به نام خدا

وب سایت آموزشی برنامه نویسی میکروکنترلرها

www.picpars.com

عنوان:

آموزش گام به گام برنامه نویسی AVR به زبان C

برنامه نویسی میکروکنترلرهای AVR با کامپایلر CodeVisionAVR

نویسنده: سید محسن قاسمیان

بهار ۱۳۹۱

چکیــده

با توجه به درخواست های مکرر کاربران مبنی بر اینکه آموزش ها در یک فایل PDF جمع آوری و منتشر شوند، به همین دلیل ما نیز تصمیم گرفتیم تمامی مطالب و آموزش های موجود را در این فایل جمع آوری کنیم و همراه با فایل های شبیه سازی شده و سورس کد ها ضمیمه این مقاله کنیم. بنابراین تمامی کاربران می توانند با مراجعه به سایت جدید ترین نسخه این مقاله را دانلود نمایند. آدرس دانلود و سایر مشخصات مقاله در آخرین صفحه همین مقاله موجود می باشد.

کلمات کلیدی

میکروکنترلر avr ، زبان سی، کدویژن، برنامه نویسی، دستورات کنترلی، توابع، کار با السیدی، کیبورد ۴×۴، سون سگمنت، تایمر، تایمر صفر، وقفه تایمر، مبدل آنالوگ به دیجتال، سنسور دمای LM35،

While, for, switch, break, continue ،if ،define, LCD, sprintf, keypad, 7segment, timer, timer interrupt, ADC, LM35,

فهرست مطالب

با کلیک کردن روی هر یک از عنوان زیر به موضوع مربوطه هدایت می‌شوید.

فهرست ضمیمه‌ها

با کلیک کردن روی هر یک از عنوان زیر به لینک ضمیمه مربوطه هدایت می‌شوید.

فهرست جدول‌ها

مقدمه

وب سایت برنامه نویسی میکروکنترلرها (http://www.picpars.com) در تاریخ یکم بهمن ۱۳۸۸ با هزینه های شخصی اینجانب راه اندازی شد. اهداف این سایت آموزش رایگان، برنامه نویسی انواع میکروکنترلرها بخصوص سری AVR و آشنایی با تازه های دنیای برق و الکترونیک می باشد. تقاضای ما از تمامی کاربران این است که در نظر سنجی های سایت و بخش نظرات هر مطلب یا مقاله شرکت کنند و با نظرات سازنده خود ما را در بهبود کیفیت سایت یاری نمائید. همچنین می توانید از طریق لینک زیر مطلب، مقاله یا پروژه خود را برای ما ارسال نمائید تا با نام خودتان در سایت منتشر شود. ارسال مطلب

هدف

هدف از ارائه این مقاله آموزش و ارتقاع سطح علمی کاربران و علاقمندان به برنامه نویسی میکروکنترلرهای AVR به زبان C می باشد. به امید خداوند یکتا این مقاله با ارائه مطالب از سطح صفر تا سطح پیشرفته تمامی نیاز های کاربران را برطرف خواهد نمود. لذا از همین جا از تمامی افراد سطح بالا برای ارائه مطالب ساده و مبتدی پوزش می طلبیم و امیدواریم ما را تا رسیدن به مطالب سطح پیشرفته یاری نمایند.

توصیه ما به تمامی افراد مبتدی و تازه کار:

برای اینکه یک برنامه نویس حرفه ای شوید، هیچ گاه دستورات آماده را Copy-Paste نــکنید و مثال های آموزشی که همراه این مقاله ارائه می شوند را مجددا خودتان خط به خط تایپ و برنامه تان را کامپایل و شبیه سازی کنید، در این صورت هم سرعت برنامه نویسی تان بالا خواهد رفت و هم تجربه های باورنکردنی بدست خواهید.

شروع یک پروژه

ایجاد یک پروژه جدید در CodeVisionAVR

ابتدا برنامه CodeVisionAVR را اجرا کنید سپس از منوی File ➔ New را کلیک کنید در پنجره ظاهر شده گزنیه project را انتخاب و OK کنید و در پنجره confirm گزینه Yes را کلیک کنید. در پنجره CodeWizardAVR گزینه AT90,ATtiny,ATmega,FPSLIC انتخاب و Ok کنید. اکنون پنجره CodeWizardAVR جلوی شماست این پنجره که به جادوگر کد معروف است به شما این امکان را می دهد که با وارد کردن یه سری اطلاعات از جمله مدل میکرو فرکانس کاری تعین پورت LCD، پیکره بندی پورت ها، تایمرها کانترها و ... کدهای مربوطه رو تولید کند که بعد از تنظیم کردن آنها از منوی File گزینه Generate, save and Exit رو کلیک کنید. و در پنجره بعدی اسمی برای فایل *.C وارد می کنیم و همین طور برای project و codeWizardAVR اسم وارد کنید. اکنون شما می توانید برنامه نویسی را در محیط CodeVisionAVR آغاز کنید.

نکته برنامه را حتما در فایل *.C که ایجاد کردید بنویسید و از تب Note هم برای نوشتن یاداشت یا نگهداری یک کپی از برنامه نوشته شده تان استفاده کنید.

در اینجا چون شما از CodeWizardAVR استفاده کردید می بینید که در فایل سی تعدادی دستور به صورت آماه و مطابق با تنظیمات ابتدایی شما ایجاد شده، توصیه من به تازه کارها این است که همه خطوط برنامه را پاک کنید، چون مقادیری که به رجیسترها نسبت داده شده است باعث اضطراب و سردرگمی شما در اولین برخورتان با این محیط خواهد شد. در گام های بعدی متوجه خواهید شد که این رجیستر ها و ... چه بودند.

حال فرض می کنیم برنامه ای نوشته اید خوب الان برای کامپایل کردن اون باید از منوی project و سپس گزینه Build All را کلیک کنید، اگر در برنامه شما خطای نوشتاری و یا سایر خطا ها موجود نباشد برنامه کامپایل شده، و فایل هگز HEX خروجی که برای شبیه سازی در پروتیوس نیاز داریم در پوشه Exe تولید می شود. (کامپایل = ترجمه برنامه به زبان ماشین)

مختصری درمورد شبیه ساز Proteus

ابتدا به معرفی ابزارهای مهم و کاربری برنامه می پردازیم:

ابزارهای سمت چپ :

Selection mode با آیکون ماوس از این ابزار برای انتخاب و جابجایی قطعات یا قسمتی از مدار استفاده می شود.

Component mode با یه آیکون مثلث شکل با کلیک کردن بر روی آن لیست قطعاتی که از کتابخانه انتخاب کرده اید را برای شما نمایش می دهد. همچنین می توانید قطعات مورد نیاز خود را از همین محل اضافه کنید.

Junction dot mode برای اتصال بر قرار کردن بین دو سیم استفاده می شود البته برنامه به طور اتوماتیک مسیر ها را اتصال (نقطه گذاری) می دهد.

Wire label mode برای برچسپ زدن عدد یا حروف بر روی یک سیم می باشد. یا نام گذاری آن سیم.

Text script mode به کمک آن می توانید یک متن یا توضیح مثلا راجب مدار بنویسید در محلی قرار دهید.

Terminals mode در این قسمت می توانید انواع ورودی و خروجی و تغذیه و زمین و ... را پیدا کنید.

Devise pins mode در این قسمت انوع پایه ها وجود دارد که شما می توانید از آنها برای ایجاد شماتیک یه قطعه خاص که در کتابخانه نیست استفاده کنید. همچنین برای ایجاد قطعه می توانید از subcircuit mode استفاده نمایید.

Generator mode در این قسمت انواع مولدها موجود می شود. از جمله مولدهای سینوسی، مربعی، پالسی، صوتی، کلاک پالس، dc و غیره...

voltage probe mode و current probe mode این دو پراب های ولتاژ و جریان هستند که به کمک آنها می توانید ولتاژ یا جریان یک سیم را مشاهد کنید. البته ولتاژ را به صورت WLO و WHI نشان می دهد. و در مورد جربان مقدار لحظه به لحظه آن را نشان می دهد.

virtual instrument mode ابزارهای واقعی: در این قسمت مهم ترین ابزارهای شبیه سازی یعنی فانکشن ژنراتور که یک مولد بسیار قوی با تنظیم دامنه و فرکانس و خروجی های مربعی و سینوسی مثلثی و دندان اره ای می باشد.

اسیلوسکوپ چهار کاناله با قابلیت تنظیم دامنه هر کانال.

انواع ولتمتر های DC AC و ابزارهای کاربردی دیگر...

ابزارهای سمت بالا:

بعد از ابزار های کپی و انتقال و برگشت به آخرین عمل انجام شده و غیره ... به ابزار های چرخش می رسیم با انتخاب کردن قطعه مورد نظر این ابزارها فعال می شود و شما می توانید آن را بچرخانید یا معکوس کنید. البته در سمت چپ هم این ابزار ها وجود

دارند با این تفاوت که ابزارهای چرخش که در سمت چپ هستند با اعمال تغیرات همیشه ثابت هستند یعنی اینکه اگر ۹۰درجه چرخش بدهیم بعد از آن هر قطعه ای که از لیست بخواهیم بیاوریم با چرخش ۹۰ درجه در صفحه ظاهر می شود. در قسمت معکوس هم همین می باشد.

Pick parts from library کتابخانه برنامه می باشد که با کلیک کردن بر روی آن پنجره ای باز می شود که در کادر اولی یعنی keywords می توانید یک قسمتی از نام قطعه را تایپ کنید تا برای شما جستجو کند و همچنین یک مربع خالی در زیر آن می باشد که اگر تیک بزنید بر اساس نام قطعه قطعاتی دیگر که نام آنها تا حدوی شبیه به هم هستند رو به شما معرفی می کند. البته کل قطعات در کتابخانه به صورت آرشیو موضوعی دسته بندی شده است و شما می توانید با کلیک کردن بر روی آن موضوع کلیه قطعات را مشاهده نمایید.

نکته به نوع قطعات در کتابخانه توجه کنید؛ ممکن است که در کتابخانه ما دو قطعه هم نام داشته باشیم که جلوی یکی عبارت active درج شده و در جلوی یکی دیگه عبارت Device و عبارت های دیگه... اگه هدف شما فقط شبیه سازی هست سعی کنید از قطعاتی که جلوی آنها عبارت active درج شده استفاده نمایید.

و یا اگر هدف شما شماتیک و بیشتر PCB هست از Device استفاده کنید. دلیل این کار این هست که قطعات active مدل PCB ندارد ولی اکثر قطعات Device و غیره ... مدل PCB دارند. حال مزیت این کار چیست؟ اگر شما از قطعاتی استفاده کنید که مدل PCB آن در کتابخانه موجود باشد به راحتی می توانید مدار را به PCB انتقال دهید و با انجام چند کار ساده به صورت اتوماتیک مسیرها را سیم کشی کنید و یک برد مدارچاپی تهیه کنید. برای اینکه متوجه شوید که PCB قطعه مورد نظر در کتابخانه وجود دارد به کادر مشکی در سمت راست توجه کنید اگه تصویر برای قطعه نشان داد یعنی اینکه موجود است و در غیره این صورت PCB ان قطعه موجود نمی باشد.

toggle wire autorouter اگر این گزینه را فعال کنید سیم کشی شما به صورت اتوماتیک زاویه بندی می شود و در غیره این صورت شما می توانید خودتان زاویه بندی کنید.

کار با نرم افزار و شروع یک شبیه سازی

بعد از اینکه قطعات مدار را از کتابخانه انتخاب کردید پنجره را ببندید و از لیست قطعات قطعه را انتخاب کنید و در صفحه به تعداد مورد نیاز کلیک کنید. بعد از چیدن قطعات باید آنها را سیم کشی کنید که برای این کار ماوس رو نزدیک پایه ها برده و یک کلیک چپ کند اکنون سیم شما به یک سر قطعه وصل شده می توانید ماوس رو حرکت بدهید و نزدیک سر دیگه قطعه ببرید اگر شما

زاویه بندی اتواتیک را فعال کنید وقتی ماوس را حرکت می دهد به صورت اتو ماتیک سیم بندی و زاویه بندی می شود. حال برای پاک کردن یه سیم می توانید ماوس را روی سیم ببرید و یک بار کلیک راست کنید و گزینه delete را بزنید و یا دو بار پشت سرهم روی سیم کلیک راست کنید تا سیم پاک شود.

بعد از سیم کشی برای شروع شبیه سازی می توانید از گزینه های پایین صفحه که شبیه به کلید های مالتی مدیا هستند استفاده کنید، برای شروع کلید Play را بزنید. اگر مدار شما از نظر قطعات و سیم کشی مشکلی نداشته باشد شبیه سازی شروع می شود و در غیر این صورت پنجره ای باز می شود و خطا ها را برای شما نمایش می دهد.

برای معرفی کد هگز (برنامه) به میکرو هم به صورت زیر عمل می کنیم:

بعد از سیم کشی مدار ماوس را روی میکرو برده و یک بار کلیک راست کنید و از منوی ظاهر شده گزینه Edit properties را انتخاب کنید. یا یک بار کلیک چپ کنید و یک بار دیگه کلیک راست کنید و یا قطعه را انتخاب کنید و کلید Ctrl+E را بزنید. بعد از باز شدن پنجره در قسمت program file آیکون پوشه را کلیک کنید و مسیر فایل هگز را مشخص کنید. در قسمت Clock frequency فرکانس کریستالی که می خواهید از آن در مدار استفاده کنید را تنظیم می کنید. نکته: فرکانسی که شما در اینجا به عنوان کلاک میکروکنترلر تعیین می‌کنید باید همان فرکانسی باشد که در کامپایلر انتخاب نموده اید، برنامه بر اساس این فرکانس کار می‌کند یعنی سرعت اجرای دستورات توسط میکرو بر اساس این فرکانس می باشد.

برای مشاهد سیگنال هر نقطه از مدار می توانید در قسمت virtual instrument mode اسیلوسکوپ OSCILLOSCOPE انتخاب کنید. این دستگاه چهار ورودی به نام های A,B,C,D داردو هر یک از سرها را می توانید به نقاط دلخواه وصل کنید. سرهایی که نمی خواهید از آنها استفاده نمایید را به چیزی وصل نکنید. بعد از وصل کردن نقاط اکنون گزینه Play را بزنید در این صورت یک پنجره که قسمتی از آن هم شطرنجی است برای شما باز می شود. در این پنجره همه چیز بر اساس رنگ می باشد یعنی اینکه رنگهای زرد و تنظیمات آن مختص سیگنال A می باشد. تنها چیزی که بین همه کانالها مشترک می باشد time division می باشد که با رنگ نارنجی مشخص شده.

برای اندازه گیری فرکانس به این صورت عمل می‌کنیم، کلید time division را آنقدر بچرخانید تا یک یا دو سیکل از موج را روی صفحه مشاهد کنید هرچه تعداد سیکل ها در صفحه کمتر باشد دقت اندازه گیری ما دقیق تر می شود. بعد از تنظیم کردن این مراحل یک ولوم دایره ای شکل بالای time division هست به نام position این ولوم کار ما را در اندازه گیری خیلی راحت می کند. شما می توانید با چرخاندن آن ابتدای شروع سیکل را به اولین خانه در سمت چپ ببرید، حال از اولین خانه سمت چپ تعداد

خانه های افقی که یک سیکل از موج در آن قرار گرفته را بشمارید این تعداد را در زمان تناوب time division ضرب کنید و عدد ۱ را بر این حاصل تقسیم کنید عدد بدست آمده فرکانس این موج می باشد.

برای اندازه گیری ولتاژ به صورت زیر عمل می کنیم:

اصول کار همانند اندازه گیری فرکانس می باشد با این تفاوت که باید تعداد خانه های عمودی یک سیکل را شمارش کنیم، یعنی از پیک پایینی تا پیک بالایی. و بعد این تعداد خانه ها را در رنج ولتاژ کانال مربوطه ضرب کنیم. این عدد به دست آمده ولتاژ پیک تا پیک سیگنال می‌باشد که اگر بر عدد ۲ تقسیم کنیم مقدار ولتاژ پیک بدست می آید.

اصول و قوانین برنامه نویسی به زبان C

مقدمات برنامه نویسی

در زبان C برای نوشتن یک توضیح در کنار دستورات و یا در هر محل دلخواه به دو روش می توانیم این کار رو انجام بدیم. ۱) اگر توضیحات فقط در یک سطر هست علامت // را در ابتدای توضیحات می نویسیم. مثال: //picpars.com این عبارت یک توضیح هست نه دستور C پس برای اینکه کامپایلر متوجه شود که این یک دستور نسیت باید به این صورت بنویسیم. ۲) اگر توضیحات بیشتر از یک سطر باشد باید در اولین سطر عبارت /* را نویسییم و در آخرین سطر هم عبارت */ را بنویسیم. مثال:

```
/*WEB: www.picpars.com
Email: picpars[at]gmail.com
designer: Seyed mohsen ghasemian*/
```

ساختمان یک برنامه زبان C :

برای نوشتن یک برنامه سی ابتدا باید توابع مورد نیاز را فراخوانی کنیم و بعد معرفی متغیرها و بعد هم تابع اصلی (main) را باید بنویسیم. در یک برنامه زبان سی تابع main لازم و ضروری است یعنی جزء ثابت می‌باشد. و نکته مهم دیگر اینکه در زبان سی بین حروف بزرگ و حروف کوچک تفاوت هست، یعنی شما می‌توانید متغیری به نام a و هم به نام A داشته باشد، که در اینجا یه مثال میزنیم:

```
#include <mega16.h>
Char a,A;
Void main(){
دستورات، حلقه ها، شمارنده و ...
}
```

نکات مهم: در زبان سی باید در انتهای هر دستور علامت ; (سیمی کالون) قرار دهیم تا دستورات از یکدیگر متمایز گردند. تابع main و سایر توابع دیگر هم با { شروع و با } پایان می پذیرد. در مواقعی که از دستورات کنترلی مانند شرط، حلقه، سوئیچ و ... استفاده می‌کنیم، اگر تنها یک دستور اجرای داشته باشیم نیازی به {} نیست. اما در صورتی که بیش از یک دستور داشته باشیم باشد دستورات در {} قرار دهیم. به مثال زیر توجه نمائید:

```
if(a==1) PORTC=55;

if(a==1){
PORTC=55;
PORTD=80;
}
```

در برنامه اولی چون یک دستور داشتیم از آکولاد استفاده نکردیم ولی در برنامه دومی چون بیش از یک دستور داشتیم ار آکولاد استفاده کردیم. همیشه یه آکولاد { باز می کنیم و باید حتما آن را با } ببندیم. برای نوشتن برنامه به زبان اسمبلی هم باید به صورت زیر عمل کنیم:

```
#asm
دستورات اسمبلی
#endasm
```

متغیرها، آرایه ها، رشته ها و مقدار دهی در زبان C

انواع داده ها (متغیرها):

Type	Size (Bits)	Range
Bit	1	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

جدول ۱ انواع داده ها

برای تعریف یک متغیر ابتدا نوع داده (Type) و سپس نام متغیر را می نویسیم:

```
Bit a;
char b;
```

بعد از تعریف نام متغیر حتما علامت ; را باید قرار داد. برای مقدار دهی اولیه یک متغیر به دو صورت می توانیم این کار را انجام دهیم:

۱- ابتدا متغیر رو تعریف کنیم و سپس آن متغیر را مقدار دهی کنیم.

۲- هنگامی که متغیر را تعریف می‌کنیم همزمان آن را مقدار دهی کنیم.

```
Char a,b,c;
ungined int j,k;
a=0x1E; b=0b00011110; b=30; j=44; k=90;

Char a=0x1E,b=0b00011110,b=30;
ungined int j=44,k=90;
```

به صورت پیش فرض مقادیر در مبنای دسیمال می باشند، اما در صورتی که بخواهیم مقدار در مبنای باینری وارد کنیم ابتدا باید 0b بنویسیم و بعد از b عدد باینری رو وارد کنیم، همچنین اگر بخواهیم مقدار را در مبنای هگزا دسیمال وارد کنیم باید اول 0x بنوسیم و بعد از آن مقدار هگز را وارد بنویسیم. **نکته:** در هر سطر فقط می توان یک نوع متغیر تعریف کرد!

آرایه ها:

آرایه ها مجموعه ای از متغیر های هم نوع هستند. برای تعریف آرایه ابتدا نوع داده، نام آرایه و تعداد عناصر آرایه را در داخل [] قرار می دهیم. متغیرهای آرایه‌ای کاربردهای فروانی دارند، استفاده از آنها در برخی برنامه ها می‌تواند به شدت حجم و پیچیدگی برنامه را کاهش دهد. با یک مثال این موضوع را بازتر میکنیم، به عنوان مثال شما رشته ریاضی فیزیک یک دبیرستان را در نظر بگیرید که چهار گروه می‌باشند و هر گروه تعداد دانش آموزان متفاوتی دارند، حال اگر ما بخواهیم آنها را به یک آرایه شبیه کنیم به این صورت می نویسیم ;{12,15,9,16}unsigned Char reyazifizek[4]= یعنی رشته ریاضی فیزیک نام آرایه با تعداد گروه‌های ۴ و تعداد دانش آموزان هر گروه نیز مشخص شده است.

آرایه تک بعدی:

```
unsigned Char a[4]={22,45,76,90};

unsigned Char a[4];
a[0]=22; a[1]=45; a[2]=76; a[3]=90;
```

نام آرایه a و از نوع ۸ بیتی unsigned Char که دارای ۴ عضو می باشد. برای دسترسی به عضوهای یک آرایه باید شماره عضو را درون [] بنویسیم. نکته قابل توجه این است که اولین عضو یک آرایه با شماره صفر شروع می شود. مانند مثال بالا

آرایه دو بعدی:

آرایه های دو بعدی نیز کاربرد های فراونی دارند، در مثال قبلی اگر ما بخواهیم تعدا دانش آموزان تمامی رشته های دبیرستان را وارد کنیم به این صورت می نویسیم: unsigned Char dabirestan [3][4]; و تعریف می‌کنیم که این دبیرستان ۳ رشته به ترتیب ریاضی (۰)، تجربی (۱) و علوم انسانی (۲) دارد و هر رشته نیز ۴ گروه با تعداد دانش آموزان مختلف دارد. به مثال زیر توجه کنید:

```
unsigned Char dabirestan[3][4]{
{12,15,9,16},{10,11,8,13},{11,13,7,10}
};
```

حالا میخواهیم تعداد دانش آموزان گروه ۳ رشته تجربی را انتخاب کنیم و درون یک متغیر دیگر کپی کنیم: a=dabirestan[1][2];

نکته: اگر ندانیم که تعداد عناصر یک آریه چند تا می‌باشد داخل [] را خالی می گذاریم و خود کامپایلر این موضوع را تشخیص خواهد داد. البته این موضوع در مواردی مانند مثال بالا صحیح می‌باشد، یعنی وقتی آرایه را تعریف و اعضای آن را مقدار دهی اولیه می‌کنیم، تنها در این صورت کامپایلر تشخیص خواهد داد و تعداد سطر و ستون آرایه را مشخص می‌کند. در صورتی که ما نخواهیم مقدار دهی اولیه انجام دهیم حتما باید تعداد سطر و ستون را وارد کنیم.

رشته ها:

رشته ها آرایه ای از کاراکترها می‌باشند و حتما باید نوع آنها Char باشد. از رشته های در صورتی که در توابع زیادی نیاز باشد از یک رشته ثابت استفاده کنیم در ابتدای برنامه آن رشته را به صورت سراسری تعریف می‌کنیم تا در تمامی توابع قابل دسترس باشد:

```
Char a[]="picpars.com";
```

در اینجا خود کامپایلر تشخیص خواهد که طول رشته picpars.com چند کاراکتر می باشد.

۱ دستورات کنترلی

ضمیمه دستوارت کنترلی **(Proj-1)** | باز کردن پوشه مربوطه

حلقه های کنترلی for

حلقه یعنی چه؟ حلقه یعنی یک مسیر بسته ای که CPU بطور مداوم دستورات داخل آن را اجرا می کند و بعد از انجام دادن آخرین دستور حلقه مجددا به ابتدای حلقه برمی‌گردد و شرط را حلقه را چک می‌کند در صورتی که شرط برقرار بود ادامه می دهد...

حلقه for بالا شمار

```
for(i=0; i<=255; i++){
      PORTA=i;
      delay_ms(200);
}
```

(برای i=0 تا زمانی که i کوچکتر مساوی ۲۵۵ متغیر i را یک واحد افزایش بده) در مثال بالا ابتدا مقدار i را برابر صفر قرار می دهیم و سپس یک شرط برای اتمام حلقه تعیین می‌کنیم، و در آخر هم مقدار متغیر را یک واحد یک واحد افزایش می‌دهیم. تا زمانی که شرط حلقه برقرار باشد دستورات داخل آن اجرا می‌شود.

حلقه for پایین شمار

```
for(i=255; i>=0; i--){
      PORTA=i;
      delay_ms(200);
}
```

(برای i=255 تا زمانی که i بزرگتر مساوی ۰ است متغیر i را یک واحد کاهش بده) این مثال مانند مثال قبلی است با این تفاوت که حلقه از بالا به پایین شمارش را انجام می دهد.

نکته: می توان بجای i++ نوشت i+=2 یعنی دو واحد دو واحد اضافه کن!

نکته: برای حلقه های پایین شمار بهتر از متغیر های علامت دار یعنی signed int i استفاده شود! (برای تست شرط منفی)

نکته: متغیرهای signed محدوده اعداد شان نصف unsigned می باشد!

استفاده از دستورات break , continue در حلقه ها

دستور break حلقه در حال اجرا را بدون هیچ قید و شرطی به اتمام می رساند و باعث می شود CPU از حلقه خارج شده و ادامه دستورات بعد از حلقه را اجرا کند.

دستور continue برعکس break عمل می‌کند، به این صورت که اگر درون یک حلقه در حال اجر دستور continue اجرا شود CPU به ابتدای حلقه رفته و مجددا شرط حلقه را چک می‌کند، در صورتی که شرط برقرار باشد حلقه ادامه می‌یابد و در غیر این صورت از حلقه خارج می‌شود. به طور خلاصه هرگاه این دستور در هر کجای حلقه اجرا شود دستورات بعد از آن اجرا نخواهند شد و به ابتدای حلقه پرش خواهد کرد.

حلقه های کنترلی while

ساختار حلقه به این صورت می باشد که باید درون () مقدار true یا false باشد. True یعنی صحیح و شامل عدد ۱ یا برقرار بودن یک شرط می باشد که در این صورت CPU تمامی دستورات حلقه را تا زمانی که شرایط ذکر شده وجود داشته باشند اجرا خواهد کرد. false یعنی غلط و شامل عدد ۰ یا عدم برقرار بودن یک شرط می باشد که در این صورت دستورات داخل اجرا نخواهند شد و CPU از حلقه بیرون خواهد آمد. حلقه while دو حالت دارد:

حلقه while

```
While(1){
PORTA=~PORTA;
delay_ms(400);
}
```

در صورتی که عدد ۱ درون () بنویسیم، در اصل یک حلقه بی قید و شرط بینهایت ایجاد کرده ایم و تنها راه خروج از آن استفاده از دستور break در حلقه می باشد. در مثال بالا تا بینهایت پورت A هر ۴۰۰ میلی ثانیه معکوس می شود.

```
while(i<10){
PORTA=~PORTA;
delay_ms(400);
i++;
}
```

در مثال بالا از شرط i<10 برای تکرار حلقه استفاده کرده ایم، به این صورت که با هر بار اجرای دستورات یک واحد به i اضافه می شود و به ابتدای حلقه برمیگردد و مجددا شرط حلقه چک می شود و در صورت صحیح بودن ادامه داده می شود. در این مثال ۱۰ بار دستورات داخل حلقه اجرا می شود و سپس از آن خارج می شود.

```
while(1){
PORTA=~PORTA;
delay_ms(400);
i++;
if(i>10) break;
}
```

در مثال بالا یک حلقه بی نهایت داریم که تنها راه خروج از آن این است که مقدار i از ۱۰ بزرگتر شود تا شرط if برقرار گردد و دستور شکسته شدن حلقه اجرا شود. به عنوان مثال در اینجا می توان از شرط یک شدن یکی از پین های پورت ها استفاده نمود، به این صورت که یک کلید به پین مربوطه متصل است و وقتی کلید فشار داده شود دستور شکسته شدن اجرا شود.

حلقه do while

عملکرد این حلقه کاملا شبیه به حلقه while می باشد، با این تفاوت که دستورات حلقه حداقل یک بار اجرا خواهند شد و در پایان شرط حلقه چک می شود. در صورتی که شرط صحیح باشد حلقه ادامه داده می شود و در غیر اینصورت حلقه ادامه داده نمی شود.

```
do{
PORTA=~PORTA;
delay_ms(400);
i++;
}while(i<10);
```

دستور کنترلی switch

این دستور یک متغیر را برسی می‌کند و به ازای مقادیر تعریف شده ثابت دستوراتی را اجرا خواهد کرد. در اصل یک مقایسه کننده می باشد که مقدار متغیر مربوطه را با تمامی موارد موجود (case) مقایسه می‌کند و در صورت برابری دستورات case مربوطه اجرا خواهند شد. برای روشن تر شدن موضوع به مثال زیر توجه نمائید.

```
i=1;
switch(i){
case 1:
PORTA=10;
break;

case 2:
PORTA=20;
break;

default:
PORTA=0b10101010;
}
```

در این مثال ابتدا متغیر i به عنوان مرجع مقایسه انتخاب شده و سپس با case های مربوطه مورد مقایسه قرار میگیرد. اگر مقدار متغیر i=1 بود عدد ۱۰ روی پورت A قرار می‌گیرد و با استفاده از دستور break حلقه مقایسه شکسته خواهد شد و سایر case ها مورد مقایسه قرار نخواهند گرفت. همچنین ما می توانیم یک پیش فرض هم تعریف کنیم تا در صورتی که هیچ یک از case ها مورد مقایسه قرار نگرفتند دستورات پیش فرض اجرا شوند. در اینجا اگر مقدار i نه ۱ بود و نه ۲ بود قسمت default اجرا خواهد شد و پین های پورت A به صورت یک در میان روشن خواهند شد. شما می‌توانید مقدار متغیر i که در ابتدا با عدد ۱ مقدار دهی شده است را تغیر دهید و برنامه را مجددا کامپایل کرده و نتیجه آن را در شبیه سازی مشاهد نمائید.

۲ دستور شرطی IF

ضمیمه دستورات شرطی و define (Proj-2) | باز کردن پوشه مربوطه

دستور شرطی if داری کاربرهای فراونی می باشد، در اکثر برنامه ها از این دستور برای کنترل اجرای برنامه استفاده می شود. ساده ترین حالت آن به صورت زیر است:

```
If(PINB.0==1) a=50;
If(a==30) PORTB.2=1;
```

همانطوری که مشاهد می‌کنید در شرط اولی گفتیم؛ اگر پین صفر پورت B یک شده مقدار متغیر a را برابر ۵۰ قرار بده. در دستور دومی گفتیم که اگر مقدار a برابر ۳۰ می باشد بیت 2 پورت B را یک کن.

```
If(a==30) PORTB.2=1;
  Else PORTB.3=1;
```

در مثال بالا گفتیم اگر مقدار a برابر ۳۰ می باشد بیت 2 پورت B را یک کن در غیر اینصورت بیت 3 پورت B را یک کن. (در غیر اینصورت=else)

حال به نمونه کامل تر از این شرط توجه نمائید:

```
If(a==30) PORTB.2=1;
 Else if(a==31) PORTB.3=1;
 Else if(a==31) PORTB.4=1;
 Else if(a==32) PORTB.5=1;
else PORTB.6=1;
```

در دستورات بالا به ترتیب شرط ها برسی می‌شود و هرکدام از آنها که برقرار بود دستور مربوطه آن اجرا خواهد شد. هیچ گونه محدودیتی برای تعداد شرط های Else if وجود ندارد و همچنین نوشتن else پایانی کاملا اختیاری می باشد و شما می توانید در صورت عدم نیاز از آن استفاده نکنید.

نکته: در صورت اجرا شدن یکی از شرط ها، مابقی شرط های بعد از آن دیگر چک نخواهد شد و CPU به دستورات بعد از else پایانی ارجاع داده می‌شود.

همانطوری که قبلا نیز اشاره کرده ایم، در صورتی که دستورات شما بیش از یک دستور باشد باید آنها در مجموعه {} قرار داد به نمونه زیر توجه نمائید:

```
If(a==30){
      PORTB.2=1;
      X=10;
}else if(a==31){
      PORTB.3=1;
      X=20;
}else if(a==32){
      PORTB.4=1;
      X=30;
} else {
      PORTB.5=1;
      X=40;
}
```

دستور کاربردی #define

Define یعنی تعریف کردن، به کمک این دستور شما می توانید مقادیر ثابتی را تعریف کنید. همچنین می‌توان از آن برای تغیر نام دادن ثابت های کامپایلر استفاده کرد.

```
#define val 170
```

در مثال بالا در، در صورتی که عدد ۱۷۰ در برنامه نیاز باشد می‌توان به راحتی کلمه val را بکار برد. از مزایای استفاده از این دستور می توان به این مورد **مهم** اشاره نمود؛ فرض کنید که شما یک برنامه طولانی نوشته اید و در خیلی از حلقه ها و توابع می خواهید یک متغیر را با عدد ۱۷۰ مورد مقایسه قرار دهید، اگر در تمامی موارد مستقیما بنویسید ۱۷۰ و در آینده بنا به تغیراتی لازم باشد که عدد ۱۷۰ به ۱۵۰ تغییر کند شما باید در تمامی برنامه بگردید و این عدد را تغیر دهید. اما در صورتی که به کمک دستور define بجای عدد ۱۷۰ کلمه val را در تمام برنامه مورد استفاده قرار داده باشد، به راحتی می توانید تغییرات خود را در آینده اعمال کنید. فقط کافی است که در همان ابتدای تعریف عدد ۱۷۰ را به ۱۵۰ تغیر دهید و مجددا برنامه را کامپایل کنید.

```
#define data_port PORTA
```

همانطور که مشاهد می کنید کلمه کلیدی PORTA به data_port که ما تعریف کرده ایم تغیر نام پیدا کرده است، بنابراین در طول برنامه دیگر لازم نیست از کلمه PORTA برای مقدار دهی پورت استفاده کنید و بلکه از data_port برای این کار استفاده می‌کنیم. مزایای این تغیر نام نیز همانند موارد بالا است که ذکر شد.

یک برنامه نویس همواره باید دید آینده نگری برای برنامه خود داشته باشد تا بتواند تغیرات احتمالی را به راحتی اعمال نماید.

در مثالی که ضمیه این قسمت شده است دستور define و دستور if در حالات مختلف بکار برده شده است، برنامه به این صورت عمل می‌کند که اگر هر کلیدی فشار داده شود بیت متناظر آن در پورت دیگر معکوس خواهد شد، به دلیل ایجاد تاخیر ۲۰۰ میلی ثانیه در صورتی که هر کلید را به صورت مدارم نگه داریم بیت مربوطه به صورت چشمک زن عمل خواهد کرد.

نکته: برای نوشتن روی پورت باید از دستور PORTX و برای خواندن از پورت از دستور PINX استفاده کرده. که در این مثال برای هر دو حالت از دستور define نیز استفاده شده.

۳ توابع در زبان C

ضمیمه ایجاد توابع **(Proj-3)** | باز کردن پوشه مربوطه

هر برنامه به زبان C مجموعه ای از یک یا چندین تابع است که یکی از آنها تابع اصلی بنام main() است، مابقی توابع از داخل این تابع اصلی فراخوانی می شوند. تابع در واقع یک زیر برنامه است که توسط برنامه نویس برای انجام عملیات خاص نوشته می‌شود، هدف اصلی ما از نوشتن برنامه ها به صورت توابع کاهش حجم برنامه نویسی و فراخوانی توابع درون یکدیگر می باشد. به طور کلی توابع به دو دسته تقسیم می شوند:

۱. توابع از پیش تعریف شده که به آنها توابع کتابخانه ای گفته می شود مانند توابع sin(x) و cos(x) و delay_ms(x) و...
۲. توابعی که توسط برنامه نویسی تعریف می‌شوند.

نوع اول توابع نوشته و آماده خود کامپایلر می باشد و در اینجا ما نوع دوم را توضیح می دهیم. در کل این گونه توابع دو حالت دارند یکی مقدار برگشتی دارند و دیگری مقدار برگشتی ندارند. **مقدار برگشتی یعنی چی؟**

هر تابع یک مقدار ورودی دارد و یک مقدار خروجی، یک تابع با قابلیت مقدار برگشتی یعنی اینکه ما یک مقداری به ورودی تابع می دهیم و در این تابع یک سری عملیات بر روی این مقدار ورودی انجام می شود و در نهایت جواب این عملیات برگشت داده می شود. و در نوع دوم تابع می تواند هم مقدار ورودی داشته باشد و هم نداشته باشد، به عنوان مثال اگر یک مقدار ورودی داشته باشد با این مقدار ورودی یک سری عملیات را انجام می دهد و در نهایت CPU برمیگردد و هیچ مقداری برگشت داده نمی‌شود. در زیر با مثال های زیادی این مطالب را روشن تر می کنیم:

محل نوشتن توابع کجاست؟ شما به دو طریق می توانید تابع ها را در محل های صحیح بنویسید:

۱. توابع را به ترتیب قبل از تابع اصلی main() بنویسید.
۲. توابع را به ترتیب بعد از تابع اصلی main() بنویسید که در این روش باید قبل از تابع main() فقط نام توابع را معرفی کنیم.

- ✓ اگر در پرانتز عبارت (void) نوشته شود یعنی تابع مقدار برگشتی ندارد.
- ✓ جهت فراخوانی توابع تودرتو باید ترتیب نوشتن توابع رعایت گردد.

```
void main(void){
دستورات
}
```

به این تابع، تابع اصلی ()main گفته می شود.

۱) نوشتن توابع قبل از main:

```
void keypad(void){
 دستورات زیر برنامه
 }

 void main(void){
   //www.picpars.com دستورات
keypad();
 }
```

در اینجا ما یک تابع با نام keypad تعریف کردیم و اگر توجه کنید قبل از تابع main تعریف شده و حالا آن را در تابع اصلی فراخوانی می‌کنیم.

۲) نوشتن تابع بعد از main:

```
void keypad();

 void main(void){
    //دستورات
    keypad();
 }
  void keypad (void){
 //دستورات زیر برنامه
 }
```

در این روش ابتدا نام تابع keypad را تعریف کردیم و در آخر بعد از main تابع keypad را می نویسیم.

تابع بدون مقدار برگشتی

```
void blink_led(){
     PORTA=~PORTA;
     delay_ms(500);
}
```

این تابع هیچ گونه مقدار برگشتی ندارد، همچنین هیچ مقدار ورودی هم ندارد و با تاخیر ۵۰۰ میلی ثانیه به صورت چشمک زن کار می‌کند. در تابع main باید به صورت ()blink_led فراخوانی شود.

```
void blink_led2(unsigned int d){
     PORTA=~PORTA;
     delay_ms(d);
}
```

این تابع هم هیچ گونه مقدار برگشتی ندارد اما یک مقدار ورودی در محدوده int (۰ تا ۶۵۵۳۵) دارد. به این صورت که در هنگام فراخوانی تابع باید یک عدد صحیح در محدوده تعریف شده درون پرانتز وارد کنیم، blink_led2(1000) . در اینجا با فراخوانی این تابع پورت A به مدت ۱۰۰۰ میلی ثانیه خاموش و روشن می شود. پس طبق تعریف از ۱ تا ۶۵۵۳۵ میلی ثانیه تاخیر می توانیم ایجاد کنیم. هدف این نوع تابع متغیر بودن زمان تاخیر می باشد.

تابع با مقدار برگشتی

```
unsigned int blink_led3(unsigned int d){
     d=d*1000;
     return d;
}
```

این تابع یک مقدار ورودی را دریافت و سپس آن را در ۱۰۰۰ ضرب کرده و حاصل را برگشت می دهد. در ادامه حاصل را درون تابع تاخیر قرار می دهیم تا بر حسب ثانیه تاخیر ایجاد شود. عملکرد این تابع مانند مثال قبل است با این تفاوت که بجای وارد کردن عدد ۱۰۰۰ میلی ثانیه ما بر حسب ثانیه عدد صحیح وارد می‌کنیم، (به عنوان مثال عدد ۱)

۴ کار با LCD کاراکتری (متنی)

ضمیمه کار با **LCD** کاراکتری **(Proj-4)**	باز کردن پوشه مربوطه
ضمیمه دیتاشیت LCD های متنی	باز کردن دیتاشیت

در این قسمت ما به آموزش LCD های کاراکتری خواهیم پرداخت و توضیح می دهیم که چگونه آن را به میکرو کنترلر متصل کنیم و چگونه آن را در زبان C معرفی و با استفاده از دستورات C به صورت نرم افزاری با آن ارتباط برقرار کنیم. این مقاله بر اساس کامپایلر CodeVisionAVR V2.05.3 به بالا می باشد، بنابراین دستورات و پیکره بندی LCD بر اساس این نسخه می باشد.

ابتدا توابع موجود در کتابخانه #include <alcd.h> را فراخوانی می کنیم. همچنین توابع استاندارد ورودی و خروجی stdio.h را نیز فراخوانی می‌کنیم.

پایه های LCD

پایه‌های LCD		
پایه	نام	عملکرد
1	VSS	زمین
2	VCC	5V+
3	VEE	کنترل درخشندگی (می توانید با یک مقاومت ۱ کیلو آن را زمین کنید)
4	RS	اگر این پایه ۰ باشد اطلاعات روی DB0-DB7 به عنوان فرمان و اگر ۱ باشد به عنوان کاراکتر پذیرفته می شود
5	R/W	اگر این پایه ۰ باشد LCD برای نوشتن آماده می شود و اگر ۱ باشد برای خواندن آماده می شود
6	E	فعال سازی LCD که با یک لبه پایین رونده می باشد
7	DB0	DB0 تا DB7 خطوط دیتا (مد ۸ بیتی)
8	DB1	
9	DB2	
10	DB3	
11	DB4	DB4 تا DB7 مدل ۴ بیتی
12	DB5	
13	DB6	
14	DB7	
15	A	5V+ از پایه ۱۵ و ۱۶ برای روشن کردن LED پس زمینه استفاده می شود
16	K	زمین

جدول ۴-۱ پایه های LCD های متنی

کد های فرمان LCD

کد HEX	عملکرد فرمان
0X01	پاک کردن صفحه نمایش
0X02	بازگشت مکان نما به سطر و ستون صفر و صفر (HOME)
0X04	انتقال مکان نما به چپ
0X05	جابجایی صفحه نمایش به راست
0X06	انتقال مکان نما به راست
0X07	جابجایی صفحه نمایش به چپ
0X08	صفحه نمایش و مکان نما خاموش
0X0A	صفحه نمایش خاموش و مکان نما روشن
0X0C	صفحه نمایش روشن و مکان نما خاموش
0X0E	صفحه نمایش روشن و مکان نما روشن
0X0F	صفحه نمایش روشن و مکان نما در حالت چشمک زن
0X10	جابجایی مکان نما به چپ
0X14	جابجایی مکان نما به راست
0X18	حرکت کل صفحه نمایش به چپ
0X38	سازمان دهی ۸ بیتی و ماتریس ۵×۷
0X28	سازمان دهی ۴ بیتی و ماتریس ۵×۷
0X10	حرکت کل صفحه نمایش به راست
0XC0	مکان نما به آغاز خط دوم انتقال می یابد

جدول ۴-۲ کد های فرمان LCD

فرمت کاراکتر های ارسالی

از آنجا که برای ارسال اطلاعات به LCD از توابع کتابخانه استاندارد ورودی و خروجی (stdio.h) استفاده می کنیم، بنابراین لازم که با نحوی فرمت بندی رشته و متغیر ها آشنا شویم. جدول زیر علاوه بر LCD در ارتباط سریال نیز کاربرد دارد.

کاراکتر	نوع اطلاعات ارسالی
c%	یک تک کاراکتر
d%	عدد صحیح علامت دار در مبنای ۱۰
i%	عدد صحیح علامت دار در مبنای ۱۰
e%	نمایش عدد ممیز شناور به صورت علمی
E%	نمایش عدد ممیز شناور به صورت علمی

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

جدول ۴-۲ فرمت کاراکتر های ارسالی

توابع LCD متنی

lcd_init(16); این تابع LCD را پیکره بندی و آماده دریافت اطلاعات می‌کند. عدد داخل پرانتز نمایانگر تعداد ستون های LCD می باشد. مثلا برای LCD های ۲×۲۰ باید عدد ۲۰ را وارد کرد.

lcd_clear(); این تابع کل صفحه نمایش را پاک می‌کند و نویسه را در ناحیه سطر صفر و ستون صفر قرار می دهد. *نحوی استفاده:*

```
lcd_clear();
```

lcd_puts(); این تابع رشته موجود در حافظه SRAM میکرو را بر روی نمایشگر چاپ می کند. *نحوی استفاده:* `lcd_puts(buf);`

lcd_putsf(); این تابع رشته موجود در حافظه FLASH میکرو را بر روی نمایشگر چاپ می کند. *نحوی استفاده:*

```
lcd_putsf("LCD test project");
```

lcd_putsfe(); این تابع رشته موجود در حافظه EEPROM میکرو را بر روی نمایشگر چاپ می کند. *نحوی استفاده:*

```
lcd_putsfe(ebuf);
```

lcd_putchar(); این تابع یک تک کاراکتر موجود در SRAM یا FLASH را بر روی نمایشگر چاپ می کند. *نحوی استفاده:*

```
lcd_putchar(c); یا lcd_putchar('H');
```

lcd_gotoxy(); این تابع نویسه را در مختصات x , y قرار می دهد، به عنوان مثال در سطر اول و ستون ۷ به این صورت می باشد.

نحوی استفاده: `lcd_gotoxy(0,6);` **نکته سطر و ستون از عدد صفر شروع می‌شود!**

lcd_write_data_(); این تابع کد های فرمان را مطابق جدول ذکر شده در بالا برای LCD ارسال می نماید. *نحوی استفاده:*

```
lcd_write_data_(0x0f); نویسه نمایشگر به صورت چشمک زن خواهد شد
```

lcd_write_byte(addr,data); *از این تابع برای نوشتن یک کاراکتر خاص (مثلا فارسی) یا کاراکتر های تعریف شده در دیتاشیت LCD استفاده می شود.*

lcd_read_byte(); *از این تابع هم برای خواندن اطلاعات موجود در RAM نمایشگر LCD استفاده می شود. از این دستور به ندرت استفاده می شود مگر در موارد خاص*

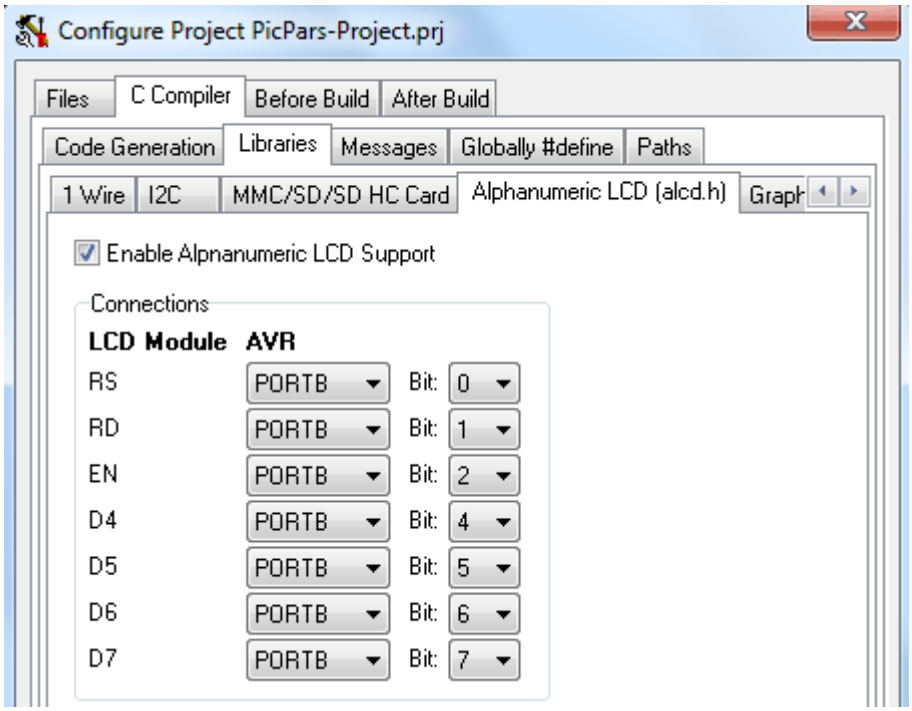
نکاتی در مورد lcd_puts(buffer);: همانطور که گفتیم این تابع یک متغیر رشته ای از نوع char بر روی نمایشگر چاپ می‌کند. در صورتی که از دستور sprintf برای آماده سازی رشته ها استفاده می‌کنید باید یک متغیر char به صورت آرایه‌ای **buffer[32]** در حافظه SRAM میکرو ایجاد نمود، تعداد آرایه بستگی به تعداد کل سطر ها و ستون ها دارد. به عنوان مثال برای یک LCD 2*16 باید از آرایه ۳۲ تایی استفاده کرد.

```
lcd_clear();
sprintf(buffer,"Temp=%.2f\xdfC\nwww.picpars.com",temp);
lcd_puts(buffer);
delay_ms(1500);
```

در مثال بالا می خواهیم دما محیط را که در متغیر temp قرار دارد با دقت دو رقم اعشار نمایش دهیم، برای این کار باید %.2f نوشت و اگر بخواهید اعشار به صورت کامل نوشته شود باید %f نوشت. کد \xdf نماد درجه ° را ایجاد می‌کند و C هم سانتیگراد می باشد. \n به معنای شکستن سطر و رفتن به ستون اول سطر بعدی می باشد. به طور کلی در تابع فوق می توان از تعداد زیادی متغیر استفاده نمود و تنها باید توسط جدول فرمت کاراکتر های ارسالی متغیرها قالب بندی و آماده ارسال شوند.

نکات مهم در تنظیمات کامپایلر

برای تعریف کردن پین های اتصال LCD به میکرو ابتدا به منوی Project ➔ Configure ➔ C Compiler ➔ Libraries ➔ Alphanumeric LCD (alcd.h) رفته و تیک Enable Alphanumeric LCD Support را بزنید حال به دلخواه خودتان پین های LCD را برای پین های میکرو تعریف کنید. (پورت دلخواه و پین های دلخواه)



برای تغیر دادن ویژگی های دستور Sprintf ابتدا به منوی Project ➔ Configure ➔ C Compiler ➔ Code Generation رفته سپس در قسمت (s)printf Features اندازه و حجم این تابع را مشخص کنید. به عنوان مثال برای اینکه بخواهید محتوای یک متغیر float را با استفاده از تابع sprintf نمایش دهید (استفاده از %f) حتما باید این گزینه را بر روی float, width, precision تنظیم کنید، در غیر اینصورت محتوای متغیر مربوطه نمایش داده نمی شود. **نکته** انتخاب این گزنیه حجم کد هگز تولید شده را افزایش می دهد.

۵ اتصال کیبورد ۴*۴ به میکروکنترلر

اتصال کیبورد ۴×۴ **(Proj-5)** | باز کردن پوشه مربوطه

به طور خلاصه، کلید های یک کیبورد به صورت ماتریسی در کنار هم قرار می‌گیرد تا مزیت مهمی به نام کاهش تعداد پین های خروجی را داشته باشند. هر کلید داری دو پایه است، اگر پایه سمت چپ همه کلید های یک ستون را به هم وصل کنیم و پایه سمت راست همه کلیدهای یک سطر را هم به هم وصل کنیم، ما یک سطر و ستون ایجاد کرده ایم حالا اگر ۱۶ کلید را به صورت سطر و ستون های ۴×۴ به روش فوق به هم متصل کنیم ما یک ماتریس ۴×۴ داریم که داری ۴ پایه ستون و ۴ پایه سطر می باشد. همانطور که ملاحضه می‌کنید با این روش می‌توانیم ۱۶ کلید را تنها با ۸ پین کنترل کنیم.

چگونه کلید ها را اسکن کنیم؟ برای اینکار باید نیمی از یک پورت را خروجی و نیم دیگر را ورودی تعریف کنیم که در این مثال ما نیمه اول یعنی PORTC.0 تا PORTC.3 را خروجی (سطرها R) و نیمه دوم یعنی PORTC.0 تا PORTC.3 را ورودی (ستون ها C) تعریف کرده ایم. (مقاومت های Pullup داخلی نیز فعال می‌شوند) توسط میکرو نیمه دوم که ورودی های ستون ها هستند را فقط برای صفر شدن ردیابی می‌کنیم و در هر بار یکی از پایه های نیمه اول را صفر کرده و مابقی را یک (این عمل باید به ترتیب از پین ۰ شروع و به پین ۳ ختم شود). اصول کار به این صورت می‌باشد که وقتی یکی از سطر ها توسط میکرو صفر می شود و کاربر در همان لحظه یکی از کلید های متناظر با این سطر را فشار دهد، فورا توسط دستورات شرطی ستون مربوط به کلید شناسایی می شود و متغیر ستون (c) برابر با یکی از اعداد ۰ تا ۳ خواهد شد. از آنجایی که مقدار c از ۲۵۵ تغیر کرده شرط انتهایی برقرار خواهد شد و با توجه به مقدار متغیر (r) که وضعیت سطر را نشان می دهد r در عدد ۴ ضرب شده و با مقدار متغیر c جمع می شود، بنابراین حاصل این عملیاتی عددی بین ۰ تا ۱۵ می باشد که توسط این عدد از آرایه arrkey مقدار واقعی کلید فشرده شده را استخراج می کنیم. در آخر هم برای تمامی ستون ها یک حلقه بینهایت شرطی تعریف کرده ایم تا زمانی که کاربر دست خود را روی کلید مربوطه بر ندارد میکرو در این حلقه بماند. تاخیر ۵۰ میلی ثانیه هم برای از بین بردن لرزش دست هنگام رها کردن کلید می باشد و در آخر هم مقدار کلید بازگشت داده می شود.

```
unsigned char scan[4]={0XFE,0XFD,0XFB,0XF7};
char key;
char arrkey[16]={
'7','8','9','/',
'4','5','6','*',
'1','2','3','-',
'C','0','=','+'};
```

آرایه scan دارای ۴ مقدار می‌باشد که هر کدام از مقادیر تنها یکی از پین های مربوطه را صفر خواهد کرد. (PORTC.0 تا PORTC.3)

آرایه arrkey نیز مقدار واقعی کلید را بازگشت می دهد. (همانطور که مشاهد می کنید به کمک این روش اسکن شما به راحتی می توانید متناسبت با چیدمان صفحه کلید تان این آرایه را تغییر دهید.)

```
#define c1 PINC.4
#define c2 PINC.5
#define c3 PINC.6
#define c4 PINC.7
#define keypad_port PORTC
```

به کمک دستوارت فوق برای آسان شدن برنامه نویسی، پایه های ستون های کیبورد را تعریف می‌کنیم. همچنین کل پورت را هم تغیر نام می دهیم. (اختیاری)

```
char keypad(){
  unsigned char r,c,k;
  DDRC=0X0F;
  keypad_port=0XFF;

   while(1){
       for (r=0; r<4; r++){
        c=255;
        keypad_port=scan[r];
        delay_us(10);
        if(c1==0) c=0;
        if(c2==0) c=1;
        if(c3==0) c=2;
        if(c4==0) c=3;

         if (c!=255){
          k=arrkey[(r*4)+c];
          while(c1==0);
          while(c2==0);
          while(c3==0);
          while(c4==0);
          delay_ms(50);
          return k;
         }
       }
   }
}
```

همانطور که مشاهد می کنید تابع از نوع برگشتی می باشد و در برنامه اصلی باید به صورت ;key=keypad() فراخوانی شود. (متغیر key باید از نوع char باشد)

بعد از دریافت کلید فشرده شده توسط دستورات زیر مقدار کلید را بر روی نمایشگر نمایش می دهیم:

```
while(1){
          key=keypad();
          lcd_clear();
          sprintf(buffer,"key=%c",key);
          lcd_puts(buffer);
        }
```

کمی بیشتر در مورد متغیر c=255: محتوای این متغیر را به صورت پیش فرض با عدد ۲۵۵ پر می‌کنیم، حال اگر کلیدی فشرده شود محتوای آن تغیر خواهد کرد و میکرو به کمک این تغیر متوجه خواهد شد که کلیدی فشار داده شده و باید ستون آن را پیدا کند و در نهایت هم مقدار واقعی را از آرایه استخراج کند. **نکته:** عدد ۲۵۵ کاملا اختیاری می‌باشد شما می توانید از هر عدد دیگری بجزء اعداد ۰ تا ۳ استفاده کنید.

۶ سون سگمنت ها

سون سگمنت ها آرایه های هفت قسمتی هستند که دارای دو نوع کاتد مشترک و آند مشترک می باشند. سون سگمنت ها نوعی LED می‌باشند و با چیدمان خاص خود می توان اعداد ۰ تا ۹ و برخی حروف را نیز نمایش داد. قسمت ها با نام های a,b,c,d,e,f,g نام گذاری شده اند و با توجه به نوع آنها کدهای متفاوتی برای روشن کردن LED باید ایجاد کرد تا یک رقم یا حروف خاص را نمایش دهند، بنابراین در همین راستا به نرم افزار PicPars Tools قسمتی اضافه کردیم تا به راحتی بتوان این کد ها را بدست آورد.

ابتدا نوع سون سگمنت (کاتد یا آند) را تعیین کرده پس با کلیک کردن بر روی هر یک از LED ها (روشن یا خاموش) کد سون سگمنت در دو مبنای هگز و دسیمال برای شما تولید می شوند. همانطور که میدانیم یک پورت میکرو داری ۸ پین می باشد و یک سون سگمنت دارای ۷ پین کنترلی و یک پین مشترک می باشد که در برخی از آنها نیز پین دیگری برای نقطه "." تعبیه شده است. حال اگر از پین نقطه صرفه نظر کنیم یا آن را لازم نداشته باشیم تنها ۷ پین آن به یک پورت میکرو وصل می شود و بین آخر یعنی X.7 باقی می ماند، بنابراین اگر بخواهیم از این بیت برای مصارف دیگری استفاده کنیم با توجه به وضعیت بیت مربوطه می توان از گزینه PORTX.7 در نرم افزار استفاده کرده و وضعیت آن را در کدهای تولید شده تعین نمود. یک بودن یا صفر بودن این بیت تاثیری در نمایش سون سگمنت نخواهد داشت و تنها وضعیت بیت هفتم (پایه هشتم پورت) را مشخص می کند.

سون سگمنت آند مشترک

شمارنده تک رقمی با اتصال مستقیم

اتصال مستقیم سون سگمنت آند مشترک به میکرو **(Proj-6)** | باز کردن پوشه مربوطه

در این روش با استفاده از نرم افزار PicPars Tools ابتدا کدهای مربوط به سون سگمنت آند مشترک را ایجاد کرده و درون یک متغیر آرایه ای به نام anode_seg قرار می دهیم. حال با استفاده از دستور for یک شمارنده ۰ تا ۹ ایجاد می کنیم و در هر بار شمارش محتوای آرایه را روی پورت قرار می دهیم. PORTC = anode_seg[i]

به همین سادگی شما قادر خواهید بود یک شمارنده تک رقمی ایجاد کنید. البته شمارنده معکوس شمار هم با یک تغیر جزئی در حلقه for قابل انجام می باشد و تنها نکته مهم این است که متغیر شمارنده از نوع علامت دار باشد. signed char i

```
for(i=9; i>=0; i--){
     PORTC = anode_seg[i];
     delay_ms(500);
   }
```

شمارنده تک رقمی با آیسی 74LS247

اتصال سون سگمنت آند مشترک به میکرو با آیسی **74LS247** **(Proj-7)** | باز کردن پوشه مربوطه

در این روش همانطور که میدانیم باید ارقام را به فرمت BCD برای آیسی مربوطه ارسال کنیم، و تنها ۴ پین از پورت میکرو برای این روش مورد استفاده قرار میگیرد، بنابراین می‌توان به یک پورت میکرو ۲ عدد سون سگمنت را اتصال داد. اما برای برنامه نویسی باید ابتدا کتابخانه bcd.h را فراخوانی کنیم تا عدد شمارنده را به فرمت BCD تبدیل کرده و بر روی پورت قرار دهیم.

سون سگمنت کاتد مشترک

شمارنده تک رقمی با اتصال مستقیم

اتصال مستقیم سون سگمنت کاتد مشترک به میکرو **(Proj-8)** | باز کردن پوشه مربوطه

عملکرد این برنامه نیز همانند قسمت آند مشترک می باشد، با این تفاوت که کدهای سون سگمنت برای نوع کاتد مشترک ایجاد شده.

شمارنده تک رقمی با آیسی 74LS248

اتصال سون سگمنت کاتد مشترک به میکرو با آیسی **74LS248** **(Proj-9)** | باز کردن پوشه مربوطه

عملکرد این برنامه نیز همانند قسمت آند مشترک می باشد، با این تفاوت که از آیسی 74LS248 استفاده شده است.

سون سگمنت چهار رقمی (کنترلی)

شمارنده چهار رقمی با سون سگمنت ۴رقمی **(Proj-10)** | باز کردن پوشه مربوطه

در این نوع سون سگمنت ها باید با توجه به زمان تریگر آنها به صورت مداوم اطلاعات هر رقم را ریفرش نمود. در مثال زیر یک سون سگمنت ۴ رقمی کاتد مشترک داریم که پین های A-G مربوط به ارقام و پین DP مربوط به ممیز ها و پین های ۱-۴ هم مربوط به هر یک از ارقام می باشد. اصول کار این نوع سون سگمنت ها به صورت زیر می باشد:

برای اینکه هر چهار رقم همزمان نمایش داده شود باید به صورت منظم و با تاخیر مناسب عمل ریفرش کردن آن را انجام دهیم. می توانید با مراجعه کردن به خصوصیات آن در پروتیوس مینیمم زمان تریگر را مشاهد کنید که در اینجا ۱میلی ثانیه است و در برنامه ما ۴ میلی ثانیه قرار داده ایم. ابتدا باید عدد پین ۱ را صفر و مابقی را یک کنیم، سپس عدد یک را روی پورت قرار دهیم. در مرحله بعد باید پین ۲ را صفر و مابقی را یک کنیم، سپس عدد ۳ را روی پورت قرار داد و این کار را تا رقم آخر باید انجام داد. (به طور کلی با صفر کردن هر پین سگمنت مربوطه برای تغیر دادن انتخاب می شود). در این مثال وضعیت کنترل پین ها را توسط آرایه control تعین کرده ایم و متناسب با عدد شمارنده مقدار آن را از آرایه cathode_seg انتخاب میکنیم و روی پورت قرار می دهیم.

تابع display() محتوای شمارنده c (چهار رقمی) را با استفاده از دستورات تقسیم و باقیمانده به اعداد تک رقمی تبدیل می‌کند و در آرایه data قرار می دهد. حال توسط یک حلقه for ارقام موجود در data را با فاصله زمانی ۴ میلی ثانیه روی پورت قرار می دهیم همچنین به کمک متغیر i حلقه for محتوای آرایه control را روی پورت قرار می دهیم. و در آخر هم باید هر ۴ پین را یک کنیم تا ثابت شود و ارقام تغیر نکنند.

```
void display(){
 unsigned int z;
 unsigned int i,c1,c2,c3,c4;

  c4 = c%10; data[3]=cathode_seg[c4];
  z=c/10; c3 = z%10;  data[2]=cathode_seg[c3];
  z=c/100; c2 = z%10;  data[1]=cathode_seg[c2];
  z=c/1000; c1 = z%10;  data[0]=cathode_seg[c1];

  for (i=0; i<4; i++){
     PORTB = control[i];
     PORTA = data[i];
     delay_ms(4);
  }
  PORTB=0x0f;
}
```

تاخیر بین شمارش ها:

برای ایجاد تاخیر بین هر شمارش ما نمی توانیم مستقیما از دستورات delay استفاده کنیم چون اگر به صورت پیوسته سون سگمنت را ریفرش نکنیم اراقام به ازای تاخیر ایجاد شده چشمک زن خواهد شد و به طور کلی نمی توان رقم مذکر را دید. برای رفع این مشکل ما یک تابع تاخیر جدیدی به نام delay_7segmeny ایجاد می‌کنیم که در آن تاخیر ورودی را به تاخیر های کوچکتر تبدیل می‌کنیم و در بی این تاخیر های کوچک عمل ریفرش را هم انجام می دهیم.

```
delay_7segmeny(unsigned int d){
  unsigned char i;
  d/=16;

  for(i=0; i<=d; i++){
      display();
  }
}
```

همانطور که مشاهد می کنیم تاخیر ورودی بر عدد ۱۶ تقسیم می شود و به تعدا این نتیجه تقسیم تابع display فراخوانی می‌شود. عدد *۱۶ از کجا آمد؟* همانطور که در تابع display مشخص است، چهار بار عملیات کنترلی اجرا و در هر مرحله ۴ میلی ثانیه تاخیر ایجاد می‌شود. بنابراین ۴×۴=۱۶ خواهد شد، یعنی در هر بار اجرای تابع display ۱۶ میلی ثانیه تاخیر ایجاد می‌شود، با تقسیم d بر ۱۶ شرط حلقه برای اجرا تابع display تعین می‌شود. به همین سادگی شما قادر خواهید بود هم سون سگمنت را ریفرش کنید و هم تاخیر مورد نیاز برای شمارش را ایجاد کنید.

۷ تایمرها

مقدمه

تایمر کانتر یکی از بخش های مهم میکروکنترلرها می باشد. در بیشتر مواقع لازم که تعدادی وقایع خارجی (با سرعت بالا) شمارش شود و یا گاهی لازم است که در یک زمان خاص و دقیق، کاری صورت گیرد. تنها توسط تایمر کانتر ها می توان این کارهای دقیق و با سرعت بالا را انجاد داد. میکروکنترلرهای AVR حداکثر دارای شش عدد تایمر کانتر هشت بیتی و شانزده بیتی هستند. برخی از آنها دارای عملکرد ساده و برخی دیگر دارای امکانات بیشتر نظیر تولید موج PWM ، حالت مقایسه CTC ، حالت تسخیر، عملکرد غیر همزمان و ... می باشند. در ادامه ما به نحوی برنامه نویسی و پیکره بندی تایمر صفر در حالت عادی همراه با مثال در محیط برنامه نویسی CodeVision می پردازیم.

تایمر کانتر صفر

تایمر کانتر صفر در AVR ها را می توان به سه مدل زیر دسته بندی کرد:

- ساده هشت بیتی
- پیشرفته هشت بیتی
- پیشرفته شانزده بیتی

ساده هشت بیتی فقط در سری At90S , ATTiny به کار رفته.
پیشرفته هشت بیتی در سری ATmega ها به غیر از ATmega8 و ATmega163 به کار رفته است. (این دو مدل میکرو از مد ساده هشت بیتی استفاده می کنند)
پیشرفته شانزده بیتی این مدل فقط در AVR های سری ATTiny2313 , ATTiny13 به کار گرفته است.

ما از مد پیشرفته هشت بیتی استفاده می کنیم و بیشتر روی codewizard برنامه CodeVision تاکید می کنم و زیاد وارد بحث رجیسترهای تایمر کانتر نمی شویم (البته تا حدوددی خواهیم گفت). شما کافی است که به اصل قضایای آنها آگاهی داشته باشید، بنابراین الازمی نیسیت که این مراحل تکراری را هر بار به صورت دستی محاسبه کنید، کافی است به کمک codewizard با تنظیم کردن وضعیت تایمر کانتر ها کد های مربوط به رجیستر ها را بدست آورید.

تایمر کانتر صفر در حالت هشت بیتی پیشرفته:

- تایمر کانتر در حالت عادی
- تایمر کانتر در حالت مقایسه CTC
- تایمر کانتر در حالت PWM سریع (تک شیب)
- تایمر کانتر در حالت PWM تصحیح فاز (دو شیب)

رجیستر های تایمر کانتر صفر:

رجیستر مقایسه خروجی **OCR0** این رجیستر هشت بیتی خواندنی و نوشتنی بوده و به طور مستقیم با مقدار شمارنده TCNT0 مقایسه می شود، از تطابق این دو برای تولید وقفه خروجی یا تولید یک شکل موج روی پایه OC0 می توان استفاده نمود. رجیستر تایمر کانتر صفر **TCNT0** این رجیستر هشت بیتی امکان دسترسی مستقیم برای خواندن و نوشتن در شمارنده را فراهم می کند. رجیستر کنترلی تایمر کانتر صفر **TCCR0** که دارای هشت بیت کنترلی است و برای انتخاب پالس ساعت، حالت خروجی هنگام تطابق مقایسه، عملکرد های PWM، و بیت مقایسه خروجی. به طور کلی با این ۸ بیت حالت های مختلفی می توان به وجود آورد که با استفاده از codewizard این رجیستر تنظیم می شود. رجیستر پرچم وقفه تایمر کانتر صفر **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 codewizard تنظیم می شوند.

با استفاده از رجیستر TCCR0 می توان فرکانس کاری تایمر را انتخاب کرد که حالت های زیر را دارد:

	0	1	2	3	4	5	6	7
TCCR0	CS00	CS01	CS02	_	_	_	_	_

توضیحات	CS00	CS01	CS02
بدون کلاک (توقف تایمر کانتر صفر)	0	0	0
بدون تقسیم کلاک	1	0	0
کلاک تقسیم بر ۸	0	1	0
کلاک تقسیم بر ۶۴	1	1	0
کلاک تقسیم بر۲۵۶	0	0	1
کلاک تقسیم بر ۱۰۲۴	1	0	1
کلاک خارجی (بر روی پین T0 با لبه پایین رونده)	0	1	1
کلاک خارجی (بر روی پین T0 با لبه بالا رونده)	1	1	1

جدول ۷-۱ انتخاب کلاک تایمر کانتر صفر

کلاک چیست و چرا باید از تقسیم کلاک استفاده کنیم؟

کلاک یا همان پالس ساعت، یک سیگنال مربعی با فرکانس ثابت (ثابت بدین معنی که نوسانات ندارد) می باشد که متناسب با آن واحد شمارنده CPU با یک سرعت ثابت دستورات ذخیره شده در حافظه فلش میکروکنترلر را اجرا می کند. هرچه این فرکانس بالاتر باشد سرعت اجرای دستورات نیز بالاتر خواهد رفت.

در بسیاری از موارد لازم است که میکرو با حداکثر سرعت دستورات را اجرا نماید تا ما بتوانیم از کوچکترین زمان‌ها استفاده لازم را داشته باشیم. همانگونه که سرعت اجرا بالا می‌رود سرعت شمارش تایمر کانتر نیز افزایش می یابد، که این مسئله در بسیاری از موارد ما را دچار مشکل خواهد کرد و نمی توانیم به زمان سنجی هایی با تاخیر زیاد دسترسی داشته باشیم بنابراین به کمک مقسم های فرکانسی تایمر کانتر، فرکانس کاری بخش تایمر مورد نظر را کاهش داده تا بتوان حداکثر استفاده را از میکروکنترلر برد.

ایجاد تاخیر دقیق به بدون وقفه (تایمر صفر)

تاخیر ۱ میلی ثانیه بدون وقفه با تایمر صفر **(Proj-11)** | **باز کردن پوشه مربوطه**

۱. **انتخاب یک تقسیم فرکانسی مناسب**. باید به گونه ای باشید که بتوان بیشتر بازه عددی را در رجیستر TCNT0 داشته باشیم. یعنی اینکه تایمر با چند شمارش سرریز نشود بلکه شمارش های زیادی را برای سریز انجام دهد، این مسئله بخصوص در وقفه تایمر بسیار مهم می باشد.
۲. **با استفاده از عدد کلاک و مقسم فرکانس** باید یک تاخیر پایه (Base) ایجاد نمود تا به کمک TCNT0 بتوان تعدای از این تاخیر را با هم جمع کرد و به تاخیر مورد نظر دست یابیم.

با رعایت دو اصل فوق می توان به راحتی تاخیر های مورد نظر را ایجاد کرد. به عنوان مثال می خواهیم با کلاک ۸ مگاهرتز و تایمر صفر ۱میلی ثانیه تاخیر دقیق ایجاد کنیم، محاسبات آن را به صورت زیر انجام می دهیم:

F=8000000 ➔ N=64 ➔ F_T=8000000/64=125000Hz

T=1/125000=8us هر پله شمارش

1ms/8ms=125 پله ➔ 255-125=130 کل پله ها ➔ TCNT0=130

بنابراین اگر شمارنده تایمر ۱۲۵ پله را طی کند ۱میلی ثانیه زمان خواهد گذشت. از آنجایی که TCNT0 هشت بیتی است و ماکزیمم آن عدد ۲۵۵ می باشد و همچنین بعد از رسیدن به ۲۵۵ سرریز رخ خواهد داد ما باید تعداد پله های محاسبه شده را از کل پله ها کم کنیم، در واقع شمارش از ۱۳۰ تا ۲۵۵ همان ۱۲۵ پله خواهد بود.

تاخیر ۱میلی ثانیه را در به صورت یک تابع نوشته ایم و در هر جایی از برنامه که به این زمان نیاز داشتیم با فراخوانی تابع timer0_1ms زمان مذکور را ایجاد می‌کنیم. با فراخوانی پی در پی این تابع می توانیم یک سیگنال مربعی با فرکانس ۵۰۰ هرتز بر روی یکی از پین های خروجی ایجاد کنیم.

```
void timer0_1ms(){
        TCCR0=0x03; // Clock value: 125.000 kHz
        TCNT0=130; //255-125=130
        while(!(TIFR & 0x02));
        TIFR |= 0x02;
    }
```

ایجاد تاخیر دقیق با وقفه (تایمر صفر)

تاخیر ۱ میلی ثانیه با وقفه تایمر صفر (Proj-12)

باز کردن پوشه مربوطه

محاسبات با وقفه نیز مانند حالت قبل می‌باشد با این تفاوت که دیگر نیازی نیست منتظر سریز تایمر بمانیم، بنابراین می توانیم به پردازش سایر برنامه بپردازیم و هرگاه تایمر سرریز شد وقفه‌ای تولید می‌شود و CPU برنامه در حال اجرا را رها کرده و برنامه های درون تابع وقفه را اجرا خواهد کرد، سپس به اجرای ادامه برنامه های قبلی خواهد پرداخت.

این رها کردن برنامه در حال اجرا در مواردی برای ما مشکل ایجاد خواهد کرد، بخصوص در مواردی که فاصله زمانی بین اجرای وقفه ها خیلی خیلی کم باشد. به عنوان مثال اگر CPU در حال ارسال اطاعات به نمایشگر LCD باشد و در این لحظه وقفه رخ دهد باعث بهم ریختگی در LCD خواهد شد که بهترین راهکار برای حل این مشکل این است که تابع ;(lcd_init را به صورت مداوم در برنامه اجرا کنیم.

```
interrupt [TIM0_OVF] void timer0_ovf_isr(void){
     TCNT0=130;
     PORTA.0=~PORTA.0;
    }
```

مسئله مهم دیگر این است که اگر شروع اولیه TCNT0 از صفر نیست باید آن را در تابع وقفه مقدار دهی کنیم.

تایمر کانتر صفر در حالت CTC

مکمل سازی OCR0 به روش CTC هر ۳۰ میلی ثانیه (Proj-13)

باز کردن پوشه مربوطه

در حالت مقایسه رجیستر TCNT0 به طور دائم با رجیستر OCR0 مقایسه، و در صورت تطابق (مساوی شدن) رجیستر TCNT0 برابر صفر می شود. از این نتیجه مقایسه می توان برای تولید شکل موج روی پایه خروجی OC0 استفاده نمود که خود چهار حالت دارد:

- در صورت تطابق هیچ عملی روی OC0 صورت نگیرد. Disconnected
- در صورت تطابق وضعیت پین OC0 معکوس شود. Toggle
- در صورت تطابق پین OC0 فقط صفر شود. Clear
- در صورت تطابق پین OC0 فقط یک شود. Set

در هنگام تطبیق مقایسه در صورت فعال بودن وقفه و تولید پرچم OCF0 می توان یک وقفه مقایسه را ایجاد نمود و از روال وقفه برای بروزرسانی مقدار رجیستر OCR0 استفاده نمود. به هر حال تغییر رجیسیتر OCR0 به یک مقدار جدید در زمانی که تایمر در حال شمارش است باید با احتیاط انجام شود، زیرا که حالت CTC دارای بافر مضاعف نمی باشد.

هدف ما تولید یک شکل موج می باشد که طول هر نیم سیکل آن 30ms است. یعنی 30ms پین در وضعیت یک و 30ms دیگر در وضعیت صفر. البته چون 30ms عدد نسبتا بالایی است و با توجه به اینکه فرکانس نوسان ساز ما 8MHz می باشد پس ما N=1024 قرار می دهیم تا فرکانس کاری تایمر برابر 7.813KHz شود تا بتوانیم این تاخیر بالا را ایجاد کنم. همانطور که می دانید با انتخاب N بالا فرکانس کاری تایمر پایین خواهد آمد و درنتیجه زمان پر شدن رجیستر ها بالا می رود.

F=8000000 ➔ N=1024 ➔ F_T=8000000/1024=7812.5Hz
T=1/7812.5=128us هر پله شمارش
30ms/128us=234 پله ➔ OCR0 =234

```
void main(void){
    PORTB=0x00;
    DDRB=0x08;
    TCCR0=0x1D;
    TCNT0=0x00
    OCR0=234;
    while (1);
}
```

نکته: از آنجایی که از OCR0 بر روی PORTB.3 قرار دارد باید این بیت را خروج تعریف کنیم.

تایمر کانتر صفر در حالت PWM

تولید موج PWM با عرض (Proj-13)

باز کردن پوشه مربوطه

PWM مخفف Pulse Width Modulation یعنی مدولاسیون پهنای باند می باشد که در بعضی مواقع به آن Pulse Duration Mode نیز می‌گویند. در این مدولاسیون پهنای پالس تولیدی را می توان تحت کنترل داشت. از کاربردهای PWM می توان به کنترل دور موتورهای AC و DC و منابع تغذیه سوئیچینگ و ... اشاره کرد. برای استفاده و راه اندازی PWM باید بیت های WGM01 , WGM00 را از رجیستر TCCR0 که ۴ حالت ایجاد می کنند را مطابق جدول زیر تنظیم کرد.

SET شدن TOV0 در	بروزرسانی OCR0 در	حد بالا	مد عملکرد	WGM00	WGM01	حالت
ماکزیمم	آنی و فوری	0xFF	عادی	0	0	0
حد پایین	حد بالا	0xFF	PWM تصحیح فاز	1	0	1
ماکزیمم	آنی و فوری	OCR0	مقایسه	0	1	2
ماکزیمم	حد بالا	0xFF	PWM سریع	1	1	3

جدول ۷-۲ انتخاب مدهای PWM

در حالت ۳ PWM سریع انتخاب شده و همانطور که در قسمت CTC توضیح دادیم رجیستر OCR0 به طور دائم با رجیستر TCNT0 مقایسه می شود و پس از برابر شدن بیت سریز مقایسه خروجی (OCF0) فعال شده و پایه OC0 مطابق با تنظیمات مربوطه تغییر وضعیت می دهد. همانطور که در جدول بالا مشاهده می کنید در حالت ۳ شماره تا xFF۰ ادامه می یابد و بعد از سریز تایمر TCNT0=0 خواهد شد و دوباره شماره ادامه داده می شود. در حالت PWM سریع پایه OC0 در تایمر صفر دارای حالت های زیر می باشد که ما در برنامه می توانیم آنها را با توجه به نیازمان انتخاب کنیم:

۱. OC0 قطع باشد
۲. OC0 معکوس نشود.
۳. OC0 معکوس شود

نکته: توجه داشته باشید که در تایمر/کانتر ۱و۲ حالت های متفاوت دیگری وجود دارد.

فرکانس خروجی PWM را می توان از طریق رابطه زیر بدست آورد:

FPWM=(fclk_IO)/(N*256)

N ضریب تقسیم کلاک سیستم است و برابر با یکی از اعداد ۱،۸،۳۲،۶۴،۱۲۸و۲۵۶،۱۰۲۴ می باشد و fclk_IO هم همان فرکانس کلاک میکروکنترلر می باشد.

منظور از عرض ۳۰٪ یعنی اینکه از ۱۰۰٪ یک پالس کامل، مقدار ۳۰٪ آن در وضعیت ۱ باشد و ۷۰٪ دیگر آن در وضعیت صفر باشد، که با یک تناسب ساده عددی بدست آوریم و با گذاشت این عدد در OCR0 می توانیم به عرض ۳۰٪ دست یابیم.

OCR0=(255*30)/100=76
FPWM=(8000000)/(1024*256)=30Hz فرکانس موج تولید شده

```
void main(void){
   PORTB.3=0;
   DDRB.3=1;//OC0 OUT PUT
   TCNT0=0;
   OCR0=76;
   TCCR0=0x6D;// Clock value: 7.813 kHz & NON INVERT
   while (1);
  }
```

تنظیمات در Codewizard:

Clock Source: System Clock
Clock Salue: 7.813kHZ Mode:
Fast PWM top=ffh
Output: Non inverted PWM 0 Overflow Intrrupt 0 Compare Match Interrupt
Timer Value: 0h Compare: 0h

چون عدد OCR0 اعشاری شد پس پالس ما دقیقا ۳۰٪ نخواهد بود که اگر این پالس را با دقت بالاتری نیاز داشته باشیم باید با استفاده از وقفه تایمر صفر اینکار را انجام داد.

ایجاد تاخیرهای دقیق با تایمرهای صفر و یک

در این مطلب تعداد زیادی مثال با تایمرهای صفر (۸ بیتی) و یک (۱۶ بیتی) جهت ایجاد زمان دقیق یک ثانیه برنامه نویسی شده اند. در این مثال ها همه به صورت وقفه و همه به صورت چک مداوم برنامه ها را نوشته ایم، بنابراین با مطالعه این مطلب و مشاهد برنامه ها دیگر نباید مشکلی در ایجاد تاخیر های دقیق با تایمر وجود داشته باشد. در ادامه ما با استفاده از یک میکروکنترلر atmega16 و یک عدد LED به همراه یک اسیلوسکوپ برای تک تک حالت ها برنامه های جداگانه با فایل های شبیه سازی جدا گانه ایجاد کرده ایم، تا این مسائل را به روشنی حل کنیم.

اولین نکته مهم در ایجاد تاخیرهای دقیق انتخاب مقسم فرکانسی تایمر یا همان Perscale می باشد.

یعنی باید به گونه انتخاب شود که با استفاده از آن بتوان اعداد صحیح و روندی برای شمارنده تایمر انتخاب کرد. برای انجام این کار باید دقت و حوصله زیادی به خرج داد تا Perscale خوبی انتخاب کرد، که خود این مسئله به فرکانس کاری میکرو وابستگی شدیدی دارد، پس برای ایجاد یک زمان دقیق ما نباید در محاسبات حتی از یک ده هزارم اعشار هم صرف نظر کنیم که برای اینکار توصیه میکنیم از ماشین حساب استفاده کنید. (در این صورت شما تاخیر های دقیقی خواهد داشت) در تمامی این مثال ها ما فرکانس کاری میکرو را ۴۰۰۰۰۰۰ هرتز انتخاب می‌کنیم و محاسبات را بر این اساس انجام می دهیم. F=4MHz همچنین برای راحتی کار از Codewizard نرم افزار استفاده می‌کنیم.

مثال های تایمر ۸ بیتی صفر

ایجاد تاخیر ۱ ثانیه با استفاده از وقفه تایمر صفر

تاخیر ۱ ثانیه با استفاده از وقفه تایمر صفر | باز کردن پوشه مربوطه

Perscale=64 انتخاب میکنیم و مد تایمر هم شمارش تا FF می باشد، همچنین وقفه تایمر صفر و وقفه سراسری را هم فعال میکنیم. بنابراین TCCR0=0x03 خواهد شد. حال محاسبات تایمر:

فرکانس کاری تایمر F_{t0}= 4MHz/64=62500Hz

مدت زمان شمارش هر پله تایمر T_{t0}=1/62500=16us

رجیستر تایمر صفر ۸ بیتی است بنابراین ۲۵۰ پله شمارش نیاز می باشد تا ۴ms ایجاد شود 16us*250=4ms

پس اگر ۲۵۰ مرتبه تایمر وقفه صادر کند ما ۱ ثانیه تاخیر دقیق ایجاد خواهیم کرد 1000/4ms=250 ⇒ 1sec=1000ms

عدد ۲۵۰ به وجود آمده در مرحله ۳ و ۴ از نظر مقدار هیچ ربطی به هم ندارند و در این مثال به طور کامل تصادفی شبیه هم شده اند. در مرحله سوم گفتیم که برای ایجاد تاخیر 4ms باید ۲۵۰ پله شمارش شود و همانطور که میدانیم این رجیستر از ۰ تا FF را شمارش می کند بنابراین ما باید در هر بار ایجاد وقفه مقدار اولیه شمارش را ۶ وارد کنیم تا شمارش از این مقدار تا ۲۵۶ ادامه یابد و در نهایت ۲۵۰ پله شمارش شود. تا این مرحله ما 4ms تاخیر ایجاد کردیم و همانطور که مشاهده می‌کنید این عدد صحیح و بدون اعشار می باشد. حال با استفاده از یک متغییر به نام c تعداد وقفه های تایمر را شمارش می کنیم. بنابراین برای ایجاد ۱ ثانیه تاخیر باید ۲۵۰ بار وقفه رخ دهد. به عنوان مثال برای ایجاد نیم ثانیه این عدد به ۱۲۵ تغییر می یابد.

```
while (1){
 if(c>=250) {
  led=~led;
  c=0;
 }
}
```

بنابراین در یک حلقه بینهایت مقدار c چک می شود تا در صورتی که به ۲۵۰ رسیده است وضعیت پورت A.0 را برعکس کند. یعنی هر ثانیه یک بار چشمک خواهد زد. همچنین می توانید با استفاده از اسیلوسکوپ این زمان دقیق را مشاهده نمائید.

ایجاد تاخیر ۱ ثانیه با استفاده از چک مداوم در تایمر صفر (بدون وقفه)

تاخیر ۱ ثانیه تایمر صفر (بدون وقفه) | باز کردن پوشه مربوطه

تمامی مراحل مربوط به محاسبات تایمر دقیقا شبیه به مثال قبلی می باشد. اما منظور از چک مداوم چیست؟ در اینجا چون ما از وقفه استفاده نکرده ایم باید میکروکنترلر را وارد یک حلقه بینهایت کنیم و تنها شرط خروج از این حلقه را یک شدن پرچم سریز، تایمر صفر قرار دهیم. اما در این حلقه بینهایت چه مسائلی باید توسط برنامه نویس رعایت شود!

```
void delay_1s(){
 for(c=0; c<=250; c++){
  CCR0=0x03;
  TCNT0=0x06;
  OCR0=0x00;

  while(!(TIFR & 0x02)){//wait for 1s
//other program with out delay
  };
  TCCR0=0x00; //timer0 off
  TIFR|=0x02; //default flag timer0 ==> change to 1
 }
}
```

برای راحتی کار ما مراحل ایجاد مدت زمان ۱ ثانیه را در یک تابع به نام delay_1s برنامه نویسی می‌کنیم. در مثال قبلی ما با استفاده از دستور if مقدار c را برای ۲۵۰ شدن برسی می کردیم ولی در اینجا با کمک یک حلقه for این شمارش را انجام می دهیم. ابتدا تایمر را روشن می‌کنیم و بعد از اتمام نیز آن را خاموش می کنیم، تنها نکته آن در درحلقه بینهایت می باشد یعنی TIFR،

همانطور که می دانیم این رجیستر مربوط به پرچم های تایمرهای ۰ و ۱ و۲ میکرو می باشد، یعنی در هر کدارم از تایمر ها که سریز شمارش رخ دهید در این رجیستر بیت مربوطه آن یک می شود. حال در تایمر صفر پرچم سرریز تایمر صفر بیت دوم می باشد که مقدار آن ۰۲ می شود. با استفاده از دستور AND بیتی یعنی TIFR & 0x02 تنها بیت دوم رجیستر TIFR یعنی (TOV0) مورد برسی قرار میگیرد و در صورتی که این بیت یک شود طبق شرط حلقه ! مخالف آن یعنی صفر است و حلقه شکسته خواهد شد. اما در صورتی که بیت TOV0 صفر باشد ! مخالف آن یک می شود و حلقه ادامه می یابد. بعد از حلقه بینهایت تایمر را خاموش می کنیم و رجیستر TIFRرا OR بیتی می کنیم یعنی تنها بیت TOV0 صفر خواهد شد و دستور بر روی مابقی بیت ها تاثیری نخواهد داشت.

نکته: بیت TOV0 در هنگام فعال بودن وقفه بعد از رخ دادن وقفه و یک شدن آن به صورت اتوماتیک صفر خواهد شد. اما در حالت چک مداوم باید توسط برنامه نویس این بیت صفر شود تا تایمر برای سرریز بعدی آماده باشد.

در حلقه بینهایت چه مسایلی را باید رعایت کرد؟ در این حلقه ما می توانیم توابع یا سایر دستورات دیگر را فراخوانی کنیم یعنی در همان مدتی که داریم عمل زمان سنجی را انجام می دهیم کار دومی هم به میکرو محول کنیم. در این حلقه نباید از دستورات یا توابعی که مدت زمان اجرای آنها بیشتر یا مساوی ms۴ است استفاده نمود، زیرا باعث گذر زمان شده و میکرو شرط حلقه را با تاخیر چک خواهد نمود، بنابراین مدت زمان ۱ ثانیه ما دقیق نخواهد شد.

با توجه به نکته فوق ما می توانیم در این مدت زمان برای کار دوم میکرو مسائلی همچون: اسکن صفحه کلید، خواندن یا نوشتن بر روی پورتها، استفاده از مبدل آنالوگ به دیجیتال و سایر عملیاتی که مدت زمان اجرای آنها توسط CPU کمتر از ms۴ باشد استفاده نمود. توصیه میکنم که از دستور sprintf و ریفرش LCD توسط دستور lcd_puts استفاده نکنید چون مدت زمان زیادی طول خواهد کشید تا اطلاعات برای LCD ارسال شود در نتیجه زمان سنجی ما دچار اختلال خواهد شد. در هر صورت توصیه من این است که از روش اول یعنی وقفه استفاد کنید، چون در این مد هیچ گونه محدویتی وجود ندارد و کارها با سرعت بیشتری پیش خواهند رفت. (مگر در شرایط خاص)

مثال های تایمر ۱۶بیتی یک

محاسبات این تایمر نیز شبیه تایمر صفر می باشد با این تفاوت که بجای عدد ۲۵۶ باید از عدد ۱۶ بیتی ۶۵۵۳۶ استفاده کرد.

ایجاد تاخیر ۱ ثانیه با استفاده از وقفه تایمر یک

تاخیر ۱ ثانیه با استفاده از وقفه تایمر صفر | باز کردن پوشه مربوطه

چون شمارنده تایمر در اینجا ۱۶ بیتی می باشد بنابراین قارد است از ۰ تا FFFF یعنی ۰ تا ۶۵۵۳۵ را بشمارد. از آنجایی که فرکانس تایمر ۶۲۵۰۰ هرتز شد و مدت زمان شمارش هر پله هم 16us شد بنابراین با تقسیم عدد ۱ بر 16us حاصل ۶۲۵۰۰ پله شمارش خواهد شد پس ۶۵۵۳۶-۶۲۵۰۰=۳۰۳۶ خواهد شد یعنی شروع شمارنده تایمر باید از عدد ۳۰ ۳۶ باشد. همانطور که ملاحضه می کنید در اینجا چون تایمر یک ۱۶ بیتی می باشد و گستره شمارش بالایی دارد دیگر نیازی به یک متغیر c برای شمارش نیست و به راحتی خود تایمر این کار را انجام خواهد داد.

```
interrupt [TIM1_OVF] void timer1_ovf_isr(void){
 TCNT1H=0xBDC >> 8;
 TCNT1L=0xBDC & 0xff;
 led=~led;
}
```

در روتین وقفه تایمر یک هم باید مجددا مقدار شمارنده برای شمارش بعدی با عدد ۳۰۳۶ بارگذاری شود. همچنین در همین روتین ما عمل معکوس سازی پین A.0 را انجام می دهیم.

ایجاد تاخیر ۱ ثانیه با استفاده از چک مداوم در تایمر یک (بدون وقفه)

تاخیر ۱ ثانیه تایمر یک (بدون وقفه) | باز کردن پوشه مربوطه

در این جا هم مانند مثال شماره ۲ می باشد با این تفاوت که بیت TOV1 برابر با ۰۴ می باشد یعنی بیت سوم رجیستر TIFR و مابقی محاسبات نیز مانند حلت قبل می باشد.

```
void delay_1s(){
 TCCR1A=0x00;
 TCCR1B=0x03;
 TCNT1H=0x0B;
 TCNT1L=0xDC;
 ICR1H=0x00;
 ICR1L=0x00;
 OCR1AH=0x00;
 OCR1AL=0x00;
 OCR1BH=0x00;
 OCR1BL=0x00;
 while(!(TIFR & 0x04)){//wait for 1s
//other program with out delay
 };
 TCCR1B=0x00; //timer0 off
 TIFR|=0x04; //default flag timer1 ==> change to 1
}
```

۸ مبدل آنالوگ به دیجیتال ADC

در این مطلب می خواهیم کار با مبدل آنالوگ به دیجیتال میکروکنترلر AVR را به زبان C و با کمک کامپایلر Code Vision توضیح دهیم. هدف ما از این مطلب این است که به چه صورتی از کانال ها مقدار آنالوگ را بخوانیم و بعد از تبدیل به دیجیتال، با این با این مقادیر عملیات مورد نظر را انجام دهیم. در اینجا از پرداختن به مسائل ریز مربوط به ADC صرف نظر می کنیم تا باعث گمراهی افراد تازه کار نشود، ما به بحث در مورد رجیستر ها و نحوی انتخاب ولتاژ مرجع و شرایط و چگونگی انتخاب فرکانس نمونه برداری و ۸ یا ۱۰ بیتی ADC خواهیم پرداخت.

رجیستر کنترلی ADMUX

7	6	5	4	3	2	1	0
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

REFS0,1 از این دو بیت برای انتخاب ولتاژ مرجع ADC استفاده می کنیم که دارای چهار حالت می باشد:

- REFS0=0 و REFS1=0 در این حالت ولتاژی که روی پایه AREF است به عنوان ولتاژ مرجع انتخاب می شود.
- REFS0=1 و REFS1=0 در این حالت ولتاژ پایه AVCC به عنوان ولتاژ مرجع انتخاب می شود.
- REFS0=0 و REFS1=1 بدون استفاده
- REFS0=1 و REFS1=1 در این حالت ولتاژ مرجع داخلی ۲.۵۶ ولت انتخاب می شود.

نکته: دقیق بودن ولتاژ مرجع در تبدیل کردن آنالوگ به دیجیتال نقش بسیار مهمی دارد. دقیق ترین ولتاژ مرجع همان ۲.۵۶ داخلی می باشد البته می توان با استفاده از تثبیت کننده های ولتاژ آن ولتاژ مرجع مورد نظر را ساخت و به پایه AREF داد.

ADLAR از این بیت برای ۸ یا ۱۶ بیتی بودن مقدار خروجی ADC استفاده می شودADCH,ADCL .
MUX0-4 از بیت ها برای انتخاب کانال ورودی و نیز انتخاب بهره تفاضلی استفاده شده است.

رجیستر ADCSRA

7	6	5	4	3	2	1	0
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

ADEN با یک کردن این بیت ADC فعال می شود.
ADSC در مد عملکرد Single با نوشتن یک در این بیت، تبدیل شروع شده و پس از پایان تبدیل به صورت خودکار صفر می شود. در مدFree ، یک کردن این بیت برای شروع تبدیل الزامی است.
ADATE با یک کردن این بیت ADC می تواند به صورت اتوماتیک با لبه بالا رونده منبع تحریک کننده شروع به تبدیل کند. منبع تحریک توسط بیت های ADTSاز رجیستر SFIOR انتخاب می شود.
ADIF بعد از اتمام تبدیل یا تغییر در رجستر داده ADC یک می شود) .از یک شدن این بیت ما متوجه می شویم که عمل تبدیل تمام شده و حالا می توانیم مقدار دیجیتال تبدیل شده را بخوانیم)

ADIE با یک کردن این بیت هرگاه عمل تبدیل به اتمام رسید یه وقفه ای صادر می شود که توسط آن زیر روال وقفه می توان مقدار داده ADC را خواند.

ADPS0-3 از بیت ها برای تعیین پالس ساعت ADC مطابق جدول زیر استفاده می کنیم:

ADPS2	ADPS1	ADPS0	تقسیم بر
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

رجیستر داده ADC (ADCH,ADCL)

در این دو رجیستر اطلاعات خروجی ADC قرار دارند که در ADCL مقدرا سبک و در ADCHمقدار سنگین قرار دارد همچنین با استفاده از ADCW می توانیم محتوای هر دو متغییر را به صورت ۱۶ بیتی بخوانیم.

رجیستر SFIOR

7	6	5	4	3	2	1	0
ADTS2	ADTS1	ADTS0	-	ACME	PUD	PSR2	PSR1

از طریق بیت های ADTS0-2 می توان منبع تحریک برای شروع تبدیل را مطابق جدول زیر انتخاب کرد:

ADTS2	ADTS1	ADTS0	منبع تحریک ADC
0	0	0	مد عملکرد آزاد
0	0	1	تحریک از طریق مقایسه کننده آنالوگ
0	1	0	تحریک از طریق وقفه خارجی صفر
0	1	1	تحریک از طریق تایمر (شمارنده) صفر (در صورتی که نتیجه مقایسه برابر شود)
1	0	0	تحریک از طریق تایمر (شمارنده) صفر (در صورت سریز شدن)
1	0	1	تحریک از طریق تایمر (شمارنده) یک (در صورتی که نتیجه مقایسه B برابر شود)
1	1	0	تحریک از طریق تایمر (شمارنده) یک (در صورت سریز شدن)
1	1	1	تحریک از طریق حالت تسخیر تایمر (شمارنده) یک

نکته: برای داشتن حداکثر تفکیک پذیری (۱۰ بیتی) باید فرکانس پالس ساعت ADC بین 50KHZ تا 200KHZ باشد، که اگر این فرکانس از این محدود خارج شود دقت ADC پایین خواهد آمد.

پس باید توجه داشت که با توجه به فرکانس نوسان ساز میکرو ضریب تقسیمی از جدول ADPS0-2 انتخاب کنیم که فرکانس واحد ADC در محدوده گفته شده قرار گیرد.

در این مثال ما می خواهیم با استفاده از دو عدد سنسور LM35 دمای دو نقطه را اندازه گیری کنیم و روی LCD نمایش دهیم در فایلی که به این مطلب پیوست شده برنامه و سایر فایل های شبیه ساز موجود می باشد.

اندازه‌گیری دما دو نقطه به کمک ADC و سنسور LM35

باز کردن پوشه مربوطه

1 ADC Enabled
Volt Ref: Int
Clock: 125KHz
Auto trigger: None

با استفاده از Code Wizard به صورت بالا ADC را تنظیم می کنیم:

```
#define ADC_VREF_TYPE 0xC0 //H1
SFIOR=0x00; //H2
ADMUX=ADC_VREF_TYPE & 0xff; //H3
ADCSRA=0x86; //H4
```

H1: با نسبت دادن ADC_VREF_TYPE 0xC0 ولتاژ مرجع داخلی انتخاب می شود که بعدا در زیر برنامه read_adc با AND و OR کردن آن با کانال انتخاب شده مقدار رجیستر ADMU ایجاد می گردد.

H2: مد عملکر آزاد انتخاب شده

H3: با AND کردن آن با FF به صورت پیش فرض کانال صفر انتخاب شده و همچنین ADLAR هم صفر می باشد بنابراین خروجی به صورت ADCW شانزده بیتی خواهد بود.

H4: در اینجا ADC فعال شده و همجنین ضریب تقسیم ۶۴ انتخاب می شود بنابراین فرکانس پالس ساعت ADC برابر ۱۲۵KHZ می شود که در محدود مجاز می باشد.

در زیرروال read_adc دستوراتی نوشته شده که بجزء یک خط آن بقیه را خود Code Wizard ایجاد می کند.

```
float t;//H1
ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);//H2
delay_us(10);//H3
ADCSRA|=0x40;//H4
while ((ADCSRA & 0x10)==0);//H5
ADCSRA|=0x10;//H6
t=((ADCW*2.56)/1023)*100;//H7
return t;//H8
```

H1: ایجاد یک متغییر از نوع اعشاری و بزرگ

H2: اگر C0 را با FF ما ADN کنیم نتیجه همان C0 خواهد شده حالا فرض کنید در هنگام فراخوانی این تابع ما ۱ را به عنوان ورودی تابع داده ایم اگر C0 را با ۱ OR کنیم آنگاه ADMUX= C1 خواهد شده و درنتیجه کانال ۱ به همراه ولتاژ مرجع داخلی انتخاب می شود.

H3: مقدار ۱۰ میلی ثانیه تاخیر برای پایدار شده ولتاژ ورودی ADC

H4: معادل باینری ADCSRA=0x86 برابر با ۱۰۰۰۰۱۱۰ می باشد و همچنین معادل باینری X40 برابر با ۰۱۰۰۰۰۰۰ می باشد. با OR کردن OR40۸۶ مقدار ۱۱۰۰۰۱۱۰ را خواهیم داشت می بینیم که بیت شماره ۶ به ۱ تغییر یافت و این بیت همان ADSC می باشد که توضیح دادیم.

H5: با استفاده از یک حلقه بی نهایت شرطی بیت ۴ یعنی ADIF یا همان پرچم اتمام تبدیل را تحت نظر می گیرم تا موقعی که تبدیل تمام و این بیت ۱ شود و شرط حلقه باطل می شود که منجر به خروج از این حلقه بی نهایت می شود.

H6: خوب حالا باید بیت ADIF که یک شده را دوباره صفر کنیم که این کار را با OR کردن بیتی با عدد ۱۰ انجام می دهیم.

H7: خوب الان عمل تبدیل تمام شده و مقدار دیجیتال آن در ADCW قرار دارد چون ولتاژ مرجع را ۲.۵۶ انتخاب کردیم اینجا مقدار ADCW را در ۲.۵۶ ضرب می کنیم و بعد حاصل را بر ۱۰۲۳ تقسیم کرده و در نهایت در ۱۰۰ ضرب می کنم تا دما برحسب درجه سانتیگراد بدست اید. عدد ۱۰۲۳ از فرمول زیر حساب می شود که n=10 بوده یعنی تفکیک پذیری ۱۰بیتی

n^2 - 1 ➔ 2^10 -1 = 1023

H8: مقدار دما که در متغییر t قرار دارد برگشت داده می شود.

در ادامه با ما باشید...

مراجع

www.picpars.com

Help CodeVisionAVR V2.05.3

لطفا با مراجعه به آدرس زیر ما را از نظرات سازنده خود بهرمند سازید. همچنین می‌توانید سوالات، مشکلات و اشکالات این مقاله را به ما گزارش دهید تا در نسخه های بعدی اصلاح کنیم. برای دانلود جدید ترین نسخه این فایل به آدرس زیر مراجعه فرمائید.

HTTP://WWW.PICPARS.COM/351/

نرم افزار **PICPARS TOOLS** ساخت کاراکتر های دلخواه نمایشگرهای متنی، محاسبه مقامت ها، محاسبه کدهای سون سگمنت، محاسبه کدهای **LED** ماتریسی ۸×۸ و ...

HTTP://WWW.PICPARS.COM/334/ **نسخه ۱.۲**

اطلاعات مقاله و نرم افزار ها

- ✓ **مدل میکرو:** سری AVR ها
- ✓ **زبان برنامه نویسی:** سی C
- ✓ **کامپایلر میکرو:** CodeVisionAVR V2.05.3 (دانلود)
- ✓ **شبیه ساز:** Proteus 7.10 SP0 (دانلود)
- ✓ نرم افزار PicPars Tools v1.2 (دانلود)

- ✓ **تاریخ تهیه:** ۱۳۹۱/۰۲/۲۸
- ✓ **تاریخ آخرین ویرایش:** ۱۳۹۱/۰۴/۱۶

- ✓ **نویسنده مقاله:** سید محسن قاسمیان
- ✓ **آدرس ایمیل نویسنده:** picpars@gmail.com

- ✓ **آدرس دانلود مقاله:** http://www.picpars.com/351/

- ✓ **آدرس سایت:** www.picpars.com
- ✓ **آدرس ایمیل سایت:** picpars@gmail.com