به نام خدا

دانشکده مهندسی برق

گزارش پروژه اختیاری

درس طراحی سیستم های میکروپروسسوری

حامد رستاقی 97101701

محمدامین حاجی خداوردیان 97101518

استاد:دکتر خسرو حاج صادقی

نیمسال دوم 99-00

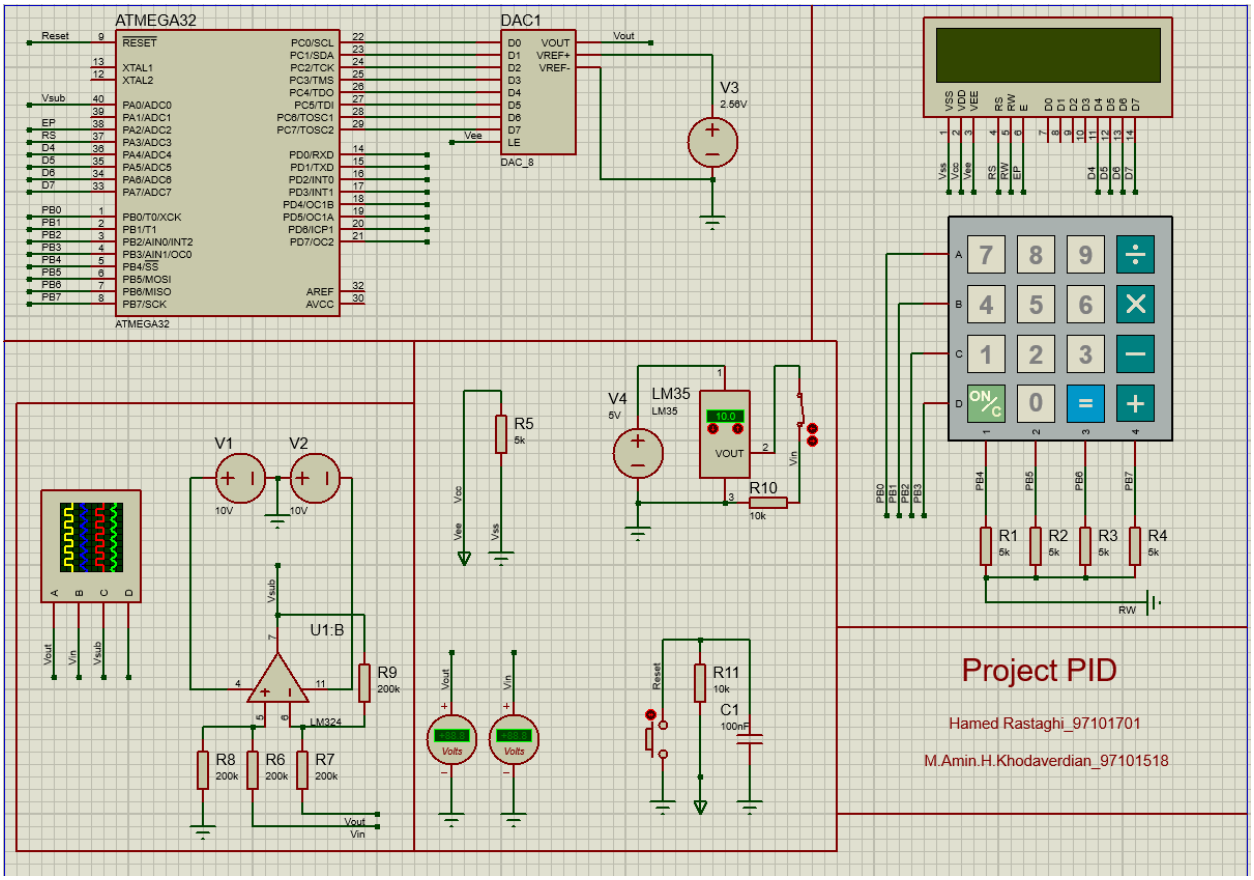
مقدمه

هدف از انجام این پروژه، ساخت یک کنترل کننده ی PID است، کنترل کننده ای که مدنظر ماست، به گونه ای طراحی میشود که 3 عدد بین 0 تا 16 را به عنوان مقادیر ثابت های K_P ، K_I و K_D از کاربر میگیریم.

همچنین کاربر قابلیت تنظیم نقطه کار مطلوب را به ازای ولتاژ های 0 تا 2.56 ولت دارد، سرعت نمونه برداری کنترل کننده PID برابر 20 میلی ثانیه است.

هدف دیگر پروژه این است که در هنگامی که کنترلر PID را استفاده نمیکنیم شمارنده ی ATMEGA32 تا 16 دقیقه شمارش انجام دهد، و پس از آن 6 بایت نوشته شده در حافظه های 0x200 تا 0x205 را به طول متوالی بر روی پورت D می فرستد.

شکل زیر شمای نهایی پروژه است که هرکدام از بخش های آن در ادامه توضیح داده خواهد شد



بخش مربوط به LCD:

یکی از محدودیت های ما در انجام این پروژه کافی نبودن تعداد پورت های ATMEGA32 است زیرا پورت های C و D برای مسائل خاصی در نظر گرفته شده است و از آن ها نمیتوانیم استفاده کنیم و محدودیت دیگر استفاده از کیبورد است.

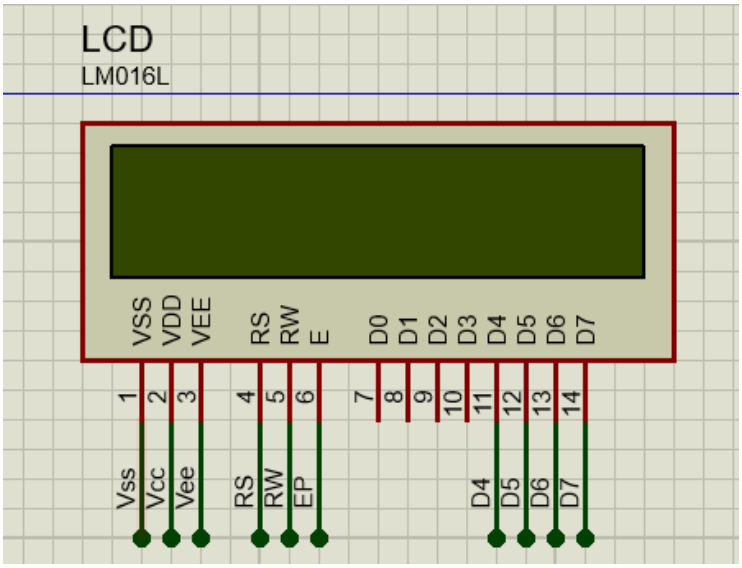
بنابراین برای رفع این محدودیت از LCD در مود 4 بیتی آن استفاده میکنیم تا بتوانیم براحتی مشکل را برطرف سازیم.

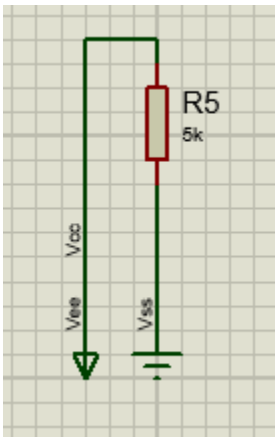
برای این کار باید LCD را با 6 پایه راه اندازی کنیم که این 6 پایه شامل D4، D5، D6 و D7 هستند که به صورت 4 بیت 4 بیت داده را از پردازنده به LCD منتقل میکنند که به ترتیب به پایه های A4 تا A7 از میکروپروسسور وصل شده اند و 2 پایه دیگر به ترتیب A3 و A2 است که برای RS و E از آن استفاده میکنیم.

چون از LCD اطلاعاتی را نمیخوایم بخوانیم (از DDRAM و یا CGRAM) بنابراین این پایه را به میکروپروسسور وصل نمیکنیم.

علت اینکه از LCDدر مود 4bit استفاده میکنیم ، این است که بتوانیم از پورت A به طور بهینه استفاده کنیم و هم از ADC آن و هم از 6بیت آن برای LCD استفاده کنیم

حال به بررسی مدار آن میپردازیم:





مدار نشان داده شده در شکل بالا مدار ساده ای برای تغذیه ی LCD است، لازم به ذکر است که در مدار بالا میتوان با تقسیم ولتاژ دو مقاومت، ولتاژ Vee که مشخص کننده ی میزان روشنایی LCD است را نیز به گونه ای بین 0 تا 5 ولت انتخاب کرد چون در ولتاژ 0 صفحه خاموش است و در 5 ولت صفحه به قدری روشن میشود که کاراکتر های نمایش داده شده قابل مشاهده نیستند بنابراین انتخاب مقداری بین آن ها منطقی است.

حال به سراغ کد های بخش LCD میرویم.

توجه: برای نوشتن کدها و تولید فایل hex از نرم افزار ATMEL STUDIO 7.0 استفاده شده است.

```
void Command( unsigned char Cmd )
{
    PORTA = (PORTA & 0x0F) | (Cmd & 0xF0);
    PORTA &= ~ (1<<PA3);
    PORTA |= (1<<PA2);
    _delay_us(1);
    PORTA &= ~ (1<<PA2);

    _delay_us(200);

    PORTA = (PORTA & 0x0F) | (Cmd << 4);
    PORTA |= (1<<PA2);
    _delay_us(1);
    PORTA &= ~ (1<<PA2);
    _delay_ms(2);
}
```

تابع بالا اولین تابعی است که آن را توضیح میدهیم. تابع Command برای اجرای دستورات مربوط به Command نوشته شده است که به صورت زیر کار میکند.

در این تابع مقدار PORTA برابر میشود با OR کردن 4 بیت MSB دستور و 4 بیت پایین OR (4 بیت LSB مربوط به PORTA را بدون تغییر نگه میداریم چون مربوط به اتصالات مربوط به RS و E هستند که در صورت نوشتن Command یا DATA تغییر میکنند.)

چون در حال فرستادن دستور هستیم باید RS را برابر 0 قرار دهیم.(پایه A3 در PORTA) سپس باید E را برابر 1 کنیم و در خط بعد مجدد آن را صفر کنیم (پایه A2 در PORTA) چون برای نوشتن داده باید یک پالس 1 به 0 تولید شود که عمل نوشتن انجام شود سپس پس از آن به اندازه 200میکرو ثانیه صبر میکنیم تا از نوشته شدن مقادیر در حافظه LCD اطمینان حاصل کنیم.

پس از آن به سراغ بخش دوم دستور می رویم (چون به صورت 4 بیتی از LCD استفاده میکنیم به همین دلیل باید آن را به صورت 2 نیبل پشت سر هم بفرستیم)این کار را با شیفت دادن دستور به سمت چپ انجام میدهیم و مجدد یک پالس 1 به 0 ایجاد میکنیم که آن نیز در حافظه LCD نوشته شود.

```
void LCDCHARACTER( unsigned char DATA )
{
    PORTA = (PORTA & 0x0F) | (DATA & 0xF0);
    PORTA |= (1<<PA3);
    PORTA|= (1<<PA2);
    _delay_us(1);
    PORTA &= ~ (1<<PA2);

    _delay_us(200);

    PORTA = (PORTA & 0x0F) | (DATA << 4);
    PORTA |= (1<<PA2);
    _delay_us(1);
    PORTA &= ~ (1<<PA2);
    _delay_ms(2);
}
```

تابع بعدی تابع LCDCHARACTER است که در بالا کد آن دیده میشود. این تابع به شدت مشابه تابع قبلی میباشد با این تفاوت که در این تابع چون قرار است داده نوشته شود باید RS برخلاف تابع Command برابر 1 قرار داده شود.

در ادامه پالس 1 به 0 در پورت E و شیفت داده داده برای فرستادن دو 4 بیت مجزا مجدد تکرار میشود.

```
void Clear()
{
    Command (0x01);
    _delay_ms(2);
    Command (0x80);
}
```

تابع بالا برای Clear کردن صفحه LCD نوشته شده است که Command 0x01 برای پاک کردن صفحه و Command 0x80 برای برگرداندن مجدد کرسر در آدرس ابتدای DDRAM یعنی 0x80 است.

```
void LCDStr (char *str)
{
    int i;
    for(i=0;str[i]!=0;i++)
    {
        LCDCHARACTER (str[i]);
    }
}
```

تابع بالا برای این است که ابتدا آدرس اول خانه یک رشته را میگیرد سپس با استفاده از یک for تا خانه آخر رشته را طی میکند و به ازای هر بایت از حافظه یکبار دستور LCDCHARACTER اجرا میشود که با این کار میتوان یک رشته از کاراکتر ها را بر روی LCD به نمایش گذاشت.

```
void LCDposition (char r, char position, char *str)
{
    if (r == 0 && position<16)
    Command((position & 0x0F)|0x80);
    else if (r == 1 && position<16)
    Command((position & 0x0F)|0xC0);
    LCDStr(str);
}
```

این تابع مشابه تابع قبل است با این تفاوت که با استفاده از این تابع میشود یک رشته را در محل دلخواه حافظه LCD نوشت، و باید مختصات شروع نوشتن را نیز در آرگومان های ورودی قرار دهیم.

برای مقدار دهی اولیه در تابع main کد زیر زده شده است:

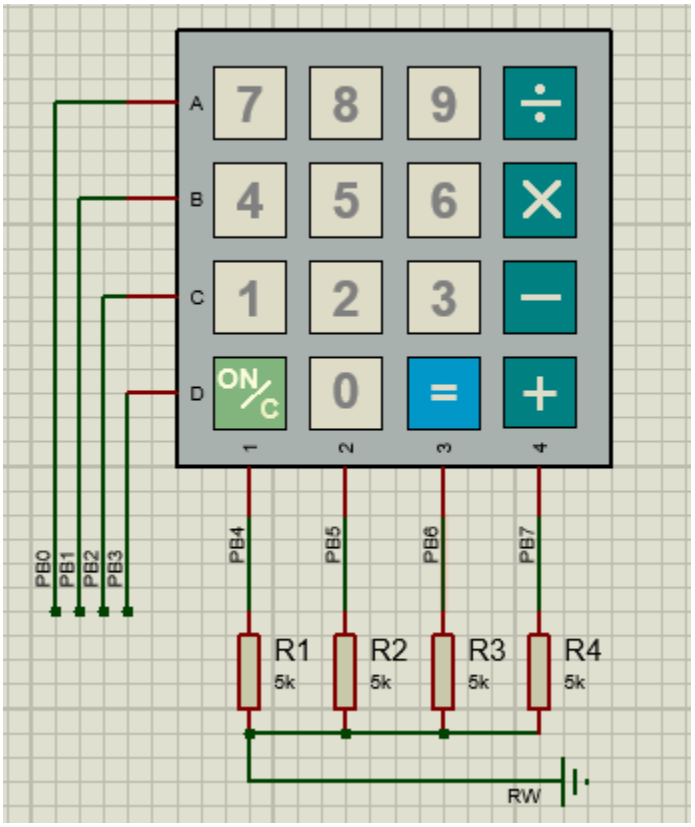
```
DDRA = 0xFC;
_delay_ms(20);
Command(0x02);
Command(0x28);
Command(0x0c);
Command(0x06);
Command(0x01);
_delay_ms(2);
Clear();
```

این تابع دستورات 0x02، 0x0C،0x28 ، 0x06 و 0x01 را برای LCD میفرستد که مشخص کننده این هستند که LCD در مود 4 بیت کار میکند و 5*7 است و همچنین کرسر چشمک زن است و پس از نوشتن هر کاراکتر کرسر به سمت راست برود و در نهایت با فرستادن دستور 0x01 پاک میشود و آماده نوشتن مقدار جدید.

بخش مربوط به کیبورد:

در این بخش با استفاده از هر 8 بیت پورت B میخواهی یک کیبورد 4 در 4 برای گرفتن ورودی از کاربر راه اندازی کنیم.

ابتدا اتصالات و سپس کد های مربوط به آن را توضیح خواهیم داد:



در بالا دیده میشود که پورت های 1 تا 4 کیبورد با یک مقاومت Pull down شده اند.

حال با استفاده از پایه های 4 تا 7 PORTB به نوبت و با سرعت بالا در هر مرحله ی پایه را 0 میکنیم و این عمل را دائما انجام میدهیم در واقع عمل scan Keypad در حال انجام شدن است که به ما کمک میکند بدانیم کدام یک از کلید ها فشرده شده است.

کدهای نوشته شده برای این بخش در پایین آورده شده است:

```
char keyboard()
{
    char key = 16;
    while(1)
    {
        PORTB = 0xF0;

        PORTB = 0b11100000;
        _delay_ms(2);
        if(!(PINB & 0b00000001))
        key = 7;
        if(!(PINB & 0b00000010))
        key = 4;
        if(!(PINB & 0b00000100))
        key = 1;
        if(!(PINB & 0b00001000))
        key = 10;
        PORTB = 0b11110000;

        PORTB = 0b11010000;
        _delay_ms(2);
        if(!(PINB & 0b00000001))
        key = 8;
        if(!(PINB & 0b00000010))
        key = 5;
        if(!(PINB & 0b00000100))
        key = 2;
        if(!(PINB & 0b00001000))
        key = 0;
        PORTB = 0b11110000;

        PORTB = 0b10110000;
        _delay_ms(2);
        if(!(PINB & 0b00000001))
        key = 9;
        if(!(PINB & 0b00000010))
        key = 6;
        if(!(PINB & 0b00000100))
        key = 3;
        if(!(PINB & 0b00001000))
        key = 11;
        PORTB = 0b11110000;

        PORTB = 0b01110000;
        _delay_ms(2);
        if(!(PINB & 0b00000001))
        key = 12;
        if(!(PINB & 0b00000010))
        key = 13;
        if(!(PINB & 0b00000100))
        key = 14;
        if(!(PINB & 0b00001000))
        key = 15;
        PORTB = 0b11110000;

        if (key == 10) main();
        if (key == 11) Timer();

        if (key != 16)
        {
            if(key < 10)
            LCDCHARACTER(48 + key);
            _delay_ms(25);
            return key;
            break;
        }
```

در هر مرحله PORTB را برابر 0xF0 قرار میدهیم سپس با توجه به ردیفی که در آن قرار گرفته ایم پورت متصل به آن ستون را برابر 0 قرار میدهیم و چک میکنیم که آیا کلیدی فشرده شده است یا نه و مقدار آن را در Key ذخیره میکنیم.

با توجه به شکل کیبورد اعداد 1 تا 9 در key مقادیر خودشان را دریافت میکنند و برای بقیه کلید ها به صورت زیر مقدار دهی شده اند:

+ = 15 - = 14 × = 13 ÷ = 12 On/c = 10

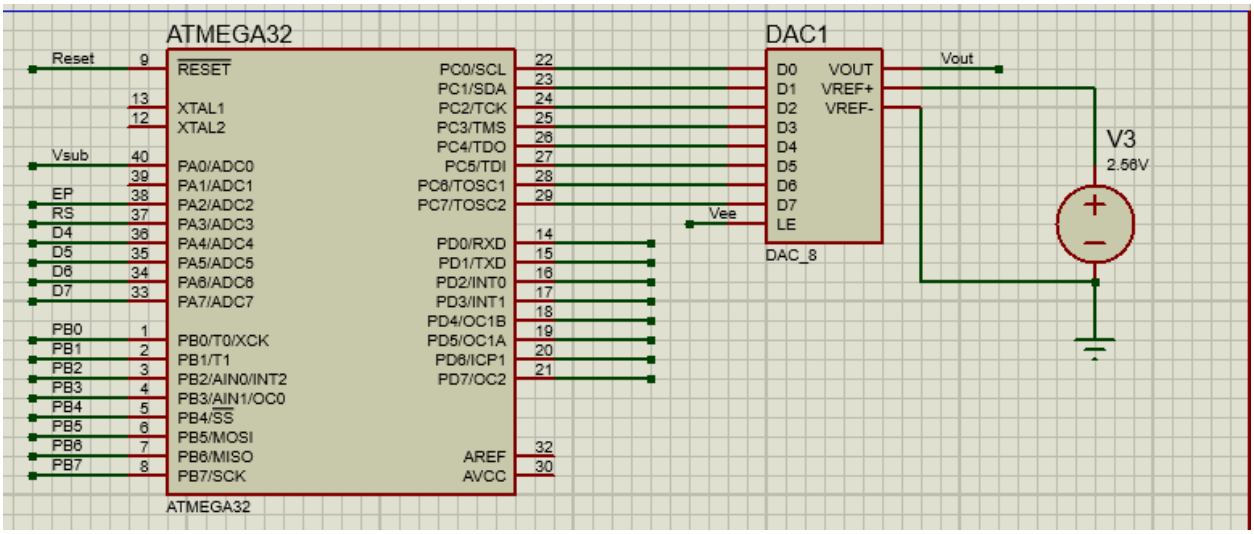
‘ = ‘ = 11

از کلید = برای رفتن از مود کنترل کننده به مود شمارنده استفاده میکنیم و از On/c برای ریست کردن کنترل کننده برای گرفتن مجدد مقادیر PID.

در صورتی که کلیدی فشرده شده باشد آن را بر روی LCD نمایش میدهیم که در انتهای کد این روابط دیده میشود که درصورتی که کلیدی فشرده شود تابع LCDCHARACTER مقدار کلید فشرده شده را با آفست کد ASCII جمع میکند و بر روی LCD نمایش میدهد.

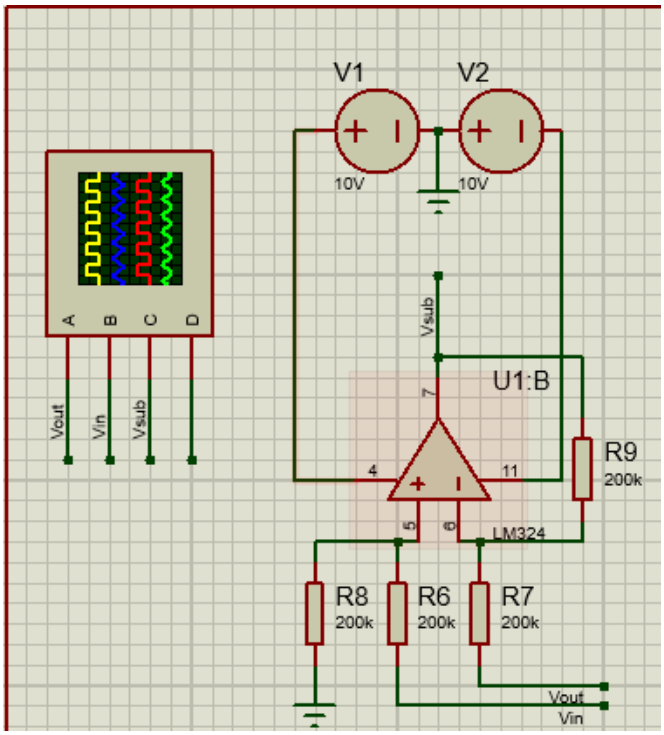
بخش مربوط به ADC:

در این پروژه ، برای دریافت نقطه ی کار از کاربر ، از یک ADC استفاده میکنیم ، همان طور که در بخش اول توضیح دادیم ، پایه ی A0 را برای ADC در نظر گرفته ایم ، اکنون با توجه به تصویر زیر ، میبینیم که ADC0 فعال است ، و همچنین کد مربوط به آن آورده شده است:



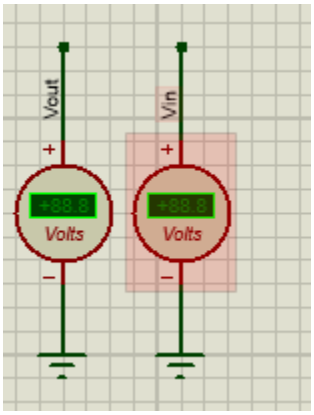
مقدار ورودی ADC برابر است با تفریق نقطه کار از خروجی مدار DAC برای برای پیاده سازی این بخش از یک تفریق کننده آنالوگ استفاده کرده ایم که شکل آن را در صفحه بعد آورده ایم.

همچنین یک منبع ولتاژ در نظر گرفته ایم تا ولتاژ رفرنس DAC را مشخص کند و مقدار آن را برابر 2.56 ولت قرار داده ایم.



شکل بالا ابتدا ولتاژ نقطه کار به Vin داده میشود و ولتاژ Vut از آن کم میشود سپس حاصل Vsub به ADC0 داده میشود تا کنترلر PID طرای شده در میکروپروسسور آن را پردازش کرده و خروجی را مشخص کند.

حال دو ولتمتر برای سنجش ورودی و ولتاژ Vout مطابق شکل زیر قرار میدهیم:



یک سنسور دماسنج نیز قرار میدهیم (LM35) که مقادیر دما را از -55 درجه تا 150 درجه برای ما بخواند و به ازای هر درجه 10 میلی ولت در خروجی تولید کند. از این سنسور به عنوان ورودی استفاده میکنیم. این سنسور برای تست استفاده شده است.

حال به بررسی کد ADC میپردازیم.

برای راه اندازی آن کافی است مقادیر مناسب با ADMUX و ADCSRA را در آن ها قرار دهیم و سپس با ست کردن بیت 6 ام در ADCSRA عمل تبدیل ولتاژ آنالوگ به دیجیتال را آغاز کنیم در ابتدای کد این عبارت را قرار میدهیم:

```
#define get_bit(reg,bitnum) ((reg & (1<<bitnum))>>bitnum)
```

و سپس با استفاده از دستور زیر مقدار ورودی ADC0 را بدست میاوریم:

```
DDRA = 0xFC;
DDRB = 0xF0;
DDRC = 0xFF;
DDRD = 0xFF;
ADMUX  = 0xC0;
ADCSRA = 0x87;
DDRA = 0xFC;

_delay_ms(20);
ADCSRA |= (1<<6);
while(get_bit(ADCSRA,6) == 1)
{

}
```

با اجرای کد بالا ابتدا بیت 6 ام ADCSRA برابر 1 میشود که باعث میشود عملیات تبدیل ولتاژ آنالوگ به دیجیتال شروع شود این عمل آنقدر طول میکشد تا بیت 6 ام مجدد 0 شود و شرط while برقرار نباشد که این تبدیل ولتاژ به یک عدد دیجیتال 10 بیتی است.

بخش مربوط به سیستم PID در حالت سیگنال گسسته:

یک کنترلر PID در سیستم پیوسته به شکل زیر تعریف میشود که ما باید آن را به حوزه z ببریم و برای اینکار S ها را باید با $\frac{2(Z-1)}{T(Z+1)}$ جایگذاری کنیم داریم:

$$PID\ CONTROLLER : SK_D + K_P + \frac{K_I}{S}$$

$$K_I * \frac{T(Z+1)}{2(Z-1)} + K_P + K_D * \frac{2(Z-1)}{T(Z+1)}$$

حال معادله رادر حوزه زمان مینویسیم:

$$PID = P[n] + I[n] + D[n]$$

$$P[n] = K_P(e[n])$$

$$I[n] - I[n-1] = K_I * \frac{T}{2} * (e[n] + e[n-1])$$

$$D[n] - D[n-1] = \frac{K_D T}{2}(e[n] - e[n-1])$$

مقدار T همان نرخ نمونه برداری ما میباشد حال مطابق با فرمول های بدست آمده کنترلر PID را طراحی میکنیم.

بخش مربوط به کنترل کننده PID:

از خط 28 تا 60 کد که در زیر آورده شده ابتدا همانطور که گفتیم به lcd مقداردهی اولیه و clear کرده ایم سپس از کاربر kp و ki و kd را دریافت کرده ایم (دو رقمی) سپس آن ها را به فرم ASCII در آورده ایم(با 48 جمع کردیم) و برای کاربر چاپ کرده ایم تا از صحت آن مطمئن شود.

```
28      Command(0x02);
29      Command(0x28);
30      Command(0x0c);
31      Command(0x06);
32      Command(0x01);
33      _delay_ms(2);
34      Clear();
35      LCDStr(" Kp :");
36      Command(0xC0);
37      Kpt = keyboard();
38      Kp  = keyboard() + Kpt*10;
39      Clear();
40      LCDStr(" Ki :");
41      Command(0xC0);
42      Kit = keyboard();
43      Ki  = keyboard() + Kit*10;
44      Clear();
45      LCDStr(" Kd :");
46      Command(0xC0);
47      Kdt = keyboard();
48      Kd  = keyboard() + Kdt*10;
49      Clear();
50      LCDStr("PID Control :");
51      Command(0xC0);
52      LCDStr("P=");
53      LCDCHARACTER(48 + Kpt);
54      LCDCHARACTER(48 + Kp - 10*Kpt);
55      LCDStr(",I=");
56      LCDCHARACTER(48 + Kit);
57      LCDCHARACTER(48 + Ki - 10*Kit);
58      LCDStr(",D=");
59      LCDCHARACTER(48 + Kdt);
60      LCDCHARACTER(48 + Kd - 10*Kdt);
```

کد مربوط به فرمول های بدست آمده:

```
ADCOut0 = ADCOut1;
ADCOut1 = ADCL|(ADCH<<8);
ADCOut1 = ADCOut1/4;
I0 = I1;
D0 = D1;
P1 = Kp*ADCOut1;
I1 = (Ki/100)*(ADCOut1 + ADCOut0) + I0;
D1 = (Kd*100)*(ADCOut1 - ADCOut0) + D0;
PORTC = P1 + I1 + D1;
```

ابتدا برای اینکه بتوانیم 10 بیت ADC را روی پورت C نمایش بدهیم آن را به 4 تقسیم میکنیم(8 بیت قابل نمایش روی پورت) در ادامه فرمول های بدست آمده را نوشته ایم که به ترتیب P1=P[n] و I1=I[n] و D1=D[n] است و P0=P[n-1] و I0=I[n-1] و D0=D[n-1] است . PORTC نیز PID ما میباشد که جمع سه مقدار P1 وi1 و D1 میباشد. سپس این خروجی PID به DAC داده شده و به ولتاژ آنالوگ Vout تبدیل میشود سپس با فیدبک و مدار تفریق کننده از ولتاژ نقطه کار کم میشود و Vsub را تشکیل میدهد دوباره به ADC0 داده میشود و این لوپ تکرار میشود.

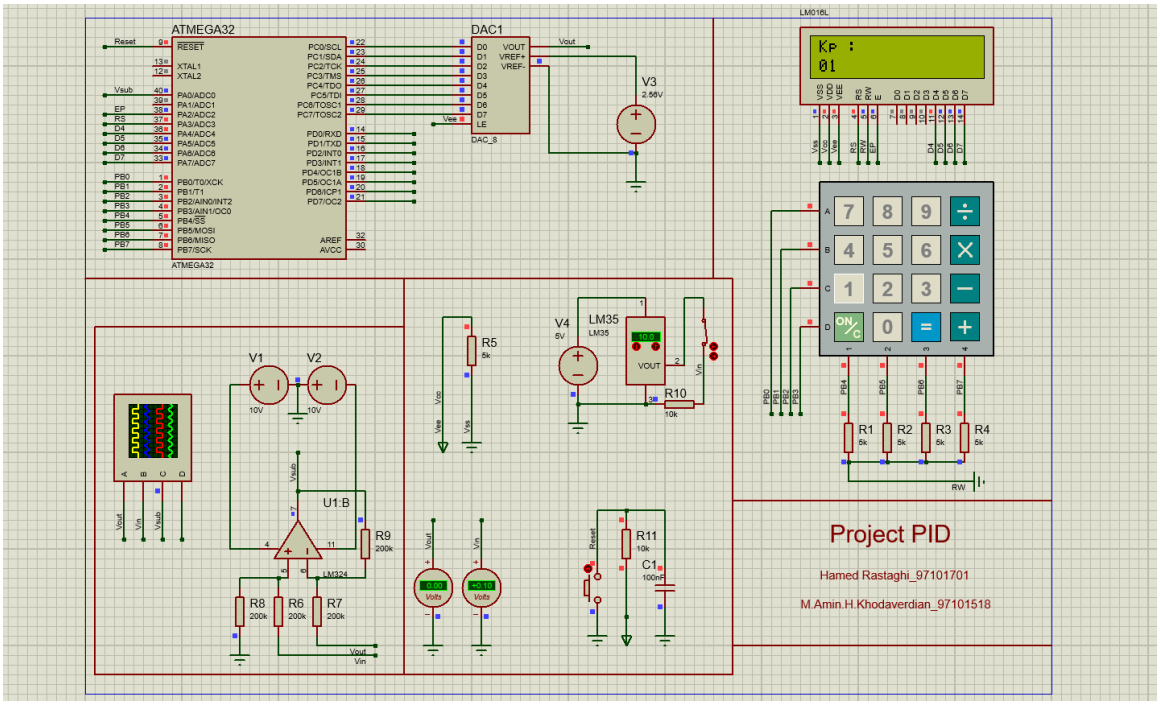
بخش مربوط شبیه سازی و خروجی PID:

با استفاده از نرم افزار ، Proteusفایل HEXکد نوشته شده را بر روی میکروپروسسورATMEGA32 قرار میدهیم و سپس عملکرد آن را بررسی میکنیم:

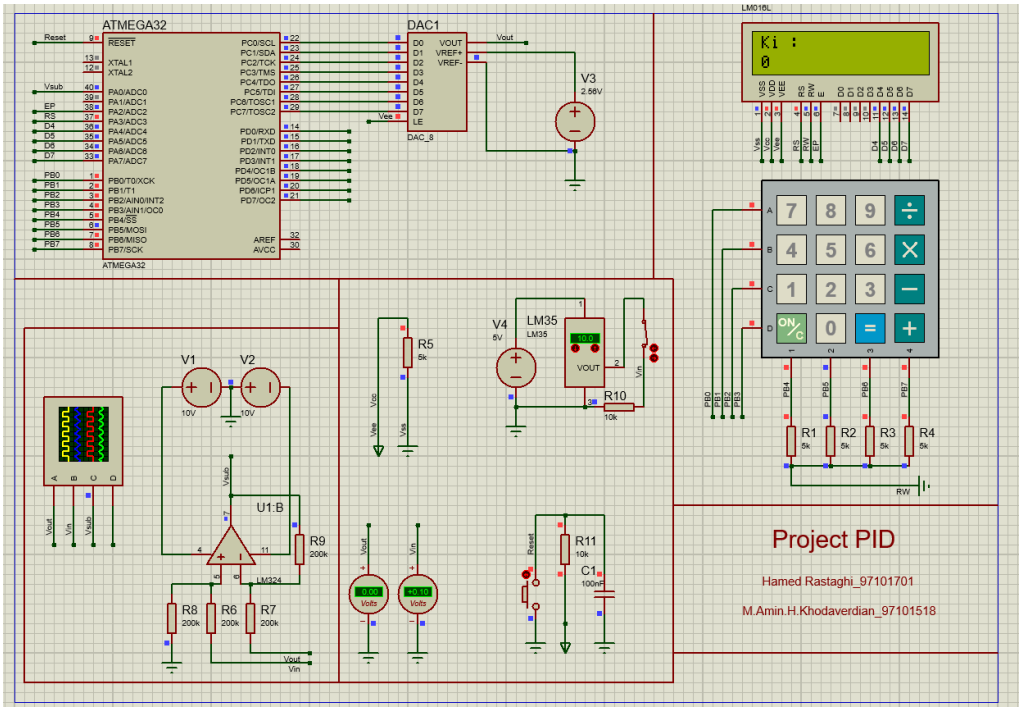
در ابتدا 3 ضریب K_p، K_I و K_D را به ترتیب از کاربر میگیرد در صورتی که در حین وارد کردن ضرایب مشکی پیش آمد با فشردن کلید On/C میتوان ورودی ها را مجدد از اول وارد کرد.

در شبیه سازی مقادیر را به ترتیب 01 و 00 و 00 قرار میدهیم.

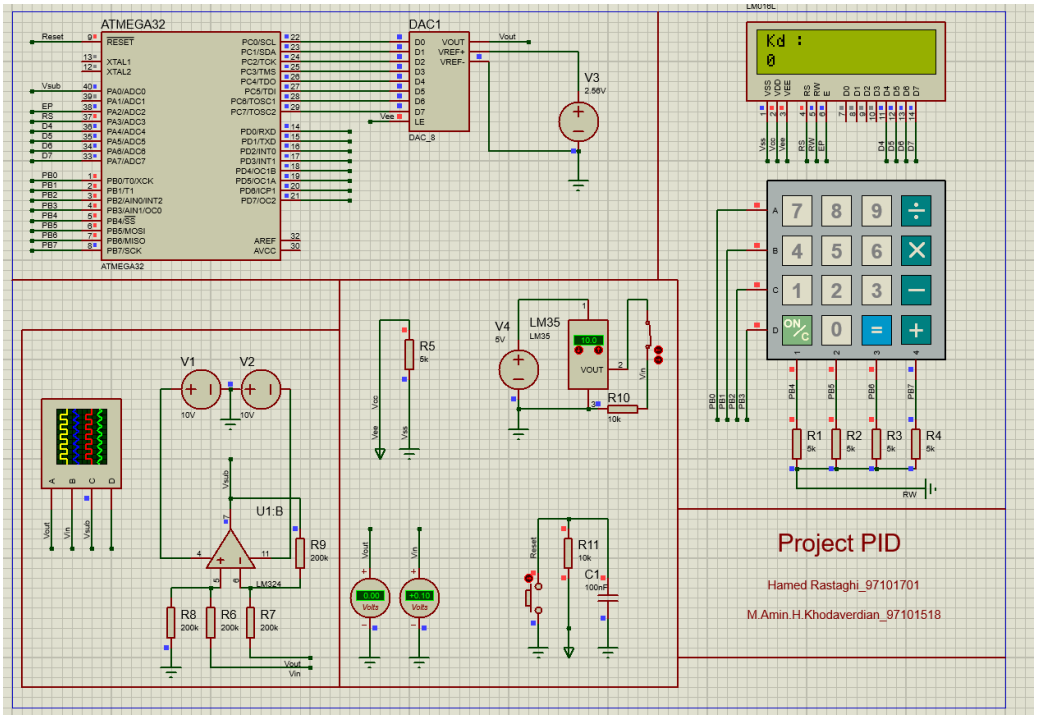
توجه: ورودی های ضرایب کنترل کننده حتما باید دو رقمی باشند و برای وارد کردن به عنوان مثال 2 باید 02 را به ترتیب وارد کرد.



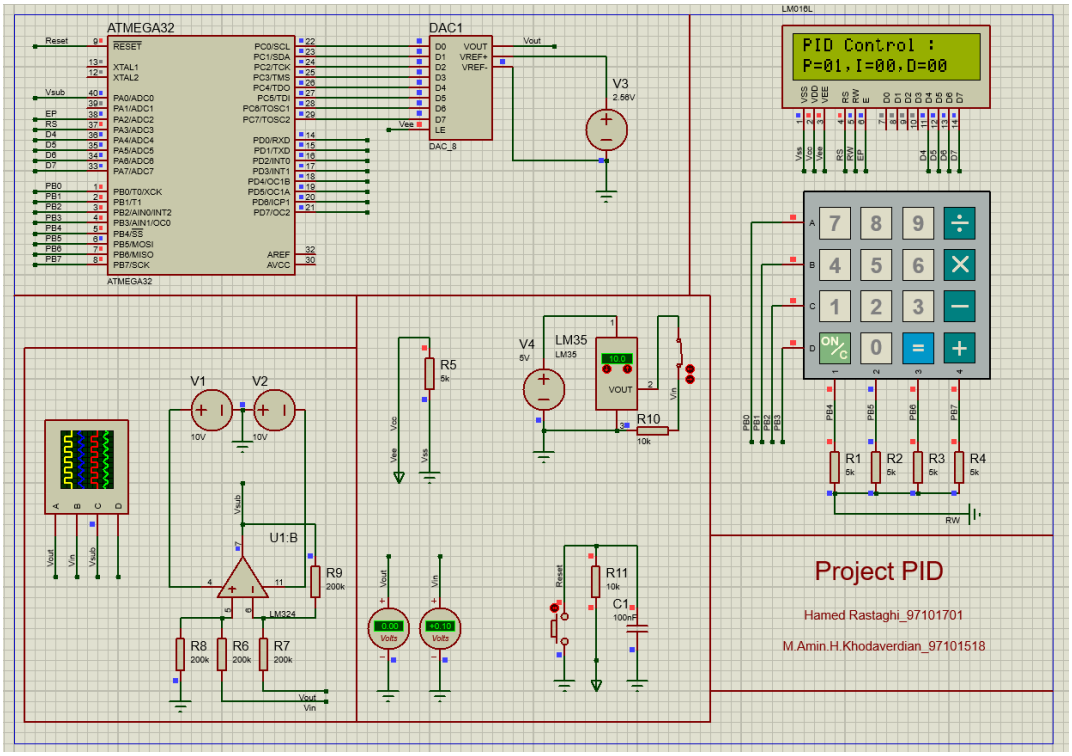
ورودی Kp



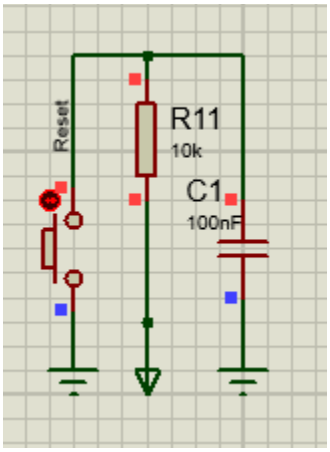
ورودی Ki



ورودی Kd



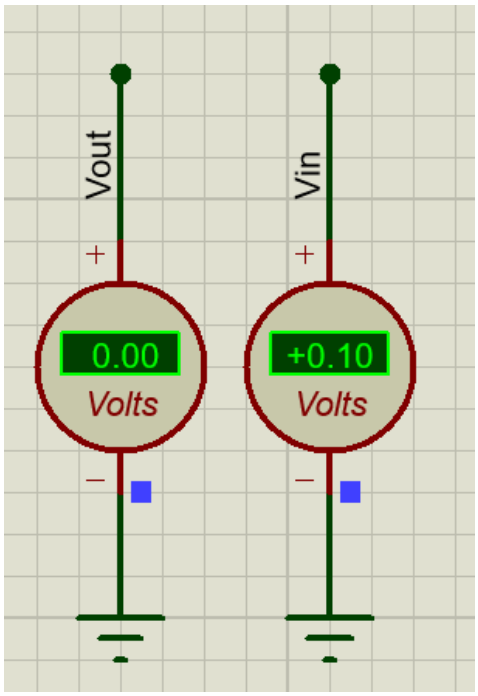
پس از وارد کردن مقادیر فوق LCD مقادیر وارد شده را به ترتیب با اسم نمایش میدهد و در صورت نیاز میتوان با استفاده از کلید Reset سیستم را ریست کرده و دوباره مقادیر را وارد کند

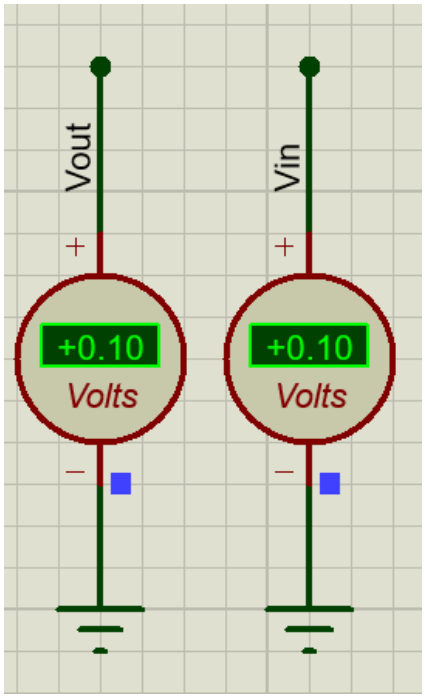


مدار Reset در بالا نشان داده شده است.

حالا با اعمال کردن ورودی به سیستم فوق خروجی را مشاهده میکنیم. لازم به ذکر است که سیستم فوق تنها یک PID است که توسط یک حلقه فیدبک هر بار ورودی خودش را از خروجی حالت قبل خود کم میکند.

با توجه به اینکه در این پروژه در مسیر PID هیچ سیستم دیگری قرار ندارد پس خروجی ما رفتار نوسانی را باید از خود نشان دهد که در زیر شکل آن نشان داده شده است که در حال نوسان کردن بین 0 و نقطه ورودی آن است:





بخش مربوط شبیه سازی و بررسی شمارنده و نمایش اعداد حافظه:

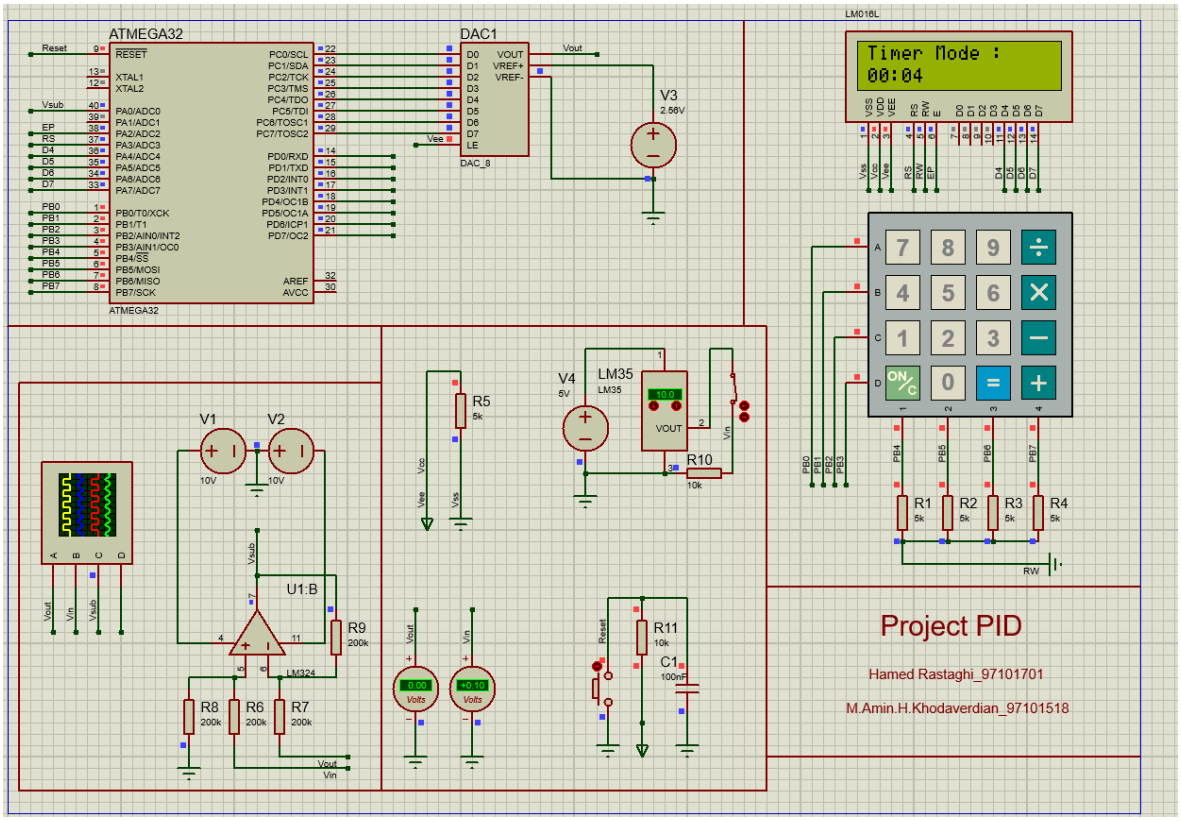
در بخش نهایی برنامه شمارنده 16 دقیقه مینویسیم. در ابتدا مقادیر آن 0 میشوند و پس از 16 دقیقه 6 عدد رندوم که در حافظه هستند را روی LCD به فرمت هگزا دسیمال نمایش میدهیم.

این 6 عدد به فاصله های 100 میلی ثانیه بر روی PORTD نیز نمایش داده خواهند شد .

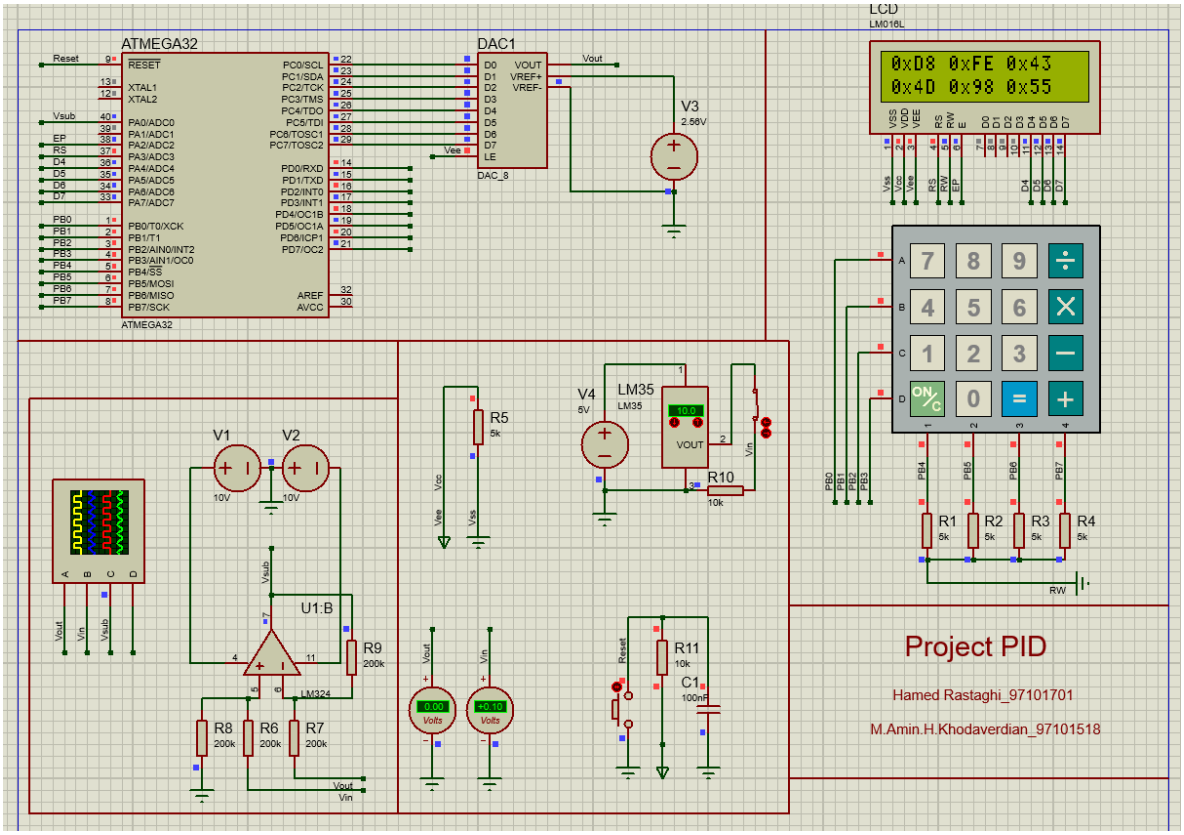
برای وارد شدن به مود شمارنده ابتدا کلید = بر از صفحه ی نمایش فشار خیدهیم سپس شمارنده روی LCD شروع به حرکت میکند و از 00:00 تا 16:00 شمارش را ادامه میدهد.

پس از آن 6 عدد از جنس کاراکتر که تولید شده است و در آدرس های ذکر شده ذخیره شده است به ترتیب با فاصله 1 ثانیه روی LCD ظاهر میشوند.

حال به شبیه سازی آن میپردازیم:



پس از فشردن کلید = مود عوض شده و برروی LCD شمارنده نشان داده خواهد شد حال باید 16 دقیقه صبر کنیم تا اعداد ذخیره شده در حافظه بر روی LCD نمایش داده شود.



پس از نوشته شدن اعداد بالا مجدد به مود کنترل کننده میرود و آماده دریافت ضرایب میشود حال اگر بخواهیم مجدد وارد تایمر شویم باید = را از روی صفحه نمایش فشار دهیم.

بررسی کد :

در این قسمت از یک تابع اصلی به نام TIMER ویک تابع کمکی به نام table استفاده کرده ایم که به توضیح آن میپردازیم.

در تابع table که در ادامه آمده است یک نیبل را به عنوان ورودی میگیریم و کد ASCII متناظر با آن را تولید میکنیم.

```
259   void table(int digit)
260   {
261       if (digit < 10) LCDCHARACTER(digit + 48);
262       else if (digit == 10) LCDCHARACTER('A');
263       else if (digit == 11) LCDCHARACTER('B');
264       else if (digit == 12) LCDCHARACTER('C');
265       else if (digit == 13) LCDCHARACTER('D');
266       else if (digit == 14) LCDCHARACTER('E');
267       else if (digit == 15) LCDCHARACTER('F');
268   }
```

در تابع اصلی که تایمر نام دارد و عکس آن در زیر آمده است دو متغیر second و minute داریم که هر ثانیه second را زیاد میکنیم و هر 60 ثانیه یک مرتبه minute را زیاد میکنیم. هربار که second زیاد میشود مقادیر متناظر را به کمک دو حلقه for در lcd نمایش میدهیم و در PORTD نیز خروجی میدهیم که با شبیه سازی میتوانیم صحت این قسمت از کد را نیز بررسی کنید.

```
278   void Timer(void)
279   {
280       int Check = 1;
281       int  Minute = 0, Second = 0;
282       Clear();
283       LCDStr("Timer Mode :");
284       Command(0xC0);
285       LCDStr("00:00");
286       while(Check != 0)
287       {
288           Command(0xC0);
289           Second = Second + 1;
290           _delay_ms(78);
291           Minute = Minute + Second / 60;
292           Second = Second % 60;
293           Minute = Minute % 60;
294           LCDCHARACTER(48 + Minute / 10);
295           LCDCHARACTER(48 + Minute % 10);
296           LCDCHARACTER(':');
297           LCDCHARACTER(48 + Second / 10);
298           LCDCHARACTER(48 + Second % 10);
299           if (Minute == 16) Check = 0;
300           _delay_ms(1);
301       }
302       Clear();
303       for( int i = 0 ; i < 3 ; i++)
304       {
305           int digit1,digit0;
306           PORTD = rand()%256;
307           LCDStr("0x");
308           digit0 = PORTD % 0x10;
309           digit1 = PORTD / 0x10;
310           table(digit1);
311           table(digit0);
312           LCDCHARACTER(' ');
313           _delay_ms(1000);
314       }
315       Command(0xC0);
```

```
316     for( int i = 0 ; i < 3 ; i++)
317     {
318         int digit1,digit0;
319         PORTD = rand()%256;
320         LCDStr("0x");
321         digit0 = PORTD & 0x10;
322         digit1 = PORTD / 0x10;
323         table(digit1);
324         table(digit0);
325         LCDCHARACTER(' ');
326         _delay_ms(1000);
327     }
328     main();
```

پایان