دستور کار آزمایشگاه

میکروپرسسور

1- مقدمه

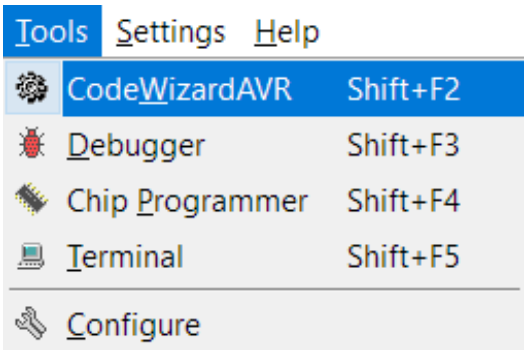
میکروکنترلرهای AVR در سال 1994 توسط شرکت ATMEL طراحی و روانه‌ی بازار شدند. این میکروکنترلرها 8 بیتی هستند به این مفهوم که در هر سیکل کاری توانایی پردازش یک داده‌ی 8 بیتی را دارند. قیمت بسیار کم، مصرف انرژی پایین و امکانات متنوع، AVR را به یکی از پرکابردترین و محبوب‌ترین میکروکنترلرها تبدیل کرده است. امروزه پلتفرم Arduino در بین علاقه‌مندان به میکروکنترلرها بسیار مورد توجه قرار گرفته است و جالب است بدانید که در بسیاری از بردهای Arduino از میکروکنترلرهای AVR استفاده می‌شود. برخی از خصوصیات کلی این میکروکنترلرها در ادامه بیان شده است:

- ✓ استفاده از معماری RISC که میکروکنترلر را قادر به اجرای اغلب دستورات در یک سیکل می‌کند.
- ✓ کارایی بالا با توجه به میزان توان مصرفی کم
- ✓ دارای حدود 133 دستورالعمل اجرایی
- ✓ دارای 32 ثبات 8 بیتی
- ✓ دارای دستوراتی که اغلب طول ثابتی برابر 16 بیت دارند.
- ✓ سرعتی حداکثر برابر با 16MIPS
- ✓ حافظه FLASH داخلی برای ذخیره کد برنامه
- ✓ حافظه EEPROM داخلی جهت نگه‌داری دائمی اطلاعات
- ✓ حافظه‌ی SRAM داخلی جهت نگه‌داری موقت اطلاعات
- ✓ مبدل آنالوگ به دیجیتال داخلی
- ✓ شمارنده‌های 8 بیتی و 16 بیتی
- ✓ قابلیت In-System Programming
- ✓ Real Time Clock داخلی
- ✓ قابلیت ارتباط سریال
- ✓ دارای Oscillator داخلی
- ✓ و بسیاری موارد دیگر

میکروکنترلرهای AVR دارای سه خانواده‌ی اصلی MEGA، XMega، Tiny و خانواده‌ای از AVRهای متناسب با کاربردهای خاص از جمله LCD، USB، PWM، CAN و غیره هستند. علاوه بر این خانواده‌ی FPSLIC که AVR مجهز به FPGA می‌باشد نیز دسته‌ای از این میکروکنترلرها را تشکیل می‌دهد.

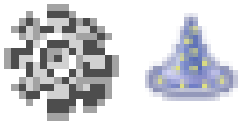
2- استفاده از Code Wizard

3-1 نرم‌افزار کدویژن دارای قابلیتی بسیار کاربردی به نام Code Wizard است که بخش بسیاری از کدهای مورد نیاز برای اعمال تنظیمات اصلی پروژه و مقادیر مربوط به بسیاری از رجیسترها را به صورت خودکار تولید می‌کند. در عمل دو راه برای دسترسی به CodeWizard وجود دارد. در راه اول، در ابتدای ساخت پروژه‌ی جدید هنگامی که نرم افزار در مورد استفاده از CodeWizard سوال می‌پرسد(شکل 2)، باید گزینه‌ی Yes را انتخاب نماییم. روش دوم دسترسی به Wizard استفاده از منوی Tools و کلیک بر روی CodeWizardAVR است که در شکل 13 نشان داده شده است.



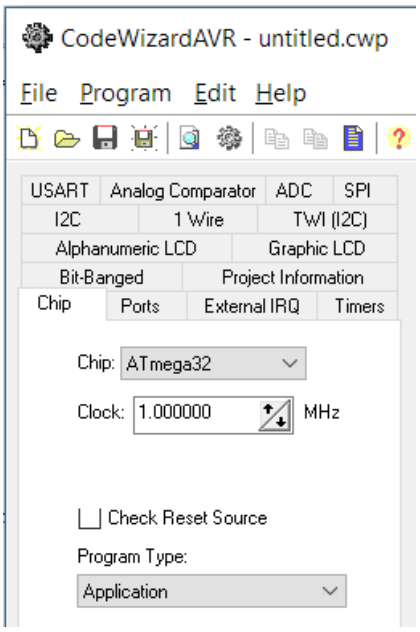
شکل 13- باز کردن CodeWizard از منوی Tools

همچنین می‌توان بر روی icon مربوط به CodeWizard کلیک کرد که با توجه به نسخه‌ی CodeVision مورد استفاده، به صورت یکی از موارد نشان داده شده در شکل 14 خواهد بود.



شکل 14- شکل icon مربوط به CodeWizard
در نسخه‌های مختلف CodeVision

در هر صورت پس از وارد شدن به CodeWizard با صفحه‌ای مشابه شکل 15 مواجه می‌شویم. البته این صفحه در ورژن‌های مختلف کمی متفاوت است اما موارد کلی در آن یکسان است.

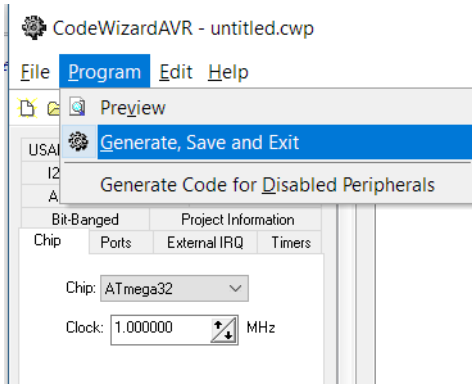


شکل 15- صفحه‌ی CodeWizard

در این مرحله در اولین گام باید نوع میکروکنترلر و کلاک را از سربرگ Chip انتخاب نماییم. سپس با توجه به مواردی که در پروژه به آن نیاز داریم، بر روی سایر سربرگ‌ها کلیک کرده و در بخش مربوطه تنظیمات مورد نظر خود را انجام می‌دهیم. سپس دو راه داریم: ساخت کد تا این مرحله یا مشاهده‌ی کد

1-1-3 ساخت کد

در صورتی که بخواهیم کدی جدید تولید کنیم که به جز تنظیمات انجام شده در CodeWizard هیچ کد دیگری در آن نباشد، در این مرحله باید از منوی Program (در صفحه‌ی CodeWizard) گزینه‌ی دوم با عنوان Generate, Save and Exit را انتخاب نماییم.



شکل 16- ساخت کد جدید

در نهایت پس از انتخاب نام مورد نظر برای پروژه و فایل‌های مرتبط با آن و همین‌طور مشخص نمودن محل ذخیره سازی، یک پروژه‌ی جدید ایجاد خواهد شد که می‌توانیم سایر کدهای مورد نیاز را در آن بنویسیم.

3-1-1 پیش نمایش کد

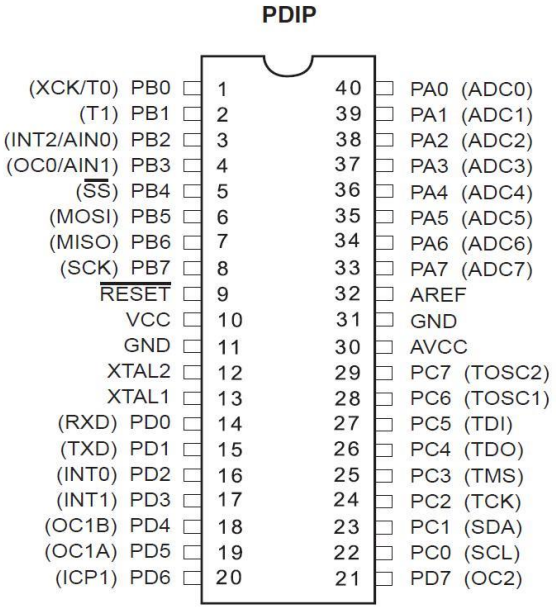
در صورتی که بخواهیم در کد یک پروژه که در حال کار روی آن هستیم تغییراتی اعمال کنیم یا مقدار برخی از رجیسترهای میکروکنترلر را تغییر دهیم، می‌توانیم از قابلیت Preview (یا قابلیت Generate در نسخه‌های جدیدتر) استفاده نمود. با انتخاب این گزینه کد مورد نظر تولید می‌شود ولی پروژه‌ای که در حال کار روی آن هستیم یا Source مربوط به آن تغییری نمی‌کند. در این حالت می‌توانیم مقادیر مورد نیاز خود را از کد تولید شده کپی نموده و در Source پروژه‌ی خود وارد نماییم.

▪ آزمایش شماره 1 : پورت‌های ورودی/خروجی (I/O Ports)

- هدف آزمایش: آشنایی با پورت‌های ورودی و خروجی و رجیسترهای مربوط به آن‌ها
- نتیجه عملی: نمایش اطلاعات دیجیتال وارد شده توسط کلیدها بر روی LED

✓ **بخش اول:** توضیحات زیر را بخوانید و به سوالات پاسخ دهید.

در میکروکنترل‌های AVR هر پورت ورودی/خروجی توسط سه رجیستر، PORTX، PINX و DDRX کنترل می‌شوند. توجه کنید که X عنوان پورت مورد نظر را نشان می‌دهد مثلاً PORTA یا PINB . پورت‌ها در واقع گروهی از پایه‌های فیزیکی میکروکنترلر هستند که در دسترس ما قرار دارند. کاربرد اصلی این پورت‌ها ورودی/خروجی است اما معمولاً هر پایه‌ی فیزیکی از پورت کاربردهای دیگری هم دارد که در آزمایش‌های بعدی با برخی از آن‌ها آشنا خواهیم شد. نحوه‌ی قرارگیری پورت‌ها در یک میکروکنترلر یا IC را می‌توان با استفاده از Pinout در دیتاشیت مشاهده کرد. به عنوان مثال محل Pinout میکروکنترل ATMEGA32 در شکل 1-1 آورده شده است.



شکل 1-1 Pinout میکروکنترل ATMEGA32

همانطور که در شکل فوق مشاهده می‌کنید، هرکدام از پورت‌ها در این میکروکنترلر دارای 8 پایه می‌باشد. به عبارت دیگر هر پورت 8 بیتی است. نکته‌ای که باید به آن توجه داشت این است که بیت‌های رجیسترهای معرفی شده در این بخش متناظر با پایه‌های پورت می‌باشند. یعنی در عمل می‌توان هریک از پایه‌های پورت را به صورت جداگانه به عنوان ورودی یا خروجی تنظیم نمود. برای تنظیم ورودی یا خروجی

بودن یک پورت(یا پایه‌ای از پورت) از رجیستر DDR(Data Direction Register) استفاده می‌شود. به عنوان مثال اگر DDRB.3=1 باشد، پایه‌ی PB3 به صورت خروجی خواهد بود یا اگر DDRA.2=0 باشد، پایه‌ی PA2 به صورت ورودی خواهد شد. وقتی بخواهیم ورودی یا خروجی بودن یک پورت را با یک خط برنامه مشخص کنیم، کافی‌است مقدار رجیستر DDR مورد نظر را به صورت عدد باینری یا هگزادسیمال محاسبه کنیم و در کد وارد نماییم. معمولاً برای سادگی از اعداد هگزادسیمال استفاده می‌شود. مثلاً می‌نویسیم DDRA=0xFF که در آن 0x مشخص کننده‌ی فرمت هگزادسیمال است و این کد به معنی آن است که تمامی پایه‌های پورت A خروجی هستند.

مد خروجی: وقتی یک پورت یا پایه به صورت خروجی تعریف شده است، می‌توان اطلاعات را از درون میکروکنترلر بر روی پایه‌های آن قرار داد. یعنی در این حالت پایه‌ها به عنوان خروجی میکروکنترلر عمل می‌کنند. در این وضعیت با مقداردهی به رجیستر PORT می‌توان پایه‌های خروجی را 0 یا 1 نمود. مثلاً دستور DDRC=0xFF تمامی پایه‌های پورت C را به صورت خروجی تعریف می‌کند. در این حالت با استفاده از PORTC=0x03 می‌توان پایه‌های PC0 و PC1 را 1 و بقیه را صفر نمود. (0x03=00000011)

مد ورودی: اگر یک پایه یا پورت به صورت ورودی تعریف شود، با مقدار دهی به رجیستر PORT می‌توان تعیین نمود که آیا آن پایه یا پورت به صورت Pull-up داخلی باشد یا این که در وضعیت امپدانس بالا (High Impedance) قرار بگیرد. به عنوان مثال دستور DDRA=0xF0 چهار بیت پایین پورت A را به عنوان ورودی و چهاربیت بالای آن را به عنوان خروجی تعریف می‌کند. حال اگر بنویسیم PORTA=0xFF مقدار قرار گرفته روی چهار بیت خروجی(یعنی PA4 تا PA7) برابر 1 خواهد بود. علاوه بر این پایه‌های ورودی نیز (PA0 تا PA3) به صورت Pull-Up هستند. در حالت ورودی می‌توان با استفاده از دستور(رجیستر) PIN مقدار قرار گرفته روی پایه‌های ورودی را خواند. معمولاً این مقدار را در یک متغیر ذخیره می‌کنیم.

سوال‌های این بخش: (پاسخ این سوالات را به TA ارائه دهید.)

1. کدی بنویسید که پایه‌های پورت C را یکی در میان ورودی و خروجی کند.(PC0 ورودی باشد.)
2. کدی بنویسید که روی پایه‌های خروجی سوال 1 ، مقدار 1 را قرار دهد و پایه‌های ورودی به صورت Pull-up باشد.
3. کدی بنویسید که پایه‌های پورت D را به صورت ورودی(High Impedance) تنظیم کند. سپس کدی به زبان C بنویسید که مقدار این پایه‌ها را خوانده و در متغیری از نوع char با نام var1 ذخیره کند. (راهنمایی: در زبان C متغیر را می توان به صورت ;char var1 تعریف نمود.)

- ✓ **بخش دوم:** با توجه به توضیحات زیر، اتصالات سخت‌افزاری مورد نیاز را با استفاده از سیم بر روی برد آموزشی برقرار کنید. سپس کد داده شده در این بخش را تکمیل نموده و برنامه کامپایل را کنید. در نهایت برد را پروگرام کرده و نتیجه را به استاد یا TA ارائه دهید.

سخت افزار: پورت B را به LED های قرار گرفته روی برد متصل نمایید. 8 عدد از کلیدهایی که در هنگام فعال بودن مقدار یک دیجیتال را به میکرومنتقل می‌کنند، را هم به پورت D متصل نمایید.(می‌توانند از DIP Switchها یا سایر انواع کلیدهای روی برد استفاده نمایید.)

نرم‌افزار: کد زیر را به صورتی تکمیل کنید که پورت B خروجی با مقدار اولیه صفر و پورت D ورودی با پایه‌های Pull-up شده‌ی داخلی باشد. در این کد هدف ما خواندن مقادیر دیجیتال روی پورت D و نمایش آن بر روی LED ها با کمک پورت خروجی B است.

```
1-   #include <mega32.h>
2-
3-    void main(void)
4-    {
5-      //Declare your variables here
6-      unsigned char var1;
7-
8-     //PORTB initialization
9-      //All Output
10-     DDRB=
11-     PORTB=
12-
13-     //PORTD initialization
14-     //All Input, internally pulled-up
15-     DDRD=
16-     PORTD=
17-
18-     while(1){
19-       //Read the value of PORTD, Store in var1
20-        var1=
21-        PORTB=var1;
22-            };
23-   }
```

توجه:

در خط 1 این نرم‌افزار مشخص شده است که از میکروکنترلر ATMEGA32 استفاده می‌شود. در صورتی که میکروکنترلر ATMEGA16 روی برد شما نصب شده است، این خط را تغییر دهید.

نکات برنامه نویسی:

- دستور #include برای اضافه کردن یک Header به برنامه به کار می‌رود. Header یا سربرگ مجموعه‌ای از دستورات را شامل می‌شود و با اضافه کردن آن به برنامه می‌توان از توابع موجود در آن سربرگ استفاده نمود. در صورتی که دستور یا تابعی را در کد بدون اضافه کردن سربرگ آن استفاده کنیم، برنامه در زمان کامپایل دچار خطا خواهد شد
- در صورت نیاز به استفاده از چند سربرگ مختلف کافی است مشابه خط 1 در **ابتدای کد** هر سربرگ را اضافه کرد.
- در زبان C برخلاف MATLAB باید قبل از استفاده از هر متغیر **نوع** و **نام** آن را تعریف کرد. (به خط 6 در کد توجه کنید). انتخاب نوع متغیر بر بازه‌ی مقادیری که می‌توان در آن ذخیره نمود تاثیر گذار است. انتخاب متغیر با بزرگترین بازه انتخابی مطمئن به نظر می‌رسد اما باید توجه داشت که در این صورت حافظه بدون ضرورت اشغال خواهد شد و این مساله به ویژه در سیستم‌هایی با حافظه محدود (مانند AVR) مشکل ساز خواهد شد. برای انتخاب مناسب نوع متغیر می‌توانید از جدول زیر کمک بگیرید.

Type	Size (Bits)	Range
bit	1	0 , 1
char	8	-128 to 127
unsigned char	8	0 to 255
signed char	8	-128 to 127
int	16	-32768 to 32767
short int	16	-32768 to 32767
unsigned int	16	0 to 65535
signed int	16	-32768 to 32767
long int	32	-2147483648 to 2147483647
unsigned long int	32	0 to 4294967295
signed long int	32	-2147483648 to 2147483647
float	32	±1.175e-38 to ±3.402e38
double	32	±1.175e-38 to ±3.402e38

همچنین استفاده از هر اسمی برای متغیر مجاز نیست و عبارت‌های زیر keyword های Reserve شده هستند:

Do	default	continue	const	char	case	bit	break
for	float	flash	extern	enum	else	eeprom	double
register	long	interrupt	int	inline	if	goto	funcused
struct	static	sfrw	sfrb	sizeof	signed	short	return
	while	volatile	void	unsigned	union	typedef	switch
Register names like PORTA, etc.							

- حتماً indentation را رعایت کنید تا ساختار کد مشخص باشد و debugging امکان پذیر شود.

اشتباهات پرتکرار(قبل از کامپایل کردن کد این موارد را بررسی کنید.):

- در انتهای هر خط از کد باید یک semi-colon (;) قرار داشته باشد.
- همه‌ی حروف نام رجیسترها مانند PORTB، PIND و DDRX باید به صورت Capital Letter تایپ شوند.
- تعداد آکولادهای باز({) و بسته (}) یکسان نیست. دقت کنید که کد درون هر تابع، حلقه یا if در زبان C باید در بین یک آکولاد باز و بسته قرار بگیرد. با رعایت indentation به راحتی می‌توانید این مساله را بررسی کنید.
- نوع AVR و مقدار کلاک را از منوی Project>Configure>C Compiler انتخاب کنید.
- فایل Source کد C را از منوی Project>Configure>Files به پروژه Add کنید.

پس از بررسی موارد فوق، کد را Build کنید و در صورتی که هیچ خطایی وجود نداشت، میکروکنترلر را پروگرم کنید. دقت کنید که برای پروگرم کردن باید از فایل hex. استفاده کنید. این فایل در محل ذخیره کردن پروژه (این آدرس در بالای CodeVision قابل مشاهده است) پس از ساختن(Build) کد ایجاد خواهد شد. اگر این فایل وجود نداشت، یا آدرس اشتباه انتخاب شده یا در هنگام کامپایل کد خطایی وجود داشته است. در صورت که در زمان کامپایل خطایی وجود داشته باشد در قسمت پایین CodeVision نمایش داده می‌شود. با مشاهده متن خطا و بررسی کد خطا را اصلاح کنید و در صورت ابهام از TA یا استاد کمک بخواهید.

- ✓ **بخش چهارم(Extension):** کد برنامه را با توجه توضیحات زیر، تغییر دهید و برد را پروگرام کرده و نتایج را به استاد یا TA ارائه دهید.

کد قبل را به صورتی تغییر دهید که ابتدا LED اول، سپس اول و دوم، بعد از آن اول تا سوم و در نهایت اول تا هشتم روشن شوند و این روند پس از روشن شدن تمام LEDها دوباره تکرار شود.

راهنمایی1: در این قسمت نیازی به پورت ورودی و خواندن از ورودی نیست.

راهنمایی2: برای اینکه روند روشن و خاموش شدن LED ها برای چشم قابل مشاهده باشد، باید در هر مرحله یک delay یا تاخیر وجود داشته باشد. برای این منظور می‌توانید از تابع delay_ms() استفاده کنید که ورودی آن میزان تاخیر با واحد میلی ثانیه است. این تابع در سربرگ delay.h قرار دارد.

در پایان این آزمایش باید موارد زیر را آموخته باشید:

- ✓ نحوه ورودی/خروجی نمودن یک پورت یا پایه
- ✓ رجیسترهای مرتبط با ورودی و خروجی و کاربرد هریک از آن‌ها
- ✓ نحوه قرار دادن مقدار روی پورت خروجی یا خواندن از پورت ورودی
- ✓ نحوه ساخت پروژه در CodeVision و پروگرم کردن میکروکنترلر
- ✓ آشنایی با ساختار کلی کد نویسی به زبان C، تعریف متغیرها و انتخاب نوع متغیر
- ✓ آشنایی با کاربرد تابع delay_ms()
- ✓ اضافه کردن یک سربرگ به برنامه و استفاده از توابع مرتبط با آن

▪ آزمایش شماره 2 : Alphanumeric LCD

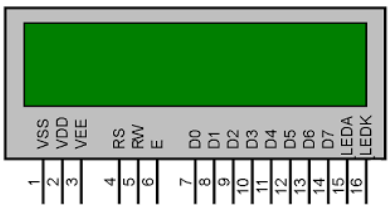
- هدف آزمایش: آشنایی با صفحه نمایش متنی و اتصال آن به میکروکنترلر
- نتیجه عملی: نمایش اطلاعات دلخواه روی صفحه نمایش متنی و استفاده از دستورات طراحی شده برای LCD

✓ **بخش اول:** توضیحات زیر را بخوانید و به سوالات پاسخ دهید.

انواع مختلفی از صفحه‌های نمایش را می‌توان برای نشان دادن اطلاعات متنی یا گرافیکی به کار برد. به عنوان مثال صفحه نمایش‌های OLED، Graphical LCD و Alphanumeric LCD از جمله صفحه نمایش-های پرکابرد هستند. در این میان صفحه نمایش‌های Alphanumeric به علت سادگی استفاده و قیمت مناسب بسیار مورد استفاده قرار می‌گیرند. این نوع از صفحه نمایش‌ها در ابعاد مختلفی تولید می‌شوند و با تعداد سطر و ستون‌ها دسته‌بندی می‌گردند. در هر ستون از هر سطر می‌توان یک کاراکتر(که می‌تواند حرف، عدد یا یک سمبول باشد) را نشان داد. به عنوان مثال در یک LCD با ابعاد 16*2 دو سطر وجود دارد و در هر سطر می‌توان 16 کاراکتر را نمایش داد. شکل ظاهری یک صفحه نمایش Alphanumeric در شکل زیر آورده شده است.

شکل 2-1 شکل ظاهری Alphanumeric LCD

LCD معرفی شده در این بخش دارای 16 پایه است که Pinout آن در شکل 2 نمایش داده شده است.



شکل 2-2 شکل ظاهری Alphanumeric LCD

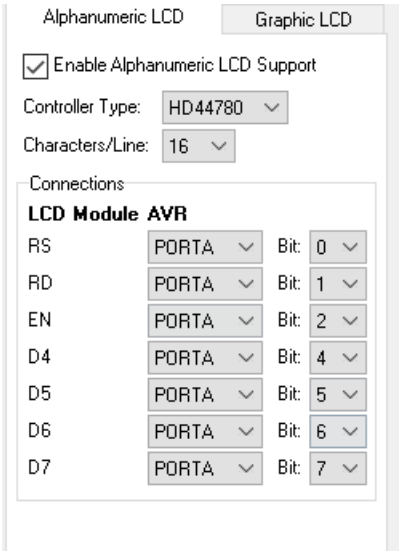
پایه‌های 15 و 16 برای روشن یا خاموش کردن نور پس زمینه، پایه‌ی 3 برای کنترل کنتراست(کم‌رنگ یا پرنگ بودن) کاراکترهای نمایش داده شده به کار می‌روند. معمولاً به پایه‌ی سوم به پایه‌ی وسط یک مقاومت

متغیر که بین گراند و 5 ولت قرار دارد،متصل می‌گردد. سایر پایه‌ها نیز تغذیه‌ی مدار نمایشگر، فرآیندهای کنترلی و انتقال دیتا را بر عهده دارند. جدول زیر خلاصه‌ی عملکرد پایه‌ها را نمایش می‌دهد.

جدول 1-2 پایه‌های Alphanumeric LCD

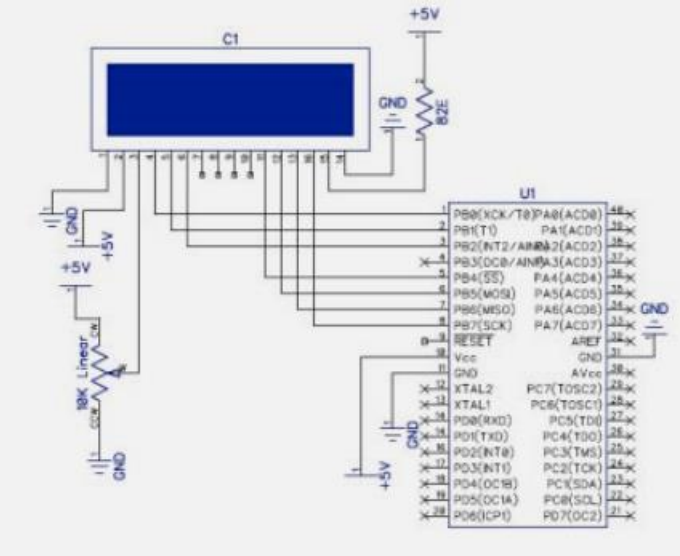
PIN NO.	Symbol	Function
1	VSS	GND
2	VDD	+5V
3	VEE	Contrast Adjustment
4	RS	Register Select Signal
5	R/W	Read/Write Signal
6	E	Enable Signal
7	D0	Data Bus Line Bit 0
8	D1	Data Bus Line Bit 1
9	D2	Data Bus Line Bit 2
10	D3	Data Bus Line Bit 3
11	D4	Data Bus Line Bit 4
12	D5	Data Bus Line Bit 5
13	D6	Data Bus Line Bit 6
14	D7	Data Bus Line Bit 7
15	A	+4.2v For Backlight
16	K	GND For Backlight

نحوه اتصال LCD به میکروکنترلر توسط کامپایلر تعیین می‌شود و می‌توان با کمک Code Wizard نحوه اتصال برای هر پورت را مشخص نمود. معمولاً پایه 5 و 6 و 7 را به سه بیت کم ارزش یک پورت (صفر تا 2) وصل می‌کنیم. بیت چهارم از پورت (پایه شماره 3) را آزاد گذاشته و بیت‌های بعدی را به ترتیب به D4 تا D7 وصل می‌کنیم. این توضیحات در شکل بعدی نمایش داده شده است.



شکل 2-3 محل اتصال پایه‌های LCD

با توجه به توضیحات ارائه شده مدار سخت‌افزاری اتصال LCD به میکروکنترلر به صورت زیر خواهد شد:



شکل 2-4 اتصال سخت‌افزاری LCD

البته دقت کنید که در برد مورد استفاده در آزمایشگاه، پایه‌های تغذیه، نور پس‌زمینه و کنترل کنتراست از طریق PCB به محل‌های متناسب وصل هستند و نیازی به اتصال آن‌ها نیست.

پس از برقراری اتصالات سخت‌افزاری باید کد مورد نیاز برای راه‌اندازی LCD را نوشت. به این منظور باید سربرگ lcd.h را به کد خود اضافه کنیم. همچنین باید با یک خط کد Assembly پورتی که به LCD اختصاص یافته است را مشخص نمود. به صورت کلی باید دقت کرد که دو نوع داده را می‌توان روی LCD نمایش داد: کاراکتر یا مجموعه‌ای از کاراکترها(String). اگر متن مورد نظر جهت نمایش مستقیم در کد نوشته شود به آن flash string می‌گوییم و در صورتی که این متن در یک آرایه کاراکتری با نامی مشخص قرار داشته باشد، آن را String می‌نامیم که دستور نمایش این دو نوع مختلف باهم متفاوت است. جدول زیر برخی از توابع سربرگ lcd.h را نمایش می‌دهد.

جدول 2-2 برخی از دستورات lcd.h

Function Name	Function	Input	Example
lcd_init()	LCD initialization	Number of characters/line	lcd_init(16);
lcd_clear()	Clears LCD	void	lcd_clear();
lcd_putsf()	Display Flash String	String in double quotation	lcd_putsf(“AVR”)
lcd_puts()	Display String	Name of char array	lcd_puts(lcdbuffer);
lcd_gotoxy(,)	Set Starting Coordinates to display	First input: Column Second input: Row	lcd_gotoxy(4,1);

البته برای اینکه متن نمایش داده شده در LCD برای چشم انسان قابل مشاهده باشد باید همواره پس از نمایش متن از تابع delay استفاده نماییم.

پیش از این گفتیم که برای مشخص کردن محل اتصال LCD در کامپایلر از یک دستور اسمبلی استفاده می‌کنیم. این دستور به صورت زیر نوشته می‌شود(برای اتصال به پورت A):

```
.equ __lcd_port=0x1B; PORTA
```

در این دستور مقدار 0x1B آدرس رجیستر PORTA در میکروکنترلر AVR است که از دیتاشیت استخراج شده است. برای پیدا کردن آدرس سایر رجیسترها می‌توان از جدول Register Summary در دیتاشیت استفاده نمود. (این جدول در انتهای دستور کار آورده شده است)

سوال‌های این بخش: (پاسخ این سوالات را به TA ارائه دهید.)

1. کد اسمبلی داده شده را برای پورت B تغییر دهید.
2. کدی بنویسید که پس از پاک کردن متن روی LCD، متن "Hello World" را در سطر دوم نمایش دهد به گونه‌ای که اولین کاراکتر در ستون سوم از این سطر قرار بگیرد.
3. به نظر شما اگر یک عدد (بدون quotation) یا متغیری به فرمت int را در ورودی lcd_puts() قرار دهیم، چه اتفاقی می‌افتد؟ پس از انجام بخش بعدی عدد 65 را در یک متغیر int ذخیره کرده و آن را به LCD ارسال کنید و نتیجه را بررسی کنید.
4. راهکاری برای کنترل نور پس‌زمینه(خاموش یا روشن بودن) توسط میکروکنترلر ارائه دهید.

- ✓ **بخش دوم:** با توجه به توضیحات زیر، اتصالات سخت‌افزاری مورد نیاز را با استفاده از سیم بر روی برد آموزشی برقرار کنید. سپس کد داده شده در این بخش را تکمیل نموده و برنامه کامپایل را کنید. در نهایت برد را پروگرم کرده و نتیجه را به استاد یا TA ارائه دهید.

سخت افزار: پورت B را به LCD کاراکتری 2 در 16 قرار گرفته روی برد متصل نمایید.

نرم‌افزار: کد این آزمایش را به صورتی تکمیل کنید که پورت B را به LCD اختصاص دهد. همچنین در سطر اول عبارت "AVR " و در سطر دوم عبارت "Microcontrollers" نمایش داده شود.

```
1-      #include <mega32.h>
2-      #include <lcd.h>
3-      #include <delay.h>
4-
5-      #asm
6-        .equ __lcd_port=   ; PORTB
7-      #endasm
8-
9-       void main(void)
10-      {
11-        //Declare your variables here
12-
13-
14-        lcd_init(16);
15-
16-        while(1){
17-          lcd_clear();
18-          delay_ms(2000);
19-          lcd_gotoxy(,);
20-          lcd_putsf("   AVR");
21-          delay_ms(2000);
22-          lcd_gotoxy(,);
23-          lcd_putsf("Microcontrollers");
24-          delay_ms(6000);
25-                };
26-      }
```

توجه:

در خط 1 این نرم‌افزار مشخص شده است که از میکروکنترلر ATMEGA32 استفاده می‌شود. در صورتی که میکروکنترلر ATMEGA16 روی برد شما نصب شده است، این خط را تغییر دهید.

نکات برنامه نویسی:

- دستورهای #asm و #endasm برای نوشتن کد اسمبلی درون یک کد C به کار می‌روند.
- حتماً indentation را رعایت کنید تا ساختار کد مشخص باشد و debugging امکان پذیر شود.

پس از بررسی موارد فوق، کد را کامپ برای انتخاب مناسب نوع متغیر می‌توانید از جدول زیر کمک بگیرید. همچنین هر اسمی را نمی‌توان برای متغیر انتخاب نمود.

پس از بررسی موارد فوق، کد را Build کنید و در صورتی که هیچ خطایی وجود نداشت، میکروکنترلر را پروگرم کنید. دقت کنید که برای پروگرم کردن باید از فایل hex. استفاده کنید. این فایل در محل ذخیره کردن پروژه (این آدرس در بالای CodeVision قابل مشاهده است) پس از ساختن(Build) کد ایجاد خواهد شد. اگر این فایل وجود نداشت، یا آدرس اشتباه انتخاب شده یا در هنگام کامپایل کد خطایی وجود داشته است. در صورت که در زمان کامپایل خطایی وجود داشته باشد در قسمت پایین CodeVision نمایش داده می‌شود. با مشاهده متن خطا و بررسی کد خطا را اصلاح کنید و در صورت ابهام از TA یا استاد کمک بخواهید.

- ✓ **بخش چهارم(Extension):** کد برنامه را با توجه توضیحات زیر، تغییر دهید و برد را پروگرام کرده و نتایج را به استاد یا TA ارائه دهید.

1. کد قبل را به صورتی تغییر دهید که متن "AVR" در ابتدای سطر اول نمایش داده شود. سپس این متن به سمت راست حرکت کرده و پس از رسیدن به انتهای سطر دوباره به سمت چپ حرکت کند.

راهنمایی1: در این قسمت می‌توانید از حلقه‌ی for استفاده نمایید.

راهنمایی2: حلقه‌ی for در زبان C به صورت زیر نوشته می‌شود که در آن نوع متغیر i (یا اسم دلخواه دیگر) باید قبل از استفاده تعریف شود. start و end نیز می‌توانند متغیر یا عدد باشند. در قسمت step نیز نحوه‌ی تغییر متغیر حلقه را مشخص می‌کند. مثلاً i++ به معنی یک واحد افزایش در هر مرتبه اجرا و i-- به معنی یک واحد کاهش در هر مرتبه اجرا می‌باشد.

ساختار حلقه‌ی for:

```
for (i=start;i<=end;step){
        //Your code here
                }
```

2. فرض کنید می‌خواهیم در صورت یک بودن یک پایه‌ی ورودی عدد 65 و در صورت 0 بودن آن عدد 120 را بر روی LCD نمایش دهیم. همچنین فرض کنید که هرکدام از این اعداد در یک متغیر int ذخیره شده است. عدد 65 در متغیر pinon و عدد 120 در متغیر pinoff ذخیره شده است. همچنین پایه‌ی ورودی مورد نظر پین PD0 است.

راهنمایی1: در این قسمت می‌توانید از ساختار if استفاده کنید.

```
if (//check condition){
        //Your code here
                }
else if(//check condition) {
        //Your code here

                }
else {
        //Your code here
    };
```

راهنمایی2: برای نمایش متغیرهای غیر کاراکتری بر روی LCD می‌توانید از دستور sprintf استفاده کنید. این دستور در سربرگ stdio.h قرار دارد. برای استفاده از آن باید ابتدا یک آرایه کاراکتری تعریف کنید.

مثال : دستور زیر را نظر بگیرید.

```
char lcd_buffer[16];
int data;
```

```
...
sprintf (lcd_buffer,"Your data is %d",data);
```

حال فرض کنید در طول اجرای برنامه مقدار متغیر data برابر با 5 شود. در این صورت پس از اجرای sprintf یک string(همان lcd_buffer که آرایه‌ای 16 تایی از کاراکترهاست) داریم که مقدار آن به صورت مقابل است: "Your data is 5"

دقت کنید که عبارت %d مشخص می‌کند که در این محل یک متغیر int باید به کاراکتر تبدیل شود. در صورتی که متغیر data از نوع char بود ، باید از %c در این محل استفاده می‌کردیم.

در پایان این آزمایش باید موارد زیر را آموخته باشید:

- ✓ نحوه راه اندازی Alphanumeric LCD
- ✓ نمایش متن بر روی LCD
- ✓ استفاده از توابع مرتبط با LCD
- ✓ نحوه به کارگیری تابع sprint و روش استفاده از آن

▪ آزمایش شماره 3 : Timer/Counter and Interrupt

- هدف آزمایش: آشنایی با ثبات‌ها و وقفه‌های تایمرها و استفاده از وقفه‌های خارجی
- نتیجه عملی: ساخت یک پالس مربعی با فرکانس دلخواه و ساخت یک شمارنده با کلاک پالس دستی

✓ **بخش اول:** توضیحات زیر را بخوانید و به سوالات پاسخ دهید.

تایمر/کانترها ابزار بسیار مفیدی برای زمان‌سنجی در سیستم‌های میکروکنترلی هستند. تایمرها در عمل همان شمارنده‌هایی هستند که در درس مدار منطقی مورد بررسی قرار گرفته‌اند. در واقع زمانی که کلاک پالس یک شمارنده از منبعی خارج از میکروکنترلر(مثلاً یک سوییچ فشاری) تامین شود، این مدار را شمارنده و زمانی که کلاک پالس از داخل میکروکنترلر به شمارنده اعمال شود، مدار را تایمر می‌نامند. میکروکنترلرهای مختلف(و مدل‌های مختلف AVR) تعداد متفاوتی تایمر داخلی دارند. به عنوان مثال میکروکنترلرهای ATMEGA16 و ATMEGA32 هرکدام 3 تایمر دارند. Timer0 و Timer2 دارای 8 بیت و Timer0 دارای 16 بیت است. تایمرها دارای کاربردهای متنوعی هستند که برخی از آن‌ها عبارتند از:

- ✓ انجام عملیاتی به طور متناوب با دوره تناوب مشخص مانند تولید موج مربعی
- ✓ سنجش زمان، مثلاً برای ساخت ساعت یا سیستم‌های اتوماسیون
- ✓ شمارش تعداد پالس‌های یک سیگنال خارجی، مانند سیستم‌های اندازه‌گیری فرکانس
- ✓ تولید PWM

تایمرها در میکروکنترلرهای AVR دارای چهار مد کاری هستند:

1. Normal: در این وضعیت رجیستر مربوط به تایمر از عدد 0 شروع به کار کرده و با هر پالس کلاک یک واحد افزایش می‌یابد. در نهایت پس از رسیدن به سقف خود (255 در تایمر 8 بیتی و 65535 در تایمر 16 بیتی) مجدداً صفر خواهد شد. در این زمان، وقفه‌ی Timer Overflow رخ می‌دهد. در عمل زمانی که این وقفه رخ دهد، با توجه به تعداد افزایش‌ها و دوره تناوب پالس کلاک اعمالی به شمارنده، می‌توان زمان سپری شده را هم محاسبه نمود.

2. Compare Match: در این وضعیت تایمر از عدد 0 شروع به شمارش می‌کند و زمانی که به یک عدد مشخص برسد، وقفه‌ی Compare Match رخ خواهد داد. این عدد توسط برنامه‌نویس یا کاربر مشخص می‌شود و در رجیستری که نام آن با OCR شروع می‌شود، ذخیره می‌گردد. به عنوان مثال برای تایمر 1 (که 16 بیتی بود) دو رجیستر OCR وجود دارد که با عنوان OCR1AL و

OCR1AH شناخته می‌شوند. این دو رجیستر هرکدام 8 بیت دارند و در کنار هم یک رجیستر 16 بیتی (L به معنی 8 بیت کم ارزش و H به معنی 8 بیت پر ارزش است) را تشکیل می‌دهد. می‌توان از این دو رجیستر به صورت مستقل از هم نیز استفاده نمود و به ازای هر یک Compare Matchای مجزا دریافت نمود.

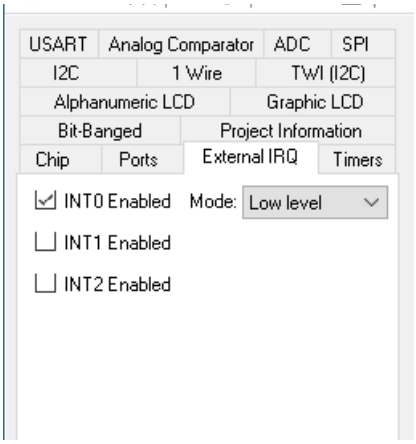
3. CTC (Clear Timer on Compare): در این وضعیت شمارنده از عدد 0 شروع به کار کرده و تا عددی مشخص شمارش را ادامه خواهد داد. این عدد مشخص معمولاً در یکی از رجیسترهای OCR ذخیره می‌گردد. پس از رسیدن به ماکزیمم تایمر دوباره صفر خواهد شد و این روند ادامه می‌یابد. تفاوت این مد کاری با حالت نرمال در این است که در مد CTC می‌توان یک پایه‌ی خروجی بر اساس رسیدن رجیستر تایمر به ماکزیمم تغییر وضعیت داد(صفر به یک یا یک به صفر). در این صورت زمانی از CPU گرفته نخواهد شد و سیستم دچار وقفه نمی‌گردد. در واقع اگر می‌خواستیم در مد نرمال این کار را انجام دهیم باید از وقفه‌ی Compare Match استفاده می‌کردیم.

4. PWM(Pulse Width Modulation): مدولاسیون پهنای باند یا PWM به منظور تولید یک موج مربعی با فرکانسی مشخص به کار می‌رود و با تغییر پهنای باند این موج مربعی می‌توان میزان انرژی آن را تعیین نمود. از PWM برای کنترل سرعت موتور DC، تنظیم نوع (Dimmer) دیودهای نوری و ... استفاده می‌گردد. در مورد PWM در یکی آزمایش‌های بعدی توضیحات بیشتری ارائه خواهد شد.

هر تایمر در AVR به جز رجیستر مربوط به شمارنده و رجیستر OCR، تعدادی رجیستر کنترلی نیز دارد. نام و کارکرد هر بیت از این رجیسترها را می‌توان در دیتاشیت میکروکنترلر مشاهده نمود. اما راه ساده‌تری برای تنظیمات وجود دارد که استفاده از CodeWizard است. با استفاده از CodeWizard و قابلیت Preview می‌توان مقدار مورد نظر هر رجیستر را مشخص نمود. در عمل ما در کد ویزارد تنظیمات مورد نظر خود را انجام می‌دهیم و پس از آن می‌توان از کد تولید شده توسط ویزارد استفاده کنیم یا با مشاهده کد، مقادیر مورد نیاز خود را از کد کپی کنیم.

در این آزمایش به جز شمارنده/تایمر از وقفه خارجی هم استفاده خواهیم کرد. میکروکنترلر مورد استفاده‌ی ما دارای 3 وقفه‌ی خارجی است که با عناوین INT0 تا INT1 در Pinout میکروکنترلر مشخص شده اند. در هنگامی که یکی پایه‌های مربوط به وقفه خارجی فعال شوند، پردازشگر میکروکنترلر سایر عملیات را متوقف کرده و عملیات مربوط به وقفه را پیگیری می‌کند. کدهای مربوط به وقفه را نیز می‌توان با استفاده از کد

ویزارد تولید نمود. همچنین می‌توان تغییری در پایه که موجب فعال شدن وقفه می‌شود را مشخص نمود. به عنوان مثال می‌توان فعال شدن وقفه را در falling edge یا rising edge سیگنال مربوط به پایه‌ی وقفه تنظیم نمود. در بسیاری از کاربردها این سیگنال توسط یک سوییچ یا کلید فشاری تولید می‌شود.



شکل 1-3 حالت‌های ممکن برای وقفه خارجی در CodeWizard

سوال‌های این بخش: (پاسخ این سوالات را به TA ارائه دهید.)

1. فرض کنید که کلاک تایمر 1 (16 بیتی) برابر با 1000 کیلوهرتز انتخاب شود. اگر بخواهیم هر نیم میلی ثانیه یک Compare Match رخ دهد، مقدار رجیسترهای OCR1AL و OCR1AH را محاسبه نمایید.
2. فرض کنید می‌خواهیم از متغییری استفاده کنیم که تنها مقدار 0 یا 1 را به خود اختصاص می‌دهد. با توجه به جدول نوع متغیرها(در آزمایش یک) بهینه‌ترین نوع متغیر از نظر حافظه را برای این کار انتخاب نمایید.
3. نام رجیسترهای تایمر معمولاً به صورت TCNTn مشخص می‌شود که در آن n نشان دهنده‌ی شماره‌ی تایمر است، مثلاً TCNT0. حال فرض کنید می‌خواهیم در کد خود در یک لحظه از مقدار رجیستر تایمر مطلع شویم، به این منظور باید چه کدی بنویسیم؟
4. میکروکنترلر مورد استفاده در آزمایشگاه چند تایمر دارد و هرکدام چند بیتی هستند؟
5. تفاوت مد نرمال و CTC در تایمرها در چیست؟
6. با توجه به Pinout در صورتی که بخواهیم از تمام وقفه‌های خارجی استفاده کنیم، LCD را به چه پورت‌هایی می‌توان متصل نمود؟

- ✓ **بخش دوم:** با توجه به توضیحات زیر، اتصالات سخت‌افزاری مورد نیاز را با استفاده از سیم بر روی برد آموزشی برقرار کنید. سپس کد داده شده در این بخش را تکمیل نموده و برنامه کامپایل را کنید. در نهایت برد را پروگرم کرده و نتیجه را به استاد یا TA ارائه دهید.

سخت افزار: پورت A را به LCD کاراکتری 2 در 16 قرار گرفته روی برد متصل نمایید. در این بخش سه هدف کلی را دنبال می‌کنیم که عبارتند از: 1- استفاده از INT0 و اتصال آن به یک کلید که در زمان فشرده شدن 0 تولید کند. 2- اعمال کلاک با یک سوییچ به تایمر صفر 3- اتصال پایه‌ی PB2 به یک LED یا Buzzer. با توجه به اهداف بیان شده، اتصال‌های لازم را روی برد برقرار کنید.

راهنمایی 1: هر دو سوییچ از را از کلیدهای فشاری انتخاب کنید که هنگام فشرده شدن صفر تولید می‌کنند.

راهنمایی 2: برای مشخص نمودن پایه‌ی مربوط به هدف شماره‌ی 2، از بخش مربوط به تایمر صفر در CodeWizard و همین طور Pinout میکروکنترلر کمک بگیرید.

نرم‌افزار: کد این آزمایش را با استفاده از CodeWizard به نحوی تکمیل کنید که:

1- تایمر یک را در مد نرمال و با فرکانس کلاک 1000 KHz به صورتی تنظیم نمایید که وقفه‌ی مربوط به Compare Match هر نیم میلی ثانیه رخ دهد. در عمل قصد داریم با این تایمر یک پالس روی یکی از پایه‌های میکروکنترلر تولید کنیم. برای این منظور، در کد مربوط به این وقفه پایه PB2 را خروجی تعریف کرده و مقدار آن را به طور متناوب صفر و یک می‌کنیم. در واقع با این کار یک پالس روی پایه‌ی PB2 ایجاد خواهد شد.

2- می‌خواهیم با کمک متغییر pulse_enable کاری کنیم که اگر INT0 فعال شد (کلید متصل به آن فشرده شد)، تولید پالس روی PB2 متوقف شود.

3- علاوه بر این دو مورد می‌خواهیم در حلقه‌ی while مقدار رجیستر مربوط به تایمر صفر را خوانده و آن را روی LCD نمایش دهیم.

```
1-      #include <mega32.h>
2-      #include <lcd.h>
3-      #include <delay.h>
4-      #include <stdio.h>
5-
6-      #asm
7-        .equ __lcd_port=   ; PORTA
8-      #endasm
9-
10-
11-       pulse_enable=1; // Declare Variable Type
12-
13-    // External Interrupt 0 service routine
14-   interrupt [EXT_INT0] void ext_int0_isr(void)
15-       {
16-         // Place your code here
17-          pulse_enable=; //Disable Pulse when INT0 is called
18-       }
19-
20-    // Timer1 output compare A interrupt service routine
21-    interrupt [TIM1_COMPA] void timer1_compa_isr(void)
22-       {
23-          // Place your code here
24-          if(!pulse_enable){
25-                                DDRB=0x04;
26-                                PORTB.2=1;
27-                               }
28-           else{
29-                  if (PORTB.2==1)PORTB.2=0;
30-                  else PORTB.2=1;
31-                 }
32-         }
33-
34-
35-  void main(void)
36-       {
37-          // Declare your local variables here
38-             char lcd_buffer[16];
39-             unsigned char timer0;
40-
41-          // Input/Output Ports initialization
42-          // Port B initialization
43-             PORTB=0x00;
44-             DDRB=0x04;
45-
46-          // Timer/Counter 0 initialization
47-          // Clock source: T0 pin Falling Edge
48-          // Mode: Normal top=0xFF
49-          // OC0 output: Disconnected
50-             TCCR0=;
51-             TCNT0=0x00;
52-              OCR0=0x00;
53-
54-
```

```
55-
56-         // Timer/Counter 1 initialization
57-         // Clock source: System Clock
58-         // Clock value: 1000.000 kHz
59-         // Mode: Normal top=0xFFFF
60-         // OC1A output: Discon.
61-         // OC1B output: Discon.
62-         // Noise Canceler: Off
63-         // Input Capture on Falling Edge
64-         // Timer1 Overflow Interrupt: Off
65-         // Input Capture Interrupt: Off
66-         // Compare A Match Interrupt: On
67-         // Compare B Match Interrupt: Off
68-           TCCR1A=0x00;
69-           TCCR1B=;
70-           TCNT1H=0x00;
71-           TCNT1L=0x00;
72-           ICR1H=0x00;
73-           ICR1L=0x00;
74-           OCR1AH=;
75-           OCR1AL=;
76-           OCR1BH=0x00;
77-           OCR1BL=0x00;
78-
79-        // External Interrupt(s) initialization
80-        // INT0: On
81-        // INT0 Mode: Falling Edge
82-        // INT1: Off
83-        // INT2: Off
84-          GICR|=0x40;
85-          MCUCR=0x02;
86-          MCUCSR=0x00;
87-          GIFR=0x40;
88-
89-        // Timer(s)/Counter(s) Interrupt(s) initialization
90-          TIMSK=0x10;
91-          TWCR=0x00;
92-
93-          lcd_init();
94-
95-         // Global enable interrupts
96-         #asm("sei")
97-
98-          while (1)
99-                {
100-                        // Place your code here
101-                          timer2=;
102-                          lcd_clear();
103-                          sprintf(lcd_buffer,"Timer0 Value: %d",);
104-                          lcd_gotoxy(0,0);
105-                          lcd_puts();
106-                          delay_ms(500);
107-
108-                      }
109-       }
```

توجه:

در خط 1 این نرم‌افزار مشخص شده است که از میکروکنترلر ATMEGA32 استفاده می‌شود. در صورتی که میکروکنترلر ATMEGA16 روی برد شما نصب شده است، این خط را تغییر دهید.

نکات برنامه نویسی:

- دستور #asm("sei") امکان استفاده از وقفه در میکروکنترلر را فعال می‌کند.
- حتماً indentation را رعایت کنید تا ساختار کد مشخص باشد و debugging امکان پذیر شود.

اشتباهات پرتکرار(قبل از کامپایل کردن کد این موارد را بررسی کنید.):

- در انتهای هر خط از کد باید یک semi-colon (;) قرار داشته باشد.
- قبل از کامپایل خطهای 8,12,18,51,70,75,76,94,102,104,106 از کد داده شده را تکمیل کنید. در صورتی که از CodeWizard استفاده کنید شماره‌ی خطها متفاوت خواهد بود و باید به محتوا توجه کنید.
- برای تکمیل خطهای فوق می‌توانید از Preview در CodeWizard استفاده نمایید.
- در خط 30 چون کد مربوط به if در یک خط نوشته شده است، نیازی به استفاده از آکولاد نیست.
- نوع AVR و مقدار کلاک را از منوی Project>Configure>C Compiler انتخاب کنید.
- فایل Source کد C را از منوی Project>Configure>Files به پروژه Add کنید.

پس از بررسی موارد فوق، کد را Build کنید و در صورتی که هیچ خطایی وجود نداشت، میکروکنترلر را پروگرم کنید. دقت کنید که برای پروگرم کردن باید از فایل hex. استفاده کنید. این فایل در محل ذخیره کردن پروژه (این آدرس در بالای CodeVision قابل مشاهده است) پس از ساختن(Build) کد ایجاد خواهد شد. اگر این فایل وجود نداشت، یا آدرس اشتباه انتخاب شده یا در هنگام کامپایل کد خطایی وجود داشته است. در صورت که در زمان کامپایل خطایی وجود داشته باشد در قسمت پایین CodeVision

نمایش داده می‌شود. با مشاهده متن خطا و بررسی کد خطا را اصلاح کنید و در صورت ابهام از TA یا استاد کمک بخواهید.

- ✓ **بخش چهارم:** به سوال‌های زیر پاسخ دهید:

1- شکل موج ایجاد شده روی PB2 را با کمک اسیلوسکوپ مشاهده و رسم نمایید. فرکانس این موج چقدر است؟

2- LED متصل به PB2 در زمان اجرای برنامه چگونه دیده می‌شود؟ این مساله را توضیح دهید.

- ✓ **بخش پنجم(Extension):** کد برنامه را با توجه توضیحات زیر، تغییر دهید و برد را پروگرام کرده و نتایج را به استاد یا TA ارائه دهید.

1. کد قبل را به صورتی تغییر دهید که اگر کلید مرتبط با INT0 دوباره فشرده شود، LED یا Buzzer هم دوباره فعال شود. توجه کنید که در کد اصلی این کلید تنها یک بار مورد استفاده قرار می‌گیرد و با فشردن آن ساخت پالس غیر فعال خواهد شد و هر چند بار که دوباره کلید را بزنیم، پالس دوباره فعال نخواهد شد.

راهنمایی1: در زبان C می‌توان NOT منطقی را به صورت زیر نوشت:

```
Variable =! Variable;        یا        Variable != Variable;
```

در پایان این آزمایش باید موارد زیر را آموخته باشید:

- ✓ کارکرد کلی تایمر/کانتر و Interrupt
- ✓ ساخت کانتر با کمک تایمرها و اعمال کلاک با سوییچ
- ✓ ساخت پالس با فرکانس دلخواه بر روی پایه‌های خروجی
- ✓ استفاده از Interrupt های خارجی

▪ آزمایش شماره 4 : مبدل آنالوگ به دیجیتال (ADC)

- هدف آزمایش: آشنایی ADC داخلی AVR
- نتیجه عملی: استفاده از سنسورهای مختلف و اتصال آن‌ها به ADC و انجام محاسبات مربوط به ADC

✓ **بخش اول:** توضیحات زیر را بخوانید و به سوالات پاسخ دهید.

همانطور که در درس‌های قبلی دیده‌اید، میکروپروسسورها مداراتی دیجیتال هستند که عملیات خود را بر روی داده‌های دیجیتال انجام می‌دهند. در یک میکروکنترلر نیز محاسبات و ارتباط با خارج (توسط پورت‌ها) به صورت دیجیتال صورت می‌گیرد. مثلاً یک پایه از میکرو می‌تواند بیانگر 0 یا 1 منطقی باشد. اما در عمل و در بسیاری از کاربردها از مدارها و سنسورهای آنالوگ استفاده می‌شود و از این رو نیازمندیم که مقدارهای آنالوگ را دیجیتال تبدیل کنیم. برای این منظور در واقع باید یک مقدار پیوسته (آنالوگ) را به یک نمونه گسسته و کوانتیزه شده تبدیل کنیم که برای این کار از مدارهای مبدل آنالوگ به دیجیتال (ADC) استفاده می‌شود. با انواع مدارهای ADC در درس‌های معماری کامپیوتر و میکروپروسسور آشنا شده‌اید. به عنوان یادآوری کافی است به یاد داشته باشید که ADC یک مقدار آنالوگ را دریافت کرده و عددی متناظر با آن (که الزاماً از نظر عددی هم اندازه با مقدار آنالوگ نیست) را به عنوان مقدار دیجیتال تحویل می‌دهد. مدارهای مبدل آنالوگ به دیجیتال کاربردهای بسیاری دارند که برخی از آن‌ها عبارتند از:

- ✓ استفاده از سنسورهای آنالوگ شامل سنسورهای دما، نور، گاز و ....
- ✓ اندازه گیری ولتاژ توسط مدارهای دیجیتال
- ✓ ایجاد ارتباط بین مدارهای آنالوگ و مدارهای دیجیتال
- ✓ کنترل سیستم‌های آنالوگ با میکروکنترلرهای دیجیتال
- ✓ استفاده از مقاومت متغیر به عنوان کنترلر صدا، سرعت و ....

میکروکنترلرهای AVR معمولاً دارای ADC داخلی هستند. به عنوان مثال میکروکنترلرهای ATMEGA16 و ATMEGA32 که در آزمایشگاه از آن‌ها استفاده می‌کنیم، هر دو دارای یک ADC با هشت کانال هستند. محل قرارگیری این کانال‌ها در Pinout میکروکنترلر مشخص شده است. مدار ADC مورد استفاده در این میکروکنترلرها به صورت پیش‌فرض از Resolution یا به عبارتی دیگر دقت ده بیت برخوردار هستند. البته می‌توان دقت این ADCها را به 8 بیت نیز کاهش داد. همانطور که در درس‌های قبلی دیده‌اید، برای کارکرد صحیح ADC به یک ولتاژ مرجع نیاز داریم. همچنین یک پایه با عنوان AVCC برای تغذیه‌ی مدار ADC در AVR تعبیه شده است.

در AVR می‌تواند سه حالت را برای ولتاژ مرجع انتخاب نمود:

1. استفاده از پایه‌ی AVCC: با انتخاب این حالت، ولتاژ متصل شده به پایه‌ی AVCC به عنوان ولتاژ مرجع در نظر گرفته می‌شود. در صورت اتصال ولتاژ جداگانه دقت کنید که با مقدار متصل به VCC بیشتر از 0.3volt اختلاف وجود نداشته باشد.

2. استفاده از پایه‌ی AREF: می‌توان این پایه را به یک ولتاژ دلخواه و مشخص (در محدوده ولتاژ مجاز برای AVR) متصل نمود و این ولتاژ را به عنوان مرجع در نظر گرفت.

3. استفاده از مرجع داخلی: در این وضعیت می‌توان از ولتاژ مرجع داخلی تولید شده توسط خود AVR استفاده نمود. این ولتاژ برابر 2.56 volt است. در دیتاشیت توصیه شده است که در این حالت برای کاهش نویز یک خازن به پایه‌ی AREF متصل شود.

پس از انتخاب ولتاژ مرجع که آن را V_{Ref} می‌نامیم، باید مشخص کنیم که هر عدد دیجیتال معادل چه مقدار آنالوگی است. برای این منظور لازم است که دقت ADC را هم بدانیم. حال فرض کنید که ولتاژ آنالوگ V_{Analog} به ADC متصل شده باشد و دقت ADC مورد استفاده n بیت باشد. در این حالت با توجه به دقت ADC تعداد 2^n عدد دیجیتال خواهیم داشت. با توجه به موارد فوق در نهایت داریم:

$$Digital\ Output = V_{Analog} \times \frac{V_{Ref}}{2^n} \qquad (1\text{-}4)$$

عدد به دست آمده از رابطه‌ی (1-4) معادل دیجیتال ولتاژ آنالوگ ورودی است. اما نکته‌ی بسیار مهمی وجود دارد که در هنگام استفاده از سنسورها باید به آن توجه کنیم. اغلب سنسورهای آنالوگ کمیتی فیزیکی مثل دما، رطوبت، غلظت گاز و ... را اندازه گیری می‌کنند و به ازای مقدار این کمیت‌ها یک ولتاژ در خروجی تولید می‌کنند. رابطه‌ی بین مقدار واقعی کمیت و ولتاژ خروجی می‌تواند خطی یا غیر خطی باشد که معمولاً نمودارهای دقیق آن در دیتاشیت سنسور ارائه می‌شود. حال فرض کنید می‌خواهیم یک سنسور آنالوگ را به AVR متصل نموده و مقدار واقعی کمیتی فیزیکی را اندازه‌گیری کنیم. برای این کار باید ابتدا رابطه‌ی ولتاژ خروجی سنسور که همان V_{Analog} است را با مقدار کمیت فیزیکی مورد نظر، از دیتاشیت استخراج نماییم. این رابطه به صورت $f(V_{Analog})$ است و f می‌تواند تابعی خطی یا غیر خطی باشد. در

نهایت برای مشخص نمودن مقدار کمیت فیزیکی با توجه به رابطه‌ی ولتاژ آنالوگ و مقدار دیجیتال خوانده شده در میکروکنترلر، کافی است از رابطه‌ی زیر استفاده نماییم:

$$Real\ Value = f(V_{Analog}) = \mathrm{f}(Digital\ Output \times \frac{2^n}{V_{Ref}}) \qquad (2\text{-}4)$$

همچنین توجه کنید که برای ADC یک کلاک تعریف می‌شود که توسط آن سرعت اندازه‌گیری مقادیر آنالوگ مشخص می‌شود. انتخاب کلاک بستگی به سرعت تغییرات کمیت مورد نظر دارد. به عنوان مثال دما در محیط معمولی تغییراتی کند دارد اما تغییرات میدان مغناطیسی در یک موتور در حال کار می‌تواند سریع باشد.

مبدل آنالوگ به دیجیتال میکروکنترلر AVR قابلیت‌های دیگری از جمله ارتباط با تایمر، ایجاد وقفه و ... هم دارد که می‌توانید این موارد را از دیتاشیت یا سایر کتاب‌های مرتبط مطالعه کنید.

نکته 1: همانطور که گفته شد، AVR مورد استفاده در آزمایشگاه دارای 8 کانال ADC است که با شماره-های صفر تا هفت مشخص شده‌اند. برای دسترسی به مقدار دیجیتال متناظر با هریک از این کانال‌ها در نرم-افزار CodeVision تابعی با عنوان read_adc() وجود دارد که ورودی آن شماره‌ی کانال ADC و خروجی آن مقدار دیجیتال مورد نظر است.

نکته 2: در میکروکنترلر رجیسترهای مختلفی از جمله ADCSRA و ADMUX برای اعمال تنظیمات ADC طراحی شده‌اند و برای مقدار دهی به آن‌ها باید از دیتاشیت استفاده کرد. اما برای افزایش سرعت عمل معمولاً به منظور مشخص کردن مقدار رجیسترها از Code Wizard استفاده می‌نماییم.

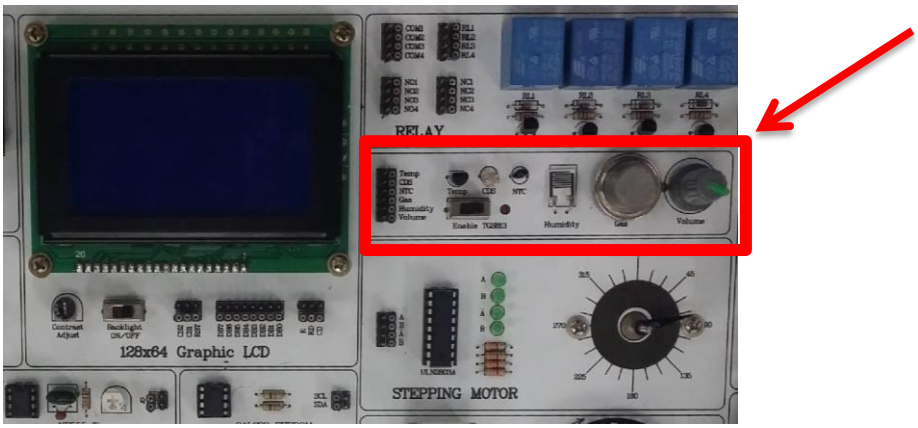
سوال‌های این بخش: (پاسخ این سوالات را به TA ارائه دهید.)

1. فرض کنید ولتاژ مرجع برابر با 4.8volt انتخاب شده باشد. با فرض 10 بیتی بودن میکرو و در اختیار داشتن سنسوری که میزان رطوبت را اندازه‌گیری می‌کند، در حالتی که خروجی سنسور 28 میلی ولت است، عدد دیجیتال متناظر را محاسبه کنید. سپس در نظر بگیرید که بین ولتاژ خروجی سنسور و رطوبت هوا یک رابطه‌ی خطی برقرار باشد به طوری که به ازای هر ده درصد رطوبت هوا، خروجی 8mv افزایش می‌یابد و در رطوبت صفر درصد، خروجی صفر است. حال مقدار رطوبت هوا را محاسبه نمایید.
2. میکروکنترلر ATMEGA32 چند ADC دارد؟ هرکدام دارای چند کانال هستند و چه دقت یا دقت‌هایی دارند؟

3. با توجه به pinout میکروکنترلر، اگر بخواهیم همزمان از ADC ، تمام وقفه‌های خارجی و LCD به صورت همزمان استفاده کنیم، باید LCD را روی کدام پورت میکروکنترلر تنظیم کنیم؟
4. اگر میکروکنترلر را به منبع تغذیه‌ی 4.8 ولتی (VCC) متصل کنیم، محدوده‌ی مجاز برای AVCC چه مقداری خواهد بود؟
5. ولتاژ مرجع داخلی میکروکنترلر مورد استفاده در آزمایشگاه چند ولت است؟
6. کلاک ADC را با توجه به چه چیزی مشخص می‌کنیم؟

- ✓ **بخش دوم:** با توجه به توضیحات زیر، اتصالات سخت‌افزاری مورد نیاز را با استفاده از سیم بر روی برد آموزشی برقرار کنید. سپس کد داده شده در این بخش را تکمیل نموده و برنامه کامپایل را کنید. در نهایت برد را پروگرم کرده و نتیجه را به استاد یا TA ارائه دهید.

سخت افزار: پورت B را به LCD کاراکتری 2 در 16 قرار گرفته روی برد متصل نمایید. در این بخش دو هدف کلی را دنبال می‌کنیم که عبارتند از: 1- اندازه گیری ولتاژ ایجاد شده توسط یک مقاومت متغیر و استفاده از سنسورهای مختلف با هدف اندازه‌گیری دما، رطوبت، شدت نور و غلظت گاز 2- نمایش مقادیر واقعی و مقدار خوانده شده توسط ADC روی صفحه نمایش
برای دستیابی به این اهداف، سنسورهای قرار گرفته در روی برد آزمایشگاه (شکل 1-4) را به ADC متصل نمایید.



شکل 1-4 سنسورهای برد آزمایشگاه

نرم‌افزار: کد این آزمایش را به نحوی تکمیل کنید که:

1- ADC میکروکنترلر با دقت 10 بیت و کلاک 62.5KHz فعال باشد. همچنین پایه‌ی AVCC به عنوان ولتاژ مرجع انتخاب شود.(دقت کنید که در برد آزمایشگاه این پایه توسط PCB به VCC وصل شده است و نیازی به اتصال ولتاژ به آن نیست.) برای اعمال این تنظیمات می‌توانید از دیتاشیت یا CodeWizard استفاده کنید.

2- Volume و سنسورهای قرار گرفته روی برد را با استفاده از ADC میکروکنترلر خوانده و به کمک دیتاشیت یا ولت‌متر (برای Volume) ، محاسبات مربوط به ADC را انجام داده و در کد وارد نمایید.

3- مقدار دیجیتال مربوط به هر ADC، مقدار ولتاژ آنالوگ متناظر و مقدار واقعی کمیت مورد اندازه‌گیری را بر روی LCD نمایش دهید. بدیهی است که نمایش این مقادیر برای همه‌ی کانال‌های ADC به صورت همزمان ممکن نیست به همین دلیل کد مربوط به هر سنسور را به صورت جداگانه پروگرم کنید و از فرمت مثال زیر(Volume) برای نمایش در LCD استفاده نمایید. در این مثال در T نوع سنسور، A مقدار آنالوگ، D مقدار دیجیتال و R مقدار واقعی کمیت را نشان می‌دهد. همچنین از جدول 1-4 برای برای انتخاب مقدار V کمک بگیرید.

شکل 2-4 فرمت نمایش روی LCD

جدول 1-4 انتخاب مقدار T

Volume	**T:V**
Gas	**T:G**
Temperature	**T:T**
Humidity	**T:H**
NTC	**T:N**
CDS	**T:C**

توجه: برای تبدیل مقدار ولتاژ به مقدار واقعی باید از دیتاشیت سنسور مورد نظر استفاده کنید.

توجه: برای سنسورهای **شدت نور، NTC، گاز** و همین‌طور برای **Volume** نیازی به تبدیل به مقدار واقعی نیست و فقط ولتاژ خوانده شده را نمایش دهید.

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

#asm
  .equ __lcd_port=   ; PORTB
#endasm

#define ADC_VREF_TYPE 0x     //ENTER VALUE

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

void main(void)
  {
        char lcd_buffer[16];
        unsigned int DigitalVal;
        float RealVal;
        float AnalogV;
      // ADC initialization, ADC Clock Frequency: 62.500 KHz
      // ADC Voltage Reference: AVCC pin
        ADMUX=ADC_VREF_TYPE & 0xFF;
        ADCSRA=   ; // Enter Value

      lcd_init(16);
      while (1)
            {
                  lcd_clear();
                  DigitalVal=read_adc(// ADC Channel) ;
                  AnalogV=// write equation based on Vref;
                  RealVal=// write equation based on sensor type;
                  lcd_clear();
                  sprintf(lcd_buffer,"T://Use Table 4.1     ,A:%4.2f ",AnalogV);
                  lcd_gotoxy(0,0);
                  lcd_puts(lcd_buffer);
                  delay_ms(500);
                  sprintf(lcd_buffer,"D:%d     ,R:%4.2f ",DigitalVal,RealVal);
                  lcd_gotoxy(0,1);
                  lcd_puts(lcd_buffer);
                  delay_ms(500);


            }
}
```

توجه:

در خط 1 این نرم‌افزار مشخص شده است که از میکروکنترلر ATMEGA32 استفاده می‌شود. در صورتی که میکروکنترلر ATMEGA16 روی برد شما نصب شده است، این خط را تغییر دهید.

نکات برنامه نویسی:

- دستور delay_us() در خط 17 مشابه با delay_ms() است با این تفاوت که میزان تاخیر در ورودی تابع به میکروثانیه مشخص می‌شود.
- همانطور که قبلاً گفتیم در کدویژن برای خواندن مقدار ADC از تابعی با نام read_adc() استفاده شده است. این تابع در بین خطوط 13 تا 24 تعریف شده است. به طور کلی در زبان C اگر تابعی مورد استفاده قرار می‌گیرد یا باید در یکی از سربرگ‌ها قرار داشته باشد و یا در خود کد تعریف شود.
- عبارت %4.2f در خط 45 و 49 بیانگر این است متغیری که باید در یک string قرار بگیرد، در کل دارای 4 کاراکتر(شامل نقطه ممیز) و دو رقم اعشار است. مثلاً 2.56 یا 4.37

اشتباهات پرتکرار(قبل از کامپایل کردن کد این موارد را بررسی کنید.):

- قبل از کامپایل خط‌های 43,42,41,35,10 از کد داده شده را تکمیل کنید. در صورتی که از CodeWizard استفاده کنید شماره‌ی خط‌ها متفاوت خواهد بود و باید به محتوا توجه کنید.
- برای تکمیل خط‌های فوق می‌توانید از Preview در CodeWizard استفاده نمایید.
- نوع AVR و مقدار کلاک را از منوی Project>Configure>C Compiler انتخاب کنید.
- فایل Source کد C را از منوی Project>Configure>Files به پروژه Add کنید.
- **مهم**: با توجه به اینکه در کد دستور sprint را برای متغیرهای float استفاده کرده‌ایم، باید قبل از کامپایل کد از مسیر Project>Configure>C Compiler و بخش (s)printf features (در سمت چپ صفحه‌ی باز شده) عبارت float, width, precision را انتخاب نمایید.

پس از بررسی موارد فوق، کد را Build کنید و در صورتی که هیچ خطایی وجود نداشت، میکروکنترلر را پروگرم کنید. در صورتی که در زمان کامپایل خطایی وجود داشت، با مشاهده متن خطا و بررسی کد خطا را اصلاح کنید و در صورت ابهام از TA یا استاد کمک بخواهید.

- ✓ **بخش چهارم:** به سوال‌های زیر پاسخ دهید:

1- سنسور دمای مورد استفاده از نوع LM35 می‌باشد. با توجه به دیتاشیت این سنسور با فرض اینکه نمودار ارائه شده را تقریباً خطی فرض کنیم، میزان حساسیت این سنسور چند میلی ولت به ازای درجه‌ی سانتی‌گراد است؟ محاسبات مربوط به ADC برای این سنسور را انجام دهید.

2- مقاومت متغیر(Volume) متصل شده به ADC چه کاربردهایی می‌تواند داشته باشد؟

- ✓ **بخش پنجم(Extension):** کد برنامه را با توجه توضیحات زیر، تغییر دهید و برد را پروگرام کرده و نتایج را به استاد یا TA ارائه دهید.

ابتدا کد را فقط برای سنسور دمای LM35 روی برد پروگرم کنید. بر روی LCD مقدار دمای محیط را نشان دهید. به نظر شما چرا این مقدار دارای نوسان زیادی است؟ ابتدا راه حل‌هایی تئوری برای حذف این نوسان ارائه دهید. سپس کد راه حل مورد تایید استاد یا TA را بنویسید و نتیجه را با حالت قبل مقایسه کنید.

راهنمایی1: ابتدا توجه کنید که مفهوم فرکانس به معنی میزان تغییرات در واحد زمان است. حال با توجه به اینکه می‌دانیم تغییرات دمای یک محیط عادی سرعت کمی دارد، پس فرکانس سیگنال دما نسبتاً پایین است. اما چیزی که روی LCD نمایش داده می‌شود دارای تغییرات سریع است و در واقع فرکانس بالایی دارد. پس در عمل یک سیگنال فرکانس بالا و یک سیگنال فرکانس پایین باهم جمع شده‌اند. راه حلی آنالوگ برای جداسازی این سیگنال‌ها و حفظ سیگنال مطلوب ارائه دهید.

راهنمایی 2: فرض کنید سیگنال فرکانس بالا نویزی با واریانس نامشخص و میانگین صفر باشد. حال با توجه به این مساله و راهنمایی 1 راهکاری ارائه دهید که بتوان نویز جمع شده با سیگنال را حذف نمود. توجه کنید که راهکار ارائه شده در این مرحله باید قابل پیاده سازی به صورت کد نویسی باشد.

در پایان این آزمایش باید موارد زیر را آموخته باشید

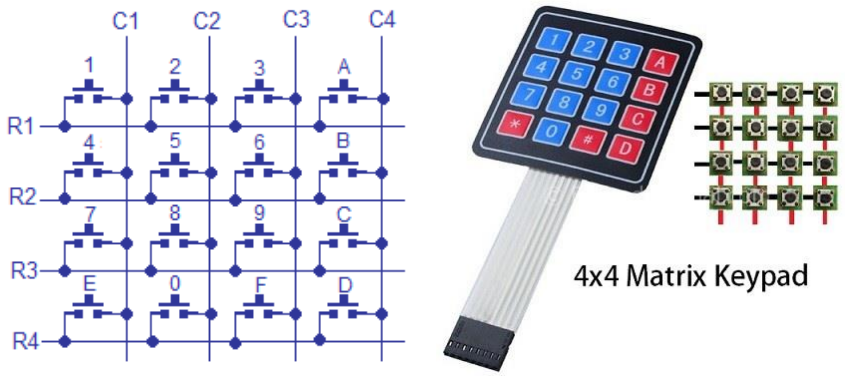
- ✓ کارکرد کلی ADC
- ✓ انجام محاسبات مربوط به ADC
- ✓ استفاده از سنسورهای مختلف و استفاده از دیتاشیت سنسورها
- ✓ حذف نوسانات مقدار خوانده شده توسط ADC با کد نویسی

▪ آزمایش شماره 5 : keypad و EEPROM

- هدف آزمایش: راه اندازی keypad و آشنایی با حافظه‌ی eeprom
- نتیجه عملی: وارد کردن اعداد مختلف توسط keypad و ذخیره‌ی این اعداد در EEPROM

✓ **بخش اول:** توضیحات زیر را بخوانید و به سوالات پاسخ دهید.

صفحه کلید 4 در 4 که به آن Hex-keypad نیز می‌گویند، یکی از پرکاربردترین ابزارهای ورود اطلاعات به میکروکنترلر توسط کاربر می‌باشد. به عنوان مثال در یک سیستم قفل الکترونیکی، ماشین حساب یا هر سیستم دیگری که در آن نیاز به ورود اعداد باشد می‌توان از کی‌پد استفاده نمود. صفحه کلید مورد استفاده در این پروژه دارای 4 ردیف و 4 ستون و مجموعاً 16 ستون می‌باشد. شکل زیر نمونه‌ای از این نوع کی‌پد و شماتیک آن را نمایش می‌دهد.



شکل 1-5 شماتیک کی‌پد(راست) – نمونه‌ای از کی‌پد(چپ)

فشردن هر کلید در واقع سطر و ستون و متناظر آن کلید را به هم متصل می‌کند. میکروکنترلر بایستی با تشخیص سطر و ستون کلید فشرده شده، شماره آن کلید را استخراج کند. راهکارهای متنوعی برای اسکن کردن کی‌پد و شناسایی کلید فشرده شده وجود دارد. می‌توان از Ring counter نرم‌افزاری، وقفه یا سایر روش‌های ممکن برای اسکن کی‌پد استفاده نمود. همچنین می‌توان سطرها یا ستون‌ها را به صورت pull-up یا pull-down متصل نمود که با توجه به این موارد برنامه‌ی مورد استفاده برای شناسایی کلید فشرده شده تغییر خواهد. نمود.

در ادامه به معرفی حافظه‌ی EEPROM می‌پردازیم. EEPROM یکی از حافظه‌های داخلی میکروکنترلر می‌باشد که خلاصه شده‌ی عبارت Electrically Erasable Programmable Read Only Memory

می‌باشد. این حافظه امکان ذخیره‌ی دائمی اطلاعات به کاربر را می‌دهد. یعنی با قطع برق یا منبع تغذیه، اطلاعات درون EEPROM پاک نخواهند شد(برخلاف حافظه‌ی SRAM). یک راه ساده برای استفاده از این حافظه تعریف متغیرها درون این حافظه است بدین صورت که کافی است قبل از تعریف نوع متغیر عبارت eeprom را تایپ کنیم. نکته‌ی دیگر در مورد مقداردهی اولیه متغیرهاست. زمانی که یک متغیر در eeprom تعریف می‌شود، مقدار اولیه‌ی تمام بیت‌های آن 1 خواهد بود. به عنوان مثال یک متغیر از نوع int که 16 بیتی باشد مقداری برابر با 0xFFFF خواهد داشت. اما باید توجه کنید که اگر در ابتدای برنامه متغیر را مقدار دهی کنیم، با هربار اجرای مجدد برنامه (Reset شدن یا قطع و وصل منبع تغذیه) این متغیر دوباره مقدار خواهد گرفت و مقدار ذخیره شده در آن تغییر خواهد کرد. به همین دلیل معمولاً مقدار دهی اولیه متغیر در برنامه‌ای جداگانه صورت می‌گیرد. اما می‌توان با یک if و در صورتی که مقدار تمام بیت‌های متغیر یک بود، آن را تغییر داد. در این صورت دیگر تغییری در مقدارهای بعدی ایجاد نخواهد شد.

سوال‌های این بخش: (پاسخ این سوالات را به TA ارائه دهید.)

1. از درس معماری کامپیوتر به یاد دارید که گاهی وقتی یک کلید را فشار می‌دهیم، این کلید چندین بار خوانده می‌شود. این مساله با عنوان debouncin شناخته می‌شود. راه حلی سخت افزاری و سپس راه حلی نرم‌افزاری برای مقابله با این مشکل ارائه کنید.
2. دستوری بنویسید که یک متغیر با نام eeRes از نوع int را در eeprom ایجاد کند.
3. اگر ستون‌های کی‌پد را pull-up کنیم و کلید دوم از ستون سوم را فشار دهیم و همزمان مقدار ولتاژ سطرهای کی‌پد را بخوانیم، این ولتاژ به صورت دیجیتال(هر سطر 0 یا 1) به چه صورتی خواهد بود.

- ✓ **بخش دوم:** با توجه به توضیحات زیر، اتصالات سخت‌افزاری مورد نیاز را با استفاده از سیم بر روی برد آموزشی برقرار کنید. سپس کد داده شده در این بخش را تکمیل نموده و برنامه کامپایل را کنید. در نهایت برد را پروگرم کرده و نتیجه را به استاد یا TA ارائه دهید.

سخت افزار: پورت B را به LCD کاراکتری 2 در 16 قرار گرفته روی برد متصل نمایید. در این بخش دو هدف کلی را دنبال می‌کنیم که عبارتند از: 1- اسکن کی‌پد و نمایش اعداد روی صفحه نمایش 2- ذخیره اعداد در eeprom

بدین منظور، پین‌های PD0 تا PD3 را به ترتیب به سطرهای 1 تا 4 (R1 تا R4) و پین‌های PD4 تا PD7 را به ستون‌های 1 تا 4 (C1 تا C4) از کی‌پد متصل نمایید.

نرم‌افزار: کد این آزمایش را به نحوی تکمیل کنید که:

1- برنامه را به صورتی تکمیل کنید که ابتدا تمامی ستون‌ها 1 شوند. آن‌گاه سطرها مقدار قرار گرفته روی پورت D را بخوانید.

2- با تاخیری بسیار کم، تمام سطرها یک می‌شوند و دوباره مقدار قرار گرفته روی پورت D را می‌خوانیم.

3- در نهایت این دو عدد خوانده شده باهم AND خواهند شد که در نتیجه 16 عدد یکتا ایجاد خواهد شد. با تکمیل جدول زیر نگاشت مورد نیاز برای نمایش اعداد 0 تا 9 و سایر کلیدها را بدست آورید.

جدول 5-1 نگاشت کی‌پد

Key	Resulting Value	Value to Display
0		0
1		1
2		2
3		3
4		4
5		5
6		6
7		7
8		8
9		9
*		10
#		11
F1		241
F2		242
F3		243
F4		244

توجه: برای محاسبه‌ی Resulting Value می‌توانید از برد آزمایشگاه و نمایش متغیر scan_result استفاده کنید.

4- در نهایت متغیری از نوع int با نام FinalRes در EEPROM تعریف کنید. که عدد نهایی در آن ذخیره شود.

```
1-      #include <mega32.h>
2-      #include <lcd.h>
3-      #include <delay.h>
4-
5-      #asm
6-        .equ __lcd_port=   ; PORTB
7-      #endasm
8-
9-      eeprom unsigned int FinalRes;
10-
11-     if (FinalRes=0xFFFF){FinalRes=0X0000;}; \\initialization
12-
13-
14-     flash unsigned char key_table[16]={  ,   ,   ,   ,
15-                                          ,   ,   ,   ,
16-                                          ,   ,   ,   ,
17-                                          ,   ,   ,      };
18-
19-
20-    flash unsigned char key_convert[16]={ 0x01, 0x02, 0x03, 0xF2,
21-                                          0x04, 0x05, 0x06, 0xF3,
22-                                          0x07, 0x08, 0x09, 0xF4,
23-                                          0x0A, 0x0A,0x0C,0xF1};
24-
25-   void main(void)
26-        {
27-             char lcd_buffer[16];
28-             unsigned char key_num, scan_result, final_key;
29-
30-
31-           lcd_init(16);
32-
33-           while (1)
34-                {
35-                      DDRD=0x0F;
36-                      PORTD=0x0F;
37-                      delay_ms(3);
38-                      scan_result=PIND;
39-
40-                      DDRD=0xF0;
41-                      PORTD=0xF0;
42-                      delay_ms(3);
43-                      scan_result=scan_result & PIND;
44-
45-                      delay_ms(15);
46-
47-                      final_key=0;
48-
49-                      if(scan_result !=0x00){
50-                          for (key_num=0; key_num<16; key_num++){
51-                                    final_key=key_convert[key_num];
52-                              }
53-                       }
54-
55-                      FinalRes=final_key;
56-
```

```
57-                 lcd_clear();
58-                 lcd_gotoxy(0,0);
59-                 sprintf(lcd_buffer,"Pressed Key: %d",final_key);
60-                 lcd_puts(lcd_buffer);
61-                 delay_ms(500);
62-
63-                 lcd_gotoxy(0,1);
64-                 sprintf(lcd_buffer,"eeprom key: %d",FinalRes);
65-                 lcd_puts(lcd_buffer);
66-                 delay_ms(500);
67-
68-
69-             }
70-     }
```

توجه:

در خط 1 این نرم‌افزار مشخص شده است که از میکروکنترلر ATMEGA32 استفاده می‌شود. در صورتی که میکروکنترلر ATMEGA16 روی برد شما نصب شده است، این خط را تغییر دهید.

نکات برنامه نویسی:

- دستور خط 11 برای مقدار دهی اولیه متغیر eeprom استفاده شده است.

اشتباهات پرتکرار(قبل از کامپایل کردن کد این موارد را بررسی کنید.):

- قبل از کامپایل خط‌های 6 و 14 تا 18 از کد داده شده را تکمیل کنید. در صورتی که از CodeWizard استفاده کنید شماره‌ی خط‌ها متفاوت خواهد بود و باید به محتوا توجه کنید.

پس از بررسی موارد فوق، کد را Build کنید و در صورتی که هیچ خطایی وجود نداشت، میکروکنترلر را پروگرم کنید. در صورتی که در زمان کامپایل خطایی وجود داشت، با مشاهده متن خطا و بررسی کد خطا را اصلاح کنید و در صورت ابهام از TA یا استاد کمک بخواهید.

✓ **بخش چهارم(Extension):** کد برنامه را با توجه توضیحات زیر، تغییر دهید و برد را پروگرام کرده و نتایج را به استاد یا TA ارائه دهید.

1- تغییری کوچک در کد برنامه ایجاد کنید تا بتوان اعداد چند رقمی را دریافت نمود. به عنوان مثال اگر ابتدا 1 و سپس 2 فشرده شود میکروکنترلر باید عدد 12 را نمایش دهد.

توجه: دقت کنید که این عدد چند رقمی باید در میکرو هم به صورت یک عدد چند رقمی ذخیره شود. یعنی اگر عدد 123 نمایش داده می‌شود در میکرو هم متغیری شامل همین عدد باشد و از آرایه یا نمایش تک تک رقم‌ها کنار هم استفاده نکنید.

2- کد خواندن از کی‌پد را به صورت یک تابع در زبان C بنویسید.

راهنمایی: یک تابع در زبان C به فرمت زیر نوشته می‌شود:

```
OutputType FunctionName( type input 1, type input 2, ...){..
                                                          ..};
```

OutPutType نوع خروجی تابع (مشابه نوع متغیرها)، type نوع ورودی و FunctionName نام تابع را مشخص می‌کند. کد تابع در بین آکولادها نوشته می‌شود و در نهایت با نوشتن دستور return قبل از نام متغیر خروجی، می‌توان آن را به بیرون تابع فرستاد. اگر یک تابع ورودی یا خروجی نداشته باشد، type متناظر با آن به صورت void خواهد بود.

برای استفاده از تابع در یک برنامه می‌توان آن را در یک سربرگ قرار داد و سربرگ را به کد اصلی اضافه نمود. همچنین می‌توان کل کد تابع را بعد از کدهای مربوط به main نوشت. در این صورت باید قبل از تابع main یک نمونه از فراخوانی تابع اضافه شود تا برنامه تابع را بشناسد.

پس از اضافه کردن تابع به نرم افزار، می‌توان در هر بخشی از برنامه(مثلاً درون while) از آن استفاده نمود.

3- (اختیاری – نمره اضافی): سخت افزار و نرم‌افزار را برای راه‌اندازی کی‌پد با استفاده از وقفه تغییر دهید.

در پایان این آزمایش باید موارد زیر را آموخته باشید

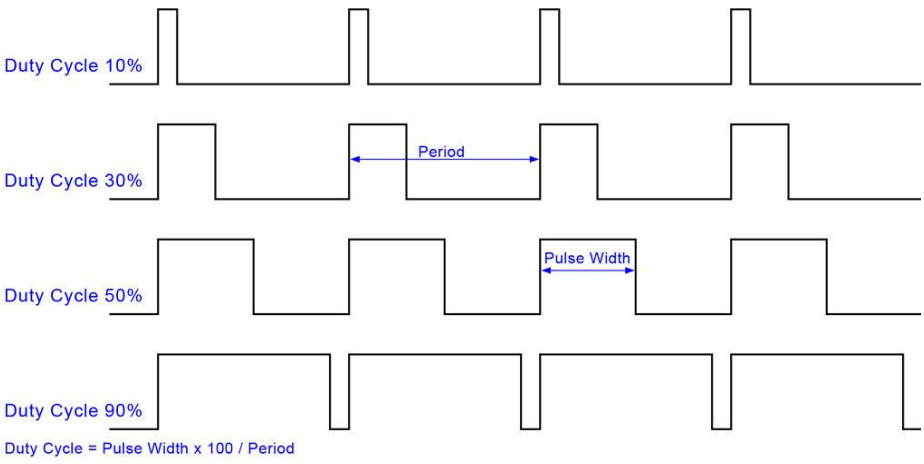
- ✓ کار با کی‌پد و اسکن کلیدهای آن
- ✓ ذخیره یک عدد در EEPROM و فراخوانی آن
- ✓ نوشتن یک تابع برای کار با کی‌پد
- ✓ دریافت اعداد چند رقمی از کی‌پد
- ✓

▪ آزمایش شماره 6 : موتورهای DC و Stepper

- هدف آزمایش: آشنایی با PWM و کنترل دور موتور DC ، راه اندازی و کنترل Stepper Motor
- نتیجه عملی: پس از این آزمایش می‌توانید از PWM برای کاربردهای مختلف از جمله کنترل سرعت موتور DC استفاده کنید. علاوه بر این قادر به راه اندازی و کنترل موتورهای Stepper خواهید بود.

✓ **بخش اول (موتور DC و PWM):** توضیحات زیر را بخوانید و به سوالات پاسخ دهید.

هر موج مربعی، دارای دو مشخصه اصلی است که عبارتند از فرکانس و سطح ولتاژ. اما این دو مشخصه شکل موج را مشخص نمی‌کند. برای درک بهتر این موضوع به شکل 6-1 دقت کنید.



شکل 6-1 چند موج مربعی با فرکانس و سطح ولتاژ یکسان

همانطور که مشاهده می‌کنید هر چهار شکل موج دارای فرکانس و سطح ولتاژ یکسان می‌باشند. اما مدت زمان یک بودن نسبت به کل دوره‌ی تناوب در این موج‌ها متفاوت است. مدت زمان یک بودن (سطح ولتاژ بالا) نسبت به کل دوره‌ی تناوب را با عبارت Duty Cycle مشخص می‌کنند. می‌توان نشان داد که ولتاژ DC و ولتاژ RMS برای یک شکل موج مربعی از روابط زیر به دست می‌آید:

$$V_{DC} = V_{High} \times Duty\ Cycle$$

$$V_{RMS} = V_{High} \times \sqrt{Duty\ Cycle}$$

پس با تغییر دادن Duty Cycle موج مربعی می‌توان ولتاژهای RMS تولید نمود و همانطور که می‌دانید ولتاژ RMS با میزان انرژی منتقل شده توسط موج رابطه‌ای مستقیم دارد. پس می‌توان از این موضوع برای کنترل سرعت موتور DC، کنترل نور یک لامپ و ... استفاده نمود. به تغییر Duty Cycle یک موج مربعی جهت تولید ولتاژ متغیر Pulse Width Modulation یا به اختصار PWM می‌گوییم. تولید موج PWM در میکروکنترلرها توسط تایمرها صورت می‌گیرد.(یادآوری: در آزمایش 3 و معرفی تایمرها به این مساله اشاره شد.) در پروژه‌های عملی، به دلیل اینکه پایه‌های ورودی/خروجی میکروکنترلر جریان دهی کمی دارند، برای اتصال موتور DC یا لامپ از IC های درایور استفاده می‌گردد. همچنین در میکروکنترلر پایه‌ای برای هر تایمر در نظر گرفته شده و موج PWM تولیدی بر روی این پایه ایجاد خواهد شد. پس کافی است این پایه را به IC درایور متصل نماییم.

برای تولید این موج در میکروکنترلر، کافی است تایمر را در حالت PWM قرار دهیم. البته توجه کنید که برخی از تایمرها می‌توانند چند PWM به صورت همزمان تولید کنند و هر PWM یک رجیستر OCR و یک پایه‌ی خروجی مربوط به خود را دارد. در این حالت تایمر از 0 شروع به شمارش می‌کند و پایه‌ی خروجی مربوط به این تایمر 0 یا 1 خواهد بود. (با توجه به Inverted یا Non Inverted بودن خروجی). تایمر شمارش را تا رسیدن به مقدار ذخیره شده در رجیستر OCR ادامه می‌دهد و زمانی که به این مقدار رسید، وضعیت خروجی را تغییر خواهد داد. سپس شمارش ادامه پیدا کرده تا تایمر به ماکزیمم مقدار رجیستر مربوط به شمارش برسد(مثلاً 255 برای 8 بیتی) و در این حالت مقدار خروجی بازهم برعکس خواهد شد(صفر به یک یا یک به صفر) و این روند ادامه می‌یابد. در واقع کلاک تایمر و تعداد بیت‌های آن مشخص کننده‌ی فرکانس موج مربعی و نسبت مقدار رجیستر OCR به ماکزیمم مقدار رجیستر شمارش در تایمر، Duty Cycle را نشان می‌دهد. در میکروکنترلر روش‌های متنوعی برای تولید PWM وجود دارد (از جمله Fast PWM و Phase Correct PWM) که می‌توانید در مورد آن‌ها در دیتاشیت میکروکنترلر یا کتاب‌های مرتبط مطالعه نمایید.

سوال‌های این بخش: (پاسخ این سوالات را به TA ارائه دهید.)

1. کدام پایه از میکروکنترلر برای ایجاد PWM مربوط به تایمر 1 مورد استفاده قرار می‌گیرد؟
2. توضیح دهید که چرا به کمک PWM می‌توان سرعت موتور DC را کنترل نمود.
3. اگر ولتاژ سطح بالای یک موج مربعی 5 ولت باشد، مقدار ولتاژ DC و RMS را برای Duty Cycle به میزان 10 درصد را محاسبه نمایید.
4. اگر تایمر شمارش ما 8 بیتی باشد، برای داشتن Duty Cycle به میزان 40 درصد، باید مقدار رجیستر OCR را چند انتخاب کنیم؟

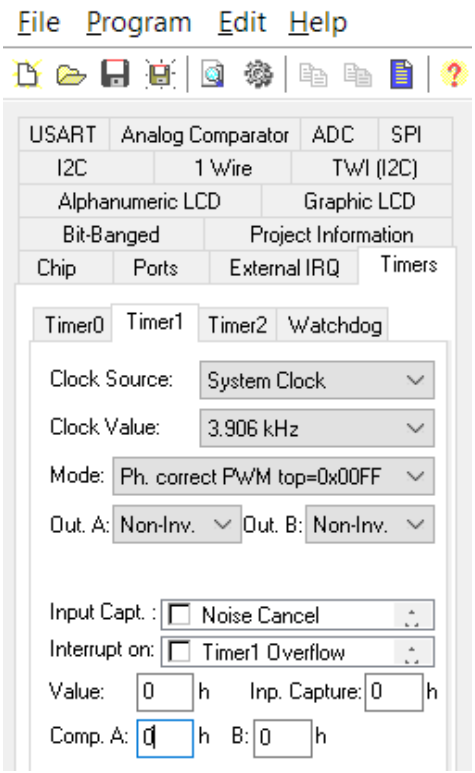
✓ **بخش دوم:** با توجه به توضیحات زیر، اتصالات سخت‌افزاری مورد نیاز را با استفاده از سیم بر روی برد آموزشی برقرار کنید. سپس کد داده شده در این بخش را تکمیل نموده و برنامه کامپایل را کنید. در نهایت برد را پروگرم کرده و نتیجه را به استاد یا TA ارائه دهید.

سخت افزار: خروجی مربوط به OCRA از تایمر 1 را به درایور موتور DC وصل کنید. خروجی مربوط به OCRB از تایمر 1 را هم به اسیلوسکوپ وصل نمایید.

نرم‌افزار: کد این آزمایش را به نحوی تکمیل کنید که:

1- تایمر 1 در حالت Phase Correct PWM به صورت 8 بیتی و با کلاک 3.906 kHz فعال باشد و هر دو خروجی آن به صورت Non Inverted باشند.
2- که موتور با موجی که Duty Cycle ای 50 درصدی دارد فعال باشد.
3- موج روی پایه‌ی مربوط به OCRB دارای Duty Cycle به میزان 20 درصد باشد.

راهنمایی: می‌توانید از CodeWizard کمک بگیرید.



شکل 2-6 ایجاد کد PWM به کمک CodeWizard

```
1-  #include <mega32.h>
2-
3-
4-   void main(void)
5-       {
6-            DDRA=  ;
7-            PORTA=;
8-
9-            DDRB=  ;
10-           PORTB=;
11-
12-           DDRC=  ;
13-           PORTC=;
14-
15-           DDRD=  ;
16-           PORTD=;
17-
18-           TCCR1A=;
19-           TCCR1B=;
20-           TCNT1H=0x00;
21-           TCNT1L=0x00;
22-           ICR1H=0x00;
23-           ICR1L=0x00;
24-           OCR1AH=;
25-           OCR1AL=;
26-           OCR1BH=;
27-           OCR1BL=;
28-
29-
30-           TIMSK=0x00;
31-
32-         while (1)
33-             {
34-
35-
36-              }
37-   }
```

توجه:

در خط 1 این نرم‌افزار مشخص شده است که از میکروکنترلر ATMEGA32 استفاده می‌شود. در صورتی که میکروکنترلر ATMEGA16 روی برد شما نصب شده است، این خط را تغییر دهید

اشتباهات پرتکرار(قبل از کامپایل کردن کد این موارد را بررسی کنید.):

- قبل از کامپایل خطهای 6 تا 27 از کد داده شده را تکمیل کنید. در صورتی که از CodeWizard استفاده کنید شماره‌ی خطها متفاوت خواهد بود و باید به محتوا توجه کنید.

پس از بررسی موارد فوق، کد را Build کنید و در صورتی که هیچ خطایی وجود نداشت، میکروکنترلر را پروگرم کنید. در صورتی که در زمان کامپایل خطایی وجود داشت، با مشاهده متن خطا و بررسی کد خطا را اصلاح کنید و در صورت ابهام از TA یا استاد کمک بخواهید.

- ✓ **بخش چهارم(Extension):** کد برنامه را با توجه توضیحات زیر، تغییر دهید و برد را پروگرام کرده و نتایج را به استاد یا TA ارائه دهید.

1- با تغییر مقدار رجیستر OCR در کد، تاثیر Duty Cycle بر دور موتور و همچنین شکل موج را بر روی اسیلوسکوپ بررسی نمایید.

2- با کمک ADC، از مقاومت متغیر (ولوم) روی برد استفاده کرده و برنامه را به صورتی تغییر دهید که بتوان با استفاده از ولوم، سرعت موتور DC را کنترل کرد.

3- تفاوت Inverted و Non Inverted بودن خروجی را بر شکل موج و سرعت موتور بررسی نمایید.

- ✓ **بخش پنج (Stepper Motor):** توضیحات زیر را بخوانید و به سوالات پاسخ دهید.

موتور استپر یا پله‌ای یک موتور DC براشلس است که یک دور چرخش کامل آن به گام‌هایی مساوی تقسیم شده است. موتورهای پله‌ای به دلیل قابلیت تبدیل پالس‌های دیجیتال به حرکت مکانیکی به سادگی توسط میکروکنترلرها قابل کنترل هستند. محور این موتورها به ازای هر پالس یک گام حرکت نموده و در هر گام از حرکت به اندازه‌ی معین(بر حسب درجه) جابجا می‌شود. مقدار این جابجایی معمولاً بین 1.8 درجه تا 90 درجه است و بستگی به ساختار موتور و نوع کنترل نمودن آن دارد. می‌توانید جزییات مربوط به ساختار و کارکرد این موتورها را در منابع مرتبط مطالعه نمایید.

در این آزمایش قصد داریم که یک موتور استپر را راه اندازی نماییم.

سوال‌های این بخش: (پاسخ این سوالات را به TA ارائه دهید.)

1. به نظر شما موتور استپر چه کاربردهایی دارد؟
2. روش راه اندازی موتور استپر را با جستجو در اینترنت یا کتاب‌های مرتبط پیدا کنید.

- ✓ **بخش ششم:** با توجه به توضیحات زیر، اتصالات سخت‌افزاری مورد نیاز را با استفاده از سیم بر روی برد آموزشی برقرار کنید. سپس کد داده شده در این بخش را تکمیل نموده و برنامه کامپایل را کنید. در نهایت برد را پروگرم کرده و نتیجه را به استاد یا TA ارائه دهید.

سخت افزار: موتور استپر را به پایه‌های میکروکنترلر وصل کنید. دو کلید از برد آموزش را هم به دو پین از میکرو کنترلر وصل کنید.

نرم‌افزار: کد این آزمایش را به نحوی بنویسید:

1- با هر بارفشردن کلید اول موتور یک گام به صورت ساعت‌گرد و با فشردن کلید دوم، موتور یک گام به صورت پاد ساعت گرد حرکت کند.

بخش هفتم(Extension): کد برنامه را با توجه توضیحات زیر، تغییر دهید و برد را پروگرام کرده و نتایج را به استاد یا TA ارائه دهید.

1- با توجه به کد نوشته شده، میزان گام موتور بر حسب درجه را محاسبه کنید. راهی پیشنهاد کنید که بتوان گام موتور را به نصف کاهش داد.

در پایان این آزمایش باید موارد زیر را آموخته باشید:

- ✓ ساخت PWM با استفاده از تایمر
- ✓ راه اندازی و کنترل سرعت موتور DC
- ✓ راه اندازی و کنترل موتور Stepper