CÙNG HỌC AVR

AVR1 – LÀM QUEN AVR

I. Giới thiệu

Khi tìm hiểu về vi điều khiển bạn sẽ bắt gặp cụm từ "**AVR 8 bits RISC Microcontroller**", trong đó AVR là tên của của 1 họ vi điều khiển do Atmel (Na Uy) sản xuất (Atmel cũng là nhà sản xuất các vi điều khiển họ 89C51 mà bạn có thể đã từng nghe đến), 8 bits là cấu trúc của thanh ghi bên trong chip, RISC (Reduced Instruction Set Computer) là 1 kiểu cấu trúc phổ biến của các bộ xử lí.

- **Tại sao AVR**: so với các chip vi điều khiển 8 bits khác, AVR có nhiều đặc tính hơn hẳn, hơn cả trong tính ứng dụng (dễ sử dụng) và đặc biệt là về chức năng.
  - Gần như chúng ta không cần mắc thêm bất kỳ linh kiện phụ nào khi sử dụng AVR, thậm chí không cần nguồn tạo xung clock cho chip (thường là các khối thạch anh).
  - Thiết bị lập trình (mạch nạp) cho AVR rất đơn giản, có loại mạch nạp chỉ cần vài điện trở là có thể làm được. một số AVR còn hỗ trợ lập trình on – chip bằng bootloader không cần mạch nạp…
  - Bên cạnh lập trình bằng ASM, cấu trúc AVR được thiết kế tương thích C.
  - Nguồn tài nguyên về source code, tài liệu, application note…rất lớn trên internet.
  - Hầu hết các chip AVR có những tính năng (features) sau:

    Ø Có thể sử dụng xung clock lên đến 16MHz, hoặc sử dụng xung clock nội lên đến 8 MHz (sai số 3%)
    Ø Bộ nhớ chương trình Flash có thể lập trình lại rất nhiều lần và dung lượng lớn, có SRAM (Ram tĩnh) lớn, và đặc biệt có bộ nhớ lưu trữ lập trình được EEPROM.
    Ø Nhiều ngõ vào ra (I/O PORT) 2 hướng (bi-directional).
    Ø 8 bits, 16 bits timer/counter tích hợp PWM
    Ø Các bộ chuyển đổi Analog – Digital phân giải 10 bits, nhiều kênh.
    Ø Chức năng Analog comparator.
    Ø Giao diện nối tiếp USART (tương thích chuẩn nối tiếp RS-232)
    Ø Giao diện nối tiếp Two –Wire –Serial (tương thích chuẩn I^2C) Master và Slaver.
    Ø Giao diện nối tiếp Serial Peripheral Interface (SPI)
    Ø ….

  - Một số chip AVR:

    AT90S1200
    AT90S2313
    AT90S2323 and AT90S2343
    AT90S2333 and AT90S4433

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.

- Trong bài viết này tôi sử dụng chip ATmega8 để làm ví dụ, tôi chọn ATmega8 vì đây là loại chip thuộc dòng AVR mới nhất, nó có đầy đủ các tính năng của AVR nhưng lại nhỏ gọn (gói PDIP có 28 chân) và low cost nên các bạn có thể mua để tự mình tạo ứng dụng.

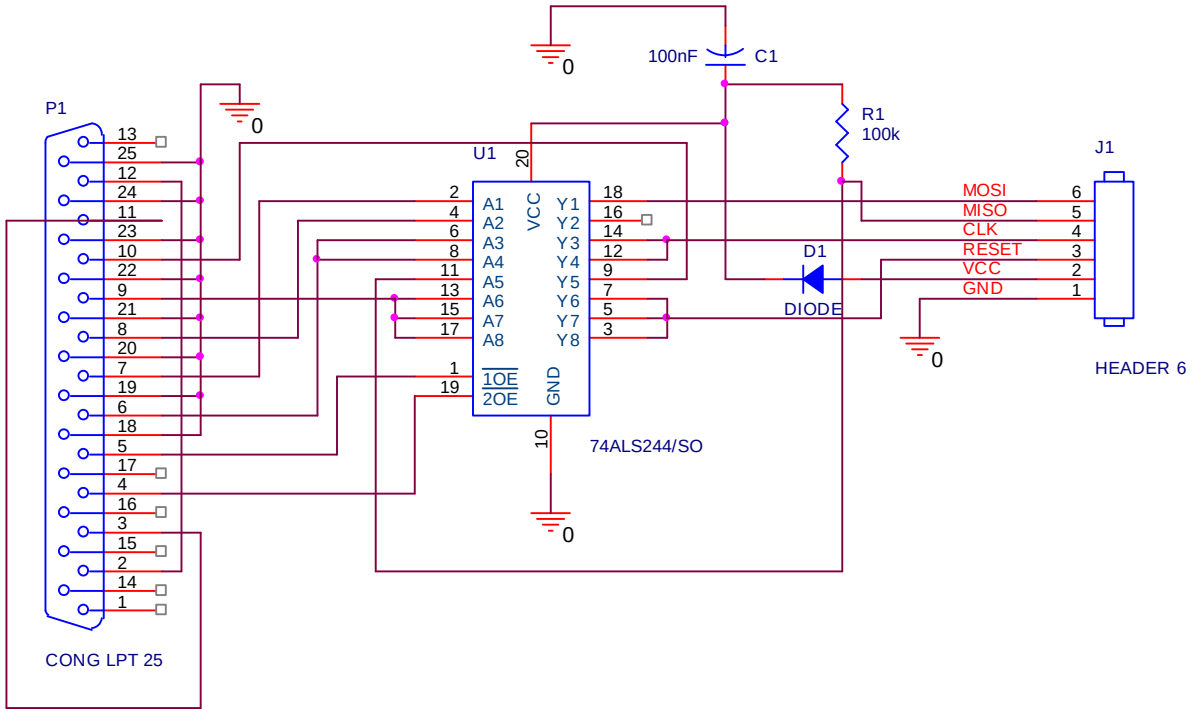
- **Tại sao ASM (Assembly):** bạn có thể không cần biết về cấu trúc của AVR vẫn có thể lập trình cho AVR bằng các phần mềm hỗ trợ ngôn ngữ cấp cao như BascomAVR (Basic) hay CodevisionAVR (C), tuy nhiên đó không phải là mục đích của bài viết này. Để hiểu thấu đáo về AVR bạn phải lập trình bằng chính ngôn ngữ của nó, ASM. Như vậy lập trình bằng ASM giúp bạn hiểu tường tận về AVR, và tất nhiên để lập trình được bằng ASM bạn phải hiểu về cấu trúc AVR….Một lý do khác bạn mà tôi khuyên bạn nên lập trình bằng ASM là các trình dịch (compiler) ASM cho AVR là hoàn toàn miễn phí, và nguồn source code cho AVR viết bằng ASM là rất lớn. Tuy nhiên một khi bạn đã thành thạo AVR và ASM bạn có thể sử dụng các ngôn ngữ cấp cao như C để viết ứng dụng vì ưu điểm của ngôn ngữ cấp cao là giúp bạn dễ dàng thực hiện các phép toán đại số 16 hay 32 bit (vốn là vấn đề khó khăn khi lập trình bằng ASM).

II. Công cụ

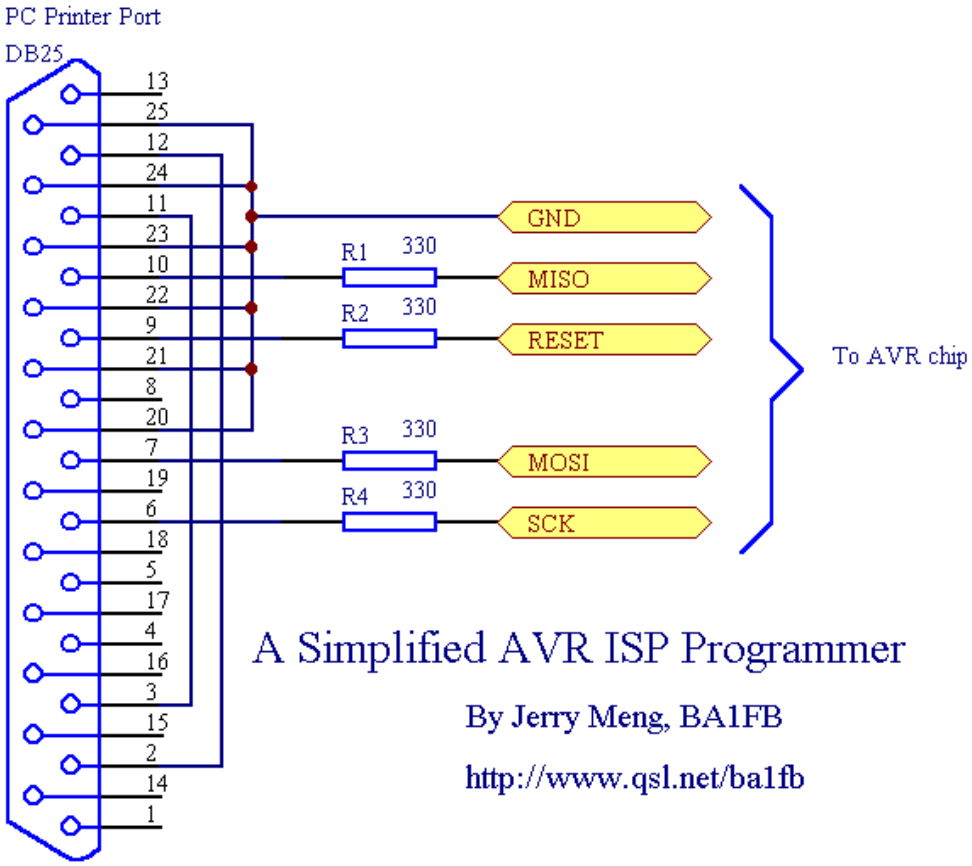
- **Trình biên dịch:** có rất nhiều trình biên dịch bạn có thể sử dụng để biên dịch code của bạn thành file intel hex để nạp vào chip, một số trình dịch quen thuộc có thể kể đến như sau:
  - **AvrStudio**: là trình biên dịch ASM chính thức cung cấp bởi Atmel, đây là trình biên dịch hoàn toàn miễn phí 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

- **Wavrasm**: cũng được cung cấp bởi Atmel, nó chính là tiền thân của AvrStudio. Hiện tại wavrasm không còn được sử dụng nhiều vì so với AvrStudio trình biên dịch này có nhiều hạng chế, nếu bạn quan tâm có thể download tại đây: ftp://auto.vnlug.org/AUTO.NLU/Softwares/3-Microcontroller/AVR/Compiler/
- **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. Việc lập trình bằng avrgcc tôi sẽ đề cập trong những phần sau.
- **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 demo (đầy đủ chức năng nhưng nhưng giới hạn dung lượng bộ nhớ chương trình 2KB) tại: http://www.hpinfotech.ro/html/download.htm. hoặc bản full tại ftp://auto.vnlug.org.
- **ICCAVR:** lập trình C cho avr, bản demo tại: http://www.imagecraft.com/
- **BascomAVR**: lập trình cho AVR bằng basic, đây là trình biên dịch khá hay và dễ sử dụng, hỗ trợ rất nhiều thư viện. Tuy nhiên rất khó debug lỗi và không thích hợp cho việc tìm hiểu AVR. Vì vậy tôi không bạn khuyến khích bạn sử dụng trình dịch này. Bạn có thể downlaod bản demo (4K limit) tại đây: http://www.mcselec.com/index.php?option=com_docman&task=cat_view&gid=73&Itemid=54
- Và còn rất nhiều trình biên dịch khác cho AVR mà tôi không kể ra đây, nhìn chung tất cả các trình biên dịch này hỗ trợ C hoặc Basic hoặc thậm chí Pascal..Việc chọn 1 trình biên dịch tùy thuộc vào mục đích, vào mức độ ứng dụng, vào kinh nghiệm sử dụng và nhiều lý do khác nữa. Ví dụ tôi thường dùng Avrstudio và avrgcc khi học sử dụng AVR và khi viết thư viện…nhưng khi cần viết chương trình ứng dụng tôi thường chọn CodeVisionAVR.
- Trong bài viết này tôi hướng dẫn bạn sử dụng AvrStudio để viết chương trình cho AVR bằng ASM.

- **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.
- **Mạch nạp:** mạch nạp cho AVR có nhiều chuẩn, có thể sử dụng cổng nối tiếp (COM) hay song song (LPT) của máy tính làm đường nạp. Nhìn chung mạch nạp cho AVR thường đơn giản, rất dễ làm, dưới đây tôi giới thiệu 2 loại mạch nạp được coi là đơn giản nhất, bạn có thể tham khảo và tự làm (phần hướng dẫn làm mạch nạp sẽ được đề cập trong 1 bài viết khác).



Hình 1 : mạch nạp theo STK200

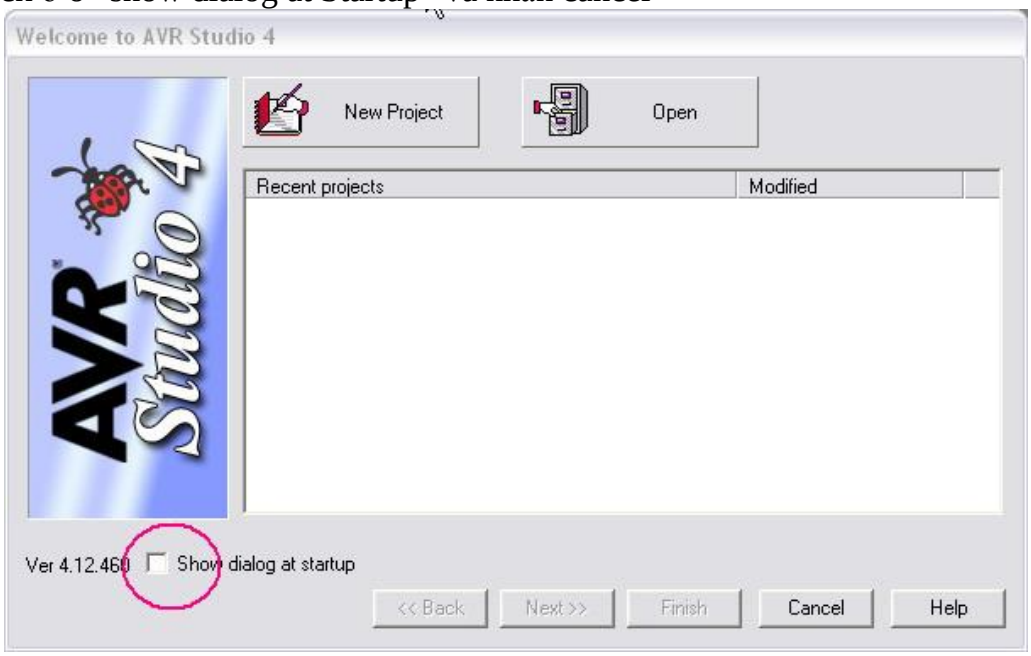


Hình2 : mạch nạp AVR ISP với chỉ 4 điện trở !!!!

- **Chương trình mô phỏng:** avr simulator là chương trình mô phỏng được tích hợp sẵn trong Avrstudio, avr simulator cho phép bạn quan sát trạng thái các thanh ghi bên trong AVR nên rất phù hợp để bạn debug chương trình. Trong bài viết cũng sẽ hướng dẫn bạn sử dụng avr simulator để mô phỏng chương trình ví dụ. Proteus là chương trình thứ hai tôi muốn nói đến, Proteus không mô phỏng hoạt động bên trong chip mà mô phỏng kết quả chương trình, nó là trình mô phỏng mạch điện tử giả thời gian thực nên bạn có thể sử dụng để kiểm tra chương trình 1 cách trực quan hơn. Proteus là 1 công cụ hữu ích khi khi các bạn chưa có điều kiện làm các mạch điện tử.

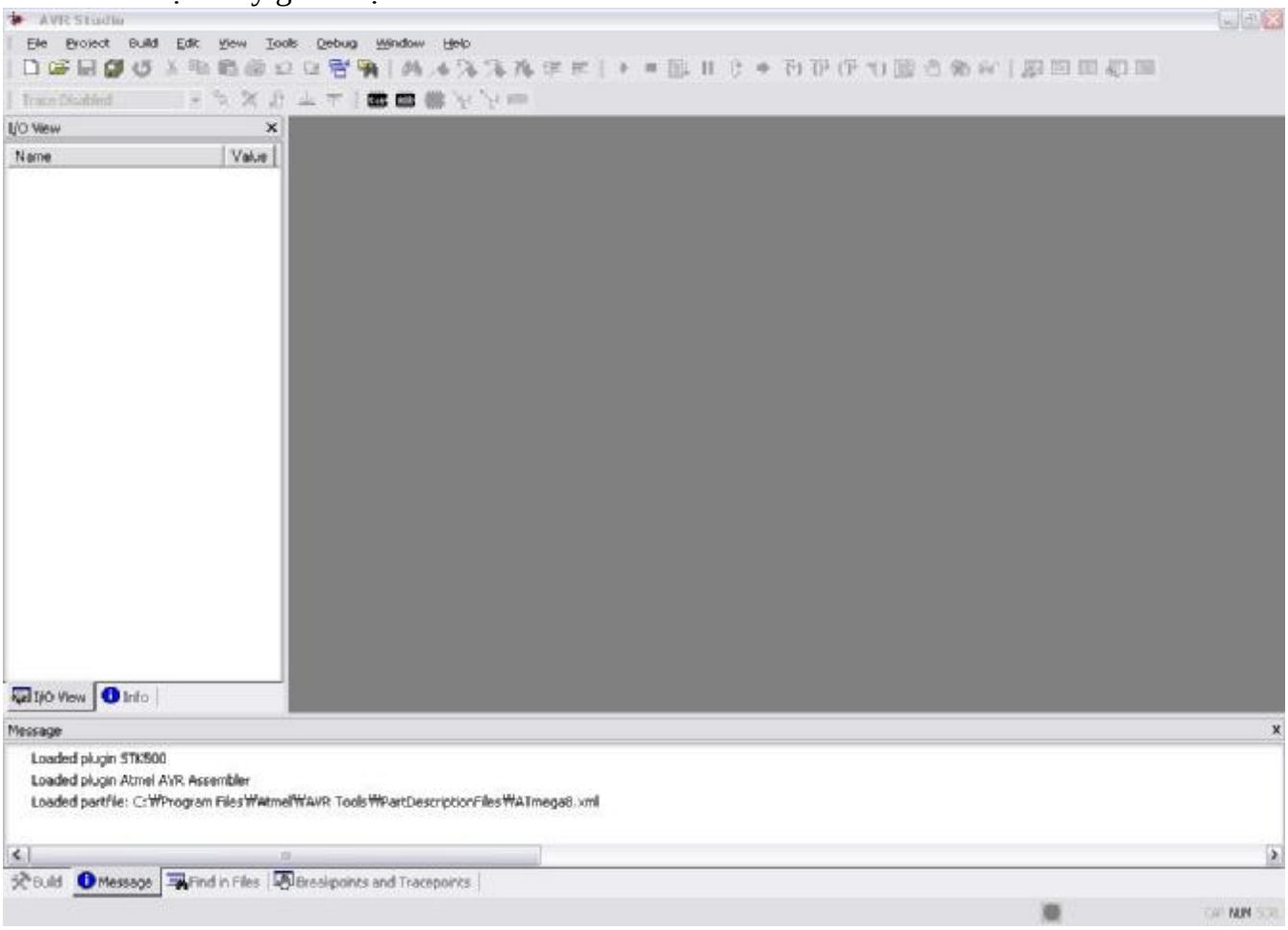
III. Viết ví dụ đầu tiên.

- Trong bài viết này tôi sử dụng 2 phần mềm là AvrStudio và Proteus. Sau khi download AvrStudio, bạn hãy cài đăt phần mềm trên máy của bạn, quá trình cài đặt rất đơn giản, bạn hãy theo các mặc định và nhấn "next" để cài đặt.
- Trong bài đầu tiên này chúng ta sẽ viết thử 1 chương trình đơn giản cho AVR sau đó chạy mô phỏng bằng Proteus. Có thể có một số câu lệnh các bạn sẽ không hiểu, nhưng đừng bận tâm quá, trong bài 2 chúng ta sẽ học về cấu trúc AVR các bạn sẽ được giải thich rõ hơn.
- Bắt đầu với AvrStudio4: bạn chạy AvrStudio từ "**Start/ All Programs/ Atmel AVR Tools/ AvrStudio 4**"
- Ở lần đầu chạy AvrStudio, 1 dialog "Welcome to AvrStudio 4" xuất hiện, hãy bỏ check ở ô "show dialog at Startup" và nhấn cancel



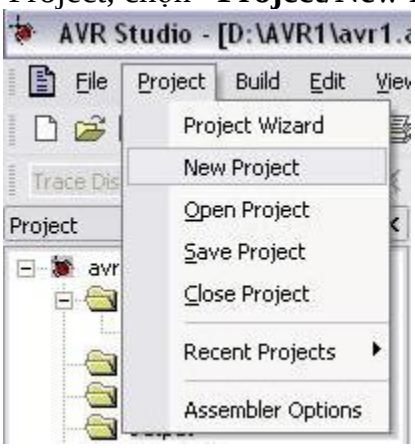
Hình 3: Welcome to AVR studio 4 Diaolg

- Bạn thấy giao diện AVR Studio 4 như sau:



Hình 4: giao diện AVR Studio

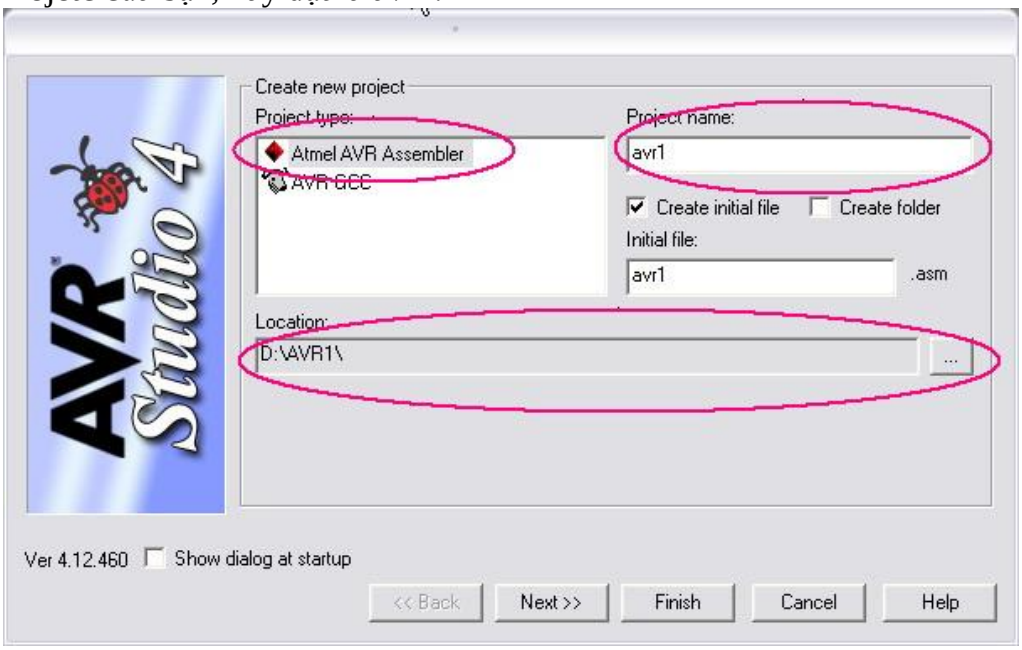
- Giao diện AVR Studio rất dễ sử dụng, vì vậy chúng ta sẽ kết hợp tìm hiểu trong lúc viết ví dụ.
- Tạo Project mới: từ menu Project, chọn **"Project/New Project"**



Hình 5: tạo Project mới

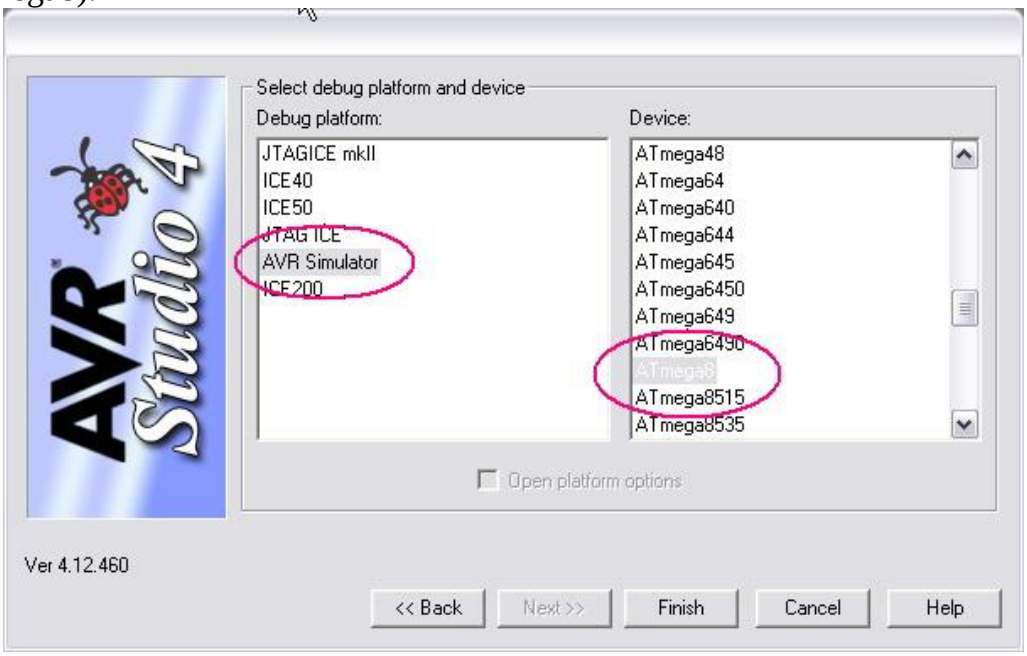
- Một dialog mới xuất hiện cho phép bạn setting Project của bạn, trong vùng "Project Type" chọn "Atmel AVR assembler", tức lập trình bằng ngôn ngữ Assembly và trình

dịch là Atmel AVR assembler (trình dịch tích hợp trong AVR Studio); "Location", chọn nơi chứa Project (trong ví dụ này tôi chọn thư mục D/AVR1); "Project name", tên Projetc của bạn, hãy đặt là avr1.



Hình 6: setting Project

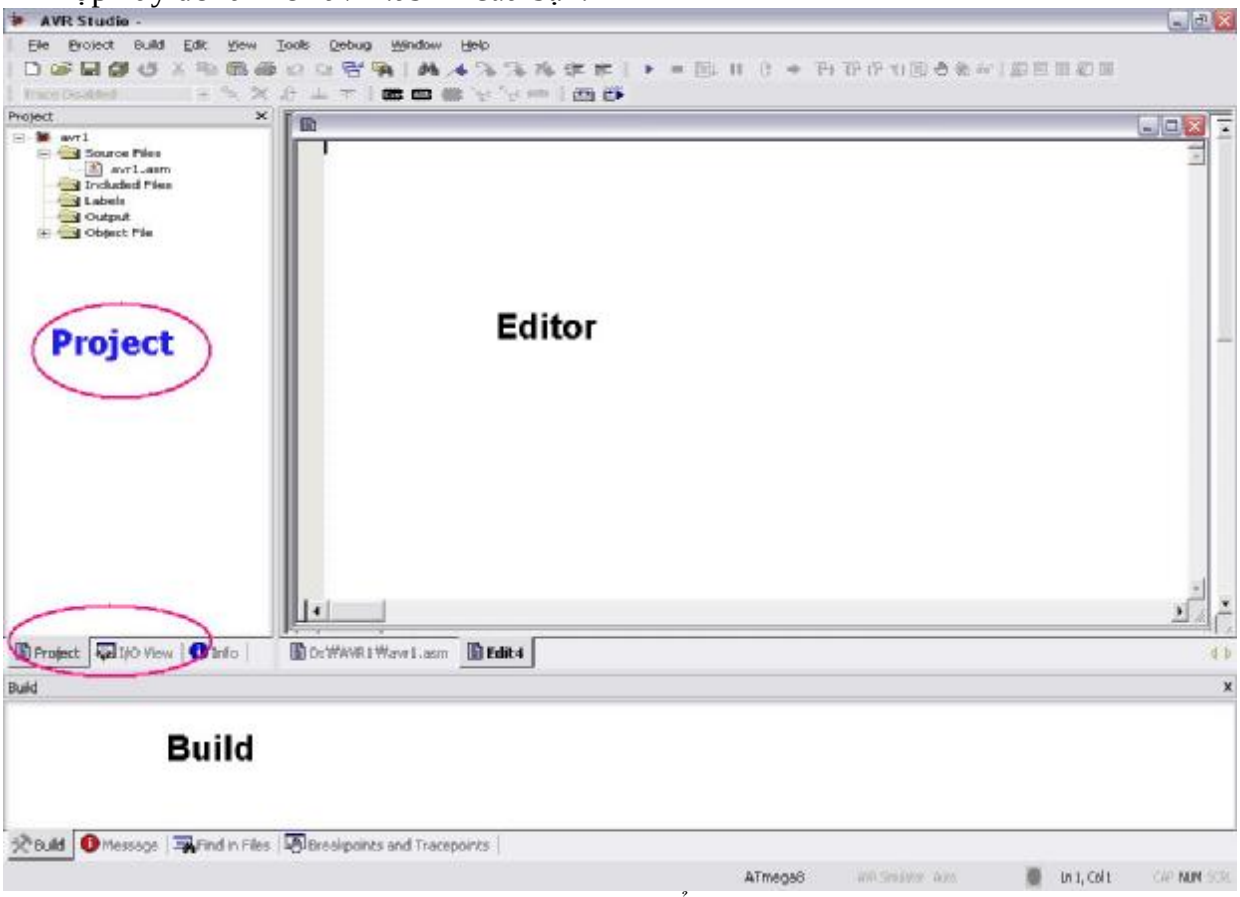
- Nhấn Next để tiếp tục chọn Platform và device, việc này phục vụ cho mục đích debug chương trình hay mô phỏng bằng avr simulator. Bạn hãy chọn "AVR Simulator" trong ô Platform và Atmega8 trong ô device (chúng ta sẽ viết chương trình cho chip Atmega8).



Hình 7: Chọn Platform và device

- Nhấn finish để kết thúc setting project, bạn thấy các cửa số của "Project" chứa các thông tin Project của bạn, bạn thấy trong mục "Source files" có 1 file "avr1.asm" là

source code của bạn. Bạn có thể nhấn vào switch tab bên dưới cửa sổ Project để xem cửa số "I/O View", cửa số này chứa thông tin chip dùng khi mô phỏng. Cửa số Build chứa thông tin kết quả biên dịch. "Editor" là vùng viết chương trình, trong trường hợp này đó là file "avr1.asm" của bạn.



Hình 8: các cửa sổ lập trình

- Bạn viết đoạn code sau vào file avr1.asm của bạn (của sổ Editor) rồi nhấn nút công cụ "Save all" trên thanh công cụ để lưu Project.

```
.CSEG
.INCLUDE "M8DEF.INC"
.ORG 0x000
        RJMP BATDAU

.ORG 0x020
BATDAU:
; KHOI TAO CAC DIEU KIEN DAU
        LDI R16, HIGH(RAMEND)
        LDI R17, LOW(RAMEND)
        OUT SPH, R16
        OUT SPL, R17
        LDI R16, 0xFF;
        OUT DDRB, R16
```

```
; CHUONG TRINH CHINH
MAIN:
        LDI R16, 0B00000001
        OUT PORTB, R16
        RCALL DELAY

        LDI R16, 0B00000010
        OUT PORTB, R16
        RCALL DELAY

        LDI R16, 0B00000100
        OUT PORTB, R16
        RCALL DELAY

        LDI R16, 0B00001000
        OUT PORTB, R16
        RCALL DELAY

        LDI R16, 0B00010000
        OUT PORTB, R16
        RCALL DELAY

        LDI R16, 0B00100000
        OUT PORTB, R16
        RCALL DELAY

        LDI R16, 0B01000000
        OUT PORTB, R16
        RCALL DELAY

        LDI R16, 0B10000000
        OUT PORTB, R16
        RCALL DELAY

        RJMP MAIN
```

```
; CHUONG TRING CON DELAY 65535 chu ky (khoang 65535us neu  xung ;clock dung cho chip la 1MHz)
DELAY:
        LDI R20, 0xFF
        DELAY0:
                LDI R21, 0xFF
                DELAY1:
                        DEC R21
                        BRNE  DELAY1
                DEC R20
                BRNE DELAY0
RET
```

- Trước khi tìm hiểu ý nghĩa đoạn code, hãy nhìn 1 lượt qua đoạn code. Trước hết việc viết HOA hay viết thường là không quan trọng, bạn có thể viết đoạn code với bất cứ hình thức nào miễn đúng cú pháp, từ khóa là được. Trong đoạn code:
    i. Bạn thấy 1 số từ có màu BLUE (ví dụ LDI, OUT, RJMP, RCALL, RET…)đó là các INSTRUCTOR, tức là các câu lệnh của ngôn ngữ ASM, bạn có thể đọc tài liệu "AVR INSTRUCTION" để tìm hiểu tất cả các INSTRUCTION. Các INSTRUCTION sau đó sẽ được trình dịch dịch thành các mã tương ứng.
    ii. Một số từ bắt đầu bằng bằng dấu chấm **"."** là các **DIRECTIVE** (ví dụ **.INCLUDE** hay **.ORG**)đó cũng là những từ khóa mặc định của ASM AVR, các DIRECTIVE không phải là mã lệnh mà chỉ là các chỉ dẫn về địa chỉ bộ nhớ, khởi động bộ nhớ, định nghĩa macro…và không được trình dịch dịch thành mã. Chi tiết về DIRECTIVE có thể tìm thấy trong các tài liệu về ASM AVR, dưới đây tôi tóm tắt các DIRECTIVE và chức năng của chúng như sau:

Directive	Description
BYTE	Reserve byte to a variable
CSEG	Code Segment
CSEGSIZE	Program memory size
DB	Define constant byte(s)
DEF	Define a symbolic name on a register
DEVICE	Define which device to assemble for
DSEG	Data Segment
DW	Define Constant word(s)
ENDM, ENDMACRO	End macro
EQU	Set a symbol equal to an expression
ESEG	EEPROM Segment
EXIT	Exit from file
INCLUDE	Read source from another file
LIST	Turn listfile generation on
LISTMAC	Turn Macro expansion in list file on
NOLIST	Turn listfile generation off
ORG	Set program origin
SET	Set a symbol to an expression

iii. Thông thường 1 INSTRUCTION được theo sau bởi 2 toán hạng – operand (tuy nhiên có nhiều trường hợp chỉ có 1 toán hạng hoặc không có toán hạng), khi đó toán hạng thứ nhất sẽ là các **THANH GHI** của AVR (như đã đề cập, chúng ta sẽ khảo sát thanh ghi AVR trong các bài sau), ví dụ : "LDI R16, 0xFF;" trong đó toán hạng "R16" là tên 1 thanh ghi trong AVR, và "0xFF" là 1 hằng số dạng hexadecimal có giá trị tương ứng là 255 dạng thập phân hay 11111111 nhị phân

iv. Các từ theo sau bởi dấu ":" là các nhãn – label (ví dụ **MAIN, DELAY**…), đó là từ do chúng ta tự đặt, nó thực chất là 1 vị trí trong bộ nhớ chương trình, có thể sử dụng nhãn như 1 chương trình con.

v. Phần đi sau dấu ";" gọi là giải thích – comment, phần này không được biên dịch, bạn có thể ghi comment ở bất cứ đâu trong chương trình với yêu cầu phải sử dụng dấu ";" trước nó.

- Giải thích đoạn code: có thể chia đoạn code trên thành 4 phần: phần đầu chứa các DIRECTIVE và lệnh RJMP dùng để xác định các địa chỉ bộ nhớ chương trình, phần 2 là khởi tạo một số điều kiện đầu cho Stack Pointer và PORT, phần 3 là chương trình chính, và phần 4 là chương trình con (chú ý đây chỉ là cách bố trí của riêng tôi, một khi đã quen thuộc, bạn có thể bố trí chương trình theo cách riêng của bạn).

i. Phần 1 và phần 2:

```
. CSEG
```

Chỉ thị **.CSEG: Code Segment** báo cho trình biên dịch rằng phần code theo sau là phần chương trình thực thi, phần này sẽ được download vào bộ nhớ chương trình của chip

```
.INCLUDE "M8DEF.INC"
```

Chỉ thị **.INCLUDE** báo cho trình biên dịch bắt đầu đọc 1 file đính kèm, trong trường hợp trên là file "M8DEF.INC", đây là file chứa các khai báo cho chip Atmega8 như thanh ghi, ngắt…cho việc truy xuất trong chương trình của bạn, đây là dòng bắt buộc, nếu bạn lập trình cho chip khác bạn hãy đổi tên file đính kèm, ví dụ "m32def.inc" cho chip ATmega32… bạn có thể tìm thấy các file này trong thư mục "C:\Program Files\Atmel\AVR Tools\AvrAssembler2\Appnotes".

```
.ORG 0x000
```

Chỉ thị **.ORG:** Set Program Origin, set vị trí trong bộ nhớ sẽ được tác động đến, trong trường hợp trên, .ORG 0x000 xác định phần code theo ngay sau sẽ nằm ở địa chỉ 000, vị trí đầu tiên, trong bộ nhớ chương trình. Và dòng lênh trong vị trí đầu tiên đó là:

```
RJMP BATDAU
```

RJMP: Relative Jump là lệnh nhảy không điều kiện đến 1 vị trí trong bộ nhớ, trong trường hợp trên là nhảy đến nhãn BATDAU, và nhãn BATDAU nằm ở vị trí 0x020 (số hexadecimal, 0x020 =32 decimal) vì nó được khai báo ngay sau DIRECTIVE .ORG 0x020

```
.ORG 0x020
BATDAU:
```

Như thế phần bộ nhớ chương trình nằm giữa 0 và 0x020 không được sử dụng trong đoạn code của chúng ta, phần này được sử dụng cho mục đích khác, đó là các vectơ ngắt (không được đề cập ở đây). Tiếp theo:

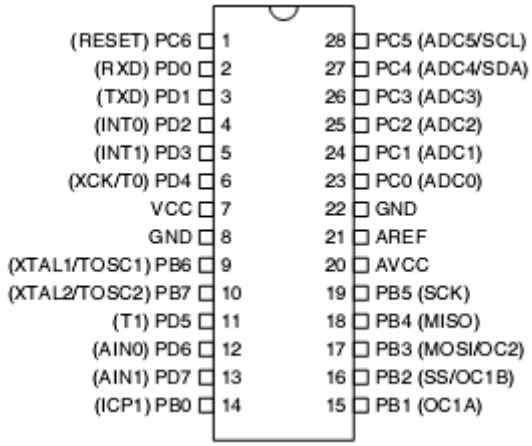
```
; KHOI TAO CÁC DIEU KIEN DAU
    LDI R16, HIGH(RAMEND)
    LDI R17, LOW(RAMEND)
    OUT SPH, R16
    OUT SPL, R17
```

Bốn dòng code trên khởi tạo cho Stack Pointer, chúng ta sẽ tìm hiểu phần này trong các bài về Stack và chương trình con.

Lời khuyên: các bạn nên khởi động 1 chương trình theo cách trên và chúng ta sẽ hiểu chúng rõ hơn sau này !

```
    LDI R16, 0xFF
    OUT DDRB, R16
```

Bạn chú ý 2 dòng trên và những gì tôi giải thích sau đây, 2 dòng này có tác dụng khởi động PORTB của chip ATmega8 tác dụng như các ngõ xuất tín hiệu (OUTPUT). Trước hết hãy quan sát chip ATmega8 trong hình sau



Hình 9: chip ATmega8

Bạn có thể thấy chip này gồm 28 chân, trông đó có các chân được ghi là PB0(chân 14), PB1(chân 15),…,PB7(chân 10), đó là các chân của PORTB. PORT là khái niệm chỉ các ngõ xuất nhập. Trong AVR, PORT có thể giao tiếp theo 2 hướng (bi – directional), có thể dùng để xuất hoặc nhận thông tin, mỗi PORT có 8 chân. Chip Atmega8 có 3 PORT có tên tương ứng là PORTB, PORTC và PORTD (một số chip AVR khác có 4 hoặc 6 PORT). PORT được coi là "cửa ngõ" then chốt của vi điều khiển.

Trong AVR, mỗi PORT liên quan đến 3 thanh ghi (8 bits) có tên tương ứng là DDRx, PINx, và PORTx với "x" là tên của PORT, mỗi bit trong thanh ghi tương ứng với mỗi chân của PORT. Trong trường hợp của Atmega8 "x" là B, C hoặc D. Ví dụ chúng ta quan tâm đến PORTB thì 3 thanh ghi tương ứng có

tên là DDRB, PINB và PORTB, trong đó 2 thanh ghi PORTB và PINB được nối trực tiếp với các chân của PORTB, DDRB là thanh ghi điều khiển hướng (Input hoặc Output). Viết giá trị 1 vào một bit trong thanh ghi DDRB thì chân tương ứng của PORTB sẽ là chân xuất (Output), ngược lại giá trị 0 xác lập chân tương ứng là ngõ nhập. Sau khi viết giá trị điều khiển vào DDRB, việc truy xuất PORTB được thực hiện thông qua 2 thanh ghi PINB và PORTB.

Quay lại với 2 dòng code của chúng ta, dòng đầu: "LDI R16, 0xFF", với LDI – LoaD Immediately, dòng lệnh có ý nghĩa là load giá trị 0xFF vào thanh ghi R16, R16 là tên 1 thanh ghi trong bộ nhớ của AVR, 0xFF là 1 hằng số có dạng thập lục phân, ký hiệu "0x" nói lên điều đó, bạn cũng có thể dùng ký hiệu khác là "$" để chỉ 1 số thập lục phân, ví dụ &FF, và 0xFF=255(thập phân)=0B11111111 (nhị phân). Như thế sau dòng đầu thanh ghi R16 có giá trị là 11111111 (nhị phân). Dòng thứ 2: "OUT DDRB, R16" nghĩa là xuất giá trị từ thanh ghi R16 ra thanh ghi DDRB, tóm lại sau 2 dòng trên giá trị DDRB như sau:

1	**1**	**1**	**1**	**1**	**1**	**1**	**1**

Có thể bạn sẽ hỏi tải sao chúng không sử dụng 1 dòng duy nhất là "LDI DDRB, 0xFF" hay "OUT DDRB, 0xFF", chúng ta không thể vì lệnh LDI chỉ cho phép thực hiện trên các thanh ghi R16,…R31 và lệnh OUT không thực hiện được với các hằng số.

Và vì DDRB=11111111 nên trong trường hợp này tất cả các chân của PORTB đã sẵn sàng cho việc xuất dữ liệu. Lúc này thanh ghi PINB không có tác dụng, thanh ghi PORTB sẽ là thanh ghi xuất, ghi giá trị vào thanh ghi này sẽ tác động đến các chân của PORTB.

ii. Phần 3: chương trình chính

```
MAIN:
    LDI R16, 0B00000001
    OUT PORTB, R16
    RCALL DELAY
```

Bạn chỉ cần chú ý 4 dòng trên trong toàn bộ phần chương trình chính, trước hết "MAIN:" chỉ là 1 nhãn do chúng ta tự đặt tên, giống như 1 "cột mốc" trong chương trình thôi. Dòng "LDI R16, 0B00000001" thì bạn đã hiểu, chỉ có 1 khác biệt nhỏ là tôi sử dụng hằng số dạng nhị phân cho bạn dễ hiểu hơn. Và dòng "OUT PORTB, R16" để xuất giá trị 0B00000001 có sẵn trong R16 ra thanh ghi PORTB, lúc này chân PB0 của chip sẽ lên 1 (5V) và các chân còn lại sẽ ở mức 0 (0V). Dòng thứ 3: "RCALL DELAY" là lệnh gọi chương trình con DELAY, tạm hoãn trước khi thực hiện các dòng lệnh tiếp theo:

```
    LDI R16, 0B00000010
    OUT PORTB, R16
    RCALL DELAY
```

Ba dòng lệnh này cũng giống ba dòng trên, nhưng giá trị xuất ra lúc này là 0B00000010, chân PB1 sẽ lên 5V và các chân khác xuống mức 0V. Và cứ như thế đến đoạn cuối:

```
LDI R16, 0B10000000
OUT PORTB, R16
RCALL DELAY
RJMP MAIN
```

Sau khi kết thức 3 dòng trên chân PB7 sẽ lên 5V, kết thúc 1 vòng xoay. Cuối cùng là quay vế đầu chương trình chính bằng dòng "RJMP MAIN"

Bây giờ chắc bạn đã đoán được chương trình của chúng ta thực hiện việc gì, đó là quét xoay vòng các chân của PORTB, nếu chúng ta kết nối các chân của PORTB với các LED, chúng ta sẽ có 1 hiệu ứng quét LED xoay vòng, chúng ta thực hiện điều này bằng phần mềm Proteus.

iii. Phần 4 - chương trinh con DELAY: đoạn chương trình này không làm gì cả ngoài việc trì hoãn 1 khoảng thời gian, tuy nhiên bạn chưa thể hiểu nó ngay được.

Đây chỉ là 1 ví dụ đơn giản, tôi cố gắng thực hiện nó theo cách dễ hiểu nhất cho bạn, vì thế đoạn code có vẻ hơi dài dòng, bạn hãy thực hiện lại đoạn chương trình chính bằng đoạn code của bạn.

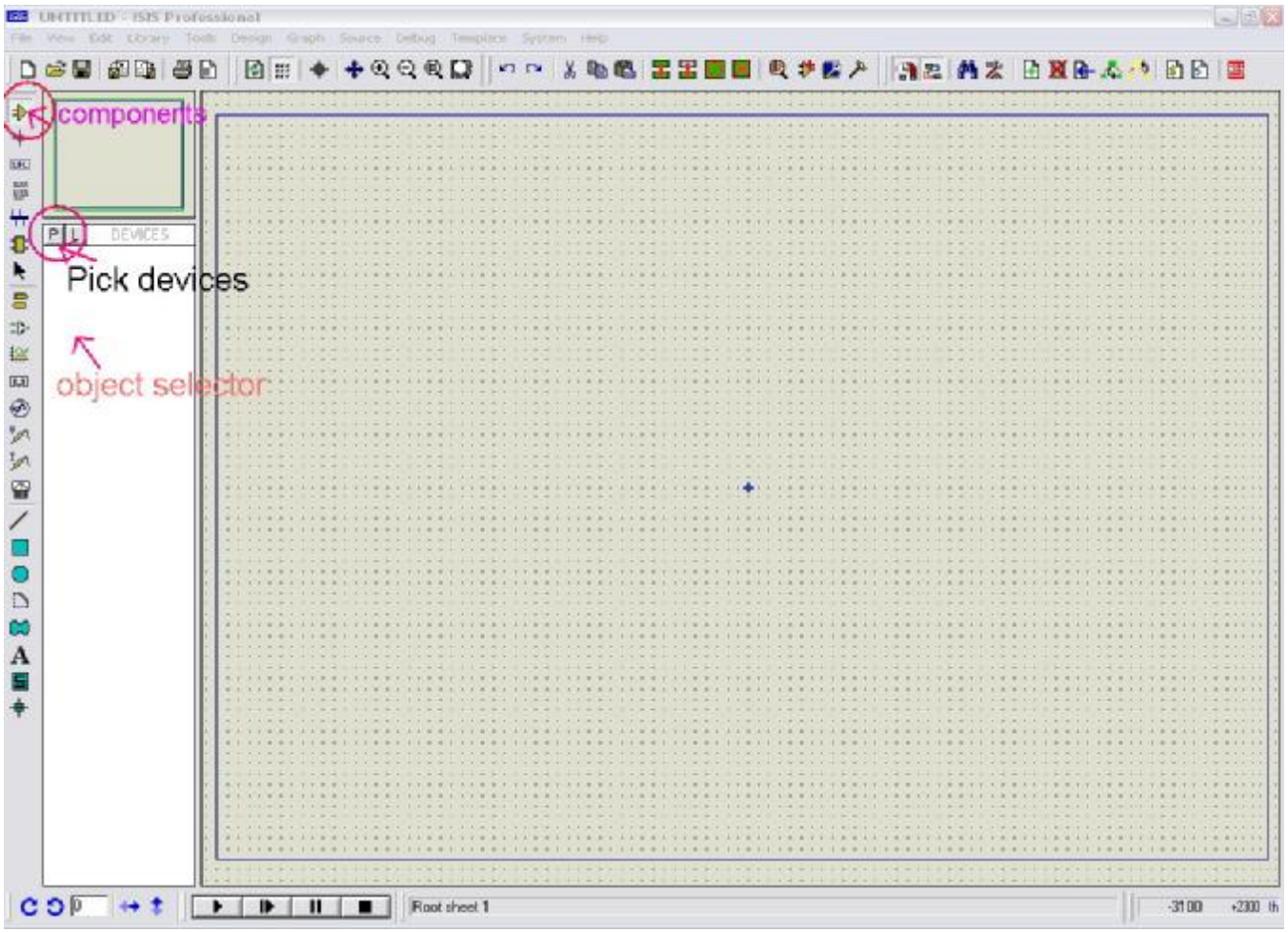
Phần cuối cùng là biên dịch đoạn code thành file intel hex để đổ vào chip, nhấn phím F7 để biên dịch.

Sau khi biên dịch bạn sẽ có 1 file tên "avr1.hex" trong thư mục project, chúng ta sẽ dùng file này đổ vào chip sau này

IV. Mô phỏng ví dụ của chúng ta bằng Proteus

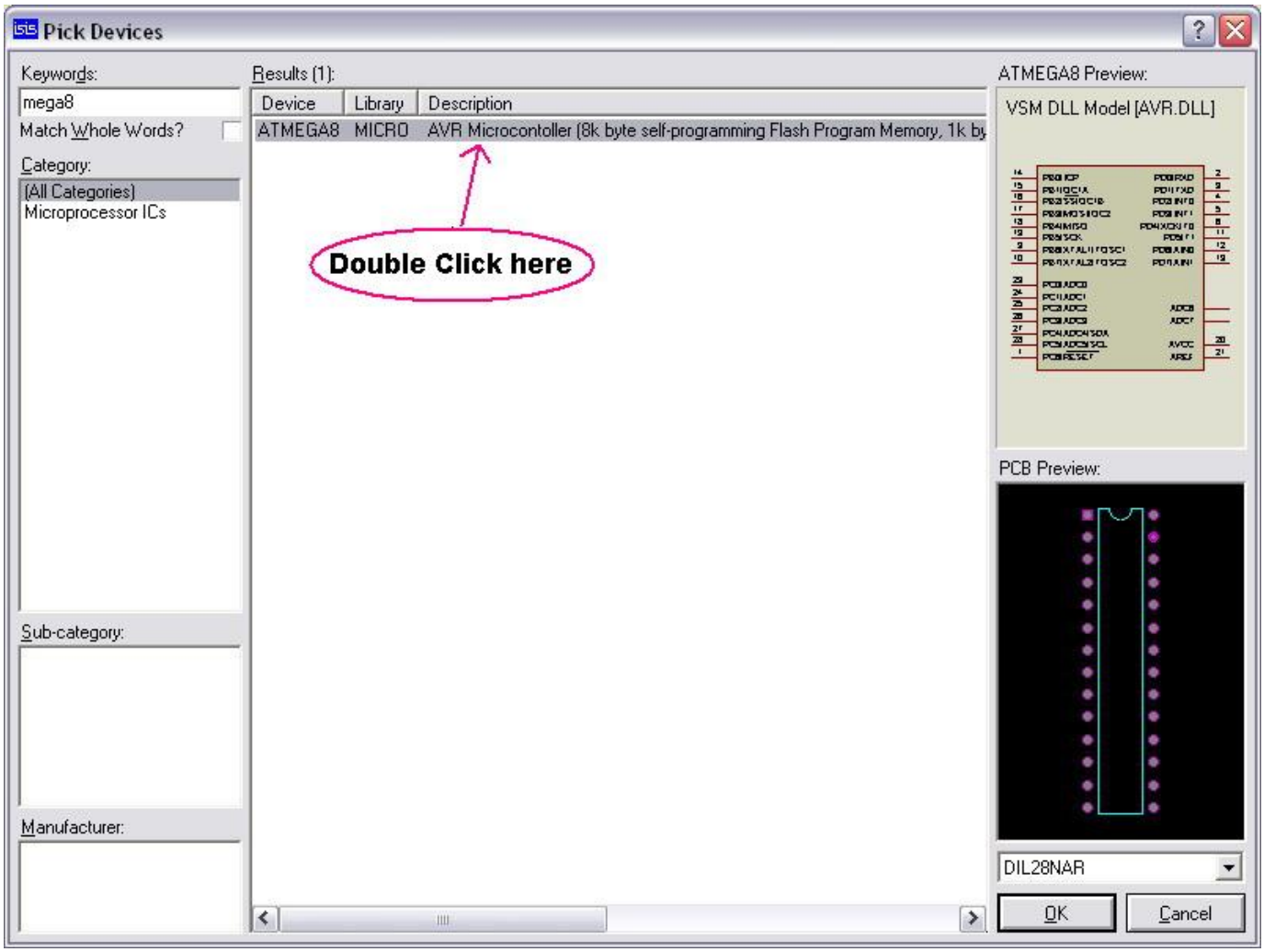
Chúng ta hãy thử nghiệm đoạn chương trình của chúng ta bằng Proteus

- **Chạy Proteus:** sau đó nhấn vào buton "Comonents" rồi "Pick Devices" để chọn linh kiện



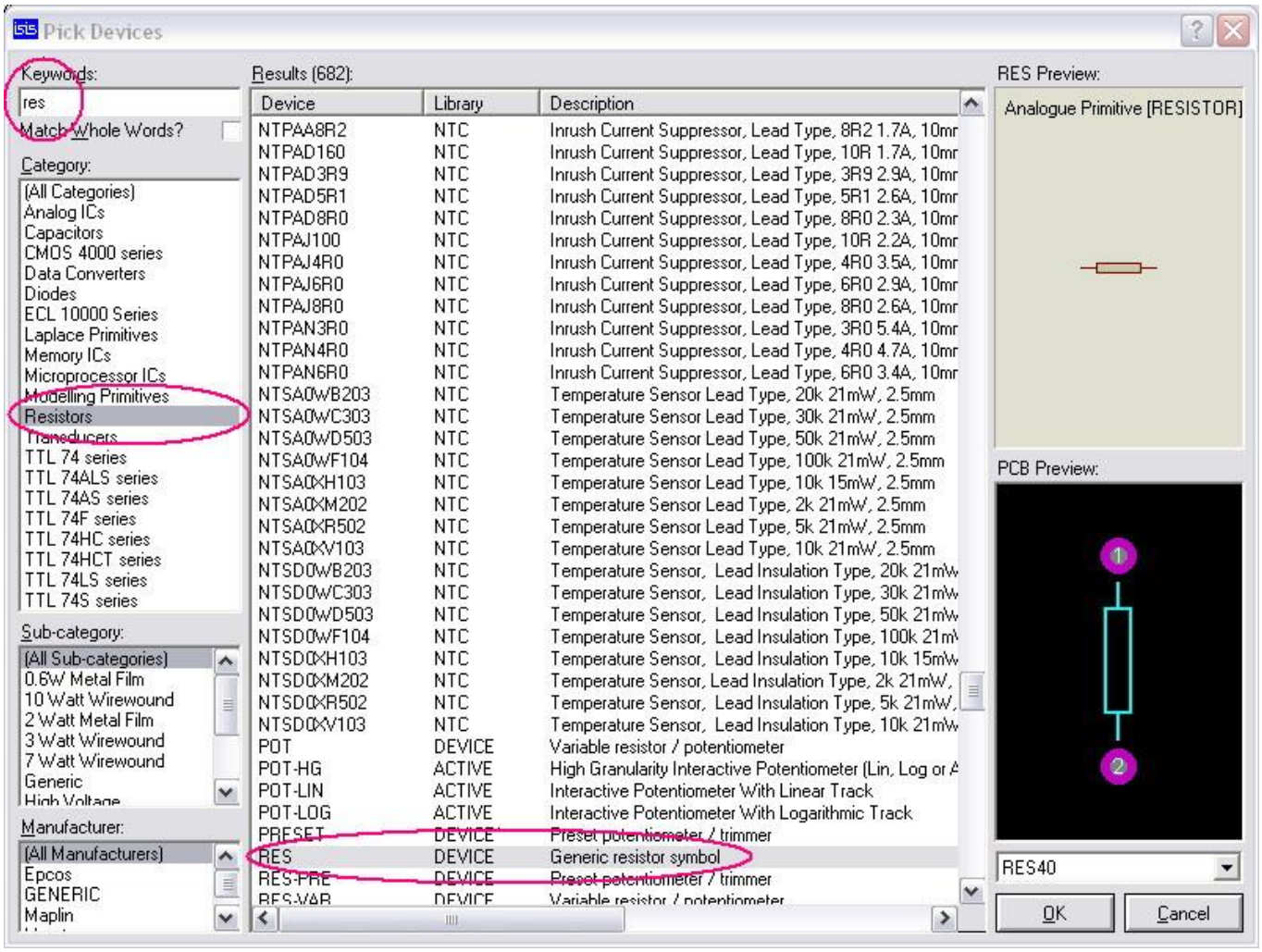
Hình 10: giao diện Proteus

- **Chọn linh kiện:** trong dialog Pick Divices, ô "Keywords" nhập mega8, bạn sẽ thấy 1 linh kiện có tên "ATMEGA8" bên cửa sổ "Results", double click vào linh kiện đó để mang nó ra cửa sổ "Object selector".



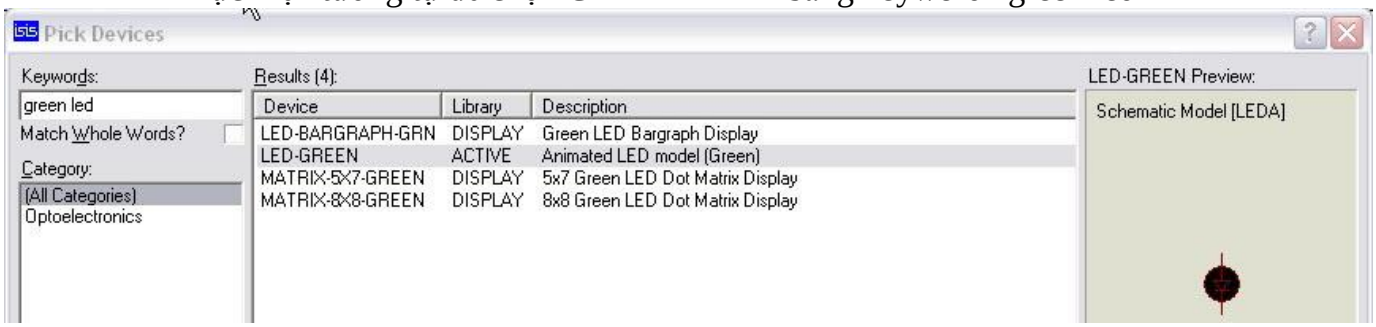
Hình 11: Pick devices

- Để tìm điện trở, bạn đánh keyword “res”, chọn “Resistors” trong “category” và Double click vào link kiện “RES” trong ô “Results”



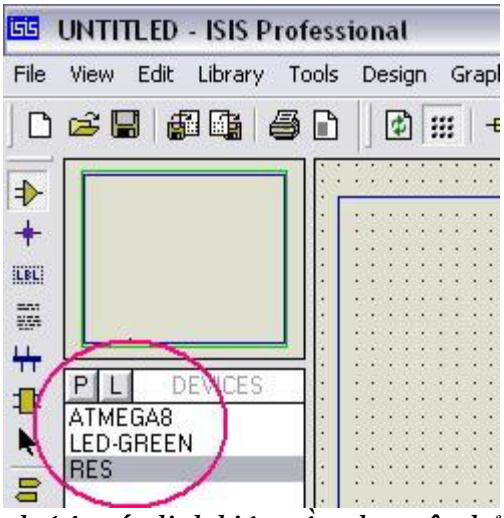
Hình 12: chọn điện trở

- Thực hiện tương tự để chọn GREEN - LED bằng keyword “green led”



Hình 13: chọn Green LED

- Sau khi chọn 3 loại linh kiện cần thiết bạn hãy nhấn OK và quay về cửa số chính, khi đó bạn thấy trong cửa sổ “Object selector” như sau:



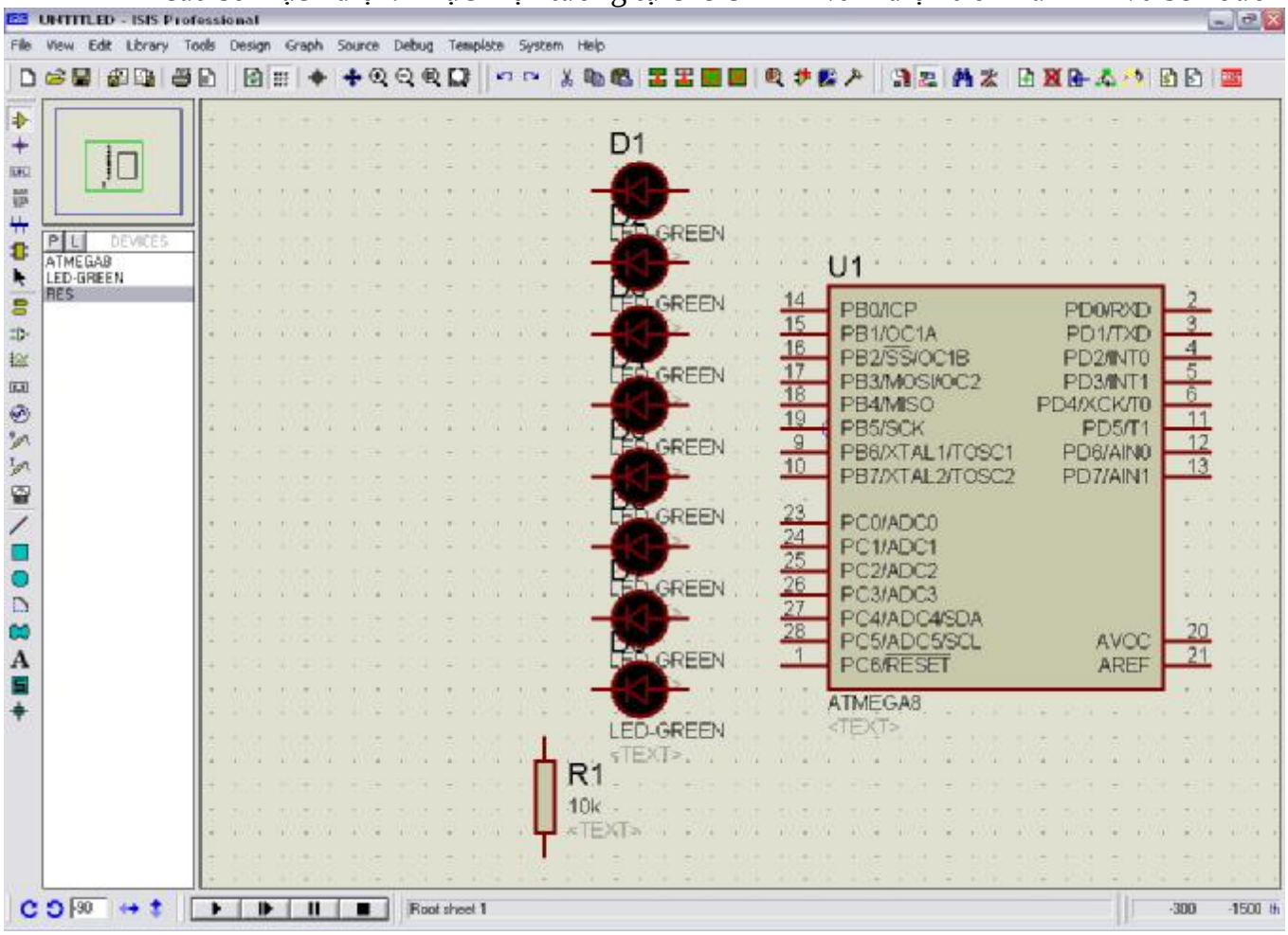
Hình 14: các link kiện cần cho mô phỏng

- Thao tác với mouse trong Proteus: khác với 1 số chương trình vẽ mạch điện khác, tháo tác mouse trong Proteus hơi lạ nên có thể gây bối rối cho bạn, hãy theo hướng dẫn sau đây:
    - Chọn linh kiện để vẽ: left – click lên tên linh kiện trong cửa sổ "Object selector"
    - Đặt linh kiện: **Left – click** lên cửa sổ mạch điện **Right click** lên linh kiện trong cửa sổ mạch điện sẽ làm cho linh kiện đó được bao bởi màu "đỏ", tức bạn đang chọn linh kiện đó.
    - Bỏ chọn linh kiện thực hiện bằng cách Right – click lên một vị trí trống trên cửa sổ mạch điện.
    - Delete linh kiện: Right – click 2 lần lên 1 linh kiện là delete linh kiện đó khỏi cửa sổ mạch điện, hoặc Right click 1 lần lên 1 linh kiện đã được chọn trước đó (có màu đỏ) cũng sẽ xóa linh kiện này.
    - Di chuyển linh kiện: chọn linh kiện trước (right – click) và drag để di chuyển linh kiện bằng mouse left
    - Xoay và lật linh kiện: chọn linh kiện cần xoay hay lật (right – click), dùng các nút công cụ để xoay hoặc lật linh kiện.

Hình 15: các nút công cụ xoay và lật linh kiện

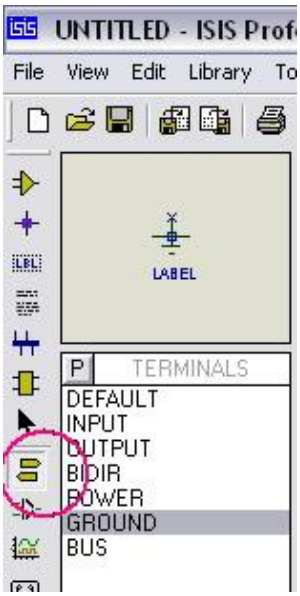
    - Hiện cửa sổ thuộc tính linh kiện: rất nhiều khi bạn cần thay đổi 1 số thuộc tính của linh kiện (ví dụ giá trị của điện trở), bạn thực hiện điều này trong cửa sổ thuộc tính của linh kiện. Để hiện cửa sổ thuộc tính của 1 linh kiện bạn hãy right – click trước (để chọn linh kiện – linh kiện sẽ đỏ lên) và sau đó left – click sau.

- Theo hướng dẫn trên, bạn hãy click vào ATMEGA8 và đặt linh kiện này lên mạch điện của bạn (đặt lên cửa sổ làm việc lớn) bằng cách left - click lên bất vị trí nào trên cửa sổ mạch điện. Thực hiện tương tự cho 8 LED và 1 điện trở như hình vẽ bên dưới



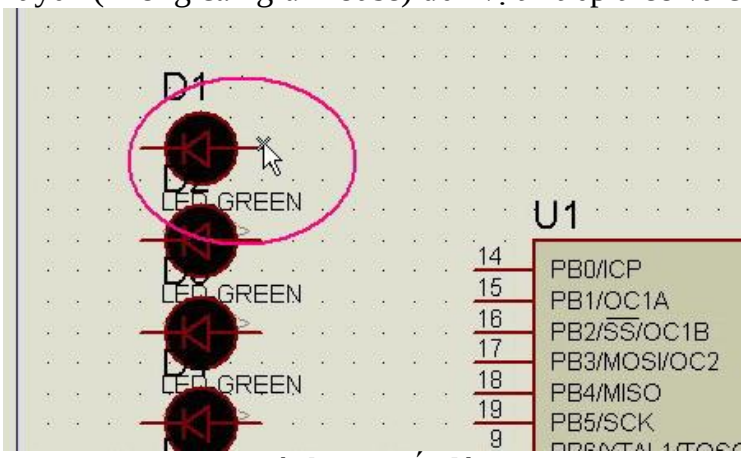
Hình 16: đặt linh kiện lên mạch điện

- Tiếp theo là đặt "mass" cho LED, nhấn vào nút công cụ "Inter – sheet Terminal" như hình bên dưới:



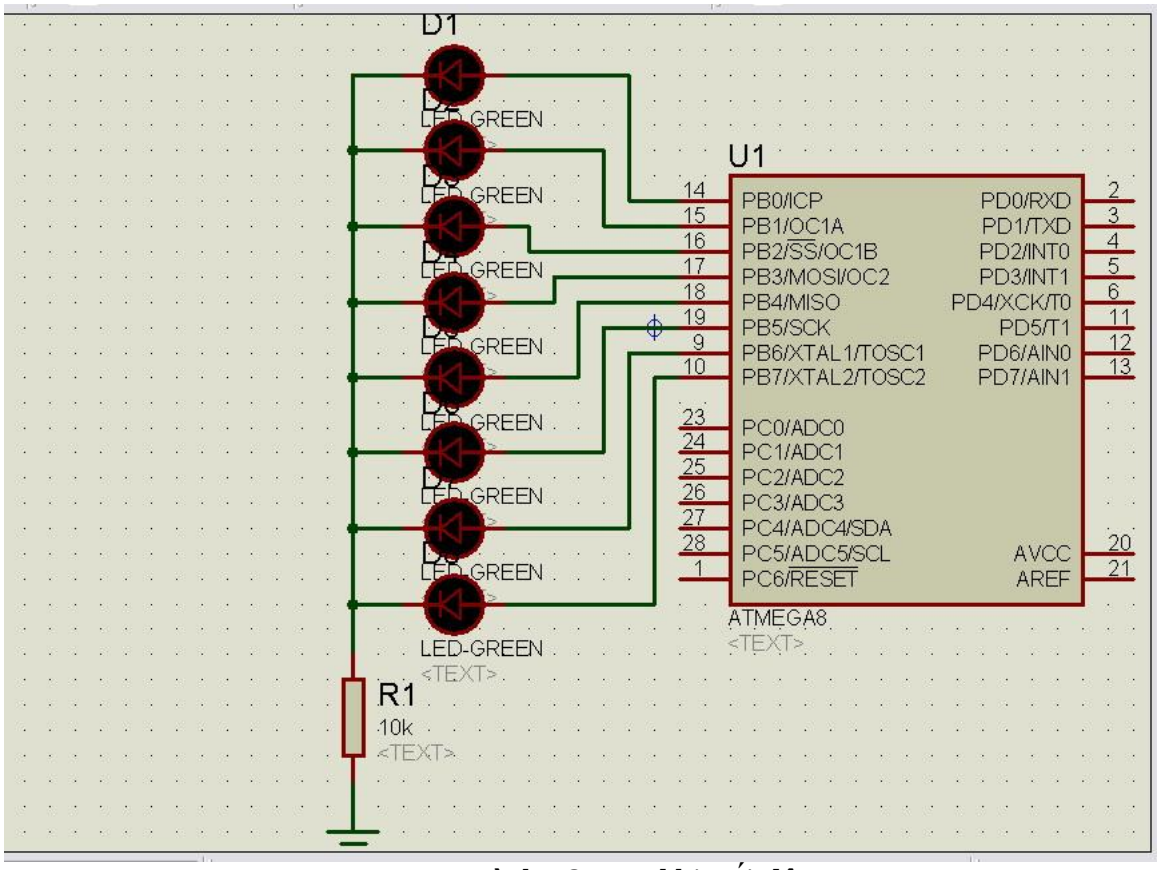
Hình 17: Nút công cụ Inter – sheet Terminal

- Bạn thấy trong cửa sổ "Object devices" có 1 số thiết bị, hãy chú ý dến "POWER" và "GROUND", đây là nguồn và mass cho mạch điện của bạn. Hãy chọn GROUND và đặt lên mạch điện của bạn.
- Bước tiếp theo, nối dây: không cần công cụ, để nối dây bạn chỉ cần rê mouse đến điểm cần nối của linh kiện, bạn sẽ thấy xuất hiện 1 dấu chéo "x", lúc đó hãy click mouse và di chuyển (không cần giữ mouse) đến vị trí tiếp theo và click lần nữa



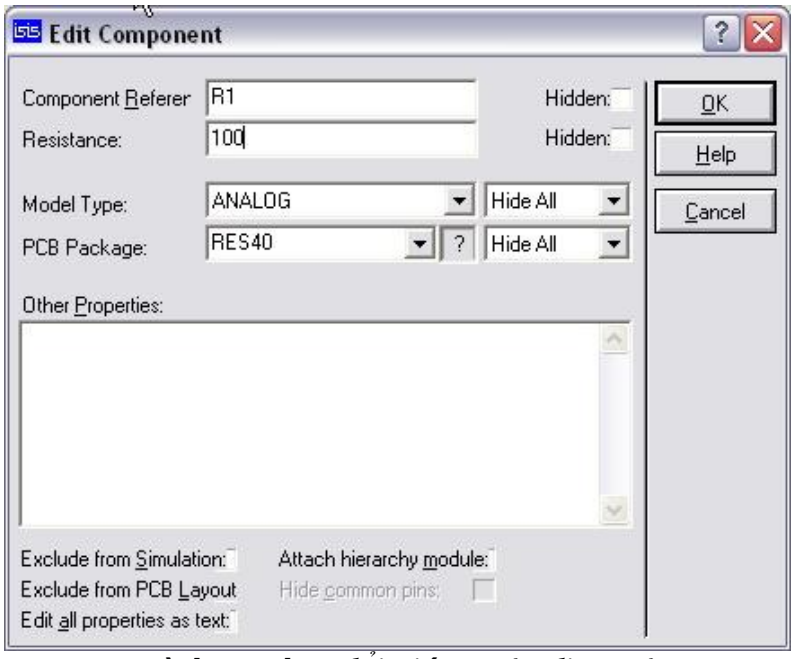
Hình 17: nối dây

- Theo cách này bạn hãy nối dây cho mạch điện của bạn, mạch điện hoàn chỉnh như sau:



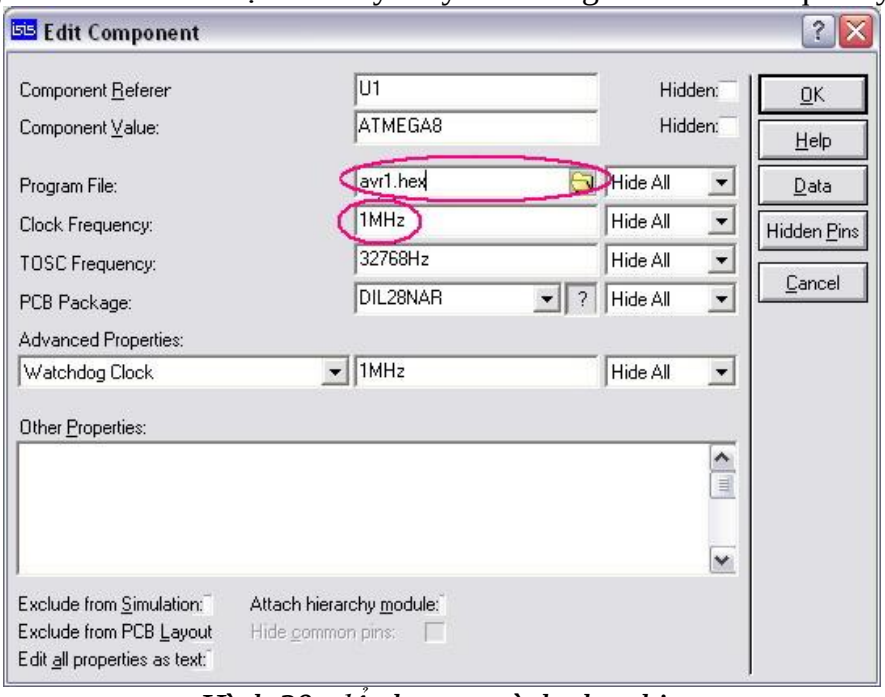
Hình 18: sau khi nối dây

- Bây giờ hãy thay đổi giá trị của điện trở, giá trị mặc định là 10k, giá trị này quá lớn, dòng điện sẽ rất nhỏ, khi mô phỏng bạn sẽ không thấy các LED sang lên. Bạn hãy tahy đổi nó thành 100 (100 Ohm). Trước hết cho hiện cửa sổ thuộc tính của điện trở (right click rồi left click lên điện trở), thay đổi ô resistance của nó:



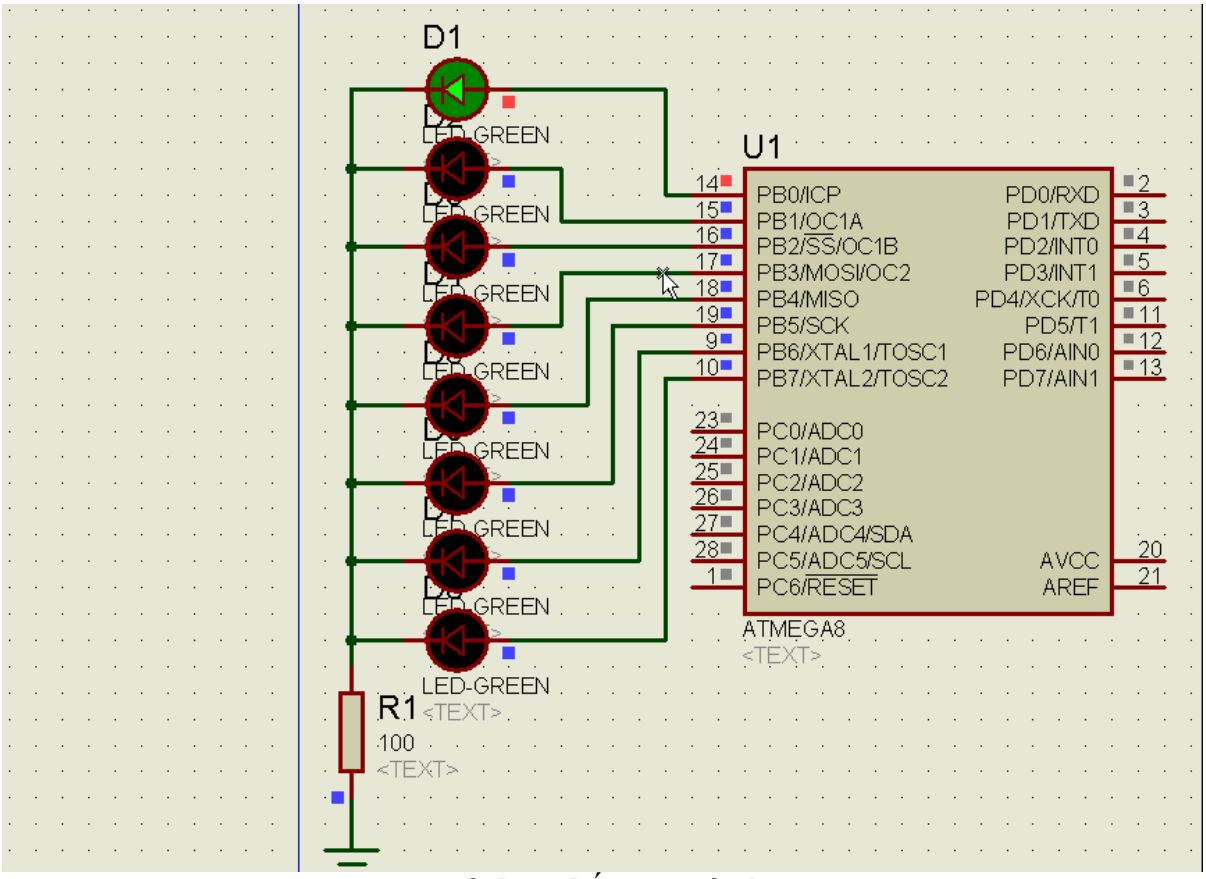
Hình 19: thay đổi giá trị của điện trở

- **Đổ chương trình vào chip Atmega8**: hãy hiện cửa sổ thuộc tính của chip Atmega8, trong ô "Program file" hãy click và tìm đến file "avr1.hex" mà bạn đã tạo trong thư mục Project sau khi biên dịch. Chú ý thay đổi thông số "Clock frequency" là 1 Mhz



Hình 20: đổ chương trình cho chip

- Hãy lưu mạch điện của bạn.
- Đây là việc cuối cùng, chạy mô phỏng, sử dụng thanh công cụ Play để chạy mô phỏng mạch điện của bạn, kết quả như sau:



Hình 21:kết quả mô phỏng

Chúng ta kết thúc bài 1 ở đây, các bạn quan tâm hãy đọc thật kỹ, mong rằng bài viết này giúp cho bạn có cái nhìn ban đầu về AVR, Trong các bài tiếp theo chúng ta sẽ tìm hiểu chi tiết hơn về AVR, chúc các bạn thành công.