دانشگاه صنعتی اصفهان

دانشکده برق و کامپیوتر

دستورکار آزمایشگاه ریزپردازنده

تهیه کنندگان

دکتر نادر کریمی

دکتر محمدحسین منشئی

آخرین ویرایش

مهندس مهدی بلوریان

۱۳۹۸

فهرست

قوانین و مقررات آزمایشگاه ریزپردازنده ۳
آزمایش اول: آشنایی با ریزپردازنده ۵
آزمایش دوم: کار با LCD کاراکتری، صفحه‌کلید ماتریسی و آشنایی با وقفه‌ها ۱۶
آزمایش سوم: آشنایی با وقفه‌ها و تایمرها ۲۶
آزمایش چهارم: کار با موتورهای الکتریکی ۳۸
آزمایش پنجم: آشنایی با مبدل آنالوگ به دیجیتال و دیجیتال به آنالوگ ۴۶
آزمایش ششم: کار با LCD گرافیکی ۶۲
آزمایش هفتم: آشنایی با ارتباط سریال RS232 ..... ۷۴
آزمایش هشتم: آشنایی با ارتباط SPI و دوسیمه و کار با حافظه EEPROM ۸۵

برای آگاهی بیشتر شما دانشجویان عزیز با قوانین و مقررات آزمایشگاه ریزپردازنده نکات زیر یادآوری می‌گردد:

۱) **دستور کار:** برای هر جلسه از آزمایشگاه یک دستور کار وجود دارد که باید آن را در طول هفته و قبل از هر جلسه مطالعه کنید.

۲) **پیش گزارش:** در دستور کار هر جلسه تعدادی سوال تئوری و شبیه‌سازی وجود دارد که **هر دانشجو به صورت انفرادی** ملزم به پاسخ‌گویی به آن‌هاست. تمام برنامه‌ها باید دارای توضیحاتی در مورد چگونگی الگوریتم استفاده شده باشد. زمان تحویل پیش‌گزارش قبل از شروع به کار در هر جلسه از آزمایشگاه است و **بعد از آن به هیچ عنوان تحویل گرفته نخواهد شد.** پیش گزارش تحت قالب یک فایل فشرده zip. می‌باشد که این فایل شامل یک فایل pdf. به همراه فایل‌های مربوط به شبیه‌سازی و کدهای نوشته به زبان C است.

۳) **حضور در جلسات:** حضور در تمامی جلسات الزامی بوده و هر غیبت غیر مجاز باعث کسر نمره می‌گردد. به علاوه تمامی اعضای گروه باید آمادگی لازم را برای پاسخگویی به سوالات احتمالی و توضیح دادن برنامه‌ها داشته باشند. یکی از روش‌های ارزشیابی برگزاری کوئیز در جلسات آزمایشگاه است.

۴) **نظم آزمایشگاه:** دانشجویان موظف‌اند از کلیه‌ی وسایلی که در اختیار آن‌ها قرار داده می‌شود مراقبت کرده و در پایان هر جلسه آن‌ها را در جای مربوط به خود قرار دهند.

۵) **فعالیت اضافه:** انجام پروژه‌های طرح‌شده توسط مربی آزمایشگاه می‌تواند نمرات اضافه برای دانشجویان داشته باشد.

۶) **ارزشیابی نهایی:** ارزشیابی نهایی به صورت برگزاری امتحان پایان‌ترم به صورت کتبی و عملی خواهد بود.

تنها برای تأکید بیشتر در پایان نکاتی یادآوری می‌شود که رعایت آن برای تمام دانشجویان الزامیست و نادیده گرفتن آن‌ها اگرچه تذکری از سوی مربی آزمایشگاه داده نشود، مطمئناً برای شما در نظر گرفته می‌شود.

- حضور به موقع در جلسات
- رعایت نظم و سکوت در جلسات
- حضور فعال در گروه
- کار کردن مناسب با دستگاه‌های موجود در آزمایشگاه
- پرهیز از هرگونه کپ زدن(!)

با تشکر

مسئولین آزمایشگاه ریزپردازنده

آزمایش اول

اهداف:

- مرور اصول برنامه نویسی
- نوشتن و خواندن بر روی پورت‌های ریزپردازنده و رجیسترهای مربوط به آن
- آشنایی با کلیدهای فشاری و کلیدهای کشویی
- آشنایی با نمایشگر 7-Segment
- معرفی هدرفایل delay.h

مقدمه:

هدف از این آزمایشگاه آشنایی با برخی از قابلیت‌های ریزپردازنده ATmega16 و برنامه‌نویسی آن در کنار سایر دستگاه‌های جانبی است.

برای نوشتن برنامه از زبان C و نرم‌افزار CodeVision استفاده می‌شود. همانطور که می‌دانید لازمه‌ی هر طرح عملی پیش‌طراحی و شبیه‌سازی نرم‌افزاری آن می‌باشد. بدین منظور قسمت‌هایی تحت عنوان شبیه‌سازی با نرم‌افزار Proteus در پیش‌گزارش قرار داده شده است. برای آشنایی مقدماتی با نرم‌افزار CodeVision ضمیمه‌ی شماره‌ی ۱ و نرم‌افزار Proteus ضمیمه شماره ۲ را **حتماً مطالعه بفرمایید**.

معرفی ورودی و خروجی های ATmega16:

ریزپردازنده ATmega16 دارای ۴۰ پایه است که ۳۲ پایه‌ی آن به عنوان پورت‌های I/O کاربرد دارد که در شکل ۲ مشخص شده است. در ATmega16 ، ۴ سری پورت ۸ بیتی وجود دارد.

در ساختار داخلی ATmega16 برای هر کدام از این پورت ها سه ثبات[1] به منظور تعیین جهت و مقدار این پورت‌ها وجود دارد که به شرح زیر می‌باشند:

۱. رجیستر داده پورت PORTx[2]
۲. رجیستر جهت داده پورت DDRx[3]
۳. بایت آدرس پایه‌های ورودی پورت PINx[4]

[1] Register
[2] PORTX data register
3 PORTX data direction register
4 PORTX input pins address

PORTA – Port A Data Register

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	

DDRA – Port A Data Direction Register

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	

PINA – Port A Input Pins Address

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	

شکل ۱ رجیسترهای پورت A

از بین رجیسترهای بالا تنها رجیستر PINx قابل خواندن است و دو رجیستر دیگر هم قابل خواندن و هم قابل نوشتن هستند.

اگر DDRxn یک پایه برابر صفر باشد پایه به صورت ورودی و اگر ۱ باشد پایه به صورت خروجی خواهد بود.

برای خواندن وضعیت پایه‌ای که به صورت ورودی تعریف شده است رجیستر PINx را می‌خوانیم. و برای تعیین وضعیت پایه‌هایی که بصورت خروجی تعریف شده‌اند بر روی رجیستر PORTx می‌نویسیم.

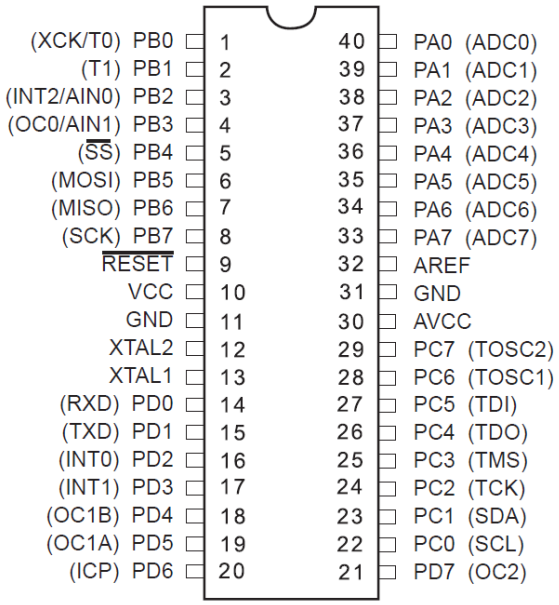
```
DDRB = 0x0F;
```

برای مثال، با استفاده از دستور بالا ۴ بیت کم ارزش پورت B به عنوان خروجی و ۴ بیت پر ارزش آن به عنوان ورودی تعریف شده است.

البته این پورت‌ها علاوه بر استفاده به عنوان I/O کاربردهای دیگری هم دارند که امکان استفاده از سایر قابلیت‌های ATmega16 مانند مبدل آنالوگ به دیجیتال، تایمر، ارتباط سریال و ... را فراهم می‌آورند که در ادامه این آزمایشگاه با این کاربردها بیشتر آشنا خواهید شد.

دسترسی به بیت‌های هر کدام از رجیسترهای نام‌برده به صورت زیر خواهد بود:

```
DDRA.3 = 1;  //set bit 3 of port A as output
PORTA.3 = 1;  //make bit 3 of port A high
```



شکل ۲پایه های ریزپردازنده ATmega16

معرفی هدر (Header) فایل :

برای اضافه کردن یک فایل هدر به ابتدای برنامه از دستور #include استفاده می‌شود که نحوه اضافه کردن آن به دو روش زیر می‌باشد:

```
#include <file_name>
#include "file_name"
```

در حالت اول کامپایلر فایل را در زیر دایرکتوری /inc در محل نصب برنامه جست‌وجو می‌کند، اما در حالت دوم ابتدا پوشه جاری برنامه را جست‌وجو می‌کند سپس به مسیر /inc رجوع می‌کند.

در پوشه INC در محل نصب CodeVision می‌توانید کلیه فایل‌های قابل include کردن که حاوی توابع و اسامی از پیش نوشته‌شده هستند را مشاهده نمایید.

معرفی هدر(Header) فایل ریزپردازنده:

ریزپردازنده‌های AVR دارای ۳۲ ثبات I/O هستند که ارسال و دریافت داده، پیکربندی ریزپردازنده و تنظیم امکانات داخلی آن از طریق این ثبات‌ها انجام می‌شود. به دو روش می‌توان مقدار این ثبات‌ها را در برنامه تنظیم نمود. روش اول تنظیم رجیسترها با استفاده از آدرس آن‌ها در حافظه ریزپردازنده است. برای مثال، رجیستر DDRx ورودی یا خروجی بودن پورت x ریزپردازنده را مشخص می‌کند. رجیستر DDRD در آدرس ۱۷ از حافظه ATmega16 قرار گرفته است. می‌توان با تنظیم خانه حافظه که این آدرس به آن اشاره می‌کند ورودی یا خروجی بودن پورت D را مشخص کرد.

روش دوم برای تنظیم ثبات‌های داخلی ریزپردازنده، استفاده از نام‌های تعریف‌شده در داخل هدر فایل است. هر ریزپردازنده هدرفایل مخصوص به خود را دارد که در آن برای تمامی رجیسترهای ریزپردازنده یک اشاره‌گر براساس نام و آدرس آن‌ها درحافظه تعریف شده است. این آدرس‌ها و اشاره‌گرها را می‌توان از Datasheet مربوطه بدست آورد. با include کردن این هدر فایل می‌توان به جای آدرس ثبات، از اسم آن ثبات استفاده کرد.

مثال: تنظیم پورت D به صورت خروجی با استفاده از آدرس آن.

```
void main(void)
{
    *(unsigned char *) 0x11=0b11111111;
    while(1) { };
}
```

مثال: تنظیم پورت D به صورت خروجی با استفاده از نام DDRD که در هدرفایل ریزپردازنده تعریف شده است.

```
#include <mega16.h>
void main(void)
{
    DDRD=0xFF;
    while(1) { };
}
```

توجه : حتما باید هدرفایل در ابتدای برنامه include شود. در غیر این صورت با خطای کامپایل مواجه خواهید شد.

معرفی هدر delay.h:

این کتابخانه شامل دو تابع void delay_us(unsigned int n) و void delay_ms(unsigned int n) است که به کمک آن‌ها می‌توان در برنامه تأخیر ایجاد کرد.

لازم به ذکر است در هنگام استفاده از این توابع اگر وقفه‌ای فعال باشد ممکن است تأخیر بیش از اندازه طولانی گردد. لذا مجاز نیستیم به طور دلخواه همیشه از این توابع استفاده نماییم.

معرفی مقاومت داخلی پورت‌ها:

هنگامی که یک پایه را به صورت ورودی تنظیم می‌کنیم گاهی ممکن است در اثر تغییرات ولتاژ، تغییرات جریان، نویز و... حالت آن پایه به صورت ناخواسته تغییرکند. با توجه به سرعت بالای ریزپردازنده این تغییر ممکن است به اشتباه برای ریزپردازنده به معنی یک یا صفر شدن پایه تلقی شود و در اجرای برنامه اخلال ایجاد کند. برای از بین بردن این تأثیرات ناخواسته باید با استفاده از مقاومت‌های pull up و یا pull down پایه‌ها از حالت شناور خارج شده، به یکی از خطوط تغذیه متصل شوند. برای این کار می‌توان از مقاومت خارجی یا مقاومت داخلی تراشه‌های AVR استفاده نمود.

برای فعال کردن مقاومت داخلی پورت های I/O باید بیت PUD: Pull-up disable از رجیستر SFIOR (بیت شماره ۲ این رجیستر) را صفر در نظر گرفت (و برای غیرفعال کردن، باید آن را یک کرد). علاوه بر این برای فعال کردن

مقاومت داخلی هرکدام از *پایه‌های ورودی* باید بیت متناظر آن‌ها را در ثبات *PORTx* یک کرد و برای غیرفعال نمودن آن باید این بیت را صفر نمود یا پایه را به صورت خروجی تعریف کرد.

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	

شکل ۳ ثبات Special Function I/O Register: SFIOR

معرفی کلیدهای کشویی:

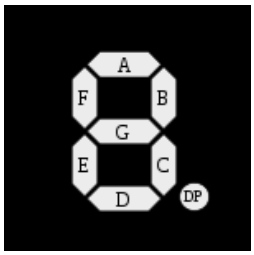
۱۰ عدد کلید کشویی به منظور تولید صفر و یک دائمی در بلوکی تحت عنوان Data Latch Switch در این مجموعه‌ی آموزشی قرار داده شده است. این کلیدها وظیفه تولید صفر و یک منطقی را بر عهده دارند. در ساختار استفاده شده در این برد آموزشی، از مقاومت‌های Pull up و یا Pull down برای جلوگیری از به وجود آمدن حالت شناور بر روی پایه‌های ریزپردازنده استفاده شده است.

معرفی Push Button:

از Push Buttonها به منظور تولید صفر و یک لحظه‌ای استفاده می‌شود. در برد آموزشی که در اختیار دارید ۸ عدد Push Button قرار دارد که از آن‌ها برای تولید صفر و یک منطقی استفاده می‌شود. با فشار دادن این کلیدها، یک سیگنال صفر یا یک در پین‌هدر متناظر ایجاد می‌کند و بلافاصله پس از رها کردن کلید، خروجی به سطح یک یا صفر باز می‌گردد.

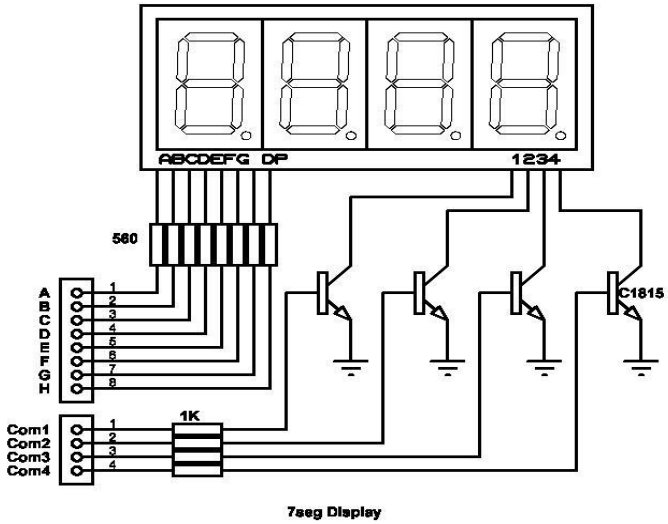
معرفی نمایشگر 7-Segment:

این نمایشگر چهار رقمی، برای نمایش اعداد و بعضی از حروف لاتین به کار می‌رود. در این ماژول برای هر 7-Segment یک پایه‌ی Enable به نام‌های A الی H از نوع پین‌هدر و کانکتور 2mm وجود دارد. اما هشت پایه‌ی داده برای هر چهار 7-Segment مشترک هستند. که یک کردن هر کدام از پایه‌های داده، موجب روشن شدن Segment متناظر با آن خواهد شد. بنابراین برای نمایش یک عدد خاص، بر روی هر 7-Segment باید داده‌ی مناسب را بر روی پایه‌های داده ارسال کرده و Enable آن را یک کرد. چنانچه این کار به صورت متناوب و با فرکانس مناسب انجام شود، می‌توان به صورت همزمان اعداد چهار رقمی دلخواه را روی چهار 7-Segment رؤیت کرد.



شکل ۴ نامگذاری بخش‌های 7-Segment

با توجه به شکل ۴ و ۵ برای نمایش عدد '۱' باید به Segmentهای B و C مقدار ۱ منطقی اعمال کرد.



شکل ۵ نحوه اتصال پایه‌های 7-Segment

پیش گزارش آزمایش اول

نام و نام خانوادگی: **شماره دانشجویی:**

---

۱) الف- ساختار مقاومت pull-up و کاربرد آن را مختصراً شرح دهید.

ب - انواع روش‌های اتصال کلید به ریزپردازنده را نام ببرید.

ج - انواع روش‌های برنامه‌ریزی ریزپردازنده‌های AVR را نام ببرید.

د - ریزپردازنده AVR چند نوع حافظه دارد؟ نحوه استفاده از آنها چگونه است؟ تفاوت آن‌ها در چیست؟

۲) نحوه‌ی عملکرد رجیسترهای DDRx, PORTx, PINx را توضیح دهید و پیکربندی لازم برای موارد زیر را بنویسید:

الف) پورت A را طوری تعریف کنید که بیت های آن یک در میان ورودی خروجی باشند.

ب) پورت B را بصورت خروجی تعریف کنید بطوریکه مقاومت Pull up آن حذف نشود.

۳) برای نمایش کاراکترهای زیر، چه مقادیری را باید به ورودی خط داده‌ی 7-Segment اعمال کرد.

d
A
H
F

۴) آزمایش ۱-۹ را با استفاده از نرم‌افزار Proteus شبیه‌سازی کنید. (کد نوشته شده به زبان C، شماتیک مدار و اطلاعات مربوط به Simulation log را در فایل پیش‌گزارش قرار دهید)

دستور کار

قسمت اول: آشنایی با رجیسترهای PORTx, PINx, DDRx

آزمایش۱-۱:

با استفاده از رشته سیم‌های مخصوص، LED های قرمز رنگ را به GND و LEDهای سبز رنگ را به منبع تغذیه 5v متصل کرده و نتیجه را مشاهده نموده و مدار معادل را در هر دو حالت گزارش نمایید.

آزمایش ۱-۲:

با توجه به ضمیمه دستور کار، ابتدا یک پروژه بسازید (بدون استفاده از Code Wizard). برنامه زیر را به پروژه خود اضافه کرده و سپس روی برد پروگرام نمایید.

```
#include <mega16.h>
void main(void)
{
    DDRB=0xFF;
    PORTB=0b01010101;
    while(1);
}
```

سوال: آیا می‌توان هر کدام از پایه‌های ریزپردازنده را به طور مجزا مقدار دهی کرد؟ چگونه؟

آزمایش ۱-۳:

کلیدهای کشویی را به پورت B و LED ها را به پورت D وصل کنید و برنامه زیر را به پروژه خود اضافه کرده و سپس روی برد پروگرام نمایید.

```
#include <mega16.h>
char number;
void main(void)
{
    DDRB = 0x00;
    DDRD = 0xFF;
    while(1)
    {
        number = PINB;
        PORTD  = number;
    }
}
```

آزمایش ۱-٤:

یکی از کلیدهای کشویی را به بیت صفر پورت B وصل کنید و LEDها را به پورت D وصل کنید. سپس برنامه زیر را اجرا کنید.

```
#include <mega16.h>
void main(void)
{
    char count = 0X00;
    DDRD = 0XFF;
    PORTD = 0X00;
    DDRB.0 = 0;
    while(1)
    {
        if (PINB.0)
        {
            count = count + 1;
            PORTD = count;
        }
    }
}
```

برنامه بالا کد یک شمارنده است که در صورت یک بودن PINB.0 که به کلید کشویی متصل است به مقدار آن اضافه می شود. با استفاده از دستور ;char count=0x00 یک متغیر ۸ بیتی تعریف شده است که مقدار اولیه آن صفر بوده مقدار شمارنده را در هر لحظه در خود نگه می دارد و هر بار مقدار آن با ریخته شدن روی PORTD بر روی LEDها نمایش داده می شود. همانطور که ملاحظه می کنید LEDها به سرعت خاموش و روشن می شوند و شمارنده به درستی عمل نمی‌کند. علت آن این است که سرعت کار ریزپردازنده بسیار زیاد است و شمارنده با سرعت بالایی کار می‌کند برای حل این مشکل باید شمارش با فاصله زمانی معلومی انجام شود. برای ایجاد تأخیر می‌توان از هدرفایل delay.h استفاده کرد. این هدر فایل شامل توابعی است که به کمک آن می‌توان تاخیرهایی در واحد میلی‌ثانیه و یا میکروثانیه ایجاد کرد. با استفاده از دستور delay_ms(unsigned int delay_time) و delay_us(unsigned int delay_time) می‌توان این توابع را فراخوانی کرد.

آزمایش ۱-۵:

در این آزمایش قصد داریم با کلیدهای فشاری و هدرفایل delay.h آشنا شویم. همانطور که در مقدمه به آن اشاره شد ساختار کلیدهای فشاری مورد استفاده در این آزمایشگاه به گونه‌ای است که خروجی کلید با فشردن آن، به زمین متصل خواهد شد.

هدرفایل delay.h را فراخوانی کنید و با استفاده از تابع delay برنامه قبل را به گونه‌ای تغییر دهید که شمارنده در صورت یک بودن کلید فشاری به صورت صعودی و در صورت صفر بودن آن به صورت نزولی با تأخیر ۰.۵ ثانیه‌ای بشمارد.

آزمایش ۱-۶:

با توجه به اطلاعاتی که از دستور shift در پیش‌گزارش بدست آورده‌اید، برنامه‌ای بنویسید که LEDها را به صورت چرخشی، از بالا به پایین به ترتیب روشن و خاموش کند.

قسمت دوم: آشنایی با نمایشگر 7-Segment

در این بخش قصد داریم یک عدد ثابت روی یکی از چهار 7-Segment مالتی‌پلکس شده نمایش دهیم. برای آشنایی با 7-Segmentهای موجود روی برد به مقدمه رجوع کنید. همانطور که می‌دانید با برقراری جریان در هر کدام از پایه‌های 7-Segment یکی از خانه‌های هشت‌گانه‌ی آن روشن می‌شود. اما برای آنکه یک عدد را از متغیر n بخوانیم و روی 7-Segment نمایش دهیم، باید کد مربوط به آن عدد را به پورت متصل به 7-Segment بدهیم. برای این‌کار می‌توان متغیر ثابت زیر را بالای main تعریف کرده و هر جا که نیاز به نوشتن عدد روی 7-Segment باشد از آن استفاده کنیم.

```
flash unsigned char digit[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D,
0x7D, 0x07, 0x7F, 0x6F};
```

در این متغیر، مثلا عضو صفرم، کد مربوط به عدد صفر، روی 7-Segment است.

```
PORTx = digit[n];
```

آزمایش ۱-۷:

در این قسمت از آزمایش قصد داریم یک عدد ثابت یک رقمی را روی یکی از نمایشگرهای 7-Segment نمایش دهیم. برای این منظور، ابتدا پایه‌ی Enable مربوط به 7-Segment اول که روی برد با علامت Digit1 مشخص شده است را به منبع تغذیه ۵ ولت متصل کنید. سپس پایه‌های A الی H را به پورت D متصل نمایید. حال برنامه‌ی زیر را روی برد پروگرام کرده و نتیجه را مشاهده نمایید.

```
#include <mega16.h>
flash unsigned char digit[]={0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D,
0x7D, 0x07, 0x7F, 0x6F};
int number = 0;
void main(void)
{
```

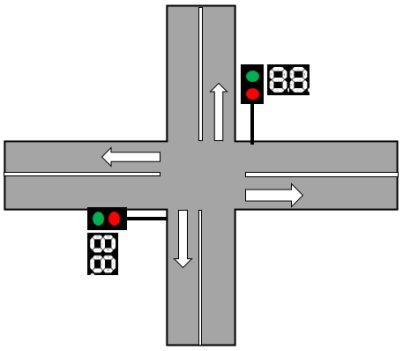
```
    DDRD = 0xFF;
    number = 5;
    while(1)
    {
          PORTD = digit[number];
    }
}
```

آزمایش ۱-۸:

با استفاده از آزمایش قبل، برنامه‌ای بنویسید که عدد یک رقمی را از روی کلید کشویی دریافت کرده و آن را روی نمایشگر 7-Segment نمایش دهد.

آزمایش ۱-۹:

در این قسمت هدف پیاده‌سازی چراغ راهنمایی با رنگ سبز و قرمز می‌باشد. مطابق شکل برای هر خیابان یک چراغ و دو 7_segment وجود دارد که در زمان قرمز بودن، مدت زمان باقی‌مانده تا سبز شدن چراغ را نشان می‌دهد. همچنین برای کنترل ترافیک توسط پلیس راهنمایی از کلید کشویی استفاده می‌شود به این صورت که پلیس توانایی نگه داشتن زمان باقی مانده یا صفر کردن آن را دارد.



آزمایش دوم

اهداف:

- آشنایی با مفهوم وقفه‌ی خارجی (External interrupt)
- کار با نمایشگر LCD کاراکتری
- کار با صفحه‌کلید ماتریسی

مقدمه:

هدف از این آزمایش آشنایی با مفهوم وقفه به عنوان مقدمه‌ای برای کار با تایمرها می‌باشد. در این آزمایش از صفحه‌کلید ماتریسی و نمایشگر LCD به عنوان ابزارهای ورودی و خروجی استفاده شده است. در ابتدا پس از آشنایی با مفهوم وقفه، نحوه‌ی خواندن اطلاعات از صفحه‌کلید ماتریسی و نمایش آن‌ها بر روی LCD را دنبال خواهیم‌کرد.

آشنایی با وقفه‌ها

در یک برنامه ریزپردازنده گاهی به بررسی وقوع یک رویداد و انجام عمل خاصی در پاسخ به آن نیاز پیدا می‌کنیم. به طور کلی برای تشخیص چنین رویدادهایی از دو روش استفاده می‌شود:

۱. روش سرکشی (Polling): در این روش، برنامه‌نویس پردازنده را طوری برنامه‌ریزی می‌کند که وقوع رویداد مورد نظر با فواصل زمانی مشخص و پی‌درپی مورد بررسی قرار گیرد و در صورت وقوع رویداد، به آن پاسخ می‌دهد. از مشکلات این روش می‌توان به هدر رفتن وقت پردازنده برای چک کردن وقوع اتفاق مورد نظر می‌باشد.

۲. روش استفاده از وقفه (Interrupt): وقفه امکانی در ریزپردازنده است که باعث می‌شود تراشه در قبال ایجاد یک رویداد لحظه‌ای (که معمولاً زمان آن قابل پیش‌بینی نیست) عمل خاصی را انجام دهد. در روش استفاده از وقفه، برنامه اصلی در حالت عادی خود اجرا می‌شود، اما به محض وقوع رویداد مورد نظر، پردازنده خط جاری برنامه را به انتها رسانده و اجرای بقیه‌ی برنامه را متوقف کرده و به صورت سخت‌افزاری با زیرروال سرویس‌دهی به آن وقفه پرش می‌کند. پس از اتمام اجرای زیرروال وقفه اجرای برنامه از سرگرفته می‌شود.

در مورد دلایل ایجاد وقفه می‌توان به دو مورد اشاره کرد:

۱. وقفه سخت‌افزاری که در پاسخ به رویداد خارجی مانند تغییر مقدار یک پایه یا رسیدن مقدار شمارنده تایمر به عدد خاصی اتفاق می‌افتد.

۲. وقفه نرم‌افزاری که در پاسخ به اجرای یک دستور در برنامه اتفاق می‌افتد. مانند وقفه‌ی تقسیم بر صفر یا وقفه‌هایی که در اثر فراخوانی تابع سیستمی توسط برنامه رخ می‌دهد. به دلیل محدودیت‌های نرم‌افزاری

که میکروکنترلرهای AVR هشت بیتی دارند این نوع وقفه‌ها (که معمولاً در صورت وجود سیستم‌عامل مطرح می‌شوند) در این آزمایشگاه به کار نمی‌آیند.

از جمله مزایای استفاده از وقفه می‌توان به موارد زیر اشاره کرد:

۱. امکان اولویت بندی درخواست وقفه از وسایل جانبی مختلف

۲. رسیدگی کردن بلادرنگ به درخواست وقفه (در صورت فعال نبودن یک درخواست وقفه با اولویت بالاتر)

۳. امکان نادیده گرفتن درخواست وقفه از یک یا چند وسیله جانبی

۴. عدم مشغول شدن پردازنده در زمان‌هایی که درخواستی برای ارائه سرویس وقفه موجود نیست.

برای فعال‌سازی وقفه‌ها در ریزپردازنده‌های AVR باید بیت هفتم از رجیستر SREG (فعال ساز عمومی) را فعال کرد. در زبان C با نوشتن عبارت (“sei”)#asm این بیت فعال می‌شود. پس از فعال کردن این بیت می‌توان هر کدام از وقفه‌های موجود در AVR را فعال کرد. همان‌طور که در جدول ۱ مشاهده می‌شود سه مورد از وقفه‌ها، تحت عنوان External interrupt بوده (وقفه‌هایINT0، INT1، INT2) و به آن‌ها وقفه های خارجی گفته می‌شود. از وقفه های خارجی معمولاً برای تشخیص پالس استفاده می‌شود. این پالس می‌تواند حاوی اطلاعات خاصی مانند اطلاعات دریافتی از یک سنسور یا IC باشد که به یکی از پایه‌های PB2 ، PD2 و PD3 متصل است. غیر از این سه وقفه و وقفه شکار تایمر، سایر وقفه‌ها، وقفه داخلی نام می‌گیرند.

در برنامه نویسی برای وقفه‌ها لازم است کد تابع یا زیرروال مربوط به وقفه درست در محلی که ریزپردازنده قرار است به آن پرش کند نوشته شود. برای راحت‌تر شدن این مسئله کامپایلر این امکان را در اختیار ما قرار می‌دهد که تابعی را که می‌نویسیم از نوع تابع وقفه تعریف نموده و سپس تنها بگوییم که مربوط به وقفه شماره چند است. به طور کلی نحوه تعریف تابع وقفه در CodeVision به صورت زیر است:

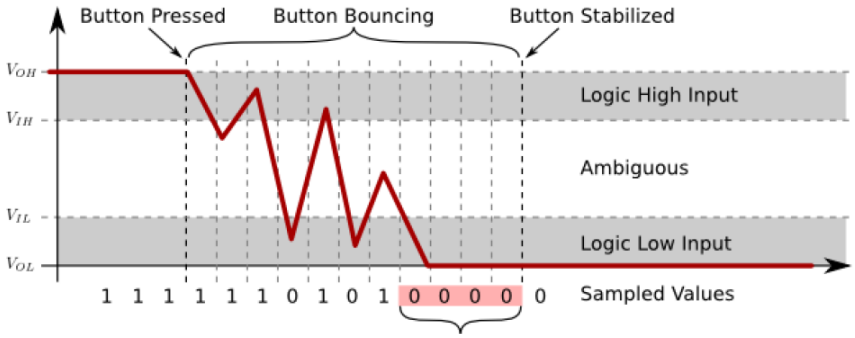
interrupt[شماره بردار وقفه] void نام تابع (void) {}

جدول ۱ – وقفه‌های ریزپردازنده

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$002	INT0	External Interrupt Request 0
3	$004	INT1	External Interrupt Request 1
4	$006	TIMER2 COMP	Timer/Counter2 Compare Match
5	$008	TIMER2 OVF	Timer/Counter2 Overflow
6	$00A	TIMER1 CAPT	Timer/Counter1 Capture Event
7	$00C	TIMER1 COMPA	Timer/Counter1 Compare Match A
8	$00E	TIMER1 COMPB	Timer/Counter1 Compare Match B
9	$010	TIMER1 OVF	Timer/Counter1 Overflow
10	$012	TIMER0 OVF	Timer/Counter0 Overflow
11	$014	SPI, STC	Serial Transfer Complete
12	$016	USART, RXC	USART, Rx Complete
13	$018	USART, UDRE	USART Data Register Empty
14	$01A	USART, TXC	USART, Tx Complete
15	$01C	ADC	ADC Conversion Complete
16	$01E	EE_RDY	EEPROM Ready
17	$020	ANA_COMP	Analog Comparator
18	$022	TWI	Two-wire Serial Interface
19	$024	INT2	External Interrupt Request 2
20	$026	TIMER0 COMP	Timer/Counter0 Compare Match
21	$028	SPM_RDY	Store Program Memory Ready

منظور از شماره بردار وقفه همان شماره ستون جدول بالاست که با نام .Vector No مشخص هستند. ولی اگر فایل هدر mega16.h را باز کنید مشاهده خواهید کرد که برای هر کدام از این شماره‌ها نام های معادلی در نظر گرفته شده که به سهولت استفاده از وقفه‌ها کمک بسیاری می‌کند. مثلاً برای استفاده از وقفه خارجی ۱ به جای نوشتن شماره ۳ می توان عبارت EXT_INT1 را به کار برد.

یکی از المان‌هایی که برای اتصال به پایه‌های وقفه خارجی ریزپردازنده استفاده می‌شود Push-Button است. Push-Button کلیدی است که در حالت عادی قطع است و مادامی که توسط کاربر فشار داده شود، وصل خواهد بود و به محض رها شدن، دوباره قطع خواهد شد. یکی از مشکلات رایج در Push-Buttonها پدیده Bouncing است. این پدیده در اثر لرزش اتصالات مکانیکی کلیدها در هنگام قطع و وصل شدن، رخ می‌دهد. چنین لرزش‌هایی موجب می‌شود پیش از رسیدن کلید به حالت دائمی، چند بار صفر و یک شود. زمان رسیدن کلید به حالت دائمی معمولاً بین ۱۰ تا ۲۰ میلی ثانیه است. به عنوان مثال در شکل زیر بعد از فشرده شدن Push-Button تا زمانی که به حالت ثابت صفر برسد چند صفر و یک دیگر تولید شده و این باعث ایجاد خطا در خروجی می‌شود.

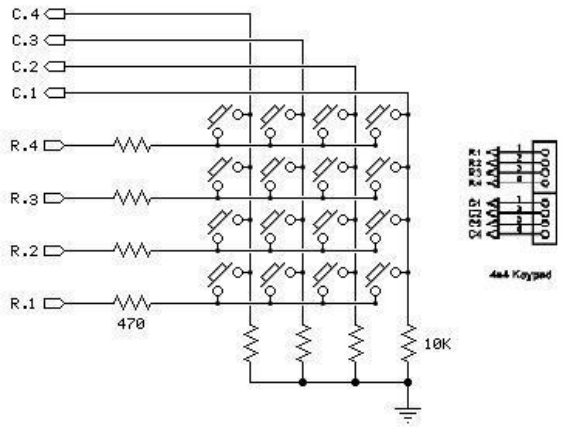


شکل ۶: اثر bouncing

یکی از روش‌های حل این مشکل استفاده از تأخیر می‌باشد که روش چندان مناسبی نیست.

آشنایی با صفحه‌کلید ماتریسی

یک عدد صفحه‌کلید ماتریسی از نوع 4x4 (دارای ۴ ردیف و ۴ ستون) توسط ۱۶ عدد کلید فشاری به منظور تولید اطلاعات مختلف در بلوکی با عنوان 4x4 keypad در این مجموعه آموزشی قرار داده شده است. عکس مربوط به شماتیک keypad در شکل زیر مشاهده می‌شود.



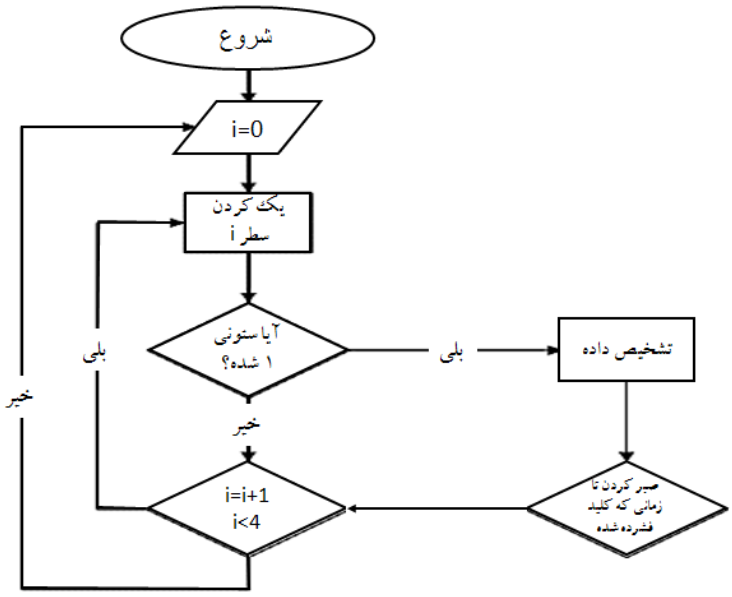
شکل ۷: ساختار صفحه کلید ماتریسی استفاده شده در برد آموزشی

نحوه کار keypad به این صورت است که باید keypad را به یکی از پورت‌های ریزپردازنده وصل کنیم. محتوای کلیدها را باید به صورت یک ماتریس در ریزپردازنده ذخیره کنیم. به عنوان مثال ماتریس زیر را در نظر بگیرید، این ماتریس کلیدهای keypad را به محتوای نشان داده شده، نگاشت می‌کند.

```
char data_key[]={
'7','8','9','a',
'4','5','6','b',
'1','2','3','c',
'*','0','#','d'};
```

بعد از اتصال Keypad به ریزپردازنده، سطرهای R1 تا R4 را به صورت خروجی و ستون‌های C1 تا C4 را به صورت ورودی تعریف می‌کنیم. برای پیدا کردن کلید فشرده شده، باید سطر و ستون مربوط به آن مشخص شود. به این منظور

سطرها را به ترتیب ۱ می‌کنیم و مقدار ستون‌ها را می‌خوانیم. هر جا که مقدار ستونی ۱ شده باشد، یعنی کلید مربوط به آن سطر و ستون فشرده شده است. فرض کنید کلید ۵ فشرده شده باشد، در این صورت با یک شدن سطر مربوطه، ستون این کلید نیز مقدار ۱ را بر می‌گرداند و سایر ستون‌ها مقدار ۰ برمی‌گردانند. با پیدا شدن سطر و ستون، می‌توان از طریق ماتریس ذخیره شده در ریزپردازنده، به محتوای کلید دست پیدا کرد.

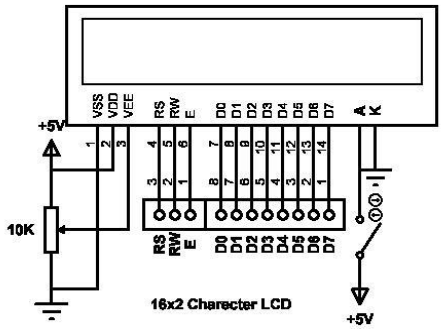


شکل ۸: الگوریتم کار کردن صفحه‌کلید ماتریسی

معرفی LCD کاراکتری:

یک عدد LCD کارکتری از نوع 16x2 (دارای ۱۶ ستون و ۲ ردیف) با نور زمینه‌ی به رنگ سبز یا آبی تحت بلوکی با عنوان 16x2 Character LCD به منظور نمایش جملات و اطلاعات دلخواه در این مجموعه ی آموزشی قرار داده شده است.

همانطور که در شکل زیر مشاهده می‌شود کاربر توسط پتانسیومتر می‌تواند کنتراست نمایشگر را در حد مطلوب تنظیم نماید. همچنین توسط یک عدد کلید کشویی می‌توان نور زمینه‌ی نمایشگر را وصل و یا قطع نمود.



شکل ۹: پایه‌های LCD

آشنایی با پایه‌ها:

پایه‌ی K و A برای تنظیم نور زمینه تعبیه شده‌اند. پایه های D0 تا D7 پایه‌های انتقال داده هستند. پایه‌ی E فعال‌ساز لچ داخلی است که برای ذخیره‌ی اطلاعات در LCD لازم است یک سیگنال با لبه‌ی پایین رونده به آن اعمال شود. پایه‌ی R/W مشخص می کند که باید اطلاعات از LCD خوانده یا روی آن نوشته شود. پایه‌ی RS برای مشخص کردن داده یا دستور بودن کد قرار گرفته روی D0 تا D7 استفاده می‌شود، در صورتی که RS=0 باشد کد به عنوان دستور و در صورتی که RS=1 باشد کد به عنوان داده (کد اسکی) تلقی می‌شود. پایه‌های VEE، VDD و VSS به ترتیب تنظیم کننده‌ی Contrast ، منبع تغذیه و زمین هستند.

ارتباط ریزپردازنده و LCD را می توان به دو صورت برقرار کرد:

- ارتباط ۴ سیمه : سریال
- ارتباط ۸ سیمه : موازی

ارتباط ۴ سیمه از جهت اشغال کردن پایه کمتر بهتر است و معمولاً از آن استفاده می‌شود. کتابخانه alcd.h نیز طوری طراحی شده است که تنها از روش ۴ سیمه استفاده کند.

مراحل ارتباط ۴ سیمه در جدول ۲ نشان داده‌شده است:

جدول ۲: مراحل لازم برای نشان دادن داده بر LCD در حالت ۴ سیمه

Step No.	Instruction RS	R/W	DB7	DB6	DB5	DB4	Display	Operation
1	Power supply on (the HD44780U is initialized by the internal reset circuit)							Initialized. No display.
2	Function set	0	0	0	1	0		Sets to 4-bit operation. In this case, operation is handled as 8 bits by initialization, and only this instruction completes with one write.
3	Function set	0	0	0	1	0		Sets 4-bit operation and selects 1-line display and 5 × 8 dot character font. 4-bit operation starts from this step and resetting is necessary. (Number of display lines and character fonts cannot be changed after step #3.)
		0	0	0	*	*		
4	Display on/off control	0	0	0	0	0	_	Turns on display and cursor. Entire display is in space mode because of initialization.
		0	0	1	1	0		
5	Entry mode set	0	0	0	0	0	_	Sets mode to increment the address by one and to shift the cursor to the right at the time of write to the DD/CGRAM. Display is not shifted.
		0	0	0	1	0		
6	Write data to CGRAM/DDRAM	1	0	0	1	0	H_	Writes H. The cursor is incremented by one and shifts to the right.
		1	0	1	0	0		

Note: The control is the same as for 8-bit operation beyond step #6.

همانطور که در مرحله ششم مشاهده می‌شود برای نمایش کاراکتر "H" در ابتدا عدد "0100" و سپس عدد "1000" بر روی باس داده قرارداده شده است. این مقدار از روی جدول ۳ تعیین می گردد.

جدول ۳

Upper 4 Bits / Lower 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			0	@	P	`	p				—	タ	ミ	α	p
xxxx0001	(2)		!	1	A	Q	a	q			。	ア	チ	ム	ä	q
xxxx0010	(3)		"	2	B	R	b	r			「	イ	ツ	メ	ß	θ
xxxx0011	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
xxxx0100	(5)		$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	σ	ü
xxxx0110	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)		'	7	G	W	g	w			ァ	キ	ヌ	ラ	q	π
xxxx1000	(1)		(	8	H	X	h	x			ィ	ク	ネ	リ	√	x̄
xxxx1001	(2)		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	-¹	y
xxxx1010	(3)		*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
xxxx1011	(4)		+	;	K	[	k	{			ォ	サ	ヒ	ロ	※	万
xxxx1100	(5)		,	<	L	¥	l	|			ャ	シ	フ	ワ	¢	円
xxxx1101	(6)		-	=	M	]	m	}			ュ	ス	ヘ	ン	£	÷
xxxx1110	(7)		.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
xxxx1111	(8)		/	?	O	_	o	←			ッ	ソ	マ	°	ö	■

در این آزمایشگاه برای اتصال LCD به یکی از پورت‌های ریزپردازنده، به عنوان مثال پورت A به صورت زیر عمل می کنیم:

پایه‌ی ریزپردازنده	پایه‌ی LCD
PA.0	RS
PA.1	E
PA.3	DB0
PA.4-PA.7	DB4-DB7

آشنایی با هدرفایل "alcd.h"

این هدرفایل حاوی دستورات آماده برای کار با LCD است که در نرم افزار CodeVision نوشته شده است. برخی از این توابع و نحوه‌ی استفاده از آن‌ها در زیر معرفی شده است:

- lcd_init() : این تابع تعداد ستون های LCD را به عنوان آرگومان ورودی دریافت می کند و مکان نمای چاپ را به خانه (0,0) می برد و همچنین مکان‌نمای چاپگر را غیر فعال می‌کند. در صورتی که ریزپردازنده ماژول LCD را تشخیص دهد این تابع مقدار ۱ را بر می گرداند در غیر این صورت مقدار صفر را بر می گرداند.

```
unsigned char lcd_init(unsigned char lcd_columns)
```

- lcd_clear() : محتوای LCD را پاک می کند.

```
void lcd_clear(void)
```

- lcd_ready() _ : کد را منتظر آماده شدن LCD نگه می دارد.

```
void _lcd_ready(void)
```

- lcd_putsf() : رشته ای که در داخل پرانتز قرار دارد را بر روی LCD نمایش می‌دهد. توجه کنید که وقتی از این تابع استفاده می کنیم که رشته str درون flash قرار داشته باشد.

```
void lcd_putsf(char flash *str)
```

- lcd_putchar() : کاراکتری که در داخل پرانتز قرار دارد را بر روی LCD نمایش می‌دهد.

```
void lcd_putchar(char c)
```

وتوابع زیر که در طول آزمایش توضیح داده خواهند شد.

- lcd_puts()

```
void lcd_puts(char *str)
```

- lcd_gotoxy(x,y)

```
void lcd_gotoxy(unsigned char x, unsigned char y)
```

پیش گزارش آزمایش دوم

نام و نام خانوادگی: **شماره دانشجویی:**

---

۱) آیا می توان عمل خواندن و نوشتن روی یک پورت را بلافاصله پشت سر هم انجام داد؟ به قسمت I/O Ports در دیتاشیت ریزپردازنده مراجعه نمایید.

۲) آزمایش‌های ۲-۳ و ۲-۴ را با استفاده از نرم‌افزار Proteus شبیه‌سازی کنید. (کد نوشته شده به زبان C شماتیک مدار و اطلاعات مربوط به Simulation log را در فایل پیش‌گزارش قرار دهید)

دستور کار

قسمت اول:آشنایی با LCD

آزمایش ۲-۱

اتصالات LCD را مطابق با آنچه که در قسمت مقدمه گفته شد، برقرار کرده و سپس برنامه‌ای بنویسید که نام خود را در سطر اول و شماره دانشجویی را در سطر دوم چاپ نماید.

قسمت دوم: کار با صفحه‌کلید ماتریسی

آزمایش ۲-۲

مطابق با آنچه که در مقدمه گفته شد، برنامه‌ای بنویسید که با فشردن هر کلید از صفحه کلید ماتریسی، عدد مربوطه روی LCD نمایش داده شود.

آزمایش ۲-۳

برنامه‌ای بنویسید که با دریافت الگوی 1234# ، LEDهای سبزرنگ روشن شده و در غیر این صورت LEDهای قرمز رنگ روشن شوند.

قسمت سوم: کار با وقفه‌ها

آزمایش ۲-٤

با اتصال دو عدد کلید فشاری به وقفه‌های خارجی ۰ و ۱، برنامه‌ای بنویسید که با فشردن کلید اول، شمارنده افزایش پیدا کرده و با فشردن کلید دوم شمارنده کاهش پیدا کند و مقدار شمارنده را نیز روی LCD نمایش دهد.

آزمایش سوم

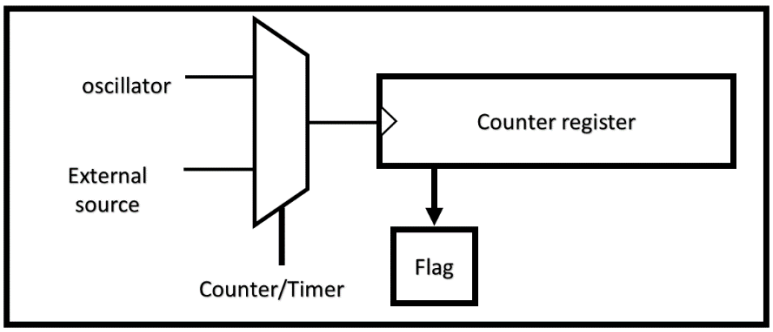
اهداف:

- آشنایی با وقفه‌ی داخلی ریزپردازنده
- آشنایی با تایمر

مقدمه

معمولاً تعداد زیادی از کارهایی که به طور روزمره با آنها سروکار داریم نیازمند اندازه‌گیری زمان یا شمارش یک اتفاق هستند. برای پیاده‌سازی چنین عملیات‌هایی با ریزپردازنده‌ها رجیسترهای تایمر در ریزپردازنده تعبیه شده‌اند. تایمرها[5] شمارنده‌هایی هستند که در صورت فعال بودن به طور موازی با پردازش‌های CPU عمل شمارش را انجام داده و شمارنده مربوط به آن‌ها افزایش می‌یابند. تایمرها مهمترین ابزار برای زمان‌سنجی در ریزپردازنده‌ها می‌باشند. ساده‌ترین مثال استفاده از تایمر به عنوان یک ثانیه شمار است که گام‌به‌گام مراحل آن را بررسی می‌کنیم.

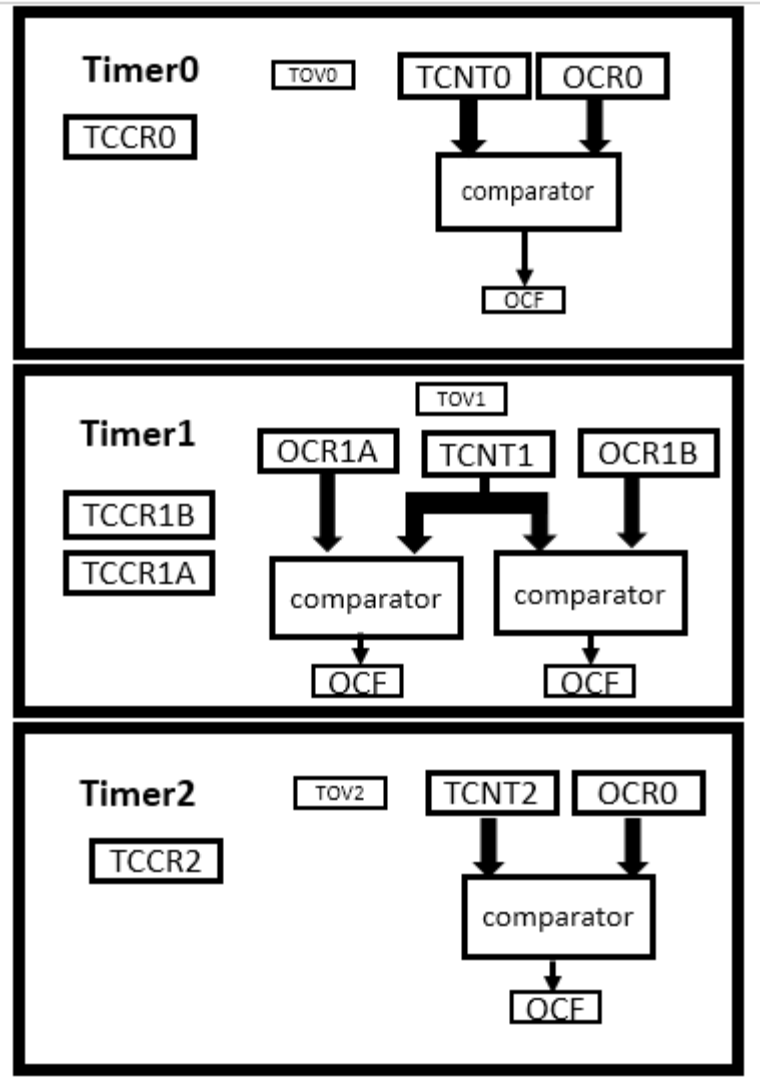
منبع شمارش تایمر می‌تواند کلاک معمولی CPU یا یک پالس خارجی باشد، که درصورتی که منبع پالس خارجی باشد از تایمرها به عنوان Counter یعنی شمارنده پالس‌های خارجی ریزپردازنده استفاده می‌شود. اساس کار تایمر شمارش با هر پالس کلاک است. به طور مثال یک ریزپردازنده با فرکانس کلاک 1MHz را در نظر بگیرید در ریزپردازنده محتوای رجیستر Counter در هر یک میکروثانیه یک بار یک عدد افزایش می‌یابد پس اگر بخواهیم یک delay به اندازه ۱۰۰ میکروثانیه ایجاد کنیم در ابتدا باید محتوا را ۱ کرده و تا زمانی که مقدار رجیستر برابر ۱۰۰ شود منتظر بمانیم.



شکل ۱۰

ریزپردازنده ATmega16 سه تایمر دارد که تایمرهای صفر و ۲، هشت بیتی و تایمر ۱ شانزده بیتی است. تعداد بیت‌های ذکرشده مشخص‌کننده حد بالای شمارنده تایمر هستند. مثلاً در تایمر ۸ بیتی شمارنده تایمر می‌تواند تا ۲۵۵ بشمارد. برای هر تایمر یک flag وجود دارد که زمانی که مقدار آن تایمر سرریز می‌کند این flag، set می‌شود.

[5] Timer



شکل ۱۱

برای استفاده از تایمرها نیاز است که با ثبات‌های آن آشنا شوید:

ثبات‌های اصلی تایمرها:

تنظیم تایمرها و استفاده از آن‌ها به وسیله‌ی ثبات‌های مربوطه انجام می‌شود. این ثبات‌ها در زیر معرفی شده‌اند. حرف n شماره‌ی تایمر را مشخص می‌کند و در ریزپردازنده ATmega16 می‌تواند صفر، یک یا دو باشد. پارامترهایی که در تنظیم تایمر موثرند، عبارتند از:

Clock Source: منبع کلاک تایمر می‌تواند، کلاک داخلی ریزپردازنده باشد یا اینکه تایمر به یک شمارنده تبدیل شده و از یکی از پایه‌های ریزپردازنده برای شمارش استفاده شود (در این حالت با آمدن هر پالس به پایه‌ی مشخصی از ریزپردازنده، مقدار آن یک واحد زیاد می‌شود).

Clock Value: در صورتی که برای تایمرها، Clock Source از نوع System Clock انتخاب شود، فرکانس تایمر توسط Clock Value تعیین می‌شود. با این کار فرکانس تایمر، مضربی از فرکانس ریزپردازنده خواهد بود (با ضریب کمتر از ۱).

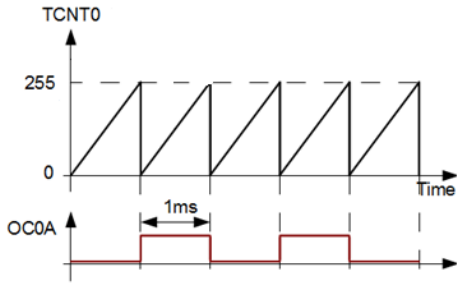
این فرآیند Pre-Scale نامیده می‌شود. در حالتی که در تنظیمات Clock Source تایمر را به Counter تبدیل کنیم تنها Timer2 قابلیت Pre-Scale کردن را خواهد داشت.

برای بدست آوردن فرکانس تایمر در حالت Pre-Scale (با مضرب N) از فرمول زیر استفاده می‌شود:

$$F_{Timer-Clock} = \frac{F_{Oscillator}}{N} \qquad \text{N = Pre-scale=1,8,64,256,1024}$$

- **Mode**: تایمرها می‌توانند شمارش را به انواع مختلف و تا مقادیر مختلف انجام دهند. چهار مد قابل استفاده عبارتند از Normal، CTC، Fast PWM و Phase Correct PWM. هر کدام از این مدها در ادامه توضیح داده شده‌اند.
  - در مد Normal Top شمارش تایمر تا بالاترین مقداری که تایمر می‌تواند بشمارد ادامه می‌یابد (برای تایمر هشت بیتی تا ۲۵۵ و برای تایمر شانزده بیتی تا ۶۵۵۳۵) و بعد دوباره از صفر شروع به شمارش می‌کند. در این حالت فرکانس موج تولید شده (OC0A) برای شمارنده x بیتی از فرمول زیر استفاده می‌شود:

$$F_{Generatedwave} = \frac{F_{Timer-Clock}}{2^{x+1}}$$



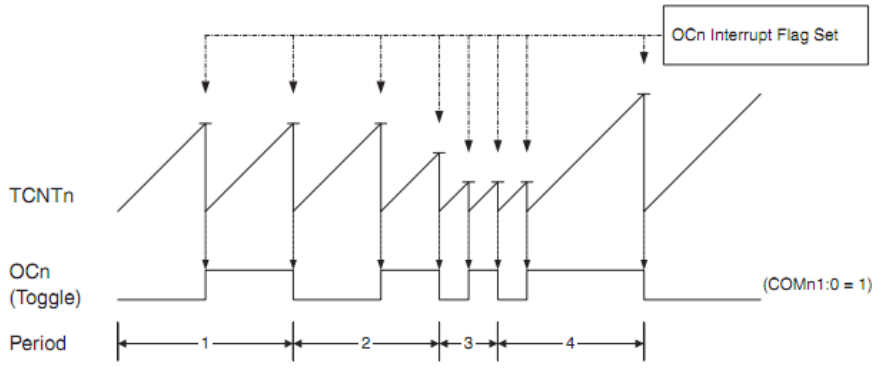
شکل ۱۲- نحوه‌ی شمارش و فعال شدن بیت OC0A در حالت شمارش Normal

همانطور که در شکل مشاهده می‌شود مقدار TCNT0 با OCR0A مقایسه شده و خروجی آن تحت عنوان OC0A تولید می‌شود.

در شکل فوق ضریب وظیفه (Duty Cycle) برابر با ۵۰ درصد است. چون مدت زمان ۱ و صفر بودن خروجی در یک دوره تناوب یکسان است. در حالت کلی ضریب وظیفه طبق رابطه‌ی زیر محاسبه می‌شود:

$$\text{ضریب وظیفه} = \frac{\text{زمان یک بودن پالس}}{\text{زمان کل یک دوره تناوب}}$$

- در مد CTC Top زمانی که تایمر به رجیستر OCRxy رسید (که در آن x شماره‌ی تایمر و y یکی از حروف A یا B است)، مقدار تایمر صفر شده و دوباره از اول شروع به شمارش می‌کند.



شکل ۱۳ – تعیین حد بالای شمارش با استفاده از ثبات OCRxy

در این شکل مقدار OCRn توسط کاربر تغییر می‌کند.

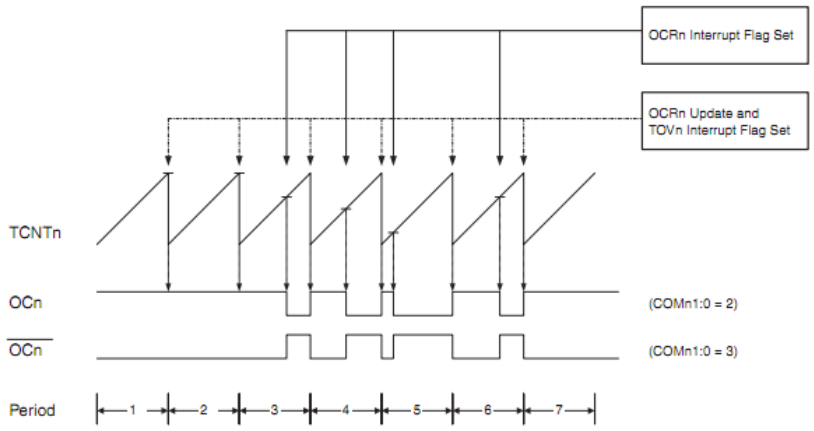
هر بار که مقدار TCNTn به OCRn برسد وقفه‌ی OCn Flag Set اتفاق می‌افتد و مقدار OCn تغییر می‌کند.

- در مد Fast PWM شمارنده مانند مد نرمال می‌شمارد و مقدار ثبات TCNTn همواره با ثبات OCRn مقایسه می‌شود. زمانی که محتوای این دو ثبات با هم برابر گردد (در حالت Non-inverted) سطح ولتاژ پایه OCn، صفر(low) می‌شود و زمانی که مقدار TCNTn به حداکثر مقدار خود برسد سطح ولتاژ پایه OCn، یک (high) می‌شود. به این صورت می‌توانیم یک موج PWM در پایه‌ی OCn تولید کنیم. دقت شود که برای این کار باید پایه‌ی OCn بصورت خروجی تعریف شود.

به طور کلی قابلیت مدهای PWM این است که امکان برنامه ریزی دوره تناوب و ضریب وظیفه را به ما می‌دهد. در نتیجه CPU برای کارهای دیگر آزاد می‌شود. برای محاسبه‌ی فرکانس موج PWM تولید شده می‌توان از فرمول زیر استفاده نمود:

$$F_{\mathrm{PWM}} = \frac{\mathrm{F}_{Timer-Clock}}{\mathrm{X}}$$

X=256 – مقدار بارگذاری شده در تایمر



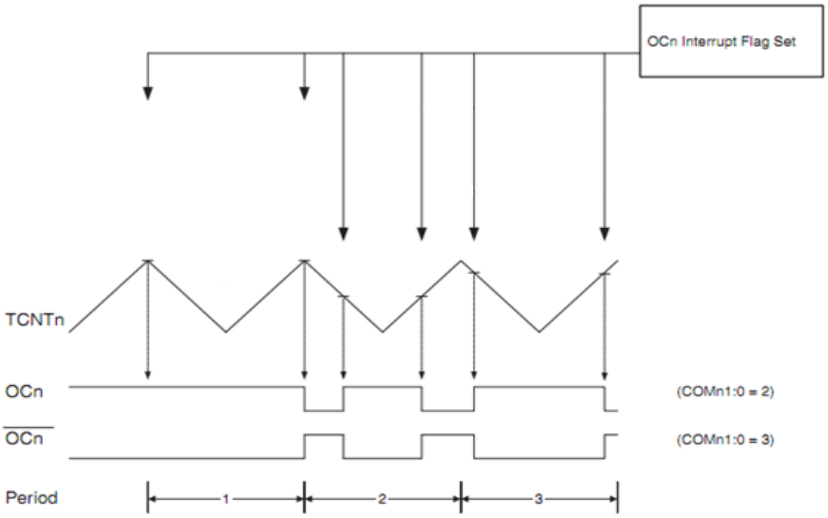
شکل ۱۴ – نحوه‌ی فعال شدن OCn در حالت Fast PWM

در مد Phase Correct PWM تایمر ابتدا تا رسیدن به حد بالا می‌شمارد. سپس تا رسیدن به صفر رو به پایین شمارش می‌کند. هرموقع که صفر شد پرچم TOVn یک می‌شود. در این مد نیز همواره ثبات TCNTn با ثبات OCRn مقایسه می‌شود و در ازای حالتی که در آن قرار دارد (inverted یا non-inverted) می‌تواند یک موج PWM برای ما تولید کند. فرکانس PWM در این مد از رابطه‌ی زیر محاسبه می‌شود:

$$F_{\mathrm{PWM}} = \frac{\mathrm{F}_{oscillator}}{\mathrm{N}*510}$$

N=1,8,64,256,1024

عدد ۵۱۰ بدلیل شمارش از ۰ تا ۲۵۵ و از ۲۵۵ تا ۰ بدست آمده است.



شکل ۱۴ – نحوه فعال شدن پایه‌ی OCn در حالت Phase Correct PWM

Output: تایمرها می‌توانند بر روی پایه‌های مشخصی از خروجی با اسامی OCx که x در آن شماره‌ی تایمر است پالس‌هایی را به طور خودکار ایجاد کند. این قابلیت برای زمانی که پالس‌های بسیار دقیق مورد نیاز است استفاده می‌شود.

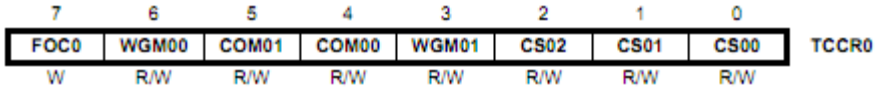
Input Capt: برای شمارش بر اساس تعداد وقفه‌های وارده به ریزپردازنده استفاده می‌شود.

Interrupt on: در این بخش مشخص می‌کنیم که تایمر در چه صورت و چه نوع وقفه‌هایی را ایجاد کند.

Value: در این پارامتر مقدار اولیه‌ی تایمر مشخص می‌شود. مثلا در مد CTC Top از این مقدار تا مقدار OCRxy شمارش انجام می‌شود.

Compare(A…B…): در این بخش می‌توان مقادیر رجیسترهای OCRxy را مشخص کنیم. این‌ها مقادیری هستند که بعداً برای تولید پالس‌ها یا مد CTC Top و یا مدهای مربوط به PWM مورد استفاده تایمر قرار می‌گیرند. تایمر همواره خود را با مقادیر این رجیسترها مقایسه می‌کند تا در صورت برابر شدن با آن‌ها عمل مشخصی را انجام دهد. دقت شود تایمر توسط کلاکی که به آن وارد می‌شود افزایش می‌یابد و حتی اگر با یکی از این دو مقدار (مقداری که ما مشخص کردیم) برابر شد همچنان به این شمارش ادامه می‌دهد. اما می‌تواند با مشاهده‌ی برابری خودش با این رجیسترها وقفه‌هایی را ایجاد کند.

به عنوان مثال برای تنظیم timer0 در ATmega16 از اطلاعات جدول‌های زیر می‌توان استفاده کرد. برای سایر تایمرها، اطلاعات لازم برای تنظیم ثبات‌ها در datasheet آمده است.



شکل ۱۵ - معرفی بیت‌های ثبات کنترلی Timer 0

در شکل ۱۵ رجیستر TCCR0 مشخص شده است. نحوه‌ی کار با بیت‌های این ثبات در جدول‌های زیر آمده است.

جدول ۴- تنظیم مد کاری تایمر با استفاده از بیت‌های WGM00 و WGM01

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	BOTTOM	MAX

جدول ۵- تنظیم نحوه‌ی فعال شدن پایه‌ی OC0 در حالت غیر PWM با استفاده از بیت‌های COM00 و COM01

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Toggle OC0 on compare match
1	0	Clear OC0 on compare match
1	1	Set OC0 on compare match

جدول ۶- تنظیم نحوه‌ی فعال شدن پایه‌ی OC0 در حالت Fast PWM با استفاده از بیت‌های COM00 و COM01

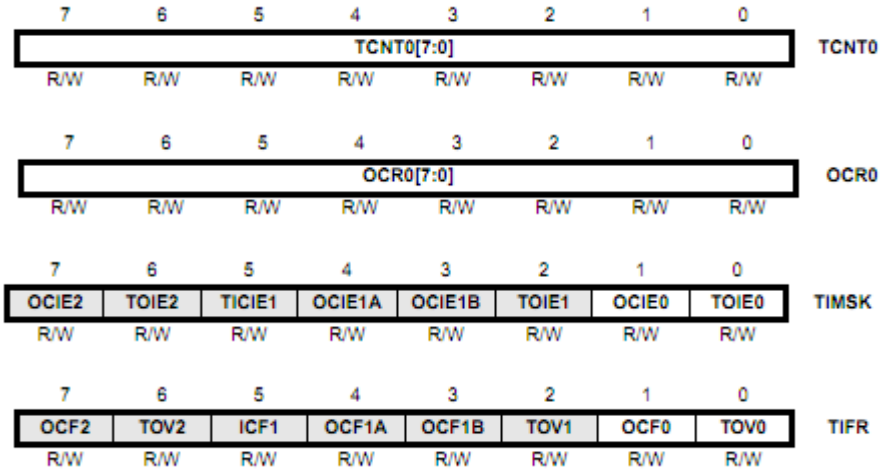
COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set OC0 at BOTTOM, (non-inverting mode)
1	1	Set OC0 on compare match, clear OC0 at BOTTOM, (inverting mode)

جدول ۷- تنظیم نحوه‌ی فعال شدن پایه‌ی OC0 در حالت Phase Correct PWM با استفاده از بیت‌های COM00 و COM01

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match when up-counting. Set OC0 on compare match when downcounting.
1	1	Set OC0 on compare match when up-counting. Clear OC0 on compare match when downcounting.

جدول ۸- تنظیم فرکانس شمارنده با استفاده از بیت‌های CS00 و CS01 و CS02

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.



شکل ۱۶- معرفی بیت‌های ثبات‌های TCNT0، OCR0، TIMSK، TIFR

هر تایمر دارای دو وقفه‌ی سرریز و تطبیق مقایسه‌ی خروجی است که به ترتیب می‌توانند با استفاده از بیت‌های TOIEn و OCIEn از ثبات TIMSK فعال یا غیر فعال شوند.

همچنین هر تایمر دارای دو پرچم سرریز (TOVn) و تطبیق مقایسه‌ی خروجی (OCFn) است که در ثبات TIFR قرار دارند.

بیت TOV0 در صورتی که برابر یک باشد، به این معناست که timer0 سرریز دارد. بیت OCF0 پرچم تطبیق مقایسه خروجی است که در صورت یک بودن به معنای تطبیق نتیجه مقایسه است.

ثبات TCNTn : این ثبات یک شمارنده است. به محض ریست شدن، محتوای آن صفر می‌شود و با هر پالسی که به آن وارد می‌شود یکی می‌شمارد. محتوای تایمر شمارنده با استفاده از TCNTn قابل دسترسی است. می‌توان مقداری را روی آن قرار داد یا از روی آن خواند.

پرچم TOVn : وقتی یک تایمر سرریز می کند محتوای این پرچم یک می‌شود.

ثبات TCCRn : برای تنظیم مدهای عملیاتی استفاده می‌شود.

پرچم OCFn : پرچم مقایسه‌ی خروجی است.

ثبات OCRn : محتوای این ثبات با محتوای TCNTn مقایسه می‌شود. اگر با هم برابر باشند پرچم OCFn یک خواهد شد.

ثبات TIMSK : بیت‌های فعالساز وقفه‌ی تایمر صفر، یک و دو را در خود دارد.

ثبات TIFR : حاوی پرچم‌های تایمرهای مختلف است.

مثال کاربردی :

اکنون برنامه ثانیه شمار را بررسی می کنیم این برنامه تا ۶۰ ثانیه می‌شمارد و آن را روی LCD نشان می‌دهد. در ابتدا با باز کردن پنجره Code Wizard و انتخاب ATmega16 در بخش chip، در لبه Timer، ۳ تایمر میکرو مشاهده می‌شود.Timer1 را انتخاب می‌نماییم.

در قسمت clock source، می‌توان منبع کلاک تایمر را تنظیم نمود که داخلی یا خارجی (لبه‌ی پایین رونده یا بالا رونده پایه T1) باشد. درصورت انتخاب منبع داخلی در قسمت clock value ، می‌توان کلاک سیستم یا تقسیمی از آن را به عنوان کلاک تایمر انتخاب نمود. تغییر در این قسمت باعث تغییر ضرایب prescale و در نهایت تغییر بیت‌های CS در رجیستر TCCRn می‌شود. در قسمت بعد می‌توان یکی از ۱۶ مد عملیاتی تایمر/کانتر را انتخاب نمود. اگر مد انتخاب شده PWM نباشد در قسمت out.A و out.B می‌توان خروجی را در ۴ وضعیت مختلف تنظیم نمود: قطع، toggle، صفر شدن سطح منطقی[6] یا یک شدن سطح منطقی[7] در زمانی که مقدار OCR و counter reg مساوی می‌شوند.

در حالت PWM خروجی می‌تواند ۳ وضعیت داشته باشد : disconnected، non-inverted و inverted

گزینه قابل تنظیم بعدی که مشاهده خواهید کرد input capt است که مخصوص مد capture می‌باشد.

تایمر/کانتر یک، دارای ۴ نوع وقفه است که در صورتی که تیک interrupt on زده شود فعال می‌شوند. این ۴ وقفه عبارتند از: سرریز، تطابق مقایسه A و B و input capture.

[6] clear
[7] set

برای مقدار دهی اولیه به رجیسترهای TCNT و OCR باید مقادیر مورد نظر را در قسمت‌های compA،value و compB وارد کرد.

```
#include <mega16.h>
#include <alcd.h>
#include <stdio.h>

int i=0,j=0,k=0;
char str[10];

interrupt [TIM1_COMPA] void timer1_compa_isr(void)
{
     i++;
     if( i==60 )
     {
        i=0;
     }
     sprintf(str,"%d ",i);
     lcd_clear();
     lcd_puts(str);
}
void main(void)
{
     TCCR0=0x0C;
     TCCR1B=0x0C;
     OCR1AH=0x7A;
     OCR1AL=0x12;
     TIMSK=0x10;
     // Global enable interrupts
     #asm("sei")
     lcd_init(16);
     lcd_clear();
     while(1);
}
```

پیش گزارش آزمایش سوم

نام و نام خانوادگی: **شماره دانشجویی:**

---

۱) آزمایش‌های ۳-۱ و ۳-۳ را با استفاده از نرم‌افزار Proteus شبیه‌سازی کنید. (کد نوشته شده به زبان C شماتیک مدار و اطلاعات مربوط به Simulation log را در فایل پیش‌گزارش قرار دهید)

دستور کار

آزمایش ۳-۱

در آزمایش مربوط به 7-segment گفتیم برای نمایش اعداد مختلف روی 7-segmentهای مالتی‌پلکس‌شده باید از روش Refreshing استفاده کنیم. یکی از روش‌های این کار استفاده از تابع delay می‌باشد. اشکالی که در این روش وجود دارد هنگامی که در یک برنامه بخش‌های مختلفی وجود دارد استفاده از delay برنامه را متوقف کرده و این وضعیت باعث از کار افتادن بخش‌های دیگر برنامه می‌شود.

بنابراین بهترین وسیله برای پیاده‌سازی این روش استفاده از تایمرها است. در این آزمایش قصد داریم تابع 7-segment را با استفاده از تایمر بنویسیم.

ثانیه‌شمار دو رقمی طراحی کنید و نتیجه را روی نمایشگر 7-segment نشان دهید.

آزمایش ۳-۲

در این آزمایش قصد داریم صدایی با فرکانس متغیر بر روی speaker داشته باشیم. خروجی یکی از تایمرها را به ورودی speaker متصل کرده و برنامه‌ای بنویسید که صوت‌هایی با فرکانس‌های مختلف تولید کرده و از speaker پخش کند. فرکانس ابتدا به تدریج افزایش پیدا کرده و سپس کاهش پیدا می‌کند.

آزمایش ۳-۳

برنامه‌ای بنویسید که فاصله زمانی فشرده شدن دو عدد کلید فشاری را اندازه‌گیری کرده و نتیجه را بر حسب میلی‌ثانیه روی LCD نمایش دهد.

آزمایش چهارم

اهداف:

- آشنایی با رله
- آشنایی با موتور DC
- آشنایی با شفت انکودر
- آشنایی با نحوه عملکرد موتور پله‌ای

مقدمه

آشنایی با رله

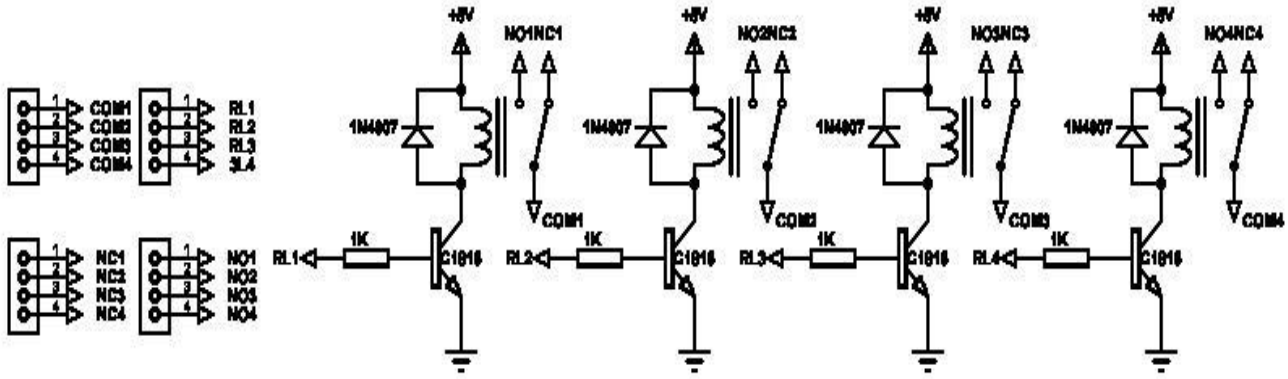
رله یک سوییچ کنترلی الکتریکی سریع یا بی‌درنگ است که با هدایت یک مدار الکتریکی دیگر باز و بسته می‌شود. این سوییچ که به طور گسترده در کنترل صنعتی، اتومبیل و بسیاری از وسایل دیگر استفاده می‌شود، اجازه ایزوله کردن دو بخش مجزا از یک سیستم برای استفاده از دو منبع ولتاژ متفاوت را می‌دهد. یک مدل از انواع رله مدل الکترومغناطیسی (EMR[8]) است که در شکل ۱۷ نشان داده شده است.

شکل ۱۷: رله

چهار عدد رله‌ی تک‌کنتاکت به منظور قطع و وصل کردن خطوط مختلف در بلوکی با عنوان Relay مطابق با شماتیک شکل ۱۸ در این مجموعه آموزشی قرار داده شده است. بلوک رله متشکل از ترانزیستور، رله و دیود می‌باشد. با اعمال ولتاژ ۵ ولت به RLx سیم‌پیچ درون رله فعال شده و مسیر عبور جریان را به سمت NOx تغییر می‌دهد. به

[8] ElectroMagnetic Relay

عنوان مثال برای کنترل موتور می‌توان پایه‌ی COMx را به ولتاژ ۵ ولت و یکی از پایه‌های NOx یا NCx را به پایه CW موتور وصل کرد. در نهایت با متصل کردن پایه‌ی RLx به یکی از پایه‌های ریزپردازنده می‌توان موتور را روشن یا خاموش نمود (در ادامه در مورد موتور الکتریکی توضیح داده خواهد شد).



شکل ۱۸: ۴ عدد رله مطابق بر برد آموزشی

جدول ۹

RL	com	NO	NC
LOW	X	مدار باز	X
HI	X	X	مدار باز

آشنایی با موتور DC

موتور DC وسیله‌ای است که پالس‌های الکتریکی را به حرکت مکانیکی تبدیل می‌کند. در موتور DC فقط دو سر سیم + و – وجود دارد. با اتصال آنها به یک منبع ولتاژ DC موتور در یک جهت و با معکوس کردن اتصال سیم‌ها موتور در جهت مخالف خواهد چرخید. حداکثر سرعت یک موتور DC در datasheet برحسب rpm (دور بر دقیقه) نشان داده شده است. در محدوده‌ی ولتاژ کاری موتور هرچه ولتاژ اعمال شده به موتور را افزایش دهیم rpm نیز بیشتر می‌شود.

موتورهای DC دو ویژگی بسیار مهم دارند:

۱- سرعت موتور به وسیله ولتاژ اعمالی به دو سر آن تعیین می‌شود.

۲- گشتاور موتور به وسیله جریانی که می‌کشد تعیین می‌شود و مقدار جریان عبوری به وسیله‌ی مقدار بار تعیین می‌شود. وجود بار باعث کاهش سرعت موتور می‌شود. با یک ولتاژ ثابت، هنگامی که بار افزایش می‌یابد، جریان مصرفی موتور افزایش خواهد یافت. اگر به موتور بیش از حد بار اعمال کنیم، موتور متوقف شده و ممکن است به علت گرمای تولید شده توسط مصرف جریان بالا، به موتور آسیب وارد شود.

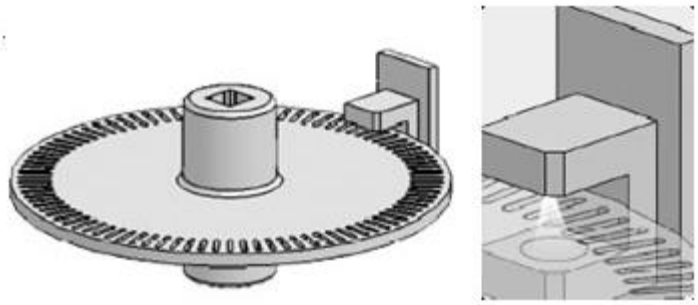
آشنایی با انکودر

انکودر حسگری است که به محور چرخ، چرخ دنده یا موتور وصل می‌شود و می‌تواند میزان چرخش را اندازه‌گیری کند. با اندازه‌گیری میزان چرخش می‌توانید جابه‌جایی، سرعت، شتاب یا زاویه چرخشی را تعیین کنید. انکودرها از نوع نوری یا لیزری می‌باشند و یک فرستنده و یک گیرنده امواج در دو سمت یک جسم مکانیکی چرخنده (دیسک شیاردار) قرار می‌گیرند و پالس‌های الکتریکی را تولید می‌کنند. به این صورت که اگر نور ارسالی توسط فرستنده از شیارهای چرخنده عبور کند توسط گیرنده دریافت می‌شود و مقدار ولتاژ خروجی یک می‌شود و زمانی که نور ارسالی به پره‌ها برخورد کند توسط گیرنده دریافت نمی‌شود و مقدار ولتاژ خروجی از گیرنده صفر می‌گردد به این ترتیب پالس‌های الکتریکی تولید می‌شود.

یک عدد موتور DC به همراه یک عدد انکودر لیزری در بلوکی باعنوان DC Motor + Encoder Rotation در این مجموعه آموزشی قرار داده شده است. کاربر با اعمال یک منطقی به پین CW می تواند موتور را در جهت عقربه های ساعت و پین CCW در جهت خلاف عقربه های ساعت موتور را به گردش درآورد.

با اعمال ولتاژ ۵ ولت به پایه‌ی CW موتور شروع به چرخش می‌کند. برای کنترل موتور با استفاده از ریزپردازنده به دلیل این که ریزپردازنده نمی‌تواند جریان کافی را برای چرخش موتور فراهم کند نیاز به یک سیستم تقویت جریان خواهیم داشت. برای تقویت این جریان از رله‌های تعبیه شده در این برد آموزشی که در قسمت بالا توضیح داده شد استفاده خواهیم نمود.

به منظور موقعیت سنجی شفت موتور یک عدد شفت انکودر لیزری ۳۶۰ پالسی روی شفت موتور بسته شده است. به این معنا که روی صفحه‌ی شفت ۳۶۰ شیار ایجاد شده است. برای کار با این انکودر ابتدا باید پایه‌ی Enable Encoder را فعال کنید. با چرخش موتور، روی پین Pulse Out پالس‌هایی متناظر با موقعیت شفت تولید می‌گردد. این Pulseها در اثر عبور نور لیزر از شیارها ایجاد می‌شوند.



شکل ۱۹

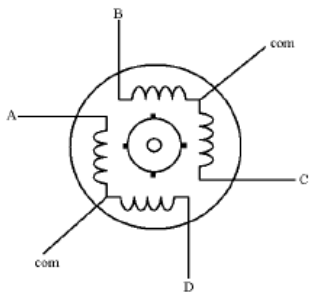
توجه: صفحه ی پره ای شکل قرار داده شده روی شفت به دقت تنظیم گردیده است. از این رو از تغییر مکان این صفحه روی شفت اکیداً خودداری نمایید.

کاربر برای استفاده از شفت انکودر می بایست با اعمال یک منطقی به پین Enable این سنسور را روشن و پالس های تولید شده را دریافت نماید.

موتور پله‌ای

موتور پله‌ای موتوری است که به ازای پالس‌های الکتریکی حرکت دورانی ایجاد می‌کند. در واقع یک موتور پله ای ترکیبی از یک موتور الکتریکی DC و یک سلونویید است. حرکت دورانی موتور دارای زاویه چرخش معینی است. از آنجایی که این موتورها می‌توانند در یک زاویه خاص قفل شوند کاربردهای گوناگونی برای آن‌ها وجود دارد. از موتورهای پله‌ای می‌توان برای جابجایی، حرکت، تعیین موقعیت و بسیاری از کارهای دیگر که در آن‌ها کنترل دقیق موقعیت یک محور، اهرم و .. مورد نیاز باشد استفاده کرد.

هر موتور پله‌ای دارای یک هسته متحرک مغناطیسی دائمی است که روتور یا شفت نام دارد و به وسیله یک بخش ثابت به نام استاتور احاطه شده است. در شکل زیر یکی از متداول‌ترین انواع موتورپله‌ای را مشاهده می‌کنید.



شکل ۲۰

این نوع موتورها دارای ۵ یا ۶ سیم می‌باشند که ۴ تای آن برای استاتور و ۲ تای آن پایه مشترک بوده و باید به VCC وصل شوند (در اکثر موتورها این دو سر وسط از داخل به هم وصل می‌شوند در نتیجه موتور دارای ۵ سیم می‌شود). نحوه‌ی عملکرد یک موتور پله‌ای تفاوت زیادی با یک موتور DC ندارد و تنها تفاوت در نحوه حرکت موتور است.

برای حرکت دادن موتورهای پله‌ای بایست پالس‌هایی با فواصل زمانی مختلف به پایه‌های چهارگانه آن‌ها اعمال کرد. ترتیب اعمال این پالس‌ها از پیش تعریف شده است. به عنوان مثال اگر پالس‌هایی مانند جدول زیر به هریک از سیم‌پیچ‌ها اعمال شود موتور درحالت از پیش تعیین شده‌ی تمام‌پله می‌چرخد:

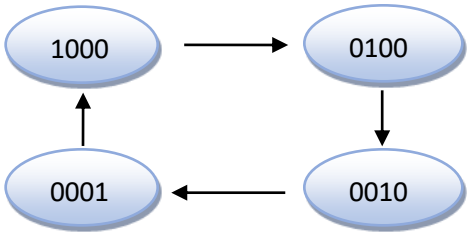
جدول ۱۰

جهت موتور	D	C	B	A
خلاف جهت عقربه های ساعت	1	0	0	0
	0	1	0	0
	0	0	1	0
	0	0	0	1

جهت موتور	D	C	B	A
در جهت عقربه های ساعت	0	0	0	1
	0	0	1	0
	0	1	0	0
	1	0	0	0

دقت کنید که ابتدا اطلاعات سطر اول جدول به چهارپایه اعمال می‌شود و پس از تأخیری سطر دوم و به همین ترتیب سطر سوم و چهارم وارد می‌شود و پس از آن دوباره باید اطلاعات سطر اول را به موتور اعمال کرد.

نمودار حالت برای مد چرخش تمام پله ساعتگرد به صورت زیر می‌باشد.



هنگامی که یک پالس به یکی از سیم‌پیچ‌ها اعمال شود، موتور به اندازه یک پله حرکت می‌کند. زاویه پله حداقل زاویه چرخش مربوط به یک پله است. این زاویه در موتورهای مختلف بین محدوده ۰.۷۲ تا ۹۰ درجه می‌باشد که متداول‌ترین زاویه پله ۱.۸ درجه است. براساس این زاویه پله باید تعداد پالس‌های مشخصی به موتور داده شود تا یک دور کامل بچرخد. آن‌ها را برای زاویه‌های مختلف در جدول ۱۱ مشاهده می‌شود:

جدول ۱۱

تعداد پله در یک دور	زاویه پله
500	0.72
200	1.8
48	7.5
24	15
4	90

برای راه‌اندازی موتورپله‌ای توسط ریزپردازنده به جریان دهی مناسب نیاز داریم پس باید از ترانزیستور یا IC های مخصوص استفاده کنیم. در این برد از درایور ULN2803A استفاده شده است.

یک عدد موتور پله ای ۶ سیمه با زاویه چرخش پله ۱۸ درجه به همراه یک عدد درایور ULN2803A در بلوکی با عنوان Stepping Motor در این مجموعه ی آموزشی قرار داده شده است.

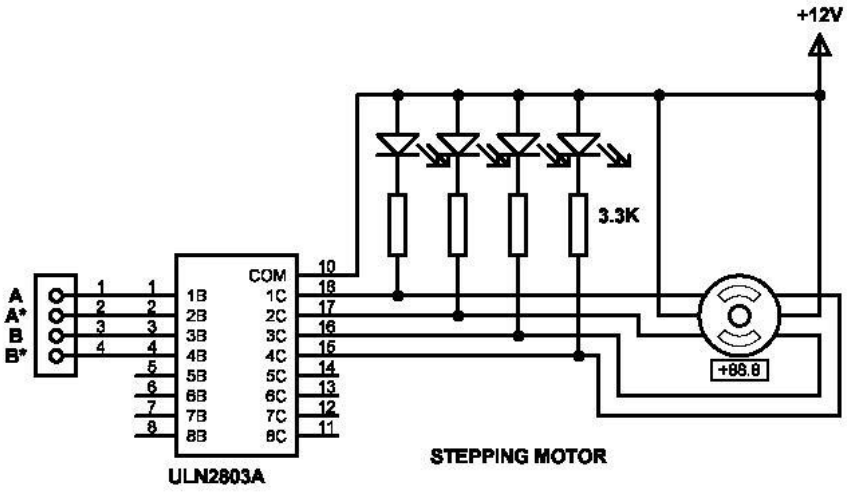
نکته: موتور مذکور با منبع تغذیه ۱۲ ولت بایاس شده است.

اطلاعات تولید شده توسط میکروکنترلر در ابتدا وارد درایور شده و پس از تقویت جریان به حد مطلوب، به سیم پیچ های استاتور موتور اعمال می شود.

۴ عدد LED به منظور نمایش اطلاعات تولید شده توسط ریزپردازنده و همچنین درک بهتر سرعت سوئیچینگ ریزپردازنده با سرعت چرخش موتور در این بلوک تعبیه شده است.

کاربر با معکوس نمودن اطلاعات ارسالی توسط ریزپردازنده می تواند جهت چرخش موتور را نیز کنترل نماید.

شماتیک مربوط به این بلوک در شکل زیر مشخص شده است.



شکل ۲۱: شماتیک داخلی موتور پله‌ای

پیش گزارش آزمایش چهارم

نام و نام خانوادگی: **شماره دانشجویی:**

---

۱) آزمایش‌های ۴-۲ و۴-۳ را با استفاده از نرم‌افزار Proteus شبیه‌سازی کنید. (کد نوشته شده به زبان C شماتیک مدار و اطلاعات مربوط به Simulation log را در فایل پیش‌گزارش قرار دهید)

دستور کار

آزمایش ٤-١

یکی از پایه‌های ریزپردازنده را به پایه RL یکی از رله‌ها متصل کرده و سپس پایه NO آن رله را به پایه CW موتور وصل کنید. و برنامه‌ای بنویسید که موتور ۰.۵ ثانیه بچرخد و ۰.۵ ثانیه متوقف باشد. این آزمایش را با تأخیرهای متفاوت انجام دهید و نتیجه را مشاهده کنید.

آزمایش ٤-٢

با استفاده از وقفه تایمر سرعت چرخش موتور DC را به دست آورده و نتیجه را روی LCD نمایش دهید. حداکثر خطا را ۵ دور در دقیقه در نظر بگیرید.

آزمایش ٤-٣

برنامه‌ای بنویسید که موتور پله‌ای به صورت ساعتگرد با سرعت ۷۰ دور بر دقیقه بچرخد.

آزمایش پنجم

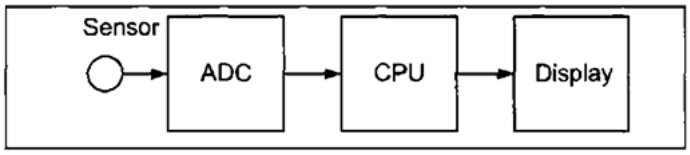
اهداف:

- آشنایی با نحوه‌ی تبدیل سیگنال آنالوگ به دیجیتال
- استفاده از مبدل ADC داخلی ریزپردازنده
- خواندن اطلاعات از سنسورهای آنالوگ
- کار با مبدل دیجیتال به آنالوگ

مقدمه

آشنایی با نحوه تبدیل سیگنال آنالوگ به دیجیتال

گاهی نیاز است که یک کمیت فیزیکی (مانند دما، شدت صدا و شدت نور و...) اندازه‌گیری شود، پردازنده‌های دیجیتال از مقادیر باینری و گسسته استفاده می‌کنند در حالیکه تمام المان‌های فیزیکی به صورت آنالوگ و پیوسته هستند. بنابراین برای پردازش کمیت‌های فیزیکی مختلف نیازمند یک تبدیل‌کننده آنالوگ به دیجیتال[9] برای تبدیل سیگنال‌های پیوسته به گسسته هستیم.



شکل ۲۲: اتصال یک ریزپردازنده به سنسور با استفاده از مبدل ADC

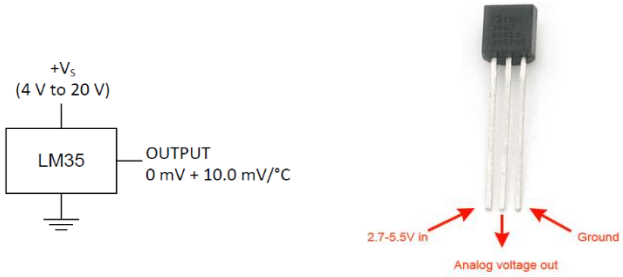
کمیت‌های فیزیکی با استفاده از یک مبدل[10] به کمیت‌های الکتریکی تبدیل می‌شوند از سنسورها می‌توان به عنوان مبدل‌های فیزیکی اشاره نمود. سنسورها دارای انواع مختلف از قبیل دما، فشار، نور و... می‌باشند.

سنسور ها مقدار یک کمیت آنالوگ را به ولتاژ یا جریان تبدیل می‌کنند، سپس این ولتاژ آنالوگ به مبدل آنالوگ به دیجیتال ریزپردازنده داده می‌شود و مبدل آنالوگ به دیجیتال مقدار ولتاژ را به کمیت دیجیتال متناظر تبدیل می‌کند سپس این مقدار دیجیتال با اعمال ریاضی به مقدار عددی متناظر تبدیل می‌شود و روی LCD یا 7segment نمایش داده می‌شود. یکی از مثال‌های بسیار کاربردی در این زمینه اندازه‌گیری دما با استفاده از سنسورهای اندازه‌گیری دما می‌باشد. در ادامه یک مثال تبدیل سیگنال آنالوگ به دیجیتال را بررسی می‌نماییم.

[9] Analog-to-digital converter
[10] transducer

در این مجموعه آموزشی دو عدد سنسور دما (LM35 وNTC)، یک عدد سنسور نور (CDS)، یک عدد سنسور رطوبت (HIS06)، یک عدد سنسور گاز شهری (TGS813) و یک ولوم ۱۰ کیلو اهم جهت شبیه سازی سنسور مولفه‌های محیطی دلخواه در بلوکی با عنوان Sensors قرار داده شده است.

فرض کنید بخواهیم دمای یک مکان را اندازه‌گیری کنیم. برای اینکار در ابتدا نیازمند یک سنسور جهت اندازه‌گیری دما هستیم. در این مثال از سنسور LM35 استفاده می‌کنیم. محدوده دمایی که این سنسور قادر به اندازه‌گیری آن می‌باشد بین -۵۵ تا ۱۵۰ درجه است و این سنسور به ازای هر درجه سانتی‌گراد ۱۰ میلی‌ولت ولتاژ خروجی را تغییر می‌دهد. یعنی به ازای دمای ۱ درجه، ولتاژ خروجی سنسور ۱۰ میلی‌ولت و به ازای دمای ۱۰۰ درجه، خروجی سنسور ۱ ولت می‌باشد.



شکل ۲۳: سنسور اندازه گیری دما LM35

این سنسور دارای ۳ پایه می‌باشد، در صورتی که سنسور روبروی شما باشد (بتوانید نوشته‌هایش را ببینید) اولین پایه سمت چپ VCC سنسور (متصل به ۵ ولت می‌شود)، پایه وسط ولتاژ خروجی (به ریزپردازنده متصل می‌شود) و پایه سوم GND سنسور است. این ولتاژ خروجی که به صورت آنالوگ است باید به مقادیر گسسته و دیجیتال تبدیل شود. پس نیازمند استفاده از مبدل آنالوگ به دیجیتال[11] ریزپردازنده هستیم. پس از تنظیم ADC ریزپردازنده می‌توان به راحتی مقدار دما را بر روی 7-Segment نشان داد.

[11] ADC

برخی از خصوصیات اصلی ADC

دقت تبدیل[12]

مبدل ADC با رزولوشن n-بیتی تعریف می‌شود که در ریزپردازنده ATmega16، n می‌تواند ۱۰ بیت یا ۸ بیت باشد. هرچه تعداد بیت دقت تبدیل بیشتر باشد ADC دارای step size کمتر است. Step Size یعنی کمترین مقدار تغییری که برای ADC قابل تشخیص است. اندازه Step Size را می‌توان با ولتاژ reference کنترل کرد.

$$Step\ Size = \frac{V_{ref}}{2^{n}}$$

ولتاژ مرجع[13]

V_{ref} یک ولتاژ ورودی برای مشخص نمودن محدوده ولتاژ آنالوگ و تنظیم اندازه Step Size است که می‌تواند بین ۵-۰ ولت باشد.

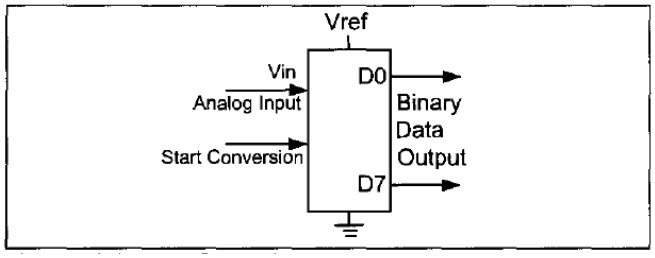
V_{ref} می‌تواند مقادیر AVCC یعنی ۵ ولت یا ولتاژ مرجع داخلی با مقدار ۲.۵۶ ولت و یا پایه‌ی خارجی AREF را اتخاذ نماید.

V_{ref} (V)	V_{in} Range (V)	Step Size (mV)
5.00	0 to 5	5/256 = 19.53
4.0	0 to 4	4/256 = 15.62
3.0	0 to 3	3/256 = 11.71
2.56	0 to 2.56	2.56/256 = 10
2.0	0 to 2	2/256 = 7.81
1.28	0 to 1.28	1.28/256 = 5
1	0 to 1	1/256 = 3.90

شکل ۲۴ ارتباط میان ولتاژ مرجع، ولتاژ ورودی و step

حداکثر ولتاژی که مبدل آنالوگ به دیجیتال می‌تواند اندازه بگیرد برابر با VCC است و اگر ولتاژ اعمالی از VCC بیشتر شود مبدل آنالوگ به دیجیتال آسیب می‌بیند (معمولاً بیشترین ولتاژ ورودی که به ADC اعمال می‌کنند ۴.۵ ولت است) و کمترین ولتاژ اعمالی برابر با GND است. ADC به ازای ولتاژ ۵ ولت در حالت ۱۰ بیتی عدد ۱۰۲۳ (و در حالت ۸ بیتی عدد ۲۵۵) و به ازای صفر ولت عدد صفر را در متغیر مربوطه قرار می‌دهد.

[12] resolution
[13] reference voltage



شکل ۲۵ بلوک دیاگرام ADC

ADC دارای دو منبع ولتاژ آنالوگ مجزا است. AVCC و AGND که AGND بایستی به زمین یا ولتاژ زمین آنالوگ متصل شود و AVCC نباید بیشتر از +/-۰.۳ نسبت به VCC اختلاف داشته باشد.

ولتاژ مرجع خارجی در صورت وجود باید به پایه AREF وصل شود که این ولتاژ بایستی بین ولتاژ موجود روی پایه‌های AGND – AVCC باشد. در غیر این صورت به VCC وصل می‌شود. ADC مقدار آنالوگ ورودی را با تقریبی که در ادامه خواهید دید به مقدار دیجیتال ۱۰ بیتی تبدیل می‌کند. کمترین مقدار، نشان دهنده مقدار آنالوگ موجود در پایه AGND و بالاترین مقدار نشان دهنده ولتاژ پایه AREF منهای یک LSB [14]است.

به این منظور ولتاژ سیگنال آنالوگ با ولتاژ ثابت V_{ref} (که قابل تنظیم است) مقایسه می‌شود. بسته به این که خروجی مبدل چند بیتی باشد از بازه‌ی (V_{ref}–0) ولتاژی به سیگنال ورودی نسبت داده می‌شود. مقدار سیگنال ورودی نباید از مقدار V_{ref} تجاوز کند.

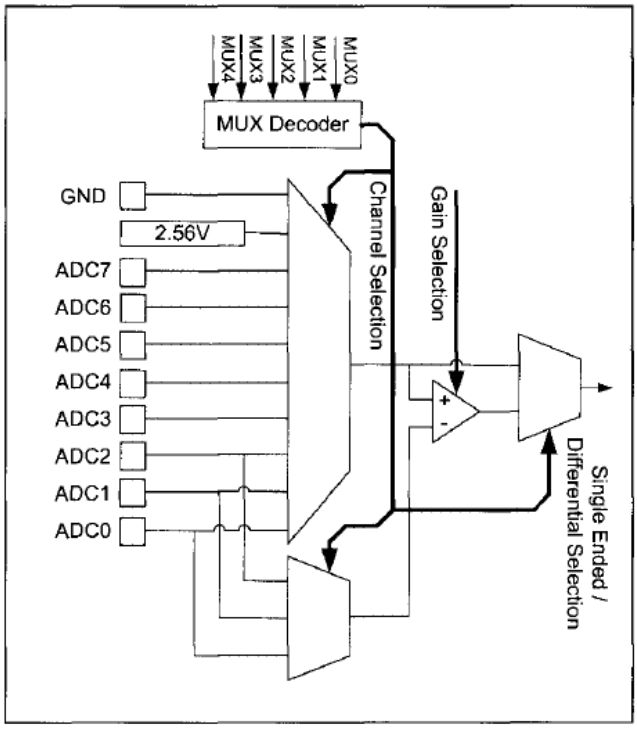
مثلا یک مبدل ۸ بیتی با ولتاژ ماکزیمم ۵ ولت می‌تواند سطوح ولتاژ با فاصله ۱۹ میلی ولت را ایجاد کند.

خروجی داده دیجیتال

ADC ۸ بیتی دارای ۸ بیت خروجی (D0-D7) است در حالی که ADC ۱۰ بیتی دارای ۱۰ بیت خروجی (D0-D9) می‌باشد.

$$D_{out} = \frac{V_i}{V_{ref}} * 2^n$$

[14] Least Significant Bit



شکل ۲۶ کانال‌های ورودی ADC - (پورت A)

رجیسترها

رجیستر کنترلی ADMUX

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

شکل ۲۷ - رجیستر کنترلی ADMUX

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

جدول ۱۲: تعیین ولتاژهای مرجع مبدل آنالوگ به دیجیتال

REFS1	REFS0	
0	0	AREF
0	1	AVCC
1	0	-
1	1	2.56

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 کننده‌های ولتاژ آن ولتاژ مرجع مورد نظر را ساخت و به پایه‌ی AREF داد.

ADLAR: از این بیت برای ۸ یا ۱۶ بیتی بودن مقدار خروجی ADC استفاده می‌شود. در صورت ۱ بودن نتیجه در بیت‌های پرارزش ADC (ADCH) و در صورت صفر بودن، نتیجه در بیت‌های کم‌ارزش ADC (ADCL) ذخیره می‌شود.

MUX0-4: (۰-۲) برای انتخاب کانال ورودی (در واقع با MUX0-2 یکی از ۷ پایه پورت A به عنوان ورودی ولتاژ آنالوگ به ریزپردازنده انتخاب می‌شود) و (۳-۴) انتخاب بهره تفاضلی استفاده شده است. از حالت بهره‌ی تفاضلی زمانی استفاده می شود که به جای یک ولتاژ ورودی آنالوگ، دو ولتاژ یکی مثبت و یکی منفی یا صفر داشته باشیم که در آن صورت اگر بهره برابر با A باشد، محاسبات به صورت زیر تغییر می کند:

$$D_{out} = A * \left(\frac{V_i^+ - V_i^-}{V_{ref}} \right) * 2^n$$

در جدول ۱۳ تنظیمات رجیستر ADMUX را برای حالت‌های تفاضلی و غیرتفاضلی مشاهده می‌کنید. در ستون Gain مقدار A مشخص شده است. در هنگام آزمایش برای استفاده از حالت تفاضلی باید Positive Differential Input را به ولتاژ ورودی آنالوگ و Negative Differential Input را به زمین وصل کنید.

جدول ۱۳ - تنظیمات قابل اعمال برای ADC با تغییر پارامترهای MUX4-0 در رجیستر ADMUX

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010		ADC0	ADC0	200x
01011		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110		ADC2	ADC2	200x
01111		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.22V (V_{BG})	N/A		
11111	0V (GND)			

رجیستر ADCSRA

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

شکل ۲۸- رجیستر ADCSRA

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 مد Free، 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 بیت‌های ADTS از رجیستر SFIOR انتخاب می شود.

ADIF: بعد از اتمام تبدیل یا تغییر در رجیستر داده ADC یک می‌شود. از یک شدن این بیت ما متوجه می‌شویم 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 یک وقفه ای صادر می‌شود که توسط آن زیر روال وقفه، می توان مقدار داده ADC را خواند.

ADPS0-3: از این بیت‌ها برای تعیین پالس ساعت ADC مطابق جدول زیر استفاده می کنیم.

جدول ۱۴ - جدول تنظیمات پیش تقسیم

ADPS2	ADPS1	ADPS0	Division Factor
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 مقدار کم‌ارزش و در ADCH مقدار پرارزش قرار دارد همچنین با استفاده از ADCW می توانیم محتوای هر دو متغیر را به صورت ۱۶ بیتی بخوانیم.

رجیستر SFIOR

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

شکل ۲۹ - رجیستر SFIOR

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

جدول ۱۵- تعیین منابع تحریک ADC

منبع تحریک ADC	ADTS0	ADTS1	ADTS2
مدعملکردآزاد	0	0	0
تحریک ازطریق مقایسه کننده آنالوگ	1	0	0
تحریک ازطریق وقفه خارجی صفر	0	1	0
تحریک ازطریق تایمر (شمارنده) صفر (در صورتی که نتیجه مقایسه برابر شود)	1	1	0
تحریک ازطریق تایمر (شمارنده) صفر (درصورت سریزشدن)	0	0	1
تحریک ازطریق تایمر (شمارنده) یک (درصورتی که نتیجه مقایسه B برابرشود)	1	0	1
تحریک ازطریق تایمر (شمارنده) یک (درصورت سریزشدن)	0	1	1
تحریک ازطریق حالت تسخیرتایمر (شمارنده) یک	1	1	1

پس باید توجه داشت که با توجه به فرکانس نوسان‌ساز ریزپردازنده ضریب تقسیمی از جدول ADPS0-2 انتخاب کنیم که فرکانس واحد ADC در محدوده‌ی گفته شده قرار گیرد.

معرفی وقفه ADC

برای جلوگیری از اتلاف زمان ریزپردازنده به جای بررسی مکرر بیت ADSC در رجیستر ADCSRA بهتر است از وقفه استفاده کنیم. (توجه کنید برای استفاده از وقفه ADC باید بیت ADIE از ثبات ADCSRA یک باشد). به محض کامل شدن تبدیل، پرچم ADIF یک می‌شود و CPU را مجبور می‌کند به محل اجرای وقفه پرش کرده و از نتیجه ADC استفاده کند.

مراحل برنامه‌نویسی ADC

۱. انتخاب یک پین به عنوان کانال ورودی ADC.
۲. فعال کردن ماژول ADC

۳. تعیین کردن سرعت تبدیل
۴. انتخاب ولتاژ مرجع در رجیستر ADMUX
۵. فعال کردن شروع تبدیل با set کردن بیت ADSC در رجیستر ADCSRA
۶. انتظار برای پایان تبدیل با سرکشی به ADIF در رجیستر ADCSRA
۷. بعد از یک شدن بیت ADIF خواندن رجیستر ADCL و ADCH برای خواندن داده خروجی. (در نظر داشته باشید که رجیستر ADCL را باید قبل از ADCH بخوانید)
۸. برای خواندن همین کانال به مرحله ۵ رجوع شود.
۹. برای عوض کردن کانال ورودی یا ولتاژ مرجع به مرحله ۴ رجوع شود.

مثال) فرض کنید می‌خواهیم دمایی در بازه‌ی ۰ تا ۵۰ درجه را اندازه‌گیری کنیم.در این صورت داریم:

T=0 => $D_{OUT} = 00\,0000\,0000$

T=50 => $D_{OUT} = \frac{V_{LM35}}{V_{REF}} * 2^{10}$

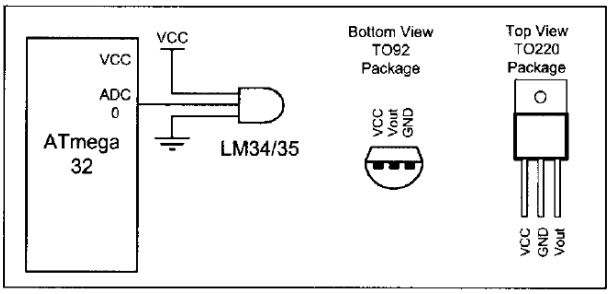
از طرفی می‌دانیم در این سنسور رابطه‌ی دما و ولتاژ خروجی عبارتست از:

$$V_{LM35} = 10mv \times T$$

بنابراین با توجه به روابط قبلی داریم:

$$T = \frac{D_{OUT} \times 500}{2^{10}} \cong \frac{D_{OUT}}{2}$$

اکنون قصد داریم تا با خواندن ولتاژ LM35 دمای محیط را بر روی LCD متصل به پورت B نمایش دهیم. برای این منظور LM35 را مطابق شکل زیر به پایه ADC0 متصل می کنیم.



شکل ۳۰ نحوه اتصال LM35 به ریزپردازنده

سپس در قسمت wizard تنظیمات مربوط به LCD و ADC را انجام می‌دهیم و قطعه کد زیر را به برنامه اضافه می‌کنیم:

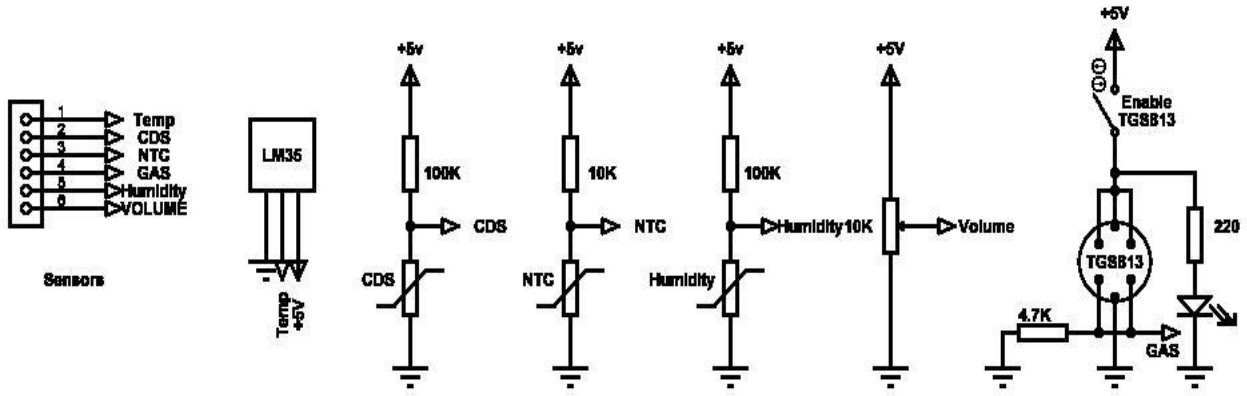
```
#include <mega16.h>
#include <lcd.h>
#include <delay.h>
#include <stdio.h>

#define ADC_VREF_TYPE 0x60

unsigned char read_adc(unsigned char adc_input)
{
      ADMUX=adc_input | (ADC_VREF_TYPE);
      delay_us(10);
      ADCSRA|=0x40;
      while ((ADCSRA & 0x10)==0);
      ADCSRA|=0x10;
      return ADCH;
}

void main(void)
{
      char str[10];
      DDRA=0x00;
      DDRB=0xFF;
      ADMUX=ADC_VREF_TYPE;
      ADCSRA=0x83;
      lcd_init(16);
      while (1)
      {
            read_adc(0);
            sprintf(str,"%d.%d",(ADCH*500)/1023,(( ADCH*500)/1023)%10);
            lcd_puts(str);
      }
}
```

معرفی چند سنسور

سنسورهای CDS، NTC و HIS06 به ترتیب سنسورهای دما، نور و رطوبت می‌باشند. این سه نوع سنسور با تغییر پارامترهای محیط تغییر مقاومت می‌دهند. از این رو به منظور ارتباط با ریزپردازنده طبق مدارهای شکل ۳۱ تغییرات مقاومت به تغییرات ولتاژ تبدیل شده است.

سنسور TGS813 سنسور آشکارساز گاز شهری با خروجی مستقیم ولتاژ می‌باشد که نیاز به هیچ گونه تبدیلی برای اعمال به ریزپردازنده ندارد. به دلیل افزایش حرارت بدنه سنسور و همچنین جریان کشی بالای این سنسور از منبع تغذیه

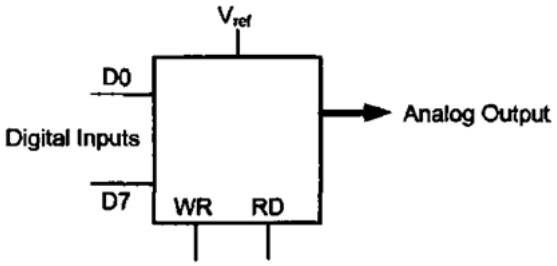
یک کلید کشویی برای وصل و قطع نمودن این سنسور در مسیر تغذیه سنسور مطابق شماتیک شکل زیر تعبیه شده است. LED موجود در این بلوک نشانگر وصل بودن و یا قطع بودن این سنسور در مدار است. همچنین یک عدد ولوم ۱۰ کیلو اهم به منظور تولید ولتاژ از سطح صفر ولت تا ۵ ولت برای شبیه‌سازی سنسورهای پارامترهای محیط در این بلوک قرار داده شده است. شماتیک مربوط به کلیه سنسورها در شکل زیر مشاهده می شود.



شکل ۳۱- شماتیک مربوط به کلیه‌ی سنسورها بر روی برد آموزشی

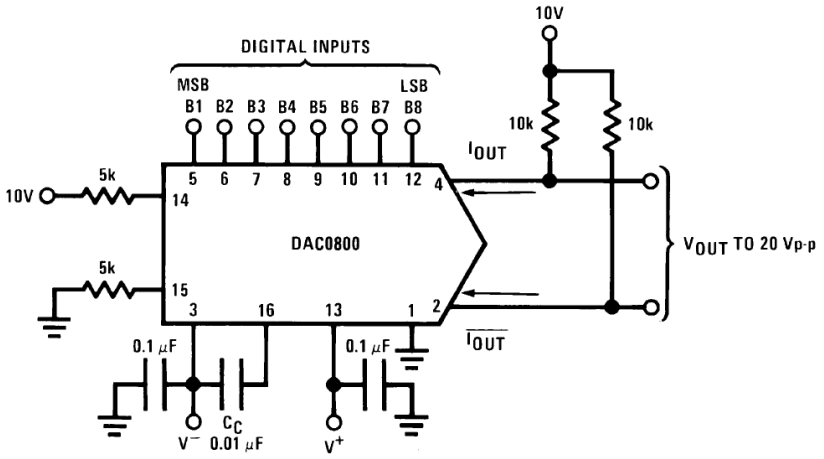
آشنایی با نحوه تبدیل سیگنال دیجیتال به آنالوگ

در این قسمت نحوه تبدیل سیگنال دیجیتال به آنالوگ را بررسی می‌کنیم. برای انجام این تبدیل دو روش نردبانی[15] و باینری وزن دار وجود دارد. در این قسمت از ماژول DAC0800 به عنوان یک تبدیل‌کننده استفاده می‌شود. نکته حائز اهمیت در انتخاب ماژول DAC مقدار resolution آن است که به تعداد ورودی‌های آن بستگی دارد که به طور معمول تعداد ورودی‌ها می‌تواند ۸، ۱۰ یا ۱۲ بیت باشد. اگر تعداد ورودی‌های DAC به تعداد n باشد تعداد سطوح خروجی آن 2^n خواهد بود. مثلاً DAC با ۸ بیت ورودی دارای ۲۵۶ سطح در خروجی ولتاژ یا جریان می‌باشد. DAC0800 یک تبدیل‌کننده با ۸ بیت ورودی و با خروجی جریان آنالوگ است.



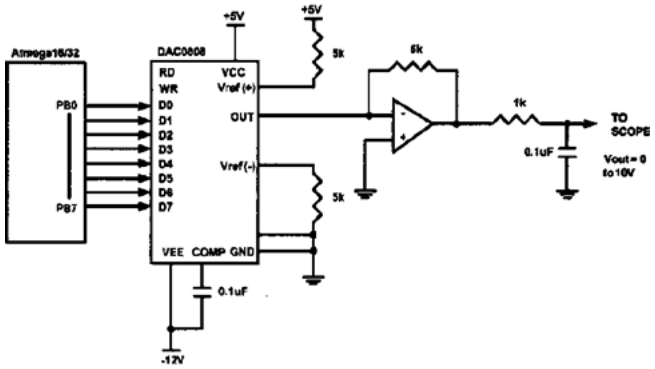
شکل ۳۲: بلوک دیاگرام

[15] ladder



شکل ۳۳ DAC0800

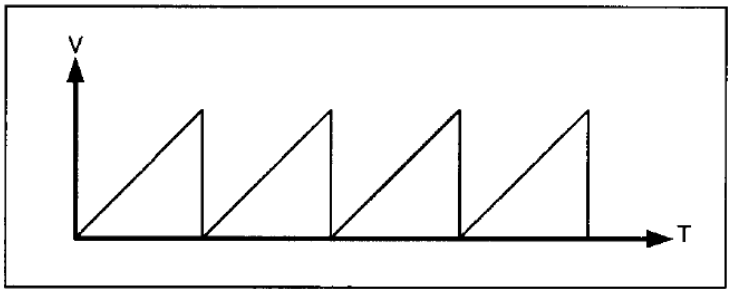
D0 کم‌ارزش‌ترین بیت ورودی و V_{ref} ولتاژ ورودی است که باید به پایه ۱۴ و ۱۵ اعمال شود تا جریان مورد نیاز I_{ref} تامین گردد. بیشترین خروجی جریان I_{out} برابر 1.99mA است. نحوه تبدیل جریان خروجی به ولتاژ شبیه به شکل ۳۴ می‌باشد.



شکل ۳۴: نحوه اتصالات به ریزپردازنده

$$I_{out} = I_{ref}\left(\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256}\right)$$

```
#include <mega16.h>
unsigned char i         //define a counter
int main (void)
{
      DDRB = OxFF;
      while (1)
      {
            PORTB = i;
            i++;
      }
      return 0;
}
```



شکل ۳۵: Step Ramp Output

پیش گزارش آزمایش پنجم

نام و نام خانوادگی: **شماره دانشجویی:**

---

۲) آزمایش‌های ۵-۳ و ۵-۲ را با استفاده از نرم‌افزار Proteus شبیه‌سازی کنید. (کد نوشته شده به زبان C شماتیک مدار و اطلاعات مربوط به Simulation log را در فایل پیش‌گزارش قرار دهید)

دستور کار

آزمایش ۵-۱

برنامه‌ای بنویسید که در خط اول LCD مقدار خوانده‌شده از Volume Controller و در خط دوم مقدار خوانده‌شده از ADC input generator را نشان دهد.

آزمایش ۵-۲ .

آزمایش قبل را با استفاده از تایمرها انجام دهید.

آزمایش ۵-۳

آزمایشی طراحی کنید که مقدار خروجی بلوک Bit Generator را با مقدار آنالوگ Volume Controller مقایسه کند. درصورتی‌که مقدار ولتاژ Volume Controller بیشتر باشد موتور پله‌ای با سرعت ۱۰۰ دور بر دقیقه شروع به چرخش کند.

آزمایش ششم

اهداف:

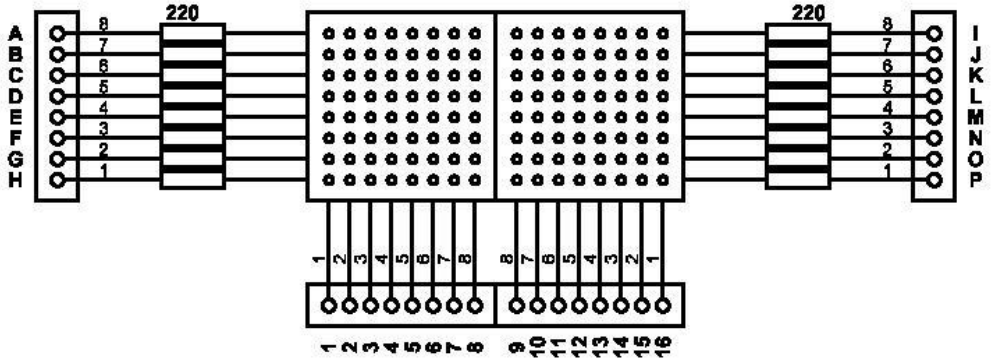
- نحوه کار با نمایشگرهای Dot-matrix
- آشنایی با نحوه کار LCDهای گرافیکی

معرفی نمایشگر Dot-matrix:

دو عدد نمایشگر Dot-matrix از نوع 8x8 (۸ ستون و ۸ ردیف) در بلوکی تحت عنوان Dot-matrix Display قرار داده شده است. Dot-matrix مورد استفاده در این مجموعه آموزشی به رنگ سبز می‌باشد.

ستون‌ها توسط اعداد ۱ تا ۱۶ و ردیف‌ها توسط حروف A تا P نامگذاری شده‌اند. به منظور روشن نمودن هر کدام از LED ها می‌توان به ستون مربوط به همان LED صفر منطقی و به ردیف متناظر یک منطقی اعمال نمود.

شماتیک مربوط به این بلوک در شکل ۳۶ مشاهده می‌شود.



شکل ۳۶ - نمایشگر Dotmatrix

آشنایی با نحوه‌ی کار نمایشگر Dot-matrix:

در این قسمت خواهیم گفت که چگونه باید یک ماتریس از LED در ابعاد ۸×۸ (۶۴ عدد LED) را کنترل کرد، یا به بیان دیگر چگونه حروف و علایم را بر روی ۶۴ عدد LED به نمایش بگذاریم. همانطور که گفته شد این ماتریس دارای ۸ سطر و ۸ ستون می‌باشد، با اتصال ماتریسی LEDها به ریزپردازنده، تعداد پایه‌های خروجی را به ۱۶ پین کاهش می‌دهیم. برای کنترل نمایشگر باید از روش جاروب کردن سطر و ستون استفاده کرد، به این صورت که ابتدا سطر اول را یک (مابقی صفر) می‌کنیم و سپس مقادیر (خاموش یا روشن بودن هر LED در این سطر) مربوط به ستون‌ها را روی ۸ پین ستون قرار می‌دهیم. حال در مرحله‌ی بعد سطر دوم را یک (مابقی صفر) می‌کنیم و سپس مقادیر مربوط به ردیف دوم را بر روی ۸ پین ستون قرار می‌دهیم و همین‌طور تا سطر هشتم ادامه می‌دهیم (عمل جاروب کردن را می‌توان با

شیفت دادن صفر منطقی روی ستون‌ها و قرار دادن مقادیر روی ردیف‌ها نیز انجام داد). به یاد داشته باشید که تمامی این مراحل باید با سرعت بالا انجام پذیرد، در برنامه‌های این دستورکار ما زمان تاخیر بین هر یک از عملیات‌های فوق را ۲ میلی ثانیه قرار داده‌ایم و از آنجایی که چشم انسان خطای دید دارد می‌توان به راحتی وضعیت هر ۶۴ LED را در لحظه مشاهده کرد. این روش همانند ایجاد کاراکترهای خاص (مثلاً علایم و حروف فارسی) درLCDهای کاراکتری می باشد.

اصول کلی ارسال داده

همانطور که گفتیم برای ارسال داده‌ها باید از روش جاروب کردن استفاده کنیم. به این صورت که در هر مرحله تنها یکی از ردیف‌ها یک شود و مابقی صفر باشند و این لحظه داده‌های مربوط به این ستون نوشته شوند، آرایه زیر این عملیات برای ما انجام می دهد.

```
unsigned char R_data[8] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
```

ایجاد فونت

به منظور ایجاد فونت‌ها یک آرایه به نام alphabet در نظر گرفته شده است که باید با حروف a-z و A-Z تکمیل گردد. در اینجا برای نمونه ۳ تا از حروف آن تکمیل شده است. در صورت تمایل می‌توان مابقی حروف را توسط نرم‌افزارهای مناسب مثل PicPars tools یا GLCD Font Creator در این زمینه، ایجاد کرد و در آرایه قرار داد.

```
char alphabet[][8]={
{ 0x06, 0x29, 0x29, 0x29, 0x1F, 0x00, 0x00, 0x00},     // a  Locate in array = 0
{ 0xFF, 0x09, 0x11, 0x11, 0x0E, 0x00, 0x00, 0x00},     // b  Locate in array = 1
{ 0x1E, 0x21, 0x21, 0x21, 0x12, 0x00, 0x00, 0x00},     // c  Locate in array = 2
{ 0x7F, 0x88, 0x88, 0x88, 0x7F, 0x00, 0x00, 0x00},     // A  Locate in array = 3
{ 0xFF, 0x91, 0x91, 0x91, 0x6E, 0x00, 0x00, 0x00},     // B  Locate in array = 4
{ 0x7E, 0x81, 0x81, 0x81, 0x42, 0x00, 0x00, 0x00},     // C  Locate in array = 5
};
```

جدول شناسایی موقعیت حروف

به ازای هر حرفی که در آرایه اضافه می‌شود باید در تابع lookup نیز یک case جدید برای شناسایی آن حرف ایجاد شود، **نکته مهم** این است که باید ترتیب قرار گرفتن رعایت شود. شماره‌ی حرف در متغیر locat ذخیره می‌شود بدین صورت که حرف اول (a) دارای مقدار صفر، حرف دوم (b) دارای مقدار ۱ می‌باشد و همین طور تا آخر. بنابراین باید آدرس locatها در تابع lookup نیز به طور صحیح وارد شود.

```
void lookup(char input)
{
switch (input)
    {
       case 'a'  :
              locat=0;
              break;
       case 'b'  :
```

```
                locat=1;
                break;
        case 'c' :
                locat=2;
                break;
        case 'A' :
                locat=3;
                break;
        case 'B' :
                locat=4;
                break;
        case 'C' :
                locat=5;
                break;
    }
}
```

معرفی LCD گرافیکی

در این آزمایش چگونگی اتصال LCD گرافیکی را به ریزپردازنده بررسی می‌نماییم.

اساس کار GLCD[16] بر مبنای Dot-matrix است، در نتیجه نیازمند یک راه‌انداز[17] برای تعیین زمان‌بندی روشن و خاموش شدن هر سطر و ستون می‌باشد. همانطور که در قسمت Dot-matrix توضیح داده‌شد برای ارسال داده‌ها از روش جاروب کردن سطر استفاده کردیم. بدین صورت که در هر مرحله تنها یکی از سطرها یک و مابقی صفر می‌شوند که در این لحظه، داده‌های مربوط به این سطر نوشته می‌شوند.

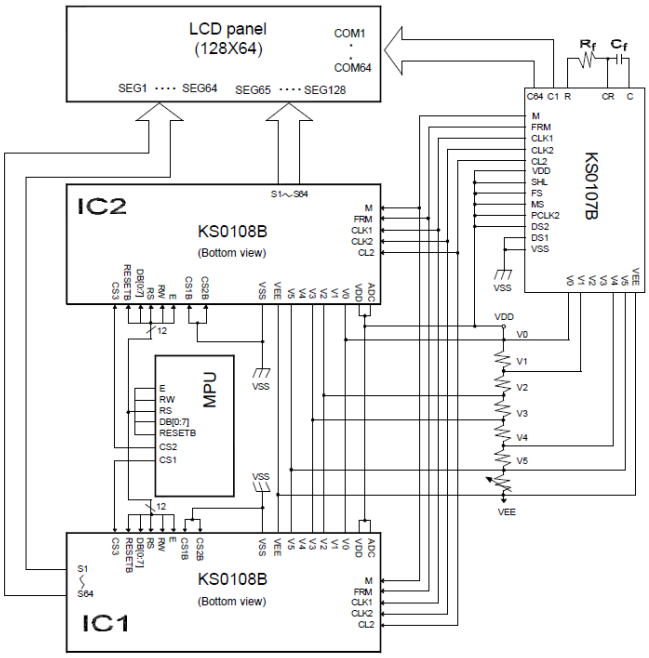
چیپِ کنترلری که این عملیات را انجام می‌دهد KS0108 نام دارد که برای راه‌اندازی GLCD به کار می‌رود. این چیپ، شامل یک RAM، ۶۴ بیت data latch و دیکدر می‌باشد. RAM برای ذخیره داده‌هایی که از طریق ریزپردازنده منتقل می‌شود و همینطور تولید نقاط ماتریس در GLCD (مربوط به نقاط ذخیره‌شده‌ای که قرار است نمایش داده شوند) استفاده می‌شود.

GLCD مورد استفاده در این آزمایشگاه دارای ابعاد 64*128 پیکسل می‌باشد که دارای دو چیپ کنترلی NT7108C و NT7107C است. این چیپ‌ها با کنترلرهای KS0108B و KS0107B سامسونگ سازگار هستند.

KS0108B یک راه‌انداز LCDهای Dot-matrix با ۶۴ کانال خروجی است که داده‌های موجود در RAM را برروی خطوط SEG قرار می‌دهد. اندازه Dot-matrix موجود ۶۴*۱۲۸ پیکسل است و به همین دلیل GLCD دارای ۲ سری از آن برای راه‌اندازی کردن ۱۲۸ سگمنت است. (شکل ۳۷ نشان دهنده‌ی این موضوع است). از طرف دیگر KS0107B

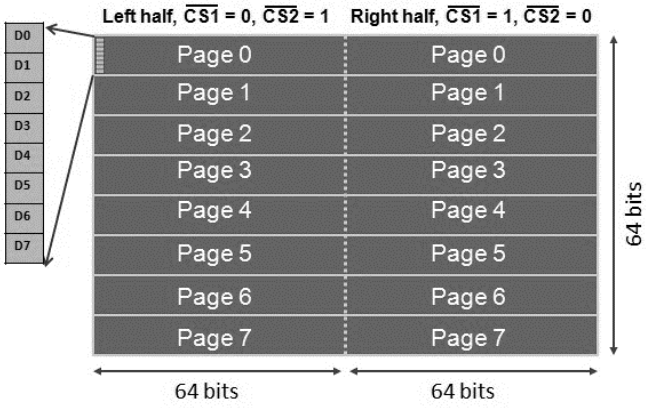
[16] Graphical LCD
[17] Drive

یک راه‌انداز ۶۴ کاناله است که برای تولید زمانبندی سیگنال‌های کنترلی برای ۲ عدد KS0108B راه‌اندازسگمنت به کار می‌رود که این همان عمل جاروب کردن سطر است. (شکل ۳۷ نشان دهنده‌ی این موضوع است)



شکل ۳۷: Application Diagram

KS0108B، ۶۴ خط display (COM1 – COM64) را راه‌اندازی می‌کند. در هر لحظه از زمان تنها یکی از خطوط COM فعال می‌باشد. اولین KS0108B (IC1) سگمنت‌های نیمه سمت چپ را فعال می‌کند و دومین KS0108B (IC2) سگمنت‌های نیمه سمت راست را فعال می‌نماید. به دو نیمه نمایشگر می‌توان به طور جداگانه با استفاده از پین های CS1[18] و CS2[19] دست یافت. هرکدام از نیمه‌ها بر پایه ۸ صفحه افقی ۰–۷ می‌باشد که هر کدام از یک بایت تشکیل شده است.



شکل ۳۸: pages

[18] Chip Select 1
[19] Chip Select 2

مقداردهی از صفحه صفر شروع می‌شود. اگر یک بایت دیتا را به GLCD انتقال دهیم در اولین ستون صفحه صفر نمایش داده می‌شود. (شکل ۳۸) اگر این کار را برای ۶۴ بایت داده تکرار نماییم سپس به نیمه دوم رفته و برای ۶۴ داده دیگر ادامه دهیم ۸ خط اول GLCD کشیده خواهد شد. برای ۸ خط دوم به همین ترتیب عمل می‌نماییم و به Page با آدرس ۱ می‌رویم. برای اینکه یک فریم کامل ۶۴*۱۲۸پیکسلی را نشان دهیم به ۲*۶۴*۱۲۸ بیت یا ۱۰۲۴ بایت داده احتیاج داریم. آدرس هر صفحه در رجیستر **X Page Register** ذخیره می‌شود. (شکل ۳۹)

آشنایی با پایه ها

جدول ۱۶- مشخصات پایه‌ها

Name	Level	Function	Connection with AVR PIN
Vss	0V	Ground	
Vcc	5.0V	+5v Supply in	
V0	variable	Contrast Adjust	
RS=CD	H/L	L:Instruction/H:Data Register Select	PX2[20]
RD	H/L	H:READ/L:WRITE SELECTION	PX1
CS1	L	Chip Select 1(segment 1-64)	PX5
CS2	L	Chip Select 2 (segment 65-128)	PX4
RST	L	RESET SIGNAL	PX3
EN	H/L	ENABLE SIGNAL	PX0
DB0	H/L	DATA IN/OUT	PY0
DB1	H/L	DATA IN/OUT	PY1
DB2	H/L	DATA IN/OUT	PY2
DB3	H/L	DATA IN/OUT	PY3
DB4	H/L	DATA IN/OUT	PY4
DB5	H/L	DATA IN/OUT	PY5
DB6	H/L	DATA IN/OUT	PY6
DB7	H/L	DATA IN/OUT	PY7
VEE		NEGATIVE 10V OUT	
LED+		LED BACKLIGHT	
LED-		LED BACKLIGHT	

[20] PX2 = PORTX2

پایه‌های GLCD به دو دسته تقسیم می‌شوند. پایه‌های داده (DB0-DB7) که وظیفه انتقال داده را برعهده دارند و دسته‌ی دیگر پایه‌های کنترلی هستند. پایه‌های کنترلی به شرح زیر می‌باشند:

R/W: زمانی که این پایه مقدار High داشته باشد، داده‌های موجود در DB[7:0] را نمایش می‌دهد و زمانی که E=H، CS2B=L و CS1B=L توسط CPU می‌تواند خوانده شود.

CS1 و CS2: زمانی که CS1=H و CS2=L باشد چیپ کنترلی KS0108B (IC1) انتخاب می‌شود و زمانی که CS1=L و CS2=H باشد چیپ کنترلی KS0108B (IC2) انتخاب می‌شود.

RS: باتوجه به جدول ۱۶ اگر RS=H باشد، GLCD دستورالعمل‌های دریافتی را اجرا می‌کند در غیر اینصورت مقدار خوانده‌شده از پایه‌های دیتا را در Data Register قرار می‌دهد.

نحوه ارسال فرمان به GLCD

Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Function
Display on/off	L	L	L	L	H	H	H	H	H	L/H	Controls the display on or off. Internal status and display RAM data is not affected. L:OFF, H:ON
Set address (Y address)	L	L	L	H	Y address (0-63)						Sets the Y address in the Y address counter.
Set page (X address)	L	L	H	L	H	H	H	Page (0-7)			Sets the X address at the X address register.
Display Start line (Z address)	L	L	H	H	Display start line (0-63)						Indicates the display data RAM displayed at the top of the screen.
Status read	L	H	Bus y	L	On/ Off	Reset	L	L	L	L	Read status. BUSY L: Ready H: In operation ON/OFF L: Display ON H: Display OFF RESET L: Normal H: Reset
Write display data	H	L	Write data								Writes data (DB0: 7) into display data RAM. After writing instruction, Y address is increased by 1 automatically.
Read display data	H	H	Read data								Reads data (DB0: 7) from display data RAM to the data bus.

شکل ۳۹: instructions

شکل بالا دستورالعمل‌های مختلفی که در GLCD می‌توان پیاده‌سازی کرد را نشان می‌دهد. برای نمونه تعدادی از دستورالعمل‌ها را بررسی می‌کنیم.

١.Display On/Off

جدول ١٧ **Display On/Off**

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	1	1	1	D

زمانی که مقدار D برابر 1 باشد داده نمایش دادهمی‌شود و وقتی D برابر 0 باشد محو می‌شود. وقتی D=0 است داده در RAM قرار دارد.

٢.Set Address (Y Address)

جدول ١٨ **Set Address (Y Address)**

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0

آدرس Y (AC0~AC5) آدرس داده داخل RAM است که در address counter قرار داده می‌شود. توجه کنید آدرس۶ بیت است زیرا هر چیپ باید ۶۴ بیت را راه‌اندازی کند. آدرسی که set می‌شود بعد از هر خواندن و نوشتن داده، یک عدد اضافه می‌شود.

٣. Set Page (X Address)

جدول ١ **Set Page (X Address)**

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0

داده‌ها در آدرس X (AC0~AC2) که نشان‌دهنده یک صفحه خاص است را نشان می‌دهد تا زمانی که صفحه بعد set شود. با توجه به جدول دستورالعمل‌ها که در شکل ۳۹ مشاهده می‌شود برای راه اندازی GLCD با استفاده از زبان C توابعی را معرفی می کنیم.

trigger(): برای ارسال دستور به GLCD ابتدا باید پایه کنترلی RS صفر شود سپس کد دستور به پورت مربوط به GLCD ارسال شود پس از ارسال دستور باید یک پالس پایین رونده در پایه E ایجاد شود بدین معنا که پایه E =1 شده و پس از مدتی برابر صفر شود. این تابع صفر و یک شدن پایه E را انجام می‌دهد.

```
void trigger()
{
	EN = 1; //EN high
	delay_us(E_DELAY);
	EN = 0; //EN low
	delay_us(E_DELAY);
}
```

glcd_on() و glcd_off(): این توابع LCD را روشن و خاموش می‌کنند. برای اینکار دستور 0x3F و 0x3E را به هر دو کنترلر می‌دهیم . چون پایه‌های CS1 وCS2، active low هستند، برای فعال شدن کنترلرها باید صفر باشند. RS هم طبق جدول ۱۶ در زمان ارسال دستور باید Low باشد.

دستور0x3F باتوجه به دستورالعمل Display On/Off بدست آمده است مقدار DB0=1 است پس داده نمایش داده می‌شود و در 0x3E مقدار DB0=0 است که داده نمایش داده نمی‌شود.

جدول ۲۰

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	1	1	1	1	1

```
void glcd_on()
{
	CS1 = 0; //Activate both chips
	CS2 = 0;
	RS = 0; //RS low --> command
	RW = 0; //RW low --> write
	DATAPORT = 0x3F; //ON command
	trigger();
}
void glcd_off()
{
	CS1 = 0; //Activate both chips
	CS2 = 0;
	RS = 0; //RS low --> command
	RW = 0; //RW low --> write
	DATAPORT = 0x3E; //OFF command
	trigger();
}
```

set_start_line(byte line): این تابع شماره سطر نمایش را عوض می‌کند. می‌توان هر کدام از شماره‌های بین ۰ تا ۶۳ را برای آن در نظر گرفت که نشان‌دهنده‌ی خط شروع می‌باشد و تأثیری بر روی داده نمایش داده شده از داخل RAM ندارد. برای تنظیم سطر مطابق شکل ۳۹ خطوط دیتا باید مقادیر زیر را داشته باشند.

جدول ۲۱

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	1	AC5	AC4	AC3	AC2	AC1	AC0

پس از دستور DATAPORT = 0xC0 | (AC0~AC5) استفاده می کنیم:

```
void set_start_line(byte line)
{
	RS = 0; //RS low --> command
	RW = 0; //RW low --> write
	CS1 = 0; //Activate both chips
	CS2 = 0;
	DATAPORT = 0xC0 | line; //Set Start Line command
	trigger();
}
```

glcd_write(byte b): یک بایت داده را روی مکانی که cursor قرار دارد می‌نویسد.

جدول ۲

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
1	1	D7	D6	D6	D5	D4	D3	D2	D1

```
void glcd_write(byte b)
{
	RS = 1; //RS high --> data
	RW = 0; //RW low --> write
	DATAPORT = b; //put data on data port
	delay_us(1);
	trigger();
}
```

معرفی هدرفایل glcd.h و نحوه استفاده در Code Vision

قطعه کد زیر چگونگی نمایش یک متن و شکل را نشان می‌دهد.

```
#include <mega16.h>
#include <glcd.h>
#include <delay.h>
#include <font5x7.h>

void main(void)
{
	GLCDINIT_t glcd_init_data;                  // >>>>>>>>1
	glcd_init_data.font=font5x7;                // >>>>>>>>2
	glcd_init_data.readxmem=NULL;               // >>>>>>>>3
	glcd_init_data.writexmem=NULL;              // >>>>>>>>4
	glcd_init(&glcd_init_data);                 // >>>>>>>>5

	glcd_outtextf("MICROPROCESSOR LAB");        // >>>>>>>>6
	delay_ms(800);                              // >>>>>>>>7
	glcd_clear();                               // >>>>>>>>8
	glcd_outtextf("Some line styles:");         // >>>>>>>>9
	glcd_line(0,10,127,10);                     // >>>>>>>>10
	glcd_setlinestyle(1,GLCD_LINE_DOT_SMALL);// >>>>>>>>11
	glcd_setlinestyle(1,GLCD_LINE_DOT_LARGE);// >>>>>>>>12
	glcd_line(0,20,127,20);                     // >>>>>>>>13

	while(1);
}
```

اتصالات GLCD در قسمت wizard مشخص می گردد. بدین صورت که بعد از ایجاد پروژه، تنظیمات پورت‌ها (جهت ورودی/خروجی) به صورت inline داخل فایل hex. ایجاد می‌شود. توجه داشته باشید که بعد از ایجاد پروژه و تنظیم پورت‌های GLCD دیگر قادر به تغییر تنظیمات پورت‌ها نخواهید بود.

برای استفاده از GLCD ابتدا نیاز به راه‌اندازی اولیه داریم که تابع glcd_init(*GLCDINIT_t) این کار را انجام می‌دهد. نوع داده‌ی GLCDINIT_t در فایل سرآیند glcd.h به صورت یک struct تعریف شده است. این نوع داده شامل یک متغیر برای تعیین فونت و دو اشاره گر برای خواندن و نوشتن از حافظه خارجی می‌باشد. خطوط ۱ الی ۴ مؤید این قضیه است.

تابع glcd_outtextf(flash char *str) یک رشته ذخیره شده در حافظه فلش را روی GLCD نشان می‌دهد. همچنین تابع glcd_line(GLCDX_t x0, GLCDY_t y0, GLCDX_t x1, GLCDY_t y1); یک خط از نقطه P(x0,y0) تا نقطه P1(x1,y1) با ضخامت (thickness) و الگوی بیتی (bit_pattern)که قبلاً تعیین شده می کشد. الگوی بیتی (bit_pattern) و ضخامت (thickness) در هدرفایل یک مقدار از پیش تعیین شده دارد که این مقدار را می‌توان با استفاده ازتابع glcd_setlinestyle(thickness,bit_pattern) تغییر داد. مقادیر قابل اتخاذ برای ضخامت (thickness) اعداد صحیح مثبت می‌باشند که این اعداد نمایان‌گر تعداد پیکسل‌های ضخامت است علاوه بر این الگوی بیتی (bit_pattern) می‌تواند توسط مقادیر GLCD_LINE_SOLID، GLCD_LINE_DOT_SMALL و GLCD_LINE_DOT_LARGE مقداردهی شود.

برای آشنایی بیشتر با توابع کاربردی GLCD می‌توانید به هدرفایل graphics.h مراجعه نمایید این هدرفایل در هدرفایل glcd.h فراخوانی شده است.

فونت

معمولاً از دو فونت با ابعاد 5x8 و 8x8 استفاده می‌شود. فونت‌های فارسی معمولا 8x8 و فونت‌های انگلیسی 5x8 می‌باشند. ایجاد فونت در GLCD همانند آنچه که در قسمت dot-matrix گفته شد می‌باشد. از هدرفایل font5x7.h استفاده می کنیم یا می‌توانیم با استفاده از نرم‌افزارهای طراحی فونت مانند قسمت dot-matrix اقدام به ایجاد فونت نماییم.

پیش گزارش آزمایش ششم

نام و نام خانوادگی: **شماره دانشجویی:**

---

۱) برای حرف اول نام و نام‌خانوادگی خود با استفاده از نرم‌افزار معرفی شده در مقدمه یک فونت فارسی مناسب طراحی کنید.

۲) باتوجه به دیتاشیت GLCD – TS 128x64 با کنترلر KS0108 تابعی برای پاک کردن GLCD بنویسید.

۳) آزمایش‌های ۶-۲، ۶-۵ و ۶-۶ را با استفاده از نرم‌افزار Proteus شبیه‌سازی کنید. (کد نوشته شده به زبان C شماتیک مدار و اطلاعات مربوط به Simulation log را در فایل پیش‌گزارش قرار دهید)

دستور کار

آزمایش ٦-١

حرف اول نام و نام‌خوانوادگی خود را بر روی نمایشگر اول و دوم dot-matrix نمایش دهید.

راهنمایی: برای پیاده‌سازی روش جاروب کردن می‌توان از قطعه کد زیر کمک گرفت.

```
{       Column = ~ C_data[i];
        Row = alphabet[locat][i];
        i++;
        if( i == 8 ) i = 0;         }
```

آزمایش ٦-٢

یک نقطه روی نمایشگر اول dot-matrix نشان دهید به این گونه که با فشردن دکمه‌های keypad در جهات مختلف حرکت کند. (با فشردن شماره ۸ به سمت بالا، ۲ به سمت پایین، ۶ به سمت راست و ۴ به سمت چپ حرکت نماید)

آزمایش ٦-٣

یک حرف بر روی نمایشگرهای dot-matrix نشان دهید که در جهت راست حرکت کند.

آزمایش ٦-٤

بر روی GLCD یک مستطیل، یک دایره و یک شش‌ضلعی نشان دهید.

راهنمایی: برای رسم اشکال هندسی از توابع موجود در هدرفایل graphics.h استفاده نمایید.

آزمایش ٦-٥

نمودار تابع $y = x^3$ را بر روی GLCD نشان دهید.

راهنمایی: از قطعه کد زیر برای محورهای مختصات استفاده کنید.

```
{
        glcd_setlinestyle(3,4);
        glcd_line(5,31,127,31);
        glcd_setlinestyle(1,GLCD_LINE_SOLID);
        glcd_line(0,32,127,32);

        glcd_setlinestyle(3,4);
        glcd_line(62,6,62,63);
        glcd_setlinestyle(1,GLCD_LINE_SOLID);
        glcd_line(63,0,63,63);
}
```

آزمایش ٦-٦

یک ساعت آنالوگ طراحی کرده و بر روی GLCD نمایش دهید.

آزمایش هفتم

اهداف:

- آشنایی با ارتباط سریال
- آشنایی با تراشه MAX232

مقدمه

انواع ارتباط سریال

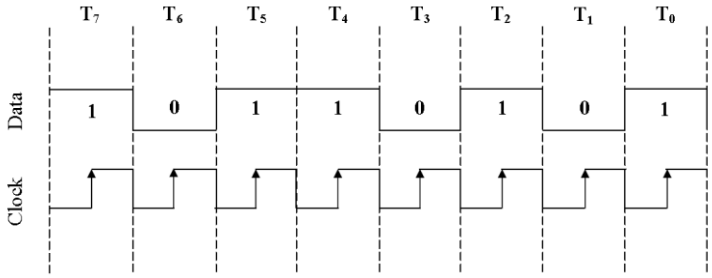
همانطور که می‌دانید، انتقال اطلاعات به دو روش سریال و موازی می‌تواند صورت گیرد. در روش موازی n بیت اطلاعات از طریق n خط داده منتقل می‌شود اما در روش سریال همه‌ی داده از طریق یک یا دو خط منتقل می‌شود. ارتباط سریال انواع مختلف دارد که از نظر حداکثر طول کابل ارتباطی، نرخ ارسال و دریافت داده و تعداد سیم‌ارتباطی، یک طرفه یا دو طرفه بودن، فرمت ارسال و دریافت داده ها، سنکرون یا آسنکرون بودن و نوع مدولاسیون با یکدیگر تفاوت دارند.

انواع ارتباط سریال Ehternet، USB، CAN، LonWorks، SATA، I2C و SPI می‌باشند.

ریزپردازنده ATmega16 به صورت سخت‌افزاری قابلیت برقراری ارتباط USART، SPI و I2C را برای ارتباط با وسایل جانبی از قبیل SD card، EEPROM، ADC و DAC دارا می‌باشد. در این آزمایش به بررسی ارتباط USART و SPI پرداخته می‌شود.

برای انتقال به روش سریال پروتکل‌های متنوعی در زمینه‌های گوناگون وجود دارد. اما به طورکلی ارتباط سریال به دو صورت می‌تواند برقرار شود:

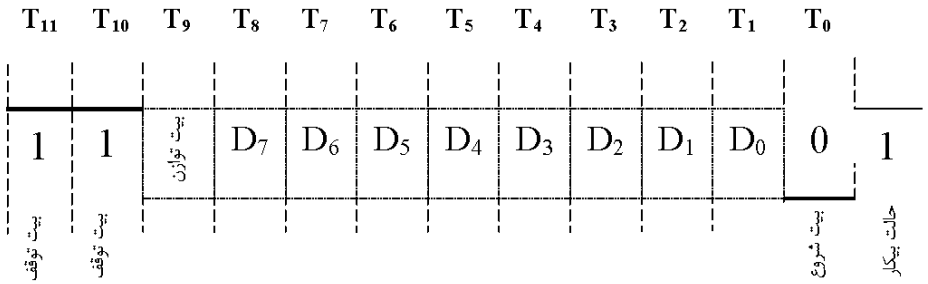
- ارتباط به صورت همگام؛ که در آن داده‌ها بر روی یک خط ارسال می‌شوند و یک خط کلاک همزمان کننده نیز وجود دارد که به همراه داده‌ها برای همگام‌سازی، توسط فرستنده ارسال می‌شود.



شکل ۴۰: ارتباط به صورت همگام

- ارتباط به صورت ناهمگام؛ که در آن داده موردنظر از طریق خط TXD ارسال شده و از خط RXD دریافت می‌شود. بنابراین در این ارتباط کلاکی برای همگام‌سازی ارسال نمی‌شود. در چنین روشی باید داده‌ها تحت

قالب‌بندی خاص و به صورت بیت به بیت با فواصل زمانی تعریف‌شده برای فرستنده و گیرنده منتقل شوند. به این فواصل زمانی، نرخ انتقال داده یا Baud Rate گفته می‌شود. در شکل ۴۱ یک قالب داده با یک بیت توازن و دو بیت توقف در ارتباط ناهمگام مشاهده می‌شود.



شکل ۴۱: ارتباط به صورت ناهمگام

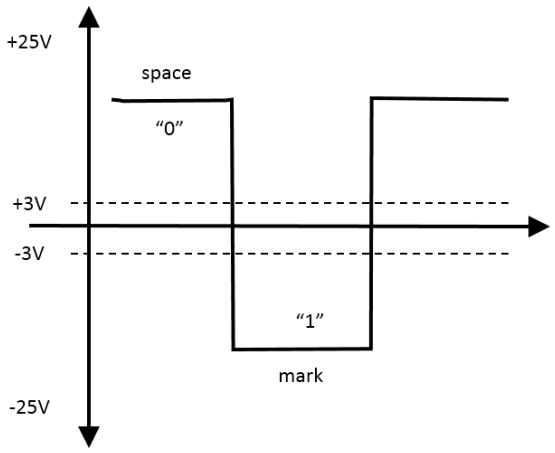
معرفی ارتباط USART (RS232)

یکی از پروتکل‌هایی که ریزپردازنده ATmega16 برای ارتباط سریال پشتیبانی می‌کنند، پروتکل USART[21] است که قابلیت برقراری ارتباط با هر دو مد همگام و ناهمگام را دارد.

از لحاظ تئوری تنها یک سیم برای انتقال اطلاعات سریال به صورت آسنکرون لازم است اما در واقعیت این مسئله عملی نیست. به عنوان مثال اگر یک بیت از اطلاعات بر اثر خطا یا نویز تغییر کند ممکن است کل اطلاعات بعد از آن یک بیت شیفت پیدا کند و پس از تفسیر و تبدیل به دیتای موازی کل اطلاعات مخدوش شود. بنابراین نیاز به استانداردهایی وجود دارد که امکان ارتباط قابل اطمینان را فراهم کند. یکی از این استانداردها RS232-C است که در سال ۱۹۶۹ توسط موسسه EIA تعریف شد. اگرچه نام این استاندارد RS232-C است اما معمولاً به نام RS232 شناخته می شود و مخفف Recommended Serial می‌باشد. این استاندارد معمولاً در پورت سریال کامپیوترهای شخصی و ماژول‌هایی مانند فرستنده/گیرنده RF، GPS و GSM برای ارتباط آن‌ها با ریزپردازنده استفاده می شود.

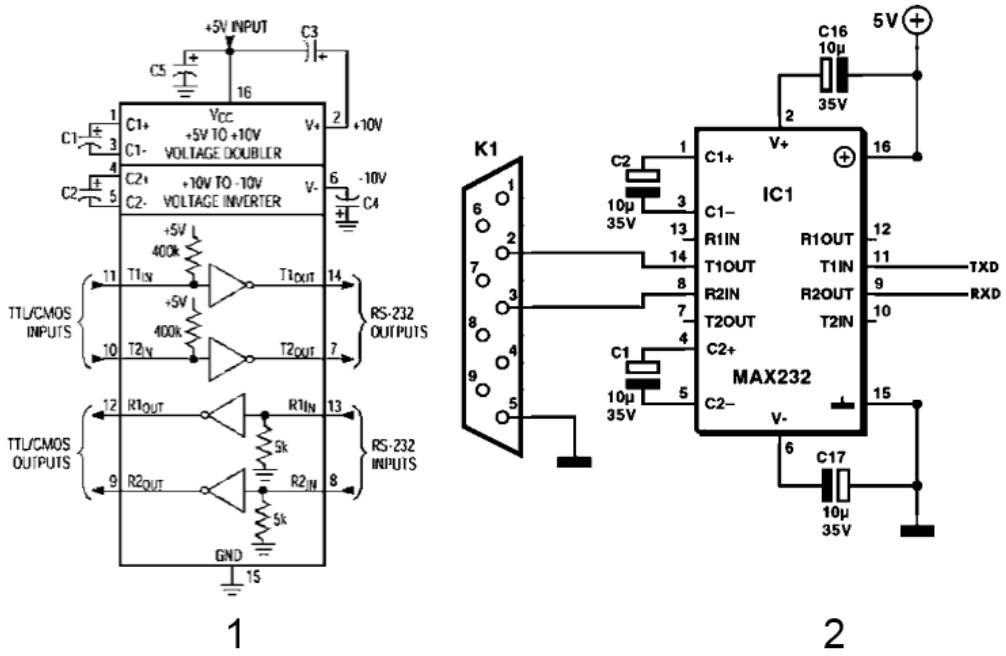
در استاندارد RS232 سطح ولتاژ ۳+ تا ۱۲+ نمایانگر وضعیت Space یا صفر منطقی و بازه ی ۳- تا ۱۲- ولت نمایشگر وضعیت Mark یا یک منطقی می‌باشد. این در حالی است که تجهیزات استاندارد TTL مثل ریزپردازنده ATmega16 در سطوح بین ۰ و ۵ ولت کار می‌کنند.

[21] Universal Synchronous Asynchronous serial Receiver/Transmitter



شکل ۴۲: سطوح ولتاژ در RS232

در نتیجه برای برقراری ارتباط بین وسایل TTL و RS232 نیاز به یک درایور مانند MAX232 احتیاج است تا سطوح ولتاژ آن‌ها را به یکدیگر تبدیل کند. MAX232 یک تراشه ی ۱۶ پایه شامل ۲ فرستنده و ۲ گیرنده است (شکل ۴۳).



شکل ۴۳: ۱) مدار داخلی max232 . این تراشه دو ورودی و دو خروجی دارد-۲) نحوه اتصال تراشه به کانکتور مادگی D9

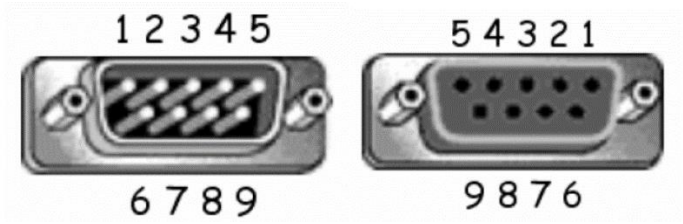
در ادامه نحوه برقراری ارتباط بین کامپیوتر و ریزپردازنده از طریق RS232 را بررسی می کنیم. برای اینکار ابتدا مدار مورد نیاز را برقرار کرده و سپس برنامه مناسب در ریزپردازنده را می‌نویسیم.

ارتباط ATmega16 و کامپیوتر

ATmega16 دارای ۲ پایه برای دریافت و انتقال داده به صورت سریال (مطابق با استاندارد RS232) با نام‌های TX و RX واقع در PD0 و PD1 دارد که مطابق استاندارد TTL می‌باشند. به همین دلیل باید این پایه‌ها را به MAX232 متصل نمود. همانطور که درشکل ۴۳ نشان داده شده TX به T1IN و RX به R2OUT متصل می‌گردند.

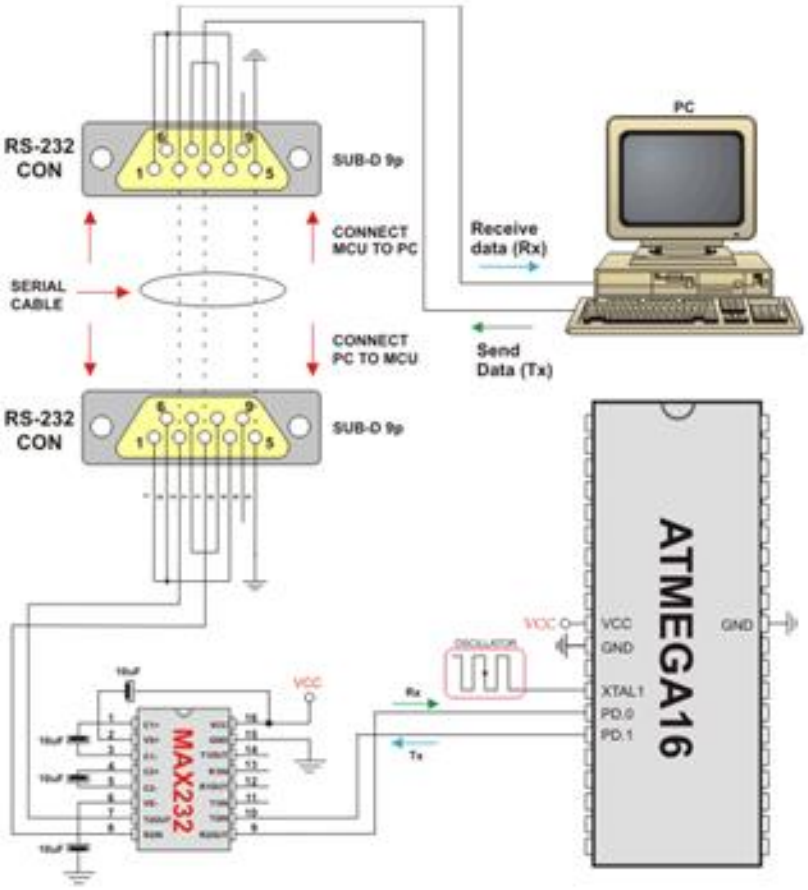
بعد از برقراری اتصال ریزپردازنده به MAX232 سپس باید این تراشه را به پورت سریال کامپیوتر متصل نمود تا این ارتباط کامل گردد.

پورت سریال در کامپیوترهای شخصی که به نام‌های COM Port و RS-232 Port هم شناخته می‌شود برای برقراری ارتباط سریال با دستگاه‌های خارجی مانند مودم، ماوس‌های سریال، قفل‌های دیجیتال و ... به کار می‌رود. این پورت به تراشه‌ی UART تعبیه شده در کامپیوتر متصل است و با استفاده از ثبات‌های این تراشه می‌توان اموری مانند ارسال و دریافت داده، خواندن وضعیت خط، کنترل سیگنال‌های handshaking، تنظیم Baud Rate و غیره را انجام داد.



شکل ۴۴: سمت راست کانکتور مادگی D9 - سمت چپ کانکتور نری D9

1	Data carrier detect
2	Receive Data (RXD)
3	Transmit Data (TXD)
4	Data terminal ready (DTR)
5	GND
6	Data set ready
7	Request to send
8	Clear to send
9	Ring indicator



شکل ۴۵: اتصال کامل ریزپردازنده به کامپیوتر از طریق درگاه سریال

تنظیمات داخل ریزپردازنده برای ارتباط سریال

در ابتدا به معرفی رجیسترهای مربوط به ارتباط USART در AVR می‌پردازیم:

رجیستر UDR: اصطلاحاً رجیستر بافر داده خوانده می‌شود و داده‌های ارسالی و دریافتی در آن ریخته می‌شوند. این رجیستر ۱۶بیتی از دو بخش RXB[0:7] و TXB[0:7] تشکیل شده است. رجیستر UDR دارای انتقال Full Duplex است و می تواند به طور همزمان اطلاعات را ارسال (با استفاده از پایه TX) و دریافت کند (با استفاده از پایه RX).

رجیستر UCSRA:

- **بیت ۷ – RXC (USART Receive Complete)**: درصورتی که داده‌های موجود در بافرگیرنده خوانده نشده باشند، این پرچم برابر یک و در غیر این صورت صفرخواهد بود.

- **بیت ٦ – TXC (USART Transmit Complete)**: زمانی که آخرین فریم فرستاده شده باشد و داده‌ای در بافر فرستنده وجود نداشته باشد، این بیت برابر یک و در غیر این صورت صفر خواهد بود.
- **بیت ٥ – UDRE (USART Data Register Empty)**: در صورت یک بودن، بافر فرستنده آماده دریافت داده‌ی جدید می‌باشد.
- **بیت ٤ – FE (Frame Error)**: اگر کاراکتر بعدی در زمان دریافت در بافر گیرنده، خطای فریم داشته باشد، این بیت یک می‌شود.
- **بیت ۳ – DOR (Data Over Run)**: زمانی که بافر گیرنده پر باشد و کاراکتر جدیدی هم درشیفت رجیستر گیرنده منتظر باشد و بیت شروع جدیدی هم تشخیص داده شود، این بیت یک می‌گردد.
- **بیت ۲ – PE (Parity Error)**: اگر در کاراکتر بعدی موجود در بافر گیرنده خطایparity وجود داشته باشد ،این بیت یک می شود.
- **بیت ۱ – UDX (Double the USART transmission Speed)**: این بیت تنها در حالت آسنکرون کاربرد دارد و در حالت سنکرون باید صفر گردد. با نوشتن یک دراین بیت نرخ ارسال به جای ۱۶ بر ۸ تقسیم می‌گردد و به این ترتیب در ارتباط آسنکرون سرعت انتقال را ۲ برابر می‌کنند.
- **بیت صفر – MPCM (Multi-Processor communication Mode)**: این بیت امکان ارتباط چند ریزپردازنده با یکدیگر را فراهم می‌کند.

رجیستر UCSRB:

- **بیت ۷ – RXCIE (RX Complete Interrupt Enable)**: با یک کردن این بیت، وقفه اتمام دریافت فعال می‌شود.
- **بیت ٦ – TXCIE (TX Complete Interrupt Enable)**: با یک کردن این بیت، وقفه مربوط به اتمام ارسال فعال می‌گردد.
- **بیت ٥ – UDRIE (USART Data Register Empty Interrupt Enable)**: با یک کردن این بیت وقفه مربوط به خالی شدن بافر فعال می‌گردد .
- **بیت ٤ – RXEN (Receiver Enable)**: با یک کردن این بیت،USART به صورت گیرنده فعال می‌گردد.
- **بیت ۳ – TXEN (Transmitter Enable)**: با یک کردن این بیت،USART به صورت فرستنده فعال می‌گردد.
- **بیت ۲ – UCSZ (Character Size)**: این بیت به صورت ترکیب با بیت‌های USCZ 1:0 تعداد بیت‌های داده در یک فریم را مشخص می‌کند.

- **بیت ۱ – RXB8 (Receive Data Bit 8)**: زمانی که فریم ۹ بیت داده داشته باشد، بیت نهم مربوط به بیت داده ازکاراکتر دریافتی را در خود جای می‌دهد.
- **بیت صفر – TXB8 (Transmit Data bit 8)**: زمانی که فریم ۹ بیت داده داشته باشد، بیت نهم مربوط به بیت داده از کاراکتر ارسالی را در خود جای می‌دهد.

رجیسترهای UBRRL و UBRRH (USART Baudrate Register)

رجیستر UBRRH و رجیستر UCSRC از یک مکان در حافظهI/O استفاده می‌کنند.

- **بیت ۱۵ – URSEL (Register Select)**: این بیت برای انتخاب دسترسی به رجیسترهایUBRRH وUCSRC به کار می‌رود و برای دسترسی به رجیستر UBRRH باید صفر شود.
- **بیت‌های ۱۴-۱۲ Reserved Bits**: این بیت‌ها برای استفاده‌های بعدی ذخیره شده‌اند.
- **بیت‌های ۱۱-۰ UBRR**: این یک رجیستر ۱۲ بیتی است که نرخ ارسالUSART راتنظیم می‌کند.

کتابخانه stdio.h

برای برقراری ارتباط با پایه‌های USART ریزپردازنده از هدرفایل مذکور استفاده می‌کنیم. تعدادی از توابع آن را در ذیل مشاهده می‌کنید.

- *تابع* printf

```
void printf ( char flash *fmtstr [, arg1,arg2,…])
```

برای فرستادن اطلاعات موجود در حافظه flash به کار می‌رود. در واقع وقتی می‌خواهیم متنی را روی پورت سریال ریزپردازنده بفرستیم به این صورت از آن استفاده می‌کنیم:

```
printf("Hello IUT");
```

- تابع scanf

```
signed char scanf(char flash *str[arg1 address,arg2 address,…])
```

این تابع می‌تواند داده‌های ورودی از پورت سریال را بخواند و آن را در یک رشته ذخیره کند.

```
char str[16];
scanf( "%s" , str );
```

- *تابع* putsf

```
void putsf(char flash *str)
```

اگر بخواهیم یک متغیر رشته‌ای را روی پورت سریال بریزیم از آن استفاده می‌کنیم.

- *تابع* gets

```
char *gets( char *str , unsigned char length )
```

این تابع یک رشته کاراکتر ASCII را به طول مشخصی از ورودی سریال دریافت نموده و در متغیر رشته‌ای str ذخیره می‌کند.

تابع char getchar(void) یک کاراکتر را به روش Polling از USART دریافت کرده و نتیجه را return می‌کند.

تابع void putchar(char c) نیزیک کاراکتر را به روش Polling از طریق USART ارسال می‌کند.

اکنون با آشنا شدن با رجیسترهای USART در ریزپردازنده می‌توانیم برنامه مورد نیاز خود را بنویسیم.

دریافت اطلاعات در کامپیوتر

برقراری ارتباط سریال در نرم‌افزار MATLAB

برای برقراری ارتباط با درگاه سریال در MATLAB، ابتدا باید آن را به صورت یک شئ تعریف کنیم. برای انجام این کار از دستور زیر استفاده می‌کنیم.

```
serial_PORT = serial('COMx','baudrate', baudrate_value);
```

سپس برای ارسال یا دریافت اطلاعات، باید درگاه مورد نظر را باز کنیم. از دستور زیر می‌توان برای این کار استفاده نمود.

```
fopen(serial_PORT);
```

برای خواندن اطلاعات از دستور fread و در پایان از دستور fclose برای بستن درگاه استفاده می‌کنیم.

برای تنظیمات بیشتر درگاه سریال به help نرم‌افزار MATLAB رجوع کنید.

پیش گزارش آزمایش هفتم

نام و نام خانوادگی: **شماره دانشجویی:**

---

۱)آزمایش ۷-۲ را با استفاده از نرم‌افزار Proteus شبیه‌سازی کنید. (کد نوشته شده به زبان C، شماتیک مدار و اطلاعات مربوط به Simulation log را در فایل پیش‌گزارش قرار دهید)

دستور کار

آزمایش ۷-۱

پایه‌های RXD و TXD ریزپردازنده را به RXD و TXD مبدل MAX232 متصل کنید و کابل RS-232 را از یک طرف به درگاه COM کامپیوتر و از طرف دیگر به مبدل وصل کنید. برنامه زیر را پروگرام نموده و نتیجه را با استفاده از نرم‌افزار TeraTerm نشان دهید. نحوه‌ی کارکرد برنامه را توضیح دهید.

```
#include <mega16.h>
void main()
{
        int i;
        UCSRA=0x00;
        UCSRB=0x08;
        UCSRC=0x86;
        UBRRH=0x00;
        UBRRL=0x33;
        i=0;
        while(1)
         {
                delay_ms(100);
                UDR = i ++;
                while(UCSRA.6 == 0);
                UCSRA.6=1;
        }
}
```

آزمایش ۷-۲

آزمایشی طراحی کنید که با دریافت فرمان "motor" از کامپیوتر، موتور پله‌ای را با سرعت ۵۰ دور بر دقیقه بچرخانـد و با دریافت فرمان "ADC"، اندازه شدت نور را با استفاده از سنسور CDS بر روی LCD نمایش دهد.

آزمایش ۷ -۳

هدف از انجام این آزمایش، برقراری ارتباط ریزپردازنده با نرم‌افزار MATLAB می‌باشد. آزمایشی طراحی کنید که تغییرات ولتاژ Volume Controller را به صورت real-time در نرم‌افزار MATLAB نمایش دهد.

راهنمایی: برای نمایش به صورت real-time از قطعه کد زیر در محیط نرم‌افزار MATLAB استفاده کنید.

```
serial_PORT = serial('COMx', 'baudrate', 9600);
fopen(serial_PORT);
a=[];
while(1)
  b=fread(serial_PORT, 8);
  for x=1:length(b)
   if isnumeric(b(x))
     a=[a,b(x)];
   end
     if(length(a)<400 || length(a)==400)
          plot(a);
          axis ([1 400 0 300]);
     else
          plot(a(end-399:end));
          axis ([1 400 0 300]);
     end
  end
  b=[];
  grid on;
  drawnow;
end
```

آزمایش هشتم

اهداف

- آشنایی با ارتباط سریال SPI
- آشنایی با ارتباط سریال دو سیمه
- توانایی مقایسه امکانات SPI و TWI و استفاده از آن‌ها
- آشنایی با حافظه‌های EEPROM سریال 24LC32 و توانایی برقراری ارتباط با آن‌ها

انواع ارتباط سریال

انواع ارتباط سریال از نظر حداکثر طول کابل ارتباطی، نرخ ارسال و دریافت داده و تعداد سیم ارتباطی، یک طرفه یا دو طرفه بودن، فرمت ارسال و دریافت داده‌ها، سنکرون یا آسنکرون بودن نوع مدولاسیون با یکدیگر تفاوت دارند.

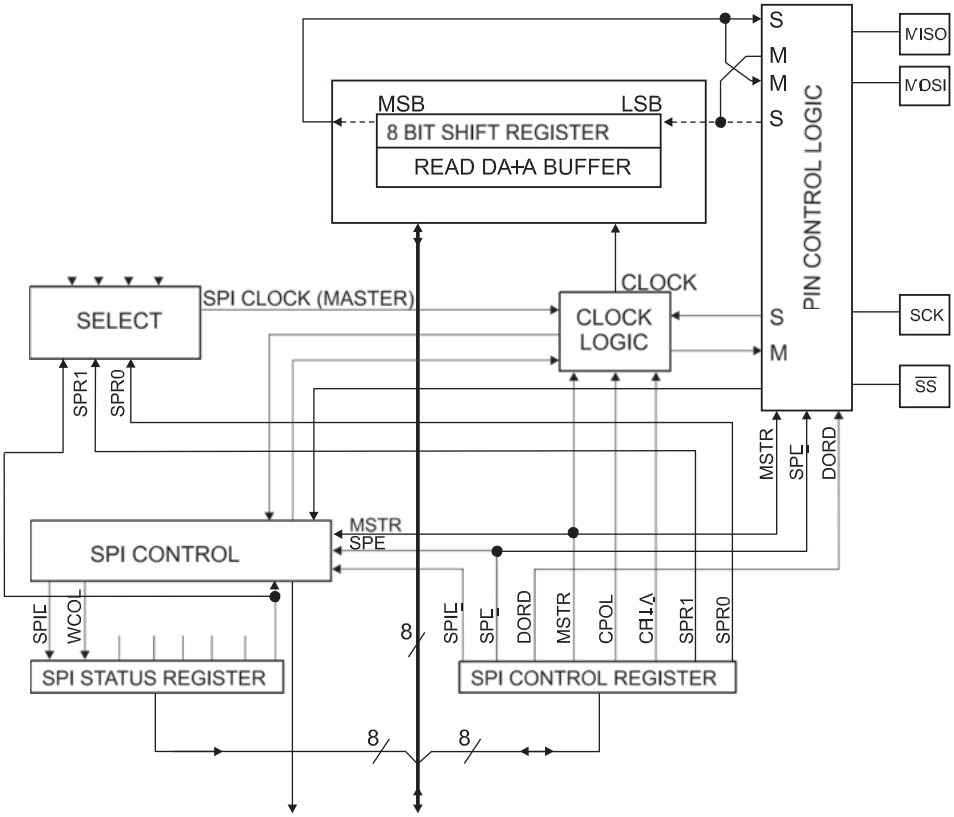
انواع ارتباط سریال Ethernet، USB، CAN، LonWorks، SATA، I2C و SPI می‌باشند.

ریزپردازنده‌های ATmega16 به صورت سخت‌افزاری قابلیت برقراری ارتباط USART، SPI وI2C را برای ارتباط با وسایل جانبی از قبیل SD card، EEPROM، ADC و DAC دارا می‌باشد. در این آزمایش به ارتباط SPI و I2C پرداخته می‌شود.

SPI

ارتباط سریال SPI یک پروتکل ارتباط سریال سنکرون با سرعت بالا بوده که می‌توان برای برقراری ارتباط بین ریزپردازنده‌های AVR مختلف و وسایل جانبی که در آن‌ها تعریف شده باشد به کار رود، خصوصیات این ارتباط به صورت زیر می‌باشد:

- full-duplex ارسال داده به صورت سنکرون توسط سه سیم
- قرارگیری در حالت های slave و master
- پرچم پایان ارسال و فعال شدن وقفه
- امکان دو برابر کردن سرعت ارسال



شکل ۴۶: ساختار داخلی SPI

در ارتباط SPI تنها دو پایه SDI(Din) و SDO(Dout) برای انتقال داده مورد استفاده قرار می‌گیرند که این پروتکل را برای بسیاری از کاربردها مناسب می‌کند. برای همزمان کردن دو device ای که به وسیله این پروتکل به یکدیگر متصل شده‌اند یک پایه کلاک نیز تعبیه شده است. آخرین پایه‌ی ارتباط SPI پایه CE[22] می‌باشد که برای initiate کردن و terminate کردن انتقال داده مورد استفاده قرار می‌گیرد. این ۴ پایه SCLK، SDO، SDI و CE پروتکل ۴–سیمه SPI را تشکیل می‌دهند. در ریزپردازنده‌های AVR پایه‌های SDI، SDO، SCLK و CE به ترتیب MOSI، MISO، SCK و SS نامیده می‌شوند.

همچنین استاندارد دیگری نیز با نام .3-wire interface bus وجود دارد. در این استاندارد پایه‌های CE و SCLK از یک پایه استفاده می‌کنند. ارتباط SPI ۴–سیمه می‌تواند به یک ارتباط ۳–سیمه تبدیل شود که در آن پایه‌های SDI و SDO با هم ادغام می‌شوند. که البته تفاوت‌های اساسی بین ارتباط SPI ۴–سیمه و ۳–سیمه وجود دارد.

به دلایل مختلف ممکن است لازم باشد تا در یک پروژه از چند ریزپردازنده استفاده نماییم. یکی از راه‌های اتصال دو ریزپردازنده ارتباط SPI است. برای مثال فرض کنید بخواهیم ورودی دیجیتال را از روی یک ریزپردازنده بخوانیم و

[22] Chip enable

برروی LCDای که به ریزپردازنده دیگر متصل است نشان دهیم یک راه برای برقراری این ارتباط استفاده از SPI می‌باشد. در ادامه به بررسی این پروتکل می‌پردازیم.

روش کار پروتکل SPI

اساس کار بر پایه دو شیفت‌رجیستر می‌باشد. ارتباط سریال به این صورت است که یک شیفت‌رجیستر به ورودی شیفت رجیستر دیگر متصل شود. با هر کلاک یک بیت از آن خارج شده به شیفت‌رجیستر سمت راست وارد می‌شود.

ارتباط SPI به دلیل وجود کلاک مشترک از نوع سنکرون است. به عبارتی پروتکل SPI از کلاک برای همگام‌سازی اطلاعات استفاده می‌کند. به وسیله‌ای که کلاک مشترک را تأمین می‌کند MASTER و به وسیله دیگر SLAVE می‌گویند که دارای یک پایه ENABLE (همان پایه SS) نیز هست که باید توسط MASTER فعال گردد. در هر کلاک داده داخل شیفت‌رجیستر شیفت یافته و یک بیت انتقال می‌یابد. همانند شکل ۴۷ در هر کلاک بیت پرارزش انتقال می‌یابد.

پایه‌ای که از شیفت رجیستر خارج می‌شود MOSI[23] و پایه‌ای که به آن وارد می‌شود MISO[24] و پایه کلاک SCLK و پایه فعال ساز SS نام دارند.

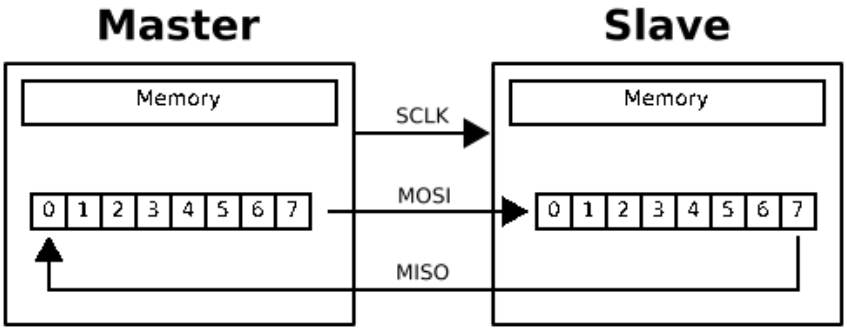
بعد از ارسال ۸ پالس محتوای دو شیفت‌رجیستر جابجا می‌شود به اینگونه که پایه SCLK خروجی کلاک برای MASTER و ورودی کلاک برای SLAVE است. با نوشتن رجیستر داده[۲۵] در MASTER، CPU شروع به تولید کلاک SPI کرده و داده‌ها از پایه MOSI خارج شده و به پایه MOSI در SLAVE وارد می‌شوند. بعد از انتقال کامل داده توسط MASTER، کلاک SPI قطع و پرچم وقفه پایان ارسال داده[26] یک می‌شود و برنامه وقفه اجرا می‌گردد. دو شیفت رجیستر ۸ بیتی در MASTER و SLAVE را می‌توان به عنوان یک شیفت رجیستر چرخشی ۱۶ بیتی در نظر گرفت. این موضوع در شکل ۴۷ دیده می‌شود. زمانی که داده‌ای از MASTER به SLAVE ارسال می‌شود می‌تواند در همان حال در جهت مخالف داده‌ای از SLAVE به MASTER انتقال یابد، به این صورت که در طول هشت کلاک SPI داده‌های MASTER و SLAVE با هم عوض می‌شود.

ارتباط SPI یک ارتباط full duplex است به این معنا که همزمان توانایی ارسال و دریافت داده را دارد.

زمانی که MASTER بخواهد از SLAVE داده دریافت کند SLAVE باید یک بایت داده بر روی شیفت رجیسترش قرار دهد و بعد از ۸ کلاک MASTER آن را دریافت خواهد کرد.

---

[23] MASTER OUT SLAVE IN
[24] MASTER IN SLAVE OUT
[25] SPI DATA REG
[26] SPIF



شکل ۴۷ ارتباط master و slave

جدول3 : جهت پایه‌های SPI

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
SS	User Defined	Input

خواندن و نوشتن در SPI

برایdeviceهایی که از طریق SPI ارتباط برقرار می‌کنند، مانند SD card به ریزپردازنده، ریزپردازنده را به عنوان MASTER و device را به عنوان SLAVE استفاده می‌نماییم و همانطور که گفتیم MASTER تامین کننده کلاک خواهد بود. در شیفت رجیستر بیت پرارزش شیفت رجیستر قبل از همه انتقال می‌یابد. در زمان انتقال پایه CE باید HIGH باشد. داده‌های انتقالی بین MASTER و SLAVE در گروه‌های ۸ تایی انتقال می‌یابند. ابتدا بایت آدرس انتقال می یابد و بلافاصله بعد از آن بایت داده برای تمایز خواندن و نوشتن همیشه در هنگام نوشتن بیت D7 برابر ۱ و در هنگام خواندن D7 برابر صفر می‌باشد.

تنظیمات کلاک و پلاریته

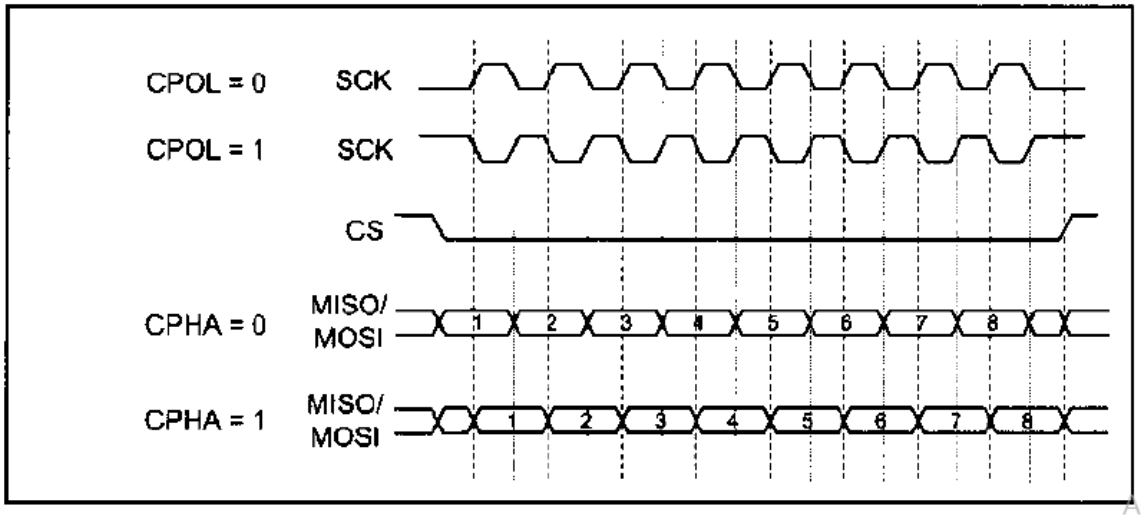
در ارتباط SPI، MASTER و SLAVE(ها) فاز و پلاریته کلاک انتقال داده باید یکسان باشد، که دارای دو حالت CPOL[27] و CPHA[28] می‌باشد. این بیت‌ها هرگونه تنظیم شوند در MASTER و SLAVE باید یکسان باشند. در زمانی که CPOL=0 مقدار اولیه پایه ساعت صفر است و زمانی که CPOL=1 مقدار اولیه پایه ساعت یک است. در زمانی که CPHA=0 است به این معنی می‌باشد که نمونه‌گیری از MISO و MOSI در اولین لبه کلاک انجام می گیرد و زمانی

[27] Clock Polarity
[28] Clock Phase

که CPHA=1 باشد یعنی نمونه‌گیری در لبه دوم کلاک انجام می‌گیرد با توجه به مقادیر این دو بیت می‌توان جدول زیر را نتیجه‌گیری کرد.

جدول ۲۴: پلاریته SPI

CPOL	CPHA	Leading edge	Trailing edge	SPI Mode
0	0	Sample(rising)	Setup(falling)	۰
0	1	Setup(rising)	Sample(falling)	۱
1	0	Sample(falling)	Setup(rising)	۲
1	1	Setup(falling)	Sample(rising)	۳



شکل ۴۸: زمان‌بندی کلاک و پلاریته

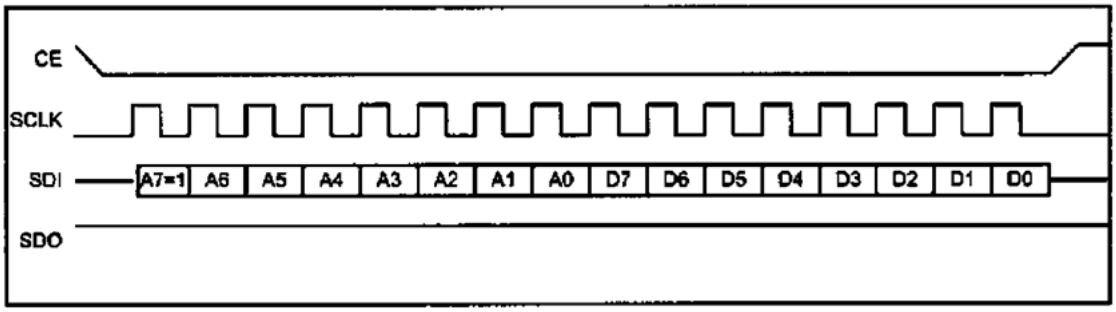
نوشتن یک تک بایت

با انجام مراحل زیر می‌توان یک بایت تکی را بر روی Device ارتباط SPI نوشت:

۱. برای شروع نوشتن CE=0 قرار می‌دهیم.
۲. ۸ بیت آدرس را در شیفت رجیستر قرار می‌دهیم و هر کلاک یک بیت داده شیفت می‌یابد. و باید در نظر داشت برای قرار گرفتن در مد نوشتن باید A7=1 باشد. (به یاد داشته باشید که بیت پرارزش A7 قبل از همه ارسال می‌شود)
۳. بعد از ارسال آدرس، SLAVE انتظار دریافت بلافاصله‌ی داده را دارد.

۴. ۸ بیت داده را در شیفت رجیستر قرار می‌دهیم و هر کلاک یک بیت داده شیفت می‌یابد. و باید در نظر داشت برای قرار گرفتن در مد نوشتن A7 باید ۱ باشد.

۵. برای نشان دادن سیکل نوشتن داده، CE را برابر یک قرار می‌دهیم.



شکل ۴۹ : نوشتن داده تک بایت

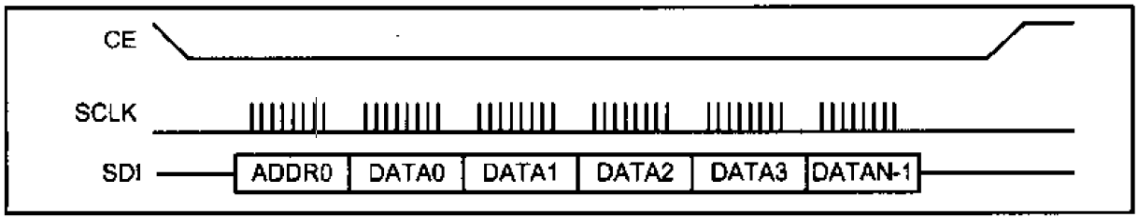
نوشتن چند بایت داده BURST

۱. برای شروع نوشتن CE=0 قرار دهیم.

۲. ۸ بیت آدرس اولین محلی که می‌خواهیم داده‌ها را در شیفت رجیستر قرار می‌دهیم و در هر کلاک یک بیت داده شیفت می‌یابد. A7 را برابر ۱ قرارمی‌دهیم.

۳. ۸ بیت داده از آدرس تعیین‌شده قرار می‌گیرند و در هر کلاک یک بیت داده انتقال می‌یابد. در طول ارسال داده، CE باید صفر بماند.

۴. در پایان ارسال CE را برابر ۱ قرار می‌دهیم.



شکل ۵۰: نوشتن داده BURST

مراحل خواندن داده از روی SPI

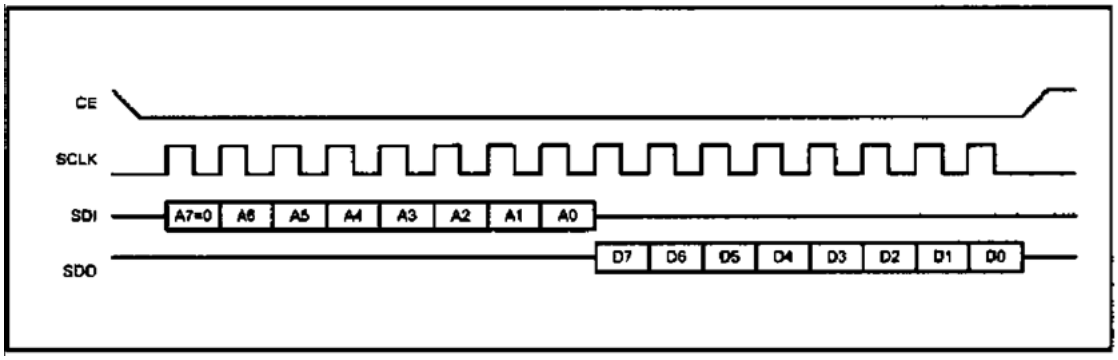
۱. برای شروع خواندن CE=0 قرار دهیم.

۲. ۸ بیت آدرس در شیفت رجیستر در MASTER قرار داده می‌شود و در هر زمان یک شیفت پیدا می‌کند. برای خواندن از روی باس A7 باید 0 باشد.

۳. بعد از ارسال ۸ بیت آدرس و خوانده شدن آن توسط SLAVE باید داده‌های مربوط به آن آدرس توسط SLAVE ارسال شود.

۴. ۸ بیت داده را در شیفت‌رجیستر قرار می‌دهیم تا شیفت یابند.

۵. برای نشان دادن پایان سیکل خواندن CE را برابر ۱ قرار می‌دهیم.



شکل ۵۱:خواندن داده از روی SPI

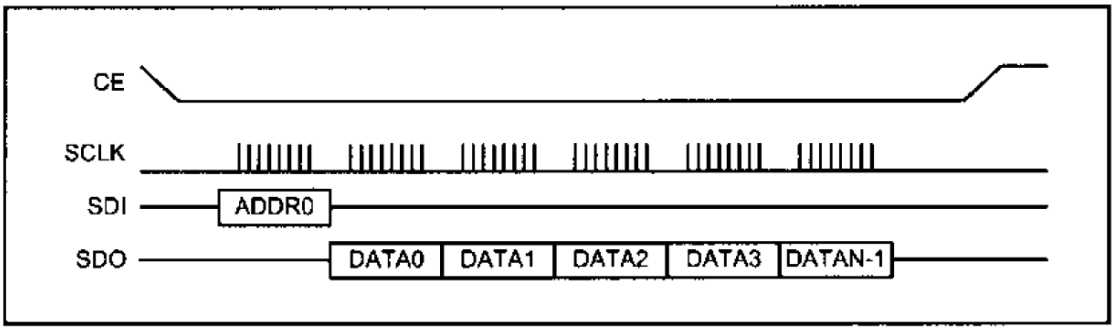
مراحل خواندن داده BURST از روی SPI

۱. برای شروع خواندن CE=0 قرار دهیم.

۲. 8 بیت آدرس اول در شیفت رجیستر در MASTER قرار داده می‌شود و در هر زمان یک شیفت پیدا می‌کند. برای خواندن از روی باس A7 باید 0 باشد. بعد از ارسال ۸ بیت آدرس اول و خوانده شدن آن توسط SLAVE.

۳. باید داده‌های مربوط به آدرس شروع توسط SLAVE ارسال شود. در تمام طول مدت ارسال CE باید صفر باشد.

۴. برای نشان دادن پایان سیکل خواندن CE را برابر ۱ قرار می‌دهیم.



شکل ۵۳:خواندن داده BURST از روی SPI

برنامه نویسی SPI بر روی ریزپردازنده های AVR

در ریزپردازنده‌های AVR پایه‌های مربوط به ارتباط SPI در پورت B قرار دارند.

ریزپردازنده‌های AVR حاوی ۲۴ ثبات I/O هستند که ۳ ثبات از آن مربوط به واسط سریال SPI است که توضیحات آن به شرح زیر می‌باشد:

SPDR: همان ثبات شیفت‌رجیستر است. داده‌ای که می‌خواهیم انتقال دهیم در این ثبات نوشته می‌شود و داده‌های دریافت‌شده از این ثبات خوانده می‌شود.

SPCR: ثبات کنترلی SPI است و تنظیمات مورد نظر مانند فرکانس کلاک، جهت شیفت و MASTER و SLAVE بودن توسط بیت‌های این ثبات تعیین می‌شود.

جدول ۴ ثبات SPCR

SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

- SPIE: فعال کردن قابلیت وقفه
- SPE: فعال کردن سخت افزار SPI
- DORD: جهت شیفت. اگر صفر باشد انتقال داده به سمت راست یعنی کم‌ارزشترین بیت ارسال می‌گردد. اگر یک باشد انتقال به سمت چپ صورت می‌گیرد. نکته مهم در این بیت یکسان بودن آن در MASTER و SLAVE است.
- MSTR: تعیین MASTER و SLAVE بودن برای MASTER بودن این بیت باید ۱ باشد.
- CPOL & CPHA: تعیین‌کننده ضرایب فازکلاک و پلاریته کلاک
- SPR0 & SPR1: ضریب کاهش فرکانس Prescaler را تعیین می‌کنند.

جدول ۲۶ ضرایب prescale

SPR1	SPR0	SPI Frequency
0	0	Fu/4
0	1	Fu/16
1	0	Fu/64
1	1	Fu/128

SPSR: ثبات وضعیت SPI بیت پرچم انتقال در این ثبات قرار دارد.

جدول ۵ ثبات SPSR

SPI	WCO	-	-	-	-	-	SPI2X

- SPIF: در SPI پس از اتمام ارسال یا دریافت یک بایت این پرچم ۱ می‌شود اگر بیت SPIE یک باشد فعال شدن این پرچم باعث فراخوانی زیر روال مربوط به SPI می‌شود.
- WCOL: اگر در حین انتقال داده، داده جدیدی در رجیستر داده SPI نوشته شود این بیت ۱ می‌گردد اگر این بیت ۱ باشد با خواندن SPSR بیتهای WCOL وSPIF پاک می‌شوند و سپس دسترسی به رجیستر داده صورت می‌گیرد.
- SPI2X: زمانی که SPI در حالت MASTER باشد با یک کردن این بیت سرعت کلاک دو برابر خواهد شد.

در ریزپردازنده‌های AVR به محض اینکه یک داده جدید در ثبات SPDR نوشته می‌شود سخت‌افزار داخلی شروع به کار می‌کند با ۸ پالس ساعت داده‌ها را ارسال کرده سپس متوقف شده و پرچم SPIF یک می‌شود. برای ارسال دستور تنها باید مقدار داده در SPDR نوشته شده و پایه فعال ساز SS در ابتدای هر ارسال صفر و در انتهای ارسال یک گردد و در MASTER برای اینکه قبل از ارسال داده قبلی داده جدیدی در SPDR نوشته نشود باید پرچم SPIF قبل از نوشتن چک شود و اگر صفر بود منتظر شویم تا یک گردد.

بعد از آشنایی با این پروتکل به حل مسئله‌ای که مطرح شد می‌پردازیم.

اتصال دو ریزپردازنده از طریق SPI

انجام تنظیمات اولیه ارتباط سریال SPI در CodeWizard

در CodeWizard بر روی لبه SPI کلیک کنید. در اینجا می‌توانید ویژگی‌های ارتباط SPI را تنظیم کنید. با انتخاب گزینه Enable SPI ارتباط در داخل ریزپردازنده برقرار می‌گردد.

اگر می‌خواهید پس از اتمام ارسال، یک وقفه تولید گردد باید گزینه SPI Interrupt را علامت بزنید.

- SPI clock rate
  کلاک استفاده‌شده در پایه SCK را برای ارسال داده‌ها مشخص می‌کند.
- Clock Phase
  موقعیت لبه سیگنال SCK را نسبت به بیت داده مشخص می‌کند.
- SPI Type
  حالت عملکرد SPI را به صورت MASTER یا SLAVE مشخص می‌کند.

- DATA Order

ترتیب ارسال بیت‌های داده را تعیین می‌کند.

انتخاب گزینه clock rate 2x سرعت کلاک SPI را دو برابر می‌کند.

برنامه کاربردی

به کمک تابع ()spi، می‌توان یک بایت را از طریق Polling روی باس SPI ارسال نمود و به صورت همزمان بایت دیگری را دریافت کرد ولی نکته مهم در ارسال داده تشخیص ابتدا و انتهای بسته‌ی داده است که چگونه آن‌ها را بفرستد و پردازش نماید. یک روش مطمئن برای انجام این کار تبدیل داده‌ها به کاراکترهای معادل و ارسال رشته‌های کارکتری روی باس SPI می‌باشد.

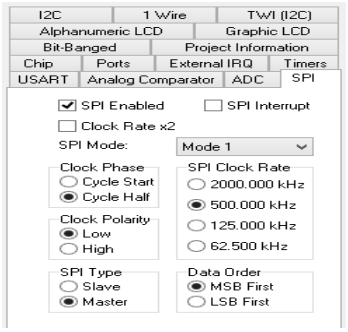
برای ارسال و دریافت کاراکتر دو تابع ()putchar و ()getchar را طوری بازتعریف می‌کنیم که بتوانند این کار را انجام دهند. برای نمونه تابع ()putchar را به صورت زیر تعریف می‌کنیم:

```
#define _ALTERNATE_PUTCHAR_
#pragma used+
void  putchar(char c)
{
     spi(c);
}
#pragma used-
```

با تعریف مجدد این تابع از آنجایی که توابع printf و puts و putsf از این تابع استفاده می‌کنند نحوه عملکرد آن‌ها نیز تغییر می‌یابد که در برنامه از آن‌ها استفاده خواهیم کرد.

انجام تنظیمات اولیه در کدویژن در MASTER

فرکانس کاری تراشه را بر روی 8MHz و پورت A را به صورت ورودی قرار می‌دهیم. سپس به دلیل اینکه در MASTER هستیم باید پایه‌های MOSI، SCK و SS واقع در پورت B را به صورت خروجی و با مقدار اولیه صفر قرار دهیم. سپس در سربرگ SPI با فعال کردن آن پارامترهای آن را مانند زیر تنظیم نماییم:



شکل ۵۴

تکمیل کد برنامه MASTER:

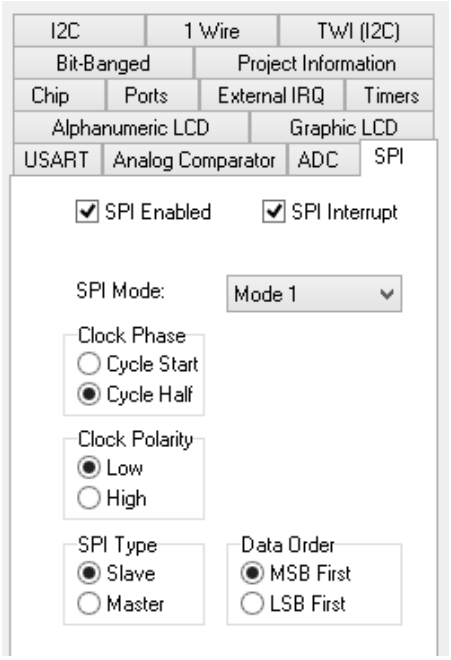
```
#include <mega16.h>
#include <delay.h>
#include <spi.h>
#include <stdio.h>
#define _ALTERNATE_PUTCHAR_
#pragma used+
void  putchar(char c)
{
      spi(c);
}
#pragma used-

char str[40];
void main(void)
{
      // Declare your local variables here
      PORTA=0x00;
      DDRA=0x00;
      PORTB=0x00;
      DDRB=0xB0;
      PORTC=0x00;
      DDRC=0x00;
      PORTD=0x00;
      DDRD=0x00;

      // SPI initialization
      // SPI Type: Master
      // SPI Clock Rate: 500.000 kHz
      // SPI Clock Phase: Cycle Half
      // SPI Clock Polarity: Low
      // SPI Data Order: MSB First
      SPCR=0x55;
      SPSR=0x00;

      while (1)
       {
```

```
            delay_ms(100);
            sprintf(str,"#PORTA=%03u \r",PINA);
            puts(str);
        }
}
```

در این برنامه ابتدا تابع putchar() برای ارسال کاراکترها روی باس SPI از نو تعریف می‌شود. مقادیر موجود روی پورت خوانده می‌شود و در یک رشته کاراکتر قرار می‌گیرد. سپس این رشته به کمک تابع puts() روی باس SPI فرستاده می‌شود. برای نشان دادن ابتدا و انتهای رشته کاراکتری به ترتیب با '#' و '\r' مشخص می‌شوند. در slave از آن‌ها استفاده خواهیم نمود.

انجام تنظیمات اولیه در کدویژن در SLAVE

در slave پایه MISO باید به صورت خروجی تنظیم گردد. به پورت A نیز یک LCD کاراکتری متصل می‌نماییم. تنظیمات فرکانس تراشه و SPI در slave مشابه با master می‌باشد با این تفاوت که وقفه SPI نیز فعال است.



شکل ۵۵

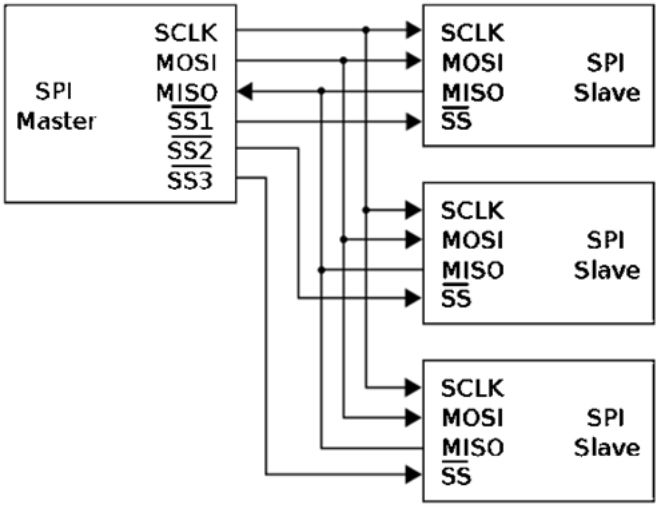
```
#include <mega16.h>
#include <alcd.h>
#include <delay.h>
#include <stdio.h>
char sentdata=0;
char buffer[32],wr_index=0;
char str[32];
interrupt [SPI_STC] void spi_isr(void)
```

```
{
      unsigned char data;
      data=SPDR;
      #asm("sei")
      if(data=='#')
      {
            buffer[0]=data;
            wr_index=1;
      }
      else if (wr_index>=1 && data!='\r')
      {
            buffer[wr_index]=data;
           wr_index++;
      }
      else if(data=='\r')
            sentdata=1;
      else
           wr_index=0;

}

void main(void)
{

      PORTB=0x00;
      DDRB=0x40;

      // SPI initialization
      // SPI Type: Slave
      // SPI Clock Phase: Cycle Half
      // SPI Clock Polarity: Low
      // SPI Data Order: MSB First
      SPCR=0xC5;
      SPSR=0x00;
      // Clear the SPI interrupt flag
      #asm
      in   r30,spsr
      in   r30,spdr
      #endasm

      lcd_init(16);

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

      while (1)
      {
            if(sentdata)
            {
                  lcd_clear();
                  lcd_gotoxy(0,0);
                  sprintf(str,"%s",buffer);
```

```
                lcd_puts(str);
                sentdata=0;
            }
        }
}
```

در برنامه فوق با دریافت هر کاراکتر توسط SPI، وقفه SPI تولید می‌شود و روتین وقفه اجرا می‌گردد. در وقفه مقدار کاراکتر گرفته‌شده در متغیر data ریخته می‌شود اگر کاراکتر دریافتی '#' باشد بیانگر ابتدای رشته و در جایگاه اول بافر قرار می‌گیرد و '\r' بیانگر پایان رشته است زمانی که دریافت شود بر روی LCD قرار می‌گیرد.

اتصال بیش از دو ریزپردازنده توسط SPI



شکل ۵۲ نحوه‌ی اتصال بیش از دو ریزپردازنده توسط SPI

ریزپردازنده‌های AVR تنها دارای یک واحد سخت-افزاری SPI و یک پورت SPI هستند و بنابراین برای اتصال بیش از دو ریزپردازنده مانند شکل روبرو عمل می‌کنیم.

باید توجه شود که در هر لحظه تنها یک ریزپردازنده می تواند MASTER باشد و بقیه باید به صورت SLAVE باشند. برای فعال سازی SLAVEها پایه‌های جداگانه باید در نظر گرفت. تنها عامل محدودکننده در تعداد وسایلی که می‌توانند از طریق SPI به یکدیگر متصل شوند ملاحظات الکترونیکی است.

ارتباط سریال دو سیمه (I2C یا TWI)

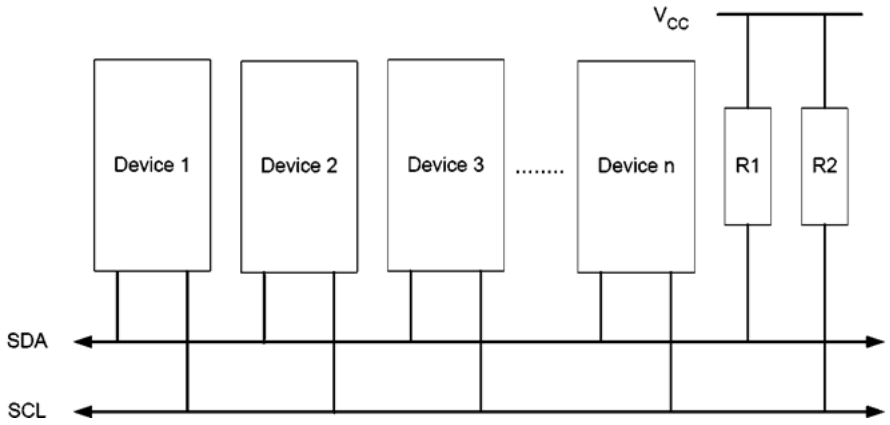
در این قسمت در مورد ارتباط I2C صحبت می‌کنیم. I2C یک پروتکل ارتباط سریال برای سنسورها و بعضی از EEPROM می‌باشد.

ارتباط سریال دو سیمه یک پروتکل ارتباطی سریال است که توسط شرکت Philips ارایه شده است اما در سال‌های اخیر به طور گسترده توسط بسیاری از کمپانی‌ها استفاده می‌شود. این پروتکل برای اتصال لوازم جانبی کم سرعت به motherboard یا embedded system ها مورد استفاده قرار می‌گیرد. I2C یک ارتباط connection-oriented به همراه acknowledge را فراهم می‌کند و تنها از ۲ پایه برای ارتباط استفاده می‌کند.

عموماً واسط ارتباط دو سیمه برای کار با ریزپردازنده‌ها مناسب است. پروتکل TWI این امکان را به ما می‌دهد تا حداکثر ۱۲۸ وسیله مختلف را تنها با استفاده از دو خط باس دو طرفه، یکی برای پالس ساعت (SCL) و دیگری برای داده (SDA)، به یکدیگر متصل کند. تنها سخت افزار خارجی که برای ایجاد این باس مورد نیاز است یک مقاومت بالاکش[۲۹] برای هر یک از خطوط باس است. تمامی وسایل متصل به باس، آدرس‌های خاص خود را دارند و نحوه ارتباط بیت آن‌ها نیز توسط پروتکل TWI مشخص می‌شود.

در تمامی وسایلی که از TWI حمایت می‌کنند، درایورهای باس بصورت open-drain یا open collector می‌باشند. این ویژگی موجب می‌شود تا آن‌ها بصورت wire-AND عمل کنند، که این برای عملکرد صحیح باس ضروری است. بنابراین یک سطح پایین در خط باس TWI، زمانی تولید می‌شود که خروجی یک یا چند وسیله صفر باشد و سطح بالای آن نیز تنها زمانی که تمام وسایل TWI در حالت امپدانس بالا باشند، حاصل می‌گردد.

تعداد وسایل مجزا برای اتصال به یک باس تنها توسط محدودیت ظرفیت باس و نیز فضای آدرس ۷ بیتی slave مشخص می‌گردد.



شکل ۵۳: اتصال بیش از دو ریزپردازنده توسط TWI

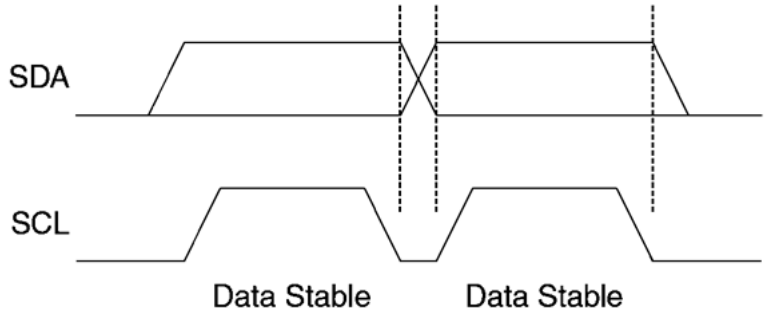
I2C می‌تواند تا ۱۲۰ وسیله مختلف را به یکدیگر متصل کند. هرکدام از این device های متصل‌شده یک node خوانده می‌شوند که هر نود می‌تواند master یا slave باشد. وظیفه تولید کلاک سیستم بر عهده‌ی master می‌باشد و slave نودی است که کلاک و آدرس را توسط master دریافت می‌کند در I2C هم master هم slave توانایی ارسال و دریافت داده را دارند. پس در کل ۴ مد برای ارسال و دریافت وجود دارد که عبارتند از master transmitter، master receiver، slave transmitter و slave receiver که هر نود می‌تواند بیش از یک مد داشته باشد اما در هر لحظه از زمان تنها در یک مد می‌تواند ایفای نقش کند.

[29] pull up

ارسال داده‌ها و فرمت فریم‌ها

به ازای هر بیت داده که بر روی باس TWI توسط SDA فرستاده می‌شود، یک پالس در خط کلاک آن را همراهی می‌کند.

زمانی که خط کلاک بالاست، سطح خط داده باید ثابت باقی بماند و تنها زمانی سطح داده می‌تواند تغییر کند که خط کلاک در حالت low باشد. تنها استثنای این قانون در تولید حالت‌های شروع و توقف است.



شکل ۵۳: تغییرات داده در TWI

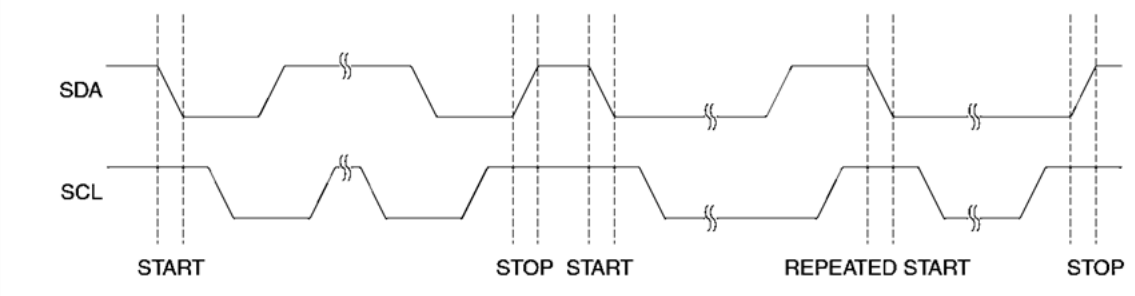
حالت‌های شروع (start) و توقف (stop)

شروع و پایان ارسال داده توسط Master صورت می‌گیرد، زمانی که Master، حالت شروع را روی باس ایجاد می‌کند ارسال آغاز می‌شود و زمانی که حالت توقف را ایجاد می کند، ارسال پایان می‌پذیرد. حالت شروع و پایان را تنها می‌توان در زمانی که پایه کلاک high است ایجاد کرد. حالت شروع زمانی رخ می‌دهد که خط کلاک (SCL) در حالت high بوده و خط داده (SDA) دارای لبه پایین رونده باشد و حالت پایان زمانی ایجاد می‌شود که خط داده در لبه بالا رونده باشد.

در بین حالت‌های شروع و توقف، باس مشغول در نظر گرفته می‌شود و Master دیگری نباید سعی در کنترل باس نماید. یک حالت خاص زمانی رخ می‌دهد که در بین یک حالت شروع و توقف، حالت شروع جدیدی ایجاد شود. به این حالت، حالت شروع مکرر[۳۰] گفته می‌شود و زمانی اتفاق می‌افتد که Master بخواهد بدون از دست دادن کنترل باس ارسال جدیدی آغاز کند. بعد از یک شروع مکرر، باس تا حالت رسیدن به حالت توقف بعدی، مشغول در نظر گرفته می‌شود.

همانطور که در شکل زیر نشان داده شده است، حالت‌های شروع و توقف با تغییر سطح خط SDA در زمانی که خط SCL بالاست، انجام می‌شود.

[30] Repeated start



شکل ۵۴ حالت‌های شروع (start) و توقف (stop)

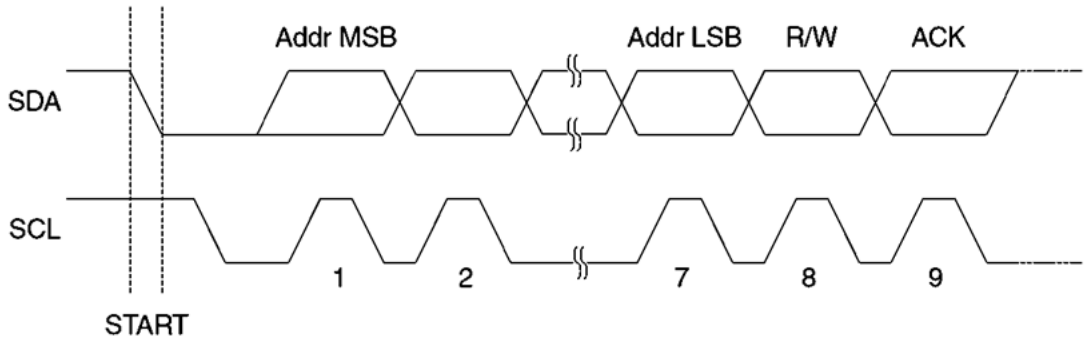
فرمت بسته‌ها

در پروتکل I2C هر داده یا آدرسی که بخواهد انتقال پیدا کند باید در بسته قرار گیرد. تمامی بسته‌های آدرس فرستاده‌شده روی باس TWI، ۹ بیت طول دارند که ۸ بیت اول در SDA توسط transmitter قرار می‌گیرد و بیت ۹ ام یک بیت کنترل خواندن/نوشتن و یک بیت تصدیق (Acknowledge) است که توسط receiver بر روی SDA ارسال می‌شود. برای گرفتن Acknowledge، transmitter در کلاک ۹ ام خط SDA را رها می‌کند تا receiver بتواند برای نشان دادن ACK باس را low کند که اگر این اتفاق رخ ندهد Acknowledge به عنوان NACK تلقی می‌شود. برای برقراری یک ارتباط کامل روند زیر باید طی شود.

فرمت بسته آدرس

تمامی بسته‌های آدرس فرستاده شده بر روی باس TWI، ۹ بیت طول دارند که هفت بیت آدرس، یک بیت کنترل خواندن/نوشتن و یک بیت تصدیق (Acknowledge) تشکیل شده اند.

اگر بیت خواندن/نوشتن یک شود، پس از آن عمل خواندن انجام می‌شود و در غیر اینصورت عمل نوشتن انجام می‌گیرد. زمانی که Slave تشخیص دهد که آدرس روی باس به آن تعلق دارد، باید در سیکل نهم SCK، با زمین کردن SDA با آن پاسخ دهد. اگر Slave آدرس‌دهی‌شده مشغول باشد یا به هر دلیلی نتواند به درخواست Master پاسخ دهد خط SDA باید در سیکل کلاک ACK بالا باقی بماند. پس از آن Master می‌تواند یک حالت توقف، یا یک حالت شروع مکرر را برای آغاز یک ارسال مجدد بفرستد.

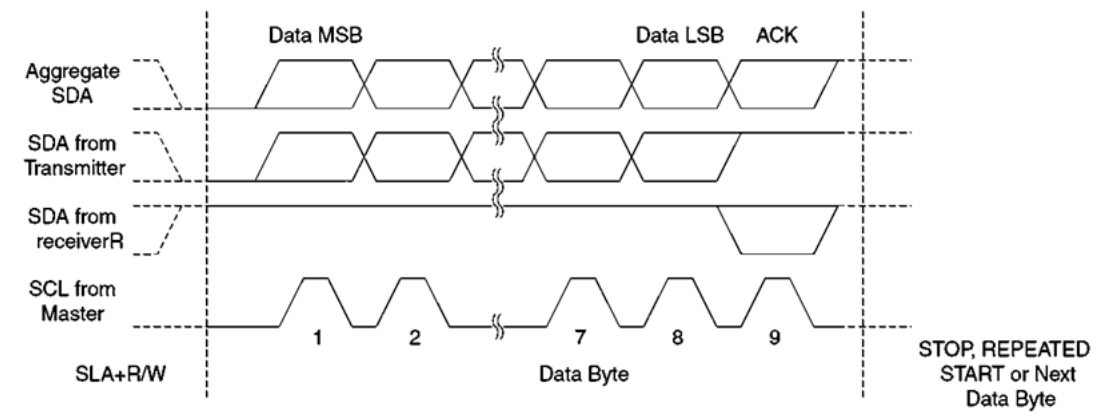


شکل ۵۵: فرمت بسته آدرس

فرمت بسته داده

یک بسته آدرس که شامل یک آدرس Slave و یک بیت خواندن و نوشتن باشد، به ترتیب به صورت SLA+R و SLA+W نشان داده می‌شود.

در ارسال آدرس Slave ابتدا بیت MSB ارسال می‌گردد. آدرس‌های Slave می‌توانند هر یک از مقادیر ۱ تا ۱۲۷ را به خود بگیرند ولی از آدرس صفر برای فراخوانی عمومی استفاده می‌گردد. زمانی که یک فراخوانی عمومی انجام می‌شود، باید تمامی Slave ها در سیکل ACK با زمین کردن خط SDA به آن پاسخ دهند. هنگامی که Master بخواهد یک پیغام را برای تمامی Slave های موجود ارسال کند، از یک فراخوانی عمومی استفاده می‌کند. در صورتی که یک آدرس فراخوانی عمومی و به دنبال آن یک بیت نوشتن ارسال گردد، تمامی Slave هایی که می‌توانند به فراخوانی پاسخ دهند، در سیکل ACK خط SDA را زمین می‌کنند. در این صورت بسته‌های داده بعدی توسط تمامی Slave هایی که به فراخوانی عمومی پاسخ داده‌اند، دریافت می‌گردد. باید توجه شود که ارسال آدرس فراخوانی عمومی و به دنبال آن، یک بیت خواندن، بدون معناست. چرا که در اینصورت چندین Slave داده‌های مختلفی را بر روی باس قرار می‌دهند که باعث مختل شدن باس می‌شود. تمام بیت‌های داده که بر روی باس TWI ارسال می‌شوند ۹ بیتی هستند که شامل یک بایت داده و یک بیت تصدیق (Acknowledge) می‌باشند.



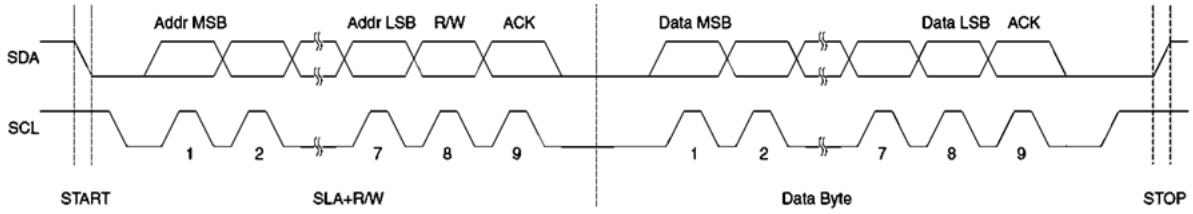
شکل ۵۶: فرمت بسته داده

نحوه تبادل داده‌ها

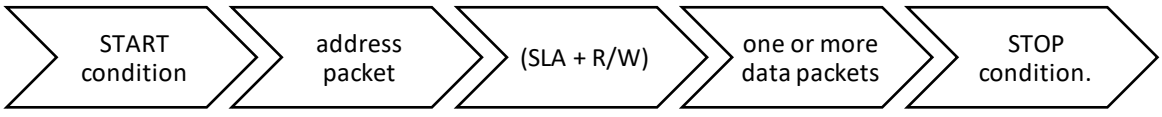
در حین ارسال داده، Master کلاک و حالت‌های شروع و توقف را تولید می‌کند، در صورتی که دریافت‌کننده تنها باید دریافت داده را تصدیق کند. یک تصدیق توسط دریافت‌کننده در سیکل نهم SCL و با زمین کردن خط SDA اعلان می‌گردد.

اگر دریافت‌کننده دریافت را بدرستی انجام ندهد، خط SDA را در حالت بالا رها می‌کند (NACK). زمانی که دریافت‌کننده آخرین بایت را دریافت کند یا به هر دلیلی نتواند بایت دیگری را دریافت کند، باید یک NACK بعد از آخرین بایت ارسال کند.

اصولاً یک ارسال، از یک حالت شروع، یک SLA+R/W و یک یا چند بسته داده تشکیل شده است. یک پیغام تهی که از یک حالت شروع و به دنبال آن یک حالت توقف تشکیل شده باشد غیر قانونی است.



شکل ۵۷: نحوه تبادل داده‌ها



به طور مثال برای نوشتن داده ۱۰۱۰۱۰۱۰ به یک slave با آدرس ۱۰۱۱۰۱۱ باید مراحل زیر طی شود:

۱. Master در زمانی که کلاک در حالت high قرار دارد یک لبه پایین رونده روی SDA برای ایجاد شرایط start قرار دهد.
۲. Master مقدار ۱۰۱۱۰۱۱۰ را روی باس قرار می‌دهد. ۷ بیت اول متعلق به آدرس slave مورد نظر و آخرین بیت که صفر است مشخص کننده عملیات نوشتن از طرف Master می‌باشد.
۳. Slave به عنوان پاسخ ACK، باس را به نشانه آمادگی برای دریافت داده صفر می‌کند.
۴. بعد از دریافت ACK، Master داده ۱۰۱۰۱۰۱۰ را روی باس قرار می‌دهد توجه داشته باشید که بیت MSB اول از همه ارسال می‌شود.
۵. زمانی که slave داده را دریافت نمود خط SDA را در حالت high (NACK) رها می‌کند. این به Master نشان می‌دهد که داده دریافت شده و به داده بیشتری نیاز نیست.
۶. بعد از دریافت NACK، Master متوجه می‌شود به داده بیشتری نیاز نیست و در نتیجه زمانی که کلاک در حالت high قرار دارد در SDA یک شرایط STOP ایجاد می‌کند و باس را آزاد می‌نماید.

نوشتن داده Burst

در این نوع نوشتن آدرس اولین مکان قرار داده می‌شود و بقیه داده‌ها آن آدرس را دنبال می‌کنند. در این مد I2C device به طور خوکار آدرس را تا زمانی که به حالت stop برسد افزایش می‌دهد.

۱. ایجاد حالت شروع
۲. آدرس slave به همراه یک صفر در انتها برای نشان دادن عملیات نوشتن
۳. ارسال آدرس اولین مکان در slave
۴. ارسال اولین داده که آدرس آن در مرحله قبل ارسال شد و در ادامه ارسال بقیه داده‌ها
۵. ایجاد حالت پایان

جدول ۶: نوشتن داده Burst

Start	Slave address	write	ACK	First location address	ACK	Data byte #1	ACK	Data byte #2	ACK	Data byte #3	ACK	Stop
S	1110110	0	A	00001111	A	00000001	A	00000010	A	00000011	A	P

خواندن داده Burst

همانند حالت نوشتن، آدرس اولین مکان قرار داده می‌شود و بقیه داده‌ها آن آدرس را دنبال می‌کنند. در این مد I2C device به طور خوکار آدرس را تا زمانی که به حالت stop برسد افزایش می‌دهد.

۱. ایجاد حالت شروع
۲. آدرس slave به همراه یک صفر در انتها برای نشان دادن عملیات نوشتن آدرس مکان اولین داده در Slave که قرار است خوانده شود.
۳. ارسال آدرس اولین مکان در slave
۴. ایجاد حالت شروع تکرار شونده (REPEATED START)
۵. قرار دادن آدرس slave (توسط خود slave برای اینکه نشان دهد چه کسی در حال انتقال داده است) به همراه یک ۱ در انتها برای نشان دادن عملیات خواندن
۶. خواندن داده از اولین مکان و ارسال داده‌ها و افزایش آدرس آن‌ها توسط slave
۷. ایجاد حالت پایان

جدول ۷: خواندن داده Burst

Start	Slave address	write	ACK	First location address	ACK	Start	Slave address	read	Data byte #1	ACK	Data byte #2	ACK	Data byte #3	ACK	Stop
S	1110110	0	A	00001111	A	S	1110110	1	00000001	A	00000010	A	00000011	A	P

توضیح رجیسترهای I2C

(TWI bit rate register) TWBR

این رجیستر فاکتور تقسیم مربوط به تولیدکننده نرخ بیت را انتخاب می‌کند. تولیدکننده نرخ بیت یک تقسیم‌کننده فرکانسی است که در حالت Master فرکانس کلاک SCL را تولید می‌کند.

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	

(TWI Control Register) TWCR

TWINT: TWI Interrupt Flag: این بیت توسط سخت‌افزار و پس از اتمام عملیات TWI یک می‌شود.

TWEA: TWI Enable Acknowledge Bit: این بیت تولید پالس تصدیق را کنترل می‌کند اگر این بیت ۱ باشد پالس ACK روی باس TWI تولید می‌شود.

TWSTA: TWI START Condition Bit: زمانی که یک وسیله در حالت Master باشد کاربر می‌تواند با نوشتن یک در بیت TWSTA حالت شروع ایجاد می‌کند.

TWSTO: TWI STOP Condition Bit: با نوشتن یک در بیت TWSTO در حالت Master یک حالت توقف روی باس TWI ایجاد می‌شود.

TWWC: TWI Write Collision Flag: زمانی که سعی شود تا با وجود صفر بودن TWINT داده‌ای در داخل رجیستر داده نوشته شود این بیت یک می‌گردد.

TWEN: TWI Enable Bit: واسط TWIرا فعال می‌کند.

Res: Reserved Bit

TWIE: TWI Interrupt Enable: زمانی که این بیت و بیت I واقع در رجیستر SREG یک شود درخواست وقفه TWI مادامی که پرچم TWINT یک باشد فعال می‌گردد.

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	

شکل ۶۳: TWCR

TWSR – TWI Status Register

TWS: TW status: این پنج بیت وضعیت منطقی TWI و باس سریال را نشان می‌دهند.

TWPS: این بیت ها می‌توانند خوانده یا نوشته شوند و تقسیم کننده نرخ بیت را کنترل کنند.

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	

شکل ۶۴: TWSR

جدول ۳۰: ضرایب prescale

TWPS1	TWPS0	Prescaler Value
0	0	1
0	1	4
1	0	16
1	1	64

TWDR–TWI Data Register

در حالت ارسال، رجیستر TWDR باید بایت بعدی که باید ارسال شود را در خود نگه دارد و در حالت دریافت باید آخرین بایت دریافت‌شده را در خود جای دهد.

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	

شکل۶۵: TWDR

TWAR –TWI (Slave) Address Register

این رجیستر باید با یک آدرس ۷ بیتی که آدرس وسیله است پر شود.

TWA: این ۷ بیت آدرس SLAVE را درخود نگاه می‌دارند.

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	

شکل ۶۶: TWAR

اکنون برای درک بهتر ارتباط I2C به بررسی این ارتباط بین ریزپردازنده و حافظه EEPROM می‌پردازیم:

حافظه EEPROM

حافظه‌های EEPROM به طور کلی به دو دسته موازی و سریال تقسیم می‌گردند. از EEPROM های موازی می‌توان از خانواده حافظه‌های 28CXX نام برد. زمانی که برای خواندن و نوشتن داده بر روی حافظه به سرعت بالایی نیاز باشد از حافظه های موازی استفاده می‌شود.

دسته دیگر از حافظه‌های EEPROM حافظه‌های سریال هستند و در مکان‌هایی که نیاز به سرعت بالا نباشد و در تعداد پایه‌ها و حجم مدار محدودیت باشد از آن‌ها استفاده می‌شود.

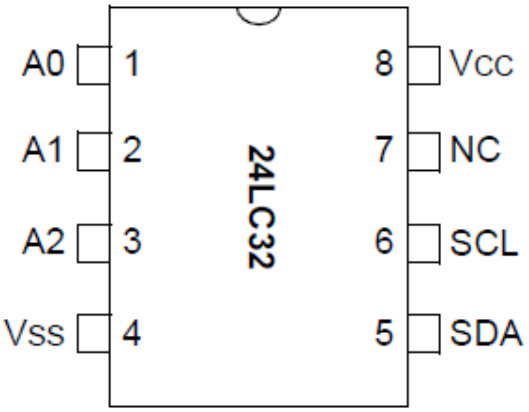
انواع مختلفی از EEPROM های سریال در بازار وجود دارد که برای برقراری ارتباط سریال از پروتکل‌های SPI ،I2C Micro wire و یک سیمه استفاده می‌کنند.

در این آزمایش به بررسی 24LC32 که به صورت I2C کار می‌کند خواهیم پرداخت.

ویژگی‌های 24LC32

- قابلیت ارتباط سریال دو سیمه (I2C)
- کار با فرکانس 100KHz (با ولتاژ 2.5V) و 400KHz (با ولتاژ 5V)
- مجهز به اشمیت تریگر و فیلتر در ورودی برای حذف نویز
- حفظ اطلاعات برای بیشتر از ۲۰۰ سال

24LC32 یک EEPROM با قابلیت پاک کردن به صورت الکتریکی است که برای کاربردهایی که ولتاژ آن‌ها در بازه 2.5 تا 6 ولت است مناسب است.



شکل ۵۸: پایه‌های 24LC32

معرفی پایه‌ها

پایه SCL (serial clock): از لبه مثبت کلاک این پایه برای ورود داده به تراشه‌ی EEPROM و از لبه منفی آن برای خروج داده از EEPROM استفاده می‌گردد.

پایه SDA (serial data): پایه SDA برای انتقال دو طرفه‌ی داده های به صورت سریال استفاده می‌شود این پایه به صورت open drain است.

پایه‌های A0، A1، A2: این پایه‌ها برای زمانی که چند device در ارتباط دو سیمه وجود دارند استفاده می‌شود. مقادیر واردشده به این پایه‌ها، آدرس deviceای را که در EEPROM به آن نگاشت شده است را مشخص می‌کند. یک device مشخص آدرس خود را با (A2, A1, A0) در بایت کنترل مقایسه می‌کند اگر یکسان بودند مشخص می‌شود آدرس متعلق به همین device است.

نحوه عملکرد

در ابتدا توجه به این نکات ضروری است که:

- شروع انتقال داده تنها زمانی صورت می‌گیرد که باس آزاد باشد.
- معمولاً پایه SDA توسط یک وسیله خارجی بالا نگه داشته می‌شود.
- داده موجود روی پایه SDA تنها زمانی که SCL پایین است تغییر می‌کند. در حین انتقال زمانی که کلاک در حالت high است خط داده باید stable باشد .
- تغییرات در خط داده هنگامی که کلاک high است ممکن است به عنوان حالت شروع یا توقف برداشت شود.

(A) Bus not busy هر دو خط داده و کلاک در حالت high قرار داشته باشند.

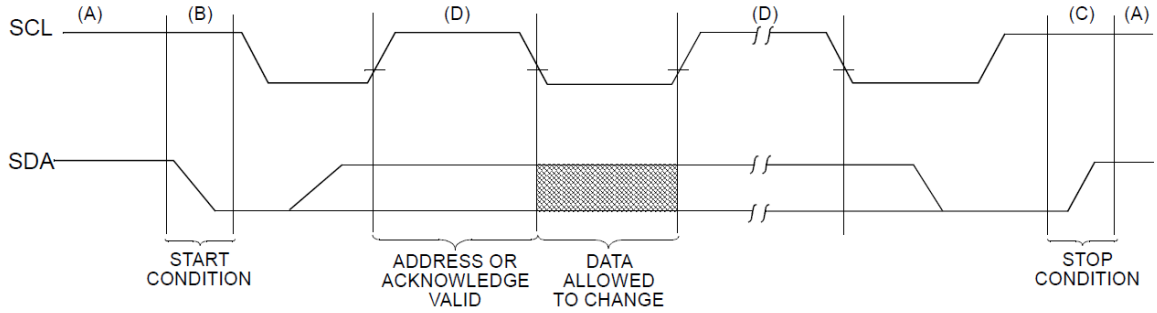
(B) Start condition تبدیل سطح پایه SDA از یک به صفر در زمانی که پایه SCL بالاست یک حالت شروع را تولید می‌کند که به دنبال آن می‌توان هر دستور دیگری را ارسال کرد.

(C) Stop condition تبدیل سطح پایه SDA از صفر به یک در زمانی که پایه SCL بالاست یک حالت توقف را تولید می‌کند. زمانی که این حالت رخ می‌دهد تمام عملیات‌ها باید پایان یابند.

(D) Data valid این حالت زمانی رخ می‌دهد که بعد از حالت start بوده و کلاک high و پایدار باشد. انتقال هر داده با start شروع و با stop به پایان می‌رسد.

Acknowledge تمامی آدرس‌ها و کلمات داده به صورت سریال و در قالب کلمات ۸ بیتی تبادل می‌شوند. EEPROM با دریافت آخرین بیت داده در کلاک نهم یک صفر ارسال می‌کند تا داده صحیح را تصدیق نماید.

در شکل زیر حالت‌ها نمایش داده شده است:

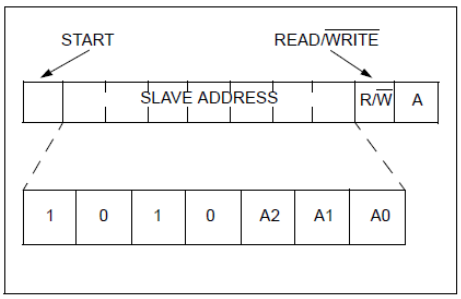


شکل ۵۹: حالت‌های مختلف باس

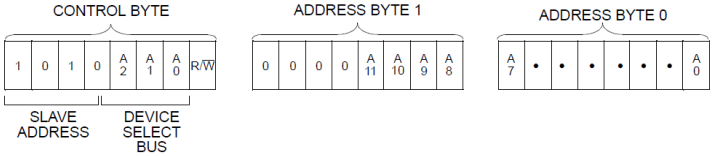
Device addressing

بایت کنترل اولین بایتی است که پس از حالت شروع از طرف Master دریافت می‌شود. این بایت دارای ۴ بیت کنترلی است برای 24LC32 کد باینری 1010 برای عملیات خواندن و نوشتن است و ۳ بیت بعدی بیت‌های device select هستند (A2, A1, A0). این بیت‌ها توسط Master برای دستیابی به ۸ device مختلف مورد استفاده قرار می‌گیرد. مقدار این ۳ بیت باید با سطح پایه های ورودی تراشه تطابق داشته باشد. آخرین بیت کنترلی، عملیاتی که باید انجام شود را نشان می‌دهد. صفر شدن آن به معنای نوشتن و یک شدن آن به معنای خواندن است.

دو بایت بعدی که ارسال می‌شود آدرس اولین بایت داده را مشخص می‌نمایند به علت آنکه تنها A11...A0 استفاده می‌شوند ۴ آدرس بالا باید صفر باشد.



Operation	Control Code	Device Select	R/W
Read	1010	Device Address	1
Write	1010	Device Address	0



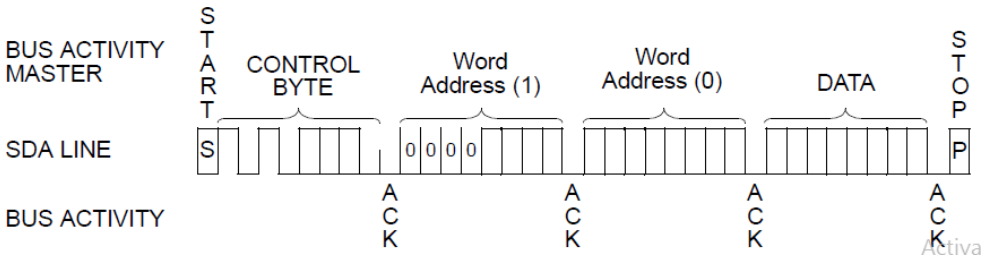
شکل ۶۰: آدرس دهی deviceها

انواع عملیات نوشتن در حافظه

بایت به بایت (byte write)

می‌توان آن را به مراحل زیر تقسیم نمود:

۱. حالت شروع: برای ایجاد حالت شروع میتوان از تابع i2c_start() در محیط کدویژن استفاده نمود.
۲. آدرس تراشه: برای ارسال آدرس تراشه می‌توان از تابع i2c_write(device add) استفاده کرد.
۳. آدرس کلمه: برای ارسال آدرسی از حافظه که داده باید در آنجا نوشته شود می‌توان از تابع i2c_write(mem add) استفاده کرد.
۴. داده: برای ارسال داده هشت بیتی می‌توان از دستور i2c_write(data) استفاده نمود.
۵. حالت توقف: برای ایجاد حالت توقف می‌توان از دستور i2c_stop() استفاده نمود.

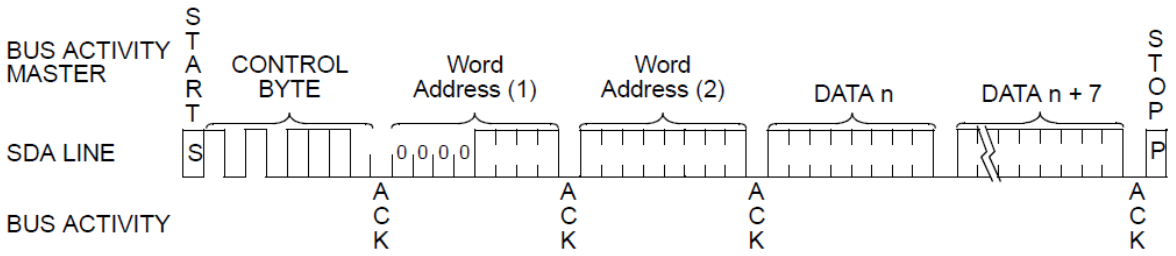


شکل ۶۱: نوشتن بایت به بایت

صفحه به صفحه (PAGE WRITE)

در این روش نحوه نوشتن در هر صفحه از حافظه مطابق شکل زیر انجام می‌گیرد در این حالت نیز نحوه شروع عمل نوشتن مشابه حالت قبل است با این تفاوت که در این حالت ریزپردازنده پس از ارسال اولین بایت داده حالت توقف را ایجاد نمی‌کند. در عوض پس از اینکه EEPROM دریافت اولین کلمه داده را تصدیق نمود ریزپردازنده می‌تواند حداکثر تا ۷ کلمه داده دیگر ارسال کند در این صورت EEPROM با دریافت هریک از آنها یک صفر پاسخ می‌دهد و در پایان هم ریزپردازنده باید با تولید حالت توقف نوشتن روی صفحه را متوقف کند.

در این روش به طور مداوم آدرس حافظه برای اشاره به محل حافظه بعدی افزایش می‌یابد. زمانی که آدرس به انتهای صفحه رسید، در صورتی که ارسال داده‌ها توسط ریزپردازنده متوقف نشود، اشاره‌گر آدرس به ابتدای همان صفحه اشاره خواهد کرد که در این صورت داده ها داخل صفحه دوباره بازنویسی می‌شوند.



شکل ۶۲: نوشتن صفحه به صفحه

انواع عملیات خواندن از حافظه

خواندن آدرس اخیر (current address read)

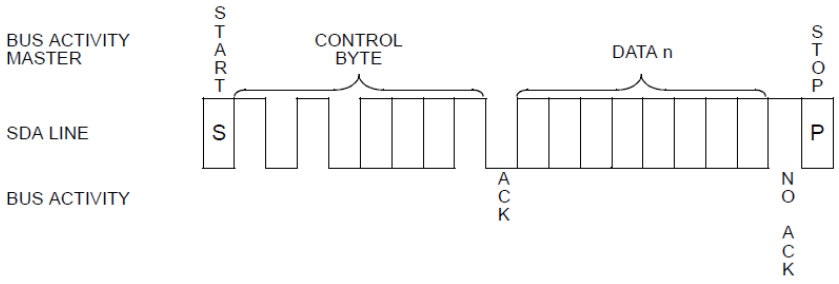
در این نحوه خواندن آدرس اخیر مطابق شکل زیر است و از مراحل زیر تشکیل شده است:

۱. حالت شروع: برای ایجاد حالت شروع می‌توان از تابع i2c_start() در محیط کدویژن استفاده نمود.

۲. آدرس تراشه: برای ارسال آدرس تراشه می‌توان از تابع i2c_write(device add) استفاده کرد.

۳. داده: برای خواندن داده مربوط به آدرس اخیر می‌توان از دستور data = i2cread(0) استفاده کرد.

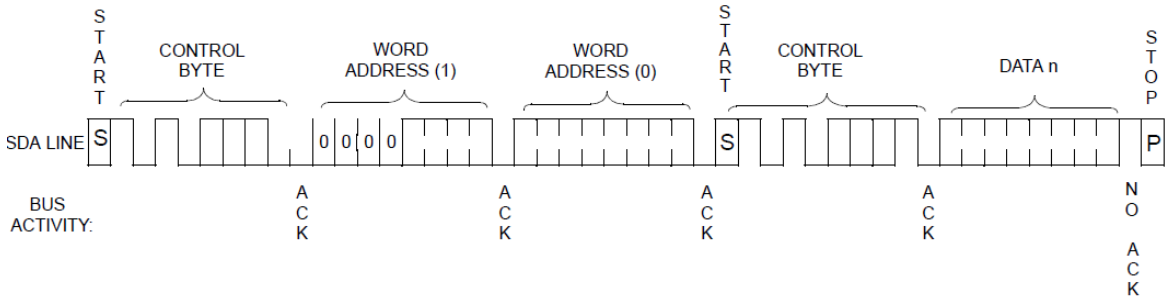
۴. حالت توقف: برای ایجاد حالت توقف می‌توان از دستور i2c_stop() استفاده نمود.



شکل ۶۳: خواندن آدرس اخیر

خواندن آدرس دلخواه (random read)

در این روش می‌توان داده موجود در یک آدرس دلخواه را از حافظه خواند. نحوه عملکرد این روش به صورت شکل زیر می‌باشد و می‌توان آن را به مراحل زیر تقسیم نمود:



شکل ۶۴: خواندن آدرس دلخواه

۱. حالت شروع: برای ایجاد حالت شروع می‌توان از تابع i2c_start() در محیط کدویژن استفاده نمود.
۲. آدرس تراشه: برای ارسال آدرس تراشه می‌توان از تابع i2c_write(device add) استفاده کرد.
۳. آدرس کلمه: برای ارسال آدرسی از حافظه که داده باید در آنجا نوشته شود می‌توان از تابع i2c_write(mem add) استفاده کرد.
۴. آدرس تراشه: برای ارسال آدرس تراشه می‌توان از تابع i2c_write(device add |1) استفاده کرد.
۵. داده: برای خواندن داده مربوط به آدرس اخیر می‌توان از دستور data = i2cread(0) استفاده کرد.
۶. حالت توقف: برای ایجاد حالت توقف می‌توان از دستور i2c_stop() استفاده نمود.

خواندن دنباله‌ای (sequential read)

از این روش می‌توان در هنگام خواندن از آدرس اخیر و یا آدرس دلخواه استفاده نمود. زمانی که ریزپردازنده یک کلمه داده را دریافت می‌کند آن را تصدیق می‌نماید. تا زمانی که EEPROM تصدیق را دریافت کند اشاره‌گر آدرس کلمه را افزایش می‌دهد و به صورت سریال، دنباله‌ای از کلمات داده را برای ریزپردازنده می‌فرستد. زمانی که آدرس به انتهای حافظه رسید از ابتدای حافظه ارسال داده‌ها را آغاز می‌کند. برای توقف ارسال دنباله‌ی داده‌ها ریزپردازنده باید به EEPROM پاسخ ندهد (آن را تصدیق نکند) و در عوض یک حالت توقف ایجاد کند.

پیش گزارش آزمایش هشتم

نام و نام خانوادگی: **شماره دانشجویی:**

---

۳) آزمایش‌های ۸-۱ و ۸-۲ را با استفاده از نرم‌افزار Proteus شبیه‌سازی کنید. (کد نوشته شده به زبان C شماتیک مدار و اطلاعات مربوط به Simulation log را در فایل پیش‌گزارش قرار دهید)

دستور کار

آزمایش ۸-۱

هدف از انجام این آزمایش برقراری ارتباط دو ریزپردازنده از طریق ارتباط SPI می‌باشد. به این گونه که ریزپردازنده اول از طریق Volume Controller یک مقدار ولتاژ دریافت کند و مقدار اندازه‌گیری شده را به ریزپردازنده دیگر ارسال کند سپس ریزپردازنده دوم مقدار دریافتی را روی LCD گرافیکی نمایش دهد.

آزمایش ۸-۲

هدف از انجام این آزمایش برقراری ارتباط ریزپردازنده و حافظه EEPROM از طریق پروتکل TWI می‌باشد. در ابتدا یک تصویر ۳۲*۳۲ تولید کرده و آن را در EEPROM ذخیره نمایید. سپس یک برنامه بنویسید که تصویر را از حافظه بخواند و روی LCD گرافیکی نشان دهد.

راهنمایی: برای تولید دیتای تصویر از نرم‌افزار LCD Vision استفاده نمایید.