
بسم الله الرحمن الرحیم

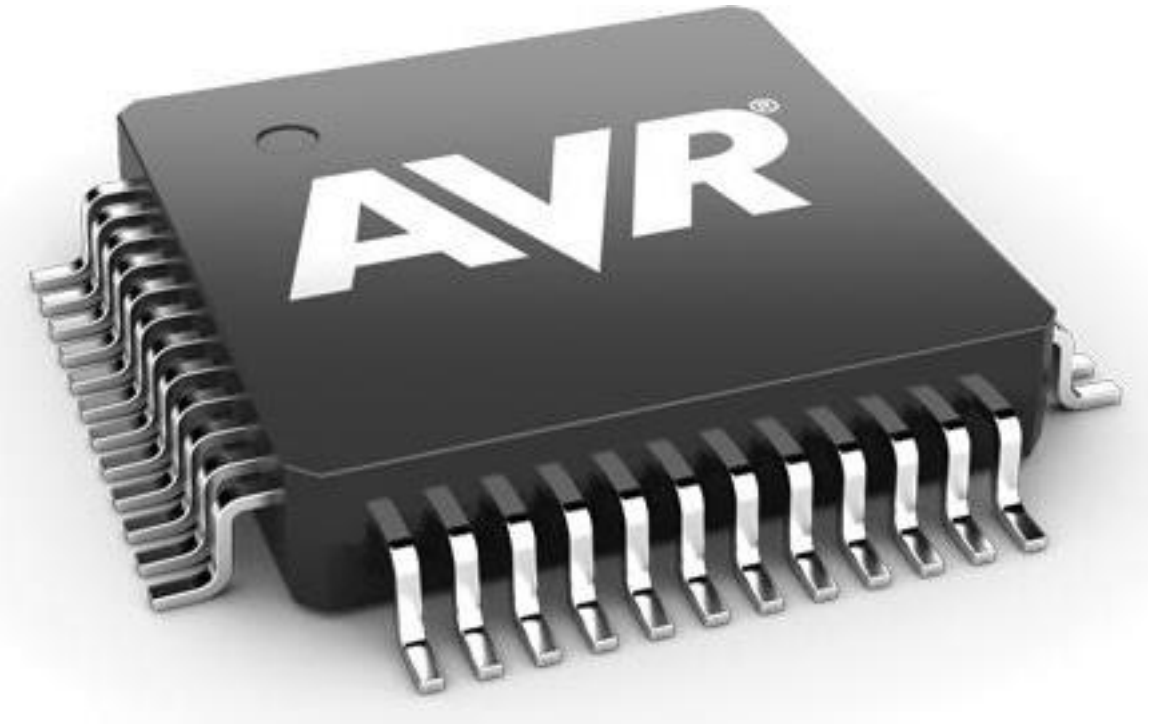
آموزش کاربردی میکروکنترلرهای AVR

از ۰ تا ۱۰۰

ElectroVolt.ir

تقدیم به همه مشتاقان الکترونیک دیجیتال

بهترین و جامع ترین جزوه آموزش میکروکنترلر AVR به زبان C



تهیه و تالیف : محمد حسین شجاع داودی

ناشر : وبسایت الکترو ولت Electrovolt.ir

"ویرایش سوم"

اسفند ۱۳۹۴

فهرست مطالب

عنوان صفحه

فصل ۱– مقدمه

۱–۱– پیشگفتار مولف

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. 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.

یکی از میکروکنترلرهای پرکاربرد و معروف AVR نام دارد که ساخت شرکت Atmel می باشد. این میکروکنترلرها جزو ساده ترین و در عین حال پر کاربرد ترین میکروکنترلرهای موجود می باشد. از میکروکنترلرهای AVR در کاربردهایی وسیع با قابلیت های متوسط (مانند پروژه های دانشگاهی، تجاری، منازل و...) استفاده می گردد.

اولین و بهترین گزینه برای ورود به دنیای الکترونیک دیجیتال، یاد گرفتن میکروکنترلرهای AVR است چرا که اصولی ترین و پایه ای ترین مباحث الکترونیک دیجیتال در آموزش این میکروکنترلرها وجود دارد که به عنوان پیش نیازی برای یادگیری سطوح بالاتر و میکروکنترلرهای قوی تر همانند ARM و آرایه های منطقی برنامه پذیر FPGA است. یادگیری این میکروکنترلرها بسیار شیرین بوده و پروژه های بسیار جالب و پرکاربردی با استفاده از آنها میتوان خلق کرد. همچنین تهیه این میکروکنترلرها بسیار راحت تر و قیمت آن نسبت به بقیه میکروکنترلرها بسیار مناسب بوده و همه علاقه مندان میتوانند لوازم مورد نیاز آن ها را تهیه و به راحتی استفاده نمایند.

خوشبختانه منابع گسترده و وسیعی در اینترنت برای یادگیری میکروکنترلرهای AVR وجود دارد که بسیار در این زمینه کمک می کنند 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 نظر خوانندگان را به این حوزه جلب کند. ضمنا ویژگی دیگر این جزوه از ۰ تا ۱۰۰ 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 میکروکنترلرهای AVR فقط به این جزوه نیاز داشته باشید.

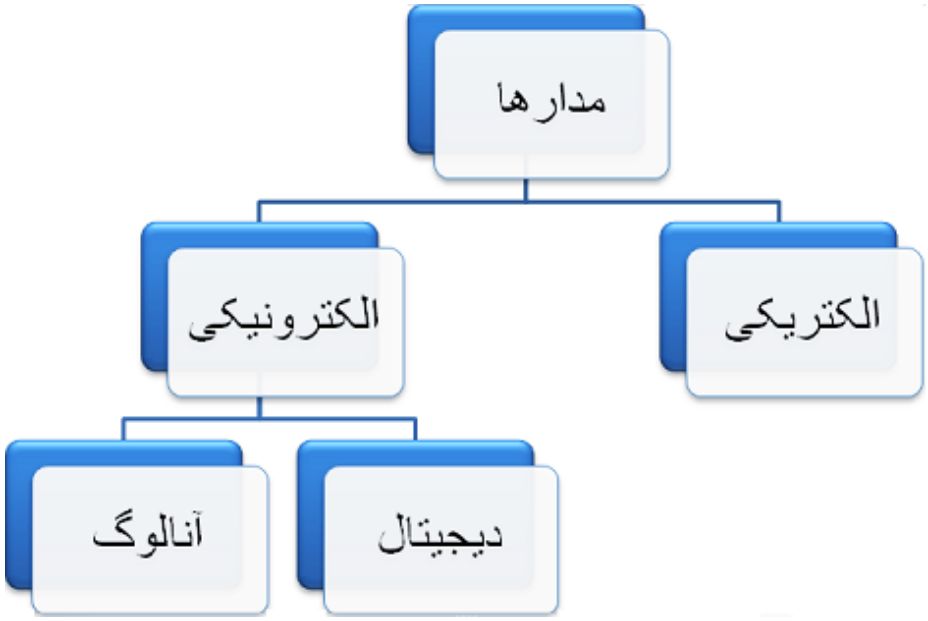
ضمنا کلیه آموزش های این جزوه در سایت الکترو ولت به صورت بخش به بخش آورده شده است که میتوانید زمانی که این جزوه در اختیار نبود، از آن استفاده نمایید. برای یادگیری حرفه ای و پیشرفته پیشنهاد می شود بسته عظیم آموزش میکروکنترلرهای AVR که شامل پروژه های شبیه سازی و عملی بسیاری به همراه توضیحات است را تهیه نمایید.

۱-۲-مدارهای الکتریکی و الکترونیکی

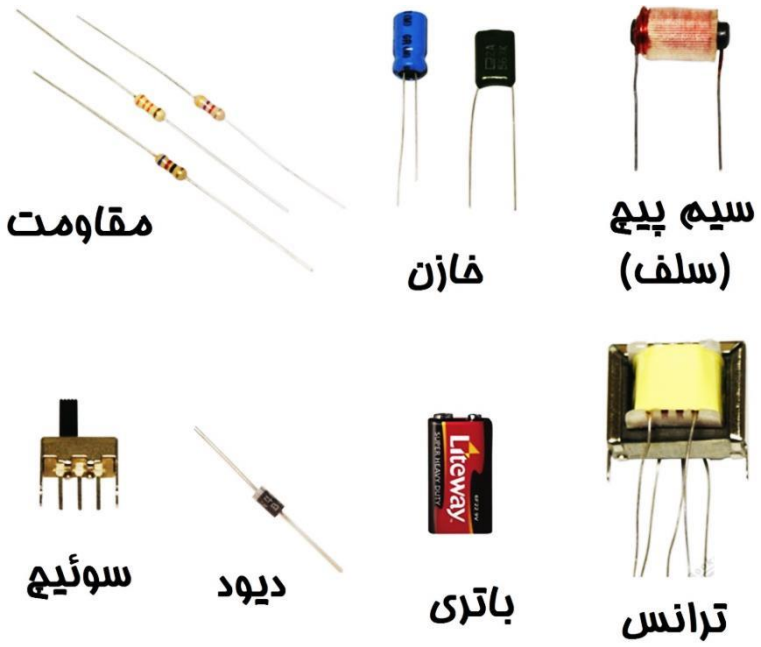
مدارهای الکترونیکی به همراه مدارهای الکتریکی دو دسته کلی از مدارات به‌شمار می‌روند.

اگر عنصرهای تشکیل دهنده مدار ، الکتریکی باشند، مدار "الکتریکی" نامیده می‌شود، و اگر عنصرهای تشکیل دهنده مدار الکتریکی و الکترونیکی باشند، مدار را "الکترونیکی" گویند.

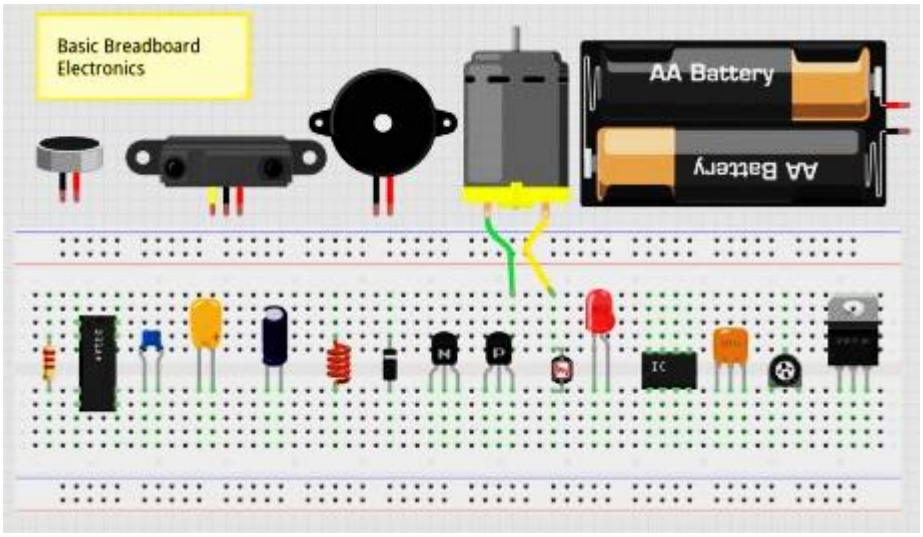
مدارهای الکترونیکی خود به دو دسته مدارهای الکترونیکی آنالوگ و مدارهای الکترونیکی دیجیتال تقسیم بندی می شوند. شکل زیر تقسیم بندی کلی مدارها را نشان می دهد.



مدارهای الکتریکی از به‌هم پیوستن المان‌های الکتریکی یا غیر فعال (مانند باتری ، مقاومت، خازن، سلف، لامپ، دیود و ...) تشکیل می شوند. به طوری که حداقل یک مسیر بسته را ایجاد کنند و جریان الکتریکی بتواند در این مسیر بسته جاری شود. شکل زیر برخی از المان های پرکاربرد در "مدارهای الکتریکی" را نشان می دهد.

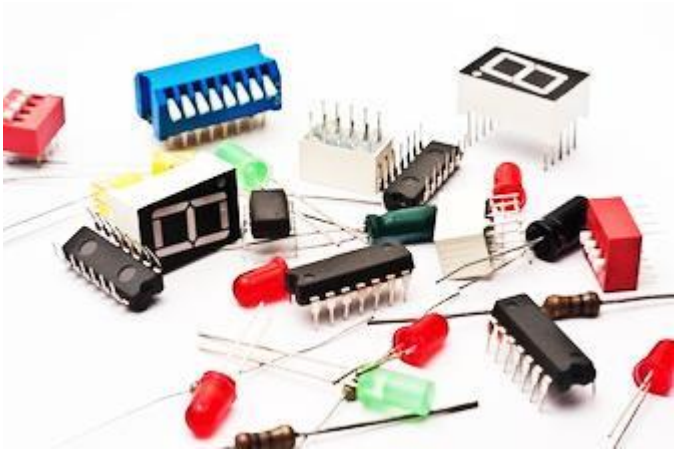


مدارهای الکترونیکی از به هم پیوستن المان‌های الکتریکی یا المانهای الکترونیکی یا ترکیبی از هر دو بوجود می‌آید. به طوری که حداقل یک مسیر بسته را ایجاد کنند و جریان الکتریکی بتواند در این مسیر بسته جاری شود. شکل زیر اجزای یک مدار الکترونیکی را نشان می دهد که علاوه بر المان های الکتریکی ، المان هایی نظیر ترانزیستور ، آی سی (چیپ) و موتورهای الکترونیکی نیز در آن دیده می 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 (ولتاژ الکتریکی برابر است با حاصل ضرب مقاومت الکتریکی در جریان الکتریکی)

برعکس مدارهای الکتریکی، مدارهای الکترونیکی 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 ، معادلات دیفرانسیل بسیار سخت و پیچیده ایجاد می‌شوند و به همین دلیل غالبا از تقریب برای قسمت‌های الکترونیکی استفاده می‌شود. مدارهای الکترونیکی خود به دو دسته دیجیتال (رقمی) و آنالوگ (قیاسی) تقسیم می‌شوند.



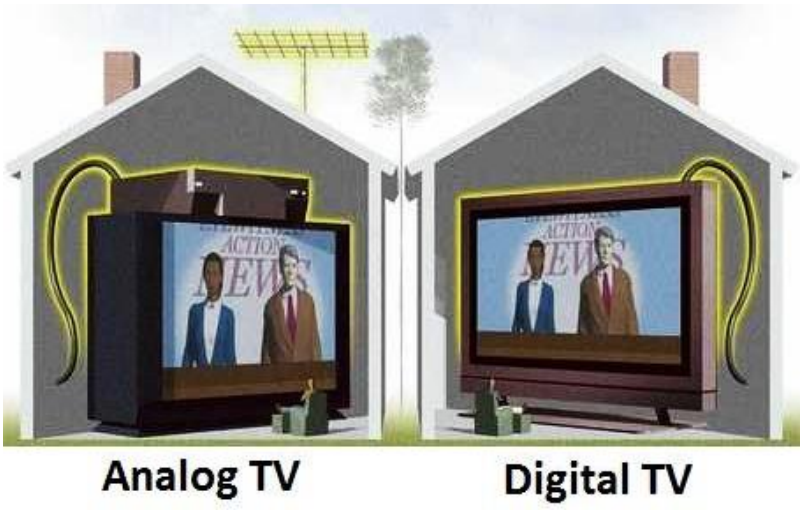
۱-۲-۱- سیستم های الکترونیکی

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.

تمامی این اعمال گفته شده ، می تواند به صورت آنالوگ و توسط سیستم الکترونیکی آنالوگ صورت گیرد و یا به صورت دیجیتال و توسط پردازنده های دیجیتالی و سیستم دیجیتال صورت گیرد. بنابراین یک سیستم الکترونیکی مانند تلویزیون میتواند به صورت یک سیستم آنالوگ یا سیستم دیجیتال کار کند.



۱-۲-۲- تفاوت سیستم دیجیتال با آنالوگ

امروزه استفاده از سیستم های الکترونیکی دیجیتال در همه ابعاد گسترش یافته است و بر سیستم های مبتنی بر طراحی آنالوگ برتری دارد. برتری ویژه‌ی سیستم‌های دیجیتال بر آنالوگ، امروزه باعث شده تمایل بسیاری برای طراحی بر پایه‌ی سیستم‌های دیجیتال به‌وجود آید. در یک سیستم دیجیتال تمامی داده ها و اطلاعات به صورت کدهای ۰ یا ۱ نمایش، پردازش و ذخیره می شود که باعث می شود مزیت هایی برای این سیستمها بوجود آید :

- **قابل برنامه ریزی بودن :** به سادگی برای اجرای هر الگوریتمی قابل تغییر و برنامه ریزی هستند.
- **سرعت بالاتر :** سرعت انجام پردازش های گوناگون بسیار بیشتر و قوی تر است.
- **دقت بیشتر و کاهش خطا :** سیگنال های دیجیتال دارای دقت بالاتر هستند و در نتیجه امکان خطای پردازش کمتری دارند.
- **حجم کمتر و توان پایین تر :** سیستم های دیجیتال روز به روز به سمت کوچکتر شدن و توان مصرفی پایین تر پیش می روند.

۱-۳- آشنایی با اجزای مدارهای الکترونیکی

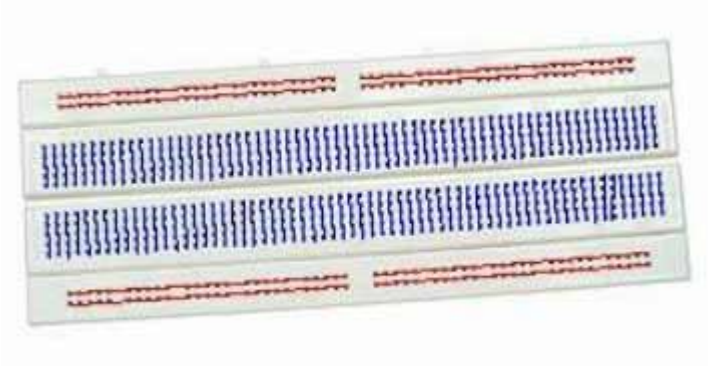
در این قسمت به آشنایی با ابزارها ، المان های مدار ، اصول اولیه و پیشنیازهای مورد نیاز برای کار با مدارهای دیجیتال خواهیم پرداخت.

۱-۳-۱- 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.

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.

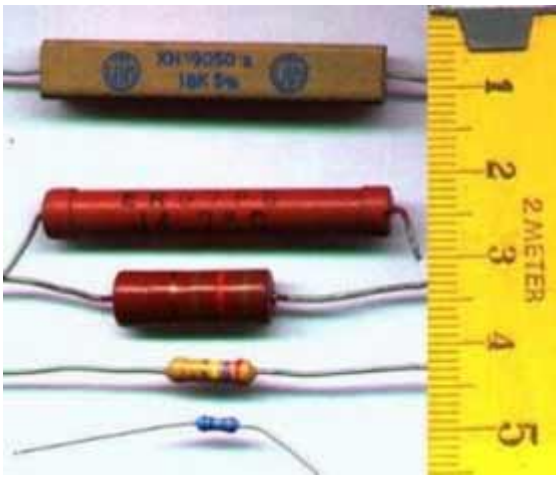


نکته ۱ : برای اتصال دو نقطه دور از هم باید از سیم های مفتولی استفاده کرد که دو سر آن به کمک سیم چین لخت شده باشد.

نکته ۲ : برای ساخت مدارهای بزرگتر میتوان برد بورد ها را به یکدیگر متصل کرد.

۱-۳-۲- 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

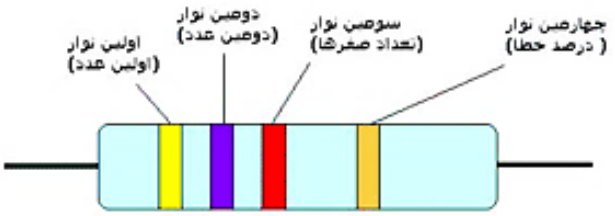
روش اول برای مقاومتهای با خطای ٪۵ به بالا استفاده می‌شود و روش دوم برای مقاومتهای دقیق و خیلی دقیق (با خطای کمتر از ٪۵) استفاده می‌شود.
در اینجا به روش اول که معمول تر است می‌پردازیم.

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.

۱-۳-۳- 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:

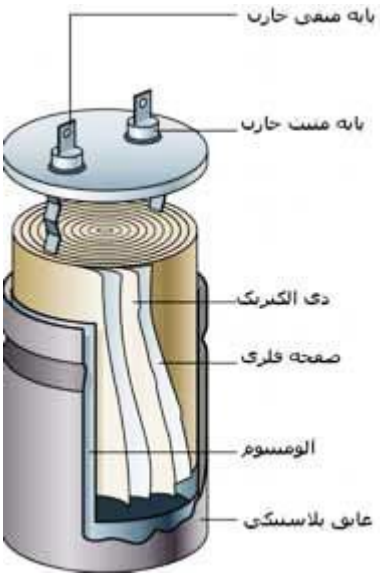
الف – صفحات رسانا

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.

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.

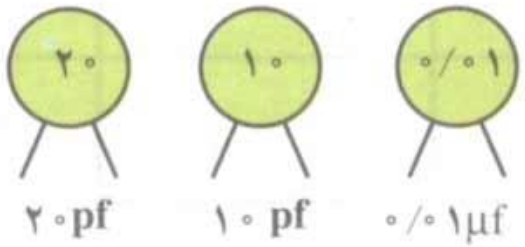
ظرفیت نوع الکترولیتی معمولا در حدود میکروفاراد است (میکرو یعنی ۱۰ به توان منفی شش). در خازن های الکترولیتی همانند شکل زیر پایه مثبت بلندتر است و پایه منفی با نوار مشخص شده است . پایه مثبت حتما می بایست به ولتاژ مثبت تر وصل شود در غیر این صورت خازن میسوزد و خطرناک است.

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

نکته مهم : 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 پیچیده تر 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:



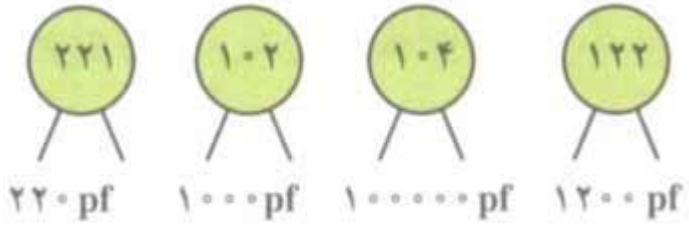
در حالتی که بر روی خازنی اعداد یک رقمی یا دو رقمی مشاهده گردید، مقدار واقعی ظرفیت ، همان عددی است که بر روی آن نوشته شده و واحد آن پیکو فاراد است. اما اگر عدد سه‌رقمی بود، در این حالت اگر آخرین رقم صفر بود، به همان ترتیب بالا عمل می‌کنیم و مقدار ظرفیت همان عدد است. اما اگر آخرین رقم عدد دیگری غیر از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

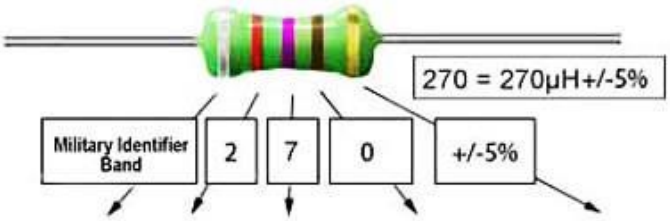


۱-۳-۴- آشنایی با سلف

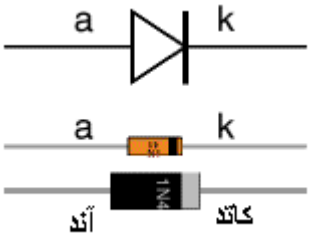
سلف (inductor) یا سیم پیچ (کویل Coil)، یک المان الکترونیکی است که در مقابل تغییرات جریان الکتریکی از خود مقاومت نشان می دهد. سلف دارای یک رسانا مانند سیم است که به صورت سیم پیچ درآمده است . هنگام عبور جریان از سلف ، انرژی به صورت میدان مغناطیسی موقت ذخیره می شود. در نتیجه سلف همانند خازن است با این تفاوت که خازن بار الکتریکی (ناشی از میدان الکتریکی) را در خود ذخیره می کند اما سلف شار الکتریکی (ناشی از میدان مغناطیسی) را در خود ذخیره می کند.

نتیجه : سلف و خازن هر دو از المان های ذخیره کننده انرژی در مدار هستند.

ظرفیت سلف بر حسب واحدی به نام هانری H بیان می شود. چون هانری واحد بزرگی است ، بیشتر از واحد های کوچکتر میکرو و میلی هانری استفاده می شود. سلف ها دارای انوع مختلفی هستند. بخش کمی از این سلف ها شبیه به مقاومت بوده و روش خواندن مقدار ظرفیت آن از روی کد های رنگی دقیقا شبه خواندن مقاومت است با این تفاوت که مقاومت بر حسب اهم است اما مقدار سلف در این روش بر حسب میکرو هانری uH بدست می آید.

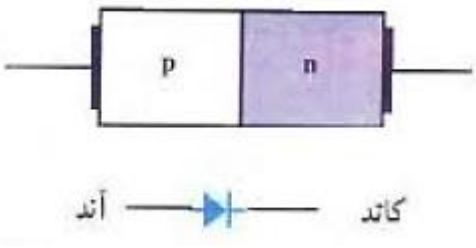


۱-۳-۵- آشنایی با دیود



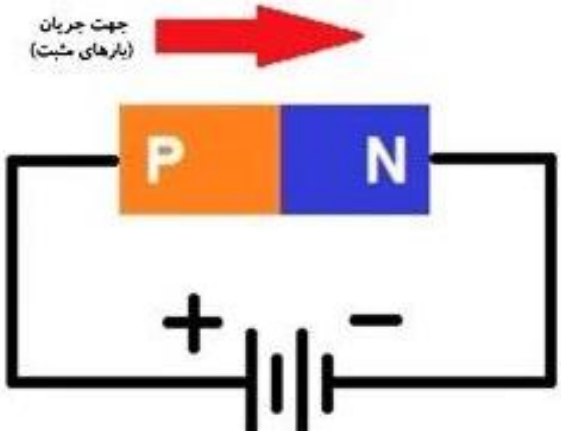
دیود (Diode) قطعه‌ای نیمه رسانا از جنس سیلیسیم یا ژرمانیم است که جریان الکتریکی را در یک جهت از خود عبور می‌دهد و در جهت دیگر در مقابل عبور جریان از خود مقاومت بسیار بالایی نشان می‌دهد. مهمترین کاربرد دیود عبور دادن جریان در یک جهت و ممانعت در برابر عبور جریان در جهت مخالف است. در نتیجه می‌توان به دیود 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 (پیوند p-n) حاصل می شود.



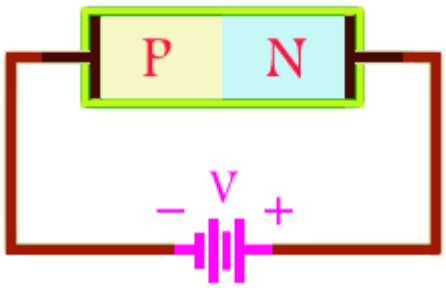
بایاس کردن اتصال P–N : هرگاه به دو سر اتصال P–N ولتاژی اعمال کنیم گوییم آن را بایاس نموده ایم . بایاس کردن اتصال P–N به دو صورت مستقیم و معکوس انجام می گیرد .

بایاس مستقیم (Forward Bias) : اگر قطب مثبت منبع تغذیه را به نیمه هادی نوع P و قطب منفی منبع تغذیه را به نیمه هادی نوع Nوصل کنیم ، دیود را در بایاس مستقیم یا موافق قرار داده ایم . در شکل زیر بایاس مستقیم دیود نمایش داده شده است.



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 ۰،۶ تا ۰،۷ ولت می‌باشد.

بایاس معکوس (Reverse Bias) : اگر قطب مثبت منبع تغذیه را به کریستال نوع N و قطب منفی آن را به کریستال نوع P متصل کنیم، دیود را در بایاس معکوس یا مخالف قرار داده ایم. در شکل زیر بایاس معکوس دیود نمایش داده شده است.



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.

نتیجه : در حالت ایده آل میتوان دیود را در حالت بایاس معکوس مدار باز (قطع) و در حالت بایاس مستقیم اتصال کوتاه (وصل) در نظر گرفت.

۱-۳-۶- 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 LED روشن 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 این نکته و از رابطه V=RI (قانون اهم) میتوان مقدار مقاومت مناسب را محاسبه کرد.

۱-۳-۷- 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 ۳-۱۲ 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
mA۱۰۰۰
به معنی جریان خروجی آداپتور است که هر چقدر بیشتر باشد ، آداپتور قوی تر بوده و می تواند وسایل بزرگتری را تغذیه کند.

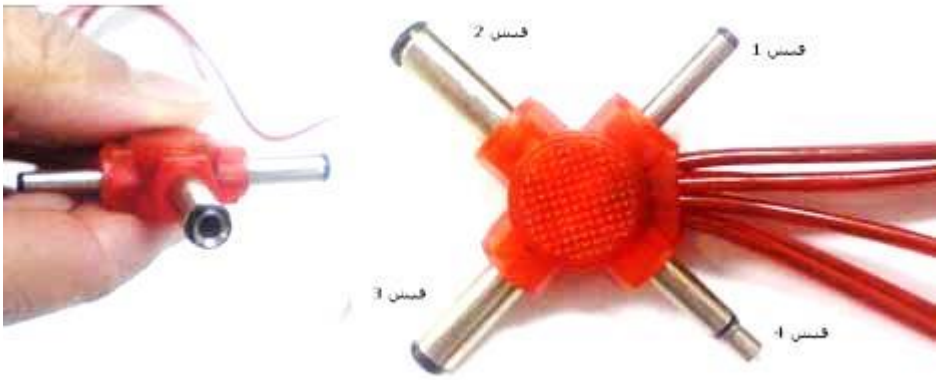
برای مثال لامپ چشمک زن ماشین ۵۰۰ میلی آمپر جریان نیاز دارد ؛ لامپ های خطر ماشین ۱۸۰۰ میلی آمپر نیاز دارد و لامپهای چراغ جلو در حدود ۸۰۰۰ میلی آمپر جریان نیاز دارد . در نتیجه با یک آداپتور ۱۰۰۰ میلی آمپر حداکثر میتوان وسیله ای را که به ۱۰۰۰ میلی آمپر برای روشن شدن احتیاج دارد، تغذیه کرد.
آداپتوری که در این پروژه استفاده می کنیم ، 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.

به صورت کلی آداپتورها دارای دو نوع فیش هستند:

§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

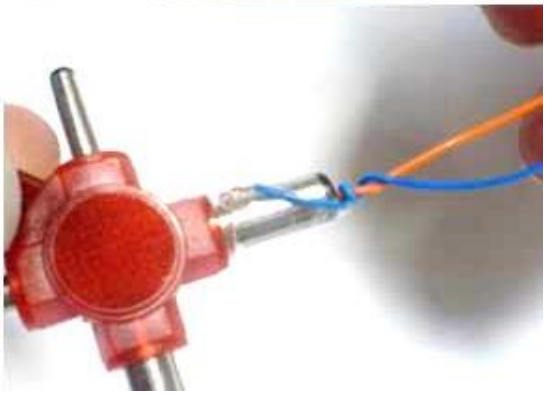


با اتصال سیمها به هر کدام از این فیش ها می توانیم مثبت و منفی آداپتور را از آن بگیریم.

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

نکته : ولتاژ منفی منبع تغذیه را GND (زمین Ground) می گویند و ولتاژ مثبت منبع تغذیه را Vcc نامگذاری می کنند. شکل زیر نمادهای مختلف به کار رفته در نقشه های مداری برای Vcc و GND را نشان می دهد.

همچنین برای منبع تغذیه مدار به جای آداپتور میتوان از یک باتری کتابی ۹ ولت به همراه یک آی سی رگولاتور (برای تنظیم ولتاژ خروجی) استفاده کرد.

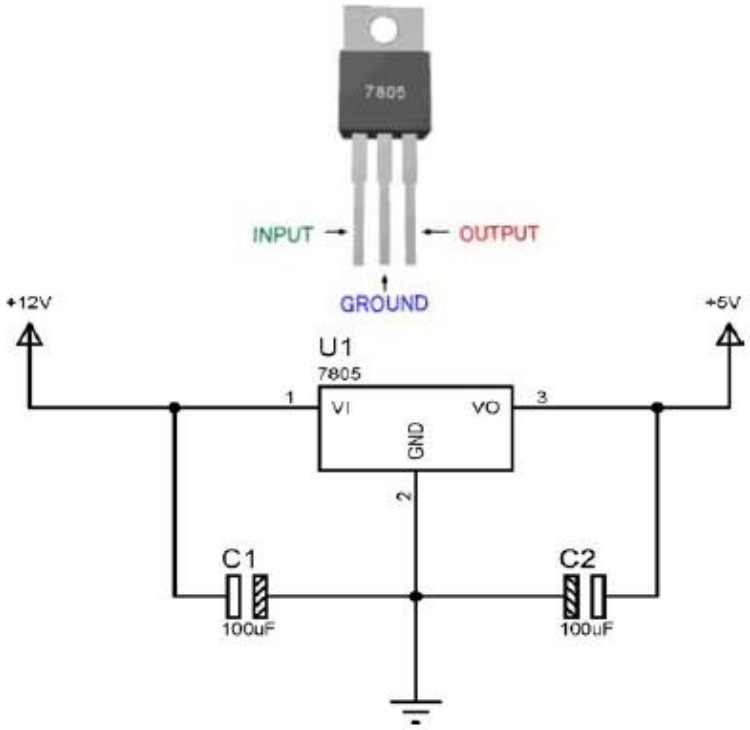
همچنین می توان از پورت USB نیز برای منبع تغذیه مدار استفاده کرد . در این حالت خروجی پورت USB، ۵ ولت بوده و احتیاجی به رگولاتور نمی باشد. پورت USB دارای ۴ سیم به صورت شکل زیر است که از دو سیم آن برای تغذیه استفاده می شود.

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

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

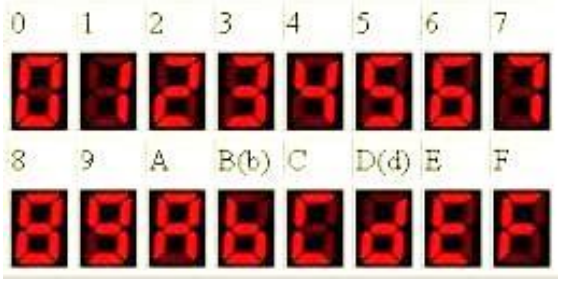
۱-۳-۸- رگولاتور یا تنظیم کننده ولتاژ:

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 78XX می 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.

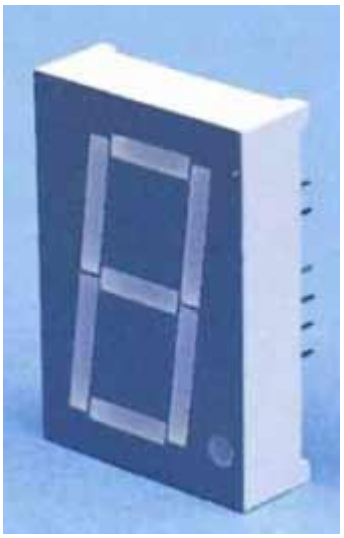


۱-۳-۹- آشنایی با سون سگمنت

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. مثل مایکروویو ، ترازوی دیجیتال ، آسانسور و...
سون سگمنت در واقع ۷ تا LED است که یکی از پایه های آنها با هم یکی شده.



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.

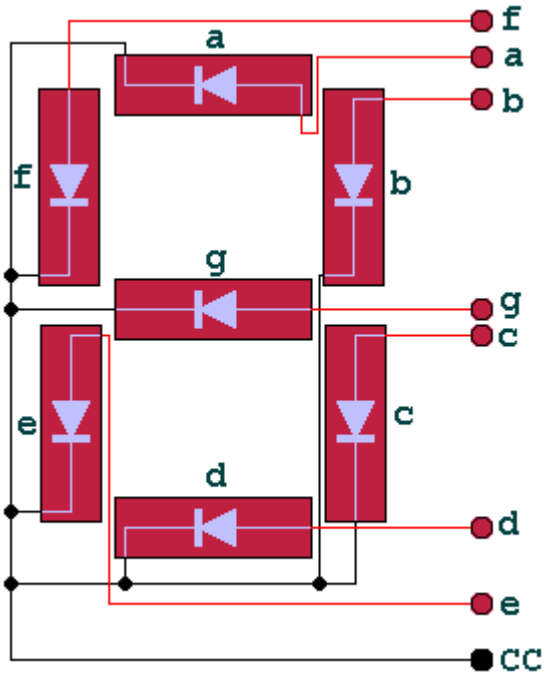


انواع سون سگمنت :

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)

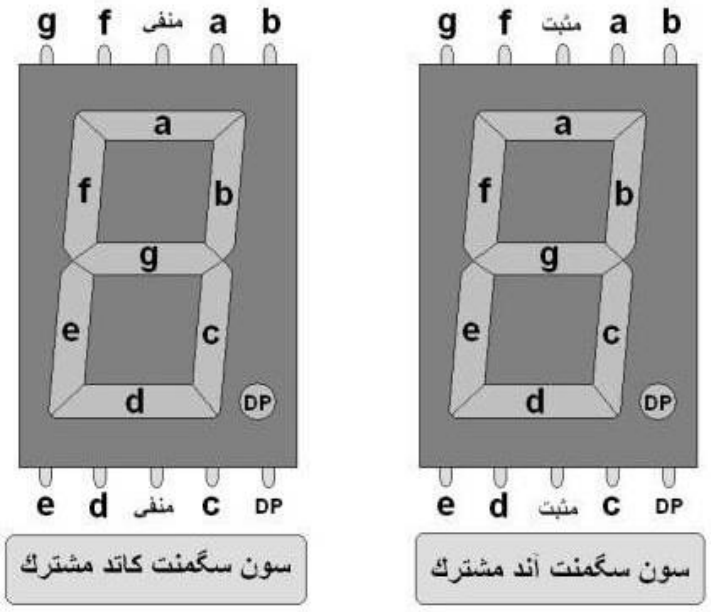
برای درک بهتر ، در شکل زیر مدار داخلی یک سون سگمنت کاتد مشترک را نشان داده شده است.



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

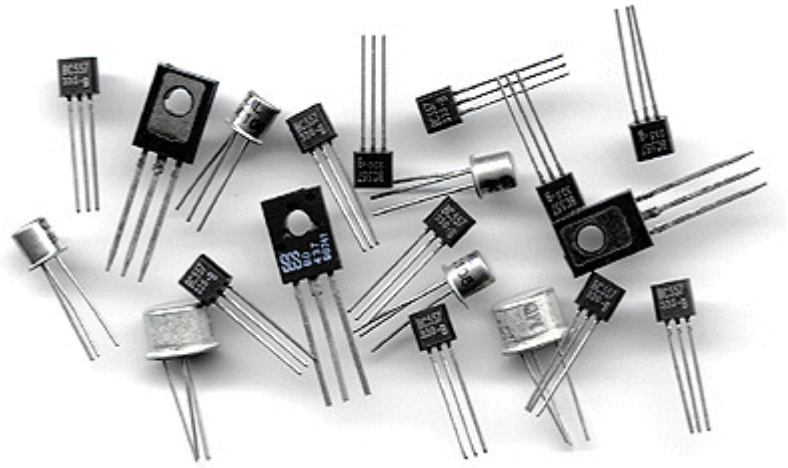
کدام از پایه های دیگر به منفی منبع تغذیه متصل گردد ، LEDمربوط به آن پایه روشن می شود.
در سون سگمنت های کاتد مشترک هم همین قانون حاکم است فقط باید توجه نمود که پایه مشترک از طریق مقاومت به مثبت منبع تغذیه وصل شود و پایه های دیگر به منفی.

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



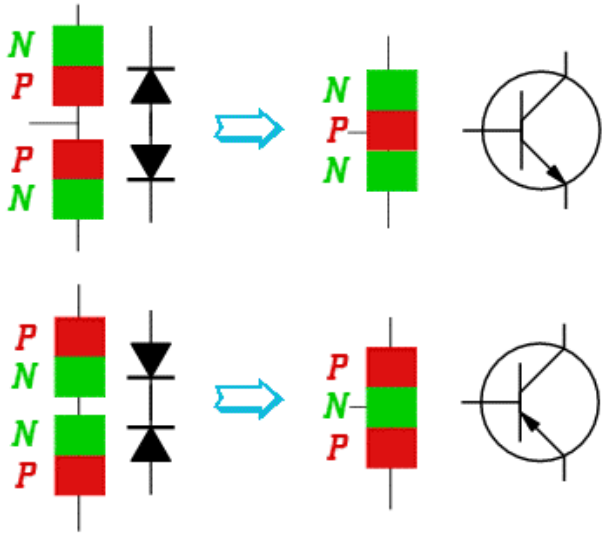
نکته مهم : سون سگمنت برای نور دهی مناسب نیاز به ولتاژ حدود ۳ ولت و جریان تقریبی ۲۰ میلی آمپر برای هر سگمنت دارد. در نتیجه برای محدود کردن ولتاژ و جریان در سون سگمنت یک راه استفاده از مقاومت بین پایه مشترک و منبع تغذیه است و راه دیگر استفاده از دیود بین پایه مشترک و منبع تغذیه است.

۱-۳-۱۰- آشنایی با ترانزیستور

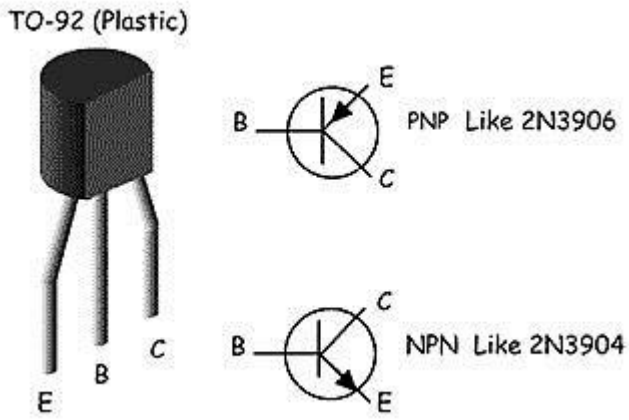


ترانزیستورها یکی از مهمترین و پرکاربردترین قطعات الکترونیکی می باشند که از پیوندهای مواد نیمه رسانا (مانند سیلیسیم و ژرمانیم) به صورت pnp یا npn تشکیل شده است. ترانزیستورها به دو دسته کلی تقسیم می‌شوند: ترانزیستورهای اتصال دوقطبی (BJT) و ترانزیستورهای اثر میدانی(FET) .

ترانزیستور را میتوان همانند دو دیود به هم چسبیده در نظر گرفت.

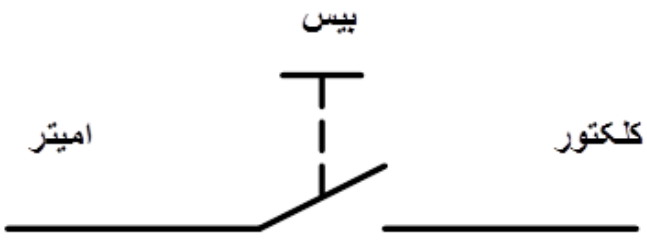


یک ترانزیستور BJT دارای سه پایه به نام های بیس (Base) ، کلکتور (Collector) و امیتر (Emitter) (می باشد. در عمل وقتی از روبرو به قسمت صاف ترانزیستور نگاه می کنید ، پایه سمت چپ امیتر است.



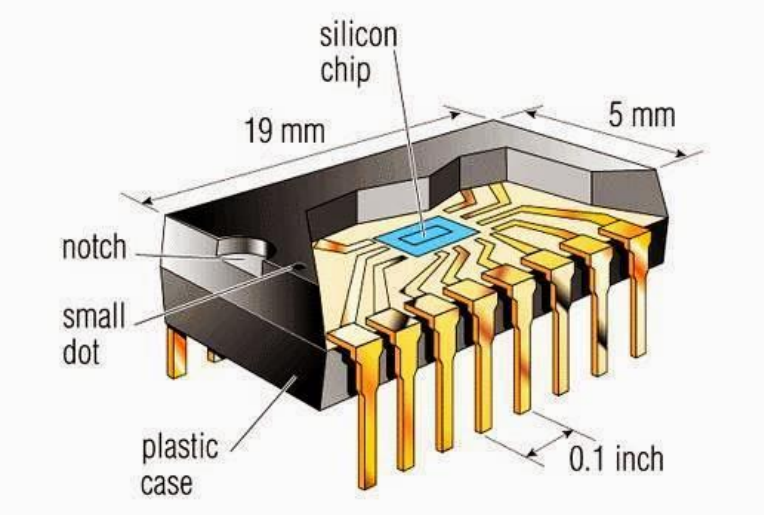
در مدارهای آنالوگ، ترانزیستورها به عنوان تقویت کننده جریان یا ولتاژ استفاده می‌شوند و در مدارات دیجیتال از آنها بعنوان یک سوئیچ الکترونیکی استفاده می‌شود.

شکل زیر مدل یک ترانزیستور دو قطبی ، 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 bjtتنها بین دو حالت ۰ و ۱ (قطع و وصل) کار می کند.



۱-۳-۱۱- آشنایی با آی سی

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.

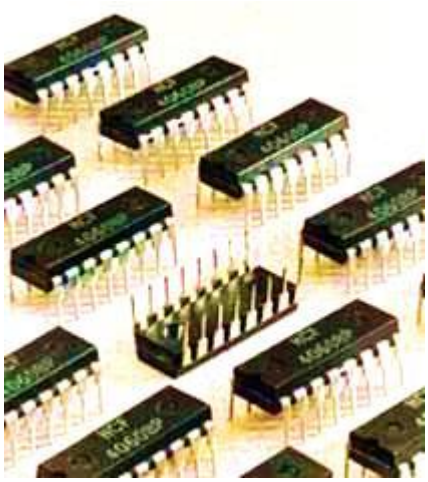


تعدا پایه ها ممکن است از ۸ پایه برای آی سی های کوچک تا ۶۴ پایه و بیشتر برای آی سی های بزرگ متغییر باشد.به منظور شناسائی هر آی سی ، روی آن شماره ای چاپ می شود و تولید کنندگان کتابچه هایی چاپ می کنند که اطلاعات مربوط به هر آی سی در آن وجود دارد. یک راه دیگر دستیابی به اطلاعات هر آی سی ، جستجو در اینترنت (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 CMOS هستند. این نامها بصورت مخفف متداول هستند و در اصل به این صورت می باشند:

TTL : Transistor-Transistor-Logic

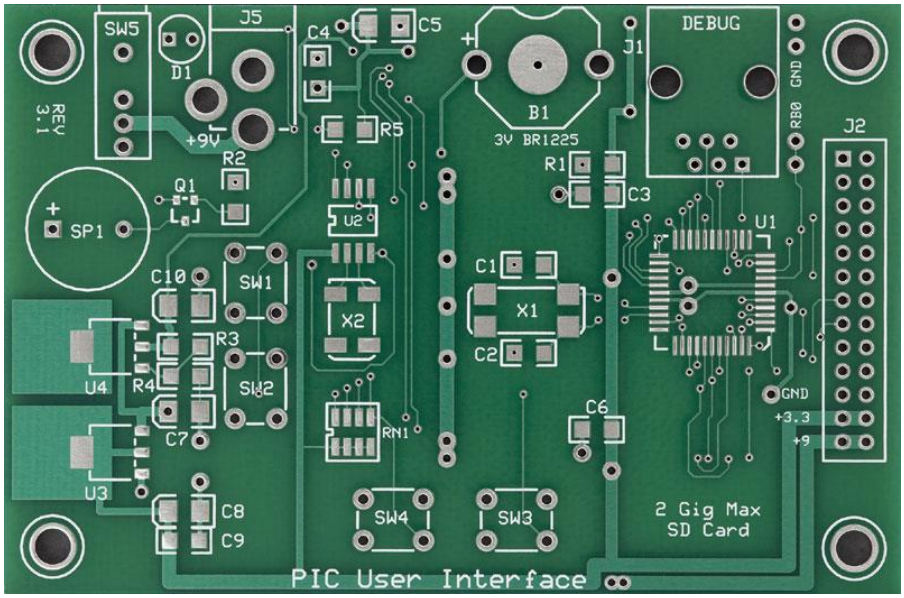
CMOS : Complementary Metal-Oxide Semiconductors

در ساخت آی سی های سری TTL از ترانزیستورهای BJT و در آی سی های خانواده CMOS ، از ترانزیستورهای اثر میدانی (FET) استفاده شده است که سریع تر و کم مصرف تر از BJT هستند.

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.

۱-۳-۱۲- برد مدار چاپی (pcb)

pcb مخفف printed circuit board و به معنای برد مدار چاپی می باشد. بُرد مدار چاپی که در کیت های آموزشی ، برد های صنعتی و یا هر دستگاه الکترونیکی مورد استفاده قرار می گیرد، شامل مجموعه ای از مدارهای الکتریکی بوده و می‌تواند یک طرفه (یک لایه مس)، دو طرفه (دو لایه مس) و یا حتی چند لایه باشد؛ بطوریکه قطعات الکترونیکی مانند مقاومت، خازن، آی سی و ... بر روی آن مونتاژ شده و جهت استفاده در تجهیزات الکترونیکی بکار می رود. شکل زیر یک نمونه pcb قبل از سوارشدن قطعات و مونتاژ آنها را نشان می دهد.



۱-۳-۱۳- المان های نصب سطحی (smd)

SMD مخفف Surface mount device و به معنی قطعات نصب شده روی سطح می باشد.
قطعات نصب سطحی ؛ قطعات الکترونیکی هستند که به مدار چاپی به روش فناوری نصب سطحی-Surface mount technology متصل شده اند. اکثر المان های مداری از قبیل خازن ، مقاومت ، سلف ، آی سی و... علاوه بر بسته بندی دارای پایه (DIP) ، بسته بندی SMD نیز دارند. شکل زیر انواع مختلف المان های SMD را نشان می دهد.

۱-۳-۱۴- آشنایی با کریستال

یکی از قطعه های به کار رفته در مدارهای دیجیتال می باشد. همانطور که از اسم آن مشخص است درون این قطعه یک ماده کریستالی همانند کوارتز قرار دارد که با برقراری ولتاژ دو سر آن شروع به نوسان می کند. از این قطعه در تولید موج دیجیتالی با فرکانس معین استفاده می شود. مثلا از کریستال ساعت (۳۲۷۶۸ هرتز) برای تولید دقیق یک ثانیه استفاده می شود. شکل زیر انواع کریستال ها را نشان می دهد.

با قرار دادن یک کریستال در مدار ، فرکانس خاص آن کریستال تولید می شود. این فرکانس خاص بر روی بدنه کریستال (با در نظر گرفتن دقت آن) قید شده است. به عنوان مثال کریستالی با ارقام ۴,۰۰۰ مگاهرتز ، فرکانس ۴ مگاهرتز را تولید می کند که تا رنج کیلوهرتز دقیق عمل می کند، اما در رنج هرتز دارای درصدی خطا می باشد (به صفر ها دقت کنید، اگر ۶ صفر بود به این معنا بود که کریستال تا رنج ۱ هرتز نیز دقیق عمل می کند). مثلا فرکانس تولیدی ۴۰۰۰۲۰۸ هرتز خواهد بود که ۲۰۸ هرتز خطا دارد.

۱-۴- اصول 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

۱-۴-۱- تفاوت الکترونیک آنالوگ و دیجیتال

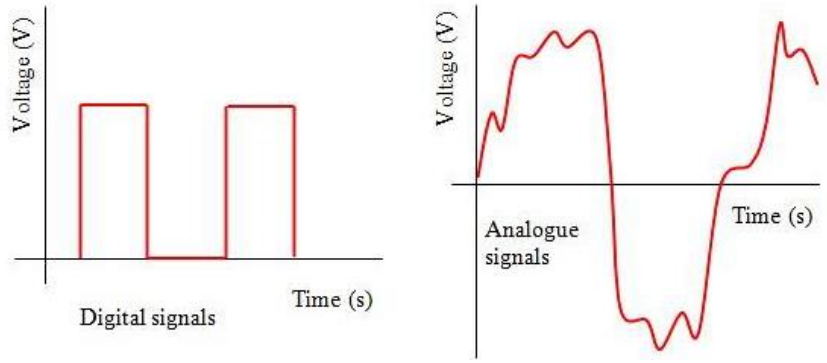
الکترونیک آنالوگ به سیستم های الکترونیکی گفته می‌شود که با سیگنال‌های پیوسته کار می‌کنند. برخلاف سیستم های الکترونیک دیجیتال که فقط از دو حالت ۰ و ۱ استفاده می‌کنند. واژه «آنالوگ» به روابط نسبی ما بین یک سیگنال و یک ولتاژ یا یک جریان برق اشاره می‌کند.
الکترونیک دیجیتال شامل مدارت و سیستم هایی است که 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 = 0وHigh = 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).

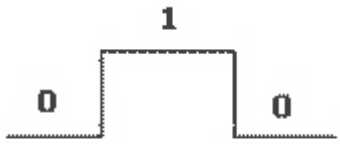
۱-۴-۲- تعریف سیگنال دیجیتال

سیگنال دیجیتال، سیگنالی است که هم از نظر زمان رخداد و هم از نظر مقدار در بازه 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.

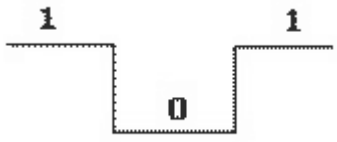


بنابراین سیگنال دیجیتال مجموعه ای از صفرها و یک های منطقی است که در واحد زمان عبور می کند . زمانی که سطح ولتاژ از Low به High می رود یک "لبه بالارونده" ایجاد می شود و سپس زمانی که از High به Low بازگردد "لبه پایین رونده" ایجاد می شود.

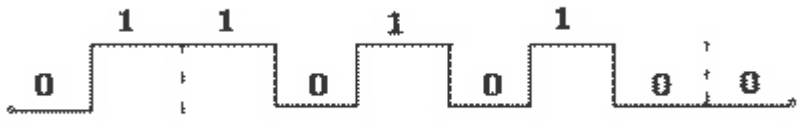
زمانی که سیگنال از Low به High رفته و بازگردد یک پله مثبت ایجاد می شود که به آن "پالس مثبت" می گویند .



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.



۱-۴-۳- 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

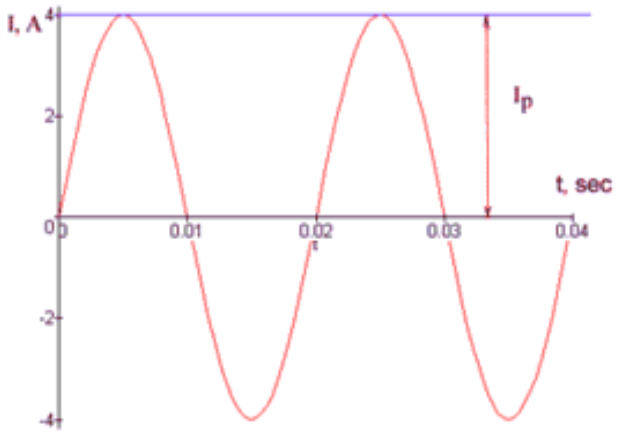
خط کش متصل به میز : به طول آن بستگی دارد ، بین ۵ تا ۱۰۰ بار

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)

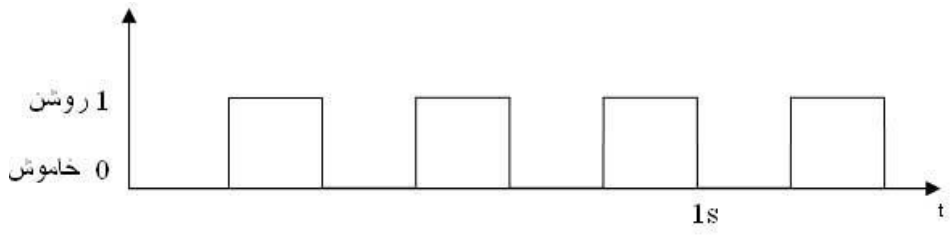
نکته : رابطه فرکانس با زمان یک رفت و برگشت (یک پریود) به صورت عکس است($f=1/T$)

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



فرکانس در سیگنالهای دیجیتال :

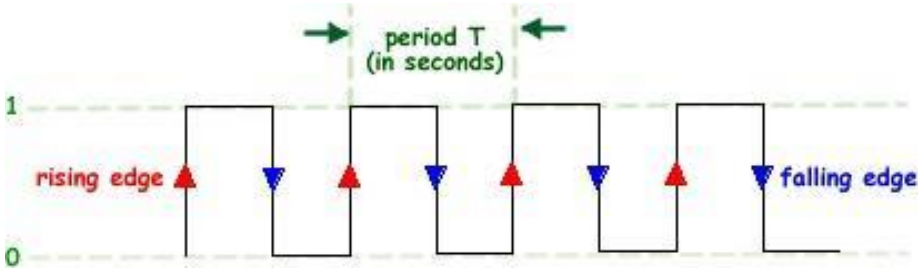
در سیستم دیجیتال ، رفت و برگشت معادل است با ۰ و ۱ شدن یک سیگنال. در نتیجه فرکانس یک سیگنال برابر می شود با تعداد ۰ و ۱ شدن ها در یک ثانیه و عکس فرکانس آن نیز دوره تناوب را نشان می دهد . مثلا در شکل زیر یک سیگنال منظم با فرکانس ۳ هرتز را مشاهده می کنید. زیرا در طول یک ثانیه سه بار سطح ولتاژ High و سپس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



فرکانس 3 هرتز

۱-۴-۴- سیگنال کلاک (Clock)

کلاک پالس یا پالس ساعت که در اکثر 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 edgeو به لبه پایین رونده آن falling edge گفته می شود.



۱-۵- 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 همان1 = "High"
-سطح ولتاژ پایین یا همان0 = "Low"

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.

نتیجه : زبان ماشین فقط ۰ و ۱ است و بنابراین هر چیزی که در یک سیستم الکترونیکی دیجیتال وارد می شود ، پردازش می شود ، ذخیره یا خارج می شود ، در مبنای ۲ انجام می گیرد.

۱–۵–۱– 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:

صدگان هزار	دهگان هزار	یکان هزار	صدگان	دهگان	یکان

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

جدول ارزش مکانی اعداد در مبنای ۲ :

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

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

مثال: عدد ۱۰۰۱۰۱ در مبنای ۲

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

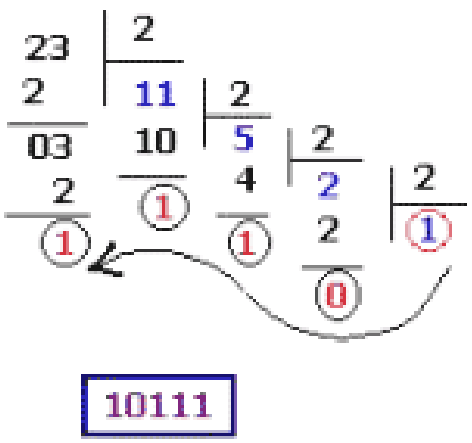
$2^5=32$	$2^4=16$	$2^3=8$	$2^2=4$	$2^1=2$	$2^0=1$
۱	۰	۰	۱	۰	۱

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

۱–۵–۲– 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:



۱–۵–۳– 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
0	**0000**
1	**0001**
2	**0010**
3	**0011**
4	**0100**
5	**0101**
6	**0110**
7	**0111**
8	**1000**
9	**1001**
A	**1010**
B	**1011**
C	**1100**
D	**1101**
E	**1110**
F	**1111**

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 = 52 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:

A3 در مبنای شانزده = ۰۰۱۱ ۱۰۱۰ در مبنای دو

E7 در مبنای شانزده = ۱۱۱۰ ۰۱۱۱ در مبنای دو

۱-۶- تعریف واحد های اندازه گیری حافظه

تمامی حافظه ها یک ظرفیت و گنجایشی دارند که به آن اندازه می توانند اطلاعات را در درون خود ذخیره نمایند. اندازه گیری این ظرفیت توسط واحدهای اندازه گیری حافظه نظیر بیت ، بایت ، کیلوبایت و... مشخص می شود. در واقع بایت واحد اندازه گیری ظرفیت حافظه، هارد دیسک و... می باشد. در زمان مشاهده لیست فایل ها توسط برنامه های نمایش دهنده فایل ها ، ظرفیت یک فایل نیز توسط بایت مشخص می گردد.

تعریف بیت Bit : کوچکترین واحد حافظه است که میتواند یک عدد باینری را در خود ذخیره نماید. بنابراین یک بیت میتواند عدد ۰ یا ۱ را در درون خود ذخیره نماید.

تعریف بایت Byte : هر ۸ بیتی که کنار هم قرار گرفته باشند تشکیل یک بایت را می دهند. با توجه به اینکه هر بیت میتواند ۰ یا ۱ باشد ، بنابراین یک بایت میتواند از ۰۰۰۰۰۰۰۰ (هشت تا ۰) تا ۱۱۱۱۱۱۱۱ (هشت تا ۱) مقدار گیرد که معادل آن در مبنای دهدهی از ۰ تا ۲۵۵ است.

برای سنجش میزان حافظه هایی که دارای بایت های فراوانی می باشند ، از "پیشوند" قبل از نام بایت استفاده می گردد. (کیلو، مگا ، گیگا نمونه هایی ازاین پیشوندها می باشند) جدول زیر این پیشوندها به همراه مخفف و مقدار آن به بایت را نشان می دهد.

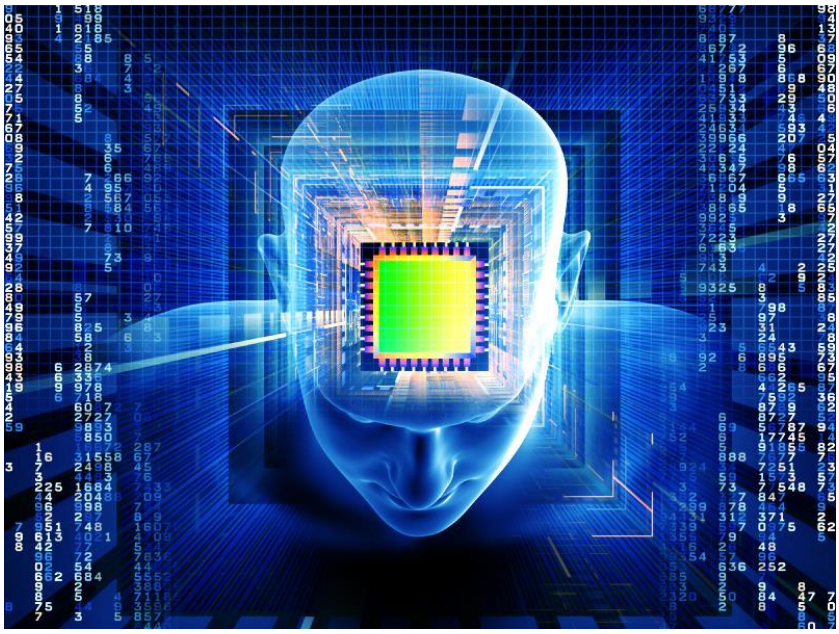
نام	مخفف	اندازه
Kilo	K	2^{10} = 1,024
Mega	M	2^{20} = 1,048,576
Giga	G	2^{30} = 1,073,741,824
Tera	T	2^{40} = 1,099,511,627,776
Peta	P	2^{50} = 1,125,899,906,842,624
Exa	E	2^{60} = 1,152,921,504,606,846,976
Zetta	Z	2^{70} = 1,180,591,620,717,411,303,424
Yotta	Y	2^{80} = 1,208,925,819,614,629,174,706,176

با توجه به جدول فوق می توان چنین برداشت کرد که : کیلو تقریبا معادل "هزار" ، مگا تقریبا معادل "میلیون" ، گیگا تقریبا معادل "میلیارد" و ... است. بنابراین زمانیکه شخصی عنوان می نماید که دارای هارد دیسکی با ظرفیت دو گیگا بایت است ، معنای سخن وی اینچنین خواهد بود : " هارد دیسک وی دارای توان ذخیره سازی دو گیگا بایت ، یا تقریبا دو میلیارد بایت و یا دقیقا 2,147,483,648 بایت است. امروزه استفاده از رسانه های ذخیره سازی با ظرفیت بالا بسیار رایج بوده و ما شاهد حضور و استفاده از بانک های اطلاعاتی با طرفیت بسیار بالا (چندین ترابایت) در موارد متعدد هستیم.

فصل ۲– ساختار میکروکامپیوتر و تفاوت آن با میکروکنترلر

مقدمه

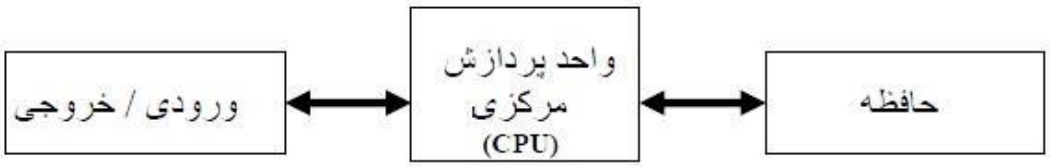
ما در عصری زندگی می کنیم که جامعه شناسان آن را عصر انقلاب کامپیوتر نهاده اند اما آنچه بیش از همه قابل تامل است این است که انقلاب اصلی تنها در ۵۰ سال اخیر و با ظهور ترانزیستور آغاز شده است . انقلابی که نسل های مختلف کامپیوتری را بوجود آورد تا به **نسل چهارم** یعنی **میکروکامپیوترها** رسید. نسلی که مبتنی بر تکنولوژی مدارات مجتمع با فشردگی بسیار زیاد (Very Large Integrated Circuit VLSI) و معماری کامپیوتر بر اساس ریز پردازنده ها (میکروپرسسورها) 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 میکروپرسسور ها، 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 میکرو کنترلرها 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.



۲–۱– تعریف کامپیوتر

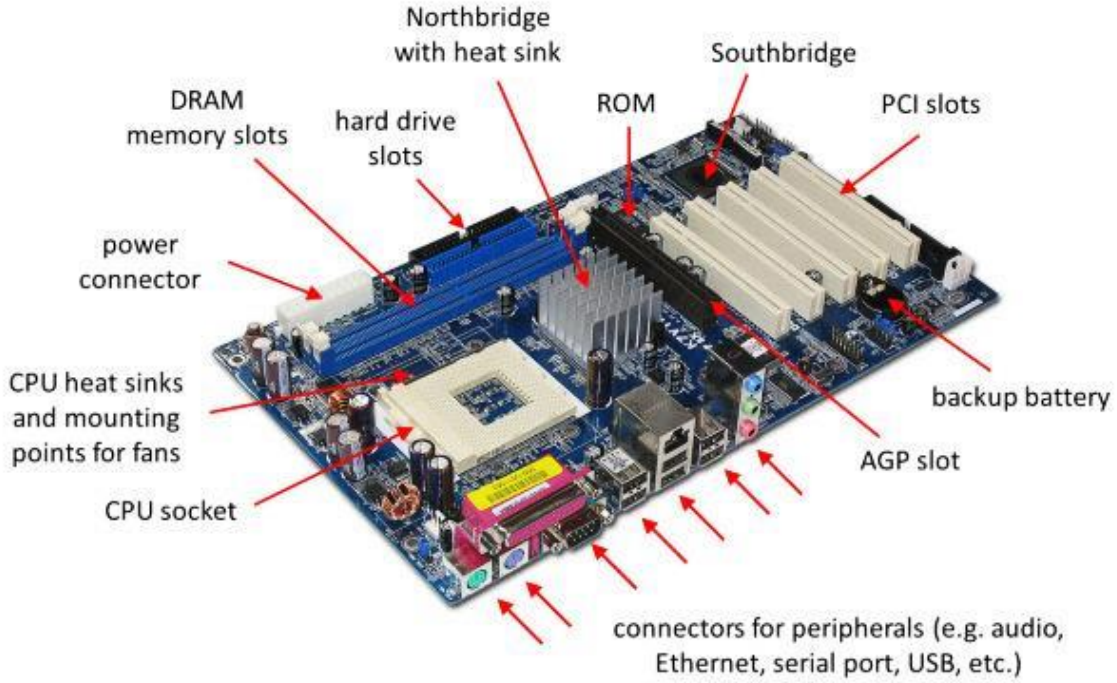
کامپیوتر به معنای محاسبه کردن می باشد و اساس کار آن از روی مغز انسان طراحی و ساخته شده است . کامپیوترهای اولیه بعد از اختراع ترانزیستور ساخته شدند و توانایی انجام محاسبات بسیار ساده را داشتند . اما امروزه از کامپیوترها برای انجام محاسبات پیچیده با سرعت بالا و همچنین برای دخیره و مقایسه اطلاعات

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.



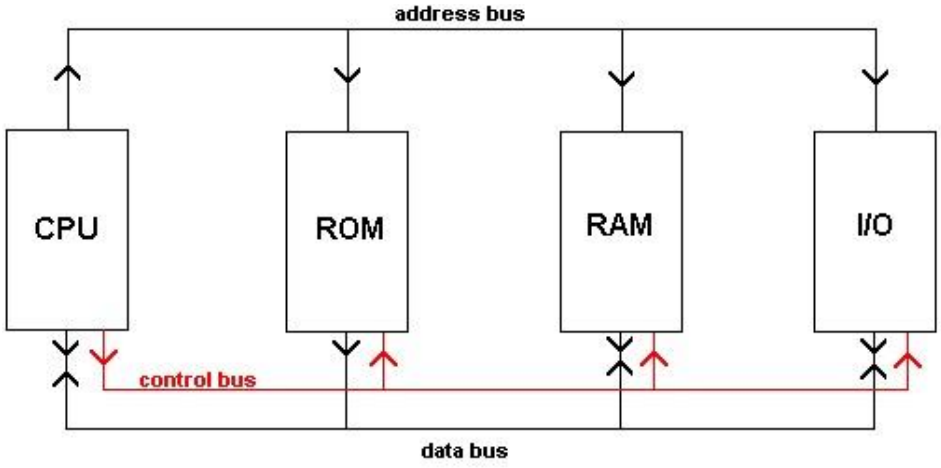
۲-۲- تعریف میکروکامپیوتر

یک سیستم میکروکامپیوتر حداقل شامل میکروپرسسور (CPU) ، حافظه موقت (RAM) ، حافظه دائمی (ROM)و قطعات ورودی/خروجی (PORT) می باشد که بوسیله گذرگاه (باسBUS) 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.



۲-۲-۱- 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

۲-۲-۲- تعریف ROM

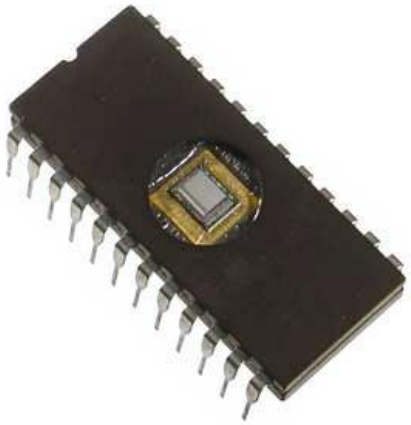
مخفف عبارت Read Only Memory به معنای حافظه فقط خواندنی می باشد. این حافظه دائمی بوده یعنی با قطع برق اطلاعات درون آن از بین نمیرود . برنامه راه اندازی سیستم و سیستم عامل (برنامه کاربر) در این حافظه قرار می گیرد . در بسیاری از کامپیوتر های امروزی بخشی از سیستم عامل روی ROM و بیشتر آن روی هارد دیسک قرار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

۲-۲-۳- 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.

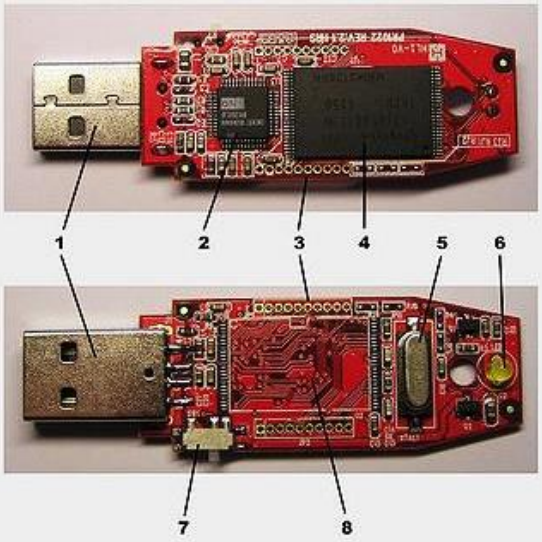
- **PROM:** تولید تراشه های ROM مستلزم صرف وقت و هزینه بالائی است . بدین منظور اغلب تولید کنندگان ، نوع خاصی از این نوع حافظه ها را که Programmable Read Only Memory نامیده می شوند ، تولید می کنند. این نوع از تراشه ها با محتویات خالی و با قیمت مناسب عرضه شده و می تواند توسط هر شخص با استفاده از دستگاه 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 . بمنظور تغییر مقدار یک سلول به صفر، از یک دستگاه خاص پروگرامر استفاده می گردد. حافظه های PROM صرفا یک بار قابل برنامه ریزی هستند ونسبت 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 کارآئی 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 از مدارات از قبل پیش بینی شده در زمان طراحی ، بمنظور حذف استفاده می گردد .در این حالت می توان تمام و یا بخش های خاصی از تراشه را را حذف کرد. بنابراین این نوع حافظه نسبت به حافظه های EEPROM سریعتر است و داده ها در آن داخل بلاک هائی که معمولا ۵۱۲ بایت می باشند ، نوشته می گردند.



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

۲-۲-۴- تعریف RAM

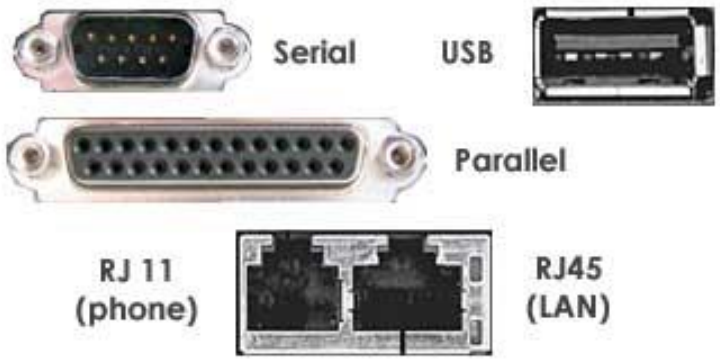
مخفف عبارت Random Access Memory به معنای حافظه با دسترسی تصادفی می باشد. این حافظه ، حافظه موقت می باشد یعنی با قطع برق اطلاعات آن از بین میرود . اطلاعات میانی سیستم شامل متغیرهای برنامه کاربر و داده های مربوط به ورودی/خروجی ها در این حافظه قرار میگیرد.

۲-۲-۵- 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 MOSFETاستفاده 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 SRAMمی باشد.

۲-۲-۶- 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.



۲-۲-۷- 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.

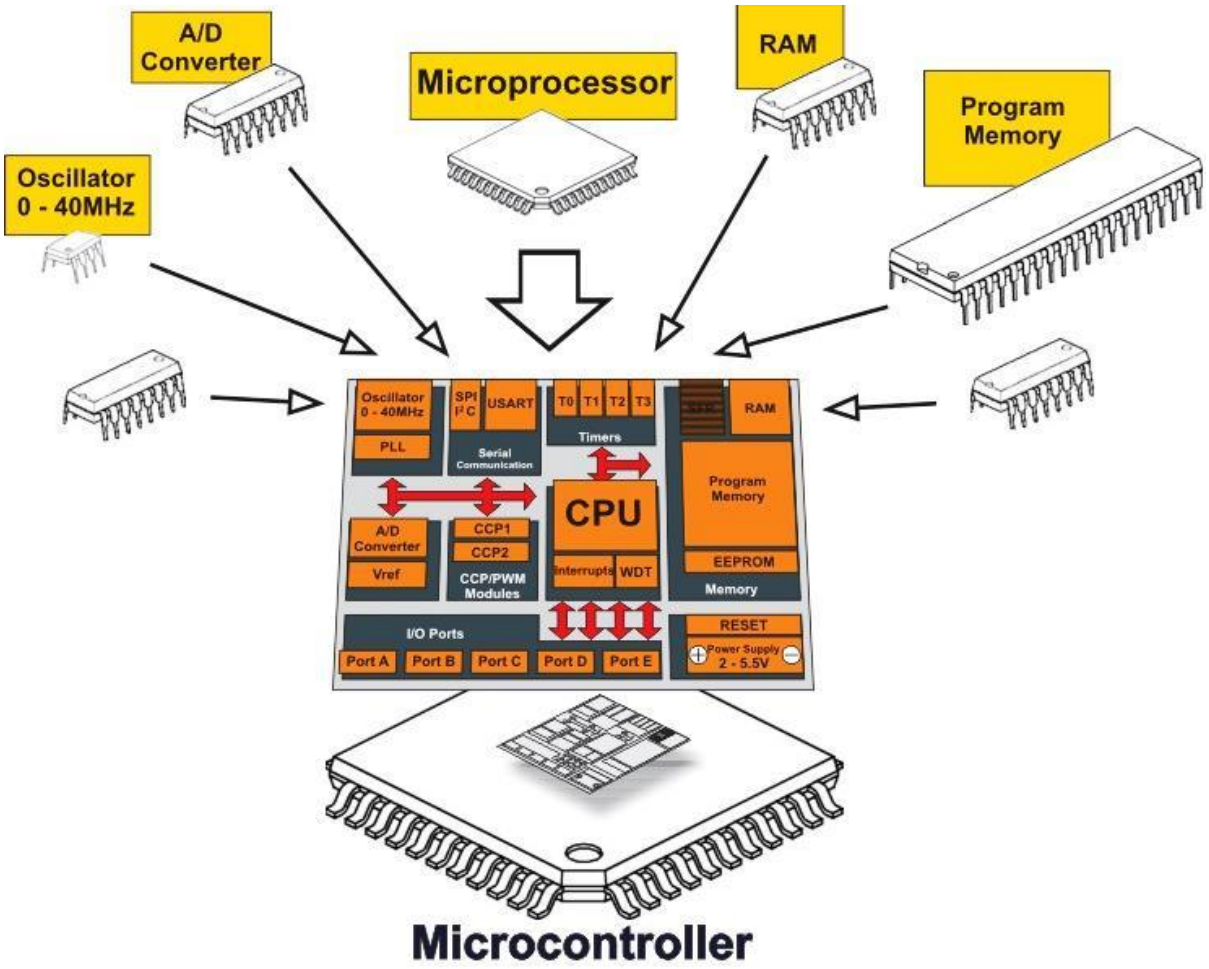
- **گذرگاه داده(Data Buss)**: جهت نقل و انتقال داده ها بین ماژولهای میکرو به کار می رود که میتواند ۸ خط (در میکرو های ۸ بیتی)، ۱۶ خط (در میکرو های ۱۶ بیتی) و ... باشد. هر چه تعداد خطوط Data بیشتر باشد سرعت انتقال داده ها بیشتر خواهد بود و توان پردازش میکروکنترلر بالاتر است.
- **گذرگاه آدرس(Address Buss)**: برای شناسایی هر وسیله (حافظه ها ، I/O ها) توسط CPU می باشد که باید آدرس منحصر به فردی به آنها تخصیص داد. هر چه تعداد خطوط آدرس بیشتر باشد

میکرو میتواند تعداد مکانهای بیشتری را آدرس دهی کند، در نتیجه میتواند حافظه بزرگتری داشته باشد.

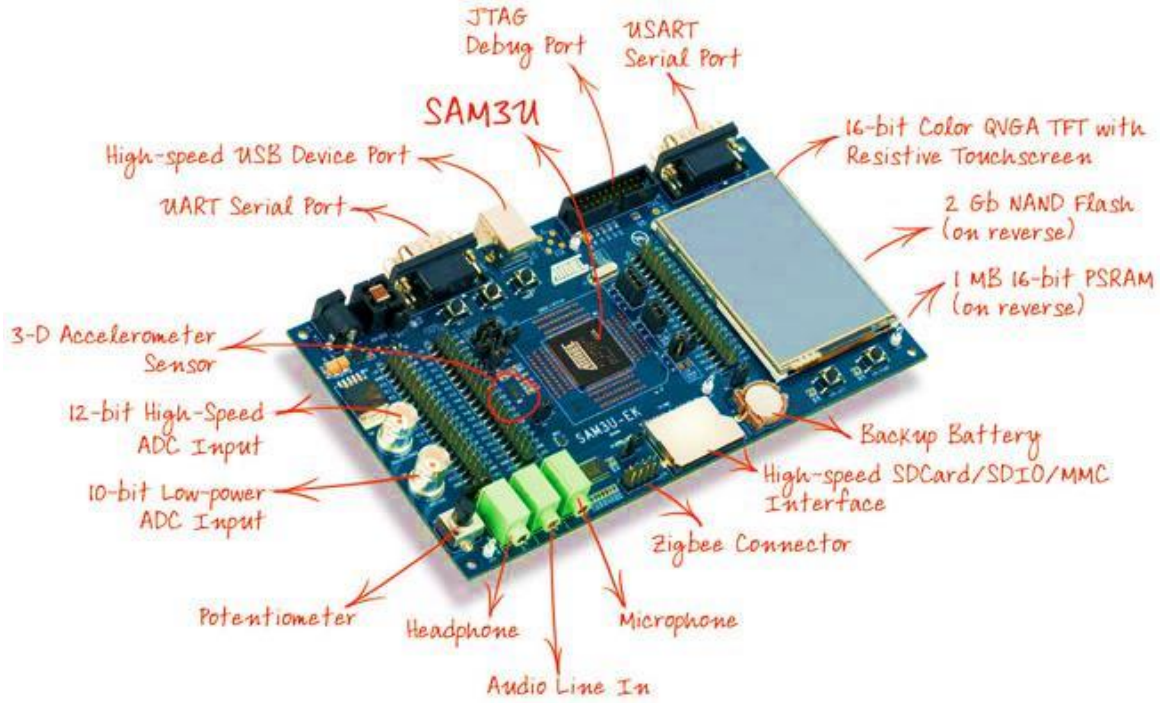
- **گذرگاه کنترل(Control Buss):** شامل خطوط کنترلی دستگاههای موجود مثل Read ، Writeو غیره است. هر چه تعداد خطوط کنترلی بیشتر باشد، میکرو امکانات کنترلی بیشتری در اختیار برنامه نویس قرار میدهد.

۲-۳- 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 کاربرد های 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 از شرکت Atmel 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.



۲-۴- 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 ۶۸۱۱ ...A1,A2,گسترش یافت. در حال حاضر میکروکنترلر های 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 شرکتهای Xilinx ، Altera و...

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:

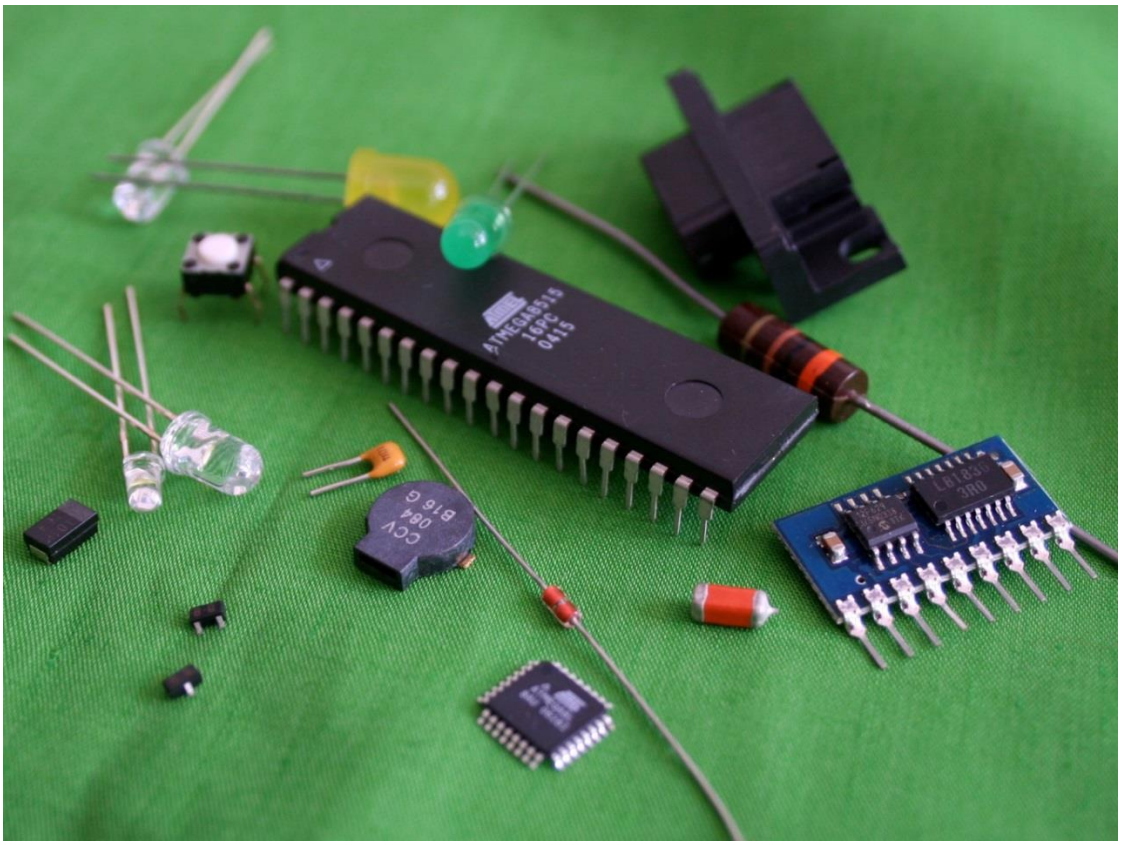
سری میکرو	تعداد زیر مجموعه ها	حداکثر فرکانس کاری	منابع یادگیری	قیمت	قدرت پردازش عمومی*	قدرت پردازش اختصاصی*	نویز پذیری	پشتیبانی از پروتکل ها
خانواده AVR	بیش از 120	300MHz	خیلی زیاد	نسبتا ارزان	متوسط	ضعیف	زیاد	متوسط
خانواده PIC	بیش از 60	40MHz	زیاد	متوسط	متوسط	متوسط	کم	خوب
خانواده ARM	بیش از 200	بیش از 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 ۳– معرفی 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.



۳-۱- معرفی و تاریخچه ساخت

AVRخانواده‌ای از میکروکنترلر ها است که شرکت ATMEL ، آن را روانه بازار الکترونیک کرده است. این میکروکنترلر های هشت بیتی به خاطر دارا بودن قابلیت برنامه‌نویسی توسط کامپایلر های 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 CMOSو استفاده از حافظه‌های کم مصرف و غیر فرار Flash و EEPROM است.

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 AVR یک 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.

۳-۲- انواع میکروکنترلرهای AVR

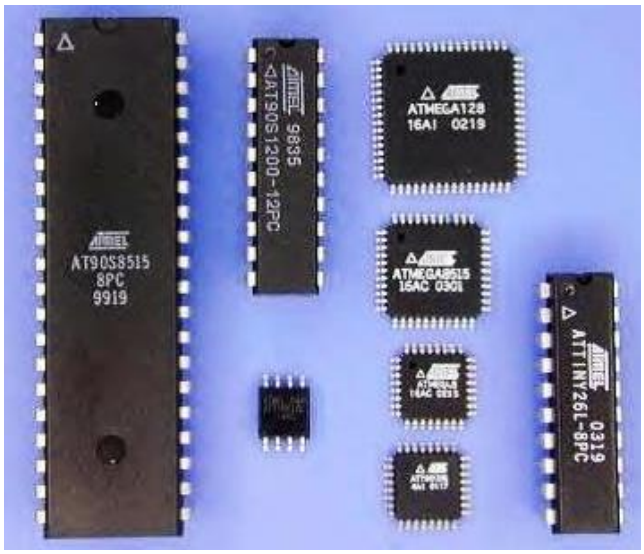
این میکروکنترلرها دارای ۴ سری می باشند که هر سری کاربردها و ویژگی های خاص خود را دارد.

۱. سری **ATtiny**: میکروکنترلرهای کوچک ، کم مصرف و پرقدرت برای کاربردهای خاص می باشند که دارای حافظه Flash بین ۰۰۵ تا ۱۶ کیلوبایت و بسته بندی بین ۶ تا ۳۲ پایه هستند.

۲. 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.

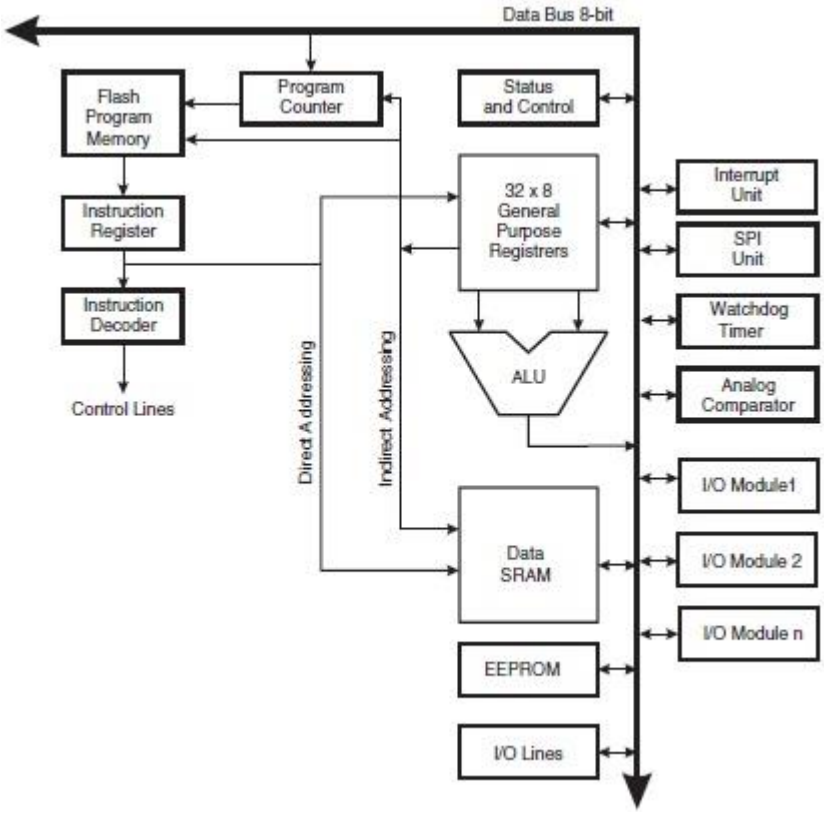
۳. سری **XMega**: جدیدترین ، پرسرعت ترین و قوی ترین نوع هستند که امکانات بیشتری نیز دارند که دارای حافظه Flash بین ۱۶ تا ۳۸۶ کیلوبایت و بسته بندی ۴۴، ۶۴ و ۱۰۰ پایه هستند.

۴. سری **AT90s**: نوع توسعه یافته میکروکنترلر ۸۰۵۱ هستند که امکانات کمتری داشته و کمتر کاربرد دارند چرا که تقریبا منسوخ شده اند.



۳-۳- 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 دیجیتال
- واحد مقایسه کننده آنالوگ
- واحد تایمر سگ نگهبان
- واحد ارتباطات سریال SPI ، TWIوUSART
- واحد برنامه ریزی و عیب یابیJTAG

۳-۳-۱- هسته مرکزی CPU (واحد پردازش مرکزی)

این واحد که بر مبنای معماری RISC ساخته شده است ، تمام فعالیت های میکروکنترلر را مدیریت کرده و تمام 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.

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

CISC: مخفف Complete Instruction Set Computer به 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 Computer به 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 میکرو کنترلر برای کارهای کوچکتر ساخته شده است و کار کوچک را با هزینه پایین تر و کیفیت بهتری انجام می دهد.

۳-۳-۲- واحد محاسبه و منطق (Arithmetic Logic Unit)

ALU یا واحد محاسبه و منطق در میکروکنترلر AVR به صورت مستقیم با تمام ۳۲ رجیستر همه منظوره ارتباط دارد. این واحد وظیفه انجام کلیه اعمال محاسباتی و منطقی را با استفاده از رجیسترهای همه منظوره و در یک دوره تناوب از کلاک بر عهده دارد. 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 واحد های 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.

۳-۳-۳- رجیسترهای CPU

رجیستر های عمومی General 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 ALUدر 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، ZL و 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 (CALL) و یا اجرای برنامه وقفه نیاز می باشد . همچنین به کمک دستورات اسمبلی PUSH و POPمیتوان به این قسمت نفوذ داشت. اشاره گر پشته یا SP دارای ۱۶ بیت است و از دو رجیستر ۸ بیتی SPLو 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 R0 با 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.

۳-۳-۴- 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 CPUفرآیند 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.

۳-۳-۵- خط لوله Pipelining

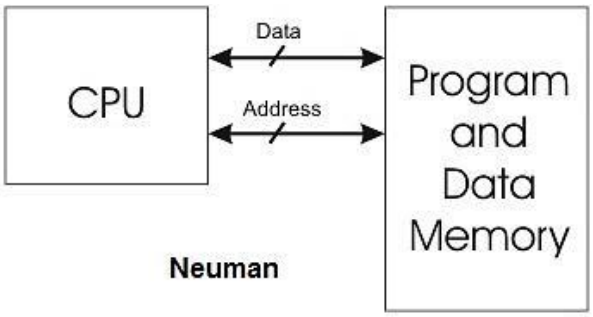
مراحل اجرای یک دستور که توسط کاربر نوشته می شود به این صورت است که بعد از روشن شدن میکرو اولین آدرس حافظه فلش که حاوی اولین دستورالعمل است واکشی می شود. بعد از واکشی ترجمه یا رمز گشایی می شود ، معماری این میکرو به صورتی طراحی شده است که قابلیت واکشی (Fetch) ، رمز گشایی (Decode)و اجرای دستور (Execute) 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 Executeرا دارد.**

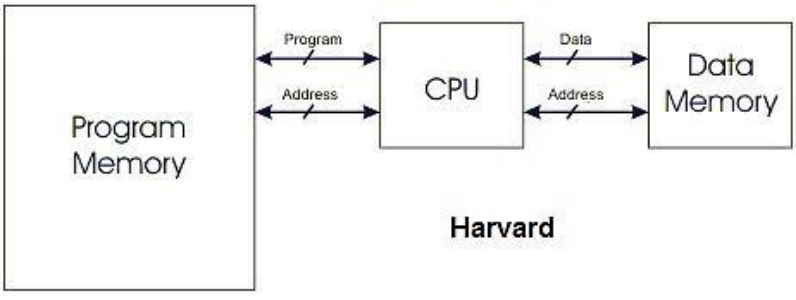
Instr. No.	Pipeline Stage						
1	IF	ID	EX				
2		IF	ID	EX			
3			IF	ID	EX		
4				IF	ID	EX	
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

۳-۴- 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 می کنند. معماری هاروارد در مقایسه با معماری سنتی فون نیومن سریع تر است. شکل زیر تفاوت دو معماری را نشان می دهد.



Neuman



Harvard

۳-۴-۱- حافظه داده SRAM

این حافظه جهت ذخیره سازی موقت اطلاعات در اختیار کاربر قرار می گیرد و با قطع تغذیه آی سی اطلاعات آن از بین می رود. 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.

۳-۴-۲- 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.

۳-۴-۳- 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.

۳-۵- معرفی دیگر واحدهای میکروکنترلرهای AVR

۳-۵-۱- 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 AVR با توجه به نوع و سری میکروکنترلر تعداد آنها بین ۱ تا ۱۰ پورت متفاوت می باشد. پورت ها در AVR به صورت PORTA ، PORTB، PORTC و... نامگذاری می شود. هر پورت ۸ بیت دارد که به صورت PORTX.0 تا PORTX.7 نامگذاری می شود.

۳–۵–۲– 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:

۱. **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.

۲. **کریستال خارجی:** در این حالت از اتصال یک کریستال با فرکانس مشخص و دو عدد خازن بین پایه های X1 و X2 استفاده می شود . جنس بلور این کریستال ها معمولا کریستال کوارتز است . در این حالت از اسیلاتور داخل میکرو استفاده می شود.

۳. **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.

۳–۵–۳– واحد تایمرها و شمارنده ها(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 ، RTC و 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.

۳–۵–۴– 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.

۳-۵-۵- 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.

۳-۵-۶- واحد ارتباطی JTAG

JTAG کوتاه 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 JTAGحافظه Flash ، EEPROM، فیوز بیت ها و قفل بیت ها را برنامه ریزی (پروگرام Program) کرد

.

۳-۵-۷- 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.

۳–۵–۸– 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.

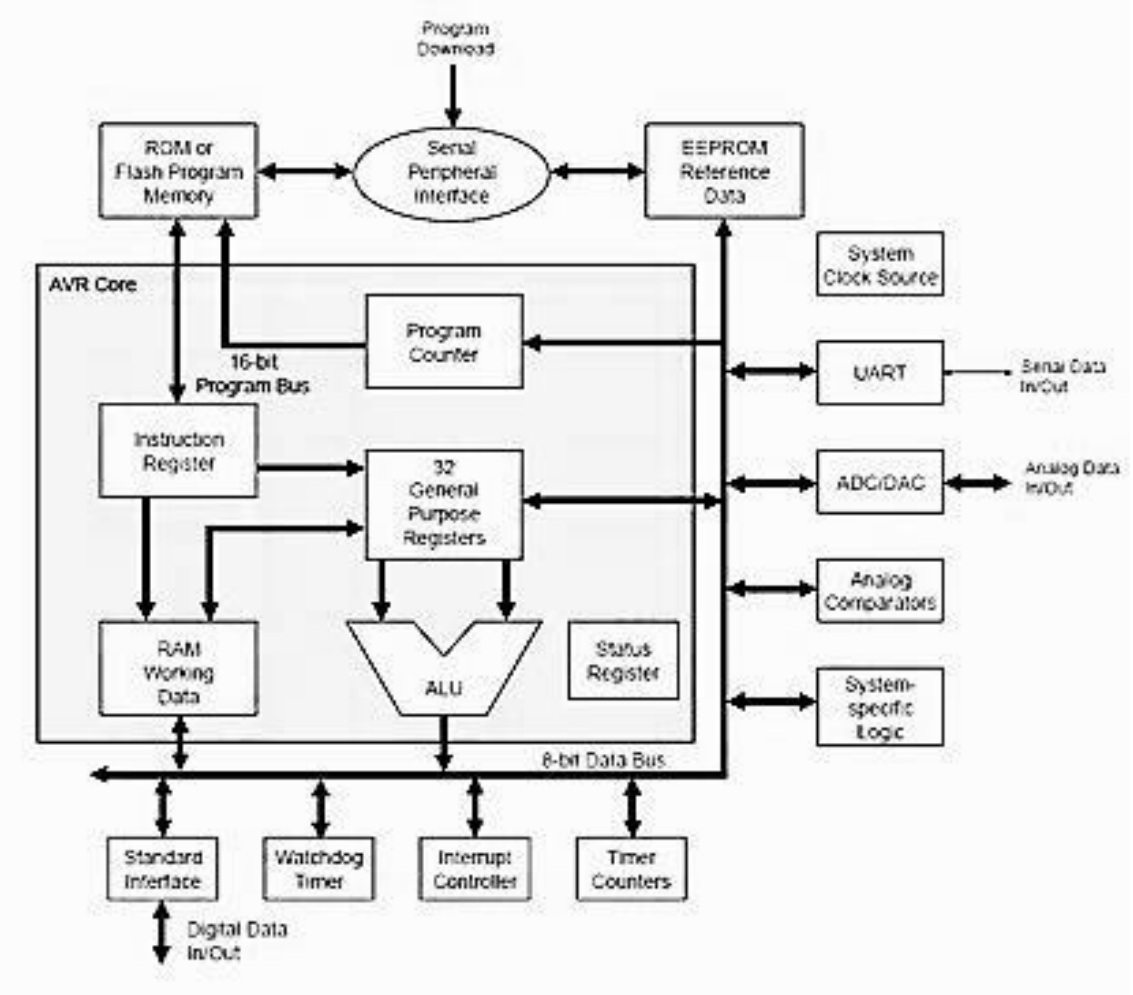
سوالی که ممکن است بوجود آید این است که با وجود مبدل آنالوگ به دیجیتال دیگر چه نیازی به این بخش می باشد؟ در جواب باید گفت سرعت عملکرد این بخش در مقایسه با مبدل انالوگ به دیجیتال بسیار بیشتر بوده و همین سرعت باعث احساس نیاز به چنین بخشی را فراهم کرده است.

۳–۵–۹– 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:

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

در پایان شکل بهتری از کلیه واحد های یک میکروکنترلر AVR 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.



۳–۶– انواع زبان های برنامه نویسی و کامپایلر های 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 AVR و 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.

۳–۷– برنامه ریزی (پروگرام) کردن میکروکنترلرهای AVR

بعد از نوشتن برنامه درون میکروکنترلر نوبت به انتقال آن به میکرو می شود که به آن اصطلاحا پروگرام کردن یا برنامه ریزی می گویند. وسیله ای که به کامپیوتر متصل می شود و توسط آن میکروکنترلر پروگرام می شود را اصطلاحا پروگرامر (Programmer) گویند.

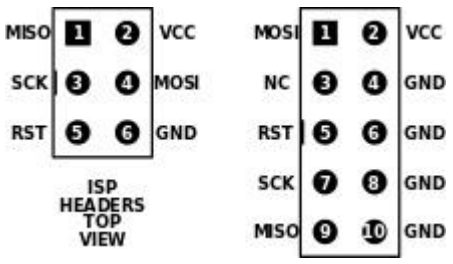
برنامه ریزی میکروکنترلرهای AVR به سه روش صورت می گیرد : برنامه ریزی سریال ، موازی وJTAG

برنامه ریزی به روش موازی سریعتر انجام می گیرد اما نحوه انجام کار آن پیچیده تر است. ضمنا تمامی میکروکنترلرهای AVR از 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 سپس میکروکنترلر روی پروگرامر قرار میگیرد و اینگونه پروگرام می شود. ساده ترین پروگرامر stk300 نام دارد که به پورت پرینتر کامپیوترهای PC وصل می شود. بهترین نوع پروگرامرهای AVR آن هایی است که از رابط USB استفاده می کند. معروفترین پروگرامرهای موجود در بازار که به پورت USBمتصل می شوند میتوان پروگرامرهای stk500 ، USBasp و isp mk2 را نام برد.

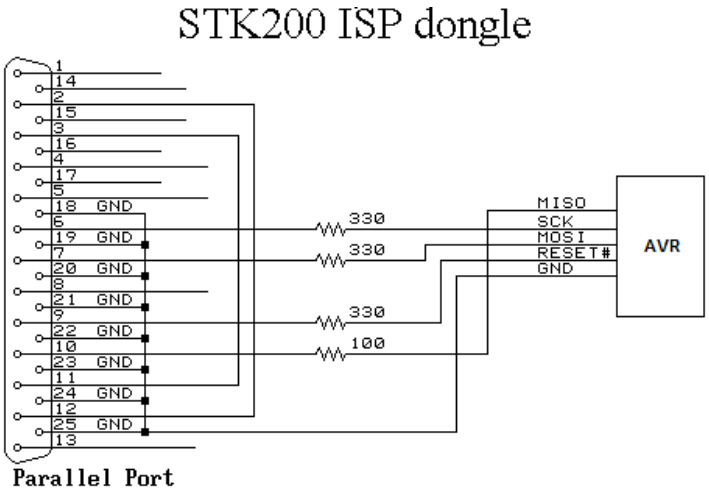
۳–۷–۱– ISP چیست ؟

یکی از راههای پروگرام کردن میکروکنترلر های AVR روش ISP یا In System Programming می باشد. در این روش میکروکنترلر درون سیستم (مدار) پروگرام می شود. بدین معنی که به جای خارج کردن میکروکنترلر از مدار و قرار دادن آن روی پروگرامر ، میکروکنترلر در درون مدار و توسط پورت استاندارد ISP ، 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.



۳-۷-۲- تهیه پروگرامر مناسب

ساده ترین نوع پروگرامر میکروکنترلرهای AVR ، Stk200 می باشد که تنها به یک پورت LPT یا همان پورت پرینتر DB25 و چند مقاومت برای ساخت نیاز دارد. بنابراین خودتان به راحتی می توانید قطعات آن را تهیه و این پروگرامر را درست کنید. در شکل زیر این پروگرامر و نحوه اتصال آن به میکروکنترلر نشان داده شده است. در یک طرف آن پورت پرینتر قرار دارد که به کامپیوتر متصل می شود و در طرف دیگر میکروکنترلر قرار دارد که باید پروگرامر مطابق شکل به پایه های مشخص شده از میکرو متصل گردد. در این شکل چون میکروکنترلر از مدار خارج نمی شود ، روش پروگرام کردن ISP می باشد.



تذکر مهم : به دلیل اینکه دسترسی به پورت پرینتر سخت است و در هنگام کار با آن مشکلات ایجاد می گردد و ضمن اینکه لپتاپ ها و بسیاری از وسایل از پورت پرینتر پشتیبانی نمی کنند ، توصیه می شود یکی از پروگرامرهای USB را از بازار تهیه نمایید.

فصل ۴– بررسی و راه اندازی Atmega32

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.

۴–۱– خصوصیات ، ویژگی ها و عملکرد ATMega32

میکروکنترلر ATMega32 یک میکروکنترلر ۸ بیتی کم مصرف از تکنولوژی CMOS و از خانواده MEGA AVR است که بر اساس معماری RISC بهبود یافته می باشد. با اجرای دستورات قدرتمند در یک سیکل کلاک این میکرو سرعتی معادل 1MIPS (مخفف Millions Instruction Per Second) در هر MHz را فراهم می کند. یعنی زمانی که فرکانس کاری میکرو ۱ مگاهرتز است ، میکرو می تواند یک میلیون دستور العمل در هر ثانیه انجام دهد. این قابلیت 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 خانواده AVR
- 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 MIPS در 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 حافظه EEPROM

    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 ارتباطJTAG

- 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(

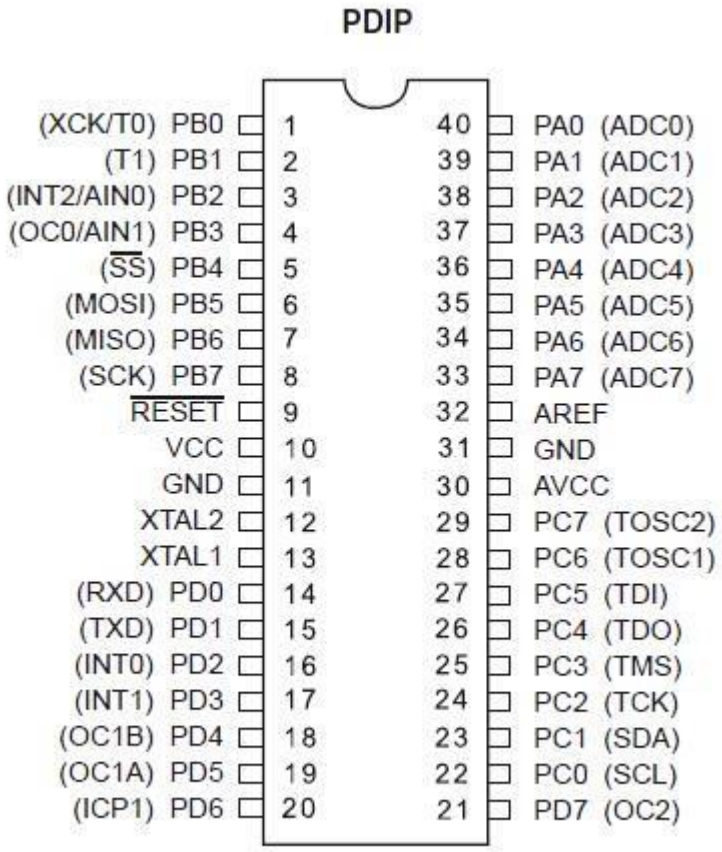
    دارای ۲ کانال تفاضلی با گین قابل برنامه ریزی ۱ ، ۱۰ و ۲۰۰ برابر کننده
  - قابلیت ارتباط با پروتکل سریال دوسیمهI2C
  - دارای پورت USART سریال قابل برنامه ریزی
  - قابلیت ارتباط سریال SPI به صورت Master یاSlave
  - دارای تایمر قابل برنامه ریزی Watchdog با اسیلاتور داخلی جداگانه
  - دارای مقایسه کننده آنالوگ 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 MHz برای ATMega32L
- ۰ تا ۱۶ MHz برای ATMega32

۴-۲- تشریح عملکرد پایه ها در ATMEGA32

همانطور که مشاهده کردید این میکرو در سه بسته بندی مختلف تولید و عرضه شده است که بسته بندی TQFP و MLF به صورت SMD (تکنولوژی روی سطح برد) برای استفاده در pcb (فیبر مدار چاپی) و بسته بندی PDIP به 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/O می 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 می ماند.

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 کانال PWM (واحد تایمر/کانتر) است.

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.

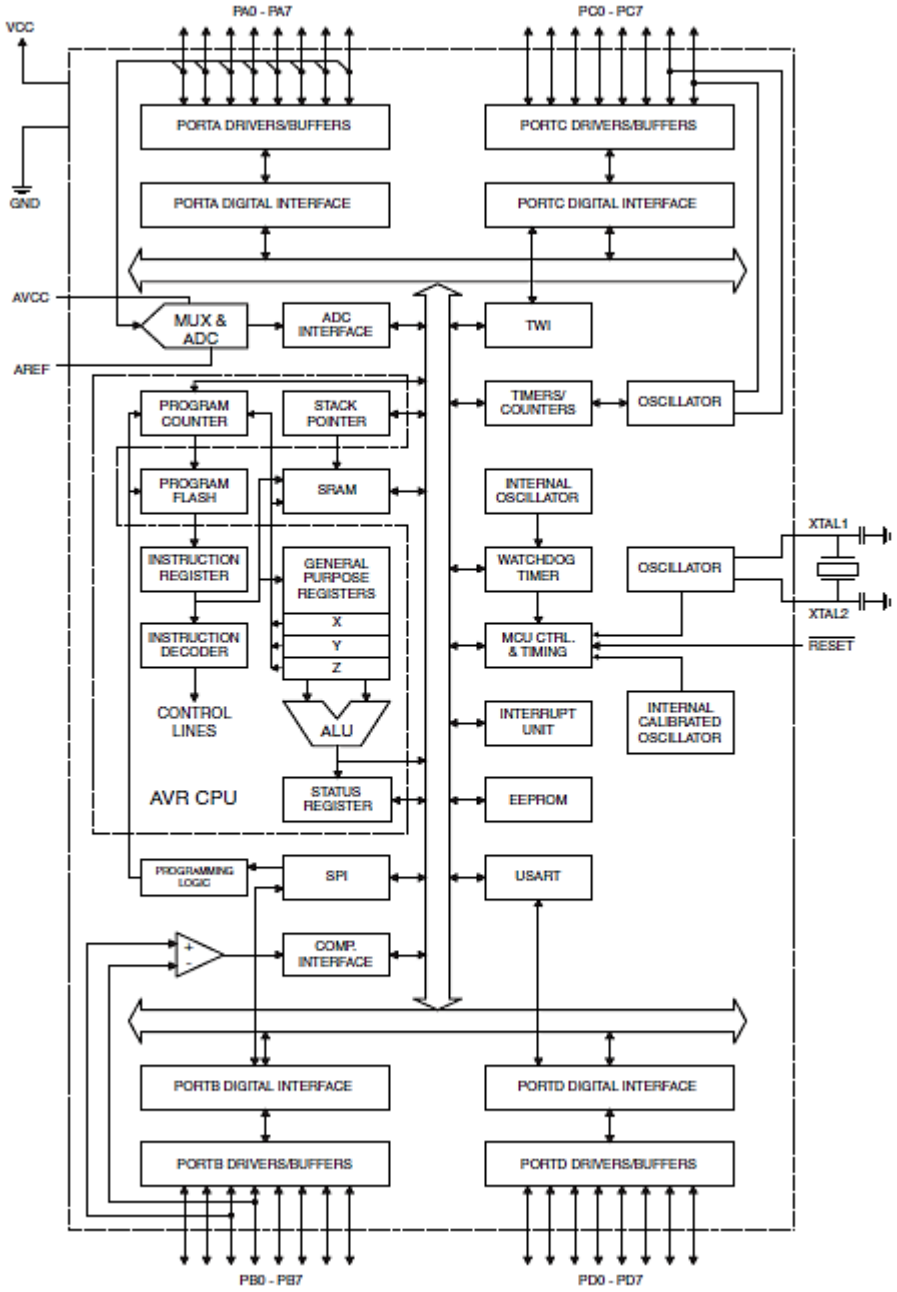
پایه ۳۲ (AREF) : پایه مربوط به ولتاژ مرجع قسمت ADC می باشد.

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.

۴-۳- 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:

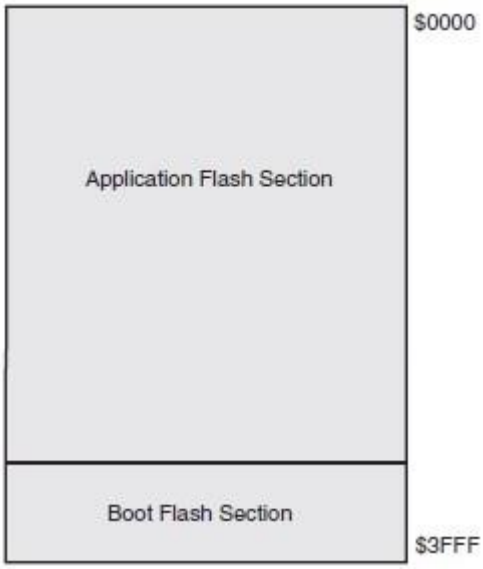


- **واحد پردازش مرکزی (CPU)** : دارای ۱۳۱ دستورالعمل می باشد که در صفحه ۳۲۷ دیتاشیت همه ۱۳۱ دستور العمل به زبان اسمبلی آورده شده است. معماری این واحد دقیقا مشابه گفته شده در فصل قبل می باشد. این واحد در شکل فوق با نقطه چین (AVR CPU) مشخص شده است. ضمنا پهنای رجیستر شمارنده برنامه (Program Counter) در 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 AVR تعداد 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 16K خانه حافظه برنامه را آدرس دهی نماید. (2^{14} =16384>16K)

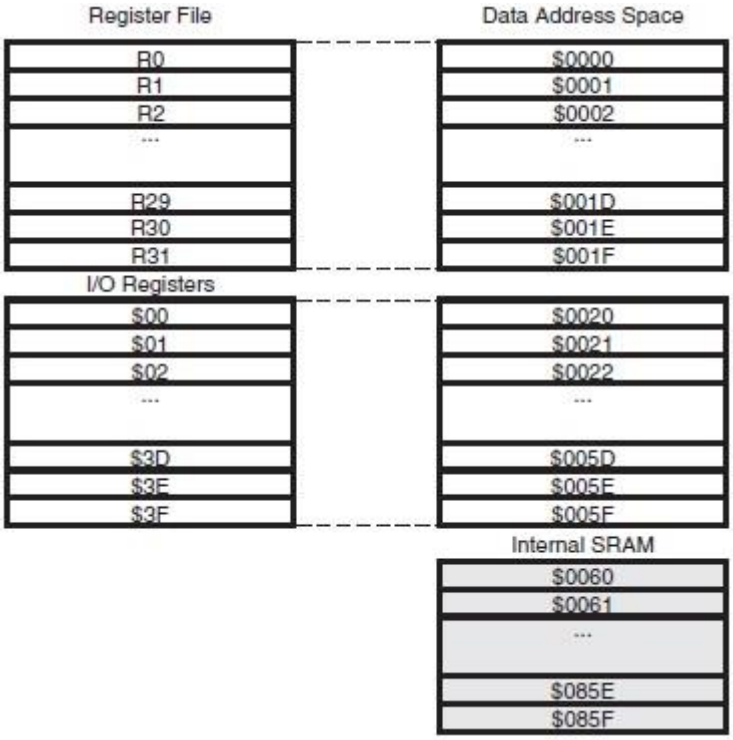
نکته : از حافظه برنامه Flash در صورتی که حجم برنامه کم بوده باشد و فضای خالی موجود باشد ، میتوان برای ذخیره دائمی اطلاعات (یعنی به عنوان حافظه داده) نیز استفاده نمود.



- **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 میتوان محتوای حافظه SRAM را حفظ کرد.



- **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 Flashدارد . 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
- واحد Wathdog Timer
- واحد کنترل وقفه
- واحد ارتباطیJTAG

- واحد مبدلADC
- واحد مقایسه کننده
- واحد ارتباطی spi
- واحد ارتباطی USART
- واحد ارتباطی TWI

تذکر : عملکرد ، معماری و رجیسترهای بقیه واحدهای جانبی فوق الذکر در این فصل و فصل های آتی کاملا تشریح خواهد شد.

۴-۴- ساختار برنامه میکروکنترلر به زبان C

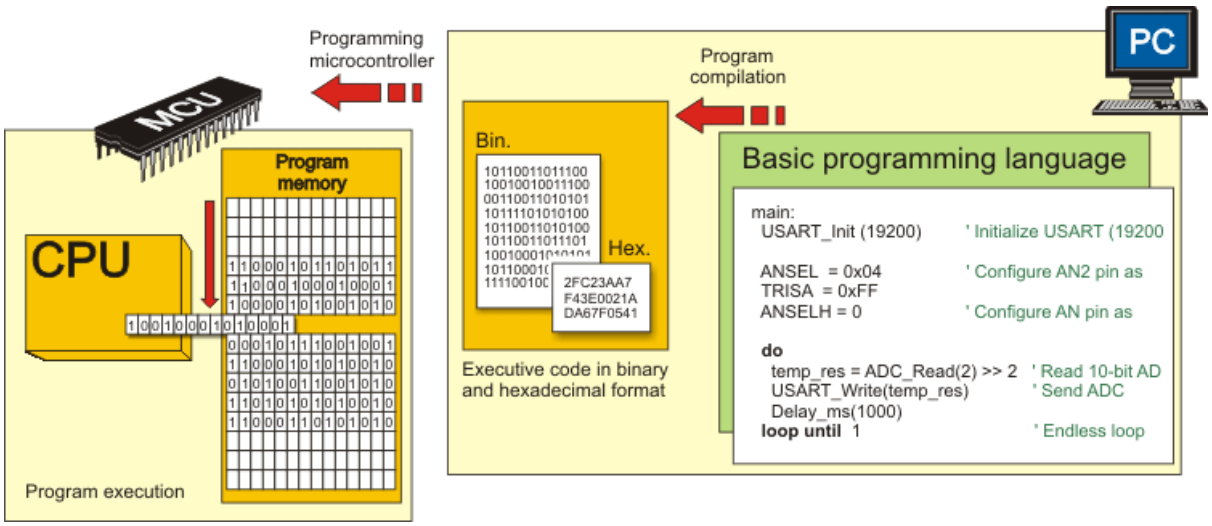
برنامه میکروکنترلر باید توسط برنامه نویس روی یک کامپایلر نوشته شود و سپس توسط پروگرامر روی میکرو پروگرام شود . برنامه ای که نوشته می شود باید طوری نوشته شود که وقتی روی آی سی پروگرام شد دائما اجرا شود و هیچگاه متوقف نشود . راه حل این مسئله قرار دادن کدهای برنامه درون یک حلقه نامتناهی است .این عمل باعث 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
  }
 }
```

۴-۴-۱- نحوه عملیاتی شدن یک برنامه توسط میکرو

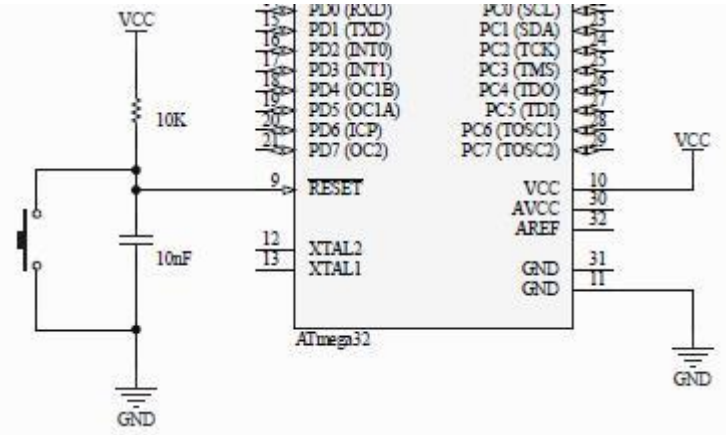
در شکل زیر نحوه نوشتن یک برنامه نمونه در کامپیوتر و پروسه تولید کد و ریخته شدن آن روی میکروکنترلر به تصویر کشیده شده است. بعد از نوشتن برنامه ، فایلی متشکل از ۰ و ۱ در مبنای ۲ تشکیل می شود ، سپس همین فایل به مبنای ۱۶ می رود و درون فایل Hex ریخته می شود. فایل Hex توسط

پروگرامر به درون حافظه Flash میکروکنترلر می رود و سپس CPU خط به خط برنامه را خوانده و اجرا می کند.



۴–۵– 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 بهتر است پایه RESET را توسط یک مقاومت مناسب (۱۰ کیلواهم) به مثبت تغذیه وصل نماییم (Pull Up) . 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 10k و 10n انتخاب 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 به تغذیه متصل می شود .

۴-۶- معرفی رجیسترهای واحدهای میکروکنترلر Atmega32

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 CodeVision و برنامه نویسی C دوباره بقیه واحد ها را ادامه خواهیم داد.

۴-۶-۱- رجیسترهای واحد I/O در AVR

تمامی میکروکنترلرهای AVR از جمله Atmega32 دارای 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 متناظر با آن به عنوان خروجی و هر بیت که ۰ باشد پایه متناظر 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 ۰ کردن منطق پایه خروجی استفاده می شود. هر بیت از این رجیستر که ۱ باشد پایه متناظر با آن ، منطق ۱ را خارج می کند و هر بیت متناظر 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 ورودی 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 می شوند.

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

نکته ۴ : از 0b برای مقدار دهی به رجیسترها در مبنای ۲ (باینری) و از 0x برای 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 default روی 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 در فصل بعدی 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 Atmega32باید نوشته شود)

در خط دوم کتابخانه مربوط به تابع تاخیر زمانی (delay.h) را اضافه می کنیم. تنها 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 ledدر 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 led به مدت ۲۵۰ میلی ثانیه خاموش میماند.

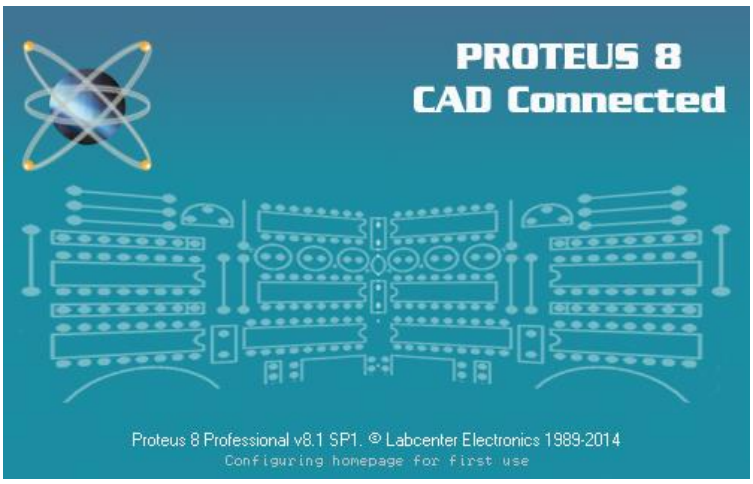
چون برنامه در حلقه نامتناهی نوشته شده است ، با اجرا شدن برنامه LED موجود روی PA.0 دائما روشن و خاموش می شود و چون زمان تاخیر ۲۵۰ میلی ثانیه است این کار ۴ بار در ثانیه انجام می شود.

فصل ۵– اصول شبیه سازی و پیاده سازی

مقدمه :

در فصل گذشته با معماری و ساختار میکروکنترلر Atmega32 آشنا شدیم. گفتیم مهمترین بخش میکروکنترلر Atmega32 که 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 Atmega32به تعداد ۶۴ رجیستر برای کنترل مجموعه سیستم میکروکنترلر وجود دارد که در قسمت SRAMحافظه 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 Proteus و 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.

۵–۱– 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 قابلیت استفاده برای کشیدن بردهای PCB را نیز فراهم کرده است و برای این کار هم محیطی مجزا از شبیه سازی در نظر گرفته 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 AVRرا 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...

۵-۲- 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 برای دانلود آن ها به لینک های زیر مراجعه نمایید. (این ورژن ها روی ویندوز ۸ و ۱۰

نیز نصب می شوند)

لینک دانلود نرم افزار پروتئوس لینک دانلود نرم افزار کدویژن

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.

۵-۳- مراحل کلی انجام یک پروژه میکروکنترلری

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

۱. **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.
۲. **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.
۳. **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.
۴. **پیاده سازی :** پروگرام کردن میکروکنترلر و بستن مدار مورد نظر روی بردبرد در این مرحله صورت می گیرد.
۵. **تست و عیب یابی :** با وصل منبع تغذیه به مدار ، تست و عیب یابی مدار در این مرحله صورت می گیرد
۶. **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.

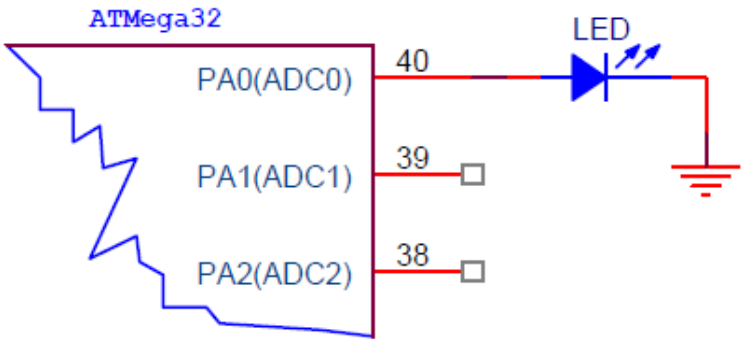
تذکر : مراحل فوق بسیار مهم هستند و در پروژه های بزرگ میبایست به ترتیب و با دقت انجام گیرد.

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 CodeVisionروی میکروکنترلر Atmega32 پیاده سازی نمایید.

حل:

مرحله اول : طراحی سخت افزار خواسته شده



مرحله دوم : طراحی نرم افزار خواسته شده

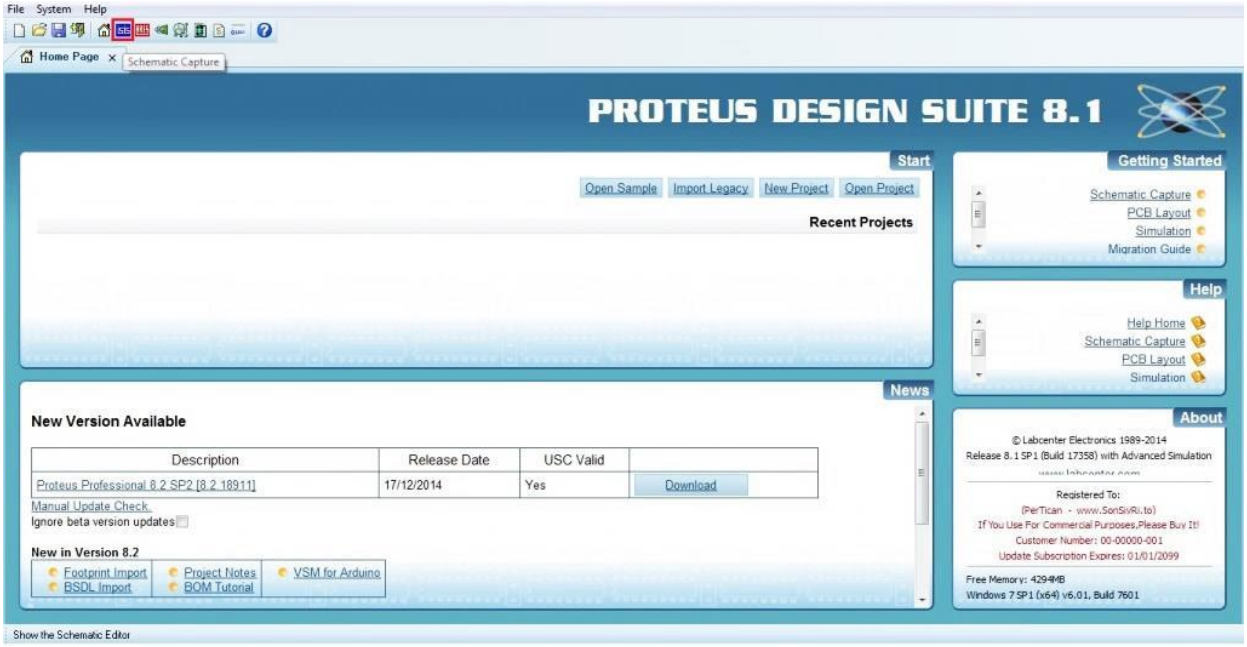
```
#include <mega32.h>
#include <delay.h>
void main(void)
{
DDRA.0=1;
while(1){
PORTA.0=1;
delay_ms(250);
PORTA.0=0;
delay_ms(250);
}
}
```

مرحله سوم : شبیه سازی توسط نرم افزارهای Proteus وCodeVision

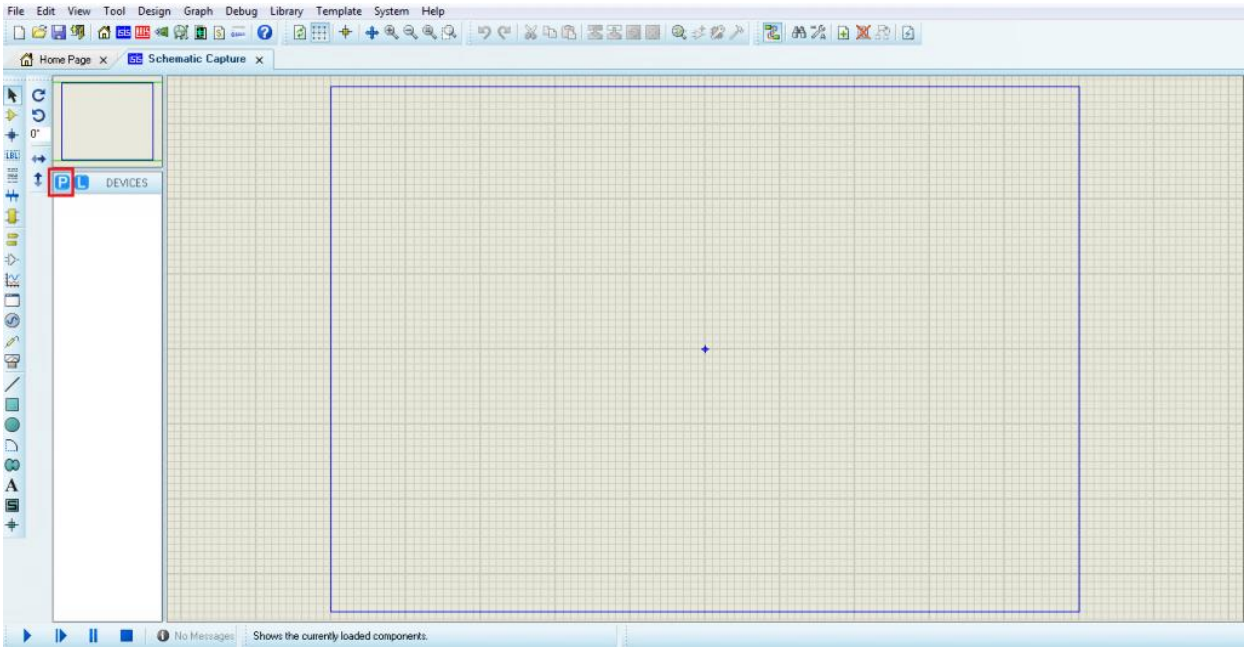
در این مرحله همیشه ابتدا به نرم افزار پروتئوس مراجعه کرده و سخت افزار طراحی شده را رسم می نماییم. سپس به نرم افزار کدویژن مراجعه کرده و نرم افزار مربوطه را کامپایل و می سازیم (Build) . سپس برنامه ساخته شده توسط کدویژن را در نرم افزار پروتئوس اضافه (add) می کنیم و مدار را شبیه سازی (Run) می کنیم . در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

۵-۴- شروع به کار با نرم افزار پروتئوس

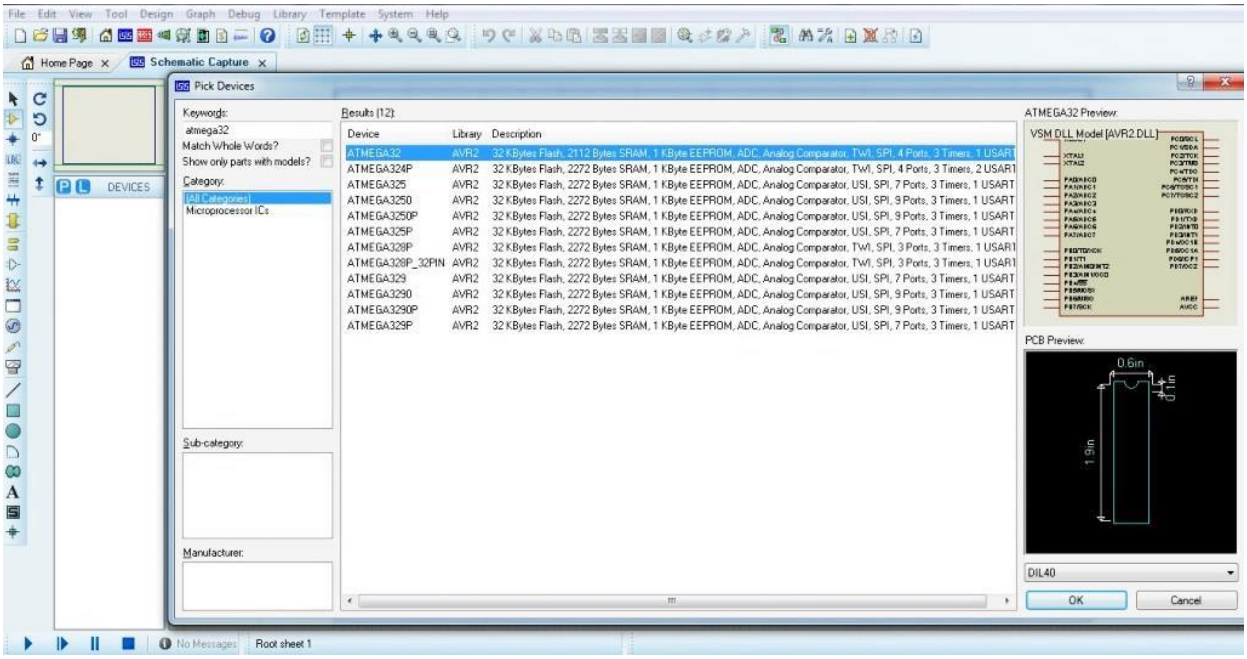
پس از نصب نرم افزار بر روی آیکون آن در صفحه دسکتاپ کلیک کرده تا نرم افزار باز شود . زمانی که نرم افزار باز شد با پنجره شکل زیر مواجه می شوید . در این مرحله بر روی آیکون آبی رنگ ISIS موجود در نوار ابزار بالایی که در شکل زیر نیز مشخص شده است ، کلیک کنید تا Schematic capture برای رسم مدار باز شود.

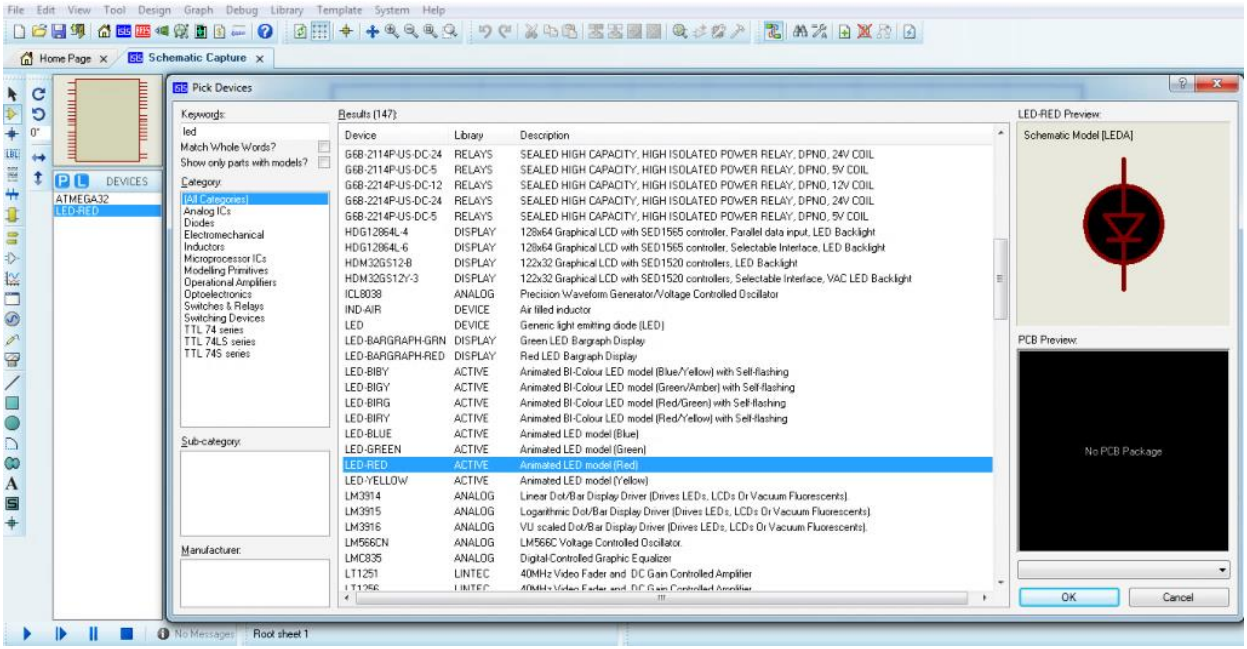


برای انتخاب المان های مداری باید روی آیکون P کلیک کنید تا صفحه جدیدی بنام Pick Devices باز شود.

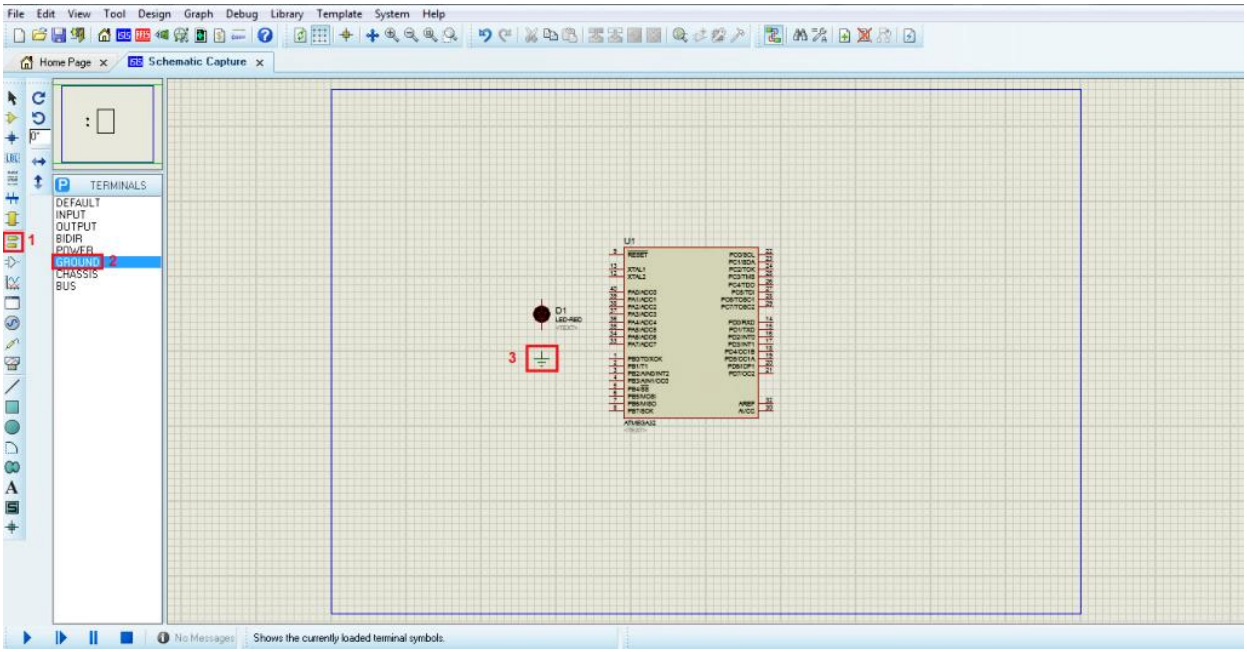


در این صفحه هر المانی نیاز داشته باشید را تایپ کرده و سپس روی نام قطعه دابل کلیک می کنید تا قابل استفاده گردد . برای این مثال یک Atmega32 و یک led نیاز داریم. نام آنها را تایپ کرده و از قسمت سمت چپ روی نام آنها دابل کلیک می کنیم. در نهایت پنجره Pick Devices را با کلیک بر روی Ok می بندیم. مراحل اجرای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

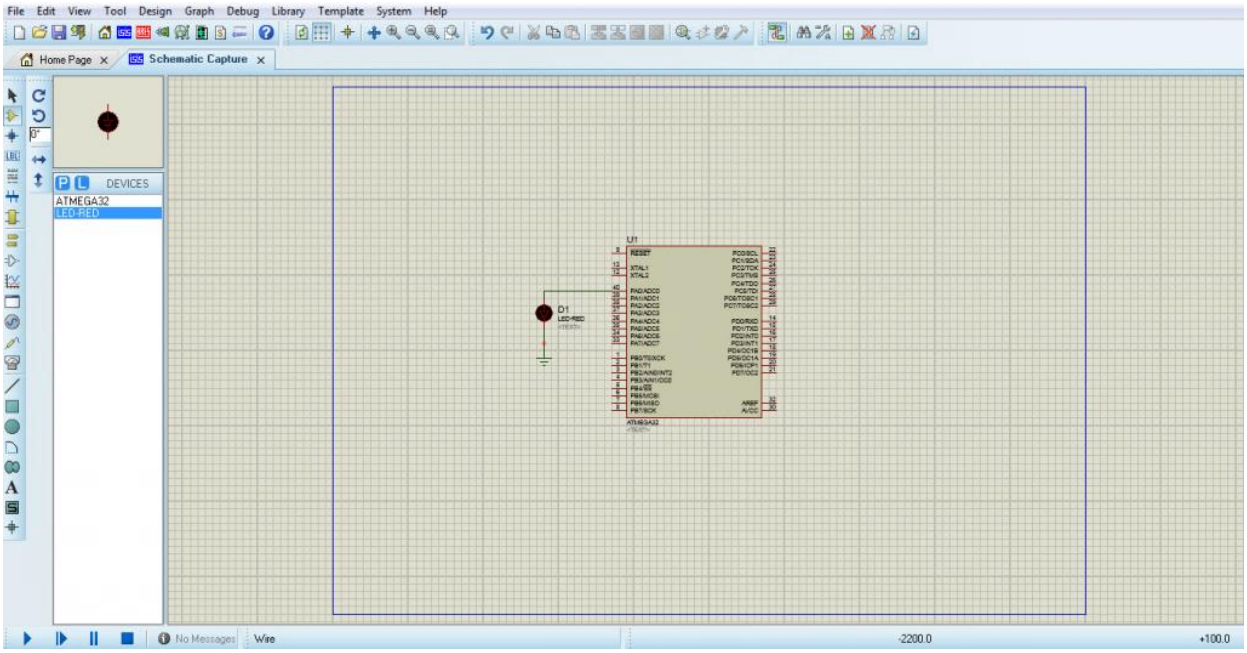




با بازگشت به صفحه اصلی مشاهده می کنید که زیر آیکون P قطعاتی را که انتخاب کرده بودیم آورده شده است. با کلیک بر روی آنها میکرو و LED را درون کادر آبی رنگ صفحه اصلی در محل مناسب خود قرار داده و مدار را تکمیل می کنیم . به یک زمین (Ground) نیز احتیاج داریم که آن را با کلیک بر روی آیکون Terminals Modeموجود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

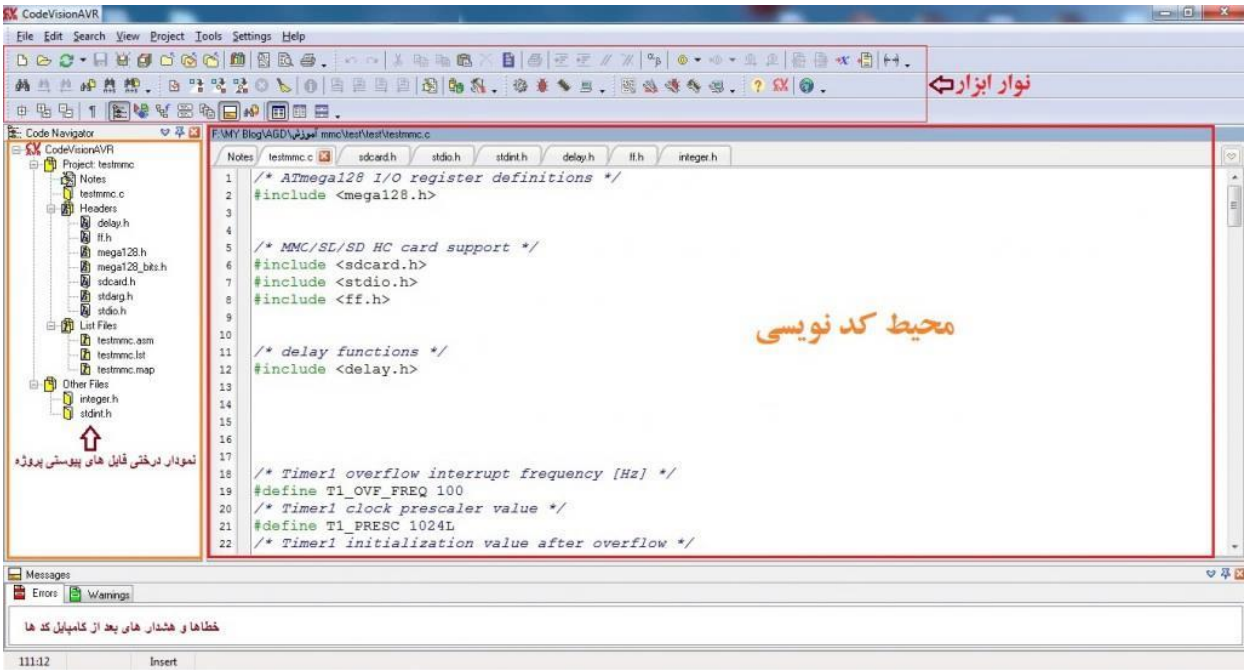


در نهایت نوبت به سیم کشی مدار می رسد . زمانی که نشانگر ماوس را در محل سیم کشی روی پایه(پین) های LED یا میکروکنترلر می برید ، مداد سبز رنگی ظاهر می شود ، در همین حال کلیک کنید و سیم کشی مدار را به صورت شکل زیر تکمیل نمایید.



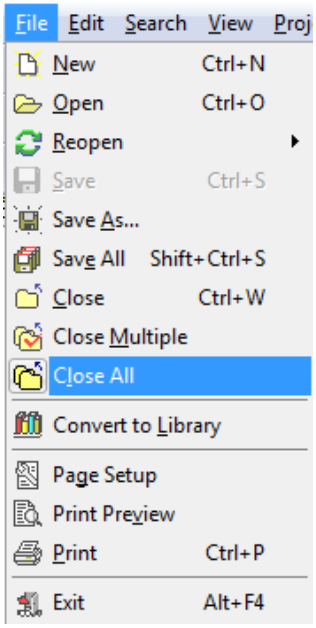
بعد از تکمیل مدار آن را از منوی File و گزینه Save با نام مناسب در یک پوشه جدید ذخیره نمایید. برای اینکه برنامه را بتوان بر روی آی سی ریخته و سپس اجرا کرد میبایست ابتدا باید فایلی که از نرم افزار CodeVisionتولید می شود را داشته باشیم . بنابراین در این مرحله به سراغ نرم افزار کد ویژن رفته و پس از نوشتن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

۵–۵– شروع به کار با نرم افزارCodeVision AVR

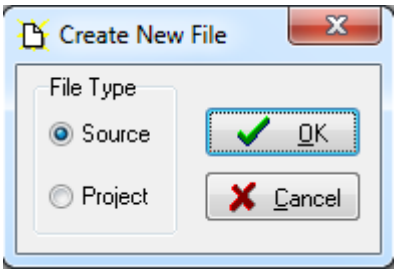


پس از نصب و اجرای این نرم افزار برای شروع پروژه ی جدید باید مراحل زیر با دقت و به ترتیب طی شود:

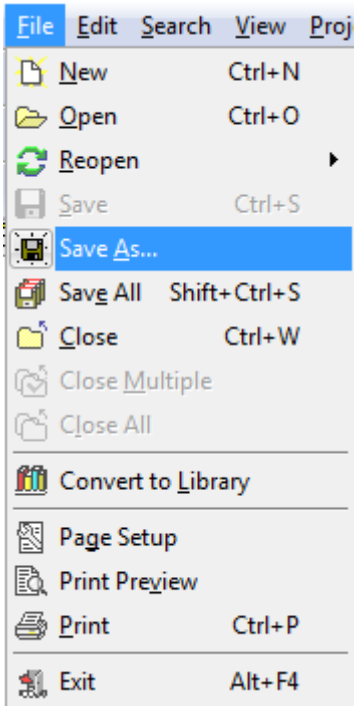
۱. ابتدا ممکن است آخرین پروژه ای که کار کرده اید باز باشد آن را از مسیر File و سپس Close All ببندید.



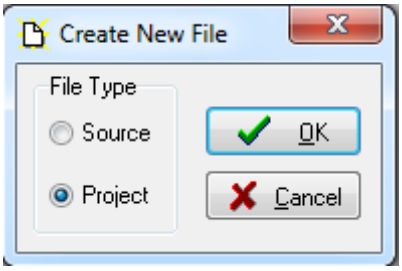
۲. از منوی File گزینه New را انتخاب کرده و سپس در پنجره باز شده ، Sourceرا انتخاب و Ok کنید.



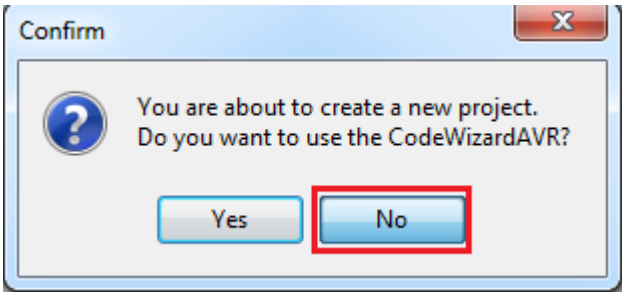
۳. فایل untitled.cباز می شود . در این مرحله می بایست از منوی File و سپس Save as آن را با نام مناسب در همان پوشه ای که فایل پروتئوس قرار دارد ، ذخیره کنید.



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

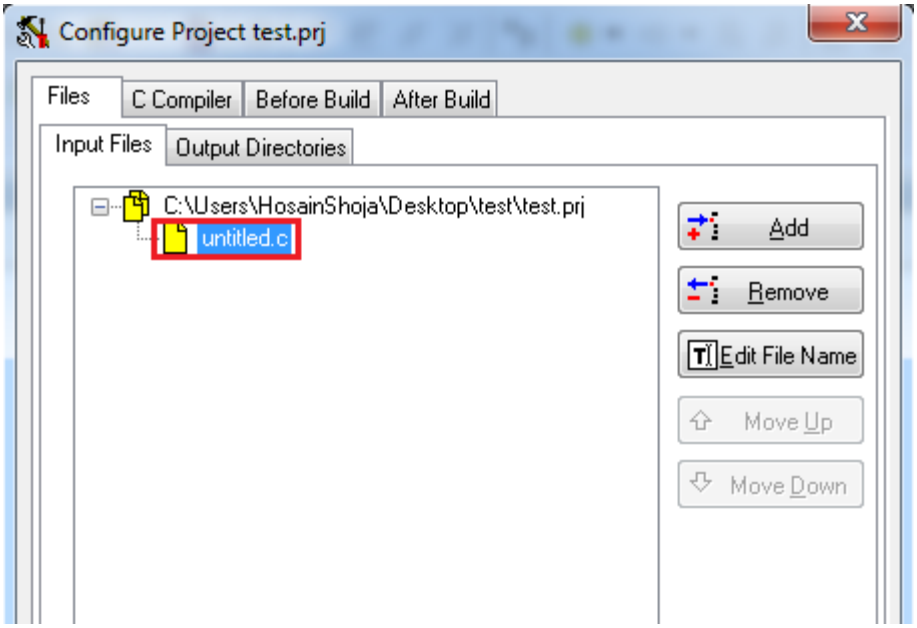


۵. پنجره ای مبنی بر اینکه آیا می خواهید از CodeWizard استفاده کنید یا خیر ؟ باز می شود آن را **No**کنید.

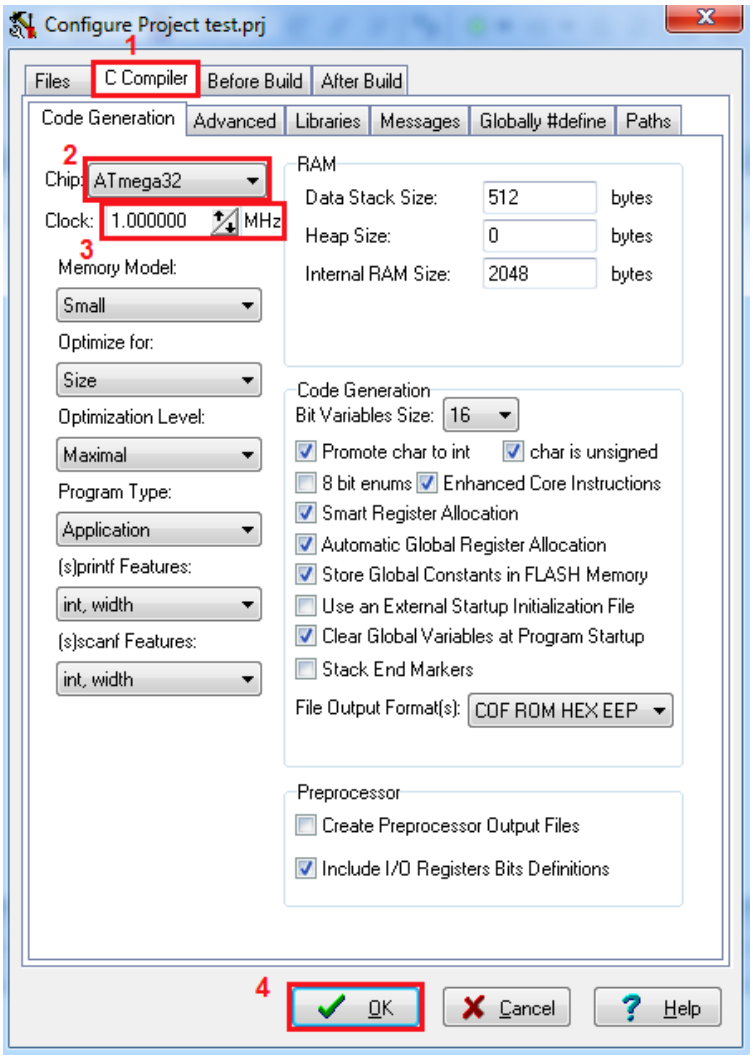


۶. پروژه را در همان مسیر قبلی با نام مناسب ذخیره کنید.

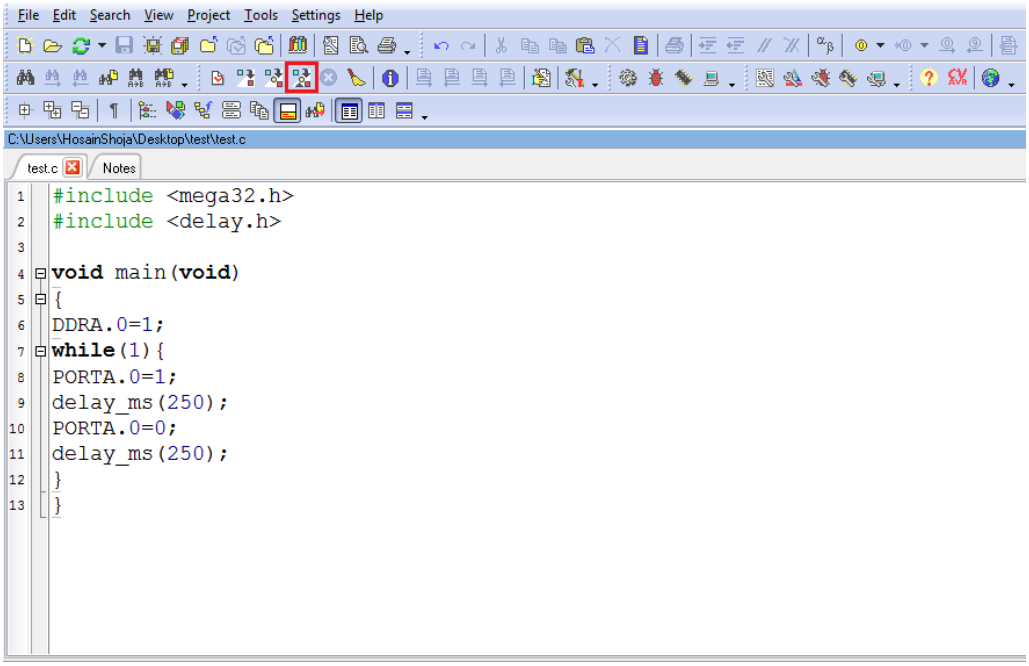
۷. پنجره Project Configure باز می شود که در این مرحله می بایست فایل ذخیره شده با پسوند C. در مرحله ۳ را با زدن دکمه Add به پروژه اضافه نمود.



۸. 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 1MHZ باید قرار گیرد) . در نهایت پنجره Project Configureرا Ok کرده و کار پروژه جدید پایان می یابد.

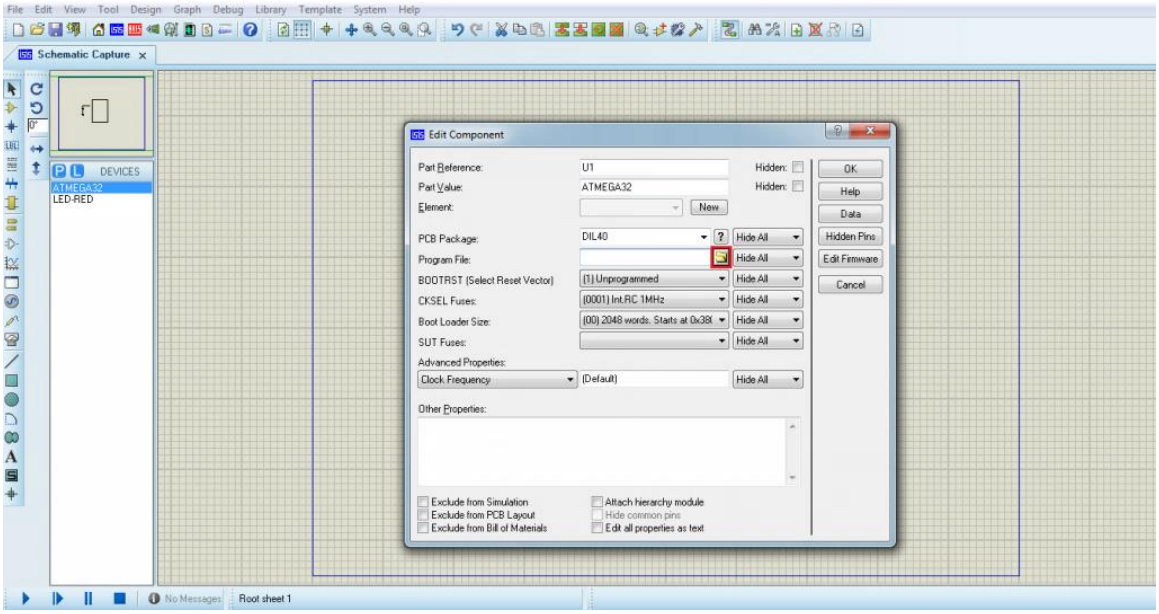


پروژه جدید ساخته شده است حالا باید برنامه دلخواه به زبان C را در این مرحله نوشت. پس در اینجا برنامه نوشته شده مثال شماره ۱ را تایپ می کنیم . بعد از نوشتن برنامه در محیط کدویژن می بایست از منوی Projectگزینه Build را انتخاب کنید تا برنامه کامپایل و ساخته شود. در صورت بروز error باید ابتدا آنها را برطرف 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.

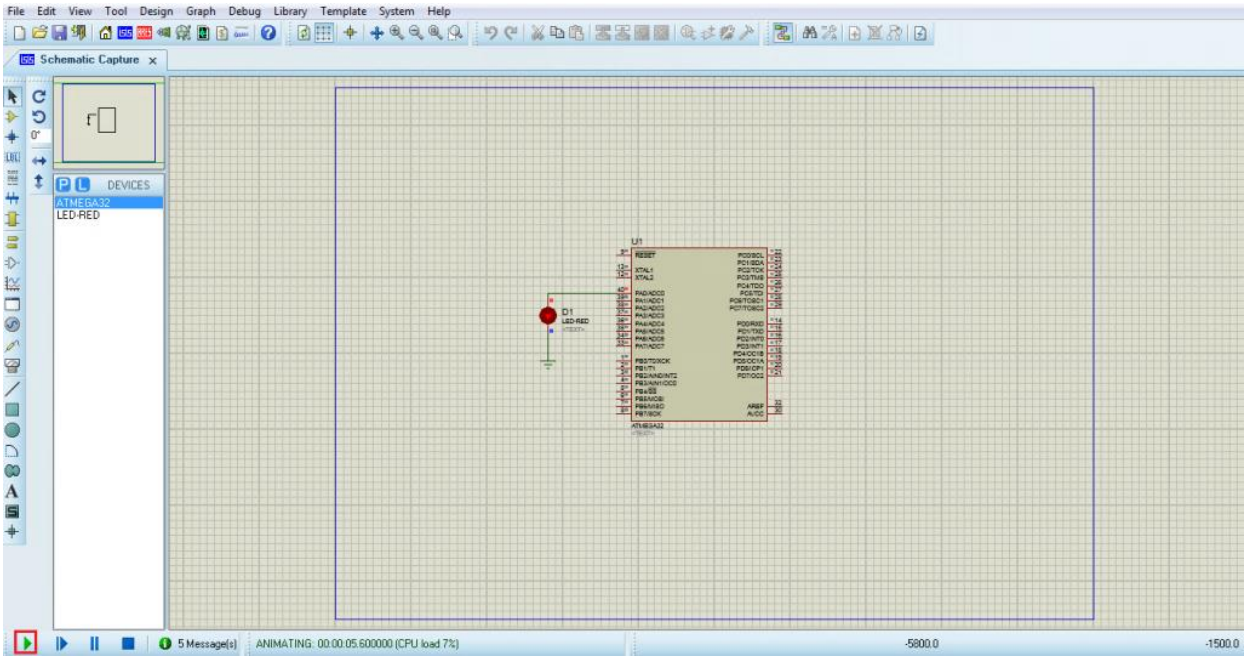


حال نوبت به شبیه سازی پروژه در پروتئوس می رسد.

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 (Browseکلیک کنید تا پنجره انتخاب فایل باز شود. حالا میبایست 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.



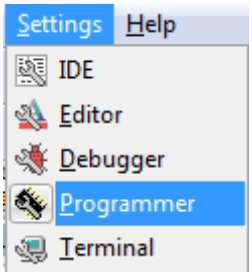
مرحله چهارم : پیاده سازی مدار

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.

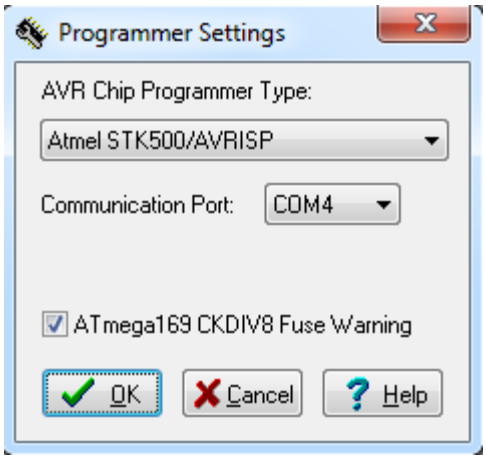
پروگرام کردن میکرو توسط نرم افزار CodeVision

بعد از وصل کردن و روشن کردن پروگرامر خود به پورت مربوطه روی PC یا لپ تاپ و نصب فایل درایور مربوطه (برخی از پروگرامرها برای راه اندازی نیاز به Driver مخصوص دارند) مراحل زیر برای پروگرام کردن را در نرم افزار CodeVision اجرا کنید:

۱. ابتدا از منوی Setting ، گزینه Programmer را انتخاب می کنیم.



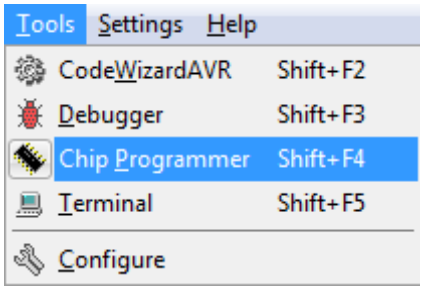
۲. از لیست موجود می بایست نوع پروگرامر را همان مدل پروگرامری که خریداری کرده اید ، انتخاب کنید. (برای من Stk500)



۳. 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 Panel مراجعه کرده و سپس در قسمت Port&LPT نام پروگرامر و پورت Com آن مشخص است.

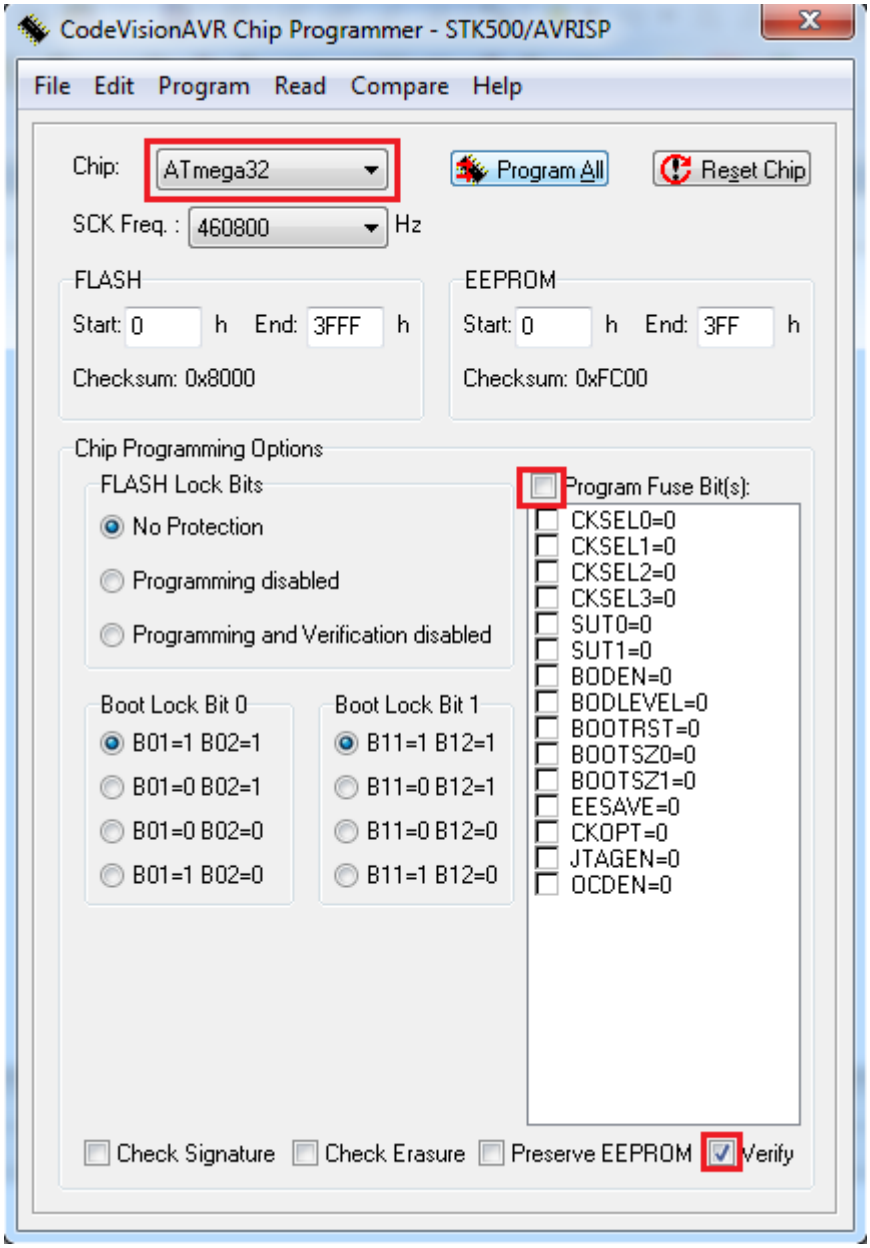
۴. میکروکنترلر را روی پروگرامر در جای مناسب و در جهت صحیح قرار می دهیم.

۵. از منوی Tools ، گزینه Chip Programmer را انتخاب می کنیم.



۶. تیک گزینه Program Fuse Bit را برمیداریم.

۷. با زدن دکمه Program All ، پروگرام آغاز می شود.



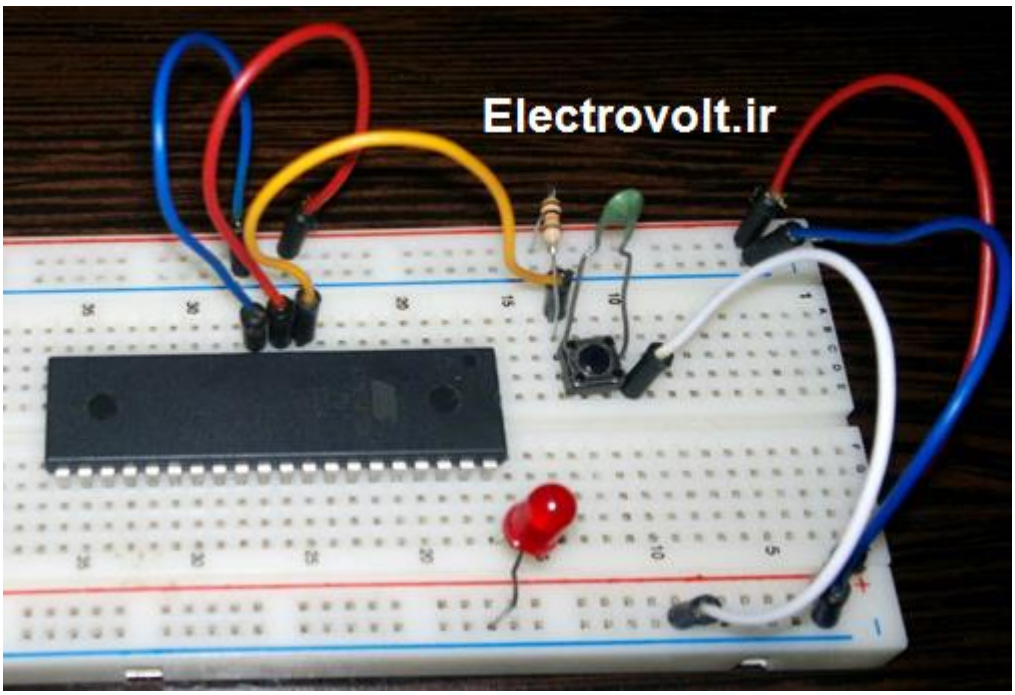
۸. در پیغامی که هنگام پروگرام ظاهر می شود از ما میخواهد تا فایلی برای ریختن دیتا درون EEPROMمعرفی کنید ، چون ما فایلی نداریم پس گزینه Cancel را انتخاب می کنیم.

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.

نکته : به علت اینکه نرم افزار پروتئوس یک شبیه ساز است ، نیازی به قرار دادن مقاومت پول آپ برای پایه ریست درون نرم افزار نیست اما در عمل و به هنگام پیاده سازی بهتر است پایه ریست پول آپ شود و برای ایجاد قابلیت ریست کردن دستی ، یک کلید و خازن هم قرار می دهیم.

شکل زیر مدار نهایی را نشان می دهد.



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.

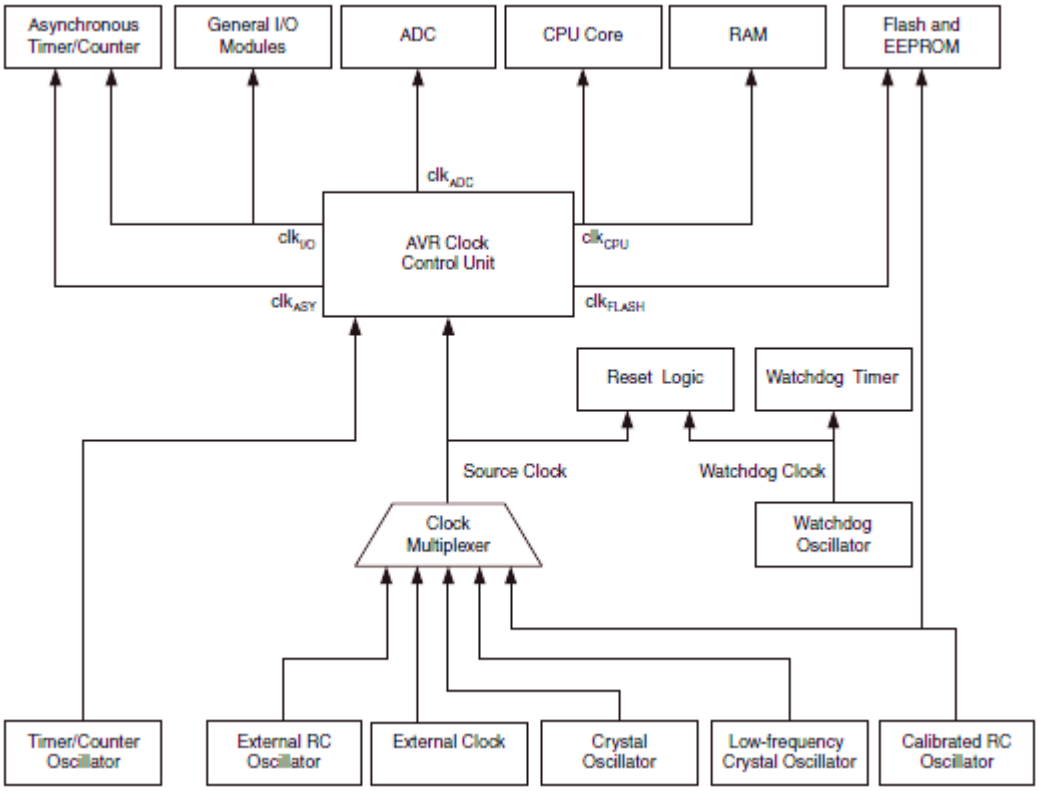
مرحله ششم : تولید ، ارتقا و بهبود

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

دانلود مثال عملی شماره ۱

۵-۶- واحد کنترل کلاک سیستم در میکروکنترلر Atmega32

این واحد یکی از مهمترین قسمت های میکروکنترلرهای AVR می باشد. 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 (انتخاب کننده) ، 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 SRAMاعمال 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 eeprom داخلی فراهم می سازد .پالس CLK/IOبرای تولید پالس ماژول های ورودی و خروجی نظیرUSART ، SPI، شمارنده ها و وقفه ها بکار برده می شود. پالس CLKASY برای راه اندازی آسنکرون تایمر یا کانتر شماره دو ، برای استفاده از فرکانس 32768HZ اسیلاتور RTC است. در این شکل همچنین نحوه اتصال واحد تایمر Watchdog نیز 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.

۵-۷- 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 AVR بسته 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 Byte و 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:

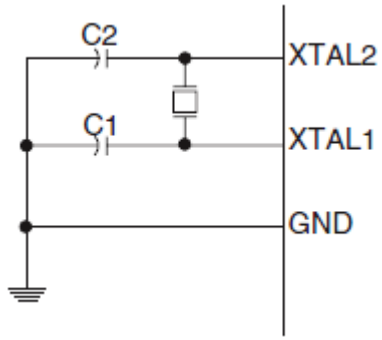
۵-۸- فیوز بیت های تنظیم کلاک

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 SUT0و 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

۵–۸–۱– 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 XTAL1 و XTAL2 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 طراحی 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 16MHZ استفاده کنیم باید فیوز بیت CKOPT فعال گردد در غیر اینصورت حداکثر کریستال خارجی ممکن برای اتصال به میکرو 8MHZ خواهد 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 Reset شده 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

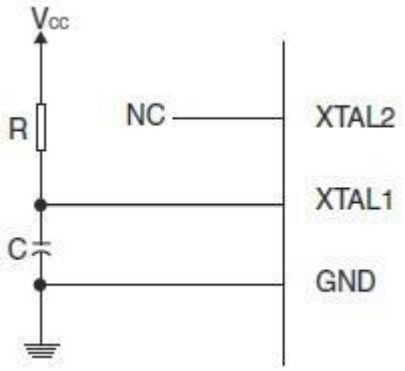
۵–۸–۲– نوسان ساز با کریستال فرکانس پائین

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 XTAL1و XTAL2 با زمین 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	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		

۵–۸–۳– نوسان ساز با RC خارجی

از اسیلاتور RC خارجی در کاربردهایی که به تغییرات 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 22pF باشد.



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 اگر فیوز بیت CKOPT فعال شود خازن 36pF داخلی بین دو پایه XTAL1 و GND فعال می گردد و در نتیجه میتوان مقدار 36pF را در محاسبات لحاظ نمود (C+36) یا حتی میتوان خازن C خارجی را حذف نمود. 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

۵-۸-۴- نوسان ساز با اسیلاتور RC کالیبره شده داخلی

اسیلاتور RC کالیبره شده داخلی می تواند فرکانس های ثابت 1MHZ، 2MHZ، 4MHZ و 8MHZ را در شرایط تغذیه 5v+ و در دمای 25 درجه ایجاد 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 USART و دیگر پروتکل ها و برنامه هایی که به زمان بسیار وابسته هستند از کریستال خارجی استفاده کنید.

نکته : فیوزبیت های میکروکنترلر ATmega32 به طور پیش فرض توسط کارخانه طوری تنظیم شده است که از اسیلاتور کالیبره شده داخلی با فرکانس 1MHZ استفاده می کند.

زمانی که اسیلاتور داخلی استفاده می شود نیازی به قرار دادن اسیلاتور خارجی نیست و پایه های XTAL1 و XTAL2 آزاد 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	

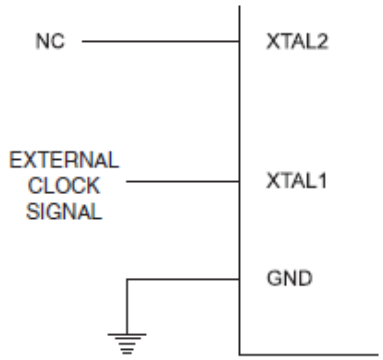
در صورت استفاده از اسیلاتور RC کالیبره شده داخلی در هر بار که میکروکنترلر Reset می شود ، مقدار رجیستر OSCCAL خوانده شده و اسیلاتور به طور خودکار تنظیم می گردد. در حالت پیش فرض مقدار این رجیستر طوری تنظیم شده است که اسیلاتور RC روی ۱ مگاهرتز کار کند. کاربرد این رجیستر برای کاهش

خطای اسیلاتور RC داخلی در شرایط مختلف است اما اگر از شرایط تغذیه ۵ ولت و دمای ۲۵درجه استفاده گردد این خطا به ۱۰٪ کاهش می یابد. با نوشتن مقدار 0x00 کمترین و با نوشتن مقدار 0xFF در 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

۵–۸–۵– 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 XTAL1 با 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		

۵–۹– تنظیم دیگر فیوز بیت ها

- **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 OCDENبرنامه میکروکنترلر را به طور آنلاین در حین اجرا توسط مدار واسطی که از ارتباط سریال JTAGاستفاده 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 SLEEP کار 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 BOOTرا 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 ۲۵۶	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 RESET از 0000 است و اگر این بیت فعال شود به آدرسی که فیوز بیت های BOOTSZ0 و BOOTSZ1 مشخص کرده اند تغییر می یابد.

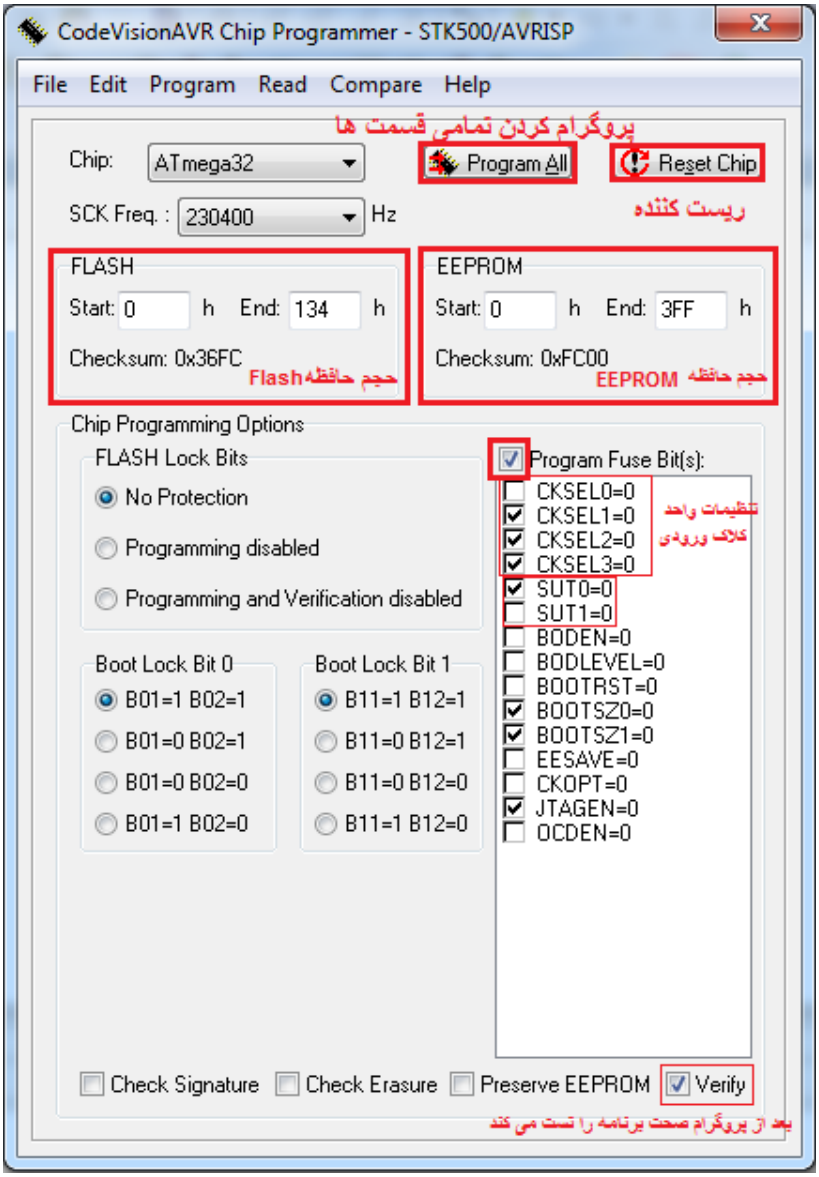
- **فیوز بیت BODEN** : مدار BROWN-OUT آشکارساز ولتاژ تغذیه است که اگر از ۲,۷ یا ۴ ولت کمتر شود میکروکنترلر را ریست می کند. برای فعال کردن این مدار، باید فیوزبیت BODEN فعال گردد. این فیوز بیت بطور پیش فرض غیرفعال است.
- **فیوز بیت BODLEVEL** : اگر فیوزبیت BODEN فعال و فیوز بیت BODLEVEL غیرفعال باشد با 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-OUT تعیین 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

۵-۱۰- تنظیم فیوز بیت ها در نرم افزار کدویژن

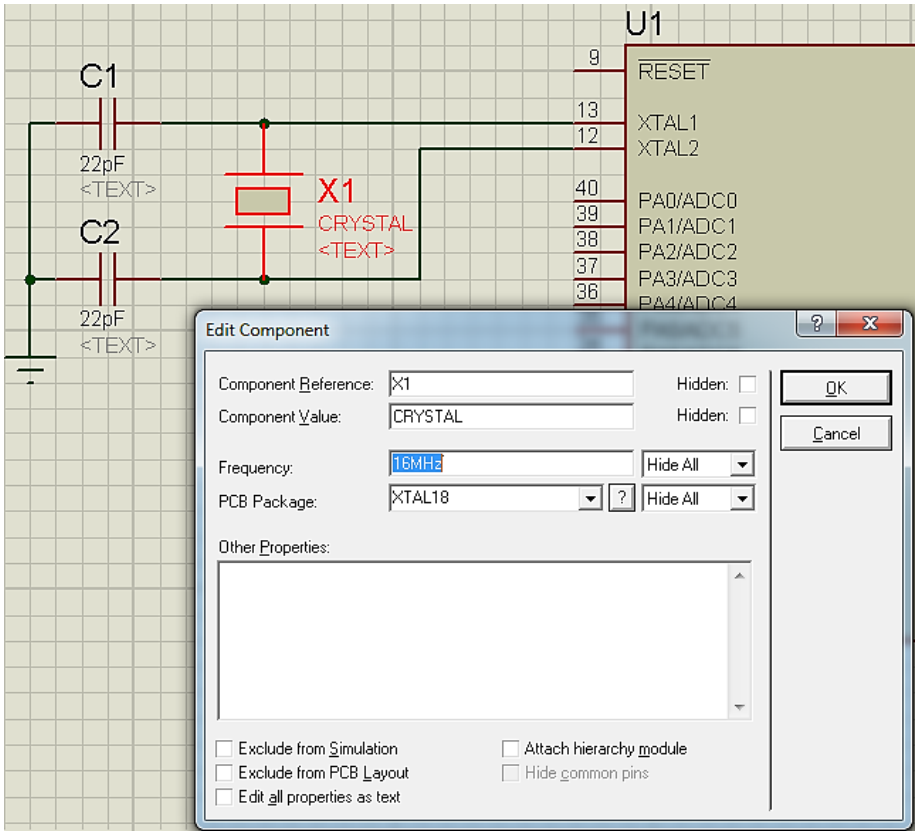
بار دیگر به نرم افزار CodeVision بر می گردیم. در شکل زیر محیط Chip Programmer و کاربرد قسمت های مختلف آن را برای تنظیمات Fuse Bits میکروکنترلر Atmega32 مشاهده می کنید. با تنظیم کردن این پنجره و سپس پروگرام کردن میکروکنترلر تغییرات مورد نظر در آن اعمال می شود. همچنین در شکل زیر تنظیمات فیوز بیت ها را در حالت default میکروکنترلر atmega32 در حالت 1Mhz داخلی (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 زیر **1MHz داخلی** درآورید.



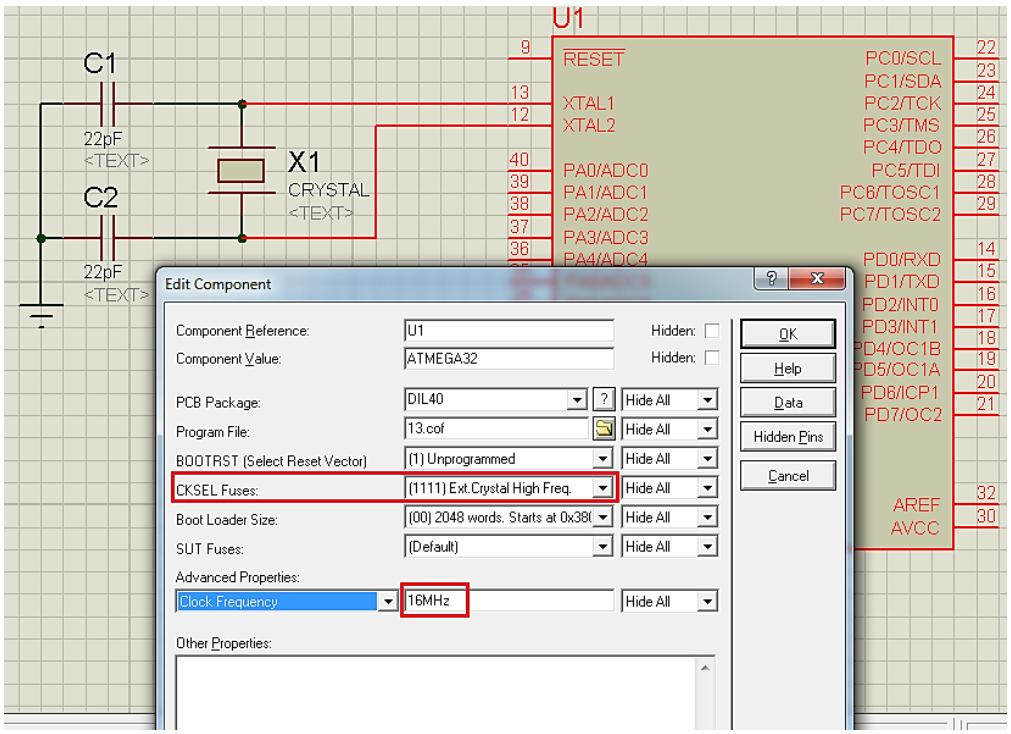
۵–۱۱– تنظیم پروتئوس در حالت استفاده از کریستال خارجی

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



فصل ۶– آموزش برنامه نویسی C

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.

۶–۱– 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 و 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.

۶-۲- 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

نکته ۱ : کلمه کلیدی auto از کامپایلر Codevision حذف شده است.

نکته ۲ : کلمات کلیدی زیر به کامپایلر کدویژن اضافه شده است:

bit	defined	interrupt
flash	eeprom	inline
	sfrb	sfrw

۶-۳- ویژگی های یک برنامه به زبانC

-هر دستور در زبان سی با ; (سمی کالن) 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.

۶–۴– 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:

۱. 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.
۲. 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.

۶–۵– 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.

۶-۶- 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
}
}
```

۶-۷- 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

۶-۷-۱- نحوه تعریف متغیرها

متغیر ها به صورت زیر تعریف می شوند:

نوع متغیر نام متغیر = مقدار اولیه;

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.

۶-۷-۲- 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.

۶-۷-۳- 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:

۱. متغیرهای عمومی(Global)
۲. 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.

۶-۷-۴- 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 SRAM ذخیره 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 SRAM پاک خواهد شد و متغیرهایی که باید ذخیره دائمی شوند می بایست در حافظه EEPROM یا FLASH تعریف و ذخیره شوند.

مثال:

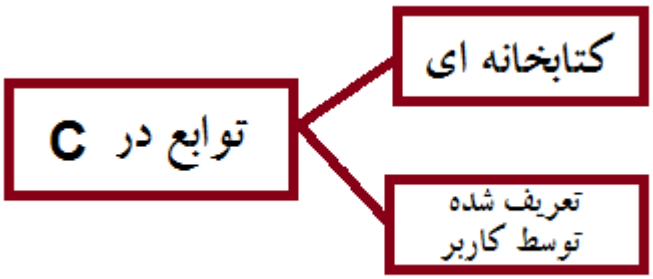
```
int a;
eeprom char b;
flash float c;
```

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.

۶-۸- 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 main نام 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.

۶-۸-۱- 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:

۱. **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 );
```

۲. **تابع با ورودی ، بدون خروجی**

مثال : تابع دارای یک ورودی int و بدون خروجی

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

۳. **تابع بدون ورودی ، با خروجی**

مثال : تابع بدون ورودی اما دارای خروجیint

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

۴. **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 void استفاده 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 return به خروجی تابع و برنامه اصلی بر گردانیم.

۶-۸-۲- 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:

۱. اعلان تابع قبل از تابع main باشد و بدنه تابع بعد از تابع main باشد.
۲. اعلان تابع و بدنه تابع هر دو قبل از تابع main باشد.
۳. 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.

۶-۸-۳- اعلان و بدنه تابع

یک تابع به صورت زیر اعلان می شود. اعلان تابع همیشه قبل از تابع main باید صورت گیرد.

;(... , 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 );
```

بدنه تابع می تواند قبل از تابع main و یا بعد از آن باشد. در صورتی که بخواهیم بدنه تابع بعد 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 );

void main ( void )
{
…
}

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

در صورتی که بخواهیم بدنه تابع قبل از تابع main 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 و دیگر بعد از تابع main چیزی نمی آید.

مثال :

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

void main ( void )
{
…
}
```

۶-۸-۴- 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 0 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.

۶-۹- تعریف ثوابت 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.

۱. استفاده از دستورConst

مثال:

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

۲. 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.

۶-۱۰- دستورات شرطی در C

اگر بخواهیم تحت شرایطی تعدادی از دستور ها اجرا شوند و یا تعدادی دیگر اجرا نشوند، باید از ساختار های شرطی استفاده کنیم. دستورات شرطی در تمام زبان های برنامه نویسی از اجزای اصلی هستند. یک دستور شرطی، شرطی که کاربر مشخص می کند را بررسی کرده و در صورت صحیح بودن یک دسته از کدهای کاربر و در صورت غلط بودن دسته‌ی دیگری از کدهای کاربر را اجرا می‌کند.

۶-۱۰-۱- دستور شرطی if

```
if( شرط ) دستور;

if( شرط ) {دستورات;}

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

همان‌طور که در بالا مشاهده می‌شود، اگر شرط موردنظر برقرار باشد، کدهای درون { } در زیر if اجرا خواهند شد. اگر شرط موردنظر برقرار نباشد، کدهای درون { } در زیر else اجرا خواهند شد به مثال زیر توجه کنید:

مثال :

```
if(a==7)
        PORTD=0b11110000;
else
        PORTD=0b00001111;
```

توضیح : در مثال بالا اگر متغیر a برابر با عدد ۷ باشد، مقدار باینری ۱۱۱۱۰۰۰۰ به رجیسترPORTD نسبت داده می‌شود و اگر برابر با ۷ نباشد، مقدار باینری ۰۰۰۰۱۱۱۱ به رجیستر PORTD نسبت داده می‌شود. دقت

شود، در سطرهای بالا دو علامت { } وجود ندارد و این به‌این‌علت است در زیر دستور if و else تنها یک دستور وجود دارد، اگر تعداد دستورات بیشتر از یک دستور باشد حتماً باید از { } استفاده شود. وجود دستور elseدر if ضروری نیست و بسته به نیاز کاربر قرار داده می‌شود.

۶-۱۰-۲- دستور شرطی switch

ساختار switch یکی از ساختار های مهم و جالب در زبان C است. از این ساختار برای تصمیم گیری های چندگانه بر اساس مقادیر مختلف یک عبارت، استفاده می شود. به طور کلی، در تمام تصمیم گیری هایی که بیش از سه انتخاب وجود داشته باشد بهتر است از ساختار switch استفاده شود. دستور شرطی switch نسبت به دستور شرطی if از توانمندی کمتری برخوردار است . این دستور فقط می تواند برابری را مورد شرط قرار دهد . قالب دستور شرطی switch را در زیر مشاهده می کنید:

```
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
}
```

گزینه Default در ساختار دستور switch ضروری نیست و بسته به نیاز کاربر قرار داده می شود. در دستور switchمقادیر صحیح مورد بررسی قرار می گیرند . هر دستور case یک شرط برابری را مورد بررسی قرار می دهد و در صورت برابر بودن ، کد های قرار گرفته در زیر آن اجرا شده و توسط دستور Break از ساختار switchخارج می شود. اگر در زیر هر case دستور Break قرار داده نشود، هر caseبا case بعدیor منطقی (یا) می شود. در بعضی شرایط به این صورت می توان ۲ مقدار را بررسی کرد.

۶-۱۱- حلقه های تکرار در C

یکی دیگر از از اجزا اصلی زبان های برنامه نویسی حلقه ها هستند. حلقه های تکرار تحت شرایط خاصی، یک یا چند دستور را چندین بار اجرا می کنند. به عنوان مثال، اگر بخواهیم تعداد ۱۰۰ عدد را از ورودی بخوانیم و

آن ها را با هم جمع کنیم. باید عمل خواندن عدد را ۱۰۰ بار تکرار کنیم. عملکرد یک حلقه به این صورت است که کد مربوط به حلقه تا زمانی که شرط حلقه برقرار باشد، تکرار می شود و تا زمانی که حلقه در حال اجرا است، برنامه در محل حلقه می ماند.

۶-۱۱-۱- حلقه while

در این نوع حلقه 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 }
```

مثال :

```
char a=0;
while( a<16 )
{
   PORTD = a;
   a++;
}
```

۶-۱۱-۲- حلقه do…while

این حلقه عملکردی بسیار شبیه به حلقه while دارد. در این حلقه یکبار کدهای درون حلقه اجرا می شود و سپس شرط حلقه بررسی می گردد. بنابراین برای اینکه شرط در آخر بررسی شود از حلقه do…while استفاده می شود.

```
do
{
      کد هایی که در زمان برقراری شرط حلقه تکرار می شوند//
}
While( شرط حلقه );
```

۶-۱۱-۳- حلقه for

از این حلقه در حالتی که تعداد دفعات تکرار حلقه از قبل مشخص باشد، به کار می رود. در این حلقه، متغیری وجود دارد که تعداد دفعات تکرار حلقه را کنترل می کند. این متغیر را اندیس حلقه تکرار یا شمارنده می گویند. اندیس حلقه دارای یک مقدار اولیه است که طی هر بار اجرای دستورات حلقه، مقداری به آن اضافه می شود. به این مقدار اضافه شده در هر بار اجرای حلقه، گام حرکت گویند. گام حرکت می تواند عددی صحیح یا اعشاری، مثبت یا منفی ویا کاراکتری باشد. یکی دیگر از اجزای حلقه for شرط حلقه است. شرط

حلقه مشخص می کند که دستورات داخل حلقه for تا کی باید اجرا شوند. اگر این شرط دارای ارزش درستی باشد، دستورات داخل حلقه اجرا می شوند وگرنه کنترل برنامه از حلقه خارج می شود .حلقه for دارای یک متغیر شمارشی است که نسبت به تعداد تکرار حلقه، طول آن مشخص شده و در ابتدای برنامه باید معرفی شود. به طور مثال اگر تعداد تکرار حلقه ایی ۲۰۰ مرتبه باشد، می توان متغیر شمارشی از نوع unsigned charمعرفی کرد، اما اگر تعداد تکرار حلقه ۱۰۰۰ مرتبه باشد، باید متغیری با طول بیشتر مثل نوع int را برای متغیر شمارشی انتخاب کرد. این متغیر شمارشی در بدنه حلقه کاربرد فراوانی دارد. دستور for را به شکل زیر می توان به کار برد:

For(مقدار اولیه شمارنده حلقه ; شرط حلقه ; گام حرکت حلقه)
{ ; کدهایی که تا زمان برقراری شرط حلقه تکرار می شود }

مثال :

```
int  i;
…
for   (i=0  ;  i<8  ;  i++)
   a[i]=i;
```

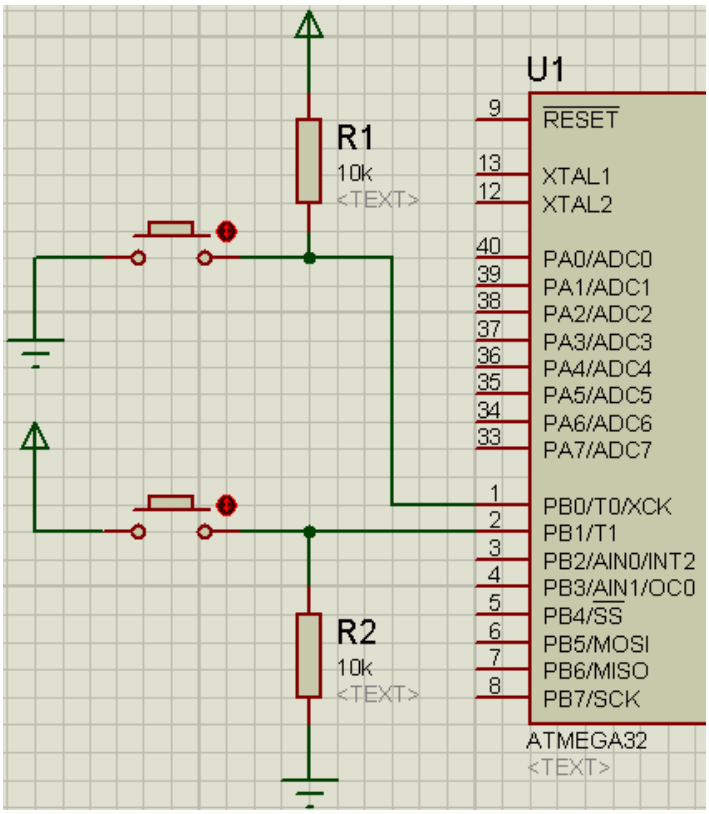
توضیح : در ابتدای حلقه for مقدار متغیر i برابر با مقدار صفر قرار داده شده است و شرط حلقه تا وقتی برقرار است که مقدار متغیر i کوچکتر از عدد ۸ باشد. گام حلقه به شکلی تنظیم شده که هر بار یک واحد به متغیر شمارشی اضافه می کند. در ابتدا مقدار متغیر i برابر با صفر است و حلقه شروع می شود، کد های درون حلقه اجرا شده سپس یک واحد به متغیر i اضافه می شود. تکرار کد های درون این حلقه، طبق شرط حلقه ۸ مرتبه می باشد و به متغیر i مقادیر ۰ تا ۷ تعلق می گیرد. از تغییرات متوالی مقدار متغیر i می توان در حلقه استفاده کرد. همان طور که در مثال بالا مشاهده می شود، یک آرایه با نام a قبلا معرفی شده است، به جای اندیس آرایهa ، متغیر i قرار گرفته است؛ بنابراین با هر بار تکرار حلقه و تغییر مقدار متغیرi ، اندیس آرایه نیز تغییر می کند و هر بار مقدار i به عضو i ام آرایه a انتساب داده می شود. به طور مثال اگر حلقه در حرکت سوم قرار گیرد، یعنی مقدار i برابر با عدد ۲ است. در این حالت کد داخل حلقه به صورت a[2]=2 می شود.

۶-۱۱-۴- دستور break و continue در 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 continue در هر نقطه از حلقه ، کدهای زیر دستور continue را رها کرده و به ابتدای حلقه باز می گردد.

۶-۱۲-اتصال کلید به میکرو

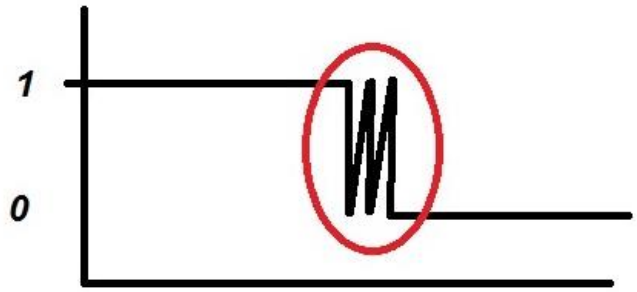
یکی از ساده ترین و پرکاربرد ترین دستگاههای ورودی کلید است . توسط کلید میتوان منطق ۰ یا ۱ را به میکرو وارد کرد و بر اساس آن پردازش را انجام داد. برای خواندن منطق کلید از رجیستر PIN استفاده می شود . نحوه استفاده از رجیستر PIN به این صورت است که : " اگر کلید زده شد آنگاه کار مورد نظر انجام شود " . عبارت 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 . اتصال کلید 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 10k برای 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) {...} //PullUp
if(PINB.1==1) {...} //PulDown
```

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 . شکل زیر این موضوع را نشان می دهد.



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 delay در 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 while با 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:

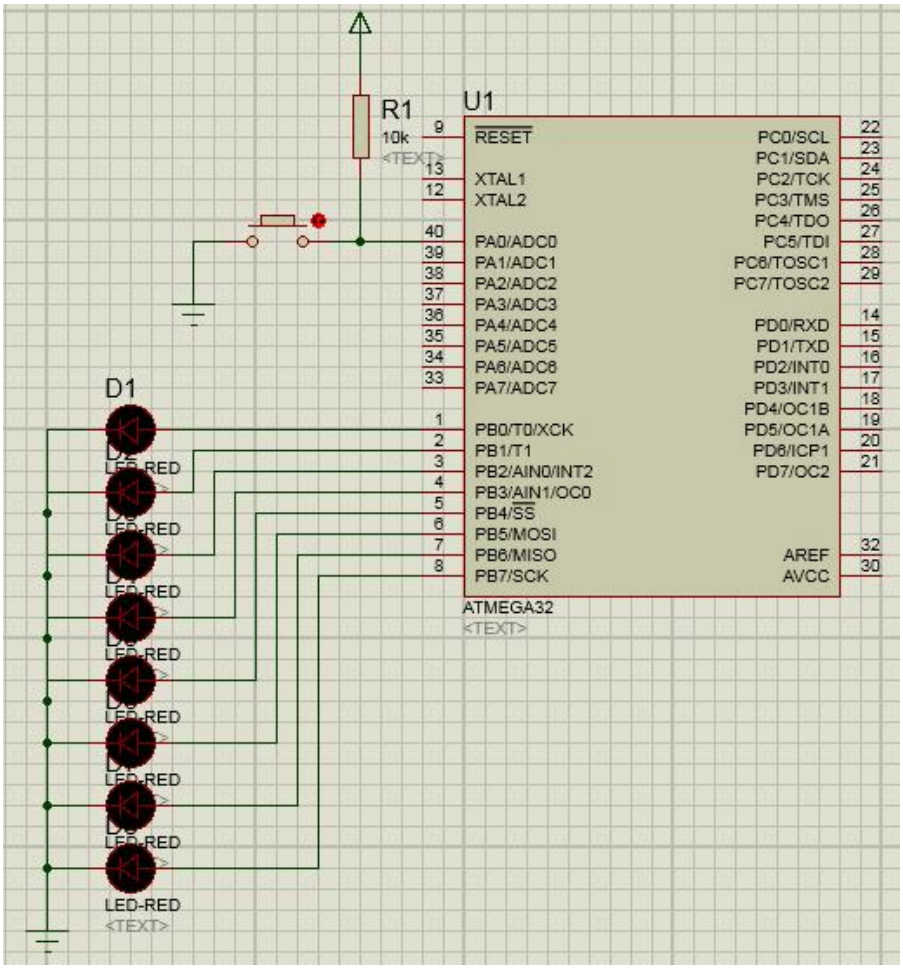
```
if((PINB.0==0 ) && (flag==0)) {
flag=1; start=!start; }
else if ( PINB.0 == 1) flag=0;
if(start){
دستورات مربوط به بعد از زدن کلید
}
```

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 ، ledهای موجود به صورت ۱. شمارنده بالا شمار ۲. شمارنده پایین شمار ۳.شمارنده جانسون ۴. شمارنده حلقوی عمل نماید . سعی کنید برنامه کمترین عیب ممکن را داشته باشد و کلید به بهترین نحو ممکن کار کند.

حل :

مرحله اول : طراحی سخت افزار



مرحله دوم : طراحی نرم افزار

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=4شده 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.

دانلود مثال عملی شماره ۲

۶-۱۳- آرایه ها در C

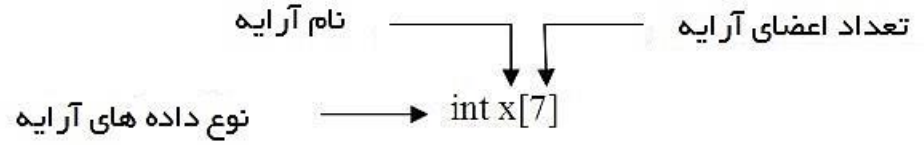
آرایه اسمی برای چند متغیر هم نوع می باشد یا به عبارت دیگر آرایه از چندین کمیت درست شده است که همگی دارای یک نام می باشد و در خانه های متوالی حافظه ذخیره می گردند. هر یک از این کمیت ها را یک عنصر می گویند، برای دسترسی به عناصر آرایه باید اسم آرایه و شماره ی اندیس آرایه را ذکر کنیم. آرایه ها

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.

۶-۱۳-۱- آرایه های 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 .



اندیس آرایه	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]
اعضای آرایه	50	60	40	20	8	6	9

میزان حافظه ای که به آرایه اختصاص داده می شود، به این شکل محاسبه 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-1است 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 ۱ :اندیس آرایه خود می تواند متغیر باشد و این قابلیت می تواند در برنامه ها کاربرد زیادی داشته باشد .

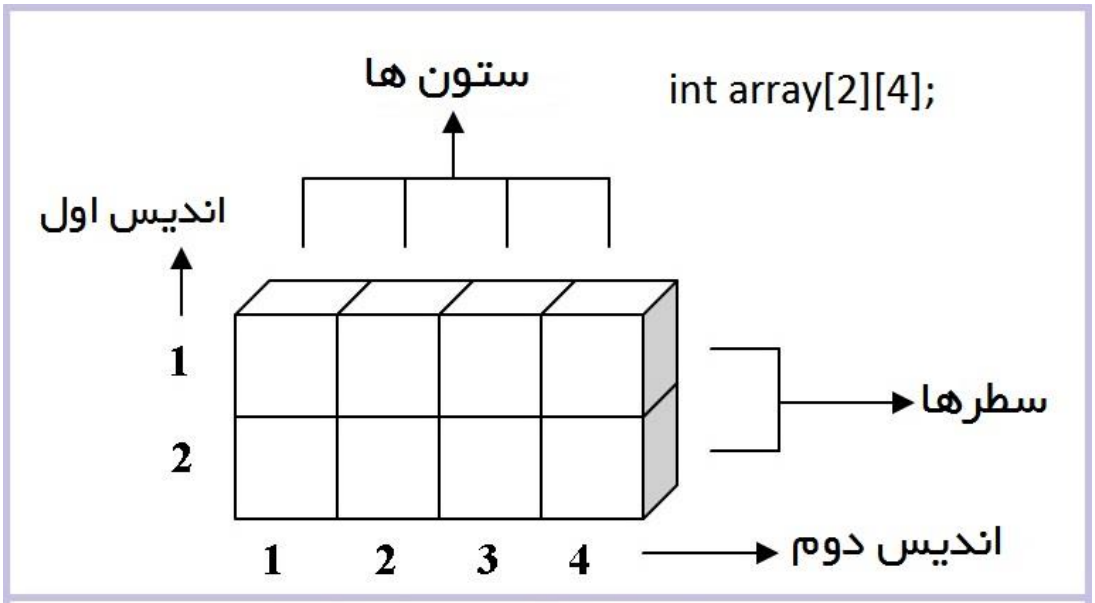
نکته ۲ :در صورتی که می خواهید آرایه به صورت دائمی ذخیره شود (مثلا وقتی که میخواهید لوگو شرکت خود را همیشه داشته باشید) باید به ابتدای تعریف کلمه کلیدی eeprom یا flash را اضافه کنید تا در حافظه های دائمی ذخیره گردد.

نکته ۳ : در صورتی که اندیس آرایه را ننویسیم ، یک آرایه با طول اتوماتیک بوجود می آید. یعنی به تعدادی که در ابتدای برنامه آرایه مقدار می گیرد ، به همان اندازه از حافظه مصرف می شود. مثال:

```
int a[]={1,2,3,4,5,6};
```

۶-۱۳-۲- 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 دو بعدی با ۸ عضو

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 intمی باشد.

نکته : تعریف آرایه ها با مجموعه عناصر زیاد در حافظه SRAM به علت محدودیت در حجم حافظه ممکن است باعث ایجاد مشکل شود. بنابراین معمولا آرایه های با حجم زیاد را در بخش خالی حافظه flash ذخیره می کنند.

۶-۱۳-۳- مقدار دهی به آرایه های چند بعدی

برای مقدار دهی اولیه 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 اولیه 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} } }
```

برای مقدار دهی ثانویه در حین برنامه میتوان به صورت زیر عمل کرد :

```
a[0][1][3]=2354;
```

نکته ۱ : برای مقدار دهی ثانویه اندیس ها از 0 تا n-1 میتواند باشد.

نکته ۲ : در مقدار اندیس آرایه های چند بعدی ، به جای عدد میتوان از متغیر استفاده کرد.

۶-۱۴- رشته ها در C

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 (مخفف American Standard Code For Information Interchange) می باشند . کدهای اسکی 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]

۶-۱۴-۱- تعریف یک کاراکتر

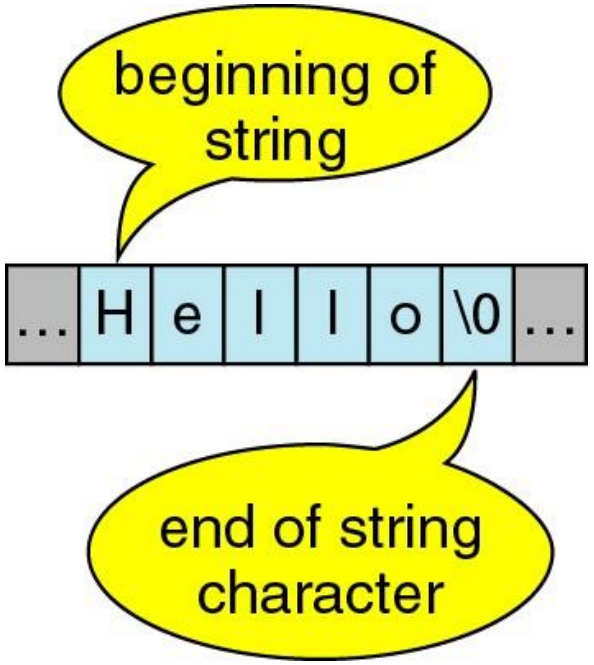
تعریف یک کاراکتر توسط نوع متغیر char صورت می گیرد و مقدار اولیه آن داخل کوتیشن (' ') قرار می گیرد . در زبان سی وقتی یک حرف بین ' و ' قرار می گیرد کد اسکی آن درون متغیر ذخیره می شود . مثال:

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

۶-۱۴-۲- تعریف رشته (آرایه ای از کاراکتر ها)

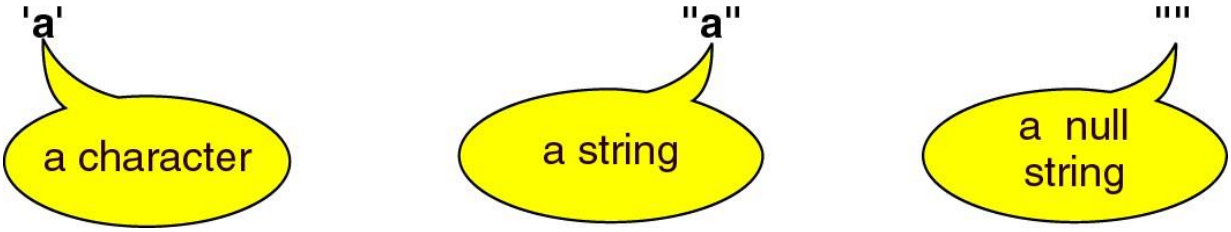
برای تعریف رشته از آرایه ای از کاراکترها استفاده می شود و مقدار اولیه آن داخل دابل کوتیشن (" ") قرار می گیرد . به‌طور مثال برای کلمهHello ، کدهای اسکی تمام حروف آن به همراه یک کاراکتر تهی که در انتهای آن‌ها قرار می‌گیرد، نیاز است.

```
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!..."
```



۶-۱۴-۳- کاراکترهای کنترلی

در محتویات یک رشته علاوه بر کاراکترهای قابل‌چاپ می‌توان کاراکترهای کنترلی قرارداد. این کاراکتر ها برای کنترل صفحه نمایش میباشند و به شکل جدول زیر میباشند:

کاری که انجام میشود	کاراکتر کنترلی
به خط بعد میرود	\n
(به اندازه ۸ فاصله به جلو میرود(تب	\t
بوق سیستم را به صدا در می آورد	\a
کاراکتر \ را چاپ میکند	\\
کاراکتر " را چاپ میکند	\”
به ۸ خط بعد میرود	\v
کاراکتر قبل از خودش را حذف میکند(بک اسپیس)	\b
کلید را مشخص میکند	\r
علامت ? را چاپ میکند	\?
علامت : را چاپ میکند	\:

۶–۱۵– عملگرها

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

۶–۱۵–۱– عملگرهای محاسباتی

عملگرهای محاسباتی، عملگرهایی هستند که اعمال محاسباتی را روی عملوندها انجام می‌دهند. عملگر ٪ برای محاسبه باقی‌مانده تقسیم به کار می‌رود. این عملگر عملوند اول را بر عملوند دوم تقسیم می‌کند (تقسیم صحیح) و باقیمانده را برمی‌گرداند. در جدول‌های زیر، عملگرهای محاسباتی و تقدم آن‌ها در یک معادله مشاهده می‌شود:

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

دو عملگر ++ و– همان‌طور که مشاهده می‌کنید در طرف چپ و راست متغیر قرارگرفته‌اند. اگر به‌تنهایی و در یک خط دستور به‌کاربرده شوند، فرقی نمی‌کند اما اگر در یک معادله به‌کاربرده شوند، اینکه طرف راست یا چپ قرار گیرند، متفاوت است. به مثال زیر توجه کنید:

```
x++;
++x;
y = ++x;
y = x++;
```

همان‌طور که در مثال مشاهده می‌کنید، در دو دستور اول به متغیر x یک واحد اضافه می‌شود. در دستور سوم، ابتدا یک واحد به مقدار متغیر x اضافه‌شده و سپس درون متغیر y قرار می‌گیرد. در دستور چهارم، ابتدا مقدار متغیر x درون متغیر y قرارگرفته و سپس یک واحد به آن اضافه می‌شود. اگر تا قبل از رسیدن به دستور چهارم، مقدار x برابر با ۱۰ باشد، پس از گذشتن از دستور چهارم، مقدار y برابر ۱۰ و x برابر با ۱۱ است.

۶-۱۵-۲- عملگرهای مقایسه‌ای و منطقی

عملگرهای مقایسه‌ای ارتباط بین عملوند ها را مشخص می کنند و عملگر های منطقی بر روی عبارات منطقی عمل می کنند. عبارات منطقی دارای دو ارزش درستی و نادرستی اند و زمانی که باید یک شرط مورد بررسی قرار بگیرد، استفاده می شوند. به طور مثال برای بررسی مساوی بودن دو متغیر از عملگر == استفاده می شود. در زبانc ، ارزش نادرستی با ۰ و ارزش درستی با مقادیر غیر صفر مشخص می شود.

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

۶-۱۵-۳- عملگرهای ترکیبی

این عملگر ها ترکیبی از عملگر مساوی و عملگرهای دیگر هستند. به طور مثال عملگر +=در نظر بگیرید، اگر به شکل x += a نوشته شود، برابر با x = x+a است؛ یعنی هر بار متغیر x را با متغیر a جمع می کند و درون متغیر x قرار می‌دهد .برای دیگر عملگر های ترکیبی نیز به همین صورت است. این عملگر ها پایین ترین تقدم را در بین عملگرهای دیگر دارند. این عملگر ها، عملگر های حسابی را شامل می شوند. در جداول زیر عملگرهای ترکیبی و تقدم آنها را میبینید.

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	عملگر
۱	%= و *= و /=
۲	-= و +=

۶-۱۵-۴- تعریف عملگرهای بیتی

عملگر های بیتی بر روی هر بیت یک بایت اثر می گذارند . به طور مثال ، aیک متغیر ۸ بیتی با مقدار باینری ۱۱۱۰۰۱۰۱ است ، اگر عملگر not را روی a اثر دهیم ، نتیجه معکوس شدن هر بیت هست a =~ a . برابر با مقدار باینری ۰۰۰۱۱۰۱۰ است .عملگرهای دیگر نیز به همین شکل اثر خود را اعمال می‌کنند
.

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

در جدول‌های زیر عملگرهای بیتی و تقدم آن‌ها را مشاهده می‌کنید:

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	عملگر
۱	~
۲	<< و >>
۳	&
۴	^
۵	|

۶-۱۵-۵- 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 .مثال:

```
y=(x+(a/(3+g)))*2;
```

همان‌طور که در مثال بالا مشاهده می‌کنید، ابتدا g+3 انجام می‌شود، در مرحله بعد (a/(3+g))انجام می‌شود، در مرحله بعدی (x+(a/(3+g))) انجام می‌شود و در انتها 2*(x+(a/(3+g))) انجام می‌شود.

۶–۱۶– تبدیل نوع در محاسبات

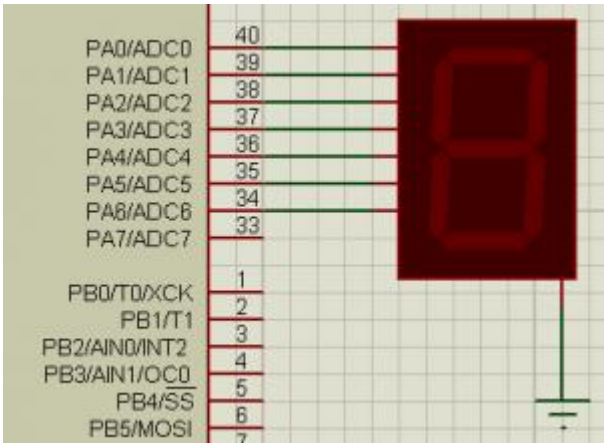
برای درک بهتر در مورد تبدیل نوع در محاسبات ، به مثال زیر توجه کنید.

```
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 int هستند باید یک تبدیل نوع در محاسبه توسط عبارت (float) در ابتدای محاسبه انجام شود.

۶–۱۷– 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)



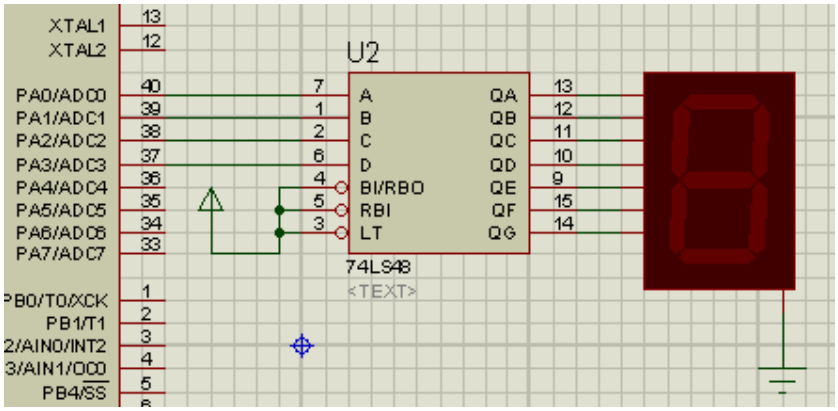
برای نشان دادن اعداد روی سون سگمنت کافی است پایه مربوطه را پس از خروجی کردن یک کنیم برای مثال برای نشان دادن عدد یک باید PA1 و PA2 را ۱ نمود بنابراین دستور PORTA=0x06 عدد ۱ را روی سگمنت نمایش می دهد . بنابراین در این روش برای راحتی هنگام کار با سون سگمنت آرایه زیر را که شامل کد سون سگمنت اعداد ۰ تا ۹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

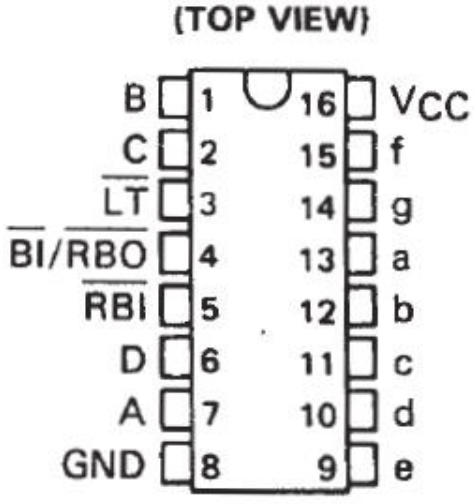
```
PORTA=seg[ i ];
```

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



۶-۱۷-۱- 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 D نشان 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.

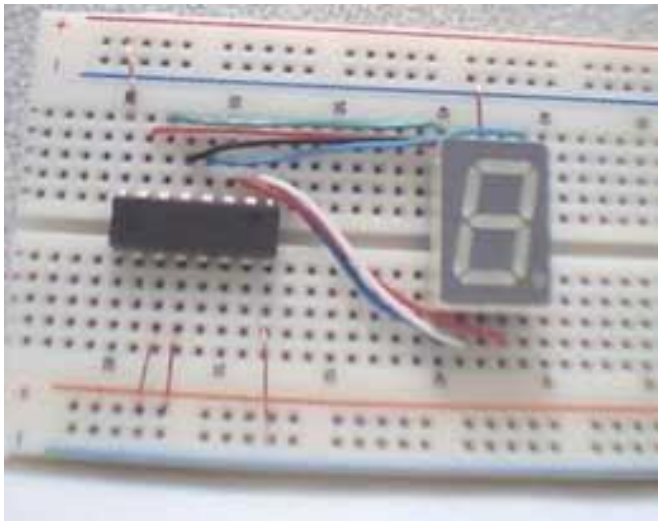
از طرفی خروجی این آی سی کد های مخصوص سون سگمنت است که پس از نمایش روی سون سگمنت و در مبنای ۱۰ نمایش می یابد. (خروجی این آی سی تنها اعداد ۰ تا ۹ میتواند باشد)

تغذیه آی سی:

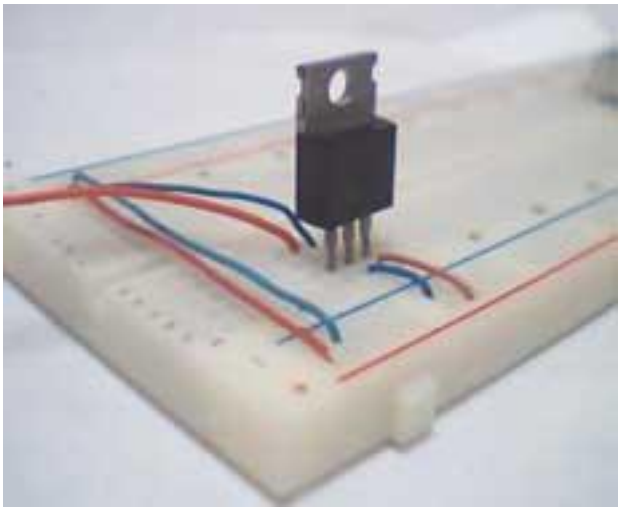
آی سی هایی که شماره آنها با ۷۴ شروع می شود ، اصطلاحا TTL نامیده می شوند .این آی سی ها نسبت به ولتاژ تغذیه بسیار حساسند و ولتاژ آنها می بایست دقیقا ۵ ولت باشد. حداکثر تغییرات ولتاژی که می توان اعمال کرد در حد ۴،۷۵ تا ۵،۲۵ ولت است.

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.



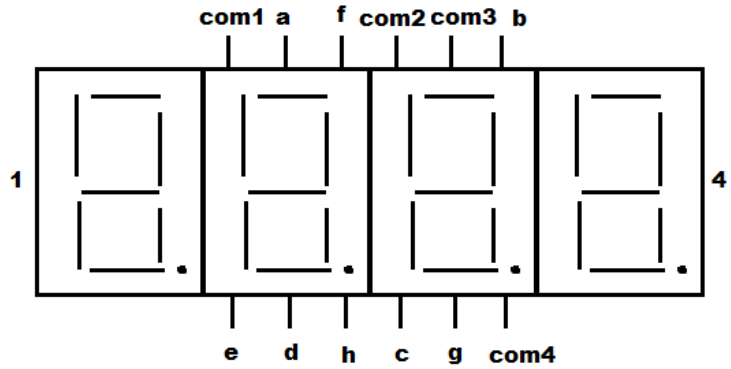
-در گوشه دیگری از بردبورد یک رگولاتور ۷۸۰۵ جا می زنیم.
-سیمی که ولتاژ مثبت آداپتور را حمل می کند به پایه input وصل می کنیم.
-سیمی که ولتاژ منفی آداپتور را حمل می کند به پایه Ground وصل می کنیم.
-پایه های output و Goround رگولاتور را به ردیف های مثبت و منفی کنار بردبورد وصل می کنیم تا آی سی ها از این خطوط تغذیه شوند.



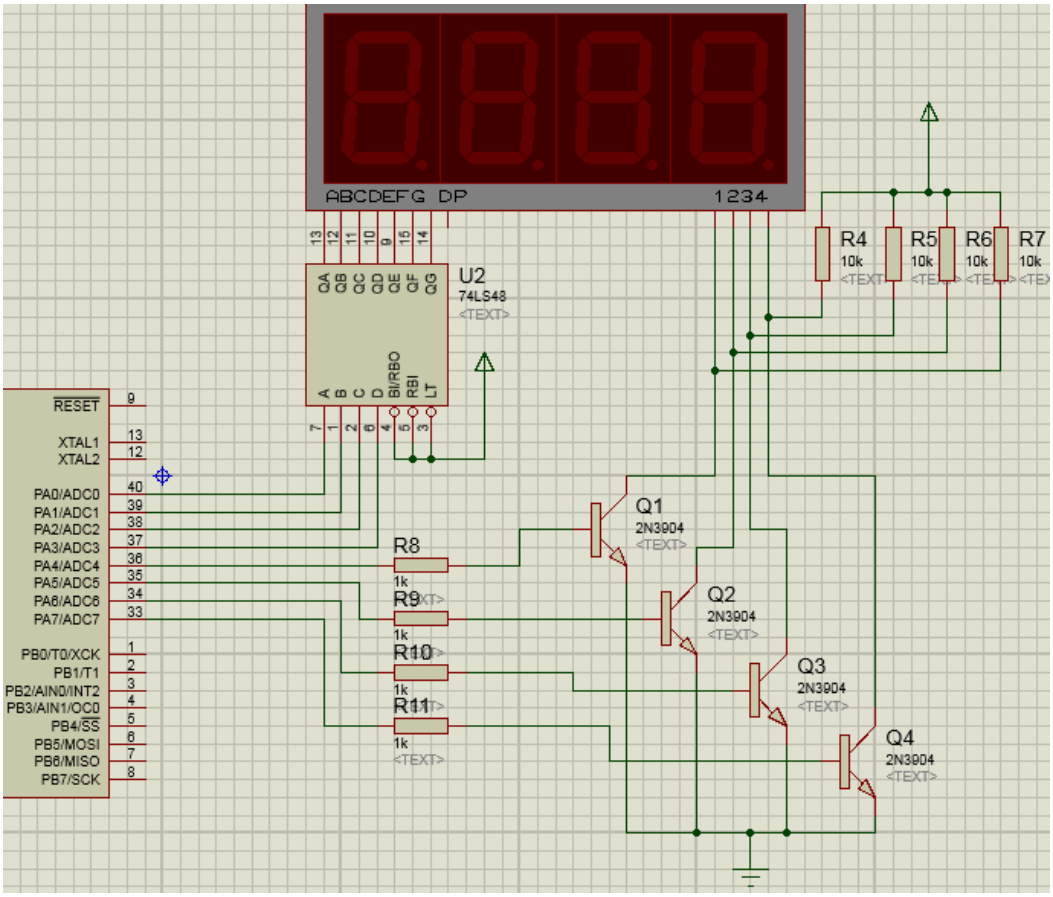
۶-۱۷-۲- سون سگمنت های مالتی پلکس

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 و...) 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.

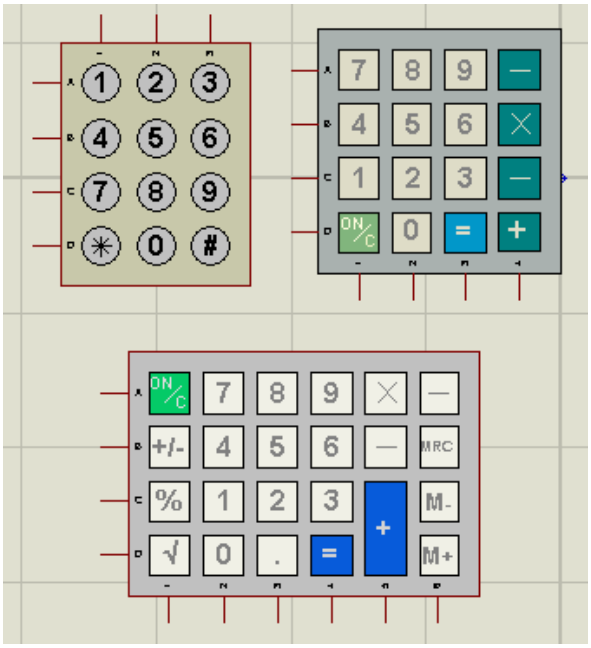
```
void display(void)
{
  PORTA=data[0];
  PORTA.4=1;
  delay_ms(5);
  PORTA.4=0;

  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...

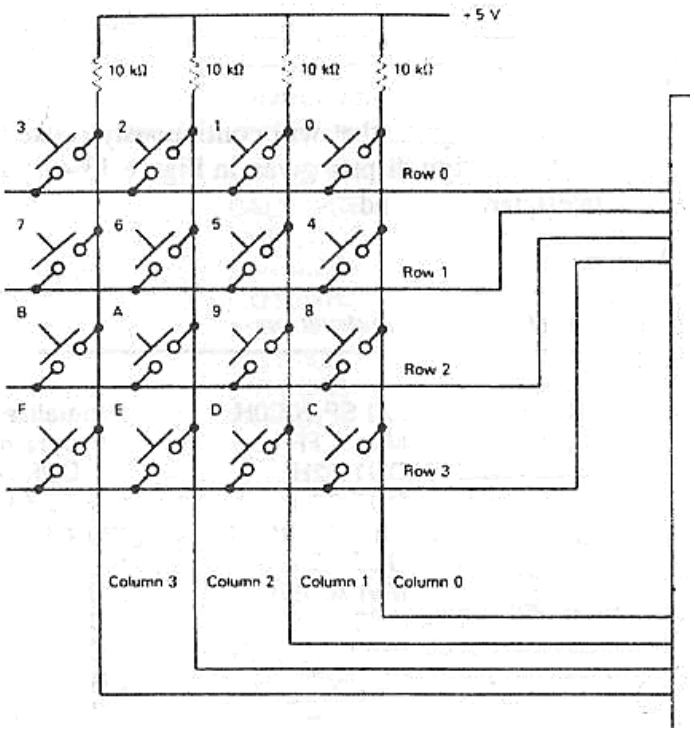
۶-۱۸- اتصال صفحه کلید به میکرو

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 proteus که در بازار نیز موجود است ، مشاهده می کنید.



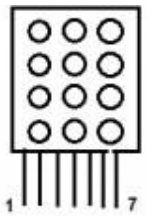
همانطور که در شکل زیر ساختار داخلی یک صفحه کلید ۴*۴ نشان داده شده است ، برای خواندن صفحه کلید ابتدا همه ستون ها را به صورت شکل زیر توسط مقاومت به تغذیه مثبت وصل کرده و از همه سطرها و ستون ها مستقیم به میکرو وصل می کنیم .سپس سطرها را به عنوان خروجی و ستون ها را به عنوان ورودی

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.

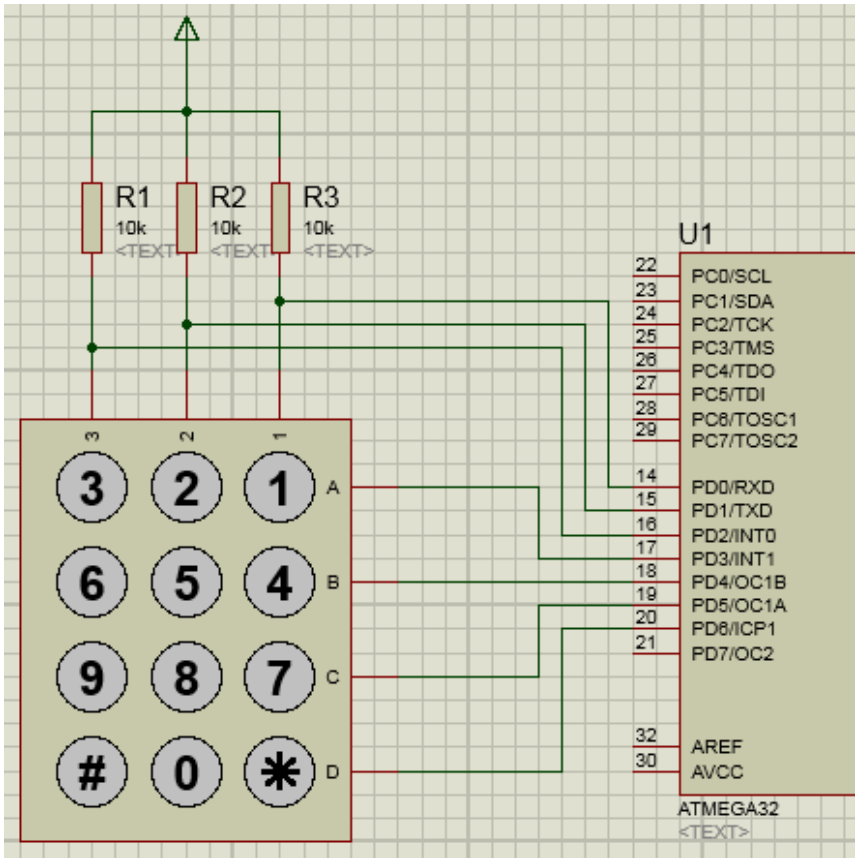
مدار داخلی صفحه کلیدهای دیگر نیز دقیقا به صورت فوق هستند . در شکل زیر نحوه چیدمان پایه های صفحه کلید ۴ در ۳ را مشاهده می کنید.



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 pullupو 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;
```

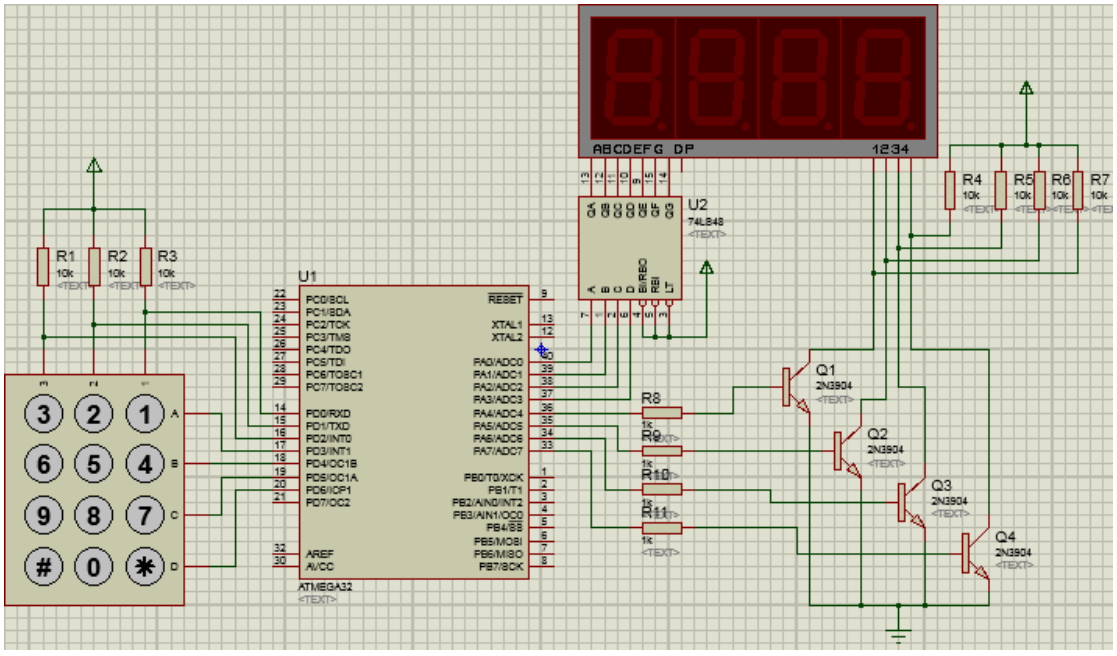
```
//---- ROW3 ----
PORTD.5=0;
delay_ms(2);
if(PIND.0==0) key=7;
if(PIND.1==0) key=8;
if(PIND.2==0) key=9;
PORTD.5=1;
//---- ROW4 ----
PORTD.6=0;
delay_ms(2);
if(PIND.1==0) key=0;
PORTD.6=1;
}
```

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 keyboardدر 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 keyبرابر 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 :

مرحله اول : طراحی سخت افزار



مرحله دوم : طراحی نرم افزار

```
#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;
   }
```

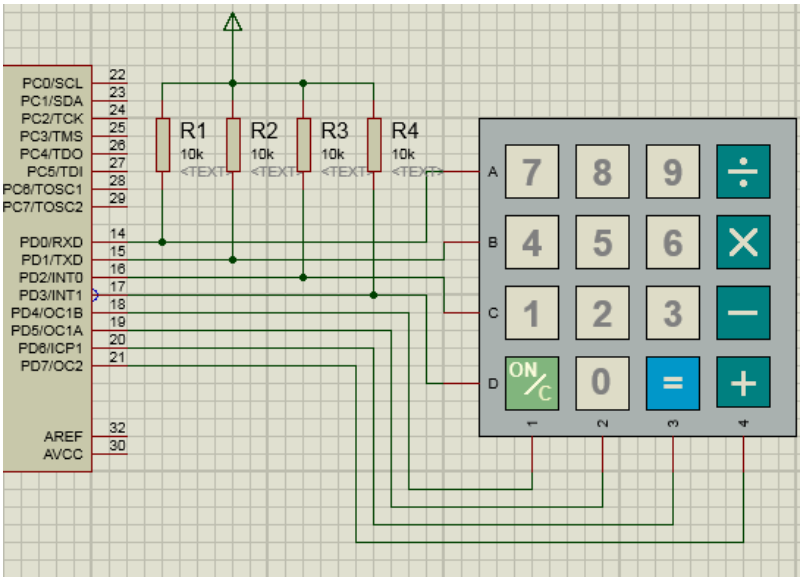
```
}
for(j=0;j<10;j++) display();
}
}
```

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 displayو 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 key ریخته 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.

دانلود مثال عملی شماره ۳

۶-۱۹-اتصال صفحه کلید ۴ در ۴ به میکرو

فرض میکنیم صفحه کلیدی با ۴ سطر و ۴ ستون را به صورت شکل زیر به میکرو متصل کرده باشیم.



در این صورت برای خواندن از صفحه کلید به روش سطری/ستونی که برای صفحه کلید ۳ در ۴ نیز گفته شد ، دقیقا به همان صورت عمل می کنیم. بنابراین برنامه به صورت زیر در می آید.

```
void keyboard(void)
{
 PORTD=0xf0;
 //---- ROW1 ----
```

```
 PORTD.4=0;
 delay_ms(2);
 if(PIND.0==0) PORTA=7;
 if(PIND.1==0) PORTA=4;
 if(PIND.2==0) PORTA=1;
 if(PIND.3==0) PORTA=10;
 PORTD.4=1;
 //---- ROW2 ----
 PORTD.5=0;
 delay_ms(2);
 if(PIND.0==0) PORTA=8;
 if(PIND.1==0) PORTA=5;
 if(PIND.2==0) PORTA=2;
 if(PIND.3==0) PORTA=0;
 PORTD.5=1;
 //---- ROW3 ----
 PORTD.6=0;
 delay_ms(2);
 if(PIND.0==0) PORTA=9;
 if(PIND.1==0) PORTA=6;
 if(PIND.2==0) PORTA=3;
 if(PIND.3==0) PORTA=11;
 PORTD.6=1;
 //---- ROW4 ----
 PORTD.7=0;
 delay_ms(2);
 if(PIND.0==0) PORTA=12;
 if(PIND.1==0) PORTA=13;
 if(PIND.2==0) PORTA=14;
 if(PIND.3==0) PORTA=15;
 PORTD.7=1;
 }
```

تمرین : برنامه فوق را با برنامه صفحه کلید ۳ در ۴ مقایسه کنید. مشاهده می شود در هر سطر برای خواندن ستون ها از ۴ دستور if اسفاده شده است.

نتیجه : برای خواندن صفحه کلید به صورت ستونی ، به تعداد ستون ها از دستور شرطی if استفاده می شود .

۶-۱۹-۱- برنامه حرفه ای تر اتصال صفحه کلید ۴ در ۴ به میکرو

روش خواندن از صفحه کلید در برنامه قبلی به صورت سطری/ستونی بود. برای حرفه ای تر شدن برنامه یک آرایه select تعریف می کنیم که بتوان از آن آرایه در حلقه for استفاده کرد و اندیس حلقه for را به جای آرگومان آرایه استفاده کرد. بنابراین با فرض اتصال یک صفحه کلید ۴ در ۴ به همان صورت شکل قبلی به میکروکنترلر ، برنامه زیر را داریم:

```
char keypad(void)
{
```

```
char code[16]={'7','8','9','/','4','5','6','*','1','2','3','-','c','0','=','+'};
unsigned char select[4]={0x0E,0x0C,0x0B,0x07};
unsigned char i,j;
   for(i=0;i<4;i++)
   {
   PORTD=select[i];
   delay_ms(2);
   if((PIND & 0xF0)!=0xF0)
   for(j=0;j<4;j++)
   {
   if((PIND & 1<<(4+j)) != 1<<(4+j))
   return code[i*4+j];
   }
   }
return 20;
PORTD=0x0F;
}
```

توضیح : عملکرد این تابع نیز دقیقا به همان صورت گفته شده در مثال عملی شماره ۳ است با این تفاوت که یک آرایه به نام code برای اصلاح اعداد صفحه کلید در نظر گرفته شده است. در صورتی که این آرایه نباشد با زدن مثلا کلید ۷ عدد ۰ نمایش داده می شود ! بنابراین به جای اینکه i*4+j به خروجی فرستاده شود code[i*j+j] به خروجی فرستاده می شود تا بدین وسیله اعداد اصلاح شود .

نکته : آرایه code را میتوان به صورت char یا unsigned char تعریف کرد. زمانی که در پروژه از 7segment استفاده می شود باید آرایه code را به صورت unsigned char تعریف کرد و در صورت استفاده از LCD باید آرایه code را به صورت char تعریف کرد. البته در صورت استفاده از سون سگمنت نمیتوان کاراکترهایی مثل ضرب ، جمع و ... را روی آن نمایش داد.

تذکر : به تفاوت های میان برنامه نوشته شده برای صفحه کلید ۳ در ۴ با صفحه کلید ۴ در ۴ توجه کنید.

فصل ۷– آموزش کدویزارد AVR

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.

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

- **معرفی** عملکرد ، ویژگی ها و وظایف واحد مورد نظر.
- **تشریح** نحوه عملکرد مداری واحد مورد نظر.
- معرفی و بررسی **رجیسترها**ی 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.

۷–۱– واحد پورت های ورودی/خروجی

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:

۱. 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.

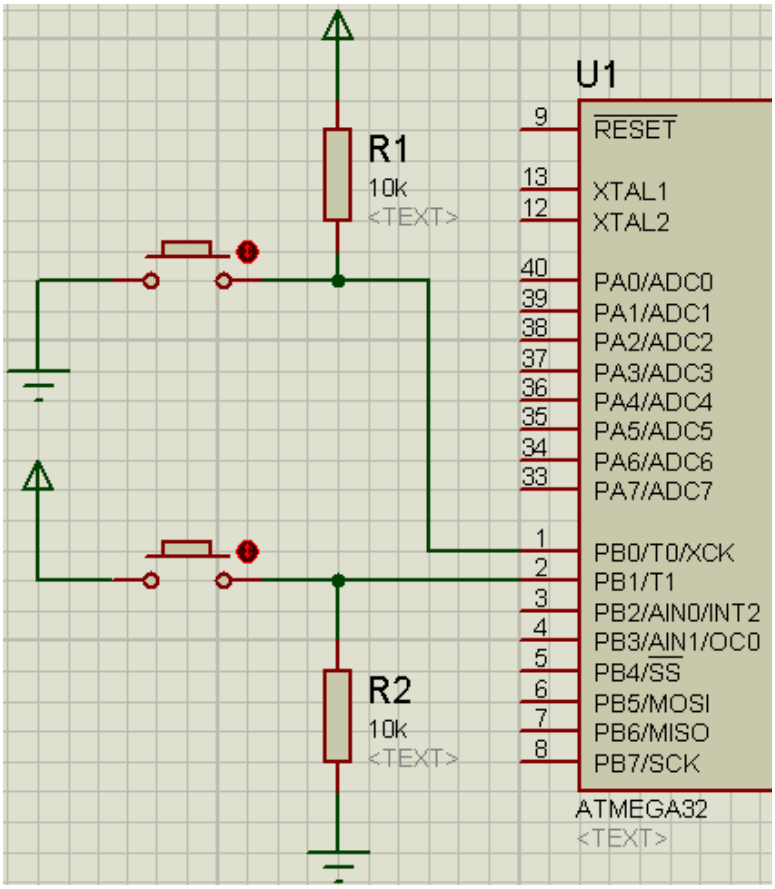
۲. در صورت استفاده از یک پایه به عنوان **خروجی** ، پایه مورد نظر می تواند دو حالت (منطق ۱ یا منطق ۰) داشته باشد. در صورتی که آن پایه منطق ۱ داشته باشد ولتاژ آن پایه حداکثر Vcc می گردد و جریانی از داخل میکرو به بیرون کشیده می شود که به آن جریان Source گویند .زمانی که پایه منطق ۰ داشته باشد ، ولتاژ آن پایه حداقل ۰ ولت می گرد و جریانی از بیرون به آن وارد می شود که به آن جریان sink گویند.

۳. 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 مقدار Vccمی کند . هر چه جریان دهی پایه بیشتر باشد افت ولتاژ پایه از مقدار Vcc نیز بیشتر می شود . همانطور

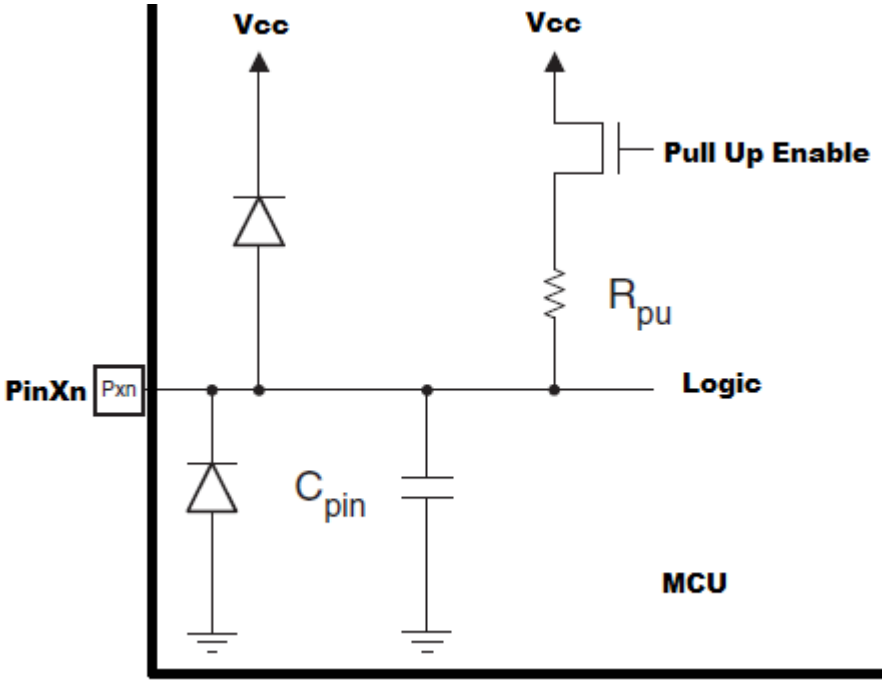
که در شکل صفحه ۳۰۸ دیتاشیت قسمتTypical Characteristics نیز برای Vcc=5vو دما ۲۵ درجه مشاهده می شود ، با افزایش جریان دهی با کاهش ولتاژ از ۵ ولت تا ۳ ولت مواجه هستیم.
برای جریان Source هم به همین صورت اگر جریانی که به پایه آی سی وارد می شود افزایش یابد ولتاژ آن از حالت ایده آل (۰ ولت) افزایش پیدا کرده به طوری که برای Vcc=5v و دمای ۲۵ درجه به ۲ ولت نیز می رسد.

۴. 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.

۷-۱-۱- رجیسترهای واحد I/O

در ATmega32چهار 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 PORT و نیز بیت PUD در رجیستر SFIOR نشان می دهد . لازم به توضیح است که رجیستر SFIOR یک 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)

۷-۱-۲- 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.n در 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:

۱. 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.
۲. رجیستر DDRX.n مربوطه ۰ باشد یعنی پایه ورودی باشد . پس در حالت خروجی مقاومت پول آپ نمیتواند فعال شود.
۳. رجیستر PORTX.n مربوطه ۱ باشد . در صورت ۰ بودن نیز مقاومت پول آپ غیر فعال است.

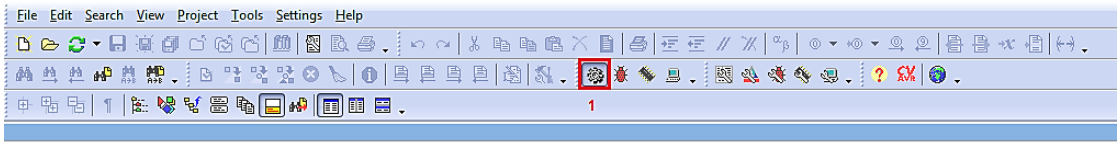
۷-۲- CodeWizardچیست ؟

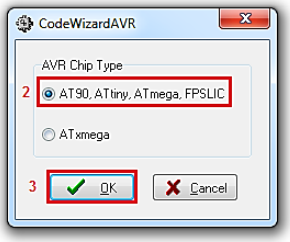
در حقیقت CodeWizard ابزار کاملی برای برنامه نویسی کلیه میکروکنترلرهای AVR است که به همراه نرم افزار CodeVision ارائه شده و بخشی از این نرم افزار می باشد . این ابزار که به جادوگر کد معروف است به شما این امکان را می دهد که با انتخاب کردن مدل میکروکنترلر ، فرکانس کاری و تنظیم و پیکره بندی کلیه واحدهای میکروکنترلر (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.

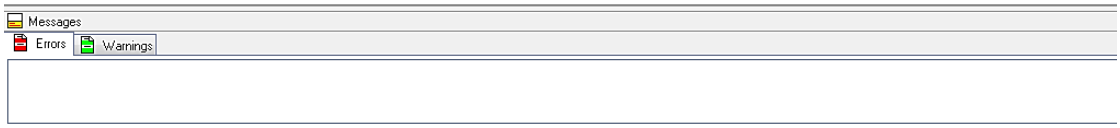
۷-۲-۱- شروع کار با ابزار CodeWizard

۱. 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.
۲. برای راه اندازی کدویزارد در صفحه اصلی نرم افزار CodeVision از منوی tools ، گزینه CodeWizard AVRرا انتخاب کنید. همچنین می توانید به جای آن از منوی ابزار بالای نرم افزار گزینه چرخ دنده را کلیک کنید (شکل زیر) . در پنجره باز شده میخواهد که نوع چیپ را از نظر

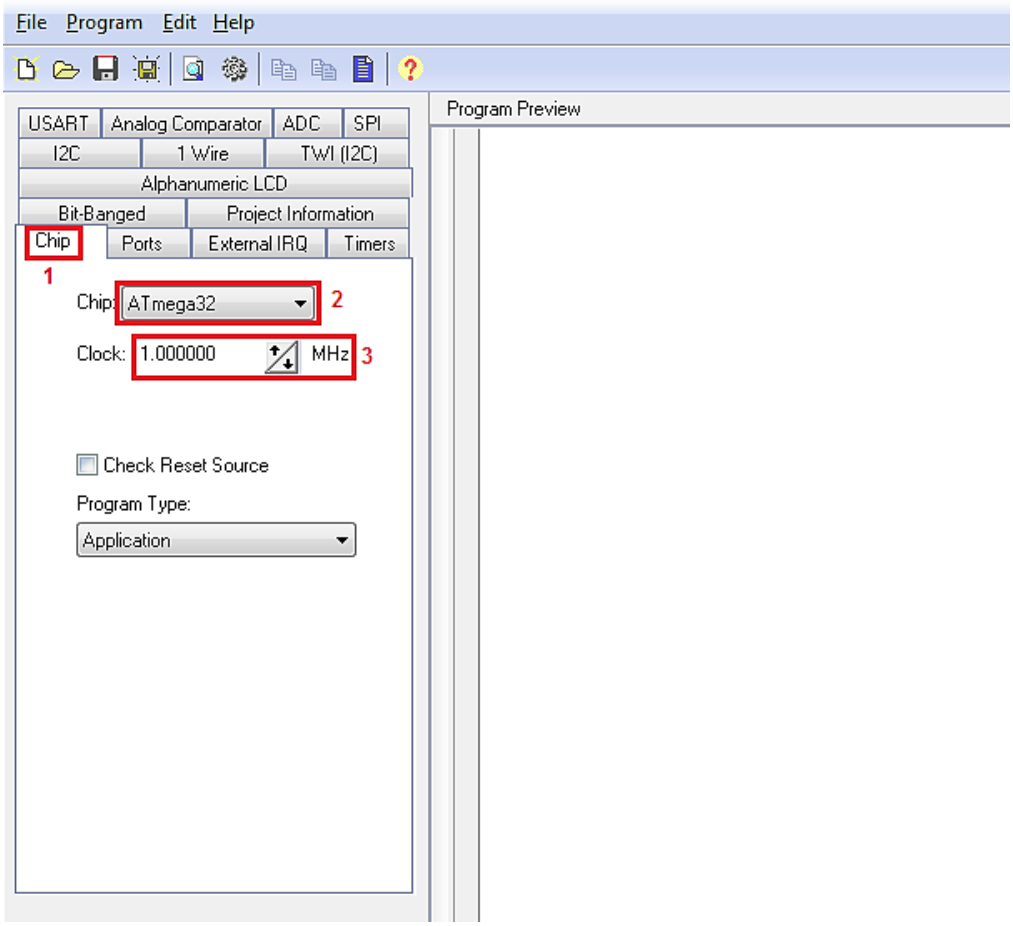
سری ساخت مشخص کنید. چون چیپ مورد استفاده ما atmega32 است ، گزینه اول atmega را انتخاب کرده و ok کنید . اکنون پنجره codeWizardAVR جلوی شماست.



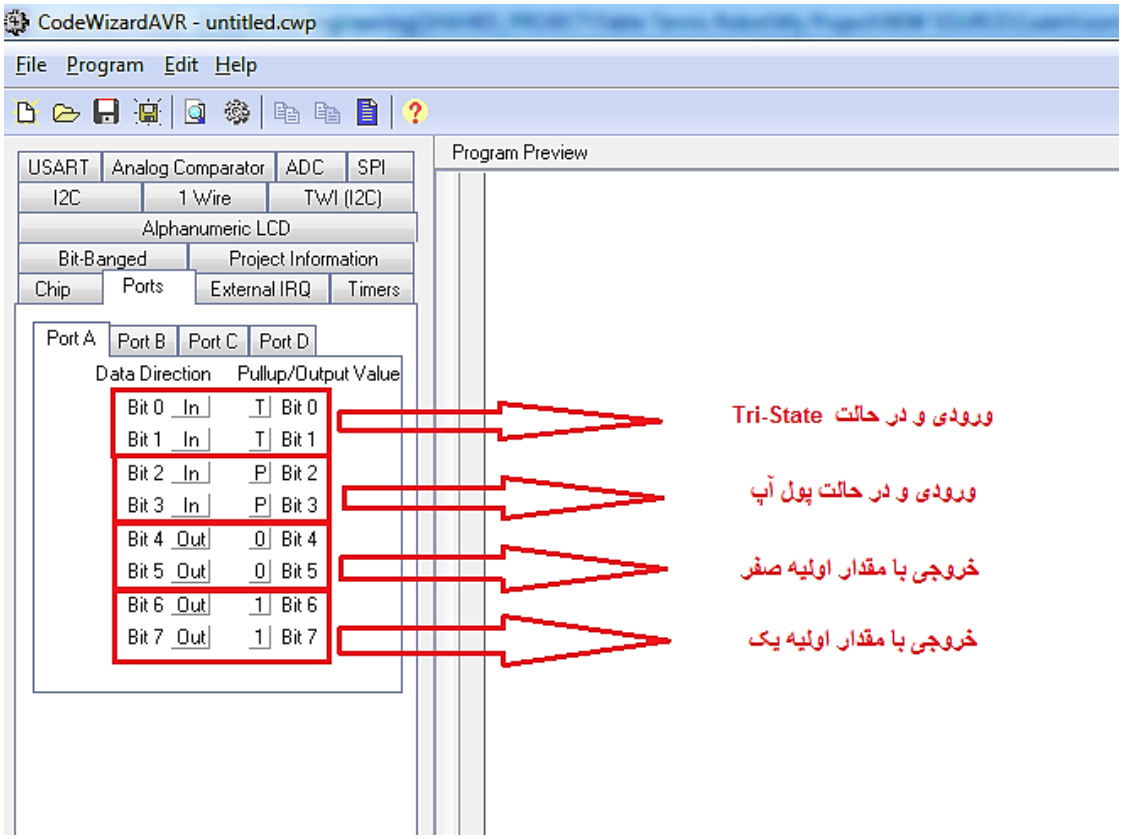




۳. در این قسمت اولین کاری که باید انجام دهید تنظیم نوع چیپ و فرکانس کاری میکروکنترلر است . دیگر گزینه ها را کاری نداریم . فرکانس کاری میکروکنترلر باید همان فرکانسی انتخاب شود که در قسمت Fuse Bits (فیوز بیت ها) تنظیم شده است. در صورتی که میکروکنترلر Atmega32 را تازه خریداری کرده اید ، فرکانس پیش فرض آن 1Mhz می باشد.



۴. 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-Stateبه 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.



۵. 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 codeWizardAVRنام 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.

۷-۳- راه اندازی LCD های کاراکتری

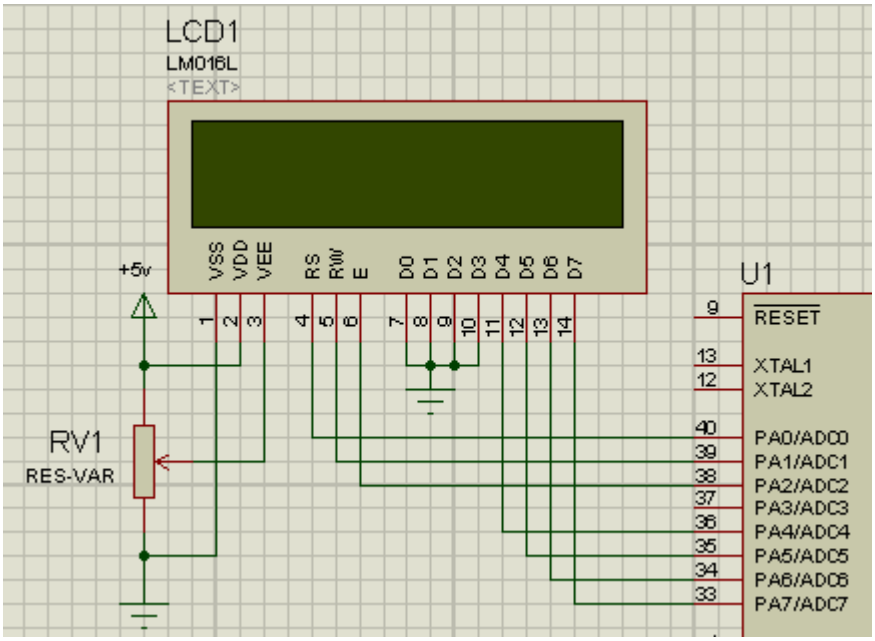
صفحه نمایش کاراکتری یکی از پرکاربرد ترین وسایل خروجی است که به میکرو وصل می شود و میتوان کاراکترهای قابل چاپ را روی آن نمایش داد. مشخصه اصلی LCD های کاراکتری تعداد سطر و ستون آنها است برای مثال LCD های ۲*۱۶ دارای ۲ سطر و ۱۶ ستون می باشند و در مجموع ۳۲ کاراکتر را میتوان با آنها نشان داد.

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.

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)

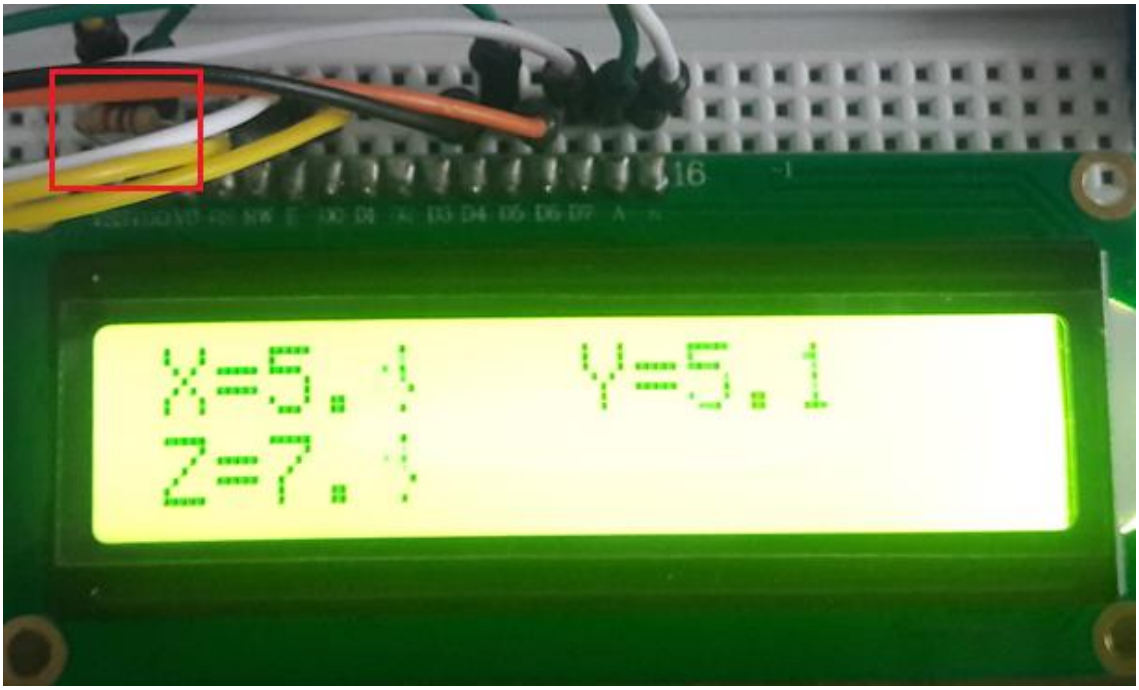
LCD ها را میتوان با ۸ خط دیتا یا ۴ خط دیتا راه اندازی کرد اما نرم افزار CodeVision تنها از ۴ خط دیتا پشتیبانی می کند که در آن تنها پایه های RS ، R/W، E و D4 تا D7 به پورت دلخواه از میکرو متصل می شود. تنها تفاوت راه اندازی ۴ خط دیتا در این است که داده های ۸ بیتی LCD به جای یکبار ، دو بار از طریق ۴ بیت 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: 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.

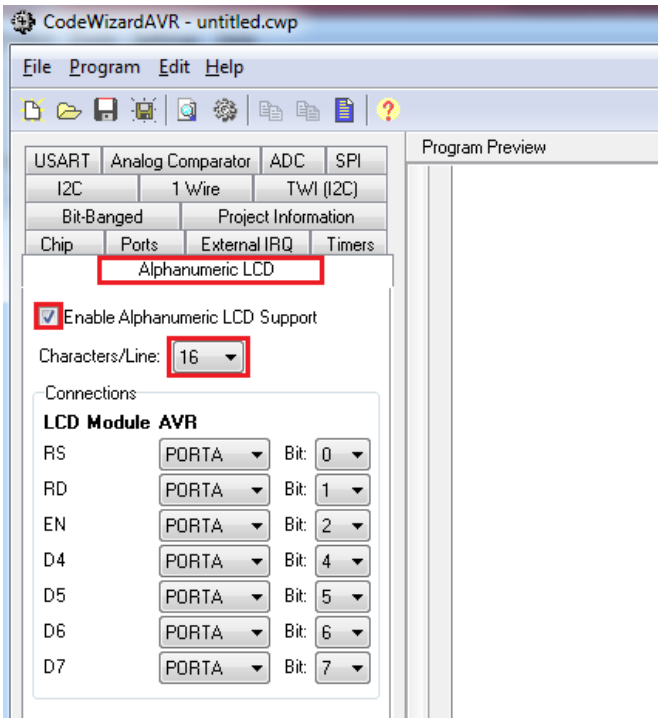
نکته ۱ : پایه سوم (VEE) برای تنظیم کنتراست نوشته های روی LCD می باشد. با وصل کردن این پایه به یک مقاومت متغیر میتوان کیفیت نمایش را تنظیم کرد.

نکته ۲ : در عمل میتوان به جای استفاده از مقاومت متغیر ، از اتصال پایه VEE توسط یک مقاومت 1K با زمین مدار ، حداکثر کنتراست را برای LCD داشت. درنتیجه کافی است یک مقاومت 1K بین پایه 3 و 1 قرار دهیم. شکل زیر این موضوع را نشان می دهد.



۷-۳-۱- تنظیمات LCD کاراکتری در کدویزارد

در سربرگ Alphanumeric LCD ابتدا تیک Enable را بزنید و سپس در قسمت Character/lineباید تعداد ستون ها را وارد نمایید و در قسمت بعدی باید نام پایه های متصل به LCD را مشخص کنید. با این کار 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 LCDدر برنامه دلخواه می توان استفاده کرد.



۷-۳-۲- توابع کار با LCD کاراکتری

۱. پاک کردن تمام LCD :

```
lcd_clear();
```

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

۲. 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.

۳. چاپ یک کاراکتر:

```
lcd_putchar(‘ کاراکتر ‘);
```

۴. چاپ یک رشته:

```
lcd_putsf(“ رشته “);
```

۵. چاپ یک متغیر رشته ای:

```
lcd_puts(نام متغیر رشته ای );
```

نکته۱ : ورودی تابع lcd_putsf یک رشته ثابت مانند "IRAN" است اما ورودی تابع lcd_putsیک متغیر رشته ای از نوع آرایه ای می تواند باشد .

نکته ۲ : 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 اضافه کرده 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 این مثال میخواهیم 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 مینویسیم.

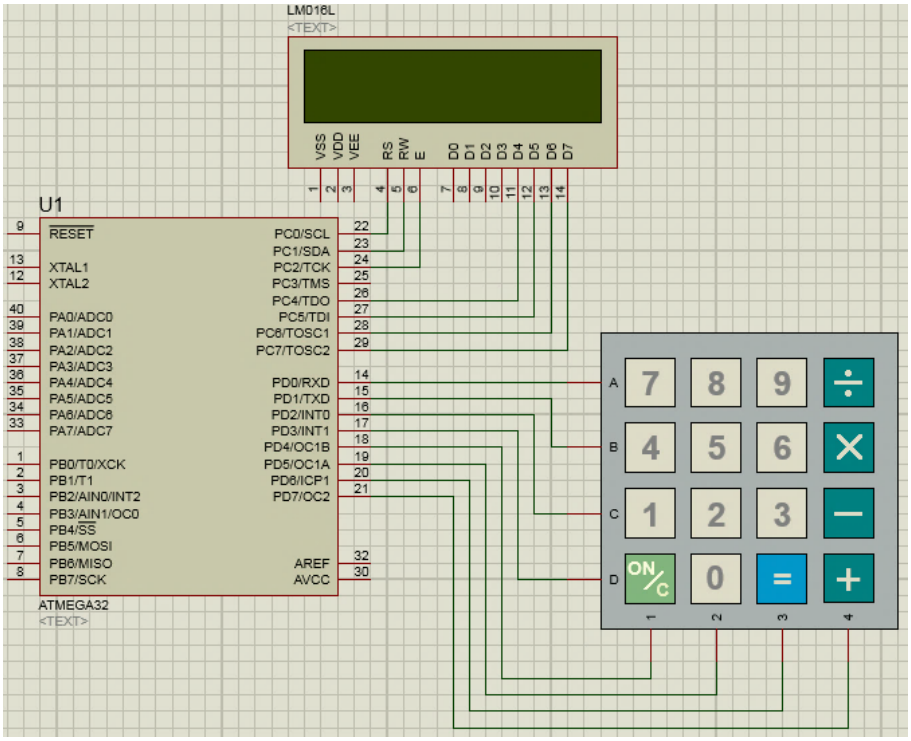
فرمت متغیرهای کاراکتری ارسالی:

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

مثال عملی شماره ۴ : با استفاده از اتصال یک LCD کاراکتری ۱۶ در ۲ و یک صفحه کلید ۴ در ۴ به میکروکنترلرAtmega32 ، 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.

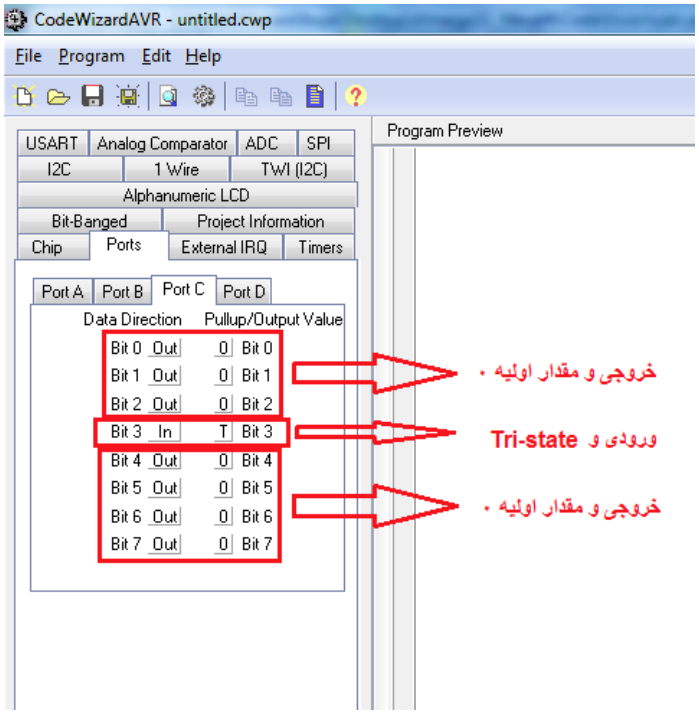
حل:

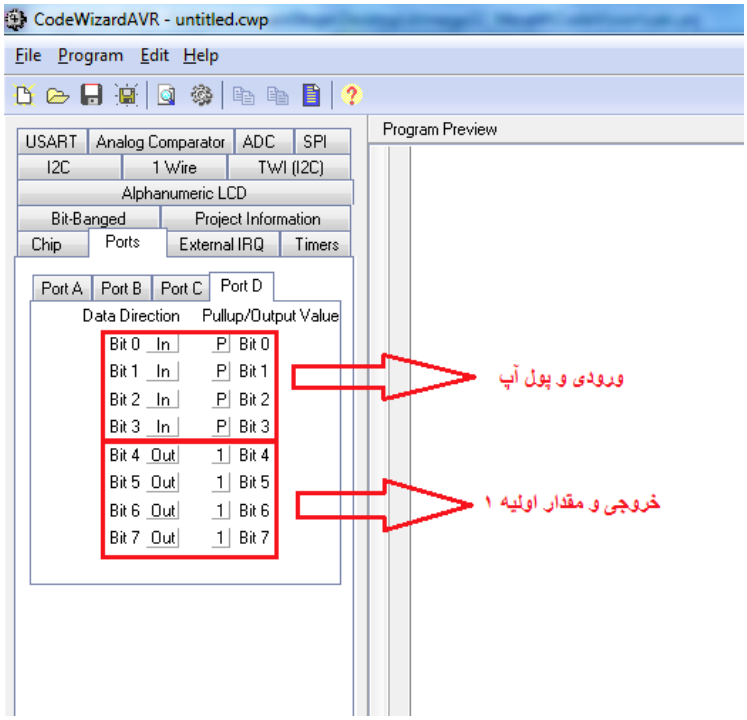
مرحله اول : طراحی سخت افزار در پروتئوس



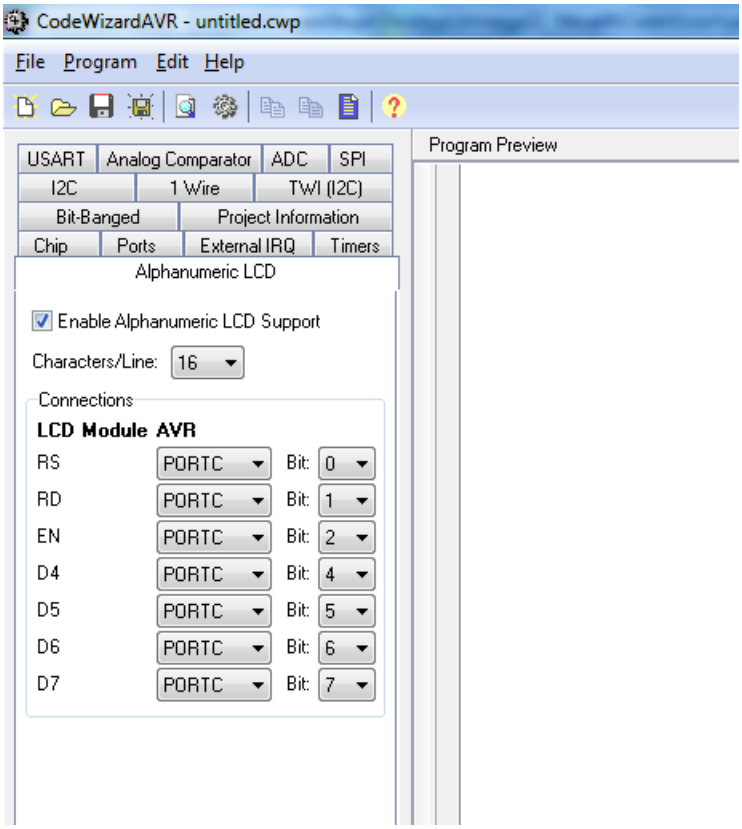
مرحله دوم : انجام تنظیمات در کدویزارد

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.





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 : تکمیل برنامه

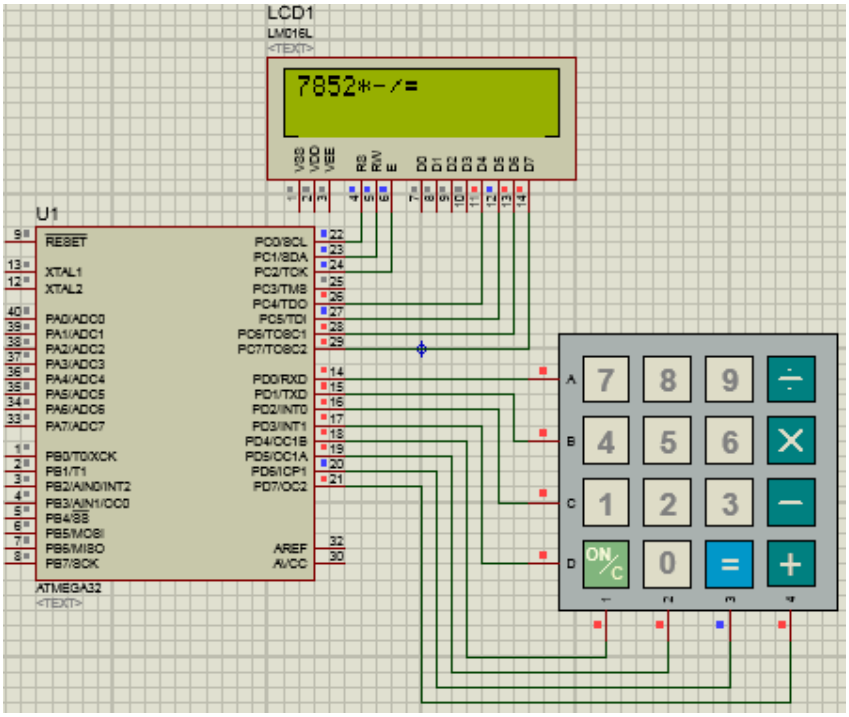
در این مرحله تابع خواندن از صفحه کلید ۴ در ۴ را با تغییراتی جزئی اضافه کرده و برنامه را به صورت زیر کامل می کنیم:

```
#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.



نکته : در هنگام پیاده سازی دقت داشته باشید که پورت C تنها زمانی در دسترس است که فیوزبیت JTAGEN غیرفعال باشد.

دانلود مثال عملی شماره ۴

۷-۴- 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 را طوری برنامه نویسی می کنیم که دائما واحد جانبی مورد نظر را بررسی کند و با آن ارتباط برقرار 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.

۷-۴-۱- 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 به دیجیتال تبدیل کرده و کارش تمام شد ، وقفه را فعال می کند. با فعال شدن وقفه ، CPU پردازش کنونی خود را رها کرده و برنامه ای را که در تابع سابروتین وقفه نوشتیم انجام می دهد. بعد از پایان تابع سابروتین وقفه 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.

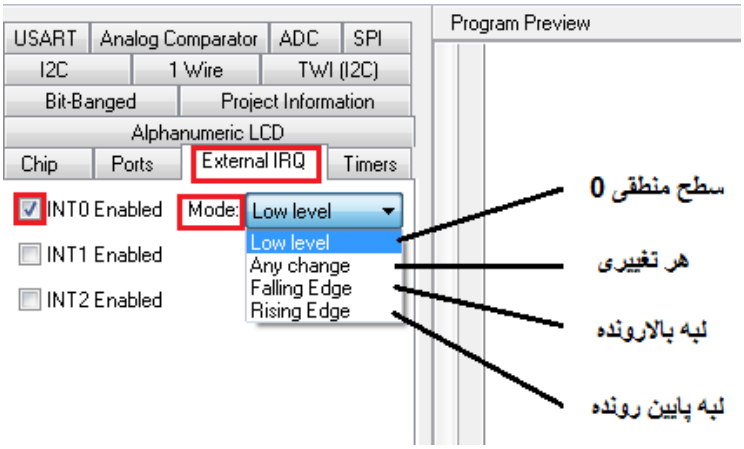
نکته : در میکروکنترلر Atmega32 تعداد ۱۸ منبع وقفه داخلی و ۳ منبع وقفه خارجی وجود دارد.

۷-۴-۲- راه اندازی واحد وقفه خارجی در Atmega32

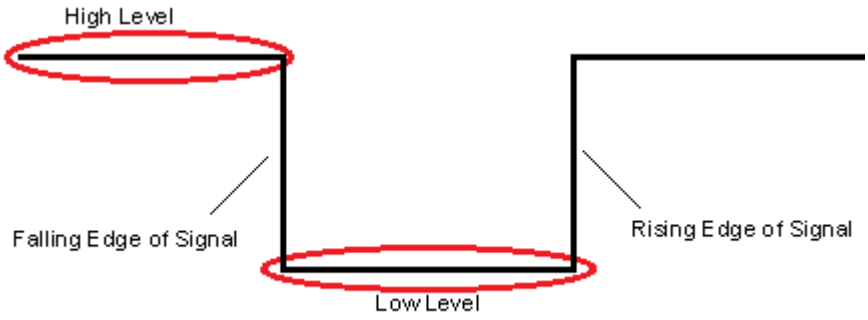
در میکروکنترلر Atmega32 سه وقفه خارجی به نامهای INT0 (پایه ۱۶) ، INT1 (پایه ۱۷) و INT2 (پایه ۳) وجود دارد.

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 رخ دهد:

- وقفه در لبه بالا رونده پالس ورودی رخ دهد.
- وقفه در لبه پایین رونده پالس ورودی رخ دهد.
- وقفه در سطح منطقی ۰ رخ دهد.
- وقفه در هر تغییر ۰ به ۱ یا بالعکس رخ دهد.



نکته : شکل لبه ها و سطوح پالسی را در یک سیگنال دیجیتال نمونه ، نشان می دهد.



تذکر : وقتی از کلید و مقاومت پول آپ شده روی پایه INTx استفاده می شود بهتر است وقفه در لبه پایین رونده رخ دهد.

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. برنامه ای که میخواهیم در زمان وقفه اجرا شود را در تابع سابروتین وقفه قرار می دهیم. تابع سابروتین وقفه فقط زمانی که وقفه خارجی روی پایه ی مورد نظر رخ دهد ، انجام می شود. به طوری که 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 استقاده 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

.

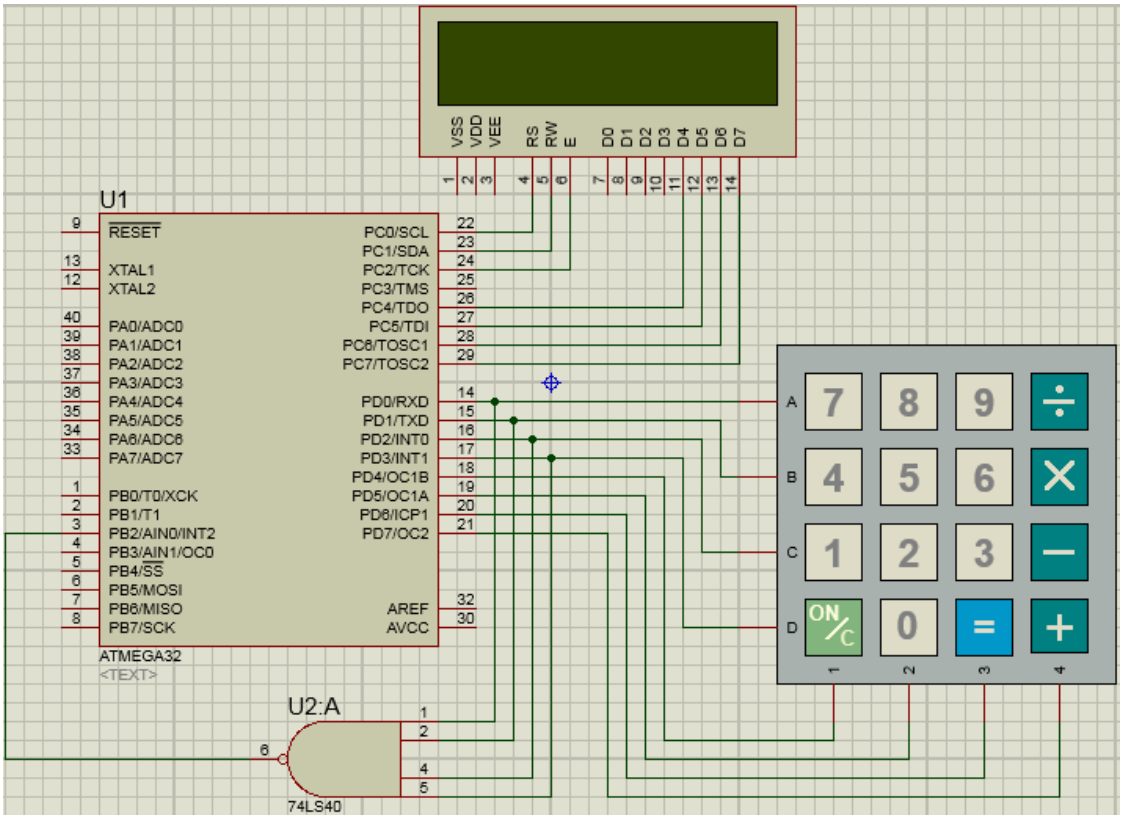
```
#asm("sei")  //set enable interrupt

#asm("cli")  //clear interrupt
```

مثال عملی شماره ۵ : یک LCD کاراکتری 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.

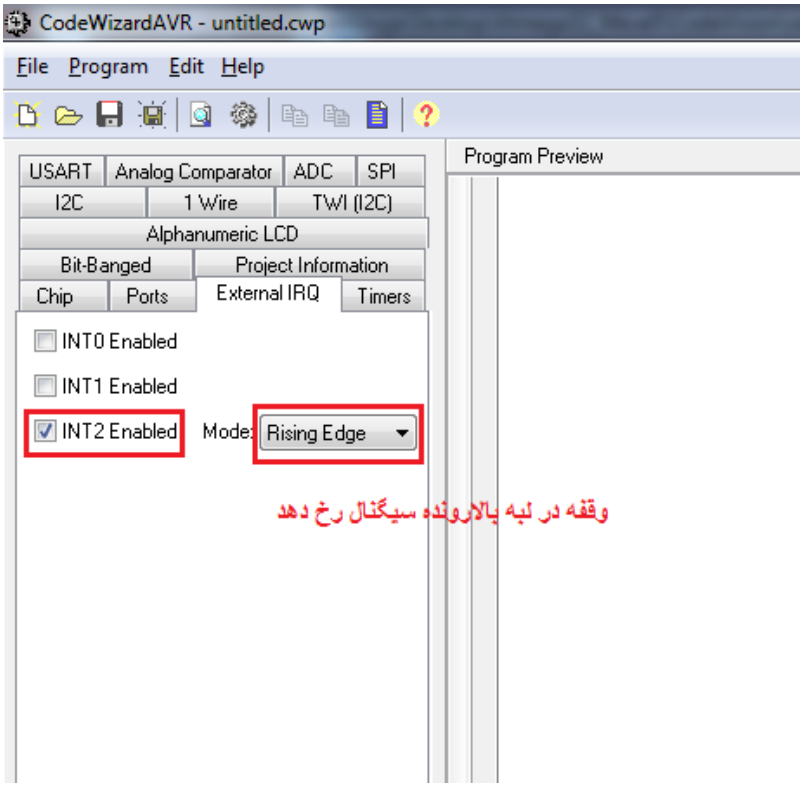
حل :

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 : تکمیل برنامه

بعد از تولید کدهای اولیه توسط کدویزارد برنامه را به صورت زیر تکمیل می کنیم :

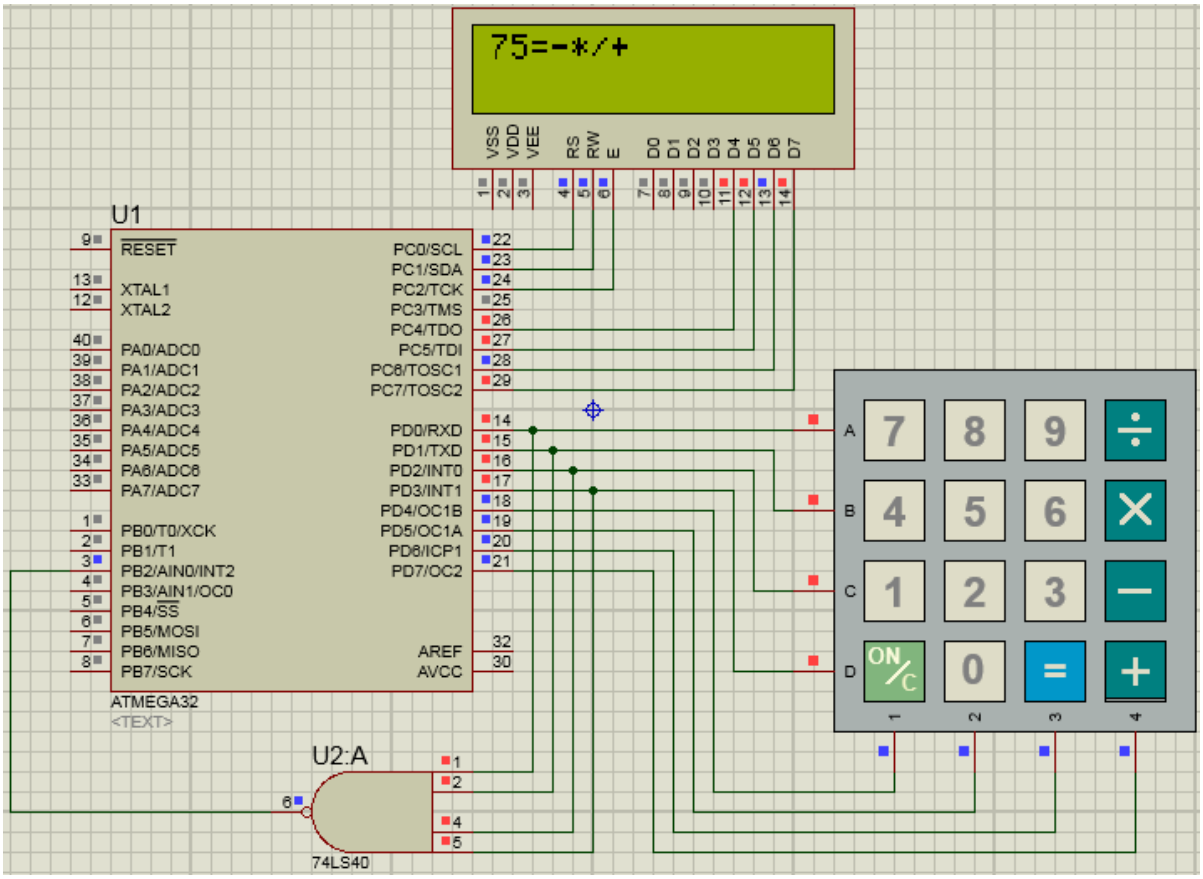
```
#include <mega32.h>
#include <delay.h>
#include <stdio.h>
#include <alcd.h>

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

void keyboard(void)
{
  //---- 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
      }
    }
}
```

تمرین: این برنامه را با برنامه مثال قبل مقایسه کنید.

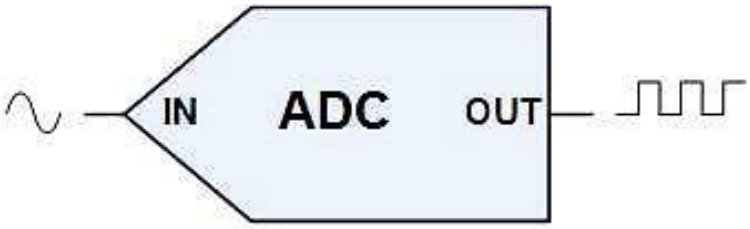
مرحله چهارم : شبیه سازی



[دانلود مثال عملی شماره ۵]

۷–۵– واحد مبدل آنالوگ به دیجیتال ADC

واحد مبدل سیگنال‌های آنالوگ به دیجیتال (Analog to Digital Converter=ADC)، مداری الکترونیکی است که سیگنال‌های پیوسته آنالوگ را به داده‌های گسسته دیجیتالی تبدیل می‌کند. همانطور که میدانید تمامی کمیت های فیزیکی ، آنالوگ هستند. کمیت های آنالوگ برای پردازش توسط میکروکنترلر ابتدا می بایست تبدیل به دیجیتال (۰ و ۱) 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:

۱. **ولتاژ مرجع (reference Voltage)** : که حداکثر ولتاژ قابل نمونه برداری را مشخص می کند.
۲. **رزولوشن یا دقت نمونه برداری (Resolution)** : تعداد بیت های خروجی دیجیتال به توان دو ، دقت نمونه برداری را مشخص می کند.
۳. **نرخ نمونه برداری (Sample Rate)** : یعنی فرکانس نمونه برداری که تعداد نمونه برداری از ولتاژ آنالوگ در واحد زمان است.
۴. **تعداد کانال ها (channels)** : تعداد ولتاژهای آنالوگی که مبدل میتواند به عنوان ورودی آنالوگ به صورت مجزا به دیجیتال تبدیل کند.
۵. **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).
۶. **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

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

۷–۵–۱– تنظیمات واحد ADC در AVR

ولتاژ مرجع 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.

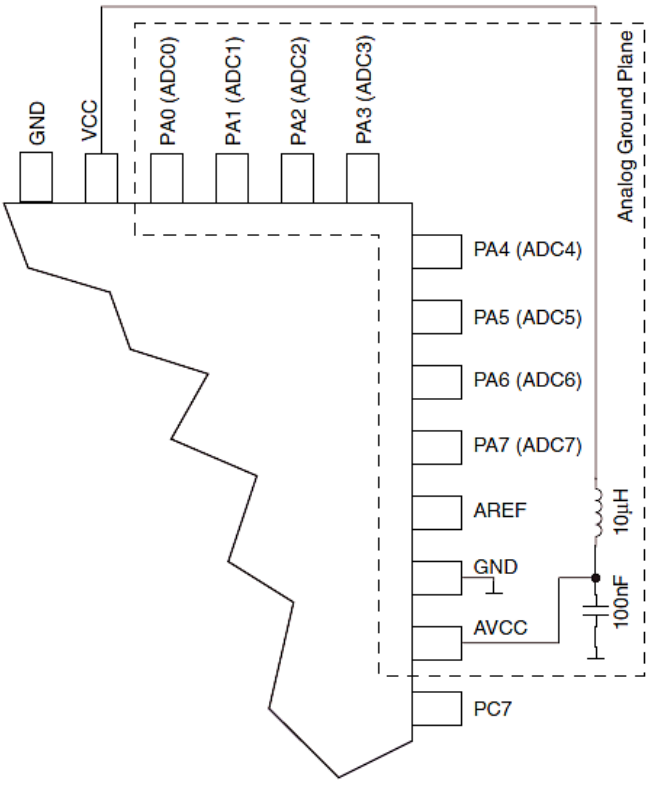
دقت نمونه برداری واحد ADC در AVR : دقت کلیه میکروکنترلرهای AVR در حالت ۸ بیتی یا ۱۰ بیتی قابل تنظیم می باشد.

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

تعداد کانال ها و بهره واحد 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.

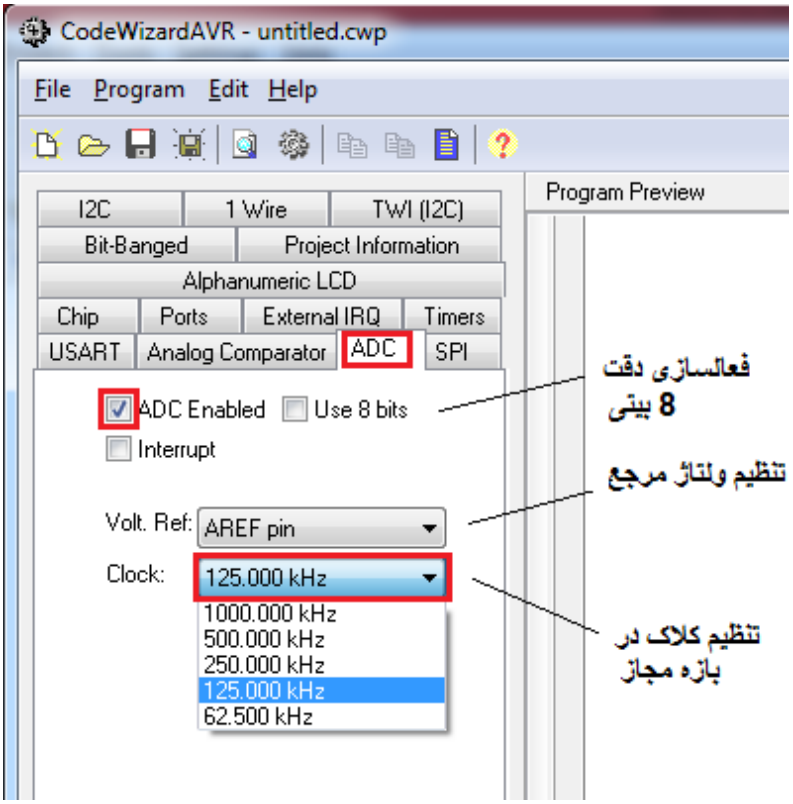
فرکانس کار واحد ADC در AVR : فرکانس پالس کلاک واحد مبدل ADC طبق پیشنهاد شرکت Atmel باید بین ۵۰ کیلوهرتز تا ۲۰۰ کیلو هرتز باشد (بازه مجاز). این فرکانس که تقسیمی قابل تنظیم از کلاک اصلی میکرو است ، بسته به نیاز کاربرد مورد نظر باید روی یک مقدار خاصی تنظیم شود.

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.

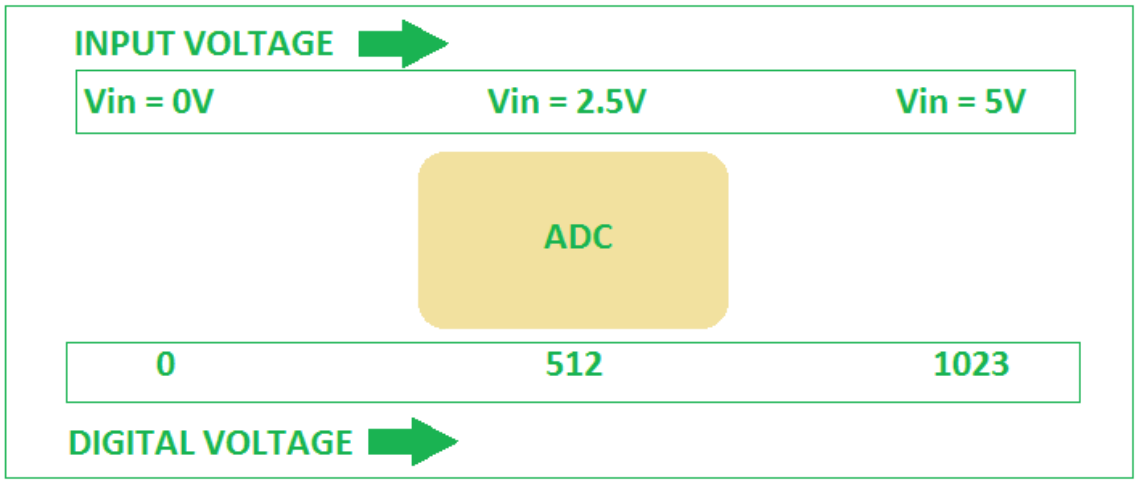


۷–۵–۲– تنظیمات کدویزارد برای راه اندازی واحدADC

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 ADC کم میشود. همچنین قابلیت ایجاد وقفه نیز با زدن تیک مربوطه فعال می شود (معمولا نیازی به فعالسازی آن نیست)



پس از تولید کد تابع 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 تابع 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.
برای مثال زمانی که ولتاژ مرجع را ۵ ولت در نظر بگیریم و از دقت ۱۰ بیتی استفاده نماییم، رابطه ورودی و خروجی، مشابه شکل زیر می شود. که در آن بازه مجاز ورودی آنالوگ بین ۰ تا ۵ ولت است و عدد دیجیتال در خروجی بین 0 تا 2^{10}-1=1023 متناسب با ورودی تغییر می کند.



اما برای استفاده از کد دیجیتال خروجی در برنامه، و فهمیدن مقدار ولتاژ 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.

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

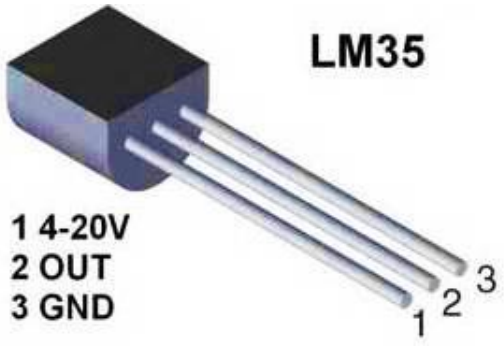
$$V_{in} = \frac{V_{ref}}{2^n - 1} \times read\_adc(channel)$$

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

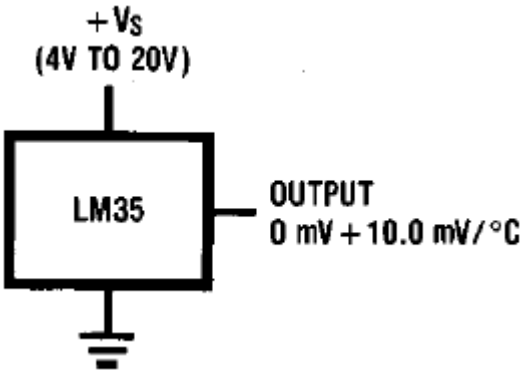
حل:

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

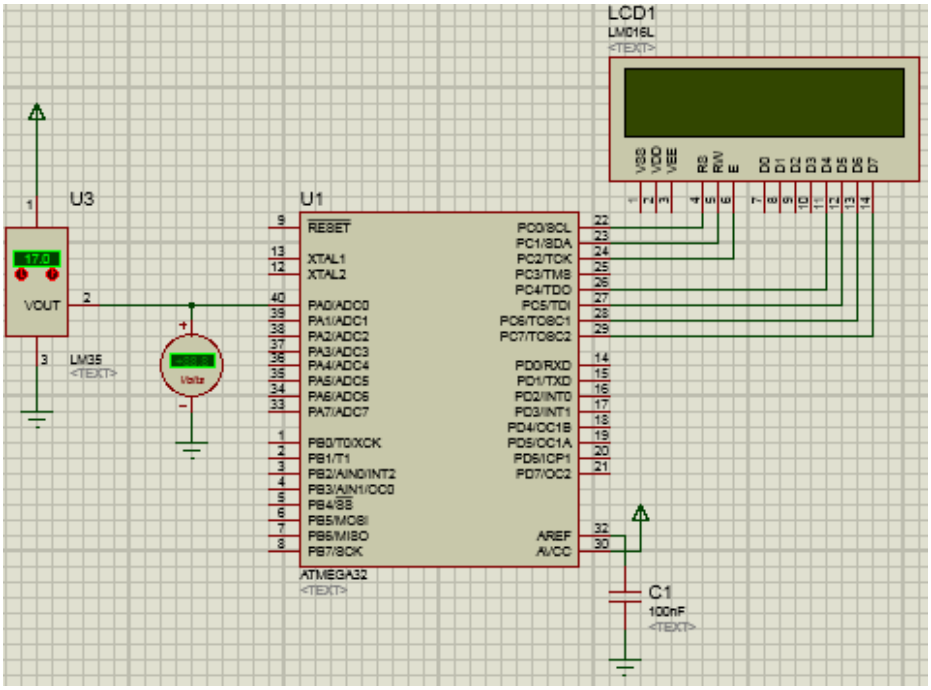
سنسور ۱۰ میلی ولت افزایش پیدا می کند. (برای مثال در دمای ۲۵ درجه اتاق می شود ۲۵۰ میلی ولت).
پایه های این سنسور به صورت شکل زیر است:



اگر به دیتاشیت این آی سی مراجعه نماییم، مدار های مختلفی برای کار در دماهای مختلف پیشنهاد کرده است. برای دماهای بین ۲ تا ۱۵۰ درجه میتوان از مدار شکل زیر استفاده کرد:

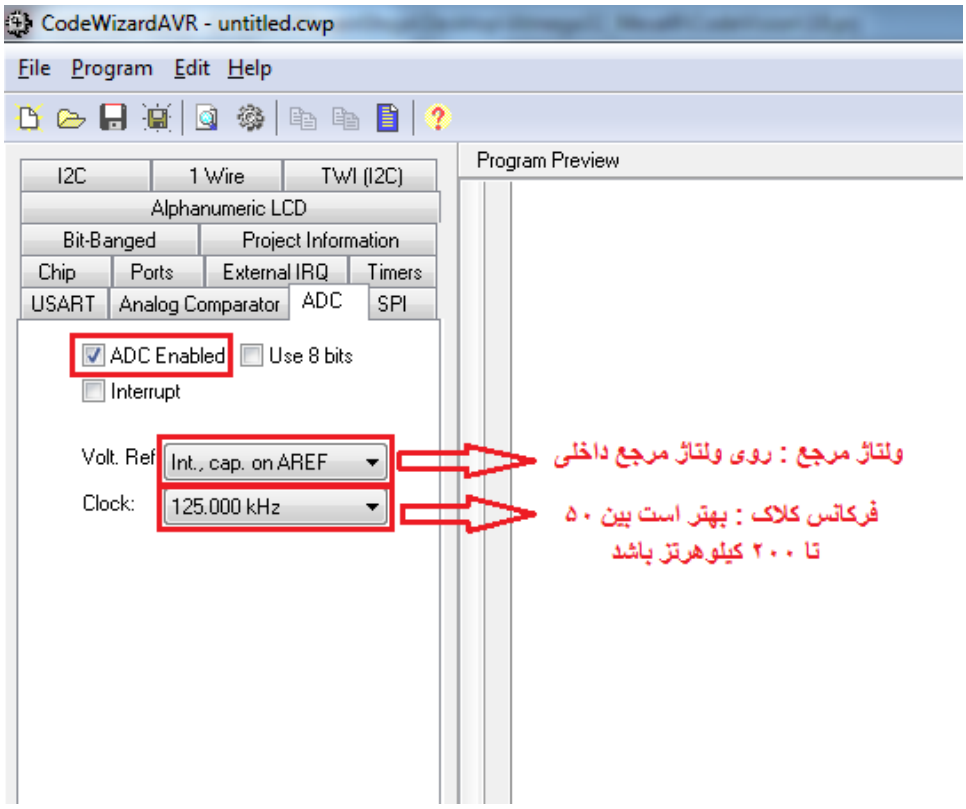


مرحله اول : طراحی سخت افزار در پروتئوس



مرحله دوم : تنظیمات کدویزارد

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



نکته : با قرار دادن ولتاژ مرجع واحد ADC روی ولتاژ داخلی ۲,۵۶ ولت، ضریب تفکیک نسبت به حالت ولتاژ مرجع ۵ ولت کوچکتر می شود و در نتیجه حساسیت دماسنج بهتر می شود.

مرحله سوم : تکمیل کدهای 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){
```

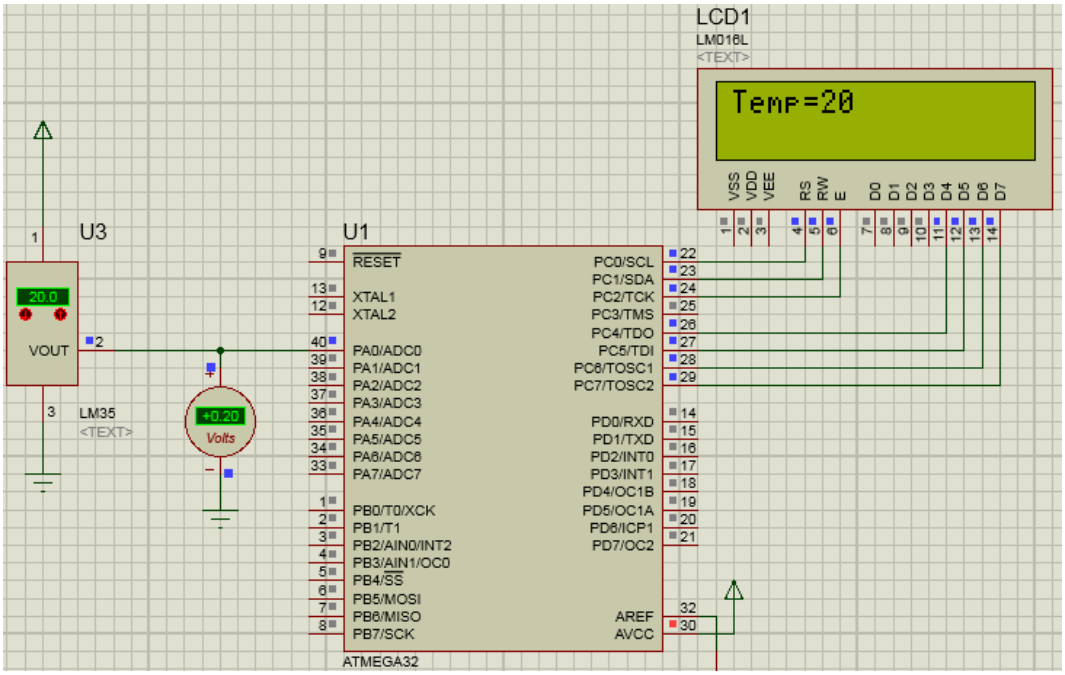
```
PORTC=0x00;
DDRC=0xF7;

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

.

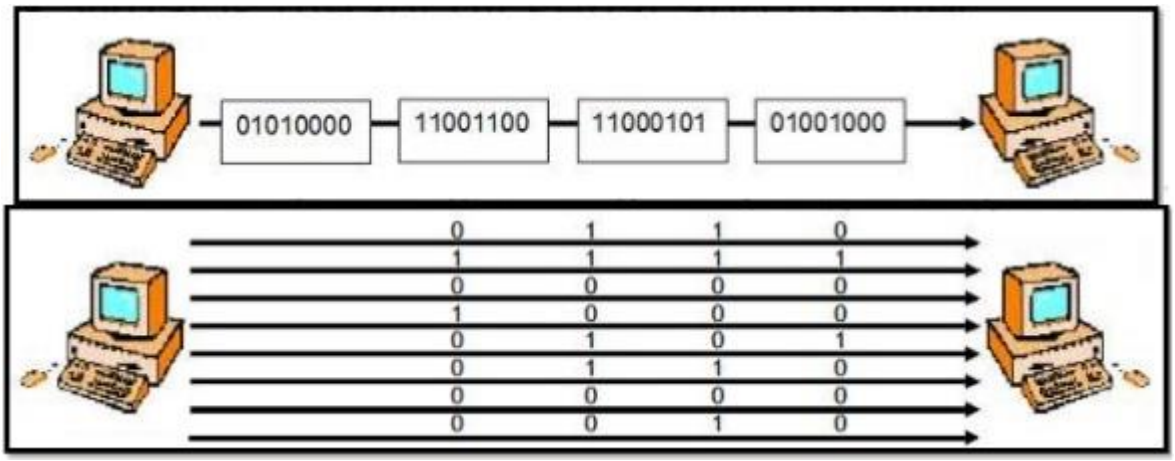


دانلود مثال عملی شماره۶

فصل ۸– راه اندازی ارتباطات سریال با کدویزارد

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.



۸–۱– 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

۸-۲- پروتکل های ارتباطی سریال و سرعت آنها

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-TRو ...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 توسط برخی از میکروکنترلرهای AVR پشتیبانی می شود.

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

۸-۳- نوع فرستنده و گیرنده

به فرستنده اطلاعات اصطلاحا Master و به گیرنده اطلاعات Slave گفته می شود Master . یا فرستنده اطلاعات را برای Slave یا 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 آموزش فرض میکنیم حداقل یکی از Master یا Slaveها میکروکنترلر AVR باشد.

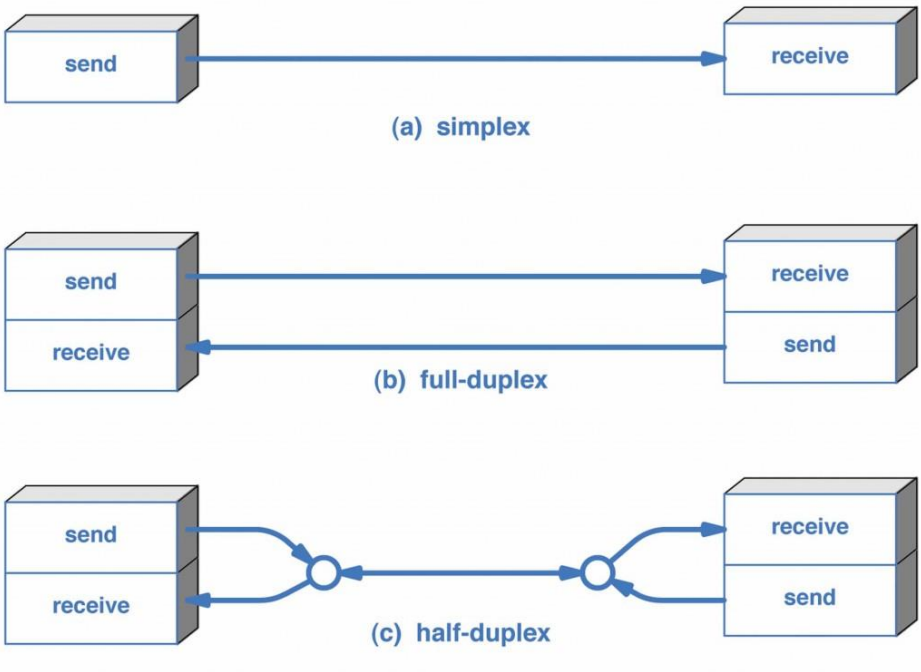
۸-۴- انواع حالت ارتباط سریال

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.

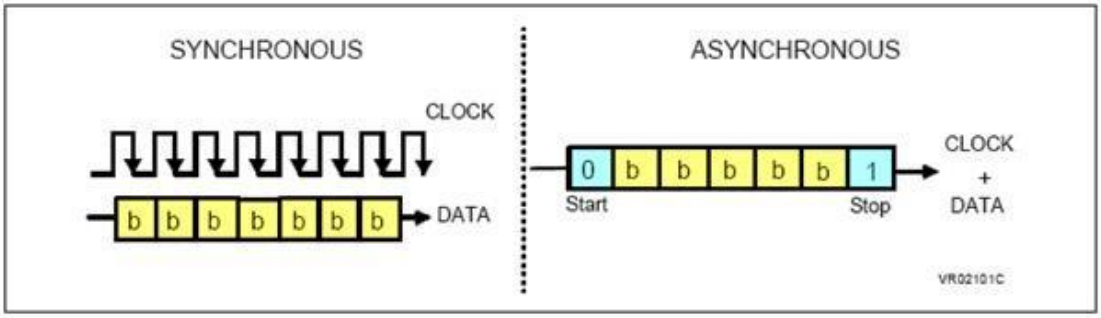


۸–۵– روش ارسال اطلاعات سریال

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.

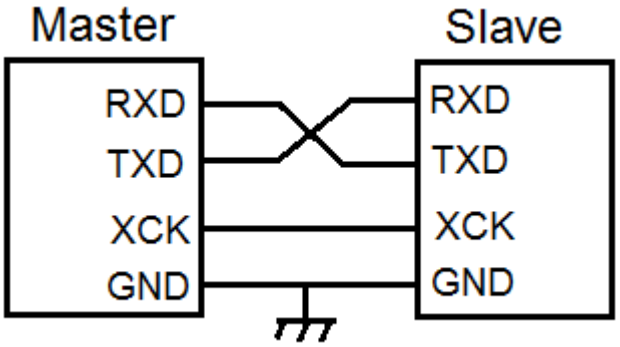
روش آسنکرون یا غیر همزمان : در این روش اطلاعات با یک سرعت استاندارد تنظیم شده و غیرقابل تغییر ارسال می شود . بنابراین دیگر پالس کلاک وجود ندارد و هر دو فرستنده و گیرنده میدانند که با چه سرعتی قرار است اطلاعات را ارسال یا دریافت کنند.



۸–۶– راه اندازی واحد USART

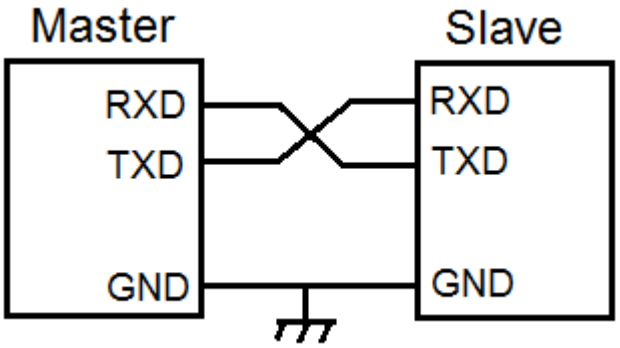
همواره یکی از مسائلی که با آن درگیر هستیم ارتباط دو یا چند دستگاه با هم است که بتوانند با یکدیگر دیتا ارسال کنند. روش های متنوعی برای این منظور وجود دارد که یکی از آنها USART می باشد. ارتباط سریال یوزارت مخفف عبارت **U**niversal **S**ynchronous And **A**synchronous Serial **R**eceiver And

Transmitterبه 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:



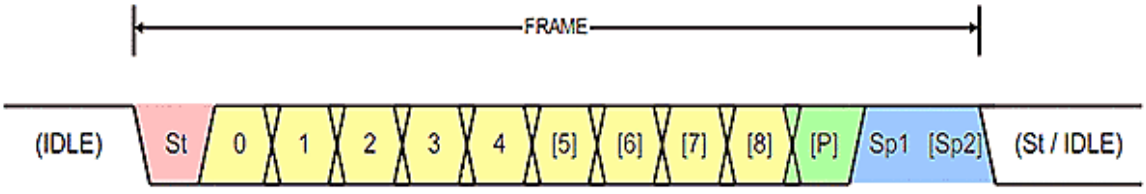
که در آن RXD برای دریافت دیتا ، TXDبرای ارسال دیتا ، XCK برای کلاک حالت سنکرون و GND زمین مشترک دو دستگاه می باشد. اما به دلیل اینکه در صنعت همواره سعی بر این است که تعداد سیم ها کم شود ، در اکثر اوقات از حالت آسنکرون استفاده می کنیم. در ارتباط آسنکرون سیم کلاک را حذف کرده و به جای آن پارامتری به نام 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.



۸-۶-۱- 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 rateعرض 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 bpsباشد 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

۸-۶-۲- پروتکل های استاندارد UART

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 اثرنویزهای 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 20Kbps بیت بر ثانیه 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 10Mbps است. در پروتکل RS485 تعداد گیرنده ها میتواند حداکثر تا ۳۲

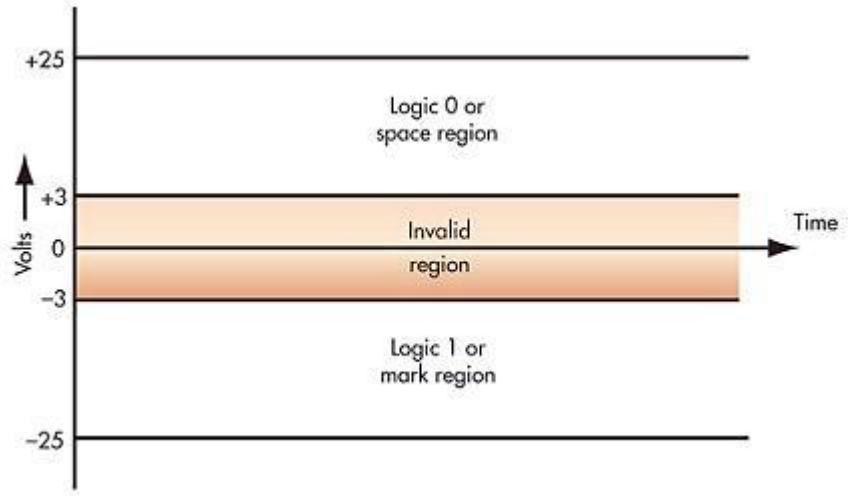
هم باشد ضمن اینکه بیشترین فاصله ۱۲۰۰ متر و بیشترین سرعت 10Mbps می باشد . شکل زیر مقایسه کلی بین این پروتکل ها را نشان می دهد.

به علت اینکه در این آموزش به فواصل و گیرنده های زیادی نیاز نداریم تنها به تشریح استاندارد RS232 کفایت می کنیم .

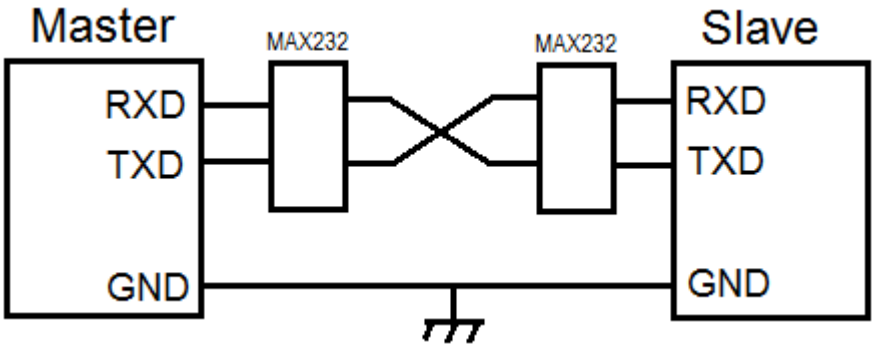
SPECIFICATIONS		RS232	RS423	RS422	RS485
Mode of Operation		SINGLE -ENDED	SINGLE -ENDED	DIFFERENTIAL	DIFFERENTIAL
Total Number of Drivers and Receivers on One Line (One driver active at a time for RS485 networks)		1 DRIVER 1 RECVR	1 DRIVER 10 RECVR	1 DRIVER 10 RECVR	32 DRIVER 32 RECVR
Maximum Cable Length		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

۸-۶-۳- استاندارد RS232

ساده ترین استاندارد برای ارتباط سریال در مسافت های بین ۱ تا ۱۵ 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 MAX232است . بنابراین در صورت استفاده از این استاندارد دو آی سی Max232 میان Master و Slave به صورت شکل زیر اضافه می شود.

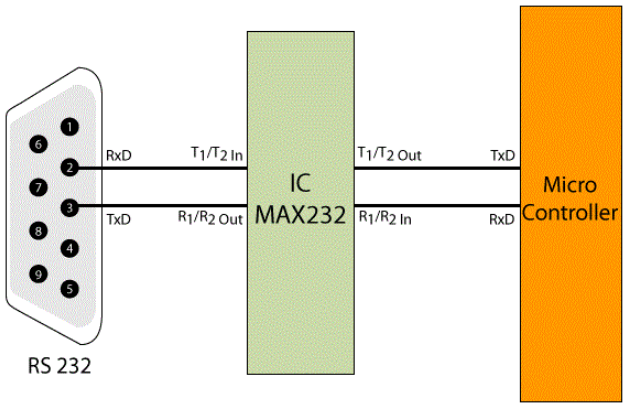


کابل مورد استفاده در پروتکل RS232 حداقل باید دارای ۳ رشته سیم باشند که یک سیم برای خط ارسال داده یا TX و یک سیم برای خط دریافت داده یا RX و سیم سوم نیز به عنوان سیم ولتاژ مرجع استفاده می شود. این کابل واسط می تواند از سیم های موازی ساده یا از سیم های جفت به هم تابیده شده باشند. طول کابل حداکثر نرخ انتقال 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

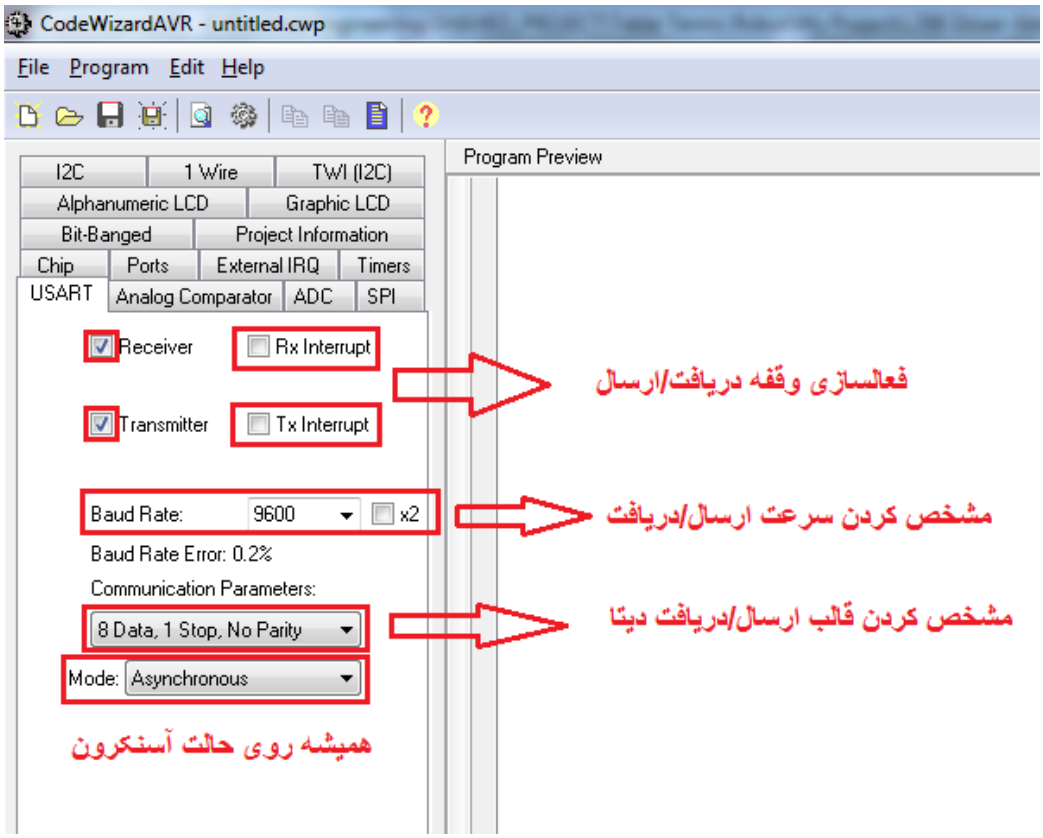
از استاندارد RS232 در پورت های سریال پشت کامپیوتر های PC (پورت DB9) استفاده شده است. بنابراین پورت سریالی که پشت PC ها وجود دارد دارای پروتکل RS232 است و با اضافه نمودن آی سی MAX232 به

صورت شکل زیر و از طریق نرم افزارهای ترمینال موجود در کامپیوتر می توان ارتباط میان میکروکنترلر و PC را برقرار نمود.



۸-۶-۴- تنظیمات واحد USART در کدویزارد

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.

۸-۶-۵- توابع پرکاربرد stdio.h در هنگام کار با واحد USART

۱- 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();

۲- 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 ‘);

۳- توابع puts و putsf :

برای ارسال یا دریافت یک رشته به صورت کامل به وسیله USART از دستورات putsf و putsاستفاده 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 “);nputs(متغیر رشته ای);

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

۴- 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 USRATگرفته و در متغیر رشته ای ذخیره می کند.

gets(متغیر رشته ای , طول رشته);

۵- 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.

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

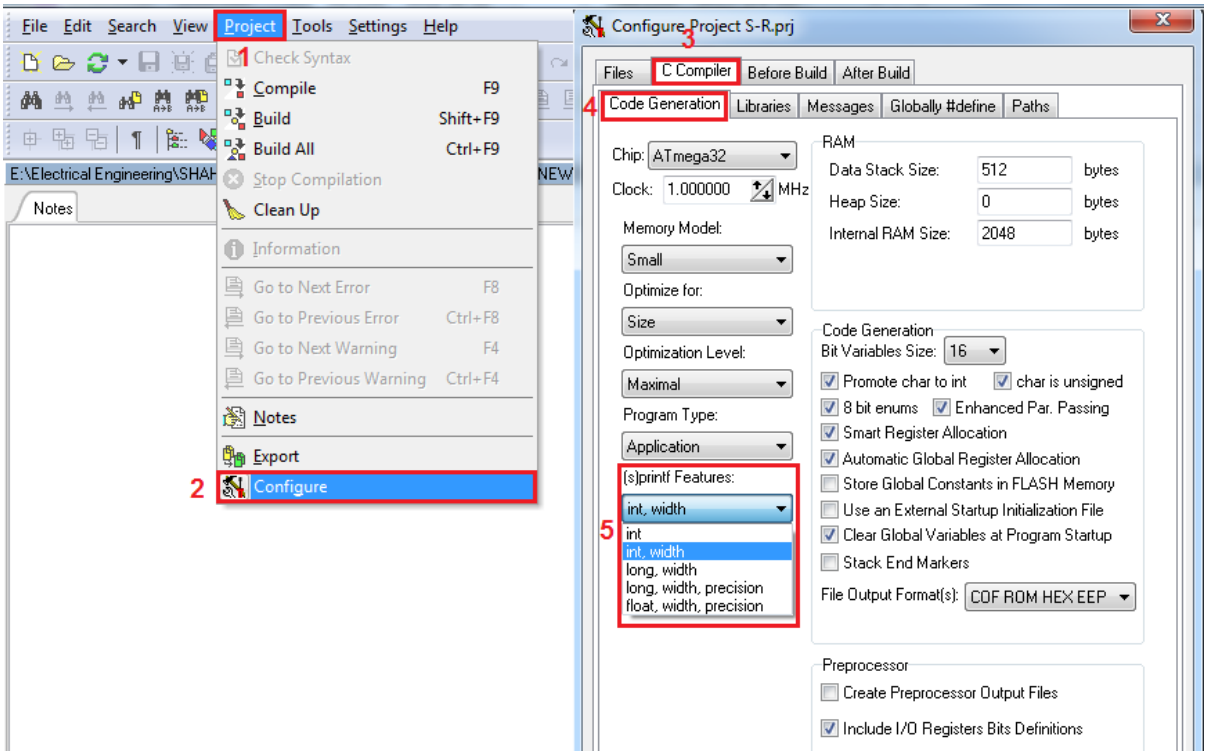
printf("A=%7.4f",i);

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 Compiler به زیر برگ Code Generation بروید . همانطور که در شکل زیر نیز مشاهده می کنید در قسمت printf feature می توان طول ، دقت و نوع تابع printf یا sprintf را در صورت وجود معین کرد.



۶- 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 int دریافت 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 feature میتوان نوع و طول ورودی را تنظیم کرد.

۷- 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;
char s[10];
sprintf(s,"Tha data is :%d",data);
```

۸- 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];
sscanf(s,"%s",a);
```

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

۸-۶-۶- 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 Bluetouthبه برخی از آنها نیاز خواهید داشت.

۱- تابع strcmp :

تابع strcmpکاراکترهای 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.

۲- 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");
```

۳- 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.

۴- تابع strlwr :

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

۵- تابع struper :

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

۶- تابع strlen :

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

۷- 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.

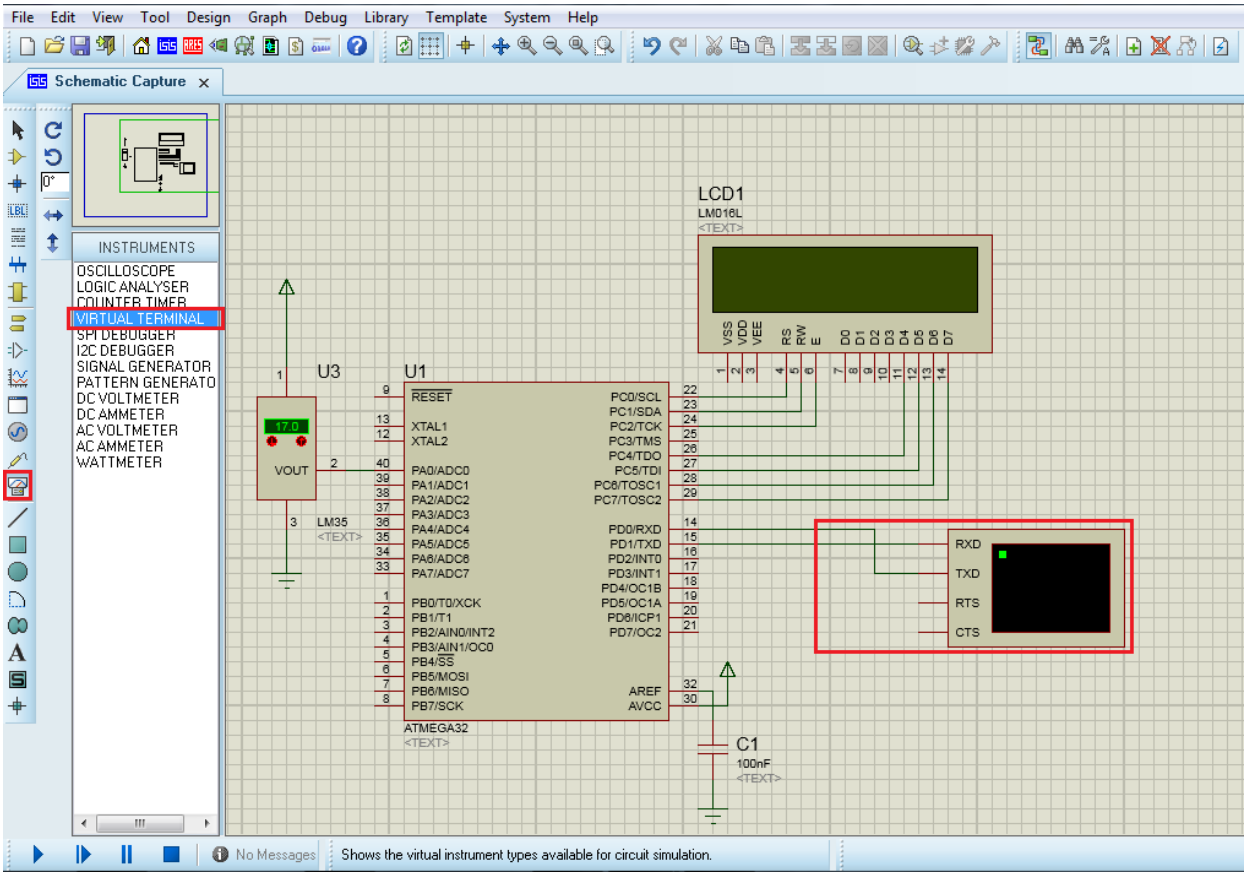
.....

مثال عملی شماره ۷ : برنامه ای با Atmega32 بنویسید که دارای یک LCD کاراکتری ۲ در ۱۶ و یک سنسور LM35 باشد . 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:

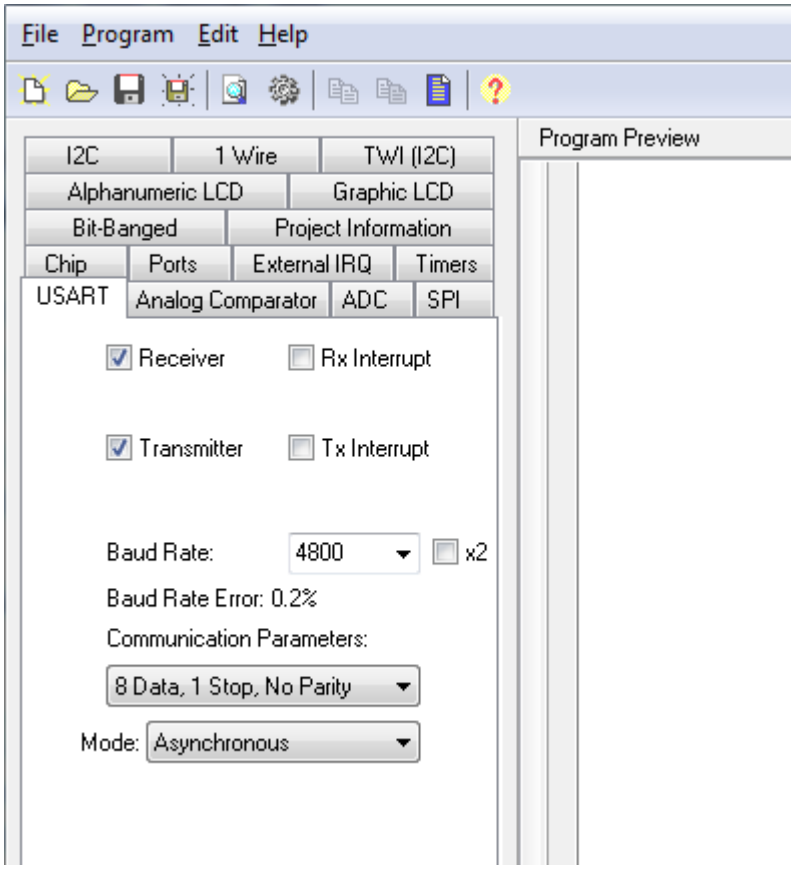
مرحله اول : پیاده سازی سخت افزار در Proteus

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.



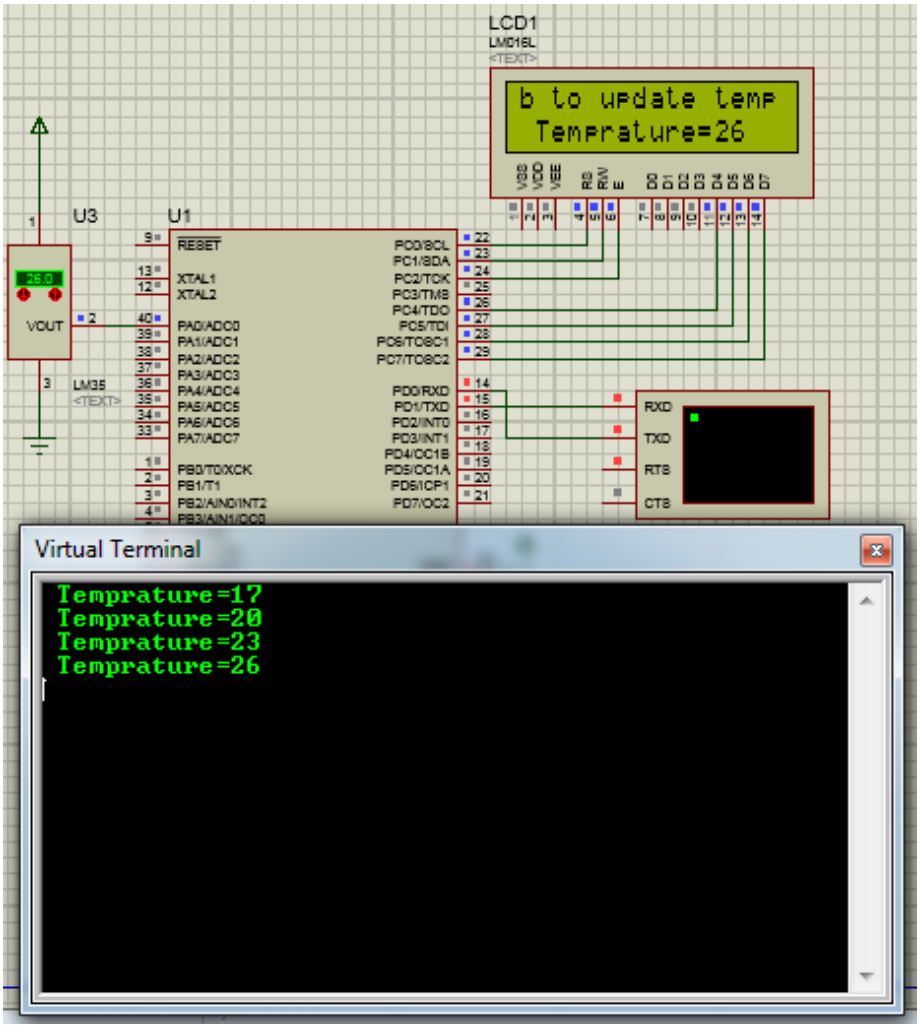
پس از تولید کد اولیه توسط کدویزارد و حذف کدها و کامنت های اضافی نوبت به برنامه نویسی پروژه می رسد . برنامه به صورتی نوشته شده است که ابتدا میکرو منتظر می ماند تا کلید b از صفحه کلید کامپیوتر زده شود سپس از adc مقدار سنسور دما را خوانده و ابتدا روی lcd نمایش داده و سپس به کامپیوتر ارسال می نماید.

```
#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

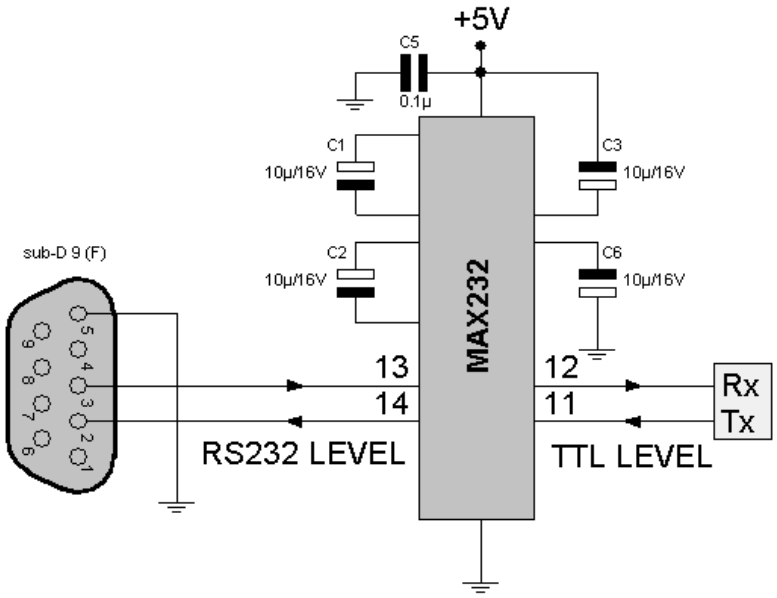
قبل از شروع شبیه سازی باید نرخ ارسال/دریافت (Baud Rate) ابزار ترمینال با نرخ ارسال/دریافت که در کدویزارد تنظیم کرده ایم یکسان باشد تا به درستی کار کند. برای این منظور روی Virtual Terminal دابل کلیک کرده و در پنجره باز شده قسمت Baud Rate را روی ۴۸۰۰ قرار می دهیم.



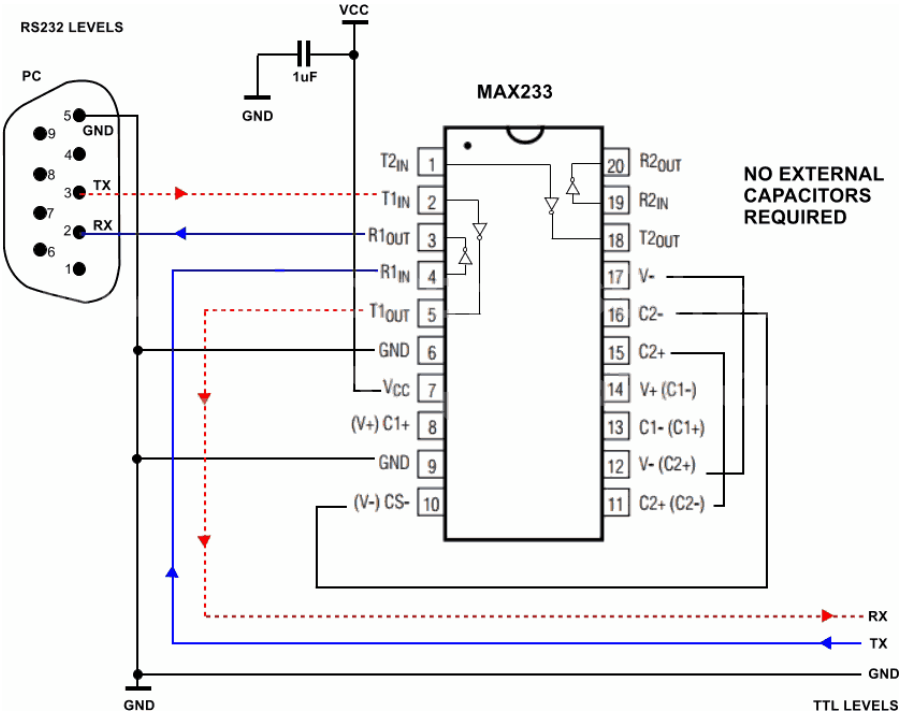
مرحله چهارم : پیاده سازی پروژه

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 Max232 استفاده گردد. نحوه اتصال این آی سی به میکروکنترلر را در شکل زیر مشاهده می کنید.

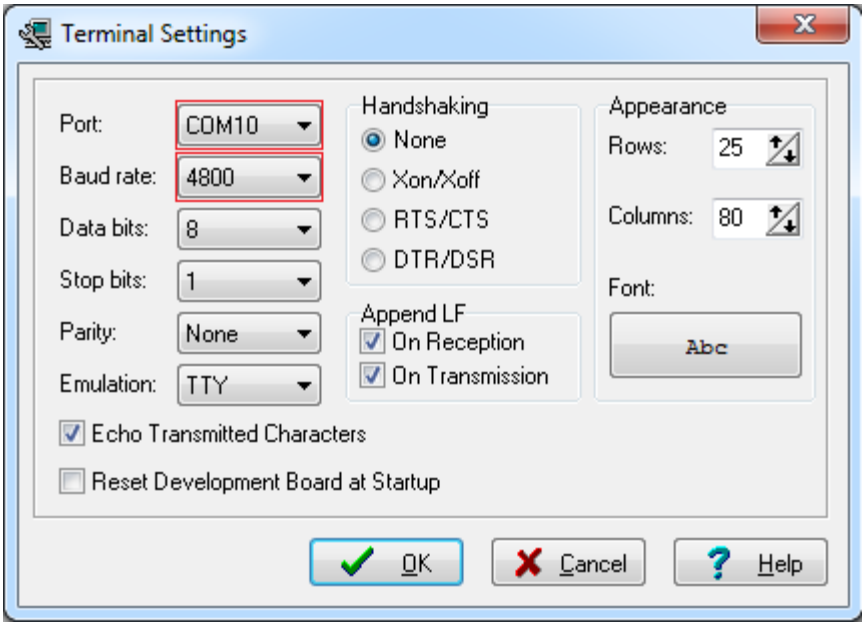


- به علت اینکه برای راه اندازی تراشه MAX232 شما احتیاج به استفاده از چندین خازن دارید ، میتوان از تراشه MAX233 استفاده کرد که در آن احتیاج به استفاده از هیچ خازنی ندارید. نحوه ارتباط این آی سی را در شکل زیر مشاهده می کنید.



- 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 COMپورتی که میکرو به آن وصل است و قاب دیتا و نرخ ارسال/دریافت را نیز مشخص می کنیم.



- سپس از طریق منوی Tools وارد Terminal می شویم و روی دکمه Connect کلیک می کنیم. حالا میتوان برای میکرو کاراکتری را فرستاد یا هر آنچه میکرو ارسال می کند را مشاهده کرد.

دانلود سورس مثال عملی شماره ۷

۸-۶-۷- ماژول های مبدل USB به سریال

در کامپیوتر های امروزی پورت USB پورتی عمومی و کاربرپسند می باشد. همچنین در لپتاپ ها و بسیاری از وسایل پورت سریال DB9 وجود ندارد. وجود این مسائل باعث می شود تا در هنگام استفاده از رابط USART برای اتصال به کامپیوتر، از مبدل های USB به سریال استفاده کنیم. ماژول های USB به سریال به طور کلی شامل انواع زیر هستند:

۱. **مبدل USB to TTL** : این ماژول ها نوعی مبدل USART به USB هستند که از یک طرف دارای Rx و Tx برای اتصال به میکرو و از طرف دیگر دارای پورت USB برای اتصال به کامپیوتر هستند.

۲. **مبدل USB to RS232 :** این ماژول ها دارای یک آی سی مبدل USART به USB به همراه یک آی سی Max232 برای تبدیل USART به RS232 هستند.

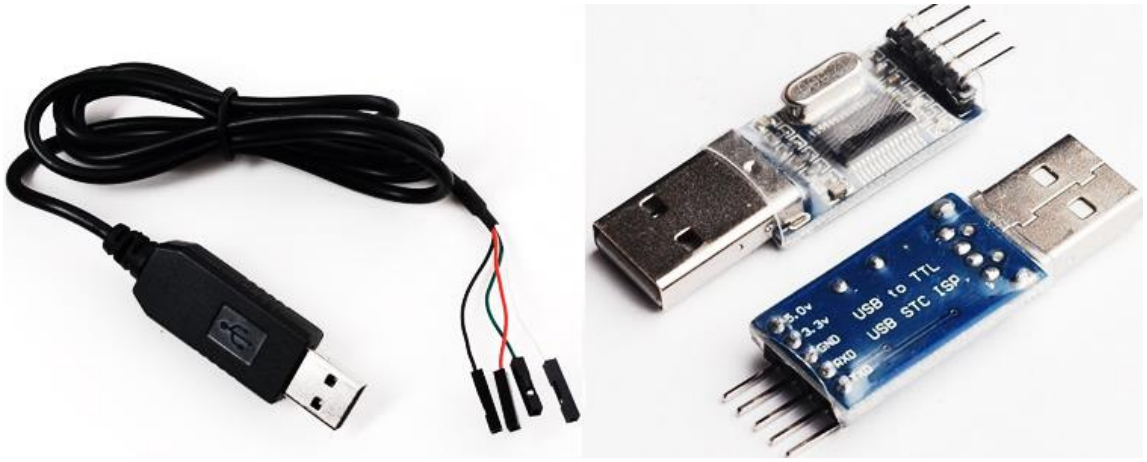
۳- **مبدل USB به RS422 و RS485 :** در این نوع مبدل ها نیز از همان آی سی های USB به سریال به همراه آی سی مبدل سریال به RS422/RS485 استفاده می شود.

۸-۶-۸- انواع مبدل های USBtoTTL

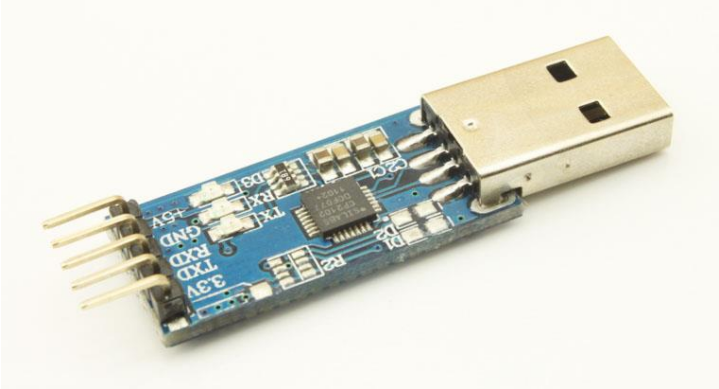
برخی از آی سی هایی که به عنوان مبدل USART به USB عمل می کنند، معرفی می شود:

۱-**مبدل سریال FT232 :** این ماژول بر اساس آی سی FT232 شرکت FTDI chip ساخته شده است.

۲-**مبدل سریال PL2303 :** این ماژول بر اساس آی سی PL2303 شرکت Prolific ساخته شده است و در دو نوع بدون کابل و دارای کابل وجود دارد.



۳- **مبدل سریال CP2102 :** این ماژول بر اساس آی سی CP2102 شرکت SiliconLAB ساخته شده است.



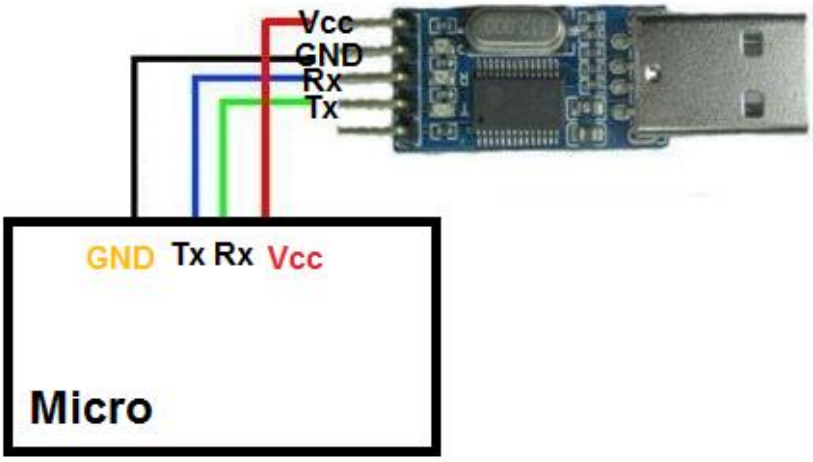
تذکر : تفاوتی میان نحوه عملکرد این آی سی ها وجود ندارد. تنها تفاوت میان این آی سی ها به قیمت آن در بازار و نیز سرعت تبدیل دیتا بر می گردد.

تذکر : هر کدام از این آی سی ها درایور خاص خود را برای راه اندازی نیاز دارد که از سایت شرکت سازنده دانلود و نصب می گردد.

توجه : در این آموزش از ماژول PL2303 استفاده می کنیم که میتوانید درایور آن را از این لینک دانلود نمایید. شکل زیر پایه های نوع سیم دار این ماژول را نشان می دهد.

۸-۶-۹- اتصال ماژول USB به میکرو

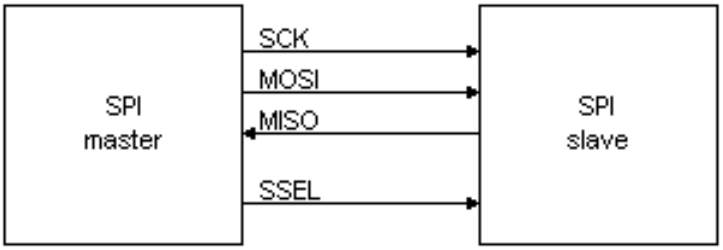
در مثال قبل به جای استفاده از تبدیل RS232 میتوان از آی سی مبدل USB به سریال نظیر PL2303 استفاده کرد. نحوه برنامه نویسی هیچ تفاوتی ندارد و تنها تفاوت در سیم بندی مدار است. همانطور که در شکل زیر مشاهده می کنید، پایه های Rx و Tx میکرو با پایه های Rx و Tx ماژول باید به صورت ضربدری به هم متصل شوند و پایه زمین هر دو نیز به هم وصل باشد. در این روش حتی میتوان تغذیه مدار را از پایه Vcc ماژول گرفت.



۸-۷- 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/Slave است. 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 Masterو Slave مورد استفاده قرار می گیرد

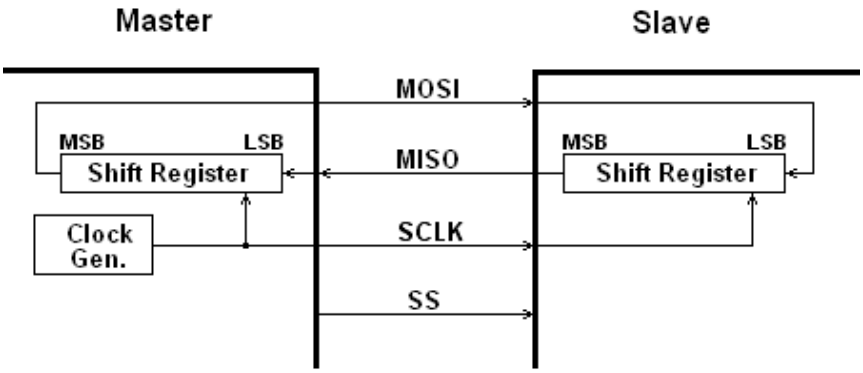
.



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

۸-۷-۱- 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 Slaveوارد 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 Masterو 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.

۸-۷-۲- خصوصیات واحد SPI در میکروکنترلرهای AVR

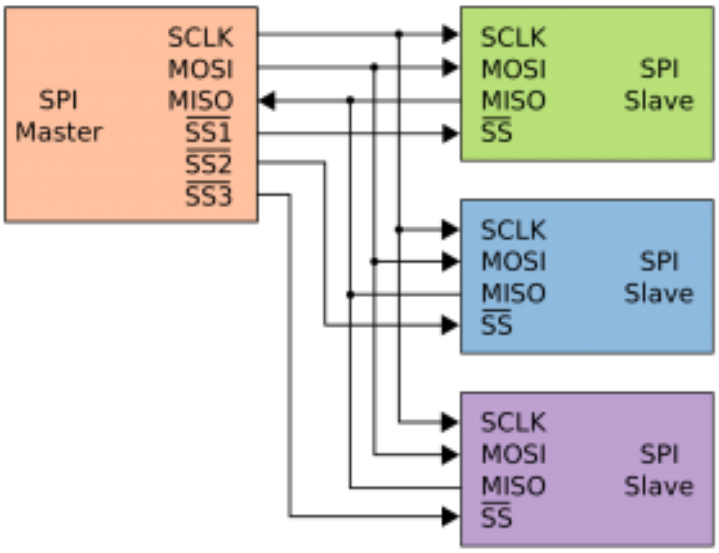
- Full-Duplex
- ارسال اطلاعات به صورت سنکرون توسط ۳ سیم
- عملکرد در حالت های Master وSlave
- اولویت در ارسال بیت ابتدایی LSB یاMSB
- قابلیت تنظیم سرعت انتقال دیتا
- قابلیت ایجاد وقفه در پایان ارسال
- بیدار شدن خودکار از حالت بیکاری(Idle)
- امکان دو برابر کردن سرعت ارسال (در برخی از AVR ها)

۸-۷-۳- شبکه بندی چندین Slave 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 ، MISO و 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

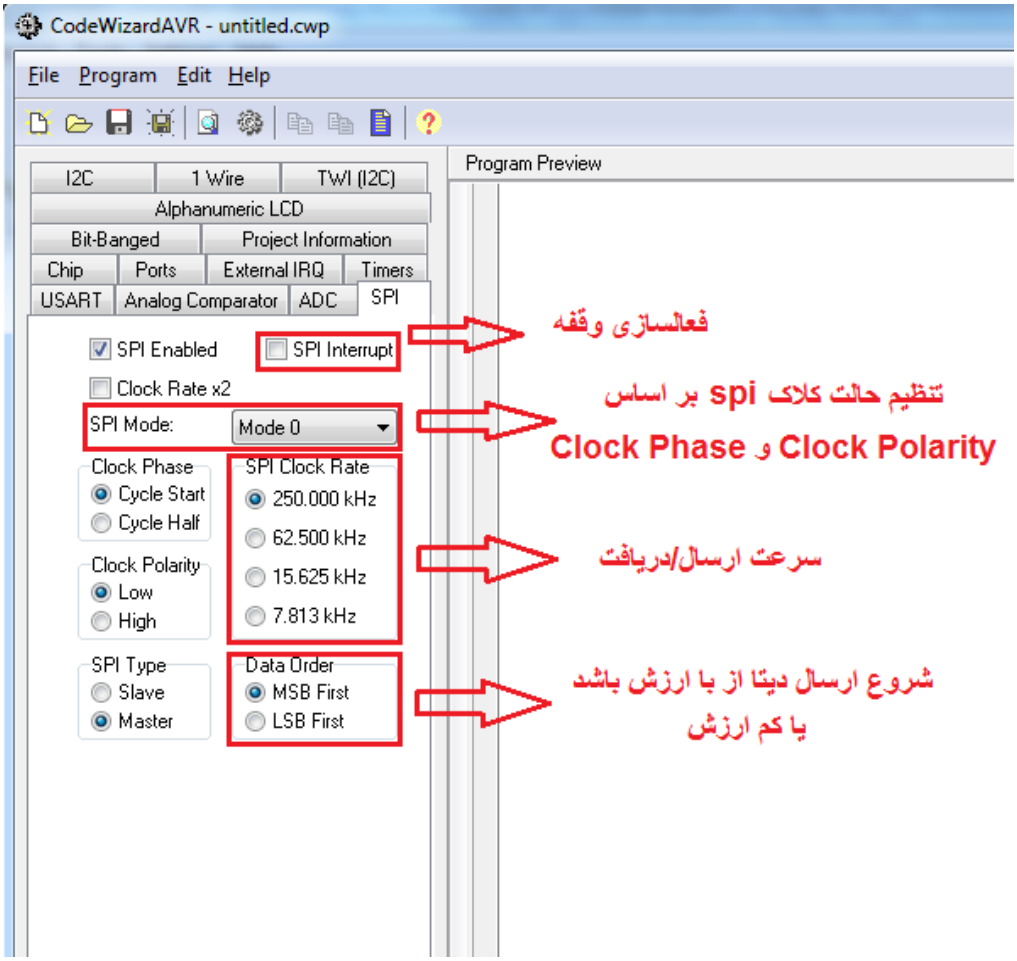
که Masterپین SS در Slave مورد نظر را Low کرده و پین SS مابقی Slave ها را Highمی کند . با انجام این کار بین Master و یکی از Slave ها ارتباط SPI برقرار می شود و دیتای آن دو مبادله می گردد.

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 Slaveاز سه پایه دیگر میکرو استفاده می گردد.



۸-۷-۴- تنظیمات واحد SPI در کدویزارد

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 Rate سرعت ارسال/دریافت دیتا توسط واحد spi مشخص می شود و با فعال کردن Clock Rate X2نیز 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 Polarityمشخص می شود که پالس کلاک در حالت بیکاری (Idle) در منطق Lowیا High قرار گیرد . براساس تنظیمات قسمت های Clock Phase و Clock Polarity چهار حالت مختلف برای ارسال/دریافت دیتا در واحد SPI بوجود می آید که به آنها Mode های واحد Spi گویند . دقت شود که مد Master و Slave ها باید یکسان باشند . در بخش Data Order نیز 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.

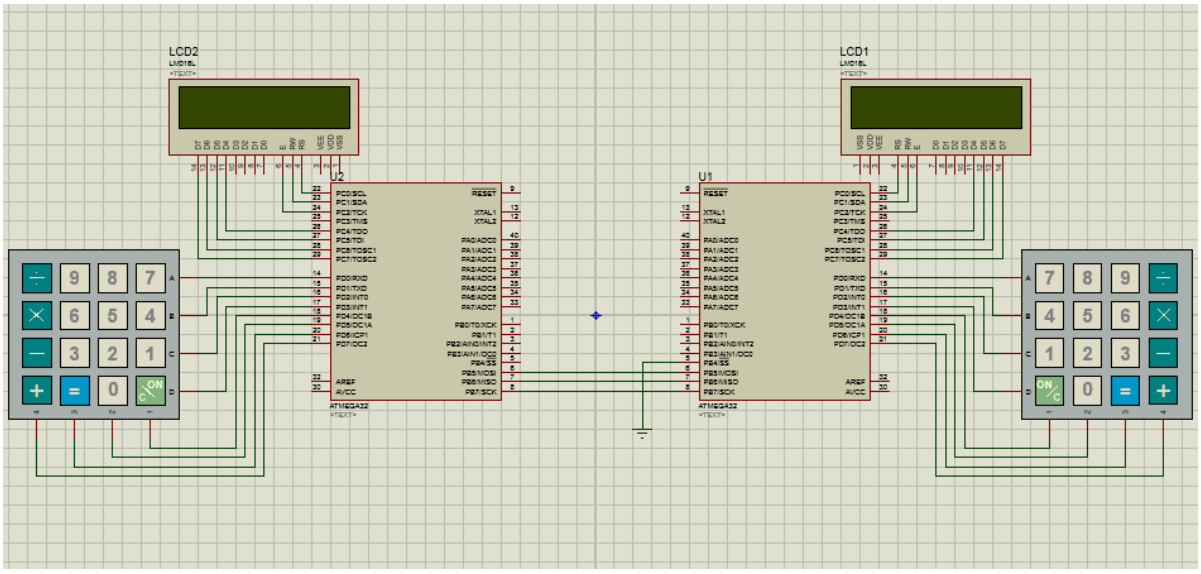
مثال عملی شماره ۸ : دو میکروکنترلر Atmega32 را از طریق پورت Spi به یکدیگر متصل کنید . به هر یک از میکروکنترلرها یک LCD کاراکتری ۲ در ۱۶ و یک صفحه کلید ۴ در ۴ متصل کرده و سپس برنامه ای

بنویسید که با زدن کلیدی از هر میکروکنترلر ، کاراکتر مورد نظر به میکروکنترلر دیگر ارسال شده و روی LCDآن نمایش داده شود.

حل:

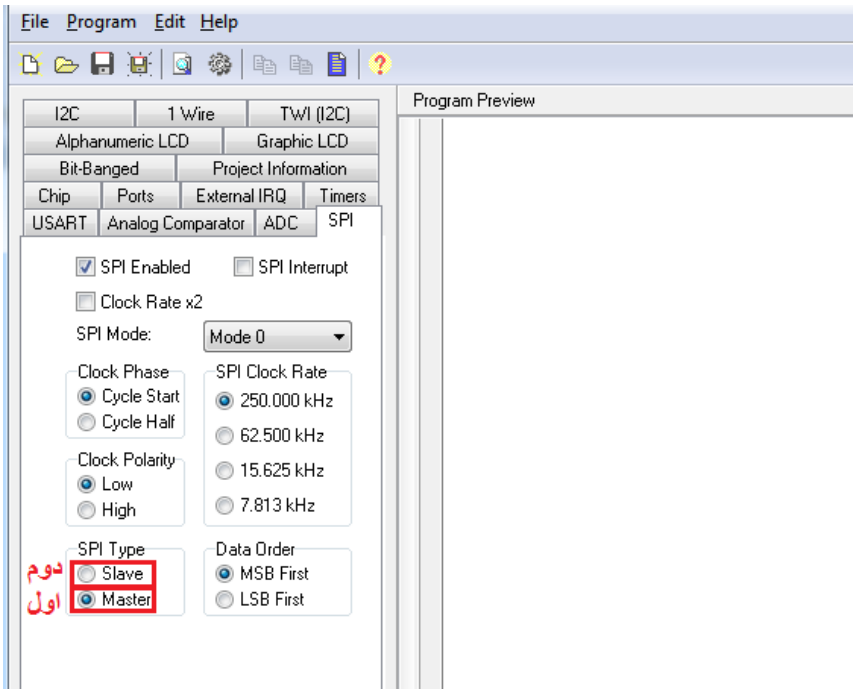
مرحله اول : پیاده سازی سخت افزار در پروتئوس

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 : تنظیمات کدویزارد

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 کد 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
            }
      }
}
```

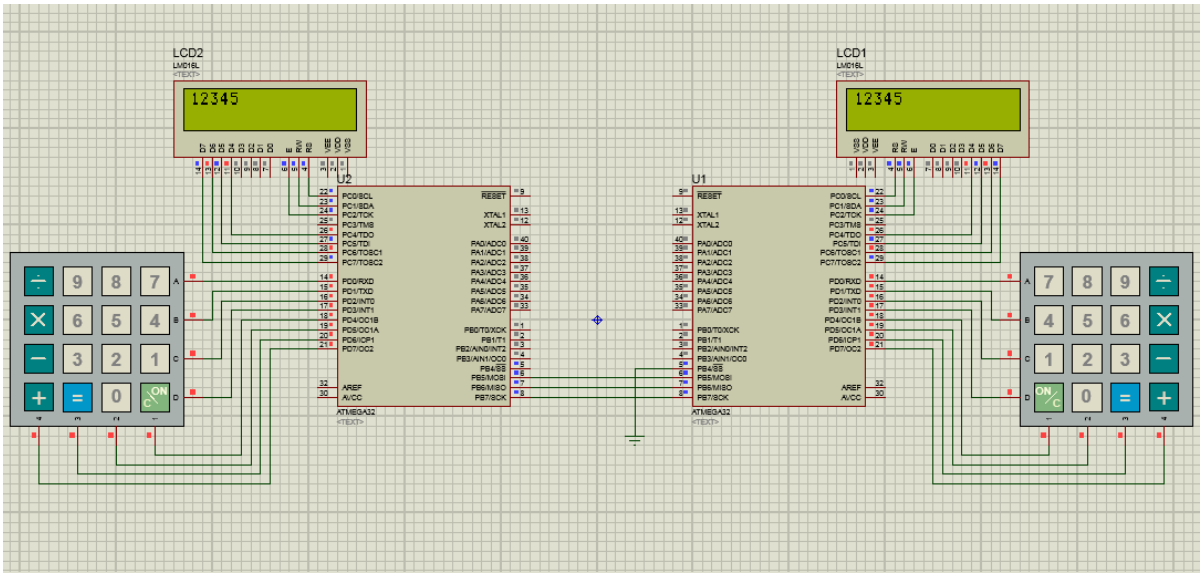
برنامه Slave :

```
#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 SPCRدر 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

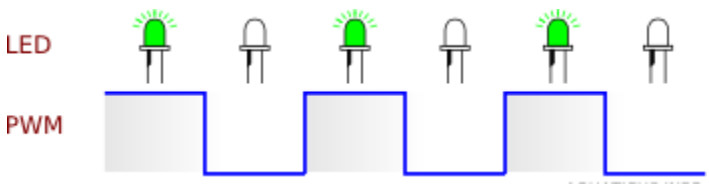
نکته خاصی در این قسمت وجود ندارد و اگر همه اتصالات و خصوصا برنامه نویسی صحیح باشد برنامه به درستی کار می کند.

[دانلود سورس مثال عملی شماره ۸]

۸-۸- راه اندازی واحد تایمر/کانتر

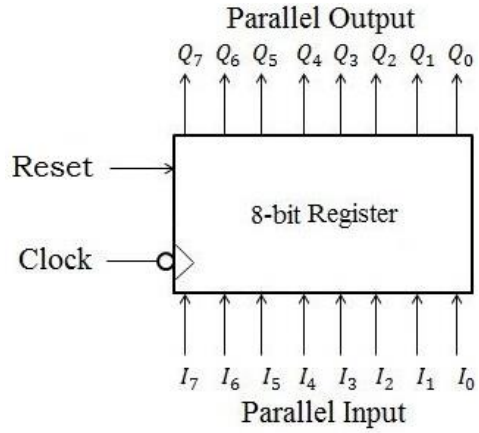
مقدمه:

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 .



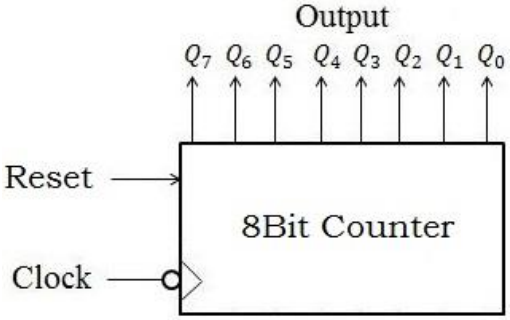
۸-۸-۱- رجیستر چیست ؟

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.



۸-۸-۲- 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....



۸-۸-۳- 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.

مفهم 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.

۸-۸-۴- واحد تایمر/کانتر در میکروکنترلرهای AVR

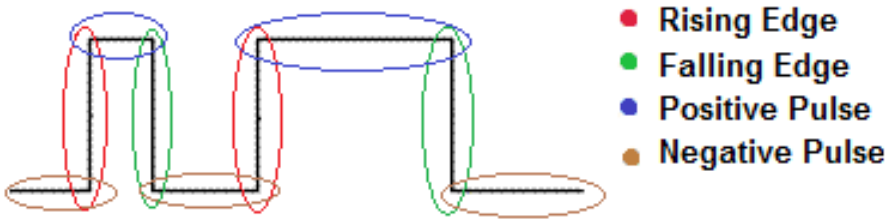
مهمترین مسائلی که در هنگام کار با واحد تایمر/کانتر بوجود می آید به شرح زیر می باشد:

- **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:

۱. **لبه بالارونده(Rising Edge)**

۲. **لبه پایین رونده(Falling Edge)**
۳. **پالس مثبت(Positive Pulse)**
۴. **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:

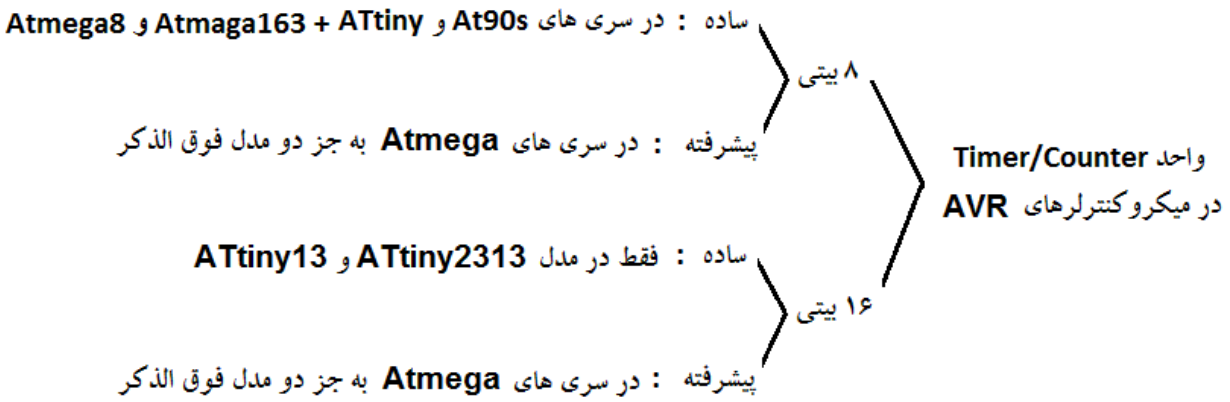
۱. **حالت ساده(عادی)**
۲. **حالت مقایسه(CTC)**
۳. **حالت PWM سریع (تک شیب)**
۴. **حالت PWM تصحیح فاز (دو شیب)**
۵. **حالت PWM تصحیح فاز و فرکانس**

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 **x یک 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.

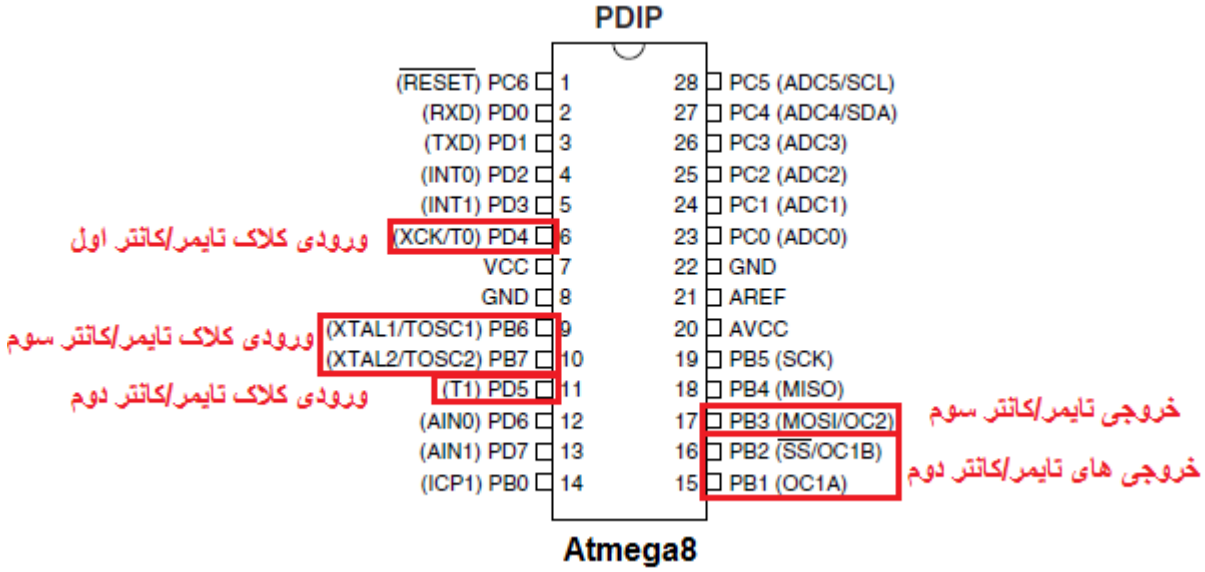
۸-۸-۵- انواع واحد تایمر/کانتر در میکروکنترلرهایAVR

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 TOSC1 و (TOSC2 تنها میتواند برای RTC (زمان سنج حقیقی) استفاده شود.

۸-۸-۶- معرفی رجیستر های واحدهای تایمر/کانتر ۸ بیتی

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:

۱. فعال یا غیر فعال بودن واحد
۲. مشخص کردن حالت کار تایمری یا کانتری (منبع کلاک)
۳. تنظیم عدد تقسیم کلاک ورودی واحد در حالت تایمری
۴. مشخص کردن نوع رخداد در حالت کانتری
۵. تنظیم حالت های مختلف کاری واحد (PWM ، CTCو ...)
۶. 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 TCNTمقایسه می شود. از تطابق این دو برای تولید وقفه خروجی یا تولید یک شکل موج روی پایه OCX می توان استفاده نمود.

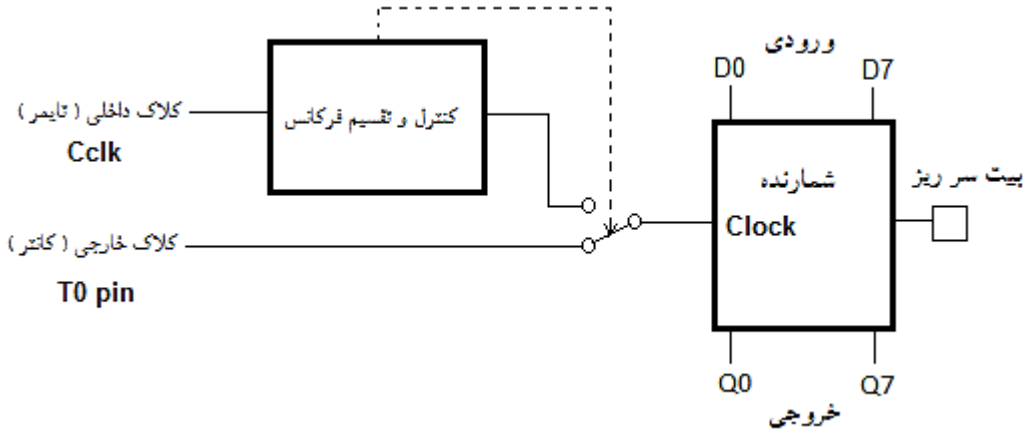
رجیستر پوشش وقفه تایمر/کانتر (TIMSK) :

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.

۸-۸-۷- 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/1024را میتوان انتخاب نمود.

نکته : در حالت استفاده از کلاک خارجی (کانتر) باید توجه داشت که حداقل زمان بین دو رخداد Cclk می باشد (Cclk همان 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 Fclk فرکانس 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:

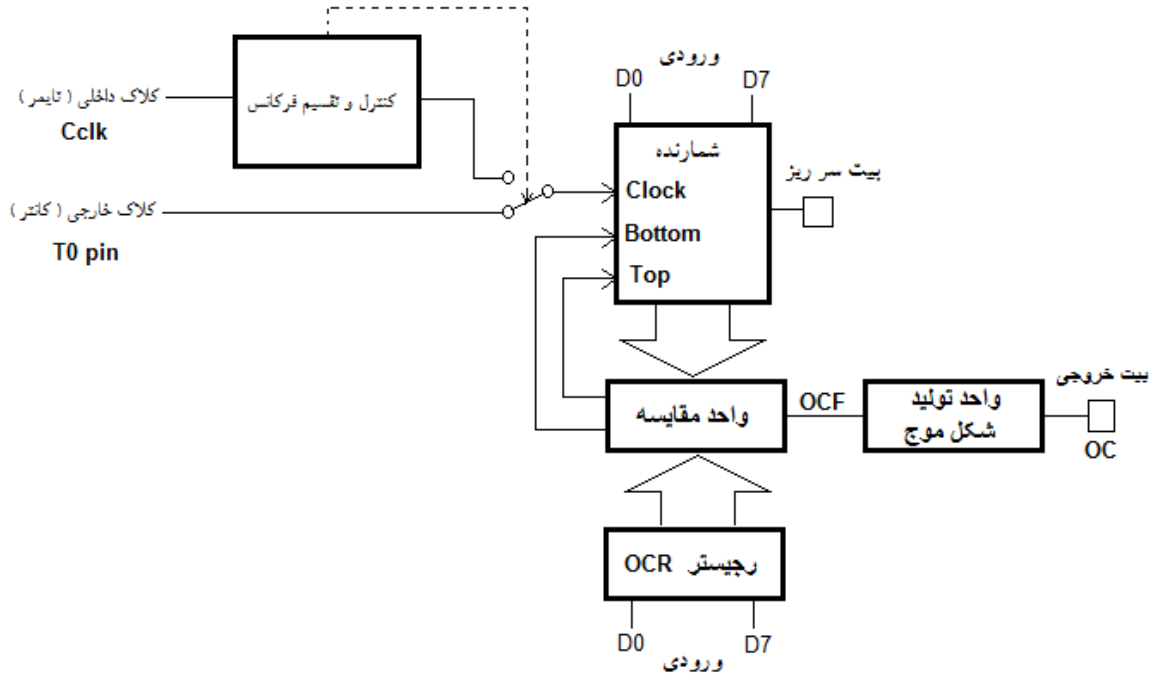
$$256 \times t = \frac{N \times 256}{f_{clk}}$$

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.

۸-۸-۸- 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 (Mode) زیر باشد:

۱. تایمر/کانتر در حالت ساده(Normal)
۲. تایمر/کانتر در حالت مقایسه(CTC)
۳. تایمر/کانتر در حالت PWM سریع(Fast PWM)
۴. 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 OCxمیکروکنترلر را دارد و در صورتی که واحد تایمر/کانتر در حالت ساده یا حالت CTC باشد برای آن چهار حالت عملکرد زیر قابل تنظیم می باشد:

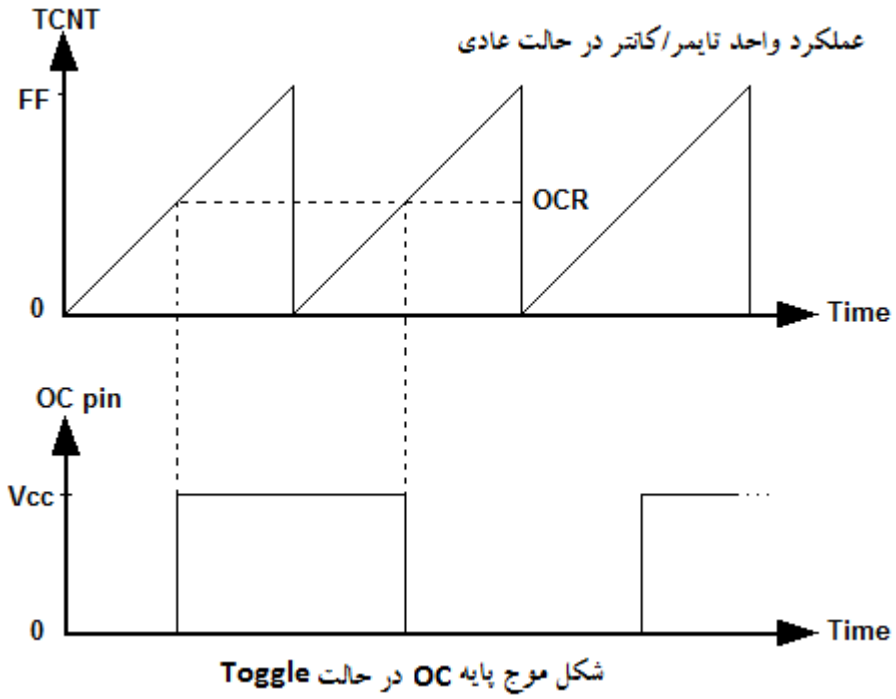
۱. پایه OC غیر فعال است(Disconnected) .
۲. پایه OC در هر بار تطبیق تغییر وضعیت دهد یعنی اگر ۰ بود ۱ شود و بالعکس(Toggle) .
۳. پایه OC در هر بار تطبیق ۰ شود(Clear) .
۴. پایه OC در هر بار تطبیق ۱ شود(Set) .

در صورتی که واحد تایمر/کانتر در حالت PWM باشد (حالت ۳ یا ۴) برای آن سه عملکرد زیر قابل تنظیم می باشد:

۱. پایه OC غیر فعال است(Disconnected) .
۲. پایه OC در حالت PWM رابطه عکس با رجیستر OCR دارد(Inverted)
۳. ایه OC در حالت PWM رابطه مستقیم با رجیستر OCR دارد(Non-Inverted)

۸-۸-۹- 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

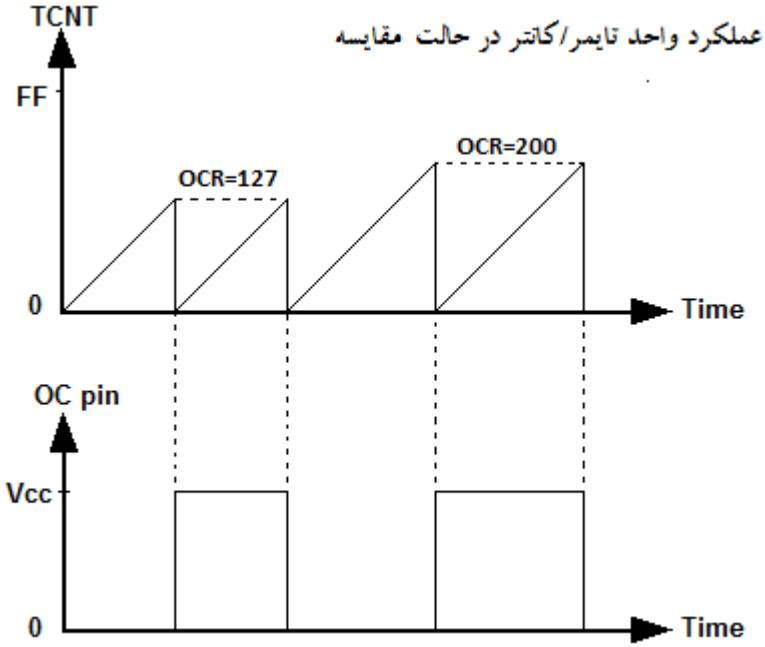
فرکانس شکل موجی که در حالت عادی روی پایه OC ایجاد می شود ، همیشه ثابت بوده و از رابطه زیر بدست می آید:

$$f_{Normal} = \frac{1}{2 \times OVF_{Time}} = \frac{f_{clk}}{2N \times (256 - TCNTx)}$$

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 OCRکه خود مقداری بین ۰ تا ۲۵۵ میتواند باشد ، ریست می شود . شکل زیر عملکرد واحد را در این حالت نشان می دهد.



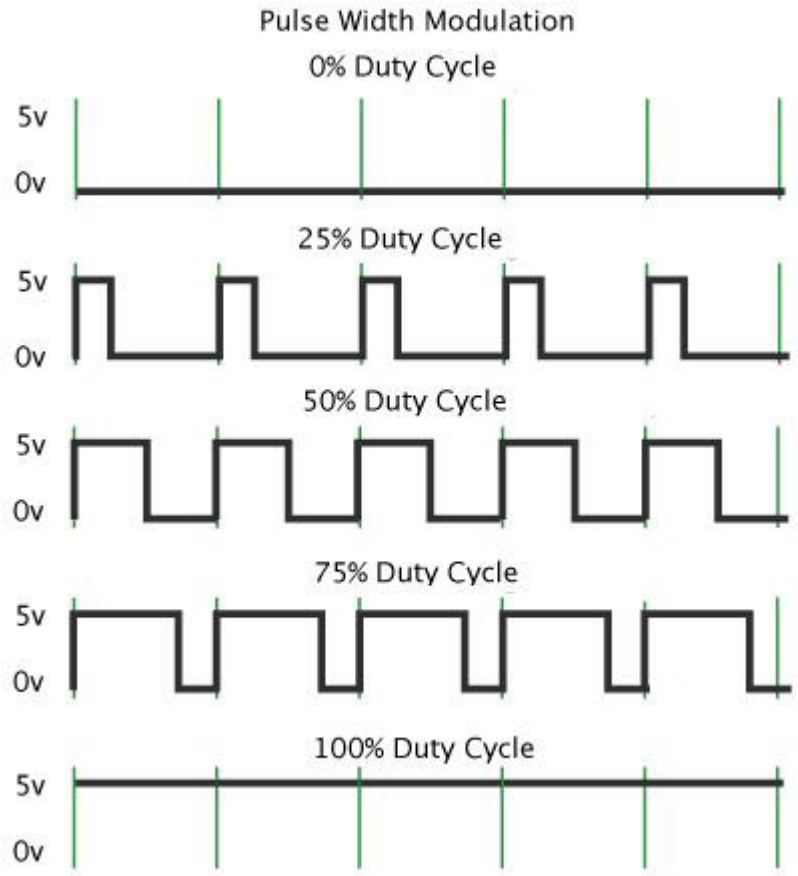
شکل موج پایه OC در حالت Toggle

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.

$$f_{ocx} = \frac{f_{clk}}{2N(OCRx + 1)}$$

۸-۸-۱۰- PWMچیست ؟

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 Cycleیک 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 PWMبا دیوتی سایکل ٪۵۰ ایجاد نمایید ولتاژ RMS شما 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 Cycle بیشتر باشد ، موتور سریعتر و هرچه کمتر باشد موتور کندتر می چرخد . در دیوتی سایکل ۱۰۰ درصد بیشترین توان به موتور اعمال می شود.

۸-۸-۱۱- 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 1/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)
{
```

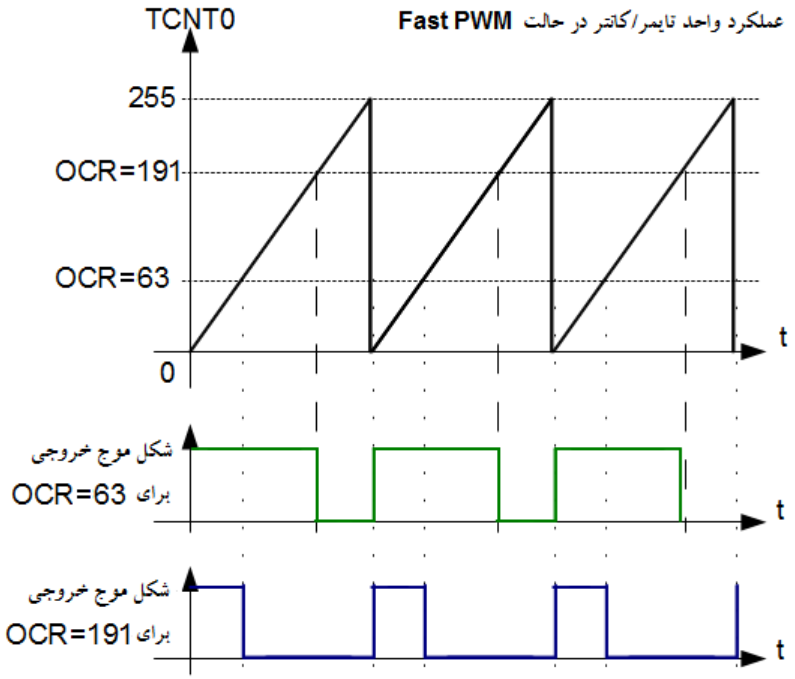
```
PORTA.0=0;
delay_ms((1-DutyCycle/100)*T);
PORTA.0=1;
delay_ms(DutyCycle/100*T);
}
```

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 PWMاز **سخت افزار مجزا از CPU** به نام واحد تایمر/کانتر در حالت PWM میتوان استفاده کرد.

۸-۸-۱۲- بررسی تایمر/کانتر ۸ بیتی پیشرفته در حالت PWM سریع(Fast PWM)

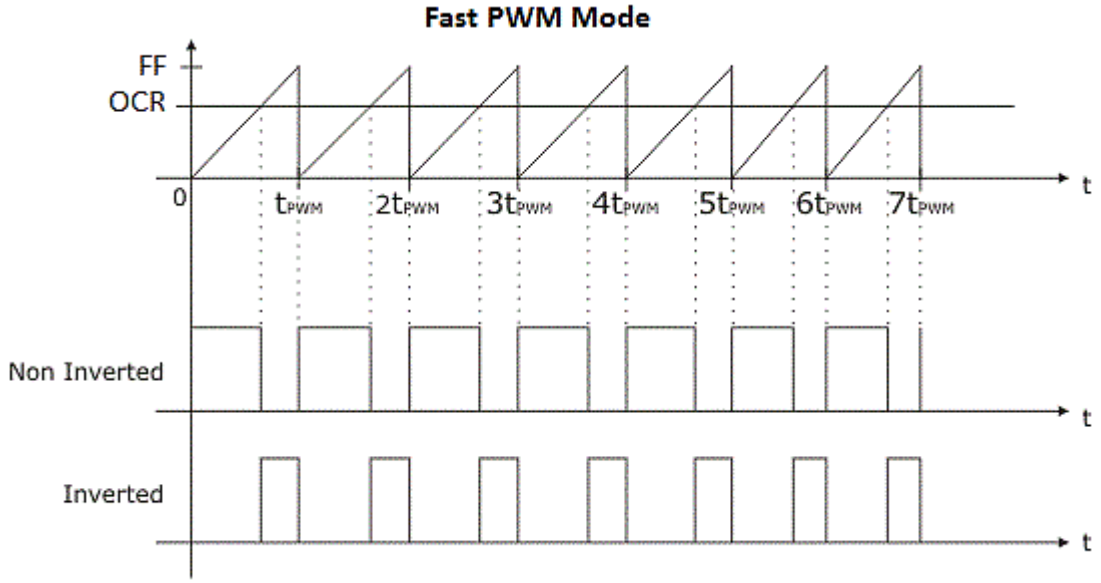
در این حالت عملکرد واحد تایمر/کانتر مانند حالت مقایسه است با این تفاوت که در زمان تطابق میان OCR و TCNTمقدار شمارنده ریست نمی شود بلکه تا مقدار ماکزیمم خود یعنی ۲۵۵ ادامه میابد و بعد از سر ریز شدن صفر می گردد . همچنین هنگام ۰ شدن مقدار شمارنده پایه OC نیز صفر می شود . در حالت PWM پایه OC در دو حالت تغییر 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.

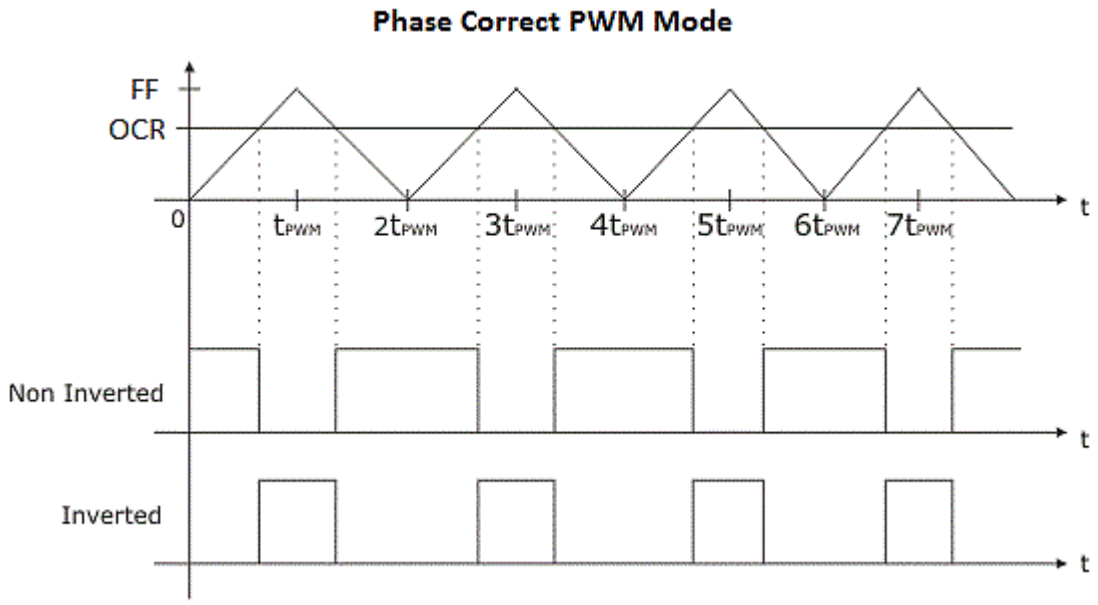
$$f_{ocx} = \frac{f_{clk}}{N \times 256}$$

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.



۸–۸–۱۳– 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.



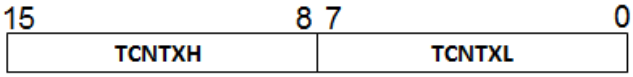
نکته : تولید PWM به روش تصحیح فاز دارای فرکانس خروجی پایین تری نسبت به حالت سریع آن است و طبق رابطه زیر محاسبه می شود که در آن N ضریب پیش تقسیم کننده و Fclk فرکانس کاری میکرو است.

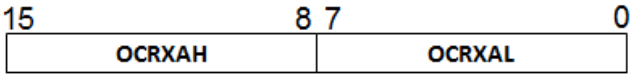
$$f_{ocx} = \frac{f_{clk}}{N \times 510}$$

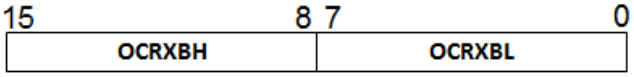
۸-۸-۱۴- معرفی اجمالی رجیسترهای تایمر/کانترهای ۱۶ بیتی

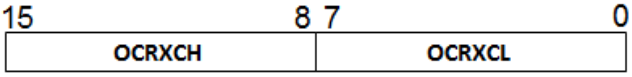
در تایمر/کانترهای ۱۶ بیتی ، شمارنده اصلی تایمر/کانتر رجیستر TCNTx می باشد . این رجیستر یک رجیستر ۱۶ بیتی است که از دو رجیستر ۸ بیتی با نامهای TCNTXLو TCNTXH تشکیل شده است و به صورت خواندنی و نوشتنی قابل دسترسی است .

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 ، OCRxBو 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)

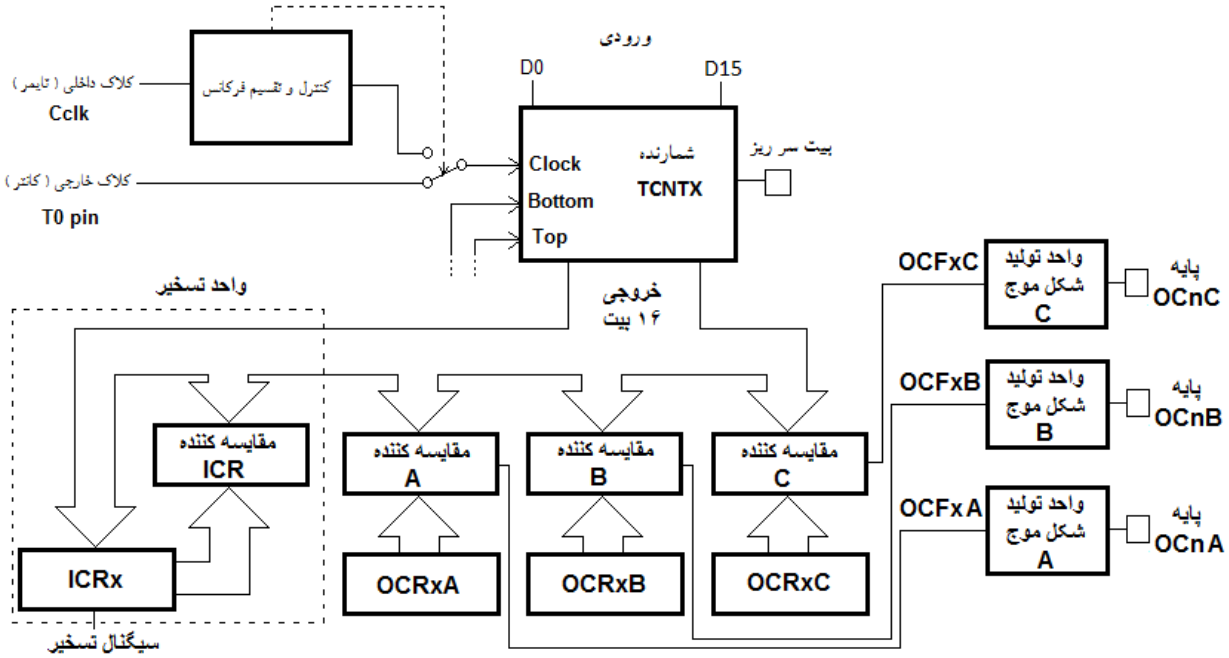
در تایمر/کانترهای ۱۶ بیتی به جای یک رجیستر تنظیمات سه رجیستر **۸ بیتی** به نام های TCCRXA ، TCCRXBو TCCRXC وجود دارد . این سه رجیستر کلیه اعمال واحد تایمر/کانتر را نظیر حالت عملکرد ، ضریب پیش تقسیم کننده ، نحوه اتصال خروجی و ... کنترل می کند.

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 AtTiny2313 استفاده 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.

۸-۸-۱۵- 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 میکروکنترلرهای 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:

۱. تایمر/کانتر در حالت ساده(Normal)
۲. تایمر/کانتر در حالت مقایسه(CTC)
۳. تایمر/کانتر در حالت PWM سریع(Fast PWM)
۴. تایمر/کانتر در حالت PWM تصحیح فاز(Phase Correct PWM)
۵. 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.

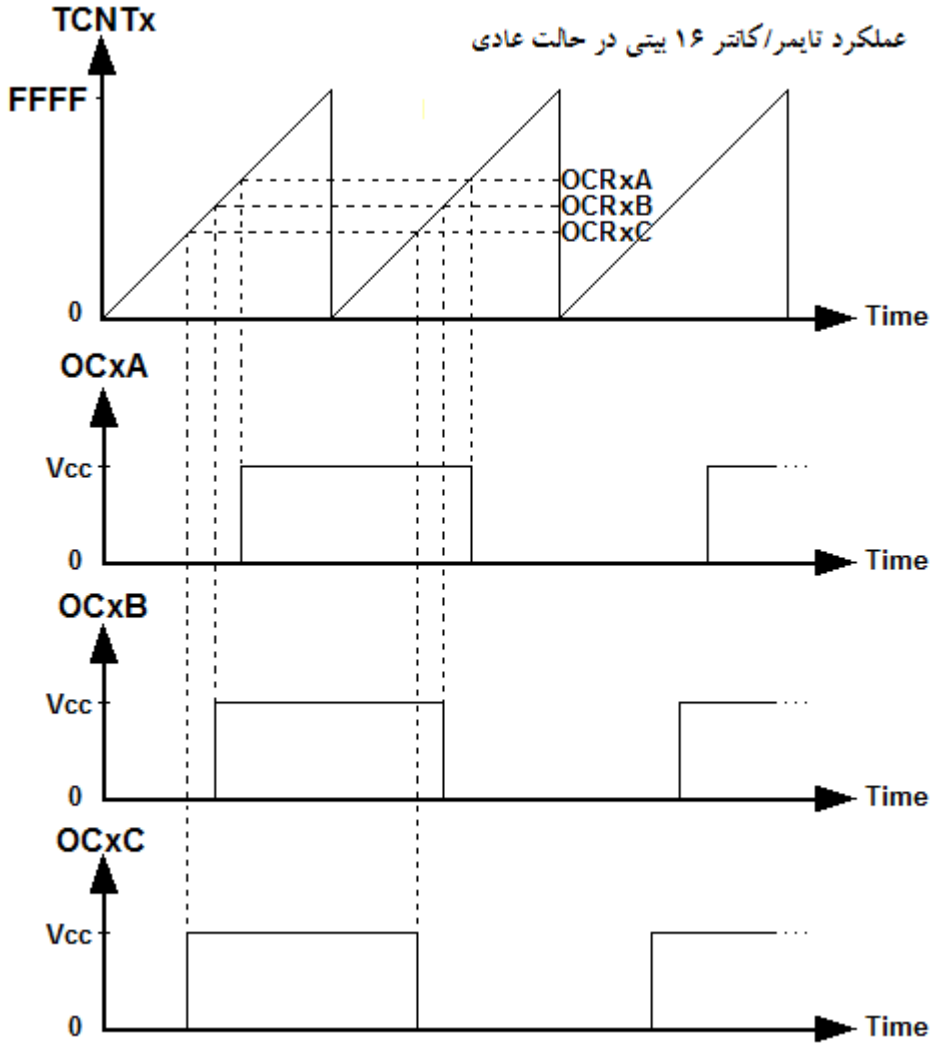
نکته : اکثر میکروکنترلرهای AVR حداکثر ۳ واحد تایمر/کانتر دارند و فقط میکروکنترلرهایAtmega64 ، Atmega128 ، Atmega162 ، Atmega2560 ، Atmega2561 ، Atmega640 ، Atmega1280و Atmega1281هستند که دارای تایمر چهارم و پنجم نیز می باشند.

نکته : اگر فیوز بیت M161C در میکروکنترلر Atmega162 که برای تطابق میکروکنترلر Atmega162با Atmega161تعبیه شده است ، فعال باشد ، تایمر/کانتر شماره ۳ در دسترس نبوده و غیر فعال می شود.

نکته : اگر فیوز بیت M103C در میکروکنترلرهای Atmega128 و Atmega64 که برای تطابق میکروکنترلرهای Atmega64 و Atmega128 با Atmega103 تعبیه شده است ، فعال باشد ، تایمر/کانتر شماره ۳ و نیز رجیستر OCR3C در دسترس نبوده و غیر فعال می شود.

۸-۸-۱۶- 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 0 بر می گردد . در هنگام شمارش رجیستر TCNTx دائما 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.



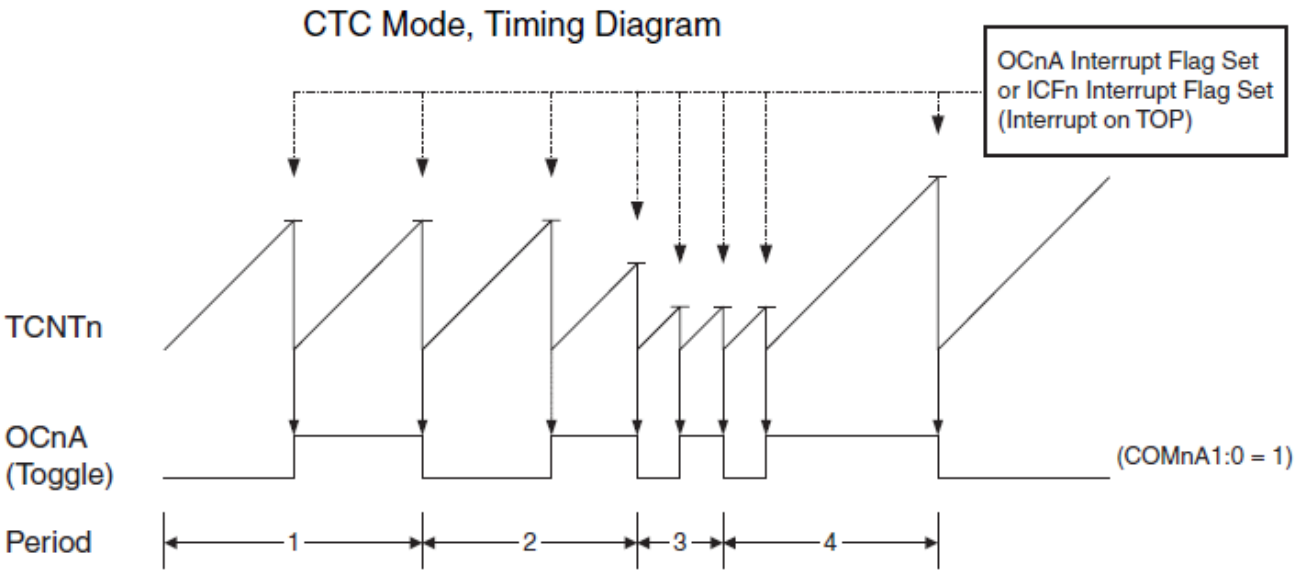
فرکانس شکل موجی که در حالت عادی روی پایه OCx ایجاد می شود ، همیشه ثابت بوده و از رابطه زیر بدست می آید:

$$f_{ocx} = \frac{f_{clk}}{2N \times (65536 - TCNTx)}$$

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.

۸-۸-۱۷- 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:

$$f_{OCnA} = \frac{f_{clk}}{2 \cdot N \cdot (1 + OCRnA)}$$

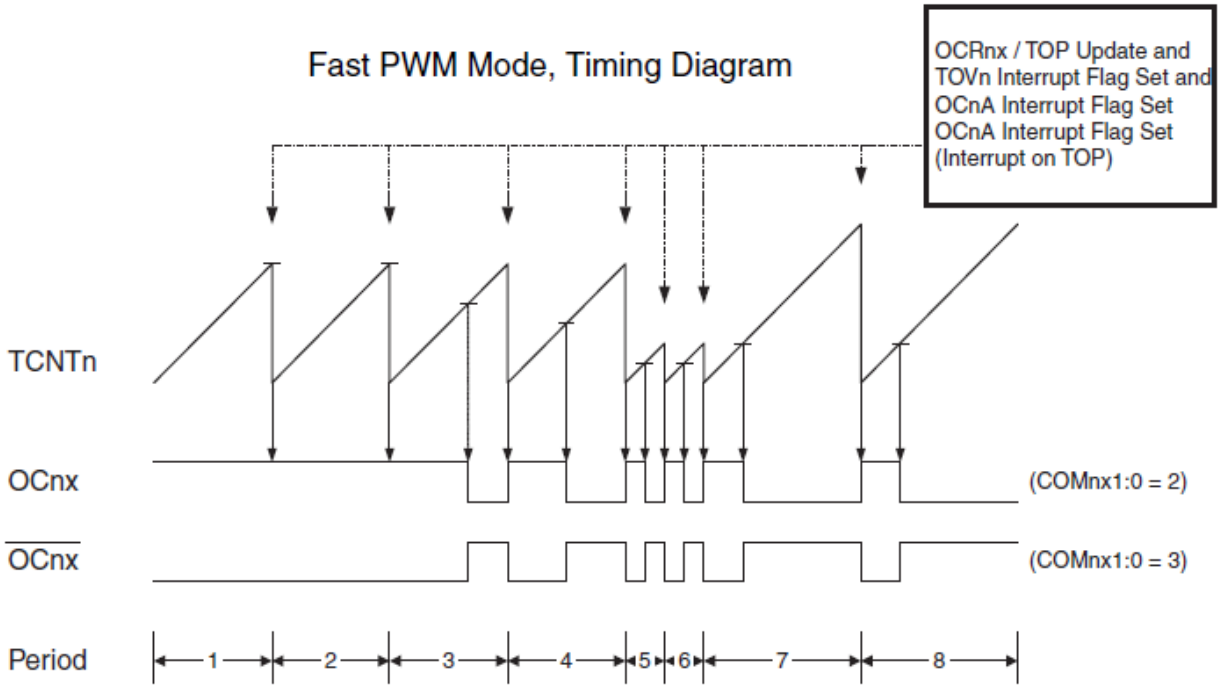
برای پایه های OCnB و OCnC نیز چنین فرمولی صادق است که در آن N ضریب پیش تقسیم کننده و Fclk فرکانس کاری میکرو است.

۸-۸-۱۸- 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:

۱. PWMسریع هشت بیت (با حداکثر مقدار(FF
۲. PWMسریع نه بیت (با حداکثر مقدار ۱(FF
۳. PWMسریع ده بیت (با حداکثر مقدار ۳(FF
۴. PWMسریع با حداکثر مقدار رجیستر ICRx
۵. PWMسریع 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 ICRx و 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.

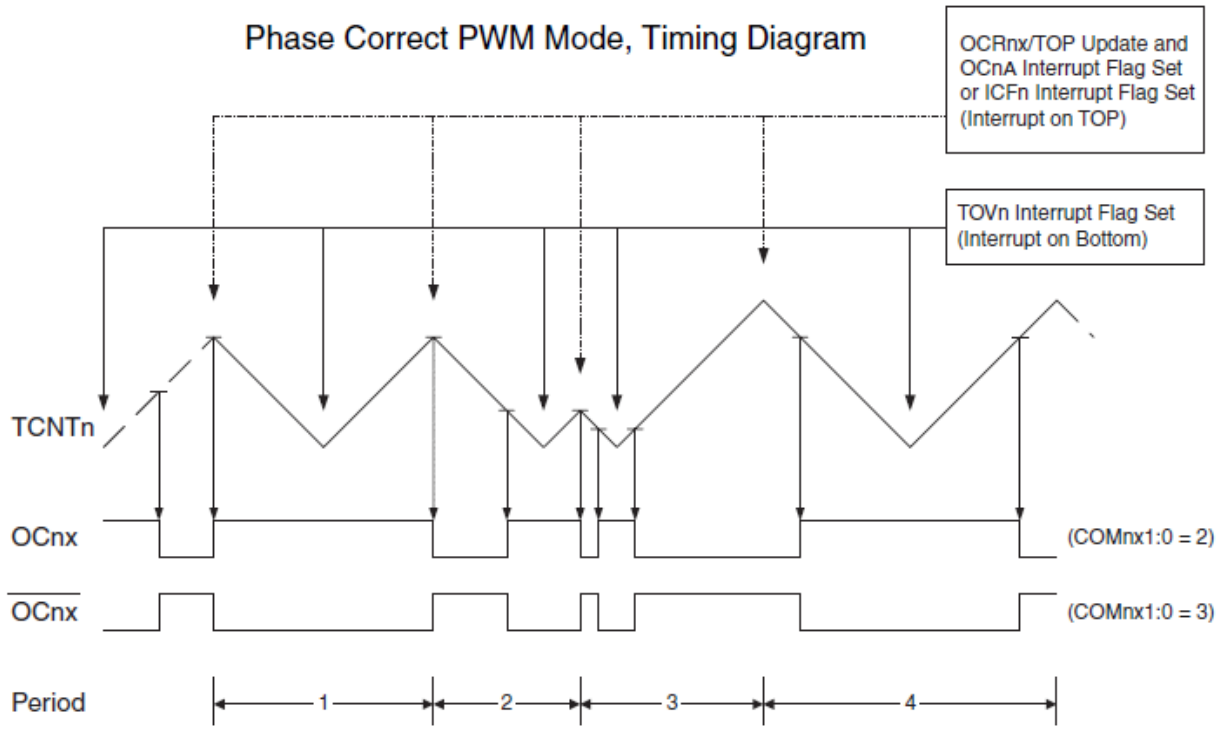
۸–۸–۱۹– تایمر/کانتر ۱۶ بیتی پیشرفته در حالت PWM تصحیح فاز(Phase Correct PWM)

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:

۱. PWMتصحیح فاز هشت بیت (با حداکثر مقدار FF)
۲. PWM تصحیح فاز نه بیت (با حداکثر مقدار 1FF)
۳. PWM تصحیح فاز ده بیت (با حداکثر مقدار 3FF)
۴. PWM تصحیح فاز با حداکثر مقدار رجیسترICRx
۵. PWM تصحیح فاز با حداکثر مقدار رجیسترOCRxA

در این حالت تایمر تا زمانی که مقدارش مطابق تنظیمات فوق یکی از مقادیر FF ، 1FF، 3FF، مقدار درون رجیستر ICRx و 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 ICRx یا 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.

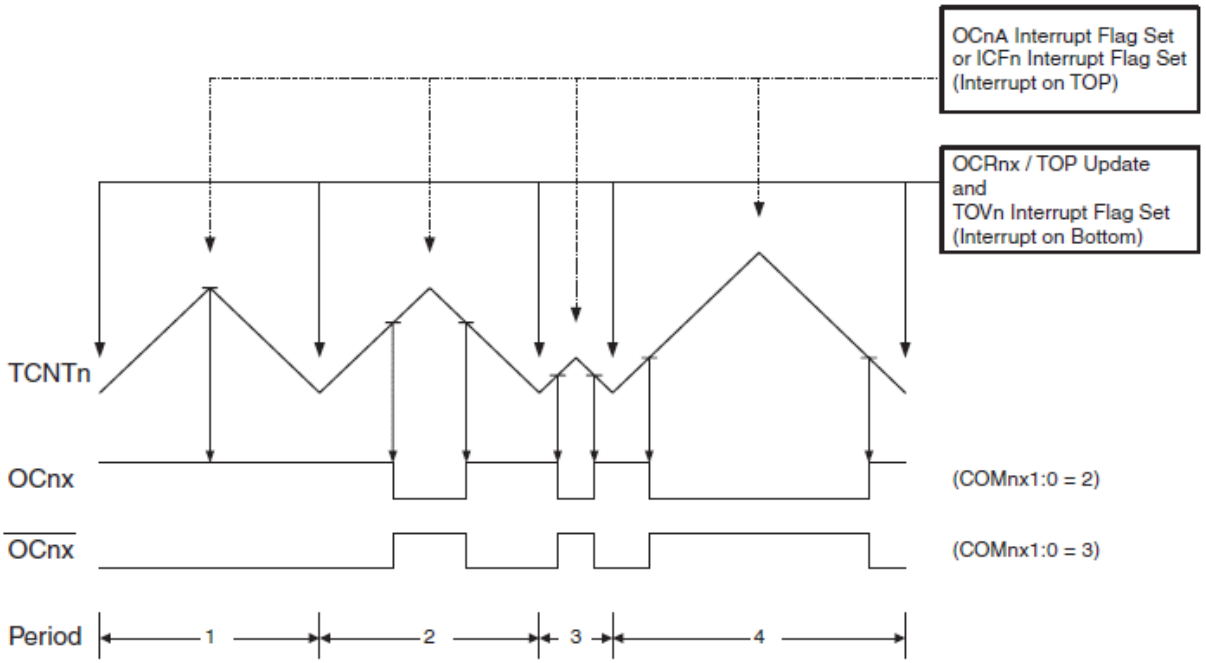
۸–۸–۲۰– تایمر/کانتر ۱۶ بیتی پیشرفته در حالت PWM تصحیح فاز و فرکانس (Phase & Frequency Correct PWM)

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:

۱. PWM تصحیح فاز و فرکانس با حداکثر مقدار رجیستر ICRx
۲. PWM تصحیح فاز و فرکانس با حداکثر مقدار رجیستر OCRxA

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,Cکنترل 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.

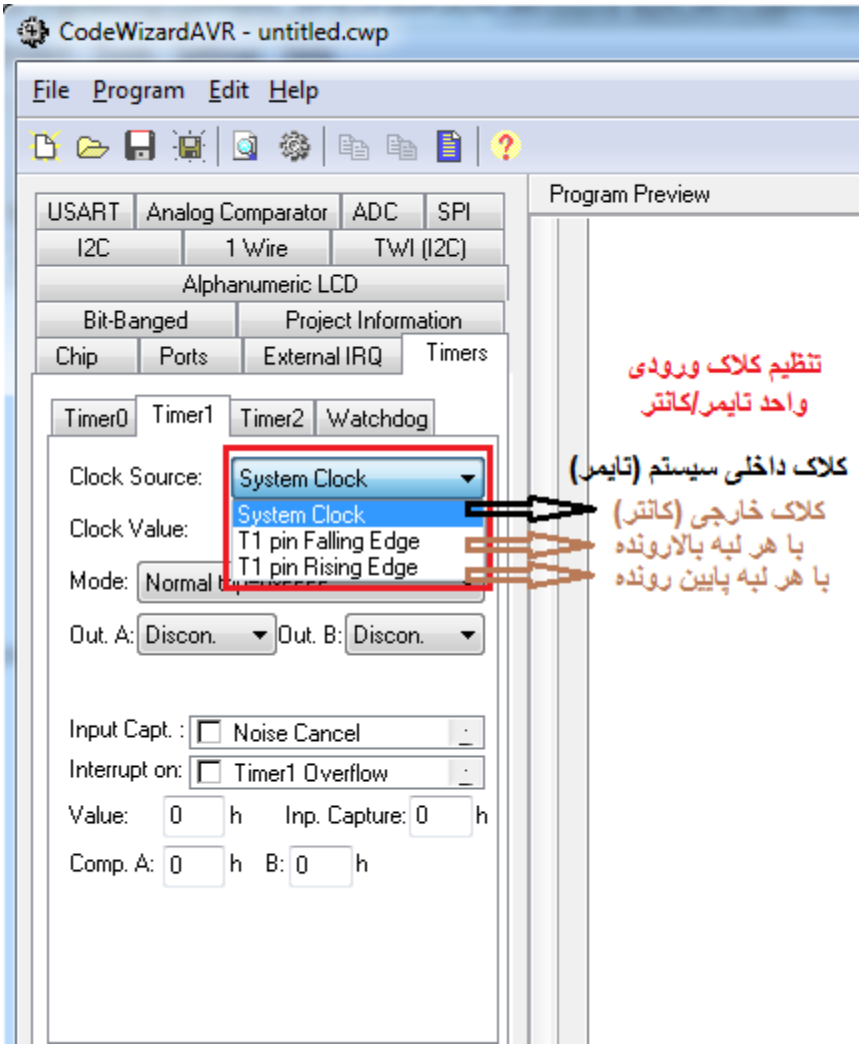
۸–۸–۲۱– تنظیمات واحد تایمر/کانتر در کد ویزاردCodeWizard

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 OCRxپایه 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.

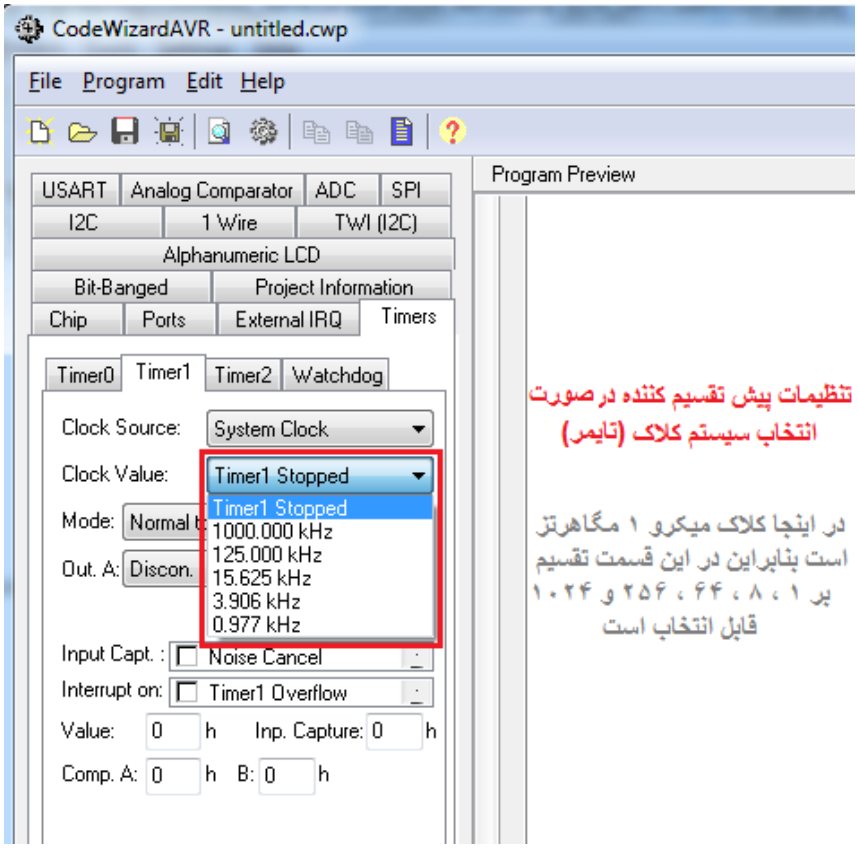
نکته : در برخی از تایمر/کانترها واحد کاهش نویز (Noise Canceler) وجود دارد که با استفاده از یک فیلتر دیجیتال ساده ، سیگنال ICPx نسبت به نویز را بهبود می بخشد به طوری که با اعمال سیگنال به پایه ICPx ، چهار نمونه یکسان از آن گرفته و در صورت برابری آنها تسخیر صورت می پذیرد.

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.

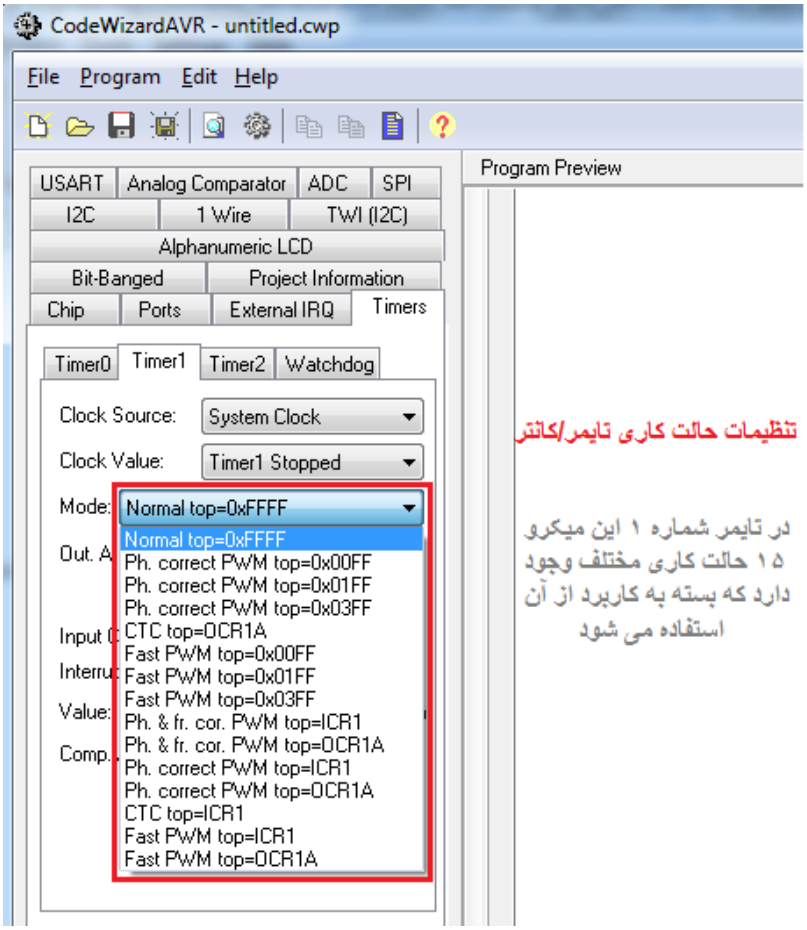
شکل اول : تنظیم حالت کار تایمری یا کانتری



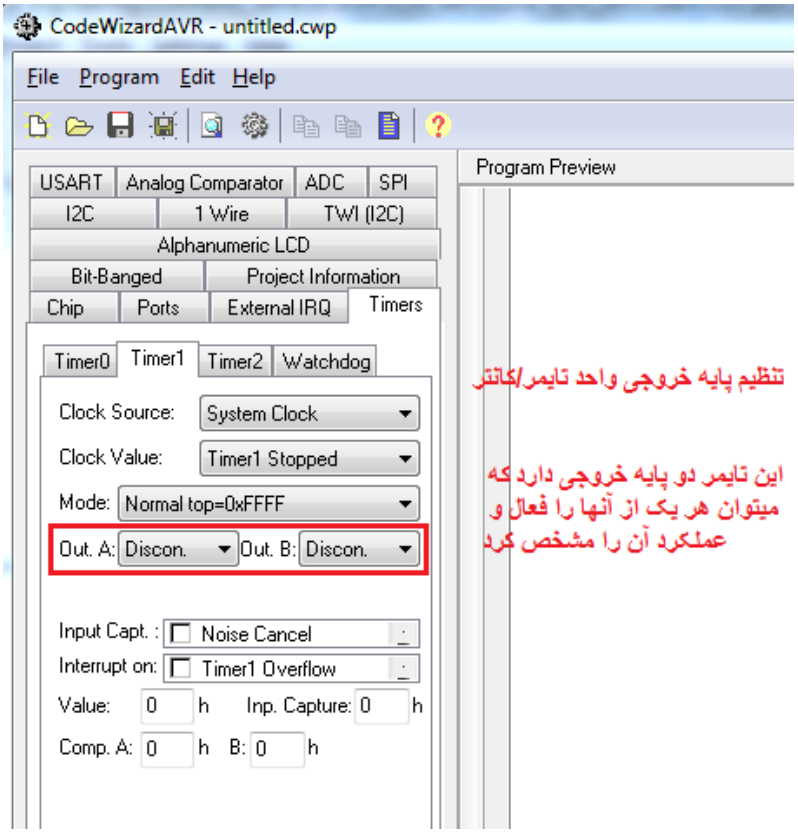
شکل دوم : تنظیم پیش تقسیم کننده در حالت تایمری



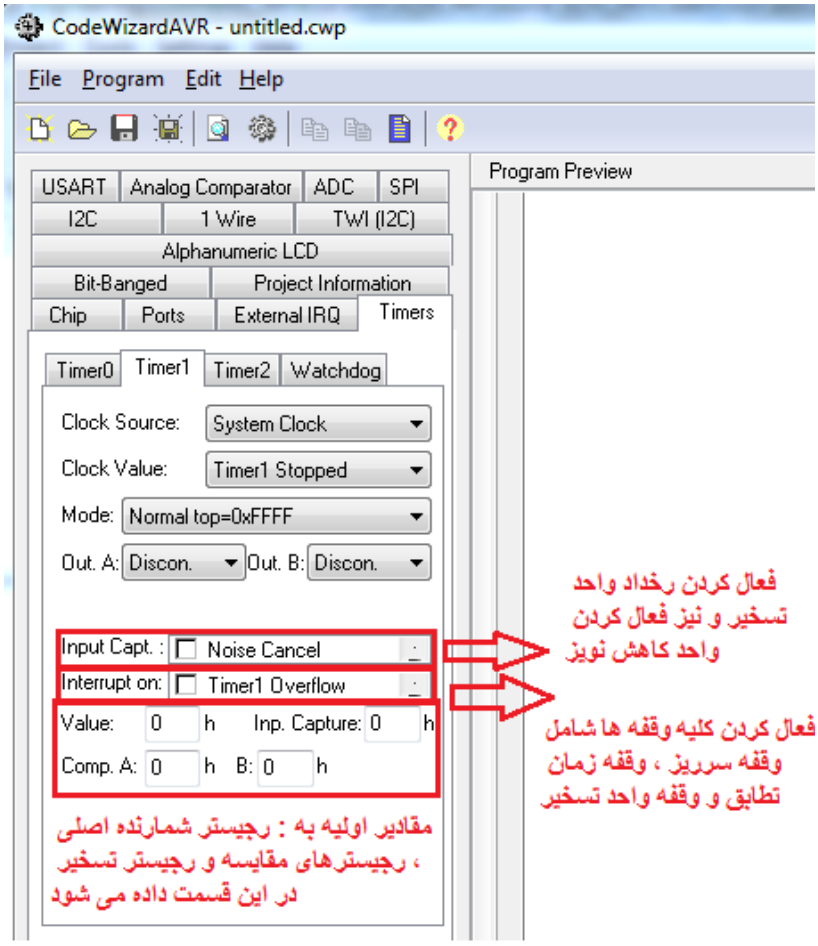
شکل سوم : تنظیم حالت عملکرد تایمر/کانتر



شکل چهارم : تنظیم خروجی



شکل پنجم : تنظیمات نهایی



۸–۸–۲۲– چند مثال شبیه سازی شده

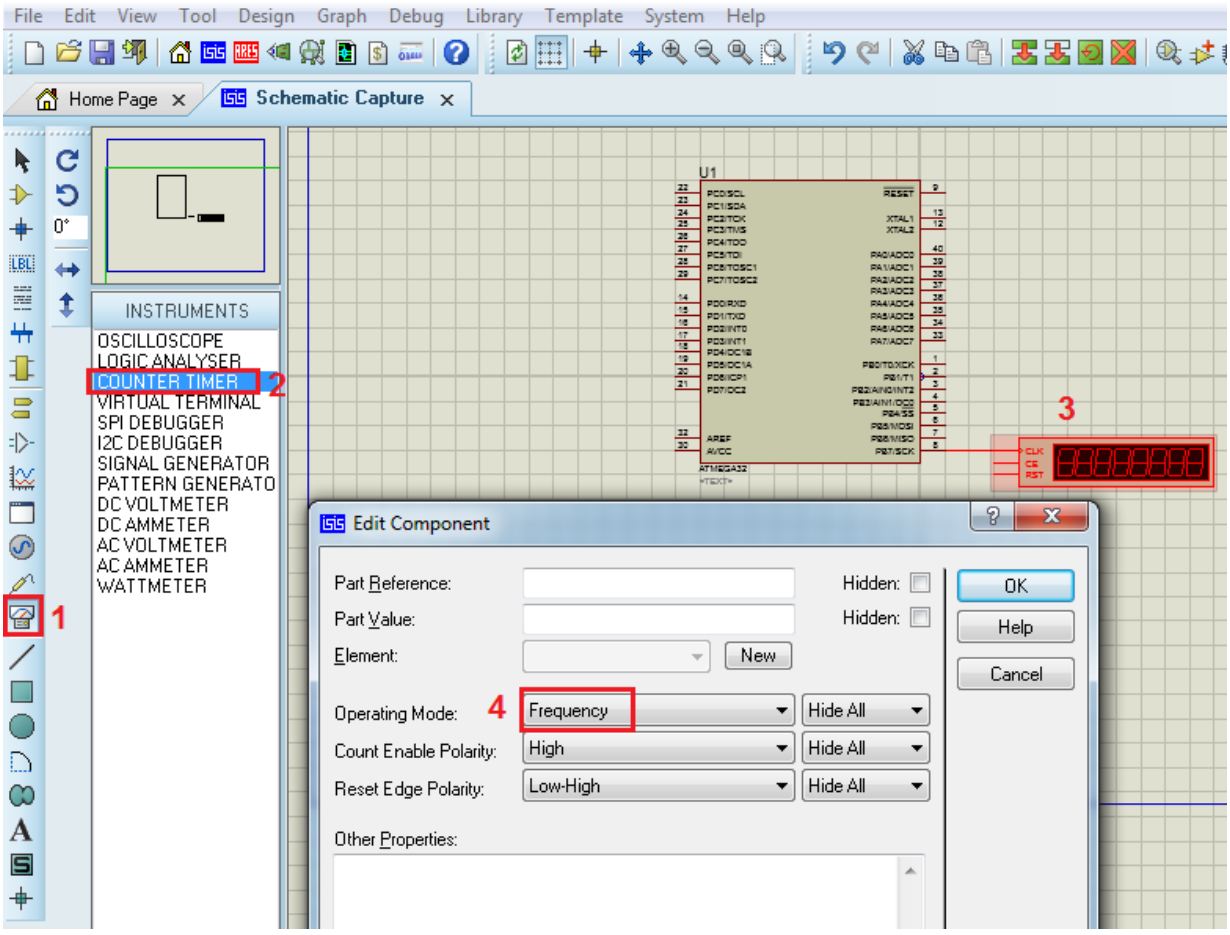
در این بخش با هدف آشنایی با تنظیمات و برنامه نویسی واحد تایمر/کانتر چند مثال ارائه می کنیم.

مثال ۱ : برنامه ای بنویسید که با استفاده از Timer0 میکروکنترلر Atmega32 که کلاک کاری آن 1Mhz است ، فرکانس 100 هرتز را روی پایه PORTB.7 تولید کند. از حالت نرمال Timer0 استفاده کنید.

حل :

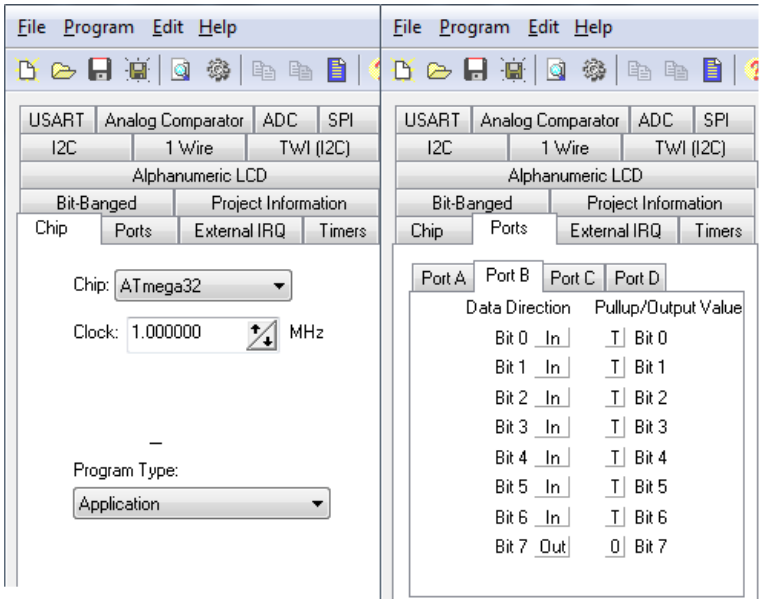
مرحله اول : طراحی سخت افزار در پروتئوس

برای اندازه گیری فرکانس تولید شده ، به یک فرکانس متر روی PORTB.7 احتیاج داریم که آن را از نوار ابزار مطابق شکل زیر به میکرو متصل می کنیم. سپس روی آن دابل کلیک کرده و Operating Mode را روی Frequency قرار می دهیم.



مرحله دوم : تنظیمات کدویزارد

در این مرحله ابتدا تنظیمات چیپ و پورت را مطابق شکل زیر انجام می دهیم.



سپس در سربرگ Timers ، ابتدا تنظیمات اولیه را انجام داده و Mode را روی نرمال قرار می دهیم. سپس یک فرکانس متناسب با فرکانس نهایی خواسته شده (۱۰۰ هرتز) انتخاب می کنیم ، به طوری که زمان سر ریز شدن آن بیشتر از زمان پریود فرکانس نهایی باشد یعنی داشته باشیم : f/256<100

فرکانس 15.625KHz انتخاب می شود چون اولین فرکانسی است که در این رابطه صدق می کند.

حال باید مقدار اولیه TCNT0 را محاسبه کنیم.

فرکانس تایمر در مد نرمال و پریود آن به صورت زیر بدست می آید:

1MHz/64=15.625KHz , T=1/15.625KHz=64us

چون تایمر ۸ بیتی است مدت زمان یکبار سرریز شدن آن برابر است با :

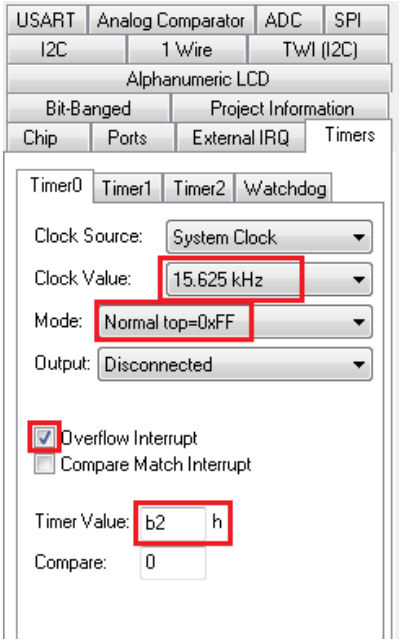
T=64*256us=16384us

فرکانسی که میخواهیم ۱۰۰ هرتز یعنی 0.01 ثانیه (۱۰ میلی ثانیه) است.

با توجه به اینکه دیوتی سایکل 50% باید باشد ، یعنی ۵ میلی ثانیه پالس خروجی ۰ و ۵ میلی ثانیه ۱ است.

در نتیجه مقدار اولیه تایمر به صورت زیر بدست می آید :

16384us-5000us=11384us , 11384/64=177.875≃178 = 0xb2



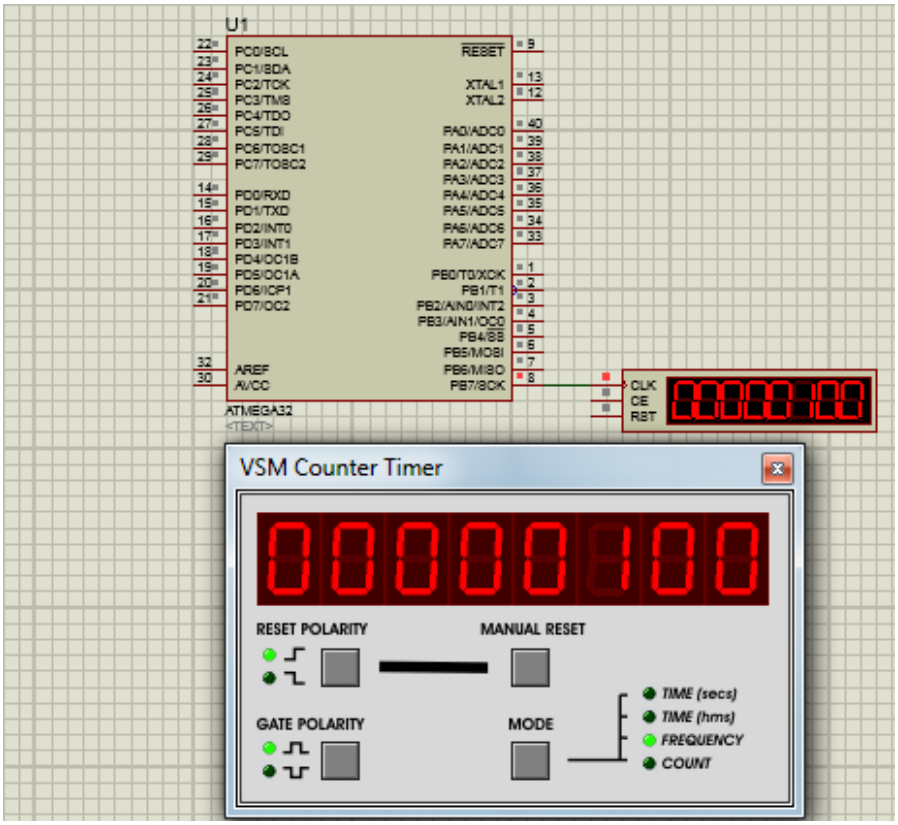
مرحله سوم : تکمیل برنامه

برنامه را به صورت زیر تکمیل می کنیم :

```
#include <mega32.h>

// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Reinitialize Timer 0 value
TCNT0=0xB2;
// Place your code here
PORTB.7=~PORTB.7;
}

// Declare your global variables here

void main(void)
{

// Port B initialization
// Func7=Out Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=0 State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x80;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 15.625 kHz
// Mode: Normal top=0xFF
// OC0 output: Disconnected
TCCR0=0x03;
TCNT0=0xB2;
OCR0=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x01;

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

while (1)
    {
    // Place your code here

    }
}
```

توضیح برنامه : تنها کاری که انجام دادیم یکی پاک کردن خطوط اضافی در برنامه تولید شده است و دیگری اضافه کردن PORTB.7=~PORTB.7; در تابع سابروتین وقفه می باشد. این خط باعث می شود تا در هر بار سرریز شدن تایمر ، مقدار قبلی PB7 ، Not شده و به خروجی برود.

مرحله چهارم : شبیه سازی



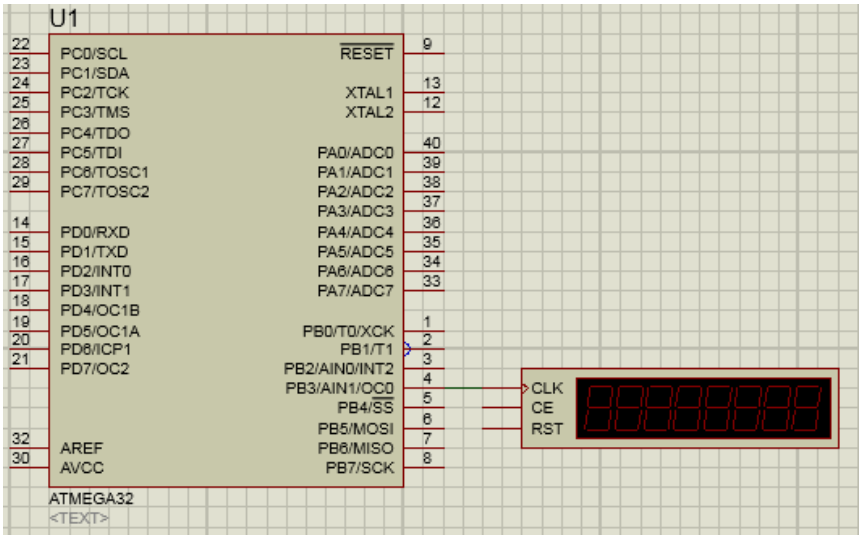
[دانلود سورس مثال ۱]

مثال ۲ : با استفاده از Atmega32 برنامه ای بنویسید که فرکانس 40KHz با زمان وظیفه 50% تولید کند. فرکانس میکروکنترلر را 8MHz درنظر گیرید و از مد CTC تایمر 0 استفاده نمایید.

حل :

مرحله اول : پروتئوس

سوال اشاره به پایه خروجی نکرده است. بهتر است به جای یک پایه دلخواه از خروجی خود تایمر استفاده کنیم. بنابراین فرکانس متر را روی خروجی تایمر ۰ یعنی OC0 قرار می دهیم.



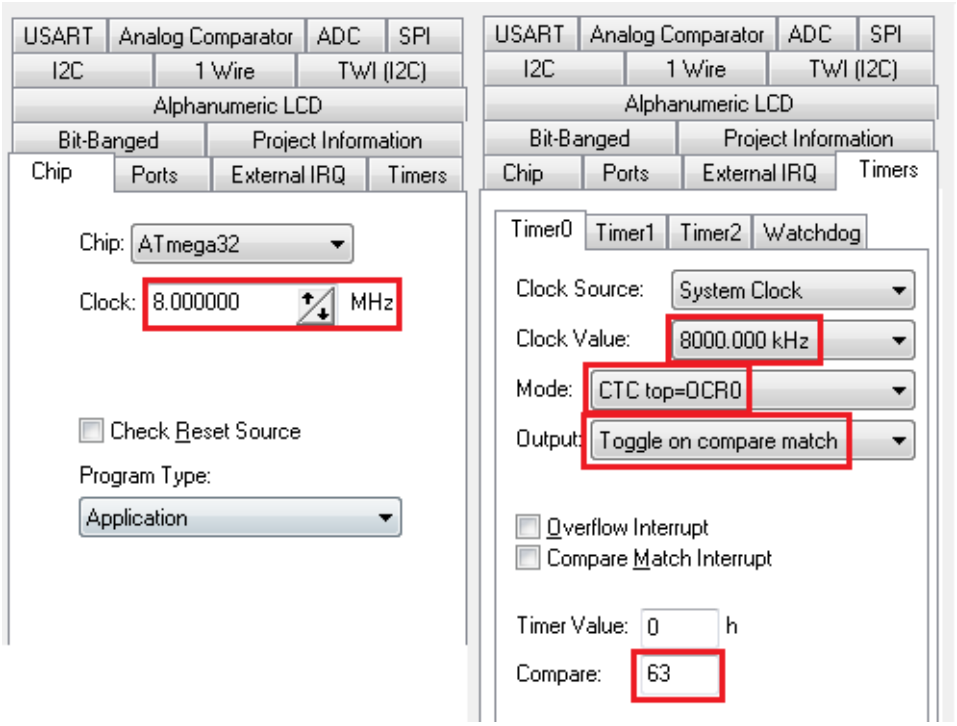
مرحله دوم : کدویزارد

در سربرگ Timer ، فرکانس کلاک واحد تایمر را به علت اینکه 4000KHz/256=15.625KHz<40KHz روی اولین گزینه N=1 قرار می دهیم. حالت عملکرد تایمر را نیز روی CTC قرار می دهیم. حال باید مقدار اولیه OCR0 را معین کنیم. فرکانس خروجی در مد CTC از رابطه زیر بدست می آمد.

$$f_{ocx} = \frac{f_{clk}}{2N(OCRx + 1)}$$

بنابراین از رابطه فوق بدست می آوریم :

$$OCR0 = \frac{8000000}{2 * 1 * 40000} - 1 = 99 = 63hex$$



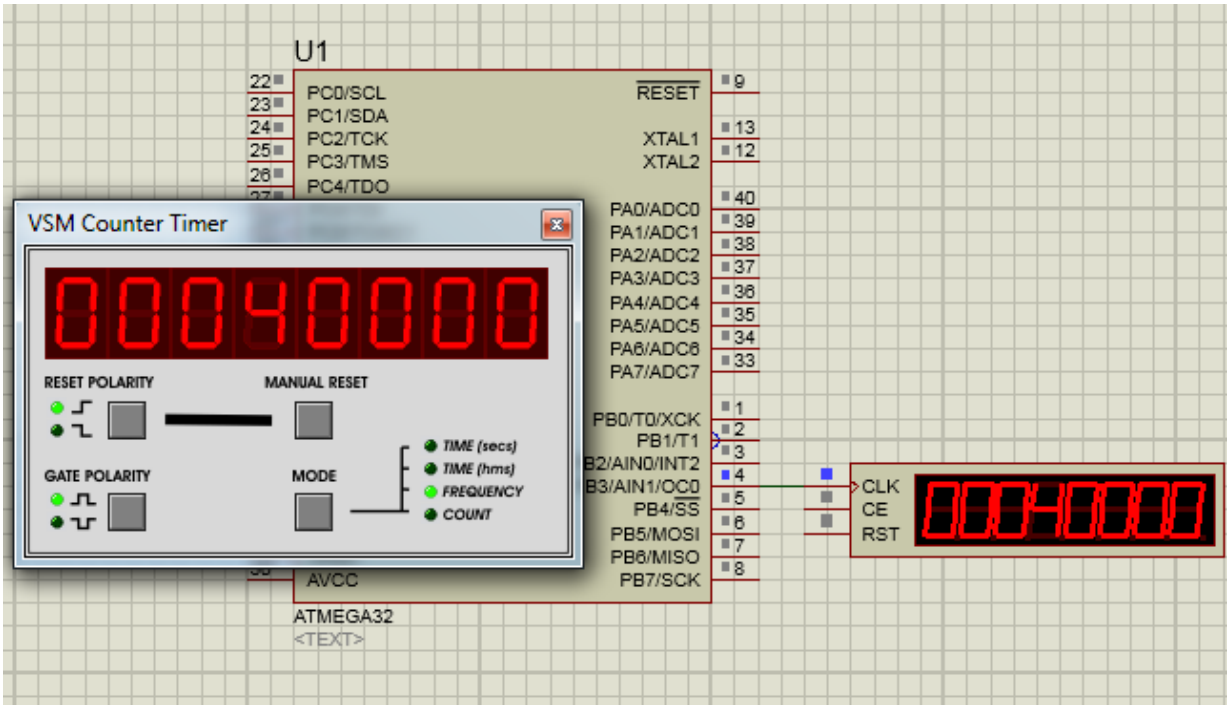
مرحله سوم : تکمیل برنامه

با جز حذف کدهای اضافی ، هیچ کد اضافی دیگری نیاز نیست و برنامه کامل است.

```
#include <mega32.h>

// Declare your global variables here

void main(void)
{
// Declare your local variables here
// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=Out Func2=In Func1=In Func0=In
```

```
// State7=T State6=T State5=T State4=T State3=0 State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x08;


// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 8000.000 kHz
// Mode: CTC top=OCR0
// OC0 output: Toggle on compare match
TCCR0=0x19;
TCNT0=0x00;
OCR0=0x63;


while (1)
    {
    // Place your code here


    }
}
```

مرحله چهارم : شبیه سازی



تذکر : فرکانس میکرو در نرم افزار پروتئوس روی 8Mhz داخلی قرار گرفته است.

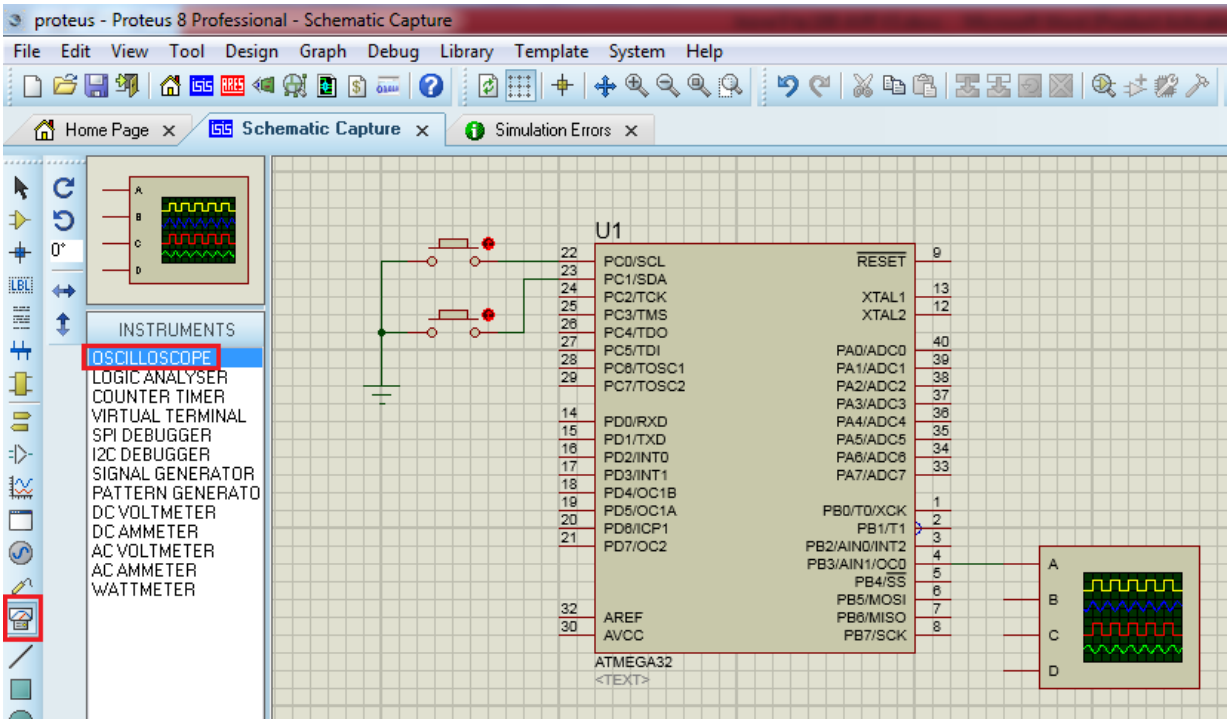
دانلود سورس مثال شماره ۲

مثال ۳ : برنامه ای با استفاده از Atmega32 بنویسید که با استفاده از تایمر شماره ۰ ، موجی با فرکانس ثابت 2KHz و دیوتی سایکل متغیر بین ۱۰ تا ۹۰ درصد تولید کند. فرکانس میکرو را خودتان انتخاب کنید.

حل :

مرحله اول : پروتئوس

دو عدد کلید برای تنظیم PWM به پورت C اضافه کردیم. به جای استفاده از فرکانس متر از Oscilloscope برای دیدن تغییرات استفاده می کنیم.



مرحله دوم : کدویزارد

می خواهیم فرکانس 2KHz داشته باشیم. یعنی پریود 500us را باید ایجاد کنیم. به علت اینکه فرکانس ثابت و دیوتی سایکل متغیر است ، پس در حالت PWM هستیم و میتوان FastPWM یا PhaseCorrect را انتخاب کرد. طبق فرمول FastPWM داریم :

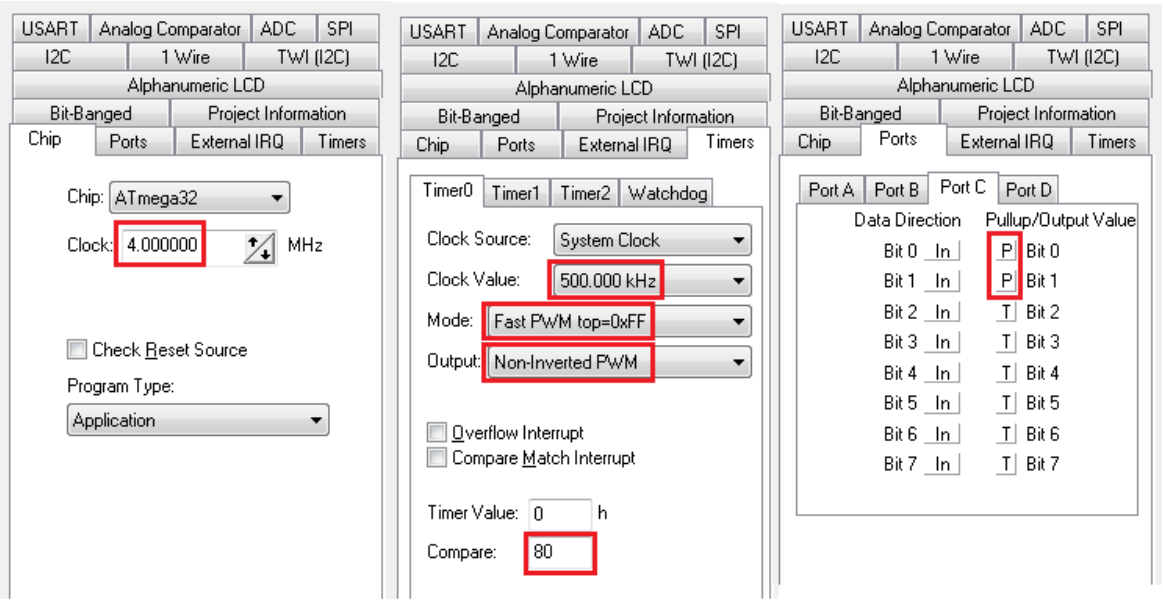
$$f_{ocx} = \frac{f_{clk}}{N \times 256}$$

در نتیجه فرکانس کلاک میکرو به صورت زیر محاسبه می شود :

$$Fclk = N \times 256 \times 2000 = N \times 512000$$

در صورتی که فرض کنیم N=8 ، Fclk=4096000 بدست می آید که تقریبا 4Mhz است. در نتیجه فرکانس کلاک میکرو را روی 4MHz قرار می دهیم و N=8 ضریب تقسیم می شود.

نکته : برای اینکه پهنای PWM در حالت عادی روی 50% باشد ، رجیستر OC0 را دقیقا نصف مقدار دهی می کنیم. بنابراین 128 یا 0x80 می شود.



مرحله سوم : تکمیل برنامه

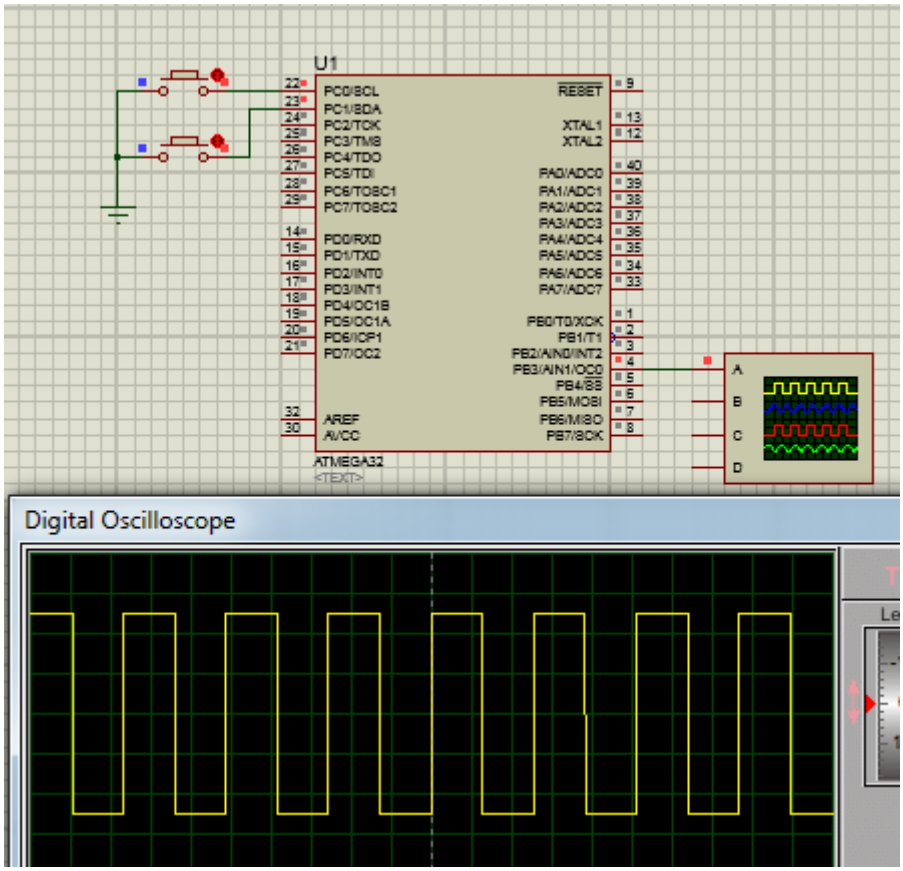
در این مرحله فقط برای کلید برنامه را در حلقه while کامل می کنیم. نیاز به اضافه کردن delay.h برای تاخیر کلید می باشد. برنامه نهایی به صورت زیر است :

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

void main(void)
{
// Port B initialization
// Func7=In Func6=In Func5=In Func4=In Func3=Out Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=0 State2=T State1=T State0=T
PORTB=0x00;
DDRB=0x08;

// Port C initialization
// Func7=In Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=T State6=T State5=T State4=T State3=T State2=T State1=P State0=P
PORTC=0x03;
DDRC=0x00;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 500.000 kHz
// Mode: Fast PWM top=0xFF
// OC0 output: Non-Inverted PWM
TCCR0=0x6A;
TCNT0=0x00;
OCR0=0x80;

while (1)
    {
```

```
if(PINC.0==0)
      {
       OCR0++;
       if(OCR0>250) OCR0=250;
       delay_ms(20);
      }
      if(PINC.1==0)
      {
       OCR0--;
       if(OCR0<10) OCR0=10;
       delay_ms(20);
      }
      }
}
```

توضیح برنامه : در حلقه while تنها دو حلقه شرطی if وجود دارد که فقط در صورت زدن یکی از کلیدها وارد آن می شود و یک واحد به مقدار کنونی OCR0 اضافه یا کم می کند. سپس توسط یک حلقه if دیگر اجازه بیشتر یا کمتر شدن رجیستر OCR0 از مقادیر ۲۵۰ و ۱۰ را نمی دهد. تاخیر هم برای رفع Bounce کلید ضروری است.

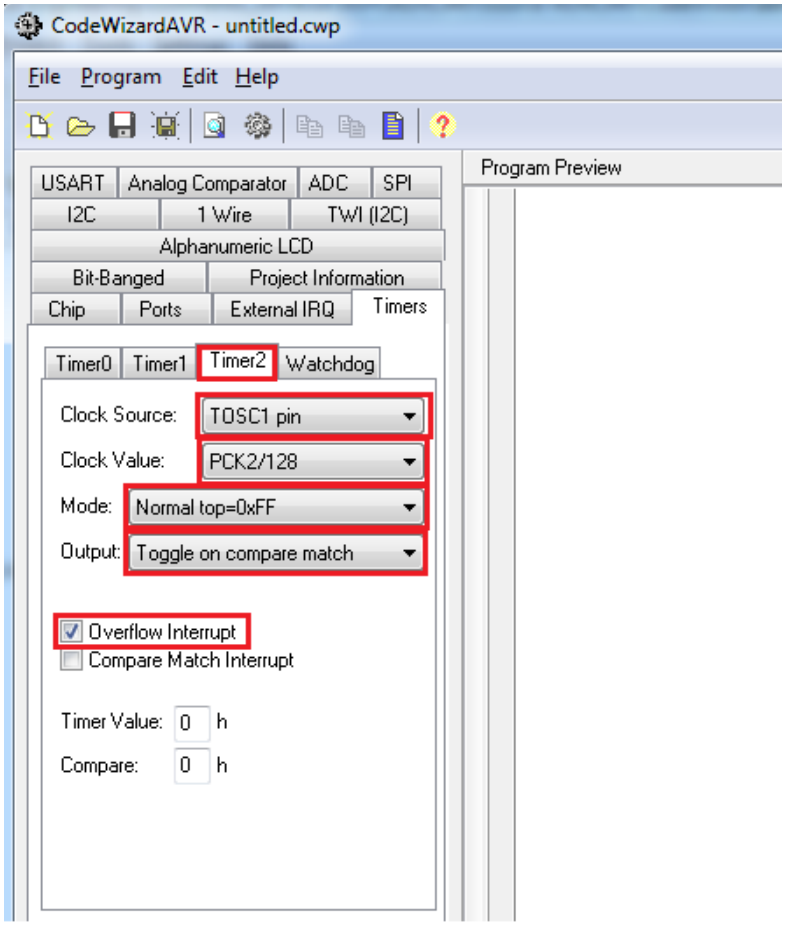
مرحله چهارم : شبیه سازی



دانلود سورس مثال ۳

۸–۸–۲۳– راه اندازی RTC در میکروکنترلرهای AVR

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 TOSC1 و 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 PCK2/128 و مد نرمال به این هدف رسید . ضمنا میتوان با فعال کردن خروجی و قرار دادن آن روی Toggle و قراردادن یک LED روی پایه OC2آن را هر ثانیه یکبار روشن و خاموش نمود . شکل زیر فعالسازی این ویژگی در Atmega32را 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.

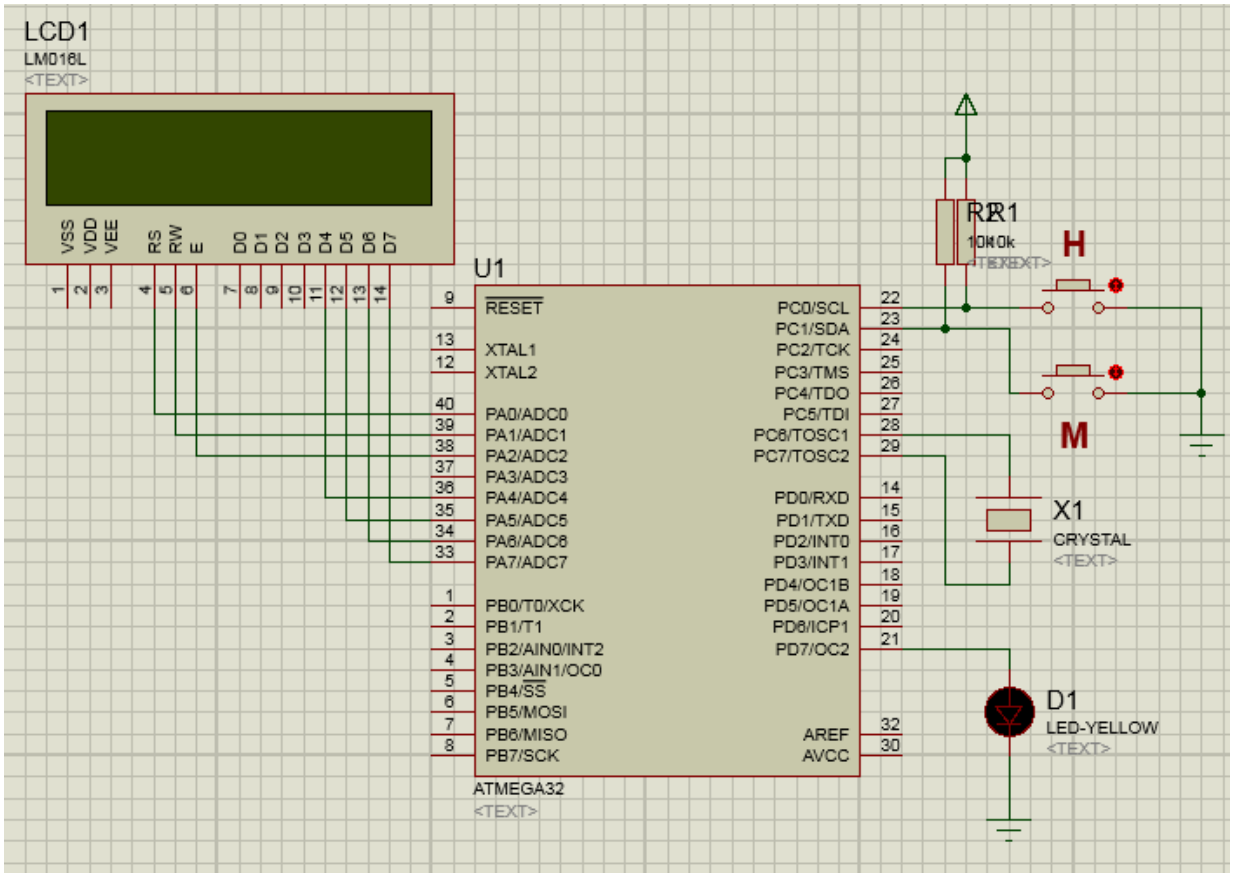


مثال ۴ : با استفاده از Atmega32 و LCD کاراکتری ، یک ساعت دیجیتال بسازید که دارای دو کلید تنظیم باشد. یک LED هم برای یک ثانیه در نظر بگیرید.

حل :

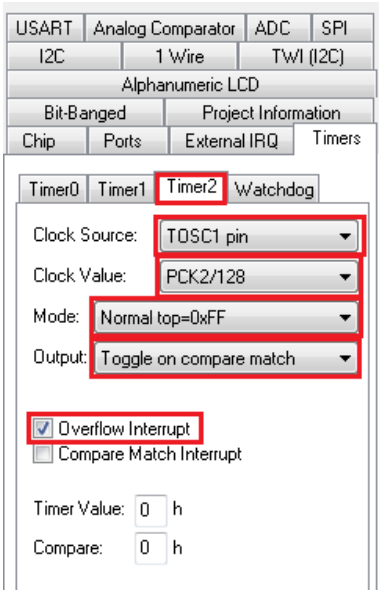
مرحله اول : پروتئوس

یک کریستال با مقدار 32768 بین پایه های TOSC1 و TOSC2 قرار می دهیم. یک LED روی پایه OC2 و دو عدد کلید به همراه یک LCD کاراکتری به میکرو متصل می کنیم.



مرحله دوم : کدویزارد

تنظیمات سربرگ های chip و LCD میکرو را طبق معمول انجام می دهیم. تنظیمات سربرگ Timer دقیقا به همان صورت گفته شده است.



مرحله سوم : تکمیل برنامه

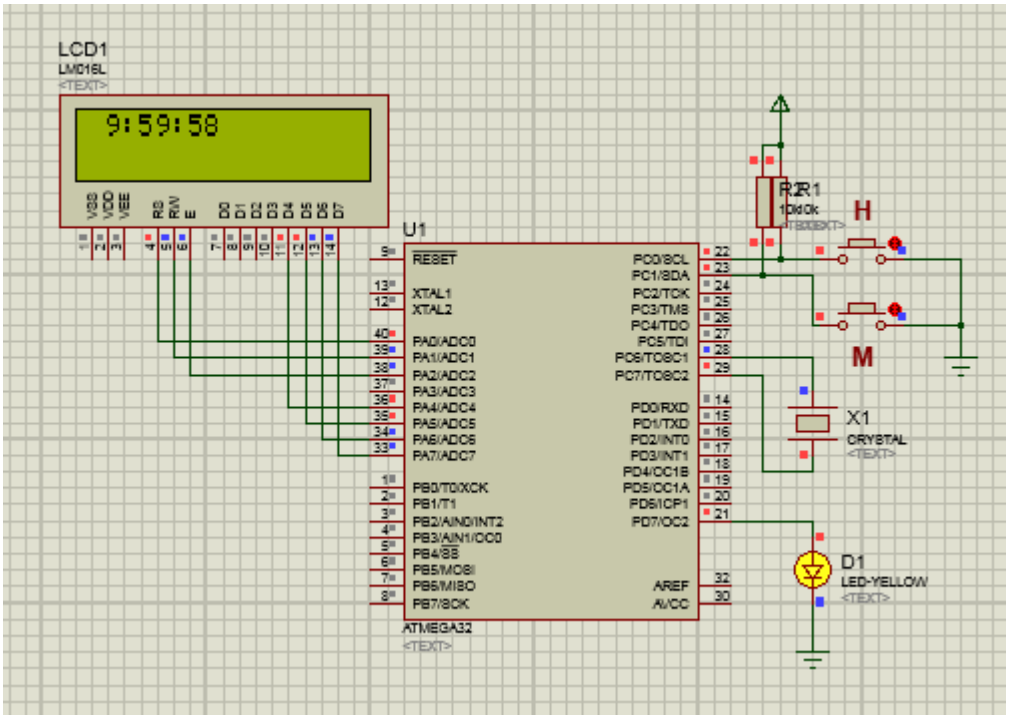
با تولید شدن کد اولیه مشاهده می شود که یک تابع سابروتین وقفه ایجاد شده است که در هرثانیه یکبار اجرا می شود. در برنامه سه متغیر s ، m و h به ترتیب برای ثانیه ، دقیقه و ساعت تعریف می کنیم. درون تابع سابروتین وقفه ثانیه را یک واحد افزایش می دهیم. و درون حلقه while کدهای اجرای کلید و نمایش روی lcd را قرار می دهیم.

```
#include <mega32.h>
#include <stdio.h>
// Alphanumeric LCD Module functions
#include <alcd.h>
#include <delay.h>
unsigned char s,m,h;
char str[10];

// Timer2 overflow interrupt service routine
interrupt [TIM2_OVF] void timer2_ovf_isr(void)
{
s++;
}

// Declare your global variables here

void main(void)
{

// Port D initialization
// Func7=Out Func6=In Func5=In Func4=In Func3=In Func2=In Func1=In Func0=In
// State7=0 State6=T State5=T State4=T State3=T State2=T State1=T State0=T
PORTD=0x00;
DDRD=0x80;



// Timer/Counter 2 initialization
// Clock source: TOSC1 pin
// Clock value: PCK2/128
// Mode: Normal top=0xFF
// OC2 output: Toggle on compare match
ASSR=0x08;
TCCR2=0x15;
TCNT2=0x00;
OCR2=0x00;

// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x40;

// Alphanumeric LCD initialization
```

```
// Connections specified in the
// Project|Configure|C Compiler|Libraries|Alphanumeric LCD menu:
// RS - PORTA Bit 0
// RD - PORTA Bit 1
// EN - PORTA Bit 2
// D4 - PORTA Bit 4
// D5 - PORTA Bit 5
// D6 - PORTA Bit 6
// D7 - PORTA Bit 7
// Characters/line: 16
lcd_init(16);

// Global enable interrupts
#asm("sei")
lcd_clear();
lcd_gotoxy(0,0);
s=45;
h=9;
m=59;
while (1)
     {
      if(PINC.0==0) { h++; delay_ms(200);}
      if(PINC.1==0) { m++; delay_ms(200);}


      if(s>=60){ s=0; m++;}
      if(m>=60){ m=0; h++;}
      if(h>=24)  h=0;


      sprintf(str,"%2d:%2d:%2d",h,m,s);
      lcd_gotoxy(0,0);
      lcd_puts(str);
     }

}
```

توضیح برنامه : در حلقه while برای برنامه کلید ، از کلید نوع یک استفاده کردیم. سپس محدوده مجاز برای متغیر های ثانیه ، دقیقه و ساعت را توسط if تعیین کردیم. برای ثانیه و دقیقه به محض بالاتر رفتن از 59 یک واحد به متغیر بالاتر می دهند و خودشان صفر می شوند. برای نمایش روی LCD نیز ابتدا با استفاده از تابع sprintf اعداد ثانیه ، دقیقه و ساعت را به یک آرایه منتقل می کنیم و سپس نمایش می دهیم.

مرحله چهارم : شبیه سازی



دانلود سورس مثال ۴

۸-۸-۲۴- 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 Enable می 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

فصل ۹– برنامه نویسی پیشرفته

مقدمه

با توجه به استفاده گسترده از زبان برنامه نویسی C در انواع میکروکنترلرها به خصوص AVR، به عنوان منعطف ترین زبان برنامه نویسی و لزوم درک این زبان پرکاربرد، شاهد عدم استفاده از تمام قابلیت های این زبان در پروژه های مربوطه هستیم. در این بخش سعی شده است تا این قابلیت های برنامه نویسی پیشرفته تشریح و در اختیار همگان قرار گیرد و نکاتی که در برنامه نویسی پیشرفته باید رعایت شود تا سیستم روان ، بدون هنگ و بهینه باشد.

۹–۱– انواع دستورات پیش‌پردازش

یکی از امکانات زبان C فرمان‌های پیش ترجمه یا پیش پردازش است. همانطور که از عنوان آن مشخص است این فرمان‌ها قبل از شروع ترجمه‌ی برنامه و در یک مرحله‌ی مقدماتی بررسی‌شده، نتیجه آن روی متن برنامه اعمال می‌گردد .استفاده از این فرمان‌ها ازیک‌طرف باعث سهولت برنامه‌نویسی شده، از طرف دیگر باعث بالا رفتن قابلیت اصلاح و جابه‌جایی برنامه می‌گردد. حال فرض کنید که یک ثابت در چندین جای برنامه‌ی شما ظاهر شود، استفاده از یک نام نمادین برای ثابت ایده‌ی خوبی به نظر می‌رسد. سپس می‌توان به‌جای قرار دادن ثابت در طول برنامه از نام نمادین استفاده کرد.

درصورتی‌که لازم باشد تا مقدار ثابت را تغییر دهیم، بدون استفاده از نام نمادین، باید همه جای برنامه را با دقت برای یافتن و جایگزین کردن ثابت نگاه کنیم، آیا می‌توانیم این کار را در C انجام دهیم؟!

C قابلیت ویژه‌ای به نام پیش پردازشگر دارد که امکان تعریف و مرتبط ساختن نام‌های نمادین را با ثابت‌ها فراهم می‌آورد. پیش پردازشگر C پیش از کامپایل شدن اجرا می‌شود.

پیش پردازنده نوعی مترجم است که دستورات توسعه‌یافته‌ای از یک‌زبان را به دستورات قابل‌فهم برای کامپایلر همان زبان تبدیل می‌کند. قبل از اینکه برنامه کامپایل شود، پیش پردازنده اجرا می‌شود و دستورات را که با نماد # شروع‌شده‌اند را ترجمه می‌کند. سپس کامپایلر برنامه را کامپایل می‌کند

دستورات پیش پردازنده در زبان سی شامل موارد زیر هستند:

دستور پیش پردازشی	توضیحات
#include	اضافه کردن فایل کتابخانه
#define	تعریف یک ماکرو
#undef	حذف یک ماکرو
#ifdef	اگر یک ماکرو تعریف شده است آنگاه
#ifndef	اگر یک ماکرو تعریف نشده است آنگاه
#if	اگر شرط برقرار بود آنگاه
#endif	انتهای بلوکif
#else	در غیر این صورت
#error	ایجاد یک خطا در روند کامپایل برنامه
#pragma	تغییر رفتا کامپایلر

نکته : پیش پردازشگر C مبتنی برخط است. یعنی هر دستور ماکرو با یک کاراکتر خط جدید به پایان می‌رسد، نه با سمیکالن.

۹-۱-۱- پیش پردازنده define

دستور defineرایج‌ترین دستور پیش پردازشی است که به آن ماکرو می گویند. این دستور هر مورد از رشته کاراکتری خاصی (که نام ماکرو هست) را با مقدار مشخص‌شده‌ای (که بدنه ماکرو هست) جایگزین می کند. نام ماکرو همانند نام یک متغیر در C است که باید با رشته تعریف‌کننده ماکرو حداقل یک فاصله داشته باشد و بهتر است جهت مشخص بودن در برنامه با حروف بزرگ نمایش داده شود .به‌عنوان‌مثال دستور #define MAX 20موجب می‌شود تا مقدار ماکروی MAX در سرتاسر برنامه برابر با ۲۰ فرض شود؛ یعنی در هر جای برنامه از ماکروی MAX استفاده شود مثل این است که از عدد ۲۰ استفاده‌شده است.

کاربرد دیگر دستورdefine در تعریف ماکروهایی است که دارای پارامتر باشند. این مورد به‌صورت زیر استفاده می‌شود:

#define <نام ماکرو> (تعریف ماکرو)اسامی پارامترها>

تعریف ماکرو مشخص می‌کند که چه عملی باید توسط ماکرو انجام گیرد. اسامی پارامترها متغیرهایی هستند که در حین اجرای ماکرو به آن منتقل می‌شوند که اگر تعداد آن‌ها بیشتر از یکی باشد، با کاما از هم جدا می شوند. از دستور undefجهت حذف ماکرویی که پیش‌تر تعریف‌شده است استفاده می‌کنیم. روش استفاده از این دستور به‌صورت زیر است:

#undef <نام ماکرو>

در اینجا نام ماکرو شناسه‌ای است که پیش‌تر توسط دستورdefine تعریف‌شده است.

۹-۱-۲- پیش پردازنده include

ضمیمه کردن فایل‌ها توسط دستور پیش پردازنده #include انجام می‌گیرد. این دستور به‌صورت های زیر مورد استفاده قرار می‌گیرد:

#include "نام فایل"

#include <نام فایل>

فایل‌های سرآیند به دودسته تقسیم می‌شوند:

۱. هدر فایل‌هایی که همراه کامپایلر C وجود دارند و پسوند همه‌ی آن‌ها h. است.

۲. هدر فایل‌هایی که توسط برنامه‌نویس نوشته می‌شوند.

روش اول دستور #include برای ضمیمه کردن فایل‌هایی استفاده می‌شود که توسط برنامه‌نویس نوشته‌شده‌اند و روش دوم برای ضمیمه فایل‌هایی استفاده می‌شوند که همراه کامپایلر وجود دارند. مثال:

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

فایل‌های سرآیند از اهمیت ویژه‌ای برخوردارند، زیرا:

۱. بسیاری از توابع مثلgetchar وputchar در فایل‌های سرآمد مربوط به سیستم، به‌صورت ماکرو تعریف‌شده‌اند.

۲. با فایل‌های سرآیندی که برنامه‌نویس می‌نویسد، علاوه بر تعریف ماکروها، می‌توان از بسیاری از تعاریف تکراری جلوگیری کرد.

۹-۱-۳- دستورات پیش‌پردازش شرطی

در حالت معمولی، دستور if برای تصمیم‌گیری در نقاط مختلف به کار می‌رود .شرط‌هایی که در دستور if ذکر می‌شوند در حین اجرای برنامه ارزشیابی می‌شوند؛ یعنی اگر شرط ذکرشده در دستور if درست باشد یا نادرست، این دستور و کلیه دستورات دیگر که در بلاک if قرار دارند ترجمه می‌شوند ولی در دستورات پیش پردازنده شرطی، شرطی که در آن ذکر می‌شود در حین ترجمه ارزشیابی می‌شود. دستورات پیش پردازنده شرطی عبارت‌اند از: #if,#else,#endif, #ifdef, #ifndef

دستور if به‌صورت زیر به کار می‌رود:

```
#if عبارت شرطی
مجموعه ی دستورات۱
#else
مجموعه دستورات۲
#endif
```

در این دستور در صورتی که عبارت شرطی بعد از if برقرار باشد ، مجموعه دستورات ۱ و در غیر این صورت مجموعه دستورات ۲ کامپایل می شود.

برخلاف دستور if درC ، حکم‌های تحت کنترل دستور #ifdef در آکولادها محصور نمی‌گردند. به‌جای آکولادها برای پایان بخشیدن به بلوک #ifdef باید از دستور #endif استفاده شود.

```
#ifdef   DEBUG
/* Your debugging statements here */
#endif
```

دستور #ifndef#عکس دستور #ifdef عمل می‌کند. اگر ماکرویی که نام آن در جلوی #ifndefقرار دارد در یک دستور #define تعریف‌نشده باشد، مجموعه حکم‌های ذکرشده کامپایل می‌گردند، در غیر این صورت کامپایل نخواهند شد. قالب کلی استفاده شبیه #ifdef است:

```
#ifdef <نام ماکرو>
مجموعه ی دستورات
.
.
.
#endif
```

دستور #ifndef هم برای پایان به دستور #endifنیاز دارد.

```
#ifndef MESSAGE
#define MESSAGE "You wish!"
#endif
```

دستور #error موجب جلوگیری از ادامه ترجمه‌ی برنامه‌ی توسط کامپایلر شده، به‌صورت زیر به کار می‌رود :

```
#error   پیام خطا
```

پیام خطا، جمله‌ای است که کامپایلر پس از رسیدن به این دستور، آن را به‌صورت زیر در صفحه‌نمایش ظاهر می‌کند:

```
error: نام فایل   شماره خط
```

۹-۱-۴- پیش پردازنده pragma

این دستور به کامپایلر این امکان را می دهد که ماکروهای مورد نظر را بدون مداخله با کامپایلرهای دیگر تولید کند. به صورت کلی از این ماکرو به صورت زیر استفاده می شود:

```
#pragma name
```

که در آن name میتواند یکی از حالت های زیر باشد:

۱. : -#pragma warnغیرفعال کردن اخطارهای صادر شده از کامپایلر

۲. : +pragma warn#فعال کردن اخطارهای صادر شده از کامپایلر
۳. : -pragma opt#بهینه کننده کد توسط کامپایلر را غیر فعال می کند
۴. : +pragma opt#بهینه کننده کد توسط کامپایلر را فعال می کند
۵. : -pragma optsize#بهینه کننده برنامه نسبت به سرعت
۶. : +pragma optsize#بهینه کننده برنامه نسبت به حجم

نکته : معادل دستور optsize را میتوانید از طریق نرم افزار کدویژن در آدرس زیر تنظیم نمایید:
Project/Configuration/C Compiler/Codevision/Optimized for

۷. : +pragma savereg#این دستور هنگامی که وقفه ای رخ دهد ، میتواند رجیسترهای R0,R1,R22,R23,R24,R25,R26,R27,R30,R31و SREG را ذخیره نماید.
۸. : -pragma savereg#این دستور مخالف دستور قبلی است و رجیسترها را پاک می کند.
۹. : +pragma regalloc#این دستور متغیرهای سراسری را به رجیسترها اختصاص می دهد.(معادل کلمه کلیدی register)
۱۰. : -pragma regalloc#این دستور برخلاف دستور قبلی متغیر سراسری را در حافظه SRAM تعریف می کند.

نکته : معادل دستور regalloc را میتوانید از طریق نرم افزار کدویژن در آدرس زیر تنظیم کنید:
Project/Configuration/C Compiler/Code Generation/Automatic Register Allocation

۱۱. : +pragma promoteacher#این دستور متغیرهای char را به int تبدیل می کند.
۱۲. : -pragma promoteacher#این دستور متغیر های int را به char تبدیل می کند.

نکته : معادل دستور promoteacher را میتوانید از طریق نرم افزار کدویژن در آدرس زیر تنظیم کنید:
Project/Configuration/C Compiler/Code Generation/Promote char to int

۱۳. : +pragma unchar#این دستور نوع داده char را به unsigned char تبدیل می کند.
۱۴. : -pragma unchar#این دستور نوع داده char را بدون تغییر و به همان صورت signed char تعریف می کند.

نکته : معادل دستور unchar را میتوانید از طریق نرم افزار کدویژن در آدرس زیر تنظیم کنید:
Project/Configuration/C Compiler/Code Generation/Char is unsigned

۱۵. pragma library#این دستور یک فایل کتابخانه ای با پسوند lib. را به برنامه پیوند میزند.

۹-۲- نحوه ساخت فایل های کتابخانه

تقسیم برنامه های بزرگ به واحد های کوچکتر یا اصطلاحا ماژولار کردن برنامه، از جهات مختلفی، بسیار سودمند است. به خوانایی برنامه کمک زیادی می کند .برنامه می تواند توسط چندین نفر نوشته شود ، تغییرات در برنامه به سهولت انجام می شود و از هر ماژول در پروژه های دیگر میتوان بهره گرفت.

هر فایل کتابخانه شامل دو فایل است:

۱. **فایل هدر (header)** : این فایل که با پسوند h. است حاوی الگوی توابع و پیش پردازنده ها (ماکرو) است.

۲. **فایل سورس (source)** : این فایل که با پسوند c. است حاوی بدنه توابعی است که الگوی آن در هدر فایل تعریف شده است.

این دو فایل در کنار هم درون پوشه پروژه اصلی قرار می گیرند. هدر فایل هایی که تنها شامل عملگرهای پیش پردازنده هستند ، نیازی به فایل سورس ندارند. در هر کامپایلر یا با هر ویرایشگر متنی (مثل Notepad) میتوان این دو فایل را ایجاد کرد و با پسوند مربوطه ذخیره کرد.

۹-۲-۱- نحوه استفاده از کتابخانه در برنامه

برای استفاده از کتابخانه ای که خود ساخته ایم ابتدا باید هدر فایل آن را بوسیله " به برنامه اضافه کنیم. به صورت زیر:

#include “headerfile.h”

بعد از اضافه کردن هدر فایل میتوانیم از ثوابت و توابعی که درون کتابخانه تعریف کرده ایم در برنامه اصلی استفاده کنیم.

۹-۲-۲- الگوی ساخت فایل هدر

برای ساخت هدر فایل با پسوند h. باید الگویی را رعایت نمود. برای مثال می خواهیم یک کتابخانه برای اتصال keypad به میکرو ایجاد کنیم. برای این کار ابتدا یک فایل با پسوند h. ساخته و سپس درون آن به صورت الگوی زیر می نویسیم
:

```
#ifndef _KEYPAD_H
#define _KEYPAD_H
#include<headerfiles.h>
#define …
…
void Functions(void);
…
#endif
```

همانطور که مشاهده می کنید الگوی نوشتن هدر فایل به این صورت است که در ابتدا با استفاده از ماکروی ifndef/endif و سپس نوشتن نام هدر فایل با حروف بزرگ و دقیقا به همان الگوی مثال زده شده (یک "_" در ابتدا و یک "_" به جای نقطه) و استفاده از ماکروی define# یک ماکرو جدید با نام KEYPAD_H_

تعریف کردیم .این گونه نوشتن را محافظت از برنامه یا Header Guard گویند. هدرگارد باعث میش شود تا ماکروها و توابع تعریف شده فقط و فقط یکبار تعریف شده باشند (جلوگیری از تعریف آنها با نام یکسان).
سپس اگر هدر فایل به کتابخانه های دیگری نیاز دارد ، آنها را با include# اضافه می کنیم. بعد از آن تعریف ثوابت را با استفاده از define# انجام می دهیم. اگر تغیر نوع در متغیر ها وجود دارد آنها را بعد از ثوابت typedefمی کنیم. در پایان الگوی تعریف توابعی که میخواهیم آنها را در فایل سورس تشریح کنیم را باید در این قسمت بیاوریم.

۹-۲-۳- الگوی ساخت فایل سورس

فایل سورس نیز دارای الگویی است که باید رعایت شود. برای ساخت فایل سورس برای keypad.h به صورت زیر عمل می کنیم:

```
#include “keypad.h”
void Functions(void){
بدنه تابع
}
…
```

همانطور که مشاهده می کنید ، در ابتدای هر فایل سورس ، فایل هدر include می شود و در خطوط بعدی تنها بدنه توابع طبق قوانین مربوط به توابع آورده می شود.

۹-۳- نوع داده sfrb و sfrw در کدویژن

در کامپایلر کدویژن این دو نوع داده ای برای دستیابی به رجیسترهای I/O موجود در حافظه SRAM میکرو اضافه شده است. در حقیقت با تعریف این دو نوع داده در هدر فایل میکروکنترلر (برای مثال هدر فایل mega32.h) قابلیت دسترسی راحت بیتی به رجیسترهای I/O برای کاربر فراهم شده است. بنابراین اگر فایل h.هر میکرویی را باز نمایید ، درون آن این نوع داده ای را مشاهده می کنید. نحوه تعریف آن به صورت زیر است
:

```
sfrb   نام رجیستر=آدرس رجیستر ;
sfrw   نام رجیستر=آدرس رجیستر ;
```

در سمت راست تساوی آدرس رجیستر مورد نظر در حافظه SRAM و در سمت چپ تساوی نام دلخواهی را وارد می کنیم. مثال:

```
sfrb PORTA=0x1b;
sfrb DDRA=0x18;
...
```

همه این تعاریف مربوط به پورت ها و رجیسترهای میکروکنترلرهای AVR در هدر فایل هر یک موجود است که با include کردن آن به برنامه این رجیسترها اضافه می شود و نیازی به تعریف مجدد در برنامه نیست. بنابراین تنها آنچه که در برنامه اصلی از آن استفاده می شود ، استفاده از عملگر نقطه (dot) برای دسترسی بیتی به رجیسترهاست که به صورت زیر است:

; ...= مقدار n . نام رجیستر

که در آن n برای رجیسترهای تعریف شده با sfrb بین ۰ تا ۸ و برای sfrw بین ۰ تا ۱۵ است.

نکته ۱ : آدرس رجیسترهای حافظه SRAM برای هر میکروکنترلری در انتهای دیتاشیت آن آورده شده است.

نکته ۲ : تفاوت بین sfrb,sfrw در این است که از sfrb برای دسترسی بیتی به رجیسترهای ۸ بیتی و از sfrw برای دسترسی بیتی به رجیسترهای ۱۶ بیتی استفاده می شود.

نکته ۳ : در معماری میکروکنترلرهای AVR تنها به رجیسترهایی که در آدرس 0 تا 1FH قرار دارند میتوان دسترسی بیتی داشت. بنابراین این محدودیت برای دستور sfrb و sfrw نیز برقرار خواهد بود.

۹-۴- 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)

۹-۴-۱- 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) ، static (برای کلاس حافظه استاتیک) ، register (برای کلاس حافظه ثبات) و extern (برای 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;
```

۹-۴-۲- 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:

۱. به صورت محلی (Local) هستند . یعنی در داخل بلاکی که تعریف شده اند، قابل دسترسی اند.
۲. 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.
۳. چندین بار می‌توانند مقدار اولیه بگیرند .

۹-۴-۳- 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 CPU به 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:

۱. 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.
۲. کلاس حافظه ثبات تنها می‌تواند برای متغیرهای محلی و همچنین پارامترهای تابع به کار گرفته شود .

۳. 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.
۴. 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.

مثال:

```
register char a;
```

نکته ۱ : فقط متغیرهایی از جنس int و char را میتوان از نوع کلاس ذخیره سازی ثبات معرفی کرد.

نکته ۲ : با استفاده از دستور پیش پردازنده زیر هم میتوان کلاس ثبات را تعریف کرد. مثال:

```
#pragma regalloc+
char a;
```

نکته ۳ : کامپایلر ممکن است به طور اتوماتیک یک متغیر را برای افزایش سرعت اجرای برنامه از نوع رجیستر تشخیص دهد حتی اگر از دستورات فوق برای آن متغیر استفاده نشده باشد. برای جلوگیری از چنین حالتی از کلمه کلیدی volatile در قبل از تعریف متغیر استفاده می شود.

۹-۴-۴- 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 extern هستند، 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 externبه 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 .به مثال زیر توجه کنید:

```
static int x,y;
int m,n;
void main(void)
{…
}
```

متغیرهای x,yدر این مثال با کلاس استاتیک و در بالای تابع main و متغیرهای m و n با کلاس حافظه عمومی معرفی شده اند. اگر کل این برنامه در درون یک فایل باشد هیچ تفاوتی بین این دو تعریف وجود ندارد. اما ممکن است یک برنامه بسیار طولانی باشد و مجبور باشیم آن را در چند فایل قرار دهیم در این صورت متغیرهای عمومی و استاتیک که در یک فایل تعریف شده است فقط و فقط در همان فایل قابل

استفاده است. اما با استفاده از کلمه extern به کامپایلر اعلام می کنیم که این یک متغیر خارجی و در یک فایل دیگر تعریف شده است. بنابراین کامپایلر برای این نوع متغیرها حافظه جدیدی در نظر نمی گیرد. به مثال زیر توجه کنید:

فایل اول:

```
static int x,y;
int m,n;
void f1(void){
m=5;
n=10;
x=4;
}
void main(void)
{…
}
```

فایل دوم:

```
extern int m,n;
int f2(void){
int x;
x=m/n;
return x;
}
```

در مثال بالا برای استفاده از متغیرهای m,n میتوان دستور extern را به کار برد. در این صورت با اجرای برنامه دوم تغییرات متغیرهای m,n همان جایی که تعریف شده است ، ذخیره می شود.

۹-۴-۵- 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

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:

۱. فقط در همان تابعی که تعریف شده اند، قابل دسترسی اند.
۲. می توانند مقدار اولیه بگیرند و فقط یکبار مقدار دهی اولیه را دریافت می‌کنند.
۳. 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.

مثال:

```
int f1(void){
int x=0;
x++;
return x;
}
```

در تابع f1 متغیر محلی x دارای حافظه اتوماتیک می باشد یعنی با هر بار فراخوانیf1 ، یک واحد به متغیر x اضافه شده و به خروجی تابع بر می گردد .مقدار اولیه x برابر صفر است. بنابراین با هر بار فراخوانیf1 ، متغیر xبه مقدار اولیه خود باز می گردد. بنابراین خروجی تابع f1 همواره ۱ است.

```
int f2(void){
static int y=0;
y++;
return y;
}
```

در تابع f2 متغیر محلی y دارای حافظه استاتیک است که با اولین بار فراخوانی تابع f2 این متغیر بوجود آمده و مقدار صفر می گیرد. با هر بار فراخوانی تابع f2 یک واحد به آن اضافه می گردد و متغیر y به مقدار اولیه خود باز نمی گردد. بنابراین در اولین فراخوانی y=1 و در دومین فراخوانی y=2 و...

متغیرهای استاتیک عمومی

متغیرهای استاتیک عمومی در خارج از توابع تعریف می شوند و 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 . به مثال زیر توجه کنید:

```
void f1(void);
void f2(void);
void main(void){
…
f1();
…
f2();
…
}
static int x,y;
void f1(void){ … }
void f2(void){ … }
```

همانطور که مشاهده می شود ، متغیرهای x,y قبل از بدنه توابع f1 و f2 تعریف شده اند. لذا این متغیرها در تابع main قابل دسترسی نیستند ولی توابع f1 و f2به آنها دسترسی دارند.

۹–۵– اشاره گرها

زمانی که یک متغیر تعریف می شود ، بخشی از حافظه را اشغال می کند. بسته به نوع متغیر این بخش میتواند یک یا چند بیت یا بایت باشد. اشاره گر خود نیز یک متغیر است که به جای ذخیره کردن داده آدرس محل قرارگیری متغیرهای دیگر را در خود ذخیره می کند. یعنی یک اشاره گر به محل ذخیره یک متغیر در حافظه اشاره می کند. هر اشاره گر می تواند آدرس متغیر هم نوع خود را در خود نگه دارد. برای مثال برای نگه داری آدرس یک متغیر int نیاز به تعریف یک اشاره گر از نوع نیاز به تعریف یک اشاره گر از نوع int می باشد. قالب تعریف یک اشاره گر را در زیر مشاهده می کنید:

نام اشاره گر * نوع اشاره گر

```
int *x;
char *y;
```

همانطور که مشاهده می شود تنها تفاوت یک اشاره گر با متغیر در * قبل از نام اشاره گر است. برای نگه داری دائمی یک اشاره گر ، محل ذخیره یک اشاره گر میتواند توسط یکی از کلمات flash,eeprom تعیین شود. در صورت تعیین نکردن محل حافظه ، اشاره گر به صورت پیش فرض در حافظه SRAM ذخیره خواهد شد.

۹–۵–۱– مقدار دهی به اشاره گر

برای اینکه آدرس محل یک متغیر را در یک اشاره گر ذخیره کنیم ، از عملگر & استفاده می کنیم. مثال:

```
int *x,y;
y=142;
x=&y;
```

در این مثال در ابتدا یک اشاره گر و یک متغیر هر دو از نوع int تعریف شده است. به متغیر y مقداری نسبت داده شده است. برای اینکه آدرس محل ذخیره متغیر y در حافظه را داشته باشیم ، از عملگر & استفاده می کنیم.

۹–۵–۲– دسترسی به محتوای یک اشاره گر

برای اینکه به محتوای یک اشاره گر که به محلی از حافظه اشاره می کند را داشته باشیم ، از عملگر * استفاده می کنیم. مثال:

```
int *x,y,z;
y=142;
x=&y;
z=*x;
```

یک متغیر z به برنامه اضافه کردیم و در آن محتوای x را نسبت دادیم. بنابراین z=142 می شود.

۹–۵–۳– عملیات روی اشاره گرها

عملیات جمع و تفریق را می توان روی متغیرهای اشاره گر انجام داد اما ضرب و تقسیم را روی یک اشاره گر نمی توان استفاده کرد. نکته مهمی که باید به آن توجه کرد این است که چون اشاره گر آدرسی در حافظه

است وقتی عملیاتی که روی آن انجام می گیرد رفتار متفاوتی دارد. برای مثال عمل جمع اشاره گر را به تعداد بایت های نوع داده آن حرکت می دهد.
مثال: چون a اشاره گری به یک عدد int است و نوع int 2 بایت دارد با عمل افزایش ۲ واحد به a اضافه می شود. یعنی به ۲ بایت بعدی حافظه اشاره می کند.

```
int a=10;
int *p;
p=&a;
p=p+2;
```

۹-۵-۴- ارتباط اشاره‌گر با آرایه و رشته

در زبان برنامه‌نویسیC ، بین آرایه‌ها با رشته‌ها و اشاره‌گرها، ارتباط نزدیکی وجود دارد. اشاره‌گرها حاوی آدرس هستند و اسم هر آرایه یا رشته نیز یک آدرس است. اسم آرایه، آدرس اولین عنصر آرایه را مشخص می‌کند؛ به‌عبارت‌دیگر، اسم هر آرایه، آدرس اولین محلی را که عناصر آرایه ازآنجا به بعد در حافظه ذخیره می‌شوند، نگهداری می‌کند؛ بنابراین اسم هر آرایه، یک اشاره‌گر است.
به مثال زیر توجه کنید:

```
int table [5];
int *p;
```

همان‌طور که در مثال صفحه قبل مشاهده می‌شود، یک آرایه با ۵ عنصر به نام table و اشاره‌گر p هر دو از نوع intمعرفی‌شده‌اند. اگر اولین عنصر آرایه table در محل ۱۰۰۰ حافظه واقع‌شده باشد، نام **آرایه** به محل ۱۰۰۰ آرایه اشاره خواهد کرد.

```
p=table;
```

چون هر دو متغیر table و p از جنس اشاره گر هستند ، بدون هیچ عملگری آنها را میتوان برابر هم قرار داد. در نتیجه میتوان گفت موارد زیر با یکدیگر معادل هستند:

*(p+1) معادل table[1]

p[2] معادل *(table+1)

*p معادل *table

۹-۵-۵- استفاده از اشاره گرها در توابع

توابعی که قبلا با آن آشنا شدیم، تنها یک مقدار را به خروجی باز می گرداندند. با استفاده از اشاره گرها در ورودی توابع ، میتوان توابعی ساخت که بیش از یک خروجی داشته باشند. در این روش که به آن فراخوانی تابع با ارجاع گفته می شود، به جای متغیرهای ورودی در هنگام تعریف تابع ، اشاره گر قرار می گیرد و در هنگام فراخوانی تابع ، به جای متغیرهای ورودی آدرس آنها قرار می گیرد .بنابراین اگر تابعی می خواهیم که چند پارامتر خروجی دارد، می توانیم پارامترهای خروجی را در لیست پارامترهای ورودی تابع و به صورت اشاره گر تعریف کنیم تا تابع آنها را پر کرده و تحویل ما دهد. به مثال زیر توجه کنید :

```
void fx(int *a,int *b,int *c);
void main(void){
int x=10,y=20,z;
fx(&x,&y,&z);
}
```

```
void fx(int *a,int *b,int *c){
int i,j,k;
i=*a;
j=*b;
i=i/2;
j=j/2;
*a=I;
*b=j;
*c=i+j;
}
```

در این مثال محتوای ورودی به متغیر های i,j ریخته می شود و سپس بر روی آن عملیات مورد نظر صورت می گیرد. در این تابع هر دو متغیر i,j تقسیم بر دو شده است. سپس مقدار جدید i به جای محتوای آدرس a قرار می گیرد. همچنین مقدار جدید متغیر j به جای محتوای آدرس اشاره گر b و جمع i,j نیز در محتوای آدرسی قرار می گیرد که اشاره گر c به آن اشاره می کند. در نتیجه در پایان برنامه x=5 y=10 و z=15 می شود. (تابعی با ۳ ورودی و ۳ خروجی)

۹-۶- ساختار

همان‌طور که تا اینجا آموختیم، آرایه‌ها می‌توانند برای جمع‌آوری گروه‌هایی از متغیرهایی با نوع مشابه مورداستفاده قرار گیرند؛ بنابراین نمی‌توان به‌عنوان‌مثال آرایه‌ای تعریف کرد که شامل پنج‌خانه از نوع صحیح و پنج‌خانه از نوع اعشاری باشد. از طرفی هم در کاربردهای مختلف برنامه‌نویسی نیاز به تعریف کردن عناصر مختلف در کنار هم و منسوب کردن یک نام به همه‌ی آن‌ها داریم تا بتوانیم مجموعه‌ی آن‌ها را به‌صورت یکجا مورد پردازش‌هایی مانند بازنویسی آن‌ها به‌صورت یکجا، ارسال کل آن‌ها به یک تابع و یا آماده‌سازی و برگرداندن همه‌ی آن‌ها به‌عنوان نتیجه یک تابع قرار دهیم.

فرض کنید داده‌های مربوط به یک دانشجو مثل شماره دانشجویی، نام خانوادگی، نام، جنسیت، تعداد واحد گذرانده شده و معدل کل را که دارای نوع‌های متفاوتی هستند را بخواهیم تحت یک نام تعریف کنیم.

یا به‌عنوان‌مثال دیگر، اگر بخواهیم اطلاعات مربوط به کارکنان شرکتی را که شامل نام کارمند (از نوع کاراکتری)، شماره کارمندی (از نوع عدد صحیح)، حقوق (از نوع عدد صحیح) و ... است، تحت یک نام ذخیره کنیم، در این صورت متغیر معمولی و آرایه پاسخگوی نیاز ما نیستند. اکنون می‌خواهیم بدانیم که چگونه قطعات داده‌هایی را که نوع یکسان ندارند، (مانند مثال فوق) جمع‌آوری کنیم.

پاسخ این است که می‌توانیم متغیرهایی با نوع‌های مختلف را با نوع داده‌ای به نام ساختار گروه‌بندی کنیم. بنابراین می‌توان گفت که ساختار در زبانC ، نامی برای مجموعه‌ای از متغیرهاست که این متغیرها می‌توانند هم نوع نباشند، یعنی می‌تواند ساختاری از انواع مختلف داده‌ها ازجمله float, int, char, unsigned charو ... را درقالب یک نام در خود داشته باشد و کاربر در هر زمان به آن‌ها دسترسی داشته باشد.

قالب تعریف ساختار را در زیر مشاهده می‌کنیم:

```
struct [structure tag]{
member definition
member definition
```

...
member definition
}[one or more structure variables];

نام ساختار یا (structure tag) از قانون نام‌گذاری برای متغیرها تبعیت می‌کند .عضوهای ساختار یا (member)، متغیرهایی هستند که قسمتی از ساختار می‌باشند و همانند یک متغیر معمولی یا آرایه، باید اسم و نوع هرکدام مشخص باشد. لیست نام‌ها یا (structure variables) هم متغیرهایی هستند که قرار است ساختمان این ساختار را داشته باشند. برای استفاده از عناصر ساختار معرفی‌شده باید متغیرهایی از نوع ساختار پس‌ازآن معرفی شود. دو روش برای این کار وجود دارد.

در زیرقالب **روش اول** را مشاهده می‌کنید:

struct نام ساختار{

عناصر ساختار

نام متغیرها};

به مثال زیر توجه کنید:

```
struct student_record{
long student_number;   /*شماره دانشجویی*/
char first_name[21];   /*نام دانشجو*/
char last_name[31];    /*نام خانوادگی*/
char gender_code;      /*کد جنسیت*/
float average;         /*معدل کل*/
int   passed_units ;   /*واحدهای گذرانده شده*/
}student1;             /* تعریف متغیری از نوع ساختمان این ساختار*/
```

در این تعریف student_record نام الگوی تعریف‌شده برای این ساختار است که می‌تواند نوشته نشود و student1نام متغیری است با ساختمان این ساختار که دارای شش عضو است. درصورتی‌که بعد از خاتمه‌ی تعریف ساختار (بعد از {) نامی نوشته نشود، فقط یک الگو تعریف‌شده است و چون متغیری با ساختمان این ساختار تعریف‌نشده، حافظه‌ای اشغال نخواهد شد. در این حالت می‌توان در ادامه‌ی برنامه از کلمه‌ی struct و نام ساختار (در اینجا (student_record برای تعریف ساختارهای موردنیاز استفاده کرد .در زیرقالب **روش دوم** را می‌بینید که پس از معرفی ساختار صورت می‌گیرد :

struct <نام ساختار> <نام متغیرها>;

```
struct s_type p1,p2;
```

در بالا متغیرهایی بانام p1,p2 از نوع ساختار s_type معرفی‌شده‌اند. هرکدام از این‌ها حاوی کل ساختار معرفی‌شده، می‌باشند. درروش دوم در زمان معرفی ساختار، متغیرهایی از نوع ساختار در انتهای آن معرفی می‌شوند.

نکته ۱ : در کامپایلر کدویژن با استفاده از کلمات کلیدی eeprom و flash این قابلیت وجود دارد که ساختارها را در ناحیه SRAM ، FLASHیا EEPROM تعریف نمود. در صورتی که نام محل ذخیره سازی ذکر نشود ، کامپایلر به طور پیش فرض حافظه SRAM را انتخاب میکند.
نکته ۲ : ساختارهایی که در حافظه flash تعریف می شوند ، از نوع ساختار ثابت می باشند و نمیتوان در عناصر آن نوشت و فقط میتوان آنها را خواند.

دسترسی به عناصر ساختار

قالب دسترسی به عناصر ساختار با استفاده از نقطه (dot) و به صورت زیر است :
نام عناصر موردنظر در ساختار. نام متغیر از نوع ساختار
ابتدا نام ساختار حاوی آن عضو و سپس نام عضو بیان‌شده و این نام‌ها با عملگر عضو ساختار که علامت آن نقطه است به یکدیگر مرتبط می‌شوند. این عملگر که دارای بالاترین تقدم عملیات بوده ترتیب اجرایش از چپ به راست است. توجه شود که به‌صورت قراردادی در دو طرف عملگر نقطه فاصله خالی گذاشته نمی‌شود. اگر عناصر از نوع آرایه باشند، ذکر اندیس آرایه جهت دستیابی به آن عنصر ضروری است.
مثال :ساختاری را تعریف کنید که اطلاعات یک دانشجو را در خود ذخیره نماید.

```
struct student{
long int id;
char name[20];
float average;
int age;
}s;
```

مثال: مقداردهی اولیه به عناصر متغیر ساختاریs

```
s.id = 860133710;
s.name = "ali";
s.average = 17.64;
s.age = 18;
```

تخصیص ساختارها

در مقداردهی اولیه می‌توان مجموعه‌ای از مقادیر را به یک ساختار نسبت داد، ولی انجام چنین کاری در متن برنامه به‌صورت دستور اجرایی امکان‌پذیر نیست .تنها عمل تخصیص که در مورد رکوردها در زبان C تعریف‌شده است، تخصیص یک ساختار به ساختار دیگری با ساختمان دقیقا یکسان (هر دو ساختار از طریق یک structتعریف‌شده باشند) هست که در این صورت محتویات هر فیلد از ساختار مبدأ به فیلد متناظر از ساختار مقصد منتقل می‌شود. ساختار مبدأ می‌تواند متغیری در برنامه یا خروجی یک تابع باشد.

اشاره‌گرها و ساختارها

در زبان C تعریف اشاره‌گر از نوع ساختار، همانند تعریف سایر انواع اشاره‌گرها امکان‌پذیر است. همان‌طور که در فراخوانی تابع می‌توانید اشاره‌گری را ارسال کنید که به آرایه ارجاع می‌دهد، می‌توانید اشاره‌گری که به ساختار اشاره می‌کند را نیز ارسال کنید.
به‌هرحال برخلاف ارسال ساختار به تابع که نسخه‌ی کاملی از ساختار را به تابع می‌فرستد، ارسال اشاره‌گر به ساختار فقط آدرس ساختار را به تابع می‌فرستد .سپس تابع می‌تواند جهت دستیابی مستقیم به اعضای

ساختار از آدرس استفاده می‌کند و از سرریزی تکرار ساختار پرهیز نماید، بنابراین این روش جهت ارسال اشاره‌گر به ساختار کارآمدتر است، نسبت به اینکه خود ساختار را به تابع ارسال کنید.
اشاره‌گر ساختار به دو منظور استفاده می‌شود:

- امکان فراخوانی به روش ارجاع را در توابعی که دارای آرگومان از نوع ساختار هستند، فراهم می‌کند.
- برای ایجاد فهرست‌های پیوندی و سایر ساختار داده‌هایی که با تخصیص حافظه پویا سروکار دارند، به کار می‌رود.

وقتی‌که ساختارها از طریق فراخوانی به روش ارجاع به توابع منتقل می‌شوند، سرعت انجام عملیات بر روی آن‌ها بیشتر می‌گردد. لذا در حین فراخوانی توابع، بهتر است به‌جای ساختار، آدرس آن را منتقل نمود. عملگر &برای مشخص کردن آدرس ساختار مورداستفاده قرار می‌گیرد.
تعریف اشاره‌گرهای ساختار مانند تعریف متغیرهای ساختار است، با این تفاوت که قبل از اسم متغیر، علامت * قرار می‌گیرد.
مثال زیر را در نظر بگیرید، در این دستورات person یک متغیر ساختار و p یک اشاره‌گر ساختار تعریف‌شده است.

```
struct bal{
float balance;
char name[80];
}person, *p;
```

اکنون دستور زیر را در نظر بگیرید

```
p = &person;
```

با این دستور، آدرس متغیر ساختار person در اشاره‌گر p قرار می‌گیرد. برای دسترسی به محتویات عناصر ساختار از طریق اشاره‌گر، باید اشاره‌گر را در داخل پرانتز محصور کنید. به‌عنوان‌مثال دستور زیر موجب دسترسی به عنصر balanceاز ساختار person می‌شود. علت قرار دادن متغیر اشاره‌گر در پرانتز، این است که تقدم عملگر نقطه از * بالاتر است.

```
(*p).balance;
```

به‌طورکلی برای دسترسی به عناصر ساختاری که یک اشاره‌گر به آن اشاره می‌کند به دو روش می‌توان عمل کرد:

- ذکر نام اشاره‌گر در داخل پرانتز و سپس نام عناصر موردنظر که با نقطه از هم جدا می‌شوند. (مثل دسترسی به عنصر balance از ساختار person توسط اشاره‌گر (p
- استفاده از عملگر <- که روش مناسب‌تری است. اگر بخواهیم با استفاده از عملگر <-به عنصر balance از ساختار person دسترسی داشته باشیم باید به طریق زیر عمل کنیم (علامت <- متشکل از علامت منها و علامت بزرگ‌تر است).

```
p -> balance;
```

در این‌جا بیان این نکته ضروری است که آرایه‌ها، اشاره‌گرها و ساختارها دارای ارتباط نزدیکی باهم هستند. در ضمن عمگرهای مربوط به آن‌ها شامل زیر نویس یا []، عضور رکورد یا نقطه و دستیابی غیرمستقیم به عضو رکورد یا <- همگی دارای بالاترین تقدم هستند. عملگرهای دستیابی غیرمستقیم یا * و استخراج آدرس یا & هم دارای تقدم دوم می‌باشند. حال اگر این عبارت‌ها همراه همدیگر در عبارتی ظاهر شوند باید کمی با دقت کد مورد نظر نوشته شود. به مثال زیر دقت کنید:

```
struct point{
```

```
float x;
float y;
};
struct line{
struct point strat;
struct point end;
char *name;
} a = {1, 1, 10, 20, "ab"};
struct line *pa, *pm,
m[] = {{2, 3, 4, 5, "cd"},{4,6,8,1, "mn"},{8,5,4,2, "xy"}};
pa = &a;
pm = &m[1];
```

با توجه به تعاریف بیان شده و اینکه ترتیب اجرای دو عملگر عضو ساختار و دستیابی غیرمستقیم از چپ به راست است، عبارت‌های زیر معادل هستند:

```
a.start.x
pa->start.x
(a.start).x
(pa->start).x
(*pa).start.x
```

دو عبارت زیر نیز یک معنی می‌دهند:

```
m[1].end.y
pm->end.y
```

با این توضیح که در اولی دستیابی از طریق اندیس خانه‌ی یکم آرایه‌ی m که یک ساختار است به عضو end و سپس به عضو y انجام شده است، ولی در دومی دستیابی به عضو end از طریق آدرس خانه‌ی یکم آرایه‌ی m که در متغیر pm قرار دارد انجام شده است.

آرایه‌ای از ساختارها

در C، نام آرایه را می‌توانید در مقابل نام ساختار قرار دهید تا آرایه‌ای از ساختارها را اعلام کنید. یکی از بیشترین موارد کاربرد ساختارها، استفاده از آن‌ها به‌عنوان عناصری از آرایه است. برای تعریف آرایه‌ای از ساختارها، ابتدا نوع ساختار را تعریف کرده سپس همانند متغیرهای معمولی، آرایه‌ای از آن نوع را تعریف می‌کنیم. برای نمونه، با فرض داشتن ساختاری به نام x حکم زیر راداریم:

```
struct x array_of_structure[8];
```

آرایه‌ای به نام array _of_structure از Struct x اعلام کرده است. این آرایه ۸ عنصر دارد، هر عنصر آن نمونه یکتایی از Struct x است. مثال زیر را در نظر بگیرید:

```
struct student{
char name[21];
int stno;
float ave;
};
struct student st[100];
```

در این دستورات، آرایه ۱۰۰ عنصری st طوری تعریف می‌شود که هر یک از عناصر آن، از نوع ساختار student است.

یونیونها

ساختمان یونیون کاملا مشابه با ساختار است با این تفاوت که به جای کلمه کلیدی structاز کلمه کلیدی unionاستفاده می شود. اما در یونیون محل مشخصی از حافظه ، بین دو یا چند متغیر به صورت اشتراکی مورد استفاده قرار می گیرد به طوری که متغیرها همزمان نمیتوانند از این محل در حافظه استفاده کند ، بلکه هر متغیر در زمان های متفاوتی می تواند این محل را مورد استفاده قرار دهد .بنابراین فضایی که یک یونیون در حافظه اشغال می کند مانند فضایی که یک ساختمان اشغال می کند نیست. بلکه یونیون بیشترین طول متغیر درون خود را) به لحاظ طول بیت) به عنوان طول خود در نظر می گیرد و این فضا را بین بقیه متغیرهای درون یونیون به اشتراک می گذارد. یونیون نیز همانند ساختمان دو روش تعریف دارد. نحوه تعریف یونیون به صورت زیر است:

```
union نام یونیون
{
عناصر یونیون
};
```

مثال:

```
union u_type
{
char I;
int x;
float y;
}
```

در این یونیون تعریف شده چون بزرگترین نوع float است که ۴ بایت حافظه را اشغال می کند ، بنابراین کل یونیون ۳۲ بیت از حافظه را اشغال میکند.

روش اول معرفی یونیون : بعد از تعریف

unionنام متغیرها نام یونیون;

مثال:

```
union u_type i1,i2;
```

روش دوم معرفی یونیون : در حین تعریف

مثال:

```
union u_type
{
char I;
int x;
float y;
}i1,i2;
```

دسترسی به عناصر یونیون : با استفاده از نقطه صورت می گیرد. مثال:

```
i1.i=33;
i2.x=2048;
i1.y=6.28;
i2.i=254;
```

نکته : هنگامی که از نوع های داده ای کوچکتر از ۳۲ بیت استفاده شود بقیه بیت ها بدون استفاده و ۰ هستند.

داده شمارشی

این نوع داده قابلیت نامگذاری یک مجموعه را می دهد. برای مثال میتوان روزهای هفته را درون یک داده شمارشی قرار داد به طوری که روز اول هفته عدد ۰ و روز آخر هفته عدد ۶ تخصیص می یابد. قالب معرفی نوع داده شمارشی را در زیر مشاهده می کنید:

```
نام نوع شمارشی enum
{
عنصر اول;
عنصر دوم;
...
عنصر آخر;
};
```

مثال:

```
enum color
{
red;
green;
blue;
yellow;
};
```

روش اول معرفی نوع شمارشی : بعد از تعریف

enumنام متغیرها نام نوع شمارشی;

```
enum color c1,c2;
```

روش دوم معرفی نوع شمارشی : در حین تعریف

مثال:

```
enum color
{
red;
green;
blue;
yellow;
}c1,c2;
```

: نحوه مقدار دهی به نوع شمارشی : مثال

```
c1=red;
c2=blue;
```

تغییر نام انواع داده ها با دستور typedef

توسط این دستور در زبان c میتوان نام داده هایی همچون int,char,float,… را به هر نام دلخواه دیگری تغییر داد. این دستور دو مزیت دارد:

۱. موجب می شود تا برای نوع داده هایی که نام طولانی دارند ، نام کوتاه تری انتخاب کرد.

۲. اگر برنامه به کامپایلر دیگری منتقل شود و کامپایلر جدید و قبلی از نظر نوع و طول داده ها مطابقت نداشته باشد، برای حل مشکل کافیست فقط در دستور typedef تغییراتی ایجاد شود.

قالب استفاده از این دستور را در زیر مشاهده می کنید:

typedef نام موجود نام جدید ;

در قالب بالا نام موجود یکی از نوع های معتبر در زبان سی است و نام جدید نامی دلخواه برای آن می باشد.

مثال:

```
typedef char str;
```

بعد از دستور بالا میتوان به جای char از str برای تعریف داده های جدید استفاده کرد. مثال:

```
str x,y;
```

لزوم برنامه نویسی به سبک ماژولار

در طراحی و توسعه ی یک پروژه لازم است به موارد زیر توجه شود:

تولید کننده ای که بخواهد برای هر سیستم جدید از ابتدا شروع به طراحی برنامه کند نمی تواند برای مدت طولانی توانایی رقابت در بازار را داشته باشد. زبان برنامه نویسی که استفاده می شود باید توانایی ایجاد کتابخانه های انعطلاف پذیر را داشته باشد، تا برنامه نویس بتواند از کتابخانه هایی که آزمایش (test)، اشکال زدایی (debug) و تایید (verify) شده اند در پروژه های آینده استفاده کند. همچنین لازم است امکان انطباق کتابخانه با میکروکنترلرهای جدید وجود داشته باشد.

کارکنان یک مجموعه تغییر می کنند و حتی اگر ثابت باشند، حافظه ی انسان بازه ی زمانی محدودی را به خاطر می آورد. هر سیستمی نیاز به ارتقا و به روز رسانی دارد و اگر برنامه ی آن به شیوه ی درستی نوشته و مستند سازی نشده باشد، درک و تغییر آن مشکل می شود. بنابراین برنامه ی خوب، برنامه ای است که امکان درک و تغییر آن در هر زمان وجود داشته باشد (نه فقط به وسیله ی طراح اولیه، بلکه به وسیله افراد دیگر.(

علاوه بر موارد فوق لازم است به این نکته توجه شود که در اجرای پروژه های بزرگ و پیچیده اگر از سبک برنامه نویسی مناسبی استفاده نشود، نگهداری و اشکال زدایی برنامه بسیار مشکل می شود، هزینه ها افزایش می یابد و امکان موفقیت طرح به حداقل می رسد. همچنین گاهی لازم است پروژه های بزرگ به اجزای کوچکتری خرد شوند و هر بخش را فرد و یا تیم مستقلی پیاده سازی کند. با استفاده از سبک برنامه نویسی ماژولار، می توانیم کتابخانه هایی ایجاد کنیم که به سادگی قابل تغییر و قابل استفاده در پروژه های دیگر هستند.

مفهوم ماژول نرم افزاری

ماژول به معنای مولفه و یا جزئی از برنامه است که خود شامل تعدادی تابع مرتبط با یکدیگر است. معمولا مجموع توابعی که داخل یک ماژول قرار می گیرند عملکرد خاصی را پیاده سازی می کنند.

برخی از ماژول ها ارتباط جانبی داخلی یا خارجی ندارند و معمولا فرآیندی را بر روی داده انجام می دهند. در این نوع ماژول ها، بخشی از برنامه به عنوان مشتری، سرویسی را از ماژول درخواست می کند و ماژول آن درخواست را پاسخ می دهد.

برخی از ماژول ها نیز ارتباط برنامه با یک سخت افزار داخلی (مانند پورت های سریال، مبدل آنالوگ به دیجیتال و ...) یا خارجی (مانند نمایشگرLCD ، صفحه کلید، سنسور و ...) را برقرار می کنند. ماژول شامل

دو بخش پیاده سازی و واسط (Interface) است. در بخش پیاده سازی، بدنه ی توابع قرار می گیرند و در بخش دوم نحوه ی استفاده از توابع، در اختیار کاربر (مشتری سرویس) قرار می گیرد.
یکی از دلایل اساسی استفاده از ماژول این است که بتوان بخش های لازم از یک موضوع را مشخص و بخش های غیر ضروری را پنهان کرد. بنابراین بخش واسط، شامل اطلاعاتی است که برای استفاده از آن موضوع مورد نیاز است و بخش پیاده سازی، شامل چگونگی انجام آن موضوع است. به پنهان سازی جزئیات غیر ضروری، Abstractionگفته می شود. از این مفهوم در طراحی بسیاری از وسایل استفاده می شود. به عنوان مثال زمانی که یک خودرو را می رانیم لزوما نیاز نیست که از نحوه ی عملکرد موتور و سایر اجزا آن آگاهی داشته باشیم، بلکه کافی است که واسط استفاده موتور(شامل پدال های کلاچ، ترمز و گاز) در اختیارمان باشد و نحوه ی به کارگیری این واسط را بدانیم. در برنامه نویسی نیز، Abstractionمفهوم بسیار ارزشمندی است که در زبان های شی گرا (مانند java و (C++ به شکل عالی و در زبان های ماژولار به شکل ابتدایی از آن پشتیبانی شده است.
متاسفانه زبان C از شی گرایی و برنامه نویسی ماژولار پشتیبانی نمی کند، اما با استفاده از فایل های سورس (source)و هدر (header) می توان به تا حد قابل قبولی به شیوه ی ماژولار برنامه نویسی کرد.

برنامه نویسی به روش ماشین حالت

یکی از روش های برنامه نویسی پیشرفته ، استفاده از ساختار ماشین حالت یا State Machine در برنامه نویسی می باشد. این سبک از برنامه نویسی دارای کمترین هنگ و خطا و بیشترین انعطاف پذیری است. در این روش از حلقه switch برای تشریح حالت های مختلف و از نوع شمارشی enum برای تدوین انواع حالت ها استفاده خواهیم کرد. بنابراین ساختار کلی این روش به صورت زیر است:

```
…
enum { s0,s1,s2,….,sn}state;
void main {
…
while(1){
کاری که هر بار باید انجام شود و حالت های مختلف بر اساس آن شکل می گیرد
switch(state)
{
case s0:
کارهایی که در حالت اول باید صورت گیرد
if( شرط رفتن به حالت بعد ) state=حالت بعدی
case s1:
کارهایی که در حالت دوم باید صورت گیرد
if( شرط رفتن به حالت بعد ) state=حالت بعدی
.
.
.
case sn:
کارهایی که در حالت آخر باید صورت گیرد
if( شرط رفتن به حالت بعد ) state=حالت بعدی
default:
```

کارهایی که در صورت برقرار نبودن هیچ یک از حالت های فوق انجام می شود
}
}
}

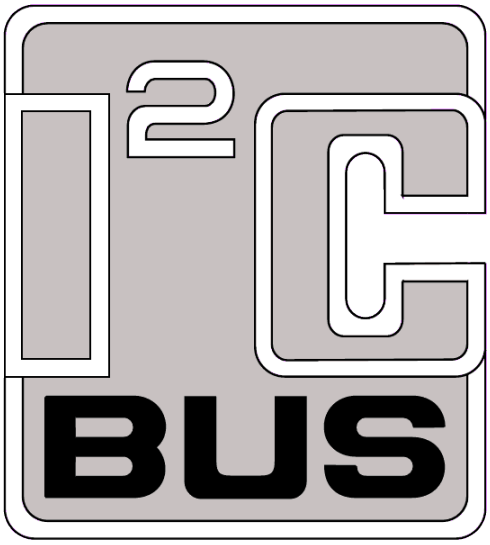
فصل ۱۰– راه اندازی ارتباط سریال I2C

مقدمه

همانطور که در قسمت های قبلی آموزش گفته شد ، در میکروکنترلرهای AVR ارتباط سریال در ۴ پروتکل زیر وجود دارد :

۱. **USART** : پایه های Rx و Tx
۲. **SPI** : پایه های MISO ، MOSI ، SCK و SS
۳. **I2C** : پایه های SDA و SCL
۴. **USB** : پایه های +D و -D

نکته : تنها برخی از میکروکنترلرهای AVR از ارتباط USB پشتیبانی می کنند و در Atmega32 ارتباط USB وجود ندارد.



۱۰–۱– معرفی ارتباط سریال I2C

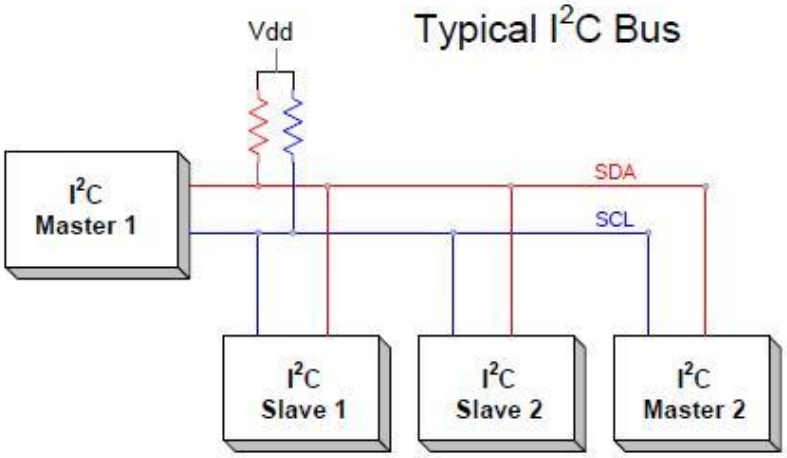
I2C مخفف عبارت Inter Integrated Circuit به معنای مدار مجتمع یکپارچه می باشد. به علت اینکه در این پروتکل تنها از دو سیم برای ارتباط دو یا چند وسیله استفاده می شود ، این پروتکل را ارتباط دو سیمه (TWI) مخفف Two Wire Interface نیز می نامند. این پروتکل توسط شرکت philips در سال ۱۹۸۲ طراحی و به کار گرفته شد.

از خصوصیات این رابط در میکروکنترلرهای AVR میتوان به موارد زیر اشاره نمود :

۱. رابطی انعطاف پذیر ، قدرتمند و با سرعتی نسبتا مناسب که فقط نیاز به دو خط انتقال دارد.
۲. قابلیت پشتیبانی از مدهای عملکرد Master و Slave در حالت های فرستنده و گیرنده.
۳. دارای ۷ بیت آدرس دهی که قابلیت ارتباط یک Master با حداکثر ۱۲۸ Slave را فراهم می کند.
۴. پشتیبانی از ارتباط تعدادی Master (Multi Masters)
۵. سرعت انتقال اطلاعات تا 400KHz

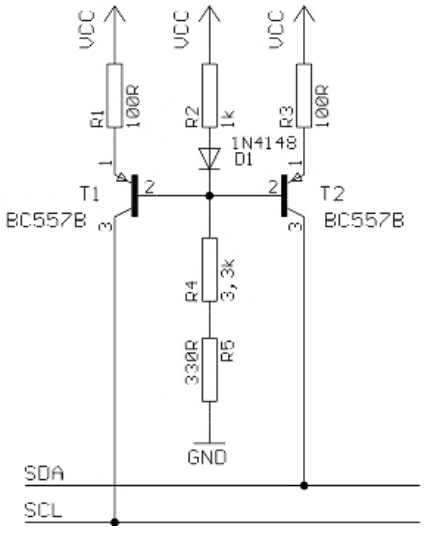
۶. حذف نویز مدارات روی گذرگاه
۷. محدودیت در نرخ چرخش (SlewRate) در خروجی درایور ها
۸. بیدار شدن خودکار میکروکنترلر از حالت Sleep به محض ارسال آدرس آن
۹. قابلیت فراخوانی عمومی همه Slave ها توسط Master

۱۰-۲- شبکه بندی Master و Slave ها در پروتکل I2C

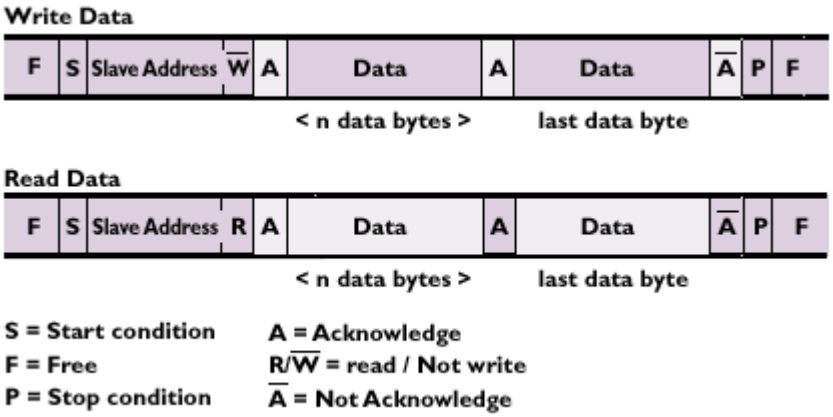


همانطور که در شکل فوق مشاهده می کنید ، در این پروتکل تمامی دستگاه ها به دو سیم SDA و SCL متصل شده و هر یک از خطوط توسط یک مقاومت بالاکش PullUp به منبع تغذیه وصل می شوند. خط SDA برای انتقال دیتای سریال (Serial Data) و خط SCL برای انتقال کلاک سریال (Serial Clock) به کار می رود. طبق استاندارد ، مقدار مقاومت پول آپ برای منبع تغذیه ۵ ولت برابر 4.7K و برای منبع تغذیه 3.3 ولت برابر 1.5K می باشد.

نکته : استفاده از پروتکل I2C در فواصل بسیار کوتاه (کمتر از ۱۰ سانتی متر) امکان پذیر است. برای فواصل طولانی تر باید تقویت سیگنال صورت گیرد. در ارتباطات i2c با فاصله بین ۱۰ تا ۷۰ سانتی متر میتوان از مدار تقویت کننده زیر استفاده کرد.

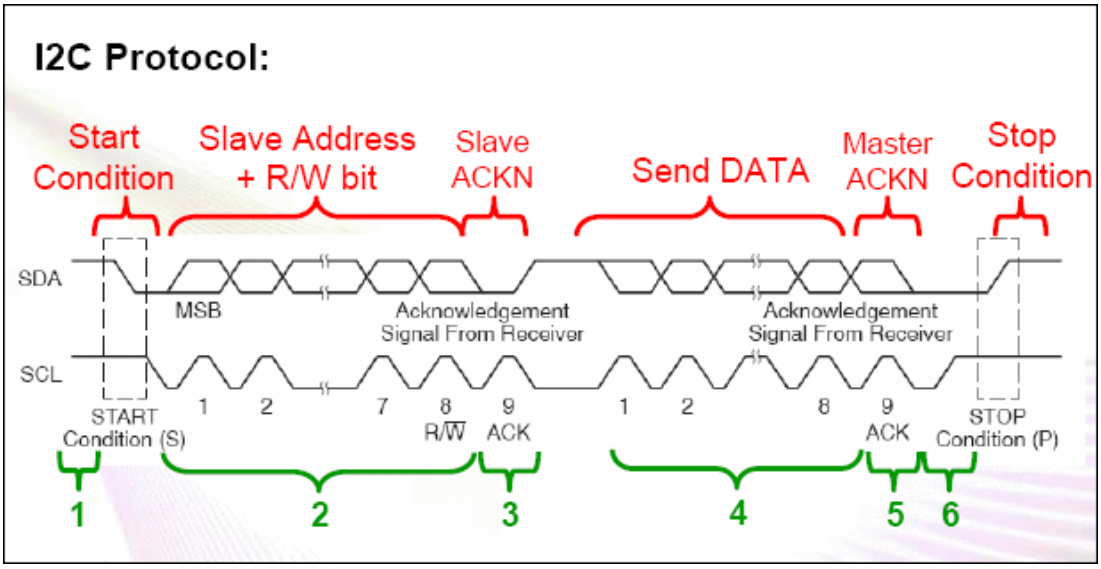


۱۰–۳–قالب بندی ارتباط در پروتکل I2C



ابتدا به منظور جلوگیری از تداخل دستگاه ها با یکدیگر ، به هر دستگاه یک آدرس منحصر به فرد بین ۰ تا ۱۲۷ اختصاص می یابد. سپس برای ارسال داده به یکی از دستگاه های متصل به باس I2C ، دستگاه Master ابتدا پالس ۷ بیتی مربوط به آدرس دستگاه مورد نظر را به صورت سریال (از MSB با ارزشترین بیت تا LSB کم ارزش ترین بیت) روی باس SDA ارسال می کند و سپس بعد از دریافت آن توسط Slave مورد نظر ، یک بیت مبتنی بر تایید در دسترس بودن Slave به Master ارسال می شود (Aknowledge) سپس دستگاه Master داده های مورد نظر را در قالب ۸ بیتی ارسال می کند و پس از ارسال هر بسته ۸ بیتی یک بیت Aknowledge دریافت می کند. دریافت داده نیز تقریبا همین قالب را دارد با تفاوت یک بیت که نشان دهنده دریافت است.

باس SCL همواره در حال ارسال کلاک از Master به تمام Slave ها است. به صورتی که هر بیتی که از باس SDA از Master ارسال می گردد ، تنها در لبه های کلاک SCL توسط Slave ها خوانده می شود. در شکل زیر دو باس SDA و SCL را به همراه عملکرد آن مشاهده می کنید.



بنابراین قالب دیتا در پروتکل I2C به صورت سه وضعیت زیر می باشد :

۱۰-۳-۱- وضعیت Stop / Start

در حالت عادی هر دو خط SCL و SDA در وضعیت سکون (High) قرار دارند. زمانی که قصد ارسال/دریافت توسط Master را داشته باشیم ، ابتدا یک لبه پایین رونده به منظور شروع ارسال/دریافت توسط Master ایجاد می شود. بعد از وضعیت شروع ، گذرگاه مشغول می باشد و هیچ Master دیگری نباید سعی کند کنترل گذرگاه را بر عهده بگیرد. در پایان عملیات نیز یک لبه بالا رونده به منظور توقف ارسال/دریافت رو گذرگاه قرار می گیرد و از آن به بعد دوباره گذرگاه به حالت آزاد (عادی) بر می گردد.

۱۰-۳-۲- وضعیت ارسال آدرس

در وضعیت آدرس ، یک بسته بندی ۹ بیتی بر روی گذرگاه داده از فرستنده به گیرنده ارسال می شود. این ۹ بیت تشکیل شده است از ۷ بیت آدرس به همراه یک بیت کنترل خواندن یا نوشتن Write/Raed و یک بیت به نام ACK است که گیرنده برای فرستنده ارسال می کند تا مشخص شود کد آدرس دریافت شده صحیح است. بنابراین زمانی که Slave آدرس خود را تشخیص داد ، باید با پایین بردن خط SDA در نهمین سیکل کلاک SCL ، بیت ACK را ارسال نماید. اگر بیت مربوط به R/W یک شده باشد ، زمانی که از وضعیت ارسال آدرس به وضعیت بعدی یعنی وضعیت ارسال/دریافت دیتا برویم ، عملکرد نوشتن Read روی Slave و در صورت صفر بودن این بیت عملکرد خواندن Write روی Slave اجرا می شود.

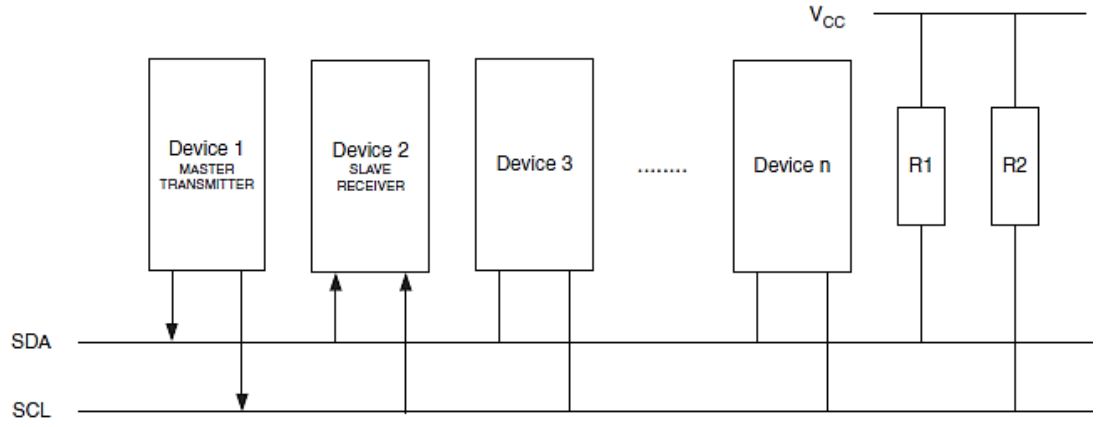
۱۰-۳-۳- وضعیت ارسال/دریافت دیتا

در این وضعیت نیز یک بسته بندی ۹ بیتی شامل ۸ بیت دیتا و یک بیت ACK وجود دارد. تعداد داده ها ممکن است بیشتر از ۸ بیت باشد. در این صورت تا زمان پایان همه داده ها ، سیستم در وضعیت ارسال/دریافت دیتا باقی می ماند تا اینکه تمامی داده های در بسته بندی ۹ بیتی ارسال/دریافت شوند. در هر بار ارسال/دریافت گیرنده می بایست بعد از دریافت ۸ بیت دیتا با Low نمودن خط SDA در نهمین سیکل کلاک SCL ، صحت دریافت را ACK نماید. در غیر این صورت با بالا رفتن خط SDA ، فرستنده متوجه عدم دریافت صحیح شده و دیتا را مجددا ارسال می نماید.

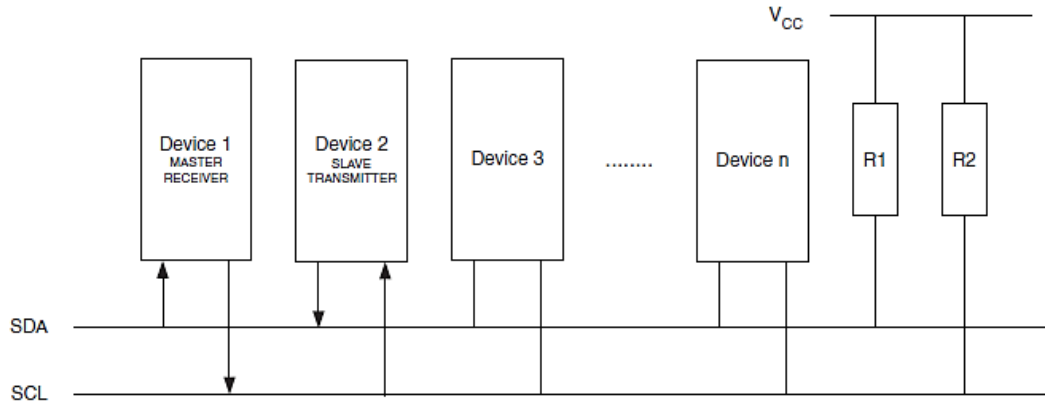
۱۰-۴- مدهای عملکرد واحد TWI

بر اساس فرستنده (Transmitter) یا گیرنده (Reciever) بودن هر میکرو و اینکه هر میکرو میتواند Master یا Slave باشد. ۴ مد عملکرد در واحد TWI به صورت زیر بوجود می آید. یعنی هر یک از میکروکنترلرهای AVR می تواند یکی از چهار حالت زیر را به خود بگیرد :

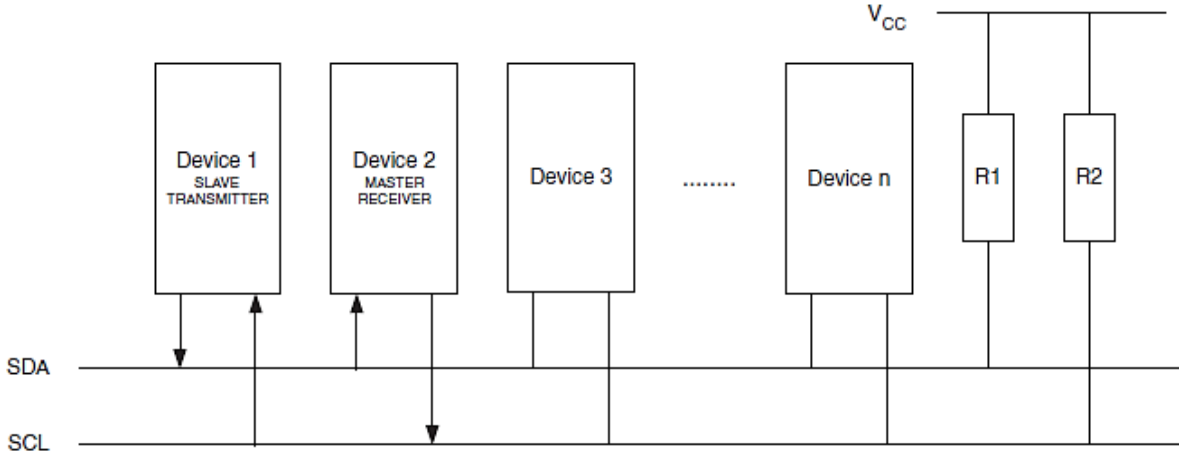
۱. **MT (Master و فرستنده) :** در این مد ابتدا توسط Master یک وضعیت Start ایجاد می شود. سپس بایت های داده به سمت گیرنده Slave ارسال می شود.



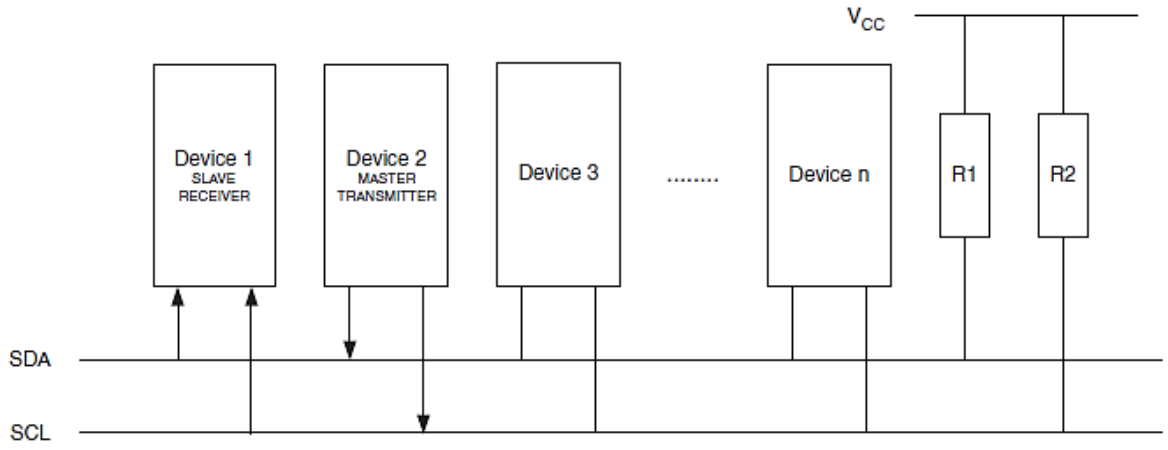
۲. **MR (Master و گیرنده)** : در این مد Master منتظر وضعیت Start می ماند و سپس تعدادی بایت از Slave ارسال و توسط Master دریافت می شود.



۳. **ST (Slave و فرستنده)** : در این مد فرستنده یک Slave است که اطلاعاتی را به یک MR ارسال می کند.



۴. **SR (Slave و گیرنده)** : در این مد گیرنده یک Slave است که اطلاعاتی را از یک MT دریافت می کند.



۱۰–۵– انواع دسترسی به رابط I2C در کدویژن

برای دسترسی به رابط دو سیمه و استفاده از آن در میکروکنترلرهای AVR بوسیله نرم افزار Codevision به دو صورت زیر میتوان عمل کرد :

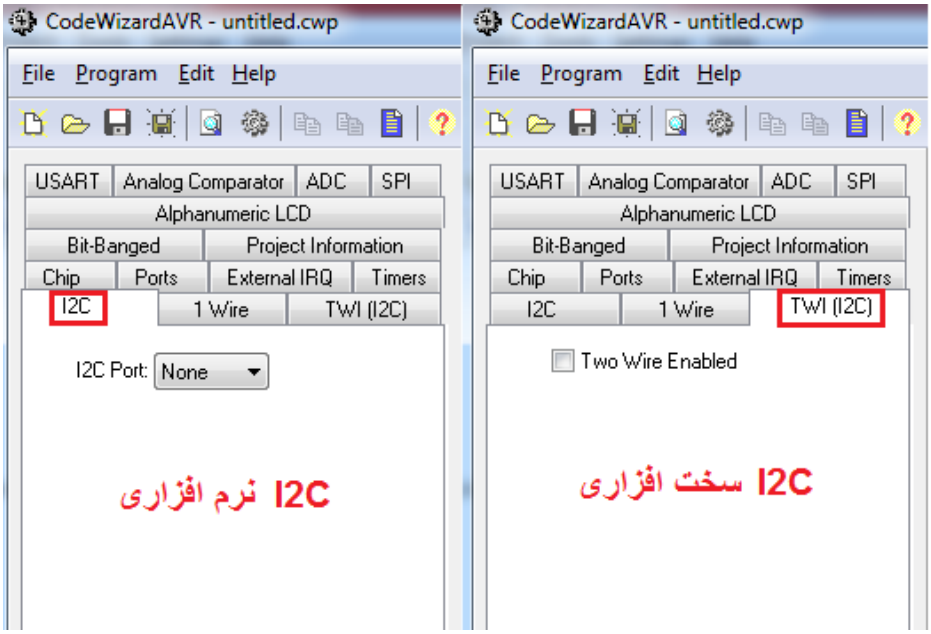
۱. دسترسی به واسط I2C با استفاده از واحد سخت افزاری TWI
۲. دسترسی به واسط I2C به صورت نرم افزاری با اضافه کردن هدرفایل i2c.h

تفاوت های استفاده از واسط I2C سخت افزاری با نرم افزاری

۱. در صورت استفاده از I2C سخت افزاری تنها میتوان پایه های SDA و SCL در میکروکنترلرهای AVR را استفاده نمود در حالی که در صورت استفاده از I2C نرم افزاری میتوان هر دو پایه دلخواه را به عنوان SDA و SCL تعریف و استفاده کرد.
۲. در صورت استفاده از I2C نرم افزاری ، بخشی از CPU درگیر تولید پالس های SDA و SCL می شود در حالی که در صورت استفاده از I2C سخت افزاری ، یک واحد مجزا درگیر می شود و سرعت برنامه بیشتر می شود.
۳. در صورت استفاده از I2C نرم افزاری ، با اضافه شدن هدرفایل مربوطه میتوان از توابع آماده موجود فقط در حالت های MR/MT استفاده کرد اما در استفاده از I2C سخت افزاری هدر فایل آماده ای نیست و باید با رجیسترها کار کرد. البته میتوان در هر چهار حالت توابع مورد نیاز را به صورت دستی وارد کرد.

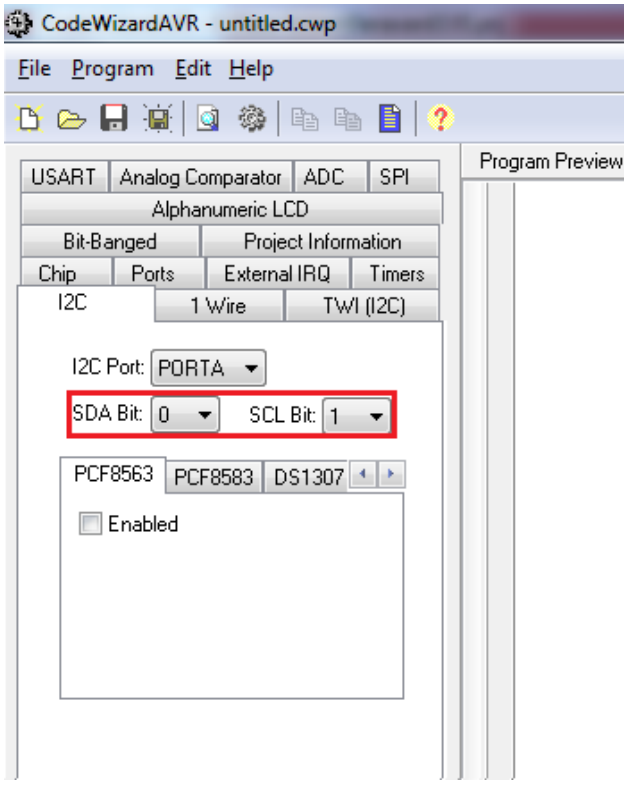
۱۰–۶– فعالسازی رابط I2C در کدویزارد

در ابزار کدویزارد برای فعالسازی I2C سخت افزاری به سربرگ TWI و برای فعالسازی I2C نرم افزاری به سربرگ i2c می رویم. شکل زیر تفاوت آن را نشان می دهد.



۱۰–۷–راه اندازی I2C نرم افزاری

برای فعالسازی واحد ارتباطی دو سیمه نرم افزاری بعد از رفتن به سربرگ i2c ، ابتدا پورتی را که میخواهیم به صورت نرم افزاری از آن به عنوان رابط استفاده نماییم را انتخاب و سپس شماره پایه پورت دلخواه را در قسمت SDA Bit و SCL Bit به ترتیب برای خطوط SDA و SCL انتخاب می نماییم. در قسمت پایین سربرگ ابزار کدویزارد برای برخی از آی سی های پرکاربرد مانند LM75 ، DS1307 ، DS1621 ، PCF8563 و PCF8583 تنظیمات خاصی درنظر گرفته است که میتوان با فعال کردن آنها توابع خاصی را برای استفاده در پروژه خود به برنامه افزود.



بعد از تولید کد توسط برنامه کدویزارد مشاهده می شود که هدرفایل i2c.h اتوماتیک به پروژه افزوده شده و تنظیمات مربوط به بیت های SDA و SCL قبل از آن اضافه شده است. با اضافه شدن این هدر فایل میتوان از توابع زیر در پروژه برای ارتباط سریال استفاده نمود.

۱۰-۷-۱- توابع موجود در کتابخانه i2c.h

۱. تابع i2c_init

این تابع گذرگاه TWI نرم افزاری را روی مقادیر SDA و SCL اولیه فراخوانی و راه اندازی می کند. به همین دلیل باید قبل از فراخوانی توابع دیگر این تابع را صدا زد. الگوی این تابع به صورت زیر است :

```
void i2c_init(void)
```

۲. تابع i2c_start

با اجرای این تابع یک وضعیت Start ایجاد می شود و اگر گذرگاه I2C آزاد باشد ، مقدار یک توسط این تابع باز می گردد و در غیر این صورت مقدار صفر باز خواهد گشت. الگوی این تابع به صورت زیر است :

```
unsigned char i2c_start(void)
```

۳. تابع i2c_stop

با اجرای این تابع یک وضعیت stop بر روی گذرگاه I2C ایجاد می شود. الگوی این تابع به صورت زیر است :

```
void i2c_stop(void)
```

۴. تابع i2c_read

این تابع از گذرگاه I2C یک بایت را می خواند. ورودی این تابع یک بیت ack است. در صورتی که ack=0 باشد بایت وارد شده صحیح ارزیابی نشده است و در صورتی که ack=1 باشد بایت وارد شده صحیح بوده است. الگوی این تابع به صورت زیر است :

```
unsigned char i2c_read(unsigned char ack)
```

۵. تابع i2c_Write

این تابع یک بایت را به گذرگاه I2C ارسال می کند. اگر گیرنده Slave بیت ACK را صادر کرده باشد ، این تابع مقدار یک را باز می گرداند و در غیر این صورت مقدار بازگشتی صفر خواهد بود. الگوی این تابع به صورت زیر است :

```
unsigned char i2c_write(unsigned char data)
```

نکته : این توابع در میکروکنترلرهای سری Atxmega پشتیبانی نمی شود.

نکته : فرکانس I2C نرم افزاری ثابت بوده و قابل تنظیم نمی باشد. در صورتی که فرکانس کاری میکروکنترلر ۱ مگاهرتز باشد ، فرکانس I2C نرم افزاری روی ۶۲٫۵ کیلوهرتز است و هنگامی که فرکانس کاری میکروکنترلر روی ۸ مگاهرتز باشد ، فرکانس I2C نرم افزاری روی ۴۰۰ کیلوهرتز می باشد. (همواره روی حداکثر فرکانس ممکن است)

نکته مهم : به علت محدودیت های استفاده از توابع I2C نرم افزاری ، فقط میتوان از این توابع در میکروکنترلر Master و در یکی از حالت های MR/MT استفاده نمود.

نتیجه : با استفاده از i2c نرم افزاری توسط یک میکروکنترلر Master میتوان با انواع Slave ها نظیر انواع سنسورها ، EEPROM ها و ... ارتباط برقرار کرد اما Slave نمیتواند خود یک میکروکنترلر باشد که از i2c نرم افزاری استفاده می کند. بنابراین برای ارتباط میان چند میکروکنترلر بوسیله پروتکل I2C باید از واحد TWI سخت افزاری استفاده کرد.

۱۰-۷-۲- نحوه استفاده از توابع i2c.h

بعد از انجام تنظیمات مربوط به I2C نرم افزاری در کدویزارد و تولید کد میتوان از توابع موجود در هدرفایل i2c.h برای ساخت توابع جدیدی در پروژه استفاده نمود. این توابع عبارتند از یک تابع READ_I2C برای خواندن از دستگاه اسلیوی که به گذرگاه I2c متصل است و یک تابع WRITE_I2C برای نوشتن در دستگاه اسلیوی که به گذرگاه I2C متصل است. ساختار درونی این دو تابع جدید برای هر وسیله ای که به گذرگاه متصل شده باشد متفاوت است. مثلا برای اتصال EEPROM به گذرگاه یک ساختاری باید رعایت شود و برای اتصال سنسورها یا وسیله های دیگر ساختار مخصوص به آن باید رعایت شود که چگونگی این ساختار از روی دیتاشیت آن قطعه بدست می آید. به مثال زیر که در آن یک EEPROM راه اندازی می شود و توابع ساخته شده در آن توجه کنید.

مثال عملی شماره ۹ : برنامه ای برای ارتباط با EEPROM سریال AT24CXX بنویسید که از طریق ارتباط نرم افزاری I2C ، دیتاهایی دلخواه را در EEPROM ذخیره کند و سپس آن ها را روی LCD کاراکتری نمایش دهد.

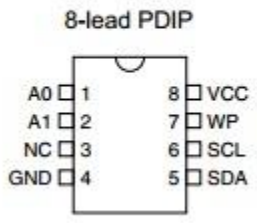
حل :

۱۰-۷-۳- معرفی آی سی های سری AT24CXX

این سری که ساخت شرکت اتمل می باشد ، یک حافظه EEPROM به حجم های ۲ ، ۴ ، ۸ ، ۱۶ ، ۶۴ ، ۱۲۸ ، ۲۵۶ ، ۵۱۲ و ۱۰۲۴ کیلوبایت را به صورت ارتباط I2C در اختیار کاربر قرار می دهد. عددی که در آخر نام قطعه مشاهده می شود نشان دهنده حجم آن می باشد برای مثال آی سی AT24C512 دارای ۵۱۲ کیلوبایت حافظه است. در شکل زیر پایه های این سری را مشاهده می کنید.

Pin Configurations

Pin Name	Function
A0–A1	Address Inputs
SDA	Serial Data
SCL	Serial Clock Input
WP	Write Protect
NC	No Connect



سه پایه A0,A1,A2 آدرس قطعه را مشخص می کند که شرکت اتمل برای اینکه امکان استفاده از چندین حافظه EEPROM در یک گذرگاه I2C را داشته باشیم ، آن را تعبیه کرده است. بنابراین تا ۸ آی سی حافظه را میتوان روی گذرگاه قرار داد.

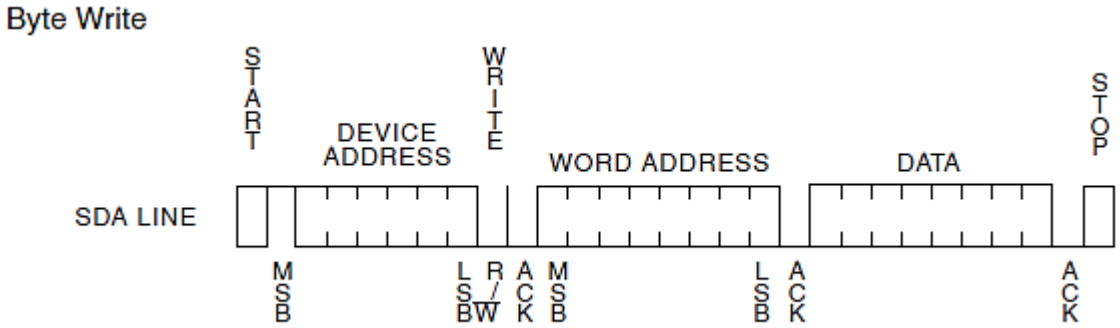
نکته : برخی از حافظه های سری AT24CXX سه آدرس قطعه ندارند و به جای آن دو یا یک یا صفر بیت آدرس دارند.

پایه حفاظت از نوشتن (WP) در صورتی که ولتاژ High داشته باشد ، دیگر نمیتوان در آی سی نوشت.

۱۰-۷-۴- عملیات نوشتن در آی سی EEPROM

طبق گفته دیتاشیت ، برای نوشتن در این آی سی از پروتکل I2C به دو صورت خاص استفاده می شود. اولی برای نوشتن یک بایت در آن و دومی نوشتن یک صفحه (page) که متشکل از چندین بایت پشت سر هم ، می باشد.

نوشتن به صورت بایتی



در نوشتن به صورت بایتی ابتدا آدرس قطعه مشخص می شود ، سپس آدرس خانه ای از حافظه که میخواهیم در آن بنویسیم مشخص می شود و در نهایت دیتای مورد نظر برای ذخیره ارسال می شود. سیگنال DEVICE ADDRESS خود به صورت زیر می باشد :

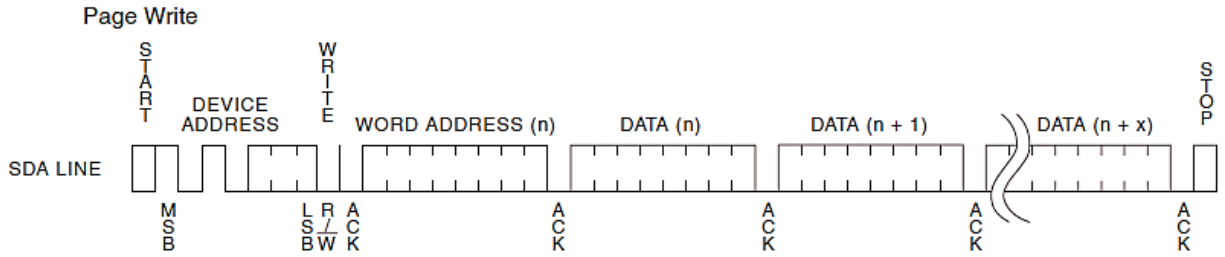
Device Address

2K	1	0	1	0	A_2	A_1	A_0	R/W
	MSB							LSB
4K	1	0	1	0	A_2	A_1	P0	R/W
8K	1	0	1	0	A_2	P1	P0	R/W
16K	1	0	1	0	P2	P1	P0	R/W

در شکل فوق که سیگنال DEVICE ADDRESS را برای حافظه های ۲ تا ۱۶ کیلوبایت مشاهده می کنید. برای بقیه حافظه ها نیز مشابه همین می باشد. به طوری که در همه آن ها ابتدا 1010 ارسال می شود و

سپس آدرس قطعه به صورت A2,A1,A0 ارسال می شود. برای قطعاتی که دو یا یک یا صفر بیت آدرس دارند 0 یا 1 بودن آن اهمیتی ندارد.

نوشتن به صورت صفحه ای



نوشتن صفحه ای همانند نوشتن بایتی است با این تفاوت که بعد از ارسال بایت اول ، بایت های دیتاهای بعدی (که در خانه بعد از WORD ADDRESS(N) در حافظه قرار خواهند گرفت) ارسال می شود و سپس سیگنال STOP ارسال خواهد شد.

۱۰-۷-۵- تابع نوشتن بایتی روی EEPROM سری AT24CXX

با توجه به ساختار گفته شده برای نوشتن بایتی روی آی سی میتوان تابع WRITE_I2C را به صورت زیر تعریف کرد :

```
void write_eeprom(unsigned char data,unsigned int address)
{
  i2c_start();
  i2c_write(write_address_bus);
  i2c_write(address);
  i2c_write(data);
  i2c_stop();
  delay_ms(10);
}
```

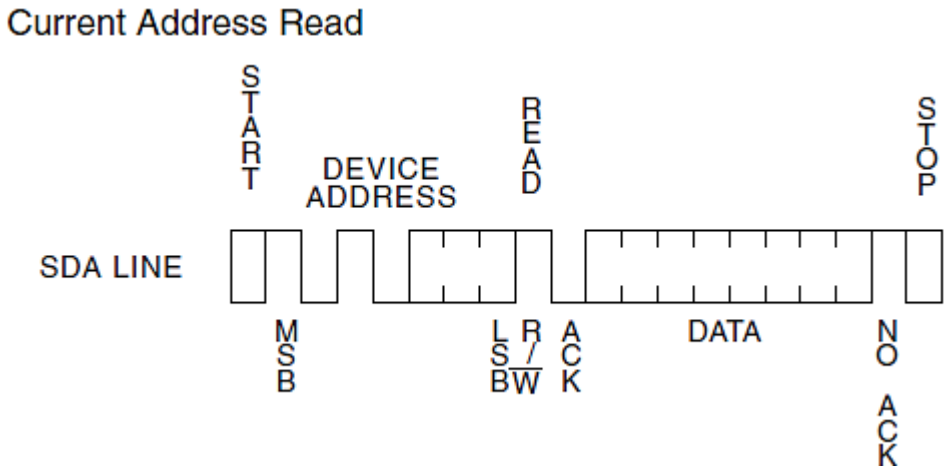
که در آن write_address_bus همان DEVICE ADDRESS می باشد که برای A2=0,A1=0,A0=0 برابر 160 است.

۱۰-۷-۶- عملیات خواندن از آی سی EEPROM

طبق گفته دیتاشیت ، خواندن از EEPROM به سه صورت امکان پذیر است :

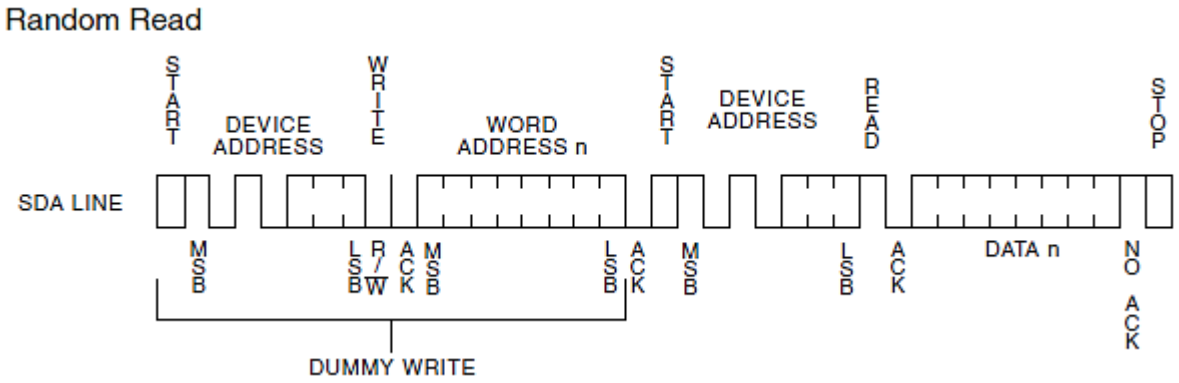
۱. خواندن از آدرس فعلی
۲. خواندن از آدرس مورد نظر
۳. خواندن متوالی

۱۰–۷–۷– خواندن از آدرس فعلی



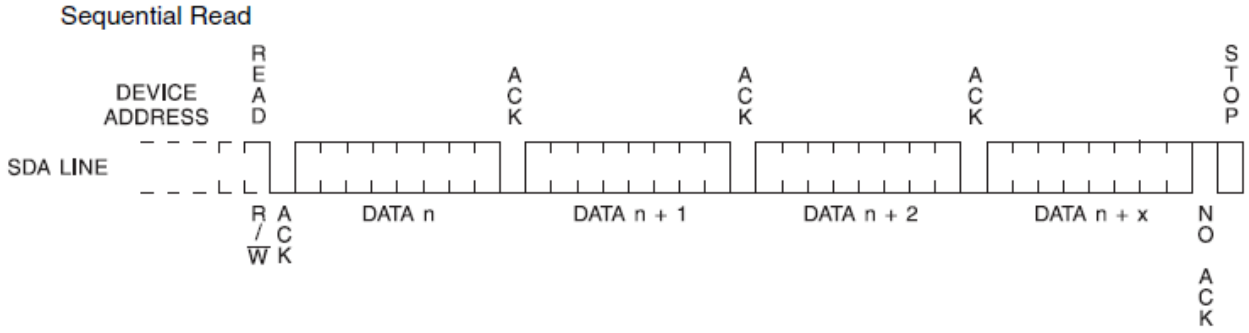
در این حالت دیتای آخرین آدرسی که وجود دارد برای Master ارسال می شود .

۱۰–۷–۸– خواندن از آدرس مورد نظر



در این حالت ابتدا آدرس قطعه با بیت R/W=0 به Slave ارسال شده و سپس آدرس خانه حافظه مورد نظر ارسال می گردد. سپس دوباره آدرس قطعه این بار با R/W=1 ارسال شده و سپس دیتایی که در آدرس فرستاده شده بود برای Master ارسال می گردد.

۱۰–۷–۹– خواندن متوالی



در این حالت از آخرین آدرس موجود یکی یکی دیتا ها را برای Master ارسال می کند.

۱۰-۷-۱۰- تابع خواندن از آدرس مورد نظر در EEPROM سری AT24CXX

با توجه به ساختار گفته شده برای خواندن از آی سی میتوان تابع READ_I2C را به صورت زیر تعریف کرد :

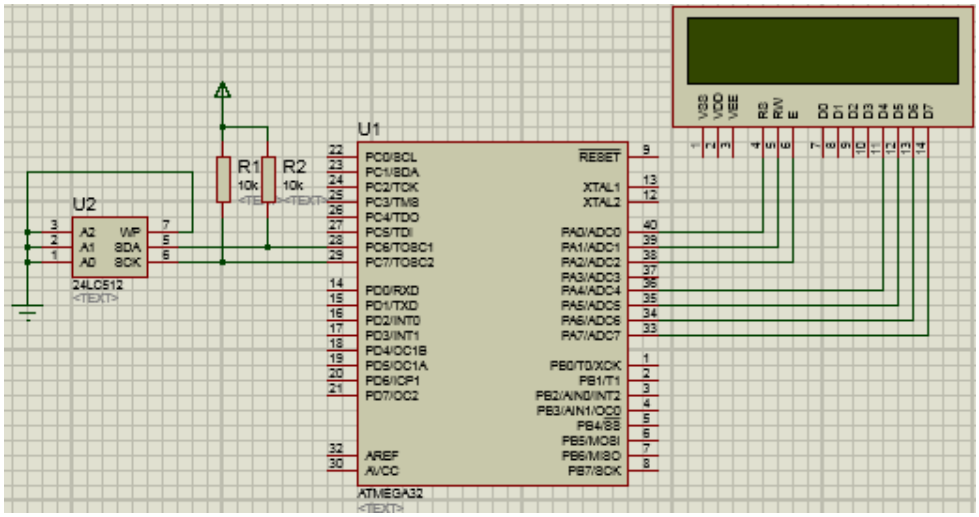
```
unsigned char read_eeprom(unsigned int address)
{
  unsigned char data_read;
  i2c_start();
  i2c_write(write_address_bus);
  i2c_write(address);
  i2c_start();
  i2c_write(read_address_bus);
  data_read=i2c_read(0);
  i2c_stop();
  return data_read;
}
```

که در آن write_address_bus همان DEVICE ADDRESS می باشد که برای A2=0,A1=0,A0=0 برابر 160 و read_address_bus با R/W=1 و برابر 161 است.

حل مثال شماره ۹ :

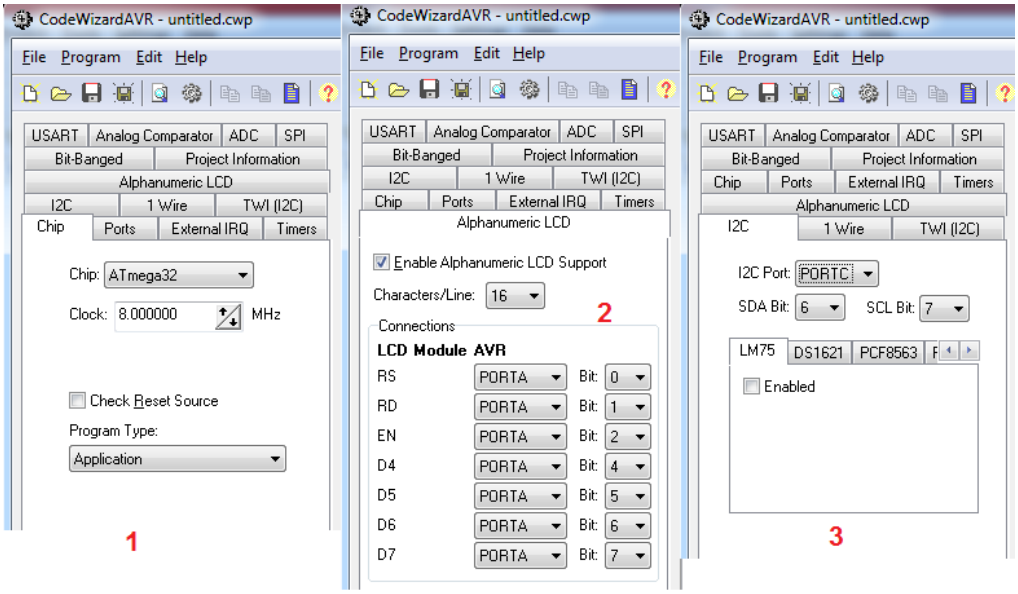
مرحله اول : طراحی سخت افزار

در این مرحله LCD و آی سی EEPROM با هر حجم دلخواهی را از کتابخانه آورده و به هر پورت دلخواهی از میکرو متصل می نماییم. در اینجا از 24C512 استفاده کردیم.



مرحله دوم : طراحی نرم افزار

در شکل زیر تنظیمات کدویزارد برای این مثال را مشاهده می کنید.



بعد از تولید کد توسط کدویزارد و حذف کدهای غیر ضروری برنامه اصلی را به صورت زیر خواهیم داشت :

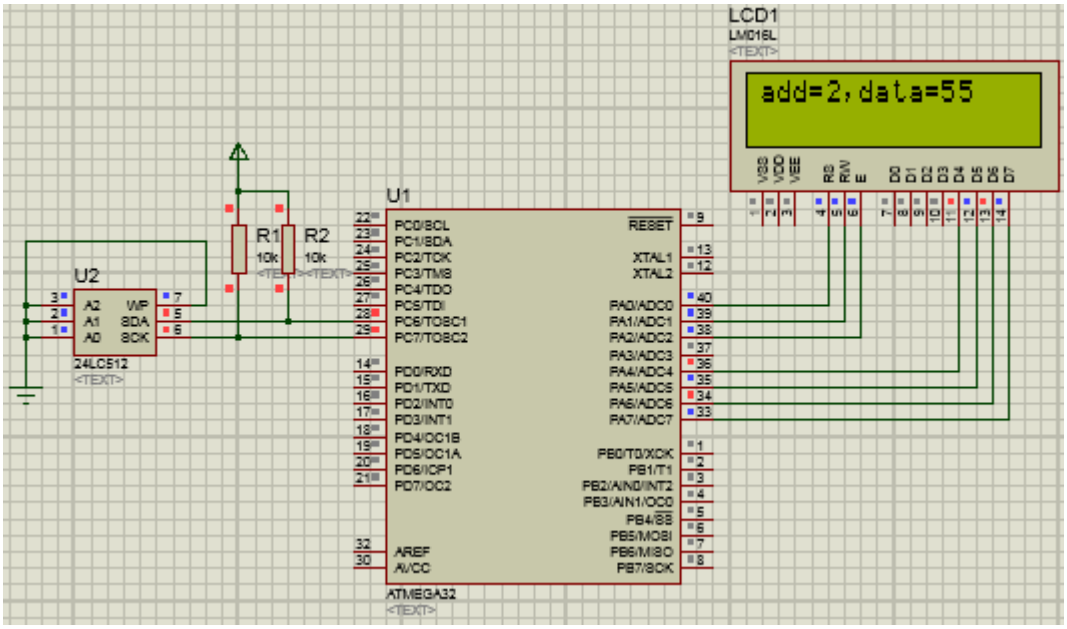
```
#include <mega32.h>
#include <delay.h>
#include <stdio.h>
// I2C Bus functions
#asm
   .equ __i2c_port=0x15 ;PORTC
   .equ __sda_bit=6
   .equ __scl_bit=7
#endasm
#include <i2c.h>
// Alphanumeric LCD Module functions
#include <alcd.h>

#define write_address_bus  160
#define read_address_bus  161

char buffer[20];
void write_eeprom(unsigned char data,unsigned int address);
unsigned char read_eeprom(unsigned int address);

void main(void)
{
unsigned char read_data,write_data;
int i;
// I2C Bus initialization
i2c_init();
// Alphanumeric LCD initialization
// Connections specified in the
// Project|Configure|C Compiler|Libraries|Alphanumeric LCD menu:
// RS - PORTA Bit 0
// RD - PORTA Bit 1
// EN - PORTA Bit 2
```

```
// D4 - PORTA Bit 4
// D5 - PORTA Bit 5
// D6 - PORTA Bit 6
// D7 - PORTA Bit 7
// Characters/line: 16
lcd_init(16);

write_data=55;
for(i=0;i<100;i++){
write_eeprom(write_data,i);
read_data=read_eeprom(i);
lcd_clear();
sprintf(buffer,"add=%d,data=%d",i,read_data);
lcd_puts(buffer);
delay_ms(100);
}
while (1);
}
//------------------------
void write_eeprom(unsigned char data,unsigned int address)
{
  i2c_start();
  i2c_write(write_address_bus);
  i2c_write((address & 0xff00)>>8);  //high byte address
  i2c_write((address & 0x00ff));     //low byte address
  i2c_write(data);
  i2c_stop();
  delay_ms(10);
}
//--------------------------
unsigned char read_eeprom(unsigned int address)
{
  unsigned char data_read;
  i2c_start();
  i2c_write(write_address_bus);
  i2c_write((address & 0xff00)>>8);  //high byte address
  i2c_write((address & 0x00ff));     //low byte address
  i2c_start();
  i2c_write(read_address_bus);
  data_read=i2c_read(0);
  i2c_stop();
  return data_read;
}
```

توضیح برنامه :

در این برنامه یک متغیر دلخواه ۸ بیتی با مقدار ۵۵ در همه خانه های حافظه EEPROM از آدرس ۰ تا ۹۹ ریخته می شود و سپس دوباره از آی سی خوانده شده و در آرایه کاراکتری buffer برای نمایش در LCD ریخته می شود. توابع خواندن و نوشتن در آی سی EEPROM به علت ۵۱۲ کیلو بایتی بودن آن تغییر کوچکی کرده است به طوری که پهنای آدرس ۱۶ بیتی است و در دو مرحله برای آی سی باید ارسال شود.

مرحله سوم : شبیه سازی در پروتئوس



[دانلود مثال عملی شماره ۹]

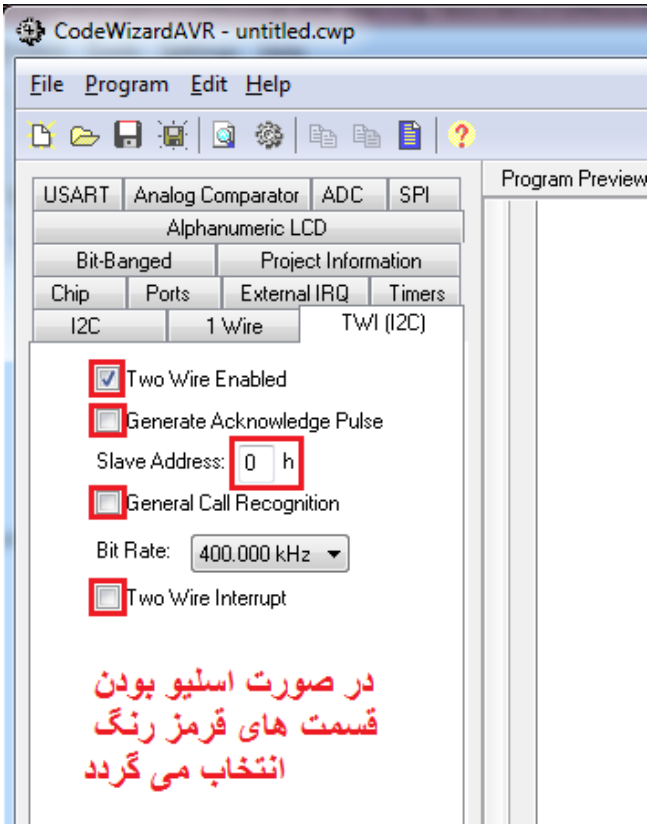
۱۰-۸- راه اندازی I2C سخت افزاری

برای فعالسازی واحد ارتباطی دو سیمه پس از رفتن به سربرگ TWI ابتدا آن را با زدن تیک مربوطه فعال می کنیم. در صورتی که میکروکنترلر در یکی از مدهای SR/ST قرار خواهد داشت ، انتخاب گزینه General Call Recognition موجب پاسخگویی این میکروکنترلر به فراخوانی عمومی (در آدرس 00Hex) می گردد. برای پاسخ گویی معمولی میکروکنترلر در یکی از مدهای SR/ST باید آدرس آن را در بخش Slave Adress تنظیم نمود.

انتخاب گزینه General Acknowledge موجب تولید پالس ACK در سه حالت زیر خواهد شد :

۱. Slave آدرس خود را تشخیص داده باشد
۲. یک فراخوانی عمومی دریافت شود (باید گزینه General Call Recognition فعال بوده باشد)
۳. داده جدیدی در مد MR یا SR دریافت شود

از بخش Bit Rate نیز جهت تنظیم حداکثر فرکانس پالس های روی خط SCL استفاده می شود. حداکثر فرکانس قابل انتخاب ۴۰۰ کیلو هرتز است. اگر نیاز به فعال کردن وقفه سریال دوسیمه داشته باشید گزینه 2wire Interrupt را فعال نمایید تا تابع سابروتین وقفه زیر در ابتدای برنامه اضافه شود.



۱۰-۸-۱- نحوه استفاده از واحد TWI سخت افزاری

برای استفاده از ارتباط I2C سخت افزاری ابتدا شبکه بندی دستگاه ها را به صورت استاندارد انجام داده و سپس تنظیمات کدویزارد را برای برنامه مورد نظر انجام می دهیم. حال در برنامه تولید شده توسط کدویزارد می بایست با رجیسترهای واحد TWI کار کرد. همانند توابعی که در قسمت قبل برای I2C نرم افزاری وجود داشت ، برای I2C سخت افزاری نیز وجود دارد که از آنها استفاده خواهیم کرد.

۱۰-۸-۲- معرفی رجیسترهای واحد TWI

۱. **TWAR** : مخفف TWI Address Register می باشد. یک رجیستر ۸ بیتی است که ۷ بیت پر ارزش آن آدرس Slave و بین 0 تا 127 است. همچنین بیت کم ارزش آن برای فعال/غیرفعال کردن قابلیت General Call یا فراخوانی عمومی است.

۲. **TWBR** : مخفف TWI Bit Rate Register می باشد. عددی که در این رجیستر قرار می گیرد طبق رابطه زیر فرکانس کلاک کاری واحد TWI (فرکانس SCL) را مشخص می کند.

$$SCL\ freq = \frac{CPU\ clock\ freq.}{16+2(TWBR).(4^{TWPS})}$$

۳. TWCR : مخفف TWI Control Register می باشد. بیت های این رجیستر به صورت شکل زیر می باشد.

Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

بیت TWIE (TWI Interrupt Enable) : فعال/غیرفعال کردن وقفه واحد TWI

بیتTWEN (TWI Enable) : فعال/غیرفعال کردن کل واحد TWI

بیتTWWC (TWI Write Collision) : اگر به هنگام یک بودن بیت TWINT اقدام به نوشتن بر روی بیت ثبات TWDR کنیم این بیت یک می شود.در صورت نوشتن بر روی TWDR وقتی که TWINT صفر است، این بیت صفر می شود.

بیتTWSTO (TWI Stop Condition Bit) : اگر در حالت Master باشیم با یک کردن این بیت یک وضعیت پایان ارسال می شود. به هنگام ارسال حالت آغاز این بیت به صورت سخت افزاری صفر می شود.

بیتTWATA (TWI Start Condition Bit) : اگر درحالت Master باشیم و این بیت را یک کنیم، در صورتی که گذرگاه آزاد باشد حالت آغاز ارسال می شود.

بیتTWEA (TWI Enable Acknowledge) : یک کردن این بیت باعث فعال شدن تایید دریافت یا ACK می شود.

نکته : اگر این بیت را صفر کنیم دستگاه در هیچ حالتی تایید دریافت ارسال نخواهد کرد، گویی از خط جدا شده است.

بیتTWINT (TWI Interrupt) : وقتی که سخت افزار واحد TWI وظیفه جاری خود را به پایان برساند این بیت ۱ می شود. چنانچه وقفه فعال باشد یک شدن این بیت باعث اجرای وقفه TWI می شود. با صفر کردن این بیت واحد TWI آغاز به کار می کند. دسترسی به ثبات های TWDR,TWSRو TWCR باید قبل از صفر کردن این بیت انجام شود. اگر در زمان یک بودن این بیت مقدار ثبات TWDR را تغییر دهیم تداخل به وجود می آید و بیت TWWCیک می شود. برای صفر کردن این بیت باید مقدار یک را درون آن بنویسیم.

۴. **TWDR :** مخفف TWI Data Register می باشد. در این رجیستر آخرین داده دریافت شده قرار می گیرد. همچنین در حالت ارسال برای ارسال داده باید داده را در داخل آن قرار دهیم. تنها زمانی که مقدار TWINT یک است میتوان به این رجیستر دسترسی داشت.

۵. **TWSR :** مخفف TWI Status Register می باشد. پنج بیت از این ثبات جهت نمایش وضعیت گذرگاه و وضعیت داخلی واحد TWI اختصاص دارد و دو بیت اول بیت های پیش تقسیم کننده پالس ساعت هستند. برای آنکه بتوانیم مقدار بیت های وضعیت را بدست آوریم باید دو بیت کم ارزش را صفر در نظر بگیریم.

۱۰-۸-۳- راه اندازی TWI در میکروکنترلر Master (هدر فایل twi_master.h)

برای استفاده راحت تر از قابلیت های واحد TWI در میکروکنترلر Master یک کتابخانه با نام twi_master.h ایجاد کردیم. با اضافه کردن این هدر فایل توابع زیر در برنامه قابل استفاده هستند.

۱. تابع twi_start()

با اجرای این تابع یک وضعیت Start ایجاد می شود.

۲. تابع twi_write(unsigned char)

این تابع ورودی خود را به Slave ارسال می کند.

۳. تابع unsigned char twi _read()

این تابع ابتدا وضعیت ACK را برای دریافت داده بوجود آورده و سپس دیتا را از Slave دریافت می کند و به خروجی تابع می فرستد.

۴. تابع twi_stop()

با اجرای این تابع یک وضعیت stop بر روی گذرگاه I2C ایجاد می شود.

تابع مد Master در حالت گیرنده (MR)

این تابع آدرس دستگاه Slave را به عنوان ورودی می پذیرد و منتظر دریافت داده از Slave مورد نظر می ماند. سپس دیتای مورد نظر را به خروجی تابع بر می گرداند.

```
unsigned char MR_TWI(unsigned char address_device)
{
unsigned char  data;
twi_start();
twi_write((address_device<<1)|0x01);
data = twi_read();
twi_stop();
return data;
}
```

تابع مد Master در حالت فرستنده (MT)

این تابع که یک آدرس و یک دیتا در ورودی خود دریافت می کند ، ابتدا آدرس دستگاه Slave و سپس دیتای مورد نظر را به گذرگاه I2C ارسال می کند.

```
void MT_TWI(unsigned char data,unsigned char address_device)
{
twi_start();
twi_write((address_device<<1)&0xFE);
twi_write(data);
twi_stop();
}
```

۱۰-۸-۴- راه اندازی TWI در میکروکنترلر Slave

برای هر تعداد میکرو کنترلر اسلیوی که در پروژه وجود دارد باید آدرس منحصر به فردی را در ابتدای برنامه به صورت دلخواه برای آن وارد نمود. برای این منظور #define Slave_address در برنامه وجود دارد. همچنین لازم است وقفه TWI آن در هنگام تنظیمات کدویزارد فعال شود. بعد از فعال شدن وقفه در زیر برنامه وقفه کد زیر نوشته می شود. در زیر برنامه وقفه متغیر TWIRecievedData مقدار دهی می شود که از آن در جاهای دیگر برنامه استفاده می شود.

تعریف وضعیت ها در میکروکنترلر Slave

کلیه وضعیت های موجود در پروتکل TWI که در رجیستر TWSR وجود دارد برای راحتی به عنوان ثابت در ابتدای برنامه تعریف می شود. چون همیشه به این ثوابت در برنامه احتیاج داریم ، آن ها را درون هدر فایل twi_slave.h می ریزیم.

```
#define Slave_address          0x45
#define START                  0x08
// Slave Transmitter
#define ST_SLA_ACK             0xA8
#define ST_ARB_LOST_SLA_ACK    0xB0
#define ST_DATA_ACK            0xB8
#define ST_DATA_NACK           0xC0
#define ST_LAST_DATA           0xC8
 // Slave Receiver
#define REP_START              0x10
#define SLA_R                  0xC9
#define SR_SLA_ACK             0x60
#define SR_ARB_LOST_SLA_ACK    0x68
#define SR_GCALL_ACK           0x70
#define SR_ARB_LOST_GCALL_ACK  0x78
#define SR_DATA_ACK            0x80
#define SR_DATA_NACK           0x88
#define SR_GCALL_DATA_ACK      0x90
#define SR_GCALL_DATA_NACK     0x98
#define SR_STOP                0xA0
```

زیر برنامه وقفه TWI در میکروکنترلر Slave

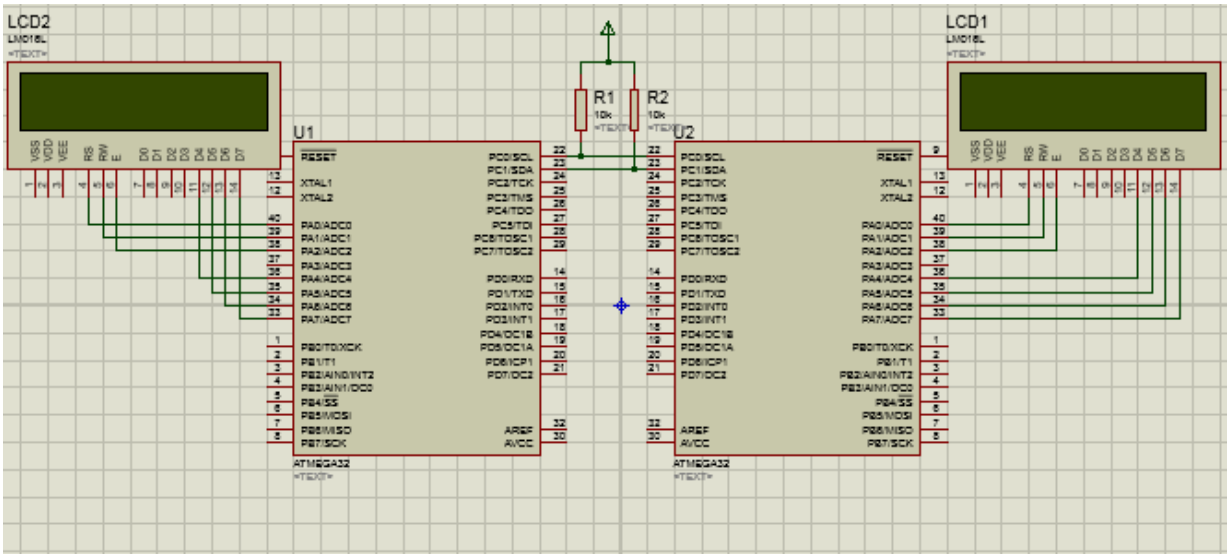
```
interrupt [TWI] void twi_isr(void)
{
        unsigned char  status;
        status = TWSR & 0xF8;
        switch(status)
  {
   case SR_GCALL_DATA_ACK:
      TWIReceiveData = TWDR;
       break;
   case SR_GCALL_DATA_NACK:
     TWCR=(1<<TWINT);break;
```

```
    case SR_DATA_ACK:
        TWIReceiveData = TWDR;
                break;
    case ST_SLA_ACK:
        TWDR=44;
       TWCR|=(1<<TWINT);
        break;
  }
         TWCR|=(1<<TWINT);
}
```

مثال عملی شماره ۱۰ : برنامه ای برای ارتباط دو میکروکنترلر Master و Slave به صورت شبکه ، از طریق واسط TWI بنویسید ، به طوری که یک عدد برای یکدیگر ارسال کرده و عدد ارسالی روی LCD طرف دیگر نمایش داده شود.

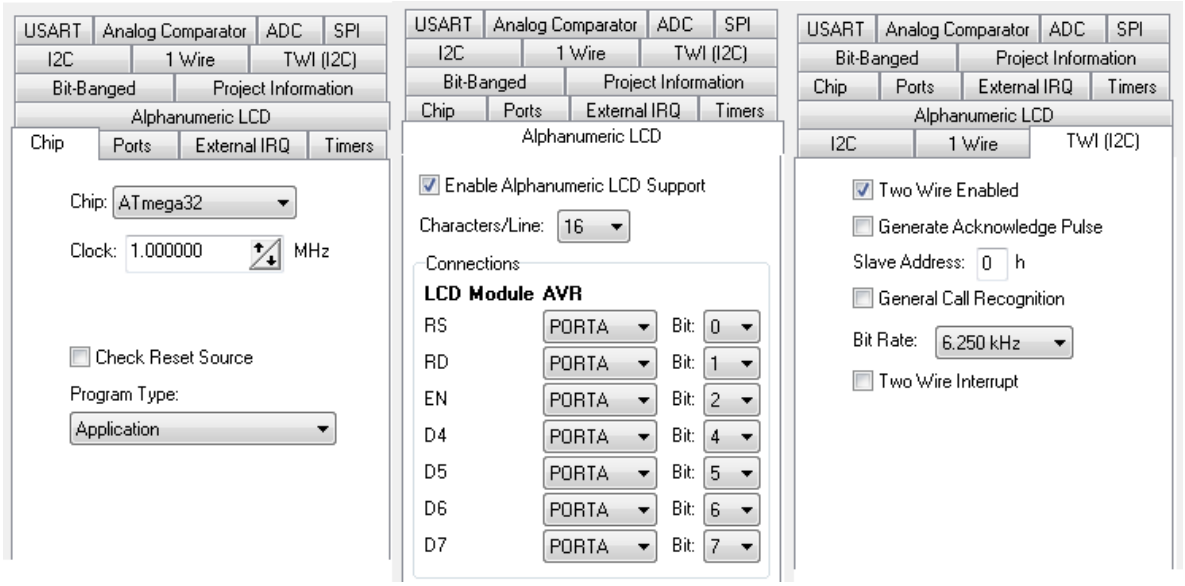
حل :

مرحله اول : رسم سخت افزار در پروتئوس



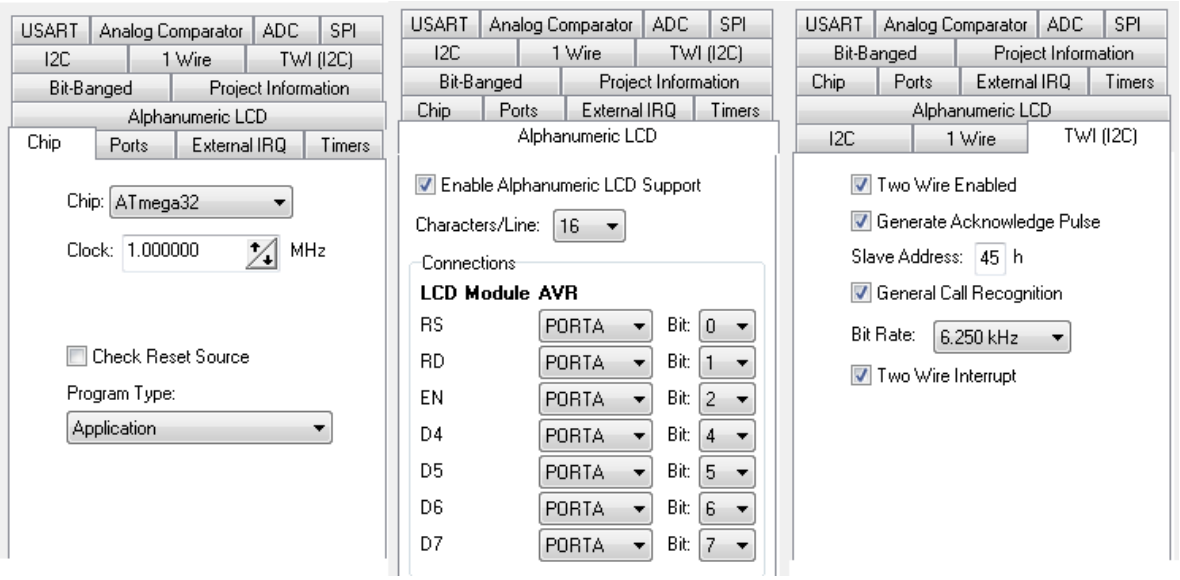
مرحله دوم : تنظیمات کدویزارد

برای میکروکنترلر Master



برای میکروکنترلر Slave

یک آدرس دلخواه مثلا 45 قرار می دهیم.



مرحله سوم : تکمیل برنامه

برای میکروکنترلر Master :

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

unsigned char MR_TWI(unsigned char address_device)
{
unsigned  char   data;
twi_start();
twi_write((address_device<<1)|0x01);
data = twi_read();
twi_stop();
return data;
}

void MT_TWI(unsigned char data,unsigned char address_device)
{
twi_start();
twi_write((address_device<<1)&0xFE);
twi_write(data);
twi_stop();
}

char buffer[50];
unsigned char Receivedata,Senddata,Slaveaddress;
```

```
void main()
{

// TWI initialization
// Bit Rate: 6.250 kHz
TWBR=0x48;
// Two Wire Bus Slave Address: 0x0
// General Call Recognition: Off
TWAR=0x00;
// Generate Acknowledge Pulse: Off
// TWI Interrupt: Off
TWCR=0x04;
TWSR=0x00;

// Alphanumeric LCD initialization
// Connections specified in the
// Project|Configure|C Compiler|Libraries|Alphanumeric LCD menu:
// RS - PORTA Bit 0
// RD - PORTA Bit 1
// EN - PORTA Bit 2
// D4 - PORTA Bit 4
// D5 - PORTA Bit 5
// D6 - PORTA Bit 6
// D7 - PORTA Bit 7
// Characters/line: 16
lcd_init(16);

Senddata=55;
MT_TWI(Senddata,GlobalAddress);
delay_ms(200);

Slaveaddress=0x45;
Senddata=10;
MT_TWI(Senddata,Slaveaddress);
delay_ms(200);

Slaveaddress=0x45;
Receivedata=MR_TWI(Slaveaddress);
lcd_gotoxy(0,0);
sprintf(buffer,"data=%d",Receivedata);
lcd_puts(buffer);

while (1)
    {
    // Place your code here

    }
}
```

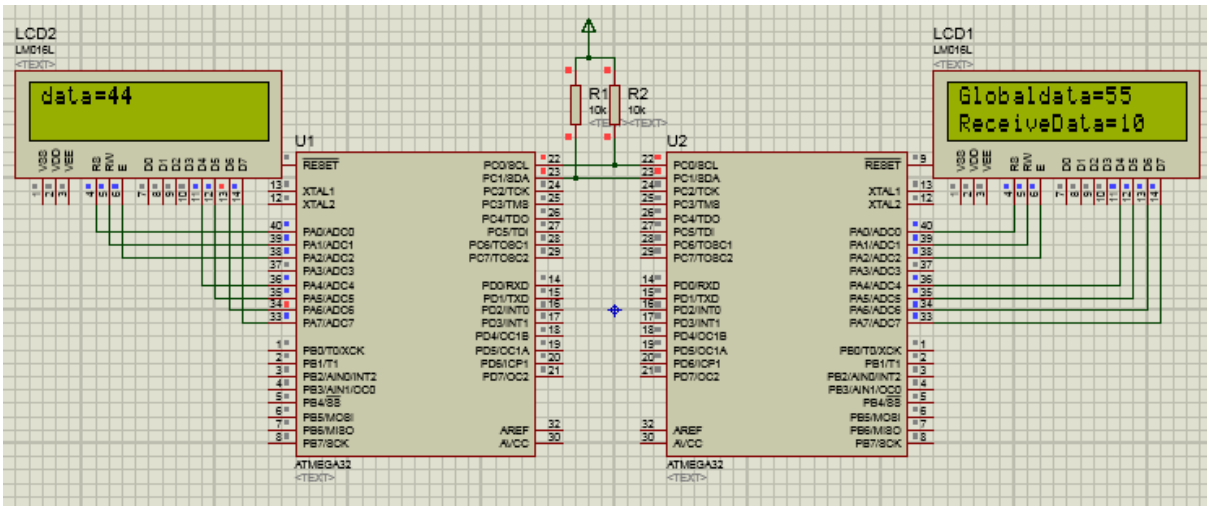
برای میکروکنترلر Slave :

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

#define Slave_address           0x45

char TWIReceiveData;
char buffer[50];

// 2 Wire bus interrupt service routine
interrupt [TWI] void twi_isr(void)
{
   unsigned char  status;
   #asm("cli")
   status = TWSR & 0xF8;
   switch(status)
  {
    case SR_GCALL_DATA_ACK:
        TWIReceiveData = TWDR;
        sprintf(buffer,"Globaldata=%d",TWIReceiveData);
        lcd_puts(buffer);
        break;

    case SR_GCALL_DATA_NACK:
        TWCR=(1<<TWINT);
        break;

    case SR_DATA_ACK:
        TWIReceiveData = TWDR;
        sprintf(buffer,"ReceiveData=%d",TWIReceiveData);
        lcd_gotoxy(0,1);
        lcd_puts(buffer);
        break;

    case ST_SLA_ACK:
        TWDR=44;
        TWCR|=(1<<TWINT);
        break;
  }
  TWCR|=(1<<TWINT);
  #asm("sei")
}

void main()
```

```
{

// TWI initialization
// Bit Rate: 6.250 kHz
TWBR=0x48;
// Two Wire Bus Slave Address: 0x45
// General Call Recognition: On
TWAR=0x8B;
// Generate Acknowledge Pulse: On
// TWI Interrupt: On
TWCR=0x45;
TWSR=0x00;

// Alphanumeric LCD initialization
// Connections specified in the
// Project|Configure|C Compiler|Libraries|Alphanumeric LCD menu:
// RS - PORTA Bit 0
// RD - PORTA Bit 1
// EN - PORTA Bit 2
// D4 - PORTA Bit 4
// D5 - PORTA Bit 5
// D6 - PORTA Bit 6
// D7 - PORTA Bit 7
// Characters/line: 16
lcd_init(16);

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



while (1)
    {
    // Place your code here


    }
}
```

توضیح برنامه : در برنامه Master طبق پروتکل I2C ، ابتدا عدد ۵۵ به تمامی Slave ها (فراخوان عمومی) ارسال می شود. سپس به میکروکنترلر Slave که آدرس آن 0x45 است ، دیتای ۱۰ ارسال می شود و در نهایت میکروکنترلر Master دیتایی را از Slave دریافت کرده و روی LCD نمایش می دهد. میکروکنترلر Slave نیز طبق پروتکل I2C عمل می کند. تمام برنامه Slave درون تابع وقفه اتفاق می افتد. به طوری که در هر وقفه ، وضعیت کنونی میکرو شناسایی شده و طبق آن وضعیت به ارسال یا دریافت داده می پردازد. در صورت دریافت دیتا از جانب Master آن را روی LCD نمایش می دهد. در صورت درخواست Master برای ارسال دیتا عدد ۴۴ برای آن ارسال می گردد.

مرحله چهارم : شبیه سازی



دانلود سورس مثال عملی شماره ۱۰

پایان

امیدوارم مباحث مطرح شده مورد استفاده و توجه شما قرار گرفته باشد. برای شروع به کار عملی با میکروکنترلرهای AVR و خرید قطعات مورد نیاز به فروشگاه الکترو ولت به نشانی Shop.Electrovolt.ir مراجعه نمایید. همچنین پروژه های شبیه سازی و عملی پیشرفته تر را میتوانید از طریق لینک زیر خریداری نمایید. با تشکر شجاع داودی

خرید و دانلود آنلاین بسته عظیم آموزش میکروکنترلرهای AVR