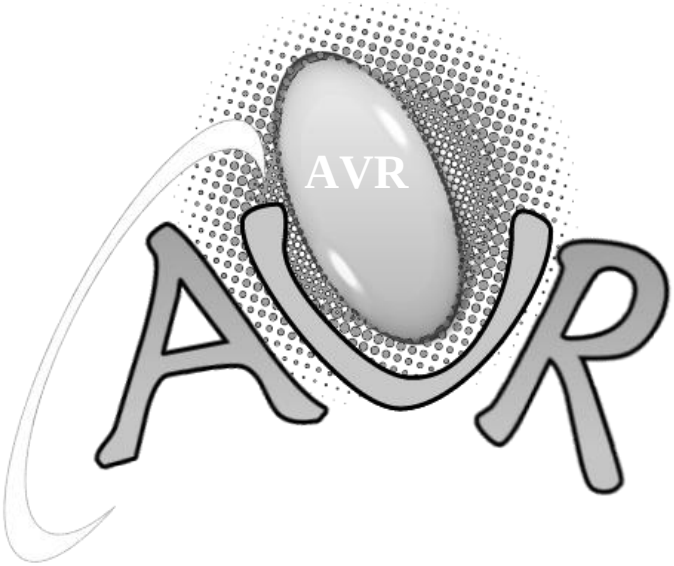
۵۳ پروژه کاربردی با میکروکنترلرهای



مولف: دکتر حامد سقایی


بسم الله الرحمن الرحیم

نهال را باران باید
تا بشوید غبار نشسته بر برگهایش
و سیرابش کند از آب حیات
و آفتاب باید
تا بتاباند
نیرو را
و محکم کند
شاخه‌های تازه روییده را

بر نام مادر
بوسه‌ای باید زد
دست‌هایی که
می‌شویند غبار خستگی روزگار را
و سیراب می‌کنند روح تشنه را
بر نام پدر
بوسه‌ای باید زد
دست هایی را
که می‌تابانند
نیرو را
و محکم می‌کنند
استواری پایه‌های زیستن را

مقدمه:

امروزه استفاده از تراشه‌های مختلف میکروکنترلر در ساخت و کنترل پروژه‌های مختلف آزمایشگاهی و صنعتی به عنوان ابزاری قدرتمند در خدمت طراحان قرار گرفته است و با ظهور این تراشه‌های منطقی برنامه‌پذیر، مدارات دیجیتال پیچیده، جای خود را به این تراشه‌ها داده‌اند.

اگرچه قابلیت‌ها و سرعت اجرای برنامه‌های نوشته شده به زبان اسمبلی غیر قابل انکار است ولی استفاده از زبان C در برنامه‌نویسی میکروکنترلرها به دلیل نزدیکی زیاد به سخت‌افزار (سطح میانی بودن زبان)، ساختار یافتگی و امکان استفاده از توابع نوشته شده آن در سایر پروژه‌ها مورد توجه بسیاری از کاربران قرار گرفته است. زبان دیگری که جذابیت زیادی را در برنامه‌نویسی میکروکنترلرهای AVR ایجاد کرده است، Basic می‌باشد. این زبان ساده‌ترین زبان برنامه‌نویسی میکروکنترلرهای AVR است.

در این کتاب سعی شده، قابلیت‌های میکروکنترلرهای سری Mega در قالب مثال‌های متنوع و کاربردی تشریح شوند. همچنین فرض شده است که خوانندگان این کتاب از آشنایی مقدماتی با میکروکنترلرهای AVR برخوردار هستند. مثال‌های کاربردی مطرح شده در این کتاب غالباً به گونه‌ای است که در کمتر کتابی یافت می‌شوند و کاربردهای این میکروکنترلرها را در مباحث مختلف الکترونیک، مخابرات، الکترونیک قدرت، رباتیک و نظایر آن نشان می‌دهند.

در هر فصل سعی شده است ضمن معرفی رجیسترهای مربوط به هر یک از قسمت‌های میکروکنترلر، تنظیمات مورد نیاز آن در محیط Wizard مربوط به نرم‌افزار CodeVision توضیح داده شوند و سپس مثال‌های متنوعی از کاربردهای آن بخش به زبان C ذکر گردند و نکات مهم آن برنامه‌ها در متن کتاب تشریح شوند.

مثال‌های مطرح شده در آن برای انواع میکروکنترلرهای سری Mega صادق هستند ولی به دلیل تنوع محصولات، قابلیت‌های موجود در تراشه ATmega16 به عنوان مبنای اصلی مطالب در این کتاب در نظر گرفته شده است. هر چند که در برخی از مثال‌ها از تراشه‌های دیگری استفاده شده است. به دلیل تنوع زیاد کامپایلرها در این کتاب، بیشترین تاکید بر کامپایلرهای زبان C و مخصوصاً CodeVision است. هر چند در فصل آخر کتاب، کامپایلر زبان Basic یعنی Bascom و کامپایلر قدرتمند دیگر زبان C یعنی WinAVR آموزش داده و پروژه‌های کاربردی مرتبط با آن‌ها ارائه می‌شوند.

به لطف ایزد منان، این مجموعه که حاصل تجربیات و مطالعات چندین ساله مولف بر روی سیستم‌های دیجیتال برنامه‌پذیر است، تکمیل شده است. لازم به ذکر است که: تمانی برنامه‌های نوشته شده در کتاب، در CD همراه کتاب نیز قرار داده شده و به طور کامل در آزمایشگاه ریزپردازنده آزمایش و بررسی شده‌اند. در CD همراه کتاب، علاوه بر برنامه‌های کتاب، تمامی فایل‌های مرتبط و فایل‌های اجرایی توسط نرم‌افزار شبیه‌ساز Proteus نیز وجود دارند تا خواننده کتاب، تنها به تحلیل، بررسی و اجرای آن‌ها بپردازد. همچنین متناسب با نیاز خوانندگان کتاب: آموزش نرم‌افزارهای مرتبط با کتاب نیز در CD قرار داده شده تا خواننده در میان دنیای تراشه‌های برنامه‌پذیر احساس تنهایی نکند. شایان ذکر است که پروژه‌های این کتاب به سادگی قابل ترکیب و استفاده در پروژه‌های بزرگ‌تر می‌باشند. امید است به لطف خداوند یکتا، این کتاب، راه را برای یادگیری، طراحی و پیاده‌سازی رهیافت‌های نوین علمی و عملی برای شما هموار سازد.

در پایان بر خود لازم می‌دانم، از اساتید بزرگواری که در دانشگاه صنعتی امیرکبیر (پلی‌تکنیک تهران)، از آنان بسیار آموختم، تقدیر به عمل آورم و این کتاب را نتیجه زحمات بسیار آن عزیزان بدانیم.

با توجه به این مهم که این کتابی عاری از اشکال نیست، در صورتی که اشکال یا اشکالاتی را در کتاب مشاهده فرمودید، ما را بخشیده و نظرات اصلاحی خود را به آدرس الکترونیکی h.saghaei@gmail.com یا از طریق وب سایت زیر برای ما ارسال نمائید تا در چاپ‌های بعدی اصلاح شوند. پیشاپیش از شما سپاسگزاریم.

فهرست

فصل اول میکروکنترلر و واسطهای جانبی

در این فصل ابتدا به اختصار بـه معرفـی میکروکنترلـر AVR مـورد اسـتفاده در ایـن کتاب پرداخته می‌شود. سپس انـواع روش‌هـای برنامـه‌ریزی میکروکنترلرهـای AVR بیان می‌شوند. همچنین روش نصب و کـار بـا نرم‌افـزار CodeVision آمـوزش داده می‌شود و در پایان، به منظور آشنایی مقدماتی خوانندگان کتاب، دستورات لازم زبـان برنامه‌نویسی C برای کار با میکروکنترلرها به اختصار توضیح داده خواهند شد.

۱-۱- معرفی میکروکنترلرهای AVR

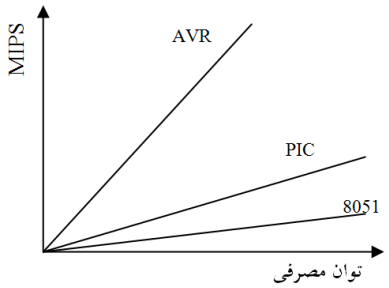
۱-۱-۱- مقدمه

«زبان‌های سطح بالا»[1] به سرعت در حال تبدیل شدن به زبان برنامه‌نویسی استاندارد برای میکروکنترلرها (MCU) هستند. زبان برنامه‌نویسی Basic و C بیشترین استفاده را در برنامه‌نویسی میکروکنترلرها دارند ولی در اکثر کاربردها، کدهای بیشتری را نسبت به زبان برنامه‌نویسی اسمبلی تولید می‌کنند. شرکت ATMEL با ایجاد تحولی در معماری ساخت تراشه‌های برنامه‌پذیر، از معماری RISC[2] برای ساخت میکروکنترلرهای AVR استفاده می‌کند که این معماری در مقایسه با معماری CISC دارای دستورات با اندازه (سایز) ثابت، رجیسترهای (ثبات‌های) همه منظوره بیشتر و دستورات ساده‌تر و قابل اجرا تنها در یک یا دو پالس ساعت هستند، می‌باشد که در نهایت منجر به افزایش سرعت اجرای دستورات توسط میکروکنترلر می‌شود. سرعت اجرای دستورات در میکروکنترلرهای AVR در مقایسه با میکروکنترلرهای ۸۰۵۱ و PIC به ترتیب ۱۲ و ٤ برابر می‌باشند.

تکنولوژی حافظه کم مصرف غیر فرار برای برنامه‌ریزی AVR مورد استفاده قرار گرفته است. در نتیجه حافظه‌های FLASH و EEPROM در داخل مدار قابل

[1] High Level Languages (HLL).
[2] Reduced Instruction Set Computer (RISC).

برنامه‌ریزی هستند. میکروکنترلرهای اولیه AVR دارای ۱ ، ۲ و ۸ کیلوبایت حافظه FLASH و به صورت کلمه‌های ۱٦ بیتی سازماندهی شده بودند. AVR ها به عنوان میکروکنترلرهای RISC با دستورات فراوان طراحی شده‌اند که باعث می‌شود حجم کد تولید شده کم و سرعت بالاتری حاصل شود. با انجام دستورات تک سیکلی، اسیلاتور با کلاک داخلی سیستم یکی می‌شود. هیچ تقسیم‌کننده‌ای در داخل AVR قرار ندارد که منجر به اختلاف فاز کلاک سیستم با سرعت اجرای دستورات شود. بیشتر میکروکنترلرها، فرکانس کلاک اسیلاتور سیستم را بر ٤ یا ۱۲ تقسیم می‌کنند که این امر منجر به کاهش سرعت اجرای دستورات می‌شود. بنابراین AVR ها ٤ تا ۱۲ بار سریعتر و توان مصرفی آن‌ها نیز ٤ تا ۱۲ بار نسبت به میکروکنترلرهای کنونی کمتر است. زیرا در تکنولوژی CMOS استفاده شده در میکروکنترلر های AVR، مصرف توان سطح منطقی متناسب با فرکانس است. شکل (۱-۱)، افزایش MIPS[1] را به علت انجام عملیات تک سیکلی AVR در مقایسه با PIC و ۸۰۵۱ نشان می‌دهد.



شکل (۱-۱): نمودار توان مصرفی برحسب اجرای میلیون دستور بر ثانیه برای میکروکنترلرهای AVR، PIC و ۸۰۵۱

۱-۱-۲- طراحی برای زبان‌های Basic و C:

زبآن‌های Basic و C بیشترین استفاده را در دنیای امروز به عنوان زبآن‌های HLL دارند. پیش از طراحی چنین معماری، بیشتر میکروکنترلرها برای زبان اسمبلی طراحی شده و کمتر از زبآن‌های HLL حمایت می‌کردند. هدف شرکت ATMEL طراحی معماری بود که هم برای زبان اسمبلی و هم برای زبآن‌های HLL مفید باشد. به عنوان مثال در زبآن‌های Basic و C می‌توان یک متغیر محلی به جای متغیر سراسری در داخل زیربرنامه تعریف کرد. بنابراین تنها در زمان اجرای زیربرنامه

[1] Million Instruction Per Seconds (MIPS).

مکانی از حافظه SRAM برای متغیر اشغال می‌شود. در صورتی که اگر متغیر به عنوان متغیر سراسری تعریف گردد، در تمام زمان اجرای برنامه مکانی از حافظه SRAM را اشغال کرده است. برای دسترسی سریع‌تر به متغیرهای محلی و کاهش کد، نیاز به افزایش رجیسترهای همه منظوره است. AVR ها دارای ۳۲ رجیستر همه منظوره هستند که مستقیما به «واحد محاسبه و منطق»[1] متصل شده‌اند و تنها در یک پالس ساعت به این واحد دسترسی پیدا می‌کنند. سه جفت از این رجیسترها می‌توانند به عنوان رجیسترهای ۱٦ بیتی استفاده شوند. نتیجه تمام موارد بیان شده، این است که میکروکنترلرهای AVR با سرعت بالا و سازماندهی RISC هستند.

میکروکنترلرهای AVR به سه خانواده اصلی AT90S AVR، Tiny AVR و Mega AVR تقسیم‌بندی می‌شوند. البته خانواده XMega نیز توسط شرکت سازنده، به صنعت معرفی شده است. در این کتاب برای پیاده‌سازی بسیاری از پروژه‌ها از میکروکنترلر ATmega16 استفاده شده است و همانطور که از نامش مشخص است از خانواده Mega AVR می‌باشد. در ادامه به خلاصه‌ای از خصوصیات و پیکربندی این میکروکنترلر می‌پردازیم.

۱-۱-۳- خصوصیات ATmega16 و ATmega16L

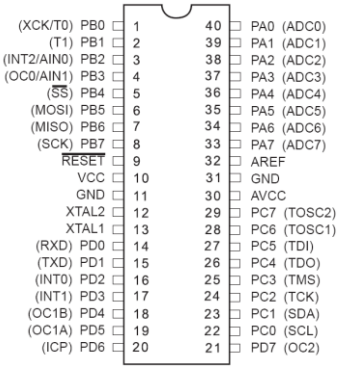
- با استفاده از معماری RISC طراحی و ساخته شده‌اند.
  - کارایی بالا و توان مصرفی کم.
  - دارای ۱۳۱ دستورالعمل، با کارایی بالا که بیشتر دستورات، تنها در یک سیکل ساعت اجرا می‌شوند.
  - دارای ۳۲ رجیستر همه‌منظوره هشت بیتی.
  - دارای بیشینه سرعت تا 16 MIPS در فرکانس 16 MHz.
- حافظه برنامه و داده غیرفرار
  - ۱٦ کیلوبایت حافظه FLASH داخلی قابل برنامه‌ریزی.
    پایداری حافظه FLASH با قابلیت ۱۰۰۰۰ بار نوشتن و پاک کردن.
  - ۱۰۲٤ بایت حافظه داخلی SRAM .
  - ۵۱۲ بایت حافظه EEPROM داخلی قابل برنامه‌ریزی.
    پایداری حافظه EEPROM با قابلیت ۱۰۰۰۰۰ بار نوشتن و پاک کردن.
  - قفل برنامه FLASH و EEPROM .
- خصوصیات جانبی
  - دو تایمر/کانتر ۸ بیتی با مقسم مجزا.

[1] Arithmetic Logic Unit (ALU).

- یک تایمر/کانتر ۱۶ بیتی با مقسم مجزا و دارای حالت‌های تسخیر و مقایسه.
- ٤ کانال مدولاسیون عرض پالس (PWM).
- ۸ کانال مبدل آنالوگ به دیجیتال ۱۰ بیتی.
- یک مقایسه کننده آنالوگ داخلی.
- Watchdog قابل برنامه‌ریزی با اسیلاتور داخلی.
- قابلیت ارتباط با پروتکل «سریال دو سیمه»[1] و I^2C.
- قابلیت ارتباط سریال SPI[2] به صورت Master یا Slave.
- USART سریال قابل برنامه‌ریزی.

- خصوصیات ویژه میکروکنترلر
  - دارای اسیلاتور RC داخلی کالیبره شده.
  - منابع وقفه[3] داخلی و خارجی.
  - توان مصرفی پایین و سرعت بالا توسط تکنولوژی CMOS .
  - ولتاژ کاری 2.7 تا 5.5 ولت برای Atmega16L و 4.5 تا 5.5 ولت برای Atmega16 .
  - فرکانس‌های کاری 0 تا 8MHz برای Atmega16L و 0 تا 16MHz برای Atmega16 .
  - ۳۲ خط ورودی خروجی قابل برنامه ریزی.

شکل (۱-۲) شمای پایه‌های میکروکنترلر Atmega16 را نشان می‌دهد. همانطور که در شکل (۱-۲) مشاهده می‌شود ATmega16 دارای چهار پورت A، B، C و D می‌باشد که افزون بر این‌که به عنوان ورودی خروجی مورد استفاده قرار می‌گیرند، کاربردهای جانبی دیگری نیز دارند. که در فصل‌های بعد به آن‌ها خواهیم پرداخت.



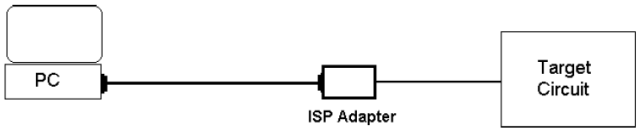
[1] TWO-WIRE
[2] Serial Peripheral Interface (SPI).
[3] Interrupt

شکل (۱-۲): شمای پایه‌های میکروکنترلر ATmega16

۱-۲- برنامه‌ریزی میکروکنترلرهای AVR

به طور کلی، چهار روش برای برنامه‌ریزی میکروکنترلرهای AVR وجود دارد که به ترتیب: برنامه‌ریزی موازی، برنامه‌ریزی توسط پروتکل JTAG، خود برنامه‌ریز[۱] و ISP[۲] می‌باشند.

برای برنامه‌ریزی میکروکنترلر AVR، از روش ISP استفاده می‌کنیم. در این روش: بدون برداشتن تراشه میکروکنترلر AVR از روی برد مربوطه، اقدام به پروگرم نمودن آن می‌نمائیم. انتقال کدهای برنامه از کامپیوتر به میکروکنترولر، مطابق شکل (۱-۳) به روش سریال می‌باشد. در روش ISP، با استفاده از ارتباط SPI (که در فصل‌های بعد توضیح داده می‌شود) می‌توانیم علاوه بر برنامه‌ریزی میکروکنترولر (Programming)، صحت برنامه‌ریزی را نیز تشخیص دهیم (Verify). در این روش با استفاده از سه خط سیگنال، برنامه‌ریزی میکروکنترلر انجام می‌شود (یک خط ورودی، یک خط خروجی و یک خط پالس ساعت (Clock). البته: پایه Reset میکروکنترلر می‌بایست در زمان برنامه‌ریزی پایین (صفر منطقی) نگه داشته شود، یعنی به صفر ولت متصل شود).



شکل (۱-۳): ساختار برنامه‌ریزی میکروکنترولر.

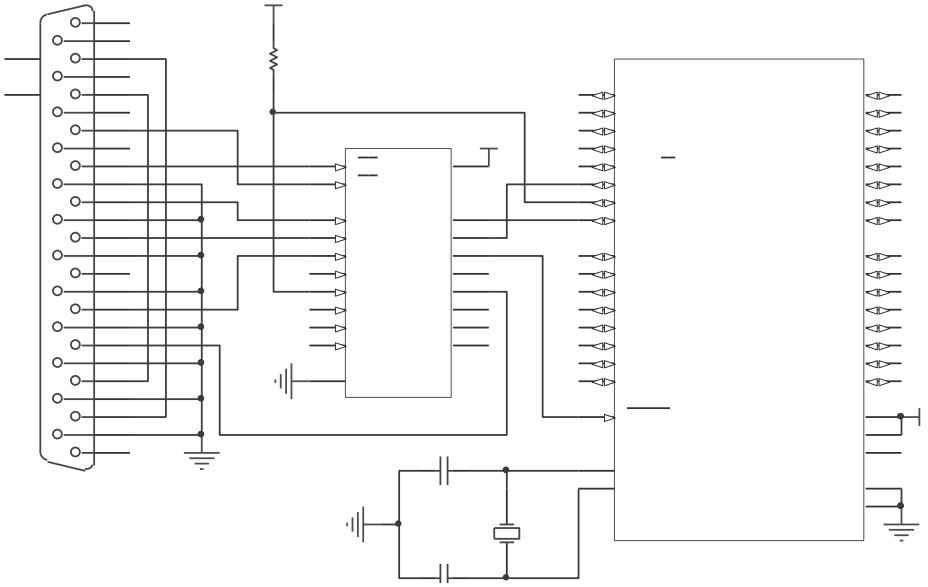
ساخت پروگرمر ISP با استفاده از کانکتور DB25 بسیار ساده است. اتصال مستقیم میکروکنترلر به کامپیوتر، ممکن است باعث ایجاد مشکلات ناشی از جریان کشی توسط میکروکنترلر و کامپیوتر شود که در نهایت منجر به سوختن میکروکنترلر می‌شود. بنابراین، با استفاده از یک تراشه راه‌انداز[۳] به منظور تامین جریان لازم در هنگام برنامه‌ریزی میکروکنترلر، مشکل مذکور برطرف خواهد شد. همچنین به این دلیل که پورت[۴] موازی در مقابل اتصال کوتاه و یا اضافه بار چندان مقاوم نیست و ممکن است با قطع و وصل تغذیه‌ی بورد، آسیب ببیند. بنابراین، استفاده از یک بافر

---

[1] Self-Programming
[2] In System Programming
[3] Driver
[4] Port

جریان ضروری است. در این روش، نمی‌توان به طور قاطع از برنامه‌ریزی کاملاً مطمئن میکرو صحبت کرد.

مدار واسط STK200/300 به منظور برنامه‌ریزی میکروکنترولر به روش ISP استفاده شده است که مطابق شکل (۱-٤) می‌باشد. تراشه 74ALS244 به عنوان بافر جریان، جهت حفاظت از پورت‌های میکروکنترلر و کامپیوتر استفاده شده است. شما می‌توانید مطابق نقشه شکل (۱-٤) مدار پروگرمر را بسازید و یا آن را از فروشگاه‌های الکترونیکی تهیه نمائید. شایان ذکر است که صحت عملکرد صحیح مدار زیر تائید می‌شود.



شکل (۱-٤): پروگرامر STK200/300

۱-۳- آشنایی با روش نصب و کار با نرم‌افزار CodeVisionAVR

۱-۳-۱- مقدمه

برای کار با میکروکنترلرهای AVR باید برنامه‌ای به یکی از زبان‌های Assembly، C یا Basic در محیط نرم افزار مربوط به آن نوشت. سپس آن را کامپایل نمود. **کامپایل نمودن برنامه:** عملی است که در آن، برنامه از زبان نوشتاری به زبان صفر و یک که توسط میکروکنترلر قابل فهم باشد، تبدیل می‌شود. در صورتی که برنامه هیچ

خطایی، شامل: خطای املایی، ساختاری و نظایر آن را نداشته باشد به درستی کامپایل شده و یک فایل به زبان صفر و یک (زبان ماشین) توسط کامپایلر تولید می‌شود. پسوند فایل‌هایی که حاوی برنامه به زبان ماشین هستند، HEX می‌باشد. اکنون برای انتقال فایل HEX ایجاد شده به درون آی‌سی، نیازمند یک دستگاه جانبی یا واسط سخت افزاری هستیم که کامپیوتر را به تراشه میکروکنترلر متصل کند و فایل HEX مربوطه را از کامپیوتر بر روی میکروکنترلر بارگذاری نماید. این واسط سخت افزاری، اصطلاحاً پروگرمر نامیده می‌شود. پس از برنامه‌ریزی کردن (پروگرم کردن)، میکروکنترلر را از پروگرمر جدا کرده و در مدار مورد نظر قرار داده (و یا اگر پروگرمر ساخته شده مطابق شکل (۱-٤) باشد بدون جدا نمون میکروکنترلر از مدار به برنامه‌ریزی آن اقدام می‌کنیم). پس از آن، عملکرد سخت افزاری آن‌را بررسی می‌کنیم.

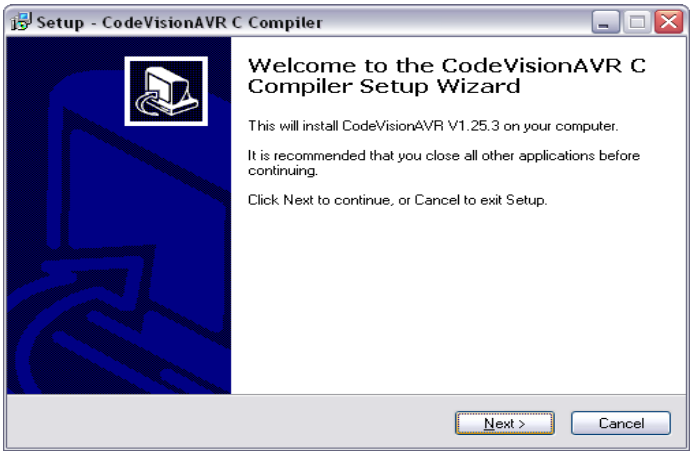
در این قسمت، نرم افزار CodeVisionAVR که یکی از کامپایلرهای قوی برای برنامه‌نویسی به زبان C می‌باشد، معرفی می‌شود. افزون بر این، روش نصب و قسمت‌های مختلف آن نیز آموزش داده می‌شود.

۱-۳-۲- آشنایی با محیط CodeVisionAVR

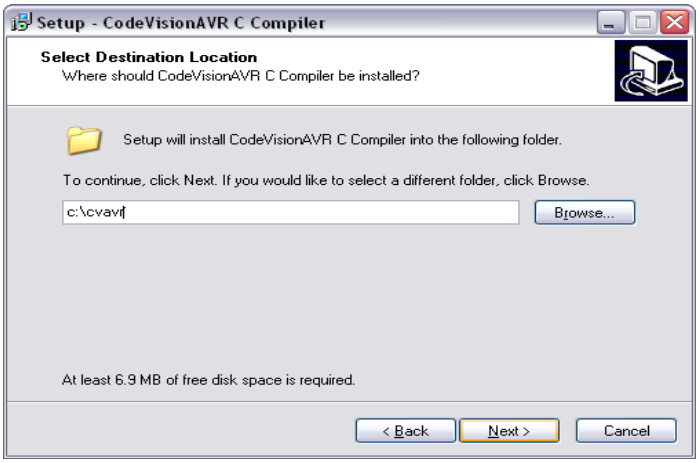
نرم افزار CodeVisionAVR دارای محیطی برای برنامه‌نویسی به زبان C است. که در این محیط، کاربر با تسلط نسبی بر زبان C قادر به نوشتن برنامه‌های بسیار کاربردی می‌شود. یکی از دلایل انتخاب این نرم‌افزار، قابلیت Wizard یا محیط راهنمای گام به گام است. محیط راهنمای گام به گام که به آن به اختصار ویزارد گفته می‌شود. این قابلیت در مقدار دهی اولیه رجیسترهای مختلف میکروکنترلر، همچنین فراخوانی برخی کتابخانه‌های موجود، کمک بسیار زیادی به کاربران می‌کند. بنابراین، به کاربران مبتدی، برنامه‌ریزی میکروکنترلرهای AVR با استفاده از این محیط توصیه می‌شود. این نرم‌افزار دارای یک کامپایلر بوده که توسط آن کدهای برنامه با پسوند Hex جهت برنامه‌ریزی میکروکنترلر تولید می‌شوند. توسط این نرم افزار و یک پروگرمر از نوع ISP می‌توان کلیه‌ی میکروکنترلرهای AVR را برنامه‌ریزی نمود. نرم‌افزار CodevisionAVR علاوه بر حمایت از کتابخانه‌های استاندارد زبان C، دارای کتابخانه‌های دقیقی برای کار با LCD های کارکتری، تولید وقفه، تنظیمات توان مصرفی و نظایر آن می‌باشد.

۱-۳-۳- روش نصب نرم‌افزار CodevisionAVR

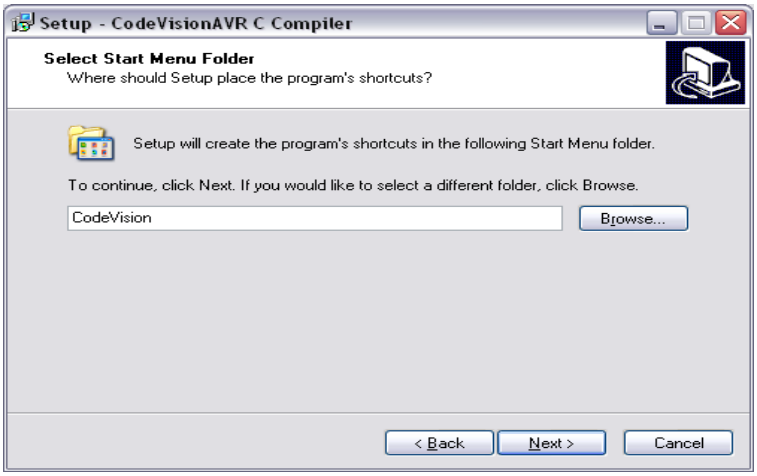
ابتدا با مراجعه به CD (که همراه با کتاب عرضه می‌شود) فایل مربوط به نرم افزار CodeVisionAVR را باز کنید و با اجرای فایل setup.exe مراحل نصب را مطابق شکل‌های زیر تا پایان ادامه دهید.



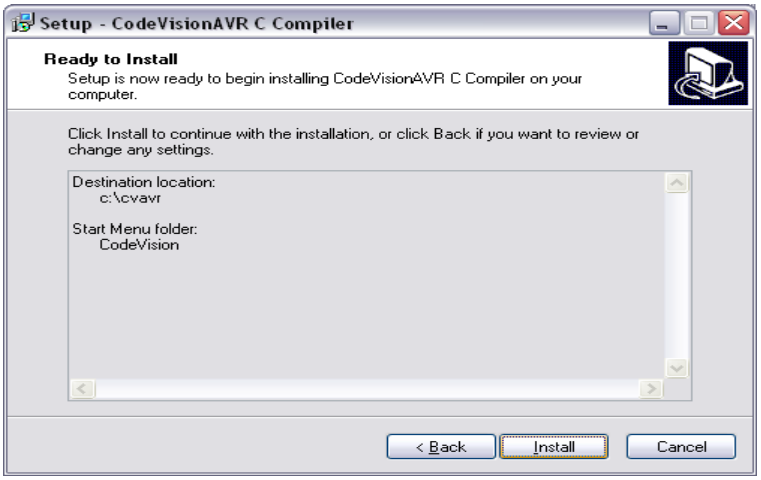
۱- بر روی گزینه Next کلیک نمائید.



۲- با انتخاب مسیر مناسب جهت نصب نرم‌افزار بر روی گزینه Next کلیک نمائید.



۳- بر روی گزینه Next کلیک نمائید.

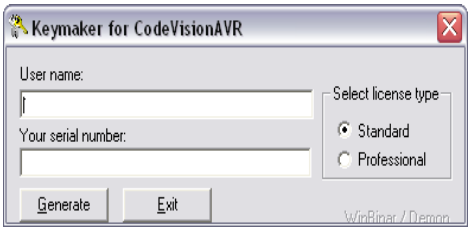


۴- بر روی گزینه Install کلیک نمائید. در صورتی که نرم‌افزار CodeVision نسخه V2.03.4 را نصب می‌کنید، این نسخه دارای قفل نرم‌افزاری نبوده و در این مرحله عملیات نصب پایان می‌پذیرد. در غیر این صورت مراحل زیر را دنبال کنید.

۵- پس از نصب برنامه، برای اجرای صحیح آن ضروری است که قفل برنامه باز شود. برای این کار، ابتدا برنامه را اجرا نمائید.

از پنجره ظاهر شده شماره سریال را یادداشت نمائید و در مرحله بعد وارد نمائید.

۶- فایل License Generator.exe را از مسیر دایرکتوری اصلی برنامه، اجرا نمائید. اکنون با وارد کردن یک نام دلخواه در قسمت User name، و در قسمت Serial number، شماره سریال مرحله‌ی ۵ را وارد کرده و دکمه‌ی Generate را فشار دهید و فایل تولید شده با پسوند dat. را در مسیر دلخواه ذخیره نمائید.



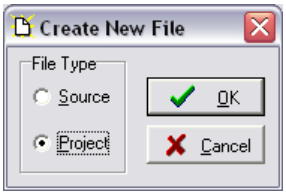
۷- اکنون با استفاده از پنجره ظاهر شده در مرحله‌ی ۵، بر روی Import کلیک کرده و آدرس فایل تولید شده در مرحله‌ی ۶ را وارد کنید. اکنون، قفل برنامه باز شده و برنامه قابل اجرا و استفاده است.

۱-۳-٤- ایجاد یک پروژه جدید

برای آشنایی با محیط CodeVisionAVR با ایجاد یک پروژه شروع می‌کنیم و مراحل زیر را مطابق شکل‌ها ادامه می‌دهیم:

۱- برنامه CodeVisionAVR را اجرا می کنیم.

۲- در صفحه‌ی باز شده (صفحه اصلی) پس از انتخاب گزینه File، گزینه New را انتخاب کرده تا پنجره‌ای مطابق شکل (۱-۵)، ظاهر شود.

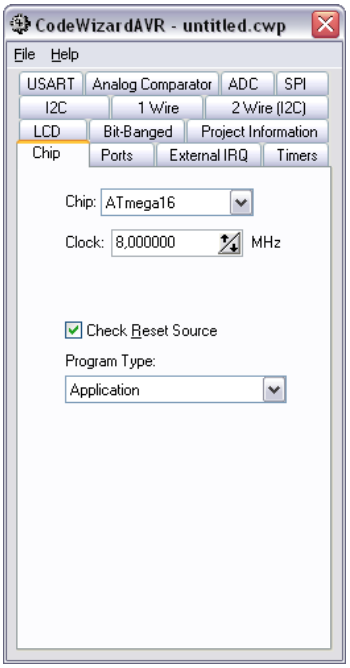


شکل (۱-۵): ایجاد پروژه جدید

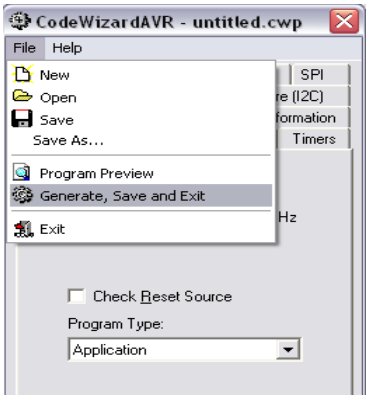
۳- از پنجره ظاهر شده در شکل (۱-۵) گزینه Project را که مربوط به انتخاب پروژه است انتخاب نموده و بر روی OK کلیک نمائید. اکنون، پنجره‌ی دیگری باز می‌شود و از شما سوال می‌شود که: آیا می‌خواهید پروژه جدید را توسط محیط ویزارد ایجاد کنید؟ با فشردن کلید Yes این عمل انتخاب می‌شود و با فشردن کلید No جهت ایجاد پروژه از ویزارد استفاده نمی‌شود.

شکل (۱-٦): انتخاب CodeWizardAVR

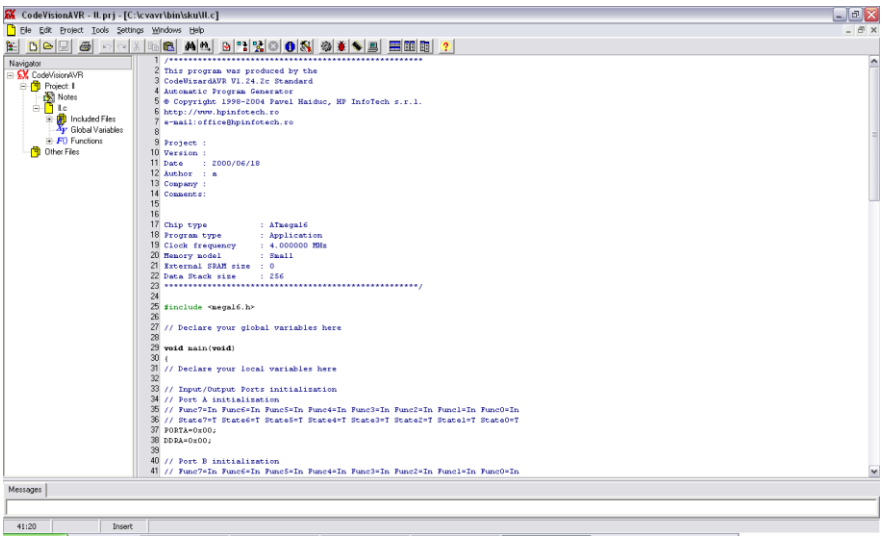
۱-۳- با کلیک بر روی گزینه Yes، مطابق شکل (۱-٦) محیط ویزارد مطابق شکل (۱-۷) باز می‌شود و توسط این محیط می‌توان با استفاده از سربرگ‌های موجود، با انتخاب میکروکنترلر مورد نظر، رجیسترهای آن را به‌صورت اولیه مقداردهی نمود. پس از اتمام مقدار دهی اولیه، با انتخاب گزینه File از منوی مربوطه، مطابق شکل (۱-۸)، بر روی گزینه Generate, Save and Exit کلیک کرده و با انتخاب یک نام برای آن، پروژه مورد نظر را ایجاد نمائید تا شکل (۱-۹) ظاهر شود. اکنون برنامه شما به صورت اولیه مقدار دهی شده است.



شکل (۱-۷): محیط CodeWizardAVR برای مقداردهی اولیه میکروکنترلرهای AVR.

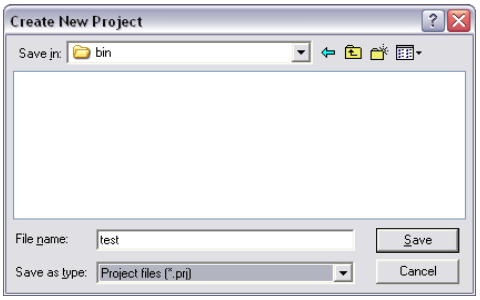


شکل (۱-۸): ذخیره‌سازی و تولید کد.



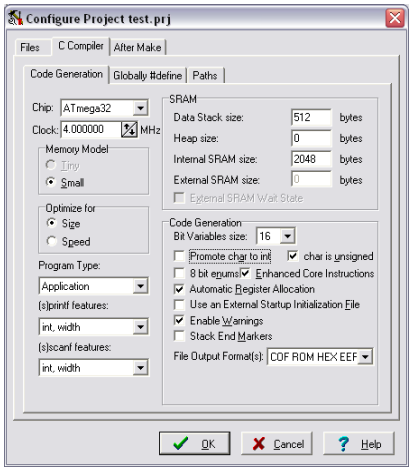
شکل (۱-۹): کدهای ایجاد شده توسط ویزارد در محیط CodevisionAVR

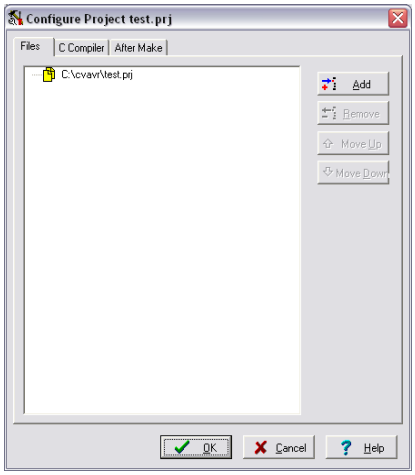
۳-۲- با کلیک بر روی گزینه No در شکل (۱-۶) پنجره زیر جهت ذخیره پروژه ظاهر می‌شود. پس از انتخاب یک نام برای پروژه مورد نظر، بر روی گزینه Save کلیک نمائید.



شکل (۱-۱۰): انتخاب یک نام جهت ذخیره پروژه

۳-۲-۱- پس از ذخیره پروژه، شکل (۱-۱۱) ظاهر می‌شود که دارای سه سربرگ File، C Compiler و After Make می‌باشد. با انتخاب سربرگ C Compiler میکروکنترلر و فرکانس کریستال آن را انتخاب نمائید. به عنوان مثال، در شکل (۱-۱۲) میکروکنترلر ATmega32 و کریستال ٤ مگاهرتز انتخاب شده است (سایر سربرگ‌ها در قسمت‌های مربوطه به تفصیل توضیح داده می‌شوند). حال، بر روی گزینه OK کلیک نمائید تا پروژه ایجاد شود.





شکل (۱-۱۱): پیکربندی پروژه **شکل (۱-۱۲): پیکربندی میکروکنترلر**

۳-۲-۲- در صفحه‌ی باز شده (صفحه اصلی) پس از انتخاب گزینه File، گزینه New را انتخاب کرده تا پنجره‌ای ظاهر شود. از پنجره ظاهر شده، گزینه Source را انتخاب و سپس فایل مورد نظر را با نام دلخواه ذخیره نمائید. دقت شود که پسوند

این فایل c. است. یعنی محیطی جهت برنامه‌نویسی به زبان C ایجاد شده است. اکنون، از منوی بالای صفحه، بر روی گزینه Project و سپس بر روی Configure کلیک نمائید تا مجدداً شکل (۱-۱۱) ظاهر شود. سپس از سربرگ Files بر روی گزینه Add کلیک نمائید و فایل ایجاد شده با پسوند c. را به پروژه اضافه نمائید و بر روی گزینه OK کلیک نمائید. اکنون پروژه مورد نظر، ایجاد شده است.

۱-٤- آشنایی با زبان C

۱-٤-۱- ساختار برنامه‌نویسی

به منظور برنامه‌نویسی به زبان C، آشنایی اولیه با دستورات آن الزامی است. برای این منظور: با معرفی مختصر ساختارها و دستورات مرتبط، مطابق زیر با یک برنامه ساده شروع می‌کنیم.

```
#include <stdio.h>
int main() {
        printf("Hello World\n");
        return 0;
    }
```

شرح برنامه:

<stdio.h>#include : فایلی به اسم stdio.h را ضمیمه می‌کند که این فایل به ما اجازه استفاده از توابع خاصی را می‌دهد. stdio کوتاه شده‌ی عبارت Standard Input/Output است. این فایل شامل توابع ورودی: مانند خواندن از صفحه کلید و توابع خروجی: مانند نمایش دادن بر روی صفحه نمایش است.

int main() :

int : عبارتی است که یک مقدار را بر می‌گرداند (return) و در ادامه بیشتر به توضیح آن می‌پردازیم.

main : نام نقطه‌ای است که برنامه از آن نقطه شروع می‌شود. پرانتزها در جلوی عبارت main به این معنی است که این تابع آرگومان ورودی ندارد.

```
int main() {
        دستورها
    }
```

{ } (آکلادها) برای این است که تمام نوشته‌ها را در یک گروه خاص قرار دهد. در مثال بالا، آکلاد مشخص می‌کند که نوشته‌ها متعلق به تابع main است. شایان ذکر است که آکلادها در زبان C کاربرد زیادی دارند.

printf ("Hello World \n");

تابع printf : یک متن را در صفحه نمایش می‌دهد. اطلاعاتی که باید این تابع نمایش دهد بین دو پرانتز قرار می‌گیرد. دقت شود که: کلمه‌ها بین دو double contention (") قرار گیرند، زیرا آن‌ها در واقع یک رشته هستند. هر یک از حروف یک کاراکتر می‌باشد و مجموعه‌ی آن‌ها یک گروه با نام رشته (string) را تشکیل می‌دهند. رشته‌ها همیشه باید بین دو " قرار می‌گیرند.

\n : به کامپایلر دستور می‌دهد به خط جدید برود، وقتی شما در متن خود enter بزنید به خط جدید نمی‌رود برای همین ما مجبور هستیم از این دستور استفاده کنیم.

باید بعد از هر دستوری یک semicolon (;) قرار دهید برای این که نشان دهد آن دستور تمام شده است.

جدول (۱-۱): دستورات برای printf

\a	Audible signal
\b	Backspace
\t	رفتن به یک tab جلوتر
\n	رفتن به یک خط جدید
\v	Vertical tab
\f	پاک کردن صفحه / رفتن به صفحه جدید
\r	Carriage return

return 0 : مقدار آرگومان برگشتی توسط دستور return مشخص می‌شود و return 0 به معنی آن است که تابع ما مقدار صفر را باز می‌گرداند.

پس از کامپایل نمودن برنامه، اگر کدهای شما اشتباه باشند، کامپایلر به شما می‌گوید که اشتباه در کدام خط رخ داده است. به منظور اجرای صحیح برنامه، اصلاح خطا شامل: خطای املایی و ساختاری الزامی است. پس از کامپایل صحیح، برنامه‌ی شما تبدیل به فایل اجرایی می‌شود. اکنون باید عبارت "Hello World" را در صفحه ملاحظه کنید.

استفاده از توضیحات (comments) : برای تشخیص بهتر هر یک از خطوط برنامه، استفاده از comment در جلوی خطوط پیشنهاد می‌شود. توضیحات را باید بعد از // یا بین /*.....*/ بنویسید. توضیحات توسط کامپایلر خوانده نمی‌شوند. توضیحات استفاده شده در اول برنامه عملکرد برنامه را نشان می‌دهند. شما همچنین می‌توانید: بین قسمت‌های مختلف برنامه از توضیحات استفاده کنید تا آن قسمت را توضیح دهید در ذیل، مثالی از قرار دادن توضیح قرار داده شده است:

```
/* Author: Hamed Saghaei
 Date: 2009/07/15
 Description:
 Writes the words "Hello World" on the screen */

#include <stdio.h>

int main()
 {
    printf("Hello World\n");      //prints "Hello World"
    return 0;
}
```

۱-٤-۲- متغیرها

متغیرها در زبان برنامه‌نویسی C، مکانی از حافظه هستند که نامی به آن‌ها تخصیص داده می‌شود و می‌توانند مقداری را در بر داشته باشند. از متغیرها برای ذخیره کردن مقادیر در حافظه استفاده می‌شود. دو نوع متغیر اصلی در زبان C وجود دارد که به ترتیب (عدد) Numeric و(حروف) Character می‌باشند.

- **متغیرهای عددی:** این نوع متغیرها می‌توانند عدد صحیح باشند و اعداد کسری یا اعشاری نمی‌توانند در این گروه قرار بگیرند.

- **متغیرهای کاراکتری:** در این نوع متغیرها: حروف و اعداد می‌توانند قرار بگیرند. البته اعداد در اینجا کمی متفاوت هستند.

- **عبارات عددی و رشته‌ای ثابت:** این نوع عبارات ثابت هستند و مقدار آن‌ها قابل تغییر نیست. گاهی برای جلوگیری از اشتباه از آن‌ها استفاده می‌شود.

تعریف متغیرها: برای تعریف متغیر در برنامه ابتدا نوع آن را باید مشخص کنیم در جدول (۱-۲) نوع متغیرها و محدوده‌ی آن‌ها مشخص شده است. البته متغیرها را به هر نحوی که مایل باشید می‌توانید نام گذاری کنید. ولی بهتر است کمتر از ۳۲ حرف داشته باشند و در ابتدای برنامه تعریف شوند.

جدول (۱-۲): انواع داده

نام	نوع	محدوده
bit	یک بیت	از ۰ و ۱
char	هشت بیت	از ۱۲۸- تا ۱۲۷
unsigned char	هشت بیت بدون علامت	از ۰ تا ۲۵۵
int	اعداد صحیح ۱۶ بیتی	از ۳۲۷۶۷- تا ۳۲۷۶۸
unsigned int	اعداد صحیح بدون علامت ۱۶ بیتی	از ۰ تا ۶۵۵۳۵
long int	اعداد صحیح ۳۲ بیتی	از ۲۱۷۴۸۳۶۴۸- تا ۲۱۷۴۸۳۶۴۷
unsigned long int	اعداد صحیح بدون علامت ۳۲ بیتی	از ۰ تا ۴۲۹۴۶۷۲۹۵
float	اعداد اعشاری	از مقادیر مثبت و منفی ۳۸-e× ۱,۱۷۵ تا مقادیر مثبت و منفی ۳,۴۰۲×e۳۸
double	اعداد اعشاری	از مقادیر مثبت و منفی ۳۰۸-e×۲,۲ تا مقادیر مثبت و منفی ۱,۸×e۳۰۸

```
int main()
{
    int a;
    char b;
    return 0;
}
```

در عبارت بالا: متغیر a از نوع int و متغیر b از نوع char تعریف شده‌اند. و مطابق زیر برنامه ذیل می‌توان در یک مکان، چندین متغیر را تعریف کرد.

```
int main()
{
    int a, b, c;
    return 0;
}
```

با توجه به این‌که تابع main، تابع اصلی برنامه است. بنابراین بازگشت مقادیر توسط این تابع صحیح نمی‌باشد. در ادامه به جای تعریف آن به روش مثال‌های قبل، به صورت زیر تعریف می‌شود. همچنین برای تعریف متغیرهای Constant (ثابت) تنها لازم است: عبارت const را قبل از نوع متغیر قرار داد.

```
void main()
{
    const float pi = 3.1415;
 }
```

- متغیرهای علامت‌دار و بدون علامت:

متغیرهای علامت‌دار (signed) می‌توانند دارای مقادیر مثبت یا منفی باشند ولی متغیرهای بدون علامت (unsigned)، تنها می‌توانند مقادیر مثبت و صفر را اختیار کنند. این عبارات، قبل از نوع متغیر تعریف می‌شوند و نبود آن دلیل بر علامت‌دار بودن (Signed) متغیر است.

```
void main()
{
    unsigned int a;
    signed int b;
}
```

- استفاده از متغیر در محاسبات:

برای دادن مقداری به متغیر از علامت تساوی (=) استفاده می‌شود.

```
void main()
{
    int a=4;    // a=4
    char b;     // b=0
    a = 3;      // a=3
    b = 'H';    // b='H'
 }
```

به منظور انجام محاسبات از عملگرهای محاسباتی زیر استفاده می‌شود.

جدول (۱-۳): عملگرهای محاسباتی

عملیات	عمگر
جمع	+
تفریق	-
ضرب	*
تقسیم	/
باقیمانده	%

برای انجام عملیات به متغیری نیاز داریم که حاصل عملیات در آن ذخیره شود.

```
void main()
{
    int a, b;
    a = 5;
    b = a + 3;    // b=8
    a = a % 3;    // a=2
}
```

در زبان برنامه‌نویسی C، می‌توان متغیری از keyboard را با استفاده از دستور scanf گرفت و توسط printf آن را چاپ کرد.

```
#include <stdio.h>
void main()
{
    int a;
    scanf("%d", &a);
    a = a * 2;
    printf("The answer is %d", a);
}
```

%d برای خواندن و چاپ کردن متغیرها از نوع int استفاده می‌شود و سایر متغیرها مطابق جدول زیر هستند.

جدول (۱-٤): خواندن و چاپ متغیرها

%d or %i	int
%c	char
%f	float
%lf	double
%s	string

۱-٤-۳- دستور If

در بیشتر مواقع لازم است که مقدار متغیری در برنامه کنترل شود. برای این منظور استفاده از حلقه If به حل مساله کمک می‌کند.

- ساختار کلی دستور If

```
if (شرط)
  {
    دستورات;
  }
else if (شرط)
  {
    دستورات;
   }
else
  {
    دستورات;
   }
```

```
#include <stdio.h>
 void main()
{
    int mark;
    char pass;
    scanf("%d", &mark);
    if (mark > 40)
        pass = 'y';
}
```

در برنامه بالا ابتدا متغیری از ورودی گرفته می‌شود و سپس شرط mark > ٤٠ امتحان می‌شود. اگر پاسخ مثبت بود، حرف Y در متغیر pass قرار داده می‌شود.

```
#include <stdio.h>
 void main()
{
    int mark;
    char pass;
    scanf("%d", &mark);
    if (mark > 40)
```

```
        pass = 'y';
    else
        pass = 'n';
}
```

در برنامه بالا ابتدا متغیری از ورودی گرفته می‌شود و سپس شرط mark > ٤٠ امتحان می‌شود. اگر پاسخ مثبت بود، حرف Y در متغیر pass قرار می‌گیرد، در غیر این صورت حرف n در pass قرار داده می‌شود.
در صورتی که در برنامه، اجرای چند دستور پس از امتحان شرط نیاز داشت، می‌باید مطابق زیر برنامه ذیل، دستورات را درون { } قرار دهیم.

```
#include <stdio.h>
void main()
{
    int mark;
    char pass;
    scanf("%d", &mark);
    if (mark > 40)
      {
        pass = 'y';
        printf("You passed");
      }
  else
     {
        pass = 'n';
        printf("You failed");
     }
}
```

همچنین می‌توان در هر دستور if چند شرط را امتحان نمود:

```
#include<stdio.h>
 void main()
 {
    int a, b;
    scanf("%d", &a);
    scanf("%d", &b);
    if (a > 0 && b > 0)
        printf("Both numbers are positive\n");
    if (a = 0 || b = 0)
        {
         printf("At least one of the numbers = 0\n");
```

```
        a++;
      }
  if (!(a > 0) && !(b > 0))
      printf("Both numbers are negative\n");
}
```

- عملگرهای منطقی:

جدول (۱-۵): عملگرهای منطقی

مساوی	==
نا مساوی	=!
بزرگتر از	>
بزرگتر یا مساوی	>=
کوچکتر از	<
کوچکتر یا مساوی	<=
و	&&
یا	||
نقیض	!

و: هر دو شرط درست باشند حاصل درست.
یا: یکی یا هر دو شرط درست باشند حاصل درست است.

۱-٤-٤- ساختار دستور switch

```
switch(variable)
{
    case var 1:
    دستور;
    break;
    case var 2:
    دستور;
    break;
    .
    .
    .
```

```
    default:
    دستور;
}
```

```
#include <stdio.h>
 void main()
{
    char fruit;
    printf("Which one is your favourite fruit:\n");
    printf("a) Apples\n");
    printf("b) Bananas\n");
    printf("c) Cherries\n");
    scanf("%c", &fruit);
    switch (fruit) {
    case 'a':
        printf("You like apples\n");
        break;
   case 'b':
        printf("You like bananas\n");
        break;
    case 'c':
        printf("You like cherries\n");
        break;
    default:
        printf("You entered an invalid choice\n");
    }
}
```

در برنامه بالا: fruit متغیر برنامه است که آن را با استفاده از case کنترل می‌کنیم اگر برابر a بود، متن You like apples در خروجی چاپ می‌شود سپس از دستور beak استفاده شده است که باعث خروج از حلقه می‌شود در نهایت اگر با هیچ‌یک از متغیرها برابر نبود با استفاده از default متن You entered an invalid choice چاپ می‌شود.

۱-٤-٥- دستورات حلقه

در برخی مواقع لازم است دستوری به صورت متوالی اجرا شود. برای این کار از دستورات حلقه استفاده می‌شود که به سه دسته تقسیم می‌شوند:

for, do while, while:

- **ساختار for**

```
for (عدد شروع ;شرط ;اضافه شدن متغیر)
{
دستورات ;
}
```

for اجازه می‌دهد از عددی تا عدد دیگر تکرار داشته باشیم. در برنامه زیر اعدد ۱ تا ۲٤ در خروجی چاپ می‌شوند.

```
 void main()
{
    int i;
    for (i = 1; i == 24; i++)
        printf("H\n");
    return 0;
}
```

در مثال بالا از i++ استفاده کردیم که برابر است با i=i+1 یعنی در هر بار اجرا، یک واحد به i اضافه می‌شود یا می‌توان از i-- استفاده کرد که هر بار اجرا، یک واحد از i کم می‌شود.

- **ساختار while**

```
while(condition)
{
   دستورات ;
}
```

تفاوت این حلقه با قبلی در این است که ما نمی‌دانیم قرار است چند بار حلقه اجرا شود در مثال زیر تعداد اجرا شدن حلقه از ورودی گرفته می‌شود در times ذخیره و در هر بار اجرا شدن حلقه یک واحد به i افزوده می‌شود تا زمانی که به تعداد times برسد از حلقه خارج شده و برنامه تمام می‌شود.

```
include <stdio.h>
void main()
{
    int i, times;
    scanf("%d", &times);
    i = 0;
    while (i <= times)
    {
```

```
        i++;
        printf("%d\n", i);
    }
}
```

- ساختار do while

ساختار do while مانند while است با این تفاوت که شرط آن در آخر امتحان می‌شود.

```
do {
    دستورات ;
   }
while(Condition)
```

```
#include <stdio.h>
 void main()
{
    int i, times;
    scanf("%d", &times);
    i = 0;
    do
    {
        i++;
        printf("%d\n", i);
    }
while (i <= times);
```

:Break and continue

Continue برای شروع Loop از ابتدا و برای خاتمه دادن از Break استفاده می‌شود.
در مثال زیر هرگز Hello چاپ نمی‌شود.

```
#include <stdio.h>
 int main()
{
    int i;
    while (i < 10)
    {
        i++;
        continue;
        printf("Hello\n");
    }
}
```

۱-٤-٦- اشاره‌گرها

اشاره‌گر[1] متغیرهایی هستند که درون خود آدرسی از حافظه را نگهداری می‌کنند و به متغیر درون آن آدرس، اشاره می‌کنند به این دلیل به آن‌ها اشاره‌گر می‌گویند. شما می‌توانید از اشاره‌گرها برای کارهای خاصی استفاده کنید. مثلاً برای گرفتن مقدار آدرسی که به آن اشاره می‌کند. اشاره‌گرها می‌توانند مشخص یا نامشخص باشند. اشاره‌گرهای مشخص به یک متغیر مشخص مثلاً int اشاره می‌کنند، و اشاره‌گرهای نامشخص می‌توانند به انواع اطلاعات اشاره کنند. برای این‌که شما عبارت یا کاراکتری را به عنوان اشاره‌گر مشخص کنید باید یک * قبل از اسم آن بگذارید، در اینجا مثالی از اشاره‌گر آورده شده است.

```
void main()
{
    int *p;
    void *up;
 }
```

شما می‌توانید آدرس یک int را درون یک اشاره‌گر قرار دهید و اشاره‌گر با استفاده از علامت & آدرس int را می‌گیرد.

```
void main()
{
    int i;
    int *p;
    i = 5;
    p = &i;
    return 0;
}
```

شما می‌توانید به مقدار int که اشاره‌گر به آن اشاره کرده دسترسی داشته باشید.

* باعث می‌شود که اشاره‌گر دوباره بازگشت داده شود، یعنی در واقع همان متغیر می‌شود و تغییر در آن به منزله‌ی تغییر در متغیر است.

```
void main()
{
    int i, j;
    int *p;
    i = 5;
    p = &i;
    j = *p;   //j = i
    *p = 7;  //i = 7
```

[1] Pointer

```
    return 0;
}
```

استفاده از اشاره‌گر برای بیان i=j در بالا راهی طولانی است. شما در ادامه با کاربردهای بیشتر اشاره‌گرها آشنا خواهید شد ولی در این قسمت، هدف آشنایی مقدماتی با این مبحث است.

۱-٤-۷- آرایه‌ها

اگر شما می‌خواستید ۵ متغیر داشته باشید، مانند زیر عمل می‌کردید:

```
int i1, i2, i3, i4, i5;
```

حال اگر ۱۰۰۰ متغیر بود چه می‌کردید؟ مسلماً فرایند بالا زمان زیادی می‌برد، ولی با استفاده از آرایه می‌توانید: با نام یک متغیر، هر تعداد که بخواهید متغیر بسازید. آرایه مانند متغیر معمولی است و برای تعریف آن کافی است فقط بعد از نام آن یک جفت براکت بیاورید و درون آن تعداد متغیر مورد نظر را بنویسید. مثال:

```
int a[5]
```

برای این‌که به مقدار متغیرهای هر خانه دست یابید شما باید نام آرایه و سپس شماره‌ی خانه‌ی مورد نظر را بیاورید. فقط به یاد داشته باشید که شماره‌ی خانه‌های آرایه از صفر شروع می‌شود. برای مثال یک آرایه به طول ۵ دارای خانه‌هایی از شماره‌ی ۰ تا ٤ است.

```
int a[5];
a[0] = 12; a[1] = 23;
a[2] = 34; a[3] = 45;
a[4] = 56;
printf("%d",a[0]);
```

- استفاده از آرایه با حلقه

استفاده از آرایه در حلقه کاربرد زیادی دارد. زیرا خانه‌های آرایه یک دنباله را طی می‌کنند که این فرایند مشابه حلقه‌ها است. برای مثال: وقتی خانه‌های آرایه صفر نشده‌اند و شما نیاز دارید که مقدار آن ها را صفر کنید باید مانند زیر از حلقه استفاده کنید:

```
int a[10];
for (i = 0;i < 10;i++)  a[i] = 0;
```

- آرایه‌های چند بعدی

آرایه‌هایی که ما قبلاً استفاده کردیم آرایه‌های یک بعدی نام دارند زیرا تنها از یک سطر تشکیل شده‌اند. آرایه‌های ۲ بعدی از چند سطر و ستون تشکیل شده‌اند، در برای تسلط بیشتر بر موضوع شکل (۱-۱۳) را ملاحظه نمائید.

آرایه‌های یک بعدی

مقدار	شماره سطر
٤	٠
٣	١
-١	٢

آرایه ی دو بعدی

	٠	١	٢
٠	١	٢	٣
١	٤	٥	٦
٢	٧	٨	٩

شکل (١-١٣): آرایه‌های چند بعدی

شما می‌توانید از آرایه‌های سه بعدی یا بیشتر نیز استفاده کنید ولی اغلب کاربردی ندارند. در ادامه برنامه‌ای آورده شده که شما را با روش تعریف یک آرایه‌ی دو بعدی و چگونه کار کردن آن آشنا می‌کند. توجه داشته باشید که مثال دارای ٢ حلقه است زیرا می‌خواهیم مقدار متغیرهای درون سطر و ستون های مختلف را تغییر دهیم.

```
int a[3][3], i, j;
    for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++)
            a[i][j] = 0;
```

١-٤-٨- رشته

رشته آرایه‌ای از کاراکترهاست که به یک ٠ یا یک کاراکتر خالی (null) ختم شود تا نشان دهد که رشته کجا تمام شده است. توجه کنید که کاراکتر null جزو رشته به حساب نمی‌آید. برای استفاده از رشته دو راه وجود دارد:

١- استفاده از آرایه‌ای از کاراکترها

٢- استفاده از اشاره‌گر رشته

آرایه کاراکتری مانند آرایه زیر تعریف می شود:

```
char ca[10];
```

شما باید مقدار هر خانه‌ی آرایه را به کاراکتر مورد نظر خود اختصاص، و کاراکتر پایانی خود را ۰ قرار دهید. به یاد داشته باشید که برای اشاره کردن به یک رشته باید از s٪ استفاده کنید.

```
char ca[10];
   ca[0] = 'H'; ca[1] = 'e';
   ca[2] = 'l';  ca[3] = 'l';
   ca[4] = 'o'; ca[5] = 0;
printf("%s", ca);
```

وقتی که شما یک مقدار خاص را به یک اشاره‌گر رشته نسبت می‌دهید کامپایلر یک صفر در انتهای آن قرار می‌دهد و دیگر نیازی نیست تا مانند آرایه کاراکتری، ۰ را در انتهای آن قرار دهید.

```
char ca[10], *sp;
scanf("%s", ca);
  sp = ca;
scanf("%s", sp);
```

شما می‌توانید یک رشته را در یک آرایه کاراکتری با استفاده از scanf بخوانید ولی برای خواندن یک اشاره‌گر رشته، کامپایلر باید آن را به یک آرایه کاراکتری برگرداند. فایل سرآمد string.h دارای توابع پرکاربردی برای کار با رشته‌هاست. در اینجا به شرح تعدادی از آن‌ها می‌پردازیم:

strcpy (مقصد ، منبع)

شما نمی‌توانید در زبان C از چنین دستوری استفاده کنید: string1 = string2. بلکه شما باید از تابع strcpy برای کپی کردن یک رشته درون یک رشته‌ی دیگر استفاده کنید. تابع strcpy رشته‌ی منبع را در رشته‌ی مقصد کپی می‌کند.

```
s1 = "abc";
s2 = "xyz";
strcpy(s1, s2); // s1 = "xyz"
```

strcat (مقصد، منبع) :

رشته‌ی منبع و مقصد را با هم ترکیب نموده و در رشته‌ی مقصد قرار می‌دهد.

```
s1 = "abc";
s2 = "xyz";
strcat(s1, s2); // s1 = "abcxyz"
```

strcmp(اول، دوم) :

این تابع رشته‌ی اول را با رشته‌ی دوم مقایسه می‌کند، اگر رشته‌ی اول بزرگتر از رشته‌ی دوم بود، تابع عددی مثبت را برگشت می‌دهد، اگر هر دو رشته مساوی

بودند تابع مقدار عددی صفر را برگشت می‌دهد و اگر رشته اول کوچکتر از رشته‌ی دوم بود تابع عددی منفی را برگشت می‌دهد.

```
s1 = "abc";
s2 = "abc";
i = strcmp(s1, s2); // i = 0
```

strlen(رشته)

این تابع تعداد کاراکترهای رشته را بازگشت می‌دهد.

```
s = "abcde";
i = strlen(s); // i = 5
```

۱-٤-۹- توابع

توابع[1] در واقع، زیر برنامه می‌باشند. شما قبلا از توابع استفاده کردید، منظور همان تابع main است. شما قبل از این‌که یک تابع را فراخوانی کنید باید آن را در ابتدای برنامه و قبل از main تعریف کنید. در مورد هر تابع می‌باید نوع return مشخص شود (یعنی نوع آرگومان برگشتی تابع مشخص شود مثلا از نوع int است یا از نوع char یا نظایر آن). در صورتی که نمی‌خواهید مقدار خاصی را return کنید از void استفاده کنید. بعد اسم منحصر به فرد تابع را بنویسید و پس از آن یک جفت پرانتز قرار دهید. در صورتی که تابع شما آرگومان ورودی دارد با مشخص کردن نوع و تعداد آرگومآن‌ها، آن‌ها را درون پرانتز جلوی اسم تابع قرار دهید و در صورتی که تابع شما آرگومان ورودی ندارد درون پرانتز عبارت void را نوشته یا آن را خالی باقی بگذارید. در آخر نیز دو عدد { } بگذارید و برنامه ی خود درون آن بنویسید. تابع تعریف شده در مثال زیر را بررسی نمائید:

```
#include <stdio.h>
 void Hello()
{
    printf("Hello\n");
}

void main()
{
    Hello();
}
```

پارامتر های تابع، مقادیری هستند که ما به یک تابع می‌دهیم تا بتواند آن‌ها را محاسبه کند. شما باید در داخل پرانتزهای پارامتر، متغیرها را بیاورید تا مقدار پارامتر را که

[1] Functions

قابل قبول است در تابع قرار دهد. دراینجا ما Function Add که دو پارامتر را با هم جمع می کند آورده شده است:

```
#include<stdio.h>
int Add(int a, int b)
{
    return  a+ b;
}

void main()
{
    int answer;
    answer = Add(5, 7);
}
```

شما می‌توانید آدرس متغیر را در یک تابع قرار دهید، در اینجا کپی صورت نمی‌گیرد و شما نیاز به اشاره‌گر دارید.

```
#include<stdio.h>

int Add(int *a, int *b)
 {
    return *a + *b;
 }

int main()
{
    int answer;
    int num1 = 5;
    int num2 = 7;
    answer = Add(&num1, &num2);
    printf("%d\n", answer);
    return 0;
}
```

۱-٤-۱۰- متغیرهای سراسری و محلی

متغیرهای محلی[1] فقط در داخل یک تابع قابل استفاده هستند و بیرون از آن نمی‌شود از این نوع متغیرها استفاده کرد. اگر شما نیاز دارید که بتوانید از یک متغیر در تمامی قسمت‌های برنامه استفاده کنید، باید آن را سراسری[2] تعریف کنید.

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

[1] Local
[2] Global

```
 int a;       // Global variables
int b;       // Global variables

int Add()
{
    return a + b;
}

int main() {
    int answer;  // Local variable
    a = 5;
    b = 7;
    answer = Add();
    printf("%d\n", answer);
    return 0;
}
```

فصل دوم آشنایی با پورت‌ها

با توجه به این‌که این کتاب با عنوان پروژه‌های کاربردی با استفاده از میکروکنترلر AVR است، فرض بر آن است که خوانندگان کتاب از آشنایی مقدماتی با میکروکنترلرهای AVR برخوردار هستند. بر مبنای این فرض، جزئیات بیشتر در رابطه با معماری ساخت این تراشه‌ها و انواع خانواده‌های آن‌ها بحث نخواهند شد. در صورت تمایل به یادگیری بیشتر می‌توانید به مراجع آخر کتاب مراجعه نمائید.

در این فصل، کار با پورت‌های موازی میکروکنترلر AVR به صورت پروژه‌های کاربردی آموزش داده می‌شود. دقت شود: کلیه برنامه‌ها به زبان برنامه‌نویسی C و در محیط نرم‌افزار CodeVisionAVR نوشته و کامپایل می‌شوند. سپس توسط نرم افزار Proteus شبیه‌سازی شده و با اجرای آن‌ها، صحت برنامه‌های نوشته شده، آزمایش و تائید می‌شود.

پروژه اول: چراغ چشمک‌زن

✍ برنامه‌ای بنویسید که یک LED متصل به یکی از پایه‌های میکروکنترلر را به صورت متوالی روشن و خاموش کند (بدون استفاده از محیط ویزارد برنامه نوشته شود)؟

فرضیات پروژه: میکروکنترلر مورد استفاده ATmega16 است و از کریستال داخلی ٤ مگاهرتز استفاده شده است. LED به پین صفرم از پورت B متصل شده است و LED به صورت متوالی یک ثانیه روشن و یک ثانیه خاموش می‌شود.

☑ حل: توضیحات لازم برای نوشتن برنامه: میکروکنترلر ATmega16 دارای چهار پورت است که به ترتیب آن‌ها را A، B، C و D می‌نامیم. هر یک از آن‌ها دارای هشت پایه (پین) هستند که به صورت صفر تا هفت شماره‌گذاری می‌شوند. به عنوان مثال: پین صفرم پورت B با PB.0 نشان داده می‌شود.

❋ یادآوری: رجیسترهای مورد استفاده برای تنظیم پورت‌ها مطابق زیر هستند: رجیسترهای پورت‌های ورودی خروجی (I/O):

۱- رجیستر DDRX : این رجیستر برای تنظیم جهت ورودی یا خروجی پایه استفاده می‌شود. به این ترتیب که: اگر بیتی در رجیستر DDRX یک شده باشد، پین متناظر با آن در حالت خروجی و با مقدار صفر در حالت ورودی قرار می‌گیرد. که مطابق جدول (۲-۱) است.

۲- رجیستر PORTX : اگر DDRX.pin=1 باشد (پایه در حالت خرجی باشد) این رجیستر برای نوشتن یک مقدار منطقی در پین استفاده می‌شود و اگر DDRX.pin=0 باشد (پایه در حالت ورودی باشد) برای تنظیم امپدانس پایه به کار می‌رود. که مطابق جدول (۲-۱) است.

جدول (۲-۱): وضعیت پایه‌ها بر اساس محتوای رجیسترهای متناظر

DDRX.PIN	PORTX.PIN	وضعیت یا مقدار پایه (پین)
1	0	"0"
	1	"1"
0	0	Tri State
	1	Pull-up (بالاکش)

- در حالت Tristate، امپدانس پایه‌ی مورد نظر بی نهایت شده و ارتباط بین پایه و مدار داخلی پورت قطع می‌گردد.
- در حالت Pull-up، پین مورد نظر با یک مقاومت بالاکش داخلی به Vcc متصل می‌گردد. این حالت ورودی برای اتصال مدارهایی که خروجی سه حالته دارند به پایه ورودی میکروکنترلر مناسب است.

۳- رجیستر PINX : این رجیستر، تنها یک رجیستر خواندنی است که مقدار منطقی روی پایه‌ها را مستقل از این که در حالت ورودی یا خروجی پیکربندی شده باشند، نشان می‌دهد.

- حداکثر جریانی که هر پین در حالت خروجی تامین می‌کند به اندازه‌ی است که بتواند یک LED را روشن نماید. به همین دلیل، می‌توان LED ها را مستقیماً به پایه‌های مورد نظر از میکروکنترلر متصل نمود (جریانی به اندازه‌ی 40mA در ولتاژ تغذیه‌ی پنج ولتی و جریان 20mA در ولتاژ تغذیه‌ی سه ولتی).

- با فعال‌سازی بیت PUD (یعنی نوشتن یک در بیت PUD) در رجیستر SFIOR می‌توان مقاومت‌های بالاکش کلیه پورت‌ها را غیر فعال نمود.
- در نرم‌افزار CodevisionAVR باید رجیستر ها با حروف بزرگ نوشته شوند.

ابتدا کلیه مراحل لازم را به منظور ایجاد یک پروژه بدون استفاده از ویزارد انجام دهید، سپس درمحیط ایجاد شده، برنامه‌ای مطابق زیر بنویسید:

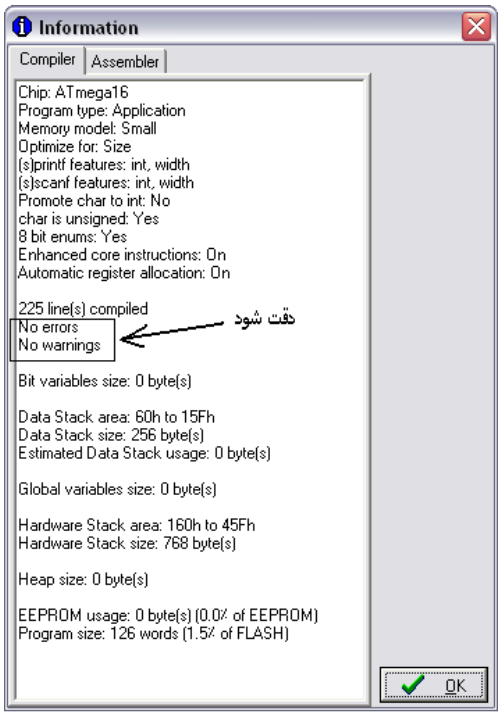
✓ **برنامه:**

```
#include <mega16.h>
#include <delay.h>
void main(void)
{DDRB.0=1;
PORTB.0=0;
while (1)
    {
        PORTB.0=1;
        delay_ms(1000);
        PORTB.0=0;
        delay_ms(1000);
  };}
```

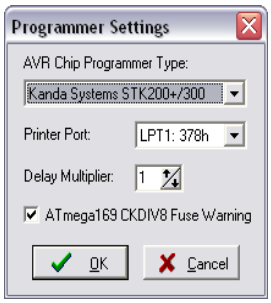
شرح برنامه: همان‌طور که مشاهده می‌شود ابتدا کتابخانه مربوط به میکروکنترلر AVR (که در اینجا ATmega16 است) و کتابخانه تاخیر، در ابتدای برنامه قرار داده شده‌اند. با استفاده از DDRB.0=1 پین صفرم از پورت B به عنوان خروجی و توسط PORTB.0=0 با مقدار صفر مقداردهی اولیه شده است. پس از اجرای برنامه، دستورات درون حلقه while برای همیشه تکرار می‌شوند. به این صورت که: ابتدا PORTB.0 یک و پس از یک ثانیه صفر می‌شود و دوباره پس از یک ثانیه یک می‌شود. در صورتی که پایه مثبت (آند) یک LED به PORTB.0 میکرو متصل و پایه منفی (کاتد) LED به زمین وصل شود این LED به‌صورت متناوب و با تاخیر یک ثانیه روشن و خاموش می‌شود.

کامپایل و برنامه‌ریزی تراشه: پس از اتمام نوشتن برنامه بر روی گزینه Compile and Make کلیک کرده یا کلیدهای Shift و F9 را به صورت همزمان فشار داده تا فایل Hex آن تولید شود. پنجره‌ای مطابق شکل (۲-۱) ظاهر می‌شود که وضعیت عمل کامپایل و ساخت فایل Hex را نشان می‌دهد و در صورتی که برنامه اشکالی (املایی، ساختاری و نظایر آن) نداشته باشد، عبارت No errors در آن ظاهر می‌شود. اکنون که برنامه به درستی کامپایل و فایل Hex ساخته

شده است، می‌باید فایل Hex به میکروکنترولر انتقال یابد (میکروکنترولر پروگرم شود). با توجه به پروگرمر ساخته شده در فصل اول، که از نوع STK200/300 است، در محیط CodeVisionAVR از گزینه‌ی Setting بر روی Programmer کلیک کرده تا پنجره‌ی شکل (۲-۲) ظاهر شود و از AVR programmer گزینه‌ی STK 200/300 را انتخاب کرده و دکمه‌ی OK را فشار داده تا پروگرمر مورد نظر انتخاب شود.

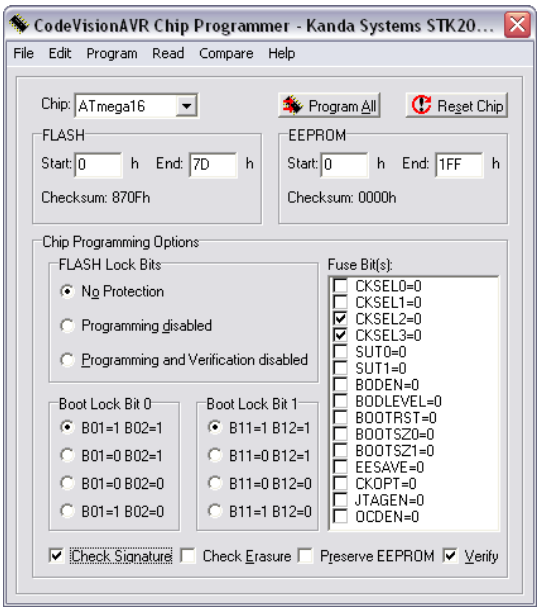


شکل (۲-۱): نتیجه‌ی عملیات کامپایل



شکل (۲-۲): انتخاب نوع پروگرمر

اکنون، نوبت پروگرم نمودن میکروکنترلر است. مطابق شکل (۱-٤) از فصل اول، میکروکنترولر را به پروگرمر متصل نموده و منبع تغذیه را روشن نمائید. با فشردن همزمان کلیدهای Shift و F4 پنجره‌ای مطابق شکل (۲-۳) ظاهر می‌شود، فیوز بیت‌ها را برای نوسان ساز داخلی ٤ مگاهرتز و سایر قسمت‌ها را مطابق شکل (۲-۳) تنظیم و سپس بر روی Program All کلیک نمائید تا میکروکنترلر مورد نظر، برنامه‌ریزی شود. به منظور آشنایی کامل با قسمت‌های مختلف نرم‌افزار، همچنین روش دقیق برنامه ریزی میکروکنترلر به CD همراه کتاب مراجعه شود.



شکل (۲-۳): روش تنظیمات فیوز بیت‌ها و برنامه‌ریزی میکروکنترلر

تمرین: برنامه زیر، برنامه یک فلاشر است: ابتدا آن را تحلیل و سپس در محیط Codevision بنویسید و اجرای آن را توسط نرم‌افزار Proteus مشاهده نمائید؟

```
#include <mega16.h>
#include <delay.h>
#define xtal 4000000
int i;
void main (void)
{DDRD = 0xFF;
 while(1)
```

```
  {   for(i = 1; i <= 128; i = i*2)
          {PORTD = i;delay_ms(100);}
      for(i = 128; i > 1; i = i/2)
          {PORTD = i;delay_ms(100);}
    }
}
```

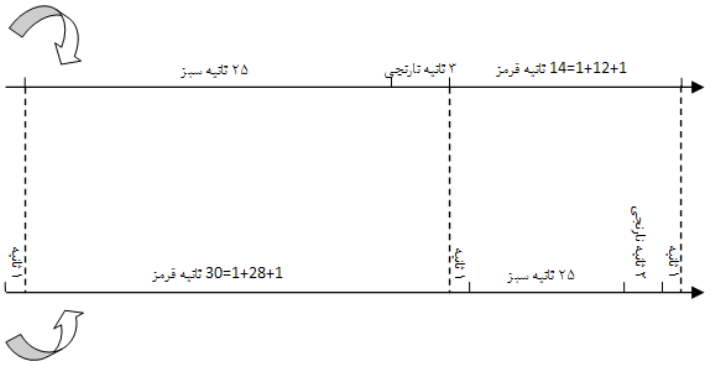
پروژه دوم: کنترل چراغ راهنمایی با نمایش مدت انتظار

✍ بدون استفاده از وقفه‌ها و تایمرهای میکروکنترلر، برنامه‌ای بنویسید که چراغ‌های راهنمایی دو زمانه یک چهار راه را کنترل کند و با پیشنهاد یک روش بهینه به منظور استفاده از پایه‌های میکروکنترلر برنامه دو شمارنده معکوس شمار دو رقمی را برای دو سمت چهار راه بنویسید؟

فرضیات پروژه: میکروکنترلر مورد نظر ATmega16 است و از کریستال داخلی ٤ مگاهرتز استفاده شده است. برای این منظور ٦ عدد LED به پایه‌های پورت D میکروکنترلر مطابق جدول (۲-۲) متصل شده‌اند. وضعیت رنگ دو چراغ نسبت به هم مانند شکل (۲-٤) است. یک شستی تغییر وضعیت (Toggle) برای این چراغ لازم است که مامور راهنمایی و رانندگی بتواند با هر بار فشردن آن وضعیت چراغ‌ها را به طور همزمان عوض کند.

جدول (۲-۲): نحوه اتصال چراغ‌های راهنمایی به پورت‌های میکروکنترلر

رنگ LED	پایه درگاه	
سبز	PD0	چراغ خیابان عریضتر
نارنجی	PD1	
قرمز	PD2	
	PD3	
سبز	PD4	چراغ خیابان باریکتر
نارنجی	PD5	
قرمز	PD6	
	PD7	



شکل (۲-٤): وضعیت و مدت زمان روشن و خاموش بودن چراغ‌های راهنما

برای این کار، شستی متصل به پایه‌ی PD3 درنظر گرفته شده است. با فشردن این شستی، تنها در زمانی که یکی از چراغ‌ها سبز و دیگری قرمز است، باید باعث شود که: چراغی که سبز است، سه ثانیه نارنجی و سپس قرمز شود و چراغی که قرمز است، چهار ثانیه بعد سبز شود.

- **راهنمایی:** در هر بار پایان یافتن کوچکترین حلقه‌ای که برای ساختن زمآن‌ها نوشته‌اید، وضعیت کلید را چک کنید. دقت کنید که فاصله‌ی بین دوبار چک کردن این وضعیت در صورت فشرده نشدن در حدود چند میلی ثانیه و در صورت فشرده شدن در حد چند ثانیه باشد، که این خود یک نوع Debounce نرم افزاری است.

❋ **ی‌ادآوری:** برای نمای‌ش اطلاعات روی چند نمایش‌گر «هفت قسمتی»[1] توسط م‌ی‌کروکنترلر و برای کاهش تعداد پای‌ه‌های به کار رفته، معمولاً از روش تسه‌ی‌م «خط – داده»[2] استفاده م‌ی‌شود. ای‌ن نحوه نمای‌ش بر پای‌ه ای‌ن مطلب استوار است که س‌ی‌ستم ب‌ی‌نایی انسان قادر به تشخ‌ی‌ص و تفک‌ی‌ک روی‌دادهایی که با سرعت ب‌ی‌ش از حدود ۱۷ بار در ثان‌ی‌ه روی م‌ی‌دهد، ن‌ی‌ست. مثلاً چشم ما، لامپ‌ی را که ۱۰۰ بار در ثان‌ی‌ه روشن و خاموش م‌ی‌شود، به طور پ‌ی‌وسته روشن م‌ی‌ب‌ی‌ند، ول‌ی با شدت نور کمتر از حالت روشن بودن دائم آن قابل روی‌ت است.

---

[1] 7-Segment
[2] Line-Data Multiplex

یک روش معمول در اتصال و بسته‌بندی[1] چند نمایش‌گر هفت قسمتی، قسمت‌های متناظر همه‌ی نمایش‌گرها به هم و در نهایت خارج کردن ۸ پایه (هفت قسمت به همراه یک نقطه) از بسته‌بندی، به عنوان پایه‌های داده‌ی نمایش‌گر و خارج کردن جداگانه‌ی خط مشترک هر نمایش‌گر به عنوان پایه‌ی فعال‌ساز (یا همان آدرس) آن نمایش‌گر، است. بنابراین از یک بسته‌ی ٤ تایی از نمایش‌گرهای ۷ قسمتی که به این روش بسته‌بندی شده است، ۱۲ پایه شامل ۸ پایه داده مشترک میان همه نمایش‌گرها و ٤ پایه فعال‌ساز مخصوص هر نمایش‌گر (یا همان پایه‌های آدرس) خارج می‌شود. بنابراین، در این نوع اتصال در هر لحظه از زمان تنها یک داده‌ی واحد می‌تواند روی تعداد دلخواهی از آن‌ها به نمایش در آید. مالتی پلکس خط–داده همان روشی است که برای نمایش یک داده‌ی دلخواه (که لزوماً برای همه‌ی نمایش‌گرها مشابه نیست.) روی این نمایش‌گرها به کار می‌رود. اگر با سرعت بالا و در کسری از ثانیه، داده‌ی مربوط به هر نمایش‌گر را همزمان با فعال کردن آن، روی خط داده ارسال کنیم، سپس داده‌ی نمایش‌گر بعد را همزمان با فعال کردن آن و به همین ترتیب تا آخرین نمایش‌گر پیش برویم و مجدداً به نمایش‌گر اول بازگشته و این کار را به طور مداوم ادامه دهیم، با توجه به همان نکته‌ای که ذکر شد، کل داده به صورت ثابت و در یک زمان، روی مجموعه‌ی نمایش‌گرها مشاهده می‌شوند. با این کار به جای استفاده از ۳۲ پایه‌ی میکروکنترلر می‌توان از ۱۲ پایه‌ی آن برای راه اندازی[2] کامل ٤ نمایش‌گر هفت قسمتی استفاده کرد.

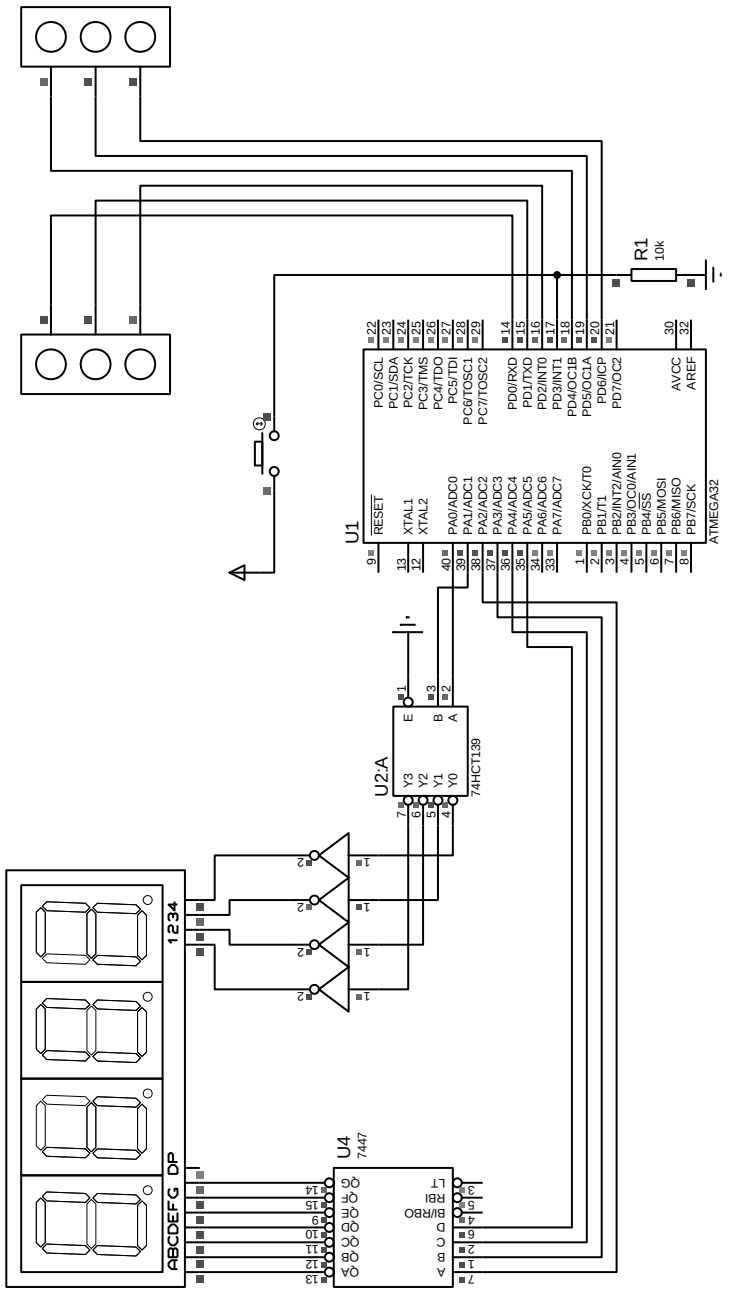
☑ **حل:** به منظور کاهش بیشتر تعداد پایه‌های به کار رفته، می‌توان از یک «مبدل کد باینری به ۷ قسمتی»[3] (تراشه ۷٤٤۷) به عنوان واسط خط داده و از یک «کدگشای ۲ به ٤ خطی»[4] به عنوان واسط خط آدرس (تراشه ۷٤۱۳۹) استفاده کرد که این امر منجر به استفاده بهینه از پایه‌های میکرو می‌شود و در شکل (۲–۵) نشان داده شده است. جهت پیاده‌سازی نرم افزاری، ابتدا کلیه مراحل لازم را به منظور ایجاد یک پروژه بدون استفاده از ویزارد انجام داده، سپس درمحیط ایجاد شده، برنامه‌ای مطابق زیر بنویسید:

[1] Package
[2] Drive
[3] BCD to 7-segment Decoder
[4] 2-to-4-Line Decoder/Demultiplexer

✓ برنامه:

```
#include <mega32.h>
#include <delay.h>
int s,x,m,n,q,r,i,b,t,d;
void output(void);
void time(void);
void change(void);
void main()
{ DDRA=0x3F; DDRD=0x77; t=2;
 while(1){
 if(t==2){s=25;x=29;}
 if(t==1){s=14;x=10;}
 m=s%10;
 n=s/10;
 q=x%10;
 r=x/10;
 time();
 }//end of while
 }
//*****************************************************
void time(void){
int f=1;
if(t==1)PORTD=0x14;
if(t==2)PORTD=0x41;
while(f){d=25;
while(d){
for(i=0;i<4;i++){
if(PIND.3==1)change();
switch(i){
case 3:b=n; break;
case 2:b=m; break;
case 1:b=r; break;
case 0:b=q; break;}//end of switch
output();
delay_ms(5);}//end of for
d--;}//end of while
if(q==0 & r==3)r=0;
if(m==0 & n!=0){n--;m=9;} else if(n==0 & m!=0){n=0;m--;} else
if(m!=0 & n!=0)m--;
if(q==0 & r!=0){r--;q=9;} else if(r==0 & q!=0){r=0;q--;} else if(q!=0 &
r!=0)  q--;
```

```
if(m==0 & n==0){if(t==1){t=2;f=0;  goto exit;}
else if(q<5&q>1)PORTD=0x42; else if(q<2)PORTD=0x44;}
if(q==0 & r==0){if(t==2){t=1;f=0;} else if(m<4 & m>1)PORTD=0x24;
else if(m==1){PORTD=0x44;r=3;q=0;}}
exit:
//end of exit
}//end of while
}
//*********************************************************
void output(void){
int c;
switch(b){
case 0:c=0x03;break;
case 1:c=0x07;break;
case 2:c=0x0B;break;
case 3:c=0x0F;break;
case 4:c=0x13;break;
case 5:c=0x17;break;
case 6:c=0x1B;break;
case 7:c=0x1F;break;
case 8:c=0x23;break;
case 9:c=0x27;break;
}//end of switch
if(i==0) PORTA=c&0xFC;
if(i==1) PORTA=c&0xFD;
if(i==2) PORTA=c&0xFE;
if(i==3) PORTA=c&0xFF;
}//end of output
//*********************************************************
void change(void){
if(t==2 & m>0){PORTD=0x42;m=0;n=0;r=0;q=4;d=25;}
if(t==1 & q>0){PORTD=0x24;m=3;n=0;r=0;q=0;d=25;}
}
```



شکل (۲-۵): سخت افزارکنترل چراغ راهنمایی و نمایش مدت انتظار با میکروکنترلر

پروژه سوم: نمایش کاراکترها بر روی نمایش‌گرهای ماتریسی

برنامه‌ای بنویسید که بر روی نمایش‌گر ماتریسی ۸×۸ حروف A، B و C را نمایش دهد؟

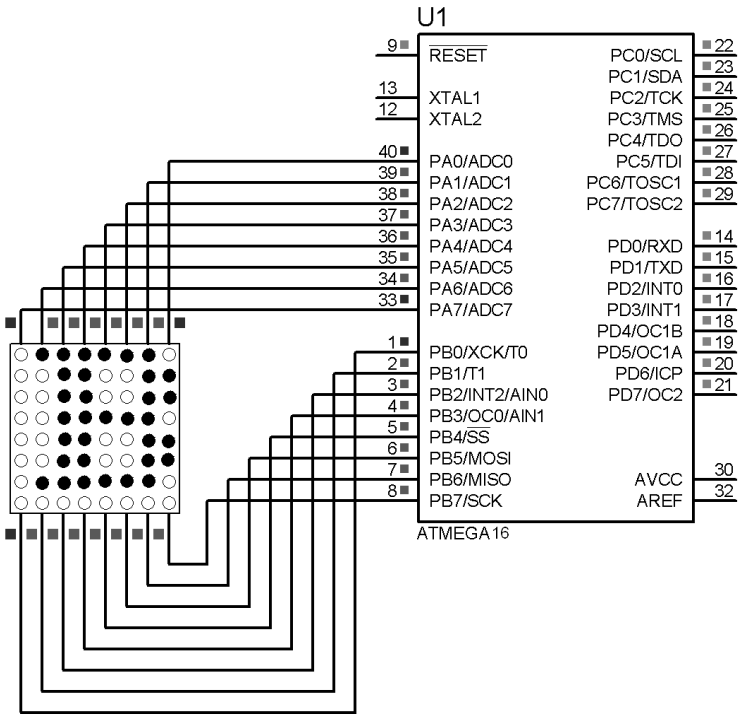
فرضیات پروژه: میکروکنترولر مورد نظر ATmega16 است و از کریستال داخلی ٤ مگاهرتز استفاده شده است. برای این منظور یک نمایش‌گر ماتریسی ۸×۸ به پایه‌های پورت A و B میکروکنترلر مطابق شکل (۲-٦) متصل شده‌اند.

حل: به منظور نمایش حروف بر روی نمایش‌گر ماتریسی، از روش جاروب نمودن سطرها و ستون‌ها استفاده می‌شود و می‌باید مطابق پروژه دوم عمل نمود.

برنامه:

```
#include <mega16.h>
#include <delay.h>
unsigned char k;  int l;
unsigned char arr1[8]={0x18, 0x3C, 0x66, 0x66, 0x7E, 0x66, 0x66, 0x00};
unsigned char arr2[8]={0x7E, 0x33, 0x33, 0x3E, 0x33, 0x33, 0x7E, 0x00};
unsigned char arr3[8]={0x1E, 0x33, 0x60, 0x60, 0x60, 0x33, 0x1E, 0x00};
void main(void)
{PORTA=0xFF; DDRA=0xFF;
 PORTB=0xFF; DDRB=0xFF;
while (1)
      {for(l=0;l<1000;l++)
          {for(k=0;k<=7;k++)
           {     PORTA=arr1[k];
                 PORTB&=~(1<<k);
                 delay_us(100);
                 PORTB=0xFF;
           }
          }
      for(l=0;l<1000;l++)
          {for(k=0;k<=7;k++)
           {     PORTA=arr2[k];
                 PORTB&=~(1<<k);
                 delay_us(100);
                 PORTB=0xFF;
           }
```

```
        }
    for(l=0;l<1000;l++)
        {for(k=0;k<=7;k++)
         {    PORTA=arr3[k];
              PORTB&=~(1<<k);
              delay_us(100);
              PORTB=0xFF;
         }
        }
    }
}
```



شکل (۲-٦): نمایش‌گر ماتریسی برای نمایش حروف

---

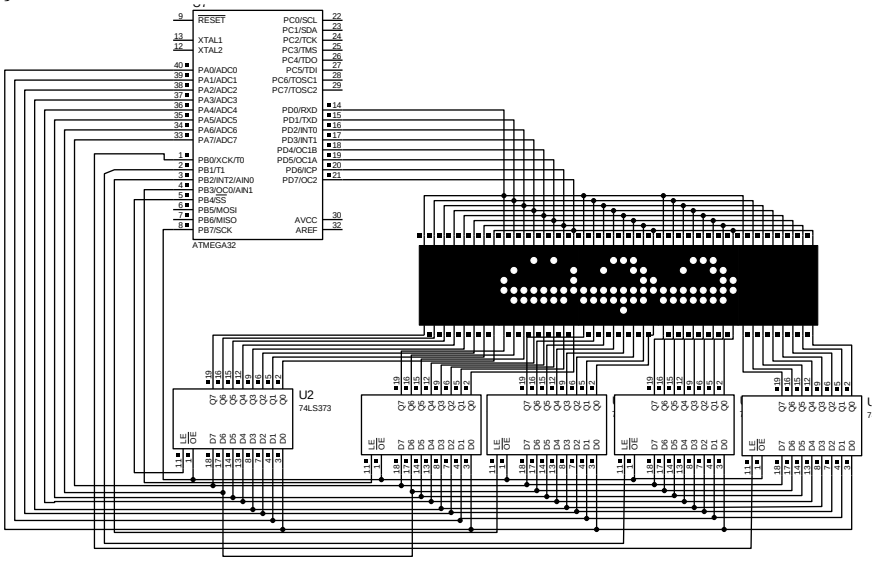
۳ **تمرین:** برنامه زیر را تحلیل نمائید و آن را در توسط نرم‌افزار CodeVision بازنویسی کنید. سپس نتایج آن را در محیط نرم‌افزار Proteus با نتایج شکل (۲-۷) مقایسه نمائید؟

```
#include <mega32.h>
#include <delay.h>
unsigned char i,j,k,n,r, lsh, pos,char1,char2,charnum;
unsigned char ar1[17][8]={
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x1C,0x26,0x03,0xFF,0xFE,0x00,0x00,
0x00,0x1C,0x26,0x03,0xFF,0xFE,0x08,0x00,
0x00,0x14,0x41,0x81,0xFF,0x7E,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00};

void main(void)
{
charnum=13;
charnum=charnum-4;
PORTA=0x00;DDRA=0xFF;
PORTB=0x00;DDRB=0xFF;
PORTD=0xFF;DDRD=0xFF;

while(1)
{
pos=0;
  for(i=0;i<=charnum;i++)
    {
      lsh=8;
        for(r=1;r<=8;r++)
          {
            lsh--; pos++;
                for(n=1;n<=33;n++)
                  {
                    for(k=0;k<=7;k++)
                      {
                          for(j=0;j<=4;j++)
                            {
```

```
                        char1=ar1[i+j][k]>>r;
                        char2=ar1[i+j+1][k]<<lsh;
                        PORTA=char1|char2;
                        PORTB|=1<<j;PORTB=0x00;
                      }
                PORTD&=~(1<<k);
                delay_us(66);
                PORTD=0xFF;
               }
            }
         }
      }
   }
}
```



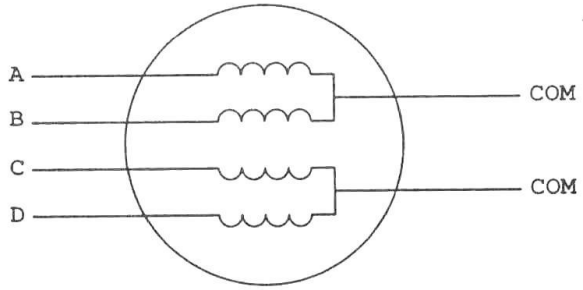
شکل (۲-۷): پیکربندی تابلوی روان ماتریسی

بازی و سرگرمی: با مراجعه به CD همراه کتاب، علاوه بر مشاهده برنامه بازی مار (Snake) در محیط CodeVision با اجرای آن در محیط Proteus سرعت عمل خود را آزمایش نمائید و در ساختار برنامه را تحلیل کنید.

موتورهای پله‌ای دارای کاربرد زیاد به منظور کنترل موقعیت است. زیرا، میزان چرخش شفت موتور متناسب با تعداد پالس‌های اعمال شده به هر فاز آن می‌باشد.

این موتورها، در کاربردهایی همچون درایورهای سخت، پرینترهای ماتریسی، کنترل بازوهای ربات و نظایر آن استفاده می‌شوند. هر موتور پله‌ای از یک هسته متحرک مغناطیسی دائمی که به آن روتور (یا شفت) می‌گویند و یک بخش ثابت که استاتور نامیده می‌شود، تشکیل شده است. موتورهای پله‌ای عموماً دارای چهار سیم‌پیچ استاتور هستند و به موتورهای چهار فاز معروف می‌باشند. یک موتور پله‌ای علاوه بر چهار سیم ذکر شده، ممکن است یک یا دو سیم برای اتصال به ولتاژ تغذیه مثبت یا منفی داشته باشد. شکل (۲-۸) ارتباط بین سیم‌پیچ‌های استاتور را نشان می‌دهد.

برای راه‌اندازی موتور پله‌ای به طور کلی دو روش «گام کامل»[1] و «نیم گام»[2] وجود دارد. که روش گام کامل، به دو رهیافت هدایت تک‌فاز و هدایت دو فاز تقسیم‌بندی می‌گردد که در ادامه به توضیح هر یک از این روش‌ها می‌پردازیم.

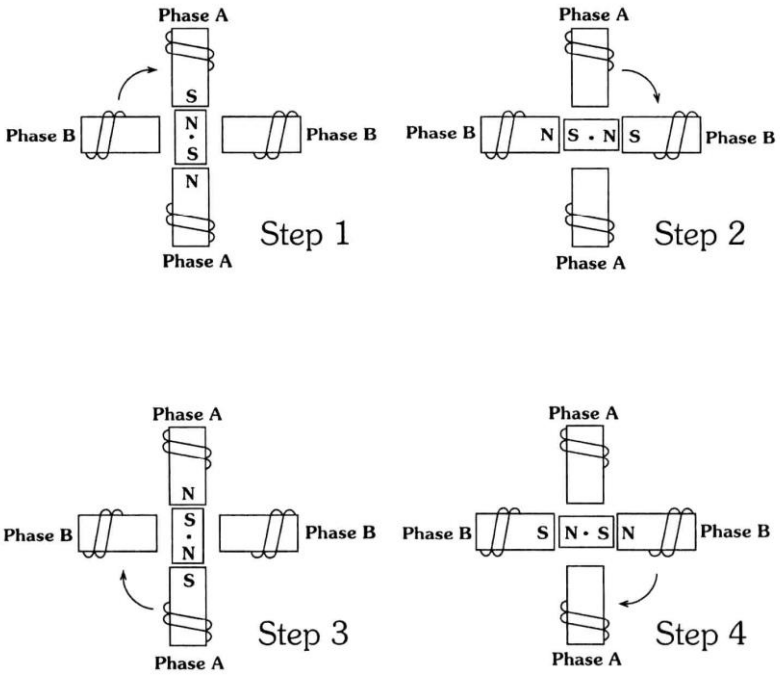


شکل (۲-۸): نحوه‌ی پیکربندی سیم‌پیچ‌های استاتور

الف- راه اندازی موتور به صورت گام کامل

۱- راه اندازی موتور به روش هدایت تک‌فاز: در این روش مطابق شکل (۲-۹)، در هر لحظه تنها یک جفت از سیم‌پیچ‌های استاتور و روتور همدیگر را جذب می‌کنند. در این صورت با فعال‌سازی مناسب سیم‌پیچ‌های استاتور چهار حالت شکل (۲-۹)، بوجود می‌آیند. به منظور راه‌اندازی موتور به صورت هدایت تک‌فاز، باید از جدول (۲-۳) استفاده کرد. با اجرای هر یک از این حالت‌ها شفت موتور یک پله جابجا می‌شود.

[1] Full Step
[2] Half Step



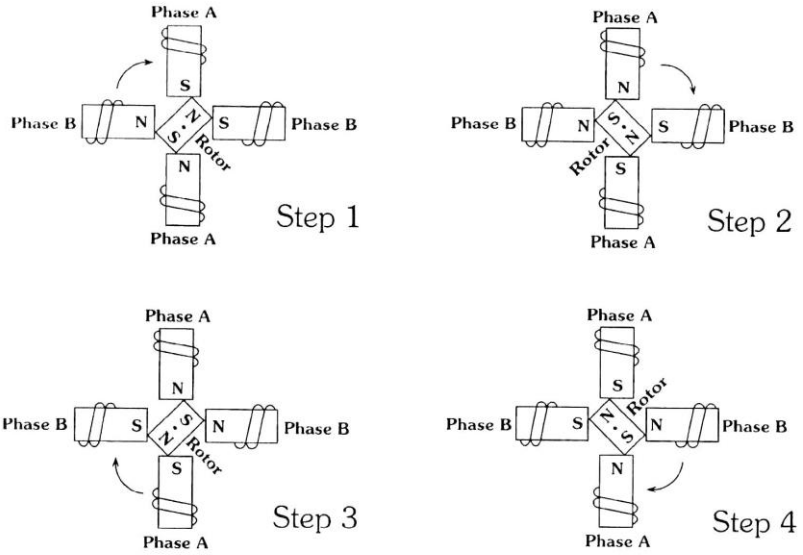
شکل (۲-۹): راه‌اندازی موتور پله‌ای به صورت هدایت تک‌فاز

جدول (۲-۳): مراحل راه‌اندازی موتور پله‌ای به صورت هدایت تک‌فاز

خلاف جهت ساعت	سیم‌پیچ D	سیم‌پیچ C	سیم‌پیچ B	سیم‌پیچ A	#مرحله	در جهت ساعت
↑	۰	۰	۰	۱	۱	↓
	۰	۰	۱	۰	۲	
	۰	۱	۰	۰	۳	
	۱	۰	۰	۰	٤	

۲- راه‌اندازی موتور پله‌ای به صورت هدایت دو فاز: در این روش، مطابق شکل (۲-۱۰) در هر لحظه هر دو جفت از سیم‌پیچ‌های استاتور با قطب‌های ناهمنام فعال می‌شوند که در این صورت قطب‌های روتور در جهت برایند قطب مخالف استاتور قرار می‌گیرند. در این حالت نیز با فعال‌سازی مناسب سیم‌پیچ‌های استاتور چهار حالت شکل (۲-۱۰) بوجود می‌آیند. تفاوت این روش با روش قبل در این است که: در این حالت گشتاور ایجاد شده افزایش

می‌یابد و برای کاربردهایی که به قدرت بیشتری نیاز دارند مفید است. در این حالت، برای راه‌اندازی موتور از جدول (۲-٤) استفاده می‌شود. با اجرای هر یک از این حالت‌ها شفت موتور یک پله جابجا می‌شود.



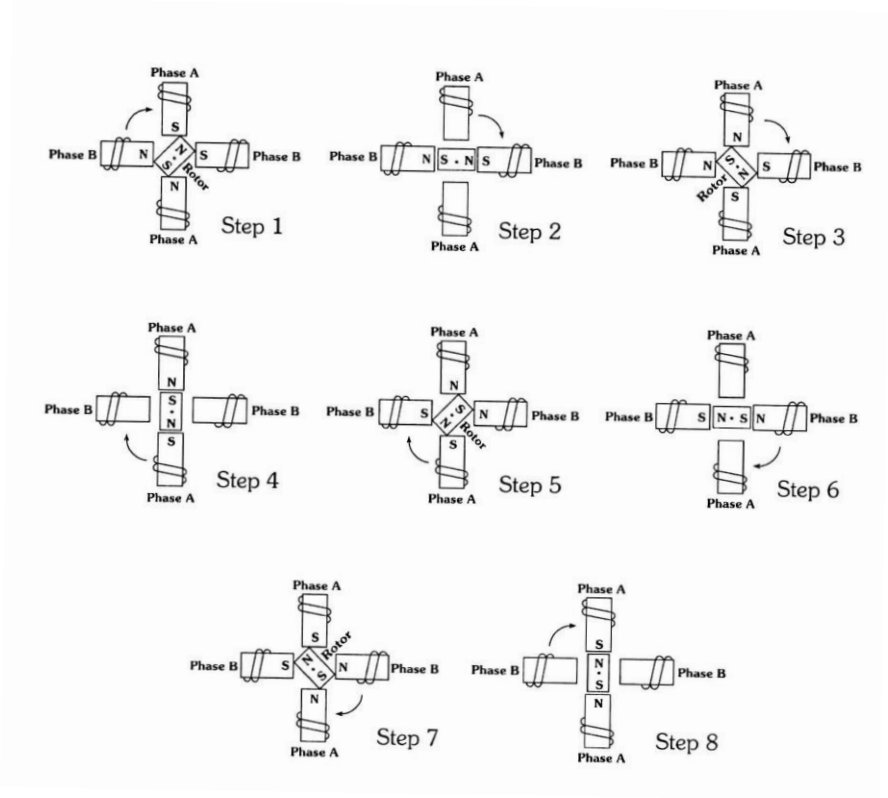
شکل (۲-۱۰): راه‌اندازی موتور پله‌ای به صورت هدایت دو فاز

جدول (۲-٤): مراحل راه‌اندازی موتور پله‌ای به صورت هدایت دو فاز

خلاف جهت ساعت	سیم‌پیچ D	سیم‌پیچ C	سیم‌پیچ B	سیم‌پیچ A	#مرحله	در جهت ساعت
↑	۱	۰	۰	۱	۱	↓
	۰	۰	۱	۱	۲	
	۰	۱	۱	۰	۳	
	۱	۱	۰	۰	٤	

ب- راه‌اندازی موتور به صورت نیم گام

در این حالت در هر بار تغییر وضعیت سیم‌پیچ‌های استاتور، میزان گردش موتور نصف حالت‌های قبل خواهد بود. در این صورت به جای چهار حالت مختلف، هشت حالت وجود خواهد داشت که با فعال شدن مناسب آن‌ها، می‌توان موتور را راه‌اندازی نمود. چگونگی عملکرد موتور در این روش، مطابق شکل (۲-۱۱) است



شکل (۲-۱۱): راه‌اندازی موتور پله‌ای به صورت نیم گام

جدول (۲-۵): مراحل راه‌اندازی موتور پله‌ای به صورت نیم گام

در جهت ساعت	#مرحله	سیم‌پیچ A	سیم‌پیچ B	سیم‌پیچ C	سیم‌پیچ D	خلاف جهت ساعت
	۱	۱	۰	۰	۰	
	۲	۱	۱	۰	۰	
	۳	۰	۱	۰	۰	
	۴	۰	۱	۱	۰	
	۵	۰	۰	۱	۰	
	۶	۰	۰	۱	۱	
	۷	۰	۰	۰	۱	
	۸	۱	۰	۰	۱	

و از جدول (۲-۵) به منظور راه‌اندازی موتور استفاده می‌شود. تفاوت این روش با روش راه اندازی گام کامل هدایت دو فاز این است که: در این روش دقت موتور به

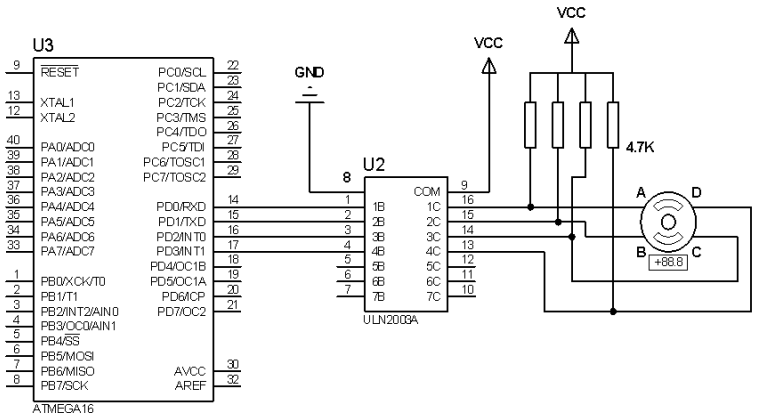
دو برابر افزایش می‌یابد ولی میزان گشتاور تولید شده در این حالت، به اندازه‌ی ۱۵ تا ۳۰ درصد کاهش می‌یابد.

زاویه پله: موتورهای پله‌ای مختلف، مشخصات گشتاور، ولتاژ و تعداد پله در دور متفاوتی دارند. تعداد پله در دور، میزان دقت موتور را نشان می‌دهد. برای بدست آوردن زاویه هر پله، می‌توان ۳۶۰ درجه را بر تعداد پله در دور تقسیم نمود. جدول (۲-۶) تعداد پله در دور و زاویه هر پله را در چند موتور مختلف را نشان می‌دهد.

جدول(۲-۶): زاویه پله برای انواع موتورهای پله‌ای

زاویه‌ی پله (درجه)	پله در دور
0.72	500
1.8	200
2.0	180
2.5	144
5	72
7.5	48
15	24

سخت‌افزار مورد نیاز برای اتصال یک موتور پله‌ای چهار فاز به میکروکنترلر در شکل(۲-۱۲) ترسیم شده است. از آنجایی که جریان دهی میکرو برای تحریک موتور پله‌ای مناسب نیست، از تراشه ULN2003 که حاوی بافر جریان می‌باشد، استفاده شده است. مشخصات تراشه‌های خانواده ULN200xA در جدول (۲-۷) آورده شده است.



شکل(۲-۱۲): سخت‌افزار مدار کنترل موتور پله‌ای ٤ فاز

جدول (۲-۷): مشخصات تراشه‌های خانواده **ULN200xA**

ULN2001A	General Purpose, DTL, TTL, PMOS, CMOS
ULN2002A	14-25V, PMOS
ULN2003A	5V TTL , CMOS
ULN2004A	6-15 V ,CMOS , PMOS

پروژه چهارم: کنترل موتورهای پله‌ای

الف- برنامه‌ای بنویسید که یک موتور پله‌ای ٤ فاز را با دقت 1.8^0 به صورت گام کامل هدایت تک‌فازی به اندازه‌ی 90^0 در جهت ساعت‌گرد بچرخاند؟

☑ حل: از آنجا که در تحریک به صورت پله کامل، هر بار تحریک روتور، منجر به چرخش روتور به اندازه 1.8^0 می‌شود. برای چرخش 90^0 باید 90/1.8=50 بار پالس فرمان تولید گردد.

✓ برنامه:

```
#include <mega16.h>
#include <delay.h>
int i=0, k=0,num;
void main(void)
{
DDRD=0x0F;
ACSR=0x80;
SFIOR=0x00;
    for(i=1;i<=13;i++)
                {
        PORTD=0b00000001;
                delay_ms(30);
                   num++;
                for(k=1;k<=3;k++)
                        {
                        PORTD=PORTD<<1;
                        delay_ms(30);
                    num++;
                    if (num==51) k=4;
                        }
        }
 }
```

✍ ب- برنامه‌ای بنویسید که یک موتور پله‌ای ٤ فاز را با دقت 1.8^0 به صورت گام کامل، به اندازه‌ی 90^0 در جهت ساعت‌گرد بچرخاند؟

☑ **حل:** برای تحریک نیم گام برای چرخش به اندازه 90^0 باید تعداد تحریک‌های هدایت تک‌فاز ۵۰ مرتبه و تعداد تحریک‌های هدایت دو فاز نیز ۵۰ مرتبه باشند به طوری که تحریک هدایت تک‌فاز و هدایت دو فاز به صورت یک در میان به طور متوالی صورت پذیرد.

✓ **برنامه:**

```
#include <mega16.h>
#include <delay.h>
int d,a,i=0,k=0,num;
void main(void)
{
DDRD=0x0F;
ACSR=0x80;
SFIOR=0x00;
for(i=1;i<=13;i++)
            {
                  d=0b00000001;
                  a=0b00000011;
        for(k=1;k<=4;k++)
                        {
                        PORTD=d;
                        d=d<<1;
                        delay_ms(30);
                         PORTD=a;
                         a=a<<1;
                         if(a==24) a=9;
                        delay_ms(30);
                        num++;
                        if(num==51) k=4;
                        }
              }
}
```

✍ پ- برنامه‌ای بنویسید که یک موتور پله‌ای ٤ فاز را با دقت 1.8^0 به صورت پله کامل، به اندازه‌ی 360^0 در جهت ساعت‌گرد با سرعت ۵۰ دور در دقیقه (RPM) بچرخاند؟

☑ **حل:** برای این که بتوان موتور را کنترل نمود، کافی است که تاخیرهای مناسب بین تحریک‌های متوالی فازها وجود داشته باشد.

- در هر دقیقه، موتور ۵۰ دور می‌چرخد. یعنی، هر دور چرخش موتور ۱۲۰۰ میلی‌ثانیه زمان می‌برد.
- در روش گام کامل، هر دور کامل یعنی چرخش ۳۶۰ درجه نیازمند ۲۰۰ مرتبه تحریک موتور پله‌ای است (تعداد تحریک‌ها در حالت هدایت تک‌فازی و هدایت دو فازی یکسان هستند). یعنی میزان تاخیر هر تحریک معادل ۶ میلی‌ثانیه است. رابطه کلی در رابطه با زمان تاخیر مطابق زیر است:

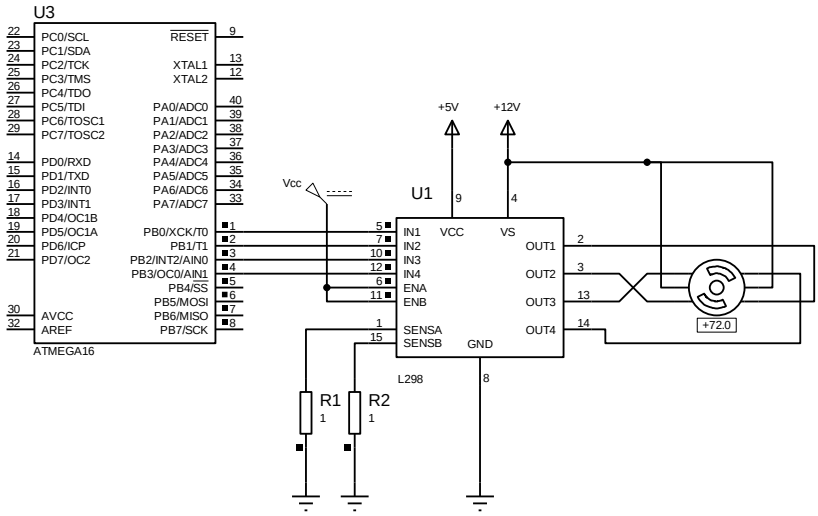
(تعداد پله در دور × سرعت برحسب RPM) /۶۰ = میزان تاخیر در تحریک با گام کامل

- **برای مطالعه:** در روش نیم گام، هر دور کامل یعنی چرخش ۳۶۰ درجه نیازمند ۴۰۰ مرتبه تحریک موتور پله‌ای است (تعداد تحریک‌های هدایت تک‌فاز ۲۰۰ مرتبه و تعداد تحریک‌های هدایت دو فاز نیز ۲۰۰ مرتبه می‌باشد). یعنی میزان تاخیر هر تحریک معادل ۳ میلی‌ثانیه است.

(تعداد پله در دور×۲× سرعت برحسب RPM) /۶۰ = میزان تاخیر در تحریک نیم گام

✓ **برنامه:**

```
#include <mega16.h>
#include <delay.h>
int i=0, k=0,num;
void main(void)
{
DDRD=0x0F;
ACSR=0x80;
SFIOR=0x00;
    for(i=1;i<=50;i++)
                {PORTD=0b00000001;
                delay_ms(6);num++;
                for(k=1;k<=3;k++)
                        { PORTD=PORTD<<1;
                        delay_ms(6);
                    num++;
                    if(num==51) k=4; }
                 }
}
```



شکل(۲-۱۳): مدار کنترل موتور پله‌ای ٤ فاز با استفاده از تراشه L298

- **نکته:** اگر بخواهیم سرعت موتور را روی مقدار نسبتاً زیاد تنظیم نمائیم، نمی‌توان از لحظه راه‌اندازی موتور، تاخیر بین تحریک‌های متوالی را بر اساس سرعت مطلوب نهایی محاسبه کرد چرا که به دلیل لختی زیاد، آرمیچر نمی‌تواند میدان مغناطیسی استاتور را دنبال نماید بنابراین باید به تدریج سرعت میدان استاتور (تحریک استاتور) را افزایش دهیم تا روتور بتواند میدان استاتور را دنبال کند و اصطلاحاً از حالت سنکرون خارج نشود.