آموزش میکروکنترلر AVR
به زبان C

این آموزش بر اساس برد آموزشی

حرفه ای AVR نوشته است

مقدمه

الکترونیک دیجیتال کار خود را با به وجود آوردن منطق صفر و یک، اختراع میکروپروسسورها و طراحی کامپیوترها آغاز کرد. میکروپروسسورهایی نظیر 8086 از شرکت اینتل و Z80 از شرکت زایلوگ، شروع کار بودند.

با ورود خانواده میکروکنترلر 8051 از شرکت اینتل، تحولی عظیم در این صنعت رخ داد و میکروکنترلرها تنها یک پردازشگر نبودند و عملیات محاسبه و منطق تنها بخشی از این تراشه بود. امکاناتی نظیر حافظه‌ها، تایمرها و ارتباطات به این تراشه افزوده شد و این تراشه مانند یک کامپیوتر کوچک به بازار عرضه شد. طولی نکشید که شرکت‌هایی نظیر Micro Chip و Atmel سری جدیدتری از میکروکنترلرها را عرضه کردند.

میکروکنترلرهای ۸ بیتی AVR ساخت شرکت Atmel از پرکاربردترین نوع میکروکنترلرهای موجود در دنیا می‌باشند و دلیل آن وجود امکانات متمایز از سایر میکروکنترلرها است.

تاریخچه میکروکنترلرهای AVR

اولین میکروکنترلر در سال ۱۹۷۱ توسط شرکت نام آشنای Intel ساخته شد و این شرکت اولین میکروکنترلر کاربردی خود را در سال ۱۹۸۰ با نام 8080 روانه بازار کرد.

کلمه میکروکنترلر از دو عبارت میکرو و کنترل تشکیل شده است که اولی واحدی یونانی به معنای یک میلیونم و دومی به معنای تحت نظارت داشتن کاری است.

با توجه به حرکت جوامع بشری به سوی هر چه کوچک‌تر کردن وسایل کاربردی، طراحان الکترونیک به تبعیت از این قانون، سعی در کوچک کردن مدار کنترلی یک پروسه و کاهش هزینه های مربوط نمودند که این امر موجب پیدایش میکروکنترلرها به عنوان وسیله‌ای که دارای حافظه، CPU، پورت‌های ورودی و خروجی و ... در یک چیپ گردید.

ما امروزه شاهد معماری‌های مختلفی از میکروکنترلرها هستیم که مهم‌ترین آن‌ها عبارتند از:

۱- AVR

۲- PIC

۳- 8051

اما تفاوت میکروکنترلرهای ۳ خانواده مذبور علاوه بر تکنولوژی ساختشان، در برنامه نویسی مورد نیاز و نحوه پروگرام کردن آن‌ها می‌باشد.

میکروکنترلرهای AVR

AVR ها میکروکنترلرهایی ۸ بیتی از نوع Cmos با توان مصرفی پایین هستند که بر اساس ساختار پیشرفته RISC با معماری Harvard ساخته شده‌اند.

RISC مخفف (Reduced Instruction Set Computer) به معنی مجموعه دستورالعمل‌های کاهش یافته و Harvard به نوعی معماری گفته می‌شود که در آن حافظه ذخیره برنامه و حافظه ذخیره داده، از هم جدا می‌باشند.

در میکروکنترلرهای AVR دستورات تنها در یک پالس ساعت اجرا می‌شوند و به این ترتیب به ازای هر یک مگاهرتز می‌تواند یک مگا دستور را در ثانیه اجرا کند، در نتیجه برنامه از لحاظ سرعت پردازش و مصرف توان بهینه می‌شود.

این میکروکنترلرها دارای ۳۲ رجیستر همه منظوره و مجموعه دستورات قدرتمندی هستند که تمام این ۳۲ رجیستر مستقیما به ALU (بخش پردازش) متصل شده‌اند، بنابراین دسترسی به دو رجیستر در یک سیکل ساعت هم امکان پذیر بوده که باعث می‌شود سرعت این میکروها نسبت به میکروکنترلرهای CISC تا ۱۰ برابر افزایش یابد.

انواع میکروکنترلرهای AVR

میکروکنترلرهای AVR با دو معماری ۸ بیتی و ۱۶ بیتی ساخته می‌شوند که در اینجا به شرح کارکرد مدل ۸ بیتی می‌پردازیم.

میکروکنترلرهای ۸ بیتی AVR به سه دسته تقسیم می‌شوند:

۱. Tiny AVR
۲. Mega AVR
۳. Xmega AVR

تفاوت بین این سه نوع به امکانات موجود در آن‌ها مربوط می‌شود. Tiny AVR ها غالبا تراشه‌هایی با تعداد پین و مجموعه دستورات کمتری نسبت به Mega AVR می‌باشند و به عبارتی از لحاظ پیچیدگی حداقل امکانات را دارند و Xmega AVRها حداکثر امکانات را داشته و Mega AVRها در بین این دو نوع هستند.

امکانات کلی ATMEGA 32

- ✓ ۳۲ رجیستر همه منظوره.
- ✓ دارای سه نوع حافظه شامل: Flash, EEprom, Sram

- ✓ توانایی برنامه ریزی تراشه در داخل مدار بدون احتیاج به پروگرامر (In System Programing).
- ✓ حفاظت از کدهای برنامه در مقابل خواندن (با قفل فیوزبیت های آن).
- ✓ قابلیت تنظیم نوسانگر برای کار توسط کریستال خارجی و داخلی و نوسانگر RC داخلی و خارجی.
- ✓ مجهز به پروتکل JTAG برای انجام عمل دیباگ، تست و اسکن وسایل جانبی تراشه و ...
- ✓ شمارنده و تایمر ۸ بیتی و ۱۶ بیتی.
- ✓ RTC با نوسانگر جداگانه.
- ✓ کانال‌های PWM با استفاده از تایمرها به صورت ۸ و ۱۶ بیتی.
- ✓ ADC های ۱۰ بیتی.
- ✓ ارتباط سریال USART با قابلیت برنامه ریزی.
- ✓ تایمر watch dog با قابلیت برنامه ریزی با نوسانگر مجزا (WTD).
- ✓ مقایسه کننده آنالوگ با امکان تعریف وقفه برای آن.
- ✓ منابع وقفه داخلی و خارجی.
- ✓ دارای حدود ۱۳۰ دستور که اکثر آن‌ها در یک سیکل ساعت اجرا می‌شوند.

تشریح پایه های ATMEGA 32

در تراشه‌های AVR پایه‌های آن‌ها علاوه بر استفاده به عنوان I\O برای یک یا چند خصوصیت دیگر نیز مورد استفاده قرار می‌گیرند که در زیر به تشریح آن‌ها می‌پردازیم:

- ❖ پایه OC1A:

خروجی مد مقایسه تایمر– کانتر ۱ و نیز خروجی موج PWM1.

- ❖ پایه OC1B:

خروجی مد مقایسه تایمر– کانتر ۱ و نیز خروجی موج PWM2.

- ❖ پایه SCK:

به عنوان کلاک ورودی و خروجی Master و Slave در ارتباط SPI استفاده می‌شود.

- ❖ پایه MISO:

به عنوان ورودی داده میکرو Master و خروجی داده میکرو Slave استفاده می‌شود.

❖ پایه MOSI:

به عنوان خروجی داده میکرو Master و ورودی داده میکرو Slave استفاده می‌شود.

❖ پایه AIN0:

به عنوان ورودی پایه مثبت مقایسه کننده آنالوگ استفاده می‌شود.

❖ پایه AIN1:

به عنوان ورودی پایه منفی مقایسه کننده آنالوگ استفاده می‌شود.

❖ پایه OC0:

در خروجی مد مقایسه‌ای تایمر– کانتر صفر مورد استفاده قرار می‌گیرد.

❖ پایه T0:

در ورودی کلاک برای کانتر صفر استفاده می‌شود.

❖ پایه T1:

در ورودی کلاک برای کانتر یک استفاده می‌شود.

❖ پایه TOSC1:

در زمان استفاده از RTC به این پایه کریستال ۳۲۷۶۸ هرتز وصل می‌شود.

❖ پایه TOSC2:

در زمان استفاده از RTC به این پایه کریستال ۳۲۷۶۸ هرتز وصل می‌شود.

❖ پایه TDI:

ورودی داده سریال در ارتباط JTAG می‌باشد.

❖ پایه TDO:

خروجی داده سریال در ارتباط JTAG می‌باشد.

❖ پایه TMS:

به عنوان ارتباط JTAG استفاده می‌شود.

❖ پایه TCK:

به عنوان ارتباط JTAG استفاده می‌شود.

❖ پایه SDA:

به عنوان خط داده در ارتباط دو سیمه (I2C) استفاده می‌شود.

❖ پایه SCL:

به عنوان خط کلاک در ارتباط دو سیمه (I2C) استفاده می‌شود.

❖ پایه OC2:

مد مقایسه‌ای تایمر– کانتر ۲ و به عنوان خروجی موج PWM2 استفاده می‌شود.

❖ پایه ICP:

به عنوان ورودی Capture تایمر– کانتر ۱ استفاده می‌شود.

❖ پایه RXD:

به عنوان ارسال کننده داده در ارتباط سریال USART استفاده می‌شود.

❖ پایه TXD:

به عنوان دریافت کننده داده در ارتباط سریال USART استفاده می‌شود.

❖ پایه AREF و AVCC:

پایه‌های تعیین کننده ولتاژ مرجع برای مبدل آنالوگ به دیجیتال می‌باشند.

❖ پایه SS:

با فعال شدن در ارتباط SPI میکروکنترلر را به میکروی SLAVE تبدیل می‌کند.

❖ پایه XCK:

به عنوان کلاک خروجی در ارتباط UART در زمان مد آسنکرون استفاده می‌شود.

❖ پایه Reset:

به عنوان پایه‌ای برای ریست کردن میکرو به کار می‌رود.

❖ پایه‌های Xtal1 و Xtal2:

پایه‌هایی جهت اتصال کریستال خارجی به میکرو می‌باشند.

❖ پایه‌های ADC0 تا ADC7:

پایه‌های ورودی مبدل آنالوگ به دیجیتال می‌باشند.

❖ پایه‌های INT0 تا INT7:

پایه‌های ورودی وقفه خارجی می‌باشند.

با توجه به نیاز روز و گرایش عده ی زیادی از دانشجویان برق و الکترونیک برآن شدیم تا اموزشی جامع را در اختیار دوستان قرار دهیم.

مبنای این آموزش بر اساس زبان سی و کامپایلر محبوب کدویژن می باشد و مثال ها و تمرینات آن همگی با استفاده از برد آموزشی AVR پیاده سازی شده است.

امید است که این آموزش شما را در رسیدن هر چه سریعتر به اهدافتان در زمینه میکروکنترلرها راهنمایی کند.

تشریح پورت ها و پین های میکروکنترلر

میکروکنترلرهای AVR جهت تبادل داده با دنیای خارج دارای یک سری پین های کنترلی می باشد. جهت استفاده از این پین ها و سایر امکانات میکروکنترلر باید در ابتدا آن ها را پیکربندی کرد.

هر پورت از میکروکنترلر دارای ۳ رجیستر به شرح زیر است:

رجیستر DDR:

Bit	7	6	5	4	3	2	1	0	
	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	DDRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

توسط این رجیستر می توانیم مشخص کنیم که پورت یا پین مورد نظر در حالت ورودی باشد یا خروجی. اگر در هر بیت ۱ نوشته شود پین مورد نظر خروجی و در هر بیت ۰ نوشته شود پین مورد نظر ورودی خواهد بود.

رجیستر PORT:

Bit	7	6	5	4	3	2	1	0	
	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	PORTA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

با استفاده از رجیستر پورت می توانیم در صورتی که پین به صورت خروجی تعریف شده باشد، اطلاعاتی (شامل صفر و یک) را به دنیای خارج بفرستیم. در هر بیت یک نوشته شود پین مورد نظر در سطح منطقی ۱ و با سطح ولتاژ VCC قرار خواهد گرفت و هر بیت ۰ شود پین مورد نظر در سطح منطقی صفر قرار خواهد گرفت.

رجیستر PIN:

Bit	7	6	5	4	3	2	1	0	
	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	PINA
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

این رجیستر در واقع یک بافر می باشد که می تواند صفر یا یک بودن پین را تشخیص دهد. یعنی اگر پین مورد نظر در حالت ورودی باشد می توانیم از دنیای بیرون میکروکنترلر به دنیای درون میکروکنترلر اطلاعات بفرستیم.

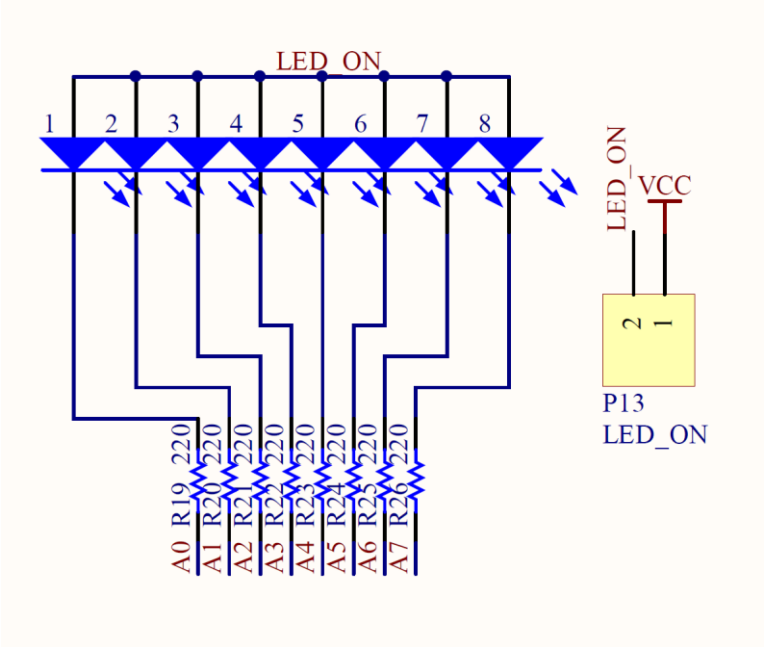
با یک مثال عملی قطعا بیشتر با عملکرد رجیسترها آشنا می شوید.

نکته: مثال ها بر اساس فرکانس اسیلاتور ۸ مگاهرتز نوشته شده اند لذا فیوزبیت کلاک را بر روی اسیلاتور ۸مگاهرتز داخلی تنظیم کنید.

مدارات عملی

۱. مدار یک رقص نور ساده را با ۸ LED طراحی کرده و برنامه مربوطه را بنویسید؟

شماتیک مربوط به اتصال LEDها:



تنظیمات اعمال شده روی برد آموزشی:

جامپر مربوط به قسمت تغذیه در مد ۵ ولت قرار داده شود و جامپر LED نیز متصل گردد.

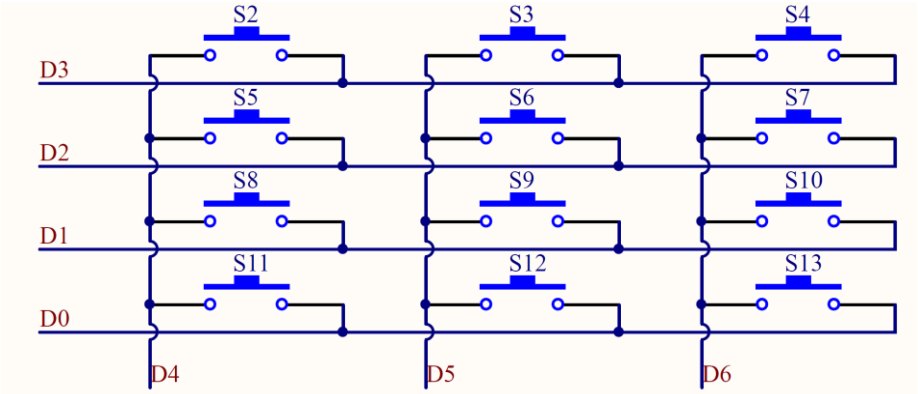
نکته: چون LED ها به پورت A متصل شده‌اند، پورت A را به عنوان خروجی در نظر بگیرید.

```
#include <mega32a.h>
#include <delay.h>
#define xtal 8000000
void main(void)
{
int i=0;
DDRA=0xff;
PORTA=0x00;

while (1)
    {
     for(i=0;i<=7;i++)
     {
      PORTA=~1<<i;
      delay_ms(500);
     }

    }
```

}

۲. با استفاده از یک کلید برنامه ای بنویسید که با فشار کلید و صفر شدن پین مربوطه، یک LED روشن شود؟

شماتیک اتصال کلیدها:

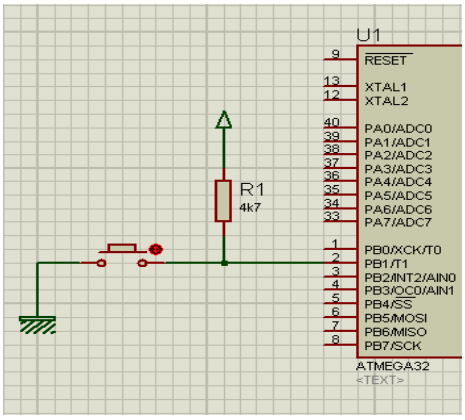


به دلیل اینکه کلیدها به صورت ماتریسی به هم متصل شده اند در نتیجه یک ستون را به صورت دستی صفر کرده و سپس می توانیم کلیدهای ردیف را چک کنیم.

نکته:

Pullup و pulldown چیست؟

وقتی که می خواهیم یک کلید را به میکروکنترلر وصل کنیم در اصل می خواهیم وجود یا عدم وجود یک سطح منطقی را بررسی کنیم. به فرض مثال ما یک طرف کلید را به کنترلر و طرف دیگر آن را به زمین وصل کرده ایم درنتیجه هدف ما وجود یا عدم وجود سطح صفر بر روی پین مورد نظر است. برای اینکه سطح صفر را تشخیص دهیم قطعا باید قبلا روی پین مورد نظر سطح یک را ایجاد کرده باشیم تا بتوان دو سطح را با هم مقایسه کرد. ایجاد این حالت اولیه و ثابت در صورت عدم وجود فشار کلید برای جلوگیری از نویز و تشخیص صحیح سطح ولتاژ توسط یه مقاومت انجام می شود که به آن مقاومت pullup گفته می شود.



مقاومت pulldown دقیقا حالتی عکس این خواهد داشت.

مقاومت pullup به صورت داخلی در میکروکنترلر AVR جاگذاری شده است و در صورت استفاده میکرو در محیط عادی می توان به استفاده از همین مقاومت داخلی کفایت کرد.

نحوه استفاده از آن به این ترتیب است که ابتدا پورت یا پین مورد نظر را در حالت ورودی قرار داده و در رجیستر PORT آن عدد یک را به ازای هر پین قرار می دهیم.

در این مثال می خواهیم از کلید متصل به D0 استفاده کنیم پس ابتدا D0 را ورودی کرده و در رجیستر PORTD.0 یک می نویسیم.

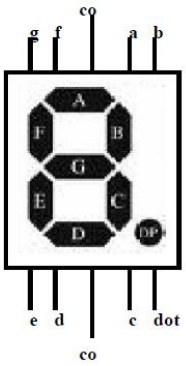
```
#include <mega32a.h>
#include <delay.h>
#define xtal 8000000
void main(void)
{

DDRA=0xff;
PORTA=0xff;
PORTD=0xf3;
DDRD=0x10;

while (1)
{
if(PIND.0==0)
PORTA=~0x01;
else
PORTA=0xff;
}
}
```

اتصال سون سگمنت به میکروکنترلر

سون سگمنت از ۸ LED تشکیل شده است که از ۷ عدد آن برای نمایش اعداد و حروف A تا F و از Led هشتم برای نمایش ممیز (Dot) استفاده می‌شود.

هر سون سگمنت تک رقمی دارای ۱۰ پایه به شرح زیر است:

- ۷ پایه که با حروف a تا g نام گذاری شده‌اند.
- ۱ پایه که با Dot نام گذاری شده است.
- ۲ پایه که پایه‌های مشترک بوده و در داخل IC به هم متصل می‌باشند.



سون سگمنت ها به دو دسته تقسیم می‌شوند:

۱- آند مشترک: پایه آند هر ۸ LED در داخل به هم وصل است و پایه کاتد آن‌ها آزاد می‌باشد.

۲- کاتد مشترک: پایه کاتد هر ۸ LED در داخل به هم وصل است و پایه آند آن‌ها آزاد می‌باشد.

اتصال سون سگمنت به میکرو و نمایش عدد بر روی آن

برای اتصال سون سگمنت به میکرو دو راه وجود دارد :

- اتصال پایه های a تا g مستقیم به یکی از پورت‌ها.
- استفاده از ای سی‌های دیکودر مانند ۷۴۴۷ و ۷۴۴۸.

نمایش اعداد تک رقمی روی سون سگمنت به روش معمولی

در این روش برای نمایش هر رقم یا حرف روی سون سگمنت ابتدا بایستی کد هگزادسیمال معادل آن را بدست آوریم. برای این کار بایستی به دو نکته توجه داشته باشیم:

۱- آند یا کاتد مشترک بودن سون سگمنت.

۲- در نمایش هر رقم یا حرف کدام LEDها روشن و کدام LEDها خاموش خواهند شد.

با توجه به دو نکته بالا کد هگزادسیمال را برای ارقام ۰ تا ۹ جهت نمایش توسط سون سگمنت کاتد مشترک را بدست می‌آوریم:

رقم	HEX	A	B	c	d	e	f	g	dot
شماره پین		PA5	PA4	PA6	PA7	PA0	PA2	PA1	
۰	F5	1	1	1	1	1	1	0	0
۱	50	0	1	1	0	0	0	0	0
۲	B3	1	1	0	1	1	0	1	0
۳	F2	1	1	1	1	0	0	1	0
۴	56	0	1	1	0	0	1	1	0
۵	E6	1	0	1	1	0	1	1	0
۶	C7	1	0	1	1	1	1	1	0
۷	70	1	1	1	0	0	0	0	0
۸	F7	1	1	1	1	1	1	1	0
۹	76	1	1	1	1	0	1	1	0

نمایش اعداد تک رقمی روی سون سگمنت با استفاده از دیکودرها

برای انجام این روش به دو نکته بایستی توجه داشت:

- انتخاب دیکدر مناسب با توجه به نوع سون سگمنت.
- وصل خروجی‌های دیکدر(QA…..QG) نظیر به نظیر به ورودی‌های سون سگمنت (a…..g).

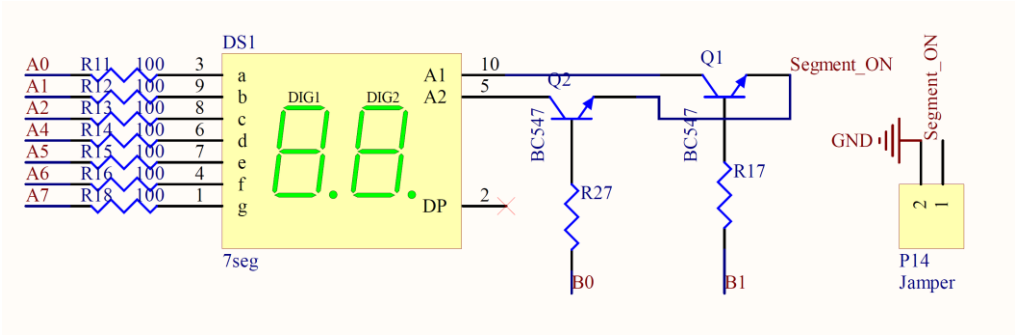
برای راه اندازی سون سگمنت آند مشترک از دیکدر ۷۴۴۷ و برای راه اندازی سون سگمنت کاتد مشترک از دیکدر ۷۴۴۸ استفاده می‌شود.

در برنامه نویسی برای نمایش عدد روی سون سگمنت با استفاده از دیکدر مستقیم از معادل دسیمال اعداد در برنامه استفاده می‌شود.

مدارات عملی :

۱- برنامه‌ی یک شمارنده ۰ تا ۹ با نمایش روی سون سگمنت را بنویسید؟

شماتیک:



تنظیمات اعمال شده روی برد آموزشی:

جامپر تغذیه را در حالت ۵ ولت قرار داده و همچنین جامپر 7seg را وصل کنید.

```
#include <mega32a.h>
#include <delay.h>
#include <alcd.h>
#define xtal 8000000

void main(void)
{

int i;
char segment[10]={0xf5,0x50,0xb3,0xf2,0x56,0xe6,0xC7,0x70,0xF7,0x76};

PORTA=0x00;
DDRA=0xff;
```

```
PORTB=0xff;
DDRB=0xff;

while (1)
    {
    for(i=0;i<=9;i++)
    {
     PORTA=segment[i];
     delay_ms(500);
    }
    }
}
```

نمایش اعداد چند رقمی روی سون سگمنت

برای نمایش اعداد چند رقمی روی سون سگمنت یک راه این است که به ازای هر رقم از یک پورت استفاده کنیم اما چون تعداد پورت‌های یک میکرو محدود است این روش، روش مناسبی نیست. همانطور که می‌دانیم چشم انسان در صورتی که ۲۵ تصویر یا بیشتر از یک شی پشت سر هم در یک ثانیه پخش شود آن را پیوسته می‌بیند و می‌توان از این خطای چشم استفاده کرد. که به روش مالتی پلکس کردن معروف است.

در این روش خطوط دیتا یعنی پایه های a تا g را به یکی از پورت‌ها وصل کرده و پایه های دیگر میکرو برای کنترل پایه مشترک سون سگمنت ها مورد استفاده قرار می‌گیرد. در این روش در هر لحظه فقط یک سون سگمنت روشن است و بقیه‌ی سون سگمنت ها خاموش می‌باشند ولی چون این عمل با سرعت بالا انجام می‌گیرد ما احساس می‌کنیم که همه‌ی آن‌ها روشن هستند.

مدارات عملی :

۱) برنامه ای برای نمایش عدد ۲۳ بر روی سون سگمنت های موجود را بنویسید؟

```
#include <mega32a.h>
#include <delay.h>
#include <alcd.h>
#define xtal 8000000

void main(void)
```

```
{
char segment[10]={0xf5,0x50,0xb3,0xf2,0x56,0xe6,0xC7,0x70,0xF7,0x76};
PORTA=0x00;
DDRA=0xff;
PORTB=0xff;
DDRB=0xff;

while (1)
    {
     PORTA=segment[2];
     PORTB=0x02;
     delay_ms(10);
     PORTB=0x00;
     PORTA=segment[3];
     PORTB=0x01;
     delay_ms(10);
     PORTB=0x00;
    }
}
```

نکات مهم در مورد کار با سون سگمنت ها:

- اگر یکی از سون سگمنت ها نسبت به بقیه نور کمی داشت زمان تاخیر یا همان Wait برنامه (روشن و خاموش بودن سون سگمنت ها) را درست انتخاب نکرده‌اید.
- برای کنترل سون سگمنت ها باید طبق مدارات بالا عمل کنید یعنی با استفاده از یک ترانزیستور آن‌ها را راه اندازی کنید چون با اتصال مستقیم آن‌ها به پایه‌های میکرو جریان زیادی را از پایه کشیده و باعث سوختن پایه مورد استفاده می شود.
- در طراحی شمارنده‌های خودکار که به صورت اتوماتیک کاهش یا افزایش می‌یابند باید تاخیری که در بین برنامه قرار می‌گیرد خیلی کم باشد در غیر اینصورت نمی‌توان روی سون سگمنت ها عدد مورد نظر را مشاهده کرد.

LCDهای کاراکتری و راه اندازی آن‌ها توسط میکروکنترلرهای AVR

LCDهای کاراکتری نمایشگرهایی با سطر و ستون مشخص برای نمایش داده‌ها می‌باشند.

در تمام این LCD تعداد پایه‌ها ثابت و برابر ۱۶ پایه بوده که نحوه اتصال آن‌ها به شرح زیر است:

نحوه اتصال پایه	سمبول	شماره پایه
اتصال به زمین	Vss	۱
اتصال به +5V	Vdd	۲
تنظیم کنتراست LCD	Vo یا VEE	۳
کنترل رجیستر	RS	۴
انتخاب مد خواندن یا نوشتن	RW	۵
فعال سازی LCD	E	۶
گذرگاه 8 تایی اطلاعات و دستورالعمل	D0 - D7	۷ – ۱۴
آند و کاتد LED پس زمینه	کاتد LED، آند LED	۱۵ ،۱۶

در این LCD می‌توان آن را به دو صورت به میکرو وصل کرد:

۱- مد ۴ سیمه: از چهار پایه گذرگاه برای اتصال LCD به میکرو استفاده می‌شود.

۲- مد ۸ سیمه : از هشت پایه گذرگاه برای اتصال LCD به میکرو استفاده می‌شود.

معایب و محاسن:

در مد هشت سیمه سرعت انتقال داده بیشتر است اما پایه بیشتری از میکرو را اشغال می‌کند ولی در مد چهار سیمه سرعت انتقال داده کمتر است اما پایه کمتری از میکرو را اشغال کرده و شلوغی سیم کشی مدار نیز نسبت به مد هشت سیمه کمتر است.

تقریبا در اکثر موارد از مد چهار سیمه استفاده می‌شود چون سرعت انتقال داده‌ی آن برای کارهایی که ما با میکرو انجام می‌دهیم مناسب است.

نحوه سیم بندی ارتباط LCD با میکرو در مد چهار سیمه

۱- خطوط دیتای Db4 الی Db7 جهت دریافت داده‌ها به ۴ پین از یک پورت میکرو که به صورت خروجی تعریف شده متصل می‌گردند.

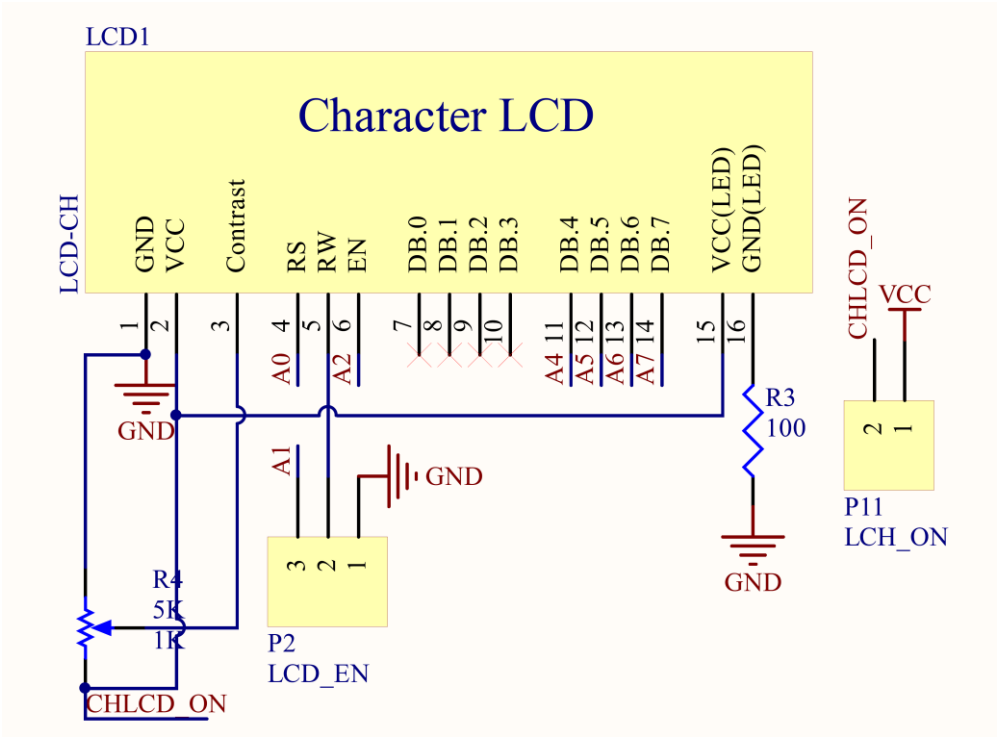
۲- خط RS برای دریافت دستورالعمل به یک پایه از پورت متصل می‌شود.

۳- خط E برای فعال سازی LCD به یک پایه از پورت متصل می‌شود.

۴- خط RW جهت مد خواندن یا نوشتن در نمایشگر که به یک پایه از پورت متصل می شود.

۵- خطوط Vss و Vdd به ترتیب به GND و +5V جهت تغذیه LCD متصل می‌شوند.

۶- خط Vo جهت تنظیم کنتراست به یک پتانسیومتر ۵ تا ۱۰ کیلو اهم متصل می‌شود.



نکات مهم:

- جهت تنظیم کنتراست هیچ گاه آن را مستقیم به Vcc یا GND نزنید.
- پایه‌های ۱۵ و ۱۶ مربوط به LED پس زمینه را مستقیم به Vcc و GND وصل نکنید چون جریان زیادی از مدار می‌کشند. در مسیر Vcc یا GND یک مقاومت در حد ۱۰۰ اهم قرار دهید.

جهت راحتی کاربر در کامپایلر کدویژن کتابخانه ای برای کار با نمایشگرهای کاراکتری در نظر گرفته شده که یک سری از توابع پایه ای آن را توضیح خواهیم داد.

جهت استفاده از این کتابخانه می توانید با بهره گیری از کدویزارد تنظیمات دلخواه را انجام داده و به صورت اتوماتیک کتابخانه و سایر تنظیمات اضافه گردد و یا به صورت دستی کتابخانه را اضافه کنید. فقط دقت داشته باشید در حالتی که دستی کتابخانه را اضافه می کنید باید از منوی LCD project/configure کاراکتری را فعال کنید و پین های مورد استفاده را برگزینید.

عملکرد	فرم کلی تابع
آماده سازی اولیه نمایشگر	Void lcd_init(unsigned char lcd_columns);
پاک کردن صفحه نمایش	void lcd_clear(void);
چاپ تک کاراکتر بر روی نمایشگر	void lcd_putchar(char c);
چاپ رشته موجود در حافظه sram	void lcd_puts(char *str);
چاپ رشته موجود در حافظه flash	void lcd_putsf(char flash *str);
پرش مکان نما به سطر و ستون دلخواه	void lcd_gotoxy(unsigned char x, unsigned char y);

مدارات عملی :

برنامه‌ای بنویسید که بتوان در دو سطر یک LCD ۲*۱۶ پیغامی را چاپ کند؟

تنظیمات روی برد:

جامپر تغذیه را در حالت ۵ ولت قرار داده، جامپر CHLCD را متصل کنید و نیز جامپر LCD_EN را به A1 متصل کنید.

```
#include <mega32a.h>
#include <delay.h>
#include <alcd.h>
void main(void)
{
PORTA=0x00;
DDRA=0xff;
// Alphanumeric LCD initialization
// Connections are specified in the
// Project|Configure|C Compiler|Libraries|Alphanumeric LCD menu:
// RS - PORTA Bit 0
```

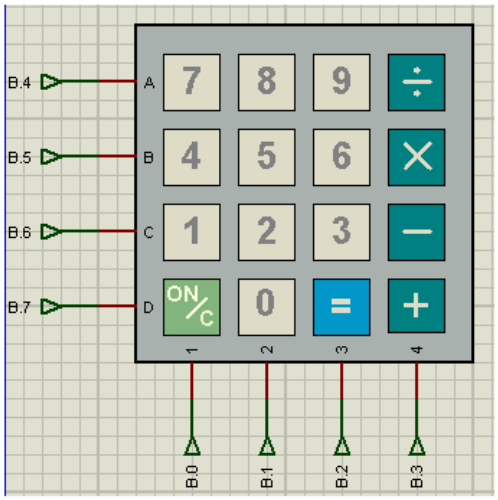
```
// RD - PORTA Bit 1
// EN - PORTA Bit 2
// D4 - PORTA Bit 4
// D5 - PORTA Bit 5
// D6 - PORTA Bit 6
// D7 - PORTA Bit 7
// Characters/line: 16
lcd_init(16);
lcd_clear();
lcd_gotoxy(0,0);
lcd_putsf("char lcd test!!");
lcd_gotoxy(3,1);
lcd_putsf("www.ECA.ir");
while (1);
}
```

اتصال کی‌پد به میکروکنترلر

همانطور که در بالا توضیح داده شد برای اتصال وسایل I/O خارجی به میکروکنترلر ابتدا باید آن را پیکربندی نمود.

کی‌پدها شامل کلیدهایی هستند که به صورت ماتریسی به هم متصل شده‌اند که باعث شده تعداد پایه‌های کلیدها کمتر شده و پورت کمتری را در میکروکنترلرها اشغال کند.

کی‌پدها در انواع گوناگونی در بازار یافت می‌شوند که متداول‌ترین آن‌ها کی‌پدهای ۴*۳ و ۴*۴ می‌باشد. در زیر نحوه اتصال کی‌پد ۴*۴ و یک نمونه از کی‌پدهای 4*4 موجود در بازار، نمایش داده شده است.



بر روی بردی که در اختیار دارید یک کی‌پد ۴*۳ قرار داده شده است که به پورت D متصل می‌باشد و به راحتی می‌توان از آن استفاده کرد. بنابراین دیگر نیاز به تهیه کی‌پد خارجی نخواهد بود.

مدارات عملی :

۱) برنامه‌ای بنویسید که اعداد وارد شده توسط یک کی‌پد ۴*۳ را روی LCD نمایش دهد؟

تنظیمات روی برد:

جامپر تغذیه را بر روی ۵ ولت قرار داده و جهت فعال سازی LCD کاراکتری جامپرهای CHLCD و LCD_EN را متصل کنید.

```
#include <mega32a.h>
#include <delay.h>
#include <alcd.h>
#include <stdio.h>
#define xtal 8000000
char buffer[];
int data;
void ref_key(void)
{
  data=12;
  DDRD=0x0f;
  PORTD=0xff;
  PORTD.0=0;
  delay_ms(10);
  if(PIND.4==0)data=10;
  if(PIND.5==0)data=0;
  if(PIND.6==0)data=11;
  PORTD=0xff;
  PORTD.1=0;
  delay_ms(10);
  if(PIND.4==0)data=7;
  if(PIND.5==0)data=8;
  if(PIND.6==0)data=9;
  PORTD=0xff;
  PORTD.2=0;
  delay_ms(10);
  if(PIND.4==0)data=4;
  if(PIND.5==0)data=5;
  if(PIND.6==0)data=6;
  PORTD=0xff;
  PORTD.3=0;
  delay_ms(10);
  if(PIND.4==0)data=1;
  if(PIND.5==0)data=2;
  if(PIND.6==0)data=3;
}
```

```
void main(void)
{

lcd_init(16);
while (1)
{
  ref_key();
  if(data<12)
  {
    sprintf(buffer,"%i",data);
    lcd_clear();
    lcd_puts(buffer);
  }
}
}
```

شروع کار با ADC

گاهی نیاز است که یک کمیت بیرونی مانند دما، شدت نور و ... اندازه گیری شود. جهت این کار از سنسورهای مربوطه استفاده می‌شود، سنسورها کمیت مورد نظر را به ولتاژ یا جریان تبدیل می‌کنند. ولتاژ تهیه شده توسط سنسور را می‌توان به یک ADC اعمال کرده و مقدار دیجیتال آن را تولید کنیم و این مقدار دیجیتال را با محاسبات ریاضی به عددی دسیمال جهت نمایش تبدیل کنیم.

ADCهای موجود در میکروکنترلرهای AVR، ۱۰ بیتی بوده و بنابراین می‌توانند اعداد بین ۰ تا ۱۰۲۳ را در خود ذخیره کنند. مثلا برای صفر ولت عدد صفر، برای پنج ولت عدد ۱۰۲۳ و برای 2.5 ولت عدد ۵۱۱ را در خود ذخیره می‌کنند. برای کار با این قابلیت باید ابتدا مثل سایر امکانات پیکربندی آن را انجام داد.

مبدل آنالوگ به دیجیتال جهت پیکربندی دارای رجیسترهایی به شرح زیر است:

رجیستر ADMUX:

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

به دلیل اینکه در اکثر مقالات و کتاب های تک تک رجیسترها توضیح داده شده در اینجا از معرفی تک تک بیت ها خودداری می شود.

توسط رجیستر بالا می توان کانال مورد نظر را انتخاب نمود و همچنین ولتاژ رفرنس مبدل را انتخاب کرد. بیت ADLAR نیز چپ چین یا راست چین بودن حالت ذخیره اطلاعات در مد ۱۰ بیت را مشخص می کند.

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

رجیستر ADCSRA:

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

رجیستر SFIOR:

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

در اینجا ما با استفاده از کدویزاد کامپایلر رجیسترها را مقداردهی می کنیم اما شناختن رجیسترهای مربوط به هر بخش نباید فراموش شود.

نکته : پایه AVcc تغذیه قسمت ADC می‌باشد و دلیل جدا بودن آن از تغذیه اصلی خود IC جلوگیری از تاثیر نویز بر روی ADC می‌باشد. دقت کنید اختلاف ولتاژ این دو پایه نباید بیشتر از ±0.3v باشد.

نکته: وقتی که از ADCها استفاده می‌شود دیگر نمی‌توان از پورت‌ها به عنوان I/O استفاده کرد.

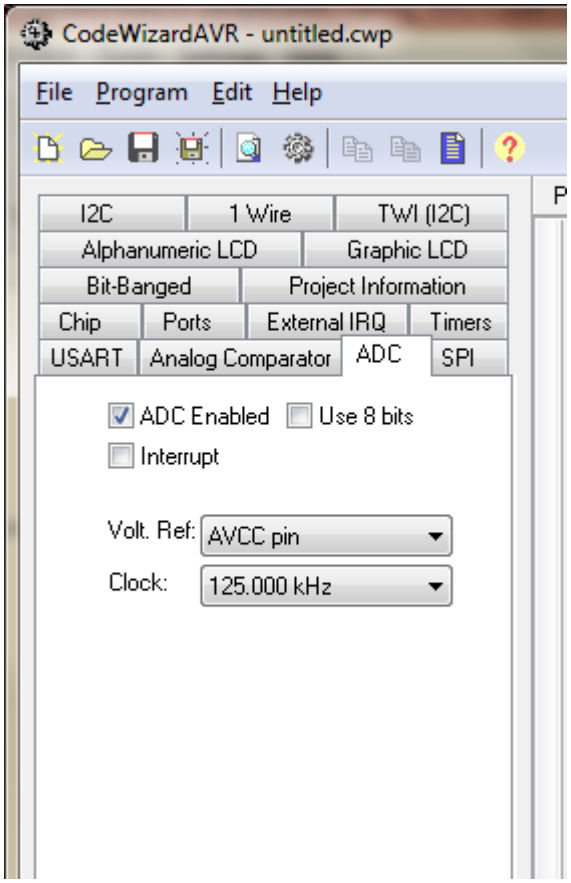
مدارات عملی :

برنامه‌ای بنویسید که توسط آن مقادیر آنالوگ ورودی به اعداد دیجیتال تبدیل شده و روی LCD نمایش داده شود؟

تنظیمات روی برد:

جامپرهای تغذیه و نمایشگر کاراکتری را متصل کرده همچنین جامپر AIN0/ADC را در حالت ADC قرار دهید.

تصویری از چگونگی تنظیمات ADC در کدویزاد:




```
 #include <mega32a.h>
#include <delay.h>
#include <stdio.h>
#include <alcd.h>

#define ADC_VREF_TYPE 0x40

// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
// Delay needed for the stabilization of the ADC input voltage
delay_us(10);
// Start the AD conversion
ADCSRA|=0x40;
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}
```

```
// Declare your global variables here

void main(void)
{

int i;
char buffer[5];
PORTA=0x00;
DDRA=0x00;
// ADC initialization
// ADC Clock frequency: 250.000 kHz
// ADC Voltage Reference: AVCC pin
ADMUX=ADC_VREF_TYPE & 0xff;
ADCSRA=0x85;
// Alphanumeric LCD initialization
// Connections are 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);
while (1)
      {
       i=read_adc(3);
       sprintf(buffer,"%d",i);
       lcd_clear();
       lcd_puts(buffer);
       delay_ms(250);
      }
}
```

همانطور که در مثال بالا ملاحظه شد این اعداد بین ۰ تا ۱۰۲۳ هستند. برای اینکه آن‌ها را قابل فهم کنیم باید از محاسبات ریاضی استفاده کنیم. به مثال زیر دقت کنید.

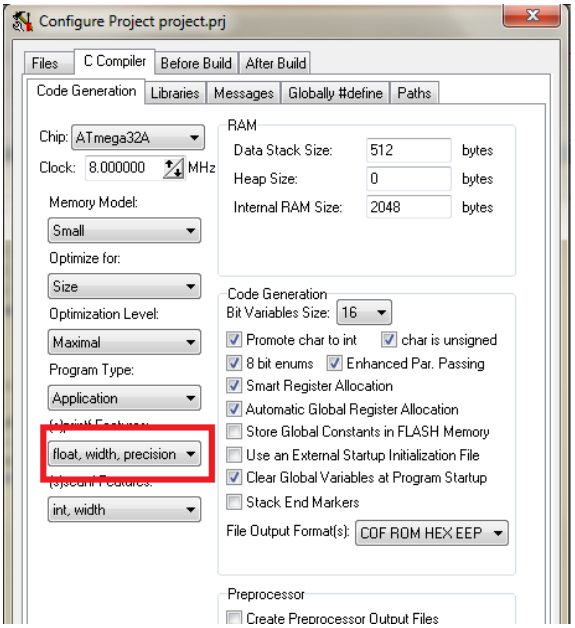
مثال : برنامه‌ی یک ولت متر ۰ تا ۵ ولت را بنویسید؟

```
#include <mega32a.h>
```

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

#define ADC_VREF_TYPE 0x40

// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
// Delay needed for the stabilization of the ADC input voltage
delay_us(10);
// Start the AD conversion
ADCSRA|=0x40;
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}

// Declare your global variables here

void main(void)
{

float i;
char buffer[5];
PORTA=0x00;
DDRA=0x00;
// ADC initialization
// ADC Clock frequency: 250.000 kHz
// ADC Voltage Reference: AVCC pin
ADMUX=ADC_VREF_TYPE & 0xff;
ADCSRA=0x85;
// Alphanumeric LCD initialization
// Connections are 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);
while (1)
      {
       i=read_adc(3);
       i=i/204.6;
       sprintf(buffer,"%3.3f V",i);
       lcd_clear();
       lcd_puts(buffer);
       delay_ms(250);
      }
}
```

دقت کنید در این قسمت از حالت اعشاری تابع sprint استفاده شده است که قبل از کامپایل باید از قسمت تنظیمات تابع آن را روی float قرار دهید.

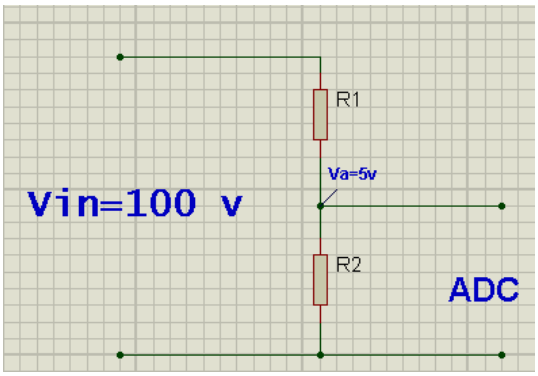


همانطور که در بالا گفته شد ماکزیمم ولتاژ اعمالی به ADCها باید ۵ ولت باشد، خوب حالا اگر بخواهیم یک ولت متر ۰ تا ۱۰۰ ولت بسازیم باید چکار کنیم؟

در این موارد باید ولتاژ اعمالی تا ۵ ولت کاهش پیدا کند که یکی از راه‌های کاهش ولتاژ، استفاده از مدارات مقاومتی است. به مثال زیر که نحوه ساخت یک ولت متر ۰ تا ۱۰۰ و فرمولات مربوطه را نمایش می‌دهد دقت کنید.

مثال : برنامه یک ولت متر ۰ تا ۱۰۰ را بنویسید؟

برای این کار ابتدا مدار مقاومتی کاهش ولتاژ ۱۰۰ را به ۵ ولت طراحی می‌کنیم:



$$R_1 = \frac{R2(Vin-Va)}{Va}$$

$$R_2 = \frac{R1*Va}{(Vin-Va)}$$

در این فرمول‌ها شما مقدار یکی از مقاومت‌ها را قید کرده و مقاومت دیگر با توجه به دو فرمول بالا بدست می‌آید. در این دو فرمول مقدار Va ثابت و برابر ۵ ولت و همچنین Vin را با توجه به نیاز خود تعیین می‌کنید.

برای طراحی این مدار مقدار R_2 را برابر ۱۰ کیلواهم انتخاب کرده و با توجه به آن و قید در فرمول R_1، مقدار R_1 برابر ۱۹۰ کیلواهم بدست خواهد آمد که مقاومت استانداردی نیست و باید با سری و موازی کردن مقاومت‌های استاندارد، آن را بدست آورد.

تنها چیزی که باقی مانده است تبدیل ۱۰۲۳ گرفته شده توسط ADC و تبدیل آن به ۱۰۰ ولت به جای ۵ ولت است که برای این کار از فرمول زیر استفاده می‌شود:

$$X = \frac{1023}{Vin}$$

که X در اینجا می‌شود: ۱۰.۲۳

حالا با توجه به مطالب بالا برنامه مدار را می‌نویسیم:

```
#include <mega32a.h>
#include <delay.h>
#include <stdio.h>
#include <alcd.h>
#define ADC_VREF_TYPE 0x40
// Read the AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
// Delay needed for the stabilization of the ADC input voltage
delay_us(10);
// Start the AD conversion
ADCSRA|=0x40;
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}
void main(void)
{
float i;
```

```
char buffer[5];
PORTA=0x00;
DDRA=0x00;
// ADC initialization
// ADC Clock frequency: 250.000 kHz
// ADC Voltage Reference: AVCC pin
ADMUX=ADC_VREF_TYPE & 0xff;
ADCSRA=0x85;
// Alphanumeric LCD initialization
// Connections are 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);
while (1)
    {
     i=read_adc(3);
     i=i/10.23;
```

```
        sprintf(buffer,"%3.3f V",i);
        lcd_clear();
        lcd_puts(buffer);
        delay_ms(250);
      }
}
```

نکات مهم:

- باید ماکزیمم ولتاژ اعمالی به ADC، ۵ ولت باشد در غیر این صورت پین مورد نظر خواهد سوخت.
- در مداراتی که احتمال رسیدن ولتاژ بالا به ADC انتظار می‌رود، باید از مدارات محافظ استفاده شود. علل خصوص در مدار آمپرمتر، طراحی مدارات محافظ خیلی مهم است.
- برای طراحی مداراتی که ولتاژ آن‌ها کمتر از ۱ ولت است می‌توان با استفاده از تقویت کننده‌های Op-Amp ولتاژ مورد نظر را تقویت کرد و به ۵ ولت یا بالاتر که کار با آن‌ها راحتر است تبدیل کرد.
- برای اندازه گیری ولتاژهای منفی ابتدا باید آن را توسط مدارات Op-Amp به ولتاژ مثبت تبدیل کرده، سپس آن را به ADC اعمال کرد، چون ADC فقط ولتاژ بین ۰ تا ۵ ولت را می‌شناسد.

وقفه‌ها

وقفه چیست؟ گاهی اوقات نیاز است که میکرو همزمان دو عمل را انجام دهد مثلا هم اطلاعاتی را در حافظه ثبت کند و هم تعداد پالس‌های اعمال شده به یک پایه را بشمارد. در عمل هیچ پردازشگری در آن واحد نمی‌تواند دو عمل را انجام دهد. برای این منظور از وقفه‌ها استفاده می‌شود.

در میکروکنترلرهای AVR دو دسته وقفه طراحی شده است:

وقفه های داخلی و وقفه های خارجی.

وقفه‌های داخلی

برای اکثر امکانات و خصوصیات یک میکروکنترلر طراحی شده و برای هر یک بیت، پرچمی که به آن بیت وقفه گفته می‌شود تعبیه شده است، تا یک شدن آن بیت، نشان دهنده وقوع وقفه برای خصوصیت مورد نظر باشد.

وقفه‌های خارجی

تعدادی از پایه‌های میکروکنترلرهای AVR را می‌توان به عنوان یک عمل دهنده وقفه پیکربندی نمود. این پایه‌ها در هر میکروکنترلر با کلمه INTx مشخص شده است که در آن x، عدد وقفه خارجی را نشان می‌دهد.

وقفه ها نیز مثل سایر امکانات میکروکنترلر جهت استفاده باید پیکربندی شوند. قبل از هر چیز متذکر می شویم که باید وقفه سراسری را در ابتدا فعال کنید. این کار توسط دستور اسمبلی زیر صورت می پذیرد:

```
#asm("sei")
```

همچنین توسط دستور#asm("cli") می توان وقفه سراسری را غیرفعال کرد.

رجیسترهای پیکربندی وقفه ها:

رجیستر GICR:

Bit	7	6	5	4	3	2	1	0	
	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
Read/Write	R/W	R/W	R/W	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

با استفاده از رجیستر بالا انتخاب می کنیم که از کدام یک از وقفه ها استفاده کنیم.

رجیستر MCUCR:

Bit	7	6	5	4	3	2	1	0	
	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

با استفاده از رجیستر بالا حساسیت وقفه تعیین می شود. به جدول زیر دقت کنید.

Table 35. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

رجیستر MCUCR برای دو اینتراپت صفر و یک می باشد و برای استفاده از وقفه خارجی ۲ باید رجیستر MCUCSR را مقداردهی کنید.

رجیستر MCUCSR:

Bit	7	6	5	4	3	2	1	0	
	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	MCUCSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	See Bit Description					

همانطور که از رجیستر بالا مشخص است وقفه خارجی ۲ فقط به دو لبه بالارونده و پایین رونده حساس می باشد.

اگر دقت کرده باشید با رخ دادن وقفه گفتیم که CPU برنامه اصلی را رها کرده و به اجرای برنامه تعیید شده در روال وقفه می پردازد. پس ما باید یک تابع جهت نوشتن برنامه وقفه م و تشخیص اینکه کدام وقفه رخ داده است داشته باشیم.

کامپایلر کدویژن به صورت اتوماتیک زیرروال را ایجاد کرده و ما فقط باید برنامه مورد نظر را در جایگاه مناسب قرار دهیم.

مقایسه کننده آنالوگ

یکی دیگر از امکانات جالب 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 و در صورتی که ولتاژ پایه AIN0 از AIN1 بیشتر باشد، خروجی مقایسه کننده ACO یک می شود.

سوالی که شاید به ذهن برخی از افراد خطور کنه 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.

رجیسترهای مربوط به واحد مقایسه کننده آنالوگ

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

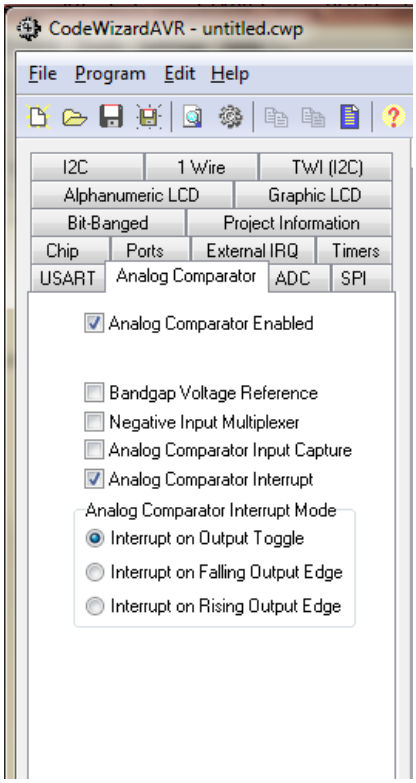
مدارات عملی :

برنامه ای بنویسید که با استفاده از واحد مقایسه کننده آنالوگ دو ولتاژ آنالوگ را با هم مقایسه کرده و در صورت افزایش ولتاژ پایه مثبت، یک LED از پورت A را روشن کند؟

تنضیمات روی برد:

جامپر تغذیه را بر روی ۵ ولت قرار داده و همچنین جامپر ADC/AIN0 را در حالت AIN0 قرار دهید و در نهایت جامپر AIN1 را جهت اعمال ولتاژ ۳.۳ ولت به عنوان رفرنس متصل کنید.

تنظیمات مربوطه در کدویزارد:




```
#include <mega32a.h>

// Analog Comparator interrupt service routine

interrupt [ANA_COMP] void ana_comp_isr(void)

{

// Place your code here

}

void main(void)

{
```

```
PORTA=0xff;
DDRA=0xff;
PORTB=0x0;
DDRB=0xf3;

// Analog Comparator initialization
// Analog Comparator: On
// Interrupt on Output Toggle
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x08;
SFIOR=0x00;
// Global enable interrupts
#asm("sei")

while (1)
    {
     if(ACSR.5==1) PORTA.0=0;
     else PORTA.0=1;
    }
}
```

تایمر کانتر صفر

یکی از بهترین امکانات میکروکنترلرهای AVR وجود تایمرها می باشد. شاید این سوال در ذهن ایجاد شود که آیا خود میکروکنترلر قادر به شمارش نیست ؟

در این صورت چه نیازی به وجود یک تایمر دیگر هست؟

در جواب این دو سوال باید گفت بله خود میکروکنترلر قدرت شمارش را دارد اما با این کار وقت CPU به شمارش تلف می شود و برای جلوگیری از به هدر نرفتن وقت CPU یک سری ماژول بر روی قطعه پیاده سازی کرده اند.

در میکروکنترلر AVR تایمرها به دو صورت ۸ و ۱۶ بیت موجود می باشند. تایمر صفر که در این قسمت توضیح داده می شود یک تایمر ۸ بیتی می باشد.

تایمر کانتر هم مثل همه ی امکانات دیگر نیاز به پیکربندی دارد در نتیجه به شرح رجیسترهای تایمر کانتر صفر می پردازیم:

رجیستر TCCR0:

Bit	7	6	5	4	3	2	1	0	
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	TCCR0
Read/Write	W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

توسط بیت های cs00 تا cs02 تنظیمات مربوط به روشن و خاموش کردن تایمر کانتر و تنظیم کلاک در حالت تایمر و لبه پالس در حالت کانتر می باشد.

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

بیت های WGM00 و WGM01 وظیفه تعیین مد عملکرد تایمر کانتر را بر عهده دارند.

Mode	WGM01 (CTC0)	WGM00 (PWM0)	Timer/Counter Mode of Operation	TOP	Update of OCR0	TOV0 Flag Set-on
0	0	0	Normal	0xFF	Immediate	MAX
1	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	1	0	CTC	OCR0	Immediate	MAX
3	1	1	Fast PWM	0xFF	TOP	MAX

و در نهایت بیت های COM00 و COM01 وضعیت پین OC0 را در حالت های مختلف مشخص می کنند.

جهت آشنایی بیشتر یک مثال عملی را انجام می دهیم.

۱. می خواهیم برنامه ای بنویسیم که هر یک ثانیه مقدار یک متغیر افزایش پیدا کرده و تا ۹۹ بشمارد و این عدد بر روی سون سگمنت ها نمایش داده شود.

فرکانس کلاک را اگر ۱مگاهرتز بگیریم و تقسیم کننده فرکانسی تایمر را بر روی ۱۰۲۴ قرار دهیم داریم:

$\frac{1000000}{1024}$ = ۹۷۶.۵۶ hz

که ما با گرد کردن آن را ۱ کیلوهرتز می گیریم.

به این ترتیب زمان تناوب تایمر ۱میلی ثانیه می شود. با توجه به این معلومات اگر تایمر ۴ بار از صفر تا ۲۵۰ را شمارش کند تقریبا می توانیم به زمان یک ثانیه دست پیدا کنیم.

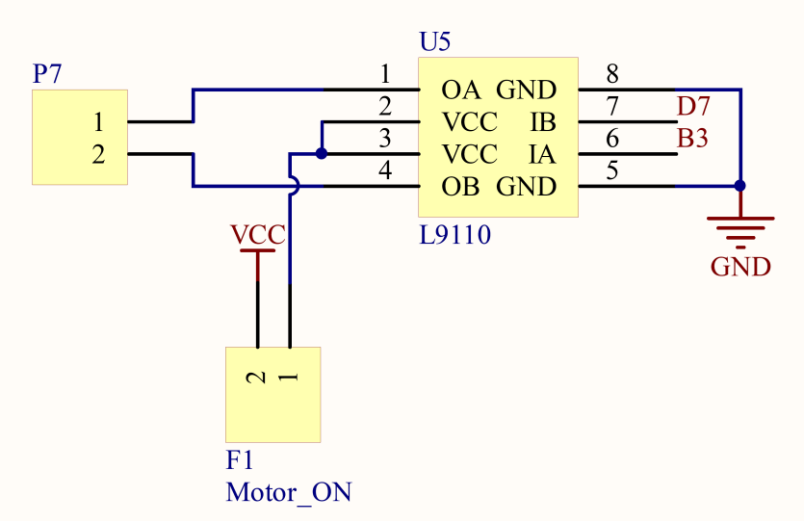
```
#include <mega32a.h>
#include <delay.h>
int x,count,i,j;
char segment[10]={0xf5,0x50,0xb3,0xf2,0x56,0xe6,0xC7,0x70,0xF7,0x76};
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
x++;
if(x>3)
```

```
{
   x=0;
   count++;
   if(count>99) count=0;
}
TCNT0=6;
}

// Declare your global variables here

void main(void)
{

PORTA=0x00;
DDRA=0xFF;
PORTB=0x00;
DDRB=0x03;
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 0.977 kHz
// Mode: Normal top=0xFF
// OC0 output: Disconnected
TCCR0=0x05;
TCNT0=6;
```

```
OCR0=0x00;
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x01;
// Global enable interrupts
#asm("sei")
while (1)
    {
     i=count/10;
     j=count-(i*10);
     PORTA=segment[j];
     PORTB=0x01;
     delay_ms(10);
     PORTB=0x00;
     PORTA=segment[i];
     PORTB=0x02;
     delay_ms(10);
     PORTB=0x00;
    }
}
```

۲. برنامه ای بنویسید که با استفاده از PWM سرعت یک موتور را هر ۲۰۰ میلی ثانیه افزایش دهد؟

شماتیک مدار:



تنظیمات اعمال شده روی برد آموزشی:

جامپر تغذیه و Motor را متصل کنید.

```
#include <mega32a.h>
#include <delay.h>
void main(void)
{

PORTB=0x00;
DDRB=0x08;
PORTD=0x00;
DDRD=0x80;
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 8000.000 kHz
// Mode: Fast PWM top=0xFF
// OC0 output: Non-Inverted PWM
```

```
TCCR0=0x69;
TCNT0=0x00;
OCR0=0x00;
while (1)
    {
     OCR0+=15;
     delay_ms(200);
    }
}
```

راه اندازی LCDهای گرافیکی سری SED

LCDهای کاراکتری محدودیت‌های زیادی دارند، مثلا نمی‌توان کلمات فارسی را به صورت پیوسته دید یا از افکت استفاده کرد، ولی در این LCDها می‌توان به هر پیکسل که نیاز است دسترسی داشت و به همین دلیل محدودیتی برای نمایش بر روی آن وجود ندارد. LCDهای گرافیکی مانند LCDهای کاراکتری در سایزهای مختلف موجود می‌باشند.

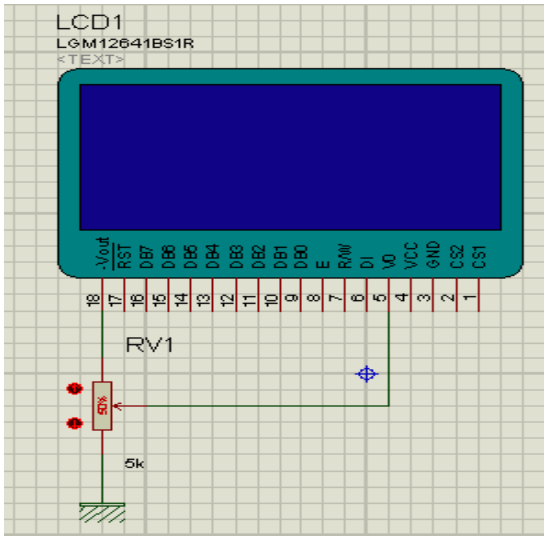
این LCD دارای دو نیم صفحه چپ و راست می‌باشد. نحوه سیم بندی در این LCDها با توجه به شرکت سازنده متفاوت بوده و بهتر است قبل از راه اندازی آن دیتاشیت LCD را مشاهده کنید.

شرح پایه‌های یک نمونه موجود در بازار:

شماره پایه	Symbol	توضیحات
۱	GND	پایه تغذیه
۲	Vcc	پایه تغذیه
۳	Vo	تنظیم کنتراست
۴	RS	ورودی دیتا و دستورالعمل
۵	RW	خواندن و نوشتن دیتا

۶	E	فعال سازی ورودی دیتا
۷ – ۱۴	Db0 To Db7	پایه های دو جهته برای خواندن یا نوشتن داده در LCD
۱۵	CS1	فعال ساز چیپ ۱
۱۶	CS2	فعال ساز چیپ ۲
۱۷	RST	بازنشانی یا ریست کردن LCD
۱۸	VEE	تولید ولتاژ منفی جهت تنظیم کنتراست
۱۹	A	آند LED پس زمینه
۲۰	K	کاتد LED پس زمینه

نحوه اتصال پایه‌ها برای تنظیم کنتراست:



در پیکربندی LCDهای گرافیکی باید به چهار نکته دقت کرد:

- معرفی فونت و کتابخانه لازم جهت کار با LCD
- تعیین سایز LCD
- Dataport: تعیین پورتی از میکروکنترلر برای اتصال به پایه‌های Db0 الی Db7

Controlport: تعیین پورتی از میکروکنترلر برای اتصال به پایه‌های کنترلی میکروکنترلر شامل: CS1, CS2, E, RST, RS, R/W.

راه اندازی نمایشگر در کدویژن:

در ورژن های جدید کامپایلر کدویژن کتابخانه ای جهت راه اندازی این نمایشگرها تعبیه شده که در اینجا برخی از توابع مهم آن را توضیح داده و سپس چند پروژه عملی با آن انجام خواهیم داد.

توابع موجود در هدر graphics.h:

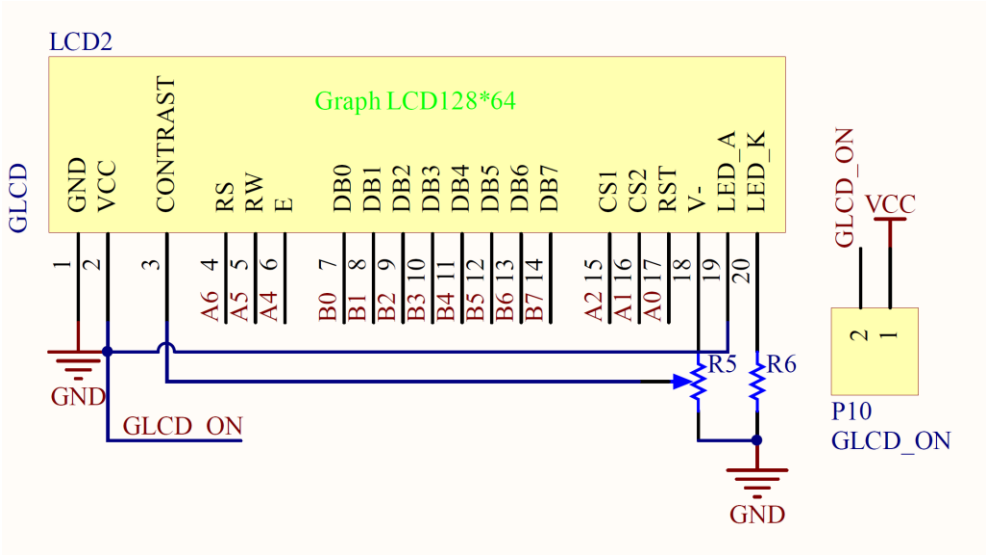
عملکرد تابع	فرم کلی تابع
پاک کردن صفحه نمایش	void glcd_clear(void);
پاک کردن اشکال گرافیکی از صفحه نمایش	void glcd_cleargraphics(void);
نمایش کاراکتر بر روی نمایشگر	void glcd_putchar(char c);
نمایش کاراکتر در مختصات مورد نظر	void glcd_putcharxy(GLCDX_t x, GLCDY_t y, char c);
نمایش رشته موجود در حافظه sram	void glcd_outtext(char *str);
نمایش رشته موجود در حافظه flash	void glcd_outtextf(flash char *str);
نمایش رشته موجود در حافظه sram در مختصات مورد نظر	void glcd_outtextxy(GLCDX_t x, GLCDY_t y, char *str);
رسم خط	void glcd_line(GLCDX_t x0, GLCDY_t y0, GLCDX_t x1, GLCDY_t y1);
رسم مستطیل	void glcd_rectangle(GLCDX_t left, GLCDY_t top, GLCDX_t right, GLCDY_t bottom);
رسم دایره	void glcd_circle(GLCDX_t x, GLCDY_t y, GLCDX_t radius);
نمایش عکس ذخیره شده در حافظه flash	unsigned long glcd_putimagef(GLCDX_t left, GLCDY_t top, flash unsigned char *pimg, GLCDBLOCKMODE_t mode);

جهت آشنایی با چگونگی عملکرد دستورات و توابع چند مثال عملی را ذکر می کنیم.

مدارات عملی :

۱- برنامه ای بنویسید که بتواند عبارت " www.ECA.ir " را بر روی LCD چاپ کند؟

شماتیک مربوط به اتصال نمایشگر گرافیکی:



تنظیمات روی برد:

جامپر تغذیه را در حالت ۵ ولت قرار داده و جامپر GLCD را نیز وصل کنید.

```
#include <mega32a.h>
#include <delay.h>
#include <glcd.h>
#include <font5x7.h>
void main(void)
{
// Graphic LCD initialization data
GLCDINIT_t glcd_init_data;
DDRA=0xff;
DDRB=0xff;
// Graphic LCD initialization
// The KS0108 connections are specified in the
```

```
// Project|Configure|C Compiler|Libraries|Graphic LCD menu:
// DB0 - PORTB Bit 0
// DB1 - PORTB Bit 1
// DB2 - PORTB Bit 2
// DB3 - PORTB Bit 3
// DB4 - PORTB Bit 4
// DB5 - PORTB Bit 5
// DB6 - PORTB Bit 6
// DB7 - PORTB Bit 7
// E - PORTA Bit 4
// RD /WR - PORTA Bit 5
// RS - PORTA Bit 6
// /RST - PORTA Bit 0
// /CS1 - PORTA Bit 2
// /CS2 - PORTA Bit 1
// Specify the current font for displaying text
glcd_init_data.font=font5x7;
// No function is used for reading
// image data from external memory
glcd_init_data.readxmem=NULL;
// No function is used for writing
// image data to external memory
glcd_init_data.writexmem=NULL;
glcd_init(&glcd_init_data);
```

```
glcd_cleargraphics();
glcd_outtextxy(45,25,"www.ECA.ir");
while (1);
}
```

۲- برنامه‌ای بنویسید که چهار دایره متحد المرکز را رسم کند؟

```
#include <mega32a.h>
#include <delay.h>
#include <glcd.h>
#include <font5x7.h>
void main(void)
{
// Graphic LCD initialization data
GLCDINIT_t glcd_init_data;
DDRA=0xff;
DDRB=0xff;
// Graphic LCD initialization
// The KS0108 connections are specified in the
// Project|Configure|C Compiler|Libraries|Graphic LCD menu:
// DB0 - PORTB Bit 0
// DB1 - PORTB Bit 1
// DB2 - PORTB Bit 2
// DB3 - PORTB Bit 3
// DB4 - PORTB Bit 4
// DB5 - PORTB Bit 5
```

```
// DB6 - PORTB Bit 6
// DB7 - PORTB Bit 7
// E - PORTA Bit 4
// RD /WR - PORTA Bit 5
// RS - PORTA Bit 6
// /RST - PORTA Bit 0
// /CS1 - PORTA Bit 2
// /CS2 - PORTA Bit 1
// Specify the current font for displaying text
glcd_init_data.font=font5x7;
// No function is used for reading
// image data from external memory
glcd_init_data.readxmem=NULL;
// No function is used for writing
// image data to external memory
glcd_init_data.writexmem=NULL;
glcd_init(&glcd_init_data);
glcd_cleargraphics();
glcd_circle(64,32,10);
delay_ms(500);
glcd_circle(64,32,15);
delay_ms(500);
glcd_circle(64,32,20);
delay_ms(500);
```

```
glcd_circle(64,32,25);

delay_ms(500);

while (1);

}
```

۳- برنامه‌ای بنویسید که بتوان یک عکس ساده را روی LCD گرافیکی نمایش دهد؟

جهت نمایش عکس ابتدا باید تصویر ذخیره شده با فرمت bmp را توسط نرم افزارهای مربوطه به کد هگز تبدیل کنید و در نهایت کد تولید شده را توسط تابع نمایش عکس به نمایشگر ارسال کنید.

جهت تبدیل کد تصاویر نرم افزاری در CD همراه قرار داده شده است.

```
#include <mega32a.h>
#include <delay.h>
#include <glcd.h>
#include <font5x7.h>
flash unsigned char pic[] = {
   128,0,64,0,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
   0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x80, 0x80, 0x40, 0x40, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x28, 0x28,
0x04, 0x04, 0x04, 0x04, 0x0C, 0xCC, 0xF8, 0xF8,
0x1E, 0x16, 0x32, 0x21, 0x21, 0x23, 0x23, 0x21,
0x20, 0x10, 0x10, 0x38, 0x00, 0x00, 0x30, 0x38,
0x38, 0xE8, 0x0C, 0x04, 0x04, 0x84, 0xC4, 0xE4,
0x34, 0x14, 0x0C, 0x0C, 0x08, 0x0C, 0x0C, 0x0A,
0x0A, 0x0A, 0x0A, 0x0A, 0x0A, 0x0A, 0x08, 0x04,
0x00, 0x00, 0x00, 0x00, 0x00, 0x12, 0x11, 0x11,
0x11, 0x19, 0x09, 0x0D, 0x0D, 0x06, 0x02, 0x03,
0x03, 0x02, 0x82, 0x84, 0x44, 0x24, 0x24, 0x94,
0xD4, 0x74, 0x3C, 0x1C, 0x06, 0x06, 0x06, 0x02,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x08, 0x0C, 0x84, 0xE4, 0xFE, 0xFF, 0x1F, 0x0D,
0x08, 0x08, 0x18, 0x1C, 0x3C, 0x74, 0x0C, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x6C, 0xFF, 0xFF, 0xF7, 0x00,

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0,
0xC0, 0xC0, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00,
0x00, 0x80, 0x80, 0x40, 0xE0, 0xE0, 0x00, 0x00,
0x00, 0x70, 0x50, 0x18, 0x88, 0x68, 0x18, 0x0C,
0x0E, 0x1B, 0x11, 0xD0, 0xF0, 0xF8, 0xBC, 0x3F,
0x13, 0x10, 0x10, 0x1C, 0x0C, 0x08, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0xE0, 0xA0, 0x90, 0xD0, 0xD0, 0xD0,
0xF0, 0x7C, 0x3F, 0x0F, 0x1B, 0x38, 0x30, 0x70,
0x70, 0xE0, 0xE0, 0x80, 0x80, 0x00, 0x80, 0x80,
0x40, 0x60, 0x20, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x07, 0x0F, 0x1F, 0x1E,
0x38, 0x70, 0x70, 0x60, 0xC0, 0xC0, 0xC0, 0xC0,
0xE1, 0x60, 0x18, 0x0F, 0x02, 0x00, 0x00, 0x3C,
0x46, 0x81, 0x80, 0xC0, 0x43, 0x67, 0x30, 0x38,
0x18, 0x0C, 0x02, 0x01, 0x00, 0x00, 0x00, 0x00,
0x00, 0x70, 0xFC, 0xFF, 0xFF, 0x87, 0xC0, 0xC0,
0x40, 0x60, 0x30, 0x18, 0x18, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x01, 0x01, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,

```
0x00, 0x00, 0x00, 0x01, 0x01, 0x01, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
void main(void)
{
// Graphic LCD initialization data
GLCDINIT_t glcd_init_data;
DDRA=0xff;
DDRB=0xff;
// Graphic LCD initialization
// The KS0108 connections are specified in the
// Project|Configure|C Compiler|Libraries|Graphic LCD menu:
// DB0 - PORTB Bit 0
// DB1 - PORTB Bit 1
// DB2 - PORTB Bit 2
// DB3 - PORTB Bit 3
// DB4 - PORTB Bit 4
// DB5 - PORTB Bit 5
// DB6 - PORTB Bit 6
// DB7 - PORTB Bit 7
// E - PORTA Bit 4
// RD /WR - PORTA Bit 5
// RS - PORTA Bit 6
// /RST - PORTA Bit 0
// /CS1 - PORTA Bit 2
// /CS2 - PORTA Bit 1
// Specify the current font for displaying text
glcd_init_data.font=font5x7;
// No function is used for reading
// image data from external memory
glcd_init_data.readxmem=NULL;
// No function is used for writing
// image data to external memory
glcd_init_data.writexmem=NULL;
glcd_init(&glcd_init_data);
glcd_cleargraphics();
glcd_putimagef(0,0,pic,GLCD_PUTCOPY);
while (1);
```

}

نکات :

- در موقع کار با LCD ها اگر تصویر شما از وسط دو نیم شده، نمایش داده شود یا ۱۸۰ درجه اختلاف داشته باشد در این صورت پایه‌های CS1 و CS2 را معکوس متصل کرده‌اید.

شروع کار با Step Motor

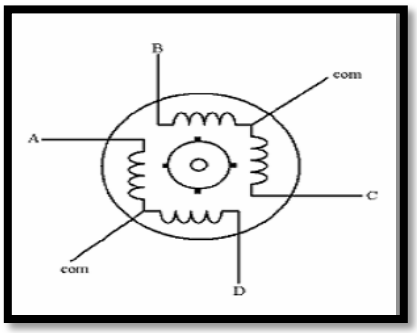
موتورهای پله‌ای وسایل الکترومکانیکی هستند که پالس‌های دیجیتالی را به یک جابجایی یا چرخش معین تبدیل می‌کنند. در کاربردهایی مانند راه اندازی دیسک سخت، چاپگرهای مغناطیسی، رباتیک و کنترل دقیق ماشین ابزارها، از موتورهای پله ای استفاده می‌شود.

انواع موتورهای پله‌ای

موتورهای پله‌ای در دو نوع پنج سیمه و شش سیمه وجود دارند که متداول‌ترین موتورهای پله‌ای شش سیمه می‌باشند.
این موتورها به موتورهای پله‌ای چهار فاز یا چهار قطبی نیز معروف هستند.
در این موتورها ۴ سیم پیچ استاتور وجد دارد که دو به دو با سیم سر وسط جفت شده‌اند.
سر یا سرهای وسط با توجه به برنامه راه اندازی موتور به Vcc یا GND وصل می‌شوند.

تشخیص پایه‌های موتور پله‌ای

با استفاده از یک اهم متر می‌توان پایه‌های موتور را تشخیص داد، به این استدلال که پایه‌های مشترک هیچ ارتباطی با هم ندارند و اندازه مقاومت الکتریکی بین هر دو سیم پیچ جفت شده، دو برابر مقاومت هر سیم پیچ نسبت به سر وسط است.



زاویه پله موتور

زاویه پله یکی از مشخصه‌های مهم موتورهای پله‌ای است که نشان می‌دهد موتور به ازای هر پله چند درجه می‌چرخد. زاویه پله در موتورهای مختلف متفاوت می‌باشد.

تعداد پله‌های یک دور کامل در هر موتور پله‌ای از رابطه زیر محاسبه می‌شود:

زاویه پله / ۳۶۰ = تعداد پله در دور

راه اندازی موتور پله‌ای

چرخش در موتورهای پله‌ای براساس جذب قطب غیرهمنام استوار است. درهر مرحله با دادن پالس الکتریکی به یکی از سیم پیچ‌های استاتور، شفت موتور به اندازه یک پله می‌چرخد تا قطب‌های غیر همنام روتور و استاتور دریک راستا قرار گیرند. بنابراین برای چرخش شفت موتور باید به صورت متوالی در هر مرحله به سیم پیچ‌های مناسب پالس الکتریکی اعمال کرد.

راه اندازی موتورهای پله‌ای به دو روش صورت می‌گیرد:

- ✓ تحریک پله کامل
- ✓ تحریک نیم پله

تحریک پله کامل

تحریک پله کامل به دو روش صورت می‌گیرد:

روش اول: تحریک یک سیم پیچ در هر مرحله

در این روش در هر مرحله و به ترتیب پالسی را برای یکی از سیم پیچ‌های D,C,B,A می‌فرستیم، تکرار این روند به صورت متوالی باعث چرخش مداوم محور موتور (روتور) خواهد شد. درصورت اجرای فرامین جدول زیر از بالا به پایین، چرخش روتور در جهت عقربه‌های ساعت و در صورت اجرای فرامین جدول از پایین به بالا، چرخش روتور در جهت خلاف عقربه‌های ساعت خواهد بود.

کد هگز	سیم پیچ D	سیم پیچ C	سیم پیچ B	سیم پیچ A	شماره پله
۰۸	۰	۰	۰	۱	۱
۰۴	۰	۰	۱	۰	۲
۰۲	۰	۱	۰	۰	۳

	۰۱	۱	۰	۰	۰	۴	

روش دوم: تحریک دو سیم پیچ در هرمرحله

در این روش در هر مرحله و به طور همزمان دو سیم پیچ تحریک می‌شوند. زاویه طی شده در این روش با روش قبل یکسان بوده و تنها تفاوت در موقعیت توقف روتور خواهد بود. دراین حالت روتور بین دو قطب تحریک شده استاتور توقف می‌کند.

فرامین لازم جهت این روش راه اندازی در جدول زیر مشاهده می‌شود.

	سیم پیچ D	سیم پیچ C	سیم پیچ B	سیم پیچ A	شماره پله	
	۱	۰	۰	۱	۱	
	۰	۰	۱	۱	۲	
	۰	۱	۱	۰	۳	
	۱	۱	۰	۰	۴	

تحریک نیم پله

برای دست یابی به پله‌های ریزتر ودرنتیجه داشتن دقت بیشتر می‌توان با جریان دهی مناسب به موتور، آن را در زوایای کوچک‌تری نسبت به پله عادی خود موتور، بچرخانیم. بنابراین با ترکیب دوروش قبلی می‌توانیم موتور را در حالت نیم پله تحریک کنیم. به عنوان مثال موتوری که دارای زاویه پله 2 می‌باشد در تحریک پله کامل با طی 180 پله یک دور کامل می‌زند. اما با تحریک نیم پله با طی 360 پله یک دور کامل خواهد زد. فرامین لازم برای تحریک نیم پله در جدول زیر آمده است.

	سیم پیچ D	سیم پیچ C	سیم پیچ B	سیم پیچ A	شماره پله	
	0	0	0	1	۱	
	0	0	1	1	۲	
	0	0	1	0	۳	
	0	1	1	0	۴	
	۰	۱	۰	۰	۵	
	۱	۱	۰	۰	۶	
	۱	۰	۰	۰	۷	
	۱	۰	۰	۱	۸	

نکته: موتورهای پله ای وسایل پرمصرفی (پرتوان) هستند ومیکروکنترلرها مستقیما توان لازم جهت راه اندازی آن‌ها را ندارند. بنابراین باید با استفاده از بافرهای جریان توان لازم جهت راه اندازی موتورپله ای را ایجاد کنیم که بافر جریان می‌تواند ترانزیستورهای دارلینگتون TIP 122 یا ICهای راه اندازی مانند ULN2803 و L298 باشد.

راه اندازی موتور پله ای با L298

IC، L298 در دو مدل تولید می‌شود که در اینجا از مدل Moltiwatt استفاده می‌شود. این قطعه توانایی جریان دهی تا ۴ آمپر و ولتاژ ۴۶ ولت را دارد. تنها عیب این قطعه نبود دیودهای هرزگرد داخل آن است که باید از بیرون به IC متصل شود.

نکته: دیودهای هرزگرد به صورت پک پل دیودی، بر روی برد موجود است و نیازی به اتصال آن‌ها در خارج از برد نیست.

نحوه اتصال پایه های L 298:

وظیفه پایه	نام پایه	Powerso20	Multiwatt 15
- بسته به نوع میکروکنترلر ولتاژی بین ۱.۵ تا ۵۰ ولت به این پایه اعمال می‌شود. - ولتاژ این پایه ۱ ولت از تغذیه موتور باید بیشتر باشد. - برای کاهش نویز بین این پایه و زمین یک خازن 100nf قرار می‌گیرد.	تغذیه موتور	۶	۴
- این پایه تغذیه خود ای سی بوده و در حدود ۷ ولت می‌باشد. - برای کاهش نویز بین این پایه و زمین یک خازن 100nf قرار می‌گیرد.	تغذیه ای سی	۱۲	۹
	زمین ای سی	۱،۱۰،۱۱،۲۰	۸
- با دادن ولتاژی بین ۲.۳ تا ۷ ولت به هر کدام از این پایه‌ها می‌توان	فعال ساز خروجی‌های A	۸، ۱۴	۶ ، ۱۱

موتورهای A و B را فعال کرد. - با زمین کردن این پایه‌ها می‌توان موتورهای A و B را غیر فعال کرد.	و B		
- این دو پایه برای کنترل موتور A از میکروکنترلر گرفته می‌شوند.	ورودی ۱ و ۲	۹ ، ۷	۷ ، ۵
- این دو پایه برای راه اندازی موتور A ، مستقیما به و پایه موتور وصل می‌شوند.	خروجی ۱ و ۲	۵ ، ۴	۳ ، ۲
- این دو پایه برای کنترل موتور B از میکروکنترلر گرفته می‌شوند.	ورودی ۳ و ۴	۱۵ ، ۱۳	۱۲ ، ۱۰
- این دو پایه برای راه اندازی موتور B ، مستقیما به و پایه موتور وصل می‌شوند.	خروجی ۳ و ۴	۱۷ ، ۱۶	۱۴ ، ۱۳
- از این دو پایه به عنوان سنسور جریان استفاده می‌شود. - توسط یک پتانسیومتر به زمین وصل می شوندکه می‌تواند جریان بار را کنترل کند. - در صورتی که کنترل جریان مهم نباشد توسط سیم به زمین وصل می‌شود.	حس کننده A و B	۱۹ ، ۲	۱۵ ، ۱
	بدون اتصال	۱۸ ، ۳	

نکته مهم: زمان تاخیری که بین پالس‌ها است، نباید خیلی کم باشد چون در غیر این صورت موتور نمی‌تواند پالس‌ها را دنبال کند و فقط در جای خود می‌لرزد.

ارتباط سریال SPI

ارتباط سریال SPI یک پروتکل ارتباطی سنکرون با سرعت بالا است که می‌تواند برای ارتباط میکروکنترولرهای AVR با یکدیگر و یا ارتباط میکروکنترولر AVR با وسیله های دیگری که قابلیت این نوع ارتباط را دارا هستند، به کار برده شود. رجیسترهای مربوط به این نوع ارتباط در تمام AVRها یکسان است.

خصوصیات ارتباط سریال SPI

- ارسال و دریافت داده همزمان
- استفاده از چهار سیم برای انتقال اطلاعات
- بیت‌های قابل برنامه ریزی برای تنظیم سرعت انتقال دیتا
- دارای پرچم وقفه اتمام ارسال
- ارتباط به صورت‌های MASTER / SLAVE
- بیدار شدن از حالت بیکاری (IDLE)

شرح عملکرد ارتباط سریال SPI

در ارتباط سریال SPI از چهار پایه SCK، MISO، MOSI، SS استفاده می‌شود. پایه SCK در مد Master به عنوان خروجی کلاک و در مد Slave به عنوان ورودی کلاک مورد استفاده قرار می‌گیرد. با نوشتن رجیستر داده SPI در Master، پردازنده شروع به تولید کلاک SPI کرده و داده‌ها از پایه MOSI خارج شده و به پایه MOSI در Slave وارد می‌شوند. بعد از انتقال کامل داده توسط Master، کلاک SPI قطع شده و پرچم وقفه پایان ارسال داده (SPIF) یک می‌شود و برنامه وقفه اجرا می‌شود. دو شیفت رجیستر ۸ بیتی در Masterو Slave را می‌توان به عنوان یک شیفت رجیستر ۱۶ بیتی در نظر گرفت. به عبارت دیگر زمانی که داده‌ای از Master به Slave ارسال می‌شود، می‌توان در همان حال در جهت مخالف، داده‌ای از Slave به Master ارسال کرد. بنابراین در طول ۸ کلاک SPI، داده‌های Master و Slave با هم عوض می‌شوند.

جهت پایه SS:

جهت پایه‌ی SS (خروجی یا ورودی بودن) درمد Master توسط کاربر تعیین می‌شود:

- اگر پایه SS به صورت ورودی به خروجی تعیین شود از آن به عنوان خروجی عادی استفاده می‌شود به این صورت که هیچ تاثیری در ارتباط SPI ندارد.

- اگر پایه SS به صورت ورودی تعیین شود بایستی High یا ۱ باشد تا عملیات Master با اطمینان انجام شود.

جهت پایه SS (خروجی یا ورودی بودن) درمد Slave توسط کاربر تعیین نمی‌شود:

در این حالت پایه SSهمیشه به عنوان ورودی می‌باشد:

- زمانیکه Lowباشد SPIفعال می‌شود (در این حالت پایه MISO خروجی و بقیه پایه‌ها ورودی هستند)
- زمانیکه High باشد SPI بیکار بوده و تمام پایه‌ها به صورت ورودی می‌باشند.

رجیسترهای ارتباط SPI:

رجیستر SPCR:

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

رجیستر SPSR:

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

رجیستر SPDR:

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

جهت راحتی کار با پروتکل spi کامپایلر کدویژن هدری را با نام spi.h در نظر گرفته که دارای یک تابع به فرم زیر است:

```
unsigned char spi(unsigned char data);
```

این تابع همانطور که از فرم کلی اش مشخص است می تواند دیتایی را دریافت و دیتایی را ارسال کند.

اگر بخواهیم دیتا ارسال کنیم باید به صورت زیر عمل کنیم:

```
Spi(data to send);
```

و در صورتی که بخواهیم دیتایی را دریافت کنیم به فرم زیر عمل می کنیم:

```
data=spi(0);
```

نکته مهم: در صورتی که می خواهید دو میکروکنترلر را با استفاده از این پروتکل به هم متصل کنید دقت کنید که فرکانس هر دو میکرو باید عینا برابر باشد در غیر این صورت اطلاعات به درستی تبادل نخواهد شد.

راه اندازی حافظه های جانبی MMC

خیلی وقت ها پیش میاد که حافظه میکروکنترلر جهت ذخیره داده ها کم میاره. خب احتمالا همگی برای اولین گزینه بریم سراغ حافظه های eeprom خارجی اما متوجه میشیم که اونا هم آنچنان چنگی به دل نمی زنن.

گذشته از این ها خیلی وقتا شاید بخوایم یه آلبوم دیجیتال یا یه waveplayer بسازیم، اون موقع اطلاعات رو کجا ذخیره کنیم؟؟؟

گزینه مناسب برمی گیرده به حافظه های MMC که توانایی حجم زیادی از داده های رو در فضایی خیلی کم دارن.

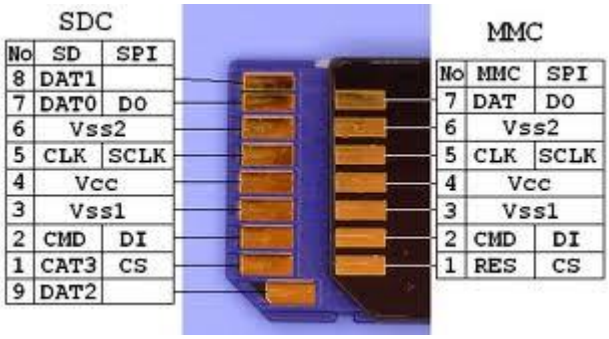
شکل ظاهری این حافظه های به صورت زیر می باشد:

ارتباط با این حافظه ها از ۲ طریق ممکن شده:

۱. پروتکل مخصوص حافظه یا mmc
۲. پروتکل spi

پروتکل انتخابی ما با توجه به سخت افزار موجود در میکروکنترلر spi بوده و می خواهیم با استفاده از این پروتکل با MMC ارتباط برقرار کرده و اطلاعاتی را از آن خوانده یا در آن بنویسیم.

پایه های حافظه MMC به صورت تصویر زیر می باشد:

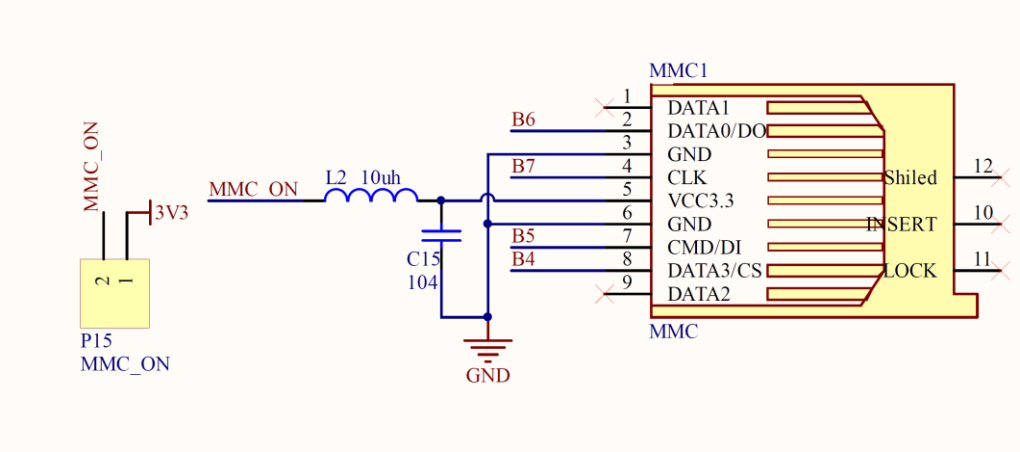


دو روش جهت ارتباط و خواندن و نوشتن اطلاعات در حافظه وجود دارد:

۱- خواندن/نوشتن به صورت سکتور سکتور
۲- خواندن/نوشتن با استفاده قوانین حاکم بر فرمت fat

در کامپایلر کدویژن جهت راه اندازی این حافظه ها کتابخانه ای در نظر گرفته شده که در این قسمت می خواهیم با استفاده از کتابخانه موجود و توابع آن کار بر روی MMC SD را آغاز کنیم.

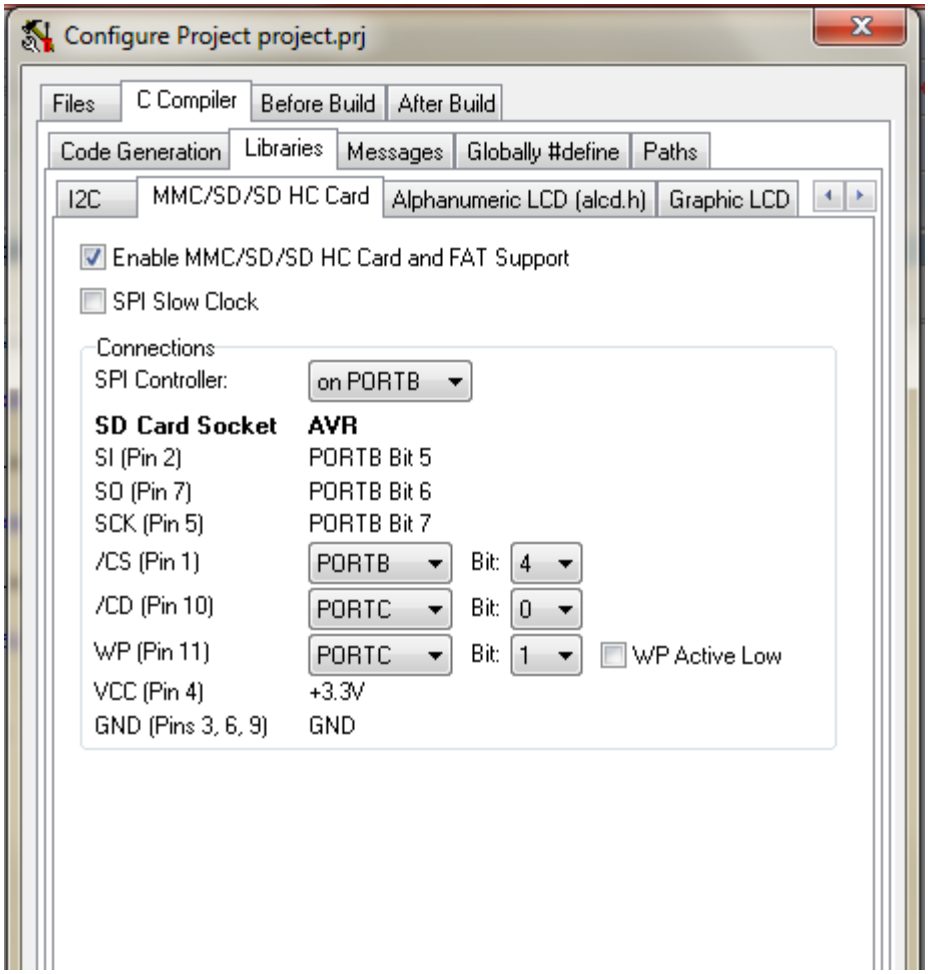
قبل از هر چیز بهتر است که شماتیک و نحوه اتصال کارت حافظه را به میکروکنترلر نمایش دهیم:



کتابخانه ای که جهت کار با این مدل از حافظه ها در نظر گرفته شده است شامل دو هدر sdcard.h که جهت کار با سکتورها و هدر ff.h جهت کار با توابع سطح بالا می باشد.

در اینجا ما به تشریح برخی توابع کتابخانه ff.h می پردازیم.

جهت کار با این کتابخانه باید یک پروژه جدید ایجاد کرده و بعد از اضافه کردن هدر نمایشگر کاراکتری و سایر هدرهای مورد نظر هدر sdcard.h و ff.h را نیز اضافه کرده و همچنین از منوی project/configure/C compailer کتابخانه را فعال کرده و پین های مورد نظر را طبق شکل زیر قرار دهید.



توابع کار با MMC SD:

قبل از تشریح توابع جا دارد یک سری موارد در مورد این کتابخانه توضیح داده شود.

در این کتابخانه یک سری ساختمان داده موجود می باشد که هر کدام عملکرد خاصی را ایفا می کنند و وظیفه نگهداری یک سری اطلاعات خاص شامل: آدرس فایل، سایز فایل و ... می باشد.

در اول تابع main برنامه این ساختمان ها باید معرفی شوند تا بتوان در توابع موجود در برنامه از آن ها استفاده کرد.

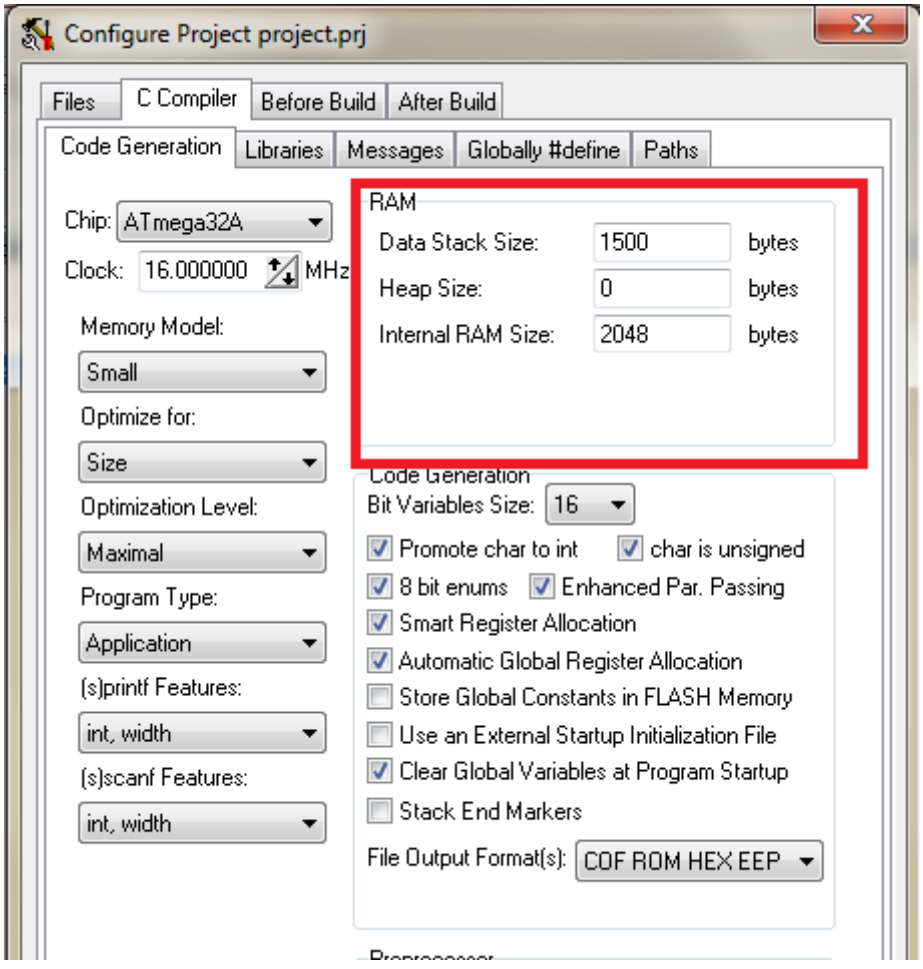
به دلیل سنگین شدن بحث و ذکر این نکته که افراد مبتدی این فایل آموزشی را مطالعه می کنند در این چاپ از ذکر این مطالب خودداری می شود.

عملکرد	نام تابع
این تابع وظیفه چک کردن موجود بودن حافظه را	disk_timerproc();

بر عهده دارد و باید هر ۱۰ میلی ثانیه در وقفه یکی از تایمرها فراخوانی شود.	
جهت باز کردن یک درایو داخل حافظه	f_mount
جهت باز کردن یک فایل	f_open
برای خواندن اطلاعات درون یک فایل از این تابع استفاده می شود.	f_read
جهت نوشتن اطلاعات در فایل باز شده	f_write
برای بستن فایل باز شده جهت ذخیره اطلاعات	f_close

جهت آشنایی هر چه بیشتر دو مثال عملی یکی شامل خواندن محتویات یک فایل txt موجود در mmc و دیگری نوشتن چند بایت اطلاعات درون یک فایل متنی txt آورده شده است.

دقت کنید که در حین برنامه های زیر کامپایلر خطایی را مبنی بر کمبود حافظه sram بیان می کند که باید از مسیر project/configure/c compiler مقدار حافظه sram را افزایش دهید.



۱- برنامه ای بنویسید که توسط آن بتوان فایل متنی با نام "ECA.txt" را باز کرده و چند بایت از آن را خواندهف بر روی نمایشگر نشان دهد؟

تنظیمات روی برد:

تغذیه را در حالت ۳.۳ ولت قرار داده و جامپرهای مربوط به نمایشگر کاراکتری را نیز قرار دهید. درنهایت جهت تغذیه و اتصال کارت حافظه جامپر MMC را وصل کنید.

```
#include <mega32a.h>
#define xtal 16000000
#include <alcd.h>
#include <sdcard.h>
#include <ff.h>
#include <delay.h>
#include <stdlib.h>
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
TCNT0=0xF5;
disk_timerproc();
}
void main(void)
{
FATFS drive;
FIL file;
unsigned int i;
```

```
char buffer[10];


TCCR0=0x05;
TCNT0=0xF5;
OCR0=0x00;
TIMSK=0x01;
#asm("sei")


lcd_init(16);


while(f_mount(0,&drive)!=FR_OK);
lcd_clear();
lcd_putsf("init OK");
delay_ms(500);
while(f_open(&file,"ECA.txt",FA_READ)!=FR_OK);
lcd_clear();
lcd_putsf("file opend");
delay_ms(500);
while(f_read(&file,buffer,5,&i)!=FR_OK);
lcd_clear();
lcd_putsf("read file");
delay_ms(500);
lcd_clear();
lcd_puts(buffer);
```

```
delay_ms(500);
while (1);
}
```

۲- برنامه ای بنویسید که یک فایل متنی را داخل MMC ایجاد کرده و چند بایت اطلاعات داخل آن بنویسد؟

```
#include <mega32a.h>
#define xtal 16000000
#include <alcd.h>
#include <sdcard.h>
#include <ff.h>
#include <delay.h>
#include <stdlib.h>

interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
TCNT0=0xF5;
disk_timerproc();
}
void main(void)
{
FATFS drive;
FIL file;
unsigned int i;
char buffer[11]={"www.ECA.ir"};
```

```
TCCR0=0x05;
TCNT0=0xF5;
OCR0=0x00;
TIMSK=0x01;
#asm("sei")
lcd_init(16);
while(f_mount(0,&drive)!=FR_OK);
lcd_clear();
lcd_putsf("init OK");
delay_ms(500);
while(f_open(&file,"ECA.txt",FA_WRITE)!=FR_OK);
lcd_clear();
lcd_putsf("file Created");
delay_ms(500);
while(f_write(&file,&buffer,5,&i)!=FR_OK);
lcd_clear();
lcd_putsf("file writed");
delay_ms(500);
f_close(&file);
lcd_clear();
lcd_putsf("file close");
while (1);
}
```

ارتباط سریال I2C

پروتکل ارتباطی I2C پروتکل ساخته شده توسط شرکت فیلیپس می‌باشد که توسط آن و از طریق دو سیم می‌توان میان میکروکنترلر و هر وسیله‌ای که دارای چنین قابلیتی باشد ارتباط برقرار کرد.

ویژگی‌های پروتکل I2C

۱. در این ارتباط از دو سیم برای انتقال دیتا استفاده می‌شود.
۲. در این ارتباط می‌توان تعداد نامحدود وسیله جانبی با آدرس سخت افزاری متفاوت را به هم متصل کرد.
۳. بالاترین فرکانس کلاک سیستم ۴۰۰ کیلوهرتز است.
۴. کلاک ارتباط I2C به شدت به کلاک سیستم (فرکانس کریستال اصلی) وابسته است.
۵. حداکثر طول کابل ارتباطی با سیم شیلددار و تقویت کننده ترانزیستوری حداکثر ۸۰ سانتی متر است.

در این ارتباط از دو پایه SDA و SCl که یکی به عنوان خط دیتا و دیگری به عنوان کلاک مورد استفاده قرار می‌گیرد، استفاده می‌شود. همچنین در مسیر ارتباط باید توسط مقاومت دو خط را Pull Up کنید.

انواع مدهای عملکرد ارتباط TWI:

۱- مد عملکرد Master ارسال کننده MT (Master Transmiter)

در این مد ابتدا توسط Master یک وضعیت شروع ایجاد می شود و یک بایت شامل آدرس ارسال می گردد که تعیین کننده فرستنده یا گیرنده بودن Master می باشد. اگر در ابتدای بایت ارسالی یعنی بیت شروع، صفر باشدMaster در حالت ارسال کننده قرار دارد.

۲- مد عملکرد Master گیرنده MR (Master Receiver)

در این مد مستر می تواند از Slave دیتا دریافت نماید. در این حالت باید ابتدا مستر یک وضعیت شروع ایجاد کند و یک بایت شامل آدرس به Slave ارسال کند که تعیین نماید مستر در حالت گیرنده می باشد. اگر در ابتدای بایت ارسالی، بیت شروع یک باشد مستر در حالت دریافت کننده قرار دارد.

۳- مد عملکرد Slave دریافت کننده SR (Slave Receiver)

در این میکروکنترلر در نقش Slave بوده و دیتا را از مستر دریافت می کند. در این مد با درخواست مستر و ارسال بایت حاوی آدرس، Slave این آدرس را بررسی می کند و در صورت درستی، یک پالس شناسایی به مستر در پاسخ ارسال می کند و به عبارتی Slave برای پذیرش دیتا اعلام آمادگی می کند.

۴- مد عملکرد Slave ارسال کننده ST (Slave Transmitter)

در این حالت میکروکنترلر در نقش Slave بوده و دیتا را به مستر ارسال می کند. در این مد با درخواست مستر و ارسال بایت حاوی آدرس با بیت شروع یک، Slave صحت آدرس را بررسی می کند و در صورت درستی یک پالس شناسایی به مستر در پاسخ ارسال می کند و Slave در این مدبه صورت فرستنده عمل می کند.

رجیسترهای ارتباط I2c

رجیستر TWBR:

Bit	7	6	5	4	3	2	1	0	
	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

رجیستر TWCR:

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	

رجیستر TWSR:

Bit	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

رجیستر TWDR:

Bit	7	6	5	4	3	2	1	0	
	TWD7	TWD6	TWD5	TWD4	TWD3	TWD2	TWD1	TWD0	TWDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	1	

رجیستر TWAR:

Bit	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

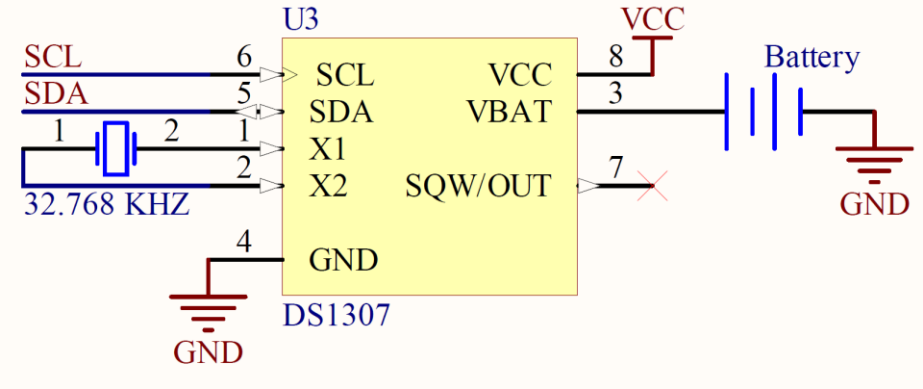
همانطور که قبلا گفته شد اوردن رجیسترها صرفا جهت آشنایی بوده و خود کدویزارد نرم افزار به طور اتوماتیک تنظیمات اولیه را انجام می دهد.

جهت آشنایی با این پروتکل دو مثال عملی آورده شده که یکی شامل راه اندازی IC ساعت DS1307 و دیگری راه اندازی حافظه های EEPROM جانبی سری AT24Cxx می باشند.

مدارات عملی :

۱- با استفاده از تراشه DS1307 موجود بر روی برد یک ساعت طراحی کنید؟

شماتیک مدار:



تنظیمات اعمال شده روی برد آموزشی:

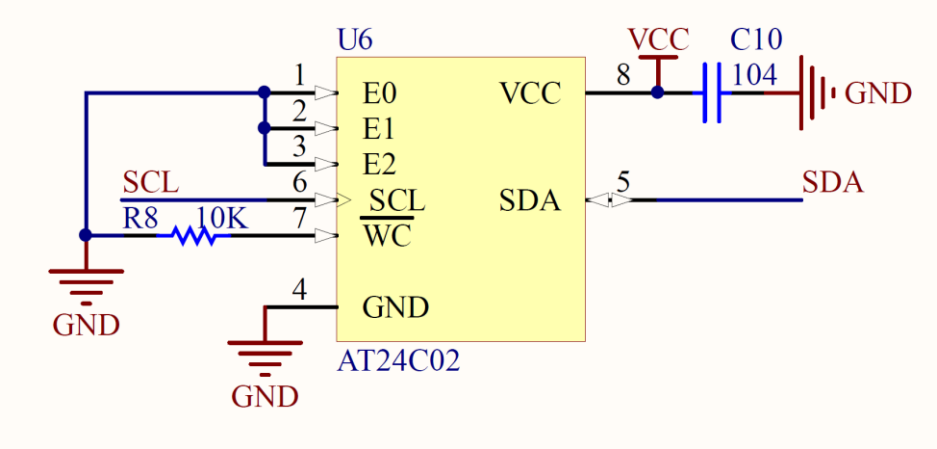
جامپر تغذیه را در حالت ۵ ولت قرار داده و جامپرهای SDA و SCL را وصل کنید. همچنین جهت نمایش اطلاعات جامپرهای مربوط به نمایشگر کاراکتری را متصل کنید.

```
#include <mega32.h>
#include <alcd.h>
#include <stdio.h>
#include <delay.h>
// I2C Bus functions
#asm
   .equ __i2c_port=0x15 ;PORTC
   .equ __sda_bit=1
   .equ __scl_bit=0
#endasm
#include <i2c.h>
// DS1307 Real Time Clock functions
#include <ds1307.h>
void main(void)
{
unsigned char buffer[16];
unsigned char s,m,h;
// TWI initialization
// TWI disabled
TWCR=0x00;
// I2C Bus initialization
i2c_init();
// DS1307 Real Time Clock initialization
```

```
// Square wave output on pin SQW/OUT: On
// Square wave frequency: 1Hz
rtc_init(0,1,0);
rtc_set_time(12,0,0);
// 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);
lcd_putsf("RTC Test");
while (1)
      {
      lcd_gotoxy(0,1);
      rtc_get_time(&h,&m,&s);
      sprintf(buffer,"%2i:%2i:%2i",h,m,s);
      lcd_puts(buffer);
      delay_ms(1000);
      }
}
```

۲- با استفاده از پروتکل I2C حافظه جانبی موجود بر روی برد را راه اندازی کرده و دیتایی را در آن ذخیره و بازیابی کنید؟

شماتیک مدار:




```
#include <mega32a.h>
#include <i2c.h>
#include <alcd.h>
#include <delay.h>
#define EEPROM_BUS_ADDRESS 0xa0//address shoroee neveshtan dar eeprom
unsigned char eeprom_read(unsigned char address)
{
unsigned char data;
i2c_start();
i2c_write(EEPROM_BUS_ADDRESS);
i2c_write(address);
i2c_start();
i2c_write(EEPROM_BUS_ADDRESS | 1);
data=i2c_read(0);
i2c_stop();
return data;
}
void eeprom_write(unsigned char address, unsigned char data)
{
i2c_start();
i2c_write(EEPROM_BUS_ADDRESS);
i2c_write(address);
i2c_write(data);
i2c_stop();
delay_ms(10);
}
void main(void)
```

```
{
unsigned char i;
lcd_init(16);
lcd_clear();
lcd_putsf("write A to e2p");
delay_ms(1000);
i2c_init();
eeprom_write(0x10,'A');
delay_ms(10);
lcd_clear();
lcd_gotoxy(0,0);
lcd_putsf("read eeprom :");
i=eeprom_read(0x10);
delay_ms(10);
lcd_gotoxy(1,1);
lcd_putchar(i);
delay_ms(1000);
while (1);
}
```

توجه: در صورت بروز خطا در عملکرد، به جای حرف A کاراکترها سیاه می شوند.

پیکربندی پروتکل ارتباطی UART

این پروتکل یک ارتباط سریال قابل برنامه ریزی، در دو حالت نرم افزاری و سخت افزاری می‌باشد که بیشتر برای ارتباط میکروکنترلرها با کامپیوتر طراحی شده است. نکته‌ای که حائز اهمیت است سطح ولتاژ در منطق TTL می‌باشد که بین ۰ تا ۵ ولت قرار دارد ولی در پروتکل RS-232 بین ۱۵- تا ۱۵+ قرار دارد که این تبدیل سطح ولتاژ توسط تراشه‌هایی مانند MAX232 و یا MAX235 انجام می‌گیرد.

انواع تبادل سریال

۱- ارسال و دریافت اطلاعات سریال به صورت سنکرون USRT

در این روش دیتای مورد نظر بر روی یک خط همراه با یک خط کلاک همزمان کننده ارسال می شود و گیرنده نیز می تواند دیتا را بر روی یک خط توسط کلاک همزمان کننده که از طرف فرستنده ارسال می شود، دریافت نماید. به طور مثال اطلاعاتی که از موس و کیبرد کامپیوتر به خروجی ارسال می شوند.

۲- ارسال و دریافت اطلاعات سریال به صورت آسنکرون UART

در این روش دیتای مورد نظر بر روی یک خط ارسال TX یا یک خط دریافت RX منتقل می شود و به همراه دیتا کلاکی ارسال نمی گردد بنابراین به اصطلاح می گوییم فرستنده و گیرنده غیرهمزمان عمل می کنند.

به دلیل وجود رجیسترهای متعدد برای این پروتکل از ذکر آن ها خودداری شده است. لذا برای آشنایی هر چه بیشتر قسمت کدویزارد نرم افزار را جهت انجام یک مثال توضیح می دهیم.

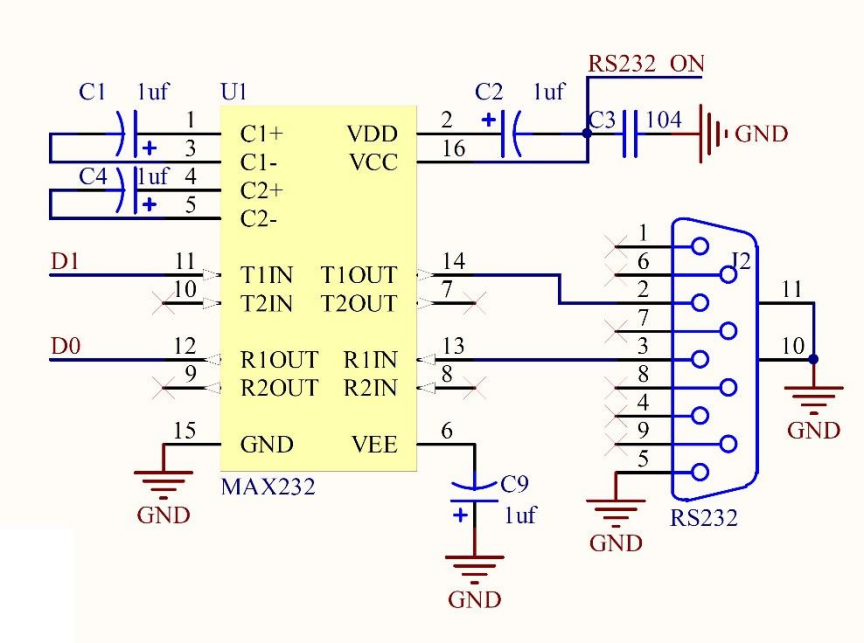
نکته:

دستوراتی که در تابع stdio.h موجود می باشد جهت کار با Usart0 میکروکنترلرها می باشد. در نتیجه وقتی از تراشه ای با دو Usart استفاده می کنید باید توابع مختلف را به صورت دستی برای کامپایلر مشخص کنید.

مدارات عملی :

برنامه ای بنویسید که ابتدا میکروکنترلر دیتایی را ارسال کند و سپس منتظر دریافت اطلاعات از کاربر باشد. با وارد کردن اطلاعات از جانب کاربر و ارسال آن به میکرو، آن را دریافت کرده و بر روی نمایشگر نمایش دهد.

شماتیک مدار:



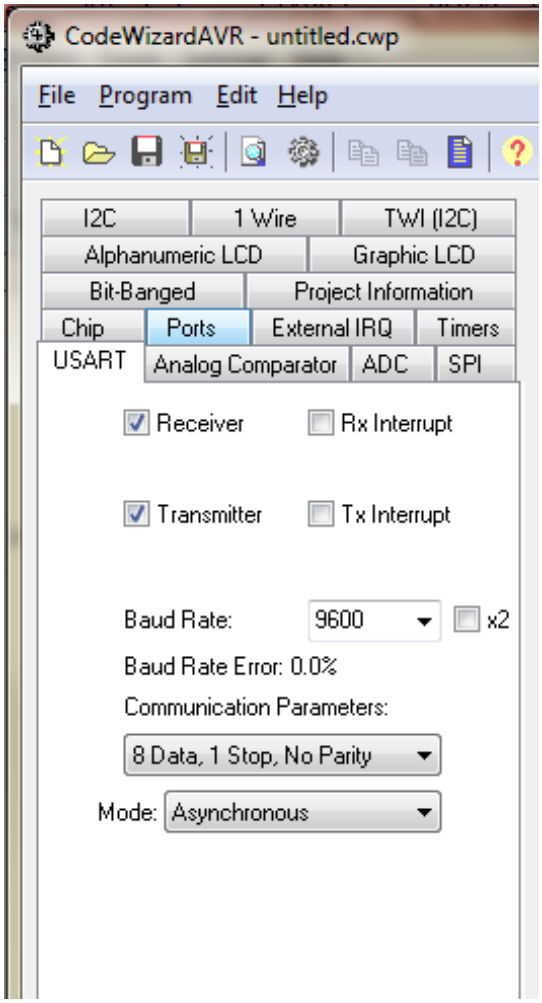
تنظیمات اعمال شده روی برد آموزشی:

جامپر تغذیه و RS232 را متصل کنید.همچنین جهت نمایش دیتا جامپرهای مربوط به نمایشگر کاراکتری را وصل کنید.

توجه: جهت رفع درصد خطا از کریستال خارجی ۱۱.۰۵۹۲ مگاهرتز استفاده شده است درنتیجه کریستال مربوطه را در جای مناسب قرار داده و فیوزبیت کلاک را بر روی کریستال خارجی قرار دهید.

توجه: ما اطلاعات را در نرم افزارهایی که پورت های کام را شناسایی می کنند مشاهده می کنیم. نمونه بارز این نرم افزارها، Hyper terminal می باشد.البته خود کامپایلر هم دارای یک ترمینال می باشد که می توانید به اختیار یکی را برگزینید.

قسمت کدویزاد:




```
#include <mega32a.h>
#include <delay.h>
#include <alcd.h>
// Standard Input/Output functions
#include <stdio.h>

void main(void)
{
char buffer[];
// USART initialization
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud Rate: 9600
UCSRA=0x00;
```

```
UCSRB=0x18;
UCSRC=0x06;
UBRRH=0x00;
UBRRL=0x47;
// Alphanumeric LCD initialization
// Connections are 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);
lcd_putsf("   www.ECA.ir");
while (1)
      {
       printf("Enter STRING??");
       scanf("%s",buffer);
       lcd_clear();
       lcd_puts(buffer);
      }
}
```