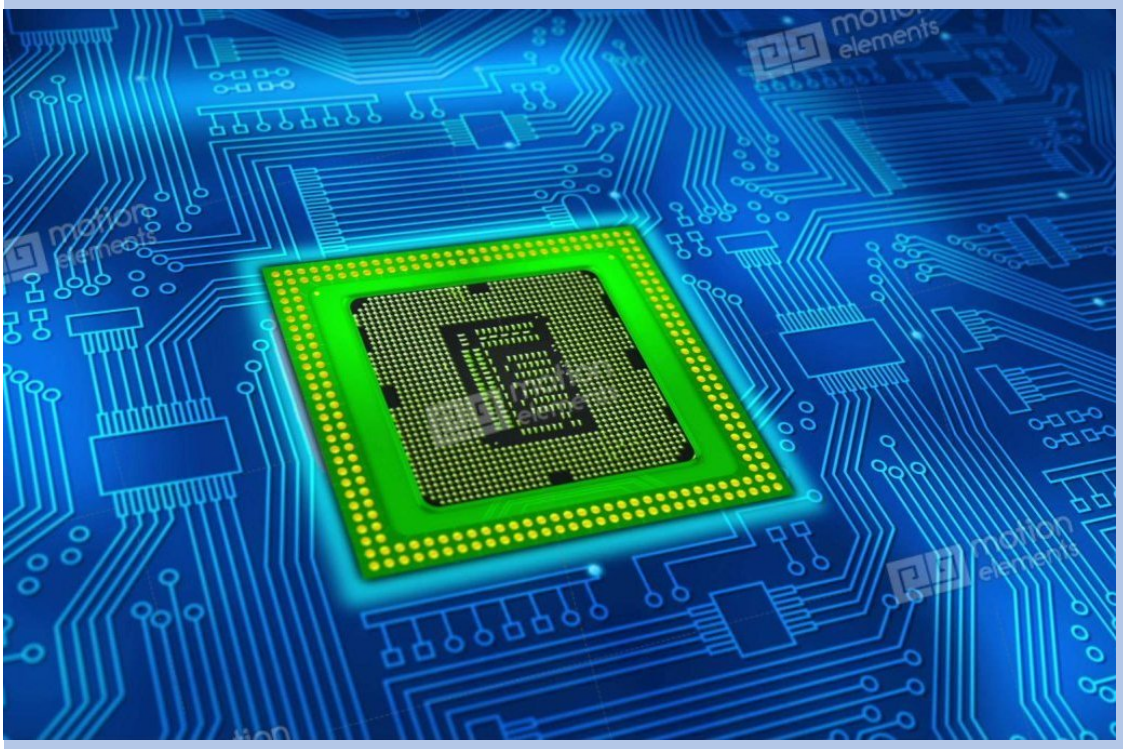
پاییز ۹۸

آزمایشگاه

ریزپردازنده‌ها



گردآوری و تنظیم: محمدعلی شفیعیان

قابل استفاده برای دانشجویان رشته‌های برق و کامپیوتر

وزارت علوم، تحقیقات و فناوری
دانشگاه جهرم
دانشکده فنی و مهندسی

آزمایشگاه ریزپردازنده‌ها

گردآورندی و تنظیم:

محمدعلی شفیعیان

عملکرد خوب به‌واسطه کسب تجربه حاصل می‌شود.
تجربه در پی عملکرد ناموفق به‌دست می‌آید.
ـ یک ضرب‌المثل ـ

پیشگفتار

امروزه با پیشرفت روزافزون علم الکترونیک و دیجیتال، استفاده از میکروکنترلرها در ساخت تجهیزات الکترونیکی مانند تجهیزات صنعتی، پزشکی، اتوماسیون و ... با رشد بسیار بالایی همراه بوده است. در زمینه طراحی و تولید میکروکنترلرها شرکت‌های بسیاری اقدام نموده‌اند. از جمله این شرکت‌ها می‌توان به دو شرکت پیشرو در این زمینه همچون ATMEL و Microchip اشاره نمود. پیشرفت میکروکنترلرها در کنار سایر تجهیزات الکترونیکی موجب می‌شود تا مهندسین با توجه به نیاز صنعت با میکروکنترلرهای مختلف نظیر ARM و AVR شرکت ATMEL و میکروکنترلرهای PIC از شرکت Microchip و سایر میکروکنترلرها کار نمایند. میکروکنترلرها تراشه‌هایی هستند که توسط یک نرم‌افزار به یکی از زبان‌های C، بیسک، پاسکال یا اسمبلی برنامه‌نویسی می‌شوند و سپس برنامه نوشته‌شده توسط ابزاری به نام پروگرامر به میکروکنترلر منتقل می‌شود. میکروکنترلرهای AVR می‌توانند تا ۱۰۰۰۰ مرتبه پاک شده و مجدداً برنامه‌ریزی شوند.

در این مجموعه، مباحث مهمی از میکروکنترلر AVR که دانشجو در درس ریزپردازنده‌ها فرا گرفته است در قالب آزمایش به‌صورت عملی مورد شبیه‌سازی قرار گرفته و پیاده‌سازی می‌شود و برای دانشجویان رشته‌های برق و کامپیوتر قابل استفاده است. آزمایش‌ها به‌گونه‌ای طراحی شده‌اند که سرفصل‌های مهم درس را پوشش دهند. در آزمایش‌های در نظر گرفته شده عمدتاً از میکروکنترلرهای ATmega32 و ATmega16 استفاده گردیده است. امید است که مجموعه گردآوری شده بتواند دانشجویان را با مفاهیم عملی میکروکنترلرها آشنا سازد.

در پایان شایسته است از تمام اساتید، دانشجویان و دوستان به‌ویژه جناب آقای مهندس محمدنبی بهمن‌زادگان جهرمی که بنده را در تهیه این مجموعه آموزشی یاری نمودند نهایت سپاس‌گزاری را داشته باشم.

محمدعلی شفیعیان
پاییز ۱۳۹۸

فهرست مطالب

آزمایش ۱ ــ پیاده‌سازی فلاشر

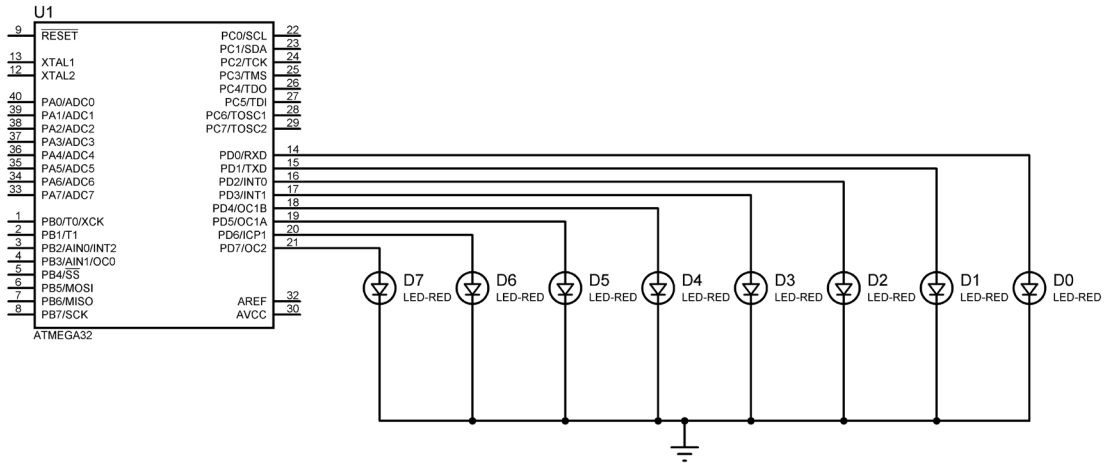
۱-۱ آزمایش اول

برنامه زیر را در محیط نرم‌افزار CodeVisionAVR نوشته و پس از بررسی درستی برنامه نوشته شده، آن را کامپایل کرده و در محیط نرم‌افزار ISIS Proteus مدار نشان داده شده در شکل ۱-۱ را شبیه‌سازی نمایید. سپس به‌کمک Programmer برنامه خود را به IC میکروکنترلر AVR منتقل کرده و با استفاده از مجموعه آموزشی موجود در آزمایشگاه، مدار نشان داده شده در شکل ۱-۱ را پیاده‌سازی کنید.

```
#include <mega32.h>
#include <delay.h>
#define xtal 8000000

int i;
void main (void)
{
    DDRD = 0xFF;
    while(1)
    {
        for(i = 1; i <= 128; i = i*2)
        {
            PORTD = i;
            delay_ms(100);
        }

        for(i = 64; i > 1; i = i/2)
        {
            PORTD = i;
            delay_ms(100);
        }
    }
}
```



شکل ۱-۱: شمای مداری آزمایش اول

۱-۲ آزمایش دوم

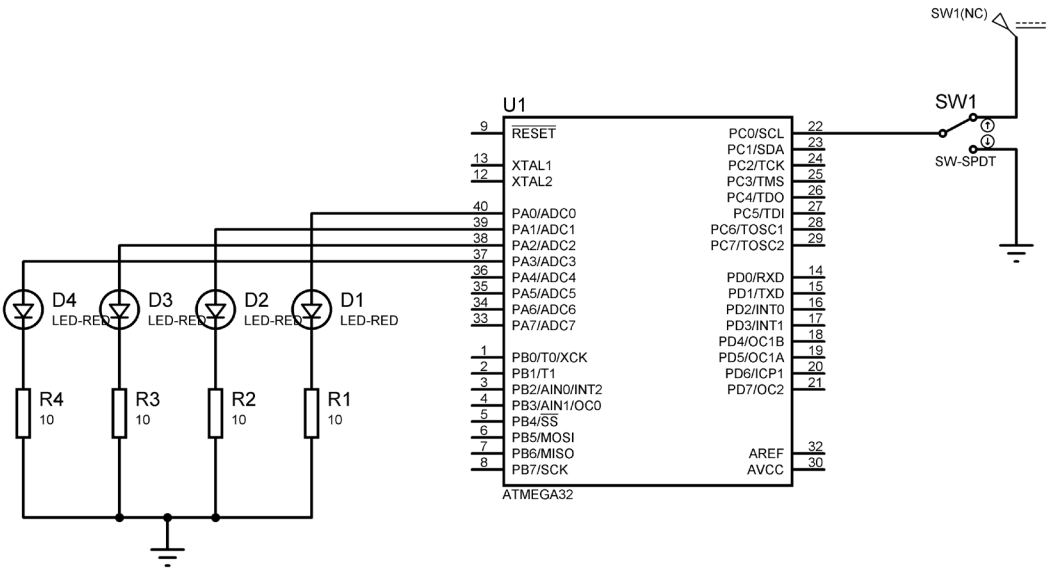
برنامه‌ای بنویسید که در مدار شکل ۱-۱، LEDها با الگوی نشان داده شده در جدول ۱-۱، روشن و خاموش شوند. مدت زمان تأخیر بین هر دو لحظه متوالی را ۱۰۰ میلی ثانیه در نظر بگیرید. برنامه را به گونه‌ای بنویسید که این عمل تا بینهایت تکرار شود. برنامه خود را با استفاده از یکی از دستورالعمل‌های حلقه که فکر می کنید مناسب باشد، بنویسید.

جدول ۱-۱: الگوی روشن و خاموش شدن LEDها

	D0	D1	D2	D3	D4	D5	D6	D7
لحظه ۱	**ON**	OFF	OFF	OFF	OFF	OFF	OFF	**ON**
لحظه ۲	OFF	**ON**	OFF	OFF	OFF	OFF	**ON**	OFF
لحظه ۳	OFF	OFF	**ON**	OFF	OFF	**ON**	OFF	OFF
لحظه ۴	OFF	OFF	OFF	**ON**	**ON**	OFF	OFF	OFF
لحظه ۵	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF
لحظه ۶	OFF	OFF	OFF	**ON**	**ON**	OFF	OFF	OFF
لحظه ۷	OFF	OFF	**ON**	OFF	OFF	**ON**	OFF	OFF
لحظه ۸	OFF	**ON**	OFF	OFF	OFF	OFF	**ON**	OFF
لحظه ۹	**ON**	OFF	OFF	OFF	OFF	OFF	OFF	**ON**

۱-۳ پرسش

۱ـ برنامه‌ای بنویسید که در مدار شکل ۱-۲، اگر کلید متصل به PortC.0 در وضعیت یک منطقی (متصل به ۵ ولت باشد)، آنگاه معادل باینری اعداد ۰ تا ۱۵ به صورت شمارش از صفر تا ۱۵ بر روی LEDهای متصل به PORT A ارسال شود و اگر در وضعیت صفر منطقی (متصل به زمین) باشد، عملی انجام نشود. این برنامه را با استفاده از حلقه‌ها بنویسید و آن را به‌گونه‌ای بنویسید که عمل شمارش تا بینهایت تکرار شود. همچنین زمان تأخیر بین هر دو شمارش متوالی را ۱۰۰ میلی‌ثانیه در نظر بگیرید.



شکل ۱-۲: شمای مداری مربوط به پرسش ۱

آزمایش ۲ ــ چگونگی استفاده از 7-Segment و ارتباط آن با میکروکنترلر

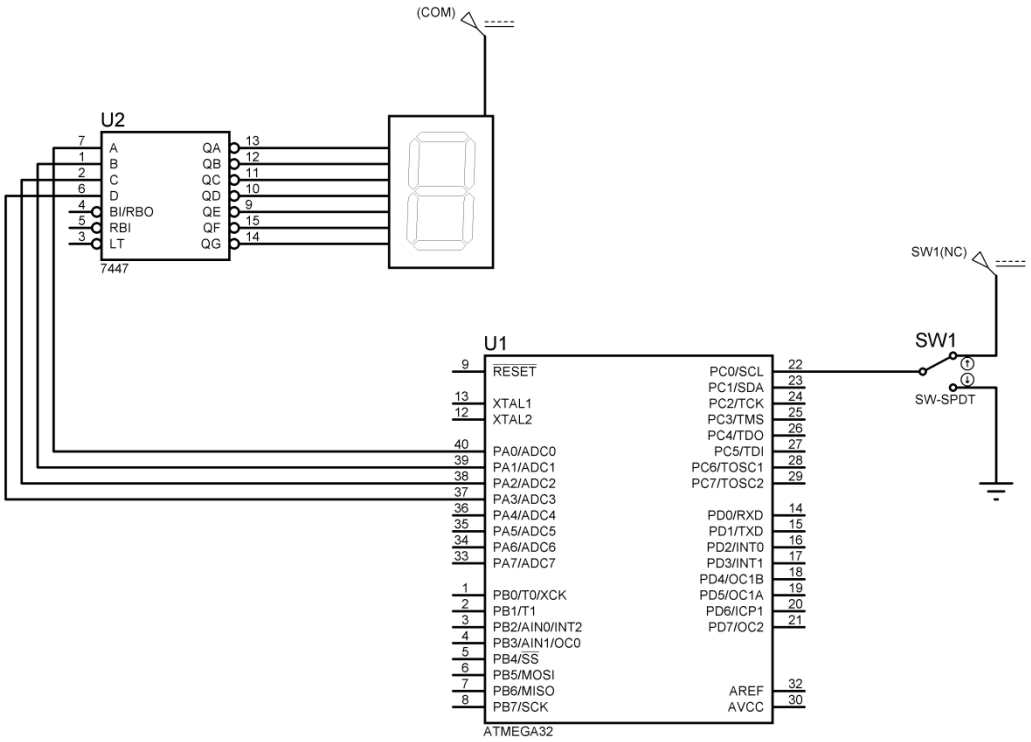
۲-۱ آزمایش اول

برنامه ای بنویسید که وضعیت پایه صفر از Port C را بررسی کند و در صورت یک بودن آن، شمارش را صفر تا ۹ و در صورت صفر بودن، شمارش را از ۹ تا صفر انجام دهد. سپس حاصل را بر روی یک 7-Segment که توسط یک IC دیکدر ۷۴۴۷ به Port A متصل شده است، نمایش دهد.

توجه : برای شبیه سازی از یک 7-Segmant آند مشترک استفاده کنید.

توجه : IC دیکدر ۷۴۴۷ یک دیکدر BCD به 7-Segment است. در این IC، عدد BCD مورد نظر به ورودی های A، B، C و D اعمال می شود که در آن A بیت کم ارزش است و خروجی متناظر با آن در پایه های a تا g قرار می گیرد. در عمل، پایه های خروجی توسط مقاومت های ۱۵۰ Ω به 7-Segment متصل می شوند.

شمای مداری این بخش از آزمایش در شکل ۲-۱ نشان داده شده است.



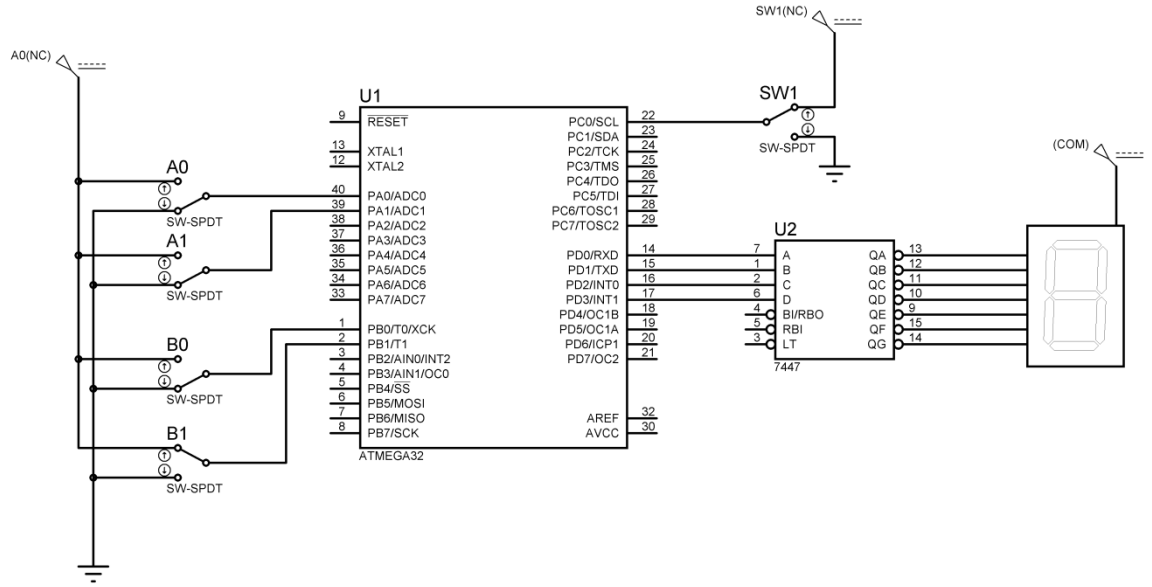
شکل ۲-۱: شمای مداری آزمایش اول

۲-۲ آزمایش دوم

برنامه ای بنویسید که داده ورودی به پایه های صفر و یک از Port A و داده ورودی به پایه های صفر و یک از Port B را برحسب وضعیت پایه صفر از Port C با یکدیگر جمع یا در هم ضرب نماید؛ به این صورت که اگر پایه صفر از Port C یک باشد دو عدد را با هم جمع و اگر صفر باشد دو عدد را در هم ضرب کند. بدیهی است که دو عدد ورودی دو بیتی بوده و بیت کم ارزش آنها به پایه های صفر از Port B و Port C متصل است. سپس نتیجه عمل جمع یا ضرب بر روی یک 7-Segment که توسط یک IC دیکدر ۷۴۴۷ به Port D متصل شده است، نمایش داده شود.

توجه : برنامه را به گونه ای بنویسید که اگر در هر لحظه از اجرای شبیه سازی وضعیت Port C.0 یا اعداد ورودی به Port A و Port B تغییر کند، متناسب با آن، خروجی نمایش داده شده نیز تغییر کند.

شمای مداری این بخش از آزمایش در شکل ۲-۲ نشان داده شده است.

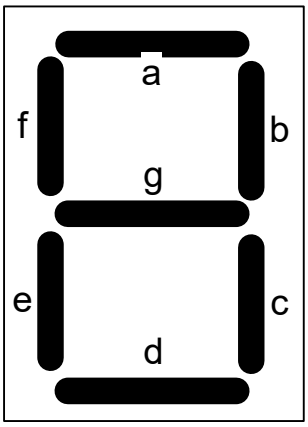


شکل ۲-۲: شمای مداری آزمایش دوم

** **نکته مهم:** در این آزمایش از دیکدر ۷۴۴۷ برای تبدیل عدد BCD به معادل 7-Segment آند مشترک استفاده شده است. اما ممکن است حالاتی رخ دهند که در آن‌ها نتوانید از دیکدر استفاده نمایید. در این صورت بایستی اعداد معادل 7-Segment را بر روی پورت میکروکنترلر قرار دهید. در جدول ۲-۱ و جدول ۲-۲ رشته اعدادی که بایستی بر روی پورت میکروکنترلر قرار دهید تا ارقام BCD نمایش داده شود آورده شده است. از آنجایی‌که دستگاه مورد استفاده در آزمایشگاه فاقد دیکدر است، برای پیاده‌سازی مدار خود باید 7-Segment را مستقیماً به پورت میکروکنترلر متصل کنید. بنابراین برنامه‌های هر دو آزمایش را به‌گونه‌ای بنویسید که در آن‌ها از دیکدر استفاده نشود و از جدول‌های بیان‌شده نیز کمک بگیرید.

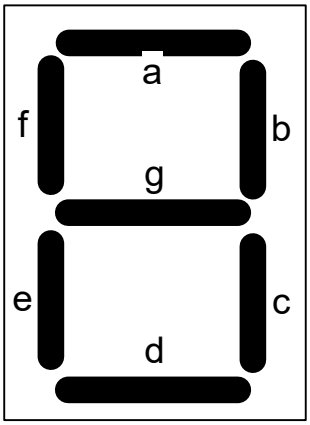
جدول ۲-۱: اعداد برای 7-Segment آند مشترک

عدد نمایش‌داده‌شده	g	f	e	d	c	b	a
0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	1
2	0	1	0	0	1	0	0
3	0	1	1	0	0	0	0
4	0	0	1	1	0	0	1
5	0	0	1	0	0	1	0
6	0	0	0	0	0	1	0
7	1	1	1	1	0	0	0
8	0	0	0	0	0	0	0
9	0	0	1	0	0	0	0



جدول ۲-۲: اعداد برای 7-Segment کاتد مشترک

عدد نمایش‌داده‌شده	g	f	e	d	c	b	a
0	0	1	1	1	1	1	1
1	0	0	0	0	1	1	0
2	1	0	1	1	0	1	1
3	1	0	0	1	1	1	1
4	1	1	0	0	1	1	0
5	1	1	0	1	1	0	1
6	1	1	1	1	1	0	1
7	0	0	0	0	1	1	1
8	1	1	1	1	1	1	1
9	1	1	0	1	1	1	1



۲-۳ پرسش

۱ـ با توجه به آزمایش ۲-۲ برنامه ای بنویسید که با توجه به وضعیت Port C.0 دو عدد ۴ بیتی ورودی به Port A و Port B را با هم جمع یا در هم ضرب نماید و نتیجه را (که ممکن است عددی دو رقمی باشد) بر روی دو عدد 7-Segment متصل به Port D نمایش دهد. همچنین اگر حاصل ضرب دو عدد بزرگتر از ۹۹ بود، بر روی دو 7-Segment عدد یا حرفی نمایش داده نشود.

آزمایش ۳ ــ اتصال LCD به میکروکنترلرهای AVR

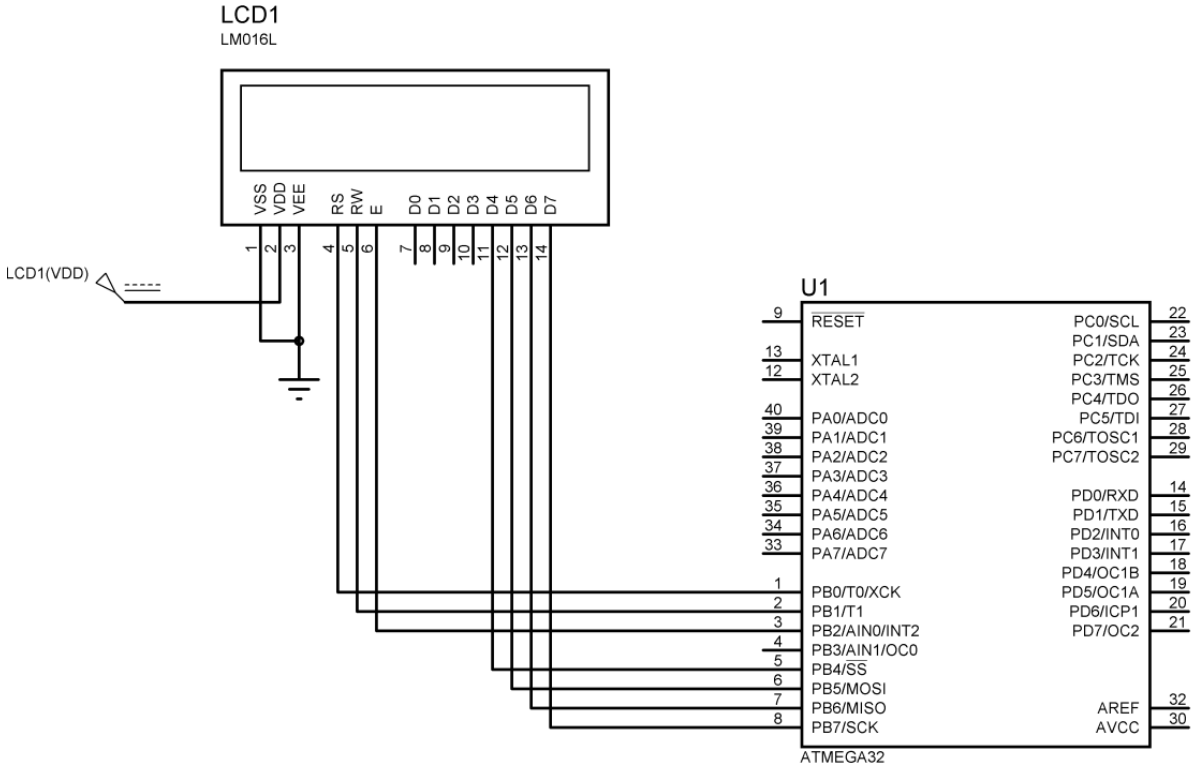
۳-۱ مقدمه

در سال های اخیر، برای نمایش اطلاعات میکروکنترلرها بیشتر از LCD استفاده می شود. دلیل آن، بهای پایین و امکان نمایش حروف، رقم و کاراکترهای گرافیکی است.

اطلاعات بر روی LCD ممکن است در یک، دو یا چهار سطر نوشته شود. این اطلاعات را می توان به صورت ۸ بیتی یا ۴ بیتی برای LCD ارسال نمود و با فرمان هایی که برای آن پیش بینی شده است، در هر لحظه می‌توان آن را پاک نمود، محل نمایش اطلاعات را تغییر داد، حروف را از طرف راست به چپ یا برعکس نوشت و بالأخره حروف یا مکان نما را روشن یا خاموش نمود.

۳-۲ آزمایش اول

در این آزمایش می خواهیم با استفاده از میکروکنترلر ATmega32 برنامه ای بنویسیم که توسط آن، پیغام Microcontroller Course بر روی یک LCD نمایش داده شود. شمای مداری این آزمایش در شکل ۳-۱ نشان داده شده است.



شکل ۳-۱: شمای مداری آزمایش اول

برای یادآوری و آشنایی بیشتر با دستورات LCD، برنامه این آزمایش در ادامه آورده شده است :

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

#asm
.equ __lcd_port=0x18;PORTB
#endasm
#include <lcd.h>

void main (void){
        PORTB = 0x00;
        DDRB = 0x00;

        lcd_init(16);
        lcd_clear();
        lcd_gotoxy(0,0);
        lcd_putsf("microcontroller”);
        lcd_gotoxy(5,1);
        lcd_putsf("course”);
}
```

این برنامه را به دقت بررسی نموده و آزمایش را انجام دهید.

۳-۳ آزمایش دوم

در همان مدار نشان داده شده در شکل ۳-۱برنامه ای بنویسید که رشته Microcontroller Course را به طور متناوب و به صورت حرف به حرف و با تأخیر زمانی اندکی بین نمایش حروف، بر روی LCD نمایش داده دهد.

۳-۴ آزمایش سوم

برنامه ای بنویسید که در هر لحظه، داده متصل به PortD را بخواند و بر روی LCD نمایش دهد.

۳-۵ پرسش

۱ـ برنامه ای بنویسید که یک رشته (مانند نام و نام خانوادگی خودتان) را به صورت متحرک بر روی LCD نمایش دهد به این صورت که رشته مذکور از سمت راست LCD وارد شود و از راست به چپ LCD را طی کند و از سمت چپ خارج شود.

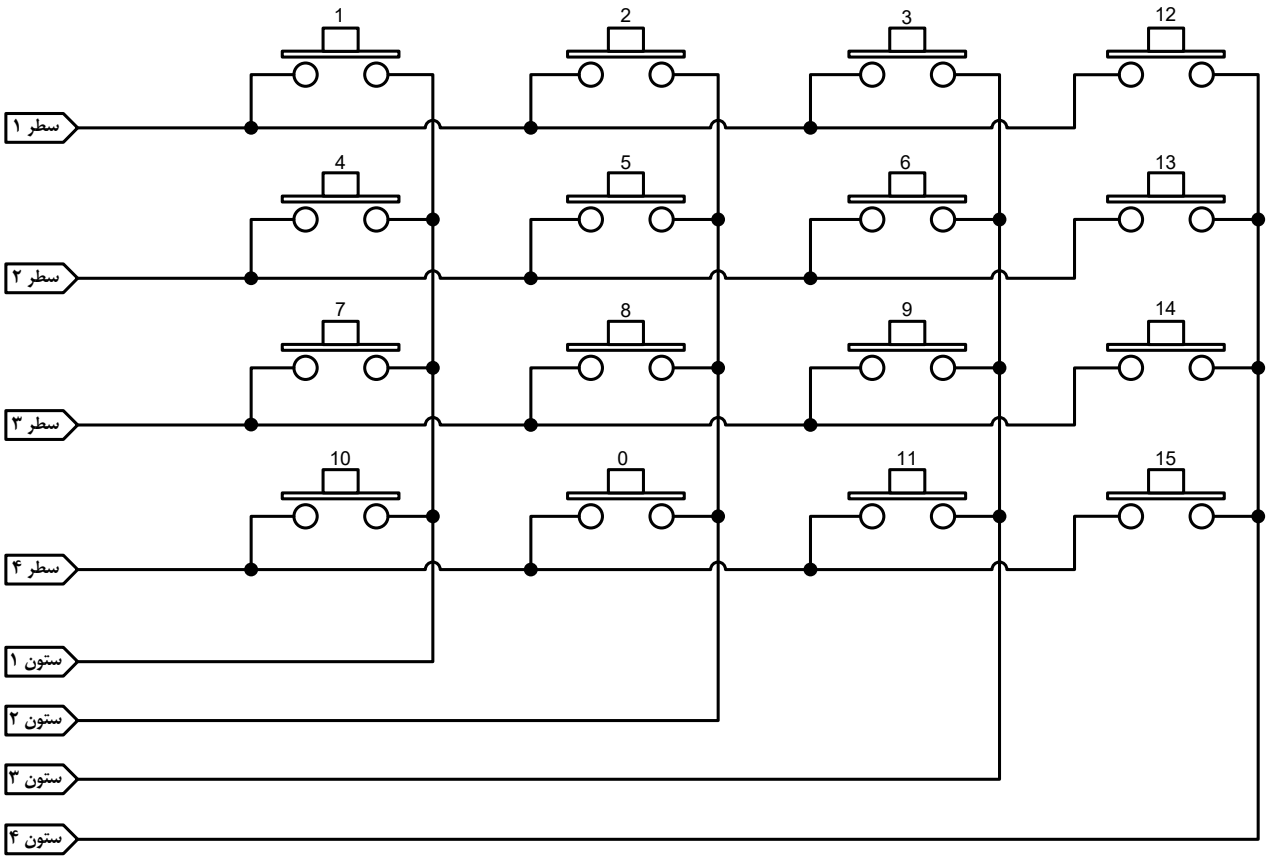
آزمایش ۴ – اتصال صفحه کلید به میکروکنترلرهای AVR

۴-۱ مقدمه

یکی از وسایلی که برای وارد کردن اطلاعات در میکروکنترلرها به کار می رود، صفحه کلید می باشد. ساختار صفحه کلید به صورت ماتریسی از سطر و ستون می باشد که با فشار هر کلید، یک سطر به یک ستون متصل می‌شود و در غیر این صورت، اتصالی بین سطر و ستون وجود ندارد.

برای خواندن از صفحه کلید توسط میکروکنترلر، معمولاً از روش اسکن صفحه کلید برای تشخیص کلید فشرده شده استفاده می‌شود. همانگونه که در شکل ۴-۱ نشان داده شده است می توان ستون ها را با یک مقاومت به سطح ولتاژ Vcc متصل نمود تا میکروکنترلر هنگامی که کلیدی فشرده نشده است، سطح منطقی High را بخواند. سطرها با توجه به الگوی چرخشی (... ←1110←1101←1011←0111 ...) اسکن می شوند. زمانی میکرو کنترلر می تواند در پایه یکی از ستون ها صفری تشخیص دهد که کلید فشرده شده، آن ستون را به سطر صفر شده وصل کند و با توجه به شماره سطر و ستون صفر شده، می توان کلید فشرده شده را تشخیص داد.

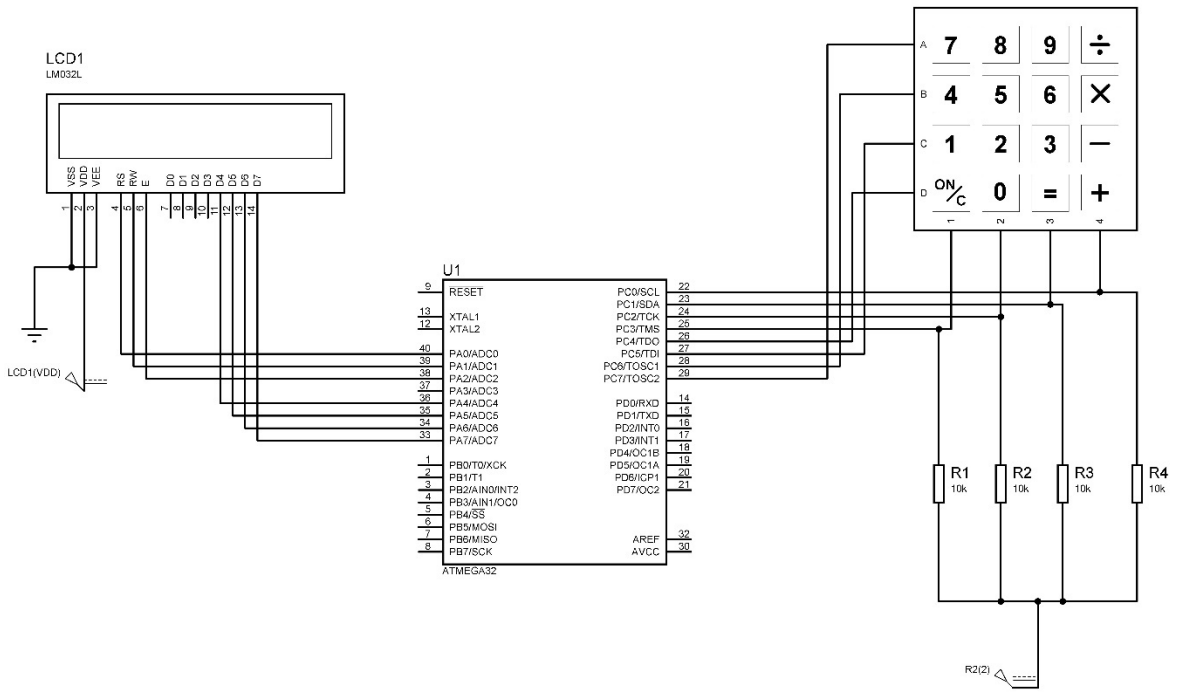
برای روشن شدن روش بالا، فرض کنید سطرهای یک صفحه کلید ۴×۴ به چهار بیت کم ارزش PortC و ستون های آن به چهار بیت پر ارزش PortC متصل شده اند. همچنین ستون ها توسط مقاومت هایی به Vcc وصل شده اند. حال اگر سطرها را صفر کنیم و هر یک از کلیدهای یک سطر را فشار دهیم، اتصال بین یک سطر و ستون برقرار می شود و ستون مربوطه نیز صفر می شود. به عنوان مثال، در شکل ۴-۱ اگر سطر یک را برابر صفر کرده باشیم، در این صورت اگر هر یک از کلیدهای ۱، ۲، ۳ یا ۱۲ را فشار دهیم، یکی از ستون های ۱ تا ۴ نیز مساوی صفر می شود؛ بنابراین اگر سطرها یا چهار بیت پر ارزش PortC یعنی PC4 تا PC7 را صفر کنیم، لذا با فشار یک کلید، ستون ۱ برابر با صفر و بقیه ستون ها مساوی ۱ می شوند یعنی مقدار ستون های PC0 تا PC3 برابر با 0111 می شود به این ترتیب کد کلیدهای سطر یک برابر با 00000111 باینری یا 0x07 در مبنای شانزده می باشد. حال با توجه به ستون صفر شده می توان تشخیص داد که کدام یک از کلیدهای سطر یک فشار داده شده است.



شکل ۴-۱: صفحه کلید ۴×۴

۴-۲ آزمایش اول

برنامه ای بنویسید که کلید فشرده شده در یک صفحه کلید ۴×۴ متصل به PortC را بر روی یک LCD متصل به PortA نمایش دهد. شمای مداری این آزمایش در شکل ۴-۲ نشان داده شده است.



شکل ۴-۲: شمای مداری آزمایش اول

برنامه ای که می توان برای اسکن صفحه کلید به کار برد، در ادامه آورده شده است :

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

unsigned char scan_key(void);
unsigned char code[4][4]={ {7,4,1,10},{8,5,2,0},{9,6,3,11},{12,13,14,15}
};
.
.
.
.
.
unsigned char scan_key(void)
{
  unsigned char i,data,num_key,temp;
  num_key=0xff;
  temp=0x70;
  for(i=0;i<4;i++){
    PORTC=temp;
    delay_ms(5);
    data=PINC & 0x0f;
    if(data==0x07)
      num_key=code[0][i];
    if(data==0x0B)
      num_key=code[1][i];
    if(data==0x0D)
      num_key=code[2][i];
    if(data==0x0E)
      num_key=code[3][i];
    temp= ((temp>>=1)  | 0x80) & 0xF0 ;

  }
  return num_key;
}
```

این برنامه را به دقت بررسی نموده و از آن به عنوان یک تابع در برنامه خود استفاده کنید به این صورت که مقدار بازگردانده شده توسط تابع مذکور را به عنوان شماره کلید فشرده شده به کار ببرید.

۴-۳ پرسش

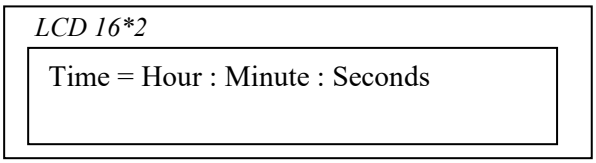
۱ـ همین برنامه را برای یک صفحه کلید ۴×۳ بنویسید.

۲ـ با استفاده از صفحه کلید ۴×۴ برنامه ای بنویسید که دو عدد را گرفته و با توجه به کلید فشرده شده در ستون چهارم (یعنی کلیدهای تقسیم، ضرب، تفریق و جمع) عمل متناظر با هر کلید را انجام داده و با فشردن کلید = نتیجه محاسبه را بر روی LCD نمایش دهد. همچنین اگر کلید ON/C فشار داده شود، صفحه نمایش LCD پاک شود.

آزمایش ۵ – ساعت دیجیتال به کمک میکروکنترلر AVR

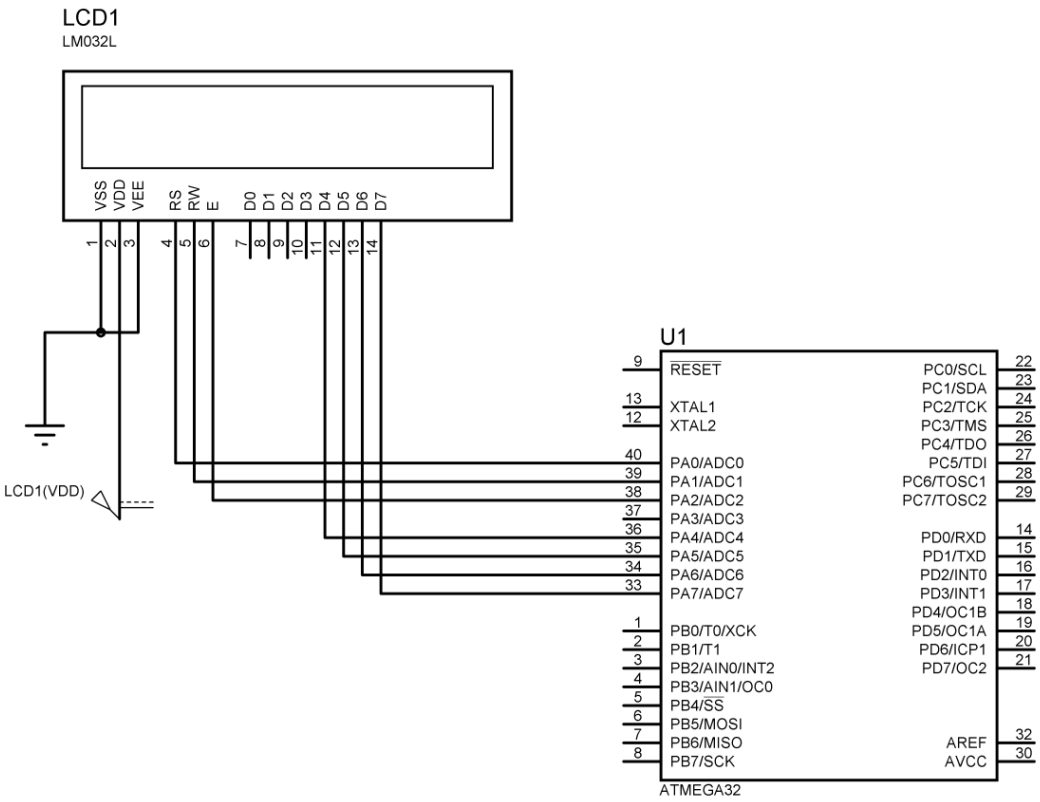
۵-۱ آزمایش اول

برنامه ای بنویسید که زمان به صورت ساعت، دقیقه و ثانیه بر روی یک LCD متصل به PortA نمایش داده شود و با شمارش و افزایش یک واحدی ثانیه شمار، دقیقه و ساعت نیز update شوند. برنامه را به گونه ای بنویسید که زمان به صورت نشان داده شده در شکل ۵-۱ بر روی LCD نمایش داده شود.



شکل ۵-۱: چگونگی نمایش زمان بر روی LCD

برای ایجاد تأخیر، از دستور delay_ms() استفاده کنید. شمای مداری این آزمایش در شکل ۵-۲ نشان داده شده است.



شکل ۵-۲: شمای مداری آزمایش اول

۵-۲ آزمایش دوم

برنامه آزمایش اول را به گونه ای بنویسید که در سطر دوم LCD شماره روز و ماه نیز نمایش داده شود.

۵-۳ پرسش

۱ـ برنامه ساعت دیجیتال را به گونه ای بنویسید که قبل از شروع به کار ساعت، تنظیم اولیه ساعت، دقیقه، ثانیه، روز و ماه توسط یک صفحه کلید انجام شود.

آزمایش ۶ – آشنایی با تایمر و کانتر در میکروکنترلر AVR

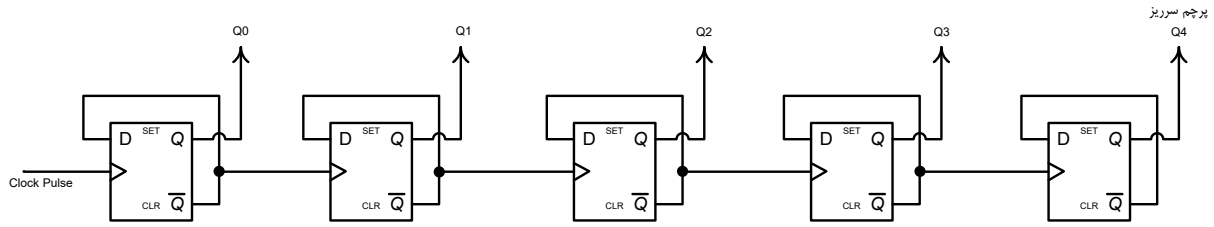
۶-۱ مقدمه

تایمر/کانتر یکی از بخش های مهم میکروکنترلرها می باشد. در بیشتر مواقع لازم که تعدادی وقایع خارجی (با سرعت بالا) شمارش شود و یا گاهی لازم است که در یک زمان خاص و دقیق، کاری صورت گیرد. تنها توسط تایمر/کانتر ها می توان این کارهای دقیق و با سرعت بالا را انجام داد.

میکروکنترلرهای AVR حداکثر دارای شش عدد تایمر کانتر هشت بیتی و شانزده بیتی هستند. برخی از آنها دارای عملکرد ساده و برخی دیگر دارای امکانات بیشتر نظیر تولید موج PWM ، حالت مقایسه CTC ، حالت تسخیر، عملکرد غیر همزمان و ... می باشند.

۶-۲ تئوری عملکرد تایمر/کانتر

تایمر و کانترها از تعدادی فلیپ فلاپ که به صورت پشت سر هم قرار گرفته و به صورت یک شمارنده آسنکرون عمل می کنند، تشکیل شده اند (شکل ۶-۱).



شکل ۶-۱: شمارنده آسنکرون

در این شمارنده پایه های Q3-Q0 به عنوان خروجی استفاده می شود. با فرض وارد نمودن مقدار 0000 و با اعمال پالس ساعت، شمارنده شروع به شمارش می نماید و حداکثر تا مقدار 1111 بالا می رود. پس از این حالت و با اعمال پالس ساعت بعدی، بیت سرریز (Q4) فعال می شود که نشان دهنده این است که شمارنده به حداکثر مقدار خود رسیده است.

هنگامی که پالس ساعت اعمالی به این شمارنده یک مقدار مشخص داشته باشد، می توان با در نظر گرفتن مدت زمان شمارش از مقدار xxxx تا 1111 و سپس سرریز شدن شمارنده، زمان های معینی را ایجاد نمود. در این حالت شمارنده به صورت تایمر عمل می کند. به عنوان نمونه برای شکل ۱ اگر پالس ساعت اعمالی دارای فرکانس ۲ Hz باشد و شمارنده نیز با مقدار پیش فرض 0000 در نظر گرفته شود، حداکثر ۸ ثانیه طول می کشد تا شمارنده سرریز شود.

هنگامی که منبع پالس ساعت اعمالی از یک منبع منظم و معین نباشد (منبع خارجی)، در این صورت می توان از شمارنده برای شمارش وقایع خارجی نظیر عبور تعداد قطعات از جلوی یک سنسور نوری استفاده نمود که در این حالت شمارنده به صورت کانتر به کار گرفته می شود.

۶-۳ تایمر/کانتر صفر

تایمر کانتر صفر در AVR ها را می توان در سه نوع عملکرد زیر دسته بندی کرد :

۱ ـ نوع عملکرد ساده هشت بیتی : این مدل فقط در سری ATTiny , AT90S به کار گرفته شده است.

۲ ـ نوع عملکرد پیشرفته هشت بیتی : این مدل در سری های ATmega (به غیر از ATmega8 و ATmega163 که از مدل ساده ۸ بیتی استفاده می کنند) به کار گرفته شده است.

۳ ـ نوع عملکرد پیشرفته شانزده بیتی : این مدل فقط در AVR های سری ATTiny2313 و ATTiny13 به کار گرفته است.

۶-۴ تایمر کانتر صفر در حالت هشت بیتی پیشرفته

حالت عملکرد هشت بیتی در سری های ATmega (به غیر از ATmega8 و ATmega163) قرار دارد. از خصوصیات تایمر/کانتر در این مدل می توان به موارد زیر اشاره کرد :

۱ ـ تایمر / کانتر در حالت عادی

۲ ـ تایمر / کانتر در حالت مقایسه CTC

۳ ـ تایمر / کانتر در حالت PWM سریع (تک شیب)

۴ ـ تایمر / کانتر در حالت PWM تصحیح فاز (دو شیب)

۶-۵ رجیستر های تایمر/کانتر صفر در حالت عملکرد هشت بیتی پیشرفته

۶-۵-۱ رجیستر مقایسه خروجی (OCR0)

این رجیستر هشت بیتی، خواندنی و نوشتنی بوده و به طور مستقیم با مقدار شمارنده TCNT0 مقایسه می شود. از تطابق این دو برای تولید وقفه خروجی یا تولید یک شکل موج روی پایه OC0 می توان استفاده نمود.

Bit	7	6	5	4	3	2	1	0
	OCR0 [7:0]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

۶-۵-۲ رجیستر تایمر کانتر صفر TCNT0

این رجیستر هشت بیتی امکان دسترسی مستقیم برای خواندن و نوشتن در شمارنده را فراهم می کند. این رجیستر، هم خواندنی است و هم نوشتنی. به هنگام خواندن، مقدار شمارش شده را از خروجی شمارنده برمی گرداند و به هنگام نوشتن، مقدار جدید را به ورودی شمارنده انتقال می دهد.

Bit	7	6	5	4	3	2	1	0
	TCNT0 [7:0]							
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

۶-۵-۳ رجیستر کنترلی تایمر/کانتر صفر TCCR0

این رجیستر دارای هشت بیت کنترلی برای انتخاب پالس ساعت، حالت خروجی هنگام تطابق مقایسه، عملکردهای PWM، و بیت مقایسه خروجی می باشد.

با استفاده از رجیستر TCCR0 می توان فرکانس پالس ساعت تایمر را انتخاب کرد که برای این کار از بیت های انتخاب پالس ساعت (CS00، CS01، CS02) استفاده می شود.

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

ـ بیت های انتخاب پالس ساعت (CS00، CS01 و CS02)

این سه بیت برای انتخاب فرکانس پالس ساعت تایمر صفر مورد استفاه قرار می گیرند. جدول ۶-۱ و جدول ۶-۲ انتخاب پالس ساعت برای میکروکنترلرهای ATmega128،ATmega64 و ATmega32 را نشان می دهد. تایمر در زمانی که هیچ منبع پالس ساعتی انتخاب نشود، غیرفعال است.

جدول ۶-۱: انتخاب پالس ساعت برای میکروکنترلرهای ATmega64 و ATmega128

CS02	CS01	CS00	Description
0	0	0	No Clock Source (Timer/Counter Stopped)
0	0	1	$CLK_{I/O}$ (No Prescaling)
0	1	0	$CLK_{I/O}/8$ (From Prescaler)
0	1	1	$CLK_{I/O}/32$ (From Prescaler)
1	0	0	$CLK_{I/O}/64$ (From Prescaler)
1	0	1	$CLK_{I/O}/128$ (From Prescaler)
1	1	0	$CLK_{I/O}/256$ (From Prescaler)
1	1	1	$CLK_{I/O}/1024$ (From Prescaler)

جدول ۶-۲: انتخاب پالس ساعت برای میکروکنترلرهای ATmega32

CS02	CS01	CS00	Description
0	0	0	No Clock Source (Timer/Counter Stopped)
0	0	1	$CLK_{I/O}$ (No Prescaling)
0	1	0	$CLK_{I/O}/8$ (From Prescaler)
0	1	1	$CLK_{I/O}/64$ (From Prescaler)
1	0	0	$CLK_{I/O}/256$ (From Prescaler)
1	0	1	$CLK_{I/O}/1024$ (From Prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

ـ بیت های حالت خروجی تطابق مقایسه (COM01:0)

این بیت ها رفتار پایه مقایسه خروجی (OC0) را کنترل می کنند. زمانی که این بیت ها هر دو صفر باشند، پایه OC0 که یکی از پایه های یک Port است، به صورت I/O معمولی استفاده می شود. اما اگر یکی یا هر دوی این بیت ها یک شوند، پایه OC0 به خروجی واحد تولید شکل موج متصل خواهد شد و دیگر نمی توان از آن به عنوان I/O معمولی استفاده نمود (جدول ۴).

ـ بیت های WGM01 و WGM00

توسط این دو بیت می توان حالت های عملکرد پشتیبانی شده تایمر/کانتر را عبارتند از : عملکرد عادی، عملکرد مقایسه ای، عملکرد با PWM سریع و عملکرد با PWM تصحیح فاز را انتخاب نمود (جدول ۶-۳).

جدول ۶-۳: انتخاب مد تولید موج

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

ـ بیت مقایسه خروجی (FOC0)

این بیت زمانی فعال می شود که بیت های WGM، یک حالت غیر PWM (عملکرد عادی و مقایسه) را انتخاب کرده باشند. در هنگام نوشتن یکِ منطقی در این بیت، یک تطابق مقایسه فوری روی واحد تولید شکل موج رخ می دهد و پایه OC0 را با توجه به تنظیم بیت های COM01 و COM00 تغییر می دهد.

در زمان یک کردن FOC0 پرچم مقایسه خروجی (OCF0) فعال خواهد شد و وقفه ای نیز تولید نمی شود. این بیت در کل، تأثیری بر هیچ رجیستر و پرچمی نمی گذارد و فقط وضعیت پایه OC0 را با توجه به تنظیمات آن تغییر می دهد.

۶-۵-۴ رجیستر پرچم وقفه تایمر کانتر صفر TIFR

از این رجیستر برای تشخیص یک سرریز یا تطابق در مقایسه در تایمر/کانتر صفر رخ داده باشد.

Bit	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

ـ بیت پرچم سرریز تایمر/کانتر صفر (TOV0)

بیت TOV0 زمانی فعال (یک) می شود که یک سرریز در تایمر/کانتر صفر به وقع بپیوندد. برای پاک کردن این بیت باید در آن یک نوشت. البته در زمان اجرای بردار وقفه، TOV0 توسط سخت افزار پاک می شود. در حالت PWM تصحیح فاز، TOV0 زمانی فعال می شود که جهت شمارش در $00 عوض شود.

ـ بیت پرچم مقایسه خروجی صفر (OCF0)

بیت OCR0 زمانی فعال (یک) می شود که یک تطابق مقایسه مابین تایمر/کانتر صفر (TCNT0) و داده درون رجیستر OCR0 به وقوع بپیوندد. برای پاک کردن این بیت باید در آن یک نوشت. البته در زمان اجرای بردار وقفه، OCR0 توسط سخت افزار پاک می شود.

۶-۵-۵ رجیستر پوشش وقفه تایمر/کانتر صفر (TIMSK)

Bit	7	6	5	4	3	2	1	0
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

ـ بیت فعالساز وقفه سرریز تایمر/کانتر صفر (TOIE0)

زمانی که بیت TOIE0 یک شود و همچنین بیت I در SREG نیز فعال باشد وقفه سرریز تایمر/کانتر صفر فعال می شود. هنگامی که بیت TOV0 در رجیستر پرچم وقفه (TIFR) فعال شود، یک وقفه درخواست خواهد شد.

ـ بیت فعالساز وقفه تطابق مقایسه خروجی تایمر/کانتر صفر

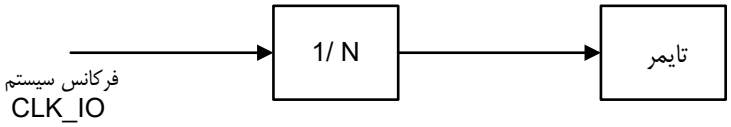
وقتی OCIE0 یک شده و بیت I (فعالساز وقفه سراسری) در SREG نیز فعال باشد، وقفه تطابق مقایسه تایمر/کانتر صفر فعال می شود. هنگامی که بیت OCF0 در رجیستر پرچم وقفه (TIFR) فعال شود، یک وقفه درخواست خواهد شد.

۶-۶ نحوه محاسبه زمان بندی برای تایمرها

برای محاسبه زمان بندی مورد نظر در تایمرها باید مراحل زیر انجام پذیرد :

۱ـ فرکانس داخلی پالس ساعت سیستم را در نظر گرفت.

۲ـ ضریب تقسیم پالس ساعت سیستم برای تولید پالس ساعت تایمر توسط بیت های CS00، CS01 و CS02 را تعیین نمود (شکل ۶-۲).



شکل ۶-۲: تعیین ضریب تقسیم پالس ساعت تایمر

۳ ـ مقدار مطلوب را رجیستر TCNTx قرار داد (x شماره تایمر مورد نظر است).

مثال ۱: با فرض نوسان ساز داخلی ۱ MHz و ضریب تقسیم N=۳۲ و TCNT0=0AH مقدار زمان تا فعال شدن پرچم سرریز را محاسبه کنید.

فرکانس پالس ساعت تایمر = $\frac{1\ MHz}{32} = 31.25\ KHz$

مدت زمان یک شمارش = $\frac{1}{32.25\ KHz} = 32\ \mu s$

تعداد پالس برای سرریز شدن تایمر = $256 - TCNT0 = 256 - 10 = 246$

مدت زمان شمارش تایمر = $246 \times 32\ \mu s = 7.872\ ms$

مثال ۲: با فرض استفاده از کریستال خارجی ۱۶ MHz برای تولید مدت زمان ۱۰ ms چه عددی را باید در تایمر یک (TCNT1) قرار داد ؟

با فرض انتخاب ضریب تقسیم ۶۴ (N=۶۴) داریم :

فرکانس پالس ساعت تایمر = $\frac{16\ MHz}{64} = 250\ KHz$

مدت زمان یک شمارش = $\frac{1}{250\ KHz} = 4\ \mu s$

تعداد پالس لازم برای مدت ۱۰ ms = $\frac{10\ ms}{4\ \mu s} = 2500$

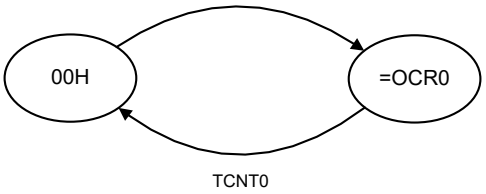
عدد لازم برای TCNT1 = $65536 - 2500 = 63036 = F63CH$

۶-۷ تایمر/کانتر صفر در حالت عادی

ساده ترین حالت عملکرد تایمر/کانتر، حالت عادی می باشد. جهت شمارش، رو به بالا (افزایشی) بوده و پس از رسیدن مقدار رجیستر TCNT0 به TOP دوباره از مقدار BOTTOM شروع به شمارش نموده و پرچم سرریز TOV0 فعال خواهد شد. مقدار TOP برابر حداکثر مقدار شمارش در یک چرخه است و BOTTOM مقداری است که با رسیدن شمارنده به آن، شمارش صفر می شود.

۶-۸ عملکرد تایمر/کانتر صفر در حالت مقایسه (CTC)

در حالت مقایسه، رجیستر TCNT0 به طور دایم با رجیستر OCR0 مقایسه و در صورت تطابق، رجیستر TCNT0 برابر با صفر می شود (شکل ۶-۳).



شکل ۶-۳: نمودار شمارش تایمر/کانتر صفر در حالت CTC

از نتیجه مقایسه می توان برای تولید شکل موج روی پایه مقایسه خروجی OC0 استفاده کرد (قسمت اول آزمایش). در هنگام تطبیق مقایسه در صورت فعال بودن وقفه و تولید پرچم OC0 می توان یک وقفه مقایسه را ایجاد نمود و از روال وقفه برای به روز رسانی مقدار رجیستر OCR0 (TOP) استفاده نمود. به هر حال تغییر رجیستر OC0 (TOP) به یک مقدار جدید در زمانی که تایمر در حال شمارش است باید با احتیاط انجام شود، زیرا حالت CTC دارای بافر مضاعف نمی باشد. حالت های متفاوت بیت های COM00 و COM01 در جدول ۶-۴آورده شده است.

جدول ۶-۴: تعیین حالت خروجی تطابق مقایسه (CTC)

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

همانگونه که از جدول ۶-۴ مشاهده می شود، چهار حالت به وجود می آید که عبارتند از:

۱ ـ پایه OC0 که پایه ای از یک Port می باشد به صورت I/O معمولی استفاده می شود.

۲ ـ پایه OC0 با خروجی مقایسه منطبق شده و در هر بار تطابق، خروجی تغییر وضعیت می دهد (Toggle).

۳ ـ پایه OC0 با خروجی مقایسه منطبق شده و در زمان تطابق، این پایه را صفر می کند.

۴ ـ پایه OC0 با خروجی مقایسه منطبق شده و در زمان تطابق، این پایه را یک می کند.

نکته: در حالت های ۲، ۳ و ۴ باید پین OC0 به صورت خروجی تعریف شود. (با نوشتن یک در DDRx).

۶-۹ عملکرد تایمر/کانتر صفر درحالت PWM سریع (تک شیب)

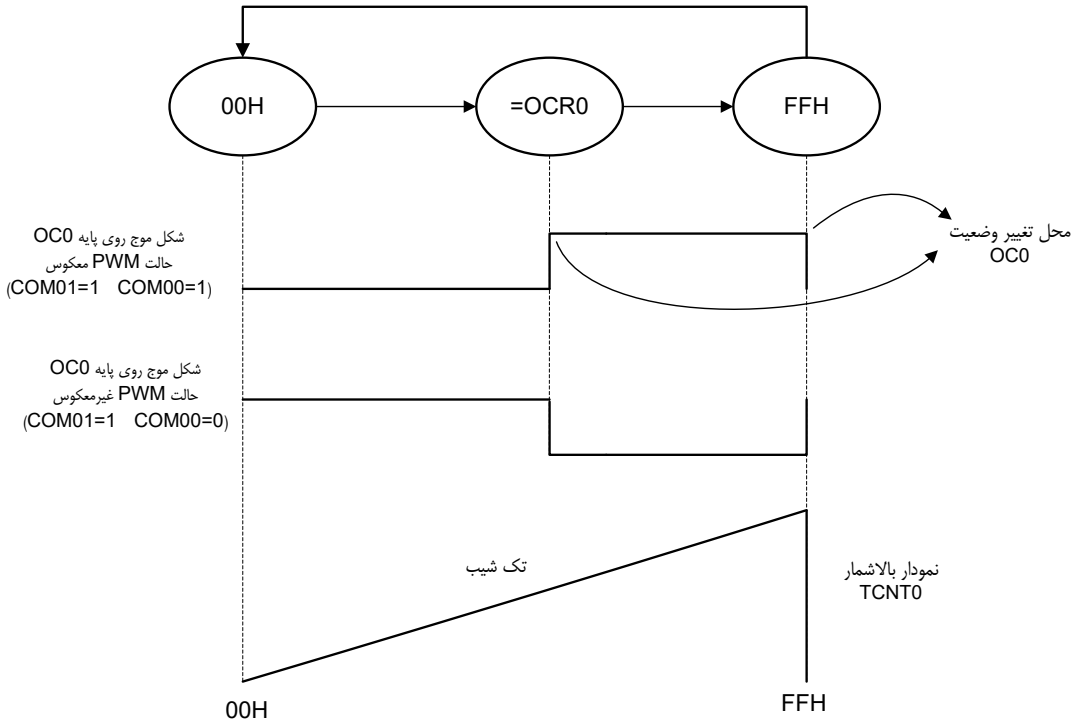
واژه PWM مخفف عبارت Pulse Width Modulation یا مدولاسیون پهنای پالس می باشد. در این مدولاسیون پهنای پالس تولیدی را می توان تحت کنترل داشت، به طوری که این پهنا در برخی مواقع می تواند متأثر از مقدار دامنه یک موج دیگر باشد. از کاربردهای PWM می توان به کنترل دور موتورهای AC، DC و منابع تغذیه سوییچینگ و ... اشاره نمود.

جهت ورود و دسترسی به مد عملکرد PWM سریع باید بیت های WGM00=1 و WGM01=1 قرار داد. همانگونه که درباره حالت مقایسه CTC توضیح داده شد، رجیستر OCR0 به طور دایم با رجیستر TCNT0 مقایسه می شود و پایه OC0 با توجه به تنظیمات مربوطه، تغییر وضعیت می دهد. عملکرد تایمر در حالت PWM سریع همانند حالت مقایسه می باشد با این تفاوت که در زمان تطابق بین OCR0 و TCNT0 رجیستر TCNT0 با مقدار 00H پر نمی شود، بلکه شمارش خود را تا حداکثر مقدار (FFH) ادامه داده و پس از سرریز شدن تایمر، رجیستر TCNT0 با مقدار 00H پر می شود و شمارش ادامه پیدا می کند. رجیستر OCR0 در حالت PWM دارای بافر مضاعف می باشد (شکل ۶-۴). باید توجه داشت که در حالت PWM پایه OC0 در دو حالت زیر تغییر وضعیت می دهد :

۱ ـ در حالت تطابق

۲ ـ در زمان سرریز تایمر

در صورتی که در حالت CTC فقط در زمان تطابق، وضعیت پایه OC0 تغییر می کند.



شکل ۶-۴: نمودار عملکر تایمر/کانتر صفر در حالت PWM سریع

در زمانی که تایمر/کانتر صفر در حالت PWM سریع قرار می گیرد، بیت های COM00 و COM01 عملکرد متفاوتی با حالت مقایسه (CTC) پیدا می کنند. این حالت ها در جدول ۶-۵ آورده شده است.

جدول ۶-۵: حالت خروجی مقایسه در PWM سریع

COM01	COM00	Description
0	0	Normal port operation, OC0 disconnected.
0	1	Reserved
1	0	Clear OC0 on compare match, set CO0 at TOP
1	1	Set OC0 on compare match, clear CO0 at TOP

در صورتی که وقفه سرریز تایمر فعال باشد (وقفه سراسری فعال)، با هر بار فعال شدن TOV0 می توان در زیربرنامه روال سرویس وقفه (ISR) رجیستر OCR0 را Update نمود تا پهنای PWM به مقدار دلخواه تنظیم شود.

فرکانس PWM خروجی را می توان از رابطه زیر محاسبه کرد :

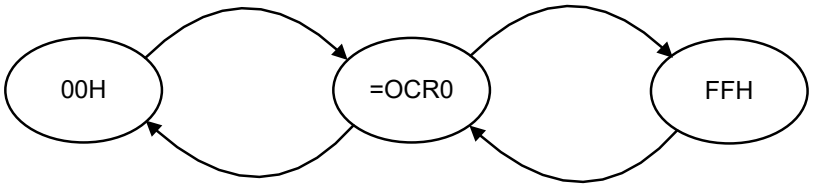
$$f_{PWM} = \frac{f_{clk\_I/O}}{N*255} \qquad (۱-۶)$$

که در آن N بیانگر ضریب تقسیم پالس ساعت سیستم برای تولید پالس ساعت تایمر می باشد و برابر یکی از اعداد ۱، ۸، ۳۲، ۶۴، ۱۲۸، ۲۵۶ و ۱۰۲۴ است.

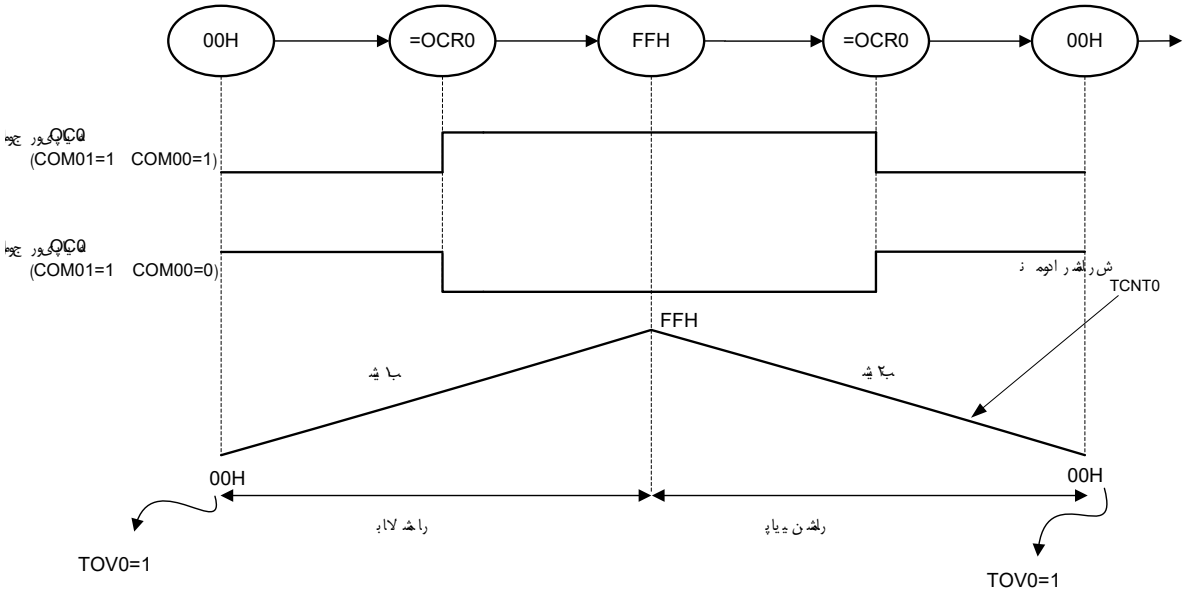
۶-۱۰ عملکرد تایمر/کانتر صفر درحالت PWM تصحیح فاز (دو شیب)

حالت PWM تصحیح فاز (WGM00=1 و WGM01=0) امکان تولید شکل موج PWM با تفکیک پذیری بالا را مهیا می سازد. در PWM تصحیح فاز مکرراً از 00H (BOTTOM) تا FFH (TOP) و سپس از FFH تا 00H شمارش می نماید. در هنگام شمارش به سمت بالا مقایسه گر، رجیستر TCNT0 را با OCR0 مقایسه می‌کند و هنگام برابر شدن، یک تغییر وضعیت روی پایه OC0 ایجاد می نماید و هنگام شمارش به سمت پایین، نیز همین عمل مجدداً تکرار می شود.

حالت عملکرد PWM به روش تصحیح فاز دارای فرکانس تولیدی پایین تری نسبت به PWM سریع می‌باشد. با این وجود با توجه به متقارن بودن عملکرد PWM تصحیح فاز، از آن بیشتر در کنترل دور موتور استفاده می شود. شکل ۶-۵ و شکل ۶-۶ نحوه عملکرد تایمر را در PWM تصحیح فاز نشان می دهند.



شکل ۶-۵: نمودار شمارش تایمر در حالت PWM تصحیح فاز



شکل ۶-۶: عملکرد تایمر در حالت PWM تصحیح فاز

یک نکته مهم این است که پرچم سرریز تایمر صفر (TOV0) زمانی فعال می شود که رجیستر TCNT0 برابر صفر باشد نه FFH. پس باید توجه داشت که در زمان شروع به کار تایمر، اگر TCNT0=0 باشد، پرچم سرریز فعال خواهد شد.

فرکانس موج PWM در حالت تصحیح فاز طبق معادله زیر قابل محاسبه است :

$$f_{PWM} = \frac{f_{clk\_I/O}}{N*510} \qquad (۲-۶)$$

که در آن N بیانگر ضریب تقسیم پالس ساعت سیستم برای تولید پالس ساعت تایمر می باشد و برابر یکی از اعداد ۱، ۸، ۳۲، ۶۴، ۱۲۸، ۲۵۶ و ۱۰۲۴ است.

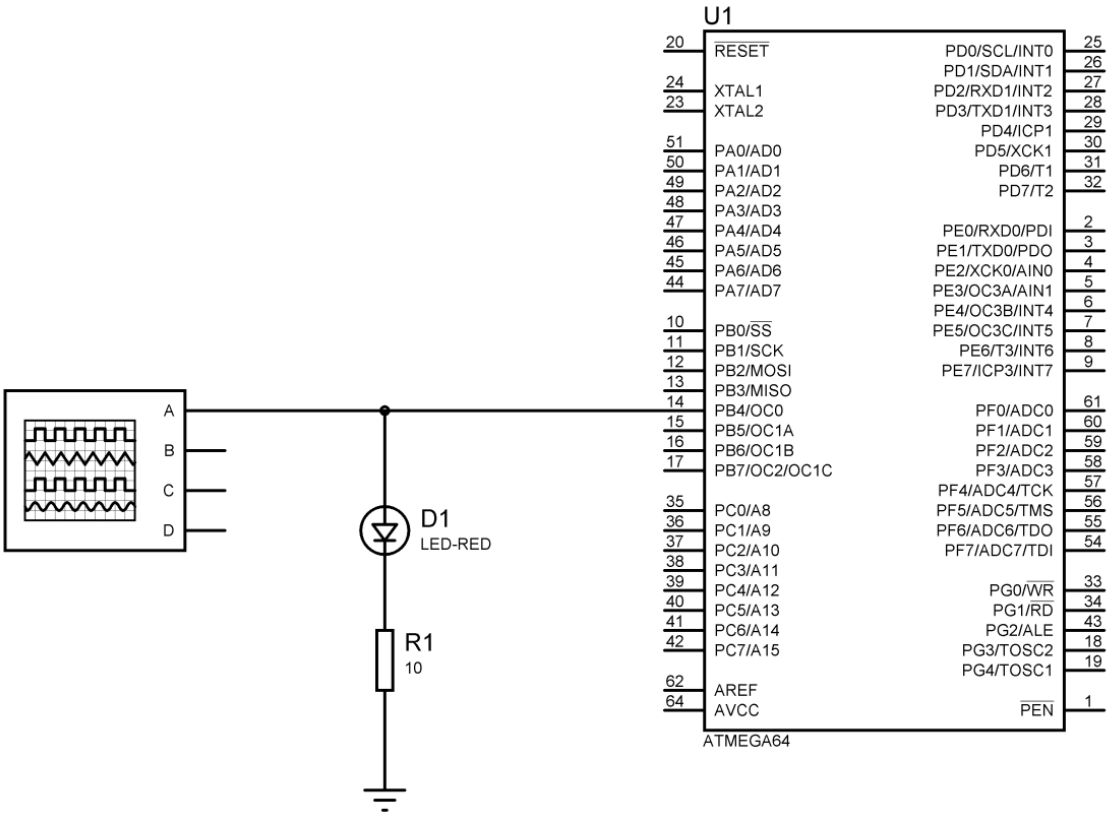
برای حالتی که تایمر/کانتر صفر در حالت PWM تصحیح فاز قرار می گیرد، بیت های COM00 و COM01 عملکرد متفاوتی پیدا می کنند که در جدول ۶-۶ آورده شده است.

جدول ۶-۶: حالت خروجی مقایسه در PWM تصحیح فاز

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

۶-۱۱ آزمایش اول

ابتدا مدار شکل ۶-۷ رابببندید.



شکل ۶-۷: مدار قسمت اول آزمایش

با استفاده از برنامه زیر می توان هر ms ۱ پایه OC0 (PORTB.4 در ATmega64) را مکمل کرد. فرکانس پالس ساعت سیستم برابر با MHz ۸ فرض شده است.

```
#include <mega64.h>
void main(void){
DDRB=0x10;
TCNT0=0x00;
OCR0=0x7C;
TCCR0=0x1C;
while (1);
}
```

در برنامه بالا مراحل زیر انجام می‌شود :

۱ ـ تعیین بیت چهارم PortB به عنوان خروجی جهت فعال شدن OC0 به صورت خروجی.

۲ ـ مقداردهی اولیه در رجیستر تایمر صفر (TCNT0) با مقدار 00H .

۳ ـ مقداردهی رجیستر مقایسه خروجی با 7CH (۱۲۴).

۴ ـ تنظیم فرکانس پالس ساعت تایمر صفر روی KHz ۱۲۵ (با فرض فرکانس پالس ساعت سیستم MHz ۸) و قرار دادن تایمر صفر در حالت عملکرد مقایسه و تنظیم عملکرد شکل موج روی پایه OC0 به صورت مکمل (Toggle) و شروع شمارش تایمر.

۵ ـ پایه OC0 (PortB.4) به صورت خودکار پس از ms ۱ مکمل می شود.

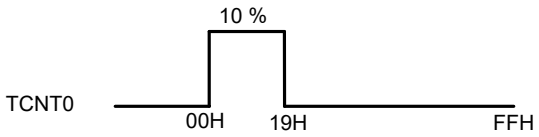
برای محاسبه زمان تایمر می توان از فرمول زیر استفاده نمود:

$$Time = \frac{N(1+OCR0)}{fclk\_IO} \qquad (۳-۶)$$

که در آن N ضریب تقسیم (۱، ۸، ۳۲، ۶۴، ۱۲۸، ۲۵۶، ۱۰۲۴) و fclk_IO فرکانس پالس ساعت سیستم می‌باشد.

۶-۱۲ آزمایش دوم

در این قسمت، می خواهیم برنامه ای را بنویسیم که یک موج PWM با عرض پالس ٪ ۱۰ را روی پایه OC0 ایجاد نماید (با استفاده از کریستال MHz ۱۶ خارجی). برای به دست آوردن پهنای پالس ٪ ۱۰ می توان به صورت زیر عمل نمود :



شکل ۶-۸: ایجاد پهنای پالس ٪ ۱۰

بنابراین داریم:

$$Duty\ Cycle = \frac{Time\ On}{Time\ On + Time\ Off} \times 100 \qquad (۶-۴)$$

و

$$\frac{255}{OCR0} = \frac{100\ \%}{10\ \%} = OCR0 = 25.5 \rightarrow \ OCR0 = 25 \qquad (۶-۵)$$

البته در این روش، یک خطا در پهنای پالس ایجاد می شود که می توان از آن صرفنظر نمود. برنامه زیر را می‌توان برای ایجاد چنین شکل موجی به کار برد.

```
#include <mega64.h>
void main(){
        DDRB=0x10;
        TCNT0=0x00;
        OCR0=25;
        TCCR0=0x6A;
        while(1);
}
```

مدار این قسمت از آزمایش، همانند قسمت قبل می‌باشد.

۶-۱۳ آزمایش سوم

با استفاده از تایمر صفر برنامه ای بنویسد که اعداد صفر تا ۹ را به ترتیب بر روی یک 7segment و با فاصله زمانی ۰/۲۵ ثانیه نشان دهد. فرکانس پالس ساعت را برابر با MHz ۱ و N را برابر با ۱۰۲۴ در نظر بگیرید.

۶-۱۴ پرسش

۱ ـ در برنامه قسمت دوم آزمایش، منظور از خطای ذکر شده چیست و چگونه می توان آن را برطرف کرد ؟ (برنامه آن را بنویسید).

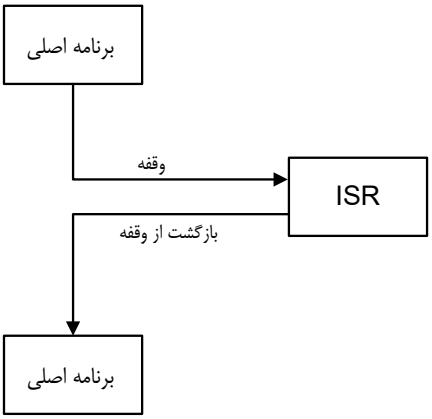
۲ـ منظور از بافر مضاعف چیست ؟

۳ ـ در برنامه ساعت دیجیتال در آزمایش ۵ برای ایجاد تأخیر به جای استفاده از دستور ()delay_ms از تایمر استفاده کنید.

آزمایش ۷ – آشنایی با وقفه و سازمان وقفه در میکروکنترلر AVR

۷-۱ مقدمه

وقفه، مکانیزمی از میکروکنترلر است که میکروکنترلر را برای پاسخ گویی به برخی از وقایع لحظه ای فعال می‌کند. وقفه ها نقش مهمی در میکروکنترلرها ایفا می کنند. در زمان رخداد یک وقفه، برنامه در حال اجرا متوقف شده و زیربرنامه وقفه شروع به اجرای برنامه خود می کند و پس از پایان زیربرنامه موردنظر، اجرای برنامه اصلی ادامه می یابد. برنامه ای را که CPU در زمان رخداد یک وقفه به آن رجوع می کند، روال سرویس وقفه (ISR) می نامند. شکل ۷-۱ نحوه عملکرد میکروکنترلر را به هنگام رخداد وقفه نشان می دهد.



شکل ۷-۱: عملکرد میکروکنترلر هنگام رخداد وقفه ها

۷-۲ مراحل اجرای یک وقفه

میکروکنترلر به محض دریافت یک وقفه، مراحل زیر را انجام می دهد :

۱ ـ دستوری که در حال اجرا می باشد را پایان داده و آدرس دستورالعمل بعدی را در پشته ذخیره می نماید.

۲ ـ رجوع به جدول بردار وقفه (محلی از حافظه) و به دست آوردن آدرس سرویس وقفه (ISR) و پرش به آن آدرس.

۳ ـ شروع مرحله اجرای ISR تا رسیدن به دستور بازگشت از وقفه (RETI).

۴ ـ برداشتن آدرس از پشته و قرار دادن آن در شمارنده برنامه (PC) و اجرای ادامه برنامه.

۲ ـ سازمان وقفه در **AVR**

در میکروکنترلرهای AVR منابع وقفه با توجه به نوع میکروکنترلر متفاوت است. به عنوان نمونه در میکروکنترلر ATmega8 تعداد ۹ منبع وقفه و در ATmega32 تعداد ۲۱ منبع وقفه وجود دارد.

۷-۳ بردار وقفه

هنگام رخداد یک وقفه، آدرسی که در شمارنده برنامه قرار می گیرد را بردار وقفه می نامند. این آدرس، همان آدرس شروع ISR مربوط به منبع وقفه است. در میکروکنترلرهای AVR برخلاف دیگر خانواده میکروکنترلرها نظیر ۸۰۵۱ که بردارهای وقفه در یک مکان ثابت و مشخص در ابتدای حافظه برنامه قرار داشتند، این قابلیت به کاربر داده شده است که مکان بردار وقفه را بین دو بخش حافظه برنامه کاربردی و BOOT حرکت دهد. پس با توجه به این موضوع می توان میکروکنترلرهای AVR را به صورت زیر تقسیم بندی نمود:

الف) AVR بدون حافظه BOOT

در این نوع میکروکنترلرها، فقط حافظه کاربردی وجود دارد، مانند سری های ATTiny و AT80s. در این AVRها بردارهای وقفه در یک مکان ثابت و مشخص در ابتدای حافظه برنامه قرار داده شده که در هنگام وقوع وقفه، CPU به آن محل از حافظه برنامه رجوع کرده و از آنجا شروع به اجرای برنامه می کند. اگر وقفه ای استفاده نشود، خانه مربوط به هر یک از وقفه ها می تواند به صورت خانه معمولی حافظه برنامه در نظر گرفته شود.

ب) AVR با حافظه BOOT

این تقسیم بندی برای میکرو کنترلرهای سری ATmega بوده (به غیر از ATmega103، ATmega603 و ATmega48) که هم حافظه BOOT و هم حافظه کاربردی را پشتیبانی می نمایند.

۷-۴ رجیستر کنترل وقفه عمومی (GICR)

ثبات کنترل وقفه عمومی در ادامه نشان داده شده است.

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

دو بیت از بیت های مهم این ثبات عبارتند از:

ـ بیت انتخاب بردار وقفه (IVSEL)

از این بیت برای انتخاب بردار وقفه استفاده می شود. زمانی که IVSEL=0 باشد، بردار وقفه در آغاز حافظه کاربردی و اگر IVSEL=1 باشد، بردار وقفه در آغاز حافظه BOOT قرار می گیرد.

ـ بیت فعال ساز تعیین بردار وقفه (IVCE)

برای تغییر IVSEL، ابتدا باید بیت IVCE را یک کرد و سپس مقدار دلخواه را در IVSEL قرار داد. بیت IVCE به طور خودکار و به صورت سخت افزاری پس از گذشت چهار سیکل صفر می شود.

** **نکته:** بیت های IVSEL و IVCE در میکروکنترلرهای ATmega64، ATmega169 و ATmega128 در رجیستر MCUCR قرار دارند.

برای جلوگیری از تغییرات غیرعادی در جدول بردار وقفه، مراحل زیر را دنبال نمایید:

۱ ـ فعال کردن بیت IVCE

۲ ـ با فعال کردن IVCE به مدت چهار سیکل فرصت دارید تا در IVSEL مقدار دلخواه خود را بنویسید. پس از این مدت به طور خودکار IVCE صفر می شود. در این مدت، وقفه ها به صورت خودکار غیرفعال خواهند بود.

۷-۵ وقفه های خارجی (External Interrupts)

میکروکنترلرهای AVR، علاوه بر وقفه های داخلی مانند وقفه تایمرها، سریال، مقایسه کننده آنالوگ و ... از وقفه های خارجی نیز پشتیبانی می کنند. این وقفه ها با توجه به نوع میکروکنترلرها می توانند از یک (در میکروکنترلر ATtiny13) تا هشت (در میکروکنترلرهای ATmega103 و ATmega128) وقفه خارجی باشند. میکروکنترلر ATmega32 از سه وقفه خارجی (INT0، INT1 و INT2) پشتیبانی می کند.

وقفه های خارجی توسط پین های INT0-7 راه اندازی می شوند. حتی اگر پین های INT0-7 به صورت خروجی پیکربندی شده باشند، وقفه ها راه اندازی خواهند شد. وقفه های خارجی می توانند با یک لبه پایین رونده یا بالا رونده و یا یک سطح پایین فعال شوند. یکی از خصوصیات وقفه ها تشخیص غیرهمزمان آنها می باشد که از این خصوصیت می توان برای بیدار کردن CPU از بخشی از مدهای sleep به غیر از مد Idle (در این مد، پالس ساعت I/O متوقف است) استفاده نمود.

۷-۶ فعال کردن وقفه ها

۷-۶-۱ فعال نمودن وقفه سراسری

پیش از به کارگیری وقفه باید بیت فعالساز وقفه عمومی (I) را فعال نمود. این بیت در رجیستر وضعیت (SREG) قرار دارد.

Bit	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

I: بیت فعالساز وقفه عمومی

همانگونه که گفته شد، خانواده AVR تا هشت (از ۰ تا ۷) وقفه خارجی را پشتیبانی می کند. بنابراین رجیسترهای داخلی مربوط به کنترل و وضعیت هر وقفه در هر میکروکنترلر ممکن است متفاوت با دیگری باشد. در ادامه، به طور نمونه به بررسی رجیسترهای مربوط به وقفه ATmega32 و ATmeg128 خواهیم پرداخت که به ترتیب ۳ و ۸ منبع وقفه خارجی را پشتیبانی می کنند. البته اساس کار وقفه در تمام سری های AVR یکسان بوده و تنها تفاوت موجود، فقط در تعداد وقفه ها ونام رجیسترها خلاصه می شود.

۷-۶-۲ رجیستر کنترل (MCUCR)

رجیستر کنترل MCU (Master Control Unit) شامل بیت های کنترل وقفه و عملکردهای AVR در حالت کلی می‌باشد.

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

ـ بیت های ISC00 و ISC01

از بیت های ISC00 و ISC01 جهت تعیین نوع فعالسازی وقفه خارجی روی لبه بالا رونده، لبه پایین رونده، حساس به سطح و یا حساس به هر تغییری بر روی پین INT0 استفاده می شود، حتی اگر پین های مربوطه به صورت خروجی پیکربندی شده باشند (جدول ۷-۱).

جدول ۷-۱: وضعیت های مختلف بیت های ISC00 و ISC01

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

مقدار موجود روی پایه INT0 قبل از تشخیص لبه ها، نمونه بردای می شوند. اگر وقفه لبه یا Toggle انتخاب شده باشد، پالس هایی که مدت زمان آنها بیش از سیکل پالس ساعت طول بکشد، وقفه را تولید خواهد کرد و در پالس های کوتاه، ضمانتی برای اجرای وقفه وجود ندارد. همچنین اگر وقفه سطح انتخاب شود، این سطح باید تا زمان اجرای دستورالعمل جاری تحت تولید وقفه پایین نگه داشته شود.

ـ بیت های ISC10 و ISC11

وظیفه این بیت ها همانند بیت های ISC00 و ISC01 اما برای INT1 است.

ـ بیت های SM2-0

از این بیت ها برای انتخاب مد sleep استفاده می شود.

ـ بیت SE

از این بیت برای فعال کردن مد sleep استفاده می شود.

** **نکته:** رجیسترهای EICRA و EICRB در ATmega128 همان وظیفه رجیستر MCUCR در ATmega32 را برعهده دارند.

Bit	7	6	5	4	3	2	1	0
	ISC31	ISC30	ISC21	ISC21	ISC11	ISC10	ISC01	ISC00
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Bit	7	6	5	4	3	2	1	0
	ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

۷-۶-۳ رجیستر وقفه و کنترل AVR (MCUCSR)

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

ـ بیت ISC2

از این بیت برای تعیین نوع فعالسازی وقفه خارجی ۲ در ATmega32 استفاده می شود.

** **نکته:** وقفه INT2 در Atmega32 فقط با لبه فعال خواهد شد.

اگر در ISC2 صفر نوشته شود، INT2 توسط یک لبه پایین رونده فعال و اگر در ISC2 یک نوشته شود، INT2 توسط یک لبه بالارونده فعال می شود.

پالس های تولیدکننده وقفه باید بیش از ۵۰۰ ns طول بکشد و در پالس های کوتاه تر از آن، ضمانتی برای تولید وقفه وجود ندارد.

** **نکته:** هنگام تغییر بیت ISC2 ممکن است وقفه ای رخ دهد؛ بنابراین پیشنهاد می شود برای جلوگیری از این حالت، ابتدا بیت INT2 در رجیستر فعالساز وقفه (GICR) را غیرفعال کنید. سپس بیت ISC2 را تغییر دهید و مجدداً بیت INT2 در رجیستر GICR را فعال نمایید.

۷-۶-۴ رجیستر کنترل وقفه عمومی (GICR)

به کمک این رجیستر می توان وقفه های خارجی را در ATmega32 فعال یا غیرفعال نمود.

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

بیت INT0 : بیت فعالساز وقفه خارجی صفر

بیت INT1 : بیت فعالساز وقفه خارجی یک

بیت INT2 : بیت فعالساز وقفه خارجی دو

** **نکته:** رجیستر EIMSK در ATmega128 همان وظیفه GICR در Atmega32 را برعهده دارد.

Bit	7	6	5	4	3	2	1	0
	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

۷-۶-۵ رجیستر پرچم وقفه عمومی (GIFR)

در زمان وقوع وقفه، پرچم مربوط به آن در این رجیستر فعال می شود.

Bit	7	6	5	4	3	2	1	0
	INTF1	INTF0	INTF2	-	-	-	-	-
Read/Write	R/W	R/W	R/W	R	R	R	R	R
Initial Value	0	0	0	0	0	0	0	0

ـ بیت INTF1 (پرچم وقفه خارجی ۱)

زمانی که یک تغییر سطح روی پین INT1 رخ دهد، بیت INTF1 یک خواهد شد و اگر بیت I در SREG و بیت INT1 در GICR یک باشند، CPU به محل بردار وقفه پرش خواهد کرد. این پرچم در حین اجرای زیرروال وقفه پاک می شود. زمانی که INT1 به صورت یک وقفه سطح پیکربندی شده باشد، به صورت خودکار صفر می‌شود.

ـ بیت INTF2 (پرچم وقفه خارجی ۲)

عملکرد این بیت همانند INTF1 است.

** **نکته:** رجیستر EIFR در ATmega128 همان وظیفه GIFR در ATmega32 را بر عهده دارد.

Bit	7	6	5	4	3	2	1	0
	INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	INTF0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

در AVRها امکان تعیین اولویت وقفه وجود ندارد و هر وقفه ای که در آدرس پایین تری قرار دارد، از اولویت بالاتری برخوردار می باشد.

۷-۷ برنامه نویسی وقفه ها در Codevision

برای استفاده از وقفه در Codevision از کلمه کلیدی interrupt به فرم زیر کمک می گیریم :

```
interrupt [Vector number] void int_expresion (void) {
        برنامه سرویس وقفه
}
```

عبارت Vector number شمارنده بردار وقفه مورد نظر بوده و از روی جدول بردار وقفه به دست می آید؛ به عنوان نمونه [۱] برای بردار Reset و [۲] برای بردار INT0 و ... . در جدول ۲ بردارهای وقفه برای میکروکنترلر ATmega32 آورده شده است.

عبارت int_expresion نام تابع سرویس دهنده وقفه بوده که می تواند توسط برنامه نویس نوشته شود.

نکته : قبل از به کارگیری این تابع باید رجیسترهای مربوط به وقفه را تنظیم نموده تا وقفه ها فعال شوند. همچنین تابع وقفه چیزی را باز نمی گرداند.

جدول ۷-۲: : بردارهای وقفه برای ATmega32

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

۷-۸ شرح آزمایش

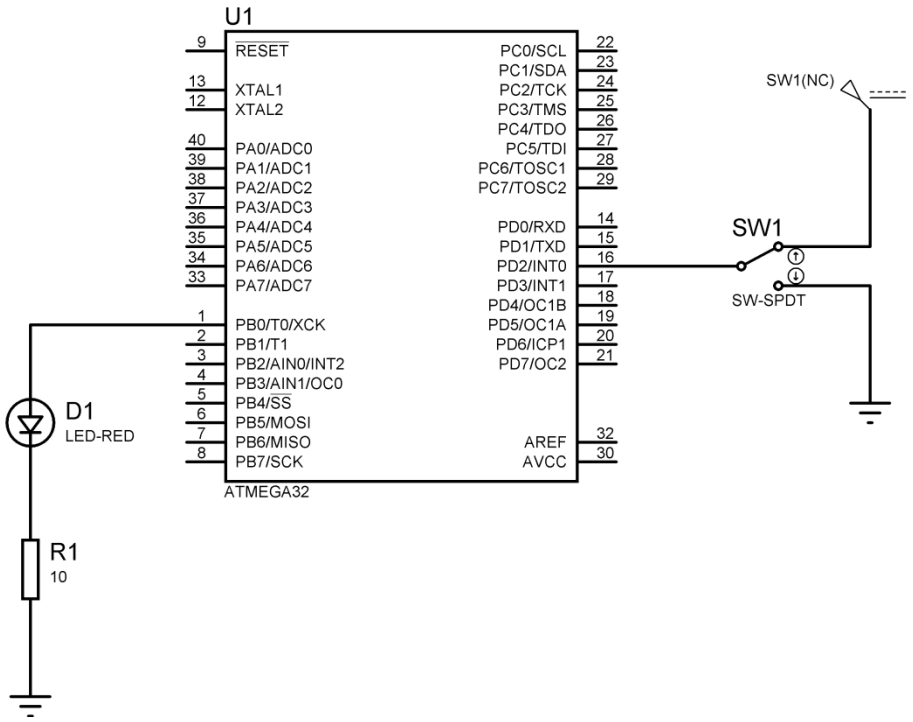
۷-۸-۱ آزمایش اول

می خواهیم برنامه ای را بنویسیم که هنگام وقوع وقفه خارجی صفر، LED متصل به PortB.0 به مدت ۱ ثانیه روشن و ۲ ثانیه خاموش شود. توجه شود که از میکروکنترلر ATmega32 استفاده نموده ایم. برای این منظور می‌توان برنامه زیر را نوشت:

```
#include <mega32.h>
#include <delay.h>
#define LED PORTB.0
void main(){
    DDRB=0xFF;          به‌صورت خروجی PORT B
    PORTB=0x00;
    GICR=0x40;          فعال نمودن وقفه خارجی صفر
    MCUCR=0x03;         فعال شدن وقفه با لبه بالارونده
    GIFR=0x40;
    #asm("sei");        فعال کردن وقفه سراسری
    while(1);           در انتظار وقفه
}
//------------------------------
interrupt [2] void ext_int0 (void)
```

```
{
    #asm("cli");          غیر فعال کردن وقفه سراسری
    LED=1;
    delay_ms(1000);
    LED=0;
    delay_ms(2000);
    #asm("sei");          فعال کردن وقفه سراسری
}
```

این برنامه را به دقت بررسی نمایید و با بستن مدار نشان داده شده در شکل ۷-۲ طرز کار آن را مشاهده و درستی برنامه را تحقیق نمایید.



شکل ۷-۲: مدار قسمت اول آزمایش

۷-۸-۲ آزمایش دوم

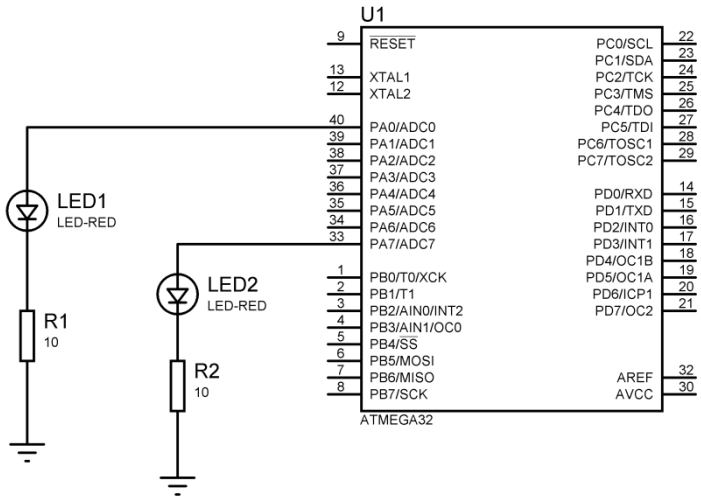
در این قسمت، می خواهیم یک وقفه داخلی را ایجاد کنیم. برای این منظور فرض می کنیم که تایمر صفر در حالت عملکرد CTC فعال شده باشد. می خواهیم هرگاه در مقایسه بین عدد موجود در رجیستر TCNT0 و رجیستر OCR0 مطابقت وجود داشت، یک وقفه از نوع تایمر صفر فعال شود و LED متصل به PortB.7 را برای ۱ ms روشن و دوباره خاموش نماید. باید توجه داشت که برای استفاده از وقفه تایمر صفر در حالت عملکرد CTC حتماً بایستی بیت OCIE0 در رجیستر TIMSK یک شود. برنامه این قسمت از آزمایش در ادامه آورده شده است.

```
#include <mega32.h>
#include <delay.h>
#define xtal 8000000
#define LED1 PORTA.0
```

```
#define LED2 PORTA.7

void main(void){
  DDRA=0xFF;
  PORTA=0x00;
  TIMSK=0x02;
  LED1=1;
  TCNT0=0x00;
  OCR0=0xFF;
  TCCR0=0x1C;
  #asm("sei");
  while (1);
}
//---------------------------------
interrupt [11] void tim0_comp(void)
{
  #asm("cli");
  LED1=0;
  LED2=1;
  delay_ms(100);
  LED2=0;
  LED1=1;
  delay_ms(100);
  #asm("sei");
}
```

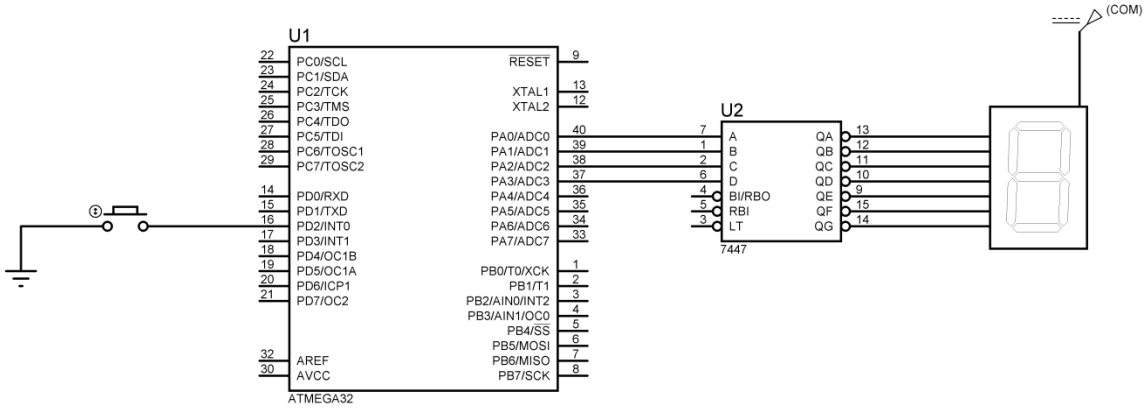
این برنامه را نیز به دقت مورد بررسی قرار داده و با بستن مدار نشان داده شده در شکل ۷-۳ طرز کار آن را مشاهده و درستی برنامه را تحقیق نمایید.



شکل ۷-۳: مدار قسمت دوم آزمایش

۷-۸-۳ آزمایش سوم

برنامه ای بنویسد که اعداد صفر تا ۹ را به ترتیب بر روی یک 7segment نشان دهد، اما هرگاه سوییچ متصل به PortD.2 فشار داده شد، شمارش آن برعکس شود و از پایان شمارش معکوس، به کار عادی شمارش از ۰ تا ۹ بازگردد. شماتیک مدار این قسمت از آزمایش در شکل ۷-۴ نشان داده شده است.



شکل ۷-۴: مدار قسمت سوم آزمایش

۷-۹ پرسش

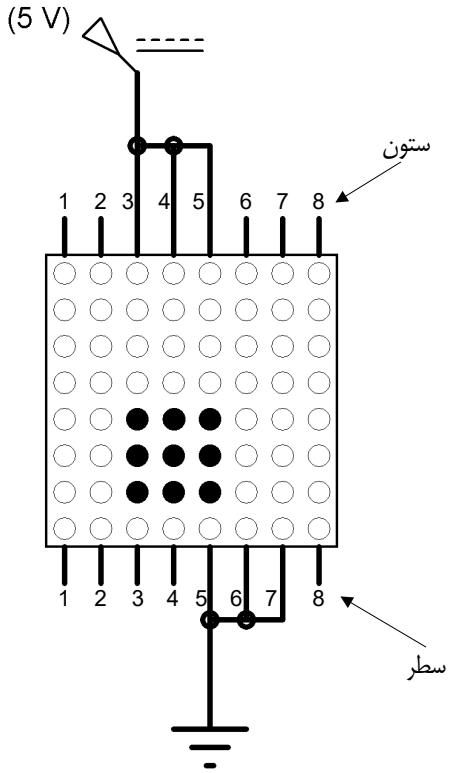
۱ ـ برنامه ای بنویسید که یک سیگنال مربعی را دریافت کند و فرکانس آن را محاسبه و بر روی یک LCD نمایش دهد.

** **راهنمایی:** در این پرسش، بایستی از تایمر به عنوان شمارنده (کانتر) استفاده شود. عملکرد این مدار فرکانس متر به این صورت است که توسط تایمر صفر، زمانی به مدت یک ثانیه تولید می شود و همزمان با آن، تایمر یک نیز شروع به شمارش تعداد پالس های اعمالی بر روی پایه T1 می کند (در صورت استفاده از ATmega16 منظور از پایه T1 همان پایه PB1 است). پس از طی زمان یک ثانیه، مقدار شمارش شده در رجیستر تایمر یک، معادل فرکانس موج روی پایه T1 است و می توان آن را بر روی LCD نمایش داد. می‌توان از هر لبه پایین رونده مانند یک وقفه خارجی استفاده کرد.

آزمایش ۸ - آشنایی با اصول اسکن Dot Matrix و نمایش اطلاعات بر روی آن (تابلو روان)

۸-۱ مقدمه

از Dot Matrix برای ساخت تابلو روان استفاده می شود. برای نمایش تصویر بر روی Dot Matrixها از اصول اسکن سطری پیروی می شود؛ به این ترتیب که هر LED را که بخواهیم روشن کنیم باید ستونی که آن LED در آن قرار دارد را یکِ منطقی (۵ ولت) و سطری که آن LED در آن قرار دارد را صفر (زمین) نماییم. یک نمونه از چنین ساختاری در شکل ۸-۱ نشان داده شده است. می توان دید که با یک کردن ستون های ۳، ۴ و ۵ و صفر کردن سطرهای ۵، ۶ و ۷ یک دسته از LEDها در سطر و ستون های متناظر روشن شده اند.



شکل ۸-۱: نحوه اسکن شدن سطرها و ستون ها در Dot Matrix

۸-۲ آزمایش اول

می خواهیم برنامه ای را بنویسیم که توسط آن، حرف A بر روی یک Dot Matrix نمایش داده شود. این برنامه در ادامه آورده شده است.

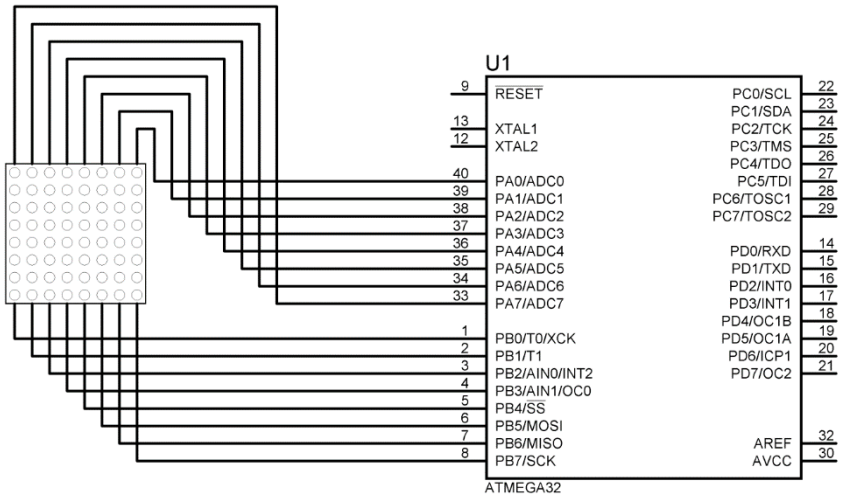
```
#include <mega32.h>
#include <delay.h>
#define xtal 8000000

unsigned char k;
flash unsigned char arr[8] = {0x18, 0x3C, 0x66, 0x66, 0x7E, 0x66, 0x66,
0x00};

void main(void)
{
  DDRA = 0xFF;
  DDRB = 0xFF;

  while(1){
  for(k=0;k<=7;k++){
    PORTA = arr[k];
    PORTB = ~(1<<k);
    delay_us(100);
    PORTB = 0xFF;
    }
  }
}
```

این برنامه را به دقت بررسی نمایید و با بستن مدار نشان داده شده در شکل ۸-۲ طرز کار آن را مشاهده و درستی برنامه را تحقیق نمایید.



شکل ۸-۲: مدار قسمت اول آزمایش

۸-۳ آزمایش دوم

با توجه به قسمت اول آزمایش، برنامه را به گونه ای تغییر دهید که بر روی Dot Matrix حروف K و Z و اعداد 3 و 6 و حروف فارسی آ و پ نمایش داده شود.

۸-۴ آزمایش سوم

برنامه ای بنویسید که اعداد صفر تا ۹ (فارسی) را با یک فاصله زمانی دلخواه، بر روی یک Dot Matrix نمایش دهد.

۸-۵ پرسش

۱ ـ به کمک چندین Dot Matrix در کنار یکدیگر، یک تابلو روان طراحی کنید که هر رشته دلخواه مانند نام خودتان را نمایش دهد. برای سادگی، رشته ها را انگلیسی در نظر بگیرید و برنامه آن را به گونه ای بنویسید که رشته ها از چپ به راست حرکت کنند.

** **نکته مهم:** در جدول ۸-۱اعدادی که باید در آرایه قرار گیرند برای حروف و اعداد انگلیسی آورده شده است. توجه شود که این اعداد در مبنای ۱۶ هستند.

جدول ۸-۱: آرایه‌های مختلف برای نمایش حروف و اعداد انگلیسی

مقدار نمایش داده شده	آرایه مورد نیاز
A	{0x18,0x3c,0x66,0x66,0x7e,0x7e,0x66,0x66}
B	{0x3c,0x22,0x22,0x3e,0x3e,0x22,0x22,0x3c}
C	{0x3c,0x42,0x40,0x40,0x40,0x40,0x42,0x3c}
D	{0x7c,0x62,0x62,0x62,0x62,0x62,0x62,0x7c}
E	{0x3c,0x20,0x20,0x3c,0x20,0x20,0x20,0x3c}
F	{0x3e,0x20,0x20,0x3c,0x20,0x20,0x20,0x20}
G	{0x1c,0x22,0x20,0x20,0x2e,0x22,0x22,0x1c}
H	{0x66,0x66,0x66,0x7e,0x7e,0x66,0x66,0x66}
I	{0x3c,0x18,0x18,0x18,0x18,0x18,0x18,0x3c}
J	{0x1f,0x04,0x04,0x04,0x04,0x04,0x24,0x18}
K	{0x22,0x24,0x28,0x30,0x30,0x28,0x24,0x22}
L	{0x60,0x60,0x60,0x60,0x60,0x60,0x7e,0x7e}
M	{0x42,0x66,0x5e,0x5a,0x42,0x42,0x42,0x42}
N	{0x41,0x61,0x51,0x49,0x45,0x43,0x41,0x41}
O	{0x3c,0x42,0x42,0x42,0x42,0x42,0x42,0x3c}
P	{0x3c,0x66,0x66,0x7c,0x60,0x60,0x60,0x60}
Q	{0x3c,0x42,0x42,0x42,0x4a,0x46,0x46,0x3d}
R	{0x3c,0x62,0x62,0x7e,0x78,0x6c,0x66,0x62}
S	{0x3c,0x42,0x20,0x10,0x08,0x04,0x42,0x3c}
T	{0x7e,0x7e,0x18,0x18,0x18,0x18,0x18,0x18}
U	{0x66,0x66,0x66,0x66,0x66,0x66,0x7e,0x3c}

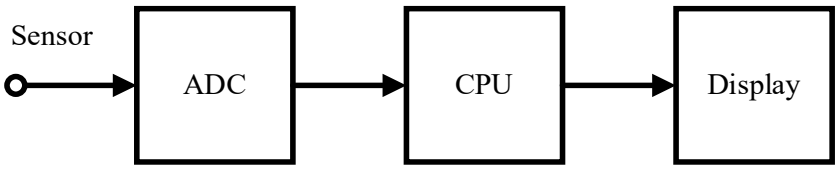
V	{0x66,0x66,0x66,0x66,0x66,0x66,0x3c,0x18}
W	{0x82,0x92,0x92,0xaa,0xc6,0x82,0x82,0x00}
X	{0x81,0x42,0x24,0x18,0x18,0x24,0x42,0x81}
Y	{0xc3,0x66,0x3c,0x18,0x18,0x18,0x18,0x18}
Z	{0xfe,0xfe,0x0c,0x18,0x30,0x60,0xfe,0xfe}
0	{0x3c,0x42,0x46,0x4a,0x52,0x62,0x42,0x3c}
1	{0x38,0x38,0x18,0x18,0x18,0x18,0x18,0x3c}
2	{0x38,0x7c,0x46,0x06,0x0c,0x18,0x30,0x7e}
3	{0x38,0x44,0x04,0x3c,0x3c,0x04,0x44,0x38}
4	{0x66,0x66,0x66,0x7e,0x3e,0x06,0x06,0x06}
5	{0x7e,0x60,0x60,0x3c,0x06,0x06,0x7c,0x38}
6	{0x38,0x44,0x40,0x78,0x44,0x44,0x38,0x00}
7	{0x7e,0x7e,0x06,0x06,0x06,0x06,0x06,0x06}
8	{0x3c,0x66,0x66,0x7e,0x7e,0x66,0x66,0x3c}
9	{0x7c,0x46,0x46,0x3e,0x02,0x02,0x42,0x3c}

آزمایش ۹ – آشنایی با مبدل آنالوگ به دیجیتال

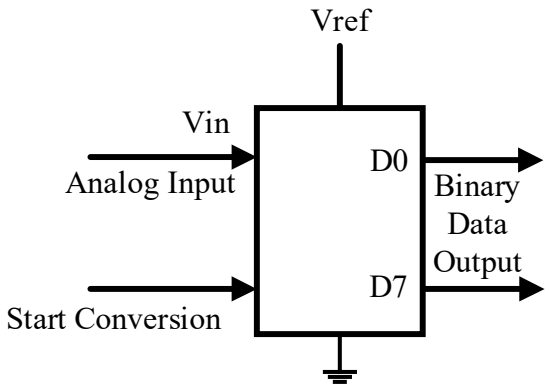
۹-۱ مقدمه

مبدل آنالوگ به دیجیتال یکی از پر استفاده‌ترین قطعات برای جمع‌آوری داده می‌باشد. در حالت کلی، دنیای خارج از میکروکنترلر ماهیت آنالوگ (پیوسته) دارد حال آن‌که کامپیوترهای دیجیتال از مقادیر باینری (گسسته) استفاده می‌کنند. دما، فشار (مایع یا گاز)، رطوبت و سرعت نمونه‌هایی از کمیت‌های فیزیکی هستند که هر روزه با آن‌ها سر و کار داریم. یک کمیت فیزیکی با استفاده از قطعه‌ای به نام مبدل (Transducer) به سیگنال الکتریکی (ولتاژ، جریان) تبدیل می‌شود. به این مبدل‌ها سنسور نیز می‌گوییم. سنسورها از دما، فشار، سرعت، نور و بسیای از کمیت‌های طبیعی یک خروجی ولتاژ (یا جریان) تولید می‌کنند. مبدل‌های آنالوگ به دیجیتال (ADC) و دیجیتال به آنالوگ (DAC) امکان ارتباط میکروکنترلر با سیگنال‌های آنالوگ را فراهم می‌کنند. مبدل ADC یک ولتاژ آنالوگ را گرفته و مقدار دیجیتالی برای آن فراهم می‌کند تا میکروکنترلر بتواند آن‌ها را خوانده و پردازش نماید. همچنین مبدل DAC یک مقدار دیجیتال را گرفته و متناسب با آن یک ولتاژ آنالوگ را تولید می‌کند. شکل ۹-۱ و شکل ۹-۲ شمای کلی اتصال سنسور به میکروکنترلر و بلوک دیگرام یک ADC هشت بیتی را نشان می‌دهند.

برخی از میکروکنترلرها دارای یک واحد داخلی مبدل آنالوگ به دیجیتال هستند که می‌تواند سیگنال‌های آنالوگ را بدون نیاز به مدارات جانبی به دیجیتال تبدیل کند.



شکل ۹-۱: اتصال میکروکنترلر به سنسور از طریق ADC



شکل ۹-۲: بلوک دیاگرام یک ADC هشت بیتی

۹-۲ مبدل آنالوگ به دیجیتال (ADC)

همان‌گونه که پیش‌تر بیان شد، برخی از میکرو کنترلرهای AVR دارای ADC داخلی هستند. برای اینکه بتوان ولتاژ آنالوگ را به داده دیجیتال تبدیل کرد از ADC استفاده می شود، به طور مثال برای ساخت آمپرمتر دیجیتال با سنسور Hall effect، از مبدلADC استفاده می‌کنیم؛ این سنسور میدان مغناطیسی ناشی از عبور جریان در یک سیم را به ولتاژ الکتریکی تبدیل می‌کند و این ولتاژ توسط ADC داخلی به داده دیجیتال تبدیل شده و جریان عبوری بر روی نمایشگر LCD به نمایش در می‌آید و کنترل لازم انجام می‌گیرد. به طور مثال: ساخت دماسنج با سنسورLM35 ، ترازوی دیجیتال با سنسور آنالوگ لودسل، متر دیجیتال با LVDT و ... نمونه های دیگری از کاربرد مبدل آنالوگ به دیجیتال می باشند. ویژگی‌های مبدل آنالوگ به دیجیتال ATmega16 به صورت زیر است:

- دقت ۱۰ بیتی و زمان تبدیل ۶۵ تا ۲۶۰ میکرو ثانیه
- ماکزیمم سرعت نمونه برداری ۱۵ kSPS
- ۸ کانال ورودی مالتی پلکسر به صورت Single Ended (منظور از Single Ended ، سیگنالی است که نسبت به زمین سنجیده می‌شود)
- ۷ کانال ورودی تفاضلی و ۲ کانال ورودی تفاضلی با اختیارات گین ۱۰× و ۲۰۰×
- اختیار تنظیم نتایج تبدیل از چپ برای خواندن از رجیستر دیتا
- محدوده ولتاژ ورودی ADC بین صفر تا Vcc
- قابلیت انتخاب ولتاژ مرجع ۲/۵۶ V داخلی
- مُد تبدیل اجرا آزاد یا مفرد
- قابلیت تریگر شدن اتوماتیک تبدیل مبدل، با منابع وقفه مختلف
- وقفه کامل شدن تبدیل ADC و مد Sleep (Noise Canceler) برای کاهش نویز

در میکروکنترلر ATmega16 از نقش دوم پایه های پورت A یعنی ADC0 تا ADC7 به‌عنوان ورودی مالتی‌پلکسر ADC استفاده می شود. در واقع در داخل تراشه AVR یک مبدل آنالوگ به دیجیتال ۱۰ بیتی وجود دارد و برای تبدیل دیتای آنالوگ باید یکی از کانال های ورودی ADC انتخاب شود این کار توسط برنامه صورت می گیرد. مبدل آنالوگ به دیجیتال استفاده شده در AVRها از نوع تقریب‌های متوالی (Successive Approximation) می‌باشند و قابلیت خواندن ولتاژ ورودی آنالوگ به صورت تک پایه است (Single Ended) و همچنین دیفرانسیلی به صورت ترکیبی از (ADC0 و ADC1) یا (ADC2 و ADC3) با بهره قابل برنامه ریزی را دارند. منظور از بهره (Gain)، تقویت سیگنال آنالوگ قبل از اعمال به ورودی ADC می‌باشد؛ در واقع این ضریب تقویت نیز داخلی است و می‌تواند سیگنال را به اندازه 0db(1X) و یا 20db(10X) و یا 46db(200X) تقویت نماید. نحوه انتخاب گین به همراه انتخاب ورودی در جدول ۹-۱ آورده شده است. توجه کنید اگر از ضریب بهره استفاده کنید دقت ADC کاهش می‌یابد. اگر از بهره های 1X و یا 10X استفاده شود دقت ۸ بیتی و اگر بهره 200X انتخاب شده باشد، دقت ۷ بیتی خواهد بود. ADC میکروکنترلر AVR دارای یک مدار مقایسه‌کننده Sample & Hold است که باعث می شود ولتاژ ورودی در حین تبدیل در سطح ثابتی قرار گیرد.

واحد ADC دارای زمین GND مجزا (پایه ۳۱ در میکروکنترلر ATmega16) و تغذیه AVCC مجزا (پایه ۳۰ در میکروکنترلر ATmega16) است و همچنین دارای پایه ولتاژ مرجع خارجی Vref می‌باشد. نحوه متصل کردن این پایه ها را در ادامه توضیح خواهیم داد. به این نکته توجه داشته باشید که تغذیه AVCC مبدل که از تغذیه ۵ ولتی مدار تأمین می شود، نباید بیشتر از ۰/۳ v± تغییرات داشته باشد. به همین منظور از یک سلف نیز استفاده می‌شود تا با تغییرات جریان ناگهانی مقابله نماید و علاوه بر آن به همراه یک خازن بکار برده می‌شود تا یک مدار فیلتر LC حذف نویز را ایجاد نماید.

جدول ۹-۱: انتخاب یکی از کانال‌های ADC

MUX 4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010(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	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x
11101		ADC5	ADC2	1x
11110	1.22 V (V_{BG})	N/A		
11111	0 V (GND)			

۹-۳ نتیجه عملیات تبدیل ADC

در صورتی که ADC تنظیم و بیت ADSC یک شده باشد، تبدیل آغاز می شود و پس از پایان تبدیل، وقفه کامل شدن عملیات ADC، در صورت فعال بودن، ایجاد می‌گردد. این نتایج که با دقت ۱۰ بیتی می باشد در دو رجیستر ADCL و ADCH قرار می‌گیرد و به طور پیش‌فرض، نتایج تبدیل از سمت راست تنظیم می شود. در ادامه نحوه تنظیم این دو رجیستر را شرح داده شده است.

نتایج تبدیل همچنین می تواند به صورت ۱۶ بیتی در برنامه نویسی خوانده شود، برای این منظور کافی است نتایج را از ADCW بخوانیم. در صورتی که از مُد Single Ended یعنی تک پایه استفاده کرده باشیم، ولتاژ ورودی را می‌توان از رابطه زیر برای تبدیل با دقت ۱۰ بیت بدست آورد:

$$ADC = \frac{V_{IN} \times 1024}{V_{REF}} \qquad (۹-۱)$$

که در آن:

ADC: نتایج دیجیتال تبدیل شده، به فرم ۱۶ بیتی است،

عدد ۱۰۲۴: به دلیل دقت ۱۰ بیتی است،

V_{REF}: ولتاژ مرجع مبدل است که در صورت انتخاب مرجع داخلی ۲/۵۶ v می‌باشد،
و V_{IN} ولتاژ ورودی بر حسب ولت است.

در صورتی که از کانال‌های تفاضلی استفاده شود، نتیجه تبدیل از رابطه زیر بدست می آید:

$$ADC = \frac{(V_{POS} \times V_{NEG}) \times Gain \times 1024}{V_{REF}} \quad (۹-۲)$$

در رابطه بالا، V_{POS} ولتاژ ورودی مثبت و V_{NEG} ولتاژ ورودی منفی، $Gain$ بهره انتخاب شده و V_{REF} ولتاژ مرجع انتخاب شده است. در این حالت نتایج تبدیل به فرم مکمل دو خواهد بود و از ۵۱۲- (0x200) تا ۵۱۱+ (0x1FF) تغییر می کند. توجه کنید که در این حالت بیت ADC9 تعیین‌کننده علامت است که اگر یک باشد نتیجه منفی و اگر صفر باشد نتیجه مثبت خواهد بود.

**** تذکر:** کانال های ورودی تفاضلی که در جدول ۹-۱ با گزینه (۱) مشخص شده‌اند، با بسته‌بندی PDIP آزمایش نشده‌اند. این ویژگی فقط برای قطعات در بسته بندی TQFP و MLF صدق می‌کند.

۹-۴ رجیسترهای مبدل آنالوگ به دیجیتال

برای استفاده از ADC می‌بایست رجیسترهای مربوط به آن را تنظیم نماییم. در صورت استفاده از ADC نقش دوم پایه انتخاب شده از پورت A تعیین می شود، به طور مثال وقتی که کانال ADC0 انتخاب شود، پایه بیرونی PA0 به عنوان ورودی سیگنال آنالوگ عمل می‌کند.

۹-۴-۱ رجیستر ADMUX (ADC Multiplexer Selection Register)

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

ـ بیت‌های MUX4:0 (Analog Channel and Gain Selection Bits)

توسط این بیت‌ها و بر اساس جدول ۹-۱ می توان ورودی کانال ADC را انتخاب کرد. همچنین توسط این بیت‌ها می‌توان بهره (Gain) ورودی در حالت تفاضلی را تعیین نمود.

ـ بیت ADLAR (ADC Left Adjust Result)

اگر این بیت یک شود آنگاه نتیجه ADC در حالت ۱۰ بیتی از سمت چپ تنظیم می‌گردد و اگر صفر باشد نتیجه از سمت راست تنظیم خواهد شد.

۹-۴-۲ رجیسترهای ADCH و ADCL (ADC Data Register)

Bit	15	14	13	12	11	10	9	8	
ADLAR=0	-	-	-	-	-	-	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Bit	15	14	13	12	11	10	9	8	
ADLAR=1	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	-	-	-	-	-	-	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

در پیکربندی فوق نحوه تنظیم نتایج تبدیل با بیت ADLAR مشخص می‌شود. پس از کامل شدن تبدیل، نتایج در دو رجیستر ADCH و ADCL قرار می‌گیرد. برای کانال‌های تفاضلی نتایج به صورت مکمل دو خواهد بود. تا زمانی که این دو رجیستر پس از تبدیل خوانده نشوند بازنویسی صورت نمی‌گیرد. همچنین لازم به ذکر است که در زمانی که نتایج از چپ تنظیم شده باشد و دقت بیشتر از ۸ بیت مورد نظر نباشد، خواندن ADCH کافی است. اما در حالت ۱۰ بیتی باید ابتدا ADCL و سپس ADCH خوانده شود. متذکر می شویم در نرم افزار کامپایلر C، دیتا به صورت ۱۰ بیتی از رجیستر ADCW خوانده می‌شود.

ـ بیت‌های REFS1:0 (Reference Selection)

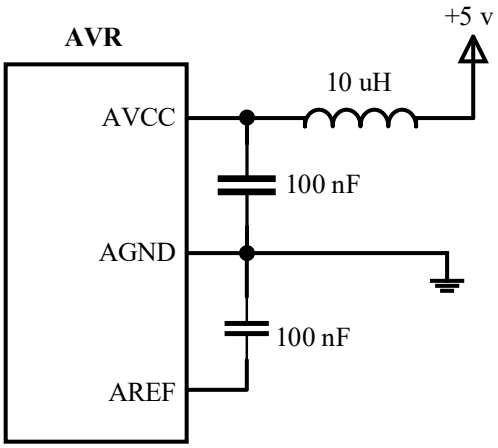
این بیت ها مطابق جدول ۹-۲ ولتاژ مرجع ADC را تعیین می کنند.

جدول ۹-۲: تعیین ولتاژ مرجعADC

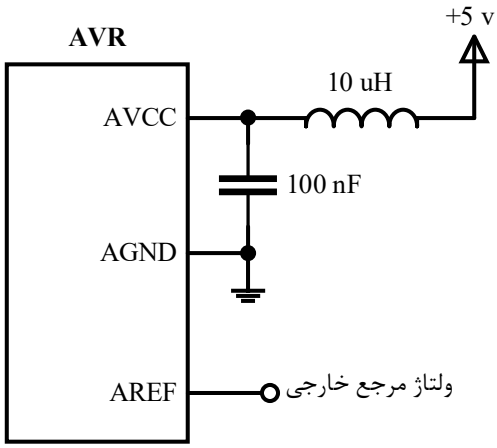
REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

ولتاژ ۲/۵۶ v مرجع داخلی، از قسمت Bandgap معماری درونی میکرو کنترلر تولید می شود. این ولتاژ در واقع با استفاده از ولتاژ تغذیه میکروکنترلر توسط نیمه هادی‌ها ایجاد می گردد.

در صورت استفاده از ولتاژ مرجع ۲/۵۶ v داخلی و یا استفاده از مرجع ۵ v+ توسط پایه AVCC باید پایه‌های ADC طبق شکل ۹-۳ متصل گردد و اگر قصد استفاده از ولتاژ مرجع خارجی را داشته باشید از شکل ۹-۴ می‌توانید استفاده نمایید.



شکل ۹-۳: چگونگی استفاده از ولتاژ مرجع داخلی



شکل ۹-۴: چگونگی استفاده از ولتاژ مرجع خارجی

۹-۴-۴ رجیستر ADCSRA (ADC Control and Status Register A)

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

ـ بیت‌های ADPS2:0 (ADC Prescaler Select)

توسط این بیت‌ها می‌توان تقسیم فرکانسی ADC را انتخاب کرد. در واقع این بیتها سرعت تبدیل را تعیین می‌کنند. فرکانس ADC از کلاک سیستم تأمین می شود. در جدول ۹-۳ نحوه تنظیم این بیت‌ها برای ضریب تقسیم سرعت فرکانسی مبدل آنالوگ به دیجیتال تعیین شده است.

جدول ۹-۳: انتخاب ضریب تقسیم فرکانسیADC

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

ـ بیت ADIE (ADC Interrupt Enable)

زمانی که این بیت به همراه بیت I در رجیستر وضعیت (SREG) یک گردند وقفه ADC پس از کامل شدن عملیات تبدیل، ایجاد می گردد.

ـ بیت ADIF (ADC Interrupt Flag)

زمانی که عملیات تبدیل ADC کامل می‌شود این بیت به نشانه کامل شدن تبدیل و بازنویسی در رجیستر ADCW یک می‌شود که در صورت فعال کردن وقفه ADC توسط بیت ADIE با یک شدن بیت ADIF روتین یا تابع وقفه اجرا می‌شود.

ـ بیت ADATE (ADC Auto Trigger Enable)

با یک کردن این بیت، مُد تحریک خودکار مبدل آنالوگ به دیجیتال فعال می‌گردد. در این مد با لبه مثبت سیگنال تحریک‌کننده که توسط بیت‌های ADTS2:0 واقع در رجیستر SFIOR تعیین می گردد، تبدیل آغاز می شود.

ـ بیت ADSC (ADC Start Conversion)

در حالت تبدیل منفرد (Single Conversion) این بیت در ابتدای هر تبدیل باید یک شود و تا انتهای تبدیل یک باقی می‌ماند و سپس توسط سخت‌افزار صفر می‌گردد ولی در حالت اجرای آزاد (Free Running) تنها برای اولین شروع باید این بیت یک گردد.

ـ بیت ADEN (ADC Enable)

با یک کردن این بیت ADC فعال می‌گردد و با صفر کردن آن ADC غیر فعال می‌شود.

۹-۴-۵ رجیستر SFIOR (Special Function IO Register)

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

ـ بیت‌های ADTS2:0 (ADC Auto Trigger Source)

اگر بیت ADATE در رجیستر ADCSRA یک شود، مقادیر این بیت‌ها منبع تحریک ADC را تعیین می‌کنند. تبدیل ADC به طور خودکار در لبه مثبت سیگنال تحریک کننده آغاز می شود. حالات مختلف این سه بیت و عملکرد هر یک از آن‌ها در نشان داده شده است.

جدول ۹-۴: انتخاب منبع تحریک‌کننده ADC

ADPS2	ADPS1	ADPS0	Trigger Source
0	0	0	Free Running mode
0	0	1	Analog Comparator
0	1	0	External Interrupt Request 0
0	1	1	Timer/Counter0 Compare Match
1	0	0	Timer/Counter0 Overflow
1	0	1	Timer/Counter1 Compare Match B
1	1	0	Timer/Counter 1 Overflow
1	1	1	Timer/Counter 1 Capture Event

۹-۵ مراحل تنظیم مبدل آنالوگ به دیجیتال

۱- پایه کانال ADC مورد نظر از پورت A را در حالت امپدانس بالا قرار می دهیم.

۲- با یک کردن بیت ACD در رجیستر ACSR مقایسه کننده آنالوگ را خاموش می‌کنیم.

۳- با تنظیم بیت های ADTS2:0 در رجیستر SFIOR می‌توانیم منبع تحریک‌کننده ADC را انتخاب کنیم. اگر در این قسمت منبع خاصی برای تحریک در نظر گرفته نشود به این معنی است که خود کاربر در برنامه ADC را برای شروع تبدیل، تحریک می کند.

۴- توسط بیت‌های MUX4: 0 در رجیستر ADMUX کانال ورودی مبدل ADC را از نوع تک پایه (Single Ended) و یا از نوع دیفرانسیلی انتخاب می‌کنیم و همچنین توسط بیت های REFS1:0 در رجیستر ADMUX ولتاژ مرجع مبدل آنالوگ به دیجیتال را تعیین می کنیم.

۵- یک کردن بیت ADEN در رجیستر ADCSRA مبدل آنالوگ به دیجیتال را فعال می نماییم. با یک کردن بیت ADIE در رجیستر ADCSRA وقفه ADC را فعال می‌کنیم و توسط بیت های ADPS2:0 در رجیستر ADCSRA سرعت تبدیل یا سرعت نمونه‌برداری را انتخاب می کنیم. در واقع با یک تقسیم فرکانسی مجزا از کلاک سیستم استفاده می‌شود.

۹-۵-۱ خواندن مبدل آنالوگ به دیجیتال به روش Polling

در صورتی که از وقفه ADC استفاده نکنیم می توانیم از تابع زیر در برنامه استفاده نماییم.

```
unsigned int read_adc(unsigned char adc_input) {
ADMUX = adc _input | ( ADC_VREF_TYPE & 0xff);        تعیین کانال آنالوگ ورودی
delay_us (10);          این تأخیر برای پایداری ولتاژ ورودیADC، نیاز می باشد
ADCSRA|=0x40;           شروع تبدیل آنالوگ به دیجیتال
while ((ADCSRA & 0x10)==0);      تست پرچم کامل شدن تبدیلA / D ، برای یک شدن
   ADCSRA|= 0x10;       پاک کردن (صفر کردن) پرچم بواسطه نوشتن یک منطقی در آن
   return ADCW;         برگشت ۱۰ بیتی رجیستر ADC به عنوان خروجی تابع
}
```

به‌طور نمونه اگر در برنامه بنویسیم: X=adc_read(2) مقدار داده تبدیل شده از کانال سوم را خوانده‌ایم.

۹-۶ شرح آزمایش

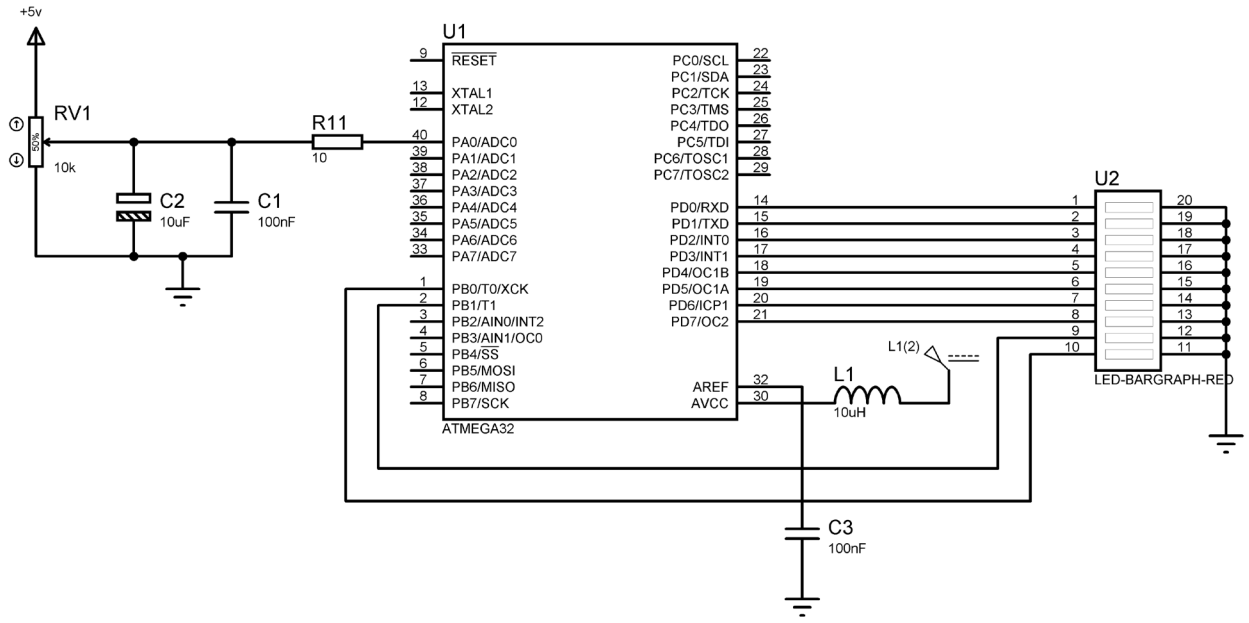
۹-۶-۱ آزمایش اول

می‌خواهیم برنامه ای بنویسید که ولتاژ سر وسط متغیر یک پتانسیومتر ۱۰ کیلواهم که به ورودی ADC0 متصل است و می‌تواند بین ۰ تا ۵ ولت تغییر کند را اندازه‌گیری کند و سپس داده‌های دیجیتال تبدیل شده روی ۱۰، دیود نورانی (LED) که به پورت‌های D و B توسط مقاومت ۲۲۰ اهم متصل هستند نمایش دهد. همچنین از روش وقفه استفاده در این قسمت از آزمایش استفاده شده است.

با توجه اینکه سیگنال تک ورودی است از نوع Single Ended استفاده می‌کنیم و همچنین سرعت نمونه‌برداری را متوسط و kHz ۲۵۰ با کریستال MHz ۸ بکار می گیریم. وقفه ADC را فعال می‌کنیم تا بعد از کامل شدن تبدیل، داده های دیجیتال را بر روی LEDها، نمایش دهیم و در برگشت از وقفه مجدداً ADC را برای شروع تبدیل جدید تحریک می‌کنیم و در هر مرحله یک تأخیر ms ۲۵۰ در نظر می‌گیریم تا نتایج بهتری داشته باشیم.

```
#include <mega32.h>          معرفی میکروکنترلر استفاده شده
#include <io.h>
#include <Delay.h>           معرفی کتابخانه تأخیر زمانی
interrupt [ADC_INT] void adc_isr(void){
    PORTD = ADCW;            قرار دادن ۸ بیت پایینی رجیستر مبدل در پورت خروجی D
    PORTB = (ADCW>>8);       قرار دادن ۸ بیت بالایی رجیستر مبدل در پورت خروجی B
    delay_ms(250);           تأخیر هر مرحله برای تبدیل مجدد
    ADCSRA = 0xCD;           شروع تبدیل آنالوگ به دیجیتال با یک کردن بیت ADSC
}
void main(){                 تابع اصلی برنامه
    PORTA = 0x00;            مقدار اولیه PORTA را صفر قرار می‌دهیم
    DDRA = 0x00;             با صفر کردن جهت دیتا، ورودی‌های PORTA در حالت امپدانس بالا قرار می‌گیرند
    PORTD = 0x00;            هشت LED کم ارزش خاموش هستند
    DDRD = 0xff;             تعریف PORTD به عنوان خروجی
    PORTB = 0x00;            دو LED پر ارزش خاموش هستند
    DDRB = 0xff;             تعریف PORTB به عنوان خروجی
    ACSR = 0x80;             مقایسه کننده آنالوگ داخلی غیر فعال است
    SFIOR = 0x00;            منبع تحریک کننده خاصی انتخاب نشده است
    ADMUX = 0x40;            انتخاب کانال ADC0 و تعیین ولتاژ مرجع ۵ ولت
    ADCSRA = 0x8D;           فعال کردن ADC و وقفه مربوط به آن و انتخاب تقسیم فرکانسی CLK/32
    #asm("sei")              فعال کردن وقفه همگانی
       ADCSRA = 0xCD;        شروع تبدیل با یک شدن بیتADSC
    while (1);               حلقه بی نهایت
}
```

در شکل ۹-۵ شماتیک برنامه فوق را مشاهده می‌فرمایید. همان‌گونه که مشاهده می‌شود ۱۰ تا LED به پورت های D و B متصل گردیده است. توجه شود که اگر از LED به جای بارگراف LED استفاده می‌کنید باید LEDها توسط مقاومت‌های ۲۲۰ اهمی به زمین متصل گردند. از آنجایی که از ولتاژ مرجع ۵ v برای این برنامه استفاده شده است پایه های ADC طبق مدار شکل ۹-۵ بسته شده است. ما در این برنامه، ولتاژ مرجع مبدل را ۵ ولت انتخاب کرده‌ایم زیرا محدوده تغییرات ولتاژ خروجی ولوم یا مقاومت متغیر استفاده شده بین صفر تا ۵ ولت می‌باشد. به‌طور مثال در پروژه دماسنج با سنسور LM35 که در آزمایش دوم آورده شده است، از ولتاژ مرجع ۲/۵۶ v داخلی استفاده کرده‌ایم چون خروجی سنسور LM35 ولتاژی در حدود میلی‌ولت تولید می‌کند که کوچکتر از ولتاژ مرجع می‌باشد. خازن عدسی C1 را در ورودی برای حذف نویز و خازن C2 را برای صاف‌کردن ولتاژ استفاده کردیم؛ شاید در برخی پروژه‌ها قرار دادن C2 اختلال‌آمیز باشد. مقاومت R1 را به‌منظور مهار جریان برگشتی ناشی از دشارژ خازن C2 در نظر می‌گیریم.

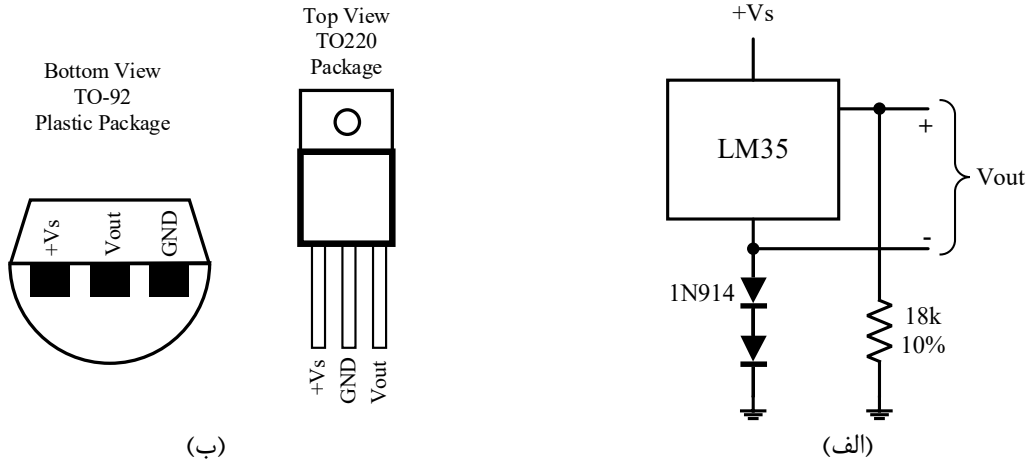


شکل ۹-۵: شماتیک مدار آزمایش اول

۹-۶-۱ آزمایش دوم، دماسنج با استفاده از سنسور LM35

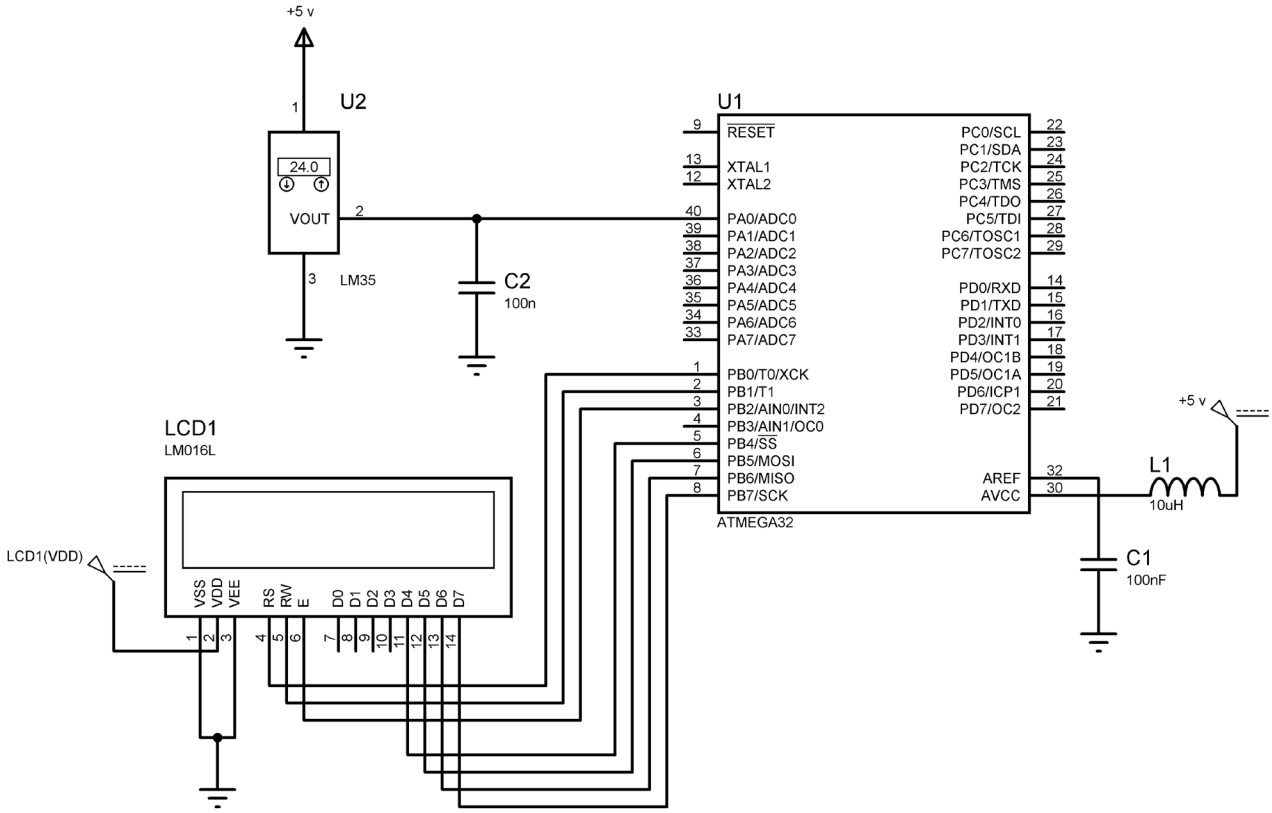
یکی از کاربردهای مبدل آنالوگ به دیجیتال داخلی میکروکنترلAVR ، می‌تواند خواندن داده آنالوگ سنسور و تبدیل آن به داده دیجیتال جهت کنترل برای سیستم مورد نظر باشد. در این آزمایش از یک سنسور خطی دما و ارزان قیمت استفاده شده است. ولتاژ خروجی این سنسور به ازای افزایش یا کاهش یک درجه سانتی‌گراد، به طور خطی mv ۱۰ تغییر می‌کند. یعنی در دمای ۲۵ درجه سانتی‌گراد، خروجی این سنسور ۲۵۰ میلی‌ولت است. در این آزمایش از ولتاژ مرجع داخلی ۲/۵۶ ولت و همچنین کانال اول ADC استفاده شده است. علت قرار دادن سلف µH ۱۰ و خازن nF ۱۰۰ متصل به پایه‌های مبدل ADC میکروکنترلر AVR، برای فیلتر کردن نویز و نوسانات تغذیه می‌باشد.

در شکل ۹-۶(ب) نمای زیر سنسور LM35 نشان داده شده است و شکل ۹-۶(الف) نحوه بایاس سنسور برای اندازه‌گیری دما بین محدوده ۵۵- تا ۱۵۰+ را نشان می‌دهد.



شکل ۹-۶: فرم بسته‌بندی و چگونگی استفاده از سنسور دمای LM35

برنامه‌ای بنویسید که در مدار شکل ۹-۷ داده سنسور دمای LM35 متصل به PORTA.0 را خوانده و مقدار آن را بر روی LCD متصل به PORTB نمایش دهد.



شکل ۹-۷: شماتیک مدار آزمایش دوم مربوط به دماسنج با استفاده از سنسور LM35

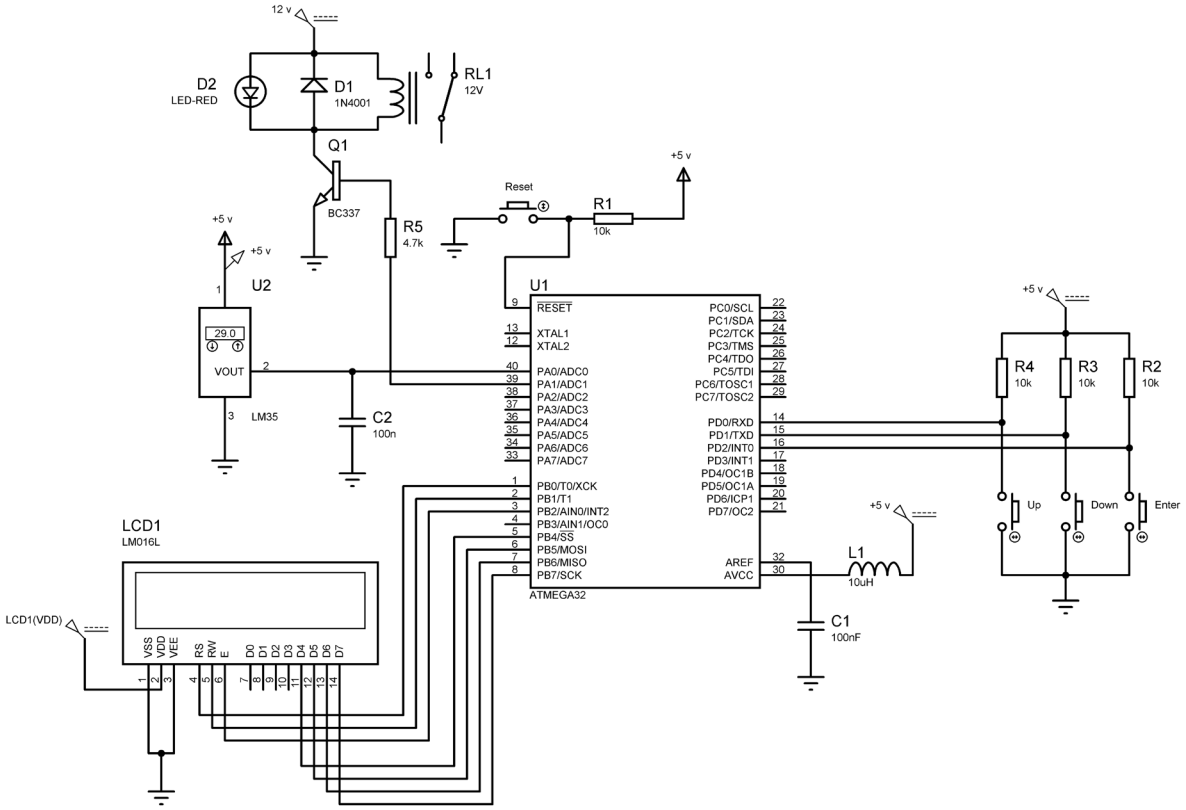
۹-۷ پرسش‌ها

۱- مدار شکل ۹-۸ را در نظر بگیرید. برنامه را به‌گونه‌ای بنویسید که توسط آن بتوان دما را تنظیم نمود و اگر دما از آستانه معینی که توسط کاربر تنظیم می‌شود، تجاوز کرد یک LED را روشن شود و یک رله را وصل گردد. مجموعه رله و LED به PORTA.1 متصل شده است. در این پرسش از سه کلید فشاری به شرح زیر استفاده شده است:

کلید Up: با فشرده نگه داشتن این کلید دمای تنظیم‌شونده افزایش می‌یابد

کلید Down: با فشرده نگه داشتن این کلید دمای تنظیم‌شونده کاهش می‌یابد.

کلید Enter: با فشردن این کلید، دما تنظیم می‌شود.



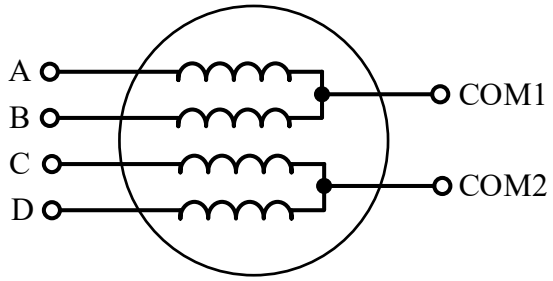
شکل ۹-۸: شماتیک مدار پرسش مربوط به سیستم کنترل دما با استفاده از سنسور LM35

۲- برنامه‌ای بنویسید که در آن، یک شکل موج پالسی با پهنای پالس۵۰٪ توسط میکروکنترلر AVR تولید شود. مدار را به‌گونه‌ای طراحی کنید که با تغییر یک پتانسومتر که به PORTA.0 متصل است دامنه شکل موج تولید شده نیز تغییر نماید. برای این کار از ADC میکروکنترلر کمک بگیرید. همچنین شکل موج تولید شده را روی اسیلوسکوپ مشاهده نمایید.

آزمایش ۱۰ – کنترل موتور پله‌ای با میکروکنترلر AVR

۱۰-۱ مقدمه

موتوری که به‌ازای پالس‌های الکتریکی، حرکت دورانی ایجاد می‌کند و دارای زاویه چرخش معینی می‌باشد موتور پله‌ای (Stepper Motor) نام دارد. این موتور چون می‌تواند در یک زاویه خاص قفل شود از کاربردهای گوناگونی برخوردار است از جمله می‌توان به استفاده آن در پرینترها، روبات‌ها، دیسک چرخان، آسانسور و ... اشاره نمود. شکل ۱۰-۱ پیکربندی سیم پیچ های استاتور موتور پله‌ای را نشان می‌دهد.



شکل ۱۰-۱: آرایش سیم‌پیچی‌های استاتور

نوع معمول موتور پله‌ای دارای چهار فاز و یک هسته متحرک مغناطیسی می‌باشد. این نوع موتورها دارای ۵ یا ۶ سیم می‌باشند که ۴ تا از سیم‌ها برای استاتور و ۲ تای آن، پایه مشترک می‌باشد که باید به مثبت تغذیه وصل شوند. در بعضی از موتورها سرهای مشترک می‌بایست از بیرون اتصال داده شوند.

برای پیدا کردن سرهای مشترک از یک مولتی متر دیجیتال استفاده کنید. همان‌طور که در پیکربندی سیم‌پیچ‌های موتور پله‌ای مشاهده می‌کنید باید بین A و B اهمی وجود داشته باشد مسلما A و B هر یک با اهم کمتری به COM1 وصل هستند و پایه های C و D نیز نسبت به COM2 همین وضعیت را خواهند داشت بنابراین خود شما باید به‌طور قراردادی پایه‌ها را بیابید زیرا سیم های موتور پله‌ای نام‌گذاری نشده اند. البته با یک منبع تغذیه ۵ ولتی نیز می‌توانید به سیم‌ها پالس دهید و از طریق جهت چرخش، آن‌ها را بیابید.

جدول ۱۰-۱ نحوه ارسال الکتریکی را به موتور پله‌ای نشان می‌دهد و آرایشی ترتیبی دارد یعنی برای اینکه موتور بچرخد می‌بایست ترتیب پالس ها رعایت شود.

جدول ۱۰-۱: ترتیب ارسال دیتا به موتور پله‌ای

راست گرد	گام پله	A	B	C	D	چپ گرد
⬇	1	1	0	0	0	⬆
	2	0	1	0	0	
	3	0	0	1	0	
	4	0	0	0	1	

جدول ۱۰-۲: زوایای استاندارد موتورهای پله‌ای و روابط آن‌ها

پله در دور SPT	زاویه پله استاندارد بر حسب درجه θs
500	0.72°
200	1.8°
180	2.0°
144	2.5°
72	5.0°
48	7.5°
24	15°

$SPS = \frac{RPM \times SPT}{60}$

SPS : پله در ثانیه

SPT : پله در دور

RPM : دور در دقیقه

$N_{pulse} = \frac{360°}{\theta s}$

N_{pulse} : تعداد پالس‌ها

$T_{full} = T_d \times N_{pulse}$

T_{full} : زمان یک دور کامل

T_d : زمان تأخیر پالس

۱۰-۲ روش نیم‌پله

در این روش دقت پله دو برابر می شود یعنی درجه استاندارد آن به نصف کاهش پیدا می‌کند و در نتیجه دسترسی به درجه‌های غیر استاندارد را ممکن می‌سازد. برای اینکه موتور پله ای به روش نیم پله کار کند باید مطابق جدول ۱۰-۳ پالس‌دهی کنیم.

جدول ۱۰-۳: ترتیب ارسال پالس‌ها به موتور پله‌ای در روش نیم‌پله

راست گرد	گام پله	A	B	C	D	چپ گرد
⬇	1	1	0	0	0	⬆
	2	1	1	0	0	
	3	0	1	0	0	
	4	0	1	1	0	
	5	0	0	1	0	
	6	0	0	1	1	
	7	0	0	0	1	
	8	1	0	0	1	

۱۰-۳ تراشه ULN2003A

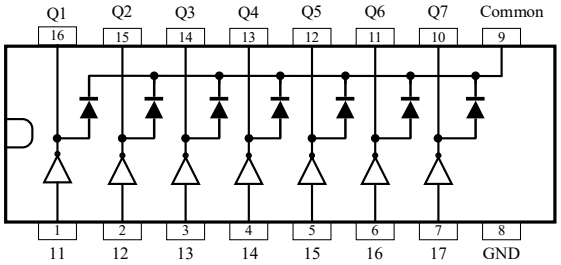
برای راه اندازی موتور پله‌ای توسط میکرو کنترلر نیاز به جریان دهی مناسب می‌باشد بنابراین می‌توان از ترانزیستور، یا آی‌سی‌های مخصوص استفاده کرد. در این آزمایش از IC راه‌انداز ULN2003A استفاده شده است. ماکزیمم جریان این تراشه ۵۰۰ mA می‌باشد. لازم به ذکر است که این تراشه کلکتور باز است بنابراین باید از مقاومت های بالاکش (Pull-Up) استفاده کرد و توجه داشته باشید که دیتای داده شده به آن NOT می‌شوند.

برای اینکه از القای برگشتی سیم پیچ های موتور جلوگیری شود باید از دیودهای هرزگرد استفاده کرد که در داخل این تراشه دیودهای هرز گرد وجود دارند.

* **تذکر:** اگر از موتور پله ای با جریان بالاتر استفاده می‌کنید می‌توانید از ترانزیستورهای قدرت یا MOSFET با جریان‌دهی بالا استفاده کنید و برای هر ترانزیستور یک دیود هرزگرد در نظر بگیرید

مشخصات ULN2003A:

- ۷ تقویت کننده زوج دارلینگتون
- ماکزیمم جریان خروجی ۵۰۰ mA
- تحمل ولتاژ خروجی تا ۵۰ ولت
- دارای دیود هرزگرد جهت ولتاژ القایی
- قابلیت موازی کردن خروجی‌ها جهت بالا بردن تحمل جریان های بالاتر



ULN2001A	General Purpose, DTL, TTL, PMOS, CMOS
ULN2002A	14 – 25v, PMOS
ULN2003A	5v, TTL, CMOS
ULN2004A	6-15v, PMOS, CMOS

شکل ۱۰-۲: مشخصات و فرم بسته بندی ULN2003A

۱۰-۴ شرح آزمایش

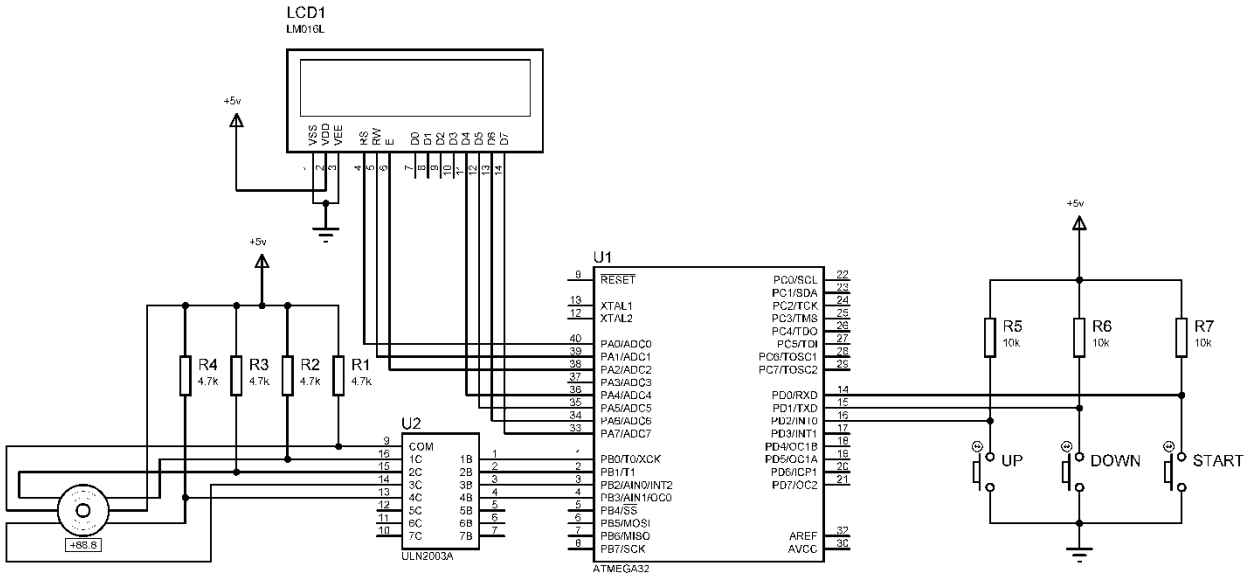
موتور پله‌ای استفاده شده در این آزمایش دارای دقت ۱/۸ درجه می‌باشد و زمان پالس‌دهی ۵۰ میلی ثانیه در نظر گرفته شده است، بنابراین زمان یک دور کامل ۱۰ ثانیه خواهد بود. اگر نیاز داشته باشید سرعت موتور پله‌ای را کنترل کنید باید زمان پالس‌دهی را تغییر دهید. در این آزمایش می‌تونید درجه مورد نظر که ضریبی از ۱/۸ است را توسط کلید فشاری‌های Up و Down بین ۳۶۰+ تا ۳۶۰- تنظیم کنید و با زدن کلید فشاری Start موتور شروع به حرکت می‌کند. اگر درجه تنظیم شده مثبت باشد راست گرد و اگر منفی باشد چپ‌گرد خواهد بود. برنامه کنترل موتور پله‌ای در ادامه آورده شده است.

```
#include <mega32.h>
#asm
  .equ __lcd_port=0x1B;
#endasm
#include <lcd.h>
#include <delay.h>
#include <stdio.h>
#define  Up     PIND.2
#define  Down   PIND.1
#define  Start  PIND.0
float grade=0.0;
unsigned char state=0;
                                        //Display
void display_lcd}()
char display_buffer[33];
lcd_clear();
sprintf(display_buffer,"Degree=%+3.1f\xdf",grade);
lcd_gotoxy(0,1);
lcd_putsf("Rotation=~");
if(state==1) lcd_putsf("Right");
```

```
if(state==2) lcd_putsf("Left ");
lcd_gotoxy(0,0);
lcd_puts(display_buffer);
{
_____________________________________________//
void stepper_m(){
unsigned int i,pulse;
unsigned char x,y;
x=0b01110111;
if(grade<0.0){
  pulse=grade/-1.8;
  state=2;
  display_lcd();
  for(i=0;i<pulse;i++){
    PORTB=x;
    y=x;
    x=x>>1;
    y=y<<7;
    x=(x|y);
    delay_ms(50);
  }
}
else{
  pulse=grade/1.8;
  state=1;
  display_lcd();
  for(i=0;i<pulse;i++){
    PORTB=x;
    y=x;
    x=x<<1;
    y=y>>7;
    x=(x|y);
    delay_ms(50);
  }
}
}
_____________________________________________//up
void inc_degree(){
  if(grade<360){
    grade+=1.8;
    display_lcd();
    delay_ms(200);
  }
}
_____________________________________________//Down
void dec_degree(){
  if(grade>-360){
    grade-=1.8 ;
    display_lcd();
    delay_ms(200);
```

```
  }
}
_____________________________________//
void main(){
  DDRB=0xFF;
  DDRD=0X00;
  lcd_init(16);
  display_lcd();
  while(1){
    if(Up==0)     inc_degree();
    if(Down==0)   dec_degree();
    if(Start==0)  stepper_m();
  };
}
```

این برنامه را به دقت بررسی نموده سپس مدار نشان داده شده در شکل ۱۰-۳ را پیاده‌سازی نموده و با اعمال ورودی‌های مختلف از درستی عملکرد آن اطمینان حاصل کنید.



شکل ۱۰-۳: شماتیک مدار کنترل موتور پله‌ای

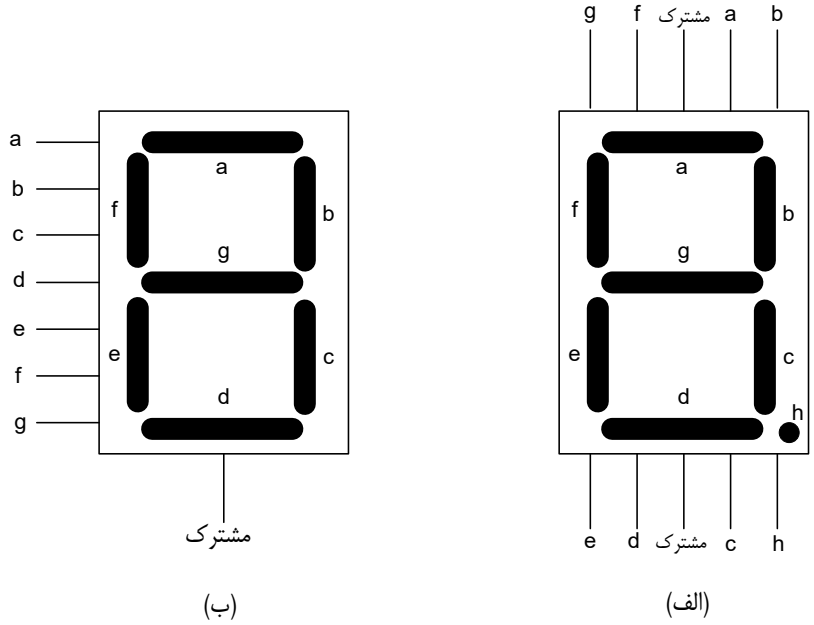
۱۰-۵ پرسش

۱- مدار شکل ۱۰-۳ را تغییر دهید به‌گونه‌ای که به جای سه کلید فشاری یک صفحه کلید قرار گیرد و با ورود زاویه توسط صفحه کلید و فشار دادن کلید مساوی یا هر کلید دلخواه دیگر، موتور پله‌ای به‌اندازه‌ای که کاربر توسط صفحه کلید تعیین کرده است به راست یا چپ گردش نماید. برنامه را به‌گونه‌ای بنویسید که اعداد به‌صورت مثبت و منفی توسط صفحه کلید دریافت شوند و متناسب با آن چرخش به راست یا چپ انجام گیرد.

آزمایش ۱۱ – پیاده سازی چراغ راهنمایی به کمک میکروکنترلر AVR

۱۱-۱ مقدمه

در این آزمایش، هدف، پیاده سازی یک چراغ راهنمایی به صورت ساده می باشد. برای نمایش اعداد از 7-segment کاتد مشترک استفاده شده است. در این نوع 7-segment پایه مشترک را به زمین متصل کرده و برای روشن کردن هر یک از LEDهای درون 7-segment پایه متناظر با آن LED را به یکِ منطقی (۵ ولت) متصل می نماییم. ارتباط بین پایه ها و LED ها در شکل ۱۱-۱ نشان داده شده است.

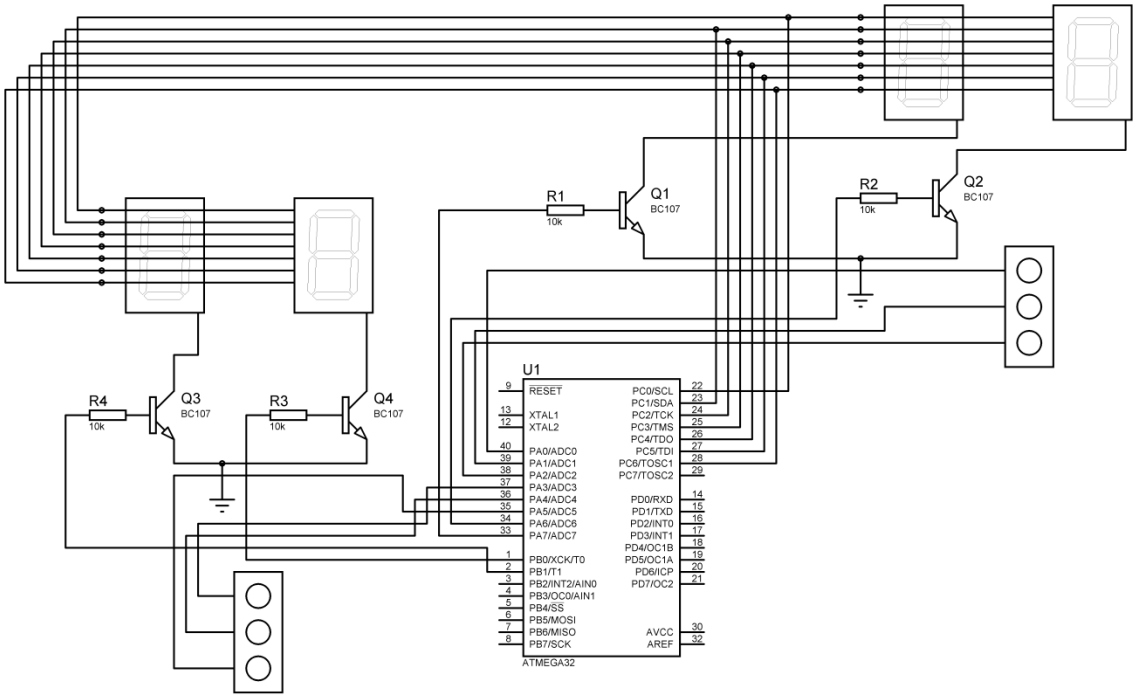


شکل ۱۱-۱: ارتباط بین پایه ها و LED ها در یک 7-segment

۱۱-۲ شرح آزمایش

برای مدار نشان داده شده در شکل ۱۱-۲ برنامه ای بنویسید که یک چراغ راهنمایی را پیاده سازی نماید؛ به این صورت که هرگاه 7-segment با رنگ سبز از ۹ به صفر شمارش می کند، چراغ راهنمایی مربوط به آن، سبز باشد و 7-segment با رنگ قرمز در کنار آن، خاموش باشد، همچنین در این حالت بایستی چراغ راهنمایی دیگر قرمز باشد و 7-segment قرمز رنگ مربوط به آن چراغ راهنمایی از ۹ به صفر شمارش نموده و 7-segment سبز خاموش باشد. پس از پایان شمارش بایستی جای آنها با یکدیگر عوض شود. البته برنامه را به گونه ای بنویسید که با پایان یافتن شمارش (از ۹ به صفر)، برای یک مدت زمانی کوتاه (مثلاً ۲ ثانیه) 7-segment های در حال شمارش در عدد صفر بمانند و چراغ راهنمایی که می خواهد به رنگ قرمز درآید، رنگ زرد را نشان دهد.

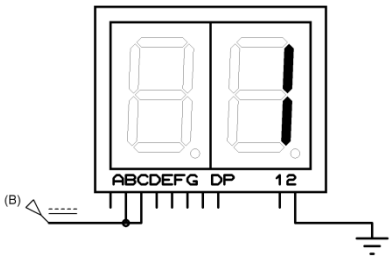
توجه: این مدار به گونه ای طراحی شده است که اگر Base هر یک از ترانزیستورهای BC107 به منطق یک (۵ ولت) متصل شود، 7-segment متصل به آن فعال می شود.



شکل ۱۱-۲: مدار به کار رفته برای پیاده سازی چراغ راهنمایی

۱۱-۳ پرسش

۱ ـ برنامه این آزمایش را برای حالتی بنویسید که از 7-segment های نشان داده شده در شکل ۱۱-۳ استفاده می‌شود. بدیهی است که در این حالت می توان شمارش را برای اعداد بزرگتر از ۹ نیز انجام داد. به عنوان نمونه شمارش را از ۳۰ به صفر انجام دهید. این نوع 7-segment نیز مشابه آنچه در این آزمایش به کار رفت، می باشد. برای فعال کردن 7-segment سمت چپ باید پایه یک، 7-segment سمت راست باید پایه ۲ و برای فعال کردن هر دو باید هر دو پایه ۱ و ۲ را به **زمین** متصل نمود. پایه های دیگر همانند یک 7-segment کاتد مشترک معمولی است. البته نوع آند مشترک آن نیز وجود دارد که اتصال پایه های آن به ۵ ولت و زمین برعکس حالتی است که در بالا شرح داده شد.



شکل ۱۱-۳: *7-segment* به کار رفته برای پرسش

پیوست الف ــ آشنایی با محیط برنامه‌نویسی نرم‌افزار CodeVision AVR

الف ــ ۱ آشنایی با محیط برنامه‌نویسی CodeVisionAVR

نرم‌افزار CodeVisionAVR، یک نرم افزار تخصصی برای رشته‌های برق و کامپیوتر می‌باشد. در واقع این نرم‌افزار یک کامپایلر برای زبان برنامه نویسی C می‌باشد که برای برنامه نویسی میکروکنترلرهای AVR از آن استفاده می‌شود. این برنامه، محیط برنامه‌نویسی و کامپایل کردن برنامه نوشته شده برای برنامه‌ریزی میکروکنترلر را فراهم می‌کند.

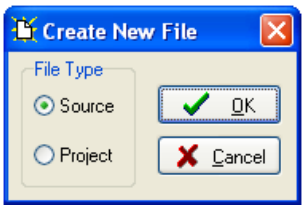
بسیاری از افراد حتی کسانی که رشته کامپیوتر می باشند با این نرم افزار به‌خوبی آشنا می‌باشند. آخرین نسخه این برنامه قدرت بسیار بیشتری پیدا کرده است و از طرفی مشکلات قبلی آن برطرف شده است. این برنامه در تمامی نسخه های ویندوز قابل نصب است.

در نرم‌افزار Codevision به دو طریق می‌توان برنامه دلخواه را ایجاد کرد که در ادامه به بررسی هر دو روش پرداخته خواهد شد:

۱- با ایجاد یک صفحه پروژه و نوشتن کل برنامه

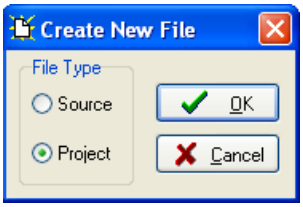
۲- با استفاده از Codewizard

در روش اول، ابتدا برنامه CodeVisionAVR را اجرا کرده سپس از منوی File گزینه New را کلیک می‌کنیم. در پنجره ظاهرشده گزینه Source را انتخاب کرده و صفحه ایجاد شده را Save می‌کنیم.



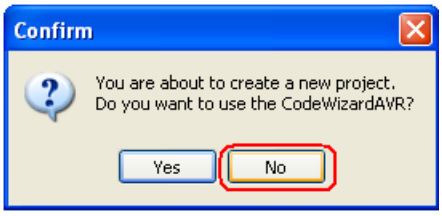
شکل الف-۱: ایجاد یک source جدید

دوباره با کلیک بر روی گزینه New یک Project ایجاد می‌کنیم.



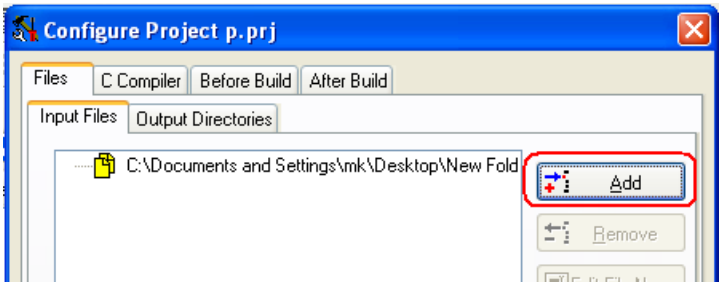
شکل الف-۲: ایجاد یک project جدید

در روش اول، چون نمی خواهیم از Codewizard استفاده کنیم، در پنجره Confirm که در شکل الف-۳ نشان‌داده شده است، گزینه No را انتخاب می‌کنیم.



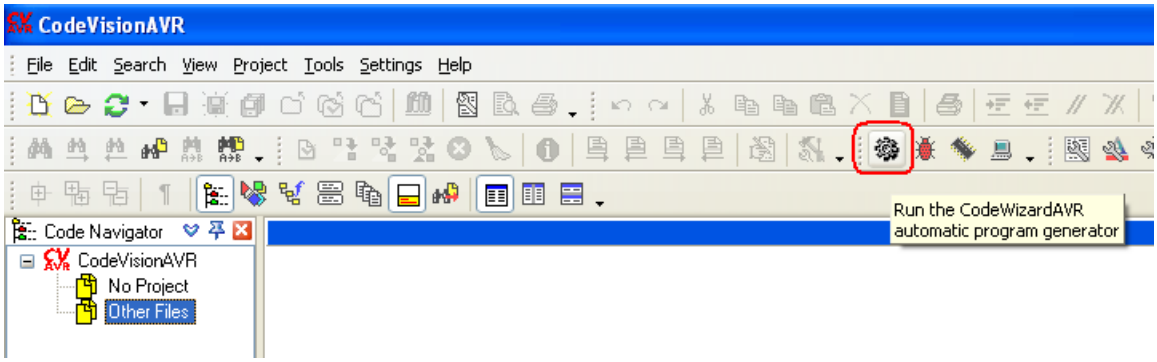
شکل الف-۳: پنجره Confirm

پس از Save این فایل در همان مسیر، در پنجره باز شده بر روی گزینه Add کلیک کرده و فایل Source ایجاد شده با پسوند C را به پروژه اضافه می‌کنیم. با استفاده از این پنجره IC مورد نظر، فرکانس کاری، امکانات جانبی و غیره را تعیین می‌کنیم.



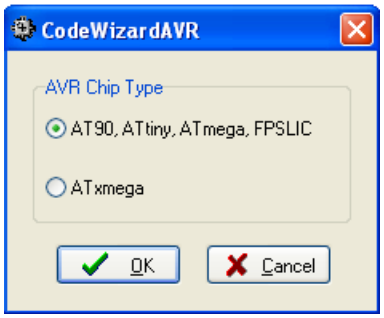
شکل الف-۴: اضافه کردن فایل source به فایل پروژه

در روش دوم که بهترین روش می‌باشد، پس از باز کردن نرم‌افزار، آیکن CodeWizard را کلیک می‌کنیم.



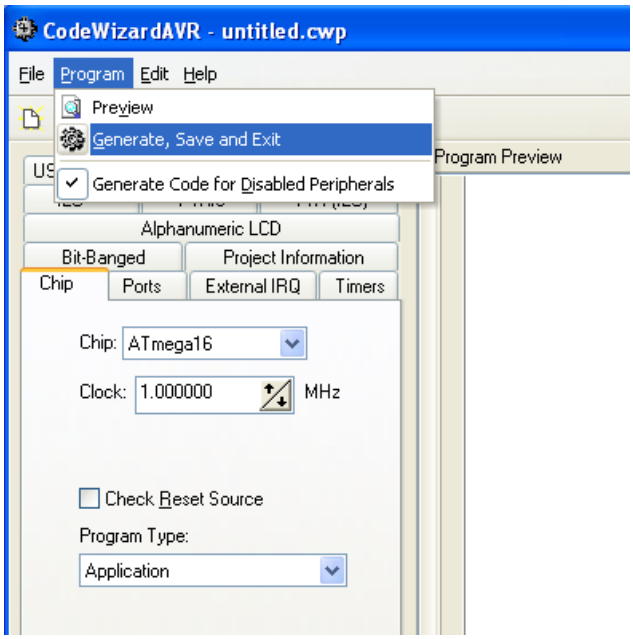
شکل الف-۵: استفاده از CodeWizard

در پنجره باز شده چون قصد استفاده از آی‌سی Xmega را نداریم گزینه اول را انتخاب می‌کنیم.



شکل الف-۶: انتخاب نوع IC

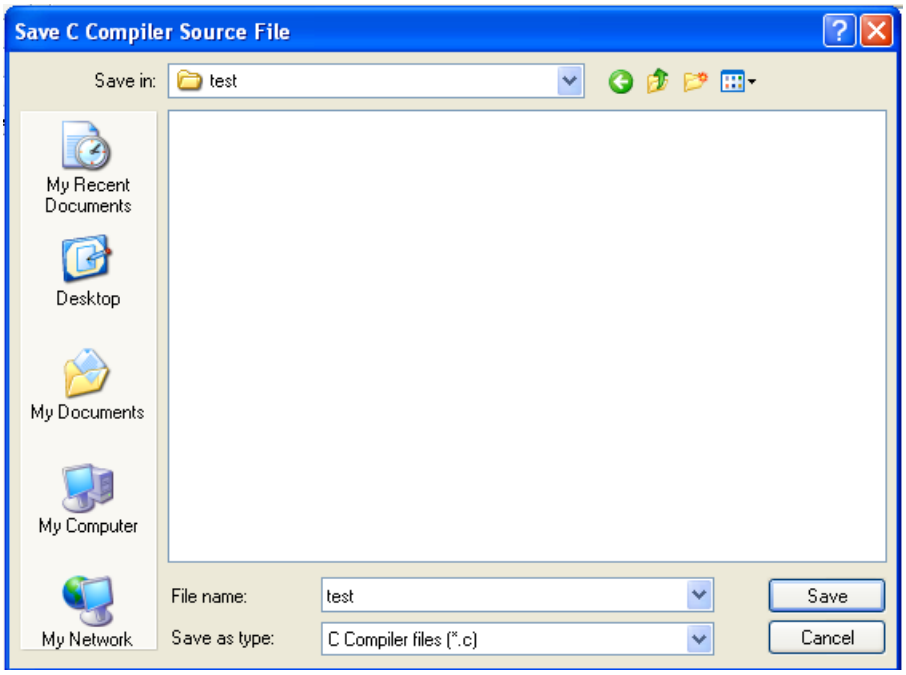
با انتخاب هر یک از آی‌سی‌های AVR امکانات آن به سربرگ‌ها افزوده می‌شود و برای فعال کردن هریک از امکانات از قبیل LCD کاراکتری، پورت سریال و ... کافی است به قسمت مربوطه برویم و تیک فعال‌سازی آن را اتخاب کنیم. همچنین یکی از روش های آشنایی با امکانات آی‌سی‌های AVR ، استفاده از CodeWizard است. پس از انتخاب امکانات مورد نظر و انجام تنظیمات مربوطه برای ساخت فایل پروژه، از منوی Program گزینه Generate, Save and Exit را کلیک می‌کنیم.



شکل الف-۷: ذخیره تنظیمات ایجادشده

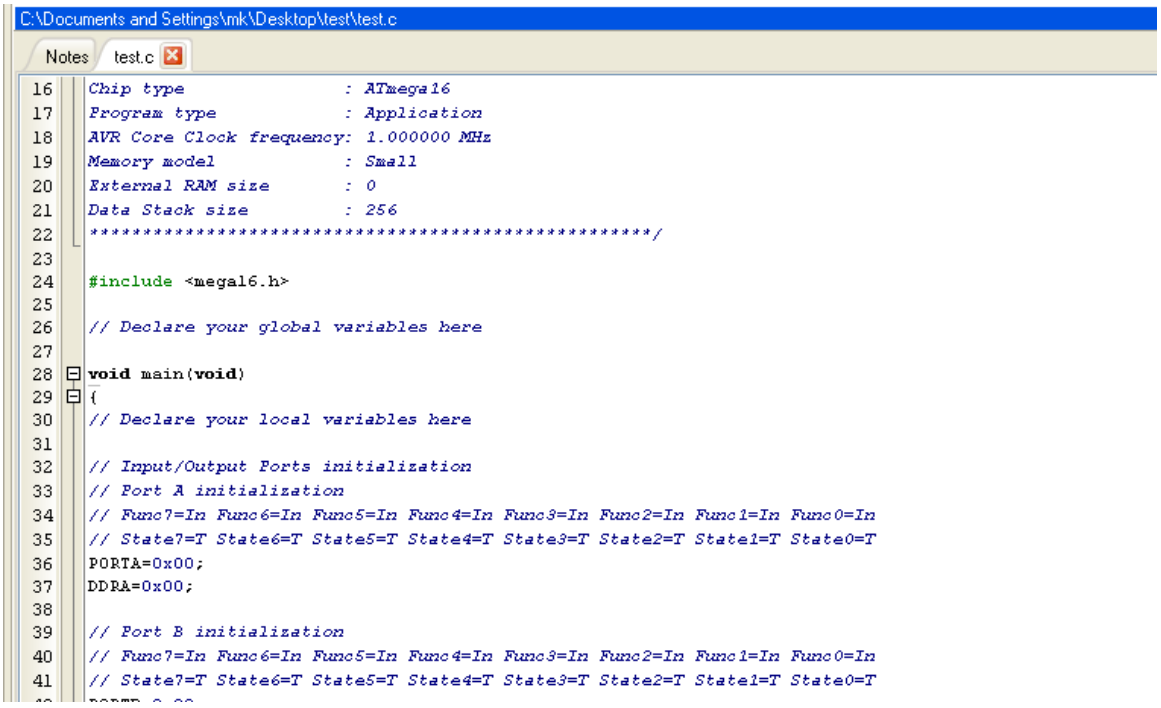
نرم افزار سه فایل برای پروژه، فایل C و فایل تنظیمات Codewizard ایجاد می کند و ما باید سه بار نام دلخواه را در آن وارد کنیم که بهتر است نام سه فایل مانند هم باشد.

** **نکته:** لازم است کلیه فایل‌هایی که می‌خواهیم ایجاد کنیم در یک پوشه مخصوص باشد تا از پراکندگی و اشتباه در جابجایی پروژه جلوگیری شود.



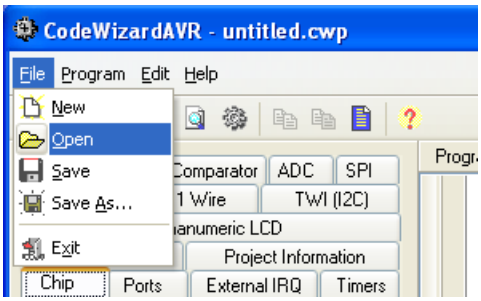
شکل الف-۸: ذخیره همه فایل‌ها در یک پوشه

پس از Save کردن مشاهده می‌کنید که تمام رجیسترها، کتابخانه‌های مربوطه، وقفه ها، تابع اصلی و حتی حلقه while(1) نوشته شده و شما فقط باید بدنه اصلی برنامه را به آن اضافه کنید.



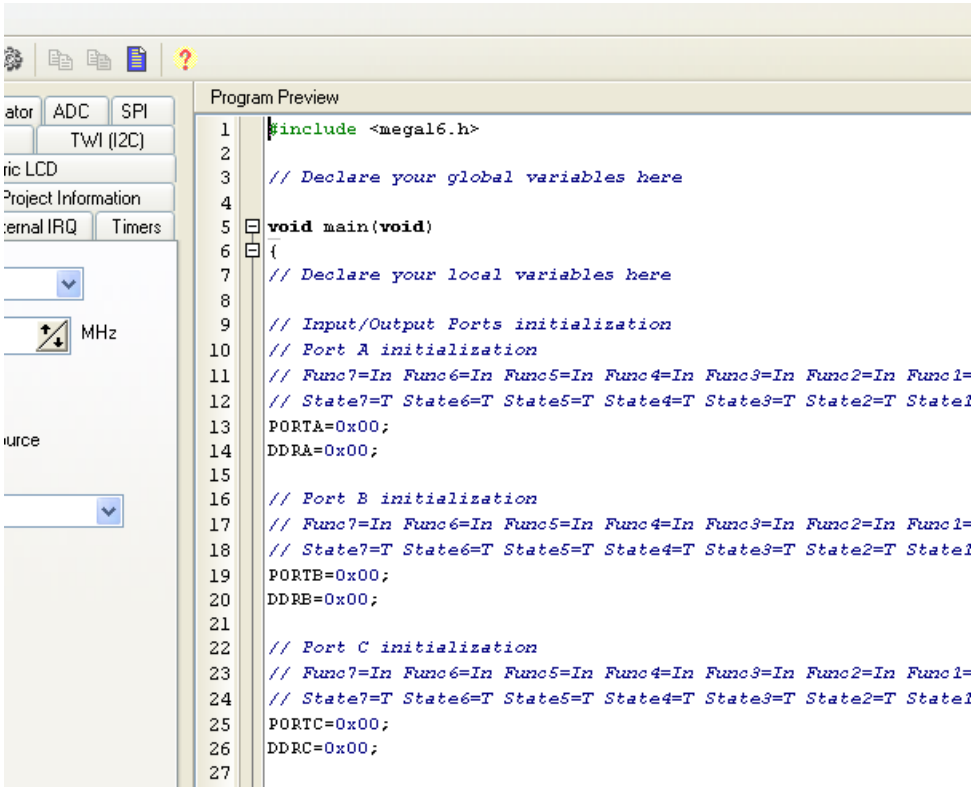
شکل الف-۹: محیط برنامه‌نویسی

اگر بخواهیم در خلال برنامه تغییراتی در رجیسترها اعمال کنیم و برای این کار به Codewizard نیاز داشته باشیم به این صورت عمل می‌کنیم که پنجره Codewizard را باز کرده و در آن از منو File گزینه Open را کلیک می‌کنیم.



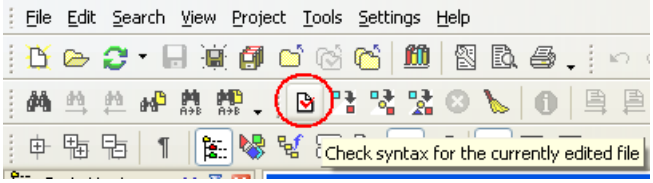
شکل الف-۱۰: مراجعه دوباره به CodeWizard

فایل Codewizard ایجاد شده برای برنامه با پسوند cwp. را باز می‌کنیم. در این حالت مشاهده می‌کنید که تمام تنظیمات انجام شده Load می‌شود. پس از تغییر تنظیمات از منو فایل گزینه Save را زده و سپس از منوی Program گزینه Program preview را انتخاب می‌کنیم. برنامه جدید طبق تنظیمات انجام شده در سمت راست صفحه ظاهر می‌شود. در اینجا رجیسترهایی را که تغییر کرده‌اند را کپی کرده و به جای مقادیر قبلی Paste می‌کنیم.



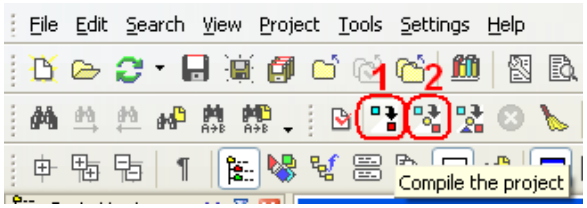
شکل الف-۱۱: مشاهده Program Preview

حین نوشتن برنامه اگر بخواهیم از وجود خطاها آگاه شویم گزینه Check syntax… را کلیلک می‌کنیم. این گزینه مواردی را که مقایر با شکل انشایی دستورات است، به ما گزارش می‌دهد.



شکل الف-۱۲: استفاده از Check syntax

پس از نوشتن برنامه، گزینه Compile the project سپس Build the project را کلیک می‌کنیم.



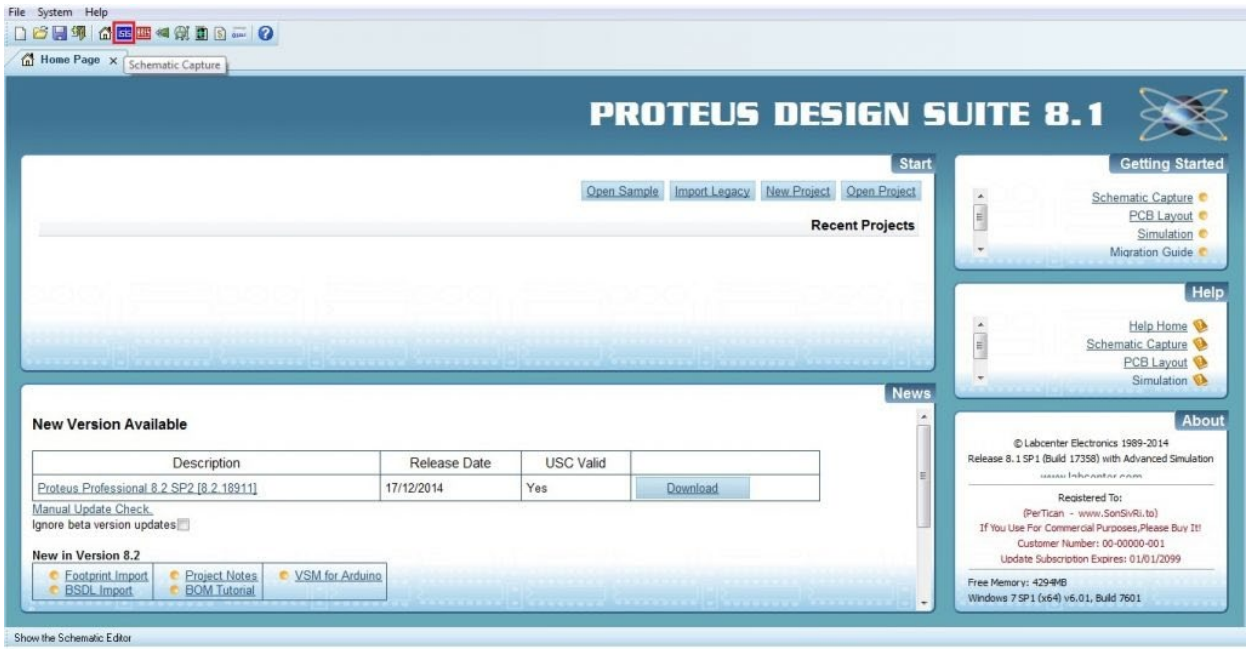
شکل الف-۱۳: کامپایل کردن برنامه نوشته‌شده

پیوست ب ــ استفاده از نرم‌افزار ISIS Proteus برای شبیه‌سازی مدارهای میکروکنترلری

نرم‌افزار ISIS Proteus بی‌شک یکی از قدرتمندترین نرم‌افزارها در زمینه شبیه‌سازی مدارها به‌ویژه مدارهای مبتنی بر میکروکنترلر و نیز طراحی PCB آن‌ها می‌باشد. این نرم افزار که محصول شرکت Lab center Electronics می باشد، به دو بخش ISIS وARES تقسیم می‌شود. از محیط ISIS برای ترسیم و تست مدار یا همان شبیه‌سازی و از محیط ARES برای تهیه نقشه PCB مدار تست شده با ISIS استفاده می‌شود. کتابخانه‌های بسیاری از قطعات الکترونیک جهت طراحی و شبیه‌سازی مدارهای الکترونیکی در این نرم‌افزار موجود است. کاربردهای این نرم افزار عبارتند از:

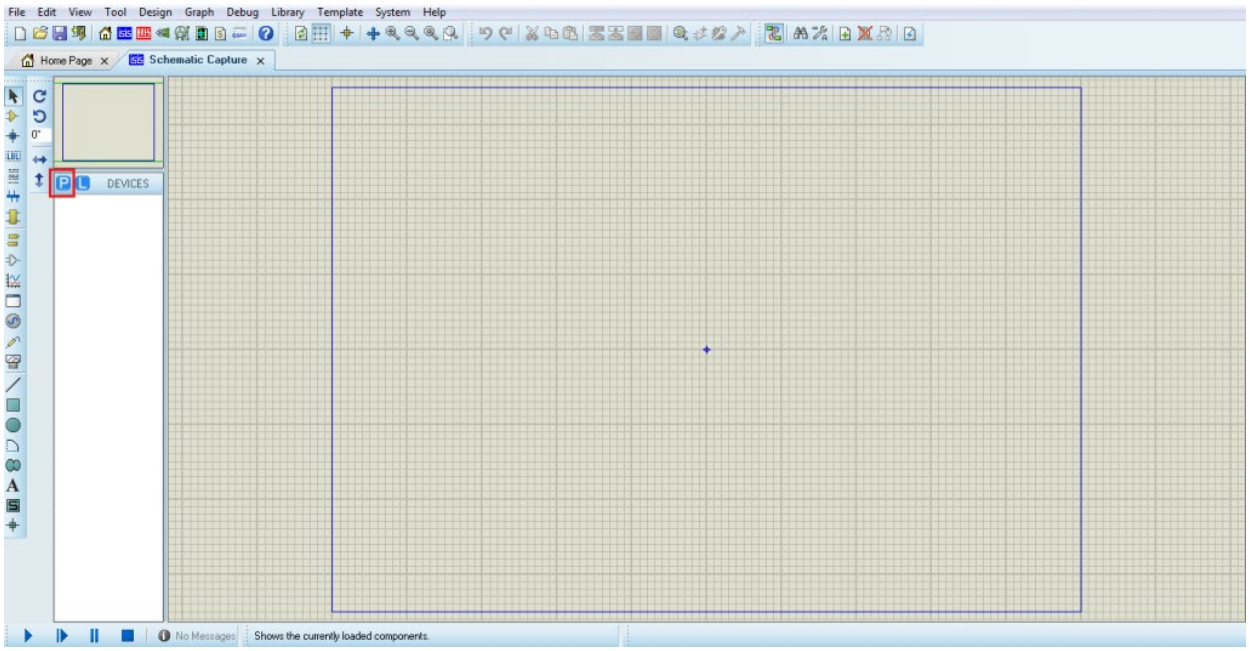
- شبیه سازی مدارهای آنالوگ و دیجیتال
- تست و آنالیز مدارها با استفاده از ابزار اندازه‌گیری مجازی مانند اسیلوسکوپ، مولتی متر، فانکشن ژنراتور و ...
- شبیه سازی تقریباً اکثر مدارهای مبتنی بر میکروکنترلرهای PIC، AVR و برخی از میکروکنترلرهای ARM
- امکان ایجاد و ویرایش قطعات الکترونیکی
- طراحی بردهای PCB یک تا ۱۶ لایه

پس از نصب نرم‌افزار بر روی آیکون آن در صفحه دسکتاپ کلیک کرده تا نرم افزار باز شود . زمانی که نرم افزار باز شد با پنجره نشان داده شده در شکل ب-۱ مواجه می‌شوید. در این مرحله بر روی آیکون آبی رنگ ISIS موجود در نوار ابزار بالایی که در شکل نیز مشخص شده است، کلیک کنید تا محیط Schematic Capture برای رسم مدار باز شود.



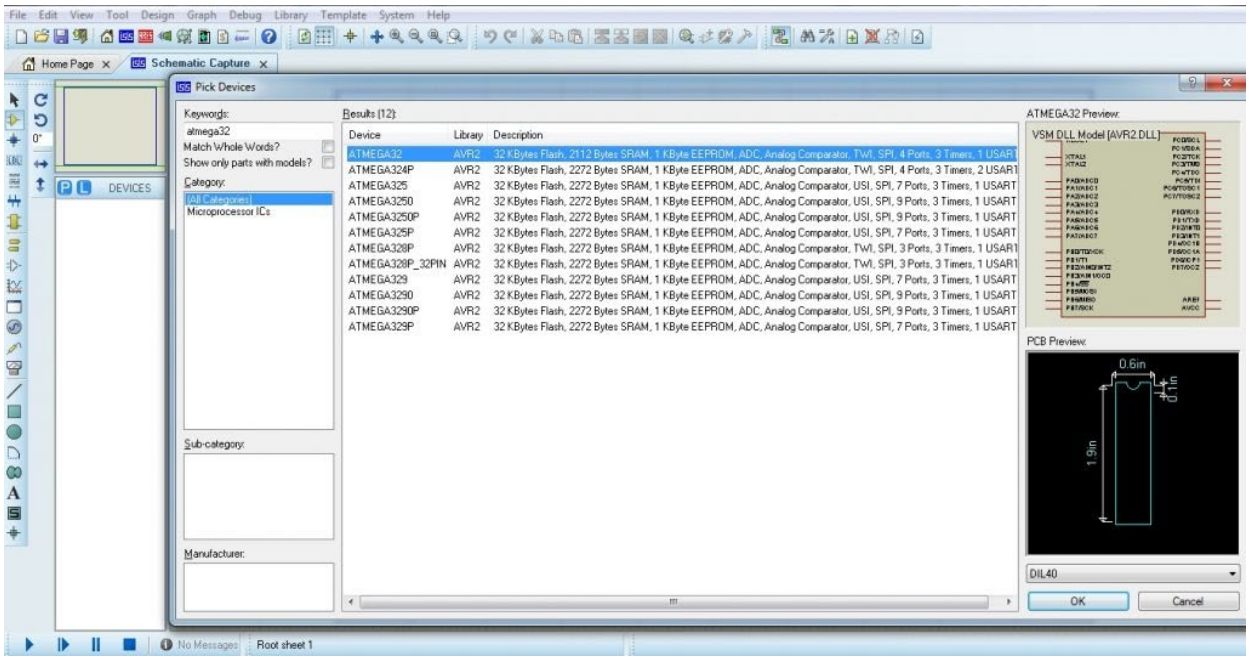
شکل ب-۱: پنجره نرم‌افزار به‌هنگام باز شدن

برای انتخاب المان های مداری باید روی آیکون P کلیک کنید تا صفحه جدیدی بنام Pick Devices باز شود. محل این آیکن در شکل ب-۲ نشان داده شده است.

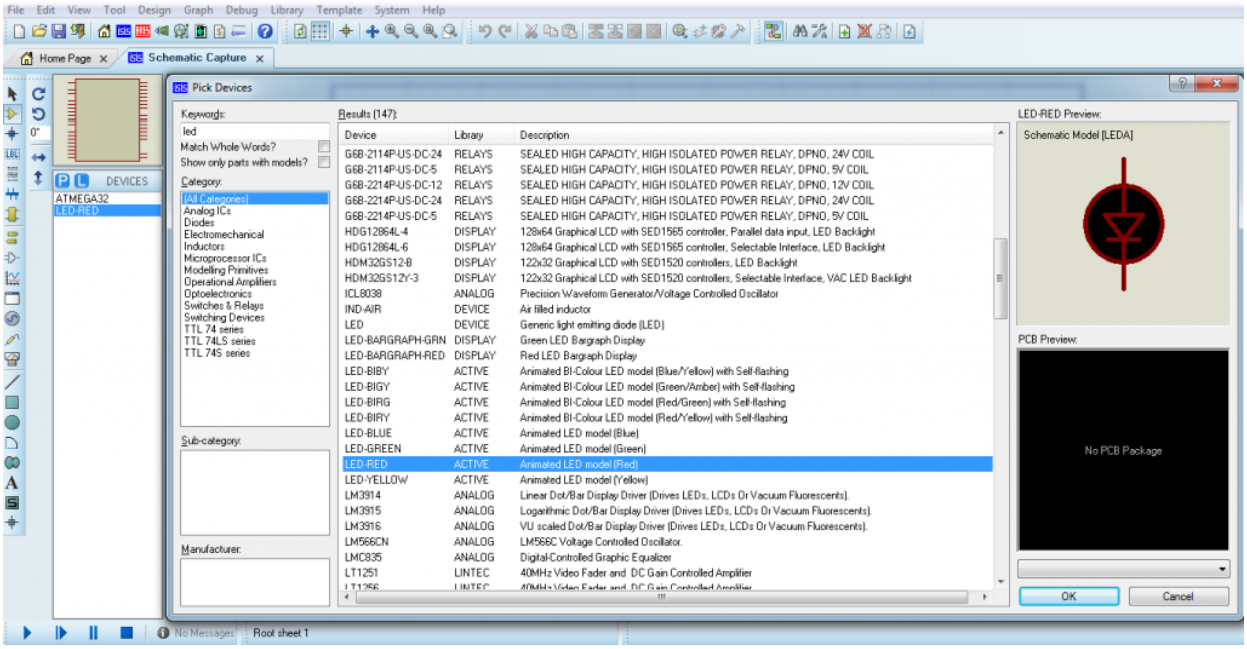


شکل ب-۲: آیکن P برای انتخاب المان‌های مداری

در این صفحه نام یا بخشی از نام هر المانی که نیاز داشته باشید را تایپ کرده و سپس روی نام قطعه دو بار کلیک کنید تا قابل استفاده گردد . برای مثال اگر به یک Atmega32 و یک LED نیاز داریم، نام آن‌ها را تایپ کرده و روی نام آنها دو بار کلیک می‌کنیم و در نهایت پنجره Pick Devices را با کلیک بر روی Ok می بندیم. مراحل اجرای کار را در شکل ب-۳ و شکل ب-۴ مشاهده می‌کنید.

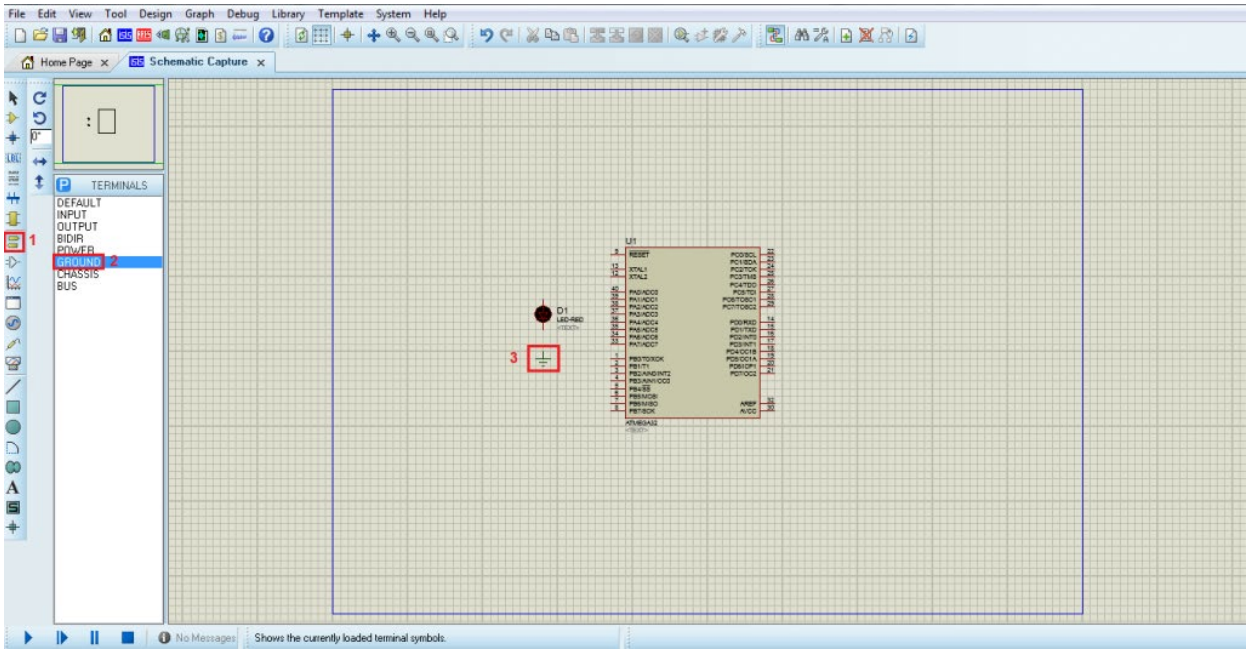


شکل ب-۳: مراحل اجرای کار برای انتخاب المان‌های مداری



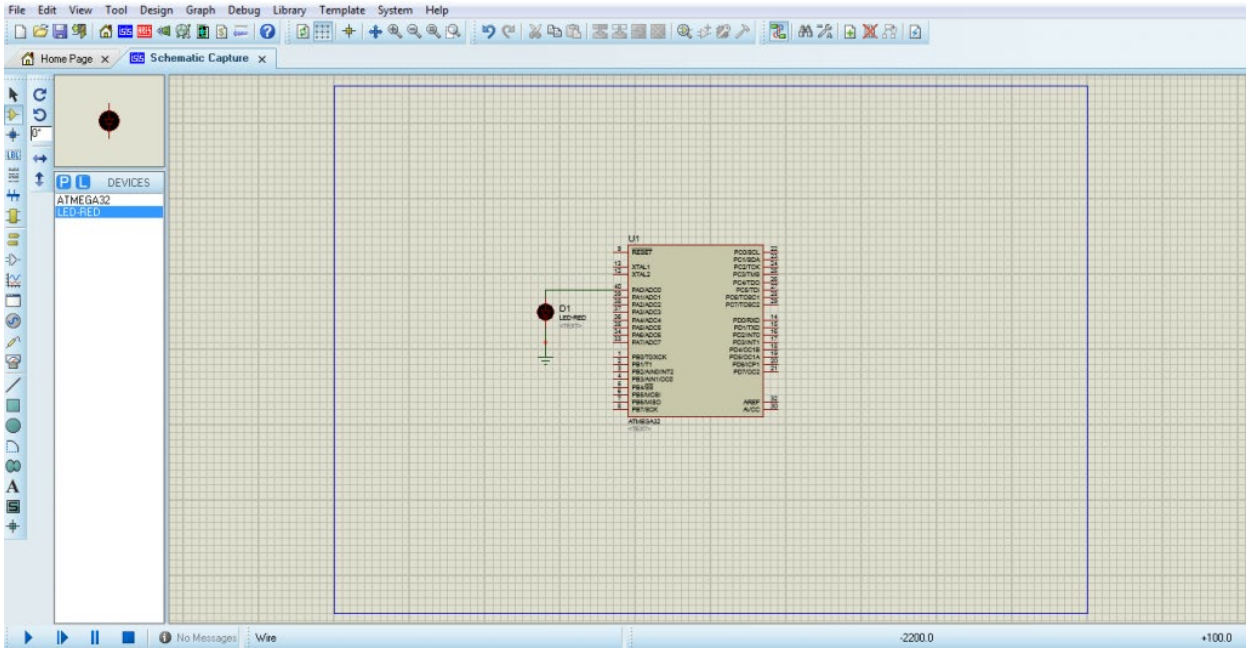
شکل ب-۴: مراحل اجرای کار برای انتخاب المان‌های مداری

با انجام مراحلی که در بالا بیان شد، زیر آیکون P قطعاتی که انتخاب کرده بودیم آورده شده است. با کلیک بر روی آنها میکروکنترلر و LED را درون کادر آبی رنگ صفحه اصلی در محل مناسب خود قرار داده و مدار را تکمیل می‌کنیم. اکنون به یک زمین (Ground) نیز احتیاج داریم که آن را با کلیک بر روی آیکن Terminals Mode موجود در نوار ابزار سمت چپ و سپس کلیک بر روی GROUND انتخاب کرده و در جای مناسب خود قرار می دهیم (شکل ب-۵).



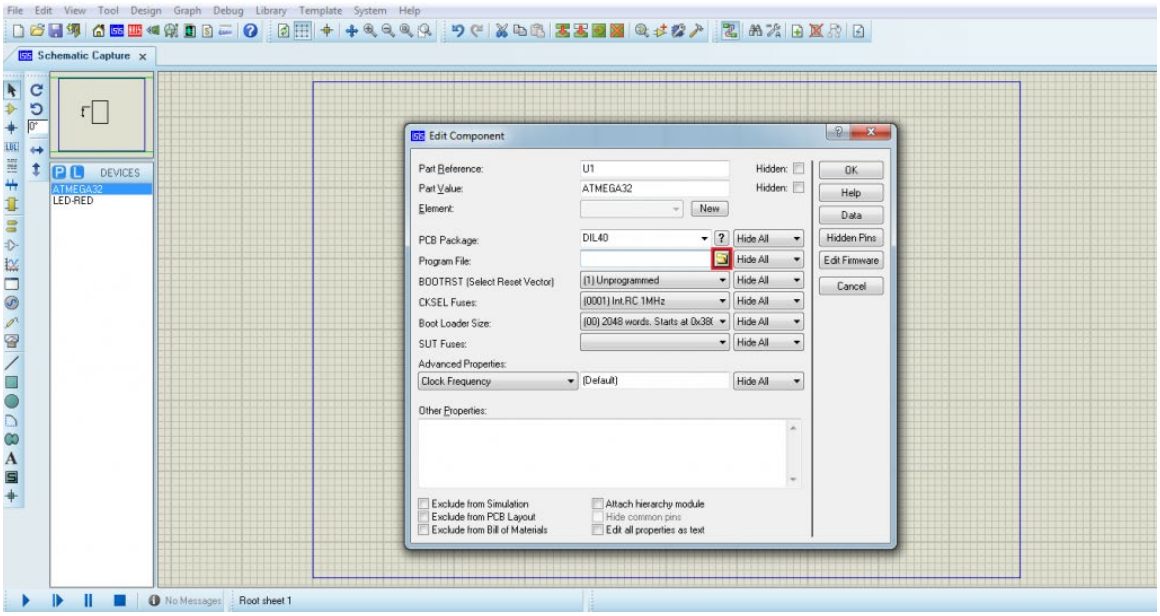
شکل ب-۵: انتخاب GROUND از منوی Terminal Mode

در نهایت نوبت به سیم کشی مدار می‌رسد. زمانی که نشانگر ماوس را در محل سیم کشی روی پایه (پین)های LED یا میکروکنترلر قرار می‌دهید، مداد سبز رنگی ظاهر می شود، در همین حال کلیک کنید و سیم کشی مدار را به صورت شکل ب-۶ تکمیل نمایید.



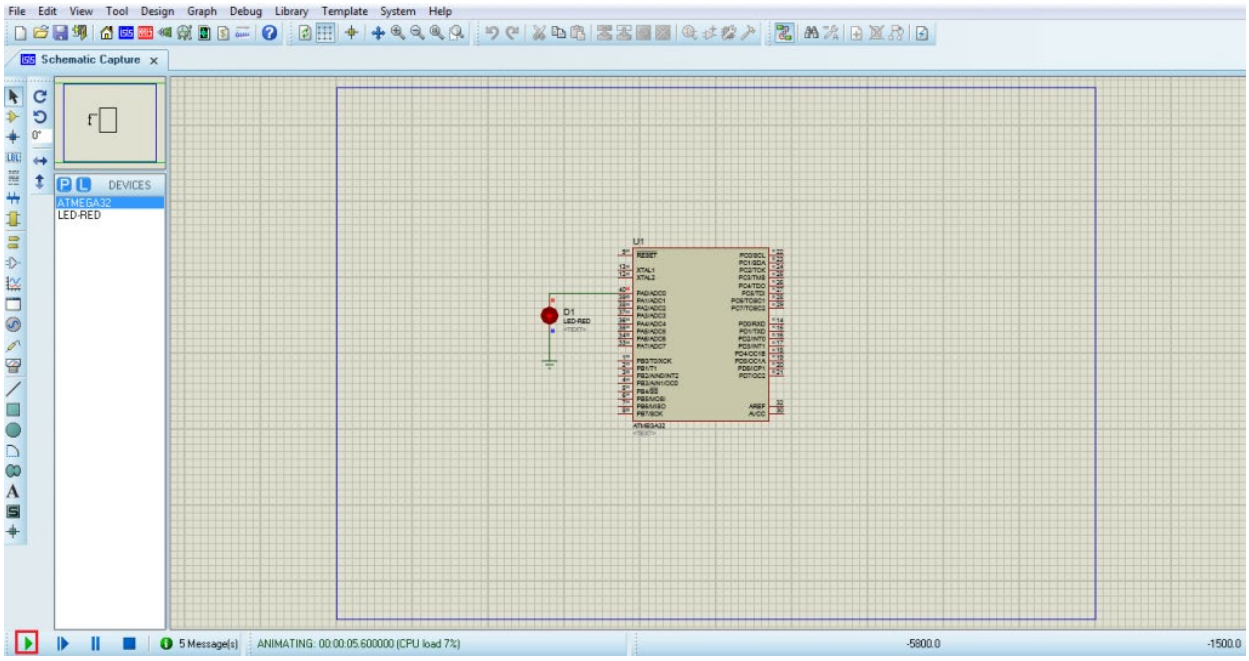
شکل ب-۶: تکمیل سیم‌کشی مدار

بعد از تکمیل مدار آن را از منوی File و گزینه Save با نام مناسب در یک پوشه جدید ذخیره نمایید. برای اینکه بتوان برنامه را بر روی آی‌سی ریخته و سپس اجرا کرد می‌بایست ابتدا باید فایل hex که از نرم افزار CodeVisionAVR تولید می‌شود را داشته باشیم. برای این منظور، روی میکروکنترلر دو بار کلیک کرده تا پنجره Edit Component باز شود. در این پنجره در قسمت Program File روی آیکون پوشه (Browse) کلیک کرده تا پنجره انتخاب فایل باز شود. حالا می‌بایست به مسیر برنامه‌ای که در CodeVisionAVR نوشته شده بروید و در آنجا داخل پوشه Exe شده و فایل با پسوند hex. را انتخاب کنید (شکل ب-۷).



شکل ب-۷: قرار دادن فایل hex بر روی میکروکنترلر

بعد از انتخاب فایل hex با دیگر تنظیمات کاری نداشته و پنجره Edit Component را و OK نمایید. با این کار برنامه نوشته شده به زبان C در نرم افزار CodeVisionAVR به داخل آی‌سی میکروکنترلر در نرم افزار پروتئوس منتقل می‌شود . حال برای شبیه سازی مدار روی دکمه Play پایین صفحه کلیک کرده تا شبیه سازی آغاز و مدار Run شود (شکل ب-۸).



شکل ب-۸: شبیه‌سازی مدار

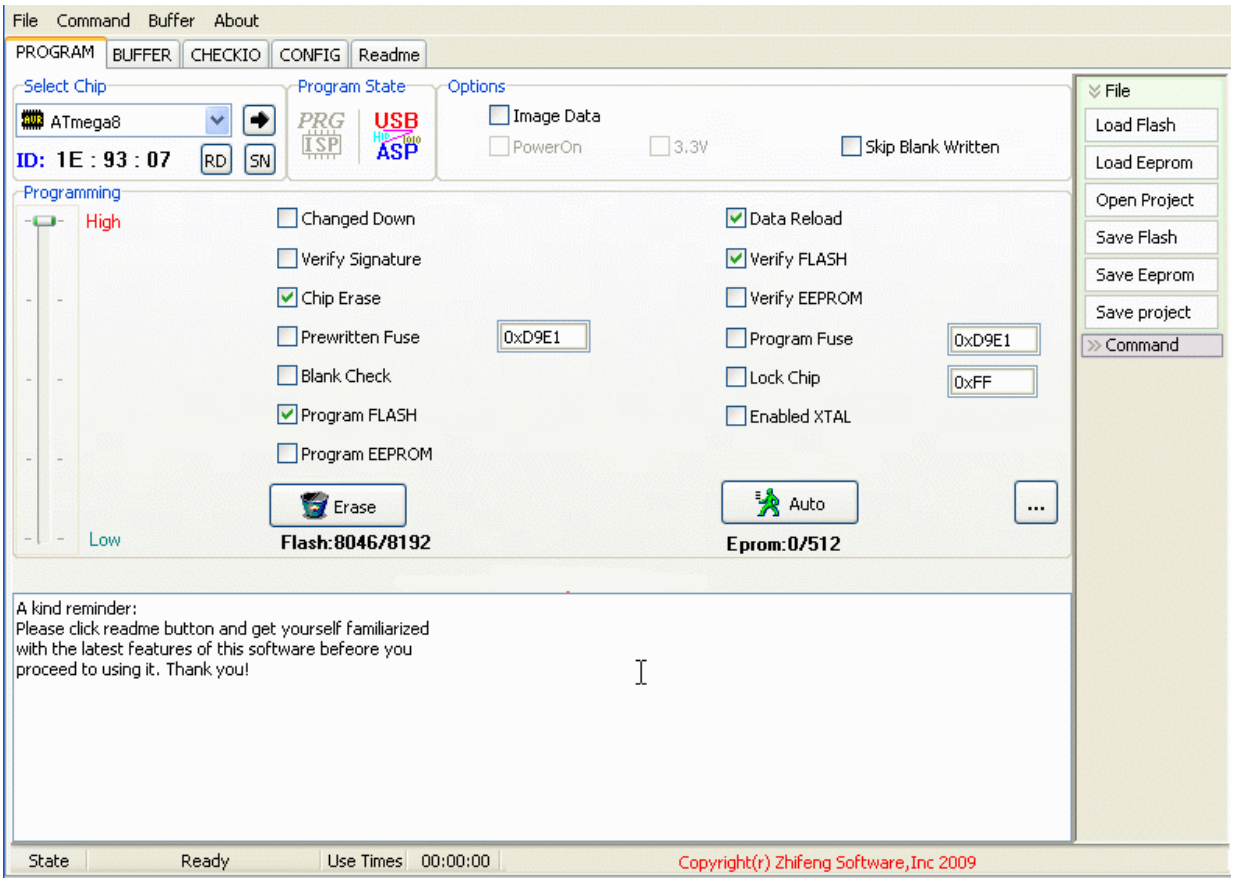
پیوست پ ــ پروگرام کردن میکروکنترلر با Progisp

پ-۱ معرفی نرم‌افزار Progisp

به منظور انتقال فایل های hex به میکروکنترلر نیاز به وسیله‌ای پروگرامر (programmer) می‌باشد و برای کار کردن با پروگرامر هم به یک نرم افزار واسط نیاز است. یکی از بهترین نرم افزارها برای این کار progisp است که از آن برای انتقال فایل‌های hex تولید شده توسط نرم افزارهایی مانند CodeVisionAVR یا Bascom-AVR به تراشه های AVR استفاده می‌شود. این نرم افزار می تواند با پروگرمر معروف USBASP به خوبی کار کند.

قابلیت‌های بسیار جالب و کلیدی در این نرم افزار از قبیل تنظیم فیوزبیت‌ها، انتقال فایل hex و فایل های EEPROM، خواندن حافظه فلش میکروکنترلر، قرار دادن میکروکنترلر در حالت Lock، تنظیم کریستال خارجی در انواع مُدهای کاری و بسیاری امکانات دیگر وجود دارد که توصیه می شود حتما از آن‌ها استفاده شود.

این نرم‌افزار نسخه‌های مختلف دارد که از نظر گرافیکی دارای تفاوت هستند اما اکثر گزینه‌های آن‌ها مشترک است. در اینجا نسخه ۱.۷.۲ معرفی شده است که محیط ساده‌تر و یکپارچه‌تری دارد. ابتدا پروگرامر را به سیستم متصل و سپس نرم افزار را باز کنید. در شکل پ-۱ شمای کلی این نرم‌افزار نشان داده شده است.



شکل پ-۱: شمای نرم‌افزار Progisp

پ-۲ آشنایی با قسمت های مختلف نرم افزار

پ-۲-۱ بخش Select Chip

از این کادر می توانید شماره آی سی میکروکنترلر را انتخاب نمایید. در این بخش با فشردن کلید RD ID یا شماره آی‌سی انتخاب شده با آی‌سی متصل شده مقایسه می شود. همچنین با فشردن نحوه اتصال پروگرامر به میکرو را نمایش داده می‌شود.

پ-۲-۲ بخش Program State

در این قسمت نوع اتصال پروگرامر را می‌توان مشخص کرد. که باید برروی USBASP باشد.

پ-۲-۳ بخش Programming

در این قسمت می‌توانید با انتخاب گزینه‌ها مشخص کنید که با زدن کلید auto کدام اعمال بر روی میکروکنترلر انجام شود. این گزینه ها به این صورت می باشد:

ـ Verify Signature: آی سی قرار گرفته بر روی میکروکنترلر با آی‌سی انتخاب شده در قسمت select chip مقایسه می‌شود.

ـ Chip Erase: حافظه فلش میکروکنترلر را پاک می‌کند.

ـ Prewritten Fuse: از این گزینه می‌توانید برای افزایش سرعت پروگرام نمودن میکروکنترلر استفاده نمایید. به این شکل که تغییر فیوز بیتی که در این آدرس قرار دارد، پروگرامر قبل از عمل پروگرام کردن فیوز بیت را تغییر می‌دهد تا سرعت پروگرام نمودن افزایش یابد؛ سپس آی‌سی را پروگرام می‌کند و در نهایت فیوز بیت تغییر داده شده را به مقدار ثبت شده در قسمت program fuse بر می‌گرداند.

ـ Blank Check: حافظه فلش میکرو را برای پروگرام کردن چک می‌کند.

ـ Program Flash: فایل hex انتخاب شده توسط کاربر را بر روی حافظه flash قرار می‌دهد.

ـ Program EEPROM: فایل EEPROM انتخاب شده توسط شما را بر روی حافظه EEPROM قرار می‌دهد.

ـ Data Reload: آخرین فایل hex انتخاب‌شده توسط کاربر را load نموده و دیگر نیاز به انتخاب فایل در هر بار پروگرام کردن نمی‌باشد.

ـ Verify FLASH: برنامه پروگرام شده بر روی میکروکنترلر با فایل hex اصلی مقایسه می‌کند.

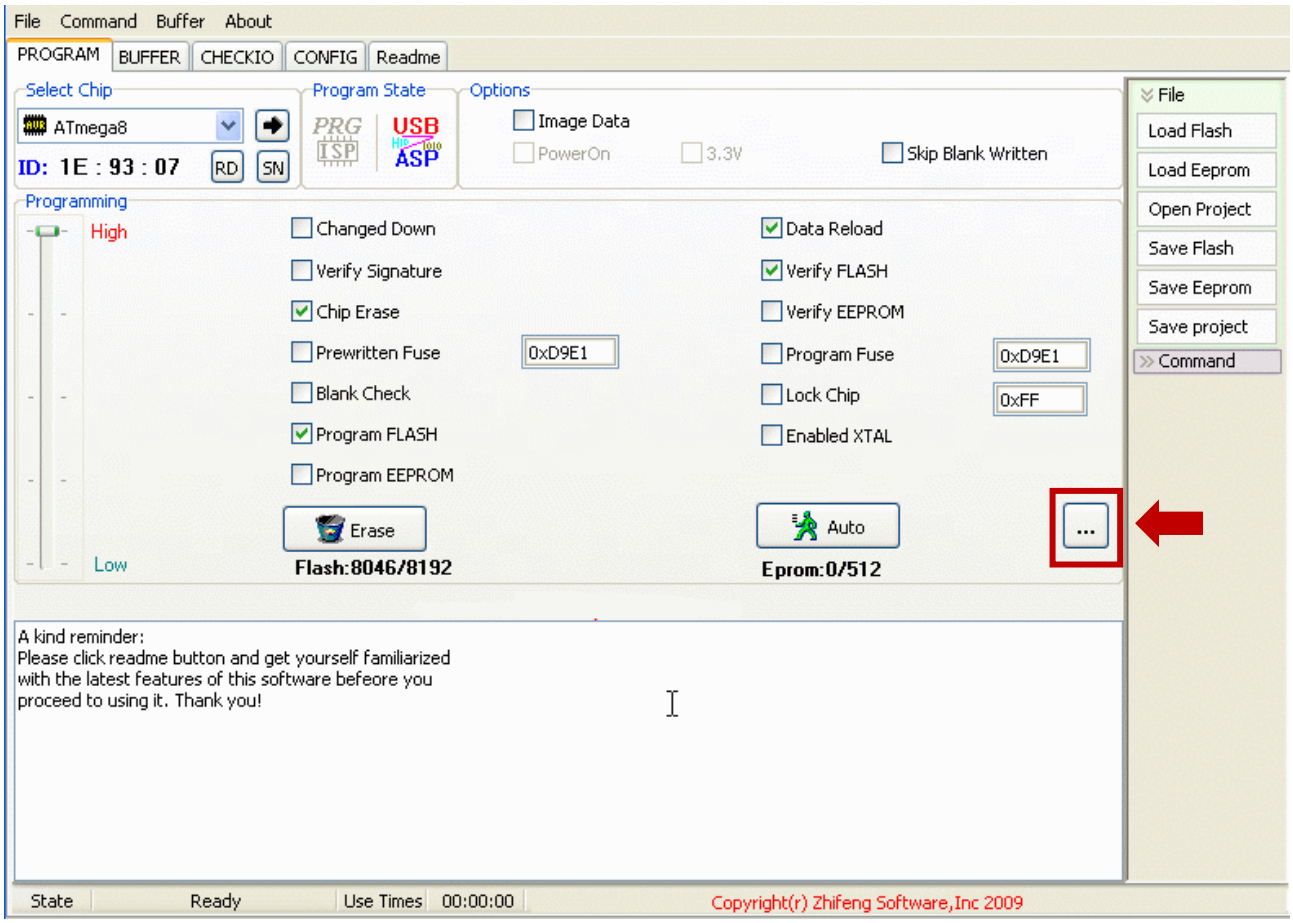
ـ Verify EEPROM: داده های EEPROM قرار گرفته بر روی حافظه EEPROM آی‌سی میکروکنترلر را با فایل اصلی چک می‌کند.

ـ Program Fuse: پروگرام کردن فیوز بیت های میکروکنترلر

ـ Lock Chip: پروگرام کردن فیوز بیت قفل میکروکنترلر

ـ Enabled XTAL: جهت فعال کردن کریستال خارجی

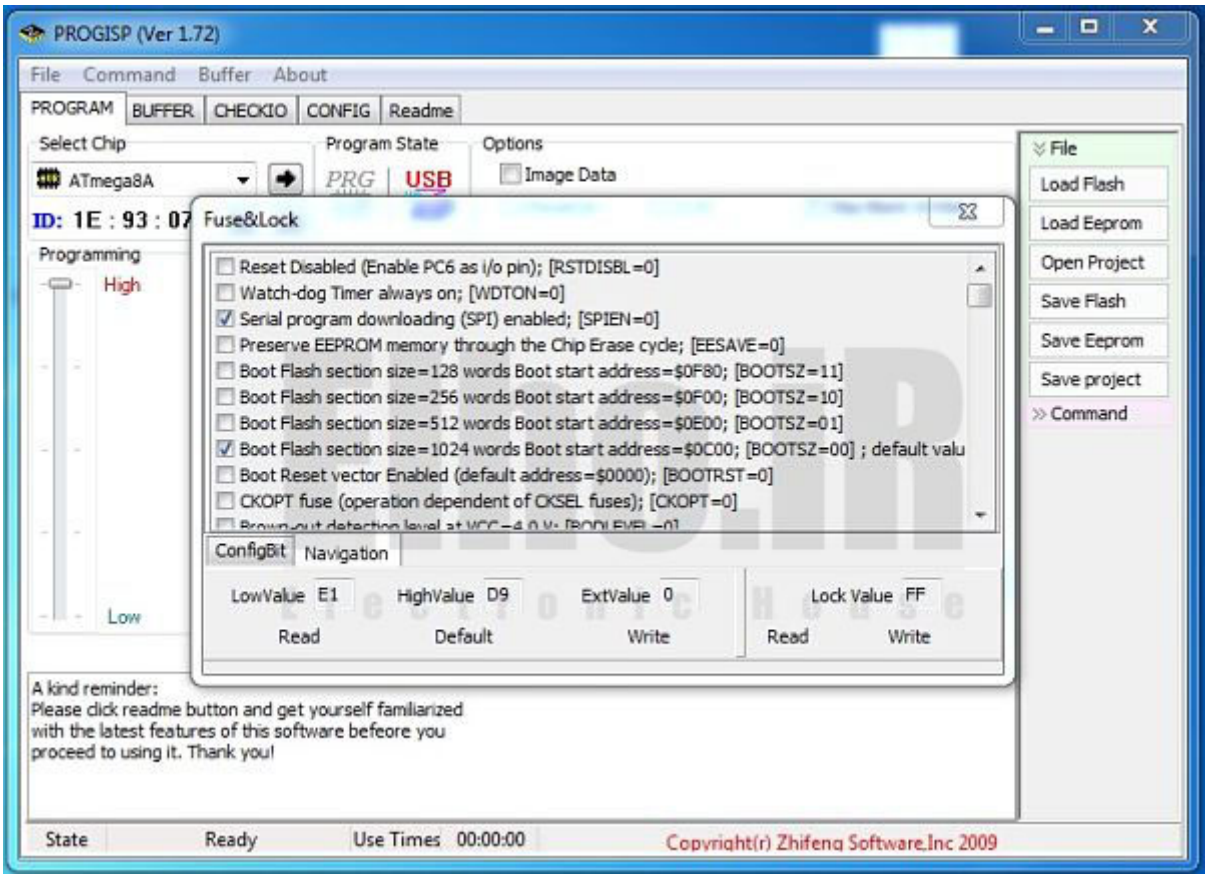
همچنین شما می توانید با انتخاب گزینه مشخص شده در شکل پ-۲ فیوز بیت های میکروکنترلر انتخاب شده را مشاهده کنید و یا با گزینه Read – Default – Write اعمال مورد نظر را انجام دهید (شکل پ-۳).



شکل پ-۲: انتخاب فیوز بیت‌های میکروکنترلر

شکل پ-۳: حالت Config Bit در پنجره Fuse&Lock

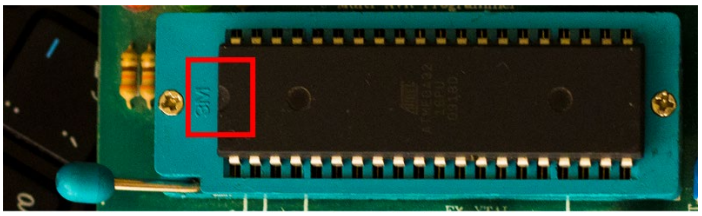
همچنین می‌توان در حالت Navigation، تنظیمات فیوز بیت هارا به راحتی تغییر داد، برای مثال کریستال داخلی را تغییر داده یا میکروکنترلر را بر روی کریستال خارجی تنظیم نمود (شکل پ-۴). برای تازه‌کارها پیشنهاد می شود در میکروکنترلرهایی که تنظیمات JTAG برای آن‌ها فعال است و از آن استفاده نمی‌کنید حتماً آن را غیرفعال کنید تا همه پایه های میکرو کنترلر قابل استفاده باشند.



شکل پ-۴: حالت Navigation در پنجره Fuse&Lock

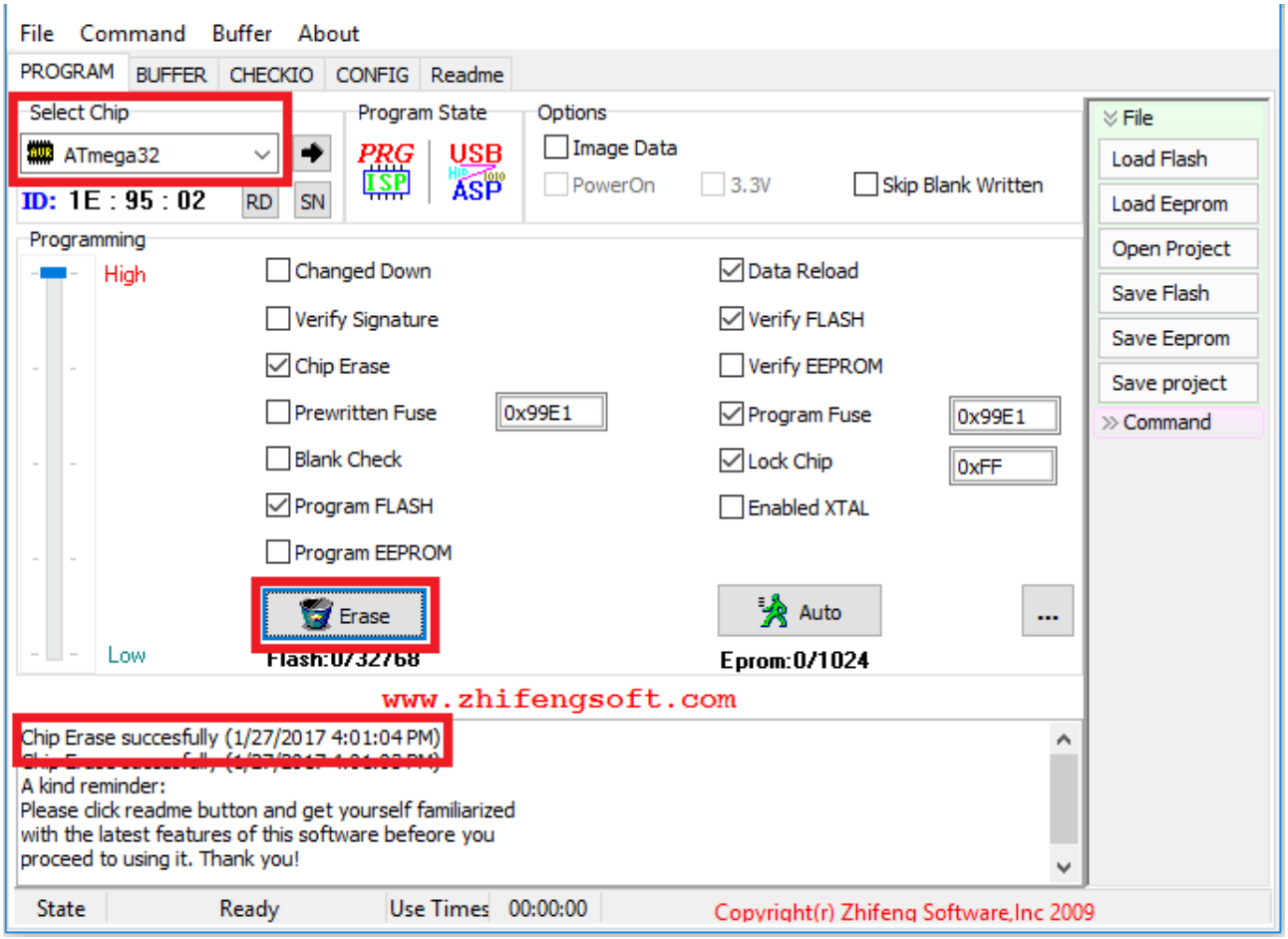
پ-۳ پروگرام کردن

برای پروگرام نمودن میکروکنترلر را در جهت صحیح (به طوری که نیم دایره بالای آن نزدیک به اهرم سوکت سبز رنگ قرار گیرد) بر روی پروگرمر قرار دهید. اهرم را پایین بیاورید تا میکروکنترلر در جای خود ثابت شود.



شکل پ-۵: نحوه قرار دادن صحیح میکروکنترلر بر روی پروگرامر

نرم‌افزار Progisp را بر روی کامپیوتر اجرا کنید و اگر پروگرامر به کامپیوتر وصل نیست، آن را متصل کنید. حال باید مدل میکروکنترلر را از منوی سمت چپ بالای نرم‌افزار انتخاب کنید. دقت کنید که مدل انتخابی‌تان دقیقاً با مدل ثبت شده بر روی میکروکنترلر برابر باشد. در صورتی که همه مراحل را به طور صحیح انجام داده باشید، با کلیک بر روی Erase ، باید با پیغام Chip Erased Successfully مواجه شوید.



شکل پ-۶: پاک کردن تراشه میکروکنترلر

برای پاک کردن تراشه میکروکنترلر و نیز پروگرام کردن ابتدا با استفاده از گزینه Load Flash فایل hex را وارد نمایید. حال سه گزینه Chip Erase و Program Flash و Verify Flash را تیک بزنید.

پس از انجام موارد بالا با زدنAuto ، این مراحل به ترتیب اجرا می‌شوند: ابتدا حافظه میکروکنترلر پاک شده. سپس برنامه روی حافظه ریخته می‌شود و در نهایت مقایسه‌ای بین برنامه داخل میکرو با فایل hex انجام می‌گردد. مورد آخر برای اطمینان از سالم بودن عمل پروگرام است. اگر همه موارد به‌درستی انجام شده باشد، یک پیغام موفقیت آمیز (Successful…) در پنل اعلانات مشاهده خواهد شد.

منابع

[۱] کاهه، علی (۱۳۸۵). میکروکنترلرهای AVR. (چاپ اول ـ ویراست ۲). تهران: مؤسسه علمی فرهنگی نص.

[۲] پرتوی‌فر، محمدمهدی؛ مظاهریان فرزاد و بینالو، یوسف (۱۳۹۰). مرجع کامل میکروکنترلرهای AVR. (چاپ هشتم ـ ویراست ۲). تهران: مؤسسه علمی فرهنگی نص.

[۳] الوندی، جابر (۱۳۹۱). میکروکنترلرهای AVR با پروژه‌های ۱۰۰٪ عملی. (چاپ پنجم ـ ویراست ۲). تهران: مؤسسه علمی فرهنگی نص.

[۴] مزیدی، محمدعلی؛ نعیمی، سپهر و نعیمی، سرمد. (۲۰۱۱). میکروکنترلرهای AVR برنامه‌نویسی اسمبلی و C. (چاپ اول). ترجمه: آناهیتا نعیمی (۱۳۸۸). تهران: مؤسسه علمی فرهنگی نص.

[۵] سیدرضی، حسن (۱۳۹۱). میکروکنترلرهای AVR. (چاپ چهارم). تهران: ناقوس.

[۶] برگه اطلاعاتی (Datasheet) میکروکنترلر ATmega32