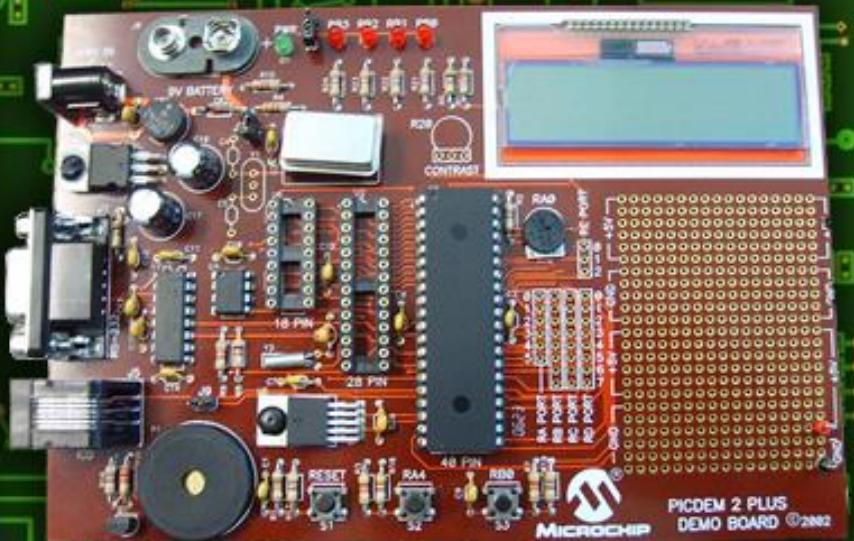


- PIC 16F877A
Mikrodenetleyici
Donanımı
- Tüm Yönüyle Hi-Tech
Derleyicisi
- Port Giriş/Çıkış, Buton
ve Döngü İşlemleri
- Kesmeler,
Zamanlayıcı/Sayıci
Birimleri, CCP, ADC
Modülleri
- Dahili ve Harici
EEPROM
- 7 Segment ve Matris
LED Display
- Tuş Takımı
- Karakter ve Grafik LCD
- RS232, I2C ve SPI
İletişimleri
- 1-Wire (Tek Hat)
İletişim
- DC, Step ve R/C Servo
Motorlar
- LM35 Sıcaklık Sensörü
- DS18B20 ve TC72
Sıcaklık Sensörleri
- DS1302 RTC
- DS1868 Dijital
Potansiyometre
- 2x20 Karakter LCD ve
3-Wire LCD Kullanımı
- PS/2 Klavye
- Nokia 3310 Ekranı
- Kayan Yazı Uygulaması

HI-TECH ile PIC Programlama



Fırat Deveci



MICROCHIP

©2009

firatdeveci.com

Ön Söz

Antik Yunan'da ilk kehribarın maddeleri çekmesiyle başlayan elektrik serüveni tarih içinde bir çok değişik formda bulunmuş, değişmiş, evrim geçirmiştir. Galvani, Volta, Faraday, Amperé ve Ohm'un katkılarıyla bilimsel alanda önemi artmaya başlayan elektrik, her dönem kendinden söz ettirmeyi başarmıştır. O kadar ki zamanın Amerikan başkanı Benjamin Franklin'den de katkılar almıştır. 19. ve 20. yüzyıllarda altın çağını yaşayan elektrik özellikle Tesla, Morse, Edison ve Bell'in katkılarıyla bambaşka bir yolda ilerlemeye başlamıştır. Özellikle 2. Dünya savaşından sonra güç yarışında belirleyici olan enerji faktörünün en önemli öğesi olan elektrik, 50'lerden sonra Shockley ve grubunun transistorü bulmasıyla bilim adamlarına ve kullanıcılarına bambaşka dünyaları aralamıştır. Özellikle 60 ve 70'li yıldandan sonra muazzam bir yükselişe geçen elektronik böylelikle, elektrikten konu itibariyle ayrılmıştır, fakat her ne olursa olsun birbirlerinin siyam ikizi olmalarını engelleyememişlerdir.

Günümüze yaklaşıkça transistorlerin belirli dizilimlerini kullanıcının değiştirebilmesiyle gelişen elektronik 80'lerin sonuna doğru yeni bir yapıyı oluşturacaktı: Mikrodenetleyiciler.

Dünya mikrodenetleyici firmalarının başında gelen Microchip'in Pic adını verdiği mikrodenetleyiciyi anlatmayı planladığım derslerde özellikle 1972'den bu yana sistem dili olarak kullanılan C temel alınacak, microchip'in yakın tarihte satın aldığı Hi-Tech Pic C derleyicisi üzerinde örnekler verilecektir.

**Fırat Deveci
Ağustos 2009
info@firatdeveci.com**

Elektrik ve elektroniğe gönül veren herkese...

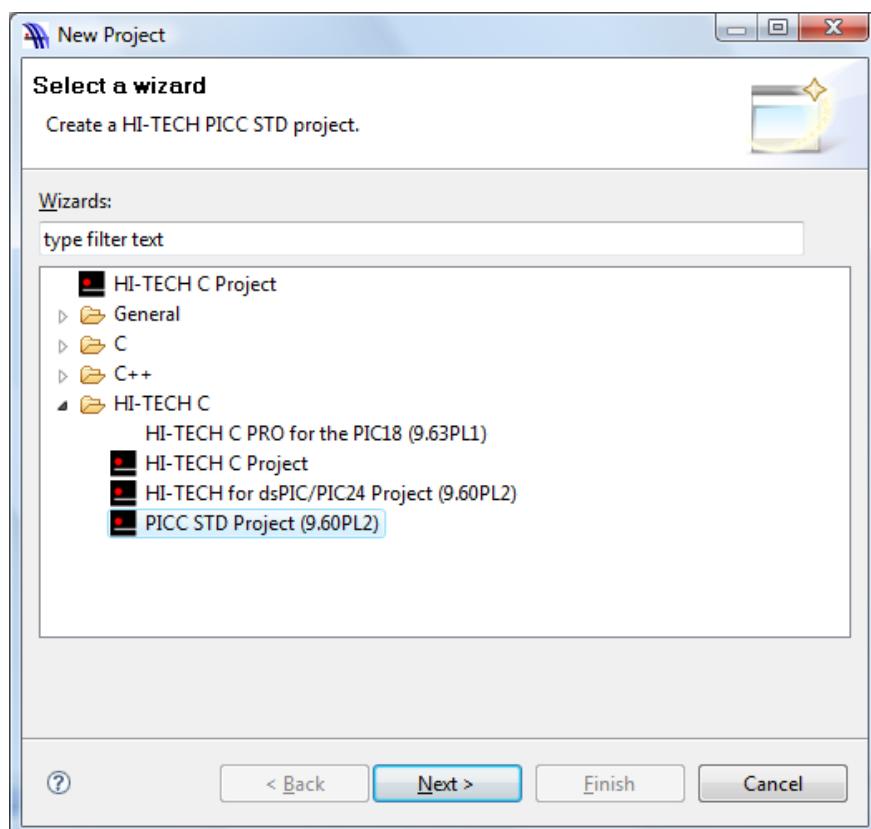
BÖLÜM 1 – PIC GİRİŞ ÇIKIŞ AYARLARI, LED, DOT MATRİS VE SEVEN SEGMENT UYGULAMALARI

1.1) Hi-Tech'te Yeni Proje Oluşturmak

Pic programlamaya başlamadan önce kısaca kullanacağımız program hakkında bilgi vermek istiyorum.

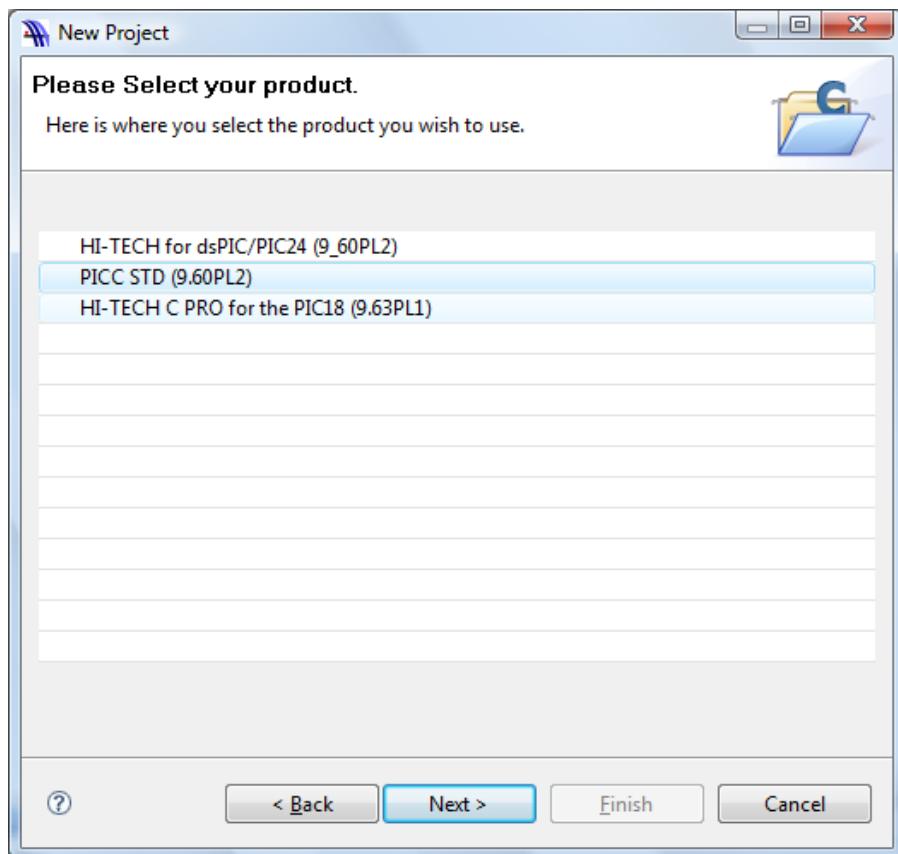
Hi-Tech, **Microchip**'in kullanıcılarına sunduğu bir derleyicidir. Tek başına bu derleyici kullanmak mümkün değildir. Onun için mutlaka bir program yazım editörüne ihtiyaç vardır. Bunun için en çok kullanılan ara yüz programları ise **MPLAB** ve **Hi-Tide**'dır. Ben uygulama yaparken özellikle görsel açıdan hoş, kullanması kolay, daha önce program yazanların alışık olduğu **Eclipse**'in bir uyarlaması olan **Hi-Tide** arayüzüni kullanacağım. Bu derleyici kullanmak isterseniz sitesinden ücretsiz indirebilirsiniz.

Hi-Tide'da proje oluşturmak için öncelikle **File/New/Project** sekmelerine basılır. Karşımıza şekil-1 deki gibi bir ekran çıkması gereklidir. Burada daha önce yüklediğimiz Hi-Tech versiyonlarını görmemiz mümkündür.



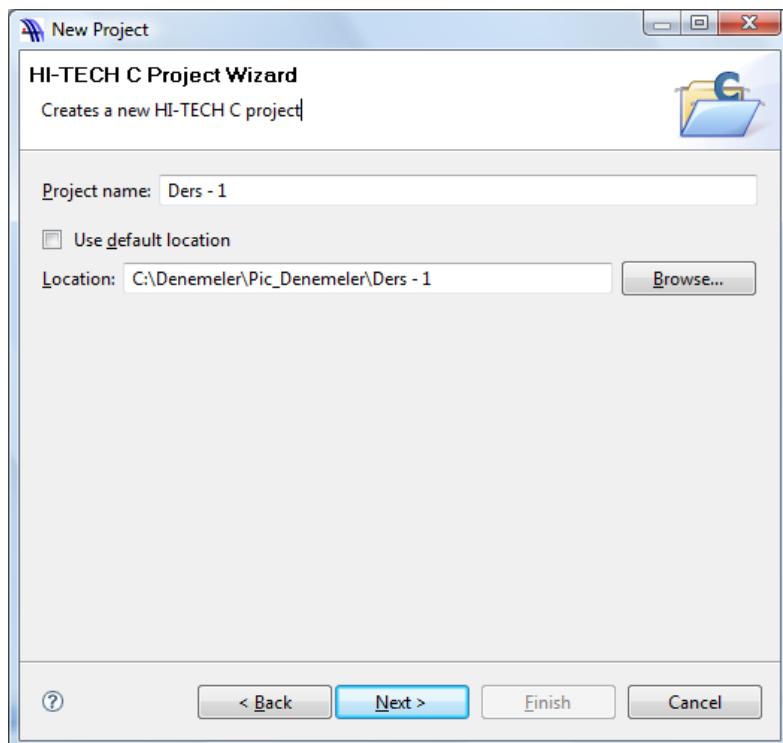
Şekil 1 – Yüklü Derleyiciler

Buraya **next** dedikten sonra asıl derleyicinin seçileceği bölüme gelinir. Biz 16f serisi picler üzerine çalışacağımız için şekil-2'de görülen **PICC STD (9.60 PL2)** seçilir. Eğer daha üst versiyonları kullanıyorsanız yine aynı şekilde yüklediğiniz derleyiciyi seçerek **next** dememiz yeterli olacaktır.



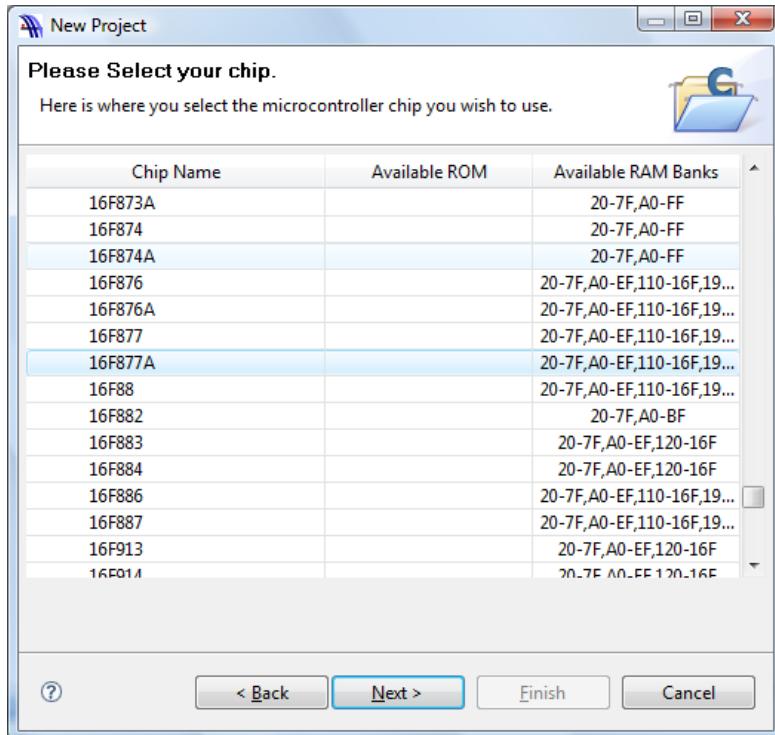
Şekil 2 - Derleyici Seçim Kısmı

Bu bölümden sonra projemizi bilgisayarımızda nereye oluşturmak istediğimize dair bir pencere bizleri karşılaşacaktır. Dilediğiniz yeri seçtikten sonra şekil-3'teki ekrana da **next** demeniz yeterli olacaktır.



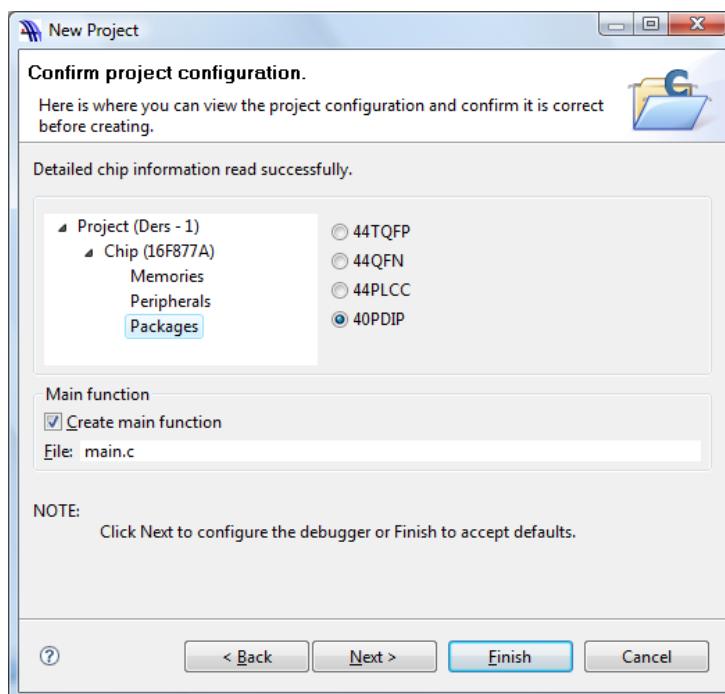
Şekil 3 - Proje Oluşturacağımız Yeri Seçme Ekranı

Bu bölümden sonra aşağıdaki ekrana **next** dememiz ve şekil-4'teki bölüme gelmemiz gerekmektedir. Bu bölüm, bu kısımdaki en can alıcı noktadır. Hangi pic ile çalışacaksak onu bu bölümden seçip **next** dememiz gerekmektedir. Ben dersler içinde bir tutarlılık olması dolayısıyla **16f877a** ile çalışmayı uygun görüyorum. Listededen bu pic'i seçip **next** diyoruz.



Şekil 4 - Pic Seçim alanı

Daha sonraki bölümde şekil-5'te görüleceği gibi kullandığımız pic'in paket yapısını seçmemiz gerekmektedir. Bizim kullandığımız pic'ler genelde DIP soket olduğundan 40DIP'i seçip finish diyoruz.

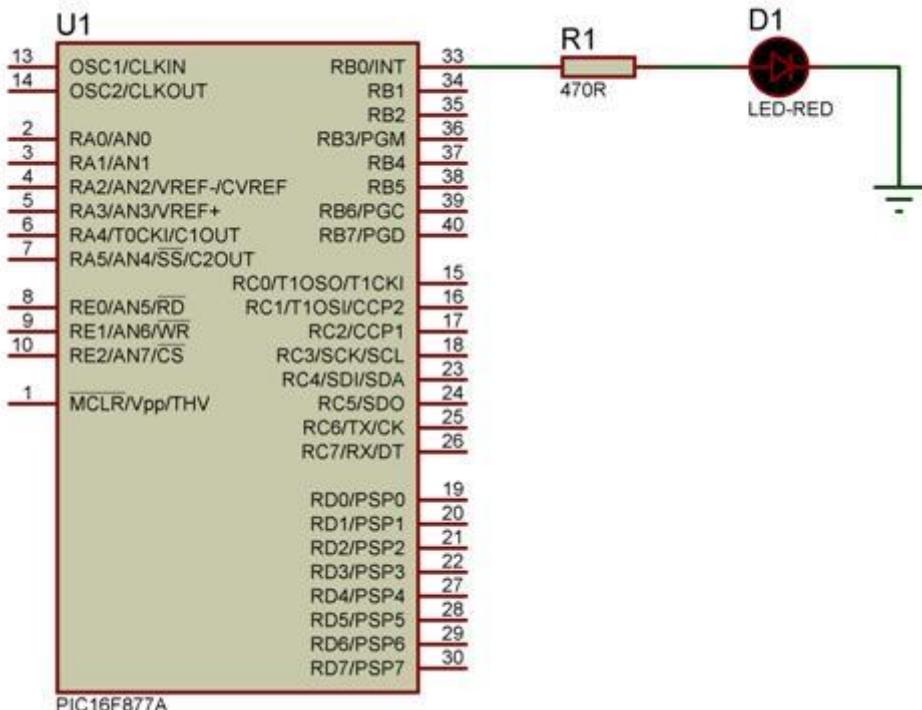


Şekil 5 - Pic Özellikleri ve Paket Yapısı

Şekil-5'teki bölüme **finish** dediğimizde **main** dosyamız kendiliğinden oluşacak ve artık kod yazmaya hazır hale gelecektir.

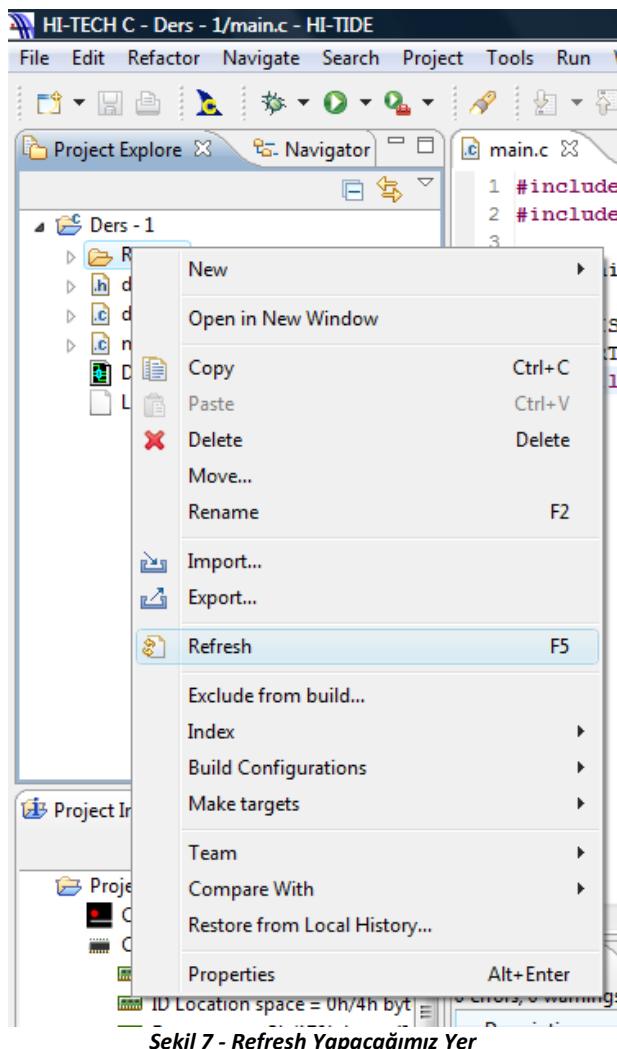
1.2) Hi-Tech'de İlk Proje

İlk uygulama olarak klasik led yakıp söndürme işlemini yapacağız. Şekil-6'daki gibi Proteus'ta devremizi oluşturuyoruz.



Şekil 6 - İlk Deneyimiz

Devremizi oluşturduktan sonra öncelikle gecikme yaratarak, aynı zamanda yavaş yavaş C'de işlemlerin nasıl yürüdüğünə dair örnek vereceğiz. Gecikme kütüphanemiz olan **delay.h** ve **delay.c** dosyalarını **Program Files\HI-TECH Software\PICC\STD\9.60\samples\delay** klasöründen alıp (siz derleyiciyi nereye kurdusunuz oradan alınız) projemizi oluşturduğumuz klasörün içine atıyoruz. Bu dosyaları proje klasörümüze ekledikten sonra **Hi-Tide** ekranından proje ismine gelip şekil-7'deki gibi **refresh** yapmamız gerekmektedir.



Şekil 7 - Refresh Yapacağımız Yer

Bu dosyaların ne işe yaradığına gelince; C'nin en büyük özelliklerinden biri kütüphane oluşturabilmemizdir. Eğer bu iki dosyayı açıp bakarsanız kullanacağımız gecikmelerin çeşitli ifadelerle sağlandığını göreceksiniz. Daha sonra kendimiz kütüphane oluşturduğumuzda bu konuya daha kolay anlayacaksınız fakat şimdilik dosyaları oraya atmakla yetinelim.

Burada dikkat edilmesi gereken bir konu da **delay.h** dosyası içindeki

#define Xtal_FREQ 4MHZ

yazan kısımdır; burada seçtiğimiz kristale göre 4MHZ kısmını 8MHZ, 16MHZ şeklinde değiştirebilirsiniz. Fakat kristal değeriniz 4Mhz ise o kısma dokunmayabilirsiniz.

Tüm bunlardan sonra artık kodumuzu yazmaya geçebiliriz, işlem olarak **ledin her yarımsaniye de bir yanıp sönmesi** olayını C kodıyla gerçekleştirelim.

```

#include <htc.h>
#include "delay.h"           // Gecikme yaratacak kütüphane

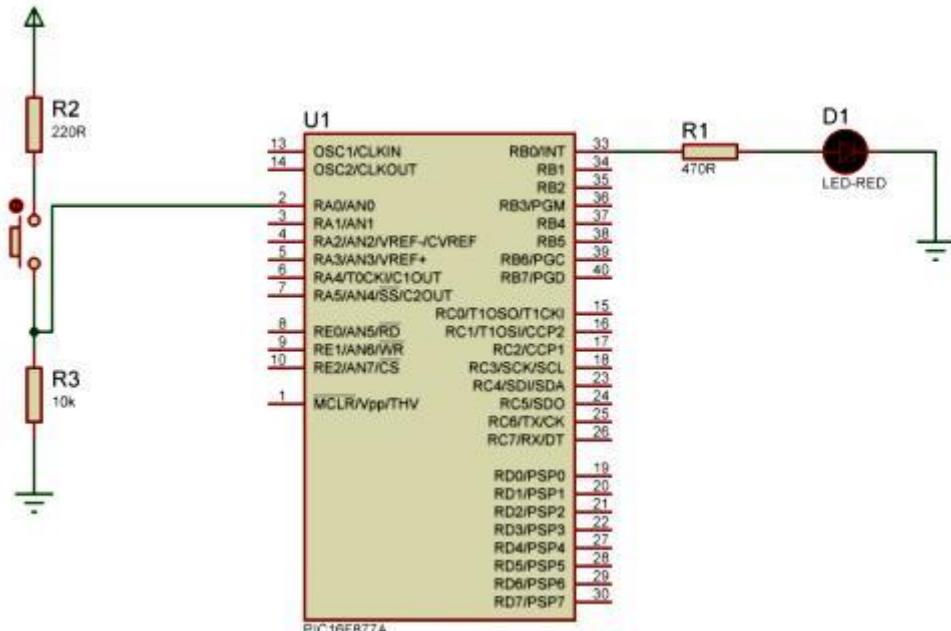
void main(void)             // Ana fonksiyon alanı
{
    TRISB=0x00;            // PORTB çıkış olarak yönlendiriliyor
    PORTB=0x00;             // PORTB'nin tüm çıkışları sıfırlanıyor
    while (1)               // Sonsuz döngüye giriliyor
    {
        RB0=1;                // Led yanacak
        DelayMs(250);DelayMs(250); // Yarım saniye beklenecek
        RB0=0;                // Led sönecek
        DelayMs(250);DelayMs(250); // Yarım saniye beklenecek
    }
}

```

Kodlarımızı şekildeki gibi yazdığımızda **save** tuşuna basarak **.hex** dosyamızı oluşturmuş olacağız. **.hex** dosyamız proje klasöründe bulunan **Release** klasörünün içinde olacaktır. Proteus'tan bu dosyayı alıp çalıştırduğumuzda yarımsaniyede bir ledin yanıp söndüğünü göreceksiniz.

1.3) Buton ile Led Kontrol

Şimdi ise devremize bir buton bağlayarak **bastiğımızda ledin yanmasını, çektiğimizde de ledin sönmesini** sağyalalım. Bunun için Proteus'ta şekil-8'deki devreyi kuralım.



Şekil 8 - Led Yakın Söndürme Uygulaması Devresi

Bu devrede **delay.h** ve **delay.c** dosyalarımızı kullanmayacağımız için proje klasöründen silmeliyiniz. **Unutmayın ki gereksiz her kütüphane ve fonksiyon pic'de boş yer kaplayacaktır.**

İstediğimiz işlemi yapan C kodunu aşağıdaki tabloda görebilirsiniz.

```

#include <htc.h>

void main(void) // Ana fonksiyon alanı
{
    ADCON1=0x07; // PORTA dijital olarak yönlendiriliyor
    TRISA=0x01; // RA0 giriş olarak yönlendiriliyor
    TRISB=0x00; // PORTB çıkış olarak yönlendiriliyor
    PORTB=0x00; // PORTB'nin tüm çıkışları sıfırlanıyor
    PORTA=0x00; // PORTA'nın tüm çıkışları sıfırlanıyor
    for(;;) // Sonsuz döngüye giriliyor
    {
        RB0=RA0; // RB0 çıkışı RA0 girişine eşitleniyor
    }
}

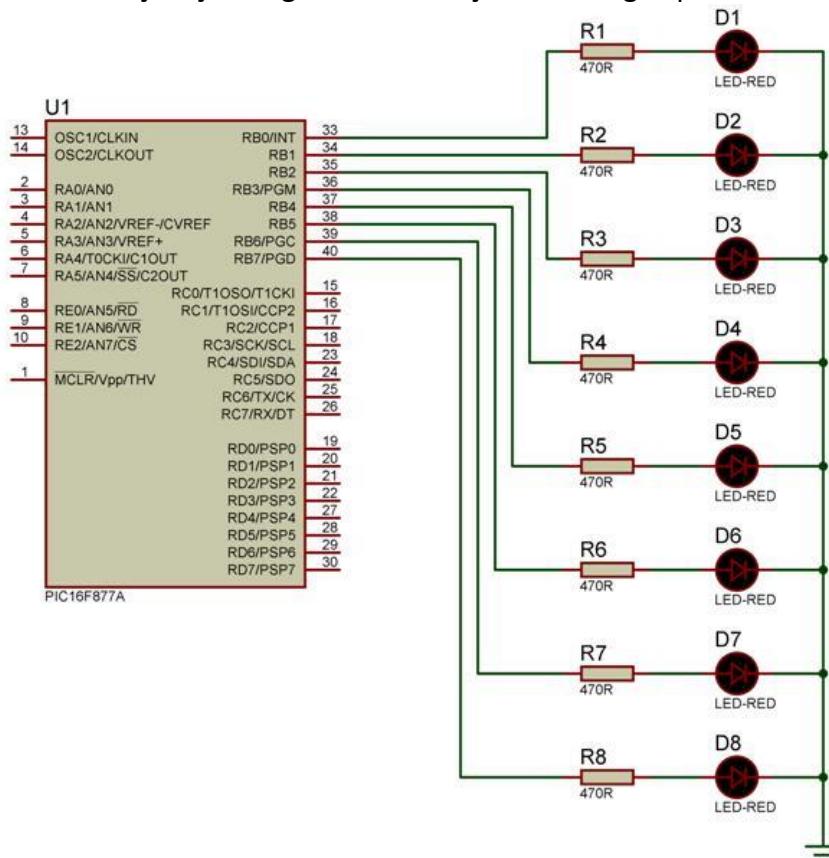
```

C programlama dilinde sonsuz döngüye **while(1)** şeklinde girilebileceği gibi **for(;;)** şeklinde de girilebilir. Ayrıca yine bir çok yolla sonsuz döngüye girmek de mümkündür.

Yukarıdaki kodlarda ilginç görülebilecek **ADCON1**, **PORTA** portunu dijital giriş çıkış yapmaya izin verir. Eğer bunu yapmasaydık **PORTA** analog olacak ve dijital olarak giriş ve çıkış işlemleri yapamayacaktık. Ayrıntılı bilgi için lütfen **datasheet**'e ve ileride anlatacağızımız **ADC** konusuna göz atınız.

1.4) Karaşımşek Uygulaması

Bu uygulamamızda ise ünlü Kara Şimşek dizisindeki Kit'in önünde bulunan led animasyonunu gerçekleştireceğiz. Ayrıca C'nin çeşitli özelliklerini de bu uygulamamızda görmek maksadıyla kodları biraz daha karmaşıklaştıracagız. Devremizi şekil-9'daki gibi proteus'ta kuralım.



Şekil 9 - Karaşımşek Devresi

```

#include <htc.h>
#include "delay.h"           // Gecikme kütüphanesi

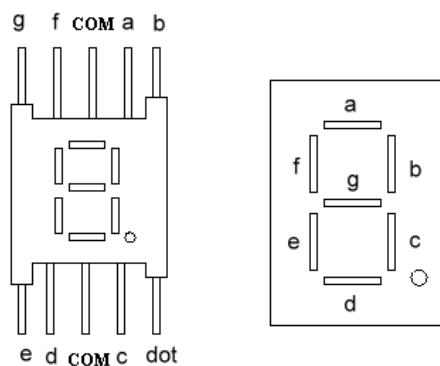
void main(void)             // Ana fonksiyon alanı
{
    char led=1;              // led şeklinde bir sabit tanımlanıyor
    TRISB=0x00;               // PORTB çıkış olarak yönlendiriliyor
    PORTB=0x00;                // PORTB'nin tüm çıkışları sıfırlanıyor
    for(;;)                  // Sonsuz döngüye giriliyor
    {
        PORTB=led;            // PORTB led değişkenine eşitleniyor
        led=led<<1;           // led birimi bir sola kaydırılıyor
        DelayMs(100);          // 100ms bekleniyor
        if(led==0x80)           // Eğer PORTB=0x80 olursa alt işlemlere
geçiliyor
        {
            for(;;)           // Tekrar sonsuz döngüye giriliyor
            {
                PORTB=led;      // PORTB led değişkenine eşitleniyor
                led=led>>1;     // led birimi bir sağa kaydırılıyor
                DelayMs(100);    // 100ms bekleniyor
                if(led==0x01)      // Eğer PORTB=0x01 olursa ikinci
sonsuz döngüden
                break;           // birinci sonsuz döngüye giriliyor
            }
        }
    }
}

```

C kodunda girilen **break** kodu içerisinde bulunan döngüden çıkmaya yarayan bir C kodudur. İleriki derslerde bazı uygulamalarda bu kodun oldukça yararlı olduğunu göreceksiniz.

1.5) Seven Segment Gösterimi, 0-9 ve 00-99 Sayıcı

Birçok uygulamada kullanılan seven segmentler esasen bir araya getirilmiş 7 ledden oluşur. Şekil-10'da tek bir seven segment'in bağlantı şékli gözükmektedir.



Seven-Segment Display

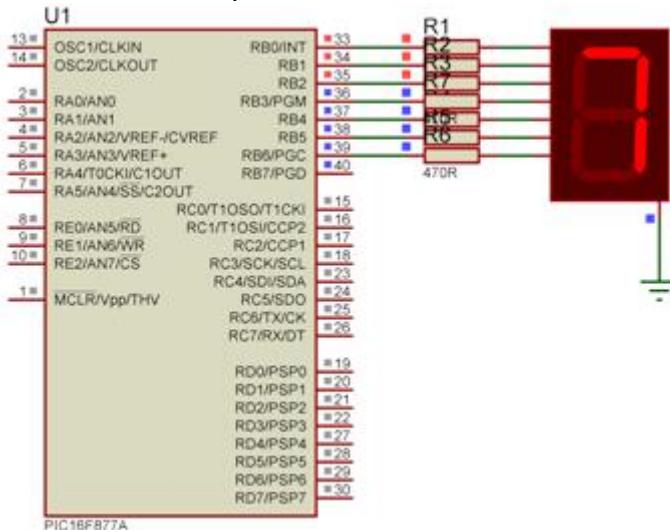
Sekil 10 - Seven Segment

Ben genelde ortak katot seven segment kullanmayı tercih ediyorum ve kodlarımı buna göre yazıyorum. Şekil-10'da da görüleceği üzere bazı sayıları ve harfleri çıkarmak için çeşitli kombinasyonlarda ledleri yakmak gerekmektedir. Bu konuda da C'nin bir diğer güzel özelliği olan **diziyi** kullanmak yazacağımız kod için gayet kullanışlı olacaktır.

Kullanacağımız diziyi şu şekilde tanımlayabiliriz;

```
const unsigned char segment[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};
```

Bu diziyi tanımladıktan sonra devremizi 0'dan 9'a ve tekrar 0'a saydıracak şekil-11'deki devreyi kurduktan sonra C kodumuzu yazalım.



Şekil 11 - 0-9 Sayıcı

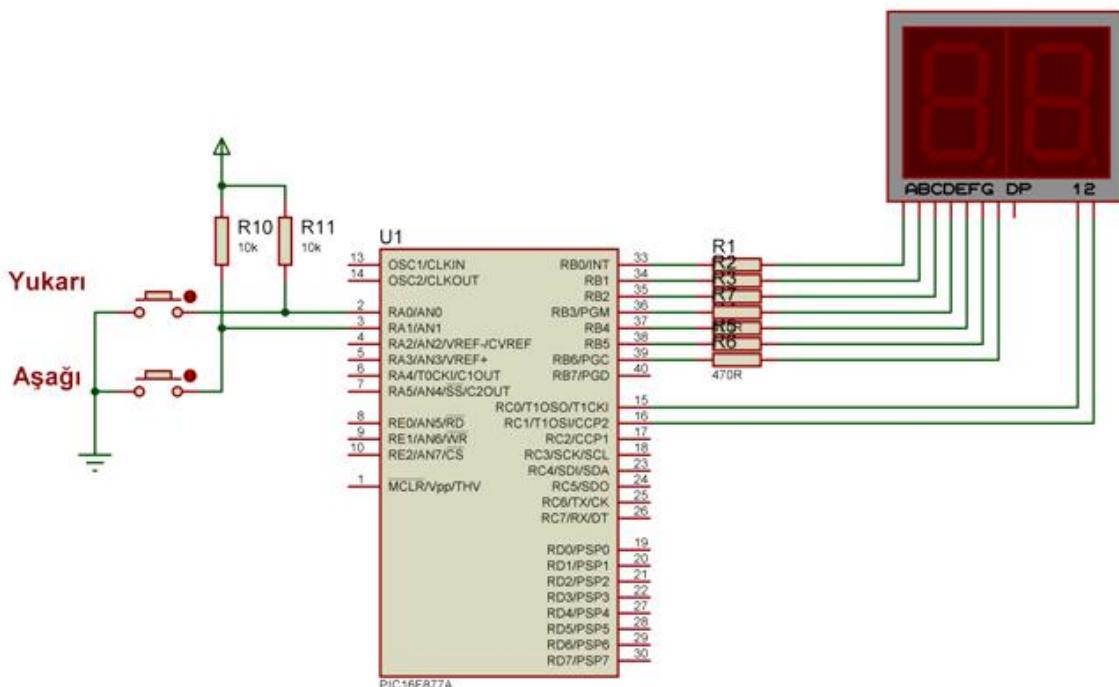
Aşağıdaki C kodunda **constant** değeri C'de sabitler için kullanılan bir metottur. Genel itibarı ile **segment[0]** seven segmentte '0' değerini gösteren dizi değeri, **segment[1]** seven segmentte '1' değerini gösteren dizi değeridir. Tüm dizi değerleri bu şekilde ayarlanmıştır.

```
#include <htc.h>
#include "delay.h"          // Gecikme kütüphanesi

const unsigned char
segment[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};

void main(void)           // Ana fonksiyon alanı
{
    char i;                // Herhangi bir değişken tanımlanıyor
    TRISB=0x00;             // PORTB çıkış olarak yönlendiriliyor
    PORTB=0x00;              // PORTB'nin tüm çıkışları sıfırlanıyor
    for(;;)                // Sonsuz döngüye giriliyor
    {
        PORTB=segment[i];   // Seven segment değerleri alınıyor
        i++;                 // i bir artırılıyor
        DelayMs(250);        // 250ms bekleniyor
        if(i>9)               // Eğer sayı 9'dan büyük ise 0'a dön
            i=0;                // Değişken 0 yapılıyor
    }
}
```

Şekildeki örneği yaptıktan sonra tarama metodunu öğretmek maksadıyla 00-99 sayıci devresini şekil-12'deki gibi tasarlayalım. Bu tasarımında sayma işlemini aşağı ve yukarı butonlarını da katarak etkinlik katalım.



Sekil 12 - 00-99 Sayıcı Devresi

Şekil-12'deki devreyi kurduktan sonra C kodumuzu yazalım.

```
#include <htc.h>
#include "delay.h" // Gecikme kütüphanesi

const unsigned char
segment[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};

void sayi_goster(char i) // Sayı göstermeye yarayan fonksiyon
{
    PORTC=0x02; // PORTC'de 2 değeri gönderiliyor
    PORTB=segment[i/10]; // i'nin 10'a bölümü gösteriliyor
    DelayMs(5); // 5ms bekleniyor
    PORTC=0x01; // PORTC'de 1 değeri gönderiliyor
    PORTB=segment[i%10]; // i'nin 10'a bölümünden kalani gösteriliyor
    DelayMs(5); // 5ms bekleniyor
}

void main(void) // Ana fonksiyon alanı
{
    int i; // Herhangi bir değişken tanımlanıyor
    ADCON1=0x07; // PORTA dijital yapılıyor
    TRISA=0x03; // PORTA'nın ilk iki pini giriş
    TRISB=0x00; // PORTB çıkış olarak yönlendiriliyor
    TRISC=0x00; // PORTC çıkış yapılıyor
    PORTA=0x00; // PORTA'nın tüm çıkışları sıfırlanıyor
    PORTB=0x00; // PORTB'nin tüm çıkışları sıfırlanıyor
    PORTC=0x00; // PORTC'nin tüm çıkışları sıfırlanıyor
    for(;;) // Sonsuz döngüye giriliyor
    {
        if(RA0==0) // RA0'pini 0 mı?
        {
            while(!RA0); // Buton bırakıldı mı diye bakılıyor
            i++; // Değişken artırılıyor
            if(i>99) // Eğer değişken 99'dan büyükse 0 oluyor
                i=0;
        }
    }
}
```

```

        }
        else if(RA1==0)           // RA1'pini 0 mı?
        {
            while(!RA1); // Buton bırakıldı mı diye bakılıyor
            i--;
            // Değişken azaltılıyor
            if(i<0)          // Eğer değişken 0'dan küçükse 99 oluyor
                i=99;
        }
        sayi_goster(i);      // O anki sayı gösteriliyor
    }
}

```

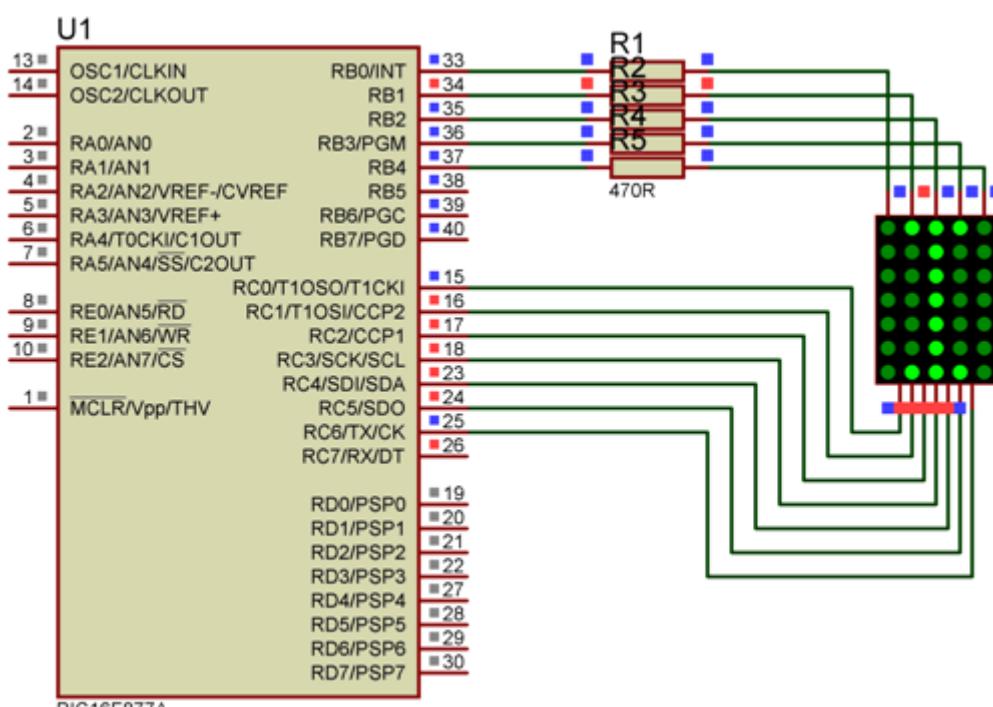
Göründüğü üzere tarama yöntemi bize port kazanımı sağlıyor. Bu özellikle büyük sayıdırma işlemlerinde, saat devrelerinde oldukça işe yarayan bir metottur. Biz burada bu işlemi özel bir fonksiyonla, elle yaptıysak da ileriki konularda kesmelerle bu işin daha da kolaylaşabileceğini göreceğiz.

1.6) Dot Matris Uygulaması

Dot matrisler özellikle tabela uygulamalarında günümüzde sıkılıca kullanılan araçlardır. İçlerinde çeşitli şekillere dizilmiş ledler bulundururlar. Günümüzde en çok kullanılanları ise 8x8 ve 5x7'lik modellerdir.

Biz bu uygulamamızda 5x7'lik dot matriste çeşitli harfleri yazmayı öğrenirken, C dilinde kullanılan dizilerin de ne kadar faydalı olduğunu tekrar göreceğiz.

Öncelikle şekil-13'teki proteus devresini tasarıyoruz.



Şekil 13 - Dot Matris Uygulaması

Bu uygulamanın çalışması için gereken kodlar ise şöyledir;

```
#include <htc.h>
#include "delay.h"                                // Gecikme kütüphanesi

unsigned char F[]={0x7f,0x09,0x09,0x09,0x01};    // F
unsigned char I[]={0x00,0x41,0x7f,0x41,0x00};    // I
unsigned char R[]={0x7f,0x09,0x19,0x29,0x46};    // R
unsigned char A[]={0x7e,0x11,0x11,0x11,0x7e};    // A
unsigned char T[]={0x01,0x01,0x7f,0x01,0x01};    // T

void harf_goster(unsigned char harf[])
{
    PORTB=0x01;          // Birinci sütun seçiliyor
    PORTC=~harf[0];     // Harfin 5 sütunundan 1.'si gönderiliyor
    DelayMs(5);          // 5ms bekleniyor
    PORTB=0x02;          // Birinci sütun seçiliyor
    PORTC=~harf[1];     // Harfin 5 sütunundan 1.'si gönderiliyor
    DelayMs(5);          // 5ms bekleniyor
    PORTB=0x04;          // Birinci sütun seçiliyor
    PORTC=~harf[2];     // Harfin 5 sütunundan 1.'si gönderiliyor
    DelayMs(5);          // 5ms bekleniyor
    PORTB=0x08;          // Birinci sütun seçiliyor
    PORTC=~harf[3];     // Harfin 5 sütunundan 1.'si gönderiliyor
    DelayMs(5);          // 5ms bekleniyor
    PORTB=0x10;          // Birinci sütun seçiliyor
    PORTC=~harf[4];     // Harfin 5 sütunundan 1.'si gönderiliyor
    DelayMs(5);          // 5ms bekleniyor
}

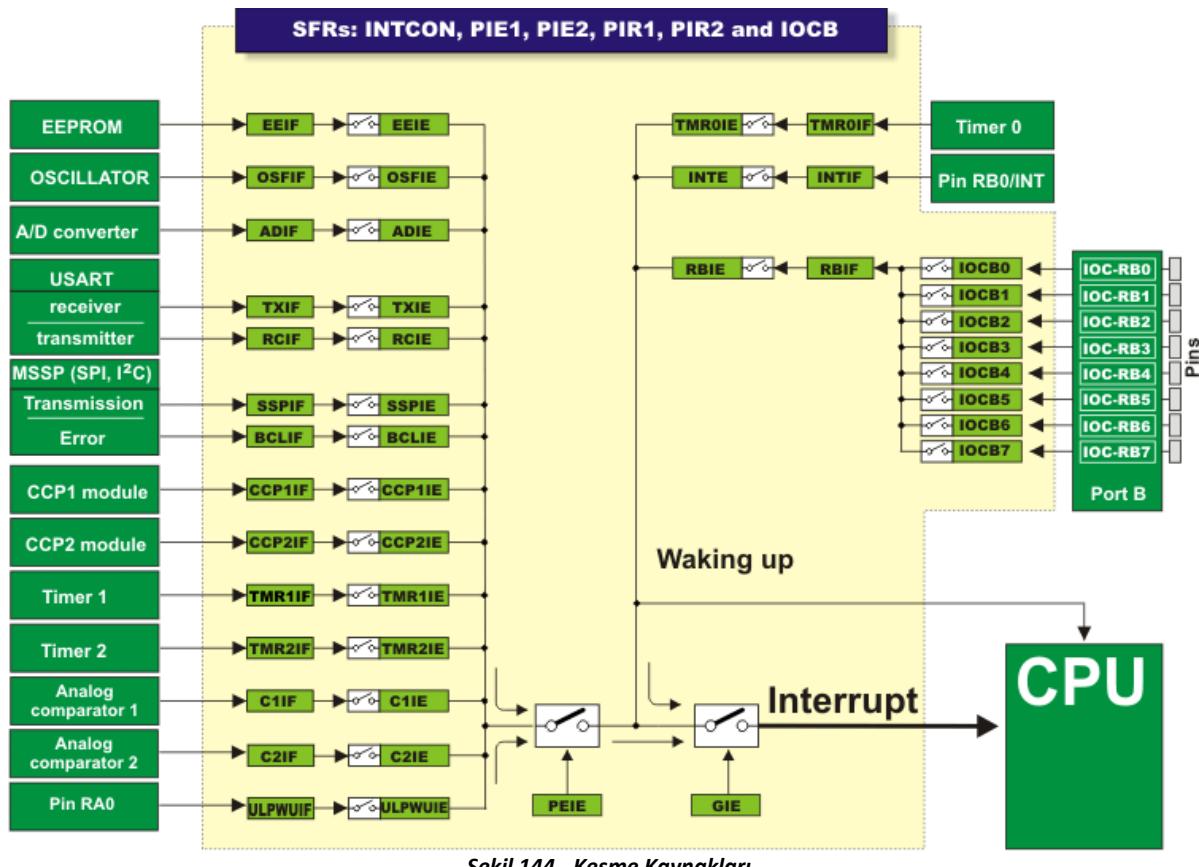
void main(void)           // Ana fonksiyon alanı
{
    int i;                // Herhangi bir değişken tanımlanıyor
    TRISB=0x00;           // PORTB çıkış olarak yönlendiriliyor
    TRISC=0x00;           // PORTC çıkış yapılıyor
    PORTB=0x00;           // PORTB'nin tüm çıkışları sıfırlanıyor
    PORTC=0x00;           // PORTC'nin tüm çıkışları sıfırlanıyor
    for(;;)               // Sonsuz döngüye giriliyor
    {
        for(i=0;i<20;i++) // Belirli süre tekrar ediliyor
            harf_goster(F); // F harfi gösteriliyor
        for(i=0;i<20;i++) // Belirli süre tekrar ediliyor
            harf_goster(I); // I harfi gösteriliyor
        for(i=0;i<20;i++) // Belirli süre tekrar ediliyor
            harf_goster(R); // R harfi gösteriliyor
        for(i=0;i<20;i++) // Belirli süre tekrar ediliyor
            harf_goster(A); // A harfi gösteriliyor
        for(i=0;i<20;i++) // Belirli süre tekrar ediliyor
            harf_goster(T); // T harfi gösteriliyor
    }
}
```

Buradaki harf değerleri internetten araştırılarak kolayca bulunabilir. İstenilen şekil ise sayılar değiştirilerek çıkartılabilir. Ayrıca fonksiyona dizi birimi göndermek de bu örnekte işlenmiş olup, kodların nasıl olduğu **comment** halinde yazılmıştır.

BÖLÜM 2 – INTERRUPT ve TIMER İŞLEMLERİ

2.1) Hi-Tech'te Interrupt İşlemleri

Interrupt ya da diğer adıyla kesme, bir çok işlemin olmazsa olmazlarındandır. Pic16f877'de 15'ten fazla kesme kaynağı bulunur. Kesme kaynaklarının ne olduğu şekil-14'te detaylı bir biçimde görülmektedir.



Şekil 144 - Kesme Kaynakları

Bu kadar çok kesme kaynağının olması Pic'in çok daha etkili bir biçimde kullanılmasına ve her bir işlemin kontrolünü kullanıcıya bırakmamasına sebebiyet veren büyük bir avantajdır.

Kesme birimlerini kısaca bir olaya benzetecek olursak;

Diyelim evde, bilgisayarınızda bir yazı yazıyorsunuz. O an telefon çaldı. Gidip telefona bakarsınız, telefonla işinizi hallettikten sonra tekrar yazınıza kaldığınız yerden devam edebilirsiniz. Veya yazı yazarken hem telefon hem de kapı aynı anda çaldı, yine önem sırasına göre bunlara cevap verip yazınıza kaldığınız yerden devam edebilirsiniz.

İşte kesmeler de aynı yukarıda anlatılan mantıkla çalışmaktadır. Telefon veya kapı çalması denilen olaylar her **bir birimin özel bayraklarına (flag)** yüklenir. Böylelikle bunlara bakılarak kesmenin oluşup oluşmadığı kontrol edilebilir.

Hi-Tech ortamında kesmeler oldukça kolay gerçekleştirilir. Temel olarak izlenecek yol şudur;

- Kesmesi izlenecek birimin flag'leri temizlenir.
- Kesmesi izlenecek birimin kesme izni verilir.
- Genel kesme izin birimleri aktif edilir.

- Kesme olmuş mu diye kontrol edilir.
- Eğer kesme olmuşsa, kesme oluşma durumunda işlenecek program çağrırlar ya da işletilir.
- Kesme durumunda işlenecek olay bittiğinde, kesme biriminin bayrağı sıfır yapılır.

Yukarıda madde madde anlatılan işlem basamakları, **tüm kesme birimleri** için geçerlidir.

Biz bu bölümde sadece **PORTB kanalındaki kesmelerle** ilgileneceğiz. Diğer kesmeler de aynı mantıkla çalıştığından, diğer kesme kaynaklarıyla ilgili bilgilere gelecek bölümlerde kısaca değineceğiz.

2.2) Dış Kesme Uygulaması

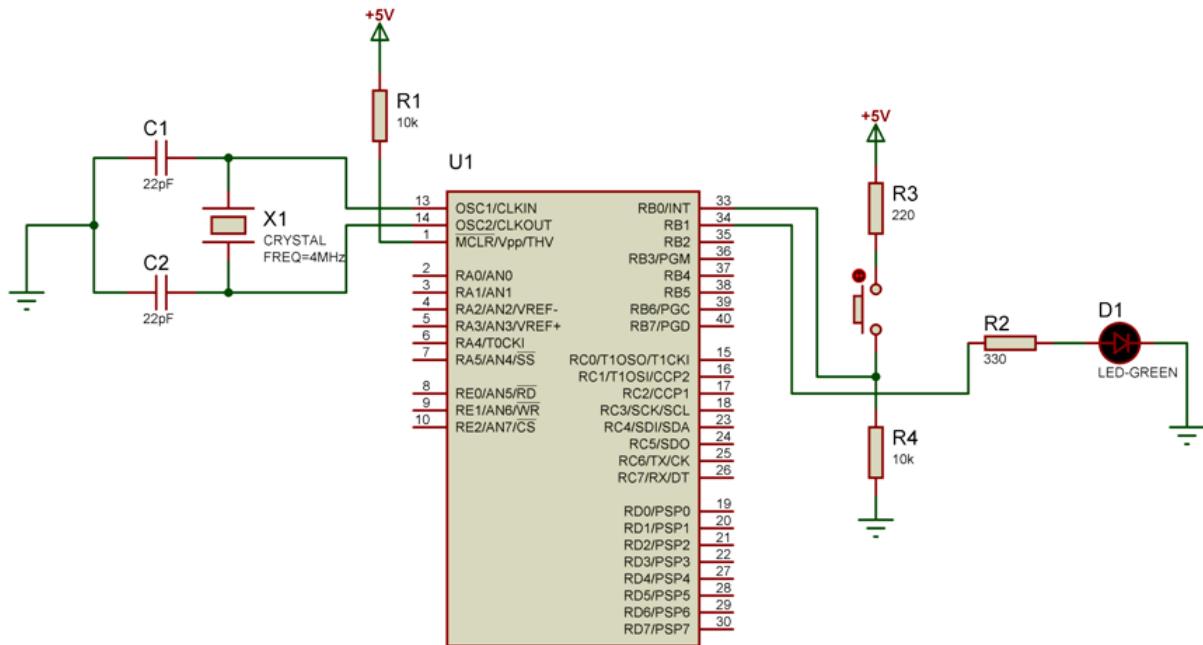
PORTB kanalında şekil-14'te de görüleceği üzere iki tür kesme kaynağı vardır. Bunlardan biri **RB0/INT kesmesi** iken diğeri **PORTB değişim kesmesidir**.

İlk deneyimizde RB0/INT dış kesmesini kullanarak led yakıp söndürme işlemleri yapacağız. RB0/INT dış kesmesini kontrol eden registerler ve görevleri şöyledir. Tüm bu değerler datasheet'ten bire bir alınmıştır.

INTE	: Dış kesme izin verme biti (1: izin verildi, 0: izin verilmedi)
INTF	: Dış kesme bayrak biti (1: kesme oluştu, 0: kesme oluşmadı)
INTEDG	: Dış kesme kenar seçme biti (1: yükselen kenar, 0: düşen kenar)
GIE	: Genel kesme izin biti

Dip not: Genel itibari ile kesme izin verme bitlerinin sonu IE ile biterken, kesme bayrak bitlerinin sonu IF ile biter.

Öncelikle Proteus'ta aşağıdaki devreyi çizelim. RB0 kesmesi yükselen kenarda meydana geldiğinde led yanacak, aynı kesme tekrar meydana geldiğinde ise led sönecektir.



Şekil 15 - Dış Kesme Uygulaması

İstediğimiz işlemi yerine getiren Hi-Tech programını aşağıdaki gibi yazarak yerine getirebiliriz.

```
#include <htc.h>

void main(void)           // Ana fonksiyon alanı
{
    TRISB=0x01;           // RB0/INT giriş, diğerleri çıkış
    PORTB=0x00;            // PORTB sıfırlanıyor

    INTF=0;                // RB0/INT kesme bayrağı temizleniyor
    INTEDG=1;               // Yükselen kenarda kesme oluşacak
    INTE=1;                 // RB0/INT izni veriliyor
    GIE=1;                  // Genel kesme izni veriliyor

    for(;;);               // İşlemci sonsuz döngüde bekletiliyor
}

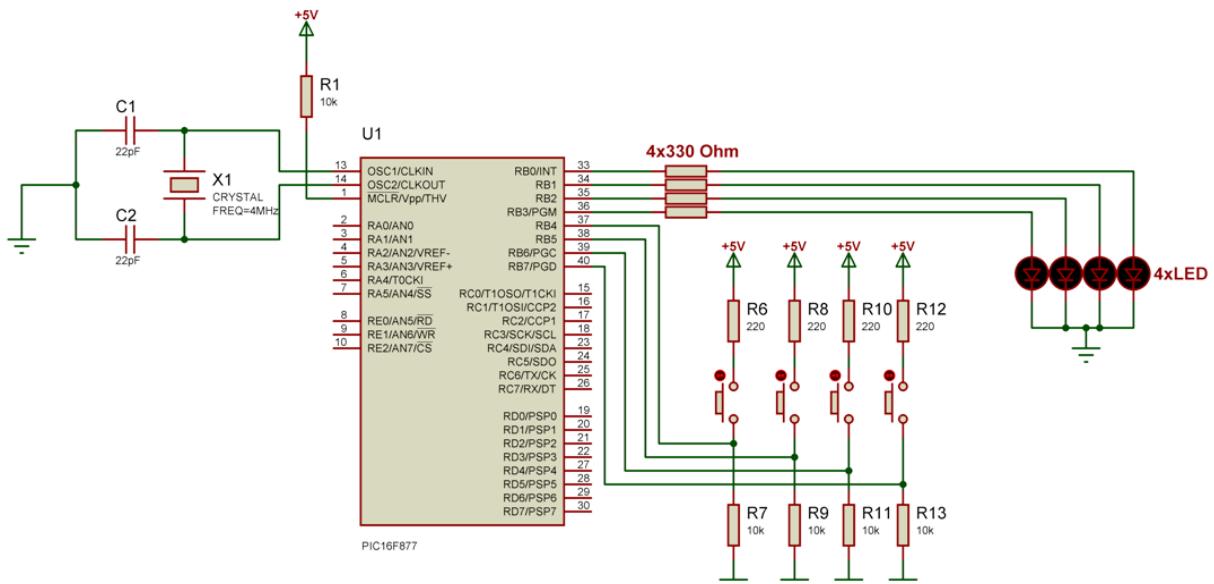
static void interrupt     // Kesme fonksiyonu
isim(void)                // Kesme fonksiyon ismi (önemsiz)
{
    char i;                // Kesme içinde bir değişken tanımlanıyor
    if(INTF)                // Dış kesme oluşmuş mu bakılıyor
    {
        i++;                  // Her bir kesmede değişken bir artırılıyor
        if(i==1)                // Değişken 1 ise
            RB1=1;              // Led yanıyor
        else if(i==2)           // Değişken 2 ise
        {
            RB1=0;                // Led söndürülüyor
            i=0;                  // Değişken sıfırlanıyor
        }
        INTF=0;                // Tekrar dış kesme alınabilmesi için kesme
    bayrağı temizleniyor
    }
}
```

Yukarıda da görüleceği üzere kesme fonksiyonu genel itibarı ile

```
static void interrupt
isim(void)
```

şeklindedir. Buradaki `isim(void)` yerine istenilen fonksiyon ismi yazılabilir. Bu önemsizdir. Yalnız şu unutulmamalıdır ki **kesme altında tanımlanan fonksiyonlara değer gönderilemez ve kesme fonksiyonu geriye değer döndüremez**. Bu kesme fonksiyonu yazarken en çok dikkat edilmesi gereken konudur. Ayrıca ana fonksiyon altında sonsuz döngüye girilen yerde istediğiniz işlemi yaptırmak mümkündür. Yalnızca kesme geldiğinde, kesme fonksiyonundaki işlemler yapılabacağından, ana fonksiyon altındaki işlemler bundan etkilenmeyecektir.

Bu uygulamadan sonra şimdi de RB4/RB7 kesmesine geçelim. **Buraya bağlayacağımız 4 adet buton ile hangi butona bastıysak o butona ait led'i yakacak** devreyi şekil-16'daki gibi kuralım.



Şekil 16 - RB4..RB7 Kesmesi

Şekildeki devreyi kurduktan sonra istediğimiz işlemi yapan kodu yazalım. RB4..RB7 kesmesini kontrol eden registerler aşağıda sıralanmıştır. Bu birimler datasheet'ten alınmıştır.

- | | |
|-------------|-----------------------------------------------------------------------|
| RBIE | : RB4..RB7 kesme izin verme biti (1: izin verildi, 0: izin verilmedi) |
| RBIF | : RB4..RB7 kesme bayrak biti (1: kesme oluştu, 0: kesme oluşmadı) |
| GIE | : Genel kesme izin biti |

Bu kaydediciler göz önüne alarak yazdığımız kodumuz aşağıdaki gibi olacaktır.

```
#include <htc.h>

void main(void)           // Ana fonksiyon alanı
{
    TRISB=0xF0;           // RB4..RB7 giriş, diğerleri çıkış
    PORTB=0x00;            // PORTB sıfırlanıyor

    RBIF=0;                // RB4..RB7 kesme bayrağı temizleniyor
    RBIE=1;                // RB4..RB7 kesme izni veriliyor
    GIE=1;                 // Genel kesme izni veriliyor

    for(;;);               // İşlemci sonsuz döngüde bekletiliyor
}

static void interrupt    // Kesme fonksiyonu
isim(void)                // Kesme fonksiyon ismi (önemsiz)
{
    char i;                // Kesme içinde bir değişken tanımlanıyor
    if(RBIF)                // RB4..RB7 kesme olmuş mu bakılıyor
    {
        if(RB4)              // ilk butona basıldıysa
            PORTB=0x01; // ilk ledi yak
        else if(RB5)          // ikinci butona basıldıysa
            PORTB=0x02; // ikinci ledi yak
        else if(RB6)          // Üçüncü butona basıldıysa
            PORTB=0x04; // Üçüncü ledi yak
        else if(RB7)          // Dördüncü butona basıldıysa
            PORTB=0x08; // Dördüncü ledi yak
    }
}
```

```

PORTB=PORTB;           // Son değişikliklerin algılanması için
PORTB'yi bir kez oku
RBIF=0;                // Tekrar dış kesme alınabilmesi için kesme
bayrağı temizleniyor
}
}

```

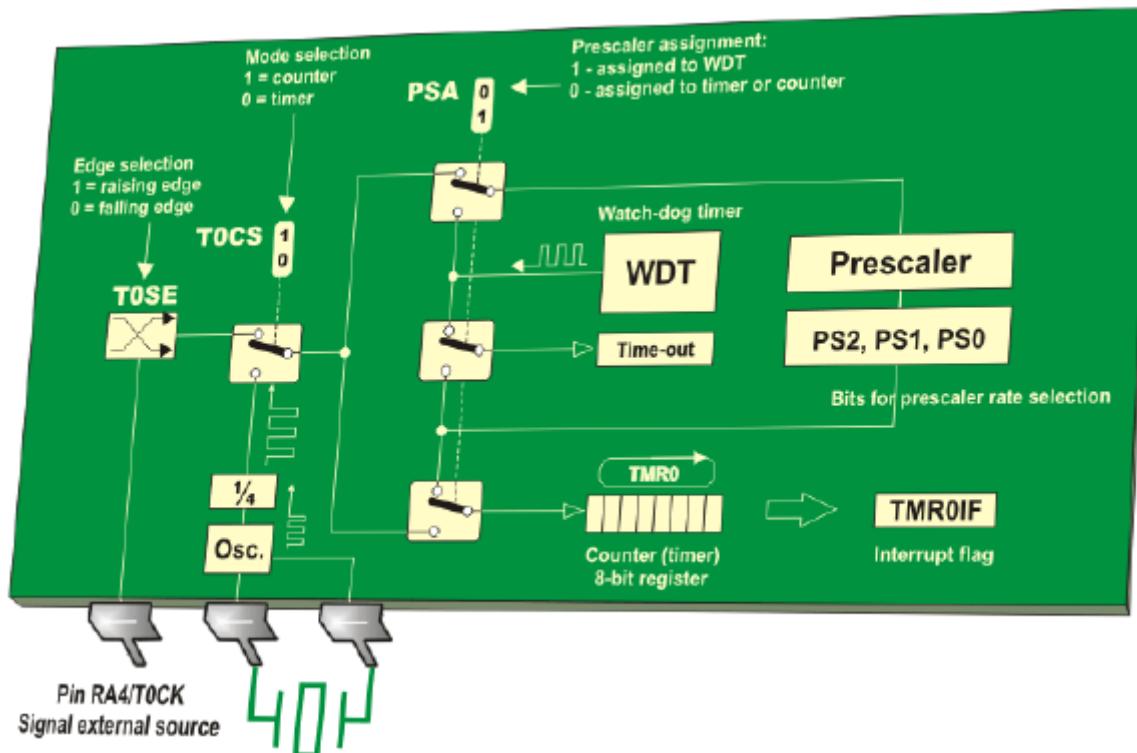
Yukarıdaki kodda ilginç olarak gözükebilecek tek yer PORTB=PORTB'dır. Fonksiyonun sürekli kesme içinde kalmaması için PORTB'deki son değişikliklerin okunması gerekmektedir. Bu sorunu da PORTB=PORTB şeklinde giderebiliriz.

Böylelikle kesme işlemlerinin nasıl olacağını gördük. Bundan sonra tüm kesmeleri aynı şekilde, önceliği de kedimiz belirleyerek oluşturabilirsiniz.

2.3) Hi-Tech'te Timer İşlemleri

Timer ya da sayıcılar pic'lerin içine yerleştirilmiş sayıma görevine yarayan birimlerdir. 16f877a'nın içinde Timer0, Timer1 ve Timer2 olmak üzere 3 timer birimi bulunmaktadır. Timer0 ve Timer2 birimleri 8 bitlik, Timer2 ise 16 bitlidir. Bu şekilde düşünüldüğünde Timer0 ve Timer2 ile 255'e kadar olan sayımlar Timer1 ile 65535'e kadar yapılabilir.

2.3.1) Timer 0 Birimi



Sekil 17 - Timer 0 Birimi

Timer 0 birimi 8 bitlik bir birim olup dışarıdan ve dahili osilatörden saat sinyali alabilir, 255'e kadar sayımla yapabilir, 255'ten 0'a dönerken TMR0IF kesme bayrağını set eder.

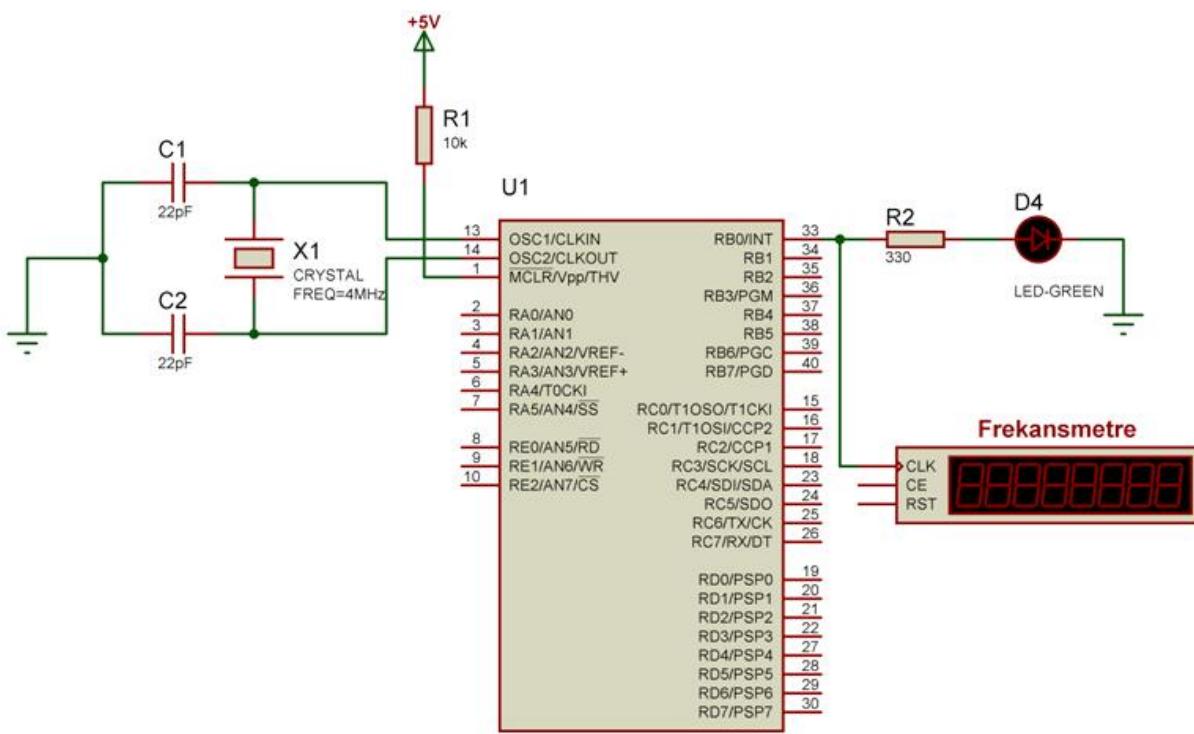
Timer 0'ı kontrol eden kaydediciler ve görevleri şöyledir;

T0CS	: Timer0 sinyal seçim biti (0: Dahili, 1: Harici)
TOSE	: Timer0 kenar seçim biti (0: Düşen kenar, 1: Yükselen kenar)
PSA	: Frekans bölücü seçim biti (0: Prescaler Timer0 için, 1: Prescaler WDT için)
PS2, PS1, PS0	: Bölüm oranını belirleyen bitler, bu değerler şekil-18'den görülebilir
TMROIE	: Timer0 kesme izin biti
TMROIF	: Timer0 kesme bayrak biti

PS2	PS1	PS0	TMR0	WDT
0	0	0	1:2	1:1
0	0	1	1:4	1:2
0	1	0	1:8	1:4
0	1	1	1:16	1:8
1	0	0	1:32	1:16
1	0	1	1:64	1:32
1	1	0	1:128	1:64
1	1	1	1:256	1:128

Şekil 18 - Prescaler değerleri

Timer0 uygulamamızda her yarımsaniyede bir led'in yanıp sönmesini sağlayalım. Öncelikle şekil-19'daki Proteus devresini çizelim ve hesaplamamızı yapalım.



Şekil 19 - Timer 0 Uygulaması - 1

Sinyalin kristalden (4MHz) geldiğini ve pic'in bu hızı 4'e otomatikman böldüğünü düşünürsek her saat darbesi 1 mikro saniyede bir gerçekleşir. Biz ise 500ms=500000us gecikme gerekiyor. O halde 500.000 kere saydırımız gerekmekte fakat Timer 0 ile en fazla 255'e

kadar saydırabiliriz. 250 birim saydırduğumuzu düşünürsek (başta o değere TMR0 birimine 50 yüklersek);

$500.000/250=2000$ yapar. Prescaler değerini 1:16 seçersek $2000:16=125$ yapar. O halde bir değişkeni 125'e kadar saydıracağız ve değişken bu değere ulaştığında istediğimiz sinyale bize verecek. Bu işlemi yapan C kodu aşağıdaki gibi olur.

```
#include <htc.h>

void main(void)           // Ana fonksiyon alanı
{
    TRISB=0x00;           // PORTB çıkış olarak ayarlanıyor
    PORTB=0x00;           // PORTB sıfırlanıyor

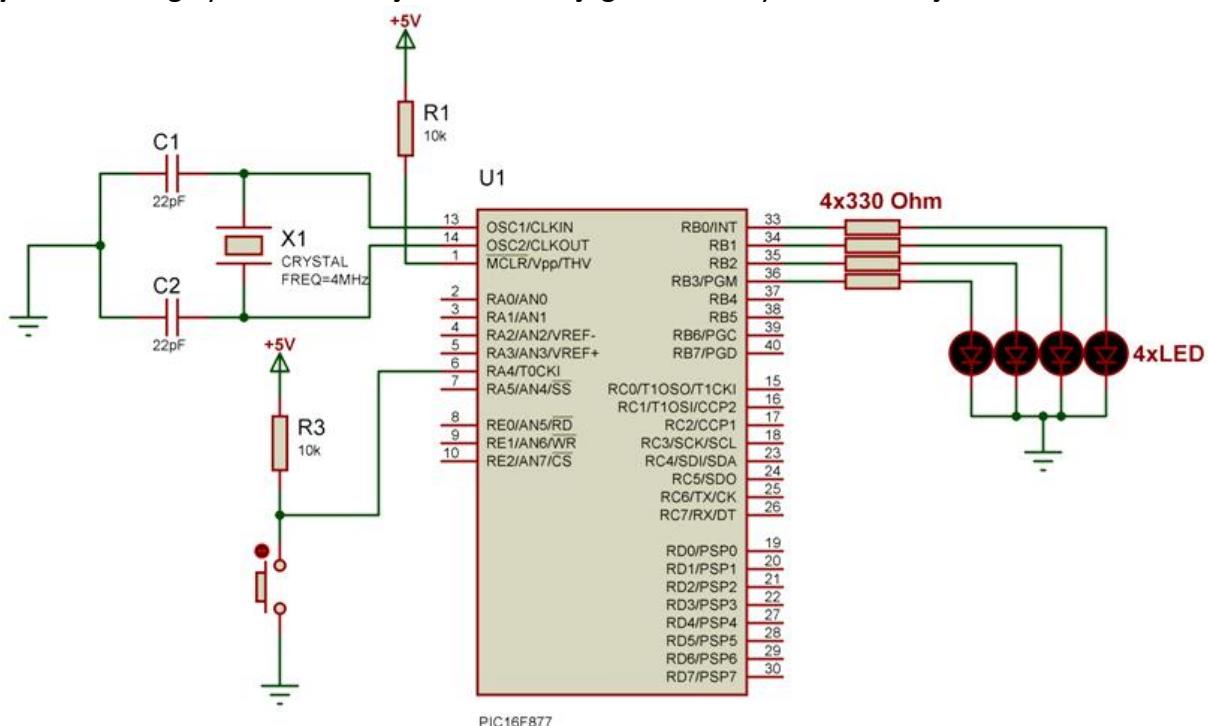
    TMR0=55;              // TMR0=55 oluyor
    TOCS=0;                // Dahili osilatör
    PSA=0;                 // Prescaler TMR0 için
    PS2=0;                 // 1:16 bölüm değeri
    PS1=1;
    PS0=1;
    TOIF=0;               // TMR0 kesme bayrağı temizleniyor
    TOIE=1;                // TMR0 kesmesine izin veriliyor
    GIE=1;                 // Genel kesme izni veriliyor

    for(;;);              // İşlemci sonsuz döngüde bekletiliyor
}

static void interrupt    // Kesme fonksiyonu
isim(void)                // Kesme fonksiyon ismi (önemsiz)
{
    char i;                // 125'e kadar sayacak değişken
    if(TMR0IF)              // TMR0 kesmesi oluşmuş mu
    {
        i++;
        if(i>0 & i<125)    // 0 ile 125 arası 1 ol
        {
            RB0=1;
        }
        else if(125<i && i<250) // 125 ile 250 arası 0 ol
        {
            RB0=0;
        }
        else if(i>250)      // 250'yi aşarsa değişken 0 olsun
        {
            i=0;
        }
        TMR0=55;             // 255-55=200 birim sayacak
        TMR0IF=0;             // Tekrar dış kesme alınabilmesi için kesme
        bayrağı temizleniyor
    }
}
```

Göründüğü üzere Timer 0 kesmesini bu yolla kullanarak oldukça kolay sinyal elde ettik. Esasen 500ms'lik zaman dilimi 1 ve 0 olması aslında sinyalimizin 1Hz'lik olduğunu gösterir, bunu da proteus dosyamıza koyduğumuz frekansmetre ile görebiliriz. Yine de oluşturduğumuz sinyal gürültüler ve sıcaklık gibi dış etkenler yüzünden değişimden ürettiğimiz sinyal tam anlamıyla verimli olmayabilir bu yüzden genel itibarı ile saat devrelerinde kullanılmaması tavsiye edilir.

Şimdi de Timer 0 için dış kaynağı kullanarak, 5 kere butona bastığımızda 4 ledin sırasıyla yanmasını sağlayalım. Bunun için öncelikle aşağıdaki devreyi Proteus'ta çizelim.



Sekil 150 - Timer 0 Uygulaması - 2

Gördüğünüz gibi buton yapımızı düşen kenara göre seçtim. Bunu göz önüne alarak yazağımız C kodu aşağıdaki gibi olacaktır.

```
#include <htc.h>

void main(void)          // Ana fonksiyon alanı
{
    ADCON1=0x07;          // PORTA dijital yapılmıyor
    TRISA=0x10;           // RA4 giriş yapılmıyor
    TRISB=0x00;           // PORTB çıkış olarak ayarlanıyor
    PORTA=0x00;            // PORTA sıfırlanıyor
    PORTB=0x00;            // PORTB sıfırlanıyor

    TMR0=-5;              // Esasen 250 yüklenmiş oluyor
    T0SE=0;                // Düşen kenar tetikleme
    T0CS=1;                // Harici osilatör
    PSA=1;                 // Prescaler TMR0 için
    PS2=0;                 // 1:1 bölüm değeri
    PS1=0;
    PS0=0;
    T0IF=0;                // TMR0 kesme bayrağı temizleniyor
    T0IE=1;                // TMR0 kesmesine izin veriliyor
    GIE=1;                 // Genel kesme izni veriliyor

    for(;;);               // İşlemci sonsuz döngüde bekletiliyor
}

static void interrupt    // Kesme fonksiyonu
isim(void)               // Kesme fonksiyon ismi (önemsiz)
{
    char i;                // Değişkenler tanımlanıyor
    if(TMR0IF)              // TMR0 kesmesi oluşmuş mu
        // Kodlar buraya yazılacak
}
```

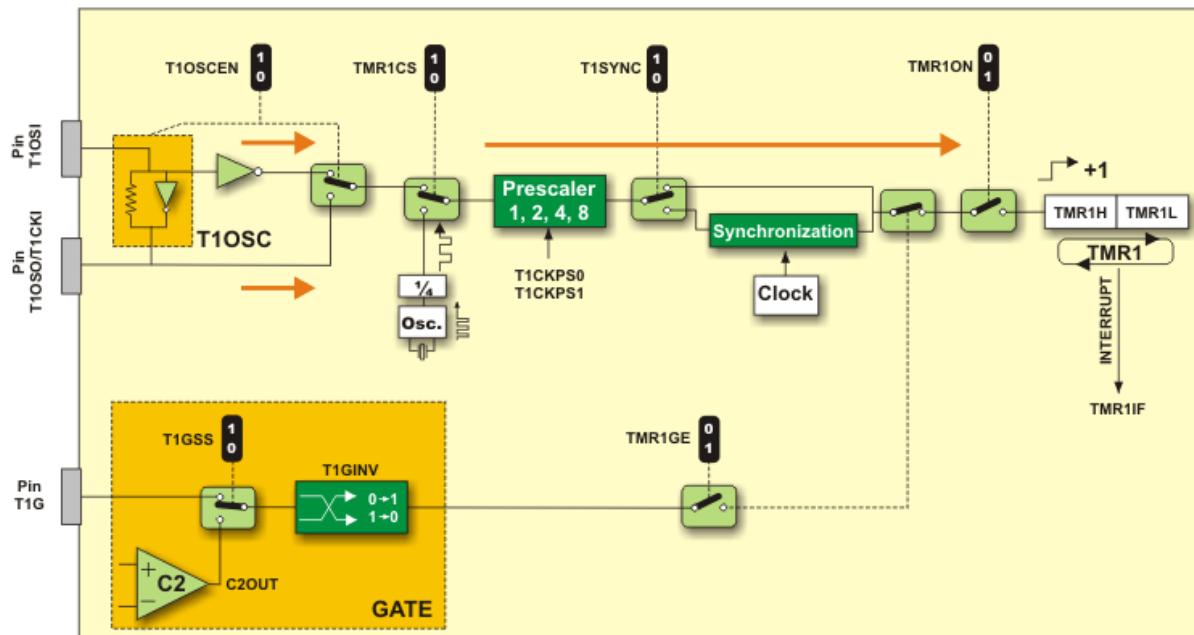
```

{
    i++;
    // PORTB'de gözükecek değer
    if(i==16) // Değişken 16 olursa onu tekrar 0 yap
        i=0;
    PORTB=i; // Değişken değerini PORTB'ye yansıtılıyor
    TMRO=-5; // Esasen 250 yüklenmiş oluyor
    TMROIF=0; // Tekrar dış kesme alınabilmesi için kesme
    bayrağı temizleniyor
}
}

```

Göründüğü üzere dışarıdan elle verdığımız sinyalde dahi pic'in kolayca çalıştığı gözükmemekte. Ayrıca Timer değişkenlerine negatif(-) atamaların yapılabileceği de görülebilir.

2.3.2) Timer 1 Birimi



Sekil 161 - Timer1 Birimi

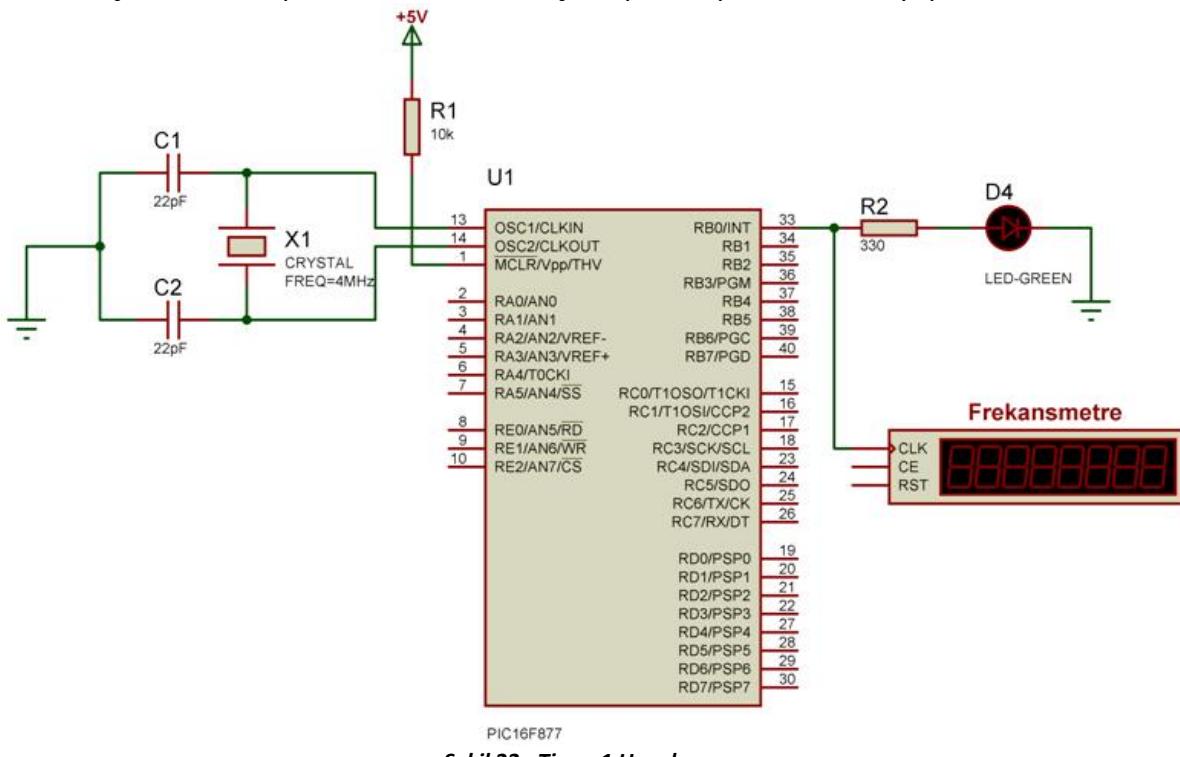
Şekil-21'de gözüken timer birimi oldukça fonksiyonel bir yapıdır. 16bitlik olması 0'dan 65535'e kadar sayımla yapabilemeye olanak sağlar. Dahili ve harici saat girişi seçimiyle istenilen uygulamalarda kullanılabilir.

Timer 1'i kontrol eden kaydediciler ve görevleri şöyledir;

- T1CKPS1, T1CKPS0** : Prescalar değeridir, (1:1, 1:2, 1:4, 1:8)
- T1OSCEN** : Timer1 osilatör kontrol biti (0: Kapalı, 1: Açık)
- TMR1CS** : Saat kaynağı seçme biti (0: Dahili, 1: Harici)
- T1SYNC** : Senkronizasyon biti (1: Senk. Yok, 0: Senk. Var)
- TMR1ON** : Timer1 açma kapama biti (0: Kapalı, 1: Açık)
- TMR1H** : Timer1 sayıcısının yüksek değerlikli bitini tutan kaydedici
- TMR1L** : Timer1 sayıcısının düşük değerlikli bitini tutan kaydedici
- TMR1IE** : Timer1 kesme izin biti
- TMR1IF** : Timer1 kesme bayrak biti

Timer 1'e özel dış kaynaklı saat frekansı alınmak istenirse gerekli pinlere düşük güçlü kristaller bağlanır. Bu kristal değerlerinden en ideal 32Khz olanıdır ve 33pf ile sürürlür.

Şimdiki uygulamamızda Timer 1 ile **bir saniyede bir yanıp sönen led uygulaması** yapalım. Öncelikle şekil-22'deki proteus devresini oluşturup, hesaplamalarımıza yapalım.



Şekil 22 - Timer 1 Uygulaması

Şimdi hesaplamamızı yapacak olursak. 4Mhz'lik kristal kullandığımızı ve pic'in bunu 4'e bölgerek kullandığını düşünürsek bize 1mikrosaniyelik bir gecikme yaratacaktır. 1.000.000 sayım için Timer 1'in öncelikle 50.000'e kadar saydıracağımızı düşünürsek;

$1.000.000/50.000=20$ değeri çıkar. Prescaler oranını 1:4 seçersek her 5 kesmede bir değer artırımı bize istediğimiz gecikme olan 1 saniyeyi sağlayacaktır. Bunu yapan C kodu aşağıda görülmektedir.

```
#include <htc.h>

void main(void)           // Ana fonksiyon alanı
{
    TRISB=0x00;           // PORTB çıkış olarak ayarlanıyor
    PORTB=0x00;             // PORTB sıfırlanıyor

    TMR1H=-50000/256;     // TMR1'e 65536-50000 yükleniyor.
    TMR1L=-50000%256;
    TMR1CS=0;              // Dahili osilatör
    T1CKPS1=1;             // Prescaler 1:4 oluyor
    T1CKPS0=0;
    T1SYNC=1;               // Senkronizasyon yok
    TMR1IF=0;                // TMR1 kesme bayrağı temizleniyor
    TMR1IE=1;                // TMR1 kesmesine izin veriliyor
    TMR1ON=1;                // TMR1 çalıştırılıyor
    PEIE=1;                  // Yardımcı kesme izni veriliyor
    GIE=1;                   // Genel kesme izni veriliyor
```

```

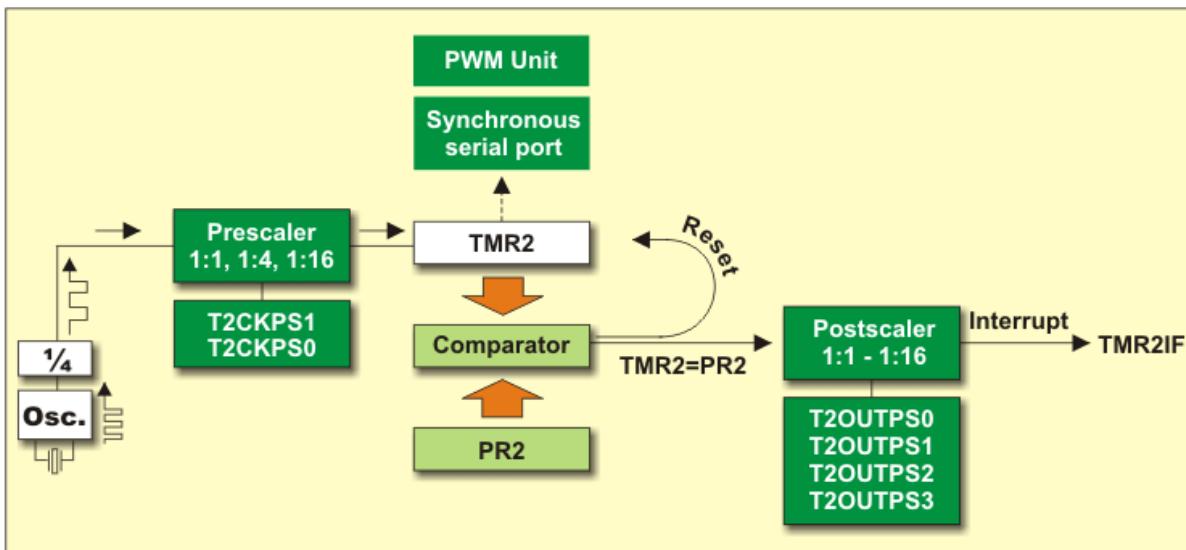
        for(;;);           // İşlemci sonsuz döngüde bekletiliyor
    }

static void interrupt  // Kesme fonksiyonu
isim(void)            // Kesme fonksiyon ismi (önemsiz)
{
    char i;           // Değişkenler tanımlanıyor
    if(TMR1IF)         // TMR1 kesmesi oluşmuş mu
    {
        i++;          // Değişken 1 artırılıyor
        if(i<5)       // Değişken 5 olursa led yansın
        {
            RB0=1;
        }
        else if(i>5) // Değişken 5'ten büyük olursa led sönsün
        {
            RB0=0;
        }
        if(i==10)     // 2 saniye olduğunda değişken sıfırlansın
            i=0;
        TMR1H=-50000/256; // TMR1'e 65536-50000 yükleniyor.
        TMR1L=-50000%256;
        TMR1IF=0;      // Tekrar dış kesme alınabilmesi için kesme
bayrağı temizleniyor
    }
}

```

Kodlarda görüldüğü üzere -50000 değeri TMR1 kaydedicisine aslında 15535 değeri yüklemektedir. PEIE ise yardımcı kesme izin bitidir.

2.3.3) Timer 2 Birimi



Şekil 23 - Timer 2 Birimi

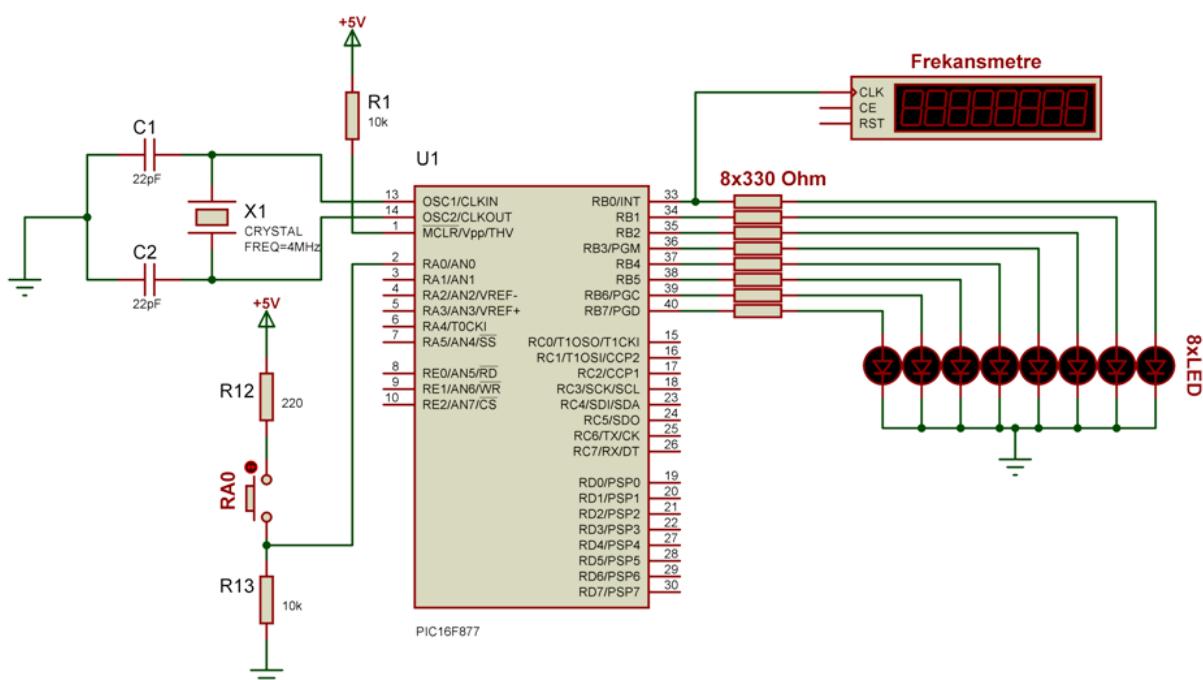
Timer 2 birimi 16f877a'da bir çok görevde kullanılan özel bir Timer'dır. Özellikle kullanım amacı PWM üretmek olan Timer 2, Timer 0 gibi 8 bitliktir. Bölme değerleri oldukça genişdir. Timer 2'yi kontrol eden kaydediciler aşağıda sıralanmıştır;

- TMR2=PR2** : Timer2 sayıcısının yüksek değerlikli bitini tutan kaydedici
TOUTPS3, TOUTPS2, TOUTPS1, TOUTPS0 : Postscale değerler, şekil-24'e bakınız.
TMR2ON : Timer2 sayıcısını açma biti
T2CKPS1, T2CKPS0 : Prescaler değerler (1:1, 1:4, 1:16)
TMR2IE : Timer2 kesme izin biti
TMR2IF : Timer2 kesme bayrak biti

TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	Prescaler Rate
0	0	0	0	1:1
0	0	0	1	1:2
0	0	1	0	1:3
0	0	1	1	1:4
0	1	0	0	1:5
0	1	0	1	1:6
0	1	1	0	1:7
0	1	1	1	1:8
1	0	0	0	1:9
1	0	0	1	1:10
1	0	1	0	1:11
1	0	1	1	1:12
1	1	0	0	1:13
1	1	0	1	1:14
1	1	1	0	1:15
1	1	1	1	1:16

Şekil 24 - Prescale Değerler

Timer 2 birimindeki uygulamamızda **Timer2 kesmesi ile bir değişkeni artırıp, değişken değerinin PORTB'ye bağlı ledlere istediğimiz zaman yansımاسını** yapacağız. Bunun için öncelikle aşağıdaki devreyi Proteus'ta kuruyoruz.



Şekil 25 - Timer 2 Uygulaması

Timer 2 sayıcısı için Poscale'i 1:16, Postscaler'i 1:16, PR2'yi de 250 yapıyoruz. Bu değerleri sağladığımızda yaklaşık her 0.064 saniyede bir kesme elde ederiz. Butona bastığımızda ise anlık sayılm değerini ledlerde gözlemlayabilir, butona basılı tuttuğumuzda ise değişim anını gözleyebiliriz. Bu işlemi yapan C kodu aşağıdadır.

```
#include <htc.h>

char i; // Genel değişken tanımlanıyor

void main(void) // Ana fonksiyon alanı
{
    ADCON1=0x07; // PORTA dijital yapılıyor
    TRISA=0x01; // RA0 giriş
    TRISB=0x00; // PORTB çıkış olarak ayarlanıyor
    PORTA=0x00; // PORTA sıfırlanıyor
    PORTB=0x00; // PORTB sıfırlanıyor

    PR2=250; // PR2 değerine 250 yükleniyor
    T2CKPS1=1; // Prescaler 1:16 oluyor
    T2CKPS0=1; // Prescale 1:16 oluyor
    TOUTPS3=1;
    TOUTPS2=1;
    TOUTPS1=1;
    TOUTPS0=1;
    TMR2IF=0; // TMR1 kesme bayrağı temizleniyor
    TMR2IE=1; // TMR1 kesmesine izin veriliyor
    TMR2ON=1; // TMR1 çalıştırılıyor
    PEIE=1; // Yardımcı kesme izni veriliyor
    GIE=1; // Genel kesme izni veriliyor

    for(;;)
    {
        if(RA0) // Butona basıldı mı
            PORTB=i; // Değişkenin değeri PORTB'ye yansıtılıyor
    }
}

static void interrupt isim(void) // Kesme fonksiyonu
{ // Kesme fonksiyon ismi (önemsiz)
    if(TMR2IF) // TMR2 kesmesi olmuş mu
    {
        i++; // Değişken 1 artırılıyor
        TMR2IF=0; // Tekrar dış kesme alınabilmesi için kesme
        bayrağı temizleniyor
    }
}
```

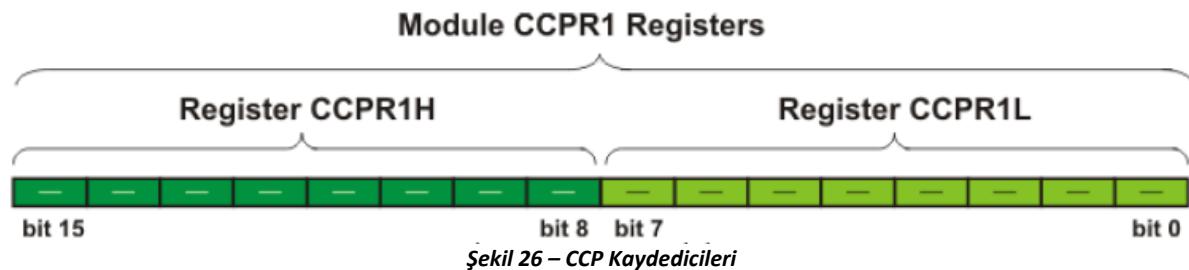
Gördüğü üzere Timer 2 birimini de kullanmak Hi-Tech altında oldukça kolay.

Bu bölümde Interrupt ve Timer işlemlerini irdeledik, özellikle register (kaydedici) birimlerin bitlerinin açık halde kullanılması, kodları uzatıyor gibi gözüke de değerleri karıştırmamak adına oldukça faydalıdır.

BÖLÜM 3 – CAPTURE/COMPARE/PWM ve TUŞ TAKIMI İŞLEMLERİ

3.1) Hi-Tech'te CCP İşlemleri

Pic içerisinde dahili olarak bulunan CCP modülü Capture (yakalama), Compare (karşılaştırma) ve PWM bölümlerinin baş harflerinin bir araya gelmesinden oluşur. Pic 16f877a'nın içinde 2 adet CCP modülü bulunmaktadır. CCP için kullanılan kaydedici şekil-26'da görüleceği üzere 16 bitlidir. CCPRx kaydedicisi Timer 1'in kaydedicisi ile ortak çalışarak Capture ve Compare işlemlerinde kullanılabilir.



Genel itibarı ile CCP işlemleri CCPxCON registeriyle kontrol edilir. x ifadesi bu bölümde kullanılan birime göre 1 ya da 2 olabilir. CCPxCON registerinin yapısı ve görevi şekil-27'de gözükmektedir.

CCP1CON REGISTER/CCP2CON REGISTER (ADDRESS 17h/1Dh)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0	bit 0

Şekil 27 – CCPxCON Kaydedicisi

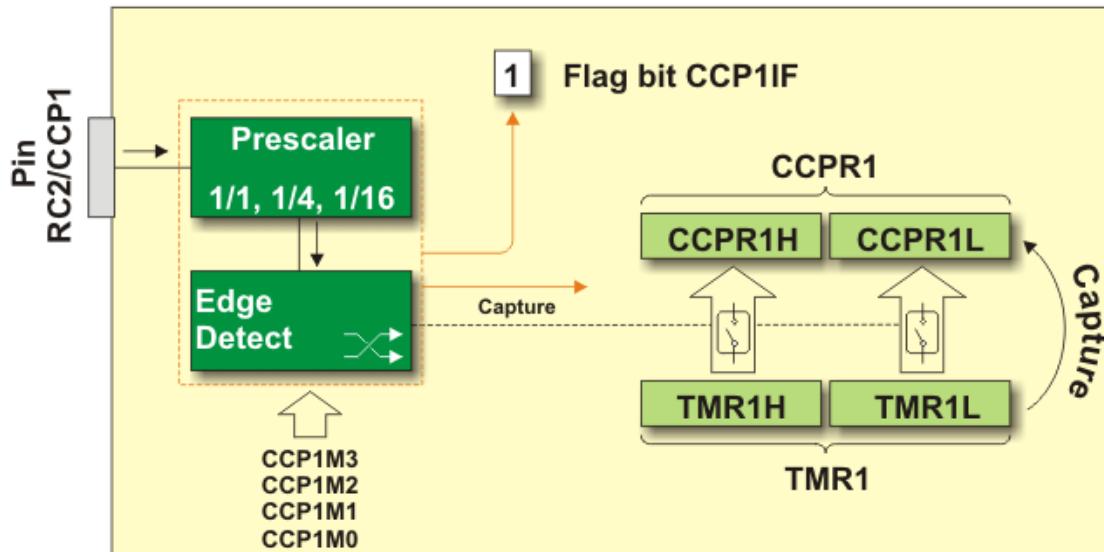
Şekil-27'de verilen kaydedicinin bit bit görevleri ise şöyledir;

CCPxX, CCPxY	: PWM düşük değerlikli bitleri. Yüksek değerlikli 8 bit ise CCPRxL kaydedicisinde bulunur.
CCPxM3, CCPxM2, CCPxM1, CCPxM0	: CCP mod seçme bitleri. Modların ne olacakları aşağıdaki tabloda gösterilmiştir.

- 0000 : CCP etkin değil
 - 0100 : Capture modu, her düşen kenarda
 - 0101 : Capture modu, her yükselen kenarda
 - 0110 : Capture modu, her 4. yükselen kenarda
 - 0111 : Capture modu, her 16. yükselen kenarda
 - 1000 : Compare modu, denklik durumunda CCPx pini 1 olsun, CCPxF bayrağı çekilsin
 - 1001 : Compare modu, denklik durumunda CCPx pini 0 olsun, CCPxF bayrağı çekilsin
 - 1010 : Compare modu, denklik durumunda CCPx pini değişmesin, CCPxF bayrağı çekilsin
 - 1011 : Compare modu, denklik durumunda CCPx pini değişmesin, CCPxF bayrağı çekilsin, timer 1 resetlensin ve ADC başlasın
 - 11xx : PWM modu

3.1.1) Capture Modu

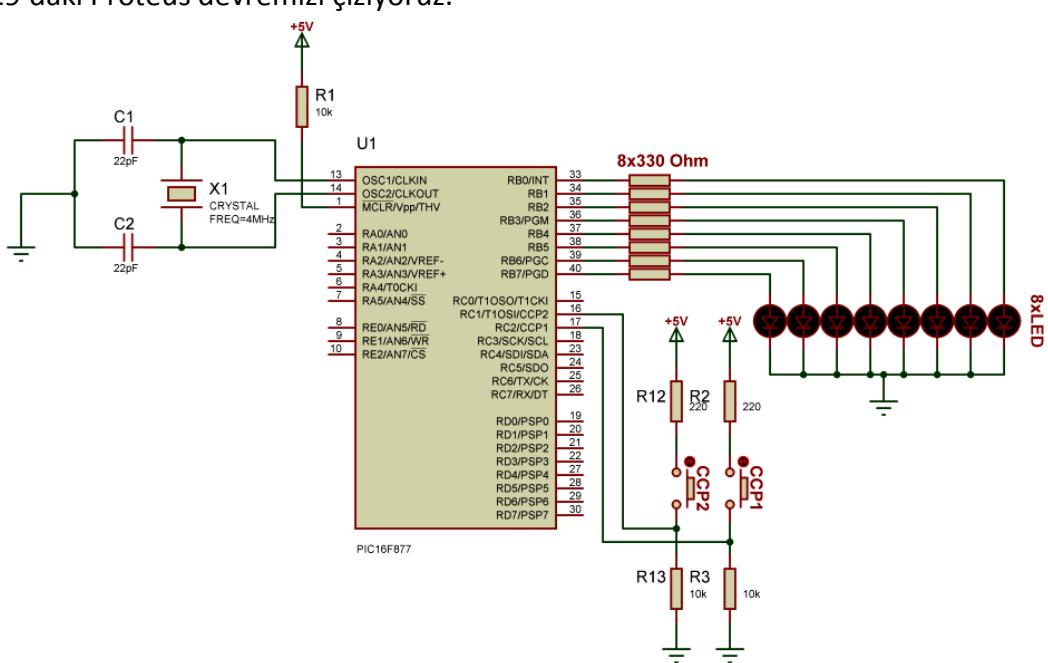
Pic içerisinde bulunan Capture birimi şekil-28'den de görüleceği üzere Timer 1 ile ortak çalışır. Capture biriminin görevi yakalama yapmaktadır. Modlarına göre **her düşen, yükselen, 4. veya 16. yükselen kenarda yakalama yaparak ne kadar süre geçtiyse, geçen süreyi Timer 1 zamanlayıcısından alarak CCP biriminin özel kaydedicilerine yükler.** Daha sonra CCP birimine kaydedilen zamanlar okunarak gerekli işlemler yapılır.



Şekil 28 – Capture Bölümü

Capture birimini kullanmak için öncelikle Timer 1 ve CCP birimi ayarları yapılır. Kesmeden yararlanılacaksa CCPxIE ve CCPxIF birimleri set edilir veya temizlenir.

Capture uygulamamızda CCP1'in her yükselen kenar yakalamasında bir değişkeni artırması, CCP2'nin ise her 4. yükselen kenar yakalamasında yine aynı değişkeni bir azaltması ve bu değişkenin PORTB'den görülmesi amaçlanmıştır. Öncelikle uygulamamızla ilgili aşağıdaki şekil-29'daki Proteus devremizi çiziyoruz.



Şekil 29 – Capture Uygulaması

Devremizi kurduktan sonra Hi-Tech'te kodumuzu yazalım.

```
#include <htc.h>

char i; // Genel değişken tanımlanıyor

void main(void) // Ana fonksiyon alanı
{
    TRISB=0x00; // PORTB çıkış olarak ayarlanıyor
    TRISC=0x06; // CCP1 ve CCP2 giriş
    PORTB=0x00; // PORTB sıfırlanıyor
    PORTC=0x00; // PORTC sıfırlanıyor

    CCP1M0=1; // CCP1 her yükselen kenar modunda
    CCP1M1=0;
    CCP1M2=1;
    CCP1M3=0;

    CCP2M0=0; // CCP1 her 4. yükselen kenar modunda
    CCP2M1=1;
    CCP2M2=1;
    CCP2M3=0;

    CCP1IF=0; // CCP1 ve CCP2 kesme bayrakları temizleniyor
    CCP2IF=0;
    CCP1IE=1; // CCP1 ve CCP2 kesme izinleri veriliyor
    CCP2IE=1;
    PEIE=1; // Yardımcı kesme izni veriliyor
    GIE=1; // Genel kesme izni veriliyor

    for(;;);
}

static void interrupt isim(void) // Kesme fonksiyonu
{ // Kesme fonksiyon ismi (önemsiz)
    if(CCP1IF) // CCP1 kesmesi varsa
    {
        i++; // Değişken bir artırılıyor
        PORTB=i; // Değişken değeri PORTB'ye yansıtılıyor

        CCP1IF=0; // Yeni CCP1 kesmesi için bayrak temizleniyor
    }

    if(CCP2IF) // CCP2 kesmesi varsa
    {
        i--; // Değişken bir azaltılıyor
        PORTB=i; // Değişken değeri PORTB'ye yansıtılıyor

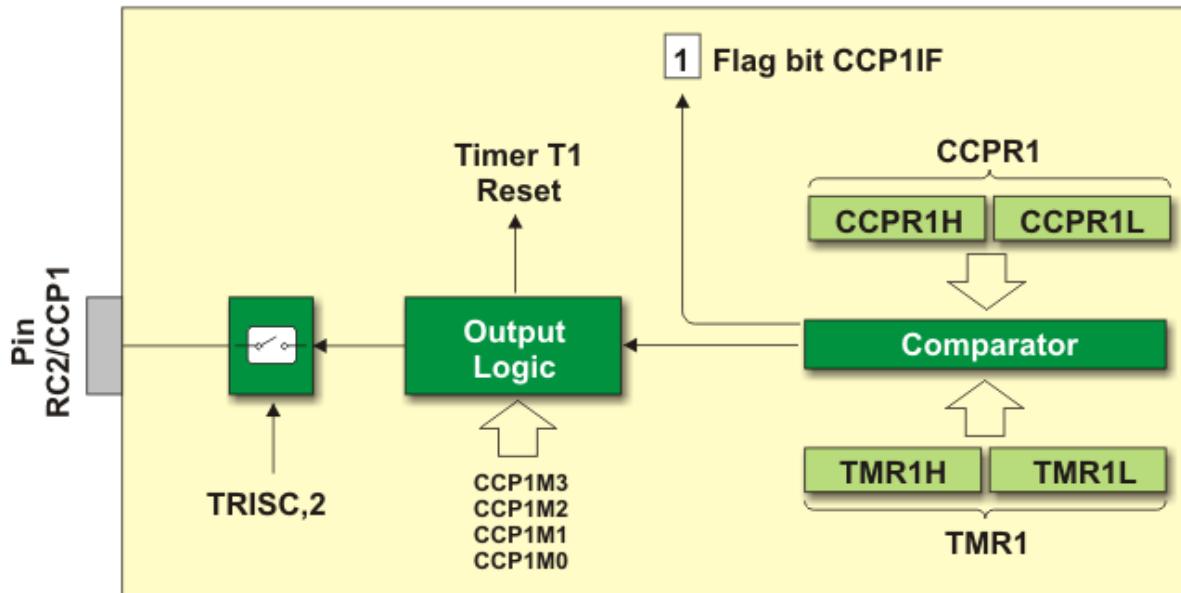
        CCP2IF=0; // Yeni CCP2 kesmesi için bayrak temizleniyor
    }
}
```

Yukarıda görüldüğü gibi genel kesme fonksiyonun içinde birden çok kesme bayrağı kontrolü yapılabilir.

3.1.2) Compare Modu

Compare modu pic içerisinde karşılaştırma işlemlerinde kullanılan birimdir. Şekil-30'dan da görüleceği üzere Timer1 ile ortaklaşa çalışırlar. Genel itibarıyle çalışma mantığı şöyledir;

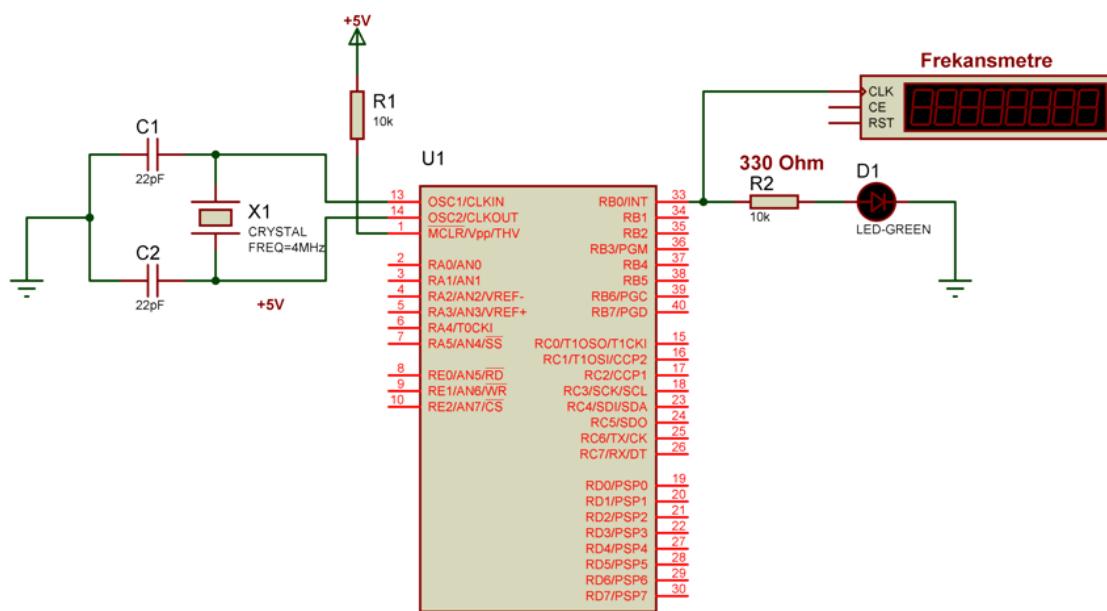
- Öncelikle Timer1 ve Compare modu ayarları yapılır,
 - CCPRxH ve CCPRxL kaydedicilerine yakalanmak istenen sayı yüklenir,
 - Genellikle Timer1 harici kaynak beslemesi kullanılır
 - TMR1 kaydedicisi ile CCPRx kaydedicisi birbirine eşit olduğunda kesme bayrağı set edilir.



Şekil 30 – Compare Birimi

Compare uygulamamızda CCP1 birimine 500 yüklenerek dahili Timer1 birimi değeri sayılarak CCP1 eşitliğinde RB0'a bağlı ledin yanıp sönmesi sağlanacaktır.

Bu uygulama için öncelikle şekil-31'deki devreyi çiziyoruz.



Şekil 31 – Compare Uygulaması

Şekildeki devreyi çalıştırın C kodu ise aşağıdaki gibidir;

```
#include <htc.h>

void main(void)           // Ana fonksiyon alanı
{
    TRISB=0x00;           // PORTB çıkış olarak ayarlanıyor
    PORTB=0x00;            // PORTB sıfırlanıyor

    TMR1CS=0;              // Timer1 Harici kaynaktan besleniyor
    T1SYNC=0;                // Senkronizasyon yok
    TMR1ON=1;                // Timer1 açılıyor

    CCP1M0=1;                // CCP1 compare modunda, timer1 resetlenecek
    CCP1M1=1;
    CCP1M2=0;
    CCP1M3=1;

    CCPR1H=500/256;        // 500'e eşitlenecek
    CCPR1L=500%256;

    CCP1IF=0;                // CCP1 ve CCP2 kesme bayrakları temizleniyor
    CCP1IE=1;                // CCP1 ve CCP2 kesme izinleri veriliyor
    PEIE=1;                  // Yardımcı kesme izni veriliyor
    GIE=1;                   // Genel kesme izni veriliyor

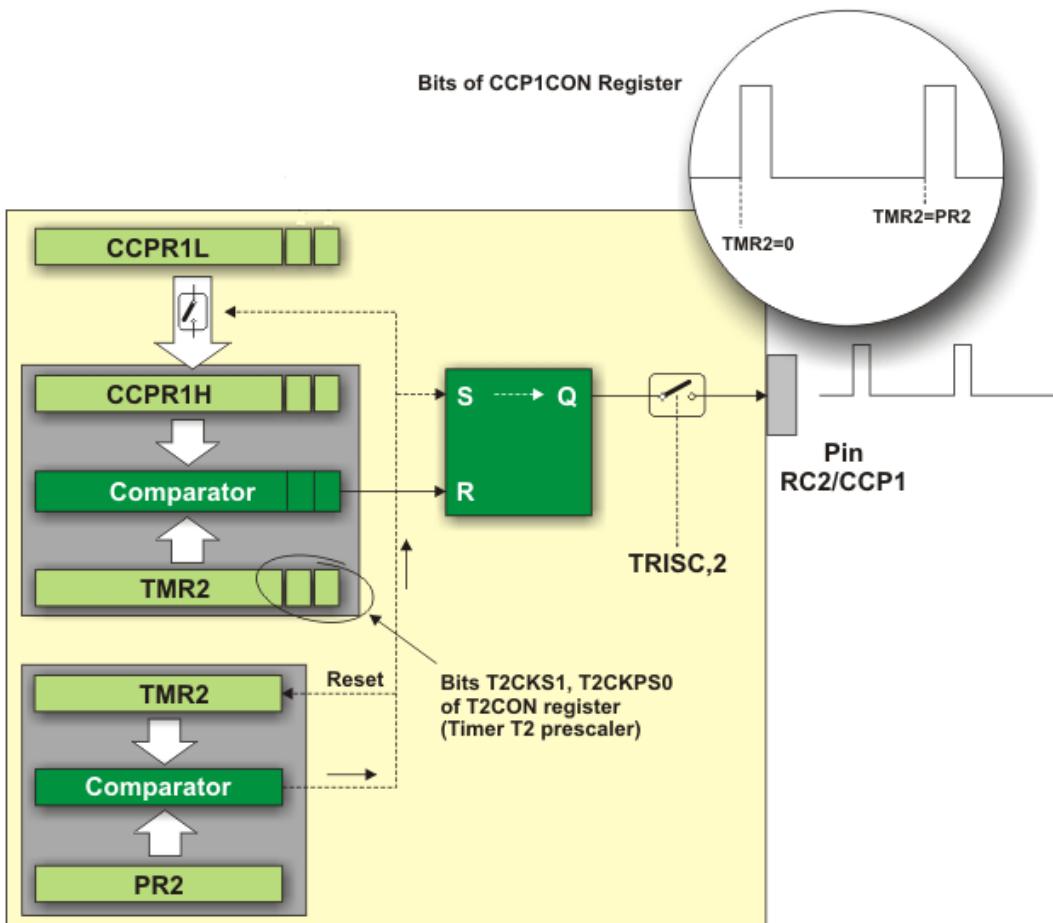
    for(;;);
}

static void interrupt      // Kesme fonksiyonu
isim(void)                 // Kesme fonksiyon ismi (önemsiz)
{
    char i,j;              // Değişkenler tanımlanıyor
    if(CCP1IF)               // CCP1 kesmesi varsa
    {
        i++;                  // Değişken bir artırılıyor
        if(i==1)                // Değişken 1 ise RB0=1 olur
            RB0=1;
        else if(i==2)          // Değişken 2 ise RB0=0 olur
        {
            RB0=0;
            i=0;
        }
        CCP1IF=0;            // Yeni CCP1 kesmesi için bayrak temizleniyor
    }
}
```

Esasen kurulan devre 1000Hz'lik sinyal oluşturacaktır. Bu değeri şekil-31'de bulunan frekansmetreden görebilirsiniz.

3.1.3) PWM Modu

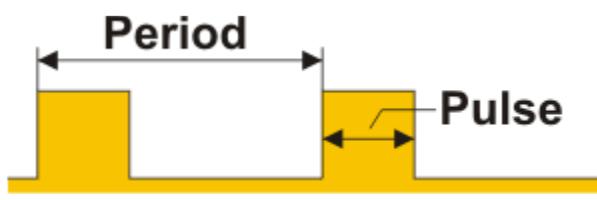
PWM, Pulse Width Modulation'ın kısaltmasıdır. 16f877a'da iki adet PWM modülü bulunmaktadır. Şekil-32'de görüldüğü üzere PWM modülü Timer 2 ile ortaklaşa çalışmaktadır.



Şekil 32 – PWM Birimi

Pic ile PWM sinyalini üretmek için aşağıdaki adımlar izlenir;

- PWM periyodunu ayarlayarak PR2 değerine yazılır,
- PWM Duty Cycle CCPRxL ve CCPxCON<5:4>'e yazılır,
- CCPx output yapılır,
- Timer 2 prescale oranı belirlenir ve Timer 2 çalıştırılır,
- CCPx modülü PWM olarak ayarlanır.



Şekil 33 – PWM Periyodu ve Doluluk Oranı

Burada önemli olan bir diğer faktörde şekil-33'te gözüken PWM periyodu ve doluluk oranıdır. Bu oranlar aşağıdaki matematiksel işlemler izlenerek hesaplanır ve ilgili kaydedicilere yüklenirler.

$$\text{PWM Periyodu} = [(PR2+1) \cdot 4 \cdot \text{Tosc. [TMR2 Prescale Değeri]}]$$

$$\text{PWM Duty Cycle} = [\text{CCPRxL:CCPxCON<5:4>} \cdot \text{Tosc. [TMR2 Prescale Değeri]}]$$

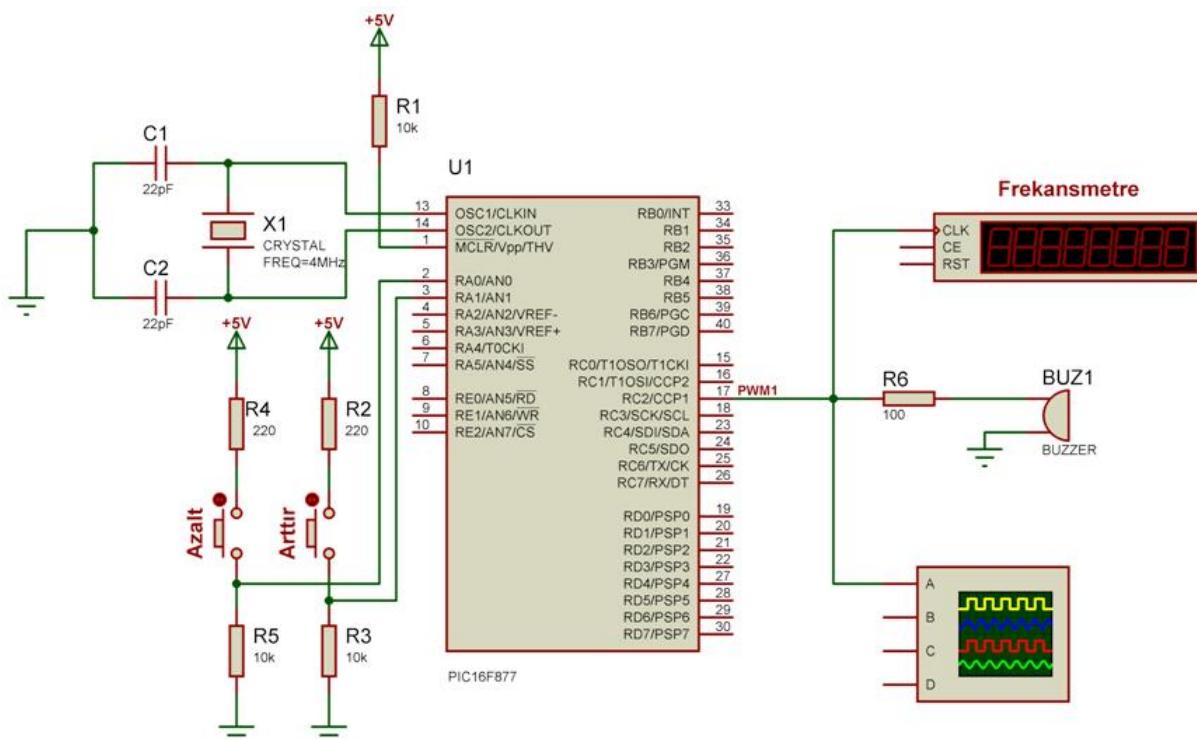
PWM sinyali aşağıdaki adımlarda oluşur;

- TMR2, PR2'ye eşit olduğunda TMR2 temizlenir,
- CCPx pini set edilir (Duty cycle=%0 ise set edilmez),
- CCPRxL'den CCPRxH'a yükleme gerçekleşir.
- Duty cycle, PWM periyodundan büyük olamaz.

PWM çözümünürlüğü ise aşağıdaki formülle hesaplanır;

$$\text{Resolution} = \frac{\log\left(\frac{F_{OSC}}{F_{PWM}}\right)}{\log(2)} \text{ bits}$$

PWM uygulamamızda pic'in CCP1 birimindeki PWM kullanarak, duty cycle'ı değiştirmeden, sadece periyodu değiştirerek buzzer sayesinde sesin nasıl değiştiğini görelim. Duty Cycle periyodunu 1ms kabul edelim. PWM periyodunu ise iki butonla artırıp azaltalım. Öncelikle bu uygulama için şekil-34'teki devreyi çizelim.



Şekil 34 – PWM Uygulaması

İstediğimiz işlemi yapan Hi-Tech kodu ise aşağıdaki gibi olacaktır.

```
#include <htc.h>

void main(void)           // Ana fonksiyon alanı
{
    char i=100;
    ADCON1=0x07;          // PORTA dijital oluyor
    TRISA=0x03;           // RA0 ve RA1 giriş
    TRISC=0x00;            // PORTC çıkış olarak ayarlanıyor
    PORTA=0x00;            // PORTA sıfırlanıyor
    PORTC=0x00;            // PORTC sıfırlanıyor

    CCP1L=0x3E;           // Duty registere 250 yükleniyor
    CCP1X=1;               // Duty cycle 1ms periyodunda
    CCP1Y=0;
    T2CKPS1=1;             // Prescaler 1:16 oluyor
    T2CKPS0=1;
    TOUTPS3=0;              // Postscale 1:1 oluyor
    TOUTPS2=0;
    TOUTPS1=0;
    TOUTPS0=0;

    CCP1M0=1;              // CCP1 PWM modunda
    CCP1M1=1;
    CCP1M2=1;
    CCP1M3=1;

    TMR2ON=1;                // Timer 2 çalıştırılıyor

    for(;;)
    {
        if(RA0)           // Azalt butonuna basılmış mı
        {
            while(RA0); // Butondan elin çekilmesi bekleniyor
            i-=5;          // PR2 birimi 5 azaltılıyor
            if(i<65)      // Değişken 65'ten küçükse
                i=65; // Tekrar 65'e eşit olsun
        }
        if(RA1)           // Artır butonuna basılmış mı
        {
            while(RA1); // Butondan elin çekilmesi bekleniyor
            i+=5;
            if(i>250)    // Değişken 250'den büyükse
                i=250; // Tekrar 250'ye eşit olsun
        }
        PR2=i;           // Değişken PR2'ye eşitleniyor
    }
}
```

İşlemler karışık gibi görünse de aslında tüm iş Duty Cycle ve PWM periyodunu ayarlamaktadır. Bu iş için internette geliştirilen birçok program bulabilirsiniz.

3.2) Tuş Takımı Uygulaması

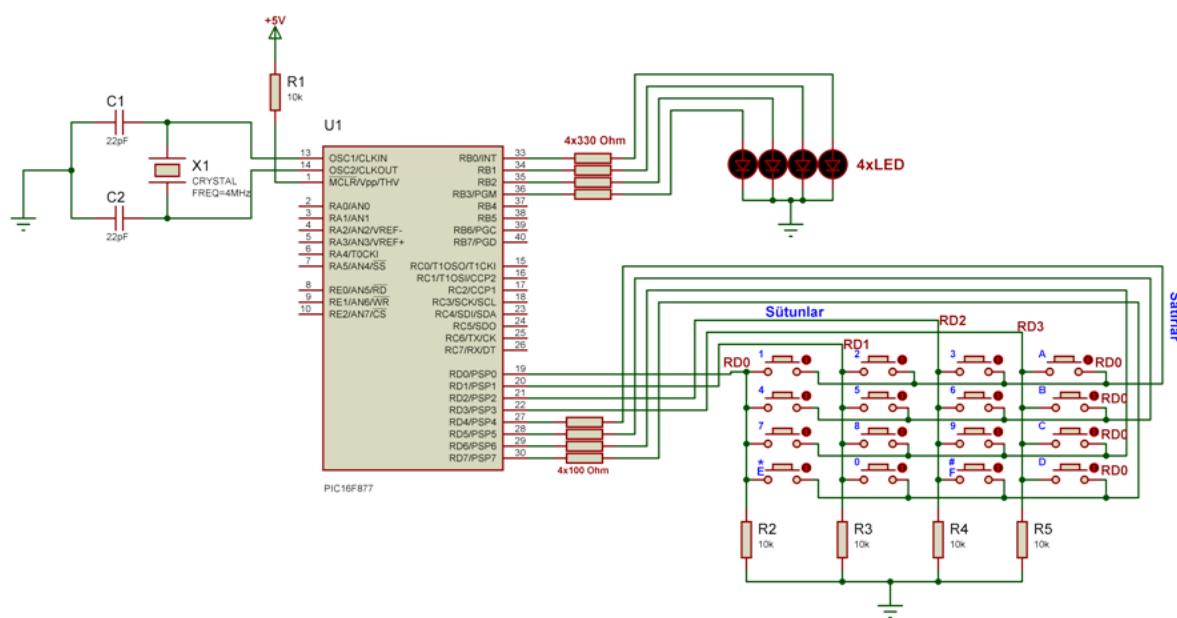
Dışarıdan bilgi girmek için en çok kullanılan ve şekil-35'te gözüken keypad, esasen belirli şekillerde oluşturulmuş pull-up butonlardan oluşur.



Şekil 35 – 4x4 Keypad

Bu bölümde keypad’ı kullanmak için tarama yöntemini irdeleyeceğiz.

Uygulamamızda D portuna bağlanmış, açık şeması bulunan keypad'in tarama yöntemiyle sürülüp, B portuna bağlı ledlere hangi butonun basıldığı yansıtılacaktır. Bunun için öncelikle şekil-36'daki devreyi çiziyoruz.



Sekil 36 – 4x4 Keypad Uygulaması - 1

Görüldüğü üzere açık devresi görülen keypad'in aslında normal pull-up butonlarından farkı yoktur. İstediğimiz işlemi yapan Hi-Tech kodu ise aşağıdaki gibi olacaktır.

```

#include <htc.h>
#include <delay.h>      // Gecikme fonksiyonu

#define sut1      RD0 // sut1 ifadesi RD0 ifadesine eşitleniyor
#define sut2      RD1 // sut2 ifadesi RD1 ifadesine eşitleniyor
#define sut3      RD2 // sut3 ifadesi RD2 ifadesine eşitleniyor
#define sut4      RD3 // sut3 ifadesi RD3 ifadesine eşitleniyor
#define sat1      RD4 // sat1 ifadesi RD4 ifadesine eşitleniyor
#define sat2      RD5 // sat2 ifadesi RD5 ifadesine eşitleniyor
#define sat3      RD6 // sat3 ifadesi RD6 ifadesine eşitleniyor
#define sat4      RD7 // sat4 ifadesi RD7 ifadesine eşitleniyor

char keypad_oku(void) // Fonksiyon ismi
{
    char tus=0;
    PORTD=0x00; // D portu çıkıştı sıfırlanıyor

    sat1=1;      // 1. satır lojik-1 yapılıyor
    if(sut1)     // 1. sütun okunuyor
    {
        DelayMs(20);
        tus=1;
    }
    if(sut2)     // 2. sütun okunuyor
    {
        DelayMs(20);
        tus=2;
    }
    if(sut3)     // 3. sütun okunuyor
    {
        DelayMs(20);
        tus=3;
    }
    if(sut4)     // 4. sütun okunuyor
    {
        DelayMs(20);
        tus=0xA;
    }
    sat1=0;      // 1. satır lojik-0 yapılıyor
    sat2=1;      // 2. satır lojik-1 yapılıyor
    if(sut1)     // 1. sütun okunuyor
    {
        DelayMs(20);
        tus=4;
    }
    if(sut2)     // 2. sütun okunuyor
    {
        DelayMs(20);
        tus=5;
    }
    if(sut3)     // 3. sütun okunuyor
    {
        DelayMs(20);
        tus=6;
    }
    if(sut4)     // 4. sütun okunuyor
    {
        DelayMs(20);
        tus=0xB;
    }
    sat2=0;      // 2. satır lojik-0 yapılıyor
}

```

```

sat3=1;      // 3. satır lojik-1 yapılıyor
if(sut1)    // 1. sütun okunuyor
{
    DelayMs(20);
    tus=7;
}
if(sut2)    // 2. sütun okunuyor
{
    DelayMs(20);
    tus=8;
}
if(sut3)    // 3. sütun okunuyor
{
    DelayMs(20);
    tus=9;
}
if(sut4)    // 4. sütun okunuyor
{
    DelayMs(20);
    tus=0x0C;
}
sat3=0;      // 3. satır lojik-0 yapılıyor
sat4=1;      // 4. satır lojik-1 yapılıyor
if(sut1)    // 1. sütun okunuyor
{
    DelayMs(20);
    tus=0x0E;
}
if(sut2)    // 2. sütun okunuyor
{
    DelayMs(20);
    tus=0;
}
if(sut3)    // 3. sütun okunuyor
{
    DelayMs(20);
    tus=0x0F;
}
if(sut4)    // 4. sütun okunuyor
{
    DelayMs(20);
    tus=0x0D;
}
sat4=0;      // 4. satır lojik-0 yapılıyor

return tus; // Fonksiyon "tus" değeri ile geri döner
}
void main(void)           // Ana fonksiyon alanı
{
    TRISB=0x00;
    TRISD=0x0F;
    PORTB=0x00;
    PORTD=0x00;

    for(;;)
    {
        PORTB=keypad_oku();
    }
}

```

Görüldüğü üzere tarama metodu özellikle büyük projelerde sorun yaratacak düzeyde uzundur. Tarama işlemini 2. bölümde gördüğümüz Timer0 kesmesiyle yaparak, programsal açıdan sürekli kontrol işlemini gereksiz hale getirip, projenize etkinlik kazandırabilirsiniz.

BÖLÜM 4 – KAREKTER LCD ve ADC İŞLEMLERİ

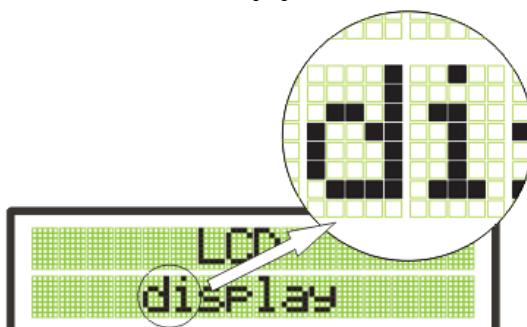
4.1) Hi-Tech'te Karekter LCD İşlemleri

Şekil-37'de bir örneği görülen karakter LCD'ler dışarıya bilgi aktarmak için kullanılan en yaygın birimlerdir. Genel itibarı ile Hitachi firmasının HD44780 entegresini ve türevlerini taşıyan karakter LCD'ler çeşitli metotlarla sürürlürler. Biz bu bölümde şekil-37'de de görülebilecek 2x16 yani 2 satır ve her satırda 16 karakter yazabilen karakter LCD'leri inceleyeceğiz.



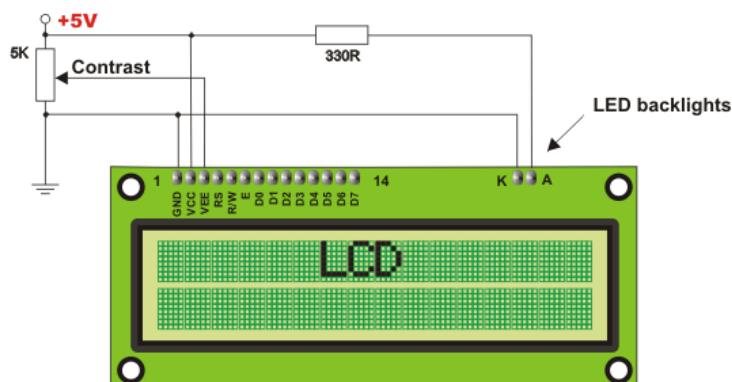
Şekil 37 – 2x16 Karekter LCD

Karakter LCD'lerin genelinde her harf şekil-38'de görüleceği gibi 5x7'lik birimler halinde şekillenirler. Altta boş kalan son birim ise imleç içindir.



Şekil 38 – 5x7 Karekter Oluşumu

LCD birimi genellikle normal entegre güç biriminden ayrı bir de arka aydınlatma ışığı gücü verilerek kullanılırlar. Bu birimin nasıl sürüleceği ise şekil-39'de gözükmektedir.



Şekil 39 – LCD Güç Bağlantısı

Karakter LCD'lerin oluşturabileceği her bir karakter ise karakter LCD'nin özel CGROM hafızasına kaydedilmişlerdir. ASCII karakter uyumu olan karakterlerin listesi şekil-40'da görülebilmektedir.

	4 higher bits of address															
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)			Ø Ø P ^ P						- ♫ Ǝ Ǝ Ø p						
xxxx0001	(2)		! 1 A Q a q							• ♫ Ǝ Ǝ ä q						
xxxx0010	(3)		" 2 B R b r							「 イ ツ × β Ǝ						
xxxx0011	(4)		# 3 C S c s							」 ウ テ モ ε *						
xxxx0100	(5)		\$ 4 D T d t							、 エ ト ャ ハ ジ						
xxxx0101	(6)		% 5 E U e u							・ オ ナ ュ シ フ						
xxxx0110	(7)		& 6 F V f v							ヲ カ ニ ョ ポ ゾ						
xxxx0111	(8)		' 7 G W g w							ア キ ヌ ラ ゴ ピ						
xxxx1000	(1)		(8 H X h x							イ ク ネ リ ジ ズ						
xxxx1001	(2)) 9 I Y i y							ウ ル ノ ル イ ピ						
xxxx1010	(3)		* : J Z j z							エ ボ ハ レ j キ						
xxxx1011	(4)		+ ; K C k {							オ サ ヒ ロ * ジ						
xxxx1100	(5)		, < L ¥ I I							ヤ シ フ ワ * フ メ						
xxxx1101	(6)		- = M] m }							ユ ス ヘ ナ キ ナ						
xxxx1110	(7)		. > N ^ n →							ミ セ ホ ム ハ ニ						
xxxx1111	(8)		/ ? O _ o ←							レ リ マ ド ö						

Şekil 40 – LCD Karakter Tablosu

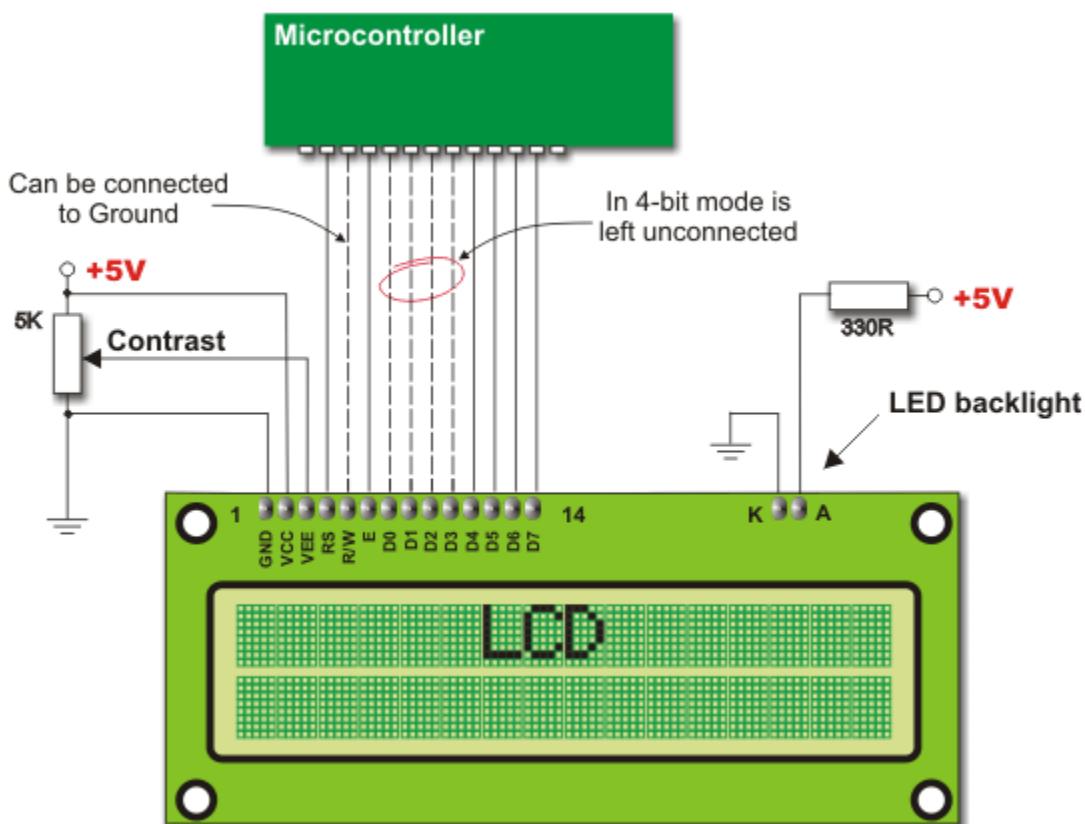
Şekil-40'da da görüleceği üzere CGROM'un ilk 8 karakterlik (0x00..0x0F) kısmı boştur ve yazılabılırdir. Bu kullanıcıya tabloda olmayan karakterleri kendisi tanımlamasına olanak sağlar.

Karakter LCD'lerin genelin 16 bacak bulunur. Bunların 14 tanesi LCD'yi kontrol etmek amaçlı kullanılırken, 15. ve 16. bacaklar genellikle LCD arka ışığı için kullanılırlar. LCD arka ışığı yazıların daha belirgin gözükmesi için gereklidir.

Bu bacakların görevini sırasıyla verecek olursak;

- 1 - GND : Toprak ucudur
- 2 - VCC : +5V verilecek uçtur
- 3 - VEE : Kontrast ucudur, bir pot vasıtasyyla +5V-0V aralığında sürülmeli
- 4 - RS : Gelen bilginin komut mu data mı olduğu bu uçla belirlenir
(0: Komut, 1: Data)
- 5 - RW : LCD'ye veri yazma ya da okuma yetkilendirme ucudur (0: Yazma, 1: Okuma)
- 6 - E : Enable ucudur, LCD'ye bilgi giriş çıkışını kontrol eden uçtur, düşen kenar tetiklemelidir
- 7..14 - Data : Data uçlarıdır, bilgi giriş çıkışları bu bacaklar sayesinde olur
- 15,16 - BL : Backlight anot, katot ucharıdır

Karakter LCD'lerin kullanılması, led, direnç sürümü gibi olmamaktadır. Karakter LCD kullanımında, enerjiyi ilk verdigimiz anda karakter LCD'yi nasıl kullanmak istedigimizi LCD'ye belirli kurallar çerçevesinde iletmemiz gerekmektedir. Biz şekil-41'deki metodu kullanarak karakter LCD'mizi çalıştıracağız.



Şekil 41 – LCD Karakter Tablosu

Bu metot LCD sürmede klasik haline gelmiştir. Şekil-41'de görülen bağlantıda bilgi gönderimi 8 bitlik veriler yerine 4 bitlik veriler halinde iki kez gönderilerek yapılmaktadır. Böylelikle bize mikro denetleyicide 4 bacak kazanımı sağlarken, karakter yazım hızında ise 2 kat yavaşlamaya neden olmaktadır. Fakat bu, çok komplike işlemler yapılmadıktan sonra gözle görülür bir yavaşlamaya neden olmamaktadır. Ayrıca biz işlemlerimizde karakter LCD'ye sadece yazım yapacağımız için RW bacağını direkt toprak hattına bağlayabilir, toplamda 6 mikro denetleyici bacağı kullanarak LCD'mizi çalıştırabiliriz.

Bunlar göz önüne alınarak yapılacak ilk ön yüklemeye aşağıdaki ilk yüklemeye bilgileri seçilmeli ve karakter LCD'ye gönderilmelidir. Bu bilgiler karakter LCD datasheet'inden bire bir alınmıştır.

- 1) Display sıfırlanmalı** : Bunun için LCD'ye 0x02 veya 0x03 gönderilip 2ms beklenir
- 2) Display silinmeli** : Bunun için LDD'ye 0x01 gönderilip 1ms beklenilmeli
- 3) Fonksiyon ayarı yapılmalı** : 4 bitlik iletişim 5x7 karakter kullanacağımızdan 0x28 gönderilip
40us beklenilmeli
- 4) Giriş modu ayarlanması** : İmleç her karakterden sonra sağa kayacağından 0x06 gönderilip 40us beklenilmeli
- 5) İmleç ayarı** : Display kapalı uygulama yapacağımızdan 0x0B gönderilir

4.1.1) LCD Kütüphanesini Oluşturmak

C'nin en büyük avantajlarından birisi kendi kütüphanelerimizi oluşturabilmek ve bunları istediğimiz zaman kullanabilmektir. **Kütüphane oluşturmada .h ve .c uzantılı iki dosya oluşturmamız gerekmektedir. .h uzantılı dosya, .c uzantılı dosyada tanımlayacağımız fonksiyon isimlerini içerirken, .c uzantılı dosya da ise kullanacağımız fonksiyonların işlevleri yer almmalıdır.** Biz oluşturacağımız LCD'de istenildiği an imlecin gözüküp, kapatılması, LCD'nin temizlenmesi gibi işlemlerin hepsini yapabilmek için tüm komutların numaralarını yazacağımız C kütüphanesine koyacağız. **Öncelikle lcd.h dosyamızı proje klasörümüzde oluştururalım, bunu yapanın en kolay yolu boş bir metin dosyası oluşturup sonundaki .txt uzantisını .h şeklinde değiştirmektir. Daha sonra Hi-Tech altında bölüm 1'de anlatılan refresh işlemi gerçekleştirildikten sonra boş lcd.h dosyası karşımıza çıkacaktır. lcd.h başlık dosyamızı aşağıdaki şekilde oluşturalım.**

```
/*
 * www.FxDev.org
 * D4,D5,D6,D7 pinlerini kullanarak 4 bit iletişim kullanır.
 * Cursor kapalıdır.
 * RW kullanılmadığı için direk toprağa bağlanabilir.
 * 2x16 LCD Kullanım Klavuzu
 * lcd_init(); ile LCD'nin ilk ayarlarını yap
 * lcd_clear(); ile LCD'yi sil
 * lcd_yaz("deneme"); şeklinde yazı yazdır.
 * veri_yolla('F'); şeklinde tek ascii kodu yazdır.
 * lcd_gotoxy(1,13); şeklinde LCD'nin istenilen yerine git.
 * www.FxDev.org
 */

#define rs RC0      //Pin tanımlamaları
#define rw RC1
#define e  RC2
#define lcd_port PORTB

/* LCD'de kullanılan komutların tanımlaması*/
#define Sil     1      // Ekrani temizler
#define BasaDon 2      // İmleci sol üst köseye getirir
#define SolaYaz 4      // İmlecin belirttiği adres azalarak gider
#define SagaYaz 6      // İmlecin belirttiği adres artarak gider
#define ImlecGizle 12      // Göstergeyi aç, cursor görünmesin
#define ImlecAltta 14      // Yanıp sönen blok cursor
#define ImlecYanson 15      // Yanıp sönen blok cursor
```

```

#define ImlecGeri    16      // Kursoru bir karakter geri kaydır
#define KaydirSaga   24      // Göstergeyi bir karakter sağa kaydır
#define KaydirSola   28      // Göstergeyi bir karakter sola kaydır
#define EkraniKapat  8       // Göstergeyi kapat (veriler silinmez)
#define BirinciSatir 128     // LCD'nin ilk satır başlangıç adresi
                           // (DDRAM adres)
#define IkinciSatir 192     // İkinci satırın başlangıç adresi
#define KarakUretAdres 64    // Karakter üretici adresini belirle
                           // (CGRAM adres)

/* LCD'de Kullanılan Fonksiyon Seçimi */
#define CiftSatir8Bit 56      // 8 bit ara birim, 2 satır, 5*7 piksel
#define TekSatir8Bit  48      // 8 bit ara birim, 1 satır, 5*7 piksel
#define CiftSatir4Bit 40      // 4 bit ara birim, 2 satır, 5*7 piksel
#define TekSatir4Bit  32      // 4 bit ara birim, 1 satır, 5*7 piksel

extern void veri_yolla(unsigned char);
extern void lcd_clear(void);
extern void lcd_yaz(const char *s);
extern void lcd_gotoxy(unsigned char x, unsigned char y);
extern void lcd_init(void);
extern void lcd_komut(unsigned char c);

```

Şekildeki kodların ne işe yaradıkları yanlarındaki açıklamalarda bulunmaktadır. Bu kodları lcd.h içine koyduktan sonra kaydedelim. Kullanım kılavuzunda da belirtildiği gibi **bacakların nasıl bağlanacağı da lcd.h dosyasında belirlenmektedir.**

Daha sonra görüldüğü üzere extern uzantılı fonksiyonların ne işe yaradıklarını açıklayan lcd.c dosyamızı aynen yukarıda anlatılan yeni metin dosyası metoduyla yaratıktan sonra aşağıdaki kodları içine kaydedelim.

```

#include <pic.h>
#include "lcd.h"      // lcd.h dosyası tanımlanıp, değerler alınıyor
#include "delay.h"    // Gecikme fonksiyonu tanımlanıyor

void lcd_busy(void)
{
    DelayUs(250);
}

void lcd_komut(unsigned char c)      // Komut gönderme fonksiyonu
{
    rw=0;                      // LCD'ye yazma yapılacak
    rs=0;                      // LCD'ye komut gönderilecek
    e=1;                       // Düşen kenar tetikleme olduğu için E önce 1
    lcd_port = (c & 0xF0); // Yüksek değerlikli bitler gönderiliyor
    e=0;                       // E, 0 yapılıyor
    lcd_busy();                // Belirli süre bekleniyor
    e=1;                       // E, 1 yapılıyor
    lcd_port = ( (c & 0x0F)<<4 ); // Düşük değerlikli bitler
gönderiliyor
    e=0;                      // E, 0 yapılıyor
    lcd_busy();                // Belirli bir süre bekleniyor
}

void veri_yolla(unsigned char c)
{
    rw=0;
    rs=1;                     // Komut yolladan tek farkı, RS'nin 1 olması

```

```

        e=1;
        lcd_port = ( c & 0xF0 );
        e=0;
        lcd_busy();
        e=1;
        lcd_port = ( (c & 0x0F)<<4 );
        e=0;
        lcd_busy();
    }

void lcd_clear(void)           // LCD siliniyor
{
    lcd_komut(0x1);
    DelayMs(2);
}

void lcd_yaz(const char * s) // LCD'ye string ifade gönderiliyor
{
    lcd_busy();
    while(*s)
        veri_yolla(*s++);
}

void lcd_gotoxy(unsigned char x,unsigned char y) //LCD'nin belli
                                                //bölgесine gidiliyor
{
    if(x==1)
        lcd_komut(0x80+((y-1)%16));
    else
        lcd_komut(0xC0+((y-1)%16));
}

void lcd_init()           // LCD ilk yükleme ayarları yapılıyor
{
    rs = 0;
    e = 0;
    rw = 0;

    DelayMs(15);
    e=1;
    lcd_komut(0x02);
    DelayMs(2);

    lcd_komut(CiftSatir4Bit);
    lcd_komut(SagaYaz);
    lcd_komut(ImlecGizle);
    lcd_clear();
    lcd_komut(BirinciSatir);
}

```

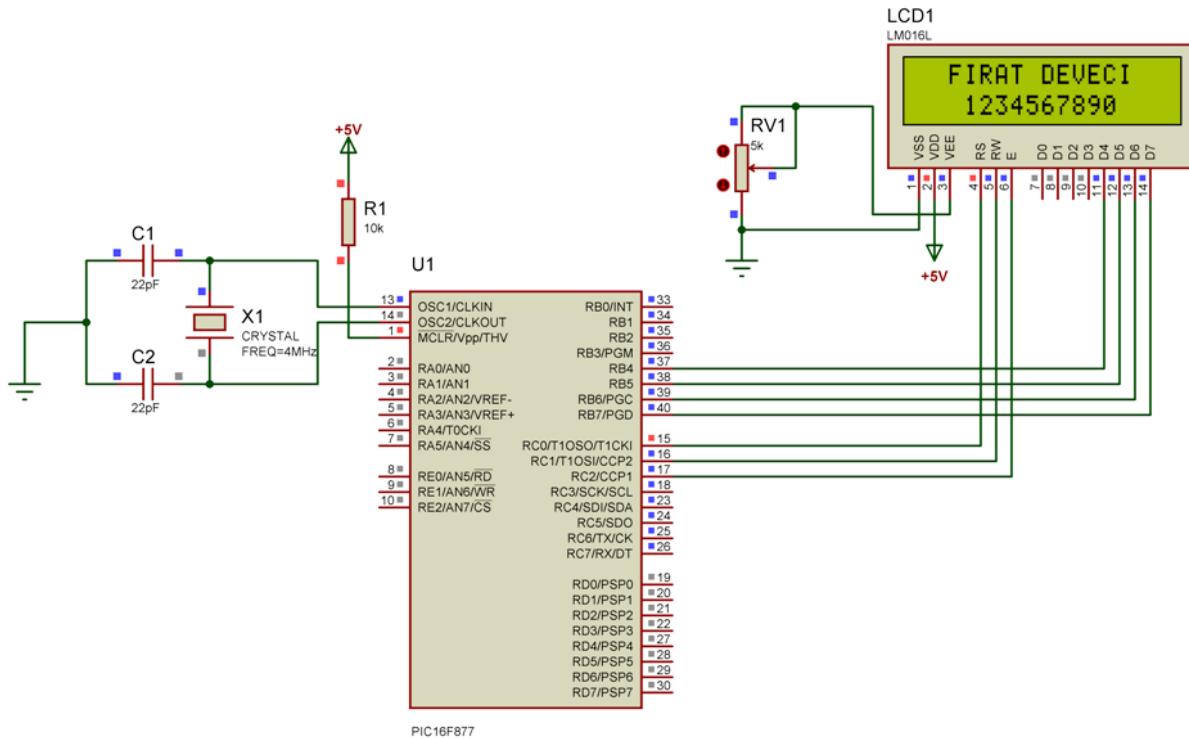
Lcd.c dosyamızı da kaydettikten sonra artık yapmamız gereken tek şey bunu ana proje dosyamız olan main.c'de kütüphane olarak tanımlayarak kullanmaktır.

Önemli Not: Sonraki konularda sıkça kullanılacak olan LCD kütüphanesi bu olacaktır. Projelerimize lcd.h dosyasını eklediğimizde, esasen eklenen kütüphane, bu oluşturduğumuz kütüphane olacaktır.

4.1.2) İlk LCD Uygulamamız

LCD kütüphanelerimizi oluşturdukten sonra ilk denememize geçelim.

İlk uygulamamızda LCD'ye adımızı soyadımızı ve ikinci satırda tüm rakamları yazalım. Öncelikle şekil-42'deki devremizi **Icd.h** kütüphanesinde tanımladığımız bacaklara dikkat ederek kuralım. Devreyi çalıştırduğumuzda LCD'nin nasıl gözükeceği de şekil-42'de gözükmektedir.



Şekil 42 – LCD Karakter Uygulaması

Bu işlemi yapan Hi-Tech kodu ise aşağıda görüldüğü üzere oldukça sadedir.

```
#include <htc.h>
#include <delay.h>           // Gecikme kütüphanesi
#include "lcd.h"              // LCD kütüphanesi tanımlanıyor

void main(void)               // Ana fonksiyon alanı
{
    TRISB=0x00;                // LCD'ye bağlı portlar çıkış yapılıyor
    TRISC=0x00;
    PORTB=0x00;
    PORTC=0x00;

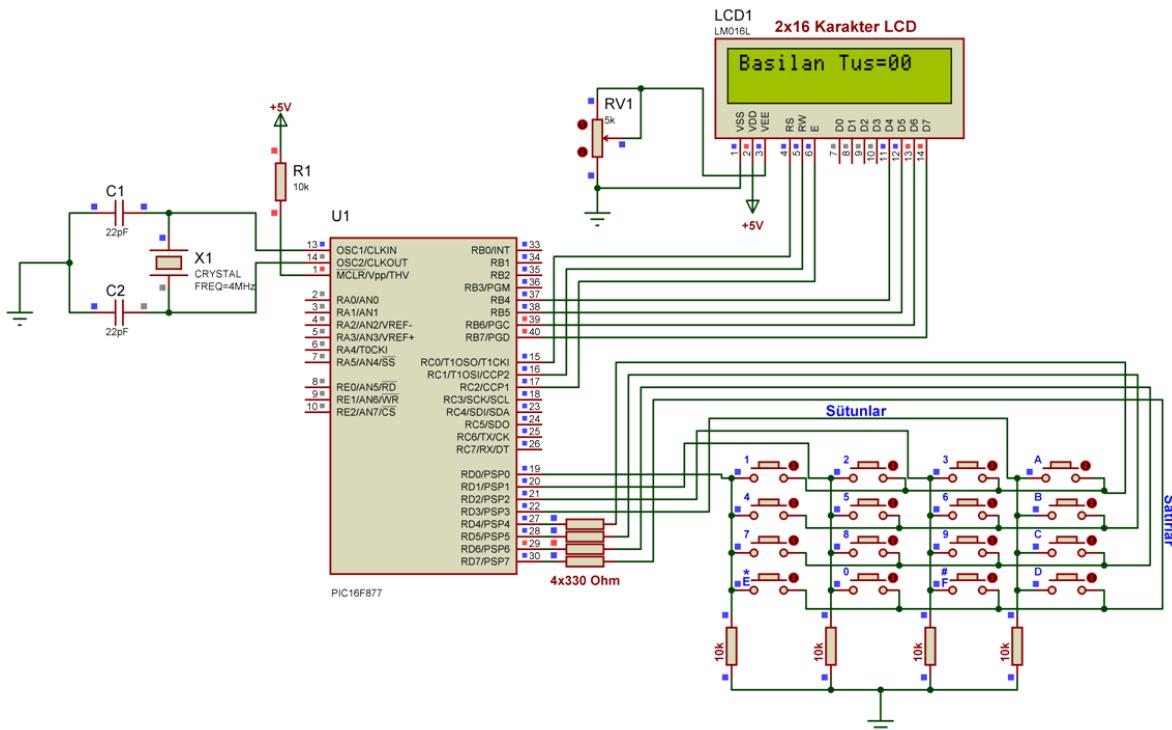
    lcd_init();                 // LCD ilk ayarları yapılıyor

    lcd_yaz(" FIRAT DEVECI");   // İlk satırda isim yazılıyor
    lcd_gotoxy(2,1);             // İkinci satırda geçiliyor
    lcd_yaz(" 1234567890");      // Rakamlar yazılıyor
    for(;;);                     // Sonsuz döngüye giriliyor
}
```

Lcd.h dosyasının “Lcd.h” şeklinde tanımlanmasının nedeni, kütüphanenin proje klasöründe olmasındandır. Son olarak sonsuz döngüye girilmesinin nedeni ise sürekli şekilde LCD’ye yazımı engellemektir. Görüldüğü üzere kütüphane kullanmak, ana kodumuzu oldukça kısaltmaktadır.

4.1.3) Tuş Takımı ve LCD Uygulaması

Bu uygulamamızda 3. bölümde gördüğümüz keypad ile LCD uygulamasını birleştirelim. **Tarama metodunu kullanarak LCD’de bastığımız tuşun gözükmesini sağlayalım.** Bunun için uygulama anını da görebileceğiniz şekil-43’teki devreyi çizelim.



Şekil 43 – Tuş Takımı ve LCD Karakter Uygulaması

Bu devreyi çalıştıracak olan Hi-Tech kodu ise aşağıdaki gibidir.

```
#include <htc.h>
#include <delay.h>           // Gecikme kütüphanesi tanımlanıyor
#include "lcd.h"              // LCD kütüphanesi tanımlanıyor

#define sut1      RD0    // sut1 ifadesi RD0 ifadesine eşitleniyor
#define sut2      RD1    // sut2 ifadesi RD1 ifadesine eşitleniyor
#define sut3      RD2    // sut3 ifadesi RD2 ifadesine eşitleniyor
#define sut4      RD3    // sut3 ifadesi RD3 ifadesine eşitleniyor

#define sat1      RD4    // sat1 ifadesi RD4 ifadesine eşitleniyor
#define sat2      RD5    // sat2 ifadesi RD5 ifadesine eşitleniyor
#define sat3      RD6    // sat3 ifadesi RD6 ifadesine eşitleniyor
#define sat4      RD7    // sat4 ifadesi RD7 ifadesine eşitleniyor

char keypad_oku(void) // Fonksiyon ismi
{
    char tus=0;
    PORTD=0x00; // D portu çıkıştı sıfırlanıyor
```

```

sat1=1;      // 1. satır lojik-1 yapılıyor
if(sut1)    // 1. sütun okunuyor
{
    DelayMs(20);
    tus=1;
}
if(sut2)    // 2. sütun okunuyor
{
    DelayMs(20);
    tus=2;
}
if(sut3)    // 3. sütun okunuyor
{
    DelayMs(20);
    tus=3;
}
if(sut4)    // 4. sütun okunuyor
{
    DelayMs(20);
    tus=0x0A;
}
sat1=0;      // 1. satır lojik-0 yapılıyor

sat2=1;      // 2. satır lojik-1 yapılıyor
if(sut1)    // 1. sütun okunuyor
{
    DelayMs(20);
    tus=4;
}
if(sut2)    // 2. sütun okunuyor
{
    DelayMs(20);
    tus=5;
}
if(sut3)    // 3. sütun okunuyor
{
    DelayMs(20);
    tus=6;
}
if(sut4)    // 4. sütun okunuyor
{
    DelayMs(20);
    tus=0x0B;
}
sat2=0;      // 1. satır lojik-0 yapılıyor

sat3=1;      // 3. satır lojik-1 yapılıyor
if(sut1)    // 1. sütun okunuyor
{
    DelayMs(20);
    tus=7;
}
if(sut2)    // 2. sütun okunuyor
{
    DelayMs(20);
    tus=8;
}
if(sut3)    // 3. sütun okunuyor
{
    DelayMs(20);
}

```

```

        tus=9;
    }
    if(sut4)      // 4. sütun okunuyor
    {
        DelayMs(20);
        tus=0x0C;
    }
sat3=0;      // 1. satır lojik-0 yapılıyor

sat4=1;      // 4. satır lojik-1 yapılıyor
if(sut1)      // 1. sütun okunuyor
{
    DelayMs(20);
    tus=0x0E;
}
if(sut2)      // 2. sütun okunuyor
{
    DelayMs(20);
    tus=0;
}
if(sut3)      // 3. sütun okunuyor
{
    DelayMs(20);
    tus=0x0F;
}
if(sut4)      // 4. sütun okunuyor
{
    DelayMs(20);
    tus=0x0D;
}
sat4=0;      // 4. satır lojik-0 yapılıyor

return tus; // Fonksiyon "tus" değeri ile geri döner
}

void main(void)           // Ana fonksiyon alanı
{
    TRISB=0x00;          // LCD için çıkış
    TRIISC=0x00;
    TRIISD=0x0F;         // Tuş takımı için giriş ve çıkış
    PORTB=0x00;
    PORTC=0x00;
    PORTD=0x00;

    lcd_init();           // LCD ilk ayarları yapılıyor
    lcd_yaz("Basilan Tus=");
    for(;;)
    {
        lcd_gotoxy(1,13); // LCD 1x13'e gidiliyor
        while(keypad_oku()!=0); // Keypad değeri 0'dan farklı mı
        veri_yolla(keypad_oku()/10+48); // Onlar yazılıyor
        veri_yolla(keypad_oku()%10+48); // Birler yazılıyor
    }
}

```

Görüldüğü üzere main fonksiyonumuz çok sade olmasına karşın kodu en çok uzatan keypad tarama fonksiyonudur. Bu fonksiyonu da kütüphane haline getirip kullanmak mümkündür.

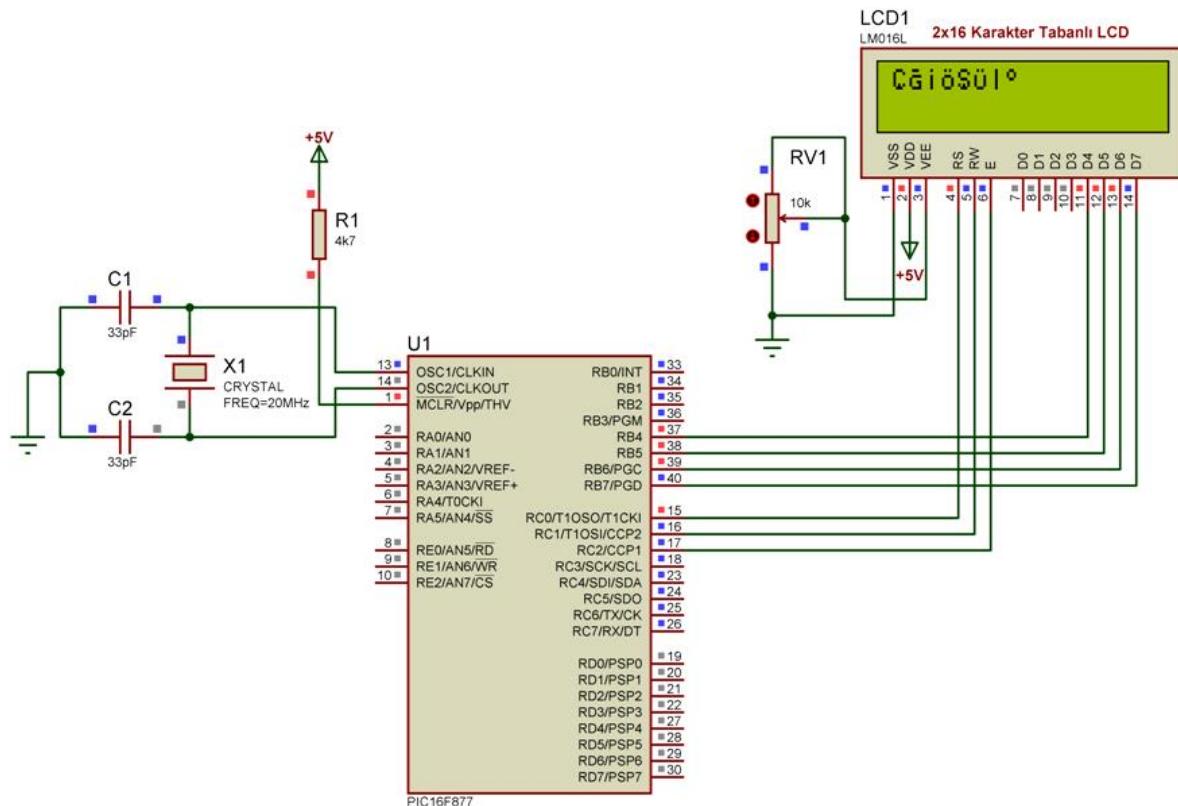
```
veri_yolla(keypad_oku() /10+48);
```

Bu bölümde dikkat edilmesi gereken bölüm yukarıdaki fonksiyon kullanımıdır. "veri_yolla" fonksiyonu LCD'ye Ascii karakter göndermeye yarayan fonksiyondur, 48 ise 0'ın Ascii kodudur, dolayısı ile yapılan işlemin sonucu 2 ise $48+2=50$ yapar ve 50, 2'nin Ascii kodudur. Aslında C'de bu işlemi yapan sprintf fonksiyonu vardır, fakat pic'de çok fazla yer işgal etmektedir, biz tüm bölümlerde "veri_yolla" vb. fonksiyonları kullanarak aslında bu işin o kadar da karmaşık olmadığını göreceğiz.

4.1.4) Özel Karakter LCD Uygulaması

Şekil-40'da da görüleceği üzere CGROM üzerinde tanımlanmış herhangi bir Türkçe karakter mevcut değildir. Fakat önceden de söylediğimiz gibi CGROM'un ilk 8 karakterlik boş kısmı yazmaya müsaittir ve dilediğimiz şekli oraya koymak mümkündür.

Bu uygulamamızda Türkçe karakterleri bu boş adreslere yerlestirecek ve kullanacağız. Öncelikle şekil-44'te uygulamasını da görebileceğiniz devreyi çiziyoruz.



Şekil 44 – Özel Karakter LCD Uygulaması

Yukarıda görüldüğü gibi Ç, Ğ, i, ö, Ş, ü, | ve derece işaretini gibi özel karakterler LCD'nin CGROM'una yüklenmiş ve gösterilmiştir. Bunu gerçekleyen C kodu ise şöyledir.

```

#include <htc.h>
#include <delay.h>      // Gecikme kütüphanesi tanimlanıyor
#include "lcd.h" // LCD kütüphanesi tanimlanıyor

void ozel_karakterler(void)
{
    // CGRAM'de 1. Adrese "Ç" Karakteri Kaydediliyor
    lcd_komut(0x40);veri_yolla(14);
    lcd_komut(0x41);veri_yolla(17);
    lcd_komut(0x42);veri_yolla(16);
    lcd_komut(0x43);veri_yolla(16);
    lcd_komut(0x44);veri_yolla(16);
    lcd_komut(0x45);veri_yolla(21);
    lcd_komut(0x46);veri_yolla(14);
    lcd_komut(0x47);veri_yolla(0);

    // CGRAM'de 1. Adrese "Ğ" Karakteri Kaydediliyor
    lcd_komut(0x48);veri_yolla(14);
    lcd_komut(0x49);veri_yolla(0);
    lcd_komut(0x4A);veri_yolla(15);
    lcd_komut(0x4B);veri_yolla(16);
    lcd_komut(0x4C);veri_yolla(19);
    lcd_komut(0x4D);veri_yolla(17);
    lcd_komut(0x4E);veri_yolla(15);
    lcd_komut(0x4F);veri_yolla(0);

    // CGRAM'de 2. Adrese "İ" Karakteri Kaydediliyor
    lcd_komut(0x50);veri_yolla(4);
    lcd_komut(0x51);veri_yolla(0);
    lcd_komut(0x52);veri_yolla(4);
    lcd_komut(0x53);veri_yolla(4);
    lcd_komut(0x54);veri_yolla(4);
    lcd_komut(0x55);veri_yolla(4);
    lcd_komut(0x56);veri_yolla(4);
    lcd_komut(0x57);veri_yolla(0);

    // CGRAM'de 3. Adrese "Ö" Karakteri Kaydediliyor
    lcd_komut(0x58);veri_yolla(10);
    lcd_komut(0x59);veri_yolla(0);
    lcd_komut(0x5A);veri_yolla(14);
    lcd_komut(0x5B);veri_yolla(17);
    lcd_komut(0x5C);veri_yolla(17);
    lcd_komut(0x5D);veri_yolla(17);
    lcd_komut(0x5E);veri_yolla(14);
    lcd_komut(0x5F);veri_yolla(0);

    // CGRAM'de 4. Adrese "Ş" Karakteri Kaydediliyor
    lcd_komut(0x60);veri_yolla(14);
    lcd_komut(0x61);veri_yolla(17);
    lcd_komut(0x62);veri_yolla(16);
    lcd_komut(0x63);veri_yolla(14);
    lcd_komut(0x64);veri_yolla(1);
    lcd_komut(0x65);veri_yolla(21);
    lcd_komut(0x66);veri_yolla(14);
    lcd_komut(0x67);veri_yolla(0);

    // CGRAM'de 5. Adrese "Ü" Karakteri Kaydediliyor
    lcd_komut(0x68);veri_yolla(10);
    lcd_komut(0x69);veri_yolla(0);
    lcd_komut(0x6A);veri_yolla(17);
    lcd_komut(0x6B);veri_yolla(17);
}

```

```

lcd_komut(0x6C);veri_yolla(17);
lcd_komut(0x6D);veri_yolla(17);
lcd_komut(0x6E);veri_yolla(14);
lcd_komut(0x6F);veri_yolla(0);

// CGRAM'de 6. Adrese "I" Karakteri Kaydediliyor
lcd_komut(0x70);veri_yolla(4);
lcd_komut(0x71);veri_yolla(4);
lcd_komut(0x72);veri_yolla(4);
lcd_komut(0x73);veri_yolla(4);
lcd_komut(0x74);veri_yolla(4);
lcd_komut(0x75);veri_yolla(4);
lcd_komut(0x76);veri_yolla(4);
lcd_komut(0x77);veri_yolla(0);

// CGRAM'de 7. Adrese ":" Karakteri Kaydediliyor
lcd_komut(0x78);veri_yolla(12);
lcd_komut(0x79);veri_yolla(18);
lcd_komut(0x7A);veri_yolla(18);
lcd_komut(0x7B);veri_yolla(12);
lcd_komut(0x7C);veri_yolla(0);
lcd_komut(0x7D);veri_yolla(0);
lcd_komut(0x7E);veri_yolla(0);
lcd_komut(0x7F);veri_yolla(0);

lcd_komut(BirinciSatir);
}

void main(void) // Ana fonksiyon alanı
{
    TRISB=0x00; // LCD için çıkış
    TRISC=0x00;
    PORTB=0x00;
    PORTC=0x00;

    lcd_init(); // LCD ilk ayarları yapılıyor
    ozel_karakterler(); // Özel karakterler yükleniyor

    veri_yolla(0); // 1. özel karakter yazdırılıyor
    veri_yolla(1); // 2. özel karakter yazdırılıyor
    veri_yolla(2); // 3. özel karakter yazdırılıyor
    veri_yolla(3); // 4. özel karakter yazdırılıyor
    veri_yolla(4); // 5. özel karakter yazdırılıyor
    veri_yolla(5); // 6. özel karakter yazdırılıyor
    veri_yolla(6); // 7. özel karakter yazdırılıyor
    veri_yolla(7); // 8. özel karakter yazdırılıyor
    for(;;);
}

```

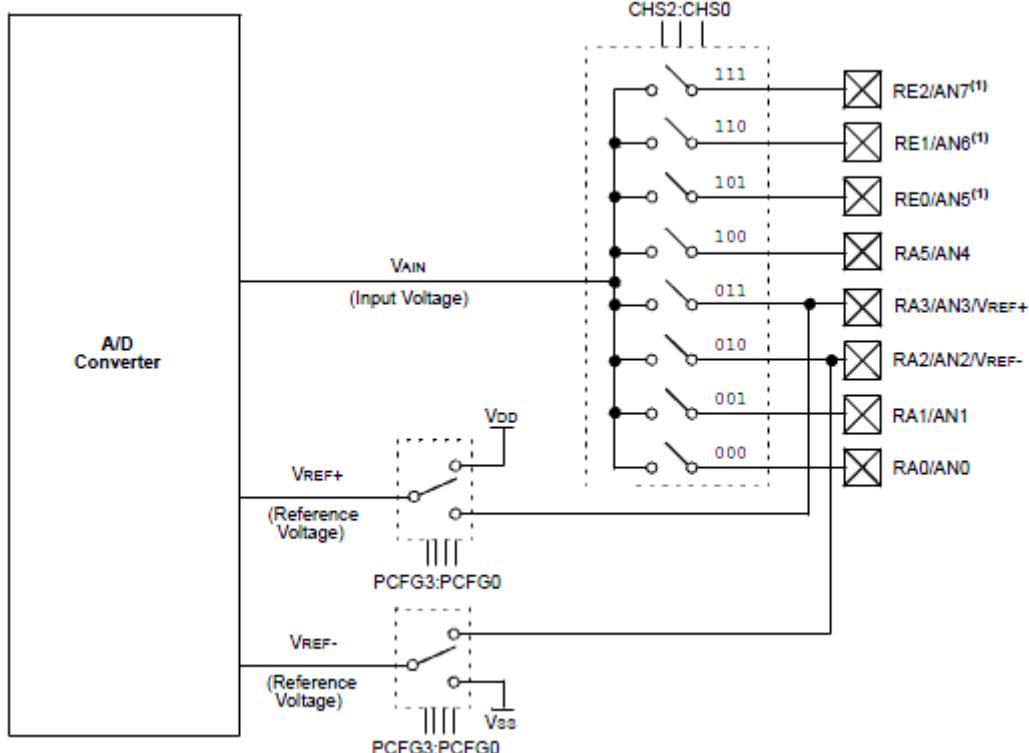
Burada dikkat edilmesi gereken konu ise CGRAM adresine yükleme yapıldıktan sonra tekrar ilk satırda dönülmesi gerektidir. Yoksa cursor LCD'nin görünmez bölgelerine geçerek, istediğimiz gibi çalışmamaktadır.

Bu bölümde LCD'nin kullanılmasının Hi-Tech altında ne kadar kolay olabileceğini görmüş olduk. Özel şekilleri oluşturan programları internetten kolaylıkla bulabilirsiniz.

4.2) ADC İşlemleri

Pic 16f877a içerisinde dahili olarak bulunan ADC modülü 0-5V aralığındaki analog bilgiyi 10 bitlik çözünürlük ile digital bilgiye dönüştürür. Bu da bize yaklaşık binde bir hassasiyet kazandırır.

Şekil-45'te de görüleceği üzere Pic16f877a içerisinde tam 8 kanal ADC birimi mevcuttur ve girişleri PORTA ve PORTE birimleri tarafından sağlanır.



Şekil 45 – ADC Birimi

ADC birimi ADCON0 ve ADCON1 adlı iki adet kaydedici ile kontrol edilir. Bu kaydedicilerin bit bit görevleri aşağıda verilmiştir.

ADCON0 REGISTER (ADDRESS 1Fh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/DONE	—	ADON bit 0

ADCS1, ADCS0 : Frekans kaynağı seçim bitleridir, ayrıntılı bilgi aşağıdaki tablodadır.

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	Fosc/2
0	01	Fosc/8
0	10	Fosc/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	Fosc/4
1	01	Fosc/16
1	10	Fosc/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

CHS2, CHS1, CHS0 : Kanal seçme bitleridir, ayrıntılı bilgi aşağıdaki tablodadır.

000 = Channel 0 (AN0)
 001 = Channel 1 (AN1)
 010 = Channel 2 (AN2)
 011 = Channel 3 (AN3)
 100 = Channel 4 (AN4)
 101 = Channel 5 (AN5)
 110 = Channel 6 (AN6)
 111 = Channel 7 (AN7)

GO/DONE (ADGO) : ADC çevrimini başlatan bittir, çevrim bitince 0 olur.

ADON : ADC birimini açan bittir.

ADCON1 REGISTER (ADDRESS 9Fh)

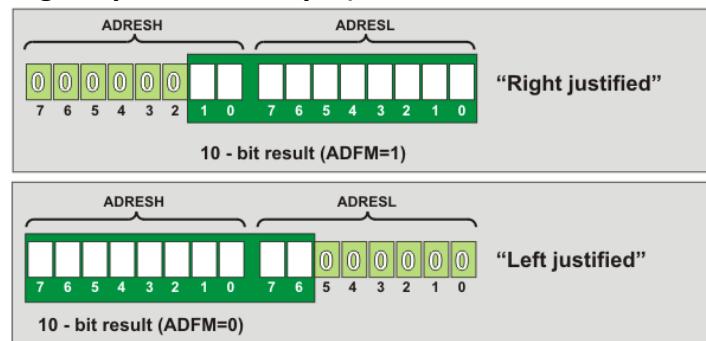
R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0

bit 7

bit 0

ADFM : Sonucun şekil-46'da görüleceği gibi ADRESH ve ADRESL kaydedicilerinde sağa mı sola mı yaslanacağına bu bit karar verir.

(1: Sağda dayalı, 0: Sola dayalı)



Şekil 46 – ADFM'in Görevi

ADCS2 : Yardımcı frekans seçme biti.

PCFG3, PCFG2, PCFG1, PCFG0 : Portların görevlerini ayarlayan bitler. Ayrıntı aşağıdaki Şekil-47'de gözükmektedir.

PCFG <3:0>	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	Vss	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	Vss	7/1
0010	D	D	D	A	A	A	A	A	VDD	Vss	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	Vss	4/1
0100	D	D	D	D	A	D	A	A	VDD	Vss	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	Vss	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	Vss	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	Vss	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	Vss	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A = Analog input D = Digital I/O

C/R = # of analog input channels/# of A/D voltage references

Şekil 47 – PCFG3, PCFG2, PCFG1, PCFG0'in Görevi

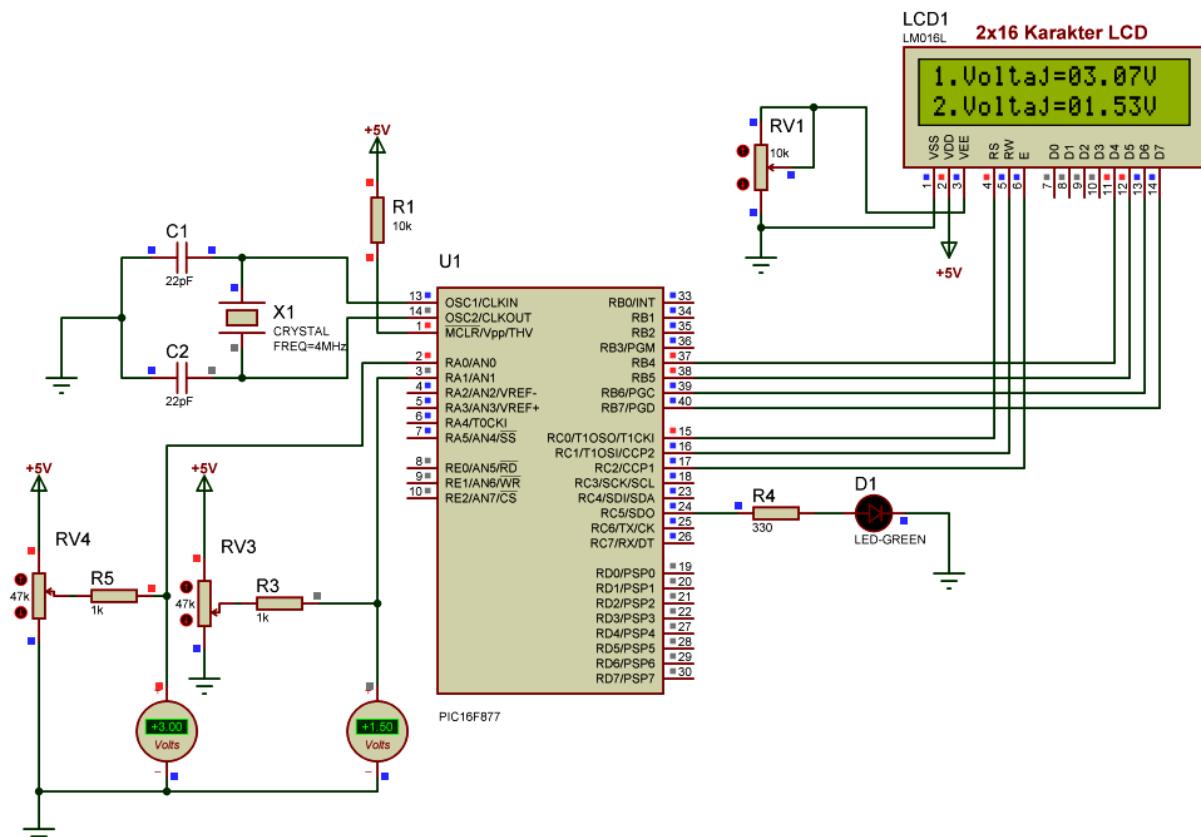
ADC birimi kesmeleri ise ADIE ve ADIF bitleri ile kontrol edilir.

ADC birimini çalıştırılmak için aşağıdaki adımlar izlenir;

- Hangi pin kullanılacaksa o seçilir,
- ADC modülü açılır,
- Kesme kullanılacaksa, gerekli birimler ayarlanır,
- Yaklaşık 20us beklenir,
- Çevrim başlatılır (ADGO=1 veya GODONE=1),
- Çevrimin bitmesi beklenir,
- ADRESH ve ADRESL, ADFM bitinin durumuna göre okunur.

4.2.1) İlk ADC Uygulaması

ADC biriminin nasıl çalıştığını öğrendikten sonra ilk uygulamamızda ilk iki analog kanalına bağlı potlardan okunan 0-5V arasındaki voltajların LCD'de görünmesi ve kesmeden faydalananarak her çevrim kesimine gidildiğinde RC5'e bağlı ledin yanıp sönmesi amaçlanmaktadır. Şimdi şekil-48'deki devremizi çizelim ve gerekli hesaplamalarımızı yapalım.



Şekil 48– İlk ADC Denemesi

16f877'de ADC 10 bitlik derinliğe sahiptir, referans voltajımızın maksimum 5V olduğunu düşünürsek;

$5/1024 = 0.0048 = 0.005$ yapar. Virgülden sonra iki basamak görmek istiyorsak bu değeri 3 kere sola ötelediğimeli yani 0.5 yapmalıyız. Böylelikle değerleri sırasıyla 1000'e, 1000'e bölümden kalanı 100'e, 100'e bölümden kalanı 10'a sonunda da 10'a bölümden kalanı alarak sağlayabiliriz. Böylelikle ondalıklı sayıları da rahatça görebiliriz. Tüm bunları yapan C kodu ise aşağıdaki şekilde olacaktır.

```

#include <htc.h>
#include "delay.h" // Gecikme kütüphanesi tanımlanıyor
#include "lcd.h"   // LCD kütüphanesi tanımlanıyor

int voltaj_1;
int voltaj_2;
char i=1;

void main(void)           // Ana fonksiyon alanı
{
    TRISA=0x03;          // Analog giriş için
    TRISB=0x00;          // LCD için çıkış
    TRISC=0x00;
    PORTB=0x00;
    PORTC=0x00;

    PCFG3=0;             // AN0 ve AN1 analog
    PCFG2=1;
    PCFG1=0;
    PCFG0=0;

    ADFM=1;              // Sağa dayalı yazılıyor
    ADON=1;               // ADC açılıyor

    ADIF=0;               // ADC bayrağı temizleniyor
    ADIE=1;               // ADC kesmesi izni veriliyor
    PEIE=1;               // Genel ve yardımcı kesme izinleri veriliyor
    GIE=1;

    lcd_init();            // LCD ilk ayarları yapılıyor

    lcd_yaz("1.Voltaj=");
    lcd_gotoxy(2,1);
    lcd_yaz("2.Voltaj=");
    for(;;)
    {
        CHS2=0;           // AN0 seçiliyor
        CHS1=0;
        CHS0=0;
        DelayUs(25);
        ADGO=1;           // Çevrim başlatılıyor
        while(ADGO)
        voltaj_1=(int)((ADRESH*256+ADRESL)/2); // Hesaplama yapılıyor
        lcd_gotoxy(1,10); // Okunan değer LCD'ye yazılıyor
        veri_yolla(voltaj_1/1000+48);
        veri_yolla((voltaj_1%1000)/100+48);
        veri_yolla('.');
        veri_yolla((voltaj_1%100)/10+48);
        veri_yolla(voltaj_1%10+48);
        veri_yolla('V');
        CHS2=0;           // AN1 seçiliyor
        CHS1=0;
        CHS0=1;
        DelayUs(25);
        ADGO=1;           // Çevrim başlatılıyor
        while(ADGO)
        voltaj_2=(int)((ADRESH*256+ADRESL)/2); // Hesaplama yapılıyor
        lcd_gotoxy(2,10); // Okunan değer LCD'ye yazılıyor
        veri_yolla(voltaj_2/1000+48);
        veri_yolla((voltaj_2%1000)/100+48);
        veri_yolla('.');
    }
}

```

```

        veri_yolla((voltaj_2%100)/10+48);
        veri_yolla(voltaj_2%10+48);
        veri_yolla('V');
    }

}

static void interrupt
led_yaz_son(void)
{
    if(ADIF)           // Çevrim bitiş kesmesi bekleniyor
    {
        i=!i;          // Her kesme de değil alınıyor
        RC5=i;          // Değer RC5'e aktarılıyor
        ADIF=0;            // Kesme bayrağı sıfırlanıyor
    }
}

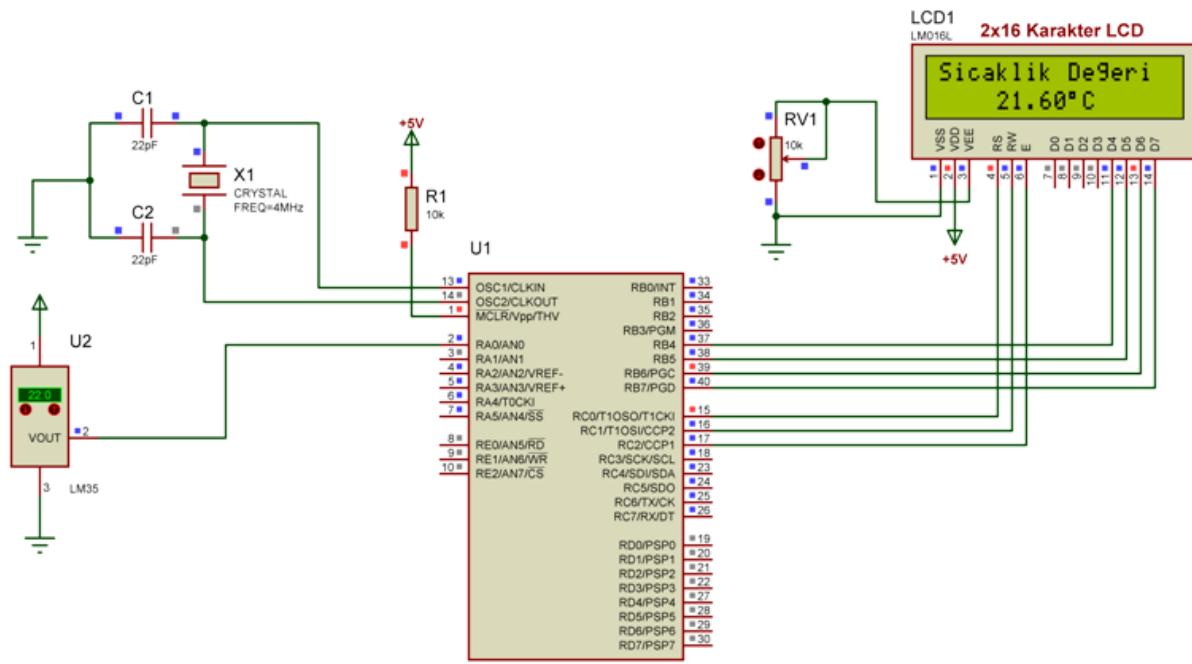
```

ADC işlemlerini belirttiğimiz sırada yaptıktan sonra devremizi çalıştırduğumızda led'in sürekli yanıp söndüğünü göreceksiniz. Bu çevrim bitiş kesmesinin doğru şekilde çalıştığını göstermektedir. Dilerseniz, ADC çevrim bitişinden sonra istediğiniz işlemi kesme de yapabilirsiniz.

4.2.2) LM35 Uygulaması

LM35, 3-4 derece hata payına sahip, kullanım kolaylığı itibariyle hassasiyet gerekmeyen sıcaklık uygulamalarında sıkça tercih edilen bir sıcaklık sensörüdür. Sensör genel itibariyle her 1 birimlik sıcaklık değişiminde 10mV voltaj değişimine giderek ölçüm yapmamızı sağlar. 16f877a'nın 10 bit derinliğindeki ADC birimi bunu algılamak için oldukça yeterlidir.

Bu uygulamamızda LM35 birimini AN0 girişine bağlayıp sıcaklık ölçümü yapılacak ve ölçülen değer LCD'ye yansıtılacaktır. Uygulamadan önce şekil-49'daki devreyi çiziyoruz.



Şekil 49 – LM35 Uygulaması

Derece işaretini LCD'nin ASCII tablosundan baktığımızda 0xDF ifadesinin bunu sağladığı görülebilir. Bu dikkate alınarak yazılan Hi-Tech kodu aşağıdaki gibi olacaktır.

```
#include <htc.h>
#include "delay.h"           // Gecikme kütüphanesi tanımlanıyor
#include "lcd.h"             // LCD kütüphanesi tanımlanıyor

void main(void)             // Ana fonksiyon alanı
{
    int sicaklik;
    TRISA=0x03;              // Analog giriş için
    TRISB=0x00;              // LCD için çıkış
    TRISC=0x00;
    PORTB=0x00;
    PORTC=0x00;

    PCFG3=1;                 // AN0 analog
    PCFG2=1;
    PCFG1=1;
    PCFG0=0;

    ADFM=1;                  // Sağa dayalı yazılıyor
    ADON=1;                   // ADC açılıyor

    lcd_init();               // LCD ilk ayarları yapılıyor

    lcd_yaz("Sicaklik Degeri");
    for(;;)
    {
        CHS2=0;                // AN0 seçiliyor
        CHS1=0;
        CHS0=0;
        DelayUs(25);
        ADGO=1;                 // Çevrim başlatılıyor
        while(ADGO);
        sicaklik=(int)((ADRESH*256+ADRESL)*48); // Hesaplama yapılıyor
        lcd_gotoxy(2,5);         // Okunan değer LCD'ye yazılıyor
        veri_yolla(sicaklik/1000+48);
        veri_yolla((sicaklik%1000)/100+48);
        veri_yolla('.');
        veri_yolla((sicaklik%100)/10+48);
        veri_yolla(sicaklik%10+48);
        veri_yolla(0xDF);        // Derece işaretini oluşturuyor
        veri_yolla('C');
    }
}
```

Gördüğü gibi bu şekilde herhangi bir ortamın, kötü de olsa, sıcaklığı kolaylıkla okunup LCD ekrana yansıtılabilir.

Bu bölümde de gördüğümüz gibi kütüphanelerin önemi bir kez daha vurgulanmıştır, ayrıca matematiksel hesapların ve hangi kaydedicilerin nerelerde kullanılacağını bilmek de işlemleri çok kolaylaştırır.

BÖLÜM 5 – DAHİLİ EEPROM

5.1) Hi-Tech'te Dahili EEPROM İşlemleri

EEPROM, Electronically Erasable Programmable Read-Only Memory yani elektriksel olarak silinip yazılabilen hafıza birimidir. Pic16f877a'nın içinde 256 byte yani 256x8 bit'lik EEPROM bulunmaktadır. Kullanacağımız işlemlerdeki bilgilerin, enerji kesildiğinde de korunmasını istiyorsak pic'in EEPROM birimini kullanabiliriz. Eğer kaydedeceğimiz değerler 256byte'tan büyük ise harici EEPROM'lar kullanarak istediğimizi yine gerçekleştirebiliriz.

Pic içerisinde bulunan EEPROM birimini EECON1 ve EECON2 kaydedicileri kontrol eder. EECON2 aslında fiziksel olarak bulunmayan bir birimdir. EECON1 kaydedicisindeki bitlerin görevleri aşağıda verilmiştir.

	R/W (x)			R/W (x)	R/W (0)	R/S (0)	R/S (0)	Features
EECON1	EEPGD	-	-	WRERR	WREN	WR	RD	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

- EEPGD** : Eeprom seçme biti (1: Program Eeprom, 0: Data Eeprom)
WRERR : Eeprom yazma hatası var (1: Hata var, 0: Hata yok)
WREN : Eeprom yazım izni verme biti (1: İzin ver, 0: İzin verme)
WR : Yazma işlemini başlatma biti (1: Yazma işlemini başlat, 0: Yazma işlemi tamamlandı)
RD : Okuma işlemini başlatma biti (1: Okuma işlemini başlat, 0: Okuma işlemi tamamlandı)
EEIE : Eeprom yazım/okuma bitti kesmesi izin biti
EIF : Eeprom yazım/okuma bitti kesmesi bayrağı

Datasheet'te belirtilen ifadelere göre Eeprom'u okumak için aşağıdaki ifadeler sırasıyla yapılmalıdır.

- Öncelikle EEADR kaydedicisine 8bitlik adres yazılır.
- RD=1 yapılarak yazım işlemi komutu verilir
- Okunan değer EEDATA kaydedicisine gelir

Datasheet'te belirtilen ifadelere göre Eeprom'a yazmak için aşağıdaki ifadeler sırasıyla yapılmalıdır.

- EEADR kaydedicisi ile Eeprom boyutu aşılmayacak şekilde adres belirtilir
- EEDATA kaydedicisi ile Eeprom'a yazılacak bilgi belirtilir
- WREN=1 ile yazma izni verilir
- EECON2=0x55 ve ardından EECON2=0xAA yapılım zorundadır
- WR=1 ile yazma işlemi başlatılır
- WR bitinin 0 olması beklenir
- WREN=0 yapılarak yazım engellenir

Bu işlemleri kütüphane haline getirerek istediğimiz zaman Eeprom işlemi yapacağımızda bunu kullanabiliriz. Yukarıdaki işlemleri yapacak **eprom.h** kütüphane başlığı aşağıdaki gibi olacaktır.

```

/*
 * www.FxDev.org
 * EEPROM Kullanım Klavuzu
 * data_eeprom_write(address, data); ile eepromun istenilen adresine
istenilen bilgi yazılır.
 * data_eeprom_read(address); ile eepromun istenilen adresi okunur.
 * www.FxDev.org
 */

extern void data_eeprom_write(unsigned char address, unsigned char data);
extern unsigned char data_eeprom_read(unsigned char address);

```

Fonksiyonları belirttiğimiz şekilde işletecek eeprom.c kodu ise aşağıdaki gibi olacaktır.

```

#include <pic.h>
#include "eeprom.h"
#include "delay.h"

void data_eeprom_write(unsigned char address, unsigned char data)
{
    EEADDR=address;
    EEDATA=data;

    WREN=1;
    EECON2 = 0x55;
    EECON2 = 0xAA;

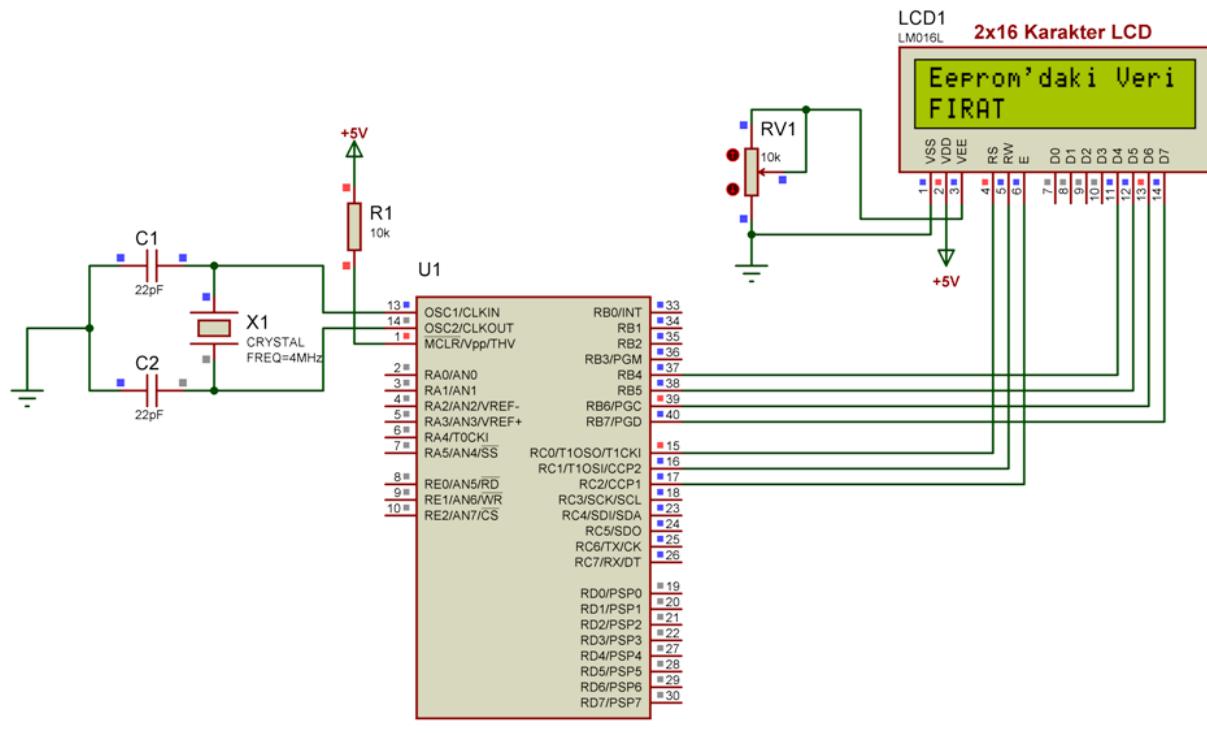
    WR=1;
    while (WR) ;
    WREN=0;
}

unsigned char data_eeprom_read(unsigned char address)
{
    EEADDR=address;
    RD=1;
    return (EEDATA);
}

```

5.1.1) Eeprom Uygulaması

Eeprom uygulamasında eeproma yazdığımız ismimizin LCD'ye yansıtılması sağlanacaktır. Bunun öncelikle şekil-50'deki devreyi çiziyoruz.



Sekil 50 –Eeprom uygulaması

Şekil-50'de çalışması gözüken devrede eeprom hafızasını görmek için Proteusta pause tuşuna basıp ve pic'e sağ tıklanarak, PIC CPU/EPROM Memory kısmına tıklanmalıdır. Eeprom hafızası şekil-51'de gözükmektedir.

PIC CPU EPROM Memory - U1	
00	46 49 52 41 54 00 00 00 FIRAT...
08	00 00 00 FF FF FF FF FF
10	FF FF FF FF FF FF FF FF
18	FF FF FF FF FF FF FF FF
20	FF FF FF FF FF FF FF FF
28	FF FF FF FF FF FF FF FF
30	FF FF FF FF FF FF FF FF
38	FF FF FF FF FF FF FF FF
40	FF FF FF FF FF FF FF FF
48	FF FF FF FF FF FF FF FF
50	FF FF FF FF FF FF FF FF
58	FF FF FF FF FF FF FF FF
60	FF FF FF FF FF FF FF FF
68	FF FF FF FF FF FF FF FF
70	FF FF FF FF FF FF FF FF
78	FF FF FF FF FF FF FF FF
80	00 00 FF FF FF FF FF FF
88	FF FF FF FF FF FF FF FF
90	FF FF FF FF FF FF FF FF
98	FF FF FF FF FF FF FF FF
A0	FF FF FF FF FF FF FF FF
A8	FF FF FF FF FF FF FF FF
B0	FF FF FF FF FF FF FF FF
B8	FF FF FF FF FF FF FF FF
C0	FF FF FF FF FF FF FF FF
C8	FF FF FF FF FF FF FF FF
D0	FF FF FF FF FF FF FF FF
D8	FF FF FF FF FF FF FF FF
E0	FF FF FF FF FF FF FF FF
E8	FF FF FF FF FF FF FF FF
F0	FF FF FF FF FF FF FF FF
F8	FF FF FF FF FF FF FF FF

Sekil 51 –Eeprom hafızası

Tüm bu işlemleri yapan C kodu ise aşağıdaki gibidir.

```
#include <htc.h>
#include "delay.h"      // Gecikme kütüphanesi tanımlanıyor
#include "lcd.h"        // LCD kütüphanesi tanımlanıyor
#include "eeprom.h"      // Eeprom kütüphanesi tanımlanıyor

void main(void)          // Ana fonksiyon alanı
{
    TRISB=0x00;           // LCD için çıkış
    TRISC=0x00;
    PORTB=0x00;
    PORTC=0x00;

    lcd_init();           // LCD ilk ayarları yapılıyor

    data_eeprom_write(0x00, 'F'); // Adımız eeproma yazılıyor
    data_eeprom_write(0x01, 'I');
    data_eeprom_write(0x02, 'R');
    data_eeprom_write(0x03, 'A');
    data_eeprom_write(0x04, 'T');

    lcd_yaz("Eeprom'daki Veri");
    lcd_gotoxy(2,1);
    veri_yolla(data_eeprom_read(0x00));
    // Eepromdaki bilgi LCD'ye yazdırılıyor
    veri_yolla(data_eeprom_read(0x01));
    veri_yolla(data_eeprom_read(0x02));
    veri_yolla(data_eeprom_read(0x03));
    veri_yolla(data_eeprom_read(0x04));
    for(;;);
}
```

Göründüğü üzere Eeprom'a yazı yazmak datasheet'te söylenildiği yapıldıktan sonra oldukça kolaydır. Bu işlemden sonra eğer Eeprom'a kayıt yaptığınız pic'e başka programlar (Eeprom'a yazım yapmayan programlar) yazdığınızda dahi, Eeprom'daki isminiz aynen korunacaktır.

BÖLÜM 6 – RS232 SERİ İLETİŞİM İŞLEMLERİ

6.1) Hi-Tech'te RS232 Seri İletişim İşlemleri

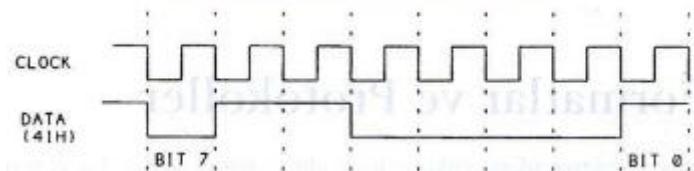
Mikrodenetleyiciler ile diğer mikrodenetleyiciler, cihazlar, bilgisayarlar arasında iletişim kurmak istenildiğinde kullanılan metodlardan bir kaçı seri ve paralel iletişimdir.

Paralel iletişimde 8 bit bilgi, 8 adet data kablosuyla gerekli aygıtlara bağlanır ve iletişim sağlanır. Fakat bu uzun mesafelerde fazla kablo kullanımına yol açmaktadır. Paralel iletişimden yavaşmasına karşın bu sorunu aşmak için kullanılan seri iletişim protokolünde ise sadece 2 adet veri kablosu kullanılır ve bilgi yine bu iki kablo üzerinden alınır veya gönderilir.

Seri haberleşme protokolü kendi içerisinde senkron ve asenkron olmak üzere ikiye ayrılır;

- **Senkron Seri İletişim:** Şekil-52'de görülen senkron seri iletişimde, yardımcı bir saat işaretini kullanılır. Gönderilecek veya alınacak bilgi bu saat işaretine uyarak gelmek veya gönderilmek zorundadır. Hızlı iletişim yapılmayıcağı zaman bu iletişim metodu kullanılır.

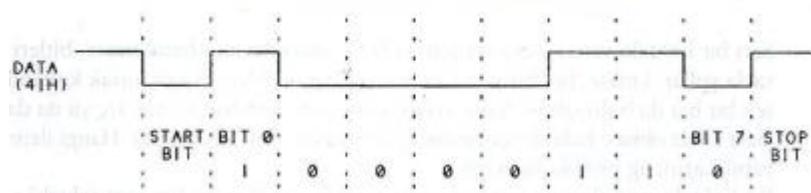
ASCII "A" (41h) karakterinin传递



Şekil 52 –Senkron Seri İletişim

- **Asenkron Seri İletişim:** Şekil-53'te görülen asenkron seri iletişim, yardımcı bir saat işaretini yerine başlangıç, bitiş bitleri ile kontrol edilir. Herhangi bir anda iletişim başlatılabilir veya durdurulabilir. İletişim bittiği zaman hat IDLE denilen duruma geçer. Senkron seri iletişime göre yavaştır.

ASCII "A" (41h) karakterinin传递



Şekil 53 –Asenkron Seri İletişim

Biz bu bölümde çok sık kullanılan asenkron seri iletişim metodunu kullanarak uygulama yapacağız.

Asenkron seri iletişime geçmeden önce bu metodla ilgili birkaç terimi öğrenmekte faydalıdır. Bu terimler aşağıda açıklanmıştır.

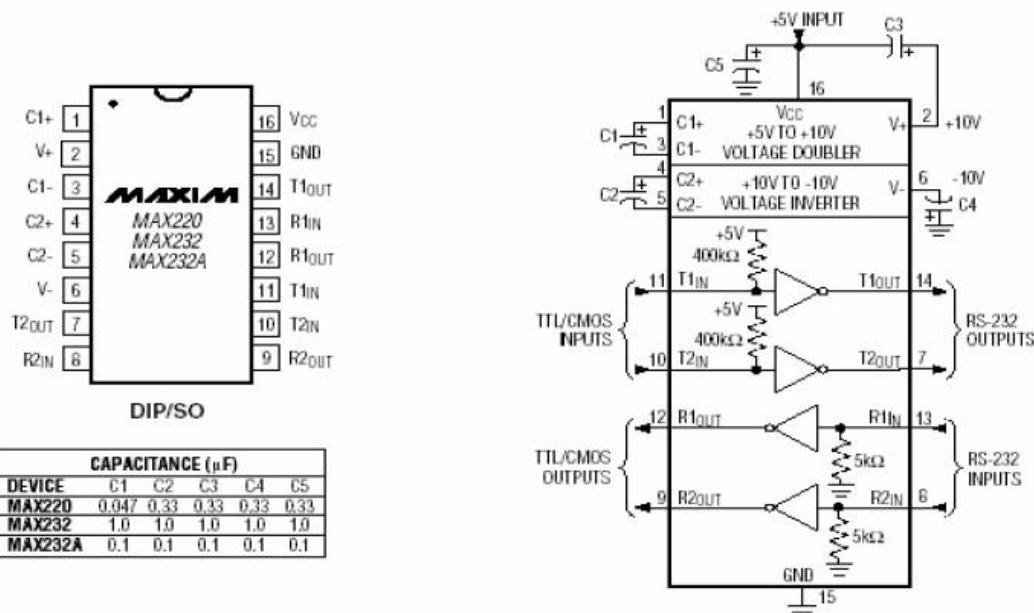
İletişim Hızı : Verici ve alıcı belirli iletim hızlarında çalışırlar. Bu konuda standart haline gelmiş iletim hızları mevcuttur.

Başlangıç Biti : Gönderilecek bilgiden önce iletişimini başlatmayı bildiren bittir.

Veri Uzunluğu: Gönderilecek verinin uzunluğudur, genel bir sabiti olmamasına karşın kolaylığı nedeniyle genellikle 8 bit uzunluğundadır.

Eşlik Biti : Veri iletimi esnasında kullanılan bu bit, veri bozukluğu kontrolünde kullanılır.
Dur Biti : Veri iletiminin bittiğini, alıcıya bildiren bitir.

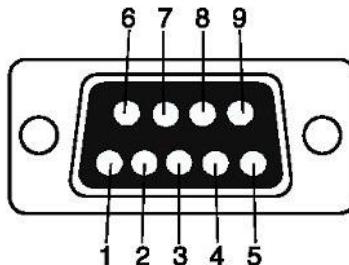
- **Max232 Entegresi:** Bilgisayarlardaki RS232 birimi logic-0 için -3V..-12V, logic-1 ise +3V..+12V arasında gerilimler üretmektedir. Oysa pic gibi TTL mantığıyla çalışan aygıtlar için bu değerler geçerli değildir. Bu değerlerin 0V..+5V aralığına indirmek için özel gerilim dengeleyici entegreler kullanılmaktadır. Günümüzde bu entegrelerin en çok kullanılanı ise Maxim firmasının ürettiği Max232 entegresidir. Bu entegrenin iç yapısı entegreyle birlikte kullanılması gereken kapasitör değerleri sekill-54'te gözükmektedir.



Sekil 54 –Max232 Entegresi

Max232 entegresini bilgisayara bağlamak için ise şekil-55'te görülen konektör kullanılır.

<u>Pin</u>	<u>Signal Description</u>
1	DCD
2	TD
3	RD
4	DTR
5	Signal GND
6	DSR
7	RTS
8	CTS
9	contact closure



Sekil 55 -RS232 Konektörü

Bizim kullanacağımız Pic16f877a içerisinde hazır olarak bulunan bir adet seri iletişim birimi bulunmaktadır. **RC6/TX** ve **RC7/RX** ucları ise bu için özel olarak ayrılmış pinlerdir.

Seri iletişimini kullanmak için pic'de **TXSTA** ve **RCSTA** kaydedicileri ayarlanmalıdır. Bu kaydedicilerin görevleri bit bit aşağıda görülmektedir.

TXSTA	R/W (0)	R/W (0)	R/W (0)	R/W (0)		R/W (0)	R (1)	R/W (0)	Features
	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4		Bit 2	Bit 1	Bit 0	

CSRC : Senkron modda, (1: Master, 0: Slave)

TX9 : İletişim modu seçme biti (1: 9 Bitlik İletişim, 0: 8 Bitlik İletişim)

TXEN : Transfere izin verme biti (1: Transfere izin ver, 0: Transfere izin verme)

SYNC : 1: Senkron mod, 0: Asenkron mod

BRGH : Hızlı/yavaş boudrate seçme biti (1: Hızlı mod, 0: Yavaş mod)

TRMT : İletim kaydedicisi boş/dolu biti (1: Dolu, 0: Boş)

TX9D : 9 bitlik modda parity veya 9. bit

RCSTA	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R (0)	R (0)	R (x)	Features
	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

SPEN : 1: Seri port açık, 0: Seri port kapalı

RX9 : Alım modu seçme biti (1: 9 bitlik iletişim, 0: 8 bitlik iletişim)

SREN : Senkron modda (1: Tek alım açık, 0: Tek alım kapalı)

CREN : Asenkron modda (1: Devamlı almaya izin ver, 0: Devamlı almaya izin verme)
Senkron modda (1: CREN sıfırlanıncaya kadar alıma izin ver, 0: Almayı kes)

ADDEN: Asenkron modda RX9=1 ise (1: 8bit yüklemeye git, 9. bit parity olarak kullanılabilir)

FERR : Alımda hata algılaması biti (1: Hata var, 0: Hata yok)

OERR : Üst üste yazım hatası kontrolü (1: Hata var, 0: Hata yok)

RX9D : 9 bitlik modda parity veya 9. bit

Bu kaydedicilerden başka seri iletişim hız ayarı yapmak için SPBRG kaydedicisine ihtiyaç vardır. Bu 8 bitlik kaydediciye yüklenenek değerler ile standart haline gelmiş hızlar ayarlanır. Bu kaydediciye yüklenenek değer aşağıdaki formülle hesaplanır.

SYNC	BRGH = 0 (Low Speed)	BRGH = 1 (High Speed)
0	(Asynchronous) Baud Rate = Fosc/(64 (X + 1)) (Synchronous) Baud Rate = Fosc/(4 (X + 1))	Baud Rate = Fosc/(16 (X + 1)) N/A
1		

Legend: X = value in SPBRG (0 to 255)

Gördüğü üzere belirlenen X değerlerine göre boudrate hızları ayarlanır. Genelde düşük hızlarda seri iletişim kurulmadığından biz hızlı iletişim metodunu kullanacağız. Bunun için gerekli X değerini şekil-56'dan bulup SPBRG registeri içerisinde yükleyebilir ya da bizim için gerekli X'i bulma formülüümüz olan $X=(FOSC/(16UL * BAUD) -1)$ eşitliğini kullanabiliriz.

BAUD RATES FOR ASYNCHRONOUS MODE (BRGH = 1)

BAUD RATE (K)	Fosc = 20 MHz			Fosc = 16 MHz			Fosc = 10 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-	-	-	-
1.2	-	-	-	-	-	-	-	-	-
2.4	-	-	-	-	-	-	2.441	1.71	255
9.6	9.615	0.16	129	9.615	0.16	103	9.615	0.16	64
19.2	19.231	0.16	64	19.231	0.16	51	19.531	1.72	31
28.8	29.070	0.94	42	29.412	2.13	33	28.409	1.36	21
33.6	33.784	0.55	36	33.333	0.79	29	32.895	2.10	18
57.6	59.524	3.34	20	58.824	2.13	16	56.818	1.36	10
HIGH	4.883	-	255	3.906	-	255	2.441	-	255
LOW	1250.000	-	0	1000.000		0	625.000	-	0

BAUD RATE (K)	Fosc = 4 MHz			Fosc = 3.6864 MHz		
	KBAUD	% ERROR	SPBRG value (decimal)	KBAUD	% ERROR	SPBRG value (decimal)
0.3	-	-	-	-	-	-
1.2	1.202	0.17	207	1.2	0	191
2.4	2.404	0.17	103	2.4	0	95
9.6	9.615	0.16	25	9.6	0	23
19.2	19.231	0.16	12	19.2	0	11
28.8	27.798	3.55	8	28.8	0	7
33.6	35.714	6.29	6	32.9	2.04	6
57.6	62.500	8.51	3	57.6	0	3
HIGH	0.977	-	255	0.9	-	255
LOW	250.000	-	0	230.4	-	0

Şekil 56 – Çeşitli Baudrate'ler için SPBRG Değerleri

Son olarak **SPBRG** değerini belirledikten sonra **RCIF ile alım, TXIF ile de gönderim kesme bayrakları** kontrolleri ile gönderim veya alım işinin bittiği kontrol edilerek **RCREG kaydedicisi ile alınan bilgi, TXREG ile de gönderilmek istenen bilgiler** istenilen şekilde kullanılabilir.

Biraz karışık görünse de uygulamada tüm bu söylediğimiz oldukça kolay bir şekilde yapılmaktadır. Seri iletişim yaparken öncelikle iletişim metotlarını (Kaç bitlik iletişi metodу, kaç baudrate hızında vb.) belirlemeli, daha sonra ise kesme bayrakları kontrolüyle iletişim kurmalıyız. Daha sonra kullanmak için seri iletişimini de kütüphane haline getirirsek usart.h dosyamız aşağıdaki gibi olacaktır.

```
/*
 *          www.FxDev.org
 *          USART Kullanım Klavuzu
 *  USART_init(); ile USART'in ilk ayarlarını yap
 *  putch('A'); şeklinde tek byte veri gönder
 *  A=getch(); ile tek byte'luk veri al
 *          www.FxDev.org
 */
#define BAUD 19200 //Baudrate
#define FOSC 4000000L //Kristal hızı
#define NINE 0 //9 bit iletişim kullanılacaksa 1 yapılmalı
#define HIGH_SPEED 1 //Hızlı iletişim yapılacaksa 1 yapılmalı

#define RX_PIN TRISC7 //Seri port pinleri
#define TX_PIN TRISC6
```

```

#define DIVIDER ((int)(FOSC/(16UL * BAUD) -1))

extern void usart_init(void);
extern void putch(unsigned char byte);
extern unsigned char getch(void);

```

Göründüğü gibi boudrate hesabı ($FOSC/(16UL * BAUD) -1$) şeklinde kütüphane başlığında tanımlanmıştır. Ayrıca kullanılan kristal de burada örneğin 4MHz için 4000000L veya 20MHz için 2000000L şeklinde tanımlanır. Standart haline gelmiş boudrate'ler ve seri port pinleri de burada tanımlanarak seri iletişim kütüphane işlemleri tamamlanır. Gerekli işlemleri yapan usart.c dosyası ise aşağıdaki gibidir.

```

#include <pic.h>
#include "stdio.h"
#include "usart.h"

void usart_init(void)
{
    unsigned char speed,nine_bits;
    unsigned int divider;

    RX_PIN = 1;           // Seri iletişim pinleri giriş olarak tanımlanıyor
    TX_PIN = 1;

    if(HIGH_SPEED==1) // Hızlı iletişim yapılacaksa
        speed=0x4;
    else               // Yaval iletişim yapılacaksa
        speed=0x00;

    if(NINE==1)          // 9 bitlik iletişim yapılacaksa
        nine_bits=0x04;
    else                // 8 bitlik iletişim yapılacaksa
        nine_bits=0x00;

    SPBRG = DIVIDER;      // Hız değeri SPBRG'ye yükleniyor
    RCSTA = (nine_bits|0x90); // Alım kaydedicisi ayarlanıyor
    TXSTA = (speed|nine_bits|0x20); // Gönderim kaydedicisi ayarlanıyor
}

void putch(unsigned char byte)
{
    // Tek byte gönder
    while(!TXIF); // Transfer tamponu boş mu

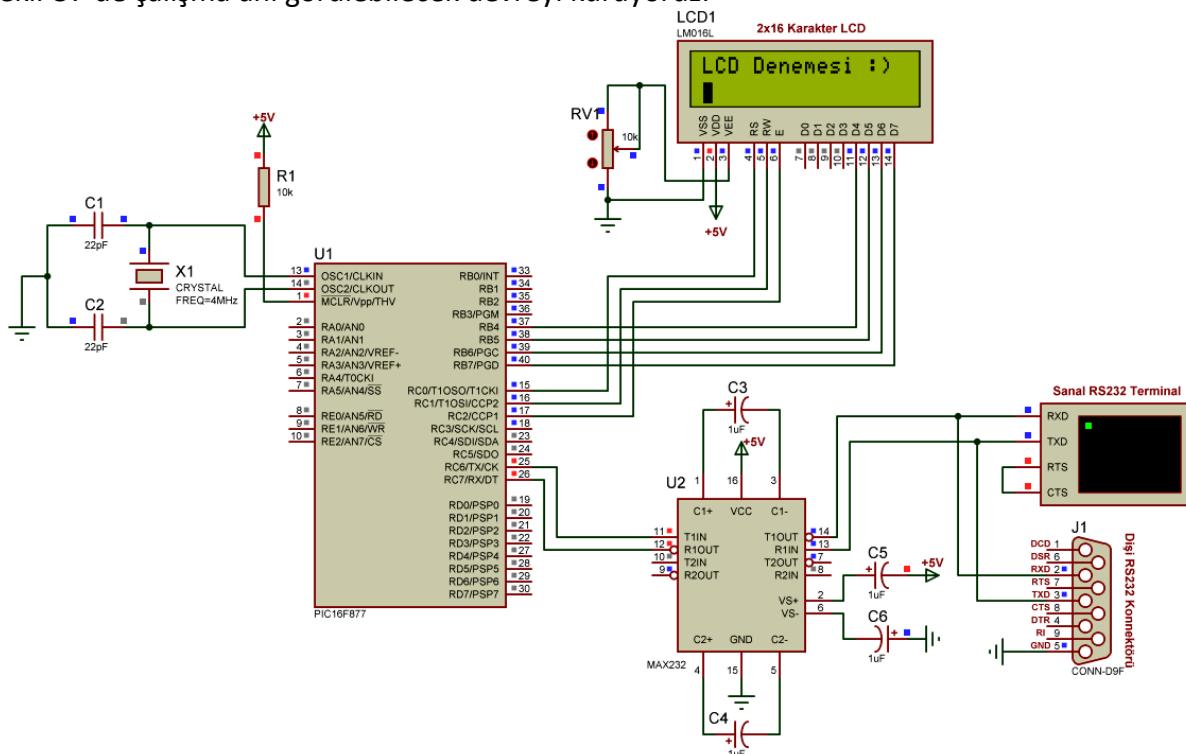
    TXREG = byte; // Tek byte gönderiliyor
}

unsigned char getch(void)
{
    // Tek byte al
    while(!RCIF); // Alım tamponu boş mu
    return RCREG; // Tek byte alınıyor
}

```

6.2) RS232 Seri İletişim Uygulaması

Kütüphanelerimizi tamamladıktan sonra şekil-57'de görülen uygulamaya gelebiliriz. **Bu uygulamada seri porttan gönderdiğimiz bilgilerin LCD'ye yazılmasını sağlayacağız.** Öncelikle şekil-57'de çalışma anı görülebilecek devreyi kuruyoruz.



Sekil 57 – RS232 Seri İletişim Uygulaması

Şekil-57'deki devrede max232'nin nasıl kullanılacağı da gösterilmiştir. Boudrate 19200 olarak kabul ettiğimizde, istediğimizi gerçekleştirecek C programı aşağıdaki gibi olacaktır.

```
#include <htc.h>
#include <stdio.h>          // printf için gerekli C standart giriş çıkış
kütüphanesi
#include "delay.h"           // Gecikme kütüphanesi tanımlanıyor
#include "lcd.h"             // LCD kütüphanesi tanımlanıyor
#include "uart.h"            // USART kütüphanesi tanımlanıyor

void main(void)             // Ana fonksiyon alanı
{
    char i,j;
    TRISB=0x00;              // LCD için çıkış
    TRISC=0xF0;              // USART pinleri giriş olarak kabul ediliyor
    PORTB=0x00;

    lcd_init();               // LCD ilk ayarları yapılıyor
    usart_init();              // USART ilk ayarları yapılıyor

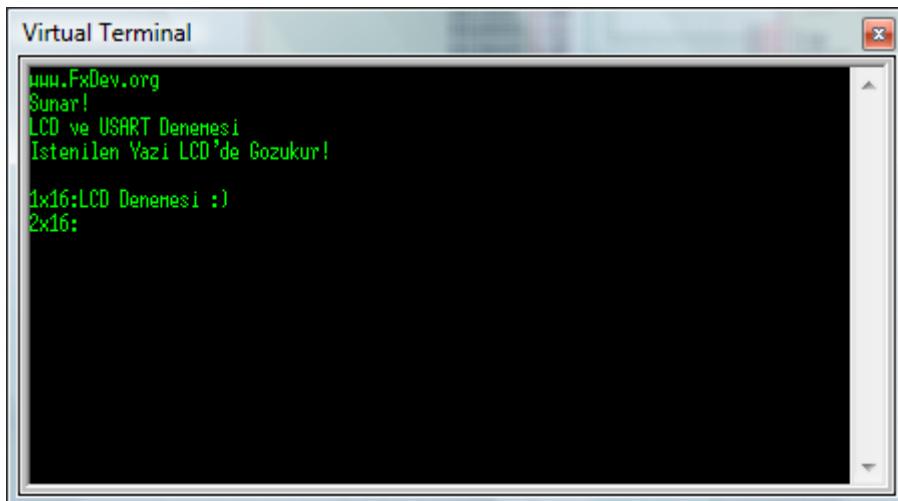
    lcd_yaz(" www.FxDev.org");
    lcd_gotoxy(2,1);
    lcd_yaz(" LCD / USART");
    DelayMs(250);DelayMs(250);DelayMs(250);DelayMs(250);
    lcd_clear();
    lcd_komut(ImlecYanSon); // Imlec yanıp sönecek
```

```

printf("www.FxDev.org\n\rSunar!");
printf("\n\rLCD ve USART Denemesi");
printf("\n\rIstenilen Yazi LCD'de Gozukur!\n\r\n\r");
printf("1x16:");
for(;;)
{
    j++;
    i=getch();           // Karekter alınıyor
    veri_yolla(i);      // LCD'ye basılıyor
    putch(i);           // Virtual terminal'e basılıyor
    if(j==16)           // İlk satıra geçiliyor
    {
        printf("\n\r2x16:");
        lcd_gotoxy(2,1);
    }
    else if(j==32)     // 2. satıra geçiliyor
    {
        veri_yolla(i);
        lcd_clear();
        printf("\n\r1x16:");
        j=0;
    }
}
}

```

Şekildeki programı yazıp Proteus'ta uygulamayı çalıştırırsanız virtual terminal ekranı şekil 58'deki gibi gözükecektir.



Şekil 58– Virtual Terminal Ekranı

Uygulamalarımızda da gördüğümüz gibi kütüphane tanımlamak işlerimizi oldukça kolay hale getirmektedir. Bu işlemi diğer piclerle de yapmak istersek sadece seri port pinlerini kütüphane başlık dosyasında tanımlamamız yeterlidir.

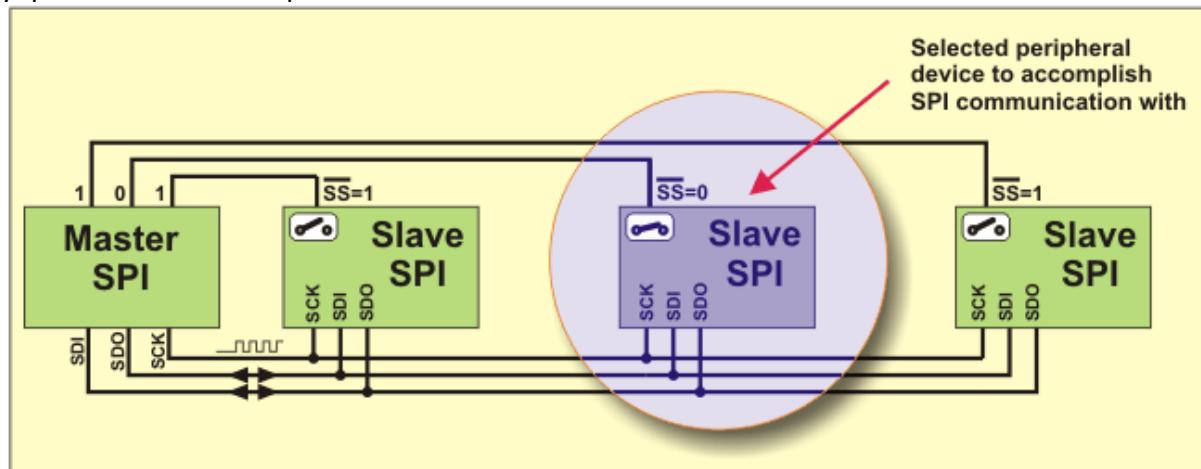
Virtual terminal programındaki görüntüyü Windows Vista öncesi işletim sistemlerinde Hyper Terminal ile ya da özel seri port yazılımları ile görmek mümkündür.

BÖLÜM 7 – SPI ve I²C VERİ İLETİŞİM İŞLEMLERİ

7.1) Hi-Tech'te SPI İletişim İşlemleri

SPI, 8 bitlik veri iletişimini yapmaya olanak sağlayan ve pic içerisinde MSSP (The Master Synchronous Serial Port) modülünde bulunan birimdir. Eeprom okuma işlemlerinde, ekran sürmelerde ve hız gerektiren bir çok işlemde SPI birimi kullanılır.

SPI birimi şekil-59'da da görüldüğü üzere 3 adet pin kullanır, eğer slave modunda iletişim yapılacaksa ekstra bir pin daha kullanılır.



Şekil 59 – Virtual Terminal Ekranı

16f877a içerisinde bir adet MSSP dolayısı ile de bir adet SPI birimi bulunur. SPI biriminin pin görevleri ise şunlardır;

SDO : Seri data çıkışı

SDI : Seri data girişi

SCK : Seri saat sinyali

Slave modunda iletişim yapılacaksa;

SS : Slave seçme pini

Pic içerisinde bulunan SPI birimini kontrol etmek için SSPSTAT ve SSPCON kaydedicileri kullanılır. Bu kaydedicilerin görevleri bit bit aşağıda verilmiştir.

SSPSTAT	R/W (0)	R/W (0)	R (0)	R (0)	R (0)	R (0)	R (0)	Features	Bit name
	SMP	CKE	D/A	P	S	R/W	UA	BF	

SMP : SDI pininden alınacak her bitin okunma zamanını belirleyen bit

0: Okuma her aktif zamanının (bit gönderme süresinin) ortasında

1: Okuma her aktif zamanının (bit gönderme süresinin) sonunda

SPI slave ise SMP=0 olmak zorundadır

CKE : 1: Transfer aktiften IDLE durumuna geçerken olacak

0: Transfer IDLE durumundan aktife geçerken olacak

5,4,3,2,1. Bitler I²C'de kullanılıyor

BF : Alım modunda (1: Alım tamamlandı (SSPBUF dolu), 0: Alım tamamlanmadı)

SSPCON	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Features
	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

- WCOL** : Gönderim yapılrken yazılıdı mı (1: Yazıldı, hata var, 0: Yazılmadı)
Başlangıcta WCOL=0 yapılmalı.
- SSPOV** : Alım yapılrken yeni alım yapıldı mı (1: Yapıldı, hata var, 0: Yapılmadı)
Başlangıcta SSPOV=0 yapılmalı.
- SSPEN** : 1: SCK, SDO, SDI ve SS pinleri SPI için ayarlandı, 0: Normal pin
- CKP** : IDLE seviyesi belirleme biti (1: IDLE yüksek seviye, 0: IDLE düşük seviye)
- SSPM3:SSPM0:** Mod seçim bitleri, aşağıda SPI için gerekli olan tanımlamalar gösterilmiştir.

SSPM3	SSPM2	SSPM1	SSPM0	Mode
0	0	0	0	SPI master mode, clock = Fosc/4
0	0	0	1	SPI master mode, clock = Fosc/16
0	0	1	0	SPI master mode, clock = Fosc/64
0	0	1	1	SPI master mode, clock = (output TMR)/2
0	1	0	0	SPI slave mode, SS pin control enabled
0	1	0	1	SPI slave mode, SS pin control disabled, SS can be used as I/O pin

SPI iletişimini yapmak için aşağıdaki adımlar sırasıyla uygulanır;

- SMP biti ayarlanır, slave modda kullanılacaksa 0 yapılır.
- SSPBUF temizlenir, böylelikle SSPBUF yeni veriler için hazır olur.
- BF=0 yapılarak SSPBUF'ın boş olduğu belirtilir.
- CKE ile transferin düşen kenarda mı yoksa yükselen kenarda mı olacağı belirlenir.
- CKP ile IDLE durumu belirlenir.
- WCOL=0 ve SSPOV=0 yapılarak başlangıcta hataların olmadığı varsayıılır.
- SSPM3, SSPM2, SSPM1 ve SSPM0 ile iletişim hızı belirlenir
- SSPEN=1 ile pinlerin SPI için kullanılacağı belirtilir

SPI birimini daha sonra da kullanmak için kütüphane haline getirirsek, kütüphane başlık dosyası **spi.h** aşağıdaki gibi olur.

```
/*
 * www.FxDev.org
 * SPI Kullanım Klavuzu
 * spi_init(); ile SPI'nin ilk ayarlarını yap
 * spi_write(0x02); şeklinde veri yaz, adres gönder
 * a=spi_read(0xFF); şeklinde veri al
 * www.FxDev.org
 */

#define IDLE      0          // IDLE durumu
#define TRANSFER  0          // IDLE'->IDLE: 0, IDLE->IDLE':1

extern void spi_init(void);
extern void spi_write(unsigned char veri);
extern unsigned char spi_read(unsigned char veri);
```

İlk yapılacak ayarlar göz önüne yazılıan spi.c ise aşağıdaki gibi olacaktır.

```
#include <pic.h>
#include "spi.h"

void spi_init(void)
{
    //SSPSTAT->SMP, CKE, BF
    //SSPCON1->WCOL, SSPOV, SSPEN, CKP, SSPM3:SSPM0

    SMP=1;           // Data 8. bitten sonra alınıyor
    SSPBUF=0;        // SSPBUF siliniyor.
    BF=0;            // SSPBUF boş

    if(TRANSFER==1) // Transfer aktiften IDLE durumuna geçerken olacak
        CKE=1;
    else           // Transfer IDLE'dan aktif durumuna geçerken olacak
        CKE=0;

    if(IDLE==1) // IDLE durumun yüksek mi düşük mü olduğu belirleniyor
        CKP=1;
    else
        CKP=0;

    WCOL=0; //Başta alım veya gönderim hatalarının olmadığı varsayılıyor
    SSPOV=0;
/*
Clock hız ayarı yapılıyor...
0101: SPI Slave, SS disabled
0100: SPI Slave, SS enabled
0011: SPI Master, Clk=TMR2/2
0010: SPI Master, Clk=Fosc/64
0001: SPI Master, Clk=Fosc/16
0000: SPI Master, Clk=Fosc/4
*/
    SSPM3=0;      // Fosc/64
    SSPM2=0;
    SSPM1=1;
    SSPM0=0;

    SSPEN=1;      //SCK, SDO, SDI, SS ser port pini oldu
}

void spi_write(unsigned char veri)
{
    BF=0;          //BF sıfırlanıyor
    SSPBUF=veri;   //BF=1, SSPBUF dolu
    while(!BF);    //SSPBUF '0' oluncaya kadar bekleniyor
}

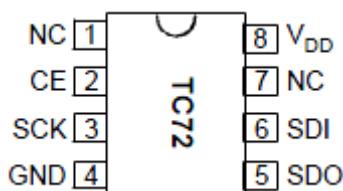
unsigned char spi_read(unsigned char veri)
{
    BF=0;          //BF sıfırlanıyor
    SSPBUF=veri;   //SCK göndermek için tanımlanmamış kod gönderiliyor
    while(!BF);

    return SSPBUF;
}
```

Burada dikkat edilmesi gereken nokta SSPBUF'a yazılan değerlerdedir. SSPBUF'a gönderilmek istenen değerler yazıldığı an SPI birimi ayarlamalarımıza göre kendi saat sinyalini, IDLE durumunu belirleyerek iletişimini kuracaktır. Alım modunda SSPBUF'a veri yazılmasının nedeni ise yeni bilginin alınması için gerekli olan saat sinyalini oluşturmaktır.

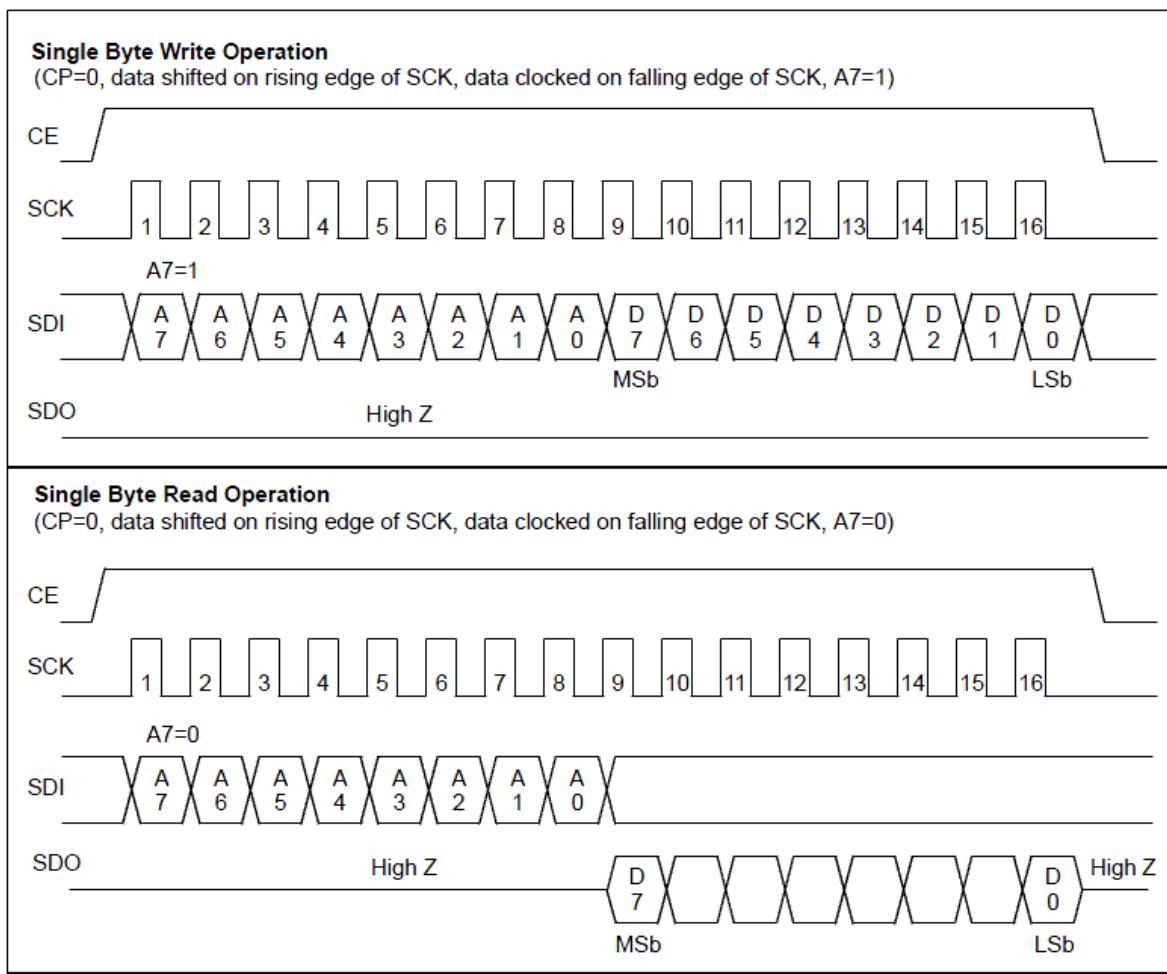
7.1.1) TC72 ile SPI Uygulaması

Bu uygulamada Microchip firmasına ait olan TC72 SPI sıcaklık sensörünü kullanacağız.
Şekil-60 da DIP soket yapısı görülebilecek TC72 sıcaklık sensörü yaklaşık 7,5MHz'a kadar çıkabilen iletişim hızına ve 10 bit çözünürlüğe sahiptir.



Şekil 60 – TC72 Pin Yapısı

TC72'nin yazma okuma zaman tablosu da şekil-61'deki gibidir. Bu tablolar datasheetten bire bir alınmıştır.



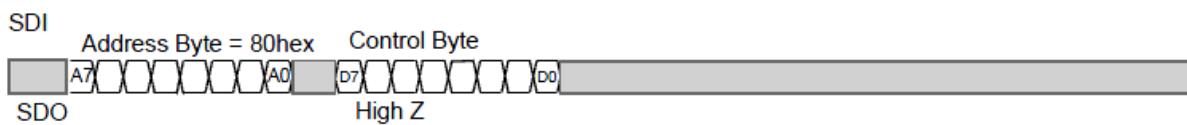
Şekil 61 – 25LC640 Zaman Diyagramları

Şekil-61'den de görüleceği üzere **TC72'nin IDLE durumu düşük seviyededir**. Tüm bit yazım gönderimleri ise düşen kenarda gerçekleşmektedir. Bunu **spi.h** kütüphane başlık dosyasında belirtmemiz gerekmektedir.

Tüm okuma ya da yazım işlemlerinden önce TC72'ye belirli komutlar gönderilmelidir. Bu komutlar şeik-62'de listelenmiştir.

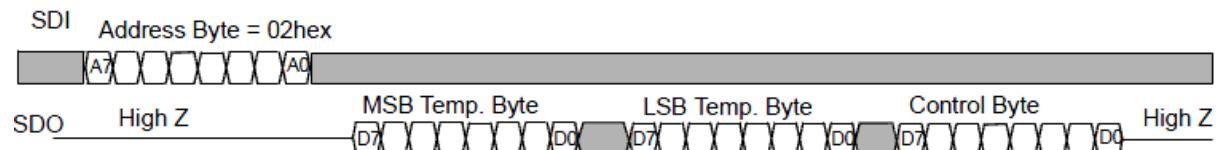
Write Operation

(CP=0, data shifted on rising edge of SCK, data clocked on falling edge of SCK, A7=1)



Read Operation

(CP=0, data shifted on rising edge of SCK, data clocked on falling edge of SCK, A7=0)

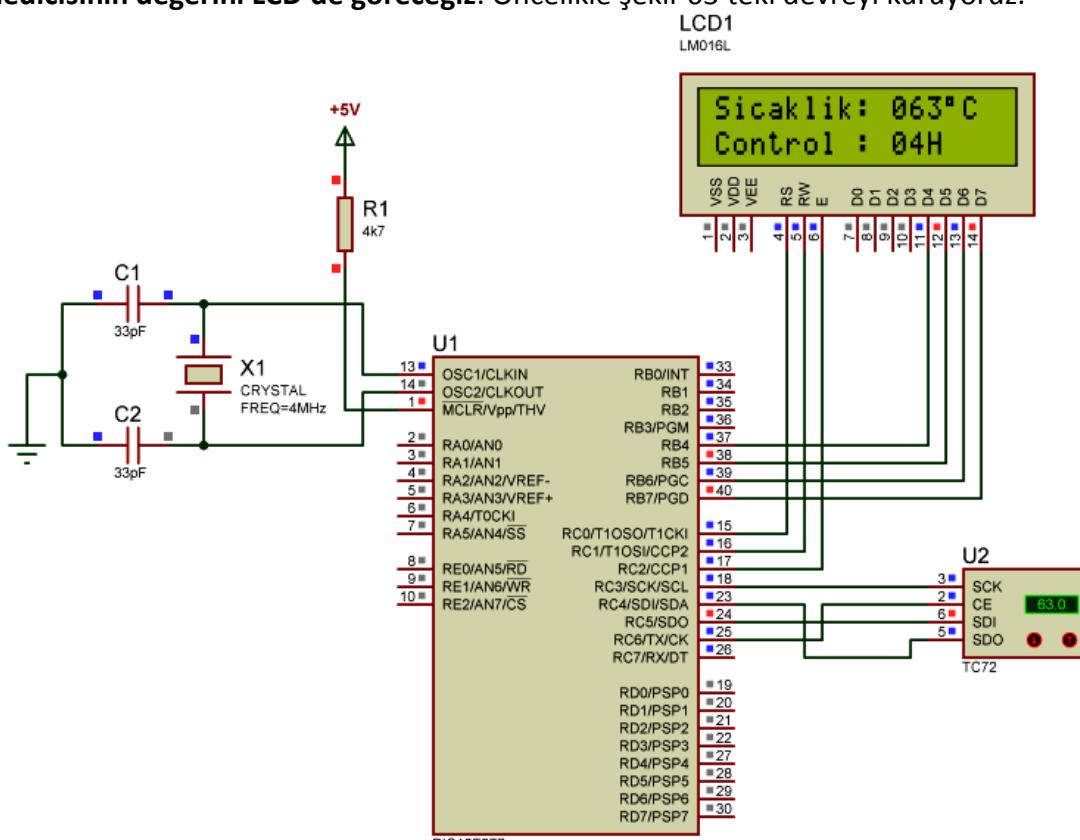


Şekil 62 – TC72 Komutları

Tüm bunları dikkate alarak TC72'den veri okumak için;

- 0x80 ile kontrol byte'ı olarak 0x04 gönderilerek sensör açılır.
- Adres bilgisi gönderilir (0x02)
- Sonra sırasıyla sıcaklık ve kontrol bilgileri alınır.

Biz bu uygulamamızda harici TC72 ile sıcaklık değerini ölçerek TC72'nin kontrol kaydedicisinin değerini LCD'de göreceğiz. Öncelikle şeik-63'teki devreyi kuruyoruz.



İstediğimizi gerçekleştiren Hi-Tech kodu ise aşağıdaki gibidir.

```
#include <htc.h>
#include "delay.h" // Gecikme kütüphanesi tanımlanıyor
#include "lcd.h"   // LCD kütüphanesi tanımlanıyor
#include "spi.h"   // SPI kütüphanesi tanımlanıyor

#define CE RC6

void tc72_init(void)      //TC72 ilk yüklemeleri yapılıyor
{
    spi_init();
    CE=1;
    spi_write(0x80);
    spi_write(0x04);
    CE=0;
}

void tc72(unsigned char *msb, unsigned char *lsb, unsigned char *control)
{
    CE=1;
    spi_write(0x02);
    *msb=spi_read(0xFF);
    *lsb=spi_read(0xFF);
    *control=spi_read(0xFF);
    CE=0;
}

void main(void)           // Ana fonksiyon alanı
{
    unsigned char onlar, ondalar, control;
    TRISB=0x00;          // LCD için çıkış
    TRISC=0x10;
    CE=0;

    lcd_init();          // LCD ilk ayarları yapılıyor
    tc72_init();          // SPI ilk ayarları yapılıyor

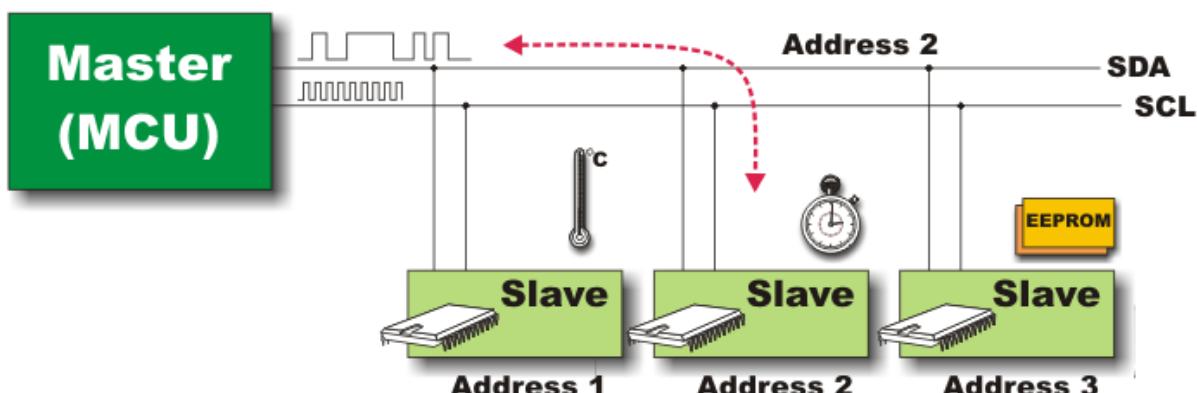
    lcd_yaz("Sicaklik:");

    for(;;)
    {
        lcd_gotoxy(1,11);
        tc72(&onlar, &ondalar, &control); // Sıcaklık ve kontrol
        veri_yolla(onlar/100+48);         // Değerleri alınıp
        veri_yolla((onlar%100)/10+48);    // LCD'ye yazdırılıyor
        veri_yolla(onlar%10+48);
        veri_yolla(0xDF);
        veri_yolla('C');
        lcd_gotoxy(2,1);
        lcd_yaz("Control : ");
        veri_yolla(control/16+48);
        veri_yolla(control%16+48);
        veri_yolla('H');
    }
}
```

Görüldüğü üzere SPI iletişimini donanım ile yapmak kodlarımızı oldukça kısaltmış ve mikroişlemciye yüklenen yükü azaltmıştır. Ayrıca LM35'tan başka bir sıcaklık sensörünün daha nasıl kullanıldığını böylelikle görmüş olduk.

7.2) Hi-Tech'te I²C İletişim İşlemleri

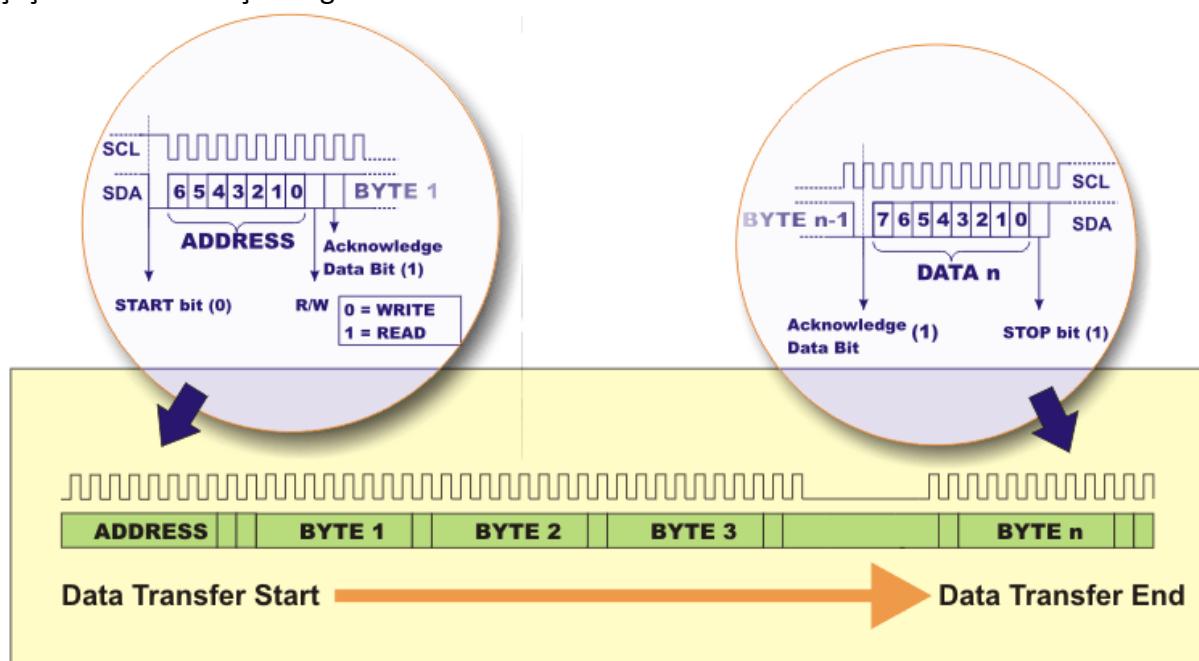
I²C (Inter-Integrated Circuit) seri iletişimimi yine SPI iletişim gibi mikrodenetleyicinin diğer elemanlarla iletişim kurmasını sağlayan ve Philips firması tarafından geliştirilen bir protokoldür. Özellikle hız gerektiren, pin sayısının önemli olduğu işlemlerde tercih edilir. Pic 16f877a içerisinde MSSP biriminde bulunan I²C biriminin diğer elemanlarla bağlantısı şekil-64'te görülebilir.



Şekil 64 – TC72 Uygulaması

I²C protokolünde bir pin seri giriş/çıkış (SDA) görevini üstlenirken diğer bir pin ise seri saat sinyali (SCL) üretmek için kullanılır. Bu iletişim metodunda cihazlardan biri master olur, master görevinde bulunan cihaz saat sinyalini kendi oluşturur. Diğer cihaz ise slave olur ve sadece bilgi giriş çıkış görevini yerine getirir.

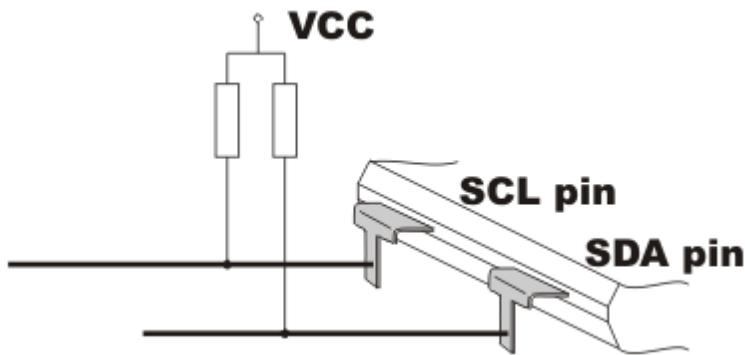
I²C protokolünde yavaş (100kbit/s), hızlı (400kbit/s) ve yüksek hızlı (3,4 Mbit/s) olmak üzere çeşitli hızlarda iletişim sağlanabilir.



Şekil 65 – I²C Veri İletim Protokolü

I^2C iletişiminde şekil-65'te görüleceği üzere ilk başta başlama biti, ardından adres bilgisi ve veri bitleri gönderilir. Veri gönderimi stop biti ile son bulur. Her bir byte gönderiminde ise karşı taraf bir alındı biti (ACK) geri gönderir. İletişim için gerekli durumları özetlersek;

- | | |
|-------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| Yol meşgul değil | : SDA ve SCL logic-1 olmalıdır. |
| Veri transferine başla | : SCL logic-1 iken, SDA logic-1'den logic-0 seviyesine getirilir. |
| Veri transferini durdur | : SCL logic-1 iken, SDA logic-0'dan logic-1 seviyesine getirilir. |
| Geçerli veri | : Her bilgi iletimi başlama biti ile başlar ve stop biti ile son bulur.
Her bilgi 1 byte şeklinde gönderilir, karşından gelen 9. bit ACK bitidir. |



Şekil 66 – I^2C Pinleri ve Pull-up dirençleri

Şekil-66'da de görüleceği üzere mikrodenetleyici ile I^2C iletişimini yapılacaksa SCL ve SDA uçlarına yaklaşık 10k pull-up direnci bağlanmalıdır.

I^2C protokolünü ayarlamak için Pic 16f877a'da SSPSTAT, SSPCON ve SSPCON2 kaydedicileri kullanılır. Bu kaydedicilerin görevleri aşağıda bit bit verilmiştir.

SSPSTAT	R/W (0)	R/W (0)	R (0)	R (0)	R (0)	R (0)	R (0)	Features	Bit name
	SMP	CKE	D/A	P	S	R/W	UA	BF	
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

- SMP** : 1: I^2C modunda düşük hızlarda (100KHz, 1MHz) slew rate kullanım dışıdır,
0: I^2C modunda yüksek (400KHz) hızlarda slew rate kullanımı aktiftir
- CKE** : 1: Transfer aktiften IDLE durumuna geçerken olacak
0: Transfer IDLE durumundan aktife geçerken olacak
- D/A** : I^2C modunda veri ve adres bilgilerini gösteren bittir.
(1: Data gönderilmiş ya da alınmış, 0: Adres gönderilmiş ya da alınmış)
- P** : I^2C modunda sonlandırma bittidir. (1: Durdurma biti son bulur, 0: Son bulmaz)
- S** : I^2C modunda başlangıç bittidir. (1: Başlangıç biti son bulur, 0: Son bulmaz)
- R/W** : SPI ya da I^2C modunda okuma mı yoksa yazma mı yaptığını belirten bittir.
(1: Okuma, 0: Yazma)
- UA** : 10 bitlik I^2C modunda adres güncelleme bittidir. UA=1 ise SSPADD kaydedicisine gönderilen adres güncellenir.
- BF** : Alım modunda (1: Alım tamamlandı (SSPBUF dolu), 0: Alım tamamlanmadı)

SSPCON	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Features
	WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

- WCOL** : Gönderim yapılrken yazılıdı mı (1: Yazıldı, hata var, 0: Yazılmadı)
Başlangıçta WCOL=0 yapılmalı.
- SSPOV** : Alım yapılrken yeni alım yapıldı mı (1: Yapıldı, hata var, 0: Yapılmadı)
Başlangıçta SSPOV=0 yapılmalı.
- SSPEN** : 1: SCL, SDA için ayarlandı, 0: Normal pin
- CKP** : IDLE seviyesi belirleme biti (1: IDLE yüksek seviye, 0: IDLE düşük seviye)
- SSPM3:SSPM0:** Mod seçim bitleri, aşağıda I²C için gerekli olan tanımlamalar gösterilmiştir.

SSPM3	SSPM2	SSPM1	SSPM0	Mode
0	1	1	0	I ² C slave mode, 7-bit address used
0	1	1	1	I ² C slave mode, 10-bit address used
1	0	0	0	I ² C master mode, clock = Fosc / [4(SSPAD+1)]
1	0	0	1	Mask used in I ² C slave mode
1	0	1	0	Not used
1	0	1	1	I ² C controlled master mode
1	1	0	0	Not used
1	1	0	1	Not used
1	1	1	0	I ² C slave mode, 7-bit address used,START and STOP bits enable interrupt
1	1	1	1	I ² C slave mode, 10-bit address used,START and STOP bits enable interrupt

SSPCON2	R/W (0)	R (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	Features
	GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN	Bit name
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	

- GCEN** : Slave modda genel çağrıma yetkilendirme bitidir. GCEN=1 olduğunda 0000h adresi çağrıldığında kesmeye gidilir.
- ACKSTAT** : I²C modunda iletim durumunda statü kabul bitidir.
(1: Alınan veri kabul edilmemiştir, 0: Alınan veri kabul edilmiştir)
- ACKDT** : I²C modunda master alım durumunda veri kabul bitidir.
(1: İletilen veri kabul edilmemiştir, 0: Veri kabul edilmişdir)
- ACKEN** : I²C modunda eş zamanlı art arda veri alımı yetkilendirme bitidir.
(1: Yetkilendirme açılır ve ACKDT veri biti ilettilir, bit donanım tarafından otomatik temizlenir)
- RCEN** : I²C modunda alım yetkilendirme bitidir.
(1: I²C modunda alım vaziyetine geçilir, 0: Alım modu devre dışı)
- PEN** : I²C modunda stop durum ayarının yapıldığı bittir.
(1: SDA ve SCL pinleri stop durumu alır, 0: Stop durumu oluşturulmaz)
- RSEN** : I²C modunda start durumu yenileme bitidir.
(1: SDA ve SCL pinlerinin start durumu tekrarlanır, 0: Yenileme oluşmaz)
- SEN** : I²C modunda start durum ayarının yapıldığı bittir.
(1: SDA ve SCL pinleri start durumu alır, 0: Start durumu oluşturulmaz)

Slew Rate: Data sinyalleri birer kare dalga gibi düşünülürse, yüksek hızlarda voltaj tepe noktası ve inen çıkan kenarların dikeylikleri bozulabilir. Slew rate bunu (0-5V, 400KHz) göz önüne alarak logic seviyeye karar verilmesini sağlar.

I²C birimi görüldüğü üzere SPI birimine kıyasla oldukça karmaşık bir yapıya sahiptir. Biz uygulamalarımızda kolaylık olması amacıyla pic'imizi master olarak kabul edeceğiz. Pic'in datasheet'inde yazanlara göre I²C birimini master ayarlamak için aşağıdaki adımları sırayla izlemeliyiz;

- **SSPCON kaydedicisinden I²C birimi master olarak ayarlanır.**

- **SSPEN=1** - SDA, SCL portları tanımlanıyor
- **CKP=1** - IDLE seviyesi 1 olacak
- **SSPM3:SSPM0=0b1000** - I²C master mode, clock=Fosc/[4(SSPAD+1)]

- **SSPCON2 kaydedicisinin tüm birimleri sonraki kontroller için sıfırlanır.**

- **SSPAD ile standart olan ve şekil -67'de görülen hızlardan biri seçilir (Örn: SSPAD=0x0A).**

- **SMP ayarlanarak slew rate kullanılıp kullanılmayacağı bildirilir.**

- **CKE ile yükselen veya düşen kenar seçimi yapılır.**

- **Kullanılacaksa kesme bayrakları temizlenir (PSPIF=0, BCLIF=0).**

F _{CY}	F _{CY} *2	BRG Value	F _{SCL} (2 Rollovers of BRG)
10 MHz	20 MHz	19h	400 kHz ⁽¹⁾
10 MHz	20 MHz	20h	312.5 kHz
10 MHz	20 MHz	3Fh	100 kHz
4 MHz	8 MHz	0Ah	400 kHz ⁽¹⁾
4 MHz	8 MHz	0Dh	308 kHz
4 MHz	8 MHz	28h	100 kHz
1 MHz	2 MHz	03h	333 kHz ⁽¹⁾
1 MHz	2 MHz	0Ah	100 kHz
1 MHz	2 MHz	00h	1 MHz ⁽¹⁾

Şekil 67 – I²C Hız Değerleri

Tüm bu söylediklerimizi bir kütüphane haline getirir ekstra fonksiyonları da yazarsak i2c.h kütüphane başlık dosyamız aşağıdaki gibi olacaktır.

```
/*
 *          www.FxDev.org
 *          I2C Kullanım Klavuzu
 * i2c_init();           ile i2c'nin ilk ayarlarını yap
 * i2c_wait();          ile IDLE için beklenir, yazım yapılmaz
 * i2c_start();         ile SCK ve SDA start durumu alır
 * i2c_restart();       ile SCK ve SDA start durumu tekrarlanır
 * i2c_stop();          ile SCK ve SDA stop durumu alır
 * veri=i2c_read(ack); ile okuma yapılır
 * ack=i2c_write(veri); ile yazma yapılır
 *
 * SSPADD hız ayarı i2c.c dosyası içerisinde yapılmalıdır.
 *          www.FxDev.org
 */

#define TRIS_SCK  TRISC3          // SCK ve SDA uçları belirlenir
#define TRIS_SDA  TRISC4

extern void i2c_init(void);
extern void i2c_wait(void);
```

```

extern void i2c_start(void);
extern void i2c_restart(void);
extern void i2c_stop(void);
extern unsigned char i2c_read(unsigned char ack);
extern unsigned char i2c_write(unsigned char i2c_data);

```

Yukarıdaki işlemleri yerine getiren **i2c.c** dosyamız ise aşağıdaki gibi olacaktır.

```

#include <pic.h>
#include "i2c.h"

void i2c_init(void)
{
    TRIS_SCK=1;           // SCK ve SDA giriş olarak ayarlanıyor
    TRIS_SDA=1;

    SSPCON      = 0x38;
    SSPCON2     = 0x00; // SSPCON2 sonraki ayarlamalar için temizleniyor
    /* 4Mhz'de
     * SSPADD=0x0A: 400KHz
     * SSPADD=0x0D: 308KHz
     * SSPADD=0x28: 100Khz
    */
    SSPADD      = 0x0A;

    CKE=0;        // Slew rate kullanılacak
    SMP=1;        // Kenar: Aktiften IDLE'a

    PSPIF=0;      // Kesme bayrakları temizleniyor
    BCLIF=0;
}

void i2c_wait(void)
{
    while (( SSPCON2 & 0x1F ) | RW );
    // IDLE için beklenir, yazım yapılmaz
}

void i2c_start(void)
{
    i2c_wait();        // IDLE beklenir
    SEN=1;            // SCK ve SDA start durumu alır
}

void i2c_restart(void)
{
    i2c_wait();        // IDLE beklenir
    RSEN=1;            // SCK ve SDA start durumu tekrarlanır
}

void i2c_stop(void)
{
    i2c_wait();        // IDLE beklenir
    PEN=1;            // SCK ve SDA stop durumu alır
}

unsigned char i2c_read(unsigned char ack)
{
    unsigned char i2c_data;

    i2c_wait();        // IDLE beklenir
    RCEN=1;            // Alım vaziyetine geçilir
}

```

```

i2c_wait();           // IDLE beklenir
i2c_data=SSPBUF;    // Veri alınır
i2c_wait();           // IDLE beklenir

if(ack)             // ACK biti durumu
{
    ACKDT=1;          // Alım kabul edilmedi
}
else
{
    ACKDT=0;          // Alım kabul edildi
}

ACKEN=1;              // ACKDT veri biti iletilir

return(i2c_data);
}

unsigned char i2c_write(unsigned char i2c_data)
{
    i2c_wait();         // IDLE beklenir
    SSPBUF=i2c_data;   // Veri gönderiliyor

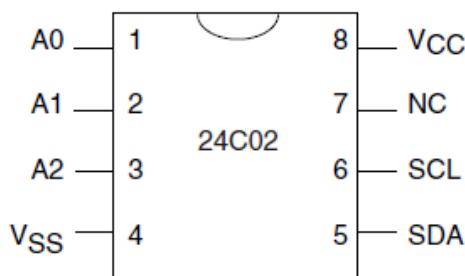
    return(ACKSTAT); // 1: Alınan veri kabul edilmemiştir
                       // 0: Alınan veri kabul edilmişdir
}
}

```

Kodlardan da görüleceği üzere anlatımı karmaşık da olsa yapılan işlemler oldukça basittir. Bu bölümden sonra I²C iletişimini kullanılmak istendiğinde, kütüphane de sadece hız ayarlaması yapmak yeterli olacaktır.

7.2.1) I²C İletişim Uygulaması

Bu uygulamamızda Atmel'in I²C iletişimini kullanan 24C02 harici eepromunu kullanacağız. Şekil-68'de DIP yapısı görülebilecek 24C02, 100Khz ve 400Khz hızlarında I²C iletişimini kullanabilecek özelliğe ve 256byte'lık hafızaya sahiptir.

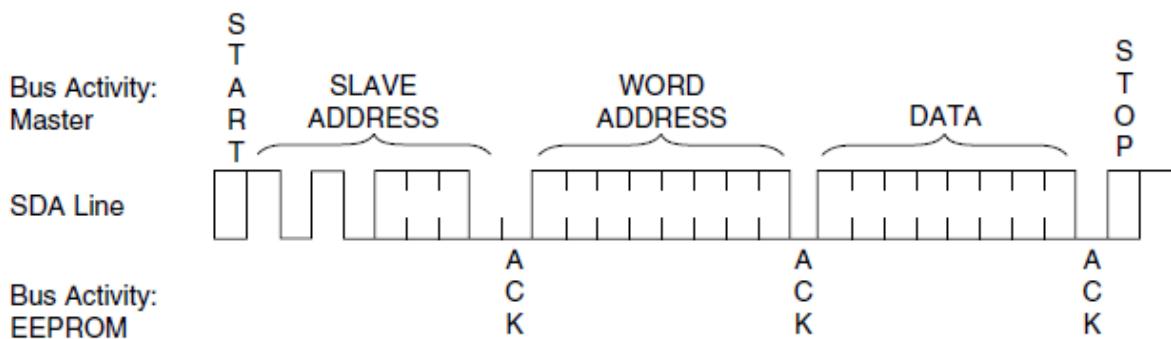


Şekil 68 – 24C02 Pin Yapısı

Şekil-68'de de görüleceği üzere SCL ve SDA pinleri pic'imizin belirli bacaklarına, Vcc +5V'ta, Vss, A0, A1 ve A2 ise toprak hattına bağlanmalıdır.

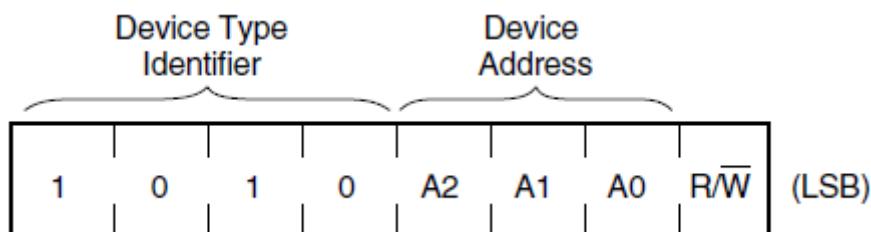
24C02'nin komut yapısı oldukça basittir böylelikle okuma ve yazma yapmak oldukça kolay bir hal alır.

Şekil-69'dan da görüleceği üzere 24C02'ye yazım yapmak için öncelikle I²C startı verilir, daha sonra slave address denilen komut gönderilir.



Şekil 69 – 24C02'ye Yazma Protokolü

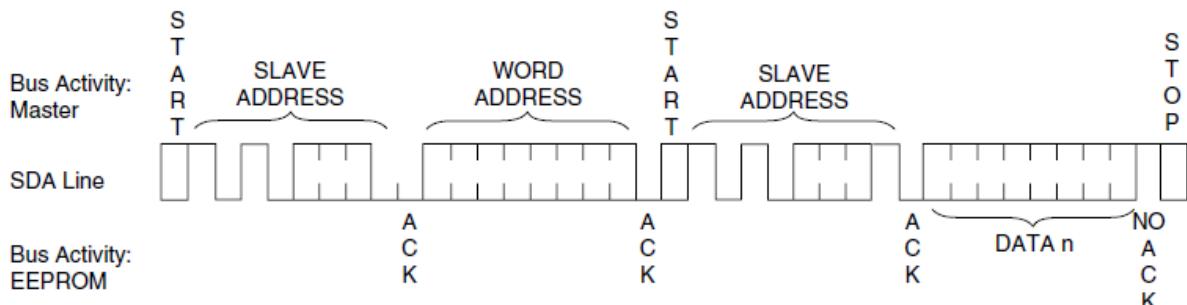
Slave address denilen komut şekil-70'de görülen yapı üzerinden kurulur.



Şekil 70 – 24C02'Slave Address Kontrol

Görüldüğü üzere yazım yapmak için 0xA0, okuma yapmak ise 0xA1 komutu kullanılır. Yazımı tamamlamak için ise öncelikle bilginin yazılacağı adres ve bilgi gönderilerek I²C stop komutu ile iletişim sonlandırılır ve yaklaşık 10-15 milisaniye beklenir.

Eepromdan bilgi okumak için ise şekil-71'de görülen iletişim metodu kullanılır.

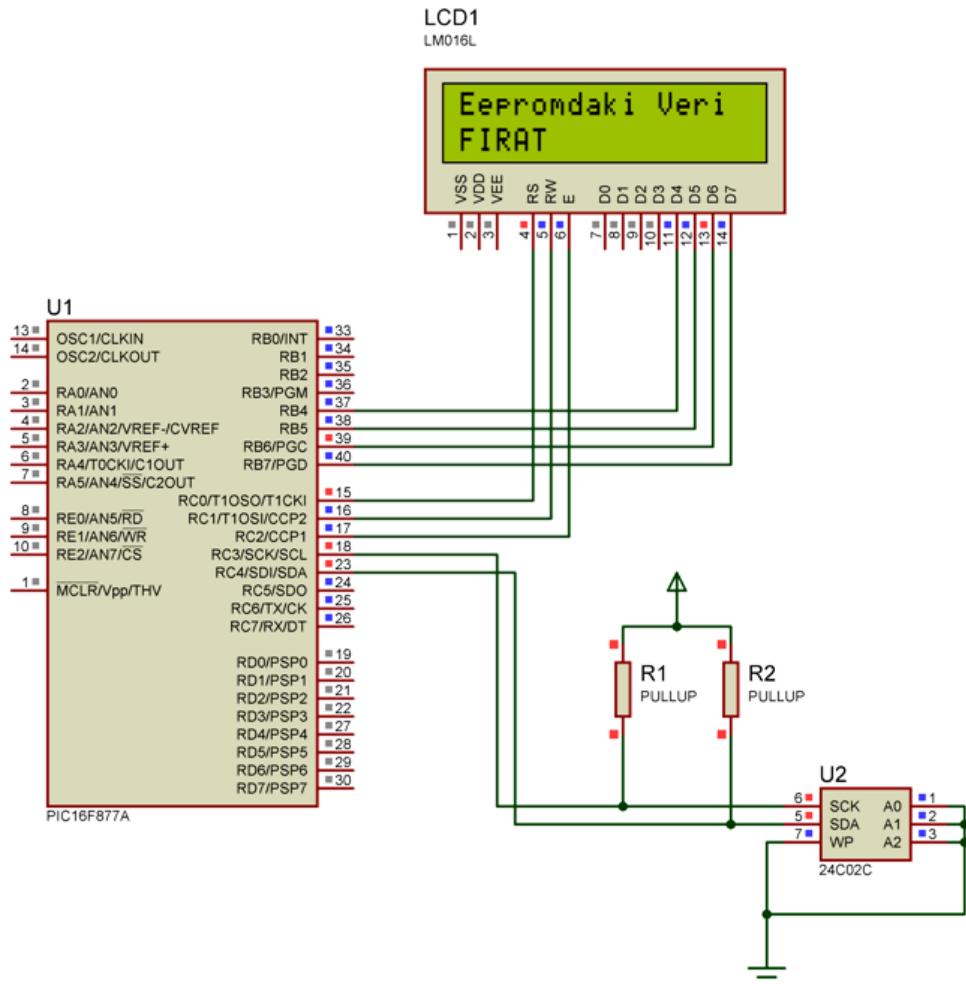


Şekil 71 – 24C02'den Okuma Protokolü

Öncelikle I²C startı verilir. Daha sonra slave address olarak 0xA0 gönderilir, daha sonrasında ise okunacak adres bilgisi gönderilerek I²C tekrar başlatılır. İkinci kez slave address olarak bu sefer okuma yapacağımızı belirten 0xA1 komutu gönderilerek, istenen adressteki bilgi alınır ve I²C stop komutu ile iletişim sonlandırılır.

Biz kütüphanemizde tüm bu işlemleri tanımladığımız için yapmamız gereken ufak bir fonksiyon yazarak istediğimiz işlemi gerçekleştirmekir.

Bu uygulamada 24C02 eepromunun ilk 0h,8h,10h,18h,20h,28h.. adreslerine ismimizi yazdırıp, tekrar eepromun bu adreslerini okuyarak ismimizi LCD'ye yazdıracağız. Uygulamaya geçmeden önce şekil-72'deki devreyi çiziyoruz.



Şekil 72 – 24C02 Uygulaması

Şekil-72'de de görüleceği üzere devremiz oldukça sadedir. Burada dikkat edilmesi gereken en önemli konu Pull-Up dirençleridir. İstediğimiz işlemi yerine getiren Hi-Tech kodu ise aşağıdaki gibidir.

```
#include <htc.h>
#include "delay.h" // Gecikme kütüphanesi tanımlanıyor
#include "lcd.h" // LCD kütüphanesi tanımlanıyor
#include "i2c.h" // I2C kütüphanesi tanımlanıyor

void write_ext_eeprom(unsigned char address, unsigned char data)
{
    i2c_start();
    i2c_write(0xA0);
    i2c_write(address);
    i2c_write(data);
    i2c_stop();
    DelayMs(15);
}

unsigned char read_ext_eeprom(unsigned char address)
{
```

```

unsigned char data;

i2c_start();
i2c_write(0xA0);
i2c_write(address);
i2c_restart();
i2c_write(0xA1);
data=i2c_read(0);
i2c_stop();
return(data);
}

void main(void)
{
    unsigned char i=0;

    PORTB=0x00;           // Portlar sıfırlanıyor
    PORTC=0x00;
    TRISB=0x00;           // Çıkışlar ayarlanıyor
    TRISC=0x00;

    lcd_init();           // LCD ve I2C ilk ayarları yapılıyor
    i2c_init();

    write_ext_eeprom(0x00, 'F'); // Ad eeproma yazdırılıyor
    write_ext_eeprom(0x08, 'I');
    write_ext_eeprom(0x10, 'R');
    write_ext_eeprom(0x18, 'A');
    write_ext_eeprom(0x20, 'T');

    // Eepromdaki bilgi LCD'ye yazdırılıyor
    lcd_yaz("Eepromdaki Veri");
    lcd_gotoxy(2,1);
    veri_yolla(read_ext_eeprom(0x00));
    veri_yolla(read_ext_eeprom(0x08));
    veri_yolla(read_ext_eeprom(0x10));
    veri_yolla(read_ext_eeprom(0x18));
    veri_yolla(read_ext_eeprom(0x20));
    for(;;);
}
}

```

Görüldüğü üzere devremiz gibi kodlarımız da oldukça sadedir. Yazma işlemimiz bittiğinde proteus simülasyonunu duraklatıp, eeproma sağ tıklatıp ‘Internal Memory’ kısmına tıklarsanız şekil-73’teki hafıza yapısını görmeniz mümkündür.

I2C Memory Internal Memory - U2			
00	46 FF FF FF	FF FF FF FF	F.....
08	49 FF FF FF	FF FF FF FF	I.....
10	52 FF FF FF	FF FF FF FF	R.....
18	41 FF FF FF	FF FF FF FF	A.....
20	54 FF FF FF	FF FF FF FF	T.....
28	FF FF FF FF	FF FF FF FF
30	FF FF FF FF	FF FF FF FF
38	FF FF FF FF	FF FF FF FF
40	FF FF FF FF	FF FF FF FF
48	FF FF FF FF	FF FF FF FF
50	FF FF FF FF	FF FF FF FF
58	FF FF FF FF	FF FF FF FF
60	FF FF FF FF	FF FF FF FF
68	FF FF FF FF	FF FF FF FF
70	FF FF FF FF	FF FF FF FF
78	FF FF FF FF	FF FF FF FF
80	FF FF FF FF	FF FF FF FF
88	FF FF FF FF	FF FF FF FF
90	FF FF FF FF	FF FF FF FF
98	FF FF FF FF	FF FF FF FF
A0	FF FF FF FF	FF FF FF FF
A8	FF FF FF FF	FF FF FF FF
B0	FF FF FF FF	FF FF FF FF
B8	FF FF FF FF	FF FF FF FF
C0	FF FF FF FF	FF FF FF FF
C8	FF FF FF FF	FF FF FF FF
D0	FF FF FF FF	FF FF FF FF
D8	FF FF FF FF	FF FF FF FF
E0	FF FF FF FF	FF FF FF FF
E8	FF FF FF FF	FF FF FF FF
F0	FF FF FF FF	FF FF FF FF
F8	FF FF FF FF	FF FF FF FF

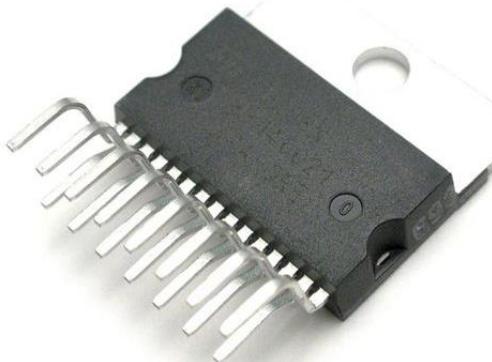
Sekil 73 – 24C02'nin Son Hali

Göründüğü üzere I²C iletişimini anlamak zor olsa da kullanmayı öğrendikten sonra iletişim kurmak, sadece donanımın özel komutlarını bilmekten daha zor olmamaktadır.

BÖLÜM 8 – MOTOR UYGULAMALARI

8.1) DC Motor

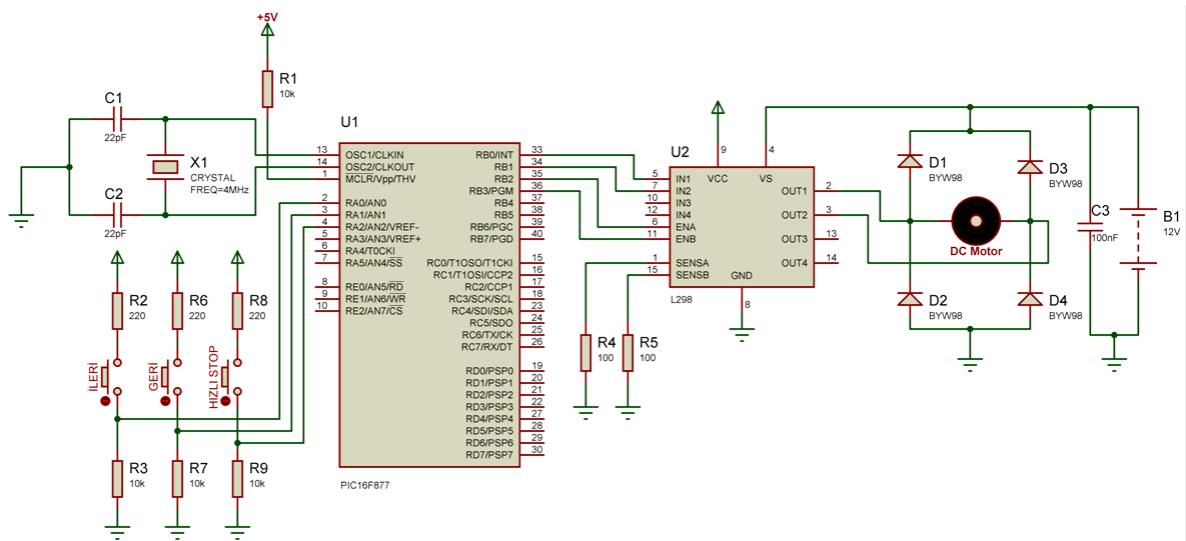
DC motorlar sabit bir mıknatıs ve içinde dönen yapı olan rotordan oluşurlar. Yüksek akım çektilkleri için direk mikrodenetleyiciye bağlanmazlar. Kontrol için H-Köprüsü denilen özel bir yapılarla mikrodenetleyicilere bağlanırlar. Günümüzde bu iş için geliştirilmiş, içerisinde H-Köprüsü bulunan L293, L298, L6201, L6202 gibi birçok entegre mevcuttur. Biz bu bölümde şekil-74'te görülen L298 entegresini temel alarak uygulamalarımızı yapacağız.



Şekil 74 – L298 Entegresi

L298 entegresine iki adet motor bağlanıp ayrı ayrı sağa, sola dönüşleri veya hızlıca durdurulmaları sağlanabilir.

Biz bu uygulamamızda bir adet DC motoru L298 entegresine bağlayıp, sağa, sola dönmesini ya da frenleme yapmasını sağlayacağız. Öncelikle şekil-75'teki devreyi proteusta çiziyoruz.



Şekil 75 – DC Motor Uygulaması

Şekil-75'ten de görüleceği üzere butonlarımız yükselen kenar için aktif olacaktır. Bunu dikkate alarak yazdığımız Hi-Tech kodu ise aşağıdaki gibi olacaktır.

```

#include <htc.h>

void main(void)
{
    ADCON1=0x07;           // PORTA dijital oluyor
    PORTA=0x00;            // Portlar sıfırlanıyor
    PORTB=0x00;
    TRISA=0x07;            // RA0, RA1, RA2 giriş
    TRISB=0x00;            // PORTB çıkış

    RB2=1;                  // A köprüsü seçiliyor
    RB3=0;

    for(;;)
    {
        if(RA0)             // İleri tuşuna mı basıldı
        {
            RB0=1;
            RB1=0;
        }
        if(RA1)             // Geri tuşuna mı basıldı
        {
            RB0=0;
            RB1=1;
        }
        if(RA2)             // Durdurulmak mı isteniyor
        {
            RB0=0;
            RB1=0;
        }
    }
}

```

Göründüğü gibi kodlarımız oldukça sadedir. Ayrıca bu motorlardan iki tane yan yana koyarak ve birkaç yer yön algılayıcı sensör ile çizgi izleyen robot yapımı oldukça basit hale gelebilmektedir.

8.2) Step Motor

Step motorlar genel itibarı ile fırçasız motorlardır. Diğer motorlardan farkı ise verilen çeşitli sinyallere göre istenilen şekilde yön alabilmesidir.

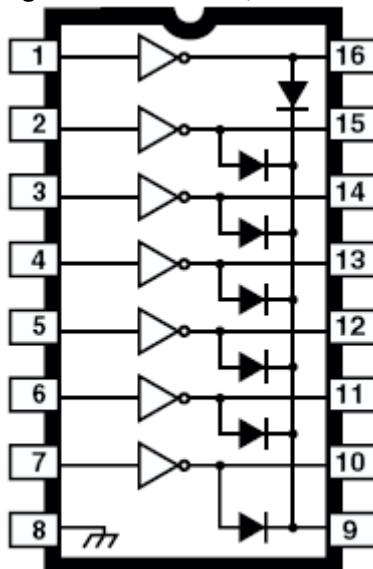
Şekil-76'da görülebilecek Step motorun iç yapısında birçok bobin ve genelde rotorda ise sabit bir mıknatıs bulunur. Çeşitli bobin çiftlerine verilen enerji ile bu mıknatıs istenilen bölgeye yönlenir.



Şekil 76 – Step Motorun İç Yapısı

Step motorlar genellikle 4-5-6 adet uça sahiptirler. Genel uygulamalarda 5 uçlu step motorlar tercih edilir. Bu uçlardan bir tanesi ortak uç iken diğerleri sinyalin verileceği uçlardır. Step motorlarda dikkat edilmesi gereken bir konu da adım cevabı denilen durumdur. Bu step motorun bir adımını ne kadar sürede aldığı belirten bir ifadedir. Eğer motora bu süreden daha kısa bir sürede sinyal gönderilirse step motor istenildiği gibi çalışmayaçaktır.

DC motorlar gibi step motorları da mikrodenetleyiciye direk bağlayarak süremeyiz. Bunun için geliştirilmiş ve şekil-77'de iç yapısı gözüken ULN2003, ULN2004 gibi entegreler kullanılır.



Şekil 77 – ULN2003 İç Yapısı

Step motorlarda bir diğer önemli konu da adım açısıdır. Örneğin 1,8 derece adım açısına sahip bir step motor 200 adım hareket sonra 360 derece dönüşünü tamamlayacaktır.

Step motorları sürmek için çeşitli metodlar geliştirilmiştir. Bunlardan en çok kullanılanları ise tam ve yarım açı yöntemleridir. Bu metodlar ile 5 uçlu step motorun nasıl sürüleceği aşağıdaki tablolarda gösterilmiştir.

Adım	Uç-1	Uç-2	Uç-3	Uç-4
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1

Tam Adım Metodu

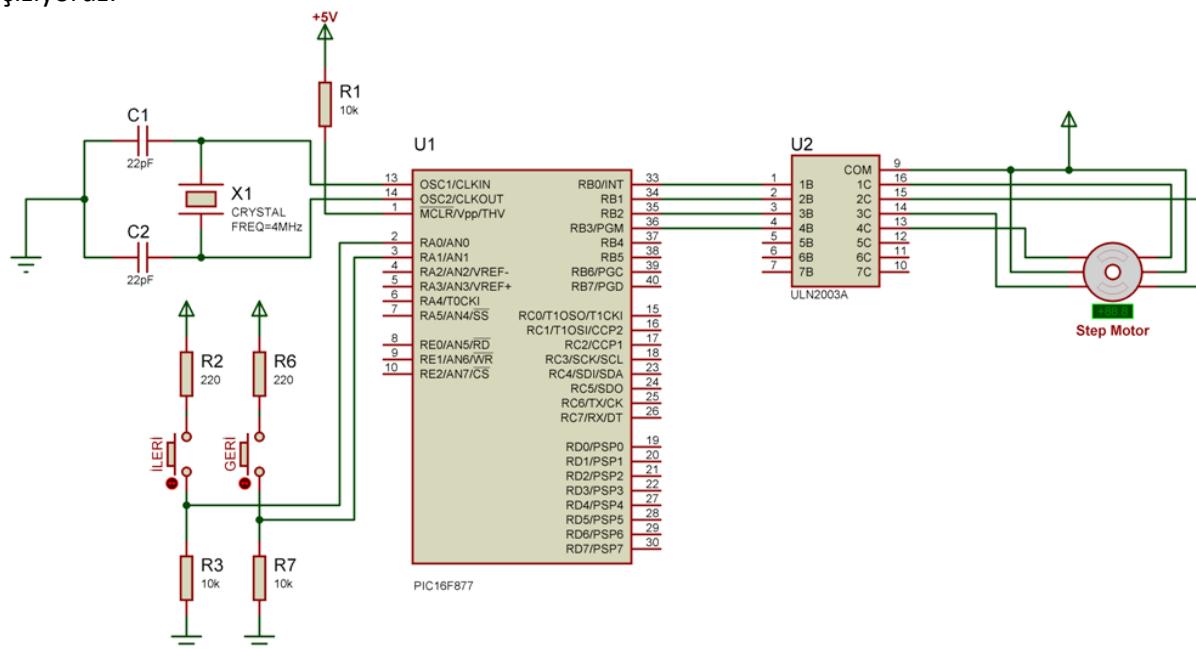
Adım	Uç-1	Uç-2	Uç-3	Uç-4
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1

Yarım Adım Metodu

Eğer step motorun her adımı 1,8 derece ise tam adım metoduyla her adım 1,8 dereceye, yarımlı adım metodu ile de 0,9 dereceye denk gelecektir. Buradan da görüldüğü üzere yarımlı adım metodu ile step motor sürüldüğünde adımlar daha hassaslaşmaktadır.

8.2.1) ULN2003 ile Step Motor Kontrolü

Bu uygulamamızda **ULN2003 entegresini ve yarımlı adım metodu kullanarak step motorun bir adım ileri ya da geri dönmelerini sağlayacağız**. Öncelikle şekil-78'deki devremizi proteusta çiziyoruz.



Şekil 78 – Step Motor Uygulaması

İstediğimiz işlemi yerine getiren Hi-Tech kodu ise aşağıdaki gibi olacaktır.

```
#include <htc.h>

// Yarımlı adım için dizi tanımlanıyor
const unsigned char
yarim_adim[]={0x01,0x03,0x02,0x06,0x04,0x0C,0x08,0x09};

void main(void)
{
    char i=0;
    ADCON1=0x07;           // PORTA dijital oluyor
    PORTA=0x00;             // Portlar sıfırlanıyor
    PORTB=0x00;
    TRISA=0x03;             // RA0, RA1 giriş
    TRISB=0x00;             // PORTB çıkış

    for(;;)
    {
        if(RA0)              // İleri tuşuna mı basıldı
        {
            PORTB=yarim_adim[i];
            while(RA0);
            if(i==7)
                i=-1;
        }
    }
}
```

```

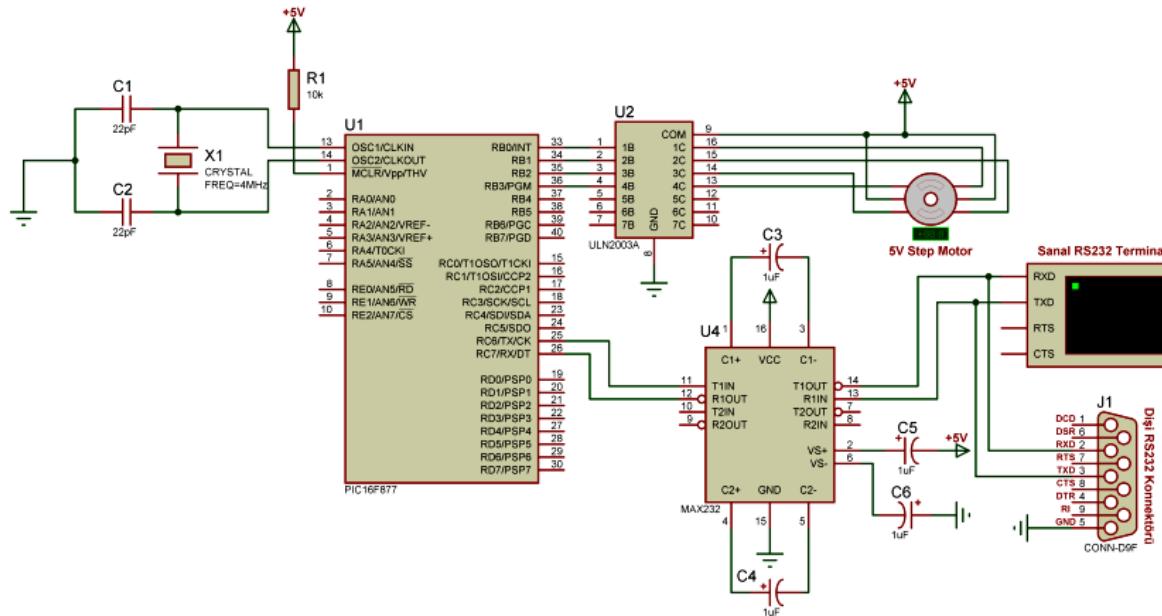
        i++;
    }
    if(RA1) // Geri tuşuna mi basıldı
    {
        if(i==0)
            i=8;
        i--;
        PORTB=yarim_adim[i];
        while(RA1);
    }
}
}

```

Kodlarda dikkat edilmesi gereken yer “yarim_adim[]” dizisidir. Yarım açı tablosunda görülen değerler buraya yazılarak işlemlerin daha kolay gerçekleşmesi sağlanmıştır.

8.2.2) Bilgisayar İle Step Motor Kontrolü

Bu uygulamamızda bilgisayardan gönderdiğimiz komutlarla step motorun sağa ya da sola sürekli dönmesini sağlayacağız. Öncelikle şekil-79'daki devreyi çiziyoruz.



Şekil 79 –Bilgisayar ile Step Motor Kontrolü

İstediğimizi yerine getiren Hi-Tech kodları ise aşağıdaki gibi olacaktır.

```

#include <htc.h>
#include <stdio.h> // printf için
#include "delay.h" // Gecikme kütüphanesi
#include "usart.h" // Seri haberleşme için

// Yarım adım için dizi tanımlanıyor
const unsigned char
yarim_adim[]={0x01,0x03,0x02,0x06,0x04,0x0C,0x08,0x09};

void main(void)
{
    char i,karakter=0;
    PORTB=0x00;
    TRISB=0x00; // PORTB çıkış
}

```

```

uart_init();

printf("ADIM MOTOR KONTROLU");
printf("\n\rwww.FxDev.org Sunar!");
printf("\r\n\r\n\rIslem Seciniz:");
printf("\r -Saga Dondur (R) ya da (r)");
printf("\r -Sola Dondur (L) ya da (l)");
printf("\r -Durdur (R),(L),(l) disinda bir tus");
printf("\r\n\r\n\rYapmak istediginiz islemi seciniz>");
for(;;)
{
    karakter=getch();
    putch(karakter);
    if(karakter=='R' || karakter=='r')
    // R ya da r tusuna mi basildi
    {
        printf("\rSaga Donuyor");
        printf("\rYapmak istediginiz islemi seciniz>");
        for(;;)
        {
            PORTB=yarim_adim[i];
            if(i==7)
                i=-1;
            i++;
            DelayMs(100);
            if(RCIF)
            {
                printf("\rMotor Durdu");
                printf("\rYapmak istediginiz islemi
seciniz>");

                break;
            }
        }
    }
    if(karakter=='L' || karakter=='l')
    // L ya da l tusuna mi basildi
    {
        printf("\rSola Donuyor");
        printf("\rYapmak istediginiz islemi seciniz>");
        for(;;)
        {
            if(i==0)
                i=8;
            i--;
            PORTB=yarim_adim[i];
            DelayMs(100);
            if(RCIF)
            {
                printf("\rMotor Durdu");
                printf("\rYapmak istediginiz islemi
seciniz>");

                break;
            }
        }
    }
}
}

```

Kodlarda da görüleceği üzere **RCIF** kesme bayrağından yararlanılmıştır. Bunun nedeni başka bilgi gönderildiğinde tamponun dolma kesme bayrağını set etmesidir. Bu kontrol edilerek sonsuz döngülerden **break** komutuyla çıkışlmıştır.

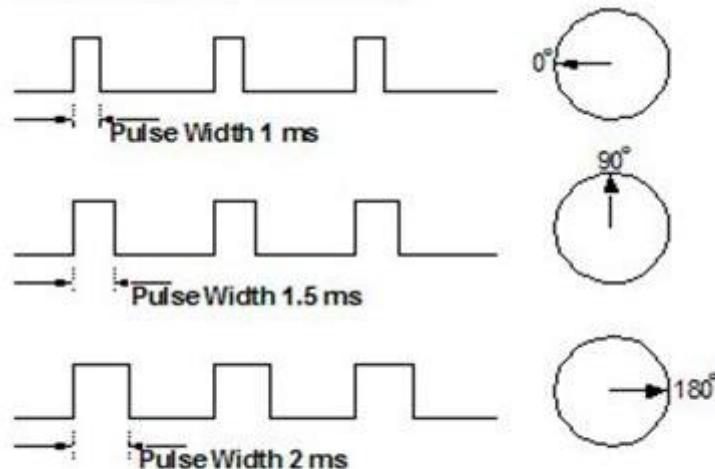
8.3) R/C Servo Motor

Servo motorlar genel itibarı ile içerlerinde bir DC motor ve şaft konum bilgisi devresi barındırırlar. Bu devre şaftın kaç derece döndüğünün algılanmasında kullanılır. Servo motorlarının iç yapısı şekil-80'de gözükmektedir.



Şekil 80 – Servo Motor İç Yapısı

Servo motorlar 20ms periyotlu, 1ms'den 2ms'e kadar değişen duty cycle'lı PWM sinyali ile sürürlürler. Verilen duty cycle periyotlarına göre servo motorun 0-180 derece aralığı arasında alacakları değerler şekil-81'de verilmiştir.

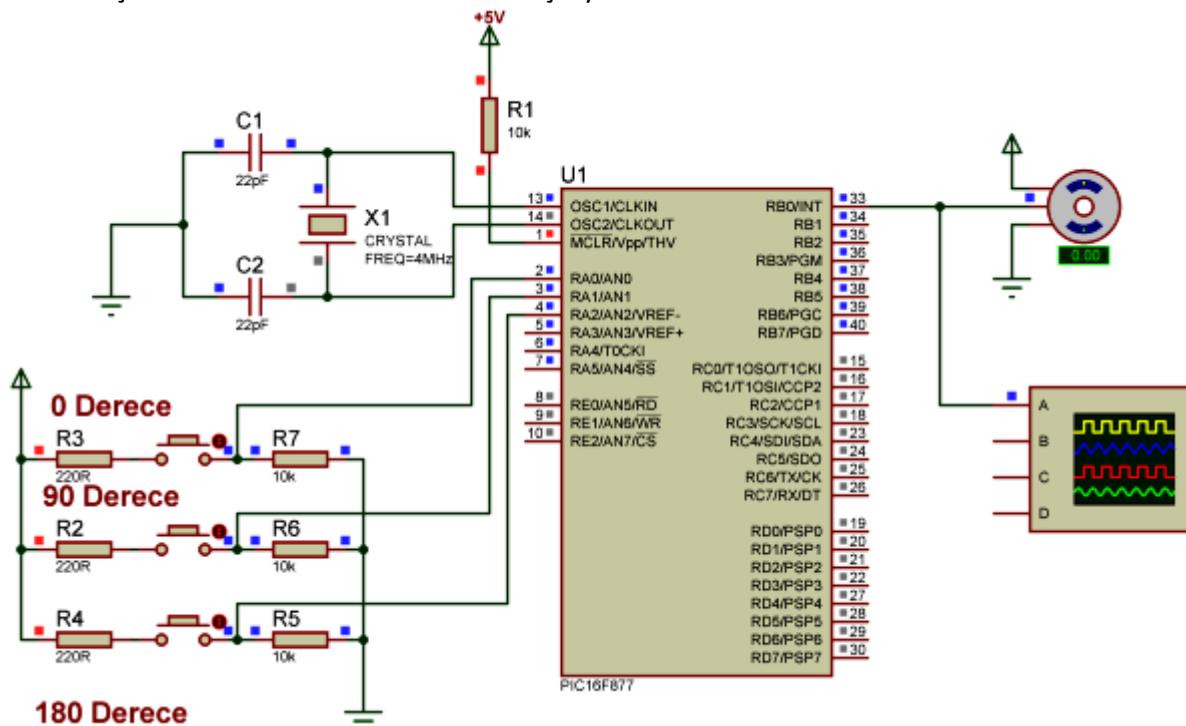


Şekil 81 – Servo Motor Kontrol Sinyalleri

Burada sinyal verilirken örneğin 1,5ms ile 1ms arası 90'a bölünerek istenilen açıya gidilebilir. Pic'in CCP modülü genellikle kullandığımız 4Mhz kristal için 50Hz'lik PWM sinyalleri üretmemektedir. Bu yüzden yukarıdaki sinyalizasyonu elle ayarlamamız gerekmektedir.

8.3.1) R/C Servo Motor Uygulaması

Bu bölümde kullanacağımız servo moturu 0 derece, 90 derece ve 180 dereceye, basacağımız butonlarla yönlendireceğiz. Sinyalizasyon işlemini hazır delay.h fonksiyonları ile sağlayacağız. Öncelikle şekil-82'deki devremizi Proteus'ta çiziyoruz.



Şekil 82 – Servo Motor Uygulaması

İstediğimiz işlemi yerine getiren Hi-Tech kodu ise aşağıdaki gibi olacaktır.

```
#include <htc.h>
#include "delay.h"           // Gecikme kütüphanesi

void main(void)
{
    char i,karakter=0;
    ADCON1=0x07;             // PORTA dijital oluyor
    PORTB=0x00;
    TRISA=0x07;              // RA0, RA1 ve RA3 giriş
    TRISB=0x00;               // PORTB çıkış

    for(;;)
    {
        if(RA0)
        {
            for(;;)
            {
                RB0=1;
                // 1ms Gecikme
                DelayUs(250);DelayUs(250);DelayUs(250);DelayUs(250);
                RB0=0;
                // 19ms Gecikme
                DelayMs(19);
                if(RA1 | RA2)
                    break;
            }
        }
    }
}
```

```

        }
        if(RA1)
        {
            for(;;)
            {
                RB0=1;
                // 1,5ms Gecikme
                DelayUs(250);DelayUs(250);DelayUs(250);DelayUs(250);
                DelayUs(250);DelayUs(250);
                RB0=0;
                // 18,5ms Gecikme
                DelayMs(18);
                DelayUs(250);DelayUs(250);
                if(RA0 | RA2)
                    break;
            }
        }
        if(RA2)
        {
            for(;;)
            {
                RB0=1;
                // 2ms Gecikme
                DelayMs(2);
                RB0=0;
                // 18ms gecikme
                DelayMs(18);
                if(RA0 | RA1)
                    break;
            }
        }
    }
}

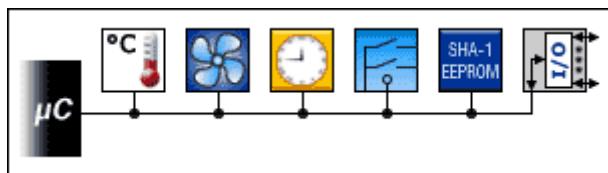
```

Simülasyonu çalıştırduğumızda 0,1 veya 0,2 derecelik hatalar gözükecektir. Bunlar gecikme fonksiyonlarının bire bir çalışmamasının sonuçlarıdır. Fakat normal uygulamalarda fark edilmeyecek kadar küçük hata paylarıdır.

BÖLÜM 9 – 1-WIRE (TEK HAT) İLETİŞİMİ ve DS18B20

9.1) 1-Wire (Tek Hat) İletişim

1-Wire ya da diğer adıyla tek hat iletişim Maxim-Dallas firması tarafından geliştirilmiş çift yönlü veri iletişimine izin veren bir protokoldür. Master olan denetleyici şekil-83'te de görüleceği üzere birden fazla cihazı aynı hattan kontrol edebilir.



Şekil 83 – Tek Hat Protokolü İle Birden Fazla Cihaz Kontrolü

1-Wire iletişime sahip tüm cihazlar kendilerine ait 64 bitlik kayıt koduna sahiptirler. Bu kodun ilk düşük değerlikli 7 biti cihazın aile kodunu, sonraki 48 bit seri numarasını ve son kalan 8 bit ise hata denetim kodunu içerir. Böylelikle tek hatta bulunan her ayrı birim kendi kayıt kodu ile çağrılarak işleme tabi tutulurlar.

1-Wire iletişimde bir diğer husus ise 1-Wire iletişimde kullanılacak hattın mutlaka pull-up direnciyle Vdd hattına bağlanması zorunluluğudur.

9.2) 1-Wire (Tek Hat) İletişim İşlemleri

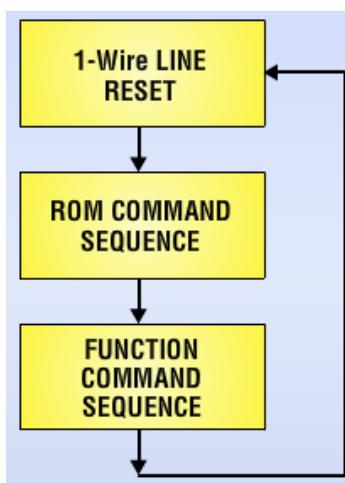
1-Wire iletişimde sıfır veya bir bilgi gönderme, cihaza reset atma gibi işlemler bir kereye mahsus tanımlanır ve tüm cihazlar aynı mantık çerçevesinde bu işlemleri kullanır.

Ayrıca 1-Wire iletişim işlemleri standart ve yüksek hızlarda olabilmektedir. 1-Wire iletişim standart hızı 15kbps iken, yüksek hız ise 115kbps'dır.

1-Wire protokolünde zamanlamalar oldukça önemli olduğu için biz işlemlerimizi standart hızda yapacağız.

1-Wire protokolü şekil-84'te de görüleceği gibi şu adımlarla işler;

- **1-Wire Hat Reseti verilir**
- **ROM komutları gönderilir**
- **Fonksiyon komutları gönderilir**

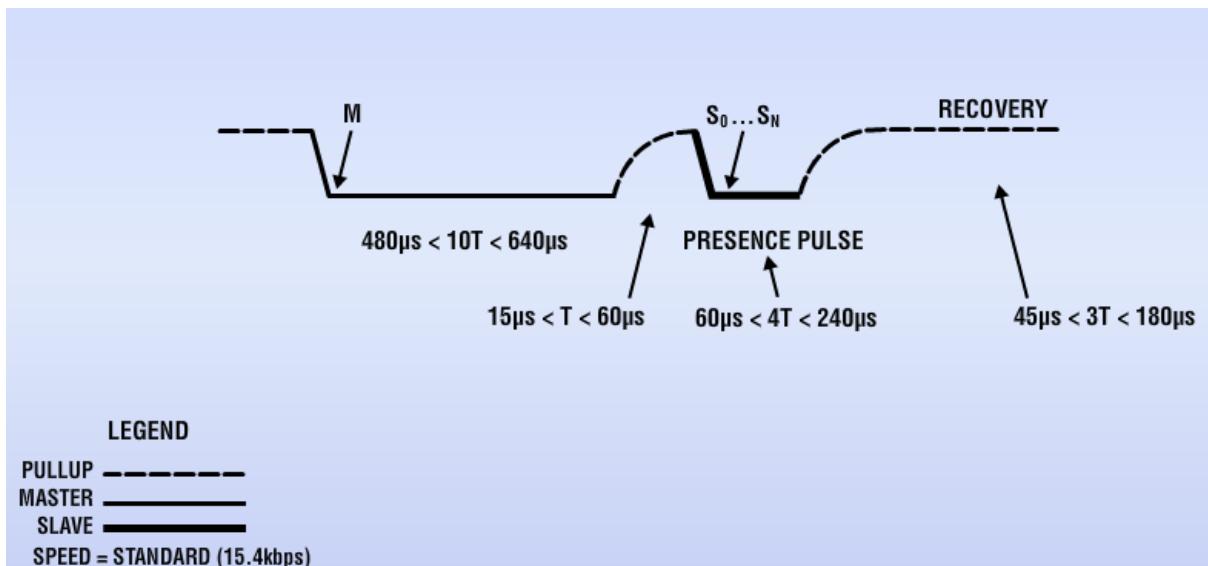


Şekil 84 – 1-Wire Adımları

9.2.1) Reset İşlemi

Şekil-85'te zaman diyagramı görülecek reset işlemi şu adımlarla gerçekleşir;

- Hat mikroişlemci tarafından logic-0 yapılır,
- 480 ile 640 mikrosaniye beklenir,
- Mikroişlemci pini giriş olarak ayarlanır ve hat pull-up direnci sayesinde logic-1 olur,
- Yaklaşık 70 mikro saniye sonra hattaki cihaz, hattı logic-0 yapar
- Böylelikle hatta cihaz olduğu algılanır.
- Hattın tekrar pullup tarafından logic-1 olması ise yaklaşık 420 mikrosaniye gerektirir.



Şekil 85 – 1-Wire Reset İşlemi Zaman Diyagramı

9.2.2) Yazım veya Okuma İşlemleri

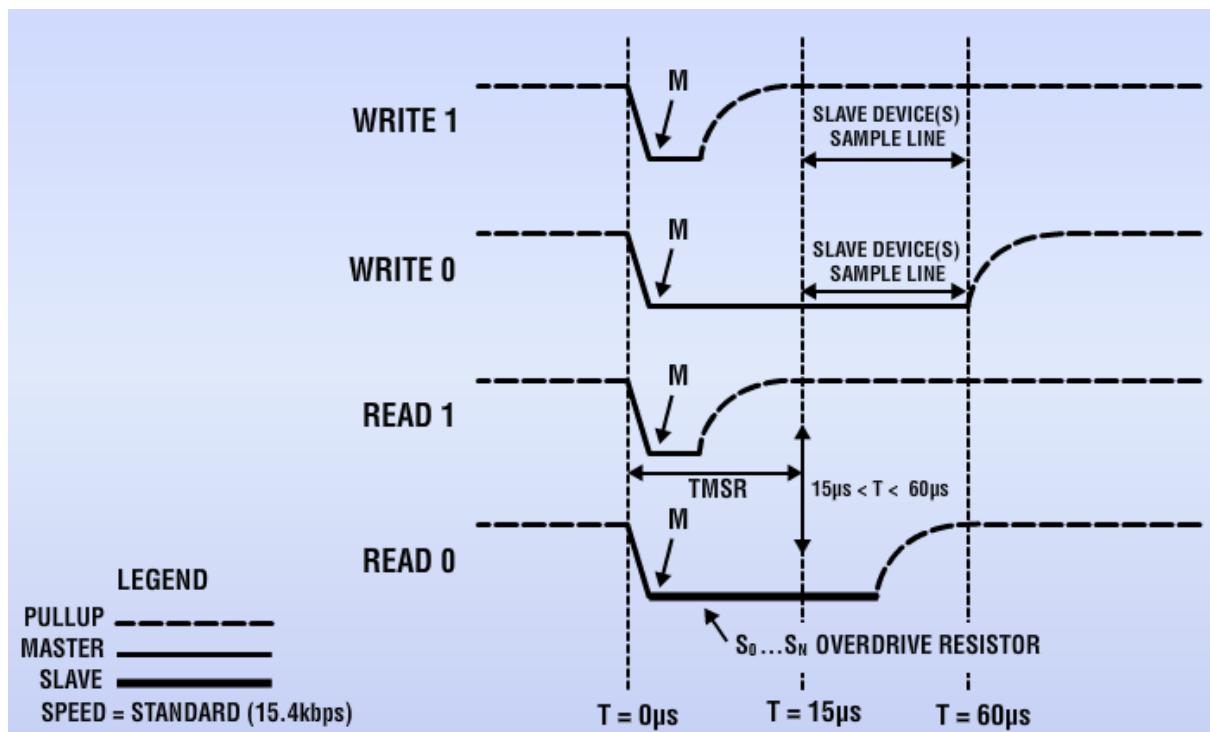
1-Wire protokolün en can alıcı noktası hatta bilgi gönderip alma kısımdır. Bunun için zamanlamalar hatta 0 veya 1 gönderme ya da hattan 0 veya 1 bilgisi alma kısımlarında özelleşmektedir.

Şekil-86'dan görüleceği gibi hatta 0 veya 1 bilgisi göndermek için aşağıdaki adımlar uygulanmalıdır;

- Hat mikrodenetleyici tarafından logic-0 konumuna getirilir,
- Hatta 0 yazılmak isteniyorsa 1-2 mikrosaniye sonra,
- Hatta 1 yazılmak isteniyorsa yaklaşık 60 mikrosaniye sonra hat mikrodenetleyici tarafından okuma konuma getirilerek pull-up direncinin hattı logic-1'e getirmesi beklenir.

Hattan 0 veya 1 bilgisi okumak için ise aşağıdaki adımlar uygulanır;

- Hat mikrodenetleyici tarafından logic-0 yapılır ve 5 mikrosaniye beklenir,
- Mikrodenetleyici hattı okuma moduna getirir,
- Eğer hat yaklaşık 10 mikrosaniye sonra pullup tarafından logic-1 olursa cihazın gönderdiği bilgi 1, eğer hat bu süre sonunda hala logic-0 ise cihazın gönderdiği bilgi logic-0'dır,
- Yaklaşık 60 mikrosaniye sonra bu iletişim son bulur.



Şekil 86 – 1-Wire Okuma Yazma Zaman Diyagramları

Tüm bu söylediklerimizi bir kütüphane haline getirirsek ileride herhangi bir 1-Wire iletişim protokolünü kullanan aygitla iletişim sağlamak istediğimizde kolaylıkla hazırladığımız dosyayı projeye ekleyip istediğimizi gerçekleştirebilir. Yukarıdaki bilgiler doğrultusunda yazdığımız ve ek birkaç özellik daha eklediğimiz **one_wire.h** dosyası aşağıdaki gibi olacaktır.

```
/*
 *          www.FxDev.org
 *          One Wire Kullanım Klavuzu
 *  onewire_reset(); ile one wire aracına reset atıp,
 *  geri dönüş değeriyle aletin hazır olur olmadığını kontrol et
 *  onewire_read_byte(DATA); şeklinde veri yazdır veya komut gönder
 *  onewire_read_byte(); şeklinde okuma yap
 *          www.FxDev.org
 */
#define TRIS_W    TRISA0      //Bağlantılar
#define WIRE      RA0

extern unsigned char onewire_reset(void);
extern unsigned char onewire_read_bit(void);
extern void onewire_write_bit(unsigned char veri);
extern unsigned int onewire_read_byte(void);
extern void onewire_write_byte(unsigned char data);
```

Bu fonksiyonları çalıştırın **one_wire.c** dosyamız ise aşağıdaki gibi olacaktır.

```
#include <pic.h>
#include "one_wire.h"
#include "delay.h"

unsigned char onewire_reset(void)
{
    unsigned char sonuc;
```

```

WIRE=0;                                // Hat sıfıra çekiliyor
TRIS_W=0;                                // Hat çıkış yapılıyor
DelayUs(250);DelayUs(250); // 640us>x>480us arasında bekleniyor
TRIS_W=1;                                // Pull-up yükselmesi bekleniyor
DelayUs(70);                                // 70us bekleniyor
sonuc=WIRE;                                // 0 Hazır, 1 Hazır değil
DelayUs(250);DelayUs(250); // x>420us'den büyük olmalı

    return sonuc;
}

unsigned char onewire_read_bit(void)
{
    unsigned char veri;

    WIRE=0;                                // Hat sıfıra çekiliyor
    TRIS_W=0;                                // 5 mikrosaniye bekleniyor
    DelayUs(5);                                // Pull-Up'ın hattı yükselmesi bekleniyor
    TRIS_W=1;                                // 10 mikrosaniye bekleniyor
    DelayUs(10);

    if(WIRE)                                // Hat okunuyor
        veri=1;                                // Okunan veri 1
    else
        veri=0;                                // Okunan veri 0
    DelayUs(60);

    return veri;
}

void onewire_write_bit(unsigned char veri)
{
    if(veri)                                // 1 gönderiliyor
    {
        TRIS_W=0;                                // Hat sıfırlanıyor
        WIRE=0;                                // 5 mikrosaniye bekleniyor
        DelayUs(5);                                // Pull-Up'ın hattı yükselmesi bekleniyor
        TRIS_W=1;                                // 60 mikrosaniye bekleniyor
        DelayUs(60);
    }
    else                                    // 0 gönderiliyor
    {
        TRIS_W=0;                                // Hat sıfırlanıyor
        WIRE= 0;                                // 60 mikrosaniye bekleniyor
        DelayUs(60);                                // Pull-Up'ın hattı yükselmesi bekleniyor
        TRIS_W=1;
    }
}

unsigned int onewire_read_byte(void)
{
    unsigned char i,veri=0, bitler=1;
    for(i=0;i<8;i++)
    {
        if ( onewire_read_bit() == 1 )
            veri |= bitler;
        bitler=bitler<<=1;
    }
    return veri;
}

```

```

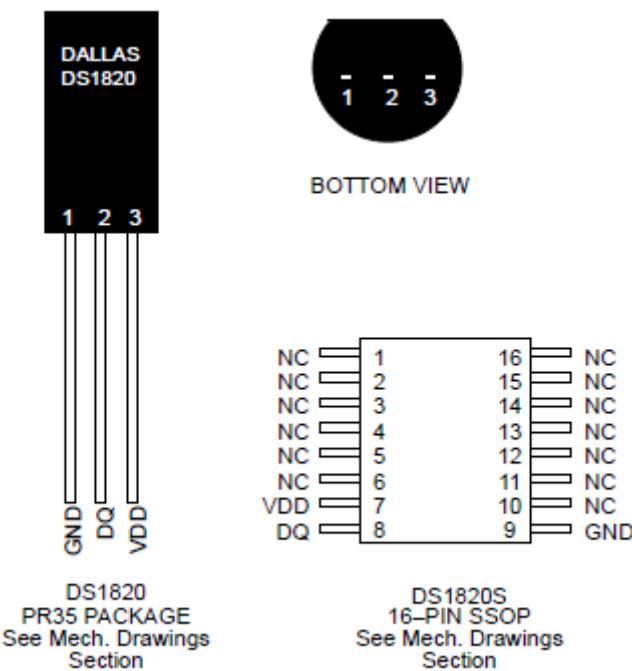
void onewire_write_byte(unsigned char veri)
{
    unsigned char bitler=1, i;
    for (i=0;i<8;i++)
    {
        if (veri&bitler)
        {
            onewire_write_bit(1);
        }
        else
        {
            onewire_write_bit(0);
        }
        bitler=bitler<<=1;
    }
}

```

Kodlarda da görüleceği üzere zamanlar bire bir alınmalıdır. 1-Wire protokolünde dikkat edilmesi gereken en büyük husus budur.

9.3) DS18B20 ile 1-Wire (Tek Hat) Uygulaması

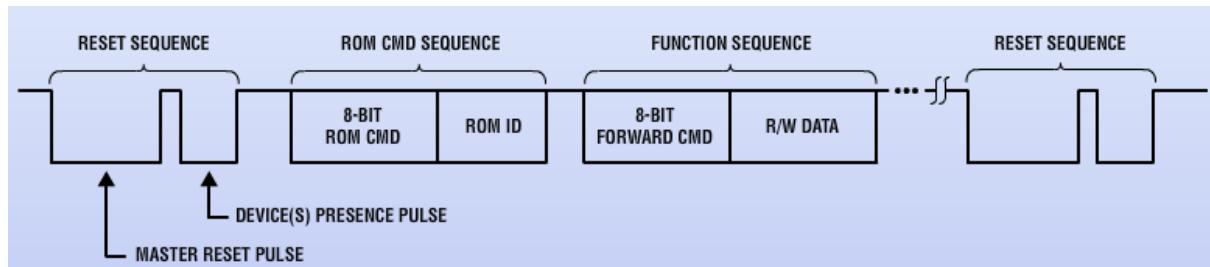
DS18B20 Maxim'in kullanıcılarına sunduğu, 1-Wire protokolü kullanan, kılıf yapısı şekil-87'de görülebilecek, 12bit çözünürlüğe sahip sıcaklık sensöründür. Tek hattan iletişim kurması, çok az enerji tüketmesi, çözünürlüğünün yüksek olması bir çok projede sıcaklık sensörü olarak kullanılmasının başlıca nedenidir. -55C ile +125C sıcaklık ölçüm aralığı ile günlük uygulamalarda oldukça yeterlidir.



Şekil 87 – 1-Wire Okuma Yazma Zaman Diyagramları

DS18B20 sensörünü çalıştırırmak için 1-Wire protokolünün yanında, sensörün anlayacağı kodlara da ihtiyacımız vardır. DS18B20 datasheetinde tüm sensör detayları anlatılsa da biz kısaca sıcaklık okuma işlemini nasıl gerçekleştirdiğimize değineceğiz.

Tüm 1-Wire cihazlarında olduğu gibi DS18B20'de de okuma veya yazım işlemleri şekil-88'deki sırayla gerçekleşir. Öncelikle reset sinyali cihaza gönderilir, daha sonrasında ROM ve fonksiyon komutları ya da bilgileri cihaza gönderilir ya da alınır.



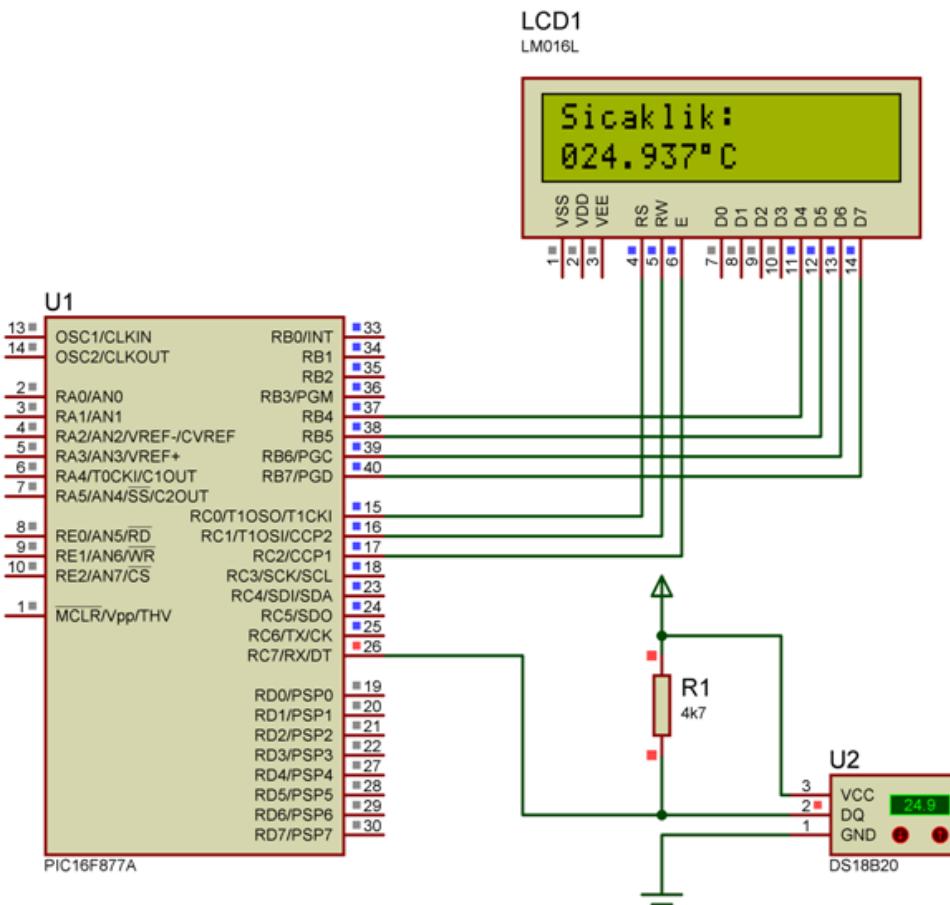
Şekil 88 – 1-Wire Cihaz Bilgi Gönderme Alma Rutinleri

Datasheet'in sıcaklık okuma konusunda bizi yönlendirmesi şekil-89'daki gibidir.

MASTER MODE	DATA (LSB FIRST)	COMMENTS
TX	Reset	Reset pulse (480–960 µs).
RX	Presence	Presence pulse.
TX	55h	Issue "Match ROM" command.
TX	44h	Issue "Convert T" command.
TX	<I/O LINE HIGH>	I/O line is held high for at least 500 ms by bus master to allow conversion to complete.
TX	Reset	Reset pulse.
RX	Presence	Presence pulse.
TX	55h	Issue "Match ROM" command.
TX	<64-bit ROM code>	Issue address for DS1820.
TX	BEh	Issue "Read Scratchpad" command.
RX	<9 data bytes>	Read entire scratchpad plus CRC; the master now recalculates the CRC of the eight data bytes received from the scratchpad, compares the CRC calculated and the CRC read. If they match, the master continues; if not, this read operation is repeated.
TX	Reset	Reset Pulse.
RX	Presence	Presence pulse, done.

Şekil 89 – Sıcaklık Okuma İçin Gerekli İşlemler

Şekil-89'daki yönergeleri dikkate alarak **DS18B20'den sıcaklık okuma işlemimizi kolayca tamamlayabiliriz.** Öncelikle şekil-90'daki devremizi çizelim.



Şekil 90 – DS18B20 Uygulaması

Uygulama anında dikkat edilmesi gereken bir diğer husus ise 4k7'lik dirençtir. Tüm 1-Wire cihazları gibi DS18B20'de pull-up dirençsiz çalışmamaktadır.

Uygulamamızı çalıştırın Hi-Tech kodu ise aşağıdaki gibi olacaktır.

```
#include <htc.h>
#include "delay.h"           // Gecikme kütüphanesi tanımlanıyor
#include "lcd.h"             // LCD kütüphanesi tanımlanıyor
#include "one_wire.h"         // 1-Wire kütüphanesi tanımlanıyor

void ds18b20(unsigned int *sicak, unsigned int *onda)
{
    unsigned char sayil,sayi2,busy=0;

    while(onewire_reset());
    onewire_write_byte(0xCC);
    onewire_write_byte(0x44);
    DelayMs(250);
    DelayMs(250);
    DelayMs(250);
    while (onewire_reset());
    onewire_write_byte(0xCC);
    onewire_write_byte(0xBE);

    sayi2 = onewire_read_byte();
    sayil = onewire_read_byte();

    while (onewire_reset());
```

```

*onda=0;
*sicak=(sayi1*16)+(sayi2>>4);
if(sayi2 & 0x08)
    *onda=500;
if(sayi2 & 0x04)
    *onda+=250;
if(sayi2 & 0x02)
    *onda+=125;
if(sayi2 & 0x01)
    *onda+=62;
}

void main(void)
{
    unsigned int sicaklik,onda;
    PORTB=0x00;           // PORTB ve PORTC sıfırlanıyor
    PORTC=0x00;
    TRISB=0x00;          // PORTB çıkış
    TRIISC=0x00;          // PORTC çıkış

    lcd_init();           // LCD ilk ayarları yapılıyor
    lcd_yaz("Sicaklik:");

    for(;;)
    {
        lcd_gotoxy(2,1);
        ds18b20(&sicaklik , &onda);
        veri_yolla(sicaklik/100+48);
        veri_yolla((sicaklik%100)/10+48);
        veri_yolla((sicaklik%10+48));
        veri_yolla('.');
        veri_yolla(onda/100+48);
        veri_yolla((onda%100)/10+48);
        veri_yolla((onda%10+48));
        veri_yolla(0xDF);
        veri_yolla('C');
    }
}

```

Kodlarımız görüldüğü gibi gittikçe karmaşıklasmaktadır. Bu örneğimiz C'nin en önemli özelliklerinden biri olan **pointer** yapısının da nasıl kullanılacağına dair örnek teşkil etmektedir. LCD ekranımızda derece işaretini çikaran 0xDF kodudur.

BÖLÜM 10 – RTC ve DIJITAL POT UYGULAMALARI

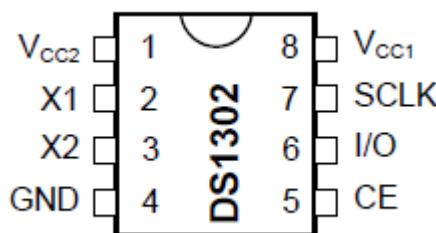
10.1) RTC İşlemleri

Pic ile tarih, saat işlemlerini yapmanın birçok yolu vardır. Bunlardan birisi kendi zaman gecikmelerimizi (`DelayMs()`, `DelayUs()` gibi) kullanmak ya da timer kesmelerinden faydalananmaktadır. Fakat her iki durumda da sıcaklık farklılıklar, parazitler, programsal sorunlar gibi birçok etmenden dolayı günde 1-2 saniyeliğe varan hatalar oluşturabilmektedir. Bunun önüne geçmek için ise bu iş için özel olarak geliştirilmiş RTC yani Real Time Clock entegreleri kullanılması gerekmektedir.

Biz bu bölümde Dallas Maxim'in ürettiği DS1302 entegresinin kütüphanesini yazarak, tarih saat uygulaması yapacağız.

10.1.1) DS1302 Entegresi

Şekil-91'de DIP soket yapısı görülebilecek DS1302 entegresi 2100 yılına kadar tarih tutabilen, 5V ile 2V aralığında çalışabilen, 2V'ta 300nA gibi çok az akım çeken, 3-Wire iletişim metodunu kullanan, ayrıca içerisinde 31 byte'lık eeprom barındıran bir yapıdır.

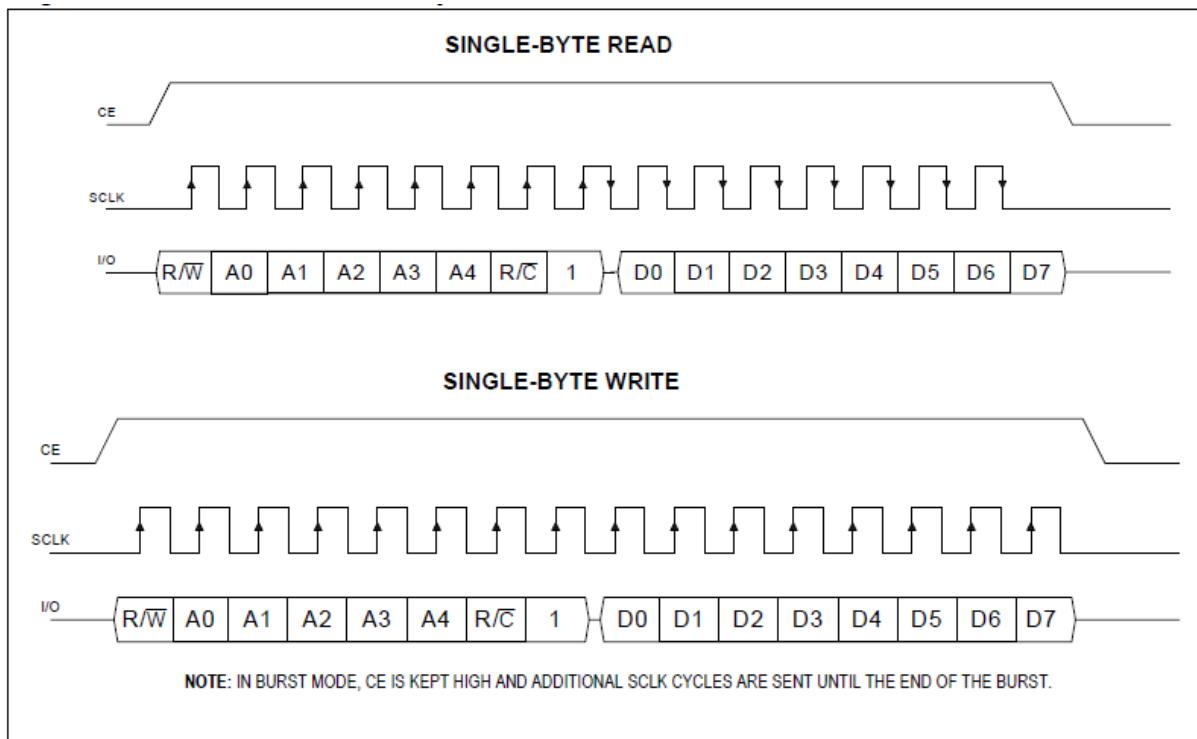


Şekil 91 – DS1302 Entegresi

Şekil-90'da görülen yapıda Vcc2 bacağına +5V, Vcc1 bacağına güç kesilse bile entegrenin çalışmasına devam etmesini sağlayacak şarj edilebilir bir pil, X1 ve X2 bacaklarına entegreye özel 32.768KHz kristal bağlanır. CE ucu entegreyi aktif etme ucudur. I/O ucu ise hem okuma hem de yazma yapmak için özelleştirilmiştir. SCLK ucu ise veri iletim ya da alımında senkronizasyonu sağlayan bacaktır. Tüm alım ya da gönderim işlemleri SCLK'nın yükselen kenarında gerçekleşmektedir.

DS1302'ye yazım ya da DS1302'den okuma yapmak için şekil-92'deki zaman diyagramı kullanılır. Şekil-92'den de görüleceği üzere DS1302'den okuma yapmak için öncelikle CE=1 olmalı. Daha sonra ise şekil-93'te görülen ve hangi adresi okumak istediğimize dair 1 byte'lık bilgi gönderilir. 1 veya 2 mikrosaniye sonra ise mikrodenetleyici okuma moduna alınarak okunmak istenen 1 byte'lık bilgi alınır.

DS1302'ye yazım işlemi de aynen okuma işlemi gibidir; öncelikle CE=1 yapılır, daha sonra yazılmak istenen adres 1 byte şeklinde gönderilir ve 1-2 mikrosaniye sonra yazılmak istenen 1 byte'lık veri gönderilir.



Şekil 92 – DS1302 Zaman Diyagramları

DS1302 kullanılarakken yapılması gereken ilk ayarlar şöyledir, tüm kodlar şekil-93'te görülebilmektedir;

- Write Protect yani yazım koruması kaldırılmalıdır, (0x8E'ye 0x00 yazılmalı),
- Vcc2 ile Vcc1 arasında diyon veya direnç şarj edilecek pile göre ayarlanır (Biz örnek olarak 0x90'a 0xA6 yazalım),
- Daha sonra saatin çalıştırılması sağlanmalıdır. (0x81 okunarak 0 olup olmadığı kontrol edilir, 0 değilse 0x80'e 0 yazılarak saat çalıştırılır)

RTC

READ	WRITE	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	RANGE
81h	80h	CH	10 Seconds			Seconds			00–59	
83h	82h		10 Minutes			Minutes			00–59	
85h	84h	12/24	0	10 AM/PM	Hour	Hour			1–12/0–23	
87h	86h	0	0	10 Date		Date			1–31	
89h	88h	0	0	0	10 Month	Month			1–12	
8Bh	8Ah	0	0	0	0	0	Day		1–7	
8Dh	8Ch	10 Year			Year			00–99		
8Fh	8Eh	WP	0	0	0	0	0	0	0	—
91h	90h	TCS	TCS	TCS	TCS	DS	DS	RS	RS	—

RAM

C1h	C0h								00–FFh
C3h	C2h								00–FFh
C5h	C4h								00–FFh
.	.								.
.	.								.
FDh	FCh								00–FFh

Şekil 93 – DS1302 Komutları

Şekil-93'te dikkat edilmesi gereken en önemli konu belirli bölgeye yazım yapılacağında veya okunacağında birer fazla ya da eksik adres byte'larının kullanılmasıdır. Örneğin saniye ayarlaması yapmak için, istediğimiz değeri 0x80 adresine yazmakta iken, saniye değerini okumak için ise 0x81 adresini kullanmak zorundayız. Aynı şey ram için de söz konusudur, mesela ilk adrese 0xC0 ile yazım yapılabilirken, 0xC1 ile okuma yapılmaktadır.

Tüm bunlar dikkate alınarak hazırladığımız kütüphanemizin **ds1302.h** dosyası aşağıdaki gibi olacaktır.

```
/*
 * www.FxDev.org
 * DS1302 Kullanım Klavuzu
 * ds1302_init(); ile RTC1302'nin ilk ayarlarını yap
 * rtc_ram_write(0xC0, 13); şeklinde 0x0C ile 0xFC arasındaki hafızaya
veri yaz
 * rtc_ram_read(0xC1); şeklinde 0x0C ile 0xFC arasındaki hafızayı oku
 * rtc_set_datetime(GUN,AY,YIL,HAFTA,SAAT,DAKIKA,SANIYE); şeklinde zamanı
ayarla
 * rtc_get_date(&GUN,&AY,&YIL,&HAFTA); şeklinde tarihi al
 * rtc_get_time(&SAAT,&DAKIKA,&SANIYE); şeklinde saati al
 *
www.FxDev.org
*/

#define TRIS_IO TRISD1 //IO portu neye bağlıysa
//TRISxy'deki xy ona göre belirlenmeli
#define SCLK RD0 //Bağlantılar
#define IO RD1
#define RST RD2

extern void send_one(void);
extern void send_zero(void);
extern void ds1302_write(unsigned char ram, unsigned char veri);
extern unsigned char ds1302_read(unsigned char ram);
extern void ds1302_init(void);
extern unsigned char get_bcd(unsigned char veri);
extern unsigned char rakam(unsigned char veri);
extern void rtc_ram_write(unsigned char address, unsigned char data);
extern unsigned char rtc_ram_read(unsigned char address);
extern void rtc_set_datetime(unsigned char day,unsigned char mth,unsigned
char year,unsigned char dow,unsigned char hr,unsigned char min, unsigned
char sec);
extern void rtc_get_date(unsigned char *day, unsigned char *mth, unsigned
char *year, unsigned char *dow);
extern void rtc_get_time(unsigned char *hr, unsigned char *min, unsigned
char *sec);
```

Yukarıda da görüleceği üzere DS1302'yi kontrol edebilmek için oldukça fazla fonksiyon kullanılmıştır. Hangi fonksiyonun ne işe yaradığı ise **ds1302.h** dosyasının kullanım kılavuzu kısmında açıklanmıştır.

Fonksiyonlarda görüleceği üzere ***hr** gibi ifadeler pointer özellikli değişkenlerdir. Yerleri pic'in herhangi bir bölgesindeki ve bu yer asla değişmez. Okuma yapmak için ise **&** ifadesiyle örneğin **&saat** şeklinde başka bir değişkene, bu belirtilen bölgedeki bilgiyi atamak mümkündür.

Tüm bu işlemleri yapan **ds1302.c** dosyası ise aşağıdaki gibi olacaktır.

```
#include <pic.h>
#include "ds1302.h"
#include "delay.h"

void send_one(void)
{
    TRIS_IO=0;
    IO=1;
    DelayUs(2);
    SCLK=0;
    DelayUs(2);
    SCLK=1;
    DelayUs(2);
}

void send_zero(void)
{
    TRIS_IO=0;
    IO=0;
    DelayUs(2);
    SCLK=0;
    DelayUs(2);
    SCLK=1;
    DelayUs(2);
}

void ds1302_write(unsigned char ram, unsigned char veri)
{
    unsigned char i;
    RST=1;
    DelayUs(2);
    for(i=0;i<8;i++)
    {
        if((ram>>i)&0x01)
            send_one();
        else
            send_zero();
    }

    for(i=0;i<8;i++)
    {
        if((veri>>i)&0x01)
            send_one();
        else
            send_zero();
    }
    RST=0;
    DelayUs(2);
    SCLK=0;
    DelayUs(2);
}

unsigned char ds1302_read(unsigned char ram)
{
    unsigned char i,veri=0;
    RST=1;
    DelayUs(2);
    for(i=0;i<8;i++)
    {
```

```

        if((ram>>i)&0x01)
            send_one();
        else
            send_zero();
    }

TRIS_IO=1;
DelayUs(2);

for(i=0;i<8;i++)
{
    SCLK=0;
    if(IO==1)
        veri=veri|(1<<i);
    else
        veri=veri|(0<<i);
    SCLK=1;
}
RST=0;
DelayUs(2);
TRIS_IO=0;
DelayUs(2);
SCLK=0;
DelayUs(2);

return veri;
}

void ds1302_init(void)
{
    unsigned char x;

RST=0;
DelayUs(2);
SCLK=0;
DelayUs(2);

ds1302_write(0x8E,0);
ds1302_write(0x90,0xA6);

x=ds1302_read(0x81);
if((x & 0x80)!=0)
    ds1302_write(0x80,0);
}

unsigned char get_bcd(unsigned char veri)
{
    unsigned char nibh;
    unsigned char nibl;

    nibh=veri/10;
    nibl=veri-(nibh*10);

    return((nibh<<4)|nibl);
}

unsigned char rakam(unsigned char veri)
{
    unsigned char i;
    i=veri;
    veri=((i>>4)&0x0F)*10;
}

```

```

        veri=veri+(i&0x0F);
        return veri;
    }

void rtc_ram_write(unsigned char address, unsigned char data)
{
    ds1302_write(address, data);
}

unsigned char rtc_ram_read(unsigned char address)
{
    return rakam(ds1302_read(address));
}

void rtc_set_datetime(unsigned char day,unsigned char mth,unsigned char year,unsigned char dow,unsigned char hr,unsigned char min, unsigned char sec)
{
    ds1302_write(0x86,get_bcd(day));
    ds1302_write(0x88,get_bcd(mth));
    ds1302_write(0x8c,get_bcd(year));
    ds1302_write(0x8a,get_bcd(dow));
    ds1302_write(0x84,get_bcd(hr));
    ds1302_write(0x82,get_bcd(min));
    ds1302_write(0x80,get_bcd(sec));
}

void rtc_get_date(unsigned char *day, unsigned char *mth, unsigned char *year, unsigned char *dow)
{
    *day = rakam(ds1302_read(0x87));
    *mth = rakam(ds1302_read(0x89));
    *year = rakam(ds1302_read(0x8d));
    *dow = rakam(ds1302_read(0x8b));
}

void rtc_get_time(unsigned char *hr, unsigned char *min, unsigned char *sec)
{
    *hr = rakam(ds1302_read(0x85));
    *min = rakam(ds1302_read(0x83));
    *sec = rakam(ds1302_read(0x81));
}

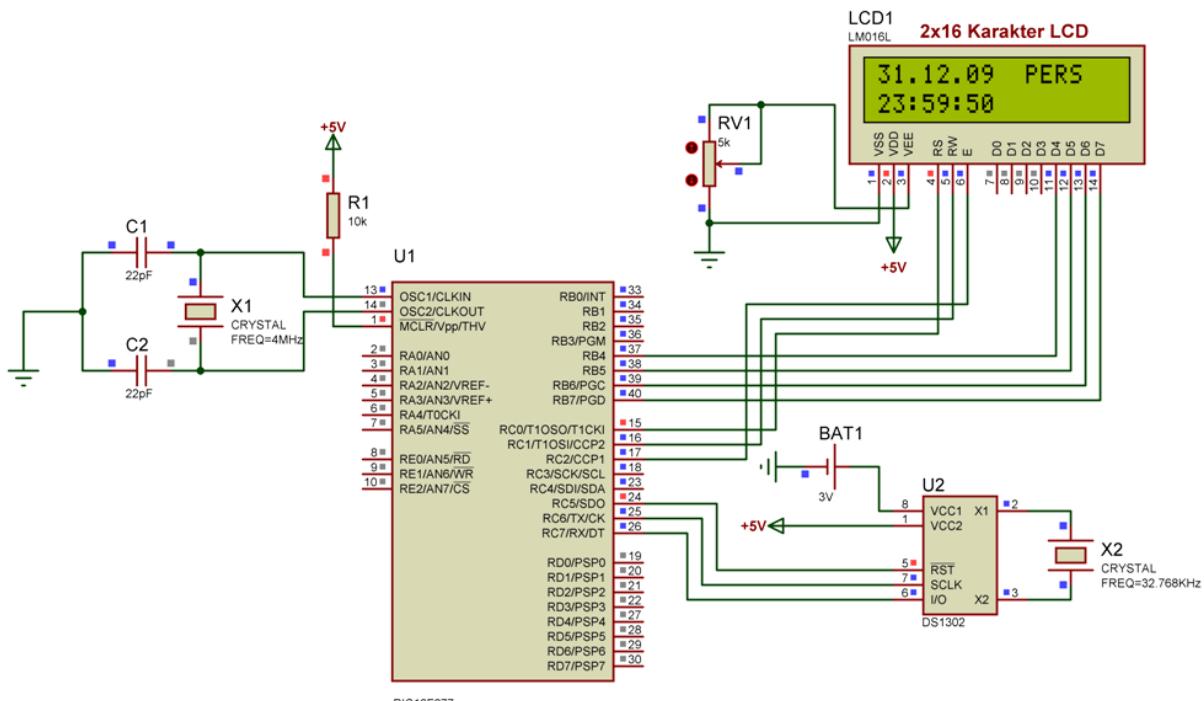
```

Yukarıda da görüleceği üzere kütüphane dosyası oldukça uzundur. Aslında işlemlere bakıldığından o kadar da karmaşık işlemlerin yapılmadığı görülecektir. Bu işlemleri kısaltmak için daha önce gördüğümüz SPI haberleşmesi kullanılabilir.

10.1.2) DS1302 Uygulaması

Bu uygulamamızda DS1302 kullanarak LCD ile tarih saat uygulaması yapacağız. Tarihimizi 31.12.2009 saat 23:59:50 Perşembe'ye ayarlayıp tarihin tam değişme anını göreceğiz. Ayrıca DS1302'nin eepromuna da adımızı kaydedeceğiz.

Uygulamamızın kodlarını yazmadan önce şekil-94'teki devreyi çiziyoruz.



Şekil 94 – DS18B20 Uygulaması

İstediğimiz işlemi yerine getirenl kodumuz ise aşağıdaki gibi olacaktır.

```
#include <htc.h>
#include "delay.h"           // Gecikme kütüphanesi tanımlanıyor
#include "lcd.h"              // LCD kütüphanesi tanımlanıyor
#include "ds1302.h"            // ds1302 kütüphanesi tanımlanıyor

void main(void)
{
    unsigned char gun,ay,yil,hafta,saat,dakika,saniye;
    PORTB=0x00;                // PORTB ve PORTC sıfırlanıyor
    PORTC=0x00;
    TRISB=0x00;                // PORTB çıkış
    TRISC=0x00;                // PORTC çıkış

    lcd_init();                 // LCD ilk ayarları yapılıyor
    ds1302_init();              // ds1302 ilk ayarları yapılıyor

    //31.12.2009 Perşembe, 23:59:50'a ayarlanıyor
    rtc_set_datetime(31,12,9,5,23,59,50);

    //İsmimizi EEPROM'a yazıyoruz
    rtc_ram_write(0xC0, 'F');
    rtc_ram_write(0xC2, 'I');
    rtc_ram_write(0xC4, 'R');
    rtc_ram_write(0xC6, 'A');
    rtc_ram_write(0xC8, 'T');
}
```

```

for(;;)
{
    // Tarih ve saat ds1302'den alınıyor
    rtc_get_date(&gun,&ay,&yil,&hafta);
    rtc_get_time(&saat,&dakika,&saniye);
    lcd_gotoxy(1,1);           // Tarih ve saat LCD'e yazdırılıyor
    veri_yolla(gun/10+48);
    veri_yolla(gun%10+48);
    veri_yolla('.');
    veri_yolla(ay/10+48);
    veri_yolla(ay%10+48);
    veri_yolla('.');
    veri_yolla(yil/10+48);
    veri_yolla(yil%10+48);
    switch(hafta)
    {
        case 1: lcd_yaz(" PAZR");break;
        case 2: lcd_yaz(" PZTS");break;
        case 3: lcd_yaz(" SALI");break;
        case 4: lcd_yaz(" CRSM");break;
        case 5: lcd_yaz(" PERS");break;
        case 6: lcd_yaz(" CUMA");break;
        case 7: lcd_yaz(" CMTS");break;
    }
    lcd_gotoxy(2,1);
    veri_yolla(saat/10+48);
    veri_yolla(saat%10+48);
    veri_yolla(':');
    veri_yolla(dakika/10+48);
    veri_yolla(dakika%10+48);
    veri_yolla(':');
    veri_yolla(saniye/10+48);
    veri_yolla(saniye%10+48);
}
}

```

Burada dikkat edilmesi gereken nokta switch/case yapısıdır. DS1302 haftanın başlangıç gününü pazar günü olarak kabul etmektedir. Dolayısı ile ayarlamalarımızı yaparken buna dikkat edilmesi gerekmektedir.

Ayrıca simülasyonu duraklatıp DS1302'nin ram'ine bakarsanız isminizin şekil-95'tekine benzer şekilde yazıldığını görebilirsiniz.

DS1302 RAM - U2									
00	46	49	52	41	54	FF	FF	FF	FIRAT...
08	FF	FF	FF	FF	FF	FF	FF	FF
10	FF	FF	FF	FF	FF	FF	FF	FF
18	FF	FF	FF	FF	FF	FF	FF	FF

Şekil 95 – DS1302 Eeprom'u

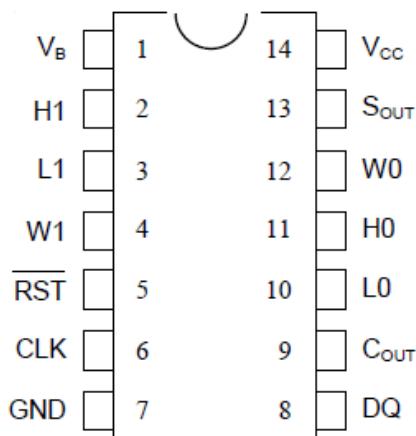
Bu uygulamaya bir önceki bölümde gördüğümüz DS18B20'yi de ekleyerek sıcaklık/tarih uygulaması yapabilirsiniz.

10.2) DS1868 ile Dijital Pot İşlemi

Dijital potansiyometreler aynı analog potansiyometlerin yaptığı işi yani belirli aralıklar arasında direnç değişikliği işlemini yapmaktadır.

Bu iş için özelleştirilmiş bir çok entegre mevcuttur. Biz Dallas Maxim firmasının ürettiği DS1868 dijital potansiyometresi ile işlemimizi gerçekleştireceğiz.

Şekil-96'da DIP yapısı görülebilecek DS1868'in içinde 256 değişik pozisyonda ayarlanabilen iki adet potansiyometre mevcuttur. DS1868'in potları sahip olduğu modellere göre 10k, 50k veya 150k'luk değerlere sahiptirler. DS1868 de DS1302 gibi 3-Wire iletişim protokolü ile kontrol edilmektedir.

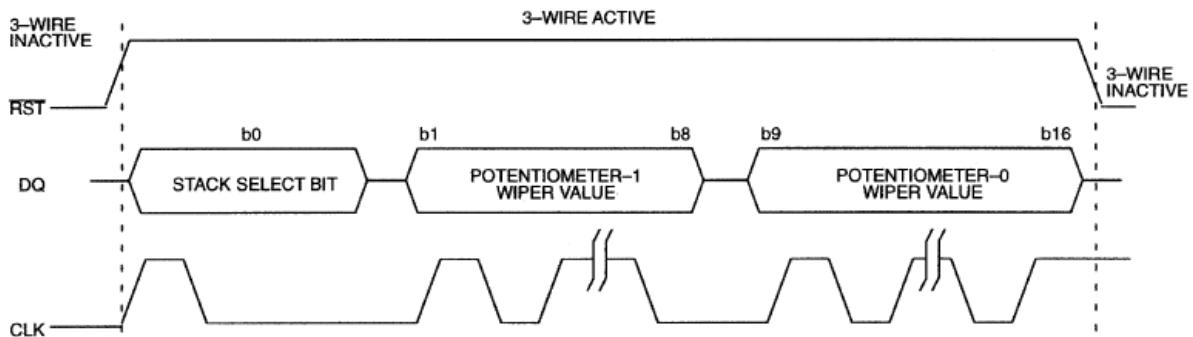


Şekil 96 – DS18B20 Uygulaması

Şekil-96'da görülen pinlerin görevleri ise şöyledir;

- Vb** : Alt tabaka ön gerilimi
- H1** : 1. Potansiyometrenin üst ucu
- L1** : 1. Potansiyometrenin alt ucu
- W1** : 1. Potansiyometrenin orta ucu
- RST** : Seri port reset girişi
- CLK** : Saat girişi
- GND** : Toprak ucu
- DQ** : Seri port data girişi
- Cout** : Kaskat port çıkışı
- LO** : 0. Potansiyometrenin alt ucu
- H0** : 0. Potansiyometrenin üst ucu
- W0** : 0. Potansiyometrenin orta ucu
- Sout** : 0. ve 1. potansiyometreler seri bağlanırsa ortak uç çıkışı
- Vcc** : +5V besleme ucu

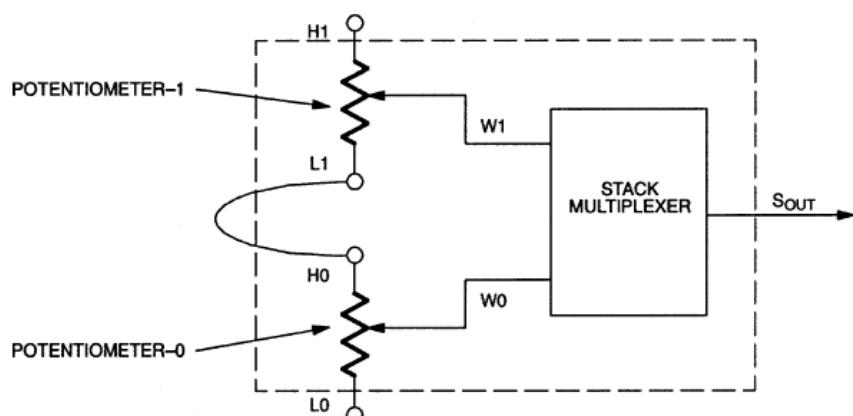
DS1868'i sürmek oldukça basittir. Şekil-97'de iletişim zaman diyagramından da görüleceği gibi DS1868'i kullanmak oldukça basittir.



Şekil 97 – DS1868 Zaman Diyagramı

Şekil-97'de de görüleceği üzere öncelikle $RST=1$ yapılmış, cihaz reset durumundan çıkarılmalıdır. Daha sonra sırasıyla stack biti, 1. potansiyometrenin 8 bitlik kodu daha sonrasında ise 0. Potansiyometrenin 8 bitlik kodu saat darbelerinin yükselen kenarlarında gönderilmelidir.

Buradaki stack select bir şekil-98'de ki gibi potansiyometreler seri bağlandığında ortak ucu belirlemek için kullanılmaktadır. Normal işlemlerde stack bit'inin 1 ya da sıfır olması bir şeyi değiştirmemektedir.



Şekil 98 – Potansiyometrelerin Seri Bağlanması

8 bitlik potansiyometre ayarı bize 1/256 oranında kontrol imkanı sağlar. Örneğin 150k'luk DS1868 kullandığımızda tek bir artış;

$150000/256=585,938$ ohm'luk artışa denk gelmektedir. Dolayısı ile hesaplarımıza bu değere göre ayarlamalıyız.

DS1868 dijital potansiyometresi kullanımı bir kütüphane haline getirirse **ds1868.h** dosyası aşağıdaki gibi olacaktır.

```
/*
 * www.FxDev.org
 * DS1868 Kullanım Klavuzu
 * ds1868_pot(POT, DEGER, STACK); şeklinde kullan
 * POT 1'den büyük olamaz
 * www.FxDev.org
 */

#define DQ RB5
#define CLK RB6
#define RST RB7
```

```
extern void one(void);
extern void zero(void);
extern ds1868_pot(unsigned char pot, unsigned char deger,unsigned char stack);
```

Yukarıda da görüldüğü üzere kütüphane başlığımız oldukça sadedir. Bu fonksiyonları çalıştıran **ds1868.c** ise aşağıdaki gibi olacaktır.

```
#include <pic.h>
#include "ds1868.h"
#include "delay.h"

void one(void)
{
    DQ=1;
    DelayUs(2);
    CLK=1;
    DelayUs(2);
    CLK=0;
    DelayUs(2);
}

void zero(void)
{
    DQ=0;
    DelayUs(2);
    CLK=1;
    DelayUs(2);
    CLK=0;
    DelayUs(2);
}

ds1868_pot(unsigned char pot, unsigned char deger,unsigned char stack)
{
    unsigned char i, veril1, veri2, kay=0x80;

    if(pot>1)
        return;

    if(pot)
        veril1=deger;
    else
        veri2=deger;

    RST=1;
    DelayUs(2);
    if(stack)
        one();
    else
        zero();
    for(i=0;i<8;i++)
    {
        if(veril1&kay)
            one();
        else
            zero();
        kay=kay>>1;
    }
    kay=0x80;
}
```

```

for(i=0;i<8;i++)
{
    if(veri2&kay)
        one();
    else
        zero();
    kay=kay>>1;
}

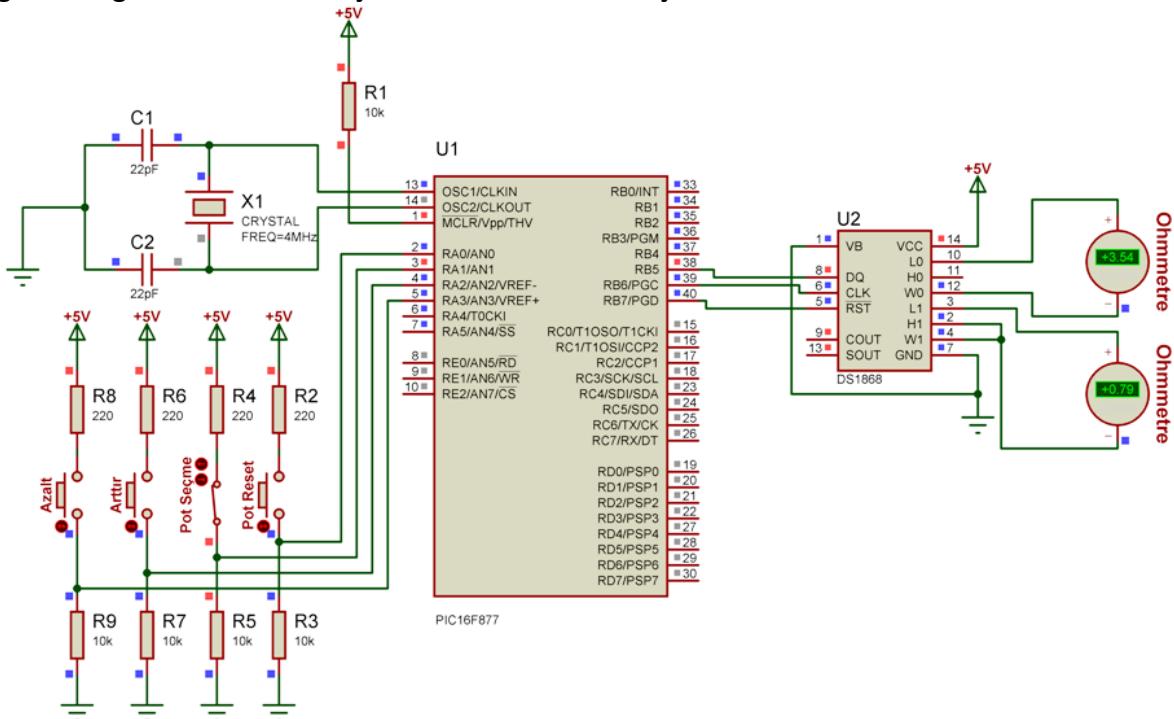
DelayUs(2);
RST=0;
}

```

Görüleceği üzere kütüphanemiz oldukça sadedir. Bu da DS1868'in kullanım kolaylığından kaynaklanmaktadır.

10.3) DS1868 ile Dijital Pot Uygulaması

Bu uygulamamızda DS1868 kullanarak dijital potun L0-W0 ve L1-W1 arasında değişen ohm değerlerini görelim. Öncelikle şekil-99'daki devremizi çizelim.



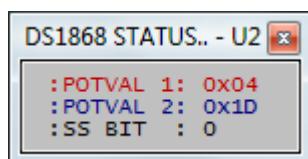
Şekil 99 – DS1868 Uygulaması

Şekil-99'da görülen devreyi çalıştıracak kodlarımız ise aşağıdaki gibidir.

```
#include <htc.h>
#include "delay.h"           // Gecikme kütüphanesi tanimlanıyor
#include "ds1868.h"          // ds1868 kütüphanesi tanimlanıyor

void main(void)
{
    unsigned i=0, j=0;
    ADCON1=0x07;
    PORTA=0x00;                  // PORTB ve PORTA sıfırlanıyor
    PORTB=0x00;
```

Kodlarımızı çalıştırduğumuzda DS1868'in kaydedicileri ise şekil-100'deki gibi olacaktır.



Şekil 100 – DS1868 Kaydedici Değerleri

BÖLÜM 11 – 2x20 LCD KULLANIMI, 74HC595 ENTEGRESİ ve 3-WIRE LCD İŞLEMLERİ

11.1) 2x20 LCD Kullanımı

2x20 LCD, daha önceki bölümlerde bahsettiğimiz 2x16 LCD'den farklı olarak ekstra 8 satır birimine daha sahip yapısıdır. Genel itibarı ile 2x16 LCD kullanımından tek farkı yazılabilceğin alanın büyüklüğüdür. Diğer tüm özellikleri 2x16 LCD ile aynıdır.

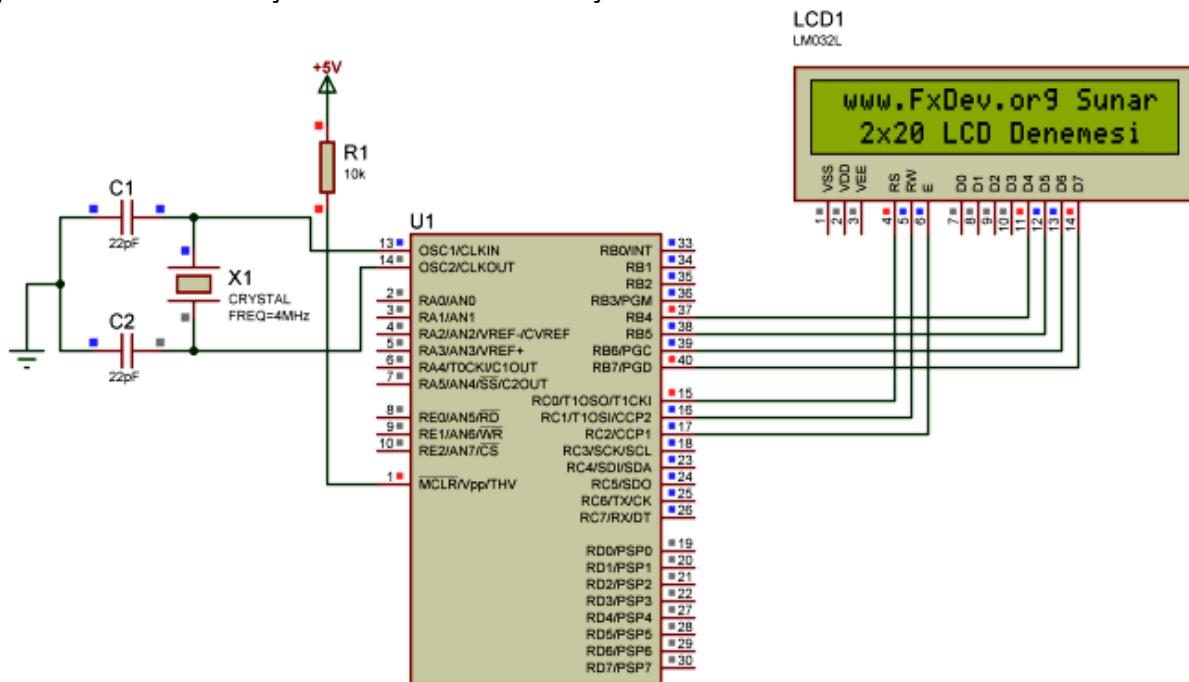
2x16 LCD kütüphanesinde değişecek tek fonksiyon `lcd_gotoxy(y,x)` fonksiyonudur. Bu fonksiyon aşağıdaki kodlarla değiştirilmelidir.

```
void lcd_gotoxy(unsigned char x,unsigned char y)
{
    if(x==1)
        lcd_komut(0x80+((y-1)%20));
    else
        lcd_komut(0xC0+((y-1)%20));
}
```

Eğer 2x16'daki kütüphane dosyasındaki fonksiyona bakarsanız tek farkın %16 bölüm oranı yerin %20 olmasıdır.

11.1.1) 2x20 LCD Uygulaması

Kütüphanemizi yukarıdaki gibi düzenledikten sonra üst ve alt satırda istedigimiz yazıyi yazdıralım. Öncelikle şekil-101'deki devremizi çizelim.



Şekil 101 – 2x20 LCD Uygulaması

Şekil-101'deki işlemi yapan kodumuz ise aşağıdaki gibidir.

```
#include <htc.h>
#include "delay.h"           // Gecikme kütüphanesi tanımlanıyor
#include "lcd.h"             // ds1868 kütüphanesi tanımlanıyor

void main(void)
{
    PORTB=0x00;
    TRISB=0x00;           // PORTB ve PORTC çıkış
    TRISC=0x00;

    lcd_init();

    lcd_yaz(" www.FxDev.org Sunar!");
    lcd_gotoxy(2,1);
    lcd_yaz(" 2x20 LCD Denemesi");

    for(;;);
}
```

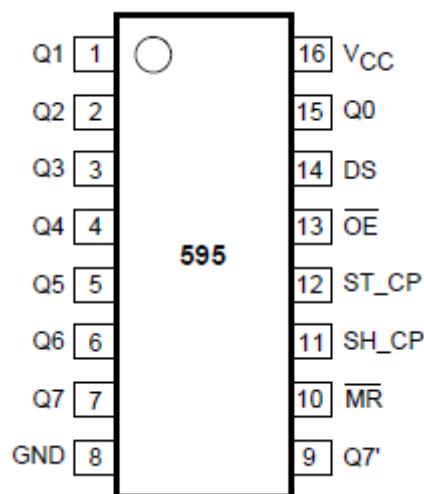
Gördüğü gibi kodlarımız oldukça sadedir. Uygulamanın gerçekleştirilmiş hali ise şekil-102'de görülebilmektedir.



Şekil 102 – 2x20 LCD Uygulamasının Gerçeklenmesi

11.2) 74HC595 Entegresi

Şekil-103'te görülebilecek 74HC595 entegresi 100Mhz'e kadar çalışabilen 8 bitlik bir shift registerdir. Bu özelliği sayesinde 74HC595 entegresi ayrıca DS ucundan girilen seri bilgileri istenildiği an Q0..Q7 uçlarından paralel bilgi olarak alınmasına olanak sağlar.



Şekil 103 – 74HC595 DIP Yapısı

Şekil-103'te görülen pinlerin görevleri ise şöyledir;

Q0..Q7- Paralel çıkışlar

DS - Seri data girişi

OE - Çıkış açık

SH_CP - Shift register saat kaynağı

ST_CP - Kaydedici registerin saat kaynağı

MR - Reset ucu

Q7' - Seri data çıkışı

GND - Toprak ucu

Vcc - +5V

74HC595'in çalışma mantığı ise şöyledir;

- OE ucu toprak hattına, MR ucu ise +5V'a bağlanır,
- DS ucundan seri bilgi girişi SH_CP'nin her yükselen kenarında shift edilerek kaydedici registere yazılır,
- 8 çıkışımız olduğu için bu işlem 8 kez gerçekleştirilir,
- Tüm bu işlemler gerçekleştirken Q0..Q7 çıkışlarında bir değişiklik olmaz,
- Son olarak kaydırma işlemi bittiğinde kaydedici registerdeki değerin Q0..Q7 uçlarına yansımıası için ST_CP ucuna yükselen kenarda sinyal verilir,
- Son olarak Q0..Q7 uçlarında, ilk giren bilgi Q7 de son giren bilgi ise Q0'da olmak koşulu ile sıralama tamamlanır.

Örnek olarak 0xA5'nın ucılara iletimini örnek verelim;

0xA5=0b10101001 olur. İlk giren Q7'de olacağına göre öncelikle A5'in en yüksek biti olan 7. Biti göndermeliyiz. Daha sonra ise 6,5,4.. şeklinde bunu sürdürmeliyiz.
C işleminde bunu aşağıdaki fonksiyon ile sağlayabiliriz.

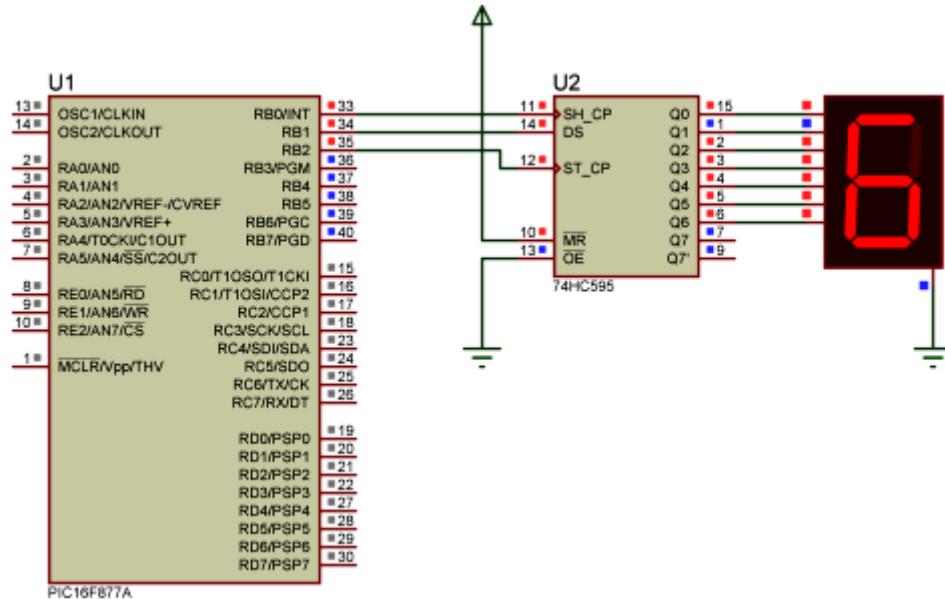
```
void three_wire_control(unsigned char temp)
{
    char i;
    Enable=0;
    for(i=0;i<8;i++)
    {
        Clock=0;
        if((temp<<i)&0x80)
            DataIO=1;
        else
            DataIO=0;
        Clock=1;
    }
    Enable=1;
}
```

İşlem takip edilirse öncelikle 7. bitin daha sonra ise diğer düşük bitlerin, en sonunda ise 0. bitin çıkışa gönderildiği görülebilir.

Bu şekilde port genişletme işlemimizde (3'ten 8'e) gerçekleştirmiştir.

11.2.1) 74HC595 ile 7-Segment Sürümü

Fonksiyonumuzu tamamladıktan sonra artık bağlantı uçlarına dikkat ederek istediğimiz 8 bitlik her aracı kontrol edebiliriz. Bu uygulamada bir adet 7-Segment ile 0-9 sayımla işlemi yapacağız. Öncelikle şekil-104'teki devremizi çizelim.



Şekil 104 – 74HC595 ile 7-Segment Uygulaması

Tanımladığımız fonksiyonu kullanarak yazdığımız kod ise aşağıdaki gibi olacaktır.

```
#include <htc.h>
#include "delay.h" // Gecikme kütüphanesi tanımlanıyor

#define Clock RB0 //74LS595 Clk girişi, yükselen kenar
#define DataIO RB1 //74LS595 Data girişi
#define Enable RB2 //74LS595 Enable girişi

// Seven segment sabitleri
const unsigned char
segment[]={0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F};

void three_wire_control(unsigned char temp) // 3-Wire kütüphanesi
{
    char i;
    Enable=0;
    for(i=0;i<8;i++)
    {
        Clock=0;
        if((temp<<i)&0x80)
            DataIO=1;
        else
            DataIO=0;
        Clock=1;
    }
    Enable=1;
}

void main(void)
{
    char i;
```

```

PORTB=0x00;
TRISB=0x00; // PORTB çıkış

for(;;)
{
    three_wire_control(segment[i]); // Seven segment
    DelayMs(250);DelayMs(250); // bilgisi gönderiliyor
    i++;
    if(i>9)
        i=0;
}
}

```

Burada dikkat edilmesi gereken konu 7-segment'in ortak katot kullanılmış olmasıdır.

11.2.2) 74HC595 ile LCD Sürümü

Görüldüğü üzere 74HC595 ile 3-Wire iletişimini kullanmak oldukça kolaydır.

Fakat bu işlemi LCD'de gerçekleştirmek için öncelikle LCD kütüphanesinde bazı değişiklikler yapmalıyız. Yapacağımız modifiyeyi örnekini tek fonksiyonda gösterirsek, kodlarımız aşağıdaki gibi olacaktır.

```

void lcd_komut(unsigned char c)
{
    unsigned char temp;
    temp = 0x00;
    temp |= 0x04;
    three_wire_control(temp);
    temp |= (c & 0xF0);
    three_wire_control(temp);
    temp &= 0xF0;
    three_wire_control(temp);
    lcd_busy();
    temp = 0x00;
    temp |= 0x04;
    three_wire_control(temp);
    temp |= ((c & 0x0F)<<4);
    three_wire_control(temp);
    temp &= 0xF0;
    three_wire_control(temp);
    lcd_busy();
}

```

Kodları dikkatle incelerseniz aslında yapılan işin tüm giriş çıkışları 8 bit haline getirmek olduğu rahatlıkla görülebilir.

Tüm kodları bu şekilde modifiye edip, son olarak 3-Wire fonksiyonumuzu da ekleyip kütüphane haline getirirsek **three_wire_lcd.h** dosyamız aşağıdaki gibi olacaktır.

```

/*
 * www.FxDev.org
 * 3 Kablolu 2x16 LCD Kullanım Klavuzu
 * 74LS595 ile birlikte kullanılmalıdır.
 * RW/RS/EN pinleri ise sırasıyla 74LS595'in Q0.,Q1. ve Q2. pinlerine
 * D4,D5,D6,D7 pinleri sırasıyla 74LS595'in Q4.,Q5.,Q6. ve Q7. pinlerine
 * bağlanmalıdır.
 * Cursor kapalıdır.
 *
 * lcd_init(); ile LCD'nin ilk ayarlarını yap

```

```

* lcd_clear(); ile LCD'yi sil
* lcd_yaz("deneme"); şeklinde yazı yazdır.
* veri_yolla('F'); şeklinde tek ascii kodu yazdır.
* lcd_gotoxy(1,13); şeklinde LCD'nin istenilen yerine git.
*
* www.FxDev.org
*/

#define Clock      RB0          //74LS595 Clk girişi, yükselen kenar
#define DataIO     RB1          //74LS595 Data girişi
#define Enable     RB2          //74LS595 Enable girişi

/* LCD'de kullanılan komutların tanımlaması*/
#define Sil         1           // Ekrani temizler
#define BasaDon    2           // Imleci sol üst köseye getirir
#define SolaYaz   4           // Imlecin belirttiği adres azalarak gider
#define SagaYaz   6           // Imlecin belirttiği adres artarak gider
#define ImlecGizle 12          // Göstergeyi aç, kursor görünmesin
#define ImlecAlta  14          // Yanıp sönen blok kursor
#define ImlecYanSon 15          // Yanıp sönen blok kursor
#define ImlecGeri  16          // Kursorü bir karakter geri kaydırır
#define KaydirSaga 24          // Göstergeyi bir karakter sağa kaydırır
#define KaydirSola  28          // Göstergeyi bir karakter sola kaydırır
#define EkraniKapat 8           // Göstergeyi kapat (veriler silinmez)
#define BirinciSatir 128         // LCD'nin ilk satır baslangıç adresi
                             // (DDRAM adres)
#define IkinciSatir 192         // Ikinci satırın baslangıç adresi
#define KarakUretAdres 64         // Karakter üreticisi adresini belirle
                             // (CGRAM adres)

/* LCD'de Kullanılan Fonksiyon Seçimi */
#define CiftSatir8Bit 56          // 8 bit ara birim, 2 satır, 5*7 piksel
#define TekSatir8Bit  48          // 8 bit ara birim, 1 satır, 5*7 piksel
#define CiftSatir4Bit 40          // 4 bit ara birim, 2 satır, 5*7 piksel
#define TekSatir4Bit  32          // 4 bit ara birim, 1 satır, 5*7 piksel

extern void three_wire_control(unsigned char temp);
extern void veri_yolla(unsigned char);
extern void lcd_clear(void);
extern void lcd_yaz(const char * s);
extern void lcd_gotoxy(unsigned char x, unsigned char y);
extern void lcd_init(void);
extern void lcd_komut(unsigned char c);

```

Tüm bu fonksiyonları yerine getiren **three_wire_lcd.c** dosyamız ise aşağıdaki gibi olacaktır.

```

#include <pic.h>
#include "three_wire_lcd.h"
#include "delay.h"

void three_wire_control(unsigned char temp)
{
    char i;
    Enable=0;
    for(i=0;i<8;i++)
    {
        Clock=0;
        if((temp<<i)&0x80)
            DataIO=1;
        else
            DataIO=0;
        Clock=1;

```

```

        }
        Enable=1;
    }

void lcd_busy(void)
{
    DelayUs(100);
}

void lcd_komut(unsigned char c)
{
    unsigned char temp;
    temp = 0x00;
    temp |= 0x04;
    three_wire_control(temp);
    temp |= ( c & 0xF0 );
    three_wire_control(temp);
    temp &= 0xF0;
    three_wire_control(temp);
    lcd_busy();
    temp = 0x00;
    temp |= 0x04;
    three_wire_control(temp);
    temp |= ( (c & 0x0F)<<4 );
    three_wire_control(temp);
    temp &= 0xF0;
    three_wire_control(temp);
    lcd_busy();
}
}

void veri_yolla(unsigned char c)
{
    unsigned char temp;
    temp = 0x00;
    temp |= 0x05;
    three_wire_control(temp);
    temp |= ( c & 0xF0 );
    three_wire_control(temp);
    temp &= 0xF1;
    three_wire_control(temp);
    lcd_busy();
    temp = 0x00;
    temp |= 0x05;
    three_wire_control(temp);
    temp |= ( (c & 0x0F)<<4 );
    three_wire_control(temp);
    temp &= 0xF1;
    three_wire_control(temp);
    lcd_busy();
}
}

void lcd_clear(void)
{
    lcd_komut(0x1);
    DelayMs(2);
}

void lcd_yaz(const char * s)
{
    lcd_busy();
    while(*s)

```

```

    veri_yolla(*s++);
}

void lcd_gotoxy(unsigned char x,unsigned char y)
{
    if(x==1)
        lcd_komut(0x80+((y-1)%20));
    else
        lcd_komut(0xC0+((y-1)%20));
}

void lcd_init()
{
    unsigned char temp=0;

    three_wire_control(temp);

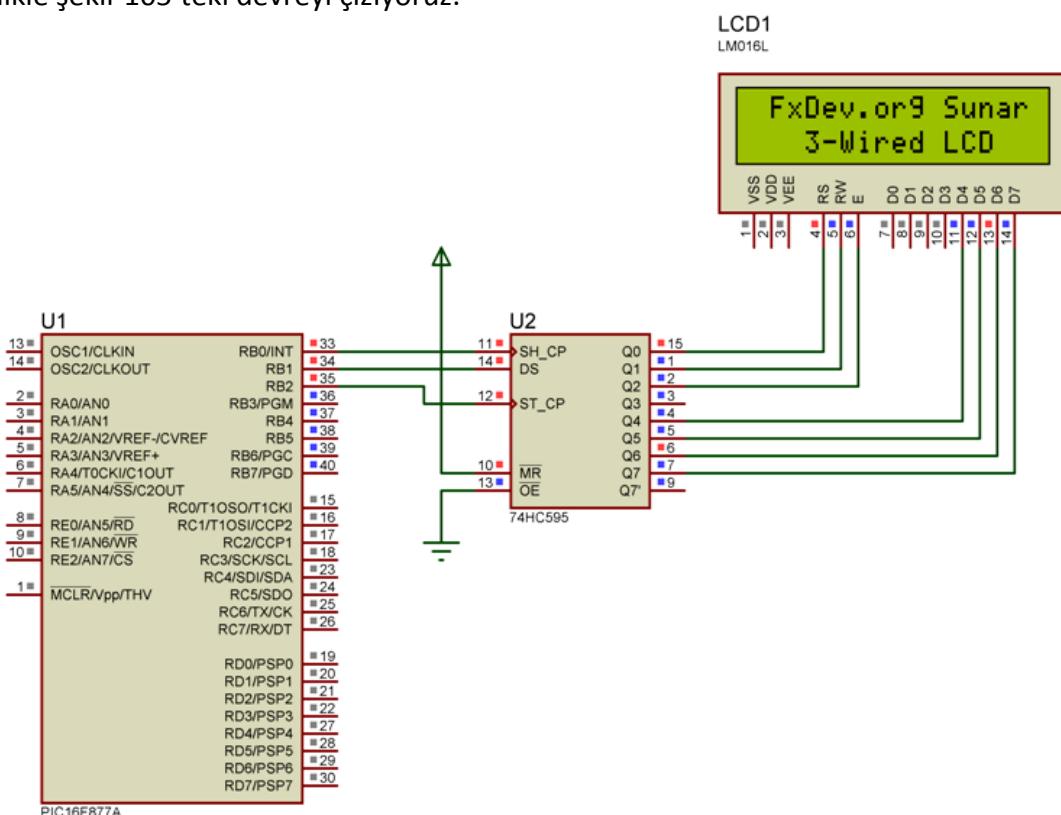
    DelayMs(15);
    lcd_komut(0x02);
    DelayMs(2);

    lcd_komut(CiftSatir4Bit);
    lcd_komut(SagaYaz);
    lcd_komut(ImlecGizle);
    lcd_clear();
    lcd_komut(BirinciSatir);
}

```

Görüleceği gibi kodlar karışık gelebilir. Fakat biraz incelediğinizde işin o kadar da karmaşık olmadığını göreceksiniz.

Kütüphanemizi tanımladıktan sonra, istediğimiz herhangi bir yazıyı LCD'mize yazdıralım. Öncelikle şekil-105'teki devreyi çiziyoruz.



Şekil 105 – 74HC595 ile 2x16 LCD Uygulaması

Şekil-105'teki devreyi çalıştırın kodumuz ise aşağıdadır.

```
#include <htc.h>
#include "delay.h"                      // Gecikme kütüphanesi tanımlanıyor
#include "three_wire_lcd.h"              // 3-Wire LCD kütüphanesi tanımlanıyor

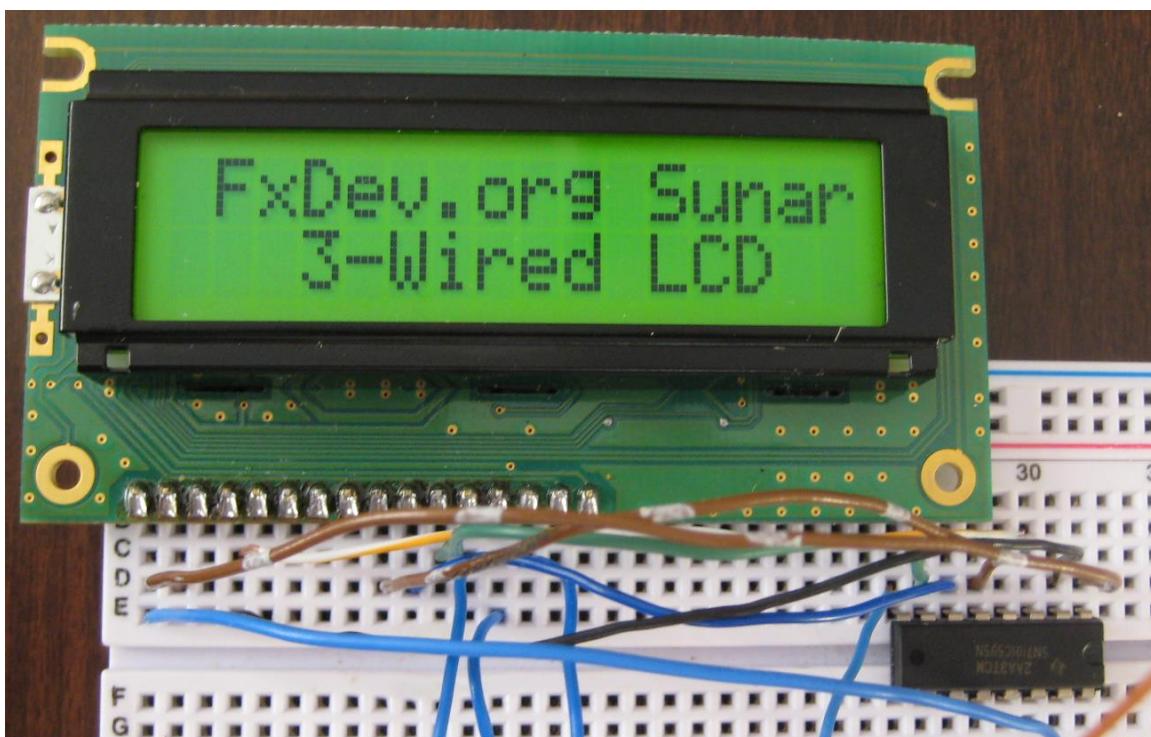
void main(void)
{
    PORTB=0x00;
    TRISB=0x00;                         // PORTB çıkış

    lcd_init();                          // LCD ilk ayarları yapılıyor

    lcd_yaz(" FxDev.org Sunar");
    lcd_gotoxy(2,1);
    lcd_yaz(" 3-Wired LCD");

    for(;;);
}
```

Gördüldüğü gibi kodlarımızın diğer LCD sürmeden hiçbir farkı yoktur. Devrenin gerçekleşmiş halini (LCD ve 74HC595'i) ise şekil-106'da görebilirsiniz.



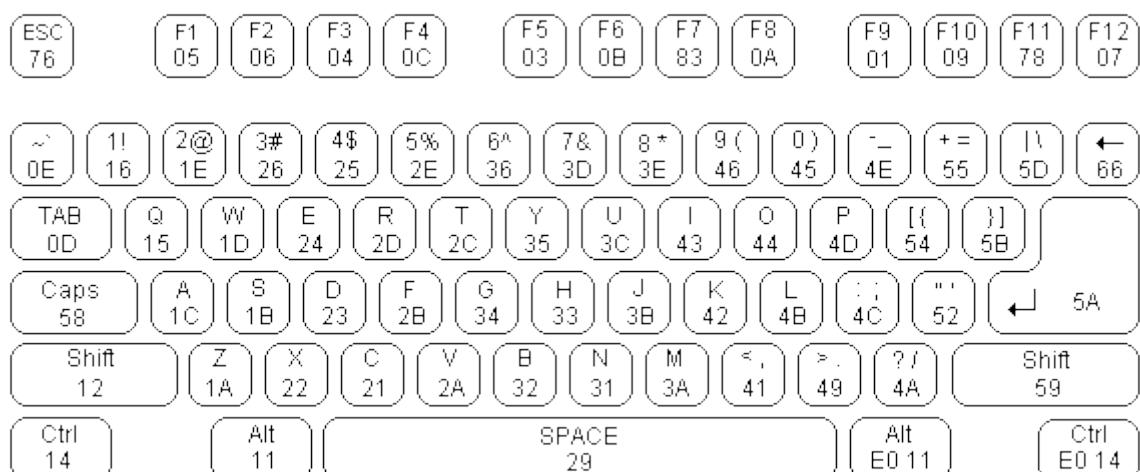
Şekil 105 – 74HC595 ile 2x16 LCD Uygulamasının Gerçeklenmesi

BÖLÜM 12 – PS/2 KLAVYE KULLANIMI

12.1) Klavye Çalışma Mantığı ve Kütüphanesi

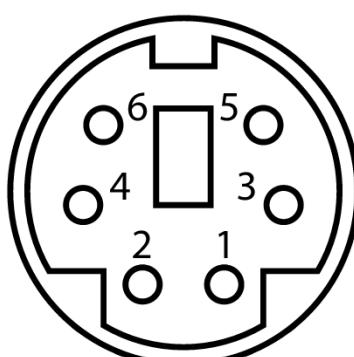
İlk kez yazı makinesinin mucidi olan Christopher Latham Sholes tarafından 1867'de ortaya çıkan klavye birimi o günden günümüze kadar teknolojiye ayak uydurması dışında pek bir değişikliğe uğramadı. Bugün birçok bilgisayarda kullanılan Q klavyenin de mucidi olan Sholes, bu kadar karmaşık harf dizilimini mühendisliğe aykırı olarak yazıların daha yavaş yazılması dolayısı ile makinelerinin daha az bozulması için oluşturdu.

Klavye çalışma mantığı oldukça basittir. Her tuş basımında klavye kendine has 8 bitlik özel tuş bilgilerini gönderir, bu kodlar çözülerek hangi tuşun basıldığı kararı alınır ve kullanılır. Basılan tuş dahilinde hangi bilgilerin bizlere gönderildiği şekil-106'daki tablolardan görülebilir.



Şekil 1706 - Klavye Özel Kodları

Klavyerler çeşitli bağlantı şekillerine sahip olsalar da dünya çapında en çok kullanılan bağlantı şekillerinden biri DIN ya da PS/2'dir. Şekil-107'de PS/2 yapısının giriş birimi görülmektedir.



Şekil 107 - PS/2 Bağlantı Şekli

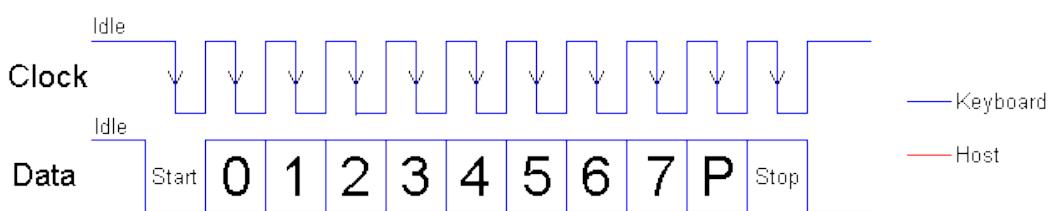
Şekil-107'de de görüleceği üzere PS/2 yapısında 6 farklı bağlantı noktası bulunmaktadır. Sırasıyla kaç numaralı pinin ne işe yaradığını söyleyecek olursak;

- 1- Data hattıdır
- 2- Boş hat, bağlantı yapılmayacak
- 3- GND, toprak hat ucudur
- 4- VDD, güç ucudur, +5V verilecek hattır
- 5- Clk ucudur, iletişim saatı buradan verilir
- 6- Boş hat, bağlantı yapılmayacak

Önemli not: Bu yapı giriş birimi olduğu için klavyeden direk hat çekileceğse, tam simetriğine göre işlem yapılmalıdır. Aksi takdirde klavyeniz zarar görebilir.

Biz iletişimimizi klavyeden kaynağımıza doğru yapacağımız için şekil-108'deki zaman diyagramını kullanacağız.

Gördüğü üzere klavye her bir tuş bilgisi için şekil-108'de görülen 20 ile 30 KHz arasında saat darbesi ve data sinyali üretmektedir.



Şekil 108 - Klavyeden Kaynağa

Şekil-108'deki zaman diyagramını yorumlayacak olursak;

- Klavyeden hiçbir tuşa basılmaz iken saat ve data yolu yüksek seviyede duracaktır.
- Klavyeden herhangi bir tuşa basıldığında ilk saat darbesinde data hattı sıfıra çekilir.
- Daha sonraki 8 saat darbesinde tuşa ait, şekil-106'daki özel kodlardan biri gönderilir.
- Daha sonra parity biti gönderilir, klavye tek parity sistemini kullanır
- En sonda da iletişimini bittiği yönünde 11. Saat darbesinde bitiş bilgisi gönderilerek iletişim sonlandırılır.

Burada belirttiğimiz bu yapıyı anlamak için aşağıdaki C fonksiyonu kullanılabilir. Parity bizim için önemli olmadığından kontrol ihtiyacı duyulmamıştır. Fonksiyon geri dönüş olarak klavyeden algıladığı özel kodu gönderecektir.

```
unsigned char get_byte(void)
{
    char i;
    unsigned char veri=0;

    for(i=0;i<11;i++) //11 bitlik veri gelecek
    {
        while(Clock); //İlk önce Clock=1 olmalı
        if( (i<9) && (i>0) ) //2->9 bitler
        {
            veri=veri>>1;
            if(Data==1)
                veri=veri | 0x80;
        }
        while(!Clock); //Son olarak Clock=0 olmalı
    }
    return veri; //Algılanan değer gönderiliyor
}
```

Klavye için yukarıdaki kodu yazmak başlı başına yeterli değildir. Klavyede her tuşa basıldığında o tuşun özel kodu, tuş bırakıldığında o tuşa ait başka bir kod ve tekrar o tuşun kendi kodu gönderilir. Örneğin bazı tuşlara basılıp bırakıldığında aşağıdaki kodlar sırasıyla klavyeden gönderilecektir;

A: 0x1C -> 0xF0 -> 0x1C
H: 0x33 -> 0xF0 -> 0x33
1: 0x16 -> 0xF0 -> 0x16

Görüldüğü üzere 0xF0 burada kilit rol oynamaktadır. Eğer sadece yukarıdaki C kodunu kullanacak olursak, tek tuşa bastığımızda sanki 3 ayrı tuşa basmış gibi olacağız. Bunu engellemek için aşağıdaki kod sistemini kullanabiliriz.

```
data=get_byte();
while(get_byte()==0xF0);
```

Burada algılama işlemini ister 0xF0'dan önce, isterseniz de sonda yapabiliriz. Daha sonrası ise algıladığımız kodun ASCII karşılığını bulmaktan ibarettir.

Tüm bunları düşünerek oluşturacağımız kütüphane dosyasının **keyboard.h** dosyası aşağıdaki gibi olacaktır.

```
/*
 * www.FxDev.org
 * Keyboard Kullanım Klavuzu
 * Shift ve Capslock tuşu ile büyük, küçük ve tuşların ikinci görevleri
 * kullanılabilir.
 * i=get_byte(); şeklinde klavyeden gelen byte'lik veriler okunur.
 * i=keyboard(); şeklinde klavyeden okunan kod direkt ascii olarak
 * gönderilir.
 * www.FxDev.org
 */

#define Clock RA0 //Pin tanımlamaları
#define Data RA1

extern unsigned char get_byte(void);
extern unsigned char keyboard(void);
```

Bu fonksiyonları çalıştırılan **keyboard.c** dosyamız ise şöyle olacaktır.

```
#include <pic.h>
#include "keyboard.h"

const unsigned char kucuk[67][2]={
    {0x0d,0x7e},{0x0e,'!'}, {0x15,'q'}, {0x16,'1'},
    {0x1a,'z'}, {0x1b,'s'}, {0x1c,'a'}, {0x1d,'w'},
    {0x1e,'2'}, {0x21,'c'}, {0x22,'x'}, {0x23,'d'},
    {0x24,'e'}, {0x25,'4'}, {0x26,'3'}, {0x29,' '},
    {0x2a,'v'}, {0x2b,'f'}, {0x2c,'t'}, {0x2d,'r'},
    {0x2e,'5'}, {0x31,'n'}, {0x32,'b'}, {0x33,'h'},
    {0x34,'g'}, {0x35,'y'}, {0x36,'6'}, {0x39,'.'},
    {0x3a,'m'}, {0x3b,'j'}, {0x3c,'u'}, {0x3d,'7'},
    {0x3e,'8'}, {0x41,0xef}, {0x42,'k'}, {0x43,3},
    {0x44,'o'}, {0x45,'0'}, {0x46,'9'}, {0x49,7},
    {0x4a,'.'}, {0x4b,'1'}, {0x4c,2}, {0x4d,'p'},
    {0x4e,'+'}, {0x52,'i'}, {0x54,5}, {0x55,0x5f},
```

```

        {0x5a,0xa3}, {0x5b,0xf5}, {0x5d,' ',''}, {0x61,'<'},
        {0x69,'1'}, {0x6b,'4'}, {0x6c,'7'}, {0x70,'0'},
        {0x71,' ',''}, {0x72,'2'}, {0x73,'5'}, {0x74,'6'},
        {0x75,'8'}, {0x79,'+'}, {0x7a,'3'}, {0x7b,'-'},
        {0x7c,'*'}, {0x7d,'9'}, {0,0}
};

const unsigned char buyuk[67][2]={
    {0x0d,0x7e}, {0x0e,'é'}, {0x15,'Q'}, {0x16,'!'},
    {0x1a,'Z'}, {0x1b,'S'}, {0x1c,'A'}, {0x1d,'W'},
    {0x1e,'"'}, {0x21,'C'}, {0x22,'X'}, {0x23,'D'},
    {0x24,'E'}, {0x25,'+'}, {0x26,'^'}, {0x29,' '},
    {0x2a,'V'}, {0x2b,'F'}, {0x2c,'T'}, {0x2d,'R'},
    {0x2e,'%'}, {0x31,'N'}, {0x32,'B'}, {0x33,'H'},
    {0x34,'G'}, {0x35,'Y'}, {0x36,'&'}, {0x39,'L'},
    {0x3a,'M'}, {0x3b,'J'}, {0x3c,'U'}, {0x3d,'/'},
    {0x3e,'('}, {0x41,0}, {0x42,'K'}, {0x43,'I'},
    {0x44,'O'}, {0x45,'='}, {0x46,')'}, {0x49,7},
    {0x4a,':'}, {0x4b,'L'}, {0x4c,1}, {0x4d,'P'},
    {0x4e,'?'}, {0x52,6}, {0x54,5}, {0x55,0x2d},
    {0x5a,0xa3}, {0x5b,4}, {0x5d,';'}, {0x61,'>'},
    {0x69,'1'}, {0x6b,'4'}, {0x6c,'7'}, {0x70,'0'},
    {0x71,' ',''}, {0x72,'2'}, {0x73,'5'}, {0x74,'6'},
    {0x75,'8'}, {0x79,'+'}, {0x7a,'3'}, {0x7b,'-'},
    {0x7c,'*'}, {0x7d,'9'}, {0,0}
};

unsigned char get_byte(void)
{
    char i;
    unsigned char veri=0;

    for(i=0;i<11;i++) //11 bitlik veri gelecek
    {
        while(Clock); //İlk önce Clock=1 olmalı
        if( (i<9) && (i>0)) //2->9 bitler
        {
            veri=veri>>1;
            if(Data==1)
                veri=veri | 0x80;
        }
        while(!Clock); //Son olarak Clock=0 olmalı
    }
    return veri; //Algılanan değer gönderiliyor
}

unsigned char keyboard(void)
{
    unsigned char data=0;
    unsigned char shift=0, i;

    data=get_byte();
    while(get_byte()==0xF0);

    if((data==0x12) || (data==0x59)) //Sağ ya da sol shifte mi basıldı
        shift=!shift; //Capslock gibi çalışacak

    if(data==0x58) //Capslock'a basıldıysa
        shift=!shift;

    for(i=0;kucuk[i][0]!=data && kucuk[i][0]; i++);
}

```

```

//Tuşun ascii karşılığı bulunuyor

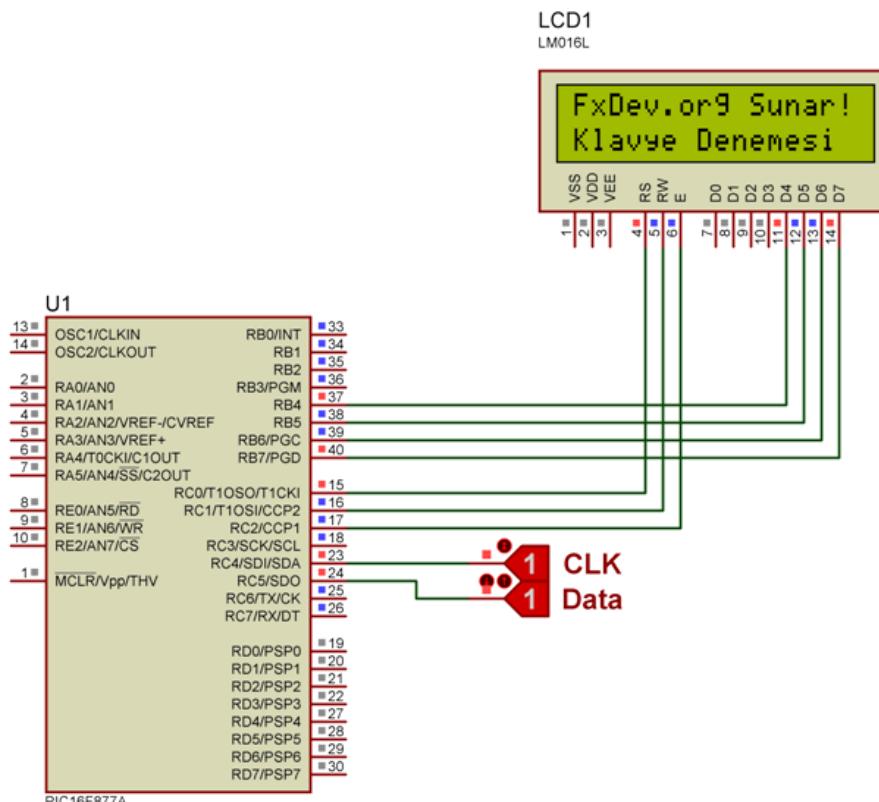
    if(kucuk[i][0]==data)
//Data ascii tablolarımızda var mı bakılıyor
    {
        if(shift==0)
//Capslock açık değilse küçük harf, aksa büyük harfin ascii değeri geri
gönderiliyor
            return kucuk[i][1];
        else
            return buyuk[i][1];
    }
}

```

Kodlarımızda da göreceğimiz üzere burada matris dizi mantığını kullandık. Klavye kodu değerlerinin neler olduğunu internetten bulup sizde istediğiniz şekilde dizi oluşturabilirsiniz.

12.2) Klavye Uygulaması

Bu uygulamada klavyede basılan herhangi bir tuşun LCD'de görünmesini sağlayacağız. Öncelikle şekil-109'daki devremizi çiziyoruz.



Şekil 109 – Klavye Uygulaması

Proteus'ta klavye almadığı için onun yerine CLK ve Data uçları giriş olarak verilmiştir. Simülasyon yapmak isterseniz şekil-108'deki zaman diyagramındaki sinyalleri elle vererek simülasyon yapabilirsiniz.

Şekil-109'deki devreyi çalıştırın kodumuz ise aşağıdaki gibidir.

```
#include <htc.h>
#include "delay.h"
#include "lcd.h"          // LCD ve Klavye kütüphaneleri
#include "keyboard.h"     // Tanımlanıyor

void main(void)
{
    unsigned char i,j;
    PORTB=0x00;
    PORTC=0x00;
    TRISB=0x00; // LCD için çıkış, klavye için girişler
    TRISC=0x30; // tanımlanıyor

    lcd_init();
    lcd_yaz("FxDev.org Sunar!");
    lcd_gotoxy(2,1);
    lcd_yaz("Klavye Denemesi");
    DelayMs(250);DelayMs(250);DelayMs(250);DelayMs(250);
    lcd_clear();
    lcd_komut(ImlecAltta); // İmlec alta geçecek

    for(;;)                  // Klavyeden basılan karakterler
    {                        // LCD'ye yazılıyor
        i=keyboard();
        j++;

        if(j==17)
        {
            lcd_gotoxy(2,1);
        }
        else if(j==33)
        {
            lcd_clear();
            lcd_gotoxy(1,1);
            j=0;
        }
        veri_yolla(i);
    }
}
```

Göründüğü gibi kodlarımız oldukça sadedir. Eğer projeniz 40-50 buton gerektiren bir devre ise, bu şekilde klavye kullanmak kolay olabilir.

BÖLÜM 13 – GRAFİK LCD

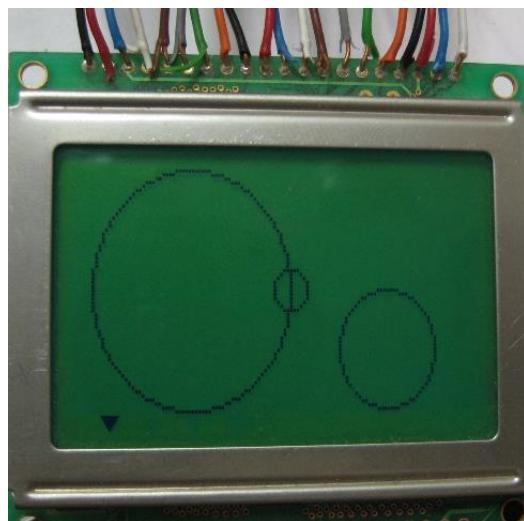
13.1) Hi-Tech GLCD İşlemleri

Grafik LCD'ler yapılarında karakter LCD'ler gibi hazır karakterler bulundurmayan, fakat karakter LCD'lerin yapamadığı grafik çizimleri rahatlıkla yapabilen LCD'lerdir. Grafik LCD'lerin çalıştırılması diğer LCD'lere göre zordur.

GLCD'lerin çalışma mantıkları ise şöyledir, belirli komutlarla istenilen bölgesi yazılabilen boş bir 124x64 bitlik DDRAM'in istenilen alanı yazıldığında GLCD'de de istenilen bölge yazılmış olur.

Örneğin DDRAM bölgesinin 23x12. adresi 1 yapıldığında GLCD'de de 23x12. pixel koyulaşır. Bu şekilde istenilen tüm işlemler GLCD'de yaptırılabılır.

Piyasada oldukça fazla çeşitte GLCD bulmak mümkündür. Gerek piyasada türevlerinin çok oluşu, gerek diğer GLCD'lere nazaran kullanımı kolay olması nedeniyle günümüzde en çok tercih edilen şekil-110'da da görülebilecek GLCD, 124x64 pixel boyutunda olan KS0108 GLCD'sidir.



Şekil 109 – KS0108 GLCD

Piyasada farklı uçlara sahip modeller olsa da GLCD'lerin genel bacak bağlantıları aşağıdaki gibidir;

Vss	- Toprak ucu
Vdd	- +5V
D/I	- Data veya komut verme ucu
R/W	- Yazma veya okuma yapıldığını belirten uç
EN	- Yetki verme ucu
DB0..DB7	- Data giriş uçları
CS1	- İlk 64x64 bitlik chip'i seçme biti
CS2	- İkinci 64x64 bitlik chip'i seçme biti
RES	- Reset ucu
Vee	- Kontrast için gerekli -10V ucu
K	- Backlight eksi ucu
A	- Backlight artı ucu

Yukarıdaki üç bağlantıları GLCD'den GLCD'ye farklılık gösterebileceğinden doğru bağlantı bacakları için GLCD'nin datasheet'ine bakmalısınız.

KS0108'de dikkat edilmesi gereken bir noktada hafıza biriminin 64x64 şeklinde iki parçadan oluşmasıdır. CS1 seçili, CS2 seçili değilken bilgiler ilk 64x64 birimlik alan için geçerliken, tam tersi durumunda gönderilen ya da alınan bilgiler ikinci alanı kapsamaktadır. Dolayısı ile kod yazılırken buna çok dikkat edilmelidir.

Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Function
Display ON/OFF	0	0	0	0	1	1	1	1	1	0/1	Controls the display on or off. Internal status and display RAM data are not affected. 0:OFF, 1:ON
Set Address	0	0	0	1	Y address (0~63)						Sets the Y address in the Y address counter.
Set Page (X address)	0	0	1	0	1	1	1	Page (0~7)			Sets the X address at the X address register.
Display Start Line	0	0	1	1	Display start line (0~63)						Indicates the display data RAM displayed at the top of the screen.
Status Read	0	1	B U S Y	0	O N / O F F	R E S E T	0	0	0	0	Read status. BUSY 0 : Ready 1 : In operation ON/OFF 0 : Display ON 1 : Display OFF RESET 0 : Normal 1 : Reset
Write Display Data	1	0	Write Data								Writes data (DB0:7) into display data RAM. After writing instruction, Y address is increased by 1 automatically.
Read Display Data	1	1	Read Data								Reads data (DB0:7) from display data RAM to the data bus.

Yukarıdaki tabloda ise KS0108'i kontrol edebilmemiz için gerekli kodlar bulunmaktadır. GLCD de görüntü elde etmek için gerekli olan fonksiyonlar ise şöyledir;

- KS0108 ilk ayarları : GLCD'yi temizlemek için de kullanılabilir, GLCD il ayarlarını yapar.
- KS0108'e yazım : GLCD'ye herhangi bir bilgiyi yazmaya yarar
- KS0108'den okuma : GLCD'den herhangi bir bilginin okunmasına yarar
- KS0108 pixel : GLCD'nin herhangi bir pixelini aktif veya pasif yapmaya yarar

Yukarıdaki dört fonksiyonla istenilen her türlü şekil, çeşitli matematiksel fonksiyonlar kullanılarak çizilebilir.

Tablodaki önergeleri dikkate alarak yazdığımız **ks0108.h** dosyası aşağıdaki gibi olacaktır.

```
/*
 * www.FxDev.org
 * KS0108 GLCD Kullanım Klavuzu
 * ks0108_init(); ile ilk ayarları yap
 * ks0108_write(0,0x55); şeklinde veri yaz
 * veri=ks0108_read(0); şeklinde veri oku
 * ks0108_pixel(1,1,1) şeklinde herhangi bir pixeli aç ya da kapa
 * ks0108_fill(1); şeklinde tüm ekran pixelleri açılır ya da kapanır
 * ks0108_line(10,10,20,50,1); şeklinde belirtilen koordinatlarda çizgi
 * ks0108_rect(10,10,20,20,1,1); şeklinde belirtilen koordinatlarda dikdörtgen çiz
 * ks0108_circle(10,10,10,1,1); şeklinde daire çiz
 * www.FxDev.org
 */

#define TRIS GLCD TRISC //Pin tanımlamaları
```

```

#define GLCD_PORT PORTC
#define CS1 RB0
#define CS2 RB1
#define DI RB2 // RS'de olabilir
#define RW RB3
#define EN RB4
#define RST RB5

extern void ks0108_init(void);
extern void ks0108_write(unsigned char chip, unsigned char veri);
extern unsigned char ks0108_read(unsigned char chip);
extern void ks0108_pixel(unsigned char x, unsigned char y, unsigned char renk);
extern void ks0108_fill(unsigned char renk);
extern void ks0108_line(unsigned char x1, unsigned char y1, unsigned char x2, unsigned char y2, unsigned char renk);
extern void ks0108_rect(unsigned char x1, unsigned char y1, unsigned char x2, unsigned char y2, unsigned char dolu, unsigned char renk);
extern void ks0108_circle(unsigned char x, unsigned char y, unsigned char r, unsigned char dolu, unsigned char renk);

```

Görüldüğü üzere bu kütüphanemizde oldukça fazla kütüphane bulunmaktadır. Oysa karakter LCD'mizde oldukça az fonksiyon vardı. Görüldüğü üzere fonksiyonlarımız içine dikdörtgen, yuvarlak, çizgi çizme gibi ekstra fonksiyonları da eklediğimizde **ks0108.c** dosyamız aşağıdaki gibi olacaktır.

```

#include <pic.h>
#include "ks0108.h"
#include "delay.h"

#define EKRANAC 0x3F
#define EKRANKAPA 0x3E

#define sag 1
#define sol 0

void ks0108_init(void)
{
    RST=1;
    EN=0;
    CS1=0;
    CS2=0;

    DI=0; //Komut verilecek

    ks0108_write(sag, 0xC0); //En üst RAM'in ilk RAM olduğu belirleniyor
    ks0108_write(sol, 0xC0);
    ks0108_write(sag, 0x40); //Sütunun en başına gidiliyor, Y=0
    ks0108_write(sol, 0x40);
    ks0108_write(sag, 0xB8); //Satırın en başına gidiliyor, X=0
    ks0108_write(sol, 0xB8);

    ks0108_write(sag, EKRANAC); //Ekranı aç
    ks0108_write(sol, EKRANAC);

    ks0108_fill(0);
}

void ks0108_write(unsigned char chip, unsigned char veri)

```

```

{
    if(chip)
    {
        CS1=0;
        CS2=1;
    }
    else
    {
        CS1=1;
        CS2=0;
    }

    RW=0;           // Veri yazma moduna alındı
    GLCD_PORT=veri;
    DelayUs(10);
    EN=1;          // Düşen kenar tetiklemeli
    DelayUs(10);
    EN=0;

    CS1=0;          // Chip seçim alanları temizleniyor
    CS2=0;
}

unsigned char ks0108_read(unsigned char chip)
{
    unsigned char veri;
    TRIS_GLCD=0xFF;
    RW=1;
    if(chip)
    {
        CS1=0;
        CS2=1;
    }
    else
    {
        CS1=1;
        CS2=0;
    }
    DelayUs(10);
    EN=1;
    DelayUs(10);
    veri=GLCD_PORT;
    EN=0;
    TRIS_GLCD=0x00;

    CS1=0;
    CS2=0;
    return veri;
}

void ks0108_pixel(unsigned char x, unsigned char y, unsigned char renk)
{
    unsigned char veri, kay=1;
    unsigned char chip=sol;

    if(x>63)           // 63'ten büyükse ikinci chipe geç
    {
        x=x-64;
        chip=sag;
    }
}

```

```

DI=0;                                // Komut verilecek
x=x&0x7F;                            // X için komutlar hazırlanıyor
x=x|0x40;
ks0108_write(chip,x);                // X için komut veriliyor
ks0108_write(chip,(y/8 & 0xBF) | 0xB8); // Y için komut veriliyor
DI=1;                                // Veri okunup, yazılacak
ks0108_read(chip);                  // Yeni adres bilgisi için
veri=ks0108_read(chip);              // GLCD iki kez okunuyor

if(renk)                           // Pixel açık mı kapalı mı
    veri=veri | kay<<y%8;
else
    veri=veri & (~(kay<<y%8));

DI=0;                                // Komut veriliyor
ks0108_write(chip,x);                // Yazım yapılıyor
DI=1;
ks0108_write(chip,veri);
}

void ks0108_fill(unsigned char renk)
{
    unsigned char i,j;

    for(i=0;i<8;i++)                  // 8 sayfa var
    {
        DI=0;                          // Komut veriliyor
        ks0108_write(sol,0x40);
        ks0108_write(sag,0x40);
        // Her seferinde, her sayfanın başına gidiliyor
        ks0108_write(sol,i | 0xB8);
        ks0108_write(sag,i | 0xB8);
        DI=1;                          // Veri gönderilecek
        for(j=0;j<64;j++)            // 64 sütun var
        {
            if(renk)
            {
                ks0108_write(sag, 0xFF);
                ks0108_write(sol, 0xFF);
            }
            else
            {
                ks0108_write(sag, 0x00);
                ks0108_write(sol, 0x00);
            }
        }
    }
}

void ks0108_line(unsigned char x1,unsigned char y1,unsigned char
x2,unsigned char y2, unsigned char renk)
{
    int addx=1, addy=1, P;
    unsigned char i, dy, dx, diff;

    if(x2>x1)
        dx = x2 - x1;
    else
    {
        dx = x1 - x2;
        addx = -1;
}

```

```

    }
    if(y2>y1)
        dy = y2 - y1;
    else
    {
        dy = y1 - y2;
        addy = -1;
    }

    if(dx >= dy)
    {
        dy *= 2;
        P = dy - dx;
        diff = P - dx;

        for(i=0; i<=dx; ++i)
        {
            ks0108_pixel(x1, y1, renk);

            if(P < 0)
            {
                P += dy;
                x1 += addx;
            }
            else
            {
                P += diff;
                x1 += addx;
                y1 += addy;
            }
        }
    }
    else
    {
        dx *= 2;
        P = dx - dy;
        diff = P - dy;

        for(i=0; i<=dy; ++i)
        {
            ks0108_pixel(x1, y1, renk);

            if(P < 0)
            {
                P += dx;
                y1 += addy;
            }
            else
            {
                P += diff;
                x1 += addx;
                y1 += addy;
            }
        }
    }
}

void ks0108_rect(unsigned char x1, unsigned char y1, unsigned char x2,
unsigned char y2, unsigned char dolu, unsigned char renk)
{
    if(dolu)

```

```

{
    unsigned char i, xmin, xmax, ymin, ymax;

    if(x1 < x2)
    {
        xmin = x1;
        xmax = x2;
    }
    else
    {
        xmin = x2;
        xmax = x1;
    }

    if(y1 < y2)
    {
        ymin = y1;
        ymax = y2;
    }
    else
    {
        ymin = y2;
        ymax = y1;
    }

    for(; xmin <= xmax; ++xmin)
    {
        for(i=ymin; i<=ymax; ++i)
        {
            ks0108_pixel(xmin, i, renk);
        }
    }
}
else
{
    ks0108_line(x1, y1, x2, y1, renk);
    ks0108_line(x1, y2, x2, y2, renk);
    ks0108_line(x1, y1, x1, y2, renk);
    ks0108_line(x2, y1, x2, y2, renk);
}
}

void ks0108_circle(unsigned char x, unsigned char y, unsigned char r,
unsigned char dolu, unsigned char renk)
{
    unsigned char a,b;
    int P;
    a=0;
    b=r;
    P=1-r;

    do
    {
        if(dolu)
        {
            ks0108_line(x-a, y+b, x+a, y+b, renk);
            ks0108_line(x-a, y-b, x+a, y-b, renk);
            ks0108_line(x-b, y+a, x+b, y+a, renk);
            ks0108_line(x-b, y-a, x+b, y-a, renk);
        }
    }
}

```

```

    {
        ks0108_pixel(a+x, b+y, renk);
        ks0108_pixel(b+x, a+y, renk);
        ks0108_pixel(x-a, b+y, renk);
        ks0108_pixel(x-b, a+y, renk);
        ks0108_pixel(b+x, y-a, renk);
        ks0108_pixel(a+x, y-b, renk);
        ks0108_pixel(x-a, y-b, renk);
        ks0108_pixel(x-b, y-a, renk);
    }

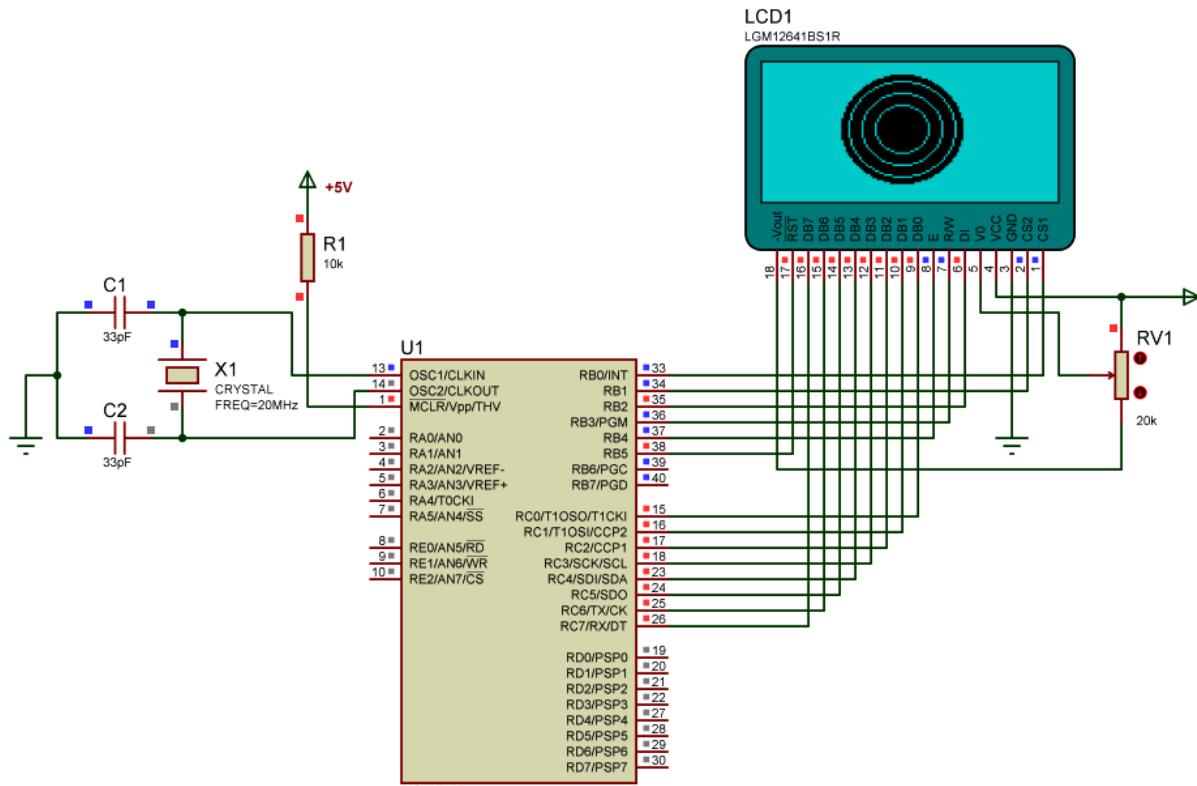
    if(P < 0)
        P+= 3 + 2 * a++;
    else
        P+= 5 + 2 * (a++ - b--);
    }while(a <= b);
}

```

Göründüğü üzere neredeyse 7 sayfa tutan bir kütüphane dosyasına sahibiz. Bu da kütüphanemizin, pic içerisinde oldukça fazla yer kaplayacağı anlamına gelmektedir. Dikdörtgen, yuvarlak, çizgi çizme gibi fonksiyonların algoritmalarını internetten bulup sizde yukarıdaki gibi dilediğiniz şekilde kullanabilirsiniz.

13.2) GLCD Uygulaması

Bu uygulamada oluşturduğumuz GLCD dosyasını kullanarak GLCD'nin istenilen bölgesine dikdörtgen, daire, çizgi çizdirelim. Bunun için öncelikle şekil-111'deki devremizi çizelim.



Şekil 111 – GLCD Uygulaması

İsteğimizi yerine getiren kodlar ise aşağıdaki gibidir.

```
#include <htc.h>
#include "delay.h"
#include "ks0108.h"

void main(void)
{
    unsigned char x1,x2,y1,y2,i;
    PORTB=0x00;
    PORTC=0x00;
    TRISB=0x00; // GLCD için çıkışlar
    TRISC=0x00; // tanımlanıyor

    ks0108_init(); // İlk ayarlar yapılıyor

    while(1)
    {
        x1=5;
        y1=5;
        x2=100;
        y2=5;
        for(i=0;i<6;i++)
        {
            // GLCD'de Çizgi çizdiriliyor
            ks0108_line(x1, y1, x2, y2, 1);
            y2=y1+=10;
            x2-=10;
            DelayMs(250);DelayMs(250);DelayMs(250);DelayMs(250);
        }

        ks0108_fill(0); // Ekran temizleniyor
        x1=y1=0;
        for (i=0;i<120;i++)
        {
            // Ekranda istenilen pikseller aktif yapılıyor
            ks0108_pixel(x1,y1,1);
            y1=x1++;
            DelayMs(50);
        }

        ks0108_fill(0); // Ekran temizleniyor
        for(i=30;i>0;i=i-3)
        {
            // GLCD'de Daire çizdiriliyor
            ks0108_circle(60, 30, i, i%2, i%2);
            DelayMs(250);DelayMs(250);DelayMs(250);DelayMs(250);
        }

        ks0108_fill(0); // Ekran temizleniyor
        x1=5;
        y1=5;
        x2=120;
        y2=63;
        for(i=0;i<6;i++)
        {
            // GLCD'de dikdörtgen çizdiriliyor
            ks0108_rect(x1, y1, x2, y2, 0, 1);
            y1=x1+=5;
            x2-=5;
            y2-=5;
        }
    }
}
```

```
        DelayMs (250);DelayMs (250);
    }
    ks0108_fill(0);           // Ekran temizleniyor
}
}
```

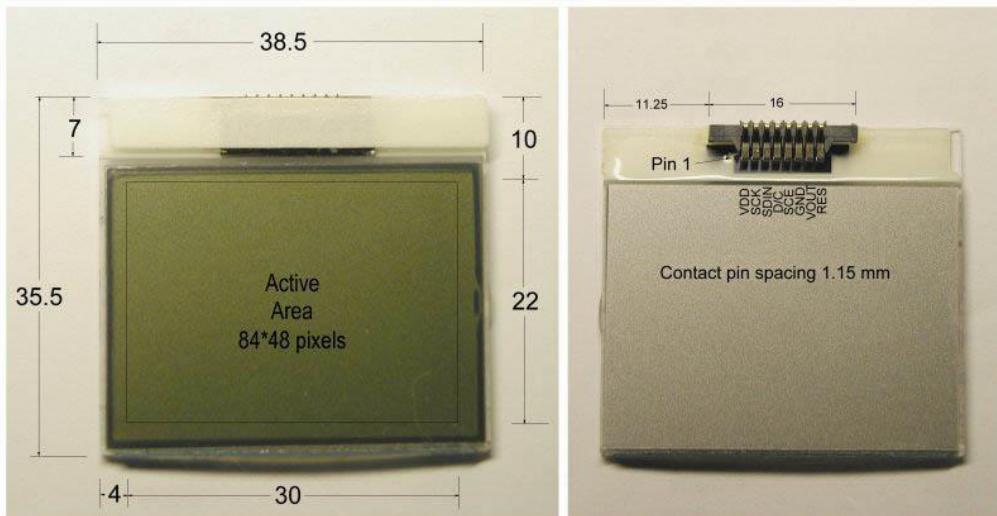
Görüldüğü üzere GLCD sürmek normal karakter LCD sürmekten oldukça zordur. Dolayısı ile birçok işlemede karakter LCD kullanılmaktadır.

Simülasyon yaparken dikkat edilecek durum pic'in 20MHz'e ayarlanmış olmasıdır. Gerçek hayatta bu değer 4MHz'dir. 20MHz yapılmasının nedeni proteusta bulunan GLCD'nin şekilleri 4MHz'de bozuk göstermesindendir.

BÖLÜM 14 – NOKIA GRAFİK LCD

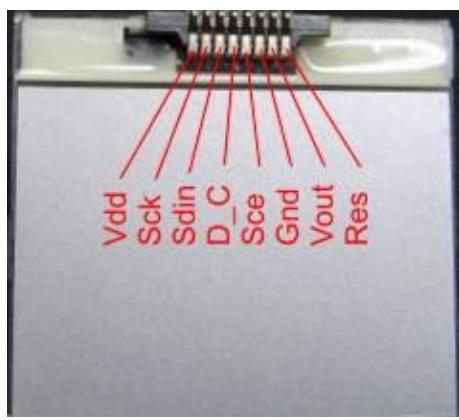
14.1) Nokia 3310 Ekranı

Şekil-112'de görülebilecek Nokia 3310 ekranı aslında Philips firmasının PCD8544 tabanlı 48x84 pixellik bir grafik LCD'dir.



Şekil 112 – Nokia 3310 Ekranı

Diğer GLCD sürümlerinden tek farkı bilgilerin paralel değil seri şekilde işlenmesidir. Seri iletişim protokolü kullanması yaşanabilecek zaman problemlerinin giderilmesini sağlamıştır.

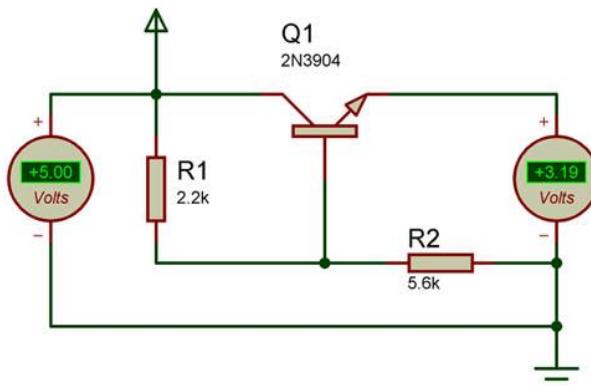


Şekil 113 – Nokia 3310 Ekranının Pinleri

Şekil-113'te Nokia3310 ekranının pinleri görülmektedir, bu pinlerin görevleri şöyledir.

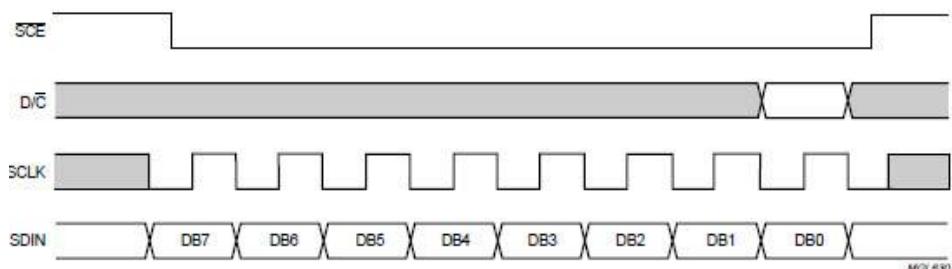
- VDD : +2,7V'tan +3.3V değerleri arasında voltaj uygulanmalıdır. Yüksek voltaj değerleri LCD'nin kolaylıkla yanmasına neden olabilir.

Voltaj devresi için aşağıdaki devre kullanılabilir;



- SCLK : SCE pini sıfırken, seri iletişim saat darbeleri buradan alınır.
- SDIN : Seri data giriş ucudur, datalar SCLK'nın yükselen kenarlarında işlenir.
- D/C' : Kontrol pinidir, bilgi data mı yoksa komut mu onu anlamaya yarar. Alınan bilgi bu pin '0' iken komut, '1' iken veridir.
- SCE : Seri iletişim aktif etme ucudur, herhangi bir bilgi gönderilmek istendiğinde '0' yapılmalıdır.
- GND : Toprak ucudur.
- Vout : Kontrast ucudur, toprak ucuna 1-10 mikro farad arasında kondansöterle bağlanmalı.
- RES : Reset ucudur, başlangıçta o yapılmalıdır, yoksa LCD zarar görebilir.

Nokia 3310 ekranına veri gönderip alma işlemlerinin yapılışı ise şekil-114'teki zaman diyagramından görülebilir.



Şekil 114 – Nokia 3310 Ekranının Zaman Diyagramı

Nokia 3310 ekranı ile iletişim kurulması için şu sıra izlenmelidir;

- Öncelikle SCE pini '0' yapılmalı.
- D/C' pininden gönderilecek bilginin komut ya da data mı olduğu seçilmeli (data için '1', komut için '0' olmalı)
- Daha sonra SCLK'nın her yükselen kenarında 1 bit bilgi gönderilmelidir.

Nokia ekranını çalıştırmak için bazı özel komutlar vardır. Bu komutlar şekil-115'te görülebilir. GLCD ve komut görevleri ise şöyledir;

- LCD'nin hafıza yapısı 6x8x84 bit şeklindedir. (48x84)
- X ekseninde 0'dan 83'e kadar adresleme yapılabilirken, Y ekseninde 0'dan 5'e kadar adresleme yapılabilir. (Kaynak: Datasheet)

Komutlar:

Table 1 Instruction set

INSTRUCTION	D/C	COMMAND BYTE								DESCRIPTION
		DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
(H = 0 or 1)										
NOP	0	0	0	0	0	0	0	0	0	no operation
Function set	0	0	0	1	0	0	PD	V	H	power down control; entry mode; extended instruction set control (H)
Write data	1	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	writes data to display RAM
(H = 0)										
Reserved	0	0	0	0	0	0	1	X	X	do not use
Display control	0	0	0	0	0	1	D	0	E	sets display configuration
Reserved	0	0	0	0	1	X	X	X	X	do not use
Set Y address of RAM	0	0	1	0	0	0	Y ₂	Y ₁	Y ₀	sets Y-address of RAM; 0 ≤ Y ≤ 5
Set X address of RAM	0	1	X ₆	X ₅	X ₄	X ₃	X ₂	X ₁	X ₀	sets X-address part of RAM; 0 ≤ X ≤ 83
(H = 1)										
Reserved	0	0	0	0	0	0	0	0	1	do not use
	0	0	0	0	0	0	0	1	X	do not use
Temperature control	0	0	0	0	0	0	1	TC ₁	TC ₀	set Temperature Coefficient (TC _X)
Reserved	0	0	0	0	0	1	X	X	X	do not use
Bias system	0	0	0	0	1	0	BS ₂	BS ₁	BS ₀	set Bias System (BS _X)
Reserved	0	0	1	X	X	X	X	X	X	do not use
Set V _{OP}	0	1	V _{OP6}	V _{OP5}	V _{OP4}	V _{OP3}	V _{OP2}	V _{OP1}	V _{OP0}	write V _{OP} to register

Table 2 Explanations of symbols in Table 1

BIT	0	1
PD	chip is active	chip is in Power-down mode
V	horizontal addressing	vertical addressing
H	use basic instruction set	use extended instruction set
D and E		
00	display blank	
10	normal mode	
01	all display segments on	
11	inverse video mode	
TC ₁ and TC ₀		
00	V _{LCD} temperature coefficient 0	
01	V _{LCD} temperature coefficient 1	
10	V _{LCD} temperature coefficient 2	
11	V _{LCD} temperature coefficient 3	

Şekil 115 – Nokia 3310 Ekranının Komutları

İlk yükleme değerleri için ise aşağıdaki işlemler yapılmalıdır;

- 1) İlk önce LCD pasif yapılmalı. Bunun için SCE=1 olmalı.
- 2) Daha sonra LCD'ye zarar vermemek için 10ms kadar RES=0 olmalı ve daha sonra RES=1 yapılmalı.
- 3) Sonra sırayla aşağıdaki işlemler uygulanır
 - 0x21: Tanımlanmış komutlara erişim açılıyor
 - 0xC8: Kontrast voltajı ayarlanıyor.(Direk VSS'e bağlı)
 - 0x13: LCD bias ayarlanıyor (1:48 modunda)
 - 0x20: Yatay mod, X otomatik artacak, komutlara erişilmiyor.
 - 0x09: LCD açılıyor
 - DDRAM temizleniyor, böylece önceden kaydedilmiş kayıtlar siliniyor.
 - 0x08: Tüm LCD segmentleri açılıyor.
 - 0x0C: LCD normal moda çalıştırılıyor.

Tüm bunları hesaba katıp kütüphane haline getirirsek 3310.h dosyası aşağıdaki gibi olacaktır.

```
/*
 * www.FxDev.org
 * 3310 LCD Kullanım Klavuzu
 * nokia_init(); ile LCD'nin ilk ayarlarını yap.
 * nokia_clean_ddram(); ile LCD nin DDRAM'i silinir.
 * nokia_write_command(0x20); şeklinde komut gönderilir.
 * nokia_write_data(0x01); şeklinde data gönderilir.
 * nokia_write_byte(0xFE); ile bir byte'luk bilgi gönderilir.
 * nokia_gotoxy(2,3); ile lcd'de belirli bölgelere gidilebilir. Sınır:
(0,0) dan (83,5) tir.
 * nokia_contrast(0x0F); ile kontrast ayarı yapılır.
 * nokia_printchar("Deneme"); şeklinde yazı yazdırılır.
 * nokia_print(i+48); şeklinde sayı yazdırılır.
 * lcdpixel (3,4); şeklinde pixel belirtilir.
 *
 * www.FxDev.org
 */

#define nok_res RB0
#define nok_sce RB1
#define nok_dc RB2
#define nok_sdin RB3
#define nok_sclk RB4

extern void nokia_init(void);
extern void nokia_clean_ddram(void);
extern void nokia_write_command(char nokia_command);
extern void nokia_write_data(char nokia_data);
extern void nokia_write_byte(char bytefornokia);
extern void nokia_gotoxy(char ynokia, char xnokia);
extern void nokia_contrast(char contrast);
extern void nokia_printchar(const char *s);
extern void nokia_print(char charsel);
extern void lcdpixel (char x, char y);
```

Yukarıdaki işlemleri yapan 3310.c dosyası ise aşağıdaki gibi olacaktır.

```
#include <pic.h>
#include "delay.h"
#include "3310.h"

const unsigned char tablo1 [240] = {
    0x00,0x00,0x00,0x00,0x00, // 20
bosluk
    0x00,0x00,0x5f,0x00,0x00, // 21 !
    0x00,0x07,0x00,0x07,0x00, // 22 "
    0x14,0x7f,0x14,0x7f,0x14, // 23 #
    0x24,0x2a,0x7f,0x2a,0x12, // 24 $
    0x23,0x13,0x08,0x64,0x62, // 25 %
    0x36,0x49,0x55,0x22,0x50, // 26 &
    0x00,0x05,0x03,0x00,0x00, // 27 '
    0x00,0x1c,0x22,0x41,0x00, // 28 (
    0x00,0x41,0x22,0x1c,0x00, // 29 )
    0x14,0x08,0x3e,0x08,0x14, // 2a *
    0x08,0x08,0x3e,0x08,0x08, // 2b +
    0x00,0x50,0x30,0x00,0x00, // 2c ,
    0x08,0x08,0x08,0x08,0x08, // 2d -
    0x00,0x60,0x60,0x00,0x00, // 2e .
    0x20,0x10,0x08,0x04,0x02, // 2f /
    0x3e,0x51,0x49,0x45,0x3e, // 30 0
```

```

        0x00,0x42,0x7f,0x40,0x00,    // 31 1
        0x42,0x61,0x51,0x49,0x46,    // 32 2
        0x21,0x41,0x45,0x4b,0x31,    // 33 3
        0x18,0x14,0x12,0x7f,0x10,    // 34 4
        0x27,0x45,0x45,0x45,0x39,    // 35 5
        0x3c,0x4a,0x49,0x49,0x30,    // 36 6
        0x01,0x71,0x09,0x05,0x03,    // 37 7
        0x36,0x49,0x49,0x49,0x36,    // 38 8
        0x06,0x49,0x49,0x29,0x1e,    // 39 9
        0x00,0x36,0x36,0x00,0x00,    // 3a :
        0x00,0x56,0x36,0x00,0x00,    // 3b ;
        0x08,0x14,0x22,0x41,0x00,    // 3c <
        0x14,0x14,0x14,0x14,0x14,    // 3d =
        0x00,0x41,0x22,0x14,0x08,    // 3e >
        0x02,0x01,0x51,0x09,0x06,    // 3f ?
        0x32,0x49,0x79,0x41,0x3e,    // 40 @
        0x7e,0x11,0x11,0x11,0x7e,    // 41 A
        0x7f,0x49,0x49,0x49,0x36,    // 42 B
        0x3e,0x41,0x41,0x41,0x22,    // 43 C
        0x7f,0x41,0x41,0x22,0x1c,    // 44 D
        0x7f,0x49,0x49,0x49,0x41,    // 45 E
        0x7f,0x09,0x09,0x09,0x01,    // 46 F
        0x3e,0x41,0x49,0x49,0x7a,    // 47 G
        0x7f,0x08,0x08,0x08,0x7f,    // 48 H
        0x00,0x41,0x7f,0x41,0x00,    // 49 I
        0x20,0x40,0x41,0x3f,0x01,    // 4a J
        0x7f,0x08,0x14,0x22,0x41,    // 4b K
        0x7f,0x40,0x40,0x40,0x40,    // 4c L
        0x7f,0x02,0x0c,0x02,0x7f,    // 4d M
        0x7f,0x04,0x08,0x10,0x7f,    // 4e N
        0x3e,0x41,0x41,0x41,0x3e}; // 4f O

const unsigned char tablo2 [240] = {
    0x7f,0x09,0x09,0x09,0x06,    // 50 P
    0x3e,0x41,0x51,0x21,0x5e,    // 51 Q
    0x7f,0x09,0x19,0x29,0x46,    // 52 R
    0x46,0x49,0x49,0x49,0x31,    // 53 S
    0x01,0x01,0x7f,0x01,0x01,    // 54 T
    0x3f,0x40,0x40,0x40,0x3f,    // 55 U
    0x1f,0x20,0x40,0x20,0x1f,    // 56 V
    0x3f,0x40,0x38,0x40,0x3f,    // 57 W
    0x63,0x14,0x08,0x14,0x63,    // 58 X
    0x07,0x08,0x70,0x08,0x07,    // 59 Y
    0x61,0x51,0x49,0x45,0x43,    // 5a Z
    0x00,0x7f,0x41,0x41,0x00,    // 5b [
    0x02,0x04,0x08,0x10,0x20,    // 5c \
    0x00,0x41,0x41,0x7f,0x00,    // 5d ñ
    0x04,0x02,0x01,0x02,0x04,    // 5e ^
    0x40,0x40,0x40,0x40,0x40,    // 5f -
    0x00,0x01,0x02,0x04,0x00,    // 60 -
    0x20,0x54,0x54,0x54,0x78,    // 61 a
    0x7f,0x48,0x44,0x44,0x38,    // 62 b
    0x38,0x44,0x44,0x44,0x20,    // 63 c
    0x38,0x44,0x44,0x48,0x7f,    // 64 d
    0x38,0x54,0x54,0x54,0x18,    // 65 e
    0x08,0x7e,0x09,0x01,0x02,    // 66 f
    0x0c,0x52,0x52,0x52,0x3e,    // 67 g
    0x7f,0x08,0x04,0x04,0x78,    // 68 h
    0x00,0x44,0x7d,0x40,0x00,    // 69 i
    0x20,0x40,0x44,0x3d,0x00,    // 6a j
    0x7f,0x10,0x28,0x44,0x00,    // 6b k
}

```

```

0x00,0x41,0x7f,0x40,0x00,    // 6c l
0x7c,0x04,0x18,0x04,0x78,    // 6d m
0x7c,0x08,0x04,0x04,0x78,    // 6e n
0x38,0x44,0x44,0x44,0x38,    // 6f o
0x7c,0x14,0x14,0x14,0x08,    // 70 p
0x08,0x14,0x14,0x18,0x7c,    // 71 q
0x7c,0x08,0x04,0x04,0x08,    // 72 r
0x48,0x54,0x54,0x54,0x20,    // 73 s
0x04,0x3f,0x44,0x40,0x20,    // 74 t
0x3c,0x40,0x40,0x20,0x7c,    // 75 u
0x1c,0x20,0x40,0x20,0x1c,    // 76 v
0x3c,0x40,0x30,0x40,0x3c,    // 77 w
0x44,0x28,0x10,0x28,0x44,    // 78 x
0x0c,0x50,0x50,0x50,0x3c,    // 79 y
0x44,0x64,0x54,0x4c,0x44,    // 7a z
0x00,0x08,0x36,0x41,0x00,    // 7b {
0x00,0x00,0x7f,0x00,0x00,    // 7c |
0x00,0x41,0x36,0x08,0x00,    // 7d }
0x10,0x08,0x08,0x10,0x08,    // 7e ~
0x78,0x46,0x41,0x46,0x78};   // 7f !
}

void nokia_init(void)
{
    nok_dc=1;
// Bilgiler DDRAM'de tutuluyor, Address counter ise otomatik artacak
    nok_sce=1;                  // Chip pasif
    DelayUs(200);

    nok_res=0;
// Başlangıçta res=0 olmak zoruna, aksi halde LCD'ye zarar gelebilir
    DelayMs(10);
    nok_res=1;

    nokia_write_command(0x21);   // LCD'ye komut yazılıyor.
    nokia_write_command(0x06);   // Vop V: 0xC8 (for 3V)
    nokia_write_command(0x13);   // LCD (1:48) modunda (bias)
    nokia_contrast(0x40);       // Kontrast ayarı yapılıyor
    nokia_write_command(0x20);
// Soldan sağa x birer artırılarak yazılıyor
// 0x22 kullanılsaydı, yukarıdan aşağı yazıcaktı.

    nokia_clean_ddram();
// DDRAM temizleniyor, aksi halde eski yazılmış karakterlerle
karıştırılabilir
    DelayMs(10);

    nokia_write_command(0x0C);   // Display normal moda dönüyor
}

void nokia_clean_ddram(void)
{
    int ddram;
    nokia_gotoxy(0,0); // 84*6=504bit Dram alanı temizleniyor
    for(ddram=0; 504>ddram; ddram++)
        nokia_write_data(0x00);
}

void nokia_write_command(char nokia_command)
{
    nok_dc      = 0; // Komut gönderimi için D/C=0;
    nok_sce = 0;     // Chip aktif
}

```

```

        nokia_write_byte(nokia_command);
        nok_sce = 1;           // Chip pasif
    }

void nokia_write_data(char nokia_data)
{
    nok_dc      = 1;   // Data gönderimi için D/C=1
    nok_sce = 0;       // Chip aktif
    nokia_write_byte(nokia_data);
    nok_sce = 1;       // Chip pasif
}

void nokia_write_byte(char bytefornokia)
// Seri veri gönderme rutini, datasheet'ten bire bir alıntı.
{
    char i;
    for(i=8;i>0;i--)
    {
        nok_sclk=0;
        DelayUs(2);
        if((bytefornokia & 0x80)==0)
            nok_sdin=0;
        else
            nok_sdin=1;
        DelayUs(2);
        nok_sclk=1;
        DelayUs(2);
        bytefornokia = bytefornokia << 1;
    }
}

void nokia_gotoxy(char xnokia,char ynokia)
// İstenilen bölgeye ulaşma fonksiyonu
{
    nokia_write_command(0x40|(ynokia & 0x07));
    // Y bu şekilde hareket ediyor: 0100 0yyy
    nokia_write_command(0x80|(xnokia & 0x7F));
    // X bu şekilde hareket ediyor: 1xxx xxxx
}

void nokia_contrast(char contrast)
{
    nokia_write_command(0x21);                      // LCD'ye komut veriliyor
    nokia_write_command(0x80 | contrast);           // Kontrast ayarı
    // Yatay adresleme modu, X birer otomatik artar.
    nokia_write_command(0x20);
}

void nokia_printchar(const char *s)
{
    while(*s)
        nokia_print(*s++);
}

// ASCII tablosunu kullanarak yazı yazdırma yarayan fonksiyon
void nokia_print(char charsel)
{
    char char_row,charpos,chardata;

    if (charsel<0x20) return;
    if (charsel>0x7f) return;
}

```

```

for(char_row=0;char_row<5;char_row++)
{
    // Seçilen karakterin ASCII kodu 50h'tan ufaksa
    if(charsel<0x50)
    {
        charpos=((charsel&0xff)-0x20)*5);
        chardata=tablo1[(charpos+char_row)];
    }
    // Seçilen karakterin ASCII kodu 50h'tan büyükse
    if(charsel>0x4f)
    {
        charpos=((charsel&0xff)-0x50)*5);
        chardata=tablo2[(charpos+char_row)];
    }

    nokia_write_data(chardata); // Datayı gönder
}
nokia_write_data(0x00); // Her zaman 1 bytelik son alan boş olacak
}

void lcdpixel (char x, char y)
{
    char offset, data;

    if (x > 84) return;
    if (y > 48) return;

    offset = y - ((y / 8) * 8);
    data = (0x01 << offset);

    nokia_gotoxy(x, (y/6));
    nokia_write_data(data);
}

```

Kütüphaneden görüleceği üzere Nokia 3310 ekranını da diğer GLCD'ler gibi kullanmak zordur.

14.2) Nokia 3310 Ekranı Uygulaması

Proteus'un kendisinde Nokia 3310 ekranı yoktur. Yalnız internette bu kütüphane bulunup proteus'a kolaylıkla eklenebilir. Bu uygulamada Nokia 3310 ekranına istediğimiz bir şekili aktaralım.

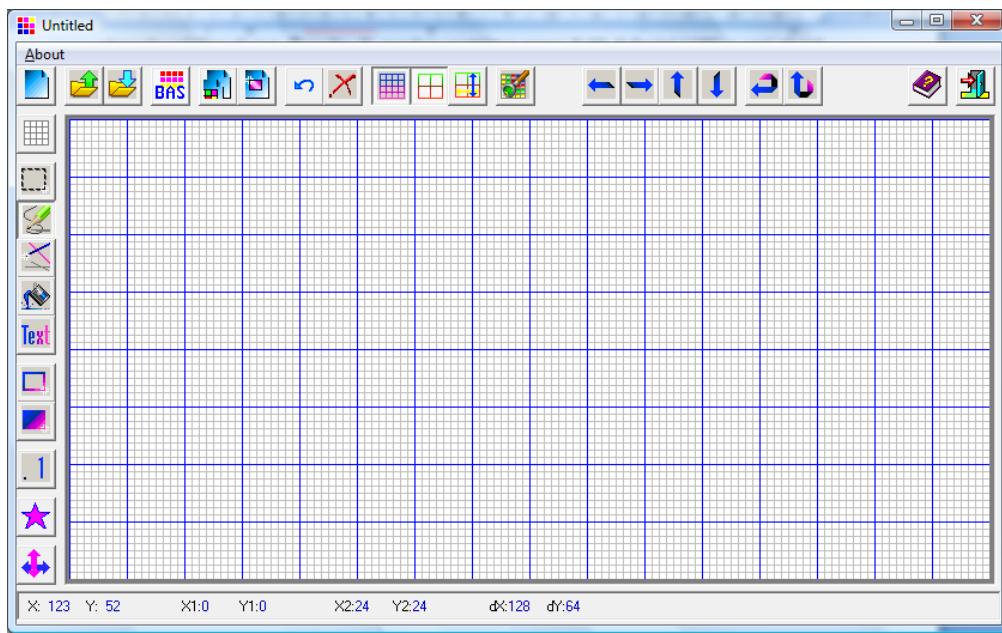
Nokia 3310 ekranının 6x8x84bit büyüklüğünde DDRAM'ı istediğimiz verilerle doldurursak, doluluk oranına göre ekranда şekiller oluşur. Örneğin ilk ayarları yaptıktan sonra 0xFF değerini LCD'ye veri olarak gönderdiğimizde ekranın 0x8 lik alanında düz bir çizgi görürüz. Aynı şekilde 0x9F gönderdiğimizde de bir pixel genişliğinde İ harfi görürüz.

LCD'de yazı yazdırma için önceden ASCII dizisi tanımlayıp, istediğimiz şekilde harfleri çağırıp kolaylıkla yazdırabiliriz.

GLCD'de herhangi bir şekilde resim çıkarmak için yine aynı şekilde diziler oluşturulmalı ve bu dizinin boyutu $84 \times 6 = 504$ bit boyutunu geçmemelidir. Bu tanımlamayı yapmak teker teker elle çok zor olacağı için FastLCD gibi programları kullanmak işinizi daha da kolaylaştıracaktır.

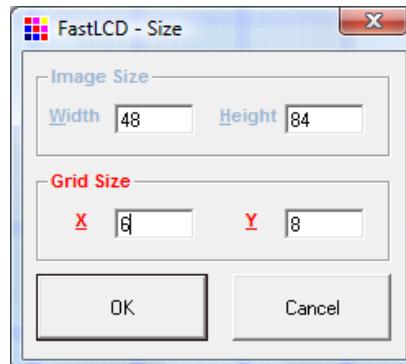
14.2.1) FastLCD Kullanımı

Programın ara yüzü şekil-116'daki gibidir. Öncelikle boyut ayarından genişlik 84, yükseklik ise 48 yapılmalıdır.



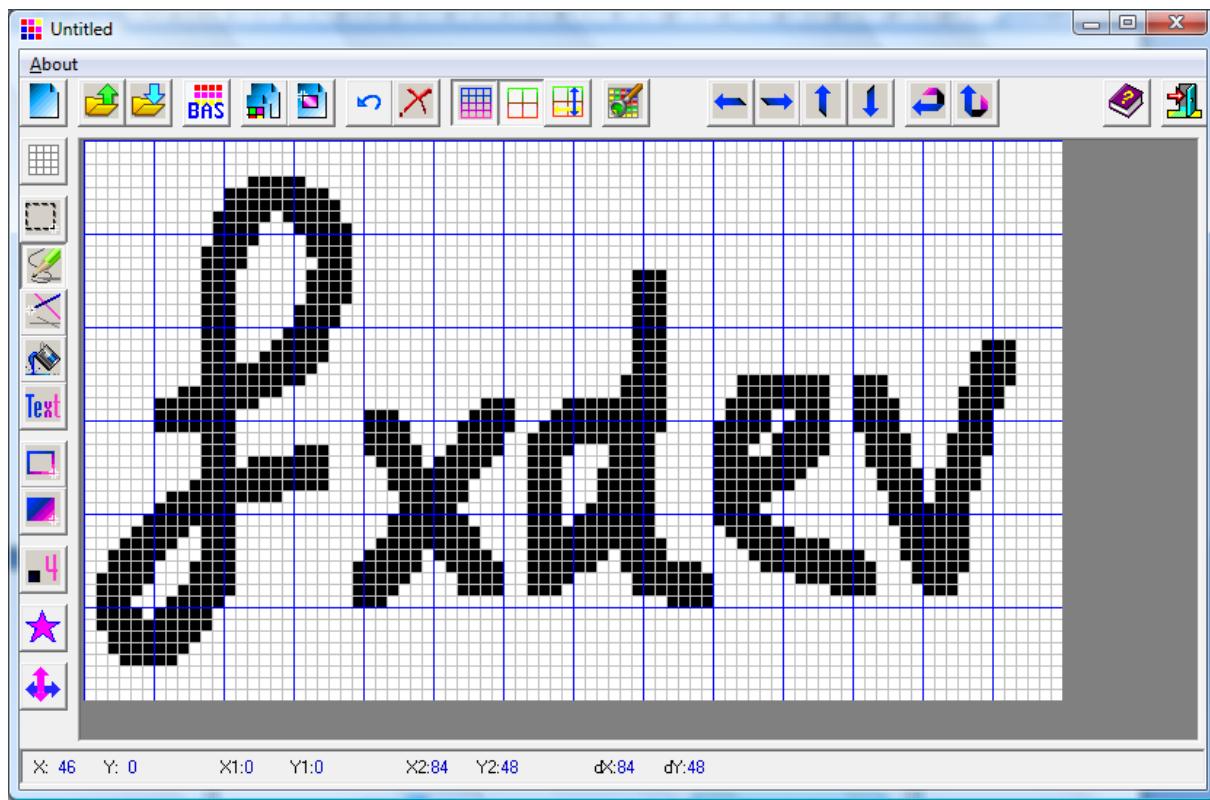
Şekil 116 – Fast LCD Programı Görüntüsü

Memory yapısı $6 \times 8 \times 84$ olduğu için karakterlerimizin boyutunu şekil-117'deki gibi 6×8 'lik seçmek uygun olacaktır.



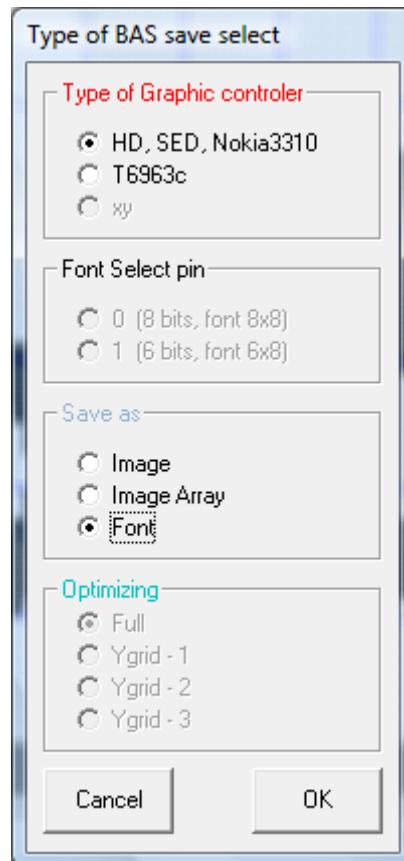
Şekil 117 – Fast LCD'de Boyut Ayarı

Ayarlar yapıldıktan sonra istenilen resmi şekil-118'deki gibi çizebiliriz.



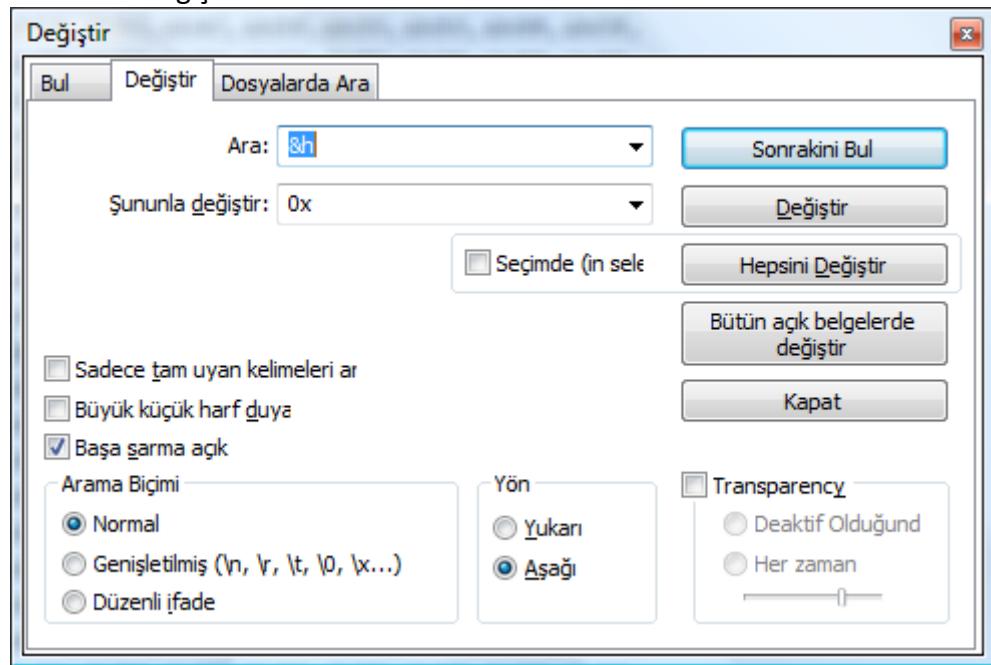
Şekil 118 – Fast LCD Programında Resim Çizimi

Daha sonra bunu şekil-118'deki menüde yazan BAS seçeneğine basıp, Nokia 3310'u seçip, özellikle save as kısmından şekil-119'da görülen image array seçilerek kaydedelim.



Şekil 119 – Kaydetme Seçenekleri

Daha sonra kaydettiğimiz dosya notepad++ gibi bir programla açılır ve şekil-120'deki gibi &h olan kısımlar Ox ile değiştirilir.

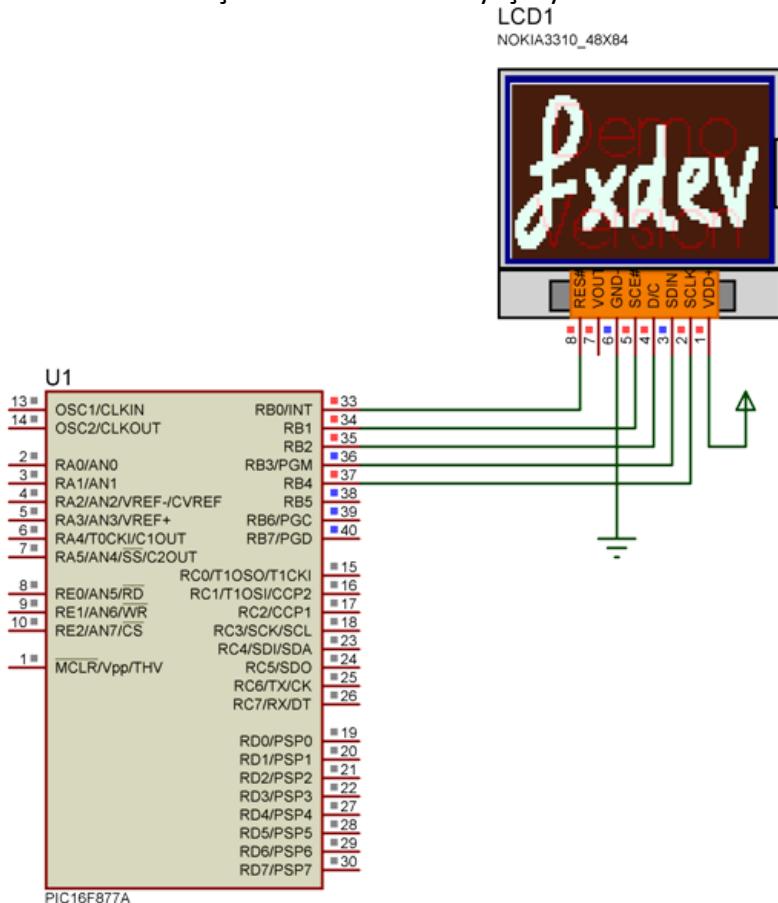


Şekil 120 – Değişecek Değerler

Bu değişiklikler olduktan sonra tanımlanan dizi örneği ise aşağıdaki gibi olacaktır.

14.2.2) Tanımlanan Dizinin Kullanılması

Dizimizi tanımladıktan sonra yapmamız gereken tek şey bu değerleri sırasıyla Nokia 3310 ekranına yazdırmaktır. Öncelikle şekil-121'deki devreyi çiziyoruz.



Şekil 121 – Nokia 3310 Uygulaması

İsteğimizi yerine getiren kodumuz ise aşağıdadır.

```

0x00,0xC0,0xF0,0xF0,0x70,0xF0,0xF0,0x00,0x00,0xF0,0xF0,0x00
,0xC0,0x00,0x00,0x00,0x00,0xE0,0xFC,0x7E,0x1E,0x00,0x00,0x00
,0x00,0x00,0x00,0x00,0x00,0x81,0xC1,0xC1,0xE1,0xFF,0xFF,0x70,0x
78,0x78,0x78,0x38,0x3C,0x3C,0x3C,0x00,0x00,0x00,0x07,0x0F,0x3F,0xFF,0xFC,0
xF0,0xF8,0xFC,0x7F,0x1F,0x0F,0x07,0x01,0x00,0xFE,0xFF,0xFF,0x07,0x03,0x83,
0xF3,0xFF,0xFF,0xFF,0x01,0x00,0x00,0x00,0x00,0xFF,0xFF,0xFB,0x79
,0x3E,0x3F,0x1F,0x0F,0x07,0x00,0x00,0x01,0x0F,0x3F,0xFF,0xFE,0xF0,0x80,0xF
0,0xFE,0xFF,0x3F,0x07,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xE0,0xF8,0xFC,0x
7E,0x3F,0x0F,0x87,0xC3,0xF1,0xFF,0xFF,0x7F,0x00,0x00,0x00,0x00,0x00,0x00,0x00
,0x00,0x00,0x00,0xF0,0xF8,0xFC,0x7F,0x3F,0x1F,0x07,0x0F,0x1F,0x7F,0x7E,
0x78,0x70,0x00,0x00,0x7F,0x7F,0x7F,0x3E,0x1F,0x1F,0x0F,0x07,0x1F,0x7F,0x7F
,0x7C,0xF0,0xF0,0xE0,0x01,0x03,0x07,0x0F,0x1F,0x1E,0x1E,0x3C,0x3C,0x3
C,0x78,0x78,0x78,0x70,0x00,0x01,0x0F,0x3F,0x7F,0x7F,0x7F,0x1F,0x00,0x00,0x
00,0x00,0x00,0x00,0x00,0x00,0x03,0x0F,0x1F,0x1E,0x1F,0x1F,0x0F,0
x07,0x03,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0
0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0
0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x0
0,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00
} ;

void main(void)
{
    int i;
    PORTB=0x00;
    TRISB=0x00; // GLCD için çıkış

    nokia_init();

    for(i=0;504>i;i++)
        nokia_write_data(fxdev[i]);

    for(;;);
}

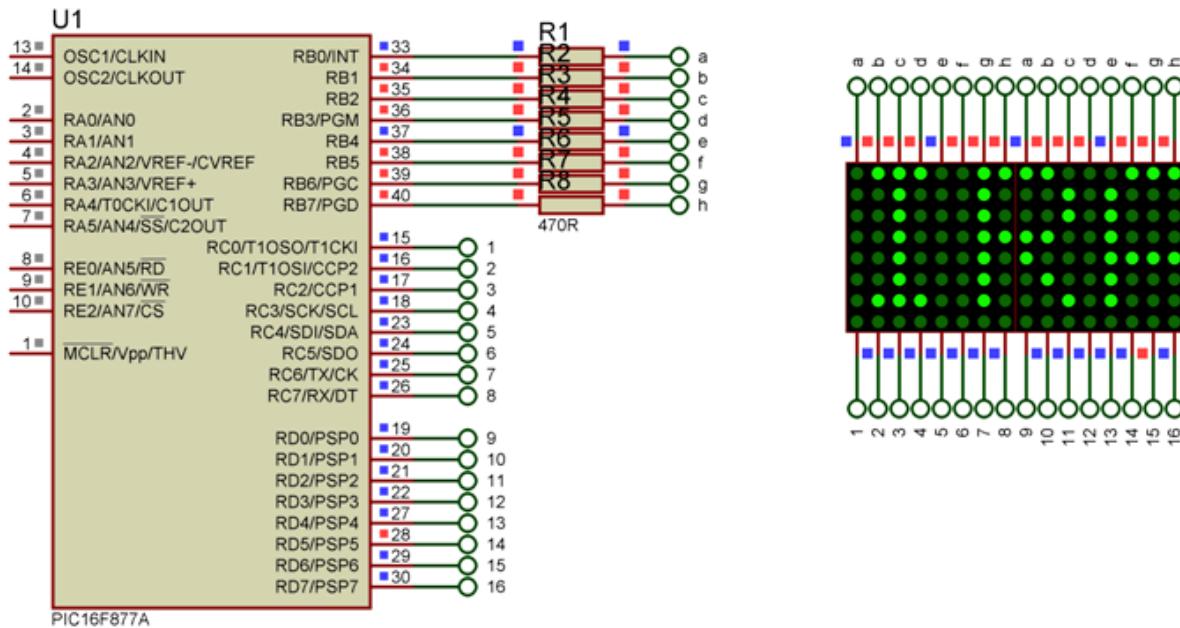
```

Gördüğü üzere kodlarımız oldukça sadedir. Tüm GLCD'lerde olduğu gibi Nokia 3310 ekranı da pic hafızasının önemli bir bölümünü kullanmaktadır.

BÖLÜM 15 – KAYAN YAZI UYGULAMASI

15.1) Kayan Yazı Uygulaması

İlk bölümde dot matrislerin nasıl çalıştığını gördük. Günümüzde özellikle reklam panolarında 5x7 ya da 8x8 matrisler yan yana koyularak kullanılmaktadır. Biz bu uygulamada iki adet 8x8 dot matris ile ismimizi bu sistemde kaydırma işlemi yapacağız. Öncelikle şekil-122'deki devremizi çizelim ve işin mantığına göz atalım.



Şekil 122 – Kayan Yazı Uygulaması

Bölüm 1'de yazı yazdırmanın ne şekilde olacağını görmüştük. Yazı kaydırmak ise, yazı yazdırmanın biraz değiştirilmiş bir halidir. Öncelikle tek bir dizi tanımlanır ve buraya yazdırmak istediğimiz yazının dizi değerleri yerleştirilir. Ben bu uygulamada adımın harflerini dizi içerisinde yerleştirdim. Başta matrislerin boş gözükmesi için 3x6 satır da 0x00 ifadesini dizime yerleştirerek aşağıdaki ifadeyi elde ettim.

```
const unsigned char kayan_yazi[] =  
{  
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 20 bosluk  
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 20 bosluk  
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 20 bosluk  
    0x7f, 0x09, 0x09, 0x09, 0x01, 0x00, // 46 F  
    0x00, 0x41, 0x7f, 0x41, 0x00, 0x00, // 49 I  
    0x7f, 0x09, 0x19, 0x29, 0x46, 0x00, // 52 R  
    0x7e, 0x11, 0x11, 0x11, 0x7e, 0x00, // 41 A  
    0x01, 0x01, 0x7f, 0x01, 0x01, 0x00, // 54 T  
    0x00, 0x00, 0x00, // 20 bosluk  
    0x7f, 0x41, 0x41, 0x22, 0x1c, 0x00, // 44 D  
    0x7f, 0x49, 0x49, 0x49, 0x41, 0x00, // 45 E  
    0x1f, 0x20, 0x40, 0x20, 0x1f, 0x00, // 56 V  
    0x7f, 0x49, 0x49, 0x49, 0x41, 0x00, // 45 E  
    0x3e, 0x41, 0x41, 0x41, 0x22, 0x00, // 43 C  
    0x00, 0x44, 0x7d, 0x44, 0x00, 0x00, // 69 i  
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 20 bosluk  
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 20 bosluk  
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // 20 bosluk  
};
```

Sizde adınızın ifadelerini Nokia 3310 ekranı uygulamasındaki karakter dizilerinden alıp, yeni oluşturacağınız dizinizin içine koyabilirsiniz.

Diziyi oluşturduktan sonra yapılması gereken ise bu ifadelerin dor matrislere basılmasıdır. Burada yine birinci bölümde anlattığımız tarama yöntemini kullanacağız. İşlemimizde tarama sırasıyla PORTC'nin 0. bitinden 7. bitine, oradan PORTD'nin 0. bitinden 7. bitine doğru olacak. Her sütunun bilgi değerini ise PORTB'den göndereceğiz. Bu işlemi 5 kere tekrarladıkten sonra ise karakterlerimizi bir birim sola kaydırarak işlemimizi devam ettireceğiz.

Tüm bu anlatılanların yapıldığı Hi-Tech kodunu ise aşağıda görebilirsiniz.

```
#include <htc.h>
#include "delay.h"
//İsmimiz tanımlanıyor
const unsigned char kayan_yazi[]=
{
    0x00,0x00,0x00,0x00,0x00,0x00,          // 20 bosluk
    0x00,0x00,0x00,0x00,0x00,0x00,          // 20 bosluk
    0x00,0x00,0x00,0x00,0x00,0x00,          // 20 bosluk
    0x7f,0x09,0x09,0x09,0x01,0x00,          // 46 F
    0x00,0x41,0x7f,0x41,0x00,0x00,          // 49 I
    0x7f,0x09,0x19,0x29,0x46,0x00,          // 52 R
    0x7e,0x11,0x11,0x11,0x7e,0x00,          // 41 A
    0x01,0x01,0x7f,0x01,0x01,0x00,          // 54 T
    0x00,0x00,0x00,                          // 20 bosluk
    0x7f,0x41,0x41,0x22,0x1c,0x00,          // 44 D
    0x7f,0x49,0x49,0x49,0x41,0x00,          // 45 E
    0x1f,0x20,0x40,0x20,0x1f,0x00,          // 56 V
    0x7f,0x49,0x49,0x49,0x41,0x00,          // 45 E
    0x3e,0x41,0x41,0x41,0x22,0x00,          // 43 C
    0x00,0x44,0x7d,0x44,0x00,0x00,          // 69 i
    0x00,0x00,0x00,0x00,0x00,0x00,          // 20 bosluk
    0x00,0x00,0x00,0x00,0x00,0x00,          // 20 bosluk
    0x00,0x00,0x00,0x00,0x00,0x00          // 20 bosluk
};

void main(void)
{
    int i,kay=1,j,k; // Değişkenler tanımlanıyor
    PORTB=0x00;
    PORTC=0x00;
    TRISB=0x00;        // Dot matris için çıkışlar
    TRISC=0x00;
    TRISD=0x00;

    for(;;)
    {
        for(i=0;i<8;i++)           // İlk matrise değerler gönderiliyor
        {
            PORTC=kay<<i;
            PORTB=~kayan_yazi[i+j];
            DelayMs(1);
        }
        kay=1;
        PORTC=0x00;
        for(i=0;i<8;i++)           // İkinci matrise değerler gönderiliyor
        {
            PORTD=kay<<i;
            PORTB=~kayan_yazi[i+8+j];
            DelayMs(1);
        }
    }
}
```

```

    kay=1;
    PORTD=0x00;

    k++;

    if(k>5)
    {
        k=0;
        j++;           // Yazıyı kaydıracak değişken
        if(j==90)
            j=0;
    }
}

```

İşlemler takip edilirse aslında işlemlerin o kadar da zor olmadığı gözükmektedir. Dilerseniz daha fazla dot matris ve decoderler kullanarak yazı alanını büyütебilirsiniz.

BÖLÜM 16 – EK BİLGİLER

16.1) Pic'in Dahili Eepromuna Başlangıçta Bilgi Yazılması

Pic ilk başladığında dahili eepromuna bilgi yazmak için aşağıdaki kod programa eklenebilir.

```

#include <htc.h>

__CONFIG( PROTECT & DPROT & WDTDIS & XT & PWRTDIS & BORDIS & WP1 & LVPDIS
& DEBUGDIS );

__EEPROM_DATA('F' , 'I' , 'R' , 'A' , 'T' , 0x00,0x00,0x00);
__EEPROM_DATA('D' , 'E' , 'V' , 'E' , 'C' , 'i' , 0x00,0x00);

//Bu şekilde ilk kodlar yazılabilir.

void main(void)
{
...
}

```

BİTİRİRKEN...

Sizlere 16 bölüm boyunca Hi-Tech derleyicisini, C programlama ve en önemli Pic'in önemli özelliklerini anlatmaya, hiçbir şeyin ezbere dayalı olmaması gereğine, her şeyin bir mantık üzerine oturduğunu göstermeye çalıştım. Sadece derleyici anlatmakla kalmayıp, birçok ekstra donanımın çalışmasını da sizlere gösterdim...

Kitabı takip eden ve katkısı olan herkese teşekkürler...

Fırat Deveci
Ağustos 2009
fxdev@fxdev.org
www.firatdeveci.com