دستور کار آزمایشگاه

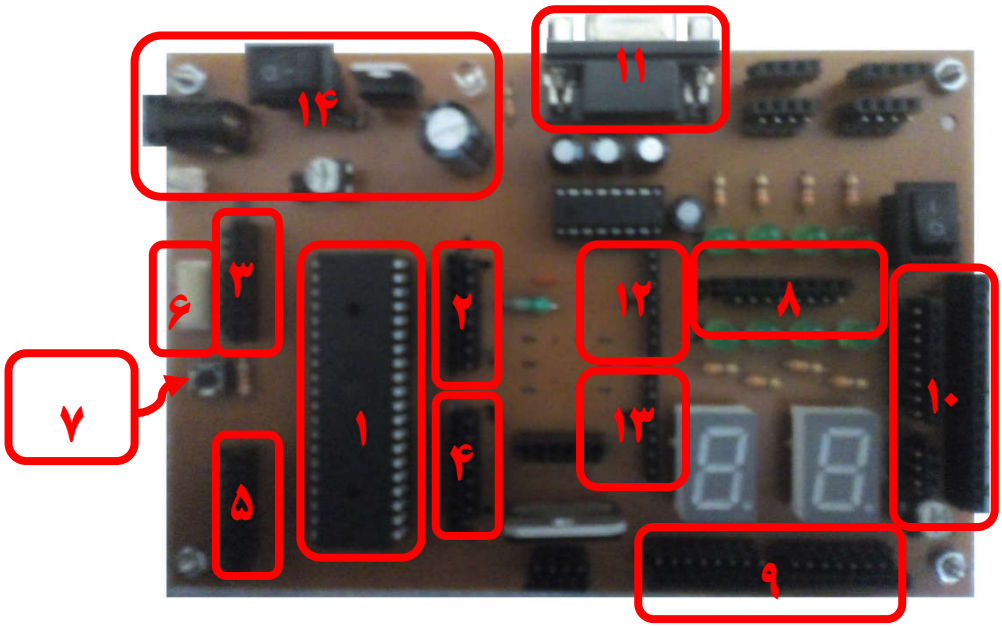
ریزپردازنده

مدرس: علی صادقی

عنوان	صفحه

مقدمه

ابتدا به معرفی برد آموزشی موجود در آزمایشگاه دیجتال می‌پردازیم. برد موجود در آزمایشگاه برای آموزش کار با امکانات میکرکنترلرهای AVR (تراشه Atmega32) طراحی شده است. اکثر قابلیتهای میکروکنترلر atmega32 با استفاده از این برد آموزشی قابل تست و ارزیابی می‌باشد. شمای این برد در شکل زیر نشان داده شده است:



در ادامه بخش‌های مختلف برد توضیح داده شده است:

۱) میکورکنترلر AVR موجود در برد (ATMEGA32).

۲) پین هدر مربوط به پورت A میکروکنترلر.

۳) پین هدر مربوط به پورت B میکروکنترلر

۴) پین هدر مربوط به پورت C میکروکنترلر

۵) پین هدر مربوط به پورت D میکروکنترلر

۶) پین هدر مربوط به برنامه‌ریزی (پروگرام کردن) تراشه به صورت ISP.

توجه شود که برای برنامه‌ریزی کردن میکروکنترلر باید تغذیه مدار وصل باشد.

۷) دکمه فشاری برای ریست کردن(راه‌اندازی مجدد) تراشه.

۸) پین هدر مربوط به LEDهای روی برد.

۹) پین هدر مربوط به نمایشگرهای Seven-Segment روی برد.

۱۰)پین هدرهای مربوط به نمایشگر LCD.

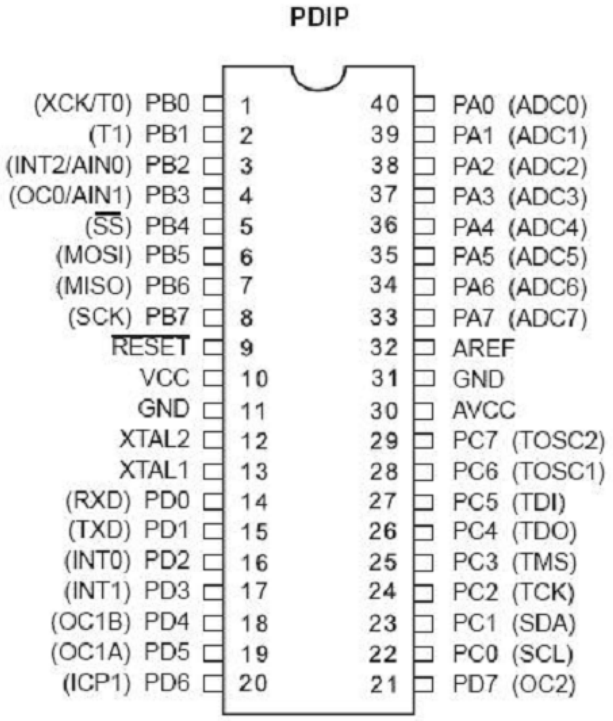
۱۱)پورت USART برای اتصال برد به کامپیوتر و دیگر بردها.

۱۲)پین هدر متصل به ولتاژ 5v.

۱۳)پین هدر متصل به زمین مدار.

۱۴)بخش تغذیه برد شامل ورودی تغذیه از طریق آداپتور، ورودی تغذیه از طریق منبع تغذیه آزمایشگاه، کلید قطع و وصل تغذیه برد، رگولاتور 5v و LED نمایشگر وضعیت تغذیه برد.

در ادامه آی سی ATMEGA و پایه های آن مشاهده می شود.



جدول ۱: پایه های مربوط به پورت A

Port Pin	Alternate Function
PA7	ADC7 (ADC input channel 7)
PA6	ADC7 (ADC input channel 6)
PA5	ADC7 (ADC input channel 5)
PA4	ADC7 (ADC input channel 4)
PA3	ADC7 (ADC input channel 3)
PA2	ADC7 (ADC input channel 2)
PA1	ADC7 (ADC input channel 1)
PA0	ADC7 (ADC input channel 0)

جدول ۲: پایه های مربوط به پورت B

Port Pin	Alternate Functions
PB7	SCK (SPI Bus Serial Clock)
PB6	MISO (SPI Bus Master Input/Slave Output)
PB5	MOSI (SPI Bus Master Output/Slave Input)
PB4	SS (SPI Slave Select Input)
PB3	AIN1 (Analog Comparator Negative Input)
PB2	OC0 (Timer/Counter0 Output Compare Match Output) INT2 (External Interrupt 2 Input)
PB1	T1 (Timer/Counter1 External Counter Input)
PB0	T0 (Timer/Counter0 External Counter Input) XCK (USART External Clock Input/Output)

جدول ۳: پایه های مربوط به پورت C

Port Pin	Alternate Function
PC7	TOSC2 (Timer Oscillator Pin 2)
PC6	TOSC1 (Timer Oscillator Pin 1)
PC5	TDI (JTAG Test Data In)
PC4	TDO (JTAG Test Data Out)
PC3	TMS (JTAG Test Mode Select)
PC2	TCK (JTAG Test Clock)
PC1	SDA (Two-wire Serial Bus Data Input/Output Line)
PC0	SCL (Two-wire Serial Bus Clock Line)

جدول ۴: پایه های مربوط به پورت D

Port Pin	Alternate Function
PD7	OC2 (Timer/Counter2 Output Compare Match Output)
PD6	ICP (Timer/Counter1 Input Capture Pin)
PD5	OC1A (Timer/Counter1 Output Compare A Match Output)
PD4	OC1B (Timer/Counter1 Output Compare B Match Output)
PD3	INT1 (External Interrupt 1 Input)
PD2	INT0 (External Interrupt 0 Input)
PD1	TXD (USART Output Pin)
PD0	RXD (USART Input Pin)

آزمایش ۱: ارتباط با پورتها

۱–۱ مقدمه

در این آزمایش هدف، آشنایی با پورتهای میکروکنترلر ATMEGA32 و راه‌اندازی LED و نمایشگرهای Seven-Segment می‌باشد.

تمامی میکروکنترلرها برای ارتباط با محیط بیرون و سیستمهای جانبی، از بلوک‌های ورودی و خروجی که با نام پورت(Port) شناخته می‌شوند، استفاده می‌نمایند. تراشه ATMEGA32 دارای ۴ پورت با نام‌های A، B، C و D می‌باشد. هرکدام از این پورتها ۸ بیتی بوده و لذا مجموعا ۳۲پین از پین‌های تراشه به این پورتها اختصاص یافته است.

هرکدام از این پورتهای چهارگانه دارای سه رجیستر اختصاصی با نامهای PORTX، PINX و DDRX می‌باشند. از آنجاییکه هریک از پورتها ۸ بیت طول دارند، لذا هرکدام از این رجیسترها نیز ۸ بیتی می‌باشند.

۱) رجیستر DDRX برای تعیین نوع ورودی یا خروجی بودن پورت استفاده می‌شود. در پارامتر "DDRX" حرف X به نام پورت اشاره می‌کند و می‌تواند یکی از حروف A، B، C یا D باشد. بیتهای رجیستر مربوط به پایه متناظر از پورت مربوطه می‌باشند. با صفر قرار دادن هر بیت، پین متناظر از نوع ورودی تعریف شده و با یک قرار دادن آن، از نوع خروجی تعریف خواهد شد.
۲) رجیستر PORTX برای مقدار دهی پینها در زمانیکه از نوع خروجی باشند استفاده می‌شود. در حالتی‌که یک پین توسط رجیستر DDRX از نوع ورودی تعریف شده باشد، رجیستر PORTX برای فعالسازی مقاومت Pull-Up آن پین می‌تواند استفاده شود(یک بودن بیت PORTX مقاومت را فعال نموده و صفر بودن، تأثیری در پین ندارد).
۳) رجیستر PINX برای خواندن مقدار داده موجود در پین استفاده می‌شود. زمانیکه یک پین از نوع ورودی تعریف شده باشد، داده ورودی در رجیستر PINX قرار خواهد داشت.

۱–۲ مراحل اجرای آزمایش

در این آزمایش تمامی پروژه ها بدون استفاده از قابلیت CodeWizard تولید خواهد شد.

۱-۲-۱ راه‌اندازی LED

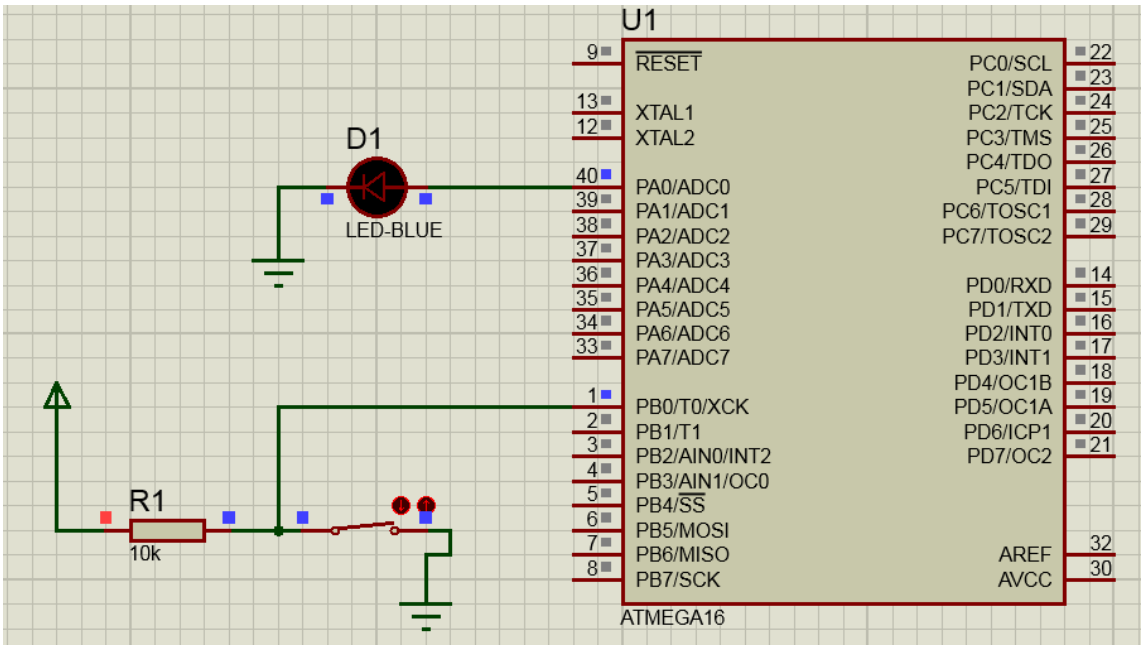
یک پروژه جدید در نرم‌افزار باز نمایید. یک source جدید باز نموده، کد زیر را در آن نوشته و به پروژه اضافه نمایید. در برنامه زیر پورت A به عنوان خروجی تعریف شده و سپس مقدار صفر برروی پینهای آن قرار داده می‌شود. در ادامه پین اول صفر شده و با تأخیرهای ۱ ثانیه برروی پورت شیفت داده می‌شود. پس از ساختن پروژه و تولید فایلهای برنامه، کد Hex را برروی برد پروگرام نمایید. خروجیهای پورت A را به LEDهای موجود برروی برد متصل نموده و نتیجه را مشاهده نمایید. خواهید دید که در هر ثانیه یکی از LEDها روشن بوده و در انتها به مدت ۱ ثانیه همه LEDها خاموش خواهند بود. این روند به صورت متناوب تکرار خواهد شد.

```
#include <mega32.h>
#include <delay.h>
void main(){
   DDRA=0xFF;
   PORTA=0x00;
   while(1){
      int i;
      PORTA.0=1;
      for (i=0; i<8; i++){
         delay_ms(1000);
         PORTA = PORTA << 1;}}}
```

۱-۲-۲ کنترل LED توسط سوییچ

در این بخش قصد داریم که با استفاده از یک سوییچ یک LED را کنترل کنیم. در این آزمایش با استفاده از پین اول پورت B، پین اول پورت A را کنترل می کنیم. یعنی با وصل کردن سوییچ روی پورت B، LED روی پورت A روشن شود و با قطع سوییچ LED خاموش شود.

مدار را همانند شکل زیر می بندیم. در پین اول پورت B، در صورتی که کلید وصل باشد، این پین به زمین وصل شده و LED خاموش می شود. اگر کلید قطع شود، پین اول پورت B به VCC وصل شده و باعث می شود که LED روشن شود.



کد برنامه به صورت زیر است:

```
#include <mega32.h>
#include <delay.h>
void main(void){
   DDRA.0=1;
   DDRB.0=0;
   PORTB.0=0;
   while(1){
            PORTA.0 = PINB.0;
            Delay_ms(100);}
```

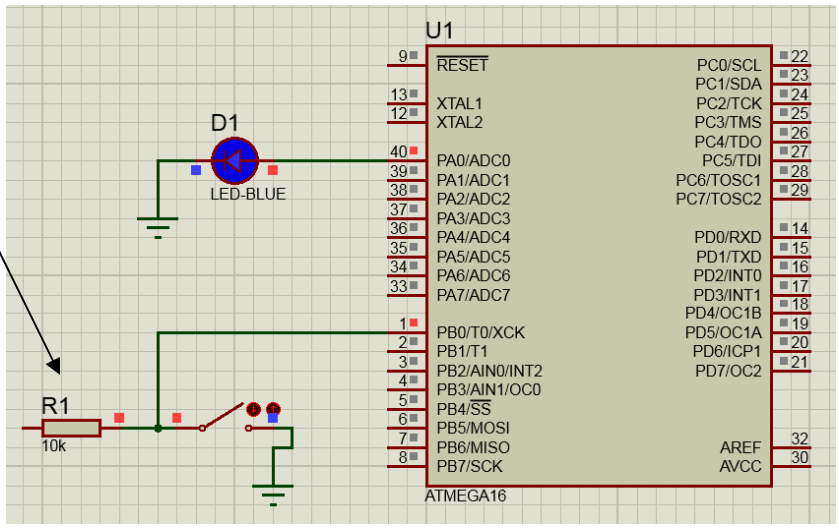
توجه:

اگر در برنامه نوشته شده، همانند کد قبل مقدار ;PORTB.0=0 باشد، تنها در صورتی که از یک Pull Up خارجی استفاده کنیم، LED بر روی پورت A روشن می شود.

ولی اگر مقدار ;PORTB.0=1 باشد، حتی بدون استفاده از Pull Up خارجی، LED بر روی پورت A روشن می شود، چون Pull up داخلی آنرا فعال کرده ایم.

به عبارت دیگر، اگر پورتی به صورت **ورودی** تعریف شود، با استفاده از رجیستر PORT آن می توان Pull Up داخلی را فعال کرد.

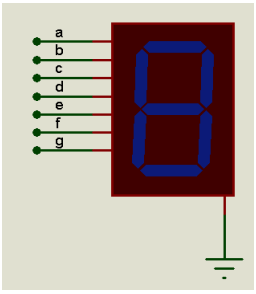
از Pull Up خارجی استفاده نشده است.



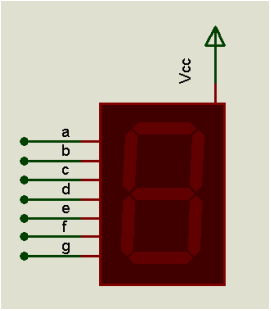
۳-۲-۱ راه‌اندازی نمایشگر Seven-Segment

در این بخش از آزمایش، به جای LED از نمایشگر Seven-Segment استفاده نوع کاتد-مشترک خواهد شد. یک پروژه جدید همانگونه که در بخش قبل توضیح داده شده، باز نموده و برنامه زیر را به پروژه اضافه نمایید.

کاتد مشترک



آند مشترک




```
#include <mega32.h>
#include <delay.h>
char Data=0;
char Seven_Segment(char Input_Data){
   char K;
   switch (Input_Data){
      case 0:  K=0x7E; return K; break;
      case 1:  K=0x0C; return K; break;
      case 2:  K=0xB6; return K; break;
      case 3:  K=0x9E; return K; break;
      case 4:  K=0xCC; return K; break;
      case 5:  K=0xDA; return K; break;
      case 6:  K=0xFA; return K; break;
```

```
      case 7:  K=0x8E; return K; break;
      case 8:  K=0xFE; return K; break;
      case 9:  K=0xDE; return K; break;
      default: K=0x00; return K; break;} }

void main(){
   DDRB=0xFF;
   PORTB=0x00;
   while(1){
      PORTB=Seven_Segment(Data);
      delay_ms(1000);
      if (Data == 9)  Data=0;
      else Data++;}}
```

پس از اجرای برنامه و تولید فایلهای نهایی، فایل Hex را برروی برد آزمایشگاه پروگرام نموده و نتایج را مشاهده نمایید. خواهید دید که اعداد ۰ تا ۹ با تأخیر ۱ ثانیه برروی نمایشگر نشان داده می‌شوند.

توجه:

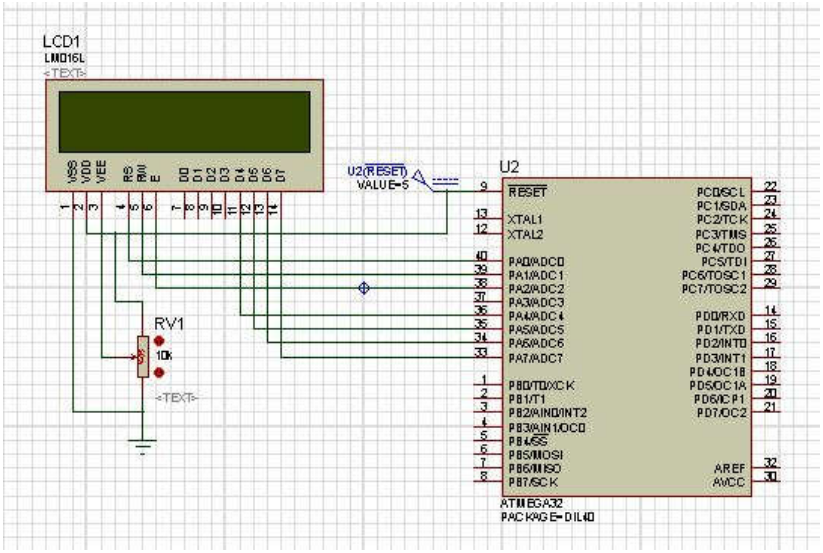
برنامه سون سگمنت را می توان به استفاده از آرایه نیز نوشت.

```
#include <mega16.h>
#include <delay.h>
char seg[10]={0x7e,0x0c,0xb6,0x9e,0x66,0x6d,0x7d,0x07,0x7f,0x6f};
int i= 0;


void main(){
    DDRA = 0xff;

    while(1){
       PORTA = seg[i];
       delay_ms(1000);
       if (i==9) i=0;
       else i++;
        }
    }
```

آزمایش ۲: نمایشگر LCD

۲–۱ مقدمه

هدف از این آزمایش راه‌اندازی نمایشگر متنی می‌باشد. نمایشگر استفاده شده یک LCD 2×16 بوده که پیکربندی اتصال آن به میکرو در شکل ۲-۱ نشان داده شده است. در شکل ۲-۱ از پایه‌های پورت A میکرو برای راه‌اندازی نمایشگر استفاده شده است که می‌تواند در برنامه نوشته شده توسط هر گروه تغییر داده شود. نمایشگرهای 16*2 دارای ۱۶ پایه می‌باشند. برروی برد آزمایشگاه، یک سوئیچ کنار نمایشگر قرار داده شده است که برای راه‌اندازی لامپ پس‌زمینه استفاده می‌شود. لامپ پس‌زمینه در تعدادی از نمایشگرهای ۲*۱۶ نیاز بوده که در صورت لزوم می‌توانید سوئیچ را روشن نمایید.



شکل ۲-۱: شماتیک مدار راه‌انداز نمایشگر متنی.

برای کار با LCD نیاز به فراخوانی کتابخانه مربوطه و استفاده از دستورات موجود می‌باشد. دستورات کار با LCD در ادامه در چند آزمایش مختلف بررسی خواهند شد.

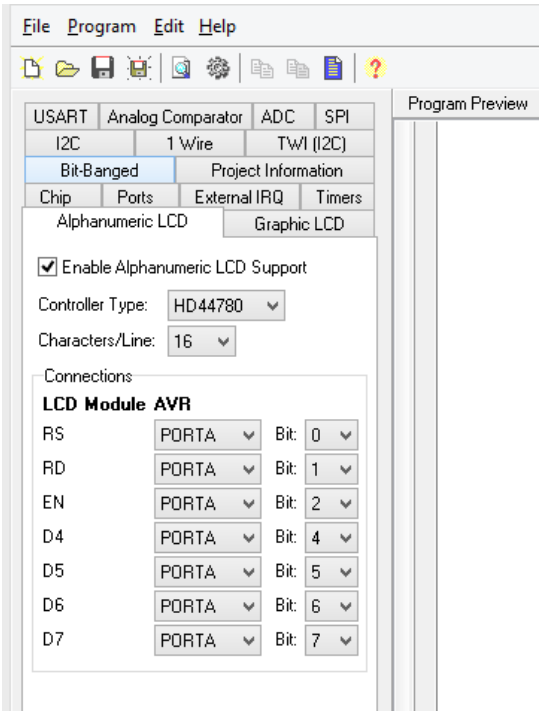
۲–۲ مراحل اجرای آزمایش

در ادامه متداولترین دستورات کار با LCD بررسی خواهند شد.

۱-۲-۲ چاپ تک کارکتر

این دستور یک کاراکتر را برروی LCD چاپ می‌نماید. دفت شود که داده ورودی به این دستور باید از نوع کارکتر بوده و موقع چاپ کد اسکی آن چاپ خواهد شد. یک پروژه جدید بدون استفاده از CodeWizard در نرم‌افزار ایجاد نموده و برنامه نوشته شده در زیر را به آن اضافه نمایید. سپس در منوی project به بخش Configure رفته و مطابق با شکل ۲-۲ تنظیمات مربوط به اتصالات LCD به میکرو انجام دهید.

```
#include <mega32.h>
#include <delay.h>
#include <alcd.h>
char Data;
void main(){
   lcd_init(16);
   lcd_clear();
   while(1){
      lcd_gotoxy(0,0);
      lcd_putchar(Data+48);
      if (Data == 9) Data = 0;
      else Data++;
      delay_ms(1000); }}
```



شکل ۲-۲: تنظیمات LCD با استفاده از CodeWizard.

پس از اجرای برنامه، فایل Hex نهایی را برروی برد آزمایشگاه پروگرام نمایید. دیده خواهد شد که اعداد ۰ تا ۹ برروی نمایشگر چاپ می‌شوند.

۲-۲-۲ چاپ داده با استفاده از دستورات تبدیل

مطابق مرحله قبل یک پروژه جدید ایجاد نموده و برنامه زیر را به آن اضافه نمایید. در این برنامه از دستور sprint برای تبدیل رشته، و از دستور lcd_puts برای چاپ رشته تولید شده استفاده شده است. دستور sprintf داده ورودی را بر اساس فرمت انتخابی تبدیل به یک رشته نموده و در آرایه کارکتری ذخیره می‌نماید. این آرایه در SRAM میکرو ذخیره شده و توسط دستور lcd_puts می‌تواند برروی LCD چاپ شود.

```
#include <mega32.h>
#include <delay.h>
#include <alcd.h>
#include <stdio.h>
char Data;
char Buf[32];
void main(){
   lcd_init(16);
   lcd_clear();
   lcd_puts("Start");
   delay_ms(5000);
   lcd_clear();
   while(1){
      sprintf(Buf,"%d",Data);
      lcd_puts("Counting");
      lcd_gotoxy(0,1);
      lcd_puts(Buf);
      Data++;
      delay_ms(1000);
      lcd_clear();}}
```

فایل Hex نهایی را برروی برد آزمایشگاه پروگرام نموده و نتایج را مشاهده نمایید. دیده خواهد شد که در خط دوم LCD، داده به فرم دسیمال چاپ می‌شود.

علاوه بر دستور lcd_puts دستور lcd_putsf نیز وجود دارد که رشته ذخیره شده در حافظه FLASH میکرو را برروی LCD چاپ می‌نماید.

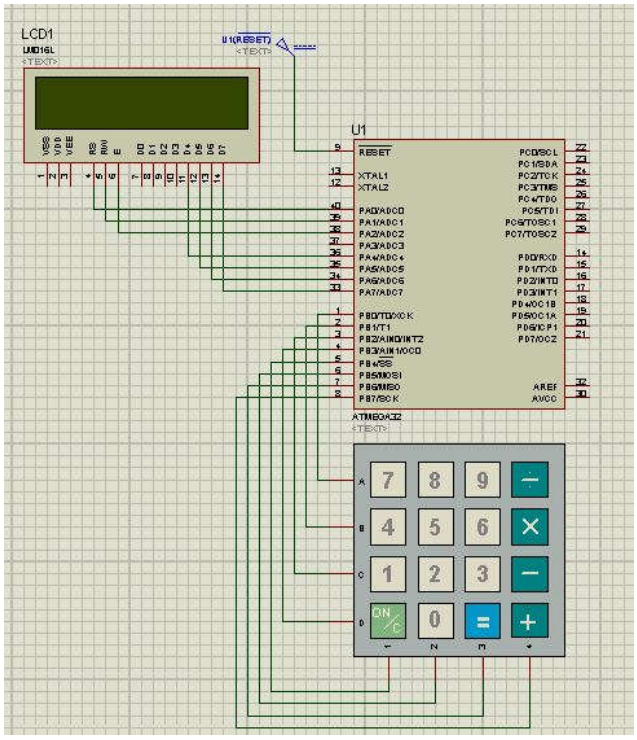
آزمایش ۳: اتصال صفحه کلید به میکروکنترلر

۳–۱ مقدمه

در این آزمایش با نحوه اتصال صفحه کلید به میکروکنترلر و خواندن داده از صفحه کلید توسط میکروکنترلر آشنا خواهید شد. صفحه کلید موجود در آزمایشگاه از نوع صفحه کلید ماتریسی 4×4 می‌باشد. به ازای هر کلید در صفحه کلید درواقع یک سوئیچ وجود دارد که با فشرده شدن سوئیچ، سطر و ستون مربوط به آن کلید، به هم متصل می‌شوند. در ادامه ابتدا یک برنامه ساده برای استفاده از صفحه کلید ارائه خواهد شد و سپس با استفاده از صفحه کلید برنامه یک ماشین حساب نوشته خواهد شد.

۳–۲ مراحل اجرای آزمایش

در این بخش اتصال صفحه کلید به میکروکنترلر و نمایش داده دریافتی بر روی LCD بررسی خواهد شد. نحوه اتصال میکرو به صفحه کلید و LCD در شکل ۳-۱ نشان داده شده است:



شکل ۳-۱: اتصالات میکروکنترلر به صفحه کلید و LCD.

۳-۲-۱ دریافت داده از صفحه کلید

یک پروژه جدید در نرم‌افزار ایجاد نموده، سپس یک Source جدید ساخته و برنامه زیر را در آن کپی نمایید. در برنامه زیر از پورت B میکروکنترلر برای ارتباط با صفحه کلید استفاده شده است. در کد زیر ۴ پین کم ارزش به سطرها ۴ پین پرارزش به ستونهای صفحه کلید متصل خواهند شد. برنامه را اجرا نموده و فایل Hex تولید شده را برروی برد آزمایشگاه پروگرام نمایید. صفحه کلید را مطابق موارد گفته شده به برد متصل نموده و برای اتصال LCD از پورت A استفاده نمایید. پس از پروگرام کردن برنامه خواهید دید که با فشرده شدن هر کلید، کارکتر مربوطه برروی LCD نمایش داده خواهد شد.

```
#include <mega32.h>
#include <alcd.h>
#include <delay.h>
unsigned char scan[4]={0XFE,0XFD,0XFB,0XF7};
char Key;
char arrkey[16]={
   '7','8','9','/',
   '4','5','6','*',
   '1','2','3','-',
   'C','0','=','+'};
#define c1 PINB.4
#define c2 PINB.5
```

```
#define c3 PINB.6
#define c4 PINB.7
#define keypad_port PORTB
char keypad(){
   unsigned char r,c,k;
   DDRB=0X0F;
   keypad_port=0XFF;
   while(1){
      for (r=0; r<4; r++){
      c=255;
      keypad_port=scan[r];
      delay_us(10);
      if(c1==0) c=0; if(c2==0) c=1; if(c3==0) c=2; if(c4==0) c=3;
      if (c!=255){
      k=arrkey[(r*4)+c];
      while(c1==0); while(c2==0); while(c3==0); while(c4==0);
      delay_ms(50);
      return k;
}}}}
void main(void){
   lcd_init(16);
   lcd_clear();
   lcd_puts("Start");
   delay_ms(1000);
   lcd_clear();
   while (1){
     Key=keypad();
     if (Key == 'C') lcd_clear();
     else lcd_putchar(Key);
     }}
```

۲-۲-۳ماشین حساب با استفاده از میکروکنترلر

همانند مرحله قبل یک پروژه جدید ساخته و برنامه زیر را در آن استفاده نمایید. برنامه زیر مربوط به یک ماشین حساب بوده که عدد و عملیات مربوطه را از یک صفحه کلید دریافت می‌نماید. با فشرده شدن کلید = نتیجه عملیات برروی LCD نمایش داده می‌شود و با فشرده شدن کلید on/c صفحه و حافظه پاک می‌شود. برنامه را اجرا نموده و برروی برد آزمایشگاه پروگرام نمایید. برای اتصال LCD از پورت A و برای صفحه کلید مطابق مرحله قبلی از پورت B استفاده نمایید.

```
#include <mega32.h>
#include <alcd.h>
#include <delay.h>
#include <stdio.h>
unsigned char scan[4]={0XFE,0XFD,0XFB,0XF7};
char Key, State=0, Operand;
char Buf1[5], Buf0[5],Buf[16];
int Num0=0, Num1=0, Result;
char arrkey[16]={
   '7','8','9','/',
   '4','5','6','*',
```

```
    '1','2','3','-',
    'C','0','=','+'};
#define c1 PINB.4
#define c2 PINB.5
#define c3 PINB.6
#define c4 PINB.7
#define keypad_port PORTB
char keypad(){
   unsigned char r,c,k;
   DDRB=0X0F;
   keypad_port=0XFF;
   while(1){
      for (r=0; r<4; r++){
      c=255;
      keypad_port=scan[r];
      delay_us(10);
      if(c1==0) c=0; if(c2==0) c=1; if(c3==0) c=2; if(c4==0) c=3;
      if (c!=255){
      k=arrkey[(r*4)+c];
      while(c1==0); while(c2==0); while(c3==0); while(c4==0);
      delay_ms(50);
      return k;
}}}}
void main(void){
   lcd_init(16);
   lcd_clear();
   lcd_puts("Start");
   delay_ms(1000);
   lcd_clear();
   while (1){
     Key=keypad();
     switch (State) {
       case 0:
          if (Key == 'C') {
             Num0=0; Num1=0; Operand=''; lcd_clear(); lcd_putchar('0');}
          else if ((Key == '+') | (Key == '-') | (Key == '/') | (Key == '*')) {
             Operand = Key; lcd_clear(); lcd_puts(Buf1);
             lcd_putchar(' '); lcd_putchar(Key); lcd_putchar(' '); State =1;}
          else {
             Num1 = Num1 * 10 + Key - 48; lcd_clear();
             sprintf(Buf1,"%d",Num1); lcd_puts(Buf1);}
          break;
       case 1:
          if (Key == 'C') {
             Num0=0; Num1=0; Operand=''; lcd_clear(); lcd_putchar('0'); State = 0;}
          else if ((Key == '1') | (Key == '2') | (Key == '3') | (Key == '4')
                | (Key == '5') | (Key == '6') | (Key == '7') | (Key == '8')
                | (Key == '9') | (Key == '0')) {
             Num0 = Num0 * 10 + Key - 48; lcd_gotoxy(0,0); lcd_puts(Buf1);
             lcd_putchar(' '); lcd_putchar(Operand); lcd_putchar(' ');
             sprintf(Buf0,"%d",Num0); lcd_puts(Buf0);}
          else if (Key == '='){
             lcd_putchar(' '); lcd_putchar(Key); lcd_gotoxy(0,1);
             if (Operand == '+') {
                Result = Num1 + Num0; sprintf(Buf,"%d",Result); lcd_puts(Buf);}
             else if (Operand == '-') {
```

```
            Result = Num1 - Num0; sprintf(Buf,"%d",Result); lcd_puts(Buf);}
          else if (Operand == '/') {
            if (Num0 == 0) {Result = 0; lcd_puts("Nan");}
            else {
               Result = Num1 / Num0; sprintf(Buf,"%d",Result); lcd_puts(Buf);}}
          else if (Operand == '*') {
            Result = Num1 * Num0 ; sprintf(Buf,"%d",Result); lcd_puts(Buf);}
          State = 0; Num0 = 0; Num1 = Result; sprintf(Buf1,"%d",Num1);
          }
       break;}
}}
```

آزمایش ۴: کار با تایمر صفر میکروکنترلر

۴-۱ مقدمه

در این آزمایش هدف راه‌اندازی تایمر میکروکنترلر AVR می‌باشد. با استفاده از تایمرهای AVR می‌توان تأخیرهای دقیق را تولید نمود. یکی از مزیتهای استفاده از تایمر در میکروکنترلرها این است که با استفاده از تایمرها یک سری عملیات در زمانهای مشخص می‌تواند بدون اینکه میکرو درگیر انجام آن شود، صورت پذیرد.

تایمر صفر در میروکنترلرهای AVR یک شمارنده ۸ بیتی می‌باشد که قابلیت بالاشمار و پایین شمار داشته و در ۴ مود کاری مختلف می‌تواند عمل نماید. این تایمر دارای دو پرچم وقفه بوده که درصورت فعال بودن وقفه‌ها، با ۱ شدن پرچم وقفه زیربرنامه وقفه به طور خودکار اجرا خواهد شد. در این آزمایش هدف استفاده از این تایمر در مودهای کاری مختلف می‌باشد.

تایمر صفر دارای سه رجیستر ۸ بیتی TCNT0، OCR0 و TCCR0 می‌باشد. برای تنظیمات مودهای کاری تایمر از رجیستر TCCR0 استفاده می‌شود. رجیستر TCNT0 مقدار لحظه‌ای تایمر را دارا بوده و با هر بار شمارش عدد داخل این رجیستر تغییر می‌کند. رجیستر OCR0 برای مقایسه با مقدار TCNT0 استفاده شده و در زمانیکه مقدار این دو رجیستر برابر باشد یا به عبارتی نتیجه مقایسه صحیح باشد در این صورت پرچم مربوط به وقفه مقایسه ۱ خواهد شد. در صورتیکه مقدار شمارش تایمر نیز به مقدار نهایی رسیده باشد، پرچم مربوط به وقفه سرریز برابر ۱ خواهد شد.

۴-۲ مراحل اجرای آزمایش

۴-۲-۱ تایمر صفر در مود Fast PWM

همانگونه که از نام این مود مشخص است، تایمر صفر در این مود برای تولید پالس PWM مورد استفاده قرار می‌گیرد. پالسهای PWM یکی از موارد بسیار پرکاربرد در صنعت به ویژه صنایع الکترونیک قدرت می‌باشند که کنترل دور موتورهای مختلف نمونه بارزی از آن می‌باشد. در این مود تایمر صفر برروی پایه OC0 که همان

پین شماره ۳ پورت B می‌باشد یک موج PWM تولید خواهد کرد. فرکانس خروجی تایمر براساس رابطه زیر مشخص می‌شود:

معادله (۴-۱)

$$F_{out} = \frac{F_{Clk}}{N \times 256}$$

در رابطه فوق F_{Clk} فرکانس میکروکنترلر بوده و N ضریب Prescaler تایمر صفر می‌باشد. برای تعیین مقدار عرض پالس نیز از رجیستر OCR0 استفاده می‌شود.

یک پروژه جدید در نرم‌افزار ایجاد نموده و برنامه زیر را به آن اضافه نمایید. در برنامه زیر تایمر صفر در مود Fast PWM و حالت Non-Inverting استفاده شده است. با تنظیم رجیسترهای مربوطه، یک پالس مربعی با عرض پالس 50% و فرکانس 500Hz درخروجی OC0 تولید خواهد شد. برنامه را اجرا نموده و برروی برد آزمایشگاه پروگرام نمایید. سپس پالس خروجی را برروی اسیلوسکوپ مشاهده نمایید.

```
#include <mega32.h>
void main(){
   TCCR0=0x6A;//Timer0 in Non-Inverting FAST PWM mode, Prescaler 8
   OCR0=0x7F;
   DDRB.3=1;// PORTB.3 as Output
   while(1){
        }}
```

۴-۲-۲ تایمر صفر در مود Phase Correct PWM

در این مود نیز همانند مود Fast PWM، تایمر صفر برای تولید پالس PWM مورد استفاده قرار می‌گیرد. در این مود تایمر هم به صورت بالاشمار و هم به صورت پایین‌شمار عمل کرده و عرض پالس موج خروجی قابلیت تنظیم بیشتری خواهد داشت. فرکانس خروجی تایمر براساس رابطه زیر مشخص می‌شود:

معادله (۴-۲)

$$F_{out} = \frac{F_{Clk}}{N \times 510}$$

در رابطه فوق F_{Clk} فرکانس میکروکنترلر بوده و N ضریب Prescaler تایمر صفر می‌باشد. برای تعیین مقدار عرض پالس نیز از رجیستر OCR0 استفاده می‌شود.

یک پروژه جدید در نرم‌افزار ایجاد نموده و برنامه زیر را به آن اضافه نمایید. در برنامه زیر تایمر صفر در مود Phase Correct PWM و حالت Non-Inverting استفاده شده است. با تنظیم رجیسترهای مربوطه، یک پالس

مربعی با عرض پالس 50% درخروجی OC0 تولید خواهد شد. برنامه را اجرا نموده و بر روی برد آزمایشگاه پروگرام نمایید. سپس پالس خروجی را بر روی اسیلوسکوپ مشاهده نموده و فرکانس خروجی را از روی پالس خروجی تعیین نمایید.

```
#include <mega32.h>
void main(){
   TCCR0=0x62;//Timer0 in Non-Inverting Phase Correct PWM mode, Prescaler 8
   OCR0=0x7F;
   DDRB.3=1;// PORTB.3 as Output
   while(1){
         }}
```

۳-۲-۴فرکانس متر با استفاده از تایمر صفر

در مود نرمال، تایمر صفر می‌تواند به عنوان شمارنده کلاک استفاده شود. شمارش تعداد کلاک در زمان مشخص، یکی از روشهای اندازه‌گیری فرکانس می‌باشد. یک پروژه جدید ایجاد نموده و برنامه زیر را به پروژه اضافه نمایید.

```
#include <mega32.h>
#include <alcd.h>
#include <delay.h>
#include <stdio.h>
char Buf[16];
unsigned char Counter=0;
unsigned int Data;
interrupt [TIM0_OVF] void timer0_ovf_isr(void) {
   Counter++;}
void main(){
   TCCR0=0x07;//Timer0 in Normal mode, External Clock
   TIMSK=0x01;
   DDRB.3=1;// PORTB.3 as Output
   lcd_init(16);
   lcd_clear();
   lcd_puts("Start");
   #asm("sei")
   while(1){
         TCCR0=0x00;
         TCNT0=0x00;
         Counter = 0;
         TIFR |= 0x01;
         TCCR0=0x07;
         delay_ms(1000);
         TCCR0=0x00;
         Data = (Counter * 256) + TCNT0;
         sprintf(Buf,"Freq = %d",Data);
         lcd_clear();
         lcd_puts(Buf);
         }}
```

در برنامه فوق تایمر صفر در مود نرمال قرار داده شده و کلاک تایمر از پایه T0 دریافت می‌شود. وقفه تایمر صفر فعال شده و با سرریز شدن تایمر به تعداد شمارش یک واحد افزوده می‌شود. تایمر صفر به مدت یک ثانیه فعال بوده و در انتهای یک ثانیه، تایمر صفر خاموش شده و با استفاده از مقدار تایمر صفر و تعداد دفعات سرریز تایمر، فرکانس پالس ورودی محاسبه می‌شود.

آزمایش ۵:مبدل آنالوگ به دیجیتال (ADC)

۵–۱ مقدمه

در این آزمایش هدف راه‌اندازی مبدل آنالوگ به دیجیتال میکروکنترلر AVR می‌باشد. میکروکنترلر ATMEGA32 دارای ۸ کانال ورودی آنالوگ برای مبدل آنالوگ به دیجیتال می‌باشد که برروی پورت A قرار دارند. مبدل موجود در این تراشه دارای رزولوشن ۱۰ بیت بوده و لذا داده خروجی مبدل عددی بین ۰ تا ۱۰۲۳ خواهد بود. برای محاسبه ولتاژ نمونه‌برداری شده می‌توانید از معادله (۵-۱) استفاده نمایید:

معادله (۵-۱)

$$V_{in} = \frac{Data \times V_{ref}}{1023} \approx \frac{Data \times V_{ref}}{1024}$$

در رابطه فوق Data مقدار باینری خروجی مبدل آنالوگ به دیجیتال می‌باشد و V_{ref} ولتاژ مرجع مبدل می‌باشد که در تنظیمات مشخص می‌شود.

۵–۲ مراحل اجرای آزمایش

۵–۲–۱ نمونه‌برداری سیگنال آنالوگ با ADC

یک پروژه جدید در نرم‌افزار ایجاد نموده و برنامه زیر را به آن اضافه نمایید. توجه شود که از پورت B برای LCD استفاده شده و فرکانس کاری میکرو نیز 1MHz می‌باشد.

```
#include <mega32.h>
#include <delay.h>
#include <alcd.h>
#include <stdio.h>
#define ADC_VREF_TYPE 0x00
int Data=0,Data1=0;
char Buf[16];
float Din;
unsigned int read_adc(unsigned char adc_input) {
   ADMUX=adc_input | (ADC_VREF_TYPE & 0xff);
   delay_us(10);
   ADCSRA|=0x40;
   while ((ADCSRA & 0x10)==0);
   ADCSRA|=0x10;
   return ADCW;}
void main(void) {
   PORTB=0x00;
```

```
DDRB=0xFF;
ADMUX=ADC_VREF_TYPE & 0xff;
ADCSRA=0x83;
lcd_init(16);
lcd_clear();
lcd_puts("No Data");
delay_ms(2000);
while (1){
   Data = read_adc(0);
   if (Data != Data1) {
      Data1 = Data;
      lcd_clear();
      sprintf(Buf,"Data = %d",Data);
      lcd_puts(Buf);
      lcd_gotoxy(0,1);
      Din = Data;
      Din = (Din*5)/1023;
      sprintf(Buf,"Voltage = %.2f",Din);
      lcd_puts(Buf);}
   delay_ms(1000);}}
```

در تنظیمات فوق ولتاژ مرجع ADC از پایه Vref دریافت شده و چون برابر 5v می‌باشد لذا در محاسبات از عدد 5 برای محاسبه ولتاژ آنالوگ ورودی استفاده شده است.

آزمایش ۶:ارتباط سریال USART

۶–۱ مقدمه

میکروهای AVR پروتکلهای ارتباط سریال مختلفی را پشتیبانی می‌کنند. هدف از این آزمایش آشنایی با ارتباط سریال USART و استفاده از آن می‌باشد. در این پروتکل ارتباطی، دو میکروکنترلر یا میکروکنترلر و PC می‌توانند تنها با استفاده از ۳ خط ارتباطی به تبادل داده بپردازند. یکی از این خطوط برای ارسال، خط دوم برای دریافت و خط بعدی زمین مشترک می‌باشد.

۶–۲ مراحل اجرای آزمایش

۶-۲-۱ ارتباط میکروکنترلر و PC با استفاده از USART

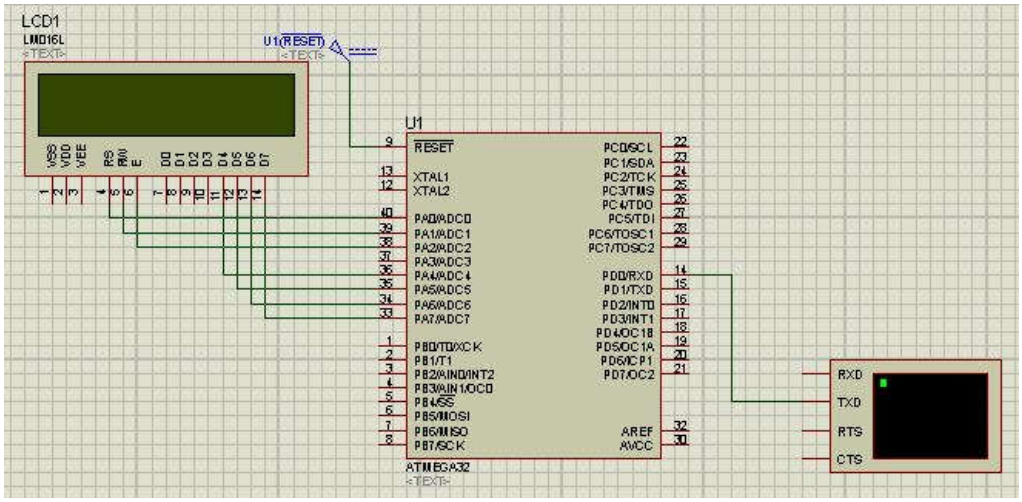
در این بخش از آزمایش، داده توسط کامپیوتر به میکروکنترلر ارسال شده و سپس بر روی LCD نمایش داده خواهد شد.

یک پروژه جدید در نرم‌افزار ایجاد نموده و برنامه زیر را به آن اضافه نمایید. فرکانس کاری میکروکنترلر را 1MHz تنظیم نموده و از پورت A برای LCD استفاده نمایید.

```
#include <mega32.h>
#include <delay.h>
#include <alcd.h>
#include <stdio.h>
char Buf[16];
char Din,Count=0;
void main(void) {
   UCSRA=0x00;
   UCSRB=0x10;
   UCSRC=0xA6;
   UBRRH=0x00;
   UBRRL=0x0C;
   lcd_init(16);
   lcd_clear();
   lcd_puts("No Data");
   delay_ms(2000);
   while (1){
      Din = getchar();
      lcd_clear();
      lcd_puts("Input = ");
      lcd_putchar(Din);
      lcd_gotoxy(0,1);
      sprintf(Buf,"Count = %d",Count);
```

```
lcd_puts(Buf);
Count++;}}
```

در برنامه فوق تنظیمات USART برای نرخ ارسال(Baud Rate) ۴۸۰۰، طول داده ۸ بیت، پریتی از نوع زوج و تعداد بیت Stop برابر یک قرار داده شده است. در شبیه‌سازی از بلوک Virtual Terminal استفاده نموده و TXD بلوک Virtual Terminal را به RXD میکروکنترلر متصل نمایید. سپس برنامه را اجرا نموده و بر روی برد آزمایشگاه پروگرام نمایید. در تست عملی از مبدل Usart به USB که در آزمایشگاه موجود است استفاده نمایید.



شکل ۶-۱: نحوه اتصال Virtual Terminal به میکروکنترلر در محیط Proteus.