ĐẠI HỌC CÔNG NGHỆ GIAO THÔNG VẬN TẢI

KỸ THUẬT VI ĐIỀU KHIỂN

(Lưu hành nội bộ)

Chủ biên: ThS. Hoàng Thế Phương

Hà Nội, 2020

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

CHƯƠNG 1: MỞ ĐẦU VỀ VI ĐIỀU KHIỂN

1.1. 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, Pisoc.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.

* Tính năng mới của họ AVR:

-Có thể sử dụng xung clock lên đến 16MHz.

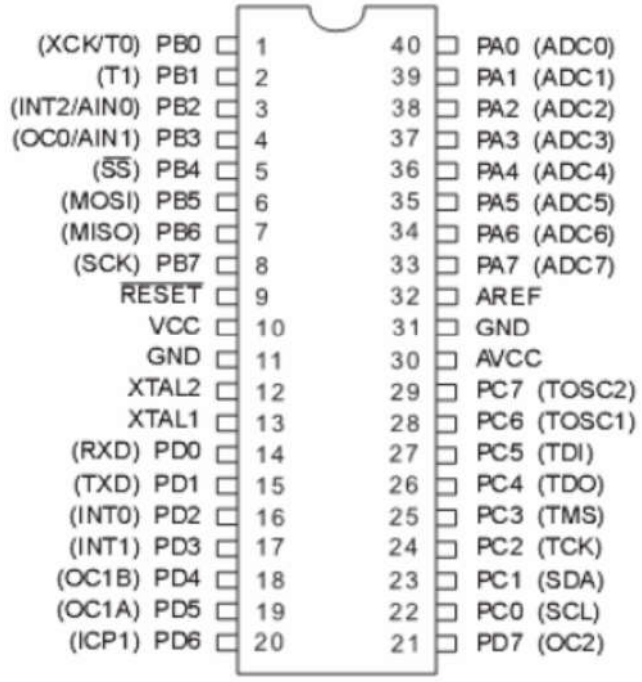
-Bộ nhớ Flash có thể lập trình nhiều lần với dung lượng lớn

- Giao diện SPI đồng bộ.
- Các đường dẫn vào/ra (I/O) 2 hướng 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.
- Bộ Timer/Counter 8 bit,16 bit.
- 1 bộ so sánh analog.
- Bộ nhớ EEPROM.
- Giao tiếp USART..vv..

* Một số chip AVR:

AT90S4414 and AT90S8515
AT90S4434 and AT90S8535
AT90C8534
ATtiny10, ATtiny11 and ATtiny12
ATtiny15
ATtiny22
ATtiny26
ATtiny28
ATmega8/8515/8535
ATmega16
ATmega161
ATmega162
ATmega163
ATmega169
ATmega32
ATmega323
ATmega103
ATmega64/128
AT86RF401.

1.2. Vi điều khiển Atmega16



Hình 1. 1. Vi điều khiển ATMEGA16

Atmega16 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 16K(flash) . - 512 byte (EEPROM). - 1 K (SRAM).

- 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.

- Giao tiếp SPI.

- Giao diện I2C.

- Có 8 kênh ADC 10 bit.

- 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.

- 1 bộ truyền nhận UART lập trình được.

Mô tả các chân:

-4 cổng vào/ra lập trình được, chia làm 4 PORT A,B,C,D.

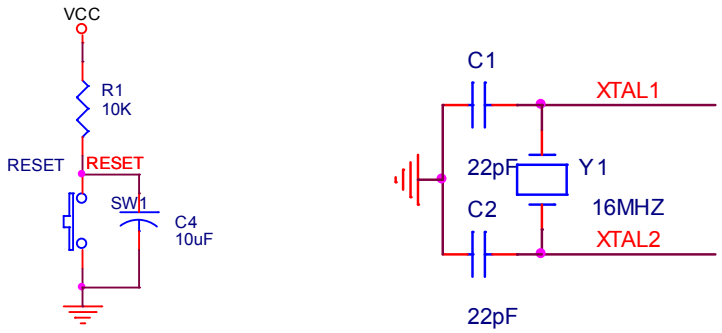
- Vcc và GND 2 chân cấp nguồn cho vi điều khiển hoạt động. Chú ý cần cấp điện áp 5V ổn định.

- 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 8M), tụ gốm (22p).

- Chân Aref thường nối lên 5v(Vcc), nhưng khi sử dụng bộ ADC thì chân này được sử dụng làm điện áp so sánh, khi đó chân này phải cấp cho nó điện áp cố định, có thể sử dụng diode zener

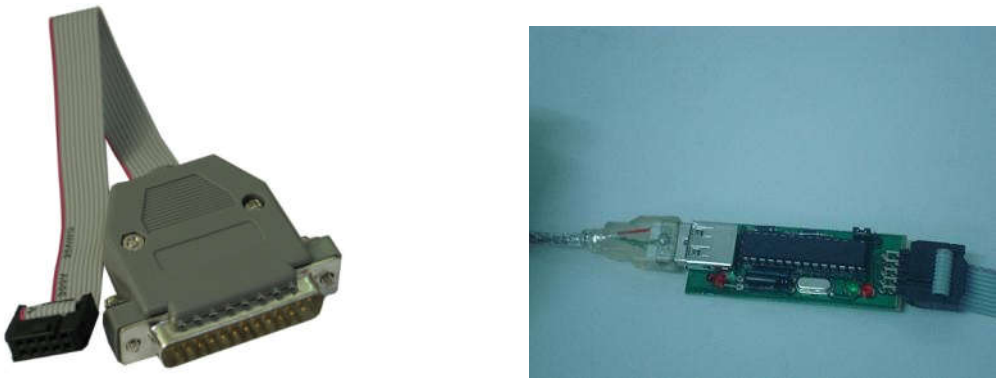
- Chân Avcc là nguồn nuôi cho bộ ADC, thường được nối lên Vcc.



1.3. Các công cụ phần cứng.

1.3.1. Mạch nạp:

Đây là 1 công cụ không thể thiếu khi thực hành VĐK.



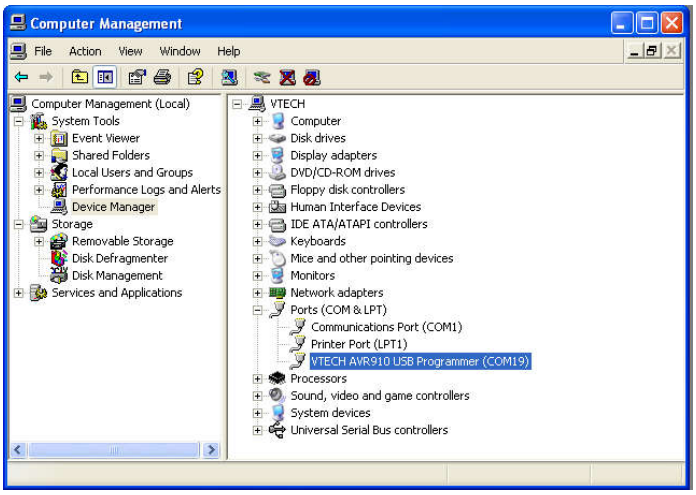
Hình 1. 2. Mạch nạp vi điều khiển

Có 1 số loại mạch nạp phổ biến bao gồm: STK200 (nạp qua cổng máy in), AVR910, STK500 (nạp qua cổng USB).

Cài đặt, cấu hình mạch nạp AVR910

Sau khi cài đặt driver cho mạch nạp xong:

Vào ***Control Panel→System→Advanced→Device Manager***, ta có hộp thoại ***Device Manager*** hiện lên:



Hình 1. 3. Hộp thoại Device manager

+Mở nhánh Ports (COM & LPT), ta sẽ thấy có thông báo mạch nạp AVR910 đang được kết nối với máy tính, cổng COM gán cho mạch là COM19. Ở đây có một số điểm cần lưu ý:

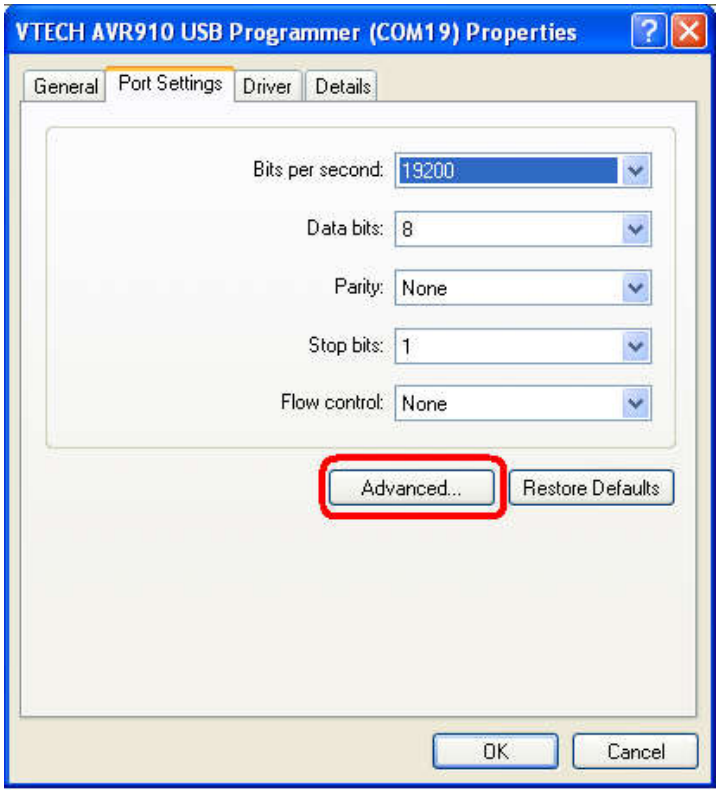
- Cần kết nối mạch nạp với máy tính thì trong phần Ports (COM&LPT) mới hiện lên thông báo về thiết bị.

- Với các máy tính xách tay không có các cổng COM và LPT thực. Phần Ports (COM & LPT) sẽ không hiện lên nếu không có thiết bị nào được gắn vào.

- Tên cổng COM gán cho thiết bị phụ thuộc vào từng máy tính, không phải đều giống nhau, tuy nhiên cần lưu ý rằng **AVR Prog** chỉ hoạt động với các cổng **COM từ 1-4** và **CodeVisionAVR** chỉ hoạt động với các cổng **COM từ 1-6**. Do đó nếu cổng COM được gán tự động ra ngoài khoảng này, ta cần tiến hành gán lại cổng COM bằng tay cho phù hợp!

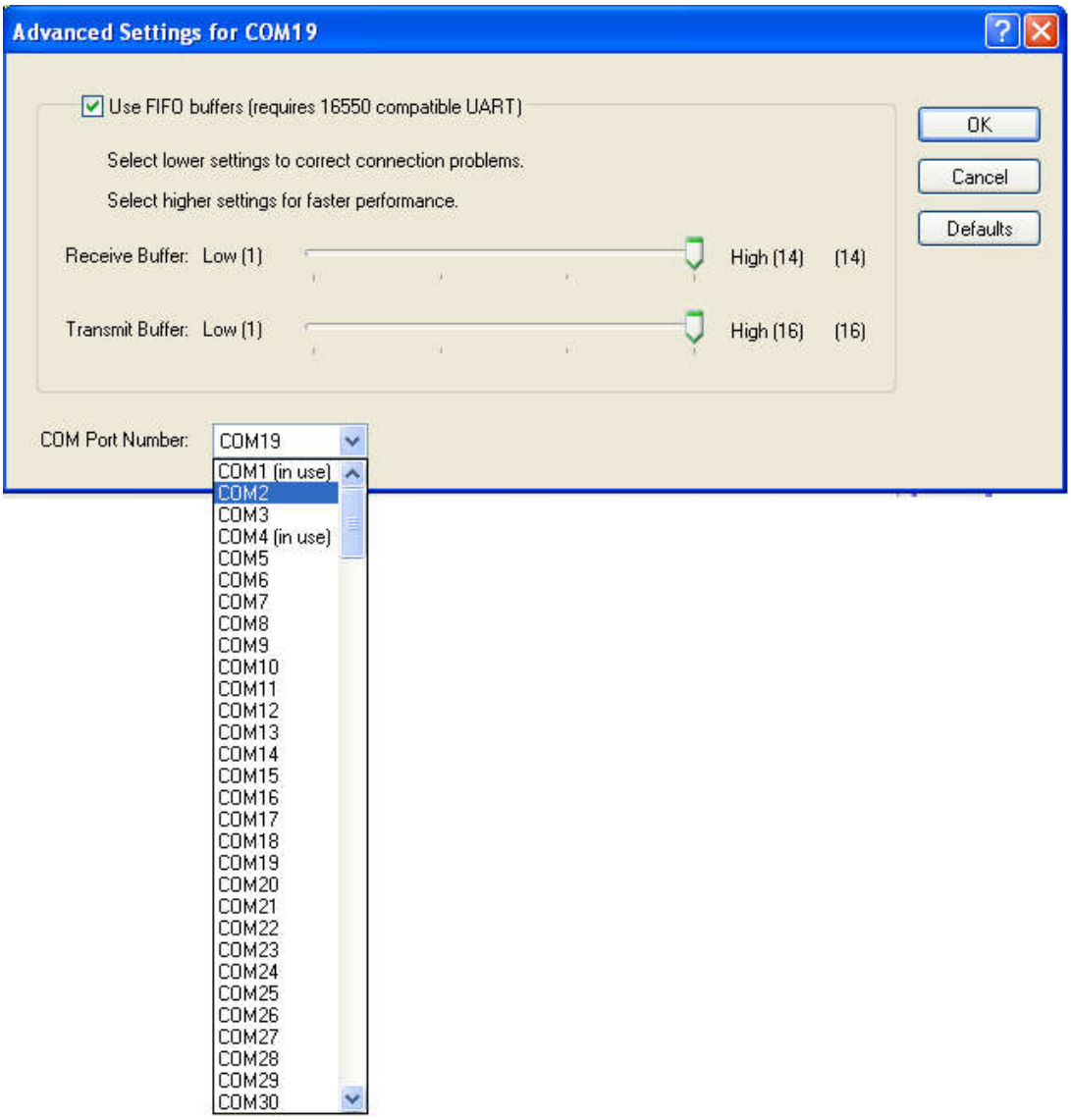
Cách gán lại tên cổng COM cho thiết bị!

-Kích đúp chuột lên nhánh ***VTECH AVR910 USB Programmer*** để mở hộp thoại ***Properties*** của thiết bị (hoặc click chuột phải vào và chọn ***Properties***:



Hình 1. 4. Cấu hình cổng COM giao tiếp

-Chọn Tab ***Port Settings,*** nhấn nút ***Advanced…*** ta sẽ có hộp thoại ***Advanced Settings:***



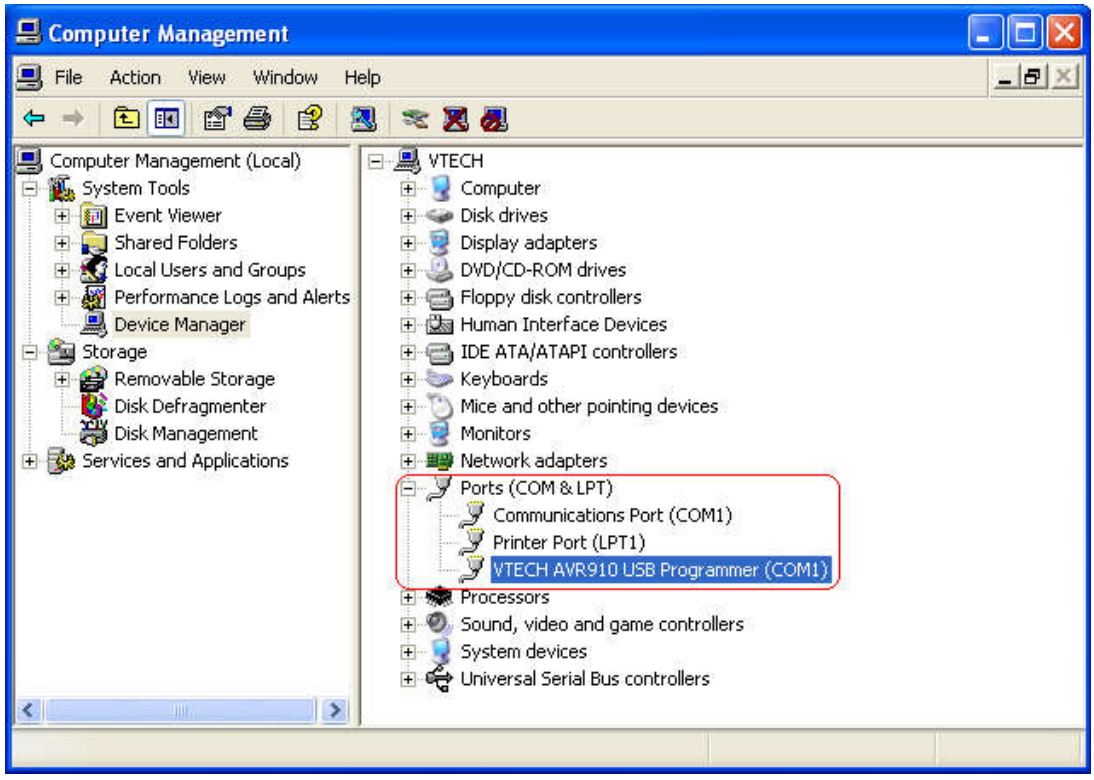
Hình 1. 5. Thay đổi cổng COM giao tiếp

-Trong danh sách COM Port Number, ta chọn một trong các cổng COM 1-4 (để có thể sử dụng với cả 2 phần mềm), tốt nhất là chọn một cổng COM không bị đánh dấu *in use* (đã được sử dụng bởi một phần mềm khác), trong trường hợp tất cả đã bị đánh dấu, ta vẫn có thể lựa chọn cổng *in use*, ở đây minh họa với cổng COM1, chọn **COM1** và nhấn **OK**.

-Ngay lập tức Windows đưa ra thông báo, nhấn ***Yes*** để chấp nhận dùng chung cổng với thiết bị khác, sau đó nhấn ***OK*** trên hộp thoại ***Properties***.

(Lưu ý rằng trong đa số các trường hợp, việc dùng chung cổng này không ảnh hưởng gì đến hoạt động của thiết bị, tuy nhiên cần lưu ý vì nếu cả 2 thiết bị dùng chung cổng cùng được kết nối thì sẽ xảy ra tranh chấp và cả 2 thiết bị sẽ không hoạt động – trường hợp này có thể xảy ra với các thiết bị Bluetooth hoặc một mạch nạp khác sử dụng COM ảo.)

-Rút mạch nạp khỏi cổng USB và cắm lại, theo dõi trong ***Device Manager*** sẽ thấy cổng COM mới đã được gán cho thiết bị:



Hình 1. 6. Kiểm tra mạch nạp trong Device manager

+ Một vài lưu ý nhỏ:

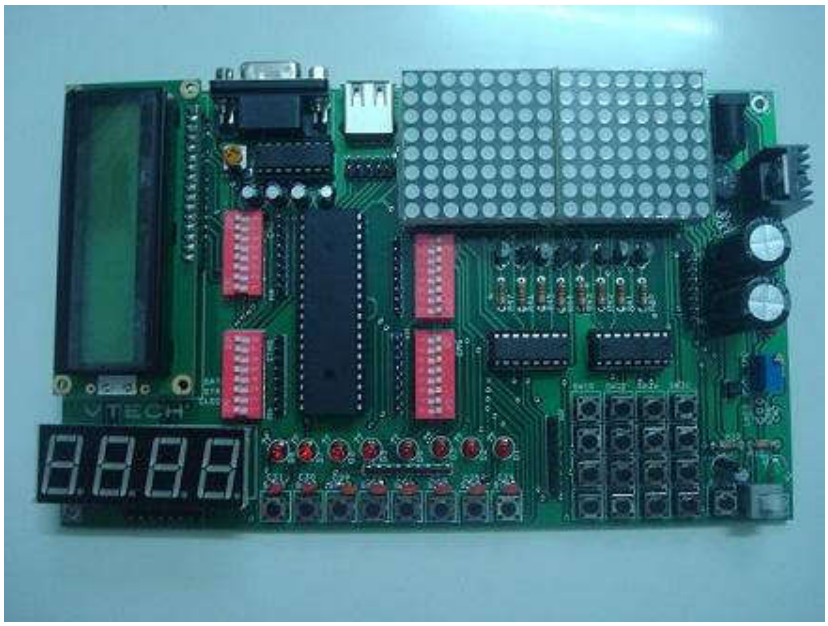
- Quá trình cài đặt trên chỉ có giá trị với một cổng USB (được chọn để kết nối với mạch nạp), máy tính thông thường có khá nhiều cổng USB, nếu cắm mạch nạp vào một cổng khác, quá trình cài đặt sẽ cần làm lại từ đầu, Hệ điều hành sẽ gán một cổng COM mới tương ứng. Vì vậy nên chọn cố định một cổng USB để kết nối với mạch nạp để có thể nắm rõ tên cổng COM tương ứng (với mục đích không cần thay đổi thiết lập khi chạy phần mềm).

- Không nên sử dụng HUB USB với mạch nạp, vì HUB USB có thể không cung cấp đủ nguồn điện cho mạch nạp hoạt động và cung cấp cho mạch được nạp.

- Việc sử dụng dây USB kéo dài có thể ảnh hưởng đến hoạt động của mạch, lưu ý sử dụng các loại dây có chất lượng tốt, có chống nhiễu

1.3.2. KIT thực hành:

Đây là 1 công cụ dùng để thực hành các chức năng của VĐK. Mỗi người học VĐK nên có 1 cái.



Hình 1. 7. KIT thực hành vi điều khiển AVR

1.3.3. Các công cụ phần mềm

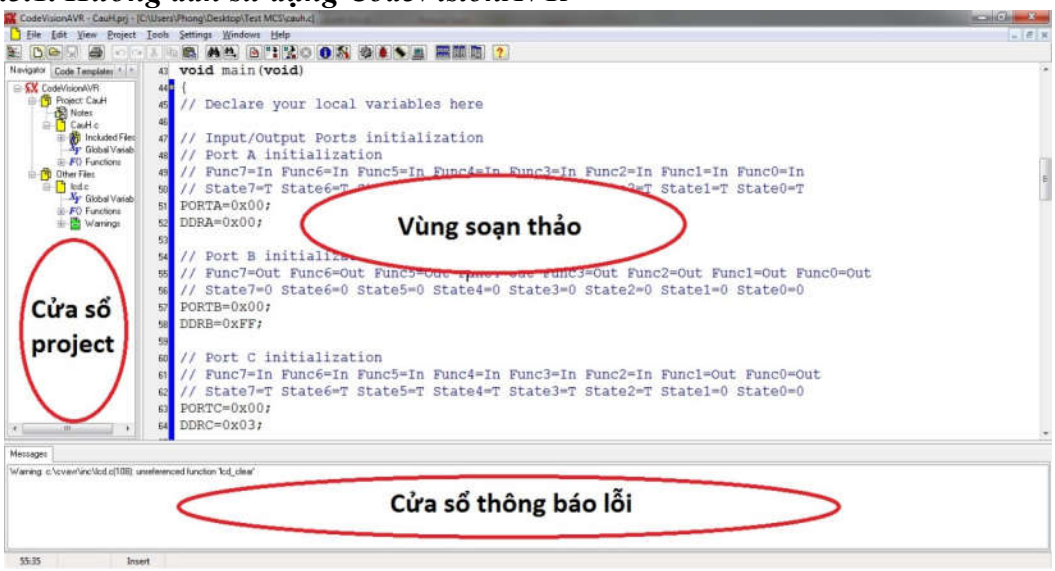
Trình biên dịch: có rất nhiều trình biên dịch bạn có thể sử dụng để biên dịch code của bạn thành file intel hex để nạp vào chip, một số trình dịch quen thuộc có thể kể đến như sau:

- **AvrStudio**: là trình biên dịch ASM chính thức cung cấp bởi Atmel, đây là trình biên dịch hoàn toàn miễn phí và tất nhiên là tốt nhất cho lập trình AVR bằng ASM. Phiên bản hiện tại là 4.12 SP4, bạn có thể download phần mềm AvrStudio tại trang web chính thức của Atmel: http://atmel.com/dyn/products/tools_card.asp?tool_id=2725

- **WinAvr hay Avrgcc**: là bộ chương trình được phát triển bởi gnu, ngôn ngữ sử dụng là C và thường được viết tích hợp với AvrStudio (dùng Avrstudio làm trình biên tập – editor). Đặc biệt bộ biên dịch này cũng miễn phí và đa số nguồn source code C được viết bằng bộ này, vì vậy nó rất lí tưởng cho bạn khi viết các ứng dụng chuyên nghiệp.
- **CodeVisionAvr**: một chương trình bằng ngôn ngữ C rất hay cho AVR, hỗ trợ nhiều thư viện lập trình. Tuy nhiên là chương trình thương mại. Bạn có thể download bản 1.25.9 tại trang web của công ty: www.vtechco.org

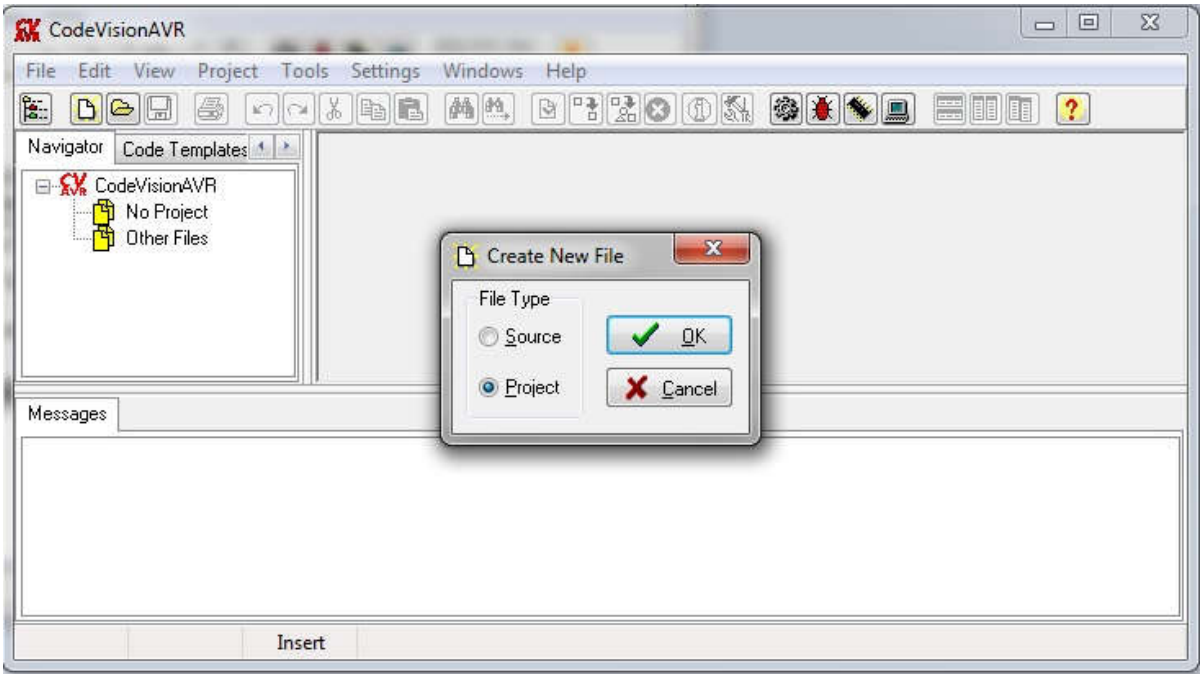
Chương trình nạp (Chip Programmer): đa số các trình biên dịch (AvrStudio, CodeVisionAVR, Bascom…) đều tích hợp sẵn 1 chương trình nạp chip hỗ trợ nhiều loại mạch nạp nên bạn không quá lo lắng. Trong trường hợp khác, bạn có thể sử dụng các chương trình nạp như Icprog hay Ponyprog…là các chương trình nạp miễn phí cho AVR. Việc chọn và sử dụng chương trình nạp sẽ được giới thiệu trong các bài sau.

1.3.3.1. Hướng dẫn sử dụng CodeVisionAVR



Hình 1. 8. Giao diện chương trình CodeVision

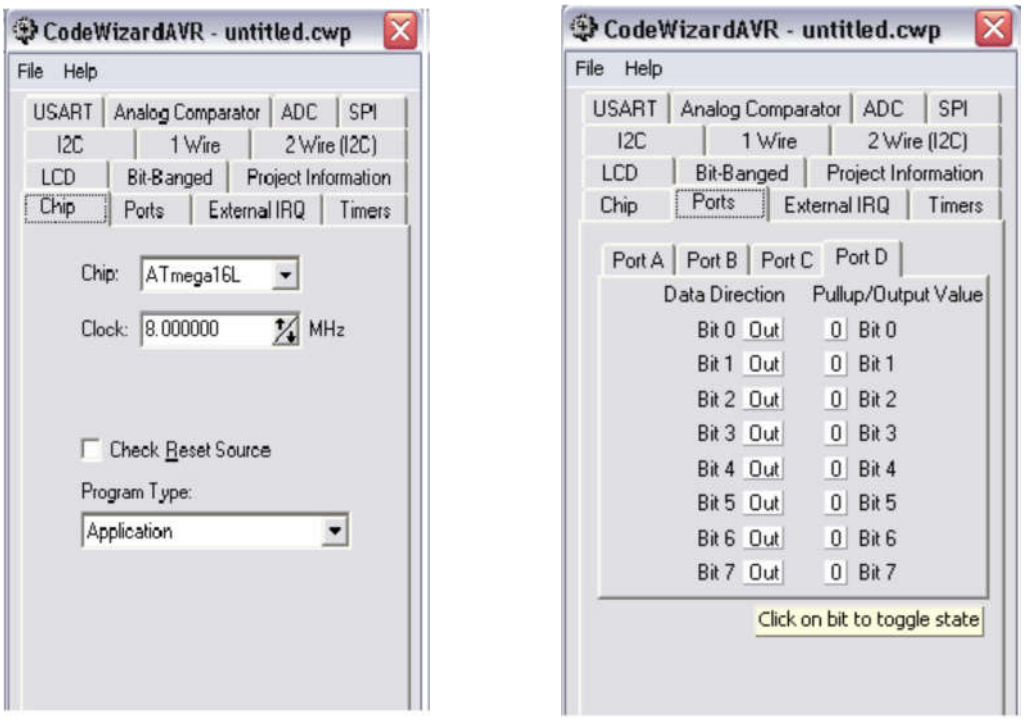
Để tạo Project mới chọn trên menu: **File → New** được như sau:



Hình 1. 9. Tạo project mới trên CodeVision

Chọn Project sau đó click chuột vào OK được cửa sổ hỏi xem có sử dụng CodeWinzard không.

Chọn Yes được cửa sổ CodeWinzardAVR như sau:

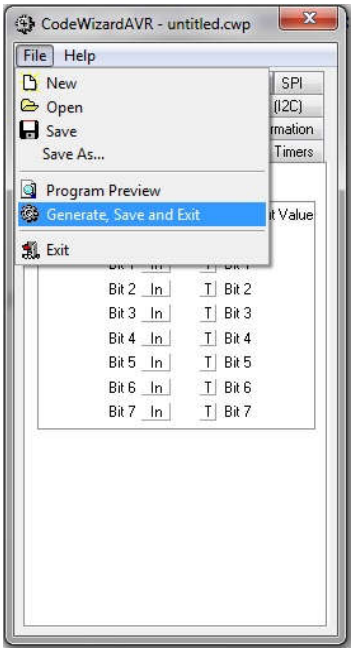


Hình 1. 10. Cửa sổ CodeWizard

Sử dụng chíp AVR nào và thạch anh tần số bao nhiêu ta nhập vào tab Chip.

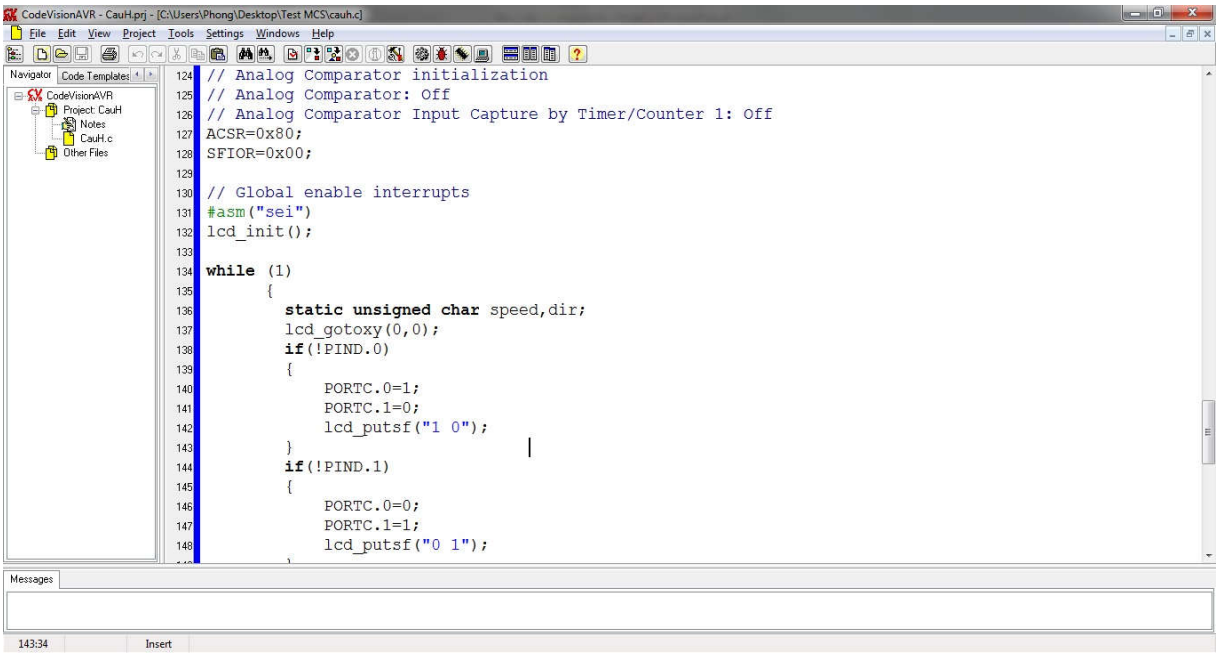
Để khởi tạo cho các cổng IO ta chuyển qua tab **Ports**.

Sau đó chọn **File→Generate, Save and Exit**, lưu lại 3 file có đuôi là .C, .prj, .cwp



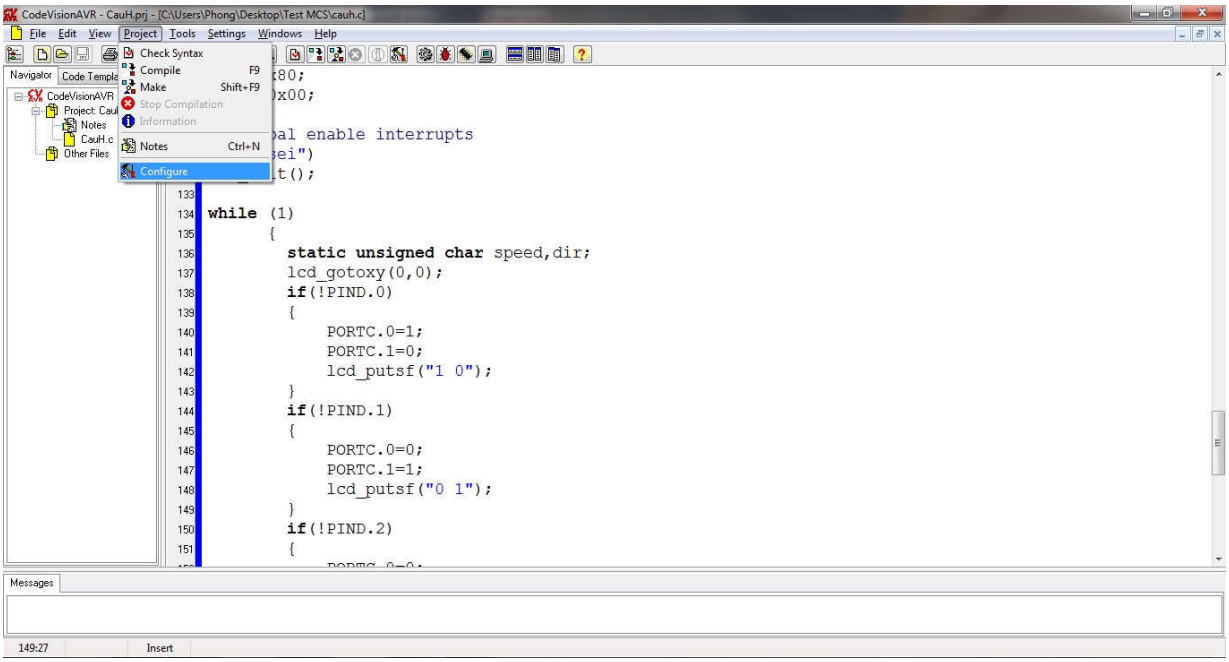
Hình 1. 11. Lưu các file khởi tạo

CodeWizard đã khởi tạo sẵn cho ta giá trị của các thanh ghi theo như khởi tạo bên trên, ta chỉ việc tiếp tục lập trình vào vùng soạn thảo.



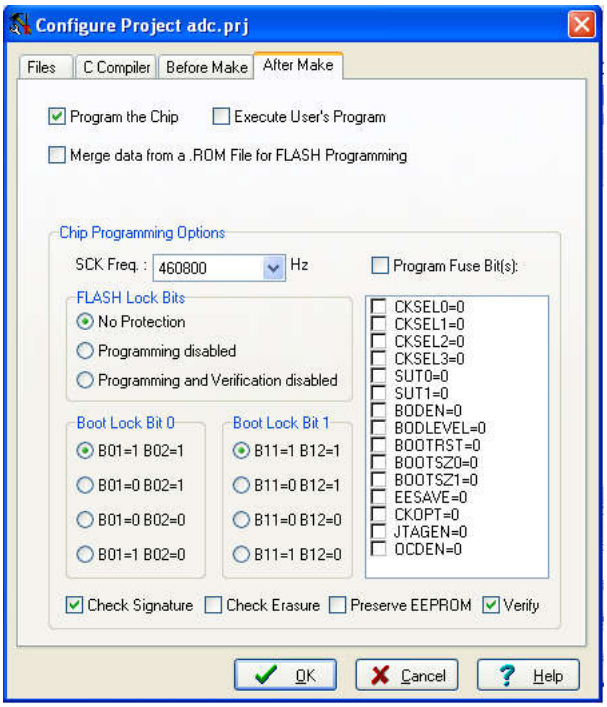
Hình 1. 12. Giao diện soạn thảo chương trình

Sau khi lập trình xong, bây giờ ta cần nạp chương trình vào VĐK. Đầu tiên ta vào **Project→Configure**:

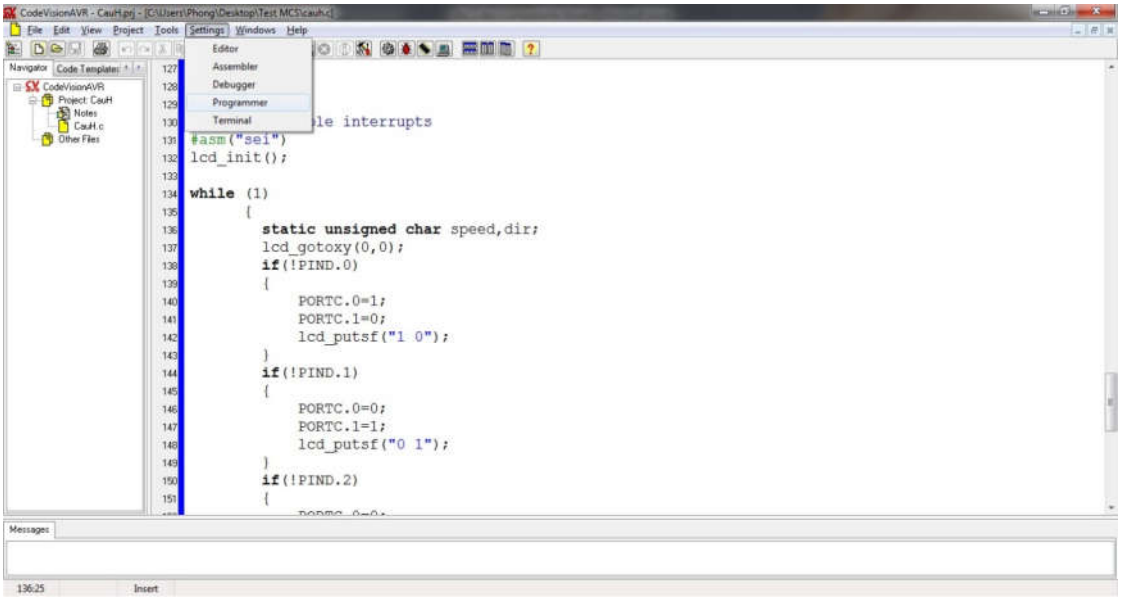


Hình 1. 13. Cấu hình cho project

Sang thẻ **Apter Make**, và tích vào ô **Program the chip:**

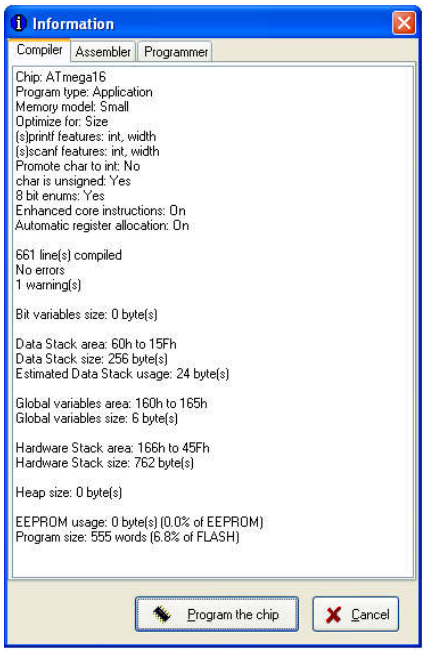


Sau đó, vào **Settings→Programmer** để cấu hình cho mạch nạp:

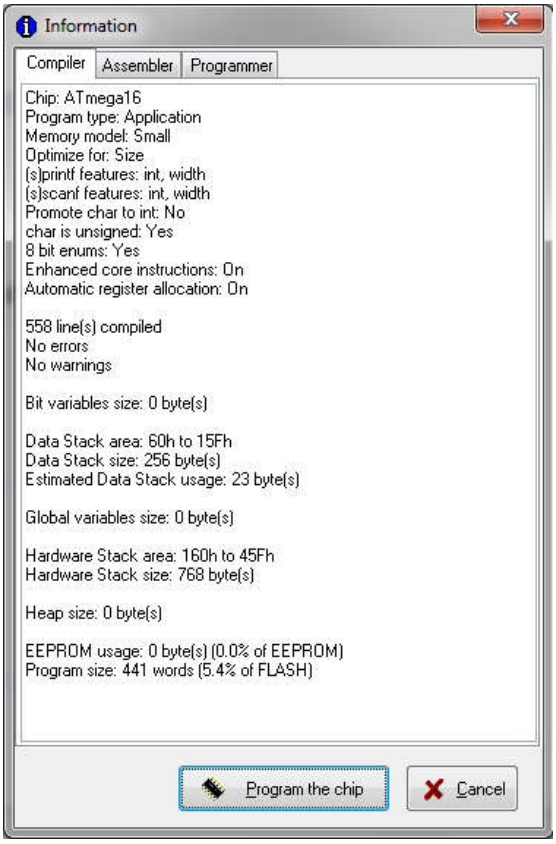


Hình 1. 14. Cấu hình cho mạch nạp

Một hộp thoại hiện ra, dòng đầu tiên ta chọn tên mạch nạp (mặc định là STK500), dòng thứ 2 chọn cổng giao tiếp với máy tính:

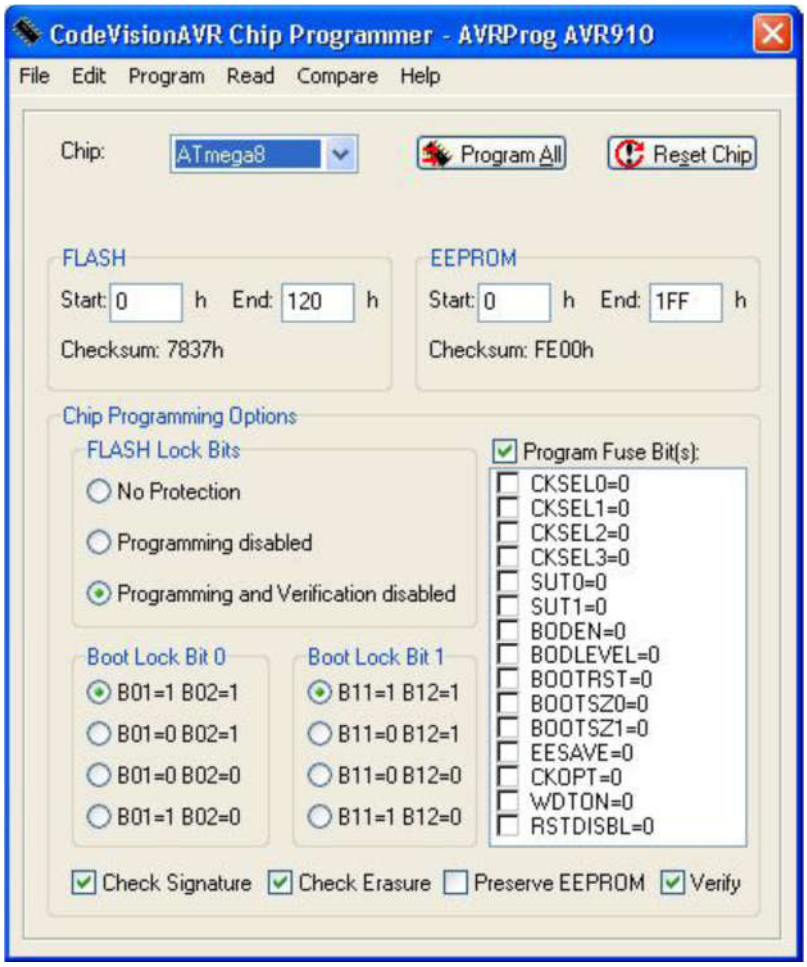


Sau đó: ấn tổ hợp phím **Shift+F9**, chọn **Program the chip** để nạp chương trình vào chip.



Hình 1. 15. Nạp chương trình vào chip

Ở đây, CodeVisonAVR có thể được sử dụng như một chương trình nạp độc lập để nạp một file HEX có sẵn, hoặc sử dụng như một phần tích hợp để nạp chính Project đang được biên dịch bởi CodeVisonAVR, tất cả đều thông qua hộp thoại ***Chip Programmer*** (menu *Tools→Chip Programmer* hoặc tổ hợp phím tắt *Shift-F4*).



Quá trình lập trình cho chíp AVR được chia làm 3 thao tác cơ bản bao gồm **Program** (nạp xuống), **Read** (đọc lên), và **Compare** (so sánh), các thao tác trên được đặt trên hệ thống menu của hộp thoại Programmer, các thao tác trên được áp dụng cho các thành phần sau trên chíp:

- Bộ nhớ chương trình (Flash)

- Bộ nhớ không mất nội dung (EEPROM)

- Các bit lưu cấu hình hoạt động (Fuse bits)

- Các bít lưu cấu hình bảo vệ (Lock bits)

Ngoài ra còn một số mục liên quan như Signature byte, Caliblation Byte… Tham khảo chi tiết trong Datasheet của mỗi loại AVR.

+Trên hộp thoại **Programmer** có các nút cơ bản bao gồm **Program All** và **Reset Chip**.

-Nút **Program All** được sử dụng để nạp tất cả các thành phần đã được thiết lập xuống chíp! Bao gồm Flash, EEPROM, Fuse bits và Lock bits. Cần đặc biệt chú ý điều này, khuyến cáo là không nên sử dụng tới nút **Program All** khi bạn chưa hiểu hết tác dụng của Fuse bits, Lock bits, hơn nữa với nhu cầu thông thường là nạp chương trình vào bộ nhớ Flash, sử dụng nút này sẽ làm kéo dài thời gian lập trình do nạp cả các thành phần không cần thiết.

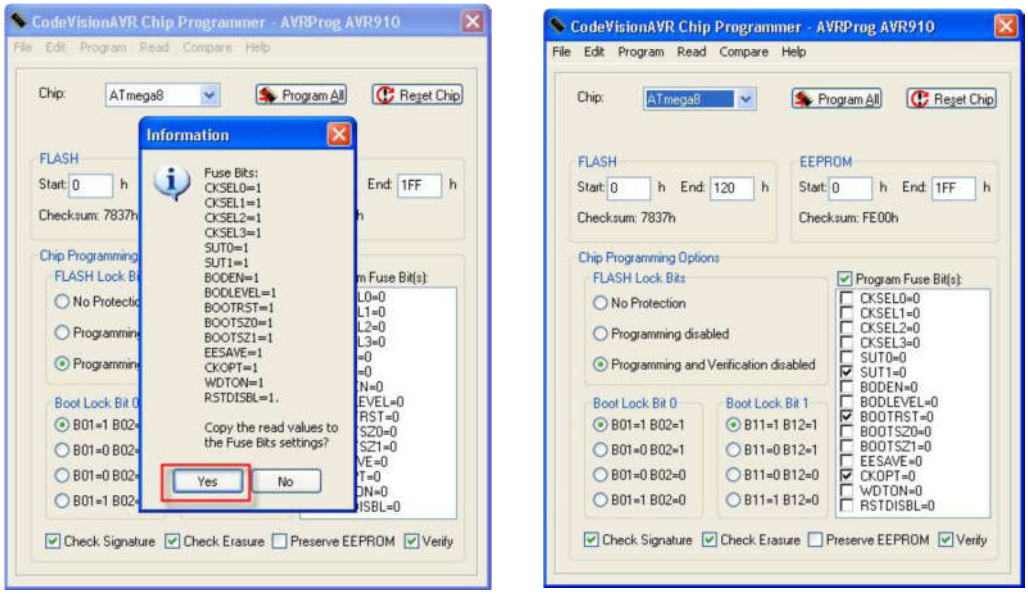
(Nút Program All rất có tác dụng khi cần nạp sản xuất một lượng lớn chíp, giúp làm giảm các thao tác thiết lập).

-Nút **Reset Chip** sẽ kích hoạt tín hiệu Reset trên mạch đích.

+Một thành phần quan trọng khác là hộp thoại chọn chíp, cần chọn đúng loại AVR trong danh sách trước khi tiến hành các thao tác nạp.

+Phía bên phải là phần Fuse bits, liệt kê danh sách các Fuse tương ứng với loại AVR đã chọn (tên và số lượng các fuse này khác nhau với từng dòng AVR). Mặc định CodeVisionAVR để trống các bit này (giá trị 1), điều đó không có nghĩa rằng các Fuse bits thực tế trên Chip cũng có giá trị 1, những người mới sử dụng thường hay nhấn nút **Program All** mà không biết rằng các Fuse bits không được thiết lập đúng. Vô tình thay đổi Fuse bits mặc định trong AVR dẫn tới việc là Chip hoạt động sai lệch mà không rõ nguyên nhân. Vì vậy, cần tham khảo kỹ các tài liệu về Fuse bits dành cho AVR, đặc biệt là Datasheet của dòng AVR đang sử dụng. Cấu hình sai fuse bits sẽ dẫn đến các sai lệnh không lường trước trong quá trình hoạt động của AVR.

Để đọc về các Fuse bits của Chip, vào menu ***Read→Fuse bit(s).***

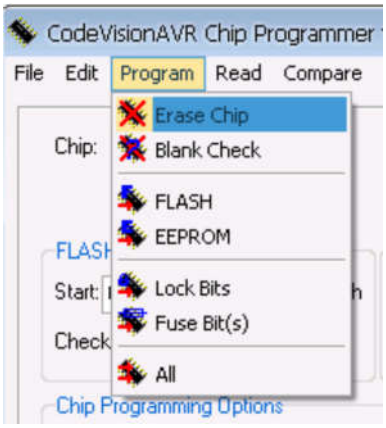


Hình 1. 16. Cấu hình fuse bit

+ Nhấn Yes trên hộp thoại **Information**, các Fuse bits sẽ được sau chép vào phần cấu hình Fuse trong hộp thoại **Programmer**.

+ Trong quá trình sử dụng, nếu sau khi nạp Flash mà xuất hiện thông báo lỗi khi **Verify**:

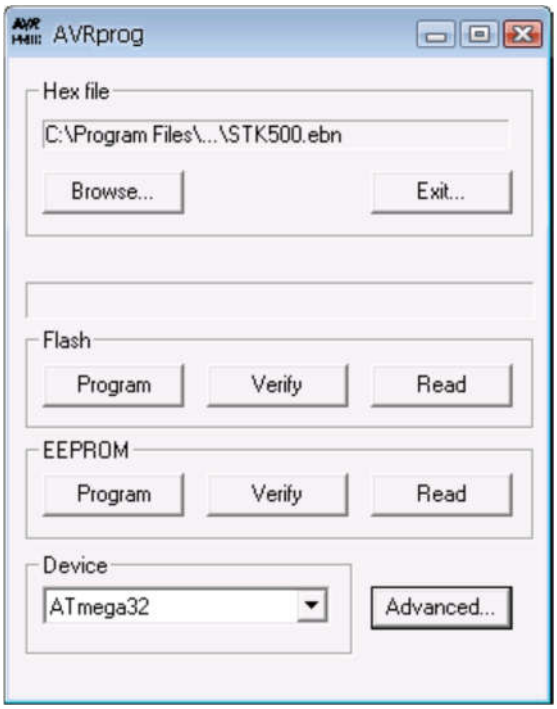
Thì cần xóa toàn bộ Flash bằng cách vào menu **Program** chọn **Erase Chip.**



Sau khi Erase Flash, ta có thể tiến hành nạp bình thường mà không còn xuất hiện thông báo lỗi.

1.3.3.2. Hướng dẫn sử dụng AVR Prog 1.4

+Phần mềm AVRProg 1.4 được tích hợp sẵn khi cài trình biên dịch AVRStudio, phần mềm này có thể hoạt động hoàn toàn độc lập với AVRStudio.



Hình 1. 17. Giao diện AVRProg

Lưu ý rằng ngay khi khởi động AVRProg, phần mềm sẽ dò từ cổng COM1-4 để tìm kiếm mạch nạp. Vì vậy cần lưu ý cắm mạch nạp vào máy tính

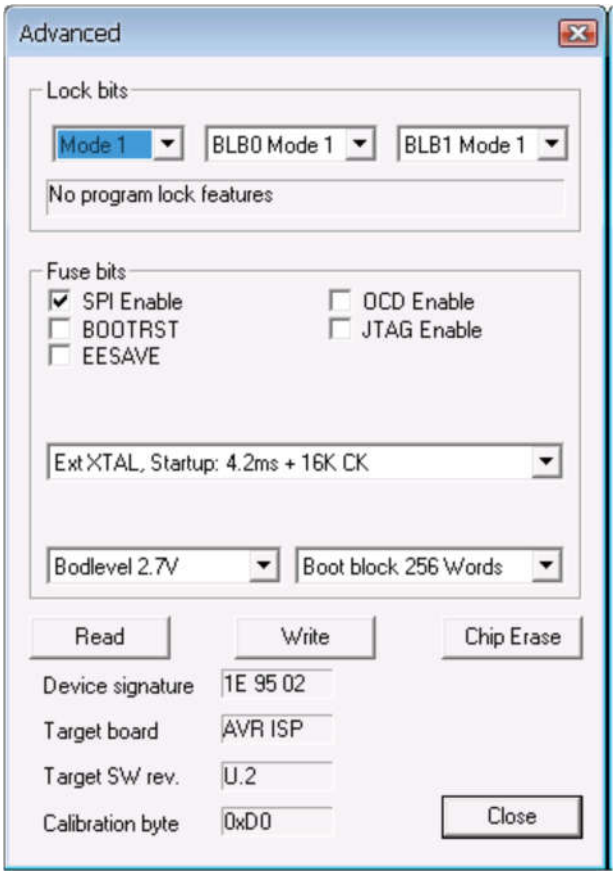
trước khi bật phần mềm. Nếu mạch nạp được thiết lập ở cổng COM lớn hơn 4, hoặc chưa cắm mạch nạp, phần mềm sẽ báo lỗi:

Trong trường hợp này, bạn cần xem lại mục "**Cách gán lại tên cổng COM cho thiết bị!"** ở trên.

Phần mềm AVRProg rất đơn giản khi sử dụng. Trước hết bạn chọn đúng loại AVR đang kết nối trong ô **Device**.

- Nhấn **Browse** để tìm file cần nạp (định dạng HEX, EBN, EEP, A90), Nhấn **Program** trong khung **Flash** nếu muốn nạp vào Flash, hoặc trong khung **EEPROM** nếu muốn nạp vào EEPROM.

- Nếu muốn thay đổi Fuse bit của chíp, click vào **Advanced**, ta sẽ có hộp thoại **Advanced**.

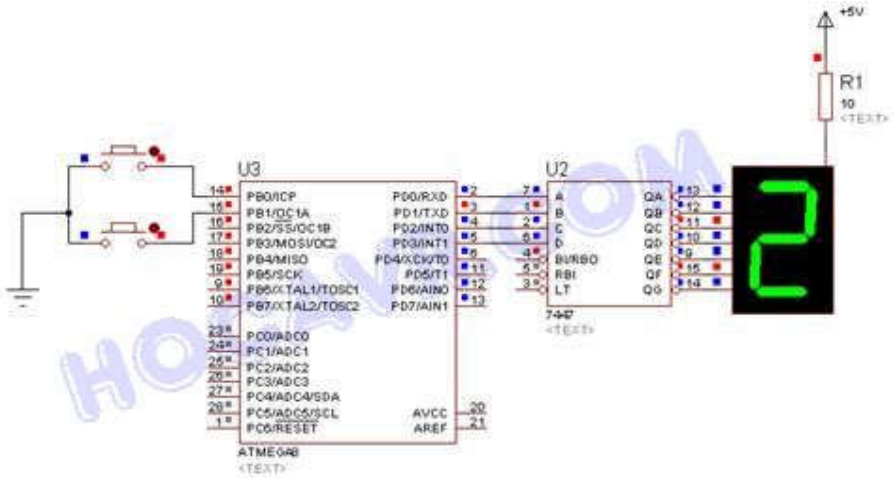


Hình 1. 18. Thay đổi fuse bit của chip

Tại hộp thoại này, ta có thể tự do điều chỉnh cấu hình hoạt động của chíp, tùy loại AVR mà các cấu hình được hiển thị khác nhau. Nêu lưu ý tìm hiểu kỹ về **Fuse bit** của AVR để chắc chắn chíp hoạt động đúng yêu cầu và tránh xảy ra tình trạng cấu hình sai làm chíp không hoạt động

CHƯƠNG 2: ĐIỀU KHIỂN VÀO/RA DỮ LIỆU

2.1. Giới thiệu



Hình 2. 1. Mạch giao tiếp nút bấm điều khiển LED 7 thanh

Một chip ATMega8 được sử dụng như một counter (bộ đếm), có thể dùng để đếm lên và đếm xuống. 2 button trong mạch điện tương ứng với 2 lệnh điều khiển: nhấn button 1 để đếm lên và button 2 để đếm xuống, giá trị đếm nằm trong khoảng từ 0 đến 9. Giá trị đếm được hiển thị trên 1 LED 7 đoạn loại anode chung, chip 7447 là chip chuyên dụng để giải mã LED 7 đoạn, nó giải mã từ giá trị BCD xuất ra bởi ATMega8 sang mã hiển thị cho LED 7 đoạn anode chung.

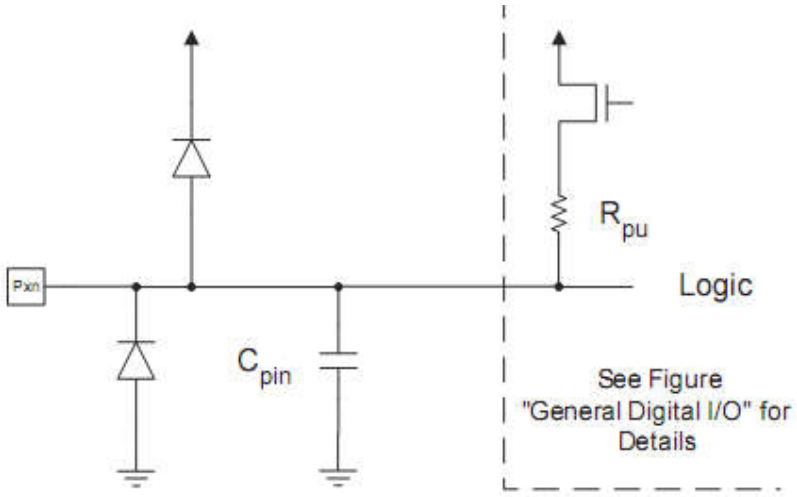
Trong ví này này, chúng ta sử dụng 2 PORT của chip ATMega8, PORTD dùng xuất dữ liệu (số đếm) ra chip 7447 và sau đó hiển thị trên LED 7 đoạn. PORTB dùng như ngõ nhập, tín hiệu từ các button sẽ được chip ATMega8 nhận thông qua 2 chân PB0 và PB1 của PORTB.

Vi điều khiển có nhiệm vụ nhận dữ liệu từ 2 nút bấm, phân biệt nút nào được ấn, sau đó đưa ra lệnh tương ứng: nhấn button 1 thì tăng biến đếm, còn nhấn button 2 thì giảm biến đếm. Sau đó sẽ xuất dữ liệu hiển thị ra 7447.

Như vậy: chức năng IN là chức năng nhận tín hiệu từ thiết bị bên ngoài (nút bấm, cảm biến), còn chức năng OUT là chức năng xuất dữ liệu để điều khiển các thiết bị (LED đơn, LED 7 thanh)

2.2. Điều khiển vào/ra với AVR

Trước hết bạn hãy quan sát mạch điện tương đương của 1 chân trong các PORT xuất nhập của AVR trong hình sau:



Hình 2. 2. Sơ đồ cấu tạo chân vi điều khiển AVR

Trong mạch điện hình trên, các diode và tụ điện chỉ có chức năng bảo vệ chân PORT, nhưng điện trở Rpu (R Pull up) đóng vai trò quan trọng như là điện trở kéo lên khi chân của PORT làm nhiệm vụ nhận tín hiệu (ngõ nhập). Tuy nhiên trong AVR, điện trở kéo lên này không phải luôn kích hoạt.

2.2.1. Các thanh ghi điều khiển vào/ra

Khi xem xét đến các cổng I/O của AVR thì ta cần xét tới 3 thanh ghi bit: ***DDRx, PORTx, PINx.*** Đây là 3 thanh ghi 8 bit và đều có thể truy nhập đến từng bit của nó bằng cách ghi thêm chỉ số vào cuối ***DDRx.n, PORTx.n, PINx.n***

- Thanh ghi ***DDRx****:*

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
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	

Hình 2. 3. Thanh ghi DDRx

Đây là thanh ghi hướng dữ liệu, dùng để thiết lập 1 chân là ***IN***, hay ***OUT***. Khi ghi giá trị logic '0' vào bất kì bit nào của thanh ghi này thì chân đó sẽ trở thành lối vào, còn ghi '1' vào bit đó thì chân đó sẽ trở thành lối ra.

Đối với ví dụ trên: ta cần thiết lập 2 chân ở PORTB làm lối vào: **DDRB.0=DDRB.1=0;** và 4 chân ở PORTD làm lối ra: **DDRD.0 = DDRD.1 = DDRD.2 = DDRD.3 = 1;**

❖ Thanh ghi ***PORTx***:

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
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	

Hình 2. 4. Thanh ghi PORTx

Khi 1 chân được thiết lập làm lối ra thì thanh ghi ***PORTx*** đóng vai trò làm thanh ghi dữ liệu, có nhiệm vụ điều khiển việc xuất dữ liệu ra chân đó. Bit ***PORTx.n*** có thể xuất ra 1 trong 2 mức logic 0 và 1. Khi ***PORTx.n=0*** thì chân tương ứng sẽ xuất ra điện áp 0V, khi ***PORTx.n=1*** thì chân tương ứng sẽ xuất ra điện áp 5V.

Còn khi chân đó được thiết lập làm lối vào thì thanh ghi ***PORTx*** có nhiệm vụ xác lập điện trở kéo lên **Rpu**. Khi **PORTx.n = 0** thì chân tương ứng sẽ không có Rpu, còn khi **PORTx.n = 1** thì chân tương ứng sẽ có Rpu.

Như vậy: nếu thiết lập DDRx = 0x00 và PORTx = 0xFF thì các chân ***PORTx*** là ngõ nhập và được kéo lên bởi 1 điện trở trong chip, nghĩa là tại trạng thái bình thường các chân của ***PORTx*** luôn ở mức cao, muốn kích để thay đổi trạng thái chân này (xuống mức thấp), chúng ta cần nối chân đó trực tiếp với GND, đấy là lý do tại sao các button trong mạch điện của chúng ta có 1 đầu nối với chân của chip còn đầu kia được nối với GND. Đây cũng là ý nghĩa của khái niệm điện trở kéo lên (Pull up resistor) trong kỹ thuật điện tử. Khi dùng 1 chân làm ngõ vào với mục đích là đọc tín hiệu nút bấm ta phải luôn kích hoạt trạng thái Rpu, còn ta sẽ không dùng Rpu trong trường hợp cần đọc tín hiệu ADC.

❖ Thanh ghi **PINx:**

Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Hình 2. 5. Thanh ghi PINx

Chứa dữ liệu nhận về khi cổng tương ứng được thiết lập làm ngõ vào. **PINx** là các cổng chỉ để đọc, dùng để đọc trạng thái logic của các cổng tương ứng.

Ví dụ: khi đọc ra và so sánh PINB.0 == 0 tức là button 1 được ấn, PINB.0==1 tức là button 1 không được ấn

2.2.2. Khởi tạo cho các cổng vào/ra

Mở cửa sổ CodeWizardAVR

❖ Ở tab "**Chip**", lựa chọn chip và tần số thạch anh sử dụng.

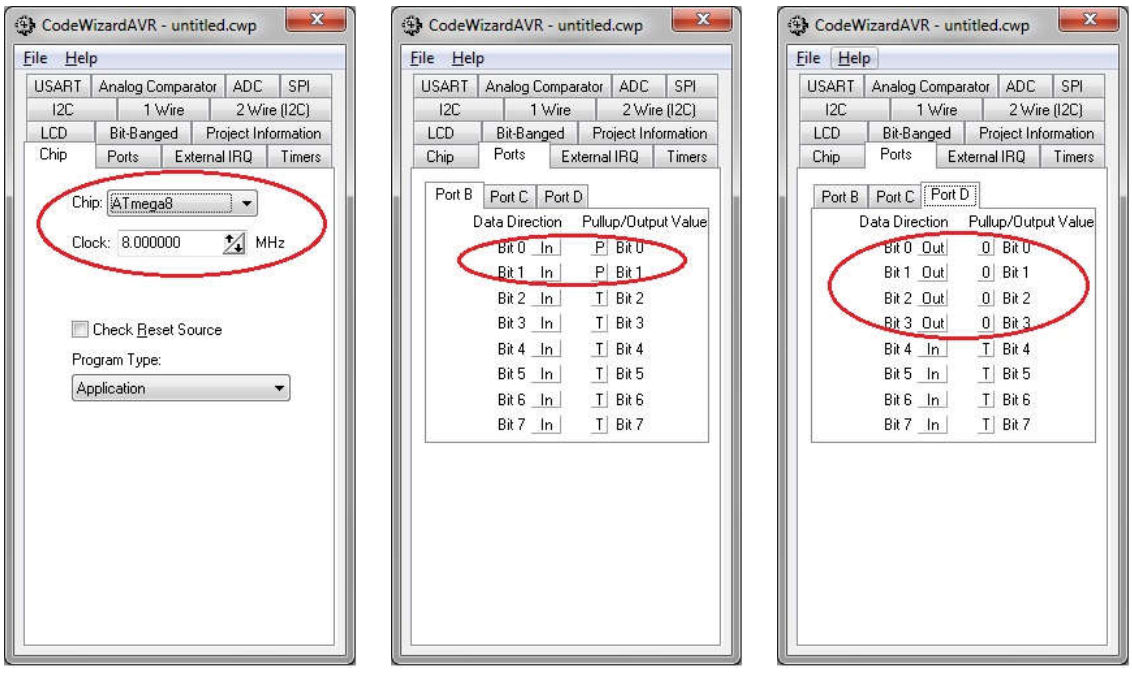
Ví dụ: sử dụng chip ATmega8 và thạch anh có tần số 8MHz

❖ Ở tab "**Ports**":

Cột **Data Direction** dùng để thiết lập hướng dữ liệu cho từng chân (tác động đến thanh ghi DDRx)

Cột **Pullup/Output Value** dùng để thiết lập trạng thái điện trở treo / giá trị xuất mặc định cho từng chân (tác động đến thanh ghi PORTx)

Ví dụ:



Hình 2. 6. Khởi tạo cổng vào/ra

✓ PORTB.0 và PORTB.1 thiết lập làm cổng vào (IN) và kích hoạt trạng thái điện trở treo để làm nhiệm vụ đọc nút bấm.

✓ PORTD.0, PORTD.1, PORTD.2, PORTD.3 được thiết lập làm cổng ra và có giá trị mặc định là 0 (xuất ra 0V).

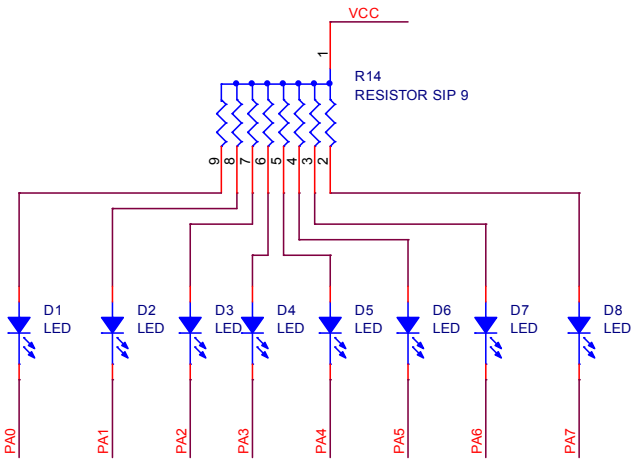
❖ Khởi tạo xong, lưu project ta được đoạn code khởi tạo cho các cổng vào ra như sau:

```
// Input/Output Ports initialization
// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=P State0=P
PORTB=0x03;
DDRB=0x00;

// Port C initialization
// Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTC=0x00;
DDRC=0x00;

// Port D initialization
// Func7=In Func6=In Func5=In Func4=In Func3=Out Func2=Out Func1=Out Func0=Out
// State7=T State6=T State5=T State4=T State3=0 State2=0 State1=0 State0=0
PORTD=0x00;
DDRD=0x0F;
```

2.3. Lập trình ứng dụng

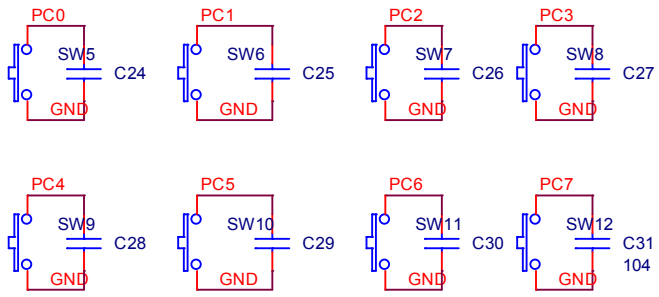
2.3.1. Giao tiếp nút bấm điều khiển LED đơn

Bài tập 1: sử dụng vi điều khiển Atmega16 nhận lệnh từ 8 nút bấm để điều khiển 8 LED đơn tương ứng.



Hình 2. 7. Mạch điều khiển LED đơn

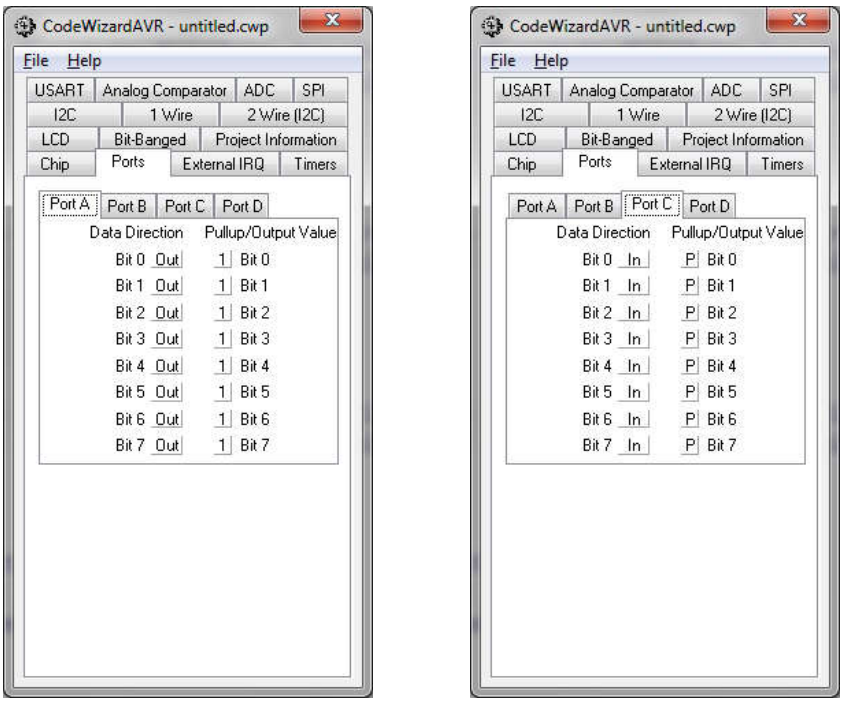
Các led đơn nối vào PORTA của ATMEGA16. Để một LED tắt ta cần đưa mức logic của chân I/O tương ứng của AVR lên mức cao(5V), để LED sáng đưa các chân I/O của AVR xuống mức thấp.



Hình 2. 8. Mạch giao tiếp nút bấm

8 nút bấm được nối vào PORTC của ATMEGA16. Khi một nút được bấm thì chân tương ứng của PORTC sẽ được kéo xuống mức thấp (PINC.n==0).

Dùng CodeWizardAVR để khởi tạo cho các cổng vào ra ở PORTA và PORTC như sau:



Hình 2. 9. Khởi tạo cổng vào/ra

PORTA thiết lập 8 bit làm cổng ra và xuất ra giá trị mặc định là 5V để tất cả các LED đều tắt. PORTC thiết lập làm cổng vào và kích hoạt trạng thái điện trở treo.

Code chương trình:

```
void main(void)
 {
        DDRA=0xff;
        DDRC=0x00;
        PORTC=0xff;
        PORTA=0xff;
        while(1)
        {
                if(PINC.0==0)
                {
                        PORTA=0b11111110;
                }
                if(PINC.1==0)
                {
                        PORTA=0b11111101;
                }
                if(PINC.2==0)
                {
                        PORTA=0b11111011;
                }
                if(PINC.3==0)
                {
                        PORTA=0b11110111;
                }
                if(PINC.4==0)
                {
                        PORTA=0b11101111;
                }
                if(PINC.5==0)
                {
                        PORTA=0b11011111;
                }
                if(PINC.6==0)
                {
                        PORTA=0b10111111;
                }
                if(PINC.7==0)
                {
                        PORTA=0b01111111;
                }
        }
 }
```

Bài tập 2: Điều khiển toàn bộ 8 LED nhấp nháy với tần số 1Hz

Để led nhấp nháy cần tạo ra một khoảng thời gian trễ, dùng hàm delay_ms(). Do đó ta khai báo thêm thư viện **delay.h** vào đầu chương trình:

#include <delay.h>.

Điều khiển LED nhấp nháy bằng cách cho LED sáng, trễ một khoảng thời gian, sau đó cho LED tắt, trễ một khoảng thời gian, và cứ lặp lại như vậy.

Trong hàm main soạn thảo code vào vòng lặp **while(1)** như sau:

```
104 while (1)
105       {
106       // Place your code here
107         PORTA = 0x00;
108         delay_ms(1000);
109         PORTA = 0xFF;
110         delay_ms(1000);
111       };
112 }
```

Sau đó tiến hành dịch và nạp chương trình, và xem kết quả.

Bài tập 3: Nhận lệnh từ nút bấm để điều khiển LED đơn

Ta cần đọc trạng thái logic của 1 bit của PORTC, nếu bit đó ở mức logic thấp thì nút tương ứng được ấn, còn nếu ở mức cao thì nút không được ấn, dựa vào đó có thể đưa ra lệnh điều khiển thích hợp.

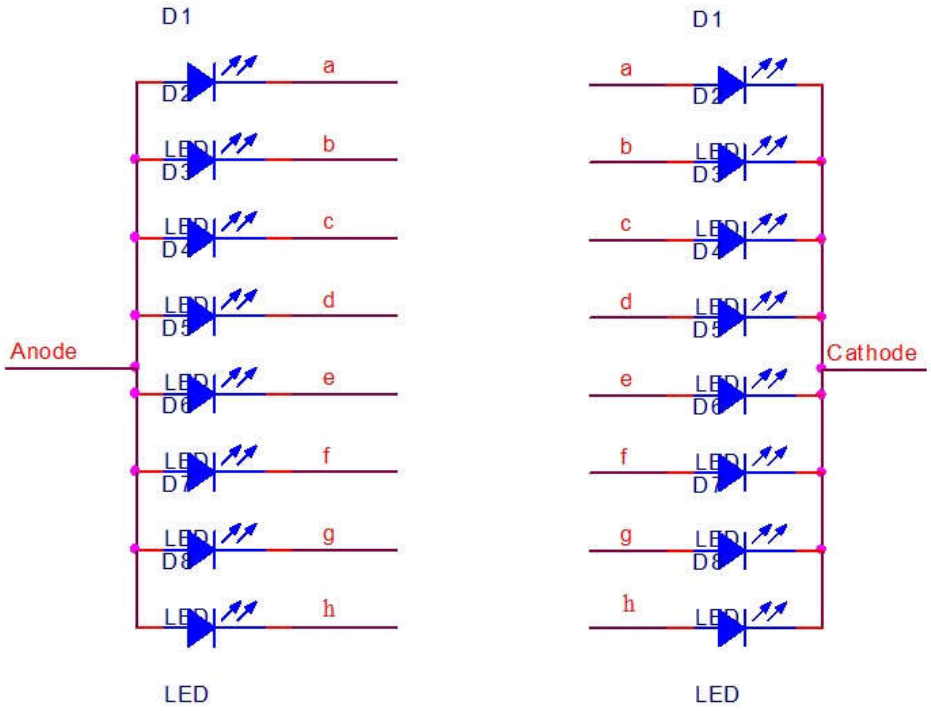
Ví dụ: ấn 1 nút thì LED tương ứng sáng, ta soạn code như sau:

```
142 while (1)
143       {
144       // Place your code here
145         if(!PINC.0) {PORTA=~0x01;}
146         else if(!PINC.1) {PORTA=~0x02;}
147         else if(!PINC.2) {PORTA=~0x04;}
148         else if(!PINC.3) {PORTA=~0x08;}
149         else if(!PINC.4) {PORTA=~0x10;}
150         else if(!PINC.5) {PORTA=~0x20;}
151         else if(!PINC.6) {PORTA=~0x40;}
152         else if(!PINC.7) {PORTA=~0x80;}
153       };
154 }
```

2.3.2. Điều khiển LED 7 thanh

2.3.2.1. Giới thiệu LED 7 thanh

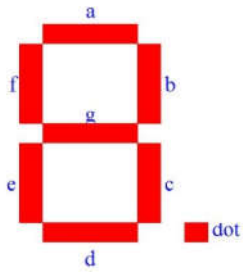
LED 7 thanh gồm 8 LED đơn được nối chung một đầu, trong đó có 7 thanh LED được sắp xếp thành hình số 8, và một LED được xếp ở vị trí dấu chấm. Cách sắp xếp như vậy giúp cho LED 7 thanh có thể hiển thị được tất cả các chữ số từ 0 đến 9, số thập phân, và một vài chữ cái đơn giản.



Hình 2. 10. Cấu tạo LED 7 thanh Anode chung và Cathode chung

✓ Phân loại LED 7 thanh: dựa vào chân nối chung của LED 7 thanh, LED 7 thanh được chia làm 2 loại là LED 7 thanh **Anode chung** và LED 7 thanh **Cathode chung**.

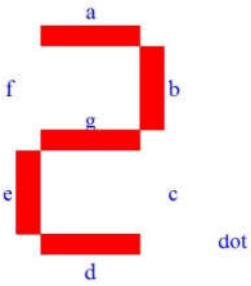
✓ Các thanh của LED 7 thanh được đặt tên là a, b, c, d, e, f, g, dot (hoặc h). Sơ đồ bố trí các thanh được thể hiện trong H 4.



Sơ đồ bố trí các thanh

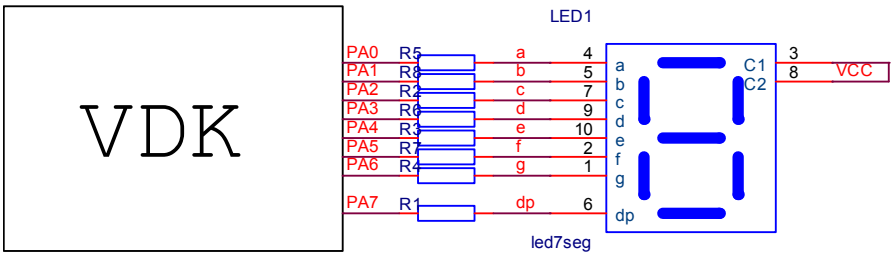
✓ Để LED 7 thanh hiển thị một số, ta thực hiện thao tác bật, tắt các LED tương ứng sao cho các thanh LED sáng ghép được thành số mong muốn.

✓ Ví dụ: để hiển thị số 2 ta thực hiện bật các thanh a, b, d, e, g; tắt thanh c, f và dot.



Cách thiết lập hiển thị số 2

2.3.2.2. Điều khiển một LED 7 thanh



Hình 2. 11. Kết nối LED 7 thanh với vi điều khiển

✓ Điều khiển LED 7 thanh giống như việc điều khiển LED đơn, với chú ý là ta phải điều khiển việc bật, tắt 8 LED cùng lúc.

✓ Đối với LED 7 thanh loại anode chung thì chân chung sẽ được nối lên nguồn (VCC), 8 chân cathode của 8 LED sẽ được nối vào 8 chân của vi điều khiển thông qua điện trở. Để điều khiển một thanh LED sáng ta thực hiện đưa chân vi điều khiển tương ứng xuống mức 0V. Điều khiển cả 8 LED cùng lúc bằng cách xuất dữ liệu đồng thời ra 1 PORT của vi điều khiển.

Để việc xuất dữ liệu được thuận lợi ta sử dụng bảng mã hiển thị LED 7 thanh

Bảng 2. 1. Bảng mã hiển thị LED 7 thanh Anode chung

Số hiển thị	**Mã hiển thị LED 7 thanh dạng nhị phân**								**Mã hiển thị LED 7 thanh dạng thập lục phân**
	h	g	e	f	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	1	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**

✓ Như vậy để LED 7 thanh hiển thị số 0 ta chỉ việc đưa lệnh như sau:

PORTA=0b11000000; *// (viết dưới dạng nhị phân)*

hoặc **PORTA=0xC0;** *// (viết dưới dạng thập lục phân).*

Thực hiện tương tự đối với các số còn lại.

2.3.2.3. Điều khiển nhiều LED 7 thanh

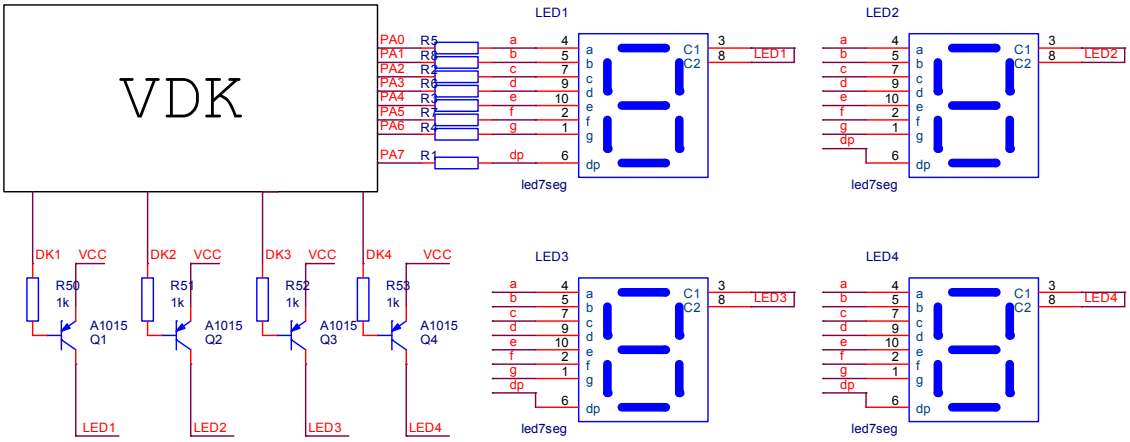
✓ Khi số lượng LED tăng lên, nếu thực hiện phương pháp điều khiển từng LED riêng biệt thì sẽ tốn rất nhiều chân của vi điều khiển. Ví dụ: nếu có 4 LED 7 thanh thì sẽ cần 4 * 8 = 32 chân vi điều khiển. Điều này sẽ gây lãng phí.

✓ Như vậy để điều khiển nhiều LED 7 thanh thì có một phương pháp khác, đó chính là phương pháp quét. Mỗi LED sẽ chỉ được sáng và hiển thị một số trong một khoảng thời gian ngắn, tuy nhiên do hiện tượng lưu ảnh trên võng mạc thì mắt người sẽ cảm nhận được tất cả các LED sáng cùng lúc.

✓ Ví dụ: để hiển thị số 8490 thì ta thực hiện lần lượt các bước sau:

Đầu tiên điều khiển LED1 sáng và hiển thị số 8; LED2, LED3, LED4 tắt. Trễ một khoảng thời gian.	
Tiếp theo điều khiển LED2 sáng và hiển thị số 4; LED1, LED3, LED4 tắt. Trễ một khoảng thời gian.	
Tiếp theo điều khiển LED3 sáng và hiển thị số 9; LED1, LED2, LED4 tắt. Trễ một khoảng thời gian.	
Tiếp theo điều khiển LED4 sáng và hiển thị số 0; LED1, LED3, LED4 tắt. Trễ một khoảng thời gian.	
Như vậy mặc dù mỗi LED chỉ sáng trong một khoảng thời gian nhưng ta vẫn cảm nhận được số 8490 sáng đồng thời.	

Với việc tại một thời điểm chỉ có một LED sáng thì ta có thể nối chung các thanh a, b, c, d, e, f, g, h của tất cả các LED lại với nhau; và cần thêm các chân để điều khiển việc bật, tắt cho từng LED. Như vậy chân Anode chung của từng LED sẽ không nối trực tiếp lên VCC nữa mà sẽ được điều khiển bởi vi điều khiển. Sơ đồ kết nối phần cứng được thể hiện như H 7; trong đó 8 chân điều khiển 8 LED được nối vào 8 chân của một PORT của vi điều khiển, 4 chân anode chung được điều khiển thông qua khối khuyếch đại transistor với các chân điều khiển là DK1, DK2, DK3, DK4 được nối vào 4 chân I/O khác của vi điều khiển.



Hình 2. 12. Sơ đồ kết nối 4 LED 7 thanh

✓ Ví dụ: lập trình hiển thị số 8490:

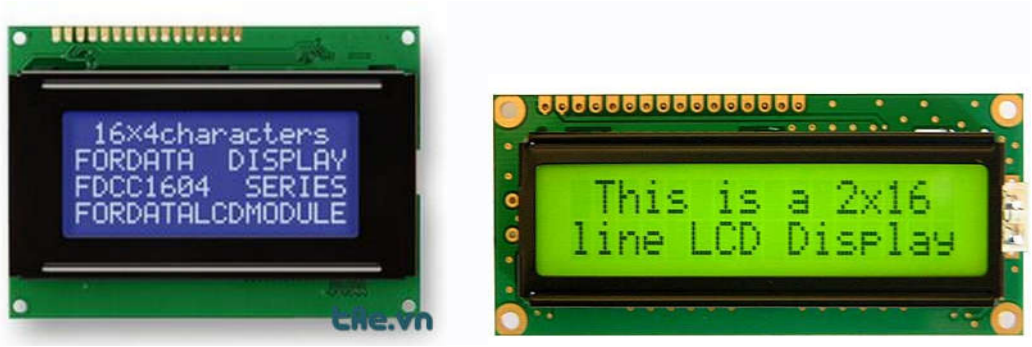
```
DK1=0;DK2=1;DK3=1;DK4=1;        // Điều khiển LED 1 sáng
PORTA=0x80;                     // Xuất dữ liệu hiển thị số 8
delay_ms(5);                    // Trễ 5(ms)
DK1=1;DK2=0;DK3=1;DK4=1;        // Điều khiển LED 2 sáng
PORTA=0x99;                     // Xuất dữ liệu hiển thị số 4
delay_ms(5);                    // Trễ 5(ms)
DK1=1;DK2=1;DK3=0;DK4=1;        // Điều khiển LED 3 sáng
PORTA=0x90;                     // Xuất dữ liệu hiển thị số 9
delay_ms(5);                    // Trễ 5(ms)
```

DK1=1;DK2=1;DK3=1;DK4=0;	*// Điều khiển LED 4 sáng*
PORTA=0xC0;	*// Xuất dữ liệu hiển thị số 0*
delay_ms(5);	*// Trễ 5(ms)*

2.3.3. Điều khiển hiển thị LCD

2.3.3.1. Giới thiệu LCD

Giống như led 7 thanh, LCD là một thiết bị ngoại vi dụng để giao tiếp với người dùng, so với led 7 thanh thì LCD có ưu điểm là hiển thị được tất cả các ký tự trong bảng mã ASCII, trong khi đó led 7 thanh chỉ hiển thị được một số ký tự, nhưng LCD lại có nhược điểm là giá thành cao và khoảng cách nhìn gần. LCD được sử dụng rất phổ biến đảm nhận vai trò hiển thị thông tin được lập trình sẵn hoặc các thông tin đã qua xử lý của bộ điều khiển hoặc vi xử lý, giúp dễ dàng giao tiếp, điều khiển cũng như giám sát hoạt động của hệ thống.



Hình 2. 13. Hình ảnh LCD thực tế

Mô tả của các chân LCD1602A:

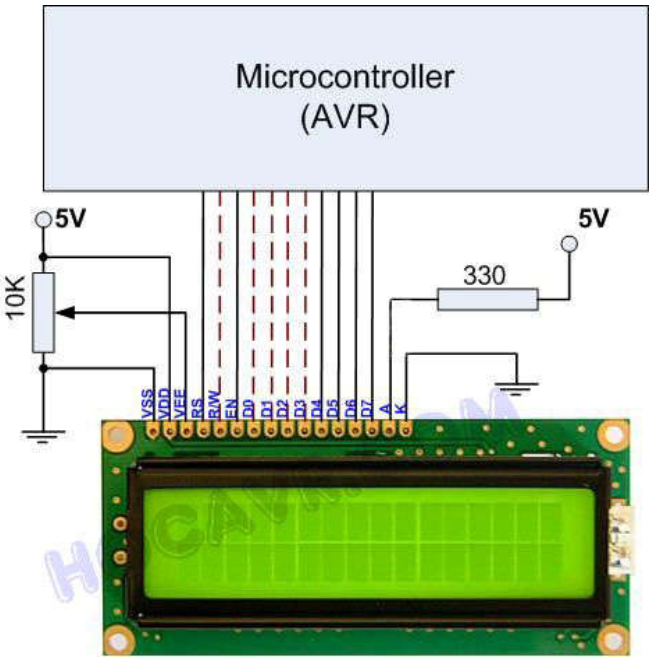
Bảng 2. 2. Ý nghĩa các chân của LCD

Chân	Ký hiệu	Mô tả
1	Vss	Chân nối đất cho LCD, khi thiết kế mạch ta nối chân này với GND của mạch điều khiển
2	VDD	Chân cấp nguồn cho LCD, khi thiết kế mạch ta nối chân này với VCC=5V của mạch điều khiển
3	VEE	Điều chỉnh độ tương phản của LCD. Tùy theo loại LCD mà có độ tương phản khác nhau, điều chỉnh điện áp chân này từ 0-5V

4	RS	Chân chọn thanh ghi (Register select). Nối chân RS với logic "0" (GND) hoặc logic "1" (VCC) để chọn thanh ghi. + Logic "0": Bus DB0-DB7 sẽ nối với thanh ghi lệnh IR của LCD (ở chế độ "ghi" - write) hoặc nối với bộ đếm địa chỉ của LCD (ở chế độ "đọc" - read) + Logic "1": Bus DB0-DB7 sẽ nối với thanh ghi dữ liệu DR bên trong LCD.
5	R/W	Chân chọn chế độ đọc/ghi (Read/Write). 0: LCD hoạt động ở chế độ ghi 1: LCD hoạt động ở chế độ đọc.
6	E	Chân cho phép (Enable). Sau khi các tín hiệu được đặt lên bus DB0-DB7, các lệnh chỉ được chấp nhận khi có 1 xung cho phép của chân E. + Ở chế độ ghi: Dữ liệu ở bus sẽ được LCD chuyển vào(chấp nhận) thanh ghi bên trong nó khi phát hiện một xung (high-to-low transition) của tín hiệu chân E. + Ở chế độ đọc: Dữ liệu sẽ được LCD xuất ra DB0-DB7 khi phát hiện cạnh lên (low-to-high transition) ở chân E và được LCD giữ ở bus đến khi nào chân E xuống mức thấp.
7-14	DB0-DB7	Tám đường của bus dữ liệu dùng để trao đổi thông tin với CPU. Có 2 chế độ sử dụng 8 đường bus này : + Chế độ 8 bit: Dữ liệu được truyền trên cả 8 đường, với bit MSB là bit DB7. + Chế độ 4 bit: Dữ liệu được truyền trên 4 đường từ DB4 tới DB7, bit MSB là DB7
15	Anode	Nguồn dương cho đèn nền
16	Cathode	GND cho đèn nền

2.3.3.2. Kết nối LCD với Vi điều khiển

Sơ đồ kết nối LCD với VĐK như hình vẽ



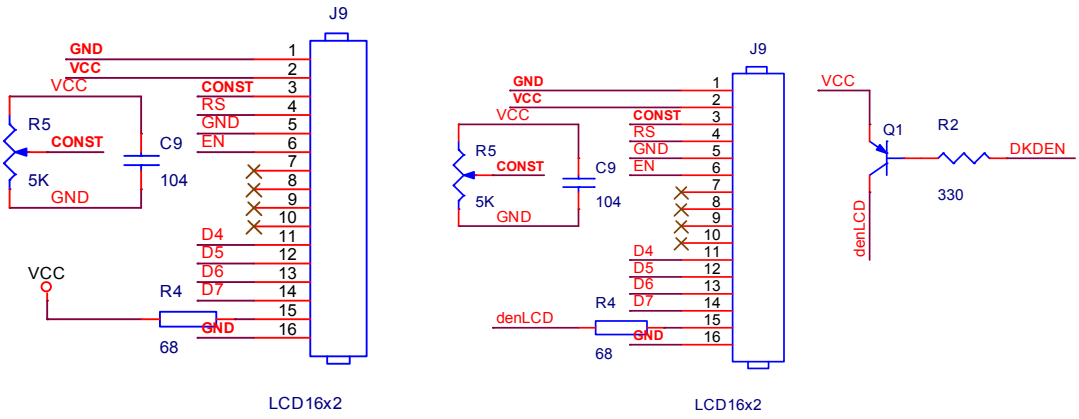
Hình 2. 14. Kết nối LCD với vi điều khiển

Để điều chỉnh được điện áp chân Vee thì sử dụng một biến trở để phân áp. LCD có thể giao tiếp với VĐK theo 2 chế độ:

- Chế độ 8 bit: cả 8 bit dữ liệu của LCD sẽ nối vào 8 bit của một PORT VĐK

- Chế độ 4 bit: 4 bit dữ liệu cao của LCD (D4-D7) sẽ nối vào 4 bit cao của một PORT VĐK

Chế độ 4 bit hạn chế về tốc độ giao tiếp nhưng tiết kiệm được chân VĐK. Các ứng dụng hiển thị LCD thì không cần tốc độ cao nên thường lựa chọn chế độ 4 bit để tiết kiệm chân cho VĐK.

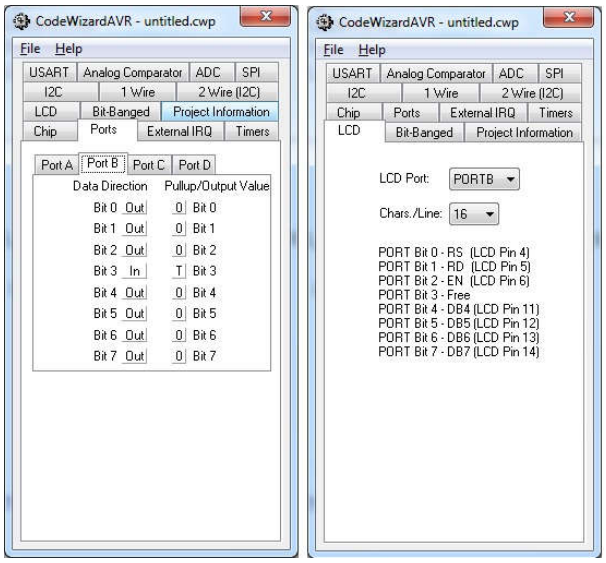


Hình 2. 15. Sơ đồ mạch giao tiếp LCD

2.3.3.3. Điều khiển hiển thị LCD

a) Sử dụng thư viện có sẵn của Codevision

Mở cửa sổ CodeWizard. LCD nối vào các chân của PORT nào thì khởi tạo các chân đó làm cổng ra. Ở Tab LCD, chọn PORT mà LCD nối vào, tiếp theo ở dưới sẽ chú thích thứ tự từng chân của LCD kết nối với chân nào của VĐK, phần cứng cần thiết lập tương ứng.



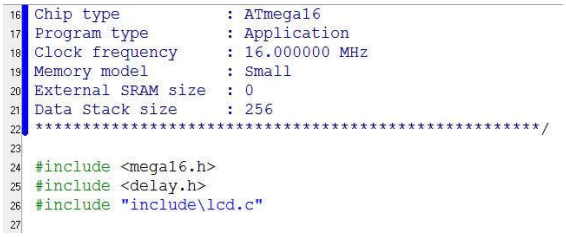
Hình 2. 16. Khởi tạo giao tiếp LCD bằng CodeWizard

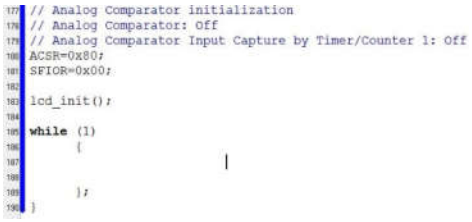
b) Sử dụng thư viện LCD tự xây dựng

Khi sử dụng thư viện lcd tự tạo thì ở cửa sổ CodeWizard không cần khởi tạo LCD, chỉ cần khởi tạo cổng điều khiển LCD làm cổng ra.

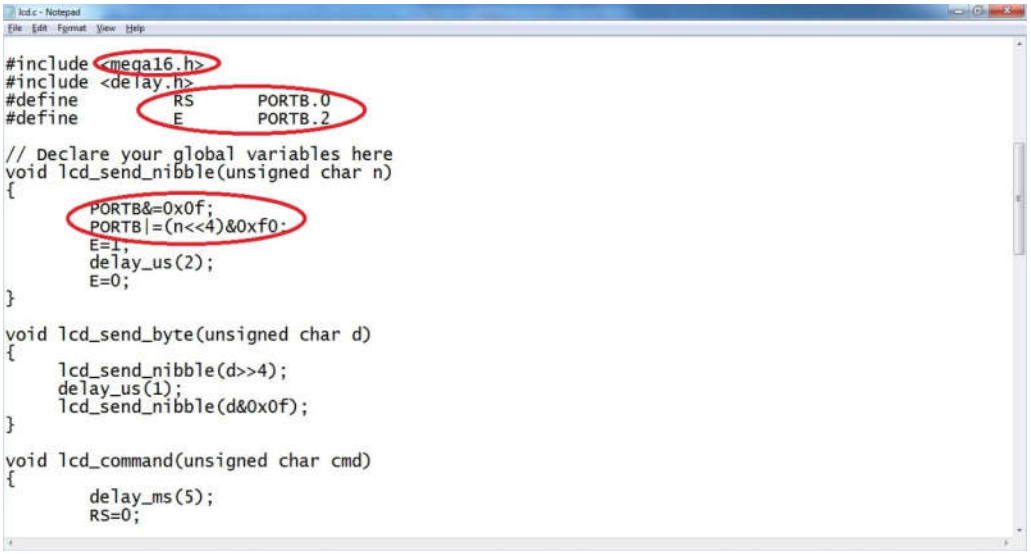
Copy thư viện "***lcd.c***" vào cùng thư mục với project, sau đó include vào đầu chương trình.

Tiếp theo trong chương trình **main**(), trước vòng lặp **while**(1) cần gọi hàm khởi tạo cho LCD để có thể sử dụng được các hàm điều khiển LCD





Chú ý: Khi thay đổi chip khác, hoặc trên phần cứng LCD nối vào PORT khác thì cần chỉnh sửa lại tên chip, các chân RS, E, PORT điều khiển trong thư viện lcd.c, còn lại các hàm giữ nguyên.



c) Các hàm điều khiển LCD

Trong thư viện LCD, có rất nhiều hàm, trong đó ta cần quan tâm tới các hàm sau:

+ **void** lcd_init(**unsigned char** number); // thư viện của CodeWizard

void lcd_init(**void**); // thư viện tự tạo

Đây là hàm khởi tạo cho LCD, được gọi 1 lần duy nhất trước khi thực hiện các hàm hiển thị lên LCD

+ **void** lcd_clear(**void**);

Hàm xóa màn hình và đưa con trỏ về tọa độ (0,0)

+ **void** lcd_gotoxy(**unsigned char** x, **unsigned char** y);

Hàm di chuyển con trỏ tới tọa độ (x,y)

- Với thư viện của CodeWizard thì x là cột (từ 0-15), y là hàng (từ 0-1)

- Với thư viện tự tạo thì x là hàng (từ 0-1), y là cột (từ 0-15)

+ **void** lcd_putchar(**char** c);

Hàm hiển thị một kí tự c lên màn hình. Có 2 cách truyền đối số:

- Truyền mã ASCII của kí tự c:

VD: lcd_putchar(61); // hiển thị chữ a

- Truyền trực tiếp kí tự đó đặt trong cặp dấu nháy đơn ‘’

VD: lcd_putchar(‘X’); // hiển thị chữ X

+ **void** lcd_putsf(**char flash** *str);

Hàm hiển thị một chuỗi kí tự lên màn hình. Chuỗi kí tự được đặt trong cặp dấu nháy kép “”

VD: lcd_putsf(“Hello”);

+ **void** lcd_putnum(**unsigned int** number);

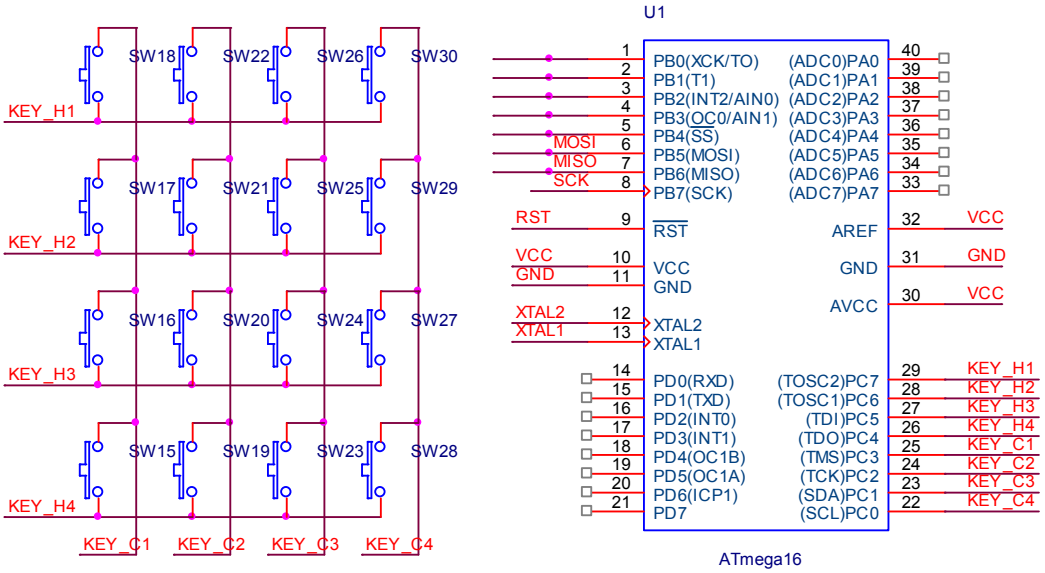
Hàm hiển thị một số lên LCD. Thư viện của Codewizard không hỗ trợ hàm này, do đó phải tự viết thêm.

VD: lcd_putnum(12345);

2.3.4. Giao tiếp với nút bấm ma trận

2.3.4.1. Kết nối nút bấm ma trận với vi điều khiển

Các nút bấm được ghép theo kiểu ma trận trong trường hợp cần dùng nhiều nút bấm, giúp tiết kiệm số chân cho VĐK.



Hình 2. 17. Kết nối nút bấm ma trận với vi điều khiển

Các nút bấm trên cùng một hàng thì nối chung chân số 1 với nhau, các nút bấm trên cùng một cột thì nối chung chân số 2 với nhau. Như vậy, đối với phím ma trận 4x4 thì chỉ cần sử dụng 4+4=8 chân của vi điều khiển là có thể lập trình giao tiếp được với 16 nút bấm.

2.3.4.2. Thuật toán lập trình

Khác với nút bấm đơn, để lập trình giao tiếp với phím ma trận, luôn luôn phải thực hiện hàm quét phím ma trận. Các bước xây dựng hàm quét phím như sau:

B1: Thiết lập các hàng là cổng ra, các cột là cổng vào và kích hoạt trở treo (nếu thực hiện quét theo hàng)

B2: **define** tên cho các hàng, các cột để thuận tiện sử dụng

```
#define     H1    PORTC.0
#define     H2    PORTC.1
```

```
#define     H3     PORTC.2
#define     H4     PORTC.3
#define     C1     PINC.4
#define     C2     PINC.5
#define     C3     PINC.6
#define     C4     PINC.7
```

B3: Khai báo một mảng toàn cục sw[16] có 16 phần tử để lưu trữ các giá trị mã hóa cho từng nút. Giá trị mã hóa là '0' nếu không ấn nút, là '1' nếu ấn nút.

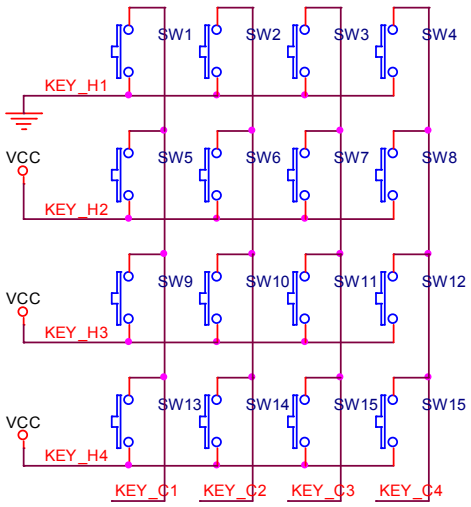
Đầu hàm quét nút bấm cần xóa hết giá trị 16 phần tử mảng về 0

B4: Thực hiện lệnh xuất H1 = 0; H2 = H3 = H4 = 1;

+ Khi ấn một nút trên hàng 1 thì chân của cột tương ứng sẽ được kéo xuống GND, đọc ra mức logic '0'

VD: ấn nút 1 thì chân C1 xuống GND, ấn nút 2 thì chân C2 xuống GND

+ Khi ấn một nút trên hàng 2 hoặc 3 hoặc 4 thì các chân cột không thay đổi (vẫn nối VCC, đọc ra mức logic '1')



Lập trình: Thực hiện lệnh xuất dữ liệu ra các hàng, đọc các chân cột tương ứng, nếu thấy chân cột nào xuống logic '0' thì mã hóa cho phần tử tương ứng của mảng bằng '1'.

H1 = 0; H2 = H3 = H4 = 1;
if(!C1)
{
	if(!C3)
	{

sw[0] = 1;
}
if(!C2)
{
sw[1] = 1;
}
	sw[2] = 1;
	}
	if(!C4)
	{
	sw[3] = 1;
	}

B5: Thực hiện lệnh xuất H2 = 0; H1 = H3 = H4 = 1;

Lập trình tương tự như **B4**

B6: Thực hiện lệnh xuất H3 = 0; H1 = H2 = H4 = 1;

Lập trình tương tự như **B4**

B7: Thực hiện lệnh xuất H4 = 0; H1 = H2 = H3 = 1;

Lập trình tương tự như **B4**

B8: Giao tiếp với nút bấm trong chương trình chính

+ Gọi hàm quét phím sw_scan() trước khi thực hiện các lệnh với nút bấm
+ So sánh từng phần tử của mảng, nếu một phần tử bằng '1' tức là nút tương ứng được ấn, khi đó sẽ thực hiện lệnh tương ứng
	while(1)
	{
	sw_scan();
	if(sw[0] == 1)
	{
	// Lệnh cho nút 1
	}
	if(sw[1] == 1)
	{
	// Lệnh cho nút 2
	}
	…
	}

CHƯƠNG 3: HOẠT ĐỘNG CỦA NGẮT NGOÀI

3.1. Khái niệm

Ngắt (**Interrupts**), là một "tín hiệu khẩn cấp" gởi đến bộ xử lí, yêu cầu bộ xử lí tạm ngừng tức khắc các hoạt động hiện tại để "nhảy" đến một nơi khác thực hiện một nhiệm vụ "khẩn cấp" nào đó, nhiệm vụ này gọi là trình phục vụ ngắt – **isr** (**interrupt service routine**). Sau khi kết thúc nhiệm vụ trong **isr**, bộ xử lí sẽ quay về thực hiện tiếp các nhiệm vụ còn dang dở

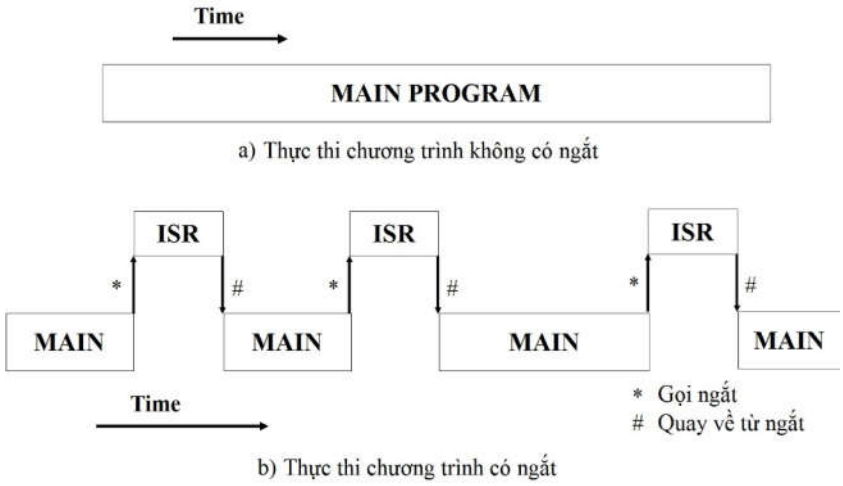
Ngắt có mức độ ưu tiên xử lí cao nhất, ngắt thường được dùng để xử lí các sự kiện bất ngờ nhưng không tốn quá nhiều thời gian.

Ví dụ: Trong giờ học trên lớp, ta đang học bài, có chuông điện thoại hoặc có bạn gọi, ta phải dừng hoạt động học bài lại để trả lời điện thoại hoặc ra gặp bạn.

Sự kiện điện thoại reo chuông, hay bạn bè gọi được gọi là sự kiện ngắt, việc ta trả lời điện thoại hay ra gặp bạn là chương trình phục vụ ngắt. Việc đang học bài được xem là chương trình chính.

Ngắt được thực hiện khi và chỉ khi cài đặt cho phép nó. Như trong ví dụ trên, nếu sự kiện ngắt- điện thoại reo xảy ra, nếu giáo viên và bản thân cho phép mình trả lời điện thoại khi đang học bài thì khi có điện thoại ta mới nghe.

Giản đồ thực thi chương trình:



Hình 3. 1. Giản đồ thực thi một chương trình

Các thuật ngữ dùng cho xử lý ngắt trong vi điều khiển:

- Nguồn ngắt: là nguyên nhân gây ra ngắt.

Ví dụ: điện thoại gọi, bạn gọi

- Sự kiện ngắt: sự kiện khi nguồn ngắt xảy ra

- Chương trình con phục vụ ngắt: là chương trình vi điều khiển xử lý khi có sự kiện ngắt xảy ra do người lập trình lập trình ra

Ví dụ: trả lời hoặc chạy ra khỏi phòng gặp bạn

- Vecto ngắt: tức địa chỉ nơi vi điều khiển nhảy tới sau khi lưu địa chỉ trả về

- Bit cho phép ngắt: bit cho phép vi điều khiển chạy chương trình con phục vụ ngắt khi có sự kiện ngắt xảy ra

Ví dụ: khi có sự kiện ngắt - điện thoại gọi, nếu ta cho phép mình nghe điên thoại, đồng thời thầy giáo cho phép thì ta mới nghe điện thoại (cho chương trình con phục vụ ngắt hoạt động).

- Cờ ngắt: là bit phản ánh trạng thái của sự kiện ngắt. Mỗi ngắt có một bit cờ. Khi bit cờ này bằng 1 nghĩa là sự kiện ngắt tương ứng với cờ đó xảy ra

Phân biệt "ngắt" và "gọi chương trình con":

- Giống nhau: Khi xảy ra điều kiện tương ứng thì CPU sẽ tạm dừng chương trình chính đang thực thi để thực thi một chương trình khác (chương trình con / chương trình xử lý ngắt) rồi sau đó (sau khi xử lý xong chương trình con / chương trình xử lý ngắt) thì CPU sẽ quay về để thực thi tiếp tục chương trình chính đang bị tạm dừng.

- Khác nhau:

	Ngắt	Chương trình con
Thời điểm xảy ra sự kiện	Không biết trước (hay xảy ra không đồng bộ với chương trình chính).	Biết trước (hay xảy ra đồng bộ với chương trình chính)
Nguyên nhân dẫn đến sự kiện	Do các tín hiệu điều khiển từ Timer, Serial port và bên ngoài chip.	Do lệnh gọi chương trình con (ACALL, LCALL)

Điểm mạnh của phương pháp ngắt:

- Bộ vi điều khiển có thể phục vụ được rất nhiều thiết bị. Mỗi thiết bị có thể nhận được sự chú ý của bộ vi điều khiển dựa trên mức ưu tiên được gán cho nó.

- Trong phương pháp ngắt thì bộ vi điều khiển còn có thể che (bỏ qua) một yêu cầu phục vụ của thiết bị.

- Không lãng phí thời gian cho các thiết bị không cần phục vụ. (Phương pháp thăm dò làm lãng phí thời gian của bộ vi điều khiển bằng cách hỏi dò từng thiết bị kể cả khi chúng không cần phục vụ)

Chú ý:

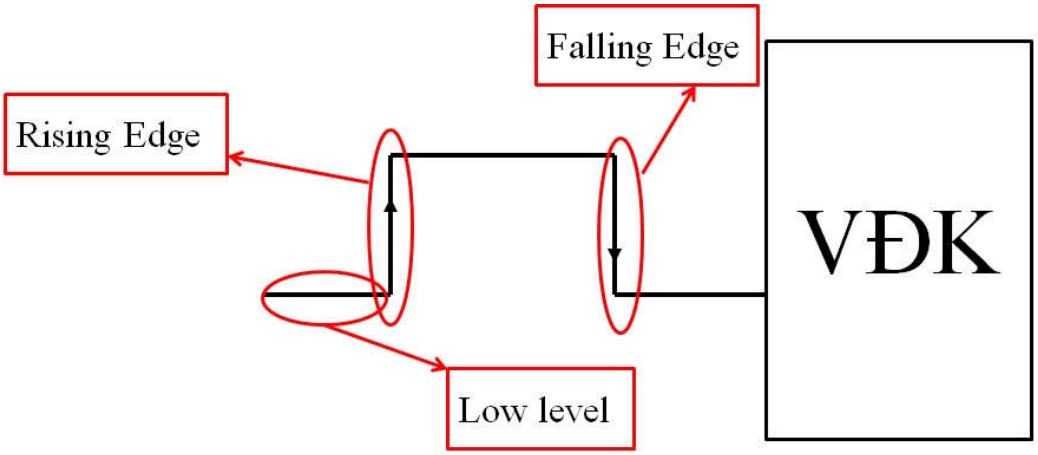
- Chương trình phục vụ trong ngắt chỉ nên viết các lệnh thực hiện nhanh, tốn ít thời gian thực thi, không dùng delay cũng như vòng lặp

- Không thể ghi vào bộ nhớ EEPROM trong chương trình phục vụ ngắt.

Các loại ngắt trong vi điều khiển AVR:

- Ngắt ngoài
- Ngắt tràn Timer
- Ngắt so sánh Timer
- Ngắt truyền thông nối tiếp

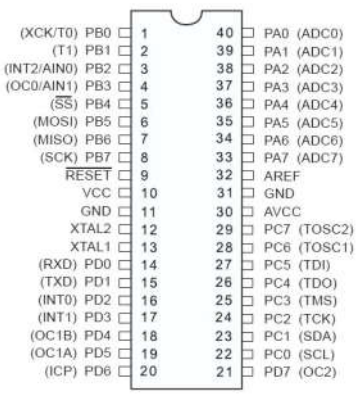
Các nguồn ngắt ngoài:



1. Rising Edge: sườn lên
2. Falling Edge: sườn xuống
3. Low level: mức điện áp thấp
4. Any change: bất kỳ sự thay đổi nào của chân

3.2. Thanh ghi điều khiển ngắt

Atmega16 có 3 ngắt ngoài: **INT0**, **INT1**, **INT2** tương ứng với các chân số 16 (PD2), 17(PD3), và 3(PB2)



- **Thanh ghi GICR (General Interrupt Control Register)**

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	

Hình 3. 2. Thanh ghi GICR

Đây là 1 thanh ghi 8 bit, trong đó 3 bit cao (bit 5, 6 và 7) là 3 bit điều khiển ngắt.

INT1: bit cho phép ngắt ngoài 1

INT0: bit cho phép ngắt ngoài 0

INT2: bit cho phép ngắt ngoài 2

Thiết lập bit tương ứng lên 1 là cho phép ngắt hoạt động, thiết lập 0 là không cho phép ngắt hoạt động

- **Thanh ghi MCUCR: (MCU Control Register)**

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	

Hình 3. 3. Thanh ghi MCUCR

Đây là thanh ghi 8 bit, trong đó 4 bit thấp dùng để điều khiển trạng thái ngắt (Interrupt Sense Control)

ISC11, ISC10 (dùng cho INT1)

ISC01, ISC00 (dùng cho INT0)

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

Thanh ghi MCUSCR: (MCU Control and Status Register)

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					

Hình 3. 4. Thanh ghi MCUSCR

Đây là thanh ghi 8 bit. Bit thứ 6 - **ISC2** dùng để điều khiển trạng thái ngắt cho **INT2**.

ISC2	Description
0	The falling Edge of INT2 generates as interrupt request
1	The rising Edge of INT2 generates as interrupt request

- **Thanh ghi GIFR (General Interrupt Flag Register)**

Bit	7	6	5	4	3	2	1	0	
	INTF1	INTF0	INTF2	–	–	–	–	–	GIFR
Read/Write	R/W	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Hình 3. 5. Thanh ghi GIFR

Đây là 1 thanh ghi 8 bit, trong đó 3 bit cao là 3 bit cờ ngắt.

INTF1: cờ ngắt ngoài 1

INTF0: cờ ngắt ngoài 0

INTF2: cờ ngắt ngoài 2

Khi xuất hiện một sự kiện ngắt thì cờ ngắt tương ứng sẽ tự động được set lên 1. Nếu ngắt tương ứng được thực thi thì cờ ngắt sẽ tự động được xóa về 0

- **Tổng kết:**

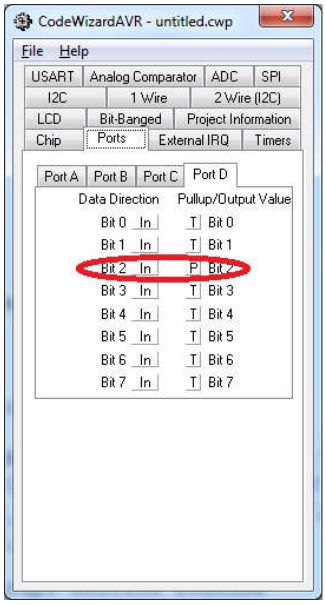
Thanh ghi	Ý nghĩa
GICR	Cho phép ngắt hoạt động hay không
MCUCR MCUSCR	Thiết lập chế độ hoạt động của ngắt
GIFR	Đọc trạng thái cờ ngắt

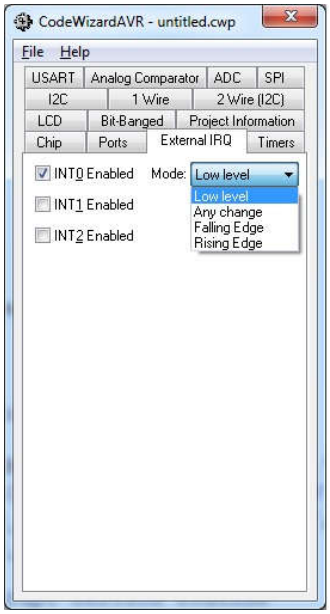
3.3. Lập trình ngắt ngoài

3.3.1. Khởi tạo

Sử dụng ngắt ngoài nào thì thiết lập chân tương ứng làm cổng vào và kích hoạt trở treo

Chọn tab **External IRQ.** Tích vào ô Enable tương ứng để cho phép ngắt hoạt động. Chọn chế độ hoạt động của ngắt ngoài ở phần Mode





Sau khi khởi tạo xong, các thanh ghi sẽ được gán giá trị như sau:

Chú ý: cần phải có lệnh **#asm("sei")** ở cuối cùng, đây là lệnh cho phép ngắt toàn cục

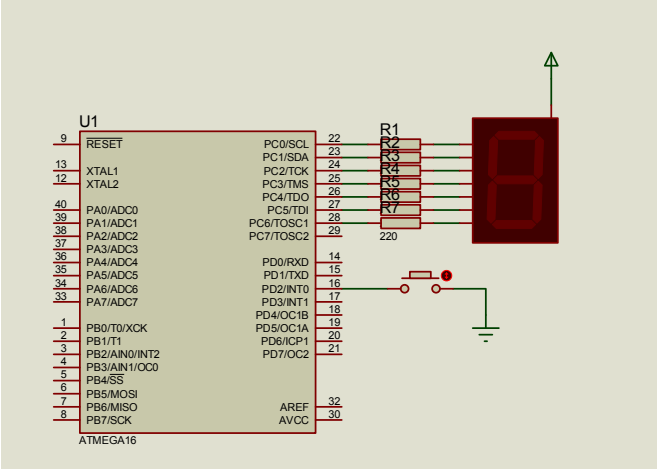
```
// External Interrupt(s) initialization
// INT0: On
// INT0 Mode: Falling Edge
// INT1: Off
// INT2: Off
GICR|=0x40;
MCUCR=0x02;
MCUCSR=0x00;
GIFR=0x40;

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

Lập trình cho chương trình con phục vụ ngắt ở hàm sau:

```
// External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{
    if(number < 9)
    {
        number++;
    }
}
```

3.3.2. Ví dụ:

Điều khiển hiển thị một LED 7 thanh. Thiết lập ngắt ngoài 0 hoạt động ở chế độ Falling Edge. Mỗi khi có sự kiện ngắt xảy ra thì số hiển thị trên LED 7 thanh tăng lên 1 đơn vị



1. Thiết lập cổng vào ra:

PC điều khiển LED → thiết lập làm cổng ra

DDRC = 0xFF;

PORTC = 0xFF;

PD2 nhận tín hiệu từ nút bấm, thiết lập làm cổng vào và kích hoạt Pull-up:

DDRD = 0x00;

PORTD |= 0x04;

2. Thiết lập thanh ghi cho phép ngắt ngoài 0:

GICR |= 0x40;

3. Thiết lập chế độ hoạt động cho ngắt ngoài 0:

MCUCR |= 0x02;

4. Cho phép ngắt toàn cục

#asm("sei")

5. Lập trình trong chương trình con phục vụ ngắt

```
26  // External Interrupt 0 service routine
27  interrupt [EXT_INT0] void ext_int0_isr(void)
28  {
29      if(number < 9)
30      {
31          number++;
32      }
33  }
```

3.4. Bài tập thực hành

- Điều khiển hiển thị một LED 7 thanh. Thiết lập ngắt ngoài 0 hoạt động ở chế độ **Rising Edge**. Mỗi khi có sự kiện ngắt xảy ra thì số hiển thị trên LED 7 thanh tăng lên 1 đơn vị.

- Điều khiển hiển thị một LED 7 thanh. Thiết lập ngắt ngoài 0 hoạt động ở chế độ **Any change**. Mỗi khi có sự kiện ngắt xảy ra thì số hiển thị trên LED 7 thanh tăng lên 1 đơn vị.

- Điều khiển hiển thị một LED 7 thanh. Thiết lập ngắt ngoài 0 hoạt động ở chế độ **Falling Edge,** ngắt ngoài 1 hoạt động ở chế độ **Falling Edge**. Mỗi khi có sự kiện ngắt

0 xảy ra thì số hiển thị trên LED 7 thanh tăng lên 1 đơn vị, sự kiện ngắt 1 xảy ra thì số hiển thị trên LED 7 thanh giảm 1 đơn vị

- Điều khiển hiển thị 4 LED 7 thanh bằng phương pháp quét. Thiết lập 2 ngắt ngoài 0 và 1 hoạt động ở chế độ Falling Edge. Mỗi khi có sự kiện ngắt 0 xảy ra thì số hiển thị trên LED 7 thanh tăng lên 1 đơn vị, sự kiện ngắt 1 xảy ra thì số hiển thị trên LED 7 thanh giảm 1 đơn vị

- Điều khiển 8 LED đơn nhấp nháy xen kẽ nhau. Thiết lập 2 ngắt ngoài 0 và 1 hoạt động. Sử dụng 2 nút bấm: 1 nút làm tăng tần số nhấp nháy, 1 nút làm giảm tần số nhấp nháy

CHƯƠNG 4: HOẠT ĐỘNG CỦA BỘ CHUYỂN ĐỔI ADC

4.1. Chức năng bộ chuyển đổi ADC

4.1.1. Khái niệm

Trong các ứng dụng đo lường và điều khiển bằng vi điều khiển, bộ chuyển đổi tương tự-số (ADC) là một thành phần rất quan trọng. Dữ liệu trong thế giới của chúng ta là các dữ liệu tương tự (analog).

Ví dụ nhiệt độ không khí buổi sáng là 25^{o}C và buổi trưa là 32^{o}C, giữa hai mức giá trị này có vô số các giá trị liên tục mà nhiệt độ phải "đi qua" để có thể đạt mức 32^{o}C từ 25^{o}C, đại lượng nhiệt độ như thế gọi là một đại lượng analog.

Trong khi đó, vi điều khiển là một thiết bị số (digital), các giá trị mà một vi điều khiển có thể thao tác là các con số rời rạc vì thực chất chúng được tạo thành từ sự kết hợp của hai mức 0 và 1.

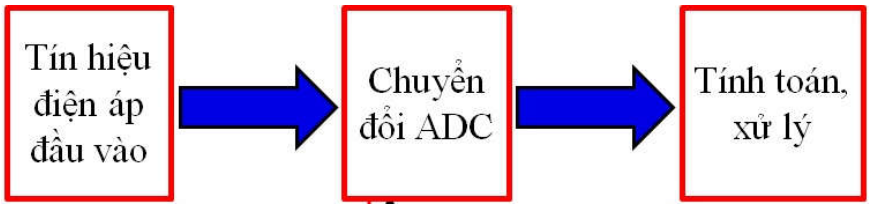
Ví dụ: dùng một thanh ghi 8 bit trong vi điều khiển để lưu lại các giá trị nhiệt độ từ 0^{o}C đến 255^{o}C. Một thanh ghi 8 bit có thể chứa tối đa 256 (28) giá trị nguyên từ 0 đến 255, như thế các mức nhiệt độ không nguyên như 28.123^{o}C sẽ không được ghi lại.

→"số hóa" (digitalize) một dữ liệu analog thành một dữ liệu digital. Quá trình "số hóa" này được thực hiện bởi một thiết bị gọi là "bộ chuyển đổi ADC (Analog to Digital Converter).

ADC trong vi điều khiển:

Tín hiệu tương tự đầu vào là điện áp từ 0 – 5V

→ Để đo các đại lượng khác (nhiệt độ, dòng điện,…) cần mạch chuyển đổi tín hiệu về dạng điện áp.

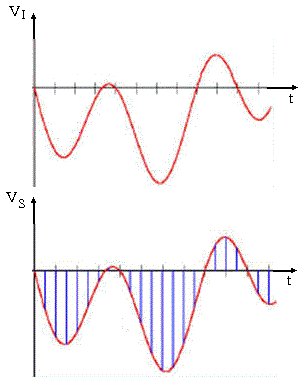


4.1.2. Các bước chuyển đổi ADC

Chuyển đổi ADC được thực hiện qua 3 bước: lấy mẫu, lượng tử hóa, mã hóa.

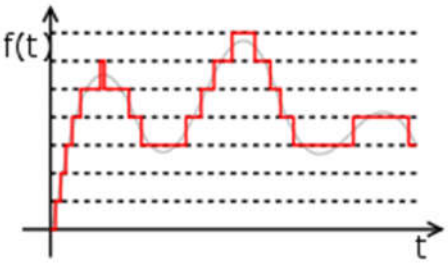
Lấy mẫu

Là quá trình rời rạc hóa tín hiệu theo miền thời gian



Hình 4. 1. Lấy mẫu

Lượng tử hóa



Là quá trình rời rạc hóa tín hiệu theo miền giá trị

Một giá trị bất kỳ của tín hiệu số đều phải biểu thị bằng bội số nguyên lần giá trị đơn vị nào đó, giá trị này là nhỏ nhất được chọn

Đơn vị được chọn theo quy định này gọi là đơn vị lượng tử (bước lượng tử)

Mã hóa

Là quá trình dùng mã nhị phân biểu thị giá trị tín hiệu số sau khi đã được lượng tử hóa

Mã nhị phân có được sau quá trình mã hóa chính là tín hiệu đầu ra của chuyển đổi ADC.

Ví dụ: sử dụng 4 bit nhị phân mã hóa tín hiệu điện áp từ 0 – 5V, với bước lượng tử là 0.5V

STT	Điện áp lượng tử (V)	Mã hóa
1	0.0	0000
2	0.5	0001
…	…	…
10	4.5	1001
11	5.0	1010

→ Không mã hóa được điện áp 0.2V → phải chọn bước lượng tử nhỏ hơn

Sử dụng 4 bit → mã hóa được 16 giá trị → để mã hóa được hết các giá trị điện áp từ 0 – 5V thì chọn bước lượng tử là: $\frac{5(V)}{16-1} = 0.33(V)$

STT	Điện áp lượng tử (V)	Mã hóa
1	0.00	0000
2	0.33	0001
3	0.66	0010
…	…	…
14	4.29	1101
15	4.62	1110
16	4.95	1111

Muốn mã hóa được điện áp chính xác hơn → bước lượng tử phải nhỏ hơn → tăng số bit mã hóa.

Số bit dùng để mã hóa ADC được gọi là độ phân giải. Độ phân giải ADC càng lớn thì đo được điện áp càng chính xác.

Mức điện áp lớn nhất khi mã hóa giá trị tối đa gọi là điện áp tham chiếu VREF

Công thức tính điện áp:

$$V = \frac{VREF}{2^4 - 1} \bullet ADC$$

ADC 8 bit:

$$V = \frac{VREF}{255} \bullet ADC$$

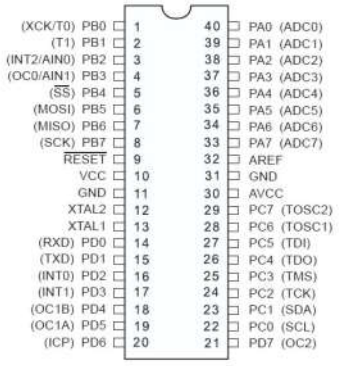
ADC 10 bit:

$$V = \frac{VREF}{1023} \bullet ADC$$

4.2. Lập trình ứng dụng ADC

4.2.1. Thiết lập các thanh ghi

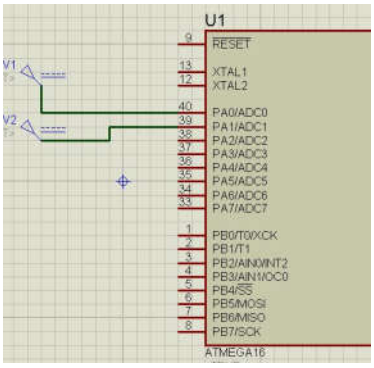
Giới thiệu bộ ADC trong vi điều khiển ATMEGA16:



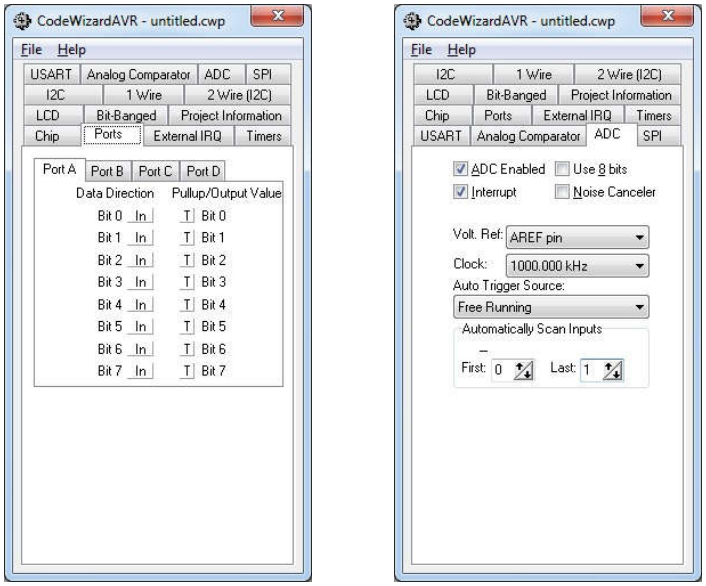
- ✓ Chân 30, 31: điện áp nguồn nuôi cho bộ ADC (5V)
- ✓ Chân 32. AREF: cấp điện áp tham chiếu
- ✓ 8 chân ADC nằm ở PORTA

Chú ý: để bộ ADC hoạt động được chính xác, cần cấp một điện áp ổn định cho chân AVCC và AREF

Nguyên tắc: sử dụng mấy tín hiệu ADC thì kết nối các đầu tương ứng vào các chân của PORTA, theo thứ tự ưu tiên từ nhỏ đến lớn



Khởi tạo cho ADC hoạt động:

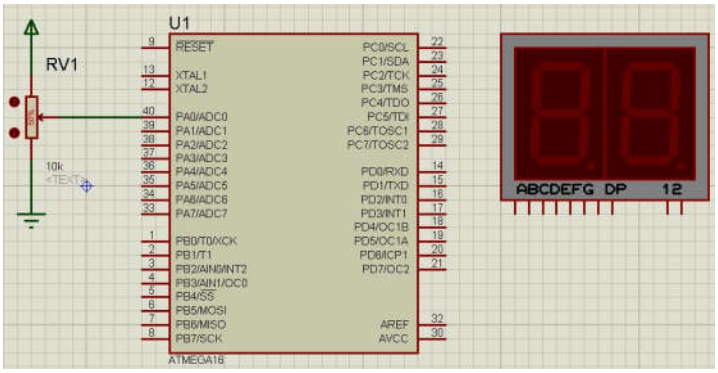


Hình 4. 2. Khởi tạo ADC

- ✓ Cổng vào / ra: phải thiết lập làm đầu vào và thả trôi
- ✓ Sang tab ADC, thiết lập như sau:
  - Tích vào ô ADC Enable để cho phép ADC hoạt động
  - Tích vào ô Interrupt để cho phép ngắt ADC
  - Nếu tích vào ô Use 8 bit thì ADC sẽ hoạt động với độ phân giải là 8 bit, nếu không thì sẽ mặc định là 10 bit
  - Volt. Ref: lựa chọn điện áp tham chiếu
  - Clock: lựa chọn tần số lấy mẫu ADC
  - Auto Trigger Source: lựa chọn Free Running
  - Automatically Scan Inputs: Tự động quét đầu vào ADC, sử dụng bao nhiêu tín hiệu ADC thì lựa chọn giá trị **First** và **Last** tương ứng

4.2.2. Lập trình ứng dụng

Sử dụng tính năng ADC của vi điều khiển, lập trình đo điện áp từ 0-5V, hiển thị kết quả lên LED 7 thanh



4.3. Bài tập thực hành

- Sử dụng 1 biến trở để điều chỉnh điện áp thay đổi từ 0-5V. Sử dụng chức năng ADC của vi điều khiển để đo điện áp, hiển thị giá trị lên LED 7 thanh với độ chính xác là 0.1V

- Sử dụng 1 biến trở để điều chỉnh điện áp thay đổi từ 0-5V. Sử dụng chức năng ADC của vi điều khiển để đo điện áp, hiển thị giá trị lên LED 7 thanh. Độ chính xác đo được có thể lựa chọn 1 trong 2 mức là 0.1V và 0.01V bằng cách sử dụng nút bấm

- Sử dụng cảm biến nhiệt độ LM35 để đo nhiệt độ, hiển thị giá trị đo được lên LED 7 thanh

- Sử dụng 8 biến trở để điều chỉnh điện áp thay đổi từ 0-5V. Sử dụng chức năng ADC của vi điều khiển để đo điện áp, hiển thị giá trị lên LED 7 thanh. Sử dụng 1 nút bấm để lựa chọn tín hiệu đo từ 0-7, sử dụng 8 LED đơn hiển thị để biết đang lựa chọn tín hiệu đo nào

CHƯƠNG 5: HOẠT ĐỘNG CỦA BỘ ĐỊNH THỜI TIMER/COUNTER

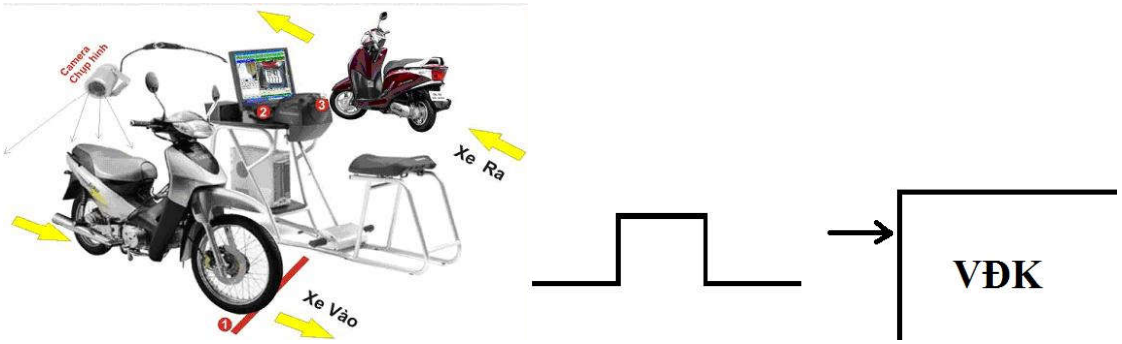
5.1. Hoạt động của bộ định thời

Bộ định thời của VĐK AVR có 2 mảng ứng dụng chính:

✓ Ứng dụng định thì (đo thời gian): dùng để tạo khoảng thời gian chính xác 1s cho đồng hồ

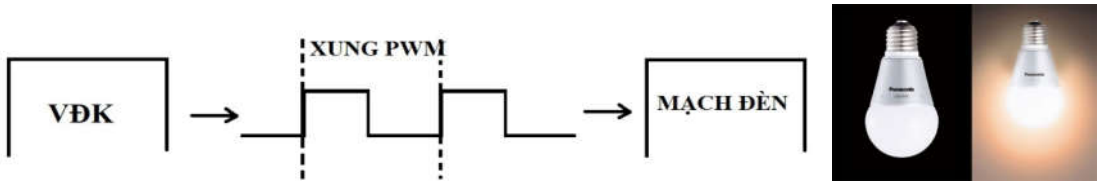
Hình 5. 1. Đồng hồ

✓ Ứng dụng đếm sự kiện: dùng để đếm một sự kiện xảy ra khi có tín hiệu từ cảm biến, có thể dùng để đếm lưu lượng xe ra vào bãi gửi xe.



Hình 5. 2. Đếm số lượng xe

Một ứng dụng phổ biến của định thì đó chính là điều chế độ rộng xung PWM: ứng dụng trong điều chỉnh độ sáng của bóng đèn, hay điều chỉnh tốc độ động cơ



Hình 5. 3. Điều chỉnh độ sáng bóng đèn

5.2. Thanh ghi điều khiển định thời

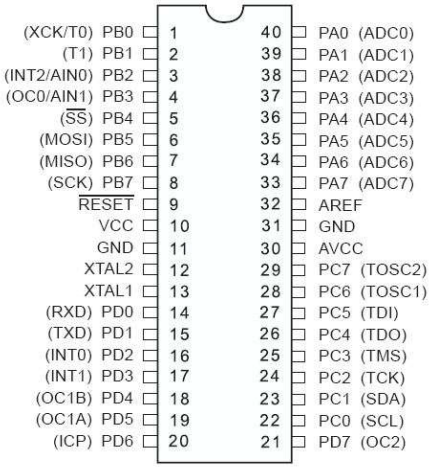
Các bộ Timer của Atmega16:

✓ Timer/Counter 0: đây là 1 bộ timer 8 bit. Có 4 chế độ hoạt động cơ bản là: Normal, CTC, Fast PWM, Phase correct PWM.

✓ Timer/Counter 1: đây là 1 bộ timer 16 bit. Nó cũng có đầy đủ các chế độ hoạt động như của Timer 0, tuy nhiên vì là bộ timer 16 bit nên phạm vi hoạt động của nó rộng hơn. Nếu như giá trị MAX trong Timer0 chỉ là 0xFF, thì giá trị MAX trong Timer1 lên đến 0xFFFF. Ngoài ra bộ Timer1 còn có thể tạo ra 2 tín hiệu PWM độc lập ở các chân OC1A (PORTD.5) và OC1B (PORTD.4), giúp dễ dàng khi điều khiển 2 động cơ cùng 1 lúc (2 động cơ chạy bánh).

✓ Timer/Counter 2: đây là 1 bộ Timer 8 bit. Nó cũng có 4 chế độ hoạt động cơ bản giống như Timer0, đó là: Normal, CTC, Fast PWM, Phase correct PWM

- ✓ Chân T0 (PB0), T1 (PB1): đầu vào cho bộ Timer / Counter
- ✓ Chân OC0 (PB3): đầu ra cho bộ Timer 0
- ✓ Chân OC1A (PD5) và OC1B (PD4): đầu ra cho bộ Timer 1
- ✓ Chân OC2 (PD7): đầu ra cho bộ Timer 2



Một số định nghĩa:

✓ **BOTTOM**: là giá trị thấp nhất mà một T/C có thể đạt được, giá trị này luôn là 0.

✓ **MAX**: là giá trị lớn nhất mà một T/C có thể đạt được, giá trị này được quy định bởi bởi giá trị lớn nhất mà thanh ghi đếm của T/C có thể chứa được. Ví dụ với một bộ T/C 8 bit thì giá trị MAX luôn là 0xFF (tức 255 trong hệ thập phân), với bộ T/C 16 bit thì MAX bằng 0xFFFF (65535). Như thế MAX là giá trị không đổi trong mỗi T/C.

✓ **TOP**: là giá trị mà khi T/C đạt đến nó sẽ thay đổi trạng thái, giá trị này không nhất thiết là số lơn nhất 8 bit hay 16 bit như MAX, giá trị của TOP có thể thay

đổi bằng cách điều khiển các bit điều khiển tương ứng hoặc có thể nhập trực tiếp thông qua một số thanh ghi.

- **Thanh ghi TCNT0 (Timer/Counter Register):**

Bit	7	6	5	4	3	2	1	0	
	TCNT0[7:0]								TCNT0
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	

Hình 5. 4. Thanh ghi TCNT0

Đây là 1 thanh ghi 8 bit chứa giá trị vận hành của Timer0. Cứ mỗi lần xuất hiện 1 tín hiệu "kích" thì giá trị của thanh ghi này tăng thêm 1 đơn vị. Giá trị MAX mà thanh ghi này có thể đạt đến là 255. Sau đó nếu xuất hiện thêm 1 xung kích nữa thì giá trị của thanh ghi này được xóa về 0.

- **Thanh ghi OCR0 (Output Compare Register):**

Bit	7	6	5	4	3	2	1	0	
	OCR0[7:0]								OCR0
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	

Hình 5. 5. Thanh ghi OCR0

Đây là 1 thanh ghi 8 bit. Giá trị vận hành của Timer TCNT0 sẽ liên tục được so sánh với thanh ghi này. Khi 2 giá trị này bằng nhau, thì xảy ra 1 sự kiện "***Compare Match***". Sự kiện này sẽ tạo ra 1 ngắt (nếu ngắt được cho phép), hay xuất tín hiệu ra chân **OC0**, tùy theo chế độ thực thi của Timer.

- **Thanh ghi TCCR0 (Timer/Counter Control Register):**

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	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	

Hình 5. 6. Thanh ghi TCCR0

Đây là thanh ghi điều khiển hoạt động của Timer.

- ✓ Các bit 0,1,2 (CS00,CS01,CS02): các bit quy định xung cấp cho hoạt động của Timer:

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.

✓ Các bit 3,6 (WGM00,WGM01): Đây là các bit chọn chế độ trong Timer

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

✓ Bit 4,5 (COM00,COM01): Đây là 2 bit quy định đầu ra trong các phép so sánh

Trong quá trình Timer đếm, giá trị của thanh ghi TCNT0 sẽ liên tục được so sánh với OCR0. Khi 2 giá trị này bằng nhau, thì sẽ có sự kiện "***Compare Match***" xảy ra. Khi đó giá trị đầu ra OC0 (PORTB.3) sẽ được xuất ra mức tín hiệu như bảng dưới đây:

- Đầu ra ở chế độ non-PWM (Normal hoặc CTC):

Compare Output Mode, non-PWM Mode

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

- Đầu ra ở chế độ Fast PWM:

Compare Output Mode, Fast PWM Mode[1]

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at TOP
1	1	Set OC0 on compare match, clear OC0 at TOP

- Đầu ra ở chế độ Phase Correct PWM:

Compare Output Mode, Phase Correct PWM Mode[1]

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.

✓ Chú ý: muốn xuất đầu ra thì chân OC0 phải để OUT, tức DDRB.3=1;

➢ **Thanh ghi TIMSK (Timer/Counter Interrupt Mask Register):**

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	

Hình 5. 7. Thanh ghi TIMSK

Đây là thanh ghi điều khiển ngắt trong Timer.

✓ Bit 0 (**TOIE0-Timer/Counter0 Overflow Interrupt Enable**): đây là bit cho phép ngắt tràn trong Timer0. Khi bit này được ghi là 1 thì cho phép ngắt tràn bộ định thời. Khi đó 1 ngắt sẽ xảy ra mỗi khi bộ Timer tràn.

✓ Bit 1 (**OCIE0-Timer/Counter0 Output Compare Match Interrupt Enable**): đây là bit cho phép ngắt so sánh trong Timer0. Khi bit này được ghi là 1 thì cho phép ngắt so sánh bộ định thời. Khi đó 1 ngắt sẽ xảy ra mỗi khi có sự kiện "***Compare Match***".

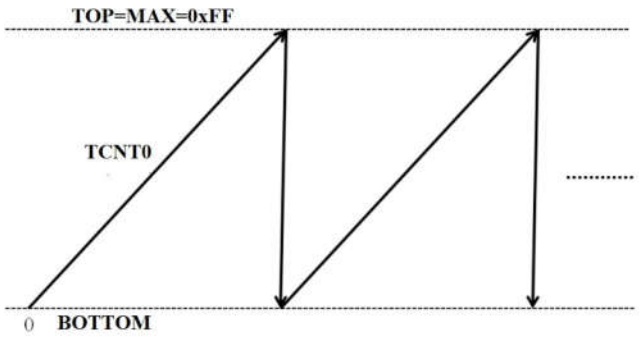
5.3. Các chế độ hoạt động của bộ định thời

5.3.1. Chế độ Normal

Thiết lập bit:

WGM00 =0

WGM01=0



Hình 5. 8. Sơ đồ hoạt động của chế độ Normal

Đây là chế độ hoạt động đơn giản nhất của Timer. Mỗi lần có 1 tín hiệu clock thì giá trị của thanh ghi TCNT0 được tăng lên 1, và tăng đến mức TOP là 255. Khi đó ở nhịp tiếp theo thì bộ Timer bị tràn, và 1 thanh ghi TCNT0 được đặt lại mức 0, và lại tiếp tục quá trình đếm như vậy. Thời gian để TCNT0 tăng lên 1 đơn vị phụ thuộc vào tần số ta chọn, tần số càng lớn thì thời gian đếm càng nhanh.

Ứng dụng:

- ✓ Đo thời gian
- ✓ Quét LED 7 thanh
- ✓ Điều chế độ rộng xung (PWM) bằng phần mềm

Tần số hoạt động của Timer: là tần số để thanh ghi TCNT0 tăng lên 1 đơn vị, được thiết lập bởi 3 bit CS00,CS01,CS02

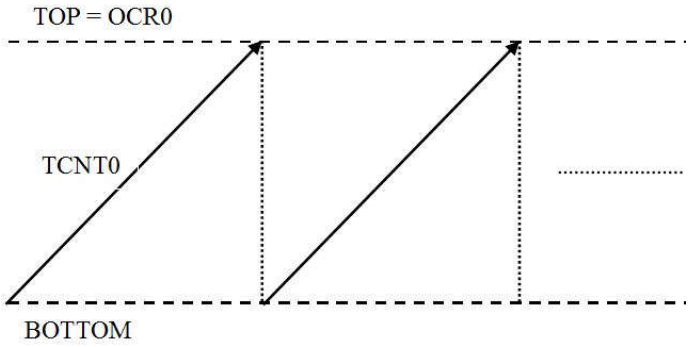
Để đếm thời gian ta lựa chọn 1 tần số phù hợp, sao cho tính toán ra thời gian đẹp, ví dụ 125kHz. Sau đó lựa chọn giá trị khởi đầu cho thanh ghi TCNT0, giá trị này cũng chọn sao cho khi tính toán ra thời gian đẹp, ví dụ TCNT0=6. Như vậy giá trị xuất phát của bộ đếm là 6, bộ đếm sẽ đếm từ 6 đến 255, sau đó 1 nhịp thì tràn, và 1 ngắt tràn được xảy ra. Nhưng ngay sau khi ngắt tràn được phục vụ thì thanh ghi TCNT0 lại trở về 0, chứ không phải là giá trị 6 mà ta muốn. Như vậy ở câu lệnh cuối cùng của chương trình phục vụ ngắt tràn ta lại đặt lệnh TCNT0=6, làm giá trị xuất phát cho lần đếm tiếp theo. Như vậy, thời gian để Timer tràn 1 lần sẽ là (256-6)/125000=2ms. Và sau 1 lần tràn ta đã có 1 thời gian là 2ms, có thể xử lí thời gian này như thế nào thì tùy người sử dụng.

5.3.2. Chế độ CTC

Thiết lập bit:

WGM01 = 1

WGM00 = 0



Hình 5. 9. Sơ đồ hoạt động của chế độ CTC

Đây là chế độ mà giá trị trong thanh ghi TCNT0 luôn được so sánh với giá trị trong thanh ghi OCR0. Khi giá trị trong thanh ghi TCNT0 bằng giá trị trong thanh ghi OCR0 thì giá trị trong thanh ghi TCNT0 sẽ bị xoá đi, và tiếp tục quá trình đếm mới. Giá trị trong OCR0 đóng vai trò là giá trị TOP cho bộ đếm. Tuy nhiên trong chế độ này nếu giá trị mới ghi vào thanh ghi OCR0 mà nhỏ hơn giá trị tức thời của bộ đếm thì thì 1 so sánh sẽ bị lỡ, khi đó bộ đếm sẽ đếm đến giá trị lớn nhất sau đó rơi xuống giá trị 0 trước khi so sánh tiếp theo xuất hiện.

Ứng dụng:

- ✓ Đo thời gian
- ✓ Tạo xung vuông

Để đếm thời gian thì cũng giống như chế độ Normal, chỉ khác ở chỗ là ở chế độ Normal thì giá trị TOP sẽ là 255, còn giá trị TOP của CTC là OCR0, giá trị này có thể thay đổi được, và tính toán thời gian bằng cách tính toán giá trị thanh ghi này, thay đổi giá trị thanh ghi này,mà không cần phải khởi tạo giá trị ban đầu TCNT0 cho bộ đếm.

Tạo xung vuông ra 1 chân: có thể tạo xung ở đầu ra của Timer (PORTB.3), hoặc tạo xung ở 1 chân bất kì. Mỗi lần bộ đếm tràn, ta đảo bit ở chân ra. Như vậy 2 lần tràn sẽ là 1 chu kì của xung vuông. Tính toán tần số hoạt động của Timer và giá trị OCR0 để tạo ra 1 tần số mong muốn. VD: muốn tạo ra xung có tần số 1kHz → chu kì

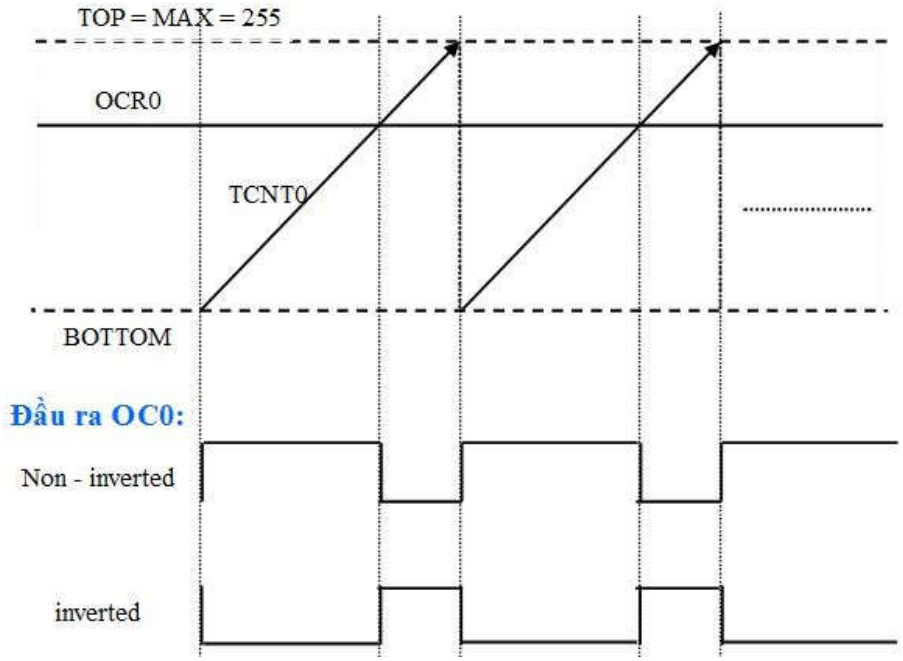
sẽ là 1ms →1 lần tràn Timer mất 500us. Chọn tần số Timer là 125kHz giá trị OCR0 cần chọn là: 500*10-6*125000+1=63.5, có thể lấy gần đúng thành 63

5.3.3. Chế độ Fast PWM

Thiết lập bit:

WGM01 = 1

WGM00 = 1



Hình 5. 10. Sơ đồ hoạt động chế độ Fast PWM

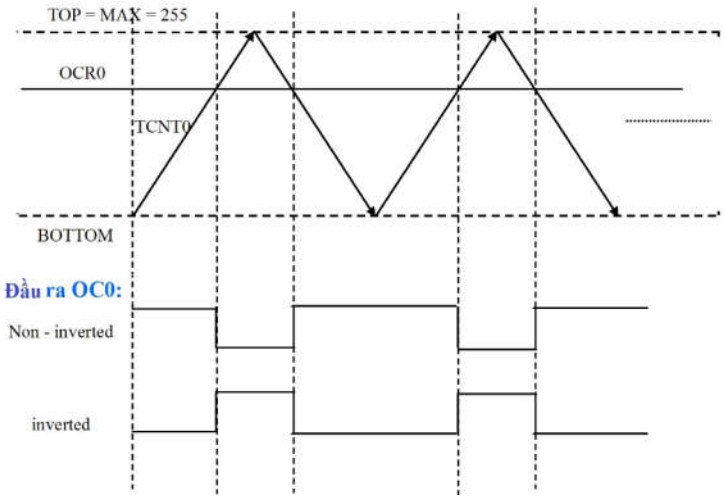
Bộ đếm sẽ đếm từ BOTTOM đến MAX sau đó khởi động lại từ BOTOM. Trong chế độ không đảo (Non-inverted) đầu ra OC0 (PORTB.3) sẽ bi xoá khi có phép toán so sánh giữa TCNT0 và thanh ghi OCR0 là bằng nhau, và sẽ được set lên 1 khi giá trị tràn và đạt về BOTTOM. Trong chế độ đảo (inverted), đầu ra sẽ được set lên 1 khi sự so sánh giữa thanh ghi TCNT0 và giá trị thanh ghi OCR0 bằng nhau và sẽ bị xoá khi giá trị tràn và đạt về BOTTOM.

5.3.4. Chế độ Phase Correct PWM

Thiết lập bit:

WGM01 = 0

WGM00 = 1



Hình 5. 11. Sơ đồ hoạt động chế độ Phase Correct PWM

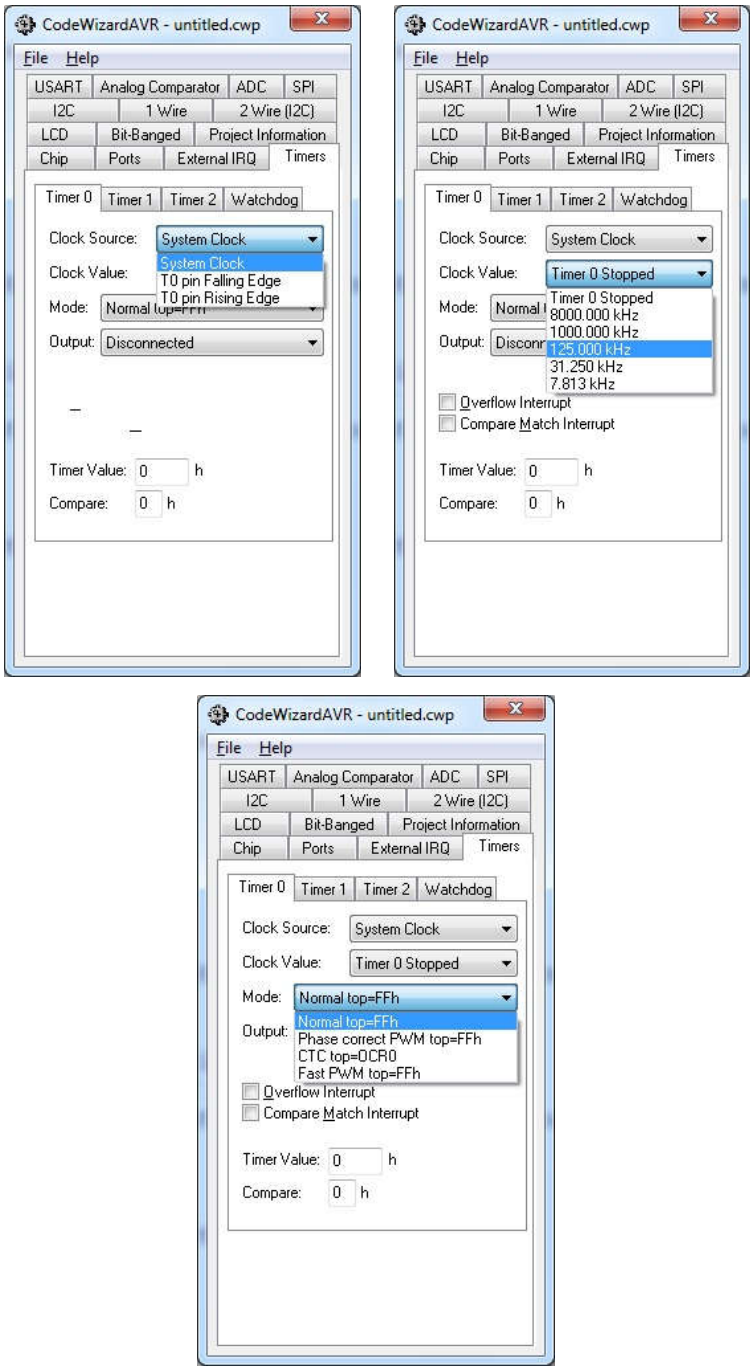
Chế độ này hoạt động dựa trên hai sườn lên xuống. Bộ đếm sẽ đếm liên tục từ giá trị BOTTOM đến giá trị MAX và sau đó đếm ngược lại từ giá trị MAX đến giá trị BOTTOM. Trong chế độ so sánh không đảo (Non-inverted) chân OC0 sẽ bị xóa khi giá trị TCNT0 bằng giá trị OCR0 trong quá trình đếm lên và sẽ được set bằng 1 khi giá trị so sánh xuất hiện trong quá trình đếm xuống. Chế độ so sánh đảo thì các giá trị là ngược lại.

Với hoạt động hai sườn xung này thì chế độ này không tạo ra được tần số lớn như chế độ một sườn xung. Nhưng do tính cân đối của hai sườn xung thì nó tốt hơn cho điều khiển động cơ. Chế độ Phase correct PWM hoạt động cố định là 8 bít. Trong chế độ này bộ đếm sẽ tăng cho đến khi đạt giá trị MAX, khi đó nó sẽ đổi chiều đếm.

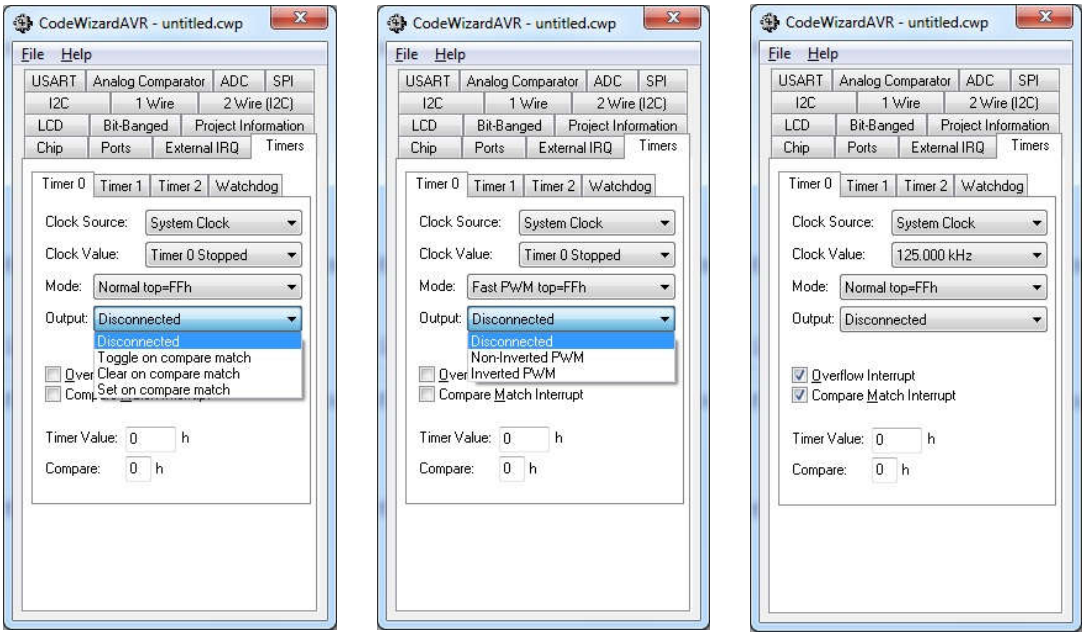
5.4. Ví dụ

Khởi tạo Timer bằng CodeWizard

- ✓ Chọn tab “Timers”, chọn tiếp tab “Timer 0”, hoặc “Timer 1”, hoặc “Timer 2”
- ✓ Clock Source: chọn nguồn clock cho bộ Timer hoạt động
  - System Clock: nguồn clock lấy từ thạch anh
  - T0 pin Falling Edge: nguồn clock cạnh xuống lấy từ đầu vào chân T0
  - T0 pin Falling Edge: nguồn clock cạnh lên lấy từ đầu vào chân T0
- ✓ Clock value: lựa chọn tần số cho bộ Timer hoạt động
- ✓ Mode: lựa chọn chế độ hoạt động cho Timer



- ✓ Output: lựa chọn đầu ra trong các phép so sánh "***Compare Match***"
- ✓ Hai ô **Overflow Interrupt** và **Compare Match Interrupt** là cho phép ngắt tràn và cho phép ngắt so sánh được thực thi hay không



Cách thiết lập Timer để đo thời gian:

- ✓ Lựa chọn tần số hoạt động: sử dụng 3 bit CS00,CS01,CS02
- ✓ Lựa chọn chế độ hoạt động Normal: sử dụng 2 bit WGM00, WGM01
- ✓ Kích hoạt ngắt tràn Timer0: đặt bit 0 của thanh ghi TIMSK bằng 1
- ✓ Tính toán thời gian:

Ví dụ: tính toán thời gian trễ 1(s) nếu sử dụng thạch anh 8MHz

Lựa chọn bộ chia tần 8, khi đó tần số hoạt động của Timer sẽ là

$$f = f_{ta} / 8 = 1(MHz)$$

→ $T = 1 / f = 10^{-6}s = 1us$ (thời gian để thanh ghi TCNT0 tăng 1 đơn vị)

TCNT0 : 0→255, được 256 chu kì

→ Thời gian để 1 lần tràn timer: $T_{tran} = 256 * 1us = 256$ us.

→ Kích hoạt ngắt tràn Timer. Ở CTCPVN sẽ dùng 1 biến dem để đếm số lần tràn.

256 us là một số không đẹp, để tính toán thời gian thì sẽ gây ra sai số.

→ Để tính thời gian được chính xác hơn thì cần thay đổi giá trị này.

→ Giá trị TOP là 255 không thay đổi được, do đó phải thay đổi giá trị xuất phát ban đầu của TCNT0.

→ TCNT0: 6→255, được 250 chu kì

→ Thời gian cho 1 lần tràn timer sẽ là 250 * 1us = 250us

Tuy nhiên, sau khi timer bị tràn thì TCNT0 lại tự động clear về 0, do đó ở chu kì hoạt động tiếp theo lại phải gán lại giá trị xuất phát ban đầu cho TCNT0

→ Ở CTCPVN:

→ Lệnh tăng giá trị của biến dem

→ Lệnh gán giá trị ban đầu cho TCNT0 = 6;