فهرست مطالب

بخش اول - آموزش AVR

بخش دوم - پروژه های تکمیلی

این جزوه در دو بخش تهیه شده است:

بخش اول شامل آموزش میکرکنترلر AVR با تاکید بر ATmega16 و ATmega32 همراه با مثال ها و پروژه های مرتبط با موضوع درس می باشد. اما، بخش دوم فقط شامل پروژه های عملی می باشد که جنبه تکمیلی داشته و با هدف افزایش توانایی کار با میکروهای AVR نوشته شده است.

این جزوه خلاصه ای از کتاب " مرجع کامل آموزش AVR" نوشته "مهندس پرتوی فر" انتشارات "نص" می باشد و تمامی پروژه ها و مثال های متن جزوه با "بسته های آموزشی AVR " شرکت " آریامِک" قابل اجرا و تست می باشند.

فیوز بیت ها، منابع کلاک و Reset

فیوز بیت ها

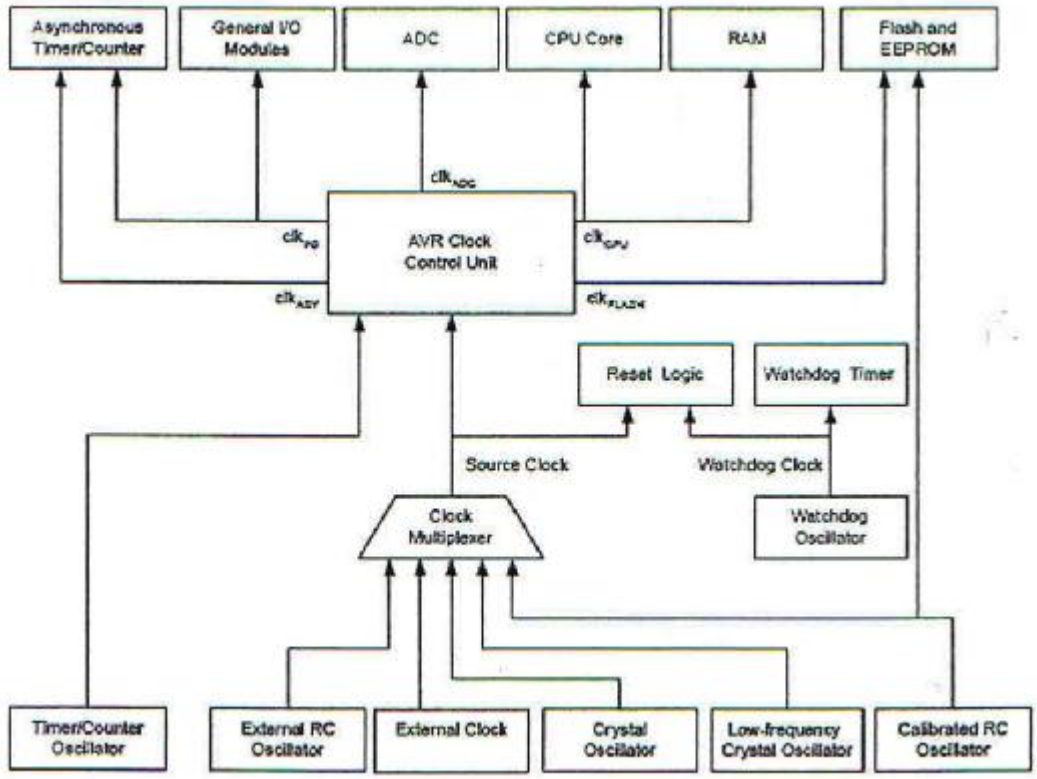
فیوز بیت ها قسمتی از حافظه Flash هستند که امکاناتی را در اختیار کاربر قرار می دهند و با Erase شـدن میکـرو از بین نمی روند. یک به معنی غیر فعال بودن و صفر فعال بودن هر بیت می باشد. قطعه ی Mega16 دارای ۲ بایت فیوز

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

بیت می باشد که یکی از آنها طبق جدول فوق می باشد.

منابع کلاک

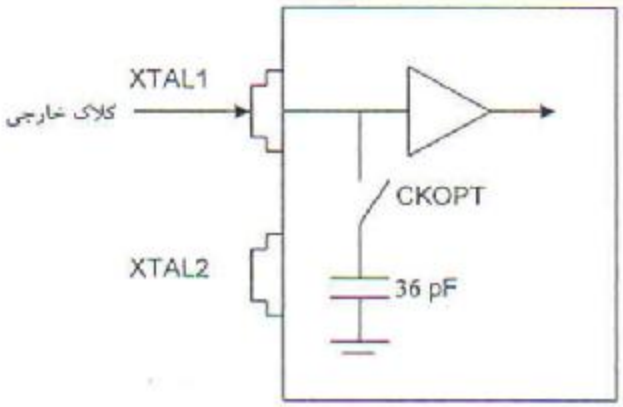
منابع کلاک در میکروهای AVR شامل: اسیلاتور RC کـالیبره شـده، اسـیلاتور کریسـتالی فرکـانس پـایین، اسـیلاتور کریستالی، کلاک خارجی، اسیلاتور RC خارجی و اسیلاتور تایمر/ کانتر می باشند.



انتخاب منبع کلاک بوسیله ی فیوزبیت های CKSEL بوده و مطابق جدول زیر می باشد. مقدار پیش فرض بیت هـای CKSEL، یک بوده و در نتیجه منبع پیش فرض، اسیلاتور RC داخلی می باشد.

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

کلاک خارجی: برای راه اندازی بوسیله منبع کلاک خارجی باید یک پالس به پین XTAL1 اعمال شود. برای قرار گرفتن در این وضعیت باید تمام بیت های CKSEL پروگرام شده (صفر شوند) و کاربر میتواند با پروگرام کردن فیوز بیت CKOPT یک خازن داخلی به ظرفیت ۳۶ پیکوفاراد را بین ورودی و زمین قرار دهد.

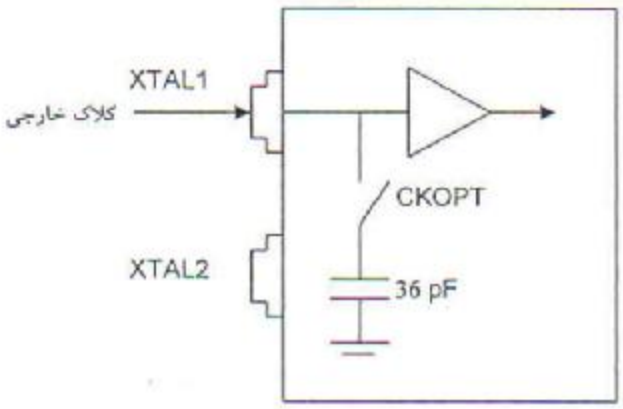


اسپیلاتور RC کالیبره شده ی داخلی: این منبع در فرکانس های ۱، ۲، ۴ و ۸ مگاهرتز موجود می باشد و مقدار آن در دمای ۲۵ درجه و ولتاژ ۵ ولت کالیبره شده است که در این وضعیت ممکن است تا ۳ درصد خطا در کلاک ایجاد شده وجود داشته باشد. فرکانس نوسان بوسیله ی فیوز بیت های CKSEL تعیین شده و مطابق جدول زیر می باشد. در این وضعیت CKOPT نباید پروگرام شود.

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

اسیلاتور RC خارجی: در کاربردهایی که دقت کلاک اهمیت زیادی ندارد می توان از این منبع استفاده کرد. پیکربندی مطابق شکل زیر بوده و فرکانس نوسان از رابطه ی $f = \frac{1}{3RC}$ بدست می آید. حداقل مقدار C برابر ۲۲

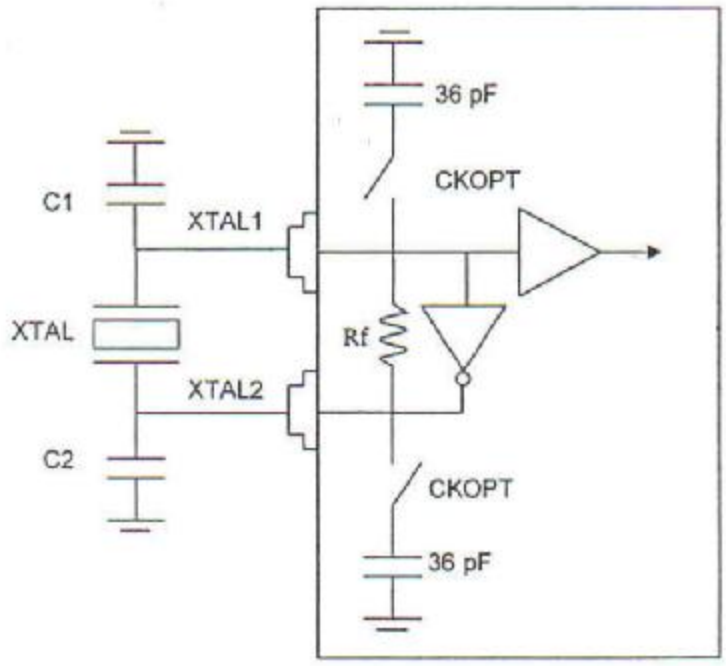
پیکو فاراد بوده و در صورتی CKOPT پروگرام شود می توان مقدار ۳۶ پیکوفاراد را نیز لحاظ نمود.



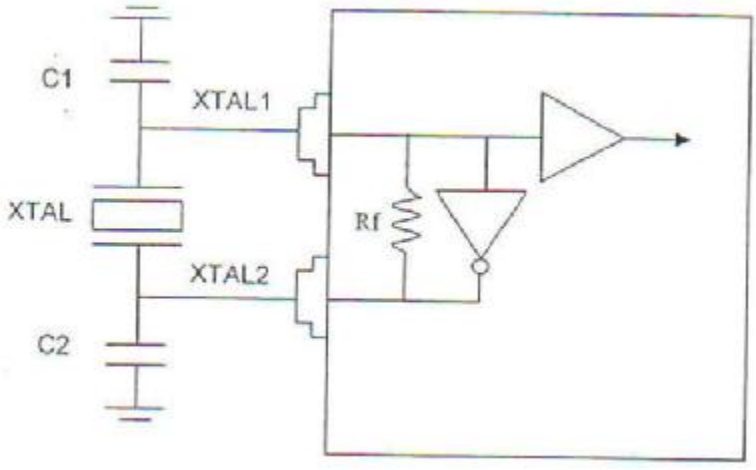
این منبع نوسان می تواند در چهار Mode کاری عمل کند که هر کدام برای یک بازه ی فرکانسی بهینه شده است و بوسیله ی فیوز بیت های CKSEL مطابق جدول زیر انتخاب می شود.

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

اسیلاتور کریستالی فرکانس پایین: این منبع کلاک می تواند کریستال های فرکانس پایین مثل کریستال ساعت با فرکانس ۳۲۷۶۸ هرتز باشد. با دادن مقدار ۱۰۰۱ به فیوز بیت های CKSEL منبع کلاک کریستال خارجی فرکانس پایین انتخاب شده و در این وضعیت پیکربندی مطابق شکل زیر می باشد. در صورت پروگرام نمودن CKOPT می توان از خازن خارجی صرفنظر نمود.



کریستال کوارتز یا رزوناتور سرامیکی: پین های XTAL1 و XTAL2 به ترتیب ورودی و خروجی یک تقویت کننده ی وارونگر هستند که می توانند به عنوان یک اسیلاتور On-chip مطابق شکل زیر پیکربندی شوند.



* به جای کریستال کوارتز می توان از رزوناتور سرامیکی استفاده نمود که از دوام بیشتری در مقابل ضربه برخوردار است و زمان Startup کمتری نیز دارد و البته نسبت به کریستال کوارتز دقت کمتری داشته و پایداری دمایی آن نیز کمتر است.

* در این وضعیت خازن های ۳۶ پیکو فاراد حذف شده و عملکرد فیوز بیت CKOPT نیز تغییر می کند. بدین ترتیب که با پروگرام شدن این بیت دامنه ی خروجی تقویت کننده ی وارونگر افزایش یافته و می توان از پین XTAL2 به عنوان کلاک برای یک وسیله ی دیگر استفاده نمود. همچین با فعال کردن CKOPT در محیط های نویزی عملکرد اسیلاتور بهبود می یابد.

* چنانچه از رزوناتور استفاده می شود برای فرکانس های بالاتر از ۸ مگا هرتز باید CKOPT پروگرام شود. اسیلاتور می تواند در سه وضعیت متفاوت نوسان کند که هر کدام برای یک محدوده ی فرکانسی بهینه شده است و آن را می توان با فیوز بیت های CKSEL مطابق جدول زیر انتخاب نمود.

CKOPT	CKSEL3..1	Frequency Range (MHz)	Recommended Range for capacitors C1 and C2 for Use with crystals (PF)
1	101(1)	0.4-0.9	-
1	110	0.9-3.0	12-22
1	111	3.0-8.0	12-22
0	101.110.111	1.0≤	12-22

* با هر یک از منابع کلاک انتخاب شده بوسیله ی فیوز بیت های CKSEL، دو بیت به نام های SUT[1:0] نیز وجود دارد که از طریق آن می توان حداکثر زمان Start-up منبع کلاک را به میکرو اعلام نمود. مقدار این بیت ها به طور پیش فرض ماکزیمم زمان Start-up را در نظر می گیرد و در صورتی که نیاز است مقدار آن را تغییر دهید مطابق جداول مربوطه در قسمت System clock and clock options در Datasheet عمل کنید.

Reset *

با Reset شدن میکرو کنترلر، تمام رجیسترهای I/O به مقدار اولیه شان تغییر می دهد و CPU شروع به اجرای دستورالعمل ها از ابتدای برنامه خواهد کرد. مدار زیر برای ریست میکرو استفاده می شود.

اصطلاح Power On Reset

اصطلاح فوق به این معناست که میکرو هنگام وصل شدن تغذیه اش ریست می شود. بنابراین وصل شدن تغذیه میکرو معادل ریست شدن آن و اجرای برنامه از ابتدا می باشد.

آشنایی با زبان C

* هر برنامه ی C حداقل یک تابع () main دارد. (این اولین تابع اجرایی است).

* یک برنامه ی میکروکنترلری در ساده ترین حالت خود به شکل زیر است:

تعاریف کلی

الگوی توابع

```
Void main ( )
{
        White (1)       {
                .
                .
                .
                        }
}
```

توابع تعریف شده

* برنامه ی ساده ی زیر رشته ی Hello world را به خروجی استاندارد ارسال می کند.

```
# include <stdio.h>
Void main ( )
{       Printf (“Hello world!”);
        While (1)
}
```

* while (1) ایجاد یک حلقه ی نامتناهی می کند.

* در انتهای هر عبارت یک سمی کالن می آید.

* Brace (آکولاد) ابتدا و انتهای یک تابع و همچنین یک بلوک را مشخص می شود.

* از " " برای مشخص کردن ابتدا و انتهای یک رشته ی متنی استفده می شود.

* از // یا /* ... */ برای نوشتن توضیحات استفاده می شود.

* شناسه ها اسامی متغیرها، ثوابت و یا توابع هستند.

شناسه ها نمی تواند از کلمات رزرو شده باشند و همچنین نمی توانند با یک کاراکتر عددی شروع شود و طول آن ها باید کمتر از ۳۱ کاراکتر باشد.

* C یک زبان Case sensitive است و بین حروف کوچک و بزرگ تفاوت قائل می شود.

* کلمات رزرو شده (دستورات) حتماً باید با حرف کوچک استفاده شوند. (مثل if, char, while, …)

متغیرها و ثوابت

* تعریف متغیر یعنی انتخاب نام مستعار برای مکانی از حافظه بصورت زیر:

* فرم تعریف متغیر: نوع متغیر نام متغیر ; مثال: char a;

انواع داده ها:

Type	Size (Bits)	Range
Bit	1	0,1
Char	8	- 128 to 127
Unsigned char	۸	0 to 255
signed char	8	-128 to127

int	16	-32768 to 32767
short int	16	-32768 to 32767
unsigned int	16	0to65535
signed int	16	-32768 to 32767
long int	32	-21147483648 to 2147483647
unsigned long int	32	o to 4294967295
signed long int	32	-2147483648 to 2147483647
float	32	± 1.17 5e-38 to ±3.402e38
double	32	±1.175e-38 to ±3.402e38

* برای تعیین محل تعریف متغیر از عملگر @ استفاده می کنیم. مثال: int b@0xA3;

* برای تعریف متغیر در حافظه ی EEPROM از کلمه ی کلیدی eeprom قبل از نام متغیر استفاده می کنیم. مثال:

eeprom int code;

* می توان در زمان تعریف به متغیر مقدار اولیه داد. مثال: char s= ‘a’;

* تعریف ثابت با کلمه یا کلیدی const ی flash انجام می شود.

مثال: const float pi=3.14;

* با کلمه ی #define می توان ماکرو تعریف کرد، در این حالت مقداری که برای ثابت تعریف می شود نوع داده را تعیین می کند. مثال: #define c 100;

عملگرهای حسابی و بیتی

عملگر	عملکرد	مثال	نتیجه
*	ضرب	۳*۲	۶
/	تقسیم	۵/۲	۲.۵

+	جمع	۳+۶	۹
-	تفریق	۳-۸	۵
٠/٠	باقیمانده	۳% ۱۰	۱
&	AND	0×0F & 0 × F0	0×00
|	OR	0×00 | 0×03	0 × 03
^	XOR	0×0F ^ 0×FF	0×F0
~	مکمل یک	~(0×F0)	0×0F
>>	شیفت به راست	0×F0>>4	0×0F
<<	شیفت به چپ	0×0F<<4	0×F0

عملگرهای یکانی

عملگر	عملکرد	مثال	نتیجه
-	قرینه	-a	a=a×-1
++	افزایش یک واحد	a++	a=a+1
--	کاهش یک واحد	a--	a=a-1

عملگرهای مقایسه ای

عملگر	عملکرد	مثال	نتیجه
>	بزرگتر	2>3	False
<	کوچکتر	‘m’>’e’	True
>=	بزرگتر یا مساوی	5>=5	True

<=	کوچکتر یا مساوی	2.5<=4	True
= =	تساوی	‘A’ = = ‘B’	False
!=	نامساوی	2!=3	True

عملگرهای منطقی

عملگر	عملکرد	مثال	نتیجه
&&	AND منطقی	(2>3)&&(1!=3)	False
||	OR منطقی	(‘a’<3)||(10)	True
!	نقیض	!(7>5)	False

عملگرهای انتساب

عملگر	عملکرد	مثال	نتیجه
=	انتساب	a=b	a⇐b
*=	ضرب و انتساب	a*=b	a=a*b
/=	تقسیم و انتساب	a/=b	a=a/b
+=	جمع و انتساب	a+=b	a=a+b
- =	تفریق و انتساب	a-=b	A=a-b
a=(value)?x:y	انتساب شرطی	اگر value مقدار صحیح باشد a برابر x و در غیر اینصورت برابر y می شود.	

آرایه ها

آرایه ها مجموعه ای از متغیرهای همنوع هستند.

* فرم تعریف: ;[تعداد عناصر] اسم متغیر نوع متغیرهای آرایه.

* مثال: int a [5];

با مقدار اولیه: int a [5]={1,2,3,4,5}

* فرم تعریف آرایه های دو بعدی: ; [تعداد عناصر ستون] [تعداد عناصر سطر] اسم متغیر نوع متغیر

مثال: int a [2] [3] = {{1,2,3},{4,5,6}}

در زبان C اندیس آرایه ها از صفر شروع می شود.

رشته ها

رشته ها آرایه ای کاراکتر هستند.

* مثال char name [ ]= “Reza”;

یا: char name [5] = {‘R’,’e’,’z’,’a’,’10’};

* رشته ها همواره به یک کاراکتر Null ختم می شوند.

تصمیم گیری و انتخاب

۱- دستور goto: پرش بدون شرط به یک برچسب انجام می شود.

۲- ساختار if – else:

```
if (شرط)
{
دستورات ۱ ;
}
Else
{
دستورات ۲ ;
}
```

* در صورتی که دستورات یک خطی باشند می توان brace را حذف نمود.

* استفاده از بلاک else اختیاری است.

مثال:

```
#include<stdio.h>
void main( ) {
        int a;
        printf("Enter a Number: ");
        scanf("%d",&a);
        if(a= =0)
                printf("You've entered zero\n");

        else if(a>0)
                printf("You've entered a positive number\n");
        else
                        printf("You've entered a negative number\n");
```

۳- ساختار Switch – Case

```
switch (مقدار)
{
case ۱ مقدار :
۱ دستورات ;
Break;
case ۲ مقدار :
```

دستورات ۲ ;

break ;

.

.

Default:

دستورات n ;

}

مثال:

```
#include<stdio.h>
void main( ) {
        int a;
        printf("Enter Month of your birth: ");
        scanf("%d",&a);
        switch (a) {
        case· 1:
        case 2:
        case 3: printf("You've born is spring\n");
        break;
        case 4:
        case 5:
        Case 6: printf("You've born is sununer\n");
        break;
        case 7:
```

```
case 8:
case 9: printf("You've born is autumn\n");
break;
case 10:
case 11:
case 12: printf("You've born is winter\n");
break;
default: printf("Error! Enter a number between 1-12\n");
```

حلقه های تکرار

۱- ساختار while:

```
while (شرط)
{
دستورات ;
}
```

```
#inclide< stdio.h>
Void main ( ) {
    Char a ;
    Printf (“Enter E to exit\ n”);
    While (a != ‘E’) a=getchar ( ) ;
}
```

۲- ساختار Do/ while:

```
Do{
دستورات ;
```

} while (شرط)

در ساختار Do/ While بر خلاف While شرط در انتهای حلقه آزمایش می شود، بنابراین دستورات داخل حلقه، حداقل یکبار اجرا می شوند.

۳- حلقه های For:

for (مقدار ابتدای حلقه ; شرط پایان ; گام)

{

دستورات ;

}

مثال:

```
#include<stdio.h>
void main( ){
        int a,i;
        long int fact=1;
        printf("Enter a Number: ");
        scanf("%d",&a);
        if(a<0)
        printf ("Error! You must Enter a positive number\n");
        else if(a==O)
        printf("Factorial of 1 is l\n");
        else{
        for(i=l;i<=a;i++)
        fact*=i;
        printf("Factorial of %d is %d\n",a,fact);
```

```
}
}
```

* دستور break باعث خروج بدون شرط از هر حلقه ای می شود.

* دستور continue باعث می شود اجرای ادامه دستورات متوقف شده و حلقه از ابتدا آغاز شود.

توابع

تعریف توابع به صورت زیر می باشد:

```
(آرگومانهای تابع و نوع آنها) نام تابع    نوع داده خروجی
{
   متغیرهای محلی
      دستورات تابع
}
```

⇐ توابع داخل یکدیگر قابل تعریف نمی باشند و جدا از هم باید تعریف گردند.

مثال:

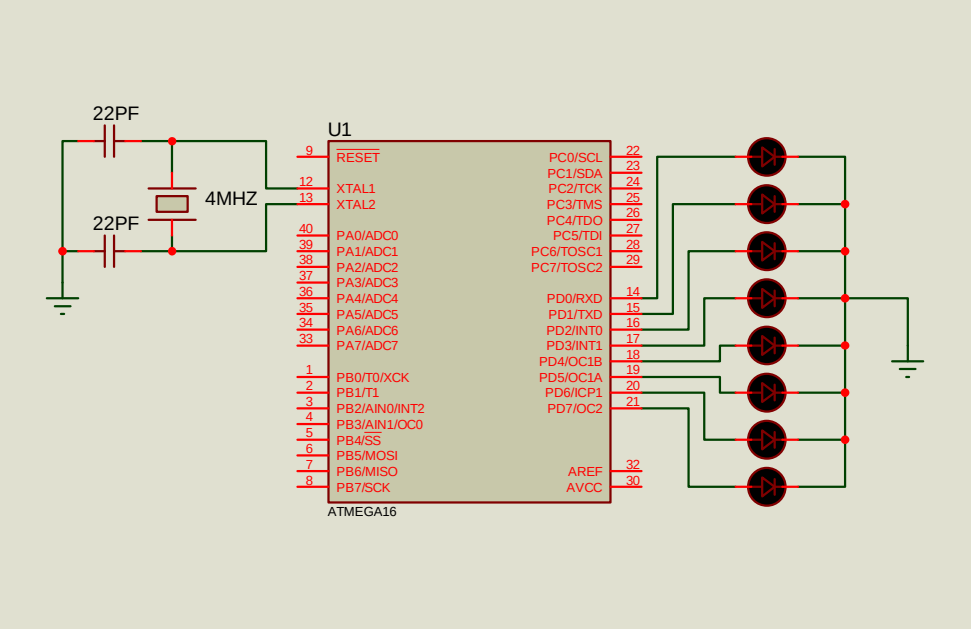
```
#include<stdio.h>
Long int cube (int x);
Void main ( ) {
        int a ;
        print f (“Enter a number:”);
        scan f (“%d” , & a);
        print f (“cube of %d is %d\n” , a, cube (a));
}
Long int cube (int x){
```

```
        return x*x*x;
}
```

مثال:

```
linclude<stdio.h>
int _max(int a,int b);
void main( ) {
        int a,b;
        printf("Enter Two Numbers:);
        scanf ("%d%d", &a, &b) ;
        printf ("Maximum of % d and % d is %d \ n" , a, b, _ max (a,b)):
}
int _ max (int a , int b) {
        If (a>b)
                return a:
        else
                return b;
}
```

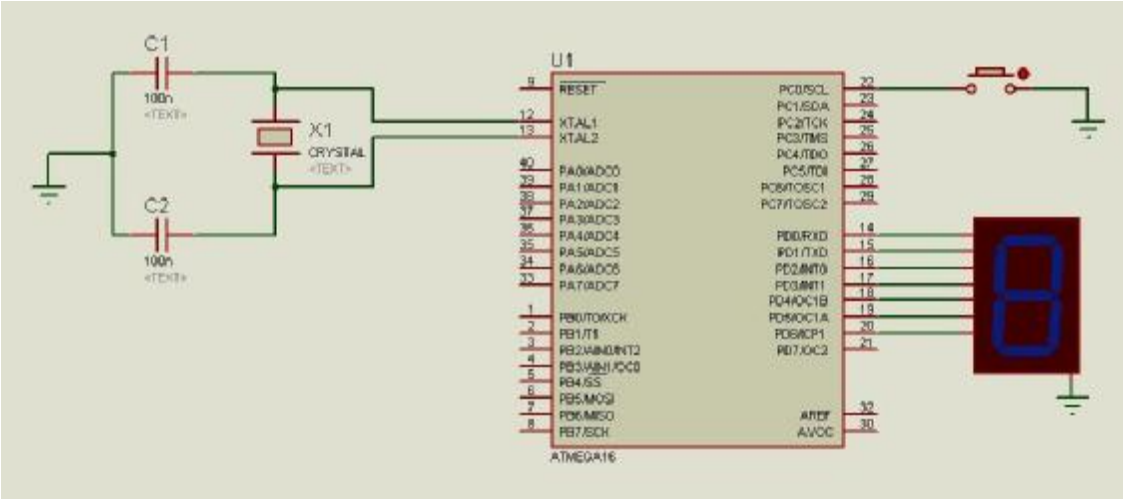
پروژه ۱: فلاشر ساده




```
//Project: LED Flasher
# include < mega 16.h>
# include< delay.h>
int i;
int main (void)
while (1)
        {
                For (i=1; I <=128; i=i*2)
                {
                PORTD= i;
                Delay_ms (100);
                }
For (i=128; i > 1; i=i/2)
{
```

```
            PORTD= i;
            Delay_ms (100);
        }
        }
}
```

پروژه ۲: کانتر یک رقمی با کلید

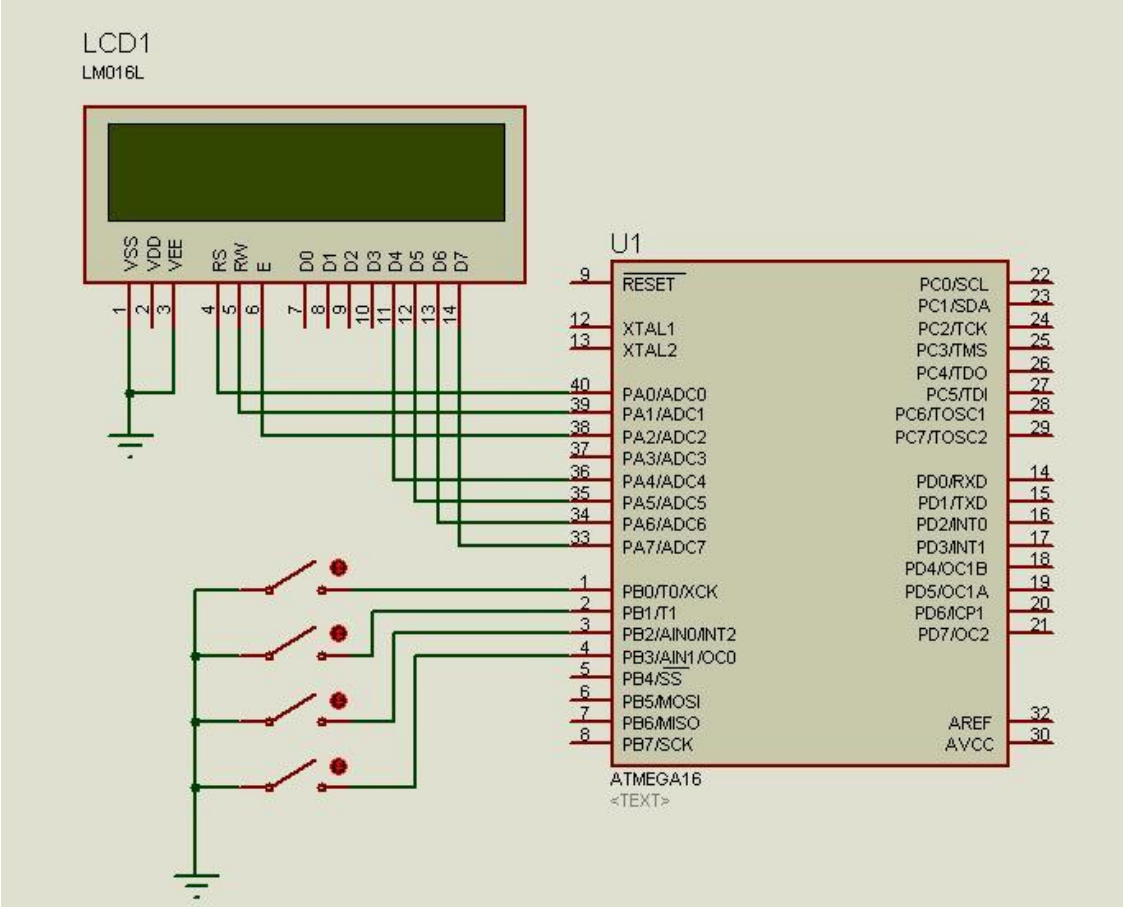



```
//Project: Key Counter
#include <mega16.h>
#define xtal 4000000
char digits[16]={0x3F,0x06,0X5B,0X4F,0x66,0x6D,
0x7D,0X07,0x7F,0x6F,0x77,0x7C,0x39,0X5E,0x79,0x71}; unsigned char;
unsigned char p_state;
unsigned char keY;
unsigned char i;
void main(void)
{
```

```
DDRD = 0xFF;
PORTD = digits[0];
DDRC = 0x00;
PORTC = 0XFF;
while(l)
{
key = PINC & 0b0000000l;
if(key==0)
{
if(key!=p_state)
{
if(i==l5)
{
                    i=0;
          PORTC=digits[0]
}
else
i++;
PORT D= digits = digits [i] ;
          p_state= Key;
          };
}
else
```

```
        P_state = 0xFF;
        }
    }
```

پروژه ۳: اتصال LCD کاراکتری به میکرو

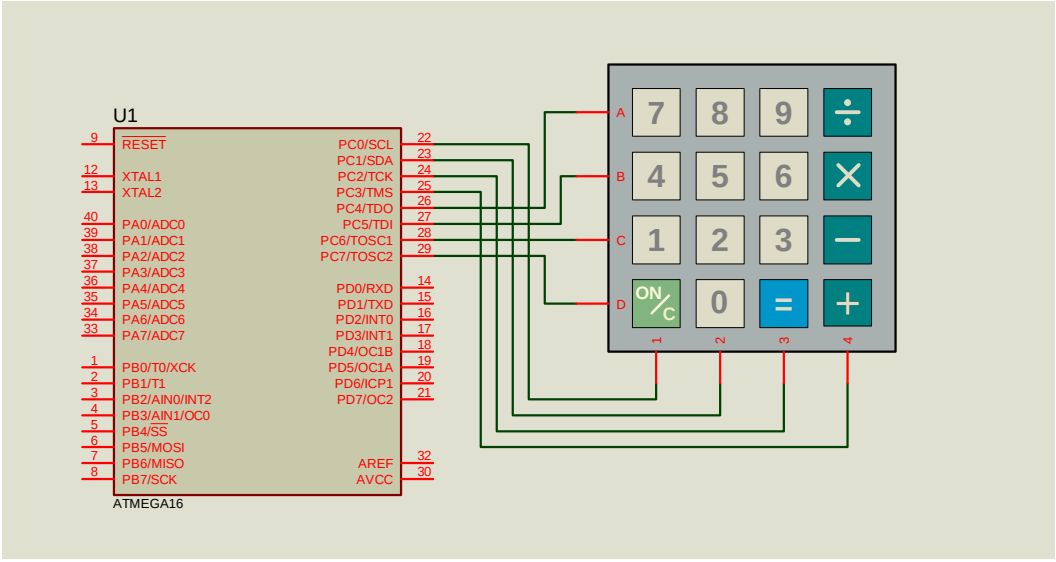



```
//Project : LCD Interfacing

#include <stdio.h>

#include <mega16.h>

#include <delay.h>

#include <lcd.h>

# de fine xtal 4000000
```

```
#asm
.egu __lcd__ port=0xlB ;PORTA
#endasm
void main(void)
{
        char buffer [20];
        unsigned char W;
        PORTB=0xFF;
        DDRB=0x00;
        lcd_init (16);
        lcd_clear () ;
        while (1)
        {
        w = ~PINB;
        if (w! =0x00)
        {
        lcd_clear();
                lcd_gotoxy(0,0);
                sprintf(buffer,"Number=%d",w);
                lcd_puts(buffer);
                delay_ms (100);
        }
        else
```

```
{
                lcd_clear( );
                lCd_putsf ("Nurnber=0")
                 delay_ms(l00);
                }
        }
}
```

پروژه ۴: اسکن صفحه کلید ماتریسی

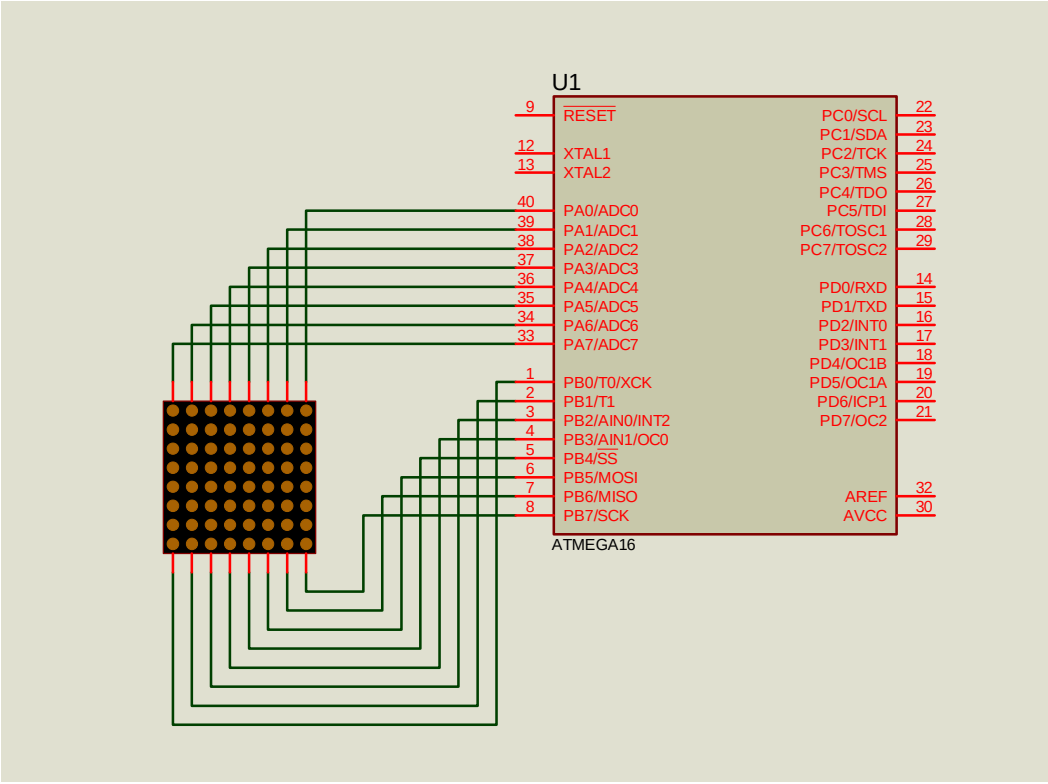



```
//Project: Keypad Scan

#inciude <mega16.h>

#include <delay.h>

#define maxkeys 16

#define xtal 4000000
```

```
unsigned char key, butnum;
flash unsigned char keytbl[16]={0xee, 0xed, 0xeb, 0xe7, 0xde, 0xdd, 0xdb, 0xd7, 0xbe,
0xbd, 0xbb, 0xb7, 0x7e, 0x7d, 0x7b, 0x77};
void main{void)
{
DDRB = 0xff;
PORTB = 0xff;
while (1)
{
DDRC = 0x0f;
PORTC = 0xf0;
delay_us (5) ;
key = PINC;
DDRC = 0xf0;
PORTC = 0x0f;
delay_us(5);
key = key I PINC;
if (key! = 0xff)
{
for (butnum=0; butnum<maxkeys; butnum++)
{
if (keytbl [butnum] ==key) break;
```

```
}
if (butnum==maxkeys) butnum=0;
else butnum++;
}
else butnum=0;
PORTB = ~butnum ;
        }
}
```

پروژه ۵: نمایشگرهای LED Dot matrix




```
//Project: Dot Matrix Display
#include <mega32.h>
```

```
#include <delay.h>
#define xtal 4000000
unsigned char k;
unsigned char arr[8]={0x18, 0x3C, 0x66, 0X66, 0x7E, 0x66, 0x66, 0x00};
//uunsigned char arr[8]={0x7E, 0x33, 0x33, 0x3E, 0x33, 0x33, 0x7E, 0x00};
//uunsigned char arr[8]={0xlE, 0x33,0x60, 0x60, 0x60, 0x33, 0xlE, 0x00};
void main(void)
{
DDRA=0xFF;
 DDRB=0xFF;
while (l){
        for(k=0;k<=7;k++) {
        PORTA=arr[k];
        PORTB=~ (l«k);
        delay_us(l00);
        PORTB=0xFF;
        }
        }
}
```

وقفه های خارجی

* سیستم اولویت وقفه ها در AVR به صورت یک سطحی می باشد.

* قطعه ی ATmega16 دارای ۲۱ منبع وقفه می باشد که ۳ مورد از آن ها وقفه ی خارجی می باشند.

شماره	نام در Code Vision	آدرس	منبع وقفه	توضیح
۱		$000	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
۲	EXT_INTO	$002	INTO	External Interrupt Request 0
۳	EXT-,INT1	$004	INT1	External Interrupt Request 1
۴	TIM2_COMP	$006	TIMER2 COMP	Timer/Counter2 Compare Match
۵	TIM2_OVF	$008	TIMER2OVF	Timer/Counter2 Overflow
۶	TIM1_CAPT	$00A	TIMER1 CAPT	Timer/Counter1 Capture Event
۷	TIM1_COMPA	$00C	TIMER1 COMPA	Timer/Counter1 Compare MatchA
۸	TIM1_COMPB	$OOE	TIMER1 COMPB	Timer/Counter1 Compare Match B
۹	TIM1_OVF	$010	TIMER1OVF	Timer/Counter1 Overflow
۱۰	TIMO_OVF	$012	TIMER0 OVF	Timer/Counter0 Overflow
۱۱	SPI_STC	$014	SPI, STC	Serial Transfer Complete
۱۲	USART_RXC	$016	USART,RXC	USART, Rx Complete
۱۳	USART;..DRE .	$018	USART, UDRE	USART Data Register Empty
۱۴	USART_TXC	$01A	USART, TXC	USART, Tx Complete
۱۵	ADC_INT	$01C	ADC	ADC Conversion Complete
۱۶	EE_RDY	$01E	EE_RDY	EEPROM Ready
۱۷	ANA_COMP	$020	ANA_COMP	Analog Comparator
۱۸	TWI	$022	TWI	Two-wire Serial Interface
۱۹	EXT_INT2	$024	INT2	External Interrupt Request 2
۲۰	TIMO_COMP	$026	TIMER0 COMP	TImer/Counter0 Compare Matm
۲۱	SPM_READY	$028	SPM_RDY	Store Program Memory Ready

* آدرس های پایین تر دارای اولویت بالاتری می باشند و در صورت درخواست همزمان دو یا چند وقفه ابتـدا بـه اولویـت بالاتر پاسخ داده می شود و پس از آن به بقیه ی وقفه ها بر حسب اولویت رسیدگی می شود.

برای فعال کردن هر یک از وقفه ها باید ابتدا باید بیت فعال ساز عمومی وقفه ها را با دستور اسمبلی SEI یا مقدار دهی

رجیستر SREG فعال نمود:

Bit	7	6	5	4	3	2	1	0
SREG	I							

* با رویداد هر وقفه ی خارجی این بیت پاک شده و در نتیجه تمام وقفه های دیگر غیر فعال می شوند. در این حالت نرم افزار می تواند با نوشتن یک روی این بیت آن را مجدداً فعال کند و باعث ایجاد وقفه های تو در تو شود. با بازگشت از ISR این بیت مجدداً یک می شود.

Bit	7	6	5	4	3	2	1	0
GICR	INT1	INT0	INT2					

* وقفه های خارجی ۰، ۱ و ۲ به ترتیب از پین های PD2 ، PD3 و PB2 تریگر می شوند.

نوع تریگر شدن هر یک از وقفه های خارجی ۰ و ۱ بوسیله ی چهار بیت اول رجیستر MCUCR تعیین می شود:

Bit	7	6	5	4	3	2	1	0
MCUCR					ISC1	ISC1	ICS0	ISC0

حالت های مختلف بیت های MCUCR [3:0]

ISC11	ISC10	نوع تریگر شدن وقفه ی صفر
0	0	سطح منطقی صفر در پین INT0
0	1	تغییر در سطح منطقی پین INT0
1	0	لبه ی پایین رونده در پین INT0
1	1	لبه ی بالا رونده در پین INT0

ISC01	ISC00	نوع تریگر شدن وقفه ی یک
0	0	سطح منطقی صفر در پین INT1
0	1	تغییر در سطح منطقی پین INT1
1	0	لبه ی پایین رونده در پین INT1
1	1	لبه ی بالا رونده در پین INT1

نوع تریگر شدن وقفه ی ۲ خارجی بوسیله بیت ۶ از رجیستر MCUCSR تعیین می شود:

Bit	7	6	5	4	3	2	1	0
MCUCR		ISC2						

* وقفه ی ۲ خارجی بر خلاف وقفه ی ۰ و ۱ تنها در دو حالت لبه ی بالا رونده و پایین رونده قابل پیکربندی است. نوشتن صفر در ISC2 باعث ترینگر شدن این وقفه با لبه ی پایین رونده و نوشتن یک باعث تریگر شدن آن با لبه ی پایین رونده می شود.

هر یک از وقفه های خارجی دارای یک بیت پرچم هستند که در صورت تریگر شدن از پین وقفه ی خارجی و فعال بودن بیت مربوطه در رجیستر GICR و فعال بودن بیت فعال ساز وقفه (I)، علاوه بر یک شدن پرچم، می تواند باعث ایجاد وقفه شود. در این حالت پس از اجرای ISR پرچم آن وقفه به صورت سخت افزاری پاک می شود.

Bit	7	6	5	4	3	2	1	0
GIFR	INTF1	INTF0	INTF2					

روتین سرویس وقفه ها در Code Vision به صورت زیر تعریف می شود:

interrupt [شماره ی بردار وقفه] void نام روتین سرویس وقفه (void)

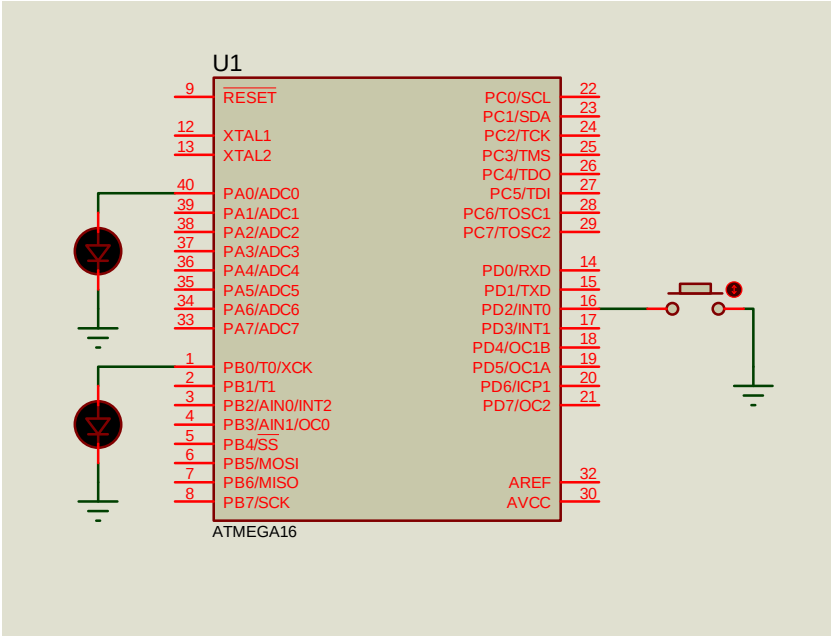
{

برنامه ی سرویس وقفه

}

شماره ی بردار وقفه ی در مورد ATMEGA16 عددی بین ۲ تا ۲۱ می باشد و مـی تـوان از نـام معـادل آن (جـدول ابتدای فصل) نیز استفاده کرد.

مثال۱)




```
#include <mega16.h>

#include <delay.h>

interrupt [2] void LED_ON (void)

{

        PORTA=0x0l;

        delaY_Ms (l000);

        PORTA=0x00;
```

```
}
void main(void)
{
        DDRB=0xFF;
        PORTB=0x00;
        DDRA=0xFF;
        PORTA=0X00;
        DDRD=0x00;
        PORTD=0xFF;
GICR=0b01000000;   // INT0: On
MCUCR= 0b00000010;// INT0 Mode: Falling Edge
#asm("sei") // Global enable interrupts
while (1)
{

        PORTB=0x0l;
        delay ms (500);
        PORTB=0x00;
        delay_ms(500);
        };
}
```

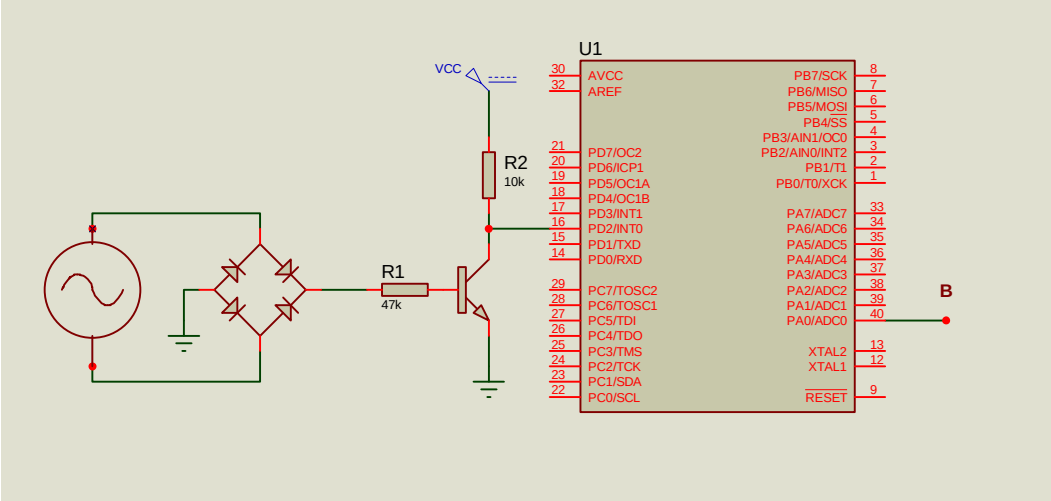
* اگر وقفه ی خارجی به صورت حساس به لبه تنظیم شده باشد و در حال اجرای یکی از وقفه ها، وقفه ی دیگری اتفـاق بیفتد، پس از خروج از ISR وقفه ی جاری به آن وقفه ها بر حسب اولویت شان پاسخ داده خواهد شد.

* در صورتی که پین وقفه به صورت خروجی تعریف شود، آنچه روی پورت نوشته می شود می توانـد باعـث ایجـاد وقفـه شود، به این ترتیب می توان وقفه ی نرم افزاری ایجاد کرد.

* وقفه های حساس به سطح در پین INT0 و INT1 و همچنین وقفه ی حساس به لبه در ITN2 به صورت آسنکرون تشخیص داده می شوند، بنابراین می توان از آن ها برای خارج کردن میکرو از همه ی Mode های کم مصرف اسـتفاده نمود. حداقل عرض پالس در این حالت ۵۰ نانو ثانیه می باشد و برای مقادیر کمتر تشـخیص وقفـه ی خـارجی تضـمین نشده است.

* کامپایلر Code Vision به صورت پیش فرض SREG و رجیسترهایی CPU را که ممکن است در ISR تغییر پیدا کنند، قبل از اجرای روتین سرویس وقفه در Stacks ذخیره (PUSH) کرده و پس از بازگشت از ISR آن ها را بازیابی (POP) می کند.

پروژه ۶: آشکار ساز عبور از صفر




```
//Project: Zero Cross Detector

#include <mega16.h>

#include <delay.h>

#define xtal 4000000
```

```
interrupt [2] void switch_(void)
{
        PORTA=0x01;
        delay_ms(l);
        PORTA=0x00;
void main (void)
{
        DDRA=0xFF;
        PORTA=0x00;
        DDRD=0x00;
        PORTD=:0xFF;
GICR=0b01000000; // INTO: ON
MCUCR=0b000000ll; // INTO Mode: Rising Edge
#asm("sei") // Global enable interrupts
while (1) ;
}
```

تایمر/ کانتر صفر

تایمر/ کـانتر صـفر یـک تـایمر ۸ بیتـی مـی باشـد کـه دارای چهـار Mode کـاری Normal،CTC، Fast PWM و Correct PWM Phase می باشد. پین T0 به عنوان ورودی کانتر و پین OC0 خروجی بخش مقایسه ی تایمر مـی باشد. این تایمر دارای سه رجیستر به نام های TCCR0، TCNT0 و OCR0 می باشد که به ترتیب جهت پیکر بندی تایمر، مقدار شمارنده و مقدار مقایسه استفاده می شوند. همچنین این تایمر در رجیسترهای TIFR و TIMSK که بـه ترتیب رجیسترهای پرچم و وقفه تایمر می باشند با دیگر تایمرها مشترک می باشد.

مهم ترین رجیستر تایمر TTCR0 می باشد که بیت های Clock Select جهت انتخاب منبع کلاک تایمر و بیت های Wave Generation Mode برای تنظیم Mode کاری تایمر و بیـت هـای Compare Match Output Mode پیکربندی پین OC0 را تعیین می کنند. عملکرد بیت FOC0 نیز بررسی خواهد شد.

TCCR0	7	6	5	4	3	2	1	0
نام بیت	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00

Mode عملکرد تایمر با توجه به بیت های WGM01 و WGM00 به این ترتیب تعیین می شود:

Mode	WGM01	WGM00	Mode عملکرد
0	0	0	Normal
1	0	1	PWM, Phase Correct
2	1	0	Clear Timer on compare Match (CTC)
3	1	1	Fast PWM

Normal Mode O

* تایمر/ کانتر ساده ی ۸ بیتی

رجیسترهای TIMER 0:

TCCR0	7	6	5	4	3	2	1	0
نام بیت	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
سطح منطقی	0	0	0	0	0	X	X	X

تایمر/ کانتر صفر و یک دارای یک Prescaler مشترک بوده وضعیت منبع کلاک با توجه به بیت های Clock Select

تعیین می شود:

CS02	CS01	CS00	وضعیت منبع کلاک تایمر
0	0	0	بدون کلاک (متوقف)
0	0	1	کلاک سیستم (بدون تقسیم)
0	1	0	/۸ کلاک سیستم
0	1	1	/۶۴ کلاک سیستم
1	0	0	/۲۵۶ کلاک سیستم
1	0	1	/۱۰۲۴ کلاک سیستم
1	1	0	لبه ی پایین رونده ی پالس خارجی (T0)
1	1	1	لبه ی بالا رونده ی پالس خارجی (T0)

مقدار تایمر در هر لحظه از طریق رجیستر TCNT قابل خواندن و نوشتن است:

Bit	7	6	5	4	3	2	1	0
TCNT0	TCNT[7:0]							

در زمان سرریز تایمر، بیت TOV0 از رجیستر TIFR یک می شود.

TCCR0	7	6	5	4	3	2	1	0
نام بیت	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
سطح منطقی	0	0	0	0	0	0	0	X

ساده ترین راه برای چک کردن سرریز تایمر روش Polling می باشد.

مثال ۱: (تولید موج مربعی با μs T=512):

```
#include<mega16.h>
#define xtal 8000000
void delay ()
}
        TCCR0=0B00000010; II Timer Clock = CLK/8
        while(!TIFR&0x0l) ; II WAit Until Overflow
        TIFR=TIFRI0B00000001; // Clear TOV0
        TCCR0=0x00; // Stop Timer0
{
void main ()
}
DDRA=0xFF;
PORTA=0x00;
TCCR0=0x00;
TCNT0=0x00;
        while (1) {
        PORTA. 0=1;
        delay( );
        PORTA. 0=0;
        delay( );
}
}
```

مثال ۲: (کاهش دوره ی تناوب به 400 µs)

```
void delay ( )
        TCNT0=0x38;          //TCNT=56
        TCCR0=0B00000010;
        while(!TIFR&0x01);
        TIFR=TIFRI 0B00000001;
        TCCR0=0x00;
}
```

در صورتی که بیت فعال ساز عمومی وقفه (I) فعال بوده و بیت های TOIE0 یا TOIE2 در رجیستر TIMSK یک باشند می توان با استفاده از وقفه، از سرریز شدن تایمر به عنوان وقفه استفاده کرد:

TIMSK	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
نام بیت	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	ICIE0	TOIE0
سطح منطقی	0	0	0	0	0	0	0	X

مثال ۳: (موج مربعی T=400 µs با استفاده از وقفه ی سر ریز تایمر صفر)

```
include <mega16.h>
include <delay.h>
#define xtal 8000000
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
PORTA^=0xFF;
TCNT0=0x38;                    //TCNT=56
```

```
}
void main (void)
{
DDRA=0xFF;
PORTA=0x00;
TCCR0=0B00000010;   // Timer Clock = CLK/8
TIMSK=0x0l;          //Enable TIMER0 Overflow Interrupt
#asm("sei")          // Global enable interrupts
while (1);
}
```

تایمر/ کانتر با عملکرد مقایسه

BIT	7	6	5	4	3	2	1	0
OCR 0/2	OCR0[7:0]							

* محتوای رجیستر OCR0 به طور پیوسته با مقدار TCNT0 مقایسه می شود و در صورت برابری باعث تغییر وضعیت پین OC0 و یا وقفه ی تطابق می شود. در حالت برابری بیت OCF0 یا OCF1 یک شده و با فراخوانی سابروتین وقفه به صورت سخت افزاری صفر می شود. در صورت عدم استفاده از وقفه کاربر می تواند با نوشتن یک روی ایـن بیـت آن را پاک کند.

TIFR	7	6	5	4	3	2	1	0
نام بیت	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
سطح منطقی	0	0	0	0	0	0	X	X

TCCR0	7	6	5	4	3	2	1	0
نام بیت	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
سطح منطقی	X	0	X	X	0	X	X	X

تعریف وضعیت پین OC0 بوسیله بیت های COM00 و COM01 می باشد:

COM01	COM00	وضعیت پین OC0
0	0	غیر فعال (I/O معمولی)
0	1	Toggle در وضعیت تطابق
1	0	Clear در وضعیت تطابق
1	1	Set در وضعیت تطابق

* در صورت یک کردن بیت FOC0 یا FOC1 به صورت آنی مقدار رجیستر TCNT0 با مقدار OCR0 مقایسه شده و در صورت تطبیق مقایسه یک تغییر وضعیت روی پین OC0 ایجاد می شود. در این وضعیت بیت OCF0 یا OCF1 یک نشده و باعث ایجاد وقفه نیز نخواهد شد.

* در تمام حالت هایی که روی پین های OC شکل موج ایجاد می شود باید این پین به صورت خروجی تعریف شده باشد.

مثال ۴: (ایجاد پالس مربعی با T=512 µS روی پین OC0)

```
#include<mega16.h>
#define xtal 8000000
void main ( )
{
```

```
DDRB=0xFF;

PORTB=0x00;

TCNT0=0x00;

TCCR0=0B00010010;  //toggle OC0 on compare match

OCR0=0x63i //OCR0=99


while (1) ;

}
```

برای استفاده از وقفه باید علاوه بر یک بودن فعال ساز عمومی وقفه ها، بیت فعال ساز مقایسه ی وقفه نیز Set شود.

TIMSK	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
نام بیت	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
سطح منطقی	X	X	0	0	0	0	X	X

در این حالت ISR به صورت زیر تعریف می شود:

```
Interrupt [TIM0_COMP] void timer 0 _ comp_isr (void)

}
```

زیر برنامه ی سرویس وقفه

```
{
```

CTC Mode -۲

در این Mode تایمر همانند وضعیت نرمال با عملکرد مقایسه عمل می کند با این تفاوت که در زمان تطابق رجیسترهای OCR0 و TCNT0 مقدار رجیستر TCNT0 صفر شده و در واقع بر خلاف حالت قبل مقدار ماکزیمم

TCNT0 عدد موجود در رجیستر OCR0 می باشد. مقدار بیت های WGM00 و WGM01 در این Mode به ترتیب برابر ۰ و ۱ می باشد.

TIMSK	7	6	5	4	3	2	1	0
نام بیت	FOC0	WGM00	COM01	COM00	WM01	CS02	CS01	CS00
سطح منطقی	X	0	X	X	1	X	X	X

در این حالت فرکانس موج ایجاد شده روی پین OC0 از رابطه ی زیر بدست می آید:

$$f_{OCo} = \frac{f_{CLK\_I/O}}{2.N.(1+OCRo)}$$

مثال ۵: (ایجاد موج مربعی با فرکانس 5KHz روی پین OC0):

```
#include<mega16.h>
#define xtal 8000000
void main ( )
{
        DDRB=0xFF;
        PORTB=0x00;
        TCNT0=0x00;
        OCR0=0x63; //OCR0=99

        TCCR0=0B00011010; //toggle OC0 on compare match

        while(l);
```

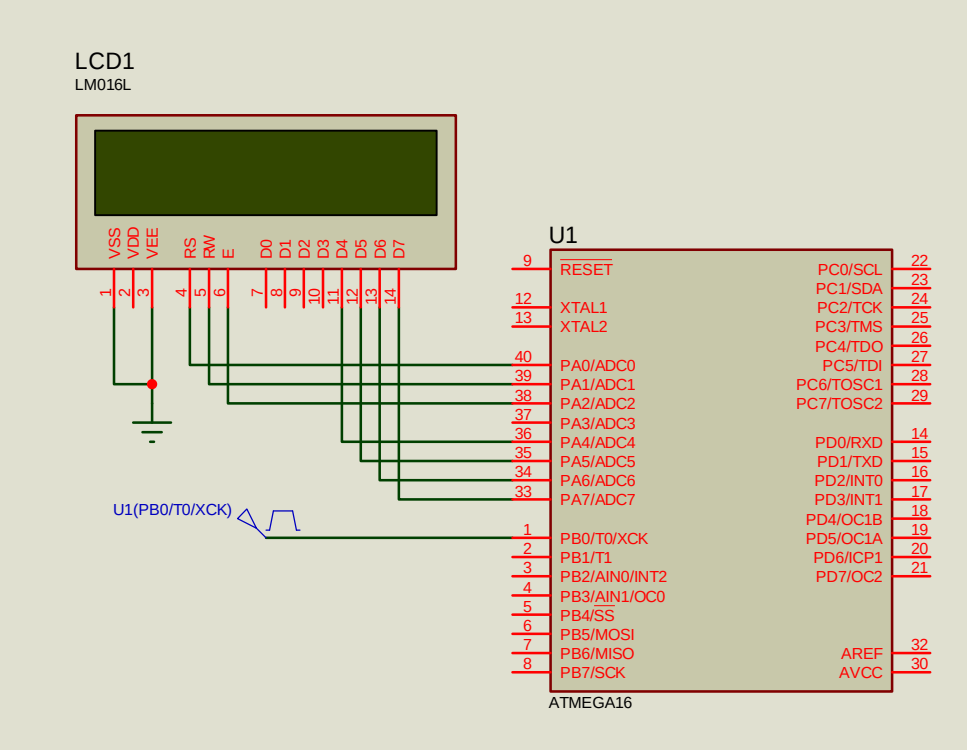
}

$$f_{OC} = \frac{8000000}{2.8.(1+99)} = 500Hz$$

وضعیت بیت های فعال ساز وقفه ی سرریز و وقفه ی مقایسه در CTC Mode همانند وضعیت نرمال می باشد. با یک بودن بیت OCIE0 وقفه ی مقایسه فعال می باشد و می توان در ISR این وقفه مقدار OCR0 را تغییر داد.

* مقدار دهی به رجیستر OCR0 باید با دقت انجام شود زیرا در Mode های غیر PWM این رجیستر دارای بافر دوبل نمی باشد. وجود بافر دوبل باعث می شود که اگر OCR0 به مقدار کمتری از TCNT0 تغییرکند، برای از دست نرفتن مقایسه ی فعالی مقدار جدید در بافر دوبل ذخیره شده و پس از سرریز این مقدار جدید در OCR0 بار گذاری شود.

پروژه ۷: فرکانس متر دیجیتال



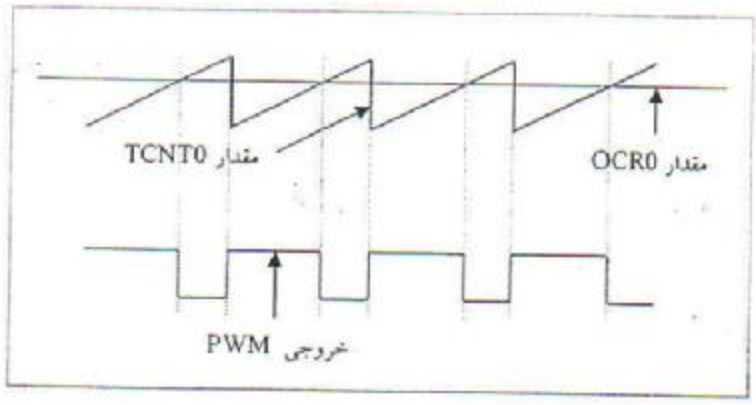

```
// Project: Frequency Meter

#include <mega16.h>

#include <delay.h>
```

```
#include <stdio.h>
#include <lcd.h>
#define xtal 8000000
#asm
                .equ __ lcd_port=0xlB ;PORTA

#endasm
unsigned long int timer0_ov;
unsigned long int in_freq;
unsigned char lcd_buff[20];
interrupt [TIM0_OVF) void timer0_ovf_isr(void)
{
timer0_ ov ++;
}
void main(void)
{
// Timer/Counter 0 initialization
// Clock source: TO pin Falling Edge
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCNT0=0x00;
OCR0=0x00;
TCCR0:": 0x00;
```

```
// Timer(s)/Counter(s) Interrupt(s) initialization TIMSK0x01;
// LCD module initialization
lcd_init(16);
while (1)
{
        TCCR0=0x06;                 // Start Timer TO pin Falling Edge

#asm("sei")             // Global enable interrupts
                delay_ms(l000);
                #asm ("Cli") ;  // Global disable interrupts
                in_freq = timer0_ov * 256 + TCNT0;
                sprintf(lcd_buff,"Freqency=%d",in_freq);
                lcd_clear();
                lCd_puts(lcd_buff);
                TCCR0=0x00;         //Stopt Timer0
                Timer0_ov=0;                //Prepare for next count
                TCNT0=0;            //Clear Timer0
                } ;
        }
```

Fast PWM Mode -۳

این حالت مشابه Mode نرمال می باشد با این تفاوت که پین OC0 فقط در حالـت برابـری رجیسـترهای TCNT0 و OCR0 تغییر حالت نمی دهد، بلکه در زمان سرریز رجیستر TCNT0 نیز مقدار این پین تغییر می کند.



مقدار بیت های WGM00 و WGM01 در این Mode برابر ۱ می باشد.

Bit	7	6	5	4	3	2	1	0
TCCR0	0	1	COM01	COM00	1	CS02	CS01	CS00

در Mode های PWM عملکرد بیت های COM00 و COM01 متفاوت از دو وضعیت قبلی و به صورت زیر می باشد:

COM01	COM00	وضعیت پین OC0
0	0	غیر فعال (I/O معمولی)
0	1	رزرو شده
1	0	Clear در وضعیت تطابق، Set در زمان سرریز (PWM غیر معکوس)
1	1	Set در وضعیت تطابق، Clear در زمان سرریز (PWM معوس)

برای محاسبه ی فرکانس موج PWM تولید شده می توان از فرمول زیر استفاده نمود:

مقدار بارگذاری شده در تایمر – ۲۵۶ = X

$$f_{PWM} = \frac{f_{clk} - I/O}{N.X}$$ N=Prescale= 1, 8, 64, 256, 1024

از وقفه ی سرریز تایمر می توان برای مقدار اولیه دادن به TCNT0 و یا تغییر مقدار OCR0 استفاده نمود، اگرچه بهتر است مقدار OCR0 در روتین وقفه ی مقایسه تغییر داده شود.

با مقدار اولیه دادن به TCNT0 می توان فرکانس موج PWM را تغییر داد.

مثال ۶: (تولید موج PWM با فرکانس 4KHz و زمان وظیفه ی ۲۰ درصد)

```
#include <mega16.h>
#define xtal 8000000
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
TCNT0=0x06;
}
void main(void)
}


PORTB=0x00;
DDRB=0x08;
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 1000.000 kHz
// Mode: Fast PWM top=FFh
// OC0 output: Non~Inverted PWM
TCCR0=0x6A; //0x7A for inverted PWM
```

```
TCNT0=0x06;
OCR0=0x38; //OCR0 = 56
// Timer(s)/Counter(s) Interrupt(s) initialization TIMSK=0x0l;
// Global enable interrupts
#asm("sei")
while (1);
}
```

$$f_{PWM} = \frac{8000000}{8(256-6)} = 4000Hz$$

$$DutyCycle = \frac{OCR0}{250} \times 100\% = \frac{50}{250} \times 100\% = 20\%$$

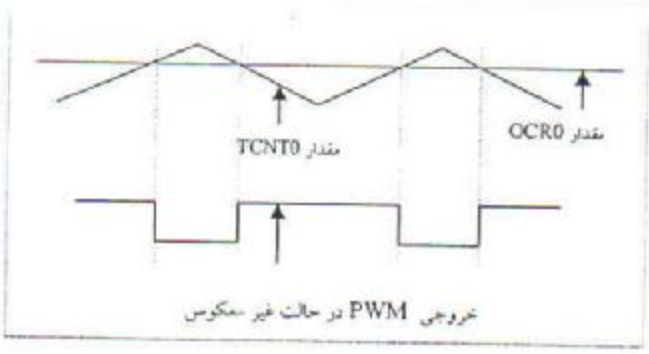
* با کمتر شدن OCR0 زمان وظیفه کمتر شده و تا حدی که مقدار صفر یک پالس سوزنی به عرض یک سیکل ایجاد خواهد کرد.

* با مقدار دهی ۲۵۶ به OCR0 مقدار مقایسه و سرریز برابر شده و پالس خروجی بسته به مقدار COM00 و COM01 همواره صفر یا یک خواهد بود.

Phase Correct PWM Mode -۴

این Mode شبیه حالت Fast PWM می باشد با این تفاوت که تایمر به صورت دو شیبه (Dual slope) عمل شمارش را انجام می دهد. به اینصورت که تایمر از عدد صفر شروع به شمارش کرده و به صورت افزایشی تا عدد 0xFF افزایش می یابد و بعد از رسیدن به این مقدار عدد موجود در بافر OCR0 در رجیستر OCR0 بارگذاری می شود. بعد از این لحظه جهت شمارش تایمر عوض شده و به صورت کاهشی تا عدد صفر می شمارد با رسیدن به این عدد پرچم سرریز تایمر یک شده و در صورت یک بودن بیت فعال ساز وقفه، برنامه به ISR سرریز تایمر منشعب شده و بیت پرچم وقفه بوسیله سخت افزار صفر می شود. مسئله ی مهم این است که در هر دو حالت شمارش افزایشی و کاهشی عمل

مقایسه بین رجیسترهای TCNT0 و OCR0 انجام می گیرد و در صورت برابری پرچم OCF0 یک شده و باعث تغییر در پین OC0 می شود.



تغییر پین OC0 مطابق جدول زیر می باشد.

COM01	COM00	وضعیت پین OC0
0	0	غیر فعال (I/O معمولی)
0	1	رزرو شده
1	0	Clear در وضعیت تطابق، Set در زمان شمارش افزایشی (PWM غیر معکوس) Set در وضعیت تطابق، در زمان شمارش کاهشی
1	1	شمارش افزایشی (PWM معکوس) Clear در وضعیت تطابق، در زمان شمارش کاهش

* در صورت تغییر مقدار رجیستر OCR0 مقدار جدید در بافر این رجیستر نوشته می شود و در زمان رسیدن به 0xFF رجیستر OCR0 به روز می شود.

* پرچم سرریز تایمر صفر زمانی فعال می شود که رجیستر TCNT0 برابر صفر شود و نه صفر، بنابراین باید دقت داشت که اگر در زمان شروع به کار مقدار این رجیستر صفر باشد پرچم سرریز فعال خواهد شد.

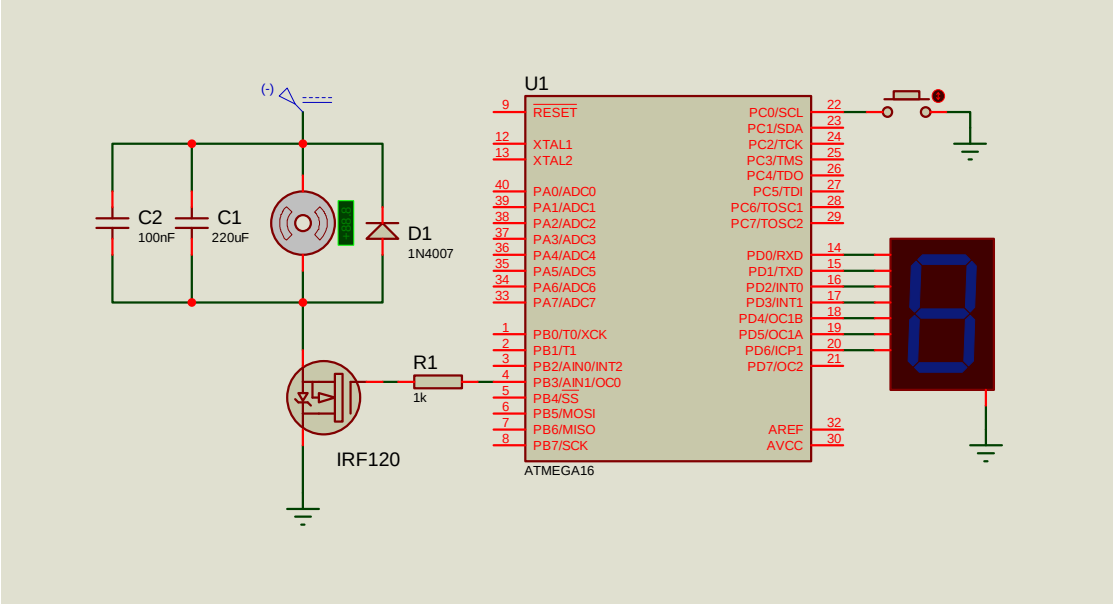
* فرکانس PWM در حالت Phase Correct از رابطه ی زیر قابل محاسبه است:

$$f_{PWM} = \frac{f_{clk\_I/O}}{N \times 510}$$ N= 1, 8, 64, 256, 1024

* رابطه ی بالا نشان می دهد فرکانس موج PWM ثابت است و ارتباطی به رجیستر های OCR0 و TCNT0 ندارد.

* در حالت PWM غیر معکوس با افزایش مقدار OCR0 مقدار متوسط موج PWM افزایش یافته و با کاهش آن مقدار متوسط کاهش می یابد و در حالت PWM معکوس، عکس این قضیه صحیح است.

پروژه ۸: کنترل موتور DC با PWM




```
//Project: DC Motor Control

include < mega16.h>

# define xtal 1000000

char digits[8]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07}; unsigned char;

unsigned char p_state;

unsigned char key;

unsigned char i;
```

```
void main(void)
{
PORTB=0x00;
DDRB=0xFF;
DDRD = 0xFF;
PORTD = digits[0];

DDRC = 0x00;
PORTC = 0xFF;
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: 15.625 kHz
// Mode: Phase correct PWM top=FFh
// OC0 output: Non-Inverted PWM
TCCR0=0x63;
TCNT0=0x00;
OCR0=10;
while (1)
{
        if (! PINC. 0)
        {
        if (key!=p_state)
{
```

```
        if(i==7)
{
        i=0;
        PORTC=digits[0];
        } else
i++;
        PORTD = digits[i];

        OCR0 = i*10+10;
        p_state=key;
        } ;
        }
        else
        p_state=0xFF;
        } ;
}
```

عملکرد تایمر دو

به طور کلی عملکرد تایمر ۲ مشابه تایمر صفر می باشد و رجیسترهای مربوطه با همان نام و دارای پسوند ۲ می باشند، با این تفاوت که تایمر ۲ بر خلاف تایمرهای صفر و یک نمی تواند از پایه خارجی T0 یا T1 کلاک دریافت کند و در عوض می توان با وصل کردن یک کریستال ۳۲,۷۶۸ کیلوهرتز به پین های TOSC1 و TOSC2 از آن در وضعیت آسنکرون جهت RTC استفاده نمود. از آنجایی که تایمر ۲ دارای Prescaler مجزا از دو تایمر ۰ و ۱ می باشد با تقسیم کریستال ۳۲۷۶۸ هرتز بر ۱۲۸ می توان به زمان سر ریز ۱ ثانیه که مناسب برای عملکرد ساعت است دست پیدا کـرد. تنظیمـات

Prescale برای این تایمر به صورت زیر می باشد:

CS02	CS01	CS00	وضعیت منبع کلاک تایمر
0	0	0	بدون کلاک (متوقف)
0	0	1	کلاک سیستم (بدون تقسیم)
0	1	0	/۸ کلاک سیستم
0	1	1	/۳۲ کلاک سیستم
1	0	0	/۶۴ کلاک سیستم
1	0	1	/۱۲۸ کلاک سیستم
1	1	0	/۲۵۶ کلاک سیستم
1	1	1	/۱۰۲۴ کلاک سیستم

پیکربندی RTC با رجیستر وضعیت اسنکرون یا ASSR انجام می شود:

Bit	7	6	5	4	3	2	1	0
ASSR					AS0	TCN0UB	OCR0UB	TCR0UB

Asynchronous Timer / Counter 2: با Set کردن این بیت منبع کلاک تایمر ۲ از کلاک سیستم به کریستال خارجی در پایه های TOSC1 و TOSC2 تغییر می کند. با تغییر دادن این بیت ممکن است مقدار رجیسترهای TCNT2، OCR2 و TCCR2 خراب شود.

Timer/ Counter 2 Update Busy: برای تضمین عملکرد صحیح در وضعیت آسنکرون رجیسترهای تایمر ۲ بر خلاف تایمر ۰ و ۱ به صورت بافر شده بروز میشوند. بدین ترتیب که وقتی روی رجیستر TCNT2 مقداری نوشته شود، بیت TCN0UB یک می شود و مقداری که در رجیستر موقتی ذخیره شده است به TCNT2 منتقل می شود. با اتمام بروز رسانی TCNT2 این بیت توسط سخت افزار صفر می شود. صفر بودن OCR0UB نشان دهنده ی آمادگی

TCNT2 برای پذیرفتن مقدار جدید است.

Output Compare Register 2 Update Busy: این بیت همانند TCN0UB بوده با این تفاوت که بر روی رجیستر OCR2 عمل می کند.

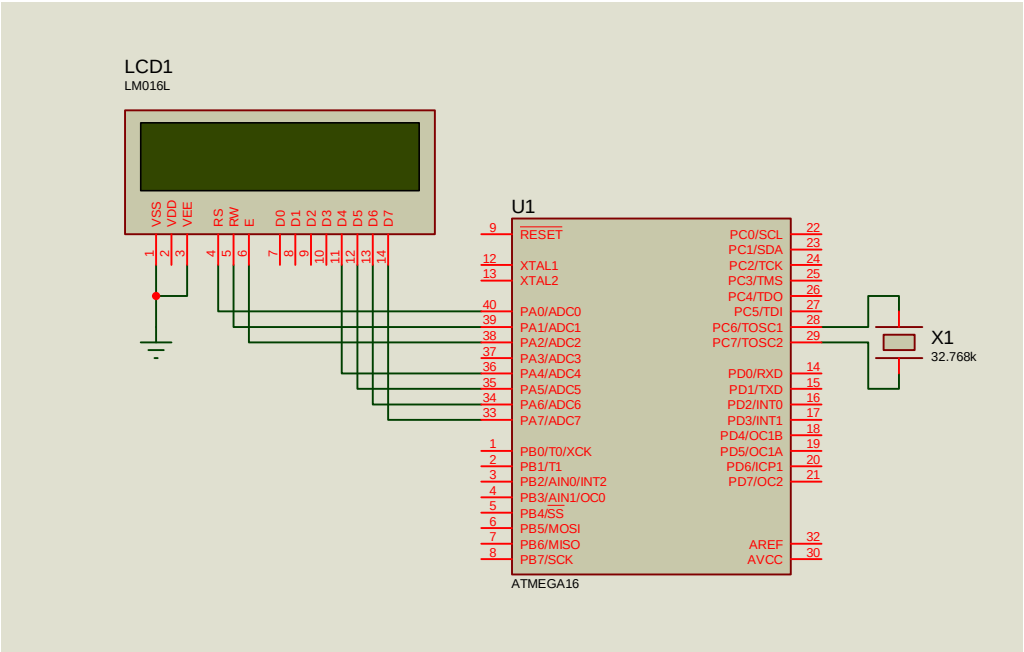
Timer/ Counter Control register 2 update busy: این بیت همانند TCN0UB بوده با این تفاوت که بر روی رجیستر TCCR2 عمل می کند.

* در حالتی که پرچم مشغول بودن یک رجیستر یک می باشد، نوشتن بر روی آن رجیستر باعث می شود که مقدار بروز شده صحیح نباشد و ممکن است باعث وقفه ی ناخواسته شود.

* مکانیسم خواندن این سه رجیستر متفاوت می باشد، بدین صورت که زمان خواندن TCNT2 مقدار خود رجیستر خوانده شده و با خواندن OR2 و TCCR2 مقدار موجود در رجیستر موقت خوانده می شود.

* تایمر ۲ در وضعیت آسنکرون در حالت Power-save نیز فعال بوده و پس از سرریز شدن تایمر از وضعیت Power-save خارج شده و در صورت فعال بودن وقفه، ISR را اجرا نموده و مجدداً وارد حالت Power-save می شود.

پروژه ۹: ساعت با RTC میکروکنترلر



```
//Project: Real Time Clock

#include <mega16.h>

#include <lcd.h>

#include <stdio.h>

#define xtal 8000000

#asm

        .equ __ lcd_port=0xlB ;PORTA

#endasm

unsigned char second, minute, hour;

unsigned char lcd_buff[l0];

interrupt [TIM2_OVF] void timer2_ovf_isr(void)

{

        if (second==59)

        {

        second=0;

        if (minute==59)

        {

        minute=0;

        if(hour==24)

        hour =0;

        else

        hour++i

        }
```

```
        else
        minute++;
        }
        else
second++;
sprintf(lcd_buff,"Time = %d:%d:%d",hour, minute, second);
lcd_clear () ;
lcd_putS(lcd_buff);
}
void main(void)
{
// Clock source: TOSC1 pin
// Clock value: PCK2/128
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x08;
TCCR2=0x05;
TCNT2=0x00;
OCR2=0x00;
// Timer(s)/Counter(s) Interrupt(s) initialization TIMSK=0x40;
Lcd_init (16);
#asm("sei") // Global enable interrupts
while (1);
```

}

تایمر/ کانتر یک

تایمر یک تایمری ۱۶ بیتی است و در آن علاوه بر امکانات تایمر صفر، یک بخش دیگر بـه نـام بخـش Capture بـه آن افزوده شده است. این بخش در زمان های خاص، عدد شمارش شده توسط تایمر یک و زمان سپری شده را ثبت کرده و از طریق آن امکان اندازه گیـری هـای زمـانی را فـراهم مـی آورد. تـایمر یـک دارای پـنج Mode کـاری بـه نـام هـای Phase Correct PWM Mode Fast PWM, CTC, Normal و Phase and Frequency Correct Mode می باشد. Mode های PWM در تایمر ۱ بسیار متنوع و دارای ۱۲ حالت PWM می باشد. در این تایمر پین T1 به عنوان ورودی کانتر و پین های OC1A و OC1B به عنوان خروجی مقایسه گر عمل می کننـد. همچـین پـین ICP1 برای ورودی بخش Capture تایمر یک می باشد. بـه علـت ۱۶ بیتـی بـودن تـایمر، رجیسـترهای TCNT1 و OCR1A و OCR1B شانزده بیتی می باشند که هر کدام دارای دو بایت L و H هستند. همچنین تایمر یک دارای دو واحد مقایسه ی مجزا می باشد که مقدار موجود در رجیسترهای OCR1A و OCR1B را با TCNT مقایسه کـرده و در صورت برابری وضعیت پین های OC1A و OC1B را تغییر می دهند. همچنین رجیستر ICR1 نیـز کـه رجیسـتر واحد Capture است رجیستری ۱۶ بیتی می باشد. رجیسترهای شانزده بیتی تایمر ۱ شـامل OCR1B, OCR1A, TCNT1 و ICR1 می باشند. رجیسترهای ۸ بیتی TCCR1A و TCCR1B کنترل تایمر را بر عهده دارند:

TCCR1A	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
نام بیت	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10

TCCR1B	7	6	5	4	3	2	1	0
نام بیت	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

Mode کاری تایمر بوسیله ی بیت های WGM10 ، WGM11 وWGM12 و WGM13 تعیین می شود:

	WGM13	WGM12	WGM11	WGM10	Mode کاری	TOP	TOV=1
0	0	0	0	0	Normal	0xFFFF	0xFFFF
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	0
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	0
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	0
4	0	1	0	0	CTC	0CR1A	0xFFFF
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	0
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	0
10	1	0	1	0	PWM, Phase Correct	lCR1	0
11	1	0	1	1	PWM, Phase Correct	OCR1A	0
12	1	1	0	0	CTC	ICR1	0xFFFF
13	1	1	0	1	رزرو شده	-	-
14	1	1	1	0	Fast PWM	ICR1	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP

* تعریف TOP: تایمر وقتی به مقدار TOP می رسد که برابر با بالاترین مقدار در رشته ی شمارش خود است. این مقدار می تواند مقادیر ثابتی مثل 0x03FF، 0x01FF و یا 0x00FF بوده و یا مقدار نگهداری شده در یکی از رجیسترهای OCR1A یا ICR1 باشد.

FOC1A و FOC1B: بیت های Force بخش مقایسه گر که عملکرد آن ها همانند FOC0 در تایمر صفر و دو می

باشد. به این ترتیب که در Mode های غیر PWM ، یک کردن این بیت بدون اینکه وقفه ای ایجاد کند در صورت تطبیق مقایسه، باعث تغییر وضعیت پین های OC1A و OC1B مطابق با وضعیت بیت های COM در TCCR1 می شود.

بیت های COM1A0 و COM1A1 و COM1B0 و COM1B1: تغییر وضعیت پین های OC1A و OC1B را در حالت تطبیق معین می کنند که مقدار آن ها بسته به Mode کاری عملکرد متفاوتی را ایجاد می کند. بنابراین در بررسی Mode های مختلف آن را مطالعه خواهیم کرد.

بیت های CS10 و CS11 و CS12: برای تعیین منبع کلاک تایمر می باشند:

وضعیت منبع کلاک تایمر	CS00	CS01	CS01
بدون کلاک (متوقف)	0	0	0
کلاک سیستم (بدون تقسیم)	1	0	0
/۸ کلاک سیستم	0	1	0
/۶۴ کلاک سیستم	1	1	0
/۲۵۶ کلاک سیستم	0	0	1
/۱۰۲۴ کلاک سیستم	1	0	1
لبه ی پایین رونده ی پالس خارجی (T0)	0	1	1
لبه ی بالا رونده ی پالس خارجی (T0)	1	1	1

بیت ICES1: بیت تعیین لبه ی ورودی بخش Capture از پین ICP1 با صفر بودن این بیت لبه ی پایین رونده و با

TIFR	7	6	5	4	3	2	1	0
نام بیت	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
سطح منطقی	0	0	X	X	X	X	0	0

یک بودن آن لبه ی بالا رونده باعث تریگر می شود.

بیت ICNC1: بیت فعال ساز حذف نویز در ورودی پین ICP1

نتایج حاصل از کارکرد تایمر ۱ در ۴ بیت رجیستر TIFR به نام های TOV1 (پرچم سرریز) OCF1A (پرچم تطابق مقایسه گر A) OCF1B (پرچم تطابق مقایسه گر B) و ICF1 (پرچم بخش Capture تایمر ۱) منعکس می شوند:

یک شدن هر یک از این پرچم ها در صورت فعال بودن بیت فعال ساز عمومی (I) و فعال بودن وقفه ی مربوطه در رجیستر TIMSK می تواند باعث انشعاب برنامه به ISR مربوط به آن وقفه شود:

TIMSK	7	6	5	4	3	2	1	0
نام بیت	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
سطح منطقی	0	0	X	X	X	X	0	0

با اجرا شدن ISR به صورت خودکار بیت پرچم وقفه صفر شده و یا می تواند با نوشتن یک روی آن بوسیله ی نرم افزار آن را پاک کرد.

۱- Normal Mode

این Mode همانند مشابه آن در تایمر صفر می باشد با این تفاوت که تایمر تا عدد 0xFFFF شمارش کرده و با رسیدن به ان تایمر سر ریز کرده و بیت TOV1 یک شده و در صورت فعال بودن وقفه می تواند باعث اجرای ISR مربوطه شود. در Mode عادی هر دو مقایسه گر A و B فعال بوده و هر کدام به طور مستقل عمل مقایسه را روی رجیستر TCNT1 و OCR1A و OCR1B انجام می دهند. در صورت برابری بیت OCF1A یا OCF1B یک شده و خروجی OC1A یا OC1B مطابق جدول زیر تغییر وضعیت داده و در صورت فعال بودن وقفه می تواند باعث ایجاد وقفه شوند.

COM1A/COM1B1	COM1A0/COM1B0	وضعیت پین OC1A
0	0	غیر فعال (I/O معمولی)
0	1	Toggle در وضعیت تطابق
1	0	Clear در وضعیت تطابق
1	1	Set در وضعیت تطابق

* در صورت استفاده از OC1A یا OC1B برای تولید شکل موج، باید این پین ها به صورت خروجی پیکبر بندی شوند.

مثال ۷: (تولید دو شکل موج با دوره تناوب ۱۳۱ میلی ثانیه و اختلاف فاز ۱۰ میلی ثانیه)

```
#include <mega16.h>

#define xtal 8000000

void_main(void)

{

PORTD=0x00;

DDRD=0x30;

// Mode: Normal top=FFFFh

TCCR1A=0x50; //toggle OC1A & OC1B

TCCR1B=0x02; //Clock/8

OCR1AH=0x00;

OR1AL=0xFF; //0CR1A=255

OCR1BH=0x28;

OCR1BL=0x0F; //OCR1B=10255

while (1) ;
```

}

$$T = 2 \times 2^{16} \times 1\mu s = 131072\mu s = 131ms$$

$$\text{اختلاف فاز} = OCR1B - OCR1A = 10255 - 255 = 10000\mu = 10ms$$

CTC Mode - ۲

در این حالت مقدار رجیستر OCR1A با ICR1 مقایسه می شود و در صورت برابری مقدار رجیستر TCNT1 برابر صفر می شود. بنابراین در این حالت مقدارTOP تایمر را با توجه به مقدار موجود در بیت های WGM مقدار رجیسترهای OCR1A یا ICR1 تعیین می کنند.

با رسیدن تایمر به مقدار TOP خود بر حسب اینکه مقدار ماکزیمم OCR1A یا ICR1 انتخاب شده باشد به ترتیب پرچم های OCF1A یا ICF1 یک شده و در صورت فعال بودن وقفه از آن می توان برای تغییر دادن مقدار مقایسه استفاده کرد. این عمل باید با دقت صورت گیرد زیرا رجیستر مقایسه ی تایمر ها فقط در Mode های PWM دارای بافر دوبل می باشند. در این حالت فرکانس موج ایجاد شده روی پایه های OC1A یا OC1B مطابق رابطه ی زیر می باشد:

$$f_{OC1x} = \frac{f_{CLK\_I/O}}{2.N.(1+OCR1A)}$$

مثال ۸: (تولید موج مربعی با فرکانس ۱ کیلوهرتز روی پایه ی OC1A)

```
#include <mega16.h>
#define xtal 8000000
void main(void)
{
PORTD=0x00;
DDRD=0x30;
// Mode: CTC top=01F3h
```

```
TCCR1A=0x40;
TCCR1B=0x0A;
OCR1AH=0x0l;
OCR1AL=0XF3; //OCRIA=499
while (l);
}
```

Fast PWM Mode -۲

بر خلاف تایمر های صفر و دو که در آن موج های PWM تولید شده دارای دقت ثابت ۸ بیتی هستند، تایمر ۱ قادر است سیگنال های PWM ای با دقت متغیر را ارائه کند، این مسئله باعث می شود که کاربر بتواند علاوه بر تغییر Duty Cycle فرکاس موج را به صورت سخت افزاری کنترل کند (بدون مقدار اولیه دادن به TCNT1)

PWM سریع دارای پنج Mode می باشد: (۵، ۶، ۷، ۱۴، ۱۵ = WGM[3:0])

۱- PWM سریع ۸ بیتی (0xFF=TOP)

۲- PWM سریع ۹ بیتی (0xFF=TOP)

۳- PWM سریع ۱۰ بیتی (0x03FF=TOP)

۴- PWM سریع با ICR1=TOP

۵- PWM سریع با OCR1A=TOP

در این Mode تایمر از صفر تا مقدار TOP خود شروع به شمارش کرده و پس از سرریز مجدداً از صفر شروع به کار می کند. در صورتی که مقایسه ی خروجی در حالت PWM غیر معکوس باشد. در حالت تطبیق مقایسه بین رجیسترهای TCNT1 و OCR1x پین OC1x یک شده و با رسیدن به مقدار TOP پاک می شود. در صورتی که خروجی PWM معکوس باشد وضعیتی عکس وجود خواهد داشت. دقت موج PWM خروجی می تواند مقادیر ثابت، ۸ ، ۹ یا ۱۰ بیتی داشته و یا بوسیله ی رجیسترهای ICR1 یا OCR1A به مقدار دلخواه تنظیم شود. در این حالت حداقل

مقدار مجاز ۲ بیت (با دادن مقدار 0x0003 به رجیسترهای ICR1 یا OCR1x) و حداکثر آن ۱۶ بیت می باشد (با دادن مقدار 0xFFFF به رجیسترهای ICR1 یا OCR1x).

دقت موج PWM بر حسب مقدار ماکزیمم از رابطه ی زیر بدست می آید:

$$\text{resolution} = \frac{\log(TOP+1)}{\log(2)}$$

با رسیدن تایمر به مقدار TOP پرچم سرریز TOV1 فعال شده و با تطبیق مقایسه نیز بیت OCF1A و یا OCF1B یک می شود. در این حالت ها اگر وقفه ی مربوطه فعال شده باشد می توان در ISR آن وقفه مقدار مقایسه را تغیر داد. باید توجه داشت که مقدار جیسترهای مقایسه باید از مقدار TOP کمتر باشد در غیر اینصورت هیچگاه مقایسه ای صورت نمی گیرد.

COM1A1 /COM1B1	COM1A0 /COM1B0	وضعیت پین OC1A یا OC1B
0	0	غیر فعال (I/O معمولی)
0	1	اگر WGM[3:0]=15 باشد: Toggle پین OC1A در وضعیت تطابق و OC1B پین I/O معمولی برای دیگر حالت های WGM[3:0] : غیر فعال (I/O معمولی)
1	0	Clear در وضعیت تطابق و Set در وضعیت TOP (PWM غیر معکوس)
1	1	Set در وضعیت تطابق و Clear در وضعیت TOP (PWM معکوس)

تغییر وضعیت پین های OC1A و OC1B در حالت تطبیق و سرریز مطابق جدول زیر خواهد بود:

فرکانس موج PWM حاصل از رابطه ی زیر بدست می آید:

$$f_{PWM} = \frac{f_{clk-I/O}}{N.(1+TOP)}$$

مثال ۹: (موج PWM با فرکانس ۱ کیلو هرتز و زمان وظیفه ی ۲۵ درصد)

```
#include <mega16.h>

#define xtal 8000000

void main(void)

{

PORTD=0x00;

DDRD=0x20;

// Mode: Fast PWM top=03FFh

// OC۱A output: Non-Inv.

// OC1B output: Disconnected

TCCR1A=0x83;

TCCR1B=0x۰A; //10 Bit PWM

TCNT1H=0x00;

TCNTlL=0x00;

OCR1AH=0x00;

OCR1AL=0xFF;

OCR1BH=0x00;

OCR1BL=0x00;

while (1);

}
```

$$f_{PWM} = \frac{8000000}{8.(1+1023)} = 976 \approx 1KHz \qquad DutyCycle = \frac{256}{1024} \times 100\% = 25\%$$

* با مقدار اولیه دادن به۱ TCNT در ISR سرریز تایمر، می توان به فرکانس دقیق ۱ کیلوهرتز رسید:

```
interrupt [TIM1_OVF] void tirner1_ovf_isr(void)
{
TCNTl=24 ;
}
TIMSK=0x04; //Enable TOVI
#asrn (" sei ") //Enable· Interrupts
```

۳- Phase Correct Mode

PWM تصحیح فاز دارای پنج Mode کاری می باشد: (۱، ۲، ۳، ۱۰، ۱۱= [3:0]WGM1)

۱- PWM تصحیح فاز ۸ بیتی (0xFF=TOP)

۲- PWM تصحیح فاز ۹ بیتی (0x1FF=TOP)

۳- PWM تصحیح فاز ۱۰ بیتی (0x03FF=TOP)

۴- PWM تصحیح فاز با ICR1=TOP

۵- PWM تصحیح فاز با OCR1A=TOP

در این Mode تایمر به طور پیوسته از مقدار صفر تا TOP و از TOP تا صفر می شمارد. در حالت PWM غیر معکوس در حالی که تایمر به صورت صعودی می شمارد در لحظه ی برابری رجیسترهای TCNT1 و OCR1x پین OC1x صفر شده و در حالت شمارش نزولی با تطابق دو رجیستر این پین یک می شود. در حالت PWM معکوس، عکس این قضیه برقرار است.

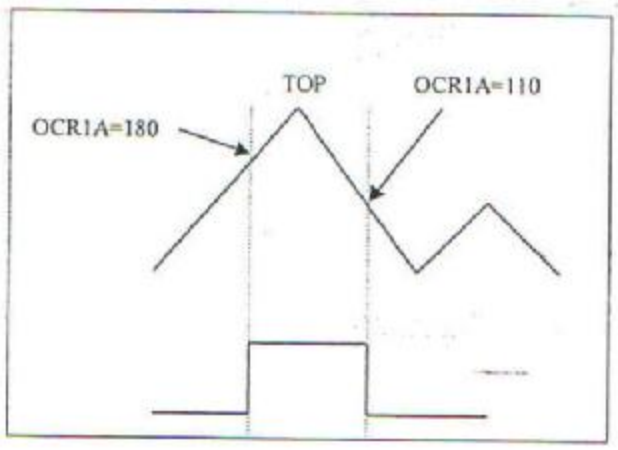
دقت موج PWM خروجی می تواند مقادیر ثابت ۸، ۹ یا ۱۰ بیتی داشته و یا بوسیله ی رجیسترهای ICR1 یا OCR1A به مقدار دلخواه تنظیم شود. در این حالت حداقل مقدار مجاز ۲ بیت (با دادن مقدار 0x0003 به رجیسترهای ICR1 یا OCE1x) و حداکثر آن ۱۶ بیت می باشد (با دادن مقدار 0xFFFF به رجیستر ICR1 یا

(OCR1x).

دقت موج PWM بر حسب مقدار ماکزیمم از رابطه ی زیر به دست می آید:

$$\text{resolution} = \frac{\log(TOP+1)}{\log(2)}$$

پرچم سرریز تایمر TOV1 با رسیدن تایمر به مقدار صفر یک خواهد شد و با تطبیق مقایسه نیز بیت OCF1A یا OCF1B یک می شود. در این حالت ها اگر وقفه ی مربوطه فعال شده باشد برنامه می تواند به ISR آن وقفه منشعب شود. مقدار مقایسه (OCRx) را در ISR یا هر زمان دیگر می توان تغییر داد اما این مقدار در بافر رجیسترهای OCR1A و OCR1B ذخیره شده و با رسیدن تایمر به مقدار TOP در خود رجیستر Load می شود بنابرین تغییر دادن مقدار رجیسترهای OCR1x به دلیل تغییر آن با رسیدن به TOP می تواند باعث خروجی PWM نامتقارن شود.



مشکل بالا در PWM تصحیح فاز و فرکانس با بروز کردن رجیسترهای OCR1x در زمان رسیدن به صفر، حل می شود.

تغییر وضعیت پین ها OC1A و OC1B در حالت تطبیق مقایسه و سرریز مطابق جدول زیر خواهد بود:

COM1A1 /COM1B1	COM1A0 /COM1B0	وضعیت پین OC1A یا OC1B

غیر فعال (I/O معمولی)	0	0
اگر WGM[3:0]=9.14 باشد: Toggle پین OC1A در وضعیت تطابق و OC1B پین I/O معمولی برای دیگر حالت های WGM[3:0] : غیر فعال (I/O معمولی)	1	0
Clear در وضعیت تطابق و شمارش صعودی Set در وضعیت تطابق و شمارش نزولی	0	1
Set در وضعیت تطابق و شمارش صعودی. Clear در وضعیت تطابق و شمارش نزولی.	1	1

فرکانس موج PWM در حالت تصحیح فاز نصف حالت Fast PWM بوده و از رابطه ی زیر بدست می آید:

$$f_{PWM} = \frac{f_{clk} - I/O}{2.N.(1+TOP)}$$

مثال ۱۰: در برنامه ی مثال قبل Mode تایمر را از Fast PWM به Phase Correct PWM تغییر داده و نصف شدن فرکانس PWM خروجی را مشاهده کنید:

```
TCCR1A= 0x83;
TCCR1B=0x02;
OCR1AL=0Xff;
```

۵- Phase and Frequency Correct Mode

همانطور که گفته شد به دلیل بروز کردن رجیستر OCR1x با رسیدن به TOP ممکن است شکل موج خروجی نامتقارن شود. بنابراین برای حل این مشکل Mode پنجم تایمر یک این رجیستر را با رسیدن به صفر بروز می کند. تفاوت دیگر این Mode و عملکرد قبلی در این است که تایمر در دو حالت زیر کار می کند: (WGM1[3:0]=9.8)

۱- PWM تصحیح فاز و فرکانس ICR1=TOP

۲- PWM تصحیح فاز و فرکانس با OCR1A=TOP

واحد Capture تایمر یک

عملکرد این واحد به این صورت است که در اثر تریگر شدن ورودی Capture از پین ICP1 یا خروجی مقایسه گر

آنالوگ مقدار موجود در رجیستر TCNT1 در رجیستر TCR1 نوشته شده و همزمان پرچم Capture تایمر یک (ICF1) یک می شود. در این زمان در صورت فعال بودن بیت پرچم ورودی Capture (TICIE1) این تریگر شدن می تواند باعث ایجاد وقفه شود. با اجرا شدن ISR به طور خودکار بیت ICF1 صفر شده و یا در صورت فعال نبودن وقفه می تواند با نوشتن یک بر روی آن پاک شود.

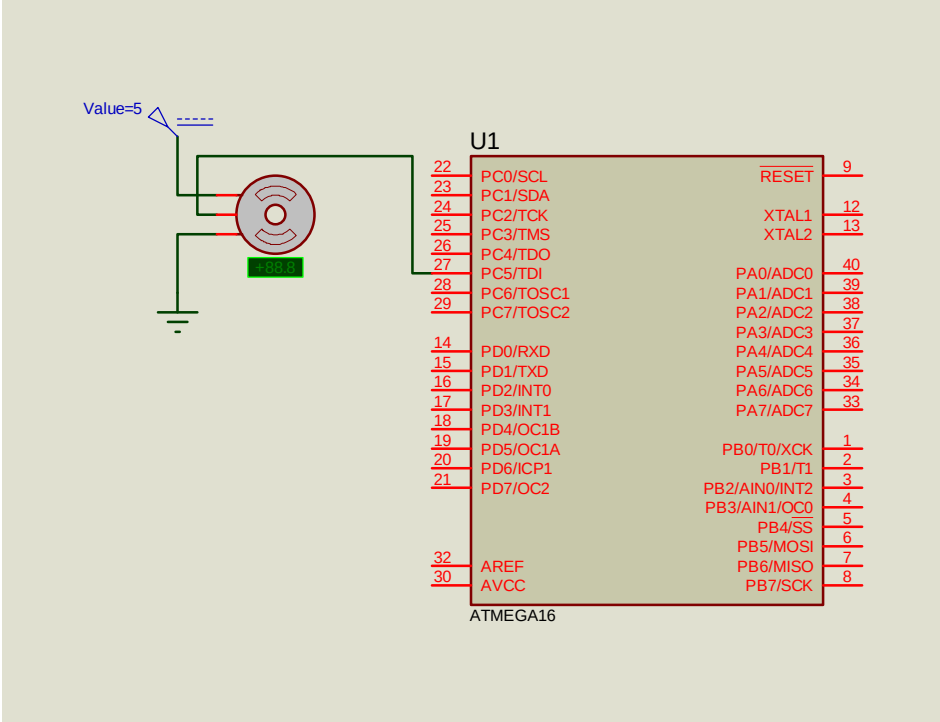
Bit	7	6	5	4	3	2	1	0
ICR1H	ICR1[15:8]							
ICR1L	ICR1[7:0]							

* رجیستر ICR1 به جز در حالتی که به عنوان TOP جهت مقایسه به کار می رود (Mode های ۸، ۱۰، ۱۲ و ۱۴) یک رجیستر فقط خواندنی است.

همانطور که گفته شد تریگر شدن واحد Capture می تواند از دو منبع مختلف صورت گیرد که این از طریق بیت ACIC در رجیستر ACSR صورت می گیرد. صفر بودن این بیت پین ICP1 و یک بودن آن خروجی مقایسه کننده ی آنالوگ را انتخاب می کند. همچنین نوع سیگنال ورودی از پین ICP1 بوسیله بیت ICES1 از رجیستر TCCR1B تعیین می شود، به این ترتیب که صفر بودن اینن بیت لبه ی پایین رونده و یک بودن آن لبه ی بالا رونده ی سیگنال ورودی را انتخاب می کنند.

ورودی Capture دارای یک واحد کاهش نویز نیز می باشد که استفاده از یک فیلتر دیجیتال ایمنی ورودی را بهبود می بخشد. این واحد با یک کردن بیت ICNC1 از رجیستر TCCR1B فعال می شود. با فعال شدن این فیلتر باید سیگنال نمونه بردرای شده روی پایه ی ICP1 برای چهار سیکل کلاک معتبر باشد.

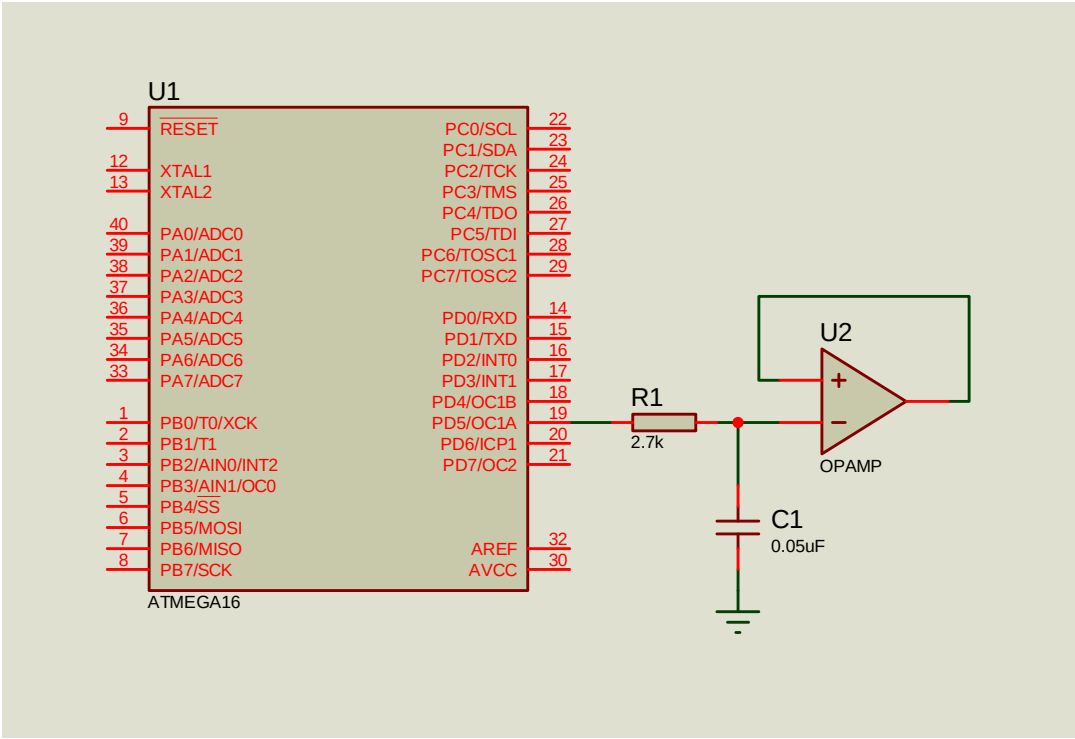
پروژه ۱۰: کنترل سرو موتور




```
//Project: Servo Motor Controller

//Chip type: ATmega16

//Clock frequency:: 8.000000 MHz

#include <mega16.h>

#inc1ude <delay.h>

#define xtal 16000000

void main (void)

{

PORTD=0x00;

DDRD=0x20;

// Timer/Counter 1 initialization
```

```
// Clock source: System Clock
// Clock value: 2000.000 kHz
// Mode: Ph. & fr. cor. PWM top=ICR1
// OC1Aoutput: Non-Inv.
// OC1B output: Discon.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x80;
TCCR1B=0x12;
ICR1H=0x4E;
ICR1L=0x20; //ICR=20000
OCR1AH=0x03;
OCR1AL=0xE8; //1000
while (1)
{
        for (OCR1A=1000;0CR1A<2000;OCR1A++)
        delaY_ms(l);
        for(OCRlA=2000;0CRlA>1000;OCR1A--:)
        delaY_ms(l);
        } ;
}
```

پروژه ۱۱: تولید موج سینوسی



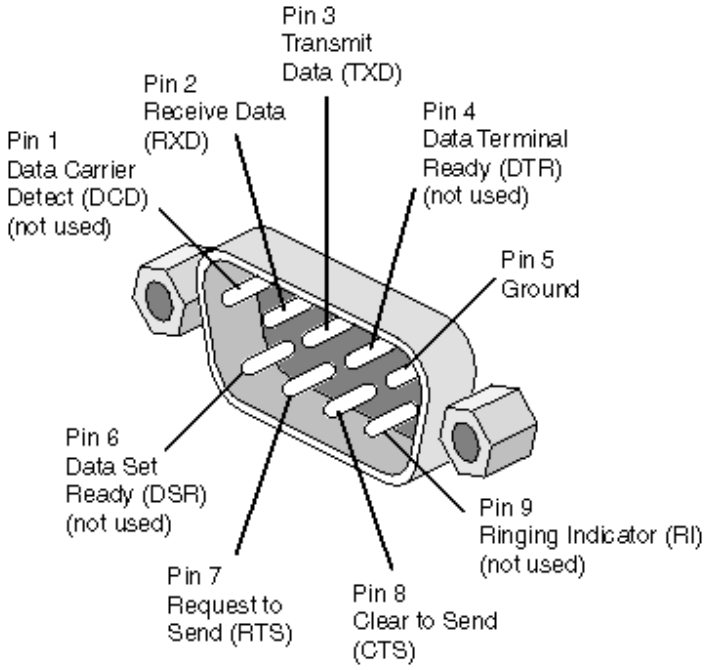

```
#include <mega16.h>

#define xtal 8000000

flash char sinewave[256]={

0x80,0x83,0x86,0x89,0x8C,0x8F,0x92,0x95,0x99,0x9C,0x9F,0xA2,0xA5,0xA8,0xAB,0

xAE,0XBl,0xB4,0xB6,0xB9,0XBC,0xBF,0xCl,0xC4,0xC7,0xC9,0xCC,0xCE,0xDl,0xD

3,0xD6,0xD8,0XDA,0xDC,0xDF,0xEl,0xE3,0xE5,0xE6,0xE8,0xEA,0xEC,0xED,0xEF,0

xFl,0xF2,0xF3,0xF5,0xF6,0xF7,0xF8,0xF9,0xFA,0xFB,0XFC,0xFC,0xFD,0xFE,0xFE,0

xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFE,0xFE,0xFE,0xFD,0xFC,0

xFC,0XFB,0xFA,0xF9,0xF8,0xF7, 0xF5, 0xF4, 0xF3, 0xF2, 0xF0, 0xEF, 0xED, 0xEB,

0xEA,0xE8, 0xE6, 0xE4, 0xE2, 0xE0, 0xDE ,0xDC,0xD9 ,0xD7

,0xD5,0xD2,0xD0,0xCE,0xCB,0xC8,0xC6,0xC3,0xCl,0xBE,0xBB,0xB8,0xB5,0xB3,0x
```

```
B0,0xAD,0xAA,0xA7,0xA4,0xAl,0x9E,0x9B,0x97,0x94,0x91,0x8E,0x8B,0x88,0x85,
0x82,0x7E,0x7B,0x78,0x75,0x72,0x6F,0x6C,0x69,0x65,0x62,0x5F,0x5c,0x59,0x56,0x5
3,0x50,0x4D,0x4B,0x48,0x45,0x42,0x3F,0x3D,0x3A,0x37,0x35,0x32,0x30,0x2D,0x2B,
0x29,0x26,0x24,0x22,0x20,0xlE,0xlC,0xlA,0xl8,0x16,0x14,0x13,0xll,0x0F,0x0E,0x0D,
0x0B,0x0A,0x09,0x08,0x06,0x05,0x05,0x04,0x03,0x02,0x02,0x0l,0x0l,0x0l,0x0l,0x0l,0
x0l,0x0l,0x0l,0x0l,0x0l,0x0l,0x0l,0x0l,0x0l,0x02,0x02,0x03,0x04,0x05,0x06,0x07,0xq8,
0x09,0x0A,0x0B,0x0D,0x0E,0x0F,0xll,0x13,0x14,0x16,0x18,0xlA,0xlC,0xlE,0x20,0x22
,0x24,0x26,0x29,0x2B,0x2D,0x30,0x32,0x35,0x37,0x3A,0x3D,0x3F,0x42,0x45,0x48,0x
4B,0x4D,0x50,0x53,0x56,0x59,0x5C,0x5F,0x62,0x65,0x69,0x6C,0x6F,0x72,0x75,0x78,
0x7B,0x7E;
} ;


char i=0;
interrupt [TIM1_COMPA] void timerl_compa_isr(void)
{
OCR1A=sinewave [i];
i++;
if (i= = 255)
i= 0;
void main (void){
DDRD=0xFF;
// Timer/Counter 1 initialization
// Clock source: System Clock
```

```
// Clock value: 8000.000 kHz
// Mode: Fast PWM top=00FFh
// OC1A output: Non-Inv
// oC1B output: Disco.
// Noise Canceler: Off
// Input Capture on Falling Edge
TCCR1A=0x81;
TCCR1B=0x09;
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0xl0;
// enable global interrups

#asm("sei") ;
while (1);
}
```

پورت سریال (RS-232)

پروتکل RS-232 در اواخر دهه ۶۰ میلادی به صورت استاندارد تعریف شد و همچنان یکی از استانداردهای پرکاربرد در کامپیوتر های شخصی و کاربردهای صنعتی است. این استاندارد هم ارتباط سریال سنکرون و هم آسنکرون را پشتیبانی کرده و به صورت Full Duplex عمل می نماید. کامپیوترهای شخصی تنها ارتباط آسنکرون را پشتیبانی می کنند و از طریق چیپ UART موجود در برد اصلی، اطلاعات را از حالت موازی به سریال یا از سریال به موازی تبدیل کرده و با تنظیمات زمانی آن را از طریق پورت سریال ارسال یا دریافت می کند.



پورت سریال دارای یک کانکتور ۹ پین می باشد و از آنجایی که این استاندارد در ابتدا برای ارتباط با مودم طراحی شـده بود، دارای پین های Handshaking و وضعیت می باشد. اما نوع خاصی از ارتباط با RS-232 به نام Null-Modem که تنها شامل پین های ارسال و دریافت است برای ارتباط با غیر از مودم استفاده می شود. بنابراین تنهـا دو پـین Rx و Tx (و البته زمین) مورد نیاز است. در شکل زیر کانکتور پورت سریال را که D9 نام دارد ملاحظه می کنید:

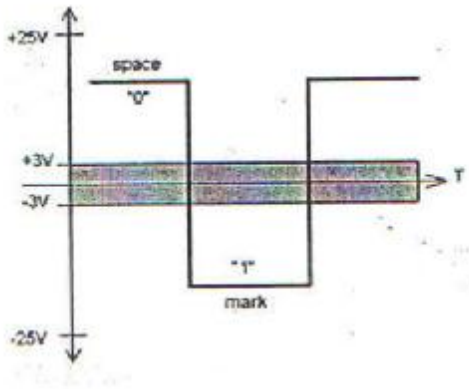
در جدول زیر عملکرد هر پین آورده شده است:

	Pin	**عملکرد**
۱	Carrier Detect	آیا مودم به یک خط تلفن متصل است؟
۲	Receive Data	کامپیوتر اطلاعات ارسال شده توسط مودم را دریافت می نماید.
۳	Transmit Data	کامپیوتر اطلاعاتی را برای مودم ارسال می دارد.
۴	Data Terminal Ready	کامپیوتر به مودم آمادگی خود را برای ارتباط اعلام می دارد.

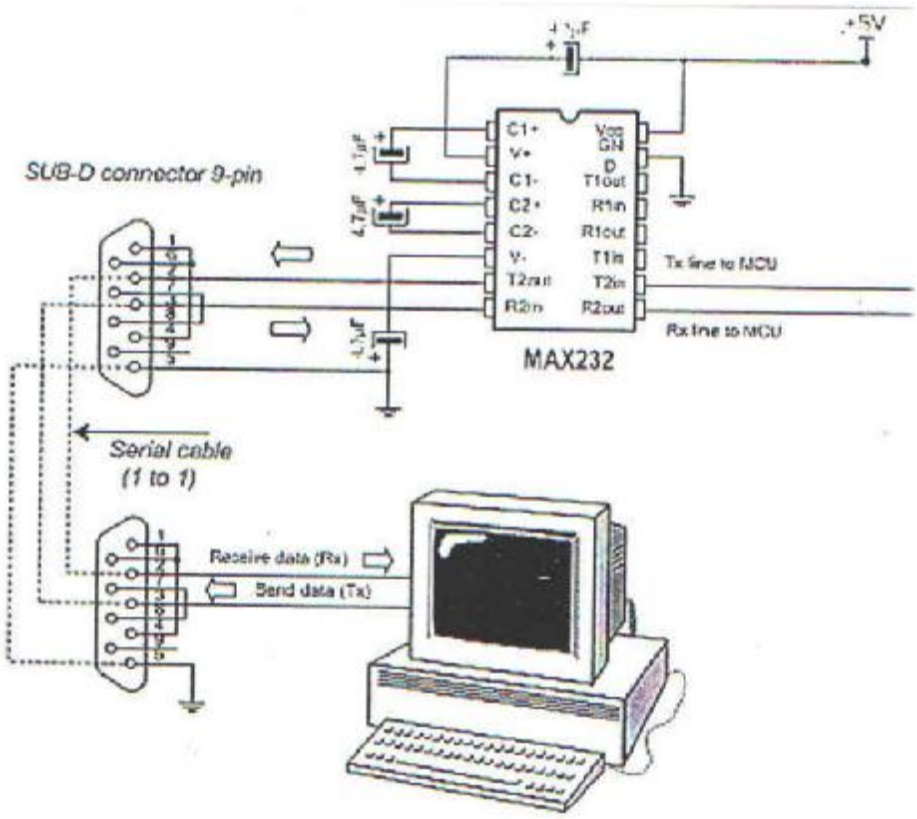
۵	Signal Ground	زمین سیگنال
۶	Data Set Ready	مودم آمادگی خود را برای ارتباط به کامپیوتر اعلام می دارد.
۷	Request To send	کامپیوتر از مودم در رابطه با ارسال اطلاعات سوال می نماید.
۸	Clear To send	مودم به کامپیوتر اعلام می نماید که می توان اطلاعاتی را ارسال دارد.
۹	Ring Indicator	زنگ تلفن تشخیص داده خواهد شد.

سطح سیگنال:

در استاندارد RS-232 سطح ولتاژ ۳+ تا ۱۲+ نمایانگر وضعیت Space یا صفر منطقی و بازه ی ۳- تا ۱۲- ولت نمایشگر وضعیت Mark یا یک منطقی می باشد.

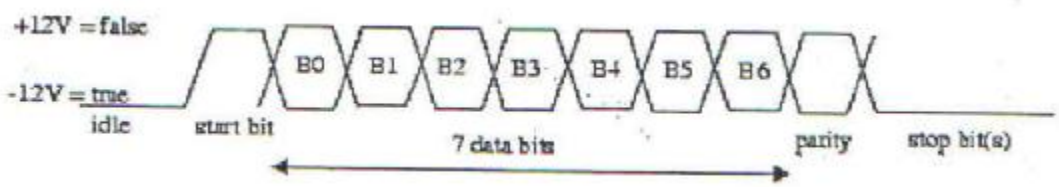


اگرچه تجهیزات استاندارد TTL با سطوح منطقی ۰ و ۵ ولت کار می کنند اما قالب اطلاعات ارسالی تفاوتی ندارد و با یک مدار تغییر سطح ولتاژ، PC می تواند با ادوت TTL ارتباط برقرار نماید. یکی از مبدل های متداول سطح RS-232 به TTL مدار مجتمع MAX232 و یا HIN232 می باشد. آی سی MAX232 یک تراشه ی ۱۶ پایه است که شامل ۲ فرستنده و ۲ مبدل مجزا است.



قالب اطلاعات:

در یک Frame اطلاعاتی که توسط بیت شروع و بیت پایان محصور شده است معمولاً ۵ تا ۸ بیت دیتا قرار می گیرد و یک بیت توازن نیز به صورت اختیاری تعریف می شود. بیت شروع متناظر با صفر منطقی است و بیت پایان (کـه ممکـن است ۱ یا ۲ بیت باشد). توسط یک شناسایی می شود. مثلاً در یک Frame که شـامل ۱۰ بیـت اسـت، هفـت بیـت آن شامل Data یک بیت آغازین و یک بیت پایانی و یک بیت توازن قبل از بیت پایان می باشد.



عملکرد USART میکروکنترلر AVR

قطعه ی ATmega16 دارای یک ماژول USART بوده که استاندارد RS-232 در دو حالت آسنکرون و سنکرون پشتیبانی می کند. دسترسی به پورت سریال AVR از طریق سه پین RXD, TXD و XCK که به ترتیب پین ارسال، دریافت و کلاک می باشند امکان پذیر است. (پین XCK فقط در Mode سنکرون کاربرد دارد).

قالب اطلاعات در USART میکروکنترلر AVR همانند UART کامپیوترهای شخصی شامل یک بیت شروع، بیت های داده، بیت اختیاری توازن و یک یا دو بیت پایان است، با این تفاوت که در AVR می تواند ۵ تا ۹ بیت Data تعریف شود. بیت توازن می تواند فرد یا زوج باشد و از طریق بیت های UPM[1:0] از رجیستر UCSRC تنظیم می شود.

رجیسترهای USART

۱- (UDR) USART Data Register

Bit	7	6	5	4	3	2	1	0
UDR(Read)	RXB[7:0]							
UDR(Write)	TXB[7:0]							

بافر دریافت و ارسال پورت سریال دارای یک آدرس مشترک به نام UDR در فضای I/O Registers می باشند. بافر ارسال، مقصد داده های نوشته شده در رجیستر UDR بوده و خواندن این رجیستر محتویات بافر دریافت را به دست می دهد. تنها زمانی می توان روی رجیستر UDR مقداری را نوشت که بیت UDRE از رجیستر UVSRA یک شده باشد و در غیر اینصورت دیتای ارسالی توسط USART نادیده گرفته می شود. با ارسال اطلاعات به بافر ارسال USART در صورتی که بیت TXEN از رجیستر UCSRB یک باشد اطلاعات در شیفت رجیستر بارگذاری شده و بیت به بیت از پین TXD ارسال می شود.

۲- USART Control and Register A

UCSRA	7	6	5	4	3	2	1	0
نام بیت	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM

Multi-processor Communication Mode: یک شدن این بیت میکروکنترلر را به حالت ارتباطات چند پردازنده ای می برد.

Double the USART Transmission Speed: با یک کردن این بیت در Mode آسنکرون Baud Rate دو برابر خواهد شد. در Mode سنکرون این بیت باید صفر باشد.

Parity Error: در صورت فعال بودن تولید بیت توازن از طریق بیت های UPM[1:0] با روی دادن خطای توازن در بافر دریافت این بیت یک می شود.

Data Over Run: با بروز Over Run این بیت یک می شود. شرایط Over Run یا لبریز وقتی روی می دهد که بافر دریافت پر باشد و شیفت رجیستر نیز محتوی داده ی جدیدی باشد و داده ی جدیدی نیز از راه برسد، یعنی یک بایت در بافر شیفت رجیستر، منتظر باشد و بیت شروع جدیدی دریافت شود. در این حالت اطلاعات جدید از بین می رود و با یک شدن بیت DOR بروز خطا اعلام می شود.

Frame Error: اگر در Frame دریافت شده بیت پایان صفر باشد این بیت یک شده و در غیر اینصورت صفر خواهد بود.

USART Data Register Empty: یک بودن این پرچم نشان دهنده ی این است که اطلاعات موجود در بافر ارسال برای شیفت رجیستر ارسال شده و بافر ارسال آماده ی دریافت کاراکتر جدید است. همچنین در صورتی که بیت UDRIE از رجیستر UCSRB یک باشد باعث ایجاد وقفه شده و تا زمانی که بیت UDRE یک ست با خارج شدن از ISR دوباره آن را اجرا می کند با نوشتن داده ی جدید در UDR پرچم صفر می شود. بعد از ریست شدن میکرو این بیت یک می شود که به معنای آماده بودن دریافت کاراکتر جدید است.

USART Transmit Complete: این پرچم زمانی یک می شود که تمام اطلاعات موجود در شیفت رجیستر به

بیرون شیفت داده شده و داده ی جدیدی در بافر ارسال وجود نداشته باشد. با فعال بودن بیت TXCIE این پرچم می تواند باعث ایجاد وقفه ی کامل شدن ارسال شود و با اجرای ISR بیت TXC توسط سخت افزار پاک شده و در غیر اینصورت نرم افزار می تواند با نوشتن یک بر روی آن، آن را پاک کند.

USART Receive Complete: این بیت با کامل شدن دریافت یک Frame در URD یک شده و پس از خواندن UDR صفر می شود.

۳- USART Control and Status register B

UCSRA	7	6	5	4	3	2	1	0
نام بیت	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8

Transmit Data Bit 8: در حالتی که از پورت سریال در Mode ۹ بیتی استاده می شود این بیت نهمین بیت کاراکتر ارسالی خواهد بود. باید توجه داشت که قبل از نوشتن در UDR باید وضعیت این بیت را مشخص کرد.

Receive Data Bit 8: در حالتی که از پورت سریال در Mode ۹ بیتی استفاده می شود این بیت نهمین بیت کاراکتر دریافتی خواهد بود. باید توجه داشت که قبل از خواندن UDR باید این بیت را خواند.

Character Size: با ترکیب این بیت و بیت های UCSZ[1:0] در رجیستر UCSRC تعداد بیت های داده (اندازه ی کاراکتر) را در یک Frame مشخص می کند.

Transmitter Enable: با یک کردن این بیت عملکرد عادی پین TxD به ارسال پورت سریال تغییر حالت داده و بعد از آن قابل استفاده به صورت I/O معمولی نیست. غیر فعال کردن این بیت در حالیکه UART سریال مشغول است تا اتمام ارسال اطلاعات تأثیر نخواهد گرفت.

Receiver Enable: با یک کردن این بیت عملکرد عادی پین RxD به دریافت پورت سریال تغییر حالت داده و بعد از آن قابل استفاده به صورت I/O معمولی نیست. با صفر کردن این بیت بافر پورت سریال خالی شده و مقدار بیت های DOR ، FE و PE نامعتبر خواهد بود.

USART Data Register Empty Interrupt Enable: با یک شدن این بیت، در صورتی که بیت فعال ساز کلی وقفه ها (I) یک باشد، فعال شدن پرچم خالی بودن بافر ارسال (UDRE) می تواند باعث ایجاد وقفه شود.

TX Complete Interrupt Enable: با یک شدن این بیت، در صورتی که بیت فعال ساز کلی وقفه ها (I) یک باشد، فعال شدن پرچم اتمام ارسال (TXC) می تواند باعث ایجاد وقفه شود.

RX Complete Interrupt Enable: با یک شدن این بیت، در صورتی که بیت فعال ساز کلی وقفه ها (I) یک باشد، فعال شدن پرچم اتمام ارسال (RXC) می تواند باعث ایجاد وقفه شود.

۴- USART Control and Status Register C

UCSRX	7	6	5	4	3	2	1	0
نام بیت	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL

رجیسترهای UCSRC و UBRRH دارای آدرس مشترکی در فضای I/O هستند، که برای انتخاب این دو از بیت URSEL استفاده می شود.

Clock Polarity: این بیت فقط در Mode سنکرون به کار برده می شود و در Mode آسنکرون باید روی آن صفر نوشته شود. عملکرد این بیت مطابق جدول زیر است:

UCPOl	تغییر داده ی ارسالی روی پین TxD	نمونه برداری از داده ی دریافتی روی پین RxD
0	لبه ی بالا رونده ی پالس XCK	لبه ی پایین رونده ی پالس XCK
1	لب ی پایین رونده ی پالس XCK	لبه ی بالا رونده ی پالس XCK

Character Size [1:0] : این بیت به همراه بیت UCSZ2 مطابق جدول زیر تعداد بیت های یک کاراکتر را تعیین می کنند، در حالت پیش فرض این بیت ها اندازه ی ۸ بیت را تعیین می کنند.

UCSZ2	UCSZ1	UCSZ0	اندازه ی کاراکتر
0	0	0	۵ بیت
0	0	1	۶ بیت
0	1	0	۷ بیت
0	1	1	۸ بیت

رزرو شده	0	0	1
رزرو شده	1	0	1
رزرو شده	0	1	1
حالت ۵ بیتی	1	1	1

Stop Bit Select: این بیت، تعداد بیت های پایان را معین می کند. در حالت پیش فرض که این بیت صفر می باشد یک بیت پایان و در صورت ۱ شدن ۲ بیت پایان در نظر گرفته می شود.

Parity Mode [1:0]: این دو بیت تنظیمات مربوط به بیت توازن را مطابق جدول زیر انجام می دهند. در صورت فعال بودن بیت توازن همراه هر Frame این بیت ایجاد شده و در گیرنده بر حسب همان تنظیمات (که باید در آنجا نیز مطابق تنظیمات فرستنده انجام شود). مقدار بیت توازن با مقدار دریافت شده مقایسه شده و در صورت بروز خطا پرچم PE از رجیستر UCSRA یک می شود.

وضعیت بیت توزان	UPM0	UPM1
غیر فعال	0	0
رزرو شده	1	0
توازن زوج	0	1
توازن فرد	1	1

USART Mode Select: UMSEL حالت کار UART را تعیین می کند. در صورتی این بیت یک باشد در حالت آسنکرون و با یک کردن آن در وضعیت سنکرون کار می کند.

Register Select: همانطور که گفته شد دو رجیستر UCSRC و UBRRH دارای آدرس مشترکی (0x40) هستند و این بیت برای انتخاب نوشتن روی یکی از این دو رجیستر می باشد. به این صورت که اگر بخواهیم روی UCSRC مقداری را بنویسیم باید در همان بایت MSB یک باشد و چنانچه بخواهیم روی UBRRH مقداری را بنویسیم باید MSB صفر باشد. به عنوان مثال دستورات زیر مقدار بیت ۱ از هر رجیستر را یک می کند:

```
UBRRH= 0x02;
UCSRC= 0x82;
```

روش خواندن این دو رجیستر به این صورت است که در صورت خواندن مقدار آدرس 0x40 و یا هر یک از اسامی مستعار این آدرس (UCSRC یا UBRRH) مقدار رجیستر UBRRH به دست خواهد آمد:

a= UBRRH

چنانچه بخواهیم مقدار رجیستر UCSRC را بدست آوریم باید در سیکل کلاک بعدی مقدار خوانده شده را به عنوان مقدار واقعی UCSRC در نظر بگیریم، به عنوان مثال:

b= UBRRH

b= UCSRC

بهتر است در صورت استفاده از وقفه، قبل از پروسه ی بالا بیت فعال ساز وقفه ها را با دستور (“cli”)#asm غیر فعال کنیم.

با خواندن UBRRH بیت URSEL صفر خوانده شده و با خواندن UCSRC این بیت یک خوانده می شود.

۵- USART Baud Rate Registers

Bit	7	6	5	4	3	2	1	0
UBRRH	URSEL	-	-	-	UBRR[11:8]			
UBRRL	UBRR[7:0]							

URSEL: همان طورکه گفته شد برای انتخاب بین UBRRH و UCSRC استفاده می شود.

UBRRH[7:4]: برای کاربردهای آینده رزرو شده هستند و برای سازگاری با قطعات آتی باید روی آن ها صفر نوشته شود.

UBRR[11:0]: این دوازده بیت برای تعیین Baud rate رابط USART مورد استفاده قرار می گیرند. به این منظور می توان از روابط زیر استفاده نمود:

Mode کاری	محاسبه Baud Rate از روی UBRR	محاسبه UBRR از روی Baud Rate
آسنکرون (U2X=0)	$Baud = \frac{f_{osc}}{16(UBRR+1)}$	$UBRR = \frac{f_{osc}}{16 \times Baud} - 1$

آسنکرون با سرعت دو برابر (U2X=1)	$Baud = \frac{f_{osc}}{8(UBRR+1)}$	$BaRR = \frac{f_{osc}}{8 \times Baud}$ -۱
عملکرد سنکرون	$Baud = \frac{f_{osc}}{2(UBRR+1)}$	$BaRR = \frac{f_{osc}}{2 \times Baud} - 1$

مقدار خطای بوجود آمده از رابطه ی زیر بدست می آید:

$$Error[\%] = \left(\frac{BaudRate_{ClosestMatch}}{BaudRate} - 1 \right) \times 100\%$$

مثال ۱: با کلاک ۸ مگاهرتز و BaudRate=2400 در Mode آسنکرون عادی، مقدار UBRR را بدست آورید.

$$BURR = \frac{8000000}{16 \times 2400} - 1 = 207.33 \Rightarrow UBRR = 207$$

$$BaudRate = \frac{8000000}{16(207+1)} = 2404$$

$$Error[\%] = \left(\frac{2404}{2400} - 1 \right) \times 100\% = 0.2\%$$

برای صفر شدن خطا می توان از کریستال 7.3728 مگاهرتز استفاده نمود اگرچه حداکثر خطای قابل قبول برای این وضعیت ۲ درصد می باشد. کریستال هایی که می توان با استفاده از آن ها به خطای Baud Rate صفر رسید ۱.۸۴۳۲، ۳.۶۸۶۴، ۷.۳۷۲۸، ۱۱.۰۵۹۲، ۱۴.۷۴۵۶ می باشد.

توابع USART در Code Vision

اعلان این توابع در فایل stdio.h قرار دارد و قبل از استفاده از آن ها باید تنظیمات اولیه USART انجام شود.

تابع ( ) getchar: این تابع به روش Polling منتظر می ماند تا پرچم RXC یک شده و بعد از آن مقدار UDR را می خواند.

تابع ( ) Putchar: این تابع به روش Polling منتظر می ماند تا پرچم UDRE یک شده و بعد از آن یک کاراکتر را در

UDR کپی می کند.

مثال ۲: (میکرو اول بعد از ۳ ثانیه عدد ۷ را برای میکرو دوم ارسال می کند و در مدت این سه ثانیه میکرو دوم بـه روش Polling منتظر دریافت یک کاراکتر می ماند و پس از دریافت آن را در PORTA می ریزد).

برنامه ی میکرو اول:

```
#include <mega16.h>
#include <delay.h>
#include <stdio.h>
#define xtal 8000000
void main(void)
{
UCSRA=0x00;
UCSRB=0x08; // USART Transmitter: On
UCSRC=0x86; //8 Data, 1 Stop, No Parity
UBRRH=0x00;
UBRRL=0x33; // USART Baud rate: 9600
delay_ms (3000);
putchar(7);
 while (1) ;
}
#include <mega16.h>
#include <stdio.h>
#define xtal 8000000
void main(void)
```

```
{
char a;
DDRA=0xFF;
PORTA=0xFF;
UCSRA=0x00;

UCSRB=0x10; // USART Receiver: On
UCSRC=0x86; //8 Data, 1 Stop, No Parity
UBRRH=0x00;
UBRRL=0x33; // USART Baud rate: 9600
a= getchar( ); // polling RXC
PORTA = a;
while (1);
}
```

توابع سطح بالای USART تنها برای صرفه جویی در وقت برنامه نویس می باشند. در صورت نیاز می توان به صورت مستقیم با رجیستر ها کار کرد. به عنوان مثال حلقه ی اصلی برنامه ی میکرو دوم را به صورت زیر نیز می توان نوشت.

```
while(! (UCSRA&0xB0)) ;
PORTA=UDR;
wh'i.Le (1) ;
```

توابع Puts ( ) , Putsf ( ) : این توابع یک رشته را توسط USART به پورت سریال ارسال می کنند. تابع Puts ( ) رشته ای را که در SRAM است و تابع Putsf ( ) رشته ای را که در Flash است به خروجی ارسال می کند. اساس تمام توابع سطح بالای USART توابع getchar ( ) , Putchar ( ) می باشند و در اینجا نیز به استفاده از اشاره گر

Y ، کاراکترها پشت سر هم بوسیله ی این توابع به رجیستر UDR ارسال می شوند.

این توابع به انتهای رشته کاراکتر LF(0x10) را اضافه می کنند.

مثال ۳: (رشته های "ariamec" و "AVR" به پورت سریال ارسال می شوند).

```
#include <mega16.h>
#include <delay.h>
#include <stdio.h>
#define xtal 8000000
flash char string1[7]="ariamec";
char string2[7]="AVR";
void main (void)
}
UCSRA=0x00;
UCSRB=0x08; // USART Transmitter: On
UCSRC=0x86; //8 Data, 1 Stop, No Parity
UBRRH=0x00;
UBRRL=0x33; // USART Baud rate: 9600
putsf(string1);
puts(string2);
while (l);
```

تابع ( )Gets: این تابع دارای دو آرگومان نام متغیر و طول است که منتظر می ماند تا کاراکترهای دریافتی به اندازه ی طول شده و سپس آن ها را داخل متغیر می ریزد.

مثال ۴: (میکرو اولی بعد از ۱ ثانیه عدد رشته ی ariamec را برای میکرو دوم ارسال می کند و در مدت این ثانیه میکرو دوم به روش polling منتظر دریافت رشته می ماند و پس از دریافت آن را در LCD نمایش می دهد).

برنامه میکرو ۱:

```
#include <mega16.h>
#include <delaY.h>
#include <stdio.h>
#define xtal 8000000
void main(void)
}
UCSRA=0x00;
UCSRB=0x08; // USART Transmitter: On
UCSRC=0x86; //8 Data, 1 Stop, No Parity
UBRRH=0x00;
UBRRL=0x33; // USART Baud rate: 9600
delaY_ms(1000);
putsf(" ariamec ");
while (1);
}
```

برنامه میکرو ۲:

```
#include <mega16.h>
#include <stdio.h>
#include <lcd.h>
#define xtal 8000000
```

```
#asm
.equ_ lcd_port=0x1B ;PORTA
#endasm
char a[10];
void main(void)
}
UCSRA=0x00;
UCSRB=0x10; // USART Receiver: On
UCSRC=0x86; //8 Data, 1 Stop, No Parity
UBRRH=0x00;
UBRRL=0x33; // USART Baud rate: 9600
lcd_init (16);
lcd_clear();
gets(a,10);
lcd_puts(a);
while(l);
}
```

تابع ( ) printf: این تابع یک رشته ی قالب بندی شده را به پورت سریال ارسال می کند. کاراکتر فرمت می تواند مطابق جدول زیر باشد:

کاراکتر فرمت	نوع اطلاعات
%c	یک کاراکتر
%d و %i	عدد صحیح علامتدار در مبنای ۱۰
%e و %E	عدد اعشاری به صورت نماد علمی

%f	عدد اعشاری
%s	رشته ی ختم شده به Null واقع درSRAM
%p	رشته ی ختم شده به Null واقع در Flash
%u	عدد صحیح بدون علامت در مبنای ۱۰
%x و. %X	عدد صحیح بدون علامت در مبنای ۱۶
%%	کاراکتر %

مثال ۵: (یک رشته ی قالب بندی شده را به پورت سریال ارسال می کند).

```
#include <mega16.h>
#include <delay.h>
#include <stdio.h>
#define xtal 8000000
int a = 100;
char b = ’ A' ;
float pi=3.14;
void main(void)
{
UCSRA=0x00;
UCSRB=0x08; // USART Transmitter: On
UCSRC=0x86; //8 Data, 1 Stop, No Parity
UBRRH=0x00;
UBRRL=0x33; // USART Baud Rate: 9600
        Printf("a=%d b=%c pi=%f",a,b,pi);
```

```
    while (1);
}
```

تابع ( ) printf این قابلیت را دارد که برنامه نویس طول میدان و دقت پارامتر را تعیین کند که این امکان را به صورت %width . precision قابل تنظیم می باشد. width نشان دهنده ی طول یک عدد است که در مورد اعداد صحیح اگر رشته از تعداد ارقام باشد به اندازه ی اضافه جای خالی در سمت چپ عدد در نظر گرفته می شود و چنانچه عدد اعشاری باشد. این مقدار بیانگر قسمت صحیح و " " و قسمت اعشاری می باشد. prescision مقدار دقت اعشار را تعیین می کند.

printf (" a= % 4.2", a);

مشخصات تابع ( ) printf در کامپایلر Code Vision را می توان از طریق:

Project/ Configure/ C Compiler/ (s) print Features

تعیین نمود.

تابع ( ) scanf : با استفاده از این تابع می تواند قالب اطلاعات دریافتی از پورت سریال را تعیین نمود. طرز کار این تابع به این صورت است:

scanf ("عبارت اول" ، عبارت دوم) ;

عبارت دوم نام متغیری است که رشته ی دریافتی در آن قرار می گیرد و عبارت اول تعیین می کند که قالب داده ی دریافتی به چه صورت باشد. به صورت پیش فرض طول رشته ی دریافتی ۲۵۶ کاراکتر است اما با افزودن طول قبل از کاراکتر فرمت می توان آن را به مقدار دلخواه کاهش داد. به عنوان مثال:

scanf ("%10d", a)

۱۰ کاراکتر را از ورودی خوانده و در متغیر a ذخیره می کند.

مشخصات تابع ( ) scanf را همانند ( ) printf می توان از:

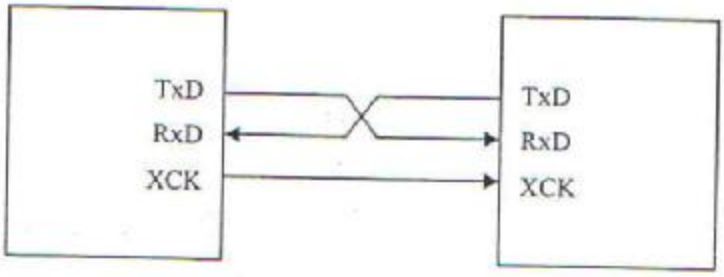
Project/ Configure/ C Compiler/ (s) pringt Features

تنظیم نمود.

توجه: توابع printf ( ) و scanf ( ) حجم زیادی از حافظه ی Flash را مصرف کرده و سرعت اجرای آن ها پایین است، بنابراین تا حد امکان از آن ها استفاده نکرده و در صورت لزوم مشخصات آن ها را به ساده ترین حالت (int, width, int) تنظیم نمایید.

استفاده از پورت سریال در وضعیت سنکرون

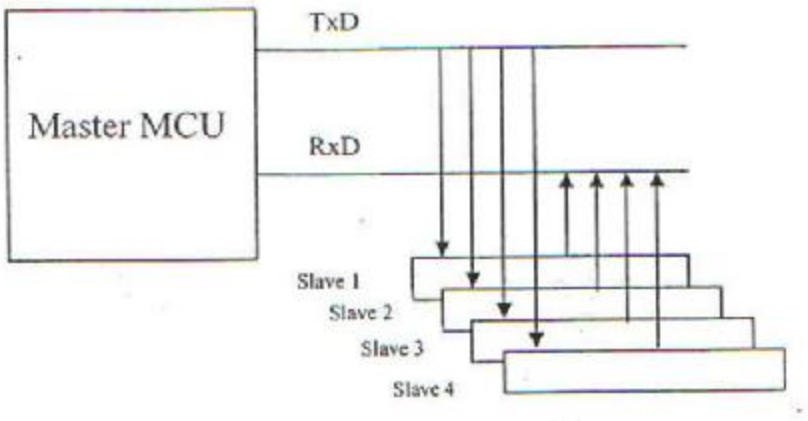
با یک کردن بیت UMSEL از رجیستر UCSRC میکرو کنترلر در این حالت قرار می گیرد و اتصالات سخت افزاری به صورت زیر می باشد:



همانطور که گفته شد بوسیله ی بیت های UCPOL می توان لبه ی نمونه برداری و تغییر بر ارسال کلاک را تعیین نمود. مسئله ای که باید در Mode سنکرون رعایت شود این است که داده باید در خلاف لبه ای که در فرستنده ارسال شده است، در گیرنده نمونه برداری شود یعنی اگر داده در لبه ی پایین رونده ارسال شده باشد در لبه ی بالا رونده دریافت می شود.

وضعیت Multi- processor

برای ایجاد شبکه بین میکرو کنترلرها از این Mode استفاده می شود. معمولاً یک میکروکنترلر به عنوان Master و بقیه به عنوان Slave ایفای نقش می کنند.



با یک کردن بیت MPCM از رجیستر UCSRA میکروکنترلر وارد این Mode خواهد شد. برای مبادله اطلاعات بین Master و Slave ، باید Slave توسط Master آدرس دهی شود که بدین منظور از بین نهم داده (TXB8) استفاده می شود. آدرس Slave ها قبلاً توسط برنامه نویس تعیین شده و تمام آن ها باید با یک شدن بیت MPCM در Mode چند پردازنده ای قرار بگیرند و منتظر دریافت ادرس از Master بماند. برای شروع Master یک فریم اطلاعاتی ۹ بیتی که هشت بیت آن شامل آدرس و بیت نهم آن یک است به میکروکنترلرها ارسال می کند. یک بودن بیت نهم باعث می شود که رفتار Slave ها با داده ی دریافتی همانند یک آدرس بوده و آن را با آدرس خود مقایسه کنند. Slave انتخاب شده بیت MPCM خود را پاک کرده و منتظر دریافت داده می ماند و بقیه Slave ها که بیت MPCM آن ها یک است همچنان منتظر دریافت آدرس می مانند. در این زمان Slave انتخاب شده با ارسال پیامی برای Master تصدیق می فرستند تا بیت نهم خود را صفر کند تا در ارسال بعدی به اشتباه Slave دیگری انتخاب نشود. پس از اتمام ارسال داده Master مجدداً به نشانه ی آدرس دهی بیت نهم خود را یک کرده و آدرس دیگری را ارسال می کند. Slave قبل که بیت MPCM آن صفر شده بود مجدداً این بیت را یک کرده و آماده ی دریافت آدرس می شود.

(TWI) I^2C Bus

استاندارد I^2C در اوایل دهه ۱۹۸۰ توسط شرکت Philips طراحی شد. در ابتدا این پروتکل به منظور ایجاد روشی ساده برای ایجاد ارتباط پردازنده با تراشه های جانبی در یک دستگاه تلویزیون ابتداع شد.

I^2C طبق تعریف شرکت فیلیپس مختلف Inter-IC می باشد که بیانگر هدف آن یعنی فراهم آوردن یک لینک ارتباطی بین مدارات مجتمع می باشد. امروزه این پروتکل به صورت عمومی در صنعت پذیرفته شده است وکاربرد آن از سطح تجهیزات صوتی و تصویری نیز فراتر رفته است. به گونه ای که امروزه در بیش از ۱۰۰۰ نوع IC مختلف به کار گرفته شده است. I^2C فضا را حفظ می کند و باعث کاهش چشمگیر هزینه ی نهایی می شود. دو خط ارتباطی به معنی Track های مسی کمتر و در نتیجه برد مدار چاپی کوچکتر و تست عیب یابی سریعتر و راحتتر می باشد. علاوه بر این در اغلب موارد حساسیت مدار الکترونیکی نسبت به تداخل امواج الکترومغناطیسی و تخلیه ی الکتروستاتیکی کاهش می یابد.

برخی از ویژگی های باس I^2C

۱- I^2C یک پروتکل سریال سنکرون می باشد و کلاک آن می تواند بدون از دست رفتن اطلاعات تغییر کند.

۲- آدرس دهی ۷ بیتی (۱۲۷ وسیله ی متفاوت بر روی باس) و نیز آدرس دهی ۱۰ بیتی در ویرایش جدید این استاندارد (۱۰۲۴ وسیله بر روی باس).

۳- باس تنها به دو خط SCL و SDA نیاز دارد. بر روی SCL توسط Master کلاک ایجاد می شود و SDA خط Data ی دو طرفه می باشد که جهت آن توسط Master کنترل می شود.

۴- بر روی باس اطلاعات ۸ بیتی به صورت دو جهته با نرخ ارسال حداکثر ۴۰۰ کیلوبیت بر ثانیه. (و ۳.۴ مگابیت بر ثانیه در وضعیت High Speed).

۵- توانایی درایو نمودن تا ۴۰۰ پیکوفاراد ظرفیت خازنی تا فاصله ی حداکثر ۴ متر.

۶- حذف Spike های ناخواسته از خط SDA با استفاده از فیلتر موجود در چیپ، به طوری که سیگنال های ورودی که عرض آن ها کمتر از ۵۰ نانوثانیه باشد، حذف می شوند.

۷- پین های SDA و SCL مجهز به کنترل کننده ی Slew Rate می باشند که کارایی باس I^2C را در فرکانس های بالا (نزدیک ۴۰۰ کیلوهرتز) بهبود می بخشد. عملکرد کنترل کننده به این صورت است که لبه های نیز سیگنال را تا حدودی صاف کرده و در واقع با حذف هارمونیک های بالا به کاهش EMI کمک می کند.

۸- عدم نیاز به طراحی مدار واسط و راه اندازی باس تنها با دو مقاومت زیرا که مدار کنترل باس به صورت مجتمع بر روی وسیله ی I^2C قرار می گیرد.

اصطلاحاتی در این استاندارد

Master: وسیله ای که شروع کننده ی ارسال اطلاعات و تولید کننده ی پالس کلاک و پایان دهنده ی ارسال اطلاعات می باشد.

Slave: وسیله ای است که توسط master آدرس دهی شده است.

فرستنده: وسیله ای که اطلاعات را در باس داده قرار می دهد.

گیرنده: وسیله ای که اطلاعات را از باس می خواند.

Arbitration: مکانیسمی که اطمینان می دهد که اگر بیش از یک Master همزمان بخواهند باس را به کنترل خود در آورند، تنها یکی از آن ها بدون از دست رفتن اطلاعات موفق شود.

دسترسی نرم افزاری به I^2C در Code Vision

کامپایلر Code Vision با ارائه یک سری توابع مربوط به باس I^2C امکان ایجاد این پروتکل را به صورت نرم افزاری به برنامه نویس می دهد. پین هایSDA و SCL باید توسط نرم افزار به صورت زیر تعیین شوند:

```
#asm
        equ _ i2c _ port=0x18
        .equ _  sda _ bit=3
        .equ _  scl _ bit=4
# endasm
```

در این قطعه کد پین های ۳ و ۴ از PORTB به عنوان SDA و SCL تعیین شده اند.

توابع I^2C در کامپایلر Code Vision

:i2c_init( )

این تابع تنظیمات اولیه ی باس I^2C را انجام داده و باید قبل از استفاده از توابع دیگر به کار برده شود.

:i2c_start ( )

این تابع یک شرایط آغاز ایجاد می کند و در صورتی که باس آزاد باشد مقدار یک را بر می گرداند و در غیر ایـن صـورت خروجی این تابع صفر خواهد بود.

:i2c_stop ( )

این تابع یک شرایط پایان بر روی باس I^2C ایجاد می کند.

:i2c _ read ( )

این تابع یک بایت را از باس I^2C خوانده و شکل کلی آن به صورت زیر می باشد:

unsigned char i2c_read (unsigned char ack)

پارامتر ack تعیین می کند که آیا پس از دریافت یک بایت acknowledgement ارسال شود یا خیر. در صـورتی کـه این پارامتر یک باشد ACK ارسال خواهد و در غیر این صورت با ایجاد نکرد ACK بـه صـورت پسـیو NACK ایجـاد خواهد شد.

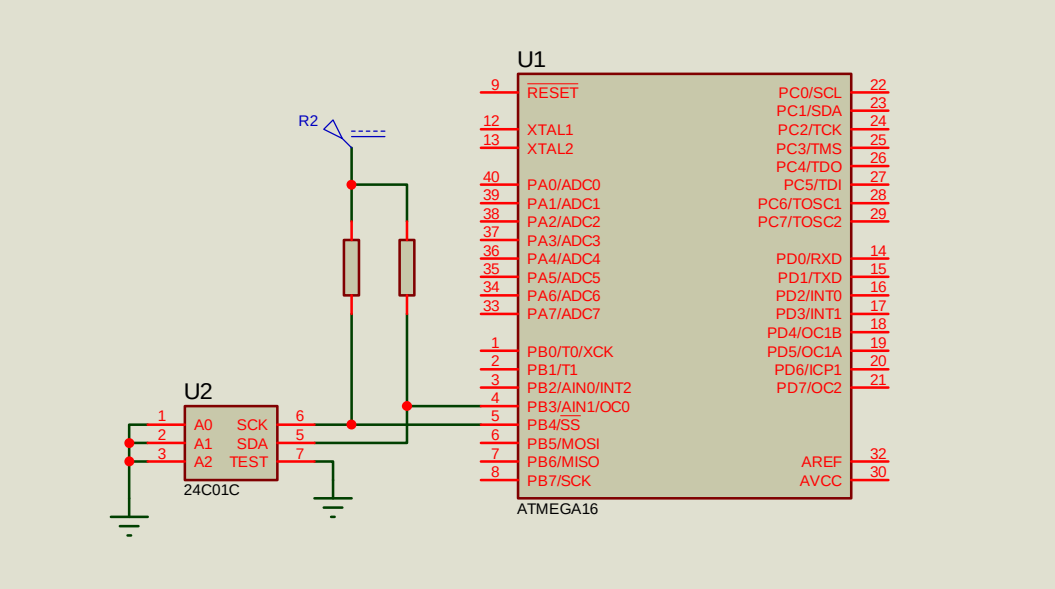
:i2c_write ( )

این تابع یک بایت را به باس I^2C ارسال کرده و شکل کلی آن به صورت زیر می باشد.

Unsigned char i2c_write (unsigned char data)

متغیر Data مقدار ارسالی به باس بوده و در صورتی کـه Slave، ACK ایجـاد کنـد ایـن تـابع مقـدار یـک و در غیـر اینصورت مقدار صفر باز می گرداند.

پروژه ۱۲: ارتباط با EEPROM های I^2C



نرم افزار:

```
# include < mega 16.h>

# define xtal 1000000

/* the 12C bus is connected to PORTB */

/* the SDA signal is bit 3 */

/* the SCL signal is bit 4 */

#asm

        .equ i2c_port=0x18

        .equ _sda_bit=3

        .equ_scl_bit=4

#endasm

/* now you can include the I2C Functions */

#include <i2c.h>
```

```
/* function declaration for delay_ms */
#include <delay.h>
#define EEPROM BUS ADDRESS 0xa0
/* read a byte from the EEPROM */
unsigned char eeprom_read(unsigned char address)
{
unsigned char data;
i2c_start();
i2c_write(EEPROM_BUS_ADDRESS) ;
i2c_write(address) ;
i2c_start();
i2c_write(EEPROM_BUS_ADDRESS I 1);
data=i2c_read(0);
i2c_stop () ;
return data;
}
/*write a byte to the EEPROM */
void eeprom_write(unsigned char address, unsigned char.data)
{
i2c_start();
i2c_write(EEPROM_BUS_ADDRSS) ;
i2c_write(address) ;
i2c_write(data);
```

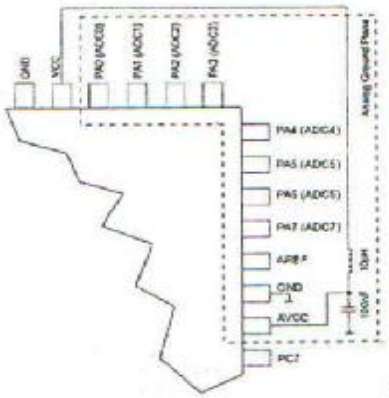
```
i2c_stop () ;
/*l0ms delay to complete the write operation */
delaY_ms(10);
}
void main(void) {
unsigned char i;
DDRD=0xFF;
/* initialize the I2C bus */
I2c_init() ;
/* write the byte 55h at address 10h */
eeprom_write(0x10,0x55);
/* read the byte from address AAh */
i=eeprom_read(0xl0);
PORTD=i;
while (1); /* loop forever */
}
```

مبدل آنالوگ به دیجیتال

عمده روش هایی که برای تبدیل آنالوگ به دیجیتال وجود دارند عبارتند از: تبدیل آنی یـا Flash ، روش تقریـب هـای متـوالی یـا Successive ApproximationT مبـدل هـای Sigma-Delta, Pipeline ADC, Ramp-Compare, Delta- Encoded و غیره که از این میان میکروکنترلرهای AVR از روش تقریب های متوالی اسـتفاده می کنند.

پین های ورودی ADC عملکرد دوم PORTA می باشند که به صورت مالتی پلکس شده به ADC اعمال می شـوند.

ولتاژ ورودی بین صفر تا ولتاژ مرجع بوده و ولتاژ مرجع از سه منبع AVCC، پین AREF و ولتاژ داخلی ۲,۵۶ ولت تأمین می باشد. جهت کاهش نویز موثر بر روی واحد ADC تغذیه ی آن به صورت جداگانه از پین AVCC تأمین می شود. ولتاژ این پین نباید بیشتر از 0,3 ولت با VCC تفاوت داشته باشد. در صورتی که از VCC به عنوان AVCC استفاده می شود، می توان بوسیله ی یک فیلتر LC این پایه به VCC متصل نمود.



رجیسترهای واحدی ADC

ADC Multiplexer Selection Register

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

Analog Channel and Gain Selection Bits [4:0]: این بیت ها تعیین می کنند که چه ترکیبی از ۸ کانال ورودی به واحد ADC متصل شده و همچین بهره ورودی تفاضلی را نیز تعیین می کنند. در حالت های Single – ended دقت ADC ۱۰ بیتی بوده که در حالت ورودی دیفرانسیل با بهره ی 1x و 10x این مقدار به ۸ بیت و با بهره ی 200x به ۷ بیت کاهش می یابد.

در صورتی که ADC مشغول انجام یک تبدیل بوده و این بیت ها تغییر کنند تا اتمام تبدیل جاری این تغییر انجام نخواهدشد تنظیمات این ۴ بیت مطابق جدول زیر می باشد: (عملکرد تفاضلی فقط بر روی Package های TQFP و MLF آزمایش شده است).

MUX4 .. 0	Single Ended Input	Poltlve Differential Input	Negative Differential Input	Gain
00000	ADCO	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	١0x
01001		ADCl	ADC0	١0x
01010(1)		ADC0	ADC0	200x
01011(1)		ADC1	ADC0	200x
01100		ADc2	ADC2	10x
01101		ADC3	ADC2	10x
01110(1)		ADC2	ADC2	200x
01111(1)		ADC3	ADC2	200x
10000		ADC0	ADCl	1x
10001		ADC1	ADCl	1x
10010		AOC2	ADCl	1x
10011		ADC3	ADCl	1x
10100		ADC4	ADCl	1x
10101		ADC5	ADCl	1x
10110		ADC6	ADCl	1x
10111		ADC7	ADCl	1lx
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
MUX4 .. 0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
11101		ADC5	ADC2	1x
11110	1.22 V (V_{BG})	N/A		
11111	0V (GND)			

ADC Left Adjust Result: بیت ADLR بر نحوه نمایش نتیجه ی تبدیل در رجیستر داده ی ADC تـأثیر مـی گذارد. نوشتن یک در این بیت آن را به صورت Left Adjust تنظیم می کند و در غیر ایـن صـورت نتیجـه بـه صـورت Right Adjust خواهد بود. تغییر این بیت به صورت آنی بر روی رجیستر داده تأثیر می گذارد.

Reference Selection Bits [1:0]: این بیت ها مرجع ولتـاژ ADC را مطـابق جـدول زیـر تعیـین مـی کننـد. در صورتی که این بیت ها در حین تبدیل تغییر کنند تا اتمام تبدیل تغییر اعمال نخواهد شد. در صورتی که از مرجع ولتـاژ داخلی استفاده می شود نباید ولتاژ خارجی به پین AREF اعمال شود. زمانی که یکی از دو ولتاژ AREF یا ۲.۵۶ ولت به عنوان مرجع انتخاب شده باشند با اتصال یک خازن ۱۰۰ نانو بین پین AREF و زمین می توان مقدارنویز را کـاهش داد.

REFS1	REFS0	ولتاژ مرجع
0	0	ولتاژ پایه ی AREF
0	1	ولتاژ پایه ی AVCC
1	0	رزرو شده
1	1	ولتاژ داخلی ۲.۵۶ ولت

ADC Control and Status Register A

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

ADC Prescaler Select Bits [2:0]: این بیت ها ضریب پیش تقسیم کننده ای را که از کلاک سیستم برای واحد

ADC کلاک تأمین می کند را مشخص می کند.

ضریب تقسیم	ADPS0	ADPS1	ADPS2
۲	۰	۰	۰
۲	۱	۰	۰
۴	۰	۱	۰
۸	۱	۱	۰
۱۶	۰	۰	۱
۳۲	۱	۰	۱
۶۴	۰	۱	۱
۱۲۸	۱	۱	۱

ADC Interrupt Enable: در صورت یک بودن بیت فعال ساز عمومی وقفه (I) و یک بودن این بیت، اتمام یک تبدیل می تواند باعث ایجاد وقفه شود.

ADC Interrupt Flag: با اتمام یک تبدیل این پرچم یک شده و در صورت فعال بودن وقفه، اجرای ISR می تواند باعث پاک شدن آن شود و در غیر اینصورت با نوشتن یک در محل این بیت می توان آن را پاک نمود.

ADC Auto Trigger Enable: عملیات تبدیل به دو صورت می تواند راه اندازی شود، Auto Trigger, Single که حالت اول با هر بار راه اندازی ADC یک تبدیل انجام شده و در وضعیت دوم ADC به صورت خودکار از طریق یکی از منابع داخلی تحریک می شود. برای قرار دادن ADC در وضعیت Auto Trigger باید این بیت یک شود. نوع منبع تریگر کننده بوسیله ی بیت های ADTS[2:0] از رجیستر SFIOR انتخاب می شود.

ADC Start Conversion: در وضعیت راه اندازی Single، برای آغاز هر تبدیل باید این بیت یک شود و در وضعیت تبدیل پیوسته (Free Running) نوشتن یک روی این بیت اولین تبدیل را موجب می شود.

ADC Enable: این بیت فعال ساز ماژول ADC بوده و با یک کردن آن می توان ADC را فعال نمود. نوشتن صفر

روی این بیت در حالی که ADC مشغول تبدیل است باعث می شود که عملیات تبدیل نیمه کاره رها شود.

The ADC Data Register

با پایان عملیات تبدیل نتیجه در این رجیستر قرار می گیرد و در صورتی که ورودی ADC به صورت دیفرانسیل باشد نتیجه به فرم مکمل ۲ نمایش داده می شود.

مطابق بیت ADLR در رجیستر ADMUX به دو صورت I,A و RA نمایش داده می شود.

ADLR=0

Bit	7	6	5	4	3	2	1	0
ADCL	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0
ADCH	-	-	-	-	-	-	ADC9	ADC8

Bit	7	6	5	4	3	2	1	0
ADCL	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2
ADCH	ADC1	ADC0	-	-	-	-	-	-

Special Function IO Register

SFIOR	7	6	5	4	3	2	1	0
نام بیت	ADTS2	ADTS1	ADTS0	-	ACME	PUD	PSR2	PSR10

ADC Auto Trigger Source: در صورتی که بیت ADATE از رجیستر ADCSRA مقدار یک داشته باشد بیت های ADTS تعیین می کنند که کدام منبع به صورت خودکار ADC را راه اندازی کند. منبع این راه اندازی لبه ی بالا رونده ی پرچم وقفه ی آن منبع می باشد و در صورتی که بخواهیم اتمام تبدیل خود ADC منبع تریگر بعدی باشد و ADC به صورت پیوسته عملیت تبدیل را انجام دهد از حالت Free Running استفاده می کنیم.

منبع راه اندازی	ADTS0	ADTS1	ADTS2
Free Running	۰	۰	۰
مقایسه کننده ی آنالوگ	۱	۰	۰
وقفه ی خارجی صفر	۰	۱	۰
تطبیق مقایسه ی تایمر صفر	۱	۱	۰
سرریز تایمر صفر	۰	۰	۱
تطبیق مقایسه ی B تایمر یک	۱	۰	۱
سرریز تایمر یک	۰	۱	۱
Capture تایمر یک	۱	۱	۱

راه اندازی ADC به صورت تک تبدیل و تبدیل خودکار

پس از انتخاب کانال مورد نظر و ولتاژ مرجع بوسیله رجیستر ADMUX ، با در نظر گرفتن اینکه بیت ADEN یک بوده باشد، نوشتن یک منطقی بر روی بیت ADSC شروع یک تبدیل را موجب خواهد شد. این بیت در حین انجام تبدیل یک بوده و با پایان ان بوسیله ی سخت افزار پاک می شود. در صورتی که قبل از اتمام تبدیل، کانال ADC تغییر کند تا پایان تبدیل جاری این تغییر تأثیر نخواهد گذاشت.

راه دیگر برای راه اندازی ADC وضعیت تحریک خودکار می باشد که این حالت با یک کردن بیت ADATE از رجیستر ADCSRA آغاز می شود. منبع تریگر بوسیله ی بیت های ADTS در رجیستر SFIOR انتخاب می شوند. زمانی که پرچم منبع تریگر یک می شود، پیش تقسیم کننده Reset شده و ADC شروع به انجام یک تبدیل می کند، بدین وسیله می توان در باز های زمانی ثابت ADC را تریگر نمود.

پرچم اتمام یک تبدیل بیت ADIF می باشد، در صورتی که ADC در وضعیت تحریک خودکار قرار گرفته و این بیت

به عنوان منبع تریگر انتخاب شود، پس از اتمام یک تبدیل، تبدیل جدیدی شروع خواهد شد. برای رسیدن به این باید وضعیت بیت های ADTS در حالت Free Running قرار گیرند و برای شروع تبدیل تنها با یک بار یک کردن ADSC آغاز شده و پس از آن تبدیلات متوالی انجام خواهد شد.

همانطور که گفته شد در صورت فعال بودن بیت ADIE بالا رفتن پرچم اتمام تبدیل (ADIF) می تواند باعث ایجاد وقفه شده و با اجرای ISR این بیت توسط سخت افزار پاک شود. در صورتی که ADC در وضعیت Single Conversion یا تک تبدیل باشد برای شروع تبدیل بعدی باید پرچم ADIF با نوشتن یک بر روی آن پاک شود، در حالیکه وضعیت Free Running انتخاب شده باشد بدون در نظر گرفتن اینکه پرچم ADIF پاک شده است یا نه تبدیلات متوالی انجام خواهد شد.

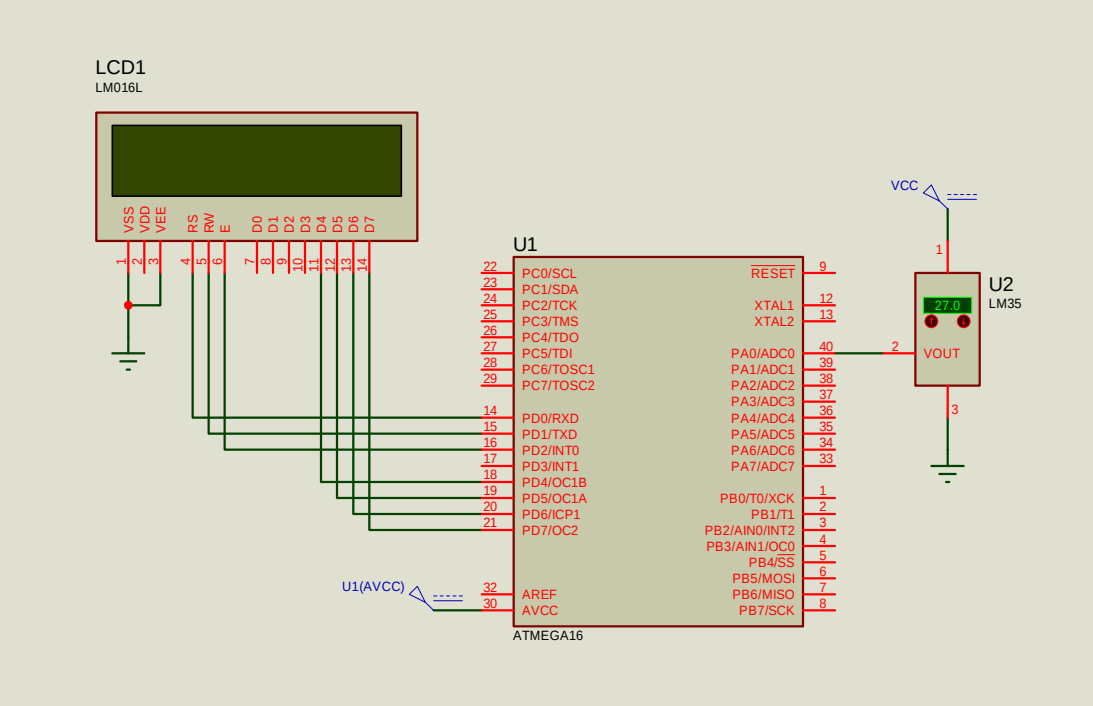
پیش تقسیم کننده و زمان بندی تبدیل

به صورت پیش فرض، مداری که بر اساس تقریب های متوالی تبدیل آنالوگ به دیجیتال را انجام می دهد برای رسیدن به ماکزیمم Resolution، نیاز به یک کلاک ورودی با فرکانسی بین ۵۰ تا ۲۰۰ کیلوهرتز دارد. ماژول ADC برای تأمین کلاک مورد نیاز دارای یک پیش تقسیم کننده می باشد که مقدار آن بوسیله ی بیت های ADPS از رجیستر ADCSRA تعیین می شود. واحد ADC برای عملکرد صحیح به فرکانس کلاکی بین ۵۰ تا ۲۰۰ کیلوهرتز نیاز دارد و در صورتی که مقدار آن خارج از این محدوده تعریف شود ممکن است عملکرد صحیحی نداشته باشد. در صورت استفاده از Mode تک تبدیل به دلیل تنظیمات اولیه ی ADC هر تبدیل حدود ۲۵ سیکل کلاک طول می کشد، در حالیکه در وضعیت Free Running هر تبدیل برای کامل شدن حدود ۱۳ کلاک زمان نیاز دارد.

فیلتر کاهش نویز ورودی

مدارات داخلی میکروکنترلر با ایجاد نویز باعث کاهش دقت مقدار خوانده شده توسط ADC می شوند، برای بهبود این مشکل می توان در زمان تبدیل میکروکنترلر را به یکی از Mode های کم توان ADC Noise Reduction یا Idle برد تا عملیات تبدیل بعد از خاموش شدن CPU انجام شود.

پروژه ۱۳: اندازه گیری دما با سنسور LM 35



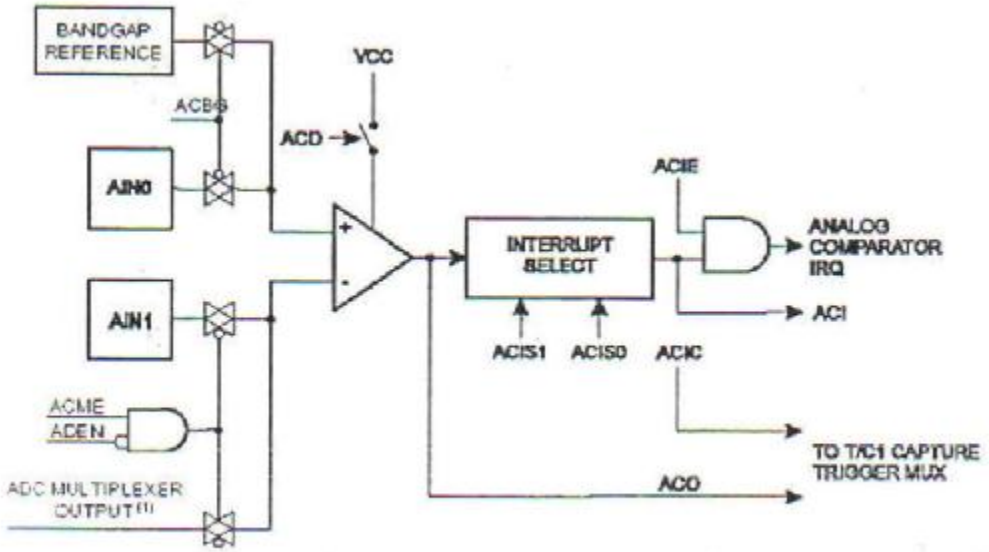

```
/*****************************************************

Project: Temprature Measurement with LM35

Chip type : ATmega16

Clock frequency : 1.000000 MHz

*****************************************************/

#include <mega16.h>

#include <delay.h>

#include <stdio.h>

#define xtal 1000000

// Alphanumeric LCD Module functions

#asm

        .equ __ lcd_port=0x12 ; //PORTD
```

```
#endasm
#include <lcd.h>
#define ADCVREFTYPE 0xC0
// Read the-AD conversion result
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc input I ADC_VREF_TYPE;
//Start the AD conversion
ADCSRA I =0x4 0;
//Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRAI=0x10;
return ADCW;
}
void main (void)
}
char lcd_buff(10);
int adc_in;
float temp;
PORTA=0x00;
DDRA=0x00;
// ADC initialization
// ADC Clock £requency: 15.625 kHz
```

```
// ADC Voltage Reference: Int 0, cap. on AREF
// ADC Auto Trigger Source: None
ADMUX=ADC_VREF_TYPEi
ADCSRA=0x86;
// LCD module initialization
lcd_init (16);
while (1)
{
        adc in=read_adc(0);
        temp=adc_in/4;
        sprintf(lcd_buff, "Temp=%5.1f C",temp);
        lcd_clear ( );
        lcd_gotoxy (0,0);
        lcd_puts (lcd_buff);
        delay_ms (1000);
        };
}
```

مقایسه کننده ی آنالوگ

ماژول مقاسیه کننده ی آنالوگ دارای دو ورودی AIN0 و AIN1 می باشد که این دو به ترتیب ورودی مثبت و منفـی واحد مقایسه کننده بود که همانند یک تقویت کننده ی عملیاتی ولتاژ روی این دو پایه را مقایسه کـرده و هنگـامی کـه ولتاژ روی پایه ی AIN0 بیشتر از ولتاژ AIN1 باشد خروجی آن یعنی بیت ACO یک می شود. این خروجی علاوه بر کاربردهای عادی می تواند برای تریگر کردن ورودی Capture تایمر یک نیز به کار رود.



رجیسترهای مقایسه کننده ی آنالوگ

Special Function IO Register

SFIOR	7	6	5	4	3	2	1	0
نام بیت	ADTS2	ADTS1	ADTS0	-	ACME	PUD	PSR2	PSR10

Analog Comparator Multuplexer Enable: ماژول مقایسه کننده ی آنالوگ این امکان را می دهد تا ورودی منفی از طریق پایه های ADC0 تا ADC7 انتخاب شود. در صورت یک بودن بیت ACME و خاموش بودن ADC (بیت ADEN در ADCSRA صفر باشد). خروجی مالتی پلکسر ADC به عنوان ورودی منفی مقایسه کننده انتخاب می شود و در غیر اینصورت پایه AIN1 ورودی منفی خواهد بود.

Analog Comparator Control and Status Register

ACSR	7	6	5	4	3	2	1	0
نام بیت	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0

وضعیت وقفه	ACIS0	ACIS1
وقفه ی مقایسه کننده در Toggle خروجی	۰	۰
رزرو شده	۱	۰
وقفه ی مقایسه کننده در لبه ی پایین رونده ی خروجی	۰	۱
وقفه ی مقایسه کننده در لبه ی بالا رونده ی خروجی	۱	۱

Analog Comparator Input Capture: با یک شدن ایـن بیـت ورودی Captue تـایمر یـک از طریـق خروجـی مقایسه کننده تریگر می شود.

Analog Camprator Interrupt Enable: در صورتی که این بیت و بیت فعال ساز عمومی وقفه ها یک باشد وقفـه ی مقایسه کننده ی آنالوگ فعال خواهد بود.

Analog Comparator Flag: این پرچم زمانی که یکی از رویدادهای تعریف شده بوسیله ی بیـت هـای ACIS1 و ACIS0 روی می دهد، بوسیله ی سخت افزار یک می شود. در این وضعیت اگر ACIE و فعال سـاز عمـومی وقفـه هـا یک باشد برنامه به ISR مقایسه کننده منشعب خواهد شد و بیت ACI توسط سخت افـزار پـاک خواهـد شـد، در غیـر اینصورت نرم افزار می تواند با نوشتن یک آن را پاک کند.

Analog Comparator Bandgap Select: با یک شدن این بیت ولتاژ Bandgap جایگزین ورودی مثبت مقایسه کننده خوماهد شد. در صورت صفر بودن بیت ACGB پین AIN0 ورودی مثبت مقایسه کننده خواهد بود.

Analog Comparator Disable: با یک شدن این بیت تغذیه ی مقایسه کننده ی آنالوگ قطع می شود. این مسئله به کاهش توان مصرفی در Mode های فعال و بیکاری کمک خواهد کرد. قبل از تغییر دادن بیت ACD بایـد وقفـه ی مقایسه کننده ی آنالوگ با پاک کردن بیت ACIE از ACSR غیر فعال شود در غیر اینصورت در زمان تغییر این بیـت

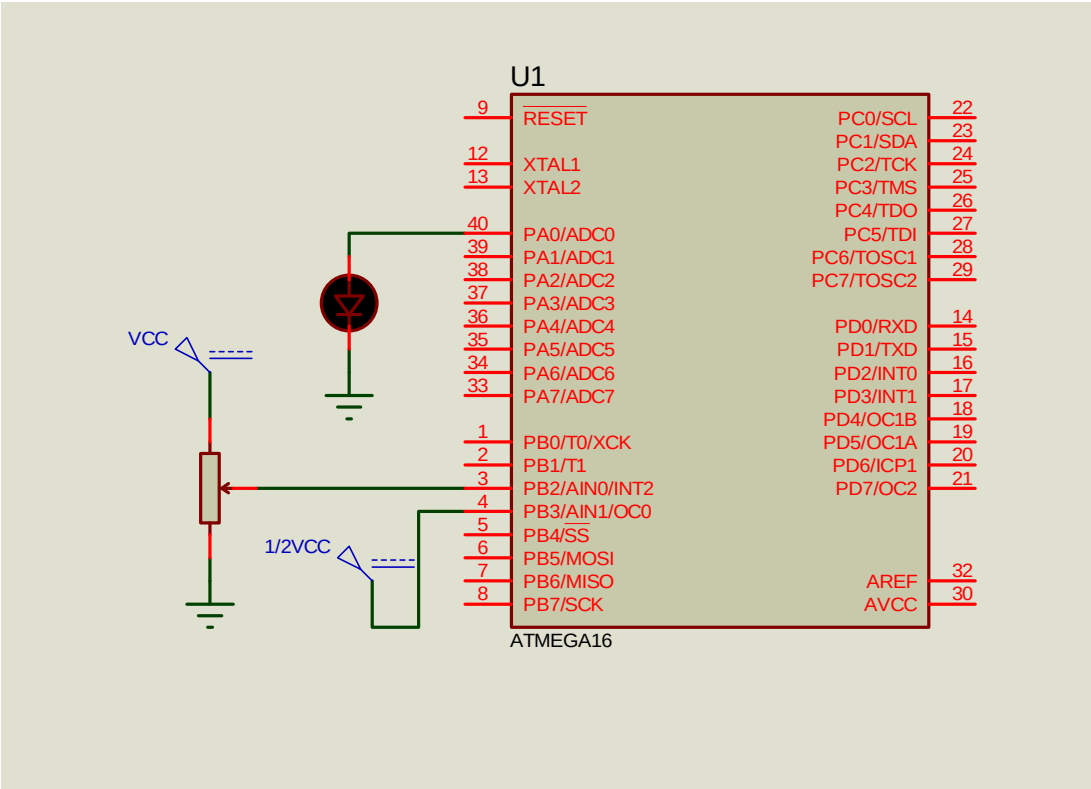
می تواند وقفه بوجود آید.

ورودی مالتی پلکس شده ی مقایسه کننده ی آنالوگ

این امکان وجود دارد که هر یک از ورودی های ADC0 تا ADC7 به عنوان ورودی منفی مقایسه کننده آنالوگ انتخاب شوند. برای استفاده از این مسئله باید (با صفر بودن بیت ADEN) مبدل آنالوگ به دیجیتال خاموش بوده و بیت ACME از SFIOR یک باشد در این وضعیت بیت های MUX[2:0] از رجیستر ADMUX ورودی منفی را مطابق جدول زیر انتخاب می کنند. مسلماً در صورتی که بیت ACME صفر بوده یا ADEN یک باشد پین AIN1 ورودی منفی مقایسه کننده ی آنالوگ خواهد بود.

ACME	ADEN	MUX[2:0]	ورودی منفی مقایسه کننده
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7

مثال: (اعلام عبور خروجی یک سنسور آنالوگ از یک سطح معین)




```
#include <mega16.h>

// Analog Comparator interrupt service routine

interrupt [ANA_COMP] void ana_comp_isr(voidl

{

PORTA=PORTAA0x0l;

}

// Declare your global variables here

void main(void)

{

// Declare your local variables here

PORTA=0x00;
```

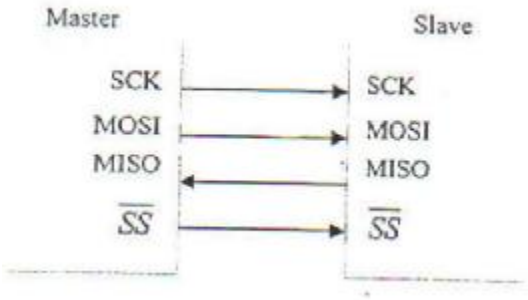
```
DDRA=0x0l;
// Analog Comparator initialization
// Analog Comparator: On
// Interrupt on Output Toggle
// Analog Comparator Input Capture by Timer Counter 1: Off
ACSR=0x08;
SFIOR=0x00 ;
// Global enable interrupts
#asm{ "sei ")
While (1);
}
```

SPI Bus

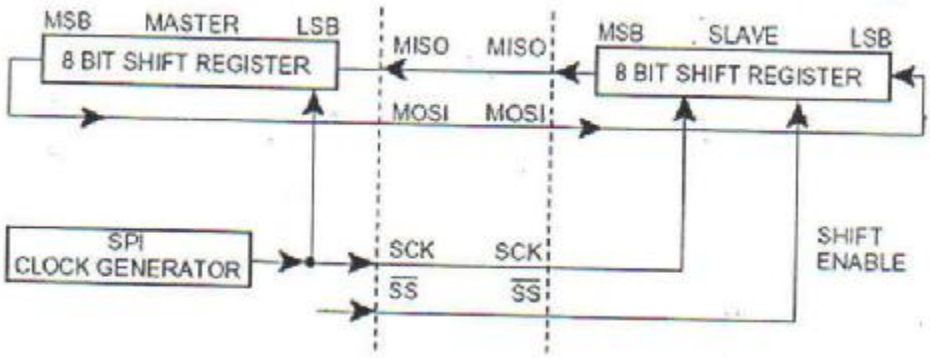
SPI که یک استاندارد سریال سنکرون می باشد سرنام Serial Peripheral Interface بوده و بوسیله شرکت موتورولا طراحی شده است. این استاندارد به لحاظ پشتیبانی از سرعت های بالا نه تنها در کاربردهای اندازه گیری بلکه در مواردی نظیر انتقال حجم بالای اطلاعات، پردازش سیگنال دیجیتال، کانال های ارتباطی و ... نیز مورد استفاده واقع می شود. سرعت چند مگابیت بر ثانیه را به راحتی توسط SPI قابل دسترسی است و در نتیجه امکان انتقال صوت فشرده نشده وتصویر فشرده شده وجود خواهد داشت.

نحوه عملکرد SPI

این استاندارد برای ایجاد یک ارتباط به چهار خط ارتباطی نیاز دارد:



همانطورکه ملاحظه می شود این چهار خط SS, MISO, MOSI, SCK بوده که به ترتیب خط کلاک، Master out slave in، Master in slave out و Slave select می باشند. نحوه ی تعامل Master و Slave در شکل زیر نشان داده شده است:

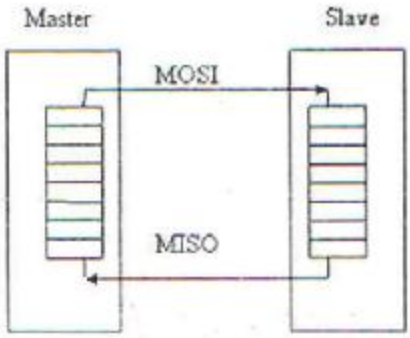


سیستم شامل دو Shift Register و یک مولد کلاک می باشد. Master با صفر کردن خط SS از Slave مورد نظر، چرخه ی ارتباطی را آماده می کند. Master و Slave داده ی مورد نظر برای ارسال را در Shift Register قرار داده و Master با ایجاد کلاک در خط SCK مبادله ی داده را آغاز می کند. اطلاعات از پین MOSI در Master خارج شده و وارد پین MOSI از Slave می شود. در طرف Slave نیز داده از پین MISO خارج شده و وارد MISO از Master می شود. بعد از اتمام ارسال یک Packet، مجدداً خط SS توسط Master یک شده و بدین ترتیب Slave با Master سنکرون می شود.

نحوه ی انتقال داده در SPI

وقتی ماژول SPI به عنوان Master پیکر بندی شده است خط SS را به صورت خودکار کنترل نمی کند و این وظیفه

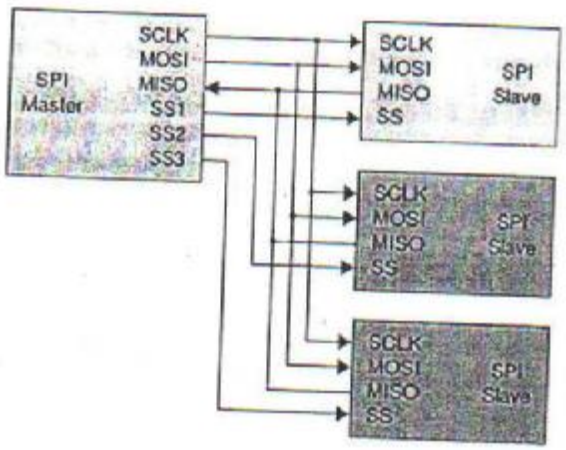
باید توسط نرم افزار، قبل از آغاز یک چرخه ی ارتباطی انجام شود. پس از صفر کردن SS ، نوشتن یک بایت در رجیستر داده (SPDR) باعث ایجاد کلاک توسط واحد تولید کلاک خواهد شد و با هر پالس داده ی موجود در Shift Register های Master و Slave یک بیت شیفت داده شده و پس از ۸ پالس پرچم SPIF به نشانه ی اتمام ارسال یک می شود. پس از این Master می تواند برای ارسال بایت بعدی آن را در SPDR نوشته و یا به نشانه ی اتمام ارسال خط SS را یک نگاه دارد.



در نقطه ی مقابل زمانیکه ماژول SPI در نقش Slave پیکربندی شده است، مادامیکه خط SS یک است پین MISO در وضعیت tri- stated می باشد. در این شرایط ممکن است که رجیستر SPDR بروز شود اما تا زمانیکه خط SS توسط Master صفر نشود انتقال انجام نخواهد شد. پس از Low شدن SS و اتمام دریافت یک بایت پرچم SPIF یک شده و در صورت یک بودن SPIE و بیت I، این رویداد می تواند باعث ایجاد وقفه شود. پس از این ممکن است Slave داده ی جدیدی را در SPDR قرار دهد منتها باید قبل از آن داده ی دریافتی را بخواند.

ارتباط شبکه ای با SPI

مطابق تصویر زیر با استفاده از پین SS می توان تعدادی Slave را کنترل نمود. Master باید تنها پین Slave SS را که می خواهد با آن ارتباط برقرار کند صفر کند و بقیه را یک نگه دارد.



تابع spi( ) در Code Vision

اعلان این تابع در فایل spi.h بوده و نحوه ی عملکرد آن در فایل spi.lib به صورت زیر می باشد:

```
unsigned char spi(unsigned char data)
{
SPDR=data;
while ((SPSR & (l«SPIF))==0);
return SPDR;
}
```

بنابراین آنجه این تابع عنوان آرگومان دریافت می کند روی باس SPI قرار می دهد و مقدار بازگشتی آن مقـدار خوانـده شده از باس است. قبل از استفاده این تابع باید SPI بوسیله ی رجیسترهای کنترل و وضعیت تنظیم شده باشد.

مثال: (ارتباط دو میکروکنترلر از طریق SPI)

برنامه میکرو Master:

```
#include <mega16.h>
#include <delay.h>
// SPI functions
```

```
#include <spi.h>

void main(void)

}

unsigned char incoming;

//SCK=Out MISO=In MOSI=Out SS=Out

PORTB=0b00000000;

DDRB=0bl0ll0000;

// 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

// SPI Enable: True

SPCR=0x71;

while(l)

}
```

```
        incoming=spi(0x77);
        delay_ms(50);
}
}
```

برنامه میکرو Slave

```
#include <mega16.h>
#include <delay.h>
// SPI functions
#include <spi.h>
void main (void)
{
unsigned char incoming;
// SCK=In MISO=Out MOSI=In SS=In
DDRB=0b0l000000;
// SPI initialization
```

```
// SPI Type: Slave

// SPI Clock Rate: 500.000 kHz

// SPI Clock Phase: Cycle Half

// SPI Clock Polarity: Low

// SPI Data Order: MSB First

// SPI Enable: True

SPCR=0x61;

while(l)

{

        incoming=spi(0x33);

        delay ms (50);

}

}
```

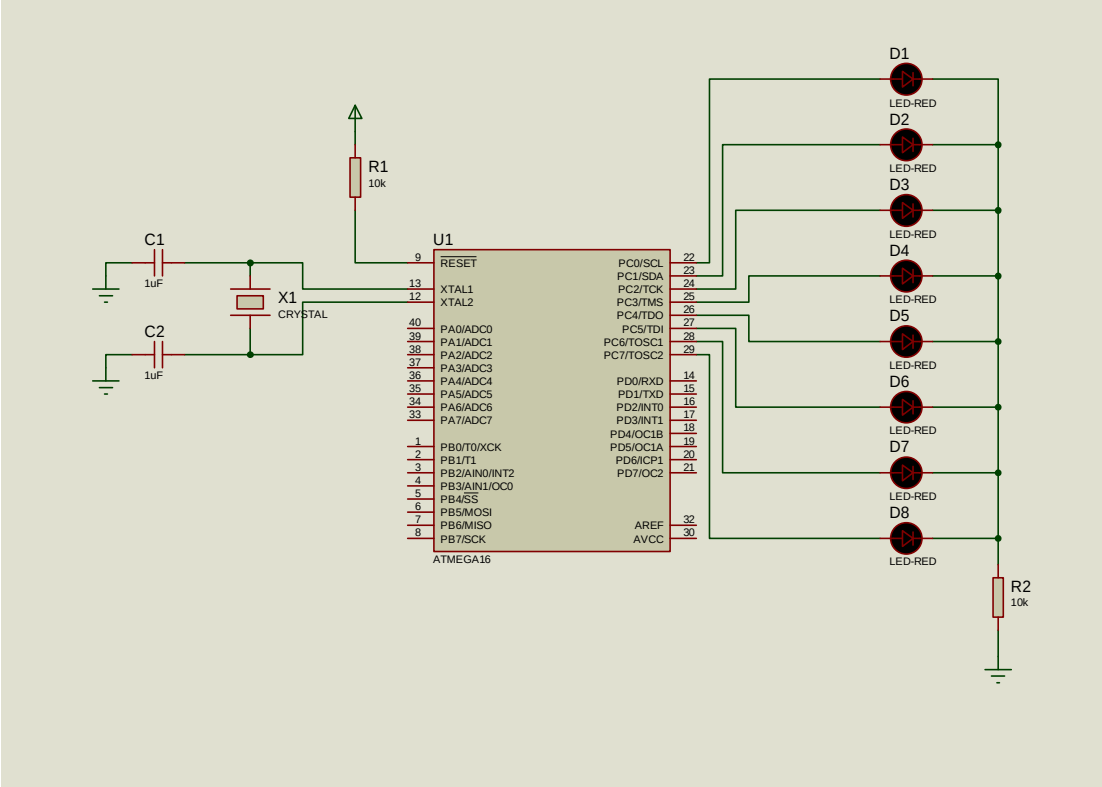
بخش دوم

پروژه های تکمیلی

LED

LED ها یکی از پر کاربردترین المانها برای نمایش اطلاعات می باشند که از جمله کاربرد آنها را می توان در تابلو روانها و ... بیان کرد.

در طراحی این ماژول LEDها به صورت کاتد مشترک می باشد لذا برای روشن کردن آنها باید آند آنها را یک کرد(آند LEDها به میکروکنترلر و کاتد آنها توسط یک مقاومت مشترک به GND متصل می باشد).



مثال ۱ :

برنامه ای بنویسید که چهار LED اول و چهار LED دوم را به ترتیب به مدت 200 ms روشن و خاموش نماید.

جهت اجرای این برنامه باید ماژول LED را توسط کابل IDC به پورت C، Main Bord متصل نمایید. برای پروگرم کردن میکرو کنترلر با USB پس از ساخت کد Hex در نرم افزارCode Vision و اتصال Main Bord به کامپیوتر، کد Hex مربوطه را در برنامه ی Progisp، Load کرده و بر روی گزینه های Erase و Auto کلیک می نمایید.

```
#include <mega16.h>
#include <delay.h>
void main(void)
{
PORTA=0x00;
DDRA=0x00;
```

```
PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0xff;

PORTD=0x00;
DDRD=0x00;

while (1)
    {
    PORTC.0=1;
    PORTC.1=1;
    PORTC.2=1;
    PORTC.3=1;
    delay_ms(200);
    PORTC.0=0;
    PORTC.1=0;
    PORTC.2=0;
    PORTC.3=0;
    PORTC.4=1;
    PORTC.5=1;
    PORTC.6=1;
    PORTC.7=1;
    delay_ms(200);
    PORTC.4=0;
    PORTC.5=0;
    PORTC.6=0;
    PORTC.7=0;
    delay_ms(200)
    };
}
```

مثال ۲ :

برنامه ای بنویسید که LEDها را از ابتدا به ترتیب به مدت ms ۲۰۰ روشن و خاموش نماید.

همانند مثال بالا باید ماژول LED را توسط کابل IDC به پورت C، Main Bord متصل نمایید. برای پروگرم کردن میکرو کنترلر با USB پس از ساخت کد Hex در نرم افزار Code Vision و اتصال Main Bord به کامپیوتر، کد Hex مربوطه را در برنامه ی Progisp، Load کرده و بر روی گزینه های Erase و Auto کلیک می نمایید.

این برنامه را می توان به دو صورت نوشت:

الف)

```
#include <mega16.h>
#include <delay.h>
void main(void)
```

```
{
PORTA=0x00;
DDRA=0x00;

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0xff;

PORTD=0x00;
DDRD=0x00;

while (1)
    {
    PORTC.0=1;
    delay_ms (200);
    PORTC.0=0;
    PORTC.1=1;
    delay_ms (200);
    PORTC.1=0;
    PORTC.2=1;
    delay_ms (200);
    PORTC.2=0;
    PORTC.3=1;
    delay_ms (200);
    PORTC.3=0;
    PORTC.4=1;
    delay_ms (200);
    PORTC.4=0;
    PORTC.5=1;
    delay_ms (200);
    PORTC.5=0;
    PORTC.6=1;
    delay_ms (200);
    PORTC.6=0;
    PORTC.7=1;
    delay_ms (200);
    PORTC.7=0;
    };
}
```

ب)

```
#include <mega16.h>
#include <delay.h>
int i;
```

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

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0xff;

PORTD=0x00;
DDRD=0x00;

while (1)
    {
    for (i=1; i<=256; i=i*2)
    {
    PORTC=i;
    delay_ms (200);
    PORTC=0;
    }
    };
}
```

مثال ۳ :

برنامه ای بنویسید که LEDها را از انتها به ترتیب به مدت ۲۰۰ ms روشن و خاموش نماید.

جهت اجرای برنامه باید همانند مثال های بالا عمل نمایید.

```
#include <mega16.h>
#include <delay.h>
int i;
void main(void)
{
PORTA=0x00;
DDRA=0x00;

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0xff;

PORTD=0x00;
DDRD=0x00;
while (1)
    {
```

```
    for(i=256;i>=1;i=i/2)
    {
    PORTC=i;
    delay_ms(200);
    PORTC=0;
    }
    };
}
```

مثال ۴ :

برنامه ای بنویسید که با استفاده از تابع rand هشت عدد LED را به صورت تصادفی روشن و خاموش نماید.

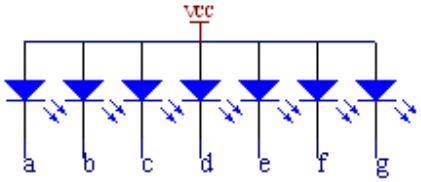
جهت اجرای برنامه باید همانند مثال های بالا عمل نمایید.

```
#include <mega16.h>
#include <delay.h>
#include <stdlib.h>
int i;
void main(void)
{
PORTA=0x00;
DDRA=0x00;

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0xff;

PORTD=0x00;
DDRD=0x00;


while (1)
    {
    for(i=0;i<=10;i++)
    {
    PORTC=rand();
    delay_ms(250);
    }
    for(i=0;i<=5;i++)
    {
    PORTC=0xFF;
    delay_ms(200);
    PORTC=0x00;
    delay_ms(200);
    }
```

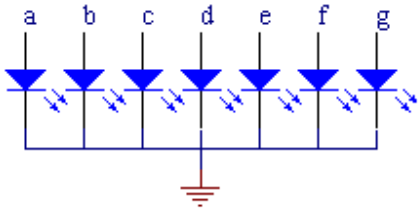
```
    };
}
```

نمایشگر هفت قسمتی (7.segment)

7. Segment یکی از پرکاربرد ترین نمایشگرها در صنعت می باشند که در دو نوع آند مشترک و کاتد مشترک ساخته می شود.

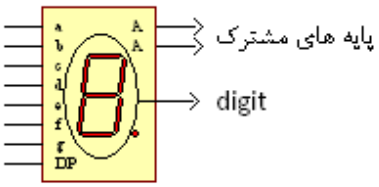


آند مشترک



کاتد مشترک

در 7.Segmentهاي آند مشترک پايه هاي مشترک توسط يک مقاومت به VCC ودر نوع کاتد مشترک پايه هاي مشترک توسط يک مقاومت به GND متصل مي باشد.



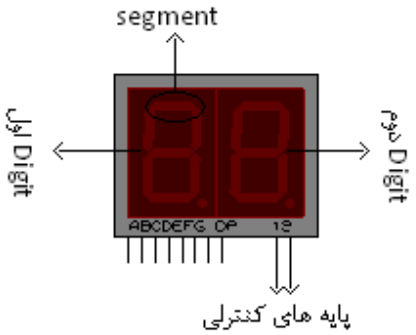
کدهای اعداد ۰ تا ۹ در دو نوع آند مشترک و کاتد مشترک درجدول زیر نشان داده شده است.

کاتد مشترک		آند مشترک	
کد	عدد	کد	عدد
0x3F	0	0xC0	0
0x06	1	0xF9	1
0x5B	2	0xA4	2
0x4F	3	0xB0	3
0x66	4	0x99	4
0x6D	5	0x92	5

6	0x83	6	0x7C
7	0xF8	7	0x07
8	0x80	8	0x7F
9	0x98	9	0x67

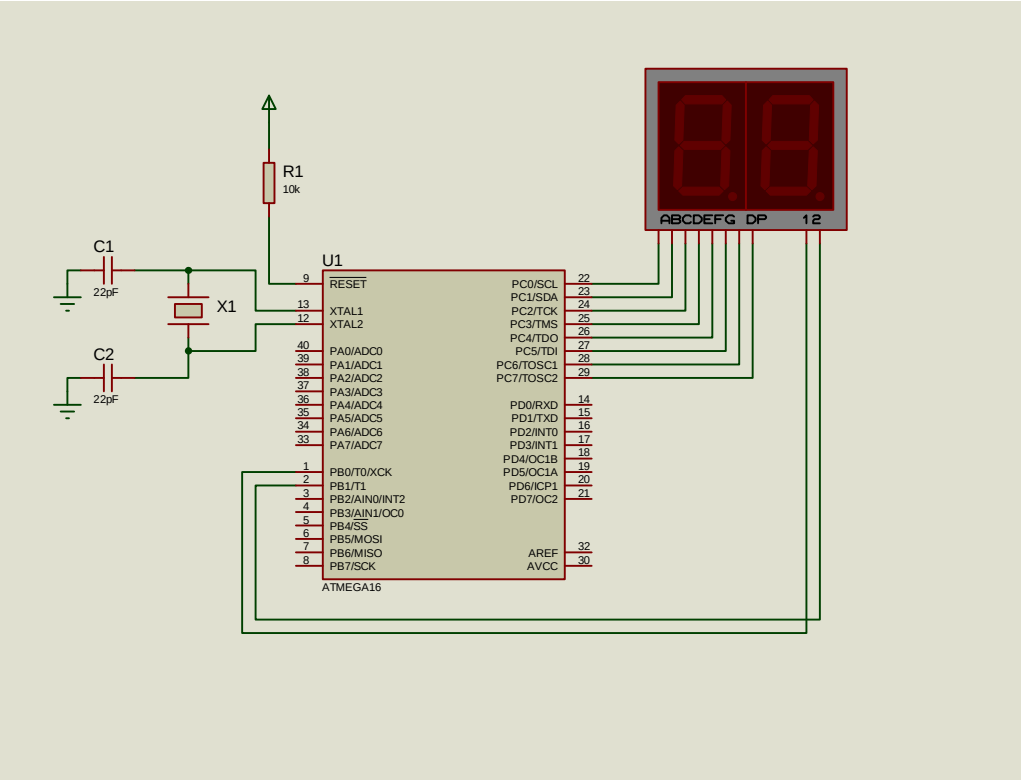
عیب عمده ی استفاده از این نمایشگرها در این است که در صورت استفاده کردن از چندین 7.segment پایه های زیادی از میکروکنترولر اشغال می شود. برای رفع این مشکل باید پایه های a تا g همه ی 7.segment را به هم وصل کنیم و پایه های مشترک آنها را کنترل کنیم.

اگر 7.segment از نوع آند مشترک باشد پایه های کنترلی با یک وپایه های دیتا با صفر و اگر کاتد مشترک باشد پایه های کنترلی با صفر و پایه های دیتا با یک فعال می شود.



در ماژول طراحی شده 7.segment از نوع آند مشترک می باشد لذا برای روشن کردن segment باید پایه ی متناظر با آن را صفر کرد. همچنین دارای دو پایه به عنوان پایه های کنترلی می باشد، این پایه ها برای تعیین اینکه کدام Digit روشن شود به کار میرود. برای روشن شدن هر Digit باید پایه ی کنترلی آن را یک کرد. پایه های کنترلی به ترتیب به PortB.0 وPortB.1 متصل می باشد.

7.segmentاستفاده شده در این برد 2 digit می باشد که هر digit شامل هفت عدد LED برای نمایش اعداد و حروف و یک عدد LED برای نمایش نقطه ی ممیز اعداد اعشاری به کار می رود، دو پایه نیز جهت کنترل در نظر گرفته شده است.



مثال ۵ :

برنامه ای بنویسید که اعداد ۰ تا ۹۹ را بر روی 7.segment نمایش دهد.

جهت اجرای این برنامه باید از ماژول 7.segment استفاده کرد. دیتای این ماژول را توسط کابل IDC به پورت C و Control ماژول را توسط کابل IDC به پورت B (کنترل digit اول توسط PORTB.0، کنترل digit دوم توسط PORTB.1 می باشد) Main Bord متصل نمایید. برای پروگرم کردن میکرو کنترلر با USB پس از ساخت کد Hex در نرم افزار Code Vision و اتصال Main Bord به کامپیوتر، کد Hex مربوطه را در برنامه ی Progisp، Load کرده و بر روی گزینه های Erase و Auto کلیک می نمایید.

```
#include <mega16.h>
#include <delay.h>
int i;
int j;
int k;
unsigned char x[10]={0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0x80,0x90};
void main(void)
{
PORTA=0x00;
DDRA=0xFF;

PORTB=0x00;
DDRB=0xFF;
```

```
PORTC=0x00;
DDRC=0xff;

PORTD=0x00;
DDRD=0x00;

while (1)
    {
    for(i=0;i<=9;i++)
    {
    for(j=0;j<=9;j++)
    {
    for(k=0;k<=99;k++)
    {
    PORTC=x[i];
    PORTB.0=1;
    PORTB.1=0;
    delay_ms(5);
    PORTC=x[j];
    PORTB.0=0;
    PORTB.1=1;
    delay_ms(5);
    }
    }
    }
        };
}
```

نمایشگر کریستال مایع LCD:

LCDها یا همان نمایشگرهای کریستال مایع یکی دیگر از پرکاربردترین نمایشگرها می باشند که به راحتی می توان اعداد، حروف بزرگ و کوچک، کاراکترهای خاص و حتی کاراکترهای تعریف شده توسط کاربر را در آن نمایش داد.
LCDها در دو نوع متنی و گرافیکی ساخته می شوند که در نوع متنی فقط قادر به نوشتن اعداد، حروف و تعدادی کاراکتر خاص هستیم. LCD های متنی در انواع مختلفی ساخته می شوند که معمولاً نوع آنها با تعداد سطرها و ستون های آن مشخص می شود.
LCD استفاده شده در اینجا از نوع ۲*۱۶ می باشد.
توصیف پایه های LCD:
VSS، VCC، VEE: پایه های VSS، VCC، VEE به GND، VCC و کنترل شدت درخشندگی LCD را بر عهده دارند.
RS (انتخاب رجیستر): در LCD دو رجیستر به نام رجیستر دستورالعمل و رجیستر داده وجود دارد. اگر RS=0 باشد، رجیستر دستور العمل، در غیر این صورت رجیستر داده انتخاب می شود که در این ماژول به پایه ی PORTC.0 میکروکنترلر متصل می باشد.

R/W (خواندن/ نوشتن): اگر R/W=1 عمل خواندن از LCD و اگر R/W=0 عمل نوشتن در LCD انجام می شود. پایه ی مذکور به PORTC.1 میکروکنترلر متصل می باشد.

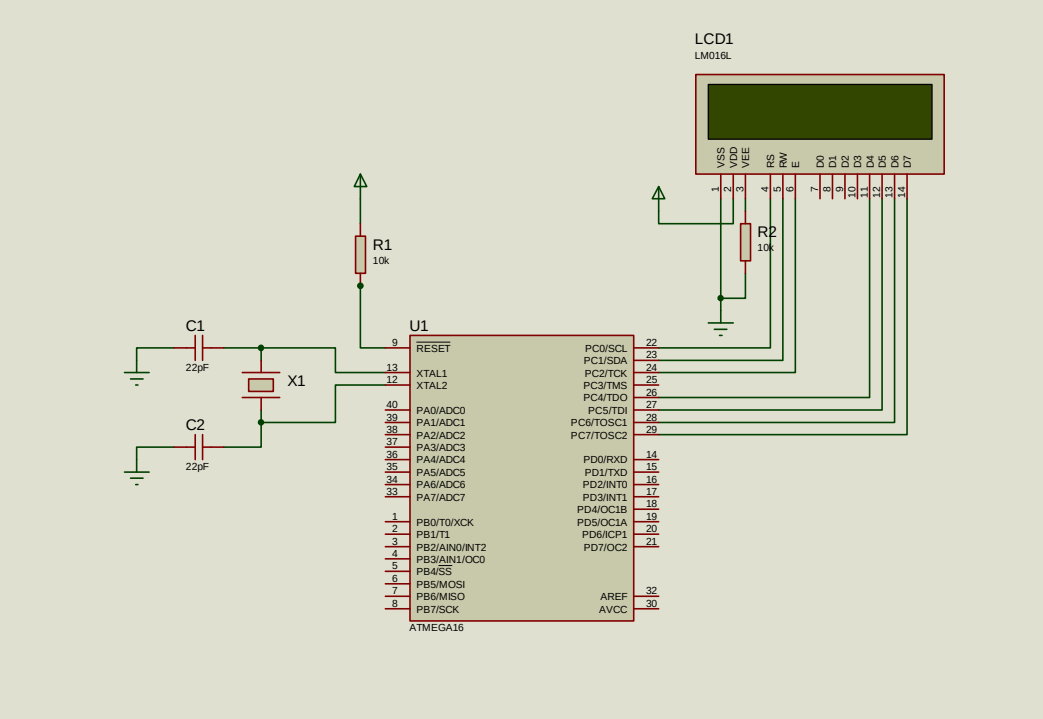
E (فعال ساز): LCD به کمک پایه ی E، اطلاعات روی پایه های داده ی خود را لچ می کند. یا اعمال یک پالس یک به صفر با حداقل پهنای 40ns می توان پایه ی E را فعال نمود.

پایه ی E به PORTC.2 میکروکنترلر متصل می باشد.

D0-D7 (هشت بیت خط داده): از این پایه ها برای و دستور به LCD و یا خواندن رجیسترهای داخلی LCD استفاده می شود. به طوری که اگر RS=1 باشد، اطلاعات روی این پایه ها به منزله ی داده تلقی شده و در غیر این صورت دستورالعمل معرفی می شود.

در انتقال اطلاعات به LCD به صورت موازی، دو انتقال چهار بیتی و هشت بیتی در دسترس است. در انتقال به صورت هشت بیت از تمام پایه های D0 تا D7 استفاده می شود، ولی در انتقال به صورت چهار بیت از پایه های D4 تا D7 استفاده خواهد شد.

در این ماژول انتقال به صورت چهار بیت می باشد که پایه های D4 تا D7 به ترتیب به پایه های PORTC.4، PORTC.5، PORTC.6 و PORTC.7 متصل می باشد. همچنین ماژول طراحی شده جهت کار در دو محیط Code Vision و BASCOM طراحی شده است، برای کار در محیط Code Vision باید Jumper در محل C قرار گیرد و جهت کار در محیط BASCOM باید Jumper در محل B قرار گیرد.



مثال ۶ :

برنامه ای بنویسید که عبارت "ariamec school" را در دو سطر بر روی LCD نمایش دهد.

جهت اجرای این برنامه باید ماژول LCD را توسط کابل IDC به پورت C، Main Bord متصل نمایید. برای پروگرم کردن نیز میکرو کنترلر با USB پس از ساخت کد Hex در نرم افزارCode Vision و اتصال Main Bord به کامپیوتر، کد Hex مربوطه را در برنامه ی Progisp، Load کرده و بر روی گزینه های Erase و Auto کلیک می نمایید.

```
#include <mega16.h>
#include <delay.h>
int i;
int j;
#asm
  .equ __lcd_port=0x15; PORTC
#endasm
#include <lcd.h>
void main(void)
{
PORTA=0x00;
DDRA=0x00;

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0xff;

PORTD=0x00;
DDRD=0x00;

// LCD module initialization
lcd_init (16);

while (1)
    {
    for (i=0; i<=10;i=i+3)
    {
    for(j=0;j<=15;j=j+2)
    {
    lcd_clear ();
    lcd_gotoxy (i, 0);
    lcd_putsf ("ariamec");
    delay_ms (400);
    lcd_clear();
    lcd_gotoxy (j, 1);
    lcd_putsf ("school");
    delay_ms (400);
    }
    }
    };
}
```

مثال ۷ :

برنامه ای بنویسید که ساعت را بر روی LCD نمایش دهد.
جهت اجرای برنامه باید همانند مثال های بالا عمل نمایید.

```
#include <mega16.h>
#include <delay.h>
#include <stdlib.h>
#asm
  .equ __lcd_port=0x15 ;PORTC
#endasm
#include <lcd.h>
unsigned int   sec,min=0,hor=0;
unsigned char  ssec[3],smin[3],shor[3];
void ref_key(void)
{
 if(PIND.0==0)
 {
 min++;
 if(min>=60) min=0;
 }
 if(PIND.1==0)
 {
 hor++;
 if(hor>=24) hor=0;
 }
}
void main(void)
{
PORTA=0x00;
DDRA=0x00;

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0x00;

PORTD=0xff;
DDRD=0x00;
lcd_init(16);
while (1)
    {

    delay_ms(1000);
    ref_key();
    sec++;
    if(sec>=60){
```

```
    sec=0;
    min++;
    if(min>=60){
    min=0;
    hor++;
    if(hor>=24){
    hor=0;
    }
    }
    }
    itoa(sec,ssec);
    itoa(min,smin);
    itoa(hor,shor);
    lcd_clear();
    lcd_gotoxy(3,0);
    lcd_puts(shor);
    lcd_putsf(":");
    lcd_puts(smin);
    lcd_putsf(":");
    lcd_puts(ssec);
    };
}
```

توضیح برنامه:

در این برنامه ابتدا در هر ثانیه یک واحد به متغیر sec افزوده می شود و پس از رسیدن به عدد ۶۰ این متغیر صفر شده و یک واحد به متغیر min افزوده می شود، این روند برای متغیرهای min و hor نیز ادامه می یابد. کلید های min و hor نیز به ترتیب برای تنظیم دقیقه و ساعت می باشند که توسط ماژول Push Button به ترتیب به پایه های PORTD.0 و PORTD.1 متصل می باشد.

LCD گرافیکی:

GLCD (LCD گرافیکی) از تعدادی پیکسل که در کنار هم قرار داده شده اند بوجود می آید. LCD های گرافیکی در انواع مختلفی ساخته می شوند که معمولاً نوع آنها با تعداد سطرها و ستون های آن مشخص می شود. عمده ترین آنها ، نوع گرافیکی ۱۲۸*۶۴ می باشد.

توصیف پایه های LCD:

پایه ی VSS: این پایه ی به GND متصل است.

پایه ی VDD: پایه ی تغذیه ی GLCD به ولتاژی بین ۴/۵ تا ۵/۵ یا همان VCC متصل است.

پایه ی VO: پایه ای جهت کنترل روشنایی صفحه که معمولاً توسط یک پتانسیومتر و از پایه ی VEE تغذیه می شود.

پایه ی RS: اگر D/I=0 باشد، رجیستر

پایه ی

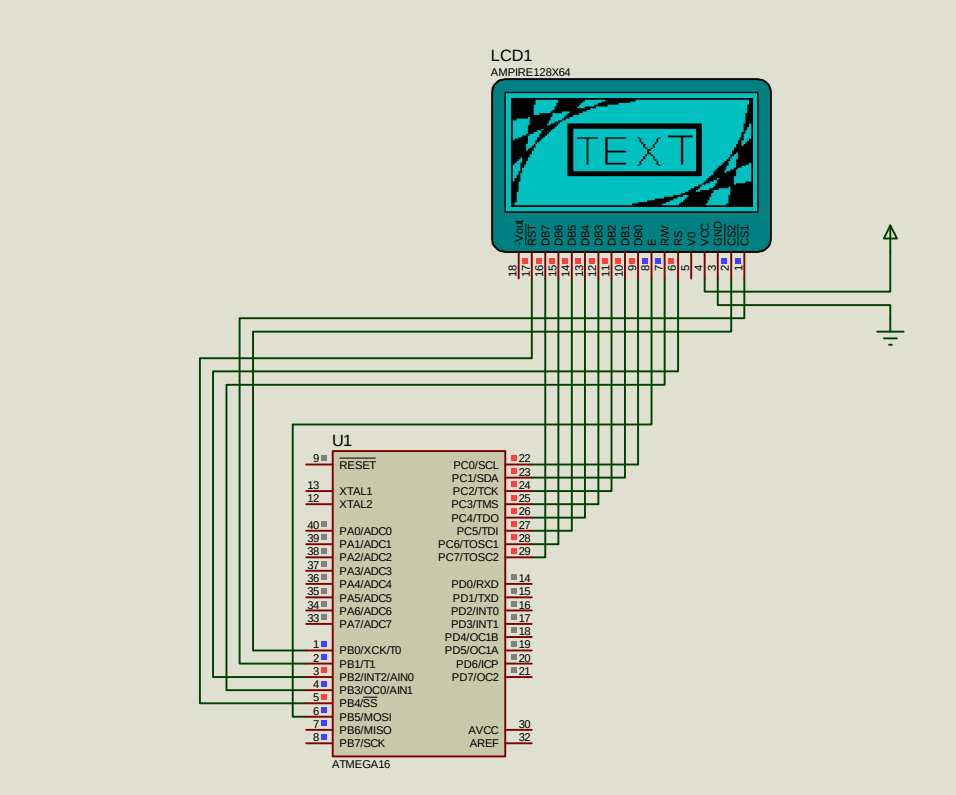
پایه ی
پایه ی
پایه ی
پایه ی
پایه ی

نرم افزارهای مختلفی برای تبدیل عکس به کدهای دیجیتالی وجود دارد از جمله می توان به نرم افزار Qt-LCDbin اشاره کرد.

نرم افزار Qt-LCDbin:

این نرم افزار برای تبدیل عکس هایی با فرمت BMP به کدهای قابل استفاده در برنامه می باشد. نحوه ی استفاده از این نرم افزار بدین صورت می باشد که باید ابتدا در قسمت Chager BMP عکسی که قبلاً آماده کرده ایم (با فرمت BMP و رزولیشن ۱۲۸*۶۴ باشد) را وارد کنیم. پس از وارد کردن عکس، از قسمت Sauvegarder BIN عکس را با فرمت BIN در حافظه ذخیره می کنیم و با استفاده از گزینه ی Charger BIN فایلی که ذخیره کرده ایم را Load می کنیم.

حال بر روی سربرگ Export ASCII رفته و بر روی گزینه ی Exporter(Presse Papier) کلیک کنید. با زدن این گزینه کدهای عکس مورد نظر ساخته می شود که با رفتن بر روی سربرگ Clipboard text کدها را کپی کرده و یا اینکه گزینه ی Exporter... را در داخل سربرگ Export ASCII زده و کدها را داخل یک فایل txt ذخیره کنید.



همانطور که در شکل بالا ملاحظه می کنید ترتیب پایه های کنترلی به صورت زیر می باشد.

LCD_E: PORTB.5
LCD_RW: PORTB.3
LCD_RS: PORTB.2

LCD_CS1: PORTB.0
LCD_CS2: PORTB.1
LCD_RST: PORTB.4

مثال ۸ :

برنامه ای بنویسید که دو عکس را به ترتیب نمایش دهد.

جهت اجرای این برنامه باید از ماژول GLCD استفاده کرد. دیتای این ماژول را توسط کابل IDC به پورت C و Control ماژول را توسط کابل IDC به پورت B، Main Bord متصل نمایید. برای پروگرم کردن میکرو کنترلر با USB پس از ساخت کد Hex در نرم افزارCode Vision و اتصال Main Bord به کامپیوتر، کد Hex مربوطه را در برنامه ی Progisp، Load کرده و بر روی گزینه های Erase و Auto کلیک می نمایید.

```
#include<mega16.h>
#include<delay.h>
#define LCD_E PORTB.5
#define LCD_RW PORTB.3
#define LCD_RS PORTB.2
#define LCD_CS1 PORTB.0
#define LCD_CS2 PORTB.1
#define LCD_RST PORTB.4

flash unsigned char image1 [1024] = {
   0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x0F, 0x01, 0x01,0x01,
0x01, 0x01, 0x01, 0x01, 0x01, 0x81, 0xE1, 0xF9, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,0xFF, 0xFF, 0xDF, 0xCF, 0xC7, 0xC3, 0xE1, 0x61, 0x61, 0x21, 0x21, 0x31, 0x11, 0x19,
0x19,0x1D, 0x1D, 0x1F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x07, 0x07, 0x07, 0x07, 0x07, 0x07,
0x03,0xFF, 0x1F, 0x1F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x0F, 0x07, 0xFF, 0xFF, 0xFC, 0xFC, 0xFC,
0Xfe,0xFE, 0xFE, 0xFE, 0x7E, 0x1E, 0x87, 0x81, 0xC1, 0xC1, 0x61, 0x21, 0x10, 0x18, 0x0C, 0x8E,
0x87,0x87, 0x83, 0x83, 0x83, 0x81, 0x81, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
0x80,0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
0x80,0xFF, 0x00, 0x00, 0x00, 0x80, 0x80, 0xC0, 0xC0, 0xE0, 0x3F, 0x1F, 0x1F, 0x0F, 0x0F, 0x87,
0xE7,0xFB, 0x7F, 0x3F, 0x1F, 0x0F, 0x07, 0x03, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF,
0xFF,0xFF, 0x03, 0x03, 0x83, 0x83, 0x83, 0x83, 0x83, 0x83, 0x83, 0x83, 0x83, 0x83, 0x83, 0x03,
0x03,0x03, 0x03, 0x83, 0x83, 0x83, 0x83, 0x83, 0x83, 0x83, 0x83, 0x83, 0x83, 0x83, 0x03, 0x03,
0x03,0xFF, 0xFE, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x87, 0xC0, 0xE0, 0x70, 0x18, 0x06, 0x03,
0x01,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF,
0xFF,0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0xFF, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x00, 0x00,
0x00,0xFF, 0x7F, 0x3F, 0x1F, 0x0F, 0xC7, 0xFF, 0x7F, 0x1F, 0x07, 0x01, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF,
0xFF,0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7F, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0x7F, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x00, 0x00,
0x00,0xFF, 0x00, 0x80, 0xE0, 0xF8, 0x0F, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x3F,
0x3F,0x3F, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38,
```

```
0x38,0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38,
0x38,0xFF, 0xFE, 0xFF, 0x7F, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0xFF, 0xFF, 0x9F, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
0x80,0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
0x80,0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
0x80,0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
0x80,0x03, 0x03, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
0x01,0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
0x01,0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01,
0x01,0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0xF1, 0xFF,
0xFF,0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
0x80,0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
0x80,0x80, 0x80, 0x80, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xFE, 0xFF, 0x7F,
0xFF,0x03, 0x03, 0x03, 0x83, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x03, 0x83,
0x03,0x03, 0x03, 0x03, 0x43, 0x43, 0x43, 0x43, 0x43, 0x43, 0xC3, 0x43, 0x43, 0x43, 0x43,
0x43,0x03, 0x03, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xF0, 0x1F, 0x07, 0x01, 0x00,
0xFF,0x00, 0x00, 0x00, 0x00, 0x03, 0x04, 0x18, 0x20, 0xC0, 0x40, 0x20, 0x18, 0x04, 0x03, 0x00,
0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xE0, 0xF8, 0xFE, 0xFF, 0xF1, 0xF8, 0xFC, 0xFE, 0xFF,
0xFF,0x00, 0x00, 0x00, 0x80, 0x60, 0x10, 0x0C, 0x02, 0x01, 0x01, 0x02, 0x0C, 0x10, 0x60, 0x80,
0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7F, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x80,0xC0, 0xE0, 0xF0, 0x18, 0x06, 0x03, 0x01, 0xF0, 0xFF, 0xFF, 0xFF, 0xFF, 0x7F, 0x7F, 0x3F,
0xFF,0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38,
0x38,0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38, 0x38,
0x38,0x38, 0x38, 0x3F, 0x3F, 0x3F, 0x00, 0x00, 0x00, 0x80, 0xC0, 0xE0, 0xF8, 0xFC, 0x7E,
0xFF,0xFF,0xFF, 0xF3, 0xF8, 0xF8, 0xFC, 0xFC, 0x3E, 0x03, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00,
0x00, 0xFF,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0x80,
0xC0, 0xE0,0xE0, 0xF0, 0xB0, 0x98, 0xC8, 0xC4, 0xC6, 0xC3, 0xC3, 0xE1, 0xE1, 0x30, 0x3C, 0x3F,
0x1F, 0x1F,0x1F, 0x1F, 0x1F, 0x0F, 0xFF, 0xFF, 0xF8, 0xFC, 0xFC, 0xFC, 0xFC, 0xFC, 0xFE, 0xFE,
0xFE, 0xFF,0xC0, 0xC0, 0xC0, 0xE0, 0xE0, 0xE0, 0xE0, 0xE0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF8, 0xF8,
0xF8, 0xF8,0xB8, 0xBC, 0x9C, 0x9C, 0x8C, 0x8C, 0x86, 0x86, 0x82, 0xC2, 0xE3, 0xF1, 0xF9, 0xFD,
0xFF, 0xFF,0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xBF, 0x8F, 0x83, 0x80, 0x80, 0x80, 0x80,
0x80, 0x80,0x80, 0x80, 0xC0, 0xFE, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0xFF
};

flash unsigned char image2[1024] = {
   0x00, 0x80, 0x80, 0x80, 0x00, 0xE0, 0xF8, 0xFC, 0xFC, 0xFC, 0xF0, 0xF8, 0xF8, 0xFC, 0xFC,
0xFC,0xF8, 0xF8, 0xFC, 0xFC, 0xFC, 0xFC, 0xF0, 0xF0, 0xE0, 0xC0, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0x40, 0x40, 0x40, 0x44, 0x44, 0x44, 0xE4, 0xF8, 0xF9, 0xFE, 0xF8, 0xFC, 0xFE,
0xFD,0x00, 0x07, 0x0F, 0x0F, 0x7F, 0xFF, 0xFF, 0xFF, 0x7F, 0x7F, 0x3F, 0x7F, 0xFF, 0xFF, 0xFF,
0xFF, 0xFF, 0x7F, 0x3F, 0x1F, 0x7F, 0x7F, 0x7F, 0x7F, 0x1F, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0x00, 0x00, 0x04, 0x04, 0x04, 0x02, 0x03, 0x23, 0x1F, 0xC7, 0x37, 0x0F, 0x0F,
0xFF,0xE0, 0x20, 0x20, 0x20, 0x60, 0x60, 0x18, 0x08, 0x0C, 0x04, 0x04, 0x04, 0x04, 0x1C, 0x10,
0x18,0x08, 0x08, 0x08, 0x18, 0xE0, 0x40, 0x60, 0x20, 0x40, 0xC0, 0x80, 0x00, 0x80, 0x80, 0x80,
0x80,0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80,
0x80,0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x80, 0x00, 0x00, 0x00, 0x00,
0x01,0x00, 0x00, 0x80, 0x00, 0x04, 0x0E, 0x0E, 0x04, 0x00, 0x00, 0x40, 0xE0, 0xE0, 0x42, 0x07,
0x07,0x02, 0x00, 0x00, 0x18, 0x38, 0x10, 0x80, 0xC0, 0xF0, 0xC8, 0xCF, 0xCF, 0x4F, 0x7F, 0x7F,
0x7F,0x7B, 0xF9, 0xF8, 0xD8, 0xC8, 0xCC, 0xCE, 0xCF, 0x4F, 0x7F, 0x7F, 0x7F, 0x7B, 0x79, 0xF8,
0xF8,0xF8, 0xC8, 0xCC, 0xCF, 0xCF, 0xCF, 0xEF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x7E, 0x7E,
0x3C,  0x00, 0x00, 0x01, 0x06, 0xBC, 0xE0, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xC0, 0xF8,
0x0E,0x03, 0x00, 0x00, 0x00, 0xFC, 0x07, 0x07, 0x07, 0xE7, 0x27, 0x25, 0x24, 0xE4, 0x24, 0x24,
0xE6,0x07, 0x07, 0x07, 0x07, 0x07, 0x07, 0x05, 0x04, 0x04, 0xE4, 0x24, 0x24, 0x24, 0xE7, 0x27,
0x27,0xE7, 0x07, 0x07, 0x07, 0xFF, 0x07, 0x07, 0x03, 0x03, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00,
0x00,0xC2, 0xC2, 0xC2, 0xC3, 0xC1, 0xC0, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFC, 0xC3, 0xC1,
0xC3,0xC6, 0xC4, 0xC4, 0xC4, 0xFF, 0x00, 0x00, 0x00, 0x0F, 0x08, 0x08, 0x08, 0x0F, 0x08, 0x08,
0x0F,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0F, 0x08, 0x08, 0x08, 0x0F, 0x08,
0x08,0x0F, 0x00, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFC, 0x02, 0x02,
0xFF,0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0x1F, 0x87, 0x80, 0xC0, 0x80, 0x00, 0x00, 0x80, 0xC3, 0xCF,
0xFF,0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0,
0xC0,0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0,
0xC0,0xC0, 0xC0, 0xC0, 0xC0, 0xFF, 0xE0, 0xE0, 0xF0, 0xF0, 0xF8, 0xF8, 0xFC, 0xFF, 0xF2, 0xF2,
0xE3,0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,0xFC, 0xF8, 0xF8, 0xFC, 0xF2, 0xE0, 0x90, 0x90, 0x08, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0x00, 0xC0, 0xE0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF0, 0xF8, 0xF8, 0xF0, 0xF8, 0xFC,
0xFC,0xFC, 0xFC, 0xFC, 0xFC, 0xFE, 0xFE, 0xFE, 0xFE, 0xFC, 0xF0, 0xF0, 0xF0, 0xE0, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x0F, 0x07, 0x7F, 0x07, 0x03, 0x0D, 0x10, 0x00, 0x01, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0x00, 0x01, 0x07, 0x0F, 0x0F, 0x07, 0x07, 0x0F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F,
0x0F,0x07, 0x0F, 0x1F, 0x1F, 0x1F, 0x1F, 0x1F, 0x0F, 0x07, 0x07, 0x07, 0x07, 0x01, 0x00, 0x00,
0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x80, 0xE0, 0x38, 0x08, 0x08, 0x08, 0x08, 0x88,
0x88, 0x18, 0x0C, 0x04, 0x06, 0x02, 0x02, 0x02, 0x82, 0xC2, 0xC4, 0x84, 0x5C, 0x70, 0x60, 0x20,
0x20,0x20, 0x20, 0x20, 0x40, 0xC0, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x38, 0x18, 0xF0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x80, 0x80, 0x80, 0xFC, 0xC7, 0x01, 0x01, 0x01, 0x00, 0x00, 0x20, 0x70, 0x70, 0x21, 0x03,
0x03, 0x01, 0x00, 0x00, 0x80, 0xC0, 0xC0, 0x80, 0x00, 0x01, 0x01, 0x10, 0x38, 0x38, 0x10, 0x00,
0x00,0x00, 0x00, 0x00, 0x00, 0x10, 0x3F, 0x60, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x80, 0x80, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

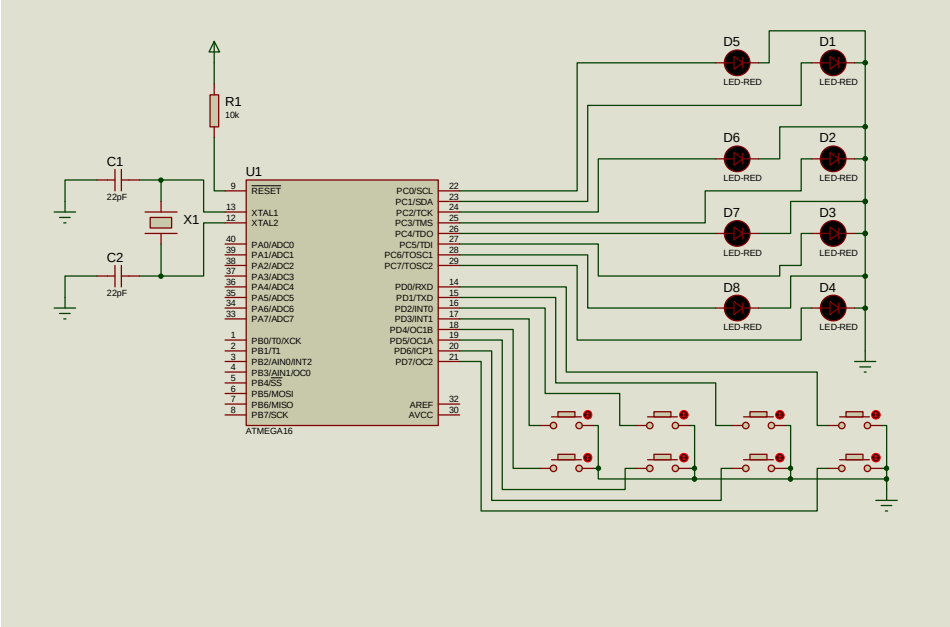
```
0x00,0x0F, 0x18, 0x30, 0x21, 0x41, 0xC0, 0x00, 0x00, 0x08, 0x1C, 0x1C, 0x08, 0x00, 0x02, 0x0E,
0x38,0xE0, 0x80, 0x00, 0x00, 0x01, 0x01, 0x00, 0x00, 0x00, 0x80, 0x70, 0x78, 0x8E, 0x80, 0x00,
0x00,0x00, 0x04, 0x0E, 0x0E, 0x24, 0xE0, 0x20, 0x31, 0x13, 0x1C, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,0x00, 0x80, 0xFF, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0,
0xC0,0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xCF, 0xC8, 0xC8, 0xC8, 0xC8, 0xC8, 0xC8, 0xC8, 0xC8, 0xC8,
0xC8,0xC4, 0xC7, 0xFE, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xFF, 0xC0, 0xC0, 0xC0, 0xC1, 0xC3,
0xC2,0xC4, 0xC4, 0xC4, 0xC4, 0xC6, 0xC3, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0, 0xC0,
0xC0,0xC1, 0xC0, 0xC0, 0xC1, 0x83, 0x83, 0x07, 0x07, 0x07, 0x07, 0x07, 0x0F, 0x0F, 0x3F, 0x7F,
0xFF,0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,0x3F, 0x0F, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x07, 0x3F, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF,0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFE, 0xFE, 0xFC, 0xF8, 0xF0, 0xE0, 0xE0, 0xE0,
0xC1,0x03, 0x03, 0x07, 0x0F, 0x0F, 0x0F, 0x1F, 0x1F, 0x1F, 0x1F, 0x3F, 0x7F, 0x7F, 0xFF, 0xF9,
0xF8,0xFC, 0xFE, 0xFE, 0xFE, 0xFC, 0xF8, 0xF8, 0xF8, 0xFC, 0xFE, 0xFE, 0xFE, 0xFD, 0xFD, 0xFF,
0xFF,0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF
};

void LCD (char DATA)
 {

 PORTC=DATA;
 delay_us(10);
 LCD_E=1;
 delay_us(10);
 LCD_E=0;
 delay_us(10);
 }
 void namayesh1(void)
     {
unsigned char x,y,cs;
unsigned int ptr;
 LCD_E=0;
 LCD_RW=0;
 delay_ms(10);
 LCD_RST=1;
 ptr=0;
 for(cs=0;cs<=1;cs++)
  {
  LCD_RS=0;
  if (cs==1)
   {
   LCD_CS1=0;
   LCD_CS2=1;
   delay_ms(1);
   }
   else
   {
```

```
 LCD_CS1=1;
 LCD_CS2=0;
 }
LCD(0x3F);
LCD(0xB8);
LCD(0xC0);
delay_ms(1);
for(x=0;x<=7;x++)
 {
   LCD_RS=0;
   LCD(0x40);
  LCD(0xB8 + x);
   LCD_RS=1;
    for(y=0;y<=63;y++)
 {
   LCD(image1[ptr]);
    ptr++;
   }
 }
}
 LCD_CS1=0;
 LCD_CS2=0;
   }
void namayesh2(void)
    {
unsigned char x,y,cs;
unsigned int ptr;
 LCD_E=0;
 LCD_RW=0;
 delay_ms(10);
 LCD_RST=1;
 ptr=0;
 for(cs=0;cs<=1;cs++)
 {
 LCD_RS=0;
 if (cs==1)
 {
 LCD_CS1=0;
 LCD_CS2=1;
 delay_ms(1);
 }
 else
 {
 LCD_CS1=1;
 LCD_CS2=0;
 }
LCD(0x3F);
```

```
LCD(0xB8);
LCD(0xC0);
delay_ms(1);
for(x=0;x<=7;x++)
 {
   LCD_RS=0;
   LCD(0x40);
  LCD(0xB8 + x);
   LCD_RS=1;
    for(y=0;y<=63;y++)
 {
   LCD(image2[ptr]);
    ptr++;
   }
 }
}
LCD_CS1=0;
LCD_CS2=0;
  }
void main(void)
{
DDRB=0xFF;
DDRC=0xFF;
 while(1)
 {
while(1)
 {
 namayesh2();
 delay_ms(1000);
break;
 };
   while(1)
 {
 namayesh1();
 delay_ms(1000);
break;
 };
  };
}
```

Push Button

برد Push Button جهت ارسال داده به میکروکنترلر استفاده می شود. در برد طراحی شده یک پایه از تمامی Push Buttonها به صورت مشترک به GND متصل می باشد، یعنی درهر حالتی صفر می فرستد.



مثال۹ :

برنامه ای بنویسید که با فشردن هر Push Button، LED متناظر با آن روشن شود.

جهت اجرای این برنامه باید ماژول LED را به پورت C و ماژول Push Button را به پورت D، Main Bord متصل کرد. برای پروگرم کردن میکرو کنترلر با USB پس از ساخت کد Hex در نرم افزارCode Vision و اتصال Main Bord به کامپیوتر، کد Hex مربوطه را در برنامه ی Progisp، Load کرده و بر روی گزینه های Erase و Auto کلیک می نمایید.

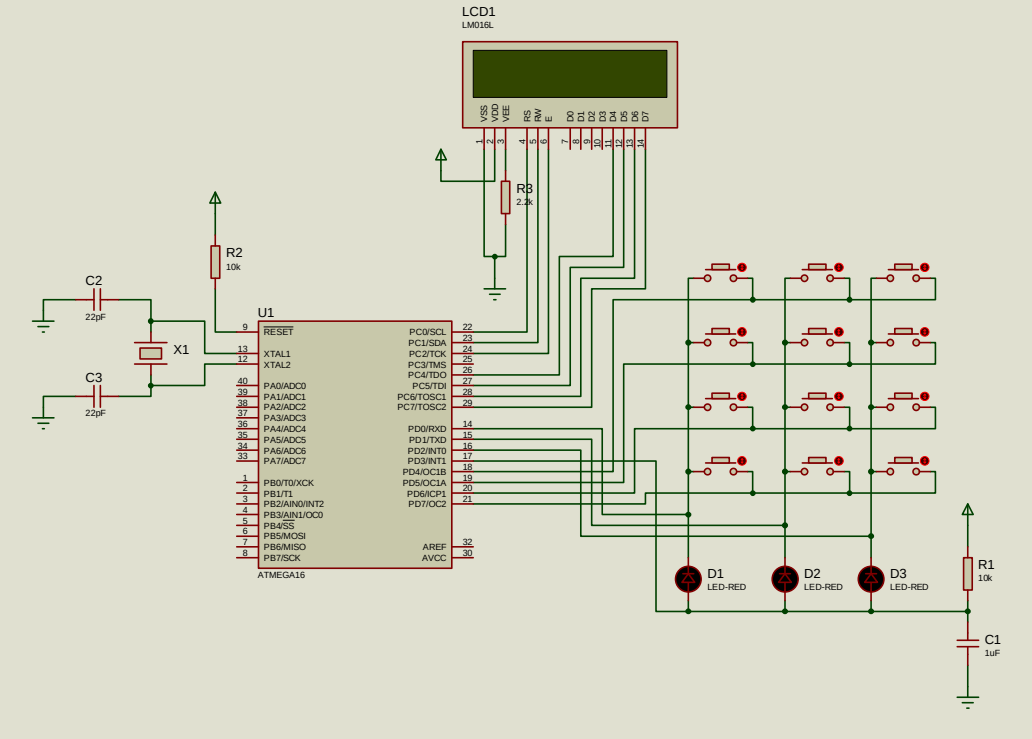
```
#include <mega16.h>
#include <delay.h>

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

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0xff;

PORTD=0xFF;
DDRD=0x00;

while (1)
    {
  if(PIND.0==0)
  PORTC.0=1;
```

```
  if(PIND.1==0)
  PORTC.1=1;
  if(PIND.2==0)
  PORTC.2=1;
  if(PIND.3==0)
  PORTC.3=1;
  if(PIND.4==0)
  PORTC.4=1;
  if(PIND.5==0)
  PORTC.5=1;
  if(PIND.6==0)
  PORTC.6=1;
  if(PIND.7==0)
  PORTC.7=1;
    };
}
```

صفحه کلید ماتریسی:

صفحه کلید برای انتقال اطلاعات کاربر برای برخی تنظیمات در داخل میکروکنترلر مورد استفاده قرار می گیرد. برای اتصال صفحه کلید به میکروکنترلر راههای مختلفی وجود دارد در ساده ترین روش چهار بیت از پورتهای میکروکنترلر به صورت ورودی و سه بیت دیگر به صورت خروجی تعریف و پایه های صفحه کلید به هم متصل می شود. در روی ورودی ها باید چهار عدد مقاومت pull – up قرار داده شود. برای تشخیص کلید فشرده شده در صفحه کلید باید هر یک از پایه هایی که به صورت خروجی تعریف شده اند را به نوبت صفر کرده و بعد از هر بار صفر کردن پایه ها چهار بیت اول را که به صورت ورودی تعریف شده اند را چک کنیم. در واقع ما باید سطرها را جاروب و ستونها را بخوانیم. نحوه خواندن کلیدها بدین گونه است که باید ابتدا سطر اول را صفر کرده و ستونهای اول تا چهارم را چک کنیم. به عنوان مثال اگر ستون اول صفر شده باشد معلوم می شود که کلید ۱ فشرده شده است و اگر هیچ یک از ستونها صفر نشده بود اینبار سطر دوم را صفر کرده و دوباره ستونها را چک می کنیم به همین منوال باید هر چهار سطر را جاروب کرده و با چک کردن ستونها تشخیص دهیم که کدامیک از کلیدها فشرده شده است. این روش یکی از ساده ترین روشها برای تشخیص کلید فشرده شده است اما دارای معایبی نیز می باشد از جمله اشغال یک پورت از میکرو و پیچیدگی تشخیص کلید فشرده شده.

در این برد PortD.3 را به عنوان پایه ی وقفه در نظر گرفته ایم. همچنین سطر ها به ترتیب از بالا به پایین به PortD.4، PortD.5، PortD.6 و Portd.7 متصل و ستون ها به ترتیب به PortD.0، PortD.1 و PortD.2 متصل می باشد.



مثال ۱۰ :

برنامه ای بنویسید که با فشردن هر کلید، عدد متناظر با آن بر روی LCD نمایش داده شود.

جهت اجرای این برنامه باید ماژول LCD را به پورت C و ماژول KeyPad را به پورت D، Main Bord متصل کرد. بر روی ماژول KeyPad کلید ON/OFF وجود دارد، هنگامی که از مد وقفه استفاده می کنید باید کلید در حالت ON باشد. برای پروگرم کردن میکرو کنترلر با USB پس از ساخت کد Hex در نرم افزارCode Vision و اتصال Main Bord به کامپیوتر، کد Hex مربوطه را در برنامه ی Progisp، Load کرده و بر روی گزینه های Erase و Auto کلیک می نمایید.

```
#include <mega16.h>
#include <delay.h>
#include <stdlib.h>
// Alphanumeric LCD Module functions
#asm
  .equ __lcd_port=0x15 ;PORTC
#endasm
#include <lcd.h>
unsigned char KeyNum;
char s[8];
void main(void)
{
PORTA=0x00;
DDRA=0x00;

PORTB=0x00;
DDRB=0x00;
```

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

PORTD=0x0f;
DDRD=0xf0;

lcd_init(16);
#asm("sei")
while (1)
 {
DDRD=0xf0;
//*****************************
PORTD=0xFF;
PORTD.4=0;
delay_ms(15);

if(PIND.0==0) KeyNum=1;
if(PIND.1==0) KeyNum=2;
if(PIND.2==0) KeyNum=3;

//*******************************
PORTD=0xFF;
PORTD.5=0;
delay_ms(15);
if(PIND.0==0) KeyNum=4;
if(PIND.1==0) KeyNum=5;
if(PIND.2==0) KeyNum=6;


//*****************************
PORTD=0xFF;
PORTD.6=0;
delay_ms(15);
if(PIND.0==0) KeyNum=7;
if(PIND.1==0) KeyNum=8;
if(PIND.2==0) KeyNum=9;

//*****************************
PORTD=0xFF;
PORTD.7=0;
delay_ms(15);
if(PIND.0==0) KeyNum=11;
if(PIND.1==0) KeyNum=0;
if(PIND.2==0) KeyNum=12;
//*****************************
```

```
PORTD=0x0F;

    itoa(KeyNum,s);
    lcd_clear();
    lcd_gotoxy(3,0);
    lcd_puts(s);
    delay_ms(50);
    };
}
```

درروش بالا میکروکنترلرها بیشتر وقت خود را اختصاص به خواندن صفحه کلید کرده است برای رفع این مشکل می توانیم صفحه کلید را با استفاده از وقفه های خارجی کنترل کرد که در این صورت دیگر لازم نیست به طور مداوم صفحه کلید جاروب شود این روش به این صورت می باشد که با فشردن هریک از کلیدها وقفه ای اتفاق می افتد. میکروکنترلر با تشخیص وقفه شروع به رفرش کردن کلیدها می کند.

مثال ۱۱ :

برنامه ای بنویسید با استفاده از interrupt که با فشردن هر کلید، عدد متناظر با آن بر روی LCD نمایش داده شود.

جهت اجرای برنامه باید همانند مثال های بالا عمل نمایید. توجه داشته باشید که در این مثال برای استفاده از مد وقفه باید کلید روی ماژول روشن باشد.

```
#include <mega16.h>
#include <delay.h>
#include <stdlib.h>
#asm
  .equ __lcd_port=0x15 ;PORTC
#endasm
#include <lcd.h>
unsigned char KeyNum;
Char s [8];
// External Interrupt 1 service routine
interrupt [EXT_INT1] void ext_int1_isr(void)
{
DDRD=0xf0;
//*****************************
PORTD=0xFF;
PORTD.4=0;
delay_ms(15);

if(PIND.0==0) KeyNum=1;
if(PIND.1==0) KeyNum=2;
if(PIND.2==0) KeyNum=3;

//*****************************
PORTD=0xFF;
```

```
PORTD.5=0;
delay_ms(15);
if(PIND.0==0) KeyNum=4;
if(PIND.1==0) KeyNum=5;
if(PIND.2==0) KeyNum=6;

//*****************************
PORTD=0xFF;
PORTD.6=0;
delay_ms(15);
if(PIND.0==0) KeyNum=7;
if(PIND.1==0) KeyNum=8;
if(PIND.2==0) KeyNum=9;

//*****************************
PORTD=0xFF;
PORTD.7=0;
delay_ms(15);
if(PIND.0==0) KeyNum=11;
if(PIND.1==0) KeyNum=0;
if(PIND.2==0) KeyNum=12;
//*****************************

PORTD=0x0F;
}

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

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0x00;

PORTD=0x0f;
DDRD=0xf0;

// External Interrupt(s) initialization
// INT0: Off
// INT1: On
// INT1 Mode: Falling Edge
// INT2: Off
GICR|=0x80;
MCUCR=0x08;
```

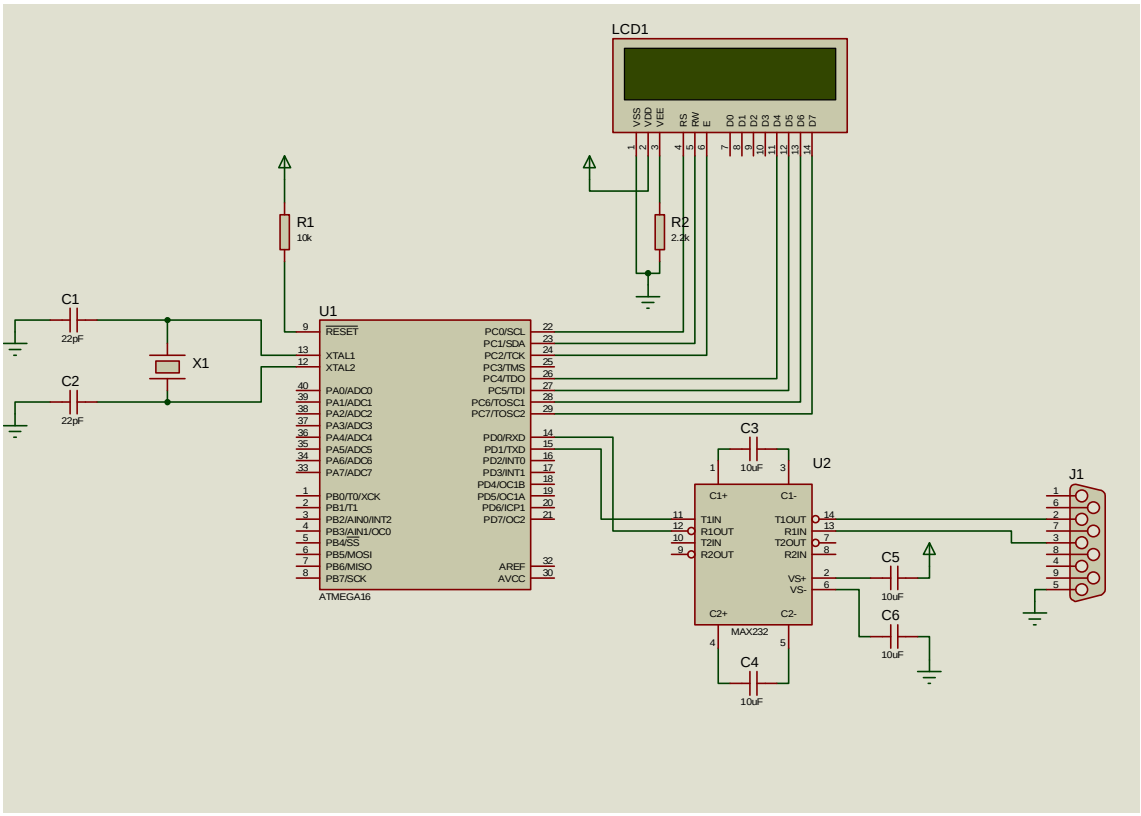
```
MCUCSR=0x00;
GIFR=0x80;
lcd_init(16);

// Global enable interrupts
#asm("sei")
while (1)
    {
    itoa(KeyNum,s);
    lcd_clear();
    lcd_gotoxy(3,0);
    lcd_puts(s);
    delay_ms(300);
    };
}
```

ارتباط سریال با کامپیوتر

برای ایجاد امکان سازگاری تجهیزات مختلف با یکدیگر یک استاندارد واسطی به نام RS232 به پا شده که این استاندارد در PC ها و تجهیزات بسیاری به کار رفته است. سطح منطقی درگاه COM با درگاه سریال میکرو متفاوت است. منطق میکرو با TTL سازگار است ومنطق COM از استاندارد RS232 پیروی می کند.

در RS232 منطق یک با ۳- و منطق صفر با ۳+ تا ۲۵+ ولت تعریف می شود. فاصله ی ۳- تا ۳+ تعریف نشده است. به همین دلیل برای ارتباط میکرو با COM نیاز به یک واسط برای تبدیل از سطح TTL به سطح RS232 و بر عکس می باشد. برای این کار می توان از مبدل های ولتاژی همچون MAX232 یا MAX233 استفاده کرد. این مبدل ها دارای دومجموعه راه انداز برای ارسال و دریافت داده می باشند. که TXD با T1، T2 و RXD با R1، R2 مشخص می شوند.

ساده ترین اتصال بین میکروکنترلر AVR و PC استفاده از سه پایه یRXD ، TXD و GND می باشد.

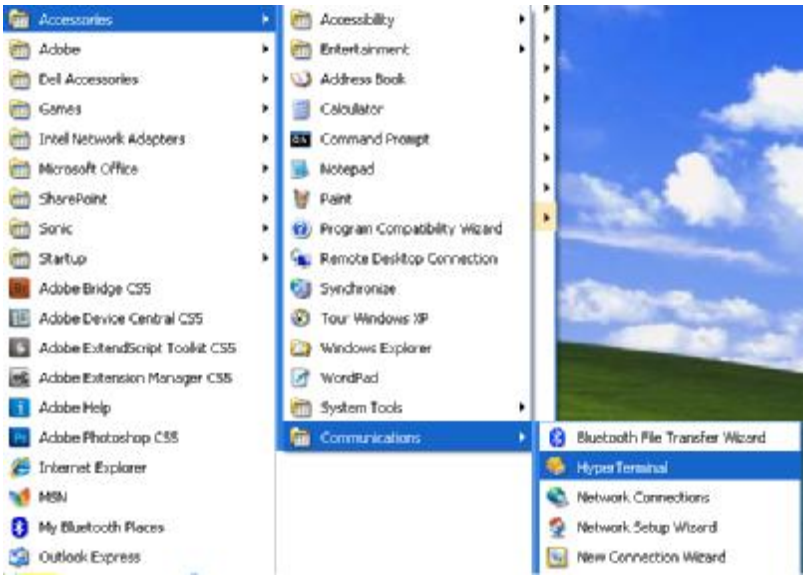


مثال ۱۱ :

برنامه ای بنویسید که با فشردن هر کلید از صفحه کلید کامپیوتر، کاراکتر متناظر با آن کلید را بر روی LCD نمایش داده شود.

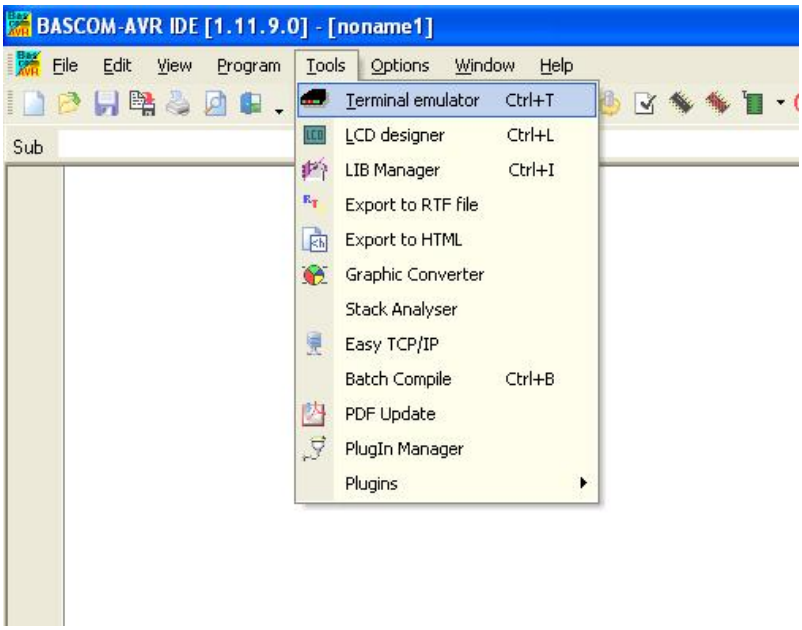
جهت اجرای این برنامه باید ماژول LCD را به پورت C و ماژول RS232 را به پورت D، Main Bord متصل کرد. برای پروگرم کردن میکرو کنترلر با USB پس از ساخت کد Hex در نرم افزارCode Vision و اتصال Main Bord به کامپیوتر، کد Hex مربوطه را در برنامه ی Progisp، Load کرده و بر روی گزینه های Erase و Auto کلیک می نمایید.

برای ارتباط با کامپیوتر از برنامه هایی همچون BASCOM و یا Hyper Terminal می توان استفاده کرد.

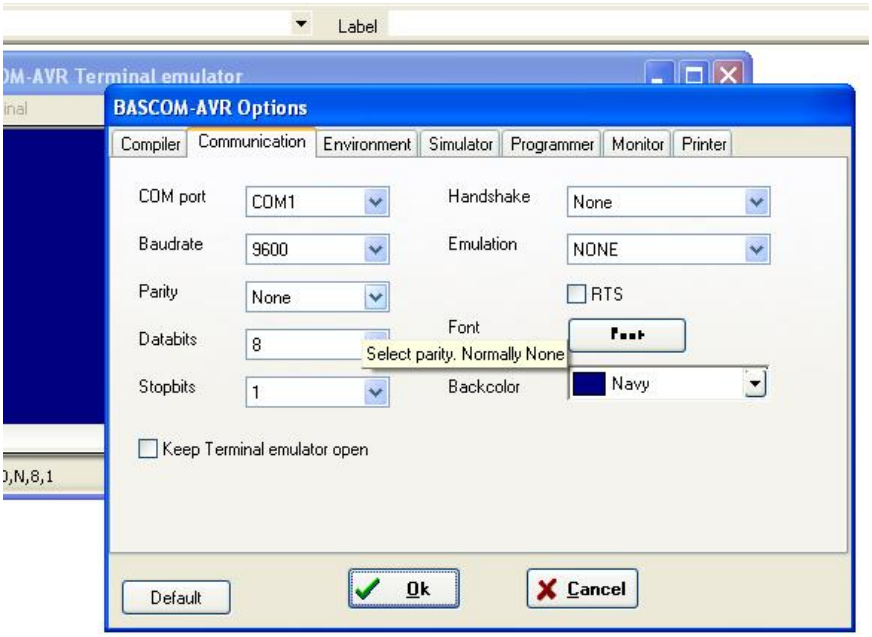


نحوه ی تنظیمات و کار در محیط BASCOM به صورت زیر می باشد:

پس از باز کردن برنامه ی BASCOM از منوی Tools گزینه ی Terminal emulator را انتخاب نمایید.

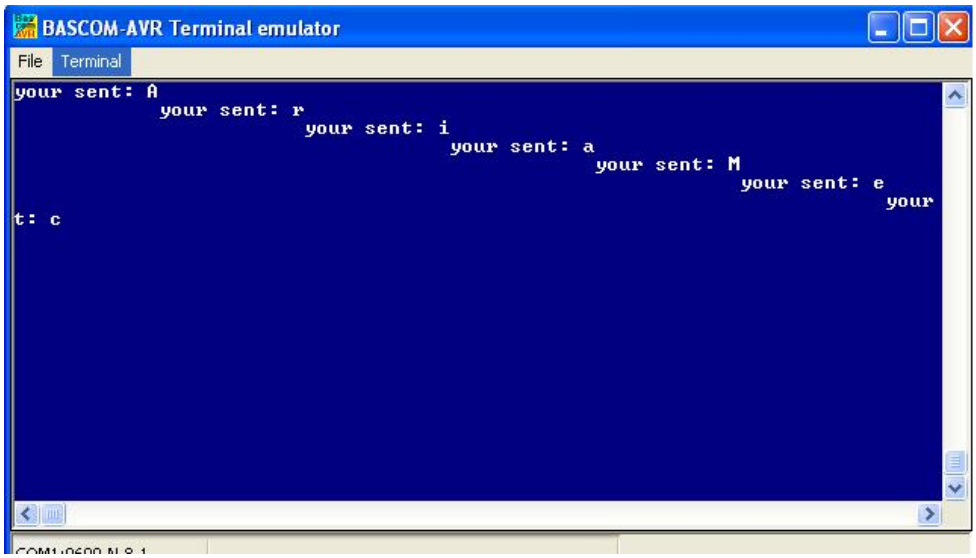


نحوه ی تنظیمات بدین صورت می باشد که میزان Baudrate برابر ۹۶۰۰، بیت توازن (Parity) گزینه ی هیچ (None) ، تعداد بیت داده (Data bits) برابر ۸ ودر نهایت بیت توقف (Stop bits) برابر ۱ می باشد.




```
#include <mega16.h>
#include <delay.h>
#include <stdio.h>
#asm
  .equ __lcd_port=0x15 ;PORTC
#endasm
#include <lcd.h>

#include <stdio.h>
char a;
char str[16];
void main(void)
{
PORTA=0x00;
DDRA=0x00;

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0x00;

PORTD=0x00;
DDRD=0x00;

// USART initialization
```

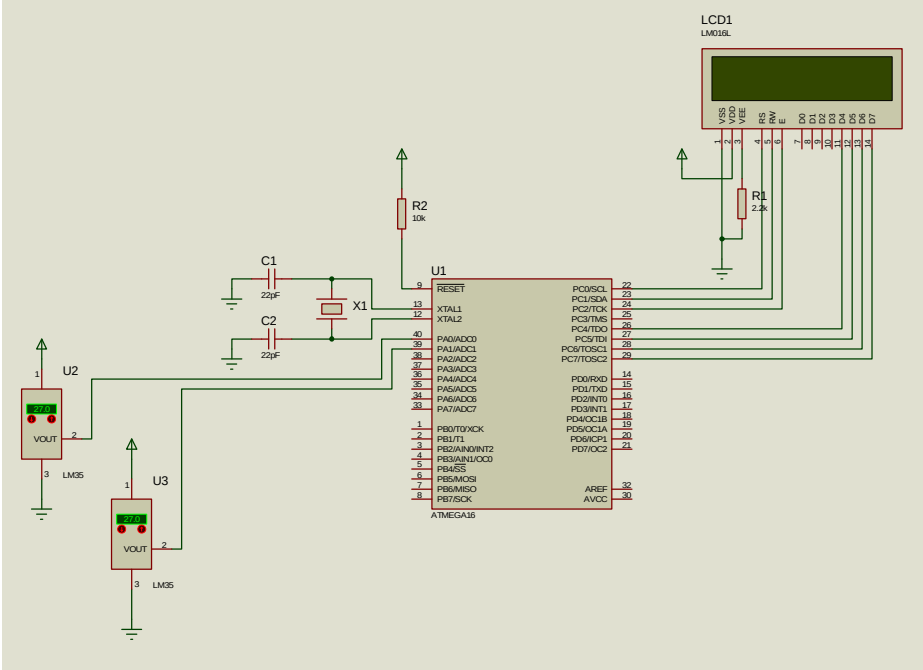
```
// Communication Parameters: 8 Data, 1 Stop, No Parity
// USART Receiver: On
// USART Transmitter: On
// USART Mode: Asynchronous
// USART Baud Rate: 9600
UCSRA=0x00;
UCSRB=0x18;
UCSRC=0x86;
UBRRH=0x00;
UBRRL=0x33;
// LCD module initialization
lcd_init(16);
sprintf(str,"Ready to recieve");
lcd_gotoxy(0,0);
lcd_puts(str);
while (1)
    {
    a=getchar();
    lcd_clear();
    lcd_gotoxy(0,0);
    sprintf(str,"recieve is %c", a);
    lcd_puts(str);
    delay_ms(3000);
    sprintf(str,"your sent: %c", a);
    puts(str);
    lcd_clear();
    sprintf(str,"Ready to recieve");
    lcd_gotoxy(0,0);
    lcd_puts(str);


    };
}
```



سنسور دمای LM35

سنسور دمای LM35 یکی از معروفترین و پرکاربردترین سنسورهای دماست که خروجی این سنسور به ازای هر درجه سانتیگراد ۱۰ میلی ولت افزایش می یابد. به عنوان مثال خروجی سنسور در دمای ۲۷ درجه سانتیگراد ۲۷۰ میلی ولت است این ولتاژ برای آشکارسازی باید به مبدلهای ADC اعمال شود. میکروکنترلرهای AVR دارای مبدل آنالوگ به دیجیتال داخلی هستند. خروجی سنسور LM35را به طور مستقیم میتوان به ورودی ADC میکروکنترلر اعمال کرد و با استفاده از تابع read _ adc () عدد داخل رجیستر ADC را خواند. این عدد به ولتاژ مرجع (Vref) مبدل بستگی دارد.

در برد پایه های خروجی سنسورها به ترتیب به PortA.0، PortA.1، PortA.2، PortA.3 و همچنین پایه های تحریک رله ها به ترتیب به PortA.4، PortA.5، PortA.6 و PortA.7 متصل می باشد.



مثال ۱۲ :

برنامه ای بنویسید که خروجی ADC بر روی LCD نمایش دهد و در صورتی که مقدار آن بیشتر از ۷۰ شود رله را تحریک نماید. جهت اجرای این برنامه باید ماژول LCD را به پورت C و ماژول LM35 را به پورت A، Main Bord متصل کرد. برای پروگرم کردن میکرو کنترلر با USB پس از ساخت کد Hex در نرم افزارCode Vision و اتصال Main Bord به کامپیوتر، کد Hex مربوطه را در برنامه ی Load،Progisp کرده و بر روی گزینه های Erase و Auto کلیک می نمایید.

```
#include <mega16.h>
#include <stdio.h>
#asm
  .equ __lcd_port=0x15 ;PORTC
#endasm
#include <lcd.h>
#include <delay.h>
#define ADC_VREF_TYPE 0xC0

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

```
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}
// Declare your global variables here
void main(void)
{
// Declare your local variables here
int t1;
int t2;
char str1[16];
char str2[16];

PORTA=0x00;
DDRA=0xFC;

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0x00;

PORTD=0x00;
DDRD=0x00;

// ADC initialization
// ADC Clock frequency: 125.000 kHz
// ADC Voltage Reference: AREF pin
// ADC Auto Trigger Source: None
ADMUX=ADC_VREF_TYPE & 0xff;
ADCSRA=0x85;
lcd_init(16);
while (1)
    {
    t1=read_adc(1);
    t2=read_adc(0);
    sprintf(str1,"ADC0 is %i ",t1);
    sprintf(str2,"ADC1 is %i ",t2);
    lcd_clear();
    lcd_gotoxy(2,0);
    lcd_puts(str1);
    lcd_gotoxy(2,1);
    lcd_puts(str2);
    delay_ms(500);
    if(t1>=70)
    {
```

```
    PORTA.4=1;
    }
    else
    {
    PORTA.4=0;
    }
    if(t2>=70)
    {
    PORTA.4=1;
    }
    else
    {
    PORTA.4=0;
    }

    };
}
```

مثال ۱۳ :

برنامه ای بنویسید که دمای محیط را خوانده و بر روی LCD نمایش دهد و در صورتی که دما کمتر از ۲۰ درجه شود با عبارت low، دما بیشتر از ۳۰ درجه شود با عبارت High و دما بین ۲۰ و۳۰ درجه باشد با عبارت Medium نمایش دهد.

جهت اجرای برنامه باید همانند مثال های بالا عمل نمایید.

```
#include <mega16.h>
#include <stdio.h>
#include <delay.h>
// Alphanumeric LCD Module functions
#asm
  .equ __lcd_port=0x15; PORTC
#endasm
#include <lcd.h>
#include <delay.h>
#define ADC_VREF_TYPE 0xC0
unsigned int read_adc(unsigned char adc_input)
{
ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
// Delay needed for the stabilization of the ADC input voltage
delay_us(10);
// Start the AD conversion
ADCSRA|=0x40;
// Wait for the AD conversion to complete
while ((ADCSRA & 0x10)==0);
ADCSRA|=0x10;
return ADCW;
}
```

```
// Declare your global variables here
void main (void)
{
// Declare your local variables here

int t1, t3;
int t2;
char str1[16];

PORTA=0x00;
DDRA=0xFC;

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0x00;

PORTD=0x00;
DDRD=0xff;

MCUCSR=0x00;
// ADC initialization
// ADC Clock frequency: 125.000 kHz
// ADC Voltage Reference: AREF pin
// ADC Auto Trigger Source: None
ADMUX=ADC_VREF_TYPE & 0xff;
ADCSRA=0x85;
// LCD module initialization
lcd_init(16);

while (1)
    {
    t1=read_adc (1);
    t2=read_adc (0);
    t3=t1+t2;
    t3=t3/2;
    t3=t3/2.5;
    sprintf (str1,"temperature: %i ", t3);
    if(t3<20)
    {
    lcd_clear ();
    lcd_gotoxy (1, 0);
    lcd_puts (str1);
    lcd_gotoxy (5, 1);
    lcd_putsf ("Low");
    delay_ms (500);
```

```
    }
    if (t3>30)
    {
    lcd_clear ();
    lcd_gotoxy (1, 0);
    lcd_puts (str1);
    lcd_gotoxy (5, 1);
    lcd_putsf("High");
    delay_ms(500);
    }
    else
    {
    lcd_clear();
    lcd_gotoxy (1, 0);
    lcd_puts(str1);
    lcd_gotoxy (5, 1);
    lcd_putsf("Medium");
    delay_ms(500);
    }
    };
}
```

راه اندازی موتور DC

موتورهای DC دارای مدارات درایو گوناگون و الگوریتم های مختلفی برای کنترل می باشد. یک موتور DC از مجموعه ای از سیم پیچ ها و آهن ربا ها تشکیل شده است که با وصل کردن ولتاژ به دو سر سیم پیچ ها موتور به حرکت در می آید.

موتورهای DC دارای دو ویژگی بسیار مهم هستند:

الف) سرعت موتور به وسیله ی ولتاژ اعمالی به دو سر آن تعیین می شود.

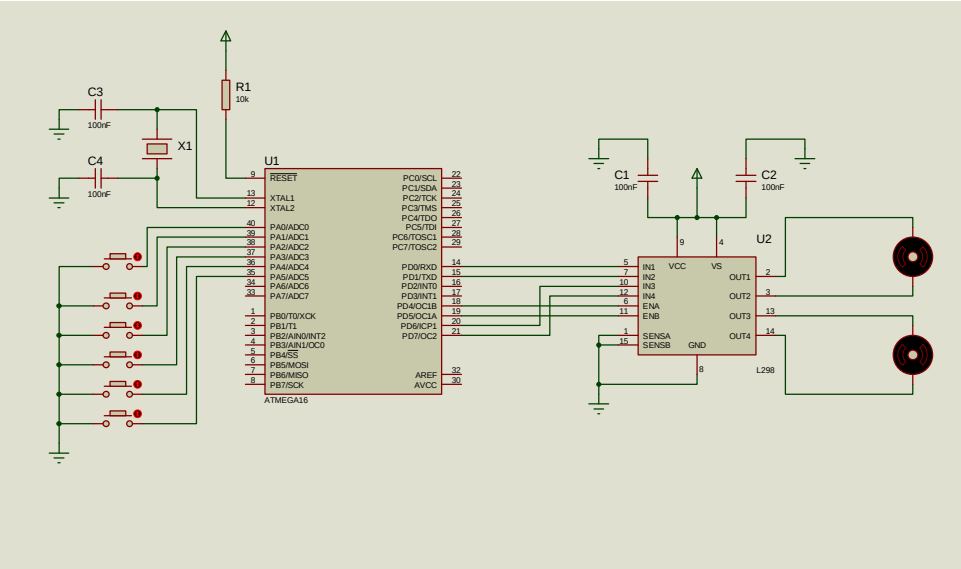
ب) گشتاور موتور به وسیله ی جریانی که از باطری می کشد تعیین می شود.

برای کنترل موتورها آی سی های مختلفی وجود دارد که می توان به درایورهای L298N, L293D, SN154410, LMD18201,... اشاره کرد، در این ماژول از درایورL298 استفاده شده است.

بر روی این برد دو Jumper جهت استفاده از ولتاژ خارجی یا ولتاژ Main Bord در نظر گرفته شده است. ولتاژ خارجی جهت جدا کردن ولتاژ موتورها از ولتاژ میکروکنترلر برای کم کردن اثر نویز می باشد.

برای استفاده از این قابلیت باید ولتاژ خارجی را به پایه های مثبت و منفی PWR(EXT) اعمال و Jumper ، VCC را برداشه و در محل خود قرار دهید.

Jumper دیگر یعنی VCC با ولتاژ پنج ولت میکروکنترلر مشترک می باشد. در مواقعی که دقت مطرح نباشد می توان از این ولتاژ استفاده کرد.



مثال ۱۴ :

برنامه ای بنویسید که با فشردن کلید ۱ موتور ۱ را چپ گرد و با DutyCycle= 80% ودر غیراین صورت با DutyCycle= 50% راستگرد و همچنین با فشردن کلید ۲ موتور ۲ را چپ گرد و با DutyCycle= 90% ودر غیراین صورت با DutyCycle= 20% راستگرد نماید.

جهت اجرای این برنامه باید ماژول Motor DC را به پورت D و ماژول Push Button را به پورت A، Main Bord متصل نماید.

برای پروگرم کردن میکرو کنترلر با USB پس از ساخت کد Hex در نرم افزارCode Vision و اتصال Main Bord به کامپیوتر، کد Hex مربوطه را در برنامه ی Progisp، Load کرده و بر روی گزینه های Erase و Auto کلیک می نمایید.

```
#include <mega16.h>
void main(void)
{
PORTA=0xff;
DDRA=0x00;

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0x00;

PORTD=0x03;
DDRD=0xF3;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 125.000 kHz
// Mode: Fast PWM top=00FFh
// OC1A output: Non-Inv.
```

```
// OC1B output: Non-Inv.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0xA1;
TCCR1B=0x0B;
TCNT1H=0x00;
TCNT1L=0x83;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0xe1;
OCR1BH=0x00;
OCR1BL=0x7d;
while (1)
    {
     if(PINA.0==0)
     {
     OCR1A=0xc8;
     PORTD.6=0;
     PORTD.7=1;
     }
     if(PINA.0==1)
     {
     OCR1A=0x7D;
     PORTD.6=1;
     PORTD.7=0;
     }
     if(PINA.1==1)
     {
     OCR1B=0xE1;
     PORTD.0=1;
     PORTD.1=0;
     }
     if(PINA.1==0)
     {
     OCR1B=0x32;
     PORTD.0=0;
     PORTD.1=1;
     }
    };
}
```

اتصال MMC به میکرو

یکی از مشکلاتی که در برخی از پروژه های الکترونیکی بوجود می آید کم بودن میزان حافظه داخلی می باشد. امروزه استفاده از کارتهای حافظه یا RAM کاربرد فراوانی در موبایل mp3 player، دوربین ها و غیره پیدا کرده است. این کارتها در اندازه ها و ظرفیتهای مختلفی به بازار ارائه شده است. اکثر آنها از پروتکل SPI برای ارتباط دهی با کنترلر استفاده می کنند. با توجه به این که این پروتکل یکی از پروتکل های پشتیبانی شده توسط میکروکنترلر AVR است این ایده را در ما ایجاد می کند که در برخی از پرژه ها که نیاز به حافظه غیرفرار بالایی است از این کارتها استفاده کنیم.

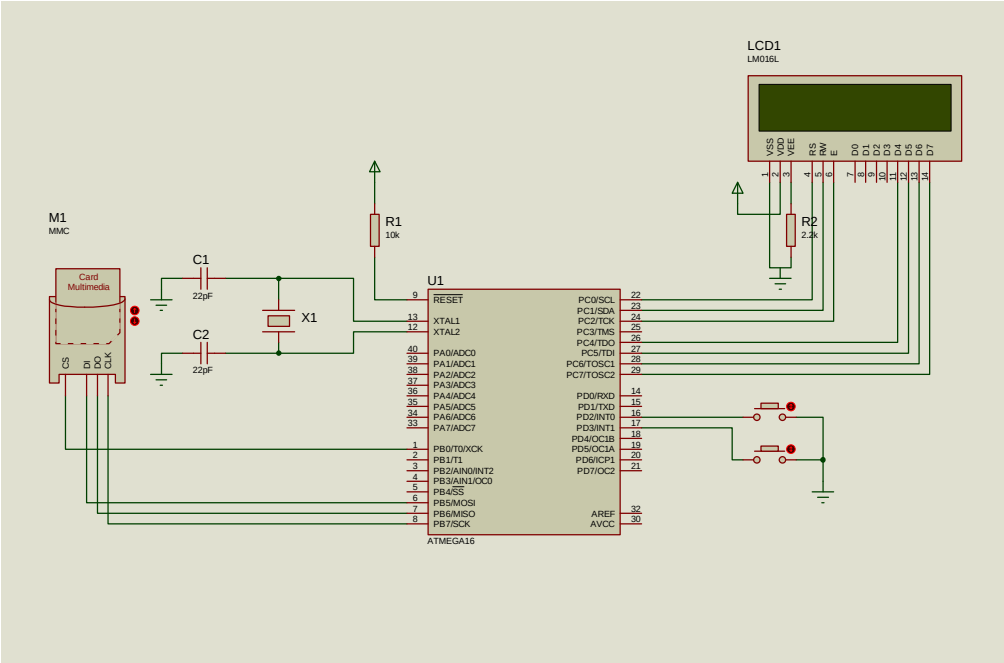
از دیگر قابلیتهای این کارتها در سرعت ارسال داده ها می باشد که قادر است در حدود 20mbs اطلاعات را انتقال دهد. کارتهای فلش در انواع مختلفی از جمله MMC و SD به بازار عرضه شده اند.

یکی از معایب MMC در این است که نحوه خواندن و نوشتن در MMC به صورت sector sector (هر سکتور شامل ۵۱۲ بایت) است. یعنی برای خواندن و نوشتن در MMC می بایست ۵۱۲ بایت را با هم خواند یا نوشت.

نرم افزار:

برای برنامه ریزی MMC می بایست از پروتکل SPI استفاده کرد. که به نظر استفاده از این پروتکل به طور مستقیم کار چندان ساده ای نمی باشد. به همین منظور کتابخانه ای جهت ارتباط با MMC طراحی شده است که شما را قادر می سازد به سادگی اطلاعات خود را در MMC نوشته یا بخوانید.

برد MMC جهت استفاده در دو محیط Code Vision و BASCOM طراحی شده است. جهت استفاده از این برد در محیط Jumper، CodeVision را در محل C و در محیط BASCOM ، Jumper را در محل B قرار دهید.



مثال ۱۵ :

برنامه ای بنویسید که اعداد زوج بین (۰ تا ۲۵۶) را درون سکتور ۲۰ و اعداد فرد (بین ۰ تا ۲۵۶) در سکتور ۵۰ ذخیره سازی کنید.

(با استفاده از کلیدهای even – counter و odd _ counter که به وقفه های خارجی صفر و یک متصل شده است می توان سکتورهای ۲۰ یا ۵۰ را از حافظه MMC خوانده و در داخل آرایه bur_mmc (که یک آرایه ۵۱۲ بایتی) است ذخیره کرد.)
جهت اجرای این برنامه باید ماژول MMC را به پورت B و ماژول LCD را به پورت C، Main Bord متصل کرد. برای پروگرم کردن میکرو کنترلر با USB پس از ساخت کد Hex در نرم افزارCode Vision و اتصال Main Bord به کامپیوتر، کد Hex مربوطه را در برنامه ی Progisp، Load کرده و بر روی گزینه های Erase و Auto کلیک می نمایید.

```
#include <mega16.h>
#include <delay.h>
#include <stdio.h>
#include <spi.h>
#include <mmc.h>
#asm
  .equ __lcd_port=0x15 ;PORTC
#endasm
#include <lcd.h>
unsigned char buf_mmc[512];
unsigned long sector_num;
unsigned int i;
unsigned char str[10];

void buf_clear(void);
void write_sector(void);

// External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{
    buf_clear();
    delay_ms(1000);
    mmc_read(20,buf_mmc);
    i=0;
}
// External Interrupt 1 service routine
interrupt [EXT_INT1] void ext_int1_isr(void)
{

    buf_clear();
    delay_ms(1000);
    mmc_read(50,buf_mmc);
    i=0;
}
void main(void)
{
PORTD=0xFF;
DDRB=0xB1;
// Analog Comparator: Off
ACSR=0x80;
```

```
// External Interrupt(s) initialization
// INT0: On
// INT0 Mode: Falling Edge
// INT1: On
// INT1 Mode: Falling Edge
// INT2: Off
GICR|=0xC0;
MCUCR=0x0A;
MCUCSR=0x00;
GIFR=0xC0;
// SPI initialization
// SPI Type: Master
// SPI Clock Rate: 2000.000 kHz
// SPI Clock Phase: Cycle Half
// SPI Clock Polarity: Low
// SPI Data Order: MSB First
SPCR=0x50;
SPSR=0x00;
mmc_init();
lcd_init(16);
write_sector();
// Global enable interrupts
#asm("sei")
while (1)
    {
    for(i=0;i<256;i++)
    {
    sprintf(str,">> %d",buf_mmc[i]);
    lcd_clear();
    lcd_puts(str);
    delay_ms(500);
    }
    };
}

void buf_clear(void){
for (i=0;i<512;i++)
  buf_mmc[i] = 0;
}
void write_sector(void){
for (i=0;i<256;i+=2)
{
buf_mmc[i/2]=i;
}
sector_num = 20;
mmc_write(sector_num, buf_mmc);
```

```
for (i=0;i<256;i+=2)
{
buf_mmc[i/2]=i+1;
}
sector_num = 50;
mmc_write(sector_num, buf_mmc);
}
```

راه اندازی موتورهای پله ای

از موتورهای پله ای می توان 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. موتور پله ای وسیله ی پرمصرفی است که پالس های الکتریکی را به حرکت مکانیکی تبدیل می کند و در بسیاری از وسایل از جمله دیسک ها، چاپگرهای ماتریسی و رباتیک کاربرد دارد.
نحوه ی عملکرد یک موتور پله ای تفاوت زیادی با یک موتور DC ندارد تنها تفاوت این دو موتور در نحوه ی حرکت محور است. یک موتور پله ای متداول دارای شش سیم می باشد که چهار سیم برای چهار سیم پیچ استاتور و دو سر مشترک برای سیم های وسط می باشد. با اعمال پالس هایی به هر یک از سیم پیچ ها موتور شروع به حرکت می نماید. برای حرکت موتور پله ای باید پالس هایی مانند جدول زیر به صورت متداول به هر یک از سیم پیچ ها اعمال شود.

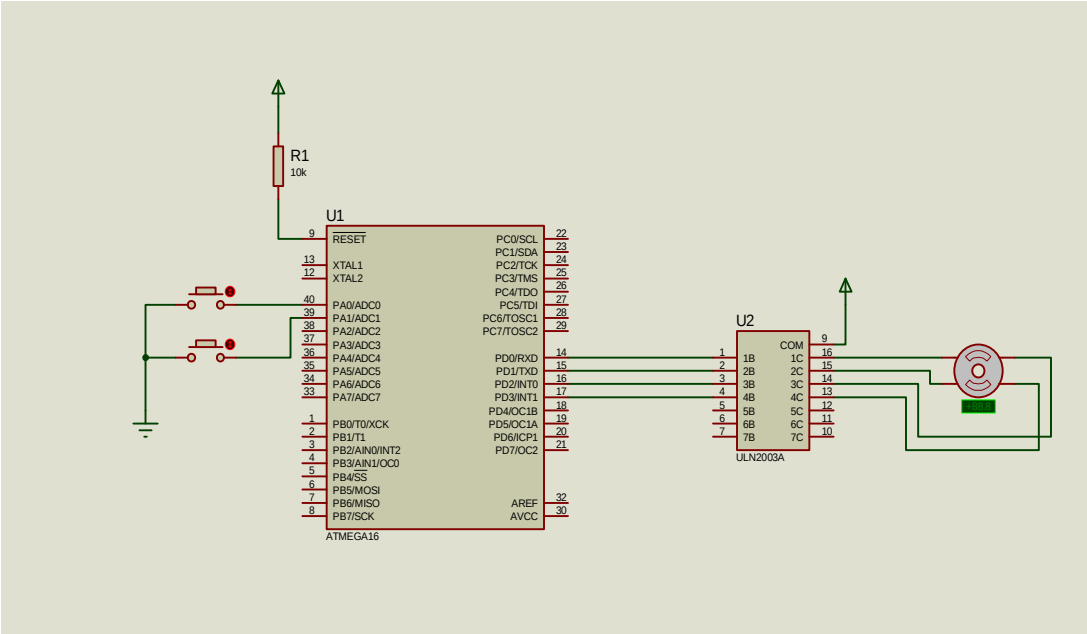
جهت موتور	D	C	B	A
در جهت عقربه های ساعت	0	0	0	1
	0	0	1	0
	0	1	0	0
	1	0	0	0

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

زاویه ی پله:
هنگامی که پالس به یکی از سیم پیچ ها اعمال شود، موتور به اندازه ی یک پله حرکت می کند. زاویه ی پله حداقل زاویه ای از چرخش مربوط به یک پله است که در موتورهای مختلف بین محدوده ی ۰/۷۲ درجه تا ۹۰ درجه می باشد و متداول ترین زاویه ی پله ۱/۸ درجه است. جدول زیر تعداد پالس هایی را که باید به هر یک از موتورها با زاویه پله ی مشخص داده شود را تعیین می کند. به عنوان مثال تعداد پالس هایی را که باید به یک موتور با زاویه پله ۱/۸ داده شود تا یک دور کامل بچرخد برابر ۲۰۰ می باشد.

تعداد پله در یک دور	زاوه ی پله
۵۰۰	۰/۷۲
۲۰۰	۱/۸

۴۸	۷/۵
۲۴	۱۵
۴	۹۰



مثال ۱۶ :

برنامه ای بنویسید که یک موتور پله ای با زاویه پله ۱/۸، ده دور چپ گرد و ده دور راست گرد سپس بچرخد.

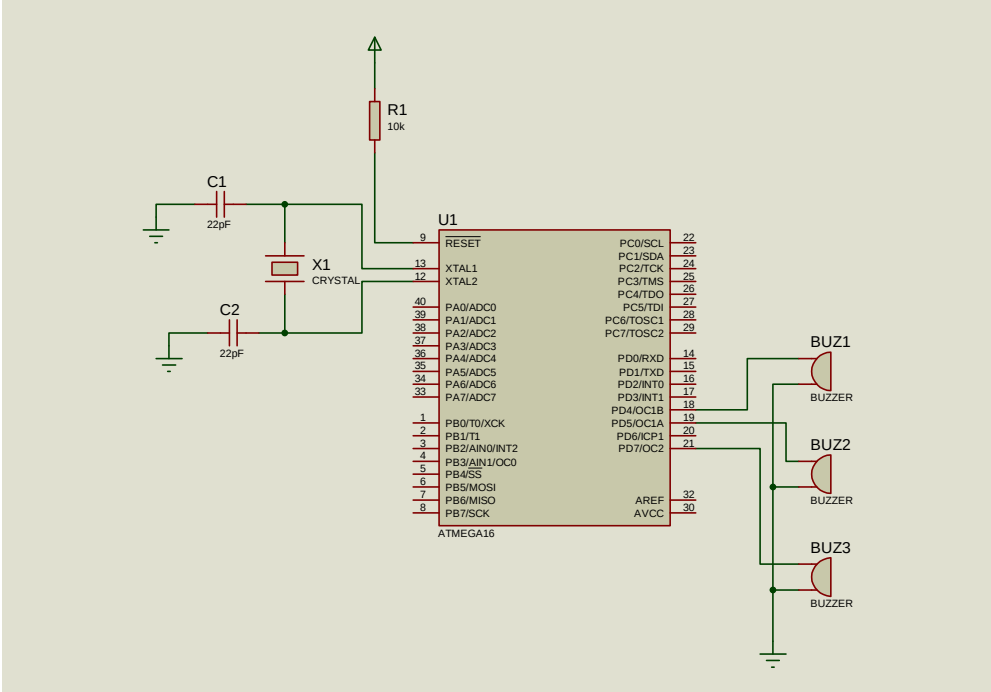
جهت اجرای این برنامه باید ماژول Step Motor را به پورت D، Main Bord متصل نمایید. برای پروگرم کردن میکرو کنترلر با USB پس از ساخت کد Hex در نرم افزار Code Vision و اتصال Main Bord به کامپیوتر، کد Hex مربوطه را در برنامه ی Progisp، Load کرده و بر روی گزینه های Erase و Auto کلیک می نمایید.

```
#include <mega16.h>
#include <delay.h>
int i;
void main(void)
{
PORTA=0xff;
DDRA=0x00;

PORTB=0x00;
DDRB=0x00;

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

```
PORTD=0x03;
DDRD=0xFF;

while(1)
{
  for(i=0;i<=500;i++)
  {
  PORTD=0b00000001;
  delay_ms(5);
  PORTD=0b00000010;
  delay_ms(5);
  PORTD=0b00000100;
  delay_ms(5);
  PORTD=0b00001000;
  delay_ms(5);
   }
   for(i=0;i<=500;i++)
  {
  PORTD=0b00001000;
  delay_ms(5);
  PORTD=0b00000100;
  delay_ms(5);
  PORTD=0b00000010;
  delay_ms(5);
  PORTD=0b00000001;
  delay_ms(5);
   }
   break;
  };
}
```

BUZZER

بیزر یا همان بلندگوهای کوچک در دو نوع ساخته می شود، نوع اول به عنوان زنگ و نوع دوم به عنوان بلندگوی پیزوالکتریک شناخته می شود.



مثال۱۷ :

برنامه ای بنویسید که با هر یک از کلیدهای برد Push Button صدای متفاوتی ایجاد شود.

```
#include <mega16.h>
void main(void)
{

PORTA=0xFF;
DDRA=0x00;

PORTB=0x00;
DDRB=0x00;

PORTC=0x00;
DDRC=0x00;

PORTD=0x00;
DDRD=0xB0;

// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
```

```
OCR0=0x00;

// Timer/Counter 1 initialization
// Clock source: System Clock
// Clock value: 125.000 kHz
// Mode: Fast PWM top=00FFh
// OC1A output: Non-Inv.
// OC1B output: Non-Inv.
// Noise Canceler: Off
// Input Capture on Falling Edge
// Timer 1 Overflow Interrupt: Off
// Input Capture Interrupt: Off
// Compare A Match Interrupt: Off
// Compare B Match Interrupt: Off
TCCR1A=0xA1;
TCCR1B=0x0B;
TCNT1H=0x00;
TCNT1L=0x06;
ICR1H=0x00;
ICR1L=0x00;
OCR1AH=0x00;
OCR1AL=0x00;
OCR1BH=0x00;
OCR1BL=0x00;

// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: 125.000 kHz
// Mode: Fast PWM top=FFh
// OC2 output: Non-Inverted PWM
ASSR=0x00;
TCCR2=0x6C;
TCNT2=0x06;
OCR2=0x00;

while (1)
    {

     if(PINA.0==0)
      {
      OCR1A=0x0C;
      OCR1B=0x0C;
      OCR2=0x0C;
      }
     if(PINA.1==0)
      {
      OCR1A=0x4E;
```

```
        OCR1B=0x4E;
        OCR2=0x4E;
        }
       if(PINA.2==0)
        {
        OCR1A=0x4B;
        OCR1B=0x4B;
        OCR2=0x4B;
        }
       if(PINA.3==0)
        {
        OCR1A=0x70;
        OCR1B=0x70;
        OCR2=0x70;
        }
       if(PINA.4==0)
        {
        OCR1A=0x96;
        OCR1B=0x96;
         OCR2=0x96;
        }
       if(PINA.5==0)
        {
        OCR1A=0xBB;
        OCR1B=0xBB;
        OCR2=0xBB;
        }
       if(PINA.6==0)
        {
        OCR1A=0xE1;
        OCR1B=0xE1;
        OCR2=0xE1;
        }
       if(PINA.7==0)
        {
        OCR1A=0xFA;
        OCR1B=0xFA;
        OCR2=0xFA;
        }

      };
}
```