ĐẠI HỌC BÁCH KHOA HÀ NỘI

TRUNG TÂM ĐÀO TẠO TÀI NĂNG VÀ CHẤT LƯỢNG CAO

GIÁO TRÌNH CƠ SỞ VÀ PHÁT TRIỂN AVR

Giảng viên hướng dẫn : **PGS.PHAN BÙI KHÔI**

Sinh viên : **NGUYỄN VĂN TOẢN** *SHSV:* **20092792**
LÊ MINH NGHĨA **20091878**
DƯƠNG VĂN HÀ **20090882**

Lớp : **KSTN-CƠ ĐIỆN TỬ K54**

HÀ NỘI, 6/2013

GIÁO TRÌNH VI ĐIỀU KHIỂN AVR

GIÁO TRÌNH VI ĐIỀU KHIỂN AVR

Lời Nói Đầu

Với nhiệm vụ được phân công : " Phát triển Kit vi điều khiển AVR " trong thời gian thực tập kỹ thuật ; bên cạnh những kiến thức sẵn có về vi điều khiển AVR, cộng thêm một số tìm hiểu bổ sung nhóm em gồm bạn Lê Minh Nghĩa , Dương Văn Hà và Nguyễn Văn Toản đã hoàn thành tốt nhiệm vụ được giao . Cụ thể, nhóm đã thực hiện test phần cứng của một số bộ Kit bằng code riêng do nhóm tự viết, thêm vào đó nhóm đã thực hiện một số bài toán mở rộng : " điều khiển đèn giao thông ; điều khiển và hiển thị tốc độ động cơ một chiều, nhiệt độ và đồng hồ thời gian thực bằng máy tính PC và hiển thị ra LCD;điều khiển động cơ bằng phím bấm, nhấn phím và hiển thị vị trí của phím được nhấn…" . Nhóm đã củng cố , bổ sung lý thuyết và đưa thêm một số ý kiến cải tiến để hoàn thiện bộ giáo trình AVR . Toàn bộ công việc nhóm thực hiện đã được trình bày cụ thể trong nội dung giáo trình .

Qua thời gian thực tập , nhóm em đã thu nhận được rất nhiều kinh nghiệm cũng như tác phong làm việc thực tế ; đây thực sự là những kiến thức rất bổ ích để nhóm em tiếp cận với môi trường làm việc chuyên nghiệp. Thay mặt nhóm , em xin được gửi lời cảm ơn chân thành nhất tới ***PGS.PHAN BÙI KHÔI*** *và thầy* ***PHẠM HỒNG THÁI*** *đã tạo điều kiện cho nhóm em có cơ hội học hỏi và nghiên cứu.*

Nhóm em xin chân thành cảm ơn Thầy !

Mục Lục

BÀI 1 : GIỚI THIỆU VỀ VI ĐIỀU KHIỂN AVR 5
1. Giới thiệu về vi điều khiển........ 5
2. Giới thiệu về vi điều khiển AVR........6
3. Lập trình cho AVR 8
BÀI 2 : GIAO TIẾP VÀO RA I/O 1414
1. Giới thiệu giao tiếp vào ra I/O 144
2. Cách cấu hình chức năng IO 144
3. Ví dụ minh họa 155
BÀI 3 : GIAO TIẾP VỚI LED 7 THANH 1919
1. Cơ bản về led 7 thanh 19
2. Nguyên lí lập trình cho led 7 thanh 200
3. Ví dụ minh họa 211
BÀI 4 : GIAO TIẾP VỚI BÀN PHÍM 27
1. Cơ bản về phím bấm........ 27
2. Chương trình ví dụ........ 27
3. Kĩ thuật chống rung bàn phím........ 28
BÀI 5 : BỘ CHUYỂN ĐỔI ADC 31
1. Giới thiệu về ADC........ 311
2. Cách cấu hình ADC trong Code Vision cho Atmega32. 322
3. Ví dụ minh họa 322
BÀI 6 : GIAO TIẾP LCD 36
1. Giới thiệu về LCD 16x2 36
2. Cách cấu hình cho LCD trong Code Vision cho Atmega32 50
3. Bài tập 52
BÀI 7 : GIAO TIẾP VỚI LED MA TRẬN........**Error! Bookmark not defined.**3
1. Cơ bản về led ma trận........ 533
2. Tạo font cho led ma trận........ 533
3. Ví dụ minh họa. 555
BÀI 8: GIAO TIẾP MÁY TÍNH........ 556

1. Cơ bản về giao tiếp RS232 566
2. Cách cấu hình module UART trong Code Vision 57
3. Ví dụ 58
BÀI 9 : GIAO TIẾP I^2C 677
1. Giới thiệu chung về I2C 677
2. Module I^2C trong Atmega32 744
3. Ví dụ 80
BÀI 10 : ĐỘNG CƠ BƯỚC 844
1. Cơ bản về động cơ bước 844
2. Các mạch điều khiển động cơ bước 844
3. Ví dụ 888
BÀI 11 : GIAO TIẾP VỚI CỔNG LPT 90
1. Cơ bản về cổng LPT 90
2. Ví dụ minh họa 933
BÀI 12 : GIAO TIẾP VỚI MA TRẬN PHÍM 945
1. Cơ bản về ma trận phím 955
2. Ví dụ minh họa 966
BÀI 13 : TIMER 989
1. Giới thiệu về timer 999
2. Ví dụ minh họa 1088
BÀI 14 : NGẮT 11111
1. Giới thiệu về ngắt 11111
2. Các bước cấu hình cho ngắt hoạt động 11313
3. Ví dụ 11414
BÀI 15 : ĐIỀU KHIỂN ĐỘNG CƠ MỘT CHIỀU 117
1. Giới thiệu về điều khiển động cơ một chiều 117
2. Ví dụ minh họa 118
BÀI 16 : GIAO TIẾP VỚI GLCD .... **Error! Bookmark not defined.**20
1. Cơ bản về GLCD .... **Error! Bookmark not defined.**20
2. Ví dụ minh họa .... **Error! Bookmark not defined.**24
BÀI TOÁN MỞ RỘNG 130

BÀI 1 : GIỚI THIỆU VỀ VI ĐIỀU KHIỂN AVR

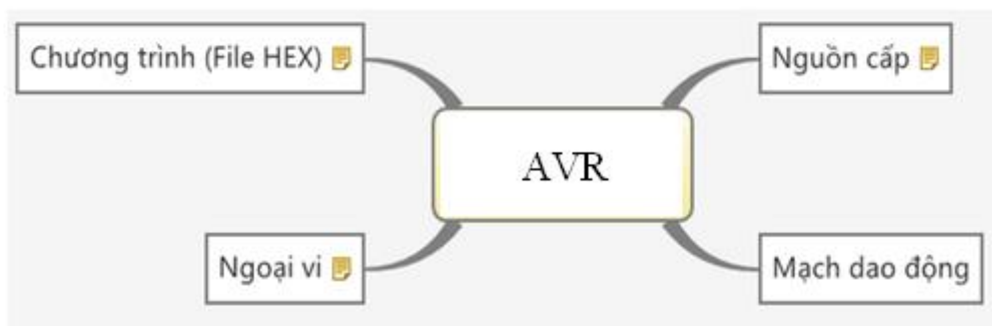
- *Giới thiệu chung về vi điều khiển.*
- *Giới thiệu về vi điều khiển Atmega32.*
- *Lập trình cho Atmega32.*

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.

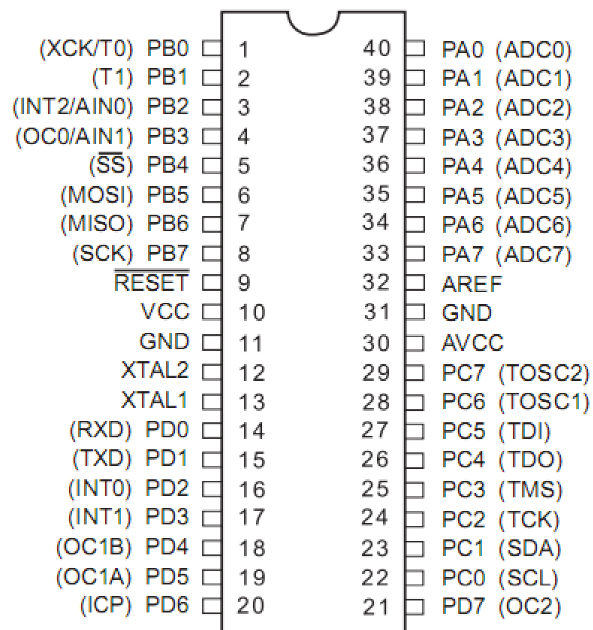
Các tính năng mới của họ AVR:

- ✓ 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 :

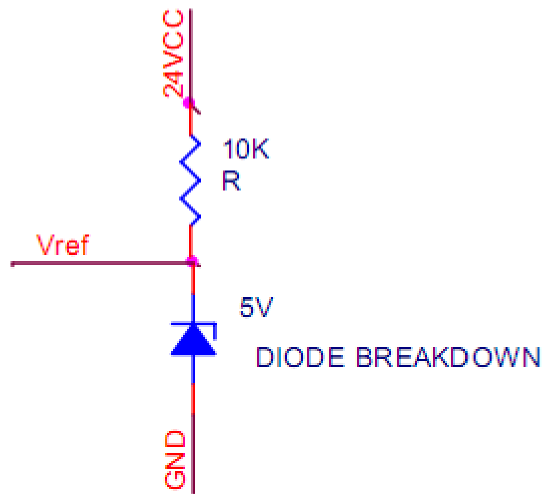
- ✓ Bộ nhớ 32KB Flash có khả năng đọc, ghi 10000 lần
- ✓ 1024 byte EEPROM có khả năng đọc, ghi 100000 lần.
- ✓ 2KB SRAM.
- ✓ 8 kênh đầu vào ADC 10 bit.
- ✓ Đóng vỏ 40 chân , trong đó có 32 chân vào ra dữ liệu chia làm 4 PORT A,B,C,D. Các chân này đều có chế độ pull_up resistors.
- ✓ Hỗ trợ các giao tiếp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

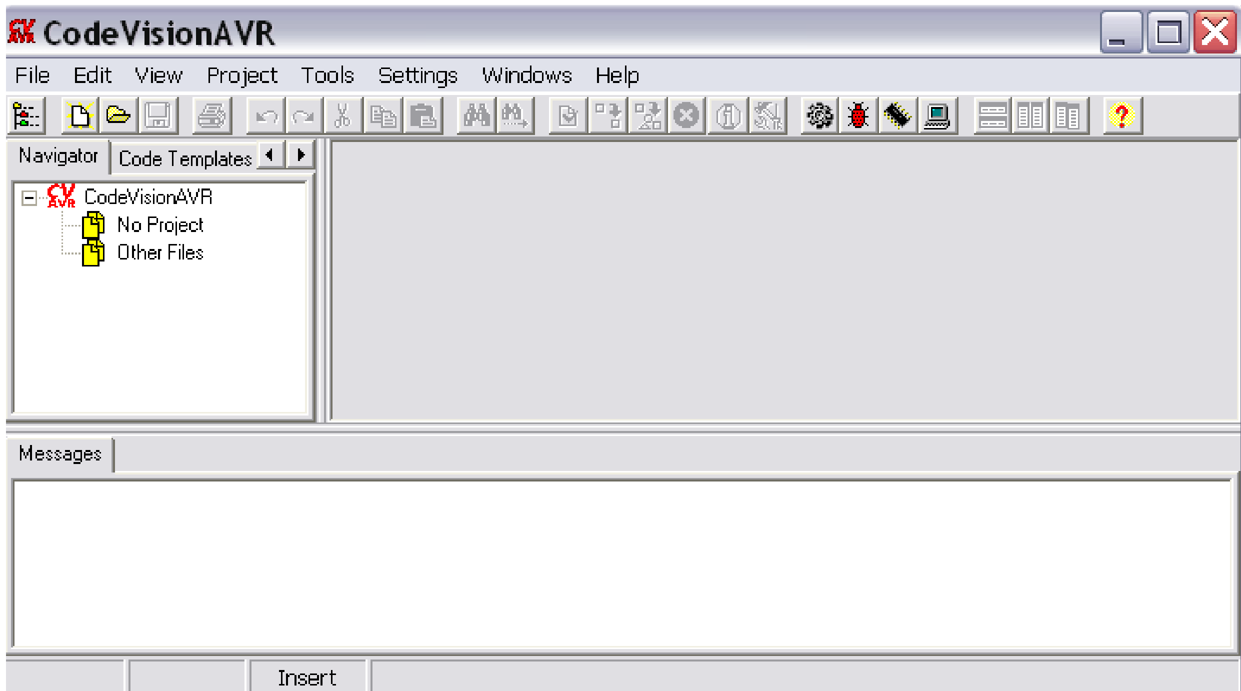
Ngôn ngữ ASM có ưu điểm là gọn nhẹ, giúp người lập trình nắm bắt sâu hơn về phần cứng. Tuy nhiên lại có nhược điểm là phức tạp, khó triển khai về mặt thuật toán, không thuận tiện để xây dựng các chương trình lớn.

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.

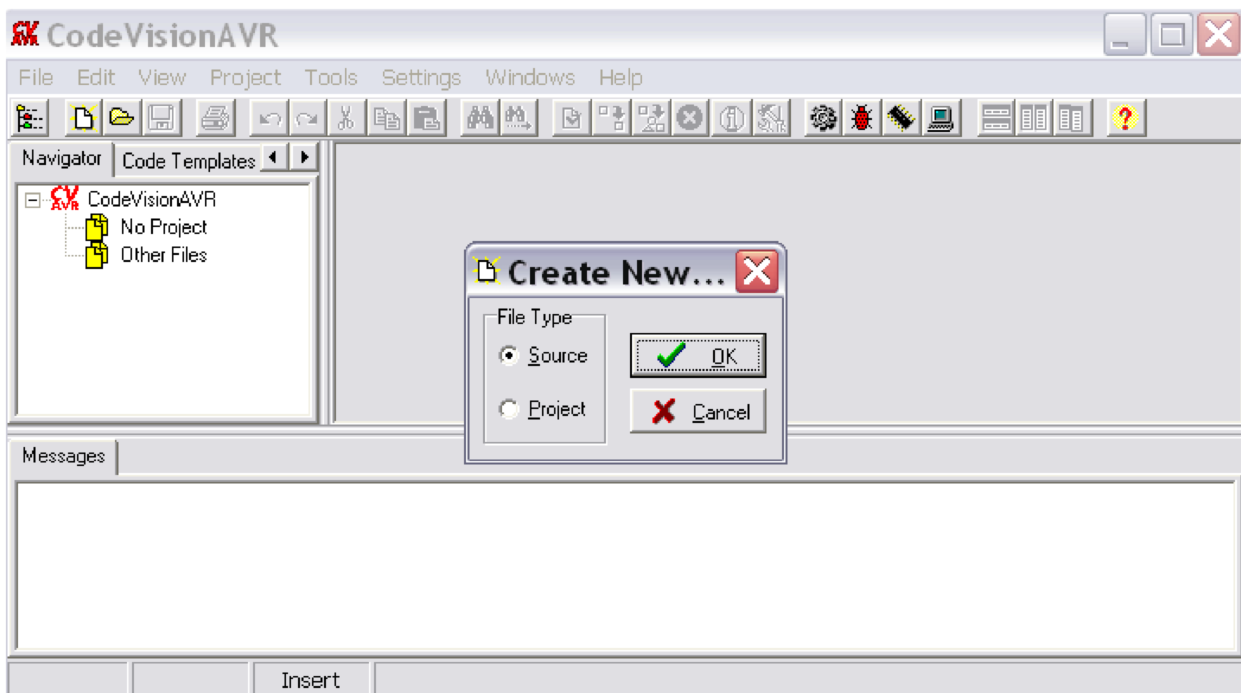
Tùy vào từng bài toán, từng yêu cầu cụ thể mà ta chọn lựa ngôn ngữ lập trình cho phù hợp.

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.

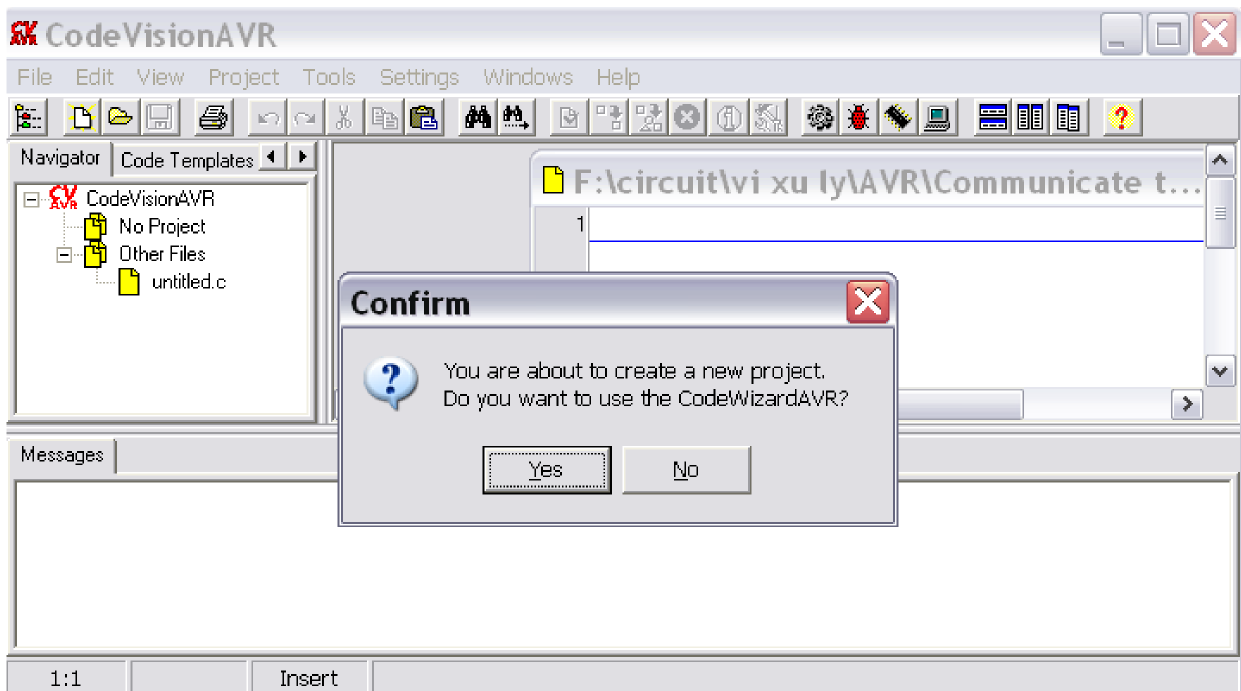


Giao diện phần mềm Code Vision

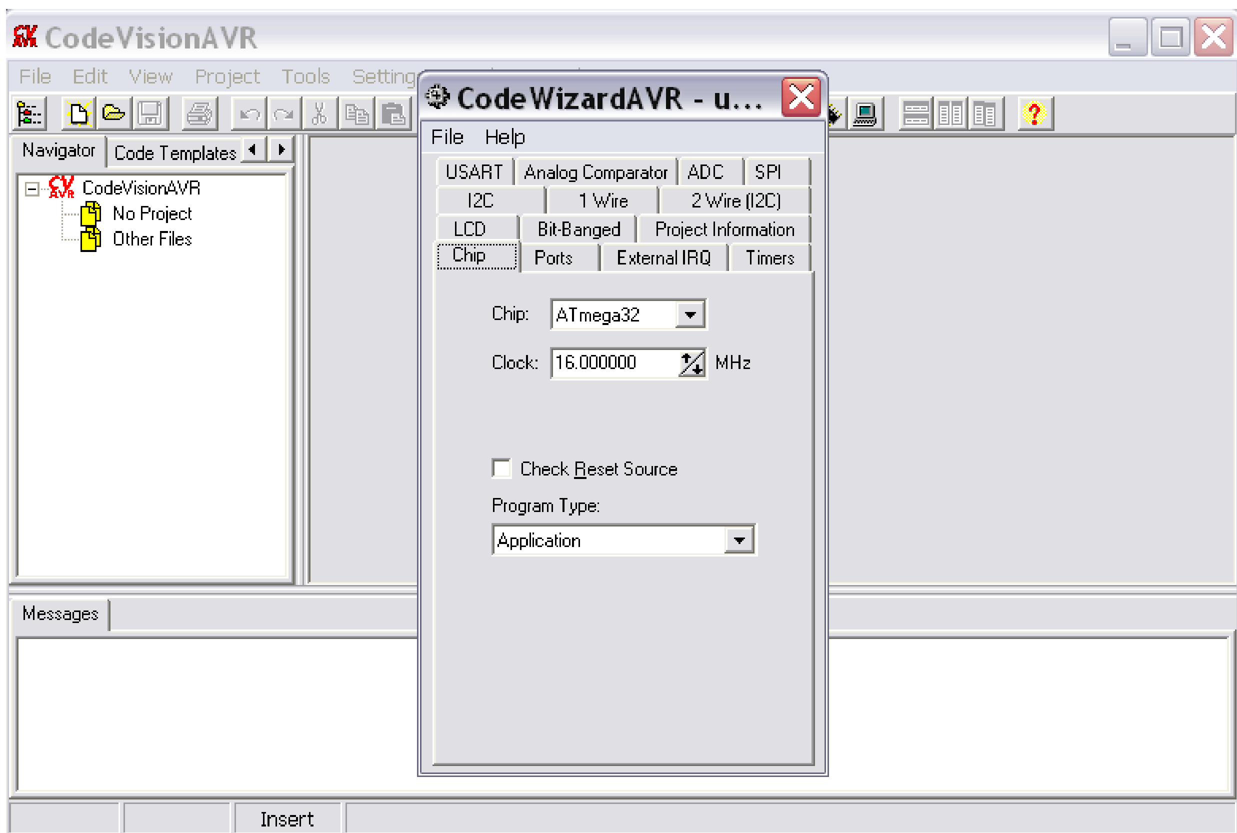
Tạo project trong Code Vision :
Để tạo Project mới chọn trên menu: File -> New được như sau:



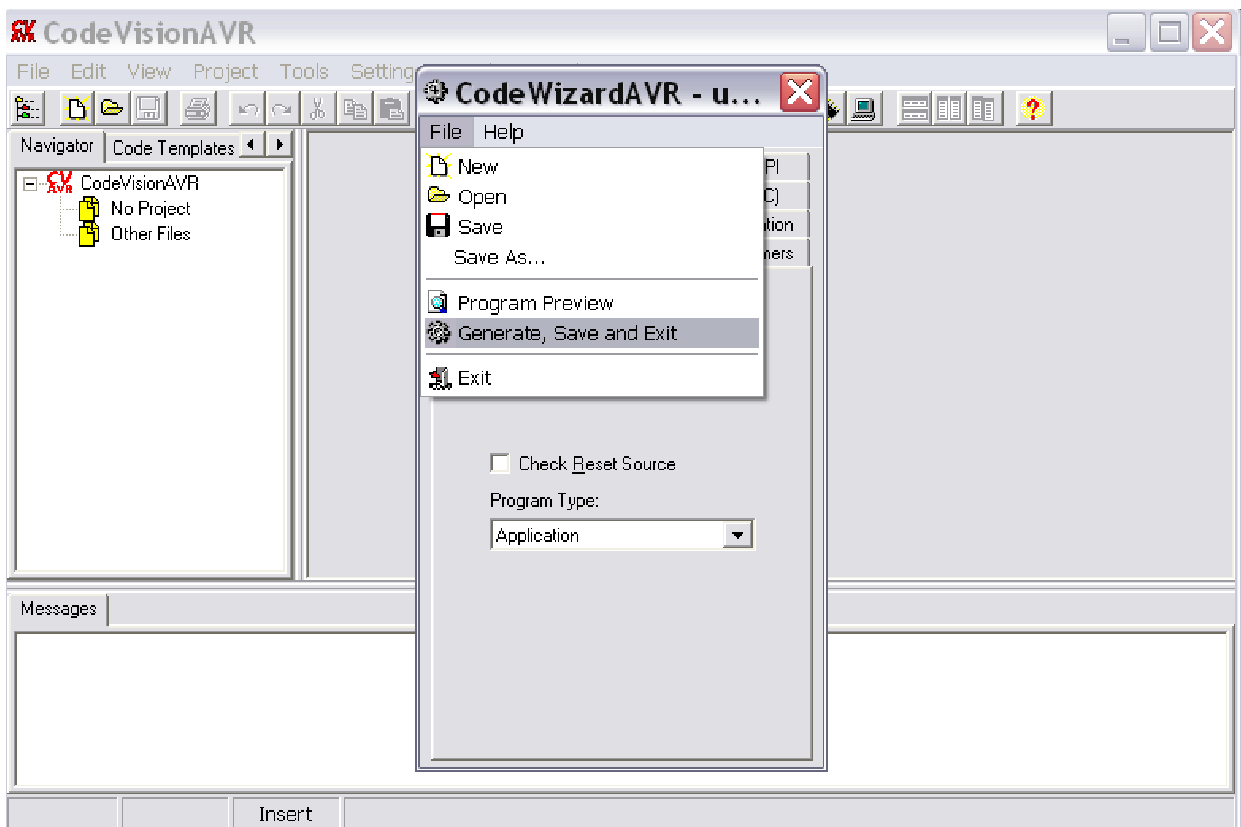
Chọn Project sau đó click chuột vào OK được cửa sổ hỏi xem có sử dụng Code Winzard không:



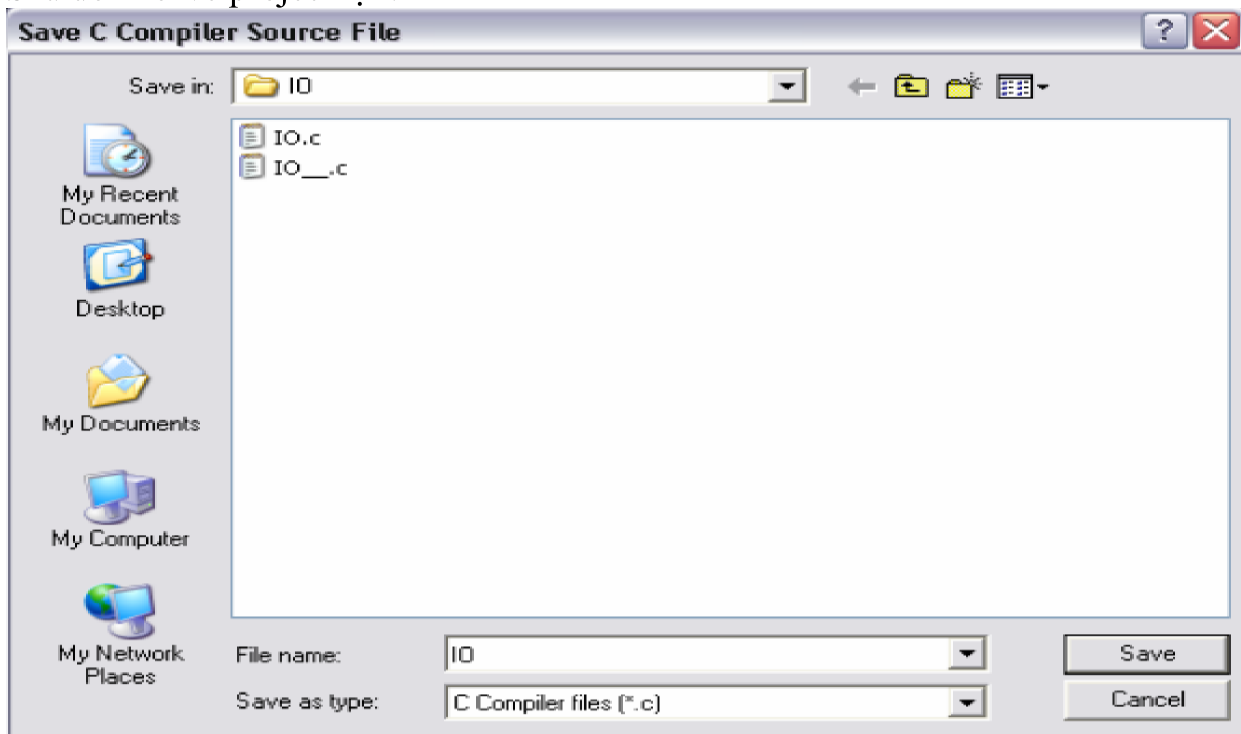
Chọn Yes được cửa sổ CodeWinzardAVR như sau :



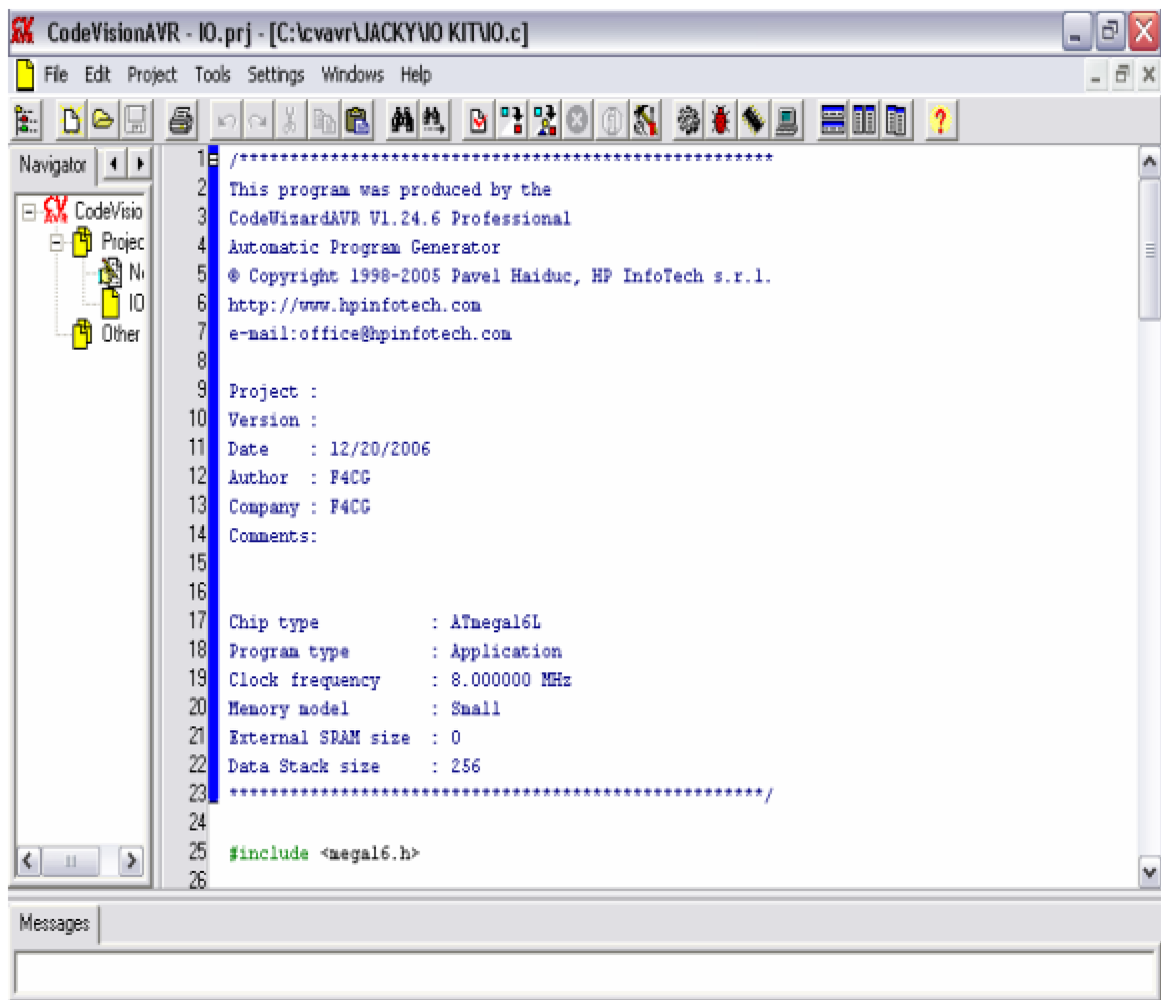
- Sử dụng chíp AVR nào và thạch anh tần số bao nhiêu ta nhập vào tab Chip. Để khởi tạo cho các cổng IO ta chuyển qua tab Ports.
- Các chân IO của AVR mặc định trạng thái IN, muốn chuyển thành trạng thái OUT để có thể đưa các mức logic ra ta click chuột vào các nút IN (mầu trắng) để nó chuyển thành OUT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

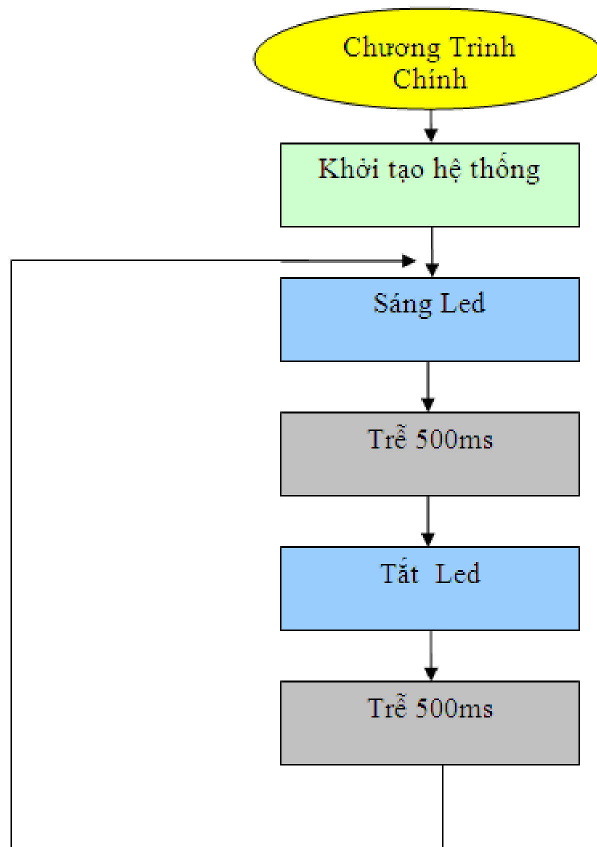
3. Ví dụ minh họa

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

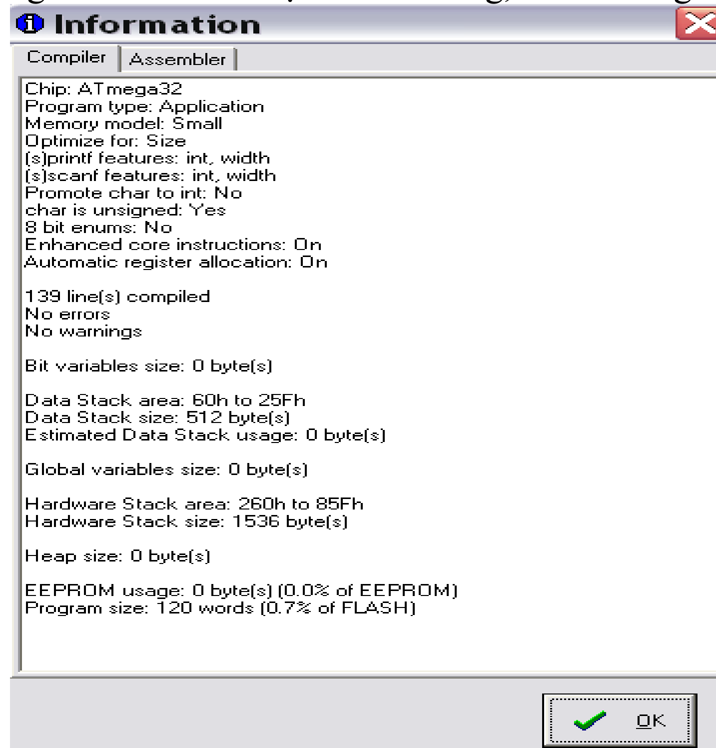
```
// Led blinking program
// Author : pk
// Date : 30/08/2009

#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 :

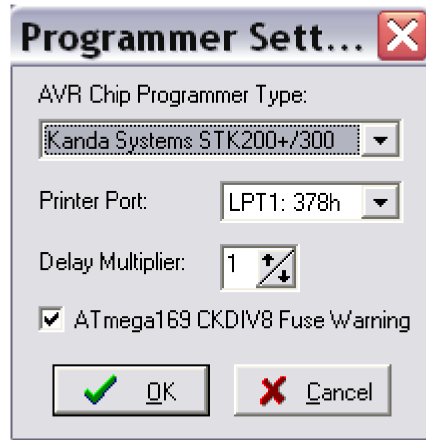


Sau khi viết xong chương trình, chúng ta nhấn Shift+F9 để biên dịch. Nếu chương trình không có lỗi và biên dịch thành công, sẽ có thông báo như sau :

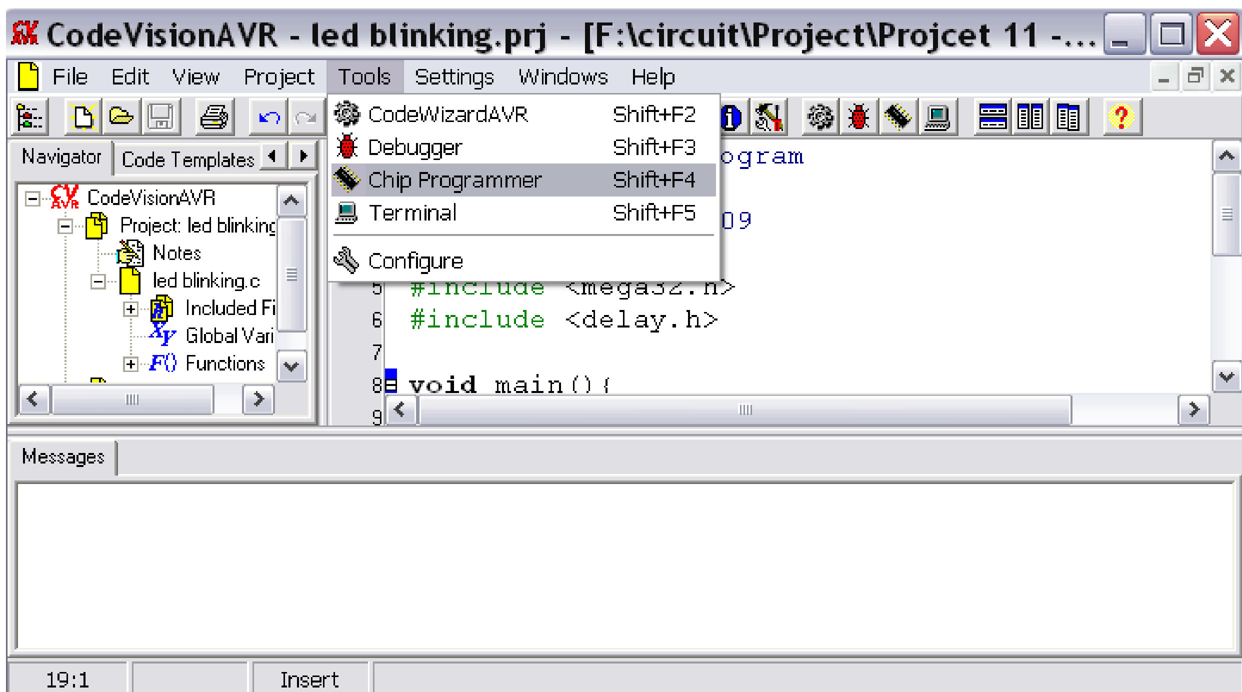


Để nạp chương trình các bạn cần cấu hình cho mạch nạp. Vào menu: Settings -> Programmer được cửa sổ như sau :

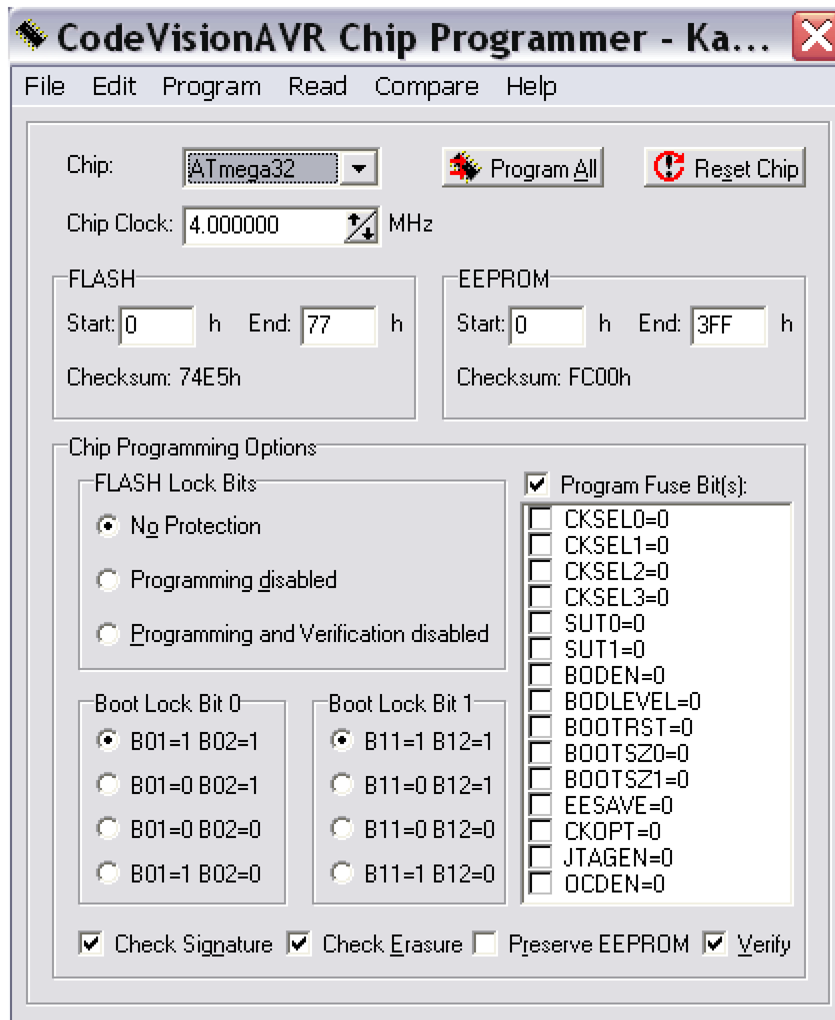
Mạch nạp ta dùng STK 200 do đó các bạn chọn Kanda *Systems STK200+/300*. Nhấp OK. Sau đó các bạn chọn trên menu : Projects -> Configure được cửa sổ như sau:



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

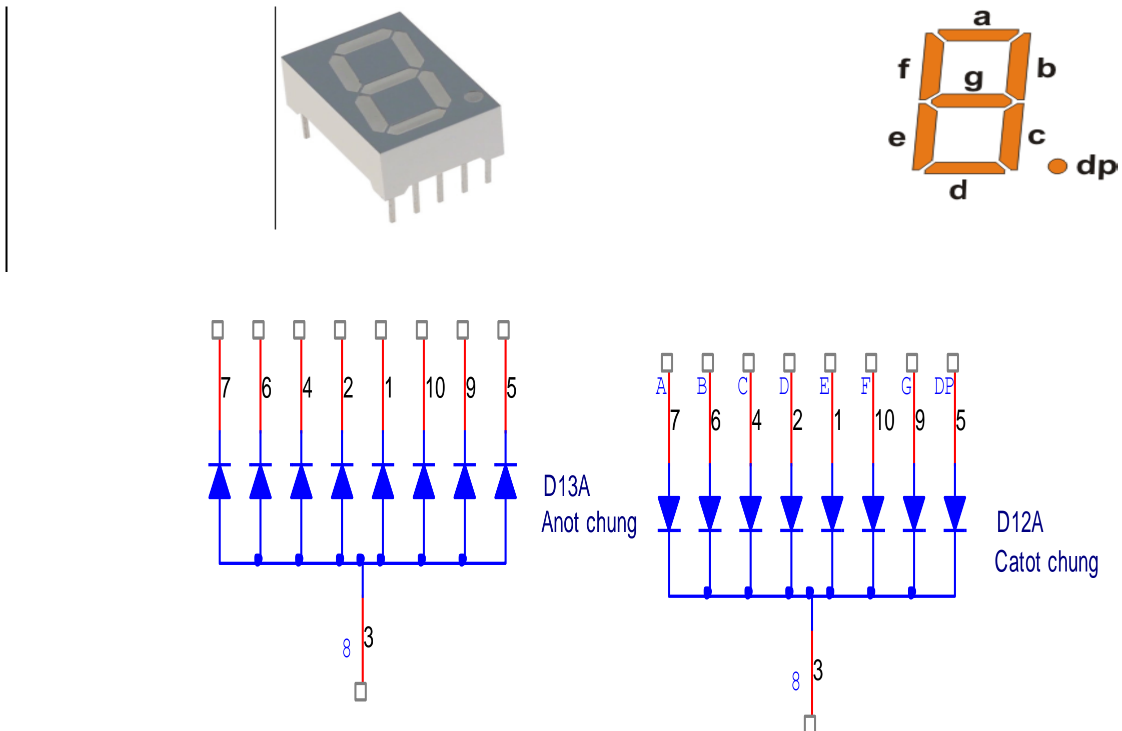
- *Cơ bản về led 7 thanh*
- *Nguyên lí lập trình led 7 thanh.*
- *Ví dụ minh họa*

1. Cơ bản về led 7 thanh

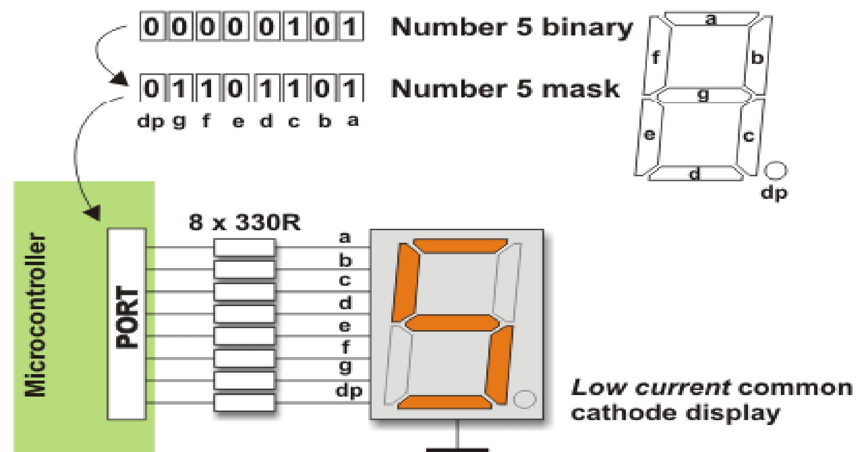
Ở bài học này, chúng ta sẽ học về giao tiếp giữa AVR và led 7 thanh ,các hiển thị số trên led 7 thanh , cũng như các giải thuật về quét led.

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 :



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 catot 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.

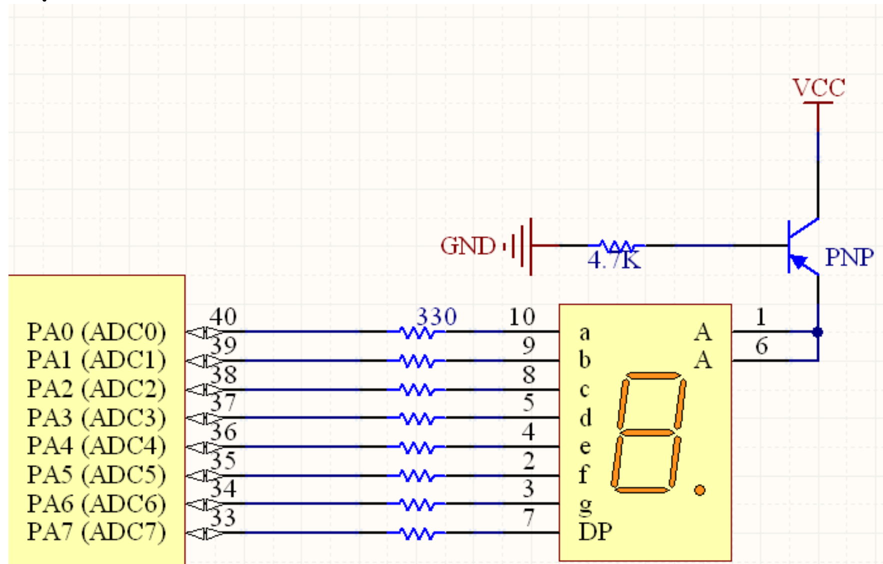
Bằng cách tương tự, ta cũng tạo được giá trị (mã) để xuất ra port của vi điều khiển để led sáng các số từ 0 đến 9. Người ta thường tạo ra 1 bảng mã như vậy như vậy để tiện sử dụng.

LED 7 thanh anot chung và LED 7 thanh catot chung có một đặc điểm khác nhau quan trọng cần phải nhớ khi lập trình cho nó đó là : Với loại anot chung thì ta phải nối đầu anot chung với nguồn 5V thì các thanh LED mới sáng , ngược lại với loại catot chung thì ta phải nối đầu catot với đất. Với từng cách nối phần cứng mà ta có mã LED riêng, vì vậy người lập trình cần phải biết cách mắc LED 7segment thế nào để tạo ra mã LED cho đúng.

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…

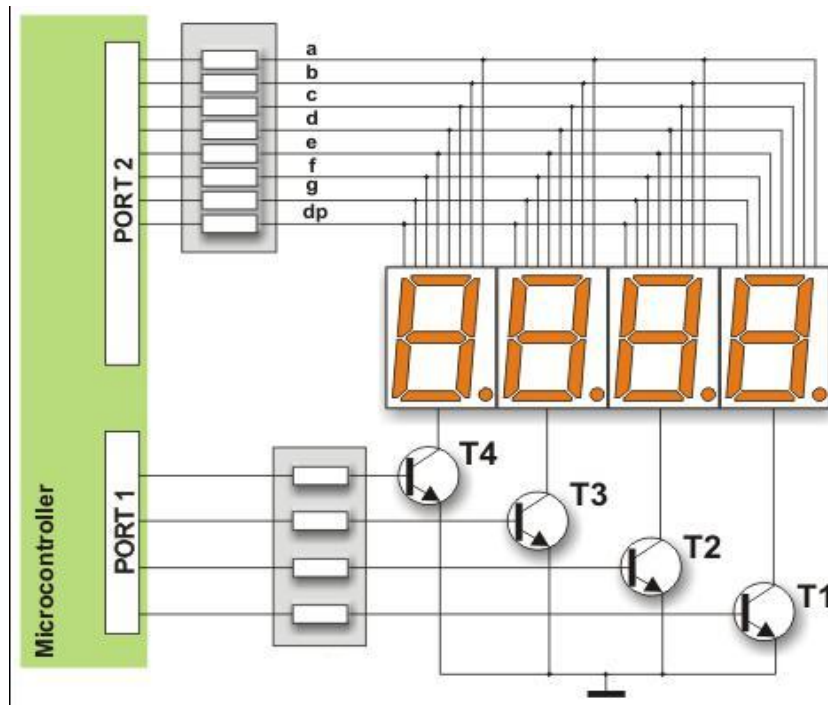
Lần lượt chúng ta xuất từng phần tử của mảng font[] ra cổng nối vào led (port B), khi chạy chương trình, chúng ta sẽ thấy led sáng từ 0 đến 9.

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.

Hướng dẫn :

Để hiển thị số 1234 ta dùng PORTB để truyền dữ liệu số ra LED và PORTA để điều khiển quét LED. Chương trình được viết như sau: (PORTB đóng vai trò như PORT2 và PORTA đóng vai trò như PORT1 trên hình vẽ):

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

unsigned char code[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F} ;

void main(void)
{
  DDRB=0xFF;            //PORTB la cong ra
  DDRA=0xFF;            // PORTA la cong ra
  PORTA=0x00;
```

```
while (1)
    {
      PORTA.0=1;         // PORTA.0 dieu khien LED 1
      PORTB=code[1];
      PORTA.0=0;
      delay_ms(50);
      PORTA.1=1;           // PORTA.1 dieu khien LED 2
      PORTB=code[2];
      PORTA.1=0;
      delay_ms(50);
      PORTA.2=1;           // PORTA.2 dieu khien LED 3
      PORTB=code[3];
      PORTA.2=0;
      delay_ms(50);
      PORTA.3=1;          // PORTA.3 dieu khien LED 4
      PORTB=code[4];
      PORTA.3=0;
      delay_ms(50);

    };

}
```

Như bạn thấy; phương pháp quét LED bằng Tranzitor như trên vừa kém hiệu quả về phần cứng và vừa kém hiệu quả về phần mềm .

Cụ thể :

Về phần cứng nếu ta quét n LED thì ta sẽ mất 1 PORT của vi điều khiển để truyền dữ liệu và n cổng ra khác của vi điều khiển để điều khiển các LED; đó là nhược điểm thứ nhất nhưng chưa hẳn quan trọng .

Về phần mềm : bởi vì quét LED bằng Tranzitor bạn không thể cấu trúc chương trình khác code trên, nếu nhìn vào code thì bạn thấy chip phải làm việc liên tục với chỉ một công việc là chọn LED và gửi dữ liệu ra LED, cứ thế chip sẽ làm việc gần như hết công suất mà chỉ làm được một việc hiển thị LED . Theo kinh nghiệm thực tiễn khi mô phỏng và khi làm mạch thật thì tôi thấy nếu chỉ quét LED theo phương pháp trên thôi thì chip đã làm việc ít nhất 80% công suất . Nói một cách đơn giản, nếu quét LED như vậy thì chip của bạn sẽ không làm thêm được việc khác. Và tất nhiên bạn không thể dùng cả con vi điều khiển chỉ để làm một mục đích duy nhất là quét LED.

Tôi có một giải pháp hữu hiệu giải quyết các vấn đề trên đó là sử dụng IC ghi dịch chốt dữ liệu 74HC595, đầu vào nối tiếp đầu ra song song.Với IC này thì bạn chỉ mất 3 chân vi điều khiển để hiển thị ra số lượng LED tùy chọn, không hạn chế số lượng . Bên cạnh đó, vì là IC ghi dịch chốt dữ liệu nên để hiển thị một số bất kì ra LED thì bạn chỉ cần truyền dữ liệu một lần nên bạn sẽ quản lý chương trình của bạn để chip chỉ phải làm việc vô cùng ít khi hiển thị LED.

Theo kinh nghiệm thì tôi đã dùng chỉ một con ATmega8 để : điều khiển động cơ, đo tốc độ động cơ, hiển thị đồng hồ thời gian thực, đo nhiệt độ môi

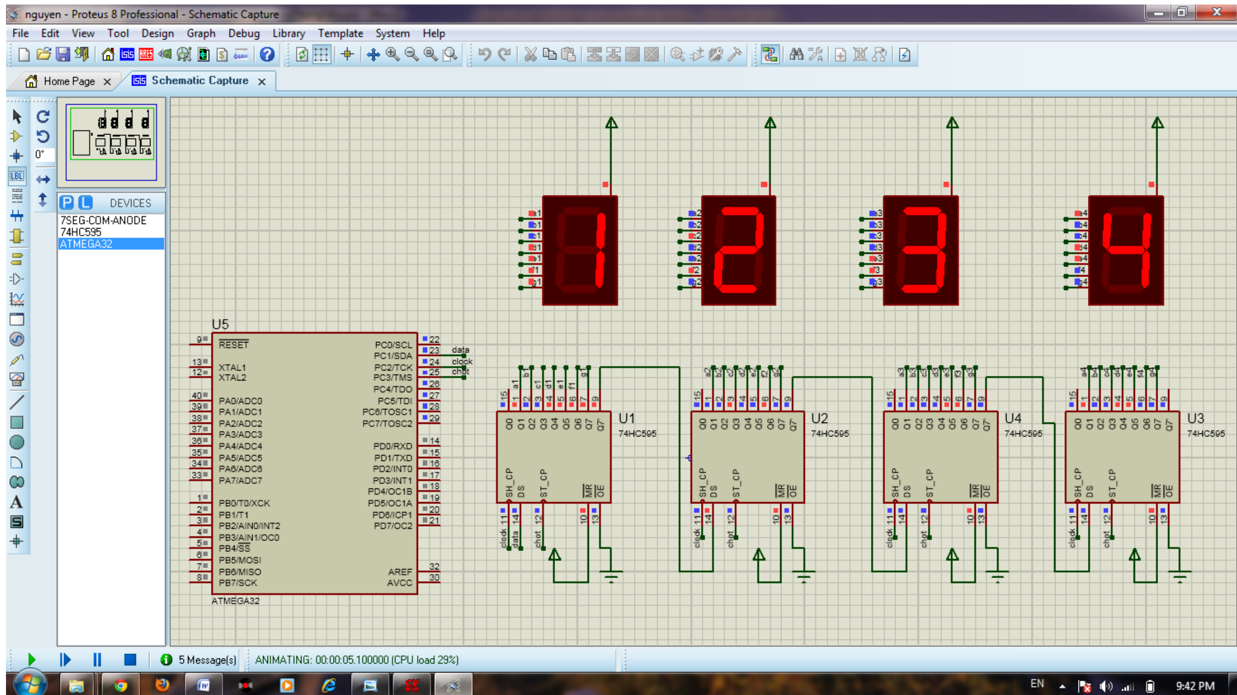
trường,giao tiếp truyền nhận với máy tính, và các giá trị trên đều được hiển thị ra LED 7 seg … vậy mà chip chỉ làm việc xấp xỉ 50% công suất.

Về IC 74HC595 bạn có thể tìm hiểu thêm trong datasheet của nó. Để giúp bạn hiểu rõ hơn thì tôi sẽ làm ví dụ trên dùng 74HC595. Tôi viết thành một hàm truyền nhận để tiện sử dụng:

Tùy vào mục đích sử dụng bạn có thể định thời để định khoảng thời gian hiện thị ra LED nếu số n thây đổi theo thời gian (ví dụ như khi đo tốc độ động cơ, đo nhiệt độ, hiển thị đồng hồ thời gian thực …), về vấn đề định thời thì bạn cần tìm hiểu trong bài Timer.

```
#include <mega32.h>

unsigned char code[]={0x7E,0x4F,0x12,0x06,0x4C,0x24,0x20,0x0F,0x00,0x04} ;
// unsigned char code[]={0x01,0x2F,0x18,0x0A,0x26,0x42,0x40,0x0F,0x00,0x02};
void HienThi (unsigned int n)    // n bat ki, trong vi du nay ta su dung n la mot so co 4 chu so
{
unsigned int a[4];
unsigned char i,j;
a[3]=n/1000;a[2]=(n%1000)/100;a[1]=((n%1000)%100)/10;a[0]=((n%1000)%100)%10;
   for (i=0;i<4;i++)
       for (j=0;j<8;j++)
            {
               PORTC.1=(code[a[i]]>>j)&(0x01);
               PORTC.2=1;
               PORTC.2=0;
            }
 PORTC.3=1;
 PORTC.3=0;
}

void main(void)
{
  DDRC=0xFF;  // PORTC la cong ra
  HienThi(1234);

while (1)
      {

      };
}
```

Kết quả mô phỏng :



Lưu ý : Khi nối tiếp các 74HC595 thì chân 9 của 74HC595 trước nối với chân data của 74HC595 sau (như hình mô phỏng) , thêm nữa là khi mô phỏng chân MR (chân 10) và chân OE (chân 13) có thể để trống nhưng trên mạch thật nếu ta không nối chân MR với nguồn 5V và OE với đất thì không thể hiện thị ra LED được, lỗi này tôi đã gặp khi làm .

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.

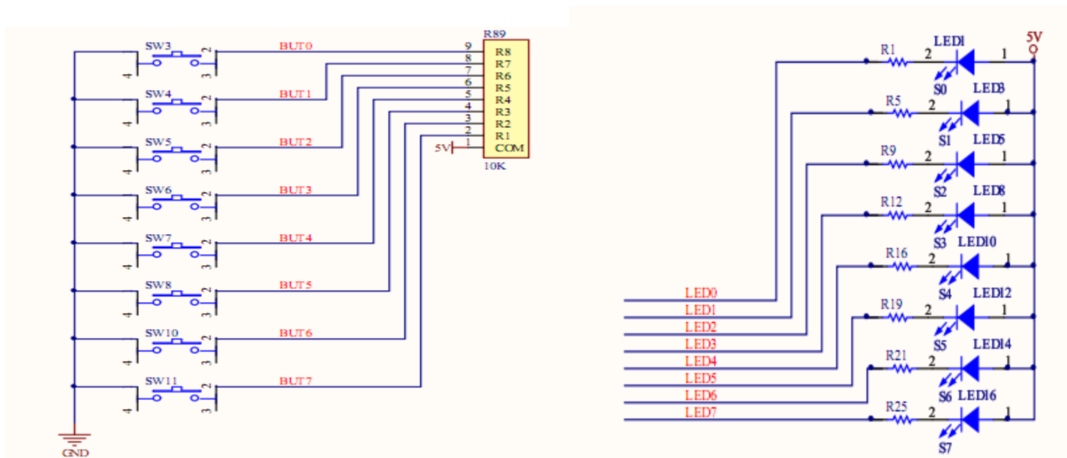
Phím bấm thông dụng nhất có cấu tạo gồm 2 đầu tiếp xúc, mỗi khi chúng ta bấm phím, 2 đầu này sẽ chạm vào nhau (xem hình vẽ ở sơ đồ nguyên lí bên dưới).

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;
    }
}
```

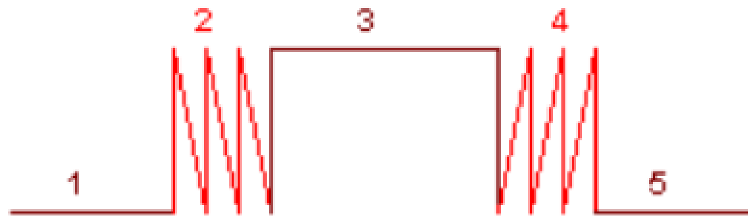
Phân tích chương trình :

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.



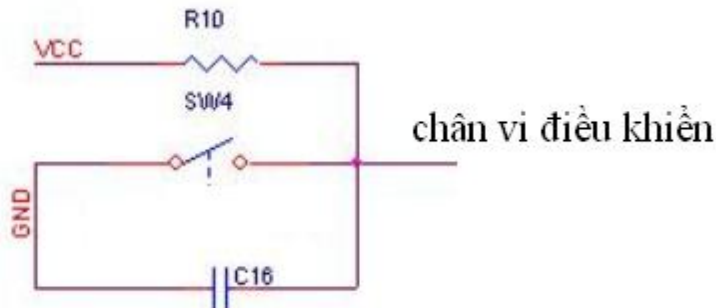
Sơ đồ xung khi bấm phím

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 :

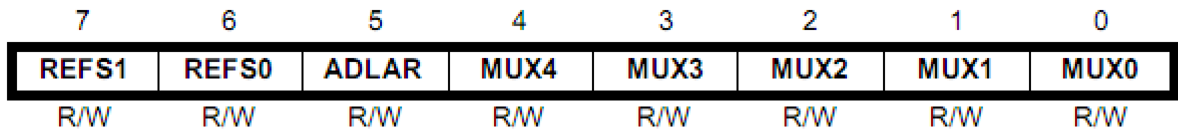
- Chân AVCC chân này bình thường khi thiết kế mạch chúng ta đưa lên Vcc(5V) nhưng khi trong mạch có sử dụng các kênh ADC của phần cứng thì chúng ta phải nối chân này lên Vcc qua 1 cuộn cảm nhằm mục đích cấp nguồn ổn định cho các kênh (đầu vào) của bộ biến đổi.
- Chân AREF chân này cần cấp 1 giá trị điện 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.

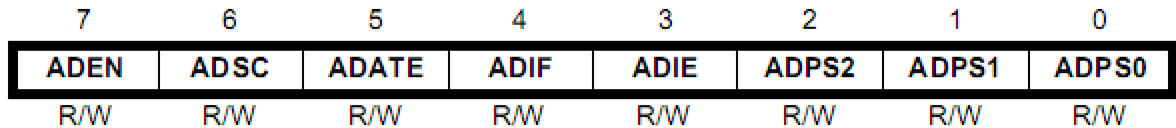
Các thanh ghi liên quan

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.

Tùy vào cách ta set các thanh ghi như thế nào mà ta chọn được điện áp tham chiếu, độ phân giải,chế độ chuyển đổi đơn kênh, chuyển đổi đa kênh … như mong muốn.

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

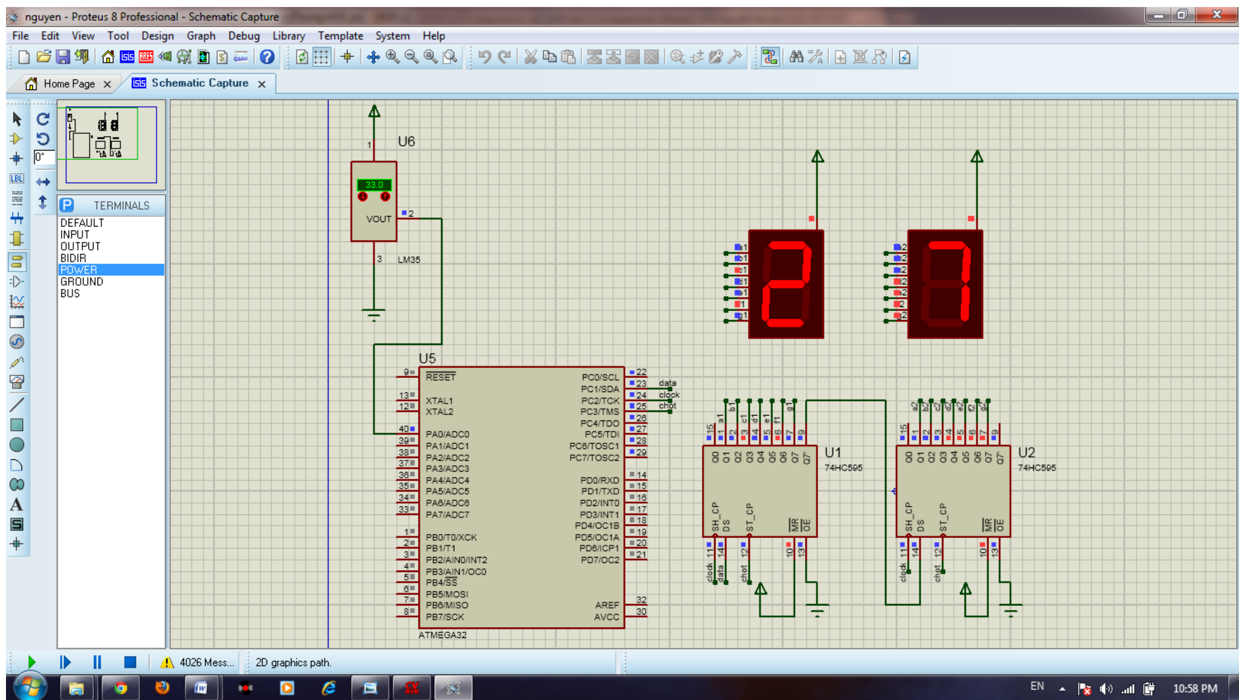
Bạn hãy phân tích chương trình trên và chỉ ra các chế độ hoạt động của module ADC được cấu hình như trên.

Để cụ thể hơn về chức năng của chuyển đổi ADC ta nêu ra một thí dụ : đo nhiệt độ môi trường và hiển thị kết quả ra LED 7 segment.
Cách làm tương tự trên, ở đây tôi chỉ đưa ra code và kết quả mô phỏng :

Lưu ý hàm HienThi ta đã viết ở phần hiển thị LED 7seg nên ta chỉ việc sử dụng, lưu ý là số n thay vì 4 chữ số thì ta thay bằng số có hai chữ số vì nhiệt độ môi trường chỉ nằm trong phạm vi hai chữ số. Thêm vào đó, do nhiệt độ là đại lượng biến đổi chậm nên ta quy định cứ sau một khoảng thời gian bằng hai lần Timer0 ngắt thì ta đọc giá trị nhiệt độ 1 lần, với chu kì đọc khá nhỏ đó thì nhiệt độ đọc được là khá chính xác. Đương nhiên phần Timer bạn phải tìm hiểu ở bài sau. Thêm nữa là ta dùng LM35 để đo nhiệt độ, cứ 1^0C ứng với 10mV nên (bạn nên chú ý để chuyển đổi cho đúng).

```
#include <mega32.h>
#include <delay.h>
 #define ADC_VREF_TYPE 0x00
 unsigned char input;
unsigned char code[]={0x7E,0x4F,0x12,0x06,0x4C,0x24,0x20,0x0F,0x00,0x04} ;
unsigned int read_adc(unsigned char adc_input)  //adc_channel
{
ADMUX=adc_input | (ADC_VREF_TYPE & 0xFF);
delay_us(10);
ADCSRA|=0x40;
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}
// Auto Update Value
   interrupt [ADC_INT] void adc_isr(void) {
   input=read_adc(0);
   ADCSRA|=0x40;//START CONVERTSION
   }
 unsigned char k;
 void HienThi(unsigned char);
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
   TCNT0=0x00;
   k=k+1;
   if(k==2)
      {
      k=0;
       HienThi(500*input/1023);
```

```
        ADCSRA|=0x40;
      }
}
void main(void)
{
  DDRC=0xFF;
  DDRA=0x00;
//Timer 0
  TCCR0=0x05;
  TCNT0=0x00;
  TIMSK=0X01;
// Set ADC
  ADMUX=ADC_VREF_TYPE  &0xFF  ;
  ADCSRA=0xCF;// 11010111 ADEN=1;ADIE=1, PrescalE=128
    #asm("sei")
 while (1)
      {
  };
}
```



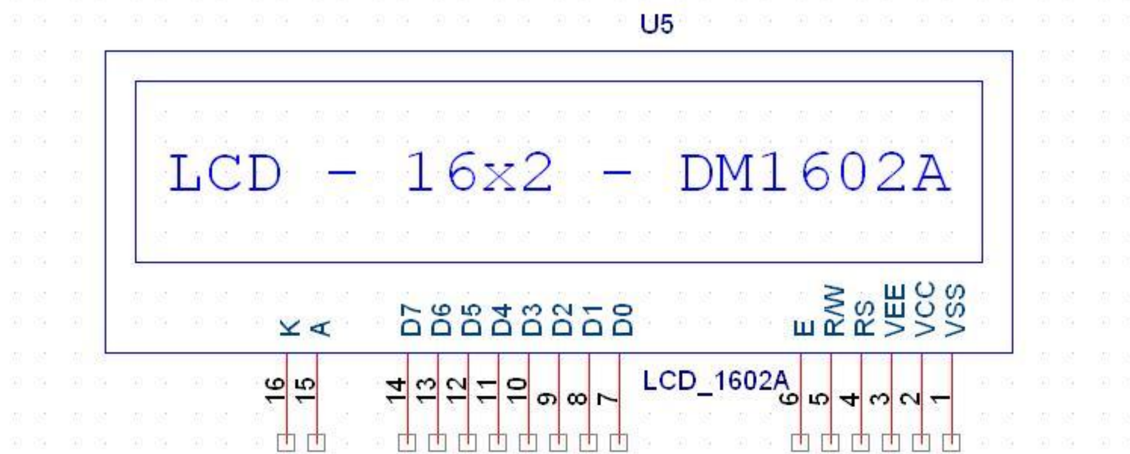
BÀI 6 : GIAO TIẾP LCD

- *Giới thiệu về LCD 16x2, Lập trình*
- *Cách cấu hình cho LCD trong Code Vision cho Atmega32*

1. Giới thiệu về LCD 16x2, Lập trình

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.

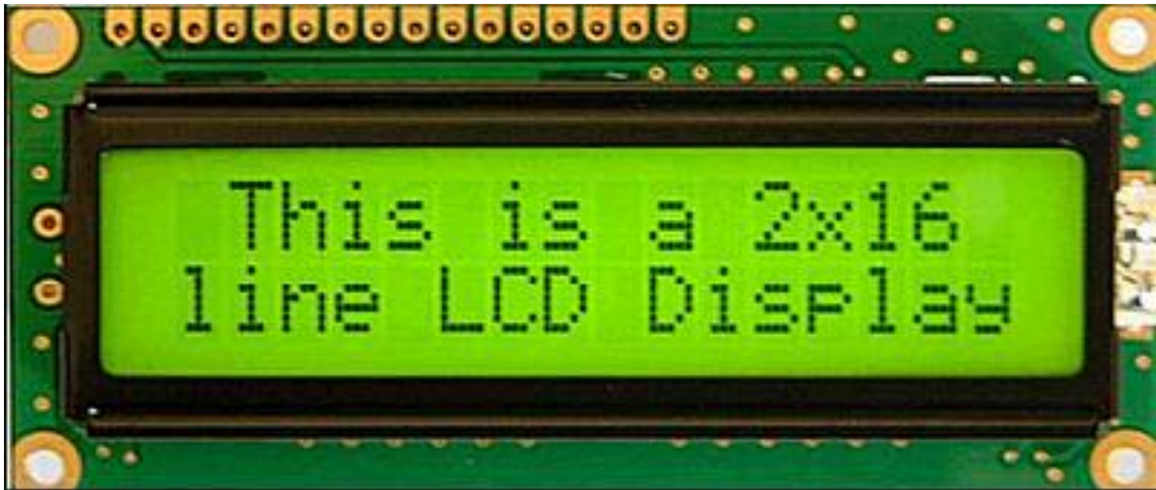
LCD là từ viết tắt của Liquid Crystal Display (màn hình tinh thể lỏng). Có nhiều loại màn hình LCD với các kích cỡ khác nhau, ví dụ như LCD 16x1 (16 cột và 1 hàng), LCD 16x2 (16 cột và 2 hàng), LCD 20x2 (20 cột và 2 hàng)… Trong bài học này, ta xét loại LCD 16x2 bán phổ biến trên thị trường.



Sơ đồ nguyên lí của LCD 16x2

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

LCD có hai thanh ram đó là DDRAM thanh này chứa mã lệnh của LCD; CGRAM chứa mã kí tự. Để truy cập vào 2 thanh ram này ta có 2 thanh ghi lệnh và thanh ghi data



Chức năng của các chân LCD :

Pin Description and Wiring Diagram

Pin No.	Symbol	External Connection	Function Description
1	VSS	Power Supply	Ground
2	VDD	Power Supply	Supply Voltage for logic (+5.0V)
3	V0	Adj Power Supply	Power supply for contrast (approx. 0.5V)
4	RS	MPU	Register select signal. RS=0: Command, RS=1: Data
5	R/W	MPU	Read/Write select signal, R/W=1: Read R/W: =0: Write
6	E	MPU	Operation enable signal. Falling edge triggered.
7-10	DB0 – DB3	MPU	Four low order bi-directional three-state data bus lines. These four are not used during 4-bit operation.
11-14	DB4 – DB7	MPU	Four high order bi-directional three-state data bus lines.
15	LED+	Power Supply	Power supply for LED Backlight (+5.0V via on-board resistor)
16	LED-	Power Supply	Ground for Backlight

Các chân VDD(VCC), VSS và V0(VEE)

Chân VDD cấp dương nguồn 5V, chân VSS nối đất, chân V0 được dùng để điều khiển độ tương phản của màn hình LCD.

RS (Register select)

Khi RS=0 thanh ghi lệnh được chọn, khi này ta có thể truyền các lệnh như xóa màn hình, nhảy con trỏ, các lệnh khác của LCD sẽ được cho trong bảng.
Khi RS=1 thanh ghi kí tự (data) được chọn, khi này ta có thể truyền các kí tự trong bảng mã ASCII. Các kí tự có mã được tra trong bảng…

R/W (Read/Write)

R/W=0 ghi dữ liệu(lệnh hoặc kí tự) từ MPU->LCD,R/W=1 đọc dữ liệu từ LCD, đối với LCD ta chỉ đọc bit busy D7

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(độ rộng này mình thấy mỗi datasheet viết một khác nên các bạn hãy cho nó lớn vừa phải để dữ liệu truyền được). 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

Các chân truyền lệnh hoặc kí tự đến LCD . Chân D7 được dùng để kiểm tra bit busy khi RS=0 và R/W=1 nếu D7=1 thì LCD đang bận LCD sẽ không nhận data từ MPU. Khi D7=0 thì LCD không bận.

Bảng mã lệnh của LCD

Command Set

Function	LCD_RS	LCD_RW	Upper Nibble				Lower Nibble			
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Clear Display	0	0	0	0	0	0	0	0	0	1
Return Cursor Home	0	0	0	0	0	0	0	0	1	-
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S
Display ON/OFF	0	0	0	0	0	0	1	D	C	B
Cursor and Display Shift	0	0	0	0	0	1	S/C	R/L	-	-
Function Set	0	0	0	0	1	0	1	0	-	-
Set CG RAM Address	0	0	0	1	A5	A4	A3	A2	A1	A0
Set DD RAM Address	0	0	1	A6	A5	A4	A3	A2	A1	A0
Read Busy Flag and Address	0	1	BF	A6	A5	A4	A3	A2	A1	A0
Write Data to CG RAM/DD RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Read from CG RAM/DD RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0

Bảng mã địa chỉ hiển thị của LCD

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Ví dụ: set địa chỉ thứ 5 thì ta gửi mã lệnh đến LCD set DD RAM address là 0x85

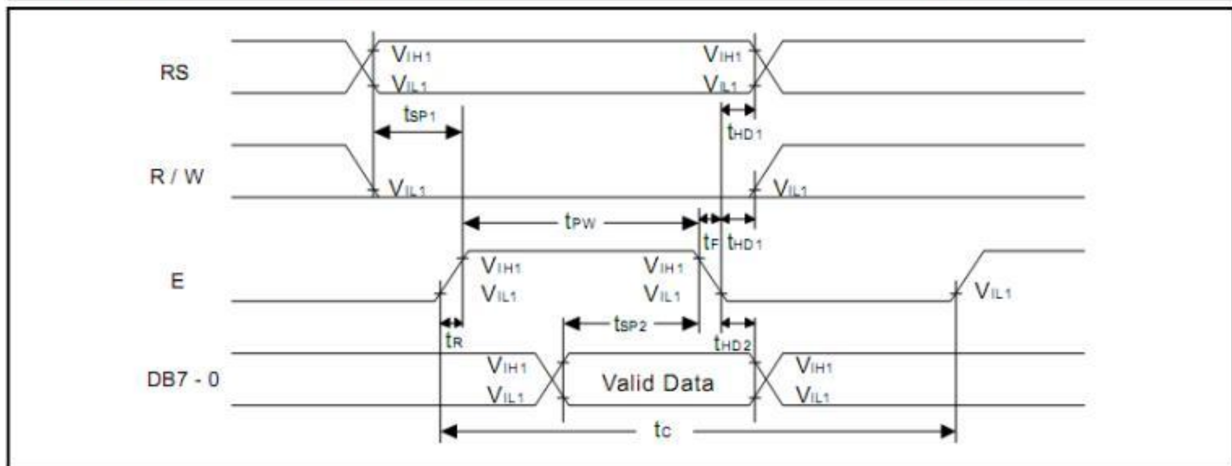
Bảng mã kí tự ascii

Built-in Font Table

Upper 4 Bits / Lower 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	@	P	`	p				ー	タ	ミ	α	p
xxxx0001	(2)		!	1	A	Q	a	q			。	ア	チ	ム	ä	q
xxxx0010	(3)		"	2	B	R	b	r			「	イ	ツ	メ	β	θ
xxxx0011	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
xxxx0100	(5)		$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	σ	ü
xxxx0110	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)		'	7	G	W	g	w			ァ	キ	ヌ	ラ	g	π
xxxx1000	(1)		(	8	H	X	h	x			ィ	ク	ネ	リ	√	x̄
xxxx1001	(2)		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010	(3)		*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
xxxx1011	(4)		+	;	K	[	k	{			ォ	サ	ヒ	ロ	×	万
xxxx1100	(5)		,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
xxxx1101	(6)		-	=	M	]	m	}			ュ	ス	ヘ	ン	£	÷
xxxx1110	(7)		.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
xxxx1111	(8)		/	?	O	_	o	←			ッ	ソ	マ	゜	ö	█

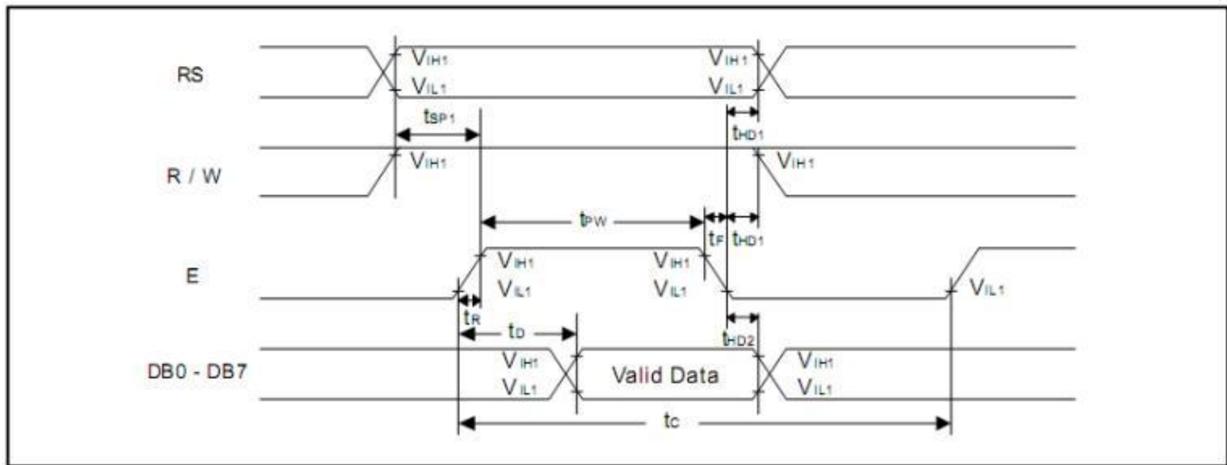
Phân khe thời gian của LCD chế độ ghi :

Characteristics	Symbol	Limit			Unit	Test Condition
		Min.	Typ.	Max.		
E Cycle Time	t_C	500	-	-	ns	Pin E
E Pulse Width	t_{PW}	230	-	-	ns	Pin E
E Rise/Fall Time	t_R, t_F	-	-	20	ns	Pin E
Address Setup Time	t_{SP1}	40	-	-	ns	Pins: RS, R/W, E
Address Hold Time	t_{HD1}	10	-	-	ns	Pins: RS, R/W, E
Data Setup Time	t_{SP2}	80	-	-	ns	Pins: DB0 - DB7
Data Hold Time	t_{HD2}	10	-	-	ns	Pins: DB0 - DB7



Phân khe thời gian của LCD chế độ đọc :

Characteristics	Symbol	Limit			Unit	Test Condition
		Min.	Typ.	Max.		
E Cycle Time	t_C	500	-	-	ns	Pin E
E Pulse Width	t_W	230	-	-	ns	Pin E
E Rise/Fall Time	t_R, t_F	-	-	20	ns	Pin E
Address Setup Time	t_{SP1}	40	-	-	ns	Pins: RS, R/W, E
Address Hold Time	t_{HD1}	10	-	-	ns	Pins: RS, R/W, E
Data Output Delay Time	t_D	-	-	120	ns	Pins: DB0 - DB7
Data hold time	t_{HD2}	5.0	-	-	ns	Pin DB0 - DB7



Căn cứ vào phân khe thời gian ta có thể lập trình cho quá trình ghi và đọc của LCD

Để lập giao tiêp được với LCD trước hết các bạn phải khởi tạo cho LCD;

1. **Function set: (bắt buộc)**

    -DL=1; giao tiếp thông qua 8bit

    -DL=0; giao tiếp thông qua 8bit D4->7

    N=0; LCD chỉ hiển thị 1 hàng

    N=1; LCD chỉ hiển thị 2 hàng

    F=0; 5x8 dot;

    F=1;5x10 dot. Chú ý: khi 2 hàng thì F=1 không có tác dụng khi đó chỉ hiể thị 5x8 dot

2. **Display on/off control**

    D=0; Tắt hiển thị/ D=1; Hiển thị

    C=0; Tắt con trỏ/ C=1; Mở con trỏ

    B=0; Tắt nháy trỏ; B=1 bật nháy trỏ

3. **Entry mode set:**

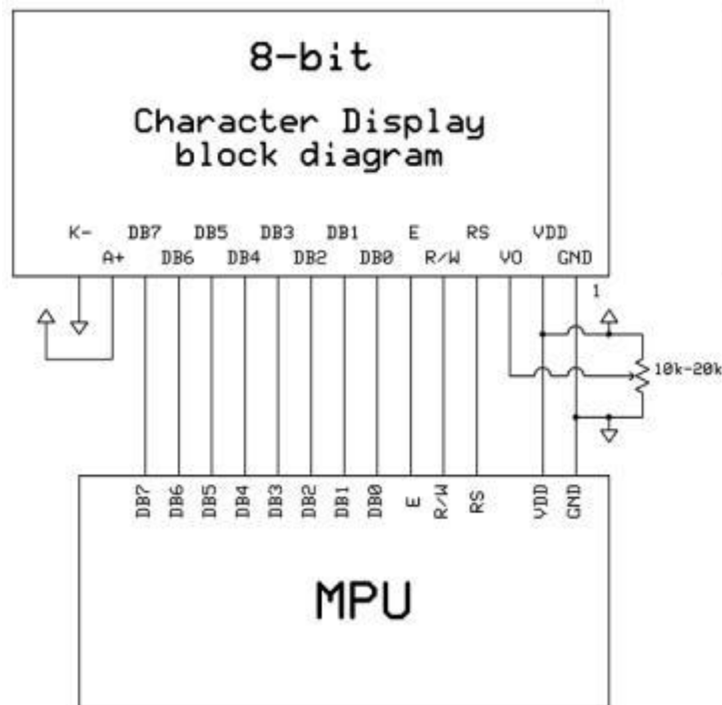
    I/D=1; con trỏ tự động tăng địa chỉ sau khi truyền một kí tự

    S =0 no shift

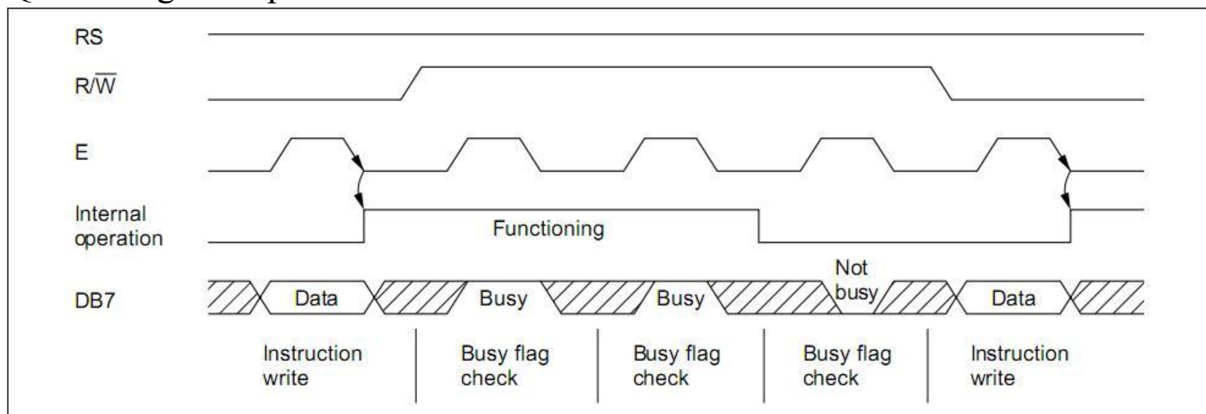
Các bạn tham khảo mã lệnh có thể tự khởi tạo theo y của mình. Chú ý khởi tạo của 4bit và 8bit khác nhau rất nhiều các bạn xem thật kĩ trong datasheet

Truyền dữ liệu theo 8bit

Sơ đồ nối chân.



Quá trình giao tiếp



Dựa phân khe thời gian ta xây dựng một số hàm đọc và viết vào LCD ; ở đây mình viết toàn bộ chương trình luôn; dùng biên dịch là codevision

Tạo thư viên: Library.h

```
#include <mega32.h>
#include <delay.h>
#define LCD_RS PORTB.0
#define LCD_RW PORTB.1
#define LCD_EN PORTB.2
#define LCD_RD7 PIND.7
#define LCD_data PORTD
```

```
char LCD_check_bit_busy();
void LCD_busy(void);
void writeCmd(unsigned char cmd);
void writeData(unsigned char data);
void LCD_init();
```

Khai báo các hàm trong Library.h bởi file Library.c

```
#include "Library.h"
char LCD_check_bit_busy()
{
char address;
LCD_RS=0;
LCD_RW=1;
DDRD=0x00;      // Công D là công vào
LCD_EN=1;
delay_us(5);
address=PIND.7;
LCD_EN=0;
DDRD=0xff;
return address;

}
void LCD_busy(void)
{
unsigned char busy;
busy=LCD_check_bit_busy();
while(busy)
{
busy=LCD_check_bit_busy();
};
}
void writeCmd(unsigned char cmd)
{
LCD_data=cmd;
LCD_RS=0;
LCD_RW=0;
LCD_EN=1;
delay_us(5);
LCD_EN=0;
LCD_busy();
}
```

```
void writeData(unsigned char data)
{
LCD_data=data;
LCD_RS=1;
LCD_RW=0;
LCD_EN=1;
delay_us(5);
LCD_EN=0;
LCD_busy();
}
void LCD_init()
{
delay_ms(100);
writeCmd(0x30);
delay_ms(30);
writeCmd(0x30);
delay_ms(10);
writeCmd(0x30);
writeCmd(0x38);//Function set: 2 Line, 8-bit,5x7 dot
writeCmd(0x0f);// Display on, curson blinking command
writeCmd(0x01);// Clear LCD
writeCmd(0x06);//Entry mode,auto increment with no shift
} LCD_RS=0;
LCD_RW=0;
LCD_EN=1;
LCD_EN=0;
LCD_busy();
}
```

Chương trình chính main.c

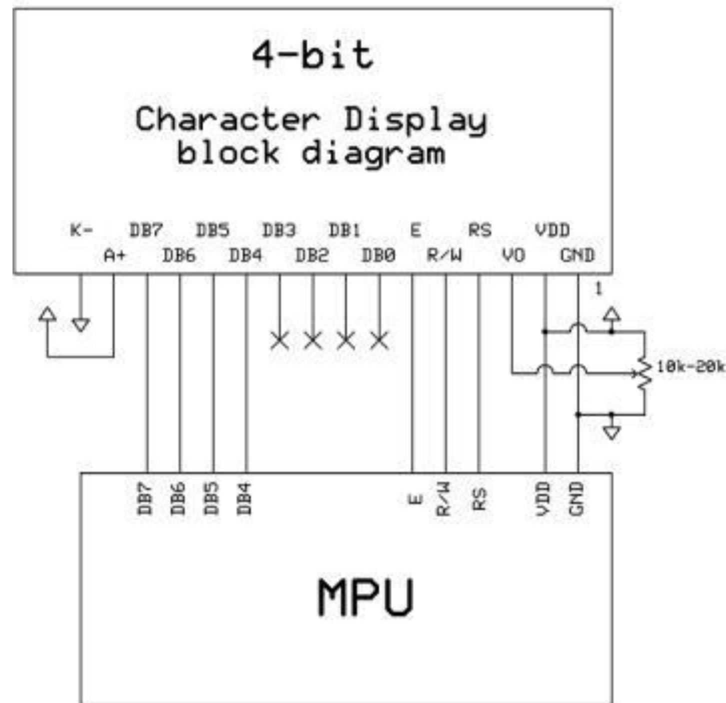
```
#include "Library.h"
void main(void)
{
DDRB=0x0f;
DDRD=0xff;
LCD_init();
writeCmd(0x80+0x02);
writeData('D');
writeData('U');
writeData('O');
writeData('N');
writeData('G');
```

```
writeData('-');
writeData('V');
writeData('A');
writeData('N');
writeData('-');
writeData('H');
writeData('A');
writeCmd(0x80+0x42);
writeData('K');
writeData('S');
writeData('T');
writeData('N');
writeData('-');
writeData('C');
writeData('D');
writeData('T');
writeData('-');
writeData('K');
writeData('5');
writeData('4');
}
```

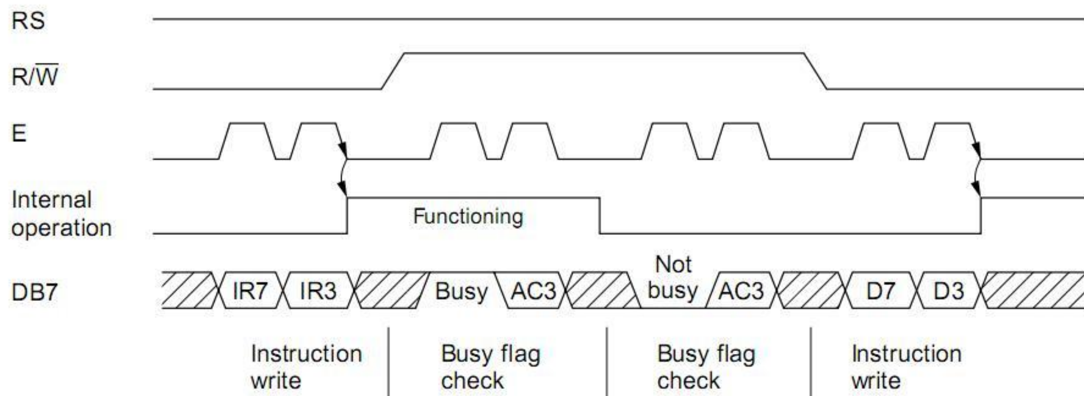
Các bạn có thể tự mình tạo ra hàm đọc chuỗi kí tự theo cách của mình dựa vào các hàm con ở trên, ở đây mình không giới thiệu.

Truyền dữ liệu theo 4bit

Sơ đồ nối chân.



Quá trình giao tiếp



Khi truyền 4bit ta truyền(đọc hoặc ghi) 4bit cao trước sau đó truyền 4bit thấp
Chương trình 4 bit:
File Library.h

```
#include <mega32.h>
#include <delay.h>
#define LCD_RS PORTB.0
#define LCD_RW PORTB.1
#define LCD_EN PORTB.2
```

```
void EN_Clock();//enable pulse width  >= 300ns  this datasheet
char LCD_check_bit_busy();
void LCD_busy();
void writeCmd(unsigned char cmd);
void writeData(unsigned char data);
void LCD_init();
void Data(unsigned char data);
```

File Library.c

```
#include "Library.h"
void Data(unsigned char data)
{
PORTB.4=data&0b00010000;
PORTB.5=data&0b00100000;
PORTB.6=data&0b01000000;
PORTB.7=data&0b10000000;
}
void EN_Clock()
{
LCD_EN=1;
delay_us(3); //enable pulse width  >= 300ns
LCD_EN=0;
}
char LCD_check_bit_busy()
{
char address_low,address_high;
LCD_RS=0;
LCD_RW=1;
DDRB=0x0f;       // 4bit cao Công D là công vào
LCD_EN=1;
delay_us(5);
address_high=PINB&0xf0;
LCD_EN=0;
delay_us(10);
LCD_EN=1;
delay_us(5);
address_low=PINB&0xf0;
LCD_EN=0;
DDRB=0xff;
address_low>>=4;
return (address_high|address_low);
```

```
}
void LCD_busy()
{
char busy;
busy=LCD_check_bit_busy();
busy=busy>>7;
while(busy)
{
busy=LCD_check_bit_busy();
busy=busy>>7;
};
}
void writeCmd(unsigned char cmd)
{
LCD_RS=0;
LCD_RW=0;
LCD_EN=1;
Data(cmd);
delay_us(5); //enable pulse width  >= 300ns
LCD_EN=0;
cmd=cmd<<4;
delay_us(10);
LCD_EN=1;
Data(cmd);

delay_us(5); //enable pulse width  >= 300ns
LCD_EN=0;
LCD_busy();
}
void writeData(unsigned char data)
{
LCD_RS=1;
LCD_RW=0;
LCD_EN=1;
Data(data);
delay_us(5); //enable pulse width  >= 300ns
LCD_EN=0;
data=data<<4;
delay_us(10);
LCD_EN=1;
```

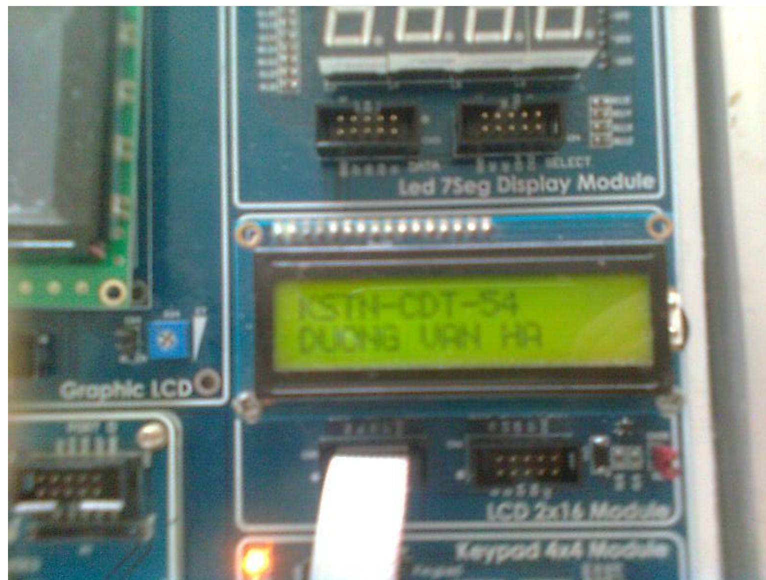
```
Data(data);
delay_us(5); //enable pulse width  >= 300ns
LCD_EN=0;
LCD_busy();
}
void LCD_init()
{
DDRB=0xff;
PORTB=0xff;
delay_ms(20);
LCD_RS=0;
LCD_RW=0;
Data(0b00110000);
delay_ms(6);
EN_Clock();
delay_ms(1);
EN_Clock();
delay_ms(1);
EN_Clock();
delay_ms(1);
Data(0b00100000);
EN_Clock();
LCD_busy();
writeCmd(0x28);//Function set: 2 Line, 4-bit,5x8 dot
writeCmd(0x10);//Set cursor
writeCmd(0x0e);// Display on, curson blinking command
writeCmd(0x01);// Clear LCD
writeCmd(0x06);//Entry mode,auto increment with no shift
}
```

File main.c

```
#include "Library.h"
void main(void)
{
LCD_init();
writeData('K');
writeData('S');
writeData('T');
writeData('N');
writeData('-');
writeData('C');
writeData('D');
```

```
writeData('T');
writeData('-');
writeData('5');
writeData('4');
writeCmd(0x80+64);      //line 2
writeData('D');
writeData('U');
writeData('O');
writeData('N');
writeData('G');
writeData(' ');
writeData('V');
writeData('A');
writeData('N');
writeData(' ');
writeData('H');
writeData('A');
}
```

Kết quả:



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 :

[LCD]	[AVR Port]
RS (pin4)	bit 0
RD (pin 5)	bit 1
EN (pin 6)	bit 2
DB4 (pin 11)	bit 4
DB5 (pin 12)	bit 5
DB6 (pin 13)	bit 6
DB7 (pin 14)	bit 7

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),…*

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);
}
```

3.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);

BÀI 7 : GIAO TIẾP VỚI LED MA TRẬN

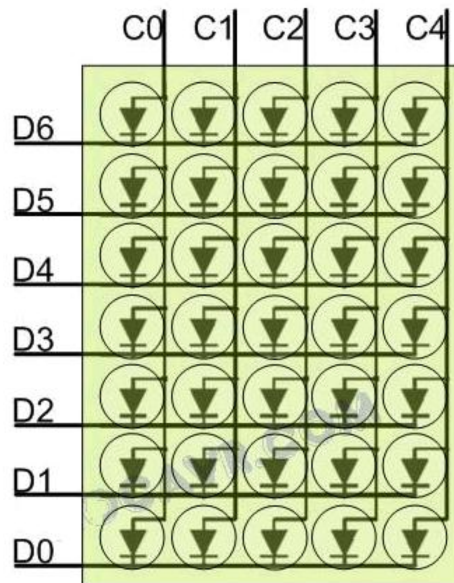
- *Cơ bản về led ma trận*
- *Cách tạo font cho led ma trận*
- *Ví dụ minh họa*

---

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 :

								18
								24
								42
								81
								ff
								81
								81
								81

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 :

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).

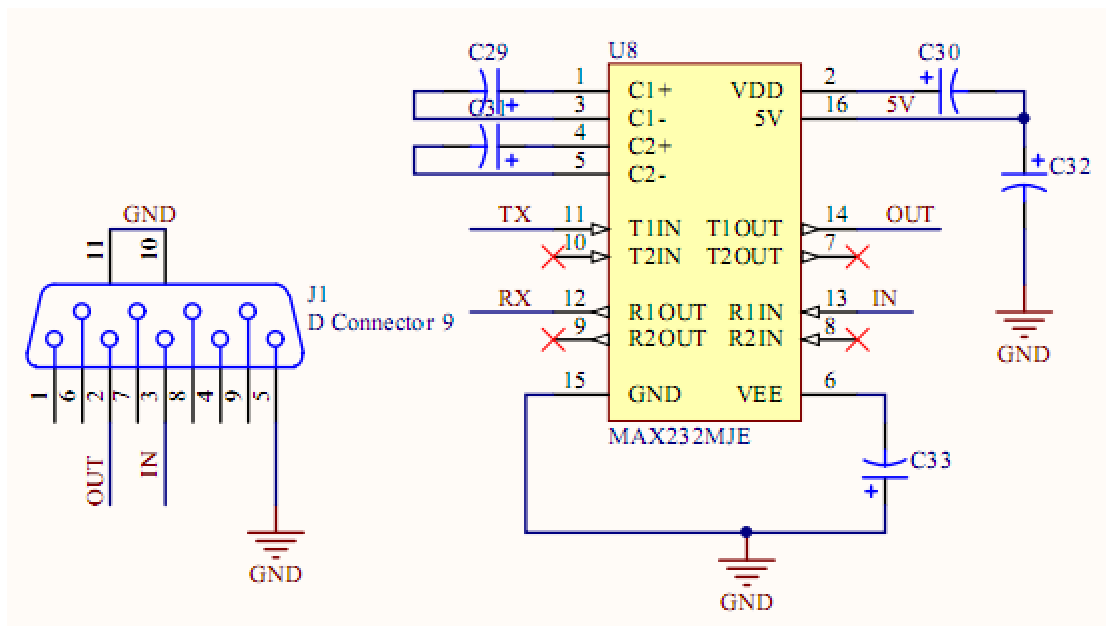
void putchar(char c)
Hàm này truyền một kí tự thông qua module UART

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 :



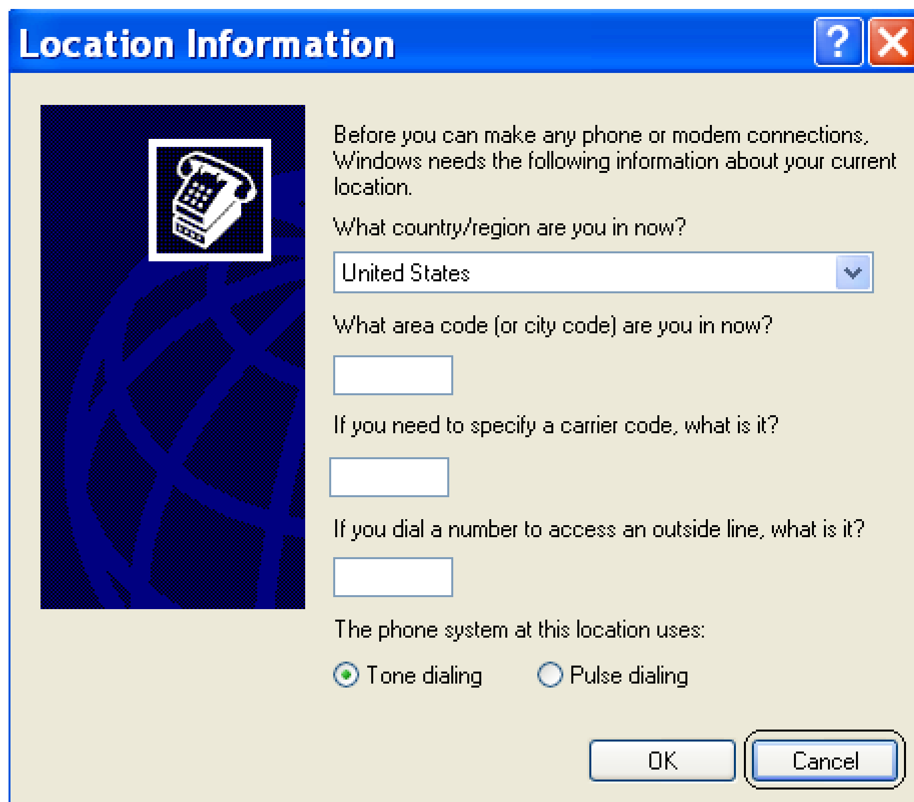
Trong mạch nguyên lí, chúng ta sử dụng thêm 1 IC max 232 để chuyển điện áp tương ứng với 2 mức logic 0 và 1 của vi điều khiển thành điện áp ở mức logic tương ứng với AVR, hai chân 11 và 12 của max232 được nối với 2 chân TX và RX của vi điều khiển.

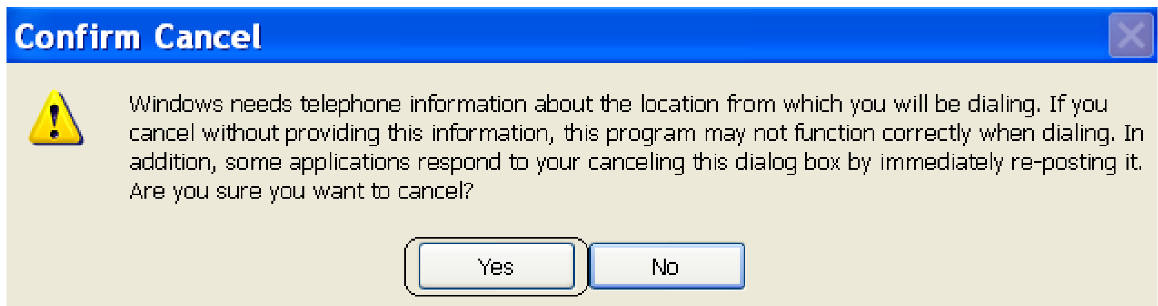
Chương trình :

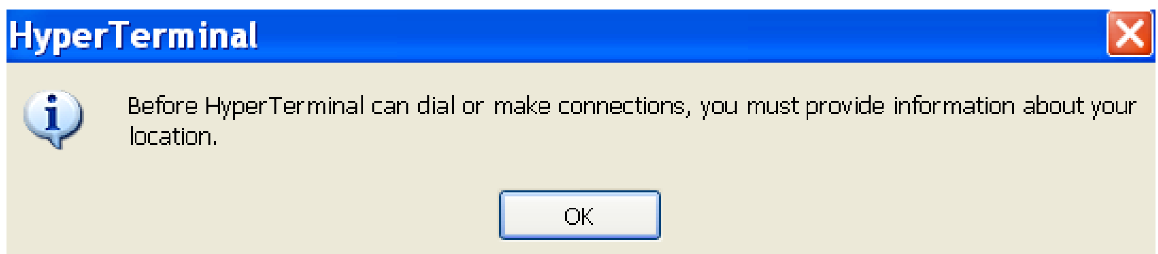
```
#include <mega32.h>
#include <delay.h>

// 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**


Location Information
Before you can make any phone or modem connections, Windows needs the following information about your current location.
What country/region are you in now?
United States
What area code (or city code) are you in now?
If you need to specify a carrier code, what is it?
If you dial a number to access an outside line, what is it?
The phone system at this location uses:
Tone dialing
Pulse dialing
OK
Cancel

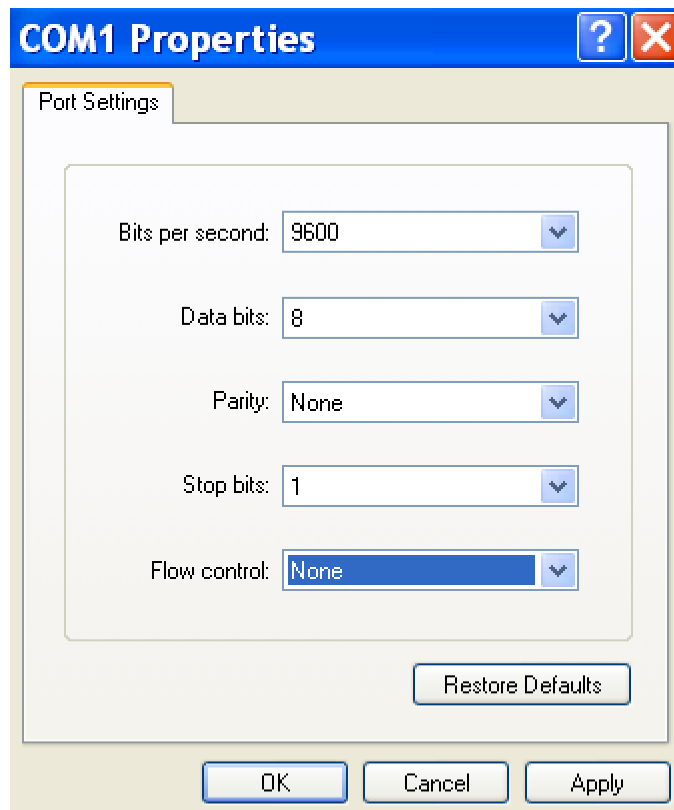

Confirm Cancel
Windows needs telephone information about the location from which you will be dialing. If you cancel without providing this information, this program may not function correctly when dialing. In addition, some applications respond to your canceling this dialog box by immediately re-posting it. Are you sure you want to cancel?
Yes
No


HyperTerminal
Before HyperTerminal can dial or make connections, you must provide information about your location.
OK

- Sau đó chúng ta nhập mô tả kết nối :
-

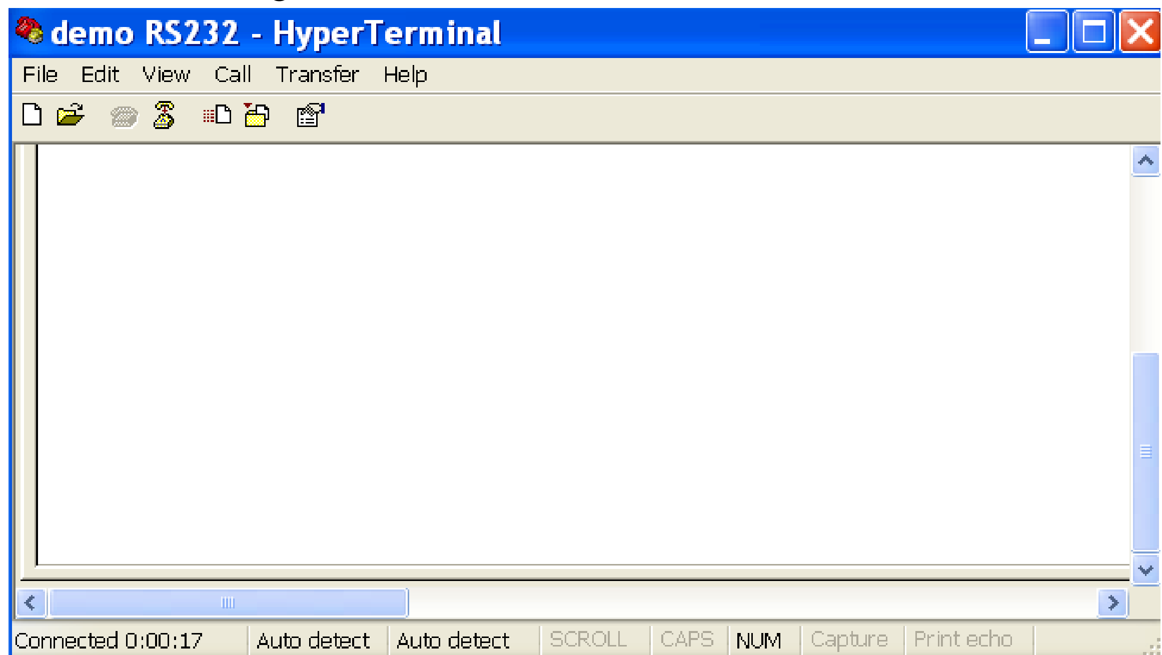
- Nếu lại xuất hiện hộp thoại nhắc nhở nhập tên thông tin khu vực, chúng ta làm như trên.
- Sau đó, chúng ta chọn các thông số để thiết lập kết nối :

Chọn cổng COM để kết nối

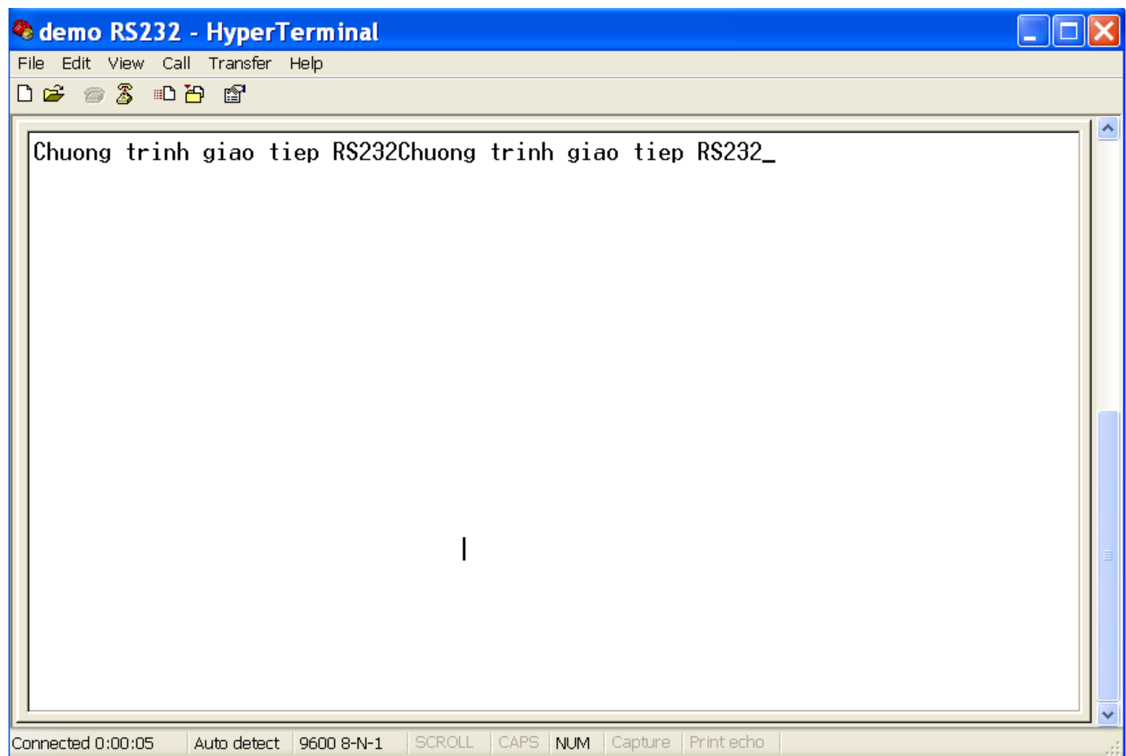


Chọn tốc độ baud, số bit dữ liệu trên 1 khung truyền và parity

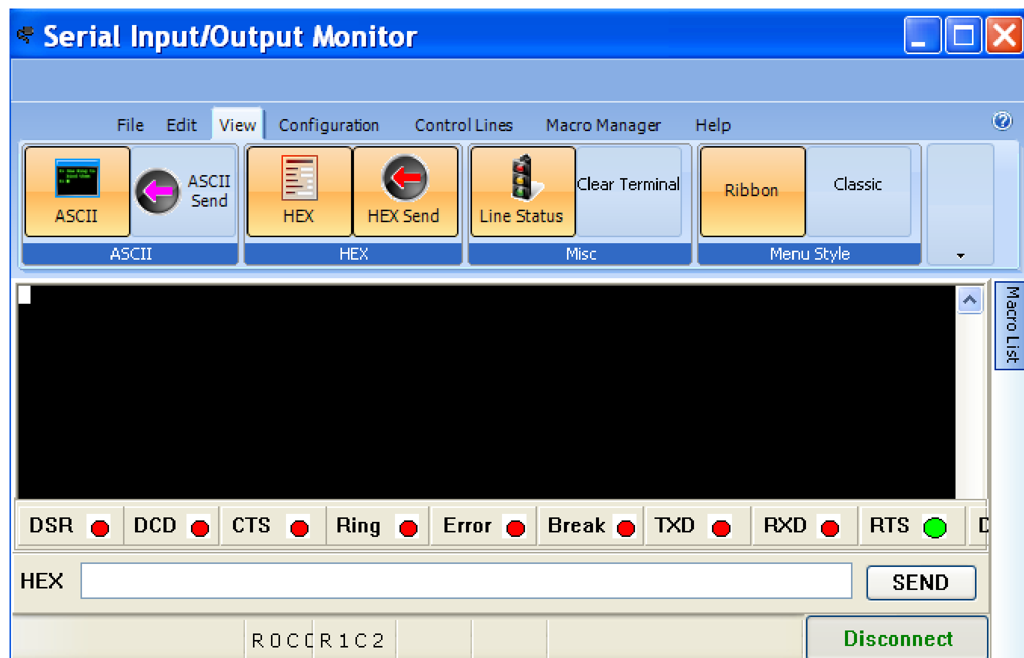
- Giao diện của chương trình như sau :

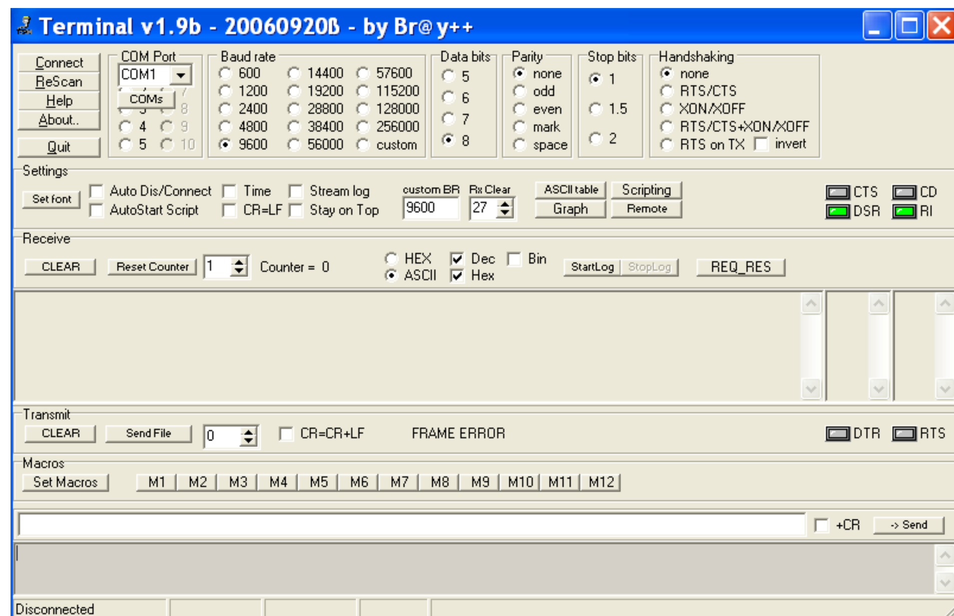


- Kết quả, chúng ta được như sau :



- 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 :





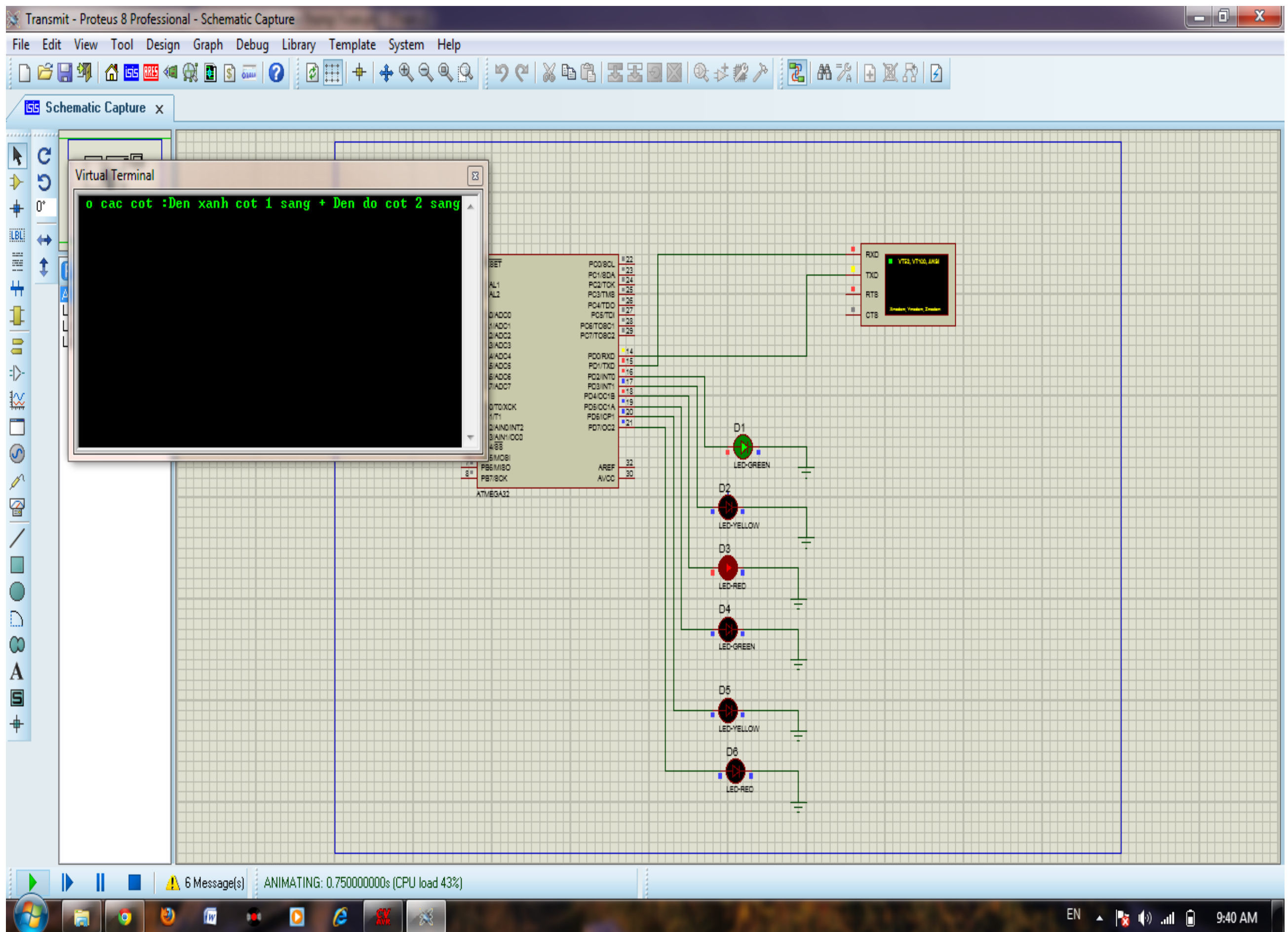
Bài tập:

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.

Những nét cơ bản về giao tiếp với máy tính đã được trình bày ở trên, sau đây là một ví dụ : Điều khiển đèn giao thông và hiển thị lên máy tính trạng thái của các đèn trên cột đèn.

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

unsigned int i=0,j=0,k=0;
//unsigned char code[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F} ;
// Timer 1 overflow interrupt service routine
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
        i=i+1;
        TCNT1=26474;
        TIFR=0x00;

        if(i<2) { PORTD.0=1; printf("\nDen xanh cot 1 sang + Den do cot 2 sang");}  // xanh1
else PORTD.0=0;
        if((i>=2)&&(i<3)) {PORTD.1=1;printf("\nDen vang cot 1 sang + Den do cot 2 sang ");} // vang1
 else PORTD.1=0;
        if(i<3) PORTD.5=1;    else PORTD.5=0;                    // do2
```

```
        if(i>=3) PORTD.2=1; else PORTD.2=0;                    // do1
        if((i>=3)&&(i<4)) {PORTD.4=1;printf("\nDen vang cot 2 sang+ Den do cot 1 sang");}  // vang2
else PORTD.4=0;
        if((i>=4)&&(i<6)) {PORTD.3=1;printf("\nDen xanh cot 2 sang + Den do cot 1 sang");}  // xanh2
  else PORTD.3=0;
        if(i==6)i=0;

}

void main(void)
{        // dung xung ngoai la 4Mhz

 DDRD=0xFF;
// PORTD la cong ra   PORTD.0,PORTD.1,PORTD.2,PORTD.3,PORTD.4,PORTD.5 lan luot la
xanh1,vang1,do1,xanh2,vang2,do2
 PORTD=0x00;
 TCCR1B=0x03;      // Prescaler clk/1024
 TCNT1=26474;      // Dat gia tri dau cua bo dem
 TIMSK=0x04;      // Cho phep ngat khi Timer 1 tran     neu co ngat tran thi TIFR=0x03 tuc TOV1=1

  #asm("sei")
   UCSRA=0x00;           // Cho phep vi dieu khien truyen nhan du lieu
   UCSRB=0x18;
   UCSRC=0x86;
   UBRRH=0x00;
   UBRRL=0x19;
   printf("\nTrang thai den o cac cot :");

 while(1)
   {

   }

 }
```



Nên nhớ khi gửi dữ liệu thì gửi qua chân TXD và nhận dữ liệu bằng chân RXD nên ta phải nối chéo, chân TXD của vi điều khiển với chân RXD của Virtual Terminal và ngược lại .

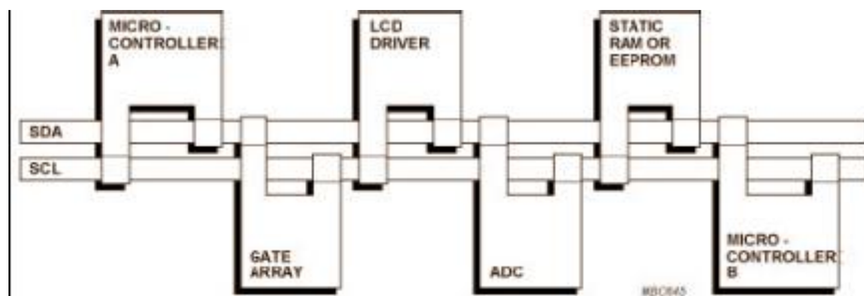
BÀI 9 : GIAO TIẾP I^2C

- *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

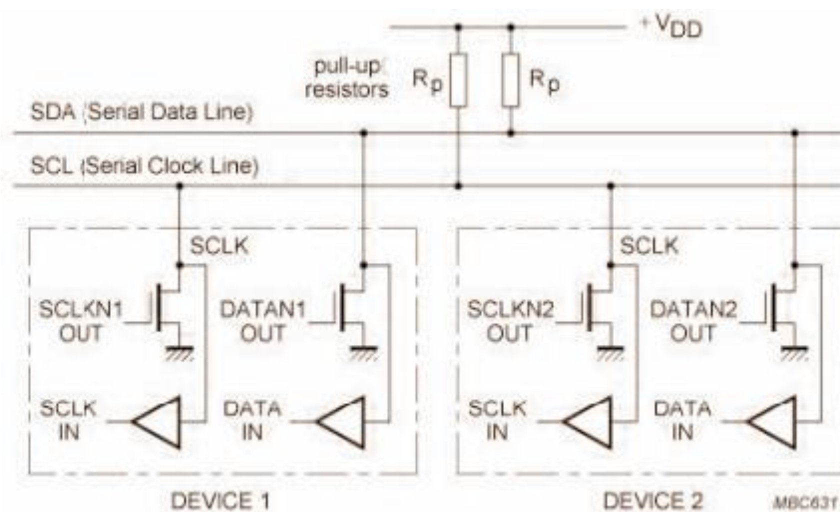
Ngày nay trong các hệ thống điện tử hiện đại, rất nhiều IC hay thiết bị ngoại vi cần phải giao tiếp với các IC hay thiết bị khác – giao tiếp với thế giới bên ngoài. Vói mục tiêu đạt được hiệu quả cho phần cứng tốt nhất với mạch điện đơn giản, Phillips đã phát triển một chuẩn giao tiếp nối tiếp 2 dây 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 bới 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.

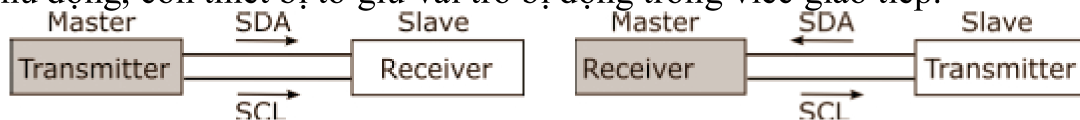


Kết nối thiết bị vào bus I^2C

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 :

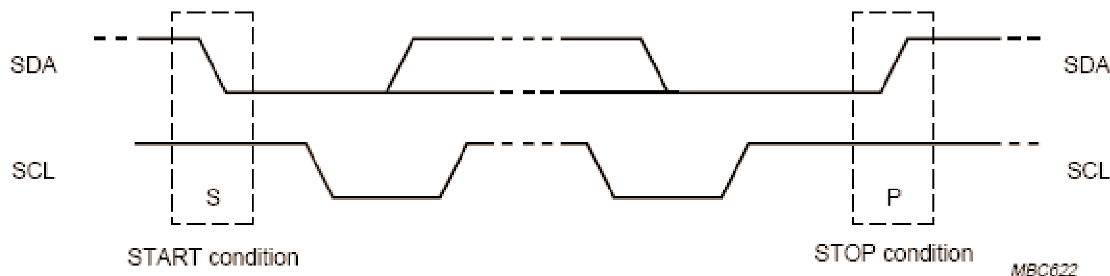
- Thiết bị A (Chủ) xác định đúng địa chỉ của thiết bị B (tớ), cùng với việc xác định địa chỉ, thiết bị A sẽ quyết định việc đọc hay ghi vào thiết bị tớ.
- Thiết bị A gửi dữ liệu tới thiết bị B
- Thiết bị A kết thúc quá trình truyền dữ liệu

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

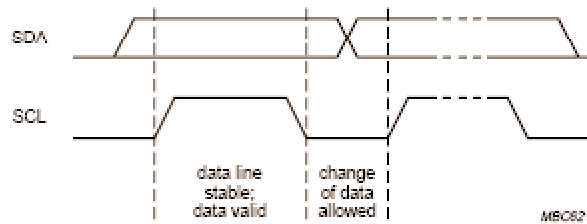
Điều kiện START : một sự chuyển đổi trạng thái từ cao xuống thấp trên đường SDA trong khi đường SCL đang ở mức cao (cao = 1; thấp = 0) báo hiệu một điều kiện START Điều kiện STOP : Một sự chuyển đổi trạng thái từ mức thấp lên cao trên đường SDA trong khi đường SCL đang ở mức cao.

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.

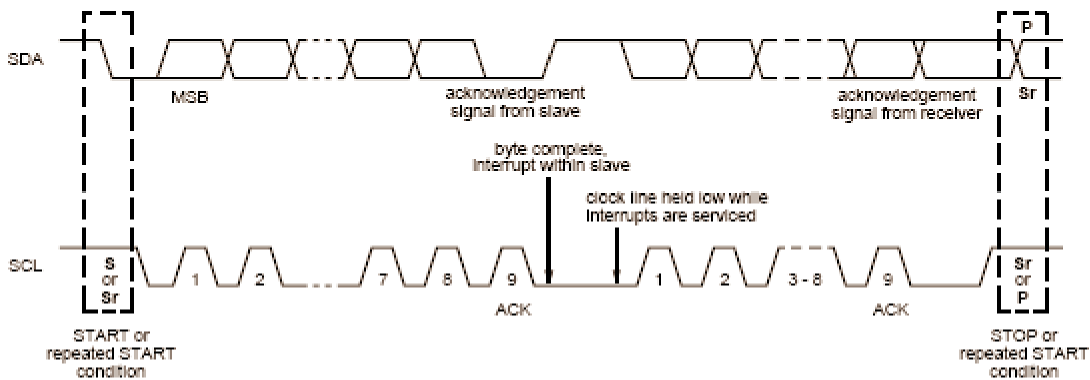
Định dạng dữ liệu truyền

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.

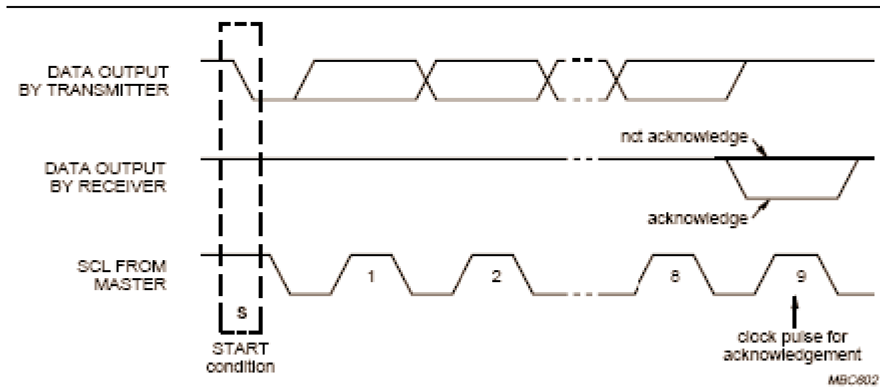


Quá trình truyền 1 bit dữ liệu

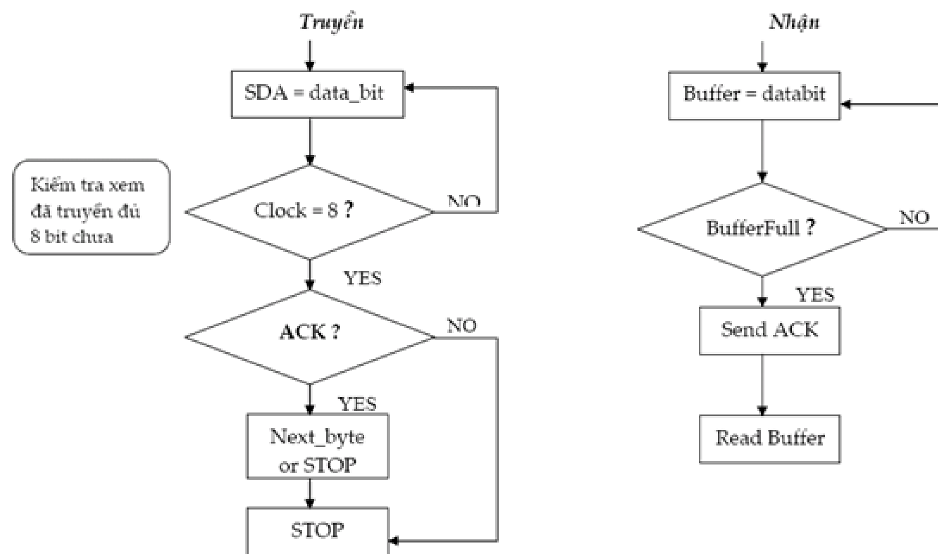
Mỗi byte dữ liệu được truyền có độ dài là 8 bits. Số lượng byte có thể truyền trong một lần là không hạn chế. Mỗi byte được truyền đi theo sau là một bit ACK để báo hiệu đã nhận dữ liệu. Bit có trọng số cao nhất (MSB) sẽ được truyền đi đầu tiên, các bít sẽ được truyền đi lần lượt. Sau 8 xung clock trên 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

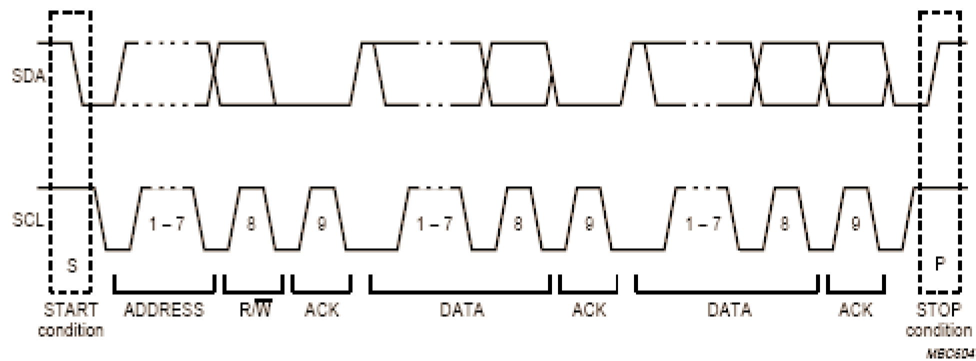
riêng do nhà sản xuất ra nó quy định. Địa chỉ đó có thể là cố định hay thay đổi. Riêng bit điều khiển hướng sẽ quy định chiều truyền dữ liệu. Nếu bit này bằng "0" có nghĩa là byte dữ liệu tiếp theo sau sẽ được truyền từ chủ đến tớ, còn ngược lại nếu bằng "1" thì các byte theo sau byte đầu tiên sẽ là 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

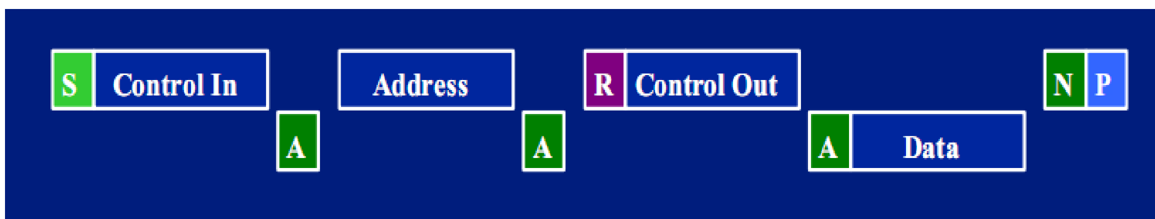
Truyền dữ liệu từ chủ đến tớ (ghi dữ liệu)

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).

Do Ds1307 là slave(IC thời gian thực) nên ta chỉ cần viết hàm truyền và nhận cho Atmega32, mình tạo riêng thư viện (Tần số thạch anh 4MHz)

File I2C_.h

```
#include <mega32.h>
void RTC_set_time(unsigned char,unsigned char,unsigned char);
void RTC_get_time(unsigned char *,unsigned char *,unsigned char *);
void I2C_init();
```

File I2C_.c

```
#include "I2C_.h"
void I2C_init()
{
  TWBR=0x0C;//TWBR=12
  TWSR=0x00;//Prescaler Value=1=>SCL=100KHz>32KHz
}
void I2C_master_transmit(unsigned char slave_address,unsigned char
register_address,unsigned char data)
{
```

```
   unsigned char write=0;
   i2c_retry:;
   TWCR=(1<<TWINT)|(1<<TWSTA)|(0<<TWSTO)|(1<<TWEN);//sent start
   while(!(TWCR&(1<<TWINT)));
   if((TWSR&0x08)!=1) TWSR=0x08;//0x08 kiem tra xem co phai la ack khong
neu khong phai thi sua thanh ack de duoc truyen tiep

   TWDR=slave_address+write;//sent slave_address
   TWCR=(1<<TWINT)|(0<<TWSTA)|(0<<TWSTO)|(1<<TWEN);
   while(!(TWCR&(1<<TWINT)));
   if((TWSR&0x38)==1) goto i2c_retry;
   if((TWSR&0x18)!=1) TWSR=0x18;//0x18 kiem tra xem co phai la ack khong
neu khong phai thi sua thanh ack de duoc truyen tiep

   TWDR=register_address;//sent register_address
   TWCR=(1<<TWINT)|(0<<TWSTA)|(0<<TWSTO)|(1<<TWEN);
   while(!(TWCR&(1<<TWINT)));
   if((TWSR&0x38)==1) goto i2c_retry;
   if((TWSR&0x28)!=1) TWSR=0x28;//0x28 kiem tra xem co phai la ack khong
neu khong phai thi sua thanh ack de duoc truyen tiep

   TWDR=data;//sent data
   TWCR=(1<<TWINT)|(0<<TWSTA)|(0<<TWSTO)|(1<<TWEN);
   while(!(TWCR&(1<<TWINT)));
   if((TWSR&0x38)==1) goto i2c_retry;
   if((TWSR&0x30)!=1) TWSR=0x30;//0x30 kiem tra xem co phai la nack khong
neu khong phai thi sua thanh nack de ngung truyen

   TWCR=(1<<TWINT)|(0<<TWSTA)|(1<<TWSTO)|(1<<TWEN);//stop*/
}
 unsigned char I2C_master_receiver(unsigned char slave_address,unsigned char
register_address)
{
   unsigned char read=1,write=0,data;
   i2c_retry1:;
   TWCR=(1<<TWINT)|(1<<TWSTA)|(0<<TWSTO)|(1<<TWEN);//sent start
   while(!(TWCR&(1<<TWINT)));
   if((TWSR&0x08)!=1) TWSR=0x08; //kiem tra xem co phai la ack khong neu
khong phai thi sua thanh ack de duoc truyen tiep

   TWDR=slave_address+write;//sent slave_address
```

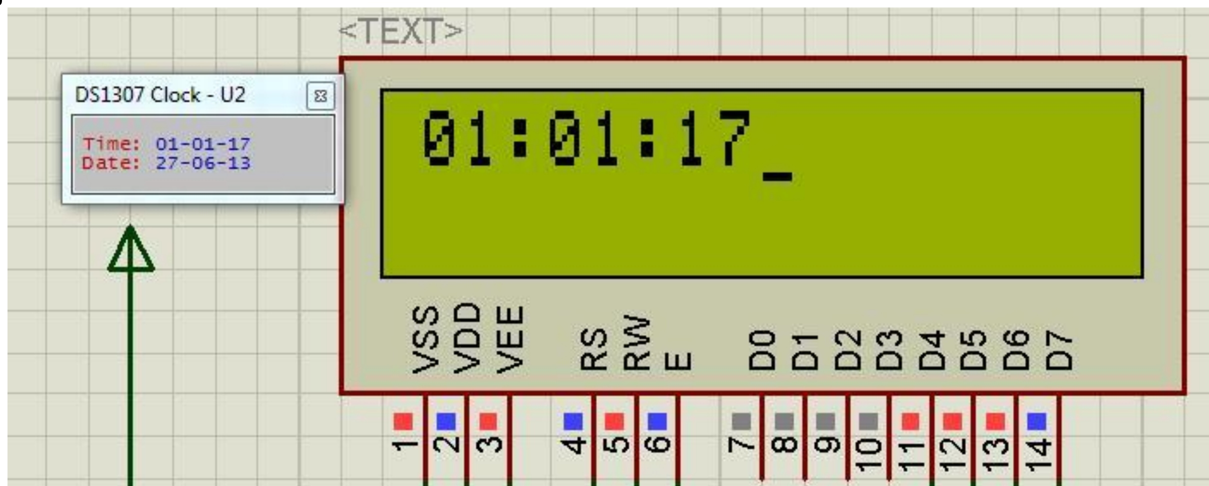
```
   TWCR=(1<<TWINT)|(0<<TWSTA)|(0<<TWSTO)|(1<<TWEN);
   while(!(TWCR&(1<<TWINT)));
   if((TWSR&0x38)==1) goto i2c_retry1;
   if((TWSR&0x18)!=1) TWSR=0x18;//kiem tra xem co phai la ack khong neu
khong phai thi sua thanh ack de duoc truyen tiep

   TWDR=register_address;//sent register_address
   TWCR=(1<<TWINT)|(0<<TWSTA)|(0<<TWSTO)|(1<<TWEN);
   while(!(TWCR&(1<<TWINT)));
   if((TWSR&0x38)==1) goto i2c_retry1;
   if((TWSR&0x28)!=1) TWSR=0x28;//kiem tra xem co phai la ack khong neu
khong phai thi sua thanh ack de duoc truyen tiep

   TWCR=(1<<TWINT)|(0<<TWSTA)|(1<<TWSTO)|(1<<TWEN);//stop

   TWCR=(1<<TWINT)|(1<<TWSTA)|(0<<TWSTO)|(1<<TWEN);//sent start
   while(!(TWCR&(1<<TWINT)));
   if((TWSR&0x38)==1) goto i2c_retry1;
   if((TWSR&0x10)!=1) TWSR=0x10;//kiem tra xem co phai la ack khong neu
khong phai thi sua thanh ack de duoc truyen tiep

   TWDR=slave_address+read;//sent slave_address
   TWCR=(1<<TWINT)|(0<<TWSTA)|(0<<TWSTO)|(1<<TWEN);
   while(!(TWCR&(1<<TWINT)));
   if((TWSR&0x38)==1) goto i2c_retry1;
   if((TWSR&0x40)!=1) TWSR=0x40;//kiem tra xem co phai la ack khong neu
khong phai thi sua thanh ack de duoc truyen tiep hay doc

   TWCR=(1<<TWINT)|(0<<TWSTA)|(0<<TWSTO)|(1<<TWEN);      //day    la
xung nack(TWEA=0) bao khong nhan data nua
   while(!(TWCR&(1<<TWINT)));
   if((TWSR&0x58)!=1) TWSR=0x58;
   data=TWDR;//read data
   TWCR=(1<<TWINT)|(0<<TWSTA)|(1<<TWSTO)|(1<<TWEN);//stop*/
   return data;
}
unsigned char dec_to_2bcd(unsigned char dec)
{
   unsigned char a,b,bcd;
   a=dec/10;
   b=dec%10;
```

```
   bcd=(a<<4)+b;
   return bcd;
}
unsigned char bcd_to_dec(unsigned char bcd)
{
   unsigned char a,b,dec;
   a=(bcd>>4)&0x0F;
   b=bcd&0x0F;
   dec=a*10+b;
   return dec;
}

void RTC_set_time(unsigned char hours,unsigned char minutes,unsigned char
seconds)
{
   I2C_master_transmit(0xD0,0x02,dec_to_2bcd(hours));
   I2C_master_transmit(0xD0,0x01,dec_to_2bcd(minutes));
   I2C_master_transmit(0xD0,0x00,dec_to_2bcd(seconds));
}
void RTC_get_time(unsigned char *hours,unsigned char *minutes,unsigned char
*seconds)
{
   unsigned char hs,ms,ss;
   hs=I2C_master_receiver(0xD0,0x02);
   ms=I2C_master_receiver(0xD0,0x01);
   ss=I2C_master_receiver(0xD0,0x00);
   *hours=bcd_to_dec(hs);
   *minutes=bcd_to_dec(ms);
   *seconds=bcd_to_dec(ss);
}
```

File main.c

```
#include "I2C_.h"
#include "Library.h"

// Declare your global variables here
unsigned char h,m,s;
void main(void)
{
DDRD=0xff;
I2C_init();
```

```
LCD_init();
RTC_set_time(1,1,1);
while(1)
{
RTC_get_time(&h,&m,&s);
writeData(0x30+h/10);
writeData(0x30+h%10);
writeData(':');
writeData(0x30+m/10);
writeData(0x30+m%10);
writeData(':');
writeData(0x30+s/10);
writeData(0x30+s%10);
delay_ms(100);
writeCmd(0x01);
}
}
```



Các bạn có thể tận dụng thư viện có sẵn 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

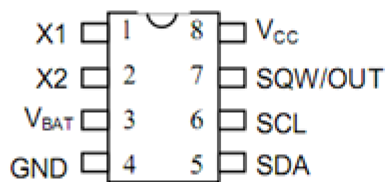
  include <i2c.h>
- ✓ Khởi tạo module I^2C thông qua hàm *i2c_init()*
- ✓ Viết các lệnh cần thiết : read, write, start, stop…

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

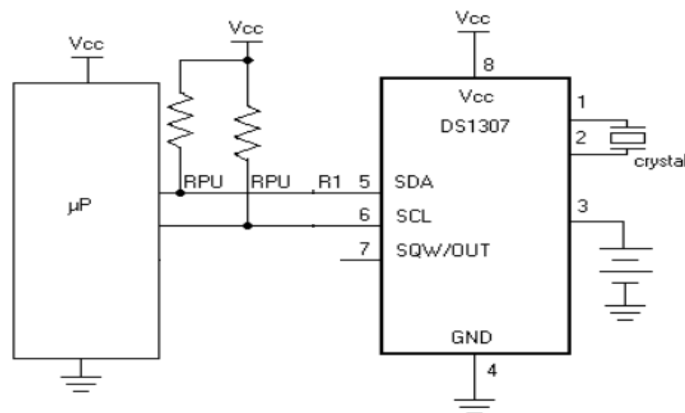
DS1307 là IC thời gian thực, sử dụng giao tiếp I^2C để giao tiếp với các thiết bị khác. Dữ liệu trong DS1307 như giờ, phút, giây… được đặt tại các địa chỉ cố định, được cung cấp bởi nhà sản xuất. Việc đọc hay ghi giờ, phút, giây… chúng ta sẽ đọc/ghi vào các địa chỉ tương ứng.



Sơ đồ chân DS1307

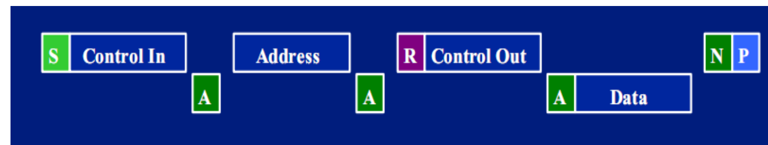
Chi tiết về chức năng và địa chỉ của các dữ liệu trong DS1307, các bạn có thể tham khảo trong datasheet.

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.

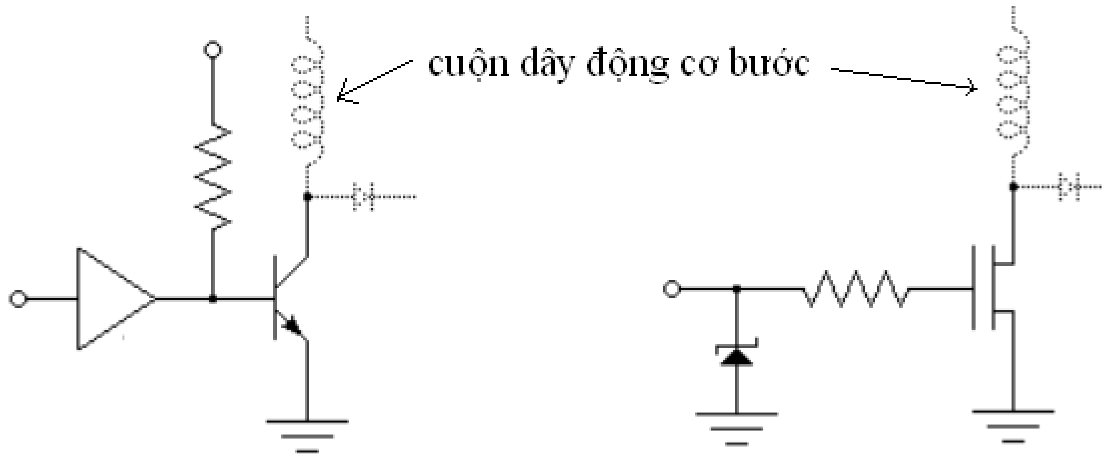
Các đặc điểm chính của động cơ bước :

- **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 nhờ 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.

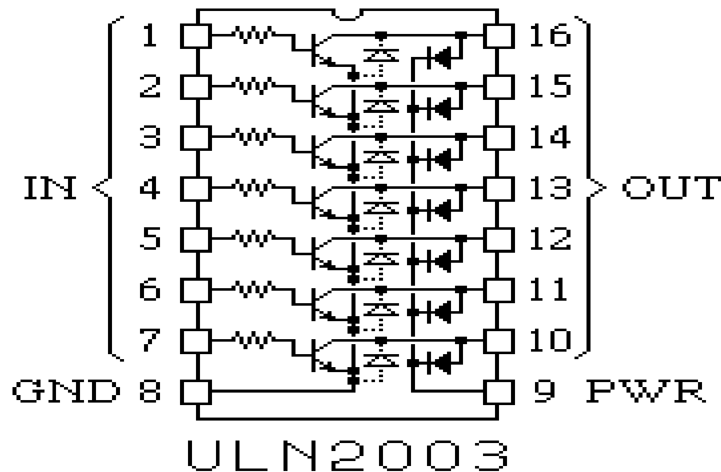
2. Các mạch điều khiển động cơ bước

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…



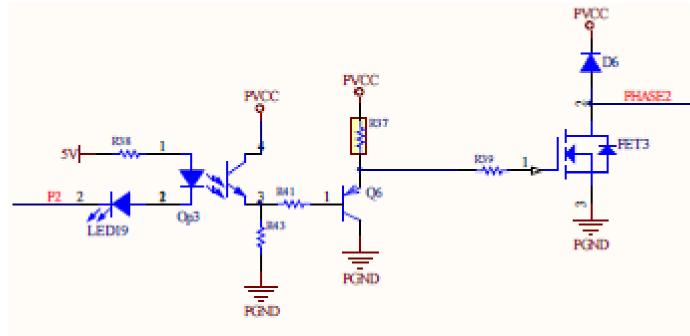
Nếu sử dụng mạch có nguyên lí như trên, chúng ta có thể sử dụng 1 IC tích hợp sẵn như ULN2003, IC họ ULN200x có đầu vào phù hợp TTL, các đầu emitor được nối với chân 8.



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.

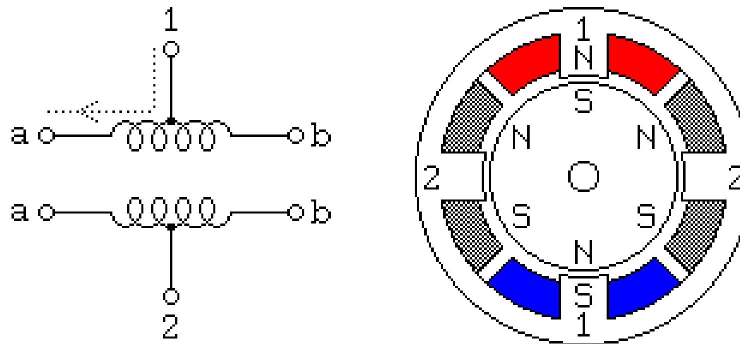
Ngoài ra, có nhiều IC tích hợp sẵn dùng để điều khiển động cơ bước, phổ biến là cặp IC L297 và L298, chuyên dụng để điều khiển động cơ bước với nguyên lí sử dụng mạch cầu H (L298), IC L297 cho phép chúng ta chọn chế độ điều khiển nửa bước hoặc đủ bước.

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.

Trong phạm vi giáo trình chúng ta chỉ xét động cơ bước đơn cực , khi hiểu rõ rõ về động cơ bước đơn cực và cách điều khiển nó thì ta có thể dễ dàng nắm bắt và điều khiển các động cơ bước còn lại.

Như trong hình, dòng điện đi qua từ đầu trung tâm của mấu 1 đến đầu a

tạo ra cực Bắc trong stator trong khi đó cực còn lại của stator là cực Nam. Nếu điện cung cấp liên tục, chúng ta chỉ cần áp điện vào hai mấu của động cơ theo dãy.

Mấu 1a : 1000100010001000100010001 Mấu 1a
Mấu 1b : 0010001000100010001000100 Mấu 1b
Mấu 2a : 0100010001000100010001000 Mấu 2a
Mấu 2b : 0001000100010001000100010 Mấu 2b
Time →
Mấu 1a : 1100110011001100110011001
Mấu 1b : 0011001100110011001100110
Mấu 2a : 0110011001100110011001100
Mấu 2b : 1001100110011001100110011
Time →

Nhớ rằng hai nửa của một mấu không bao giờ được kích cùng một lúc. Cấp tín hiệu như trên thì động cơ bước sẽ được điều khiển theo một bước . Nếu kết hợp hai dãy với nhau cho phép ta điều khiển nửa bước .

Mấu 1a 11000001110000011100000111
Mấu 1b 00011100000111000001110000
Mấu 2a 01110000011100000111000001
Mấu 2b 00000111000001110000011100

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).

Đơn giản ta chỉ việc cấp xung theo dãy xung điều khiển nửa bước thay vì dùng dãy xung điều khiển cả bước, cách cấp xung điều khiển cả bước và nửa bước như thế nào tôi đã nêu ở trên (các dãy logic cấp vào các mấu), ta đã có code ví dụ cộng thêm dãy xung cần cấp và bạn chỉ cần làm một việc đơn giản đó là thay vào code của bài trên là ta đã có bài toán điều khiển theo chế độ nửa bước .

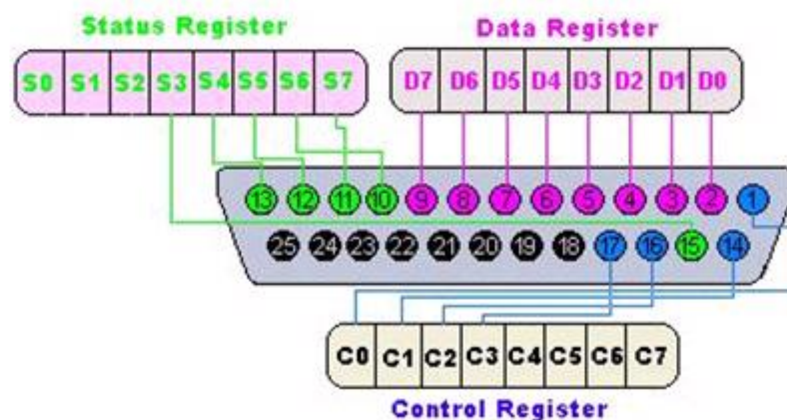
BÀI 11 : GIAO TIẾP VỚI CỔNG LPT

- *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.

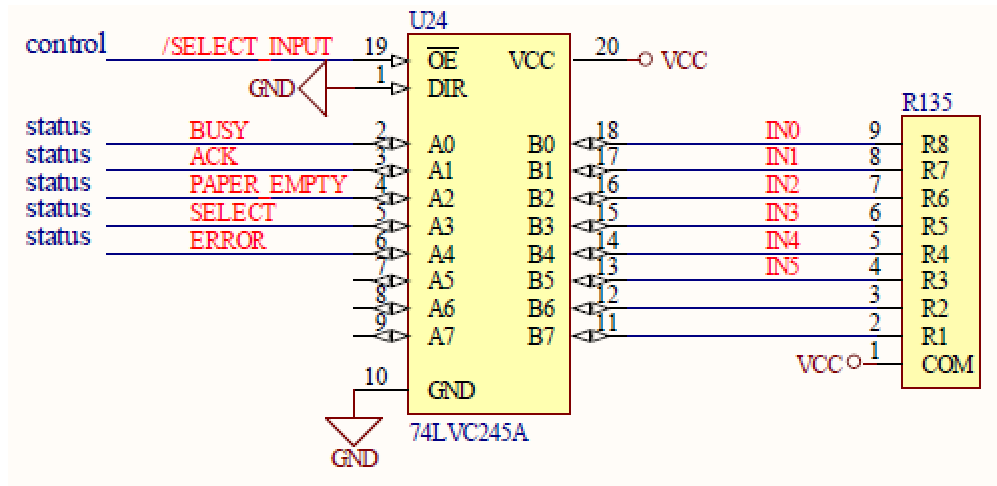
Qua cách mô tả chức năng của từng tín hiệu riêng lẽ ta có thể nhận thấy các đương dẫn dữ liệu có thể chia thành 3 nhóm:

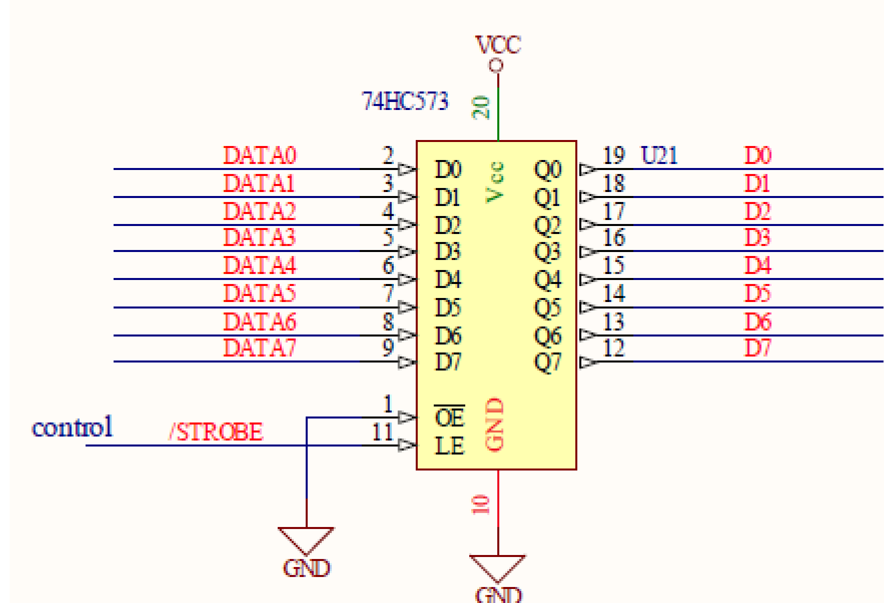
- Các đường dẫn tín hiệu, xuất ra từ máy tính PC và điều khiển máy tính, được gọi là các đường dẫn điều khiển.
- Các đường dẫn tín hiệu, đưa các thông tin thông báo ngược lại từ máy in về máy tính, được gọi là các đường dẫn trạng thái.
- Đường dẫn dữ liệu, truyền các bit rieng lẽ của các ký tự cần in.

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;
    }
}
```

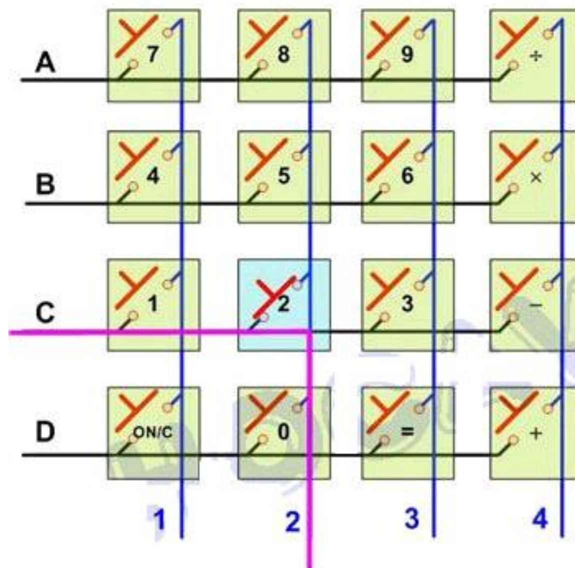
Đoạn mã trên đọc dữ liệu gửi xuống từ cổng LPT (các đường từ D0 đến D7) thong qua Port B, sau đó xuất dữ liệu đó ra Port C. Phần mềm giao tiếp với cổng LPT các bạn có thể tự viết, dùng các ngôn ngữ lập trình như Visual Basic, hay C++, C#...

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.

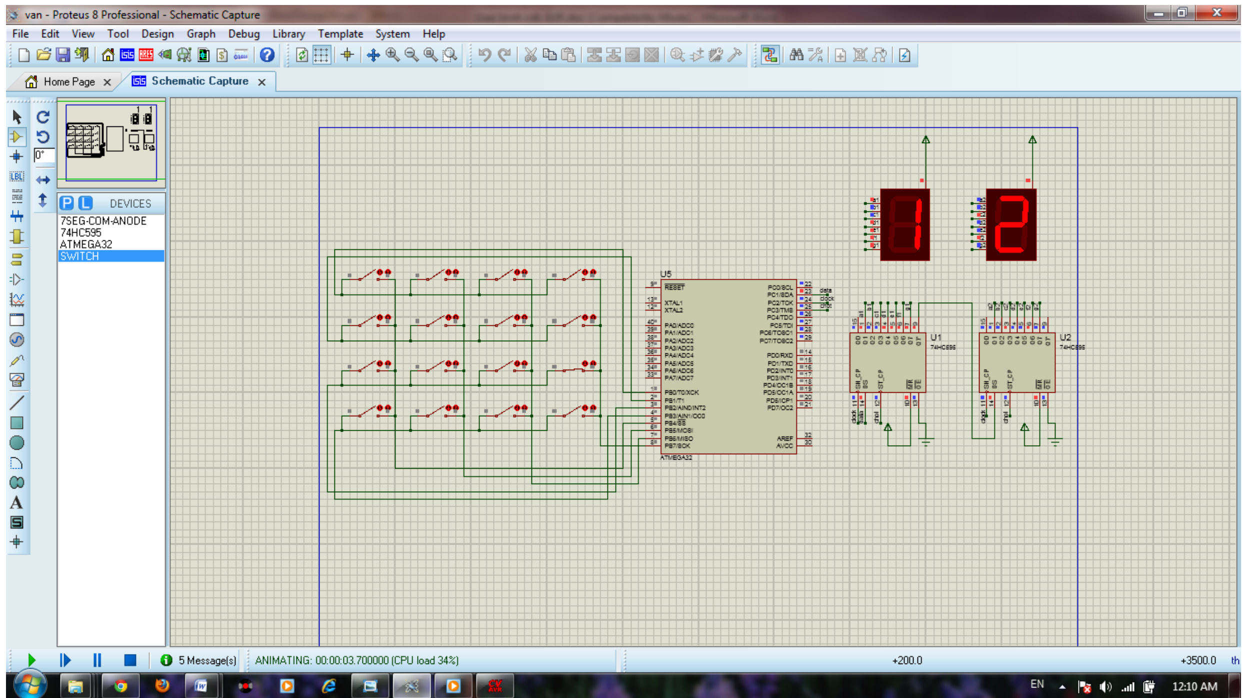
Thuật toán như trên, nhưng cách viết như trên là dài dòng , dưới đây tôi đưa ra cách code ngắn gọn hơn và kết hợp giải bài tập đưa ra :

Trước tiên ta thiết lập công thức thứ tự của các nút bấm khi ta quét đến hàng thứ j và cột thứ k : khi j và k chạy từ 1 đến n (với n là số hàng , số cột) :

$$j*k+(j-1)*(n-k)$$

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

unsigned int j=0,k=0;
unsigned char code[]={0x7E,0x4F,0x12,0x06,0x4C,0x24,0x20,0x0F,0x00,0x04} ;
void main(void)
{
  // dung xung ngoai la 4Mhz
 DDRB=0x0F;            // 4 bit cao cua PORTB la cong vao va 4 bit thap cua PORTB la cong ra de noi voi ma tran phim
 DDRC=0xFF;            // PORTC la cong ra de hien thi LED
 while(1)
  {

    for(j=0;j<4;j++)
    {
      PORTB=PORTB|2^j;
      for(k=4;k<8;k++)
      {
        if(PINB|2^k)
```

```
        {
         unsigned int a;
         a=(j+1)*(k-3)+j*(7-k);
        HienThi(a);
        }
      }
     PORTB=0x00;
   }

 }
```



BÀI 13 : TIMER

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

---

1. Giới thiệu về timer

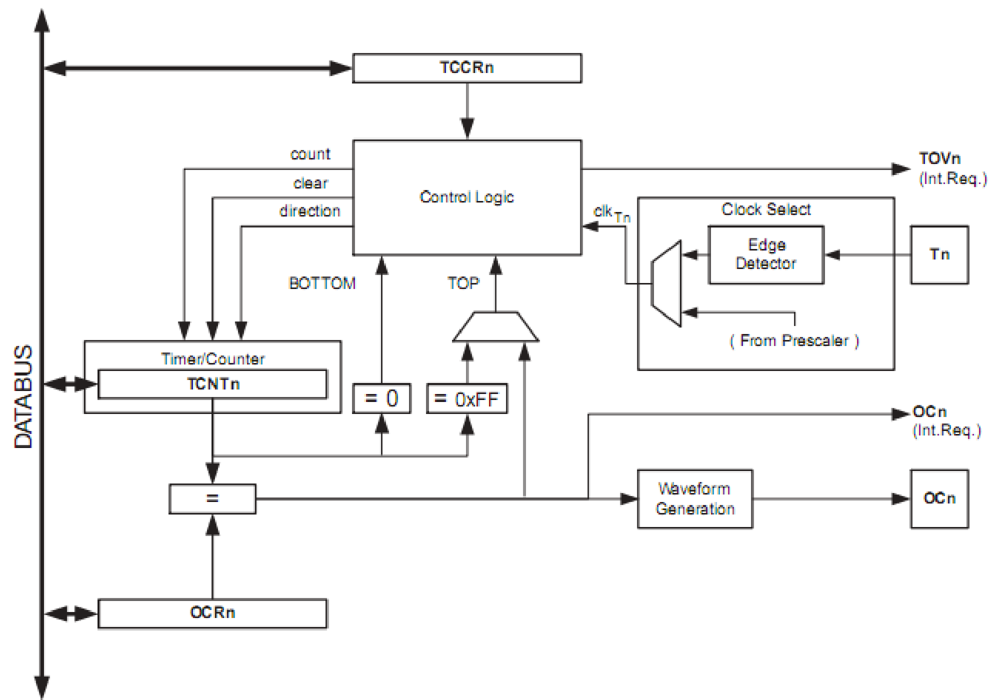
Timer là một trong những module rất quan trọng trong vi điều khiển, sử dụng timer, chúng ta có thể lập trình các tác vụ diễn ra 1 cách chính xác theo thời gian đã định trước, có thể đếm số xung nối vào đầu vào timer…Hầu như tất cả các loại vi điều khiển đều có timer, số lượng timer ở mỗi dòng vi điều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

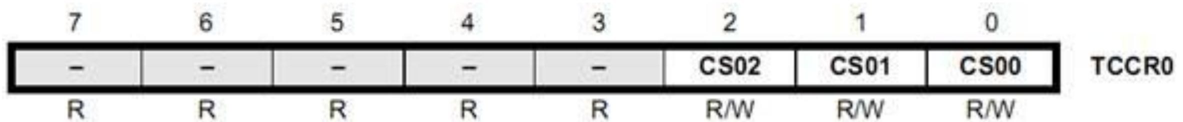
Trong Atmega32 có 3 timer : Timer 0, timer 1 và timer 2

*) **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 từ thanh ghi TCCR0. Tuy là thanh ghi 8 bit nhưng thực chất chỉ có 3 bit có tác dụng đó là CS00, CS01 và CS02.



7	6	5	4	3	2	1	0	
–	–	–	–	–	CS02	CS01	CS00	TCCR0
R	R	R	R	R	R/W	R/W	R/W	

Các bit CS00, CS01 và CS02 gọi là các bit chọn nguồn xung nhịp cho T/C0 (Clock Select). Chức năng các bit này được mô tả trong bảng 1.

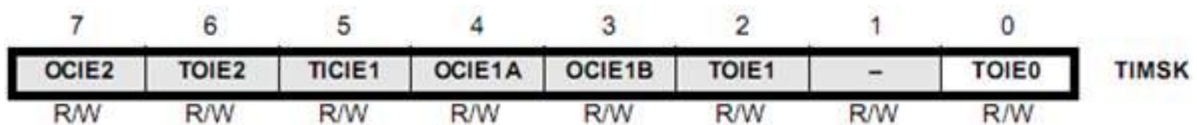
Bảng 1: chức năng các bit CS0X

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

Thanh ghi TCNT0 dùng để chứa giá trị vận hành 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.

Thanh ghi TIMSK: là thanh ghi mặt nạ cho ngắt của tất cả các T/C trong Atmega8, trong đó chỉ có bit TOIE0 tức bit số 0 (bit đầu tiên) trong thanh ghi này là liên quan đến T/C0, bit này có tên là bit cho phép ngắt khi có tràn ở T/C0. Tràn (Overflow) là hiện tượng xảy ra khi bộ giá trị trong thanh ghi TCNT0 đã đạt đến MAX (255) và lại đếm thêm 1 lần nữa.



Khi bit TOIE0=1, và bit I trong thanh ghi trạng thái được set (xem lại bài 3 về điều khiển ngắt), nếu một “tràn” xảy ra sẽ dẫn đến ngắt tràn.

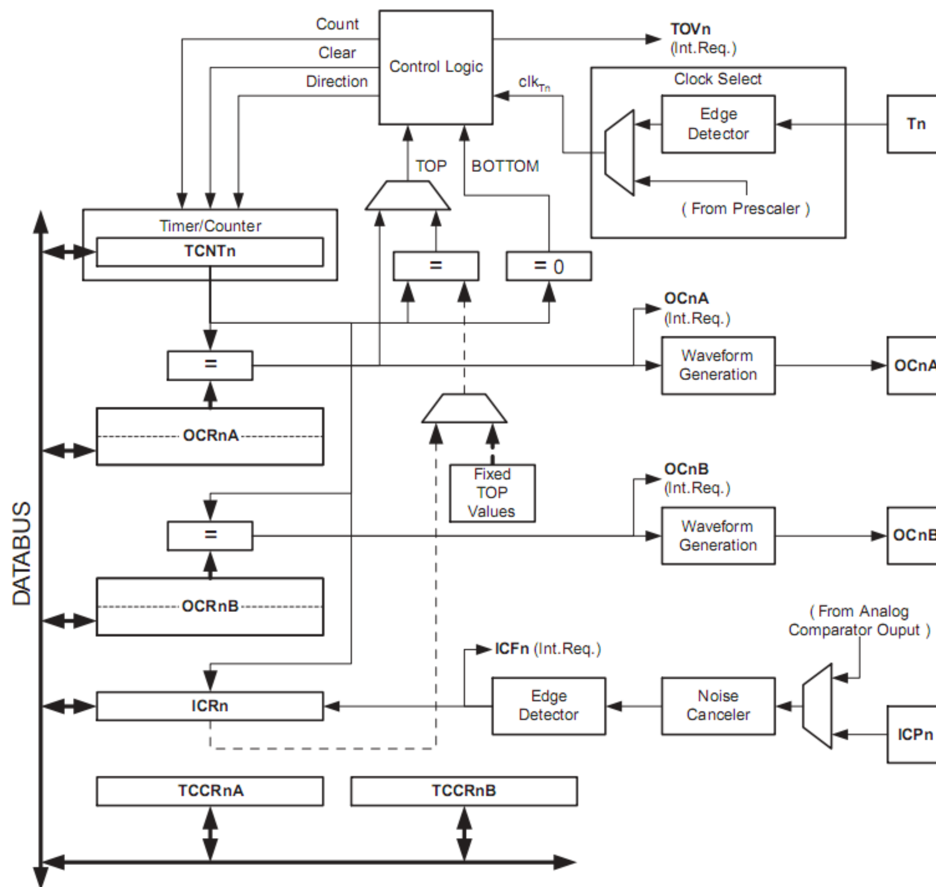
Thanh ghi TIFR là thanh ghi cờ nhớ cho tất cả các bộ T/C. Trong thanh ghi này bit số 0, TOV0 là cờ chỉ thị ngắt tràn của T/C0. Khi có ngắt tràn xảy ra, bit này tự động được set lên 1. Thông thường trong điều khiển các T/C vai trò của thanh ghi TIFR không quá quan trọng.

Hoạt động : T/C0 hoạt động rất đơn giản, hoạt động của T/C được “kích” bởi một tín hiệu (signal), cứ mỗi lần xuất hiện tín hiệu “kích” giá trị của thanh ghi TCNT0 lại tăng thêm 1 đơn vị, thanh ghi này tăng cho đến khi nó đạt mức MAX là 255, tín hiệu kích tiếp theo sẽ làm thanh ghi TCNT0 trở về 0 (tràn), lúc này bit cờ tràn TOV0 sẽ tự động được set bằng 1. Với cách thức hoạt động như thế có vẻ T/C0 vô dụng vì cứ tăng từ 0 đến 255 rồi lại quay về 0, và quá trình lặp lại. Tuy nhiên, yếu tố tạo sự khác biệt chính là tín hiệu kích và ngắt tràn, kết hợp 2 yếu tố này chúng ta có thể tạo ra 1 bộ định thời gian hoặc 1 bộ đếm sự kiện. Trước hết bạn hãy nhìn lại bảng 1 về các bit chọn xung nhịp cho T/C0. Xung nhịp cho T/C0 chính là tín hiệu kích cho T/C0. Xung nhịp này có thể tạo bằng nguồn tạo dao động của chip (thạch anh, dao động nội trong chip…). Bằng cách đặt giá trị cho các bit CS00, CS01 và CS02 của thanh ghi điều khiển TCCR0, chúng ta sẽ quyết định bao lâu thì sẽ kích T/C0 một lần. Ví dụ mạch ứng dụng của bạn có nguồn dao động clk = 1MHz tức chu kỳ 1 nhịp là 1us (1 micro giây), bạn đặt thanh ghi TCCR0=5 (tức SC02=1, CS01=0, CS00=1). Căn cứ theo bảng 1, tín hiệu kích cho T/C0 sẽ bằng clk/1024 nghĩa là sau 1024us thì T/C0 mới được kích 1 lần, nói cách khác giá trị của TCNT0 tăng thêm 1 sau 1024us (chú ý là tần số được chia cho 1024 thì chu kỳ sẽ tăng 1024 lần). Quan sát 2 dòng cuối cùng trong bảng 1 bạn sẽ thấy rằng tín hiệu kích cho T/C0 có thể lấy từ bên ngoài (External clock

source), đây chính là ý tưởng cho hoạt động của chức năng đếm sự kiện trên T/C0. Bằng cách thay đổi trạng thái chân T0 (chân 6 trên chip Atmega8) chúng ta sẽ làm tăng giá trị thanh ghi TCNT0 hay nói cách khác T/C0 có thể dùng để đếm sự kiện xảy ra trên chân T0. Dưới đây chúng ta sẽ xem xét cụ thể cách điều khiển T/C0 theo 1 chế độ định thời gian và đếm.

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

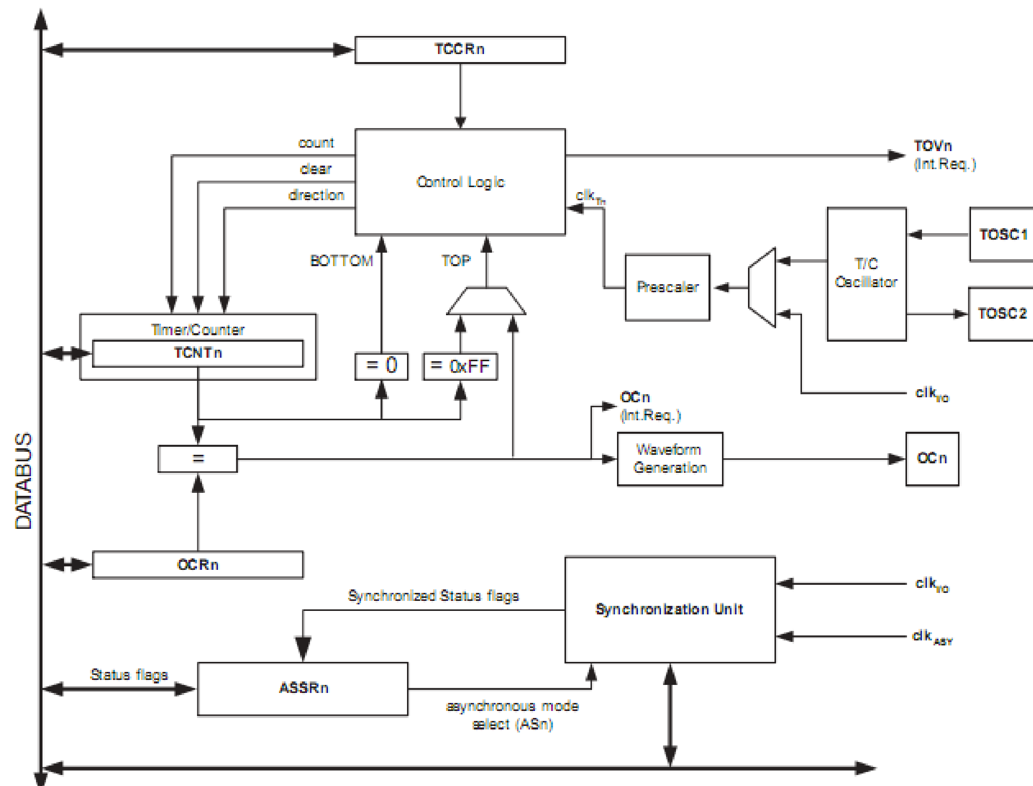
- Có 2 module Output Compare hoạt động độc lập
- Hai thanh ghi đệm chứa giá trị đặt trước
- Một module Capture
- Chế độ tự động nạp lại
- Chế độ PWM với chu kì có thể thay đổi được
- Bộ tạo tần số
- Bộ đếm sự kiện (Counter)
- Có 4 nguồn ngắt độc lập



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. Ở đây có sự khác biệt đó là Timer1 là thanh ghi 16 bit nên nó có một số khác biệt sau :

- TCNT1 được tạo nên từ hai thanh ghi 8 bit đó là TCNT1H và TCNT1L .

Bit	7	6	5	4	3	2	1	0	
	TCNT1[15:8]								TCNT1H
	TCNT1[7:0]								TCNT1L
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	

- TCCR1A và TCCR1B : điều khiển hoạt động của T/C1

Bit	7	6	5	4	3	2	1	0	
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	W	W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Nhìn chung để "thuộc" hết cách phối hợp các bit trong 2 thanh ghi TCCR1A và TCCR1B là tương đối phức tạp vì T/C1 có rất nhiều mode hoạt động, chúng ta sẽ khảo sát chúng trong phần các chế độ hoạt động của T/C1 bên dưới. Ở đây, trong thanh ghi TCCR1B có 3 bit khá quen thuộc là CS10, CS11 và CS12. Đây là các bit chọn xung nhịp cho T/C1 như truong T/C0. Bảng 2 sẽ tóm tắt các chế độ xung nhịp trong T/C1.

Bảng 2: chức năng các bit CS12, CS11 và CS10.

CS12	CS11	CS10	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

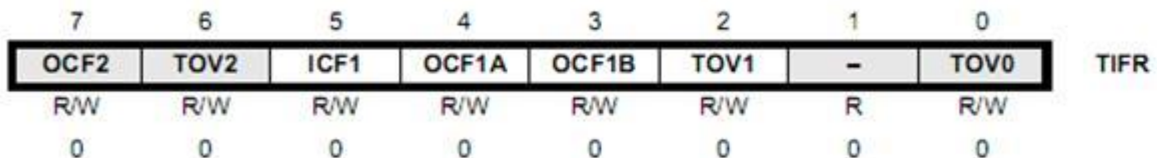
- OCR1A và OCR1B :

7	6	5	4	3	2	1	0	
OCR1A[15:8]								OCR1AH
OCR1A[7:0]								OCR1AL
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

7	6	5	4	3	2	1	0	
OCR1B[15:8]								OCR1BH
OCR1B[7:0]								OCR1BL
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

- ICR1 : là khái niệm mới của T/C1 là Input Capture. Khi có 1 sự kiện trên chân ICP1 (chân 14 trên Atmega8), thanh ghi ICR1sẽ "capture" giá trị của thanh ghi đếm TCNT1. Một ngắt có thể xảy ra trong trường hợp này, vì thế Input Capture có thể được dùng để cập nhật giá trị "TOP" của T/C1.

- TIFR là thanh ghi cờ nhớ cho tất cả các bộ T/C. Các bit từ 2 đến 5 trong thanh ghi này là các cờ trạng thái của T/C1.



Các mode hoạt động: có tất cả 5 chế độ hoạt động chính trên T/C1. Các chế độ hoạt động cơ bản được quy định bởi 4 bit Waveform Generation Mode (WGM13, WGM12, WGM11 WGM10) và một số bit phụ khác. 4 bit Waveform Generation Mode lại được bố trí nằm trong 2 thanh ghi TCCR1A và TCCR1B (WGM13 là bit 4, WGM12 là bit 3 trong TCCR1B trong khi WGM11 là bit 1 và WGM10 là bit 0 trong thanh ghi TCCR1A) vì thế cần phối hợp 2 thanh ghi TCCR1 trong lúc điều khiển T/C1. Các chế độ hoạt động của T/C1 được tóm tắt trong bảng sau 3:

Bảng 3: các bit WGM và các chế độ hoạt động của T/C1.

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation[1]	TOP	Update of OCR1x	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

Các Timer/Counter được sử dụng phổ biến với chế độ thường(định thời), chế độ counter,tạo xung PWM, ngoài ra nó còn một số chế độ khác nhưng do không được dùng phổ biến nên tôi không nói trong giáo trình này.

+) Chế độ thường (bộ định thời): Định một khoảng thời gian mà ta mong muốn khi cho giá trị của thanh ghi TCNTn tăng từ một giá trị khởi điểm nào đó đến giá trị tràn bộ timer (giá trị đặt trước phụ thuộc vào thời gian ta yêu cầu) .

+) Chế độ Counter : để đếm các sự kiện xảy ra ở chân Tn , mỗi khi có một sự kiện xảy ra ở chân Tn thì giá trị trong thanh ghi TCNTn tăng lên 1 đơn vị cho đến khi tràn. Có thể đếm theo sườn lên hoặc sườn xuống tùy vào việc bạn set thanh ghi TCCRn.

+) Tạo xung điều rộng PWM : Tạo ra xung có độ rộng thay đổi tùy ý, rất có ý nghĩa trong điều khiển động cơ. Tùy vào việc bạn set giá trị cho thanh ghi OCR mà bạn sẽ có độ rộng xung PWM như mong muốn. Trong chế độ Fast PWM, 1 chu kỳ được tính trong 1 lần đếm từ BOTTOM lên TOP (single-slope), vì thế mà chế độ này gọi là Fast PWM (PWM nhanh). Có tất cả 5 mode trong Fast PWM tương ứng với 5 cách chọn giá trị TOP khác nhau (tham khảo bảng 3). Việc xác lập chế độ hoạt động cho Fast PWM thực hiện thông qua 4 bit WGM và các bit chọn dạng xung ngõ ra, Compare Output Mode trong thanh ghi TCCR1A, nhìn lại 2 thanh ghi TCCR1A và TCCR1B.

7	6	5	4	3	2	1	0	
COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A

7	6	5	4	3	2	1	0	
ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B

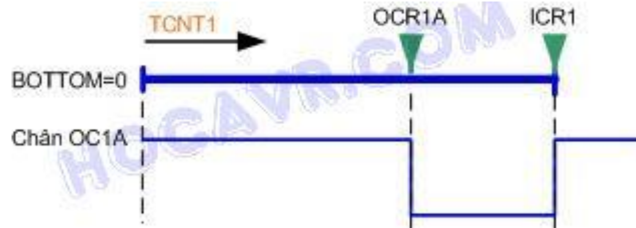
Chú ý các bit COM1A1, COM1A0 và COM1B1, COM1B0 là các bit chọn dạng tín hiệu ra của PWM (Compare Output Mode bits). COM1A1, COM1A0 dùng cho kênh A và COM1B1, COM1B0 dùng cho kênh B. Hãy đối chiếu bảng 4.

Bảng 4: mô tả các bit COM trong chế độ fast PWM.

COM1A1/ COM1B1	COM1A0/ COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	WGM13:0 = 15: Toggle OC1A on Compare Match, OC1B disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B disconnected.
1	0	Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at TOP
1	1	Set OC1A/OC1B on Compare Match, clear OC1A/OC1B at TOP

Tôi sẽ giải thích hoạt động của Fast PWM kênh A thông qua 1 trường hợp cụ thể, mode 14 (WGM13=1, WGM12=1, WGM11=1, WGM10=0). Trong mode 14, giá trị TOP (cũng là chu kỳ của PWM) được chứa trong thanh ghi ICR1, khi hoạt

động thanh ghi TCNT1 tăng giá trị từ 0, giả sử các bit phụ COM1A=1, COM1A0=0, lúc này trạng thái của chân OC1A (chân 15) là HIGH (5V), khi TCNT1 tăng đến bằng giá trị của thanh ghi OCR1A thì chân OC1A được xóa về mức LOW (0V), thanh ghi đếm TCNT1 vẫn tiếp tục tăng đến khi nào nó bằng giá trị TOP chứa trong thanh ghi ICR1 thì TCNT1 tự động reset về 0 và chân OC1A trở về trạng thái HIGH, cái này gọi là "Clear OC1A/OC1B on Compare Match, set OC1A/OC1B at TOP" mà bạn thấy trong hàng 4 bảng 4. Hình 10 mô tả cách tạo xung PWM trên chân OC1A ở mode 14.



Hình 10: Fast PMW mode 14.

Rõ ràng chúng ta có thể điều khiển cả time period và duty cycle của PWM bằng 2 thanh ghi ICR1 và OCR1A. Thông thường giá trị của ICR1 được tính toán và gán cố định, giá trị của OCR1A được thay đổi để thực hiện mục đích điều khiển (như thay đổi vận tốc động cơ). Chú ý là nếu chúng ta set các bit phụ ngược lại: COM1A=0, COM1A0=1, thì tín hiệu PWM trên chân OC1A sẽ có phần "LOW" từ 0 đến OCR1A và "HIGH" từ OCR1A đến ICR1, đây gọi là "set OC1A/OC1B on Compare Match, clear OC1A/OC1B at TOP" (ngược với tín hiệu trên hình 10). Hoạt động của fast PWM kênh B hoàn toàn tương tự, trong đó thanh ghi ICR1 cũng chứa TOP của PWM kênh B và thanh ghi ICR1B chứa duty cycle. Như vậy 2 kênh A và B có cùng tần số hay Time period và duty cycle được điều khiển độc lập. Chân xuất tín hiệu PWM của kênh B là chân OC1B (chân 16 trên Atmega8).

Các mode 5, 6 và 7 của Fast PWM hoạt động hoàn toàn tương tự mode 14. Điểm khác nhau cơ bản là giá trị TOP(Time period). Trong các mode này giá trị TOP không do thanh thi ICR1 định nghĩa mà là các hằng số không đổi. Với mode 5, tức mode 8 bits, (WGM13=0, WGM12=1, WGM11=0, WGM10=1) giá trị TOP là 1 hằng số, TOP = 255 (số 8 bits lớn nhất). Với mode 6, tức mode 9 bits, (WGM13=0, WGM12=1, WGM11=1, WGM10=0) giá trị TOP là 1 hằng số, TOP = 511 (số 9 bits lớn nhất). Và với mode 7, tức mode 10 bits, (WGM13=0, WGM12=1, WGM11=1, WGM10=1) TOP =1023 (số 10 bits lớn nhất). Mode 15 cũng là Fast PWM trong đó TOP do OCR1A quy định, vì thế mà tín hiệu ra ở kênh A hầu như không phải là 1 xung, nó chỉ thay đổi trạng thái trong 1 clock. Theo tôi, để sử dụng Fast PWM bạn nên dùng mode 14 đã được giải thích trên. Các mode 5, 6, 7 cũng có thể dùng nhưng không nên dùng mode 15.

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

Bài 1 : Bạn hãy tính toán xem với cấu hình timer như trên thì sau bao lâu PORT A đảo giá trị một lần.

Bài 2 : Viết chương trình trễ n(ms) sử dụng timer.

Lời giải :

Bài 1: Như trên bạn thấy, cứ sau một lần Timer1 tràn thì PORTA đảo giá trị một lần; giá trị đặt trước của Timer1 là 0x00; và Prescale =64, vậy cứ sau một khoảng thời gian t= $(64.2^{16})/(4.10^{6}) = 1.048576$ (s).

Lưu ý ở trên ta lấy tần số của chip 4.10^6(Hz).

Bài 2 : Để viết chương trình trễ n(ms)có rất nhiều cách và cách viết tổng quát nhất là dùng ngắt timer (có thể trễ với thời gian bất kì),nhưng với giới hạn ms thì chỉ cần dùng trong phạm vi thời gian tràn của Timer1 thì bạn cũng đã có thể dùng thoải mái :

```
#include <mega32.h>
void main(void)
{
 TCCR1=0x05;   Prescale=1024
 TCNT1= 1- (4.10^6/1024).n.10^-3 ;// với n là khoảng thời gian cần trễ n(ms)
while(TCNT1<65500)
   {
   }

  /* if(TCNT1>65500)       // neu tiep tuc dung Timer 1 thi ta dung doan nay, neu khong thi bo qua
   {
    TCNT1= 1- (4.10^6/1024).n.10^-3 ;
   } */

}
```

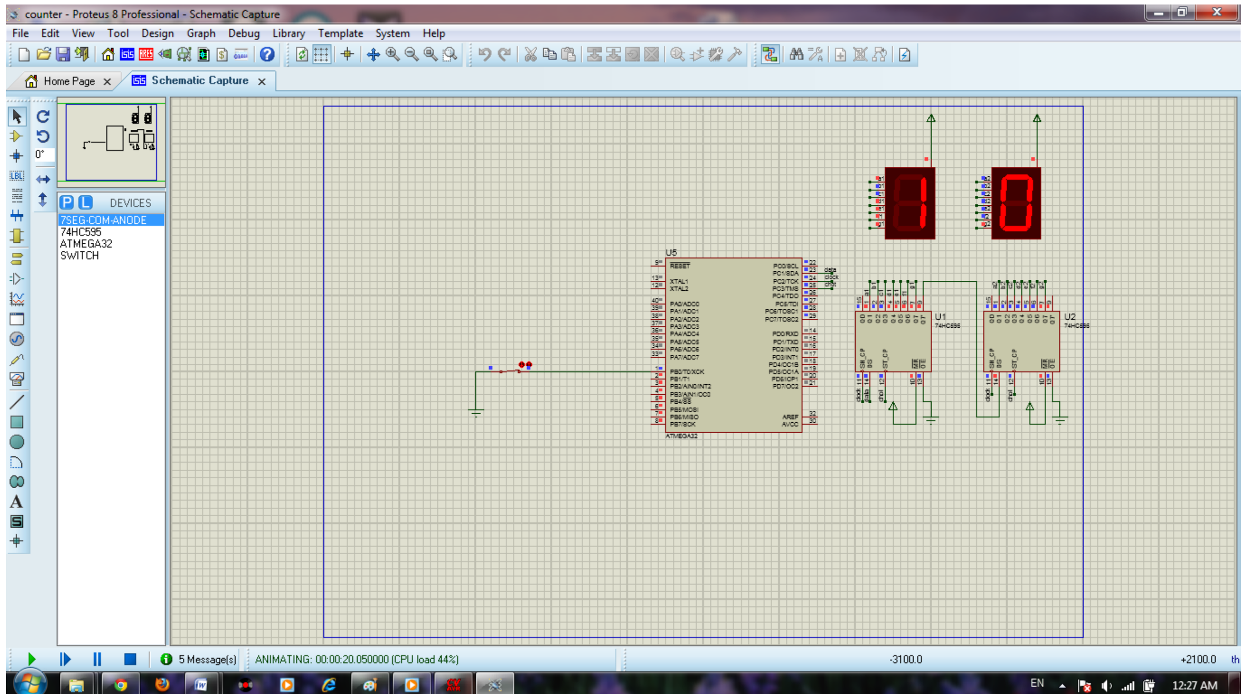
Bài 3 : Bạn hãy dùng ngắt Timer để định thời n(s).

Vừa rồi là ví dụ về chế độ dịnh thời của Timer, sau đây chúng ta sẽ làm một ví dụ để hiểu rõ hơn về chế độ Counter (Từ ví dụ này bạn có thể ứng dụng để đếm tốc độ động cơ , tất nhiên bạn phải tìm hiểu một chút về động cơ và encoder để có thể tính chính xác tốc độ của nó). Về phần tạo xung PWM theo mình thì bạn nên tìm hiểu sau khi đến với bài điều khiển động cơ một chiều, và để hiểu rõ hơn bạn có thể tham khảo cách tạo xung PWM trong phần bài tập mở rộng ở cuối giáo trình.

Mô tả ví dụ về Counter : Cứ mỗi lần phím được nhấn thì giá trị counter tăng lên 1 đơn vị và hiển thị ra LED 7 segment, khi giá trị counter quá 20 thì reset bộ đếm (đây chỉ là một ví dụ tôi nghĩ ra, bạn có thể làm vô vàn ví dụ với counter):

```
#include <mega32.h>

unsigned char code[]={0x01,0x4F,0x12,0x06,0x4C,0x24,0x20,0x0F,0x00,0x04} ;
void HienThi (unsigned int n)     // n bat ki, trong vi du nay ta su dung n la mot so co 4 chu so
{
unsigned int a[4];
unsigned char i,j;
```

```
a[1]=((n%1000)%100)/10;a[0]=((n%1000)%100)%10;
    for (i=0;i<2;i++)
        for (j=0;j<8;j++)
                {
                    PORTC.1=(code[a[i]]>>j)&(0x01);
                    PORTC.2=1;
                    PORTC.2=0;
                }
 PORTC.3=1;
 PORTC.3=0;
}

void main(void)
{
  DDRC=0xFF;  // PORTC la cong ra
  DDRB=0x00; // PORTB la cong vao
  PORTB=0xFF;
  TCCR0=0x06; // dem theo suon xuong
  TCNT0=0;

while (1)
      {
        if(TCNT0>20) TCNT0=0;
        HienThi(TCNT0);
      };
}
```



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

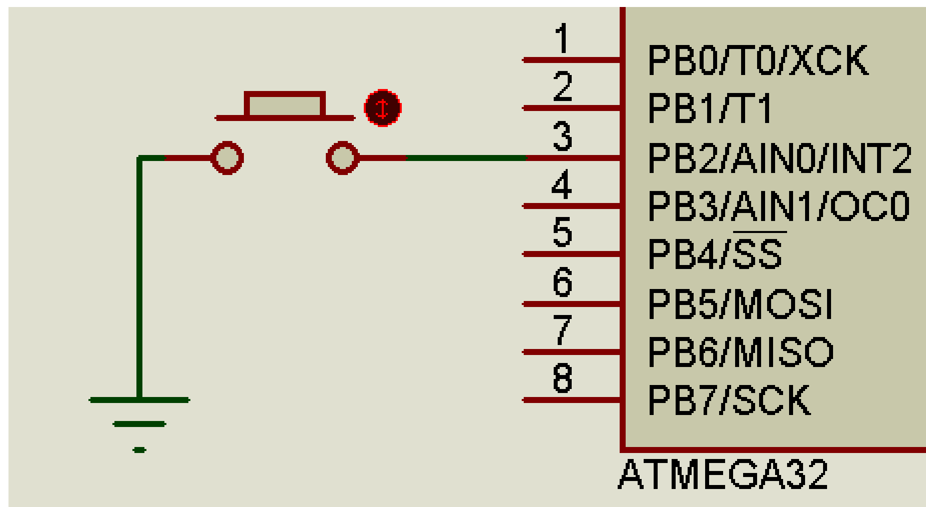
- Ngắt ngoài :
    - Đặt chế độ cho ngắt : ngắt theo sườn lên (xuống), hay ngắt theo mức.
    - Cho phép ngắt toàn cục.
    - Viết chương trình cho ngắt ngoài

- 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.



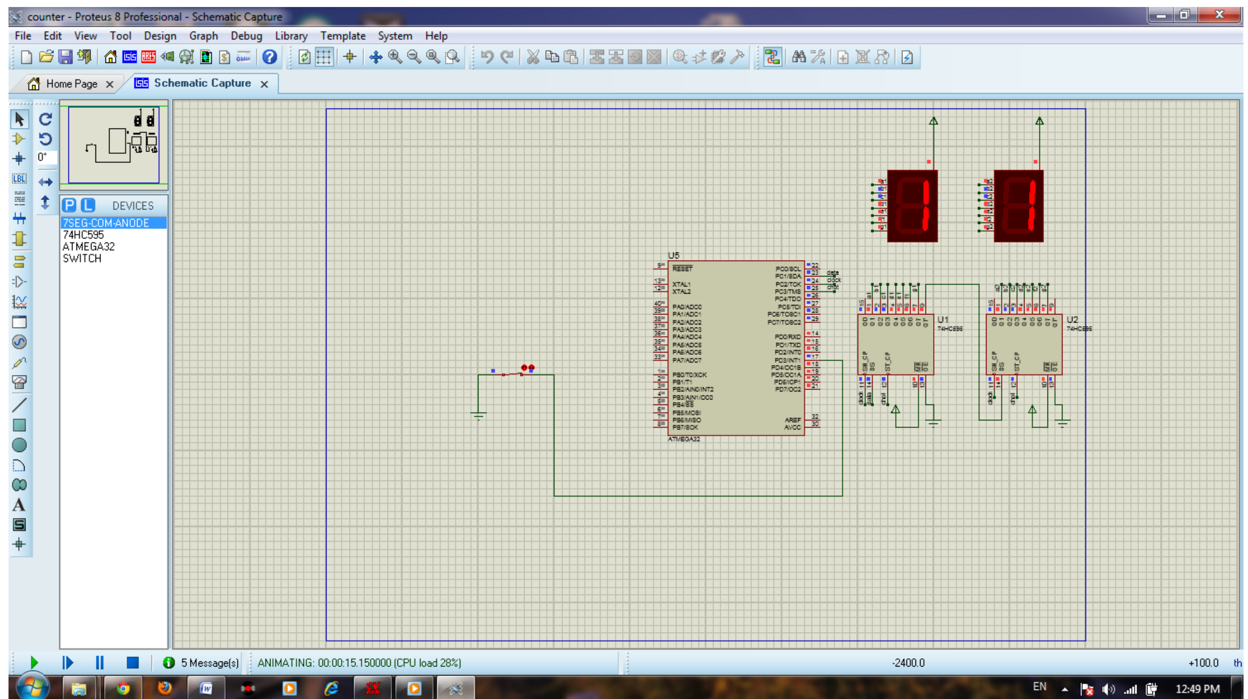
Chương trình

```
#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.

Ở trên bạn đã có ví dụ về ngắt ngoài và ngay bây giờ tôi sẽ kết hợp ngắt ngoài và ngắt timer để bạn có thể hiểu một cách tổng quát và phân biệt rõ hơn về hai loại ngắt này : Ta dùng ví dụ của phần counter ở trên nhưng thay vì dùng counter thì ta sẽ dùng ngắt ngoài để đếm số lần ấn phím và sau 30s thì reset lại kết quả :




```
#include <mega32.h>

unsigned int b,l;


unsigned char code[]={0x01,0x4F,0x12,0x06,0x4C,0x24,0x20,0x0F,0x00,0x04} ;
void HienThi (unsigned int n)
{
unsigned int a[2];
unsigned char i,j;
a[1]=((n%1000)%100)/10;a[0]=((n%1000)%100)%10;
   for (i=0;i<2;i++)
      for (j=0;j<8;j++)
            {
               PORTC.1=(code[a[i]]>>j)&(0x01);
               PORTC.2=1;
               PORTC.2=0;
```

```
                }
 PORTC.3=1;
 PORTC.3=0;
}
interrupt [EXT_INT1] void ext_int1_isr(void)
{
 b=b+1;
 HienThi(b);
}
 interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
      l=l+1;
      if(l==3)
      {
       b=0;
       TCNT1=26474;
       TIFR=0x00;
       HienThi(b);
       }
}


void main(void)
{
  DDRC=0xFF;  // PORTC la cong ra
  DDRD=0x00; // PORTD la cong vao
  PORTD=0xFF;
  TCCR1B=0x03;            // Prescaler clk/1024
  TCNT1=26474;            // Dat gia tri dau cua bo dem
  TIMSK=0x04;
  MCUCR=0x08;  // Ngat theo suon xuong o chan INT1
  GICR=0x80;   // Cho phep ngat INT1
  #asm("sei")


while (1)
      {
      };
}
```

Bài 15: ĐIỀU KHIỂN ĐỘNG CƠ MỘT CHIỀU

- Giới thiệu điều khiển động cơ một chiều

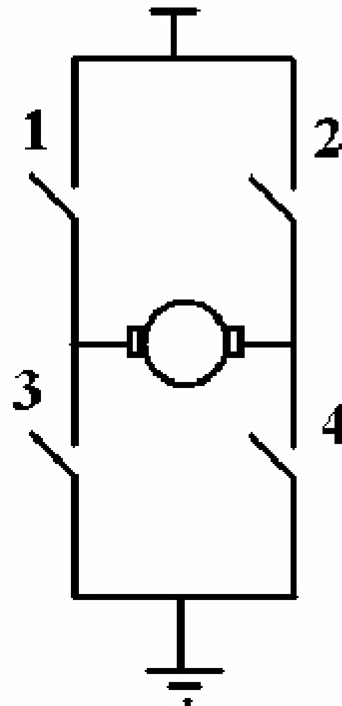
-Ví dụ minh họa

1. Giới thiệu điều khiển 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 hiển thông qua điện áp cấp vào 2 đầu động cơ (nói một cách tổng quát đó là sự thay đổi năng lượng cung cấp cho động cơ làm việc) . Động cơ một chiều được ứng dụng rộng rãi trong công nghiệp ; có rất nhiều cách điều khiển động cơ , nhưng cách phổ biến nhất hiện nay là dùng xung điều rộng PWM để điều khiển . Tư tưởng của phương pháp đó là thay đổi độ rộng xung điều khiển để thay đổi mức năng lượng cấp cho động cơ thông qua việc thay đổi điện áp ; từ đó tốc độ động cơ sẽ được thay đổi .

Để nắm được tư tưởng của việc điều khiển động cơ một chiều thì trước tiên ta tìm hiểu nguyên lý của mạch cầu H , điều đó sẽ giúp ta hình thành tư tưởng chung.

Mạch cầu H là một trong những mạch phổ biến để điều khiển động cơ một chiều . Sơ đồ nguyên lý :



Có 4 kháo chuyển 1,2,3,4 . Tại một thời điểm luôn luôn có hai khóa mở và hai khóa đóng . Nếu khóa 1 và 4 đóng thì dòng điện sẽ chạy từ trái qua phải , làm động cơ quay . Nếu khóa 2 và 3 đóng thì dòng điện chay từ phải qua trái ,

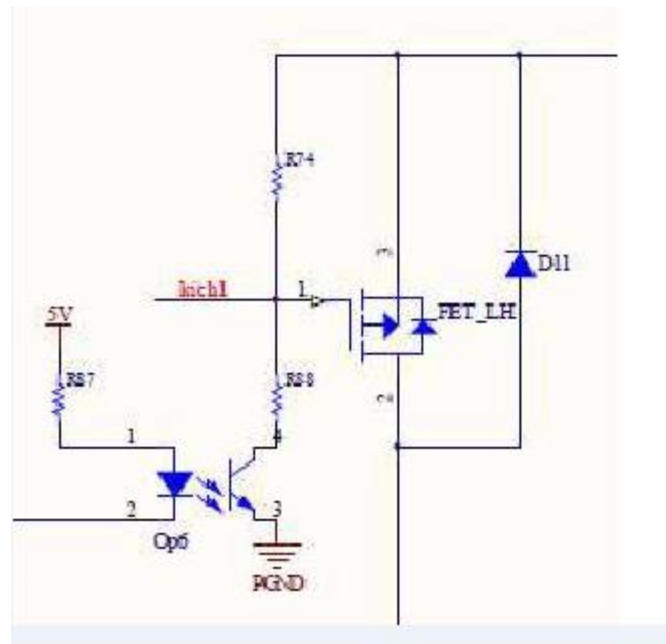
động cơ sẽ quay theo chiều ngược lại . Tránh để cho 1 và 3 hoặc 2 và 4 cùng đóng , vì nó sẽ gây ra hiện tượng đoản mạch . Thay đổi việc đóng mở các van giúp tat hay đổi chiều quay của động cơ .

Nếu chúng ta cho 2 van 1 và 4 đóng mở liên tục thì điện áp đặt lên đông cơ sẽ có dạng xung , vì vậy tốc độ động cơ sẽ thay đổi khi ta thay đổi độ rộng của xung .

Lưu ý: để động cơ làm việc mượt thì quá trình đóng ngắt các van phải nhanh và không được rung .

2. Ví dụ minh họa.

Sau đây ta xét một ví dụ đơn giản để các bạn bước đầu làm quen với điều khiển động cơ :



Mạch nguyên lý

Mạch sử dụng Mosfet để điều khiển động cơ , diode D11 có tác dụng bảo vệ cho Fet , Op6 dùng để cách li mạch động cơ với mạch điều khiển để đảm bảo an toàn cho mạch điều khiển khỏi dòng từ động cơ dội lại .

Sau đây là code của bài toán :

```
#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 : Ở trên ta đã nêu ra một ví dụ đơn giản về điều khiển động cơ , bạn hãy viết lại code trên sử dụng modul PWM của timer .

Lưu ý : Trong phần bài toán mở rộng chúng tôi có một bài điều khiển động cơ một chiều dung modul PWM của timer1 , cac bạn có thể tham khảo để hoàn thiện hơn về kiến thức

Bài 16: GIAO TIẾP VỚI GLCD

- Giới thiệu GLCD

-Nguyên lí dao tiếp GLCD

-Ví dụ minh họa

1. Giới thiệu GLCD

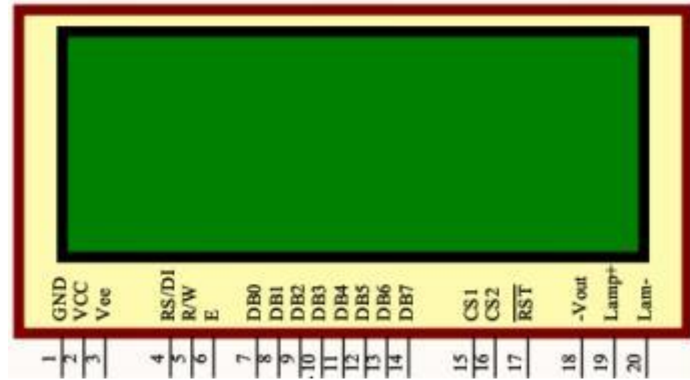
Graphic LCD (gọi tắt là GLCD) loại chấm không màu là các loại màn hình tinh thể lỏng nhỏ dùng để hiển thị chữ, số hoặc hình ảnh. Khác với Text LCD, GLCD không được chia thành các ô để hiển thị các mã ASCII vì GLCD không có bộ nhớ CGRAM (Character Generation RAM). LCD 128x64 có 128 cột và 64 hàng tương 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

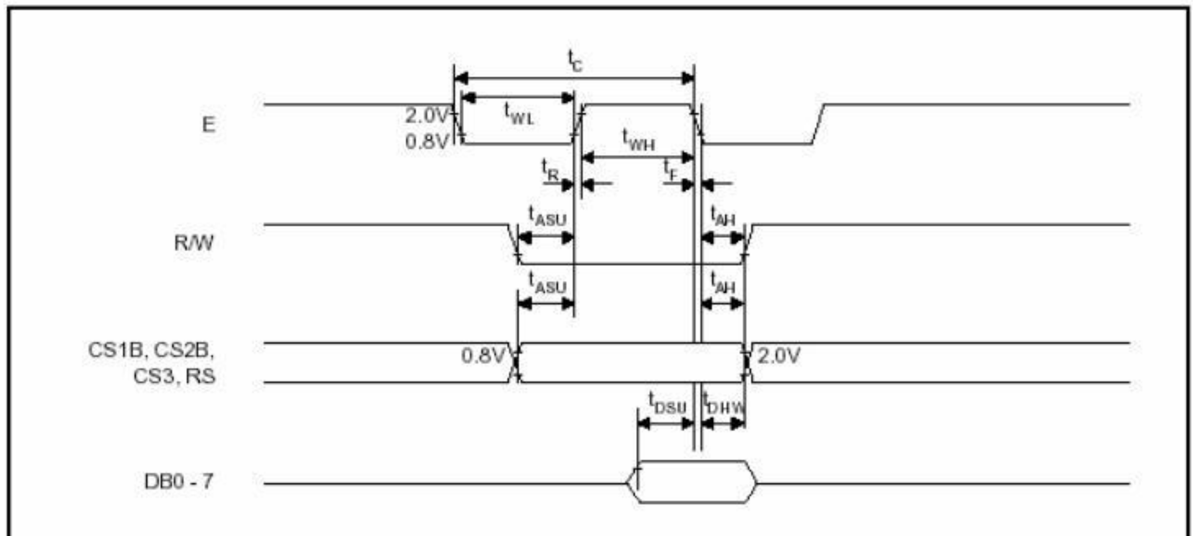
Chân VSS được nối trực tiếp với GND, chân VDD nối với nguồn +5V, một biến trở khoảng 20K được dùng để chia điện áp giữa Vdd và Vee cho chân Vo, bằng cách thay đổi giá trị biến trở chúng ta có thể điều chỉnh độ tương phản của GLCD. Các chân điều khiển RS, R/W, EN và các đường dữ liệu được nối trực tiếp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

Bảng chức năng của các chân GLCD :

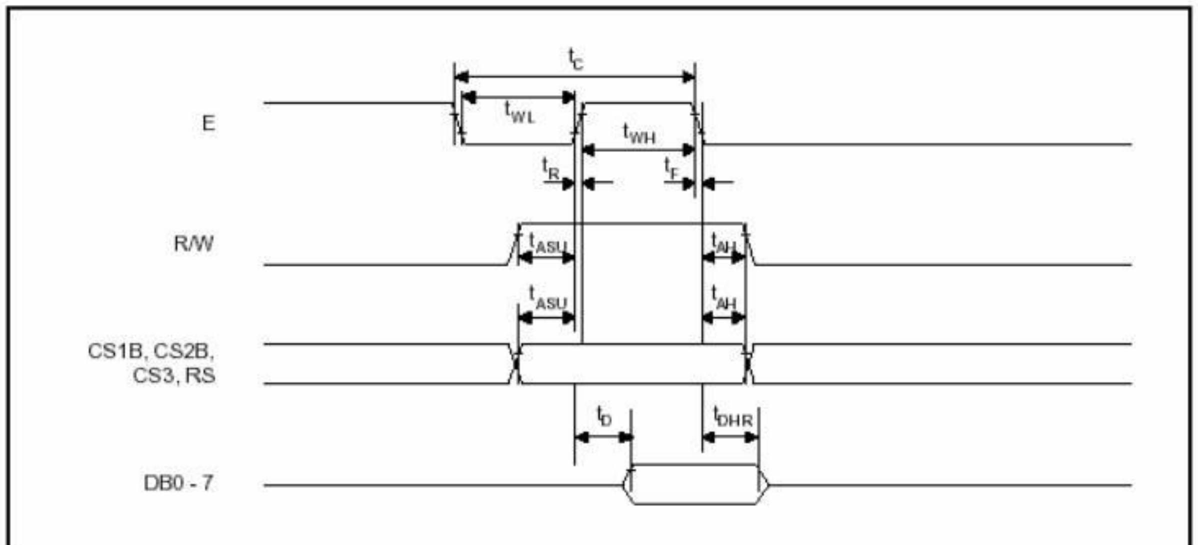
PIN NO.	SYMBOL	DESCRIPTION	FUNCTION
1	VSS	GROUND	0V (GND)
2	VDD	POWER SUPPLY FOR LOGIC CIRCUIT	+5V
3	V0	LCD CONTRAST ADJUSTMENT	
4	RS	INSTRUCTION/DATA REGISTER SELECTION	RS = 0 : INSTRUCTION REGISTER RS = 1 : DATA REGISTER
5	R/W	READ/WRITE SELECTION	R/W = 0 : REGISTER WRITE R/W = 1 : REGISTER READ
6	E	ENABLE SIGNAL	
7	DB0	DATA INPUT/OUTPUT LINES	8 BIT: DB0-DB7
8	DB1		
9	DB2		
10	DB3		
11	DB4		
12	DB5		
13	DB6		
14	DB7		
15	CS1	CHIP SELECTION	CS1=1,CHIP SELECT SIGNAL FOR IC1
16	CS2	CHIP SELECTION	CS2=1,CHIP SELECT SIGNAL FOR IC2
17	RST	RESET SIGNAL	RSTB=0,DISPLAY OFF,DISPLAY FROM LINE 0.
18	VEE	NEGATIVE VOLTAGE FOR LCD DRIVING	-10V
19	LED+	SUPPLY VOLTAGE FOR LED+	+5V
20	LED-	SUPPLY VOLTAGE FOR LED-	0V

Bảng phân khe thời gian giao tiếp

- **WRITE OPERATION**



- **READ OPERATION**



MPU Interface

Characteristic	Symbol	Min	Typ	Max	Unit
E cycle	t_C	1000	–	–	ns
E high level width	t_{WH}	450	–	–	ns
E low level width	t_{WL}	450	–	–	ns
E rise time	t_R	–	–	25	ns
E fall time	t_F	–	–	25	ns
Address set-up time	t_{ASU}	140	–	–	ns
Address hold time	t_{AH}	10	–	–	ns
Data set-up time	t_{DSU}	200	–	–	ns
Data delay time	t_D	–	–	320	ns
Data hold time (write)	t_{DHW}	10	–	–	ns
Data hold time (read)	t_{DHR}	20	–	–	ns

Bảng mã lệnh dùng cho GLCD

Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Function
Display on/off	L	L	L	L	H	H	H	H	H	L/H	Controls the display on or off. Internal status and display RAM data is not affected. L: OFF, H: ON
Set address (Y address)	L	L	L	H	Y address (0 - 63)						Sets the Y address in the Y address counter.
Set page (X address)	L	L	H	L	H	H	H	Page (0 - 7)			Sets the X address at the X address register.
Display start line (Z address)	L	L	H	H	Display start line (0 - 63)						Indicates the display data RAM displayed at the top of the screen.
Status read	L	H	Busy	L	On/ Off	Rese t	L	L	L	L	Read status. BUSY L: Ready H: In operation ON/OFF L: Display ON H: Display OFF RESET L: Normal H: Reset
Write display data	H	L	Write data								Writes data (DB0:7) into display data RAM. After writing instruction, Y address is increased by 1 automatically.
Read display data	H	H	Read data								Reads data (DB0:7) from display data RAM to the data bus.

2. Ví dụ minh họa

Căn cứ vào phân khe thời gian ta có thể lập trình hàm đọc và viết cho GLCD

File library.h

```
// | ---------------------- y
// |
// |
// |
// |
// x
#include <mega32.h>
#include <delay.h>
#include <math.h>
#define GLCD_DI PORTB.4 //Data/Instruction=1/0
#define GLCD_RW PORTB.3 //Read/Write=1/0
#define GLCD_EN PORTB.2 //Enable signal
#define GLCD_Data PORTD
#define GLCD_Busy_bit PIND.7
#define GLCD_Cs1 PORTB.0  //select LCD1
#define GLCD_Cs2 PORTB.1  //select LCD2
void GLCD_Select_chip(unsigned char select_chip);
void GLCD_Reset_chip();
unsigned char Check_busy_bit(unsigned char select_chip);
void GLCD_Wait_busy(unsigned char select_chip);
void GLCD_Write_Cmd(unsigned char cmd,unsigned char select_chip);
void GLCD_Write_Data(unsigned char data,unsigned char select_chip);
void GLCD_Address_X(unsigned char page,unsigned char
select_chip);//x=0:7   page
void GLCD_Address_Y(unsigned char y,unsigned char
select_chip);//y=0:63
void GLCD_Display_Z(unsigned char z,unsigned char
select_chip);//z=0:63  control display (dieu khien su chuyen dong len
xuong)
void GLCD_goto_XY(unsigned char x,unsigned char y);// display dot x,y
void GLCD_Rect();
void GLCD_Draw_Circle(unsigned char X1,unsigned char Y1,unsigned
char R);
```

File library.c

```
#include "library.h"
void GLCD_Select_chip(unsigned char select_chip)
{
if(select_chip==1) // Select chip 1
{
GLCD_Cs1=1;
GLCD_Cs2=0;
}
else if(select_chip==2) // Select chip 1
{
GLCD_Cs1=0;
GLCD_Cs2=1;
}
}
void GLCD_Reset_chip()
{

GLCD_Cs1=0;
GLCD_Cs2=0;

}
unsigned char Check_busy_bit(unsigned char select_chip)
{
unsigned char address;
GLCD_DI=0;
GLCD_RW=1;
GLCD_Select_chip(select_chip);
DDRD=0x00;
GLCD_EN=1;

address=GLCD_Busy_bit;
delay_us(5);    //thoi gian doc dia chi tu lcd chi dung khi vdk co tan so cao
khoang >8Mhz
GLCD_EN=0;
GLCD_RW=0;
```

```
GLCD_DI=1;
DDRD=0xff;
GLCD_Reset_chip();
return address;
}
void GLCD_Wait_busy(unsigned char select_chip)
{
unsigned char busy_bit;
busy_bit=Check_busy_bit(select_chip);
while(busy_bit)
{
 busy_bit=Check_busy_bit(select_chip);
}
}
void GLCD_Write_Cmd(unsigned char cmd,unsigned char select_chip)
{
GLCD_EN=0;
GLCD_RW=0;
GLCD_DI=0;
GLCD_Select_chip(select_chip);
GLCD_EN=1;
GLCD_Data=cmd;
GLCD_EN=0;
GLCD_Wait_busy(select_chip);
GLCD_RW=1;
GLCD_DI=1;
GLCD_Reset_chip();

}
void GLCD_Write_Data(unsigned char data,unsigned char select_chip)
{
GLCD_EN=0;
GLCD_RW=0;
GLCD_DI=1;
GLCD_Select_chip(select_chip);
```

```
GLCD_EN=1;
GLCD_Data=data;
GLCD_EN=0;
GLCD_Wait_busy(select_chip);
GLCD_RW=1;
GLCD_DI=0;
GLCD_Reset_chip();

}
//Display on
void GLCD_Address_X(unsigned char page,unsigned char select_chip)
{
 GLCD_Write_Cmd(0xb8+page,select_chip);
}
void GLCD_Address_Y(unsigned char y,unsigned char select_chip)
{
 GLCD_Write_Cmd(0x40+y,select_chip);
}
void GLCD_goto_XY(unsigned char x,unsigned char y)
{
if(y<64)
{
GLCD_Address_X(x,1);
GLCD_Address_Y(y,1);
}
else
{
GLCD_Address_X(x,2);
GLCD_Address_Y(y-64,2);
}
}
void GLCD_Rect()
{
unsigned char i,chip;
for(i=0;i<=127;i++)
```

```
{
if(i<64) chip=1;
else chip=2;
GLCD_goto_XY(0,i);
GLCD_Write_Data(0x01,chip);
GLCD_goto_XY(7,i);
GLCD_Write_Data(0x80,chip);
}
for(i=0;i<=7;i++)
{
GLCD_goto_XY(i,0);
GLCD_Write_Data(0xff,1);
GLCD_goto_XY(i,127);
GLCD_Write_Data(0xff,2);
}
}
```

File main.c

```
#include "library.h"

// Declare your global variables here
void main(void)
{
DDRD=0xff;
DDRB=0xff;
GLCD_Rect();
}
```

BÀI TẬP MỞ RỘNG

Bài 1: (Bàn phím với LED 7 thanh): Đếm sô lần ấn phím và hiện thị lên LED 7 thanh, sử dung vđk Atmega32.

- Giải thuật:
  + Khởi tạo đầu vào/ra cho Atmega32
  + Phát hiện phím ấn
  + Chống rung của phím bấm (bằng phần mềm hoặc phần cứng)
  + Đợi phím bấm được nhả ra
  + Đếm số lần ấn phím, hiển thị ra LED
- Ghép nối phần cứng:
  + PORTB nối với các chân của LED 7 thanh. Cụ thể cách ghép nối và mã code tương ứng là:

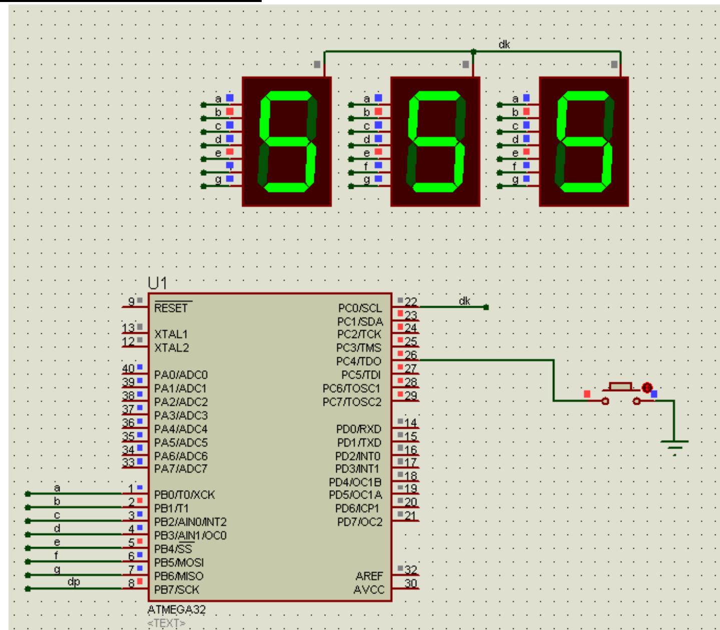
Số	PB.7	PB.6	PB.5	PB.4	PB.3	PB.2	PB.1	PB.0	Mã Hex
	dp	g	f	E	d	c	B	a	
0	1	1	0	0	0	0	0	0	C0
1	1	1	1	1	1	0	0	1	F9
2	1	0	1	0	0	1	0	0	A4
3	1	0	1	1	0	0	0	0	B0
4	1	0	0	1	1	0	0	1	99
5	1	0	0	1	0	0	1	0	92
6	1	0	0	0	0	0	1	0	82
7	1	1	1	1	1	0	0	0	F8
8	1	0	0	0	0	0	0	0	80
9	1	0	0	1	0	0	0	0	90

  + phím ấn được nối với chân PINC.4
  + Các chân điều khiển của LED 7 thanh được nối với các chân PORTC0:3
- Lập trình cho chip trên phần mềm CodeVision.

```
#include <mega32.h>
#include <delay.h>
#define ctac              PINC.4
#define LED7SEG        PORTB
// 4 bit dieu khien 4 LED7SEG PORTC0:3
unsigned char dem = 0;             // so lan an phim
unsigned char phim_an();            // phat hien nut duoc bam
unsigned char so_lan_an_phim(char x);
void hien_thi(unsigned char dem);

void main(void)
{
PORTB=0x00;
DDRB=0xFF;
PORTC=0xFF;
DDRC=0x0F;      //0000 1111
```

```
while (1)
      {
      phim_an();
      hien_thi(dem);
      };
}
unsigned char phim_an()
{
if (ctac==0)                              // co phim duoc an
        {
        delay_ms(25);                     // chong rung phim
        while (ctac==0)     {;};          // doi cho den khi phim duoc nha ra
        delay_ms(25);                     // chong rung phim
        dem++;
        if (dem==10)        dem = 0;
        }
return dem;
}

void hien_thi(unsigned char dem)
{
// PORTB.7 - dp, g-6, f-5, e-4, d-3, c-2, b-1, a-0
unsigned char code[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
LED7SEG = code[dem];
}
```

- <u>Mô phỏng trên phần mềm Proteus</u>:



Bài 2 (Đèn giao thông). Điều khiển các đèn xanh, đỏ, vàng (ở 2 làn đường giao nhau 1 và 2) và hiển thì thời gian các đèn sáng, đếm ngược hiển thị trên LED 7 thanh.

Đầu vào: Thời gian đèn xanh1, xanh2, vàng1, vàng2 sáng của 2 làn đường (đơn vị: giây) (thời gian đèn đỏ sáng được xác định theo công thức đỏ1=xanh2+vàng1, đỏ2=xanh1+vàng1).
Đầu ra: điều khiển các đèn xanh, đỏ, vàng, và đếm ngược thời gian đèn sáng hiển thị ra LED 7 thanh.

- Giải thuật:

  + Lập biểu đồ thời gian các đèn sáng.
  Nhận thấy là chu kì (tổng thời gian 3 đèn sáng) là T=đỏ+vàng+xanh
  Ta có thể kết hợp giữa timer và counter của VĐK tạo ra các khoảng thời gian trong 1 chu kì này, so sánh các khoảng thời gian và điều khiển các đèn sáng.
  + Khởi tạo đầu vào ra, timer/counter của Atmega32.
  Trong Atmega32 có 3 timer/counter. Có thể chọn 1 trong 3 timer/counter để tạo ra khoảng thời gian đơn vị.
  Trong bài này timer/counter0 được chọn, tạo ra khoảng thời gian đơn vị là 8ms, với tần số dao động của thạch anh là f = 4MHz, prescale = 256, initial value = 131.
  + So sánh các khoảng thời gian, điều khiển các đèn sáng, và hiển thị thời gian đếm ngược của mỗi đèn khi sáng.

- Ghép nối phần cứng:

  + 6 chân vđk PORTA0:5 điều khiển 6 đèn giao thông.
  + PORTB: đưa dữ liệu ra 2 LED 7 thanh, hiện thị thời gian làn đường 1
  + PORTC: đưa dữ liệu ra 2 LED 7 thanh, hiện thị thời gian làn đường 2
  + PORTA.6, PORTA.7: điều khiển 2 LED ở làn đường 1 (dùng để quét LED)
  +PORTD.0, PORTD.1: điều khiển 2 LED ở làn đường 2
  Chú ý: cách ghép nối này có thể được thay đổi tùy ý trong chương trình để phù hợp với phần cứng

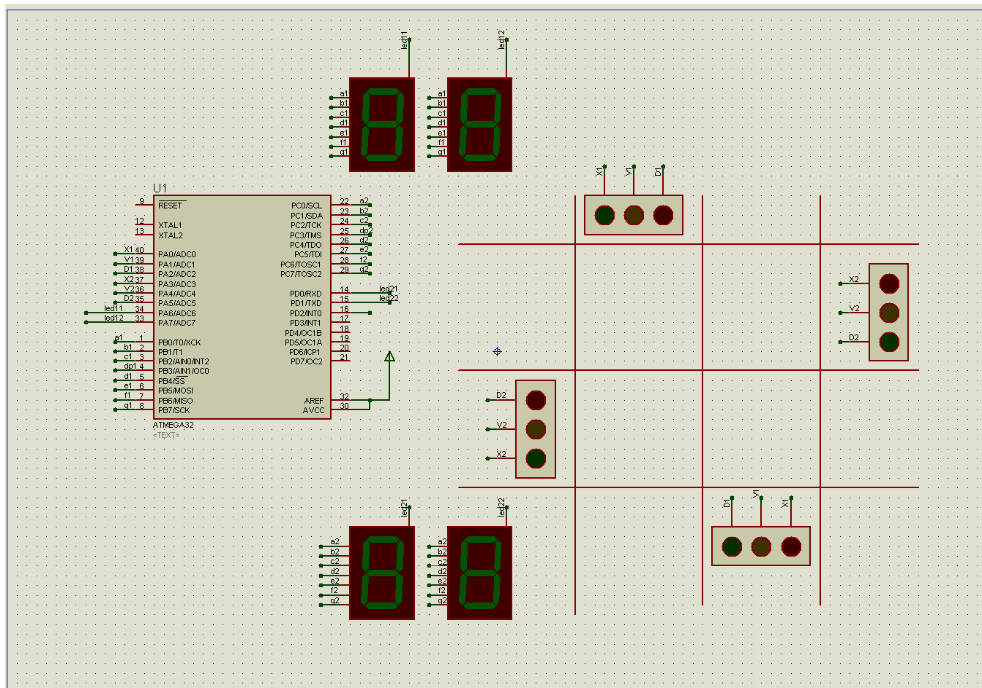
- Chương trình được soạn trên phần mềm CodeVision:

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

#define LED_X1  PORTA.0
#define LED_V1  PORTA.1
#define LED_D1  PORTA.2
#define LED_X2  PORTA.3
#define LED_V2  PORTA.4
#define LED_D2  PORTA.5

#define LED7SEG1    PORTB   // led dem cho lan duong 1
#define LED7SEG2    PORTC   // LED dem cho lan duong 2

#define Led11   PORTA.6     // bit dieu khien LED 7 thanh thu nhat (hang chuc) cua LED7SEG1
#define Led12   PORTA.7
#define Led21   PORTD.0
```

```
#define Led22   PORTD.1

const int den_xanh1 =  11;
const int den_xanh2 =  11;      //
const int den_vang1  =  6;       //
const int den_vang2 =   6;

int den_do1, den_do2;

// PORTB.7 - dp, g-6, f-5, e-4, d-3, c-2, b-1, a-0
unsigned char code[]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};

// Declare your global variables here
int counter=0;      //  0..(1/8m)*60=0..7500
unsigned char gt1;  //gt=1,den xanh 1 sang,gt=2 den vang 1 sang, gt=3 den do 1 sang
unsigned char gt2;
int value1=0, value2=0;
char chuc1=0, donvi1=0, chuc2=0, donvi2=0;

// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Reinitialize Timer 0 value
TCNT0=0x83;             //256-125=131=0x83
// Place your code here
// moi khi tran xay ra thi duoc 8ms
counter++;
if (counter == 125*(den_xanh1 + den_vang1 + den_do1))       counter=0;// het 1 chu ki

if (counter >=0 && counter <= 125*den_xanh1)       {gt1=1; LED_X1=1; LED_V1=0;
LED_D1=0;}
if (counter >= 125*den_xanh1 && counter <= 125*(den_vang1+den_xanh1))       {gt1=2;
LED_X1=0; LED_V1=1; LED_D1=0;}
if (counter >= 125*(den_vang1+den_xanh1) && counter <=
125*(den_xanh1+den_do1+den_vang1))   {gt1=3; LED_X1=0; LED_V1=0; LED_D1=1;}

if (counter >=0 && counter <= 125*den_do2)        {gt2=3; LED_X2=0; LED_V2=0;
LED_D2=1;}
if (counter >= 125*den_do2 && counter<= 125*(den_do2+den_xanh2))      {gt2=1;
LED_X2=1; LED_V2=0; LED_D2=0;}
if (counter >= 125*(den_do2+den_xanh2) && counter <=
125*(den_do2+den_xanh2+den_vang2))     {gt2=2; LED_X2=0; LED_V2=1; LED_D2=0;}
}

void hien_thi_led()      // dem nguoc thoi gian cac den
{
if (gt1==1) // den xanh 1 sang, den nguoc tu gia tri [den_xanh...0]
       value1 = den_xanh1 - counter/125;
if (gt1==2)     value1 = den_vang1 + den_xanh1 - counter/125;
if (gt1==3)     value1 = den_do1 + den_xanh1 + den_vang1 - counter/125;

if (gt2==3)     value2 = den_do2 - counter/125;
if (gt2==1)     value2 = den_do2 + den_xanh2 - counter/125;
if (gt2==2)     value2 = den_do2 + den_xanh2 + den_vang2 - counter/125;
```

```
value1--;  value2--;
chuc1 = value1/10;
donvi1= value1%10;
chuc2 = value2/10;
donvi2= value2%10;
}

void main(void)
{
// Declare your local variables here
DDRA=0xFF;          PORTA=0x00;
DDRB=0xFF;          PORTB=0x00;
DDRC=0xFF;          PORTC=0x00;
DDRD=0xFF;          PORTD=0x00;

gt1=1;
gt2=3;

den_do2 = den_xanh1 + den_vang1;
den_do1 = den_xanh2 + den_vang2;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 15.625 kHz
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x04;         // prescale=256, CS02=1, CS01=CS00=0;
TCNT0=0x83;         // 256-125=131=0x83
OCR0=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x01;         // TOIE0=1, cho phep co ngat khi tran

// Global enable interrupts
#asm("sei")

while (1)
    {
    Led11 = 1; Led12 = 0;
    LED7SEG1 = code[chuc1];
    Led11 = 0; Led12 = 1;
    LED7SEG1 = code[donvi1];
    Led21 = 1; Led22 = 0;     //PORTD=1;
    LED7SEG2 = code[chuc1];
    Led21 = 0; Led22 = 1;     //PORTD=2;
    LED7SEG2 = code[donvi2];

    if ((counter%120)==0)
        {
        hien_thi_led();
        }
    };}
```

- Mô phỏng trên phần mềm Proteus:



Bài 3: Điều khiển tốc độ động cơ 1 chiều theo từng mức bằng ma trận bàn phím

- Sử dụng khả năng xuất xung điều rộng (PWM – Pulse Width Modulation) của Atmega32 để điều khiển tốc độ động cơ DC
- Giải thuật:

  + Khai báo vào ra cho chip.

  Cài đặt các bit cho timer/counter 1 của mega32 để làm việc ở chế độ xuất xung điều rộng. Cụ thể:

  Trong bài toán này: tần số xung nhịp vào ra là 4MHz, Mode 14 (Fast PWM) được sử dụng, chu kì 20ms, do đó prescale của T/C1 được chọn là 8, giá trị TOP là TOP = ICR1=10000.

  Các chân PD4(OCB1), PD5(OCA1) được khai báo là các đầu ra xuất xung PWM.

  PORTB nối với LCD

  PORTA nối với ma trận bàn phím 4x4

  + Nhận biết phím bấm từ ma trận bàn phím, thay đổi giá trị của các thanh ghi OCR1A, OCR1B để có mức điều khiển tương ứng.
- Lập trình trên phần mềm CodeVision

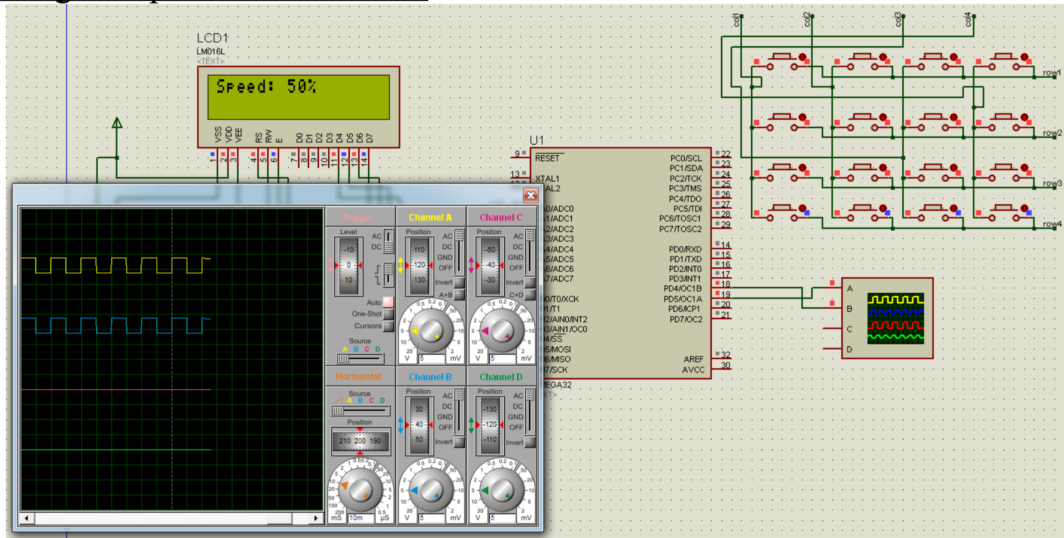
```
#include <mega32.h>

#asm
   .equ __lcd_port=0x18 ;PORTB
#endasm

#include <lcd.h>
#define TOP 10000
float ratio=0.5;            // t/T=ratio
```

```
// Timer 1 overflow interrupt service routine
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
// Place your code here

}

// Timer 1 output compare A interrupt service routine
interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
// Place your code here
OCR1A = TOP*ratio;
}

// Timer 1 output compare B interrupt service routine
interrupt [TIM1_COMPB] void timer1_compb_isr(void)
{
// Place your code here
OCR1B = TOP*ratio;
}

// PORTB - LCD

void main(void)
{
int i,row=0, col=0, phim = -4;
int phim_old =-4,phim_new =-4;
char code[]={0xEF, 0xDF, 0xBF, 0x7F};
//         {1110.1111, 1101.1111, 1011.1111, 0111.1111}
PORTA=0xFF;
DDRA=0xF0;     // col1-PINA.0, col2-PINA.1,..., ROW1=PORTA.4,ROW2=PORTA.5
PORTB=0xFF;
DDRB=0xFF;     // PORTB connects with LCD, is ouput

// Port D initialization
PORTD=0x00;
DDRD=0x30;         //0011.0000 PORTD.4=OC1B, PORTD.5=OC1A

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 500.000 kHz, prescale = 8
// Mode: Fast PWM top=ICR1 , mode = 14
// OC1A output: Non-Inv.   , COM1A1=1,COM1A0=0
// OC1B output: Non-Inv.   , COM1B1=1,COM1B0=0
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: On
// Input Capture Interrupt: Off
// Compare A Match Interrupt: On
// Compare B Match Interrupt: On
TCCR1A=0xA2;       //1010.0010
TCCR1B=0x1A;       //0001.1010
```

```
TCNT1H=0x00;        // Bottom = 0x0000
TCNT1L=0x00;

ICR1H = 0x27;       // Top = ICR1 = 0x2710 = 10000 = 10.000*2us = 20ms = Time period
ICR1L = 0x10;
OCR1A = TOP*ratio;      // 50%
OCR1B = TOP*ratio;      // 50%

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x1C;

// Global enable interrupts
#asm("sei")
lcd_init(16);
lcd_putsf("Speed: 50%");
while (1)
    {
    for (i=0;i<4;i++)
        {
        PORTA = code[i];
        if (PINA.0 == 0)        {row = i+1; col = 1;} //column 1
        if (PINA.1 == 0)        {row = i+1; col = 2;}
        if (PINA.2 == 0)        {row = i+1; col = 3;}
        if (PINA.3 == 0)        {row = i+1; col = 4;}
        }
    // xAC dinh phim duoc bam la phim nao
    phim = 4*row+col-4;
    phim_new = phim;
    if (phim >= 0 && phim_new!=phim_old)          // co phim duoc bam
        {
        if (phim == 1)
            {
            ratio = 0.2;
            lcd_control(1);
            lcd_putsf("Speed: 20%");
            }
        else {
            if (phim == 2)
                {
                ratio = 0.4;
                lcd_control(1);
                lcd_putsf("speed: 40%");
                }
            else {
                if (phim==3)
                    {
                    ratio = 0.75;
                    lcd_control(1);
                    lcd_putsf("Speed: 75%");
                    }
                else
                    {
                    ratio=0.5;
                    lcd_control(1);
```

```
                        lcd_putsf("Speed: 50%");
                        }
                    }
                }

            }
        };
    }
```

- Mô phỏng trên phần mềm Proteus:



Bài 4: Điều khiển và hiển thị tốc độ động cơ 1 chiều, nhiệt độ, thời gian bằng máy tính PC và màn hình LCD.

- Nội dung:

  Kết hợp các module: điều khiển tốc độ động cơ, đo nhiệt độ bằng LM35, đồng hồ thời gian thực DS1307, giao tiếp với máy tính PC, hiển thị trên LCD.

- Ghép nối phần cứng:

  + LCD nối với PORTB của chip Atmega32
  + Chân ADC0 (PA0) nối với LM35
  + 2 chân I2C là SCL và SDA của DS1307 được nối với 2 chân SCL, SDA tương ứng trên vđk.
  + Cổng comm nối với 2 chân RxD, TxD trên PORTD của Atmega32
  + Chân AREF được nối lên nguồn 5V thông qua điện cảm để làm điện áp tham chiếu.

- Giải thuật:

  + Khai báo đầu vào ra, truyền thông UART, I2C, chuyển đổi ADC với điện áp tham chiếu trên chân AREF, các timer/counter0 để làm bộ định thời xử lý từng việc, timer/counter 1 dùng để điều khiển tốc độ động cơ
  + Giao tiếp với máy tính PC thông qua phần mềm Hercule.

- Chương trình được thực hiện trên phần mềm CodeVision

```
#include <mega32.h>
#include <ds1307.h>
// I2C Bus functions
#asm
   .equ __i2c_port=0x15 ;PORTC
   .equ __sda_bit=1
   .equ __scl_bit=0
#endasm
#include <i2c.h>

// My global variables
#include <stdlib.h>
#define TOP     10000       // TOP - timer1
int timer_counter0=0;       // for timer0
float pwm_ratios=0.5;
unsigned char h=0,m=0,s=0;        // hour, minute, second
int temperature=0;          // gia tri nhiet do duoc chuyen doi
int setting=0;              // setting=0 Normal, setting=1; setting clock, setting=2 setting PWM

void menu();
void menu_clock();
void menu_pwm();

// Alphanumeric LCD Module functions
#asm
   .equ __lcd_port=0x18 ;PORTB
#endasm
#include <lcd.h>

#define RXB8 1
#define TXB8 0
#define UPE 2
#define OVR 3
#define FE 4
#define UDRE 5
#define RXC 7

#define FRAMING_ERROR (1<<FE)
#define PARITY_ERROR (1<<UPE)
#define DATA_OVERRUN (1<<OVR)
#define DATA_REGISTER_EMPTY (1<<UDRE)
#define RX_COMPLETE (1<<RXC)

// USART Receiver buffer
#define RX_BUFFER_SIZE 8
char rx_buffer[RX_BUFFER_SIZE];

#if RX_BUFFER_SIZE<256
unsigned char rx_wr_index,rx_rd_index,rx_counter;
#else
unsigned int rx_wr_index,rx_rd_index,rx_counter;
#endif

// This flag is set on USART Receiver buffer overflow
```

```
bit rx_buffer_overflow;

// USART Receiver interrupt service routine
interrupt [USART_RXC] void usart_rx_isr(void)
{
char status,data;
status=UCSRA;
data=UDR;
if ((status & (FRAMING_ERROR | PARITY_ERROR | DATA_OVERRUN))==0)
   {
   rx_buffer[rx_wr_index]=data;
   if (++rx_wr_index == RX_BUFFER_SIZE) rx_wr_index=0;
   if (++rx_counter == RX_BUFFER_SIZE)
      {
      rx_counter=0;
      rx_buffer_overflow=1;
      };
   };
}

#ifndef _DEBUG_TERMINAL_IO_
// Get a character from the USART Receiver buffer
#define _ALTERNATE_GETCHAR_
#pragma used+
char getchar(void)
{
char data;
while (rx_counter==0);
data=rx_buffer[rx_rd_index];
if (++rx_rd_index == RX_BUFFER_SIZE) rx_rd_index=0;
#asm("cli")
--rx_counter;
#asm("sei")
return data;
}
#pragma used-
#endif

// USART Transmitter buffer
#define TX_BUFFER_SIZE 8
char tx_buffer[TX_BUFFER_SIZE];

#if TX_BUFFER_SIZE<256
unsigned char tx_wr_index,tx_rd_index,tx_counter;
#else
unsigned int tx_wr_index,tx_rd_index,tx_counter;
#endif

// USART Transmitter interrupt service routine
interrupt [USART_TXC] void usart_tx_isr(void)
{
if (tx_counter)
   {
   --tx_counter;
```

```
  UDR=tx_buffer[tx_rd_index];
  if (++tx_rd_index == TX_BUFFER_SIZE) tx_rd_index=0;
  };
}

#ifndef _DEBUG_TERMINAL_IO_
// Write a character to the USART Transmitter buffer
#define _ALTERNATE_PUTCHAR_
#pragma used+
void putchar(char c)
{
while (tx_counter == TX_BUFFER_SIZE);
#asm("cli")
if (tx_counter || ((UCSRA & DATA_REGISTER_EMPTY)==0))
  {
  tx_buffer[tx_wr_index]=c;
  if (++tx_wr_index == TX_BUFFER_SIZE) tx_wr_index=0;
  ++tx_counter;
  }
else
  UDR=c;
#asm("sei")
}
#pragma used-
#endif

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

// Timer 1 overflow interrupt service routine
interrupt [TIM1_OVF] void timer1_ovf_isr(void)
{
// Reinitialize Timer 1 value
//TCNT1H=0x2710 >> 8;
//TCNT1L=0x2710 & 0xff;
// Place your code here

}

// Timer 1 output compare A interrupt service routine
interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
// Place your code here

}

// Timer 1 output compare B interrupt service routine
interrupt [TIM1_COMPB] void timer1_compb_isr(void)
{
// Place your code here

}

#include <delay.h>
```

```
#define FIRST_ADC_INPUT 0
#define LAST_ADC_INPUT 7
unsigned int adc_data[LAST_ADC_INPUT-FIRST_ADC_INPUT+1];   // du lieu ADC duoc
chuyen doi, nhiet do t=500*acd/1023
#define ADC_VREF_TYPE 0x00

// ADC interrupt service routine
// with auto input scanning
interrupt [ADC_INT] void adc_isr(void)
{
static unsigned char input_index=0;
// Read the AD conversion result
adc_data[input_index]=ADCW;
// Select next ADC input
if (++input_index > (LAST_ADC_INPUT-FIRST_ADC_INPUT))
   input_index=0;
ADMUX=(FIRST_ADC_INPUT | (ADC_VREF_TYPE & 0xff))+input_index;
// Delay needed for the stabilization of the ADC input voltage
delay_us(10);
// Start the AD conversion
ADCSRA|=0x40;
}

// Timer 0 overflow interrupt service routine
// Chia khoang thoi gian thuc hien cac cong viec
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
char str2[5],str[3];
// Reinitialize Timer 0 value
TCNT0=0x65;     //0x65=101d, 256-101=155*16us=1.68ms
// Place your code here
timer_counter0++;
if (timer_counter0==595)    // 1s, update time, ds1307
        {
        rtc_get_time(&h,&m,&s);
        timer_counter0=0;
        }
if (timer_counter0%5==0)   // update temperature, 5*1.68=8.64ms for convertsion ADC
        {
        temperature = 500*adc_data[0]/1023;
        }
if (timer_counter0%500==0)   // update pwm_ratio
        {
        OCR1A = TOP*pwm_ratios;
        OCR1B = TOP*pwm_ratios;
        }
if (timer_counter0 %550==0) //display LCD
        {
        if (setting==0) //lcd_control(1);
                {
                lcd_clear();
                lcd_putsf("Temp: ");
                itoa(temperature,str);
```

```
            lcd_puts(str);
            lcd_putsf(" oC");

            lcd_gotoxy(0,1);
            lcd_putsf("PWM: ");
            ftoa(100*pwm_ratios,2,str2);
            lcd_puts(str2);
            lcd_putsf(" %");
            }
        if (setting==1)
            {
            lcd_clear();
            lcd_putsf("Setting...time!");
            }
        if (setting==2)
            {
            lcd_clear();
            lcd_putsf("Setting...PWM!");
            }
        }
}

void main(void)
{
unsigned char hh=0,mm=0,ss=0;
char c=27,c_clock=27,c_pwm=27;
unsigned char d,mth,y;

// Declare your local variables here
PORTA=0x00;
DDRA=0x00;

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0x00;

PORTD=0x00;
DDRD=0x30;   //0011.0000 PORTD.4-OC1B, PORTD.5-OC1A as output for PWM

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 62.500 kHz, 4M/64, prescale=64, 64/4M=16us*105=1.68ms
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x03;
TCNT0=0x65;
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 500.000 kHz, prescale = 8
// Mode: Fast PWM top=ICR1 , mode = 14
```

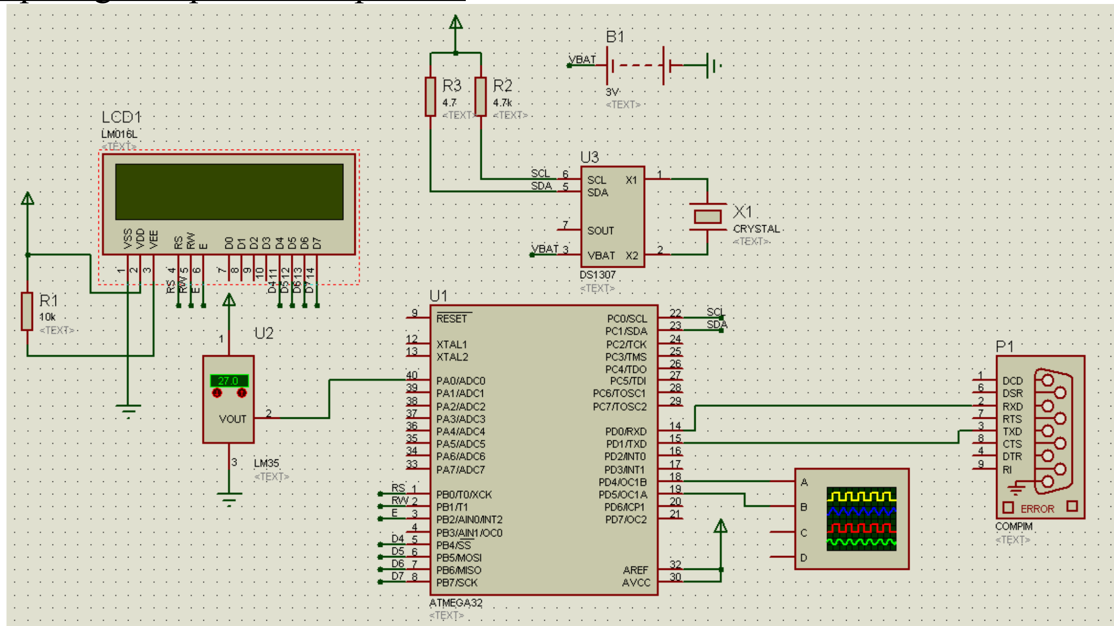
```
// OC1A output: Non-Inv.   , COM1A1=1,COM1A0=0
// OC1B output: Non-Inv.   , COM1B1=1,COM1B0=0
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: On
// Input Capture Interrupt: Off
// Compare A Match Interrupt: On
// Compare B Match Interrupt: On
TCCR1A=0xA2;        //1010.0010
TCCR1B=0x1A;        //0001.1010

TCNT1H=0x00;        // Bottom = 0x0000
TCNT1L=0x00;

ICR1H = TOP >> 8;       // Top = ICR1 = 0x2710 = 10000 = 10.000*2us = 20ms = Time
period
ICR1L = TOP & 0xFF;
OCR1A = TOP*pwm_ratios;      // 50%
OCR1B = TOP*pwm_ratios;      // 50%

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;

// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x1D;

// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud Rate: 9600
UCSRA=0x00;
UCSRB=0xD8;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x19;

// Analog Comparator initialization
// Analog Comparator: Off
```

```
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;

// ADC initialization
// ADC Clock frequency: 31.250 kHz
// ADC Voltage Reference: AREF pin
ADMUX=FIRST_ADC_INPUT | (ADC_VREF_TYPE & 0xff);
ADCSRA=0xCF;

// I2C Bus initialization
i2c_init();
rtc_set_time(12,12,12);
rtc_set_date(23,6,13);
// LCD module initialization
lcd_init(16);

lcd_putsf("Welcome");
delay_ms(100);
rtc_get_time(&h,&m,&s);

menu();

// Global enable interrupts
#asm("sei")

while (1)
    {
    // Place your code here
    c_pwm=27;c_clock=27;c=27;
    c = getchar();
    if (c=='0')
        {
        printf("\n-------Menu------\n");
        c=27;
        menu();
        }
    if (c=='1')
        {
        printf("\n-------Clock------\n");
        menu_clock();
        do
            {
            c_clock=getchar();
            if (c_clock=='0')
                {
                rtc_get_time(&h,&m,&s);
                rtc_get_date(&d,&mth,&y);
                printf("\n Time:%d:%d:%d , Date: %d/%d/%d\n",h,m,s,d,mth,y);
                c_clock=27;
                menu_clock();
                }
            if (c_clock=='1')
                {
```

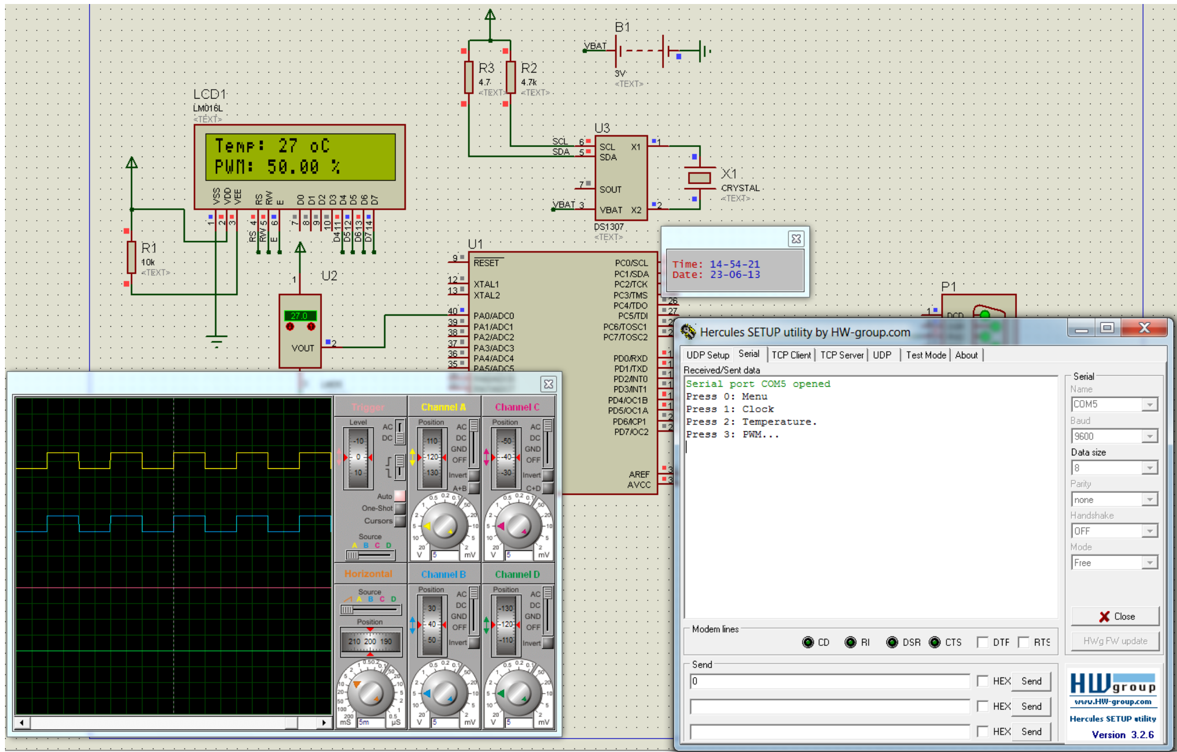
```
                        printf("Setting time\n");
                        setting=1;     // setting time for display LCD
                        printf("Hour = ");
                        scanf("%d",&hh);
                        printf("\nMinute = "); scanf("%d",&mm);
                        printf("\nSecond = "); scanf("%d",&ss);
                        i2c_init();
                        rtc_set_time(hh,mm,ss);
                        setting=0;
                        c_clock=27;
                        menu_clock();
                        }
                }
            while (c_clock=='0' || c_clock=='1' || c_clock==27);
            }
    if (c=='2')
            {
            printf("\nTemperature is: %d",temperature);
            c=27;
            menu();
            }
    if (c=='3')
            {
            printf("\n----------PWM--------\n");
            menu_pwm();
            do
                {
                c_pwm=getchar();
                if (c_pwm=='0')
                        {
                        printf("\nPWM: %f percent",pwm_ratios);
                        c_pwm=27;
                        menu_pwm();
                        }
                if (c_pwm=='1')
                        {
                        setting=2; //setting pwm for display LCD
                        printf("\nSetting PWM");
                        printf("\npwm_ratios = ");
                        scanf("%f",&pwm_ratios);
                        setting=0;
                        c_pwm=27;
                        menu_pwm();
                        }
                }
            while (c_pwm=='0' || c_pwm=='1' || c_pwm==27);

            }
    };
}

void menu()
{
  printf("Press 0: Menu\n");
```

```
    printf("Press 1: Clock\n");
    printf("Press 2: Temperature.\n");
    printf("Press 3: PWM...\n");
}

void menu_clock()
{
    printf("Press 0: Show time and date\n");
    printf("Press 1: Setting time and date\n");
    printf("Press 2: Exit.\n");
}

void menu_pwm()
{
    printf("Press 0: Show PWM\n");
    printf("Press 1: Setting PWM\n");
    printf("Press 2: Exit.\n");
}
```

- <u>Mô phỏng trên phần mềm proteus</u>



-



LCD1
LM016L
Temp: 27 oC
PWM: 50.00 %
R1
10k
U2
VOUT
R3
4.7
R2
4.7k
B1
VBAT
3V
U3
SCL
SDA
SOUT
VBAT
X1
X2
DS1307
X1
CRYSTAL
U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
Time: 14-54-21
Date: 23-06-13
P1
Hercules SETUP utility by HW-group.com
UDP Setup
Serial
TCP Client
TCP Server
UDP
Test Mode
About
Received/Sent data
Serial port COM5 opened
Press 0: Menu
Press 1: Clock
Press 2: Temperature.
Press 3: PWM...
Serial
Name
COM5
Baud
9600
Data size
8
Parity
none
Handshake
OFF
Mode
Free
Close
HWg FW update
Modem lines
CD
RI
DSR
CTS
DTR
RTS
Send
HEX
Send
HW group
www.HW-group.com
Hercules SETUP utility
Version 3.2.6
Trigger
Channel A
Channel C
Horizontal
Channel B
Channel D

Tài liệu tham khảo

1. *Datasheet ATmega8 , ATmega 32 (ATmel Co.) .*
2. *Datasheet DS1307, LED 7seg , 74HC_HCT595_CNV , ...*
3. *Kỹ thuật vi xỷ lý (Văn Thế Minh).*
4. *Kỹ thuật vi xử lý và lập trình cho hệ vi xử lý (Đỗ Xuân Tiến).*
5. *HocAVR.com .*