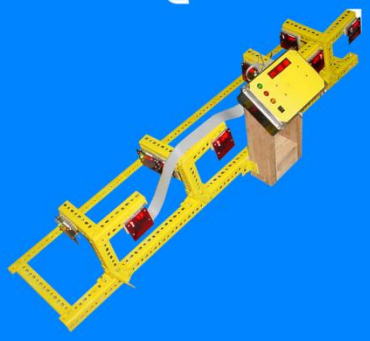
کنترلر آسانسور چهار طبقه

http://youtu.be/dqKeHD5WNcc

کنترلر چراغ راهنمایی



http://youtu.be/0ADfFPOzIUE

کنترلر پارکینگ

http://youtu.be/51oqLRxXcHk

کنترلر حرارت کوره رنگرزی

http://youtu.be/mJfT4z1oCeo

PLCGoods.net

دانلود برنامه و شبیه سازی پروژه های صنعتی با PLC و میکروکنترلر

آموزش کار با میکروکنترلرهای AVR

به همراه آموزش زبان C

تحت نرم افزار Codevision

نویسنده

محمد کرامتی

به سفارش

www.plcgoods.net

دانلود رایگان مقالات و پروژه های صنعتی از سایت های:

www.plcgoods.net
www.plcgoods.com
www.plc-doc.com

فهرست

مقدمه

میکروپروسسور و میکروکنترلر ها

شاید تاکنون نام میکرو کنترلر را شنیده باشید. آی سی های میکرو، آی سی ها ی قابل برنامه ریزی هستند که با نوشتن برنامه، کار مورد نظر را برای ما انجام می دهند .

میکرو پروسسور نیز نمونه ی ساده تری از میکروکنترلرها است. به این صورت که میکرو پروسسور تقریباً یک ROM تنها می باشد که برنامه درون آن ریخته و اجرا می شود، اما برای اجرای عملیات نیازمند قطعات جانبی می باشند که باعث پچیده شدن مدار و سخت تر شدن طراحی آن می شود. به همین علت در این مبحث به میکرو کنترلر ها پرداخته ایم که امکانات بیشتر و جامعیت وسیعتری دارند. آی سی های میکرو دارای انواع گوناگونی هستند که به تعدادی از آنها اشاره شده است:

AVR (Atmel) 6811 (موتورولا) 8051 (Atmel) Z8_3 (زایلوگ) PIC 16X (میکروچیپ)

از بین آی سی های فوق ، خانواده ی AVR علاوه بر سادگی، امکانات بالایی نیز دارد. البته خانواده های دیگر از قبیل PIC و ARM نیز امکانات زیاد و سرعت بالایی دارند ولی میکروکنترلر AVR علاوه بر داشتن امکانات کافی، قیمت نسبتاً پایینی نیز دارد . این آی سی ها در سه گروه tiny، 90s و mega ساخته شده اند(البته در سالهای اخیر سری Xmega نیز به بازار ایران عرضه شده است) . امکانات و قیمت این چهار گروه به ترتیب زیر است:

Tiny < 90s < mega < Xmega

با مراجعه به کاتالوگ هر یک از این آی سی ها می توان درباره امکانات و سرعت آنها اطلاعاتی بدست آورد. به طور کلی بالاترین سرعتی که میکروهای AVR دارند 16MHz است.

همانطور که گفتیم، اجرای توابع به کمک میکرو نیازمند برنامه ریزی آنهاست. زبانی که برای این آی سی ها قابل فهم می باشد، کد است. اگر ما بخواهیم برنامه ها را به صورت کد بنویسیم، هم زمان زیادی می برد و هم خطای کار زیاد است. برای این کاراز نرم افزارهایی که زبان قابل فهم برای انسان را به زبان قابل فهم برای آی سی تبدیل می کنند، استفاده می شود. به این نرم افزار ها "کامپایلر"(مبدل) می گویند. امروزه کامپایلر های مختلفی وجود دارد که هر کدام با یک یا دو زبان برنامه نویسی کار می کنند، مانند زبان اسمبلی، C و Basic. در ادامه زبان بیسیک و C را که در میکرو کاربرد بیشتری دارند، با هم مقایسه می کنیم:

زبان بیسیک

محاسن: سادگی، سرعت بسیار بالا، حجم کد تولید شده بسیار پایین

معایب: در برنامه های بزرگ کار با آن بسیار مشکل است

زبان C

محاسن: برنامه های هزار خطی نیز در آن انجام می پذیرد

معایب: سرعت کمتر نسبت به بیسیک، حجم کد تولید شده بالاتر، یک خط زبان C گاهی نیاز به 2 KB کد ماشین دارد

با جمع بندی موارد بالا و با توجه به اینکه برنامه های ما اکثراً سنگین هستند زبان C را برای برنامه نویسی این آی سی ها انتخاب کرده ایم. نرم افزار Codevision یک کامپایلر زبان C برای AVR است. در این مبحث با این برنامه بیشتر آشنا می شوید.

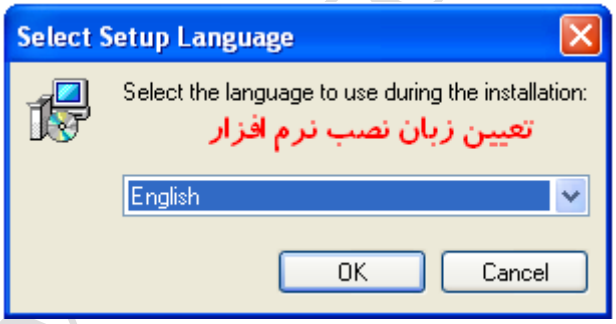
با تشکر ویژه از استاد ابراهیم زارعی که در تبیین این مطالب برای بنده نقش مؤثری داشته اند.

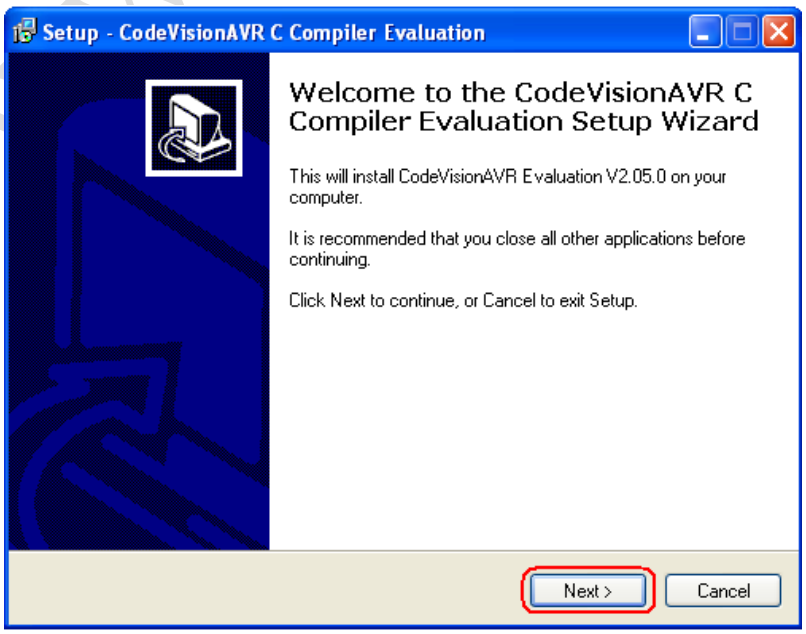
فصل اول

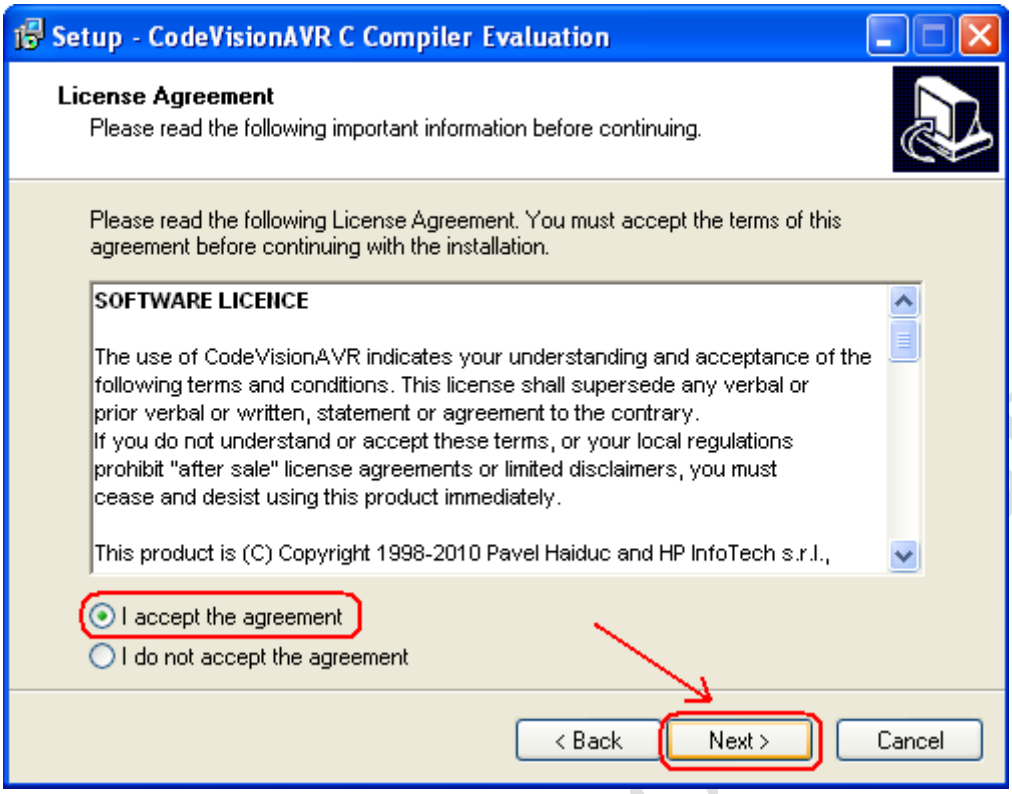
نصب نرم افزار Codevision

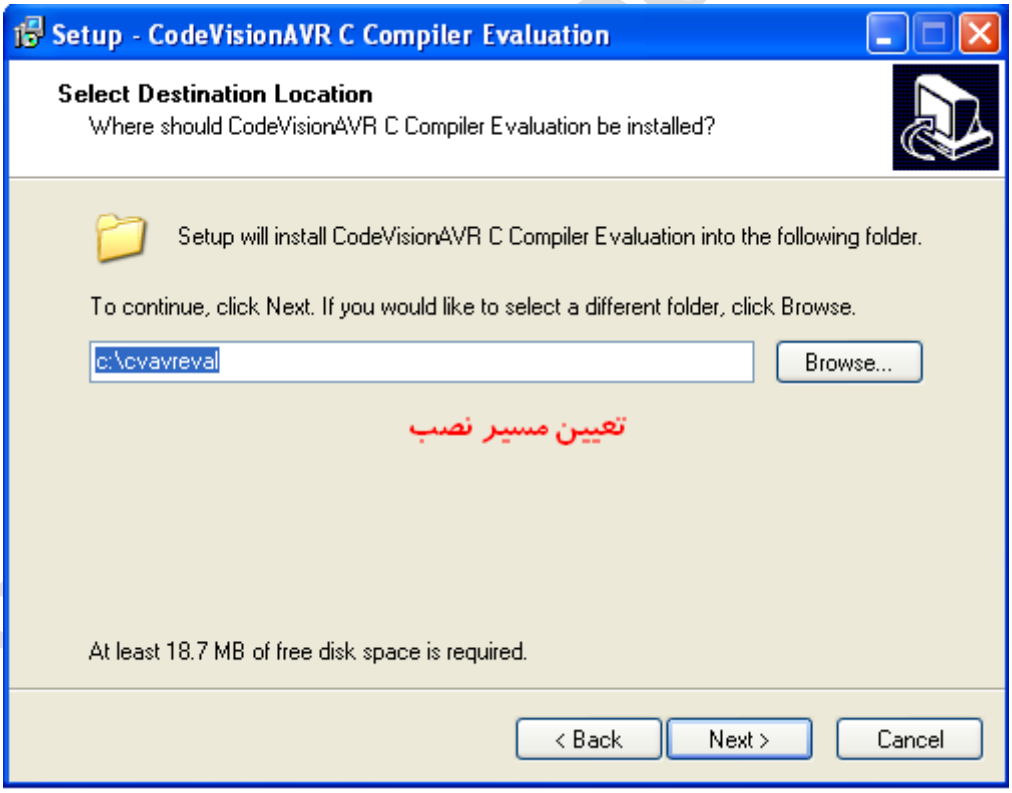
نصب برنامه ی Codevison به سادگی صورت می پذیرد. مراحل نصب به صورت مصور در زیر نشان داده می شود:

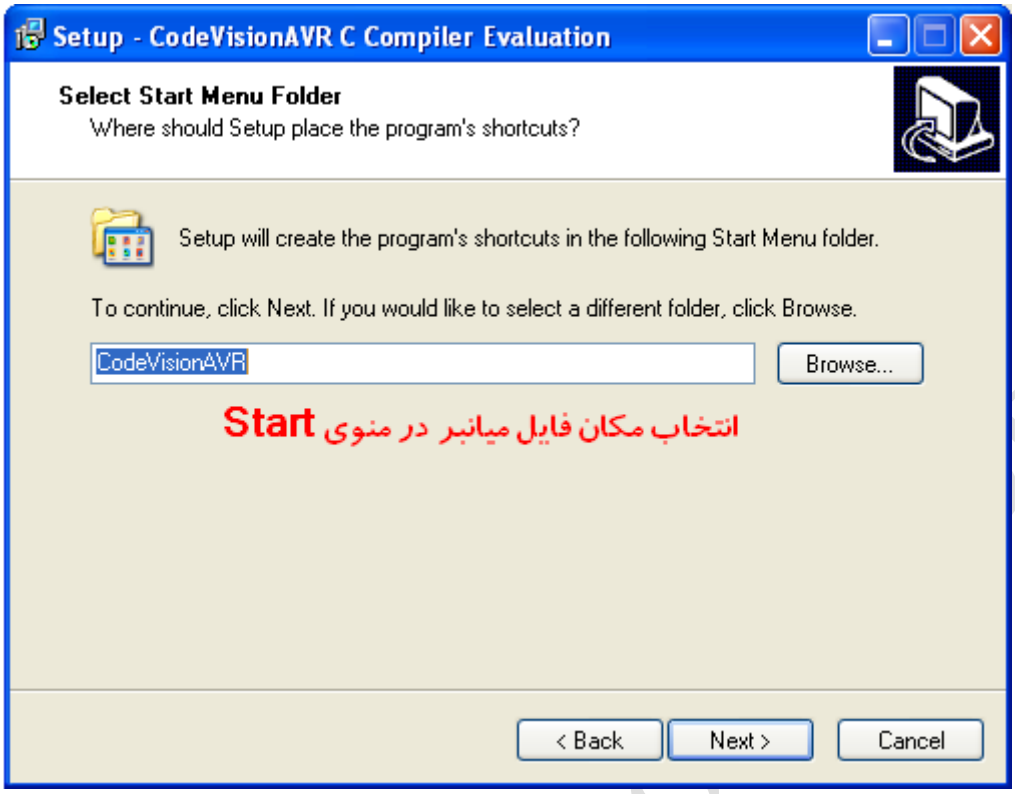
ابتدا بر روی فایل CodeVisionAVR 2.05.0 Professional.exe کلیک می کنیم.

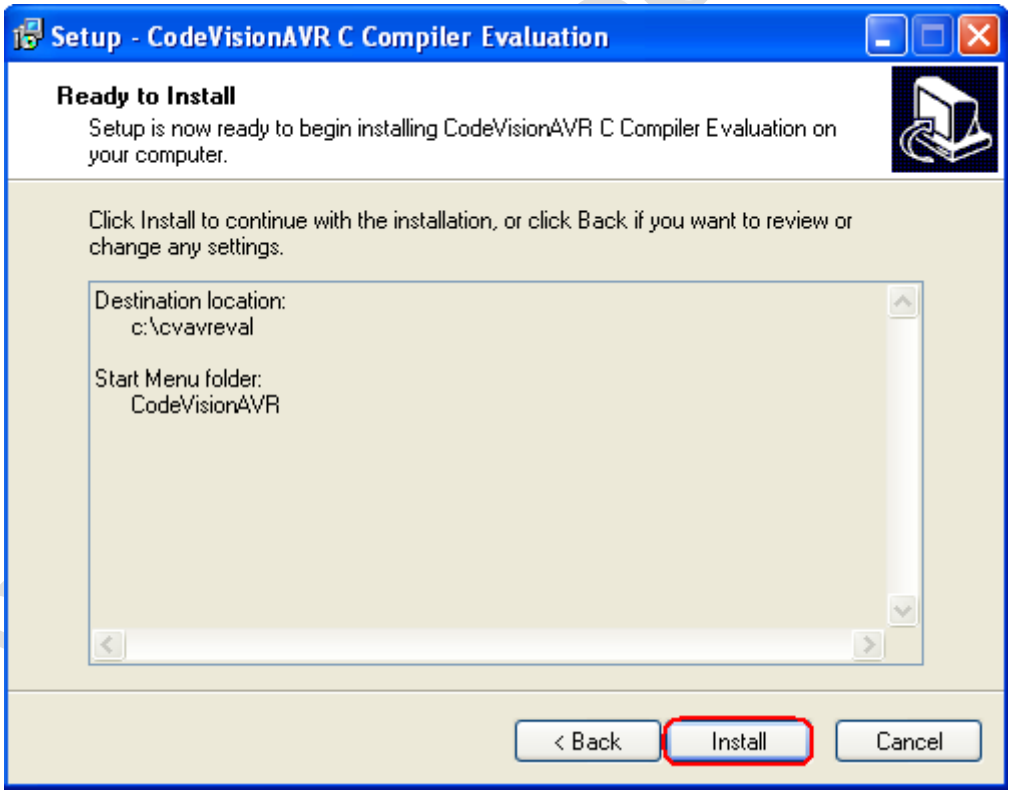


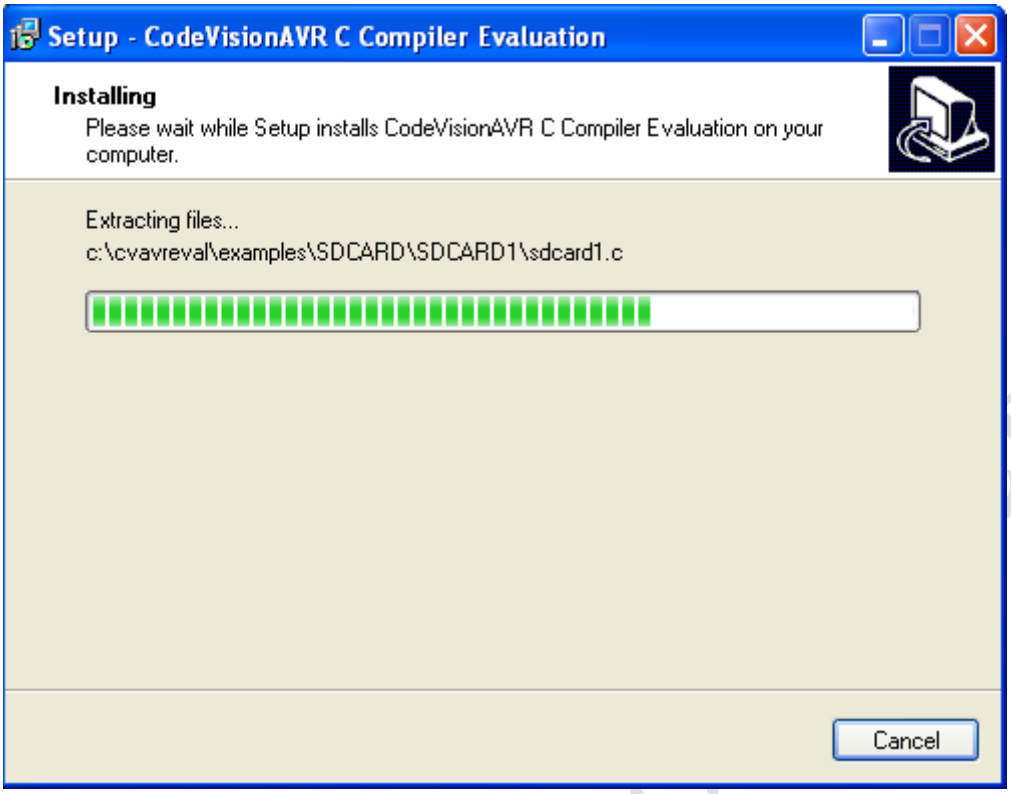


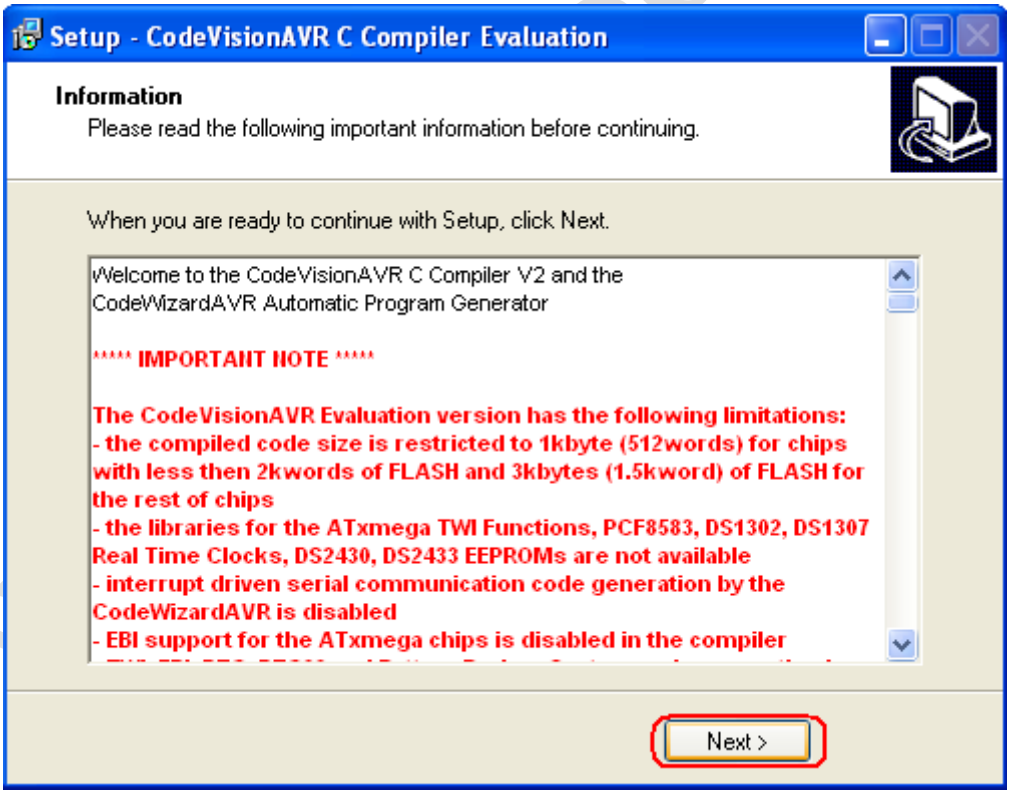

Setup - CodeVisionAVR C Compiler Evaluation
License Agreement
Please read the following important information before continuing.
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.
SOFTWARE LICENCE
The use of CodeVisionAVR indicates your understanding and acceptance of the following terms and conditions. This license shall supersede any verbal or prior verbal or written, statement or agreement to the contrary.
If you do not understand or accept these terms, or your local regulations prohibit "after sale" license agreements or limited disclaimers, you must cease and desist using this product immediately.
This product is (C) Copyright 1998-2010 Pavel Haiduc and HP InfoTech s.r.l.,
I accept the agreement
I do not accept the agreement
< Back
Next >
Cancel

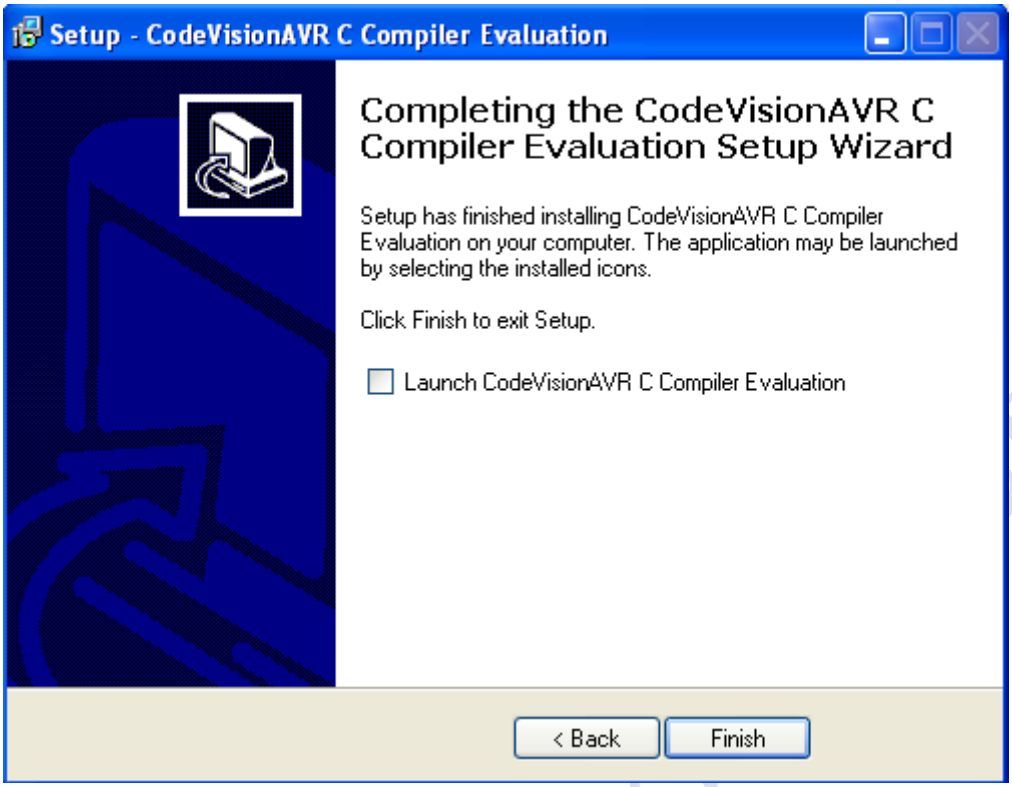

Setup - CodeVisionAVR C Compiler Evaluation
Select Destination Location
Where should CodeVisionAVR C Compiler Evaluation be installed?
Setup will install CodeVisionAVR C Compiler Evaluation into the following folder.
To continue, click Next. If you would like to select a different folder, click Browse.
c:\cvavreval
Browse...
تعیین مسیر نصب
At least 18.7 MB of free disk space is required.
< Back
Next >
Cancel


Setup - CodeVisionAVR C Compiler Evaluation
Select Start Menu Folder
Where should Setup place the program's shortcuts?
Setup will create the program's shortcuts in the following Start Menu folder.
To continue, click Next. If you would like to select a different folder, click Browse.
CodeVisionAVR
Browse...
انتخاب مکان فایل میانبر در منوی Start
< Back
Next >
Cancel


Setup - CodeVisionAVR C Compiler Evaluation
Ready to Install
Setup is now ready to begin installing CodeVisionAVR C Compiler Evaluation on your computer.
Click Install to continue with the installation, or click Back if you want to review or change any settings.
Destination location:
c:\cvavreval
Start Menu folder:
CodeVisionAVR
< Back
Install
Cancel


Setup - CodeVisionAVR C Compiler Evaluation
Installing
Please wait while Setup installs CodeVisionAVR C Compiler Evaluation on your computer.
Extracting files...
c:\cvavreval\examples\SDCARD\SDCARD1\sdcard1.c
Cancel


Setup - CodeVisionAVR C Compiler Evaluation
Information
Please read the following important information before continuing.
When you are ready to continue with Setup, click Next.
Welcome to the CodeVisionAVR C Compiler V2 and the
CodeWizardAVR Automatic Program Generator
***** IMPORTANT NOTE *****
The CodeVisionAVR Evaluation version has the following limitations:
- the compiled code size is restricted to 1kbyte (512words) for chips with less then 2kwords of FLASH and 3kbytes (1.5kword) of FLASH for the rest of chips
- the libraries for the ATxmega TWI Functions, PCF8583, DS1302, DS1307 Real Time Clocks, DS2430, DS2433 EEPROMs are not available
- interrupt driven serial communication code generation by the CodeWizardAVR is disabled
- EBI support for the ATxmega chips is disabled in the compiler
Next >



پس از نصب اگر نرم افزار اجرا شد آن را بسته و فایل cvavr.exe **را از پوشه** Crack **کپی کرده و در آدرس زیر** Paste **می کنیم:**

مسیر نسب\cvavreval\bin

سپس فایل ریخته شده را اجرا می کنیم.

فصل دوم

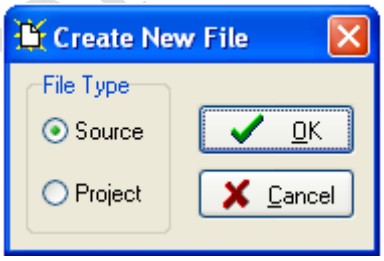
آموزش نرم افزار Codevision

در نرم افزار Codevision به دو طریق می توان برنامه دلخواه را ایجاد کرد.

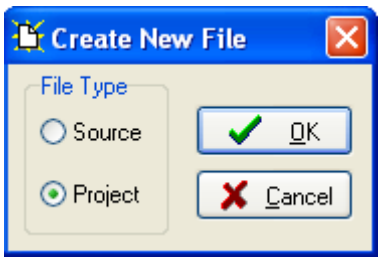
۱ -با ایجاد یک صفحه پروژه و نوشتن کل برنامه

۲ -با استفاده از Codewizard

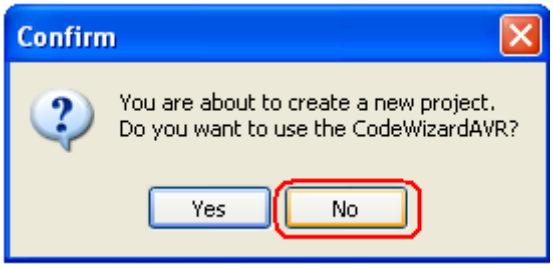
در روش اول از منو فایل گزینه New را کلیک می کنیم. در این پنجره گزینه Source را اتخاب می کنیم.



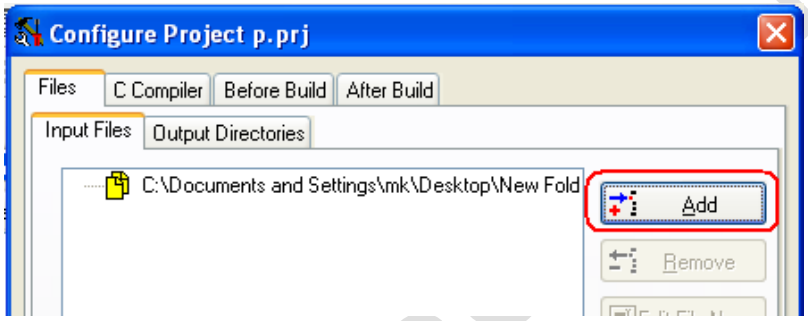
صفحه ایجاد شده را Save کرده و دوباره با کلیک بر روی گزینه New یک Project ایجاد می کنیم.



چون نمی خواهیم از Codewizard استفاده کنیم گزینه No را انتخاب می کنیم.

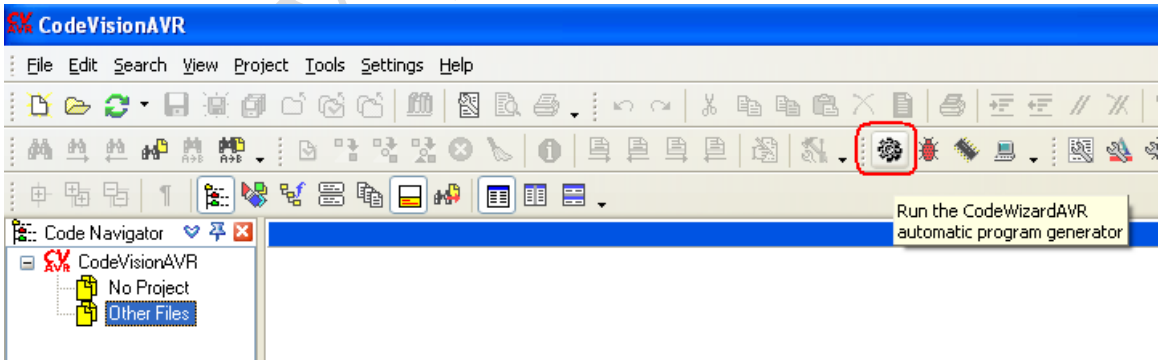


پس از Save این فایل در همان مسیر، در پنجره باز شده بر روی گزینه Add کلیک کرده و فایل Source ایجاد شده با پسوند C را به پروژه اضافه می کنیم.

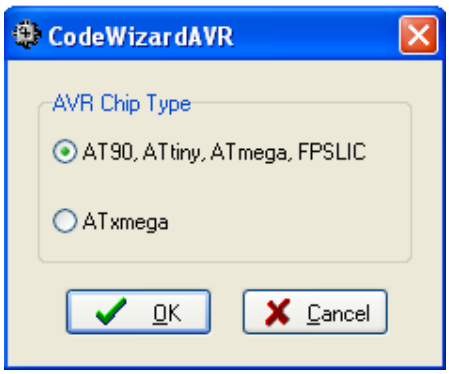


با استفاده از این پنجره آی سی مورد نظر، فرکانس کاری، امکانات جانبی و ... را تعیین می کنیم.

در روش دوم که بهترین روش می باشد، پس از باز کردن نرم افزار، آیکن CodeWizard را کلیک می کنیم



در پنجره باز شده چون قصد استفاده از آی سی Xmega را نداریم گزینه اول را انتخاب می کنیم.

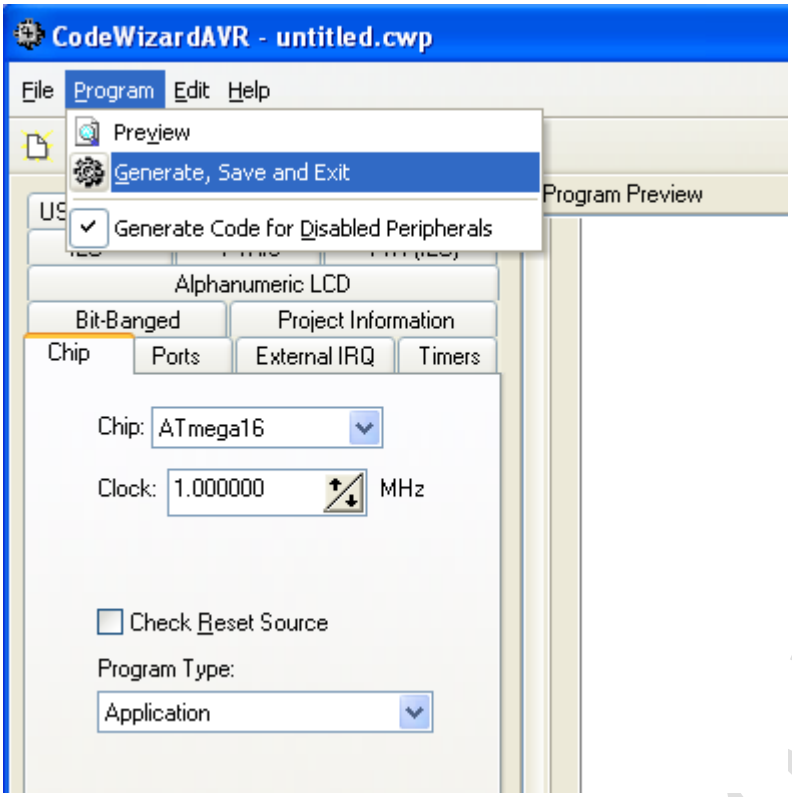


با انتخاب هر یک از آی سی های AVR امکانات آن به سربرگ ها افزوده می شود و برای فعال کردن هریک از امکانات از قبیل LCD کاراکتری، پورت سریال و ... کافی است به قسمت مربوطه برویم و تیک فعال سازی آن را اتخاب کنیم. (روش استفاده از هر قسمت این پنجره در مبحث مورد نظر آورده می شود)

همچنین یکی از روش های آشنایی با امکانات آی سی های AVR، استفاده از CodeWizard است

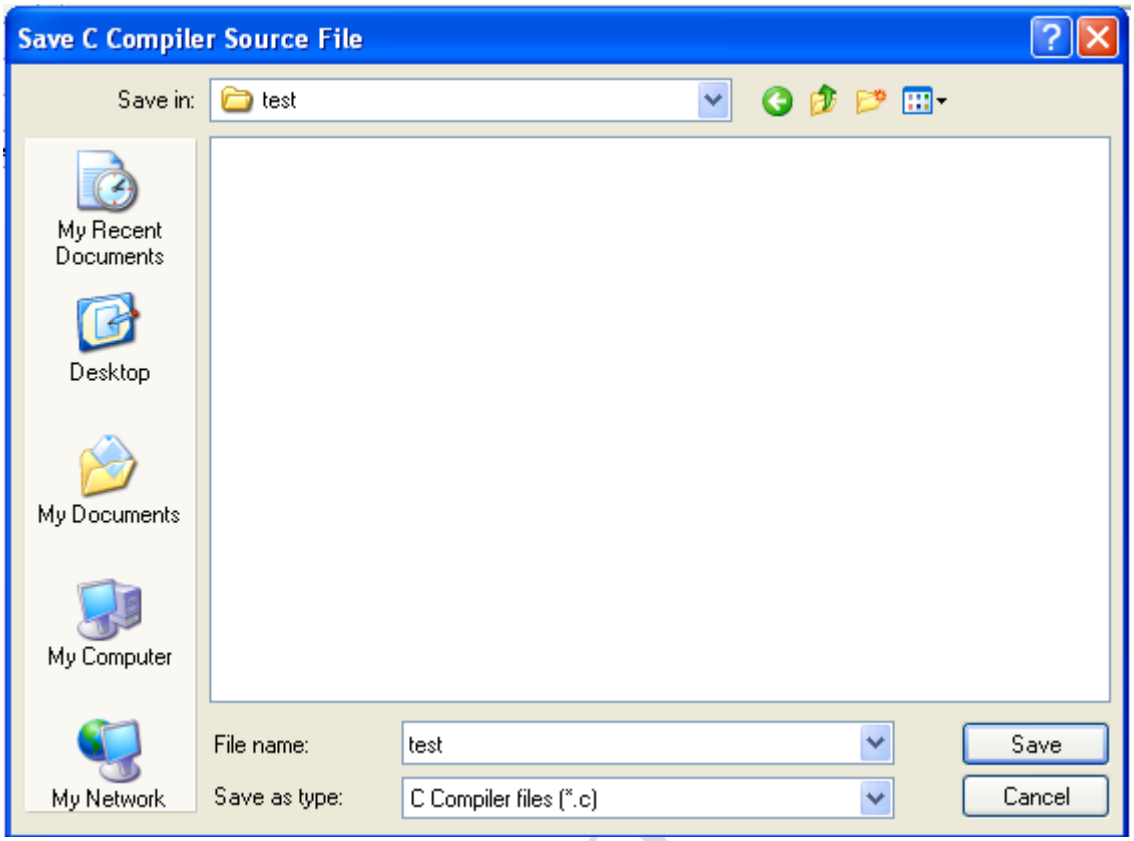
برای شروع ما آی سی mega16 که امکانات زیادی برای آموزش دارد را انتخاب می کنیم.

پس از انتخاب امکانات مورد نظر و انجام تنظیمات مربوطه برای ساخت فایل پروژه، از منو Program گزینه Generate, Save and Exit را کلیک می کنیم.



نرم افزار سه فایل برای پروژه، فایل C و فایل تنظیمات Codewizard ایجاد می کند و ما باید سه بار نام دلخواه را در آن وارد کنیم که بهتر است نام سه فایل مانند هم باشد.

لازم است کلیه فایل هایی که می خواهیم ایجاد کنیم در یک پوشه مخصوص باشد تا از پراکندگی و اشتباه درجابجایی پروژه جلوگیری شود.



پس از Save **کردن مشاهده می کنید که تمام رجیسترها، کتابخانه های مربوطه، وقفه ها، تابع اصلی و حتی حلقه** while(1) **نوشته شده و شما فقط باید بدنه اصلی برنامه را به آن اضافه کنید.**

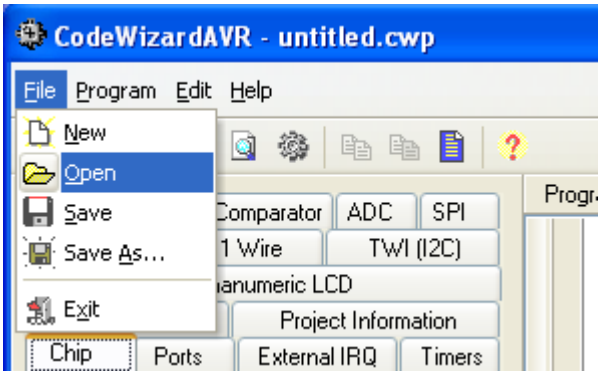
C:\Documents and Settings\mk\Desktop\test\test.c

Notes | test.c

```
16  Chip type               : ATmega16
17  Program type            : Application
18  AVR Core Clock frequency: 1.000000 MHz
19  Memory model            : Small
20  External RAM size       : 0
21  Data Stack size         : 256
22  *****************************************************/
23
24  #include <mega16.h>
25
26  // Declare your global variables here
27
28  void main(void)
29  {
30  // Declare your local variables here
31
32  // Input/Output Ports initialization
33  // Port A initialization
34  // Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
35  // State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
36  PORTA=0x00;
37  DDRA=0x00;
38
39  // Port B initialization
40  // Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
41  // State7=T State6=T State5=T State4=T State3=T State2=T State1=T State0=T
```

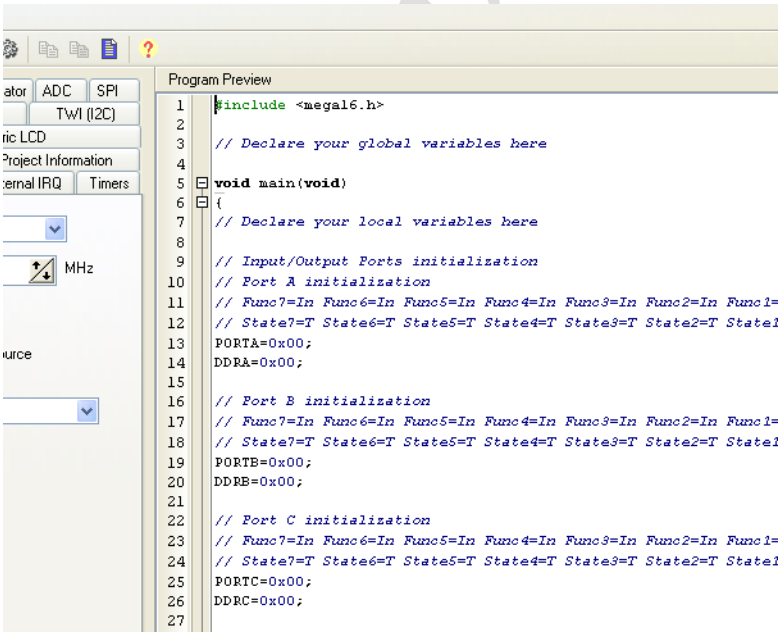
اگر بخواهیم در خلال برنامه تغییراتی در رجیسترها اعمال کنیم و برای این کار به Codewizard نیاز داشته باشیم به صورت زیر عمل می کنیم:

پنجره Codewizard را باز کرده و در آن از منو File گزینه Open را کلیک می کنیم.



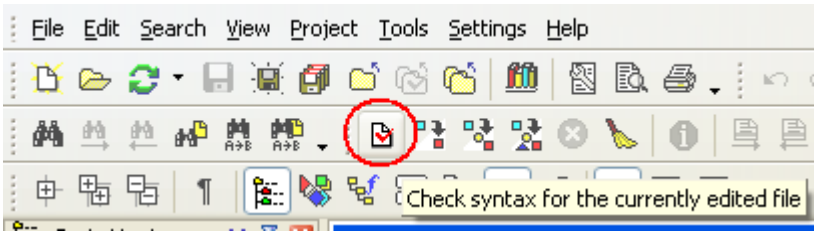
فایل Codewizard ایجاد شده برای برنامه با پسوند cwp را باز می کنیم. در این حالت مشاهده می کنید که تمام تنظیمات انجام شده Load می شود.

پس از تغییر تنظیمات از منو فایل گزینه Save را زده و سپس از منو Program گزینه Program preview را انتخاب می کنیم. برنامه جدید طبق تنظیمات انجام شده در سمت راست صفحه ظاهر می شود.

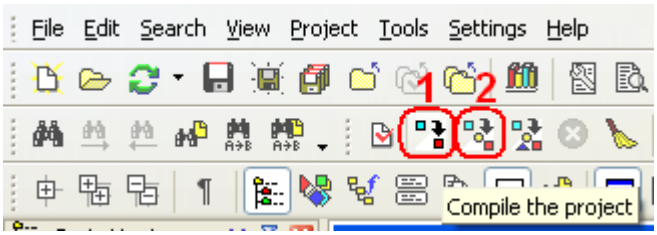


در اینجا رجیسترهایی را که تغییر کرده اند را کپی کرده و به جای مقادیر قبلی Paste می کنیم.

حین نوشتن برنامه اگر بخواهیم از وجود خطاها آگاه شویم گزینه Check syntax... را کلیک می کنیم. این گزینه مواردی را که مقایر با شکل انشایی دستورات است، به ما گزارش می دهد.



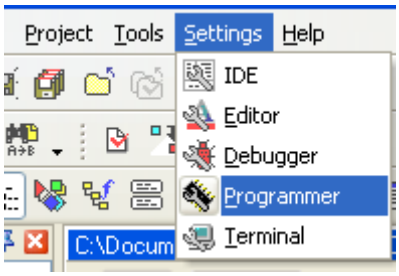
پس از نوشتن برنامه، گزینه Compile the project سپس Build the project را کلیک می کنیم.



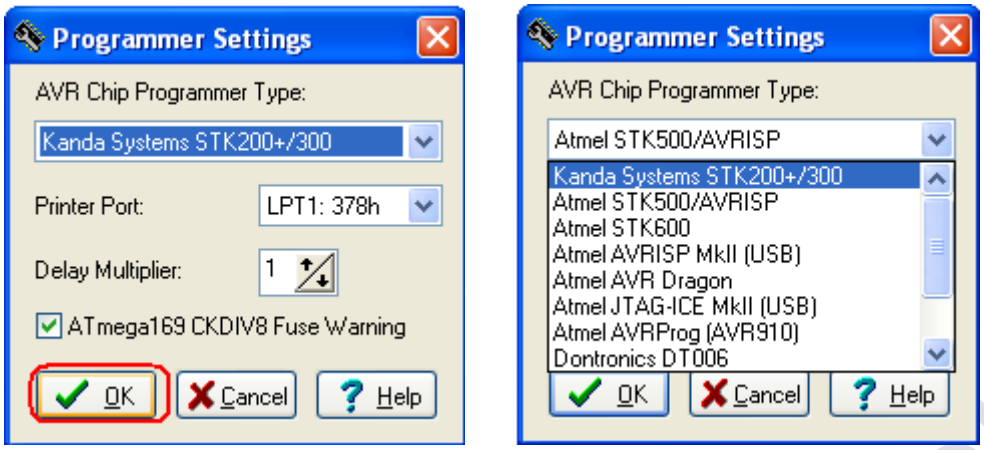
پروگرم کردن آی سی

اگر از پروگرمر متصل به پورت پارالل استفاده می کنید باید تنظیمات زیر را انجام دهید:

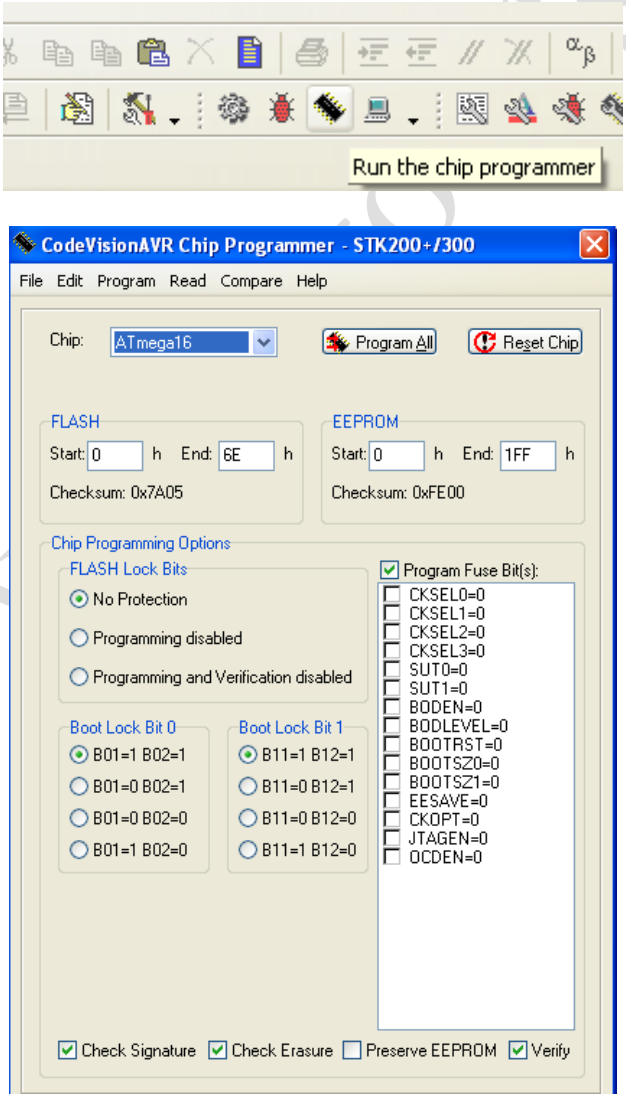
از منو Settings گزینه Programmer را انتخاب می کنیم.



در این پنجره از منو Programmer type گزینه STK 200+/300 را انتخاب کرده و OK می کنیم.



برای پروگرام کردن برنامه نوشته شده بر روی آی سی، گزینه Run the chip programmer **را کلیک می کنیم.**



ابتدا از گزینه Program **گزینه** Erase Chip **را می زنیم تا محتویات قبلی آی سی پاک شود.**

سپس از همین منو گزینه FLASH را زده تا برنامه داخل Rom ریخته شود. اگر از eeprom نیز استفاده کرده ایم باید گزینه EEPROM را نیز کلیک کنیم.

برای تغییر فیوزبیت ها و یا قفل ها، تغییرات لازم را در پنجره اصلی انجام داده و گزینه های Fuse Bits و یا Lock Bits را نیز کلیک می کنیم.

همچنین مراحل پروگرم کردن را نیز می توان به صورت اتوماتیک انجام داد. به این صورت که تمام تنظیمات مربوطه را انجام داده و گزینه Program All را کلیک می کنیم.

تذکر مهم: اگر در برنامه از eeprom استفاده نکرده اید و گزینه Program All را استفاده کرده اید پس از انجام Programming و Verify شدن، کامپایلر از شما می پرسد که eeprom خالی است، آیا می خواهید برنامه ی Load کنید. در این حالت شما باید گزینه Cancel را انتخاب کنید در غیر این صورت آی سی غیر قابل استفاده می شود.

فصل سوم

آموزش زبان C

تعاریف اولیه

کامپایلر: کامپایلر به معنای مفسر و مبدل می باشد و به نرم افزار هایی گفته می شود که زبان برنامه نویسی را به زبان ماشین تبدیل می کنند.

کلمه کلیدی: کلمه ی کلیدی در این مبحث به منظور لغتی خاص است که برای کامپایلر تعریف شده باشد.

CPU: یا پردازنده به سیستمی اطلاق می شود که قرار است برنامه را اجرا و پردازش کند.

کاراکتر: یک کاراکتر برابر یک حرف ، عدد و یا یک علامت می باشد. به دسته ای از کاراکتر ها یک رشته می گویند.

اولین قدم، الگوریتم

اولین قدم برای برنامه نویسی تشکیل الگوریتم کار می باشد. الگوریتم در واقع تعیین مراحل کار می باشد.

در زیر مثالی برای تبیین مفهوم الگوریتم آورده شده است:

مثال: الگوریتم تبدیل مبنای ۱۰ به مبنای غیر ۱۰:

۱) عدد اول را A و مبنای غیر ۱۰ را X بنام و یک نقطه در گوشه ای بنویس.

۲) A را بر X تقسیم صحیح کن و خارج قسمت ها را B و باقیمانده را Z بنام.

۳) باقیمانده را در سمت چپ نقطه ی بند ۱ بنویس.

۴) اگر B صفر نشد B را درون A بریز و برو به ۲.

۵) پایان.

روند برنامه نویسی با استفاده از الگوریتم

۱) اولین کار، الگوریتم و تشخیص متغیرها

۲) تست الگوریتم

۳) تبدیل الگوریتم به زبان مورد نظر

چهارچوب برنامه نویسی در C

برای جلوگیری از سردرگمی در نوشتن برنامه در ابتدا به شما شکل کلی یک برنامه گفته می شود و شما می توانید برنامه ی خود را با توجه به این چهارچوب بنویسید. با هر کدام از این بندها در ادامه آشنا می شوید.

۱ -معرفی کتابخانه ها (توابع کتابخانه ای)

۲ -متغیرها و لوازم اضافی مانند Sbit و ...

۳ -توابع جانبی فرعی

۴ -تابع اصلی

توضیحات اضافی: این توضیحات برای فهم راحت تر برنامه توسط برنامه نویس نوشته می شود و در هر جای برنامه نیز می توانند استفاده شوند. برای جلوگیری از اشتباه شدن

توضیحات با بدنه ی برنامه و برای اینکه در برنامه تأثیر نگذارند، قبل از هر خط توضیح دو علامت / قبل از آن می گذاریم همچنین اگر توضیحات از یک خط بیشتر شد علامت /* را در ابتدای توضیحات و علامت */ را در انتهای آن می گذاریم.

نکات مهم

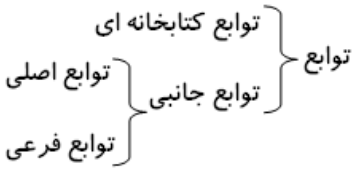
- ✓ پورت ها ، پین ها، رجیستر ها و بیت های مربوط به رجیسترها با حروف بزرگ نوشته می شوند.
- ✓ همه ی دستورات با حروف کوچک نوشته می شوند.
- ✓ پایان هر خط برنامه علامت ; (سمیکالان) قرار می گیرد.
- ✓ در برنامه نویسی به زبان C ، در ابتدای اعداد باینری 0b و در ابتدای اعداد هگز 0x قرار می دهیم.

```
a=0b0010/101;      (باینری)

b=0xc5;            (هگزا)

C=65;              (دهدهی)
```

- ✓ ابتدای زیر برنامه ها و دستورهای شرطی و حلقه ها با { (آکولاد باز) و انتهای آنها با } (آکولاد بسته) مشخص می شود.
- ✓ در جاهایی که فقط یک دستور داریم نیازی به آکولاد باز و بسته نیست.
- ✓ هرگاه signed یا unsigned را به کار نبریم کامپایلر آن را signed فرض می کند.
- ✓ متغیر float برای اعداد اعشاری مناسب است.
- ✓ نام برنامه ی اصلی main است.

تابع

هر برنامه C از چند تابع تشکیل شده است . دسته بندی توابع به شکل زیر است:



۱ -توابع کتابخانه ای: این توابع به صورت زیر تعریف می شوند:

```
# include < نام کتابخانه .h>
```

هر یک از این توابع یک سری از دستورات جانبی در مورد موضوعی خاص را در اختیار ما می گذارد به عنوان مثال اگر بخوهیم از عملگرهای ریاضی مثل جذر ، توان ، Sin و ... استفاده کنیم تابع math.h را فراخوانی می کنیم.

توابع کتابخانه ای همیشه در ابتدای برنامه نوشته می شوند.

۲- توابع جانبی: که به دو دسته اصلی و فرعی تقسیم می شوند:

توابع فرعی:

این توابع قبل یا بعد از تابع اصلی نوشته می شوند و به شکل زیر تعریف می شوند:

```
(نام و نوع متغیر ورودی) نام دلخواه    نوع برگشتی
{
        برنامه مورد نظر
}
```

فراخوانی توابع فرعی نیز به صورت زیر است:

```
(نام متغیرها یا مقدار آنها) نام تابع;
```

```
void delay (void)
{
   char i,j;
   for(i=0;i<100;i++)
      for(j=0;j<100;j++);
}
void main (void)
{
   char a=0;
   while(1)
   {
      a++;
      if(a>9)a=0;

      delay();
   }
}
```

تابع فرعی

نحوه ی استفاده از تابع فرعی

اگر تابع برگشتی نداشته باشد نوع آن Void (به معنای پوچ) تعریف می شود.

تابع اصلی: این تابع همیشه ثابت بوده و برنامه ی اصلی داخل آن نوشته می شود. تابع اصلی برگشتی ندارد پس هیشه به صورت زیر تعریف می شود:

```
Void  main (void)
{
        برنامه اصلی
}
```

متغیر

متغیر یک کاراکتر یا یک رشته از حروف و اعداد است که از آن به عنوان یک پیمانه استفاده می کنیم در مثال زیر به مفهوم متغیر بیشتر پی می برید:

فرض کنید می خواهید برنامه ای بنویسید که دو عدد را از کاربر گرفته ، آنها را با هم جمع و حاصل جمع را بر تعداد آنها تقسیم کند. هنگامی که شما این برنامه را می نویسید اطلاعی از مقدار این اعداد ندارید پس دو حرف (مثلاً a,b) را انتخاب کرده و اعمال ضرب و تقسیم را بر اساس این

متغیر ها انجام داده و جواب را به صورت پارامتری چاپ می کنید با این کار کاربر هر عددی را وارد کند ، برنامه آنها را مساوی a,b قرار داده و اعمال ضرب و جمع را بر اساس آنها انجام می دهد.

در مثال بالا برنامه ی توضیح داده شده اهمیتی ندارد و فقط هدف این است که شما با مفهوم متغیر آشنا شوید. در قسمت های بعد شیوه ی تعریف یک متغیر توضیح داده می شود.

گنجایش متغیر در کامپیوتر

متغیرها هر کدام ظرفیتی را اشغال می کنند. در برنامه نویسی باید ظرفیت معینی را به هر متغیر به میزان نیازش اختصاص داد یعنی اگر متغیر حافظه کمتری نیاز دارد نباید حافظه ی زیادی برای آن قرار دهیم. این امر در برنامه نویسی میکرو که حافظه ی کمی در اختیار داریم حساسیت بیشتری دارد.

انواع متغیرها

همانطور که گفتیم هر نوع متغیر ظرفیتی را اشغال می کند، در جدول زیر انواع متغیر در زبان C برحسب تعداد بایت اشغال کننده و دامنه آورده شده است :

نوع متغیر	میزان حافظه مورد استفاده (بیت)	محدوده
Bit	1	0 to 1
signed char	8	-128 to +127
unsigned char	8	0 to 255
Enum	16	-32768 to +32767
signed short	16	-32768 to +32767
unsigned short	16	0 to 65535
signed int	16	-32768 to +32767
unsigned int	16	0 to 65535
signed long	32	-2147483648 to 2147483647
unsigned long	32	0 to 4294967295
Float	32	±1.175494E-38 to ±3.402823E+38
Double	32	از نزدیک صفر تا 10^{308}
Sbit	1	0 to 1
Sfr	8	0 to 255
sfr16	16	0 to 65535

ثابت ها

ثابت ها، متغیرهایی هستند که می خواهیم در طول برنامه تغییر نکنند. برای تعریف یک ثابت به شیوه های زیر عمل می کنیم:

```
flash char a;          flash char a=5;
const char b;
```

بخاطر داشته باشید که دو کلمه ی کلیدی const **و** flash **فرقی با همدیگر ندارند.**

ثابتها به هیچ وجه قابل تغییر نیستند. به مثال های زیر توجه کنید:

```
flash char a;
char b;
a++;          (برنامه نمی تواند این خط را اجرا کند)
b=b*a;        (این دستور قابل قبول است و می تواند اجرا شود)
```

> متوجه شدید که ثابت ها هم مانند متغیرها می توانند مقدار دهی اولیه **نشوند**، اما اگر به آنها مقدار اولیه ندهیم ، کامپایلر مقدار اولیه آنها را به صورت پیش فرض **صفر** در نظر می گیرد. با توجه به اینکه ما از ثابتها استفاده کرده ایم باید بدانید که در طول برنامه نمی توان مقدار آنها را تغییر داد و مقدار آنها صفر باقی می ماند.

ثابت های شمارشی

با استفاده از کلمه ی کلیدی enum **می توان گروهی از ثابت ها را با مقدار اولیه به ترتب زیر تعریف کرد:**

```
enum(a,b,c);
```

در بالا مقدار a **برابر صفر، مقدار** b **برابر یک و مقدار** c **برابر ۲ خواهد بود.**

```
enum(a=2,b,c);
```

در این حالت a=2,b=3,c=4 **می باشد.**

عملگر ها

عملگر ها ابزار کمکی برای سهولت در برنامه نویسی هستند مانند: + ، – و ... . عملگر های که ما بیشتر از آنها استفاده می کنیم در جدول زیر مرتب شده اند.

عملگر حسابی:	**باقی مانده تقسیم (٪)**	**تقسیم(خارج قسمت) (/)**	**ضرب (*)**	**کاهش به اندازه یک واحد (--)**	
افزایش به اندازه یک واحد (++)	**منفی یگانی (–)**	**مثبت یگانی (+)**	**تفریق باینری (–)**	**جمع باینری (+)**	
عملگر بیتی :	**XOR بیتی (^)**	**OR بیتی (|)**	**AND بیتی (&)**	**مکمل ۱ (~)**	**NOT منطقی (!)**
عملگر منطقی :	**OR منطقی(یا) (||)**	**AND منطقی (وَ) (&&)**			
عملگر مقایسه :	**مخالف است با (!=)**	**برابر است با (==)**	**بزرگتر یا مساوی (>=)**	**بیشتر از (>)**	**کوچکتر یا مساوی (<=)** **کمتر از (<)**
عملگر شیفت :	**شیفت به راست (»)**	**شیفت به چپ («)**			
عملگر اشاره گر :	**محتوای آدرس (*)**	**آدرس (&)**			

عملگر ها را به صورت زیر نیز می توان به کار برد: a+=b; → a=a+b

نکته : نتیجه همیشه در متغیر سمت چپ ریخته می شود.

الویت ها در C

همانطور که ما برای محاسبات ریاضی الویت هایی قائل هستیم عملگرهای زبان C نیز دارای الویت برای محاسبه می باشند که به ترتیب در جدول زیر تنظیم شده است:

Level	Operators
1	( )
2	NOT, HIGH, LOW
3	+ (unary), - (unary)
4	*, /, MOD
5	+, -
6	SHR, SHL
7	AND, OR, XOR
8	>=, <=, =, <>, <, >, GTE, LTE, EQ, NE, LT, GT

متغیر های آرایه ای

وقتی به تعداد زیادی متغیر نیاز داریم، آنها را از نوع آرایه ای تعریف می کنیم، ولی دارای نام مشترک.

انواع متغیر های آرایه ای:

۱ -یک بعدی : ;[تعداد آرایه] نام دلخواه نوع متغیر

۲ -دو بعدی : ;[تعداد ستون][تعداد سطر] نام دلخواه نوع متغیر

آرایه ها را به صورت های مختلفی می توان استفاده کرد که در زیر به چند مورد آن اشاره می شود:

```
char a[10];  //a[0],...,a[9]//
a[0]=25;
a[1]=10;
.
.
.
a[9]=a[0]+a[1];
```

```
char a[2][3];
a[0][0]=15;
a[0][1]=...;
a[0][2]=...;
a[1][0]=...;
.
.
.
```

آرایه ی ماتریسی

```
char a[5]={0,6,5,-15};
char a[8]={'m','o','h','a','m','m','a','d',};
char a[2][3]={{5,8,9},{7,6,-4}};
```

برای نوشتن مجموعه کاراکتر ها با هم (مثلاً یک کلمه) از دابل کوتیشن (") استفاده می کنیم:

Char a[]=”mohammad”

اگر مقدار اولیه کلمه باشد نیازی به نوشتن تعداد کاراکترها نیست.

a[0]=’m’; a[1]=’o’; a[2]=’h’; a[3]=’a’; a[4]=’m’; a[5]=’m’; a[6]=’a’; a[7]=’d’;

کنترل برنامه در C

دستورات حلقه و پرش

این دستورات به دو دسته تقسیم می شوند:

۱) **شرطی: با بررسی یک شرط کار داخل حلقه را انجام می دهد.**

۲) **پرشی (غیر شرطی): بدون هیچ شرطی به جای خواسته شده می رود.**

دستورات شرطی

if – else

شرط if

با استفاده از این شرط، cpu شرطی را کنترل می کند و در صورت برقراری آن را اجرا می کند در غیر این صورت از آن گذشته و بقیه ی برنامه را انجام می دهد .

```
if (شرط)
{
        ...
}
```

شرط if-else

در این دستور، اگر شرط برقرار باشد اولین حلقه اجرا می شود در غیر این صورت حلقه ی دوم اجرا می شود.

```
if(شرط)
{
        ...
}
else
{
        ...
}
```


اگر چند شرط داشته باشیم می توانیم چند دستور if را پشت سر هم بنویسیم .

for

کاربرد این حلقه در مواقعی است که نیاز داشته باشیم به تعداد معینی یک کار انجام شود. در این حلقه، به تعداد تعیین شده حلقه تکرار می گردد و شیوه ی نوشتن آن به صورت زیر است:

```
for(مقدار اولیه;شرط(مقدار نهایی);عملکرد)
{
        ...
}
```

از forهای تو در تو می توان برای ایجاد تأخیر استفاده کرد:

```
for(i=0;i<100;i++)
        for(j=0;j<100;j++);
```


حلقه ی for(;;) یک حلقه ی بینهایت است یعنی cpu دیگر نمی تواند از آن خارج شود.

for(;;); در این خط گیر می افتد cpu

do-while

در این حلقه ابتدا یک بار cpu وارد حلقه می شود و در انتها شرطی را بررسی می کند.

```
do
{
        ...
} while(شرط)
```

در این حلقه cpu حداقل یکبار دستورات داخل حلقه را انجام می دهد.

while

این حلقه درست مانند حلقه ی do-while است ولی برعکس آن عمل می کند به این صورت که ابتدا شرطی را بررسی می کند و سپس در صورت برقراری شرط وارد حلقه می شود. تا زمانی که شرط صادق باشد حلقه تکرار می شود.

```
while(شرط)
{
        ...
}
```


مانند for(;;) حلقه ی while(1) نیز یک حلقه ی بی نهایت است.


به خاطر داشته باشید که در برنامه نویسی میکرو هیچ وقت نباید برنامه به پایان برسد. چون در این صورت مدار از کار می افتد.

مثال: برنامه ای بنویسید که از ۰ تا ۱۰۰ بشمارد.

```
Void main (void)
{
Unsigned char k = 0;
   While (k < 100)
   {
      K = k + 1;
   }
}
```

switch – case

در این شرط، یک متغیر را انتخاب می کنیم سپس آن را با متغیرهای مختلف مقایسه می کنیم.در صورتی که با هرکدام از متغیر ها برابر باشد کارهای جلوی آن اجرا می شود. همچنین انتهای برنامه ی هر case کلمه ی کلیدی break را می نویسیم تا بعد از اجرای برنامه های آن case ، cpu از حلقه ی switch بیرون بپرد.

switch(متغیر)

{

case(متغیر یا مقدار اول) : برنامه های حلقه اول ;

case(متغیر یا مقدار دوم) : برنامه های حلقه دوم ;

...

defult : برنامه ی پیش فرض

}

در انتهای حلقه ی switch می توان یک پیش فرض (default) قرار داد تا در صورتی که هیچکدام از case ها استفاده نشوند default انتخاب شود.

دستورات پرش غیر شرطی

break

cpu هر گاه به این دستور برسد بدون قید و شرط از حلقه بیرون می پرد.

continue

cpu با رسیدن به این دستور به اول حلقه پرش می کند و شرط را کنترل می نماید و در صورت برقراری حلقه را از اول تکرار می نماید.

goto

برای هر خط برنامه می توان یک برچسب(Lable) با هر نام دلخواه تعریف کرد و توسط این دستور به آن پرش کرد.

نام برچسب :

...

goto نام برچسب ;

تفاوت while و if

در حلقه ی while تا وقتی که شرط درست باشد، کامپایلر حلقه را دور می زند اما در if یک بار برنامه را انجام می دهد و تا زمانی که go to را ننویسیم برنامه را دور نمی زند.

نحوه ی استفاده از پورت ها و پین های میکرو

اگر بخواهیم از پورت های میکرو به عنوان خروجی استفاده کنیم، آن پورت را مساوی جواب خروجی قرار می دهیم:

```
a=b*5;
PORTC=a;     or →      PORTC=b*5;
```

به همین صورت اگر بخواهیم فقط بر روی یک پین میکرو مقداری قرار دهیم به صورت زیر عمل می کنیم:

```
PORTA.0=a;        PORTC.5=0;
```

برای پین ها و پورت های خروجی از کلمه PORT و برای پین ها و پورت های ورودی از کلمه PIN استفاده می کنیم:

```
PORTB=56;                    ali=PINA;
PORTC.0=1;                   b=PIND.3;
```

هر یک از پورت ها دارای یک رجیستر DDR می باشند که اگر بخواهیم از پین به عنوان خروجی استفاده کنیم باید بیت مربوطه در رجیستر DDR برابر یک و اگر بخواهیم به عنوان ورودی استفاده کنیم بیت مربوطه را برابر صفر قرار می دهیم.

در مثال زیر مقدار پورت A برابر صفر است ولی پین اول آن یعنی PORTA.0 خروجی و بقیه پین های آن ورودی تعریف شده است:

```
PORTA=0x00;
DDRA=0x01;
```

آدرس دهی به یک پورت

چون خروجی هر پورت میکرو، ۸ بیت دارد آنها را به دو دسته ی ۴ بیتی تقسیم کرده و در مبنای ۱۶ می نویسیم. نوشتن این کد ها در مبنای هگزا به صورت زیر است:

```
0X**
```

که در آن 0 عدد ثابت، X در مبنای هگز بودن آن و ستاره ها کد ما را مشخص می کند.

فصل چهارم

آشنایی با AVR

مزایای AVR نسبت به میکروکنترلر 8051

۱ -سرعت بالاتر

۲ -تجهیزات جانبی بیشتر

۳ -دارای انواع حافظه های جانبی کم مصرف

۴ -ایزوله نسبت به نویز

انواع میکروکنترلرهای خانواده AVR

خانواده AVR میکر.کنترلرهای خود را بر حسب امکانات به ۴ دسته تقسیم بندی کرده است:

۱ -Tiny AVR

۲ -90S AVR

۳ -Mega AVR

۴ -Xmega AVR

در این دسته بندی ها میکروکنترلر ATmega16 از خانواده mega AVR به علت قیمت مناسب و امکانات کافی برای آموزش انتخاب شده است.

برای شروع به کار با این آی سی لازم است با بعضی اصطلاحات و امکانات این خانواده آشنا شویم:

تایمر Watch dog

یک تایمر ۸ بیتی داخلی است که جهت جلوگیری از هنگ کردن میکرو استفاده می شود. با فعال کردن این تایمر نسبت به فرکانسی که برای آن انتخاب شده است میکروکنترلر ریست می شود. به همین خاطر در قسمت های مختلف برنامه این تایمر را صفر می کنیم. بنابراین اگر برنامه از روال عادی کار خود خارج شد، میکرو ریست می شود. توضیح کامل دستورات این تایمر در فصل های بعد آمده است.

Start up

مدت زمانی که بعد از هربار ریست طول می کشد تا میکرو دوباره فعالیت خود را شروع کند.

مدهای Sleep

به کمک این مدها، قسمت هایی از میکرو که به آنها نیاز نداریم از کار می افتد و بدین ترتیب در توان مصرفی صرفه جویی می شود. مانند:

مد idle (بیکاری): در این مد کلاک CPU قطع می شود اما سایر قسمتها به کار خود ادامه می دهند.

مد Power Down: در این مد اسیلاتور خارجی قطع شده و قسمتهایی که با این اسیلاتور کار می کنند از کار افتاده ولی سایر قسمتها به کار خود ادامه می دهند.

وقفه

گاهی نیاز است با تحریک یک پایه از بیرون یا انجام یک عمل در برنامه، کاری به صورت سریع انجام شود. با فعال شدن یک وقفه برنامه در هر جایی که باشد به تابع وقفه می رود و دستورات آن را اجرا می کند. یکی از توانایی های AVR وجود وقفه های فراوان در آن است.

EEPROM

یکی از حافظه های خواندنی و نوشتنی میکرو است که بر خلاف RAM با قطع برق اطلاعات آن پاک نمی شود.

برای استفاده از EPROM کافی است درابتدای تعریف متغیر کلمه کلیدی eeprom نوشته شود.

مثال:

```
eeprom  int  a=5;
```

فیوز بیت

فیوزبیت ها قسمتی از حافظه Flash Rom میکرو است که با برنامه ریزی آنها امکانات ویژه ای در اختیار ما قرار می گیرد. به عنوان مثال استفاده از اسیلاتورداخلی یا خارجی با فرکانس بالاتر، تعیین منابع ریست میکرو و ...

معرفی فیوز بیت ها

۱- INT CAP: **با انتخاب این فیوزبیت یک خازن داخلی بین پایه های کریستال با زمین قرار می گیرد و دیگر نیاز به نصب خازن خارجی نداریمو به صورت پیش فرض فعال نمی باشد.**

۲- CKSEL 3...0: **برای انتخاب نوع کلاک به کار می رود و به صورت پیش فرض برای همه ی میکروها RC داخلی انتخاب شده است.**

۳- SPIEN: **به صورت پیش فرض فعال می باشد و میکرو را می توان از طریق پروتکل SPI برنامه ریزی نمود.**

۴- FSTRT: **برای انتخاب سریع ترین زمان Start up بکار می رود. بصورت پیش فرض غیر فعال می باشد.**

۵- OCDEN: **اگر فعال شود به قسمت هایی از میکرو اجازه می دهددر مدهای Sleep کار کند و بصورت پیش فرض غیر فعال می باشد.**

۶- JTAGEN: **در صورت فعال بودن این بیت می توان میکرو را از طریق پروتکل JTAG برنامه ریزی کرد و به صورت پیش فرض فعال می باشد.**

۷- CKOPT: **با برنامه ریزی این فیوزبیت خازن 36pf برای RC خارجی بین پایه XTAL1 و زمین قرار می گیرد و دیگر نیازی به خازن خارجی نداریم.**

با انتخاب این فیوزبیت حداکثر فرکانس RC خارجی، 16MHz و در صورت عدم انتخاب حداکثر 8MHz می باشد و به صورت پیش فرض نیز غیر فعال است.

۸- EESAVE: با فعال کردن این فیوزبیت اطلاعات EEPROM ذخیره شده و با Erase کردن میکرو از بین نمی رود و بصورت پیش فرض نیز غیر فعال است.

۹- BOOTSZ0 و BOOTSZ1: به کمک این دو بیت مقدار حافظه Boot را مشخص می کنیم و بصورت پیش فرض هم غیر فعال است.

۱۰- BOOTRST: در صورت فعال بودن این بیت بعد از هربار Reset، CPU کار خود را از اولین خانه ی حافظه Boot شروع می کند. به صور پیش فرض غیر فعال است.

۱۱- BODLEVEL: در حالت عادی هرگاه ولتاژ تغذیه از 2.7v کمتر شود Reset اتفاق می افتد. اما در صورت برنامه ریزی این فیوزبیت هرگاه ولتاژ تغذیه از 4v کمتر شود Reset اتفاق می افتد و بصورت پیش فرض غیر فعال است.

۱۲- BODEN: برای اینکه فیوزبیت فوق عمل کند لازم است که این فیوزبیت برنامه ریزی شود.

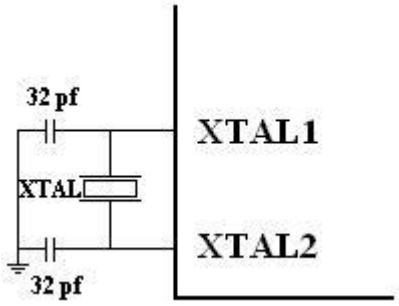
۱۳- SUT0 و SUT1: به کمک این دو فیوزبیت زمان Start up را تنظیم می کنیم و به صورت پیش فرض طولانی ترین زمان Start up انتخاب شده است.

فیوزبیت هایی که در بالا توضیح داده شد در اکثر میکروهای سری mega وجود دارند. برای آشنایی با کار سایر بیت ها می توانید به Datasheet آی سی مورد نظر مراجعه کنید.

منابع Clock در AVR:

برای راه اندازی CPU نیاز به یک منبع Clock می باشد تا نوسان مورد نیاز میکرو را با فرکانس دقیق و بالا تأمین کند. در ادامه چند نمونه از منابعی که AVR می تواند استفاده کند توضیح داده شده است.

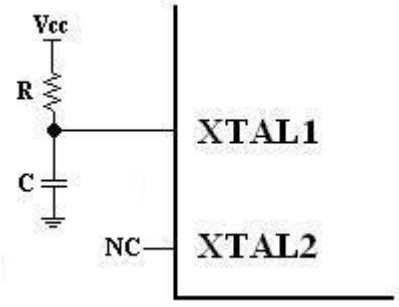
۱- کریستال خارجی: با اتصال یک کریستال دلخواه بین پایه های XTAL1 و XTAL2 می توان پالس مورد نیاز میکرو را تأمین کرد. باید توجه داشته باشید که فرکانس کریستال با فرکانس وارد شده در تنظیمات اولیه پروژه برابر باشد در غیر اینصورت محاسبات تایمرها اشتباه می شود. خازن های 32pf برای نویزگیری می باشد. (شکل ۴-۱)



شکل ۴-۱: اتصال کریستال به میکرو

۲- **کریستال خارجی فرکانس پایین:** کریستال 32.768 KHz که بین پایه های TOSC1 و TOSC2 قرار گرفته و از آن برای طراحی ساعت استفاده می کنیم.

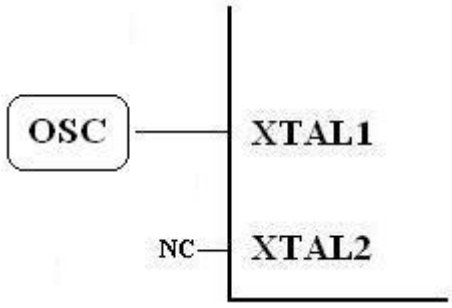
۳- **RC خارجی:** طبق شکل ۴-۲ می توان یک RC را به پایه XTAL1 متصل کرد. فرکانس کار این مدار از رابطه زیر بدست می آید.



شکل ۴-۲: اتصال RC خارجی به میکرو

۴- **RC داخلی:** تمام میکروکنترلرهای خانواده AVR دارای منبع کلاک RCداخلی می باشند که بصورت پیش فرض برای میکرو به عنوان منبع کلاک انتخاب شده است و بطور پیش فرض 1MHz می باشد که با تغییر فیوزبیت ها می توان آن را تا 8MHz ارتقاء داد.

۵- **اسیلاتور خارجی:** طبق شکل ۴-۳ می توان یک اسیلاتور خارجی را بطور مستقیم به میکرو متصل کرد.



شکل ۴-۳: اتصال اسیلاتور خارجی به میکرو

منابع Reset در AVR:

با انجام عمل reset تمام رجیسترها به حالت پیش فرض برگشته و کار برنامه از اول شروع می شود. AVR را می توان از خارج و هم به صورت داخلی reset کرد.

۱- **External Reset: هرگاه پایه Reset میکرو صفر شود، میکرو reset می شود.**

اگر بیشتر از چند ثانیه این پایه را صفر نگه داریم میکرو می سوزد.

۲- **هرگاه ولتاژ تغذیه از ولتاژ آستانه reset کمتر شود عمل reset اتفاق می افتد.**

۳- **هرگاه تایمر watchdog سرریز کند reset اتفاق می افتد.**

۴- **تا زمانی که در حلقه JTAG رجیستر reset یک منطقی باشد.**

فصل پنجم

اتصال 7seg به AVR

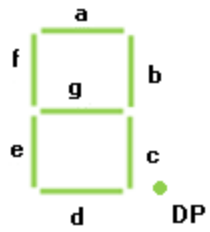
اتصال 7seg به میکرو به صورت مستقیم

فرض کنید برنامه ی شمارنده ای را نوشته اید و می خواهید آن را با میکرو پیاده سازی کنید. همانطور که می دانید خروجی میکرو در مبنای باینری است که برای نمایش دادن آن بر روی 7seg نیاز به دکودر BCD→7seg (آی سی ۷۴۴۷) می باشد. استفاده از دکودر هم حجم کار ما را زیادتر می کند و هم به دلیل۸ بیتی بودن خروجی میکرو و ۴ بیتی بودن این آی سی ها نتیجه ی مطلوب را به ما نمی دهد. برای این کار می توانیم با کمی تغییر در برنامه، آن را به گونه ای تنظیم کنیم تا بدون نیاز به مبدل اعداد روی 7seg نشان داده شوند. به مثال زیر دقت کنید:

مثال: برنامه ای بنویسید که اعداد صفر تا ۹ را شمرده و مستقیماً از میکرو به 7seg وصل شود.

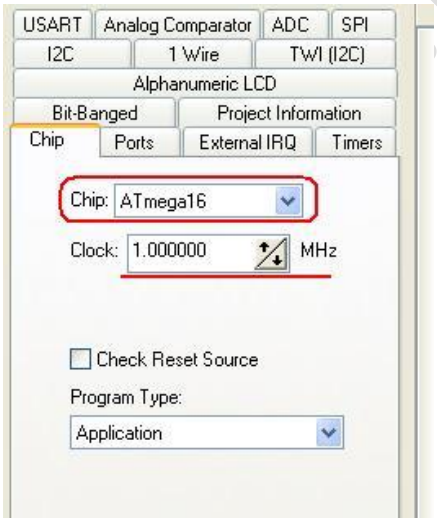
این برنامه را می توان با استفاده از آرایه ها به راحتی نوشت به این صورت که پین های پورت مورد نظر را به پایه های 7seg متصل کرده و برای هر عدد چراغ هایی را که لازم است روشن شوند، ۱ می کنیم و مجموعه اعداد بدست آمده برای هر عدد را به صورت کد هگزِ آن می نویسیم.

کد های داخل آرایه در برنامه زیر به صورتی بدست آمده اند که مستقیماً BCD را به 7seg تبدیل می کنند. طراحی این کدها در جدول زیر آمده است: (توجه شود که مدار به صورت High Active طراحی شده است، یعنی با ولتاژ سطح بالا (۱ منطقی) کار می کند.)



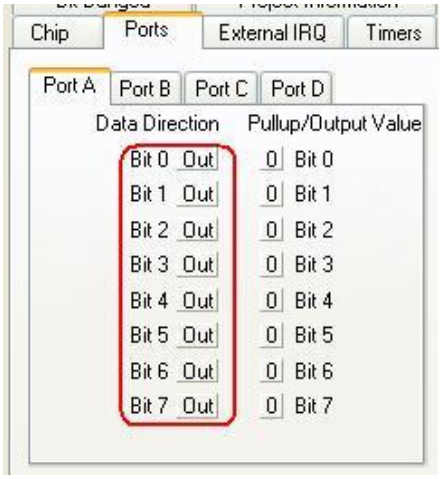
عدد دهدهی	کد	DP	G	f	e	d	C	b	A
0	0xC0	1	1	0	0	0	0	0	0
1	0xF9	1	1	1	1	1	0	0	1
2	0xA4	1	0	1	0	0	1	0	0
3	0xB0	1	0	1	1	0	0	0	0
4	0x99	1	0	0	1	1	0	0	1
5	0x92	1	0	0	1	0	0	1	0
6	0x82	1	0	0	0	0	0	1	0
7	0xD8	1	1	0	1	1	0	0	0
8	0x80	1	0	0	0	0	0	0	0
9	0x90	1	0	0	1	0	0	0	0

برای نوشتن این برنامه پس از باز کردن نرم افزار Codevision پنجره Codewizard را باز کرده و در آن نوع آی سی را mega16 و فرکانس کار را 1MHz انتخاب می کنیم.



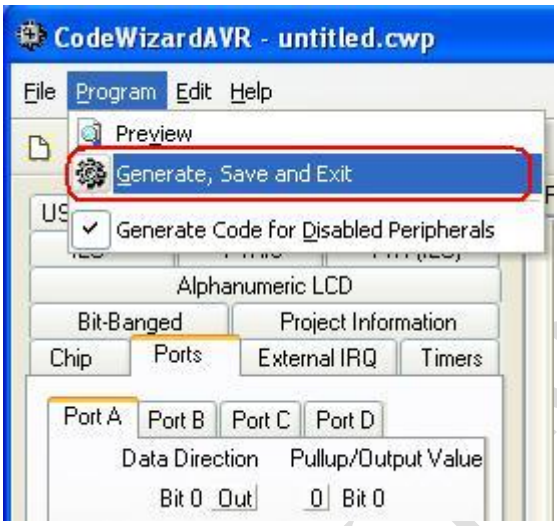
شکل ۵-۱

همچنین باید پین های یک پورت (مثلاً PORTA) را خروجی تعریف کنیم.



شکل ۵-۲

در آخر از منو Program گزینه Generate, Save and Exit را کلیک می کنیم.



شکل ۵-۳

همانطور که قبلا نیز گفتیم در یک پوشه مخصوص فایل ها را ترجیحاً با نام یکسان Save می کنیم.

مشاهده می کنید که تمام پین های رجیستر DDRA بصورت خروجی فعال شده اند:

```
void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTA=0x00;
DDRA=0xFF;
```

شکل ۵-۴

ابتدا آرایه ای را که در بالا توضیح دادیم تعریف می کنیم:

```
#include <mega16.h>

// Declare your global variables here
unsigned char dsp[10]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xd8,0x80,0x90};
void main(void)
{
```

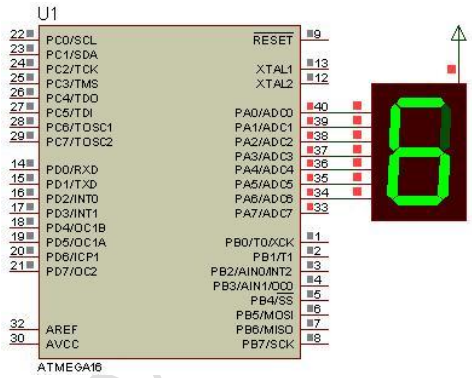
شکل ۵-۵

برنامه زیر را برای نمایش عدد دلخواه بر روی 7seg می نویسیم. به جای عدد داخل براکت هر عددی را که بنویسیم بر روی 7seg نمایش داده می شود.

```
while (1)
      {
      // Place your code here
        PORTA=dsp[6];
      }
}
```

شکل ۵-۶

(در اتصال پایه های 7seg باید دقت شود که پایه a کم ارزش ترین و پایه g پر ارزش ترین جایگاه را دارد بنابراین پایه a به PORTA.0 و پایه g به PORTA.6 متصل می شود.)



شکل ۵-۷

می توان برنامه فوق را به گونه ای نوشت که اعداد را پیاپی به فاصله یک ثانیه نمایش دهد. برای این کار کافی است تغییرات زیر را در برنامه اعمال کنیم.

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

// Declare your global variables here
unsigned char i;
unsigned char dsp[10]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xd8,0x80,0x90};
void main(void)
{
```

شکل ۵-۸

```
while (1)
      {
      // Place your code here
        for(i=0;i<10;i++)
        {
            PORTA=dsp[i];
            delay_ms(1000);
        }
      }
}
```

شکل ۵-۹

در این برنامه از متغیر i برای شمارش استفاده شده است.

آشنایی با کتابخانه delay.h

این کتابخانه برای ایجاد تأخیر زمانی استفاده می شود و دارای دو دستور زیر است:

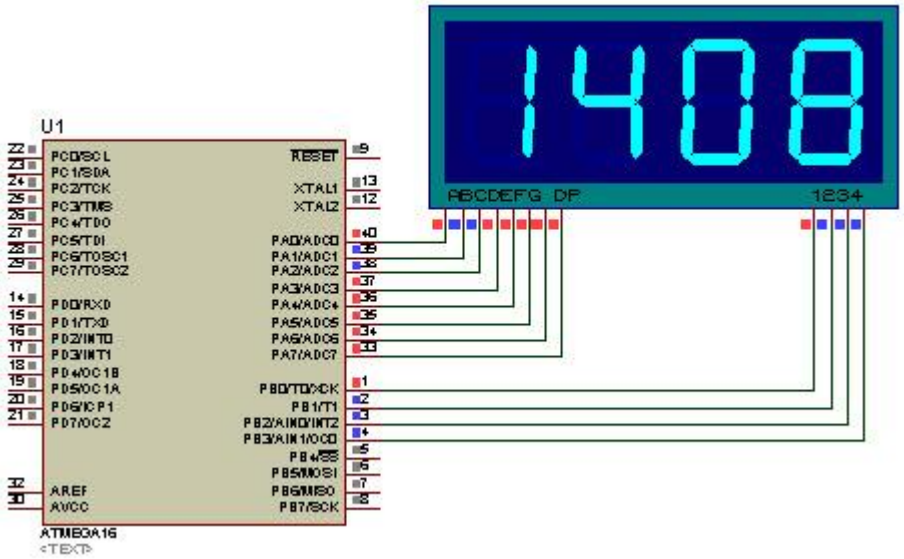
برای ایجاد تأخیر در حد میکروثانیه delay_us(عدد مورد نظر);

برای ایجاد تأخیر در حد میلی ثانیه delay_ms(عدد مورد نظر);

اتصال بیش از یک 7seg به یک پورت میکرو به صورت مستقیم

برنامه ی بالا فقط توانایی شمارش اعداد تک رقمی را دارد و برای شمارش اعداد ۲ رقمی به دو پورت نیاز است. در این مورد می توان از خطای دید انسان استفاده کرد به این صورت که اعداد را به صورت پشت سر هم با سرعت بالا به پورت بدهیم و با آدرس دهی از طریق یک پورت دیگر به پایه های تعیین کننده 7seg ، آن عدد را نمایش دهیم. سخت افزار گفته شده طبق شکل ۵-۱۰ می باشد.(البته در عمل برای اتصال پایه های مشترک 7seg به میکرو لازم است از ترانزیستور یا بافر استفاده شود تا آی سی میکرو آسیب نبیند.)

سخت افزار مورد نیاز به صورت شکل ۵-۱۰ بسته می شود:



شکل ۵-۱۰

برنامه زیر اعداد صفر تا ۹۹۹۹ را شمرده و بر روی 7seg نشان می دهد:

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

// Declare your global variables here
unsigned int k;
unsigned char a,b,c,d,i,dsp[10]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xd8,0x80,0x90};
void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTA=0x00;
DDRA=0xFF;

// 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=T State0=T
PORTB=0x08;
DDRB=0x0F;

while (1)
      {
      // Place your code here
        for(i=0;i<2;i++)
        {
            PORTB=0x08;
            PORTA=dsp[a];
            delay_ms(7);

            PORTB=0x04;
            PORTA=dsp[b];
            delay_ms(7);

            PORTB=0x02;
            PORTA=dsp[c];
            delay_ms(7);

            PORTB=0x01;
            PORTA=dsp[d];
            delay_ms(7);
        }
        a=k%10;
        b=(k/10)%10;
        c=(k/100)%10;
        d=(k/1000)%10;
        k++;
        if(k>9999)
            k=0;
      }
}
```

شکل ۵-۱۱

در برنامه فوق پایه مشرک مورد نظر که به میکرو متصل است یک شده و دیتای مرتبط با آن سگمنت بر روی پورت A ریخته می شود. برای اینکه نور حاصل از عدد نمایش داده شده دیده شود، باید زمان کوتاهی یک سگمنت روشن مانده سپس آن را خاموش و سگمنت بعدی را روشن

کنیم. این کار با استفاده از دستور delay انجام می گیرد. حلقه for برای تنظیم سرعت نمایش و متغیر k صرفاً برای شمارش اعداد استفاده شده است و می توان برای نمایش عدد دلخواه تنها متغیر k را برابر یک عدد خاص قرار داد.

چون در هر لحظه ما فقط یک عدد یک رقمی را بر روی پورت ارسال می کنیم، باید ارقام عدد چهار رقمی را به رقم های جداگانه تبدیل کنیم. برای اینکار از دستورات / و ٪ استفاده شده و یکان، دهگان، صدگان و هزارگان به ترتیب در متغیرهای a، b، c و d قرار می گیرند. همانطور که در فصل سوم گفتیم علامت / خارج قسمت یک تقسیم و علامت % باقیمانده را بر می گرداند. بنابراین اگر یک عدد چهار رقمی را بر ۱۰ تقسیم کنیم، باقیمانده آن برابر یکان عدد اصلی است. همچنین اگر عدد را بر ۱۰ تقسیم کنیم یکان حذف شده و با گرفتن باقیمانده عدد حاصل، دهگان عدد اصلی بدست می آید. به همین ترتیب می توان تا n رقم پیش رفت.

در برنامه فوق می توان با یک تکنیک ساده از تکرار جلوگیری کرد. برنامه اصلاح شده در شکل ۵-۱۲ آمده است:

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

// Declare your global variables here
unsigned int k;
unsigned char a[4],i,j,dsp[10]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xd8,0x80,0x90};
void main(void)
{
// Declare your local variables here

// Input/Output Ports initialization
// Port A initialization
// Func7=Out Func6=Out Func5=Out Func4=Out Func3=Out Func2=Out Func1=Out Func0=Out
// State7=0 State6=0 State5=0 State4=0 State3=0 State2=0 State1=0 State0=0
PORTA=0x00;
DDRA=0xFF;

// 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=T State0=T
PORTB=0x08;
DDRB=0x0F;

j=0x08;
while (1)
      {
      // Place your code here
        for(i=0;i<4;i++)
        {
            PORTB=j;
            PORTA=dsp[a[i]];
            delay_ms(7);
            j=j>>1;
            if(j==0)
                j=0x08;
        }
        a[0]=k%10;
        a[1]=(k/10)%10;
        a[2]=(k/100)%10;
        a[3]=(k/1000)%10;
        k++;
        if(k>9999)
            k=0;
      }
}
```

شکل ۵-۱۲

نوشتن یک برنامه با استفاده از دستور delay به هیچ وجه مطلوب ما نیست و اینگونه برنامه ها با استفاده از تایمرها نوشته می شود. روش استفاده از تایمرها AVR در فصل های بعدی آمده است.

باید توجه داشت که با این روش وصل کردن بیشتر از ۶ 7seg به یک پورت نتیجه ی مطلوبی نمی دهد.

فصل ششم

اتصال LCD کاراکتری به mega16

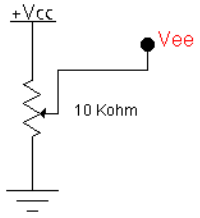
برای اتصال LCD به میکرو ابتدا باید با پایه ها و شیوه ی عملکرد آن آشنا شویم. در این مبحث با LCD ۲×۱۶ کار می کنیم بقیه ی نمایشگرهای کاراکتری نیز مشابه این نمایشگر می باشند.

در جدول زیر شماره پایه، نام پایه و عملکرد آن آمده است:

شماره پایه	نام پایه	عملکرد
1	Vss	زمین ، GND
2	Vcc	تغذیه مثبت ، 5v
3	Vee	تنظیم نور کاراکترها (کنتراست)
4	RS	اگر RS=0 باشد مقدار ورودی به عنوان یک دستور می باشد اما اگر RS=1 باشد مقدار ورودی یک داده برای چاپ شدن است
5	R/$\overline{W}$	اگر بخواهیم در LCD بنویسیم این پایه باید صفر باشد و اگر بخواهیم از LCD مقداری را بخوانیم باید آن را یک کنیم
6	E	پس از انجام هر عملیات ارسال یا دریافت باید پایه ی E را یکبار صفر و یکبار یک کنیم تا اطلاعات ثبت شوند
7 – 14	$DB_0 \rightarrow DB_7$	مسیر ورود و خروج اطلاعات LCD
15	Anod	تغذیه ی مثبت چراغ LCD
16	Katod	تغذیه ی منفی چراغ LCD

پایه ی Vee می تواند به صورت زیر برای کم و زیاد کردن نور نوشته ها به کار رود:

(اگر سر فلش به طرف منفی باشد، بیشترین نور، و اگر به طرف مثبت باشد، کمترین نور را خواهیم داشت.)



شیوه کار با LCD به این صورت است که ابتدا پایهR/W را یک کرده و RS را صفر می کنیم سپس دستورات فرمت، نحوه نوشتن و ... را به LCD ارسال می کنیم. همچنین بعد از ارسال هر دستور پایه En را یکبار صفر و یک می کنیم. بعد از آن برای ارسال دیتا پایه RS را یک کرده و دیتا را بایت به بایت به LCD ارسال می کنیم.

نرم افزار Codevision دارای یک کتابخانه کامل به نام alcd.h (در نسخه های قدیمی تر lcd.h) می باشد که تمام اعمال لازم را انجام می دهد و کافی است ما از دستورات آن استفاده کنیم. دستورات این کتابخانه به شرح زیر می باشد:

1- lcd_init (تعداد ستون);

این تابع تنظیمات اولیه lcd را انجام می دهد.

2- lcd_clear ();

پاک کردن تمام lcd. با این دستور مختصات lcd نیز به نقطه اول بر می گردد.

3- lcd_gotoxy (x,y);

برای رفتن به خانه مشخص از lcd استفاده می شود.

آدرس اولین خانه lcd (0,0) است.

4- lcd_putchar ();

برای نمایش یک کارکتر استفاده می شود.

مثال: x='B' lcd_putchar (x);

5- lcd_puts (**رشته ی مورد نظر**);

برای نمایش یک رشته (ذخیره شده در Ram)

6- lcd_putsf (**رشته ی مورد نظر**);

برای نمایش یک رشته (ذخیره شده در Rom)

برای اینکه تفاوت دو دستور بالا را متوجه شوید به مثال زیر دقت کنید:

x="mohammad" lcd_putsf ("mohammad");

lcd_puts (x);

کاراکترها را به صورت کد ASCII **دریافت می کند بنابراین اگر بخواهیم عددی بر روی** lcd **نمایش دهیم حتما باید در قالب رشته ارسال شود. برای اینکار یعنی تبدیل اعداد و حروف به کد اسکی لازم است با دو دستور زیر آشنا شوید:**

1-ftoa (**متغیر رشته ای مقصد، تعداد ارقام بعد از اعشار، متغیر عدد مورد نظر**);

تابع فوق در کتابخانه stdlib.h **قرار دارد.**

2-sprintf (**متغیر رشته ای مقصد**,"**کاراکترهای دلخواه و کاراکترهای فرمت**" ,**متغیرهایی که بجای کاراکتر فرمت قرار میگیرند**);

این تابع در کتابخانه stdio.h **قرار دارد. کاراکترهای فرمت در ادامه آورده شده است:**

یک کاراکتر (%C)	**اعداد صحیح دهدهی مثبت و منفی** (%d)	**اعداد صحیح دهدهی مثبت و منفی** (%i)
نمایش علمی عدد همراه با حرف e (%e)	**نمایش علمی عدد همراه با حرف** E (%E)	**عدد اعشاری ممیز شناور** (%f)
اعداد اعشاری ممیز شناور	**اعداد اعشاری ممیز شناور**	**اعداد مبنای ۸ مثبت**

(%g)	(%G)	(%o)
رشته ای از کاراکتر ها (عبارت رشته ای) (%s)	اعداد صحیح بدون علامت (مثبت) (%u)	اعداد مبنای ۱۶ مثبت با حروف کوچک (%x)
اعداد مبنای ۱۶ مثبت با حروف بزرگ (%X)	Pointer (اشاره گر) (%P)	علامت ٪ (%%)
(%n) موجب می شود تا تعداد کاراکترهایی که تا قبل از این کاراکتر به خروجی منتقل شده اند شمارش شده در پارامتر متناظر با آن قرار می گیرند		

به عنوان مثال عدد ذخیره شده در متغیر a را با استفاده از دو دستور بالا به رشته تبدیل می کنیم تا با قابلیت های دستورها آشنا شوید.

۱ -با استفاده از دستور ftoa

ابتدا باید یک متغیر آرایه ای تعریف کنیم تا رشته حاصل را در آن قرار دهیم:

```
unsigned char dsp[5];
```

سپس در هر قسمت از برنامه که نیاز بود از دستور ftoa استفاده کرده و رشته حاصل را با دستور lcd_puts بر روی lcd نمایش می دهیم.

```
ftoa(a,1,dsp);
lcd_puts(dsp);
```

برای مثال اگر عدد 635.254 در متغیر a قرار داشته باشد با در نظر گرفتن اینکه مقدار رشته ۵ تایی تعریف شده است، مقدار حاصل برابر 635.2 می شود. اگر نیاز به نمایش اعشار نداشته باشیم می توانیم بجای عدد ۱ صفر قرار دهیم.

ممیز عدد نیز به عنوان یک کاراکتر در نظر گرفته می شود که در مثال فوق ۴ رقم به همراه نقطه ممیز یک رشته ۵ تایی را تشکیل می دهد.

۲ -با استفاده از دستور sprint

یکی از مزایای این دستور تبدیل همزمان و ادغام چند رشته با یکدیگر است. فرض می کنیم متغیرهای sec=17 و min=5 و hour=20 اعداد یک ساعت می باشند و می خواهیم آنها را برای نمایش بر روی lcd آماده کنیم.

```
unsigned char dsp[16];
sprintf(dsp,"Time is=%d:%d:%d",hour,min,sec);
lcd_puts(dsp);
```

نتیجه نمایش داده شده بر روی lcd به صورت زیر است:

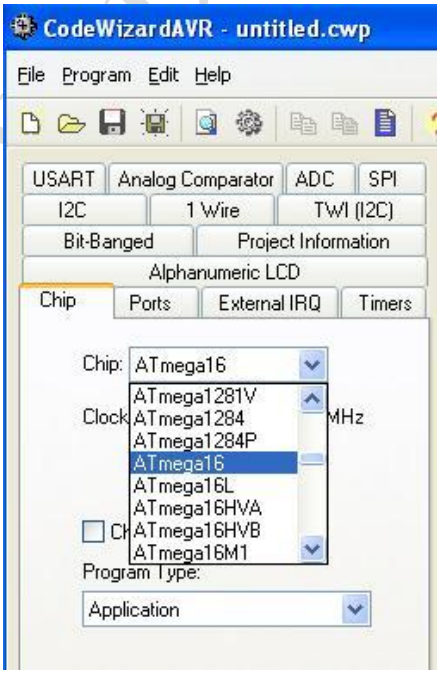
```
Time is=20:5:17
```

مثال: دستور sprint را به شکل زیر نوشته و پس از نمایش بر روی lcd نتیجه را با حالت قبل مقایسه کنید.

```
sprintf(dsp,"Time is=%2d:%2d:%2d",hour,min,sec);
```

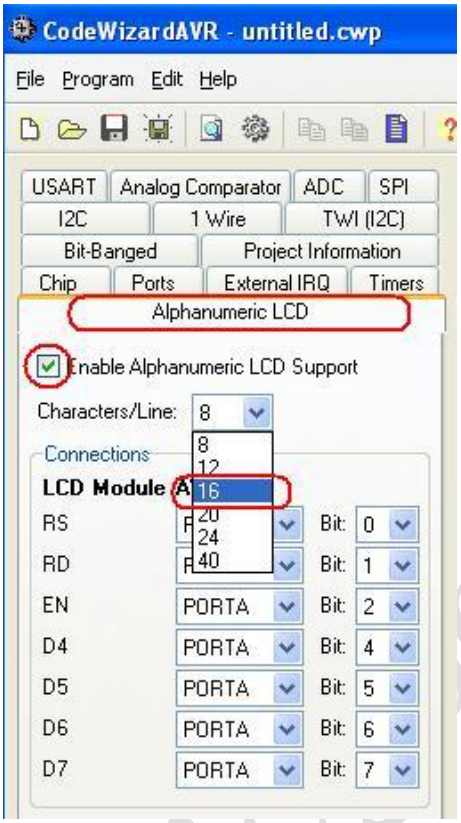
آشنایی با قسمت lcd کاراکتری در Codewizard:

پس از باز کردن codewizard آی سی mega16 را انتخاب می کنیم:



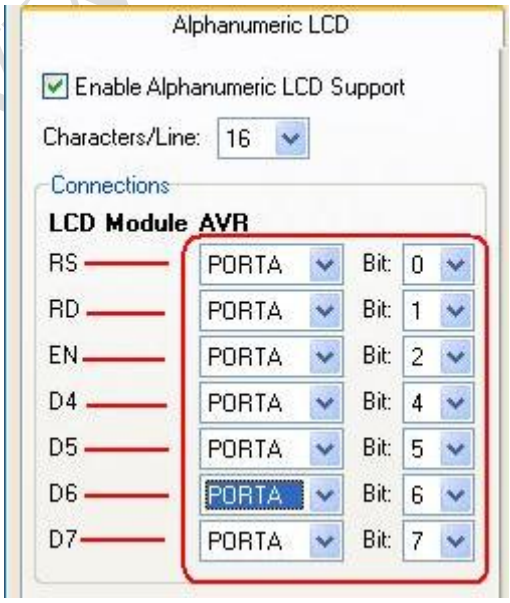
شکل ۶-۱

در سربرگ Alphanumeric LCD گزینه Enable را کلیک کرده و تعداد ستون LCD را تعیین می کنیم (مثلا برای lcd ۲ در ۱۶ عدد ۱۶ را انتخاب می کنیم)



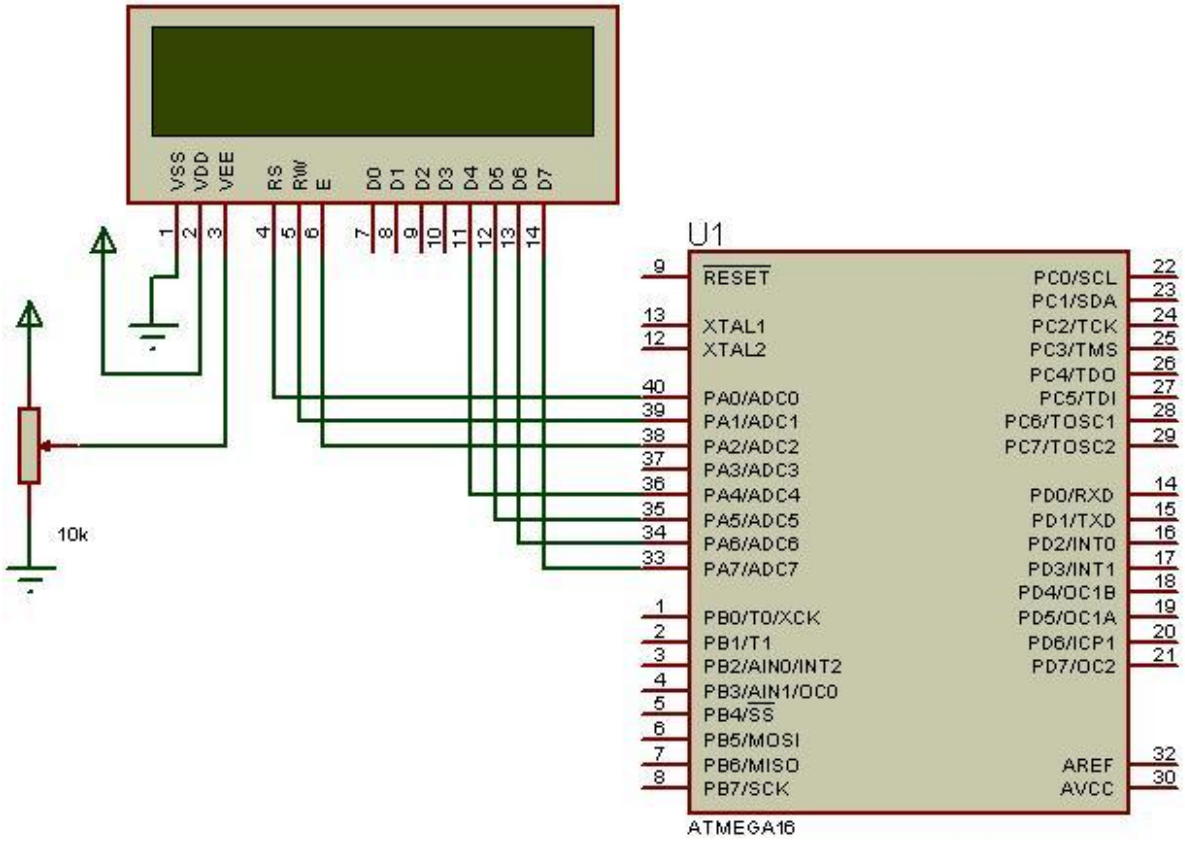
شکل ۶-۲

به صورت پیش فرض lcd به PORTA متصل و بیت های آن به شکل زیر تعیین شده اند که در صورت نیاز می توانیم هر کدام از پایه های آن را به پورتی دیگر وصل کنیم.



شکل ۶-۳

اتصال lcd به میکرو در سخت افزار



شکل ۶-۴

تمرین ۱: برنامه ساعت را نوشته و آن را بر روی lcd کاراکتری نمایش دهید.

فصل هفتم

تایمر و کانتر

تفاوت تایمر و کانتر:

تایمر یک شمارنده است که با تعیین فرکانس یا سرعت آن، به صورت شروع به شمارش می کند. (به عنوان مثال می تواند برای ساعت یک ساعت استفاده شود)

اما در کانتر سرعت شمارش با استفاده از پایه های خارجی کنترل شده و تعداد شمارش را ما تعیین می کنیم. (مثلاً برای شمارش تعداد بسته ها در یک خط تولید)

میکروکنترلر mega16 **یه تایمر/کانتر دارد:**

۱ -تایمر/کانتر صفر (۸ بیتی)

۲ -تایمر/کانتر ۱(۱۶ بیتی)

۳ -تایمر/کانتر ۲ (۸ بیتی)

این تایمر/کانترها هم به صورت تایمر و هم بصورت کانتر قابلیت کار دارند.

تایمر/کانترها رجیسترهای مختلفی دارند که در اینجا با ۳ رجیستر مهم و کاربردی آشنا می شوید:

1- TCNTx

مقدار لحظه ای تایمر یا کانتر در این رجیستر نگهداری می شود. این رجیسترها ۸ بیتی بوده و مقدار آنها از صفر تا ۲۵۵ تغییر می کند. (کاراکتر x **نشان دهنده شماره تایمر است. به عنوان مثال** TCNT0 **یا** TCNT2**)**

2-TCCRx

رجیستر تنظیمات تایمر/کانتر. مقدار این رجیستر طبق تنظیمات codewizard تعیین می شود. برای آشنایی با بیت های این رجیستر ۸ بیتی می توانید به دیتاشیت mega16 مراجعه کنید. برای خاموش کردن تایمر کافی است بیت های clock آن را صفر کرده یا مقدار TCCR را برابر صفر قرار دهیم.

3-OCRx

مقدار این رجیستر که توسط برنامه نویس تعیین می شود با مقدار TCNT مقایسه شده و طبق تنظیمات وقفه تطابق فعال یا پایه خروجی تغییر می کند.

پایه های ورودی کانترها T0 و T1 و T2 بوده و خروجی تایمرها نیز پایه های OC0 و OC1A و OC1B (به علت ۱۶ بیتی بودن) و OC2 می باشد.

مدهای کاری تایمرها

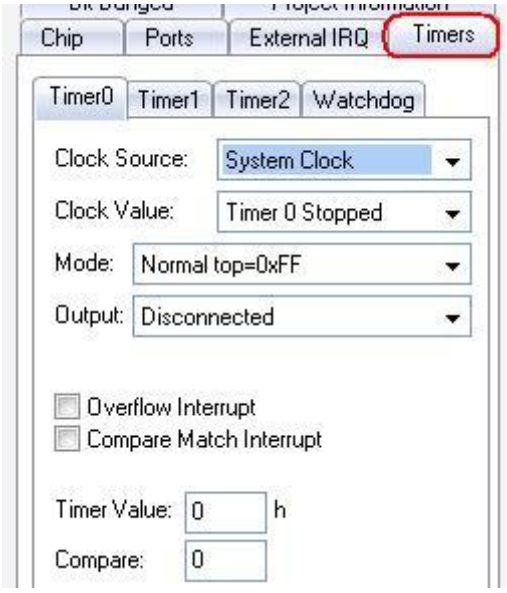
در ادامه ۵ مد کاری برای این تایمرها آورده شده است. البته در بعضی از تایمرها بیشتر ازدو یا سه مد کاری نداریم.

1- Normal mode

در این مد تایمر شروع به شمارش نموده و پس از سرریز شدن مقدار آن صفر شده و دوباره شمارش می کند. در این حالت اگر وقفه سرریز فعال باشد با سرریز شدن، تابع وقفه اجرا می شود.

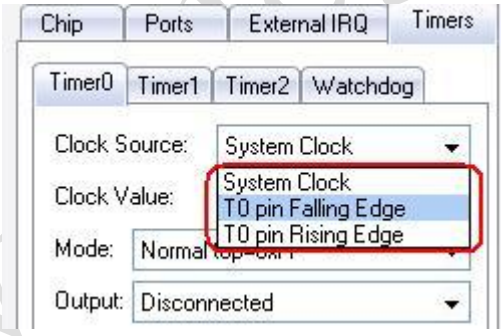
تمرین ۲: برنامه ای بنویسید که بر روی پایه OC0 (پایه خروجی تایمر/کانتر صفر) یک قطار پالس با فرکانس 1Hz ایجاد شود.

آشنایی با قسمت تایمر در Codewizard



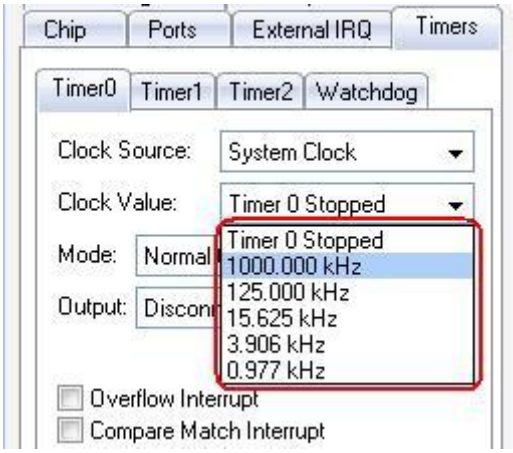
شکل ۷-۱

ابتدا برای تعیین تایمر بودن یا کانتر بودن از قسمت Clock Source گزینه System Clock را برای حالت تایمر و گزینه Tx Pin را برای حالت کانتر انتخاب می کنیم. در حالت کانتر می توان تحریک پایه خارجی کانتر را بالا رونده یا پایین رونده تعیین کرد.



شکل ۷-۲

با فرض اینکه در حالت تایمر قرار داریم از قسمت Clock Value مقدار فرکانس کاری تایمر را انتخاب می کنیم. همانطور که گفتیم تایمر صفر و دو ۸ بیتی هستند یعنی ماکزیمم تعدادی که این تایمرها شمارش می کنند ۲۵۶ عدد است. فرکانس تایمر نیز بیانگر سرعت شمارش یک رقم از ۲۵۶ رقم است. پس اگر ما فرکانس تایمر را 1 MHz انتخاب کنیم زمان آن طبق رابطه T=1/F برابر 1us بوده و زمان شمارش کل تایمر 256 us می شود.



شکل ۷-۳

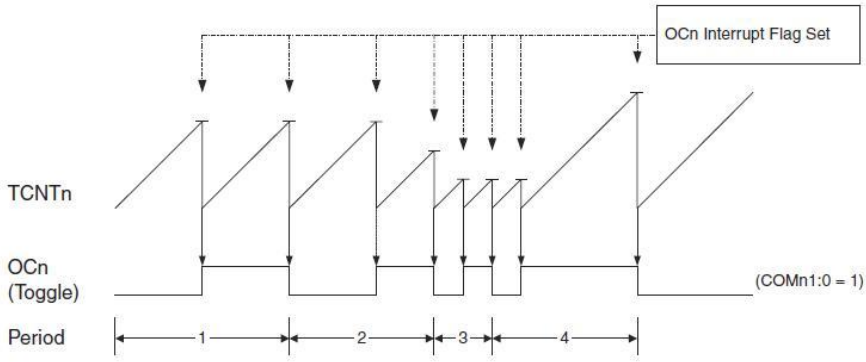
فرکانس هایی که در codewizard انتخاب می کنید تقسیمی از کلاک اصلی است که در سربرگ Chip انتخاب شده است پس اگر فرکانس اصلی را تغییر دهیم این فرکانس ها نیز متناسب با آن تغییر می کند.

در قسمت های بعد نیز مد تایمر را انتخاب و با توجه به مد انتخاب شده می توانیم وضعیت پایه خروجی را را در قسمت Output مشخص کنیم. با توجه به امکانات تایمر دو گزینه وقفه سر ریز و وقفه تطابق قرار دارد که در صورت نیاز می توانیم آنها را فعال کنیم.

2- CTC mode

در این مد هرگاه مقدار رجیستر OCRx با رجیستر TCNTx برابر شد مقدار تایمر صفر شده و طبق تنظیمات پایه خروجی یک شده، صفر شده یا معکوس می شود. این مد یک وقفه (compare match) نیز دارد که با برابری دو رجیستر OCR و TCNT به داخل وقفه می رود.

کارکرد این مد طبق دیاگرام زیر است:



شکل ۷-۴

3- Fast PWM

PWM چیست؟

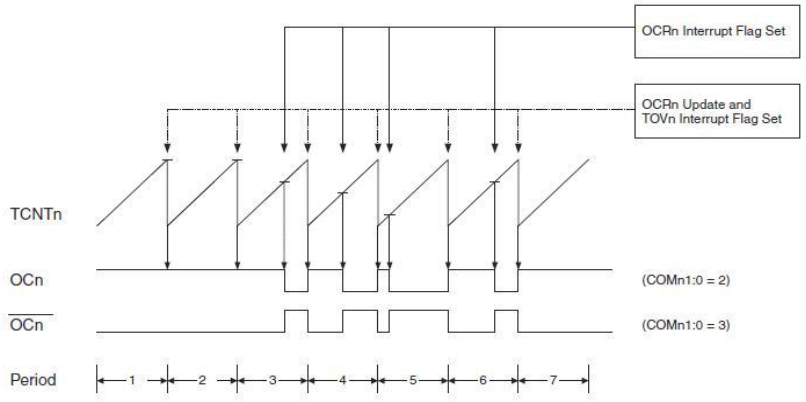
PWM یا مدولاسیون عرض پالس یک روش برای کنترل ولتاژ یا کنترل دور موتورها می باشد. در این پالس ها فرکانس ثابت بوده و duty cycle تغییر می کند.

این مد شبیه مد CTC است با این تفاوت که پریود ثابت بوده و عرض پالس تغییر می کند. در این مد خروجی در ابتدا صفر است و تایمر مانند مد normal از صفر تا ماکزیمم شمرده و با پر شدن دوباره صفر می شود. هر گاه مقدار تایمر با رجیستر OCR معادل خود برابر شد خروجی یک می شود و با رسیدن تایمر به ماکزیمم دوباره خروجی صفر می شود.

اگر وقفه Compare match فعال باشد هرگاه مقدار TCNT با OCR برابر شد برنامه به داخل وقفه می رود.

اگر پایه خروجی تایمر را در حالت non-inverted قرار دهیم پایه خروجی ابتدا صفر بوده و با رسیدن به OCR یک می شود (طبق توضیحات بالا) اما اگر خروجی در حالت inverted باشد پایه خروجی یک بوده و با رسیدن به OCR صفر می شود.

دیاگرام تغییرات این مد در شکل ۷-۵ آورده شده است:



شکل ۷-۵

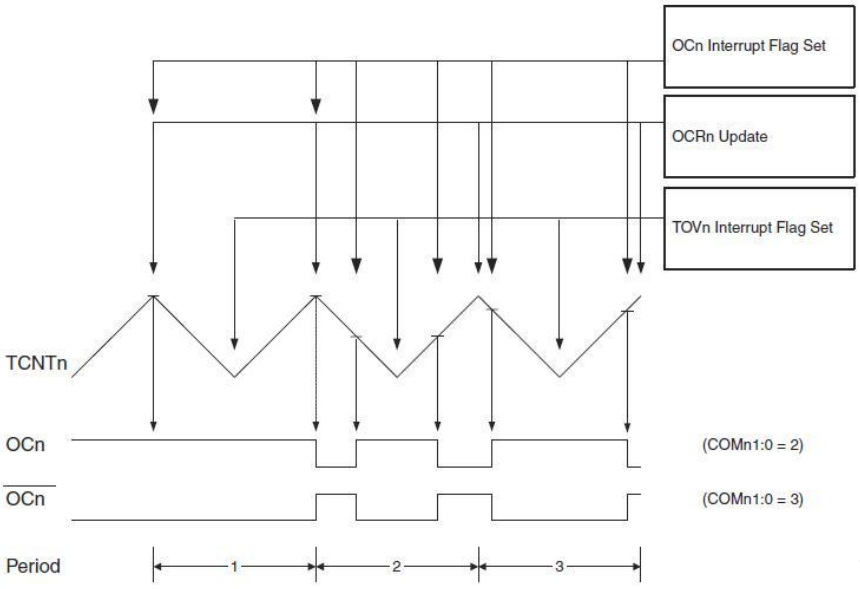
تمرین ۳: برنامه ای بنویسید که سرعت یک موتور DC رفته رفته زیاد شده و با رسیدن به مقدار ماکزیمم، سرعت آن به مرور کم شود و این سیکل ادامه پیدا کند.

4- Phase Correct PWM

این مد نیز شبیه مد fast PWM است. در این مد تایمر از صفر شروع به شمارش می کند، با رسیدن تایمر به مقدار رجیستر OCR پایه خروجی طبق تنظیمات (non-invrted یا inverted) تغییر وضعیت می دهد. با رسیدن تایمر به ماکزیمم، تایمر شروع به شمارش به صورت پایین شمار می کند تا به صفر برسد. در حالت پایین شمار نیز هرگاه دوباره به OCR رسید خروجی معکوس می شود.

اگر وقفه Compare match فعال باشد هرگاه مقدار TCNT با OCR برابر شد برنامه به داخل وقفه می رود.

دیاگرام این مد به صورت زیر است:



شکل ۷-۶

تمرین ۴: برنامه فوق را با مد Phase Correct PWM تکرار کنید و نتیجه را مقایسه نمایید.

عملکرد حالت non-inverted و inverted نیز مانند مد قبلی است.

5- Capture

این مد فقط در تایمر یک قرار دارد. با فعال کردن این مد هرگاه پایه ICP را از بیرون تحریک کنیم محتوای تایمر/کانتر درون رجیستر ICR ریخته می شود.

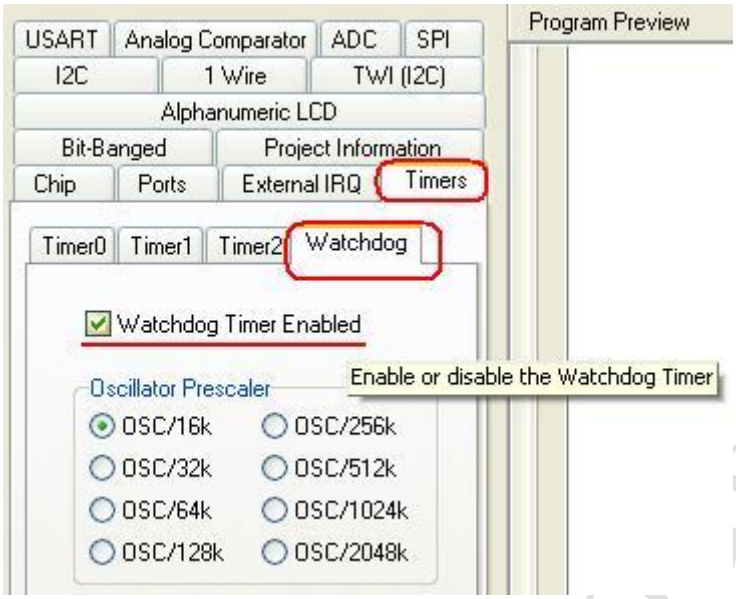
با فعال کردن گزینه Noise Canceler در codewizard تحریکی در پایه ICP معتبر است که حداقل ۴ ماشین سیکل طول بکشد.

تنظیمات قسمت Codewizard بقیه مدها نیز مانند مد نرمال بوده و تفاوت چندانی ندارد.

تایمر Watch dog

در فصل چهارم توضیح مختصری در مورد این تایمر گفته شد. برای فعال کردن تایمر سگ نگهبان در پنجره Codewizard و در قسمت Timers به اخرین سربرگ یعنی Watchdog رفته و گزینه Watchdog timer Enabled را فعال می کنیم. ملاحظه می کنید که باید یک فرکانس برای آن انتخاب کنیم که در حقیقت تقسیمی از فرکانس اسیلاتور اصلی است. هر چه

فرکانس انتخابی بالاتر باشد تایمر زودتر پر شده و میکرو را ریست می کند و اگر فرکانس پایینی را انتخاب کنیم فرصت بیشتری برای ریست کردن این تایمر داریم.



شکل ۷-۷

یکی از روش های استفاده ازاین تایمر به این شکل است که بیت های یک متغیر ۸ بیتی را در خلال برنامه یکی یکی، یک کنیم و در آخر اگر همه بیت ها یک شده بودند (که نشانگر اجرای صحیح برنامه است) تایمر را با دستور زیر ریست می کنیم.

```
#asm("wdr")
```

برای غیر فعال کردن این تایمر باید بیت های چهارم و پنجم از رجیستر WDTCR یکبار یک و یکبار صفر شوند.

```
WDTCR|=0x18;
WDTCR&=0xE7;
```

برای فعال کردن دوباره آن کافی است بیت های فوق را دوباره یک کنیم

```
WDTCR|=0x18;
```

فصل هشتم

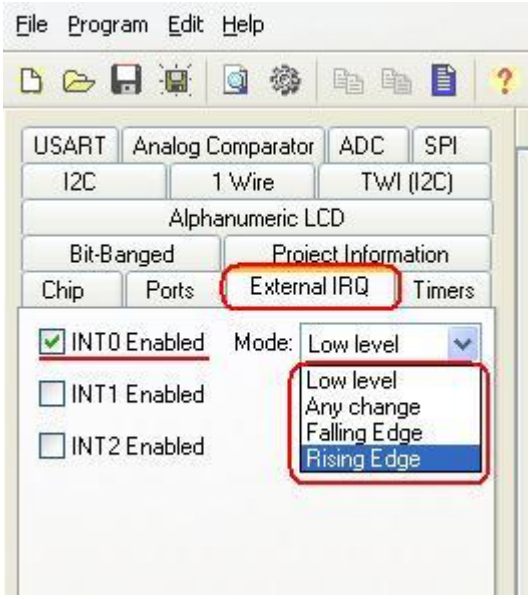
وقفه های خارجی

همانطور که در فصل چهارم گفتیم، وقفه یکی از امکانات بسیار کاربردی میکروهای خانواده AVR است. علاوه بر وقفه هایی که در قسمت های مختلف میکرو وجود دارد (مانند وقفه سرریز یا تطابق در تایمرها) میکروکنترلر ATmega16 دارای سه وقفه خارجی نیز می باشد که با تحریک پایه آنها از بیرون برنامه سریعا به داخل تابع وقفه رفته و دستورات آن را اجرا می کند.

سه بیت با ارزش رجیستر GICR مربوط به فعال سازی و یا غیر فعال سازی وقفه های خارجی است.

همچنین با استفاده از رجیستر MCUCR نوع تحریک پایه های وقفه ها مشخص می شود. وقفه ها عموماً با لبه بالارونده، لبه پایین رونده، سطح صفر و یا تغییر وضعیت فعال می شوند.

برای فعال سازی وقفه های خارجی از پنجره Codewizard به سربرگ External IRQ رفته و با انتخاب هرکدام از وقفه ها و تعیین نوع تحریک آنها، وقفه را فعال می کنیم.



شکل ۸-۱

تمرین ۵: برنامه ای بنویسید که فرکانس ورودی به پایه INT0 را محاسبه کرده و آن را بر روی LCD بر حسب Hz نمایش دهد.

فصل نهم

مبدل آنالوگ به دیجیتال (ADC)

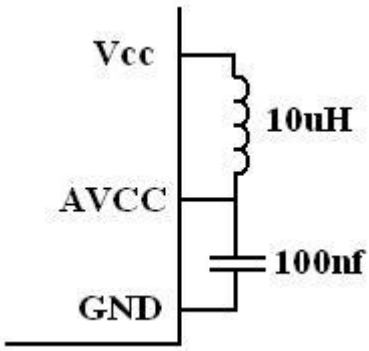
میکروکنترلر ATmega16 دارای ۸ مبدل آنالوگ به دیجیتال ۱۰ بیتی است. همانطور که می دانید یک مبدل آنالوگ به دیجیتال برای تبدیل ولتاژ ورودی به مقادیر دیجیتال نیازمند یک ولتاژ مرجع است. در خانواده AVR سه نوع ولتاژ مرجع برای ADC وجود دارد:

۱ -پایه AREF: برای اتصال یک ولتاژخارجی به میکرو استفاده می شود.

۲ -پایه AVCC: ولتاژ این پایه در حدود Vcc±0.3 می باشد.

۳ -داخلی: ولتاژ مرجع داخلی حدود 2.4v

در حالتی که از پایه AVCC به عنوان ولتاژ مرجع استفاده می کنیم بهتر است برای کاهش نویز این پایه را مطابق شکل ۹-۱ با یک سلف و خازن به Vcc و GND وصل کنیم.



شکل ۹-۱

هنگامی که از پایه AREF به عنوان ولتاژ مرجع استفاده می کنیم می توان این پایه را با یک خازن 100nf به زمین وصل کرد.

حداکثر ولتاژ پایه AREF برابر ولتاژ تغذیه است و همچنین ولتاژ ورودی آنالوگ نیز نباید از Vref بیشتر شود.

ولتاژ پله یا حساسیت ADC

میزان تغییرات ولتاژ آنالوگ ورودی که به ازای آن عدد باینری یک واحد اضافه می شود را ولتاژ پله می گویند.

ولتاژ پله یک ADC از رابطه زیر بدست می آید:

$$\text{ولتاژ پله} = \frac{\text{ولتاژ مرجع}}{2^n}$$

که در این معادله n تعداد بیت ADC است.

به عنوان مثال اگر ولتاژ مرجع 5v و تعداد بیت های ADC ۱۰ بیت باشد ولتاژ پله حدوداً 5mv خواهد بود یعنی به ازای هر 5mv یک واحد به خروجی ADC اضافه می شود.

رجیسترهای ADC

۱- ADCW: رجیستری ۱۶ بیتی است (ADCL و ADCH) که نتیجه در آن ریخته می شود.

۲- ADCSRA: رجیستر کنترلی ADC. بیت های این رجیستر در ادامه آمده است:

ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADEN: هرگاه یک شود ADC فعال شده و هرگاه صفر شود ADC غیرفعال می شود.

ADSC: هرگاه یک شود عمل تبدیل آغاز می شود.

در حالت Free فقط یکبار این بیت یک می شود. ;ADCSRA|=0x40

اما در حالت single هربار که عمل تبدیل را انجام دهیم این بیت را یک می کنیم و بهتر است در انتهای تابع تبدیل نوشته شود.

ADATE: هرگاه این بیت یک شود در حالت free running تریگر خودکار در لبه های مثبت پالس تحریک اتفاق می افتد.

ADIF: هرگاه عمل تبدیل کامل شود این بیت یک می شود و در صورت فعال بودن تابع وقفه CPU به داخل تابع وقفه می پرد.

ADIE: اگر بخواهیم از وقفه ADC استفاده کنیم باید این بیت را یک نماییم.

ADPS0-2: این سه بیت کلاک ADC را تعیین می کند.

۳- رجیستر SFIOR: سه بیت با ارزش این رجیستر (ADTS0-2) ۸ منبع تحریک ADC را مشخص می کنند.

روش های استفاده از ADC:

۱- به صورت معمولی: در این روش تابعی تحت عنوان read_adc فعال می شود که عمل تبدیل را برای کانال های مختلف انجام می دهد و در لوپ اصلی می توان مقدار هر کانال را با دستور زیر خواند:

متغیر مقصد=read_adc(شماره کانال);

در این حالت کانال ۲ را تبدیل کرده و نتیجه را در متغیر a میریزد a=read_adc(2);

۲- استفاده از وقفه ADC: در این حالت تابع وقفه فعال می شود و در صورتی که در codewizard کانال هایی را که مد نظر هست مشخص کرده باشیم، scan کرده و مقادیر آنها را در متغیر آرایه ای adc_data می ریزد. و در لوپ اصلی می توان به صورت زیر مقدار باینری هر کانال را خواند:

a=adc_data[شماره کانال];

۳- استفاده از حالت مد Idle: این حالت تقریبا شبیه حالت اول است با این تفاوت که هم تابع وقفه ساخته می شود و هم تابع read_adc و برای خواندن مقادیر هر کانال باید در لوپ اصلی به صورت زیر بنویسیم:

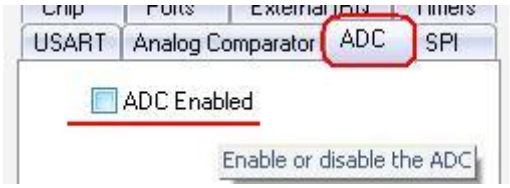
a=read_adc(شماره کانال);

در این حالت در تابع read_adc قبل از دستور return باید دستور فعال شدن ADC را بنویسیم:

ADCSRA|=0x40;

آشنایی با قسمت ADC در Codewizard

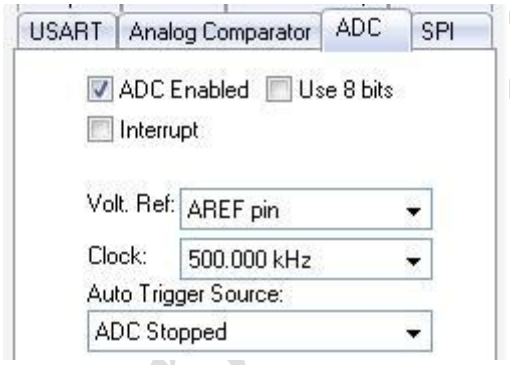
ابتدا برای فعال کردن ADC گزینه ADC Enabled را می زنیم.



شکل ۹-۲

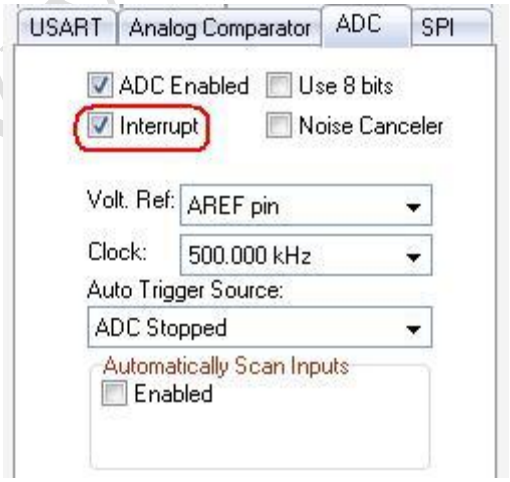
اگر بخواهیم دقت ADC را محدود کنیم گزینه use 8 bits را کلیک می کنیم. بصورت پیش فرض ADC ۱۰ بیتی می باشد.

همانطور که در متن توضیح داده شد روش های مختلفی برای استفاده از ADC وجود دارد. اگر گزینه Interrupt را فعال نکنیم حال اول ایجاد می شود.



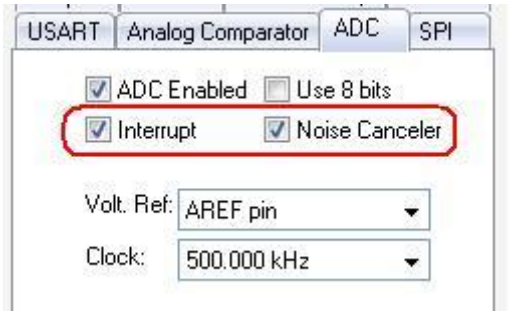
شکل ۹-۳

با فعال کردن این گزینه حالت دوم فعال شده



شکل ۹-۴

و در صورت فعال کردن گزینه Noise Canceler حالت سوم ایجاد می شود.



شکل ۹-۵

در همه حالات ولتاژ مرجع و فرکانس کاری را می توان تغییر داد.

تمرین ۶: با استفاده از مبدل آنالوگ به دیجیتال و سنسور LM35 یک دماسنج طراحی کنید و هر یک دقیقه یکبار دما را بر روی LCD نمایش دهید.

فصل دهم

پورت سریال

USART: ارسال و دریافت بیت به بیت اطلاعات

معرفی رجیسترهای کاربردی در پورت سریال

۱ -رجیستر UDR:

رجیستری جهت ارسال و دریافت اطلاعات به صورت سریال

رجیستر ارسال UDR با رجیستر دریافت کاملا متفاوت می باشد فقط دارای نام مشترک هستند.

۲ -رجیستر UCSRA:

رجیستر کنترلی پورت سریال می باشد.

Rxc	Txc	UDRE				U2X	

U2X: با یک شدن این بیت میزان Baud rate دو برابر می شود.

Rxc: هرگاه در دریافت سریال آخرین بیت UDR کامل شود این بیت یک می شود.

(در این حالت اگر تابع وقفه دریافت سریال فعال باشد برنامه به داخل تابع وقفه می رود.)

Txc: هرگاه در ارسال سریال آخرین بیت UDR ارسال شود این بیت یک می شود.

(در این حالت اگر تابع وقفه ارسال سریال فعال باشد برنامه به داخل تابع وقفه می رود.)

UDRE: هر وقت به هنگام ارسال، UDR آماده گرفتن اطلاعات جدید باشد، این بیت یک می شود.

نحوه ارسال سریال اطلاعات:

```
while(!UCSRA.5);
```

```
UCSRA.5=0;
UDR=a;
```

نحوه دریافت سریال اطلاعات:

```
while(!UCSRA.7);
UCSRA.7=0;
a=UDR;
```

ارسال و دریافت سریال به شکلی که گفته شد کار برنامه نویسی را مشکل می کند. در نرم افزار codevision کتابخانه ای به نام <stdio.h> برای پورت سریال نوشته شده است که استفاده از دستورات آن کار ارسال و دریافت را آسان تر می کند. ولی باید توجه داشته باشید که استفاده از توابع پورت سریال برنامه را سنگین می کند.

توابع ارسال:

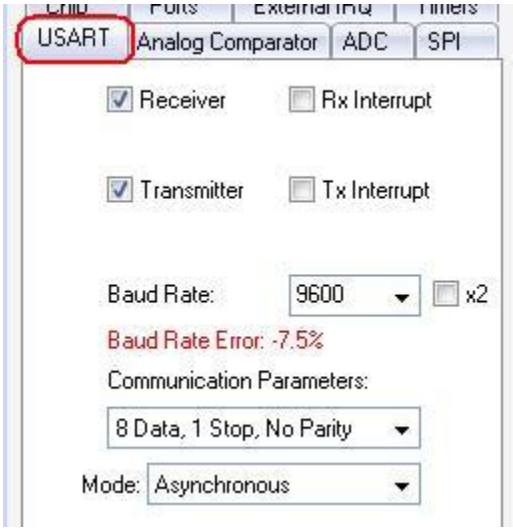
printf(); puts(); putchar(); putsf();

توابع دریافت:

getchar(); gets(); scanf();

آشنایی با قسمت پورت سریال در Codewizard

با رفتن به سربرگ USART خواهیم دید که دو گزینه برای فعال سازی وجود دارد. استفادهبه عنوان فرستنده یا گیرنده. البته هر دو را نیز می توان فعال نمود. با فعال سازی هرکدام از گزینه ها مشخصات زیر ظاهر می شود.



شکل ۱۰-۱

Interrupt که اگر فعال کنیم بعد از فرستادن ۸ بیت (در فرستنده) و یا دریافت ۸بیت (در گیرنده) برنامه به داخل تابع وقفه می رود.

Baud rate: سرعت ارسال و دریافت اطلاعات را مشخص می کند اگر گزینه x2 را بزنیم سرعت دو برابر می شود.

Communication parameters: مشخص کننده نوع بسته دیتا ارسالی و دریافتی می باشد قسمت مد نیز برای همزمان سازی ارسال و دریافت و تعیین میکرو به عنوان Master و Slave با پلاریته کلاک مثبت یا منفی است.

تمرین ۷: بدون استفاده از توابع پورت سریال نام خود را بصورت سریال ارسال کرده و نتیجه را در پنجره Hyper Terminal مشاهده کنید.

تمرین ۸: به کمک توابع پورت سریال برنامه ای بنویسید که طول و عرض یک مستطیل را از پورت سریال دریافت کرده و محیط و مساحت آن را محاسبه و با پورت سریال ارسال کند.

فصل یازدهم

مقایسه کننده آنالوگ

میکروکنترلر ATmega16 علاوه بر ۸ ورودی ADC دارای یک مقایسه کننده آنالوگ است که با اعمال دو ولتاژ به ورودی های مقایسه کننده نتیجه را اعلام می کند. این ورودی ها که بر روی میکرو با پایه های AIN0(ورودی مثبت) وAIN1(ورودی منفی) می باشند با یکدیگر مقایسه شده و اگر ولتاژ ورودی مثبت بیشتر از ورودی منفی شود خروجی مقایسه کننده آنالوگ (ACO)یک می شود.

به دلیل اینکه مقایسه کننده آنالوگ دارای خروجی مشخص بر روی پایه های میکرو نیست از وقفه آن استفاده می کنیم.

یکی از منابع تحریک کننده مد Capture در تایمر ۱ خروجی مقایسه کننده آنالوگ است.

وقفه مقایسه کننده آنالوگ با چند شرط برقرار می شود:

۱- toggle: **هرگاه خروجی مقایسه کننده آنالوگ تغییر وضعیت دهد.**

۲- rising edge: **هرگاه در خروجی مقایسه کننده آنالوگ لبه بالا رونده اتفاق افتد.**

۳- falling edge: **هرگاه در خروجی مقایسه کننده آنالوگ لبه پایین رونده اتفاق افتد.**

رجیستر ACSR رجیستر کنترلی مقایسه کننده آنالوگ است که در ادامه با بیت های آن آشنا می شوید:

ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0

ACD: اگر یک شود مقایسه کننده آنالوگ از کار می افتد و پیش فرض یک است.

ACBG: اگر یک شود از داخل یک ولتاژ ثابت به ورودی مثبت مقایسه کننده داده می شود.

ACO: با یک شدن این بیت خروجی مقایسه کننده مستقیماً به ACO وصل می شود.

ACI: با توجه به نوع تنظیمات تحریک وقفه هرگاه یک شودCPU به تابع وقفه پرش می کند.

ACIE: با یک شدن این بیت وقفه مقایسه کننده فعال می شود.

ACIC: با یک شدن این بیت تحریک مد Capture با خروجی مقایسه کننده آنالوگ اتفاق می افتد.

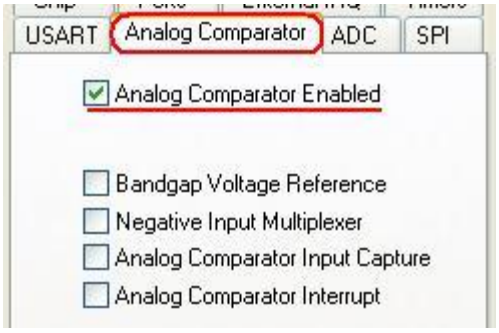
ACIS0_1: جهت انتخاب نوع تحریک وقفه می باشد.

اگر بیت SFIOR.3 یک شود (ACME) یکی از ورودی های ADC به عنوان پایه منفی انتخاب می شود.

سه بیت کم ارزش رجیستر DMUX (MUX0_2) جهت انتخاب یکی از ورودی های ADC به عنوان ورودی آنالوگ منفی انتخاب می شود. توجه داشته باشید که در این حالت باید بیت ADE از رجیستر ADCSRA صفر باشد.

تنظیمات قسمت مقایسه کننده آنالوگ در Codewizard

با مراجعه به سربرگ Analog Comparator و تایید گزینه Analog Comparator Enabled مقایسه کننده آنالوگ فعال می شود.



شکل ۱۱-۱

با انتخاب گزینه اول یعنی Bandgap Voltage **یک ولتاژ داخلی ثابت به ورودی مثبت مقایسه کننده اعمال می شود.**

Negative Input Multiplexer: **جهت تعیین یکی از ورودی های** ADC **به عنوان ورودی منفی**

Analoge Comparator Input Capture: **با تایید این گزینه تحریک مد** Capture **از تایمر یک با خروجی مقایسه کننده اتفاق می افتد.**

Analog Comparator Interrupt: **برای فعال سازی وقفه مقایسه کننده آنالوگ. با تایید آن سه گزینه برای چگونگی اتفاق افتادن وقفه پیش روی ماست:**

۱ -هر گونه تغییر وضعیت در خروجی مقایسه کننده

۲ -ایجاد لبه پایین رونده در خروجی

۳ -ایجاد لبه بالا رونده در خروجی

تمرین ۹: برنامه ای بنویسید که دمای دو نقطه را با یکدیگر مقایسه کرده و اگر دمای نقطه اول ازدمای نقطه دوم بیشتر شد Cooler **روشن و اگر کمتر شد** Heater **روشن شود.**

فصل دوازدهم

آشنایی با ارتباط سریال SPI

مزایای این رابط به شرح زیر است:

۱- **استفاده از مد** Idle

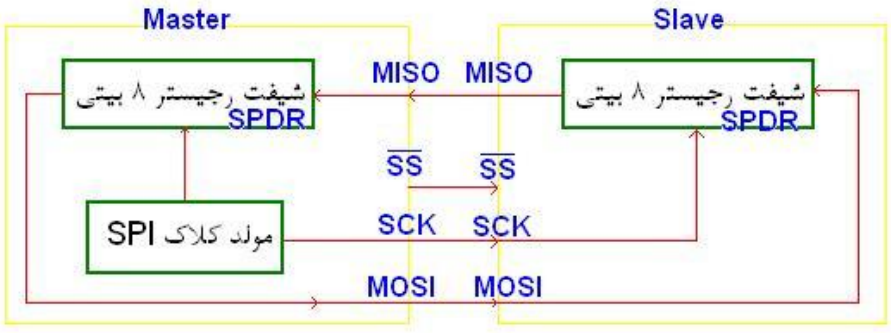
۲- **داشتن تابع وقفه**

۳- **استفاده بصورت** Master **و** Slave

۴- **قابلیت دوبرابر شدن کلاک**

۵- **ارتباط دوطرفه همزمان** (Full Duplex)

نحوه ارتباط سخت افزاری دو دستگاه دارای رابط SPI **طبق شکل زیر است:**



شکل ۱۲-۱

در پروتکل SPI **هرگاه** Master **پایه** SS **را صفر کند ارتباط سریال آغاز می شود.**

پایه های SS و SCK و MOSI برای Slave به عنوان ورودی و برای Masterبه عنوان خروجی تعریف می شود. همچنین پایه MISO برای Master ورودی و برای Slave خروجی است.

آشنایی با رجیسترهای مهم SPI:

۱ -رجیستر SPDR: برای نگهداری اطلاعات ارسالی و دریافتی می باشد.

۲ -رجیستر SPSR: بیت آخر این رجیستر تحت عنوان SPIF هرگاه که ارسال داده یا دریافت آن کامل شود این بیت یک می شود تا CPU به تابع وقفه برود.

۳ -رجیستر SPCR:

SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

SPIE: با یک شدن این بیت وقفه SPI فعال می شود.

SPE: با یک شدن این بیت پروتکل SPI فعال می شود.

DORD: اگر یک شود اولین بیت ارسالی LSB است.

MSTR: اگر یک شود میکرو به عنوان Master انتخاب می شود.

CPOL: این بیت مشخص می کند که پایه SCK در حالت بیکاری صفر باشد یا یک.

CPHA: صفر و یک بودن این بیت مشخص می کند که در کدام لبه کلاک نمونه برداری انجام شود.

SPR0-1: این دو بیت میزان کلاک SPI را مشخص می کند

استفاده از کتابخانه SPI

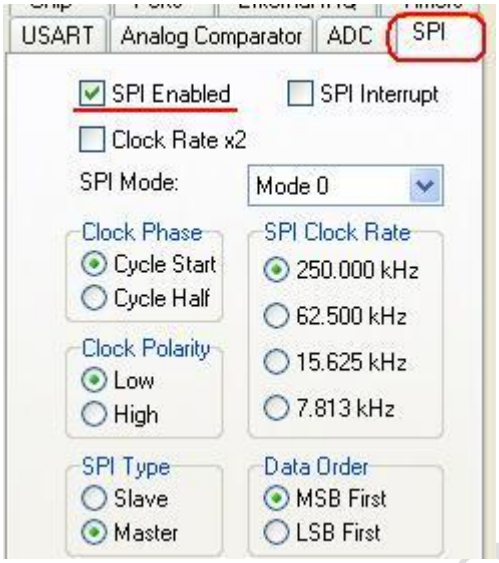
با استفاده از این کتابخانه و با دستور SPI() می توان ارسل و دریافت SPI را براحتی انجام داد.

a=spi(b); ارسال و دریافت a=spi(); دریافت

علاوه بر تابع فوق با استفاده از وقفه SPI نیز می توان عمل ارسال و دریافت را انجام داد. به این صورت که هرگاه عمل ارسال و دریافت کامل شود CPU به تابع وقفه پرش می کند.

تنظیمات قسمت پروتکل SPI در Codewizard

با تایید گزینه SPI Enabled از سربرگ SPI این پروتکل فعال می شود.



شکل ۱۲-۲

هر یک از قسمت ها در ادامه توضیح داده شده است:

SPI Interrupt: **وقفه SPI را فعال می کند.**

Clock Rate x2: **کلاک (سرعت) SPI را دو برابر می کند.**

SPI Mode: **مد SPI را تغییر می دهد. (تغییر مد تنها منجر به تغییر پارامترهای فاز و پلاریته کلاک می شود که خود ما نیز می توانیم آنها را تغییر دهیم)**

Clock Phase: **مشخص می کند که در کدام لبه کلاک نمونه برداری انجام شود.**

Clock Polarity: **مشخص می کند که پایه SCK در حالت بیکاری صفر باشد یا یک.**

SPI Clock Rate: **جهت تعیین کلاک (سرعت انتقال اطلاعات)**

SPI Type: **برای انتخاب میکروکنترلر به عنوان Master یا Slave**

به علت اینکه در SPI برای همزمانی کلاک از سوی Master صادر می شود، اگر میکرو را به عنوان Slave انتخاب کنیم قسمت انتخاب کلاک حذف می شود.

Data Order: اگر بخواهیم ارسال دیتا از بیت کم ارزش شروع شود LSB First و اگر بخواهیم از بیت پر ارزش شروع شود MSB First را انتخاب می کنیم.

تمرین ۱۰: با استفاده از پروتکل SPI یک میکروکنترلر را به دو میکرو دیگر متصل کرده و به هرکدام از میکروها یک رشته ارسال شود. در Slaveها رشته های دریافتی بر روی LCD نمایش داده شود.

اتصال حافظه های MMC به میکروکنترلر

حافظه های MMC با پروتکل SPI کار می کنند و پایه های آن بصورت زیر به میکروکنترلر وصل می شود:

DO → MISO

DI → MOSI

CS → SS

SCK → SCK

استفاده از کتابخانه mmc.h

با استفاده از دستورات زیر که در این کتابخانه قرار دارد می توان با حافظه های MMC ارتباط برقرار کرد:

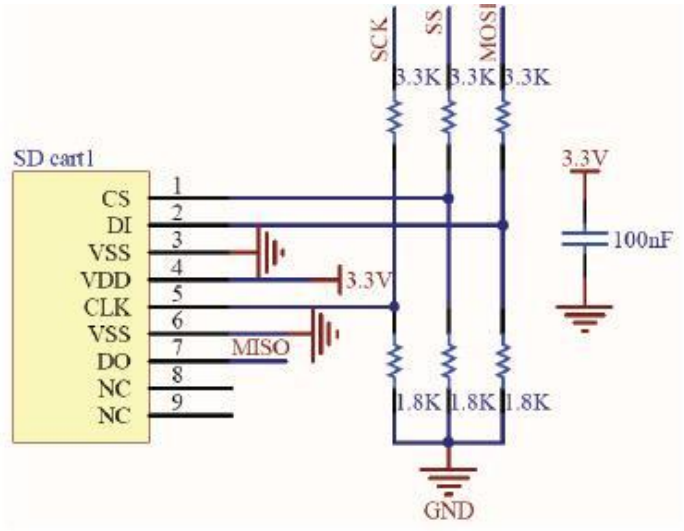
1- mmc_init();
2- mmc_reset();
3- mmc_write(متغیری که قرار است اطلاعاتش در آن قرار گیرد , شماره سکتور);

خانه های mmc بصورت سکتور دسته بندی شده اند که به طور معمول هر 512 بایت یک سکتور است.

4- mmc_read(متغیری که نتیجه خواندن در آن ریخته می شود , شماره سکتور);

تمرین ۱۱: یک حافظه SD را به میکروکنترلر متصل کرده و دو رشته دلخواه را در ۲ سکتور از آن ریخته و سپس خوانده و بر روی LCD نمایش دهید.

با توجه به اینکه ولتاژ کاری MMC، 3.3v می باشد باید بر روی پایه های DI, CS, SCK یک تقسیم مقاومت بصورت زیر قرار داد.



شکل ۱۲-۳

برای شبیه سازی حافظه MMC در نرم افزار پروتئوس باید فایل cardimage.mmc را در آن load کرد. برای اینکار بر روی mmc کلیک راست کرده و گزینه Edit Properties را انتخاب می کنیم. در این پنجره از قسمت Card Image File فایل فوق را load می کنیم.

نکته مهم این است که بعد از هر بار اجرا و توقف کار برنامه باید فایل فوق را با نرم افزار notepad باز کرده و محتویات آن را پاک کنیم.

فصل سیزدهم

ارتباط دوسیمه I2C

I2C نوعی ارتباط سریال است که فقط از دو سیم جهت ارتباط استفاده می کند.

پایه های این پروتکل به شرح زیر است:

SDA: جهت ارسال و دریافت Data

SCL: جهت تنظیم Clock

مزایای این پروتکل عبارتند از:

۱ -**استفاده از دو سیم**

۲ -**فراخوانی عمومی**

۳ -**ارتباط Master با حداکثر ۱۲۷ Slave مختلف**

۴ -**دارای وقفه**

فرمت بسته آدرس در i2c:

حداکثر ۹ بیتی می باشد. ۷ بیت با ارزش جهت آدرس Slave مورد نظر.

بیت هشتم جهت تعیین حالت خواندن یا نوشتن (اگر صفر باشد یعنی می خواهیم در Slave بنویسیم و اگر یک باشد یعنی می خواهیم از Slave بخوانیم)

بیت نهم بیت تصدیق می باشد. هربار Slave بخواهد به درخواست Master پاسخ مثبت دهد این بیت صفر شده و به عبارت دیگر خط ارتباطی SDA صفر می شود

فرمت بسته داده در i2c:

۸ بیت اول داده مورد نظر می باشد و بیت نهم بیت تصدیق است.

برای اتصال Slave های مختلف به Master کافی است آنها را بصورت موازی و مستقیماً به Master وصل کنیم.

پایه های SDA و SCL باید Pull up شوند تا در حالت بیکاری در وضعیت یک باشند.

استفاده از کتابخانه I2C.h

در این کتابخانه دستورات لازم جهت ارسال و دریافت موجود است. در پروتکل I2C هر بایت ارسال یا دریافت باید یک شروع و یک پایان داشته باشد.

دستورات این کتابخانه عبارتند از:

1- i2c_init();

برای آماده سازی اولیه پروتکل I2C

2- i2c_start();

شروع پروتکل I2C را مشخص می کند.

3- i2c_stop();

پایان پروتکل I2C را مشخص می کند.

4- i2c_write(داده یا آدرس مورد نظر);

جهت ارسال یک داده و یا مشخص کردن آدرس از آن استفاده می شود.

دستور i2c_write اگر برای آدرس دهی استفاده شود تکلیف بیت کم ارزش جهت خواندن یا نوشتن باید مشخص شود.

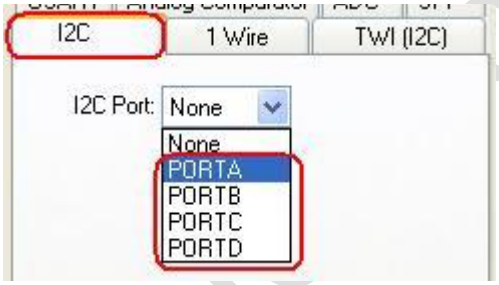
5- i2c_read();

در دستور i2c_read اگر داخل پرانتز عدد صفر را بنویسیم بعد از هر بایت خواندن اطلاعات بیت تصدیق را ایجاد نمی کند.

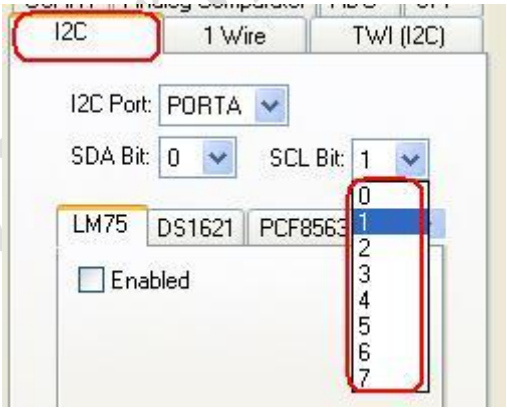
تنظیمات قسمت پروتکل I2C در Codewizard

یکی از مزایای این پروتکل در AVR این است که می توانیم دو پایه ورودی و خروجی آن یعنی دیتا و کلاک را خودمان انتخاب کنیم.

برای فعال سازی این پروتکل به سربرگ I2C رفته و از قسمت I2C Port یکی از پورت ها را انتخاب و سپس دو پین آن را به عنوان دیتا و کلاک تعیین می کنیم. (SDA بیت دیتا و SCL بیت کلاک می باشد)



شکل ۱۳-۱



شکل ۱۳-۲

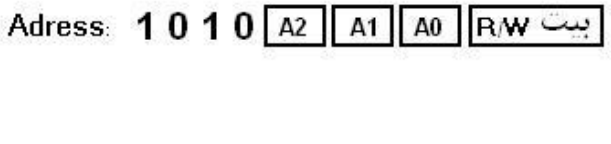
در اینجا چند آی سی که می توانند با پروتکل I2C با میکروکنترلر AVR ارتباط برقرار کنند را می بینید. به عنون مثال می توانیم سنسور دمای LM75 یا آی سی ساعت یعنی DS1307 را فعال کنیم. توضیح چگونگی کار با آی سی ساعت DS1307 در قسمت های بعدی آمده است.

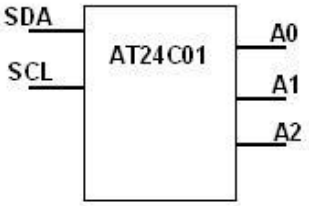
استفاده از EEPROM های سری AT24C**

به جای دو ستاره، حجم EEPROM بر حسب کیلو بیت قرار می گیرد. این خانواده دارای ظرفیت های مختلفی (1kbit, 2kbit, 64kbit) می باشد.

این سری از EEPROM ها با پروتکل I2C کار می کنند.

به عنوان نمونه در خانواده 1kB و 2kB آدرس بصورت زیر است:





شکل ۱۳-۳

آدرس دهی EEPROM های سریال بایت بایت می باشد.

پایه write protect (wp):

یک پایه کنترلی می باشد (Low Active) به طوری که اگر صفر باشد می توان عمل خواندن و نوشتن را انجام داد.

مراحل نوشتن در EEPROM سریال:

۱- شروع کار i2c_start();
۲- آدرس Slave مورد نظر به کمک دستور i2c_write();
۳- آدرس خانه ی حافظه مورد نظر از Slave انتخاب شده به کمک دستور i2c_write();
۴- ارسال داده مورد نظر به کمک دستور i2c_write(); (داده را باید بصورت بایتی وارد کنیم)
۵- پایان کار i2c_stop();

برای خانواده های 1k و 2k می توان مرحله ۴ را برای ۸ بایت داده مختلف تکرار کرد.

برای خانواده های با ظرفیت بالاتر می توان این مرحله را برای ۱۶ بایت داده مختلف تکرار کرد.

مراحل خواندن از EEPROM سریال:

۱- **شروع کار** ;i2c_start()

۲- **آدرس Slave مورد نظر به کمک دستور** ;i2c_write()

بیت R/W در این مرحله صفر است.

۳- **آدرس خانه ی حافظه مورد نظر از Slave انتخاب شده به کمک دستور** ;i2c_write()

۴- ;i2c_start()

۵- **آدرس Slave مورد نظر به کمک دستور** ;i2c_write()

بیت R/W در این مرحله باید یک باشد.

۶- **دریافت داده با دستور** ;i2c_read()

۷- **پایان کار** ;i2c_stop()

برای خانواده های 1k و 2k می توان مرحله ۶ را ۸ بار تکرار کرد. برای خانواده های با ظرفیت بالاتر می توان این مرحله را ۱۶ بار تکرار کرد.

تمرین ۱۲: برنامه ای بنویسید که نام شما را در EEPROM ذخیره کرده و سپس آن را خوانده و بر روی LCD نمایش دهد.

RTC در AVR

خانواده AVR مجهز به RTC (ساعت واقعی) می باشند. که این حالت توسط تایمر شماره ۲ فعال می شود.

برای RTC از کریستال 32.768 KHz بین دو پایه TOSC1 و TOSC2 استفاده می کنیم.

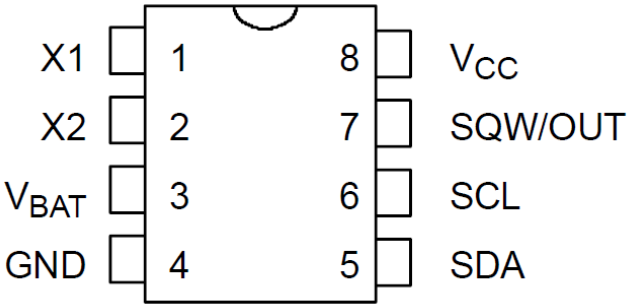
طرز ساخت 1 Hz

اگر منبع کلاک تایمر ۲، پایه TOSC انتخاب و مقدار کلاک آن CLK/128 تعیین شود هر شمارش تایمر ۲ برابر 1/256 ms می شود که با توجه به اینکه این تایمر ۸ بیتی است و ۲۵۶ رقم شمارش می کند پس سر ریز تایمر هر یک ثانیه یکبار اتفاق می افتد. برای استفاده از این یک ثانیه دقیق، می توان وقفه تایمر ۲ را فعال کرد و در تابع وقفه برنامه ساعت را بنویسیم. بنابراین هر یک ثانیه یکبار برنامه به داخل تابع وقفه می رود و دستورات را اجرا می کند.

تمرین: یک ساعت و تقویم به کمک تایمر ۲ طراحی کرده و ساعت را در سطر بالا و تقویم را در سطر پایین LCD نمایش دهید.

آشنایی با آی سی DS1307

این آی سی یک RTC می باشد که با پروتکل i2c با پردازنده ها از جمله AVR ارتباط برقرار می کند. توضیح پایه های این آی سی در ادامه آمده است.



شکل ۱۳-۴

X1 و X2: این پایه ها به کریستال 32.768 KHz متصل می شود.

V_{BAT} : باطری پشتیبان 3v از جنس لیتیوم (این باطری در زمان قطع برق زمان را محاسبه می کند تا با قطع و وصل برق ساعت نیاز به تنظیم مجدد نداشته باشد. به دلیل مصرف پایین آی سی یک باطری بدون برق حدود ده سال اطلاعات ساعت را محاسبه می کند)

SDA: باس داده i2c

SCL: باس کلاک i2c

همانطور که در مبحث i2c گفته شد دو پایه فوق حتما باید Pull up شوند.

SQW/OUT: این پایه در صورتی که بیت SQWE از رجیستر DS1307 یک باشد می تواند یک پالس مربعی با فرکانس های مختلف تولید کند.

معرفی رجیستر DS1307

OUT	0	0	SQWE	0	0	RS1	RS0

RS0,1: جهت تعیین فرکانس پالس مربعی پایه SQW/OUT

SQWE: در صورتی که یک باشد می تواند یک پالس مربعی با فرکانس های مختلف تولید کند

OUT: جهت تعیین وضعیت پایه SQW/OUT در حالت بیکاری. اگر این بیت صفر باشد پایه SQW/OUT در حالت بیکاری صفر است و اگر یک باشد پایه فوق در حالت بیماری یک می باشد.

کتابخانه ds1307.h

با استفاده از این کتابخانه می توان با چند دستور ساده با این آی سی ارتباط برقرار کرد.

توابع کتابخانه:

1) rtc_init(rs.sqwe,out);

rs: جهت تعیین کلاک که باید از صفر تا ۳ باشد. طبق جدول زیر:

0	1 Hz
1	4.096 kHz
2	8.192 kHz
3	32.768 kHz

sqwe: بیت SQWE رجیستر DS1307

Out: بیت OUT رجیستر DS1307

2) rtc_get_time(&hour,&min,&sec);

برای خواندن مقادیر ساعت و دقیقه و ثانیه از آی سی.

متغیرهای فوق باید از نوع unsigned char تعریف شوند.

3) rtc_set_time(hour,min,sec);

جهت تنظیم ساعت

4) rtc_get_date(&day,&month,&year);

برای خواندن مقادیر روز و ماه و سال از آی سی.

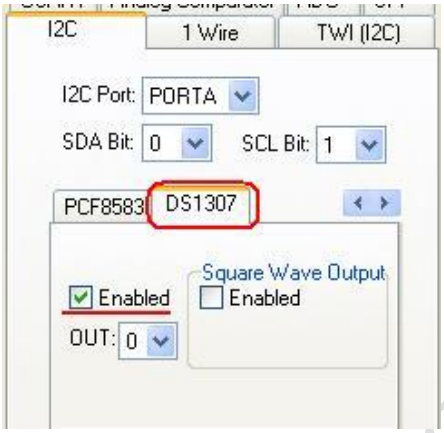
متغیرهای فوق باید از نوع unsigned char تعریف شوند.

5) rtc_set_date(day,month,year);

جهت تنظیم تاریخ

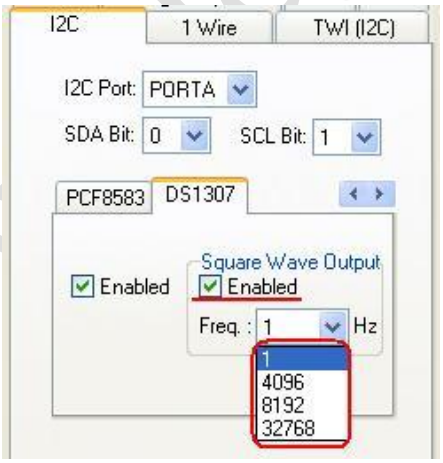
تنظیمات قسمت آی سی DS1307 در Codewizard

برای فعال سازی ارتباط این آی سی با میکرو باید پروتکل I2C را فعال کنیم. بنابراین به قسمت I2C رفته و پس از انتخاب پورت و پین های دلخواه، در قسمت پایین سربرگ DS1307 را انتخاب و گزینه Enabled را تایید می کنیم



شکل ۱۳-۵

اگر بخواهیم از موج مربعی ایجاد شده بر روی پایه SQW/OUT استفاده کنیم گزینه Enabled از قسمت Square Wave Ouput را انتخاب و سپس فرکانس نوسان این پایه را تعیین می کنیم.



شکل ۱۳-۶

همچنین اگر پایه SQW/OUT را فعال نکرده باشیم با استفاده از قسمت OUT می توانیم صفر یا یک بودن این پایه را در وضعیت بیکاری مشخص کنیم.

تمرین: برنامه تمرین قبل را با استفاده از آی سی DS1307 بنویسید.

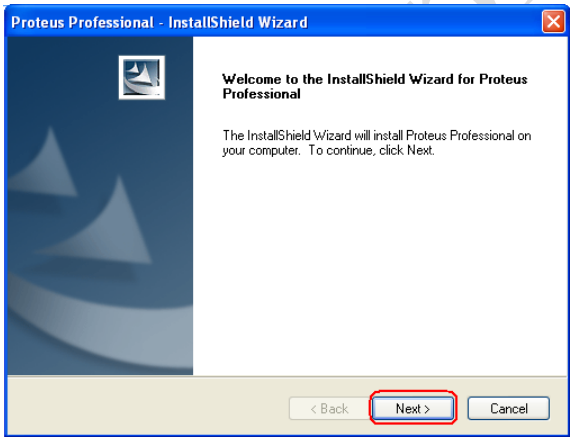
فصل چهاردهم

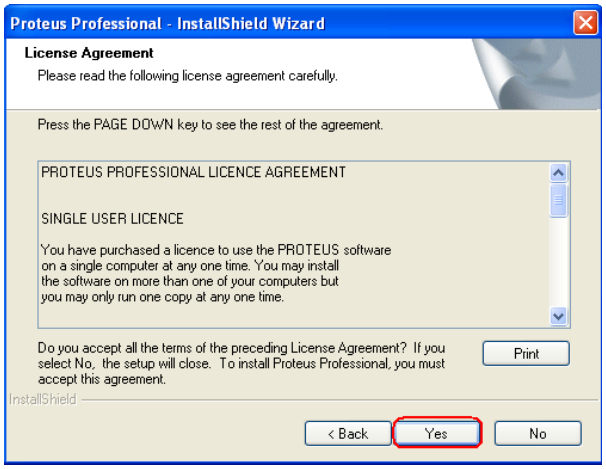
شبیه سازی در پروتئوس

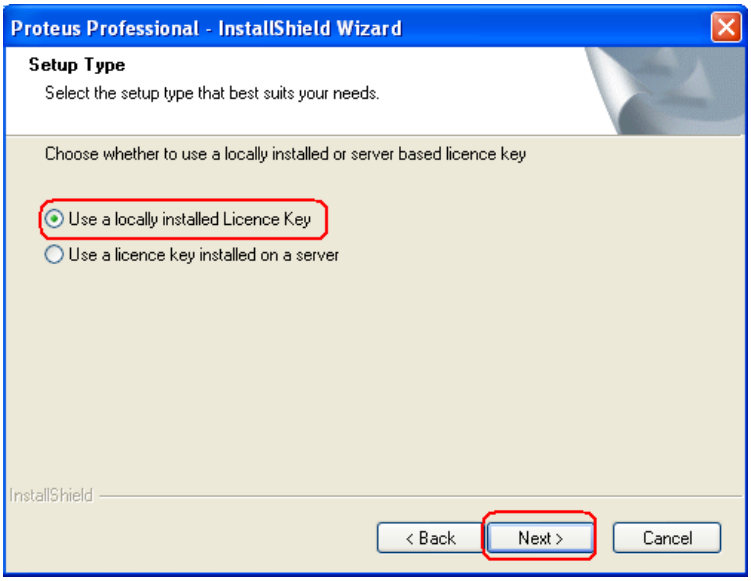
نصب نرم افزار Proteus

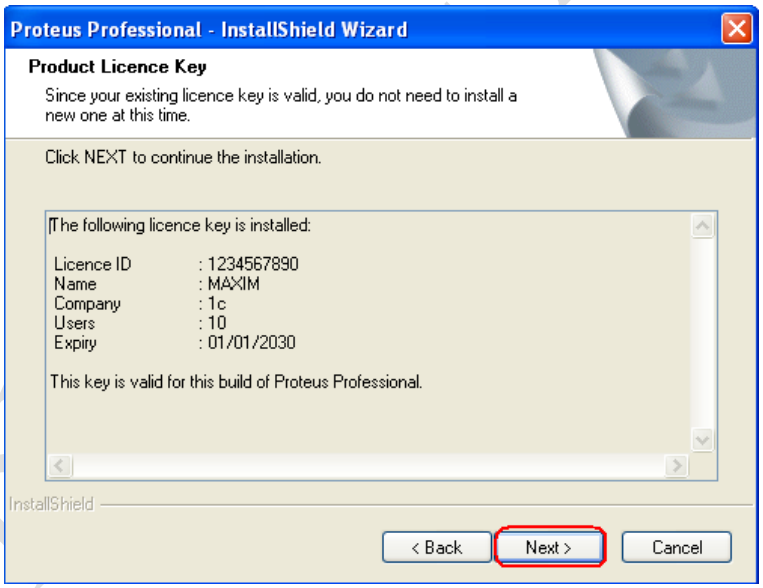
نصب نرم افزار Proteus مانند دیگر نرم افزار ها است اما به دلیل نکاتی در نصب آن در اینجا آورده شده است.

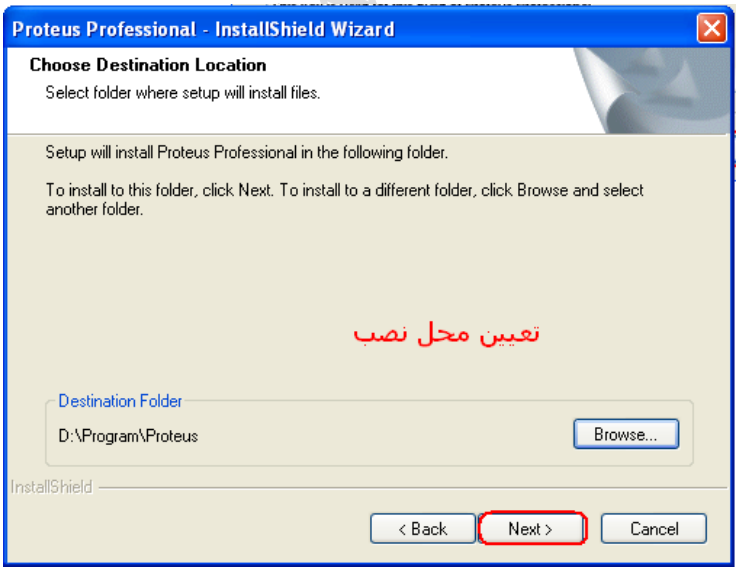
در پوشه Proteus 7.1 SP2 Full فایل Setup71.exe را اجرا می کنیم.

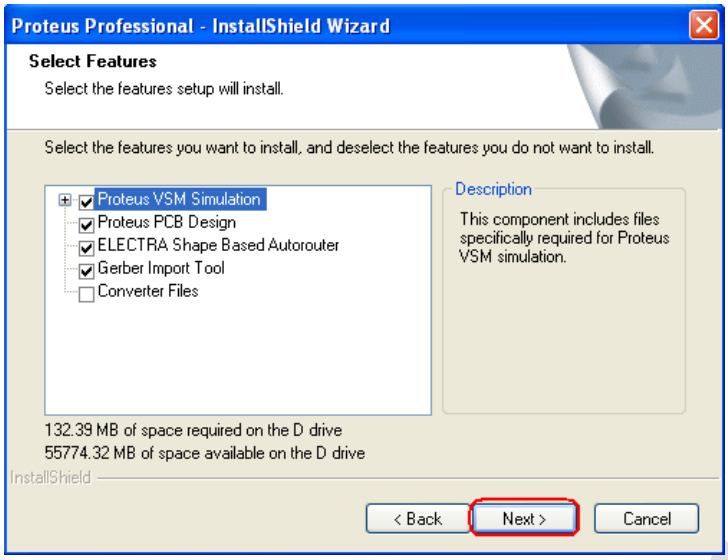


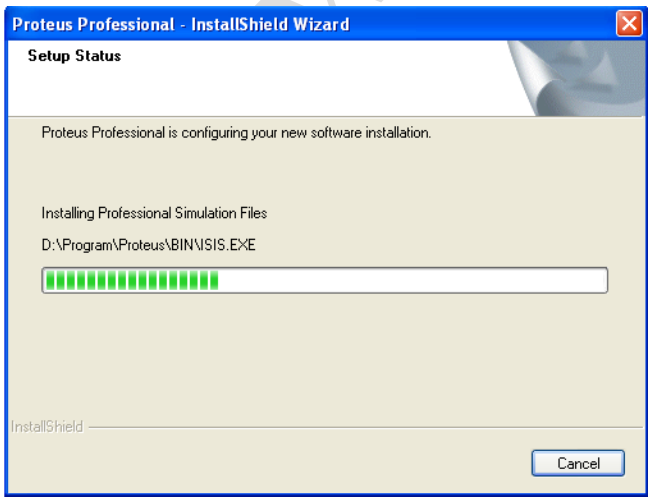



Proteus Professional - InstallShield Wizard
Setup Type
Select the setup type that best suits your needs.
Choose whether to use a locally installed or server based licence key
Use a locally installed Licence Key
Use a licence key installed on a server
InstallShield
< Back
Next >
Cancel


Proteus Professional - InstallShield Wizard
Product Licence Key
Since your existing licence key is valid, you do not need to install a new one at this time.
Click NEXT to continue the installation.
The following licence key is installed:
Licence ID : 1234567890
Name : MAXIM
Company : 1c
Users : 10
Expiry : 01/01/2030
This key is valid for this build of Proteus Professional.
InstallShield
< Back
Next >
Cancel

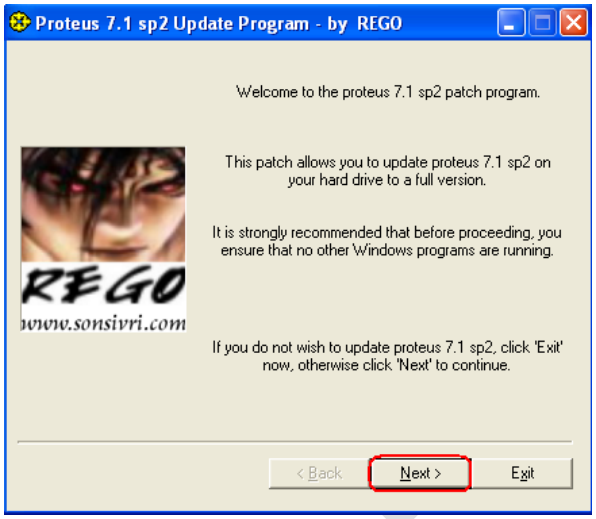

Proteus Professional - InstallShield Wizard
Choose Destination Location
Select folder where setup will install files.
Setup will install Proteus Professional in the following folder.
To install to this folder, click Next. To install to a different folder, click Browse and select another folder.
تعیین محل نصب
Destination Folder
D:\Program\Proteus
Browse...
InstallShield
< Back
Next >
Cancel

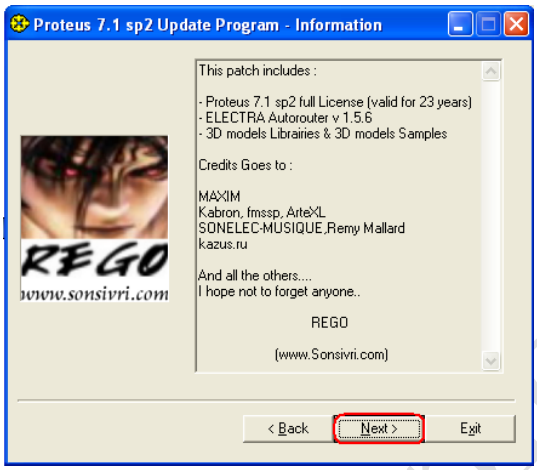


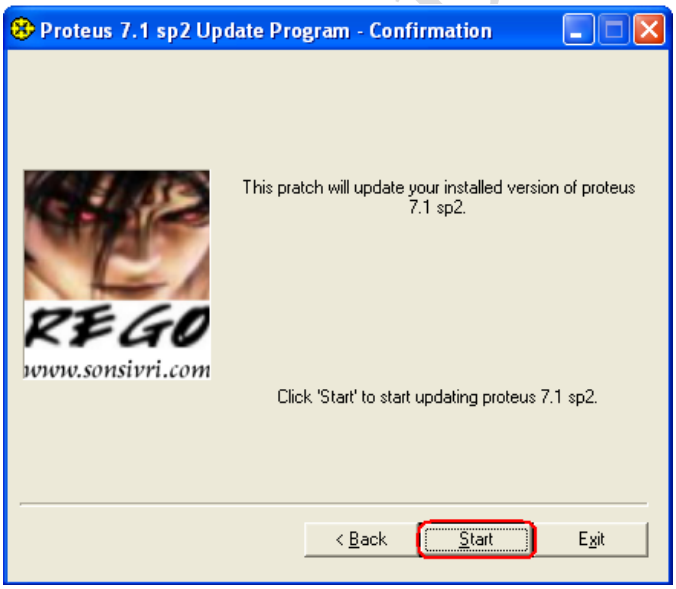


سپس از آدرس زیر فایل Patch.exe را اجرا می کنیم

...\Proteus 7.1 SP2 Full\PROTEUS_7.1_SP2-v1.0\PROTEUS 7.1 SP2



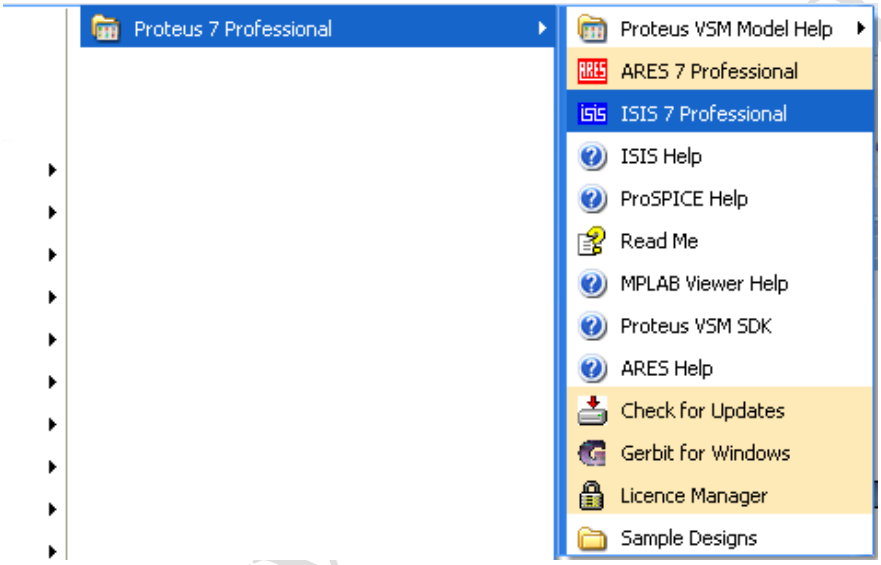


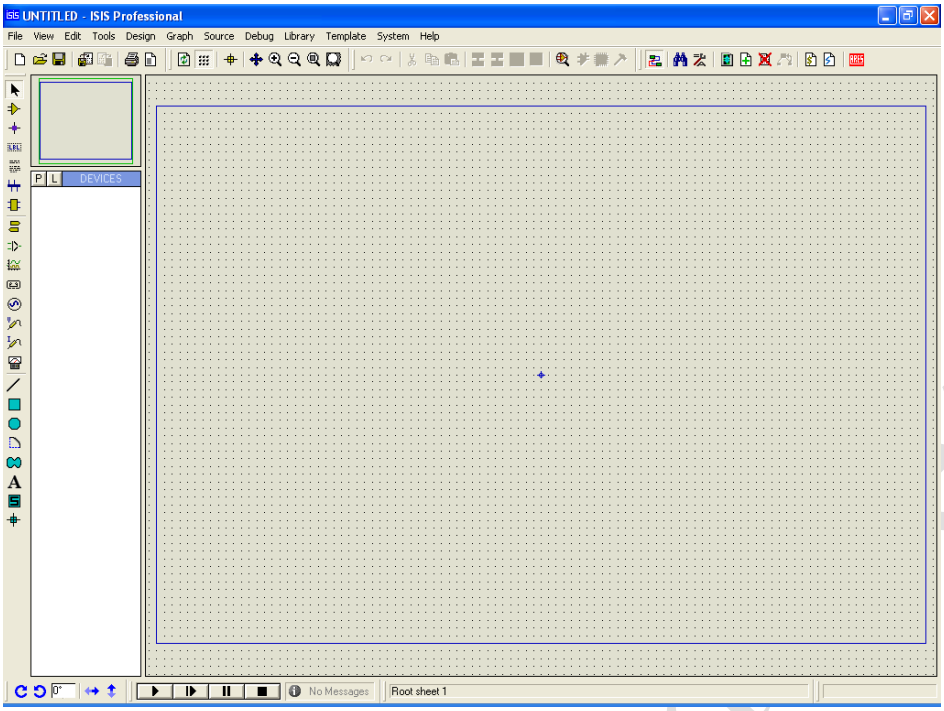


آموزش کار با نرم افزار Proteus

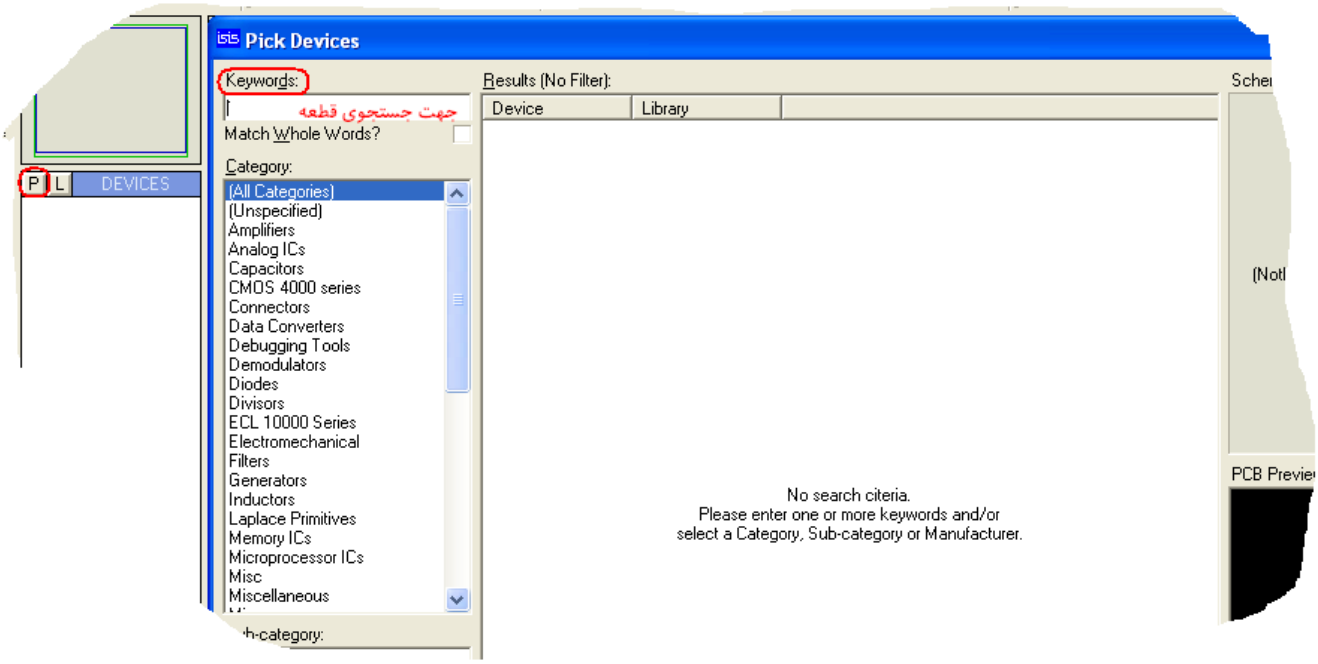
نرم افزار Proteus عمل شبیه سازی مدار به صورت واقعی با حداقل خطا را انجام می دهد. به صورتی که شما می توانید قبل از بستن یک مدار الکترونیکی، آن را در محیط Proteus تست کرده و نواقص احتمالی آن را برطرف کنید. در این فصل از آموزش میکرو ، آموزش مقدماتی این نرم افزار را برای شما آورده ایم تا بتوانید برنامه هایی را که می نویسید در همان لحظه به صورت عملی مشاهده کنید.

برای باز کردن برنامه، از محل نصب آن و یا میانبری که در Start menu ایجاد کرده اید، گزینه ی ISIS 7 Professional را اجرا کنید.

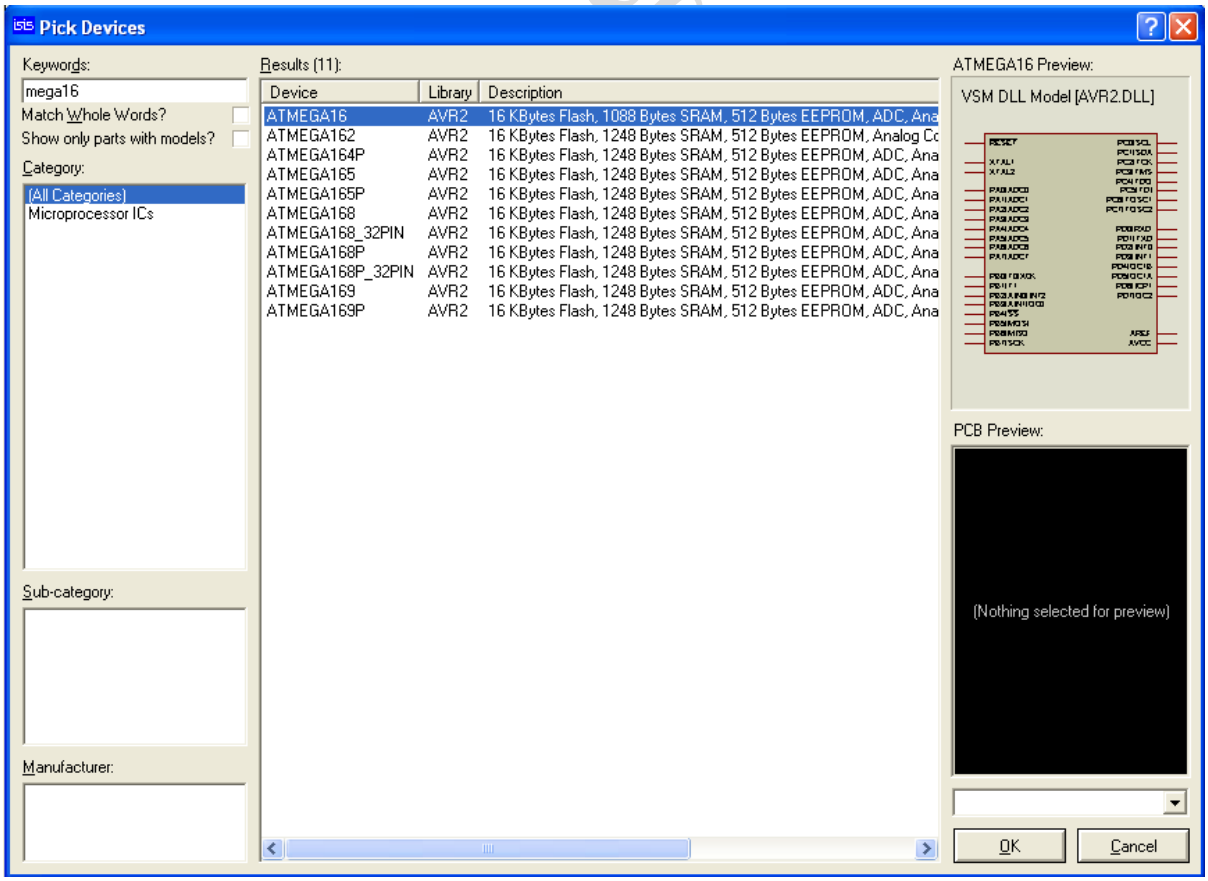




شما می توانید برای فراخوانی یک قطعه از کتابخانه بر روی کلید P (Pick Devices) کلیک کنید. در پنجره ی باز شده می توانید قطعه مورد نظر خود را پیدا کرده و بر روی آن دابل کلیک کنید تا به لیست قطعات سمت چپ صفحه اضافه شود. با توجه به زیاد بودن تعداد قطعات ، پیدا کردن یک قطعه خاص در این لیست کار دشواری است . برای همین شما می توانید با استفاده از قسمت keywords (کلمه کلیدی) که در بالای نام قطعات قرار دارد ، نام قطعه مورد نظر خود یا قسمتی از آن را وارد کرده تا آن قطعه برای شما نمایش داده شود.

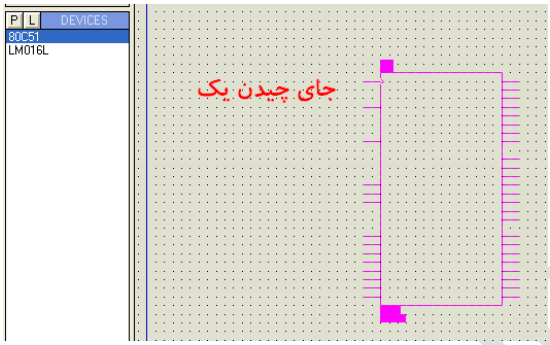


برای مثال شما می توانید برای فراخوانی آی سی میکرو ATmega16 در قسمت جستجو نام mega16 را وارد کنید. همانطور که مشاهده می کنید در سمت راست صفحه شمای فنی آن نشان داده شده است.



پس از فراخوانی تمامی قطعه ها، پنجره ی فوق را بسته و با کلیک بر روی هرکدام از قطعه های انتخاب شده می توانید آن را در صفحه جای گذاری کنید. هنگام که تمامی قطعات درجای خود قرار گرفتند با کلیک کردن بر روی پایه ها می توانید، آن پایه را به پایه ی دیگری متصل کنید. چون آی سی های میکرو در عمل باید پروگرام شوند می توانید با باز کردن صفحه ی جزئیات آی سی از قسمت Program File به آدرس مکان ذخیره ی برنامه رفته و فایل HEX (هِگز) آن را باز کنید در پایان برای اجرای مدار کلید Play را فعال کنید.

برای استفاده از منابع تغذیه و دستگاه های اندازه گیری می توانید از منوی آماده به کار در سمت چپ صفحه استفاده کنید.



تا زمانی که برنامه در حال اجراست شما نمی توانید کاری در مدار انجام دهید.