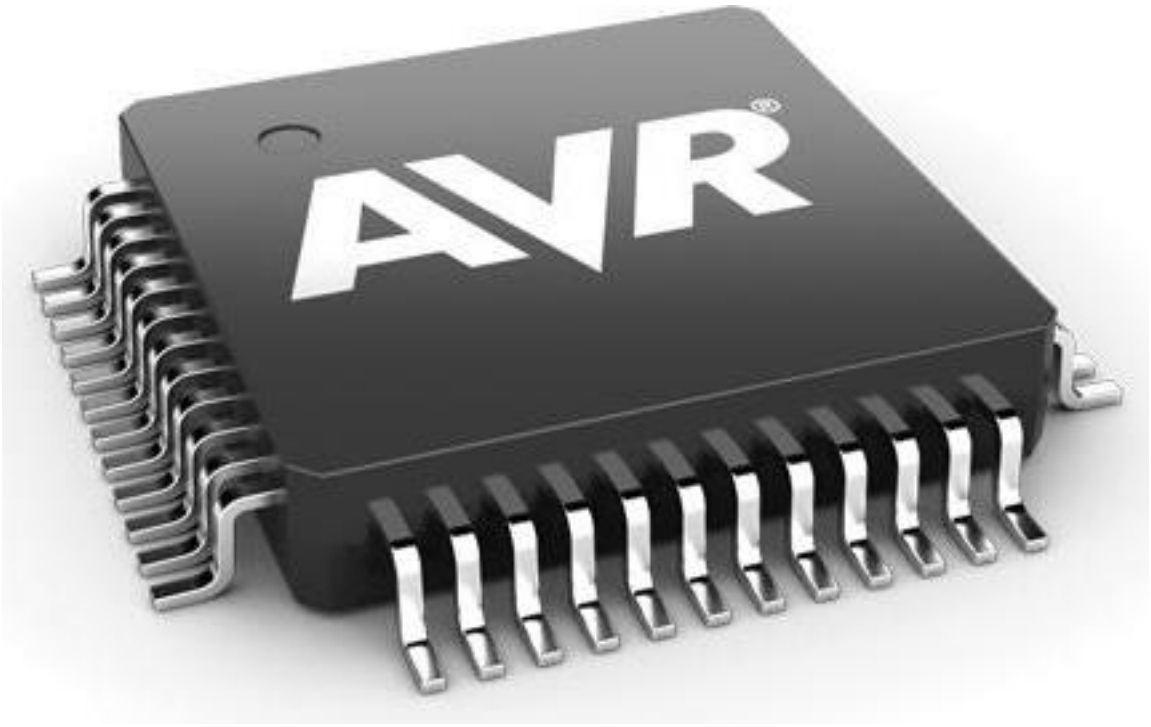
آموزش کاربردی میکروکنترلرهای AVR از 0 تا 100

ElectroVolt.ir

تقدیم به همه مشتاقان الکترونیک

بهترین و جامع ترین جزوه آموزش میکروکنترلر به زبان C



تهیه و تالیف : محمد حسین شجاع داودی

ناشر : وبسایت الکترو ولت ElectroVolt.ir

" ویرایش دوم "

شهریور 1394

سر فصل مطالب :

منابع و مراجع :

1. مرجع کامل میکروکنترلرهای AVR ، پرتوی فر ، مظاهریان ، بیانلو ، انتشارات نص
2. جزوه آموزش AVR مقدماتی و پیشرفته ، مهندس فتاحی ، مجتمع فنی تهران
3. جزوه آموزش برنامه نویسی C ویژه میکروکنترلرها ، مهندس نجفی ، مجتمع فنی تهران
4. آشنایی با میکروکنترلرهای AVR و نرم افزار CodeVision ، مهندس امیر ره افروز
5. آموزش سریع میکروکنترلرهای AVR ، مهندس رضا سپاس یار
6. جزوه میکروکنترلرهای AVR ، دکتر پویان ، دانشگاه شاهد
7. دیتاشیت Atmega32 ، شرکت atmel

مقدمه مولف

با گسترش روز افزون علم الکترونیک و تولید متنوع محصولات دیجیتال و ورود آن به زندگی روزمره ، نیاز به یادگیری هر چه بیشتر در این حوزه برای کلیه افرادی که با آن سروکار دارند ضروری به نظر می رسد چرا که با درک عمیق تر و کسب علم و مهارت بیشتر میتوان به سطوح بالایی از توانایی تولید و خلق کاربردهای جدید دیجیتالی دست یافت. برای روشن شدن قضیه بگذارید سوالی مطرح کنم . آیا می دانستید که پردازنده به کار رفته در فضاپیما آپولو 11 در سال 1969 ، هزار برابر ضعیف تر و کندتر از پردازنده گوشی های تلفن همراه هوشمند امروزی است ؟!

بله به راستی که با هر یک از پردازنده های موجود در گوشی های هوشمند امروزی میتوان کارهای فوق العاده ای علاوه بر کارهایی که شما بلد هستید انجام داد که شما احتمالا آنها را بلد نیستید. هر چه اساتید ، دانشجویان ، دانش آموزان برنامه نویسان ، علاقه مندان و کلیه کسانی که با طراحی ، ساخت و ارتقای سخت افزاری یا نرم افزاری محصولات الکترونیکی سروکار دارند ، از نحوه آن بیشتر بدانند ، هم بهتر میتوانند از انواع امکانات ارائه شده در آن وسیله استفاده کرده و طرز کارکرد آن را نیز درک کنند و هم میتوانند خلاقیت داشته باشند و آثار متنوع و بزرگ تری را خلق کنند.

امروزه از میکروکنترلرهای ARM بیشترین استفاده را به علت یکپارچه کردن سیستم های کنترلی با سرعت پردازش بالا ، توان مصرفی کم ، قیمت ارزان تر و حجم کمتر می شود. به طوری که امروزه هسته‌های پردازنده ARM به عنوان رایج‌ترین پردازنده 32 بیتی با سرعت پردازش چند مگاهرتز تا چند گیگاهرتز در طیف وسیعی از سیستم‌های نهفته و قابل حمل مورد استفاده قرار می‌گیرند. به طوری که امروزه اغلب تلویزیون ها ، تلفن‌های هوشمند ، تبلت ها ، خودروها و ... از این هسته پردازشی بهره می‌برند.

برای یادگیری و درک این میکروکنترلرها بهتر است از یادگیری نوع ساده میکروکنترلرها یعنی AVR شروع به کار کرد. همانطور که برای یادگیری مثلا توابع مثلثاتی ابتدا لازم است جدول ضرب و سپس هندسه و پیش نیازهای آن را بلد باشیم ، برای یادگیری ARM نیز باید پیش نیازهای آن شامل اصول الکترونیک دیجیتال ، اجزای مدارهای الکترونیکی ، زبان برنامه نویسی و... را بدانیم. بدون دانستن این اصول اولیه هرگز موفق به آپولو هوا کردن نخواهیم شد!

خوشبختانه منابع گسترده و وسیعی در اینترنت و کتابخانه ها برای یادگیری میکروکنترلرهای AVR وجود دارد که بسیار در این زمینه کمک میکنند اما به علت اینکه اغلب با نگاه سطحی یا بسیار عمیق گفته شده اند ، یا جوابگوی کلیه نیازهای علاقه مندان و مشتاقان یادگیری نیست و یا فقط قشر خاصی توانایی درک و هضم آن را دارند. جزوه موجود که با نگاهی نو برای مخاطب عام و با زبانی ساده بیان شده است ، امید است بتواند به طور عمیق و مفهومی خوانندگان را به این حوزه جلب کند. ضمنا ویژگی دیگر این جزوه از 0 تا 100 بودن آن است یعنی سعی شده است

تمام قواعد و قوانین مورد نیاز برای کار ، از رنگ مقاومت ها گرفته تا فرمول های مورد نیاز برای طراحی پروژه های میکروکنترلری ، آورده شده باشد تا در هر زمان و در هر مکان برای کار با میکرو کنترلرهای AVR فقط به یک جزوه pdf شده نیاز داشته باشید و لاغیر !

ضمنا کلیه آموزش های این جزوه در اینترنت و در سایت الکترو ولت به صورت بخش به بخش آورده شده است که میتوانید در هر زمان و در هر مکانی که نیاز داشتید و این pdf در اختیار نبود ، از آن استفاده نمایید.

قطعا تهیه و تدوین چنین جزوه ای هر چند که با تلاش های بسیار هم صورت گرفته باشد ، خالی از اشکال علمی یا ویرایشی نبوده و نیست. لذا کلیه نظرات ، پیشنهادات و انتقادات شما را از طریق ایمیل زیر با جان و دل پذیرایم.

ایمیل : hosainshoja@yahoo.com

نحوه مطالعه این جزوه

این جزوه هم برای آن دسته از علاقه مندانی که تاکنون هیچ زمینه آشنایی و کار با قطعات الکترونیکی نداشتند و هم برای دانشجویان ، برنامه نویسان و مهندسان الکترونیک در هر سطحی کارایی دارد. اگر تاکنون با میکروکنترلرها آشنایی نداشتید و یا احساس میکنید در مباحث پایه ای و مفهومی مشکل دارید از ابتدای جزوه ، از فصل اول با ما همراه باشید اما اگر چنانچه احساس میکنید خیلی مشکل نخواهید داشت باز هم پیشنهاد میکنیم از ابتدای فصل اول با ما همراه باشید ولی اگر خیلی دیگه اصرار داشتید و توانایی خودتون را در زمینه داشتن همه پیشنیازها و خصوصا برنامه نویسی C بالا می بینید ، به سراغ فصل هفتم و هشتم یعنی اصل کار بروید.

نکته مهم : مطالب این جزوه به صورت کاملا پیوسته ، دقیق و بخش به بخش طراحی شده است و کلیه مطالب را به تدریج پوشش خواهد داد تا خواننده از پایه و رفته رفته همانند کلاس درس با مطالب آشنا شود پس عجله نداشته باشید و با حوصله از ابتدا شروع به خواندن و سپس تجزیه و تحلیل ذهنی نمایید و تا تسلط بر بخش قبلی به بخش بعدی نروید.

تذکر مهم : علاوه بر خواندن و تجزیه و تحلیل ذهنی که برای فهم موضوع مورد نیاز است ، برای تسلط بر موضوع نیاز به کار کردن عملی خودتان در کنار این جزوه می باشد. بنابراین با خرید قطعات و لوازم مورد نیاز کار که در ادامه با آنها آشنا می شوید ، کلیه مثال ها را چندین بار مطالعه و برخی از آنها را خودتان پیاده سازی کنید (یعنی با فراموش

کردن جواب سوال فرض کنید چنین سوالی وجود دارد و از صفر خودتان شروع به طراحی ، شبیه سازی و نهایتا پیاده سازی نمایید) تا کاملا بر موضوع مسلط شوید.

فصل های اول و دوم فصل های مهمی هستند که اصول ابتدایی هر کار الکترونیکی می باشند و شما می بایست آنها را کاملا مسلط باشید چون در فصل های بعدی دیگر به آنها اشاره نخواهد شد.

فصل سوم قطعا ذهن شما را به چالش خواهد کشید و نسبت به کلیت موضوع میکروکنترلرها آشنا خواهد کرد و در آینده از نظر درک و مفهومی خیلی کمک خواهد کرد.

فصل چهارم کمی در نگاه اول ممکن است دشوار و حتی غیر ضروری به نظر آید اما توصیه میکنم بارها و بارها آن را مطالعه و نحوه عملکرد یک میکروکنترلر را مسلط شوید چون در بخش های بعدی آموزش کمکتان خواهد کرد.

فصل پنجم ، ششم و هفتم مهمترین فصل های میکروکنترلرهای AVR هستند که پایه حساب میشوند و در تمامی پروژه ها به مطالب آموخته شده در این فصول نیاز خواهید داشت. مثال های این بخش بسیار مهم بوده و توصیه به یادگیری عمیق می گردد.

و اما فصل هشتم که نقطه عطف این جزوه با سایر آموزش ها می باشد و آن هم توضیحات ساده و بر اساس شکل به همراه توضیحات سخت افزار مورد نظر و تنظیمات کدویزارد است که باعث درک بیشتر عملکرد میکروکنترلر و ساده تر شدن برنامه نویسی می گردد. مثال های عملی این بخش را نیز دانلود و بررسی کرده و ساده از آنها عبور نکنید.

آماده اید ؟؟؟ !

فصل اول : آشنایی با اصول و اجزای مدارهای الکتریکی و الکترونیکی

مقدمه :

مدارها در حالت کلی به دو دسته مدارهای **الکتریکی** و مدارهای **الکترونیکی** تقسیم می‌شوند. مدارهای الکتریکی از به‌هم پیوستن المان‌های الکتریکی (مقاومت، خازن، سلف، لامپ، و ...) تشکیل می شوند. مدارهای الکترونیکی از بهم پیوستن المانهای الکتریکی یا المانهای الکترونیکی (دیود، ترانزیستور، IC، و ...) یا ترکیبی از آن دو بوجود می‌آید به طوری که حداقل یک مسیر بسته را ایجاد کنند و جریان الکتریکی بتواند در این مسیر بسته جاری شود. در مدارهای الکتریکی محیط حرکت الکترون و به طور کلی جنس تشکیل دهنده اجزای مدار به هیچ عنوان اهمیت ندارند ، بلکه رابطه ریاضی بین ولتاژ و جریان این اجزای الکتریکی مهم هستند. مثل مقاومت و رابطه معروف $V=RI$ برای آن . در حقیقت ، در تحلیل این مدارها کمتر به ساختمان این قطعات توجه می‌شود. در مدارهای الکترونیکی برعکس مدارهای الکتریکی ، علاوه بر رابطه ریاضی ولتاژ و جریان قطعه ، به محیط عبور الکترون توجه کرده و در کل این جنس و نحوه ساخت اجزا است که خیلی اهمیت دارد. در تحلیل برخی از مدارهای الکترونیکی چون معادلات دیفرانسیل بسیار سخت و پیچیده ایجاد می‌شوند ، غالبا از تقریب برای قسمت‌های الکترونیکی استفاده می‌شود. مدارهای الکترونیکی خود به دو دسته **دیجیتال** (رقمی) و **آنالوگ** (قیاسی) تقسیم می‌شوند .

مدارهای الکترونیکی برای ایفا کردن وظایف مختلفی استفاده می‌شوند. کاربردهای اصلی مدارهای الکترونیکی عبارتند از:

1. کنترل و پردازش داده‌ها
2. تبدیل و توزیع توان الکتریکی
3. اجرای عملیات خاص

هر ردیف این کاربردها با ایجاد و آشکارسازی میدان الکترومغناطیسی و جریان الکتریکی سرو کار دارند. گرچه از انرژی الکتریکی در سال‌های انتهایی قرن ۱۹ برای انتقال پیام به وسیله تلگراف و تلفن استفاده می‌شد اما بیشتر پیشرفت‌های مربوط به علم الکترونیک پس از ساخت رادیو شکل گرفت. در یک نگاه ساده، یک سیستم الکترونیکی را می‌توان به سه بخش تقسیم کرد:

- ورودی: حسگرهای الکترونیکی و مکانیکی (یا مبدل‌های انرژی). این تجهیزات سیگنال‌ها یا اطلاعات را از محیط خارج دریافت کرده و سپس آنها را به جریان، ولتاژ یا سیگنال‌های دیجیتال تبدیل می‌کنند.

- پردازشگر سیگنال: این مدارها در واقع وظیفه اداره کردن، تفسیر کردن و تبدیل سیگنال‌های ورودی برای استفاده آنها در کاربرد مناسب را بر عهده دارند. معمولاً در این بخش پردازش سیگنال‌های مرکب بر عهده پردازشگر سیگنال‌های دیجیتال است.
- خروجی: فعال کننده‌ها یا دیگر تجهیزات (مانند مبدل‌های انرژی) که سیگنال‌های ولتاژ یا جریان را به صورت خروجی مناسب در خواهند آورد (برای مثال با ایفای یک وظیفه فیزیکی مانند چرخاندن یک موتور).

برای مثال یک تلویزیون دارای هر سه بخش بالا است. ورودی تلویزیون سیگنال‌های پراکنده شده را دریافت کرده (به وسیله یک آنتن یا کابل) و آنها را به ولتاژ و جریان مناسب برای کار دیگر تجهیزات تبدیل می‌کند. پردازشگر سیگنال پس از دریافت داده‌ها از ورودی اطلاعات مورد نیاز مانند میزان روشنایی، رنگ و صدا را از آن استخراج می‌کند. در نهایت قسمت خروجی این اطلاعات را دوباره به صورت فیزیکی در خواهد آورد این کار به وسیله یک لامپ اشعه کاتدیک و یک بلندگوی آهنربایی انجام خواهد شد.

آشنایی با اجزای مدارهای الکتریکی و الکترونیکی

در این قسمت به آشنایی با ابزارها ، المان های مدار اصول اولیه و پیشنیازهای مورد نیاز برای کار با مدارهای دیجیتال و میکروکنترلرها خواهیم پرداخت.

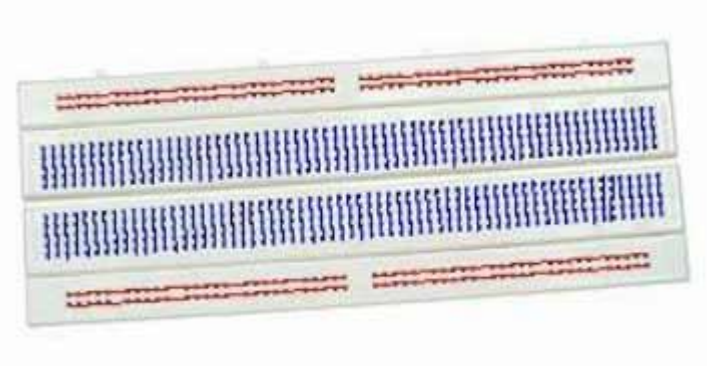
۱ – آشنایی با بردبورد :

بردبورد (Breadboard) وسیله‌ای است که به ما در چیدمان اولیه و آزمایشی مدار کمک می‌کند. بیشتر افرادی که در زمینه پروژه‌های الکترونیک کار می‌کنند ، ابتدا مدار خود را بر روی بردبورد می‌بندند و پس از جواب گرفتن آنرا بر روی مدارات چاپی یا بردهای سوراخ‌دار مسی پیاده می‌کنند.

لایه های داخلی برد بورد از نوارهای فلزی)معمولا مسی) تشکیل شده است که در لایه تحتانی و بدون هیچ اتصالی با یکدیگر در پایین بورد قرار دارند. توسط حفره های پلاستیکی این لایه های فلزی تا بالای بورد هدایت شده اندو این ما را قادر می سازد تا اجزای الکترونیکی را به یکدیگر متصل کنیم.

برای استفاده از بردبورد کافیست پایه های قطعات را درون شکاف مورد نظرفرو بریم (به این شکافها اصطلاحا سوکت میگویند.) و این سوکتها طوری طراحی شده اند که قطعات را کاملا محکم در خود بگیرند و هر حفره یا همان سوکت پایه قطعه را به لایه مسی تحتانی متصل می کند.
هر سیم که وارد این حفره ها می شود گره یا node نامیده میشود و هر گره را نقطه ای از مدار می نامند که حداقل باعث متصل شدن دو قطعه به یکدیگر شده است.
نکته : در بردبورد وقتی می خواهیم بین دویا چند قطعه اتصال الکترونیکی برقرار کنیم باید یکی از پایه هایشان با هم تشکیل گره بدهند. برای این کار کافیست پایه آنها را در حفره هایی که همگی در راستای لایه مسی مشترکی هستند قرار دهیم. برای اتصال دو نقطه دور از هم باید از سیم های مفتولی استفاده کرد که دو سر آن به کمک سیم چین لخت شده باشد.

در بردبوردهایی که در ایران معمول شده ، دو بخش قرینه هم داریم که شکاف میان دو بخش محلی برای جا زدن آی سی ها است .در دو طرف شکاف میانی شاهد بخشی هستیم که در آن هر پنج سوراخ که در راستای عمودی در یک امتداد هستند، به هم وصل شده اند و در حکم یک گره هستند. در بخش های بیرونی تر بردبورد ، هر طرف دو ردیف افقی داریم که تا میانه راه به هم وصل هستند. یعنی در هر یک از دو طول بردبورد ، چهار ردیف افقی بیست و پنج سوراخی به هم متصل هستند. از این لایه ها بیشتر برای اتصال منابع ولتاژ استفاده می شود.
در شکل زیر ردیفهایی که به هم وصل هستند با خطوط رنگی نشان داده ایم. هر خط قرمز یا آبی با اینکه شامل چند سوکت است ولی فقط یک گره می باشد.

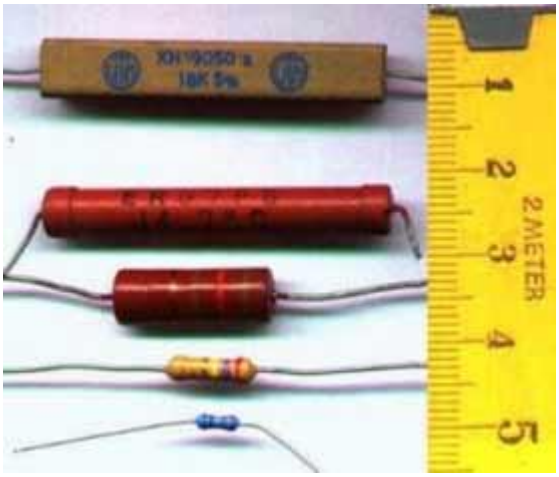


۲ – آشنایی با مقاومت الکتریکی:

مقاومت پرکاربردترین قطعه در مدارهای الکترونیکی است.

زمانی که جریان الکتریکی از داخل مقاومت عبور می‌کند با مانع مواجه می‌شود.

نکته : مقدار مقاومت وابسته به مدار نیست و فقط به جنس و شکل ماده مقاوم بستگی دارد.



مقاومتها ممکن است که ثابت یا متغییر باشند. مقاومتهای متغیر **پتانسیومتر** نیز خوانده می‌شوند.

تشخیص مقدار مقاومت با استفاده از نوارهای رنگی :

مقاومتهای توان کم دارای ابعاد کوچک هستند ، به همین دلیل مقدار مقاومت و خطا را نمی توان روی آنها نوشت ، در نتیجه بوسیله نوارهای رنگی روی آن مشخص می کنند.

این روش به دو شکل صورت می‌گیرد:

۱- روش چهار نواری

۲- روش پنج نواری

روش اول برای مقاومتهای با خطای ٪۵ به بالا استفاده می‌شود و روش دوم برای مقاومتهای دقیق و خیلی دقیق (با خطای کمتر از ٪۵) استفاده می‌شود.

در اینجا به روش اول که معمول تر است می‌پردازیم.

به جدول زیر توجه نمائید. هر کدام از این رنگها معرف یک عدد هستند:

سیاه	0
قهوه‌ای	1
قرمز	2
نارنجی	3
زرد	4
سبز	5
آبی	6
بنفش	7
خاکستری	8
سفید	9

دو رنگ دیگر هم روی مقاومتها به چشم می‌خورد: طلایی و نقره‌ای ، که روی یک مقاومت یا فقط طلایی وجود دارد یا نقره‌ای.اگر یک سر مقاومت به رنگ طلایی یا نقره‌ای بود ، ما از طرف دیگر مقاومت ، شروع به خواندن رنگها می‌کنیم. و عدد متناظر با رنگ اول را یادداشت می‌کنیم. سپس عدد متناظر با رنگ دوم را کنار عدد اول می‌نویسیم .سپس به رنگ سوم دقت می‌کنیم. عدد معادل آنرا یافته و به تعداد آن عدد ، صفر می‌گذاریم جلوی دو عدد قبلی(در واقع رنگ سوم معرف ضریب است). عدد بدست آمده ، مقدار مقاومت برحسب اهم است. که آنرا می‌توان به کیلواهم نیز تبدیل کرد.

ساخت هر مقاومت با خطا همراه است. یعنی ممکن است ٪۵ یا ٪۱۰ یا ٪۲۰خطا داشته باشیم . اگر یک طرف مقاومت به رنگ **طلایی** بود ، نشان دهنده مقاومتی با خطا یا تولرانس **۵** ٪ است و اگر **نقره‌ای** بود نمایانگر مقاومتی با خطای ٪**۱۰** است.اما اگر مقاومتی **فاقد** نوار چهارم بود، بی رنگ محسوب شده و تولرانس آن را **۲۰** ٪در نظر می‌گیریم.

سوال : مقدار واقعی مقاومت زیر در چه بازه ای قرار دارد؟

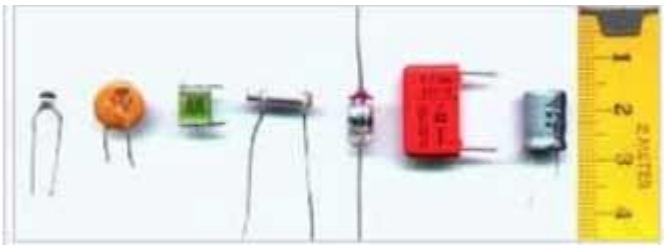
جواب:

از سمت چپ شروع به خواندن می‌کنیم. رنگ زرد معادل عدد ۴ ، رنگ بنفش معادل عدد ۷ ، رنگ قرمز معادل عدد ۲ ، و رنگ طلایی معادل تولرانس ٪۵ می‌باشد. پس مقدار مقاومت بدون در نظر گرفتن تولرانس ،

مساوی ۴۷۰۰ اهم ، یا ۴،۷ کیلو اهم است و برای محاسبه خطا عدد۴۷۰۰ را ضربدر ۵ و تقسیم بر ۱۰۰ می‌کنیم، که بدست می‌آید: ۲۳۵ . پس مقدار واقعی مقاومت چیزی بین ۴۴۶۵ اهم تا ۴۹۳۵ اهم می‌باشد.

۳ – آشنایی با خازن :

خازن یک المان الکتریکی است که می‌تواند انرژی الکتریکی را در خود ذخیره کند. هر خازنی که توانایی ذخیره انرژی الکتریکی بالاتری داشته باشد ، ظرفیت بالاتری دارد. واحد اندازه گیری ظرفیت خازن "فاراد" می باشد. ظرفیت خازن را با حرف C که ابتدای کلمه capacitor است نمایش می‌دهند.



ساختمان داخلی خازن از دوقسمت اصلی تشکیل می‌شود:

الف – صفحات رسانا

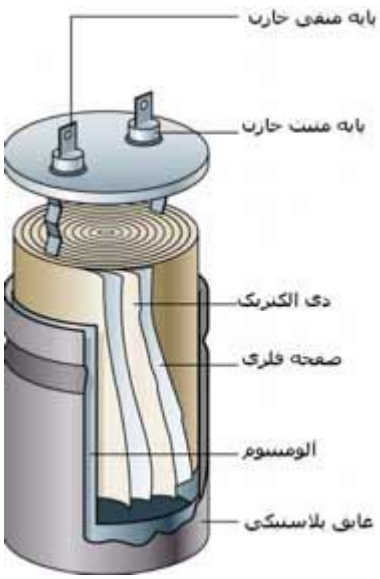
ب – عایق بین رساناها (دی الکتریک)

بنابراین هرگاه دو رسانا در مقابل هم قرار گرفته و در بین آنها عایقی قرار داده شود ، تشکیل خازن می دهند.

معمولا صفحات رسانای خازن از جنس آلومینیوم ، روی و نقره با سطح نسبتا زیاد بوده و در بین آنها عایقی (دی الکتریک) از جنس هوا ، کاغذ ، میکا ، پلاستیک ، سرامیک ، اکسید آلومینیوم و اکسید تانتالیوم استفاده می شود.

هر چه ضریب دی الکتریک یک ماده عایق بزرگتر باشد آن دی الکتریک دارای خاصیت عایقی بهتر است.

ساختار داخلی یک خازن را در شکل زیر مشاهده می کنید.



خازن ها دو نوع هستند : ثابت و متغیر

خازن های ثابت :

این خازن ها دارای ظرفیت معینی هستند که در وضعیت معمولی تغییر پیدا نمی کنند. خازن های ثابت را بر اساس نوع ماده دی الکتریک به کار رفته در آنها تقسیم بندی و نام گذاری می کنند و از آنها در مصارف مختلف استفاده می شود . از جمله این خازنها می توان انواع سرامیکی ، میکا ، ورقه ای ، الکترولیتی و ... را نام برد . اگر ماده دی الکتریک طی یک فعالیت شیمیایی تشکیل شده باشد آن را خازن الکترولیتی و در غیر این صورت آن را خازن خشک گویند.

اساس کار خازن های متغیر :

به طور کلی با تغییر سه عامل می توان ظرفیت خازن را تغییر داد ": فاصله صفحات" ، "سطح صفحات" و" نوع دی الکتریک ". اساس کار این خازن ها روی تغییر این سه عامل می باشد.

تشخیص ظرفیت خازن ها:

متداول ترین نوع خازن ها نوع سرامیکی (عدسی) و نوع الکترولیتی است.

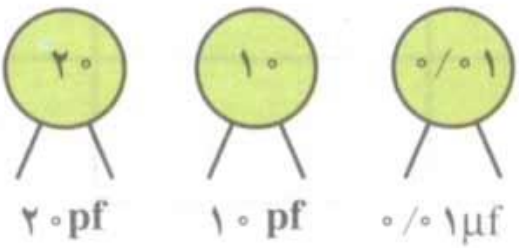
–ظرفیت نوع عدسی معمولا در حدود پیکوفاراد است (پیکو یعنی ۱۰ به توان منفی دوازده) . در این خازن ها پایه مثبت و منفی فرقی نمیکند.

ظرفیت نوع الکترولیتی معمولا در حدود میکروفاراد است (میکرو یعنی ۱۰ به توان منفی شش).

نکته مهم : در خازن های الکترولیتی همانند شکل زیر پایه مثبت بلندتر است و پایه منفی با نوار مشخص شده است . پایه مثبت حتما می بایست به ولتاژ مثبت تر وصل شود در غیر این صورت خازن میسوزد و خطرناک است.

بر روی خازن های الکترولیتی معمولا مقدار ظرفیت دقیق نوشته می شود. پایه منفی هم با یک نوار سفید رنگ که روی آن علامت منفی وجود دارد مشخص می گردد (توجه به مثبت و منفی بودن پایه ها در خازن الکترولیتی مهم است ولی در عدسی نه)

تشخیص مقدار ظرفیت در خازن های عدسی کمی متفاوت است. در اغلب مواقع واحد ظرفیت بر روی بدنه ی خازن قید نمی شود. در این صورت چنان چه این عدد از یک کوچکتر باشد ظرفیت بر حسب میکرو فاراد و چنان چه عدد بزرگتر از یک باشد ظرفیت بر حسب پیکوفاراد است. شکل زیر را ببینید:



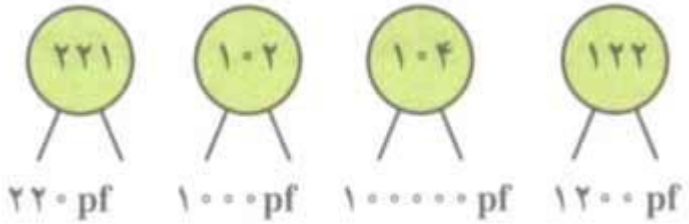
در حالتی که بر روی خازنی اعداد یک رقمی یا دو رقمی مشاهده گردید، مقدار واقعی ظرفیت ، همان عددی است که بر روی آن نوشته شده و واحد آن پیکو فاراد است. اما اگر عدد سه‌رقمی بود، در این حالت اگر آخرین رقم صفر بود، به همان ترتیب بالا عمل می‌کنیم و مقدار ظرفیت همان عدد است. اما اگر آخرین رقم عدد دیگری غیر از صفر بود به این ترتیب عمل می کنیم:

اولین رقم را رقم اول ، دومین رقم را رقم دوم و سومین رقم را تعداد صفر قرار خواهیم داد و واحد را نیز همان پیکوفاراد می‌گیریم . مقدار بدست آمده را می‌توان به واحدهای دیگر تبدیل نمود.

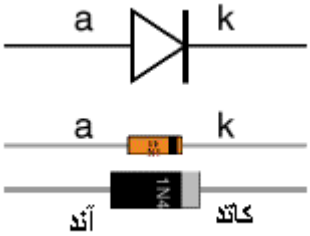
به عنوان مثال:

ظرفیت خازنی که روی آن نوشته شده۵۰۳ برابر است با ۵۰۰۰۰ پیکوفاراد= ۵۰ نانوفاراد = ۰/۰۵ میکروفاراد.

همچنین به مثال زیر توجه کنید:



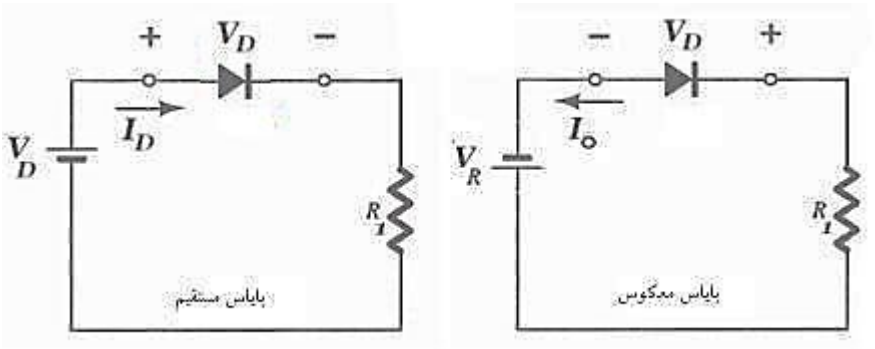
۴ – آشنایی با دیود ها:



دیود (Diode) قطعه‌ای نیمه رسانا از جنس سیلیسیم یا ژرمانیم است که جریان الکتریکی را در یک جهت از خود عبور می‌دهد و در جهت دیگر در مقابل عبور جریان از خود مقاومت بسیار بالایی نشان می‌دهد. مهمترین کاربرد دیود عبور دادن جریان در یک جهت و ممانعت در برابر عبور جریان در جهت مخالف است. در نتیجه می‌توان به دیود مثل یک شیر الکتریکی یک طرفه نگاه کرد. از این ویژگی دیود برای تبدیل جریان متناوب به جریان مستقیم استفاده می شود. سیلیسیوم (Si) و ژرمانیوم (Ge) دو نیمه رسانای معروف هستند که هر کدام ۴ الکترون ظرفیت دارند. اگر آن ها را با عناصری که اتم‌هایشان ۵ الکترون ظرفیت دارند، نظیر ارسنیک (As) یا فسفر (P) ترکیب کنیم ، نیمه رسانای نوع n تولید می شود و اگر آن ها را با عناصری که اتم هایشان ۳ الکترون ظرفیت دارند ، نظیر بور (B) یا

آلومینیوم (Al) ترکیب کنیم ، نیمه رسانای نوع p تولید می شود. دیود در اثر اتصال یک نیمه رسانای نوع n به یک نیمه رسانای نوع p (پیوندp-n) حاصل می شود.

از لحاظ الکتریکی یک دیود هنگامی جریان را از خود عبور می دهد که شما با برقرار کردن ولتاژ در جهت درست (+ به آند و –به کاتد) آنرا آماده به کار کنید. مقدار ولتاژی که باعث می‌شود تا دیود شروع به هدایت جریان الکتریکی نماید ولتاژ آستانه هدایت نامیده می‌شود که چیزی حدود 0.6 تا 0.7 ولت می‌باشد. اما هنگامی که شما ولتاژ معکوس به دیود متصل می‌کنید (+ به کاتد و – به آند) جریانی از دیود عبور نمی‌کند ، مگر جریان بسیار کمی که به جریان نشتی معروف است که در حدود چند μA یا حتی کمتر می‌باشد. این مقدار جریان معمولاً در اغلب مدارهای الکترونیکی قابل صرفنظر کردن بوده و تأثیری در رفتار سایر المانهای مدار نمی‌گذارد. بنابراین در حالت **ایده آل** میتوان دیود را در حالت **بایاس معکوس** ، **مدار باز** و در حالت **بایاس مستقیم** ، **اتصال کوتاه** در نظر گرفت.



۵ -آشنایی با : LED

LED ها ظاهرا به شکل لامپ های کوچکی هستند که با برقراری جریان مستقیم در آنها ، نور تولید می کنند . ولی در واقع ساختار آنها شباهتی به لامپهای رشته ای ندارد.

LEDها دو پایه دارند ، یکی منفی و یکی مثبت. پایه بلندتر مثبت است که باید به ولتاژ مثبت تر متصل شود تا LEDروشن شود . در غیر این صورت روشن نخواهد شد.

LEDها از خانواده دیود ها هستند و چون با عبور جریان از آنها ، نور تولید می شود ، در مدارات الکترونیکی کاربرد زیادی دارند.

بعضی وقتها هدف مدار به نحوی روشن کردن LED ها می باشد ولی گاهی نقش آنها صرفا نمایش عبور جریان از یک شاخه است و معمولا برای روشن شدن کامل به حداقل ولتاژ ۳ ولت مستقیم احتیاج دارند.

نکته ۱ : هیچ گاه LED را بدون اینکه با یک مقاومت مناسب سری کرده باشید ، مستقیما به منبع ولتاژ وصل نکنید زیرا باعث کاهش طول عمر و سوختن آن میشود.

نکته ۲ : برای اینکه LED نوردهی مناسبی داشته باشد ، بهترین جریان گذرنده از آن ۲۰ میلی آمپر است و در این حالت بسته به منبع تغذیه ، افت ولتاژی که دو سر LED می افتد بین ۲،۲ تا ۳ ولت میتواند باشد که ما مقدار این افت ولتاژ را ۲،۵ ولت در نظر می گیریم.

بنابراین با در نظر گرفتن نکته فوق و از رابطه V=RI (قانون اهم) میتوان مقدار مقاومت مناسب را برای ولتاژ تغذیه ای که به آن متصل می شود ، محاسبه کرد.

۶ – آشنایی با منابع تغذیه:

برای راه اندازی مدارهای الکترونیکی در صورت امکان بهتر است از منابع تغذیه آزمایشگاهی استاندارد استفاده نمود که توانایی تولید جریان DC را با ولتاژهای مختلفی دارا هستند البته قیمت آن نسبتا زیاد است. اگر منبع تغذیه در اختیار نبود می توان از آداپتورهای موجود در بازار استفاده کرد که قیمت ارزان تری دارند. همچنین میتوان از باتری با جریان دهی مناسب و یا از پورت USB کامپیوتر نیز استفاده نمود.

آداپتور دستگاهی است که ولتاژ متناوب برق شهر را می گیرد و ولتاژ آن را کم می کند و سپس آن را یکسو می‌کند(یعنی به ولتاژ DC تبدیل می کند). ولتاژ برق شهر ۲۲۰ ولت است و چنانچه به بدن اتصال پیدا کند ، خطر مرگ دارد ولی ولتاژ خروج آداپتور در حدود ۳تا۱۲ ولت است و برای بدن هیچ ضرری ندارد و با آرامش می توانید با آن کار کنید. تمام قطعات الکتریکی ماشینها با ولتاژ ۱۲ولت کار می کنند که در صورت تماس با بدن انسان ، خطر جانی نداشته باشد.روی آداپتور ها اصولا این موارد را می نویسند :

Input : 220v AC 50/60Hz

به معنی ولتاژ برق شهر است

Output : 3-12 V DC

به معنی ولتاژ خروجی اداپتور است

1000mA

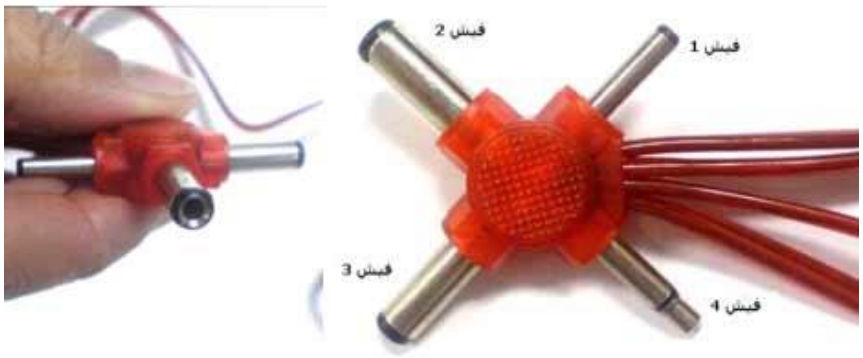
به معنی جریان خروجی آداپتور است که هر چقدر بیشتر باشد ، آداپتور قوی تر بوده و می تواند وسایل بزرگتری را تغذیه کند. برای مثال لامپ چشمک زن ماشین ۵۰۰ میلی آمپر جریان نیاز دارد ؛ لامپ های خطر ماشین ۱۸۰۰ میلی آمپر نیاز دارد و لامپهای چراغ جلو در حدود ۸۰۰۰ میلی آمپر جریان نیاز دارد . در نتیجه با یک آداپتور ۱۰۰۰ میلی آمپر حداکثر میتوان وسیله ای را که به ۱۰۰۰ میلی آمپر برای روشن شدن احتیاج دارد ، تغذیه کرد.

آداپتوری که در شکل نشان داده شده است ، استفاده های زیادی دارد. و به وسایل مختلفی نظیر ، ضبط ، رادیو ،

واکمن و ... وصل می شود (البته با ولتاژ مشخص) بنابراین روی سیم آن فیشهای مختلفی وصل شده که به انواع این وسایل وصل شوند.کلا دو نوع فیش داریم :

نوع اول – داخل آن مثبت و خارج آن منفی مانند فیش های ۱ و ۲ و ۳ در شکل زیر

نوع دوم – سر آن مثبت و بدنه آن منفی مانند فیش ۴ در شکل زیر

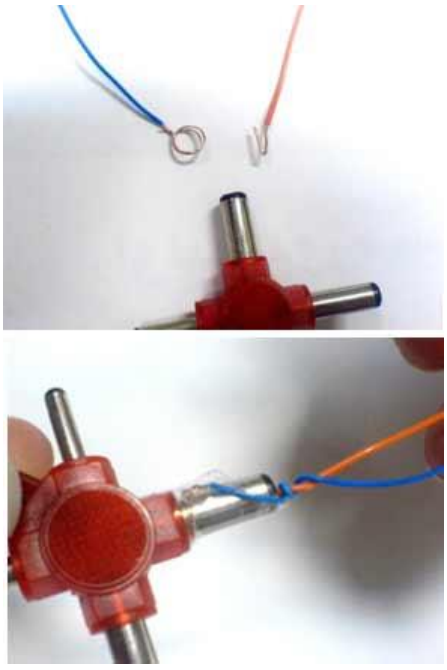


با اتصال سیمها به هر کدام از این فیش ها می توانیم مثبت و منفی آداپتور را از آن بگیریم.

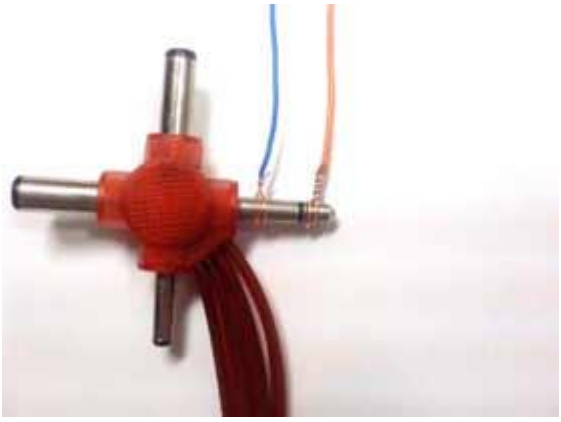
عموما برای اینکه به سادگی بتوانیم بین مثبت و منفی تمایز قائل شویم ، سیمهای تیره تر (آبی یا سیاه) را به منفی و سیمهای روشنتر (قرمز یا زرد) را به مثبت وصل می کنیم تا بتوانیم در یک نگاه مثبت و منفی را تشخیص دهیم.

سیمها را مطابق شکل به فیشها وصل می کنیم.

۱ – اتصال به فیش نوع اول



۲ – اتصال به فیش نوع دوم



به جای آداپتور میتوان از یک باتری کتابی ۹ ولت به همراه یک آی سی رگولاتور (برای تنظیم 5 ولت) استفاده کرد.

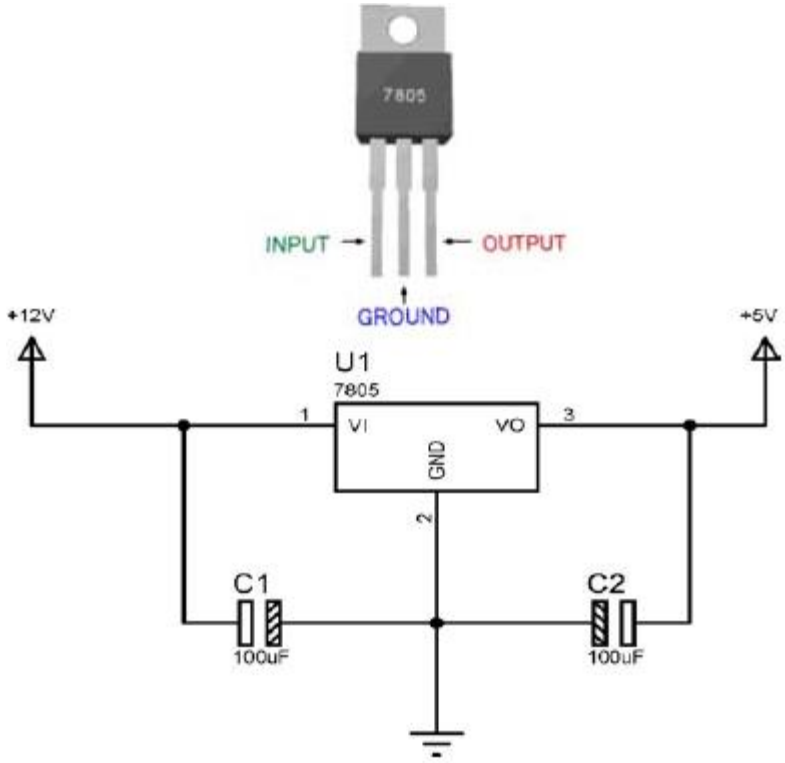
همچنین می توان از پورت USB نیز برای منبع تغذیه مدار استفاده کرد . در این حالت خروجی پورت USB ، ۵ ولت بوده و احتیاجی به رگولاتور نمی باشد . پورت USB دارای ۴ سیم به صورت شکل زیر است که از دو سیم آن برای تغذیه استفاده می شود.

Pin	Signal	Color	Description
1	VCC		+5V
2	D-		Data -
3	D+		Data +
4	GND		Ground

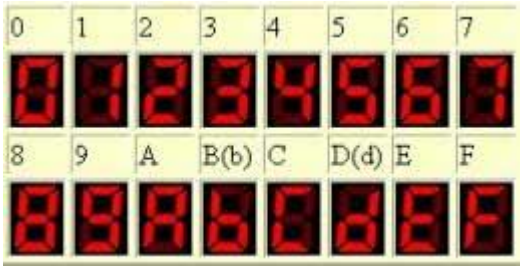
تذکر : هیچگاه نباید دو سر منبع تغذیه به مدت طولانی به هم وصل شود زیرا باعث سوختن آن می گردد.

7- رگولاتور یا تنظیم کننده ولتاژ:

در اکثر سیستم های الکتریکی و الکترونیکی نیازمند داشتن ولتاژ ورودی (منبع تغذیه) با ولتاژ ثابت هستیم اما با افزایش جریانی که مدار می کشد ، ولتاژ منبع تغذیه ورودی کاهش پیدا می کند . برای داشتن ولتاژی تنظیم شده و ثابت از المانی به نام رگولاتور (Regulator) که دارای سه پایه به صورت شکل زیر است استفاده میشود . در واقع این آی سی از نوسانات ولتاژی جلوگیری کرده و خروجی دقیق و تمیز به ما می دهد . نام گذاری این آی سی ها به صورت 78XXمی باشد که در آن ۲ رقم سمت راست که به صورت XX نشان داده شده نشان دهنده ولتاژ خروجی است. مثلا رگولاتور ۷۸۰۵ ، دارای ولتاژ خروجی ۵ ولت می باشد. برای اینکه این آی سی خروجی تمیز و دارای نوسان پایین بدهد معمولا ورودی آن را بیشتر از ۵ ولت (مثلا ۱۲ ولت) در نظر می گیرند.



8 – آشنایی با سون سگمنت :



سون سگمنت یک قطعه الکترونیکی است که برای نمایش اعداد و گاهی هم برای نمایش ساده تعدادی از حروف انگلیسی بکار می رود. حتی کسانی که با الکترونیک سر و کار ندارند هم این قطعه را بارها در وسایل الکترونیکی دیده اند. وسایلی مثل مایکروویو ، ترازوی دیجیتال ، آسانسور و...

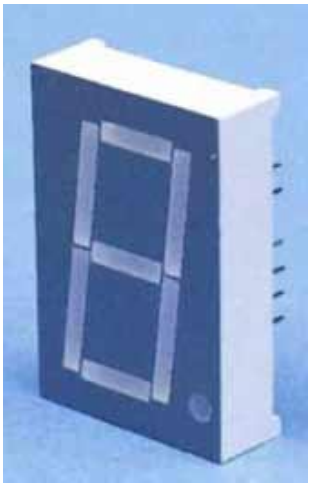
سون سگمنت در واقع ۷ تا LED است که یکی از پایه های آنها با هم یکی شده.

انواع سون سگمنت :

۱ – آند مشترک (در این نوع ، پایه مثبت همه LED ها یکی شده است)

۲ – کاتد مشترک (در این نوع ، پایه منفی همه LED ها یکی شده است)

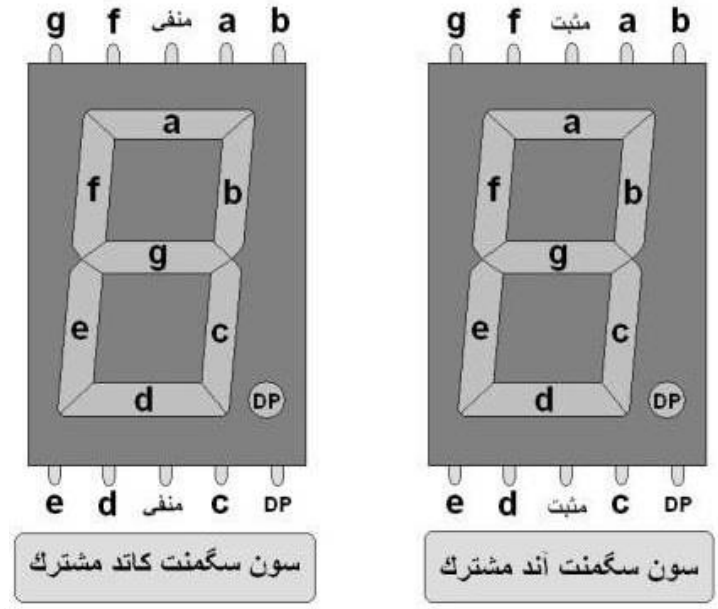
هر سون سگمنت به طور معمول ۱۰ پایه دارد .دو تای آنها مثل هم هستند و نقش پایه مشترک را دارند (استفاده از یکی از آنها کافی است) ۷ پایه هم به تک تک LED ها اختصاص دارد و یک پایه هم متعلق به نقطه ای است که در گوشه صفحه سگمنت قرار گرفته و اگر لازم باشد بعنوان ممیز از آن استفاده می شود. این نقطه هم در واقع یک LED دیگر است.



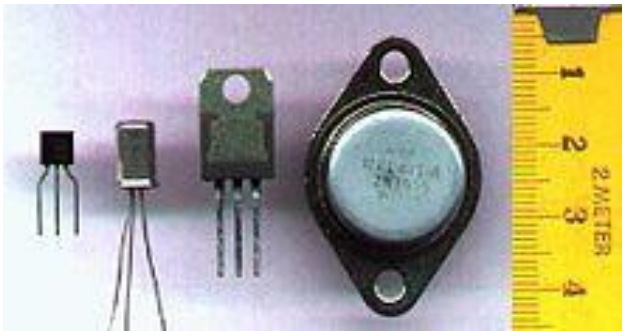
در سون سگمنت های آند مشترک ، پایه مشترک را به یک مقاومت مناسبی در حدود یک کیلو اهم (برای محدود کردن جریان) وصل کرده و سر دیگر مقاومت را به سر مثبت منبع تغذیه وصل می کنند. حال اگر هر کدام از پایه های دیگر به منفی منبع تغذیه متصل گردد ، LEDمربوط به آن پایه روشن می شود.

در سون سگمنت های کاتد مشترک هم همین قانون حاکم است فقط باید توجه نمود که پایه مشترک از طریق مقاومت به مثبت منبع تغذیه وصل شود و پایه های دیگر به منفی.

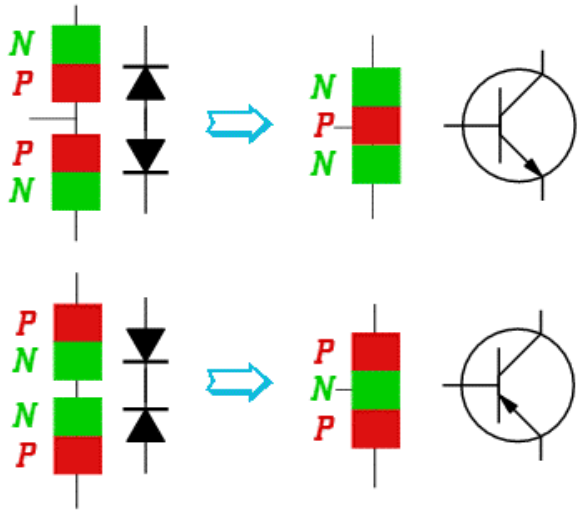
پایه های بکار رفته در سون سگمنت ها بصورت شکل زیر هستند



9 –آشنایی با ترانزیستور ها:

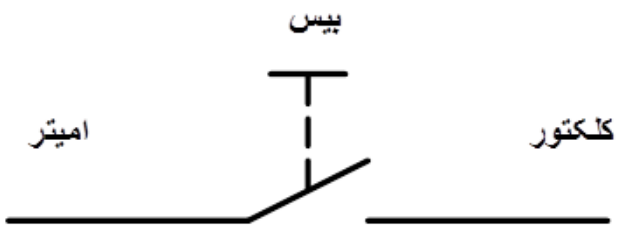


ترانزیستورها یکی از مهمترین و پرکاربردترین قطعات الکترونیکی می باشند که از پیوندهای مواد نیمه رسانا (مانند سیلیسیم و ژرمانیم) به صورت pnp یا npn تشکیل شده است .

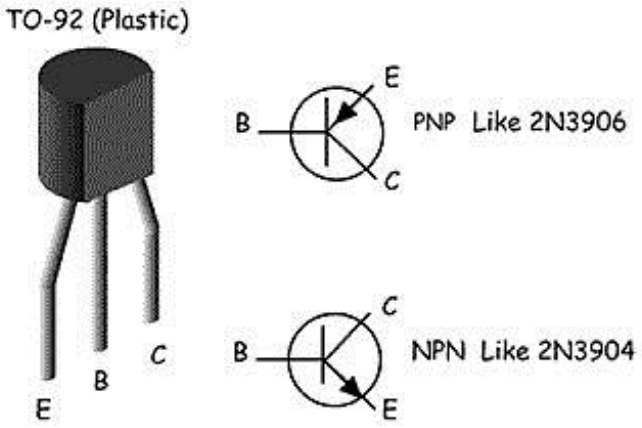


ترانزیستورها به دو دسته کلی تقسیم می‌شوند: ترانزیستورهای اتصال دوقطبی (BJT) و ترانزیستورهای اثر میدانی (FET) . از ترانزیستورهای bjt در دو کاربرد مختلف استفاده می شود . یکی به عنوان تقویت کننده جریان در مدارهای آنالوگ و دیگری به عنوان کلید قطع و وصل جریان در مدارهای دیجیتال .

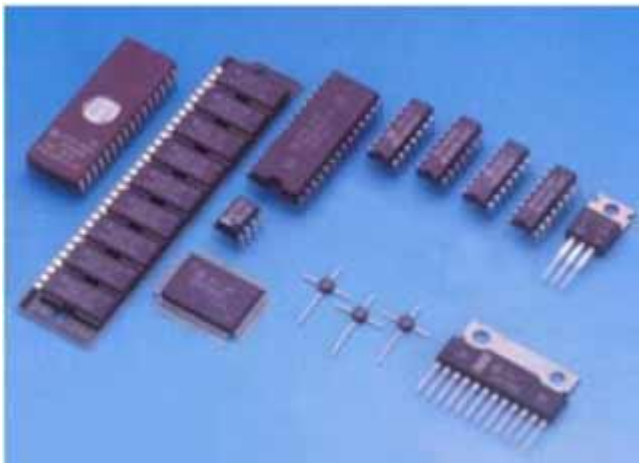
شکل زیر مدل یک ترانزیستور دو قطبی را هنگامی که بعنوان کلید در مدارهای دیجیتال استفاده می‌شود را نشان می دهد. در این حالت جریان گذرنده از کلکتور-امیتر توسط جریان بیس کنترل می شود و ترانزیستور bjt تنها بین دو حالت ON و Off کار می کند.



یک ترانزیستور BJT دارای سه پایه به نام های بیس (Base) ، کلکتور (Collector) و امیتر (Emitter) می باشد. همانطور که در شکل زیر نیز مشاهده می کنید ، در عمل وقتی از روبرو به قسمت صاف ترانزیستور نگاه کنید ، پایه سمت چپ امیتر ، پایه وسط بیس و پایه سمت راست کلکتور است.



10 – آی سی ها :



حروف اختصاری IC از دو کلمه انگلیسی integrated circuit به معنی مدار مجتمع گرفته شده است.

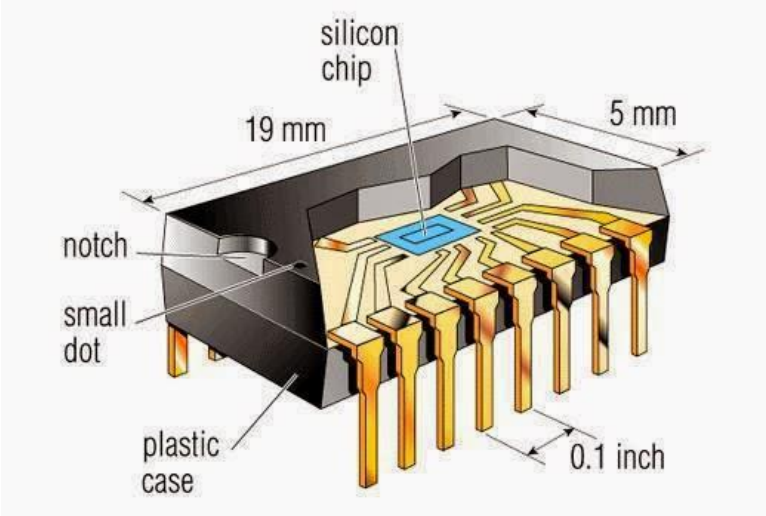
مدارهای دیجیتال با استفاده از آی سی که یک نیمه هادی از جنس سیلیکون است ، ساخته می شوند.

آی سی از قطعات الکترونیکی متعددی تشکیل شده است که از داخل به هم مرتبط هستند. این مجموعه "تراشه" یا "chip"نامیده می شود.

تراشه در داخل یک بسته سیاه رنگ قرار گرفته و ما هیچ گونه دسترسی به آن نداریم. اصولا نیازی هم نداریم که به داخل آی سی دسترسی داشته باشیم یا حتی بدانیم از چه قطعاتی تشکیل شده است.

ارتباط آی سی با فضای بیرون از طریق تعدادی پایه فلزی برقرار است که به آنها پین هم گفته می شود. اصولا ما با

ساختار داخل آی سی ها کاری نداریم و آنها را فقط به عنوان یک جعبه سیاه در نظر می گیریم. چیزی که برای ما مهم است و در طراحی و ساخت مدار باید به آن توجه کنیم ، فقط پایه ها هستند.

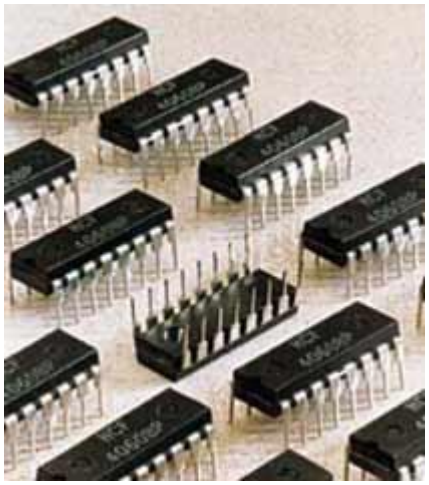


تعداد پایه ها ممکن است از ۸ پایه برای آی سی های کوچک تا ۶۴ پایه و بیشتر برای آی سی های بزرگ متغیر باشد.به منظور شناسائی هر آی سی ، روی آن شماره ای چاپ می شود و تولید کنندگان کتابچه هایی چاپ می کنند که اطلاعات مربوط به هر آی سی در آن وجود دارد. یک راه دیگر دستیابی به اطلاعات هر آی سی ، جستجو در اینترنت (خصوصا در سایت گوگل) است. بدین ترتیب می توان توضیحات و اطلاعات مربوط به هر آی سی را در متنی با عنوان Datasheet پیدا کرد.

چه انگیزه‌ای باعث اختراع ICشد؟

پیش از اختراع IC ،مدارهای الکترونیکی از تعداد زیادی قطعه یا المان الکتریکی تشکیل می‌شدند. این مدارات فضای زیادی را اشغال می‌کردند و توان الکتریکی بالایی نیز مصرف می‌کردند. و این موضوع ، امکان بوجود آمدن نقص و عیب در مدار را افزایش می‌داد. همچنین سرعت پایینی هم داشتند.

IC، تعداد زیادی عناصر الکتریکی را که بیشتر آنها ترانزیستور هستند، در یک فضای کوچک درون خود جای داده است و همین پدیده است که باعث شده امروزه دستگاه‌های الکترونیکی کاربرد چشمگیری در همه جا و در همه زمینه‌ها داشته باشند.



انواع آی سی :

آی سی ها در انواع و اندازه های متفاوتی در بازار موجود هستند .طراحان و سازندگان مدارات الکترونیکی با توجه به کاربرد و توانایی های هر آی سی ، دست به انتخاب می زنند.

خانواده های متفاوتی از نظر تکنولوژی ساخت آی سی در بازار موجودند که مشهورترین آنها خانواده TTL و خانواده CMOSهستند. این نامها بصورت مخفف متداول هستند و در اصل به این صورت می باشند:

TTL : Transistor-Transistor-Logic

CMOS : Complementary Metal-Oxide Semiconductors

ولتاژ تغذیه آی سی های TTL 5 ولت می باشد و ترانزیستورهای آن از نوع BJT بوده و منطق 0 یا 1 (ON و Off) دارند . ترانزیستورهای موجود در آی سی های خانواده CMOS ، از نوع FET بوده و بنابراین سریع تر و کم مصرف تر هستند و ولتاژ تغذیه این آی سی ها تا 15 ولت نیز می تواند باشد.

البته بسیاری از آی سی ها هم در بازار موجود هستند و استفاده از آنها متداول است، ولی در هیچ کدام از این دو خانواده قرار نمی گیرند.

فصل دوم : آشنایی با اصول اولیه الکترونیک دیجیتال

مقدمه

کلمه دیجیتال را تا به حال بارها شنیده اید. مخصوصا در این دوران که عصر تکنولوژی است.

سوال : خوب فکر کنید و ببینید در چه مواردی این کلمه را شنیده اید؟ چند وسیله دیجیتالی می توانید نام ببرید؟

جواب:

- گوشی های تلفن دیجیتال
- گوشی های موبایل
- کامپیوتر
- دوربین دیجیتال

تفاوت الکترونیک آنالوگ و دیجیتال:

الکترونیک آنالوگ به سیستم های الکترونیکی گفته می‌شود که با سیگنال‌های پیوسته کار می‌کنند. برخلاف سیستم های الکترونیک دیجیتال که فقط از دو حالت ۰ و ۱ استفاده می‌کنند. واژه "آنالوگ" به روابط نسبی ما بین یک سیگنال و یک ولتاژ یا یک جریان برق اشاره می‌کند.

الکترونیک دیجیتال شامل مدارت و سیستم هایی است که فقط دو حالت در آنها وجود داشته باشد.

سیستمی که بخواهد فقط دو حالت داشته باشد : مثل سوئیچهای باز و بسته یا لامپهای روشن و خاموش را سیستم دیجیتال گویند.

سیستمی که بر عکس مورد قبلی ، حالتهای زیاد و پیوسته ای دارد : مثل یک شیر آب که می تواند به مقدارهای مختلفی باز شود و دبی های مختلفی آب را از خودش عبور دهد. و فقط دو حالت باز و بسته ندارد. چنین سیستمی را سیستم آنالوگ یعنی پیوسته گویند .

گفتیم که در یک سیستم دیجیتال فقط دو حالت وجود دارد. این حالتها توسط دو سطح ولتاژ متفاوت نشان داده می شوند .ولتاژ سطح بالا (high) و ولتاژ سطح پایین. (low)

در سیستم دیجیتال ، ترکیبی از این دو حالت یک "کد" نامیده می شود و برای نمایش اعداد ، علامات ، حروف الفبای لاتین و سایر انواع اطلاعات مورد استفاده قرار می گیرد.

سیستم اعداد دو حالته را اصطلاحا "دودوئی" یا "باینری" می نامند. در این سیستم فقط دو رقم داریم : ۰ و ۱

بنابراین:

Low = 0 و High = 1

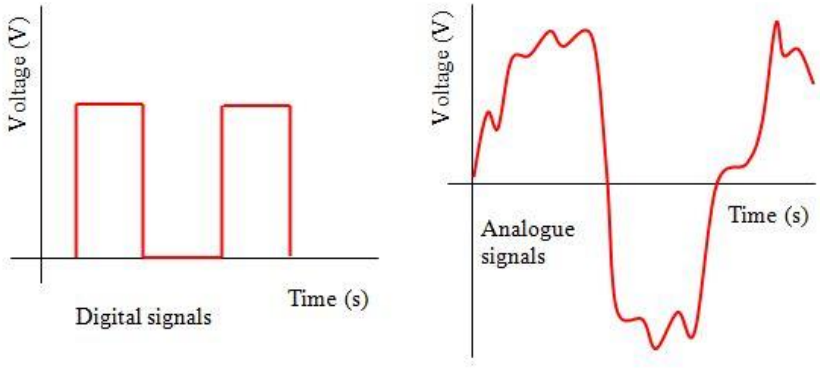
ولتاژهایی که برای نمایش ۰ و ۱ استفاده می شوند ، سطوح منطقی نامیده می شود که شامل سطح بالا و سطح پایین هستند. آی سی های خانواده TTL در محدوده ولتاژی ۰ تا ۵ ولت کار می کنند بنابراین منطق ۰ برابر صفر ولت و منطق ۱ برابر ۵ ولت است.

مثلا اگر ولتاژ ۰،۵ ولت باشد ، مدار مقدار منطقی ۰ یا همان Low را در نظر می گیرد و اگر ولتاژ ۳،۵ ولت باشد ، مدار مقدار منطقی ۱ یا همان High را در نظر می گیرد.

در این سیستم ولتاژ بین محدوده ی ۰،۸ تا ۲ ولت نباید استفاده شود ، چون بی معنی است (محدوده غیر مجاز).

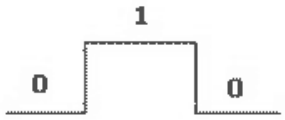
سیگنال دیجیتال :

سیگنال دیجیتال، سیگنالی است که هم از نظر زمان رخداد و هم از نظر مقدار در بازه ی خاصی محدود شده باشد. سیگنال دیجیتال در مقابل سیگنال آنالوگ تعریف می‌شود، که در آن حدودی برای این پارامترهای تعریف نمی‌شود. سیگنال دیجیتال از نظر ریاضی سیگنالی است که فقط از صفرها و یک‌های منطقی تشکیل شده باشد. این یک و صفرها ممکن است به شیوه‌های مختلفی نشان داده شوند که به این شیوه، کدینگ سیگنال گویند.

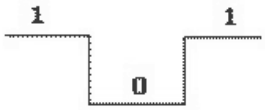


بنابراین سیگنال دیجیتال مجموعه ای از صفرها و یک های منطقی است که در واحد زمان عبور می کند . زمانی که سطح ولتاژ از Low به High می رود یک "لبه بالارونده" ایجاد می شود و سپس زمانی که از High به Low بازگردد "لبه پایین رونده" ایجاد می شود.

زمانی که سیگنال از Low به High رفته و بازگردد یک پله مثبت ایجاد می شود که به آن "پالس مثبت" می گویند .

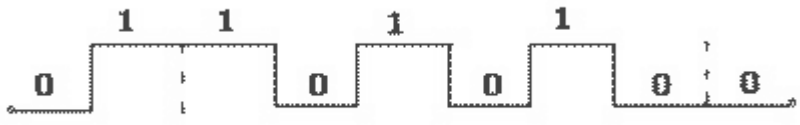


برعکس زمانی که سطح ولتاژ از High به Low رفته و سپس به High بازگردد ، یک پله منفی ایجاد می شود که به آن "پالس منفی" می گویند.



سیگنال دیجیتال ترکیبی از پالسهای مثبت و منفی است.

یک سیگنال دیجیتال دربرگیرنده اطلاعات بصورت باینری می باشد. نمونه ای از این سیگنال در زیر نشان داده شده است .کد سیگنال زیر دارای ۹ بیت است و از چپ به راست به صورت ۰۱۱۰۱۰۱۰۰ نوشته می شود.



مفهوم فرکانس :

پدیده های تکرار شونده بسیار زیادند

–حرکت بال پرندگان

-حرکت یک خط کش که از یک طرف به میز متصل است

–فلاشر ها

–موج برق شهر

–و . . .

هر کدام از این پدیده ها با چه سرعتی تکرار می شوند (چند بار در ثانیه ؟)

بال گنجشک : ۵ بار

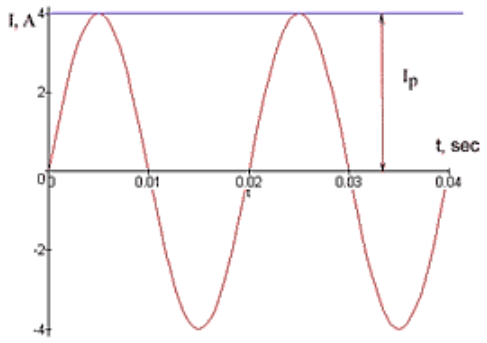
خط کش متصل به میز : به طول آن بستگی دارد ، بین ۵ تا ۱۰۰ بار

موج برق شهر : ۵۰ بار

به تعداد تکرار در ثانیه ، "فرکانس" گفته می شود و آن را با واحد هرتز بیان می کنیم. (مثلا فرکانس برق شهر ۵۰ هرتز است)

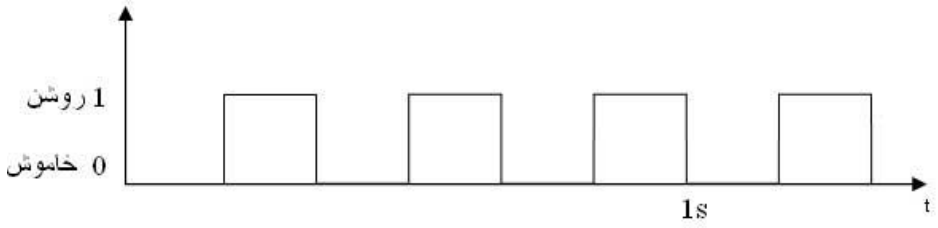
نکته : رابطه فرکانس با زمان یک رفت و برگشت (یک پریود) به صورت عکس است($f=1/T$)

در شکل زیر یک سیگنال متناوب **آنالوگ** با دوره ۰،۰۱ ثانیه را مشاهده می کنید.



فرکانس در سیگنالهای دیجیتال :

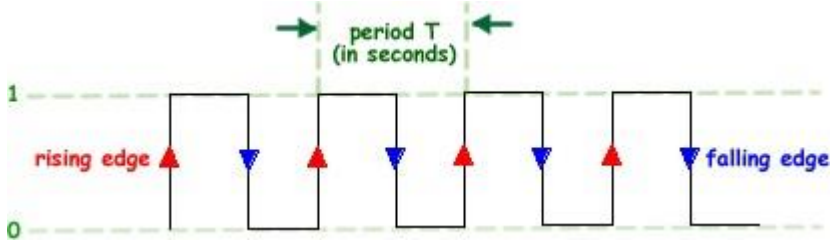
در سیستم دیجیتال ، رفت و برگشت معادل است با ۰ و ۱ شدن یک سیگنال. در نتیجه فرکانس یک سیگنال برابر می شود با تعداد ۰ و ۱ شدن ها در یک ثانیه و عکس فرکانس آن نیز دوره تناوب را نشان می دهد . مثلا در شکل زیر یک سیگنال منظم با فرکانس ۳ هرتز را مشاهده می کنید. زیرا در طول یک ثانیه سه بار سطح ولتاژ High و سپس Low شده است.



فرکانس 3 هرتز

تعریف سیگنال کلاک (Clock)

کلاک پالس یا پالس ساعت که در تمامی سیستم های دیجیتال وجود دارد به **یک سیگنال دیجیتال متناوب با دوره تناوب ثابت** گفته می شود که تمام سیستم با آن هماهنگ است . یک پالس کلاک پایه و اساس انجام شدن یک کار یا بخشی از یک کار در واحد زمان است. هر پالس کلاک در واحد زمان در دو سطح بالا و پایین نوسان میکند. به این صورت، با هر بار تکرار شدن این وضعیت، یک کار و یا بخشی از آن انجام میشود. . اگر سیستم دیجیتال را به یک کارخانه بسیار هماهنگ تشبیه کنیم که خطوط تولید مختلفی دارد ، کلاک را میتوان به یک فرد طبل به دستی که همه کارخانه گوش به فرمان آن فرد هستند تشبیه کرد که با هر ضرب طبل همه خطوط تولید یک مرحله به پیش می روند . عکس دوره تناوب پالس کلاک سرعت کار سیستم را نشان می دهد که یکی از مهمترین معیارهای سنجش سرعت پردازش و انتقال اطلاعات ، سرعت نوسان کلاک پالس در واحد زمان است. در شکل زیر یک پالس کلاک را مشاهده می کنید . به لبه بالارونده کلاک rising edge و به لبه پایین رونده آن falling edge گفته می شود.



آشنایی با سیستم اعداد باینری

مقدمه

آیا تا به حال به این نکته توجه کرده اید که خواندن اعداد توسط یک آی سی ، یا بطور کلی تر انتقال اطلاعات بین سخت افزارها و مدارات دیجیتالی به چه صورت انجام می شود؟
انسان ها بعنوان موجودات زنده ای که دارای مغز بسیار پیچیده تری نسبت به سایر موجودات زنده هستند ، ارتباطات بسیار پیچیده و سطح و بالایی با هم دارند. روابط بین انسانها را با روابط بین حیوانات یا گیاهان مقایسه کنید.
سوال : آیا می دانید ابداع و استفاده از چه ابزاری موجب شد تا ارتباط بین انسانها به این حد از پیشرفت برسد؟
جواب : زبان

شاید به دلیل اهمیت فوق العاده زیاد این قضیه است که انسان را "حیوان ناطق" می نامند. زیرا تا زمانی که زبان بعنوان وسیله ارتباطی بین انسانها شکل نگرفته بود و آنها بصورت ابتدایی در غارها و جنگل ها زندگی می کردند ، هیچ پیشرفت و تمدنی نداشتند .
ولی با شکل گیری زبان برقراری ارتباط در سطح بالایی تقویت شد ، تمدن ها شکل گرفتند و علم و دانش به سرعت گسترش پیدا کرد. امروزه ارتباط بین آدمها فقط در حد حرف زدن پیرامون رفع احتیاجات روزمره نیست ، بلکه آنها به راحتی در مورد موضوعات پیچیده علمی ، منطقی ، فلسفی و حتی انتزاعی و خیالی با یکدیگر تبادل نظر انجام می دهند.
با این مقدمه به سراغ سخت افزارها و مدارات دیجیتالی که ماشین های بی جانی هستند برمی گردیم.
اگر بتوان زبانی ابداع کرد که این قطعات الکترونیکی بی جان بوسیله آن بتوانند با یکدیگر ارتباط برقرار کنند و توسط آن اطلاعات را پردازش یا منتقل کنند ، روح پیدا می کنند و قابل استفاده می شوند.
این زبان قبلا ابداع شده و به آن "زبان ماشین" گفته می شود. این زبان مجموعه گسترده ای از کد می باشد. یعنی هر یک از حروف ، اعداد ، علائم و نشانه ها در این زبان دارای یک کد خاص هستند .
تمام این کدها از کنار هم قرار گرفتن تعدادی صفر و یک تشکیل می شوند. که این صفر ها و یک ها را ما خودمان قرارداد کردیم و ابزارهای الکترونیکی را بر این مبنا ساخته ایم:

-سطح ولتاژ بالا یا همان "High" = 1
-سطح ولتاژ پایین یا همان "Low" = 0

به سیستمی که در آن هر عدد مجموعه ای از ارقام { ۰ و ۱ } باشد ، "سیستم باینری" یا "سیستم دودوئی" یا "سیستم اعداد در مبنای ۲ " گفته می شود.

اعدادی که ما در زندگی روزمره برای شمردن و محاسبات استفاده می کنیم هر کدام مجموعه ای از ارقام { ۰ و ۱ و ۲ و ۳ و ۴ و ۵ و ۶ و ۷ و ۸ و ۹ } می باشد. به این سیستم "ده دهی" یا " سیستم اعداد در مبنای ۱۰" گفته می شود. اعداد واقعیت هایی در جهان بیرونی هستند و اینکه در چه سیستمی بیان شوند فقط به قرارداد بین انسانها بستگی دارد. اینکه ما از سیستم اعداد در مبنای ۱۰ استفاده می کنیم فقط قرارداد است و همین اعداد می توانند در هر مبنای دیگری بیان شوند.

تبدیل اعداد از مبنای ۲ به مبنای ۱۰ :

برای یک ماشین تنها مبنای ۲ معنا دارد . همه عملیات ها (عملیات منطقی ، ضرب ، جمع ، تقسیم ، تفریق و ...) ، همه داده ها و ذخیره اطلاعات به صورت باینری صورت میگیرد . بنابراین لازم است مبنای ماشین را به خوبی یاد بگیریم . در این قسمت با نحوه تبدیل اعداد از مبنای ۱۰ به مبنای ۲ و بالعکس آشنا شویم .

جدول ارزش مکانی اعداد در مبنای ۱۰:

یکان	دهگان	صدگان	یکان هزار	دهگان هزار	صدگان هزار

این جدول به همین صورت ادامه دارد تا ارزش مکانی های بالاتر . اینگونه جدول بندی در واقع این معنی را می دهد:

$10^5=100000$	$10^4=10000$	$10^3=1000$	$10^2=100$	$10^1=10$	$10^0=1$

مثال: عدد ۲۳۰۵۴۱

$$230541 = (1\times10^0)+(4\times10^1)+(5\times10^2)+(0\times10^3)+(3\times10^4)+(2\times10^5)$$

$10^5=100000$	$10^4=10000$	$10^3=1000$	$10^2=100$	$10^1=10$	$10^0=1$
۲	۳	۰	۵	۴	۱

اما جدول ارزش مکانی اعداد در مبنای ۲ بصورت زیر است:

$2^5=32$	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$

این جدول نیز به همین صورت ادامه دارد تا ارزش مکانی های بالاتر.

مثال: عدد ۱۰۰۱۰۱ در مبنای ۲:

$$(100101)_2 = (1\times2^0)+(0\times2^1)+(1\times2^2)+(0\times2^3)+(0\times2^4)+(1\times2^5)$$
$$= 1+0+4+0+0+32 = 37$$

$2^5=32$	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$
۱	۰	۰	۱	۰	۱

پس عددی که در این مثال در مبنای ۲ به ما داده شده بود ، همان عدد ۳۷ است در مبنای متداول ۱۰.

تبدیل اعداد از مبنای ۱۰ به مبنای ۲ :

اگر بخواهیم اعداد متداول و رایج خودمان را که در مبنای ۱۰ هستند به کدهای ۰ و ۱ تبدیل کنیم ، باید از روشی به نام "تقسیم های متوالی" استفاده کنیم .

در این روش عدد مورد نظر را بر ۲ تقسیم کرده ، خارج قسمت و باقیمانده آن را مشخص می کنیم. اگر خارج قسمت بزرگتر از ۱ بود مجددا آن را بر ۲ تقسیم می کنیم و خارج قسمت و باقیمانده تقسیم جدید را مشخص می کنیم.

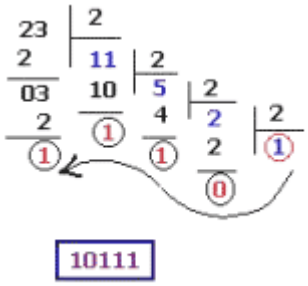
این تقسیمات متوالی بر ۲ را ادامه می دهیم تا جایی که خارج قسمت ۱ شود .

باقیمانده هر مرحله را نیز جداگانه مشخص می کنیم.

آخرین خارج قسمت را که ۱ است بعنوان اولین رقم عدد مورد نظرمان در مبنای ۲ در نظر می گیریم و و در ادامه به ترتیب باقیمانده های تقسیم ها را از آخر به اول بعنوان ارقام بعدی می نویسیم.

مثال: عدد ۲۳ در مبنای ۲ را محاسبه کنید.

جواب:



اعداد در مبنای ۱۶:

اعداد در مبنای ۱۶ دارای رقم های ۰ تا ۱۵ می باشد . در استفاده از اعداد در مبنای ۲ مشکلی که وجود دارد طولانی بودن رقمهای آن است. مثلا عدد ۲۵۵ در مبنای ۲ به عدد ۱۱۱۱۱۱۱۱ تبدیل میشود در حالی که همین عدد در مبنای ۱۶ به صورت FF نمایش داده می شود. بنابراین بهتر است در برخی کاربردها از مبنای ۱۶ استفاده کرد . جدول زیر اعداد در مبنای ۱۶ و معادل باینری هر عدد را نشان می دهد.

مبنای ۱۶	مبنای ۲
۰	۰۰۰۰
۱	۰۰۰۱
۲	۰۰۱۰
۳	۰۰۱۱
۴	۰۱۰۰
۵	۰۱۰۱
۶	۰۱۱۰
۷	۰۱۱۱
۸	۱۰۰۰
۹	۱۰۰۱
A	۱۰۱۰
B	۱۰۱۱
C	۱۱۰۰
D	۱۱۰۱
E	۱۱۱۰
F	۱۱۱۱

برای تبدیل یک عدد مبنای ۲ به مبنای ۱۶ ، ابتدا آن عدد را از سمت راست چهار رقم چهار رقم جدا می کنیم ، اگر تعداد ارقام مضرب چهار نیست در سمت چپ به تعداد لازم ۰ قرار می دهیم . سپس برای هر گروه چهار تایی معادل مبنای ۱۶ آن را قرار می دهیم . برای مثال :

۱۰۱۰۰۱۰ مبنای دو = ۰۱۰۱ ۰۰۱۰ مبنای دو = ۵۲ مبنای شانزده

۱۱۰۱۱۱۰۱ مبنای دو = ۱۱۰۱ ۱۱۰۱ مبنای دو DD = مبنای شانزده

برای تبدیل یک عدد مبنای شانزده به دودویی معادلش ، به جای هر عدد در مبنای ۱۶ معادل چهار رقمی آن در مبنای ۲ را جایگزین کنید. برای مثال:

3 A در مبنای شانزده = ۰۰۱۱ ۱۰۱۰ در مبنای دو

7 E در مبنای شانزده = ۱۱۱۰ ۰۱۱۱ در مبنای دو

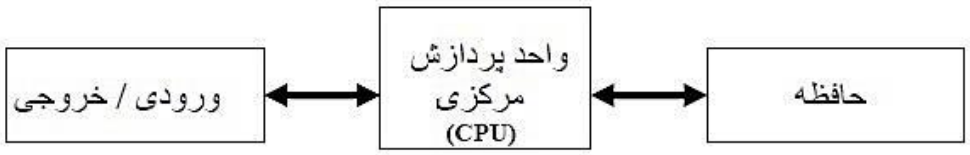
فصل سوم : معرفی ساختار میکروکامپیوتر و تفاوت آن با میکروکنترلر

مقدمه

ما در عصری زندگی می کنیم که جامعه شناسان آن را عصر انقلاب کامپیوتر نهاده اند اما آنچه بیش از همه قابل تامل است این است که انقلاب اصلی تنها در ۵۰ سال اخیر و با ظهور ترانزیستور آغاز شده است. انقلابی که نسل های مختلف کامپیوتری را بوجود آورد تا به **نسل چهارم** یعنی **میکروکامپیوترها** رسید. نسلی که مبتنی بر تکنولوژی مدارات مجتمع با فشردگی بسیار زیاد یعنی VLSI (مخفف Very Large Integrated Circuit) و معماری کامپیوتر بر اساس ریزپردازنده ها (میکروپرسسور ها) می باشند . اما پیشرفت های بعدی که هنوز ادامه داشت به جایی رسید که به سمت کوچکتر ، کم هزینه تر ، با سرعت بیشتر و توان مصرفی پایین تر شدن پیش رفت و **نسل پنجم** که همان **میکروکنترلرها** هستند شکل گرفت . میکروکنترلرها با هزینه بسیار کمتر از میکروپرسسورها ، دارای سیستمی کوچکتر اما یکپارچه ، دارای امکانات جانبی بیشتری می باشد و با بهره گیری از معماری جدیدتر میتواند حتی سریعتر از میکروپرسسورها باشد و مزیت ویژه آن نویز پذیری پایین تر و هنگ کردن کمتر است. با ساخت میکروکنترلرها تحول شگرفی در ساخت تجهیزات الکترونیکی نظیر لوازم خانگی ، صنعتی ، پزشکی ، تجاری و ... به وجود آمده است که بدون آن تصور تجهیزات و وسایل پیشرفته امروزی غیر ممکن است. به عنوان مثال می توان از ربات ها ، تلفن همراه تبلت ها و انواع سیستم های کنترلی دیگر نام برد.

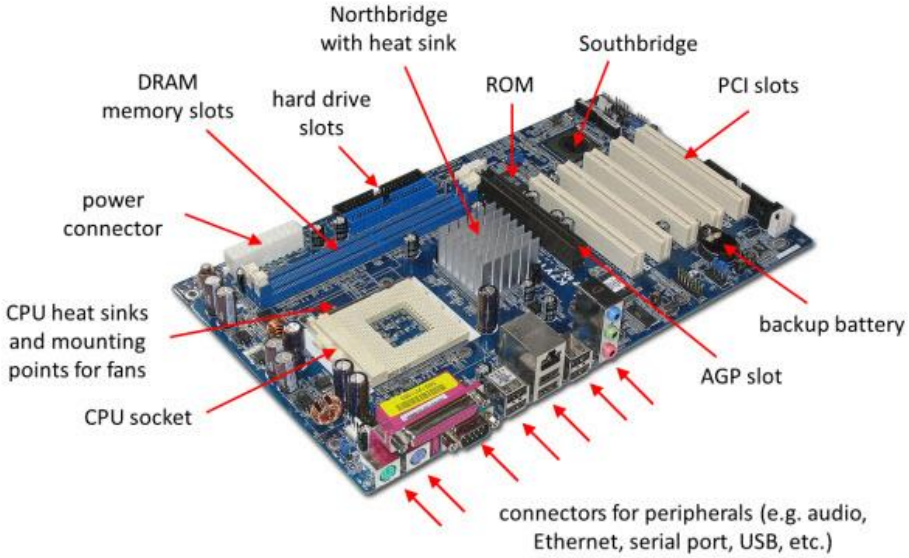
تعریف کامپیوتر

کامپیوتر به معنای محاسبه کردن می باشد و اساس کار آن از روی مغز انسان طراحی و ساخته شده است .
کامپیوترهای اولیه بعد از اختراع ترانزیستور ساخته شدند و توانایی انجام محاسبات بسیار ساده را داشتند . اما امروزه از کامپیوترها برای انجام محاسبات پیچیده با سرعت بالا و همچنین برای دخیره و مقایسه اطلاعات استفاده می شود . به زبان ساده تر یک کامپیوتر از سه واحد ورودی ، واحد پردازش و واحد خروجی تشکیل شده است. یک کامپیوتر داده ها را از ورودی گرفته و پس از پردازش مناسب و انجام محاسبات روی آنها ، فرمانهایی را به خروجی ارسال می کند.

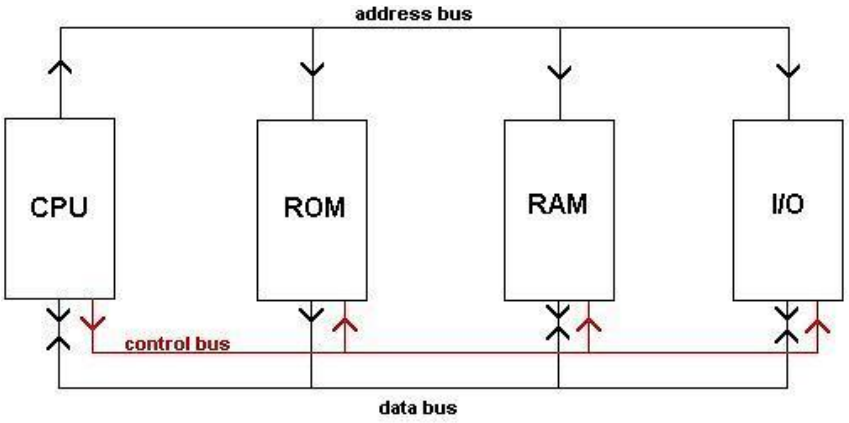


تعریف میکروکامپیوتر

یک سیستم میکروکامپیوتر حداقل شامل میکروپرسسور (CPU) ، حافظه موقت (RAM) ، حافظه دائمی (ROM) و قطعات ورودی/خروجی (PORT) می باشد که بوسیله گذرگاه (BUS) به هم ارتباط دارند و همه مجموعه روی یک برد اصلی به نام مادر برد قرار می گیرند. تمام کامپیوترهای خانگی امروزی مانند PC ها و لپتاپ ها از این نوع هستند که علاوه بر واحد های حداقلی فوق قطعات و واحدهای دیگری نیز به آنها اضافه شده است مانند شکل زیر.



طرز کار یک میکروکامپیوتر به این صورت است که ابتدا برنامه ایی که در حافظه دائمی قرار داده شده است اجرا میشود تا سیستم بوت شده و در حالت آماده به کار قرار گیرد. سپس بر اساس برنامه کاربر یا برنامه ای که توسط سیستم عامل به CPU داده میشود ، دستورات در CPU اجرا شده و برحسب نوع برنامه داده های مناسب در صورت نیاز از ورودی یا از حافظه گرفته شده و بعد از پردازش به خروجی ارسال می شود. شکل یک سیستم میکروپرسسوری حداقلی به صورت زیر است.



تعریف میکروپروسسور (CPU)

CPU مخفف عبارت Central Processor Unit به معنای واحد پردازنده مرکزی می باشد . میکروپروسسور تراشه (IC) ای است که از مدارات منطقی دیجیتال ساخته شده است که وظایف آن به شرح زیر است :

- انجام محاسبات ریاضی ، منطقی و بیتی
- انجام دادن دستورالعمل ها (حلقه های شرطی و ...)
- ارتباط با حافظه
- کنترل تجهیزات جانبی
- پاسخ دادن به وقفه ها

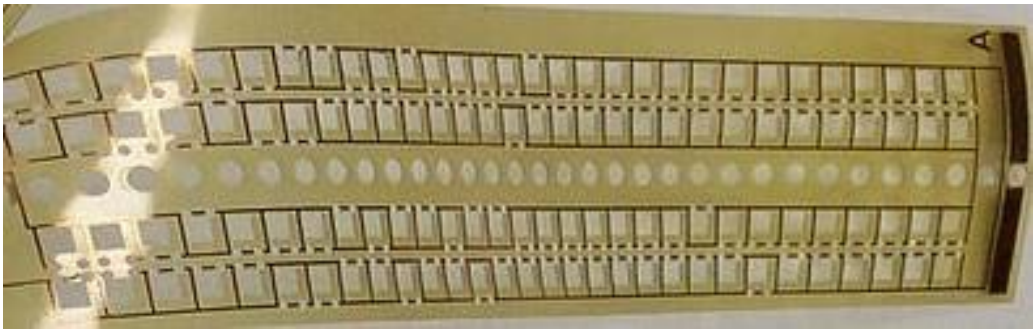
تعریف ROM

مخفف عبارت Read Only Memory به معنای حافظه فقط خواندنی می باشد. این حافظه دائمی بوده یعنی با قطع برق اطلاعات درون آن از بین نمی رود. برنامه راه اندازی سیستم و سیستم عامل (برنامه کاربر) در این حافظه قرار می گیرد. در بسیاری از کامپیوتر های امروزی بخشی از سیستم عامل روی ROM و بیشتر آن روی هارد دیسک قرار دارد.

انواع حافظه های ROM

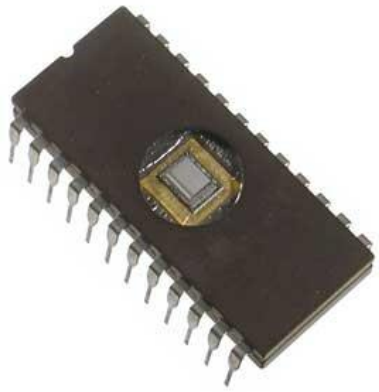
- **ROM**: در این نوع حافظه که توسط کارخانه و فقط برای یکبار پروگرام می شود ، شامل شبکه ای از سطر ها و ستون های ماتریسی است که در نقاطی به نام بیت به هم میرسند. در صورتیکه خطوط مربوطه بخواهد "یک" باشد برای اتصال از دیود استفاده می شود و اگر بخواهد مقدار "صفر" باشد خطوط به یکدیگر متصل

نخواهند شد. دیود ، صرفا امکان حرکت " جریان " را در یک جهت ایجاد می کند ، بنابراین در صورتیکه دیود در نقطه مورد نظر ارائه گردد ، جریان هدایت شده و سلول یک خوانده می شود و در صورتیکه مقدار سلول صفر باشد یعنی در محل برخورد سطر و ستون دیودی وجود ندارد.



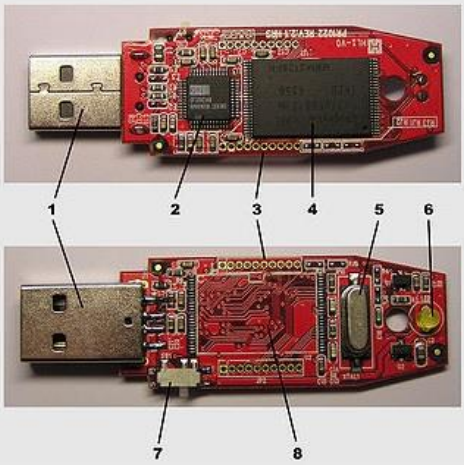
- **PROM** : تولید تراشه های ROM مستلزم صرف وقت و هزینه بالائی است . بدین منظور اغلب تولید کنندگان ، نوع خاصی از این نوع حافظه ها را که Programmable Read Only Memory نامیده می شوند ، تولید می کنند. این نوع از تراشه ها با محتویات خالی و با قیمت مناسب عرضه شده و می تواند توسط هر شخص با استفاده از دستگاه های خاصی برنامه ریزی گردند. ساختار این نوع از تراشه ها مشابه ROM بوده با این تفاوت که در محل برخورد هر سطر و ستون از یک فیوز استفاده می گردد. با توجه به اینکه تمام سلول ها دارای یک فیوز می باشند ، درحالت اولیه یک تراشه PROM دارای مقدار اولیه "یک" است . به منظور تغییر مقدار یک سلول به صفر ، از یک دستگاه خاص پروگرامر استفاده می گردد. حافظه های PROM صرفا یک بار قابل برنامه ریزی هستند و نسبت به RAM شکننده تر بوده و یک جریان حاصل از الکتریسیته ساکن ، می تواند باعث سوخته شدن فیور در تراشه شود و مقدار یک را به صفر تغییر نماید. از طرف دیگر PROM دارای قیمت مناسب بوده و برای نمونه سازی داده برای یک ROM ، قبل از برنامه ریزی نهائی کارآیی مطلوبی دارند.

- **EPROM** : که مخفف Erasable programmable read only memory است . این نوع حافظه ها همانند PROM هستند با این تفاوت که در آنها امکان پاک کردن حافظه توسط تاباندن مدت زمانی اشعه فرابنفش به حافظه بوجود آمد. بنابراین روی آی سی آنها شیاری تعبیه شده است که اشعه ماورای بنفش بتواند مستقیما به بخش اصلی حافظه بتابد.



- **EEPROM**: این نوع حافظه که Electrically Erasable Programmable ROM است ، می توان الکترون های هر بیت را با استفاده از یک نرم افزار و به کمک پروگرامر به وضعیت طبیعی برگرداند. بنابراین دیگر برای بازنویسی تراشه نیاز به جدا نمودن تراشه از محل نصب شده نخواهد بود و برای تغییر بخشی از تراشه نیاز به پاک نمودن تمام محتویات نخواهد بود. اعمال تغییرات در این نوع تراشه ها مستلزم بکارگیری یک دستگاه اختصاصی نخواهد بود.

- **Flash**: تراشه های EEPROM در هر لحظه تنها یک بیت خاص را تغییر می دهد و فرآیند اعمال تغییرات در تراشه کند است و در مواردی که می بایست اطلاعات با سرعت تغییر یابند ، سرعت لازم را ندارد. تولیدکنندگان با ارائه Flash Memory که یک نوع خاص از حافظه های EEPROM می باشد به محدودیت اشاره شده پاسخ لازم را داده اند. در حافظه Falsh داده ها داخل بلاک هایی که معمولا ۵۱۲ بایت می باشند ، نوشته می گردند. در کنار حافظه Flash یک کنترلر قرار دارد که توسط آن تمام اعمال مربوط به راه اندازی ، ذخیره و بازخوانی حافظه کنترل می شود. با اضافه شدن کنترلر می توان تمام و یا بخش های خاصی از تراشه را را حذف کرد که باعث سریعتر شدن این نوع حافظه نسبت به حافظه های EEPROM می گردد.



Internals of a typical USB flash drive

1	USB Standard, Male A-plug
2	USB mass storage controller device
3	Test point
4	Flash memory chip
5	Crystal oscillator
6	LED (Optional)
7	Write-protect switch (Optional)
8	Space for second flash memory chip

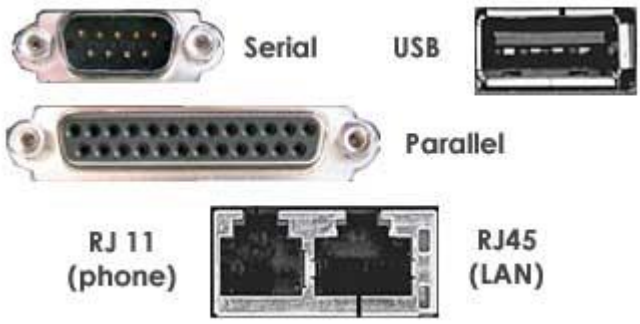
تعریف RAM: مخفف عبارت Random Access Memory به معنای حافظه با دسترسی تصادفی می باشد. این حافظه ، حافظه موقت می باشد یعنی با قطع برق اطلاعات آن از بین می رود . اطلاعات میانی سیستم شامل متغیرهای برنامه کاربر و داده های مربوط به ورودی/خروجی ها در این حافظه قرار می گیرد.

انواع حافظه های RAM

- **SRAM** : مخفف Static RAM به معنای حافظه ایستا است. در این نوع حافظه ها که از فلیپ فلاپ ساخته شده اند ، دارای سرعت خواندن و نوشتن بالا و توان مصرفی پایین می باشند.
- **DRAM** : مخفف Dynamic RAM به معنای حافظه پویا است. در این نوع حافظه ها از ترانزیستور MOSFET استفاده شده است که باعث کاهش سرعت خواندن و نوشتن و همچنین افزایش توان مصرفی نسبت به SRAM می باشد ولی در عوض حجم کمتری را اشغال می کند و ارزان تر از SRAM می باشد.

تعریف PORT

به معنای درگاه می باشد و وظیفه ارتباط میکروکامپیوتر با دستگاههای جانبی را بر عهده دارد . که این دستگاه های جانبی ورودی یا خروجی هستند برای همین به آنها I/O نیز می گویند. پورت ها انواع مختلفی دارند مانند پورت USB، پورت سریال ، پورت موازی و ... که در شکل زیر مشاهده می کنید.



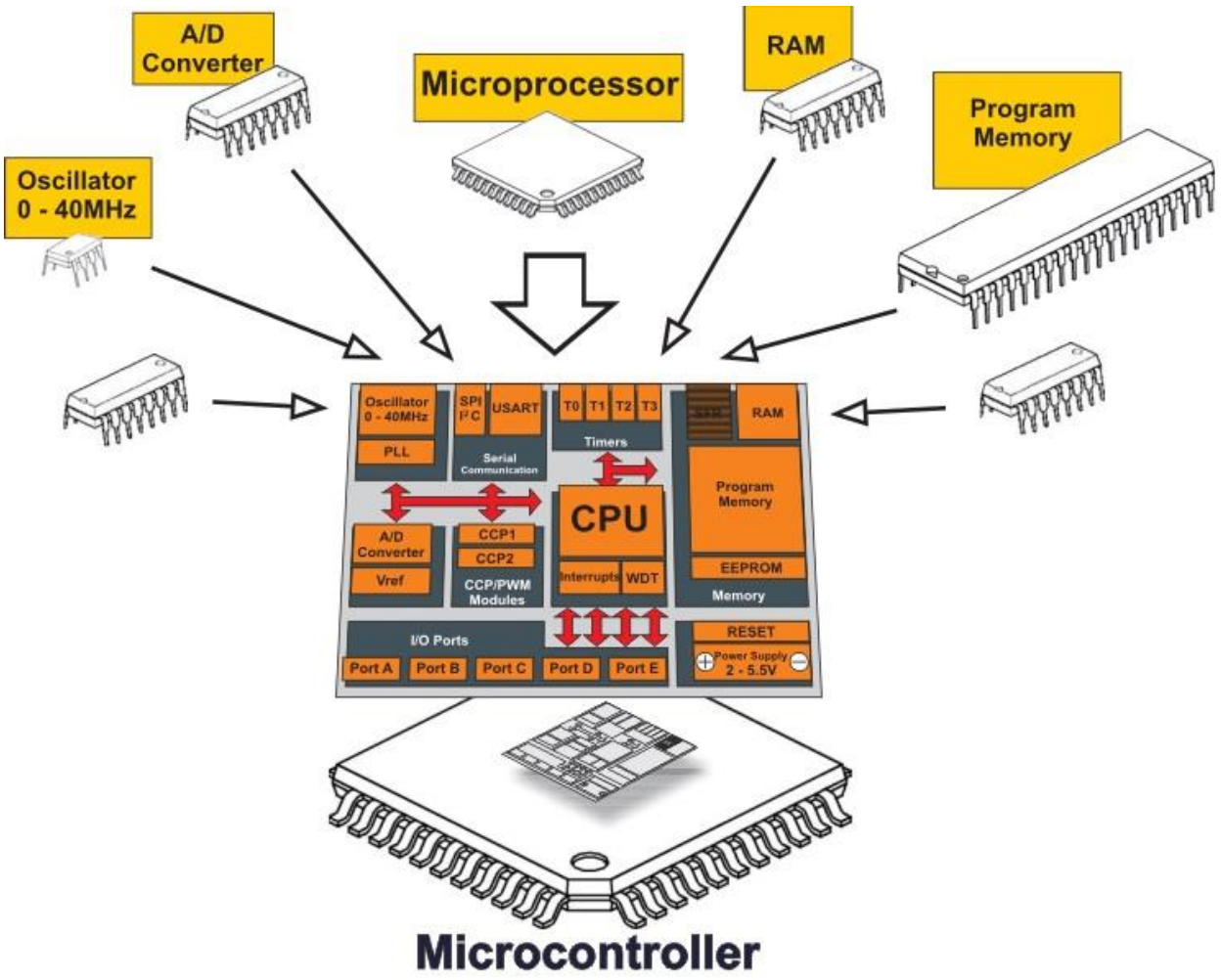
تعریف BUS

به معنای گذرگاه می باشد و وظیفه انتقال سیگنال دیجیتال را برعهده دارد و در عمل هیچ چیز جز مجموعه سیم های کنار هم قرار گرفته نیست. در هر میکروکامپیوتر ۳ نوع باس وجود دارد که وظیفه ی انتقال دیتا ، آدرس و سیگنالهای کنترلی را بر عهده دارد.

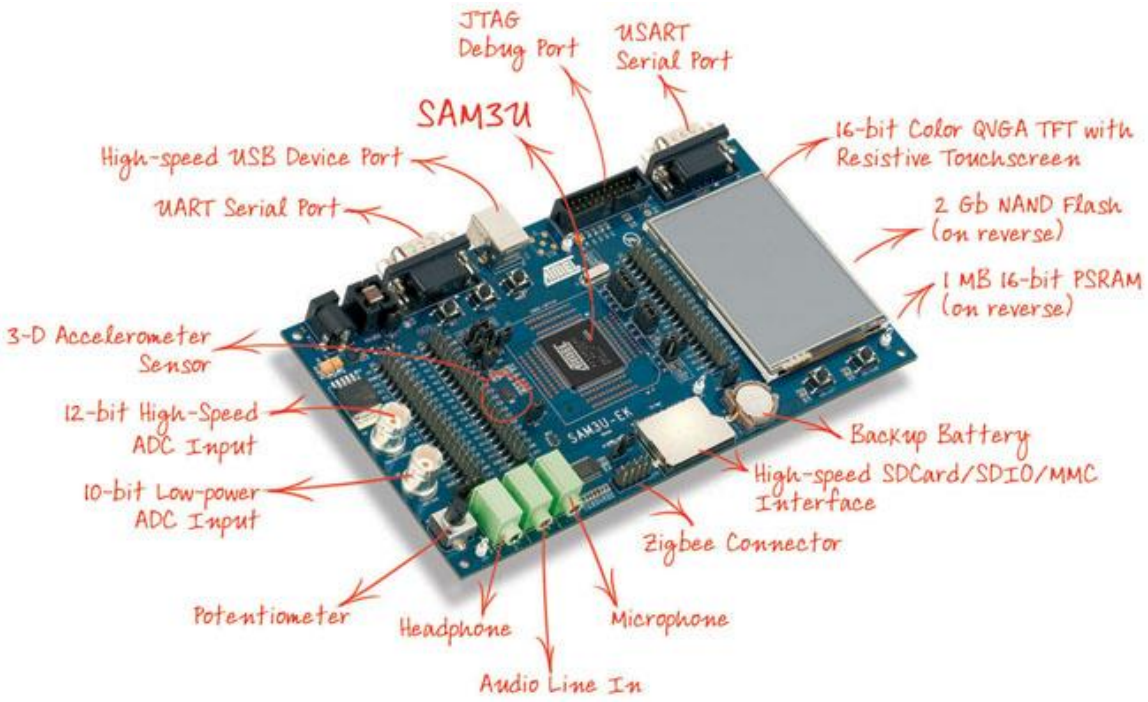
- **گذرگاه داده (Data Buss)** : جهت نقل و انتقال داده ها بین قسمت های مختلف به کار می رود که میتواند ۸ خط (در میکرو های ۸ بیتی)، ۱۶ خط (در میکرو های ۱۶ بیتی) و ... باشد. هر چه تعداد خطوط Data بیشتر باشد سرعت انتقال داده ها بیشتر خواهد بود و توان پردازش میکروکامپیوتر بالاتر است.
- **گذرگاه آدرس (Address Buss)** : برای شناسایی هر وسیله حافظه یا I/O توسط CPU می باشد که باید آدرس منحصر به فردی به آنها تخصیص داد. هر چه تعداد خطوط آدرس بیشتر باشد میکروکامپیوتر میتواند تعداد مکانهای بیشتری را آدرس دهی کند ، در نتیجه میتواند حافظه بزرگتری داشته باشد.
- **گذرگاه کنترل (Control Buss)** : شامل خطوط کنترلی دستگاههای موجود مثل فرمان نوشتن Read ، خواندن Writeو غیره است. هر چه تعداد خطوط کنترلی بیشتر باشد، میکروکامپیوتر امکانات کنترلی بیشتری در اختیار برنامه نویس قرار میدهد.

تعریف میکروکنترلر

هنگامی که قطعات سازنده یک میکروکامپیوتر در یک تراشه و در کنار هم قرار گیرند ، یک میکروکنترلر به وجود می آید. در واقع میکروکنترلر یک آی سی شامل یک CPU ، به همراه مقدار مشخصی از حافظه های RAM و ROM ، پورت های ورودی/خروجی و همچنین واحد های جانبی دیگری نظیر تایمر ، رابط سریال و ... می باشد. به عبارت دیگر میکروکنترلر یک تراشه الکترونیکی قابل برنامه ریزی است که استفاده از آن باعث افزایش سرعت و کارآیی مدار در مقابل کاهش حجم و هزینه مدار می گردد.



امروزه از میکروکنترلر ها به علت کوچکی و سادگی در بسیاری از وسایل الکترونیکی استفاده می شود. میکروکنترلرها کاربردهای وسیعی در صنایع الکترونیکی پیدا کرده اند و با توجه به ویژگی ها و امکانات اصلی و جانبی که ارائه می دهند ، انتخاب شده و به کار برده می شوند. از ویژگی ها و امکانات اصلی مثلا : سرعت پردازنده ، قدرت پردازش اطلاعات ، میزان ذخیره اطلاعات ، نویز پذیری ، فرکانس کاری ، توان مصرفی و ... امکانات جانبی مثل پشتیبانی از پروتکل های ارتباطی ، ارتباط با وسایل جانبی ، پاسخ به وقفه ها ، تایمر ها و کانتر ها ، مبدل های آنالوگ به دیجیتال و ... در شکل زیر یک سیستم میکروکنترلری را مشاهده می کنید که دارای انواع پورت ها و صفحه نمایش ، امکانات کنترلی و جانبی مختلف است در حالی که نسبت به سیستم میکروکامپیوتری کوچکتر ، کم هزینه تر و شاید هم پر سرعت تر باشد.



انواع میکروکنترلرها

اولین میکروکنترلر در سال۱۹۷۱ توسط شرکت نام آشنای intel ساخته شد و این شرکت اولین میکروکنترلر کاربردی خود را در سال ۱۹۸۰با نام ۸۰۸۰ روانه بازار کرد. بعد از آن میکروکنترلر توسط شرکت اینتل با سری چیپهای...,AT8050,8051,8052 ، شرکت زایلوگ باسری چیپهای ...,Z8601,8602,8603 و شرکت موتورولا با سری چیپهای 6811 ...,A1,A2گسترش یافت. در حال حاضر میکروکنترلرهای پرکاربرد موجود دارای انواع زیر هستند که هریک کاربرد ها و ویژگی های مخصوص به خود را دارند :

- خانواده AVR : ساخت شرکت ATMEL
- خانواده PIC : ساخت شرکت MicroChip
- خانواده ARM : ساخت شرکت های ATMEL ، NXP، STM و ...
- خانواده FPGA : ساخت شرکت های Xilinx وAltera

هر یک از خانواده های فوق دارای زیرمجموعه های بسیاری می باشد اما به صورت کلی میتوان آنها را به صورت جدول زیر مقایسه نمود :

سری میکرو	تعداد زیر مجموعه ها	حداکثر فرکانس کاری	منابع یادگیری	قیمت	قدرت پردازش عمومی*	قدرت پردازش اختصاصی*	نویز پذیری	پشتیبانی از پروتکل ها
خانواده AVR	بیش از 120	300MHz	خیلی زیاد	نسبتا ارزان	متوسط	ضعیف	زیاد	متوسط
خانواده PIC	بیش از 60	40MHz	زیاد	متوسط	متوسط	متوسط	کم	خوب
خانواده ARM	بیش از 200	بیش از 1GHz	متوسط	متوسط	بالا	بالا	کم	خیلی خوب
خانواده FPGA	بیش از 200	بیش از 1GHz	متوسط	متوسط	متوسط	بالا	کم	متوسط

*منظور از قدرت پردازش عمومی و اختصاصی ، سرعت و قدرت پردازش اطلاعات در مصارف عمومی)مانند کار های کنترلی و...) اختصاصی (مانند پردازش تصویر و...) می باشد.

فصل چهارم : معرفی و ساختار میکروکنترلرهای AVR

مقدمه

تا این بخش از آموزش متوجه شدیم که میکروکنترلرها انواع مختلفی دارند و بسته به نوع کاری که مورد نظر است ، یکی از خانواده های میکروکنترلرها که برای انجام آن مناسب تر است ، انتخاب می شود . مثلا اگر سرعت پردازش بسیار بالا بدون هنگ کردن و قابلیت تغییر کل برنامه در یک کاربرد خاص در یک پروژه نیاز باشد (مانند پروژه های نظامی و فرکانس بالا) بهتر است به سراغ FPGA و انواع آنها رفت . اگر در یک پروژه سرعت نسبتا بالا و قابلیت پشتیبانی از انواع ارتباطات جانبی مانند پورت USB ، ارتباطات سریال و ... مورد نیاز باشد (مانند استفاده در تلفن همراه ، تبلت ها ، پروژه های پردازش سیگنال ، تلویزیون ها و ...) بهتر است از میکروکنترلرهای ARM استفاده کرد . اگر در یک پروژه سرعت بالا مورد نظر نباشد و فقط درست و بدون نقص انجام شدن کار مورد نظر باشد (مانند پروژه های صنعتی) از میکروکنترلرهای PIC استفاده می شود که در محیط های پرنویز مانند کارخانه ها بیشتر از آنها استفاده می شود . و در نهایت اگر در کاربردهایی معمولی و متوسط با قابلیت های متوسط (مانند پروژه های دانشگاهی ، منازل و...) مورد نظر باشد از میکروکنترلرهای AVR بیشتر استفاده می گردد . بنابراین یاد گرفتن میکروکنترلرهای AVR در مرحله اول ضروری است چرا که از نظر معماری و کاربردها ساده تر بوده و مباحث اصلی و پایه ای در این مرحله وجود دارد که بدون دانستن آنها به مرحله بالاتر رفتن برای یک **مهندس** الکترونیک اشتباه است.

معرفی میکروکنترلر AVR

AVR خانواده‌ای از میکروکنترلر ها است که شرکت ATMEL ، آن را روانه ی بازار الکترونیک کرده است. این میکروکنترلر های هشت بیتی به خاطر دارا بودن قابلیت برنامه‌نویسی توسط کامپایلرهای زبان‌های برنامه نویسی سطح بالا ، مورد توجه قرار می‌گیرند. این میکروکنترلر ها از معماری RISC برخوردارند .همچنین شرکت اتمل کوشیده‌است تا با استفاده از معماری پیشرفته و دستورهای بهینه، حجم کد تولید شده را پایین آورده و سرعت اجرای برنامه را بالا ببرد .یکی از مشخصات این نوع میکروکنترلر ها بهره گیری از تکنولوژی CMOS و استفاده از حافظه‌های کم مصرف و غیر فرار Flash و EEPROM است.

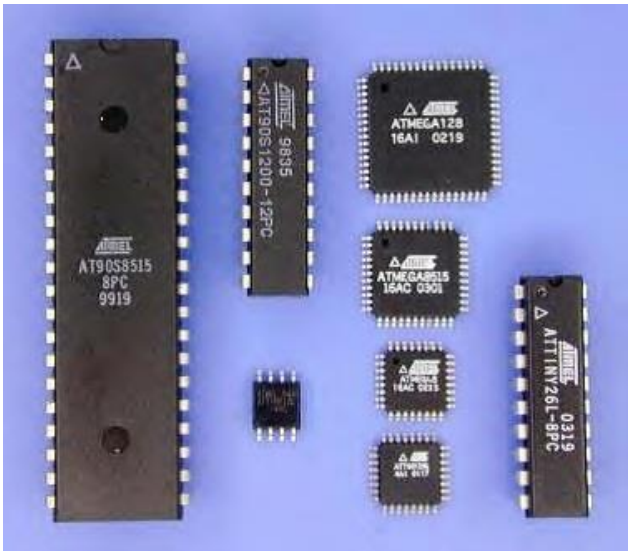
تاریخچه ساخت

میکروکنترلر AVR در سال ۱۹۹۶ توسط شرکت ATMEL ساخته‌شد. معماری این میکروکنترلر توسط دانشجویان دکترای دانشگاه صنعتی نروژ Alf-Egil Bogen و Vegard Wollan طراحی شد. شرکت اتمل می‌گوید نام AVRیک مخفف نیست و به نام خاصی اشاره نمی‌کند اما به نظر می‌رسد که این نام مخفف **A**lf (Egil Bogen) and **V**egard (Wollan)'s **R**ISC processor است.

انواع میکروکنترلرهای AVR

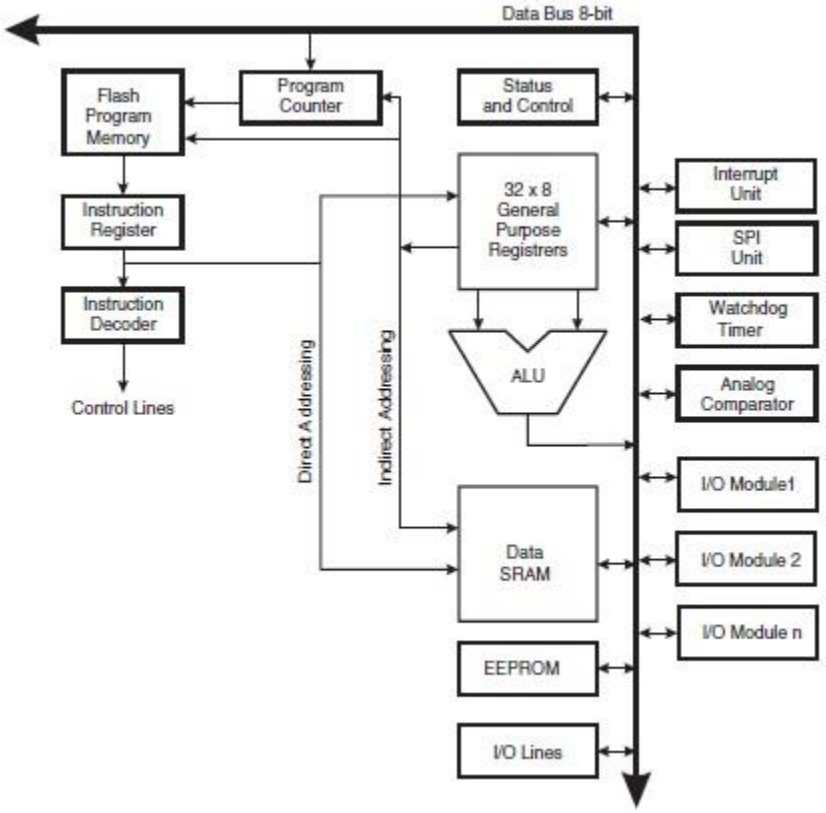
این میکروکنترلرها دارای ۴ سری می باشند که هر سری کاربردها و ویژگی های خاص خود را دارد.

1. سری **ATtiny** : میکروکنترلرهای کوچک ، کم مصرف و پرقدرت برای کاربردهای خاص می باشند که دارای حافظه Flash بین ۰٫۵ تا ۱۶ کیلوبایت و بسته بندی بین ۶ تا ۳۲ پایه هستند.
2. سری **ATMega** : این سری دارای امکانات وسیع و دستورالعمل های قوی می باشد که دارای حافظه Flashبین ۴ تا ۵۱۲ کیلوبایت و بسته بندی بین ۲۸ تا ۱۰۰ پایه هستند.
3. سری **XMega** : جدیدترین ، پرسرعت ترین و قوی ترین نوع هستند که امکانات بیشتری نیز دارند که دارای حافظه Flash بین ۱۶ تا ۳۸۶ کیلوبایت و بسته بندی ۴۴، ۶۴ و ۱۰۰ پایه هستند.
4. سری **AT90s** : نوع توسعه یافته میکروکنترلر ۸۰۵۱ هستند که امکانات کمتری داشته و کمتر کاربرد دارند چرا که تقریبا منسوخ شده اند.



معماری و ساختار میکروکنترلرهای AVR

شکل زیر معماری میکروکنترلرهای AVR را نشان می دهد . به طور کلی یک میکروکنترلر AVR از نظر ساختار داخلی **حداکثر** دارای واحدهای زیر می باشد :



- واحد پردازش مرکزیCPU
- واحد حافظه برنامهFlash
- واحد حافظه دادهEEPROM
- واحد حافظه دادهSRAM
- واحد ورودی و خروجیI/O
- واحد کنترل کلاک ورودی
- واحد کنترل وقفه
- واحد تایمر و کانتر
- واحد مبدل آنالوگ به دیجیتالADC
- واحد مقایسه کننده آنالوگ
- واحد تایمر سگ نگهبان
- واحد ارتباطات سریال SPI ، TWI و USART
- واحد برنامه ریزی و عیب یابیJTAG

هسته مرکزی(واحد پردازش مرکزی یا CPU)

این واحد که بر مبنای معماری RISC ساخته شده است ، تمام فعالیت های میکروکنترلر را مدیریت کرده وتمام عملیات لازم بر روی داده ها را انجام می‌دهد . همچنین وظیفه ارتباط با حافظه ها و کنترل تجهیزات جانبی را بر عهده دارد . درون هسته AVR به تعداد ۳۲ رجیستر همه منظوره ، واحد محاسبه و منطق (ALU) ، واحد رمز گشایی دستور ID ، رجیستر دستورالعمل IR ، رجیستر شمارنده برنامه PC ، رجیستر وضعیت SREG و اشاره گر پشته SP قرار دارند.

انواع معماری پردازنده ها

در ساختار پردازنده ها (CPU) دو معماری متفاوت CISC و RISC استفاده می شود.

CISC : مخفف Complete Instruction Set Computerبه معنای کامپیوتر با دستورالعمل کامل می باشد . در این ساختار تعداد دستورالعمل ها بسیار زیاد است . دستورها پیچیده بوده و با سرعت پایین اجرا می شوند ولی برنامه نویسی آن ساده تر)سطح بالا) است CPU . های میکروکامپیوترهای PC و لپ تاپ ها از این نوع هستند.

RISC : مخفف Reduced Instruction Set Computerبه معنای کامپیوتر با دستورالعمل کاهش یافته می باشد . در این ساختار تعداد دستورالعمل ها کمتر بوده و دستورات ساده تر هستند و با سرعت بسیار زیادی اجرا می شوند اما برنامه نویسی آن دشوار تر (سطح پایین) است CPU . های میکروکنترلرها از این نوع هستند.

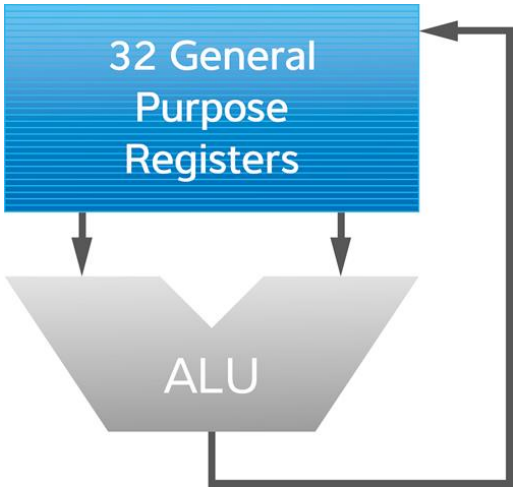
	CISC	RISC
1	Complex instructions taking multiple cycles	Simple instructions taking 1 cycle
2	Any instruction may reference memory	Only LOADS/STORES reference memory
3	Not pipelined or less pipelined	Highly piplined
4	Instructions interpreted by the microprogram	Instructions executed by the hardware
5	Variable format instructions	Fixed format instructions
6	Many instructions and modes	Few instructions and modes
7	Complexity in the microprogram	Complexity is in the compiler
8	Single register set	Multiple register sets

گفتیم که یکی از مزیت های میکروکنترلرها نسبت به میکروکامپیوترها سرعت انجام بالای دستورالعمل ها در آن ها است . این مزیت از همین معماری منحصر به فرد میکروکنترلرها ناشی می شود . در واقع نقطه قوت میکروکنترلرها که باعث شده که به عنوان نسل پنجم شناخته شوند همین RICS بودن آنها است که باعث می شود دستورات ساده را در سیکل کلاک کمتری نسبت به CISC انجام دهند . ایده اصلی RISC اولین بار توسط جان کوکی از IBM و در سال ۱۹۷۴ شکل گرفت، نظریه او به این موضوع اشاره داشت که یک کامپیوتر تنها از ۲۰ درصد از دستورات نیاز دارد و ۸۰ درصد دیگر، دستورات غیرضروری هستند. پردازنده‌های ساخته شده براساس این طراحی از دستورات کمی پشتیبانی می‌کنند به این ترتیب به ترانزیستور کمتری نیز نیاز دارند و ساخت آنها نیز کم هزینه است. با کاهش تعداد ترانزیستورها و اجرای دستورات کمتر، پردازنده در زمان کمتری دستورات را پردازش می‌کند. اما در CISC

مجموعه‌ای از دستورات بصورت فشرده و با آدرس دهی مختلف به یکباره پردازش می‌شوند، مثل اعداد اعشاری یا تقسیم که در طراحی RISC وجود ندارند. از آنجایی که دستورات در RISC ساده‌تر هستند پس سریعتر اجرا می‌شوند و نیاز به ترانزیستور کمتری دارند، ترانزیستور کمتر هم به معنی دمای کمتر، مصرف پایین‌تر و فضای کمتر است که آن را برای ابزارهای موبایل مناسب می‌کند. مثلا اگر عمل ضرب توسط یک میکروکنترلر در ۲ سیکل کلاک انجام شود و کلاک میکرو روی ۱۰ مگاهرتز تنظیم شده باشد ، عمل ضرب دو عدد به مدت ۲۰۰ نانو ثانیه طول میکشد ، در حالی که همان عمل ضرب در کامپیوتر با ۱۰۰ کلاک انجام می شود که اگر کلاک آن را ۱ گیگاهرتز در نظر بگیریم ، عمل ضرب ۱۰۰ نانو ثانیه طول می کشد . این مثال نشان می دهد که دستور ضرب در یک میکروکامپیوتر با قیمت مثلا یک میلیون تنها دو برابر سریعتر از یک میکروکنترلر با قیمت ۱۰ هزار تومان است . پس یک کامپیوتر برای کارهای بزرگ تر مناسب است و یک میکروکنترلر برای کارهای کوچکتر ساخته شده است و کار کوچک را با هزینه پایین تر و کیفیت بهتری انجام می دهد.

واحد محاسبه و منطق(Arithmetic Logic Unit)

ALUدر میکروکنترلر AVR به صورت مستقیم با تمام ۳۲ رجیستر همه منظوره ارتباط دارد. عملیاتهای محاسباتی با رجیسترهای همه منظوره در یک کلاک سیکل اجرا می شوند به طور کلی عملکرد ALU را می توان به سه قسمت اصلی ریاضیاتی، منطقی و توابع بیتی تقسیم بندی کرد در برخی از ALU های توسعه یافته در معماری میکروکنترلرهای AVR از یک ضرب کننده با قابلیت ضرب اعداد بدون علامت و علامتدار و نیز اعداد اعشاری استفاده شده است.



مفهوم ثبات یا رجیستر

رجیستر ها نوعی از حافظه های موقت هستند (مانند RAM)که از فلیپ فلاپ ها ساخته می شوند و میتوانند ۸ بیتی ، ۱۶ بیتی ، ۳۲ بیتی یا بیشتر باشند. از رجیستر ها به صورت گسترده در تمام ساختار و واحدهای میکروکنترلرها استفاده می شود . میکروکنترلرهای AVR هشت بیتی هستند ، بدین معنا که تمامی رجیستر ها در آن ، ۸ بیتی هستند. مهمترین مسئله که در هنگام برنامه نویسی میکروکنترلرها با آن مواجه هستیم نحوه صحیح مقدار دهی رجیسترهای آن میکروکنترلر می باشد. اگر میکروکنترلر را به یک کارخانه تشبیه کنیم که این کارخانه دارای بخشهای زیادی است و در هر بخش یک اتاق کنترل وجود دارد و در هر اتاق کنترل تعدادی کلیدهای کنترلی وجود دارد ، این کلیدهای کنترلی همان رجیسترها هستند که مهمترین وظیفه یک برنامه نویس شناختن این کلیدها و مقداردهی مناسب آنها در برنامه است.

رجیستر های عمومیGeneral Purpose Registers

میکروکنترلرهای AVR دارای ۳۲ رجیستر همه منظوره هستند این رجیسترها قسمتی از حافظه SRAM میکروکنترلر می باشند . تعداد این رجیستر ها ۳۲ عدد بوده و از R0 تا R31 شماره گذاری می شوند . هر رجیستر دارای ۸ بیت است که به طور مستقیم با واحد ALU در ارتباط است . رجیستر های R26 تا R31 به منظور آدرس دهی غیر مستقیم ، در فضای حافظه داده و برنامه استفاده میشوند که به آن ها رجیستر های اشاره گر می گویند و به ترتیب به صورت XL ، XH، YL ، YH، ZLو ZH نامگذاری می شوند . یک امکان برای دو رجیستر جدا از هم فراهم شده است که بتوان توسط یک دستورالعمل خاص در یک سیکل کلاک به آن دسترسی داشت . نتیجه این معماری کارایی بیشتر کدها تا ده برابر سریع تر از میکروکنترلر های با معماری CISC است.

رجیستر دستور Instruction Register

این رجیستر که در هسته پردازنده قرار دارد کد دستور العملی را که از حافظه برنامه FLASH خوانده شده و باید اجرا شود را در خود جای می دهد.

واحد رمز گشایی دستور Instruction Detector

این واحد تشخیص می دهد کد واقع در IR مربوط به کدام دستور العمل است و سیگنال های کنترلی لازم برای اجرای آن را صادر می نماید.

رجیستر شمارنده برنامه Program Counter

این رجیستر در واقع شمارنده آدرس دستورالعمل های برنامه کاربر است و در هر مرحله به آدرس خانه بعدی حافظه فلش که باید اجرا شود اشاره می کند . یعنی با اجرای هر دستور العمل محتوای رجیستر PC یک واحد افزایش یافته و به آدرس دستور العمل بعدی اشاره می کند.

رجیستر وضعیت Status & Control Register

این رجیستر ۸ بیتی واقع در هسته اصلی میکرو بوده و بیت های آن تحت تاثیر برخی عملیات CPU فعال میشوند . این بیت ها به ترتیب از بیت ۰ تا بیت ۷ به صورت زیر هستند:

پرچم کری C : این پرچم هنگامی که عملیات محاسباتی کری دهد یک می شود

پرچم صفر Z : این پرچم هنگامی که نتیجه یک عملیات محاسباتی یا منطقی صفر شود فعال می شود

پرچم منفی N : این پرچم هنگامی که نتیجه یک عملیات محاسباتی یا منطقی منفی شود فعال می شود

پرچم سرریز V : این پرچم زمانی که نتیجه یک عملیات علامت دار نادرست شود(سرریز) فعال می شود

پرچم علامت S : این پرچم همواره نتیجه XOR بین دو پرچم N و V می باشد

پرچم نیم کری H : اگر هنگام عملیات جمع یک انتقال از بیت ۳ به ۴ در یکی از رجیستر های عمومی یا نتیجه نیم بایت پایین (بیت های ۰ تا ۳) بزرگتر از ۹ باشد ، این پرچم فعال خواهد شد

بیت T : از این بیت در هنگام استفاده از دستورات اسمبلی BLD و BST به عنوان مقصد یا مبدا استفاده می شود.

بیت فعالساز وقفه سراسری I : در صورت یک بودن این بیت وقفه سراسری فعال می شود و در غیر این صورت هیچ وقفه ای رخ نمی دهد . این بیت در هنگام رخ دادن وقفه صفر و دوباره یک می شود

رجیستر اشاره گر پشته Stack Pointer

پشته قسمتی از فضای حافظه داده SRAM است که جهت ذخیره اطلاعات و معمولا در اجرای دستور فراخوانی یا CALL و یا اجرای برنامه وقفه نیاز می باشد . همچنین به کمک دستورات اسمبلی PUSH و POP میتوان به این قسمت نفوذ داشت . اشاره گر پشته یا SP دارای ۱۶ بیت است و از دو رجیستر ۸ بیتی SPL و SPH تشکیل شده است . در ابتدا وقتی سیستم روشن می شود CPU از محل استقرار حافظه پشته اطلاعی ندارد (پیش فرض (SP=0 بنابراین باید آدرسی از حافظه SRAM که مربوط به فضای پشته است را به اطلاع سیستم رساند.

با اجرای دستورPUSH R0 ، محتوای رجیستر R0با محتوای خانه ای از فضای پشته که SP به آن اشاره میکند جایگزین شده و بعد از آن SP یک واحد کاهش می یابد . همچنین با اجرای دستور POP محتوای حافظه پشته بیرون آمده و SP یک واحد افزایش می یابد.

نحوه عملکرد واحد CPU

در هنگام روشن شدن میکرو کنترلر یک مقدار از پیش تعیین شده در درون رجیستر PC قرار می گیرد که این مقدار از پیش تعیین شده همان آدرسی از حافظه فلش است که کد دستور اول در آن قرار دارد . سپس CPU فرآیند خواندن دستور العمل را انجام می دهد که به این مرحله Fetch می گویند . سپس CPU بعد از خواندن کد دستور آن را در رجیستر IR قرار می دهد تا واحد رمز گشایی دستور (ID) کد موجود در IR را تجزیه و تحلیل کرده و عملی که باید انجام گیرد را مشخص می کند . (Decode) سپس با ارسال سیگنال های کنترلی ، سایر داده های مورد نیاز دستورالعمل را فراخوانی و واکشی کرده (Memory Fetch) و سرانجام ALU عملیات مشخص شده را بر روی داده ها انجام می دهد . (Execute) در نهایت مقدار PC تغییر کرده و آدرس بعدی که باید اجرا شود در آن قرار میگیرد و چرخه ادامه می یابد.

نکته : برنامه نویس به رجیسترهای درون CPU کاری ندارد . رجیسترهای درون هسته مرکزی ، رجیسترهای خود سیستم هستند که هر یک وظیفه مشخصی دارند و مقدار آنها دائما تغییر می کند . رجیسترهای فوق الذکر فقط برای شناخت بهتر AVR و درک عملکرد این واحد معرفی شدند و در عمل یک برنامه نویس نیازی به استفاده از آن ندارد اما برای یک مهندس بهتر است فراتر از یک برنامه نویس باشد و درک عمیق تری از سخت افزار داشته باشد.

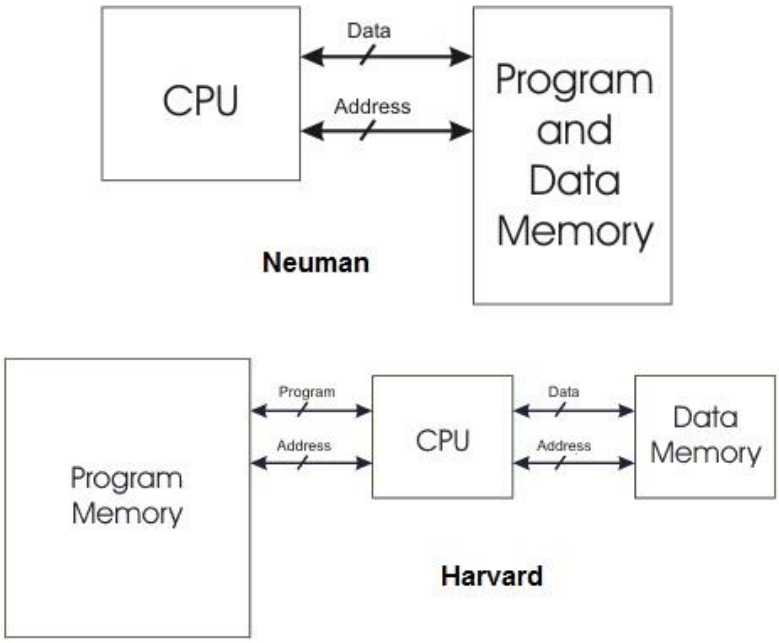
خط لوله Pipelining

مراحل اجرای یک دستور که توسط کاربر نوشته می شود به این صورت است که بعد از روشن شدن میکرو اولین آدرس حافظه فلش که حاوی اولین دستورالعمل است واکشی می شود . بعد از واکشی ترجمه یا رمز گشایی می شود ، معماری این میکرو به صورتی طراحی شده است که قابلیت واکشی (Fetch) ، رمز گشایی (Decode) و اجرای دستور (Execute) را به صورت **پشت سر هم** را دارد . به عبارت دیگر وقتی یک دستور در مرحله اجراست دستور بعدی در مرحله رمز گشایی و دستور بعد از آن در مرحله واکشی قرار دارد تا همزمان همه واحد ها بیکار نباشند و سرعت پردازش چند برابر شود . به طور کلی معماری پایپ لاین در پردازنده را می توان به خط تولید کارخانه تشبیه کرد طوری که وقتی خط تولید شروع به کار می کند هیچ کدام از بخش ها بیکار نمی مانند و هر کدام وظایف مربوط به خود را به طور مداوم انجام می دهند. در شکل زیر نحوه انجام پایپ لاین توسط میکرو نشان داده شده است . در سیکل اول کلاک ، تنها دستور اول از حافظه واکشی می شود ، در سیکل دوم کلاک ، دستور اول که واکشی شده بود ، به واحد رمزگشایی میرود و دستور دوم واکشی می شود . در سیکل سوم کلاک دستور اول وارد مرحله اجرا می شود در حالی که دستور دوم به واحد رمزگشایی میرود و دستور سوم واکشی می شود و همین طور این کار دائما تکرار می شود pipelining . یکی از مزیت های بسیار مهم معماری RISC محسوب می شود که باعث افزایش سرعت می شود . پایپ لاین در میکروکنترلر های مختلف میتواند مراحل متفاوتی داشته باشد میکروکنترلرهای با ۳ ، ۵ ، ۸ و حتی ۱۳ مرحله پایپ لاین نیز وجود دارند اما در میکروکنترلرهای AVR ، پایپ لاین **فقط ۲ مرحله Fetch و Execute را دارد.**

Instr. No.	Pipeline Stage						
1	IF	ID	EX				
2		IF	ID	EX			
3			IF	ID	EX		
4				IF	ID	EX	
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

معماری حافظه در AVR

در میکروکنترلر های AVR از معماری "هاروارد" استفاده شده است بطوریکه در این معماری حافظه میکروکنترلر به دو قسمت حافظه برنامه (از نوع Flash) و حافظه داده (از نوع EEPROM) تقسیم می شود و هر کدام از این دو حافظه از گذرگاه مجزا استفاده میکنند . معماری هاروارد در مقایسه با معماری سنتی فون نیومن سریع تر است. شکل زیر تفاوت دو معماری را نشان می دهد.



حافظه داده SRAM

این حافظه جهت ذخیره سازی موقت اطلاعات در اختیار کاربر قرار می گیرد . این حافظه به طور مستقیم در اختیار CPU نمی باشد ، برای دستیابی به آن از یکی از ۳۲ رجیستر عمومی به عنوان واسط استفاده میشود . یعنی جهت انجام هر عملیات دیتا نمیتواند مستقیما از SRAM به ALU منتقل شود بلکه باید ابتدا به یکی از ۳۲ رجیستر عمومی برود و از آنجا به ALU منتقل شود . در نتیجه برای استفاده از SRAM جهت انجام عملیات زمان بیشتری صرف می شود.

حافظه داده EEPROM

این حافظه که توسط رجیستر های عمومی ۳۲ گانه با CPU در ارتباط است دائمی بوده و با قطع برق از بین نمی رود .از این حافظه میتوان در مواقعی که مقدار یک داده در برنامه مهم بوده و نباید با قطع برق یا ریست شدن از بین برود استفاده کرد.

حافظه برنامه FLASH

این حافظه دائمی پرسرعت برای ذخیره برنامه کاربر استفاده می گردد . در واقع حافظه فلش در میکروکنترلر به دو قسمت تقسیم شده است بخشی برای بوت شدن سیستم و بخشی برای برنامه کاربر مورد استفاده قرار می گیرد. فیوز بیت ها (Fuse Bits) و بیت های قفل (Lock Bits) نیز در قسمت بوت لودر حافظه فلش وجود دارند. در قسمت دیگر این حافظه برنامه کاربر وجود دارد که در هنگام کار میکرو خط به خط خوانده شده و اجرا می شود.

واحد ورودی/خروجی(Input/Output)

این واحد وظیفه ارتباط میکرو با محیط پیرامون را به صورت موازی (پاراللParallel) برقرار می کند . اساسا ارتباط میکرو با محیط پیرامون به دو شکل موازی و سریال صورت می گیرد . واحد ورودی/خروجی به صورت موازی و واحد ارتباط سریال به صورت سریال با محیط پیرامون میکرو در ارتباط است . این واحد که با آن پورت نیز گفته می شود ، دارای چند رجیستر و بافر است و در نهایت به صورت پایه(پینPin) از میکرو خارج می شود . در میکروکنترلرهای AVRبا توجه به نوع و سری میکروکنترلر تعداد آنها بین ۱ تا ۱۰ پورت متفاوت می باشد . پورت ها در AVR به صورت PORTA، PORTB، PORTCو... نامگذاری می شود. هر پورت ۸ بیت دارد که به صورت PORTX.0تا PORTX.7 نامگذاری می شود.

واحد کنترل کلاک ورودی

این واحد وظیفه تامین کلاک میکرو را بر عهده دارد . منابع کلاک میکرو به دو دسته منابع کلاک خارجی و کلاک داخلی تقسیم می شود . همانطور که می دانید پالس کلاک یک پالس منظم و دارای فرکانس ثابت است که تمامی واحد های میکرو با لبه های آن پالس کار می کنند . برای تولید پالس کلاک به دو چیز احتیاج می باشد یکی کریستال و دیگری اسیلاتور . خوشبختانه در این واحد هم اسیلاتور وجود دارد و هم کریستال اما بر حسب نیاز به داشتن فرکانس های مختلف احتیاج به اسیلاتور خارجی و یا کریستال خارجی نیز داریم . بنابراین منابع کلاک در میکروکنترلرهای AVR به صورت زیر است:

1. **منبع کلاک خارجی:** در این حالت از اتصال منبع کلاک با فرکانس مشخص به پایه X1 میکرو استفاده می شود . در این حالت پایه X2 آزاد است.
2. **کریستال خارجی:** در این حالت از اتصال یک کریستال با فرکانس مشخص و دو عدد خازن بین پایه های X1و X2استفاده می شود . جنس بلور این کریستال ها معمولا کریستال کوارتز است . در این حالت از اسیلاتور داخل میکرو استفاده می شود.
3. **RCاسیلاتور داخلی:** RC اسیلاتور داخلی یک خازن و یک مقاومت است که با فرکانس مشخصی نوسان می کند. در این حالت هر دو پایه X1 و X2 آزاد است.

واحد تایمرها و کانترها(Timers & Counters)

در این واحد از یک سخت افزار خاص چندین استفاده متفاوت می شود . این سخت افزار خاص شامل چند رجیستر و چند شمارنده هستند که کارایی های متفاوتی دارند . کاربرد این واحد به عنوان تایمر ، کانتر ، RTCو PWM می باشد . رجیستر اصلی مورد استفاده در این واحد رجیستر تایمر نام دارد . اگر یک پالس ورودی منظم (مانند تقسیمی از پالس کلاک میکرو) به این واحد اعمال کنیم رجیستر تایمر شروع به زیاد شدن کرده و در زمان مشخصی پر و سرریز می شود. بنابراین میتوان با تنظیم مناسب پالس ورودی به این واحد زمان های مختلف را ایجاد کرد. اگر پالس ورودی تایمر را طوری تنظیم کنیم که رجیستر تایمر هر یک ثانیه سر ریز شود ، در این صورت RTC یا زمان سنج حقیقی ساخته می شود. در زمان استفاده از RTC به کریستال ۳۲۷۶۸ هرتز که به کریستال ساعت معروف است نیاز داریم . اگر رجیستر تایمر را قبل از سر ریز شدن ریست کنیم ، در این صورت میتوان با اعمال پالس خروجی از میکرو یک PWM یا مدولاسیون پهنای عرض پالس را روی یکی از پایه های میکرو داشت . اما اگر پالس ورودی به این واحد

نامنظم باشد (مانند پالسی که از بیرون به پایه میکرو اعمال می شود) در این صورت یک شمارنده تعداد پالسهای ورودی روی رجیستر تایمر ساخته می شود.

واحد تایمر سگ نگهبانWhatchdog

این واحد متشکل از یک رجیستر تایمر و یک اسیلاتور است که با فعالسازی آن رجیستر تایمر شروع به افزایش می کند تا اینکه سرریز شود. با سرریز شدن رجیستر تایمر سگ نگهبان میکروکنترلر ریست می شود. کاربرد این واحد جهت جلوگیری از هنگ کردن میکروکنترلر است . در پروژه های دارای اهمیت میکرو نباید هنگ کند زیرا ممکن است خطرات و عواقب بدی داشته باشد . این واحد وظیفه دارد در صورت هنگ نمودن میکروکنترلر بلافاصله آن را ریست کند تا در کسری از ثانیه میکرو دوباره شروع به کار نماید.

واحد کنترل وقفهInterrupt

در هنگام بروز وقفه ، پردازنده کار فعلی خود را رها کرده و به اجرای وقفه مورد نظر می پردازد . علت بوجود آمدن واحد کنترل وقفه این است که باعث می شود پردازنده کمتر درگیر شود . در حالتی وقفه وجود نداشته باشد ، پردازنده مجبور است طی فواصل زمانی مشخصی چندین بار به واحد مورد نظر سرکشی کرده و بررسی کند که دیتای خواسته شده از آن واحد آماده است یا خیر که اغلب آماده نبوده و وقت پردازنده تلف می شود . برای مثال فرض کنید که یک صفحه کلید به ورودی میکرو متصل است ؛ در حالت سرکشی پردازنده باید یکی یکی کلید های صفحه کلید را در فواصل زمانی مشخص سرکشی کند تا در صورت فشرده شدن یک کلید پردازش مورد نظر را انجام دهد ، اما در حالتی که واحد کنترل وقفه فعال باشد ، پردازنده آزاد است و تا زمانی که کلیدی زده نشده است پردازنده به صفحه کلید کاری ندارد . سپس در صورتی که کلیدی زده شود واحد کنترل وقفه یک سیگنال وقفه به پردازنده مبنی بر اینکه ورودی آمده است ارسال می کند تا پردازنده برای یک لحظه کار خود را رها کرده و به صفحه کلید سرویس می دهد ، کلیدی که زده شده را شناسایی کرده و پردازش مورد نظر را روی آن انجام می دهد.

واحد ارتباطی JTAG

JTAGکوتاه شده ی عبارت Joint Test Access Group ، یک پروتکل ارتباطی بر روی دستگاه ها می باشد که این توانایی را ایجاد می کند که بتوان برنامه نوشته شده موجود در میکرو را خط به خط اجرا نمود تا در صورت وجود اشکال در برنامه نویسی آن را برطرف (عیب یابی debug) کرد . همچنین میتوان توسط JTAG حافظه Flash ، EEPROM، فیوز بیت ها و قفل بیت ها را برنامه ریزی (پروگرام Program) کرد.

واحد مبدل آنالوگ به دیجیتال(ADC)

همانطور که میدانید تمامی کمیت های فیزیکی ، آنالوگ هستند . کمیت های آنالوگ برای پردازش توسط میکروکنترلر ابتدا می بایست تبدیل به دیجیتال شوند. تبدیل ولتاژ ورودی آنالوگ به کد دیجیتال متناسب با آن ولتاژ ورودی توسط این واحد انجام می پذیرد . مسائلی که در هنگام کار با این واحد درگیر آن هستیم یکی سرعت نمونه برداری و دیگری ولتاژ مرجع VREF است . ولتاژ مرجع در این واحد به عنوان مرجعی برای سنجش ولتاژ آنالوگ ورودی به کار می رود به صورتی که بازه ی مجاز ولتاژ ورودی بین ۰ تا VREF است. همچنین سرعت نمونه برداری مسئله ی مهمی است که در بروزرسانی سریعتر اطلاعات نقش دارد.

واحد مقایسه کننده آنالوگ

یکی دیگر از امکانات موجود در میکروکنترلرهای AVR واحد مقایسه آنالوگ می باشد که با استفاده از آن می توان دو موج آنالوگ را با هم مقایسه کرد. عملکرد این قسمت مشابه عملکرد آپ امپ در مد مقایسه کننده می باشد با این تفاوت که در صورتی که ولتاژ پایه مثبت از پایه منفی بیشتر باشد ، خروجی مقایسه کننده یک می شود.

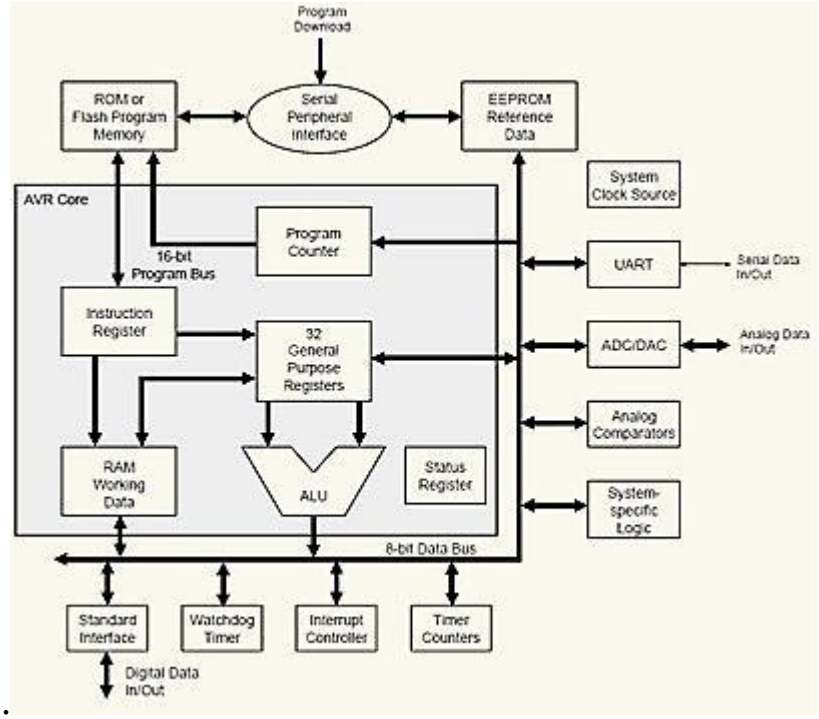
سوالی که ممکن است بوجود آید این است که با وجود مبدل آنالوگ به دیجیتال دیگر چه نیازی به این بخش می باشد؟ در جواب باید گفت سرعت عملکرد این بخش در مقایسه با مبدل انالوگ به دیجیتال بسیار بیشتر بوده و همین سرعت باعث احساس نیاز به چنین بخشی را فراهم کرده است.

واحد ارتباطات سریال

تبادل دیتا با محیط خارجی میکروکنترلر علاوه بر واحد ورودی/خروجی که به صورت موازی است ، می تواند از طریق این واحد به صورت سریال انجام گیرد . مهمترین مسئله در ارتباطات سریال یکی پروتکل ارتباطی و دیگری سرعت ارسال (Baud Rate) است. پروتکل های ارتباطی سریال که توسط میکروکنترلرهای AVR پشتیبانی می شود عبارتند از:

1. پروتکل : spi دارای سرعت بالا می باشد . از طریق این پورت میکروکنترلر را میتوان پروگرام کرد.
2. پروتکل : USART سرعت پایین تری دارد. برای مسافت های طولانی و ماژول های ارتباطی مناسب است.
3. پروتکل : TWI یا پروتکل دوسیمه بیشتر برای ارتباط با المانهای جانبی سرعت پایین است.

در پایان شکل بهتری از واحد های یک میکروکنترلر و ارتباط آنها با یکدیگر را مشاهده می کنید . همانطور که از شکل نیز پیداست در پردازنده های AVR پهنای ارتباطی داده (Data Bus) دارای ۸ بیت و پهنای ارتباطی برنامه (پهنای دستورالعمل ها) دارای ۱۶ بیت می باشد



انواع زبان های برنامه نویسی میکروکنترلرهای AVR

زبان های برنامه نویسی نظیر اسمبلی ، C، C++، بیسیک و. . .

هر کدام از این زبان های برنامه نویسی یک نرم افزار مترجم (کامپایلر) مخصوص به خود را دارند که برنامه در درون آن نرم افزار توسط برنامه نویس نوشته می شود و سپس از طریق سخت افزاری به نام پروگرامر روی میکروکنترلر برنامه ریزی می شود . برای برنامه ریزی میکروکنترلر ها مستلزم دیدگاهی بر تلفیق سخت افزار و نرم افزار هستیم . پس از سالها فکر و بررسی در زمینه برنامه ریزی این نوع المان ها و استفاده از زبان های برنامه نویسی مختلف ، زبان برنامه نویسی C به عنوان یکی از بهترین و منعطف ترین زبان های برنامه نویسی در این زمینه معرفی شد . امروزه به جرات میتوان گفت که زبان برنامه نویسی C را می توان برای برنامه ریزی هر نوع میکروکنترلری به کار برد . زبان برنامه نویسی C با قرار گرفتن در سطح میانی بین زبانهای برنامه نویسی دیگر ، دسترسی کاربران را هم به بخش های سطح پایین و سطح بالا در برنامه نویسی فراهم میکند.

از کامپایلرهای معروف میتوان به AVR Studio ، CodeVision AVRو Bascom AVR اشاره کرد . ما از کامپایلر زبان C به نام CodeVision AVR استفاده می کنیم زیرا استفاده از آن به دلیل وجود ساختار CodeWizard (جادوگر کد) راحت تر است.

برنامه ریزی (پروگرام) کردن میکروکنترلرهای AVR

برنامه ریزی میکروکنترلرهای AVR به سه روش صورت می گیرد : برنامه ریزی سریال ، موازی وJTAG

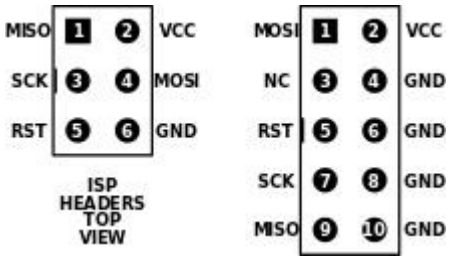
برنامه ریزی موازی سریعتر انجام می گیرد اما نحوه انجام کار آن پیچیده تر است. ضمنا تمامی میکروکنترلرهای AVRاز برنامه نویسی موازی و برنامه ریزی JTAG پشتیبانی نمی کنند . بنابراین برنامه ریزی سریال که از طریق رابط سریال spi صورت می گیرد و تمامی میکروکنترلرهای AVR از آن پشتیبانی می کنند بهترین گزینه است.

در برنامه ریزی سریال حافظه فلش از طریق رابط spi پروگرام می شود و برنامه نوشته شده کاربر درون میکرو قرار می گیرد .برقراری ارتباط بین PC و میکروکنترلر توسط سخت افزار جانبی به نام پروگرامر صورت می گیرد . پروگرامرها انواع مختلفی دارند . پروگرامرها از طریق پورت USB ، پورت سریال و پورت پرینتر می توانند به PC وصل شده و میکروکنترلر روی پروگرامر قرار میگیرد و اینگونه پروگرام می شود. ساده ترین پروگرامر stk300 نام دارد که به پورت

پرینتر وصل می شود . از معروفترین پروگرامرهای موجود در بازار که به پورت USB متصل می شوند میتوان stk500و isp mk2را نام برد.

ISPچیست ؟

یکی از راههای پروگرام کردن میکروکنترلر های AVR روش ISP یا In System Programming می باشد که در آن میکروکنترلر درون مجموعه سیستمی که قرار دارد و بردون خارج شدن از آن پروگرام می شود . در حقیقت در این روش از همان برنامه ریزی سریال با استفاده از پروتکل Spi استفاده می شود که تحت اتصال استاندارد ISP به صورت شکل زیر درآمده است.



پایان فصل چهارم

خسته نباشید ! امیدوارم تا اینجا مباحث مفید بوده باشد. این فصل به علت تخصصی بودن مباحث کمی ممکن است کسل کننده به نظر آید اما در آینده متوجه خواهید شد که چقدر تعریف ساده ، کامل و خوبی از کلیات میکروکنترلر AVR و به خصوص معماری آن ارائه شده است. توصیه می شود ابتدا یکبار با دقت این فصل را مطالعه کنید و بعدا نیز آن را چند بار مرور نمایید.

فصل پنجم : معرفی و راه اندازی میکروکنترلرATmega32

مقدمه

در بخش قبلی آموزش ، معماری و ساختار **کلی** میکروکنترلرهای AVR را با هم بررسی کردیم . برای اینکه شما بتوانید بهترین ، مناسب ترین و کم هزینه ترین میکروکنترلر AVR را برای پروژه خاص خود انتخاب کنید ، شرکت Atmel انواع این میکروکنترلرها را در بسته بندی های گوناگون ، با امکانات متغیر و ساختار داخلی کمی متفاوت با آنچه که در فصل قبل گفته شد را تولید می کند ، به طوری که : تمامی میکروکنترلرهای AVR از نظر معماری کلی خصوصا معماری هسته مرکزی (CPU) تقریبا یکسان بوده اما از نظر مقدار حافظه SRAM ، EEPROM، FLASH و تعداد امکانات جانبی (نظیر پورت ها ، تایمر/کانتر ، ارتباطات سریال و ...) می توانند با یکدیگر متفاوت باشند . میکروکنترلر معروف و پرکاربرد ATMEGA32 که به طور کامل با آن آشنا خواهید شد ، تقریبا اکثر امکانات جانبی را داراست . بنابراین علت اینکه این نوع میکرو را برای تشریح ، راه اندازی و آموزش انتخاب کردیم این است که اولا این میکرو پرقدرت و پرکاربرد است و ثانیا کامل بوده و تقریبا تمامی امکانات جانبی را داراست. در نتیجه مسلط بودن به این میکروکنترلر ، باعث خواهد شد که به تمامی میکروکنترلرهای AVR تسلط لازم را داشته باشید.

خصوصیات ، ویژگی ها و عملکرد ATMega32

میکروکنترلر ATMega32 یک میکروکنترلر ۸ بیتی کم مصرف از تکنولوژی CMOS و از خانواده MEGA AVRاست که بر اساس معماری RISC بهبود یافته می باشد . با اجرای دستورات قدرتمند در یک سیکل کلاک این میکرو سرعتی معادل 1 MIPS (Millions Instruction Per Second) در هر MHz (یعنی زمانی که فرکانس کاری میکرو ۱ مگاهرتز است ، میکرو می تواند یک میلیون دستور العمل در هر ثانیه انجام دهد) فراهم میکند که به طراح سیستم این اجازه را می دهد تا توان مصرفی میکرو را برحسب سرعت پردازش مورد نیاز پروژه مدیریت کند.

خصوصیات و ویژگی های این میکرو که ترجمه صفحه اول **دیتاشیت Atmega32** می باشد ، به شرح زیر است:

- کارایی بالا ، توان مصرفی کم ، یک میکروکنترلر ۸ بیتی از خانوادهAVR
- دارای معماری RISC بهبود یافته

– دارای ۱۳۱ دستورالعمل قدرتمند که اکثر آنها تنها در یک سیکل کلاک اجرا می شوند

– دارای ۳۲ رجیستر عمومی ۸ بیتی همه منظوره

– عملکرد کاملا پایدار

– سرعتی حداکثر تا ۱۶ MIPSدر فرکانس ۱۶MHz

– دارای ضرب کننده جداگانه داخلی که در دو کلاک سیکل عمل ضرب را انجام می دهد

- دارای حافظه غیر فرار برای برنامه و دیتا

– ۳۲ کیلوبایت حافظه فلش داخلی قابل برنامه ریزی

پایداری : تا ۱۰،۰۰۰ بار نوشتن و پاک کردن

– ۱۰۲۴ بایت حافظهEEPROM

پایداری : تا ۱۰۰،۰۰۰ بار نوشتن و پاک کردن

- ۲ کیلوبایت حافظه SRAM داخلی
- دارای قفل برنامه برای حفاظت نرم افزاری از حافظه

- قابلیت ارتباط به صورت JTAG تحت استاندارد(IEEE std. 1149.1)

- قابلیت مشاهده تمامی رجیسترها و متوقف کردن برنامه روی دستور خاص جهت عیب یابی
- برنامه ریزی حافظه فلش ، EEPROM، فیوزبیت ها و بیت های قفل (Lock Bits) از طریق ارتباط JTAG

- خصوصیات جانبی میکروکنترلر

- دو تایمر/کانتر ۸ بیتی و یک تایمر/کانتر ۱۶ بیتی
- یک کانتر زمان حقیقی (RTC) با اسیلاتور جداگانه
- ۴ کانال برایPWM
- ۸ کانال برای مبدل آنالوگ به دیجیتال ۱۰ بیتی

۸ کانال معمولی یک طرفه

۷ کانال تفاضلی (که این قابلیت فقط در بسته بندی TQFP وجود دارد(

دارای ۲ کانال تفاضلی با گین قابل برنامه ریزی ۱ ، ۱۰ و ۲۰۰ برابر کننده

- قابلیت ارتباط با پروتکل سریال دوسیمهI2C
- دارای پورت USART سریال قابل برنامه ریزی
- قابلیت ارتباط سریال SPI به صورت Master یاSlave

- دارای تایمر قابل برنامه ریزی Watchdog با اسیلاتور داخلی جداگانه
- دارای مقایسه کننده آنالوگ داخلی

• خصوصیات ویژه میکروکنترلر

- ریست خودکار میکرو در هنگام روشن شدن
- شناسایی ولتاژ تغذیه ورودی قابل برنامه ریزی و ریست خودکار میکرو
- دارای اسیلاتور RC داخلی کالیبره شده
- منابع وقفه داخلی و خارجی
- دارای ۶ حالت خواب (Sleep Mode) برای کم مصرف شدن میکرو

• ورودی/خروجی و بسته بندی

- دارای ۳۲ خط ورودی/خروجی قابل برنامه ریزی
- در بسته بندی های مختلف ۴۰پایه PDIP ،

۴۴ پایه TQFP و ۴۴ پایهMLF

• ولتاژهای کاری

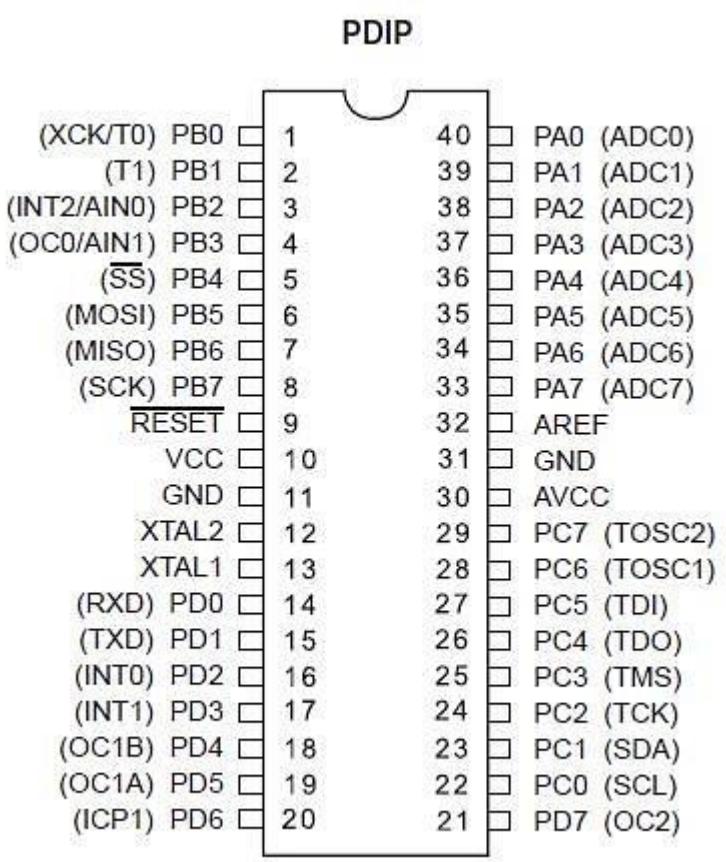
- ۷ تا ۵،۵ ولت درATMega32L
- ۵ تا ۵،۵ ولت درATMega32

• فرکانس های کاری

- ۰ تا ۸ MHzبرایATMega32L
- ۰ تا ۱۶ MHzبرایATMega32

تشریح عملکرد پایه ها درATMEGA32

همانطور که مشاهده کردید این میکرو در سه بسته بندی مختلف تولید و عرضه شده است که بسته بندی TQFP و MLFبه صورت SMD (تکنولوژی روی سطح برد) برای استفاده در pcb (فیبر مدار چاپی) و بسته بندی PDIPبه صورت DIP (پایه دار) مناسب برای استفاده در بردبرد می باشد. بسته بندی PDIP این میکرو که در شکل زیر مشاهده می کنید ۴۰ پایه دارد. دارای ۴ پورت ورودی/خروجی (PA تا PD) که هر پورت آن دارای ۸ بیت است (از ۰ تا ۷) . منظور از پورت همان ورودی/خروجی های موازی (پارالل) واحد I/Oمی باشد.



پایه های ۱ و ۲ (T0 و T1) : به ترتیب مربوط به ورودی کلاک خارجی تایمر ۰ و تایمر ۱ می باشد.

پایه های ۳ و ۴ (AIN0 و AIN1) : به ترتیب پایه مثبت و منفی واحد مقایسه کننده آنالوگ می باشد.

پایه های ۵ تا ۸ (MISO ، MOSI، SCK، SS) : این پایه ها مربوط به ارتباط spi (واحد ارتباط سریال) می باشد.

پایه ۹ (RESET) : پایه ریست میکرو است که تا زمانی که به منطق صفر وصل شود میکرو در حالت ریست میماند.

پایه ۱۰ (VCC) : پایه تغذیه مثبت دیجیتال

پایه ۱۱ (GND) : پایه تغذیه منفی یا زمین دیجیتال

پایه های ۱۲ و ۱۳ (XT1 و XT2) : ورودی های واحد کلاک میکرو کنترلر است.

پایه های ۱۴ و ۱۵ (RXD و TXD) : به ترتیب گیرنده و فرستنده دیتا مربوط به ارتباط USART (واحد ارتباط سریال) می باشد.

پایه های ۱۶ ، ۱۷ و ۳ (INT0 ، INT1 ، INT2) : به ترتیب مربوط به وقفه های خارجی صفر ، یک و دو است.

پایه های ۴ ، ۱۹ ، ۱۸ ، ۲۱ (OC0 ، OC1A، OC1Bو OC2) : مربوط به خروجی های واحد تایمر/کانتراست.

پایه های ۲۲ و ۲۳ (SCL ، SDA) : مربوط به پروتکل دوسیمه (واحد ارتباط سریال) می باشد.

پایه های ۲۴ تا ۲۷ (TCK ، TMS، TDO، TDI) : مربوط به ارتباط JTAG می باشد.

پایه های ۲۸ و ۲۹ (TOSC1 و TOSC2) : مربوط به واحد RTC می باشد . در زمان استفاده از RTC به این دو پایه کریستال ۳۲۷۶۸ هرتز وصل می‌شود.

پایه ۳۰ (AVCC) : ولتاژ تغذیه آنالوگ واحد ADC است.

پایه ۳۱ (GND) : زمین آنالوگ واحد ADC است.

پایه ۳۲ (AREF) : پایه مربوط به ولتاژ مرجع قسمت ADC می باشد.

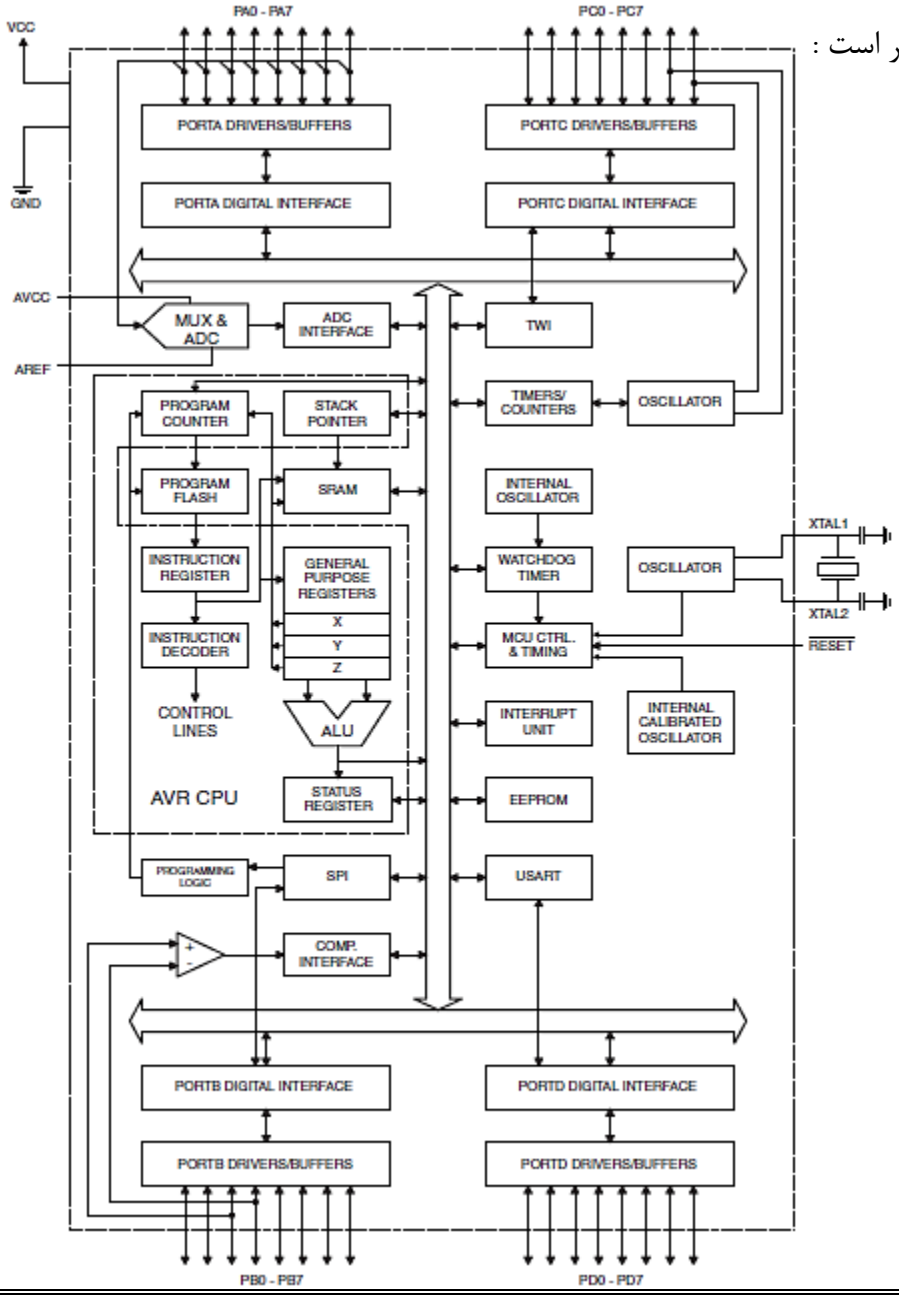
پایه های ۳۳ تا ۴۰ (ADC0 تا ADC7) : کانال های ورودی آنالوگ مربوط به واحد ADC می باشد.

نکته : همانطور مشاهده می شود بعضی از پایه ها دو یا سه عملکرد دارند که همزمان نمیتوان از چند عملکرد استفاده کرد و در آن واحد تنها یک کاربرد از آنها استفاده می گردد . مثلا اگر از پورت A به عنوان ورودی/خروجی پارالل استفاده شود دیگر نمیتوان از قابلیت واحد ADC میکروکنترلر استفاده نمود.

نکته : همانطور که میدانید میکروکنترلرهای AVR میکروکنترلرهای ۸ بیتی هستند ، بنابراین پهنای باس داده ، همه رجیسترها ، پورت های ورودی/خروجی همگی ۸ بیتی هستند.

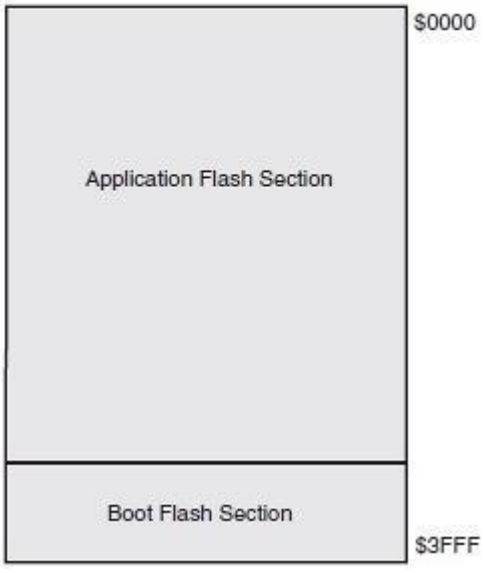
معماری و ساختار داخلی میکروکنترلرAtmega32

همانطور که از پایه ها و ویژگی های میکروکنترلر پیداست ، این میکروکنترلر دارای واحد های زیر می باشد و معماری داخلی آن مطابق شکل زیر است :



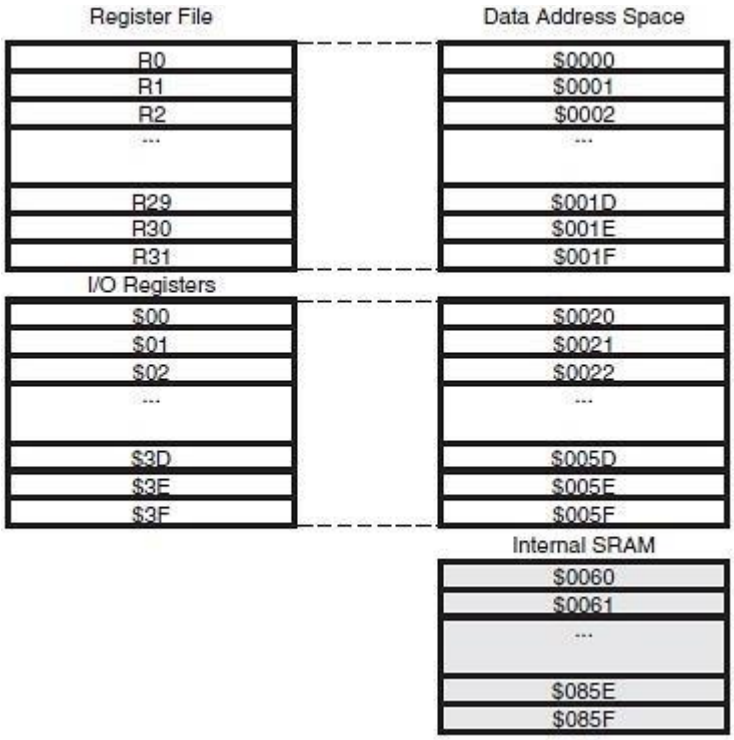
- **واحد پردازش مرکزی (CPU)** : دارای ۱۳۱ دستورالعمل می باشد که در صفحه ۳۲۷ دیتاشیت همه ۱۳۱ دستور العمل به زبان اسمبلی آورده شده است . معماری این واحد دقیقا مشابه گفته شده در فصل قبل می باشد . این واحد در شکل فوق با نقطه چین (AVR CPU) مشخص شده است . ضمنا پهنای رجیستر شمارنده برنامه (Program Counter) در این میکروکنترلر ۱۴ بیت می باشد.

- **واحد حافظه برنامه Flash** : این واحد که در شکل فوق با نام Program Flash در کنار CPU نشان داده شده است ، در این میکروکنترلر دارای ۳۲ کیلوبایت طول و ۸ بیت عرض برای ذخیره برنامه می باشد. برنامه نوشته شده به زبان C بعد از کامپایل شدن درون این واحد به یکی از سه صورت موازی (پارالل) ، JTAG و Spi پروگرام می شود تا توسط CPU خط به خط خوانده و اجرا شود . همچنین در هنگام راه اندازی میکروکنترلر (بوت شدن) ، تنظیمات راه اندازی میکروکنترلر شامل فیوز بیت ها (Fuse Bits) و بیت های قفل (Lock Bits) که در بخشی از حافظه Flash به نام Boat Loader هستند ، اعمال می شوند . بنابراین حافظه فلش از دو قسمت برنامه (Application) و بوت (Boat) تشکیل می شود که در شکل زیر نشان داده شده است . سمت چپ شکل آدرس خانه های حافظه را نشان می دهد که از ۰۰۰۰ در مبنای هگز تا 3FFF هگز نامگذاری می شود. از آن جایی که همه دستورالعمل ها در میکروکنترلرهای AVRتعداد ۱۶ یا ۳۲ بیت دارند ، حافظه فلش در این میکروکنترلر به صورت ۱۶ کیلوبایت طول در ۱۶ بیت عرض سازماندهی شده است . برای همین رجیستر PC در cpu دارای ۱۴ بیت عرض می باشد تا بتواند هر 16Kخانه حافظه برنامه را آدرس دهی نماید.



نکته : از حافظه برنامه Flashدر صورتی که حجم برنامه کم بوده باشد و فضای خالی موجود باشد ، میتوان برای ذخیره دائمی اطلاعات (یعنی به عنوان حافظه داده) نیز استفاده نمود.

- **واحد حافظه داده SRAM** : این واحد که در نزدیکی CPU قرار دارد دارای حجم ۲ کیلوبایت و پهنای ۸ بیت می باشد . به طوری که ۳۲ خانه اول در آن مربوط به General Register File یا همان رجیسترهای عمومی که در حقیقت جزو واحد CPU بوده و به طور مستقیم با واحد CPU در ارتباط است . ۶۴ خانه بعدی حافظه SRAM مربوط به نگهداری و ذخیره کلیه رجیسترها می باشد . رجیسترهای تنظیمات تمامی واحدها (واحد ورودی/خروجی ، واحدهای سریال ، تایمرها و کانترها و ...) در حقیقت در این قسمت قرار دارد . و این یعنی همه واحد ها از SRAM دستور کار می گیرند . برای مشاهده لیست تمامی رجیسترهای موجود در این قسمت به صفحه ۳۲۵ دیتاشیت مراجعه نمایید . بقیه خانه های حافظه همان واحد SRAM اصلی می باشد که جهت نگهداری اطلاعات و داده های برنامه که یا دائما تغییر می کنند (مثلا متغیرهایی که در برنامه توسط کاربر تعریف می شود) یا داده هایی که در زمان محدودی به آنها نیاز داریم (مثلا متغیرهای توابعی که بعد از پایان کار تابع به طور خودکار از بین می رود) می باشد . در شکل زیر نحوه سازماندهی این حافظه را مشاهده می کنید . در شکل سمت راست حافظه SRAM را به صورت خانه هایی که آدرس آنها داخلش نوشته شده است و در شکل سمت چپ معادل آن خانه ها را مشاهده می کنید.



نکته : فقط با قطع تغذیه محتویات SRAM از بین میرود و در صورتی که میکروکنترلر را Reset دستی یا خارجی کنیم محتوای حافظه SRAM صفر نمی شود.

- **واحد حافظه داده EEPROM** : این حافظه که در این میکروکنترلر ، ۱ کیلوبایت (۱۰۲۴ بایت) است ، جزء حافظه های ماندگار می باشد که در صورت قطع تغذیه میکروکنترلر پاک نمی گردد و میکروکنترلر در هر زمان می تواند اطلاعاتی را در این حافظه بنویسد و یا اطلاعاتی را از آن بخواند با این تفاوت که خواندن و نوشتن در حافظه eeprom زمان تاخیر طولانی تری از حافظه های SRAM و Flash دارد .از این حافظه زمانی استفاده می شود که میکروکنترلر باید دیتایی را در خود ثبت کند و بعدا آن دیتا را به کاربر اعلام کند و در صورتی که میکروکنترلر Reset شد با تغذیه آن قطع گردید داده ذخیره شده از بین نرود.

- واحد ورودی/خروجی
- واحد کنترل کلاک
- واحد تایمر/کانتر
- واحدWathdog Timer
- واحد کنترل وقفه
- واحد ارتباطیJTAG
- واحد مبدلADC
- واحد مقایسه کننده
- واحد ارتباطیspi
- واحد ارتباطیUSART
- واحد ارتباطیTWI

تذکر : عملکرد ، معماری و رجیسترهای بقیه واحدهای جانبی فوق الذکر در جای خود کاملا تشریح خواهد شد.

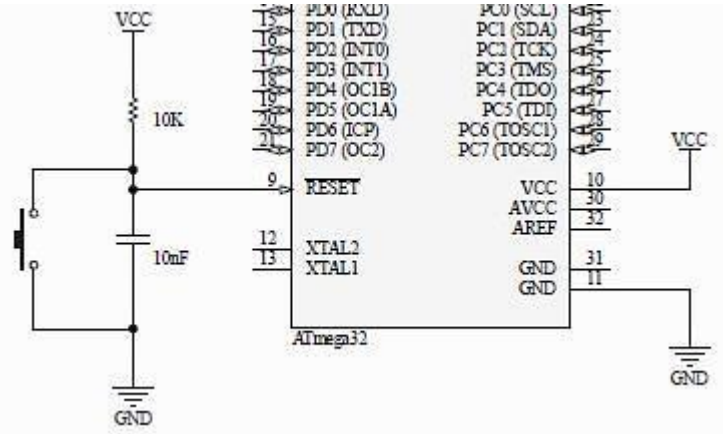
ساختار برنامه میکروکنترلر به زبان C

برنامه میکروکنترلر باید توسط برنامه نویس روی یک کامپایلر نوشته شود و سپس توسط پروگرامر روی میکرو پروگرام شود .برنامه ای که نوشته می شود باید طوری نوشته شود که وقتی روی آی سی پروگرام شد دائما اجرا شود و هیچگاه متوقف نشود . راه حل این مسئله قرار دادن کدهای برنامه درون یک حلقه نامتناهی است . این عمل باعث می شود تا میکروکنترلر هیچگاه متوقف نشود و بطور مداوم عملکرد مورد انتظار را اجرا کند. بنابراین ساختار یک برنامه به زبان C که قرار است در کامپایلر CodeVision نوشته شود به صورت زیر در می آید.

```
#include < HeaderFiles.h >

محل معرفی متغیرهای عمومی ، ثوابت و توابع

void main (void)

{

کدهایی که در این محل قرار میگیرند فقط یکبار اجرا می شوند

معمولا مقدار دهی اولیه به رجیستر ها در این ناحیه انجام می شود

while(1)

  {

کدهایی که باید مدام در میکروکنترلر اجرا شوند

  }

}
```

حداقل سخت افزار راه اندازی میکروکنترلر Atmega32

برای راه اندازی این میکرو کافی است پایه های تغذیه دیجیتال (۱۰ و ۱۱) را به منبع تغذیه با ولتاژ مناسب وصل کرده و همچنین میبایست پایه RESET را توسط یک مقاومت مناسب (۱۰ کیلواهم) به مثبت تغذیه وصل نماییم (Pull Up کنیم) برای اینکه بتوانیم به صورت دستی میکرو را هر زمان که خواستیم ریست کنیم ، احتیاج به یک سوئیچ یا Push Button داریم . بنابراین مدار کامل شده به صورت زیر است. علت وجود خازن در مدار زیر جلوگیری از نوسانات ولتاژ در لحظه اتصال کلید و همچنین برای آرام آرام زیاد شدن ولتاژ در هنگام شروع به کار مجدد میکرو است. مقدار خازن و مقاومت میزان زمان افزایش ولتاژ تا Vcc در هنگام وصل منبع تغذیه را تعیین می کند که در پروژه های حرفه ای حتما 10Kو 10nانتخاب می شود. پس از اتصال میکرو به صورت شکل زیر ، میکرو روشن شده و طبق برنامه ای که کاربر در حافظه فلش آن پروگرام کرده است (توسط پروگرامر و نرم افزار کامپایلر) ، شروع به کار می کند.



نکته : پایه های ۳۰ ، ۳۱ و ۳۲ به ترتیب ولتاژ تغذیه آنالوگ ، زمین آنالوگ و ولتاژ مرجع برای واحد ADCمی باشند و تنها در هنگام استفاده از واحد ADC باید متصل شود.

تشریح عملکرد و رجیسترهای واحدهای میکروکنترلرAtmega32

هر یک از واحدهای گفته شده در Atmega32 به جز CPU ، رجیسترهایی دارد که تنظیمات واحد مورد نظر را مشخص می کند . برنامه نویس باید رجیسترهای مربوط به هر واحد را شناخته و آنها را بسته به پروژه مورد نظر به درستی مقدار دهی نماید . در ادامه به معرفی هر یک از واحد ها و رجیسترهای مربوط به هر واحد خواهیم پرداخت . در این فصل ابتدا با رجیسترهای واحد I/O آشنا خواهیم شد و سپس در فصل هشتم بعد از آموزش CodeVisionو برنامه نویسی C دوباره بقیه واحد ها را ادامه خواهیم داد.

رجیسترهای واحد I/O در AVR:

تمامی میکروکنترلرهای AVR از جمله Atmega32دارای واحد I/O (ورودی/خروجی) می باشند . در ATmega32چهار پورت ورودی/خروجی وجود دارد که برای هر یک پین هایی خروجی درنظر گرفته شده است که در تشریح پایه ها نیز گفته شد . هر یک از این ۴ پورت ورودی/خروجی دارای سه رجیستر به شرح زیر است . در نتیجه مجموعا ۱۲ رجیستر ۸ بیتی برای واحد ورودی/خروجی وجود دارد که همگی در حقیقت درون SRAM می باشند که با مقدار دهی آنها در برنامه این واحد کنترل می شود.

- **DDRX**: یک رجیستر ۸ بیتی که مشخص کننده جهت ورودی یا خروجی بودن هر یک از پایه ها است . هر بیت از این رجیستر که ۱ باشد ، جهت پایه متناسب با آن به عنوان خروجی و هر بیت که ۰ باشد پایه متناسب با آن ورودی می گردد . در حالت default همه بیت های این رجیستر ۰ هستند یعنی تا زمانی که پایه ای را خروجی نکرده باشیم همه پایه ها ورودی هستند.

- **PORTX**: در صورت انتخاب شدن رجیستر DDRX به عنوان خروجی از این رجیستر برای ۱ یا صفر کردن منطق پایه خروجی استفاده می شود . هر بیت از این رجیستر که ۱ باشد پایه متناسب با آن منطق ۱ را خارج می کند و هر بیت که ۰ باشد منطق ۰ در خروجی ظاهر می گردد. در حالت default همه بیت های این رجیستر ۰ هستند.

- **PINX**: در صورت انتخاب شدن رجیستر DDRX به عنوان خروجی از این رجیستر برای خواندن منطق پایه های پورت استفاده می شود . هر منطقی که یکی از پایه های پورت داشته باشد ، بیت مورد نظر در رجیستر PINX همان منطق را به خود می گیرد. در حالت default همه بیت های این رجیستر ۰ هستند.

نکته ۱ : به جای X در بالا نام پورت مورد نظر قرار میگیرد (A ، B، C وD)

نکته ۲ : نام تمامی رجیستر ها از جمله رجیسترهای فوق با حروف **بزرگ** نوشته میشوند . بقیه حروف در برنامه همگی کوچک نوشته میشوند.

نکته ۳ : برای مقدار دهی به هر رجیستر در برنامه به زبان C از عملگر مساوی = استفاده میشود.

نکته ۴ : ۰ bبرای مقدار دهی به رجیسترها در مبنای ۲ (باینری) و ۰ xبرای مقدار دهی به رجیسترها در مبنای ۱۶ (هگز) به کار میرود.

مثال:

```
DDRA=255;

DDRA=0b11111111;

DDRA=0xFF;
```

توضیح : هر سه خط فوق در واقع یک کار را انجام می دهند و آن هم خروجی کردن تمامی ۸ بیت پورت Aاست اما با این تفاوت که خط اول در مبنای ۱۰ ، خط دوم به صورت باینری و خط آخر به صورت هگز نوشته شده است . (مبنای هگز راحت تر است)

```
PORTA=0b01010101;

PORTA=0x0F;
```

توضیح : بعد از اینکه پورت A را خروجی تعریف کردیم در خط اول یکی در میان به خروجی ها منطق ۱ تزریق کردیم و در خط دوم نیمه پایینی پورت (۴بیت کم ارزش) را ۱ و بقیه را ۰ کردیم . (زمانی که رجیستر DDR تنظیم میشود همگی منطق ها به صورت defaultروی منطق صفر هستند)

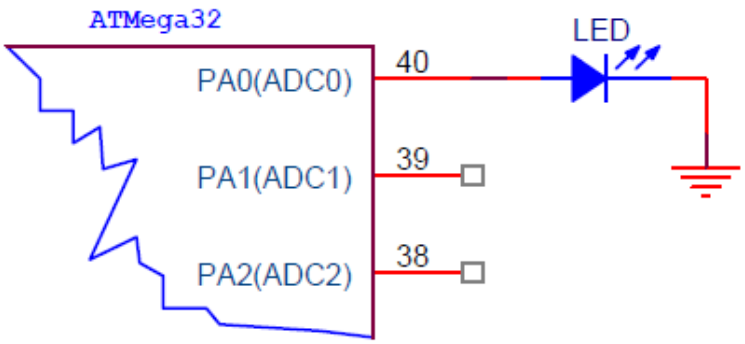
```
PORTA.0=1;

PORTA.5=0;
```

توضیح : علاوه بر ۰ و ۱ کردن کل یک پورت میتوان تنها منطق یکی از پایه ها را تنظیم کرد این کار توسط یک نقطه و نوشتن بیت مورد نظر صورت می گیرد . برای مثال در خط اول بیت صفرم پورت A (پایه ۴۰ میکرو) منطق ۱ و در خط دوم بیت پنجم پورت A (پایه ۳۵) منطق ۰ به خود می گیرد.

تذکر : از رجیستر PIN تنها در حالتی که پایه ورودی باشد (مانند وقتی که کلیدی به عنوان ورودی به پایه وصل باشد) ، استفاده میشود . مثالهای استفاده از رجیستر PIN را زمانی که نحوه اتصال کلید را آموزش دهیم ، خواهیم گفت.

مثال عملی شماره ۱ : برنامه ای بنویسید که LED موجود روی PA.0 را ۴ بار در ثانیه به صورت چشمک زن روشن و خاموش کند . سپس آن را در نرم افزار Proteus شبیه سازی کرده و پس از اطمینان از عملکرد صحیح برنامه توسط نرم افزار CodeVision روی میکروکنترلر Atmega32 پیاده سازی نمایید.

حل : ابتدا سخت افزار مثال خواسته شده را روی کاغذ رسم می نماییم تا درک بهتری از مثال داشته باشیم سپس برنامه را به زبان C می نویسیم.




```
#include <mega32.h>

#include <delay.h>

void main(void)
```

```
{
DDRA.0=1;
while(1){
PORTA.0=1;
delay_ms(250);
PORTA.0=0;
delay_ms(250);
}
}
```

توضیح برنامه:

در خط اول ابتدا کتابخانه مربوط به Atmega32 را اضافه می کنیم (در همه برنامه های کار با Atmega32 باید نوشته شود)

در خط دوم کتابخانه مربوط به تابع تاخیر زمانی (delay.h) را اضافه می کنیم . زمانی که این کتابخانه به برنامه اضافه شود میتوان از تابع delay_ms برای تاخیر زمانی در برنامه استفاده کرد .

در خط پنجم ، پورت PA.0 (بیت صفرم پورت A) به عنوان خروجی تنظیم می شود.

در خط هفتم ، برنامه منطق ۱ را به پورت PA.0 که خروجی شده بود ، تخصصیص می دهد . در نتیجه led در این حالت روشن می شود.

در خط هشتم ، برنامه هیچ کاری انجام نداده و ۲۵۰ میلی ثانیه منتظر می ماند سپس به خط بعدی میرود . در نتیجه این خط led به مدت ۲۵۰ میلی ثانیه روشن میماند.

در خط نهم ، برنامه منطق ۰ را به پورت PA.0 که خروجی شده بود ، تخصصیص می دهد . در نتیجه led در این حالت خاموش می شود.

در خط دهم ، برنامه هیچ کاری انجام نداده و ۲۵۰ میلی ثانیه منتظر می ماند سپس به خط بعدی میرود . در نتیجه این خط ledبه مدت ۲۵۰ میلی ثانیه خاموش میماند.

چون برنامه در حلقه نامتناهی نوشته شده است ، با اجرا شدن برنامه LED موجود روی PA.0 دائما روشن و خاموش می شود و چون زمان تاخیر ۲۵۰ میلی ثانیه است این کار ۴ بار در ثانیه انجام می شود.

پایان فصل پنجم

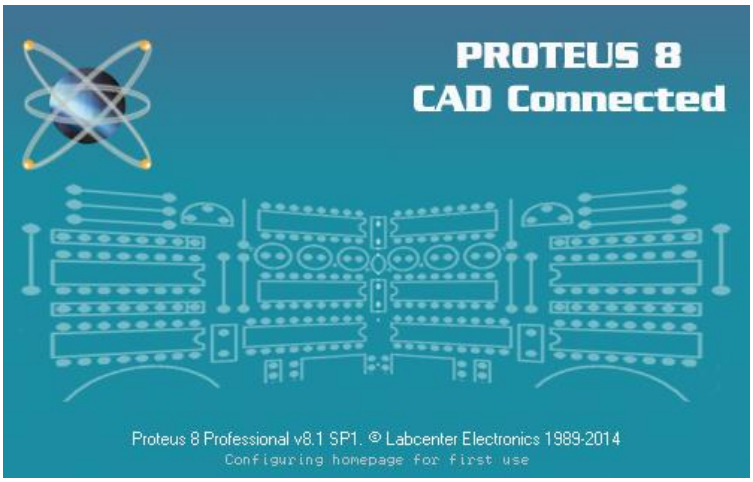
در این فصل کمی جدی تر به قضیه نگاه کردیم و اولین برنامه با میکروکنترلر AVR که چراغ چشمک زن بود را طراحی کردیم. در فصل آینده بعد از آموزش Proteus و CodeVision به حل ادامه مثال (شبیه سازی و پیاده سازی) خواهیم پرداخت . به خود کمی استراحت دهید و اجازه دهید مطالب در ذهنتان تثبیت شود و سپس به فصل بعدی بروید.

فصل ششم : شروع به کار با نرم افزارهای Proteus وCodeVision

مقدمه

در فصل گذشته با معماری و ساختار میکروکنترلر Atmega32 آشنا شدیم. گفتیم مهمترین بخش میکروکنترلر Atmega32که با آن سر و کار داریم رجیسترها هستند. رجیسترها کنترل و تنظیمات تمام بخش های میکروکنترلر را بر عهده دارند و باید به خوبی با نحوه عملکرد آنها آشنا شد. گفتیم که در Atmega32 به تعداد ۶۴ رجیستر برای کنترل مجموعه سیستم میکروکنترلر وجود دارد که در قسمت SRAM حافظه قرار دارد . با تنظیم و مقدار دهی به این رجیسترها در برنامه میتوان برنامه های مختلف را راه اندازی و اجرا نمود . هر چه بیشتر با این رجیسترها کار کرده و مسلط به برنامه ریزی آنها باشید بهتر میتوانید برنامه نویسی کرده و از امکانات میکروکنترلر حداکثر استفاده را ببرید. بعد از رجیسترها ، فیوز بیت ها نیز بخش مهمی از میکروکنترلر می باشد که همانند رجیسترها کنترل و تنظیمات بخش های دیگری از میکروکنترلر نظیر فرکانس کلاک میکرو و ... را بر عهده دارد. در این فصل ابتدا با نرم افزارهای Proteusو CodeVision آشنا خواهید شد و مثال عملی گفته شده در فصل قبل را شبیه سازی و پیاده سازی خواهیم کرد و سپس با فیوز بیت ها و تنظیمات آن آشنا خواهید شد. در فصل های آتی نیز تقریبا تمامی ۶۴ رجیستر ضمن آموزش برنامه نویسی C و انجام پروژه های مربوطه بررسی خواهد شد.

معرفی کلی نرم افزارهای Proteus وCodeVision



پروتئوس (Proteus) نرم‌افزاری برای شبیه سازی مدارات الکترونیک ، بخصوص مدارات مبتنی بر میکروکنترلر می باشد. اصلی ترین کار این نرم افزار شبیه سازی است اما قابل استفاده برای کشیدن بردهای PCB هم بوده و برای این کار هم محیطی مجزا در نظر گرفته شده است. کتابخانه های بسیاری از قطعات الکترونیک جهت طراحی و شبیه سازی مدارات الکترونیکی در این نرم افزار موجود است. این نرم افزار محصول شرکت Lab center Electronics می باشد

کاربردهای این نرم افزار عبارتند از:

- شبیه سازی مدارات آنالوگ و دیجیتال
- تست و آنالیز مدارات با استفاده از ابزار اندازه گیری مجازی مانند اسیلوسکوپ ، مولتی متر ، فانکشن ژنراتور و....
- شبیه سازی تقریبا اکثر مدارات با میکرو کنترلرهای PIC ، AVRو برخی از میکروکنترلرهایARM
- امکان ایجاد و ویرایش قطعات الکترونیکی
- طراحی بردهای PCB یک تا ۱۶ لایه

کدویژن (CodeVision AVR) یک نرم افزار کامپایلر زبان C است که برای برنامه نویسی ، برنامه ریزی (پروگرام) و عیب یابی (debug) کلیه میکروکنترلرهای AVR می باشد . این نرم افزار که دارای محیط برنامه نویسی توسعه یافته نیز می باشد ، بیشتر به علت تولید کدهای اتوماتیک توسط ساختار CodeWizard (جادوگر کد) مشهور شده است. این قابلیت دسترسی راحت به تنظیمات رجیسترهای میکروکنترلرهای AVR را فراهم می کند.

امکاناتی که این نرم افزار فراهم می کند به شرح زیر است :

- کامپایلر استاندارد زبانC
- پشتیبانی از تمام میکروکنترلرهایAVR
- دارای قابلیت تولید خودکار برنامه(CodeWizard)
- پشتیبانی از اکثر پروگرامرهایAVR
- پشتیبانی از ارتباط JTAG برای عیب یابی
- و...

دانلود و نصب نرم افزارهای Proteus وCodeVision

بهترین و کامل ترین ورژن موجود برای نرم افزار Proteus ، نسخه ۸،۱ و برای نرم افزار CodeVision نسخه ۲،۵،۳ می باشد که از لینک های زیر قابل دانلود و نصب می باشند . (این ورژن ها روی ویندوز ۸ نیز نصب می شوند)

CodeVision AVR v2.05.3

Proteus Professional v8.1

تذکر : برای نصب نرم افزار ها حتما طبق راهنمای موجود در پوشه دانلود شده ، با دقت و مرحله به مرحله اقدام نمایید تا بعدا دچار مشکل نشوید.

مراحل کلی انجام یک پروژه میکروکنترلری

به طور کلی وقتی که قرار است یک پروژه با میکروکنترلرهای AVR ، PIC یا ARM انجام دهید ، بعد از مشخص شدن هدف پروژه و صرفه اقتصادی آن مراحل زیر به وجود می آید:

1. **طراحی سخت افزار :** در این مرحله می بایست بر اساس هدف پروژه و شرایط مکانی به کارگیری پروژه ، نوع و مقدار تک تک المان های سخت افزار مورد نیاز طراحی و روی کاغذ آورده شود.
2. **طراحی نرم افزار :** در این مرحله ابتدا الگوریتم یا فلوچارت مورد نیاز رسم و سپس برنامه نویسی مورد نظر بر اساس آن نوشته می شود.
3. **شبیه سازی :** قبل از پیاده سازی عملی ، تست صحت عملکرد مدار در این مرحله توسط نرم افزارهای مناسب (در اینجا Proteus و (CodeVision صورت می گیرد.
4. **پیاده سازی :** پروگرام کردن میکروکنترلر و بستن مدار مورد نظر روی بردبرد در این مرحله صورت می گیرد .
5. **تست و عیب یابی :** با وصل منبع تغذیه به مدار ، تست و عیب یابی مدار در این مرحله صورت می گیرد.
6. **تولید ، ارتقا و بهبود :** در نهایت بعد از بررسی مدار و اطمینان صحت عملکرد آن ، در این مرحله برای مدار مورد نظر فیبر مدار چاپی (pcb) تولید می شود . پشتیبانی پروژه که ارتقا و بهبود عملکرد مدار می باشد ، بعد از تولید پروژه و بازخورد مشتریان بوجود می آید.

تذکر : مراحل فوق بسیار مهم هستند و در پروژه های بزرگ میبایست به ترتیب و با دقت انجام گیرد.

مثال عملی شماره ۱ : برنامه ای بنویسید که LED موجود روی PA.0 را ۴ بار در ثانیه به صورت چشمک زن روشن و خاموش کند . سپس آن را در نرم افزار Proteus شبیه سازی کرده و پس از اطمینان از عملکرد صحیح برنامه توسط نرم افزار CodeVision روی میکروکنترلر Atmega32 پیاده سازی نمایید.

حل:

مرحله اول : طراحی سخت افزار خواسته شده

مرحله دوم : طراحی نرم افزار خواسته شده

```
#include <mega32.h>

#include <delay.h>

void main(void)

{

DDRA.0=1;

while(1){

PORTA.0=1;
```

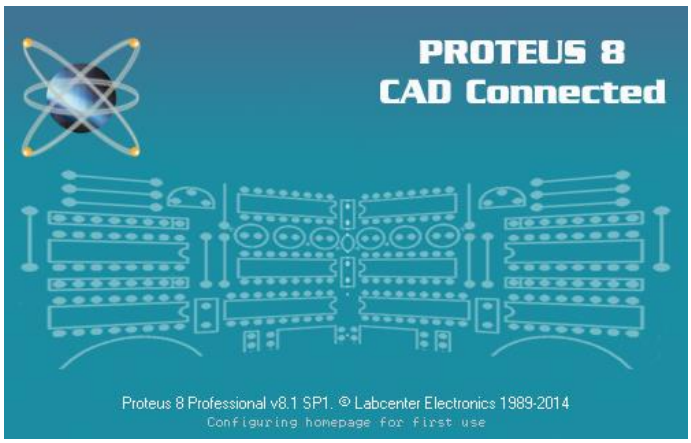
```
delay_ms(250);

PORTA.0=0;

delay_ms(250);

}

}
```

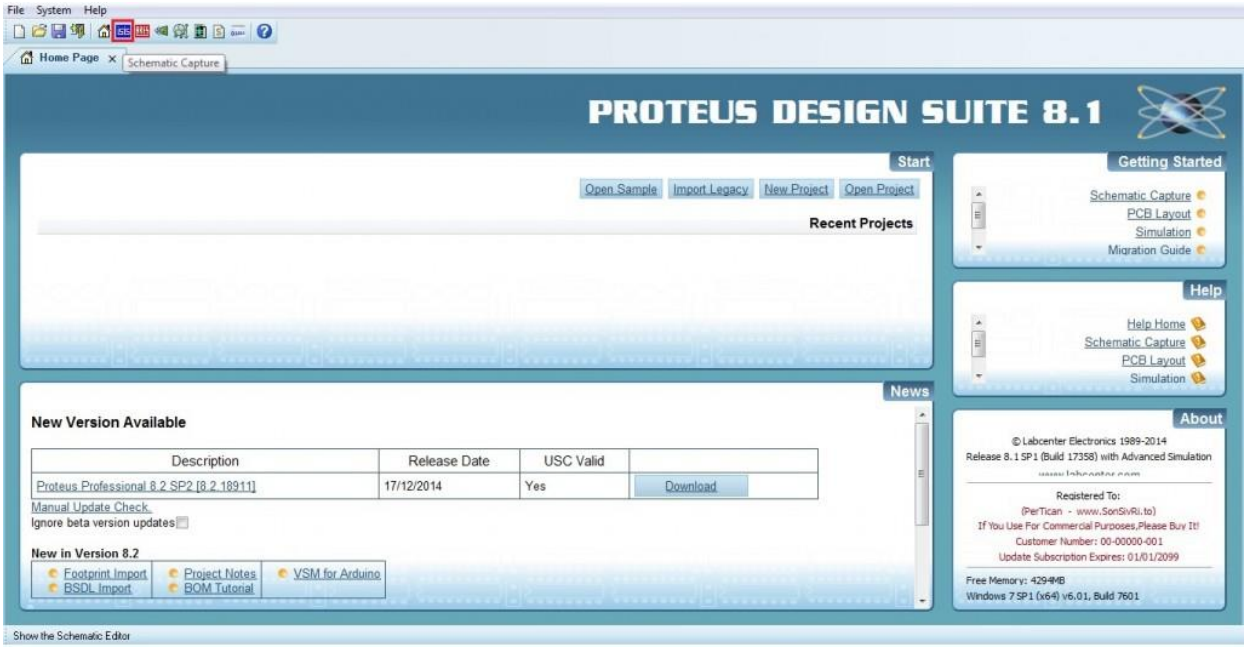
مرحله سوم : شبیه سازی توسط نرم افزارهای Proteus وCodeVision

در این مرحله همیشه ابتدا به نرم افزار پروتئوس مراجعه کرده و سخت افزار طراحی شده را رسم می نماییم . سپس به نرم افزار کدویژن مراجعه کرده و نرم افزار مربوطه را کامپایل و می سازیم . (Build) سپس برنامه ساخته شده توسط کدویژن را در نرم افزار پروتئوس اضافه (add)می کنیم و مدار را شبیه سازی (Run) می کنیم . در صورت جواب گرفتن در این مرحله به مرحله پیاده سازی خواهیم رفت.

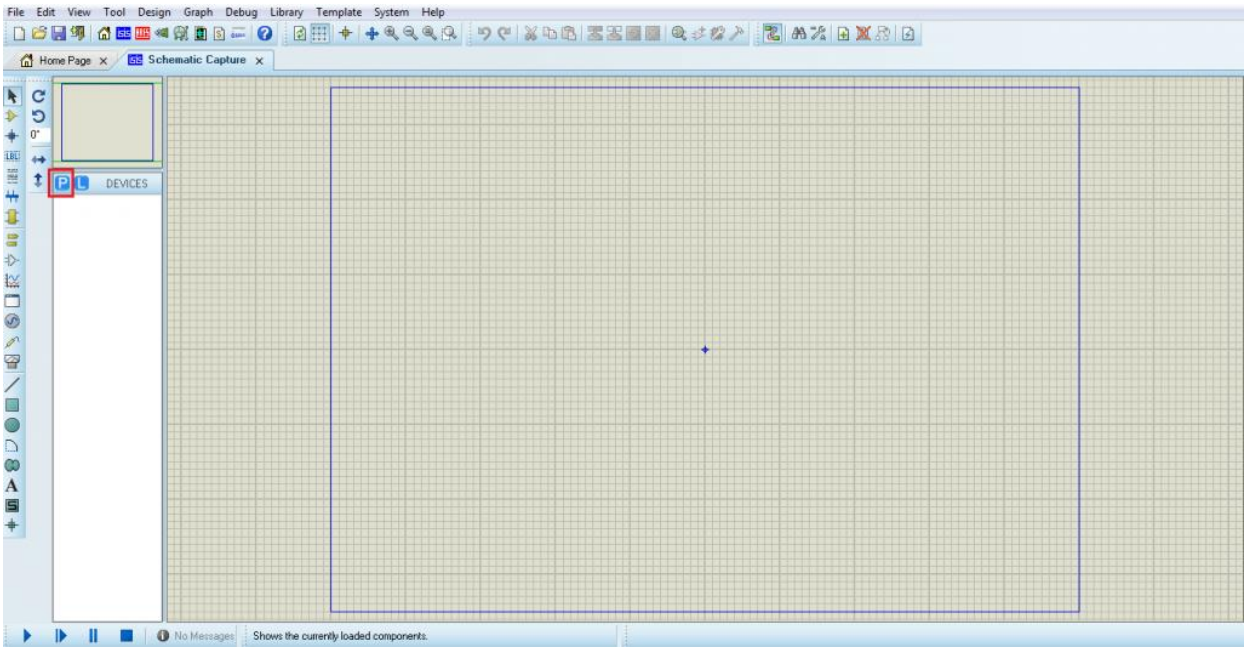
شروع به کار با نرم افزار پروتئوس

پس از نصب نرم افزار بر روی آیکون آن در صفحه دسکتاپ کلیک کرده تا نرم افزار باز شود . زمانی که نرم افزار باز شد با پنجره شکل زیر مواجه می شوید . در این مرحله بر روی آیکون آبی رنگ ISIS موجود در نوار ابزار بالایی که در شکل زیر نیز مشخص شده است ، کلیک کنید تا Schematic captureبرای رسم مدار باز شود.

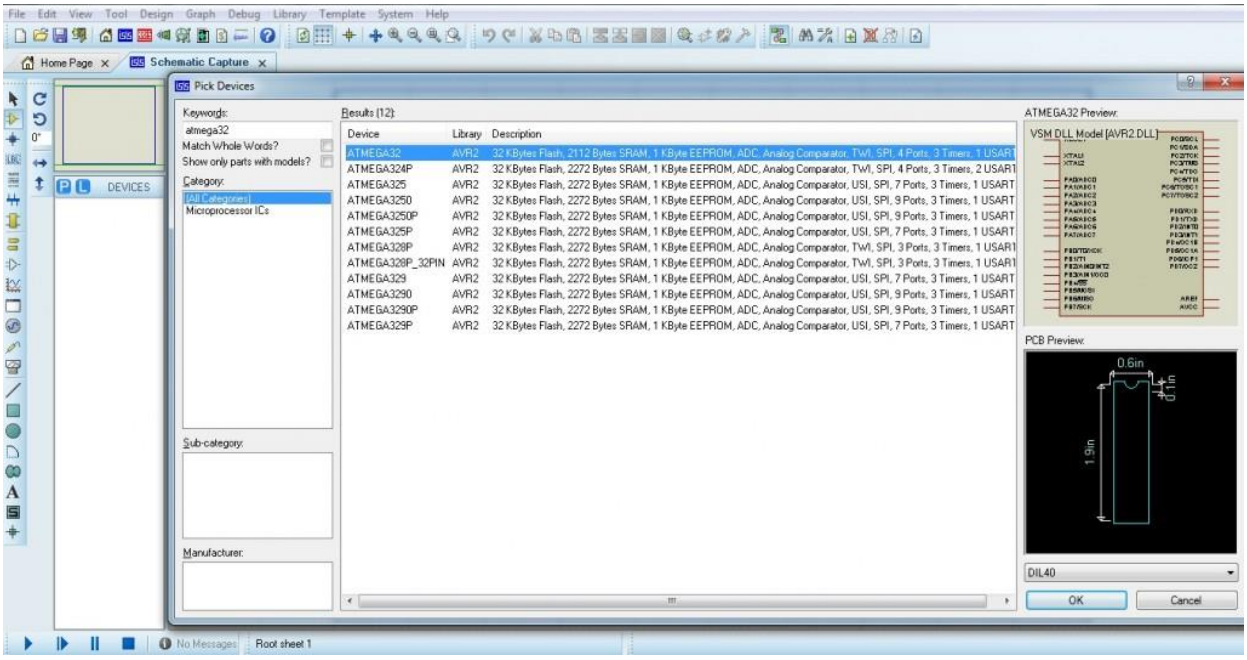


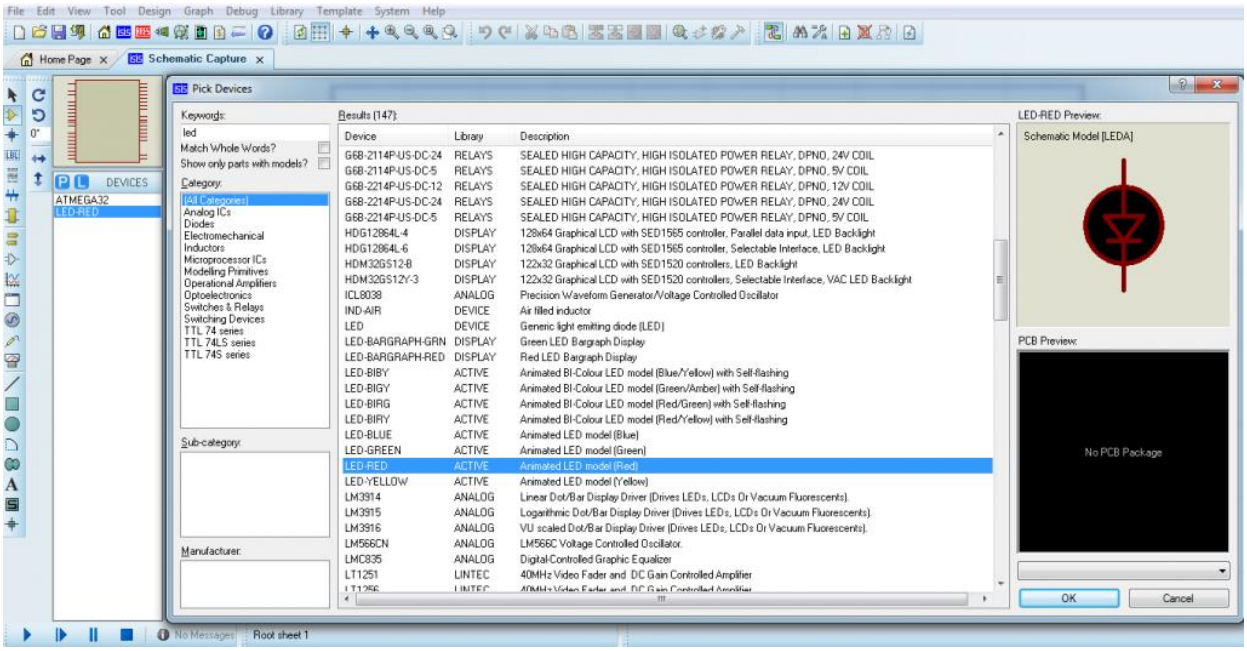


برای انتخاب المان های مداری باید روی آیکون P کلیک کنید تا صفحه جدیدی بنام Pick Devices باز شود.

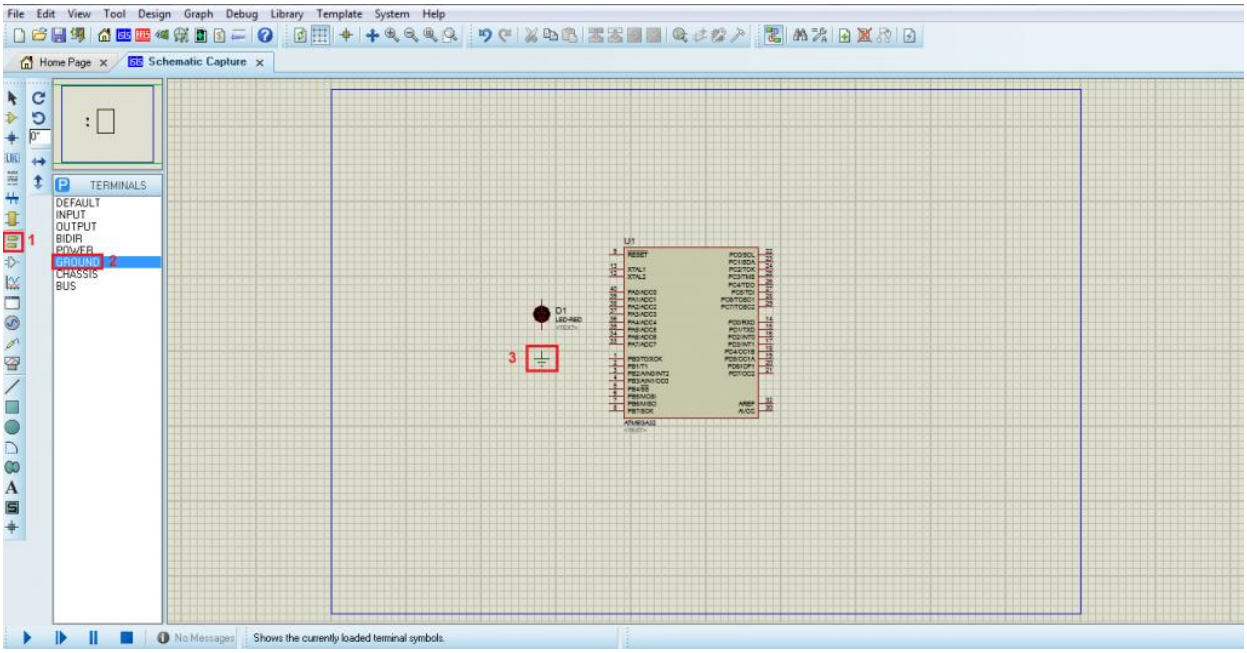


در این صفحه هر المانی نیاز داشته باشید را تایپ کرده و سپس روی نام قطعه دابل کلیک می کنید تا قابل استفاده گردد . برای این مثال یک Atmega32 و یک led نیاز داریم . آنها را تایپ کرده و روی نام آنها دابل کلیک کرده و در نهایت پنجره Pick Devices را با کلیک بر روی Ok می بندیم . مراحل اجرای کار را در زیر مشاهده می کنید.

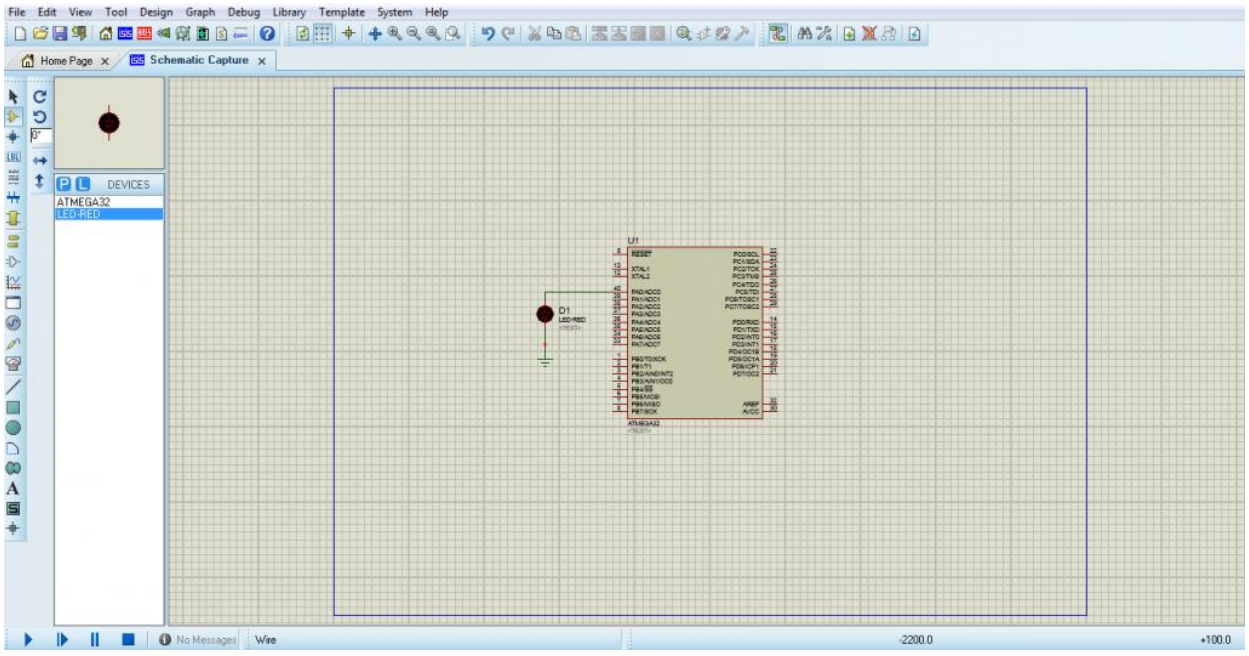




مشاهده می کنید زیر آیکون P قطعاتی که انتخاب کرده بودیم آورده شده است . با کلیک بر روی آنها میکرو و LED را درون کادر آبی رنگ صفحه اصلی در محل مناسب خود قرار داده و مدار را تکمیل می کنیم . به یک زمین (Ground) نیز احتیاج داریم که آن را با کلیک بر روی آیکون Terminals Mode موجود در نوار ابزار سمت چپ و سپس کلیک بر روی GROUND انتخاب کرده و در جای مناسب خود قرار می دهیم.



در نهایت نوبت به سیم کشی مدار می رسد . زمانی که نشانگر ماوس را در محل سیم کشی روی پایه(پین) های LEDیا میکروکنترلر می برید ، مداد سبز رنگی ظاهر می شود ، در همین حال کلیک کنید و سیم کشی مدار را به صورت شکل زیر تکمیل نمایید.



بعد از تکمیل مدار آن را از منوی File و گزینه Save با نام مناسب در یک پوشه جدید ذخیره نمایید . برای اینکه برنامه را بتوان بر روی آی سی ریخته و سپس اجرا کرد میبایست ابتدا باید فایلی که از نرم افزار CodeVision تولید می شود را داشته باشیم . بنابراین در این مرحله به سراغ نرم افزار کد ویژن رفته و پس از نوشتن برنامه برای شبیه سازی عملکرد مدار باز خواهیم گشت.

شروع به کار با نرم افزارCodeVision AVR

پس از نصب و اجرای این نرم افزار برای شروع پروژه ی جدید باید مراحل زیر با دقت و به ترتیب طی شود:

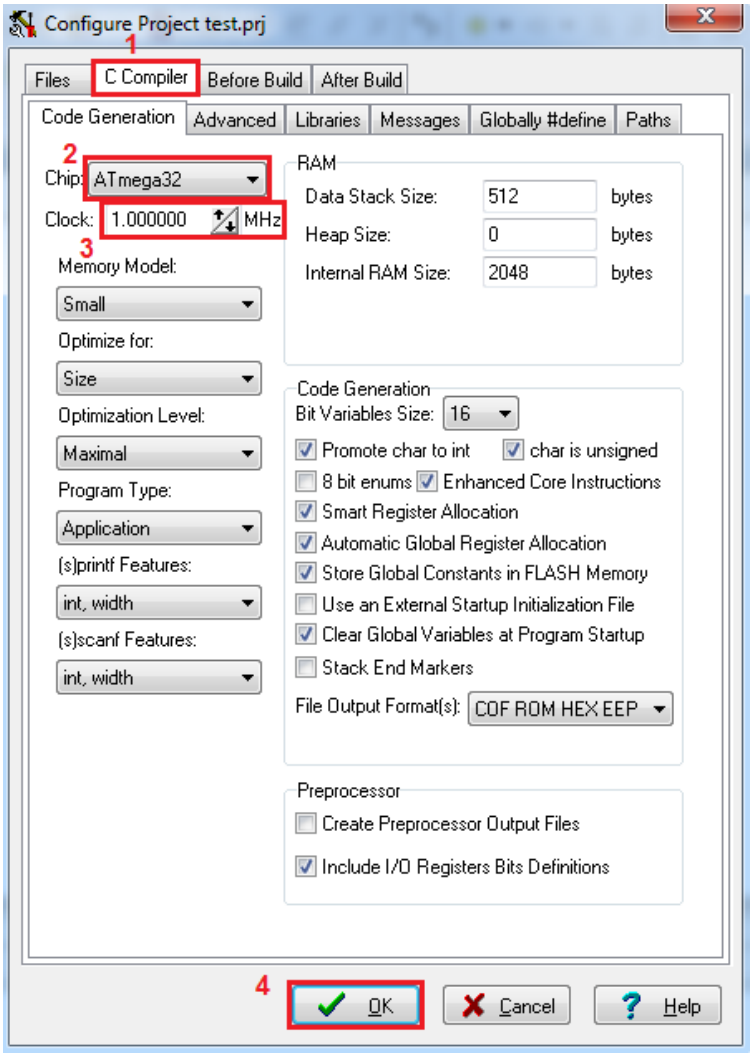
1. ابتدا ممکن است آخرین پروژه ای که کار کرده اید باز باشد آن را از مسیر File و سپس Close All ببندید
2. از منوی File گزینه New را انتخاب کرده و سپس در پنجره باز شده ، Sourceرا انتخاب و Ok کنید.

3. فایل untitled.c باز می شود . در این مرحله می بایست از منوی File و سپس Save as آن را با نام مناسب در همان پوشه ای که فایل پروتئوس قرار دارد ، ذخیره کنید.

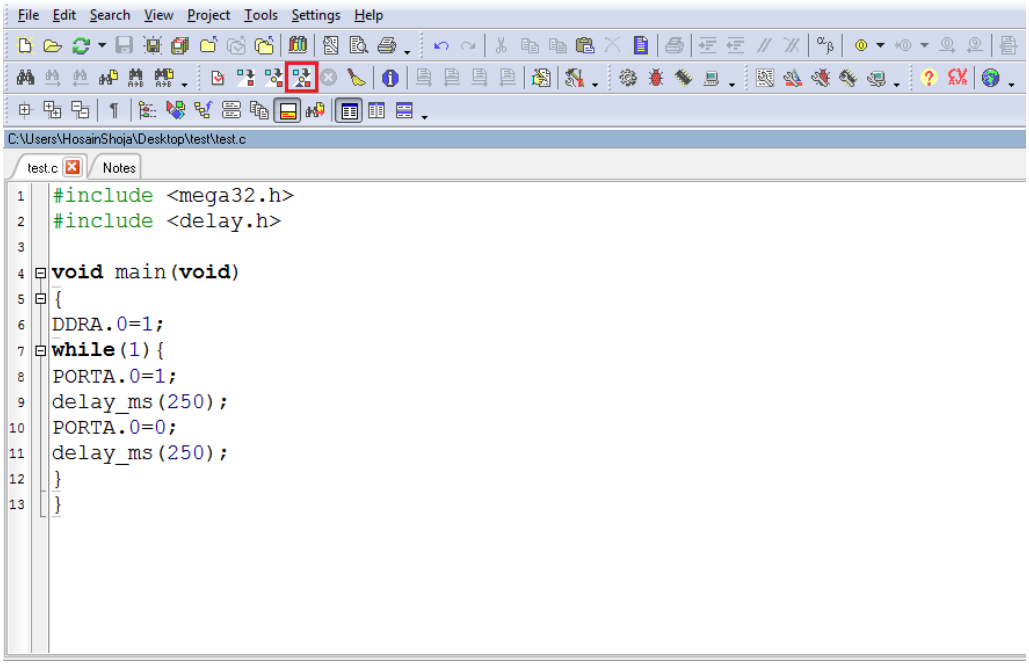
4. از منوی File گزینه New را دوباره انتخاب کرده اما این بار در پنجره باز شده ، Projectرا انتخاب و Ok کنید.

5. پنجره ای مبنی بر اینکه آیا می خواهید از CodeWizard استفاده کنید یا خیر ؟ باز می شود آن را **No** کنید.
6. پنجره Project Configure باز می شود که در این مرحله می بایست فایل ذخیره شده با پسوند C. در مرحله ۳ را با زدن دکمه Add به پروژه اضافه نمود.

7. در این مرحله به سربرگ C Compiler رفته و در قسمت Chip نوع چیپ را روی Atmega32 قرار می دهیم و در قسمت Clock فرکانس کلاک را روی فرکانس کاری میکرو تنظیم می کنیم (برای atmega32 کلاک در حالت default روی ۱ MHZباید قرار گیرد) . در نهایت پنجره Project Configure را Ok کرده و کار پروژه جدید پایان می یابد.

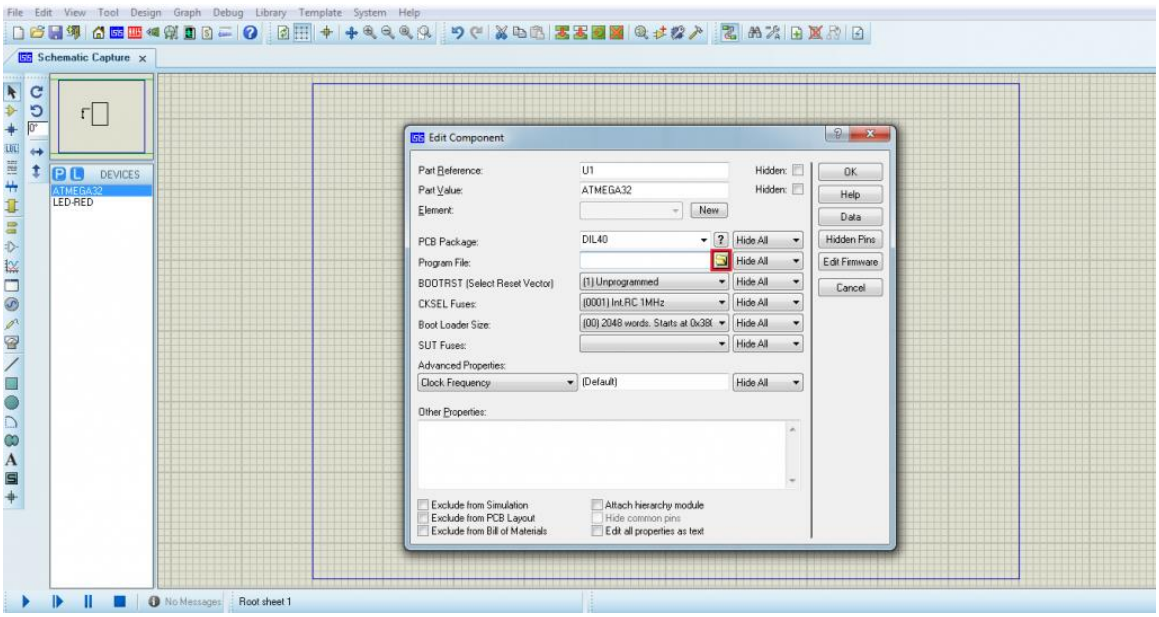


پروژه جدید ساخته شده است حالا باید برنامه دلخواه به زبان C را در این مرحله نوشت . پس در اینجا برنامه نوشته شده مثال شماره ۱ را تایپ می کنیم. بعد از نوشتن برنامه در محیط کدویژن می بایست از منوی Project گزینه Buildرا انتخاب کنید تا برنامه کامپایل و ساخته شود. در صورت بروز error باید ابتدا آنها را برطرف کرده تا برنامه ساخته شود . ساخت برنامه بدین معنی است که فایلی با پسوند Hex ساخته میشود که این فایل به زبان میکرو است و میتوان آن را روی میکرو پروگرام کرد یا برای شبیه سازی در proteus به کار برد.

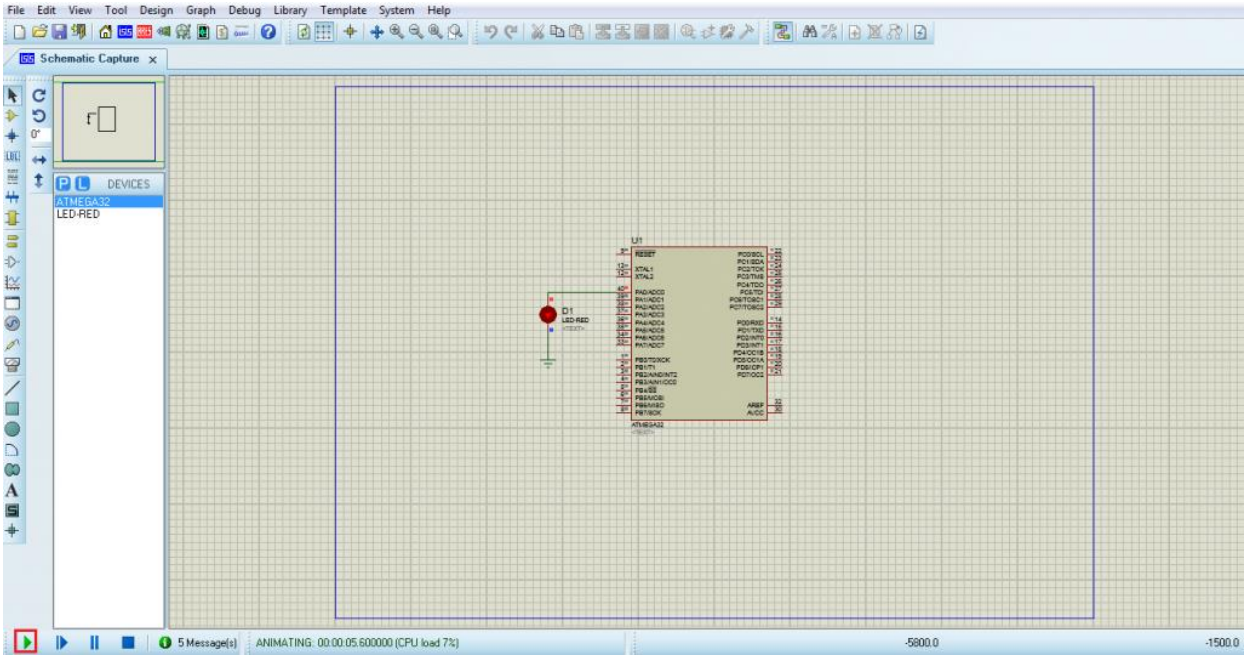


حال نوبت به شبیه سازی پروژه در پروتئوس می رسد.

برای شبیه سازی دوباره به نرم افزار پروتئوس باز می گردیم . در نرم افزار Proteus ، روی میکروکنترلر دابل کلیک کرده تا پنجره Edit Component باز شود. در این پنجره در قسمت Program File روی آیکون پوشه (Browse) کلیک کنید تا پنجره انتخاب فایل باز شود . حالا می بایست به مسیر برنامه ای که در کدویژن نوشتید بروید و در آنجا داخل پوشه Exe شده و فایل با پسوند Hex را انتخاب کنید .



بعد از انتخاب فایل hex با دیگر تنظیمات کاری نداشته و پنجره Edit Component را و Ok میکنیم . با این کار برنامه نوشته شده به زبان C در نرم افزار کدویژن به داخل آی سی میکروکنترلر در نرم افزار پروتئوس ریخته می شود . حال برای شبیه سازی مدار روی دکمه Play پایین صفحه کلیک کرده تا شبیه سازی آغاز و مدار Run شود . مشاهده می کنید که led هر ثانیه چهار بار روشن و خاموش می شود.

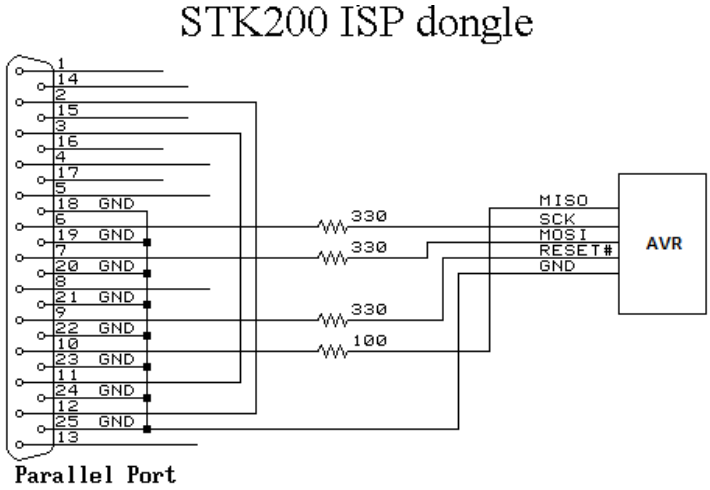


مرحله چهارم : پیاده سازی مدار

در این مرحله ابتدا میکروکنترلر را توسط پروگرامر و نرم افزار کدویژن ، برنامه ریزی (Program) کرده و سپس مدار را بر روی برد برد می بندیم.

پروگرام کردن میکرو توسط نرم افزار CodeVision

ساده ترین نوع پروگرامر میکروکنترلرهای AVRبه نام Stk200 می باشد که تنها به یک پورت LPT یا همان پورت پرینتر DB25برای ساخت نیاز دارد و خودتان هم میتوانید آن را درست کنید . برای ساخت و سپس اتصال به PC و میکرو بوسیله stk200 از شکل زیر می توان استفاده نمود . در غیر این صورت بهتر است یکی از پروگرامر های موجود در بازار را تهیه کنید.

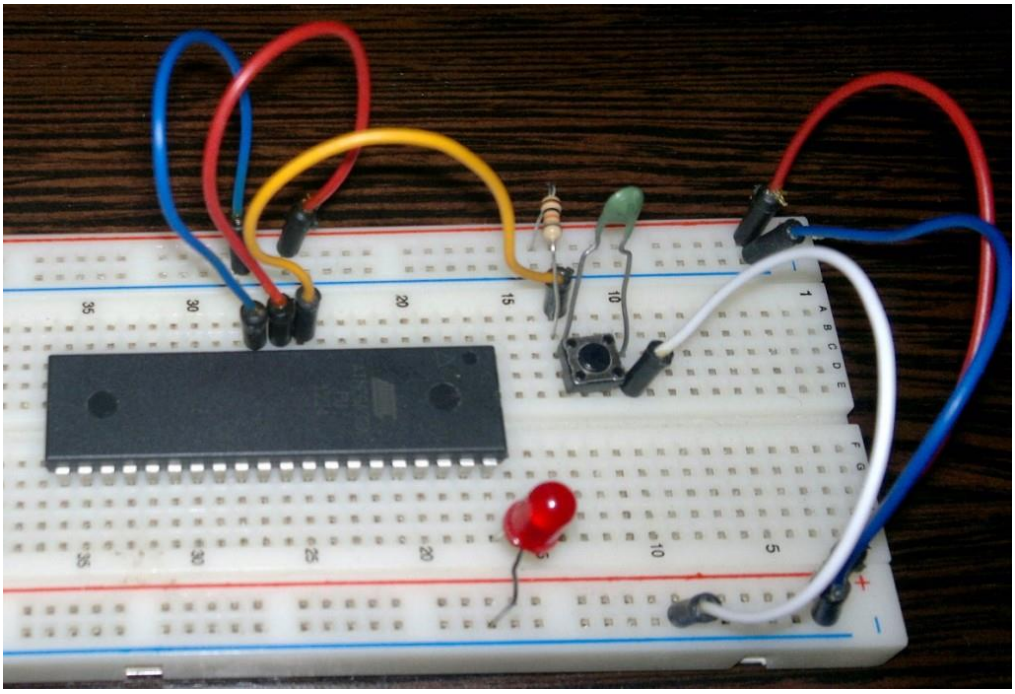


بعد از وصل کردن و روشن کردن پروگرامری که خریداری کرده اید به پورت مربوطه روی PC یا لپ تاپ و نصب فایل درایور مربوطه (در صورت نیاز) مراحل زیر برای پروگرام کردن را در نرم افزار CodeVision اجرا کنید :

1. ابتدا از منوی Setting ، گزینه Programmer را انتخاب می کنیم.
2. از لیست موجود می بایست نوع پروگرامر را همان مدل پروگرامری که خریداری کرده اید ، انتخاب کنید.
3. در قسمت Communication Port می بایست آن پورتی که پروگرامر به آن وصل است را تنظیم کرد . اگر پورتی که پروگرامر به آن وصل است را نمی دانید کافیست به قسمت Device Manager در Control Panelمراجعه کرده و سپس در قسمت Port&LPT نام پروگرامر و پورت Com آن مشخص است.
4. میکروکنترلر را روی پروگرامر در جای مناسب و در جهت صحیح قرار می دهیم.
5. از منوی Tools ، گزینه Chip Programmer را انتخاب می کنیم.
6. با زدن دکمه Program All ، پروگرام آغاز می شود.
7. در پیغامی که هنگام پروگرام ظاهر می شود از ما میخواهد تا فایلی برای ریختن دیتا درون EEPROM معرفی کنید ، چون ما فایلی نداریم پس گزینه Cancel را انتخاب می کنیم.

نکته مهم : دقت کنید که حتما تیک گزینه Program Fuse Bit زده نشده باشد تا فیوزبیت های میکروکنترلر در حالت default بدون تغییر بماند.

بعد از پروگرام کردن میکروکنترلر کافی است آن را وسط بردبرد قرار داده و led را بین پایه ۴۰ و زمین (در جهت صحیح) متصل نماییم و بقیه مدار را به صورت مدار حداقلی که در فصل قبل به آن اشاره کردیم ببندیم . بنابراین مداری مشابه شکل زیر را خواهیم داشت.



مرحله پنجم : تست و عیب یابی

در صورت رعایت دقیق همه مراحل گفته شده با وصل منبع تغذیه مناسب مدار بدون هیچ مشکلی کار خواهد کرد.

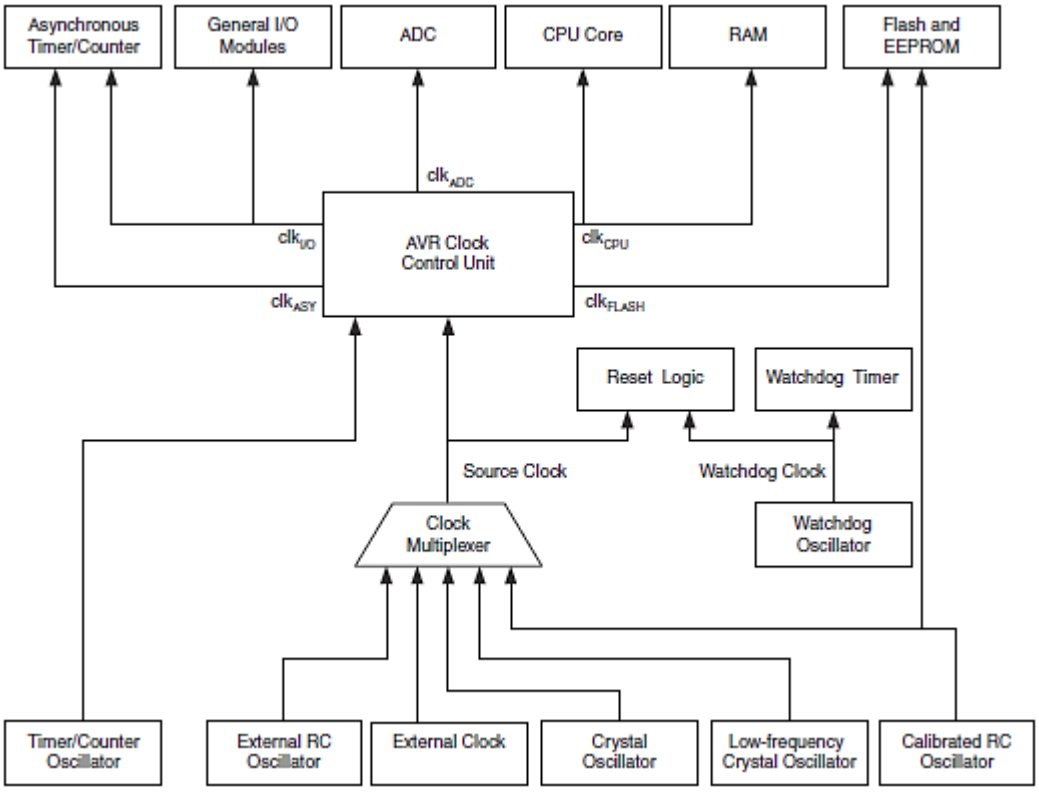
مرحله ششم : تولید ، ارتقا و بهبود

در این مثال ساده ، هدف یادگیری بود و برنامه نیازی به تولید ، ارتقا و بهبود ندارد.

پایان مثال عملی شماره ۱

واحد کنترل کلاک سیستم در میکروکنترلر Atmega32

همانطور که قبلا نیز گفتیم این واحد وظیفه تامین کلاک ورودی و پخش آن به تمامی واحد های میکروکنترلر را بر عهده دارد . شکل زیر بلوک دیاگرام این واحد را برای میکروکنترلر Atmega32 نشان می دهد. این واحد به کلی توسط فیوز بیت ها (Fuse Bits) تنظیم و کنترل می شود و در معماری AVR برای تنظیم و کنترل این واحد رجیستر خاصی در نظر گرفته نشده است.



همانطور که در شکل مشاهده می کنید منابع کلاک توسط یک مالتی پلکسر ، پالس لازم را به واحد کنترل کننده کلاک ارسال می کند. در واقع کلاک لازم جهت راه اندازی سیستم می تواند تنها یکی از منابع زیر باشد و امکان استفاده همزمان از آنها وجود ندارد.

- اسیلاتور RC (خازن – مقاومت) خارجی
- کلاک خارجی
- کریستال اسیلاتور خارجی
- کریستال اسیلاتور فرکانس پایین خارجی
- اسیلاتور RC داخلی

بعد از وارد شدن پالس کلاک به واحد کنترل کلاک ، تقسیمی از کلاک وارد شده به کلیه واحدهای میکرو کنترلر می رود . همانطور که در شکل نیز مشاهده می کنید ، پالس CLKCPU به هسته مرکزی (CPU) و SRAM اعمال می شود. پالس CLKADC ، کلاک لازم را جهت واحد مبدل آنالوگ به دیجیتال فراهم می کند . پالس CLKFLASHکلاک لازم را بر ای حافظه Flash و eepromداخلی فراهم می سازد. پالس CLK/IO برای تولید پالس ماژول های ورودی و خروجی نظیرUSART ، SPI، شمارنده ها و وقفه ها بکار برده می شود.پالس CLKASYبرای راه اندازی آسنکرون تایمر یا کانتر دو برای استفاده از فرکانس ۳۲,۷۶۸ MHZ اسیلاتور RTC است . در این شکل همچنین نحوه اتصال واحد تایمر Watchdogنیز نشان داده شده است . تایمر سگ نگهبان از یک اسیلاتور مجزای داخلی استفاده می کند که پالس کلاک مورد نیاز خود را تامین کرده و احتیاجی به منبع ورودی کلاک ندارد.

فیوز بیت ها در میکروکنترلرهای AVR

فیوز بیت ها قسمتی در حافظه **فلش** هستند که با تغییر آنها امکانات میکروکنترلر نظیر تنظیم منبع کلاک میکرو ، روشن و خاموش کردن امکان JTAG و ... را میتوان کنترل کرد . میکرو کنترلر های AVRبسته به نوع قابلیتی که دارند دارای فیوز بیت های متفاوتی هستند . در میکروکنترلرهای AVR حداکثر سه بایت برای این منظور در نظر گرفته می شود . توجه کنید که برنامه ریزی فیوزبیت ها باید قبل از قفل (Lock) تراشه صورت گیرد . برای اینکه بدانید میکروکنترلری که با آن کار می کنید دارای چه ویژگی و چه فیوز بیت هایی می باشد ، به قسمت System Clock & Clock Options در Datasheet آن میکرو مراجعه نمایید.

فیوز بیت ها در میکروکنترلر Atmega32

میکروکنترلر ATMEGA32 دارای **۲ بایت** (۱۶ بیت) فیوز بیت (Fuse Bit) می باشد که ۸ بیت کم ارزش آن Low Byteو ۸ بیت دیگر آن High Byte نامگذاری شده است . در شکل زیر نام ، عملکرد و پیش فرض این فیوز بیت ها را مشاهده می کنید.

شماره بیت	Low Byte	عملکرد	پیش فرض
۰	CKSEL0	انتخاب منبع کلاک	۱
۱	CKSEL1		۰
۲	CKSEL2		۰
۳	CKSEL3		۰
٤	SUT0	انتخاب زمان Startup	۰
٥	SUT1		۱
٦	BODEN	فعال ساز آشکار ساز Brown-out	۱
۷	BODLEVEL	تنظیم سطح ولتاژ Brown-out	۱

شماره بیت	High Byte	عملکرد	پیش فرض
۰	BOOTRST	انتخاب بردار Reset بخش Boot	۱
۱	BOOTSZ0	انتخاب اندازه ی Bootloader	۰
۲	BOOTSZ1		۰
۳	EESAVE	حفاظت از EEPROM در زمان Erase	۱
٤	CKOPT	انتخاب عملکرد کلاک	۱
٥	SPIEN	فعال ساز پروگرام شدن از طریق SPI	۰
٦	JTAGEN	فعال ساز پورت JTAG	۰
۷	OCDEN	فعال ساز اشکال زدایی از طریق JTAG	۱

نکته : در تمام جداول مربوط به فیوز بیت ها ، ۰ به معنای بیت برنامه ریزی شده (PROGRAMMED) و ۱ به معنای بیت برنامه ریزی نشده (UNPROGRAMMED) می باشد.

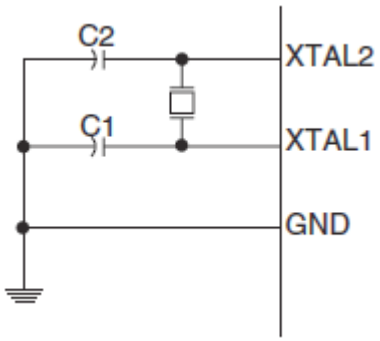
در ادامه به معرفی و بررسی هر یک از فیوز بیت های فوق می پردازیم:

- **فیوز بیت هایCKSEL0 ، CKSEL1 ، CKSEL2 ، CKSEL3 ، CKOPTوSUT0 ، SUT1**: فیوز بیت CKOPT میتواند برای دو حالت مختلف استفاده شود . یعنی زمانی که محیط پر نویز باشد و زمانی که از کریستال خارجی استفاده شود این بیت برنامه ریزی می شود (۰) در بقیه حالت نیازی به برنامه ریزی این بیت نیست (۱) .همچنین فیوز بیت های CKSEL0 تا CKSEL3 برای تنظیم منبع کلاک میکرو می باشد و بسته به منبع ورودی کلاک به یکی از صورت های جدول زیر باید برنامه ریزی شود. فیوز بیت های SUT0 و SUT1 نیز زمان راه اندازی (Start-up) را در هنگام متصل کردن منبع تغذیه تعیین می کنند . که این زمان در هر یک از حالت های جدول زیر متفاوت است.

Device Clocking Option	CKSEL3..0
External Crystal/Ceramic Resonator	1111 - 1010
External Low-frequency Crystal	1001
External RC Oscillator	1000 - 0101
Calibrated Internal RC Oscillator	0100 - 0001
External Clock	0000

نوسان ساز با کریستال خارجی

برای استفاده از کریستال خارجی،باید فیوز بیت های CKSEL3.0 را بین ۱۰۱۰ تا ۱۱۱۱ برنامه ریزی کنیم . پایه های خارجی XTAL1وXTAL2 طبق شکل زیر توسط دو خازن عدسی (خازن های بالانس) با مقادیر یکسان به یک کریستال با فرکانس حداکثر تا ۱۶ مگاهرتز متصل می گردند.



بنابراین در حالتی که از کریستال خارجی استفاده می شود ، مدار فوق می بایست به میکرو متصل گردد و تنظیمات فیوز بیت ها طبق جدول زیر بر اساس فرکانس کاری مورد نیاز میکرو انجام شود.

CKOPT	CKSEL3..1	Frequency Range (MHz)	Recommended Range for Capacitors C1 and C2 for Use with Crystals (pF)
1	101[1]	0.4 - 0.9	–
1	110	0.9 - 3.0	12 - 22
1	111	3.0 - 8.0	12 - 22
0	101, 110, 111	1.0 ≤	12 - 22

خازن های C1 و C2 را معمولا 22PF انتخاب می کنند . وظیفه این خازن حذف نویز الکترومغناطیس اطراف کریستال می باشد که طبق جدول فوق با توجه به کریستال استفاده شده تعیین می شوند. در PCB برای حذف نویز ، معمولا بدنه کریستال خارجی را به زمین وصل می کنند اما نباید بدنه کریستال حرارت بیند زیرا ممکن است به آن آسیب برسد.

هنگام استفاده از کریستال خارجی بهتر است با فعال کردن بیت CKOPT دامنه نوسان اسیلاتور را حداکثر کرد. در این حالت اسیلاتور بصورت Rail-to-Rail عمل می کند یعنی مثلا اگر تغذیه میکروکنترلر ۵ ولت باشد حداکثر دامنه پالس ساعت نیز ۵ ولت خواهد بود. این حالت برای محیط های پر نویز مانند کارخانه های صنعتی بسیار مناسب است . البته فعال کردن این فیوز بیت به اندازه چند میلی آمپر جریان مصرفی میکروکنترلر را افزایش می دهد.

نکته : در میکروکنترلرهای بدون پسوند L اگر بخواهیم از کریستال ۱۶ MHZ استفاده کنیم باید فیوز بیت CKOPT فعال گردد در غیر اینصورت حداکثر کریستال خارجی ممکن برای اتصال به میکرو ۸ MHZخواهد بود .

زمان Start-up برای استفاده از کریستال خارجی توسط فیوز بیت های SUT0 و SUT1 طبق جدول زیر تعیین می گردد.منظور از زمان Start-up مدت زمانی است که طول میکشد به محض وصل شدن تغذیه به میکروکنترلر و پایدار شدن نوسانات کریستال اسیلاتور خارجی ، میکروکنترلر Resetشده و برنامه را شروع به اجرا کند . این مدت زمان در حدود چند میلی ثانیه می باشد که توسط جدول زیر قابل تنظیم است.

CKSEL0	SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
0	00	258 CK(1)	4.1 ms	Ceramic resonator, fast rising power
0	01	258 CK(1)	65 ms	Ceramic resonator, slowly rising power
0	10	1K CK(2)	–	Ceramic resonator, BOD enabled
0	11	1K CK(2)	4.1 ms	Ceramic resonator, fast rising power
1	00	1K CK(2)	65 ms	Ceramic resonator, slowly rising power
1	01	16K CK	–	Crystal Oscillator, BOD enabled
1	10	16K CK	4.1 ms	Crystal Oscillator, fast rising power
1	11	16K CK	65 ms	Crystal Oscillator, slowly rising power

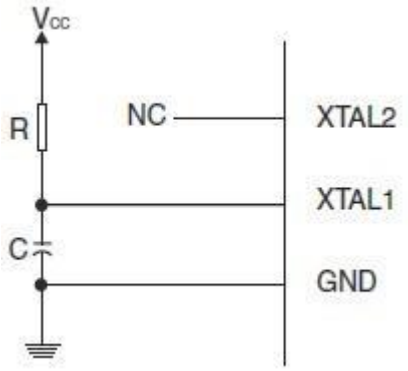
نوسان ساز با کریستال فرکانس پائین

منظور از کریستال فرکانس پائین ، استفاده از کریستال ساعت (۳۲۷۶۸ هرتز) می باشد . در صورتی که فیوز بیت های CKSEL 3..0 به صورت ۱۰۰۱ برنامه ریزی شوند پالس ساعت سیستم از کریستال خارجی فرکانس پایین استفاده می کند. در این حالت اگر فیوز بیت CKOPT فعال شود خازن داخلی بین دو پایه XTAL1 و XTAL2 فعال می گردد. با فعال شدن این خازن که مقدار آن ۳۶ pFاست ، احتیاجی به قرار دادن خازن های C1 و C2 نیست و میتوان مستقیما این کریستال را روی دو پایه XTAL1 و XTAL2 نصب کرد . همچنین زمان Start-upمورد نیاز میکرو در این حالت ، طبق جدول زیر تعیین می شود.

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	1K CK(1)	4.1 ms	Fast rising power or BOD enabled
01	1K CK(1)	65 ms	Slowly rising power
10	32K CK	65 ms	Stable frequency at start-up
11	Reserved		

نوسان ساز با RC خارجی

از اسیلاتور RC خارجی در کاربردهایی که به تغییرت زمان و فرکانس حساسیت نداشته باشیم استفاده می کنیم . در این حالت مدار زیر باید به میکرو متصل شود که در آن فرکانس نوسانات اسیلاتور از رابطه (F = 1 /3RC) بدست می آید و مقدار خازن C حداقل باید ۲۲ pFباشد.



بنابراین در حالتی که از کریستال خارجی استفاده می شود ، مدار فوق می بایست به میکرو متصل گردد و تنظیمات فیوز بیت ها طبق جدول زیر بر اساس فرکانس کاری مورد نیاز میکرو انجام شود.

CKSEL3..0	Frequency Range (MHz)
0101	0.1 - 0.9
0110	0.9 - 3.0
0111	3.0 - 8.0
1000	8.0 - 12.0

در این حالت نیز اگر فیوز بیت CKOPT فعال شود خازن داخلی بین دو پایه XTAL1 و XTAL2 فعال می گردد. با فعال شدن این خازن که مقدار آن 36pF است ، احتیاجی به قرار دادن خازن های C1 و C2 نیست و میتوان مستقیما این کریستال را روی دو پایه XTAL1 و XTAL2 نصب کرد . همچنین زمان Start-up مورد نیاز میکرو در این حالت ، طبق جدول زیر تعیین می شود.

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	18 CK	–	BOD enabled
01	18 CK	4.1 ms	Fast rising power
10	18 CK	65 ms	Slowly rising power
11	6 CK(1)	4.1 ms	Fast rising power or BOD enabled

نوسان ساز با اسیلاتور RC کالیبره شده داخلی

اسیلاتور RC کالیبره شده داخلی می تواند فرکانس های ثابت ۱ MHZ، ۲ MHZ، ۴ MHZو ۸ MHZرا در شرایط تغذیه +۵ Vو در دمای ۲۵ cایجاد نماید. فرکانس کاری این اسیلاتور به شدت به ولتاژ تغذیه، درجه حرارت محیط و مقدار بایت رجیستر OSSCAL وابسته می باشد.

از آنجایی که این نوع نوسان ساز به دما و ولتاژ وابسته است پیشنهاد می کنیم در موقع استفاده از تبادل سریال USARTو دیگر پروتکل ها و برنامه هایی که به زمان بسیار وابسته هستند از کریستال خارجی استفاده کنید . همچنین فیوزبیت های میکروکنترلر ATmega32 به طور پیش فرض توسط کارخانه طوری تنظیم شده است که از اسیلاتور کالیبره شده داخلی با فرکانس ۱ MHZاستفاده می کند.

زمانی که اسیلاتور داخلی استفاده می شود نیازی به قرار دادن اسیلاتور خارجی نیست و پایه های XTAL1 و XTAL2آزاد گذاشته می شود و همچنین در این نوسان ساز، نباید فیوز بیت های CKOPT فعال باشد. مقدار فرکانس این اسیلاتور توسط فیوز بیت های CKSEL3.0 طبق جدول زیر تعیین می شود.

CKSEL3..0	Nominal Frequency (MHz)
0001(1)	1.0
0010	2.0
0011	4.0
0100	8.0

زمان Start-up میکرو نیز توسط فیوز بیت های SUT0 و SUT1 طبق جدول زیر تنظیم می گردد.

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	6 CK	–	BOD enabled
01	6 CK	4.1 ms	Fast rising power
10(1)	6 CK	65 ms	Slowly rising power
11	Reserved		

رجیستر کالیبراسیون OSCCAL

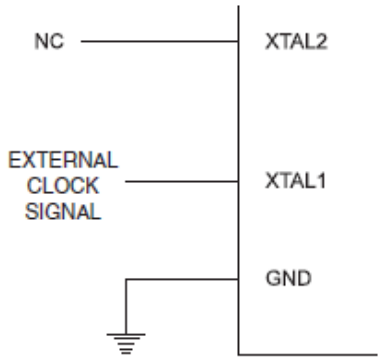
Bit	7	6	5	4	3	2	1	0	
	CAL7	CAL6	CAL5	CAL4	CAL3	CAL2	CAL1	CAL0	OSCCAL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

در صورت استفاده از اسیلاتور RC کالیبره شده داخلی در هر بار که میکروکنترلر Reset می شود ، مقدار رجیستر OSCCALخوانده شده و اسیلاتور به طور خودکار تنظیم می گردد. در حالت پیش فرض مقدار این رجیستر طوری تنظیم شده است که اسیلاتور RC روی ۱ مگاهرتز کار کند . کاربرد این رجیستر برای کاهش خطای اسیلاتور RC داخلی در شرایط مختلف است اما اگر از شرایط تغذیه ۵+ vو یا دمای ۲۵ cاستفاده گردد این خطا به ٪۱۰ کاهش می یابد . با نوشتن مقدار ۰ x00کمترین و با نوشتن مقدار ۰ xFFدر این رجیستر بیشترین فرکانس ممکن برای کالیبراسیون انتخاب می گردد . تنظیم دقیق این کالیبراسیون در فرکانس های به جز ۱ ، ۲ ، ۴ و ۸ مگاهرتز خیلی تضمینی نیست . شکل زیر درصد تغییرات این رجیستر را در حالت تنظیم آن روی مقادیر مختلف نشان می دهد.

OSCCAL Value	Min Frequency in Percentage of Nominal Frequency (%)	Max Frequency in Percentage of Nominal Frequency (%)
$00	50	100
$7F	75	150
$FF	100	200

نوسان ساز با کلاک خارجی

در صورت تنظیم فیوز بیت های CKSEL3..0 به صورت "۰۰۰۰" میکروکنترلر AVR پالس کلاک ورودی خود را از یک منبع خارجی دریافت می کند . در این حالت پایه XTAL2 بدون اتصال بوده و پایه XTAL1 به منبع تولید پالس دیجیتال متصل می شود . با فعال نمودن فیوز بیت CKOPT نیز میتوان خازن ۳۶ پیکوفارادی بین پایه XTAL1با زمین را فعال کرد . نحوه اتصال کلاک خارجی به میکرو را در شکل زیر مشاهده می کنید.



در این حالت نیز فیوز بیت های SUT1 و SUT2 طبق جدول زیر می باشند.

SUT1..0	Start-up Time from Power-down and Power-save	Additional Delay from Reset (V_{CC} = 5.0V)	Recommended Usage
00	6 CK	–	BOD enabled
01	6 CK	4.1 ms	Fast rising power
10	6 CK	65 ms	Slowly rising power
11	Reserved		

- **فیوز بیت EESAVE** : مقدار پیش فرض آن ۱ بوده و برای اینکه آیا در هنگام پاک کردن (Erase) میکرو حافظه EEPROM پاک شود یا نه ، از این فیوز بیت استفاده می شود . اما زمانی که مقدار این بیت را صفر کنیم ، محتویات EEPROM در هنگام پاک کردن (Erase) میکرو حفظ می شود . این بیت بطور پیش فرض غیرفعال است.
- **فیوز بیت JTAGEN** : این فیوز بیت که به صورت پیش فرض در حالت برنامه ریزی شده می باشد ، برای فعال کردن ارتباط JTAG می باشد . برنامه ریزی شدن این فیوز بیت در حالت پیش فرض باعث شده است

تا پورت C برای ارتباط I/O واحد ورودی/خروجی در دسترس نباشد . برای آزاد کردن پورت C می بایست این فیوز بیت را از حالت برنامه ریزی خارج نمود.

- **فیوز بیت OCDEN (On Chip Debug Enable)** : زمانیکه فیوز بیت ارتباط دهی JTAG فعال شده باشد و همچنین برنامه میکروکنترلر را قفل نکرده باشیم می توان با فعال کردن فیوزبیت OCDEN برنامه میکروکنترلر را به طور آنلاین در حین اجرا توسط مدار واسطی که از ارتباط سریال JTAG استفاده می کند توسط نرم افزار مشاهده کرد. به این نوع آنالیز امولاتور (Emulator)یا شبیه ساز سخت افزاری گفته می شود. همچنین برنامه ریزی شدن این بیت به قسمتهایی از میکرو امکان می دهد که در مدهای SLEEPکار کنند .درهر صورت فعال کردن این بیت مصرف توان میکرو کنترلر را افزایش می دهد. این بیت بصورت پیش فرض برنامه ریزی نشده (۱) است.

- **فیوز بیت SPIEN** : با فعال کردن این فیوز بیت می توان میکرو را از طریق ارتباط دهی سریال SPI برنامه ریزی کرد. این فیوز بیت بطور پیش فرض فعال است.
- **فیوز بیت های BOOTSZ0 و BOOTSZ1** : این دو بیت، مقدار حافظه اختصاص داده شده BOOT را طبق جدول زیر تعیین می کنند. در زمان برنامه ریزی شدن فیوز بیت BOOTRST اجرای برنامه از آدرس حافظه BOOT آغاز خواهد شد.

BOOTSZ1	BOOTSZ0	اندازه ی Boot	Pages	آدرس بردار Reset
۱	۱	Word ۱۲۸	۲	$1F80
۱	۰	Word ۲۵٦	٤	$F00
۰	۱	Word ۵۱۲	۸	$E00
۰	۰	Word ۱۰۲٤	۱٦	$C00

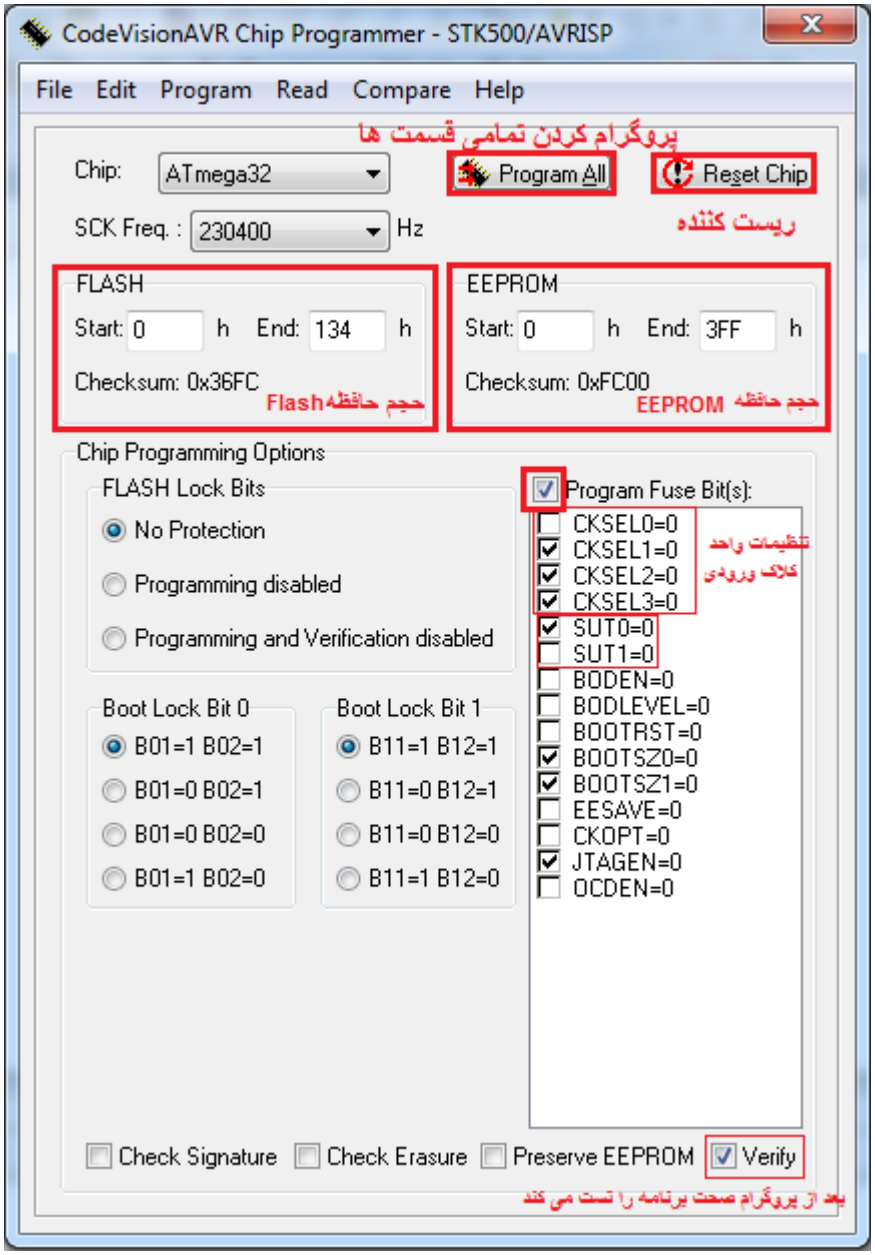
- **فیوزبیت BOOTRST** : این بیت برای انتخاب بردار Reset است اگر غیر فعال باشد آدرس بردار RESETاز $۰۰۰۰ است و اگر این بیت فعال شود به آدرسی که فیوز بیت های BOOTSZ0 و BOOTSZ1مشخص کرده اند تغییر می یابد.

- **فیوز بیت BODEN** : مدار BROWN-OUTآشکارساز ولتاژ تغذیه است که اگر از ۲,۷ یا ۴ ولت کمتر شود میکروکنترلر را ریست می کند. برای فعال کردن این مدار، باید فیوزبیت BODEN فعال گردد. این فیوز بیت بطور پیش فرض غیرفعال است.
- **فیوز بیت BODLEVEL** : اگر فیوزبیت BODEN فعال و فیوز بیت BODLEVEL غیرفعال باشد با کاهش ولتاژ VCC کمتر از ۲,۷ ولت میکروکنترلر ریست می شود اما اگر فیوز بیت BODLEVEL را فعال کنیم آنگاه با کاهش ولتاژ VCC کمتر از ۴ ولت میکروکنترلر ریست می شود .مطابق جدول زیر سطح ولتاژ BROWN-OUTتعیین می شود. فیوز بیت BODLEVEL بصورت پیش فرض غیر فعال است.

BODEN	BODLEVEL	سطح ولتاژ Brown-out
۱	۱	غیر فعال
۱	۰	غیر فعال
۰	۱	Vcc=2.7v
۰	۰	Vcc=4.0v

تنظیم فیوز بیت ها در نرم افزار کدویژن

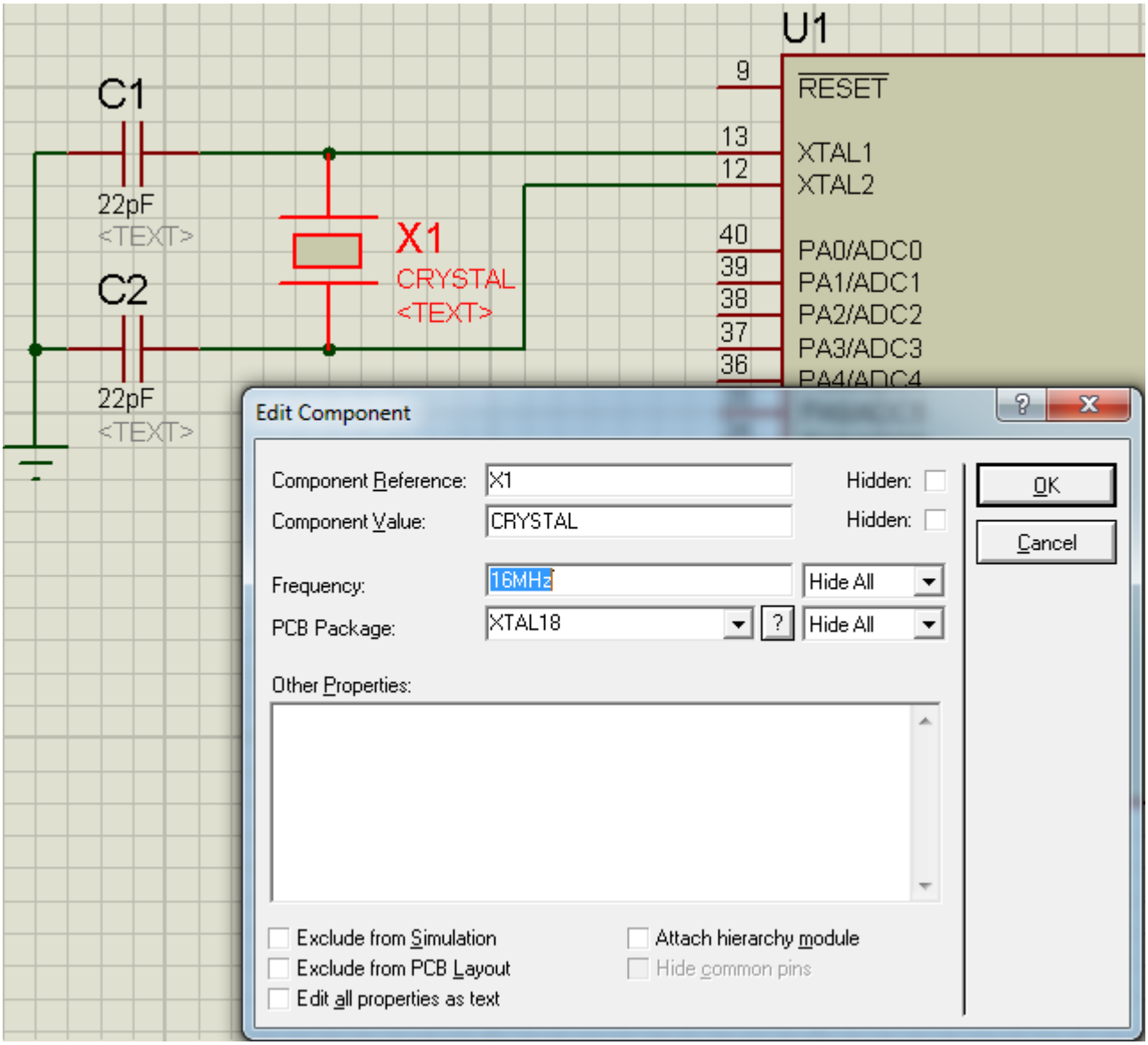
بار دیگر به نرم افزار CodeVision بر می گردیم . در شکل زیر محیط Chip Programmer و کاربرد قسمت های مختلف آن را برای تنظیمات Fuse Bits و Lock Bits میکروکنترلر Atmega32 مشاهده می کنید . با تنظیم کردن این پنجره و سپس پروگرام کردن میکروکنترلر تغییرات مورد نظر در آن اعمال می شود . همچنین در شکل زیر تنظیمات فیوز بیت ها را در حالت defaultمیکروکنترلر atmega32 در حالت ۱ Mhzداخلی (با پروگرامر Stk500) ، مشاهده می کنید . با تغییر این تیک ها میتوان فیوزبیت ها را برنامه ریزی نمود .



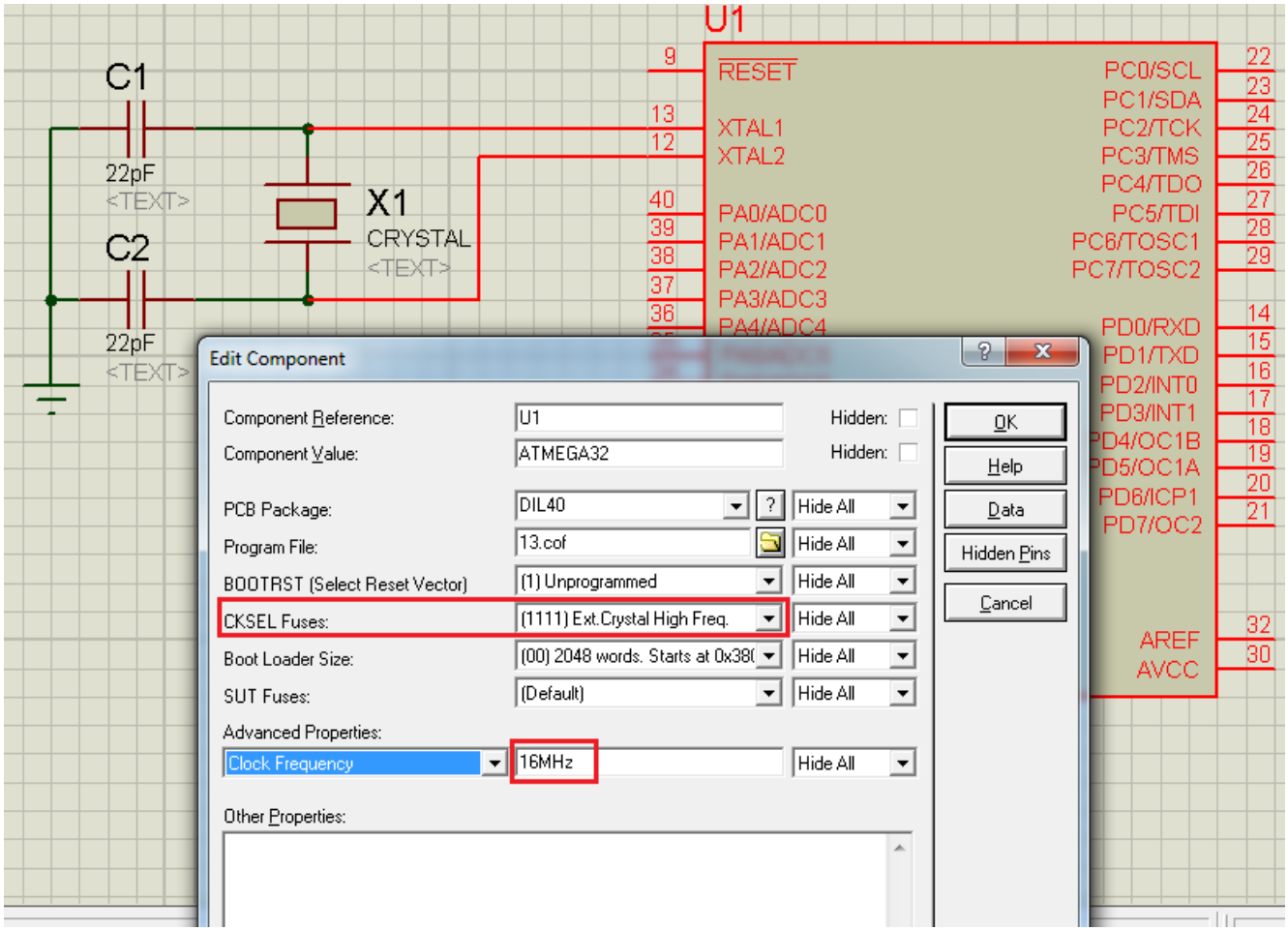
توجه : تنها یکبار فیوزبیت ها را به طور صحیح برنامه ریزی کرده و سپس تیک گزینه Program Fuse Bits را بردارید تا احیانا و اشتباها آنها را تغییر ندهید . اگر احیانا تغییر کرده باشد می توانید مجددا به صورت شکل فوق به تنظیمات پیش فرض (**1MHz داخلی**) درآورید.

تنظیم پروتئوس در حالت استفاده از کریستال خارجی

فرض کنید فیوز بیت ها را طوری تنظیم کرده ایم که در حالت کریستال خارجی کار کند . در این صورت برای استفاده از کریستال خارجی در شبیه سازی با تایپ کردن عبارت crystal ، آن را از کتابخانه انتخاب و به میکرو وصل می کنیم . همچنین دو عدد خازن ۲۲ یا ۲۷ پیکوفاراد را نیز به صورت شکل زیر متصل می نماییم . بعد از اتصال مدار روی کریستال دابل کلیک کرده و فرکانس کار آن را انتخاب می کنیم . در اینجا ما فرکانس کریستال را ۱۶ مگاهرتز فرض کردیم.



سپس برای تنظیمات راه اندازی میکرو بوسیله کریستال خارجی ابتدا روی میکرو دابل کلیک کرده تا پنجره تنظیمات میکرو باز شود سپس می بایست بر اساس فرکانس کاری کریستال خارجی ، قسمت فیوز بیت ها را تنظیم کرده و سپس در قسمت بعد فرکانس کریستال متصل شده به میکرو را به صورت شکل زیر وارد نمایید.



حال با انجام مراحل فوق میکروکنترلر روی کریستال ۱۶ مگاهرتز تنظیم شده و با play کردن مدار در پروتئوس شبیه سازی آغاز می شود.

پایان فصل ششم

این فصل پایه کار بخش های بعدی است لذا دست به کار شوید و تمام مراحل این فصل را حداقل یکبار خودتان انجام دهید و قبل از تسلط بر این فصل ، به فصل بعدی نروید.

فصل هفتم : آموزش برنامه نویسی به زبان C و انجام پروژه با میکروکنترلرAtmega32

مقدمه

در فصل های گذشته به این نکته اشاره کردیم که قدرتمندترین زبان برنامه نویسی میکروکنترلرها زبان c و c++ می باشد . همچنین اشاره کردیم که برنامه نویسی برای یک ماشین بر مبنای پردازنده های RISC با برنامه نویسی برای یک ماشین بر مبنای پردازنده های CISC تفاوت اساسی دارد و آن هم حساسیت بیشتر RISC نسبت به CISC می باشد که برنامه نویس را مجبور می کند تا با دقت بیشتر و درک بیشتر سخت افزار برنامه نویسی کند . با این دید از میکروکنترلر ها در این فصل **گذری** بر برنامه نویسی به زبان C ویژه میکروکنترلر ها خواهیم داشت و نکات و مفاهیمی که یک برنامه نویس میکرو باید آنها را رعایت کند به همراه انجام مثال ها و پروژه های عملی مربوطه شبیه سازی خواهیم کرد.

معرفی کوتاه زبانC

زبان برنامه‌نویسی سی ، زبانی همه منظوره، ساخت یافته و روندگرا می‌باشد که در سال ۱۹۷۲ توسط دنیس ریچی در آزمایشگاه‌های بل ساخته شد . به طور کلی زبان های برنامه نویسی را می توان در سه سطح دسته بندی کرد : ،زبان های سطح بالا ،زبان های سطح میانی ،زبان های سطح پایین . زبان سی یک زبان سطح میانی است که در آن هم می توان به سطح بیت و بایت و آدرس دسترسی داشت و هم از مفاهیم سطح بالا که به زبان محاوره ای انسان نزدیکتر است (مانند حلقه های شرطی if ... else و حلقه های تکرار for و while و ...) ، بهره گرفت . در زبان سی هیچ محدودیتی برای برنامه نویس وجود ندارد و هر آنچه را که فکر می کنید ، می توانید پیاده سازی کنید . ارتباط

تنگاتنگی بین زبان c و اسمبلی وجود دارد ، به این صورت که می توان از برنامه نویسی اسمبلی در هر کجای برنامه زبان سی استفاده کرد.

کلمات کلیدی در زبانC

زبان سی زبان کوچکی است چرا که در آن تعداد کلمات کلیدی ۳۲ کلمه است . کم بودن کلمات کلیدی در یک زبان مبنی بر ضعف آن نیست . زبان بیسیک حاوی ۱۵۰ کلمه کلیدی است اما قدرت زبان سی به مراتب بالاتر است . زبان سی به حروف کوچک و بزرگ حساس است . (Case Sensitive) در این زبان بین حروف کوچک و بزرگ تفاوت است و تمام کلمات کلیدی باید با حروف کوچک نوشته شوند . در شکل زیر تمام کلمات کلیدی زبان سی را مشاهده می کنید.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

ویژگی های یک برنامه به زبانC

-هر دستور در زبان سی با ; به پایان می رسد.

-حداکثر طول هر دستور ۲۵۵ کاراکتر است.

-هر دستور می تواند در یک یا چند سطر نوشته شود.

-برای توضیحات تک خطی از // در ابتدای خط استفاده می شود و یا توضیحات چند خطی در بین */ و /* قرار می گیرد.

ساختار یک برنامه به زبان C در کامپیوتر

هر زبان برنامه نویسی دارای یک ساختار کلی است . این ساختار یک قالب را برای برنامه نویس فراهم می کند . ساختار کلی یک برنامه به زبان C را در زیر مشاهده می کنید.

```
#include < HeaderFiles.h >

محل معرفی متغیرهای عمومی ، ثوابت و توابع

void main (void)
{
محل مربوط به کدهای برنامه
}
```

بنابراین همانطور که مشاهده می شود:

1. خطوط ابتدایی برنامه ، دستور فراخوانی فایل های سرآمد (Header Files) می باشد . فایل های سرآمد فایل هایی با پسوند h هستند که حاوی پیش تعریف ها و الگو های توابع می باشند.
2. قالب اصلی برنامه ب مبنای تابعی به نام main بنا شده است . تابعی که اصولا ورودی و خروجی ندارد و کدهای اصلی برنامه را در خود دارد.

تفاوت برنامه نویسی برای کامپیوتر و میکروکنترلر

ساختار فوق یک قالب کلی در برنامه نویسی زبان C در کامپیوتر (ماشین CISC) با هدف اجرا در کامپیوتر را نشان می دهد .برنامه نویسی میکروکنترلر ها (ماشین RISC) با هدف اجرا در میکروکنترلر ، با برنامه نویسی در کامپیوتر کمی متفاوت است . تفاوت اصلی در کامپیوتر این است که عامل اجرای برنامه ، در زمان نیاز به عملکرد آن ، یک کاربر است . هر زمان که کاربر نیاز به عملکرد برنامه داشته باشد آن را اجرا می کند و نتیجه را بررسی می کند . اما در میکروکنترلر رفتار یک آی سی است که عامل اجرای برنامه منبع تغذیه است . با وصل منبع تغذیه برنامه شروع به کار می کند و تا زمانی که منبع تغذیه وصل است باید کارهای مورد نیاز را دائما اجرا نماید .

ساختار برنامه میکروکنترلر به زبان C

برنامه ای که کاربر می نویسد باید طوری نوشته شود که وقتی روی آی سی پروگرام شد دائما اجرا شود. راه حل این مسئله قرار دادن کدهای برنامه درون یک حلقه نامتناهی است . این عمل باعث می شود تا میکروکنترلر هیچگاه متوقف نشود و بطور مداوم عملکرد طراحی شده توسط کاربر را اجرا کند. بنابراین ساختار یک برنامه به صورت زیر در می آید.

```
#include < HeaderFiles.h >

محل معرفی متغیرهای عمومی ، ثوابت و توابع

void main (void)
```

{

کدهایی که در این محل قرار میگیرند فقط یکبار اجرا می شوند

معمولا مقدار دهی اولیه به رجیستر ها در این ناحیه انجام می شود

while(1)

{

کدهایی که باید مدام در میکروکنترلر اجرا شوند

}

}

متغیرها در زبان C

یک متغیر محدوده ای از فضای حافظه است که با یک نام مشخص می شود. یک متغیر بسته به نوع آن می تواند حامل یک مقدار عددی باشد . یک متغیر می تواند در محاسبات شرکت کند و یا نتیجه محاسبات را در خود حفظ کند. در کل میتوان گفت که نتایج بخش های مختلف یک برنامه ، در متغیر ها ذخیره می شود. در جدول زیر انواع متغیرها ، فضایی که در حافظه اشغال می کنند و بازه مقدار پذیری آنها را در **کامپایلر کد ویژن** مشاهده می کنید.

Type	Size (Bits)	Range
bit	1	0 , 1
bool, _Bool	8	0 , 1
char	8	-128 to 127
unsigned char	8	0 to 255
signed char	8	-128 to 127
int	16	-32768 to 32767
short int	16	-32768 to 32767
unsigned int	16	0 to 65535
signed int	16	-32768 to 32767
long int	32	-2147483648 to 2147483647
unsigned long int	32	0 to 4294967295
signed long int	32	-2147483648 to 2147483647
float	32	±1.175e-38 to ±3.402e38
double	32	±1.175e-38 to ±3.402e38

نحوه تعریف متغیرها

متغیر ها به صورت زیر تعریف می شوند:

; مقدار اولیه = نام متغیر نوع متغیر

مثال:

Unsigned char A=12;

int a,X,j;

توضیح : در خط اول یک متغیر ۸ بیتی بدون علامت با نام A که تنها میتواند مقادیر ۰ تا ۲۵۵ بگیرد ، با مقدار اولیه ۱۲ تعریف شده است . در خط دوم نیز ۳ متغیر علامت دار با نام های a و X و j که هر سه مقدار اولیه ۰ دارند تعریف شده است.

نکته : در صورت عدم تعریف مقدار اولیه در هنگام تعریف یک متغیر ، مقدار اولیه در حالت default برابر ۰ تعریف می شود.

ویژگی های نام متغیر

-اولین کاراکتر نام متغیر عدد نمیتواند باشد.

-نام متغیر بیشتر از ۳۱ کاراکتر مورد استفاده نیست.

-نام متغیر تنها ترکیبی از حروف a تا z و A تا Z و اعداد و کاراکتر _ می تواند باشد.

انواع متغیر ها از نظر محل تعریف در برنامه

متغیرها از نظر مکانی که در برنامه تعریف می شوند ، به دو دسته کلی تقسیم می شوند:

1. متغیرهای عمومی(Global)
2. متغیرهای محلی(Local)

متغیر هایی که قبل از تابع main تعریف می شوند را متغیرهای عمومی گویند و در همه جای برنامه می توان به آن دسترسی داشت . اما متغیرهای محلی در بدنه توابع تعریف می شوند و در بیرون از آن تابع ، دسترسی به آن ممکن نیست. در واقع با تعریف یک متغیر عمومی در ابتدای برنامه ، مقدار مشخصی از حافظه برای همیشه به آن متغیر تخصیص می یابد اما متغیرهای محلی تنها در زمان احتیاج تعریف شده و در حافظه می نشینند و بعد از مدتی از حافظه پاک می شوند.

محل تعریف متغیرها در حافظه میکروکنترلر

زمانی که یک متغیر به صورتی که در بالا گفته شد ، تعریف می شود آن متغیر در اولین مکان خالی در حافظه SRAMذخیره می شود . برای تعریف متغیر در حافظه EEPROM از کلمه کلیدی eeprom و برای تعریف متغیر در حافظه FLASH از کلمه کلیدی flash قبل از تعریف متغیر استفاده می شود . بنابراین باید توجه داشت که با قطع منبع تغذیه کلیه حافظه SRAMپاک خواهد شد و متغیرهایی که باید ذخیره دائمی شوند می بایست در حافظه EEPROMیا FLASH تعریف و ذخیره شوند . مثال:

```
int a;

eeprom char b;

flash float c;
```

نکته : همانطور که قبلا گفتیم ، حافظه Flash ، حافظه برنامه کاربر است یعنی برنامه به زبان C بعد از کامپایل و ساخته شدن توسط کدویژن و پروگرام شدن روی میکروکنترلر در حافظه Flash ذخیره می شود . بنابراین برای تعریف متغیر در حافظه Flash تنها در صورت خالی بودن قسمتی از آن امکان پذیر است.

نکته : حافظه SRAM تنها با قطع تغذیه پاک می شود و میتوان کاری کرد که با ریست شدن میکرو متغیرهای عمومی موجود در SRAM ریست نشده و مقدار قبلی خود را حفظ نمایند.

توابع در زبانc

تابع یکی از مهمترین بخش های زبان سی می باشد . یک تابع همانند دستگاهی است که مواد اولیه را دریافت می کند و بعد از انجام عملیات مورد نظر روی آنها خروجی مطلوب را تحویل می دهد . توابع در زبان C یا توابع کتابخانه ای هستند یا توابعی هستند که کاربر بر حسب نیاز برنامه خود اضافه می کند .

زبان سی دارای توابعی است که از قبل نوشته شده‌اند، و توابع کتابخانه‌ای نامیده می‌شوند. در واقع فرایندهایی که پر کاربرد هستند و در اغلب برنامه‌ها مورد استفاده قرار می‌گیرند به صورت توابع مستقل قبلاً نوشته شده‌اند و درون فایلهایی قرار داده شده اند . با اضافه کردن فایل های سرآمد که تعریف آن توابع در آنها قرار دارد می توان از آن توابع استفاده کرد.

تابع اصلی برنامه نویسی به زبان C تابع mainنام دارد که در تمامی برنامه ها وجود داشته و بدون ورودی و خروجی است . توابع دیگر را می توان در بالای تابع main ، در پایین تابع main و یا در فایل های کتابخانه ای تعریف کرد.

انواع توابع در زبان c

هر تابع مجموعه ای از دستورات است که بر روی داده ها پردازش انجام می دهد. ورودی و خروجی یک تابع مقادیری هستند که تابع در برنامه دریافت می کند و به برنامه باز می گرداند. توابع بر اساس ورودی و خروجی به ۴ دسته زیر تقسیم می شود:

1. **تابع با ورودی ، با خروجی**

مثال : تابع با دو ورودی از جنس کاراکتر و یک خروجی از جنس کاراکتر

```
Char F1( char x , char y );
```

2. **تابع با ورودی ، بدون خروجی**

مثال : تابع دارای یک ورودی int و بدون خروجی

```
void F2( int x );
```

3. تابع بدون ورودی ، با خروجی

مثال : تابع بدون ورودی اما دارای خروجیint

```
int F3(void);
```

4. تابع بدون ورودی ، بدون خروجی

مثال : تابع بدون ورودی و خروجی

```
void F4(void);
```

بنابراین در اولین قدم باید مشخص کنیم که این تابع چه خروجی را به ما می دهد (در اصطلاح برنامه نویسی بر می گرداند) و فقط به ذکر نوع خروجی بسنده می کنیم ، یعنی مثلا اگر عدد صحیح برگرداند از int ، اگر کاراکتر برگرداند از char و به همین ترتیب برای انواع دیگر و اگر هیچ مقداری را برنگرداند از void استفاده می کنیم.

در قدم بعدی نام تابع را مشخص می کنیم . یک تابع باید دارای یک نام باشد تا در طول برنامه مورد استفاده قرار گیرد .هر نامی را می توان برای توابع انتخاب نمود که از قانون نامگذاری متغیرها تبعیت می کند، اما سعی کنید که از نامهایی مرتبط با عمل تابع استفاده نمایید.

همینطور باید ورودیهای تابع را نیز مشخص کنیم که در اصطلاح برنامه نویسی به این ورودیها، پارامترها یا آرگومان تابع نیز گفته می شود. اگر تابع بیش از یک پارامتر داشته باشد باید آنها را با استفاده از کاما از یکدیگر جدا نماییم و اگر تابع پارامتری نداشت از کلمه voidاستفاده می کنیم. نام پارامتر نیز میتواند هر نام دلخواهی باشد . بخاطر داشته باشید که قبل از نام هر پارامتر باید نوع آنرا مشخص نماییم و بدانیم که کامپایلر هیچ متغیری بدون نوع را قبول نکرده و در صورت برخورد با این مورد از برنامه خطا می گیرد و در نتیجه برنامه را اجرا نخواهد کرد .

بنابراین در زبان c توابع حداکثر دارای یک خروجی می باشند و اگر تابعی خروجی داشته باشد آن را باید با دستور returnبه خروجی تابع و برنامه اصلی بر گردانیم.

تعریف توابع در زبانc

برای نوشتن و اضافه کردن یک تابع باید دقت کرد که هر تابع سه بخش دارد : اعلان تابع ، بدنه تابع و فراخوانی تابع

نحوه تعریف تابع به یکی از سه صورت زیر است:

1. اعلان تابع قبل از تابع main باشد و بدنه تابع بعد از تابع main باشد.
2. اعلان تابع و بدنه تابع هر دو قبل از تابع main باشد.
3. اعلان تابع و بدنه تابع درون فایل کتابخانه ای باشد.

فراخوانی تابع نیز در درون تابع main صورت می گیرد . در صورتی که اعلان و فراخوانی تابع درون فایل کتابخانه ای باشد فقط نیاز به فراخوانی آن در main است.

نکته : هیچ تابعی را نمیتوان درون تابعی دیگر تعریف نمود و فقط به صورت های گفته شده صحیح است اما میتوان از فراخوانی توابع در داخل یکدیگر استفاده کرد بدین صورت که تابعی که داخل تابع دیگر فراخوانی می شود باید در برنامه زودتر تعریف شود.

اعلان و بدنه تابع:

;(, ... نام ورودی دوم نوع ورودی دوم , نام ورودی اول نوع ورودی اول) نام تابع نوع داده خروجی تابع

مثال:

```
void sample ( int x, int y );
```

در صورتی که بخواهیم اعلان تابع قبل از main باشد آن را به صورت فوق اعلان می کنیم و بعد از تابع main به صورت زیر دوباره اعلان را به همراه بدنه تابع می نویسیم .

{ (, ... نام ورودی دوم نوع ورودی دوم , نام ورودی اول نوع ورودی اول) نام تابع نوع داده خروجی تابع

{ بدنه تابع : دستوراتی که تابع انجام می دهد

مثال:

```
void sample ( int x, int y )
{
   .
   .
   .
}
```

در صورتی که بخواهیم قبل از تابع main اعلان و بدنه تابع باشد ، به همان صورت فوق این کار را انجام می دهیم با این تفاوت که یکجا هم اعلان و هم بدنه قبل از main تعریف می شود.

فراخوانی تابع:

صدا زدن تابع درون برنامه را فراخوانی گویند . در فراخوانی توابع باید نام تابع و مقدار ورودی ها را بیان کنیم که به آن آرگومانهای تابع نیز گفته می شود . در مقدار دهی آرگومان تابع در هنگام فراخوانی دیگر نباید نوع آرگومانها را ذکر کنیم. مثال:

```
void main(void)
{
...
   sample (a, b);
...
}
```

در مورد نوع خروجی تابع دو حالت وجود دارد. اول اینکه اگر تابع بدون خروجی باشد لازم نیست از void استفاده کنیم و دوم اینکه اگر تابع دارای خروجی باشد باید آنرا برابر با مقدار متغیر از همان نوع قرار دهیم تا مقدار برگشتی را در متغیر مذکور ریخته و در جای مناسب از آن استفاده نماییم . مثال:

```
#include <mega32.h>

#include <delay.h>


unsigned char i=0;


unsigned char count (void) {

i++;

delay_ms(300);

return i;

}

void main (void) {

DDRA=0xff;

PORTA=0x00;

while(1){

PORTA= count();

}

}
```

توضیح : در برنامه فوق ابتدا هدرفایل مربوط به میکروکنترلر Atmega32 به برنامه اضافه می شود . با اضافه شدن این فایل برنامه تمام رجیسترهای میکروکنترلر را می شناسد . سپس هدر فایل delay برای اینکه بتوان از تابع delay_ms استفاده کرد به برنامه اضافه شده است . سپس یک متغیر ۸ بیتی از نوع عدد بدون علامت (unsigned char) با مقدار اولیه صفر تعریف می شود . یک تابع دارای خروجی و بدون ورودی اعلان و بدنه آن تعریف شده است . در تابع main ابتدا رجیستر DDRA به منظور اینکه تمام ۸ بیت موجود در پورت A خروجی شود ، به صورت 0xff مقدار دهی شده است . مقدار اولیه منطق پایه های خروجی پورت A در خط بعدی همگی برابر ۰ شده است . در

نهایت در حلقه بی نهایت while ، تابع فراخوانی شده است و مقدار خروجی که تابع برمیگرداند در درون PORTA ریخته می شود . در صورتی که روی هر پایه از پورت LED قرار دهیم ، برنامه به صورت شمارنده عمل خواهد کرد و در هر مرحله تعدادی LED روشن می گردد.

نکته : نوع متغیر رجیسترها در هدرفایل mega32.h همگی به علت ۸ بیتی بودن رجیسترها در میکروکنترلرهای AVR، از نوع unsigned char می باشد.

کلاس های حافظه متغیرها

کلاس حافظه هر متغیر دو چیز اساسی را برای آن متغیر ، تعیین می‌کند:

•مدت حضور یا همان طول عمر (Life Time) آن متغیر

•محدوده قابل دسترسی بودن متغیر در برنامه(Scope)

پس با توجه به این دو مورد که در بالا ذکر شد، ما می‌توانیم برنامه‌هایی را بنویسیم که:

•از منابع حافظه کامپیوتر به خوبی بهره ببرند و بی مورد حافظه اشغال نشود.

•سرعت اجرای بالاتری دارند.

•دچار خطای کمتر و همچنین عیب یابی آسان تری باشند.

۴ نوع کلاس‌های حافظه در زبان C به صورت زیر تعریف شده است:

•اتوماتیک(Automatic)

•خارجی(External)

•استاتیک(Static)

•ثبات(Register)

نحوه تعریف کلاس حافظه یک متغیر

برای تعیین نوع کلاس حافظه برای متغیرها کافی است نام کلاس مورد نظر را در هنگام تعریف متغیر به ابتدای آن اضافه کنیم:

; مقدار اولیه = نام متغیر نوع متغیر نوع کلاس حافظه

که نوع کلاس حافظه با استفاده از کلمات کلیدی) auto برای کلاس حافظه اتوماتیک (،) staticبرای کلاس حافظه استاتیک (،) registerبرای کلاس حافظه ثبات) و) externبرای کلاس حافظه خارجی) تعیین می‌گردد . به عنوان مثال در کد زیر دو متغیر a و) b با مقدار دهی اولیه ۱۰ برای(b ، از نوع عدد صحیح تعریف شده اند که کلاس حافظه آن‌ها static می‌باشد.

```
static int a,b=10;
```

کلاس حافظه اتوماتیک

این کلاس که پر کاربردترین کلاس حافظه هست با کلمه کلیدی auto مشخص می‌شود . اگر نوع کلاس حافظه متغیری را ذکر نکنیم، کامپایلر خود به خود auto در نظر می‌گیرد . متغیرهایی که در داخل توابع تعریف می شوند از این نوع هستند که با فراخوانی تابع به طور اتوماتیک ایجاد می شوند و با پایان یافتن تابع به طور اتوماتیک از بین می روند . این نوع متغیرها دارای خواص زیر هستند:

1. به صورت محلی (Local) هستند . یعنی در داخل بلاکی که تعریف شده اند، قابل دسترسی اند.
2. هنگام ورود یک متغیر به یک تابع یا بلاک، به آن حافظه اختصاص داده می‌شود و این حافظه هنگام خروج از تابع یا بلاک، پس گرفته می‌شود.
3. چندین بار می‌توانند مقدار اولیه بگیرند.

کلاس حافظه ثبات

متغیرهای کلاس حافظه ثبات (register) در صورت امکان در یکی از ثبات‌های) CPU در AVR یکی از ۳۲ رجیستر همه منظوره) قرار می‌گیرند؛ لذا سرعت انجام عملیات با آن‌ها به علت نزدیک بودن و دسترسی مستقیم CPUبه آن بسیار بالاست و در نتیجه موجب افزایش سرعت اجرای برنامه می‌شود . معمولا متغیرهای شمارنده حلقه های تکرار را از این نوع تعریف می کنند . این کلاس دارای ویژگی ها و محدودیت‌های زیر است:

1. همان طور که در بالا ذکر شد، متغیر از نوع ثبات در صورت امکان در یکی از ثبات‌های CPU قرار می‌گیرد . زیرا به دلیل کم بودن تعداد ثبات‌هایCPU ، تعداد محدودی متغیر می‌توانند در ثبات‌ها قرار بگیرند. پس اگر تعداد متغیر هایی که از نوع کلاس حافظه ثبات تعریف شده اند زیاد باشند، کامپایلر کلاس حافظه ثبات را از متغیرها حذف می‌کند.
2. کلاس حافظه ثبات تنها می‌تواند برای متغیرهای محلی و همچنین پارامترهای تابع به کار گرفته شود.
3. انواع متغیر که می‌توانند دارای کلاس حافظه ثبات باشند، در کامپیوترهای مختلف، متفاوت است . دلیل این امر هم این است که متغیرهای مختلف، تعداد بایت متفاوتی را به خود اختصاص می‌دهند.
4. آدرس در مفهوم کلاس حافظه ثبات بی معنی است زیرا متغیرها در ثبات‌های CPU قرار می‌گیرند و نه در RAM. پس در مورد ان کلاس حافظه، نمی‌توان از عملگر &برای اشاره به آدرس متغیرها استفاده کرد.

کلاس حافظه خارجی

اگر برنامه‌هایی که می نویسیم، طولانی باشند، می‌توانیم آن را به قسمت‌های کوچکتری تقسیم کنیم که به هر قسمت آن واحد (یا همان Unit) گفته می‌شود . اگر بخواهیم که متغیر هایی را که در واحد اصلی تعریف شده اند را در واحدهای فرعی استفاده کنیم و دیگر آنها را دوباره در واحدهای فرعی تعریف نکنیم، می‌توانیم متغیرهای مورد نظر را با استفاده از کلاس حافظه خارجی تعریف کنیم . بدین منظور باید این متغیرها در واحد اصلی به صورت عمومی تعریف شده باشند و در واحد فرعی از کلمه کلیدی extern قبل از تعریف این متغیرها استفاده کنیم.

طول عمر متغیر هایی که از کلاس حافظه externهستند، از هنگام شروع برنامه تا پایان آن است و همچنین این متغیرها در سراسر برنامه قابل دسترسی هستند . طول عمر آنها برابر با طول عمل بلاک می باشد . دستور extern به

کامپایلر اعلام می کند که برای این متغیر ها ، حافظه جدیدی در نظر نگیرد ، بلکه از همان حافظه ای که در جای دیگر برنامه به آن اختصاص یافته استفاده کند.

کلاس حافظه استاتیک

این کلاس را می‌توانیم برای دو دسته از متغیرها به صورت زیر به کار ببریم:

•متغیرهای استاتیک محلی

•متغیرهای استاتیک عمومی

- متغیرهای استاتیک محلی در توابعی کاربرد دارند که نمی خواهیم مقداری که متغیر موجود در تابع به خود گرفته با خاتمه عملیات تابع پاک شود . بنابراین در توابعی که متغیرها با کلاس حافظه استاتیک معرفی شده باشند ، بعد از خاتمه عملیات تابع ، مقدار نهایی خود را حفظ کرده و در فراخوانی بعدی تابع آن مقدار را به خود می گیرد . متغیرهای تعریف شده در این کلاس دارای خواص زیر می باشد:

1. فقط در همان تابعی که تعریف شده اند، قابل دسترسی اند.
2. می توانند مقدار اولیه بگیرند و فقط یکبار مقدار دهی اولیه را دریافت می‌کنند.
3. در هنگام خروج از تابع، مقادیر متغیرها، آخرین مقداری خواهد بود که در تابع به آن‌ها اختصاص یافته است و هنگام اجرای دوباره تابع، مقدار اولیه نمی‌گیرند.

- متتغیرهای استاتیک عمومی فقط در یک واحد از برنامه، از جایی که تعریف می‌شوند، به بعد قابل دسترسی اند . استفاده از متغیر های استاتیک عمومی از یک طرف موجب می شود تا متغیر ها در جایی که به آنها نیاز است تعریف شوند و از طرف دیگر ، فقط توابعی که به آنها نیاز دارند می توانند از آنها استفاده کنند.

ثابت ها در زبانC

ثابت یک مقدار مشخص است که در ابتدای برنامه قبل از main تعریف می شود و یک نام به آن تعلق می گیرد. این مقدار هیچگاه قابل تغییر توسط کدهای برنامه نمی باشد. معمولا مقادیر ثابت عددی که در طول برنامه زیاد تعریف می شود را یک ثابت با نامی مشخص تبدیل می کنند .برای تعریف ثابت می توان به دو روش زیر عمل کرد.

1. استفاده از دستورConst

مثال:

```
const float pi=3.14;
```

2. استفاده از دستور#define

مثال:

```
#define pi 3.14
```

تعریف ثابت ها معمولا در ابتدای برنامه با استفاده از دستور #define صورت می گیرد زیرا این دستور پیش پردازنده بوده و به بهبود برنامه کمک می کند . به طور کلی در زبان سی دستوراتی که با #آغاز می شوند پیش پردازنده هستند یعنی کامپایلر ابتدا آنها را پردازش و سپس بقیه برنامه را کامپایل می کند.

دستورات شرطی در زبان C

دستور شرطی if

```
if( شرط ) دستور;

if( شرط ) {دستورات;}

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

دستور شرطی switch

Switch(عامل مورد شرط)

{

Case مقدار اول :

کدهایی که در صورت برابر بودن عامل شرط با مقدار اول باید اجرا شود

Break;

Case مقدار دوم :

کدهایی که در صورت برابر بودن عامل شرط با مقدار دوم باید اجرا شود

Break;

.

.

.

Default :

کدهایی که در صورت برابر نبودن عامل شرط با هیچ یک از مقادیر باید اجرا شود

}

حلقه های تکرار در زبان C

حلقه while

در ابتدا شرط حلقه مورد بررسی قرار می گیرد اگر شرط برقرار باشد یکبار کدهای درون حلقه اجرا می شود و دوباره شرط حلقه چک می شود و این روند تا زمانی که شرط برقرار است ادامه می یابد.

while (شرط حلقه)

{ ; کدهایی که تا زمان برقراری شرط حلقه تکرار می شود }

برای اینکه شرط در آخر بررسی شود از حلقه do…while استفاده می شود.

حلقه for

For(مقدار اولیه شمارنده حلقه ; شرط حلقه ; گام حرکت حلقه)

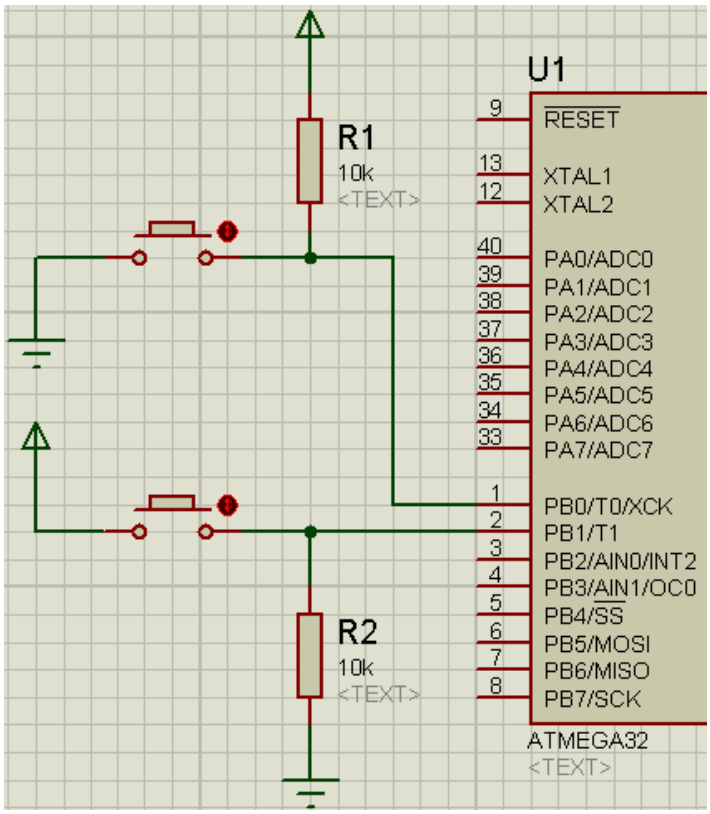
{ ; کدهایی که تا زمان برقراری شرط حلقه تکرار می شود }

دستور breakو continueدر حلقه ها

برنامه با دیدن دستور break در هر نقطه از حلقه ، در همان نقطه از حلقه خارج شده و کدهای زیر حلقه را اجرا می کند . دستور break در حلقه خروج بدون شرط از حلقه است. برنامه با دیدن دستور continueدر هر نقطه از حلقه ، کدهای زیر دستور continue را رها کرده و به ابتدای حلقه باز می گردد.

اتصال کلید به میکرو

یکی از ساده ترین و پرکاربرد ترین دستگاههای ورودی کلید است . توسط کلید میتوان منطق ۰ یا ۱ را به میکرو وارد کرد و بر اساس آن پردازش را انجام داد. برای خواندن منطق کلید از رجیستر PINاستفاده می شود . نحوه استفاده از رجیستر PIN به این صورت است که " : اگر کلید زده شد آنگاه کار مورد نظر انجام شود " . عبارت اخیر معادل استفاده از دستور شرطی if به صورتی است که اگر کلید زده شده بود یک منطق و در غیر این صورت منطق دیگری وارد پایه میکرو شود . بنابراین اتصال کلید را به دو صورت PullUp و PullDown انجام می شود که در شکل زیر مشاهده می کنید . در کلید pull up در حالتی که کلید زده نشده منطق ۱ و در حالت فشردن کلید منطق ۰ وارد میکرو می شود . معمولا از مقاومت ۱۰ kبرای این کار استفاده می شود.



بنابراین برای فهماندن کلید به میکرو به صورتی که " اگر کلید زده شد آنگاه کار مورد نظر انجام شود " برای کلیدهای فوق به صورت زیر می شود:

```
if(PINB.0==0) {...} //ForPullUp

if(PINB.1==1) {...} //ForPulDown
```

اما مشکلی که در کلید وجود دارد ، بوجود آمدن لرزش یا bounce در هنگام قطع و وصل کلید است . در زمانی که کاربر کلید را فشار می دهد تقریبا ۲۰ میلی ثانیه طول می کشد تا منطق کلید از ۰ به ۱ (یا بالعکس) ثابت شود. تا قبل این بعلت وجود جرقه کلید منطق ثابتی ندارد در زمانی که کاربر دست خود را از روی کلید بر می دارد نیز تقریبا ۲۰ میلی ثانیه طول می کشد تا ۰ و ۱ شدن های کلید تمام شده و کلید به منطق اولیه خود برگردد . در واقع این مشکل زمانی برای ما مسئله ساز می شود که زمانی که کاربر کلید را یکبار فشار می دهد و انتظار دارد تا میکرو متوجه یکبار فشار دادن آن شود اما به علت قرار گرفتن if در درون حلقه نامتناهی while چندین بار شرط PINB.0==0 برقرار شده و کار مورد نظر چندین بار انجام می شود . راه حل این مشکل ساده است و آن هم قرار دادن مقداری delayدر حلقه if است . بنابراین برای حل این مشکل بسته به نوع برنامه یکی از سه روش زیر قابل استفاده است:

کلید نوع ۱:

```
if(PINB.0==0) {

دستورات مربوط به بعد از زدن کلید

delay_ms(200);

}
```

توضیح : به محض فشار دادن کلید توسط کاربر شرط if برقرار شده و دستورات مورد نظر اجرا می شود سپس به علت ایجاد تاخیر زیاد توسط تابع delay_ms (در اینجا ۲۰۰ میلی ثانیه) با این کار احتمال اینکه زمانی که برنامه در حلقه while به if می رسد و شرط برقرار باشد ، کاهش می یابد . مزیت این کلید این است که در صورتی که کاربر کلید را فشار داده و نگه دارد تقریبا در هر ۲۰۰ میلی ثانیه یکبار کار مورد نظر صورت می گیرد . عیب این روش نیز این است که هنوز احتمال دارد که زمانی که یکبار کلید زده شود دوبار کار مورد نظر انجام شود.

کلید نوع ۲:

```
if(PINB.0==0) {

delay_ms(20);

while(PINB.0==0);
```

```
دستورات مربوط به بعد از زدن کلید

}
```

توضیح : به محض فشار دادن کلید توسط کاربر شرط if برقرار شده و برنامه به مدت ۲۰ میلی ثانیه صبر می کند تا منطق کلید ثابت شود و از منطقه bounce عبور کند سپس توسط حلقه whileبا همان شرط برقراری کلید در این مرحله برنامه تا زمانی که کلید توسط کاربر فشرده شده است در حلقه گیر می کند و هیچ کاری انجام نمی دهد . به محض اینکه کاربر دست خود را بر می دارد ، شرط برقرار نبوده و خط بعدی یعنی دستورات مربوطه اجرا می شود . مزیت این روش این است که در هر بار فشردن کلید برنامه تنها یکبار اجرا می شود . معایب این روش این است که تا زمانی که کاربر کلید را نگه داشته اتفاقی نمی افتد و به محض رها کردن کلید کار مورد نظر انجام می شود.

کلید نوع ۳:

```
if((PINB.0==0 ) && (flag==0)) {

flag=1; start=!start; }

else if ( PINB.0 == 1) flag=0;

if(start){

دستورات مربوط به بعد از زدن کلید

}
```

توضیح : این کلید به صورت start/stop عمل می کند یعنی بار اولی که کاربر کلید را فشار می دهد دستورات مربوط به بعد از زدن کلید دائما اجرا می شود تا زمانی که کاربر دست خود را از روی کلید رها کرده و دوباره کلید را فشار دهد ، در این صورت دستورات دیگر اجرا نمی شود . دو متغیر از نوع bit با نام های flag و start با مقدار اولیه ۰ برای این کلید باید تعریف شود . زمانی که کاربر برای اولین بار کلید را فشار می دهد شرط if برقرار شده و flag=1 و start=1 می شود . در این صورت شرط if دوم برقرار بوده و دستورات مربوطه با هر بار چرخش برنامه درون حلقه نامتناهی while یکبار اجرا می شود . زمانی که کاربر دست خود را از روی کلید بر می دارد و منطق ۱ وارد میکرو می شود flag=0 شده و برنامه دوباره آماده این می شود که کاربر برای بار دوم کلید را فشار دهد . زمانی که کاربر بار دوم کلید را می فشارد start=0 شده و دستوراط مربوطه اجرا نخواهد شد سپس با برداشته شدن دست کاربر از روی کلید،

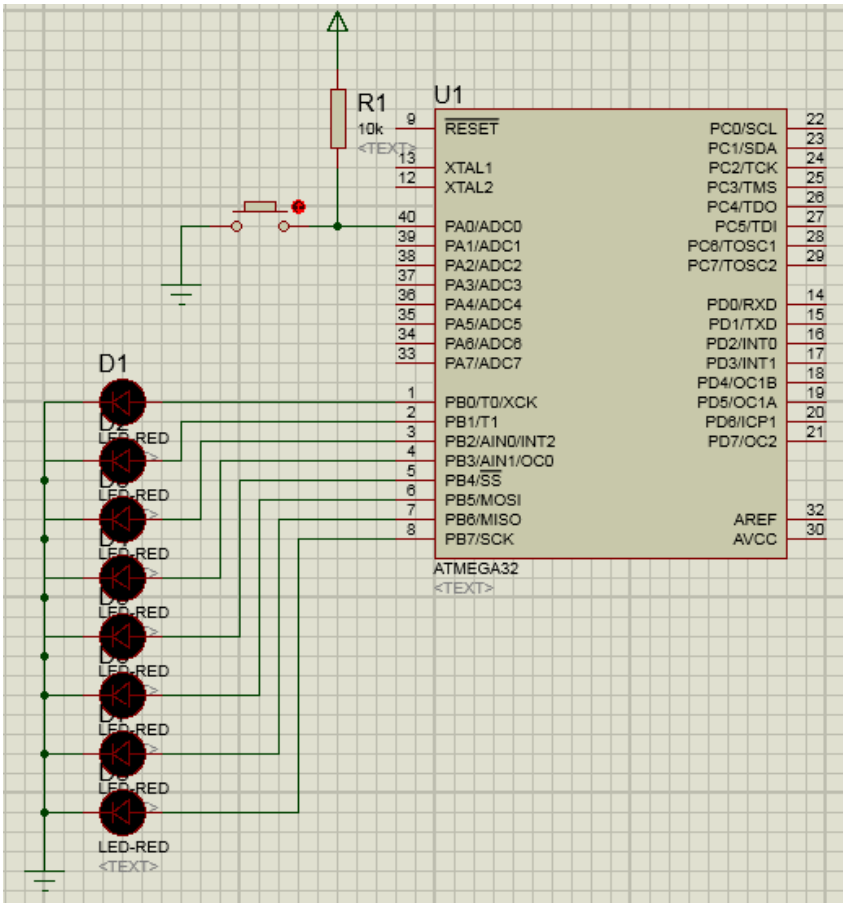
همه چیز به حالت اول بر میگردد . این کلید طوری نوشته شده است که bounce در آن کمترین تاثیر مخرب ممکن را دارد.

مثال عملی شماره ۲ : برنامه ای بنویسید که یک کلید روی پورت PA0 و ۸ عدد LED روی پورت B وجود داشته باشد و با هر بار زدن کلید ، led های موجود به صورت :

1. شمارنده بالا شمار
2. شمارنده پایین شمار
3. شمارنده جانسون
4. شمارنده حلقوی

عمل نماید . سعی کنید برنامه کمترین عیب ممکن را داشته باشد و کلید به بهترین نحو ممکن کار کند.

حل : بعد از طراحی سخت افزار و نرم افزار روی کاغذ به سراغ نرم افزار proteus رفته و مدار طراحی شده را به صورت شکل زیر رسم می کنیم.



سپس به سراغ نرم افزار CodeVision رفته و طبق مراحل گفته شده در فصل قبل پروژه جدید را می سازیم و فرکانس میکرو را روی ۱ Mhzداخلی قرار می دهیم . برنامه خواسته شده به صورت زیر است.

```
#include <mega32.h>

#include <delay.h>


unsigned char i,j=0;

bit flag=1;


void main (void) {

DDRB=0xff;

PORTB=0x00;


while(1){

if(PINA.0==0){

 delay_ms(25);

 i++;

 if(i==5) i=0;

 j=0;

 PORTB=0x00;

 flag=1;

 while(PINA.0==0);
```

```
 }
if(i==1){
 PORTB++;
 delay_ms(200);
 }
if(i==2){
 PORTB--;
 delay_ms(200);
 }
if(i==3){
 PORTB=0x01;
 PORTB=PORTB<<j;
 delay_ms(200);
 if(flag) j++; else j--;
 if(j==7) flag=0;
 if(j==0) flag=1;
 }
if(i==4){
 if(flag) PORTB=PORTB+(0x01<<j);
 else     PORTB=PORTB-(0x01<<j);
 delay_ms(200);
```

```
if(flag) j++; else j--;

if(j==8) flag=0;

if(j==255) flag=1;

}

}

}
```

توضیح : متغیر i حالت کار میکرو را نشان می دهد که با هر باز زدن کلید یک واحد به متغیر i اضافه می شود . در حالت پیش فرض مقدار این متغیر ۰ است و برنامه هیچ کاری انجام نمی دهد تا زمانی که کاربر کلید را فشار دهد . با اولین باری که کلید زده می شود i=1 شده و برنامه مربوط به شمارنده بالا شمار اجرا می شود . با دومین باری که کلید زده می شود i=2 شده و برنامه مربوط به شمارنده پایین شمار اجرا می شود . دفعه سوم i=3 شده و برنامه به صورت شمارنده حلقوی و در نهایت زمانی که برای بار چهارم کلید زده شود i=4 شده و برنامه شمارنده جانسون می شود . متغیر flag از آن جهت ایجاد شده است که زمانی که شمارنده جانسون و حلقوی به آخر رسیدند ، برنامه از آخر به اول شروع به کار کند و زیباتر می شود.

نکته : همانطور که مشاهده کردید در برنامه از حلقه for استفاده نکردیم تا برنامه روان تر و حرفه ای تر باشد.

سورس کدویژن و پروتئوس برنامه فوق را می توانید از لینک زیر دانلود و نحوه کار را مشاهده نمایید .

دانلود مثال عملی شماره ۲

آرایه ها در C

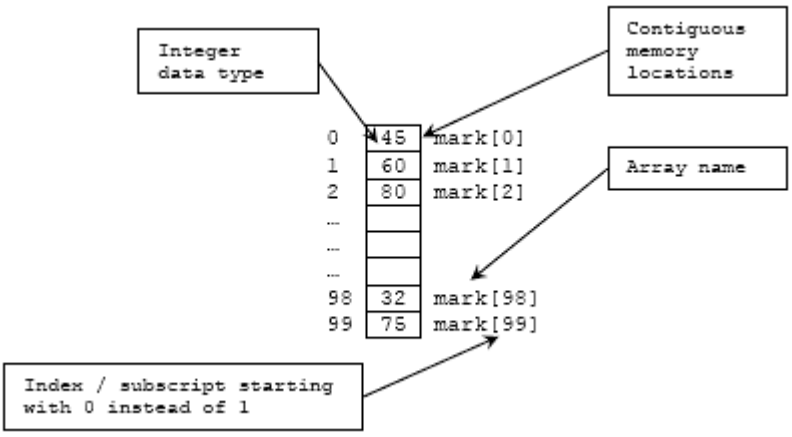
آرایه اسمی برای چند متغیر هم نوع می باشد یا به عبارت دیگر آرایه از چندین کمیت درست شده است که همگی دارای یک نام می باشد و در خانه های متوالی حافظه ذخیره می گردند. هر یک از این کمیت ها را یک عنصر می گویند، برای دسترسی به عناصر آرایه باید اسم آرایه و شماره ی اندیس آرایه را ذکر کنیم. آرایه ها در زبان سی از جایگاه ویژه ای برخوردارند، به طوری که در پروژه های خود به طور مکرر به آن برخورد خواهید کرد زیرا ارسال و دریافت داده به صورت رشته (آرایه ای از کاراکترها) و سریال انجام می شود.

تعریف آرایه یک بعدی:

نوع متغیر نام آرایه [تعداد خانه ها] = { مقدار اول , مقدار دوم , ... };

با تعریف آرایه به همان مقدار خانه های حافظه بسته به نوع متغیر و تعداد خانه های آرایه تخصیص می یابد که در شکل زیر مثالی از آن را مشاهده می کنید . میزان حافظه ای که به آرایه اختصاص داده می شود،به این شکل استفاده می شود:

(طول آرایه) ضرب در (طول نوع آرایه) = میزان حافظه آرایه (برحسب بایت)



برای دسترسی به هر یک از خانه ها آدرس آن را لازم داریم. آدرس هر خانه از ۰ تا n-1 است که در آن n تعداد خانه های تعریف شده است . هر عضو آرایه به تنهایی می تواند در محاسبات شرکت کند . مثال:

unsigned char a[5]={7,12,0,99,1};

```
a[0]=a[4]*a[2];
```

نکته : اندیس آرایه خود می تواند متغیر باشد و این قابلیت می تواند در برنامه ها کاربرد زیادی داشته باشد.

نکته : در صورتی که می خواهید آرایه به صورت دائمی ذخیره شود (مثلا وقتی که میخواهید لوگو شرکت خود را همیشه داشته باشید) باید به ابتدای تعریف کلمه کلیدی eeprom یا flashرا اضافه کنید تا در حافظه های دائمی ذخیره گردد.

آرایه های چند بعدی

در تعریف آرایه دو بعدی باید ۲ اندیس و در تعریف آرایه سه بعدی باید ۳ اندیس و در تعریف آرایه n بعدی باید n اندیس را ذکر کرد . آرایه های چند بعدی در نمایشگر های lcd کاربرد دارند . به عنوان مثال:

```
int table [10] [10];
```

یک آرایه دو بعدی بنام table را تعریف میکند که دارای ۱۰ سطر و ۱۰ ستون است و نوع عناصر آن int است.

```
int k [5] [10] [15];
```

آرایه ای سه بعدی بنام k را تعریف می کند که دارای ۵ سطر ، ۱۰ ستون و ۱۵ ارتفاع است و نوع عناصر آن int می باشد.

مقدار دهی به آرایه های چند بعدی

برای مقدار دهی به آرایه های دو بعدی سطر ها را به ترتیب پر می کنیم . مثال:

```
Int a[2][3]={ {3,1,2} , {7,4,6} }
```

برای مقدار دهی به آرایه های سه بعدی ابتدا سطرهای طبقه اول و سپس سطرهای بقیه طبقات را مقدار دهی می کنیم . مثال:

```
Int a[2][3][4]={ { {1,2,3,4} ,{5,4,3,2} , {2,3,4,5} } , { {7,6,5,4} , {9,0,8,7} , {6,7,4,1} } }
```

رشته ها

در زبان سی برای نمایش کلمات و جملات از رشته ها استفاده می شود . رشته همان آرایه ای از کاراکتر ها است که حاوی اطلاعاتی می باشد . تمام کاراکتر ها شامل اعداد و حروف و برخی کاراکترهای دیگر که روی صفحه کلید کامپیوتر وجود دارند ، دارای کد شناسایی ASCII (American Standard Code For Information Interchange) می باشند . کدهای اسکی که توسط استاندارد آمریکایی در سال ۱۹۶۷ ابداع شد و در سال ۱۹۸۶ دست خوش تغییراتی شد.

کاراکتر ست اسکی خود به دو نوع تقسیم می‌شود. نوع ۷ بیتی که با نام اسکی استاندارد (Standard ASCII) شناخته شده و دارای ۲ به توان ۷ یعنی ۱۲۸ کاراکتر مختلف است که از ۰ تا ۱۲۷ استفاده می‌شوند. نوع دیگر آن حالت ۸ بیتی است که با نام اسکی توسعه یافته (Extended ASCII)شناخته شده و دارای ۲ به توان ۸ یعنی ۲۵۶ کاراکتر مختلف است که از ۰ تا ۲۵۵ استفاده می‌شود. حالت توسعه یافته جدا از حالت استاندارد نیست بلکه از ۰ تا ۱۲۷ کاراکتر اول آن درست مانند حالت استاندارد بوده و فقط بقیه کاراکترها (از ۱۲۸ تا ۲۵۵) به آن اضافه شده است . کاراکترهای اضافی دارای هیچ استانداردی نبوده و ممکن است در دستگاه‌ها و کامپیوترهای مختلف فرق داشته باشد و به منظور ایجاد کاراکترهای زبان دوم) مثلا زبان فارسی) ایجاد شده است . یعنی ممکن است در یک کامپیوتر کاراکتر اسکی ۱۵۰ معادل حرف û و در کامپیوتر دیگر که روی زبان دوم فارسی تنظیم شده است ، معادل حرف ب باشد . اما کاراکترهای قبل از ۱۲۸ همگی ثابت هستند .کاراکترهای فارسی در اینکدینگ Iranian System شرکت ایرانیان سیستم که یکی از قدیمی ترین اینکدینگ‌های ASCII فارسی است را می‌توانید در این لینک ببینید.

در هر دو نوع ذکر شده (۷ و ۸ بیتی) تعداد ۳۲ کاراکتر اول (یعنی از ۰ تا ۳۱) و آخرین کاراکتر (۱۲۷) با عنوان کاراکترهای کنترلی (Control Characters) شناخته می‌شود. این کاراکترها غیرقابل چاپ بوده و فقط برای کنترل متن مورد استفاده قرار می‌گیرد (مثلاً مشخص کننده ابتدای هدر، حذف، کنسل و ...). بقیه کاراکترها یعنی از ۳۲ تا ۱۲۶ قابل چاپ هستند. این کاراکترها شامل نمادها، حروف و اعداد انگلیسی هستند .در حالت توسعه یافته، از کاراکترهای ۱۲۸ تا ۲۵۵ نیز قابل چاپ هستند.

در شکل زیر این ۹۵ کاراکتر قابل چاپ انگلیسی را به همراه کد اسکی آن در مبنای دسیمال مشاهده می کنید.

Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char
32	[space]	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	[backspace]

تعریف یک کاراکتر

تعریف یک کاراکتر توسط نوع متغیر char صورت می گیرد و مقدار اولیه آن داخل کوتیشن (' ') قرار می گیرد . در زبان سی وقتی یک حرف بین ' و ' قرار می گیرد کد اسکی آن درون متغیر ذخیره می شود . مثال:

```
char c='H';
```

تعریف رشته (آرایه ای از کاراکتر ها)

برای تعریف رشته از آرایه ای از کاراکترها استفاده می شود و مقدار اولیه آن داخل دابل کوتیشن (" ") قرار می گیرد . مثال:

```
char s[10]="Hello!";
```

اگر تعداد خانه های آرایه ذکر شود ، آرایه سایز مشخصی دارد و در صورتی که تعداد کاراکترهای عبارت یا جمله ای که درون آن میریزیم بیشتر از سایز آرایه باشد ، عبارت ناقص ذخیره خواهد شد و در صورتی که تعداد کاراکترهای مورد نظر کمتر باشد بقیه آرایه خالی خواهد بود . اگر تعداد خانه های آرایه ذکر نشود یعنی سایز آرایه بر اساس مقداری که درون آن ریخته می شود محاسبه شود . مثال:

```
char str[]="Hello!..."
```

عملگرها

با استفاده از عملگرها می توان روی اعداد ، متغیرها ، آرایه ها ، رشته ها و ... عملیات حسابی ، منطقی ، مقایسه ، بیتی ، بایتی و ... انجام داد.

عملگرهای محاسباتی

عملگر	نام	مثال
-	تفریق و علامت منفی	z = x - y یا -x
+	جمع	z = x + y
*	ضرب	z = x * y
/	تقسیم	z = x / y
%	باقیمانده تقسیم	z = x % y
--	یک واحد کاهش	x -- یا -- x
++	یک واحد افزایش	x ++ یا ++ x

تقدم عملگر های محاسباتی

تقدم	عملگر
۱	++ و --
۲	- علامت منفی
۳	/ و * و %
۴	+ و -

عملگرهای رابطه ای و منطقی

عملگر	نام	مثال
>	بزرگتر	x > y
>=	بزرگتر مساوی	x >= y
<	کوچکتر	x < y
<=	کوچکتر مساوی	x <= y
==	متساوی	x == y
!=	نا مساوی	x != y
!	نقیض	!(x > y)
&&	و	x > y && z > w
||	یا	x > y || z > w

تقدم عملگرهای رابطه ای و منطقی

تقدم	عملگر
۱	!
۲	< و =< و > و =>
۳	== و =!
۴	&&
۵	||

عملگرهای ترکیبی

عملگر	نام	مثال
+=	انتساب جمع	x += y
-=	انتساب تفریق	x -= y
*=	انتساب ضرب	x *= y
/=	انتساب تقسیم	x /= y
%=	انتساب باقیمانده تقسیم	x %= y

تقدم عملگرهای ترکیبی

تقدم	عملگر
۱	%= و *= و /=
۲	-= و +=

تعریف عملگرهای بیتی

x	y	x & y	x | y	x ^ y	~x
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

جدول صحت عملگر های بیتی

عملگرهای بیتی

عملگر	نام	مثال
&	And	x = 11101101 , y = 00001111 → x & y = 00001101
|	Or	x = 11101101 , y = 00001111 → x | y = 11101111
^	Xor	x = 11101101 , y = 00001111 → x ^ y = 11100010
~	Not	x = 00001111 → x = ~ x → x = 11110000
<<	شیفت به چپ	x = 00000111 → x = x << 2 → x = 00011100
>>	شیفت به راست	x = 11000000 → x = x >> 3 → x = 00011000

تقدم عملگرهای بیتی

تقدم	عملگر
۱	~
۲	<< و >>
۳	&
۴	^
۵	|

تقدم کلی در عملگرها

تقدم	عملگر	تقدم	عملگر
۸	&	۱	()
۹	^	۲	! و ~ و ++ و - و sizeof
۱۰	|	۳	/ و * و %
۱۱	&&	۴	+ و -
۱۲	||	۵	>> و <<
۱۳	?	۶	> و =< و < و =>
۱۴	/= و *= و %= و -= و +=	۷	!= و ==

همانطور که مشاهده می شود عملگر پرانتز دارای بیشترین اولویت است . با استفاده از این عملگر میتوان انجام محاسبات را اولویت داد به طوری که اولویت اول همیشه با داخلی ترین پرانتز است و سپس به ترتیب تا پرانتز بیرونی اولویت دارند.

تبدیل نوع در محاسبات

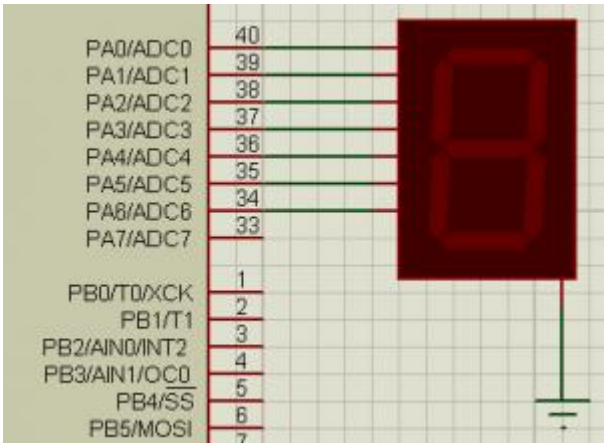
برای درک بهتر در مورد تبدیل نوع در محاسبات ، به مثال زیر توجه کنید.

```
int a=9,b=2;

float x,y,z;

x=a/b;

y=(float) a/b;

z=9.0/2.0;
```

در مثال بالا با اینکه متغیرهای x و y و z هر سه از نوع float هستند ولی مقدار x برابر با ۴ و مقدار y و z برابر با ۴،۵ است. اگر متغیر های a و b از نوع float بودند ، مقدار x نیز برابر ۴،۵ می شد اما چون متغیر های a و b از نوع intهستند باید یک تبدیل نوع در محاسبه توسط عبارت (float) در ابتدای محاسبه انجام شود.

اتصال سون سگمنت به میکرو

یکی از نمایشگرهای پرکاربرد سون سگمنت است که می توان توسط آن اعداد و برخی حروف ها را نشان داد . روش های متفاوت و مختلفی برای راه اندازی سون سگمنت توسط avr وجود دارد . ساده ترین روش اتصال سون سگمنت به میکرو استفاده از مداری به شکل زیر است . همانطور که مشاهده می شود در این روش از ۸ بیت (یک پورت) به طور کامل استفاده می شود) .نرم افزار proteus بیت هشتم که به digit سون سگمنت وصل می شود را ندارد(



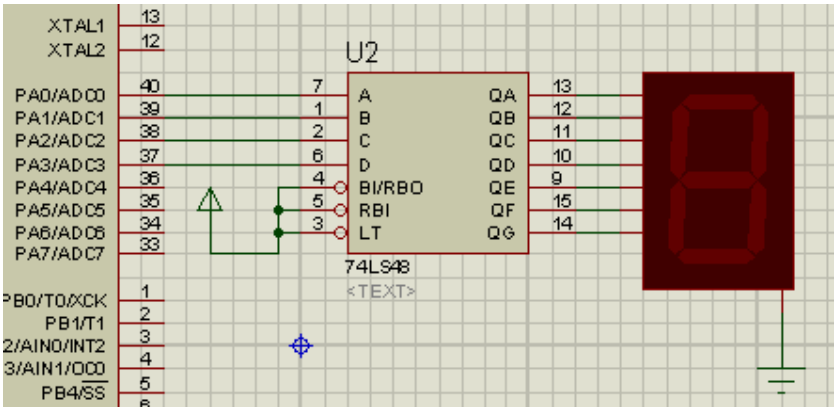
برای نشان دادن اعداد روی سون سگمنت کافی است پایه مربوطه را پس از خروجی کردن یک کنیم برای مثال برای نشان دادن عدد یک باید PA1 و PA2 را ۱ نمود بنابراین دستور PORTA=0x06 عدد ۱ را روی سگمنت نمایش می دهد . بنابراین در این روش برای راحتی هنگام کار با سون سگمنت آرایه زیر را که شامل کد سون سگمنت اعداد ۰ تا ۹ می باشد ، تعریف می کنیم.

```
unsigned char seg[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
```

هر عددی که بخواهیم نمایش دهیم کافی است داخل [ ] قرار دهیم تا به کد سون سگمنت تبدیل شود . برای مثال:

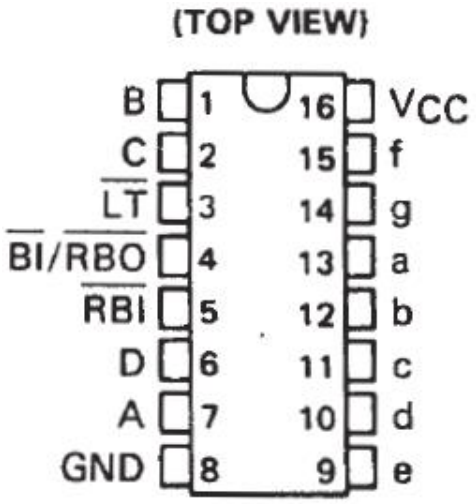
```
PORTA=seg[ i ];
```

همانطور که دیدید در این روش پایه های زیادی از میکرو صرف نمایش تنها یک سون سگمنت می شود. بجای استفاده از مدار فوق میتوان از آی سی ۷۴۴۸ استفاده کرد که در این صورت تنها ۴ بیت از پورت میکرو اشغال می شود و همچنین بعلت تبدیل اتوماتیک کدهای سون سگمنت دیگر نیازی به استفاده از آرایه تعریف شده در قبل نیست . همانطور که در شکل زیر نیز مشاهده می کنید در این اتصال نصف پورت اشغال می شود.



راهنمای آی سی ۷۴۴۸:

کار این آی سی خواندن یک عدد باینری ۴بیتی بوسیله پایه های ورودی اش و نمایش آن روی سون سگمنت می باشد . این آی سی ۱۶ پایه دارد و نحوه شماره بندی پایه های آن بصورت زیر می باشد:



نکته :در استفاده از آی سی ۷۴۴۸ ، حتما می بایست از سون سگمنت کاتد مشترک استفاده کنید. برای استفاده از سون سگمنت آند مشترک آی سی دیگری به نام ۷۴۴۷ وجود دارد.

شماره پایه	نام پایه	توضیح مختصر
۱	B	یکی از پایه های ورودی / ۰ یا ۱
۲	C	یکی از پایه های ورودی / ۰ یا ۱
۳	LT	به ولتاژ مثبت وصل شود / همیشه ۱
۴	BI/RBO	به ولتاژ مثبت وصل شود / همیشه ۱
۵	RBI	بدون هیچ اتصال / همیشه ۰
۶	D	یکی از پایه های ورودی / ۰ یا ۱
۷	A	یکی از پایه های ورودی / ۰ یا ۱
۸	GND	همیشه به ولتاژ منفی وصل می شود
۹	e	خروجی آی سی برای پایه e سون سگمنت/۰ یا ۱
۱۰	D	خروجی آی سی برای پایه d سون سگمنت/۰ یا ۱
۱۱	C	خروجی آی سی برای پایه c سون سگمنت/۰ یا ۱
۱۲	B	خروجی آی سی برای پایه b سون سگمنت/۰ یا ۱
۱۳	A	خروجی آی سی برای پایه a سون سگمنت/۰ یا ۱
۱۴	G	خروجی آی سی برای پایه g سون سگمنت/۰ یا ۱
۱۵	F	خروجی آی سی برای پایه f سون سگمنت/۰ یا ۱
۱۶	Vcc	همیشه به ولتاژ مثبت وصل می شود

نحوه کار آی سی:

ورودی این آی سی همانطور که در جدول مشاهده کردید ، شامل چهار پایه ۱ ، ۲ ،۶ و ۷ (که با حروف A ، B، Cو Dنشان داده می شود) می باشد. هر کدام از این پایه می توانند ۰ یا ۱ باشند (ولتاژ ۰ یا ۵ ولت داشته باشند). پس عدد ورودی ما به آی سی یک عدد باینری (در مبنای ۲) می باشد که فقط چهار رقم دارد.

از طرفی خروجی این آی سی کد های مخصوص سون سگمنت است که پس از نمایش روی سون سگمنت و در مبنای ۱۰ نمایش می یابد .(خروجی این آی سی تنها اعداد ۰ تا ۹ میتواند باشد)

تغذیه آی سی:

آی سی هایی که شماره آنها با ۷۴ شروع می شود ، اصطلاحا TTL نامیده می شوند. این آی سی ها نسبت به ولتاژ تغذیه بسیار حساسند و ولتاژ آنها می بایست دقیقا ۵ ولت باشد. حداکثر تغییرات ولتاژی که می توان اعمال کرد در حد ۴،۷۵ تا ۵،۲۵ ولت است.

پیاده سازی مدار روی بردبورد:

-ابتدا آی سی ۷۴۴۸ را روی بردبورد جا می زنیم (روی شیار وسط برد بورد قرار می گیرد)

-در قسمت دیگری از برد بورد یک سون سگمنت کاتد مشترک قرار داده و جا می زنیم(سون سگمنت نیز روی شیار برد بورد قرار می گیرد و هر دسته از پایه ها در یک سمت)

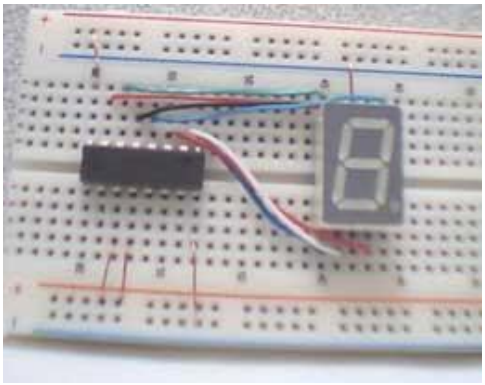
–مطابق جدول مربوط به پایه های آی سی ۷۴۴۸ ، پایه های ۳ و ۴ و ۱۶ را به ردیف ولتاژ مثبت برد بورد وصل می کنیم.

–پایه ۸ به ولتاژ منفی وصل می شود.

–یکی از پایه های مشترک سون سگمنت را به ولتاژ منفی وصل می کنیم.

–هر کدام از پایه های خروجی آی سی را با سیم به پایه هم نامش روی سون سگمنت وصل می کنیم.

–ورودی های ۷۴۴۸ نیز که باید به میکرو وصل شود مطابق شکل وصل می کنیم.

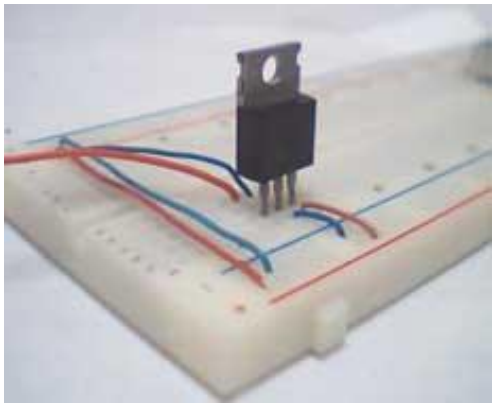


-در گوشه دیگری از بردبورد یک رگولاتور ۷۸۰۵ جا می زنیم.

–سیمی که ولتاژ مثبت آداپتور را حمل می کند به پایه input وصل می کنیم.

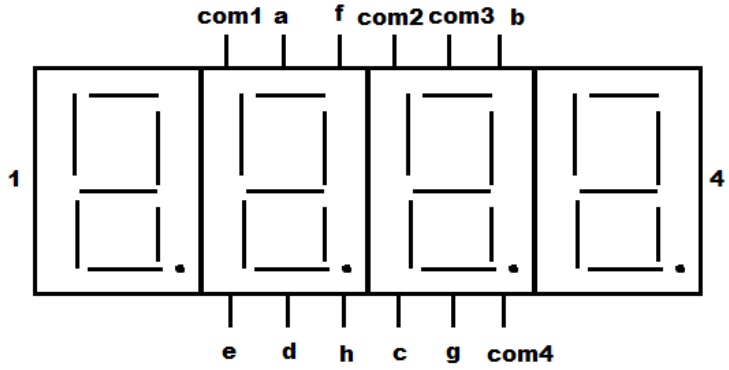
–سیمی که ولتاژ منفی آداپتور را حمل می کند به پایه Ground وصل می کنیم.

–پایه های output و Goround رگولاتور را به ردیف های مثبت و منفی کنار بردبورد وصل می کنیم تا آی سی ها از این خطوط تغذیه شوند.

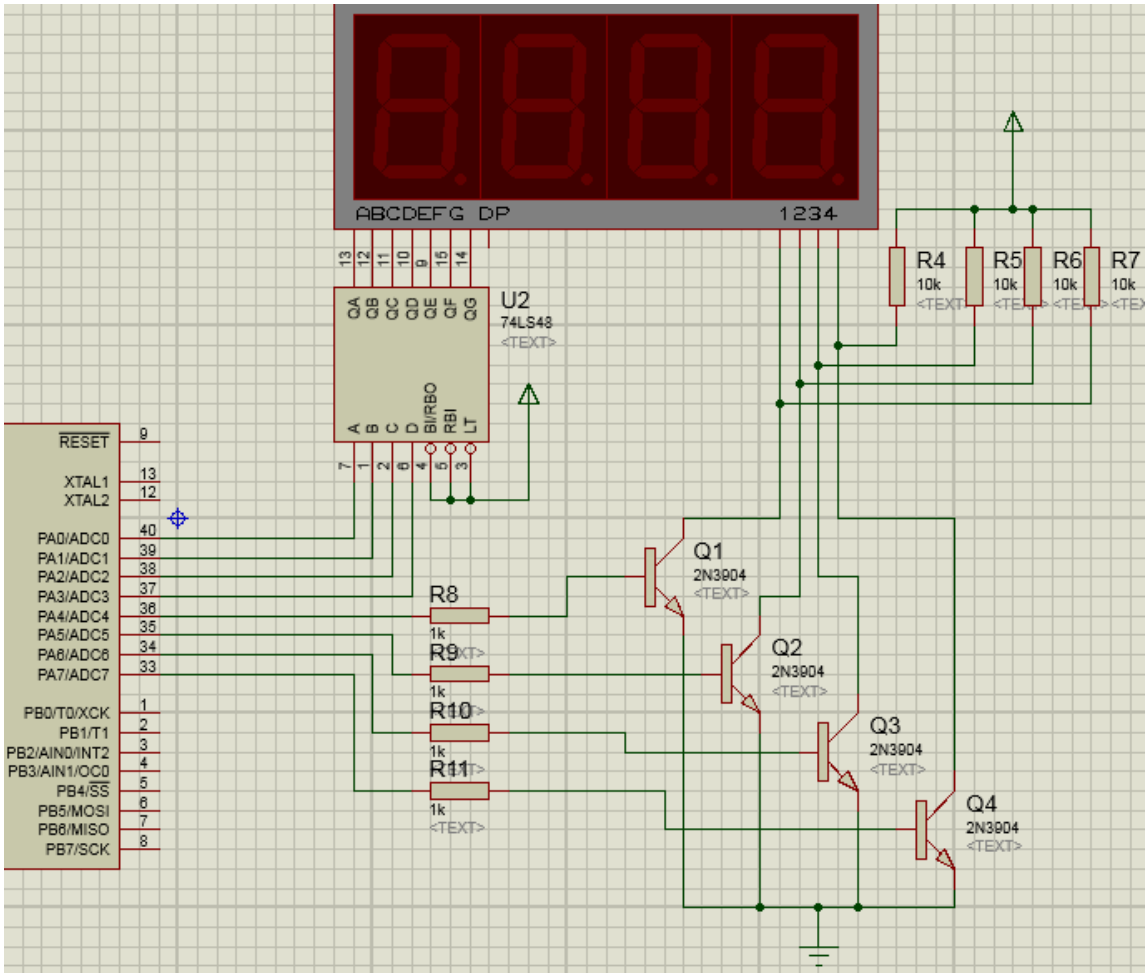


سون سگمنت های مالتی پلکس

در برخی طراحی ها ممکن است به بیش از یک سون سگمنت نیاز باشد . در این طراحی ها از سون سگمنت مالتی پلکس شده استفاده می شود. تنها تفاوت این سون سگمنت در این است که ۸ بیت دیتا (a ، b، ... c) برای همه سگمنت ها با هم یکی شده است (مشترک هستند). پایه های سون سگمنت مالتی پلکس شده ۴ تایی را در شکل زیر مشاهده می کنید . در صورت اتصال پایه های Com سون سگمنت مربوطه روشن می شود.



بنابراین مدار مورد نظر در این طراحی به صورت زیر است . وجود ترانزیستور npn در این طراحی بعلت تامین مناسب جریان برای هر سگمنت است. زیرا طبق دیتاشیت هر پایه میکرو جریان بیش از ۲۰ میلی آمپر را نمیتواند تامین کند اما در حالتی که چندین سگمنت همزمان روشن است جریانی در حدود ۱۰۰ میلی آمپر مورد نیاز است و بنابراین اگر ترانزیستور نباشد سگمنت ها به خوبی روشن دیده نخواهد شد.



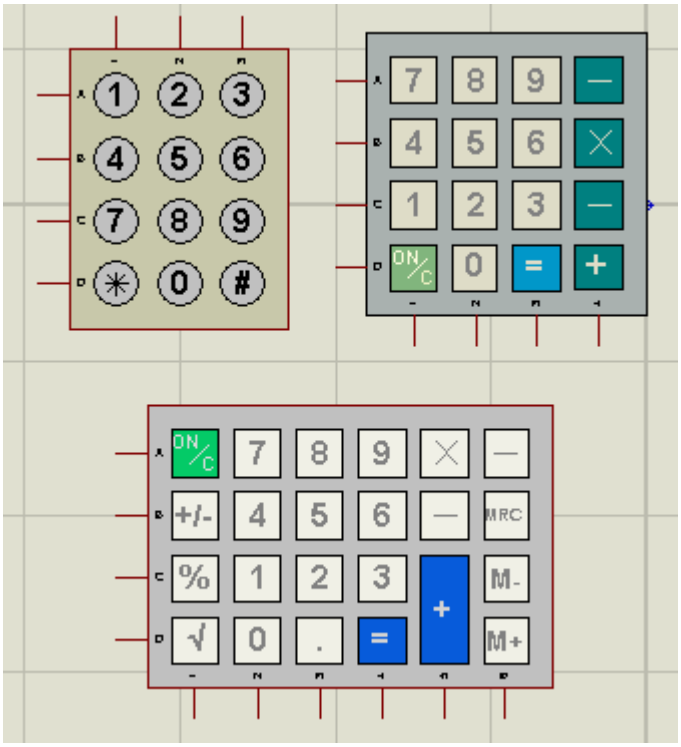
روش نمایش بر روی این نمایشگر به این صورت است که ابتدا همه کاتد ها خاموش است یعنی دارای منطق یک است (مقاومت های ۱۰ کیلو اهمی برای همین منظور طراحی شده اند) سپس دیتایی که میخواهیم روی سگمنت اول نمایش دهیم روی پایه های دیتا قرار می گیرد . سپس کاتد سگمنت اول را برای مدت کوتاهی صفر می کنیم تا عدد مورد نظر نشان داده شود (یک کردن Base ترانزیستور توسط میکرو باعث اتصال زمین به کاتد سون سگمنت می شود) سپس دوباره کاتد سگمنت اول را یک می کنیم و این کار را برای سگمنت های بعدی نیز تکرار می کنیم . اگر این کار با سرعت انجام شود همه سون سگمنت ها همزمان روشن دیده خواهد شد . برای استفاده از سون سگمنت شکل فوق تابعی به نام display به صورت زیر تعریف می کنیم.

```
void display(void)
{
 PORTA=data[0];
 PORTA.4=1;
 delay_ms(5);
 PORTA.4=0;

 PORTA=data[1];
 PORTA.5=1;
 delay_ms(5);
 PORTA.5=0;

 PORTA=data[2];
 PORTA.6=1;
 delay_ms(5);
 PORTA.6=0;

 PORTA=data[3];
 PORTA.7=1;
 delay_ms(5);
 PORTA.7=0;
 }
```

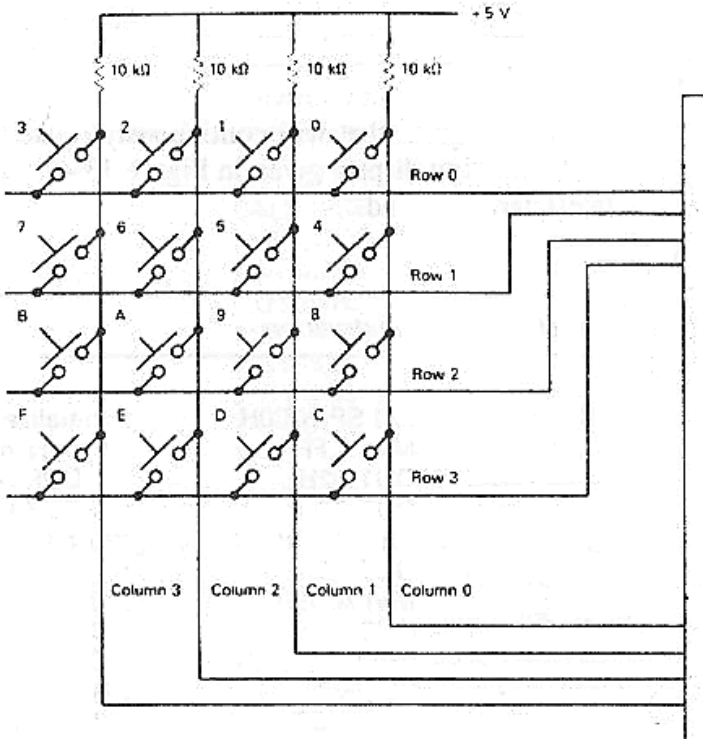
توضیح : قبل از این تابع در برنامه آرایه data از نوع unsigned char و سایز ۴ تعریف شده و در برنامه این آرایه مقدار مورد نظری که میخواهیم نمایش دهیم را به خود می گیرد . سپس با فراخوانی تابع display در این تابع ابتدا دیتای مربوط به سون سگمنت اول روی پورت قرار می گیرد سپس با ۱ کردن پایه com و صبر کردن به مدت ۵ میلی ثانیه ، دیتای مورد نظر روی سون سگمنت به نمایش در می آید . سپس سون سگمنت اول را خاموش کرده و دیتای دوم روی پورت قرار می گیرد و...

اتصال صفحه کلید به میکرو

صفحه کلید نیز یک وسیله ورودی پرکاربرد دیگر است که دارای مجموعه ای از کلیدها است که به صورت ماتریسی به هم بسته شده اند. همانطور که می دانیم اتصال صفحه کلید به میکروکنترلرها در بسیاری از موارد برای ما مهم و کاربردی است ، به عنوان مثال شما می خواهید یک ماشین حساب طراحی کنید یا یک قفل رمز دار و یا هر سیستم دیگری که نیاز است از کاربر اطلاعاتی توسط صفحه کلید دریافت شود. صفحه کلید ها انواع مختلفی دارند. از صفحه کلید تلفن گرفته تا صفحه کلید کامپیوتر ، به تعداد سطر ها و ستون های آن خروجی دارند . صفحه کلیدهای پرکاربرد موجود در بازار معمولا ۴ سطر و ۳ یا ۴ ستون دارند . در شکل زیر انواع صفحه کلیدها را در نرم افزار proteusکه در بازار نیز موجود است ، مشاهده می کنید.

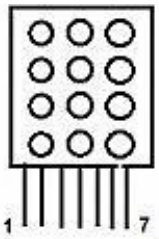


همانطور که در شکل زیر ساختار داخلی یک صفحه کلید ۴*۴ نشان داده شده است ، برای خواندن صفحه کلید ابتدا همه ستون ها را به صورت شکل زیر توسط مقاومت به تغذیه مثبت وصل کرده و از همه سطرها و ستون ها مستقیم به میکرو وصل می کنیم . سپس سطرها را به عنوان خروجی و ستون ها را به عنوان ورودی میکرو تنظیم می کنیم.



بنابراین در حالت عادی همه سطرها ۱ و ستون ها نیز چون توسط مقاومت به Vcc متصل هستند ۱ خوانده می شوند . در شکل فوق اگر Row0 توسط میکرو ۰ شود و بقیه ردیف ها همان ۱ باشند ، هر کدام از دکمه های ردیف اول که زده شود ، سیگنال ۰ مستقیم توسط ستون ها به پایه میکرو منتقل شده و توسط میکرو خوانده می شود . بدین ترتیب می توان با خواندن منطق ستون ها به دکمه زده شده پی برد.

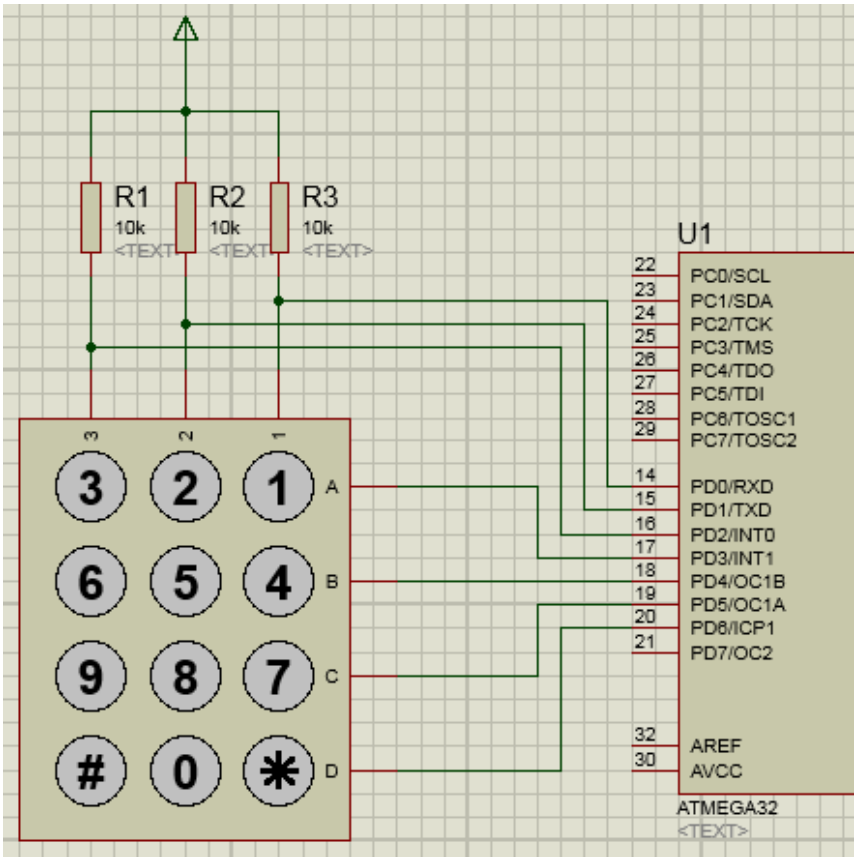
مدار داخلی صفحه کلیدهای دیگر نیز دقیقا به صورت فوق هستند . در شکل زیر نحوه چیدمان پایه های صفحه کلید ۴ در ۳ را مشاهده می کنید.



پایه های کی پد به ترتیب از چپ به راست

OUTPUT ARRANGEMENT	
OUTPUT PIN NO.	SYMBOL
1	COL 2
2	ROW 1
3	COL 1
4	ROW 4
5	COL 3
6	ROW 3
7	ROW 2

بنابراین برای اتصال هرگونه صفحه کلید کافی است همه ستون ها از یک طرف به مقاومت pullup و از طرف دیگر به میکرو ، و همه سطر ها نیز مستقیم به میکرو متصل گردد . در اینجا صفحه کلید ۴ در ۳ را به صورت زیر به پورت D میکرو متصل کردیم .



برای خواندن صفحه کلید ابتدا یک سطر را صفر و سطرهای بعدی را یک می کنیم . سپس کمی صبر میکنیم (۱ تا ۲ میلی ثانیه (و ستون ها را می خوانیم اگر همه ستون ها یک باشد یعنی در آن سطر کلیدی زده نشده است ، آن سطر را یک و سطر بعدی را صفر می کنیم و دوباره ستون ها را می خوانیم و اگر باز هم یک بود به سراغ سطر های بعدی می رویم تا اینکه تمام سطر ها خوانده شود و این کار را با سرعت بالا ادامه می دهیم . اما اگر کلیدی زده شود سطر و ستونی که کلید روی آن قرار دارد به هم وصل می شوند در نتیجه در هنگام خواندن سطرها متوجه صفر شدن می شویم . در برنامه نیز میتوان تابعی مانند تابع زیر را تعریف کرد که تمام این کارها را انجام دهد.

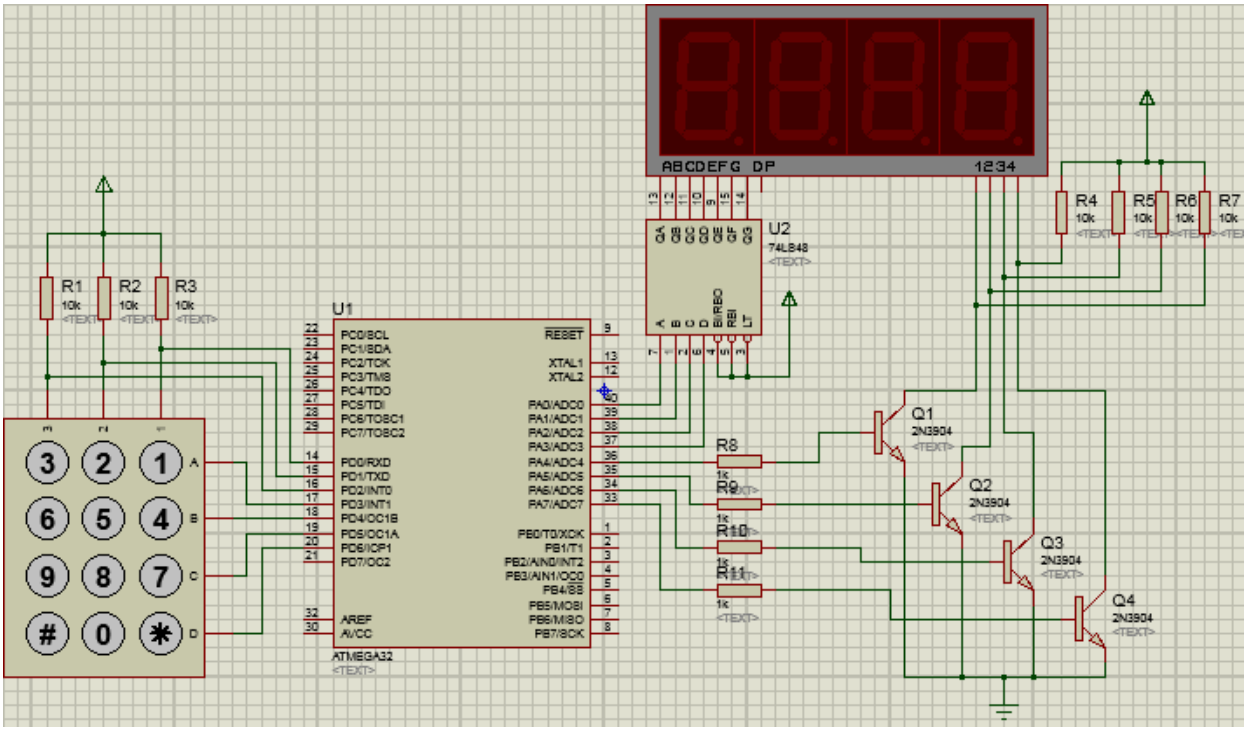
```
void keyboard(void)
{
 //---- ROW1 ----
 PORTD.3=0;
 delay_ms(2);
 if(PIND.0==0) key=1;
 if(PIND.1==0) key=2;
 if(PIND.2==0) key=3;
 PORTD.3=1;
 //---- ROW2 ----
 PORTD.4=0;
 delay_ms(2);
 if(PIND.0==0) key=4;
 if(PIND.1==0) key=5;
 if(PIND.2==0) key=6;
 PORTD.4=1;
```

```
//---- ROW3 ----
PORTD.5=0;
delay_ms(2);
if(PIND.0==0) key=7;
if(PIND.1==0) key=8;
if(PIND.2==0) key=9;
PORTD.5=1;
//---- ROW4 ----
PORTD.6=0;
delay_ms(2);
if(PIND.1==0) key=0;
PORTD.6=1;
}
```

توضیح : قبل از این تابع در برنامه متغیر key را از جنس unsigned char و مقدار اولیه دلخواه به جز اعداد ۰ تا ۹ تعریف کرده ایم . با فراخوانی تابع keyboard در برنامه تابع فوق اجرا می شود . این تابع برای همه سطرها ابتدا سطر مورد نظر را صفر کرده و بعد از تاخیر منطق ستون ها را می خواند در صورت ۰ بودن منطق هر یک یعنی کلید مورد نظر زده شده است و بنابراین مقدار متغیر key برابر با عدد کلید زده شده می شود . متغیر key نماینده کلید زده شده است که در برنامه از آن استفاده می شود.

مثال عملی شماره ۳ : یک صفحه کلید ۴ در ۳ و یک سون سگمنت مالتی پلکس شده mpx4 کاتد مشترک را به میکروکنترلر Atmega32 وصل نمایید . برنامه ای بنویسید که با فشار دادن صفحه کلید عدد مربوطه از صفحه کلید دریافت و روی سون سگمنت نمایش داده شود.

حل : بعد از طراحی سخت افزار و نرم افزار روی کاغذ به سراغ نرم افزار proteus رفته و مدار طراحی شده را به صورت شکل زیر رسم می کنیم.



سپس به سراغ نرم افزار CodeVision رفته و طبق مراحل گفته شده در فصل قبل پروژه جدید را می سازیم و فرکانس میکرو را روی 1Mhz داخلی قرار می دهیم . برنامه خواسته شده به صورت زیر است.

```
#include <mega32.h>

#include <delay.h>

unsigned char data[4]={0x0f,0x0f,0x0f,0x0f};

void display(void){
```

```
register unsigned char i;
unsigned char select[4]={0x80,0x40,0x20,0x10};
for(i=0;i<4;i++){
PORTA=data[i];
PORTA=PORTA | select[i];
delay_ms(5);
PORTA=0x0f;
}
}
unsigned char keyboard(void){
register unsigned char i,j;
unsigned char select[4]={0xF0,0x68,0x58,0x38};
for(i=0;i<4;i++){
PORTD=select[i];
delay_ms(2);
if((PIND & 0x07 )!= 0x07){
for(j=0;j<3;j++)
if((PIND & (1<<j))==0)
return i*3+j+1;
delay_ms(2);
}
PORTD=0xf8;
}
```

```
return 20;
}
void main (void){
unsigned char j,key=20;
unsigned int i=0,i1;
DDRD=0xf8;
PORTD=0xf8;
DDRA=0xff;
PORTA=0x0f;
while(1){
key=keyboard();
if(key==11)key=0;
if((key!=20) && (key<10)) {
i=i*10+key;
key=20;
i1=i;
   for(j=0;j<4;j++)
   {
      data[j]=i1%10;
      i1=i1/10;
   }
}
for(j=0;j<10;j++) display();
```

```
}

}
```

توضیح : آرایه data همان اعداد روی سگمنت هستند که چون در تمام برنامه به آن احتیاج داریم در ابتدای برنامه و به صورت global تعریف شده است . توابع display و keyboard دقیقا کار همان توابع گفته شده را انجام می دهند و حتی میتوان آن ها را با هم جایگزین کرد . تنها تفاوت این دو تابع با قبل این است که به صورت برنامه وار و حرفه ای تر نوشته شده اند تا خطوط کمتری اشغال و حجم حافظه برنامه را کاهش دهد . در حلقه while ابتدا عدد گرفته شده از صفحه کلید درون متغیر keyریخته می شود . تابع keyboard در صورتی که هیچ کلیدی زده نشده باشد مقدار ۲۰ را بر می گرداند و اگر مقدار key برابر ۱۱ باشد یعنی عدد ۰ را کاربر فشار داده و توسط if مقدار آن اصلاح می شود . در صورتی که کاربر کلیدی را زده باشد و کلیدی که زده شده یکی از کلید های ۰ تا ۹ باشد شرط if دوم برابر شده و داخل آن می شود . متغیر i همان عددی است که روی ۴ عدد سون سگمنت توسط متغیرهای data باید نمایش داده شود . بنابراین در حلقه for موجود عدد i به ۴ قسمت (یکان ، دهگان و ...) تبدیل می شود و هر کدام روی متغیر data مربوطه قرار می گیرد . و در نهایت تابع display برای ۱۰ بار اجرا می شود که اگر یک بار اجرا شود برنامه بسیار سریع عمل میکند.

سورس کدویژن و پروتئوس برنامه فوق را می توانید از لینک زیر دانلود و نحوه کار را مشاهده نمایید .

دانلود مثال عملی شماره ۳

پایان فصل هفتم

خسته نباشید ! یکی از مهمترین و پایه ای ترین فصول را با موفقیت پشت سر گذاشتید، سعی کنید مباحث و مثال های این فصل را اینقدر بخوانید تا مسلط شوید. سپس خودتان دست به کار شده و آن ها را از صفر در پروتئوس شبیه سازی کنید.

فصل هشتم : آموزش CodeWizard و واحدهای میکروکنترلرAtmega32

مقدمه

در فصل های گذشته به طور مقدماتی با نحوه کار با واحد I/O و رجیسترهای مربوط به تنظیم آنها آشنا شدیم و با استفاده از آن ها کلید ، صفحه کلید و نمایشگر سون سگمنت را راه اندازی کردیم . در این فصل نیز ابتدا به شرح مجدد واحد I/O و سپس معرفی ابزار Codewizard (جادوگر کد) خواهیم پرداخت و در ادامه به معرفی و بررسی واحد های دیگر نظیر تایمرها و کانترها ، ارتباطات سریال ، وقفه ها و ... می پردازیم . همانطور که در فصول چهارم و پنجم نیز گفته شد ، میکروکنترلر Atmega32 دارای واحد های جانبی زیر است :

- واحد ورودی/خروجی : در فصل چهارم معرفی و در فصل پنجم بررسی شد در فصل هفتم نیز پروژه های مربوطه با آن انجام شد.
- واحد کنترل کلاک : در فصل چهارم معرفی و به طور کامل در فصل ششم بررسی شد.
- واحد تایمر/کانتر : در فصل چهارم معرفی و در این فصل کاملا تشریح می شود.
- واحد Wathdog Timer : در فصل چهارم معرفی و در این فصل کاملا تشریح می شود.
- واحد کنترل وقفه : در فصل چهارم معرفی و در این فصل کاملا تشریح می شود.
- واحد ارتباطی JTAG : در فصل چهارم معرفی و **در آینده** بررسی خواهد شد.
- واحد مبدل ADC : در فصل چهارم معرفی و در این فصل کاملا تشریح می شود.
- واحد مقایسه کننده : در فصل چهارم معرفی و **در آینده** بررسی خواهد شد.
- واحد ارتباطی spi : در فصل چهارم معرفی و در این فصل کاملا تشریح می شود.
- واحد ارتباطی USART : در فصل چهارم معرفی و در این فصل کاملا تشریح می شود.
- واحد ارتباطی TWI : در فصل چهارم معرفی و **در آینده** بررسی خواهد شد.

تذکر : علاوه بر مباحث فوق بخش " مدیریت توان و مدهای کاهش توان" نیز یکی از مباحث بسیار مهم در میکروکنترلرهای AVR می باشد که **در آینده** بررسی خواهد شد.

معرفی و بررسی واحدهای میکروکنترلرAtmega32

هر یک از واحدهای یک میکروکنترلر از سه دیدگاه قابل بررسی است:

1. **معرفی** عملکرد ، ویژگی ها و وظایف واحد مورد نظر.
2. **تشریح** نحوه عملکرد مداری واحد مورد نظر.
3. بررسی **رجیسترهای** تنظیمات واحدمورد نظر .

هر یک از واحدهای گفته شده در Atmega32 به جز CPU ، رجیسترهایی دارد که تنظیمات واحد مورد نظر را مشخص می کند . برنامه نویس باید رجیسترهای مربوط به هر واحد را شناخته و آنها را بسته به پروژه مورد نظر به درستی مقدار دهی نماید . برای اینکه این مقداردهی راحت تر ، سریعتر و با دقت بیشتر صورت گیرد از ابزار CodeWizard استفاده می شود در این فصل ابتدا با رجیسترهای واحد I/O بیشتر آشنا خواهیم شد و سپس ضمن آموزش CodeWizard بقیه واحد ها را نیز به ترتیب در این بخش و بخش های بعدی آموزش خواهیم داد.

معرفی و تشریح واحد ورودی/خروجی(پورت های I/O)

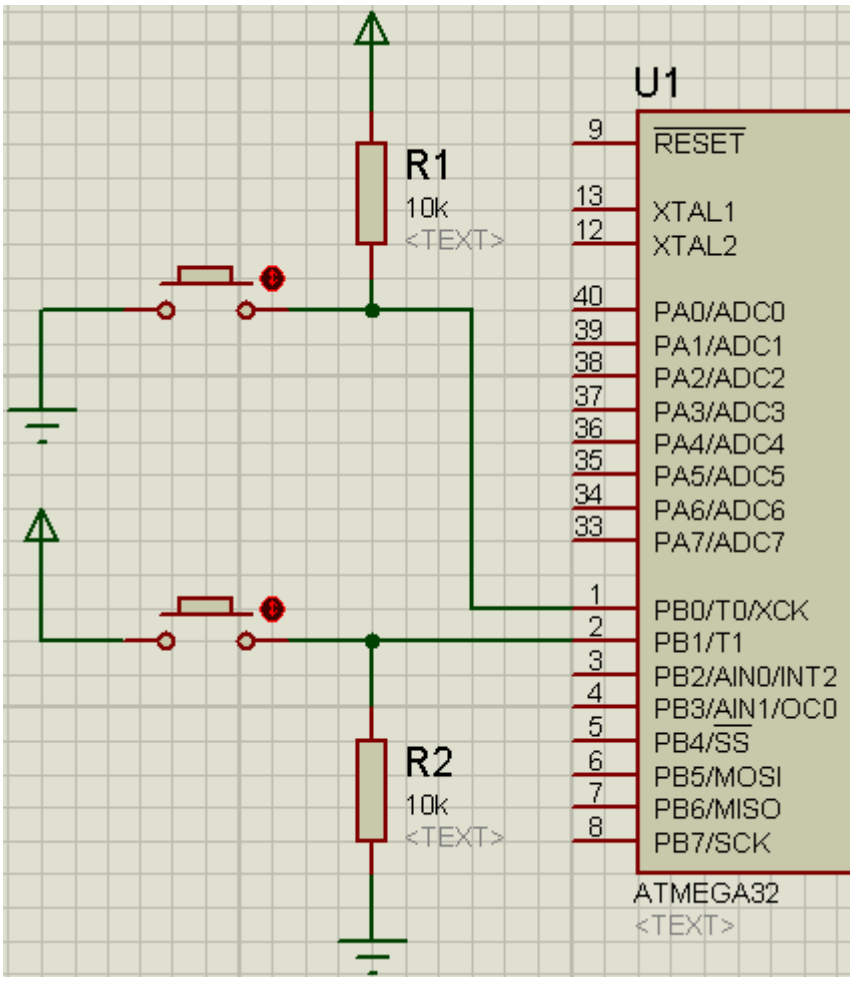
پورت های ورودی/خروجی یکی از مهم ترین واحدهای هر میکروکنترلر می باشد که به منظور استفاده عمومی و کاربردهای همه منظوره طراحی شده است. به طور کلی نکات زیر در مورد واحد I/O در میکروکنترلرهای AVR وجود دارد:

1. هر یک از پایه های میکروکنترلر ممکن است چندین عملکرد از جمله پورت ورودی/خروجی باشد که به صورت همزمان نمی توان از آنها استفاده نمود. بنابراین در صورت استفاده از واحد I/O در یک پایه ، فقط به عنوان پورت ورودی یا خروجی همه منظوره (General Purpose I/O) استفاده می شود.
2. در صورت استفاده از یک پایه به عنوان **خروجی** ، پایه مورد نظر می تواند دو حالت (منطق ۱ یا منطق ۰) داشته باشد. در صورتی که آن پایه منطق ۱ داشته باشد ولتاژ آن پایه **حداکثر** Vcc می گردد و جریانی از داخل میکرو به بیرون کشیده می شود که به آن جریان Source گویند. در صورتی که پایه منطق ۰ داشته باشد ، ولتاژ آن پایه **حداقل** ۰ ولت می گرد و جریانی از بیرون به آن وارد می شود که به آن جریان sink گویند.

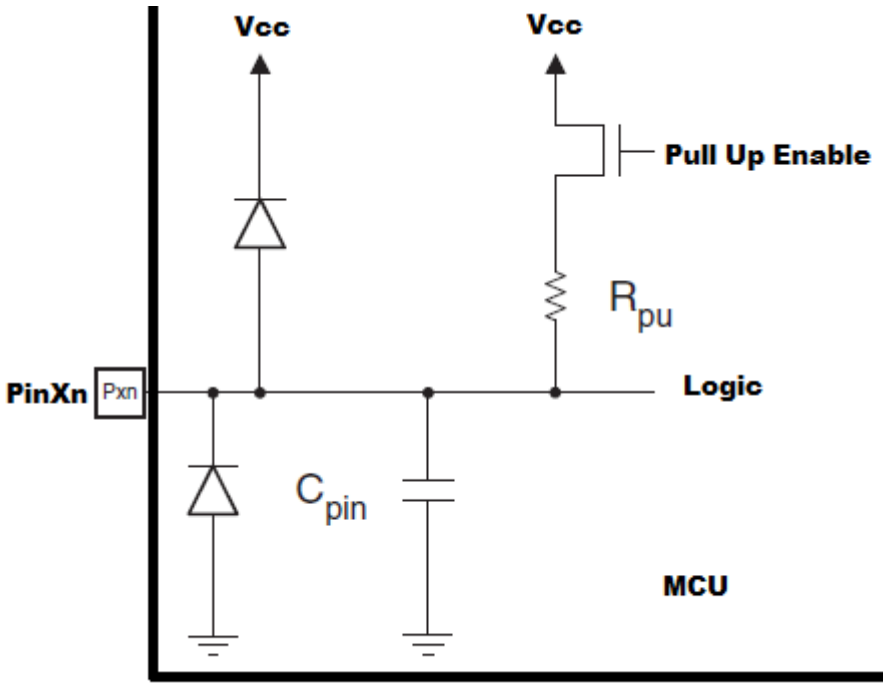
3. حداکثر جریان Source و Sink در میکروکنترلر Atmega32 برای وقتی که Vcc=5v و دما ۲۵ درجه باشد ، به مقدار ۶۰ میلی آمپر می رسد . با افزایش جریان Source ، ولتاژ منطق ۱ پایه شروع با کاهش از مفدار Vcc می کند . هر چه جریان دهی پایه بیشتر باشد افت ولتاژ پایه از مقدار Vcc نیز بیشتر می شود . همانطور که در شکل صفحه ۳۰۸ دیتاشیت قسمت Typical Characteristics نیز برای Vcc=5v و دما ۲۵ درجه مشاهده می شود ، با افزایش جریان دهی با کاهش ولتاژ از ۵ ولت تا ۳ ولت مواجه هستیم. برای جریان Source هم به همین صورت اگر جریانی که به پایه آی سی وارد می شود افزایش یابد ولتاژ آن از حالت ایده آل (۰ ولت) افزایش پیدا کرده به طوری که برای Vcc=5v و دمای ۲۵ درجه به ۲ ولت نیز می رسد.

4. در صورت استفاده از یک پایه به عنوان **ورودی** ، پایه مورد نظر می تواند یکی از سه حالت زیر را داشته باشد:

- حالت اتصال پایه به منطق ولتاژی ۰ یا ۱ از بیرون : مانند شکل زیر



- حالت دارای مقاومت Pull Up داخلی : در این حالت یک مقاومت Rpu از داخل میکروکنترلر تنها به صورت پول آپ میتواند وصل شود. شکل زیر این مقاومت را به همراه دیگر اجزای داخلی هر پایه نشان می دهد . هر پایه واحد I/O دارای دیودهای هرزگرد ، خازن و مقاومت پول آپ می باشد.



نکته : مقدار مقاومت پول آپ به طور تقریبی برابر ۵ کیلو اهم است.

- حالت بدون اتصال (مدار باز یا High Z) : پایه هایی که به هیچ جایی متصل نیستند به صورت بدون اتصال ، مدار باز یا امپدانس بالا (High Z) هستند (در حالت پیش فرض کلیه پایه ها ورودی و بدون اتصال هستند).

نکته Tri-State : به معنای **سه حالته** می باشد . یعنی پایه در سه حالت بدون اتصال ، اتصال به ۰ و اتصال به ۱ میتواند قرار گیرد.

رجیسترهای واحد I/Oدر Atmega32

در ATmega32 چهار پورت ورودی/خروجی وجود دارد که برای هر یک پین هایی خروجی درنظر گرفته شده است . هر یک از این ۴ پورت ورودی/خروجی دارای سه رجیستر DDR ، PINو PORT هستند که در نتیجه مجموعا ۱۲ رجیستر ۸ بیتی برای واحد ورودی/خروجی وجود دارد که همگی در حقیقت درون SRAM می باشند که با مقدار دهی آنها در برنامه این واحد کنترل می شود .همانطور که در فصول قبل توضیح داده شد ، رجیستر DDR برای تعیین جهت ورودی یا خروجی بودن ، رجیستر PORT برای تعیین مقاومت PullUp در حالت ورودی بودن پایه و تعیین منطق ۰ یا ۱ در حالت خروجی بودن پایه کاربرد دارد و رجیستر PIN نیز برای خواندن منطق اعمال شده از بیرون در حالت ورودی می باشد .شکل زیر حالت های مختلف پایه ها را بسته به بیت n ام رجیستر DDR و PORTو نیز بیت PUD در رجیستر SFIOR نشان می دهد . لازم به توضیح است که رجیستر SFIORیک رجیستر ۸ بیتی برای تنظیمات خاص پایه ها می باشد که بیت دوم آن PUDیا PullUp Disable مربوط به غیر فعال کردن کلیه مقاومت های PullUp می باشد

.

DDxn	PORTxn	PUD (in SFIOR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

نحوه فعالسازی مقاومت پول آپ

همانطور که در شکل فوق نیز مشاهده می کنید ، در حالتی که رجیستر DDRX.n صفر باشد یعنی پایه ورودی بوده و در صورتی که رجیستر PORTX.n را یک کنیم ، مقاومت پول آپ داخلی فعالسازی می شود .بنابراین رجیستر PORTX.nدر حالت خروجی بودن پایه ، منطق ۰ یا ۱ پایه را مشخص می کند و در حالت ورودی بودن پایه فعال

بودن یا نبودن مقاومت پول آپ را مشخص می کند . بنابراین برای فعال شدن مقاومت پول آپ داخلی باید سه شرط زیر برقرار باشد:

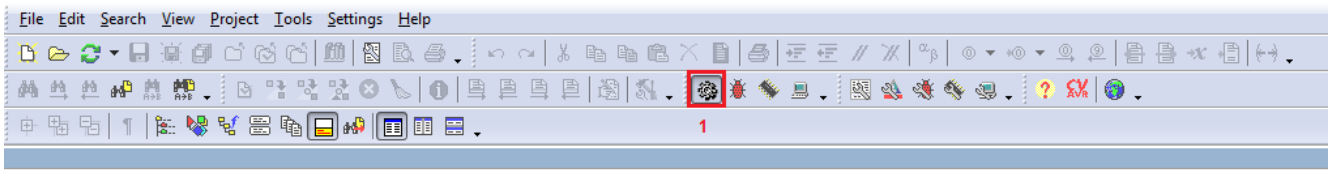
1. بیت PUD یا Pull Up Disable غیر فعال یا ۰ باشد که همیشه برقرار است مگر زمانی که میکروکنترلر ریست باشد یا هنگ کرده باشد.
2. رجیستر DDRX.n مربوطه ۰ باشد یعنی پایه ورودی باشد . پس در حالت خروجی مقاومت پول آپ نمیتواند فعال شود.
3. رجیستر PORTX.n مربوطه ۱ باشد . در صورت ۰ بودن نیز مقاومت پول آپ غیر فعال است.

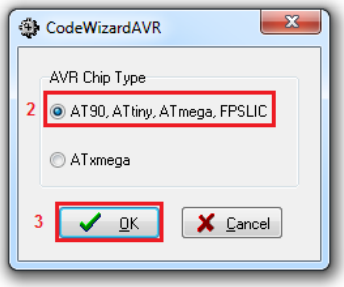
CodeWizardچیست ؟

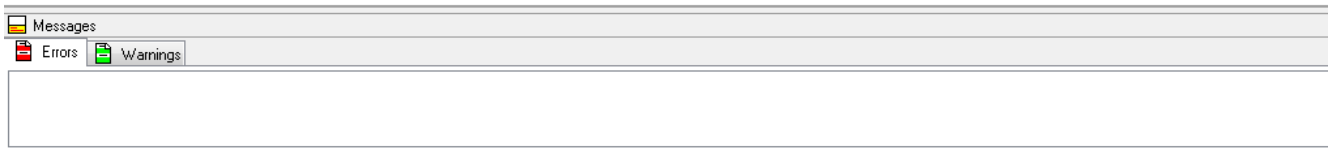
در حقیقت CodeWizard ابزار کاملی برای برنامه نویسی کلیه میکروکنترلرهای AVR است که به همراه نرم افزار CodeVision ارائه شده و بخشی از این نرم افزار می باشد. این ابزار که به جادوگر کد معروف است به شما این امکان را می دهد که با انتخاب کردن مدل میکروکنترلر ، فرکانس کاری و تنظیم و پیکره بندی کلیه واحدهای میکروکنترلر (نظیر پورت ها ، تایمر/کانترها و ...) از میان گزینه های آماده موجود در ابزار ، کدهای مربوط به **تنظیمات اولیه رجیسترها** به زبان C را به طور اتوماتیک تولید کند .

شروع کار با ابزارCodeWizard

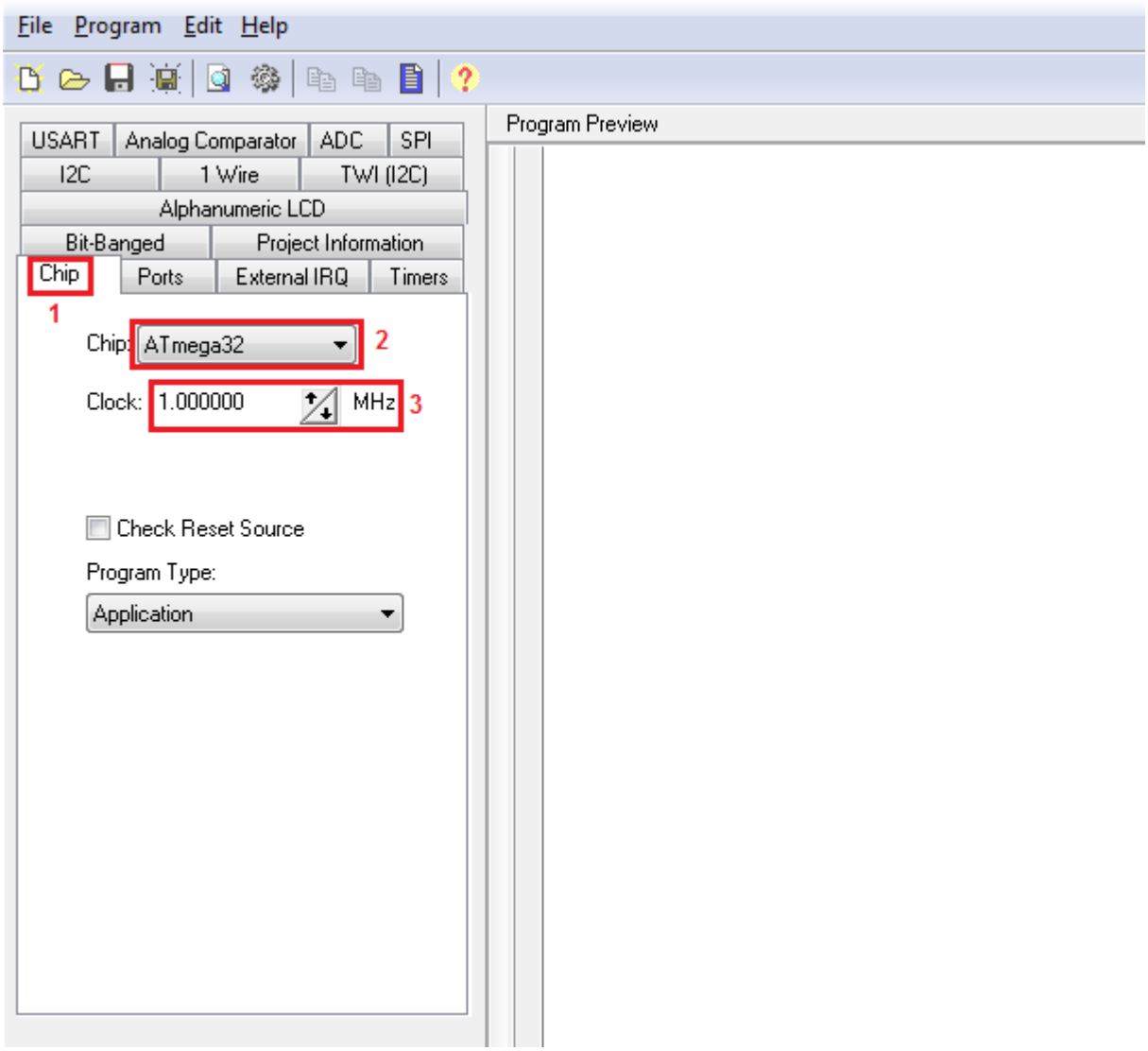
1. ابتدا به صفحه اصلی نرم افزار CodeVision AVR رفته و اگر پروژه ای باز است از منوی File و گزینه Close All آن را می بندیم.
2. برای راه اندازی کدویزارد در صفحه اصلی نرم افزار CodeVision از منوی tools ، گزینه CodeWizard AVR را انتخاب کنید. همچنین می توانید به جای آن از منوی ابزار بالای نرم افزار گزینه چرخ دنده را کلیک کنید. در پنجره باز شده میخواهد که نوع چیپ را از نظر سری ساخت مشخص کنید. چون چیپ مورد استفاده ما atmega32 است ، گزینه اول atmega را انتخاب کرده و ok کنید . اکنون پنجره codeWizardAVRجلوی شماست.



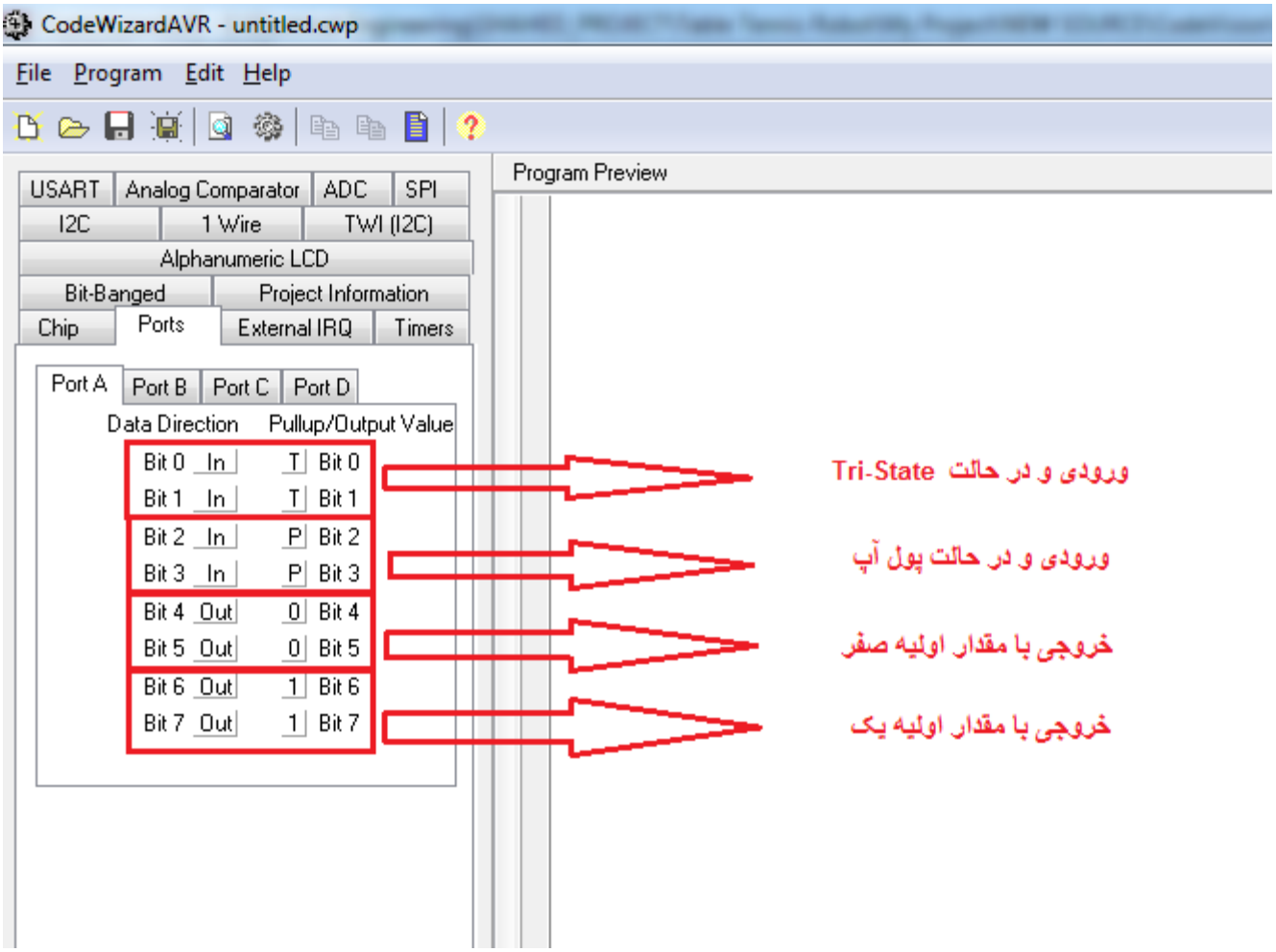




3. در این قسمت اولین کاری که باید انجام دهید تنظیم نوع چیپ و فرکانس کاری میکروکنترلر است . دیگر گزینه ها را کاری نداریم . فرکانس کاری میکروکنترلر باید همان فرکانسی انتخاب شود که در قسمت Fuse Bits (فیوز بیت ها) تنظیم شده است. در صورتی که میکروکنترلر Atmega32را تازه خریداری کرده اید، فرکانس پیش فرض آن1Mhz می باشد.



4. در مرحله بعد وارد سربرگ Port می شویم . در این مرحله باید طبق سخت افزار طراحی شده پورت ها را تنظیم کرد. همانطور که مشاهده می شود ، پورت ها را میتوان ورودی in یا خروجی out نمود و در صورت ورودی بودن با تغییر T یعنی Tri-State به P میتوان مقاومت PullUp داخلی میکرو را نیز فعال کرد و در صورت خروجی بودن نیز میتوان با تغییر ۰ و ۱ مقدار اولیه منطق پورت خروجی را تعیین نمود . سربرگ های دیگر هر کدام مختص واحدهای دیگر میکروکنترلر است که در صورت لزوم در این مرحله باید تنظیم شود. در شکل زیر مثالی از نحوه این تغییرات را مشاهده می کنید.



5. بعد از پایان تنظیمات از منوی File گزینه Generate, save and Exit را کلیک کنید. اکنون سه بار باید فایل های مربوط به پروژه را با نام ترجیحا یکسان ذخیره کنید . در پنجره اول اسمی برای فایل با پسوند C. وارد می کنیم و ذخیره می کنیم و همین طور در پنجره های بعدی برای project و codeWizardAVR نام وارد کرده و ذخیره کنید. مشاهده می کنید که کد ویزارد بخش اولیه برنامه شما را ایجاد کرده است . اکنون شما می توانید شروع به ادامه برنامه نویسی با تغییر یا اضافه کردن کدهای ساخته شده نمایید.

راه اندازی LCD های کارکتری با کدویزارد

صفحه نمایش کاراکتری یکی از پرکاربرد ترین وسایل خروجی است که به میکرو وصل میشود و میتوان کاراکترهای قابل چاپ را روی آن نمایش داد. مشخصه اصلی LCD های کاراکتری تعداد سطر و ستون آنها است برای مثال LCDهای ۲×۱۶ دارای ۲ سطر و ۱۶ ستون می باشند و در مجموع ۳۲ کاراکتر را میتوان با آنها نشان داد.

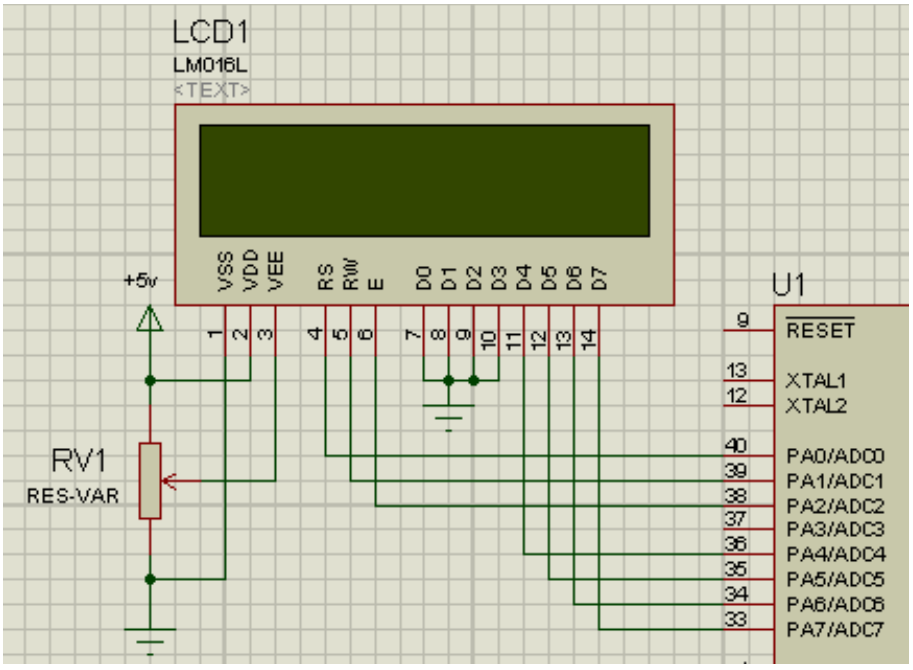
در نرم افزار پروتئوس نیز انواع LCD های کاراکتری و گرافیکی در کتابخانه optoelectronics موجود است که میتوان از آنها برای شبیه سازی این قطعه استفاده کرد . در این آموزش ما از LM016L که یک LCD کاراکتری با ۲ سطر و ۱۶ ستون است ، بیشترین استفاده را خواهیم کرد. بنابراین با تایپ کردن LM016L میتوان این قطعه در پروتئوس را به پروژه اضافه کرد.

ال سی دی های کارکتری اغلب دارای ۱۴ پایه هستند که ۸ پایه برای انتقال اطلاعات ۳ پایه برای کنترل نرم افزاری یک پایه برای کنترل سخت افزاری و دو پایه تغذیه می باشد در جدول زیر اطلاعات پایه ها را مشاهده می کنید:

پایه	نام	توضیحات
۱	vss	پایه (-) تغذیه
۲	vcc	پایه(+) تغذیه
۳	VEE	کنترل درخشندگی صفحه
۴	RS	انتخاب دستور داده
۵	R/w	فعال ساز خواندن یا نوشتن
۶	E	فعال ساز
۷	DB0	دیتا
۸	DB1	دیتا
۹	DB2	دیتا

۱۰	DB3	دیتا
۱۱	DB4	دیتا
۱۲	DB5	دیتا
۱۳	DB6	دیتا
۱۴	DB7	دیتا
۱۵	A	پایه + روشنایی صفحه(+بک لایت)
۱۶	k	پایه – روشنایی صفحه(- بک لایت)

LCDها را میتوان با ۸ خط دیتا یا ۴ خط دیتا راه اندازی کرد اما نرم افزار CodeVision تنها از ۴ خط دیتا پشتیبانی می کند که در آن تنها پایه های RS ، R/W، Eو D4 تا D7 به پورت دلخواه از میکرو متصل می شود . تنها تفاوت راه اندازی ۴ خط دیتا در این است که داده های ۸ بیتی LCD به جای یکبار ، دو بار از طریق ۴ بیت ارسال می شوند . مزیت این راه اندازی نیز در این است که اتصال LCD به میکروکنترلر تنها یک پورت از میکرو را اشغال می کند . پایه های LCD کاراکتری ۲ در ۱۶ و نحوه وصل شدن به میکروکنترلر را در شکل زیر مشاهده می کنید.



تذکر مهم : در نرم افزار Proteus و به هنگام شبیه سازی میتوان پایه های ۱ ، ۲ ، ۳ ، ۷ ، ۸ ، ۹ و ۱۰ را بدون اتصال گذاشت اما در عمل و پیاده سازی تمامی پایه ها باید به محل مربوطه مطابق شکل فوق متصل باشد.

تنظیمات CodeWizard برای راه اندازی LCD کاراکتری

در ابزار کدویزارد و بعد از تنظیمات چیپ، فرکانس و پورت ها در آن به سربرگ Alphanumeric LCD رفته و ابتدا تیک Enable را بزنید و سپس در قسمت Character/line باید تعداد ستون های LCD مورد استفاده را وارد نمایید و در قسمت بعدی باید نام پایه های میکرو که به LCD متصل می شود را مشخص کنید. با این کار پس از تولید کد توسط ابزار کد ویزارد مشاهده می کنید که در برنامه هدر فایلی با نام alcd.h و تابع راه انداز LCD به صورت lcd_init(16) به برنامه اضافه می شود که عدد ۱۶ در آن نشان دهنده تعداد ستون های LCD است . با اضافه شدن این هدر فایل از توابع زیر برای کار با LCD در برنامه دلخواه می توان استفاده کرد.

توابع کار با LCD کاراکتری:

1. پاک کردن تمام LCD :

```
lcd_clear();
```

این تابع lcd را پاک کرده و مکان نما را در سطر و ستون صفر قرار می دهد.

2. رفتن به ستون x و سطر y ام :

```
lcd_gotoxy(x , y );
```

تابع فوق مکان نمای ال سی دی را در سطر x و ستون y قرار می دهد و باید به جای x و y عدد سطر و ستون مورد نظر جا گذاری شود.

3. چاپ یک کاراکتر:

```
lcd_putchar(‘ کاراکتر ‘);
```

4. چاپ یک رشته:

```
lcd_putsf(“ رشته “);
```

5. چاپ یک متغیر رشته ای:

lcd_puts(نام متغیر رشته ای);

نکته : ورودی تابع lcd_putsfیک رشته ثابت مانند ”IRAN“ است اما ورودی تابع lcd_puts یک متغیر رشته‌ای از نوع آرایه ای می تواند باشد . برای مقدار دهی به یک متغیر رشته ای میتوان از تابع sprintf استفاده کرد. برای استفاده از این تابع که در کتابخانه stdio.h می باشد ، ابتدا هدر فایل را به برنامه اضافه کرده و سپس به صورت زیر عمل می کنیم :

```
…
#include <stdio.h>
…
unsigned char c[16];
int i;
…
sprintf(c,”temp=%d”,i);
lcd_puts(c);
…
```

توضیح : با اضافه کردن stdio.h میتوان از تابع sprintf در برنامه استفاده کرد. در مثال فوق می خواهیم دمای اتاق را در رشته ای به صورت temp=%d که در آن %d یک عدد متغیر است و در جایی از برنامه به آن مقدار دهی کردیم ، بریزیم و سپس روی lcd نمایش دهیم . پس در آرگومان تابع sprintf ابتدا نام متغیر رشته ای را نوشته سپس رشته مورد نظر را به صورت جدول زیر و در آخر نام متغیر را می نویسیم.

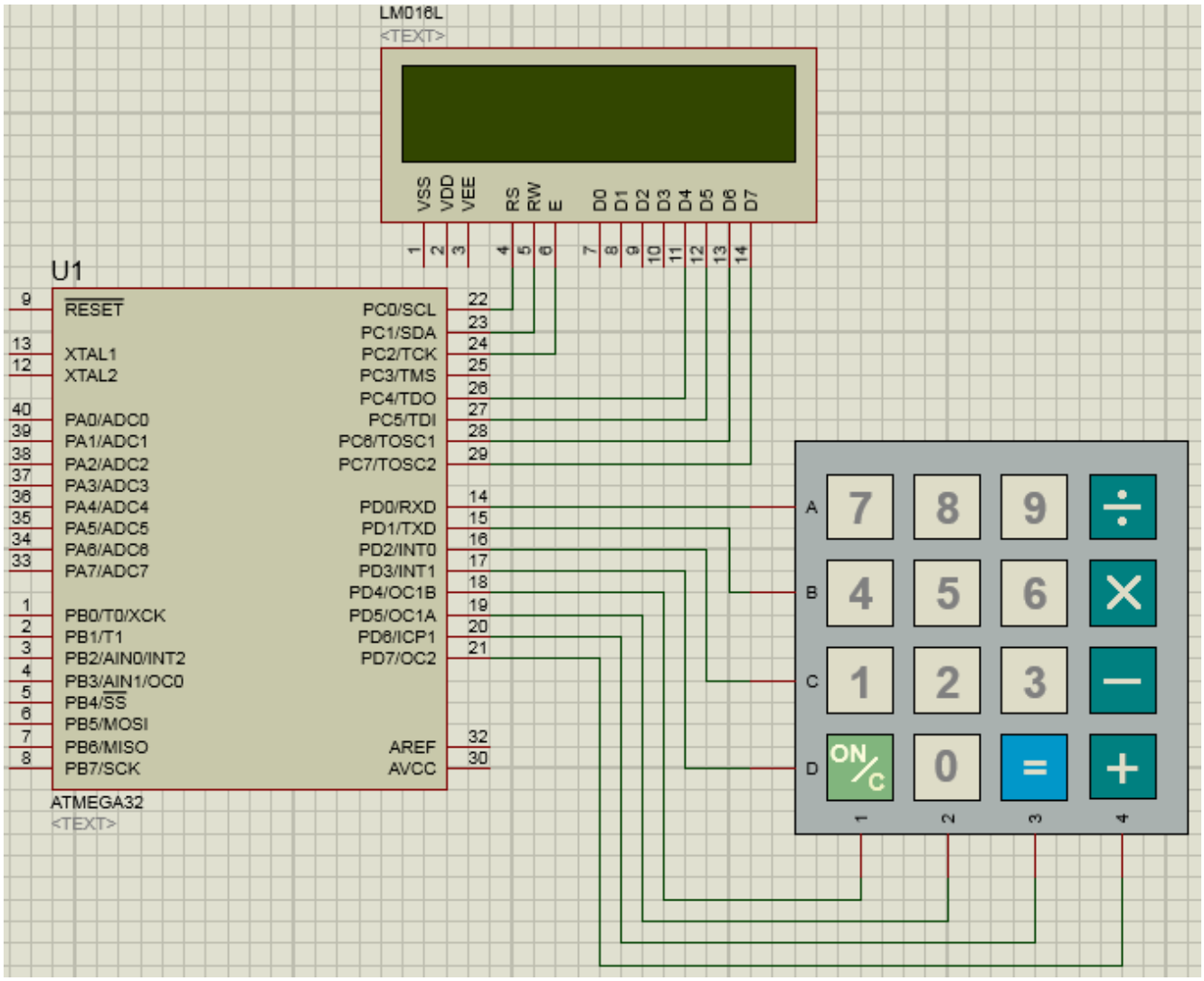
فرمت متغیرهای کاراکتری ارسالی:

کاراکتر	نوع اطلاعات ارسالی
%c	یک تک کاراکتر
%d	عدد صحیح علامت دار در مبنای ۱۰
%i	عدد صحیح علامت دار در مبنای ۱۰
%e	نمایش عدد ممیز شناور به صورت علمی
%E	نمایش عدد ممیز شناور به صورت علمی
%f	عدد اعشاری
%s	عبارت رشته ای واقع در حافظه SRAM
%u	عدد صحیح بدون علامت در مبنای ۱۰
%X	به فرم هگزا دسیمال با حروف بزرگ
%x	به فرم هگزا دسیمال با حروف کوچک
%p	عبارت رشته ای واقع در حافظه FLASH
%%	نمایش علامت ٪

مثال عملی شماره ۴ : با استفاده از اتصال یک LCD کاراکتری ۱۶ در ۲ و یک صفحه کلید ۴ در ۴ به میکروکنترلر Atmega32 ، برنامه ای بنویسید که با فشار دادن هر کلید کاراکتر مربوطه روی نمایشگر چاپ شود . برنامه را توسط کدویزارد نوشته و راه اندازی صفحه کلید را با استفاده از مقاومت پول آپ داخلی انجام دهید.

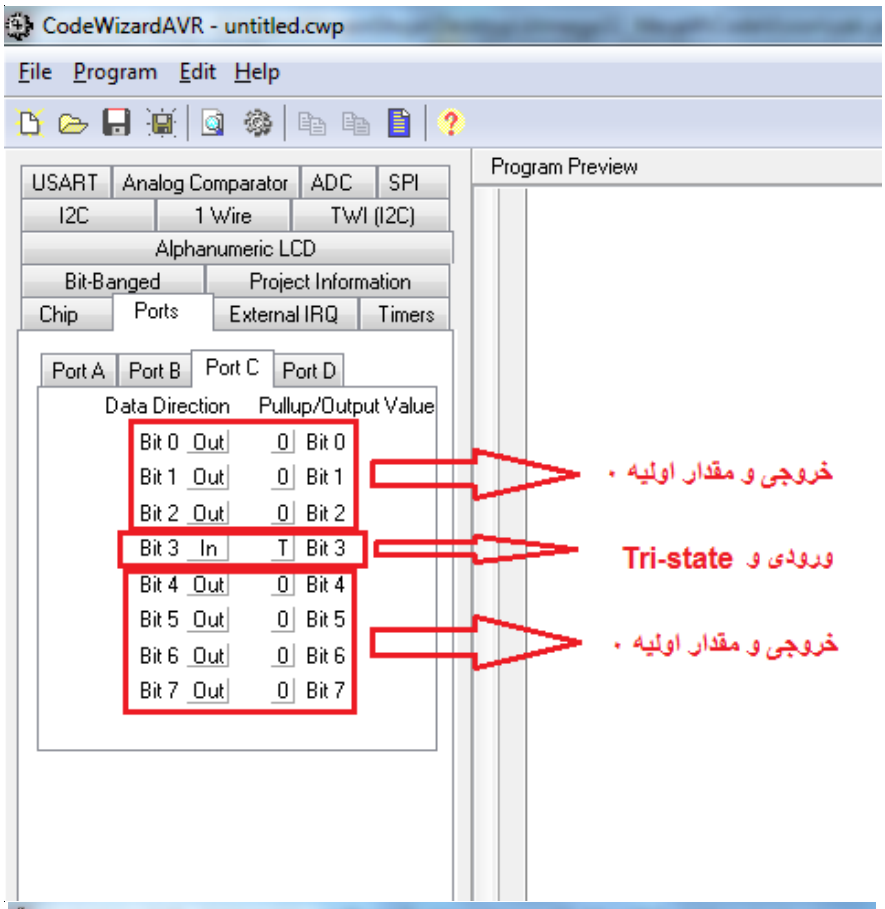
حل:

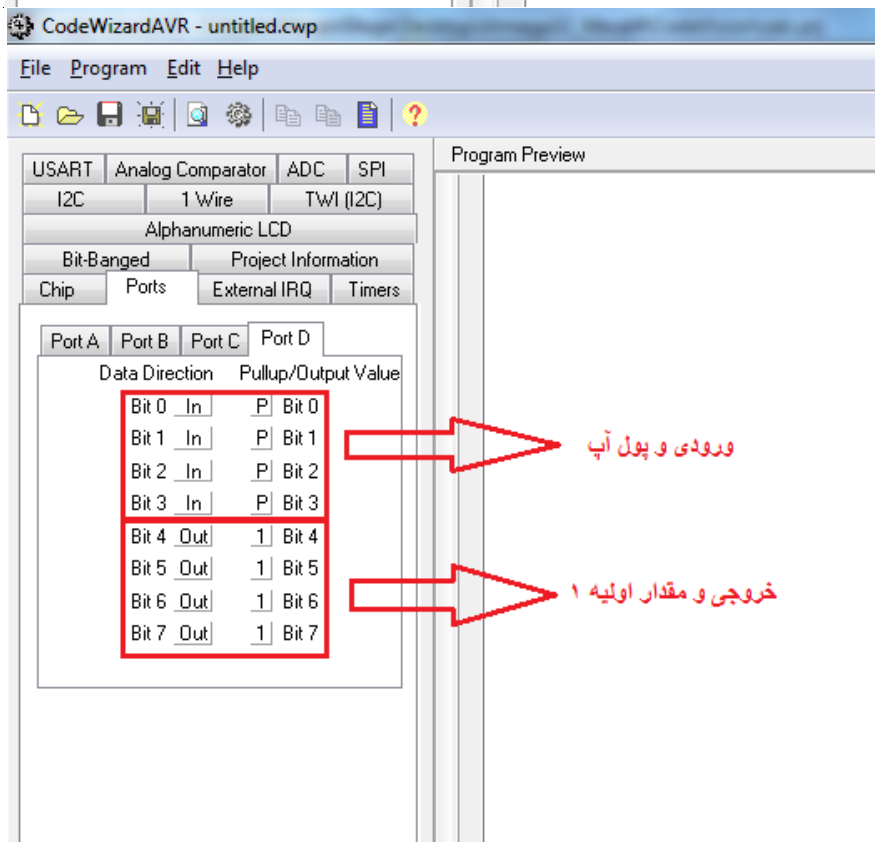
مرحله اول : طراحی سخت افزار در پروتئوس



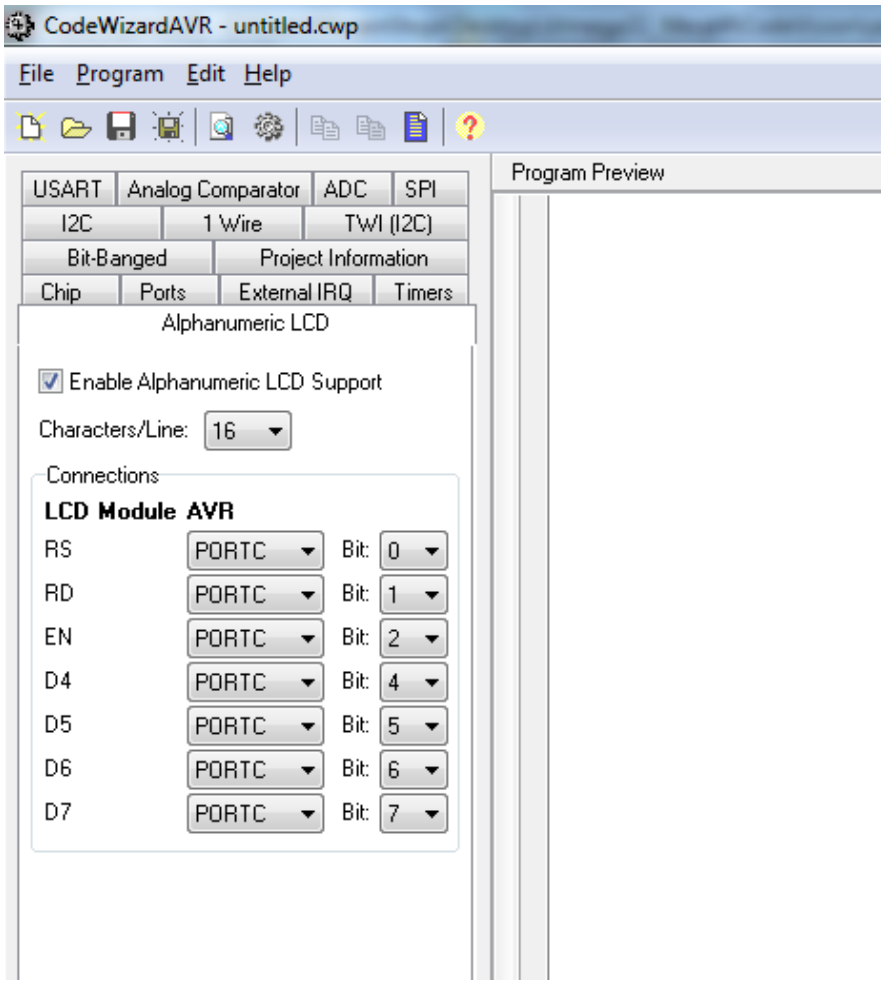
مرحله دوم : انجام تنظیمات در کدویزارد

در این مرحله بعد از مشخص کردن نوع چیپ و فرکانس کاری میکرو ، در سربرگ port تنظیمات پورت C و D را طبق سخت افزار طراحی شده و به صورت زیر انجام می دهیم.


CodeWizardAVR - untitled.cwp
File Program Edit Help
USART Analog Comparator ADC SPI
I2C 1 Wire TWI (I2C)
Alphanumeric LCD
Bit-Banged Project Information
Chip Ports External IRQ Timers
Port A Port B Port C Port D
Data Direction Pullup/Output Value
Bit 0 Out 0 Bit 0
Bit 1 Out 0 Bit 1
Bit 2 Out 0 Bit 2
Bit 3 In T Bit 3
Bit 4 Out 0 Bit 4
Bit 5 Out 0 Bit 5
Bit 6 Out 0 Bit 6
Bit 7 Out 0 Bit 7
Program Preview
خروجی و مقدار اولیه ۰
ورودی و Tri-state
خروجی و مقدار اولیه ۰


CodeWizardAVR - untitled.cwp
File Program Edit Help
USART Analog Comparator ADC SPI
I2C 1 Wire TWI (I2C)
Alphanumeric LCD
Bit-Banged Project Information
Chip Ports External IRQ Timers
Port A Port B Port C Port D
Data Direction Pullup/Output Value
Bit 0 In P Bit 0
Bit 1 In P Bit 1
Bit 2 In P Bit 2
Bit 3 In P Bit 3
Bit 4 Out 1 Bit 4
Bit 5 Out 1 Bit 5
Bit 6 Out 1 Bit 6
Bit 7 Out 1 Bit 7
Program Preview
ورودی و پول آپ
خروجی و مقدار اولیه ۱

سپس به سربرگ Alphanumeric LCD رفته و تنظیمات آن را نیز به صورت زیر انجام می دهیم.



تنظیمات کدویزارد پایان می یابد . بعد از ذخیره و خروج از کدویزارد (Save,Generate & Exit) کد مورد نیاز تولید شده است . در این مرحله بهتر است با حذف کامنت ها و قسمت های اضافی برنامه حجم کد برنامه را کاهش دهیم.

مرحله سوم : نوشتن برنامه

در این مرحله تابع خواندن از صفحه کلید ۴ در ۴ را اضافه کرده و برنامه را به صورت زیر کامل می کنیم:

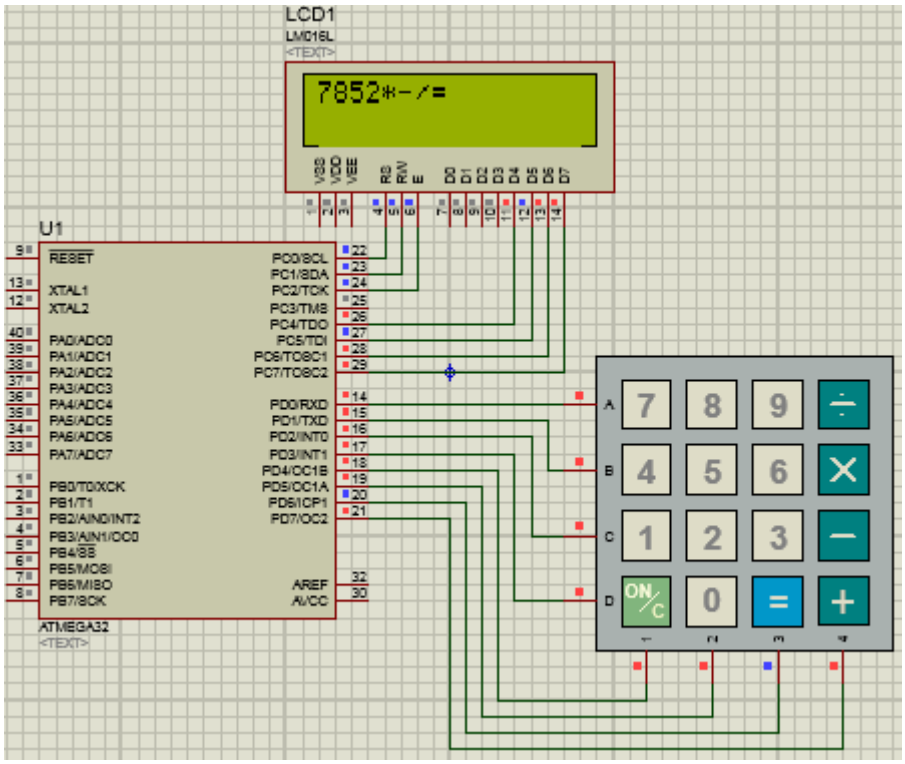
```
#include <mega32.h>
#include <delay.h>
#include <stdio.h>
#include <alcd.h>
unsigned char c[16];
unsigned char key=20;
void keyboard(void)
{
 key=20; //default value
 //---- ROW1 ----
 PORTD.4=0;
 delay_ms(2);
 if(PIND.0==0) key=7;
 if(PIND.1==0) key=4;
 if(PIND.2==0) key=1;
 if(PIND.3==0) lcd_clear();
 PORTD.4=1;
 //---- ROW2 ----
 PORTD.5=0;
```

```
delay_ms(2);
if(PIND.0==0) key=8;
if(PIND.1==0) key=5;
if(PIND.2==0) key=2;
if(PIND.3==0) key=0;
PORTD.5=1;
//---- ROW3 ----
PORTD.6=0;
delay_ms(2);
if(PIND.0==0) key=9;
if(PIND.1==0) key=6;
if(PIND.2==0) key=3;
if(PIND.3==0) { lcd_putchar('='); delay_ms(300); }
PORTD.6=1;
//---- ROW4 ----
PORTD.7=0;
delay_ms(2);
if(PIND.0==0) { lcd_putchar('/'); delay_ms(300); }
if(PIND.1==0) { lcd_putchar('*'); delay_ms(300); }
if(PIND.2==0) { lcd_putchar('-'); delay_ms(300); }
if(PIND.3==0) { lcd_putchar('+'); delay_ms(300); }
PORTD.7=1;
```

```
 }
void main(void)
{
PORTA=0x00;
DDRA=0x00;
PORTB=0x00;
DDRB=0x00;
PORTC=0x00;
DDRC=0xF7;
PORTD=0xFF;
DDRD=0xF0;
lcd_init(16);
lcd_clear();
while (1){
   keyboard();
   if(key!=20){
      sprintf(c,"%d",key);
      lcd_puts(c);
      delay_ms(300);
      }
     }
}
```

مرحله چهارم : شبیه سازی

بعد از اضافه کردن فایل hex در نرم افزار پروتئوس برنامه را run می کنیم.



پایان مثال عملی شماره 4

سورس کدویژن و پروتئوس برنامه فوق را میتوانید از لینک زیر دانلود و نحوه کار را مشاهده نمایید.

دانلود مثال عملی شماره ۴

معرفی و تشریح واحد وقفه های خارجی

اساسا وقفه زمانی مورد نظر می باشد که دستگاههای جانبی متعددی کنار میکرو وجود داشته باشد . برای سرویس دهی به وسایل جانبی که در اطراف میکروکنترلر می باشد ، دو روش وجود دارد . روش اول سرکشی منظم به دستگاههای جانبی موجود و روش دوم مراجعه به آن دستگاه فقط در زمان بروز وقفه می باشد.

در حالت سرکشی میکرو طوری برنامه نویسی شده است که دائما واحد جانبی مورد نظر را بررسی کند و با آن ارتباط برقرار می کند . برای مثال وقتی که یک صفحه کلید به میکرو متصل کردیم ، میکرو توسط تابعی به نام keyboard که در حلقه نامتناهی نوشته شده بود ، دائما تمام کلیدهای صفحه کلید را بررسی می کرد تا اینکه کاربر کلیدی را وارد کند و...

اما در هنگام بروز وقفه ، پردازنده کار فعلی خود را رها کرده و به اجرای وقفه مورد نظر می پردازد . علت بوجود آمدن واحد کنترل وقفه این است که باعث می شود پردازنده کمتر درگیر شود. در حالتی وقفه وجود نداشته باشد ، پردازنده مجبور است طی فواصل زمانی مشخصی چندین بار به واحد مورد نظر سرکشی کرده و بررسی کند که دیتای خواسته شده از آن واحد آماده است یا خیر که اغلب آماده نبوده و وقت پردازنده تلف می شود . اما در حالتی که واحد وقفه فعال است ، پردازنده آزاد است تا زمانی که دیتای واحد مورد نظر آماده شود. سپس واحد وقفه یک سیگنال وقفه به پردازنده مبنی بر آماده بودن دیتا ارسال می کند تا پردازنده متوجه شده و دیتا را پردازش کند.

انواع منابع وقفه در میکروکنترلرهایAVR

منبع وقفه در میکروکنترهای AVR دو نوع می باشد : ۱-داخلی ۲-خارجی

۱-وقفه داخلی : تقریبا تمام امکانات داخلی میکرو دارای وقفه بوده مانند تایمر/کانترها و پروتکل های ارتباطی و مقایسه کننده ها و مبدل آنالوگ به دیجیتال . یعنی مثلا وقتی که میکرو مقدار آنالوگ را ابه دیجیتال تبدیل کرده و کارش تمام شد ، وقفه را فعال می کند و برنامه ای را که در تابع سابروتین وقفه نوشتیم انجام می دهد و بعد ادامه ی برنامه را اجرا می کند. هر کدام از این وقفه ها را در محل مربوطه توضیح خواهیم داد.

۲-وقفه خارجی : در میکرو پایه هایی به نام INTx وجود دارد که زمانی تحریک شوند میکرو به تابع سابروتین وقفه پرش می کند و برنامه نوشته شده را اجرا می کند . این وقفه ها می توانند با یک لبه بالا رونده یا پایین رونده و یا یک منطقی تحریک شوند.

نکته : در میکروکنترلر Atmega32 تعداد ۱۸ منبع وقفه داخلی و ۳ منبع وقفه خارجی وجود دارد.

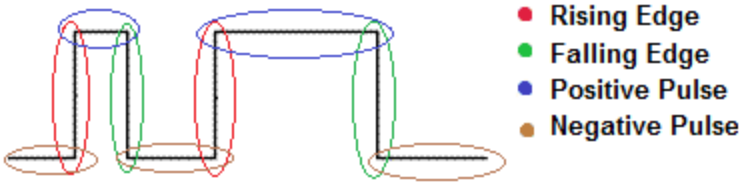
راه اندازی واحد وقفه خارجی در Atmega32

در میکروکنترلر Atmega32 سه وقفه خارجی به نامهای INT0 (پایه ۱۶) ، INT1 (پایه ۱۷) و INT2 (پایه ۳) وجود دارد.

با فعالسازی یک یا چند وقفه خارجی در سربرگ External IRQ در برنامه CodeWizard ، پایه مربوطه به آن به عنوان ورودی تنظیم می شود . سپس بر اساس تنظیمات دلخواه کاربر وقفه میتواند در یکی از ۴ حالت مختلف زیر رخ‌دهد :

- وقفه در لبه بالا رونده پالس ورودی رخ دهد. (Rising Edge)
- وقفه در لبه پایین رونده پالس ورودی رخ دهد. (Falling Edge)
- وقفه در سطح منطقی ۰ رخ دهد. (Low Level)
- وقفه در هر تغییر ۰ به ۱ یا بالعکس رخ دهد. (Any Change)

شکل زیر انواع رخدادهای مختلف ممکن برای یک پالس دیجیتال نمونه را نشان می دهد.



تذکر : وقتی از کلید و مقاومت پول آپ شده استفاده می شود بهتر است وقفه در لبه پایین رونده رخ دهد. چون کلید پول آپ در حالت عادی منطق 1 دارد و با زدن کلید ابتدا یک لبه پایین رونده ایجاد می شود.

با فعال کردن و انجام تنظیمات واحد وقفه خارجی در کدویزارد یک تابع به ابتدای برنامه اضافه می شود که تابع سابروتین وقفه نام دارد. زمانی که وقفه خارجی روی پایه ی مورد نظر رخ دهد ، میکرو در هر کجای اجرای برنامه اصلی در حلقه while نامتناهی که باشد کار خود را رها کرده و تابع سابروتین وقفه را اجرا می کند و بعد از پایان تابع سابروتین وقفه به همان مکان از برنامه اصلی بر میگردد و ادامه برنامه را اجرا می کند.

نکته : برای فعال سازی و غیرفعالسازی کلیه وقفه ها با هم ، از دستورات اسمبلی زیر استفاده می شود . خط اول اجازه سراسری فعالسازی وقفه خارجی را می دهد یعنی فقط بعد از آن وقفه اجازه رخ دادن دارد. برای غیر فعال کردن وقفه در برنامه از عبارت خط دوم استفاده می شود . معمولا در ابتدای سابروتین وقفه ، وقفه را غیر فعال کرده و در پایان سابروتین ، وقفه را دوباره فعال می کنند تا از بروز وقفه مجدد زمانی که برنامه درون تابع سابروتین وقفه قرار دارد جلوگیری شود.

```
#asm(“sei”);  //set enable interrupt

#asm(“cli”);  //clear interrupt
```

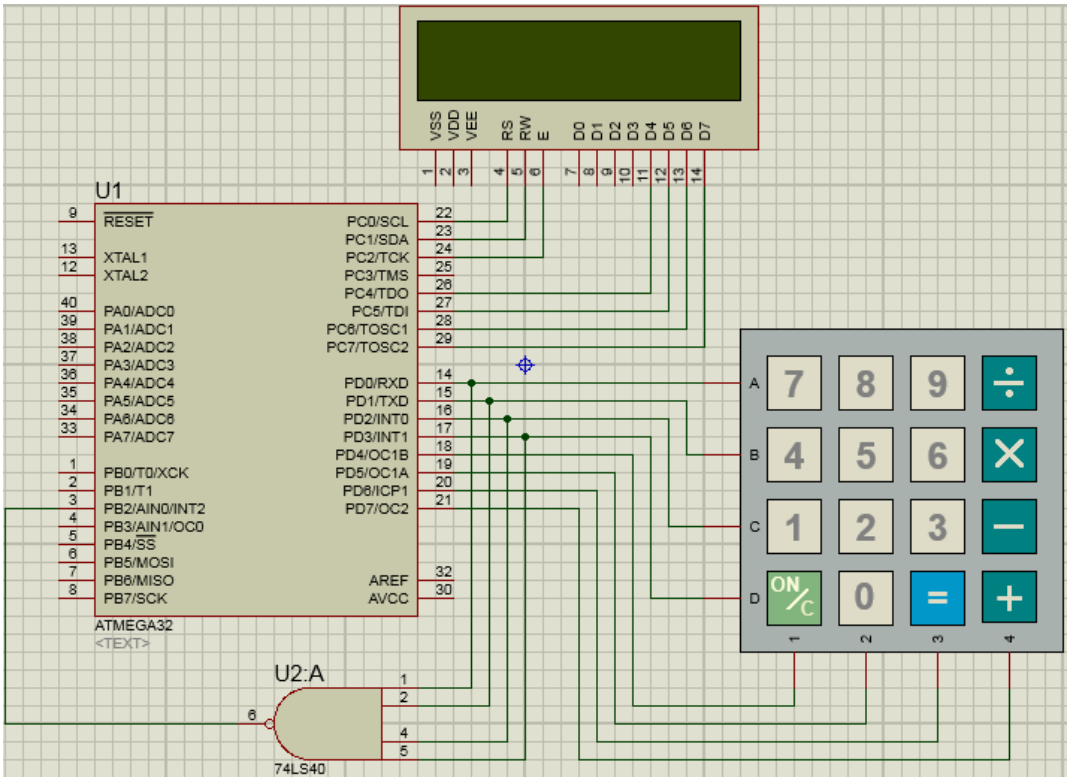
مثال عملی شماره ۵ : یک LCDکاراکتری ۲ در ۱۶ و یک صفحه کلید ۴ در ۴ به روش وقفه به میکروکنترلر متصل کرده و برنامه ای بنویسید که با فشار دادن هر کلید کاراکتر مربوطه روی نمایشگر چاپ شود.

حل :

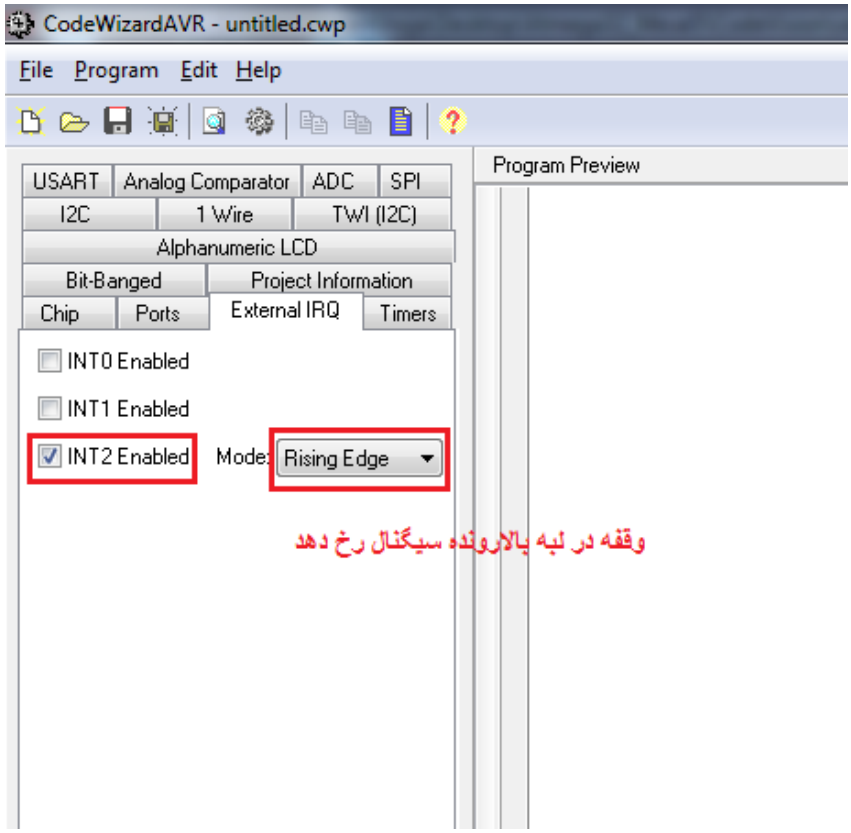
راه اندازی صفحه کلید ۴ در ۴ با وقفه خارجی

مرحله اول : طراحی سخت افزار در پروتئوس

در این مرحله باید سخت افزاری را طراحی کنیم تا به محض اینکه هر یک از کلید ها توسط کاربر زده شود ، پالس وقفه ایجاد شود و برنامه به تابع سابروتین وقفه برود و در آنجا اینکه کدام کلید زده شده محاسبه گردد و چاپ شود . روش های متفاوتی برای تولید پالس وقفه وجود دارد با استفاده از دیود ، گیت های منطقی و ... است اما در این طراحی از یک گیت NOR با ۴ ورودی (آی سی ۷۴۴۰) و اتصال آنها به سطرهای صفحه کلید (ورودی های پول آپ شده میکرو) به صورت شکل زیر استفاده کردیم.



مرحله دوم : انجام تنظیمات در کدویزارد



مرحله سوم : نوشتن برنامه

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

unsigned char c[16];
unsigned char key=20;

void keyboard(void)
{
  //---- ROW1 ----
  PORTD.4=0;
  delay_ms(2);
  if(PIND.0==0) key=7;
  if(PIND.1==0) key=4;
  if(PIND.2==0) key=1;
  if(PIND.3==0) lcd_clear();
  PORTD.4=1;
  //---- ROW2 ----
  PORTD.5=0;
  delay_ms(2);
```

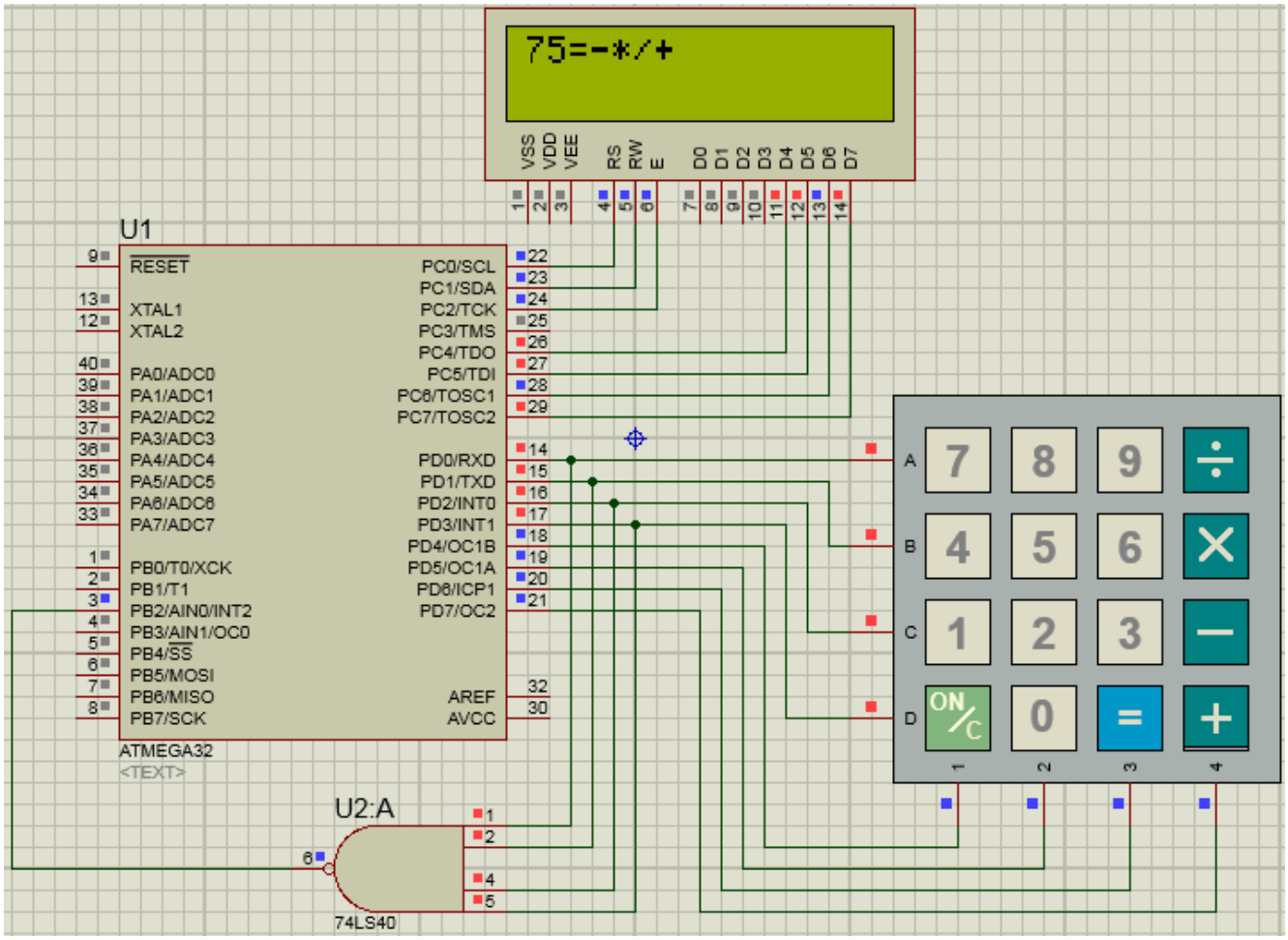
```
if(PIND.0==0) key=8;
if(PIND.1==0) key=5;
if(PIND.2==0) key=2;
if(PIND.3==0) key=0;
PORTD.5=1;
//---- ROW3 ----
PORTD.6=0;
delay_ms(2);
if(PIND.0==0) key=9;
if(PIND.1==0) key=6;
if(PIND.2==0) key=3;
if(PIND.3==0) { lcd_putchar('='); delay_ms(300); }
PORTD.6=1;
//---- ROW4 ----
PORTD.7=0;
delay_ms(2);
if(PIND.0==0) { lcd_putchar('/'); delay_ms(300); }
if(PIND.1==0) { lcd_putchar('*'); delay_ms(300); }
if(PIND.2==0) { lcd_putchar('-'); delay_ms(300); }
if(PIND.3==0) { lcd_putchar('+'); delay_ms(300); }
PORTD.7=1;
}
```

```
interrupt [EXT_INT2] void ext_int2_isr(void)
{
unsigned char i;
#asm("cli");
PORTD=0xFF;
for(i=0;i<5;i++)
keyboard();
PORTD=0x0F;
#asm("sei");
}

void main(void)
{
PORTA=0x00;
DDRA=0x00;

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0xF7;

PORTD=0x0F;
```

```
DDRD=0xF0;


GICR|=0x20;
MCUCR=0x00;
MCUCSR=0x40;
GIFR=0x20;


lcd_init(16);
lcd_clear();


#asm("sei");


while (1){
   if(key!=20){
      sprintf(c,"%d",key);
      lcd_puts(c);
      delay_ms(300);
      key=20; //default value
      }
     }
}
```

*** این برنامه را با برنامه مثال قبل مقایسه کنید.**

مرحله چهارم : شبیه سازی



سورس کدویژن و پروتئوس برنامه فوق را میتوانید از لینک زیر دانلود و نحوه کار را مشاهده نمایید.

دانلود مثال عملی شماره ۵

معرفی و تشریح واحد مبدل آنالوگ به دیجیتال ADC

همانطور که میدانید تمامی کمیت های فیزیکی ، آنالوگ هستند . کمیت های آنالوگ برای پردازش توسط میکروکنترلر ابتدا می بایست تبدیل به دیجیتال شوند. تبدیل ولتاژ ورودی آنالوگ به کد دیجیتال متناسب با آن ولتاژ ورودی توسط این واحد انجام می پذیرد . در هر مبدل آنالوگ به دیجیتال شش مساله اساسی زیر وجود دارد:

1. ولتاژ مرجع (reference Voltage) : که حداکثر ولتاژ قابل نمونه برداری را مشخص می کند.
2. رزولوشن یا دقت نمونه برداری (Resolution) : تعداد بیت های خروجی دیجیتال به توان دو ، دقت نمونه برداری را مشخص می کند.
3. نرخ نمونه برداری (Sample Rate) : یعنی فرکانس نمونه برداری که تعداد نمونه برداری از ولتاژ آنالوگ در واحد زمان است.
4. تعداد کانال ها (channels) : تعداد ولتاژهای آنالوگی که مبدل میتواند به عنوان ورودی آنالوگ به صورت مجزا به دیجیتال تبدیل کند.
5. ضریب بهره کانال (gain) : که مشخص می کند ورودی آنالوگ قبل از تبدیل به دیجیتال در چه عددی ضرب شود (برای سیگنال های ضعیف کاربرد دارد)
6. روش نمونه برداری (ADC type) : به طور کلی ۶ روش زیر برای تبدیل سیگنال های آنالوگ به دیجیتال وجود دارد:

- روش موازی یا همزمان
- روش پله ای
- روش تقریب متوالی(Successive Approximation)
- روش تک شیب
- روش دو شیب
- روش تبدیل ولتاژ به فرکانس

*در کلیه میکروکنترلرهای AVR از روش تقریب متوالی در مبدل ADC استفاده شده است.

ولتاژ مرجع واحد ADC در AVR

در یکی از سه حالت زیر میتواند قرار گیرد:

- AVCC: یعنی همان ولتاژ تغذیه واحد که به پایه تغذیه آنالوگ یا AVCC متصل می شود.
- AREF: ولتاژ مرجع خارجی که ولتاژی دلخواه بین ۰ تا Vcc است و به پایه AREF متصل می شود.
- داخلی internal : ولتاژی داخلی است که مقدار آن ثابت و برابر ۲،۵۶ ولت است.

نکته : اگر از ولتاژ مرجع داخلی استفاده می کنید باید یک خازن ۱۰۰ نانو فاراد بین پایه AREF با زمین قرار دهید.

دقت نمونه برداری واحد ADC در AVR

دقت کلیه میکروکنترلرهای AVR در حالت ۸ بیتی یا ۱۰ بیتی قابل تنظیم می باشد.

سرعت نمونه برداری واحد ADC در AVR

در میکروکنترلرهای AVR سرعت نمونه برداری حداکثر ۱۵ هزار نمونه در ثانیه (15 KSPS) است.

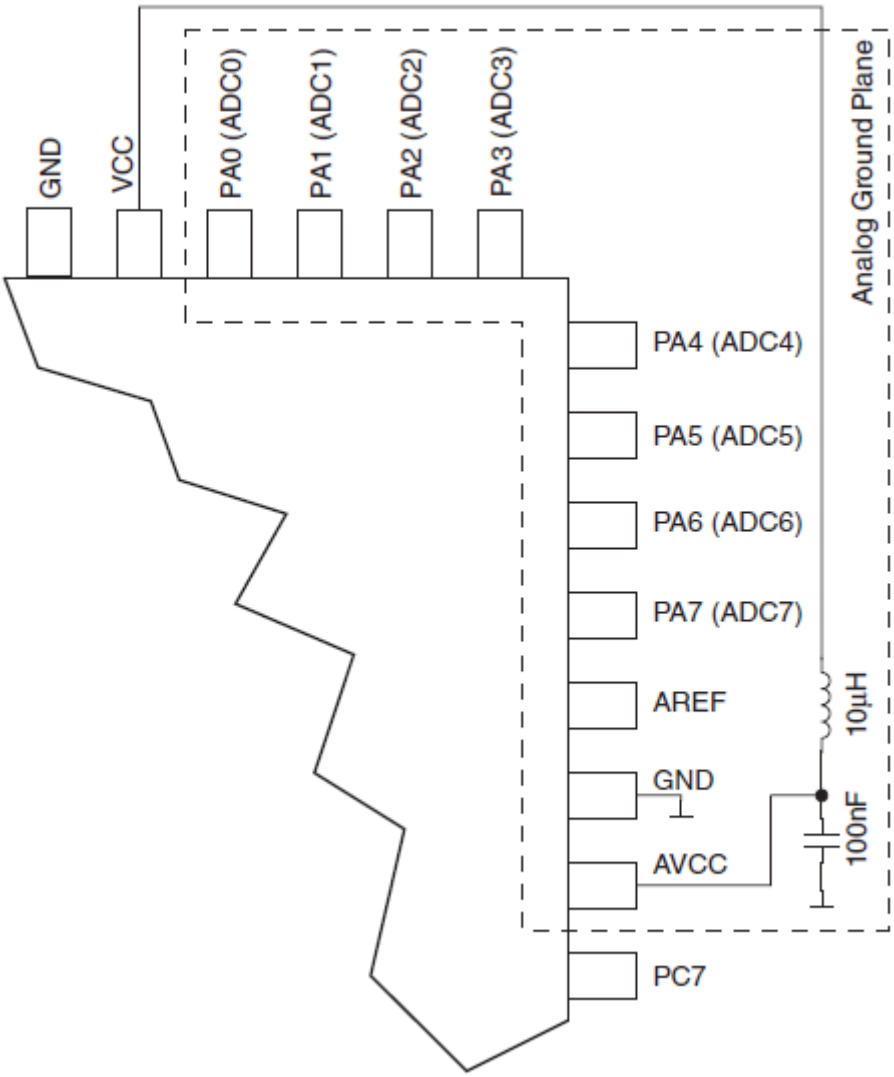
تعداد کانال های ADC در میکروکنترلرهای AVR

در میکروکنترلرهای مختلف متفاوت است برای مثال Atmega32 دارای ۸ کانال مجزا (از پایه های ADC0 تا ADC7) و atmega8 دارای ۶ کانال مجزا می باشد.

فرکانس کار واحد ADC در AVR

فرکانس پالس واحد مبدل ADC طبق پیشنهاد شرکت Atmel باید بین ۵۰ کیلوهرتز تا ۲۰۰ کیلو هرتز باشد (بازه مجاز) . این فرکانس که تقسیمی قابل تنظیم از کلاک اصلی میکرو است ، بسته به نیاز کاربرد مورد نظر باید روی یک مقدار خاصی تنظیم شود.

نکته : برای استفاده از این واحد ابتدا باید تغذیه آن (پایه های ۳۰ و ۳۱) را وصل نمود . برای اینکه نویز محیط روی عملکرد این واحد تاثیر کمتری بگذارد شرکت atmel مدار زیر را برای اتصال تغذیه این واحد پیشنهاد می کند که یک فیلتر متشکل از سلف ۱۰ میکروهانری و خازن ۱۰۰ نانو فاراد مطابق اتصالی به صورت شکل زیر است.



تنظیمات CodeWizaed برای راه اندازی واحد ADC

در نرم افزار به سربرگ ADC رفته و پس از فعالسازی باید ولتاژ مرجع و کلاک را در قسمت های مربوطه مشخص کرد. همانطور که میدانید واحد ADC دارای یک رجیستر ۱۰ بیتی است که میتوان با تیک زدن گزینه Use 8 Bit تنها از ۸ بیت آن استفاده کرد. با استفاده از ۸ بیت دقت ADCکم میشود. همچنین قابلیت ایجاد وقفه نیز با زدن تیک مربوطه فعال می شود .

پس از تولید کد تابع (شماره کانال) read_adc به برنامه اضافه می شود که از این تابع میتوان در برنامه هنگام نیاز به مبدل ADC استفاده کرد . برای استفاده از این مبدل کافیست تا شماره یکی از ۸ کانالی که میخواهیم را در تابع قرار داده تا کد ۱۰ یا ۸ بیتی تبدیل شده به دیجیتال را در خروجی تابع تحویل دهد. اما این کد دیجیتال است و برای استفاده از آن و فهمیدن مقدار آنالوگی که در ورودی بوده است از فرمول زیر استفاده می شود. که در آن n دقت نمونه برداری و برابر ۱۰ یا ۸ است ، V_{ref}برابر مقدار ولتاژ رفرنس و channel شماره کانال ورودی (عددی بین ۰ تا ۷) است.

تعریف ضریب تفکیک : ضریب تفکیک مشخص می کند که حساسیت واحد ADC چقدر است یعنی به ازای چقدر تغییر در سیگنال آنالوگ ورودی ، عدد دیجیتال خروجی مبدل ADC ، یک واحد تغییر می کند.

$$\text{ضریب تفکیک} = \frac{V_{ref}}{2^n - 1}$$

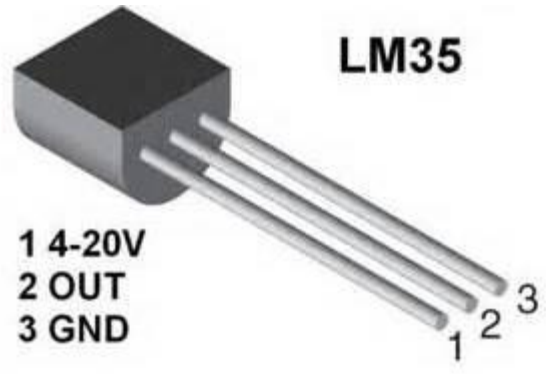
$$V_{in} = \frac{V_{ref}}{2^n - 1} \times read\_adc(channel)$$

مثال عملی شماره ۶ : با استفاده از یک سنسور دماسنج LM35 و LCD کاراکتری ۲ در ۱۶ برنامه دماسنج دیجیتال را بنویسید.

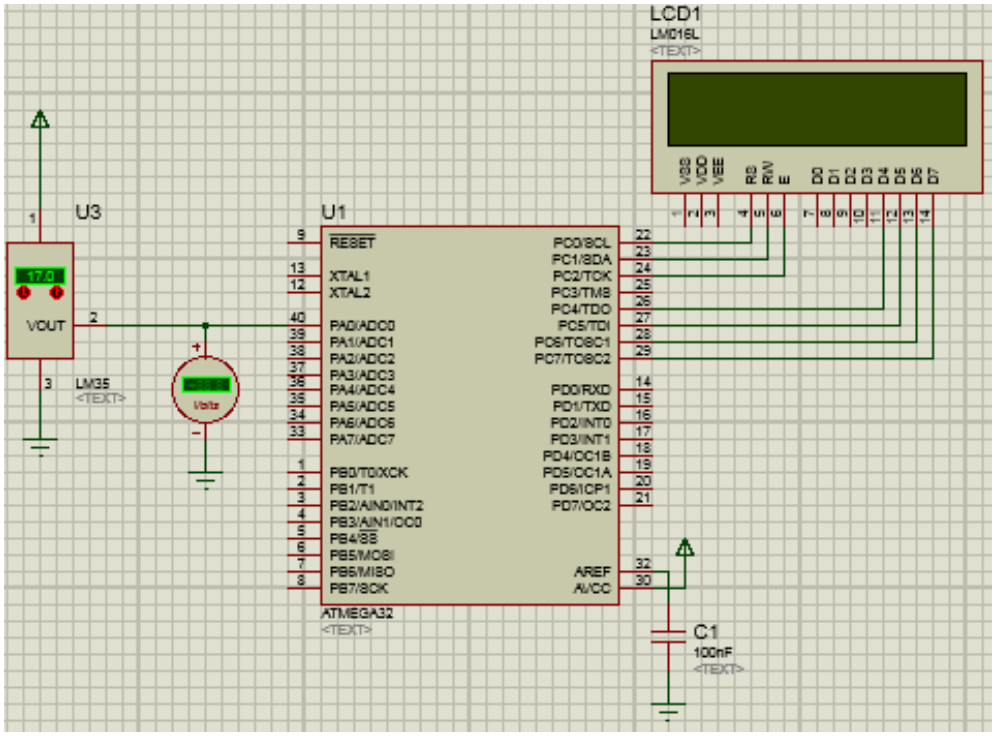
حل:

راه اندازی سنسور LM35

خروجی این سنسور ولتاژی از نوع آنالوگ می باشد که در دمای ۰ درجه ، ولتاژ پایه خروجی آن نیز ۰ می شود و به ازای افزایش هر درجه سانتی گراد دمای محیط اطرافش ، خروجی سنسور ۱۰ میلی ولت افزایش پیدا می کند . پایه های این سنسور به صورت شکل زیر است:

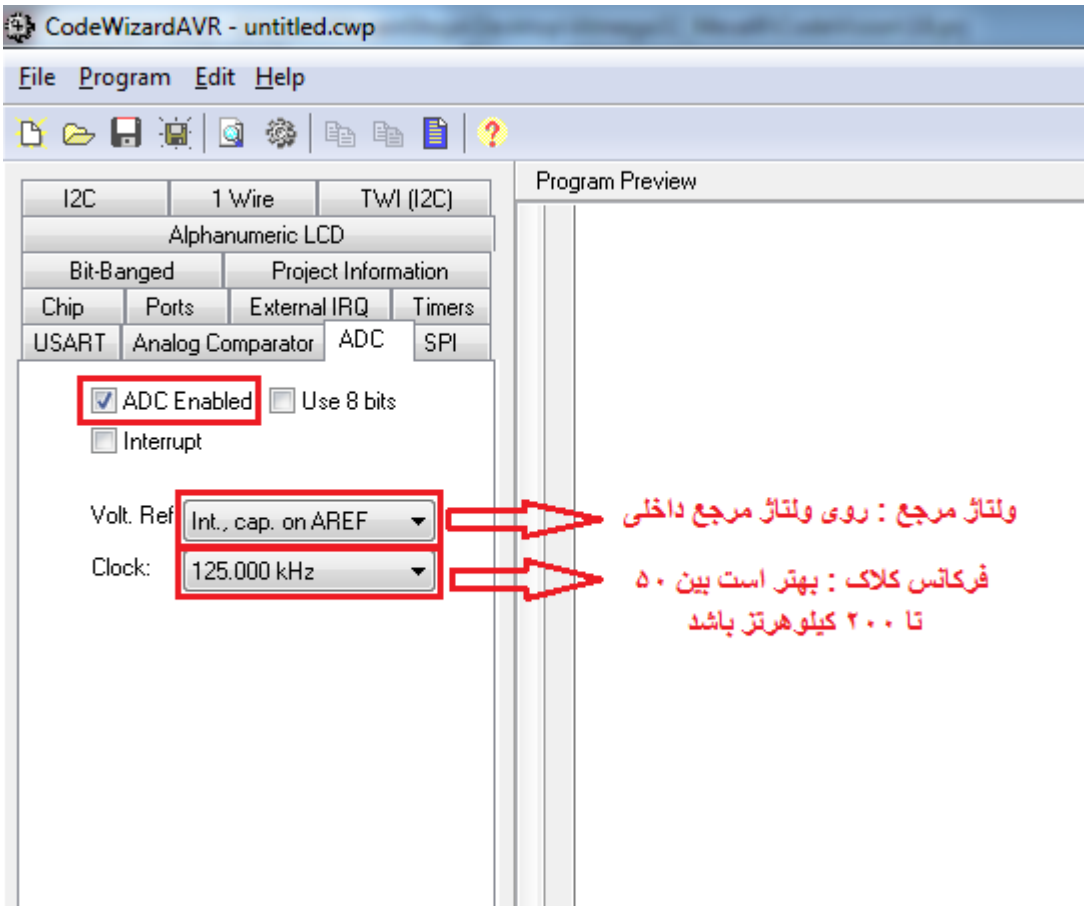


مرحله اول : طراحی سخت افزار



مرحله دوم : تنظیمات کدویزارد

بعد از انجام تنظیمات سربرگ های chip ، portو Alphanumeric LCD به همان صورت مثال شماره ۴ به سربرگ ADC رفته و تنظیمات را به صورت زیر انجام می دهیم :



مرحله سوم : نوشتن برنامه

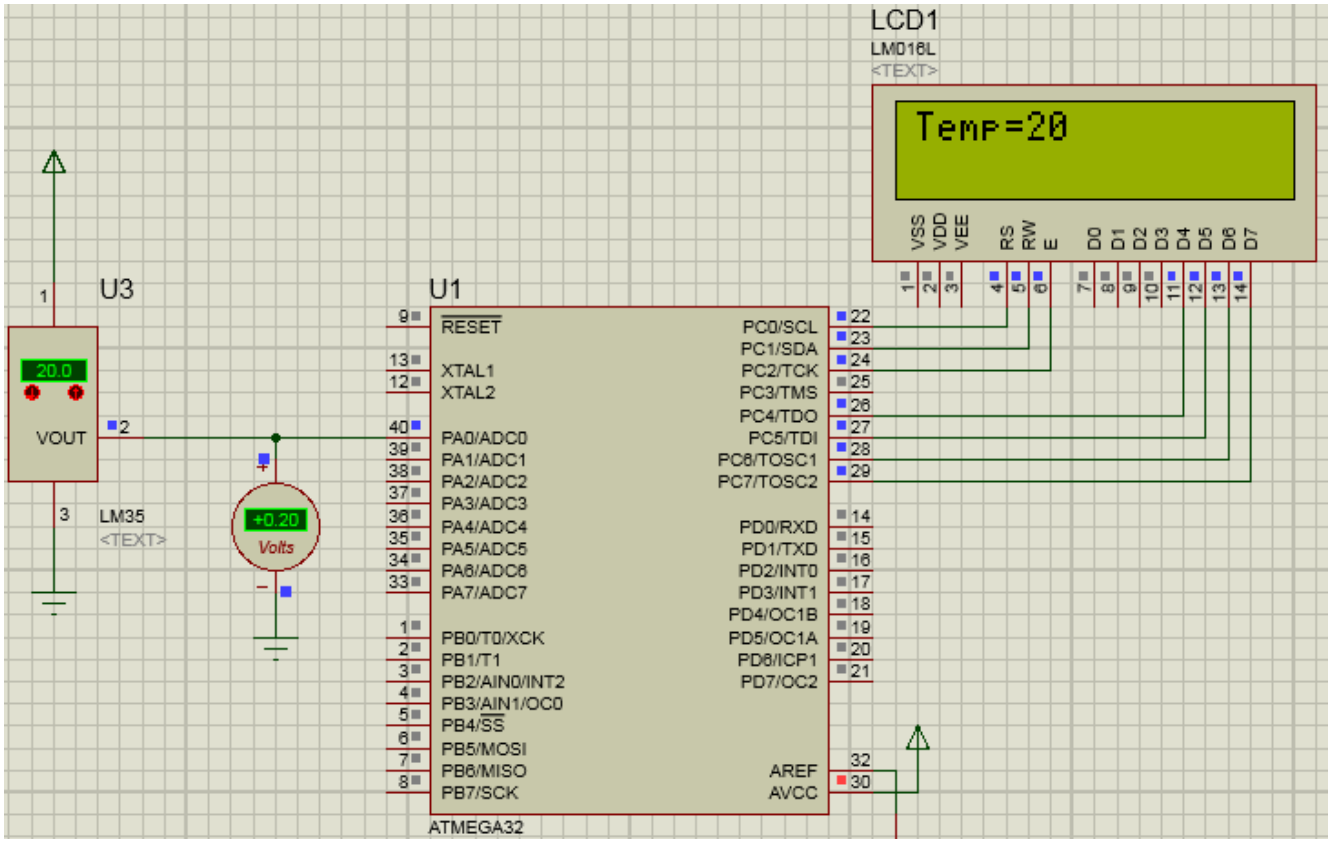
```
#include <mega32.h>

#include <delay.h>

#include <stdio.h>

#include <alcd.h>

#define ADC_VREF_TYPE 0xC0
```

```
unsigned int read_adc(unsigned char adc_input){
ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
delay_us(10);
ADCSRA|=0x40;
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}
unsigned int a;
char s[15];
void main(void){
PORTC=0x00;
DDRC=0xF7;
ADMUX=ADC_VREF_TYPE & 0xff;
ADCSRA=0x83;
lcd_init(16);
while (1){
a=read_adc(0);
sprintf(s,"Temp=%d  ",a/4);
lcd_gotoxy(0,0);
lcd_puts(s);
} }
```

توضیح : بعد از حذف کامنت ها و برنامه های اضافی از برنامه تولید شده توسط کدویزارد ، ضریب تفکیک را از رابطه گفته شده بر حسب میلی ولت تقریبا ۲،۵۰ محاسبه می کنیم اما عددی که باید نمایش دهیم ولتاژ نیست بلکه دماست پس چون به ازای هر ۱۰ میلی ولت باید دما یک واحد افزایش یابد یعنی به ازای ۴ برابر ضریب تفکیک بنابراین در برنامه عدد دیجیتال گرفته شده را تقسیم بر ۴ می کنیم.

مرحله چهارم : شبیه سازی

بعد از اضافه کردن فایل hex در نرم افزار پروتئوس برنامه را run می کنیم.



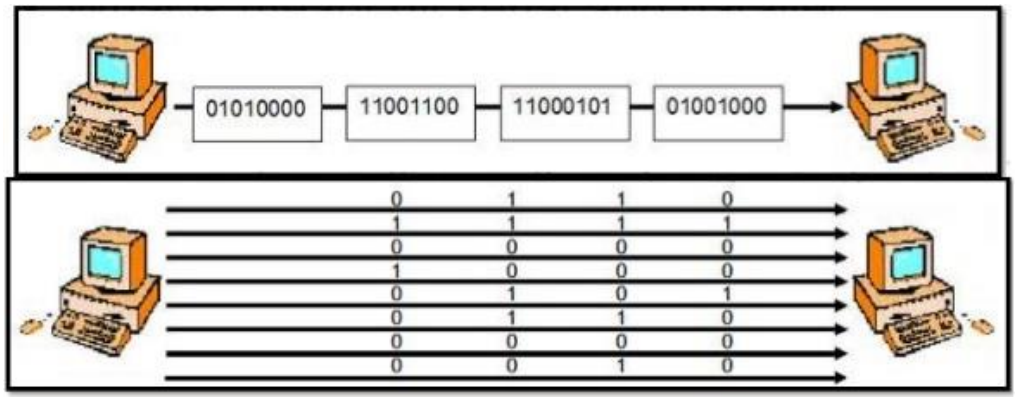
سورس کدویژن و پروتئوس برنامه فوق را میتوانید از لینک زیر دانلود و نحوه کار را مشاهده نمایید.

دانلود مثال عملی شماره ۶

ادامه فصل هشتم : آموزش واحدهای ارتباط سریال میکروکنترلرAtmega32

مقدمه:

اساسا انتقال اطلاعات به دو شکل موازی و سریال صورت می گیرد . در ارتباط موازی n بیت اطلاعات توسط n خط موازی منتقل می شود اما در ارتباط سریال اطلاعات از طریق یک خط به صورت پشت سر هم انجام می گیرد. شکل زیر نحوه ارتباط سریال و موازی را مابین دو کامپیوتر نشان می دهد . همانطور که مشاهده می شود به علت اینکه در انتقال سریال n بیت داده از طریق یک خط عبور می کند ، سرعت آن نسبت به انتقال موازی کمتر است اما به علت اینکه از سیم کمتری استفاده می شود در انتقال اطلاعات در فواصل بالا بر ارتباط موازی ارجحیت دارد.



ارتباطات سریال و موازی در میکروکنترلرها

تبادل دیتا با محیط خارجی میکروکنترلر علاوه بر واحد ورودی/خروجی می تواند از طریق واحدهای ارتباطی سریال صورت گیرد. واحد ورودی/خروجی به صورت موازی و واحدهای ارتباط سریال به صورت سریال با محیط پیرامون میکرو در ارتباط است . مهمترین مسائلی که در ارتباط سریال با آن درگیر هستیم به شرح زیر است:

- پروتکل ارتباطی سریال
- سرعت ارتباط سریال
- نوع فرستنده و گیرنده در ارتباط سریال
- انواع حالت های ارتباط سریال
- روش ارسال سنکرون یا آسنکرون

انواع پروتکل های ارتباطی سریال در میکروکنترلرهای AVR

مهمترین مسئله در ارتباطات سریال یکی **پروتکل** ارتباطی و دیگری **سرعت** ارسال و دریافت اطلاعات است. پروتکل های ارتباطی سریال که توسط میکروکنترلرهای AVR و اکثر میکروکنترلرهای دیگر پشتیبانی می شود عبارتند از :

- پروتکل **spi** : دارای سرعت بسیار بالا می باشد و در فواصل بسیار کوتاه از آن استفاده می شود. از طریق این ارتباط میکروکنترلر ها را نیز میتوان پروگرام کرد.
- پروتکل **USB** : دارای سرعت بالا می باشد و در فواصل کوتاه از آن استفاده می شود. یک پروتکل جهانی استاندارد برای ارتباط با اکثر دستگاههای جانبی است.
- پروتکل **USART** : دارای سرعت متوسط می باشد و در مسافت های طولانی از آن استفاده می شود . همچنین برای ارتباط با اکثر ماژول های ارتباطی مانند GSM ، GPS، HM-TR و... کاربرد دارد.
- پروتکل **TWI**یا همان **I2C** : یا پروتکل دوسیمه بیشتر برای ارتباط با المانهای جانبی نظیر سنسورها و ماژول های سرعت پایین و در فواصل کوتاه است.

تذکر : پروتکل ارتباط سریال USB تنها توسط میکروکنترلرهای AVR سری **Xmega**پشتیبانی می شود.

نکته : از پروتکل های مهم ارتباطی سریال که توسط میکروکنترلرهای AVR پشتیبانی نمی شود میتوان پروتکل های Ethernet ، CAN، HDMIو I2S را نام برد.

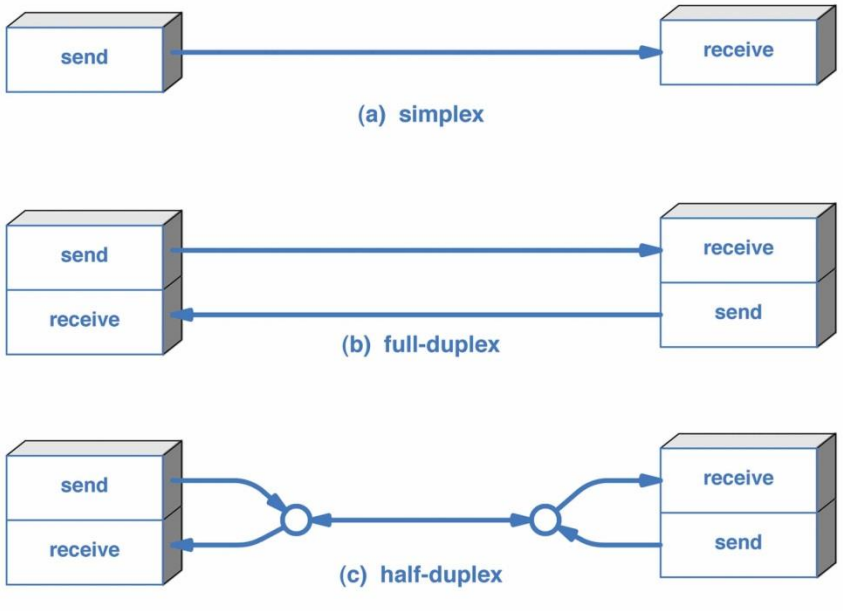
نوع فرستنده و گیرنده در ارتباط سریال

به فرستنده اطلاعات اصطلاحا Master و به گیرنده اطلاعات Slave گفته می شود . Master یا فرستنده اطلاعات را برای Slaveیا گیرنده ارسال می کند . Master همواره یک دستگاه است اما Slave میتواند چندین دستگاه مختلف باشد که اطلاعات را از Master دریافت می کنند . در حالت کلی فرستنده یا گیرنده ها میتواند هر یک از دستگاههای دیجیتالی مانند کامپیوتر ، لپ تاپ ، تبلت ، میکروکنترلر ، آی سی یا هر ماژول ارتباطی باشد که ارتباط سریال را پشتیبانی می کند اما در ادامه بحث فرض میکنیم حداقل یکی از Master یا Slave ها میکروکنترلر AVR باشد.

انواع حالت ارتباط سریال :

به طور کلی ۳ روش ارتباط بین فرستنده و گیرنده وجود دارد که میتوان در میکروکنترلرهای AVR نیز آنها را پیاده کرد:

- روش **یک طرفه** یا ساده : (Simplex) در این روش اطلاعات فقط در یک جهت انجام می گیرد.
- روش **نیم دوطرفه** : (Half Duplex)در این روش اطلاعات در هر دو جهت میتواند انجام گیرد اما در هر لحظه فقط در یک جهت امکان پذیر است.
- روش **دو طرفه** : (Full Duplex)در این روش اطلاعات در یک لحظه میتواند در دو جهت انجام گیرد.



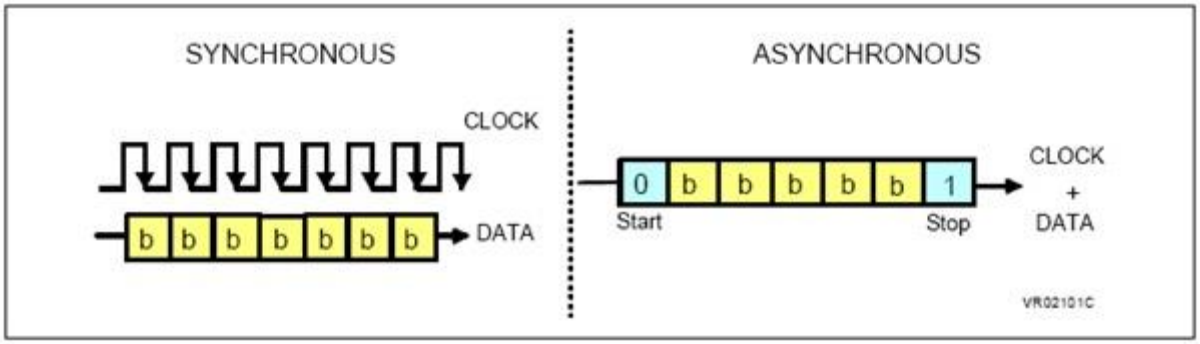
روش ارسال اطلاعات سریال :

دو روش کلی برای انتقال سریال اطلاعات وجود دارد که در میکروکنترلرهای AVR هر دو روش وجود دارد:

- **روش سنکرون یا همزمان :** در این روش به همراه خط ارتباط سریال یک خط دیگر برای کلاک وجود دارد به طوری که در هر لبه بالارونده کلاک ، یک نمونه از سیگنال اطلاعات برداشته می شود و کنار هم گذاشته

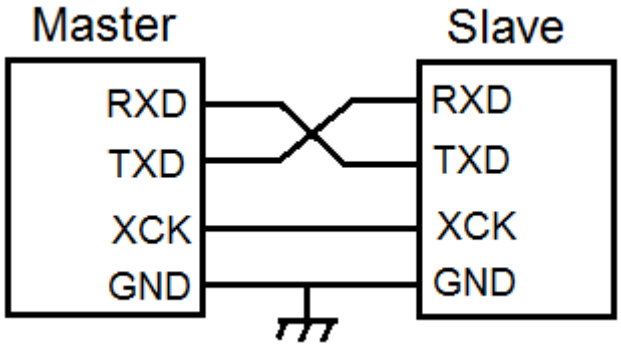
می شود . بنابراین سرعت نمونه برداری را سرعت کلاک مشخص می کند و میتواند سرعت افزایش یا کاهش یابد.

- **روش آسنکرون یا غیر همزمان :** در این روش اطلاعات با یک سرعت استاندارد تنظیم شده و غیرقابل تغییر ارسال می شود . بنابراین دیگر پالس کلاک وجود ندارد و هر دو فرستنده و گیرنده میدانند که با چه سرعتی قرار است اطلاعات را ارسال یا دریافت کنند.

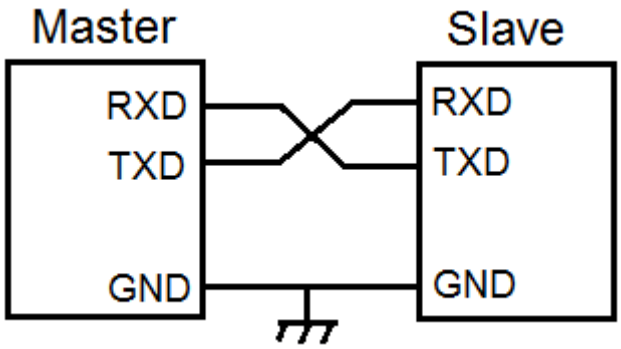


معرفی و تشریح واحد ارتباطی سریال USART

همواره یکی از مسائلی که با آن درگیر هستیم ارتباط دو یا چند دستگاه با هم است که بتوانند با یکدیگر دیتا ارسال کنند. روش های متنوعی برای این منظور وجود دارد که یکی از آنها USART می باشد. ارتباط سریال یوزارت مخفف عبارت **U**niversal **S**ynchronous And **A**synchronous Serial **R**eceiver And **T**ransmitter به معنای "فرستنده/گیرنده جهانی سریال سنکرون/آسنکرون" می باشد . همانگونه که از اسم این واحد مشخص است ، واحد USART در میکروکنترلرهای AVR از دو حالت سنکرون و آسنکرون پشتیبانی می کند . در حالت ارتباط سنکرون از سیم بندی زیر بین فرستنده و گیرنده استفاده می شود:

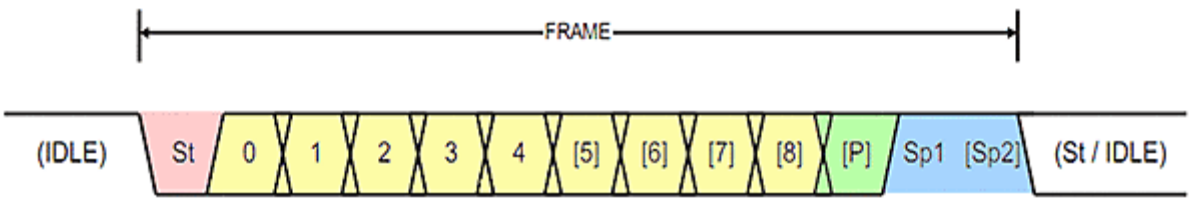


که در آن RXD برای دریافت دیتا ، TXDبرای ارسال دیتا ، XCK برای کلاک حالت سنکرون و GND زمین مشترک دو دستگاه می باشد . اما به دلیل اینکه در صنعت همواره سعی بر این است که تعداد سیم ها کم شود ، در اکثر اوقات از حالت آسنکرون استفاده می کنیم . در ارتباط آسنکرون سیم کلاک را حذف کرده و به جای آن پارامتری به نام نرخ ارسال یا Baud Rate را اضافه کنیم که مشخص می کند که در هر ثانیه چند بیت ارسال یا دریافت می شود . این ارتباط می تواند بین دو یا چند دستگاه مختلف صورت بگیرد برای مثال می تواند ارسال/دریافت دیتا بین دو یا چند میکرو ، یک یا چند میکرو با یک کامپیوتر ، یک میکرو با ماژول های ارتباطی مختلف نظیر ماژول GSM ، ماژول GPS و ... باشد . شکل زیر نحوه ارتباط سریال آسنکرون را بین Master و Slave نشان می دهد . به ارتباط یوزارت در حالت آسنکرون **UART** گفته می شود . در ادامه این آموزش به علت متداول بودن ، فقط ارتباط آسنکرون شرح داده خواهد شد.



قالب ارسال/دریافت دیتا در پروتکل UART (آسنکرون)

هنگام ارسال دیتا علاوه بر خود دیتا تعدادی بیت کنترلی نیز همراه با آن ارسال می شود که به این مجموعه اصطلاحا یک فریم (frame) گفته می شود.



بیت شروع START

در وضعیتی که ارسال و دریافت صورت نمی گیرد خط انتقال در حالت یک منطقی است. با ایجاد یک لبه ی پایین رونده توسط فرستنده ، گیرنده از فرستاده شدن اطلاعات آگاه شده و آماده ی دریافت می شود.

بیت های داده DATA

بیت های داده اطلاعات اصلی را منتقل می کند و می تواند بین ۵ تا ۹ بیت متغیر باشد . انتخاب تعداد این بیت ها با کاربر است و باید در فرستنده و گیرنده به صورت یکسان تنظیم شود.

بیت توازن PARITY

پس از ارسال بیت های داده فرستنده می تواند بیت توازن را ارسال کند. استفاده از این بیت اجباری نبوده و در صورت استفاده می تواند در آشکارسازی خطا کمک کند.

بیت یا بیت های پایان STOP

در ادامه ی بیت های داده یا بیت توازن در صورت استفاده دست کم ۱ بیت پایان ارسال می شود. بیت پایان همواره یک است و وجود بیت دوم (SP2) دلخواه است.

مفهوم Baud Rate

Baud rate عرض هر بیت را مشخص می کند . دو طرف ارتباط باید از عرض هر بیت اطلاع داشته باشند. اگر در یک ارتباط سریال baud rate برابر ۹۶۰۰ bps باشد به این معنی است که فرستنده باید ۹۶۰۰ بیت را در یک ثانیه (Bit Per Second) ارسال کند. در این صورت عرض هر بیت برابر می شود با :

T_{bit}=1/baud rate= 1/9600 = 104 us

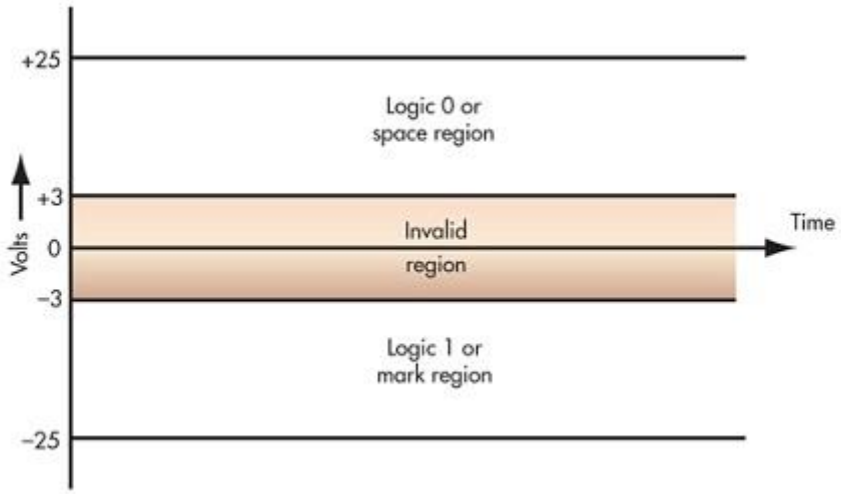
پروتکل های ارتباطی UART تحت استاندارد های RS423 ، RS232 ، RS422 و RS485

استفاده از پروتکل UART معمولی برای مسافتهای بیشتر از چند متر و با Baud Rate بالا و در محیطهای با نویز زیاد ، قابل اطمینان و پیاده سازی نیست . چرا که اولا در مسافتهای طولانی اثر نویزهای محیطی بیشتر می شود و ثانیا در فرکانسهای بالا ، تشعشع خط فرستنده ، روی گیرنده اثر می گذارد . برای اینکه دستگاه ها بتوانند در فاصله ی بیشتری از هم قرار گیرند و از طریق پروتکل UART با هم ارتباط داشته باشند ، استانداردهایی تدوین شدند .
پروتکل RS232 فراگیرترین و معمول ترین استاندارد جهت انتقال دیتا می باشد . در استاندارد RS232 در سرعت 20 Kbps حداکثر طول کابل قابل استفاده ۷۰۵ متر می تواند باشد تا داده ها تقریبا به صورت صحیح به مقصد برسند.
پروتکل RS422 شباهت زیادی به RS232 دارد ولی تا ۱۰ گیرنده را پشتیبانی می کند. این پروتکل که از خطوط بالانس شده برای انتقال داده استفاده می کند ، اثر نویز پذیری را بشدت کاهش داده است به طوری که بیشترین فاصله بین فرستنده و گیرنده در این پروتکل ۱۲۰۰ متر و بیشترین سرعت برابر Mbps۱۰ است. در پروتکل RS485 تعداد گیرنده ها میتواند حداکثر تا ۳۲ هم باشد ضمن اینکه بیشترین فاصله ۱۲۰۰ متر و بیشترین سرعت Mbps۱۰ می‌باشد . شکل زیر مقایسه کلی بین این پروتکل ها را نشان می دهد . به علت اینکه در این آموزش به فواصل و گیرنده های زیادی نیاز نداریم تنها به تشریح استاندارد RS232 کفایت می کنیم .

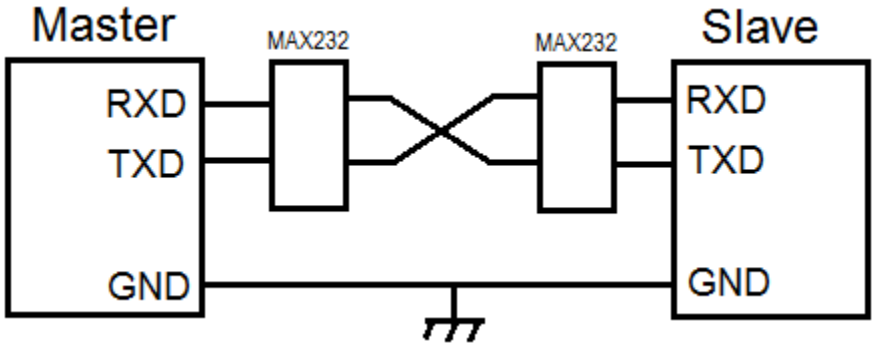
SPECIFICATIONS		RS232	RS423	RS422	RS485
Mode of Operation		SINGLE -ENDED	SINGLE -ENDED	DIFFERENTIAL	DIFFERENTIAL
Total Number of Drivers and Receivers on One Line (One driver active at a time for RS485 networks)		1 DRIVER 1 RECVR	1 DRIVER 10 RECVR	1 DRIVER 10 RECVR	32 DRIVER 32 RECVR
Maximum Cable Length		50 FT.	4000 FT.	4000 FT.	4000 FT.
Maximum Data Rate (40ft. - 4000ft. for RS422/RS485)		20kb/s	100kb/s	10Mb/s-100Kb/s	10Mb/s-100Kb/s
Maximum Driver Output Voltage		+/-25V	+/-6V	-0.25V to +6V	-7V to +12V
Driver Output Signal Level (Loaded Min.)	Loaded	+/-5V to +/-15V	+/-3.6V	+/-2.0V	+/-1.5V
Driver Output Signal Level (Unloaded Max)	Unloaded	+/-25V	+/-6V	+/-6V	+/-6V
Driver Load Impedance (Ohms)		3k to 7k	>=450	100	54
Max. Driver Current in High Z State	Power On	N/A	N/A	N/A	+/-100uA
Max. Driver Current in High Z State	Power Off	+/-6mA @ +/-2v	+/-100uA	+/-100uA	+/-100uA
Slew Rate (Max.)		30V/uS	Adjustable	N/A	N/A
Receiver Input Voltage Range		+/-15V	+/-12V	-10V to +10V	-7V to +12V
Receiver Input Sensitivity		+/-3V	+/-200mV	+/-200mV	+/-200mV
Receiver Input Resistance (Ohms), (1 Standard Load for RS485)		3k to 7k	4k min.	4k min.	>=12k

استاندارد RS232

ساده ترین استاندارد برای ارتباط سریال در مسافت های بیشتر از چندین متر می باشد . اساس کار این پروتکل در تغییر سطوح منطقی ولتاژ است به طوری که سطح ۱ منطقی (۵ ولت) را به ۲۵ ولت و سطح ۰ منطقی (۰ ولت) را به -۲۵ ولت می رساند . در طول مسیر ممکن است سطح این ولتاژ کاهش یابد اما تا +۳ ولت برای منطق ۱ و تا -۳ ولت نیز برای منطق ۰ قابل قبول است . شکل زیر محدوده تغییرات این استاندارد را نشان می دهد.



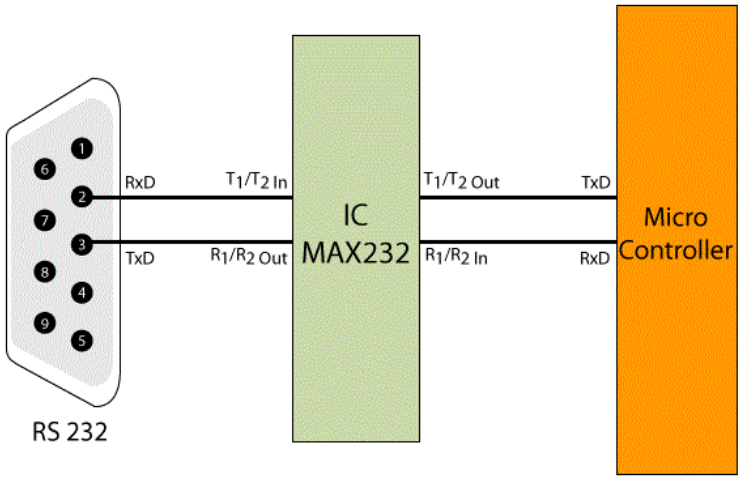
تبدیل سطوح ولتاژ توسط آی سی های مختلفی صورت می گیرد که مشهورترین آن ها به نام آی سی MAX232 است . بنابراین در صورت استفاده از این استاندارد دو آی سی Max232 میان Master و Slave به صورت شکل زیر اضافه می شود.



کابل مورد استفاده در پروتکل RS232 حداقل باید دارای ۳ رشته سیم باشند که یک سیم برای خط ارسال داده یا TXو یک سیم برای خط دریافت داده یا RX و سیم سوم نیز به عنوان سیم ولتاژ مرجع استفاده می شود. این کابل واسط می تواند از سیم های موازی ساده یا از سیم های جفت به هم تابیده شده باشند. طول کابل حداکثر نرخ انتقال داده (Buad Rate) را محدود می کند به طوری که طول کابل را در نرخ های انتقال پایین تر می توان طولانی تر در نظر گرفت.

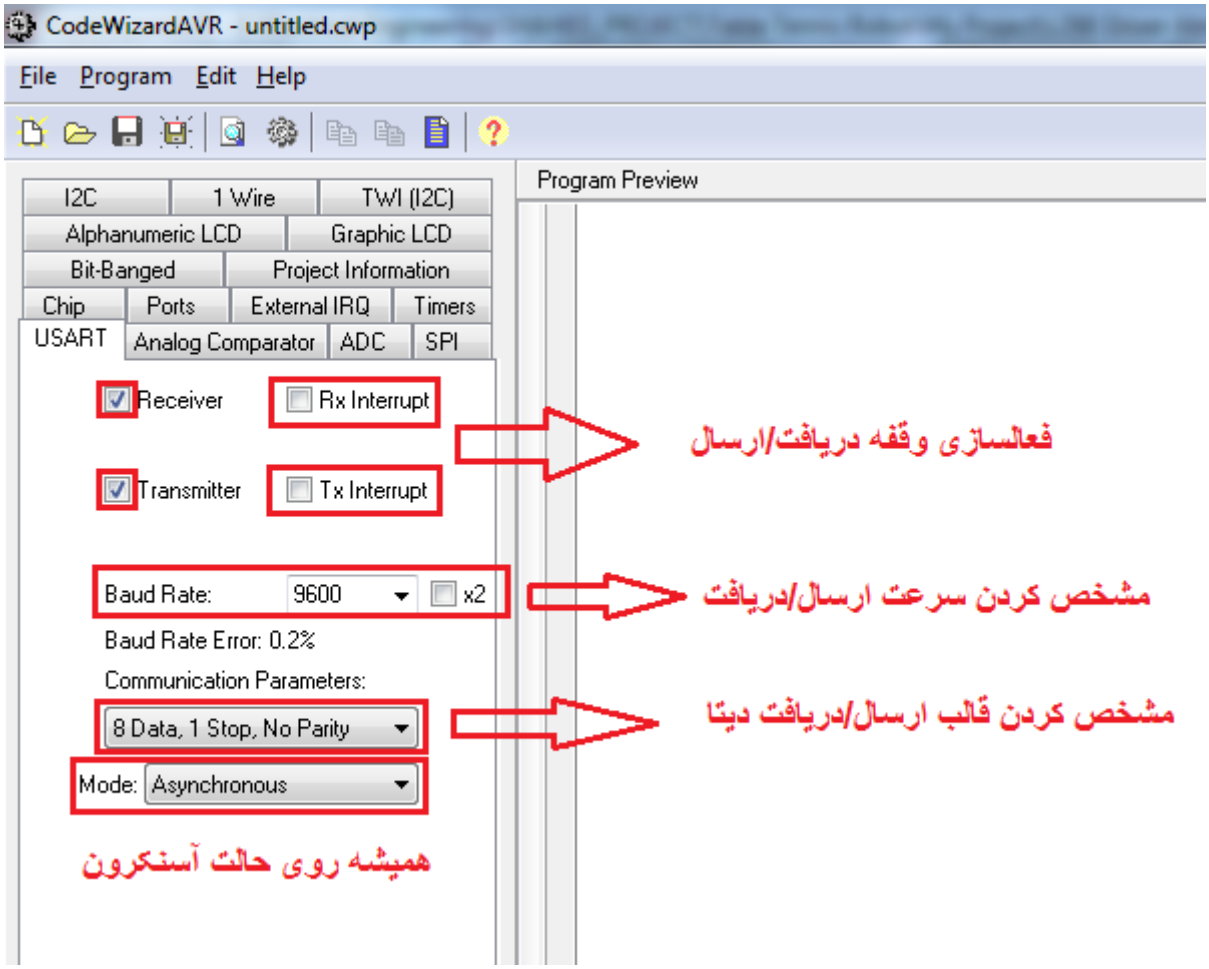
نرخ Bd (Baud)	بیشترین طول (FT)	بیشترین طول [m]
19200	50	15
9600	500	150
4800	1000	300
2400	3000	900

از استاندارد RS232 در پورت های سریال پشت کامپیوتر های PC (پورت DB9) استفاده شده است. بنابراین پورت سریالی که پشت PC ها وجود دارد دارای پروتکل RS232 است و با اضافه نمودن آی سی MAX232 به صورت شکل زیر و از طریق نرم افزارهای ترمینال موجود در کامپیوتر می توان ارتباط میان میکروکنترلر و PC را برقرار نمود .



تنظیمات واحد USART در کدویزارد Codewizard

پس از رفتن به سربرگ USART ، بر حسب نیاز به حالت یک طرفه یا دوطرفه می توان Reciever ، Transmitterیا هر دو را فعال کرد . اگر گزینه فعالسازی وقفه (interrupt) نیز فعال شود ، وقفه داخلی برای ارسال/دریافت فعال می شود و به برنامه تابع سابروتین مربوطه اضافه می گردد . سپس در قسمت Baud Rate نرخ ارسال/دریافت دیتا را مشخص می کنیم . نرخ ارسال/دریافت باید طوری باشد که مقدار درصد خطا که زیر آن نوشته شده است ، مقدار قابل قبولی باشد . با فعال کردن گزینه x2 نیز میتوان نرخ ارسال/دریافت را توسط مدار ضرب کننده دو برابر کرد . در قسمت Communication Parameter نوع قاب دیتا را مشخص می کنیم . توجه شود که قاب دیتا و نرخ ارسال/دریافت در گیرنده و فرستنده باید یکی باشد . در قسمت Mode نیز سنکرون یا آسنکرون بودن ارتباط را مشخص می کنیم (این قسمت همیشه روی آسنکرون است).



پس از تولید کد توسط برنامه کدویزارد مشاهده می شود کتابخانه stdio.h به برنامه اضافه می شود . درون این کتابخانه توابع کار با USART وجود دارد که در طول برنامه نویسی می توان از آنها برحسب نیاز استفاده کرد.

توابع پرکاربرد stdio.h در هنگام کار با واحدUSART

1. **تابع getchar**

این تابع بدون ورودی و دارای خروجی از نوع char می باشد . با نوشتن دستور زیر تابع منتظر می ماند تا یک کاراکتر توسط USART دریافت شود و سپس مقدار دریافت شده را به داخل یک کاراکتر از قبل تعریف شده باز می گرداند . توجه شود تا زمانی که داده از USART وارد نشده باشد برنامه منتظر می ماند و هیچ کاری انجام نمی دهد.

نام کاراکتر=getchar();

2. **تابع putchar**

این تابع که دارای یک ورودی از نوع char و بدون خروجی می باشد ، کاراکتر ورودی را توسط USART ارسال می کند.

putchar(‘ کاراکتر ‘);

3. **توابع puts و putsf**

برای ارسال یا دریافت یک رشته به صورت کامل به وسیله USART از دستورات putsf و puts استفاده می شود. تفاوت puts با putsf در این است که تابع puts رشته ی موجود در SRAM را و putsf رشته ذخیره شده در فلش را ارسال می کند.

putsf(“ رشته “);

puts(متغیر رشته ای);

نکته : برای مقدار دهی به یک متغیر رشته ای از تابع sprintf استفاده می شود.

4. تابع gets

برای دریافت رشته ها از واحد USART و ذخیره آنها روی یک متغیر رشته ای از این تابع استفاده می شود . این تابع دارای دو ورودی و بدون خروجی می باشد . ورودی اول این تابع رشته ای است که میخواهیم اطلاعات دریافت شده روی آن ذخیره شود و ورودی دوم این تابع که از نوع unsigned char است طول رشته را مشخص می کند . تا زمانی که به تعداد معین کاراکتر دریافت نشود ، کاراکترهای دریافت شده را از USRAT گرفته و در متغیر رشته ای ذخیره می کند.

gets(متغیر رشته ای , طول رشته);

5. تابع printf

تفاوت دستور printf با puts یا putsf در این است که می توان با دستور printf متغیر ها و رشته ها را با هم و با فرمت دلخواه ارسال کرد. برای مثال میخواهیم دمای اتاق را ارسال کنیم ، به صورت زیر میتوان این کار را انجام داد . با نوشتن کد زیر تمامی کاراکترهای موجود در عبارت " Temp=%d " به سرعت و پشت سر هم از طریق واحد یوزارت ارسال می شود و به جای %d در عبارت فوق دمای اتاق که در متغیر i قرار دارد ، ارسال می شود.

printf("Temp=%d",i);

جدول زیر فرمت متغیرهای کاراکتری قابل ارسال توسط تابع printf را نشان می دهد.

کاراکتر	نوع اطلاعات ارسالی
%c	یک تک کاراکتر
%d	عدد صحیح علامت دار در مبنای ۱۰
%i	عدد صحیح علامت دار در مبنای ۱۰
%e	نمایش عدد ممیز شناور به صورت علمی
%E	نمایش عدد ممیز شناور به صورت علمی
%f	عدد اعشاری
%s	عبارت رشته ای واقع در حافظه SRAM

%u	عدد صحیح بدون علامت در مبنای ۱۰
%X	به فرم هگزا دسیمال با حروف بزرگ
%x	به فرم هگزا دسیمال با حروف کوچک
%p	عبارت رشته ای واقع در حافظه FLASH
%%	نمایش علامت ٪

تعیین طول (width) و دقت (precision) خروجی در تابع printf

تابع printf این قابلیت را دارد که طول داده ارسالی و دقت آن را تعیین نماید . طول و دقت بعد از کاراکتر ٪ و قبل از حروف نشان دهنده فرمت به صورت دقت.طول نوشته می شود . برای مثال میخواهیم دقت یک عدد اعشاری را تا ۴ رقم اعشار و طول آن را تا ۷ رقم در نظر بگیریم باید آن را در تابع printf به صورت زیر بنویسیم:

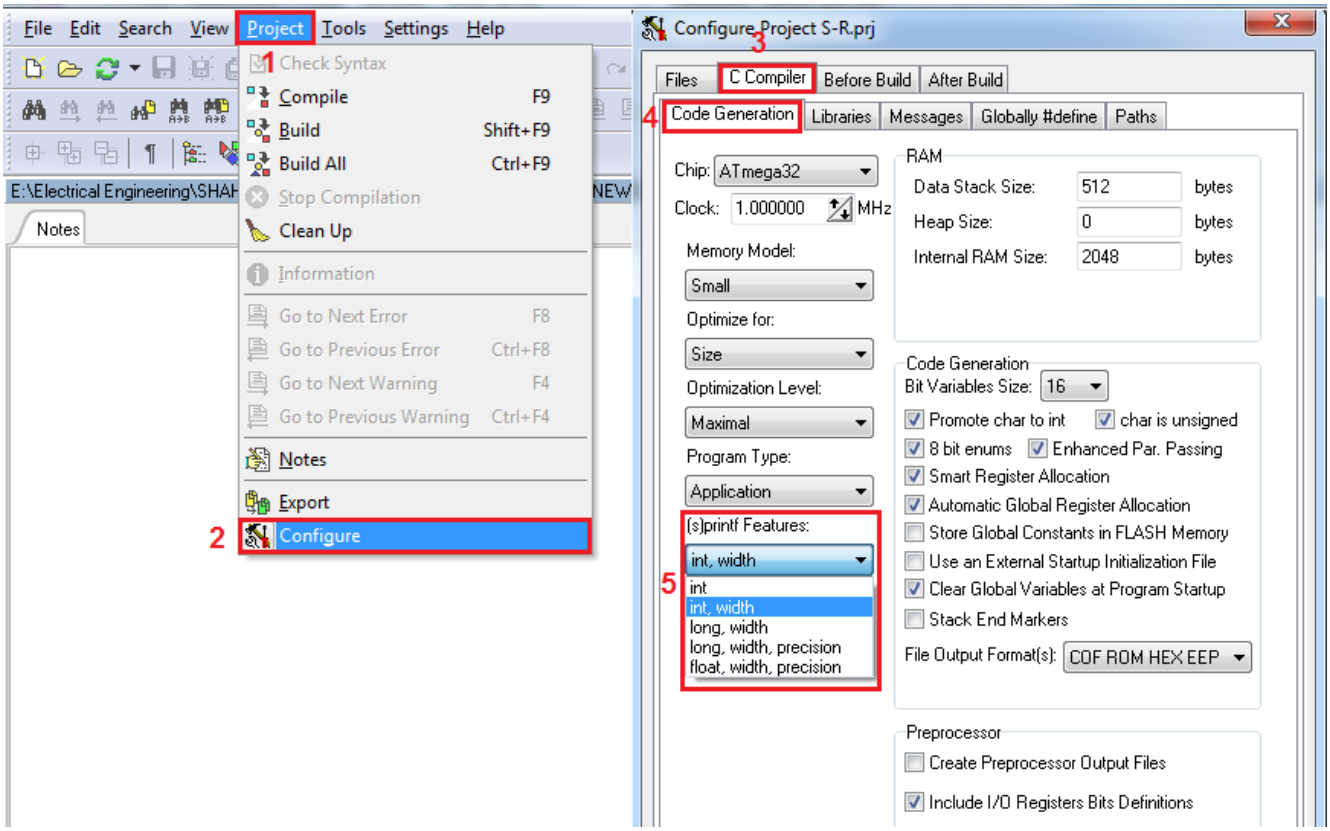
```
printf("A=%7.4f",i);
```

در مثال فوق بعد از =A به اندازه ۷ کاراکتر قرار میگیرد که حداکثر ۴ تا از آن برای اعشار و ۳ تا از آن برای قسمت صحیح عدد است . در صورتی که طول متغیر i کمتر از ۷ کاراکتر را اشغال کند به همان تعداد کاراکتر خالی سمت چپ عدد قرار می گیرد.

نکته : برای ارسال عددی از نوع Long از کاراکتر ویژه 'l' استفاده می شود . این کاراکتر ویژه را میتوان بعد از کاراکتر ٪ و قبل از کاراکترهای فرمت c ، d، u و x به کار برد . مثال:

```
printf("A=%lu",A);
```

نکته مهم : با توجه به اینکه تابع printf حجم زیادی از حافظه برنامه را به خود اختصاص می دهد ، در نرم افزار CodeVision برای استفاده بهینه از این تابع کاربر می تواند تنظیمات نوع عملکرد تابع printf را بر حسب نیاز تغییر دهد . برای این کار از منوی Project گزینه ی Configure Project را انتخاب کرده و در سربرگ C Compilerبه زیر برگ Code Generation بروید . همانطور که در شکل زیر نیز مشاهده می کنید در قسمت printf featureمی توان طول ، دقت و نوع تابع printfیا sprintf را در صورت وجود معین کرد.



6. تابع scanf

می تواند رشته یا متغیر را از ورودی با یک فرمت مشخص دریافت و در یک آرایه ذخیره کند. مثال:

```
scanf("Temp=%d",A);
```

در مثال فوق مقدار مورد نظر را به صورت intدریافت کرده و به صورت فرمت مشخص شده در متغیر A ذخیره می کند. برای استفاده بهینه از این تابع نیز در نرم افزار CodeVision در همان قسمت Code Generation بخش scanf featureمیتوان نوع و طول ورودی را تنظیم کرد.

7. تابع sprintf

این تابع همانند printf عمل می کند با این تفاوت که خروجی آن به جای ارسال توسط واحد USART در یک آرایه که در آرگومان اول تابع مشخص می شود ، ذخیره می شود . مثال:

```
int data=10;\nchar s[10];

sprintf(s,"Tha data is :%d",data);
```

8. تابع sscanf

این تابع همانند scanf عمل می کند با این تفاوت که ورودی آن به جای دریافت از واحد USART از یک آرایه که در آرگومان اول تابع مشخص می شود ، گرفته می شود . مثال:

```
char a[10];\nsscanf(s,"%s",a);
```

پس از اجرای دستور فوق محتویات آرایه s به آرایه a منتقل می شود.

توابع پرکاربرد کتابخانه string.h برای کار با رشته ها

در صورتی که با رشته ها و آرایه های رشته ای کار می کنید ، میتوانید با اضافه کردن هدر فایل string.h از توابع مفید این کتابخانه در برنامه استفاده نمایید . در پروژه هایی مانند راه اندازی ماژول های GSM ، GPSو Bluetouth به برخی از آنها نیاز خواهید داشت.

1. تابع strcmp

تابع strcmp کاراکترهای دو رشته را باهم مقایسه کرده و یک عدد صحیح نسبت به میزان تفاوت دو عدد با هم به خروجی تابع برگردانده می شود.

strcmp(str1,str2);

- اگر str1<str2 باشدمقدار برگردانده شده عددی کوچکتر از صفر خواهد بود.
- اگر str1=str2 باشد مقدار برگردانده شده برابر صفر خواهد بود.
- و اگر str1>str2 باشد مقدار برگردانده شده عددی بزرگتر از صفر خواهد بود.

2. تابع strcpy

به علت اینکه مقدار دادن به متغیر رشته ای بطور مستقیم امکان پذیر نیست ، از طریق این تابع رشته ای در رشته ی دیگر قرار می گیرد . مثال:

strcpy(name,"ALI");

3. تابع strncpy

تابع strncpy تعداد مشخصی از کاراکتر های یک رشته را در رشته ی دیگر کپی می کند. مثال:

strncpy(str1,str2,n);

در مثال فوق str1 رشته ای است که به تعداد n کاراکتر از کاراکترهای str2 در آن کپی می شود.

4. تابع strlwr

رشته ای را به عنوان ورودی پذیرفته و کلیه ی حروف بزرگ آن را به کوچک تبدیل می کند.

5. تابع struper

رشته ای را به عنوان ورودی پذیرفته و کلیه ی حروف کوچک آن را به بزرگ تبدیل می کند.

6. تابع strlen

از این تابع برای تعیین طول یک رشته مورد استفاده قرار می گیرد.

7. تابع strrev

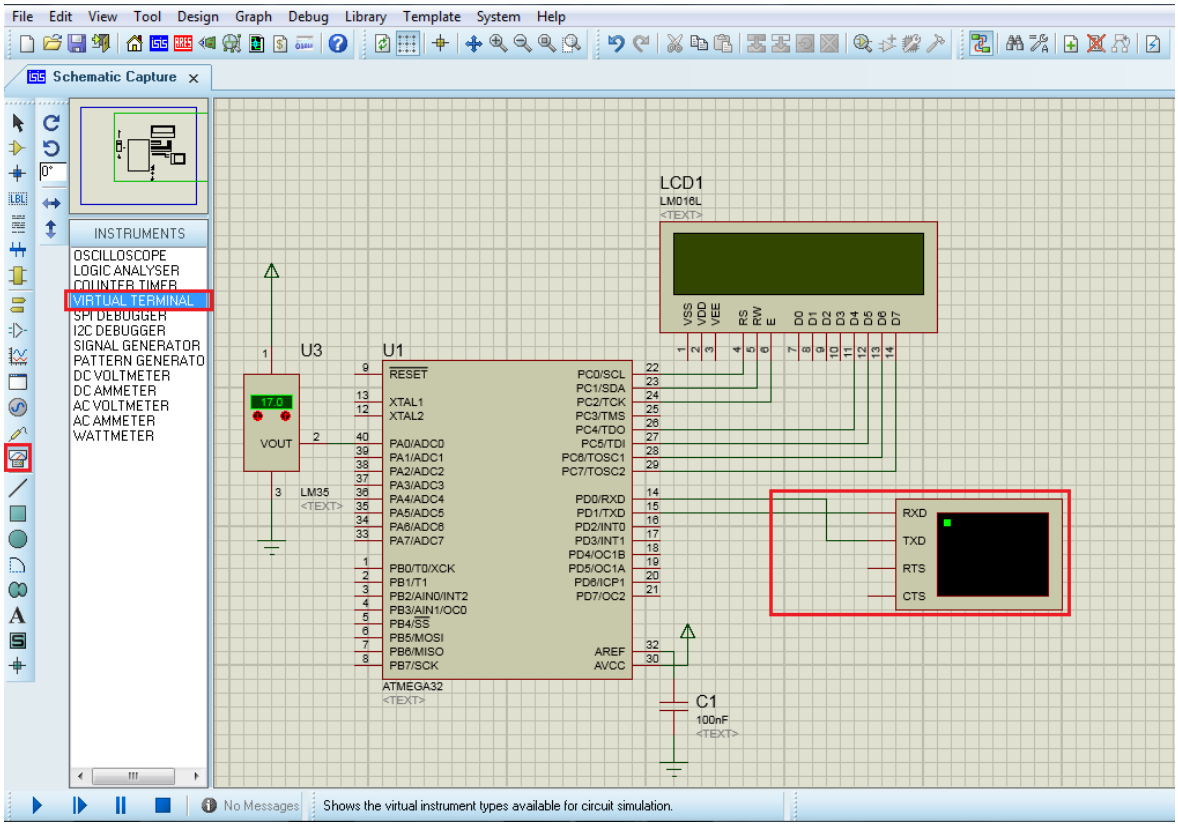
این تابع کاراکترهای یک رشته را معکوس می کند یعنی کاراکتر ابتدایی را به انتهای آن رشته منتقل و برای کاراکترهای بعدی نیز عمل معکوس کردن را انجام می دهد.

مثال عملی شماره ۷ : برنامه ای با Atmega32 بنویسید که دارای یک LCD کاراکتری ۲ در ۱۶ و یک سنسور LM35باشد . میکروکنترلر از طریق پورت سریال به کامپیوتر وصل است . کاربر میتواند با هر بار زدن کلید b از صفحه کلید کامپیوتر ، دمای سنسور را روی LCD به نمایش درآورده و به کامپیوتر ارسال و نمایش دهد . برنامه را ابتدا در پروتئوس شبیه سازی کرده و سپس پیاده سازی نمایید.

حل:

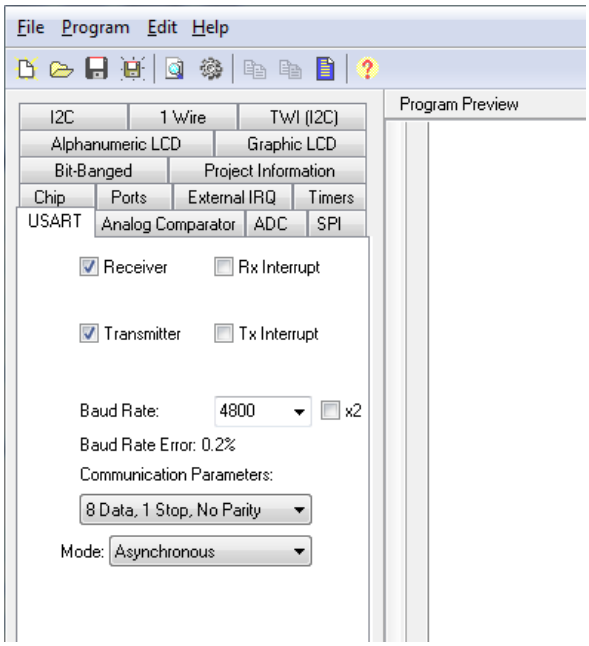
مرحله اول : پیاده سازی سخت افزار درProteus

پس از قراردادن Atmega32 و LM35 و LM016L در نرم افزار پروتئوس ، همانطور که در شکل زیر نیز مشاهده می کنید ، برای شبیه سازی اتصال میکرو به کامپیوتر از قطعه Virtual Terminal استفاده می شود . برای قرار دادن این قطعه از نوار ابزار سمت چپ instrument را انتخاب کرده و از منو Virtual Terminal را انتخاب کرده و به صورت ضربدری به میکرو متصل می کنیم.



مرحله دوم : پیاده سازی نرم افزار توسط کدویژن

پس از انجام تنظیمات کدویزارد مربوط به سربرگ های port ، chip، Alphanumeric LCDو ADC به همان صورت توضیح داده شده در مثال ۶ به سربرگ USART رفته و تنظیمات مربوطه را به صورت شکل زیر انجام دهید.



پس از تولید کد اولیه توسط کدویزارد و حذف کدها و کامنت های اضافی نوبت به برنامه نویسی پروژه می رسد . برنامه به صورتی نوشته شده است که ابتدا میکرو منتظر می ماند تا کلید b از صفحه کلید کامپیوتر زده شود سپس از adc مقدار سنسور دما را خوانده و ابتدا روی lcd نمایش داده و سپس به کامپیوتر ارسال می نماید.

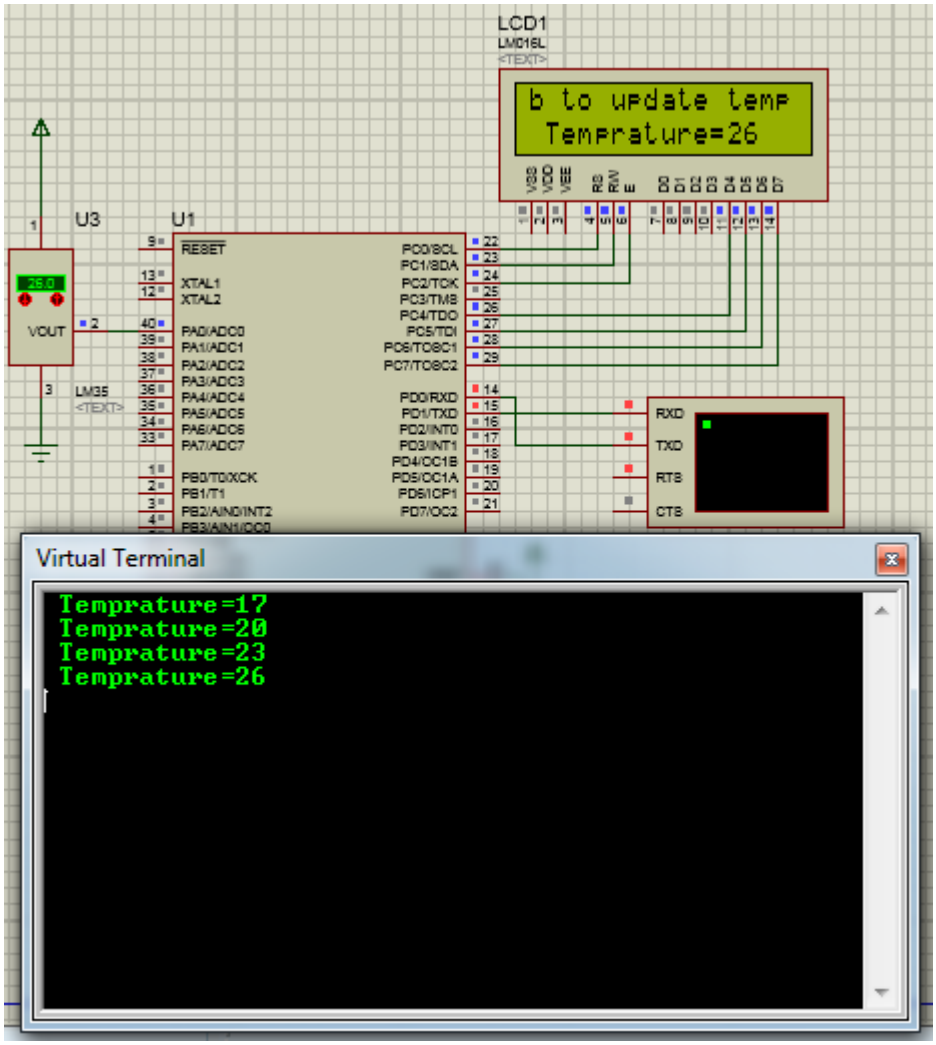
```
#include <mega32.h>

#include <delay.h>

#include <alcd.h>

#include <stdio.h>

#include <stdlib.h>

#define ADC_VREF_TYPE 0xC0

int a;

char s[16],c='b';

unsigned int read_adc(unsigned char adc_input){

ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);

delay_us(10);

ADCSRA|=0x40;

while ((ADCSRA & 0x10)==0);

ADCSRA|=0x10;

return ADCW;

}

void main(void){

DDRC=0xF7;

PORTC=0x00;
```

```
// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud Rate: 2400
UCSRA=0x00;
UCSRB=0x18;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x19;
// ADC initialization
// ADC Clock frequency: 125.000 kHz
// ADC Voltage Reference: Int., cap. on AREF
ADMUX=ADC_VREF_TYPE & 0xff;
ADCSRA=0x83;
lcd_init(16);
lcd_clear();
lcd_gotoxy(0,0);
lcd_putsf("b to update temp");
while(1){
c=getchar();
a=read_adc(0);
```

```
sprintf(s," Temprature=%d  ",a/4);
lcd_gotoxy(0,1);
lcd_puts(s);
if(c=='b'){
 puts(s);
 putchar(13);
 putchar(10);
}
}
}
```

مرحله سوم : شبیه سازی برنامه در پروتئوس

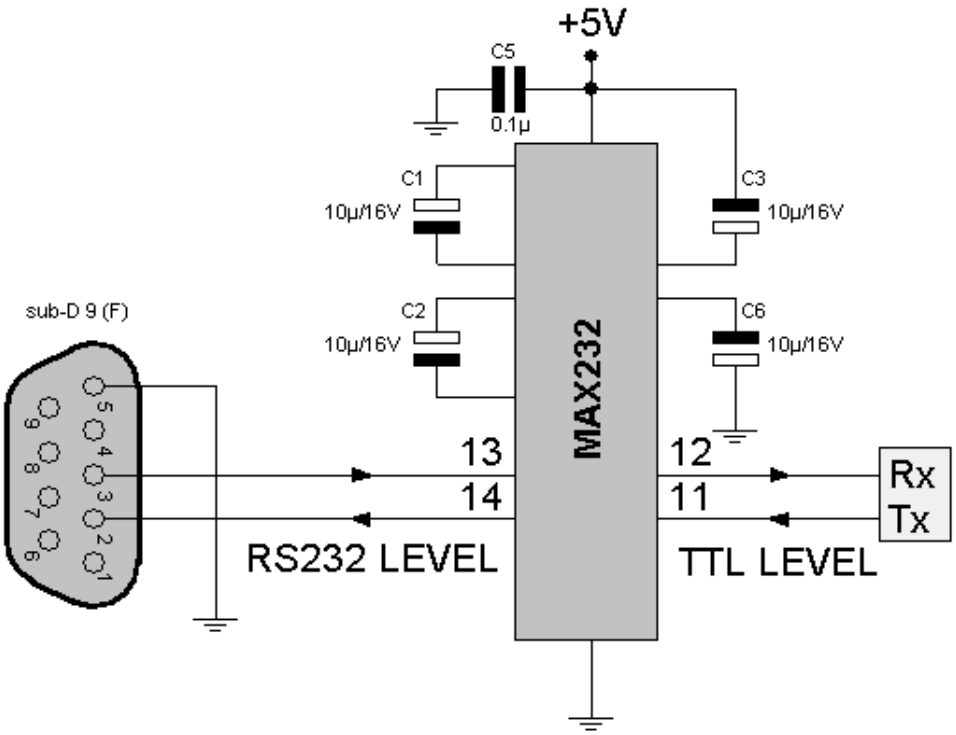
قبل از شروع شبیه سازی باید نرخ ارسال/دریافت (Baud Rate) ابزار ترمینال با نرخ ارسال/دریافت که در کدویزارد تنظیم کرده ایم یکسان باشد تا به درستی کار کند . برای این منظور روی Virtual Terminal دابل کلیک کرده و در پنجره باز شده قسمت Baud Rate را روی ۴۸۰۰ قرار می دهیم.



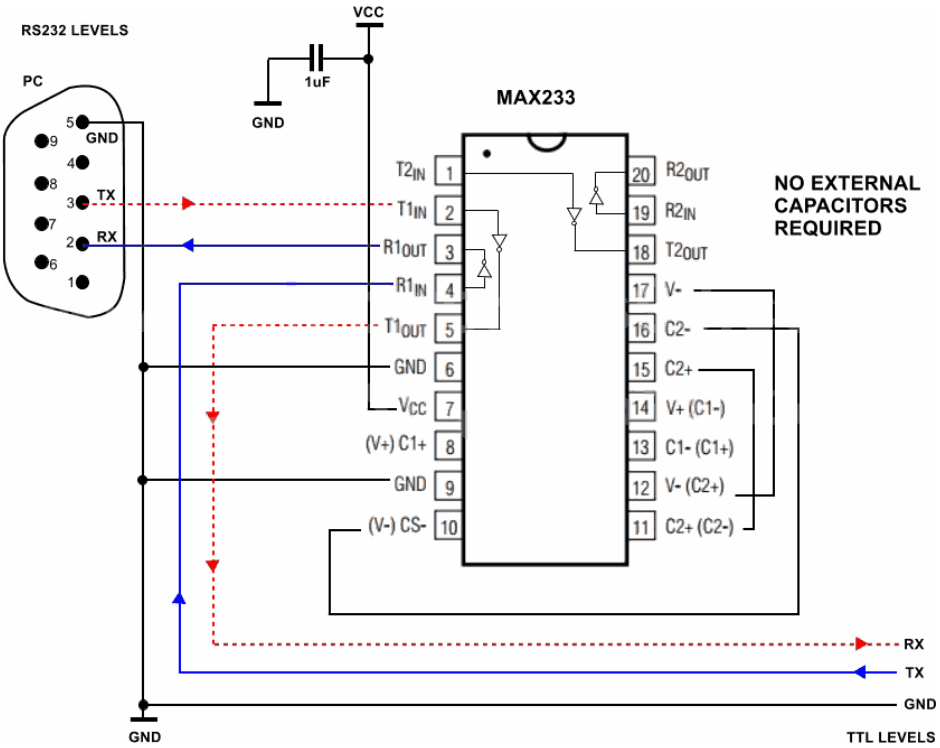
مرحله چهارم : پیاده سازی پروژه

در هنگام پیاده سازی پروژه می بایست نکات زیر را در خصوص ارتباط سریال رعایت نمود:

- از آن جایی که برای اتصال میکرو به کامپیوتر از استاندارد RS232 استفاده می شود نیاز است که از یک عدد آی سی Max232استفاده گردد . نحوه اتصال این آی سی به میکروکنترلر را در شکل زیر مشاهده می کنید.

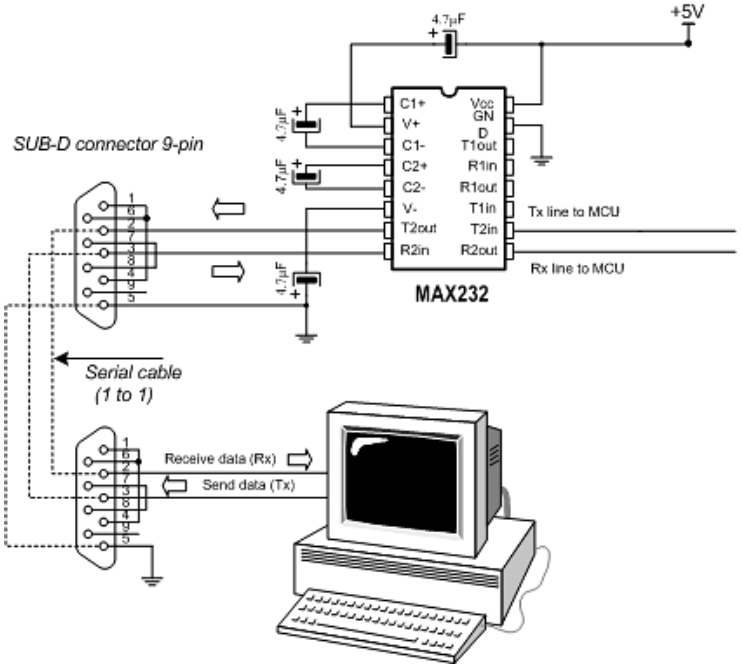


- به علت اینکه برای راه اندازی تراشه MAX232شما احتیاج به استفاده از چندین خازن دارید ، میتوان از تراشه MAX233 استفاده کرد که در آن احتیاج به استفاده از هیچ خازنی ندارید .نحوه ارتباط این آی سی را در شکل زیر مشاهده می کنید.



- بعد از اتصال میکرو به پورتRS232 ، نیاز به یک کابل ارتباطی سریال (DB 9) برای اتصال میکرو به پورت سریال PC دارید که در شکل زیر آن را مشاهده می کنید.

- پس از بعد از پروگرام کردن و کامل کردن اتصالات ، برای شروع ارتباط کامپیوتر با میکروکنترلر نیاز به یک اینترفیس نرم افزاری در PC داریم که از طریق آن با پورت سریال و میکرو ارتباط برقرار کنیم . نرم افزار های متعددی به نام Terminal برای این منظور طراحی شده اند که میتوان از همه آنها استفاده نمود . در نرم افزار کدویژن ابزاری برای این منظور تعبیه شده است که در منوی Tools قرار دارد . بنابراین بعد از اتصال کابل سریال به کامپیوتر و وصل کردن منبع تغذیه به میکرو و روشن نمودن آن ، از منوی Setting در نرم افزار کدویژن Terminal را انتخاب کرده و تنظیمات مربوط به COM پورتی که میکرو به آن وصل است و قاب دیتا و نرخ ارسال/دریافت را نیز مشخص می کنیم . سپس از طریق منوی Tools وارد ترمینال می شویم و حالا میتوان برای میکرو کاراکتری را فرستاد یا هر آنچه میکرو ارسال می کند را مشاهده کرد.

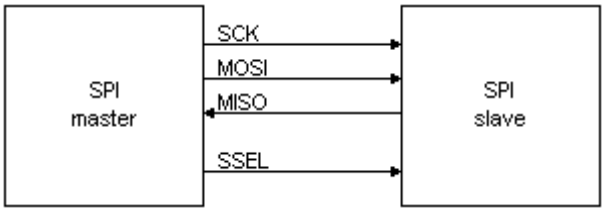


شما میتوانید سورس کامل این پروژه در نرم افزار کدویژن و پروتئوس را از لینک زیر دریافت و نتایج را مشاهده کنید.

دانلود سورس مثال عملی شماره ۷

معرفی و تشریح واحد ارتباط سریال SPI

SPIیا serial peripheral interface یک ارتباط سریع سریال است که بوسیله ی شرکت موتورلا طراحی شده است SPI .تنها به صورت full-Duplex عمل کرده و امکان ارسال و دریافت همزمان را دارد. از واسط SPI برای انتقال اطلاعات با سرعت انتقال بالا و برای فواصل کوتاه استفاده می کنند. در باس SPI ارتباط دو وسیله به صورت Master/Slaveاست .آغاز کننده ی ارتباط همیشه Master است و فقط Master است که می تواند شروع به انتقال کند و Slave باید منتظر دریافت اطلاعات بماند. برای استفاده از واحد SPI سیم بندی زیر بین Master و Slaveمورد استفاده قرار می گیرد.



نتیجه : ارتباط سریال بوسیله پروتکل SPI دارای ۴ پایه است که به صورت شکل فوق به هم متصل می شود.

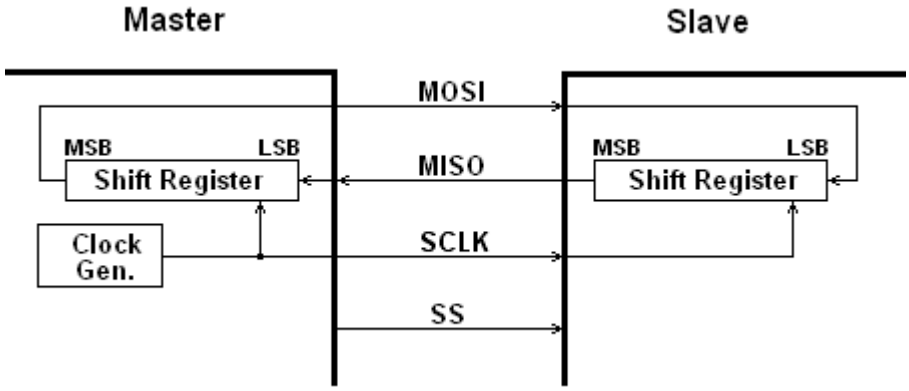
MISO: Master Input Slave Output

MOSI: Master Output Salve Input

SCK: Serial Clock

SSEL: Slave Select

طرز کار واحدSPI



سیستم دارای دو بخش Master و Slave است. در بخش Master سیستم دارای یک شیفت رجیستر ۸ بیتی و مولد پالس است و بخش slave فقط شامل یک شیفت رجیستر هشت بیتی است. کلاک این دو رجیستر از واحد تولید کلاک در وسیله ی Master تامین می شود. با اعمال هر پالس کلاک به طور هم زمان یک بیت از شیفت رجیستر Masterخارج شده و وارد شیفت رجیستر Slave می شود. و یک بیت نیز از شیفت رجیستر Slave وارد شیفت رجیستر Master خواهد شد. پایه ی SS فعال ساز شیفت رجیستر Slave است و تا زمانی که بوسیله ی Master صفر (**Low**) نشود بیتی منتقل نخواهد شد.

در صورتی که محتوای این رجیستر ها ۸ بیت شیفت پیدا کند محتویات رجیستر داده ی Master و Slave با یکدیگر تعویض می شود. یعنی به صورت چرخشی (Shift) محتوای master با slave عوض می شود یعنی محتویات slave به master منتقل شده و در مقابل محتوای master نیز با slave تعویض خواهد شد.

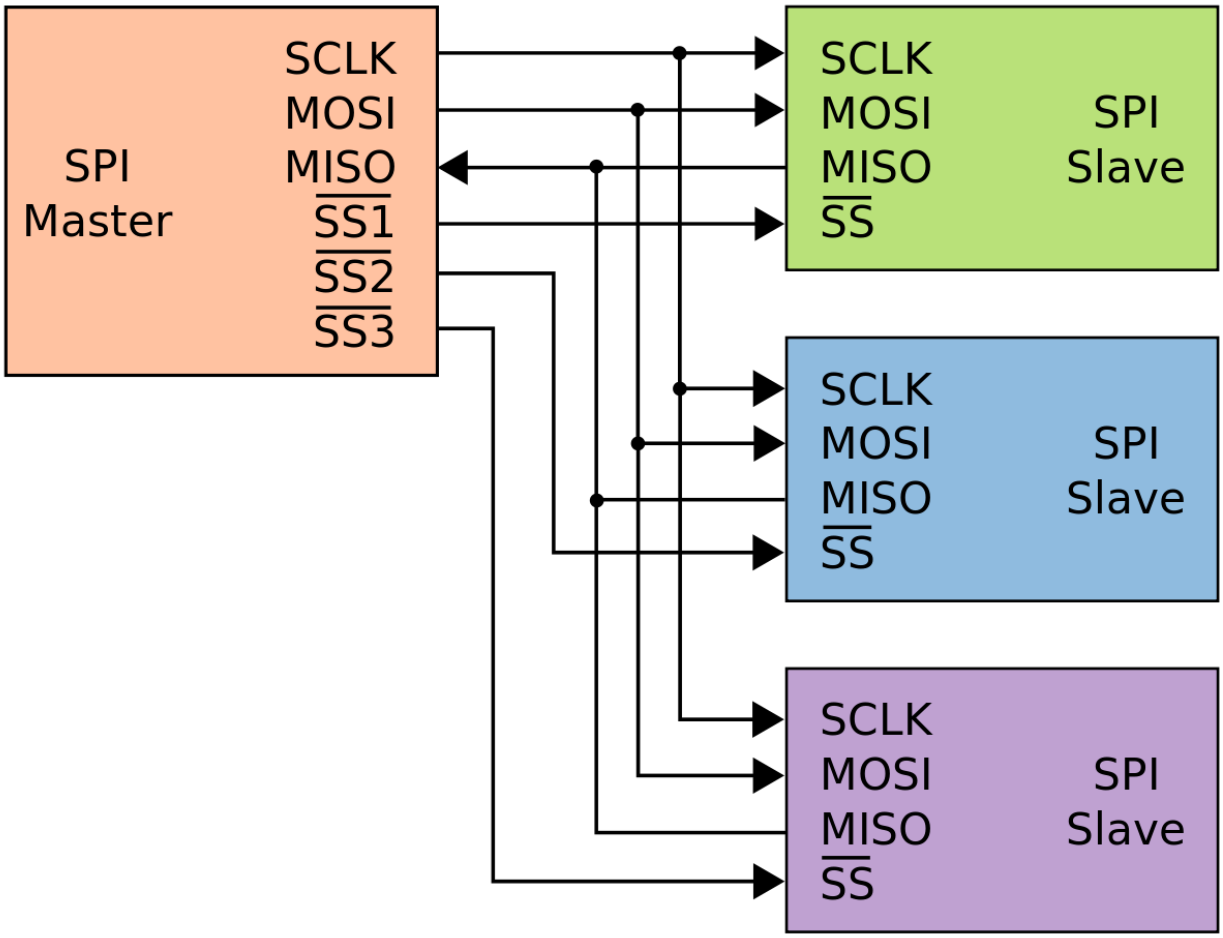
خصوصیات واحد SPI در میکروکنترلرهای AVR

- Full-Duplex ارسال اطلاعات به صورت سنکرون توسط ۳ سیم
- عملکرد در حالت های Master وSlave
- اولویت در ارسال بیت ابتدایی LSB یاMSB
- قابلیت تنظیم سرعت انتقال دیتا
- قابلیت ایجاد وقفه در پایان ارسال
- بیدار شدن خودکار از حالت بیکاری(Idle)
- امکان دو برابر کردن سرعت ارسال (در برخی از AVR ها)

شبکه بندی چند Slave توسط یک Master در پروتکل SPI

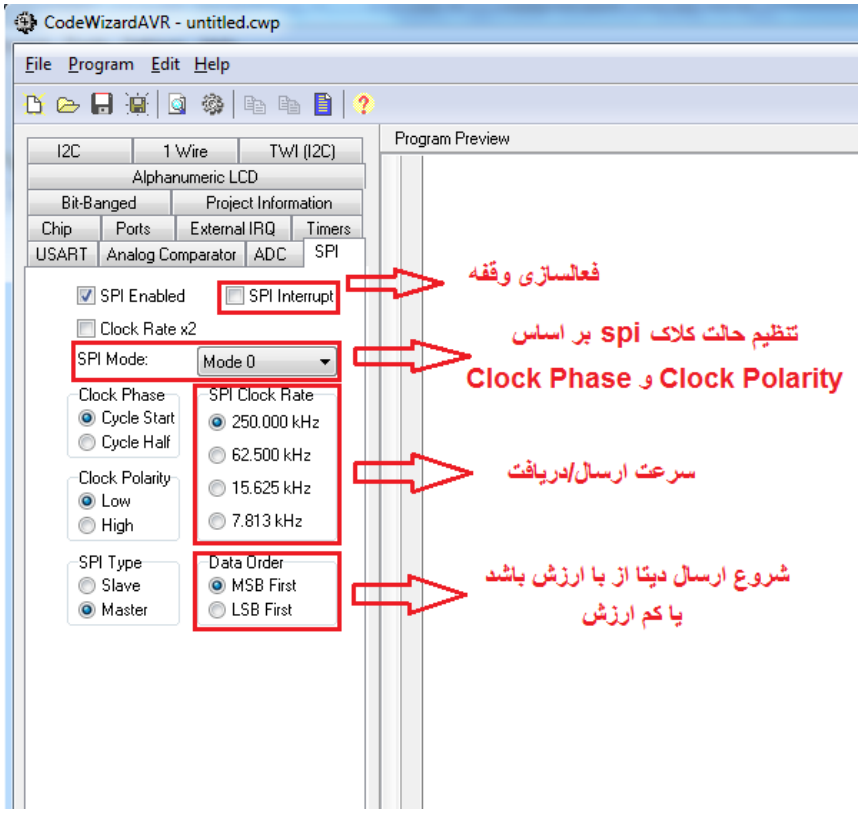
با استفاده از پروتکل SPI میتوان از ارتباط یک Master با چندین Slave استفاده کرد . همانطور که در شکل زیر نیز مشاهده می کنید ، هر سه سیم SCK ، MISOو MOSI به تمامی Slave های درون شبکه به یکدیگر متصل هستند اما Master توسط Slave Select انتخاب می کند که با کدام Slave ارتباط برقرار کند به طوری که Masterپین SS در Slave مورد نظر را Low کرده و پین SS مابقی Slave ها را High می کند . با انجام این کار بین Master و یکی از Slave ها ارتباط SPI برقرار می شود و دیتای آن دو مبادله می گردد.

نکته : پین هایSS1 ، SS2و SS3 در Master میتواند هر پایه ای از پورت های خروجی میکروکنترلر باشد . یعنی پایه SS در Master بدون استفاده است و برای انتخاب Slave از سه پایه دیگر میکرو استفاده می گردد.



تنظیمات واحد SPI در کدویزارد CodeWizard

پس از رفتن به سربرگ Spi و فعال کردن واحد Spi ، تنظیمات چگونگی ارتباط Spi به راحتی و با انتخاب گزینه های موجود صورت می گیرد . در صورت فعالسازی SPI interrupt وقفه داخلی مربوط به این واحد فعال می شود که پس از تولید کد میتوان برنامه مورد نظر را داخل تابع سابروتین وقفه اضافه شده نوشت . در قسمت SPI Clock Rateسرعت ارسال/دریافت دیتا توسط واحد spi مشخص می شود و با فعال کردن Clock Rate X2 نیز میتوان سرعت کلاک انتخاب شده در SPI Clock Rate را دوبرابر کرد . در قسمت SPI Type نوع Master یا Slave بودن میکرو مشخص می گردد . در بخش Clock Phase موقعیت لبه پالس کلاک نسبت به بیت های داده را مشخص می کند به طوری که وقتی روی Cycle Start قرار گیرد عمل نمونه برداری در لبه بالارونده انجام گرفته و عمل شیفت در لبه پایین رونده اتفاق می افتد و در حالت Cycle Half برعکس حالت فوق است یعنی عمل نمونه برداری در لبه پایین رونده انجام و شیفت در لبه بالا رونده صورت می گیرد . در بخش Clock Polarity مشخص می شود که پالس کلاک در حالت بیکاری (Idle) در منطق Low یا High قرار گیرد . براساس تنظیمات قسمت های Clock Phaseو Clock Polarity چهار حالت مختلف برای ارسال/دریافت دیتا در واحد SPI بوجود می آید که به آنها Mode های واحد Spiگویند . دقت شود که مد Master و Slave ها باید یکسان باشند . در بخش Data Orderنیز مشخص می شود که اولین بیت ارسالی در شیفت رجیستر بیت با ارزش (بیت هشتم یا MSB) باشد یا بیت کم ارزش (بیت اول یا LSB) باشد.



پس از تولید کد توسط برنامه کدویزارد مشاهده می شود کتابخانه spi.h به برنامه اضافه می شود . درون این کتابخانه توابع کار با واحد SPI وجود دارد که در طول برنامه نویسی می توان از آنها برحسب نیاز استفاده کرد . مهم ترین تابع که در برنامه ها از آن استفاده می شود تابع spi است که دارای ۸ بیت ورودی و ۸ بیت خروجی است . با فراخوانی این تابع در برنامه به طور همزمان ۸ بیتی که در ورودی تابع قرار دارد به پورت spi ارسال می شود و ۸ بیت دیگر خروجی تابع نیز از پورت spi دریافت می شود و در یک متغیر از نوع unsigned char قرار می گیرد . مثال:

```
unsigned char send=0x45;

unsigned char recieve;

recieve=spi(send);
```

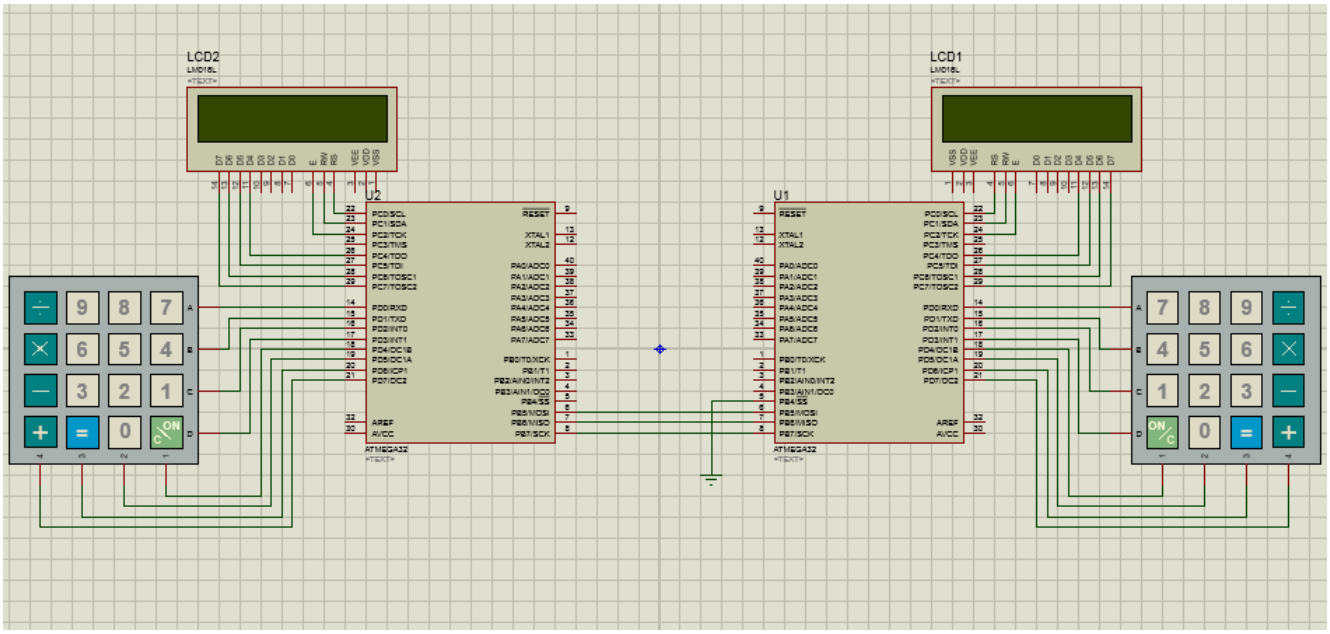
توضیح : در خط سوم مقدار 0x45 از طریق رابط سریال spi به میکروکنترلر Master/Slave ارسال می شود و هر آنچه که از آن میکروکنترلر دریافت شده است درون متغیر recieve ریخته می شود.

مثال عملی شماره ۸ : دو میکروکنترلر Atmega32 را از طریق پورت Spi به یکدیگر متصل کنید . به هر یک از میکروکنترلرها یک LCD کاراکتری ۲ در ۱۶ و یک صفحه کلید ۴ در ۴ متصل کرده و سپس برنامه ای بنویسید که با زدن کلیدی از هر میکروکنترلر ، کاراکتر مورد نظر به میکروکنترلر دیگر ارسال شده و روی LCD آن نمایش داده شود.

حل:

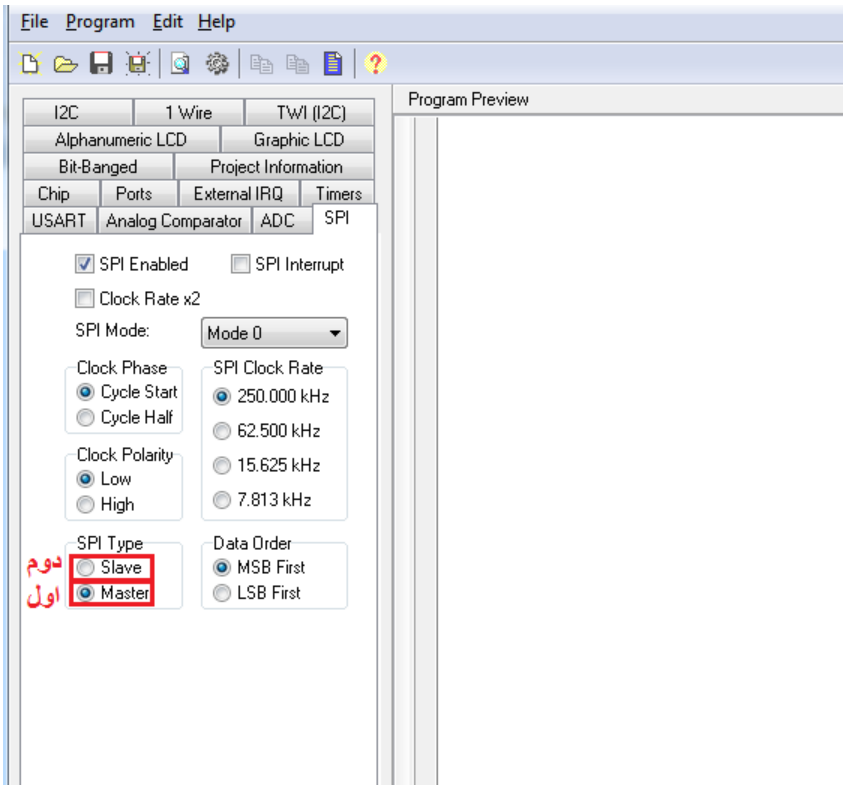
مرحله اول : پیاده سازی سخت افزار درProteus

بعد از قرار دادن المانها و سیم کشی آنها ، ارتباط spi میان دو میکرو را به صورت زیر برقرار کرده سپس یکی از آنها را به دلخواه به عنوان Master و دیگری را Slave انتخاب کرده و Slave Select را به زمین متصل می کنیم تا همواره فعال گردد.



مرحله دوم : پیاده سازی نرم افزار توسط codewizard

در این مرحله می بایست دوبار از نرم افزار کد ویزارد استفاده کرده و دو برنامه یکی برای Master و دیگری برای Slave تولید کرد . پس از تنظیم سربرگ های PORT و Alphanumeric LCD در کدویزارد به همان صورت مثال شماره ۴ به سراغ سربرگ SPi می رویم و تنظیمات زیر را انجام می دهیم.



بعد از تولید کد و حذف کامنت های اضافی به سراغ نوشتن برنامه های Master و Slave می رویم .

تذکر : در هنگام کار با رابط SPI باید سعی کنیم برنامه های Master و Slave تا جای ممکن شبیه به هم باشد .

نکته مهم : محل قرار گیری تابع delay_ms در برنامه و نیز مقدار آن در برنامه بسیار اهمیت دارد و کم یا زیاد بودن آن باعث ایجاد باگ و مشکل می گردد .

برنامه Master

```
#include <mega32.h>
#include <delay.h>
#include <stdio.h>
#include <alcd.h>
#include <spi.h>
char s[2],in=20,key=20;
void start_lcd(void){
char txt1[] ="  SPI    init  ";
char txt2[] =" With  Atmega32";
lcd_init(16);
lcd_clear();
lcd_gotoxy(0,0);
lcd_puts(txt1);
lcd_gotoxy(0,1);
lcd_puts(txt2);
delay_ms(1000);
lcd_clear();
```

```
}
void keyboard(void)
{
 //---- ROW1 ----
 PORTD.4=0;
 delay_ms(2);
 if(PIND.0==0) key='7';
 if(PIND.1==0) key='4';
 if(PIND.2==0) key='1';
 if(PIND.3==0) key=12;
 PORTD.4=1;
 //---- ROW2 ----
 PORTD.5=0;
 delay_ms(2);
 if(PIND.0==0) key='8';
 if(PIND.1==0) key='5';
 if(PIND.2==0) key='2';
 if(PIND.3==0) key='0';
 PORTD.5=1;
 //---- ROW3 ----
 PORTD.6=0;
 delay_ms(2);
 if(PIND.0==0) key='9';
```

```
 if(PIND.1==0) key='6';
 if(PIND.2==0) key='3';
 if(PIND.3==0) key='=';
 PORTD.6=1;
 //---- ROW4 ----
 PORTD.7=0;
 delay_ms(2);
 if(PIND.0==0) key='/';
 if(PIND.1==0) key='*';
 if(PIND.2==0) key='-';
 if(PIND.3==0) key='+';
 PORTD.7=1;
 }
void main(void)
{
PORTA=0x00;
DDRA=0x00;
PORTB=0x00;
DDRB=0xB0;
PORTC=0x00;
DDRC=0xF7;
PORTD=0xFF;
DDRD=0xF0;
```

```
// SPI initialization
// SPI Type: Master
// SPI Clock Rate: 250.000 kHz
// SPI Clock Phase: Cycle Start
// SPI Clock Polarity: Low
// SPI Data Order: MSB First
SPCR=0x50;
SPSR=0x00;
start_lcd();
while (1){
   keyboard();
   delay_ms(200);
   in=spi(key);
   key=20;//default value
   if(in==12) lcd_clear();
   else if(in!=20){
         sprintf(s,"%c",in);
         lcd_puts(s);
         in=20; //default value
         }
     }
}
```

برنامه Slave

```
#include <mega32.h>
#include <delay.h>
#include <stdio.h>
#include <alcd.h>
#include <spi.h>
char s[2],in=20,key=20;
void start_lcd(void){
char txt1[] ="  SPI    init  ";
char txt2[] =" With  Atmega32";
lcd_init(16);
lcd_clear();
lcd_gotoxy(0,0);
lcd_puts(txt1);
lcd_gotoxy(0,1);
lcd_puts(txt2);
delay_ms(1000);
lcd_clear();
}
void keyboard(void)
{
  //---- ROW1 ----
```

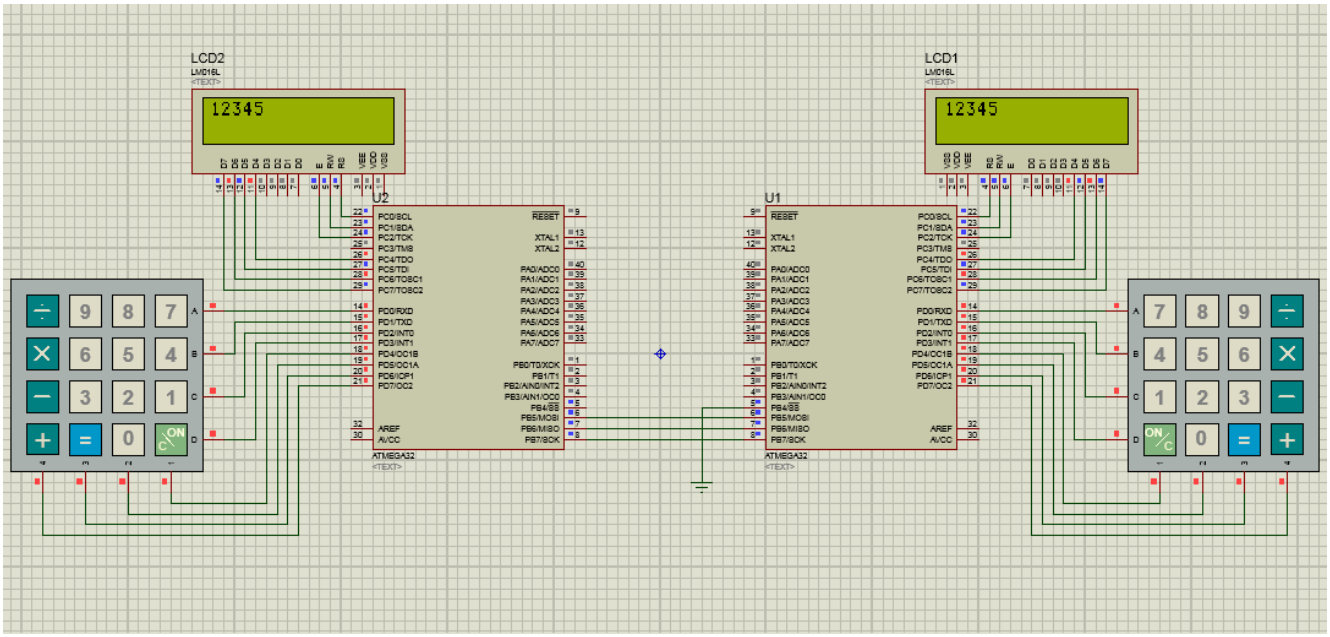
```
PORTD.4=0;
delay_ms(2);
if(PIND.0==0) key='7';
if(PIND.1==0) key='4';
if(PIND.2==0) key='1';
if(PIND.3==0) key=12;
PORTD.4=1;
//---- ROW2 ----
PORTD.5=0;
delay_ms(2);
if(PIND.0==0) key='8';
if(PIND.1==0) key='5';
if(PIND.2==0) key='2';
if(PIND.3==0) key='0';
PORTD.5=1;
//---- ROW3 ----
PORTD.6=0;
delay_ms(2);
if(PIND.0==0) key='9';
if(PIND.1==0) key='6';
if(PIND.2==0) key='3';
if(PIND.3==0) key='=';
PORTD.6=1;
```

```
 //---- ROW4 ----
 PORTD.7=0;
 delay_ms(2);
 if(PIND.0==0) key='/';
 if(PIND.1==0) key='*';
 if(PIND.2==0) key='-';
 if(PIND.3==0) key='+';
 PORTD.7=1;
 }
void main(void)
{
PORTA=0x00;
DDRA=0x00;
PORTB=0x00;
DDRB=0x40;
PORTC=0x00;
DDRC=0xF7;
PORTD=0xFF;
DDRD=0xF0;
// SPI initialization
// SPI Type: Slave
// SPI Clock Phase: Cycle Start
// SPI Clock Polarity: Low
```

```
// SPI Data Order: MSB First
SPCR=0x40;
SPSR=0x00;
start_lcd();
while (1){
   keyboard();
   delay_ms(200);
   in=spi(key);
   key=20;//default value
   if(in==12) lcd_clear();
   else if(in!=20){
         sprintf(s,"%c",in);
         lcd_puts(s);
         in=20; //default value
         }
     }
}
```

توضیح : همانطور که مشاهده می کنید برنامه Master و Slave تنها در یک خط (مقدار دهی رجیستر SPCR در تابع main) با هم تفاوت دارند و این نوع برنامه نویسی باعث کمتر شدن مشکلات در هنگام پیاده سازی می گردد. همچنین در کل برنامه تنها از یکبار استفاده از تابع delay و به اندازه ۲۰۰ میلی ثانیه کفایت شده است که باعث بهبود عملکرد میکرو می گردد .

مرحله سوم : شبیه سازی برنامه در پروتئوس



مرحله چهارم : پیاده سازی پروژه

نکته خاصی در این قسمت وجود ندارد و اگر همه اتصالات و خصوصا برنامه نویسی صحیح باشد برنامه به درستی کار می کند.

شما میتوانید سورس کامل این پروژه در نرم افزار کدویژن و پروتئوس را از لینک زیر دریافت و نتایج را مشاهده کنید.

دانلود سورس مثال عملی شماره ۸

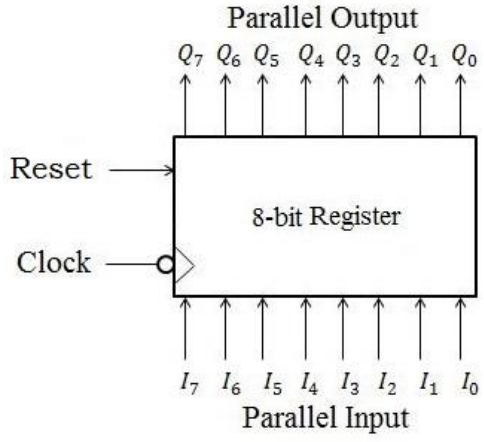
ادامه فصل هشتم : آموزش واحدهای تایمر/کانتر میکروکنترلر Atmega32

مقدمه:

یکی از مهمترین واحدهای میکروکنترلر **واحد تایمر/کانتر** می باشد که در اکثر پروژه های مهم وجود آن ضروری است . این واحد از نظر سخت افزاری متشکل از یک شمارنده اصلی و چندین رجیستر برای تنظیمات می باشد به طوری که با اعمال تنظیمات متفاوت چندین کاربرد مختلف از این سخت افزار خاص می شود . از مهمترین کاربردهای این سخت افزار میتوان به تایمر (زمان سنج) ، کانتر (شمارنده) ، Real Time Clock (زمان سنج حقیقی) و PWM (مدولاسیون عرض پالس) اشاره کرد . در این فصل ابتدا به تشریح مفاهیم مربوطه می پردازیم و سپس مشابه بخش های قبلی به تشریح سخت افزاری و آموزش کدویزارد به همراه پروژه مربوطه خواهیم پرداخت.

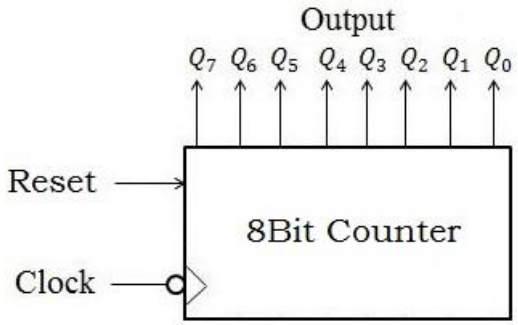
رجیستر چیست ؟

رجیستر ها نوعی از حافظه های موقت هستند که معمولا ۸ بیتی ، ۱۶ بیتی ، ۳۲ بیتی یا ۶۴ بیتی هستند . بدین معنی که توانایی ذخیره کردن همان تعداد بیت را دارند . در میکروکنترلرهای AVR همه رجیستر ها ۸ بیتی هستند یعنی یک بایت را ذخیره می کنند . هر رجیستر به جز ورودی و خروجی دارای سیگنال کلاک و ریست نیز می باشد که به محض آمدن لبه سیگنال کلاک ورودی به خروجی منتقل می شود یا به عبارت دیگر ورودی تنها در زمان آمدن لبه سیگنال کلاک ذخیره می شود . شکل زیر یک رجیستر ۸ بیتی را نشان می دهد . در صورتی که ریست فعال شود تمام ۸ بیت در خروجی ۰ می گردد و در صورتی که ریست غیر فعال بوده باشد و لبه سیگنال کلاک بیاید ، ۸ بیت ورودی عینا در رجیستر ذخیره شده و سپس به خروجی می رود.



کانتر یا شمارنده چیست ؟

شمارنده ها نیز نوعی حافظه هستند که می توانند هر تعداد بیت دلخواه پهنا داشته باشند . وظیفه شمارنده ، شمردن تعداد پالس های سیگنال ورودی کلاک است . بنابراین یک شمارنده از یک ورودی کلاک و یک ورودی ریست و تعدادی خروجی (n تا) تشکیل شده است . خروجی شمارنده با آمدن سیگنال کلاک ورودی یک واحد افزایش پیدا می کند . بنابراین هنگامی که ریست فعال باشد تمام خروجی ها ۰ می شود سپس با آمدن هر پالس کلاک یک واحد به مقدار خروجی اضافه می شود ، تا اینکه نهایتا خروجی به مقدار (2^n-1) برسد . شکل زیر یک شمارنده ۸ بیتی را نشان می دهد . خروجی این شمارنده تعداد پالس های کلاک ورودی را از ۰ تا ۲۵۵ می شمارد . یعنی با آمدن اولین پالس کلاک ، خروجی ۰۰۰۰۰۰۰۱ ، دومین پالس کلاک خروجی ۰۰۰۰۰۰۱۰ ، سومین پالس کلاک ۰۰۰۰۰۰۱۱ و ... تا اینکه در نهایت خروجی با آمدن ۲۵۶ پالس کلاک به ۱۱۱۱۱۱۱۱ می رسد و با آمدن کلاک بعدی دوباره همه خروجی ها ۰ می شود....



واحد تایمر/کانتر چیست ؟

بخش اصلی یک واحد تایمر/کانتر یک شمارنده (counter) می باشد که در بالا توضیحات آن داده شد . در کنار این شمارنده تعدادی رجیستر جهت تنظیمات واحد و چندی مدارات منطقی دیگر وجود دارد . رجیستر های تنظیمات واحد ، عملکرد این واحد را مشخص می کند مثلا مشخص می کند که کلاک ورودی از چه منبعی باشد ، یا حالت کار واحد در حالت تایمری باشد یا کانتری و...

مفهوم تایمر : تایمر یا زمان سنج می تواند حسی از زمان را بوجود آورد مثلا هر n میکروثانیه یکبار عملی انجام شود.

مفهوم کانتر : کانتر یا شمارنده می تواند تعداد دفعات وقوع یک رخداد را بشمارد مثلا تعداد افرادی که از یک گیت رد می شود.

تفاوت اصلی در مفهوم تایمر با کانتر : همانطور که از تعاریف فوق مشخص است در تایمر نظم و ترتیب وجود دارد یعنی بعد از گذشت مدتی میتوان حدس زد چند بار آن عمل مورد نظر انجام شده اما در کانتر لزوما اینطور نیست و ممکن است هر تعدادی رخداد در یک بازه زمانی بوجود آید.

نتیجه : در صورتی که به شمارنده (Counter) پالس کلاکی متناوب با دوره تناوب ثابت اعمال کنیم ، به دلیل اینکه زمان اضافه شدن به مقدار شمارنده را می دانیم ، تایمر بوجود می آید و در صورتی که پالس کلاک به صورت نامتناوب و غیر منظم باشد ، کانتر بوجود می آید.

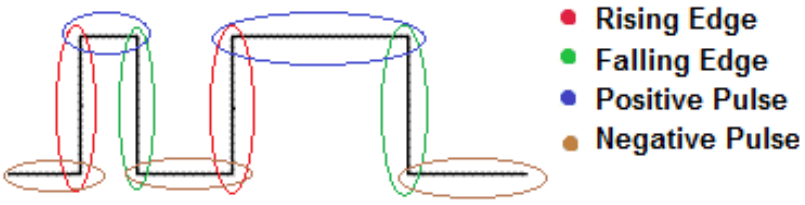
مهمترین ویژگی های یک واحد تایمر/کانتر در میکروکنترلرهای AVR

مهمترین مسائلی که در هنگام کار با واحد تایمر/کانتر بوجود می آید به شرح زیر می باشد:

- **طول شمارنده واحد :** تعداد بیت های شمارنده(کانتر) که میتواند ۸ ، ۱۶ ، ۳۲ و ... باشد. در میکروکنترلرهای AVR تنها طول ۸ بیتی و ۱۶ بیتی وجود دارد.
- **رجیسترهای تنظیمات واحد :** همواره تعدادی رجیستر در کنار واحد به منظور انجام تنظیمات وجود دارد که بسته به ساده یا پیچیده بودن تعداد آنها کم یا زیاد می شود.
- **مد (mode) یا حالت کار واحد :** برای این واحد دو حالت کار متفاوت وجود دارد . حالت کار تایمری و حالت کار کانتری.
- **تنظیمات کلاک ورودی واحد :** در حالت **تایمری** کلاک ورودی تقسیمی از کلاک اصلی می باشد و از **داخل** میکروکنترلر به واحد وارد می شود اما در حالت **کانتری** کلاک واحد از طریق پایه مربوطه (پایه های Tx) و از **خارج** میکروکنترلر به واحد اعمال می شود.
- **مشخص کردن رخداد در حالت کانتری :** زمانی که واحد در حالت کانتری کار میکند باید نوع رخدادی که میخواهیم شمرده شود را تنظیم نماییم . این رخداد یکی از انواع رخدادها در سیگنال ورودی به شرح زیر است:

1. **لبه بالارونده(Rising Edge)**
2. **لبه پایین رونده(Falling Edge)**
3. **پالس مثبت(Positive Pulse)**
4. **پالس منفی(Negative Pulse)**

در شکل زیر انواع رخداد های فوق که در یک نمونه سیگنال مشخص شده است را مشاهده می کنید.



- **مشخص کردن حالت کار واحد تایمر/کانتر :** زمانی که واحد در حالت تایمری کار می کند میتوان از آن در حالت های مختلف زیر استفاده کرد:

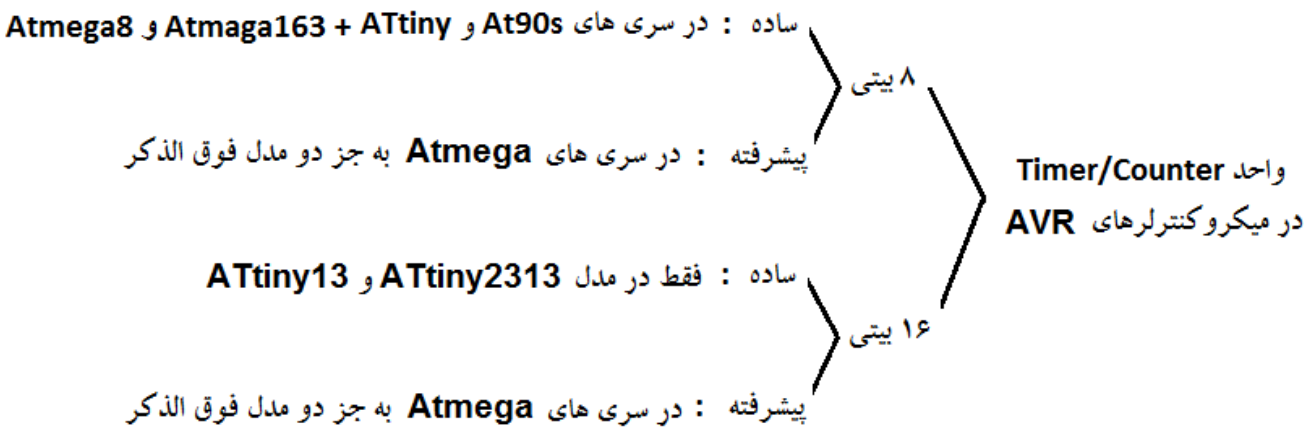
1. **حالت ساده(عادی)**
2. **حالت مقایسه(CTC)**
3. **حالت PWM سریع (تک شیب)**
4. **حالت PWM تصحیح فاز (دو شیب)**
5. **حالت PWM تصحیح فاز و فرکانس**

تذکر : نحوه عملکرد دقیق واحد تایمر/کانتر در حالت های فوق ، در بخش مربوط به هر یک تشریح خواهد شد.

- **فعال یا غیر فعال بودن خروجی :** هر یک از واحدهای تایمر/کانتر ، **حداکثر** سه خروجی دارد که هر کدام از خروجی ها به یکی از پایه های میکروکنترلر متصل می شود . پایه هایی که مربوط به واحد تایمر/کانتر هستند در میکروکنترلرهای AVRبا نام **OCx**مشخص می شوند که در آن **xیک عدد بین ۰ تا ۴ است** . در صورت فعال بودن خروجی واحد ، پایه مربوطه از حالت I/O معمولی (واحد ورودی/خروجی) خارج می شود و به خروجی واحد تایمر/کانتر متصل می گردد.

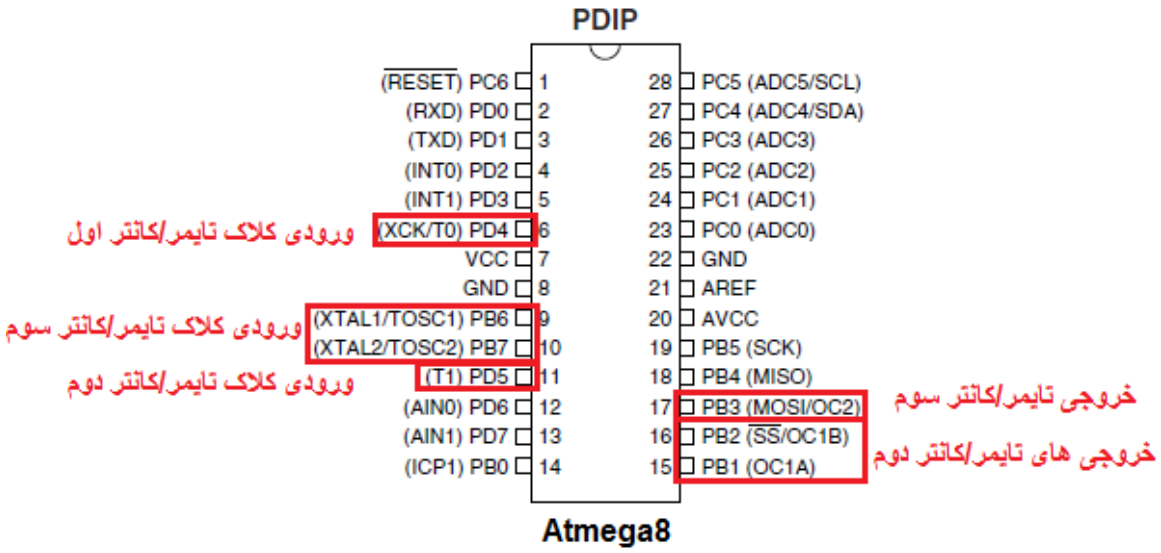
انواع واحد تایمر/کانتر در میکروکنترلرهایAVR

بسته به پیچیدگی تنظیمات مورد نیاز آن واحد ، در میکروکنترلر های AVR دو نوع تایمر/کانتر ساده و پیشرفته در ابعاد ۸ و ۱۶ بیتی وجود دارد . شکل زیر انواع این واحد را به همراه میکروکنترلر AVR مورد نظر نشان می دهد . لازم به **تذکر** است که هر میکروکنترلر AVR حداقل یک و حداکثر پنج واحد تایمر/کانتر دارد که هر کدام از آنها میتواند یکی از ۴ حالت زیر باشد.



تذکر : هر میکروکنترلر AVR دارای تعدادی واحد تایمر/کانتر می باشد (حداقل ۱ تا حداکثر ۵ واحد) که به ترتیب از شماره ۰ نامگذاری می شود . هر یک از این واحد ها به خودی خود میتواند ساده ، پیشرفته ، ۸ بیتی یا ۱۶ بیتی باشد.

مثال : میکروکنترلر Atmega8 دارای ۳ واحد تایمر/کانتر می باشد . تایمر/کانتر شماره ۰ به صورت ساده ۸ بیتی ، تایمر/کانتر شماره ۱ به صورت پیشرفته ۱۶ بیتی و تایمر/کانتر شماره ۲ به صورت پیشرفته ۸ بیتی می باشد . پایه های خروجی مربوط به هریک از واحدهای تایمر/کانتر را در شکل زیر مشاهده می کنید.



نکته : همانطور که در شکل فوق نیز مشاهده می کنید ، در تایمر/کانتر های ساده ۸ بیتی پایه خروجی وجود ندارد ، در تایمر/کانترهای پیشرفته ۸ بیتی ۱ خروجی و در تایمر/کانترهای پیشرفته ۱۶ بیتی ، دو خروجی و در برخی میکروکنترلرها سه خروجی وجود دارد.

نکته : همانطور که در شکل فوق نیز مشاهده می کنید ، کلیه ورودی های واحدهای تایمر/کانتر ، کلاک حالت کار **کانتری** می باشند که در حالت کار تایمری بدون استفاده است . ضمنا ورودی کلاک تایمر/کانتر سوم (پایه های TOSC1و TOSC2) تنها میتواند برای RTC (زمان سنج حقیقی) استفاده شود.

معرفی اجمالی رجیستر های واحد تایمر/کانتر

این رجیسترها که نام آنها در همه واحد های تایمر/کانتر اعم از ساده ، پیشرفته ، ۸ بیتی یا ۱۶ بیتی یکسان هستند و به طور اتوماتیک توسط کدویزارد مقدار دهی می شوند ، به شرح زیر می باشد:

رجیستر کنترل تایمر/کانتر (TCCRX)

این رجیستر که مخفف Timer Counter Control Register است ، وظیفه کنترل کلیه تنظیمات واحد تایمر کانتر را بر عهده دارد . این تنظیمات که در کد ویزارد نیز انجام می شود ، به شرح زیر است:

1. فعال یا غیر فعال بودن واحد
2. مشخص کردن حالت کار تایمری یا کانتری (منبع کلاک)
3. تنظیم عدد تقسیم کلاک ورودی واحد در حالت تایمری
4. مشخص کردن نوع رخداد در حالت کانتری
5. تنظیم حالت های مختلف کاری واحد PWM) ، CTCو ...)
6. فعال یا غیر فعال کردن خروجی واحد (پایه های OCX)

رجیستر تایمر/کانتر (TCNTX)

این رجیستر ۸ بیتی ، مقدار شمارنده در هر لحظه را به صورت اتوماتیک در خود ذخیره می کند . این رجیستر امکان دسترسی مستقیم برای خواندن و نوشتن در شمارنده را فراهم می کند. به طوری که این رجیستر هنگام خواندن مقدار شمارش شده رو برمیگرداند و به هنگام نوشتن مقدار جدید را به شمارنده انتقال می دهد.

رجیستر مقایسه خروجی (OCRX)

این رجیستر هشت بیتی خواندنی و نوشتنی بوده و به طور مستقیم با مقدار شمارنده TCNT مقایسه می شود. از تطابق این دو برای تولید وقفه خروجی یا تولید یک شکل موج روی پایه OCX می توان استفاده نمود.

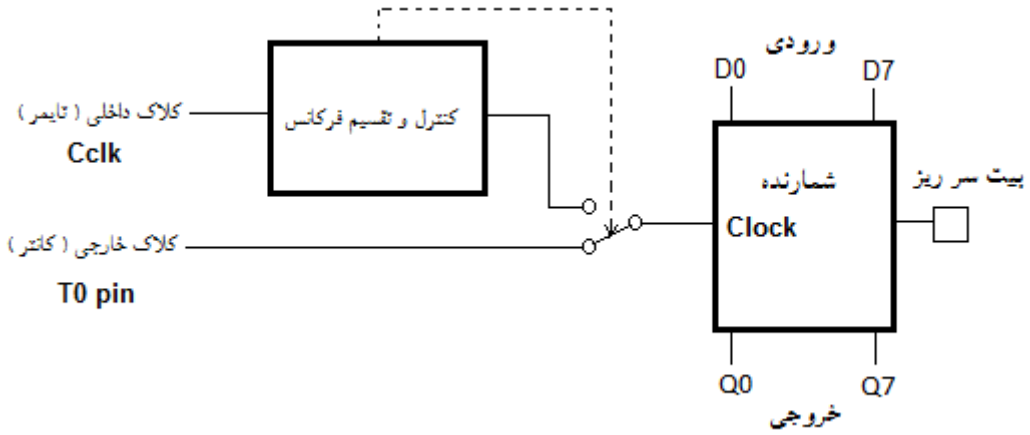
رجیستر پوشش وقفه تایمر/کانتر (TIMSK)

این رجیستر برای تنظیمات وقفه در هنگام سر ریز شدن تایمر/کانتر یا در هنگام تطبیق مقایسه (Compare Match) مورد استفاده قرار می گیرد.

رجیستر پرچم سر ریز تایمر/کانتر (TIFR)

که در این رجیستر بیت TOV0 زمانی یک می شود که یک سر ریز در تایمر یا کانتر صفر رخ داده باشد و بیت های دیگر ... که همه توسط کدویزارد تنظیم می شوند.

معرفی و تشریح تایمر/کانتر ساده ۸ بیتی



همانطور که در شکل فوق مشاهده می کنید ، شمارنده دارای ۸ بیت ورودی می باشد که به کمک آن میتوان عدد دلخواه را در هر لحظه در طول برنامه روی شمارنده (رجیستر TCCNT) بارگذاری نمود . همچنین خروجی شمارنده نیز ۸ بیت می باشد که با آمدن هر یک رخداد ، یک واحد به آن اضافه می گردد تا اینکه مقدار شمارنده به حداکثر مقدار خود یعنی ۲۵۵ برسد . بعد از اینکه مقدار شمارنده به ۲۵۵ رسید با آمدن رخداد بعدی ، بیت سر ریز ۱ می شود و همزمان مقدار شمارنده نیز ریست می گردد یعنی همه خروجی ها ۰ می شود.

در کنار شمارنده ، واحد کنترل و تقسیم فرکانس وجود دارد . این واحد وظیفه کنترل منبع کلاک ورودی را دارد که در حالت تایمری ، کلاک داخلی انتخاب می شود و در حالت کانتری نیز کلاک خارجی انتخاب می شود . همچنین در این واحد در صورت انتخاب کلاک داخلی میتوان تقسیمی از آن را به عنوان کلاک شمارنده انتخاب و به شمارنده داد که به آن پیش تقسیم کننده (Prescaler) گویند به طوری که توسط این واحد یکی از Cclk ، Cclk/8 ، Cclk/64 ، Cclk/256و Cclk/1024 را میتوان انتخاب نمود.

نکته : در حالت استفاده از کلاک خارجی (کانتر) باید توجه داشت که حداقل زمان بین دو رخداد Cclk می باشد (Cclkهمان فرکانس کاری میکروکنترلر می باشد)

محاسبه زمانبندی سر ریز شدن تایمر در حالت ساده:

فرض کنید که از واحد تایمر/کانتر در حالت تایمری استفاده می کنیم . در این صورت مدت زمان یک شمارش یعنی زمانی که طول می کشد تا یک واحد به مقدار رجیستر TCNT اضافه شود از رابطه زیر بدست می آید که در آن N ضریب پیش تقسیم کننده و Fclkفرکانس کاری میکروکنترلر می باشد.

$$t = \frac{N}{f_{clk}}$$

چون برای سر ریز شدن تایمر نیاز است تا تمامی ۸ بیت شمارنده ، شمارش شود بنابراین ۲۵۶ کلاک مورد نیاز است . در نتیجه مدت زمانی که طول می کشد تا تایمر از ۰ تا ۲۵۵ رفته و پرچم سر ریز فعال شود برابر است با:

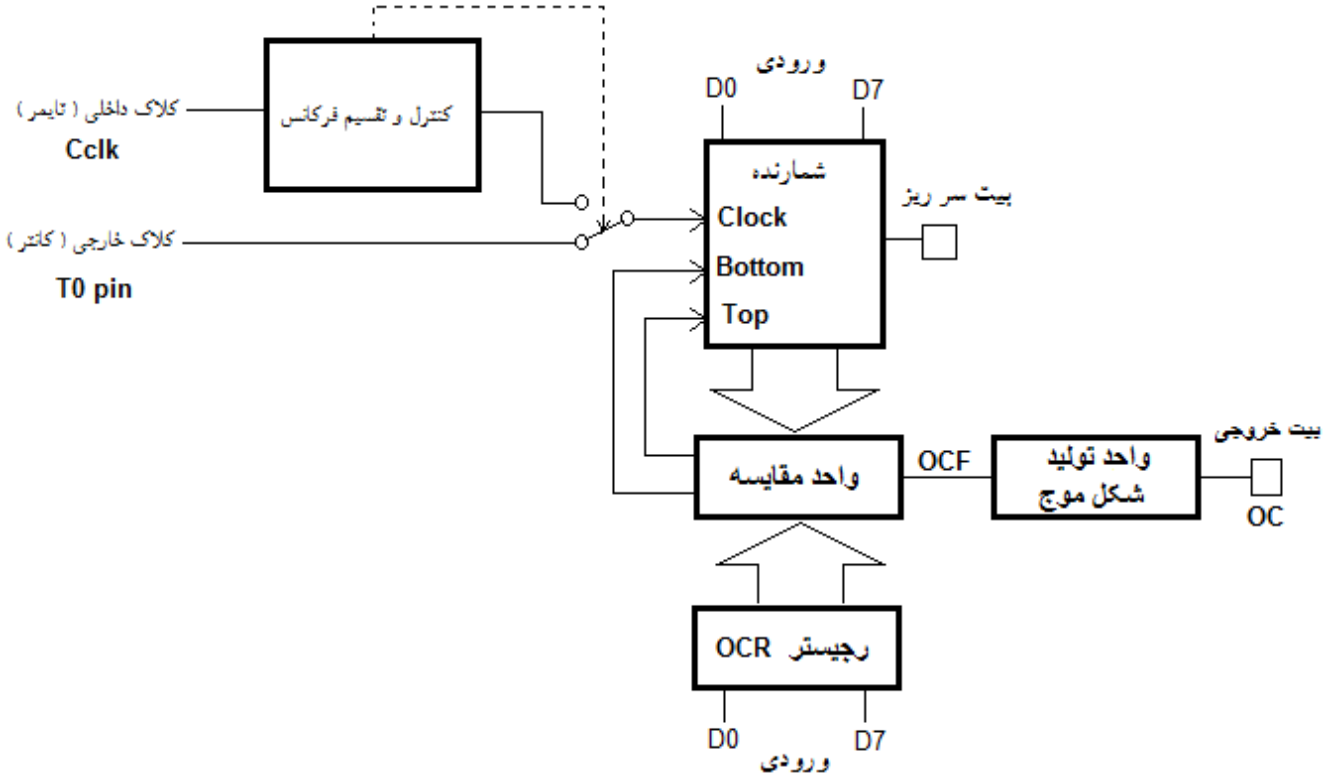
$$256 \times t = \frac{N \times 256}{f_{clk}}$$

اگر شمارش از صفر آغاز نشده باشد یعنی در ابتدای برنامه به رجیستر TCNT مقداری داده باشیم ، زمانی کمتر از زمان فوق طول می کشد تا پرچم سر ریز فعال شود که این زمان برابر است با:

$$OVF_{Time} = \frac{N \times (256 - TCNTx)}{f_{clk}}$$

که در آن TCNTx مقدار اولیه رجیستر TCNT در هنگام شروع به کار واحد تایمر/کانتر می باشد.

معرفی و تشریح تایمر/کانتر پیشرفته ۸ بیتی



همانطور که در شکل فوق مشاهده می کنید ، در حالت پیشرفته واحد مقایسه ، واحد تولید شکل موج و رجیستر ایجاد شده و نیز خود شمارنده و واحد کنترل آن پیچیدگی و قابلیت های بیشتری نسبت به حالت ساده دارد . در این واحد خروجی شمارنده توسط واحد مقایسه کننده با رجیستر هشت بیتی OCR مقایسه می شود . هرگاه این دو مقدار با هم منطبق شود (Match) ، مقایسه کننده پرچم OCF را فعال کرده و وارد واحد تولید شکل موج می گردد.

در تایمر کانترهای ۸ بیتی در حالت پیشرفته علاوه بر حالت عادی تایمری/کانتری که همانند ۸ بیتی ساده است ، حالت های دیگر نیز وجود دارد . به طور کلی عملکرد تایمر/کانتر ۸ بیتی پیشرفته می تواند در یکی از حالت های (Modeزیر باشد:

1. تایمر/کانتر در حالت ساده(Normal)
2. تایمر/کانتر در حالت مقایسه(CTC)
3. تایمر/کانتر در حالت PWM سریع(Fast PWM)
4. تایمر/کانتر در حالت PWM تصحیح فاز(Phase Correct PWM)

واحد تولید شکل موج وظیفه چگونگی اتصال یا عدم اتصال خروجی واحد مقایسه به پایه OCx میکروکنترلر را دارد و در صورتی که واحد تایمر/کانتر در حالت ساده یا حالت CTCباشد برای آن چهار حالت عملکرد زیر قابل تنظیم می باشد:

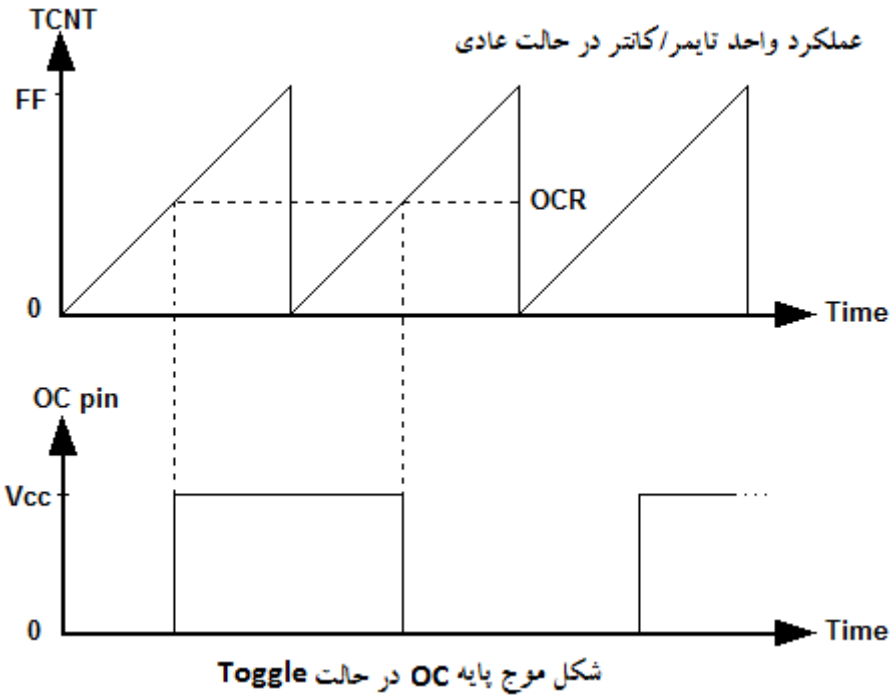
1. پایه OC غیر فعال است(Disconnected) .
2. پایه OC در هر بار تطبیق تغییر وضعیت دهد یعنی اگر ۰ بود ۱ شود و بالعکس(Toggle) .
3. پایه OC در هر بار تطبیق ۰ شود(Clear) .
4. پایه OC در هر بار تطبیق ۱ شود(Set) .

و در صورتی که واحد تایمر/کانتر در حالت PWM باشد (حالت ۳ یا ۴) برای آن سه عملکرد زیر قابل تنظیم می باشد:

1. پایه OC غیر فعال است(Disconnected) .
2. پایه OC در حالت PWM رابطه عکس با رجیستر OCR دارد(Inverted)
3. |ایه OC در حالت PWM رابطه مستقیم با رجیستر OCR دارد(Non-Inverted)

بررسی تایمر/کانتر ۸ بیتی پیشرفته در حالت ساده(Normal)

عملکرد واحد در این حالت همان عملکرد حالت ۸ بیتی ساده می باشد که تشریح شد . همانطور که در شکل زیر نیز نشان داده شده است ، در این حالت شمارش از ۰ تا ۲۵۵ می باشد و بعد از سر ریز شدن به ۰ بر می گردد . ویژگی که در حالت ۸ بیتی پیشرفته بوجود آمده ، آن است که در صورت فعال بودن پایه خروجی میتوان شکل موج زیر را در حالت Toggle روی پایه OCX ایجاد کرد.



شکل موج پایه OC در حالت Toggle

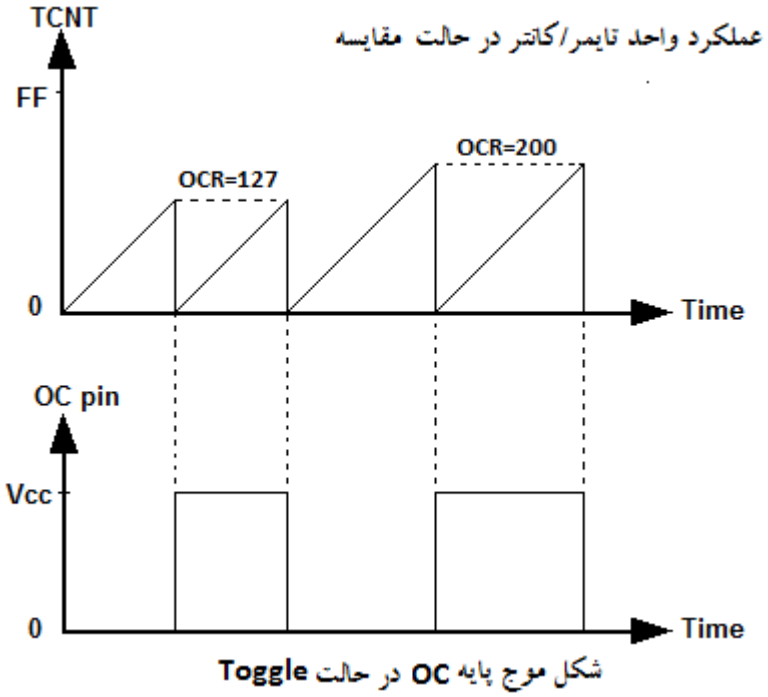
فرکانس شکل موجی که در حالت عادی روی پایه OC ایجاد می شود ، همیشه ثابت بوده و از رابطه زیر بدست می آید:

$$f_{Normal} = \frac{1}{2 \times OVF_{Time}} = \frac{f_{clk}}{2N \times (256 - TCNTx)}$$

که در آن N ضریب پیش تقسیم کننده و Fclk فرکانس کاری میکرو و TCNTx مقدار اولیه رجیستر TCNT در لحظه شروع به کار تایمر است.

بررسی تایمر/کانتر ۸ بیتی پیشرفته در حالت مقایسه(CTC)

در این حالت مقدار شمارش شده در شمارنده با هر بار آمدن کلاک علاوه بر اضافه شدن با مقدار OCR مقایسه می شود و در صورت برابر شدن این دو مقدار شمارنده ریست می شود . در حالت نرمال وقتی شمارنده به مقدار حداکثر خود یعنی ۲۵۵ میرسید ، ریست میشد اما در این حالت به محض رسیدن به مقدار OCR که خود مقداری بین ۰ تا ۲۵۵ میتواند باشد ، ریست می شود . شکل زیر عملکرد واحد را در این حالت نشان می دهد.



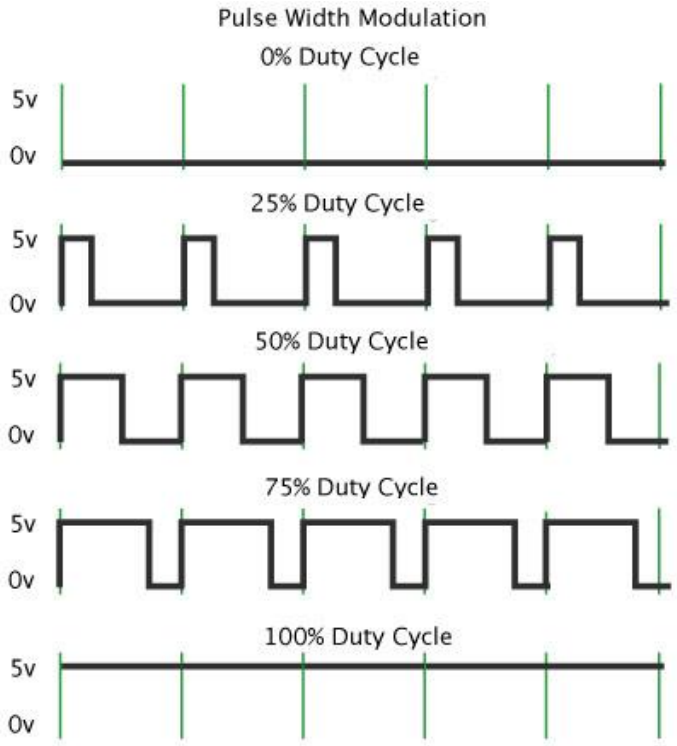
شکل موج پایه OC در حالت Toggle

از این حالت میتوان برای تولید شکل موج با فرکانس متغیر روی پایه خروجی (OC pin) استفاده نمود به طوری که با تغییر رجیستر OCR در طول برنامه فرکانس پایه OC خروجی طبق رابطه زیر تغییر می کند . در این رابطه N ضریب تقسیم فرکانس کلاک کاری میکرو (Fclk) است که درون واحد تایمر/کانتر تنظیم می شود و OCRx مقدار رجیستر OCR در واحد تایمر/کانتر شماره x است.

$$f_{ocx} = \frac{f_{clk}}{2N(OCRx + 1)}$$

PWM چیست ؟

مخفف عبارت Pulse Width Modulation به معنای "مدولاسیون عرض پالس" می باشد . کاربرد PWM در میکروکنترلر ها برای مصارف مختلفی مانند کنترل شدت نور LED ها ، کنترل سرعت انواع موتور های DC ، انتقال پیام ، مبدل های ولتاژ و ... است . در واقع با استفاده از این تکنیک میتوان موجی با فرکانس ثابت اما عرض پالس های متفاوت بوجود آورد . عرض پالس را دیوتی سایکل (Duty Cycle) نیز می گویند. دیوتی سایکل مدت زمان ۱ بودن به مدت زمان کل پریود در هر سیکل موج است که معمولا بر حسب درصد (%) نمایش داده می شود. به فرض مثال اگر Duty Cycle یک موج PWM برابر با ٪۴۰ باشد بدان معنی است که در هر سیکل ٪۴۰ ولتاژ برابر VCC و در ٪۶۰ اوقات ولتاژ برابر ۰ است. همانگونه که می دانید در چنین حالتی ولتاژ موثر یا Vrms برابر با ٪۴۰ VCC خواهد بود. به فرض مثال شما اگر با یک میکرو با تغذیه ۵ V، موج PWM با دیوتی سایکل ٪۵۰ ایجاد نمایید ولتاژ RMSشما برابر ٪۵۰ VCC یا به عبارتی ۲،۵ ولت خواهد بود. در شکل زیر تعدادی موج PWM با فرکانس ثابت و دیوتی سایکل متفاوت نمایش داده شده است.



تذکر : با استفاده از ویژگی PWM در میکروکنترلرهای AVR میتوان انواع موتورهای DC را به پایه OC متصل کرده و سرعت آنها را از طریق تغییر توان کنترل کرد به طوری که هر چه Duty Cycleبیشتر باشد ، موتور سریعتر و هرچه کمتر باشد موتور کندتر می چرخد . در دیوتی سایکل ۱۰۰ درصد بیشترین توان به موتور اعمال می شود.

تولید PWM به روش نرم افزاری و بدون استفاده از واحد تایمر

برای تولید موج PWM در روش نرم افزاری ، نیاز به تولید یک موج با فرکانس ثابت و دیوتی سایکل متغیر داریم . فرض می کنیم فرکانس مطلوب f باشد در این صورت زمان هر پریود ۱ f/می شود . میخواهیم برنامه ای بنویسیم که فرکانس دلخواه و دیوتی سایکل را بر حسب درصد داشته باشد و شکل موج مورد نظر را برای مثال روی پایه PA.0 ایجاد کند . بنابراین برنامه به صورت زیر می باشد:

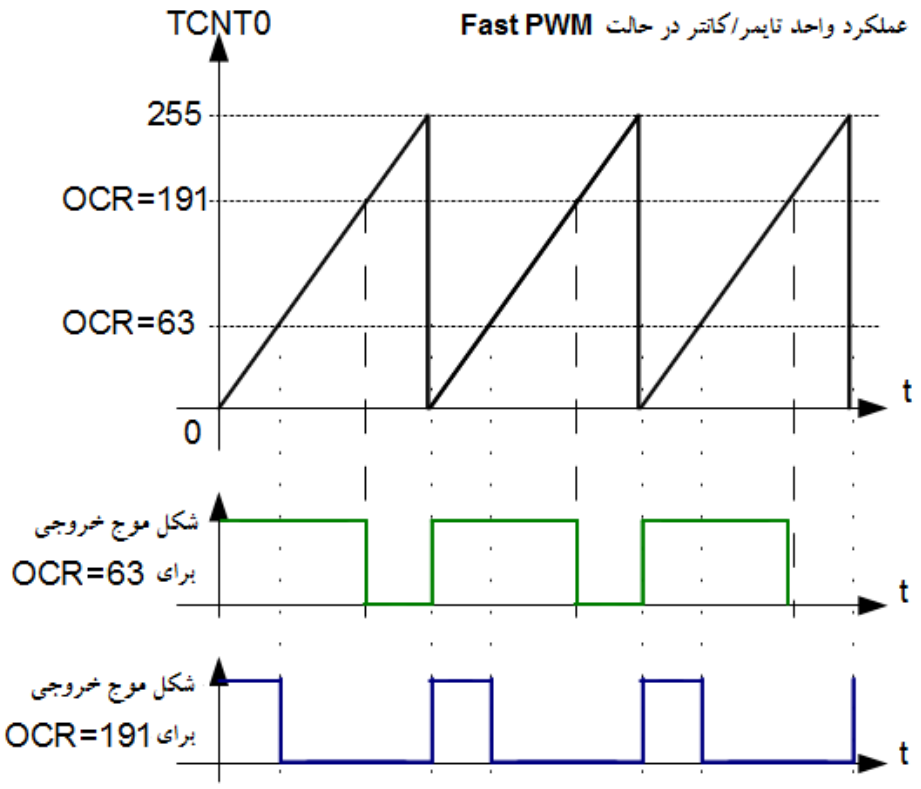
```
...
unsigned int f=1000; //PWM Frequency
unsigned int T=1/f; //PWM Period
unsigned int DutyCycle=60; //Percent Of DutyCycle
...
while(1)
{
PORTA.0=0;
delay_ms((1-DutyCycle/100)*T);
PORTA.0=1;
delay_ms(DutyCycle/100*T);
}
```

توضیح برنامه : در ابتدا فرکانس ، پریود و دیوتی سایکل مورد نیاز برای PWM را تعریف می کنیم . برنامه درون حلقه نامتناهی دائما اجرا می شود و نتیجه آن تولید شکل موجی مربعی با فرکانس تقریبا ۱ کیلوهرتز (زیرا مجموع تاخیرهای موجود در برنامه به اندازه T است) و دیوتی سایکل تقریبا ۶۰ درصد است.

نکته : تولید PWM به روش فوق دقیق نبوده و به علت اینکه CPU میکروکنترلر برای تولید موج PWM درگیر می شود کاربرد چندانی ندارد اما برای تولید دقیق موج PWM از **سخت افزار مجزا از CPU** به نام واحد تایمر/کانتر در حالت PWM میتوان استفاده کرد.

بررسی تایمر/کانتر ۸ بیتی پیشرفته در حالت PWM سریع(Fast PWM)

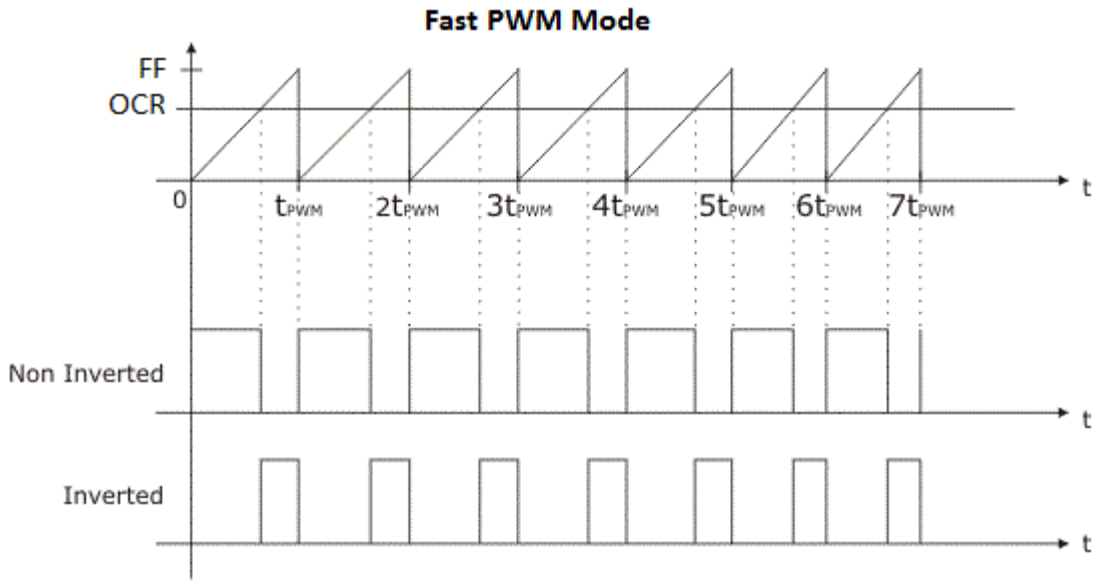
در این حالت عملکرد واحد تایمر/کانتر مانند حالت مقایسه است با این تفاوت که در زمان تطابق میان OCR و TCNTمقدار شمارنده ریست نمی شود بلکه تا مقدار ماکزیمم خود یعنی ۲۵۵ ادامه میابد و بعد از سر ریز شدن صفر می گردد . همچنین هنگام ۰ شدن مقدار شمارنده پایه OC نیز صفر می شود . در حالت PWM پایه OC در دو حالت تغییر وضعیت می دهد یکی در زمان تطبیق OCR و TCNT و دیگری در زمان سر ریز شدن TCNT در حال که در حالت CTC پایه خروجی تنها در زمان تطابق تغییر می کرد . این عملکرد باعث می شود که موج خروجی دارای فرکانس ثابت باشد و عرض پالس (Duty Cycle) توسط رجیستر OCR کنترل می شود به طوری که هر چقدر مقدار OCR بیشتر باشد عرض پالس کاهش می یابد و بالعکس.



در این حالت فرکانس پایه خروجی ثابت است و دیوتی سایکل بین ۰ تا ۱۰۰ درصد تغییر می کند . فرکانس خروجی از رابطه زیر بدست می آید که در آن N ضریب پیش تقسیم کننده و Fclk فرکانس کاری میکروکنترلر است.

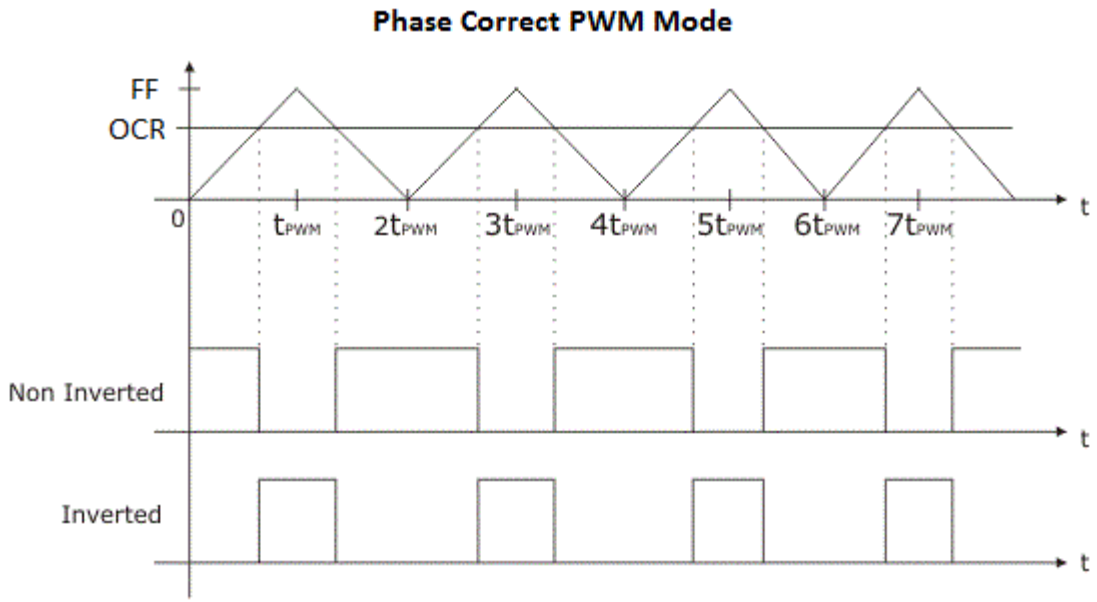
$$f_{ocx} = \frac{f_{clk}}{N \times 256}$$

نکته مهم : در حالت PWM پایه خروجی میتواند قطع (Disconnect) یا به صورت معکوس (Inverted) ویا به صورت غیر معکوس (Non-Inverted) باشد . شکل موج خروجی در حالت معکوس با شکل موج خروجی حالت غیر معکوس Not یکدیگر هستند به طوری که مثلا اگر دیوتی سایکل در حالت معکوس ۸۰ درصد باشد ، دیوتی سایکل در حالت غیر معکوس ۲۰ درصد است . شکل زیر علاوه بر نشان دادن عملکرد حالت PWM این موضوع را نیز نشان می دهد.



بررسی تایمر/کانتر ۸ بیتی پیشرفته در حالت PWM تصحیح فاز(Phase Correct PWM)

در این حالت موج PWM تولید شده دارای دقت بالاتری نسبت به حالت قبل است چرا که شیفت فاز ناخواسته که در حالت PWM سریع ممکن است رخ دهد ، در این حالت اصلاح شده است . از این حالت برای کارهای با دقت بیشتر (مثلا راه اندازی سروو موتور یا ارسال دیتای مخابراتی (مورد استفاده قرار می گیرد . تنها تفاوت این حالت با حالت قبل این است که شمارش ابتدا در جهت افزایشی و سپس در جهت کاهشی است . بنابراین ابتدا شمارنده از ۰ تا ۲۵۵ و سپس از ۲۵۵ تا ۰ می شمارد و فقط در هربار تطابق با OCR، خروجی تغییر وضعیت می دهد.



نکته : تولید PWM به روش تصحیح فاز دارای فرکانس خروجی پایین تری نسبت به حالت سریع آن است و طبق رابطه زیر محاسبه می شود که در آن N ضریب پیش تقسیم کننده و Fclk فرکانس کاری میکرو است.

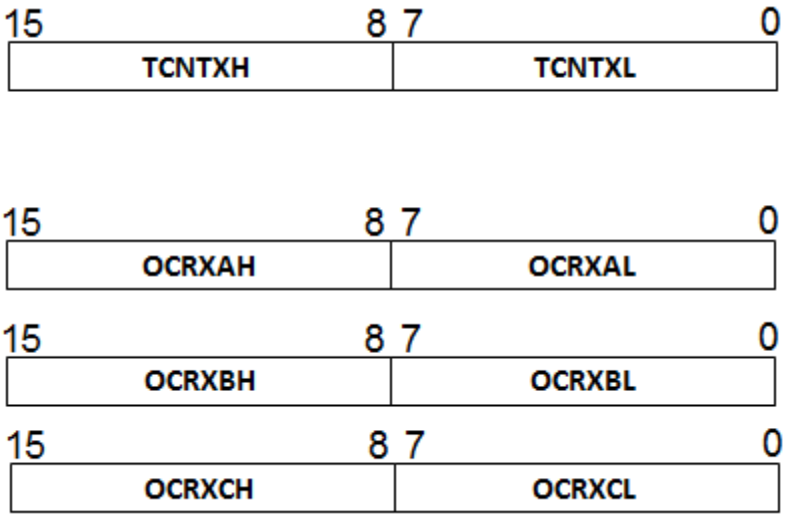
$$f_{ocx} = \frac{f_{clk}}{N \times 510}$$

معرفی اجمالی رجیسترهای تایمر/کانترهای ۱۶ بیتی

در تایمر/کانترهای ۱۶ بیتی ، شمارنده اصلی تایمر/کانتر رجیستر TCNTx می باشد . این رجیستر یک رجیستر ۱۶ بیتی است که از دو رجیستر ۸ بیتی با نامهای TCNTXL و TCNTXH تشکیل شده است و به صورت خواندنی و نوشتنی قابل دسترسی است .

رجیستر های مقایسه خروجی تایمر/کانتر در تایمر/کانترهای ۱۶ بیتی به جای یک رجیستر ۸ بیتی ، از ۳ رجیستر ۱۶ بیتی با نامهای OCRxA ، OCRxB و OCRxC تشکیل شده است .این رجیسترها که به صورت خواندنی و

نوشتنی هستند هر یک واحد مقایسه مجزایی دارند که خروجی آنها به واحد تولید شکل موج و نهایتا به یکی از پایه های میکروکنترولر به نامهای OCXA ، OCXBو OCXC متصل می شود.



نکته : رجیستر OCRXC فقط در برخی از میکروکنترلرهای AVR که دارای چهار یا پنج واحد تایمر/کانتر هستند نظیر Atmega128 و ... وجود دارند . (استثنا Atmega162 : که با وجود ۴ تایمر/کانتر این رجیستر را ندارد)

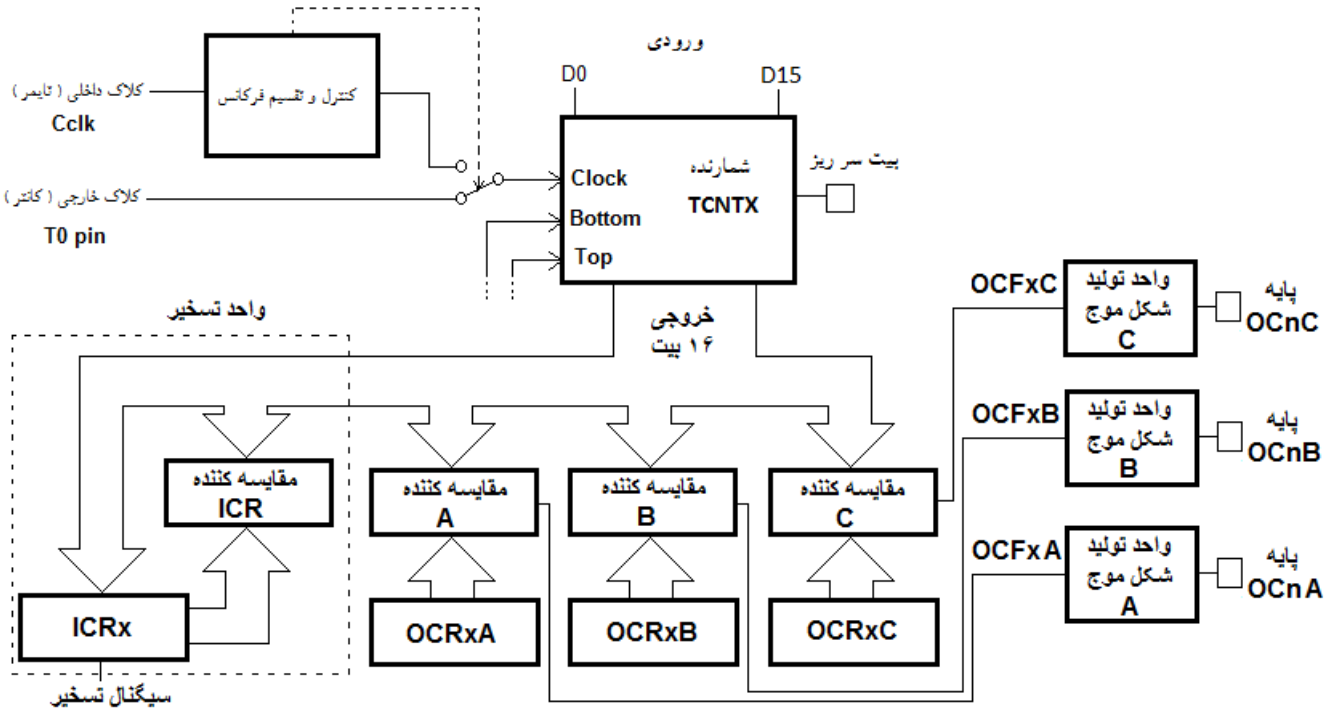
در تایمر/کانترهای ۱۶ بیتی به جای یک رجیستر تنظیمات سه رجیستر **۸ بیتی** به نام های TCCRXA ، TCCRXBو TCCRXC وجود دارد . این سه رجیستر کلیه اعمال واحد تایمر/کانتر را نظیر حالت عملکرد ، ضریب پیش تقسیم کننده ، نحوه اتصال خروجی و ... کنترل می کند.

رجیستر های TIFR و TIMSK در اینجا نیز به همان صورت و برای همان کاربردهای ذخیره پرچم (Flag) و پوشش وقفه وجود دارند.

رجیستر ۱۶ بیتی دیگری به نام ICxR مربوط به واحد تسخیر نیز وجود دارد که در زمان رخ دادن تسخیر محتوای رجیستر TCNTx به این رجیستر منتقل می شود.

تذکر : به علت اینکه از تایمر/کانتر ۱۶ بیتی ساده تنها در دو میکروکنترلر AtTiny13 و AtTiny2313استفاده شده است از بیان آن اجتناب می کنیم زیرا با توضیح و بررسی تایمر/کانتر ۱۶ بیتی پیشرفته ، نوع ساده در این دو میکروکنترلر نیز پوشش داده می شود.

معرفی و تشریح تایمر/کانتر پیشرفته ۱۶ بیتی



تایمر/کانتر ۱۶ بیتی ساختاری همانند تایمر/کانتر ۸ بیتی پیشرفته دارد با این تفاوت که علاوه بر ۱۶ بیتی شدن پهنای شمارنده اصلی ، تنظیمات و کنترل های بیشتر و پیچیده تری در واحد تایمر/کانتر وجود دارد به طوری که به جای یک رجیستر مقایسه ، سه رجیستر مقایسه و در برخی میکروها دو رجیستر مقایسه وجود دارد . همچنین به تعداد این رجیستر های مقایسه ، واحد مقایسه و واحد تولید شکل موج نیز وجود دارد . و همچنین یک واحد دیگر به نام واحد تسخیر به آن اضافه شده است . واحد تسخیر میتواند در زمان رخداد خارجی روی پایه ICPx (در برخی میکروکنترلرها ICx) محتوای رجیستر ۱۶ بیتی TCNTx را در رجیستر ۱۶ بیتی ICRx ذخیره کرده و وقفه تسخیر را نیز فعال نماید. بدین ترتیب واحد تسخیر میتواند رخدادهای خارجی را تسخیر کرده و به آن یک عملکرد خاص نسبت دهد.

به طور کلی عملکرد تایمر/کانتر ۱۶ بیتی پیشرفته در یکی از حالت های زیر می باشد:

1. تایمر/کانتر در حالت ساده(Normal)
2. تایمر/کانتر در حالت مقایسه(CTC)
3. تایمر/کانتر در حالت PWM سریع(Fast PWM)
4. تایمر/کانتر در حالت PWM تصحیح فاز(Phase Correct PWM)

5. تایمر/کانتر در حالت PWM تصحیح فاز و فرکانس(Phase & Frequency Correct PWM)

نکته : واحد تایمر/کانتر پیشرفته ۱۶ بیتی در صورت وجود در هر یک از میکروکنترلرهای AVR معمولا تایمر/کانتر شماره ۱ یا شماره ۳ است.

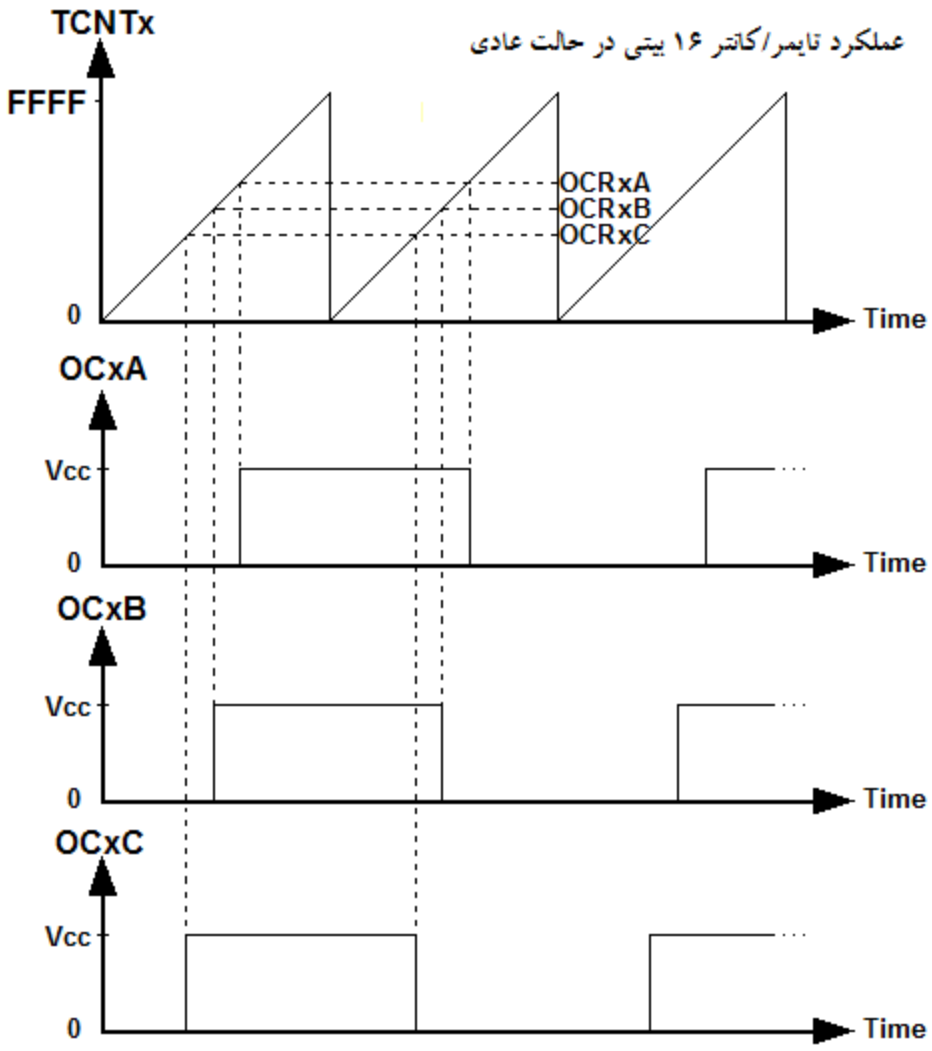
نکته : اکثر میکروکنترلرهای AVR حداکثر ۳ واحد تایمر/کانتر دارند و فقط میکروکنترلرهای Atmega64 ، Atmega128 ، Atmega162 ، Atmega2560 ، Atmega2561 ، Atmega640 ، Atmega1280و Atmega1281هستند که دارای تایمر چهارم و پنجم نیز می باشند.

نکته : اگر فیوز بیت M161Cدر میکروکنترلر Atmega162 که برای تطابق میکروکنترلر Atmega162 با Atmega161تعبیه شده است ، فعال باشد ، تایمر/کانتر شماره ۳ در دسترس نبوده و غیر فعال می شود.

نکته : اگر فیوز بیت M103Cدر میکروکنترلرهای Atmega128 و Atmega64 که برای تطابق میکروکنترلرهای Atmega64و Atmega128 با Atmega103 تعبیه شده است ، فعال باشد ، تایمر/کانتر شماره ۳ و نیز رجیستر OCR3Cدر دسترس نبوده و غیر فعال می شود.

تایمر/کانتر ۱۶ بیتی پیشرفته در حالت ساده(Normal)

ساده ترین حالت عملکرد این واحد می باشد که در این حالت شمارش رو به بالا بوده و مقدار رجیستر TCNTx از (0000 Hex) تا (FFFF Hex) می باشد و بعد از سر ریز شدن به ۰ بر می گردد . در هنگام شمارش رجیستر TCNTxدائما با رجیسترهای (OCRxA ، OCRxB، OCRxC) مقایسه شده و در صورت برابر شدن هر یک از آنها بیت تطابق مربوطه (OCFxA ، OCFxB، OCFxC) روشن شده و خروجی مربوط به آن با توجه به تنظیمات واحد تولید شکل موج روی پایه های مربوطه (OCxA ، OCxB، OCxC) ظاهر می شود.



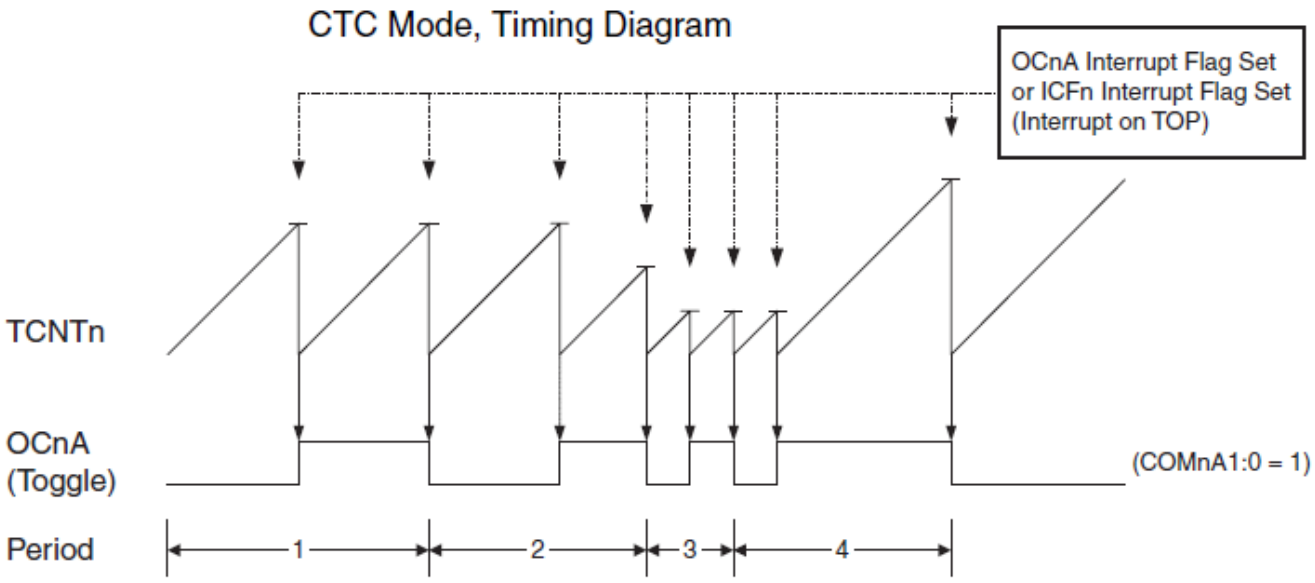
فرکانس شکل موجی که در حالت عادی روی پایه OCx ایجاد می شود ، همیشه ثابت بوده و از رابطه زیر بدست می آید:

$$f_{ocx} = \frac{f_{clk}}{2N \times (65536 - TCNTx)}$$

که در آن N ضریب پیش تقسیم کننده و Fclk فرکانس کاری میکرو و TCNTx مقدار اولیه رجیستر TCNTx در لحظه شروع به کار تایمر است.

تایمر/کانتر ۱۶ بیتی پیشرفته در حالت مقایسه(CTC)

در حالت مقایسه رجیستر TCNTx به طور دائم با رجیستر OCRxA یا ICRx مقایسه می شود و در صورت برابری ، رجیستر TCNTx برابر صفر می شود . شکل زیر نحوه عملکرد تایمر/کانتر در این حالت را نشان می دهد.



با تغییر هر یک از رجیستر های OCRxA,B,C میتوان فرکانس های مختلفی روی پایه های OCxA,B,C ایجاد کرد که فرکانس هر یک از رابطه زیر بدست می آید:

$$f_{OCnA} = \frac{f_{clk}}{2 \cdot N \cdot (1 + OCRnA)}$$

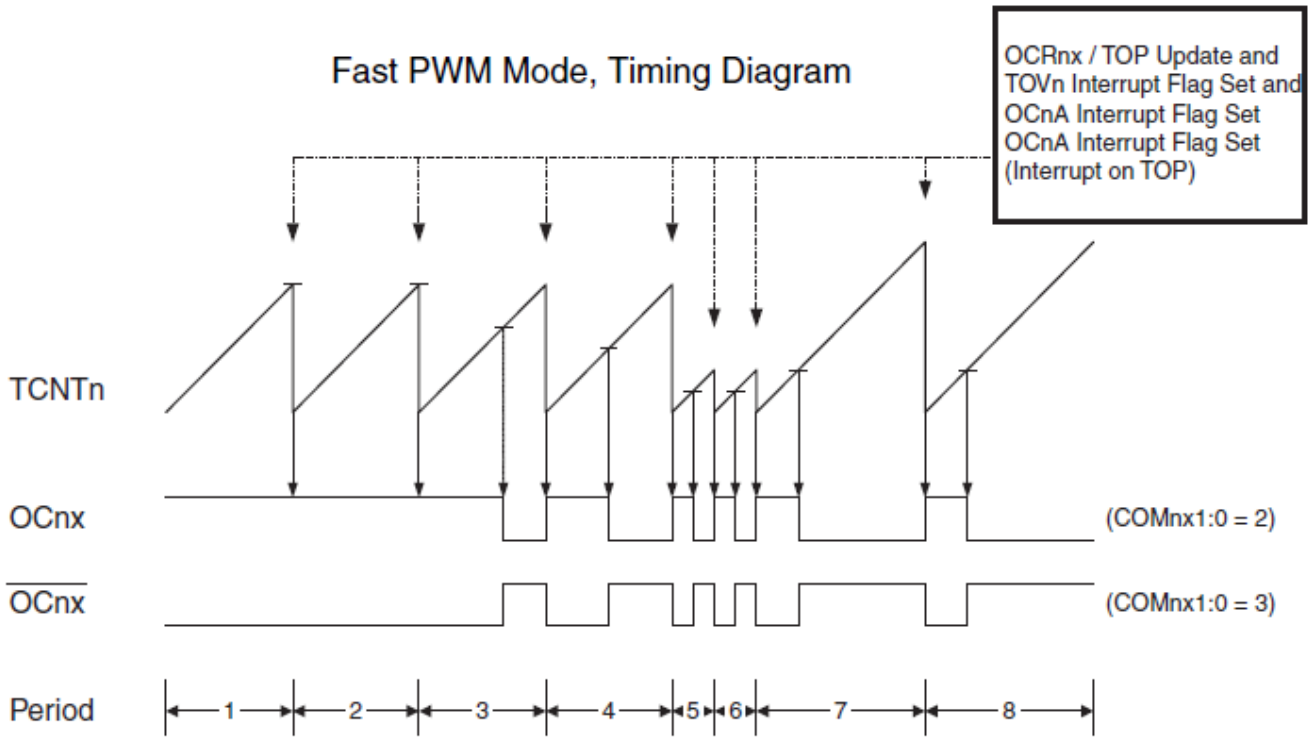
برای پایه های OCnB و OCnC نیز چنین فرمولی صادق است که در آن N ضریب پیش تقسیم کننده و Fclk فرکانس کاری میکرو است.

تایمر/کانتر ۱۶ بیتی پیشرفته در حالت PWM سریع(Fast PWM)

در این حالت تولید شکل موج PWM با فرکانس بالا قابل انجام است . این حالت خود حداکثر دارای پنج مد عملکرد زیر است:

1. PWM سریع هشت بیت (با حداکثر مقدار FF)
2. PWM سریع نه بیت (با حداکثر مقدار 1FF)
3. PWM سریع ده بیت (با حداکثر مقدار 3FF)
4. PWM سریع با حداکثر مقدار رجیستر ICRx
5. PWM سریع با حداکثر مقدار رجیستر OCRxA

در این حالت تایمر تا زمانی که مقدارش مطابق تنظیمات فوق یکی از مقادیر FF ، ۱ FF، ۳ FF، مقدار درون رجیستر ICRxو یا مقدار درون رجیستر OCRxA شود ، به صورت افزایشی ادامه می یابد سپس پرچم سر ریز فعال شده و تایمر ریست می شود . در زمان سر ریز تایمر و نیز زمان تطابق آن با رجیستر OCRxA,B,C مربوطه تغییر حالت در خروجی رخ می دهد . شکل زیر نحوه عملکرد تایمر/کانتر در این حالت را نشان می دهد.



نکته : همواره باید توجه داشت که مقدار حداکثر شمارش همیشه معادل یا بیشتر از مقدار رجیسترهای مقایسه باشد . اگر این مقدار کمتر باشد هیچگاه تطابق رخ نمی دهد و تایمر به درستی کار نمی کند.

فرکانس موج PWM بسته به مقدار حداکثری که دارد ، دارای فرکانس ثابت و دیوتی سایکل متغیر است . دیوتی سایکل توسط رجیسترهای OCRxA,B,C کنترل می شوند و فرکانس موج PWM از رابطه زیر بدست می آید:

$$f_{OCnxPWM} = \frac{f_{clk}}{N \cdot (1 + TOP)}$$

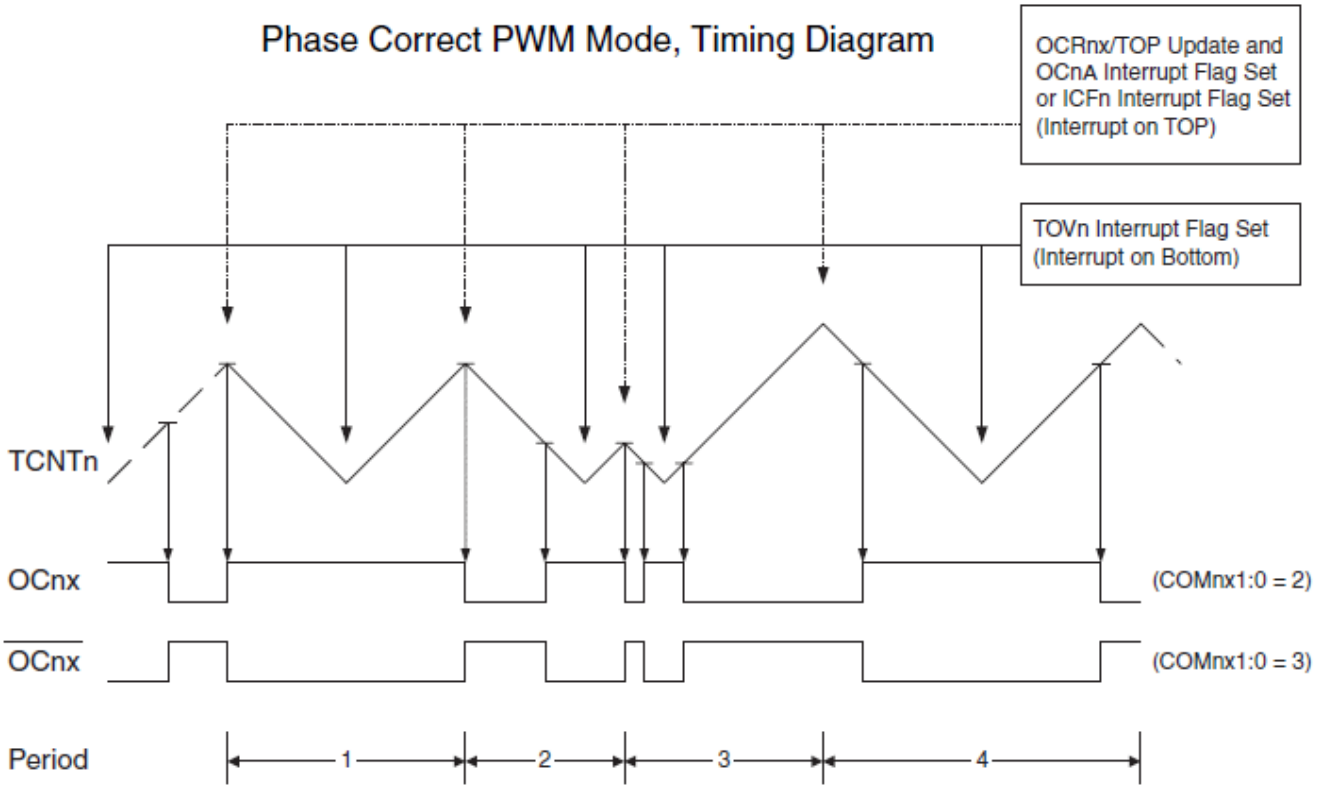
که در آن N ضریب پیش تقسیم کننده و Fclk فرکانس کاری میکرو و TOP مقدار حداکثری است که در هر یک از حالت های پنج گانه فوق الذکر مشخص است.

تایمر/کانتر ۱۶ بیتی پیشرفته در حالت PWM تصحیح فاز(Phase Correct PWM)

در این حالت تولید موج PWM با تفکیک بالا و فرکانس پایینتر نسبت به حالت سریع در دسترس می باشد . این حالت خود حداکثر دارای پنج مد عملکرد زیر است:

1. PWMتصحیح فاز هشت بیت (با حداکثر مقدار FF)
2. PWMتصحیح فاز نه بیت (با حداکثر مقدار 1FF)
3. PWMتصحیح فاز ده بیت (با حداکثر مقدار 3FF)
4. PWMتصحیح فاز با حداکثر مقدار رجیستر ICRx
5. PWMتصحیح فاز با حداکثر مقدار رجیستر OCRxA

در این حالت تایمر تا زمانی که مقدارش مطابق تنظیمات فوق یکی از مقادیر FF ، ۱ FF، ۳ FF، مقدار درون رجیستر ICRxو یا مقدار درون رجیستر OCRxA شود ، به صورت افزایشی ادامه می یابد سپس جهت شمارش تغییر نموده و از حداکثر مقدار به سمت صفر شمارش به صورت نزولی ادامه می یابد . در لحظه رسیدن به مقدار حداکثر ، پرچم ICRxیا OCxA و در لحظه رسیدن به صفر ، پرچم سر ریز فعال می شود . شکل زیر نحوه عملکرد تایمر/کانتر در این حالت را نشان می دهد.



در این حالت نیز دیوتی سایکل توسط رجیسترهای OCRxA,B,C کنترل می شوند و رابطه زیر برای فرکانس موج خروجی را داریم:

$$f_{OCnxPCPWM} = \frac{f_{clk}}{2 \cdot N \cdot TOP}$$

که در آن N ضریب پیش تقسیم کننده و Fclk فرکانس کاری میکرو و TOP مقدار حداکثری است که در هر یک از حالت های پنج گانه فوق الذکر مشخص است.

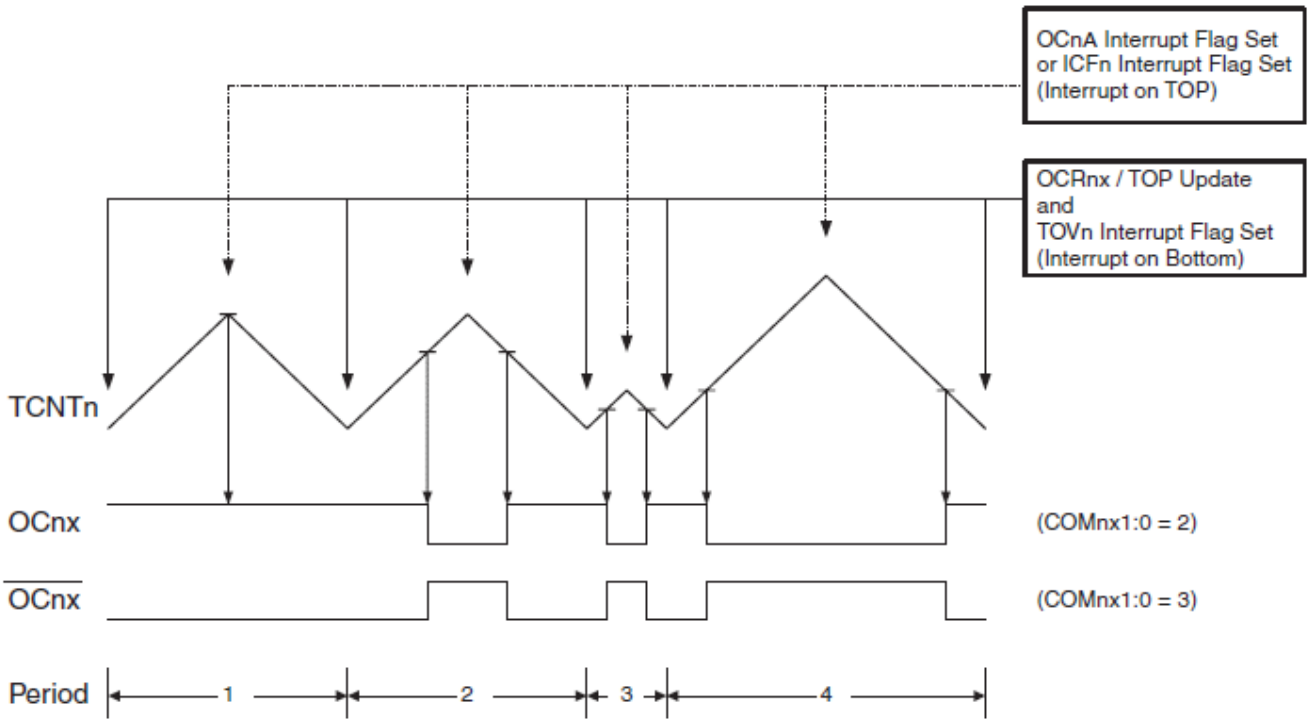
تایمر/کانتر ۱۶ بیتی پیشرفته در حالت PWM تصحیح فاز و فرکانس(Phase & Frequency Correct PWM)

عملکرد واحد تایمر/کانتر در این حالت مشابه عملکرد حالت تصحیح فاز می باشد . تنها تفاوت این حالت با حالت تصحیح فاز زمان بروزرسانی رجیسترهای OCRxA,B,C است که در حالت تصحیح فاز در زمان حداکثر مقدار تایمر بود اما در این حالت در زمان مقدار صفر اتفاق می افتد . این عمل باعث بهبود موج PWM و متقارن شدن آن می گردد . این حالت خود حداکثر دارای دو مد عملکرد زیر است:

1. PWMتصحیح فاز و فرکانس با حداکثر مقدار رجیسترICRx
2. PWMتصحیح فاز و فرکانس با حداکثر مقدار رجیسترOCRxA

شکل زیر نحوه عملکرد تایمر/کانتر در این حالت را نشان می دهد.

Phase and Frequency Correct PWM Mode, Timing Diagram



در این حالت نیز دقیقا همانند حالت قبل ، دیوتی سایکل توسط رجیسترهای OCRxA,B,C کنترل می شوند و رابطه زیر برای فرکانس موج خروجی را داریم:

$$f_{OCnxPCPWM} = \frac{f_{clk}}{2 \cdot N \cdot TOP}$$

که در آن N ضریب پیش تقسیم کننده و Fclk فرکانس کاری میکرو و TOP مقدار حداکثری است که در هر یک از حالت های پنج گانه فوق الذکر مشخص است.

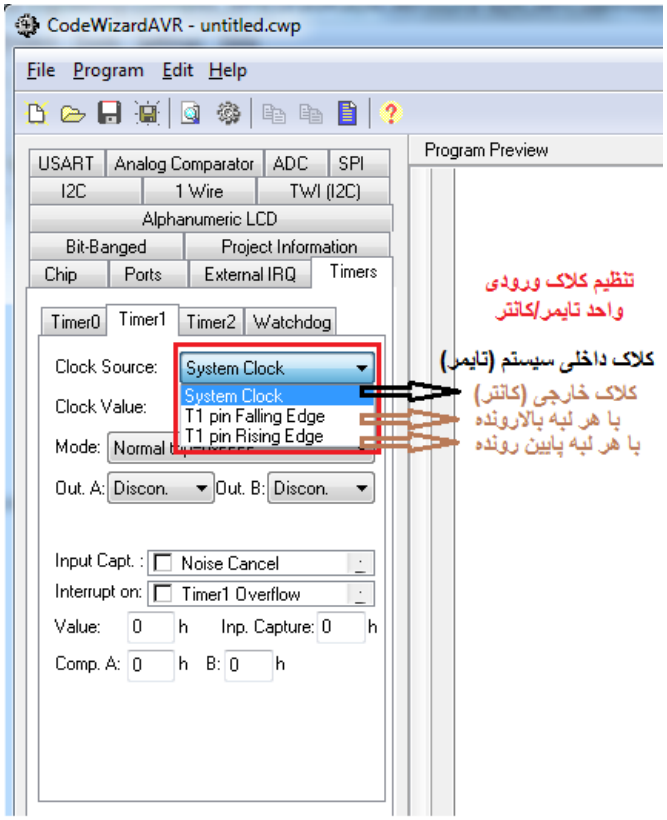
تنظیمات واحد تایمر/کانتر در کد ویزاردCodeWizard

در کد ویزارد برگه ی Timers مربوط به تنظیمات تایمر/کانتر میکرو می باشد. هر میکروکنترلری که انتخاب شود کدویزارد به طور اتوماتیک برگه ی Timer را بر حسب نوع تایمر/کانترهای همان میکروکنترلر و امکانات و تنظیمات آن آپدیت می کند . در قسمت Clock Source ، ورودی تحریک تایمر مشخص می شود که میتواند از پایه خارجی میکرو (کانتر) یا از کلاک خود میکرو (تایمر) باشد . در صورتی که ورودی تحریک از کلاک سیستم باشد آنگاه باید در قسمت Clock Value ، ضریب پیش تقسیم کننده (N) مشخص شود . در قسمت Mode ، حالت کار تایمر/کانتر مشخص می گردد به طوری که کلیه حالت های ممکن در این قسمت وجود دارد و انتخاب می شود . در قسمت Output مشخص می شود که هنگام تساوی دو رجیستر TCNTx و OCRx پایه ی خروجی OCx چگونه عمل کند. در دو گزینه ی بعدی امکان فعال شدن وقفه در زمان سرریز و یا در زمان تساوی دو رجیستر TCNTx و OCRxفراهم می شود . در قسمت Timer Value مقدار شروع عدد داخل رجیستر TCNT مشخص می شود . در قسمت Compare نیز عدد داخل رجیستر OCR مشخص می شود.

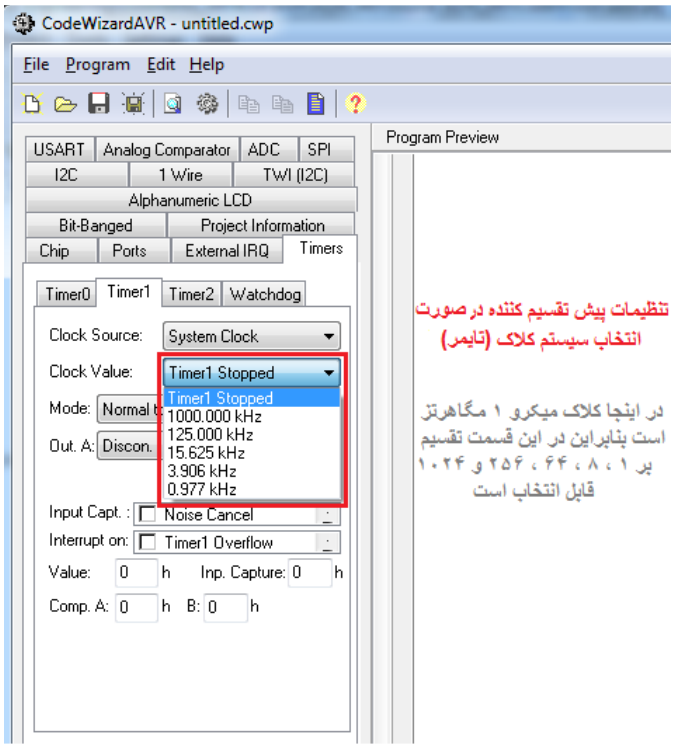
نکته : در برخی از تایمر/کانترها واحد کاهش نویز (Noise Canceler) وجود دارد که با استفاده از یک فیلتر دیجیتال ساده ، سیگنال ICPx نسبت به نویز را بهبود می بخشد به طوری که با اعمال سیگنال به پایه ICPx ، چهار نمونه یکسان از آن گرفته و در صورت برابری آنها تسخیر صورت می پذیرد.

در شکل های زیر نحوه تنظیمات واحد تایمر/کانتر به طول کامل برای تایمر شماره ۱ میکروکنترلر Atmega32 (پیشرفته ۱۶ بیتی) و با کلاک کاری میکرو ۱ مگاهرتز آورده شده است.

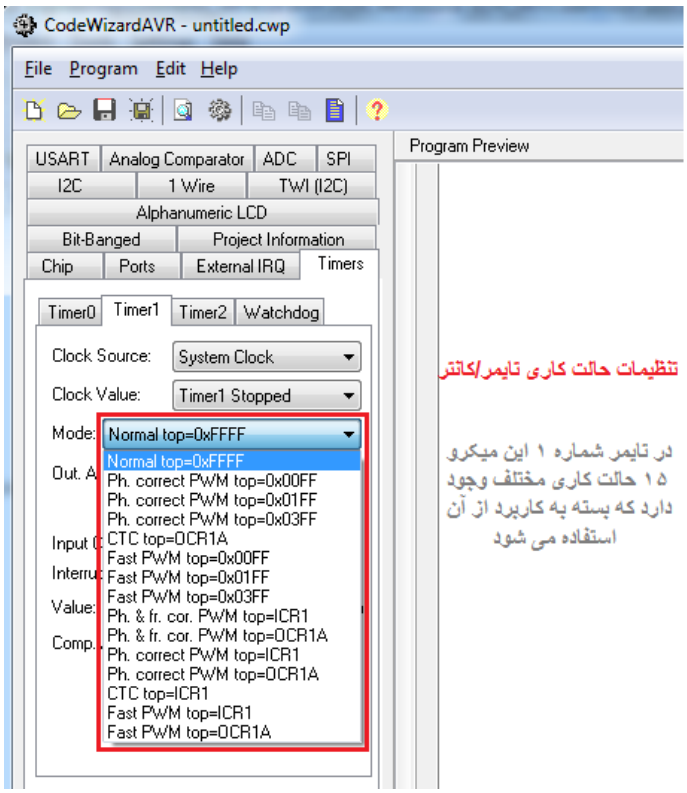
شکل اول : تنظیم حالت کار تایمری یا کانتری



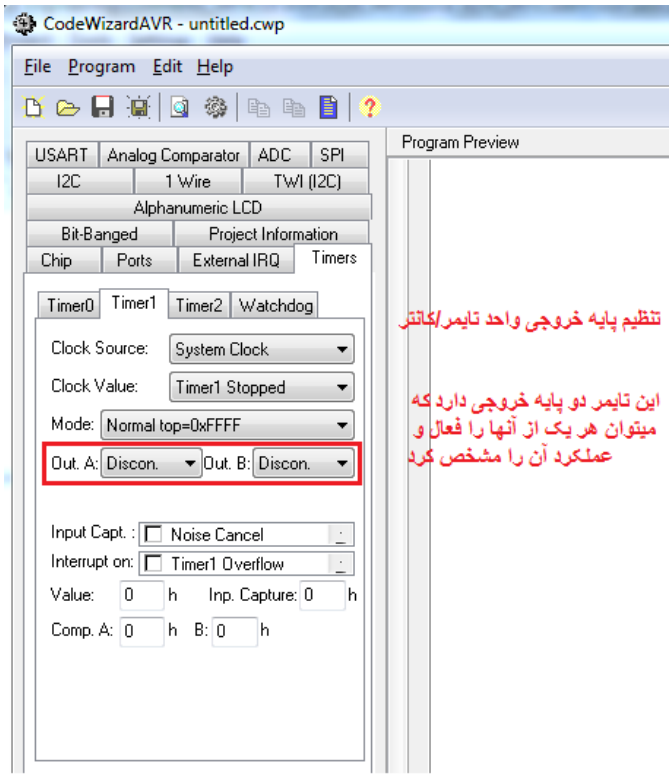
شکل دوم : تنظیم پیش تقسیم کننده در حالت تایمری



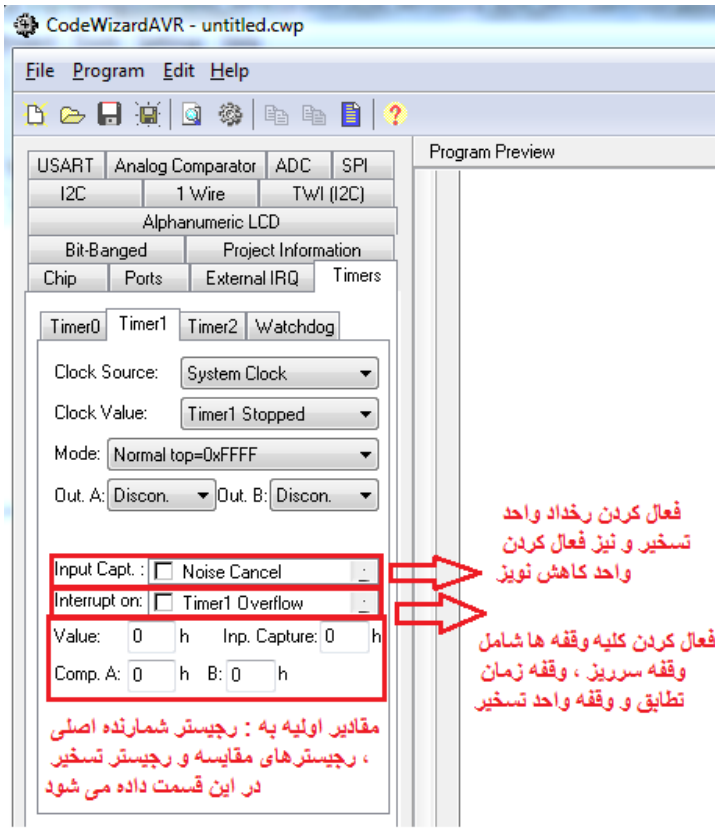
شکل سوم : تنظیم حالت عملکرد تایمر/کانتر



شکل چهارم : تنظیم خروجی

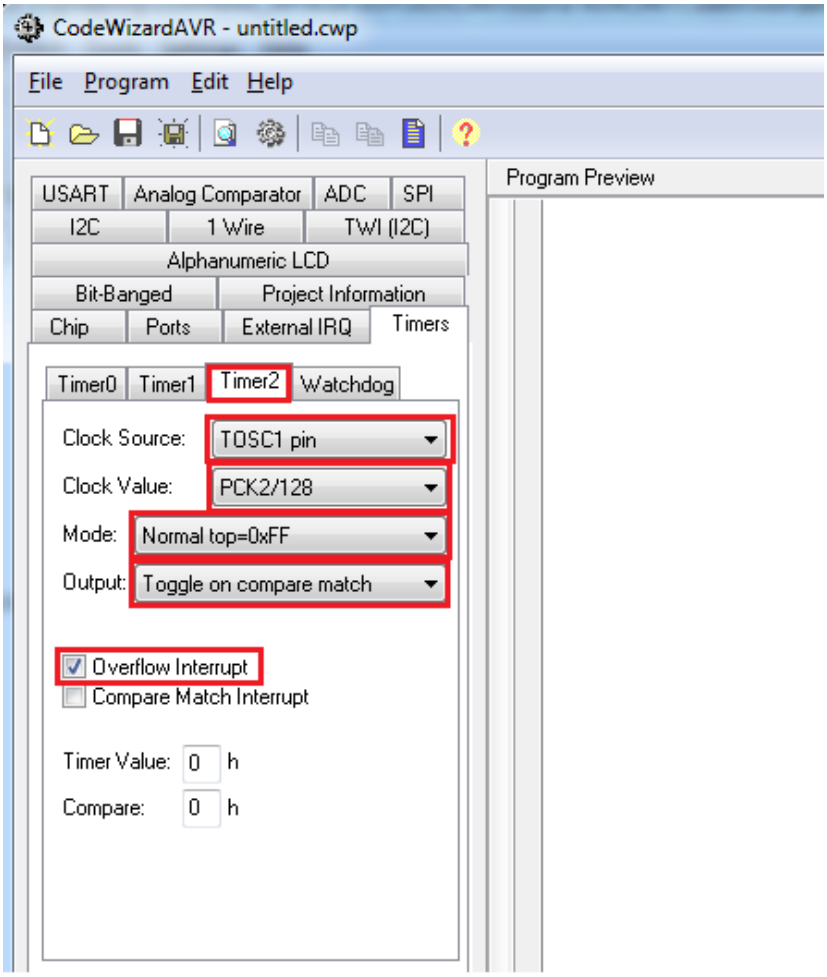


شکل پنجم : تنظیمات پایانی



راه اندازی RTC در میکروکنترلرهایAVR

قابلیت ویژه ای که در برخی از میکروکنترلرهای AVR وجود دارد این است که میتوان از واحد تایمر/کانتر به عنوان زمان سنج حقیقی (Real Time Clock) استفاده کرد . با فعالسازی این ویژگی میتوان ۱ ثانیه دقیق را در داخل میکرو ایجاد و از آن در پروژه ها استفاده کرد . در برخی از میکروکنترلرها تایمر شماره ۰ و در اکثر میکروکنترلرها تایمر شماره ۲ این ویژگی را دارد . زمانی که این حالت فعال شود کلاک تایمر از کریستال خارجی ۳۲۷۶۸ هرتز که بین دو پایه به نام های TOSC1و TOSC2 تامین می شود که به منظور تولید ۱ ثانیه دقیق طراحی شده است . کافی است تا با راه اندازی این تایمر در کد ویزارد و قرار دادن یک کریستال با فرکانس ۳۲۷۶۸ هرتز به این دو پایه و تنظیم ضریب تقسیم در کدویزارد روی PCK2/128و مد نرمال به این هدف رسید . ضمنا میتوان با فعال کردن خروجی و قرار دادن آن روی Toggle و قراردادن یک LED روی پایه OC2 آن را هر ثانیه یکبار روشن و خاموش نمود . شکل زیر فعالسازی این ویژگی در Atmega32 را نشان می دهد . با فعالسازی وقفه تایمر میتوان کاری که در هر ثانیه در برنامه باید انجام شود را درون تابع روتین وقفه انجام داد.



تایمرسگ نگهبان

به تایمری خاص اشاره دارد که وظیفه آن نگهداری و نظارت بر کار میکروکنترلر است. این تایمر مجهز به اسیلاتور RC داخلی برای خود است که پس از شمارش و سرریز شدن، این قابلیت را دارد که میکروکنترلر را بصورت داخلی ریست کند .این تایمر در مواردی کاربرد دارد که امکان قفل کردن تراشه وجود دارد و به این وسیله پس از قفل کردن میکروکنترلر، دیگر امکان ریست کردن WDT وجود ندارد و به همین دلیل WDT شمارش خود را انجام داده و سرریز می شود و در نتیجه میکروکنترلر را ریست می کند تا از حالت قفل خارج شود. برنامه میکروکنترلر باید بگونه ای باشد که در حین اجرا، تایمر سگ نگهبان (Watch Dog Timer) بصورت مداوم قبل از سرریز شدن، صفر شود .

تنظیمات تایمر Watchdogدر کدویزارد

در سربرگ Timers سربرگی مخصوص تنظیمات تایمر سگ نگهبان وجود دارد. با فعالسازی این قسمت و زدن تیک Enableمی بایست ضریبی برای تقسیم فرکانس تایمر انتخاب نمود. فرکانس اصلی نوسان ساز تایمر در حالت عادی ۱ مگاهرتز در ولتاژ تغذیه ۵ ولت می باشد . بر اساس تایمی که میخواهیم تا اگر میکرو هنگ کرد تایمر سگ نگهبان آن را ریست کند باید این ضریب را مشخص نمود که در جدول زیر به طور کامل آورده شده است . برای جلوگیری از سر ریز شدن تایمر سگ نگهبان باید به طور مداوم آن را در طول برنامه ریست کرد . برای ریست کردن تایمر از دستور زیر در طول برنامه استفاده می شود.

#asm(“WDR”); //Reset Watchdog Timer

WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 3.0V	Typical Time-out at V_{CC} = 5.0V
0	0	0	16K (16,384)	17.1 ms	16.3 ms
0	0	1	32K (32,768)	34.3 ms	32.5 ms
0	1	0	64K (65,536)	68.5 ms	65 ms
0	1	1	128K (131,072)	0.14 s	0.13 s
1	0	0	256K (262,144)	0.27 s	0.26 s
1	0	1	512K (524,288)	0.55 s	0.52 s
1	1	0	1,024K (1,048,576)	1.1 s	1.0 s
1	1	1	2,048K (2,097,152)	2.2 s	2.1 s

از اینکه تا اینجا با ما همراه بودید بسیار ممنون و متشکریم.

اما این پایان آموزش ها نیست ! در حال ویرایش و تکمیل این جزوه هستیم . بنابراین منتظر ویرایش های بعدی این جزوه در هفته های آتی باشید. به محض تکمیل مباحث AVR به سراغ میکروکنترلرهای ARM خواهیم رفت و همانند این جزوه برای این میکروکنترلرها نیز ارائه خواهیم کرد.

پس

با ما همراه باشید...

به سایت ما سر بزنید...

و به دوستان خود نیز سایت ما را پیشنهاد کنید...

مطمئنا همراهی شما باعث دلگرمی ما و سرعت گرفتن آموزش ها خواهد بود...

با تشکر شجاع داودی

ElectroVolt.ir