cho phù hợp.



Sơ đồ ghép nối DS1307 và vi điều khiển

- Đọc dữ liệu từ DS1307 :




```
unsigned char data;
i2c_start();
i2c_write(0xd0);
i2c_write(address);
i2c_start();
i2c_write(0xd1);
data=i2c_read(0);
i2c_stop();
```

- Ghi dữ liệu vào DS1307 :


```
i2c_start();
i2c_write(0xd0);
i2c_write(address);
i2c_write(data);
i2c_stop();
```

Chương trình :

```
#include <mega32.h>
#include <delay.h>
#include <lcd.h>

#asm
    .equ __lcd_port = 0x12

    .equ __i2c_port = 0x18
    .equ __sda_bit = 3
    .equ __scl_bit = 4
#endasm

#include <i2c.h>
//=============================================================================
unsigned char rtc_read(unsigned char address);
void rtc_write(unsigned char address,unsigned char data);
//=============================================================================
void main(){
    int sec;

    lcd_init(16);
    i2c_init();

    rtc_write(0x00,5);

    while(1){
        sec = rtc_read(0x00);
        lcd_clear();
        lcd_putchar(sec/10 + 48);
        lcd_putchar(sec%10 + 48);
        delay_ms(200);
    }
}

//=============================================================================
```

```
// Doc du lieu tu DS1307
unsigned char rtc_read(unsigned char address){
   unsigned char data;
   i2c_start();
   i2c_write(0xd0);
   i2c_write(address);
   i2c_start();
   i2c_write(0xd1);
   data=i2c_read(0);
   i2c_stop();
   return data;
}

//=============================================================================
// Ghi du lieu vao DS1307
void rtc_write(unsigned char address,unsigned char data){
   i2c_start();
   i2c_write(0xd0);
   i2c_write(address);
   i2c_write(data);
   i2c_stop();
}
```

Bài tập :

- *Dựa vào chương trình mẫu ở trên, hãy viết chương trình sử dụng DS1307, LCD và các phím bấm cần thiết để làm một lịch vạn niên*

BÀI 10 : ĐỘNG CƠ BƯỚC

- *Cơ bản về động cơ bước.*
- *Các mạch điều khiển động cơ bước*
- *Ví dụ minh họa*

1. Cơ bản về động cơ bước

Động cơ bước là loại động cơ đơn giản, có độ chính xác cao, điều khiển dễ dàng, kích thước nhỏ gọi và được ứng dụng rất rộng rãi trong các lĩnh vực điều khiển chuyển động, các động cơ dùng trong đầu đĩa CD, trong ổ cứng... hầu hết là các động cơ bước.

Động cơ bước hiện nay đã đạt tới độ chính xác rất cao, có thể quay $1,8^o$ mỗi bước.

Các đặc điểm chính của động cơ bước :

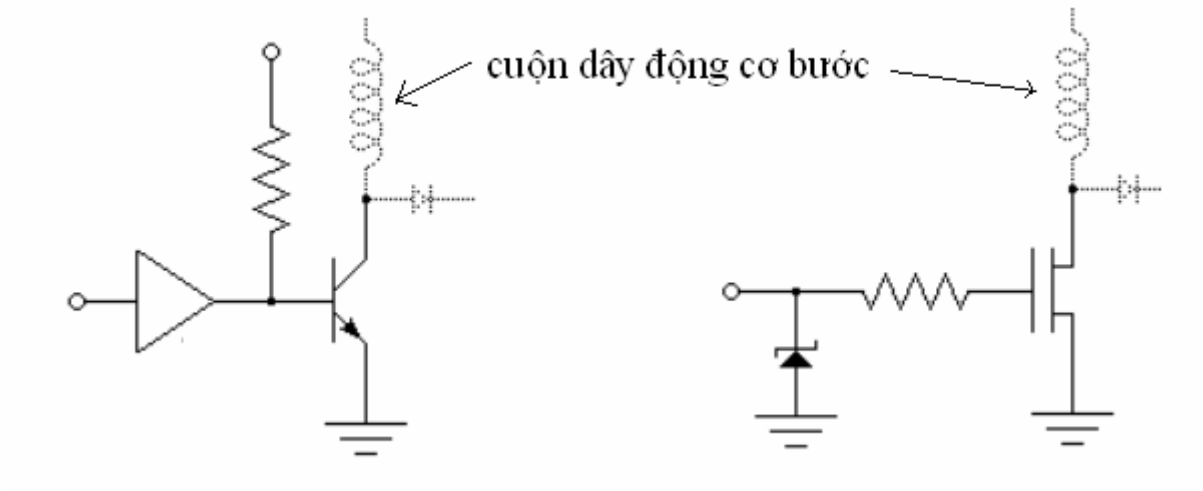
- **Không chổi than :** Không xảy ra hiện tượng đánh lửa chổi than làm tổn hao năng lượng, tại một số môi trường đặc biệt (hầm lò...) có thể gây nguy hiểm.
- **Tạo được mômen giữ :** Một vấn đề khó trong điều khiển là điều khiển động cơ ở tốc độ thấp mà vẫn giữ được mômen tải lớn. Động cơ bước là thiết bị làm việc tốt trong vùng tốc độ nhỏ. Nó có thể giữ được mômen thậm chí cả vị trí nhừ vào tác dụng hãm lại của từ trường rotor.
- **Điều khiển vị trí theo vòng hở :** Một lợi thế rất lớn của động cơ bước là ta có thể điều chỉnh vị trí quay của roto theo ý muốn mà không cần đến phản hồi vị trí như các động cơ khác, không phải dùng đến encoder hay máy phát tốc (khác với servo).
- **Độc lập với tải :** Với các loại động cơ khác, đặc tính của tải rất ảnh hưởng tới chất lượng điều khiển. Với động cơ bước, tốc độ quay của rotor không phụ thuộc vào tải (khi vẫn nằm trong vùng momen có thể kéo được). Khi

momen tải quá lớn gây ra hiện tượng **trượt**, do đó không thể kiểm soát được góc quay.

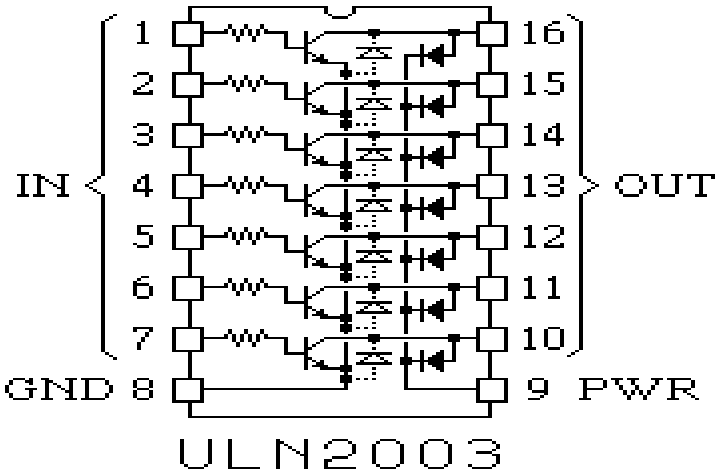
2. Các mạch điều khiển động cơ bước

Có 3 cách điều khiển động cơ : điều khiển đủ bước, nửa bước và vi bước. Độ chính xác tăng dần theo thứ tự trên.

Xét về cấu tạo thì động cơ bước cũng có cấu tạo gồm các cuộn dây, mạch điều khiển động cơ bước gần giống với mạch điều khiển của các thiết bị như relay, động cơ 1 chiều…



Nếu sử dụng mạch có nguyên lí như trên, chúng ta có thể sử dụng 1 IC tích hợp sẵn như ULN2003, IC họ ULN200x có đầu vào phù hợp TTL, các đầu emitor được nối với chân 8.

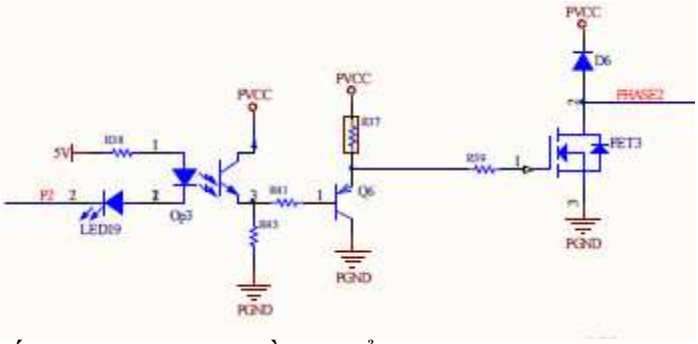


Mỗi transitor darlington được bảo vệ bởi hai diode. Một mắc giữa emitor tới collector chặn điện áp ngược lớn đặt lên transitor. Diode thứ hai nối collector với

chân 9. Nếu chân 9 nối với cực dương của cuộn dây, tạo thành mạch bảo vệ cho transitor.

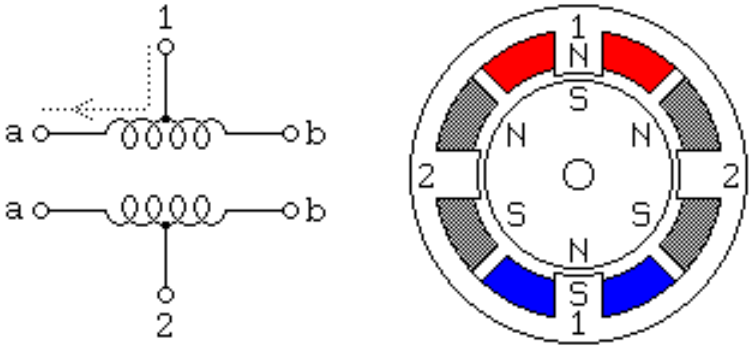
Ngoài ra, có nhiều IC tích hợp sẵn dùng để điều khiển động cơ bước, phổ biến là cặp IC L297 và L298, chuyên dụng để điều khiển động cơ bước với nguyên lí sử dụng mạch cầu H (L298), IC L297 cho phép chúng ta chọn chế độ điều khiển nửa bước hoặc đủ bước.

Động cơ bước trong kit thí nghiệm là động cơ 6 dây, trong đó có 2 dây nguồn và 4 dây pha, chiều quay của động cơ phụ thuộc vào thứ tự điện áp cấp cho các pha này, sau đây là sơ đồ nguyên lí điều khiển 1 pha :



Cực P2 được nối vào chân vi điều khiển, chúng ta sử dụng opto để cách li giữa phần công suất và phần điều khiển, điện áp cấp cho các pha của động cơ được điều khiển thông qua FET. Các pha khác có sơ đồ nguyên lí tương tự hình trên.

Việc nhận biết các đầu dây rất đơn giản, chúng ta cùng xem qua sơ đồ sau :



Chúng ta chỉ việc đo điện trở giữa các đầu dây với nhau, đầu dây nào thông với 2 đầu dây khác và điện trở dây dẫn giữa đầu dây đó với 2 đầu dây còn lại bằng nhau thì đó là đầu số 1 hoặc đầu số 2, hai đầu này có vai trò như nhau nên không cần phân biệt 2 đầu này.

Giờ ta phải xác định thứ tự cấp điện áp vào các đầu dây a,b để điều khiển động cơ quay. Chúng ta nối nguồn vào 2 đầu chung 1,2, sau đó lần lượt cấp điện

áp vào các đầu dây còn lại, cho tới khi đạt tới 1 thứ tự cấp điện áp nào đó mà động cơ chỉ quay theo 1 chiều thì chúng ta ghi lại thứ tự đó và coi như đó là thứ tự chuẩn để điều khiển động cơ, muốn động cơ quay theo chiều ngược lại, chúng ta chỉ việc cấp điện áp vào 4 đầu dây theo thứ tự ngược lại.

3. Ví dụ

Chương trình sau sẽ điều khiển động cơ bước 6 đầu dây quay theo 1 chiều cố định, các đầu dây được nối vào Port D (Xem phần define trong chương trình).

```
#include <mega32.h>
#include <delay.h>

#define time_delay 1

#define P0   PORTD.6
#define P1   PORTD.4
#define P2   PORTD.2
#define P3   PORTD.0

void main(){

    DDRD = 0xFF;

    while(1){
        P2 = 0;
        delay_ms(time_delay);
        PORTD = 0xFF;

        P1 = 0;
        delay_ms(time_delay);
        PORTD = 0xFF;

        P3 = 0;
        delay_ms(time_delay);
        PORTD = 0xFF;

        P0 = 0;
        delay_ms(time_delay);
        PORTD = 0xFF;
    }
}
```

Chủ yếu chúng ta điều khiển ở 2 chế độ là bước đủ và nửa bước, chế độ vi bước chỉ sử dụng khi yêu cầu độ chính xác cao.

Ở chế độ bước đủ, chúng ta lần lượt cấp xung vào các pha của động cơ, còn ở chế độ nửa bước, chúng ta cấp cùng 1 lúc xung vào 2 pha kế tiếp nhau của động cơ.

Tốc độ quay của động cơ bước phụ thuộc vào thời gian chuyển giữa 2 lần cấp xung kế tiếp nhau vào các đầu dây. Trong chương trình trên, thời gian cấp xung là *time_delay.*

Bài tập

Chương trình trong ví dụ điều khiển động cơ bước theo chế độ bước đủ (cấp xung vào 1 cuộn dây tại 1 thời điểm), bạn hãy viết chương trình điều khiển động cơ bước theo chế độ nửa bước. (cấp xung vào 2 cuộn dây kế tiếp nhau tại một thời điểm).

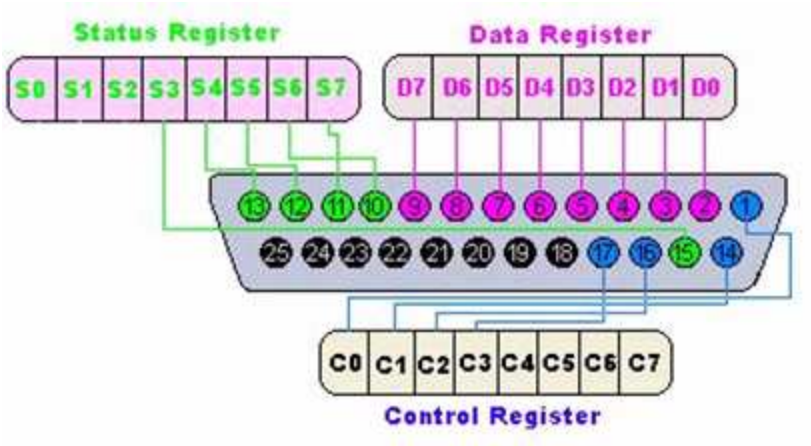
BÀI 11 : GIAO TIẾP VỚI CỔNG LPT

- *Cơ bản về cổng LPT*
- *Ví dụ minh họa*

1. Cơ bản về cổng LPT

LPT là viết tắt của chữ **L**ine **P**rint **T**erminal, giao tiếp LPT là giao tiếp song song nhằm mục đích nối máy tính PC với máy in. Về sau, cổng song song đã phát triển thành một tiêu chuẩn không chính thức. Tên gọi của cổng song song bắt nguồn từ kiểu dữ liệu truyền qua cổng này : các bit dữ liệu được truyền song song hay nói cụ thể hơn là byte nối tiếp còn bit song song.

Cho đến nay cổng song song có mặt ở hầu hết các máy tính PC được sản xuất trong những năm gần đây. Cổng song song còn được gọi là cổng máy in hay cổng Centronics. Cấu trúc của cổng song song rất đơn giản với tám đường dữ liệu, một đường dẫn mass chung, bốn đường dẫn điều khiển để chuyển các dữ liệu điều khiển tới máy in và năm đường dẫn trạng thái của máy in ngược trở lại máy tính. Giao diện song song sử dụng các mức logic TTL, vì vậy việc sử dụng trong mục đích đo lương và điều khiển có phần đơn giản.



Sơ đồ cổng LTP

Khoảng cách cực đại giữa cổng song song máy tính PC và thiết bị ngoại vi bị hạn chế vì điện dung kí sinh và hiện tượng cảm ứng giữa các đường dẫn có thể làm biến dạn tín hiệu. Khoảng cách giới hạn là 8m, thông thường chỉ cỡ 1,5 – 2 m. Khi khoảng cách ghép nối trên 3m nên xoắn các đường dây tín hiệu với đường nối đất theo kiểu cặp dây xoắn hoặc dùng loại cáp dẹt nhiều sợi trong đó mỗi đường dẫn dữ liệu điều nằm giữa hai đường nối mass.

Tốc độ truyền dữ liệu qua cổng song song phụ thuộc vào linh kiện phần cứng được sử dụng. Trên lý thuyết tốc độ truyền đạt giá trị 1 Mbit/s, nhưng với khoảng cách truyền bị hạn chế trong phạm vi 1m. Với nhiều mục đích sử dụng thì khoảng cách này đã hoàn toàn thõa đáng. Nếu cần truyền trên khoảng cách xa hơn, ta nên nghĩ đến khả năng truyền qua cổng nối tiếp hoặc USB. Một điểm cần lưu ý là : việc tăng khoảng cách truyền dữ liệu qua cổng song song không chỉ làm tăng khả năng gây lỗi đối với đường dữ liệu được truyền mà còn làm tăng chi phí của đường dẫn.

Sau đây là chức năng của các đường dẫn tín hiệu:

Strobe (1)

Với một mức logic thấp ở chân này, máy tính thông báo cho máy in biết có một byte đang sẵn sàng trên các đường dẫn tín hiệu để được truyền.
D0 đến D7

Các đường dẫn dữ liệu

Acknowledge

Với một mức logic thấp ở chân này, máy in thông báo cho máy tính biết là đã nhận được kí tự vừa gửi và có thể tiếp tục nhận.

Busy (bận – 11)

Máy in gửi đến chân này mức logic cao trong khi đang đón nhận hoặc in ra dữ liệu để thông báo cho máy tính biết là các bộ đệm trong máy tính biết là các bộ đệm trong máy tính đã bị đầy hoặc máy in trong trạn thái Off-line.

Paper empty (hết giấy – 12)

Mức cao ở chân này có nghĩa là giấy đã dùng hết.

Select (13)

Một mức cao ở chân này, có nghĩa là máy in đang trong trạng thái kích hoạt (On-line)

Auto Linefeed (tự nạp dòng)

Có khi còn gọi là Auto Feed. Bằng một mức thấp ở chân này máy tính PC nhắc máy in tự động nạp một dòng mới mỗi khi kết thúc một dòng.
Error (có lỗi)

Bằng một mức thấp ở chân này, máy in thông báo cho máy tính là đã xuất hiện một lỗi, chẳng hạn kẹt giấy hoặc máy in đang trong trạng thái Off-Line.
Reset (đặt lại)

Bằng một mức thấp ở chân này, máy in được đặt lại trạng thái được xác định lúc ban đầu.
Select Input

Bằng một mức thấp ở chân này, máy in được lựa chọn bởi máy tính.

Cáp nối giữa máy in và máy tính bao gồm 25 sợi, nhưng không phải tất cả điều được sử dụng mà trên thực tế chỉ có 18 sợi được nối với các chân cụ thể. Nhận xét này giúp chúng ta tận dụng những cáp nối mà trong lõi đã bị đứt một hai sợi.

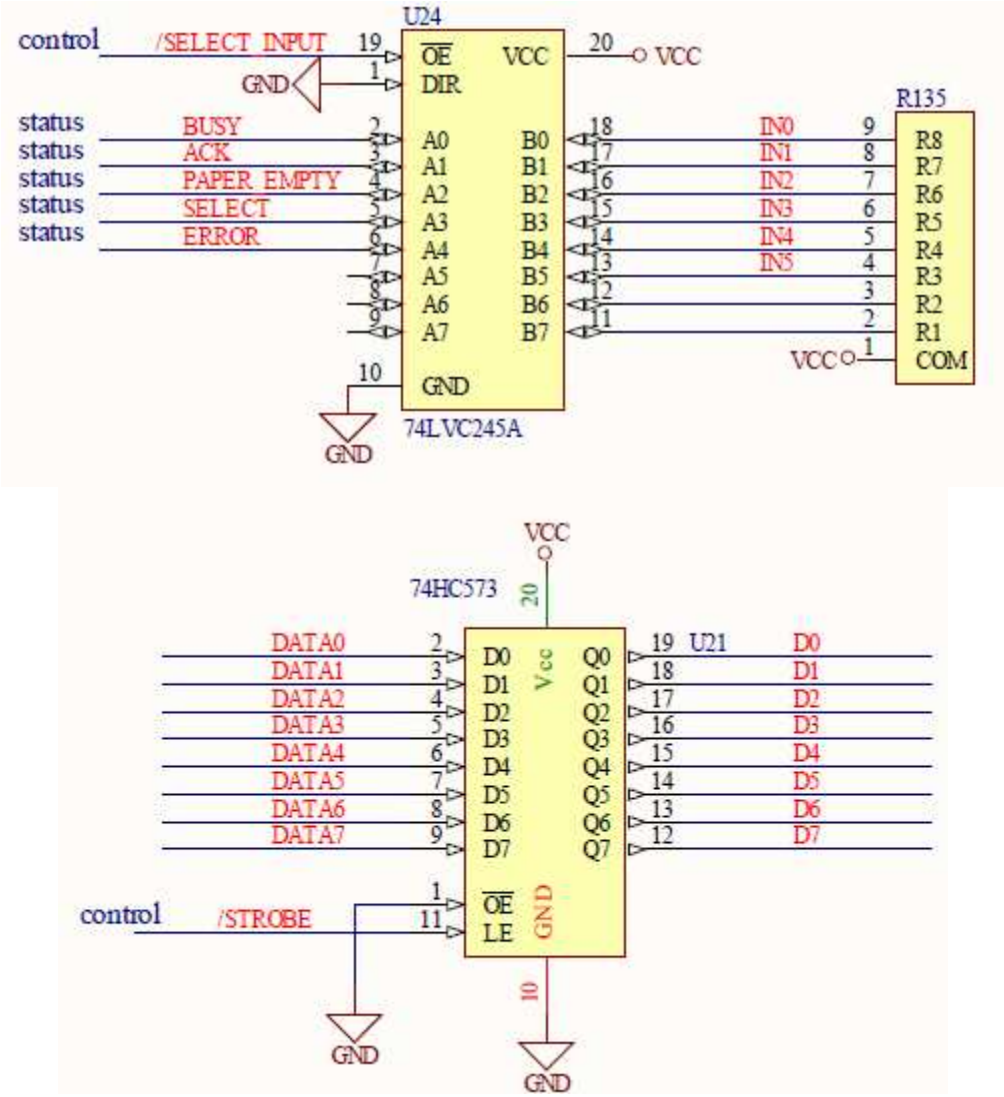
Qua cách mô tả chức năng của từng tín hiệu riêng lẽ ta có thể nhận thấy các đương dẫn dữ liệu có thể chia thành 3 nhóm:

- Các đường dẫn tín hiệu, xuất ra từ máy tính PC và điều khiển máy tính, được gọi là các đường dẫn điều khiển.
- Các đường dẫn tín hiệu, đưa các thông tin thông báo ngược lại từ máy in về máy tính, được gọi là các đường dẫn trạng thái.
- Đường dẫn dữ liệu, truyền các bit rieng lẽ của các ký tự cần in.

Từ cách mô tả các tín hiệu và mức tín hiệu ta có thể nhận thấy là: các tín hiệu Acknowledge, Auto Linefeed, Error, Reset và Select Input kích hoạt ở mức thấp. Thông qua chức năng của các chân này ta cũng hình dung được điều khiển cổng máy in.

2. Ví dụ minh họa

Máy tính sẽ gửi dữ liệu (dạng 8 bit) thông qua các đường data, từ DATA0 đến DATA7. Và sẽ nhận dữ liệu phản hồi từ thiết bị thông qua các đường điều khiển, sau đây là sơ đồ kết nối :



Do hình thức giao tiếp là giao tiếp song song, nên lập trình khá đơn giản, đoạn code sau đây dùng để nhận dữ liệu từ cổng LPT và xuất ra led, led được nối với PORT B, dữ liệu nhận từ cổng LPT được nối vào PORT C.

```
// Chuong trinh giao tiep qua cong LPT
// Tac gia : pk

#include <mega32.h>
#include <delay.h>

void main(){
    DDRB = 0xFF;
    DDRC = 0x00;

    while(1){
        PORTB = PINC;
    }
}
```

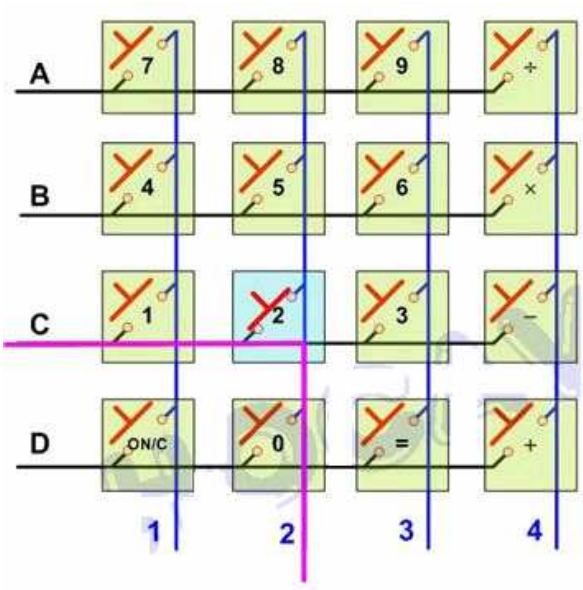
Đoạn mã trên đọc dữ liệu gửi xuống từ cổng LPT (các đường từ D0 đến D7) thong qua Port B, sau đó xuất dữ liệu đó ra Port C. Phần mềm giao tiếp với cổng LPT các bạn có thể tự viết, dùng các ngôn ngữ lập trình như Visual Basic, hay C++, C#...

BÀI 12 : GIAO TIẾP VỚI MA TRẬN PHÍM

- *Cơ bản về ma trận phím*
- *Ví dụ minh họa*

1. Cơ bản về ma trận phím

Giống như led ma trận, ma trận phím là tập hợp các phím đơn, được nối với nhau thành dạng ma trận.



Ma trận phím 4x4

Việc giao tiếp với bàn phím ma trận cũng tương tự như giao tiếp với led ma trận, chúng ta cũng có 2 kiểu là quét theo hang và quét theo cột.

Sau đây chúng ta sẽ cùng tìm hiểu cách quét phím theo hàng :

- Ban đầu, chúng ta cấp điện áp (giả sử là 5V – mức logic 1) vào hang A, các hang còn lại cấp mức logic 0.
- Sau đó, chúng ta kiểm tra mức logic tại các cột 1,2,3,4, nếu cột nào có mức logic 1 thì phím tương ứng ở cột đó được nhấn. Giả sử cột 1 có mức logic 1 thì phím 7 được nhấn.

- Tương tự, chúng ta lần lượt cho các hang B, C, D có mức logic 1, các hang còn lại có mức logic 0, thông qua việc đọc mức logic tại các cột, chúng ta sẽ biết được phím nào được nhấn.

2. Ví dụ minh họa

Sau đây là chương trình minh họa cách quét phím, bàn phím gồm 8 phím được nối vào Port B, giá trị của các phím sau khi đọc được đưa ra port C.

```
#include <mega32.h>
#include <delay.h>

//=====================================================
#define COL_1 PORTB.0
#define COL_2 PORTB.1
#define COL_3 PORTB.2
#define COL_4 PORTB.3

#define ROW_1 PINB.4
#define ROW_2 PINB.5
#define ROW_3 PINB.6
#define ROW_4 PINB.7
//=====================================================
void main(){
    DDRB = 0x0F;
    DDRC = 0xFF;

    PORTC = 0xFF;

    while(1){
        COL_1 = 0;
            if(ROW_1 == 0) PORTC = 1;
            if(ROW_2 == 0) PORTC = 2;
            if(ROW_3 == 0) PORTC = 3;
            if(ROW_4 == 0) PORTC = 4;
        COL_1 = 1;

        COL_2 = 0;
            if(ROW_1 == 0) PORTC = 5;
            if(ROW_2 == 0) PORTC = 6;
            if(ROW_3 == 0) PORTC = 7;
            if(ROW_4 == 0) PORTC = 8;
        COL_2 = 1;
```

```
        COL_3 = 0;
            if(ROW_1 == 0) PORTC = 9;
            if(ROW_2 == 0) PORTC = 10;
            if(ROW_3 == 0) PORTC = 11;
            if(ROW_4 == 0) PORTC = 12;
        COL_3 = 1;

        COL_4 = 0;
            if(ROW_1 == 0) PORTC = 13;
            if(ROW_2 == 0) PORTC = 14;
            if(ROW_3 == 0) PORTC = 15;
            if(ROW_4 == 0) PORTC = 16;
        COL_4 = 1;
    }
}
```

Bài tập

Chương trình trên chỉ đọc giá trị của phím bấm và xuất giá trị (nhị phân) ra Port C, bạn hãy viết chương trình để đọc giá trị của phím và xuất ra led 7 thanh.

BÀI 13 : TIMER

- *Giới thiệu về timer trong Atmega32*
- *Ví dụ minh họa*

1. Giới thiệu về timer

Timer là một trong những module rất quan trọng trong vi điều khiển, sử dụng timer, chúng ta có thể lập trình các tác vụ diễn ra 1 cách chính xác theo thời gian đã định trước, có thể đếm số xung nối vào đầu vào timer…Hầu như tất cả các loại vi điều khiển đều có timer, số lượng timer ở mỗi dòng vi điều khiển có thể khác nhau.

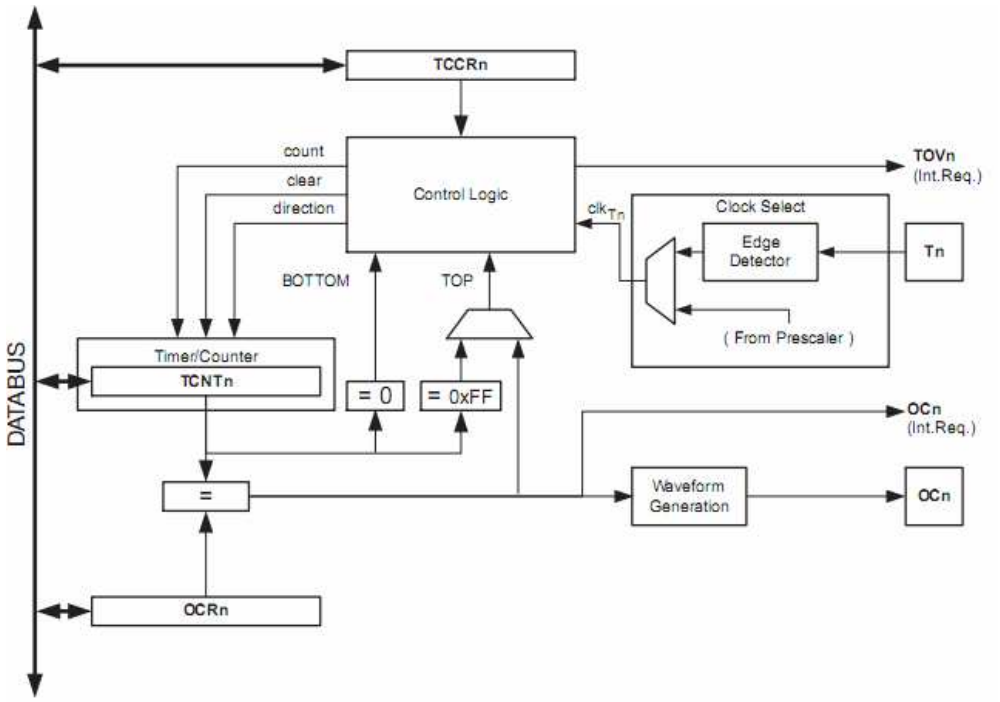
Timer có ứng dụng rộng rãi trong thực tế, giả sử như chúng ta muốn tạo ra 1 khoảng thời gian chính xác là 1ms để làm 1 tác vụ nào đó, hay như chúng ta muốn đếm sản phẩm đi qua băng chuyền, hoặc đếm số xung từ encoder… Tất cả các công việc đó chúng ta có thể hoàn toàn thực hiện bằng timer.

Có 2 chế độ hoạt động đối với timer, chế độ timer và chế độ counter. Với chế độ timer thì xung đưa vào module timer là xung nhịp của hệ thống, còn với chế độ counter thì xung đưa vào module timer là xung bên ngoài.

Trong Atmega32 có 3 timer : Timer 0, timer 1 và timer 2

Timer 0 : Là timer 8 bit, có các tính năng sau :

- Sử dụng làm bộ định thời
- Tự xóa khi đạt tới 1 giá trị đặt trước (Tự động nạp lại)
- Tạo xung
- Đếm sự kiện (Counter)
- Hỗ trợ prescaler



Sơ đồ cấu tạo của Timer 0

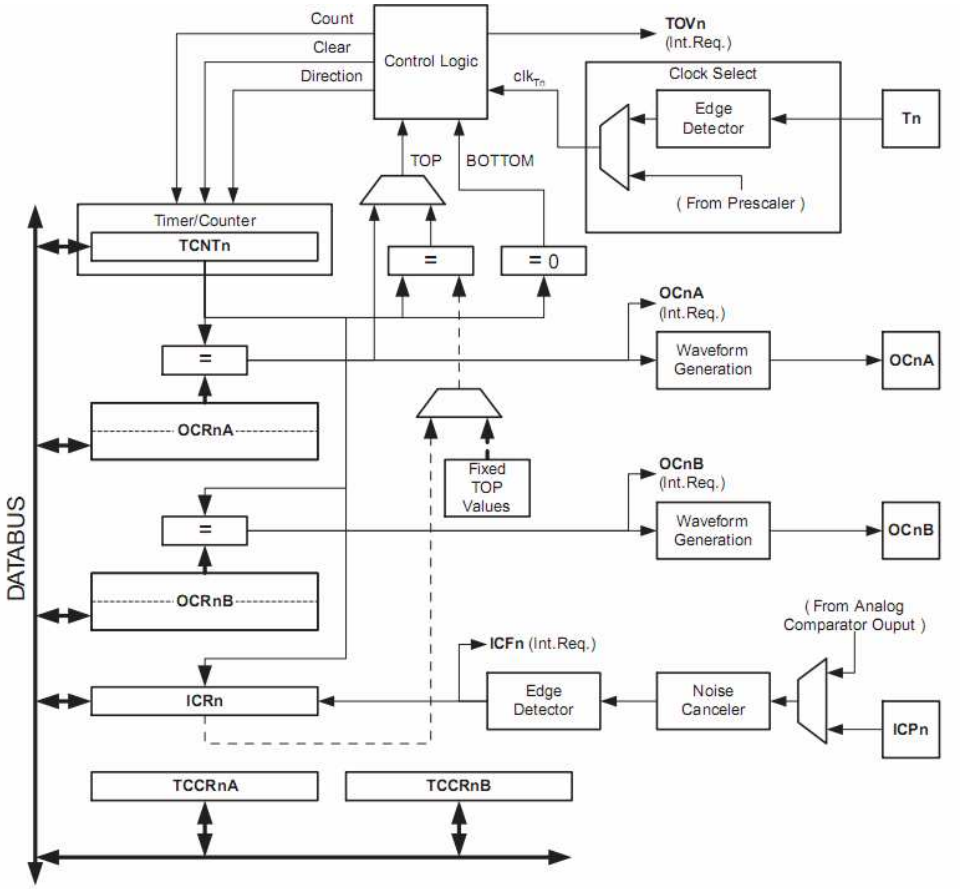
Các chế độ hoạt động được cài đặt trừ thanh thi TCCR0.

Thanh ghi TCNT0 dùng để chứa giá trị của Timer 0, thanh ghi này có thể được đọc hay ghi.

Thanh ghi OCR0 là giá trị đặt trước dùng để so sánh với giá trị trong thanh ghi TCNT0, khi giá trị 2 thanh ghi này bằng nhau sẽ tạo ra 1 tín hiệu ra chân OC0.

Timer 1 : Là Timer 16 bit, có những tính năng sau :

- Có 2 module Output Compare hoạt động độc lập
- Hai thanh ghi đệm chứa giá trị đặt trước
- Một module Capture
- Chế độ tự động nạp lại
- Chế độ PWM với chu kì có thể thay đổi được
- Bộ tạo tần số
- Bộ đếm sự kiện (Counter)
- Có 4 nguồn ngắt độc lập

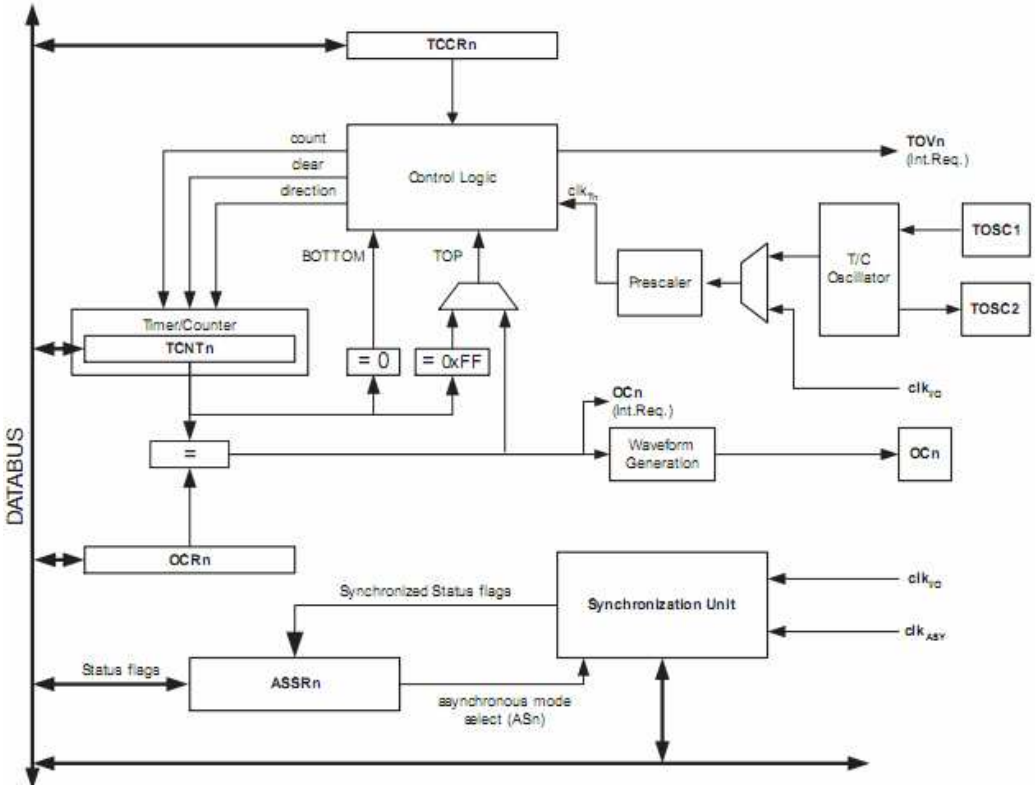


Sơ đồ cấu tạo của Timer 1

Timer 2 : Là timer 8 bit, có các tính năng sau :

- Tự xóa khi đạt tới 1 giá trị đặt trước (Tự động nạp lại)
- Bộ tạo tần số

- Hỗ trợ PWM
- Đếm sự kiện (Counter)
- Hỗ trợ prescaler.
- Hỗ trợ clock input từ thạch anh 32KHz qua chân I/O.



Sơ đồ cấu tạo của Timer 2

Các thanh ghi cho Timer 1 và timer 2 cũng tương tự như timer 0, các bạn có thể tham khảo trong datasheet để biết rõ hơn.

2. Ví dụ minh họa

Ví dụ sau sẽ sử dụng timer để đảo mức logic tại port A, chế độ hoạt động của timer là chế độ đơn giản nhất, chúng ta sử dụng timer 1

Để quan sát giá trị logic tại port A, chúng ta mắc vào đó 8 led đơn.

Chương trình

```
#include <mega32.h>

void main(void){
    unsigned int i = 0;

    PORTA=0x00;
    DDRA=0xFF;        // Dat PORTA là PORT ra

    TCCR1B=0x03;      // Enable Timer 1, Prescale = 64
    TCNT1H=0x00;      // Gia tri khoi dau cho Timer
    TCNT1L=0x00;

    while (1){
        i = TCNT1;
        if(i >= 65500){
            PORTA ^= 0xFF;
            TCNT1 = 0;
        }
    }
}
```

Bài tập

- Bạn hãy tính toán xem với cấu hình timer như trên thì sau bao lâu PORT A đảo giá trị một lần.
- Viết chương trình trễ n(ms) sử dụng timer.

BÀI 14 : NGẮT

- *Giới thiệu về ngắt*
- *Cách cấu hình cho ngắt trong Atmega32.*
- *Ví dụ minh họa với ngắt ngoài*
- *Ví dụ minh họa với ngắt timer*

1. Giới thiệu về ngắt

Giống với timer, ngắt cũng là 1 trong những module rất quan trọng của vi điều khiển, sử dụng ngắt sẽ giúp chúng ta không phải mất thời gian kiểm tra liên tục 1 đoạn chương trình nào đó, ngoài ra, chúng ta có thể sử dụng ngắt để đồng thời cho vi điều khiển cùng 1 lúc làm nhiều nhiệm vụ.

Chúng ta cùng hình dung 1 ví dụ đơn giản về ngắt như sau :

Mỗi gia đình đều có 1 cái chuông cửa, cái chuông cửa đó đóng vai trò như 1 ngắt, mỗi khi có ai đó bấm chuông (xảy ra ngắt), chúng ta xuống mở cửa để cho người đó vào. Nếu như không có chuông cửa, chúng ta phải liên tục kiểm tra xem có ai ở cổng hay không để mở cửa, làm như thế sẽ mất thời gian hơn rất nhiều.

Một chương trình ngắt cũng giống như 1 chương trình con, khi điều kiện xảy ra ngắt thỏa mãn, vi điều khiển sẽ tạm dừng chương trình đang thực hiện để nhảy tới chương trình ngắt, sau khi thực hiện xong chương trình ngắt, vi điều khiển lại tiếp tục thực hiện công việc mà trước đó nó đang làm.

Điểm khác biệt giữa chương trình ngắt và chương trình con là chương trình ngắt không có đối số truyền vào và không được phép gọi (call) từ 1 chương trình chính hay 1 chương trình con khác.

Vi điều khiển Atmega32 có rất nhiều loại ngắt, cụ thể từng loại, chúng ta có thể tham khảo trong datasheet. Trong bài học này, chúng ta chỉ xem xét 2 loại ngắt là ngắt ngoài và ngắt timer.

Ngắt ngoài

Trong Atmega32 có 3 ngắt ngoài là INT0, INT1, INT2. Các ngắt này được cấu hình bởi 2 thanh ghi MCUCR và MCUCSR

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Thanh ghi MCUCR

ISCx0	ISCx1	Mô tả
0	0	Ngắt INTx xảy ra khi chân INTx ở mức thấp
0	1	Ngắt INTx xảy ra khi có thay đổi mức logic ở chân INTx
1	0	Ngắt theo sườn xuống
1	1	Ngắt theo sườn lên

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

Thanh ghi MCUCSR

ISC2	Mô tả
0	Ngắt theo sườn xuống
1	Ngắt theo sườn lên

Thanh ghi cho phép ngắt GICR :

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Các bit INT0, INT1, INT2 được dùng để enable các ngắt tương ứng, khi các bit này được set lên 1, các ngắt tương ứng sẽ được enable.

Ngắt timer

Ngắt timer xảy ra khi tràn timer hoặc khi giá trị trong timer đạt tới một giá trị đặt trước. Cấu hình timer các bạn có thể tham khảo ở bài timer, để cấu hình ngắt cho timer, chúng ta sử dụng thanh ghi TIMSK :

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TOIE0 : Bit này đặt chế độ ngắt cho timer 0, nếu được set bằng 1 thì ngắt timer 0 sẽ xảy ra khi tràn timer.

OCIE0 : Bit này đặt chế độ ngắt cho timer 0, nếu được set bằng 1 thì ngắt timer 0 sẽ xảy ra khi giá trị của timer 0 (TCNT0) đạt tới giá trị trong thanh ghi đặt trước (OCR0)

TOIE1 : Bit này đặt chế độ ngắt cho timer 1, nếu được set bằng 1 thì ngắt timer 1 sẽ xảy ra khi tràn timer.

OCIE1B : Khi bit này được set lên 1, ngắt timer 1 xảy ra khi OCR1B = TCNT1

OCIE1A : Khi bit này được set lên 1, ngắt timer 1 xảy ra khi OCR1A = TCNT1

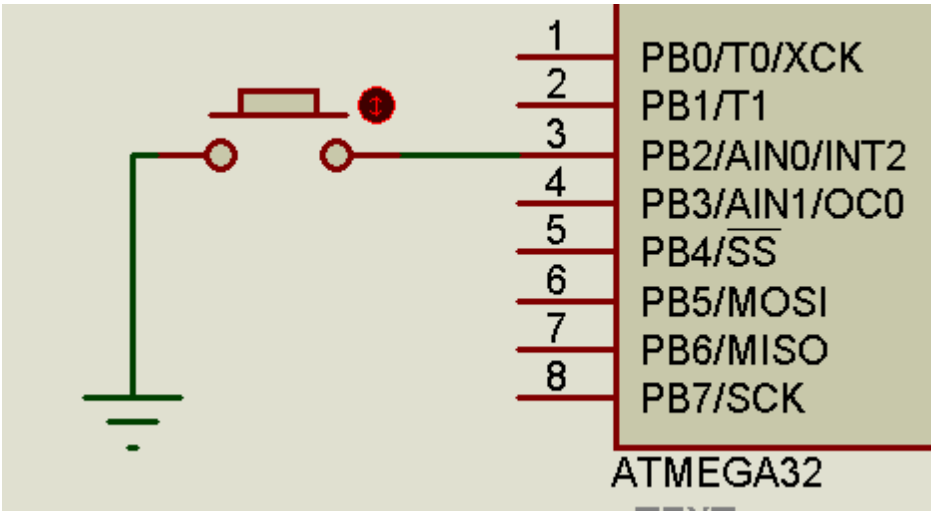
TOIE2 : Bit này đặt chế độ ngắt cho timer 2, nếu được set bằng 1 thì ngắt timer 2 sẽ xảy ra khi tràn timer.

2. Các bước cấu hình cho ngắt hoạt động

- Ngắt ngoài :
    - Đặt chế độ cho ngắt : ngắt theo sườn lên (xuống), hay ngắt theo mức.
    - Cho phép ngắt toàn cục.
    - Viết chương trình cho ngắt ngoài
- Ngắt timer :
    - Đặt chế độ cho timer (xem phần timer)
    - Cho phép ngắt timer.
    - Cho phép ngắt toàn cục.
    - Viết chương trình cho ngắt timer.
- Các loại ngắt khác cũng cấu hình tương tự như 2 loại ngắt trên.

3. Ví dụ

Ví dụ sau sẽ thao tác với ngắt ngoài INT2, mỗi khi có ngắt ngoài xảy ra, chúng ta đảo mức logic tại PORTA. Chúng ta có thể mắc led vào PORTA để quan sát mức logic tại port A.



Chương trình

```
#include <mega32.h>

interrupt [EXT_INT2] void ext_int2_isr(void){
     PORTA ^= 0xFF;

}

void main(){
// Cau hinh PORTB là cong vao, su dung pull-up resistor
     PORTB = 0xFF;
     DDRB = 0x00;

     DDRA = 0xFF;
     PORTA = 0;

     MCUCSR = 0x40;   // Ngat INT2 theo suon xuong
     GICR|=0x20;      // Cho phep ngat INT2

     #asm("sei");     // Cho phep ngat toan cuc

     while(1);
}
```

Bài tập

- Dựa vào ví dụ trên, bạn hãy cấu hình để sử dụng với các ngắt INT0, INT1 và các ngắt timer.

BÀI 15 : ĐIỀU KHIỂN ĐỘNG CƠ MỘT CHIỀU

- *Giới thiệu về động cơ 1 chiều*
- *Ví dụ minh họa*

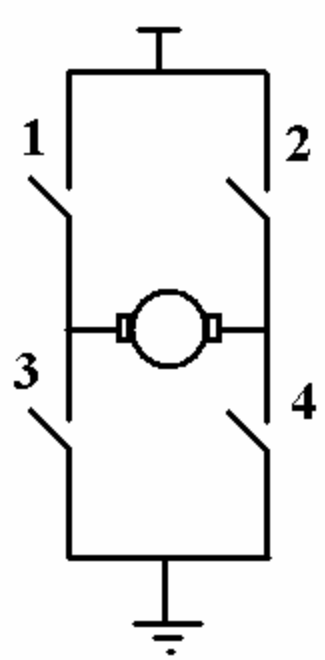
1. Giới thiệu về động cơ một chiều

Động cơ một chiều là loại động cơ có cấu tạo và cách điều khiển đơn giản nhất, tốc độ động cơ được điều khiển thông qua điện áp cấp vào 2 đầu động cơ. Động cơ một chiều được ứng dụng rất rộng rãi trong các hệ thống tự động.

Cho đến nay, có rất nhiều phương pháp dùng để điều khiển động cơ một chiều, bài giảng này sẽ trình bày cách sử dụng module PWM để điều chế điện áp đặt lên hai đầu động cơ, do đó điều khiển được tốc độ động cơ.

Mạch cầu H

Mạch cầu H là một trong những mạch phổ biến để điều khiển động cơ một chiều, sở dĩ gọi là mạch cầu H vì mạch có hình chữ H. Sơ đồ nguyên lí của mạch cầu H như sau :



Có 4 khóa chuyển 1,2,3,4. Tại một thời điểm luôn luôn có 2 khóa mở và 2 khóa đóng.

Giả sử khóa 1 và khóa 4 đóng, dòng điện chạy qua động cơ sẽ chạy từ trái qua phải, động cơ sẽ quay theo 1 chiều.

Nếu khóa 2 và 3 đóng, dòng điện đi qua động cơ sẽ có chiều từ phải qua trái, như vậy động cơ sẽ quay theo chiều ngược lại

Tránh để hai khóa 1 và 3 hoặc 2 khóa 2 và 4 cùng đóng, như vậy sẽ gây ra hiện tượng đoản mạch.

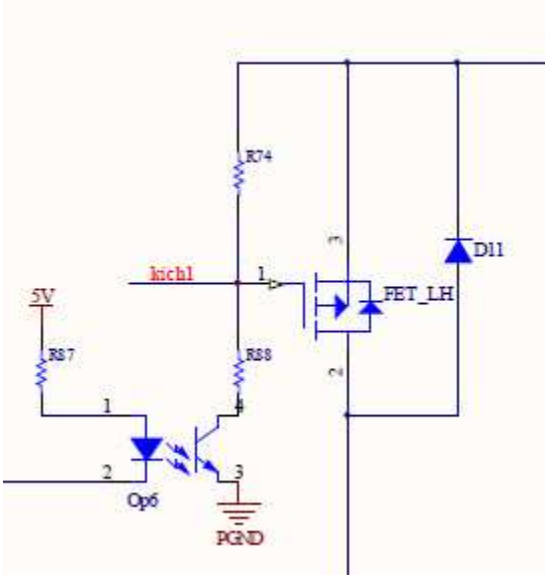
Như vậy với việc thay đổi việc đóng mở các van, chúng ta có thể thay đổi chiều quay của động cơ.

Bây giờ chúng ta cần điều khiển tốc độ động cơ, giả sử chúng ta cho 2 van 1 và 4 cùng đóng mở liên tục (giống như phần PWM), điện áp đặt lên động cơ sẽ có dạng xung, nếu tốc độ đóng mở thấp, động cơ sẽ quay giật cục vì lúc có điện áp, lúc không có điện áp. Nếu tốc độ đóng mở cao (khoảng trên 15KHz) thì do quán tính nên chúng ta sẽ thấy động cơ quay trơn đều.

Trong thực tế, các khóa chuyển trong hình trên có thể dùng các transistor, hay Mostfet, không nên dùng relay, vì relay có tốc độ đóng mở thấp.

2. Ví dụ minh họa

Sau đây là sơ đồ nguyên lí 1 van của module điều khiển động cơ được sử dụng trong ví dụ :



Mạch trên sử dụng Mosfet để điều khiển động cơ, diode D11 có tác dụng bảo vệ cho FET, Opto Op6 có tác dụng cách li mạch động cơ với mạch điều khiển, điều này sẽ đảm bảo an toàn cho phần mạch điều khiển.

Đoạn chương trình sau sẽ minh họa việc điều khiển động cơ 1 chiều quay thuận, quay ngược :

```
#include <mega32.h>

#define MotorPWM    PORTC.6
#define MotorDir    PORTC.4

void main(){
    PORTC = 0;
    DDRC = 0xFF;

    MotorPWM = 0;    // Cho phep dong co quay
    MotorDir = 0;    // Chieu dong co

    while(1);
}
```

Bài tập

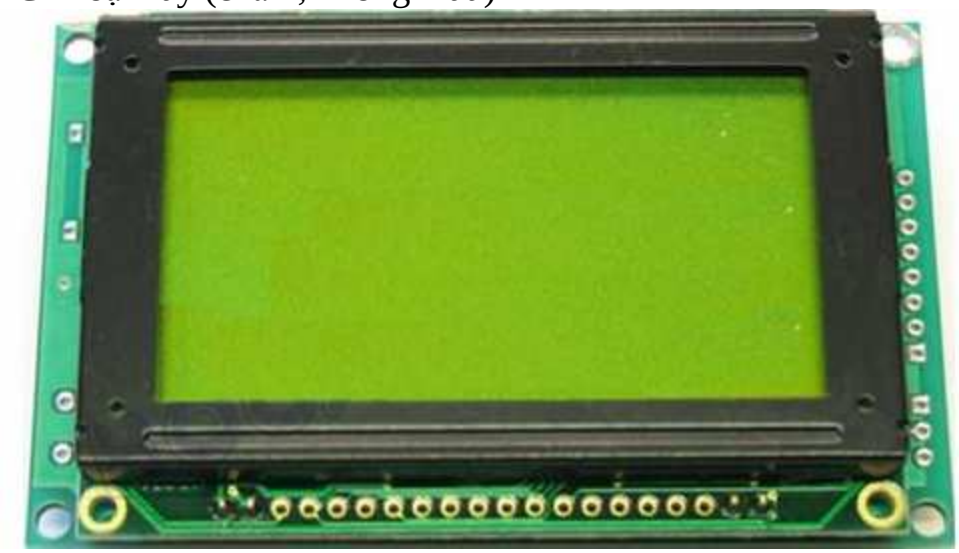
Để có thể điều khiển được tốc độ động cơ 1 chiều, chúng ta thay đổi điện áp cấp vào chân MotorPWM bằng cách điều xung thông qua module PWM của vi điều khiển. Bài tập dành cho bạn là hãy vận dụng bài học về timer, thiết lập chế độ PWM để điều khiển tốc độ động cơ.

BÀI 16 : GIAO TIẾP VỚI GLCD

- *Cơ bản về GLCD*
- *Ví dụ minh họa*

1. Cơ bản về GLCD

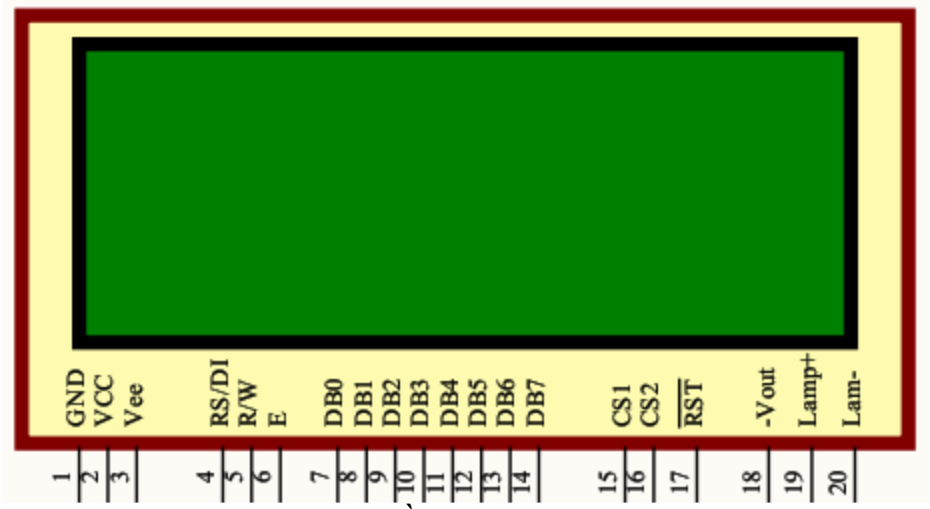
Graphic LCD (gọi tắt là GLCD) loại chấm không màu là các loại màn hình tinh thể lỏng nhỏ dùng để hiển thị chữ, số hoặc hình ảnh. Khác với Text LCD, GLCD không được chia thành các ô để hiển thị các mã ASCII vì GLCD không có bộ nhớ CGRAM (Character Generation RAM). GLCD 128x64 có 128 cột và 64 hàng tương ứng có 128x64=8192 chấm (dot). Mỗi chấm tương ứng với 1 bit dữ liệu, và như thế cần 8192 bits hay 1024 bytes RAM để chứa dữ liệu hiển thị đầy mỗi 128x64 GLCD. Tùy theo loại chip điều khiển, nguyên lý hoạt động của GLCD có thể khác nhau, bài này sẽ giới thiệu loại GLCD được điều khiển bởi chip KS0108 của Samsung, có thể nói GLCD với KS0108 là phổ biến nhất trong các loại GLCD loại này (chấm, không màu)



Hình ảnh GLCD

Chip KS0108 chỉ có 512 bytes RAM (4096 bits = 64x64) và vì thế chỉ điều khiển hiển thị được 64 dòng x 64 cột. Để điều khiển GLCD 168x64 cần 2 chip KS0108, và thực thế trong các loại GLCD có 2 chip KS0108, GLCD 128x64 do đó tương tự 2 GLCD 64x64 ghép lại

Các GLCD 128x64 dùng KS0108 thường có 20 chân trong đó chỉ có 18 chân là thực sự điều khiển trực tiếp GLCD, 2 chân (thường là 2 chân cuối 19 và 20) là 2 chân Anode và Cathode của LED nền. Trong 18 chân còn lại, có 4 chân cung cấp nguồn và 14 chân điều khiển+dữ liệu. Khác với các Text LCD HD44780U, GLCD KS0108 không hỗ trợ chế độ giao tiếp 4 bit, do đó bạn cần dành ra 14 chân để điều khiển 1 GLCD 128x64.



Sơ đồ chân GLCD

Chân VSS được nối trực tiếp với GND, chân VDD nối với nguồn +5V, một biến trở khoảng 20K được dùng để chia điện áp giữa Vdd và Vee cho chân Vo, bằng cách thay đổi giá trị biến trở chúng ta có thể điều chỉnh độ tương phản của GLCD. Các chân điều khiển RS, R/W, EN và các đường dữ liệu được nối trực tiếp với vi điều khiển. Riêng chân Reset (RST) có thể nối trực tiếp với nguồn 5V.

EN (Enable): cho phép một quá trình bắt đầu, bình thường chân EN được giữ ở mức thấp, khi một thực hiện một quá trình nào đó (đọc hoặc ghi GLCD), các chân điều khiển khác sẽ được cài đặt sẵn sàng, sau đó kích chân EN lên mức cao. Khi EN được kéo lên cao, GLCD bắt đầu làm thực hiện quá trình được yêu cầu, chúng ta cần chờ một khoảng thời gian ngắn cho GLCD đọc hoặc gởi dữ liệu. Cuối cùng là kéo EN xuống mức thấp để kết thúc quá trình và cũng để chuẩn bị chân EN cho quá trình sau này.

RS (Register Select): là chân lựa chọn giữa dữ liệu (Data) và lệnh (Instruction), vì thế mà trong một số tài liệu bạn có thể thấy chân RS được gọi là chân DI (Data/Instruction Select). Chân RS=1 báo rằng tín hiệu trên các đường DATA

(D0:7) là dữ liệu ghi hoặc đọc từ RAM của GLCD. Khi RS=0, tín hiệu trên đương DATA là một mã lệnh (Instruction).

RW (Read/Write Select): chọn lựa giữa việc đọc và ghi. Khi RW=1, chiều truy cập từ GLCD ra ngoài (GLCD->AVR). RW=0 cho phép ghi vào GLCD. Giao tiếp với GLCD chủ yếu là quá trình ghi (AVR ->GLCD), chỉ duy nhất trường hợp đọc dữ liệu từ GLCD là đọc bit BUSY và đọc dữ liệu từ RAM. Đọc bit BUSY thì chúng ta đã khảo sát cho Text LCD, bit này báo GLCD có đang bận hay không, việc đọc này sẽ được dùng để viết hàm wait_GLCD. Đọc dữ liệu từ RAM của GLCD là một khả năng mới mà Text LCD không có, bằng việc đọc ngược từ GLCD vào AVR, chúng ta có thể thực hiện nhiều phép logic hình (hay mặt nạ, mask) làm cho việc hiển thị GLCD thêm thú vị.

CS2 và CS1 (Chip Select): như tôi đã trình bày trong phần trên, mỗi chip KS0108 chỉ có khả năng điều khiển một GLCD có kích thước 64x64, trên các GLCD 128x64 có 2 chip KS0108 làm việc cùng nhau, mỗi chip đảm nhiệm một nữa LCD, 2 chân CS2 và CS1 cho phép chọn một chip KS0108 để làm việc. Thông thường nếu CS2=0, CS1=1 thì nửa trái được kích hoạt, ngược lại khi CS2=1, CS1=0 thì nửa phải được chọn. Chúng ta sẽ hiểu rõ hơn cách phối hợp làm việc của 2 nửa GLCD trong phần khảo sát bộ nhớ của LCD.

Tổ chức bộ nhớ.

Chip KS0108 có một loại bộ nhớ duy nhất đó là RAM, không có bộ nhớ chứa bộ font hay chứa mã font tự tạo như chip HD44780U của Text LCD. Vì vậy, dữ liệu ghi vào RAM sẽ được hiển thị trực tiếp trên GLCD.

Mỗi chip KS0108 có 512 bytes RAM tương ứng với 4096 chấm trên một nửa (64x64) LCD. RAM của KS0108 không cho phép truy cập từng bit mà theo từng byte, điều này có nghĩa là mỗi lần chúng ta viết một giá trị vào một byte nào đó trên RAM của GLCD, sẽ có 8 chấm bị tác động, 8 chấm này nằm trên cùng 1 cột. Vì lý do này, 64 dòng GLCD thường được chia thành 8 pages, mỗi page có độ cao 8 bit và rộng 128 cột (cả 2 chip gộp lại). Hình 3 mô tả "bề mặt" một GLCD và cũng là cách sắp xếp RAM của các chip KS0108.

Tổ chức RAM của 2 chip KS0108 trái và phải hoàn toàn tương tự, việc đọc hay ghi vào RAM của 2 chip cũng được thực hiện như nhau. Chúng ta sẽ chọn nửa trái GLCD để khảo sát. Như bạn thấy trên hình 3, 64 dòng từ trên xuống dưới được chia thành 8 "dãy" mà ta gọi là 8 pages. Page trên cùng là page 0 và page dưới cùng la page 7. Trong các GLCD, page còn được gọi là địa chỉ X (X

address), hay nói cách khác X=0 là địa chỉ của page trên cùng, tương tự như thế, X=7 là địa chỉ của page dưới cùng. Mỗi page chứa 64 cột (chỉ xét 1 chip KS0108), mỗi cột là một byte RAM 8 bit, mỗi bit tương ứng với 1 chấm trên LCD, bit có trọng số thấp (LBS - tức bit D0 như trong hình 3) tương ứng với chấm trên cao nhất. Bit có trọng số cao nhất (MBS - tức bit D7 như trong hình 3) tương ứng với chấm thấp nhất trong 1 page. Thứ tự các cột trong 1 page gọi là địa chỉ Y (Y address), như thế cột đầu tiên có địa chỉ Y = 0 trong khi cột cuối cùng có địa chỉ Y là 63.

Bằng cách phối hợp địa chỉ X và địa chỉ Y chúng ta xác định được vị trí của byte cần đọc hoặc ghi. Chip KS0108, tất nhiên, sẽ hỗ trợ các lệnh di chuyển đến địa chỉ X và Y để ghi hay đọc RAM

Tập lệnh cho chip KS0108.

So với HD44780U của Text LCD, lệnh cho KS0108 của GLCD đơn giản và ít hơn và vì thế viết chương trình điều khiển GLCD cũng tương đối dễ hơn Text LCD. Có tất cả 7 lệnh (Instruction) có thể giao tiếp với KS0108. Tôi sẽ lần lượt giải thích ý nghĩa và cách sử dụng của từng lệnh.

Display ON/OFF – Hiển thị GLCD: lệnh này cho phép GLCD hiển thị nội dung trên RAM ra "bề mặt" GLCD. Để viết lệnh này cho GLCD, 2 chân RS và RW cần được kéo xuống mức thấp (RS=0: đây là Instrucion, RW=0: AVR->GLCD). Mã lệnh (code) được chứa trong 7 bit cao (D7:1) và bit D0 chứa thông số. Quan sát bảng 2, dễ thấy mã lệnh nhị phân cho Display ON/OFF là 0011111x (0x3E+x) trong đó x=1: cho phép GLCD hiển thị, x=0: tắt hiển thị.

Set Address – chọn địa chỉ: đúng hơn đây là lệnh chọn cột hay chọn địa chỉ Y. Hai bit D7 và D6 chứa mã lệnh (01000000=0x40=64) và 6 bit còn lại chứa chỉ số của cột muốn di chuyển đến. Chú ý là mỗi nửa GLCD có 64 cột nên cần 6 bit để chứa chỉ số này (26=64). Vậy lệnh này có dạng 0x40+Y. Ví dụ nếu chúng ta muốn di chuyển đến cột 36 chúng ta ghi vào GLCD mã lệnh: 0x40+36. Hai chân RS và RW được giữ ở mức thấp khi thực hiện lệnh này.

Set Page – chọn trang: lệnh cho phép chọn page (hay địa chỉ X) cần di chuyển đến, do GLCD chỉ có 8 pages nên chỉ cần 3 bit để chứa địa chỉ page. Mã lệnh cho lệnh này có dạng 0xB8+X. Trong đó biến X là chỉ số page cần di chuyển đến. Hai chân RS và RW được giữ ở mức thấp khi thực hiện lệnh này.

Display Start Line – chọn line đầu tiên: hay còn gọi là lệnh "cuộn", lệnh này cho phép di chuyển toàn bộ hình ảnh trên GLCD (hay RAM) lên phía trên một số dòng nào đó, chúng ta gọi là LOffset. Số lượng LOffset có thể từ 0 đến 63 nên cần 6 bit chứa giá trị này. Mã lệnh Display Start Line có dạng 0xC0+LOffset. Hai chân RS và RW được giữ ở mức thấp khi thực hiện lệnh này. Khi di chuyển GLCD lên phía trên, phần dữ liệu phía trên bị che khuất sẽ "cuộn" xuống phía dưới. Hình 5 là một ví dụ "cuộn" GLCD lên 20 dòng.

Status Read – đọc trạng thái GLCD: đây là một trong 2 lệnh đọc từ GLCD. Cũng giống như với Text LCD, lệnh đọc trạng thái GLCD chủ yếu để xét bit BUSY (bit thứ 7) xem GLCD có đang bận hay không, lệnh này sẽ được dùng để viết một hàm wait_GLCD chờ cho đến khi GLCD rảnh. Vì đây là lệnh đọc từ GLCD nên chân RW phải được set lên mức 1 trước khi thực hiện, chân RS vẫn ở mức thấp (đọc Instruction).

Write Display Data – ghi dữ liệu cần hiển thị vào GLCD hay RAM: vì đây là 1 lệnh ghi dữ liệu hiển thị nên chân RS cần được set lên 1 trước khi thực hiện, chân RW giữ ở mức 0. Lệnh này cho phép ghi một byte dữ liệu vào RAM của KS0108 và cũng là dữ liệu sẽ hiển thị lên GLCD tại vị trí hiện hành của 2 con trỏ địa chỉ X và Y. 8 bit dữ liệu này sẽ tương ứng với 8 chấm trên cột Y ở page X. Chú ý là sau lệnh Write Display Data, địa chỉ cột Y tự động được tăng lên 1 và vì thế nếu có một dữ liệu mới được ghi, dữ liệu mới sẽ không "đè" lên dữ liệu cũ. Việc tăng tự động địa chỉ Y rất có lợi cho việc ghi dữ liệu liên tiếp, nó giúp giảm thời gian set lại địa chỉ cột Y. Sau khi thực hiện ghi ở cột Y=63 (cột cuối cùng trong 1 page, đối với 1 chip KS0108), Ysẽ về 0.

Read Display Data – đọc dữ liệu hiển thị từ GLCD (cũng là dữ liệu từ RAM của KS0108): lệnh đọc này mới so với Text LCD, nó cho phép chúng ta đọc ngược 1 byte dữ liệu từ RAM của KS0108 tại vị trí hiện hành về AVR. Sau khi đã đọc được giá trị tại vị trí hiện hành, chúng ta có thể thực hiện các phép Logic như đảo bit, or hay and…làm tăng khả năng thao tác hình ảnh. Trước khi thực hiện đọc chúng ta cần di chuyển đến vị trí muốn đọc bằng 2 lệnh set địa chỉ X và Y, sau khi đọc giá trị địa chỉ page X và cột Y không thay đổi, do đó nếu đọc tiếp mà không di chuyển địa chỉ thì vẫn thu được giá trị cũ.

2. Ví dụ minh họa

Phần này sẽ trình bày ví dụ minh họa để giao tiếp với GLCD, các hàm giao tiếp với GLCD được viết trong file myglcd.c, sơ đồ chân nối cũng thể hiện rõ trong file myglcd.h, sau đây là chương trình :

```
#include "mega32.h"
#include "stdio.h"
#include "myGLCD.h"
#include "delay.h"

void main(void){
    uint8_t start=33, Line=0, Col=0,i;
    GLCD_Init();
    GLCD_Clr();

    //In cac ky tu trong bang font7x8----
    GLCD_Clr();
    for (i=start;i<start+97;i++){
        GLCD_PutChar78(Line, Col, i);
        Col+=8;
        if (Col>127){Col=0; Line++;}
        delay_ms(20);
    }
    delay_ms(1000);

    while(1){

    };
}
```

Bài tập

Dựa vào driver đã cung cấp ở trên, bạn hãy lập trình để hiển thị các hình 2D như hình chữ nhật, hình vuông, hình tròn…